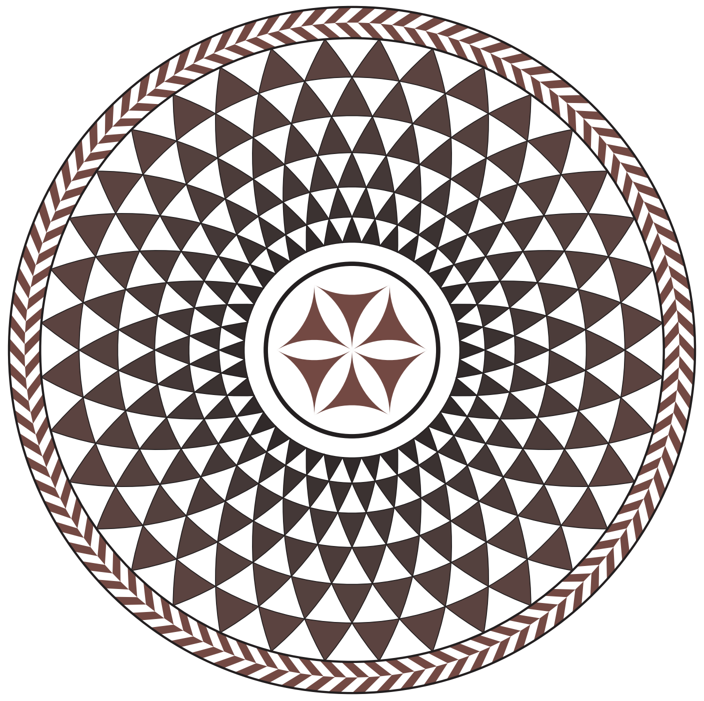
A TALE OF TWO SEQUENCES Interpretable and Linguistically-Informed Deep Learning for Natural Language Processing Jasmijn Bastings

A TALE OF TWO SEQUENCES

Interpretable and Linguistically-Informed Deep Learning
for Natural Language Processing



JASMIJN BASTINGS

A TALE OF TWO SEQUENCES

Interpretable and Linguistically-Informed Deep Learning
for Natural Language Processing

ILLC Dissertation Series DS-2020-09

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/

This work was supported by the Netherlands Organisation for Scientific Research (NWO), VICI 277-89-002.

Cover graphic by Daniel Steger 🅭🅯, modified for color.
Printed and bound by Printservice Ede.

ISBN: 978-90-830912-1-1

A TALE OF TWO SEQUENCES

Interpretable and Linguistically-Informed Deep Learning
for Natural Language Processing

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen
op donderdag 8 oktober 2020, te 16.00 uur

door

Jasmijn Bastings

geboren te Eindhoven

Promotiecommissie

Promotor:	Prof. Dr. K. Sima'an	Universiteit van Amsterdam
Co-promotor:	Dr. I.A. Titov	University of Edinburgh
Co-promotor:	Dr. W. Ferreira Aziz	Universiteit van Amsterdam
Overige leden:	Prof. Dr. L. Specia	Imperial College London
	Prof. Dr. A. Søgaard	University of Copenhagen
	Dr. R. Fernandez Rovira	Universiteit van Amsterdam
	Dr. W.H. Zuidema	Universiteit van Amsterdam
	Prof. Dr. L.W.M. Bod	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica *to my mother and sister*

for supporting me, for leading the way into science

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

Jasmijn Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an (2017). "Graph Convolutional Encoders for Syntax-aware Neural Machine Translation." In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1947–1957.

Jasmijn Bastings, Marco Baroni, Jason Weston, Kyunghyun Cho, and Douwe Kiela (2018). "Jump to better conclusions: SCAN both left and right." In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, pp. 47–55.

Jasmijn Bastings, Wilker Aziz, and Ivan Titov (July 2019). "Interpretable Neural Predictions with Differentiable Binary Variables." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 2963–2977.

Diego Marcheggiani, Jasmijn Bastings, and Ivan Titov (2018). "Exploiting Semantics in Neural Machine Translation with Graph Convolutional Networks." In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 486–492.

For all publications with JB as first author, all code was written by JB and all experiments conducted by JB, and JB was advised by the other authors, who also contributed to the writing of the original publications. For Marcheggiani et al. (2018), JB wrote the code and contributed to the experiments, while writing was done by DM and IT.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

BLEU	Bilingual Evaluation Understudy
BPE	Byte Pair Encoding
CNN	Convolutional Neural Network
DGM	Deep Generative Model
DL	Deep Learning
DLVM	Deep Latent-Variable Model
ELBO	evidence lower bound
HMM	Hidden Markov Model
GCN	Graph Convolutional Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MLA	Miniature Language Acquisition
MSE	Mean Squared Error
MT	Machine Translation
MTL	Multi-task Learning
NER	Named Entity Recognition
MLP	Multilayer Perceptron
NLP	Natural Language Processing
POS	Part of Speech
NMT	Neural Machine Translation
RAAM	Recursive Auto-Associative Memory
RHAM	Recursive Hetero-Associative Memory
RNN	Recurrent Neural Network

SGD	Stochastic Gradient Descent
SRL	Semantic Role Labeling
SRN	Simple Recurrent Network
SMT	Statistical Machine Translation
VAE	Variational Auto-Encoder
VI	Variational Inference
WMT	Workshop on Statistical Machine Translation

Be yourself; everyone else is already taken.
— Oscar Wilde

ACKNOWLEDGEMENTS

What a journey it has been! I remember very clearly the day I decided that I wanted to go back to academia. I was living in Sweden, and just a little while earlier Khalil had contacted me that he wanted to publish one of my Master's projects at LREC. This worked out, and so we found ourselves in Iceland not much later, a country that I instantly fell in love with. Looking back, I believe this was an important moment for me, because it proved to me that I could be successful in academia. So it wasn't so strange that, when I was ready for something new, I contacted Khalil asking about PhD opportunities. It turns out he had one, and I remember that I didn't have long to submit my application. A few months later I moved to Amsterdam and so the journey began. Khalil and Ivan, I am very thankful for the trust that you gave me at that point, after being away from academia for so long. And I truly hope that I managed to pay back that trust. I brushed up my machine learning skills, and soon we were on our way. Not much later Wilker joined the team. I think this was one of the best decisions that we made during the PhD, although I don't remember exactly how it happened. The beginning was not easy. There was a lot to learn and once things finally got moving it turned out that another research group had just published something extremely similar to what we were trying to do, perhaps even better. It took some time, but then we finally had a breakthrough with our syntax-aware neural machine translation. What a relief that was! (Maybe not just for me?) From then on things started to fall into place. Khalil, Ivan, Wilker, I couldn't have done it all without you and I am very thankful for your advice throughout the years.

I would also like to thank the members of my committee: Lucia, Anders, Raquel, Jelle, and Rens. Thank you for taking the time to read and discuss my work.

There are obviously a lot more people to thank. I would like to start with my office friends: first Raquel, Milos, Phong, and Sophie, and later Samira, Bas and Bryan. It was a pleasure interacting with you and having you around through all the ups and downs. Diego, Iacer, Joanna, Marialaura, and Miguel, thanks for all the good times, and for the many burgers and drinks. It was really nice having you around. Thanks Tom, for the good times and the workouts. Thanks Renske for always being there. And thanks Sander, Tessa, Truke, and Ingrid, it was nice to live a bit closer to you during the PhD years, and I hope our friendship continues for many years to come. Thanks Alvaro, for being my oldest friend, and always there to support me.

The list is still not complete. I'd also like to thank all the other amazing people at UvA and ILLC that I met throughout the years: Amir, Arnold, Bastiaan, Bushra, Carlos, Desmond, Dieuwke, Ehsan, Gideon, Giovanni, Hoang, Iris, Joachim, Joost, Laura, Lena, Maartje, Marco, Marion, Michael, Mostafa, Nadine, Nal, Rianne, Ronald, Sara, Sandro, Serhii, Stella, Thomas, Yfke. Thanks for all the good times we had!

And then there was the amazing ILLC staff. Thanks Jelle for the many discussions that we had. Thanks Ulle for helping me complete my first TA assignment. Thanks Benno, Leen, Katia, Raquel, Sonja and Yde. Jenny and Peter, thanks for helping me whenever I needed something, and for sponsoring the ILLC drinks. (Wilker I hope to ever match your cocktail making skills!). Thanks Tanja for being there from the very start of my PhD, and thanks ILLC office, Debbie, Karine, Patty for helping with whatever issue came up.

I'd also like to thank Douwe, Jason, Marco and Kyunghyun for having me as an intern in New York. I learned so much, and I'm still trying to pay it forward.

Finally, thank you Julia, for being by my side for many of the PhD years, and my family, for supporting me all this time.

While this journey is coming to an end, a new one has already begun.

1 INTRODUCTION

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."

— Lewis Carroll, Alice in Wonderland

Deep Learning (DL) – the learning of representations in particular using deep artificial neural networks – has swiftly taken over our field of Natural Language Processing (NLP). In a seminal paper, Collobert et al. (2011) were one of the first to show just how powerful neural networks can be. Instead of training separate models with manually-engineered task-specific features, they trained a *single* neural network to perform Part of Speech (POS) tagging, chunking, Semantic Role Labeling (SRL) and Named Entity Recognition (NER), thereby popularizing Multi-task Learning (MTL). Before this approach, systems would suffer from error propagation (with e.g. errors from a POS-tagger propagating into a parser), or, in an attempt to mitigate that, tasks would be correlated in the output space only. By using a single neural network to predict all levels of structures at once, much better performance could be achieved, with little feature engineering.

Collobert and Weston received the ICML Test of Time Award for an earlier version of this work (Collobert and Weston, 2008).

Years before, and arguably somewhat ahead of its time, Bengio et al. (2003) showed that an MLP could be effective in language modeling ("predicting the next word of a sequence").[1] By using continuous representations, the *curse of dimensionality* could be used to our advantage. Whereas words such as 'cat' and 'dog' would be distinct units in an n-gram language model, a neural network would figure out that they share many features predictive of the words to come.

Interestingly, none of the techniques in Collobert et al. (2011) were *novel*: the Multilayer Perceptron (MLP) and backpropagation (Rumelhart et al., 1985; Werbos, 1982) were already there for a while. With improvements in computer hardware, for the Graphics Processing Unit (GPU) in particular, it seems that the time was ripe for efficient training of neural networks for language processing. A lot more work was to follow.

1 Note that Schwenk and Gauvain (2002) proposed a similar model, while acknowledging the origin of their algorithm to Yoshua Bengio.

Initially, many papers focused on learning high quality distributed representations for words from unlabeled data. Mikolov et al. (2013) introduced `word2vec` with its Skip-gram and CBOW models, and Pennington et al. (2014) introduced `Glove`. Both methods would become popular for learning pre-trained word embeddings, which can be used as a starting point for a neural network model trained on a downstream task.

Another milestone was the popularization of Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). LSTMs are a special kind of Recurrent Neural Network (RNN) (Elman, 1990) that take care of the *vanishing gradient* problem; a problem that prevented Elman's simple RNNs from being trained to capture longer sequences. Since LSTMs model sequential data, they are a natural choice for encoding sentences – sequences of words – for NLP systems. They make for excellent baselines for most (if not all) NLP tasks, e.g., sentiment analysis, dialogue systems, and machine translation.

The formulation of LSTM in use today is described in e.g. Gers et al. (2000) and differs from the original formulation: a forget gate was added.

A few years later, Manning (2015) compared the rise of Deep Learning to a *tsunami*. By then it was clear that neural networks are here to stay, and that their performance is undeniable. However, Manning warned for a Kaggle effect, and that we should not focus on numbers, e.g. higher POS-tagging accuracy, but to use our new Deep Learning tools for science, such as analyzing the change of meaning of words over time.

Kaggle is an online platform for machine learning competitions, where the submission with the highest score wins the challenge.

1.1 LINGUISTICALLY-INFORMED DEEP LEARNING

In the sub-field of Machine Translation (MT), phrase-based methods dominated for more than a decade in the yearly WMT evaluation challenge (Bojar et al., 2014). This changed quickly after neural encoder-decoder models were applied to the problem by Sutskever et al. (2014) and Cho et al. (2014).[2] While they did not beat phrase-based (and syntax-based) MT systems from the very start, with the introduction of the attention mechanism by Bahdanau et al. (2015) to increase the resolution, and sub-word units by Sennrich et al. (2016b) to deal with large (open) vocabularies, now virtually all state-of-the-art systems rely on neural networks.

The rise of deep learning caused a shift from exploiting linguistic features and structures, such as POS-tags, dependency

2 The next chapter will mention many noteworthy prior works that proposed neural machine translation before these works. However, they did not reach the level of performance that these works reached.

and constituency trees, to relying solely on the input words, and treating a sentence as a mere *sequence* of words. Hence, Sutskever et al. (2014) is referred to as 'sequence-to-sequence learning'.

We have now set the stage for this thesis. With deep learning becoming more and more popular and breaking performance records across a variety of NLP tasks, we can naturally ask ourselves some questions about what that means for the methods that we used to rely on, and where to go from here.

Objective (Part I)

In the first part of this thesis we investigate the following research questions:

1. Does the performance of deep learning on NLP tasks mean that linguistic structures are now obsolete?
2. Is there a way to reap the performance benefits of deep neural networks, while still making use of linguistic structure such as dependency graphs?
3. Can we induce useful structure in an unsupervised way, while learning to perform an NLP task?

We will try to answer these questions for the task of machine translation. Chapter 3 deals with the first two questions, and looks at dependency trees and semantic role labeling structures. Since neural networks also excel at predicting those linguistic structures, in Chapter 4 we investigate if we can predict the (latent) structure of a sentence at the same time as learning to perform translation, without relying on the structure.

1.2 INTERPRETABILITY

There are two common criticisms of neural networks. The first is their lack of interpretability (see, e.g., Belinkov and Glass (2019)), which is why they are referred to as *black boxes*. It is simply hard to grasp what they have learned, and for example, it is not clear what it means that a particular learned weight is set to -0.1 after training. The second criticism is that neural networks have a hunger for labeled data. They need many data examples to learn a task and to generalize well. In Part II of this thesis, we look into these criticisms. While there are many ways to make a model more interpretable, in Chapter 5 we present one particular method for doing so with an application to text classification, by uncovering which part of the input text is sufficient for making the correct prediction. We will discuss how the proposed method fits into the landscape of interpretability methods.

Instead of designing a model to be more interpretable, we can also analyze a model as is, without modifying it. In Chapter 6, we look at how neural networks generalize, which is essential to learn a task. We, humans, can generalize from only a few examples; by using the principle of compositionality we can build complex meanings from simple parts. We look into a method to investigate the generalization capability of neural sequence-to-sequence models such as Bahdanau et al. (2015), and identify and remedy an important issue with the approach. By doing so we also find that neural networks still have difficulty on tasks that are easy for humans.

Objective (Part II)

In the second part of this thesis we will address the following research questions. On the topic of interpretability:

1. Can we make text classifiers more interpretable by having them provide an explanation, a rationale, by showing which part of the input document was used for classification? And how can we do that while still admitting gradient-based optimization?
2. Can we control the properties of the explanation?

And regarding generalization:

3. Do neural sequence-to-sequence models generalize systematically? How do we correctly test for this?

1.3 CONTRIBUTION

This thesis has two main contributions:

1. It proposes a method to add linguistic bias to Neural Machine Translation (NMT) models using the Syntactic Graph Convolutional Network (GCN), a special version of GCN designed for NLP tasks, and shows its effectiveness on

English-German and English-Czech machine translation task. While we used GCNs to condition on syntax (Bastings et al., 2017) and semantics (Marcheggiani et al., 2018), they can be used to incorporate any kind of graph structure into a neural network.

2. It proposes a method to construct interpretable text classifiers that can be trained using backpropagation. These text classifiers show which part of the input document is used for classification, and which part is not used. The selected part of the document makes for the rationale, the explanation. The proposed method makes it easier to incorporate a (hybrid) binary mask in any end-to-end neural model, and thus has potential uses outside of the applications explored in this thesis.

Apart from these main contributions, the thesis contributes a model that induces a latent graph and conditions on it in the context of NMT, while keeping the graph-inducing component completely separate from the NMT model. One important side contribution is therefore to show how to design and fairly evaluate a model conditioning on a latent graph, by not leaking information between the components and while keeping the number of parameters the same across comparisons.

Lastly, the thesis advances work on the analysis of neural networks in NLP. In particular, it fixes an important issue with the popular SCAN benchmark for testing strong generalization, by identifying and remedying an issue with (the lack of) target-side dependencies of the SCAN data set.

1.4 OVERVIEW

We will now look at a brief overview of the contents of this thesis. Chapter 2 covers background topics useful for reading the rest of this thesis. It includes topics such as word alignment, statistical machine translation, neural machine translation, and machine translation evaluation. The thesis is then divided into two parts, which we cover separately.

1.4.1 *Part I*

Part I deals with linguistically-informed deep learning:

Chapter 3 investigates if neural machine translation models, making use of deep learning, can still benefit from linguistic structure. To do so it presents a method to incorporate such structure into deep neural networks using graph convolutional networks.

Chapter 4 then explores if we can induce useful structure with translation as a downstream task, instead of feeding such structure in the form of linguistic knowledge. It presents a model with two separate components, so as to make sure possible improvements arise from the induced structure, and that induces structure alone.

1.4.2 *Part II*

Part II deals with the 'black box' aspect of neural networks. While neural networks perform well on NLP tasks, how do we know if we can trust them? And how do they work?

Chapter 5 focuses on text classification tasks, and presents a differentiable approach for a model to learn to provide a rationale, an explanation, together with its predictions.

Chapter 6 looks at the systematic generalization behavior of neural sequence-to-sequence models, and how to benchmark it. It finds an issue with a popular benchmark, SCAN, and proposes to remedy it with its inverse problem, NACS.

Finally, Chapter 7 makes concluding remarks for this thesis and looks into possible future work.

2 BACKGROUND

"It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of light, it was the season of darkness, it was the spring of hope, it was the winter of despair."

—— Charles Dickens, A Tale of Two Cities

This thesis relies heavily on sequence-to-sequence models. One of the main applications for such models is Machine Translation (MT), which is concerned with the automatic translation of a sentence from one language into another. We therefore start with a thorough treatment of the Neural Machine Translation (NMT) literature. But before we do so, we should acknowledge that the ideas for using statistics to map sequences from one language to another can be traced back all the way to (Weaver, 1949, 1955):

> When I look at an article in Russian, I say : "This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode."

It took many decades for computers to become powerful enough for it to be demonstrated in practice. Brown et al. (1990) and Brown et al. (1993) proposed using the EM-algorithm to induce word alignments between sentences in a parallel corpus, and when you have alignments you can also compute the probability of a word being the translation of another word. Figure 2.1 shows an example sentence pair and alignment. Word-based models like this became a stepping stone for the later phrase-based Statistical Machine Translation (SMT) models (Koehn et al., 2003; Och and Ney, 2004). By heuristically extracting phrases from the word alignments, and learning to score and combine them, translation systems improved dramatically as they could now take local context into account. Phrase-based systems were topping the Workshop on Statistical Machine Translation (WMT) yearly translation evaluation campaign until neural approaches made their entry (Bojar et al., 2014, 2015).

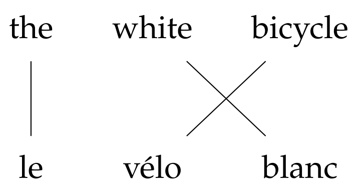


Figure 2.1: An example alignment.

2.1 NEURAL MACHINE TRANSLATION

In NMT (Cho et al., 2014; Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014) we model the probability

$$P_\theta(y_1, y_2, \ldots, y_N \mid x_1, x_2, \ldots x_M) \tag{2.1}$$

of an (e.g., English) target sentence $y_1, y_2, \ldots, y_N$ given a (e.g., French) source sentence $x_1, x_2, \ldots x_M$ *directly* using a neural network.[1] In short, we will write:

$$P_\theta(y_1^N \mid x_1^M) \tag{2.2}$$

and using the chain rule, we can express it in this way:

$$P_\theta(y_1^N \mid x_1^M) = \prod_{i=1...N} P_\theta(y_i \mid x_1^M, y_1^{i-1}) \tag{2.3}$$

This means that there are no Markov assumptions; the probability of output word y_i is conditioned on all previously generated words y_1^{i-1}. Just like before we have a parallel corpus with sentence pairs $\langle x_1^M, y_1^M \rangle$, and our goal is to fit the parameters θ of our neural network to maximize the log likelihood of the data. All modern neural models in this section share this objective, but they differ in their neural architecture.

In the following, we will cover the developments in modern neural machine translation chronologically, to then discuss some early work on the topic that was later rediscovered. We will start with recurrent encoder-decoder models (§2.1.1), then turn to recurrent models with attention (§2.1.2), alternative architectures (§2.1.3), and finally early NMT models (§2.1.4).

1 We will use the convention from machine learning to use x for input and y for output.

2.1.1 *Recurrent Encoder-Decoder Models*

The breakthrough of neural machine translation came with the encoder-decoder models of Cho et al. (2014) and Sutskever et al. (2014).[2] An encoder module encodes the source sentence into a vector, while a decoder generates the target sequence one word at a time from that vector. We will now discuss the encoder and decoder separately.

2.1.1.1 *Encoder*

We associate each word type in the source data (and analogously the target data) with an ID, creating the vocabularies $V^{(\mathrm{src})}$ and $V^{(\mathrm{trg})}$. Each word that we input to our network is then represented by a one-hot vector: a vector the size of the vocabulary that consists of all zeros except for one index (at the position of the word ID) where it contains a one. If we then multiply the one-hot vector with a word embedding matrix $E^{(\mathrm{src})} \in \mathbb{R}^{d_{\mathrm{emb}} \times |V^{(\mathrm{src})}|}$, we select the exactly one column from that matrix: the word embedding for that word.

The input to the neural network is a sequence of one-hot encoded vectors $\mathbf{x}_1, \ldots, \mathbf{x}^M$. This sequence is then encoded into a vector by a Recurrent Neural Network (RNN) using the following recursive definition:

$$\mathbf{h}_j = \mathrm{RNN}(E^{(\mathrm{src})}\mathbf{x}_j, \mathbf{h}_{j-1}) \tag{2.4}$$

where we can use an zero vector for $\mathbf{h}_0$, and then feed one word embedding $E^{(\mathrm{src})}\mathbf{x}_j$ at each time step. After feeding the complete input sequence we end up with a final vector $\mathbf{c} = \mathbf{h}_M$ which, as a result of training, captures the meaning of the source sentence. In other words, we learn a representation $\mathbf{c}$ of the source sentence x_1^M. This vector is called the *context vector*.

We use RNN as a general recurrent function here, that takes an input at each time step and updates its hidden state. A crucial insight here was that the simple RNN of Elman (1990) has difficulty learning to encode long sequences, because of the vanishing gradient problem.[3] Therefore, Cho et al. (2014) propose a Gated Recurrent Unit (GRU) as the recurrent function, and

2 For conciseness we will not discuss the earlier approach of Kalchbrenner and Blunsom (2013), that can also be seen as an encoder-decoder, but that uses (de)convolutional networks and a simple RNN.

3 See Bengio et al. (1994) and Pascanu et al. (2013) for further background on the difficulty of training RNNs.

Sutskever et al. (2014) use the Long Short-Term Memory (LSTM) of Hochreiter and Schmidhuber (1997).[4]

2.1.1.2 *Decoder*

After encoding the source sentence, the decoder predicts the target sequence one word at a time. The decoder is also a GRU or LSTM, just like the encoder, but with separate parameters. Its recursive definition is:[5]

$$\mathbf{h}_i^{(\text{dec})} = \text{RNN}([E^{(\text{dec})}\mathbf{y}_{i-1}; \mathbf{c}], \mathbf{h}_{i-1}^{(\text{dec})}) \tag{2.5}$$

For each time step i, an output vector $\mathbf{o}_i \in \mathbb{R}^{|V^{(\text{trg})}|}$ is computed in the following way:[6]

$$\mathbf{t}_i = W^t[E^{(\text{dec})}\mathbf{y}_{i-1}; \mathbf{c}; \mathbf{h}_i^{(\text{dec})}] \tag{2.6}$$

$$\mathbf{o}_i = \text{softmax}(W^o\mathbf{t}_i) \tag{2.7}$$

where $E^{(\text{dec})}\mathbf{y}_{i-1}$ is the word embedding of the previous target word, $\mathbf{c}$ the context vector that was computed by the encoder, and $W^o, W^{o'}$ learned parameters. Since the softmax function makes the output vector all positive and sum to one, we have that

$$P(y_1^N \mid x_1^M) = \prod_{i=1}^{N} \mathbf{o}_i^\top \mathbf{y}_i \tag{2.8}$$

SPECIAL TOKENS. In the first decoder RNN step we need to provide $\mathbf{y}_0$, for which we use the special beginning-of-sequence (BOS) token `<s>`. At the end of each sentence we append an end-of-sequence (EOS) token `</s>`, so that the model can learn when it is done generating the output. This also allows us to stop generating when we see the EOS token at test time, when we do not have the reference sentence and therefore do not know the length of the translation. A padding token `<pad>` (discussed later) is the third special token.

4 The used LSTM implementation follows the definition of Graves (2013), since the formulation of Hochreiter and Schmidhuber (1997) did not yet include a forget gate.

5 Sutskever et al. (2014) do not condition on $\mathbf{c}$ in this way, instead they condition on it by initializing the first decoder hidden state $\mathbf{h}_0^{(\text{dec})}$ with a projection from $\mathbf{c}$.

6 For clarity we leave out the max-out layer that was used in addition to this in Cho et al. (2014), since it was not adopted by the community.

2.1.1.3 *Learning*

We maximize the conditional log-likelihood of the data

$$\max_{\theta} \frac{1}{|\mathcal{D}|} \sum_{\langle x_1^M, y_1^N \rangle \in \mathcal{D}} \log P(y_1^N \mid x_1^M, \theta) \tag{2.9}$$

where $\mathcal{D}$ is our data set of sentence pairs that we assume to be independent and identically distributed (i.i.d.) and θ are the parameters of the model.

We optimize our parameters towards this goal by minimizing the cross-entropy loss:

$$L(x_1^M, y_1^M, \theta) = \sum_{i=1}^{N} \mathrm{CE}(\mathbf{y}_i, \mathbf{o}_i) = -\sum_{i=1}^{N} \log \mathbf{o}_i \mathbf{y}_i^{\top} \tag{2.10}$$

where $\mathbf{y}_i \in \{0, 1\}^{|V_y|}$ is the one-hot encoded target word for time step i and $\mathbf{o_i} = f(\mathbf{y}_{i-1}, \mathbf{h}^{(dec)}, \mathbf{c})$ is the output of our decoder for that time step. The cross-entropy loss expresses that we would like to give the correct word a probability of 1.0 at the cost of all other possible words. Since target $\mathbf{y}_i$ is non-zero only for one index and zero elsewhere, we obtain the cross-entropy loss by simply taking the negative log of the output unit at the index of the target word, i.e., the log-likelihood of the observation.

After computing the cross-entropy loss for each target word, and summing each of those individual losses, we can use back-propagation to obtain gradients of the loss with respect to the parameters θ. To obtain less noisy gradient estimates, and to speed up training, it is common to compute the loss for multiple sentences in a minibatch at the same time, and then to divide the loss by the number of sentences in the minibatch. If the sentences are not the same length, we simply fill the empty slots using a special `<pad>` token which is ignored in the loss.

We then take a small step in the direction of the gradient to update our parameters. We can do so with SGD (see e.g. Bottou (2012)) or by using a more advanced optimizer with adaptable learning rate such as Adam (Kingma and Ba, 2015), which is the default optimizer for many NMT implementations.

2.1.2 *Encoder-Decoder Models with Attention*

Observing that it is difficult to capture the meaning of the source sentence in a single vector $\mathbf{c}$, Bahdanau et al. (2015) propose an *attention mechanism* that dynamically constructs a context vector $\mathbf{c}_i$ for each decoder time step i. We will again discuss the encoder and decoder separately.

2.1.2.1 *Encoder*

The encoder remains largely the same, except that we now encode the source sentence using a bidirectional RNN (Graves and Schmidhuber, 2005; Schuster and Paliwal, 1997), which consists of a forward RNN (f) that processes the sentence from left to right (as before), and a backward RNN (b) with separate parameters that goes from right to left.

$$\mathbf{h}_j^f = \mathrm{RNN}(E^{(src)}\mathbf{x}_j, \mathbf{h}_{j-1}^f) \quad \mathbf{h}_j^f \in \mathbb{R}^d \tag{2.11}$$
$$\mathbf{h}_j^b = \mathrm{RNN}(E^{(src)}\mathbf{x}_j, \mathbf{h}_{j+1}^b) \quad \mathbf{h}_j^b \in \mathbb{R}^d \tag{2.12}$$

After obtaining the hidden states we concatenate them for each position to again have a sequence of hidden states $\mathbf{h}_1, \ldots, \mathbf{h}_M$:

$$\mathbf{h}_j = [\mathbf{h}_j^f; \mathbf{h}_j^b] \qquad \forall_{j \in 1,..,M} \qquad \mathbf{h}_j \in \mathbb{R}^{2d} \tag{2.13}$$

The concatenation doubles the size of the source word representations, unless we use $\lfloor d/2 \rfloor$ as the output dimension for the forward and backward RNN.

2.1.2.2 *Decoder*

The biggest change that Bahdanau et al. (2015) make is in the decoder. Different from before, we will use *all* source representations $\mathbf{h}_j$, not just the last one. We still generate one output token at a time using an RNN (a GRU or LSTM), but now the context vector changes for each time step:

$$\mathbf{h}_i^{(dec)} = \mathrm{RNN}([E^{(dec)}\mathbf{y}_{i-1}; \mathbf{c}_i], \mathbf{h}_{i-1}^{(dec)}) \tag{2.14}$$

Note how the context vector $\mathbf{c}_i$ is now indexed with index i. It is computed using an attention mechanism, that computes a score between the previous decoder state $\mathbf{h}_{i-1}^{(dec)}$ and *each* source word representation $\mathbf{h}_j$:

$$e_{ij} = \mathrm{score}\left(\mathbf{h}_{i-1}^{(dec)}, \mathbf{h}_j\right) \tag{2.15}$$

The scores are then turned into a probability distribution over source positions using a softmax function:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_k \exp e_{ik}} \qquad \sum_k \alpha_{ik} = 1 \tag{2.16}$$

Now that we have a weight for each source position, the context vector $\mathbf{c}_i$ is defined as a weighted sum:

$$\mathbf{c}_i = \sum_{j=1}^{M} \alpha_{ij} \mathbf{h}_j \tag{2.17}$$

There are different ways the score function can be implemented. The only requirement is that the function takes two vectors (previous decoder state, encoder state) as input, and returns a scalar energy term. Bahdanau et al. (2015) use an MLP to do so, also referred to as 'additive attention':

$$\text{score}\left(\mathbf{h}_{i-1}^{(\text{dec})}, \mathbf{h}_j\right) = \mathbf{v}^\top \tanh\left(W_a \mathbf{h}_{i-1}^{(\text{dec})} + U_a \mathbf{h}_j\right) \tag{2.18}$$

where $\mathbf{v}_a$, W_a, and U_a are learned parameters. In a later work, Luong et al. (2015c) use bilinear attention instead, also referred to as 'multiplicative attention':

$$\text{score}\left(\mathbf{h}_{i-1}^{(\text{dec})}, \mathbf{h}_j\right) = \mathbf{h}_{i-1}^{(\text{dec})\top} W_a \mathbf{h}_j \tag{2.19}$$

INPUT FEEDING. Luong et al. (2015c) use a slightly different sequence of steps for the decoder. Their decoder update is given by $\mathbf{h}_i^{(\text{dec})} = \text{RNN}\left([E^{(\text{dec})}\mathbf{y}_{i-1}; \mathbf{t}_{i-1}], \mathbf{h}_{i-1}^{(\text{dec})}\right)$, after which an 'attention vector' is computed: $\mathbf{t}_i = \tanh\left(W_{att}[\mathbf{h}_i^{(\text{dec})}; \mathbf{c}_i]\right)$. The output vector is then obtained by $\mathbf{o}_i = W_o \mathbf{t}_i$. Since we feed the previous attention vector $\mathbf{t}_{i-1}$ to the decoder this is called input feeding. This sequence also gives a more direct path from input to prediction compared to Bahdanau et al. (2015).

DEEPER. It is possible to stack multiple RNN layers on top of each other to make the model deeper. The first layer takes word embeddings as input, where each subsequent layer simply takes the output of the layer below as input. Sometimes a residual layer is used (He et al., 2016), where the input to a layer is added to its output. For more details on deeper models and best practices, see e.g. Wu et al. (2016a).

2.1.3 *Alternative Models*

While recurrent models have dominated the field ever since their inception, recently two other non-recurrent architectures were found to work well on the problem of translation. Those are Convolutional Neural Network (CNN)-based and the Transformer. We will discuss both of these briefly.

2.1.3.1 *Convolutional Models*

Gehring et al. (2017b) propose an alternative architecture based on a Convolutional Neural Network (CNN). We will describe it briefly, as a simple version of it will be used in the next chapter.

ENCODER. We encode each sentence using 1-dimensional convolutions. We first choose a window size k for our convolutions. Say we choose $k = 5$. We then add $\lfloor k/2 \rfloor$ padding tokens to the left and the right of each sentence. To each word embedding we add a learned *position embedding*, so that the model has access to the absolute position of each word in the sentence. The position embedding is a learned vector, just like a word embedding, and position embeddings are learned for each position up to a certain length (e.g. 50). Then we define a weight matrix $W \in \mathbb{R}^{kd \times d}$ for our convolution, where d is the size of our word embeddings and model. For each word position j, we compute a new representation with:

$$\mathbf{h}_j = W[E\mathbf{x}_{j-\lfloor k/2 \rfloor}; \ldots ; E\mathbf{x}_j; \ldots ; E\mathbf{x}_{j+\lfloor k/2 \rfloor}] \tag{2.20}$$

that is, we concatenate the word representation (word embedding + position encoding) for word j with its neighboring words to make a vector of size kd, and then project it to get a new representation for that word. We can do this multiple times, where with each layer we increase the receptive field of each word representation.

DECODER. The target sentence is encoded in a similar way, with the addition of a mask so that only previous words are considered in the window, and not future words. After encoding the source and the target sequence, attention scores are computed between each encoder and decoder word representation using a dot product:

$$e_{ij} = \mathbf{h}_j^{\text{enc}} \cdot \mathbf{h}_i^{\text{dec}} \tag{2.21}$$

The attention scores are then used to compute a context vector which is in turn used to predict the next target word.

2.1.3.2 *Transformer*

Vaswani et al. (2017) propose an architecture, the Transformer, that only consists of attention mechanisms, and nothing else. We will again describe the encoder and decoder separately.

ENCODER. After looking up the word embedding for an input word, a position encoding is added, and the resulting sequence of word embeddings is stacked to form matrix $X \in \mathbb{R}^{M \times d}$, where M is the sentence length and d the dimensionality of the embeddings. In contrast to the Convolutional model of Gehring et al. (2017b) that learns position embeddings for each absolute word position, here a position encoding is a fixed (not learned) vector. For a word at position i a position encoding is defined as, for even dimensions j: $\sin\left(i/10000^{2j/d}\right)$ and uneven dimensions j: $\cos\left(j/10000^{2j/d}\right)$. The position encodings have the same dimensionality as the word embeddings which allows for summing them. Note that the position encoding for position $i + k$ can be expressed as a linear function of the position encoding for i, so position encodings allow the model to learn functions sensitive to relative positions.

We define the following learnable parameters:[7]

$$A \in \mathbb{R}^{d \times d_a} \quad B \in \mathbb{R}^{d \times d_a} \quad C \in \mathbb{R}^{d \times d_o} \tag{2.22}$$

where d_a is the dimensionality of the attention (inner product) space and d_o the output dimensionality. Transforming the input matrix with these matrices into new word representations H

$$A \in \mathbb{R}^{d \times d_a} \quad B \in \mathbb{R}^{d \times d_a} \quad C \in \mathbb{R}^{d \times d_o} H = \underbrace{\mathrm{softmax}\left(XA\,B^\top X^\top\right)}_{\text{self-attention}} XC \tag{2.23}$$

which have been updated by attending to all other source words. The Transformer uses multi-headed attention, in which this transformation is computed k times, one time for each head with different parameters A, B, C.

After computing all k Hs in parallel, we concatenate them and apply layer normalization[8] (Ba et al., 2016) and a final feed-forward layer[9]:

$$H = [H^{(1)}; \ldots; H^{(k)}] \tag{2.24}$$

$$H' = \text{layer-norm}(X + H) \tag{2.25}$$

$$H^{(\text{enc})} = \text{feed-forward}(H') \tag{2.26}$$

We set $d_o = d/k$, so that $H \in \mathbb{R}^{M \times d}$. Multiple of these layers can be stacked by setting $X = H^{(\text{enc})}$ and repeating the computation.

7 This explanation of the Transformer was adapted from Michael Collins https://youtu.be/jfwqRMdTmLo

8 Layer norm on a vector $\mathbf{x}$ is defined as $\frac{x-\mu}{\sigma} \odot \mathbf{g} + \mathbf{b}$, with $\mu = \mathbb{E}[\mathbf{x}]$ and $\sigma = \sqrt{\mathbb{E}[(\mathbf{x} - \mu)^2]}$, using learned parameters $\mathbf{g}$ and $\mathbf{b}$.

9 Defined as $W' \max(0, W\mathbf{x} + \mathbf{b}) + \mathbf{b}'$

DECODER. The decoder is similar to the encoder, but takes the stacked *target* embeddings $Y \in \mathbb{R}^{N \times d}$ as input:

$$H = \underbrace{\mathrm{softmax}\big(YA\,B^\top Y^\top\big)}_{\text{masked self-attention}} YC \tag{2.27}$$

For each target position attention to future input words is inhibited by setting those attention scores to $-\mathrm{inf}$ before the softmax. After obtaining $H' = \text{layer-norm}(Y + H)$, and before the feed-forward layer, we compute multi-headed attention again, but now between intermediate decoder representations H' and final encoder representations $H^{(\mathrm{enc})}$:

$$Z = \underbrace{\mathrm{softmax}\big(H'A\,B^\top H^{(\mathrm{enc})\top}\big)}_{\text{src-trg attention}} H^{(\mathrm{enc})}C \tag{2.28}$$

$$H^{(\mathrm{dec})} = \text{feed-forward}(\text{layer-norm}(H' + Z)) \tag{2.29}$$

Target words are predicted with a projection $H^{(\mathrm{dec})}W_{\mathrm{out}}$.

2.1.4 *Early neural machine translation models*

Even though Cho et al. (2014) and Sutskever et al. (2014) are widely credited with inventing modern neural machine translation, there is earlier work where neural networks are applied to the task. These early approaches typically use simpler architectures and much smaller data sets, and do not always give a probability $p(y \mid x)$ for their translations.

2.1.4.1 *Feed-forward*

Allen (1987) uses a feed-forward neural network to translate English sentences into Spanish. Figure 2.2 shows this architecture. It is formalized as follows:

Because Allen (1987) uses a feed-forward neural network, only a fixed number of input and output words are supported.

$$\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2, \dots, \mathbf{x}_M] \tag{2.30}$$

$$\mathbf{h} = \sigma(W\mathbf{x} + \mathbf{b}) \tag{2.31}$$

$$\mathbf{y} = \sigma(W^{(y)}\mathbf{h} + \mathbf{b}^{(y)}) \tag{2.32}$$

where $\mathbf{x}$ is the concatenation of all input vectors, $\mathbf{y}$ the concatenated outputs, σ the sigmoid activation function, and $W \in \mathbb{R}^{D_h \times D_i}$, $\mathbf{b} \in \mathbb{R}^{D_h}$, $W^{(y)} \in \mathbb{R}^{D_y \times D_h}$, $\mathbf{b}^{(y)} \in \mathbb{R}^{D_y}$ learned parameters. The English vocabulary only consists of 31 words (encoded using 5 bits), whereas the Spanish one has 40 words (encoded using 6 bits). In Spanish adjectives and determiners are marked

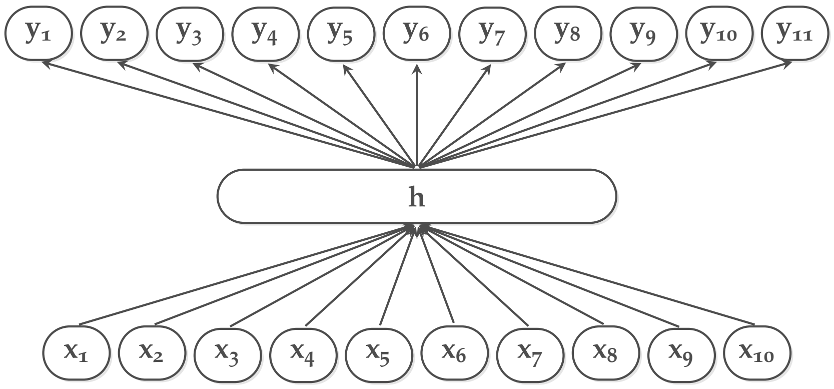


Figure 2.2: The feed-forward network of Allen (1987) supports 10 input words (each 5 units) and 11 output words (each 6 units), with 1 to 3 hidden layers of 150 units.

for gender, hence the larger vocabulary. 3310 simple sentences were generated, with a maximum length of 10 for English and 11 for Spanish, out of which 33 were used as a test set. Given the feed-forward architecture with a fixed input and output size, only sentences of up to 10 source (11 target) words are supported. This means that the input layer contains 50 units, and the output layer 66 units. For shorter sentences unused word positions are padded.

Allen experiments with three feed-forward architectures to map the input layer to the output layer: direct mapping (50-66), one hidden layer of 150 units (50-150-66), and three hidden layers of 150 units (50-150-150-150-66).

The best and largest model reached a training error of 0.027 which meant an average of 1.3 words incorrect on the test set.

It should be noted that Allen (1987) randomly pairs words with their binary encoding. This means the distance between two related words is arbitrary. However, Allen does note that a hierarchical or similarity based scheme could have been used, e.g., based on parts of speech, gender, and number features.

2.1.4.2 *RAAM*

Chrisman (1991) uses a dual-ported version of the Recursive Auto-Associative Memory (RAAM) architecture of Pollack (1990) to perform translation. RAAM can encode any kind of tree structure as long as it has a bounded branching factor, and a list or sequence is a special case of such a structure. In contrast to the feed-forward network of Allen (1987), RAAM can encode (and

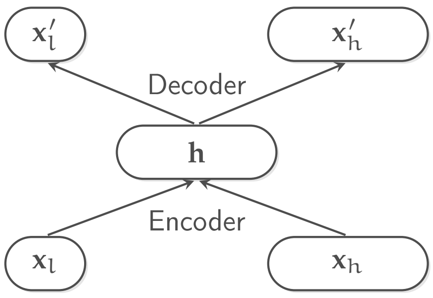


Figure 2.3: RAAM has input layers $\mathbf{x}_l$ and $\mathbf{x}_h$, a hidden layer $\mathbf{h}$, and output layers $\mathbf{x}'_l$ and $\mathbf{x}'_h$. A sequence can be encoded by feeding the first item as $\mathbf{x}_l$ with an empty $\mathbf{x}_h$. Subsequent items can be encoded by copying $\mathbf{h}$ to input $\mathbf{x}_h$ and feeding the item as $\mathbf{x}_l$. Decoding works by copying a representation to the hidden layer, after which the *last* item appears on $\mathbf{x}'_l$ and the representation of the remainder of the sequence on $\mathbf{x}'_h$.

decode) arbitrarily long sentences. This advantage is however limited in practice by the number of units of its hidden state and the ability of gradients to propagate. Figure 2.3 shows the RAAM architecture and explains how it recursively encodes (or decodes) a sequence.

This basic RAAM can be used to learn representations of sentences in one language, but it cannot perform translation. To enable this, Chrisman adds two extra input layers and two extra output layers to the model, connecting them to the same hidden layer $\mathbf{h}$. The resulting architecture is shown in Figure 2.4 and is called the dual-ported RAAM. The idea is now that $\mathbf{h}$ is trained to represent sentences in both the source language as well as sentences in the target language. This is done by minimizing multiple loss functions L simultaneously:

1. $L\big(\text{dec1}(\text{enc1}(\mathbf{x})), \mathbf{x}\big)$ (for each subsequence)
2. $L\big(\text{dec2}(\text{enc2}(\mathbf{y})), \mathbf{y}\big)$ (for each subsequence)
3. $L\big(\text{dec2}(\text{enc1}(\mathbf{x})), \mathbf{y}\big)$ (for final sentence representation)
4. $L\big(\text{dec1}(\text{enc2}(\mathbf{y})), \mathbf{x}\big)$ (for final sentence representation)

Here we use $\mathbf{x}$ for input (and target) $[\mathbf{x_l}; \mathbf{x_h}]$, i.e., the concatenation of the two input layers for the source language. We use $\mathbf{y}$ in a similar way but for the target language. L is the loss function, defined as:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}||\hat{\mathbf{y}} - \mathbf{y}||_2^2 \tag{2.33}$$

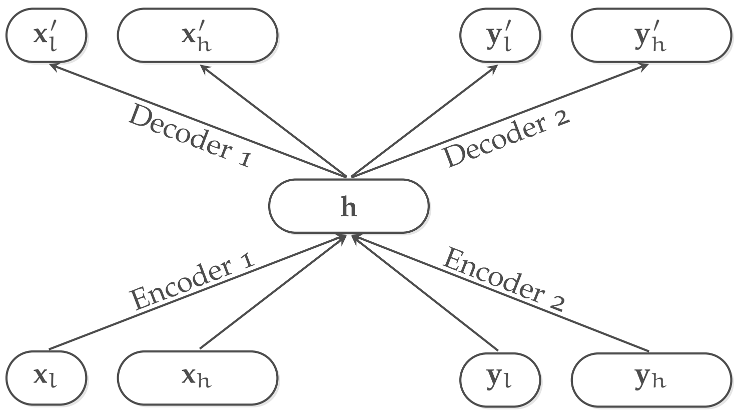


Figure 2.4: The Dual-Ported RAAM of Chrisman (1991). The architecture can be seen as consisting of two auto-encoders that share their hidden layers, forcing the representations of a sentence in two different languages to be close to each other.

expressing that we want to minimize the Euclidean distance between the output and the target. We use the shorthands 'enc' and 'dec' for Encoder and Decoder, respectively.

A corpus of 216 English-Spanish sentence pairs was created from a vocabulary of 36 English and 36 Spanish words. In contrast to Allen (1987), words are encoded with a localist encoding scheme based on linguistic features, such as word category (e.g., verb, noun, pronoun, determiner, adjective, etc.) and plurality (e.g., first person singular), in a fashion similar to Pollack (1990).

The hidden layer was set to 40 units, English words had 21 units, and Spanish words 19. After providing the correct output length[10], the model was able to translate 89% of unseen sentences correctly (exact match).

Chrisman (1991) concludes with some suggestions not explored in the paper, but that will turn out to resurface in modern NMT systems. These include many-to-1 translation (multiple source languages), auxiliary losses (e.g. sentence classification), and what it referred to as a 'coupled hybrid architecture' where the two auto-encoders to have their own hidden layers, but they are coupled using a learned transformation. This last suggestion is quite reminiscent of the later encoder-decoder models of Cho et al. (2014) and Sutskever et al. (2014).

10 Chrisman (1991) did not yet make use of an end-of-sequence symbol which caused some output sentences to be too short.

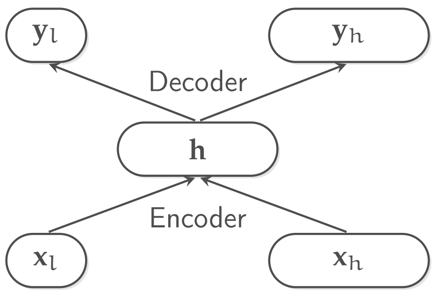


Figure 2.5: RHAM first encodes the source sequence into hidden state **h** by repeatedly feeding the next item and copying back the hidden state to the input, and then directly decodes **h** into the target sequence, without the need for auto-encoding the source and target sequences.

2.1.4.3 *RHAM*

Forcada and Ñeco (1997) observe that the RAAM-based approach of Chrisman (1991) requires one-to-one translations: it is not possible for two sentences in one language to have the exact same translation. To remedy this they propose Recursive Hetero-Associative Memory (RHAM). Figure 2.5 shows RHAM. While the architecture is similar to RAAM, it is used in a different way: there is no auto-encoding of the source and the target sequences, but instead the hidden state is directly decoded into the target sequence. If the source and target are the same, then RHAM is identical to RAAM. Forcada and Ñeco also use a end-of-sequence representation so that it can be detected when to stop the decoding process.

The training loss consists of the distance of each output with the encoding of the corresponding target word, as well as a loss for learning the end-of-sequence representation. Optimization was done using Alopex (Unnikrishnan and Venugopal, 1994), which is based on local correlations between weight changes and the global error measure, and not using backpropagation.[11]

In contrast to previous work experiments are done on artificial data sets generated by three automata over the simple alphabet $\{0, 1\}$ for both source and target. Because empty inputs/outputs are possible, the lengths of the input and output sequences are not always the same. Important in this paper is that the data sets contain many-to-one translations, with multiple source sequences mapping to the same target sequence.

11 A discussion of Alopex is however outside the scope of this thesis. It would have been possible to use backpropagation to train this model.

The test set consisted of the longest (9 and 10 items) sequences which were not used for training. For the most challenging automaton the generalization performance was 89% using 8 hidden units. Using fewer or more hidden decreased the performance.

2.1.4.4 *Second-order RNN*

Neco and Forcada (1997) also experiment on artificial data generated by automata, however they use a modified second-order[12] discrete-time Recurrent Neural Network (RNN) (Giles et al., 1990) instead of RAAM or RHAM. The modifications that Neco and Forcada make to the RNN allow it to produce output already before the complete source sequence has been read. Two extra output bits are used as control signals: one for advancing the input (`AdvIn`), and one for the output length (`OutLen`). When $\text{AdvIn} < 0.5$, an empty token is presented to the neural network in the next time step, otherwise the next input token is presented. `OutLen` predicts the output length in $[0, 1]$ of the current output token. The training loss is based on the squared difference of the predicted output with the correct output sequence and their length difference. The model is trained on four data sets generated by automata containing sequences of up to 11 items. Generalization is then tested on all sequences up to 15 items. For each of the automata generalization performance was 93% or higher.

2.1.4.5 *SRN*

Castano and Casacuberta (1997) use the Simple Recurrent Network (SRN) of Elman (1990) for translation. Figure 2.6 shows their model. An SRN has a simple recursive formulation to update the hidden state. Given the previous hidden state $\mathbf{h}_{i-1}$ and input vector for the current time step $\mathbf{x}_i$, the new hidden state is obtained by:

$$\mathbf{h}_i = \sigma(W^{(h)}\mathbf{h}_{i-1} + W^{(x)}\mathbf{x}_i + \mathbf{b}) \tag{2.34}$$

12 In this context 'second-order' refers to how the receptive field of a neuron is defined. A first-order receptive field for a neuron k would be $v_k = \sum_j w_{a,kj} h_j + \sum_i w_{b,ki} x_i$, whereas a second-order field would be $v_k = \sum_i \sum_j w_{kij} h_j x_i$, i.e., using a single weight connecting it to two inputs. See Giles et al. (1990) and Haykin (1994).

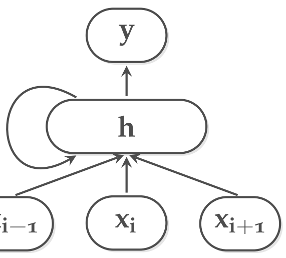


Figure 2.6: The SRN-based model of Castano and Casacuberta (1997).

un cuadrado mediano y claro y un círculo claro tocan a un círculo y un cuadrado mediano y oscuro
a medium light square and a light circle touch a circle and a medium dark square

Figure 2.7: An example MLA-MT sentence pair.

where σ is the sigmoid activation function. At each time step an output is produced using:

$$\mathbf{y}_i = \phi(W^{(y)}\mathbf{h}_i + \mathbf{b}^{(y)}) \tag{2.35}$$

where ϕ is an activation function for the final output, in this case also a sigmoid.

For better performance a *window* of words around the current input word was provided as input at every time step. Input and output words were represented by one-hot vectors.

Experiments were done on the Miniature Language Acquisition (MLA) data set of Feldman et al. (1990) which was turned into an MT data set by Castellanos et al. (1994). It consists of sentences describing simple visual scenes, together with their translations. Figure 2.7 shows an example sentence pair. Additional experiments were done on an extended version of the data set, where objects could be removed and added, which caused the vocabularies to grow to 30 words each. 500 and 1500 sentence pairs were used for training on MLA-MT, and 500 and 3000 sentence pairs were used to train on the extended version. For evaluation, for each of the two tasks, three independently generated test sets with 2000 sentence pairs were used. The best model, using 60 hidden units and a window of 3+3 input words, reached almost 100% test accuracy (exact match).

2.2 DEALING WITH INFREQUENT WORDS

Naively constructing vocabularies for the source and the target data by including all word types results in several problems. First of all the embedding matrices grow large and might be hard to fit into memory, and it will be difficult to learn a good representation for infrequent words. To remedy this only the top-k (e.g. $k = 50000$) most frequent words can be included, and excluded words replaced with a special `<unk>` token. This has the downside that we can only produce translations using the most frequent words, and that here and there an `<unk>`-token appears in the output.

As an initial solution the `<unk>`-tokens can be replaced with the source-token mostly attended to in that time step. (Jean et al., 2015; Luong et al., 2015b), but this does not fully solve the problem of having to translate with an open vocabulary.

Sennrich et al. (2016b) propose encoding rare and unknown words as a sequence of sub-word units. The vocabulary of sub-word units is computed from the training data using a Byte Pair Encoding (BPE) method. That name is slightly misleading because the method operates over characters and not bytes. We start with all characters found in our training set as our symbols. Each word is represented as a sequence of characters, plus a special end-of-word symbol. Then, for a fixed number steps, we perform:

1. Count all symbol pairs
2. Merge the most frequent one into a new symbol

The number of steps (merge operations) determines the number of created sub-word units, and is the only hyperparameter of the procedure. A popular number of steps is 32000 (Wu et al., 2016a), but the best number could be lower and needs to be determined experimentally (Ding et al., 2019).

2.3 EVALUATION

So far we have not discussed how to evaluate the quality of our translations. How do we know our system is working well? The best but expensive way is to ask humans to evaluate the translations, but this is slow and expensive. Among many proposals for automatic evaluation, the Bilingual Evaluation Understudy (BLEU) score of Papineni et al. (2002) is the only metric that is consistently used in the literature. The assumption is that good translations resemble the translations of a professional translator in some way. We therefore require one or more reference translations for each source sentence in an evaluation data set, and some way to score each translation. BLEU is based on modified n-gram precision, which is computed on a per-sentence basis. Papineni et al. (2002) discuss the following example:

```
Candidate:   the the the the the the the.
Reference 1: The cat is on the mat.
Reference 2: There is a cat on the mat.
```

A simple unigram precision would result in a score of 7/7, since each word (`the`) in the candidate appears in the references. However, the candidate is obviously a bad translation. Modified n-gram precision fixes this by clipping the total count of each candidate word by the maximum count of that word in one of the references:

$$\text{count}_{\text{clip}}(\text{ngram}) = \min\Big(\text{count}(\text{ngram}), \text{max\_ref\_count}(\text{ngram})\Big) \tag{2.36}$$

The modified unigram precision of the candidate then becomes 2/7. We can perform a similar computation using bi-grams, tri-grams, etc. More generally, modified n-gram precision is defined as:

$$p_n = \frac{\sum_{C \in \text{candidates}} \sum_{\text{ngram} \in C} \text{count}_{\text{clip}}(\text{ngram})}{\sum_{C' \in \text{candidates}} \sum_{\text{ngram}' \in C'} \text{count}(\text{ngram}')} \tag{2.37}$$

The BLEU metric uses unigrams, bi-grams, tri-grams and 4-grams, and combines each of the scores using a geometric mean. Since modified n-gram precision does not penalize for getting the sentence length wrong, there is an additional brevity penalty (BP). This makes the final BLEU definition as follows:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leqslant r \end{cases} \tag{2.38}$$

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{4} \frac{1}{4} \log p_n\right) \tag{2.39}$$

ALTERNATIVE METRICS. Various other automatic metrics have been proposed, each aiming to reach better correlation with human judgments. METEOR (Denkowski and Lavie, 2014) aligns the candidate to the reference using word stems, synonyms, and paraphrases. TER (Snover et al., 2006) measures the editing effort for a human to change the candidate into the reference. BEER (Stanojević and Sima'an, 2014; Stanojević and Sima'an, 2015) uses learning-to-rank training, and exploits character n-grams for measuring lexical accuracy and permutation trees for measuring word order. ChrF (Popović, 2015) is based exclusively on character n-grams. RIBES (Isozaki et al., 2010) is another metric that measures word order differences.

COMPARING BLEU SCORES. One particular issue with BLEU is that, if the pre- and post-processing of the data (e.g. lowercasing, tokenization) is not exactly the same, the scores of different systems cannot be compared. Details like these are not always mentioned in the literature. Post (2018) raises this issue and provides a tool, SacreBLEU[13], to compute BLEU scores from detokenized output. SacreBLEU uses an internal tokenizer, provides a version string on how the score was computed, and contains the references for many popular benchmarks.

2.4 SUMMARY

In this chapter we covered the following topics:

- Neural Machine Translation
  - Recurrent encoder-decoder models
  - Convolutional models
  - Transformer
  - Early NMT models
- Evaluation of MT output

13 `https://github.com/mjpost/sacreBLEU`

Part I

INCORPORATING AND INDUCING STRUCTURE

With the success of deep learning, the field of statistical machine translation abandoned systems that made use of linguistic structures such as dependency graphs, and traded them in for neural models that only need a flat word sequence as input. Do linguistic structures still play a role in modern translation systems? And how would we incorporate them?

3 INCORPORATING LINGUISTIC STRUCTURE

While early models for machine translation using neural networks quickly reached state-of-the-art performance, they treat the translation task as a simple sequence-to-sequence transduction. However, language is hierarchical in nature, and by exploiting that it should be possible to give an inductive bias to a neural network that should be beneficial when processing language. In this chapter we take a look at neural machine translation models, and equip them with a recently proposed mechanism – syntactic graph convolutional networks – so as to condition them on the syntactic structure of the input.

CHAPTER HIGHLIGHTS

Problem Statement

- Current neural machine translation models treat the problem of translation as sequence-to-sequence transduction, without explicitly exploiting the hierarchical nature of language.
- Models with the right inductive bias for a task can be expected to work better on it, especially given less data.

Research Question

- How can we condition neural networks on linguistic structure, in particular neural machine translation models?
- Does having access to syntactic/semantic information improve translation performance?

Research Contributions

- Syntactic Graph Convolutional Networks (GCNs) are proposed for use in neural machine translation.
- Syntactic GCNs are shown to be effective on English-German and English-Czech translation tasks.

3.1 INTRODUCTION

Neural Machine Translation (NMT) (Cho et al., 2014; Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014) is one of the success stories of using deep learning in Natural Language Processing (NLP). As we saw in §2.1, NMT treats the problem of translating from one language into another language as a sequence transduction problem: an encoder neural network reads in a sequence of words and a decoder network produces the translation one word at a time. Quickly after being proposed, NMT systems started to outperform 'traditional' phrase-based approaches on many language pairs in the yearly WMT news translation evaluation campaign (see e.g. Bojar et al., 2016 and Sennrich et al., 2016a), especially after adding an attention mechanism (Bahdanau et al., 2015; Luong et al., 2015c).

NMT models are also referred to as encoder-decoders or more generally sequence-to-sequence models.

Despite their early successes, encoder-decoders do not explicitly model syntax. They process their input and output sequentially, despite evidence that language is of a hierarchical nature.[1] For example, in Berwick and Chomsky (2016), we find the following two example sentences:

1. birds that fly *instinctively* swim
2. *instinctively* birds that fly swim

Just by looking at the words of sentence (1) it is not clear if the word 'instinctively' applies to 'fly' or to 'swim'. However, in the second sentence, 'instinctively' unambiguously modifies 'swim'. While 'instinctively' is closer to 'fly' in word distance, it is closer to 'swim' in terms of *structural distance*. The word 'fly' is simply one level deeper than the word 'swim', as we can see when we look at its tree structure in Figure 3.1[2]:

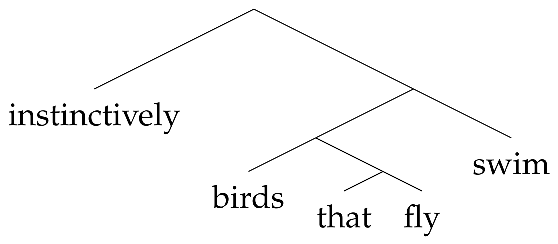


Figure 3.1: Example of hierarchical structure in language.

The reason that the aforementioned NMT systems ignore syntax is possibly the lack of simple and effective methods for in-

1 Just how much hierarchy is a matter of debate. See e.g. Frank et al. (2012).
2 Based on Figure 4.1 in Berwick and Chomsky (2016).

corporating structured information in neural sentence encoders at the time they were conceived, and the lack of such structure in the data. While alternative methods do exist, they are either only implicitly encode structure using MTL (Eriguchi et al., 2017; Hashimoto and Tsuruoka, 2017a; Luong et al., 2015a; Nădejde et al., 2017) or are too restrictive by explicitly encoding phrases (Eriguchi et al., 2016). We will discuss these alternatives at the end of this chapter.

Our goal is to provide the encoder with access to rich syntactic information. We will let the model decide which aspects of syntax are beneficial for the translation task, and do not place hard constraints on the interaction between syntax and our model. We do so because placing hard syntactic constraints tends to hurt performance for phrase-based SMT systems (Chiang, 2010; Smith and Eisner, 2006; Zollmann and Venugopal, 2006), and it is plausible that the same claims hold for NMT.

Attention-based NMT systems (Bahdanau et al., 2015; Luong et al., 2015c) represent source sentence words as continuous feature vectors in the encoder and use these vectors when generating a translation. We would like to automatically incorporate information about the *syntactic neighborhood* of source words into these feature vectors, and with that potentially improve quality of the translation output. Dependency trees represent syntactic relations between words. For example, in Figure 3.2, *Sherpa* is the subject of the predicate *climbed*, and *mountain* is its object. Using dependency trees to obtain syntactic neighborhoods is a natural choice, since our feature vectors correspond to words.

In order to produce syntax-aware feature representations of words, we exploit a Graph Convolutional Network (GCN) (Defferrard et al., 2016; Duvenaud et al., 2015; Kearnes et al., 2016; Kipf and Welling, 2016). GCNs are explained in more detail in §3.3, but we give a simple introduction here. A GCN can compute a latent feature representation for each node in a given graph. The representation of a node is based on its k-th order neighborhood, i.e., nodes at most k hops away from the

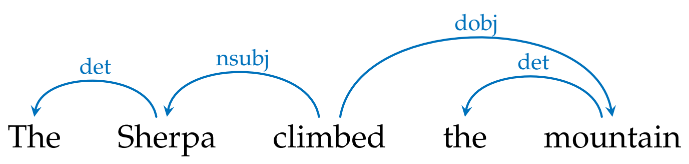


Figure 3.2: An example dependency tree.

node (Gilmer et al., 2017).[3]. In our case our nodes are words in a sentence and the graph is given by a dependency tree. A GCN is generally simple and computationally inexpensive, and here we can simply see it as a layer that takes in a word representation and produces an enhanced word representation.

To support the kinds of graphs for linguistic structures, we propose using a specific type of GCN, the Syntactic GCN, a variant sensitive to edge direction and edge labels. Syntactic GCNs were also shown to be effective for semantic role labeling by Marcheggiani and Titov (2017). Since we can see a GCN as an extra layer enhancing our word representations, it is straightforward to add one to the encoder of an attention-based encoder-decoder model, because we already have a continuous representation for each source word. NMT models are trained end-to-end, so a GCN ends up capturing linguistic properties (provided by the graph) that are useful for translation specifically. Although we could start with word embeddings as initial word representations, we will see that it is more effective to use a GCN layer on top of a RNN or CNN layer, enriching their states with syntactic information.

In this chapter we will focus on dependency syntax and dependency-based semantic-role labeling structures (Surdeanu et al., 2008) as our graphs, but we could also condition on AMR semantic graphs (Banarescu et al., 2012)[4] and co-reference chains. We will cover the following research questions:

- How can we condition neural networks on linguistic structure, in particular neural machine translation models?
- Does having access to syntactic/semantic information improve translation performance?

In the remainder of this chapter, we recap neural machine translation (§3.2), describe the GCN (§3.3) and syntactic GCN (§3.4), define our models (§3.5), perform machine translation experiments (§3.6), discuss related work (§3.7), and finally we conclude (§3.8).

3 GCNs are one type of Graph Neural Networks; others have been proposed as well e.g. Graph Attention Networks (Veličković et al., 2018) and Gated Graph Neural Networks (Li et al., 2016b). These kinds of graph networks are related and can be unified under a message passing framework (Gilmer et al., 2017).

4 See Beck et al. (2018) for an example of conditioning on AMR.

3.2 NEURAL MACHINE TRANSLATION

We use the attention-based NMT model of Bahdanau et al. (2015) as our baseline. Even though we already discussed it in §2.1, we will briefly recap it here.

Given a dataset $\mathcal{D}$ consisting of $|\mathcal{D}|$ i.i.d. sentence pair observations $\langle x = x_1, \ldots, x_M, y = y_1, \ldots, y_N \rangle$, we train a neural network $f_\theta(\cdot)$ with parameters θ to minimize the negative log-likelihood

$$-\frac{1}{|\mathcal{D}|} \sum_{\langle x,y \rangle \in \mathcal{D}} \log p_\theta(y \mid x)$$

using gradient descent (Bottou and LeCun, 2004; Robbins and Monro, 1951).

We describe our NMT baselines for this chapter in terms of the encoder, the decoder, and the attention mechanism that conditions the decoder on the encoder. While one baseline is identical to Bahdanau et al. (2015), in others we modify the encoder.

3.2.1 *Encoders*

An encoder is a function that takes a source sentence as input and produces a sequence of continuous representations, one vector for each word. We will use recurrent, convolutional and bag-of-words encoders, and will recap them briefly.

RECURRENT. As we saw in §2.1.1, an RNN can be defined with the following recursive function:

$$\mathbf{h}_j = \text{RNN}(E^{(\text{src})}\mathbf{x}_j, \mathbf{h}_{j-1})$$

where $E^{(\text{src})}$ is a learned word embedding matrix, and $\mathbf{x}_j$ the one-hot encoded input word at time step j.

We will use the function RNN in an abstract way, and it could be realized using a LSTM (Graves, 2013; Hochreiter and Schmidhuber, 1997) or a GRU (Cho et al., 2014). In this chapter we follow Bahdanau et al. (2015) in using a bidirectional GRU, consisting of a forward (f) and a backward (b) GRU, capturing the past and future context of each word:

$$\mathbf{h}_j = [\mathbf{h}_j^f; \mathbf{h}_j^b] \qquad \forall_{j \in 1,..,M} \qquad \mathbf{h}_j \in \mathbb{R}^{2d} \tag{3.1}$$

where d is the hidden size of each GRU.

CONVOLUTIONAL. A CNN encoder (described in §2.1.3.1) applies a fixed-size window over the input sequence to capture the local context of each word (Gehring et al., 2016, 2017b).

$$\mathbf{h}_j = W[E^{(\text{src})}\mathbf{x}_{j-\lfloor k/2 \rfloor}; \ldots; E^{(\text{src})}\mathbf{x}_j; \ldots; E^{(\text{src})}\mathbf{x}_{j+\lfloor k/2 \rfloor}]$$

where k is the size of the window. One advantage of this approach over the RNN is that it allows for fast parallel computation, while sacrificing sensitivity to non-local context. To remedy the loss of context, multiple CNN layers can be stacked.

BAG OF WORDS. In a bag-of-words (BoW) encoder each word is simply represented by its word embedding. To give the decoder a sense of word position, position embeddings (PE) are added. There are different strategies for defining position embeddings, and in this chapter we choose to learn a vector (randomly initialized) for each absolute word position up to a certain maximum length. We will use a maximum length of 50 here, and use the last position embedding for all positions greater than that.[5] We represent the j-th word as follows:

$$\mathbf{h}_j = E^{(\text{src})}\mathbf{x}_j + \mathbf{p}_j$$

where $E^{(\text{src})}$ is the word embedding matrix, $\mathbf{x}_j$ a one-hot vector, and $\mathbf{p}_j$ the j-th position embedding.

3.2.1.1 *Decoder*

A decoder produces the target sentence conditioned on the source sentence representations. We will use the decoder of Bahdanau et al. (2015), which is implemented as a GRU conditioned on an additional input $\mathbf{c}_i$, the context vector, which is dynamically computed at each time step using an attention mechanism. The probability of a target word y_i is now a function of the decoder RNN state, the previous target word embedding, and the context vector. See §2.1.2.2 for more details.

3.3 GRAPH CONVOLUTIONAL NETWORKS

In the previous section we discussed three different sentence encoders: recurrent, convolutional, and bag-of-words. We now shift our focus to methods that we can use to condition on hierarchical structure in the form of a graph. We start by covering

5 See 2.1.3 for more background on position encodings.

the Graph Convolutional Network (GCN) as proposed by Kipf and Welling (2016)[6], before modifying it.

A GCN is a multi-layer neural network that operates directly on a graph, encoding information about the neighborhood of a node as a real-valued vector. In each GCN layer, information flows along edges of the graph; in other words, each node receives messages from all its *direct* neighbors.

When multiple GCN layers are stacked on top of each other, information from further away in the graph gets integrated. For example, imagine we apply two GCN layers to a graph with nodes, where each node is represented by a vector. In the second layer, a node will receive information from its direct neighbors, but this information already includes information from their respective neighbors. The number of GCN layers regulates the distance the information travels: with k layers a node receives information from neighbors at most k hops away.

Formally, consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of n nodes, and $\mathcal{E}$ is a set of edges. Every node is assumed to be connected to itself, i.e. $\forall v \in \mathcal{V} : (v, v) \in \mathcal{E}$. Now, let $X \in \mathbb{R}^{d \times n}$ be a matrix containing all n nodes with their features, where d is the dimensionality of the feature vectors. In our case, X will contain word embeddings, but in general it can contain any kind of features. For a 1-layer GCN, the new node representations are computed as follows:

$$\mathbf{h}_v = \rho\left(\sum_{u \in \mathcal{N}(v)} W\mathbf{x}_u + \mathbf{b}\right) \tag{3.2}$$

where $W \in \mathbb{R}^{d \times d}$ is a weight matrix and $\mathbf{b} \in \mathbb{R}^d$ a bias vector.[7] ρ is an activation function, e.g. a ReLU. $\mathcal{N}(v)$ is the set of neighbors of v, which we assume here to always include v itself. As stated before, to allow information to flow over multiple hops, we need to stack GCN layers. The recursive computation is as follows:

$$\mathbf{h}_v^{(j+1)} = \rho\left(\sum_{u \in \mathcal{N}(v)} W^{(j)}\mathbf{h}_u^{(j)} + \mathbf{b}^{(j)}\right) \tag{3.3}$$

where j indexes the layer, and $\mathbf{h}_v^{(0)} = \mathbf{x}_v$.

6 For a comprehensive overview of alternative GCN architectures, see Gilmer et al. (2017).

7 We dropped the normalization factor used by Kipf and Welling (2016), as it is not used in syntactic GCNs of Marcheggiani and Titov (2017).

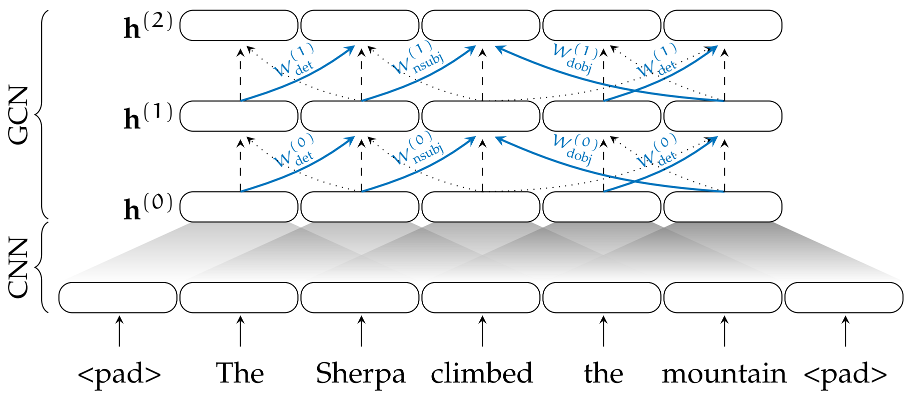


Figure 3.3: A 2-layer syntactic GCN on top of a CNN. Loop connections are depicted with dashed edges, syntactic ones with solid (dependents to heads) and dotted (heads to dependents) edges. Gates and some labels are omitted for clarity.

3.4 SYNTACTIC GCNS

Marcheggiani and Titov (2017) generalize the GCN to operate on directed and labeled graphs.[8] This makes it possible to use linguistic structures such as dependency trees, where directionality and edge labels play an important role. They also integrate edge-wise gates which let the model regulate contributions of individual dependency edges. We will now look at these modifications, since we will use them in our proposed models.

DIRECTIONALITY. In order to deal with directionality of edges, separate weight matrices are used for incoming and outgoing edges. We follow the convention that in dependency trees heads point to their dependents, and thus *outgoing* edges are used for head-to-dependent connections, and *incoming* edges are used for dependent-to-head connections. Modifying the recursive computation for directionality, we arrive at:

$$\mathbf{h}_v^{(j+1)} = \rho\left(\sum_{u \in \mathcal{N}(v)} W^{(j)}_{\mathrm{dir}(u,v)} \mathbf{h}_u^{(j)} + \mathbf{b}^{(j)}_{\mathrm{dir}(u,v)}\right) \tag{3.4}$$

where $\mathrm{dir}(u,v)$ selects the weight matrix associated with the directionality of the edge connecting u and v (i.e. W_{IN} for u-to-v, W_{OUT} for v-to-u, and W_{LOOP} for v-to-v). Note that self loops

8 For an alternative approach to integrating labels and directions, see applications of GCNs to statistical relation learning (Schlichtkrull et al., 2018).

are modeled separately, so there are now three times as many parameters as in a non-directional GCN.

LABELS. Making the GCN sensitive to labels is straightforward given the above modifications for directionality. Instead of using separate matrices for each direction, separate matrices are now defined for each direction and label combination:

$$\mathbf{h}_v^{(j+1)} = \rho\left(\sum_{u \in \mathcal{N}(v)} W_{\mathrm{lab}(u,v)}^{(j)} \mathbf{h}_u^{(j)} + \mathbf{b}_{\mathrm{lab}(u,v)}^{(j)}\right) \tag{3.5}$$

where we incorporate the directionality of an edge directly in its label.

Importantly, to prevent over-parametrization, only bias terms are made label-specific, in other words: $W_{\mathrm{lab}(u,v)} = W_{\mathrm{dir}(u,v)}$. The resulting syntactic GCN, shown on top of a CNN (see next section), is illustrated in Figure 3.3.

EDGE-WISE GATING. Syntactic GCNs also include gates, which can down-weight the contribution of individual edges. They also allow the model to deal with noisy predicted structure, i.e. to ignore potentially erroneous syntactic edges. For each edge, a scalar gate is calculated as follows:

$$g_{u,v}^{(j)} = \sigma\left(\mathbf{h}_u^{(j)} \cdot \hat{\mathbf{w}}_{\mathrm{dir}(u,v)}^{(j)} + \hat{b}_{\mathrm{lab}(u,v)}^{(j)}\right) \tag{3.6}$$

where σ is the logistic sigmoid function, and $\hat{\mathbf{w}}_{\mathrm{dir}(u,v)}^{(j)} \in \mathbb{R}^d$ and $\hat{b}_{\mathrm{lab}(u,v)}^{(j)} \in \mathbb{R}$ are learned parameters for the gate. The computation becomes:

$$\mathbf{h}_v^{(j+1)} = \rho\left(\sum_{u \in \mathcal{N}(v)} g_{u,v}^{(j)} \left(W_{\mathrm{dir}(u,v)}^{(j)} \mathbf{h}_u^{(j)} + \mathbf{b}_{\mathrm{lab}(u,v)}^{(j)}\right)\right) \tag{3.7}$$

3.5 MODELS

In this chapter we focus on exploiting structural information on the source side in the encoder. The hypothesis is that using an encoder that incorporates syntax will lead to more informative representations of words, and that these representations, when used as context vectors by an attentive decoder, will lead to an improvement in translation quality as measured by BLEU. Consequently, we will use the decoder of Bahdanau et al. (2015)

and keep that part of the model constant.[9] In all models we use the GRU (Cho et al., 2014) as the recurrent unit.

We will now discuss three models with encoders of increasing complexity. Each of these models employs a GCN to condition on linguistic structure. They are different in how much information the word representations contain when they are fed to the GCN.

3.5.1 *Model 1: BoW + GCN*

The first and simplest model consists of a bag-of-words encoder (see §3.2.1)) with a GCN on top. The inputs to the GCN are therefore the word embeddings summed with the position embedding for their absolute position in the sentence. Since this encoder captures the linear ordering information only in a very crude way (through the position embeddings), the structural information provided by GCN should be highly beneficial.

3.5.2 *Model 2: Convolutional + GCN*

The second model consists of a CNN encoder with a GCN on top, as shown in Figure 3.3. A CNN encoder is fast, but by definition only uses a limited window of context for each word. Instead of the approach used by Gehring et al. (2016), i.e., stacking multiple CNN layers on top of each other, we use a GCN to enrich the one-layer CNN representations. Note that, while Figure 3.3 shows a CNN with a window size of 3, we will use a larger window size of 5 in our experiments. We expect this model to perform better than *BoW + GCN*, because of the additional local context captured by the CNN.

3.5.3 *Model 3: Recurrent + GCN*

The third and most powerful model employs a bidirectional GRU. After encoding the source sentence with it, we use the resulting hidden states as input to a GCN. As mentioned before, this is the most challenging setup, as RNNs have been shown capable of capturing at least some degree of syntactic information without explicit supervision (Linzen et al., 2016), and

9 Note that we do not use a maxout layer in the decoder, as is common practice.

hence they should be hard to improve upon by incorporating treebank syntax.

3.5.4 *Advantages of using GCN layers*

Instead of relying on linear order only, the GCN layers will allow the encoder to 'teleport' over parts of the input sentence, along dependency edges, connecting words that otherwise might be far apart. The model might not only benefit from this teleporting capability however; also the nature of the relations between words (e.g. dependency relation types) may be useful, and the GCN can exploit this information (see §3.4 for details).

3.5.5 *Multiple GCN layers*

Marcheggiani and Titov (2017) did not observe improvements from using multiple GCN layers when conditioning a semantic role labeling model on syntactic structure. However, propagating information from further in the tree should be beneficial in principle. We hypothesize that the first layer is the most influential one, capturing most of the syntactic context, and that additional layers only modestly modify the representations. To ease optimization, we add a residual connection (He et al., 2016) between the GCN layers, when using more than one layer:

$$\mathbf{h}^{(l+1)} = \mathrm{GCN}(\mathbf{h}^{(l)}) + \mathbf{h}^{(l)}$$

where l is the index of the layer, and $\mathbf{h}^{(0)}$ is the input to the first GCN layer.

3.6 EXPERIMENTS

Experiments are performed using the Neural Monkey toolkit[10] (Helcl and Libovický, 2017), which implements our baseline model (Bahdanau et al., 2015) in TensorFlow[11]. The modifications to Neural Monkey for running the experiments in this chapter, including the Syntactic GCN, are available on Github.[12]

HYPERPARAMETERS. We use the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.001 (0.0002 for CNN

10 https://github.com/ufal/neuralmonkey
11 https://www.tensorflow.org/
12 https://github.com/bastings

models[13]). The batch size is set to 80. Between layers we apply dropout with a probability of 0.2, and in experiments with a GCN we use the same value for edge dropout. We train for 45 epochs, evaluating the BLEU performance of the model every epoch on the validation set. For testing, we select the model with the highest validation BLEU. L2 regularization is used with a value of 10^{-8}. All the model selection (incl. hyperparameter selections) was performed on the validation set. In all experiments we obtain translations using a greedy decoder, i.e., we greedily select the output token with the highest probability at each time step.

In the following, we will first conduct an artificial reordering experiment (§3.6.1), and then machine translation experiments using dependency syntax (§3.6.2) and (in addition) semantic role labeling structures (§3.6.3).

3.6.1 *Reordering artificial sequences*

Before we perform machine translation experiments, we would like to get an intuition for the capabilities of GCNs. We define a reordering task where randomly permuted sequences need to be put back into the original order. We encode the original order using edges, and test if a GCN-based model can successfully exploit them. Note that this task is not meant to provide a fair comparison to a recurrent baseline. The input (besides the edges) simply does not carry any information about the original ordering, so our baselines cannot possibly solve this task, since they lack a mechanism to condition on the edges.

DATA. From a vocabulary of 26 types (a-z), we generate random sequences of 3-10 tokens. We then randomly permute them, pointing every token to its original predecessor with a label sampled from a set of 5 labels. Additionally, we point every token to an *arbitrary* position in the sequence with a label from a distinct set of 5 'fake' labels. We sample 25000 training and 1000 validation sequences.

MODEL. We use the *Recurrent + GCN* model, i.e., the encoder is a bidirectional GRU with a 1-layer GCN on top. We use 32, 64 and 128 units for embeddings, GRU units and GCN layers, respectively.

13 As Gehring et al. (2016) also observed, Adam seems to be too aggressive for CNN models, hence we use a lower learning rate.

RESULTS. After 6 epochs of training, the model learns to put permuted sequences back into order, reaching a validation BLEU of 99.2. Figure 3.4 shows that the mean values of the bias terms of gates (i.e., $\hat{b}$ in Equation 3.6) for real and fake edges are far apart, suggesting that the GCN learns to distinguish them. Interestingly, this illustrates why edge-wise gating is beneficial. A model without gates would not understand which of the two outgoing arcs is fake and which is genuine, because only biases b would then be label-dependent. Consequently, it would only do a mediocre job in reordering. Although using label-specific matrices W would also help, this would not scale to the real scenario (see §3.4).

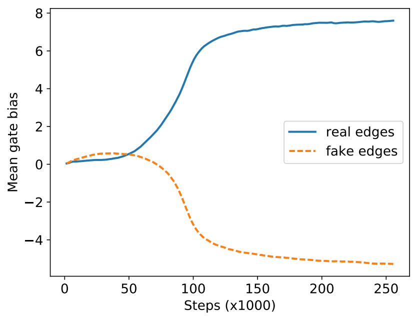


Figure 3.4: The mean gate bias for real (useful) and fake (non-useful) labels suggests the GCN learns to distinguish them.

3.6.2 *Syntax-aware Neural Machine Translation*

The artificial reordering experiment in the previous section shows that a GCN layer can do what it was designed for: conditioning on structural information, including labels, and exploiting it when useful to solve a task. We now turn to a real scenario, and condition a machine translation model on syntactic information in the form of dependency trees.

DATA. For the machine translation experiments we use the English-German and English-Czech News Commentary v11 data from the WMT16 translation task (Bojar et al., 2016).[14] For English-German we also train on the full WMT16 data set to test our hypothesis in a large data set scenario. As our validation set and test set we use `newstest2015` and `newstest2016`, respectively.

14 http://www.statmt.org/wmt16/translation-task.html

PRE-PROCESSING. The English sides of the corpora are tokenized and parsed into dependency trees by SyntaxNet,[15] using the pre-trained Parsey McParseface model. The Czech and German sides are tokenized using the Moses tokenizer.[16] Sentence pairs where either side is longer than 50 words are filtered out after tokenization.

VOCABULARIES. For the English sides, we construct vocabularies from all words except those with a training set frequency smaller than three. For our target languages Czech and German, to deal with rare words and phenomena such as inflection and compounding, we learn Byte Pair Encoding (BPE) codes (see §2.2) as described by Sennrich et al. (2016b). Given the size of our data sets, and following Wu et al. (2016b), we use 8000 BPE merge operations to obtain robust frequencies for our subword units. For the larger full WMT data set we use 16000 BPE merge operations. Data set statistics are summarized in Table 3.1 and vocabulary sizes in Table 3.2.

	Train	Val.	Test
English-German	226822	2169	2999
English-German (full)	4500966	2169	2999
English-Czech	181112	2656	2999

Table 3.1: The number of sentences in our data sets.

	Source	Target
English-German	37824	8099 (BPE)
English-German (full)	50000	16000 (BPE)
English-Czech	33786	8116 (BPE)

Table 3.2: Vocabulary sizes.

HYPERPARAMETERS. We use 256 units for word embeddings, 512 units for GRUs (800 for En-De full data set experiment), and 512 units for convolutional layers (or equivalently, 512 'channels'). The dimensionality of the GCN layers is equivalent to

15 https://github.com/tensorflow/models/tree/master/research/syntaxnet
16 https://github.com/moses-smt/mosesdecoder

the dimensionality of their input. We report results for 2-layer GCNs, as we find them most effective (see ablation studies below).

BASELINES. We provide three baselines, each one with a different encoder: (1) a bag-of-words encoder, (2) a convolutional encoder with window size $w = 5$, and (3) a recurrent baseline that uses a bidirectional GRU. All baselines use the decoder of Bahdanau et al. (2015). See §3.2.1 for details. To account for model size we add an extra non-linear layer, equivalent to a GCN with self-loop only, after encoding the source sentence.

EVALUATION. We report (cased) BLEU scores (Papineni et al., 2002) using `multi-bleu`[17]. To evaluate lexical selection we also report BLEU_1 scores, where only unigrams are considered when computing BLEU (see §2.3). BLEU_4 is the standard BLEU score using $n = 1,..,4$. To evaluate reordering performance we report Kendall τ reordering scores (see e.g. Stanojević and Sima'an, 2015). We found that TER (Snover et al., 2006), BEER (Stanojević and Sima'an, 2014), and RIBES (Isozaki et al., 2010) (see §2.3) metrics were consistent with those results and resulted in the same ranking.

3.6.2.1 *Results*

ENGLISH-GERMAN. Table 3.3 shows test results on English-German. Unsurprisingly, the bag-of-words baseline performs the worst. We expected the BoW+GCN model to make easy gains over this baseline, which is indeed what happens. BLEU_1 and BLEU_4 scores go up by +4.3 and +2.7, respectively. Next we look at the convolutional encoders. The CNN baseline reaches a higher BLEU_4 score than the BoW models, but interestingly its BLEU_1 score is lower than the BoW+GCN model. The CNN+GCN model improves over the CNN baseline by +1.9 and +1.1 for BLEU_1 and BLEU_4, respectively. The BiRNN, the strongest baseline, reaches a BLEU_4 of 14.9. Interestingly, GCNs still manage to improve the result by +2.3 BLEU_1 and +1.2 BLEU_4 points. Finally, we observe a big jump in BLEU_4 by using the full English-German WMT'16 data set and beam search (with a beam size of 12). The recurrent baseline (RNN) now reaches 23.3, while adding a GCN achieves a score of 23.9.

17 Equivalent to SacreBLEU with these settings: `BLEU+case.mixed+numrefs.1+smooth.exp+tok.none+version.1.3.6`

	Kendall	$BLEU_1$	$BLEU_4$
BoW	0.3352	40.6	9.5
+ GCN	0.3520	44.9	12.2
CNN	0.3601	42.8	12.6
+ GCN	0.3777	44.7	13.7
RNN	0.3984	45.2	14.9
+ GCN	0.4089	47.5	16.1
RNN (full)	0.5440	53.0	23.3
+ GCN	0.5555	54.6	23.9

Table 3.3: Test results for English-German.

	Kendall	$BLEU_1$	$BLEU_4$
BoW	0.2498	32.9	6.0
+ GCN	0.2561	35.4	7.5
CNN	0.2756	35.1	8.1
+ GCN	0.2850	36.1	8.7
BiRNN	0.2961	36.9	8.9
+ GCN	0.3046	38.8	9.6

Table 3.4: Test results for English-Czech.

ENGLISH-CZECH. Table 3.4 shows test results on English-Czech. While it is difficult to obtain high absolute BLEU scores on this dataset, we can still see similar relative improvements. Again the BoW baseline scores worst, with the BoW+GCN easily beating that result. The CNN baseline scores $BLEU_4$ of 8.1, but the CNN+GCN improves on that, this time by +1.0 and +0.6 for $BLEU_1$ and $BLEU_4$, respectively. Interestingly, $BLEU_1$ scores for the BoW+GCN and CNN+GCN models are higher than both baselines so far. Finally, the recurrent baseline scores a $BLEU_4$ of 8.9, but it is again beaten by the RNN+GCN model with +1.9 $BLEU_1$ and +0.7 $BLEU_4$.

EFFECT OF GCN LAYERS. How many GCN layers do we need? Every layer gives us an extra hop in the graph and expands the syntactic neighborhood of a word. Table 3.5 shows valida-

	En-De		En-Cs	
	$BLEU_1$	$BLEU_4$	$BLEU_1$	$BLEU_4$
RNN	44.2	14.1	37.8	8.9
+ GCN (1L)	45.0	14.1	38.3	9.6
+ GCN (2L)	46.3	14.8	39.6	9.9

Table 3.5: Validation BLEU for English-German and English-Czech for 1- and 2-layer GCNs.

tion BLEU scores as a function of the number of GCN layers. For English-German, using a 1-layer GCN improves $BLEU_1$, but surprisingly has little effect on $BLEU_4$. Adding an additional layer gives improvements on both $BLEU_1$ and $BLEU_4$ of +1.3 and +0.73, respectively. For English-Czech, performance increases with each added GCN layer. Performance stopped improving when adding a third layer.

EFFECT OF SENTENCE LENGTH. We hypothesize that GCNs should be more beneficial for longer sentences: these are likely to contain long-distance syntactic dependencies which may not be adequately captured by RNNs but directly encoded in GCNs. To test this, we partition the validation data into five buckets and calculate BLEU for each of them. Figure 3.5 shows that GCN-based models outperform their respective baselines rather uniformly across all buckets. This is a surprising result. One explanation may be that syntactic parses are noisier for longer sentences, and this prevents us from obtaining extra improvements with GCNs.

DISCUSSION. The results suggest that the syntax-aware representations provided by the GCN-enhanced models consistently lead to improved translation performance as measured by $BLEU_4$. Other metrics, such as TER, BEER, and RIBES, show the same trend and also rank the GCN models the highest. Consistent gains in terms of Kendall tau and $BLEU_1$ indicate that improvements correlate with better word order and lexical (BPE subword) selection, two phenomena for which syntax is crucial.

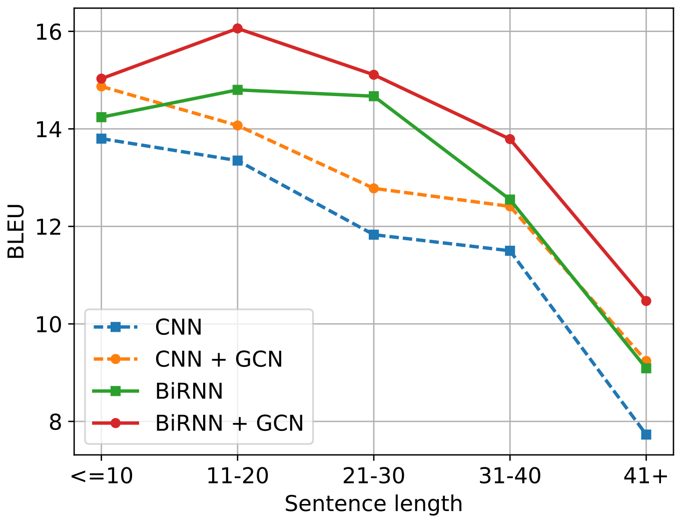


Figure 3.5: Validation BLEU per sentence length.

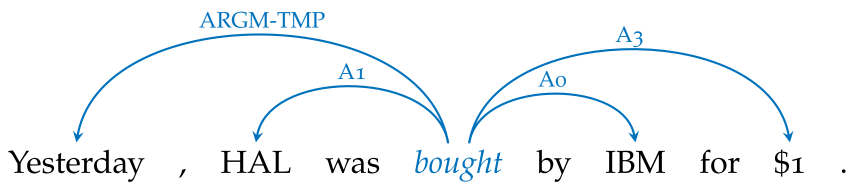


Figure 3.6: Example sentence with semantic role structure.

3.6.3 *Linguistically-informed NMT*

In the previous section we conditioned NMT models on dependency syntax, which gave rise to the name 'syntax-aware NMT'. While the results suggested that this was successful, dependency trees are not the only linguistic structure that we can condition on to bias our model. As mentioned before, a GCN can condition on any kind of graph structure. Based on Marcheggiani et al. (2018), in this section we briefly look at what happens when we condition on dependency-based Semantic Role Labeling (SRL) structures. We can condition on these semantic structures alone, or we can combine them with the dependency syntax that we used in the previous section. Rather than calling this approach 'semantics-aware' NMT, we will adopt the term 'linguistically-informed', proposed later by Strubell et al. (2018) in the context of SRL, and define it to mean conditioning on syntax and/or semantics. Syntax and semantics bring complementary information: where syntax shows how the sentence is structured, the semantic structure describes its meaning.

Semantic structures abstract away from the surface form of a sentence, and can provide us with the answers to the question: Who did what to whom, when and where? For example, imagine we wanted to find events on the web that describe the big company A buying another company B. We could find a sentence like 'A buys B' but we could also find 'The amazing B bought by A for $1'. Many more surface forms could describe the same events, using synonyms, active or passive constructions, etc. Figure 3.6 shows the SRL structure for an example sentence, identifying the semantic role 'to buy', the actor `A0` (the buyer, IBM), the patient `A1` (the thing being bought, HAL), the amount `A3`, and the time `ARGM-TMP`. If we expressed the sentence in a different way, we would still obtain the same semantic structure.

Like machine translation itself, the idea of exploiting semantics in MT is an old one (Bar-Hillel, 1960; Weaver, 1955).[18] And even though semantics has been exploited in phrase-based SMT systems (Aziz et al., 2011; Baker et al., 2012; Bazrafshan and Gildea, 2013; Jones et al., 2012; Liu and Gildea, 2010; Wu and Fung, 2009), this study is the first to do so for NMT.

In the following, we will use Propbank-style SRL structures (Palmer et al., 2005), to be precise their dependency version (Surdeanu et al., 2008) as shown in Figure 3.6, to condition our NMT models on semantics. Our hypothesis is that such semantic structures can improve translation quality, and that they provide complementary improvements compared to when conditioning on syntax alone.

MODELS. We use the two best performing models from the previous section: the convolutional and the recurrent models. Like in the previous section, the GCN has three weight matrices: W_{IN}, W_{OUT}, and W_{LOOP}. When we condition on both syntax and semantics, we first apply a GCN layer for syntax, and then another GCN layer (with separate parameters) for semantics. However, another way to combine the information from the syntactic and semantic graphs is to share the self-loop weights

18 However, Bar-Hillel (1960) also write: "Fully automatic, high quality translation is not a reasonable goal, not even for scientific texts." We can do better now than could be envisioned back then, even without linguistic input.

W_{LOOP}, and use graph-specific W_{IN}, W_{OUT} matrices. We change Equation 3.7 to compute $\mathbf{h}_v^{(j+1)}$ as follows:

$$\rho\Big(W_{\text{LOOP}}^{(j)}\mathbf{h}_v^{(j)} + \sum_{u\in\mathcal{N}_{syn}(v)}\big(W_{\text{dir}(u,v)}^{(j)}\,\mathbf{h}_u^{(j)}\big) + \sum_{u\in\mathcal{N}_{sem}(v)}\big(W_{\text{dir}(u,v)}^{\prime(j)}\,\mathbf{h}_u^{(j)}\big)\Big) \quad (3.8)$$

where $W_{\text{dir}(\cdot)}$ are used for the syntactic graph, $W'_{\text{dir}(\cdot)}$ for the semantic graph, and W_{LOOP} is shared. Bias terms and gates are left out for clarity.[19] In this case (only) $\mathcal{N}(v)$ does not contain node v itself. Later, in the ablation experiments, we will see that the stacking approach works better. In the following all test results are therefore reported using the stacking approach.

DATA. We will use the English-German NewsCommentary v11 and WMT'16 data from the previous section, with the same pre-processing (see §3.6.2). To obtain SRL structures, we parse the English side of the data set with the neural SRL parser of Marcheggiani et al. (2017).

EXPERIMENTS. Table 3.6 shows the results for models trained on News Commentary. While conditioning on SRL structures (`+Sem`) helps, BLEU does not increase as much over the baseline as it does for dependency syntax (`+Syn`). Interestingly though, combining syntax and semantics does improve over conditioning on semantics alone, while it gives a modest boost for the CNN models. Table 3.7 shows the results for the full WMT'16 setting. Somewhat surprisingly, given that they have generally fewer edges, conditioning on SRL graphs gives better results now than conditioning on dependency syntax, while conditioning on both gives the best result. It is possible that with more data, more syntactic information is captured by the RNN, while SRL structures remain useful.

ANALYSIS. To select the hyperparameters and the best way to combine syntactic and semantic structures, we performed various ablation experiments on the News Commentary data set. Table 3.8 shows the results. First, we can see that removing the self-loop for the baseline results in the same performance, showing that just adding extra depth is not helpful, and that improvements come from syntax/semantics. We can also see that stacking a semantic GCN on top of syntactic one (`+Syn+Sem`, as

19 Gates for semantic edges are computed analogously to the syntactic gates, but with separate parameters.

	RNN	CNN
Baseline	14.9	12.6
+Sem	15.6	13.4
+Syn	16.1	13.7
+Syn + Sem	15.8	14.3

Table 3.6: Test BLEU, En–De, News Commentary v11.

	RNN
Baseline	23.3
+Sem	24.5
+Syn	23.9
+Syn + Sem	24.9

Table 3.7: Test BLEU, En–De, full WMT'16.

done in the above experiments) works better than incorporating both graphs inside a *single* GCN layer (`+SemSyn`). One explanation for why the stacked approach works better is because it might allow for a greater interaction between the syntactic and semantic layers.

3.7 RELATED WORK

The proposals in this chapter are not the only ones that aim to incorporate linguistic structures in NMT or other neural models for different NLP tasks. We review various related methods, both for incorporating linguistic features on the source side (in the encoder) as for the target side (in the decoder).

LINGUISTIC INPUT FEATURES. A simpler way to incorporate linguistic features is to add them as additional input features to the word representations. Sennrich and Haddow (2016) use features such as POS-tags, lemmas and dependency labels. Each feature has its own embedding matrix, and the distributional representation of a feature is then concatenated to the word embedding. While not the same as conditioning on the full dependency tree, embedding the dependency label with a word together with other features makes for improvements of about one BLEU point on WMT'16 German-English, English-

	RNN	CNN
Baseline (with 1L self loop)	14.1	12.1
Baseline (without self loop)	14.1	12.1
+Sem (1L)	14.3	12.5
+Sem (2L)	14.4	12.6
+Sem (3L)	14.4	12.7
+Syn (2L)	14.8	13.1
+SemSyn (1L)	14.1	12.7
+Syn (1L) + Sem (1L)	14.7	12.7
+Syn (1L) + Sem (2L)	14.6	12.8
+Syn (2L) + Sem (1L)	14.9	13.0
+Syn (2L) + Sem (2L)	14.9	13.5

Table 3.8: Validation BLEU, News Commentary only

German, and English-Romanian. Vanmassenhove and Way (2018) use the same technique to incorporate semantic supersense tags and syntactic supertag features. The technique can also be used to simply indicate if a word is cased or not (i.e., an extra feature concatenated to the word embedding indicates if the word starts with a capital), as is done by Levin et al. (2017), which could be useful when optimizing for robustness, as it allows for using the same word embedding for a word invariant of its case, without losing the information whether the word was cased or not originally. Alternatively, it is also possible to simply add tokens in the input sequence that mark a feature, such as *case* in this case, as done by Bérard et al. (2019). This has the downside of making the sequence longer, but allows using an NMT system out of the box.

LINGUISTIC OUTPUT FEATURES. Garcia-Martinez et al. (2016) use factors on the decoder/target side, in contrast to Sennrich and Haddow (2016) who use them on the encoder/source side. They predict the lemma of each output word together with a number of factors. After prediction these are then mapped to an inflected word with a mapping function defined a priori. The approach can increase the effective vocabulary size and reduce unknown words, and was shown to be competitive with BPE in a spoken language translation scenario. Tamchyna et al.

(2017) propose a similar method of predicting lemmas and morphological features. As they predict the lemmas and features as an interleaved sequence, they can use a standard NMT model, in contrast to Garcia-Martinez et al. (2016).

TREE-TO-SEQUENCE. Eriguchi et al. (2016) parse English sentences with an HPSG parser and then use a Tree-LSTM (Tai et al., 2015) to encode the internal nodes of the tree. They still use a standard decoder, but now word and node representations have to compete under the same attention mechanism. The approach proposed in this chapter, conditioning on a graph using a GCN, can be done in parallel, and results in the same number of representations as before applying the GCN. In contrast, Eriguchi et al. (2016) obtain a representation for each node in the tree, including leaves, and the computation cannot be done in parallel. It is not trivial how the information from internal nodes and leaf-nodes is best incorporated into the NMT model, hence they are treated as equals in this model. Both approaches require parsing the source side of the parallel data.

LINEARIZED PARSE TREES. Instead of employing a modeling technique, an alternative way to represent a tree is to linearize it, and then use a sequence-to-sequence model to encode or decode it. Aharoni and Goldberg (2017) propose neural string-to-tree by predicting linearized parse trees instead of predicting output words alone. For example, the target output sequence for 'Jane had a cat.' would be `(ROOT (S (NP Jane )NP (VP had (NP a cat )NP )VP . )S )ROOT`. Currey and Heafield (2018a) serialize the parse trees of source sentences in a similar manner and use hierarchical attention (Libovický and Helcl, 2017) to learn source representations based on both the original (word) input sequence as well as the serialized parse tree. Currey and Heafield (2019) again condition on both sequences and serialized parse trees, but this time using a Transformer (Vaswani et al., 2017), and obtain modest gains in BLEU in low-resource scenarios, but not on larger data sets.

MULTI-TASK LEARNING. In Multi-task Learning (MTL) (Caruana, 1997) a model (or a part thereof) is supervised by multiple objectives, so that the learning of one task benefits from the others. This can be achieved e.g. by using the encoded source or target word representations for an additional prediction task. Sharing model parameters with a syntactic parser is a popular approach to obtaining syntax-aware representations in NMT. Luong et al. (2015a) predict linearized constituency parses as an additional task. Eriguchi et al. (2017) multi-task with a target-side RNNG parser (Dyer et al., 2016) and improve on various language pairs with English on the target side. Nădejde et al. (2017) multi-task with CCG tagging, and also integrate syntax on the target side by predicting a sequence of words interleaved with CCG supertags.

SYNTACTIC CONSTRAINTS. Stahlberg et al. (2016) prune a lattice from a hierarchical phrase-based model (`Hiero`) and use it to constrain the search space of an NMT decoder. Hiero trees are not syntax-aware, but instead constrained by symmetrized word alignments.

LATENT STRUCTURE. Hashimoto and Tsuruoka (2017a) add a syntax-inspired encoder on top of a bidirectional LSTM layer. They encode source words as a learned average of potential parents emulating a relaxed dependency tree. While their model is trained purely on translation data, they also experiment with pre-training the encoder using treebank annotation and report modest improvements on English-Japanese. We will discuss this model in more detail in the next chapter.

GRAPH-TO-SEQUENCE. After this work was first published, others have explored further uses of GCNs in NLP. Beck et al. (2018) propose an alternative method to condition on graphs using Gated Graph Neural Networks instead of GCNs. As in this chapter, they condition on dependency trees, but also AMR semantic graphs. In contrast to our approach, theirs allows edges to have their own representations, which allows them to be incorporated in a richer way. Zhang et al. (2018) use GCNs to improve relation extraction using pruned dependency trees. Damonte and Cohen (2019) and Song et al. (2018) use them to condition on AMR graphs to generate sentences that more accurately convey the semantics captured by the graph. Alon et al. (2019) generate natural language from code snippets, conditioning on the underlying code structure with GCNs. Finally, De Cao et al. (2019) propose a question answering setup where entities are nodes in a graph, and relations such as co-reference are edges, and use GCNs to enable multi-step reasoning.

3.8 conclusion

In this chapter we presented a simple and effective approach to integrating graph-based linguistic structures such as dependency trees and dependency-based SRL structures into NMT models. We saw consistent BLEU improvements for two challenging language pairs: English-German and English-Czech.

Since GCNs are capable of encoding any kind of graph-based structure, their applicability is not limited to the structures we covered in this chapter. This was demonstrated by subsequent work, where they were used e.g., to condition on AMR (Beck et al., 2018), structured code snippets (Alon et al., 2019), and co-reference structures for question answering (De Cao et al., 2019).

One downside of the presented approach is that it requires a method to obtain the structures to condition on. In the next chapter, we will look at inducing these structures, so that they no longer need to be provided as input.

4 INDUCING LATENT STRUCTURE

In the previous chapter we saw that linguistic structure predicted by a supervised parser can be beneficial for NMT. In this chapter we will investigate a more challenging setup: we incorporate sentence structure as a *latent variable* in a standard NMT encoder-decoder and induce it in such a way as to benefit the translation task, eliminating the need for supervised parsers.

CHAPTER HIGHLIGHTS

Problem Statement

- Linguistic structure can be beneficial for NMT, but requires the translation data set to be parsed.
- These structures are defined over *words*, but state-of-the-art NMT systems rely on sub-word units.

Research Question

- Can we eliminate the need for supervised parsers, and *induce* a latent structure over a (sub-)word sequence instead?
- How do we induce such a structure, and does it benefit the translation task?

Research Contributions

- We present a model with two separate components: a graph sampler and a translation component, with completely separate parameters.
- We show that it is possible to induce useful graphs over sub-word sequences for simpler encoders such as bag-of-words and convolutional encoders, but that on top of LSTM encoders the sampled graphs become trivial.

4.1 INTRODUCTION

In the previous chapter we used a Graph Convolutional Network (GCN) to encode linguistic inductive bias about the syntactic and semantic structure of the source sentence, which was then exploited by an attentive decoder. We saw that this added inductive bias can be beneficial for Neural Machine Translation (NMT). To apply our GCN we had to provide an external syntactic and/or semantic parse for each source sentence at training and test time. Other works, e.g. Eriguchi et al. (2016) and Hashimoto and Tsuruoka (2017b), have shown that different methods such as multi-task learning can also be successful at adding such a bias. In this chapter we want to get rid of our reliance on supervised parsers, and consider a more challenging setting: We will incorporate sentence structure as a latent variable in a standard NMT encoder-decoder and induce it in such a way as to benefit the translation task.

Inducing latent structure while incurring a downstream loss was previously explored for tasks such as sentiment analysis and textual entailment (Choi et al., 2018; Kim et al., 2017; Maillard and Clark, 2018; Yogatama et al., 2017), and mainly focuses on latent trees rather than (more generally) graphs. Interestingly, Williams et al. (2018) showed that these learned tree structures do not correspond to syntactic or semantic generalizations, but that they can be as useful as having access to predicted parses.

Our goal is to investigate under which conditions induced latent structures can be beneficial for NMT. Although we would like these structures to be discrete (for example for better interpretability), we do not enforce discreteness in order to avoid high-variance estimators. Instead, we induce structure in the form of weighted densely-connected graphs that can exhibit various degrees of sparsity.

We propose a probabilistic model with two components:

1. a *graph component* that stochastically samples a latent graph conditioned on the source sentence;

2. a graph-informed *translation component* that conditions on the sampled graph and the source sentence to predict the target sentence using a recurrent decoder.

Figure 4.1 shows the architecture. Using two distinct components lets us disentangle their effects and study in which conditions useful structure gets induced. To that end, we keep the

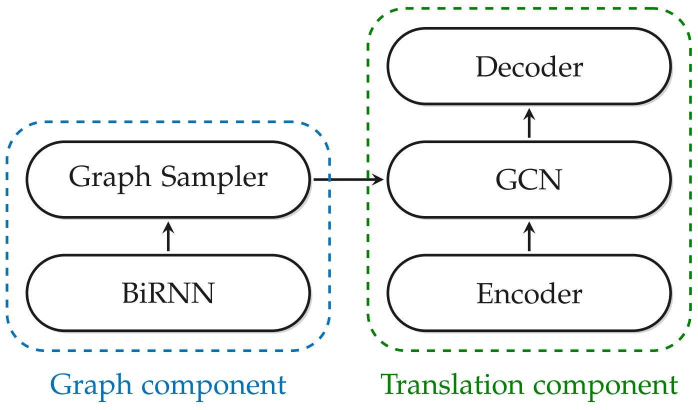


Figure 4.1: Model architecture.

architecture of the graph component fixed across experiments and vary the encoder of the translation component (e.g. RNN, CNN, or embeddings). We observe that with RNNs, likely due to their expressiveness, the model makes no or very limited use of the latent graph apparatus. In contrast, with CNN encoders the model finds purpose to latent graphs such as encoding useful, potentially long-distance, dependencies in the source sentences.

4.2 BACKGROUND

There are a few new concepts in this chapter that require our attention before we continue with the model definition: probabilistic and deep generative models, the reparameterization trick, and the Concrete distribution.

4.2.1 *Deep Generative Models*

In contrast to the discriminative models that we have seen so far, a generative model is described in terms of a collection of random variables (RVs), and we are interested in learning the joint distribution over all those variables (Kingma and Welling, 2019). While the generative model describes the data generation process, it can be turned into a discriminator using Bayes' rule. We can take an auto-encoder (see e.g. Vincent et al., 2010) as an example. A discriminative auto-encoder assumes the input $\mathbf{x}$ given, projects it into a continuous space $\mathbf{z}$, and predicts a reconstruction $\mathbf{x}'$ from $\mathbf{z}$. It is trained with an objective that encourages $\mathbf{x}'$ to be close to the original $\mathbf{x}$: the reconstruction loss. Importantly, it does not model the input data itself, and therefore has no idea about its distribution. In contrast, the generative view would be that x was generated from a latent variable z, and so we are interested in modeling $p(x, z) = p(x \mid z)\, p(z)$. Figure 4.2 shows the graphical model. One advantage is that now we can express what kind of distribution $p(z \mid x)$ should be, for example a Gaussian distribution $\mathcal{N}(\mu; \sigma^2)$, where mean μ and variance σ^2 are predicted by the neural network. When a neural network predicts the parameters of a distribution for our model, we call it a Deep Generative Model (DGM), and a Deep Latent-Variable Model (DLVM) in particular when the model involves a latent variable. We can also express a preference for the shape of $p(z \mid x)$ by keeping it close to a prior $p(z)$ on expectation, e.g. $\mathcal{N}(0; 1)$, a Gaussian with mean 0 and variance 1. During training our goal is to learn the model parameters θ, so as to maximize the *marginal likelihood*

$$p_\theta(x) = \int p_\theta(x \mid z)\, p_\theta(z)\, dz \tag{4.1}$$

However, this is intractable since we need to integrate over the entire latent space z. In the next section we will see how we can learn the parameters without this intractable marginalization. We also solve an important issue, which is what we are truly interested in for this chapter: because during training z is stochastically sampled from (an approximation of) the distribution $p_\theta(z \mid x)$, backpropagation breaks. We will see how a Variational Auto-Encoder (VAE) solves this issue using the *reparameterization trick*, and after our exposition of the VAE, we will use that trick in our model to sample latent graphs without breaking backpropagation in §4.3.

4.2.2 *Variational Auto-Encoders and the Reparameterization Trick*

The VAE was concurrently proposed by Kingma and Welling (2014) and Rezende et al. (2014), and provides a method to train deep generative models, such as the one we just described, without the need to integrate over the latent space z. We will cover the most important parts of the VAE based on Kingma and Welling (2019), and refer to that work for further details and derivations.

To train our model, we turn it into a discriminator: we follow the generative view in the opposite direction, and go from $x \to z \to x'$. So, we need to compute the *posterior distribution*

$$p_\theta(z \mid x) = \frac{p_\theta(x, z)}{p_\theta(x)} \tag{4.2}$$

but it is intractable, since $p_\theta(x)$ is intractable. However, we can approximate $p_\theta(z \mid x)$ using an *inference model* $q_\phi(z \mid x)$ that is also a neural network but with its own variational parameters ϕ. Crucially, we choose q_ϕ to be tractable, and fit its parameters with the goal that $q_\phi(z \mid x) \approx p_\theta(z \mid x)$. Formally, we use the inference model to optimize a lowerbound of the marginal log-likelihood:

$$\log p_\theta(x) = \mathcal{L}_{\phi,\theta}(x) + \mathrm{KL}\big[q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)\big] \tag{4.3}$$

where $\mathcal{L}$ is called the evidence lower bound (ELBO), computed as:

$$\mathcal{L}_{\phi,\theta}(x) = \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x, z)}{q_\phi(z \mid x)}\right] \tag{4.4}$$

From Eq. 4.3 it becomes clear that the ELBO is indeed a lower bound, since

$$\mathcal{L}_{\phi,\theta}(x) = \log p_\theta(x) - \mathrm{KL}\big[q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)\big] \tag{4.5}$$

with the KL-term being 0 at best (when $q_\phi(z \mid x) = p_\theta(z \mid x)$), in which case $\log p_\theta(x)$ remains. By minimizing the negative ELBO, we maximize a lowerbound of the log likelihood, while at the same time minimizing the KL-divergence between the approximation $q_\phi(z \mid x)$ and the true posterior $p_\theta(z \mid x)$.

We can now estimate the gradient using Monte Carlo, taking a single sample from $q_\phi(z \mid x)$ when we need to compute its expectation. However, it is not yet clear how gradients flow through the sampling procedure, and we will discuss that next.

REPARAMETERIZATION TRICK. One issue that arises is that we sample from $q_\phi(z \mid x)$, which breaks backpropagation since we cannot compute a gradient through a sample. This is solved by Kingma and Welling (2014) by using a reparameterization trick, also called a 'change of variables'. A sample from $q_\phi(z \mid x)$ is expressed as a (differentiable) transformation of distribution parameters ϕ, x, and ϵ from a fixed noise source. For example, for Gaussian q, $\epsilon \sim \mathcal{N}(0, 1)$, then $z = \mu + \sigma * \epsilon$. Whenever the

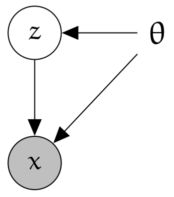


Figure 4.2: A simple graphical model: the variational auto-encoder. Each observation x is generated from a latent variable z.

CDF of a distribution is invertible, we can use this trick to obtain a sample from it, by first sampling Uniform noise and then transforming it using the inverse CDF. We will use the trick when we sample a latent graph later in this chapter. For further reading, see Kingma and Welling (2019).

4.2.3 *The Concrete Distribution*

In this chapter we are interested in predicting a structure, a latent graph, for each source sentence. The structures that we used in the previous chapter, dependency trees and semantic role labeling structures, were both discrete. If we want to replace those structures, then it makes sense to look for a distribution that allows us to sample discrete vectors.

A first candidate could be using the Gumbel-Max trick. Our goal is to sample from a categorical distribution D with unnormalized class probabilities $\boldsymbol{\alpha} = \alpha_1, \dots, \alpha_k$:

$$D \sim \text{Categorical}\left(\frac{\boldsymbol{\alpha}}{\alpha_0}\right) \tag{4.6}$$

with $\alpha_0 = \sum_{i=1}^{k} \alpha_i$. To do so we start with k Gumbel random variables G_1^k, expressed as a transformation of a Uniform random variable:

$$G_k = -\log(-\log(U_k)) \qquad U_k \sim \mathrm{Uniform}(0,1) \tag{4.7}$$

The Gumbel-Max trick says that we can obtain a discrete random variable as follows:

$$\text{Categorical}\left(\frac{\boldsymbol{\alpha}}{\alpha_0}\right) \sim \arg\max_k \log \alpha_k + G_k \tag{4.8}$$

where α_k is the potential for $P(X = k)$. In other words, we can obtain a discrete sample by sampling Gumbel noise, adding $\log \alpha_k$ to it, and taking the arg max.[1] This is what we wanted,

1 The choice of Gumbel noise seems arbitrary, but it has a theoretical justification. See e.g. Hazan et al. (2016).

however we still have a remaining issue: we cannot backpropagate through the arg max operator.

The Concrete distribution (Jang et al., 2017; Maddison et al., 2017), also called the Gumbel-Softmax distribution, starts out with the Gumbel-Max trick that we just discussed, but then smooths the arg max operator in order to let gradients pass through. It is defined as follows:

$$R(\boldsymbol{\alpha}, \lambda) \sim \text{softmax}\left[\frac{\log(\boldsymbol{\alpha}) + [G_1, \dots, G_k]}{\lambda}\right] \tag{4.9}$$

where $\boldsymbol{\alpha}$ are the parameters of the distribution, and λ is the softmax temperature. This gives a k-dimensional vector of positive numbers that sum to 1. Compared to the Gumbel-Max trick, the arg max is replaced by a smoother tempered softmax, that becomes approximately discrete as the temperature $\lambda \to 0$. Higher temperatures give us less variance, while lower temperatures give sparser samples. Hence, Jang et al. (2017) suggest annealing the temperature during training, even though they do not do so in their paper. Note how, just like the reparameterization trick that we discussed for the VAE, the Concrete distribution transforms noise from a fixed random source; it just happens to use Gumbel noise instead of Uniform noise.

Now that we have a distribution, Concrete, from which we can get approximately discrete samples (i.e., as the temperature is lowered more and more values will be close to zero), we are ready to formalize our model.

4.3 MODEL

Our model is a Deep Generative Model (DGM); a probabilistic model whose components are parameterized by neural networks. In this chapter we will therefore describe machine translation from a probabilistic perspective. We view the source sentence as a random sequence X_1^m, and the target sentence as a random sequence Y_1^n, and model the conditional likelihood of source-target random sequences. Random variable X takes on values in vocabulary $\mathcal{X}$ of the source language, and random variable Y takes on values in vocabulary $\mathcal{Y}$ of the target language. To each source word, we associate a (latent) random variable Z which selects a position in the source sentence as that word's *head*.[2] Formally, Z takes on values in the set of

2 We use the word *head* in allusion to syntactic heads in dependency graphs, but note that our notion of head is purely data-driven.

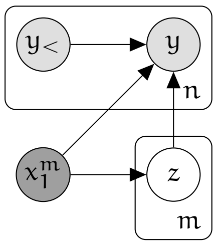


Figure 4.3: Conditional independences of the latent graph model.

source word positions $\mathcal{Z} = \{1, \ldots, m\}$. A sequence Z_1^m of head variables can be seen as a sequence of m directed edges, each from X_i to X_{Z_i}, and for that reason we interpret it as a random *graph*.

We start by defining a joint distribution over target sentences and graphs:

$$\begin{aligned} P(y_1^n, z_1^m \mid x_1^m) &= P(z_1^m \mid x_1^m) P(y_1^n \mid x_1^m, z_1^m) \\ &= \underbrace{\prod_{i=1}^{m} P(z_i \mid x_1^m)}_{\text{Graph component}} \times \underbrace{\prod_{j=1}^{n} P(y_j \mid x_1^m, z_1^m, y_{<j})}_{\text{Translation component}} \end{aligned} \tag{4.10}$$

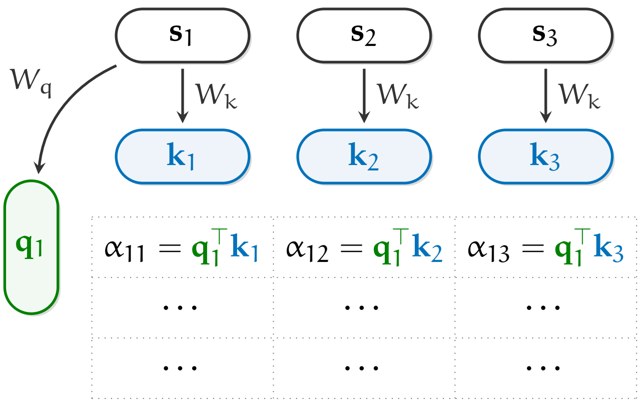
where we first generate a graph z_1^m conditioned on the source sentence, and then generate a translation conditioned on the source and the graph, by generating one target word at a time without Markov assumptions.

Figure 4.3 shows the graphical model. We can see that the source sentence is observed (shaded gray), and that the latent graph (m head distributions z) conditions on it. The target sentence y_1^n conditions on both the source sentence x_1^m and the latent graph z_1^m. In the next sections will look at the model components in more detail.

The conditional likelihood of observations is obtained by marginalizing all possible latent graphs:

$$\begin{aligned} P(y_1^n \mid x_1^m) &= \sum_{z_1=1}^{m} \cdots \sum_{z_m=1}^{m} P(y_1^n, z_1^n \mid x_1^m) \\ &= \mathbb{E}_{P(Z_1^m \mid x_1^m)} \left[P(y_1^n \mid x_1^m, Z_1^m) \right] . \end{aligned} \tag{4.11}$$

Since the translation model makes no Markov assumption, this marginalization is intractable. While we could use a differentiable and unbiased estimator such as the score function method (Glynn, 1990; Williams, 1992) (also known as REINFORCE), this



$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
W_q
W_k
W_k
W_k
$\mathbf{k}_1$
$\mathbf{k}_2$
$\mathbf{k}_3$
$\mathbf{q}_1$
$\alpha_{11} = \mathbf{q}_1^\top \mathbf{k}_1$
$\alpha_{12} = \mathbf{q}_1^\top \mathbf{k}_2$
$\alpha_{13} = \mathbf{q}_1^\top \mathbf{k}_3$

Figure 4.4: Computation of head potentials (Concrete parameters) $\boldsymbol{\alpha}$.

typically suffers from high variance.[3] In the following, we will present an alternative formulation that enables efficient inference and parameter estimation by means of a continuous relaxation and reparameterized gradients.

4.3.1 *Graph Component*

The graph component conditions on the source sentence x_1^m and samples for each source position i an m-dimensional probability vector

$$Z_i \mid x_1^m \sim \text{Concrete}(\boldsymbol{\alpha}_i, \lambda) \tag{4.12}$$

whose k^{th} component z_{ik} represents the relative strength of the edge from x_i to x_k. Then, altogether, z_1^m can be seen as the adjacency matrix of a weighted fully-connected graph over the source words. By analogy to dependency parsing, we can see each z_i as the parameter vector of a Categorical distribution over the candidate *heads* of x_i, which is why we call the Concrete parameter $\boldsymbol{\alpha}_i \in \mathbb{R}^m$ a vector of *head potentials*. Given a sequence of source word embeddings, we obtain hidden states $\hat{\mathbf{s}}_1^m$ using a bi-directional LSTM (Graves and Schmidhuber, 2005; Schuster and Paliwal, 1997). From these hidden states, we then create 'key' and 'query' (or 'head' and 'dependent', by analogy) representations for each state $\hat{\mathbf{s}}_i$ using linear projections:

$$\mathbf{k}_i = \mathbf{W}_K \, \hat{\mathbf{s}}_i \qquad \mathbf{q}_i = \mathbf{W}_Q \, \hat{\mathbf{s}}_i \tag{4.13}$$

3 Effective variance control techniques do exist (Gu et al., 2015; Tucker et al., 2017), but they typically require multiple assessments of likelihood terms, which for memory-intensive NMT models is undesirable.

with learned parameters $\mathbf{W}_K, \mathbf{W}_Q \in \mathbb{R}^{d_K \times d}$. We then obtain head potentials using a scaled dot product:

$$\alpha_{ik} = \begin{cases} \frac{1}{\sqrt{d_K}} \mathbf{q}_i^\top \mathbf{k}_k & \text{if } i \neq k \\ -\infty & \text{if } i = k \end{cases} \tag{4.14}$$

Figure 4.4 shows this graphically. Similar projections are used by Dozat and Manning (2017) and Vaswani et al. (2017). Importantly, they break the symmetry of the dot product, which is crucial to model a *directed* graph.[4]

As we saw in §4.2, the Concrete density also takes a temperature parameter λ, which we made a global hyperparameter, with a decaying scheme described in §4.4.

4.3.2 *Translation Component*

The translation component conditions on the source sentence x_1^m, a sampled graph z_1^m, and a target prefix $y_{<j}$ to sample a target word

$$Y_j \mid x_1^m, z_1^m, y_{<j} \sim \text{Categorical}(\boldsymbol{\pi}_j) \tag{4.15}$$

for $j = 1, \ldots, n$. We use an attention-based encoder-decoder similar to the one in §3.5 to compute the Categorical parameters $\boldsymbol{\pi}_j$ at each time step j. First we obtain an encoding $\mathbf{s}_1^m$ of the source sentence, which is independent of the representations used by the graph component, and then we use a GCN to enhance these representations given the neighborhood defined by the graph z_1^m. After obtaining such enriched representations we employ a standard attentive decoder (Luong et al., 2015c).

ENCODER. Just as we did in §3.5, we experiment with three different encoders for the translation component:

1. word embeddings summed with *position encodings*;
2. a single-layer convolutional encoder (Gehring et al., 2017a), with window size 5, summed with position encdoings;
3. a bi-directional LSTM.

In contrast to §3.5, this time our position encodings are fixed time series as proposed by Vaswani et al. (2017), instead of a learned parameter vector per position.

4 We mask out the diagonal ($i = k$) to demote induction of trivial edges (from a word to itself).

GRAPH CONVOLUTION. We now employ a Graph Convolutional Network (GCN) (Bastings et al., 2017; Marcheggiani and Titov, 2017), also used in the previous chapter, to incorporate graph z_1^m into source word representations $\mathbf{s}_1^m$:

$$\mathbf{s}_i = \text{GCN}(\mathbf{s}_1^m, z_1^m)[i] \tag{4.16}$$

Since we induce unlabeled graphs, we do not use any label-specific GCN parameters. This means that the GCN has a particularly simple update rule:

$$\text{GCN}\left[S = \mathbf{s}_1^m, Z = z_1^m\right] = \text{ReLU}\left(ZSW + SW_{\text{loop}} + \mathbf{b}\right) \tag{4.17}$$

where we leave out gates for clarity.

GATES. Two scalar gates in [0, 1] are computed for each word representation $\mathbf{s}_j$: one inhibits information from the self-loop connections, and one inhibits information coming *from* word j. We can define the gates as a diagonal matrices:

$$G = \text{diag}\left(\sigma(W_g S)\right) \tag{4.18}$$

$$G_{\text{loop}} = \text{diag}\left(\sigma(W_{g_{\text{loop}}} S)\right) \tag{4.19}$$

$$\text{GCN}\left[S = \mathbf{s}_1^m, Z = z_1^m\right] = \text{ReLU}\left(ZGSW + G_s SW_{\text{loop}}\right) \tag{4.20}$$

where gates G scale the strengths of the *columns* in adjacency matrix Z, and the self-loop gates G_{loop} scale the *rows* of the representations SW_{loop}. In other words, if the gate for word j in G is 0.9, then column j in Z is multiplied with 0.9 by computing ZG. And if the gate for word j in G_{loop} is 0.5, then row j in SW_{loop} is multiplied with that amount.

The GCN creates an elegant interface between the graph component and the translation component which prevents the former from "leaking" parameters or representations to the latter. Graph $Z = z_1^m$ is the only information that is shared from the graph component to the translation component.

DECODER. Our decoder is based on Luong et al. (2015c); for the jth prediction an LSTM attends to the (graph-informed) source word representations. See §2.1.2.2.

	Train	Dev	Test	Vocabularies
De-En	153K	7282	6750	32010/22823
Ja-En	2M	1790	1812	16384 (SPM)

Table 4.1: Data set statistics.

4.3.3 *Parameter estimation*

We estimate the parameters of our model to maximize a lower bound on marginal likelihood

$$\int p_\theta(z_1^m \mid x_1^m) \log p_\theta(y_1^n \mid x_1^m)\, dz_1^m \tag{4.21}$$

obtained by application of Jensen's inequality. We get unbiased gradient estimates for this objective by sampling a single graph per source sentence. The Concrete density is a location family (Maddison et al., 2017), thus we can reparameterize samples from the graph component, enabling parameter estimation via backpropagation (Kingma and Welling, 2014).

CONNECTION TO VI. Our lowerbound can be seen as a special case of the evidence lowerbound (ELBO) (Jordan et al., 1999), where we choose to use the generative graph model component as a variational approximation, i.e.:

$$q_\phi(z_1^m \mid x_1^m, y_1^n) \triangleq p_\theta(z_1^m \mid x_1^m) \tag{4.22}$$

Under this Variational Inference (VI) view, our model can be seen as an instance of a variational auto-encoder (Kingma and Welling, 2014).

4.4 EXPERIMENTS

We build our models on top of TensorFlow NMT[5] (Luong et al., 2015c) and experiment on German↔English and Japanese↔English tasks. Data set statistics are summarized in Table 4.1.

DE↔EN. We train on IWSLT14 with the same splits and preprocessing as Ranzato et al. (2016). This is a small but popular data set for which we can compare to strong external baselines.

5 https://github.com/tensorflow/nmt

JA↔EN. We train on the Asian Scientific Paper Excerpt Corpus (ASPEC) (Nakazawa et al., 2016) as pre-processed by the WAT 2017 Small-NMT task[6] using SentencePiece[7] (Kudo, 2018). We use the provided dev and test sets, and compare against the benchmark provided by the workshop organizers.

4.4.1 *Baselines*

For our baselines we train our models *without* the graph sampler, varying the encoder. We add a dense layer with ReLU activation and residual connection on top of the encoder, to make our baselines stronger and to keep the number of parameters for the translation component equal.[8] Doing so makes sure that the graph-enhanced models do not benefit from extra depth in the translation component compared to the baselines.

4.4.2 *Hyperparameters*

We optimize using Adam (Kingma and Ba, 2015). For De-En, we use 256 hidden units, a learning rate of 3e-4, and dropout 0.3. For Ja-En, we use 512 units, a learning rate of 2e-4, and dropout 0.2. Word representations (query and key) are projected down to $d_k = 256$ units when calculating head potentials. Our batch size is set to 64. Beam search is used with beam size 10 and with a length penalty of 1.0.

CONCRETE TEMPERATURE. For the graph component we define an initial temperature λ_0 and apply exponential decay based on the number of network updates. After t updates, the temperature is $\lambda_0 \times d^{\lfloor t/t_d \rfloor}$ with decay rate d and decay steps t_d. We set $\lambda_0 = 2$, $d = 0.99$, and t_d 1 epoch.

4.4.3 *Evaluation*

We use SacréBLEU[9] (Post, 2018) to report all BLEU scores. For German-English we report case-sensitive tokenized BLEU scores to compare with previous work. For Japanese-English, we report detokenized BLEU for English using the 13a tokenizer (which is `mteval-v13a` compatible). For Japanese we report tok-

6 http://lotus.kuee.kyoto-u.ac.jp/WAT/WAT2017/snmt/index.html
7 An alternative to BPE sub-word units.
8 This is identical to a GCN layer with self-loops only.
9 https://github.com/mjpost/sacreBLEU

		IWSLT14		WAT17	
	Encoder	De-En	En-De	Ja-En	En-Ja
Ext. baseline	LSTM	27.6	-	-	28.5
Baseline	Emb.	22.7	17.9	18.1	18.1
Baseline	CNN	23.6	19.1	23.0	24.6
Baseline	LSTM	27.6	22.4	26.0	28.7
Latent Graph	Emb.	24.0	18.7	23.2	24.3
Latent Graph	CNN	24.6	20.3	24.6	26.7
Latent Graph	LSTM	27.2	22.4	26.0	29.1

Table 4.2: Latent Graph Results

enized BLEU on the segmentation from SentencePiece in accordance with the Small-NMT shared task.

4.4.4 *Results*

Table 4.2 lists our results. We observe that the baselines with LSTM encoders outperform the CNN ones, to be followed by the word embedding baselines. This is not surprising, since the LSTM is the only baseline that can fully capture the context of a word. The CNN baseline performs surprisingly well, despite having a receptive field of only five words.

We observe that substantial gains in BLEU score can be made when latent graphs are incorporated into models with word embedding and CNN encoders. This suggests that the graphs are capturing useful relations outside of the receptive fields of those encoders. However, for the LSTM encoders the latent graphs do not seem beneficial overall. We look into this in the next section.

4.5 DISCUSSION

What dependencies are the graphs capturing? The analysis of our graphs is somewhat nontrivial as they are not truly discrete and lack gold-truth parse trees.

We first measure the distance between each word and its most-likely head word. If this distance is small on average, then words typically select their neighboring word as head, whereas

	Mean head distance	
Encoder	Ja-En	En-Ja
Emb.	4.0 ±6.9	3.8 ±5.6
CNN	6.1 ±6.5	6.7 ±7.1
RNN	4.3 ±6.5	2.0 ±5.4

Table 4.3: Mean head distance for En-Ja.

	Mean entropy	
Encoder	Ja-En	En-Ja
Emb.	0.49 ±0.18	0.42 ±0.18
CNN	1.21 ±0.28	1.47 ±0.30
RNN	0.51 ±0.20	0.00 ±0.01

Table 4.4: Mean Entropy for En-Ja.

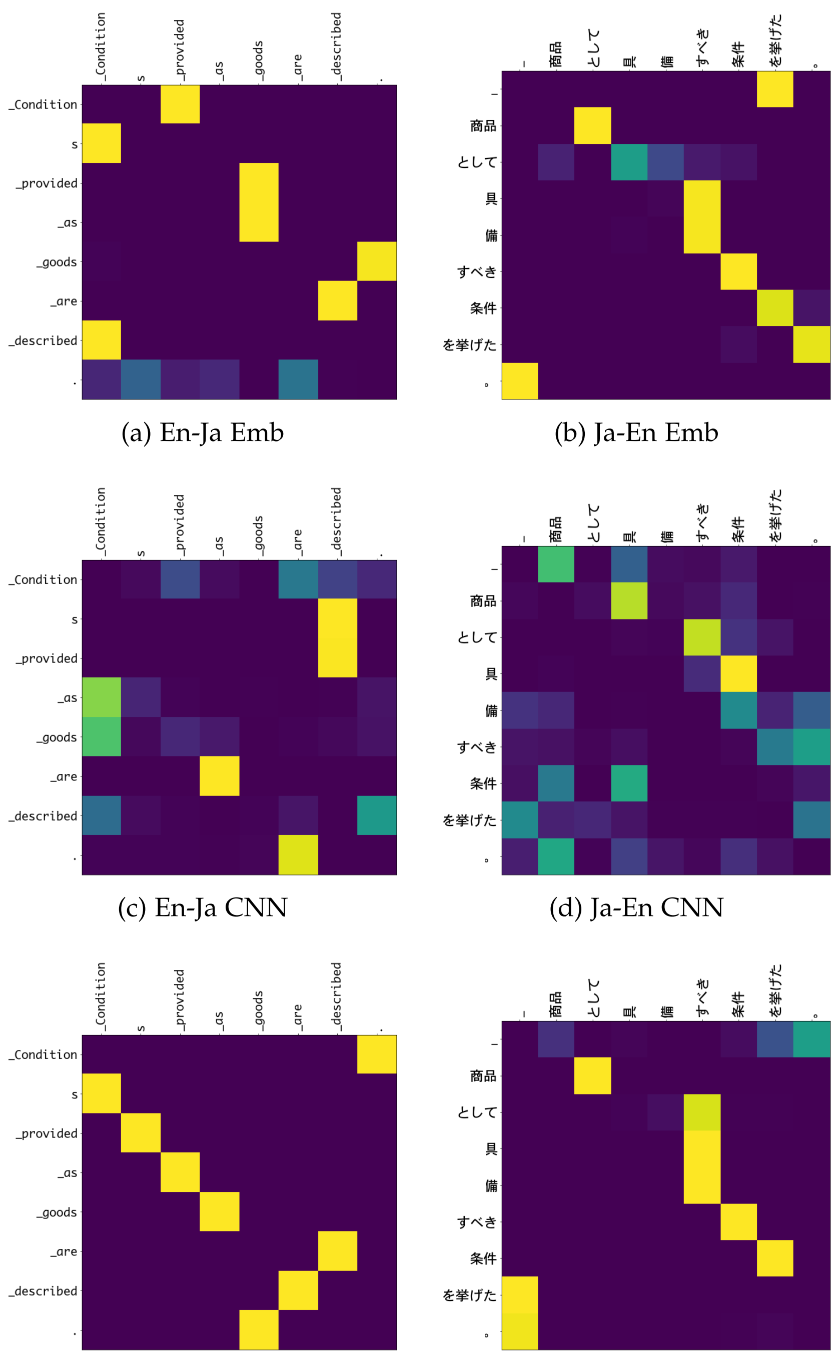
if it is larger then this suggests potentially interesting non-local dependencies. Table 4.3 lists the average head distances for En-Ja, together with the variance over all distances. We find that with LSTM encoders words typically select their heads nearby, whereas with the other encoders heads are also found further away. Figure 4.5 indeed shows this for an example sentence. Inspection reveals that for the LSTM case the graphs became trivial, confirming that it already captures non-local dependencies.

We also wonder how sparse our graphs are. To find out, we interpret the adjacencies in the graph as Categorical head distributions and report average entropy (normalized by sentence length) in Table 4.4. If each word was to select its head uniformly, this would result in a value of 28.7. However, we observe much lower values, indicating that our graphs are relatively sparse.

4.6 RELATED WORK

We discuss related work on inducing graphs in NMT and on inducing trees on other tasks.

Hashimoto and Tsuruoka (2017b) induce latent graphs on the source-side for English-Japanese NMT, optionally pre-training the graphs using Penn treebank parses of Wall Street Journal



(a) En-Ja Emb (b) Ja-En Emb

(c) En-Ja CNN (d) Ja-En CNN

(e) En-Ja RNN (f) Ja-En RNN

Figure 4.5: Example latent graphs for English-Japanese.

articles. The latent graphs are therefore semi-supervised, getting supervision from the treebank but not from the translation data. They report improvements over a vanilla encoder-decoder with attention, especially in lower resource scenarios, of which most improvement seems to be related to the pre-training of the graphs. Since vocabularies are created on the word level, and not the now more common sub-word/BPE level, some tricks are used to deal with resulting large vocabulary sizes, such as speeding up the softmax layer. These tricks and memory requirements make it difficult to make a direct comparison. In contrast, we induce the latent graphs stochastically instead of deterministically, and investigate the conditions under which this is successful with different kinds of encoders.

Tran and Bisk (2018) also induce relaxed graphs deterministically on the source side. However, in contrast to Hashimoto and Tsuruoka (2017b), they use structured attention (Kim et al., 2017; Liu and Lapata, 2018) to obtain differentiable non-projective trees to add a structural bias to the deterministically induced dense graphs. In contrast, our graphs are sparse and stochastic, and consist of independently sampled head distributions.

Both Hashimoto and Tsuruoka (2017b) and Tran and Bisk (2018) induce graphs on top of LSTM-based sentence encodings and attend directly to a transformation of the same encodings and/or additional context vectors. In this chapter, instead, we introduce a clear-cut separation that largely reduces the risk of over-parameterization. Our stochastic induction also opens the possibility to explore other sparsity induction priors (e.g. symmetric Dirichlet). In contrast to Hashimoto and Tsuruoka, we also operate directly on sub-word sequences for En-Ja, eliminating word-level dependency pre-training.

Currey and Heafield (2018b) introduce an unsupervised tree-to-sequence model for NMT. They adapt the Gumbel Tree LSTM of Choi et al. (2018), where (hard) decisions about the tree structure of a sentence are learned using the Straight-through Gumbel-Softmax (Jang et al., 2017; Maddison et al., 2017). A binary Tree LSTM (Le and Zuidema, 2015; Tai et al., 2015; Zhu et al., 2015) is used to merge two children into a parent representation at every step. Since there is no supervision, *each pair* of adjacent nodes is considered for merging, and only one of those is selected by sampling. The process recurses until there is a single root node, at which point the tree is done. This creates a problem where it is unclear how to condition on the resulting tree, just as in the supervised case of Eriguchi et al. (2016). They solve this by propagating information from the root of the tree back into the leaf nodes, which are then used by the decoder. Their method results in improvements over recurrent baselines for low resource language pairs, despite that the induced trees do not resemble linguistic trees.

Apart from work on inducing trees and graphs with NMT as a downstream loss, there has been lots of prior work on inducing trees with other tasks as a downstream loss, mostly on natural language inference (Bowman et al., 2015). All these methods need to provide a solution to the problem of learning to make discrete decisions about the tree structure.

Socher et al. (2011) were the first to both parse a sentence and use the result for a downstream task. However the tree structure was not induced, but supervised with an auxiliary task. Bowman et al. (2016a) propose SPINN, which uses an LSTM to learn a sequence of shift-reduce operations resulting in a binary tree that is then encoded using a Tree LSTM. The tree is given during training time, but can be predicted during test time. Yogatama et al. (2017) use a similar model, but without supervision. They learn the shift-reduce operations using REINFORCE (Williams, 1992) in a completely unsupervised fashion. Maillard and Clark (2018) also model shift-reduce operations, and induce sentence representations by learning a differentiable composition function based on a Tree LSTM that encodes the complete hypothesis space of a binary bracketing chart parser. Choi et al. (2018) use a Straight-Through Gumbel Softmax estimator to learn discrete decisions, and recursively encode a sentence as described above in the context of NMT for Currey and Heafield (2018b). They evaluate on SNLI and sentiment analysis, and find that the resulting models perform at least as well as other models. Williams et al. (2018) reimplement the REINFORCE-based and ST Gumbel-Softmax-based models of Yogatama et al. (2017) and Choi et al. (2018) to compare them within the same code base. They find that only the model of Choi et al. (2018) outperforms a simple LSTM baseline on NLI, that the resulting trees are not consistent across restarts, that the parses tend to be shallower than treebank syntax, and that they do not correspond to Penn Treebank-style parses. However, the learned trees can be as useful (for natural language inference) as having access to predicted parses.

While the previous works mostly focused on NLI and sentiment classification, a few works focus on inducing structure using language modeling as a task. Shen et al. (2018) propose Parsing-Reading-Predict Networks (PRPN), that learn to induce structure while exploiting it to learn a better language model, while using backpropagation to learn the parsing network. The parsing network computes the 'syntactic distance' between all successive words in a sentence, and then makes soft constituent decisions based on that distance. The reading network computes a memory vector relevant for the current time step, and the predict network makes the next token prediction based on all memory vectors syntactically and directly related to the next token. The model not only achieves good language modeling performance, its parses are also highly correlated to the constituents in the Penn Treebank WSJ10 data set, the subset of the Penn Treebank WSJ section that consists of sentences of length 10 or less. Htut et al. (2018) fix a few deficiencies in the experimental setup of Shen et al. (2018), but conclude that the results hold and constitute a success in latent tree learning. Finally, Kim et al. (2019) propose a generative model called Unsupervised Recurrent Neural Network Grammar. They use amortized variational inference to deal with the problem of having to marginalize over all possible latent trees. To provide a structural bias they use a neural CRF constituency parser as their inference network. The model is as good as its supervised counterpart on language modeling for English and Chinese, matches the unsupervised constituency parsing performance of Shen et al. (2018) on English, but does worse on Chinese.

LINGUISTIC STRUCTURE. This chapter is also related to work on exploiting linguistic structure in NMT without inducing it. Section §3.7 gives an overview.

TRANSFORMERS. In this chapter we induced stochastic graphs on the source side, which were then incorporated into an NMT model using a GCN. It is worth noting the relation to Transformers (see Section 2.1.3.2. In a transformer, deterministic graphs ('adjacency matrices') are computed using *self-attention*. The representation of a word ('a node') is then a weighted sum over all representations, where the weight is the value in the adjacency matrix. This can be seen as graph convolution over a dense graph, taking the token representations as nodes. A difference is that in Transformers multiple graphs are independently computed, and the resulting node representations are concatenated and fed through a feed-forward neural network. This process is then repeated for multiple layers. Instead, we learned a single stochastic graph, which we used to enhance word embedding, CNN, and LSTM representations.

4.7 CONCLUSION

In this chapter we presented a probabilistic model with separate graph induction and translation components. We studied if our induced latent graphs are beneficial using three different encoders. In the case of LSTM encoders the induced graphs turned out to be (largely) trivial, while for the simpler word embedding and CNN encoders they contain useful, potentially long-distance dependencies. One explanation for the trivial graphs for the LSTM case is the relation to Transformers, where some heads also show trivial connections, suggesting that structure may be redundant if representations are already rich enough.

Part II

INTERPRETABILITY

Even if deep neural networks give us higher accuracies on natural language processing tasks such as sentiment analysis, they do not tell us *how* they arrive at their predictions. In this part, we delve into *interpretable* neural models, and we study how neural sequence-to-sequence models *generalize*: do they generalize like we humans do?

5 INTERPRETABLE NEURAL PREDICTIONS

Can we trust the predictions that neural networks make? In this chapter we focus on text classifiers and make them more interpretable by having them provide a justification—a *rationale*—for their predictions. We approach this problem by jointly training two neural network models: a latent model that selects a rationale (i.e. a short and informative part of the input text), and a classifier that learns from the words in the rationale alone. We achieve interpretability by knowing which part of the input text is used for prediction, and which part is not.

CHAPTER HIGHLIGHTS

Problem Statement

- Neural networks are good at making classifications, but they do not provide a rationale for how they arrive at their predictions.
- Current methods to jointly learn how to classify and provide a rationale rely on the REINFORCE estimator, which can have high variance.

Research Question

- How can we make text classifiers provide rationales, without resorting to the REINFORCE estimator?
- How can we control the properties, such as the length, of the rationale?

Research Contributions

- We proposed the HardKuma distribution, which allows backpropagation through sampled rationales using a stretch-and-rectify technique.
- Our HardKuma-based rationales achieve better accuracies than the REINFORCE-based ones.

5.1 INTRODUCTION

Neural networks are bringing incredible performance gains on text classification tasks (Devlin et al., 2019; Howard and Ruder, 2018; Peters et al., 2018). However, this power comes hand in hand with a desire for more interpretability, even though its definition may differ (Lipton, 2016). While it is useful to obtain high classification accuracy, with more data available than ever before it also becomes increasingly important to *justify* predictions. Imagine having to classify a large collection of documents, while verifying that the classifications make sense. It would be extremely time-consuming to read each document to evaluate the results, and doing so would only answer the question of *what* predictions were made, not *why* they were made. Importantly, if we do not know why a prediction was made, we do not know if we can trust it (Molnar, 2019).

Doshi-Velez and Kim (2017) argue that the need for interpretability arises from an incomplete problem definition: while we might optimize for accuracy, we often also care about something else, such as safety, fairness, nondiscrimination, robustness, trust, or providing the right to explanation. While each of these is hard to quantify, if our model is interpretable, if it can explain its predictions, then at least we have some means to verify that it complies with these auxiliary criteria. Even though there are cases where we do not care about interpretability, usually well-studied and well-performing ones such as optical character recognition (OCR), clearly in other cases interpretability is essential, for example when a doctor needs to make a diagnosis based on the predictions.

In this chapter we will focus on one specific definition of interpretability that is specific to our application: What if the model could provide us the most important parts of a document, as a justification for its predictions?[1] We use a setting that was pioneered by Lei et al. (2016). A *rationale* is defined to be a *short* yet *sufficient* part of the input text; short so that it makes clear what is most important, and sufficient so that a correct prediction can be made from the rationale alone. One neural network learns to extract the rationale, while another neural network, with separate parameters, learns to make a prediction from just the rationale. Figure 5.1 shows these two neural networks. Lei et al. model the rationale extraction by assigning

1 We discuss other approaches to interpretability in §5.8, as well as how our approach fits into a taxonomy of such approaches.

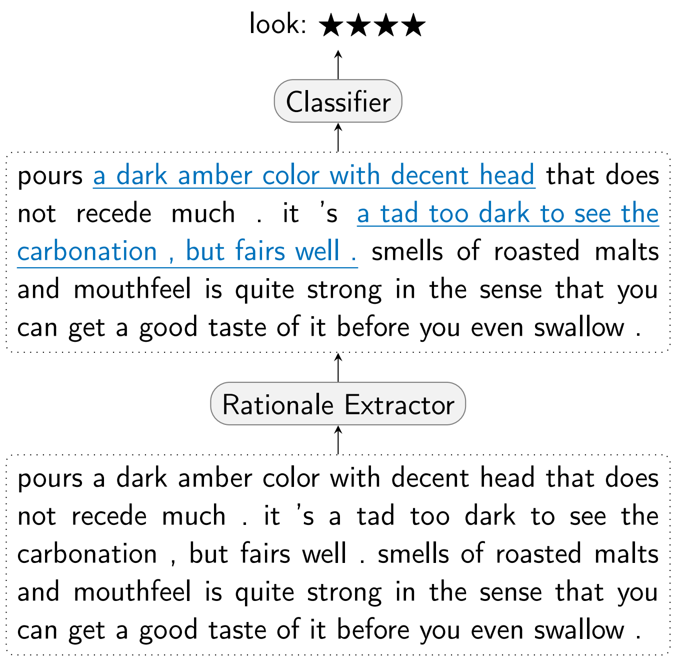

look: ★★★★
Classifier
pours a dark amber color with decent head that does not recede much . it 's a tad too dark to see the carbonation , but fairs well . smells of roasted malts and mouthfeel is quite strong in the sense that you can get a good taste of it before you even swallow .
Rationale Extractor
pours a dark amber color with decent head that does not recede much . it 's a tad too dark to see the carbonation , but fairs well . smells of roasted malts and mouthfeel is quite strong in the sense that you can get a good taste of it before you even swallow .

Figure 5.1: Rationale extraction for a beer review.

a binary Bernoulli variable to each input word. The rationale then consists of all the words for which a 1 was sampled. Because gradients do not flow through discrete samples, the rationale extractor is optimized using REINFORCE (Williams, 1992). Penalties on the number of selected words and transitions make sure the rationales are short and coherent.

In §5.2 we propose an alternative to purely discrete selectors for which gradient estimation is possible without REINFORCE. Instead, we rely on the reparameterization of a random variable that exhibits both continuous and discrete behavior using a stretch-and-rectify technique (Louizos et al., 2018). In §5.3 we use this technique to construct a new distribution, the Hard Kumaraswamy, HardKuma for short. To promote short rationales, in §5.4 we employ a relaxed form of L_0 regularization (Louizos et al., 2018), penalizing the objective as a function of the expected proportion of selected text. We also propose the use of Lagrangian relaxation in §5.5 to target a specific text selection rate. In §5.6 we provide a model for text classification, which we use in our experiments in §5.7. We discuss related work in §5.8 and conclude in §5.9.

5.2 LATENT RATIONALE

We would like to make neural text classifiers interpretable by (i) uncovering which parts of the input text contribute features for classification, and (ii) making decisions based on just a fraction of the input text (a *rationale*). Figure 5.1 illustrates our setup. Lei et al. (2016) approach (i) by inducing binary latent selectors that control which input positions are available to a neural network that learns features for classification/regression, and (ii) by regularizing their models using sparsity-inducing penalties on latent assignments. In this section we take a probabilistic view of their approach, which will then more naturally lead to the method that we propose in this chapter.

In text classification, an input sequence $x = \langle x_1, \ldots, x_n \rangle$ is mapped to a distribution over target labels:

$$Y \mid x \sim \text{Categorical}\left(f(x;\theta)\right) , \tag{5.1}$$

where a neural network $f(\cdot;\theta)$ parameterizes the distribution, with θ denoting the parameters of the neural network layers of f. In other words, neural network maps from data space (e.g. sentences, short paragraphs, or premise-hypothesis pairs) to the categorical parameter space (i.e. a vector of class probabilities). A target y is typically a categorical outcome, such as a sentiment class or an entailment decision, but with an appropriate choice of likelihood it could also be a numerical score (continuous or integer).[2]

Lei et al. (2016) augment this model with a collection of *latent variables* which we denote by $z = \langle z_1, \ldots, z_n \rangle$. These variables are responsible for regulating which portions of the input x contribute with predictors (i.e. features) to the classifier. The model formulation changes as follows:

$$\begin{aligned} Z_i \mid x &\sim \text{Bernoulli}\left(g_i(x;\phi)\right) \quad \text{for } i = 1, \ldots, |x| \\ Y \mid x, z &\sim \text{Categorical}\left(f(x \odot z;\theta)\right) \end{aligned} \tag{5.2}$$

where a neural network $g(\cdot;\phi)$ predicts a sequence of n Bernoulli parameters, one per latent variable, and the classifier is modified so that z_i indicates whether or not x_i is available for encoding. We can think of the sequence z as a binary gating mechanism used to select a rationale, which we denote by $x \odot z$.[3]

2 E.g., Gaussian for regression, Poisson for ordinal regression.

3 We use an element-wise product (also known as Hadamard product) here to select those x_i for which z_i is nonzero.

We estimate the parameters of this model by maximizing a lower bound $\mathcal{E}(\phi, \theta)$ on the log-likelihood of the data derived by application of Jensen's inequality:[4]

$$\begin{aligned} &\log P(y \mid x) = \log \mathbb{E}_{P(z|x,\phi)} [P(y \mid x, z, \theta)] \\ &\overset{\text{JI}}{\geqslant} \mathbb{E}_{P(z|x,\phi)} [\log P(y \mid x, z, \theta)] = \mathcal{E}(\phi, \theta) \,. \end{aligned} \tag{5.3}$$

These latent rationales approach the first objective, namely, uncovering which parts of the input text contribute features for a decision. However, mind that a neural network controls the Bernoulli parameters, and nothing prevents it from selecting the entire input sequence, thus defaulting to a standard text classifier. Therefore, to promote short rationales, Lei et al. (2016) impose sparsity-inducing penalties on latent selectors. They penalize for the total number of selected words, L_0 in (5.4), as well as for the total number of transitions, *fused lasso* in (5.4), and approach the following optimization problem

$$\min_{\phi,\theta} -\mathcal{E}(\phi, \theta) + \mathbb{E}\left[\lambda_0 L_0(z) + \lambda_1 \text{Lasso}(z)\right] \tag{5.4}$$

via gradient-based optimization, where λ_0 and λ_1 are small fixed hyper-parameters. This objective is intractable, and the lowerbound in particular requires marginalization of $O(2^n)$ binary sequences. For that reason, Lei et al. sample latent assignments and work with gradient estimates using REINFORCE (Williams, 1992).

To summarize: the key ingredients are binary latent variables and sparsity-inducing regularization, both resulting in non-differentiability. In this chapter we propose to replace the Bernoulli variables by rectified continuous random variables (Socci et al., 1998), as they exhibit both discrete and continuous behavior. They also allow for using the reparameterization trick (Kingma and Welling, 2014), so gradient estimation is possible without REINFORCE. Following Louizos et al. (2018), we construct one such variable and relax L_0 regularization so as to promote short rationales with a differentiable objective. In §5.3 we introduce this distribution, and in §5.4 we employ a Lagrangian relaxation to target a pre-specified text selection rate. Finally, in §5.6 we present an example for sentiment classification.

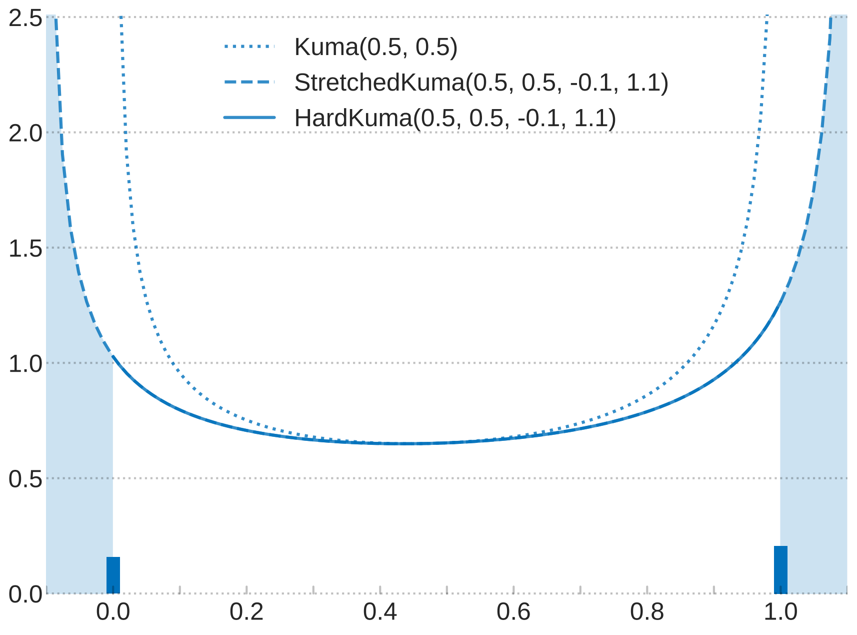


Figure 5.2: The HardKuma distribution: we start from a Kuma$(0.5, 0.5)$, and stretch its support to the interval $(-0.1, 1.1)$, finally we collapse all mass before 0 to $\{0\}$ and all mass after 1 to $\{1\}$.

5.3 HARD KUMARASWAMY DISTRIBUTION

In this section we introduce a novel distribution that exhibits both continuous and discrete behavior, and is an essential part of our model. With non-negligible probability, samples from this distribution evaluate to *exactly* 0 or *exactly* 1. We construct our distribution in three steps:

1. we start from a distribution over the open interval $(0, 1)$ (see dotted curve in Figure 5.2);
2. we *stretch* its support from $l < 0$ to $r > 1$ in order to include $\{0\}$ and $\{1\}$ (see dashed curve in Figure 5.2);
3. we collapse the probability mass over the interval $(l, 0]$ (left shaded area) to $\{0\}$, and similarly, the probability mass over the interval $[1, r)$ (right shaded area) to $\{1\}$ (see solid curve and bars in Figure 5.2).

This *stretch-and-rectify* technique was proposed by Louizos et al. (2018), who rectified samples from the BinaryConcrete (or

4 This can be seen as variational inference (Jordan et al., 1999) where we perform approximate inference using a data-dependent prior $P(z \mid x, \phi)$.

GumbelSoftmax) distribution (Jang et al., 2017; Maddison et al., 2017). Here we adapt their technique to the Kumaraswamy distribution motivated by its close resemblance to a Beta distribution, for which we have stronger intuitions. For example, its two shape parameters transit rather naturally from unimodal to bimodal configurations of the distribution. In the following, we introduce our new distribution formally.[5]

5.3.1 *Kumaraswamy distribution*

The Kumaraswamy distribution (Kumaraswamy, 1980) is a two-parameters distribution over the *open* interval $(0, 1)$. We denote a Kumaraswamy-distributed variable by $K \sim \mathrm{Kuma}(a, b)$, with $a \in \mathbb{R}_{>0}$ and $b \in \mathbb{R}_{>0}$ controlling the shape of the distribution. The dotted curve in Figure 5.2 illustrates the density of $\mathrm{Kuma}(0.5, 0.5)$. Appendix A.1 provides more details, including the pdf and cdf of the Kumaraswamy.

The Kumaraswamy is a close relative of the Beta distribution, though not itself an exponential family, with a simple cdf whose inverse

$$F_K^{-1}(u; a, b) = \left(1 - (1 - u)^{1/b}\right)^{1/a} , \tag{5.5}$$

for $u \in [0, 1]$, can be used to obtain samples

$$F_K^{-1}(U; \alpha, \beta) \sim \mathrm{Kuma}(\alpha, \beta) \tag{5.6}$$

by transformation of a uniform random source $U \sim \mathcal{U}(0, 1)$. We can use this fact to reparameterize expectations (Nalisnick and Smyth, 2016):

$$\mathbb{E}_K\left[\psi(k)\right] = \mathbb{E}_{\mathcal{U}(0,1)}\left[\psi(F_K^{-1}(u; \alpha, \beta))\right] . \tag{5.7}$$

where ψ is a function making use of the Kuma variable.

5.3.2 *Rectified Kumaraswamy*

We stretch the support of the Kumaraswamy distribution to include 0 and 1. The resulting variable $T \sim \mathrm{Kuma}(a, b, l, r)$ takes

5 We use uppercase letters for random variables (e.g. K, T, and H) and lowercase for assignments (e.g. k, t, h). For a random variable K, $f_K(k; \alpha)$ is the probability density function (pdf), conditioned on parameters α, and $F_K(k; \alpha)$ is the cumulative distribution function (cdf).

on values in the open interval (l, r) where $l < 0$ and $r > 1$, with cdf

$$F_T(t; a, b, l, r) = F_K((t-l)/(r-l); a, b)\ . \tag{5.8}$$

We now define a rectified random variable, denoted by $H \sim \text{HardKuma}(a, b, l, r)$, by passing a sample $T \sim \text{Kuma}(a, b, l, r)$ through a hard-sigmoid, i.e. $h = \min(1, \max(0, t))$. The resulting variable is defined over the *closed* interval $[0, 1]$. Although there is 0 probability of sampling $t = 0$, sampling $h = 0$ corresponds to sampling any $t \in (l, 0]$, a set whose mass under $\text{Kuma}(t|a, b, l, r)$ is available in closed form:

$$\mathbb{P}(H = 0) = F_K\left(\tfrac{-l}{r-l}; a, b\right)\ . \tag{5.9}$$

That is because all negative values of t are deterministically mapped to zero. Similarly, samples $t \in [1, r)$ are all deterministically mapped to $h = 1$, whose total mass amounts to

$$\mathbb{P}(H = 1) = 1 - F_K\left(\tfrac{1-l}{r-l}; a, b\right)\ . \tag{5.10}$$

See Figure 5.2 for an illustration. Appendix A.1 provides the complete derivations.

5.3.3 *Reparameterization and gradients*

REPARAMETERIZATION. Because we built our rectified variable on a Kumaraswamy, we can use the reparameterization trick, and construct it by transforming random noise from a uniform variable $U \sim \mathcal{U}(0, 1)$. First we sample from U, then transform the result into a Kumaraswamy variable via the inverse cdf (5.11a), then shift and scale the result to cover the stretched support (5.11b), and finally, we apply a rectifier in order to get a sample in the closed interval $[0, 1]$ (5.11c).

$$k = F_K^{-1}(u; a, b) \tag{5.11a}$$
$$t = l + (r - l)k \tag{5.11b}$$
$$h = \min(1, \max(0, t))\ , \tag{5.11c}$$

We denote this $h = s(u; a, b, l, r)$ for short.

GRADIENTS. The transformation has two discontinuity points, namely, $t = 0$ and $t = 1$. The probability of sampling $t = 0$ or $t = 1$ is zero however, so these points are not a problem for differentiability. We can consider the case where we need derivatives of a function $\mathcal{L}(u)$ of the underlying uniform variable u, as when we compute reparameterized gradients in variational inference. Using the chain rule we can write:

$$\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial h} \times \frac{\partial h}{\partial t} \times \frac{\partial t}{\partial k} \times \frac{\partial k}{\partial u} , \tag{5.12}$$

The term $\frac{\partial \mathcal{L}}{\partial h}$ depends on a differentiable observation model and poses no challenge; the term $\frac{\partial h}{\partial t}$ is the derivative of the hard-sigmoid function, which is 0 for $t < 0$ or $t > 1$, 1 for $0 < t < 1$, and undefined for $t \in \{0, 1\}$; the term $\frac{\partial t}{\partial k} = r - l$ follows directly from (A.4, left); the term $\frac{\partial k}{\partial u} = \frac{\partial}{\partial u} F_K^{-1}(u; a, b)$ depends on the Kumaraswamy inverse cdf (A.3) and also poses no challenge. Thus the only two discontinuities happen for $t \in \{0, 1\}$, which is a 0 measure set under the stretched Kumaraswamy: we say this reparameterization is differentiable *almost everywhere*, a useful property which essentially circumvents the non-differentiable points of the rectifier.

5.4 CONTROLLED SPARSITY

Following Louizos et al. (2018), we relax the non-differentiable penalties by computing them on expectation under our latent model $p(z \mid x, \phi)$. Thanks to the tractable Kumaraswamy cdf, the expected value of $L_0(z)$ is known in closed form

$$\begin{aligned} \mathbb{E}_{p(z|x)}\left[L_0(z)\right] &\overset{\text{ind}}{=} \sum_{i=1}^{n} \mathbb{E}_{p(z_i|x)}\left[\mathbb{I}[z_i \neq 0]\right] \\ &= \sum_{i=1}^{n} 1 - \mathbb{P}(Z_i = 0) , \end{aligned} \tag{5.13}$$

where $\mathbb{P}(Z_i = 0) = F_K\left(\frac{-l}{r-l}; a_i, b_i\right)$. This quantity is a tractable and differentiable function of the parameters ϕ of the latent model. We can also compute a relaxation of fused lasso by computing the expected number of zero-to-nonzero and nonzero-to-zero changes:

$$\begin{aligned} &\mathbb{E}_{p(z|x)}\left[\sum_{i=1}^{n-1} \mathbb{I}[z_i{=}0, z_{i+1}{\neq}0]\right] + \mathbb{E}_{p(z|x)}\left[\sum_{i=1}^{n-1} \mathbb{I}[z_i{\neq}0, z_{i+1}{=}0]\right] \overset{\text{ind}}{=} \\ &\sum_{i=1}^{n-1} \mathbb{P}(Z_i{=}0)(1 - \mathbb{P}(Z_{i+1}{=}0)) + (1 - \mathbb{P}(Z_i{=}0))\mathbb{P}(Z_{i+1}{=}0) \end{aligned} \tag{5.14}$$

In both cases, we make use of the assumption that latent variables are independent given x. In Appendix A.2.1.2 we discuss how to estimate the regularizers for a model $p(z_i \mid x, z_{<i})$ that conditions on the prefix $z_{<i}$ of sampled HardKuma assignments.

5.5 LAGRANGIAN RELAXATION

We can use the regularizers to promote sparsity, but just how much text will our final model select? Ideally, we target specific values r and solve a constrained optimization problem. In practice, constrained optimization is very challenging. Since we derived differentiable sparsity-inducing regularizers, we employ Lagrangian relaxation instead (Boyd et al., 2004), and approach an unconstrained problem:

$$\max_{\lambda \in \mathbb{R}^+} \min_{\phi, \theta} -\mathcal{E}(\phi, \theta) + \lambda^\top (R(\phi) - r) \tag{5.15}$$

where $R(\phi)$ is a vector of regularizers, e.g. expected L_0 and expected fused lasso, and λ is a vector of Lagrangian multipliers λ. During training we update λ according to the update scheme used in Rezende and Viola (2018). Note how this approach differs from the treatment of Lei et al. (2016) shown in (5.4) where regularizers are computed for assignments, rather than on expectation, and where λ_0, λ_1 are fixed hyper-parameters.

5.6 SENTIMENT CLASSIFICATION

As a concrete example, consider the case of sentiment classification where x is a sentence and y is a 5-way sentiment class (from very negative to very positive). The model consists of

$$\begin{aligned} Z_i &\sim \text{HardKuma}(a_i, b_i, l, r) \quad \text{for } i = 1, \ldots, |x| \\ Y \mid x, z &\sim \text{Categorical}(f(x \odot z; \theta)) \end{aligned} \tag{5.16}$$

where the shape parameters $a, b = g(x; \phi)$, i.e. two sequences of n strictly positive scalars, are predicted by a neural net, and the support boundaries (l, r) are fixed hyper-parameters.

We first specify an architecture that parameterizes latent selectors and then use a reparameterized sample to restrict which parts of the input contribute encodings for classification:[6]

$$\begin{aligned} \mathbf{e}_i &= \mathrm{emb}(x_i) & a_i &= f_a(\mathbf{h}_i; \phi_a) \\ \mathbf{h}_1^n &= \mathrm{birnn}(\mathbf{e}_1^n; \phi_r) & b_i &= f_b(\mathbf{h}_i; \phi_b) \\ u_i &\sim \mathcal{U}(0, 1) & z_i &= s(u_i; a_i, b_i, l, r) \end{aligned} \tag{5.17}$$

where $\mathrm{emb}(\cdot)$ is an embedding layer, $\mathrm{birnn}(\cdot; \phi_r)$ is a bidirectional encoder, $f_a(\cdot; \phi_a)$ and $f_b(\cdot; \phi_b)$ are feed-forward transformations with softplus outputs, and $s(\cdot)$ turns the uniform sample u_i into the latent selector z_i (see §5.3). We then use the sampled z to modulate inputs to the classifier:

$$\begin{aligned} \mathbf{e}_i &= \mathrm{emb}(x_i) \\ \mathbf{h}_i^{(\mathrm{fwd})} &= \mathrm{rnn}(\mathbf{h}_{i-1}^{(\mathrm{fwd})}, z_i\, \mathbf{e}_i; \theta_{\mathrm{fwd}}) \\ \mathbf{h}_i^{(\mathrm{bwd})} &= \mathrm{rnn}(\mathbf{h}_{i+1}^{(\mathrm{bwd})}, z_i\, \mathbf{e}_i; \theta_{\mathrm{bwd}}) \\ \mathbf{o} &= f_o(\mathbf{h}_n^{(\mathrm{fwd})}, \mathbf{h}_1^{(\mathrm{bwd})}; \theta_o) \end{aligned} \tag{5.18}$$

where $\mathrm{rnn}(\cdot; \theta_{\mathrm{fwd}})$ and $\mathrm{rnn}(\cdot; \theta_{\mathrm{bwd}})$ are recurrent cells such as LSTMs (Hochreiter and Schmidhuber, 1997) that process the sequence in different directions, and $f_o(\cdot; \theta_o)$ is a feed-forward transformation with softmax output. Note how z_i modulates features $\mathbf{e}_i$ of the input x_i that are available to the recurrent composition function.

We then obtain gradient estimates of $\mathcal{E}(\phi, \theta)$ via Monte Carlo (MC) sampling from

$$\mathcal{E}(\phi, \theta) = \mathbb{E}_{\mathcal{U}(0,I)}\left[\log P(y|x, s_\phi(u, x), \theta)\right] \tag{5.19}$$

where $z = s_\phi(u, x)$ is a shorthand for element-wise application of the transformation from uniform samples to HardKuma samples. This reparameterization is the key to gradient estimation through stochastic computation graphs (Kingma and Welling, 2014; Rezende et al., 2014).

DETERMINISTIC PREDICTIONS. At test time we make predictions based on what is the most likely assignment for each z_i. We arg max across configurations of the distribution, namely, $z_i = 0$, $z_i = 1$, or $0 < z_i < 1$. When the continuous interval is more likely, we take the expected value of the underlying Kumaraswamy variable.

6 We describe architectures using blocks denoted by $\mathrm{layer(inputs; subset\ of\ parameters)}$, boldface letters for vectors, and the shorthand $\mathbf{v}_1^n$ for a sequence $\langle \mathbf{v}_1, \ldots, \mathbf{v}_n \rangle$.

SVM (Lei et al., 2016)	0.0154
BiLSTM (Lei et al., 2016)	0.0094
BiRCNN (Lei et al., 2016)	0.0087
BiLSTM	0.0089
BiRCNN	0.0088

Table 5.1: MSE on the BeerAdvocate test set.

5.7 EXPERIMENTS

We perform experiments on multi-aspect sentiment analysis to compare with previous work, as well as experiments on sentiment classification and natural language inference. All models were implemented in PyTorch[7]. Appendix A.2 provides implementation details.

GOAL. When rationalizing predictions, our goal is to perform as well as systems using the full input text, while using only a *subset* of the input text, leaving unnecessary words out for interpretability. Since the classifier only sees the rationales, and can learn only from those and not the full input text, it is a challenge (but not impossible) to do as well as the full-text case.

5.7.1 *Multi-aspect Sentiment Analysis*

In our first experiment we compare directly with previous work on rationalizing predictions (Lei et al., 2016). We replicate their setting for multi-aspect sentiment analysis.

DATA. A pre-processed subset of the BeerAdvocate[8] data set is used (McAuley et al., 2012). It consists of 220,000 *beer reviews*, where multiple aspects (e.g. look, smell, taste) are rated. As shown in Figure 5.1, a review typically consists of multiple sentences, and contains a 0-5 star rating (e.g. 3.5 stars) for each aspect. Lei et al. mapped the ratings to scalars in $[0, 1]$.

MODEL. We use the models described in §5.6 with two small modifications: 1) since this is a regression task, we use a sig-

7 https://pytorch.org/
8 https://www.beeradvocate.com/

moid activation in the output layer of the classifier rather than a softmax,[9] and 2) we use an extra RNN to condition z_i on $z_{<i}$:

$$\begin{aligned} a_i &= f_a(\mathbf{h}_i, \mathbf{s}_{i-1}; \phi_a) && (5.20a) \\ b_i &= f_b(\mathbf{h}_i, \mathbf{s}_{i-1}; \phi_b) && (5.20b) \\ \mathbf{s}_i &= \text{rnn}(\mathbf{h}_i, z_i, \mathbf{s}_{i-1}; \phi_s) && (5.20c) \end{aligned}$$

For a fair comparison we follow Lei et al. by using RCNN[10] cells rather than LSTM cells for encoding sentences on this task. Since this cell is not widely used, we verified its performance in Table 5.1. We observe that the BiRCNN performs on par with the BiLSTM (while using 50% fewer parameters), and similarly to previous results.

EVALUATION. A test set with sentence-level rationale annotations is available. The *precision* of a rationale is defined as the percentage of words with $z \neq 0$ that is part of the annotation. We also evaluate the predictions made from the rationale using Mean Squared Error (MSE).

BASELINES. For our baseline we reimplemented the approach of Lei et al. (2016) which we call *Bernoulli* after the distribution they use to sample z from. We also report their attention baseline, in which an attention score is computed for each word, after which it is simply thresholded to select the top-k percent as the rationale.

RESULTS. Table 5.2 shows the precision and the percentage of selected words for the first three aspects. The models here have been selected based on validation MSE and were tuned to select a similar percentage of words ('Sel.'). We observe that our Bernoulli reimplementation reaches a precision ('Prec.') similar to previous work, doing a little bit worse for the 'look' aspect. Our HardKuma managed to get even higher precision, and it extracted exactly the percentage of text that we specified (see §5.5).[11] Figure 5.3 shows the MSE for all aspects for various percentages of extracted text. We observe that HardKuma does

9 From a likelihood learning point of view, we would have assumed a Logit-Normal likelihood, however, to stay closer to Lei et al. (2016), we employ mean squared error.

10 An RCNN cell can replace any LSTM cell and works well on text classification problems. See appendix A.2.

11 We tried to use Lagrangian relaxation for the Bernoulli model, but this led to instabilities (e.g. all words selected).

Method	Look		Smell		Taste	
	Prec.	Sel.	Prec.	Sel.	Prec.	Sel.
Attention (Lei et al.)	80.6	13	88.4	7	65.3	7
Bernoulli (Lei et al.)	96.3	14	95.1	7	80.2	7
Bernoulli *(reimpl.)*	94.8	13	95.1	7	80.5	7
HardKuma	98.1	13	96.8	7	89.8	7

Table 5.2: Precision (% of selected words that was also annotated as the gold rationale) and selected (% of words not zeroed out) per aspect. In the attention baseline, the top 13% (7%) of words with highest attention weights are used for classification. Models were selected based on validation loss.

better with a smaller percentage of text selected. The performance becomes more similar as more text is selected.

5.7.2 *Sentiment Classification*

We also experiment on the Stanford Sentiment Treebank (SST) (Socher et al., 2013). There are 5 sentiment classes: very negative, negative, neutral, positive, and very positive. Here we use the HardKuma model described in §5.6, a Bernoulli model trained with REINFORCE, as well as a BiLSTM.

RESULTS. Figure 5.4a shows the classification accuracy for various percentages of selected text. We observe that HardKuma outperforms the Bernoulli model at each percentage of selected text. HardKuma reaches full-text baseline performance already around 40% extracted text. At that point, it obtains a *test* score of 45.84, versus 42.22 for Bernoulli and 47.4±0.8 for the full-text baseline.

ANALYSIS. We wonder what kind of words are dropped when we select smaller amounts of text. For this analysis we exploit the word-level sentiment annotations in SST, which allows us to track the sentiment of words in the rationale. Figure 5.4b shows that a large portion of dropped words have neutral sentiment, and it seems plausible that exactly those words are not important features for classification. We also see that HardKuma drops (relatively) more neutral words than Bernoulli.

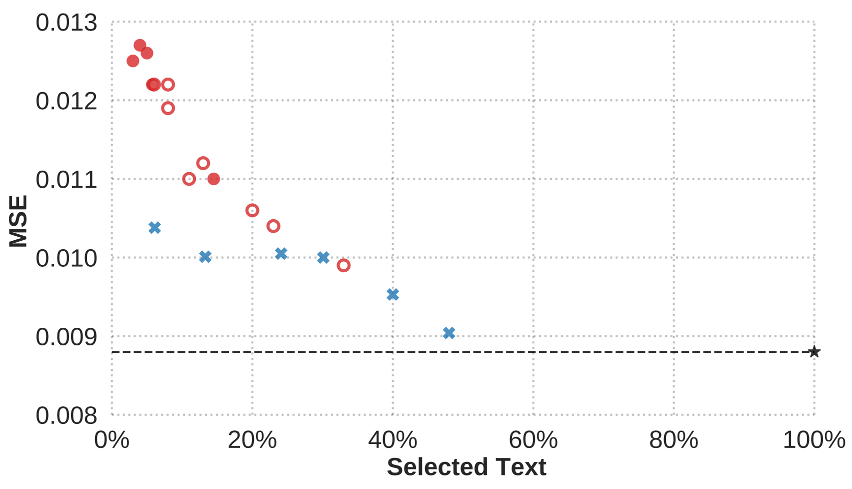


Figure 5.3: MSE of all aspects for various percentages of extracted text. HardKuma (blue crosses) has lower error than Bernoulli (red circles; open circles taken from Lei et al. (2016)) for similar amount of extracted text. The full-text baseline (black star/dashed line) gets the best MSE.

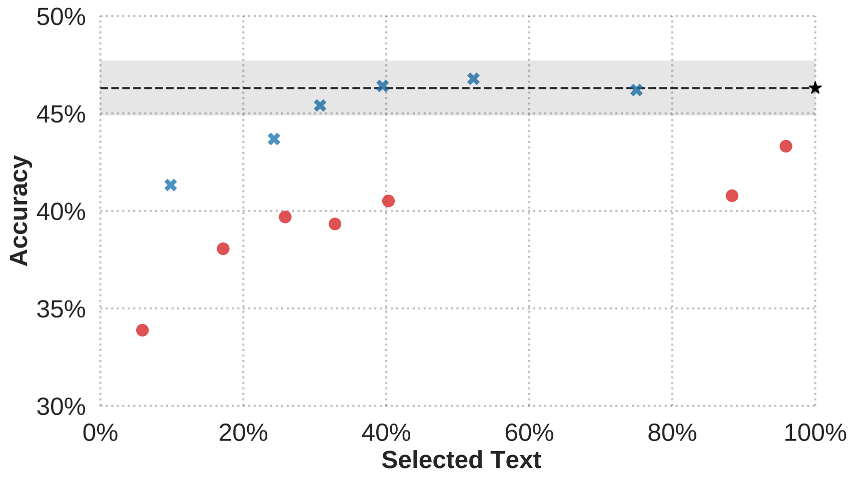
5.7.3 *Natural Language Inference*

In Natural language inference (NLI), given a premise sentence $x^{(p)}$ and a hypothesis sentence $x^{(h)}$, the goal is to predict their relation y which can be contradiction, entailment, or neutral. As our dataset we use the Stanford Natural Language Inference (SNLI) corpus (Bowman et al., 2015).

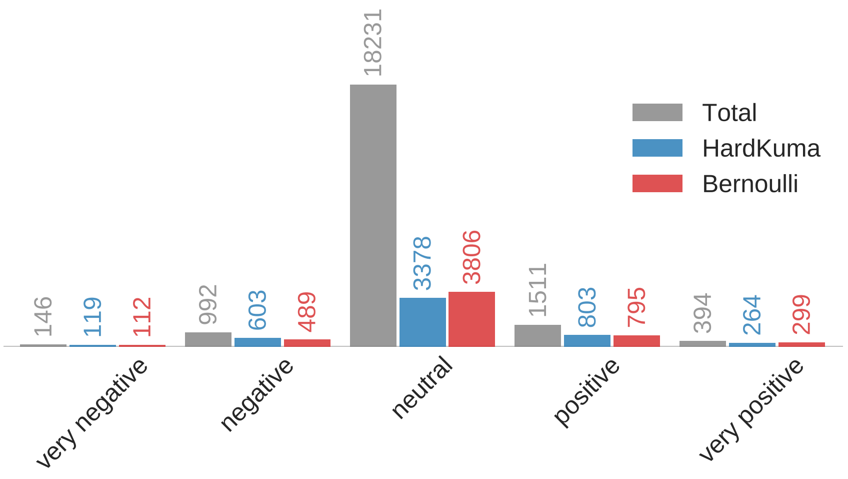
BASELINE. We use the Decomposable Attention model (DA) of Parikh et al. (2016).[12] DA does not make use of LSTMs, but rather uses attention to find connections between the premise and the hypothesis that are predictive of the relation. Each word in the premise attends to each word in the hypothesis, and vice versa, resulting in a set of comparison vectors which are then aggregated for a final prediction. If there is no link between a word pair, it is not considered for prediction.

MODEL. Because the premise and hypothesis interact, it does not make sense to extract a rationale for the premise and hypothesis independently. Instead, we replace the attention between premise and hypothesis with HardKuma attention. While in the baseline a similarity matrix is softmax-normalized across

12 Better results e.g. Chen et al. (2017) and data sets for NLI exist, but are not the focus of this paper.



(a) SST validation accuracy for various percentages of extracted text. HardKuma (blue crosses) has higher accuracy than Bernoulli (red circles) for similar amount of text, and reaches the full-text baseline (black star/-dashed line, $46.3 \pm 2\sigma$ with $\sigma = 0.7$) around 40% text.



(b) The number of words in each sentiment class for the full validation set, the HardKuma (24% selected text) and Bernoulli (25% text).

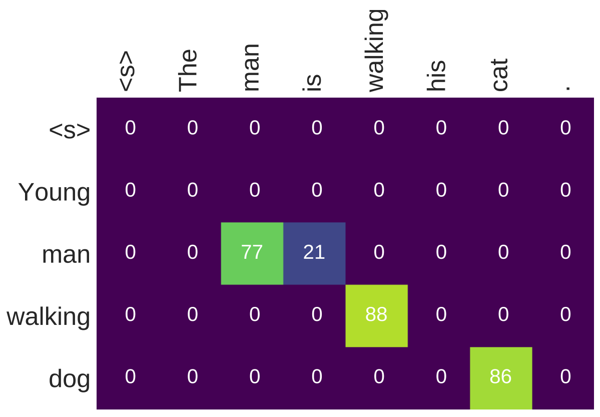
Figure 5.4: SST results.

Model	Dev	Test
LSTM (Bowman et al., 2016a)	–	80.6
DA (Parikh et al., 2016)	–	86.3
DA *(reimplementation)*	86.9	86.5
DA with HardKuma attention	86.0	85.5

Table 5.3: SNLI results (accuracy).

rows (premise to hypothesis) and columns (hypothesis to premise) to produce attention matrices, in our model each cell in the attention matrix is sampled from a HardKuma parameterized by (a, b). To promote sparsity, we use the relaxed L_0 to specify the desired percentage of non-zero attention cells. The resulting matrix does not need further normalization.

RESULTS. With a target rate of 10%, the HardKuma model achieved 8.5% non-zero attention. Table 5.3 shows that, even with so many zeros in the attention matrices, it only does about 1% worse compared to the DA baseline. Figure 5.5 shows an example of HardKuma attention, with additional examples in Appendix A.2. We leave further explorations with HardKuma attention for future work.



	<s>	The	man	is	walking	his	cat	.
<s>	0	0	0	0	0	0	0	0
Young	0	0	0	0	0	0	0	0
man	0	0	77	21	0	0	0	0
walking	0	0	0	0	88	0	0	0
dog	0	0	0	0	0	0	86	0

Figure 5.5: Example of HardKuma attention between a premise (rows) and hypothesis (columns) in SNLI (cell values shown in multiples of 10^{-2}).

5.8 RELATED WORK

This chapter has connections with work on interpretability, learning from rationales, sparse structures, and rectified distributions. We discuss each of those areas.

5.8.1 *Interpretability*

Machine learning research has been focusing more and more on interpretability (Gilpin et al., 2018). However, there are many nuances to it (Lipton, 2016), such as different definitions and ways of evaluating when it is achieved. After defining a taxonomy of interpretability methods, we place our work in the context of the most relevant other approaches, in particular those that are applicable to text.

TAXONOMY. We can make a taxonomy of approaches to interpretability according to the following criteria (Molnar, 2019):

- Intrinsic/Post-hoc: is the method intrinsically interpretable, or do we try to interpret a model after it has been trained?
- Result of the method: what does the method produce? E.g. feature importance values, learned weights, an interpretable approximation, or data points.
- Model specific/model agnostic? Can the method be applied to any machine learning model, or only to a specific subset?
- Local/Global: does the method explain one single prediction, or does it globally explain the model?

We could say that our method, as well as Lei et al. (2016), is *intrinsically interpretable*, since the model learns to provide rationales during training, even though neural networks themselves are not intrinsically interpretable. The *result* is highlighted text, or per-word feature importance, for each document. The method is *model specific*, since we built it for text classification, although it could be applied to other tasks. Finally, our method highlights what is important for individual examples (the rationale), so in that sense it provides local explanations. Note, however, that the mechanism to provide rationales was trained globally on the whole dataset, and unlike LIME (see below) was not approximated to explain just this single example.

We will now describe other methods for interpretability along the first axis: intrinsic or post-hoc.

INTRINSICALLY INTERPRETABLE MODELS. One strategy is to extract a simpler, interpretable model from a neural network, though this comes at the cost of performance. Popular options are linear and logistic regression, decision rules, and decision trees. For example, Thrun (1995) extract if-then rules from another model, while Craven and Shavlik (1996) extract decision trees. Note however that, while it is clear how a decision tree arrives at a decision, if it contains a large amount of nodes its interpretability can be questioned. Friedman and Popescu (2008) proposed RuleFit, which learns an interpretable model that, unlike linear models, also detects feature interactions.

POST-HOC EXPLANATIONS. Another strategy is to try and interpret an already trained model, in contrast to Lei et al. (2016) and our approach where the rationale is jointly modeled in the original model. The advantage of such an approach is that we can use lots of already trained models, or can focus on training one to the highest performance, and then worry about interpretability afterwards. Ribeiro et al. (2016) make any classifier interpretable by approximating it locally with a linear proxy model in an approach called LIME. LIME works by (1) perturbing the input text by randomly dropping (or masking) input words, (2) passing these perturbations through the model to get predictions, and (3) fitting a linear model to the set of perturbations and predicted labels. The coefficients of the linear model are now used as feature importance and serve as the explanation. LIME relies on a few hyper-parameters, such as kernel width, and while its advantage is that it can explain any black-box model, its disadvantages are that setting the hyper-parameters differently might result in different explanations, and that it is relatively expensive to generate explanations, because many (by default, 5000) perturbations need to be passed through the model. The fact that (the original version of) LIME drops input words, breaking sentence structure, led to another method called LIMSSE (Poerner et al., 2018). In contrast to LIME, LIMSSE generates perturbations by sampling n-grams (of size 1-6) from the original input, which at least keep structure intact locally, but can still result in unnatural inputs. Alvarez-Melis and Jaakkola (2017) propose a framework that returns input-output pairs that are causally related, and similar to LIME they achieve this by perturbing inputs, albeit in a more sophisticated way by sampling from a variational auto-encoder (Bowman et al., 2016b). In contrast to LIME, their result is not a simpler local approximation, but a summary of operation. The above methods are related to Li et al. (2016a), who investigated how representation erasure (e.g., removing a word type from the test set) influences the predictions. Removing a word type from the test set throughout results in a more global importance score for that word, rather than one that approximates its importance on a single example.

Shapley values, originally from Shapley (1953) and used for explanation by Štrumbelj and Kononenko (2014), are a method from game theory that allow us to fairly (i.e., taking into account coalitions of features) distribute the prediction among the input features, resulting in feature importance scores. Lundberg and Lee (2017) combine LIME and Shapley values in a method called SHapley Additive exPlanations (SHAP). They also compute Shapley values for a single data example, but the result of the method is a linear model explaining that example just like LIME. The example perturbations are weighted by a SHAP kernel, and performing linear regression with that kernel results in Shapley values.

5.8.2 *Learning from rationales*

Our work is different from approaches that aim to improve classification using rationales as an additional input (Zaidan and Eisner, 2008; Zaidan et al., 2007; Zhang et al., 2016). Instead, our rationales are latent and we are interested in uncovering them. We only use annotated rationales for evaluation.

5.8.3 *Sparse Representations*

There is also work on making word vectors more interpretable. Faruqui et al. (2015) make word vectors more sparse, and Herbelot and Vecchi (2015) learn to map distributional word vectors to model-theoretic semantic vectors.

5.8.4 *Opinion summarization*

Similarly to Lei et al. (2016), Titov and McDonald (2008) extract informative fragments of text by jointly training a classifier and a model predicting a stochastic mask, while relying on Gibbs sampling to do so. Their focus is on using the sentiment labels as a weak supervision signal for opinion summarization rather than on rationalizing classifier predictions.

5.8.5 *Sparse layers*

Also arguing for enhanced interpretability, Niculae and Blondel (2017) propose a framework for learning sparsely activated attention layers based on smoothing the max operator. They derive a number of relaxations to max, including softmax itself, but in particular, they target relaxations such as sparsemax (Martins and Astudillo, 2016) which, unlike softmax, are sparse (i.e. produce vectors of probability values with components that evaluate to exactly 0). Their activation functions are themselves solutions to convex optimization problems, to which they provide efficient forward and backward passes. The technique can be seen as a deterministic sparsely activated layer which they use as a drop-in replacement to standard attention mechanisms. In contrast, in this paper we focus on binary outcomes rather than K-valued ones. Niculae et al. (2018) extend the framework to structured discrete spaces where they learn sparse parameterizations of discrete latent models. In this context, parameter estimation requires exact marginalization of discrete variables or gradient estimation via REINFORCE. They show that distributions are often sparse enough for exact marginal inference.

Peng et al. (2018) propose SPIGOT, a proxy gradient to the non-differentiable $\arg\max$ operator. This proxy requires an $\arg\max$ solver (e.g. Viterbi for structured prediction) and, like the straight-through estimator (Bengio et al., 2013), is a biased estimator. Though, unlike ST it is efficient for structured variables. In contrast, in this work we chose to focus on unbiased estimators.

5.8.6 *Rectified Distributions.*

The idea of rectified distributions has been around for some time. The rectified Gaussian distribution (Socci et al., 1998), in particular, has found applications to factor analysis (Harva and Kaban, 2005) and approximate inference in graphical models (Winn and Bishop, 2005). Louizos et al. (2018) propose to stretch and rectify samples from the BinaryConcrete (or GumbelSoftmax) distribution (Jang et al., 2017; Maddison et al., 2017). They use rectified variables to induce sparsity in parameter space via a relaxation to L_0. We adapt their technique to promote sparse *activations* instead. Rolfe (2017) learns a relaxation of a discrete random variable based on a tractable mixture of a point mass at zero and a continuous reparameterizable density, thus enabling reparameterized sampling from the half-closed interval $[0, \infty)$. In contrast, with HardKuma we focused on giving support to both 0s and 1s.

5.8.7 *Rationalizing Predictions*

In this chapter we used the framework of Lei et al. (2016) where one component generates a (binary) mask over the inputs (the rationale), and the other component learns to predict by seeing only the input features (tokens) not masked out. The result of methods like ours and Lei et al. are models that are transparent about the input features that they use, rendering them more interpretable. In a follow-up work, Yu et al. (2019) view this setup as a cooperative game between the two components, and make the component that generates the mask sensitive to the prediction outcome, and use an adversary to make sure useful information is not left out of the rationale. However, they do not compare to our work.

Very recently, a new benchmark ERASER was proposed by DeYoung et al. (2019) that can be used to evaluate rationales from models like ours. It is in line with a recent call by Jacovi and Goldberg (2020) for a more nuanced evaluation of explanations, and faithfulness in particular. ERASER can be used to evaluate both 'plausibility' (computing the overlap with human annotations) and 'faithfulness' (how much the rationale influences the prediction). Faithfulness is split into *comprehensiveness* (i.e., were all features that are required for a prediction part of the rationale?), and *sufficiency* (i.e., is there enough signal in the rationale to obtain a prediction?).

5.9 CONCLUSIONS

In this chapter we presented a differentiable approach to extractive rationales, including an objective that allows for specifying how much text is to be extracted. To allow for reparameterized gradient estimates and support for binary outcomes we introduced the HardKuma distribution. Apart from extracting rationales, we showed that HardKuma has further potential future uses, which we demonstrated on premise-hypothesis attention in SNLI.

One advantage of our approach, and that of Lei et al. (2016), is that the provided rationales are designed to be faithful and trained to be as sufficient as possible given the constraints (e.g., how much text is to be selected). However, it must be noted that there is a possibility that the rationale extractor module can make decisions in its own. For example, it could communicate to the generator by always leaving out a certain word for a certain class. While we did not observe this behavior, it is difficult to rule out.

Very recently the ERASER benchmark (DeYoung et al., 2019) was proposed. It will be interesting to evaluate approaches like ours more thoroughly, and to investigate the relation between the constraints (e.g., percentage of selected text) and measures such as sufficiency and comprehensiveness.

6 TESTING FOR STRONG GENERALIZATION

If you were told what *to dax* means, then could you *dax twice*? Chances are that you could, for example if *to dax* means doing a crazy dance. We – humans – can generalize patterns after seeing only a few examples. We know what it means to do something *twice*. It is not at all given that neural networks share this generalization capability; they tend to require lots of training examples to learn. In this chapter we study how the generalization capabilities of neural networks can be tested.

CHAPTER HIGHLIGHTS

Problem Statement

- Lake and Baroni (2018) proposed SCAN to test strong generalization of neural networks, but it is not clear if doing well on SCAN implies strong generalization.

Research Question

- Can we replicate and corroborate the results by Lake and Baroni (2018) using attention and early stopping?
- Is SCAN a good enough test for strong generalization in neural sequence-to-sequence models, and if not, how do we improve it?

Research Contributions

- We found that SCAN does not always test for strong generalization, since it lacks target-side dependencies.
- To remedy this we proposed NACS, which requires target-side dependencies to be captured.
- Results on generalization tests show that strong generalization remains an open problem.

jump
`JUMP`

turn around left
`LTURN LTURN LTURN LTURN`

jump thrice and turn left twice
`JUMP JUMP JUMP LTURN LTURN`

jump opposite left after walk twice
`WALK WALK LTURN LTURN JUMP`

walk opposite left twice and run
`LTURN LTURN WALK LTURN LTURN WALK RUN`

Figure 6.1: SCAN maps commands to actions.

6.1 INTRODUCTION

Can recurrent sequence-to-sequence models exhibit the same strong generalization that humans are capable of, by virtue of our capacity to infer the meaning of a phrase from its constituent parts (i.e., compositionality)? This is what Lake and Baroni (2018) investigate, providing empirical tests for this long-standing goal (Fodor and Pylyshyn, 1988). Compositional generalization might be a fundamental component in making models drastically less sample-thirsty than they currently are. Lake and Baroni introduce the SCAN data set (§6.2), meant to study such generalization to novel examples. It consists of simple command-action pairs, in which more complex commands are composed of simpler ones. Figure 6.1 shows a few examples.

SCAN consists of three kinds of generalization tests, namely with respect to:

1. a random subset of the data ('simple');
2. commands with action sequences longer than those seen during training ('length');
3. commands that compose a primitive in novel ways that was only seen in isolation during training ('primitive').

In the latter case, the training set would for example only include the command 'jump' in isolation (not composed with other words or phrases), after which the test set includes all other commands containing 'jump', e.g. 'jump opposite left after walk twice'.

In this chapter we take a closer look at SCAN. We start with the observation (§6.3) that the target-side dependencies in the data are simple, a consequence of SCAN being generated from a phrase-structure grammar. We will see (§6.6) that this favors simple sequence-to-sequence models (§6.5) to obtain good accuracies (even models that do not have access to previous outputs), because they have the right inductive bias. However, these simple models are not interesting, do not use composition in any interesting way, and their performance is therefore not a realistic indicator of their generalization capability. We hence propose NACS (§6.4) as a more realistic alternative: SCAN with commands and actions flipped, i.e., mapping actions back to their original commands. This is harder, because different commands may map to the same action sequence, and it introduces target-side dependencies, so that previous outputs need to be remembered.

We will see that well-tuned attention-based models do achieve a certain degree of generalization on SCAN, and, as hypothesized, simpler models do better there. However, the models still struggle in the more demanding NACS setup, which is offered as a challenge for future work.

6.2 SCAN

SCAN stands for Simplified version of the CommAI Navigation tasks (Mikolov et al., 2016). Each example in SCAN is constructed by first sampling a command $X = (x_1, \ldots, x_T)$ from a finite phrase-structure grammar with start symbol C:

$$
\begin{aligned}
&\mathrm{C} \rightarrow \mathrm{S \ and \ S} \mid \mathrm{S \ after \ S} \mid \mathrm{S}\\
&\mathrm{S} \rightarrow \mathrm{V \ twice} \mid \mathrm{V \ thrice} \mid \mathrm{V}\\
&\mathrm{V} \rightarrow \mathrm{D}_{[1]} \ \mathrm{opposite} \ \mathrm{D}_{[2]} \mid \mathrm{D}_{[1]} \ \mathrm{around} \ \mathrm{D}_{[2]} \mid \mathrm{D} \mid \mathrm{U}\\
&\mathrm{D} \rightarrow \mathrm{U \ left} \mid \mathrm{U \ right} \mid \mathrm{turn \ left} \mid \mathrm{turn \ right}\\
&\mathrm{U} \rightarrow \mathrm{walk} \mid \mathrm{look} \mid \mathrm{run} \mid \mathrm{jump}
\end{aligned}
$$

For each command, the corresponding target action sequence $Y = (y_1, \ldots, y_{T'})$ then follows by applying a set of *interpretation functions*, such as

$$
\begin{aligned}
[\![\mathrm{jump}]\!] &= \mathrm{JUMP}\\
[\![\mathrm{u \ around \ left}]\!] &= \mathrm{LTURN} \ [\![\mathrm{u}]\!] \ \mathrm{LTURN} \ [\![\mathrm{u}]\!]\\
& \ \mathrm{LTURN} \ [\![\mathrm{u}]\!] \ \mathrm{LTURN} \ [\![\mathrm{u}]\!]\\
[\![x_1 \ \mathrm{after} \ x_2]\!] &= [\![x_2]\!] \ [\![x_1]\!]
\end{aligned}
$$

of which only the last function requires global reordering, which occurs at most once per command. Figure 6.1 shows examples of commands and their action sequences as obtained by the interpretation functions. The commands can be decoded compositionally by a learner by discovering the interpretation functions, enabling generalization to unseen commands. The total data set is finite but large: it contains 20910 unambiguous commands. Figure 6.2 lists the full set of interpretation functions.

6.3 SCAN ALLOWS INADEQUATE MODELS TO PERFORM WELL

We observe an important property of the data set generation process for SCAN: the temporal dependencies of the action sequence are limited to the phrasal boundaries of each sub-phrase, and each sub-phrase spans at most 24 actions (in the case of "jump around left thrice" which causes three repetitions of LTURN JUMP LTURN JUMP LTURN JUMP LTURN JUMP). Crucially, even rules that require repetition (such as 'thrice'), as well as global reordering, can be resolved by simple counting and without remembering previously generated outputs, due to the limited depth of the phrase-structure grammar (see e.g. Rodriguez and Wiles, 1998).

This observation has two important implications:

1. Because SCAN is largely a phrase-to-phrase transduction problem, any machine learning method that aims at solving SCAN needs to include an *alignment mechanism* between the source and target sequences. Such an alignment mechanism could work fairly accurately by simply advancing a pointer. Somewhat contrary to the observation by Lake and Baroni (L&B), we therefore hypothesize that an attention mechanism (Bahdanau et al., 2015) always helps when a neural conditional sequence model (Cho et al., 2014; Sutskever et al., 2014) is used to tackle any variant of SCAN.

2. We speculate that any algorithm with strong long-term dependency modeling capabilities can be *detrimental* in terms of generalization, because such an approach might inappropriately capture spurious target-side regularities in the training data. We thus hypothesize that *less powerful decoders generalize better* on to unseen action combinations on SCAN when equipped with an attention mechanism.

$[\![\text{walk}]\!] = \text{WALK}$
$[\![\text{look}]\!] = \text{LOOK}$
$[\![\text{run}]\!] = \text{RUN}$
$[\![\text{jump}]\!] = \text{JUMP}$

$[\![\text{turn left}]\!] = \text{LTURN}$
$[\![\text{turn right}]\!] = \text{RTURN}$

$[\![u \text{ left}]\!] = \text{LTURN } [\![u]\!]$
$[\![u \text{ right}]\!] = \text{RTURN } [\![u]\!]$

$[\![x \text{ twice}]\!] = [\![x]\!]\ [\![x]\!]$
$[\![x \text{ thrice}]\!] = [\![x]\!]\ [\![x]\!]\ [\![x]\!]$

$[\![\text{turn around left}]\!] = \text{LTURN LTURN LTURN LTURN}$
$[\![\text{turn around right}]\!] = \text{RTURN RTURN RTURN RTURN}$
$[\![u \text{ around left}]\!] = \text{LTURN } [\![u]\!] \text{ LTURN } [\![u]\!] \text{ LTURN } [\![u]\!] \text{ LTURN } [\![u]\!]$
$[\![u \text{ around right}]\!] = \text{RTURN } [\![u]\!] \text{ RTURN } [\![u]\!] \text{ RTURN } [\![u]\!] \text{ RTURN } [\![u]\!]$

$[\![\text{turn opposite left}]\!] = \text{LTURN LTURN}$
$[\![\text{turn opposite right}]\!] = \text{RTURN RTURN}$
$[\![u \text{ opposite left}]\!] = [\![\text{turn opposite left}]\!]\ [\![u]\!]$
$[\![u \text{ opposite right}]\!] = [\![\text{turn opposite right}]\!]\ [\![u]\!]$
$[\![x_1 \text{ and } x_2]\!] = [\![x_1]\!]\ [\![x_2]\!]$
$[\![x_1 \text{ after } x_2]\!] = [\![x_2]\!]\ [\![x_1]\!]$

Figure 6.2: The interpretation functions for translating SCAN commands to actions.

To summarize: good performance on SCAN does *not necessarily indicate* the capability of a model to strongly generalize. SCAN favors simpler models that need not (or do not) capture target-side temporal dependencies, which might not work well on more realistic sequence-to-sequence problems, such as machine translation, where strong auto-regressive models are needed for good results (Bahdanau et al., 2015; Kaiser and Bengio, 2016).

6.4 NACS: ACTIONS TO COMMANDS

By simply flipping the source x_1^m and target y_1^n of each example, we obtain a data-set that suddenly features strong target-side dependencies. Even when the mapping $p(y \mid x)$ from the source to target is simple, the opposite $p(x \mid y) \propto p(y \mid x)\, p(x)$ is non-trivial due to the complexity of the prior $p(x)$. The inclusion of $p(x)$ naturally induces strong dependencies among the output tokens, while maintaining the original properties of SCAN that were intended to test various aspects of systematic generalization.

NACS naturally makes the mapping that needs to be learned stochastic and multi-modal (sensitive to both commands and actions). For instance, an action sequence of the form $[\![x_1]\!][\![x_2]\!]$ could be mapped to either $[\![x_1 \text{ and } x_2]\!]$ or $[\![x_2 \text{ after } x_1]\!]$, both of which are correct. In order for a model to decide whether to output "and" or "after", it is necessary for it to remember what has already been generated (i.e., $[\![x_1]\!]$ or $[\![x_2]\!]$).

Another example is `LTURN LTURN LTURN LTURN`, which can be translated into either "turn around left" or two repetitions of "turn opposite left". Deciding whether to output "and" after the first phrase requires the model to remember whether "around" was generated previously.

In §6.6 we experimentally evaluate the proposed NACS task using the same scenarios as SCAN ('simple', 'length' and 'primitive'). We observe that NACS prefers more advanced models that could capture long-term dependencies in the output (now a command sequence) better. However, we notice that even these powerful models, equipped with GRUs and attention, cannot systematically generalize to this task, as was also observed by Lake and Baroni (2018). Based on this observation, we believe that NACS (or perhaps a combination of SCAN and NACS) is better suited for evaluating any future progress in this direction.

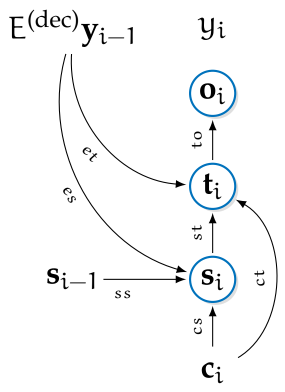


Figure 6.3: The decoder of Bahdanau et al. (2015)

6.5 SEQUENCE-TO-SEQUENCE MODELS

In this section, we briefly describe the sequence-to-sequence models we use for evaluating on SCAN and its proposed sibling NACS. For a detailed description see §2.1.1.

We directly model the probability of a target sequence given a source sequence $p(y_1^n \mid x_1^m)$. Our encoder-decoder is modeled after Cho et al. (2014) and our attention-based encoder-decoder after Bahdanau et al. (2015). The attention-based decoder is a function that takes as input the previous target word embedding $E^{(dec)}\mathbf{y}_{i-1}$, the context vector $\mathbf{c}_i$, and the previous decoder hidden state $\mathbf{s}_{i-1}$: $\mathbf{s}_i = f(E^{(dec)}\mathbf{y}_{i-1}, \mathbf{c}_i, \mathbf{s}_{i-1})$. See Figure 6.3.

The prediction for the current time step is then made from a pre-output layer $\mathbf{t}_i = W_e E^{(dec)}\mathbf{y}_{i-1} + W_c\mathbf{c}_i + W_s\mathbf{s}_i$. We do not apply a max-out layer and directly obtain the output by $\mathbf{o}_i = W_o\mathbf{t}_i$. For the encoder-decoder without attention, the prediction is made directly from decoder state $\mathbf{s}_i$. We vary the recurrent cell, experimenting with simple RNN (Elman, 1990), GRU (Cho et al., 2014), and LSTM cells (Hochreiter and Schmidhuber, 1997). For conciseness we only report results with RNN and GRU cells in the main text, and LSTM results in Appendix B.

In this chapter we investigate the properties of both SCAN and NACS using recurrent sequence-to-sequence models for evaluation. We leave the investigation of alternative architectures (see, e.g., Chen et al., 2018; Gehring et al., 2017b; Vaswani et al., 2017) for future work.

6.6 EXPERIMENTS

6.6.1 *Settings*

Our models are implemented in PyTorch and trained using mini-batch SGD with an initial learning rate of 0.2, decayed by 0.96 each epoch. We use a batch size of 32, 256 hidden units (64 for embeddings), and a dropout rate of 0.2. We test on all SCAN/NACS tasks[1], as well as on the Fr-En Machine Translation (MT) task that L&B used. The reported results are averaged over three runs for each experiment. Models *with* attention are marked as such with +*Attn*, e.g. 'GRU +Attn'.

VALIDATION SET. L&B split each SCAN subtask into a training set (80%) and a test set (20%). They train for a fixed number of updates (100k) and evaluate on the test set. Because any training run without early stopping may have missed the optimal solution (Caruana et al., 2001), we believe their results may not reflect the reality as closely as they could. We thus augment each of the SCAN variants with a validation set that follows the training distribution but contains examples that are not contained in the corresponding training set. This allows us to incorporate early stopping in our experiments so that they are better benchmarks for evaluating future progress. For each experiment we remove 10% of the training examples to be used as a validation set.

EVALUATION. Following L&B we measure performance according to sequence-level accuracy, i.e., whether the generated sequence entirely matches the reference. This metric is also used for early stopping. For NACS, an output (command) is considered correct if its interpretation ('back-mapping') produces the input action sequence.

ABLATIONS. To test our hypothesis, we *remove* the connections from the previous target word embedding $E^{(dec)}\mathbf{y}_{i-1}$ to the decoder state and the pre-output layer (*es* and *et* in Figure 6.3), so that the current prediction is not informed by previous outputs. If our hypothesis in §6.3 is correct, then these simpler models should still be able to make the correct predictions on SCAN, but not on NACS.

1 https://github.com/bastings/NACS

	Simple		Length	
	SCAN	NACS	SCAN	NACS
GRU	100.0 ±0.0	99.0 ±0.1	14.4 ±0.8	12.9 ±1.2
RNN +Attn	100.0 ±0.0	99.8 ±0.1	9.6 ±0.9	19.4 ±0.7
RNN +Attn -Dep	100.0 ±0.0	61.1 ±0.3	11.7 ±3.2	0.5 ±0.2
GRU +Attn	100.0 ±0.0	99.8 ±0.1	18.1 ±1.1	17.2 ±1.9
GRU +Attn -Dep	100.0 ±0.0	51.2 ±1.2	17.8 ±1.7	2.0 ±1.4
L&B best	99.8	-	20.8	-
L&B best overall	99.7	-	13.8	-

Table 6.1: Test scores on the simple and length tasks. *+Attn* marks attention, *-Dep* has the connections from the previous target word embedding removed (e_s and e_t in Figure 6.3). L&B$_{\text{best}}$ is the best reported score by L&B, and L&B$_{\text{best overall}}$ is the score for their best-scoring model all tasks considered.

6.6.2 *Results and Analysis*

SIMPLE AND LENGTH. Table 6.1 shows the results on the first two SCAN and NACS tasks: 'simple' and 'length'. We can see that that all model variants perform (near) perfectly on the SCAN simple task, where generalization to a random subset of the data is tested. While this is impressive, results for the severed models (+Attn -Dep) on the simple task for NACS show that it is possible to have a perfect accuracy on SCAN, while at the same time failing to do well on NACS.[2] Crucially, these results show that a (near) perfect score on SCAN does not imply strong generalization. A model can exploit the determinism and lack of target-side temporal dependencies of SCAN by developing a simple translation strategy such as advancing a pointer and translating word by word, and the use of such a simple strategy is not revealed by SCAN.

For the length experiment accuracies are low, and it seems difficult to generalize to sequences longer than those seen during training. Interestingly, the severed (-Dep) simple RNN does slightly better than the non-severed model. Also on the length task we can see that NACS is harder to solve, and that the severed models have no chance at all at solving it.

2 The LSTM models follow a similar pattern. See Appendix B.

	Turn left		Jump	
	SCAN	NACS	SCAN	NACS
GRU	53.4 ±11.7	47.5 ±4.7	0.0 ±0.0	0.0 ±0.0
RNN +Attn	81.1 ±14.7	44.1 ±0.9	1.9 ±1.2	0.3 ±0.3
RNN +Attn -Dep	92.0 ±5.8	18.6 ±1.0	2.7 ±1.7	0.0 ±0.0
GRU +Attn	59.1 ±16.8	55.9 ±3.5	12.5 ±6.6	0.0 ±0.0
GRU +Attn -Dep	90.8 ±3.6	16.9 ±1.2	0.7 ±0.4	0.0 ±0.0
L&B best	90.3	-	1.2	-
L&B best overall	90.0	-	0.1	-

Table 6.2: Test scores on the primitive (turn left, jump) task. *+Attn* marks attention, *-Dep* has the connections from the previous target word embedding removed (*es* and *et* in Figure 6.3). L&B$_{\text{best}}$ is the best reported score by L&B, and L&B$_{\text{best overall}}$ is the score for their best-scoring model all tasks considered.

PRIMITIVE. Table 6.2 shows the results for the two 'primitive' task. In the first, 'turn left' is used as the primitive, and in the second 'jump' is used. Remember that in this setting all commands that are not exactly the chosen command ('turn left' or 'jump') by itself, are part of the test set, while all other commands make the training and validation set.

We observe remarkably higher scores for the 'turn left' task. That task benefits from `TURNL` occurring on the target-side in other contexts during training, which is not the case for 'jump'.[3] Again, we see that NACS is harder than SCAN, and importantly, the severed (-Dep) models seem to have an advantage here on SCAN, even though they are too simple for more realistic scenarios. The results for 'jump' are strikingly low. It seems really difficult to generalize to new commands composed with it. We will discuss this next.

ADDITIONAL JUMP COMMANDS. Note that for the 'primitive' task there is no evidence in the training data that 'jump' is a verb, and it might be difficult to learn a good embedding for it. This might explain the poor generalization performance. For that reason, Table 6.3 shows results where an in-

3 See Lake and Baroni (2018) for a discussion.

		1	2	4	8	16	32
RNN +Attn	SCAN	35.0 ±2.8	48.6 ±8.1	77.6 ±2.6	89.2 ±3.8	98.7 ±1.3	99.8 ±0.1
RNN +Attn -Dep	SCAN	29.5 ±10.5	53.3 ±10.2	82.4 ±4.7	98.8 ±0.8	99.8 ±0.1	100.0 ±0.0
GRU +Attn	SCAN	58.2 ±12.0	67.8 ±3.4	80.3 ±7.0	88.0 ±6.0	98.3 ±1.8	99.6 ±0.2
GRU +Attn -Dep	SCAN	70.9 ±11.5	61.3 ±13.5	83.5 ±6.1	99.0 ±0.4	99.7 ±0.2	100.0 ±0.0
L&B	SCAN	0.1	0.1	4.1	15.3	70.2	89.9
RNN +Attn	NACS	2.8 ±0.8	9.3 ±7.3	24.7 ±4.2	43.7 ±4.4	57.1 ±5.2	69.1 ±2.1
RNN +Attn -Dep	NACS	0.4 ±0.1	0.9 ±0.2	2.4 ±0.3	3.9 ±0.3	9.3 ±0.3	15.9 ±1.4
GRU +Attn	NACS	5.5 ±1.8	9.2 ±2.8	11.0 ±1.5	21.9 ±2.4	23.5 ±0.6	42.0 ±1.5
GRU +Attn -Dep	NACS	0.1 ±0.1	0.6 ±0.2	2.0 ±0.2	3.2 ±0.2	5.8 ±1.1	10.9 ±0.8

Table 6.3: Test scores on the 'jump' task with additional commands. +Attn marks attention, -Dep has the e_s and e_t connections removed (Figure 6.3). The test set contains all jump commands except the 32 used for training. Columns indicate how many commands with 'jump' were added to the training set, such as 'jump around left thrice.'

creasing (small) amount of additional (composed) 'jump' commands were added to the training data. We can see that performance quickly goes up when adding more commands.[4] Again, the simpler models (+Attn -Dep) perform better here on SCAN, but not on NACS.

machine translation. For the final experiment we repeat the basic English-French MT experiment of Lake and Baroni (2018) in both directions. Note that this data contains only very short sentence pairs, and is not at all comparable to the larger-scale MT experiments of the previous chapters. However, it is slightly more realistic than SCAN and NACS. Table 6.4 shows that models that perform well on NACS also perform well here, with the GRU outperforming the other cells (see Appendix B for other cell types).

In a setting similar to the 'primitive' task, we also added the sentence pair "I am daxy"/"je suis daxiste" to the training set, with the goal to test if eight *novel* sentences that contain 'daxy' are correctly translated. In our setting with mini-batching and early-stopping, the GRU gets 70.8% (En-Fr) and 54.2% (Fr-En) of the daxy-sentences right, which is surprisingly good compared to the 12.5% reported in Lake and Baroni (2018).

4 L&B performed this experiment without attention, which we show has a large positive impact.

	En-Fr	Fr-En
GRU +Attn	32.1 ±0.3	37.5 ±0.6
GRU +Attn -Dep	30.2 ±0.3	35.9 ±0.3

Table 6.4: Results (BLEU) on the Machine Translation experiment for both directions using a GRU. See Appendix B for results using SRN and LSTM cells.

6.7 RELATED WORK

Ever since Fodor and Pylyshyn (1988) conjectured that neural networks are unable to show strong generalization, many attempts were made to show that the opposite is true, leading to inconclusive evidence. For example, Phillips (1998) found that feed-forward nets and RNNs do not always generalize to novel 2-tuples on an auto-association task, while Wong and Wang (2007) and Brakel and Frank (2009) found that RNNs *can* show systematic behavior in a language modeling task.

In the context of analyzing RNNs, Rodriguez and Wiles (1998) found that a simple RNN can develop a symbol-sensitive counting strategy for accepting a simple (palindrome) context-free language. Weiss et al. (2018) show that LSTMs and simple RNNs with ReLU-activation can learn to count unboundedly, while GRUs cannot. Linzen et al. (2016) probed the sensitivity of LSTMs to hierarchical structure (not necessarily in novel constructions). Instead of a binary choice, with SCAN a sequence-to-sequence model productively generates an output string.

Liska et al. (2018) found that a small number of identical RNNs trained with different initializations show compositional behavior on a function composition task, suggesting that more specific architectures may not be necessary. Loula et al. (2018) extend the 'primitive' experiment that we also conducted in this chapter to settings where the model only needs to recombine functional words such as 'around' and 'right' that are seen many times during training. This setup remedies the issue that we discussed that it might be difficult to learn the embedding of a word only seen in isolation, such as 'jump' in the 'primitive' experiment. Loula et al. find that models can generalize if they see a pattern often enough times. For example, seeing 'X around right' during training enables generalization to 'jump around right'. However, models still fail to generalize in more compositional ways, such as infering 'around right' after learning 'right' during training. Dessì and Baroni (2019) also look at the SCAN tasks, but replace the recurrent networks with convolutional ones. They find that convolutional neural networks perform better on SCAN, but do not generalize in a systematic way. Lake and Baroni (2018) introduced the SCAN data set to study systematic compositionality in recurrent sequence-to-sequence models, including gating mechanisms and attention. This chapter complements that work and aims to facilitate future progress by showing that SCAN does not necessarily test for strong generalization. Baroni (2019) reviews the recent advances in understanding the systematic generalization behavior of neural networks, including the ones we discussed here, in more detail. He argues that the findings as discussed here should be of interest to linguists and cognitive scientists, because they provide a fresh perspective on the possible computational strategies related to linguistic productivity beyond rule-based compositionality, including how to generalize less systematically.

More recently, Keysers et al. (2020) contribute to the generalization debate a carefully constructed question answering dataset that is a lot more realistic than SCAN, even though, like SCAN, it does not have any ambiguity like real language. They make sure that the generalization that is measured is about combining known parts (atoms), which they call 'compound divergence', and show that accuracy is negatively correlated with the ability to recombine known parts. Finally, Ruis et al. (2020) proposed a grounded version of SCAN, gSCAN, that grounds a generalization task in a grid world. Also in this grounded setting they show that models have difficulty generalizing.

6.8 CONCLUSIONS

In the quest for strong generalization, benchmarks measuring progress are an important component. The existing SCAN benchmark allows too simple models to shine, without the need for compositional generalization. In this chapter we proposed NACS to remedy that. NACS still requires systematicity, while introducing stochasticity and strong temporal dependencies on the target side. Arguably, a good benchmark needs at least those properties, in order not to fall prey to trivial solutions, which do not work on more realistic use-cases for sequence-to-sequence models such as machine translation.

7 CONCLUSIONS

This thesis covered two main topics: linguistically-informed deep learning for NLP, and interpreting and analyzing neural networks for NLP. We will now go over the main findings.

7.1 LINGUISTICALLY-INFORMED DEEP LEARNING FOR NLP

In Chapter 3 we asked the question if neural networks can benefit from linguistic structure in the form of dependency trees and dependency-based semantic role labeling structures. To answer this question, we first conditioned an attention-based neural machine translation model on dependency trees obtained by parsing the training data with SyntaxNet. We saw consistent improvements in BLEU score when conditioning on dependency syntax, even though the margin got smaller as we used more training data. After conditioning on just syntactic trees, we also wondered if semantic role labeling structures can be beneficial, and found that they can have somewhat complementary strengths. While we focused on recurrent models in this chapter, later work by Currey and Heafield (2019) reached similar conclusions for a Transformer architecture, where improvements can be achieved for lower resource scenarios, with the performance gap closing when adding more data. An interesting avenue for future work is to explore what happens exactly when more data is added, and if there is still a role for linguistic structures as data sets grow. Even if the gap closes with more training data, there are many languages in the world for which we do not have data sets as large as those available for English, and therefore it remains relevant to look at incorporating linguistic information in deep neural networks. Another avenue for future work is to investigate if the performance gap closes because syntax and semantics are already successfully captured by a model, and to find better ways to extract such knowledge. While there is some evidence that this happens to some extent (e.g. Linzen et al., 2016), the linguistic rules that are captured might not be optimal or complete, which could hurt generalization.

In Chapter 4 we wondered if we could do away with the externally parsed structures, and induce them instead with translation as a downstream task. We used a probabilistic model with separate graph induction and translation components, and studied if our induced latent graphs are beneficial using three different encoders. In the case of LSTM encoders the induced graphs turned out to be quite trivial, reminiscent of how some individual Transformer heads turn out to be trivial, while for the simpler word embedding and CNN encoders they contain useful, potentially long-distance dependencies. Other works, e.g. Williams et al. (2018), also induced structure, and also found that this structure does not resemble linguistic structure such as treebank syntax, even though it can be beneficial for a certain task. Recently, much focus has gone into the study of how much linguistic information is captured by pre-trained sentence encoders such as BERT (Devlin et al., 2019). For example, it is possible to uncover syntactic distance from the representations of these models (Hewitt and Manning, 2019), and using diagnostic classifiers (Hupkes et al., 2018) or 'probes' they can be shown to contain information usable for e.g., POS-tagging, parsing, and semantic role labeling (see e.g., Tenney et al., 2019a,b; Voita and Titov, 2020). Despite this progress in analysis, it remains an open question if syntactic knowledge is a byproduct of task-specific learning of continuous representations, and how systematic such knowledge is. Very recently, Raganato et al. (2020) show that, for a Transformer, each head but one can be given a fixed attention pattern based on positional information (e.g., attend to the previous token) without a loss in performance, and sometimes even at a gain. This exemplifies that we are still not sure how much structure is, and should be captured by neural network models for NLP. As we find more answers, insights that are obtained from research in this direction also have the potential to inform the field of linguistics.

7.2 INTERPRETING AND ANALYZING NEURAL NETWORKS

In Chapter 5 we focused on interpretability, and made a neural text classifier interpretable by having it show which parts of the input document are used for classification, and which parts aren't. This explanation is what we called the rationale. In contrast to previous work, we proposed a method that allows for training such a model using backpropagation, without having to rely on the high-variance REINFORCE estimator. Our method made use of a stretch-and-rectify technique (Louizos et al., 2018) that we used to devise a novel distribution, the HardKuma. In addition, we proposed using a Lagrangian relaxation to target a specific text selection rate, which makes it easy to find a trade-off between interpretability and classification performance. With our method we found that we can produce rationales that overlap more with human annotations than previous work. For future research, the HardKuma could be used to make different kinds of models more interpretable, such as those used for question answering or fact checking. While our method modifies a model to make it interpretable, recently there has been a lot of interest in methods that try to interpret already trained models (see e.g., Poerner et al., 2018). It would make sense to compare these various methods, e.g., using the ERASER benchmark (DeYoung et al., 2019). One particular challenge is to find a way to better apply post-hoc methods to textual data; in particular to find ways to generate more natural input perturbations. A final concern regarding evaluation is faithfulness (see e.g., Jacovi and Goldberg, 2020). If we are concerned about uncovering the true reasoning process of a model, then evaluating rationales with human annotations might not be adequate.

In Chapter 6 we investigated the generalization properties of sequence-to-sequence models, such as the ones we use for neural machine translation. We looked at a popular benchmark, SCAN, and found that it allows too simple models to shine, without the need for compositional generalization. We proposed to turn the SCAN problem around, which resulted in the NACS tasks. NACS still requires systematicity, while introducing stochasticity and strong temporal dependencies on the target side. Arguably, a good benchmark needs at least those properties, in order not to fall prey to trivial solutions, which do not work on more realistic use-cases for sequence-to-sequence models such as machine translation. The importance of the point made in this chapter also is made clear by the fact that other, newer benchmarks, such as Keysers et al. (2020), like SCAN also do not have ambiguity and therefore fail to be fully effective tests.

7.3 FINAL WORDS

Our field has seen a rapid adoption of deep neural networks to solve a multitude of NLP tasks. Observing that, this thesis contributes in two ways:

- we investigated the potential of explicit linguistic bias in end-to-end neural models, and advanced the field by contributing techniques and experimental results;
- we proposed a method to make neural predictions more interpretable, and an improved benchmark for studying generalization performance.

While it is undeniable that lots of work needs to happen in both these areas, this thesis tried to contribute in a positive way to more understanding of the black box neural networks now that they have become ubiquitous.

Part III

APPENDIX

A INTERPRETABLE NEURAL PREDICTIONS

A.1 Kumaraswamy distribution

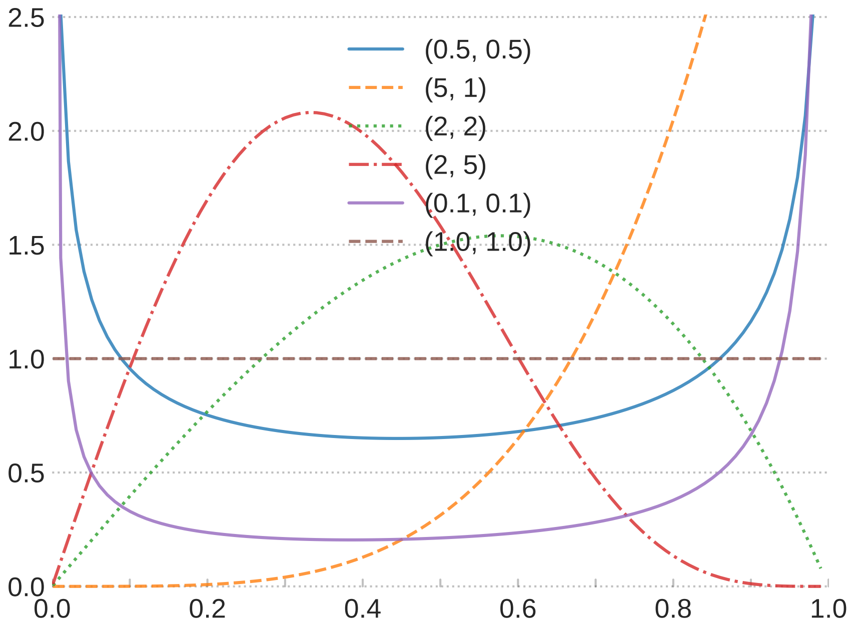


Figure A.1: Kuma plots for various (a, b) parameters.

A Kumaraswamy-distributed variable $K \sim \text{Kuma}(a, b)$ takes on values in the open interval $(0, 1)$ and has density

$$f_K(k; a, b) = abk^{a-1}(1 - k^a)^{b-1} \,, \tag{A.1}$$

where $a \in \mathbb{R}_{>0}$ and $b \in \mathbb{R}_{>0}$ are shape parameters. Its cumulative distribution takes a simple closed-form expression

$$F_K(k; a, b) = \int_0^k f_K(\xi|a, b) d\xi \tag{A.2a}$$

$$= 1 - (1 - k^a)^b \,, \tag{A.2b}$$

with inverse

$$F_K^{-1}(u; a, b) = \left(1 - (1 - u)^{1/b}\right)^{1/a} \,. \tag{A.3}$$

A.1.1 *Generalized-support Kumaraswamy*

We can generalize the support of a Kumaraswamy variable by specifying two constants $l < r$ and transforming a random variable $K \sim \text{Kuma}(a, b)$ to obtain $T \sim \text{Kuma}(a, b, l, r)$ as shown in (A.4, left).

$$t = l + (r - l)k \qquad\qquad k = {}^{(t-l)}/_{(r-l)} \tag{A.4}$$

The density of the resulting variable is

$$\begin{aligned} & f_T(t; a, b, l, r) && \text{(A.5a)} \\ & = f_K\left(\tfrac{t-l}{r-l}; a, b\right) \left|\frac{dk}{dt}\right| && \text{(A.5b)} \\ & = f_K\left(\tfrac{t-l}{r-l}; a, b\right) \frac{1}{(r-l)} && \text{(A.5c)} \end{aligned}$$

where $r - l > 0$ by definition. This *affine* transformation leaves the cdf unchanged, i.e.

$$\begin{aligned} F_T(t_0; a, b, l, r) &= \int_{-\infty}^{t_0} f_T(t; a, b, l, r)dt \\ &= \int_{-\infty}^{t_0} f_K\left(\tfrac{t-l}{r-l}; a, b\right) \frac{1}{(r-l)}dt \\ &= \int_{-\infty}^{\frac{t_0-l}{r-l}} f_K(k; a, b)\frac{1}{(r-l)}(r-l)dk \\ &= F_K\left(\tfrac{t_0-l}{r-l}; a, b\right) . \end{aligned} \tag{A.6}$$

Thus we can obtain samples from this generalized-support Kumaraswamy by sampling from a uniform distribution $\mathcal{U}(0, 1)$, applying the inverse transform (A.3), then shifting and scaling the sample according to (A.4, left).

A.1.2 *Rectified Kumaraswamy*

First, we stretch a Kumaraswamy distribution to include 0 and 1 in its support, that is, with $l < 0$ and $r > 1$, we define $T \sim \text{Kuma}(a, b, l, r)$. Then we apply a *hard-sigmoid* transformation to this variable, that is, $h = \min(0, \max(1, t))$, which results in a *rectified* distribution which gives support to the closed interval $[0, 1]$. We denote this rectified variable by

$$H \sim \text{HardKuma}(a, b, l, r) \tag{A.7}$$

whose distribution function is

$$\begin{aligned} & f_H(h; a, b, l, r) = \\ & \mathbb{P}(h = 0)\delta(h) + \mathbb{P}(h = 1)\delta(h - 1) \\ & + \mathbb{P}(0 < h < 1)\frac{f_T(h; a, b, l, r)\mathbb{1}_{(0,1)}(h)}{\mathbb{P}(0 < h < 1)} \end{aligned} \tag{A.8}$$

where

$$\begin{aligned} & \mathbb{P}(h = 0) = \mathbb{P}(t \leqslant 0) \\ & = F_T(0; a, b, l, r) = F_K(^{-l}/_{(r-l)}; a, b) \end{aligned} \tag{A.9}$$

is the probability of sampling exactly 0, where

$$\begin{aligned} & \mathbb{P}(h = 1) = \mathbb{P}(t \geqslant 1) = 1 - \mathbb{P}(t < 1) \\ & = 1 - F_T(1; a, b, l, r) \\ & = 1 - F_K(^{(1-l)}/_{(r-l)}; a, b) \end{aligned} \tag{A.10}$$

is the probability of sampling exactly 1, and

$$\mathbb{P}(0 < h < 1) = 1 - \mathbb{P}(h = 0) - \mathbb{P}(h = 1) \tag{A.11}$$

is the probability of drawing a continuous value in $(0, 1)$. Note that we used the result in (A.6) to express these probabilities in terms of the tractable cdf of the original Kumaraswamy variable.

A.2 IMPLEMENTATION DETAILS

A.2.1 *Multi-aspect Sentiment Analysis*

Our hyperparameters are taken from Lei et al. (2016) and listed in Table A.1. The pre-trained word embeddings and data sets are available online at `http://people.csail.mit.edu/taolei/beer/`. We train for 100 epochs and select the best models based on validation loss. For the MSE trade-off experiments on all aspects combined, we train for a maximum of 50 epochs.

For the Bernoulli baselines we vary L_0 weight λ_1 among $\{0.0002, 0.0003, 0.0004\}$, just as in the original paper. We set the fused lasso (coherence) weight λ_2 to $2 * \lambda_1$.

For the HardKuma models we set a target selection rate to the values targeted in Table 5.2, and optimize to this end using the Lagrange multiplier. We chose the fused lasso weight from $\{0.0001, 0.0002, 0.0003, 0.0004\}$.

Optimizer	Adam
Learning rate	0.0004
Word embeddings	200D (Wiki, fixed)
Hidden size	200
Batch size	256
Dropout	0.1, 0.2
Weight decay	$1 * 10^{-6}$
Cell	RCNN

Table A.1: Beer hyperparameters.

A.2.1.1 *Recurrent Unit*

In our multi-aspect sentiment analysis experiments we use the RCNN of Lei et al. (2016). Intuitively, the RCNN is supposed to capture n-gram features that are not necessarily consecutive. We use the bigram version (filter width $n = 2$) used in Lei et al. (2016), which is defined as:

$$\begin{aligned}
\boldsymbol{\lambda}_t &= \sigma(W^{\lambda}\mathbf{x}_t + U^{\lambda}\mathbf{h}_{t-1} + \mathbf{b}^{\lambda}) \\
\mathbf{c}_t^{(1)} &= \boldsymbol{\lambda}_t \odot \mathbf{c}_{t-1}^{(1)} + (1 - \boldsymbol{\lambda}_t) \odot W_1\mathbf{x}_t \\
\mathbf{c}_t^{(2)} &= \boldsymbol{\lambda}_t \odot \mathbf{c}_{t-1}^{(2)} + (1 - \boldsymbol{\lambda}_t) \odot (\mathbf{c}_{t-1}^{(1)} + W_2\mathbf{x}_t) \\
\mathbf{h}_t &= \tanh\left(\mathbf{c}_t^{(2)} + \mathbf{b}\right)
\end{aligned}$$

A.2.1.2 *Expected values for dependent latent variables*

The expected L_0 is a chain of nested expectations, and we solve each term

$$\begin{aligned}
&\mathbb{E}_{p(z_i|x,z_{<i})}\left[\mathbb{I}[z_i \neq 0] \mid z_{<i}\right] \\
&= 1 - F_K\left(\tfrac{-l}{r-l}; a_i, b_i\right)
\end{aligned} \tag{A.12}$$

as a function of a sampled prefix, and the shape parameters $a_i, b_i = g_i(x, z_{<i}; \phi)$ are predicted in sequence.

A.2.2 *Sentiment Classification (SST)*

For sentiment classification we make use of the PyTorch bidirectional LSTM module for encoding sentences, for both the rationale extractor and the classifier. The BiLSTM final states are concatenated, after which a linear layer followed by a softmax produces the prediction. Hyperparameters are listed in Table A.2. We apply dropout to the embeddings and to the input of the output layer.

Optimizer	Adam
Learning rate	0.0002
Word embeddings	300D Glove (fixed)
Hidden size	150
Batch size	25
Dropout	0.5
Weight decay	$1 * 10^{-6}$
Cell	LSTM

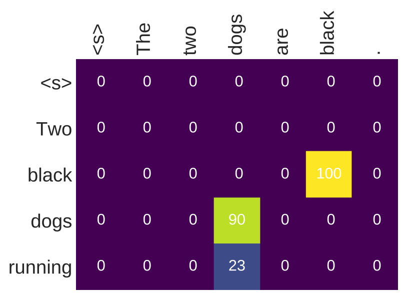
Table A.2: SST hyperparameters.

A.2.3 *Natural Language Inference (SNLI)*

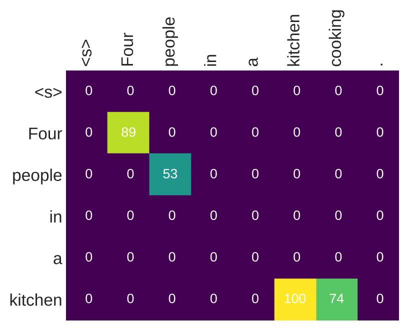
Our hyperparameters are taken from Parikh et al. (2016) and listed in Table A.3. Different from Parikh et al. is that we use Adam as the optimizer and a batch size of 64. Word embeddings are projected to 200 dimensions with a trained linear layer. Unknown words are mapped to 100 unknown word classes based on the MD5 hash function, just as in Parikh et al. (2016), and unknown word vectors are randomly initialized. We train for 100 epochs, evaluate every 1000 updates, and select the best model based on validation loss. Figure A.2 shows a correct and incorrect example with HardKuma attention for each relation type (entailment, contradiction, neutral).

Optimizer	Adam
Learning rate	0.0001
Word embeddings	300D (Glove, fixed)
Hidden size	200
Batch size	64
Dropout	0.2

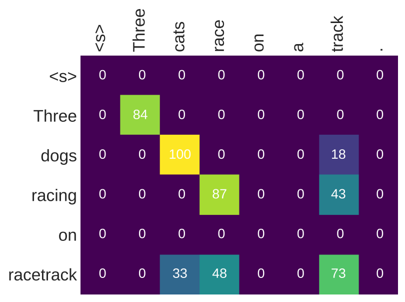
Table A.3: SNLI hyperparameters.



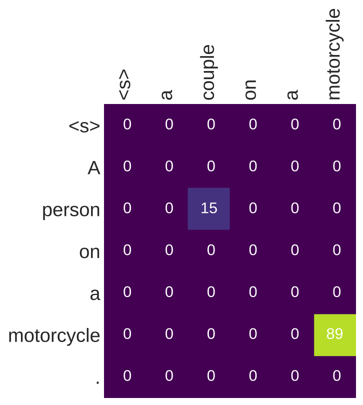
(a) Entailment (correct)



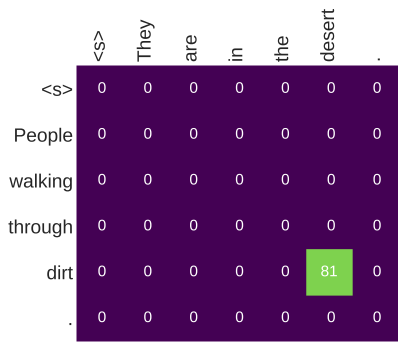
(b) Entailment (incorrect)



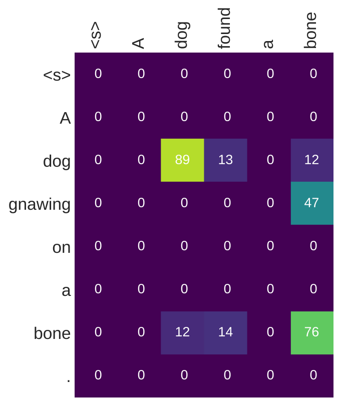
(c) Contradiction (correct)



(d) Contradiction (incorrect)



(e) Neutral (correct)



(f) Neutral (incorrect)

Figure A.2: HardKuma attention in SNLI for entailment, contradiction, and neutral.

B ADDITIONAL SCAN AND NACS RESULTS

We follow the experiments of §6.6, but include the full set of results with all three types of cells (simple RNN, GRU, LSTM).

	Simple	Length	Turn left	Jump
RNN	75.6 ±5.4	0.2 ±0.0	26.7 ±12.8	0.0 ±0.0
GRU	100.0 ±0.0	14.4 ±0.8	53.4 ±11.7	0.0 ±0.0
LSTM	99.8 ±0.1	10.1 ±2.0	56.5 ±0.8	0.1 ±0.0
RNN +Attn	100.0 ±0.0	9.6 ±0.9	81.1 ±14.7	1.9 ±1.2
RNN +Attn-Dep	100.0 ±0.0	11.7 ±3.2	92.0 ±5.8	2.7 ±1.7
GRU +Attn	100.0 ±0.0	18.1 ±1.1	59.1 ±16.8	12.5 ±6.6
GRU +Attn-Dep	100.0 ±0.0	17.8 ±1.7	90.8 ±3.6	0.7 ±0.4
LSTM +Attn	100.0 ±0.0	15.6 ±1.6	83.8 ±16.8	9.7 ±2.9
LSTM +Attn-Dep	100.0 ±0.0	12.5 ±1.3	57.6 ±3.8	0.8 ±0.5
L&B best	99.8	20.8	90.3	1.2
L&B best overall	99.7	13.8	90.0	0.1

Table B.1: SCAN test scores on the simple, length, and primitive (turn left and jump) tasks. For '+Attn-Dep' models we removed the connections from the previous target word embedding to the decoder state and the pre-output layer.

	Simple	Length	Turn left	Jump
RNN	26.9 ±0.2	0.2 ±0.1	26.4 ±12.0	0.0 ±0.0
GRU	99.0 ±0.1	12.9 ±1.2	47.5 ±4.7	0.0 ±0.0
LSTM	99.1 ±0.1	10.9 ±1.3	42.9 ±2.9	0.0 ±0.0
RNN +Attn	99.8 ±0.1	19.4 ±0.7	44.1 ±0.9	0.3 ±0.3
RNN +Attn-Dep	61.1 ±0.3	0.5 ±0.2	18.6 ±1.0	0.0 ±0.0
GRU +Attn	99.8 ±0.1	17.2 ±1.9	55.9 ±3.5	0.0 ±0.0
GRU +Attn-Dep	51.2 ±1.2	2.0 ±1.4	16.9 ±1.2	0.0 ±0.0
LSTM +Attn	99.1 ±0.2	17.1 ±2.0	48.3 ±1.7	0.0 ±0.0
LSTM +Attn-Dep	38.9 ±0.9	1.0 ±0.5	17.2 ±1.2	0.0 ±0.0

Table B.2: NACS test scores on the simple, length, and primitive (turn left and jump) tasks. For '+Attn-Dep' models we removed the connections from the previous target word embedding to the decoder state and the pre-output layer.

	0	1	2	4	8	16	32
RNN	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.1 ±0.0	0.1 ±0.1	0.5 ±0.3	1.4 ±0.3
GRU	0.1 ±0.0	0.2 ±0.1	0.6 ±0.2	2.5 ±1.1	3.3 ±0.9	13.1 ±2.4	42.4 ±2.5
LSTM	0.1 ±0.0	0.3 ±0.2	1.3 ±0.2	3.8 ±1.8	2.5 ±1.1	6.5 ±2.7	21.3 ±1.4
RNN +Attn	3.5 ±3.0	35.0 ±2.8	48.6 ±8.1	77.6 ±2.6	89.2 ±3.8	98.7 ±1.3	99.8 ±0.1
RNN +Attn-Dep	2.7 ±1.7	29.5 ±10.5	53.3 ±10.2	82.4 ±4.7	98.8 ±0.8	99.8 ±0.1	100.0 ±0.0
GRU +Attn	12.5 ±6.6	58.2 ±12.0	67.8 ±3.4	80.3 ±7.0	88.0 ±6.0	98.3 ±1.8	99.6 ±0.2
GRU +Attn-Dep	0.7 ±0.4	70.9 ±11.5	61.3 ±13.5	83.5 ±6.1	99.0 ±0.4	99.7 ±0.2	100.0 ±0.0
LSTM +Attn	7.8 ±0.9	40.2 ±9.3	37.7 ±10.7	50.3 ±13.9	62.2 ±7.7	94.0 ±2.7	98.6 ±1.0
LSTM +Attn-Dep	0.8 ±0.6	39.0 ±6.5	43.6 ±17.6	66.0 ±1.6	86.1 ±2.3	98.7 ±1.6	99.8 ±0.2
L&B	0.1	0.1	0.1	4.1	15.3	70.2	89.9

Table B.3: SCAN test scores for jump with additional composed commands.

	0	1	2	4	8	16	32
RNN	0.0 ±0.0	0.1 ±0.0	0.1 ±0.1	0.2 ±0.0	0.7 ±0.2	0.4 ±0.0	0.8 ±0.1
GRU	0.0 ±0.0	0.3 ±0.2	0.4 ±0.1	0.3 ±0.2	1.0 ±0.4	5.8 ±0.1	20.8 ±2.2
LSTM	0.0 ±0.0	0.6 ±0.4	0.5 ±0.3	0.7 ±0.0	1.0 ±0.3	3.7 ±0.4	11.4 ±1.2
RNN +Attn	0.3 ±0.3	2.8 ±0.8	9.3 ±7.3	24.7 ±4.2	43.7 ±4.4	57.1 ±5.2	69.1 ±2.1
RNN +Attn-Dep	0.0 ±0.0	0.4 ±0.1	0.9 ±0.2	2.4 ±0.3	3.9 ±0.3	9.3 ±0.3	15.9 ±1.4
GRU +Attn	0.0 ±0.0	5.5 ±1.8	9.2 ±2.8	11.0 ±1.5	21.9 ±2.4	23.5 ±0.6	42.0 ±1.5
GRU +Attn-Dep	0.0 ±0.0	0.1 ±0.1	0.6 ±0.2	2.0 ±0.2	3.2 ±0.2	5.8 ±1.1	10.9 ±0.8
LSTM +Attn	0.0 ±0.0	2.1 ±0.2	3.7 ±0.9	6.6 ±0.5	12.5 ±2.5	21.8 ±2.6	34.2 ±1.7
LSTM +Attn-Dep	0.0 ±0.0	0.4 ±0.2	0.9 ±0.1	1.5 ±0.2	1.9 ±0.3	3.2 ±0.6	7.4 ±0.9

Table B.4: NACS test scores for jump with additional composed commands.

	En-Fr	Fr-En
RNN +Attn	29.1 ±0.4	34.9 ±0.8
RNN +Attn-Dep	27.5 ±0.7	32.9 ±0.8
GRU +Attn	32.1 ±0.3	37.5 ±0.6
GRU +Attn-Dep	30.2 ±0.3	35.9 ±0.3
LSTM +Attn	31.5 ±0.2	36.9 ±1.1
LSTM +Attn-Dep	28.7 ±0.2	34.0 ±0.1

Table B.5: Results (BLEU) on the Machine Translation experiment for both directions.

	En-Fr	Fr-En
RNN +Attn	79.2 ±15.6	41.7 ±5.9
RNN +Attn-Dep	66.7 ±5.9	41.7 ±5.9
GRU +Attn	70.8 ±11.8	54.2 ±5.9
GRU +Attn-Dep	58.3 ±5.9	45.8 ±11.8
LSTM +Attn	75.0 ±10.2	41.7 ±15.6
LSTM +Attn-Dep	50.0 ±10.2	41.7 ±5.9

Table B.6: Machine Translation: accuracy on eight novel sentences containing 'daxy' ('daxiste').

	En-Fr	Fr-En
RNN +Attn	66.7 ±5.9	20.8 ±5.9
RNN +Attn-Dep	66.7 ±5.9	29.2 ±15.6
GRU +Attn	62.5 ±0.0	33.3 ±5.9
GRU +Attn-Dep	66.7 ±5.9	25.0 ±20.4
LSTM +Attn	66.7 ±5.9	25.0 ±10.2
LSTM +Attn-Dep	62.5 ±0.0	25.0 ±17.7

Table B.7: Machine Translation: accuracy on eight novel sentences containing 'tired' ('fatigué').

BIBLIOGRAPHY

Roee Aharoni and Yoav Goldberg (2017). "Towards String-To-Tree Neural Machine Translation." In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 132–140.

Robert B Allen (1987). "Several studies on natural language and back-propagation." In: *Proceedings of the IEEE First International Conference on Neural Networks*. Vol. 2. S 335. IEEE Piscataway, NJ, p. 341.

Uri Alon, Omer Levy, and Eran Yahav (2019). "code2seq: Generating Sequences from Structured Representations of Code." In: *International Conference on Learning Representations*.

David Alvarez-Melis and Tommi Jaakkola (2017). "A causal framework for explaining the predictions of black-box sequence-to-sequence models." In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 412–421.

Wilker Aziz, Miguel Rios, and Lucia Specia (July 2011). "Shallow Semantic Trees for SMT." In: *Proceedings of the Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland: Association for Computational Linguistics, pp. 316–322.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). "Layer normalization." In: *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. San Diego, CA.

Kathryn Baker, Michael Bloodgood, Bonnie J Dorr, Chris Callison-Burch, Nathaniel W Filardo, Christine Piatko, Lori Levin, and Scott Miller (2012). "Modality and negation in simt use of modality and negation in semantically-informed syntactic mt." In: *Computational Linguistics* 38.2, pp. 411–438.

Laura Banarescu et al. (2012). "Abstract meaning representation (AMR) 1.0 specification." In: *Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544.

Yehoshua Bar-Hillel (1960). "The present status of automatic translation of languages." In: *Advances in computers*. Vol. 1. Elsevier, pp. 91–163.

Marco Baroni (2019). "Linguistic generalization and compositionality in modern artificial neural networks." In: *CoRR* abs/1904.00157. arXiv: 1904.00157.

Jasmijn Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an (Sept. 2017). "Graph Convolutional Encoders for Syntax-aware Neural Machine Translation." In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1947–1957.

Marzieh Bazrafshan and Daniel Gildea (Aug. 2013). "Semantic Roles for String to Tree Machine Translation." In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 419–423.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn (July 2018). "Graph-to-Sequence Learning using Gated Graph Neural Networks." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 273–283.

Yonatan Belinkov and James Glass (Mar. 2019). "Analysis Methods in Neural Language Processing: A Survey." In: *Transactions of the Association for Computational Linguistics* 7, pp. 49–72.

Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. (1994). "Learning long-term dependencies with gradient descent is difficult." In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin (2003). "A neural probabilistic language model." In: *Journal of machine learning research* 3.Feb, pp. 1137–1155.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville (2013). "Estimating or propagating gradients through stochastic neurons for conditional computation." In: *arXiv preprint arXiv:1308.3432*.

Alexandre Bérard, Ioan Calapodescu, and Claude Roux (2019). "Naver Labs Europe's Systems for the WMT19 Machine Translation Robustness Task." In: *arXiv preprint arXiv:1907.06488*.

Robert C Berwick and Noam Chomsky (2016). *Why only us: Language and evolution*. MIT press.

Ondřej Bojar et al. (June 2014). "Findings of the 2014 Workshop on Statistical Machine Translation." In: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, pp. 12–58.

Ondřej Bojar et al. (Sept. 2015). "Findings of the 2015 Workshop on Statistical Machine Translation." In: *Proceedings of the Tenth Workshop on Statistical Machine Translation*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1–46.

Ondřej Bojar et al. (Aug. 2016). "Findings of the 2016 Conference on Machine Translation." In: *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*. Berlin, Germany: Association for Computational Linguistics, pp. 131–198.

Léon Bottou (2012). "Stochastic gradient descent tricks." In: *Neural networks: Tricks of the trade*. Springer, pp. 421–436.

Léon Bottou and Yann LeCun (2004). "Large scale online learning." In: *Advances in neural information processing systems*, pp. 217–224.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning (Sept. 2015). "A large annotated corpus for learning natural language inference." In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 632–642.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts (Aug. 2016a). "A Fast Unified Model for Parsing and Sentence Understanding." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1466–1477.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio (Aug. 2016b). "Generating Sentences from a Continuous Space." In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, pp. 10–21.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Philémon Brakel and Stefan Frank (2009). "Strong systematicity in sentence processing by simple recurrent networks." In:

31th Annual Conference of the Cognitive Science Society (COGSCI-2009). Cognitive Science Society, pp. 1599–1604.

Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin (1990). "A statistical approach to machine translation." In: *Computational linguistics* 16.2.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer (1993). "The mathematics of statistical machine translation: Parameter estimation." In: *Computational linguistics* 19.2, pp. 263–311.

Rich Caruana (1997). "Multitask learning." In: *Machine learning* 28.1, pp. 41–75.

Rich Caruana, Steve Lawrence, and C Lee Giles (2001). "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping." In: *Advances in neural information processing systems*, pp. 402–408.

Asuncion Castano and Francisco Casacuberta (1997). "A connectionist approach to machine translation." In: *Fifth European Conference on Speech Communication and Technology*.

Antonio Castellanos, Isabel Galiano, and Enrique Vidal (1994). "Application of OSTIA to machine translation tasks." In: *International Colloquium on Grammatical Inference*. Springer, pp. 93–105.

Mia Xu Chen et al. (2018). "The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation." In: *arXiv preprint arXiv:1804.09849*.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen (2017). "Enhanced LSTM for Natural Language Inference." In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 1657–1668.

David Chiang (July 2010). "Learning to Translate with Source and Target Syntax." In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 1443–1452.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (Oct. 2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." In: *Proceedings of the 2014 Conference on Empirical Meth-*

ods in Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee (2018). "Learning to compose task-specific tree structures." In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Lonnie Chrisman (1991). "Learning recursive distributed representations for holistic computation." In: *Connection Science* 3.4, pp. 345–366.

Ronan Collobert and Jason Weston (2008). "A unified architecture for natural language processing: Deep neural networks with multitask learning." In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 160–167.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa (2011). "Natural language processing (almost) from scratch." In: *Journal of Machine Learning Research* 12.Aug, pp. 2493–2537.

Mark Craven and Jude W Shavlik (1996). "Extracting tree-structured representations of trained networks." In: *Advances in neural information processing systems*, pp. 24–30.

Anna Currey and Kenneth Heafield (2018a). "Multi-Source Syntactic Neural Machine Translation." In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 2961–2966.

— (July 2018b). "Unsupervised Source Hierarchies for Low-Resource Neural Machine Translation." In: *Proceedings of the Workshop on the Relevance of Linguistic Structure in Neural Architectures for NLP*. Melbourne, Australia: Association for Computational Linguistics, pp. 6–12.

— (2019). "Incorporating Source Syntax into Transformer-Based Neural Machine Translation." In: *Proceedings of the Fourth Conference on Machine Translation*. Florence, Italy: Association for Computational Linguistics, pp. 24–33.

Marco Damonte and Shay B. Cohen (June 2019). "Structural Neural Encoders for AMR-to-text Generation." In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 3649–3658.

Nicola De Cao, Wilker Aziz, and Ivan Titov (June 2019). "Question Answering by Reasoning Across Documents with Graph Convolutional Networks." In: *Proceedings of the 2019 Confer-*

ence of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, pp. 2306–2317.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst (2016). "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering." In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3837–3845.

Michael Denkowski and Alon Lavie (2014). "Meteor universal: Language specific translation evaluation for any target language." In: *Proceedings of the ninth workshop on statistical machine translation*, pp. 376–380.

Roberto Dessì and Marco Baroni (July 2019). "CNNs found to jump around more skillfully than RNNs: Compositional Generalization in Seq2seq Convolutional Networks." In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 3919–3923.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Minneapolos, USA: Association for Computational Linguistics.

Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C. Wallace (2019). *ERASER: A Benchmark to Evaluate Rationalized NLP Models*. arXiv: 1911.03429 [cs.CL].

Shuoyang Ding, Adithya Renduchintala, and Kevin Duh (2019). "A Call for Prudent Choice of Subword Merge Operations." In: *MT Summit*. arXiv: 1905.10453.

Finale Doshi-Velez and Been Kim (2017). "Towards A Rigorous Science of Interpretable Machine Learning." In: *arXiv e-prints*, arXiv:1702.08608, arXiv:1702.08608. arXiv: 1702.08608 [stat.ML].

Timothy Dozat and Christopher D. Manning (2017). "Deep Biaffine Attention for Neural Dependency Parsing." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Toulon, France.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams (2015). "Convolutional networks on graphs for learning molecular fingerprints." In: *Advances in neural information processing systems*, pp. 2224–2232.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith (June 2016). "Recurrent Neural Network Grammars." In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 199–209.

Jeffrey L Elman (1990). "Finding structure in time." In: *Cognitive science* 14.2, pp. 179–211.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka (Aug. 2016). "Tree-to-Sequence Attentional Neural Machine Translation." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 823–833.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho (2017). "Learning to Parse and Translate Improves Neural Machine Translation." In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 72–78.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A. Smith (2015). "Sparse Overcomplete Word Vector Representations." In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1491–1500.

Jerome A Feldman, George Lakoff, Andreas Stolcke, and Susan Hollbach Weber (1990). "Miniature language acquisition: A touchstone for cognitive science." In: *Proceedings of the 12th Annual Conference of the Cognitive Science Society*. Citeseer, pp. 686–693.

Jerry A Fodor and Zenon W Pylyshyn (1988). "Connectionism and cognitive architecture: A critical analysis." In: *Cognition* 28.1-2, pp. 3–71.

Mikel L Forcada and Ramón P Ñeco (1997). "Recursive hetero-associative memories for translation." In: *International Work-*

Conference on Artificial Neural Networks. Springer, pp. 453–462.

Stefan L Frank, Rens Bod, and Morten H Christiansen (2012). "How hierarchical is language use?" In: *Proceedings of the Royal Society B: Biological Sciences* 279.1747, pp. 4522–4531.

Jerome H Friedman, Bogdan E Popescu, et al. (2008). "Predictive learning via rule ensembles." In: *The Annals of Applied Statistics* 2.3, pp. 916–954.

Mercedes Garcia-Martinez, Loïc Barrault, and Fethi Bougares (2016). "Factored Neural Machine Translation Architectures." In: *International Workshop on Spoken Language Translation (IWSLT'16)*. Seattle, United States.

Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin (2016). "A Convolutional Encoder Model for Neural Machine Translation." In: *CoRR* abs/1611.02344.

Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin (July 2017a). "A Convolutional Encoder Model for Neural Machine Translation." In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 123–135.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin (2017b). "Convolutional sequence to sequence learning." In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. JMLR, pp. 1243–1252.

Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins (Oct. 2000). "Learning to Forget: Continual Prediction with LSTM." In: *Neural Computation* 12.10, pp. 2451–2471.

C Lee Giles, Guo-Zheng Sun, Hsing-Hen Chen, Yee-Chun Lee, and Dong Chen (1990). "Higher order recurrent networks and grammatical inference." In: *Advances in neural information processing systems*, pp. 380–387.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl (2017). "Neural Message Passing for Quantum Chemistry." In: *Proceedings of the 34th International Conference on Machine Learning, ICML*. Sydney, Australia, pp. 1263–1272.

Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal (2018). "Explaining Explanations: An Overview of Interpretability of Machine Learning." In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, pp. 80–89.

Peter W. Glynn (Oct. 1990). “Likelihood Ratio Gradient Estimation for Stochastic Systems.” In: *Commun. ACM* 33.10, pp. 75–84.

Alex Graves (2013). “Generating sequences with recurrent neural networks.” In: *arXiv preprint arXiv:1308.0850*.

Alex Graves and Jürgen Schmidhuber (2005). “Framewise phoneme classification with bidirectional LSTM and other neural network architectures.” In: *Neural Networks* 18.5, pp. 602–610.

Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih (2015). “MuProp: Unbiased backpropagation for stochastic neural networks.” In: *International Conference on Learning Representations*.

Markus Harva and Ata Kaban (2005). “A variational Bayesian method for rectified factor analysis.” In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 1. IEEE, pp. 185–190.

Kazuma Hashimoto and Yoshimasa Tsuruoka (Sept. 2017b). “Neural Machine Translation with Source-Side Latent Graph Parsing.” In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 125–135.

— (2017a). “Neural Machine Translation with Source-Side Latent Graph Parsing.” In: *CoRR* abs/1702.02265.

Simon Haykin (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Tamir Hazan, George Papandreou, and Daniel Tarlow (2016). *Perturbations, Optimization, and Statistics*. MIT Press.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). “Deep residual learning for image recognition.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Jindřich Helcl and Jindřich Libovický (2017). “Neural Monkey: An Open-source Tool for Sequence Learning.” In: *The Prague Bulletin of Mathematical Linguistics* 107, pp. 5–17.

Aurélie Herbelot and Eva Maria Vecchi (2015). “Building a shared world: mapping distributional to model-theoretic semantic spaces.” In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 22–32.

John Hewitt and Christopher D. Manning (June 2019). “A Structural Probe for Finding Syntax in Word Representations.” In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Hu-*

man Language Technologies, Volume 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4129–4138.

Sepp Hochreiter and Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural computation* 9.8, pp. 1735–1780.

Jeremy Howard and Sebastian Ruder (2018). "Universal Language Model Fine-tuning for Text Classification." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 328–339.

Phu Mon Htut, Kyunghyun Cho, and Samuel Bowman (2018). "Grammar Induction with Neural Language Models: An Unusual Replication." In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 4998–5003.

Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema (2018). "Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure." In: *Journal of Artificial Intelligence Research* 61, pp. 907–926.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada (Oct. 2010). "Automatic Evaluation of Translation Quality for Distant Language Pairs." In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, pp. 944–952.

Alon Jacovi and Yoav Goldberg (2020). *Towards Faithfully Interpretable NLP Systems: How should we define and evaluate faithfulness?* arXiv: 2004.03685 [cs.CL].

Eric Jang, Shixiang Gu, and Ben Poole (2017). "Categorical Reparameterization with Gumbel-Softmax." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Toulon, France.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio (July 2015). "On Using Very Large Target Vocabulary for Neural Machine Translation." In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1–10.

Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight (Dec. 2012). "Semantics-Based Machine Translation with Hyperedge Replacement Grammars." In: *Proceedings of COLING 2012*. Mumbai, India: The COLING 2012 Organizing Committee, pp. 1359–1376.

Michael I. Jordan, Zoubin Ghahramani, TommiS. Jaakkola, and LawrenceK. Saul (1999). "An Introduction to Variational Methods for Graphical Models." In: *Machine Learning* 37.2, pp. 183–233.

Łukasz Kaiser and Samy Bengio (2016). "Can active memory replace attention?" In: *Advances in Neural Information Processing Systems*, pp. 3781–3789.

Nal Kalchbrenner and Phil Blunsom (2013). "Recurrent Continuous Translation Models." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA, pp. 1700–1709.

Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley (2016). "Molecular graph convolutions: moving beyond fingerprints." In: *Journal of computer-aided molecular design* 30.8, pp. 595–608.

Daniel Keysers et al. (2020). "Measuring Compositional Generalization: A Comprehensive Method on Realistic Data." In: *International Conference on Learning Representations*.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush (2017). "Structured Attention Networks." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Toulon, France.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis (June 2019). "Unsupervised Recurrent Neural Network Grammars." In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 1105–1117.

Diederik P. Kingma and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. San Diego, USA.

Diederik P. Kingma and Max Welling (2014). "Auto-Encoding Variational Bayes." In: *International Conference on Learning Representations (ICLR)*. Banff, Canada.

Diederik P. Kingma and Max Welling (June 2019). "An Introduction to Variational Autoencoders." In: *arXiv e-prints*, arXiv:1906.02691, arXiv:1906.02691. arXiv: 1906.02691 [cs.LG].

Thomas N. Kipf and Max Welling (2016). "Semi-Supervised Classification with Graph Convolutional Networks." In: *CoRR* abs/1609.02907.

Philipp Koehn, Franz J. Och, and Daniel Marcu (2003). "Statistical Phrase-Based Translation." In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 127–133.

Taku Kudo (July 2018). "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 66–75.

Ponnambalam Kumaraswamy (1980). "A generalized probability density function for double-bounded random processes." In: *Journal of Hydrology* 46.1-2, pp. 79–88.

Brenden M. Lake and Marco Baroni (2018). "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks." In: *International Conference on Machine Learning (ICML)*.

Phong Le and Willem Zuidema (June 2015). "Compositional Distributional Semantics with Long Short Term Memory." In: *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*. Denver, Colorado: Association for Computational Linguistics, pp. 10–19.

Tao Lei, Regina Barzilay, and Tommi Jaakkola (2016). "Rationalizing Neural Predictions." In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 107–117.

Pavel Levin, Nishikant Dhanuka, and Maxim Khalilov (2017). "Machine translation at Booking.com: Journey and lessons learned." In: *arXiv preprint arXiv:1707.07911*.

Jiwei Li, Will Monroe, and Dan Jurafsky (2016a). "Understanding Neural Networks through Representation Erasure." In: *CoRR* abs/1612.08220. arXiv: 1612.08220.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel (2016b). "Gated graph sequence neural networks." In: *International Conference on Learning Representations*.

Jindřich Libovický and Jindřich Helcl (July 2017). “Attention Strategies for Multi-Source Sequence-to-Sequence Learning.” In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 196–202.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg (2016). “Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies.” In: *Transactions of the Association for Computational Linguistics* 4, pp. 521–535.

Zachary Chase Lipton (2016). “The Mythos of Model Interpretability.” In: *ICML Workshop on Human Interpretability in Machine Learning (WHI 2016)*.

Adam Liska, Germán Kruszewski, and Marco Baroni (2018). “Memorize or generalize? Searching for a compositional RNN in a haystack.” In: *CoRR* abs/1802.06467. arXiv: 1802.06467.

Ding Liu and Daniel Gildea (Aug. 2010). “Semantic Role Features for Machine Translation.” In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Beijing, China: Coling 2010 Organizing Committee, pp. 716–724.

Yang Liu and Mirella Lapata (2018). “Learning Structured Text Representations.” In: *Transactions of the Association for Computational Linguistics* 6, pp. 63–75.

Christos Louizos, Max Welling, and Diederik P. Kingma (2018). “Learning Sparse Neural Networks through L_0 Regularization.” In: *International Conference on Learning Representations*.

João Loula, Marco Baroni, and Brenden Lake (Nov. 2018). “Rearranging the Familiar: Testing Compositional Generalization in Recurrent Networks.” In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, pp. 108–114.

Scott M Lundberg and Su-In Lee (2017). “A Unified Approach to Interpreting Model Predictions.” In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 4765–4774.

Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser (2015a). “Multi-task Sequence to Sequence Learning.” In: *CoRR* abs/1511.06114.

Thang Luong, Ilya Sutskever, Quoc Le, Oriol Vinyals, and Wojciech Zaremba (July 2015b). “Addressing the Rare Word

Problem in Neural Machine Translation." In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 11–19.

Thang Luong, Hieu Pham, and Christopher D. Manning (Sept. 2015c). "Effective Approaches to Attention-based Neural Machine Translation." In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1412–1421.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh (2017). "The Concrete Distribution: A Continous Relaxation of Discrete Random Variables." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Toulon, France.

Jean Maillard and Stephen Clark (July 2018). "Latent Tree Learning with Differentiable Parsers: Shift-Reduce Parsing and Chart Parsing." In: *Proceedings of the Workshop on the Relevance of Linguistic Structure in Neural Architectures for NLP*. Melbourne, Australia: Association for Computational Linguistics, pp. 13–18.

Christopher D. Manning (2015). "Computational Linguistics and Deep Learning." In: *Computational Linguistics* 41.4, pp. 701–707. eprint: https://doi.org/10.1162/COLI_a_00239.

Diego Marcheggiani and Ivan Titov (2017). "Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling." In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1506–1515.

Diego Marcheggiani, Anton Frolov, and Ivan Titov (Aug. 2017). "A Simple and Accurate Syntax-Agnostic Neural Model for Dependency-based Semantic Role Labeling." In: *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*. Vancouver, Canada: Association for Computational Linguistics, pp. 411–420.

Diego Marcheggiani, Jasmijn Bastings, and Ivan Titov (2018). "Exploiting Semantics in Neural Machine Translation with Graph Convolutional Networks." In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 486–492.

Andre Martins and Ramon Astudillo (2016). "From softmax to sparsemax: A sparse model of attention and multi-label classification." In: *International Conference on Machine Learning*, pp. 1614–1623.

Julian McAuley, Jure Leskovec, and Dan Jurafsky (2012). "Learning attitudes and attributes from multi-aspect reviews." In: *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE, pp. 1020–1025.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean (2013). "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*, pp. 3111–3119.

Tomas Mikolov, Armand Joulin, and Marco Baroni (2016). "A roadmap towards machine intelligence." In: *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, pp. 29–61.

Christoph Molnar (2019). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/.

Maria Nădejde, Siva Reddy, Rico Sennrich, Tomasz Dwojak, Marcin Junczys-Dowmunt, Philipp Koehn, and Alexandra Birch (Sept. 2017). "Predicting Target Language CCG Supertags Improves Neural Machine Translation." In: *Proceedings of the Second Conference on Machine Translation*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 68–79.

Toshiaki Nakazawa, Manabu Yaguchi, Kiyotaka Uchimoto, Masao Utiyama, Eiichiro Sumita, Sadao Kurohashi, and Hitoshi Isahara (May 2016). "ASPEC: Asian Scientific Paper Excerpt Corpus." In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2016)*. Portorož, Slovenia: European Language Resources Association (ELRA), pp. 2204–2208. ISBN: 978-2-9517408-9-1.

Eric Nalisnick and Padhraic Smyth (2016). "Stick-breaking variational autoencoders." In: *arXiv preprint arXiv:1605.06197*.

Ramon P Neco and Mikel L Forcada (1997). "Asynchronous translations with recurrent neural nets." In: *Proceedings of International Conference on Neural Networks (ICNN'97)*. Vol. 4. IEEE, pp. 2535–2540.

Vlad Niculae and Mathieu Blondel (2017). "A regularized framework for sparse and structured neural attention." In: *Advances in Neural Information Processing Systems*, pp. 3338–3348.

Vlad Niculae, André F. T. Martins, and Claire Cardie (2018). "Towards Dynamic Computation Graphs via Sparse Latent Structure." In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 905–911.

Franz Josef Och and Hermann Ney (2004). "The Alignment Template Approach to Statistical Machine Translation." In: *Computational Linguistics* 30.4, pp. 417–449.

Martha Palmer, Daniel Gildea, and Paul Kingsbury (2005). "The proposition bank: An annotated corpus of semantic roles." In: *Computational linguistics* 31.1, pp. 71–106.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu (2002). "BLEU: a method for automatic evaluation of machine translation." In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pp. 311–318.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit (2016). "A Decomposable Attention Model for Natural Language Inference." In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 2249–2255.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks." In: *International conference on machine learning*, pp. 1310–1318.

Hao Peng, Sam Thomson, and Noah A. Smith (2018). "Backpropagating through Structured Argmax using a SPIGOT." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1863–1873.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global Vectors for Word Representation." In: *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018). "Deep Contextualized Word Representations." In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237.

Steven Phillips (1998). "Are feedforward and recurrent networks systematic? Analysis and implications for a connectionist cognitive architecture." In: *Connection Science* 10.2, pp. 137–160.

Nina Poerner, Hinrich Schütze, and Benjamin Roth (July 2018). "Evaluating neural network explanation methods using hybrid documents and morphosyntactic agreement." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 340–350.

Jordan B Pollack (1990). "Recursive distributed representations." In: *Artificial Intelligence* 46.1-2, pp. 77–105.

Maja Popović (Sept. 2015). "chrF: character n-gram F-score for automatic MT evaluation." In: *Proceedings of the Tenth Workshop on Statistical Machine Translation*. Lisbon, Portugal: Association for Computational Linguistics, pp. 392–395.

Matt Post (Oct. 2018). "A Call for Clarity in Reporting BLEU Scores." In: *Proceedings of the Third Conference on Machine Translation: Research Papers*. Belgium, Brussels: Association for Computational Linguistics, pp. 186–191.

Alessandro Raganato, Yves Scherrer, and Jörg Tiedemann (2020). *Fixed Encoder Self-Attention Patterns in Transformer-Based Machine Translation*. arXiv: 2002.10260 [cs.CL].

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba (2016). "Sequence level training with recurrent neural networks." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Danilo J. Rezende and Fabio Viola (Oct. 2018). "Taming VAEs." In: *arXiv e-prints*, arXiv:1810.00597, arXiv:1810.00597. arXiv: 1810.00597 [stat.ML].

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra (2014). "Stochastic Backpropagation and Approximate Inference in Deep Generative Models." In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pp. 1278–1286.

Marco Ribeiro, Sameer Singh, and Carlos Guestrin (2016). ""Why Should I Trust You?": Explaining the Predictions of Any Classifier." In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. San Diego, California: Association for Computational Linguistics, pp. 97–101.

Herbert Robbins and Sutton Monro (Sept. 1951). “A Stochastic Approximation Method.” In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407.

Paul Rodriguez and Janet Wiles (1998). “Recurrent neural networks can learn to implement symbol-sensitive counting.” In: *Advances in Neural Information Processing Systems*, pp. 87–93.

Jason Tyler Rolfe (2017). “Discrete variational autoencoders.” In: *ICLR*.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake (2020). *A Benchmark for Systematic Generalization in Grounded Language Understanding*. arXiv: `2003.05161 [cs.CL]`.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling (2018). “Modeling relational data with graph convolutional networks.” In: *European Semantic Web Conference*. Springer, pp. 593–607.

Mike Schuster and Kuldip K. Paliwal (Nov. 1997). “Bidirectional recurrent neural networks.” In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681.

Holger Schwenk and Jean-Luc Gauvain (2002). “Connectionist language modeling for large vocabulary continuous speech recognition.” In: *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 1. IEEE, pp. I–765.

Rico Sennrich and Barry Haddow (2016). “Linguistic Input Features Improve Neural Machine Translation.” In: *Proceedings of the First Conference on Machine Translation (WMT16)*. Vol. abs/1606.02892. Berlin, Germany.

Rico Sennrich, Barry Haddow, and Alexandra Birch (Aug. 2016a). “Edinburgh Neural Machine Translation Systems for WMT 16.” In: *Proceedings of the First Conference on Machine Translation*. Berlin, Germany: Association for Computational Linguistics, pp. 371–376.

— (Aug. 2016b). “Neural Machine Translation of Rare Words with Subword Units.” In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1715–1725.

Lloyd S Shapley (1953). "A value for n-person games." In: *Contributions to the Theory of Games* 2.28, pp. 307–317.

Yikang Shen, Zhouhan Lin, Chin wei Huang, and Aaron Courville (2018). "Neural Language Modeling by Jointly Learning Syntax and Lexicon." In: *International Conference on Learning Representations*.

David Smith and Jason Eisner (June 2006). "Quasi-Synchronous Grammars: Alignment by Soft Projection of Syntactic Dependencies." In: *Proceedings on the Workshop on Statistical Machine Translation*. New York City: Association for Computational Linguistics, pp. 23–30.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul (2006). "A study of translation edit rate with targeted human annotation." In: *In Proceedings of Association for Machine Translation in the Americas*, pp. 223–231.

Nicholas D. Socci, Daniel D. Lee, and H. Sebastian Seung (1998). "The Rectified Gaussian Distribution." In: *Advances in Neural Information Processing Systems 10*. Ed. by M. I. Jordan, M. J. Kearns, and S. A. Solla. MIT Press, pp. 350–356.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning (July 2011). "Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions." In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, pp. 151–161.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts (Oct. 2013). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 1631–1642.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea (July 2018). "A Graph-to-Sequence Model for AMR-to-Text Generation." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1616–1626.

Felix Stahlberg, Eva Hasler, Aurelien Waite, and Bill Byrne (Aug. 2016). "Syntactically Guided Neural Machine Translation." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 299–305.

Miloš Stanojević and Khalil Sima'an (June 2014). "BEER: BEtter Evaluation as Ranking." In: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, pp. 414–419.

Miloš Stanojević and Khalil Sima'an (2015). "Evaluating MT systems with BEER." In: *The Prague Bulletin of Mathematical Linguistics* 104.1, pp. 17–26.

Miloš Stanojević and Khalil Sima'an (Oct. 2014). "Fitting Sentence Level Translation Evaluation with Many Dense Features." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 202–206.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum (2018). "Linguistically-Informed Self-Attention for Semantic Role Labeling." In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 5027–5038.

Erik Štrumbelj and Igor Kononenko (2014). "Explaining prediction models and individual predictions with feature contributions." In: *Knowledge and information systems* 41.3, pp. 647–665.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre (Aug. 2008). "The CoNLL 2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies." In: *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*. Manchester, England: Coling 2008 Organizing Committee, pp. 159–177.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le (2014). "Sequence to Sequence Learning with Neural Networks." In: *Neural Information Processing Systems (NIPS)*. Montreal, Quebec, Canada, pp. 3104–3112.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning (2015). "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks." In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1556–1566.

Aleš Tamchyna, Marion Weller-Di Marco, and Alexander Fraser (Sept. 2017). “Modeling Target-Side Inflection in Neural Machine Translation.” In: *Proceedings of the Second Conference on Machine Translation*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 32–42.

Ian Tenney, Dipanjan Das, and Ellie Pavlick (July 2019a). “BERT Rediscovers the Classical NLP Pipeline.” In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 4593–4601.

Ian Tenney et al. (2019b). “What do you learn from context? Probing for sentence structure in contextualized word representations.” In: *International Conference on Learning Representations*.

Sebastian Thrun (1995). “Extracting rules from artificial neural networks with distributed representations.” In: *Advances in neural information processing systems*, pp. 505–512.

Ivan Titov and Ryan McDonald (June 2008). “A Joint Model of Text and Aspect Ratings for Sentiment Summarization.” In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, pp. 308–316.

Ke Tran and Yonatan Bisk (2018). *Inducing Grammars with and for Neural Machine Translation*.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein (2017). “REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models.” In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 2624–2633.

KP Unnikrishnan and Kootala P Venugopal (1994). “Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks.” In: *Neural Computation* 6.3, pp. 469–490.

Eva Vanmassenhove and Andy Way (July 2018). “SuperNMT: Neural Machine Translation with Semantic Supersenses and Syntactic Supertags.” In: *Proceedings of ACL 2018, Student Research Workshop*. Melbourne, Australia: Association for Computational Linguistics, pp. 67–73.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). “Attention is all you need.” In: *Advances in Neural Information Processing Systems*, pp. 6000–6010.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio (2018). "Graph Attention Networks." In: *International Conference on Learning Representations*.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol (2010). "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." In: *Journal of machine learning research* 11.Dec, pp. 3371–3408.

Elena Voita and Ivan Titov (2020). *Information-Theoretic Probing with Minimum Description Length*. arXiv: 2003.12298 [cs.CL].

Warren Weaver (1949). "The mathematics of communication." In: *Scientific American* 181.1, pp. 11–15.

— (1955). "Translation." In: *Machine translation of languages* 14, pp. 15–23.

Gail Weiss, Yoav Goldberg, and Eran Yahav (2018). "On the practical computational power of finite precision RNNs for language recognition." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics.

Paul J Werbos (1982). "Applications of advances in nonlinear sensitivity analysis." In: *System modeling and optimization*. Springer, pp. 762–770.

Adina Williams, Andrew Drozdov, and Samuel R. Bowman (2018). "Do latent tree learning models identify meaningful structure in sentences?" In: *Transactions of the Association for Computational Linguistics* 6, pp. 253–267.

Ronald J Williams (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* 8.3-4, pp. 229–256.

John Winn and Christopher M Bishop (2005). "Variational message passing." In: *Journal of Machine Learning Research* 6.Apr, pp. 661–694.

Francis CK Wong and William SY Wang (2007). "Generalisation towards combinatorial productivity in language acquisition by simple recurrent networks." In: *Integration of Knowledge Intensive Multi-Agent Systems, 2007. KIMAS 2007. International Conference on*. IEEE, pp. 139–144.

Dekai Wu and Pascale Fung (June 2009). "Semantic Roles for SMT: A Hybrid Two-Pass Model." In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the*

North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers. Boulder, Colorado: Association for Computational Linguistics, pp. 13–16.

Yonghui Wu et al. (2016a). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." In: *CoRR* abs/1609.08144. arXiv: 1609.08144.

— (2016b). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." In: *CoRR* abs/1609.08144.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling (2017). "Learning to Compose Words into Sentences with Reinforcement Learning." In: *International Conference on Learning Representations (ICLR)*. Toulon, France.

Mo Yu, Shiyu Chang, Yang Zhang, and Tommi Jaakkola (Nov. 2019). "Rethinking Cooperative Rationalization: Introspective Extraction and Complement Control." In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, pp. 4094–4103.

Omar Zaidan and Jason Eisner (Oct. 2008). "Modeling Annotators: A Generative Approach to Learning from Annotator Rationales." In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Honolulu, Hawaii: Association for Computational Linguistics, pp. 31–40.

Omar Zaidan, Jason Eisner, and Christine Piatko (2007). "Using "Annotator Rationales" to Improve Machine Learning for Text Categorization." In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*. Rochester, New York: Association for Computational Linguistics, pp. 260–267.

Ye Zhang, Iain Marshall, and Byron C. Wallace (Nov. 2016). "Rationale-Augmented Convolutional Neural Networks for Text Classification." In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 795–804.

Yuhao Zhang, Peng Qi, and Christopher D. Manning (2018). "Graph Convolution over Pruned Dependency Trees Improves Relation Extraction." In: *Proceedings of the 2018 Conference* *on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 2205–2215.

Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo (2015). "Long short-term memory over recursive structures." In: *International Conference on Machine Learning*, pp. 1604–1612.

Andreas Zollmann and Ashish Venugopal (2006). "Syntax Augmented Machine Translation via Chart Parsing." In: *Proceedings of the Workshop on Statistical Machine Translation*. StatMT '06. New York City, New York: Association for Computational Linguistics, pp. 138–141.

ABSTRACT

A Tale of Two Sequences: Interpretable and Linguistically-Informed Deep Learning for Natural Language Processing

Deep Learning (DL) has swiftly taken over our field of NLP. It caused a shift from exploiting linguistic features and structures, such as POS-tags, dependency and constituency trees, to relying solely on the input words, and treating a sentence as a mere *sequence* of words. As performance records in NLP benchmarks keep being broken, we can ask ourselves: are linguistic structures now obsolete? Is there still a way to make use of them?

In the first part of this thesis, we try to answer these questions in the context of machine translation. We find that we can exploit a Graph Convolutional Network (GCN) to condition a neural machine translation model on linguistic input structures, and we show empirically that we can gain performance improvements while conditioning on syntactic dependency structures, semantic role labeling structures, and both. In addition to conditioning on explicit linguistic structure, we also investigate if we can *induce* structure in a machine translation setting. We find that it is possible to learn useful structure on top of word embeddings and CNN representations, while obtaining trivial (mostly diagonal) structure on top of LSTM representations. This latent structure is related to the now popular Transformer model, which can be seen as performing graph convolution over dense graphs.

In the second part of the thesis, we look at two common criticisms of neural networks: (1) their lack of interpretability, and (2) their hunger for labeled data to generalize well. We first study neural text classifiers, and make them interpretable by having them provide an explanation, a rationale, for their predictions. This is done by showing exactly which part of the input text is used for classification, rendering the model more transparent than a model that does not provide a rationale. We show that our method is more aligned with human rationales than previous work. Finally, we investigate generalization of neural networks. In particular, we look at the SCAN benchmark and find obtaining a high score does not have to imply strong

generalization behavior, due to the simple nature of the data set. We propose a remedy for this problem in the form of the NACS data set.

SAMENVATTING

Een Verhaal over Twee Sequenties: Interpreteerbare en Taalkundig-geïnformeerde Deep Learning voor Natuurlijke Taalverwerking

Deep Learning (DL) heeft het vakgebied van natuurlijke taalverwerking (NLP) abrupt overgenomen. Het veroorzaakte een verschuiving van het benutten van taalkundige kenmerken en structuren, zoals POS-tags, dependentiebomen en syntaxisbomen, naar het uitsluitend gebruikmaken van woorden, en het behandelen van een zin als niets anders dan een opeenvolging van woorden. Aangezien prestatierecords in NLP-benchmarks keer op keer worden verbroken, kunnen we ons afvragen: zijn taalkundige structuren nu achterhaald? Of is er toch nog een manier om er gebruik van te maken?

In het eerste deel van dit proefschrift proberen we deze vragen te beantwoorden in de context van automatische vertaling. We zien dat we een Graph Convolutional Network (GCN) kunnen gebruiken om een neuraal vertalingsmodel op taalkundige structuren te conditioneren, en we laten empirisch zien dat we prestatieverbeteringen kunnen behalen met het conditioneren op dependentiestructuren, semantische structuren, en beide. Bovenop het conditioneren op expliciete taalkundige structuren, onderzoeken we ook of we structuur kunnen *induceren* met een automatisch vertalingsmodel. We zien dat het mogelijk is om nuttige structuur te leren bovenop woordembeddings en CNN representaties, terwijl we triviale (veelal diagonale) structuur krijgen bovenop LSTM representaties. Deze latente structuur is gerelateerd aan het inmiddels populaire Transformer model, dat gezien kan worden als het toepassen van graph convolution over een volledige graaf.

In het tweede deel van het proefschrift nemen we een kijk op twee kritiekpunten van neurale netwerken: (1) hun gebrek aan interpreteerbaarheid, en (2) hun honger naar geannoteerde data om goed te generaliseren. Eerst bestuderen we neurale textclassificatiemodellen, die we interpreteerbaar maken door hen een uitleg, een rationalisering, te laten geven voor hun predicties. Dit doen we door te laten zien welke delen van de invoer worden gebruikt voor classificatie, en daardoor een model te creëren dat transparanter is dan een model dat geen uitleg geeft. We laten zien dat onze methode beter aansluit bij de uitleg die mensen geven dan eerdere methodes. Tenslotte onderzoeken we generalisatie van neurale netwerken. In het bijzonder kijken we naar de SCAN benchmark, en zien we dat het behalen van een hoge score geen sterke generalisatie hoeft te betekenen, vanwege de eenvoud van de dataset. We komen met een simpele oplossing voor dit probleem in de vorm van de NACS dataset.