UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Modeling Structure with Deep Neural Networks for Natural Language Processing

Tsvetomila Borisova Mihaylova

Supervisor: Doctor André Filipe Torres Martins

Co-Supervisor: Doctor Vlad Niculae

Thesis approved in public session to obtain the PhD Degree in
Electrical and Computer Engineering

Jury final classification: Pass with Distinction

2022

TÉCNICO LISBOA

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Modeling Structure with Deep Neural Networks for Natural Language Processing

Tsvetomila Borisova Mihaylova

Supervisor: Doctor André Filipe Torres Martins

Co-Supervisor: Doctor Vlad Niculae

Thesis approved in public session to obtain the PhD Degree in
Electrical and Computer Engineering

Jury final classification: Pass with Distinction

Jury

2022

Abstract

Many natural language processing (NLP) problems have underlying structure, which expresses relations and constraints. The majority of the modern approaches to solving NLP problems rely on large pretrained language models, which in many cases serve as monolith black-boxes and do not allow the practitioner to be aware of the underlying structure.

In this thesis, we propose new models, interpret and combine existing approaches related to modeling and predicting structure in language in deep learning models. We experiment with several natural language processing tasks, such as machine translation, natural language inference, sentiment classification and dependency parsing. We address structure as a model output, as a latent variable in the middle of the model, and we propose a new model which allows flexible ways of modeling relations between variables.

First, we address an important limitation of auto-regressive sequence-to-sequence models, exposure bias: at training time, models maximize the likelihood of the next word given the gold target prefix, but at inference time, they condition on their own previous predictions, which may lead to error propagation. To avoid this, we propose adapting a technique, scheduled sampling, to transformer-based models.

Then, we address modeling structure with discrete latent variable models. A challenge with these models is that they often require computing an arg-max for the latent structure, but this operation has null gradient, precluding the use of the gradient backpropagation for training the model end to end. We propose a family of structured straight-through gradient methods based on the SPIGOT algorithm, developing a framework which allows designing new surrogate gradient methods based on the observations.

Lastly, we propose undirected neural networks – a new energy-based model which combines the strengths of factor graphs and neural networks, allowing different directions and orders of computation. We show how undirected neural networks subsume many existing architectures. We prove that any feed-forward neural network can be presented as an undirected neural network and we demonstrate the effectiveness of undirected neural networks with specific examples on several problems involving language and vision.

Keywords: machine learning, structure, structured prediction, neural networks, natural language processing, latent structures, scheduled sampling, factor graphs, modularity

Resumo

Muitos problemas de processamento de linguagem natural (PLN) têm estrutura subjacente, que expressa relações e restrições. A maioria das abordagens existentes emprega modelos de linguagem pré-treinados com uma grande quantidade de parâmetros, consistindo em caixas pretas monolíticas que escondem do utilizador a estrutura subjacente ao problema.

Nesta tese, propomos novos modelos que generalizam e combinam várias abordagens para modelação e previsão de estrutura em linguagem, empregando modelos de aprendizagem profunda. Experimentamos com os modelos propostos em várias tarefas de processamento de linguagem natural, incluindo tradução automática, inferência em linguagem natural, classificação de sentimentos e análise sintáctica de dependências. Abordamos a estrutura de duas formas: como uma saída do modelo e como uma variável latente intermédia, culminando num novo modelo que permite formas flexíveis de modelizar as relações entre as variáveis.

Em primeiro lugar, abordamos uma importante limitação dos modelos autoregressivos para sequências, o *viés de exposição*: durante o treino, os modelos maximizam a probabilidade da próxima palavra dado o prefixo de referência, porém, depois de treinados e durante o processo de inferência, eles condicionam as previsões ao prefixo que o próprio modelo gerou, o que pode levar à propagação de erros. Para evitar este problema, adaptamos uma técnica, amostragem programada, para modelos baseados em transformadores.

Em seguida, abordamos a estrutura de modelação para modelos com variáveis latentes discretas. Um desafio com estes modelos é que normalmente exigem o cálculo de um maximizador para a estrutura latente, uma operação que apresenta gradiente nulo, impossibilitando o uso do algoritmo da retropropagação do gradiente para treinar o modelo de ponta a ponta. Para colmatar esta lacuna, propomos uma família de métodos de gradientes directos estruturados baseados no algoritmo SPIGOT.

Por fim, propomos redes neuronais não direccionadas – um novo modelo baseado em funções de energia que combina os pontos fortes dos gráficos de factores e das redes neuronais, permitindo diferentes direcções e ordens de computação. Mostramos como as redes neuronais não direccionadas incluem várias arquiteturas existentes e provamos que qualquer rede neuronal "feed-forward" pode ser representada como uma rede neural não direccionada. Demonstramos a eficácia das redes neuronais não direccionadas emdiversos problemas envolvendo linguagem e visão.

Palavras-Chave: aprendizagem automática, estrutura, previsão estruturada, redes neuronais, processamento de linguagem natural

Acknowledgements

Going through my PhD has been such a challenging journey and I am grateful I have been around so many people who have supported me through it.

I am very grateful to my supervisors André Martins and Vlad Niculae and I hope to some day be able to pay all their support forward. I thank Andre for believing in me at the beginning of this PhD, for not giving up on me when I was not at my best, for giving me the space to work on things that were hard and it was not clear whether they would turn into a success. Also, I thank him for putting together a team of such amazing people. I thank Vlad for being always available and supportive, for explaining different concepts in accessible ways, for showing by example that people matter much more than results and that teaching and supervision can be inclusive and professional at the same time.

If it had not been for the people in my master studies in Sofia University, I probably would not have even applied for this PhD. I am grateful to my master thesis supervisors Preslav Nakov and Ivan Koychev. I thank Preslav for giving so much to the Bulgarian NLP community, for encouraging me to do research and publish papers even though I had no experience, for being so friendly, supportive and a really inspiring mentor. I thank Prof. Koychev for all the support during and after my master studies and for giving me the opportunity to teach in the university. I thank Ivelina Nikolova for supervising me and my colleagues when we participated in our first shared task. I really enjoyed the discussions and running the endless experiments and when she told me that this is what PhD students do all day, I decided I wanted to pursue a PhD. I also thank my other colleagues from the university – Pepa, Georgi, Yasen, Martin, Vladislav – with whom I have written my first papers, participated in NLP competitions, have had interesting and thought-provoking discussions and who I am always happy to talk to and meet at a conference somewhere in the world.

Before my PhD, I worked as a software developer for ten years and during that time, I have had the opportunity to learn, to travel and to work with great people – mentors and peers – who also became my friends. I thank Todor, Marian Vigenin, Marian Mitov, Evgeni, Dimitar, Stoyan, Daria, Margarita, Valentina, Simeon, Kiril, Bozhidar, Vasil, Lubomir and all my other colleagues throughout the years. I thank the people from ABLE for being such an inspiration.

During my PhD I have been through hard times and I am happy to have had my friends around me. Huge thanks to Krasimira, Ina, Krasimira and Nikolay, Dilyana and Plamen for being there for me when I needed their support. I thank my neighbours from my hometown, Tsvetelina and Krasimira, for spending the last few summers in Bulgaria together.

This PhD would not have been the same without my colleagues from DeepSPIN. They were very supportive when I needed it, we have shared great time during conferences, classes in the university, celebrations. I thank Gonçalo for being my guide in everything Portuguese, especially for showing me the garden of Arco do Cego, Ben – for always been available for a beer, Erick for remembering we had oranges for Christmas in Eastern Europe, Marcos for always being so kind and Pedro for being so down-to-earth. I am also very happy to know and to have had meetings with the rest of the SARDINEs - Taya, Chryssa, Nuno, Patrick, António.

During our stay in Portugal, the people from the Bulgarian school in Lisbon have provided me and my family with great support. I thank Eva, Bobi and Rumyana for taking care of our son every Saturday morning while we were living in Portugal, for celebrating Bulgarian holidays together, for helping me with moving out.

I am immensely grateful for having my family and their love and support. My parents Mariela and Boris have always, unconditionally, been there for me. They have struggled a lot to give me, my brother and my sister a good education. They have taught us to work hard, to be patient and to fight for what we believe is right. I thank my mom for always be ready to take care of my son, especially for the times when she flew to Portugal when I needed her. I thank my brother Todor for the long discussions about our shared interest in programming and for telling me about the master program in Information Retrieval, where my academic journey began. I am so grateful to have my little sister Mihaela, I thank her for so many things – for being such an inspiration, for sharing hard and joyful moments, for being there for me whenever I need someone to talk to. I am grateful for my carefree childhood in my hometown in Northwestern Bulgaria, for my grandparents, my uncle and aunt, for my cousins Gergana and Yanina.

This PhD would not have been possible if my husband Svetlin had not decided to selflessly support me in it by moving to Portugal with me and our one-year-old son. Making it work with a new job in a foreign country with a small child has been quite a challenge and I am really grateful for him being there, for sharing the hard times, for sharing the work at home, for taking care of our son when I was traveling for conferences. This thesis would not have been written if it weren't for him.

I dedicate this thesis to my amazing son Leon who has taught me what are the things in life that really matter. Watching him grow up to becoming such an incredible person is the best thing I have ever experienced and is an infinite source of strength and motivation.

Contents

List of Tables

List of Figures

Notation

Vectors and matrices

a	Vector values
A	Matrix and tensor values
A	Abstract factor graph variables
e_k	One-hot vector with all zeros except in the k^{th} coordinate
1_d	d-dimensional vector of ones (or tensor, if d is a tuple)

Sets and functions

$\mathbb{R}^d_+$	$\{x \in \mathbb{R}^d : x \geq 0\}$	Non-negative real numbers
$\iota_{\mathcal{X}}$	$\iota_{\mathcal{X}}(x) \coloneqq 0$ if $x \in \mathcal{X}$, and $\iota_{\mathcal{X}}(x) \coloneqq +\infty$ otherwise	Indicator function
$\Psi^*(t)$	$\sup_{x\in\mathbb{R}^d} \langle x, t\rangle - \Psi(x)$	Fenchel conjugate of a function $\Psi : \mathbb{R}^d \to \mathbb{R}$
$(\nabla\Psi^*)(t)$	$\operatorname{argmax}_{x\in\mathbb{R}^d} \langle x, t\rangle - \Psi(x)$	The unique maximizer, when Ψ is strictly convex and Ψ^* is differentiable
$\triangle_d$	$\{\alpha \in \mathbb{R}^d_+, \langle 1_d, \alpha\rangle = 1\}$.	$(d-1)$-dimensional simplex
$\Vert A\Vert$	$\sqrt{\langle A, A\rangle}$	Frobenius norm
$\langle A, B\rangle$	$\sum_{i_1=1}^{d_1} \cdots \sum_{i_n=1}^{d_n} a_{i_1\ldots i_n} b_{i_1\ldots i_n}$	Frobenius inner product of two tensors with matching dimensions $A, B \in \mathbb{R}^{d_1\times\ldots\times d_n}$
$\langle a, b\rangle$	$a^\top b$	Frobenius inner product for vectors
$\langle A, B\rangle$	$\operatorname{Tr}(A^\top B)$	Frobenius inner product for matrices
$(A \otimes B)_{i_1,\ldots,i_m,j_1,\ldots,j_n}$	$a_{i_1,\ldots,i_m} b_{j_1,\ldots,j_m}$	Outer product of two tensors $A \in \mathbb{R}^{c_1,\ldots,c_m}, B \in \mathbb{R}^{d_1,\ldots,d_n}$
$a \otimes b$	$ab^\top$	Outer product for vectors
$\mathcal{H}(y)$	$-\sum_i y_i \log y_i$	The Shannon entropy of a discrete distribution $y \in \triangle_d$
$\Pi_{\mathcal{D}}(s)$	$\operatorname{argmin}_{d\in\mathcal{D}} \Vert s - d\Vert_2$	Euclidean projection of s onto a set $\mathcal{D}$
$\mathbb{E}_{z\sim p}[h(z)]$	$\sum_{z\in\mathcal{Z}} p(z)h(z)$	Expectation of a function $h : \mathcal{Z} \to \mathbb{R}^D$ under distribution $p \in \Delta^{\vert\mathcal{Z}\vert}$
$\operatorname{conv}(\mathcal{Z})$	$\left\{\mathbb{E}_{z\sim p}[z] \mid p \in \Delta_{\vert\mathcal{Z}\vert}\right\}$	Convex hull of the (finite) set $\mathcal{Z} \subseteq \mathbb{R}^K$

CHAPTER 1

Introduction

Contents

1.1 Motivation

In recent years, deep learning has led to a breakthrough in natural language processing (NLP) applications. Adding *attention* (Bahdanau et al., 2014) to recurrent neural networks led to better sequence-to-sequence (Sutskever et al., 2014) models. Later, the transformer model (Vaswani et al., 2017) builds an encoder-decoder consisting entirely of attention layers, which significantly improved the performance for many tasks and became a state-of-the-art architecture for NLP. The next leap was the creation of big pretrained transformer-based language models, the first one such model being BERT (Devlin et al., 2019). As of writing this thesis, a big part of NLP practice is roughly related to loading a large pretrained model, fine-tuning it with some existing data, and modeling an output of choice. In most cases, practitioners cannot easily look at what is inside the big language model and treat it as a black box.

The approach of treating the models as monolithic black boxes misses the modeling of linguistic structure. Many interesting NLP problems have underlying structure (Smith, 2011), which expresses relations and constraints. Some examples for NLP tasks with useful structure are machine translation, dependency parsing, word alignment, etc. The structure can be in the input, can be predicted as a model output, or it can be modeled as a latent variable (Fig. 1.1). In some cases, it might make sense for the end task to be broken down into connected subtasks.

In this thesis, we address some of the limitations of the monolith neural network models by taking into account linguistic structure. We propose improvements and provide insights about some aspects of how neural networks function. We make connections between existing concepts to shed light on and improve some aspects of how neural models work. We next describe the main problems addressed in the thesis.

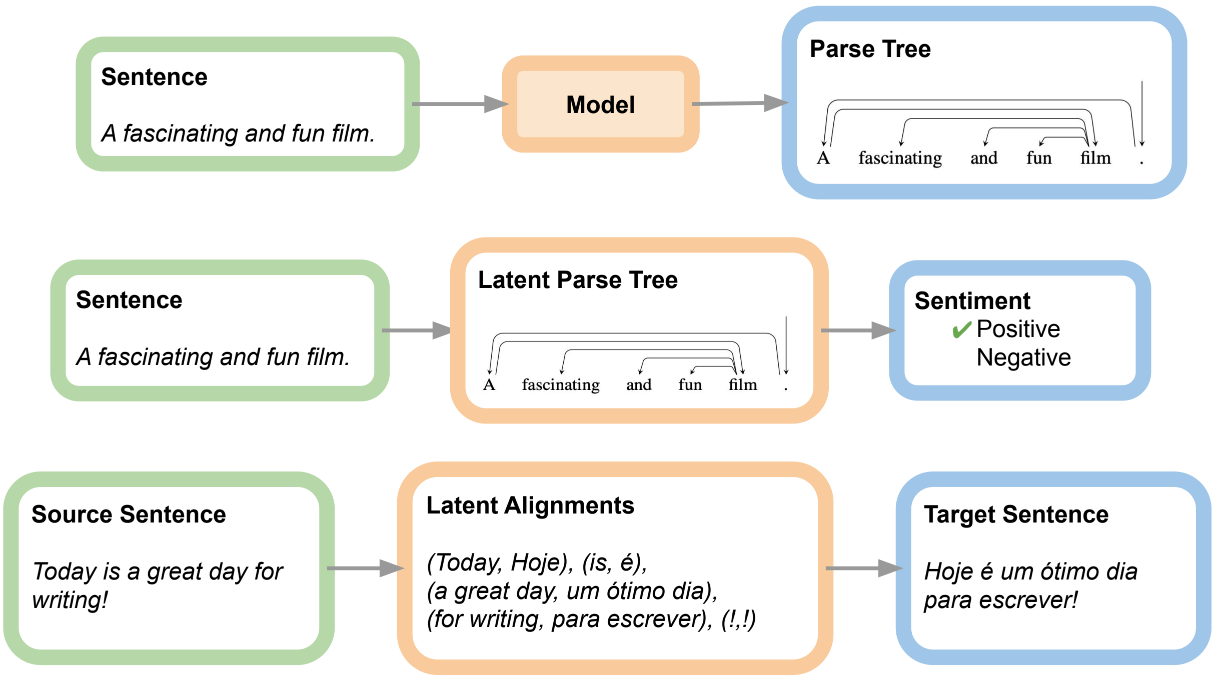

Sentence
A fascinating and fun film.
Model
Parse Tree
A fascinating and fun film .
Sentence
A fascinating and fun film.
Latent Parse Tree
A fascinating and fun film .
Sentiment
✔ Positive
Negative
Source Sentence
Today is a great day for writing!
Latent Alignments
(Today, Hoje), (is, é),
(a great day, um ótimo dia),
(for writing, para escrever), (!,!)
Target Sentence
Hoje é um ótimo dia para escrever!

Figure 1.1: Examples of structure in NLP tasks. Top: dependency parsing as a model output. Middle: dependency parsing as a latent structure in the middle of the model. Bottom: Machine translation with latent word alignments. The structure is in the input and output (sequences) and in the middle of the model (word alignments).

1.2 Related Work and Contributions

1.2.1 Exposure Bias, Scheduled Sampling, and Transformers

Before the adoption of transformers, the state of the art in many NLP tasks, such as machine translation, was based on sequence-to-sequence recurrent neural networks (RNN) with global attention (Sutskever et al., 2014; Bahdanau et al., 2014). These models were typically trained with teacher forcing, i.e. the decoder makes each token prediction conditioned on the preceding elements in the gold target sequence. This differs from the procedure used at inference time, which predicts the next token based on the sequence predicted from the model so far. A problem arising from this type of discrepancy – *exposure bias* – was noticed by Ranzato et al. (2015).

A common approach for addressing the problem with exposure bias is using a scheduled strategy for deciding when to use teacher forcing and when to use the model predictions (Bengio et al., 2015). For a recurrent decoder, applying scheduled sampling works as follows: for generation of each word, the model decides whether to condition on the gold embedding from the given target (teacher forcing) or on the model prediction from the previous step. Bengio et al. (2015) proposed scheduled sampling for sequence-to-sequence RNN models: a method where the embedding used as the input to the decoder at time step $t + 1$ is picked randomly between the gold target and the `argmax` of the model's output probabilities at step t. The Bernoulli probability of picking one or the other changes over training epochs according to a schedule that makes the probability of choosing the gold target decrease across training steps. Goyal et al. (2017) proposed an approach based on scheduled sampling which backpropagates through the model decisions. At each step, when model predictions are used, instead of the `argmax`, they use a weighted average of all word embeddings, weighted by the prediction probabilities. With this technique, they achieve better results than the standard scheduled sampling. Ranzato et al. (2015) took ideas from scheduled sampling and the REINFORCE algorithm (Williams, 1992) and combined the teacher forcing training with optimization of the sequence level loss. In the first epochs, the model is trained with teacher forcing and for the remaining epochs they start with teacher forcing for the first t time steps and then switch to REINFORCE (sampling from the model) until the end of the sequence. They decrease the time for training with teacher forcing t as training continues until the whole sequence is trained with REINFORCE in the final epochs. In addition to the work of Ranzato et al. (2015), other methods that are also focused on sequence-level training are using for example actor-critic (Bahdanau et al., 2016) or beam search optimization (Wiseman and Rush, 2016). These methods directly optimize the metric used at test time (e.g. BLEU). Another proposed approach to avoid exposure bias is SEARN (Daumé et al., 2009). In SEARN, the model uses its own predictions at training time to produce sequence of actions, then a search algorithm determines the optimal action at each step and a policy is trained to predict that action.

In **Chapter 3: Scheduled Sampling for Transformers**, we address the problem of *exposure bias in transformers* and adapt *scheduled sampling to Transformer* models by using two decoders instead of one.

1.2.2 Backpropagation through Discrete Latent Structures

The high-level language tasks modeled with big deep models would benefit from uncovering underlying structures such as trees, sequence tags, or segmentations. The benefits might not necessary be related to improvement on downstream task metrics, such as accuracy, but could be beneficial for interpretability. Before the rise of neural networks for NLP, it was common to use pipeline approaches where an external, pretrained model is used to predict, e.g., syntactic structure. The benefit of this approach is that the predicted tree is readily available for debugging, but the downside is that the errors can propagate throughout the pipeline (Finkel et al., 2006; Sutton and McCallum, 2005; Toutanova, 2005). In contrast, the most common current deep learning approaches do not usually model any underlying structure, which makes the deep neural models harder to interpret. The best of both worlds could be *modeling structure as latent variables*. Since most linguistic structures are discrete, we are interested in modeling discrete latent variables. For example, in sentiment classification, where the sentiment of a given text document needs to be predicted, the document topic (categorical) or the dependency parse tree (structured) could be modeled as a latent variable. The simplest case for a discrete latent variable is a categorical one, as in the example above this is the document topic, which is predicted from a list of predefined topics. An example of such a variable is shown in Fig. 1.2 – the vector s predicts a score for each of five categories $\{1, ..., 5\}$. One category is selected with $z = \text{argmax}(s)$ and is used in further computations.

Discrete latent variable learning is often tackled in **stochastic computation**

graphs by estimating the gradient of an expected loss. An established method is REINFORCE, also known as the score function estimator (SFE) (Glynn, 1990; Williams, 1992; Kleijnen and Rubinstein, 1996). REINFORCE is widely used in NLP, for tasks including minimum risk training in NMT (Shen et al., 2016; Wu et al., 2018) and latent linguistic structure learning (Yogatama et al., 2017; Havrylov et al., 2019). In this thesis, we focus on the alternative strategy of **surrogate gradients**, which doesn't require stochasticity and can be applied in deterministic cases too. Examples are the straight-through estimator (Hinton, 2012; Bengio et al., 2013) and the structured projection of intermediate gradients optimization technique (SPIGOT; Peng et al. 2018). A popular alternative is to **relax** an argmax into a continuous transform such as softmax or sparsemax (Martins and Astudillo, 2016b), as seen for instance in soft attention mechanisms (Bahdanau et al., 2014; Luong et al., 2015; Vaswani et al., 2017), or structured attention networks (Kim et al., 2017; Maillard et al., 2017; Liu and Lapata, 2018; Mensch and Blondel, 2018; Niculae et al., 2018a). In between surrogate gradients and relaxation is **Gumbel softmax**, which uses the Gumbel-max reparametrization to sample from a categorical distribution, applying softmax either to relax the mapping or to induce surrogate gradients (Jang et al., 2017; Maddison et al., 2016). Gumbel-softmax has been successfully applied to latent linguistic structure as well (Choi et al., 2018; Maillard and Clark, 2018; Corro and Titov, 2019a,b).

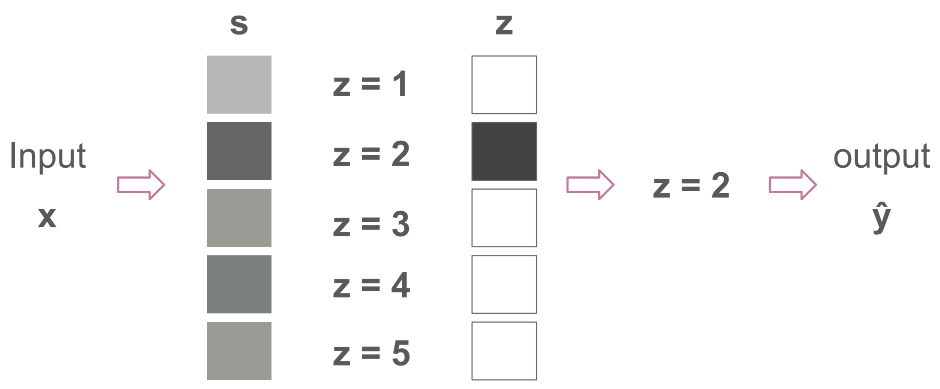


Figure 1.2: Discrete latent categorical variable in the middle of the computation graph.

In **Chapter 4: Surrogate Gradients for Latent Structure Learning**, we address the problem of *backpropagation through discrete latent structures*. We focus on one of the approaches for overcoming this problem – using surrogate gradients. We give an explanation of how SPIGOT (Peng et al., 2018) – a method using *surrogate gradients for structure* works. Based on that explanation, we propose a framework from which we derive modifications of SPIGOT, which we call SPIGOT-CE and SPIGOT-EG.

1.2.3 Structuring Neural Computation and Modularity

Before neural networks became the dominating paradigm, structured prediction problems in vision or natural language processing were often represented as factor graphs (Bakır et al., 2007; Smith, 2011; Nowozin et al., 2014). In recent years, neural networks have become the model of choice for working with these applications. Unlike factor graphs – which emphasize the modularity of the problem – neural networks typically work end-to-end, relying on rich representations captured at the encoder level (often pretrained), which are then propagated to a task-specific decoder and are usually monolithic mappings from inputs to outputs, with a fixed computation order. This limitation prevents neural networks from capturing different directions of computation and interaction between the modeled variables.

The idea of modular training of neural networks has a long history. Bottou and Gallinari (1991) propose a framework for training architecture composed of several modules. The modular neural network approach has been used in robotics (Bradley, 2010), where many modules interact and each module would be responsible for a specific task, such as object recognition, processing information from sensors, movement, etc. Neural module networks have been used for NLP applications, such as visual question answering (Andreas et al., 2016) or reasoning (Gupta et al., 2019). Neural Module Networks (Andreas et al., 2016) were applied to visual question answering and provide fine-grained modules for different parts of the task (in the case for VQA, for example, there are modules for a specific color, for recognizing a specific object, etc.). This architecture has been applied to reasoning over text (Gupta et al., 2019). Another line of work (Kirsch et al., 2018) proposes a model that flexibly chooses neural modules based on the data to be processed. They treat the choice of module as a latent variable in a probabilistic model and learn both the decomposition and module parameters end-to-end by maximizing a variational lower bound of the likelihood.

The modular training approach can be related to multi-task learning (Caruana, 1997). For example, a syntax module can be used for predicting several higher level tasks, such as natural language inference (NLI), sentiment analysis, machine translation. In hierarchical multi-task learning, the lower layers of the model learn *low-level NLP tasks* and the later layers learn *higher-level NLP tasks*, which depend on the representations of the low-level tasks (Hashimoto et al., 2016; Sanh et al., 2019). The greedy layer-wise training of neural networks (Hinton et al., 2006) trains layers sequentially starting from bottom (input) layer in an unsupervised way – each layer learns a higher-level representation of the layer below. Bengio et al. (2007) study the algorithm empirically and extend it to cases where the inputs are continuous or where the variable cannot be predicted in a supervised task. The Infomax principle, developed by Linsker (1988) argues that the brain learns to process its perceptions by maximally preserving the information of the input activities in each layer. Recent work uses this principle to create models based on maximizing the mutual information between the input and learned higher-level representations of modules in the model (Oord et al., 2018; Hjelm et al., 2018; Löwe et al., 2019). This work uses modular approach and trains each module to maximally preserve the information of its inputs using the InfoNCE loss.

The idea of deriving new neural network architectures from the inference process of a graphical model has been explored in work such as mean-field networks

(Li and Zemel, 2014) and deep unfolding (Hershey et al., 2014). Some work uses the neural network as a scorer for the structure components and then an output layer calculates the optimal structure (Durrett and Klein, 2015; Fonseca and Martins, 2020; Corro and Titov, 2019b).

In **Chapter 5: Undirected Neural Networks**, we propose *undirected neural networks (UNN)* – a novel energy-based framework in which neural computation is specified with factor graphs instead of directed computation graphs and allows flexibility in defining relations between the model variables. We show how this framework subsumes or relates to many existing neural network architectures. We prove that every feed-forward network can be presented as an undirected neural network, and demonstrate examples of different UNN architectures, including undirected attention.

While our models are targeting NLP applications, they can easily be applied to other domains, as we show in Chapter 5. In this chapter, we demonstrate how undirected neural networks can be used for an NLP problem, but also for sequence completion and image classification and generation.

1.3 Publications

During my PhD, I have co-authored the following work:

- **Scheduled Sampling for Transformers** (Mihaylova and Martins, 2019). In this paper, we explore a way to apply scheduled sampling, a method used in training recurrent neural networks (Bengio et al., 2015) to the Transformer model (Vaswani et al., 2017). The paper was published in the proceedings of the ACL Student Research Workshop 2019. Chapter 3 is based on this work.

- I am one of the authors of the tutorial **Latent Structured Models for Natural Language Processing** which was presented at ACL 2019 (Martins et al., 2019) and RANLP 2019. The work in Chapter 4 is inspired by and builds on parts of this tutorial.

- The paper **Understanding the SPIGOT Mechanics: Surrogate Gradients for Latent Structure Learning** has been accepted to EMNLP 2020 (Mihaylova et al., 2020). It explores into more details training of models with latent structures using gradient surrogate methods, gives insights into such methods and proposes extensions. Chapter 4 is based on this work.
- The paper **Modeling Structure with Undirected Neural Networks** was accepted to ICML 2022. Chapter 5 is based on this work.

1.4 Thesis Outline

- **Chapter 2: Background** introduces the basic concepts needed for understanding the contributions explained in this thesis, such as neural network architectures and NLP tasks used in the thesis; models with latent structures, exposure bias, energy-based models.
- **Chapter 3: Scheduled Sampling for Transformers** presents out work on applying scheduled sampling (Bengio et al., 2015) to the Transformer (Vaswani et al., 2017) architecture.
- **Chapter 4: Surrogate Gradients for Latent Structure Learning** presents our work on surrogate gradients for latent structure learning, which sheds light on the mechanics of SPIGOT (Peng et al., 2018) – a method for learning latent structures.
- **Chapter 5: Undirected Neural Networks** presents our proposed energy-based model which combines neural networks and factor graphs to allow flexibility in expressing the relationship between the model variables.
- In **Chapter 6: Conclusion** we summarize our contributions and provide suggestions for future development of the ideas in this thesis.

CHAPTER 2

Background

Contents

This chapter reviews previous work related to the main ideas of the thesis.

First, in Section 2.1, we describe the main NLP tasks we use throughout the thesis. Section 2.2 give an overview of two main such architectures, in order to provide background for the work in Chapter 3. Section 2.3 describes the necessary background for understanding our recent work, described in Chapter 4. Section 2.4 background revisits some models which motivate our work on Undirected Neural Networks, described in Chapter 5.

2.1 Natural Language Processing Tasks

We provide short descriptions of the Natural Language Processing (NLP) tasks we use throughout the thesis.

2.1.1 Dependency Parsing

Dependency parsing is a structured prediction NLP task that, given a sentence, predicts a directed tree structure which defines the grammatical dependencies between the words in the sentence (Jurafsky and Martin, 2014). Fig. 2.1 shows an example of a dependency tree. In this thesis, we predict dependency parse trees in Chapter 5. In Chapter 4 we do not predict the dependency trees in a supervised way, but we are interested in finding latent dependency trees for a downstream task, such as sentiment analysis, natural language inference or machine translation.

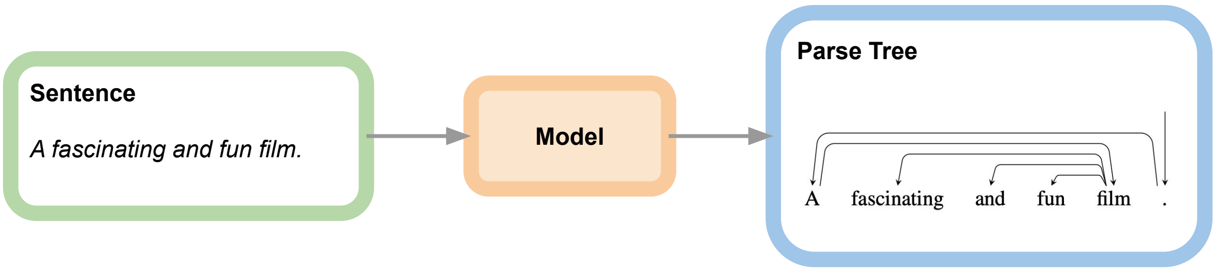


Figure 2.1: An example of dependency parsing.

2.1.2 Natural Language Inference

Natural Language Inference (NLI) is a classification NLP task which takes as an input two sentences, called a premise and a hypothesis and classifies the relationship between those two sentences: entailment (the truth of the premise implies the truth of the hypothesis), contradiction (the truth of the premise implies the falsity of the hypothesis) or neutral (the premise neither entails nor contradicts the hypothesis) (Eisenstein, 2018), see Fig. 2.2. We use NLI for the experiments in Chapter 4.

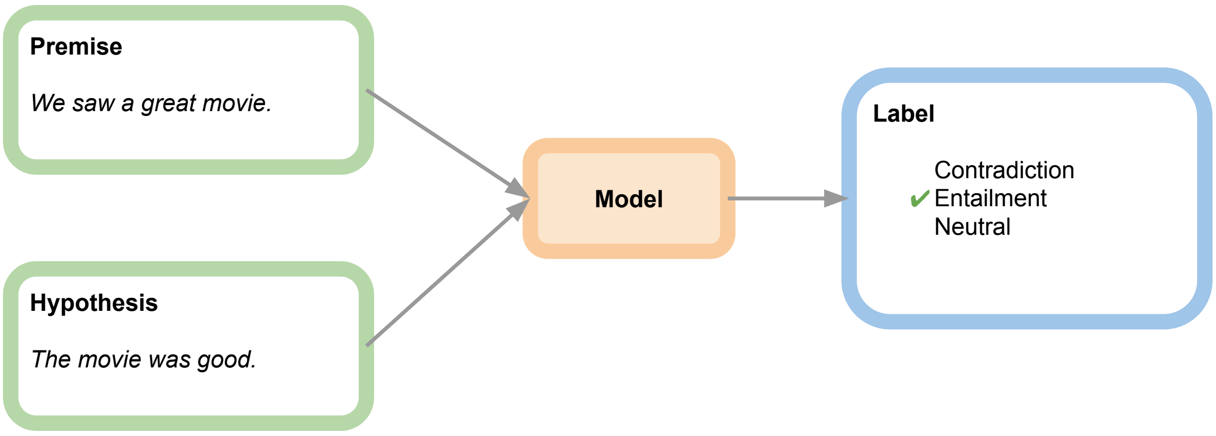


Figure 2.2: An example for natural language inference.

2.1.3 Sentiment Classification

Sentiment classification is an NLP task (Fig. 2.3) which, provided a text sequence, predicts the sentiment of this sequence. The predicted sentiment can have different granularity. It can be a number (for example, from 1 to 5, where higher number means more positive sentiment) or can be binary - positive or negative. We use sentiment classification as downstream task in Chapter 4.

2.1.4 Machine Translation

Machine Translation (MT) is a sequence-to-sequence NLP task which translates an input sequence in a source language to an output sequence in a target language

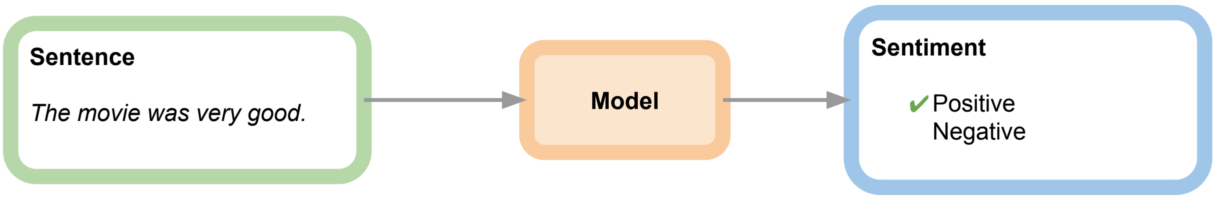


Figure 2.3: An example for sentiment classification with two classes.

(Eisenstein, 2018), as shown on Fig. 2.4. Word **alignment** can be useful for training the MT models or can be extracted from the trained model. The alignments match words or phrases from the source sentence to words or phrases in the target sentence, as shown on Fig. 2.4. We use machine translation for experimenting with scheduled sampling for transformers in Chapter 3.

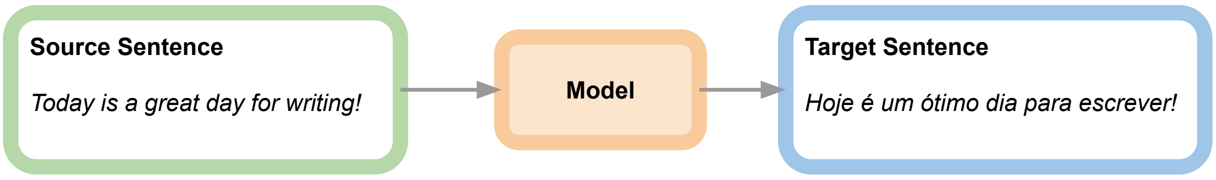

Source Sentence
Today is a great day for writing!
Model
Target Sentence
Hoje é um ótimo dia para escrever!

Figure 2.4: An example for machine translation from English to Portuguese.

2.2 Neural Network Architectures for NLP

2.2.1 Recurrent Neural Networks (RNN)

Recurrent neural networks - RNN (Rumelhart et al., 1986; Werbos, 1990) are neural networks in which the computation is done in steps and the output from the previous step is fed to the next one. This allows modeling output with arbitrary length, the weights of the states are shared, therefore parameter count does not grow with sequence length. A downside is that the computation can be too slow. In the scope of this thesis, we are interested in RNNs for sequence-to-sequence tasks, where an input sequence $x_1, ..., x_n$ is transformed into an output sequence $y_1, ..., y_m$ (Cho et al., 2014; Sutskever et al., 2014). In the encoder of the RNN, the hidden states are represented as $h_t = f(x_t, h_{t-1})$ and the representation of the input sequence is compressed into a vector $c = q(h_1, ..., h_n)$. Then, in the decoder, the probability of modeling the output sequence is optimized:

$$p(y) = \prod_{i=1}^{m} p(y_t|y_1, ..., y_{t-1}, c) \tag{2.1}$$

A common building block for the hidden state is the long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997).

Adding attention - a mechanism that allows the output to focus on different parts of the input (Bahdanau et al., 2014) greatly improves the performance of these models. In the attention mechanism, the vector c is calculated as

$$c_i = \sum_{i=1}^{n} \alpha_{ij} h_j \tag{2.2}$$

where α_{ij} are the attention weights showing the probability with which each output element i attends to the input element j.

$$\alpha_{ij} = \frac{\exp(a_{ij})}{\sum_{k=1}^{n} \exp(a_{ik})} \tag{2.3}$$

where $a_{ij} = a(s_{i-1}, h_j)$ is a scorer for the hidden state in the decoder s_{i-1} for the output y_i and the hidden state of the decoder h_j for input x_j.

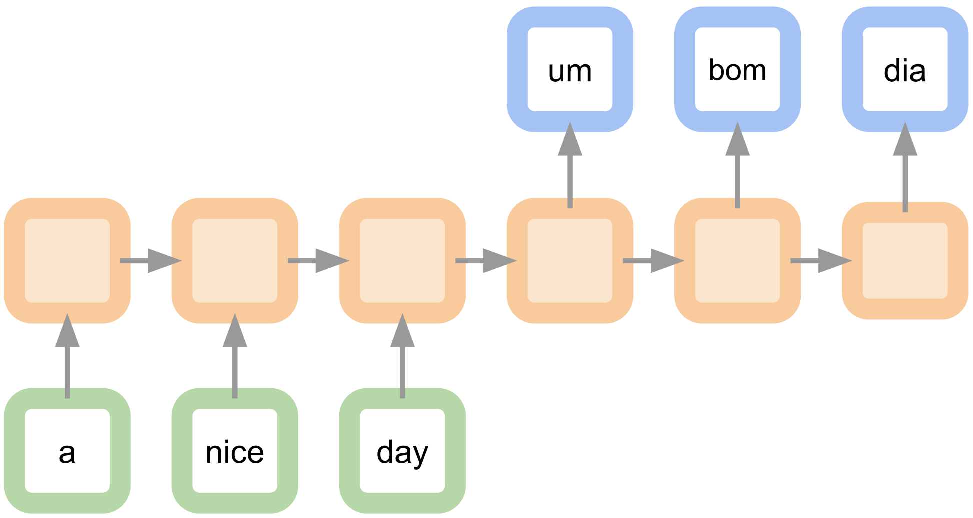


Figure 2.5: Recurrent neural network for sequence-to-sequence prediction, such as machine translation.

2.2.2 Transformers

Introducing the Transformer (Vaswani et al., 2017) led to new state of the art results in sequence-to-sequence prediction and became the model of choice for these models. Its key is that its encoder and decoder are composed of multiple attention layers. It models self-attention on the input sequence in the encoder and the output sequence in the decoder and cross-attention between the output and the input sequence in the decoder. The architecture of the transformer model is shown in Fig. 2.6. In the transformer, all the positions are predicted simultaneously, therefore self-attention is masked so that each element in the sequence attends only to the elements before it. Positional embeddings added to the element embeddings ensure the order of the sequence is kept. A common attention is the scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2.4}$$

where Q, K, V are queries, keys and values and d_k is the dimension of the queries and keys.

2.3 Models with Latent Structures

2.3.1 Structured Prediction Preliminaries

We assume a general latent structure model involving input variables $x \in \mathcal{X}$, output variables $y \in \mathcal{Y}$, and latent discrete variables $z \in \mathcal{Z}$. We assume that $\mathcal{Z} \subseteq \{0,1\}^K$, where $K \leq |\mathcal{Z}|$ (typically, $K \ll |\mathcal{Z}|$): ***i.e.***, the latent discrete variable z can be represented as a K-th dimensional binary vector. This often results from a decomposition of a structure into parts: for example, z could be a dependency tree for a sentence of L words, represented as a vector of size $K = O(L^2)$, indexed by pairs of word indices (i, j), with $z_{ij} = 1$ if arc $i \to j$ belongs to the tree,

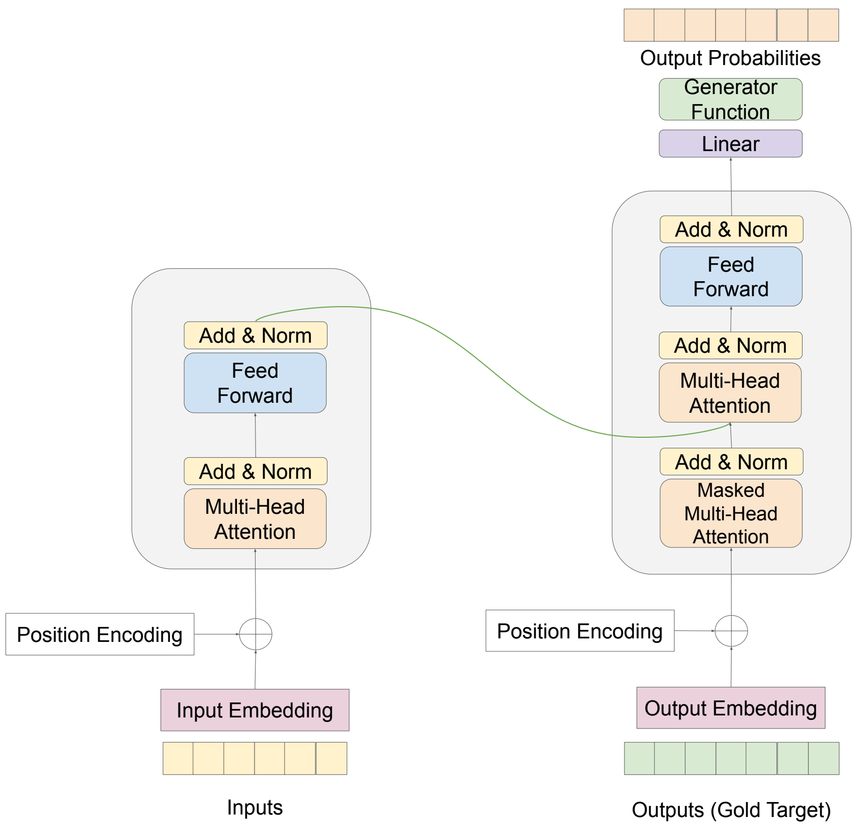


Figure 2.6: Architecture of the Transformer model. From Vaswani et al. (2017).

and 0 otherwise. This allows us to define the **score** of a structure as the sum of the scores of its parts. Given a vector $s \in \mathbb{R}^K$, containing scores for all possible parts, we define

$$\text{score}(z) := s^\top z. \tag{2.5}$$

Background. In the context of structured prediction, the set $\mathcal{M} := \text{conv}(\mathcal{Z})$ is known as the *marginal polytope*, since any point inside it can be interpreted as some marginal distribution over parts of the structure (arcs) under some distribution over structures. There are three relevant problems that may be formulated in a structured setting:

- Maximization (MAP inference): finds a highest scoring structure, $\texttt{MAP}(s) = \arg\max_{z \in \mathcal{Z}} s^\top z$.
- Marginal inference: finds the (unique) marginal distribution induced by the scores s, corresponding to the Gibbs distribution where $p(z) \propto \exp\big(\text{score}(z)\big)$. The so-

lution maximizes the entropy-regularized objective

$$\texttt{Marg}(s) = \operatorname*{argmax}_{\mu \in \mathcal{M}} s^\top \mu + \tilde{H}(\mu), \tag{2.6}$$

where $\tilde{H}$ is the maximum entropy among all distributions over *structures* that achieve marginals μ (Wainwright and Jordan, 2008a):

$$\tilde{H}(\mu) = \max_{\substack{p \in \triangle_{|\mathcal{Z}|} \\ \mathbb{E}_p[z]=\mu}} - \sum_{z \in \mathcal{Z}} p(z) \log p(z). \tag{2.7}$$

- SparseMAP: finds the (unique) *sparse* marginal distribution induced by the scores s, given by a Euclidean projection onto the marginal polytope: (Niculae et al., 2018a)

$$\begin{aligned} \texttt{SparseMAP}(s) &= \Pi_{\mathcal{M}}(s) \\ &= \operatorname*{argmax}_{\mu \in \mathcal{M}} s^\top \mu - \frac{1}{2}\|\mu\|^2. \end{aligned} \tag{2.8}$$

Unstructured setting. As a check, we consider the encoding of a categorical variable with K distinct choices, encoding each choice as a one-hot vector e_k and setting $\mathcal{Z} = \{e_1, \dots, e_K\}$. In this case, $\operatorname{conv}(\mathcal{Z}) = \triangle_K$. The optimization problems above then recover some well known transformations, as described in Table 2.1.

	unstructured	structured
vertices	e_k	z_k
interior points	p	μ
maximization	`argmax`	`MAP`
expectation	`softmax`	`Marg`
Euclidean projection	`sparsemax`	`SparseMAP`

Table 2.1: Building blocks for latent structure models.

2.3.2 Latent Structure Models

Throughout, we assume a classifier parametrized by ϕ and θ, which consists of three parts:

- An **encoder function** f_ϕ which, given an input $x \in \mathcal{X}$, outputs a vector of "scores" $s \in \mathbb{R}^K$, as $s = f_\phi(x)$;
- An **argmax node** which, given these scores, outputs the highest-scoring structure:

$$\hat{z}(s) = \arg\max_{z \in \mathcal{Z}} s^\top z. \tag{2.9}$$

- A **decoder function** g_θ which, given $x \in \mathcal{X}$ and $z \in \mathcal{Z}$, makes a prediction $\hat{y} \in \mathcal{Y}$ as $\hat{y} = g_\theta(x, z)$. We will sometimes write $\hat{y}(z)$ to emphasize the dependency on z. For reasons that will be clear in the sequel, we must assume that the decoder also accepts ***average structures***, ***i.e.***, it can also output predictions $g_\theta(x, \mu)$ where $\mu \in \operatorname{conv}(\mathcal{Z})$ is a convex combination (weighted average) of structures.

Thus, given input $x \in \mathcal{X}$, this network predicts:

$$\hat{y} = g_\theta\left(x, \overbrace{\arg\max_{z \in \mathcal{Z}} f_\phi(x)^\top z}^{\hat{z}(s)}\right). \tag{2.10}$$

To train this network, we minimize a loss function $L(\hat{y}, y)$, where y denotes the target label; a common example is the negative log-likelihood loss.

The gradient *w.r.t.* the decoder parameters, $\nabla_\theta L(\hat{y}, y)$, is easy to compute using automatic differentiation on g_θ. The main challenge is propagate gradient information **through the argmax node** into the encoder parameters. Indeed, we have:

$$\nabla_\phi L(\hat{y}, y) = \frac{\partial f_\phi(x)}{\partial \phi} \underbrace{\frac{\partial \hat{z}(s)}{\partial s}}_{=0} \nabla_z L(\hat{y}(\hat{z}), y) = 0, \tag{2.11}$$

so no gradient will flow to the encoder. We list below the three main categories of approaches that tackle this issue.

Introducing stochasticity. Replace the argmax node by a stochastic node where z is modeled as a random variable Z parametrized by s (*e.g.*, using a Gibbs distribution). Then, instead of optimizing a deterministic loss $L(\hat{y}(\hat{z}), y)$, optimize the

expectation of the loss under the predicted distribution:

$$\mathbb{E}_{Z \sim p(z;s)}[L(\hat{y}(Z), y)]. \tag{2.12}$$

The expectation ensures that the gradients are no longer null. This is sometimes referred to as *minimum risk training* (Smith and Eisner, 2006; Stoyanov et al., 2011), and typically optimized using the *score function estimator* (SFE; Glynn, 1990; Williams, 1992; Kleijnen and Rubinstein, 1996).

Relaxing the argmax. Keep the network deterministic, but relax the argmax node into a continuous function, for example replacing it with softmax or sparsemax (Martins and Astudillo, 2016b). In the structured case, this gives rise to structured attention networks (Kim et al., 2017) and their SparseMAP variant (Niculae et al., 2018a). This corresponds to moving the expectation inside the loss, optimizing $L\big(\hat{y}(\underbrace{\mathbb{E}_{Z \sim p(z;s)}[Z]}_{\mu}), y\big)$.

Inventing a surrogate gradient. Keep the argmax node and perform the usual forward computation, but backpropagate a different, non-null gradient in the backward pass. This is the approach underlying straight-through estimators (Hinton, 2012; Bengio et al., 2013) and SPIGOT (Peng et al., 2018). This method introduces a mismatch between the measured objective and the optimization algorithm. In Chapter 4 of this thesis, we propose a novel, principled justification for inducing surrogate gradients. In what follows, we assume that:

- We can compute the gradient

$$\gamma(\mu) := \nabla_\mu L(\hat{y}(\mu), y) \tag{2.13}$$

  for any μ, *e.g.*, by using automatic differentiation on the decoder;[1]

[1]This gradient would not exist if the decoder g_θ were defined only at vertices $z \in \mathcal{Z}$ and not mean points $\mu \in \mathcal{M}$.

- We want to replace the null gradient $\nabla_s L(\hat{y}(\hat{z}), y)$ by a surrogate $\tilde{\nabla}_s L(\hat{y}(\hat{z}), y)$.

2.3.3 Straight-Through Estimator

The straight-through estimator was introduced in a lecture (Hinton, 2012), where it was described as a way to backpropagate through a step function. A multilayer neural network consisting of logistic units $p(s = 1) = \frac{1}{1+e^{-z}}$ is trained with backpropagation. In the forward pass, a binary value is sampled and the backward pass is done as if in the forward pass no sampling was done. They report an unpublished result where this kind of training does worse on the training set but performs significantly better on the test set.

We use the same intuition for backpropagating through nodes which have an ***argmax*** on the forward pass $z = argmax(s)$ during training and use the identity matrix as the derivative $\frac{\partial z}{\partial s}$.

The straight-through estimator was described also in Bengio et al. (2013).

2.3.4 SPIGOT

The structured projection of intermediate gradients optimization technique (SPIGOT), is a method for backpropagating through neural networks that include hard-decision structured predictions (e.g., parsing) in intermediate layers (Peng et al., 2018). Unlike STE's gradient proxy, SPIGOT aims to respect the constraints in the argmax problem.

SPIGOT introduces a projection step that aims to keep the "updated" $\hat{z}$ in the feasible set. Of course, we do not directly update $\hat{z}$; backpropagation continues through s and onward to the parameters. But the projection step alters the parameter updates in the way that the proxy for $\nabla_s L$ is defined.

$$\nabla_s L = \hat{z} - proj_P(\hat{z} - \eta \nabla_{\hat{z}} L) \tag{2.14}$$

In Chapter 4, we give a more intuitive explanation of this update.

2.4 Energy-Based Learning

Another set of methods relevant for this thesis is Energy-based learning. The background in this section provides the necessary preliminaries for Chapter 5.

Energy-based models, in contrast to predicting an output y from an input x, model an energy function of both x and y, denoted as $E(x, y)$ and aim to minimize this fundtion. The energy can be viewed as a measure of compatibility between the variables - the more compatible the values of the variables, the lower the energy. On inference, the observed variables are given and the values of the unobserved variables are assigned so that minimize the model energy. Training the model means finding an energy function which outputs low energies when correct values are assigned to the unobserved variables and high energies when incorrect values are assigned to the unobserved variables (LeCun et al., 2006). For example, for the observed input x, values of y are searched that minimize the model energy. The goal is to find y^*, chosen from a set Y, for which $E(x, y)$ is the smallest: $y^* = argmin_y E(x, y)$.

2.4.1 Boltzmann Machines

A Boltzmann Machine (Ackley et al., 1985) is a network of symmetrically connected, neuronlike units that make stochastic decisions about whether to be on or off. Boltzmann machines have a simple learning algorithm that allows them to discover interesting features in datasets composed of binary vectors. The learning algorithm is intractable in networks with many layers of features. (Hinton, 2007) Boltzmann machines are pairwise fully connected networks with N visible units $x \in \{0, 1\}^N$ and M hidden units $x \in \{0, 1\}^M$. They define the following energy function:

$$E(x, h) = -x^T V x - h^T W x - h^T U h - a^T x - b^T h \tag{2.15}$$

The probability of a configuration is defined as:

$$P(x,h) = \frac{\exp(-E(x,h))}{Z} \tag{2.16}$$

(where Z is the partition function) and maximizing the probability corresponds to minimizing the energy. Bilinear factor energy is defined as:

$$E_{x,h}(x,h) = -h^T W x \tag{2.17}$$

and unary energies are defined as:

$$E_x(x) = -x^T V x - a^T x + \iota_{\{0,1\}^N}(x) \tag{2.18}$$

$$E_h(h) = -h^T U h - b^T h + \iota_{\{0,1\}^M}(h) \tag{2.19}$$

2.4.2 Restricted Boltzmann Machines

Restricted Boltzmann machines are special cases of Boltzmann machines where the connections between visible and hidden units form a bipartite graph. In this case, we have:

$$E_{X,H}(x,h) = -h^T W x \tag{2.20}$$

$$E_X(x) = -a^T x + \iota_{\{0,1\}^N}(x) \tag{2.21}$$

$$E_H(h) = -b^T h + \iota_{\{0,1\}^M}(h) \tag{2.22}$$

2.4.3 Deep Boltzmann Machines

Deep Boltzmann machines are similar to restricted Boltzmann machines, but with more layers. Here the energies are defined as follows:

$$E_{X,H_1}(x, h_1) = -h_1^T W_1 x \tag{2.23}$$

$$E_{H_{k-1},H_k}(h_{k-1}, h_k) = -h_k^T W_k x \tag{2.24}$$

$$E_X(x) = -a^T x + \iota_{\{0,1\}^N}(x) \tag{2.25}$$

$$E_{H_k}(h_k) = -b_k^T h_k + \iota_{\{0,1\}^{M_h}}(h_k) \tag{2.26}$$

CHAPTER 3

Scheduled Sampling for Transformers

Contents

Scheduled sampling is a technique for avoiding one of the known problems in sequence-to-sequence generation: exposure bias. It consists of feeding the model a mix of the teacher forced embeddings and the model predictions from the previous step in training time. The technique has been used for improving the model performance with recurrent neural networks (RNN) (Bengio et al., 2015). In the Transformer model, unlike the RNN, the generation of a new word attends to the full sentence generated so far, not only to the last word (see Section 2.2.2), and it is not straightforward to apply the scheduled sampling technique. We propose some structural changes to allow scheduled sampling to be applied to Transformer architecture, via a two-pass decoding strategy. Experiments on two language pairs achieve performance on par with a teacher-forcing baseline and show that this technique is promising for further exploration.

3.1 Introduction

Before the transformers (Vaswani et al., 2017) became adopted, neural machine translation (NMT) relied on a sequence-to-sequence model with global attention (Sutskever et al., 2014; Bahdanau et al., 2014), trained with maximum likelihood estimation (MLE). These models are typically trained by teacher forcing, in which the model makes each decision conditioned on the gold history of the target sequence. This tends to lead to quick convergence but is dissimilar to the procedure used at decoding time, when the gold target sequence is not available and decisions are conditioned on previous model predictions.

Ranzato et al. (2015) point out the problem that using teacher forcing means the model has never been trained on its own errors and may not be robust to them—a phenomenon called *exposure bias*. This has the potential to cause problems at translation time, when the model is exposed to its own (likely imperfect) predictions.

A common approach for addressing the problem with exposure bias is using a scheduled strategy for deciding when to use teacher forcing and when not to (Bengio et al., 2015). For a recurrent decoder, applying scheduled sampling is trivial: for generation of each word, the model decides whether to condition on the gold embedding from the given target (teacher forcing) or the model prediction from the previous step.

In the Transformer model (Vaswani et al., 2017), the decoding is still autoregressive, but unlike the RNN decoder, the generation of each word conditions on the whole prefix sequence and not only on the last word and in training time, the whole output sequence is predicted using one decoder pass with masking. While we can still use the standard approach where at each step we predict one word conditioning on the prefix from the gold sequence or the model prediction, this would significantly slow down the computation, because for the prediction of each word, we would have one decoder pass. Since the Transformer achieves state-of-the-art results and has become a default choice for many natural language processing problems, it is interesting to adapt and explore the idea of scheduled sampling for it, and, to our knowledge, no way of doing this had been proposed when we started working on this idea. Our approach allows to apply scheduled sampling for transformers while keeping the parallelization of the decoder forward pass on GPU during training using masking.

Our contributions in this chapter are:

- We propose a new strategy for using scheduled sampling in Transformer models by making two passes through the decoder in training time.
- We compare several approaches for conditioning on the model predictions when they are used instead of the gold target.
- We test the scheduled sampling with transformers in a machine translation task on two language pairs and achieve results on par with a teacher forcing baseline (with a slight improvement of up to 1 BLEU point for some models).[1]

[1]The source code is on: https://github.com/deep-spin/scheduled-sampling-transformers

3.2 Related Work

Bengio et al. (2015) proposed scheduled sampling for sequence-to-sequence RNN models: a method where the embedding used as the input to the decoder at time step $t+1$ is picked randomly between the gold target and the `argmax` of the model's output probabilities at step t. The Bernoulli probability of picking one or the other changes over training epochs according to a schedule that makes the probability of choosing the gold target decrease across training steps. The authors propose three different schedules: linear decay, exponential decay and inverse sigmoid decay. Fig. 3.1 shows an example of how this model works.

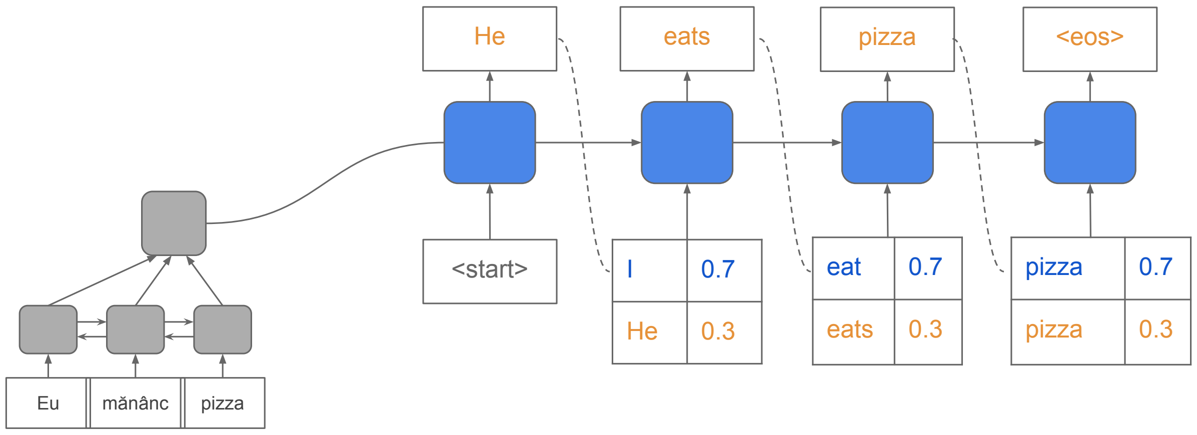


Figure 3.1: Example for scheduled sampling with recurrent neural networks (Bengio et al., 2015) for a machine translation task for a language pair. At each step for word prediction, the gold label or the word predicted from the previous step are fed with a probability (in this example: 0.7 for the gold label and 0.3 for the model prediction).

Goyal et al. (2017) proposed an approach based on scheduled sampling which backpropagates through the model decisions. At each step, when the model decides to use model predictions, instead of the `argmax`, they use a weighted average of all word embeddings, weighted by the prediction probabilities. They experimented with two options: a softmax with a temperature parameter, and a stochastic variant using Gumbel Softmax (Jang et al., 2016) with temperature. With this technique, they achieve better results than the standard scheduled sampling. Fig. 3.2 shows an example of how this model works.

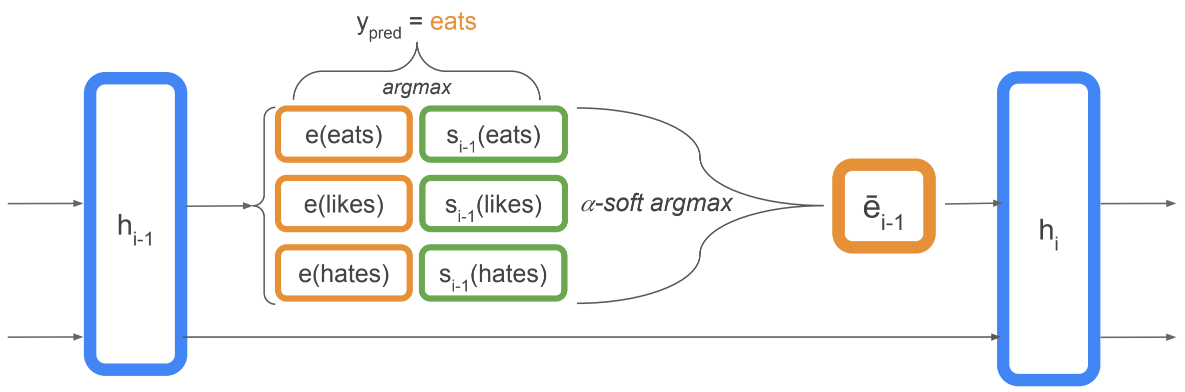


Figure 3.2: Differentiable Scheduled Sampling for Credit Assignment by Goyal et al. (2017) using continuous approximation or argmax. The weighted sum of the predictions from the previous step is feeded to the next time step.

Our work extends Bengio et al. (2015) and Goyal et al. (2017) by adapting their frameworks to Transformer architectures.

Ranzato et al. (2015) took ideas from scheduled sampling and the REINFORCE algorithm (Williams, 1992) and combine the teacher forcing training with optimization of the sequence level loss. In the first epochs, the model is trained with teacher forcing and for the remaining epochs they start with teacher forcing for the first t time steps and use REINFORCE (sampling from the model) until the end of the sequence. They decrease the time for training with teacher forcing t as training continues until the whole sequence is trained with REINFORCE in the final epochs. Other methods that are also focused on sequence-level training are using for example actor-critic (Bahdanau et al., 2016) or beam search optimization (Wiseman and Rush, 2016). These methods directly optimize the metric used at test time (e.g. BLEU). Another proposed approach to avoid exposure bias is SEARN (Daumé et al., 2009). In SEARN, the model uses its own predictions at training time to produce sequence of actions, then a search algorithm determines the optimal action at each step and a policy is trained to predict that action. The main drawback of these approaches is that the training becomes much slower. The method we focus on in this chapter is comparable in training time with a force-decoding baseline.

3.3 Scheduled Sampling with Transformers

With recurrent neural networks (RNN), in the training phase we generate one word at each time step, and we feed the previous word as an input in the next time step. This sequential decoding makes it simple to apply scheduled sampling - at each time step, with some probability, instead of using the previous word in the gold sequence, we use the word predicted from the model on the previous step.

The Transformer model (Vaswani et al., 2017), which achieves state-of-the-art results for many natural language processing tasks, is also an autoregressive model. The generation of each word attends to all previous words in the sequence, not only to the last generated word. The model is based on multiple *self-attention layers*, which directly model relationships between all words in the sentence, regardless of their respective position. The order of the words is encoded through position embeddings, which are summed with the corresponding word embeddings. Using position masking in the decoder ensures that the generation of each word depends only on the previous words in the sequence and not on the following ones. Because generation of a word in the Transformer attends to all previous words in the sequence and not just the last word, it is not trivial to apply scheduled sampling to it, where, in training time, we need to choose between using the gold target word or the model prediction. In order to allow usage of scheduled sampling with the Transformer model, we needed to make some changes in the Transformer architecture.

3.3.1 Two-decoder Transformer

The model we propose for applying scheduled sampling in transformers makes two passes on the decoder. Its architecture is illustrated on Fig. 3.3. We make no changes in the encoder of the model. The decoding of the scheduled transformer

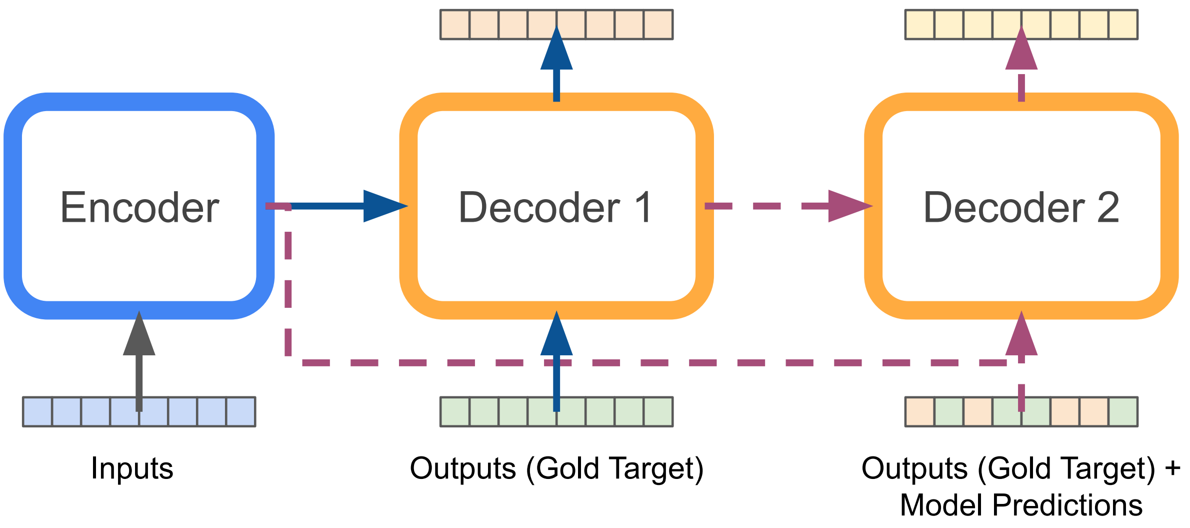


Figure 3.3: Transformer model adapted for use with scheduled sampling. The two decoders on the image share the same parameters. The first pass on the decoder conditions on the gold target sequence and returns the model predictions. The second pass conditions on a mix of the target sequence and model predictions and returns the result. The model always backpropagates through the second decoder pass. Backpropagated through the first decoder pass is performed only in a part of the experiments.

has the following steps:

1. **First pass on the decoder: get the model predictions.** On this step, the decoder conditions on the gold target sequence and predicts scores for each position as a standard transformer model. Those scores are passed to the next step.

2. **Mix the gold target sequence with the predicted sequence.** After obtaining a sequence representing the prediction from the model for each position, we imitate scheduled sampling by mixing the target sequence with the model predictions: For each position in the sequence, we select with a given probability whether to use the gold token or the prediction from the model. The probability for using teacher forcing (i.e. the gold token) is a function of the training step and is calculated with a selected schedule. We pass this “new reference sequence” as the reference for the second decoder. The vectors used from the model predictions can be either the embedding of the highest-scored

word, or a mix of the embeddings according to their scores. Several variants of building the vector from the model predictions for each position are described below.

3. **Second pass on the decoder: the final predictions.** The second pass of the decoder uses as output target the mix of words in the gold sequence and the model predictions. The outputs of this decoder pass are the actual result from the models.

It is important to mention that the two decoders are identical and share the same parameters. We use the same decoder for the first pass, where we condition on the gold sequence and the second pass, where we condition on the mix between the gold sequence and the model predictions. Also, it is important to mention that the input and the output embeddings of the decoder are shared, in order to allow feeding the output of the first decoder as an input to the second decoder pass.

3.3.2 Embedding Mix

For each position in the sequence, the first decoder pass gives a score for each vocabulary word. We explore several ways of using those scores when the model predictions are used.

- The most straight-forward case is to not mix the embeddings at all and pass the `argmax` from the model predictions, i.e. use the embedding of the vocabulary word with the highest score from the decoder.
- We also experiment with mixing the `top-k` embeddings. In our experiments, we use the weighted average of the embeddings of the top-5 scored vocabulary words. As weights we use the output probabilities from the first decoder pass and normalize them to sum to one for the top five words.

- Inspired by the work of Goyal et al. (2017), we experiment with passing a mix of the embeddings with `softmax` with temperature. Using a higher temperature parameter makes a better approximation of the `argmax`:

$$\bar{e}_{i-1} = \sum_{y} e(y) \frac{\exp(\alpha s_{i-1}(y))}{\sum_{y'} \exp(\alpha s_{i-1}(y'))} \tag{3.1}$$

  where $\bar{e}_{i-1}$ is the vector which will be used at the current position, obtained by a sum of the embeddings of all vocabulary words, weighted by a softmax of the scores s_{i-1}.

- An alternative of using argmax is sampling an embedding from the softmax distribution. Also based on the work of Goyal et al. (2017), we use the Gumbel Softmax (Maddison et al., 2016; Jang et al., 2016) approximation to sample the embedding:

$$\bar{e}_{i-1} = \sum_{y} e(y) \frac{\exp(\alpha(s_{i-1}(y)) + G_y)}{\sum_{y'} \exp(\alpha(s_{i-1}(y') + G_{y'}))} \tag{3.2}$$

  where $U \sim \mathrm{Uniform}(0,1)$ and $G = -\log(-\log U)$.

- Finally, we experiment with passing a `sparsemax` mix of the embeddings (Martins and Astudillo, 2016a).

3.3.3 Decay strategy

With the scheduled sampling method, the teacher forcing probability continuously decreases over the course of training according to a predefined function of the training steps. The training starts with all teacher forcing, i.e. all tokens from the gold sequence are fed to the next step, and then as training processes, the probability of using a gold token decreases in favor of increasing the probability of feeding a token predicted by the model. The decrease of the probability is being calculated with the so called *decay strategy*. The decay strategy determines the teacher forcing ratio t for training step i, that is, the probability of doing teacher forcing at each position in the sequence. The following decay strategies have been proposes (also illustrated in Fig. 3.4):

- Linear decay: $t(i) = \max\{\epsilon, k - ci\}$, where $0 \leq \epsilon < 1$ is the minimum teacher forcing probability to be used in the model and k and c provide the offset and slope of the decay.

- Exponential decay: $t(i) = k^i$, where $0 \leq k < 1$ is a hyperparameter used to adjust the decay.

- Inverse sigmoid decay: $t(i) = \frac{k}{k + \exp \frac{i}{k}}$, where $k \geq 1$ is a hyperparameter used to adjust the decay.

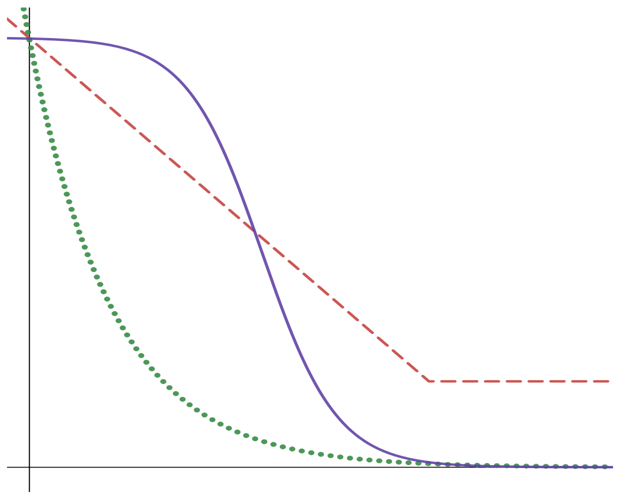


Figure 3.4: Teacher forcing decay schedules, showing how the probability of using the gold token in the next decoding step decreases over the course of training (in the beginning of the training, the probability of using the gold token is 100%). The following schedules are shown: linear (red dashed line), exponential (green dotted line) and inverse sigmoid (purple line).

Among the decay strategies proposed for scheduled sampling, we found that *linear decay* is the one that works best for our data.

3.3.4 Weights update

We calculate Cross Entropy Loss based on the outputs from the second decoder pass. For the cases where all vocabulary words are summed (Softmax, Gumbel Softmax, Sparsemax), we try two variants of updating the model weights.

- Only backpropagate through the decoder which makes the final predictions, based on mix between the gold target and the model predictions. This is marked in Table 3.2 as **No backprop**.
- Backpropagate through the second, as well as through the first decoder pass which predicts the model outputs. This setup resembles the differentiable scheduled sampling proposed by Goyal et al. (2017). This is marked in Table 3.2 as **Backprop through model decisions**.

3.4 Experiments

Encoder model type	Transformer
Decoder model type	Transformer
# Enc. & dec. layers	6
Heads	8
Hidden layer size	512
Word embedding size	512
Batch size	32
Optimizer	Adam
Learning rate	1.0
Warmup steps	20,000
Maximum training steps	300,000
Validation steps	10,000
Position Encoding	True
Share Embeddings	True
Share Decoder Embeddings	True
Dropout	0.2 (DE-EN)
Dropout	0.1 (JA-EN)

Table 3.1: Hyperparameters shared across models

We report experiments with scheduled sampling for Transformers for the task of machine translation. We run the experiments on two language pairs:

Experiment	**DE–EN**		**JA–EN**	
	Dev	**Test**	**Dev**	**Test**
Teacher Forcing Baseline	35.05	**29.62**	18.00	19.46
No backprop				
Argmax	23.99	20.57	12.88	15.13
Top-k mix	35.19	29.42	**18.46**	20.24
Softmax mix $\alpha = 1$	35.07	29.32	17.98	20.03
Softmax mix $\alpha = 10$	35.30	29.25	17.79	19.67
Gumbel Softmax mix $\alpha = 1$	**35.36**	29.48	18.31	20.21
Gumbel Softmax mix $\alpha = 10$	35.32	29.58	17.94	**20.87**
Sparsemax mix	35.22	29.28	18.14	20.15
Backprop through model decisions				
Softmax mix $\alpha = 1$	33.25	27.60	15.67	17.93
Softmax mix $\alpha = 10$	27.06	23.29	13.49	16.02
Gumbel Softmax mix $\alpha = 1$	30.57	25.71	15.86	18.76
Gumbel Softmax mix $\alpha = 10$	12.79	10.62	13.98	17.09
Sparsemax mix	24.65	20.15	12.44	16.23

Table 3.2: Experiments with scheduled sampling for Transformer. The table shows BLEU score for the best checkpoint on BLEU, measured on the validation set. The first group of experiments do not have a backpropagation pass through the first decoder. The results from the second group are from model runs with backpropagation pass through the second as well as through the first decoder.

- IWSLT 2017 German–English (DE–EN, Cettolo et al. (2017)).
- KFTT Japanese–English (JA–EN, Neubig (2011)).

We use byte pair encoding (BPE; (Sennrich et al., 2016)) with a joint segmentation with 32,000 merges for both language pairs.

Hyperparameters used across experiments are shown in Table 3.1. All models were implemented in a fork of OpenNMT-py (Klein et al., 2017). We compare our model to a **teacher forcing baseline**, i.e. a standard transformer model, without scheduled sampling, with the hyperparameters given in Table 3.1. We did hyperparameter tuning by trying several different values for dropout and warmup steps, and choosing the best BLEU score on the validation set for the baseline model.

We used *linear decay schedule* throughout the training, i.e. the probability of choosing the gold token at each position is starting from 1 at the beginning of the training and going to 0 at the end of the training. We experimented also with inverse sigmoid decay schedule, but the steeper decrease in the usage of the gold target token was leading to the model performance degrading very quickly.

The results from our experiments are shown In Table 3.2. The scheduled sampling which uses only the highest-scored word predicted by the model does not have a very good performance. The models which use mixed embeddings (the top-k, softmax, Gumbel softmax or sparsemax) and only backpropagate through the second decoder pass, perform slightly better than the baseline on the validation set, and one of them is also slightly better on the test set. The differentiable scheduled sampling (when the model backpropagates through the first decoder) have much lower results. The performance of these models starts degrading too early, therefore the lower results in Table 3.2. We expect that using more training steps with teacher forcing at the beginning of the training would lead to better performance.

3.5 Discussion

In this chapter, we presented our approach to applying the scheduled sampling technique to Transformers. Because of the specifics of the decoding, applying scheduled sampling is not straightforward as it is for RNN and required some changes in the way the Transformer model is trained, by using a two-step decoding. We experimented with several schedules and mixing of the embeddings in the case where the model predictions were used. We tested the models for machine translation on two language pairs. The experimental results showed that our scheduled sampling strategy gave better results on the validation set for both language pairs compared to a teacher forcing baseline and, in one of the tested language pairs (JA–EN), there were slightly better results on the test set.

The case where we backpropagate through the second as well through the first decoder, gives worse results than when we only backpropagate through the second decoder. This finding suggests that our current approaches to backprop through discrete choices may not always work well and are not sufficiently understood.

Slightly after this work was published, another paper proposed the idea of parallel scheduled sampling (Duckworth et al., 2019) which is very similar to this work. They are more focused on the theoretical explanation of the problem, while we looked at it from a more practical point of view. There have also been follow-ups to our work. For example, Liu et al. (2021) improve the scheduled sampling for transformers by using the confidence of prediction of the target token to decide whether to feed it or the gold one. Korakakis and Vlachos (2021) show that even though scheduled sampling addresses exposure bias by increasing model reliance on the input sequence, it also leads to output degradation due to catastrophic forgetting, both in RNN and transformers. They mitigate the problem by using Elastic Weight Consolidation (Kirkpatrick et al., 2016).

CHAPTER 4

Surrogate Gradients for Latent Structure Learning

Contents

Latent structure models are a powerful tool for modeling language data: they can mitigate the error propagation and annotation bottleneck in pipeline systems, while simultaneously uncovering linguistic insights about the data. One challenge with end-to-end training of these models is the argmax operation, which has null gradient. We focus on *surrogate gradients*, a popular strategy to deal with this problem. We explore latent structure learning through the angle of *pulling back* the downstream learning objective. In this paradigm, we discover a principled motivation for both the straight-through estimator (STE) as well as the recently-proposed SPIGOT—a variant of STE for structured models. Our perspective leads to new algorithms in the same family. We empirically compare the known and the novel pulled-back estimators against the popular alternatives, yielding new insight for practitioners and revealing intriguing failure cases.

4.1 Introduction

Natural language data is *rich in structure*, but most of the structure is not visible at the surface. Machine learning models tackling high-level language tasks would benefit from uncovering underlying structures such as trees, sequence tags, or segmentations. Traditionally, practitioners turn to *pipeline* approaches where an external, pretrained model is used to predict, *e.g.*, syntactic structure. The benefit of this approach is that the predicted tree is readily available for inspection, but the downside is that the errors can easily propagate throughout the pipeline and require further attention (Finkel et al., 2006; Sutton and McCallum, 2005; Toutanova, 2005). In contrast, deep neural architectures tend to eschew such preprocessing, and instead learn soft hidden representations, not easily amenable to visualization and analysis.

The best of both worlds would be to *model structure as a latent variable*, combining the transparency of the pipeline approach with the end-to-end unsupervised representation learning that makes deep models appealing. Moreover, large-capacity model tend to rediscover structure from scratch (Tenney et al., 2019), so structured latent variables may reduce the required capacity.

Learning with discrete, combinatorial latent variables is, however, challenging, due to the intersection of *large cardinality* and *null gradient* issues. For example, when learning a latent dependency tree, the latent parser must choose among an exponentially large set of possible trees; what's more, the parser may only learn from gradient information from the downstream task. If the highest-scoring tree is selected using an *argmax* operation, the gradients will be zero, preventing learning.

One strategy for dealing with the null gradient issue is to use a **surrogate gradient**, explicitly overriding the zero gradient from the chain rule, as if a different computation had been performed. The most commonly known example is the *straight-through estimator* (STE; Bengio et al., 2013), which pretends that the *argmax* node was instead an *identity* operator. Such methods lead to a fundamental mismatch between the objective and the learning algorithm. The effect of this mismatch is still insufficiently understood, and the design of successful new variants is therefore challenging. For example, the recently-proposed SPIGOT method (Peng et al., 2018) found it beneficial to use a projection as part of the surrogate gradient.

In this chapter, we study surrogate gradient methods for deterministic learning with discrete structured latent variables. Our contributions are:

- We propose a novel motivation for surrogate gradient methods, based on optimizing a **pulled-back loss**, thereby inducing pseudo-supervision on the latent variable. This leads to new insight into both STE and SPIGOT.
- We show how our framework may be used to derive new surrogate gradient methods, by varying the loss function or the inner optimization algorithm used for inducing the pseudo-supervision.
- We experimentally validate our discoveries on a controllable experiment as well

$$\begin{tikzcd}
 & & & & \theta & \\
\text{(a)} & x \arrow[r, "f_\phi"] & s \arrow[r, "\text{argmax}"] & \hat{z} \arrow[r, "g_\theta"] & \hat{y} & L(\hat{y}, y) \arrow[ul, bend right] \arrow[dll] \\
 & & \phi & \times & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & & & \theta & \\
\text{(b)} & x \arrow[r, "f_\phi"] & s \arrow[r, "\text{argmax}"] & \hat{z} \arrow[r, "g_\theta"] & \hat{y} & L(\hat{y}, y) \arrow[ul, bend right] \\
 & & \phi & & & \\
 & & & & \ell(\hat{z}, z) \arrow[ull, bend right] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & & & \theta & \\
\text{(c)} & x \arrow[r, "f_\phi"] & s \arrow[r, "\text{argmax}"] & \hat{z} \arrow[r, "g_\theta"] & \hat{y} & L(\hat{y}, y) \arrow[ul, bend right] \arrow[ddl, dotted] \\
 & & \phi & & & \\
 & & & & \ell(\hat{z}, \mu) \arrow[ull, bend right] &
\end{tikzcd}$$

Figure 4.1: A model with a discrete latent variable z. Given an input x, we assign a score $s_z = [f(x)]_z$ to each choice, and pick the highest scoring one, $\hat{z}$, to predict $\hat{y} = g_\theta(\hat{z})$. For simplicity, here g_θ does not access x directly. (a). Since argmax has null gradients, the encoder parameters ϕ do not receive updates. (b). If ground truth supervision were available for the latent z, ϕ could be trained jointly with an auxiliary loss. (c). As such supervision is not available, we induce a best-guess label μ by pulling back the downstream loss. This strategy recovers the STE and SPIGOT estimators.

as on English-language sentiment analysis and natural language inference, comparing against stochastic and relaxed alternatives, yielding new insights, and identifying noteworthy failure cases.[1]

While the discrete methods do not outperform the relaxed alternatives using the same building blocks, we hope that our interpretation and insights would trigger future latent structure research.

4.2 Related Work

Discrete latent variable learning is often tackled in **stochastic computation graphs**, by estimating the gradient of an expected loss. An established method is the score function estimator (SFE) (Glynn, 1990; Williams, 1992; Kleijnen and Rubinstein,

[1]The source code is on: `https://github.com/deep-spin/understanding-spigot`.

1996). SFE is widely used in NLP, for tasks including minimum risk training in NMT (Shen et al., 2016; Wu et al., 2018) and latent linguistic structure learning (Yogatama et al., 2017; Havrylov et al., 2019). We focus on the alternative strategy of **surrogate gradients**, which allows learning in deterministic graphs with discrete, argmax-like nodes, rather than in stochastic graphs. Examples are the **straight-through estimator (STE)** (Hinton, 2012; Bengio et al., 2013) and the structured projection of intermediate gradients optimization technique (SPIGOT; Peng et al. 2018). Recent work focuses on studying and explaining STE. Yin et al. (2019) obtained a convergence result in shallow networks for the unstructured case. Cheng et al. (2018) show that STE can be interpreted as the simulation of the projected Wasserstein gradient flow. STE has also been studied in binary neural networks (Hubara et al., 2016) and in other applications (Tjandra et al., 2019). Other methods based on the surrogate gradients have been recently explored (Vlastelica et al., 2020; Meng et al., 2020).

A popular alternative is to **relax** an argmax into a continuous transform such as softmax or sparsemax (Martins and Astudillo, 2016b), as seen for instance in soft attention mechanisms (Vaswani et al., 2017), or structured attention networks (Kim et al., 2017; Maillard et al., 2017; Liu and Lapata, 2018; Mensch and Blondel, 2018; Niculae et al., 2018a). In-between surrogate gradients and relaxation is **Gumbel softmax**, which uses the Gumbel-max reparametrization to sample from a categorical distribution, applying softmax either to relax the mapping or to induce surrogate gradients (Jang et al., 2017; Maddison et al., 2016). Gumbel-softmax has been successfully applied to latent linguistic structure as well (Choi et al., 2018; Maillard and Clark, 2018). For sampling from a structured variable is required, the **Perturb-and-MAP** technique (Papandreou and Yuille, 2011) has been successfully applied to sampling latent structures in NLP applications (Corro and Titov, 2019a,b).

4.3 SPIGOT as the Approximate Optimization of a Pulled Back Loss

We next provide a novel interpretation of SPIGOT as the minimization of a "pulled back" loss. SPIGOT uses the surrogate gradient:

$$\begin{aligned}\tilde{\nabla}_s L(\hat{y}(\hat{z}), y) &= \hat{z} - \Pi_{\mathcal{M}}\left(\hat{z} - \eta\gamma\right) \\ &= \hat{z} - \texttt{SparseMAP}(\hat{z} - \eta\gamma),\end{aligned} \tag{4.1}$$

highlighting that SparseMAP (Niculae et al., 2018a) computes an Euclidean projection (Eq. (2.8)).

4.3.1 Intermediate Latent Loss

To begin, consider a much simpler scenario: if we had supervision for the latent variable z (*e.g.*, if the true label z was revealed to us), we could define an **intermediate loss** $\ell(\hat{z}, z)$ which would induce nonzero updates to the encoder parameters. Of course, we do not have access to this z. Instead, we consider the following alternative:

Definition 1 (Pulled-back label)**.** A guess $\mu \in \mathcal{M} = \operatorname{conv}(\mathcal{Z})$ for what the unknown $z \in \mathcal{Z}$ should be, informed by the downstream loss.

Fig. 4.1 provides the intuition of the pulled-back label and loss. We take a moment to justify picking $\mu \in \mathcal{M}$ rather than directly in $\mathcal{Z}$. In fact, if $K = |\mathcal{Z}|$ is small, we can enumerate all possible values of z and define the guess as the latent value minimizing the downstream loss, $\mu = \operatorname{argmin}_{z \in \mathcal{Z}} L(\hat{y}(z), y)$. This is sensible, but intractable in the structured case. Moreover, early on in the training process, while g_θ is untrained, the maximizing vertex carries little information. Thus, for robustness and tractability, we allow for some uncertainty by picking a convex combination $\mu \in \mathcal{M}$ so as to approximately minimize

$$\mu \approx \operatorname*{argmin}_{\mu \in \mathcal{M}} L(\hat{y}(\mu), y). \tag{4.2}$$

For most interesting predictive models $\hat{y}(\mu)$ (*e.g.*, deep networks), this optimization problem is non-convex and lacks a closed form solution. One common strategy is the **projected gradient algorithm** (Goldstein, 1964; Levitin and Polyak, 1966), which, in addition to gradient descent, has one more step: projection of the updated point on the constraint set. It iteratively performs the following updates:

$$\mu^{(t+1)} = \Pi_{\mathcal{M}} \left(\mu^{(t)} - \eta_t \gamma(\mu^{(t)})\right) , \tag{4.3}$$

where η_t is a step size and γ is as in Eq. (2.13). With a suitable choice of step sizes, the projected gradient algorithm converges to a local optimum of Eq. (4.2) (Bertsekas, 1999, Proposition 2.3.2). In the sequel, for simplicity we use constant η. If we initialize $\mu^{(0)} = \hat{z} = \operatorname{argmax}_{z \in \mathcal{Z}} s^\top z$, **a single iteration** of projected gradient yields the guess:

$$\mu^{(1)} = \Pi_{\mathcal{M}} \left(\hat{z} - \eta \gamma(\hat{z})\right) . \tag{4.4}$$

Treating the induced μ as if it were the "ground truth" label of z, we may train the encoder $f_\phi(x)$ by **supervised learning**. With a **perceptron loss**,

$$\begin{aligned} \ell_{\text{Perc}}(\hat{z}(s), \mu) &= \max_{z \in \mathcal{Z}} s^\top z - s^\top \mu \\ &= s^\top \hat{z} - s^\top \mu \, , \end{aligned} \tag{4.5}$$

a single iteration yields the gradient:

$$\nabla_s \ell_{\text{Perc}}(\hat{z}, \mu^{(1)}) = \hat{z} - \mu^{(1)} \, , \tag{4.6}$$

which is precisely the SPIGOT gradient surrogate in Eq. (4.1). This leads to the following insight into how SPIGOT updates the encoder parameters:

SPIGOT minimizes the **perceptron loss** between z and a pulled back target computed by **one projected gradient step** on $\min\limits_{\mu \in \mathcal{M}} L(\hat{y}(\mu), y)$ starting at $\hat{z} = \texttt{MAP}(s)$.

This construction suggests possible alternatives, the first of which uncovers a well-known algorithm.

Relaxing the $\mathcal{M}$ constraint. The constraints in Eq. (4.2) make the optimization problem more complicated. We relax them and define $\mu \approx \operatorname{argmin}_{\mu \in \mathbb{R}^K} L(\hat{y}(\mu), y)$. This problem still requires iteration, but the projection step can now be avoided. One iteration of gradient descent yields $\mu^{(1)} = \hat{z} - \eta\gamma$. The perceptron update then recovers a novel derivation of **straight-through** with identity (STE-I), where the backward pass acts as if $\frac{\partial \hat{z}(s)}{\partial s} \overset{!}{=} \text{Id}$ (Bengio et al., 2013),

$$\nabla_s \ell_{\text{Perc}}(\hat{z}, \mu^{(1)}) = \hat{z} - (\hat{z} - \eta\gamma) = \eta\gamma. \tag{4.7}$$

This leads to the following insight into straight-through and its relationship to SPIGOT:

Straight-through (STE-I) minimizes the **perceptron loss** between z and a pulled back target computed by **one gradient step** on $\min\limits_{\mu \in \mathbb{R}^K} L(\hat{y}(\mu), y)$ starting at $\hat{z} = \texttt{MAP}(s)$.

From this intuition, we readily obtain new surrogate gradient methods, which we explore below.

4.4 New Surrogate Gradient Methods

Multiple gradient updates. Instead of a single projected gradient step, we could run multiple steps of Eq. (4.3). We would expect this to yield a better approximation of μ. This comes at a computational cost: each update involves running a forward and backward pass in the decoder g_θ with the current guess $\mu^{(t)}$, to obtain $\gamma(\mu^{(t)}) \coloneqq \nabla_\mu L\big(\hat{y}(\mu^{(t)}), y\big)$.

Different initialization. The projected gradient update in Eq. (4.4) uses $\mu^{(0)} = \hat{z} = \operatorname{argmax}_{z\in\mathcal{Z}} s^\top z$ as the initial point. This is a sensible choice, if we believe the encoder prediction $\hat{z}$ is close enough to the optimal μ, and it is computationally convenient, because the forward pass uses $\hat{z}$, so $\gamma(\hat{z})$ is readily available in the backward pass, thus the first inner iteration comes for free. However, other initializations are possible, for example $\mu^{(0)} = \texttt{Marg}(s)$ or $\mu^{(0)} = 0$, at the cost of an extra computation of $\gamma(\mu^{(0)})$. We do not consider alternate initializations for their own sake; they are needed for the following two directions.

Different intermediate loss: SPIGOT-CE. For simplicity, consider the unstructured case where $\mathcal{M} = \triangle$, and use the initial guess $\mu^{(0)} = \texttt{softmax}(s)$. Replacing ℓ_{Perc} by the cross-entropy loss $\ell_{\text{CE}}(\mu^{(0)}, \mu^{(1)}) = -\sum_{k=1}^{K} \mu_k \log \mu_k^{(0)}$ yields

$$\nabla_s \ell_{\text{CE}}(\mu^{(0)}, \mu^{(1)}) = \mu^{(0)} - \Pi_\triangle(\mu^{(0)} - \eta\gamma). \tag{4.8}$$

In the structured case, the corresponding loss is the CRF loss (Lafferty et al., 2001), which corresponds to the KL divergence between two distributions over structures. In this case, we initialize $\mu^{(0)} = \texttt{Marg}(s)$ and update

$$\nabla_s \ell_{\text{CE}}(\mu^{(0)}, \mu^{(1)}) = \mu^{(0)} - \Pi_{\mathcal{M}}(\mu^{(0)} - \eta\gamma). \tag{4.9}$$

Exponentiated gradient updates: SPIGOT-EG. In the unstructured case, optimization over $\mathcal{M} = \triangle$ can also be tackled via the exponentiated gradient (EG) algorithm (Kivinen and Warmuth, 1997), which minimizes Eq. (4.2) with the following multiplicative update:

$$\mu^{(t+1)} \propto \mu^{(t)} \odot \exp(-\eta_t \nabla_\mu L(\hat{y}(\mu^{(t)}), y)), \tag{4.10}$$

where $\odot$ is elementwise multiplication and thus each iterate $\mu^{(t)}$ is strictly positive, and normalized to be inside $\triangle$. EG cannot be initialized on the boundary of $\triangle$, so again we must take $\mu^{(0)} = \texttt{softmax}(s)$. A single iteration of EG yields:

$$\begin{aligned} \mu^{(1)} &\propto \mu^{(0)} \odot \exp(-\eta\gamma) \\ &= \texttt{softmax}(\log \mu^{(0)} - \eta\gamma) \\ &= \texttt{softmax}(s - \eta\gamma). \end{aligned} \tag{4.11}$$

It is natural to use the cross-entropy loss, giving

$$\nabla_s \ell_{\text{CE}}(\mu^{(0)}, \mu^{(1)}) = \mu^{(0)} - \texttt{softmax}(s - \eta\gamma), \tag{4.12}$$

i.e., the surrogate gradient is the difference between the softmax prediction and a "perturbed" softmax. To generalize to the structured case, we observe that both EG and projected gradient are instances of mirror descent under KL divergences (Beck and Teboulle, 2003). Unlike the unstructured case, we must iteratively keep track of both ***perturbed scores*** and ***marginals***, since $\texttt{Marg}^{-1}$ is non-trivial. This leads to the following mirror descent algorithm:

$$\begin{aligned} s^{(0)} &= s, \quad \mu^{(0)} = \texttt{Marg}(s^{(0)})\,, \\ s^{(t+1)} &= s^{(t)} - \eta\gamma(\mu^{(t)})\,, \\ \mu^{(t+1)} &= \texttt{Marg}(s^{(t)})\,. \end{aligned} \tag{4.13}$$

With a single iteration and the CRF loss, we get

$$\nabla_s \ell_{\text{CE}} = \texttt{Marg}(s) - \texttt{Marg}(s - \eta\gamma)\,. \tag{4.14}$$

Algorithm 1 sketches the implementation of the proposed surrogate gradients for the structured case. The forward pass is the same for all variants: given the scores s for the parts of the structure, it calculates the MAP structure z. The surrogate gradients are implemented as custom backward passes. The function GradLoss uses automatic differentiation to compute $\gamma(\mu)$ at the current guess μ; each call involves thus a forward and backward pass through g_θ. Due to convenient initialization, the first iteration of STE-I and SPIGOT come for free, since both $\mu^{(0)}$ and $\gamma(\mu^{(0)})$ are available as a byproduct when computing the forward and, respectively, backward pass through g_θ in order to update θ. For SPIGOT-CE and SPIGOT-EG, even with $k = 1$ we need a second call to the decoder, since $\mu^{(0)} \neq \hat{z}$, so an additional decoder call is necessary for obtaining the gradient of the loss with respect to $\mu^{(0)}$. The unstructured case is essentially identical, with Marg replaced by softmax.

4.5 Experiments

Armed with a selection of surrogate gradient methods, we now proceed to an experimental comparison. For maximum control, we first study a synthetic unstructured experiment with known data generating process. This allows us to closely compare the various methods, and to identify basic failure cases. We then study the structured case of latent dependency trees for sentiment analysis and natural language inference in English. Full training details are described in Appendix A.

4.5.1 Categorical Latent Variables

For the unstructured case, we design a synthetic dataset from a mixture model $z \sim \text{Categorical}(1/K)$, $x \sim \text{Normal}(m_z, \sigma I)$, $y = \text{sign}(w_z^\top x + b_z)$, where m_z are randomly

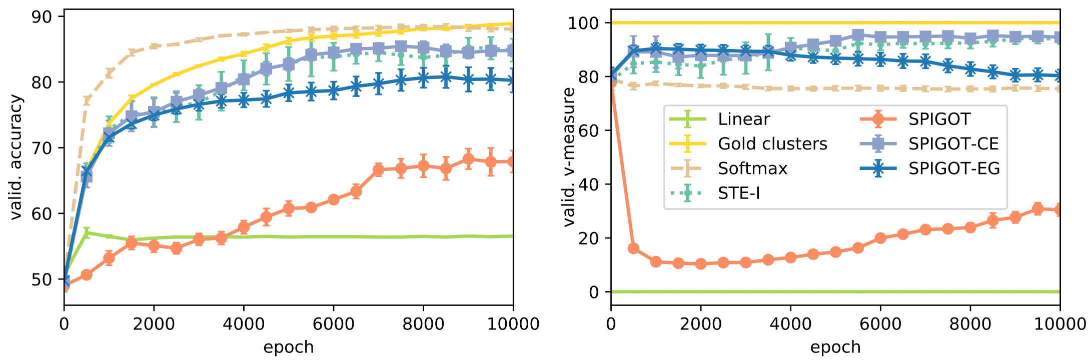


Figure 4.2: Learning curves on synthetic data with 10 clusters. Softmax learns the downstream task fast, but mixes the clusters, yielding poor V-measure. SPIGOT fails on both metrics; STE-I and the novel SPIGOT-CE work well.

placed cluster centers, and w_z, b_z are parameters of a different ground truth linear model for each cluster. Given cluster labels, one could learn the optimal linear classifier separating the data in that cluster. Without knowing the cluster, a global linear model cannot fit the data well. This setup provides a test bed for discrete variable learning, since accurate clustering leads to a good fit. The architecture, following Section 2.3.2, is:

- **Encoder**: A linear mapping from the input to a K-dimensional score vector: $s = f_\phi(x) = W_f x + b_f$, where $\phi = (W_f, b_f) \in \mathbb{R}^{K\times\dim(\mathcal{X})} \times \mathbb{R}^K$ are parameters.
- **Latent mapping**: $\hat{z} = \rho(s)$, where ρ is argmax or a continuous relaxation such as `softmax` or `sparsemax`.
- **Decoder**: A bilinear transformation, combining the input x and the latent variable z:

$$\hat{y} = g_\theta(x, \hat{z}) = \hat{z}^\top W_g x + b_g, \tag{4.15}$$

where $\theta = (W_g, b_g) \in \mathbb{R}^{K\times\dim(\mathcal{X})} \times \mathbb{R}$ are model parameters. If $\hat{z} = e_k$, this *selects* the kth linear model from the rows of W_g.

We evaluate two baselines: a linear model, and an *oracle* where $g_\theta(x, z)$ has access to the true z. In addition to the methods discussed in the previous section, we evaluate *softmax* and *sparsemax* end-to-end differentiable relaxations, and the STE-S variant which uses the `softmax` backward pass while doing `argmax` in the forward pass. We also compare *stochastic* methods, including score function estimators (with an optional moving average control variate), and the two Gumbel estimator variants (Jang et al., 2017; Maddison et al., 2016): Gumbel-Softmax with relaxed `softmax` in the forward pass, and the other using `argmax` in the style of STE (hence dubbed ST-Gumbel).

Results. We compare the discussed methods in Table 4.1. Knowledge of the data-generating process allows us to measure not only downstream accuracy, but also **clustering quality**, by comparing the model predictions with the known true z. We measure the latter via the V-measure (Rosenberg and Hirschberg, 2007), a clustering score independent of the cluster labels, *i.e.*, invariant to permuting the labels (between 0 and 100, with 100 representing perfect cluster recovery). The linear and gold cluster oracle baselines confirm that cluster separation is needed for good performance. Stochastic models perform well across both criteria. Crucially, SFE requires variance reduction to performs well, but even a simple control variate will do.

Deterministic models may be preferable when likelihood assessment or sampling is not tractable. Among these, STE-I and SPIGOT-{CE,EG} are indistinguishable from the best models. Surprisingly, the vanilla SPIGOT fails, especially in cluster recovery. Finally, the relaxed deterministic models perform very well on accuracy and learn very fast (Fig. 4.2), but appear to rely on mixing clusters, therefore they remarkably fail to recover cluster assignments.[2] This is in line with the structured results of Corro and Titov (2019b). Therefore, if latent structure recovery is less important than downstream accuracy, relaxations seem preferable.

[2]With relaxed methods, the V-measure is always calculated using the argmax, even though g_θ sees a continuous relaxation.

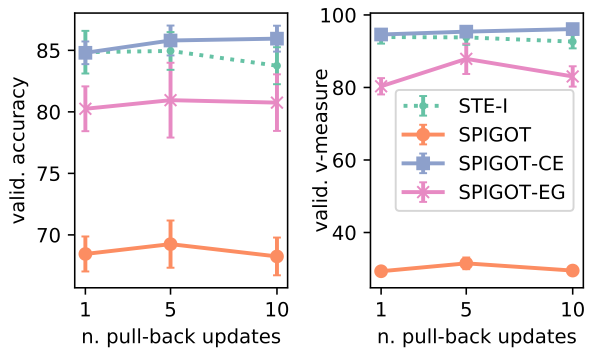


Figure 4.3: Impact of multiple gradient update steps for the pulled-back label, on the synthetic example with 10 clusters. For each point, the best step size η is chosen.

Impact of multiple updates. One possible explanation for the failure of SPIGOT is that SPIGOT-CE and SPIGOT-EG perform more work per iteration, since they use a softmax initial guess and thus require a second pass through the decoder. We rule out this possibility in Fig. 4.3: even when tuning the number of updates, SPIGOT does not substantially improve. We observe, however, that SPIGOT-CE improves slightly with more updates, outperforming STE-I. However, since each update step performs an additional decoder call, this also increases the training time.

4.5.2 Structured Latent Variables

For learning structured latent variables, we study sentiment classification on the English language Stanford Sentiment Treebank (SST) (Socher et al., 2013), and Natural Language Inference on the SNLI dataset (Bowman et al., 2015).

Sentiment Classification

The model predicts a latent projective arc-factored dependency tree for the sentence, then uses the tree in predicting the downstream binary sentiment label. The model has the following components:

- **Encoder:** Computes a score for every possible dependency arc $i \to j$ between words i and j. Each word is represented by its embedding h_i,[3] then processed by an LSTM, yielding contextual vectors $\overleftrightarrow{h_i}$. Then, arc scores are computed as

$$s_{i \to j} = v^\top \tanh\big(W^\top [\overleftrightarrow{h_i}; \overleftrightarrow{h_j}] + b\big). \tag{4.16}$$

- **Latent parser:** We use the arc scores vector s to get a parse $\hat{z} = \rho(s)$ for the sentence, where $\rho(s)$ is the `argmax`, or combination of trees, such as `Marg` or `SparseMAP`.
- **Decoder:** Following Peng et al. (2018), we concatenate each $\overleftrightarrow{h_i}$ with its predicted head $\overleftrightarrow{h}_{\text{head}(i)}$. For relaxed methods, we average all possible heads, weighted by the corresponding marginal: $\overleftrightarrow{h}_{\text{head}(i)} := \sum_j \mu_{i \to j} \overleftrightarrow{h_j}$. The concatenation is passed through an affine layer, a `ReLU` activation, an attention mechanism, and the result is fed into a linear output layer.

For marginal inference, we use `pytorch-struct` (Rush, 2020). For the SparseMAP projection, we use the active set algorithm (Niculae et al., 2018a). The baseline we compare our models against is a BiLSTM, followed by feeding the sum of all hidden states to a two-layer ReLU-MLP.

Results. The results from the experiments with the different methods are shown in Table 4.2. As in the unstructured case, the relaxed models lead to strong downstream classifiers. Unlike the unstructured case, SPIGOT is a top performer here. The effect of tuning the number of gradient update steps is not as big as in the unstructured case and did not lead to significant improvement. This can be explained by a "moving target" intuition: since the decoder g_θ is far from optimal, more accurate μ do not overall help learning.

Natural Language Inference

We build on top of the decomposable attention model (DA; Parikh et al., 2016). Following the setup of Corro and Titov (2019b), we induce structure on the premise

[3]Pretrained GloVe vectors (Pennington et al., 2014).

and the hypothesis. For computing the score of the arc from word i to j, we concatenate the representations of the two words, as in Eq. (4.16). In the *decoder*, after the latent parse tree is calculated, we concatenate each word with the average of its heads. We do this separately for the premise and the hypothesis. As baseline, we use the DA model with no intra-attention.

Results. The SNLI results are shown in Table 4.2. Here, the straight-through (argmax) methods are outperformed by the more stable relaxation-based methods. This can be attributed to the word-level alignment in the DA model, where soft dependency relations appear better suited than hard ones.

4.6 Conclusions

In this chapter, we provide a novel motivation for straight-through estimator (STE) and SPIGOT, based on pulling back the downstream loss. We derive promising new algorithms, and novel insight into existing ones. Unstructured controlled experiments suggest that our new algorithms, which use the cross-entropy loss instead of the perceptron loss, can be more stable than SPIGOT while accurately disentangling the latent variable. Differentiable relaxation models (using softmax and sparsemax) are the easiest to optimize to high downstream accuracy, but they fail to correctly identify the latent clusters. On structured NLP experiments, relaxations (SparseMAP and Marginals) tend to overall perform better and be more stable than straight-through variants in terms of classification accuracy. However, the lack of gold-truth latent structures makes it impossible to assess recovery performance. We hope that our insights, including some of our negative results, may encourage future research on learning with latent structures.

Algorithm 1: Surrogate gradients pseudocode: common forward pass, specialized backward passes.

```
Parameters: step size $\eta$, n. iterations $k$

Function Forward($s$, $x$, $y$):
    return $\hat{z} \leftarrow$ MAP($s$)                                  // Eq. (2.9)

Function GradLoss($\mu, x, y$):
    return $\gamma \leftarrow \nabla_\mu L(\hat{y}(\mu), y)$              // Eq. (2.13)

Function BackwardSPIGOT($s, x, y$):
    $\mu^{(0)} =$ MAP($s$)
    for $t \leftarrow 1$ to $k$ do
        $\gamma \leftarrow$ GradLoss($\mu^{(t-1)}, x, y$)
        $\mu^{(t)} \leftarrow \Pi_{\mathcal{M}}(\mu^{(t-1)} - \eta\gamma)$   // Eq. (4.3)
    return $\mu^{(0)} - \mu^{(k)}$                                       // Eq. (4.6)

Function BackwardSTE-I($s, x, y$):
    $\mu^{(0)} =$ MAP($s$)                                                // Eq. (4.7)
    for $t \leftarrow 1$ to $k$ do
        $\gamma \leftarrow$ GradLoss($\mu^{(t-1)}, x, y$)
        $\mu^{(t)} \leftarrow \mu^{(t-1)} - \eta\gamma$
    return $\mu^{(0)} - \mu^{(k)}$

Function BackwardSPIGOT-CE($s, x, y$):
    $\mu^{(0)} \leftarrow$ Marg($s$)                                      // Eq. (4.9)
    for $t \leftarrow 1$ to $k$ do
        $\gamma \leftarrow$ GradLoss($\mu^{(t-1)}, x, y$)
        $\mu^{(t)} \leftarrow \Pi_{\mathcal{M}}(\mu^{(t-1)} - \eta\gamma)$
    return $\mu^{(0)} - \mu^{(k)}$

Function BackwardSPIGOT-EG($s, x, y$):
    $(s^{(0)}, \mu^{(0)}) \leftarrow (s,$ Marg($s$)$)$                     // Eq. (4.13)
    for $t \leftarrow 1$ to $k$ do
        $\gamma \leftarrow$ GradLoss($\mu^{(t-1)}, x, y$)
        $s^{(t)} \leftarrow s^{(t-1)} - \eta\gamma$
        $\mu^{(t)} \leftarrow$ Marg($s^{(t)}$)
    return $\mu^{(0)} - \mu^{(k)}$
```

	(3 clusters)		(10 clusters)	
Model	**Accuracy**	**V-measure**	**Accuracy**	**V-measure**
Baselines				
Linear model	68.05±0.09	0.00±0.00	60.00±0.06	0.00±0.00
Gold cluster labels	92.40±0.06	100.00±0.00	88.50±0.10	100.00±0.00
Relaxed				
Softmax	93.15±0.33	66.88±0.97	86.45±0.33	75.07±1.18
Sparsemax	92.95±0.38	71.35±16.60	83.75±1.32	76.13±3.89
*Gumbel-Softmax	94.25±3.42	100.00±6.80	80.45±0.77	89.68±1.10
Argmax				
*ST-Gumbel	93.85±3.25	100.00±6.80	81.25±0.68	91.52±1.46
*SFE	68.45±0.33	47.73±17.65	59.80±0.58	55.56±3.30
*SFE w/ baseline	94.20±0.08	100.00±0.00	84.70±0.97	96.83±0.85
STE-S	86.95±4.01	84.44±11.61	75.95±1.10	82.83±2.75
STE-I	92.60±0.23	100.00±0.00	84.50±1.43	94.48±1.35
SPIGOT	77.90±1.26	20.53±1.85	68.80±1.02	29.24±2.24
SPIGOT-CE	93.40±2.64	97.08±13.92	83.50±0.87	94.88±1.39
SPIGOT-EG	92.70±3.04	100.00±8.27	79.40±2.03	82.29±2.15

Table 4.1: Discrete latent variable learning on synthetic data: downstream accuracy and clustering V-measure. Median and standard error reported over four runs. We mark stochastic methods with *.

Model	**SNLI**	**SST**
Relaxed		
Marginals	83.45	85.01
SparseMAP	83.61	**85.35**
Argmax		
*Perturb-and-MAP	82.92	83.80
STE-S	83.32	81.10
STE-I	83.17	81.00
SPIGOT	**84.80**	83.52
SPIGOT-CE	83.01	79.20
SPIGOT-EG	82.88	84.84

Table 4.2: SST and SNLI average accuracy over three runs, with latent dependency trees. Baselines are described in Section 4.5.2. We mark stochastic methods marked with *.

CHAPTER 5

Undirected Neural Networks

Contents

Neural networks are powerful function estimators, leading to their status as a paradigm of choice for modeling structured data. However, unlike other structured representations that emphasize the modularity of the problem – *e.g.*, factor graphs – neural networks are usually monolithic mappings from inputs to outputs, with a fixed computation order. This limitation prevents them from capturing different directions of computation and interaction between the modeled variables.

In this chapter, we combine the representational strengths of factor graphs and of neural networks, proposing ***undirected neural networks (UNNs)***: a flexible framework for specifying computations that can be performed in any order. For particular choices, our proposed models subsume and extend many existing architectures: feed-forward, recurrent, self-attention networks, auto-encoders, and networks with implicit layers. We demonstrate the effectiveness of undirected neural architectures, both unstructured and structured, on a range of tasks: tree-constrained dependency parsing, convolutional image classification, and sequence completion with attention. By varying the computation order, we show how a single UNN can be used both as a classifier and a prototype generator, and how it can fill in missing parts of an input sequence, making them a promising field for further research.

5.1 Introduction

Factor graphs have historically been a very appealing toolbox for representing structured prediction problems (Bakır et al., 2007; Smith, 2011; Nowozin et al., 2014), with wide applications to vision and natural language processing applications. In the last years, neural networks have taken over as the model of choice for tackling these applications. Unlike factor graphs – which emphasize the modularity of the problem – neural networks typically work end-to-end, relying on rich representations captured at the encoder level (often pretrained), which are then propagated to a task-specific decoder.

In this chapter, we combine the representational strengths of factor graphs and neural networks by proposing **undirected neural networks** (UNNs) – a framework in which outputs are not computed by evaluating a composition of functions in a given order, but are rather obtained *implicitly* by minimizing an energy function which factors over a graph. For particular choices of factor potentials, UNNs subsume many existing architectures, including feedforward, recurrent, and self-attention neural networks, auto-encoders, and networks with implicit layers. When coupled with a coordinate-descent algorithm to minimize the energy, the computation performed in an UNN is similar (but not equivalent) to a neural network sharing parameters across multiple identical layers. Since UNNs have no prescribed computation order, the exact same network can be used to predict any group of variables (outputs) given another group of variables (inputs), or vice-versa (*i.e.*, inputs from outputs). Our contributions are:

- We present UNNs and show how they extend many existing neural architectures.
- We provide a coordinate descent inference algorithm, which, by an "unrolling lemma" (Lemma 1), can reuse current building blocks from feed-forward networks in a modular way.
- We develop and experiment with multiple factor graph architectures, tackling both structured and unstructured tasks, such as natural language parsing, image classification, and image prototype generation. We develop a new undirected attention mechanism and demonstrate its suitability for sequence completion.[1]

5.2 Undirected Neural Networks

Let $\mathcal{G} = (V, F)$ be a factor graph, *i.e.*, a bipartite graph consisting of a set of variable nodes V and a set of factor nodes F, where each factor node $f \in F \subseteq 2^V$ is linked

[1]The source code is available on: https://github.com/deep-spin/unn

to a subset of variable nodes. Each variable node $\mathsf{X} \in V$ is associated with a representation vector $x \in \mathbb{R}^{d_\mathsf{X}}$. We define unary energies for each variable $E_\mathsf{X}(x)$, as well as higher-order energies $E_f(x_f)$, where x_f denotes the values of all variables linked to factor f. Then, an assignment defines a total energy function

$$E(x_1, \dots, x_n) \coloneqq \sum_i E_{\mathsf{X}_i}(x_i) + \sum_f E_f(x_f). \tag{5.1}$$

For simple factor graphs where there is no ambiguity, we may refer to factors directly by the variables they link to. For instance, a simple fully-connected factor graph with only two variables X and Y is fully specified by $E(x, y) = E_\mathsf{X}(x) + E_\mathsf{Y}(y) + E_{\mathsf{XY}}(x, y)$.

The energy function in Eq. (5.1) induces preferences for certain configurations. For instance, a globally best configuration can be found by solving $\operatorname{argmin}_{x,y} E(x, y)$, while a best assignment for Y given a fixed value of X can be found by solving $\operatorname{argmin}_y E(x, y)$.[2] We may think of, or suggest using notation, that X is an ***input*** and Y is an ***output***. However, intrinsically, factor graphs are not attached to a static notion of input and output, and instead can be used to infer any subset of variables given any other subset.

In our proposed framework of UNNs, we define the computation performed by a neural network using a factor graph, where each variable is a representation vector (***e.g.***, analogous to the output of a layer in a standard network). We design the factor energy functions depending on the type of each variable and the desired relationships between them. Inference is performed by minimizing the joint energy with respect to all unobserved variables (***i.e.***, hidden and output values). For instance, to construct a supervised UNN, we may designate a particular variable as "input" X and another as "output" Y, alongside several hidden variables H_i,

[2]We only consider deterministic inference in factor graphs. Probabilistic models are a promising extension.

compute

$$\hat{y} = \arg\min_{y} \min_{h_1,\ldots,h_n} E(x, h_1, \ldots, h_n, y)\,, \tag{5.2}$$

and train by minimizing some loss $\ell(\hat{y}, y)$. However, UNNs are not restricted to the supervised setting or to a single input and output, as we shall explore.

While this framework is very flexible, Eq. (5.2) is a non-trivial optimization problem. Therefore, we focus on a class of energy functions that renders inference easier:

$$\begin{aligned} E_{\mathsf{X}_i}(x_i) &= -\langle b_{\mathsf{X}_i}, x_i\rangle + \Psi_{\mathsf{X}_i}(x_i)\,, \\ E_f(x_f) &= -\left\langle W_f, \bigotimes_{\mathsf{X}_j \in f} x_j \right\rangle\,, \end{aligned} \tag{5.3}$$

where each Ψ_{X_i} is a strictly convex regularizer, $\otimes$ denotes the outer product, and W_f is a parameter tensor of matching dimension. For pairwise factors $f = \{\mathsf{X}, \mathsf{Y}\}$, the factor energy is bilinear and can be written simply as $E_{\mathsf{XY}}(x, y) = -x^\top W y$. In factor graphs of the form given in Eq. (5.3), the energy is convex in each variable separately, and block-wise minimization has a closed-form expression involving the Fenchel conjugate of the regularizers. This suggests a block coordinate descent optimization strategy: given an order π, iteratively set:

$$x_{\pi_j} \leftarrow \operatorname*{argmin}_{x_{\pi_j}} E(x_1, \ldots, x_n)\,. \tag{5.4}$$

This block coordinate descent algorithm is guaranteed to decrease energy at every iteration and, for energies as in Eq. (5.3), to converge to a Nash equilibrium (Xu and Yin, 2013, Thm. 2.3); in addition, it is conveniently learning-rate free. For training, to tackle the bi-level optimization problem, we unroll the coordinate descent iterations, and minimize some loss with standard deep learning optimizers, like stochastic gradient or Adam (Kingma and Ba, 2014).

The following result, proved in Appendix C, shows that the coordinate descent algorithm (Eq. (5.4)) for UNNs with multilinear factor energies (Eq. (5.3)), corresponds to standard forward propagation on an unrolled neural network.

Lemma 1 (Unrolling Lemma). *Let $\mathcal{G} = (V, F)$ be a pairwise factor graph, with multilinear higher-order energies and strictly convex unary energies, as in Eq.* (5.3). *Then, the coordinate descent updates* (5.4) *result in a chain of affine transformations (i.e., pre-activations) followed by non-linear activations, applied in the order π, yielding a traditional computation graph.*

We show next an undirected construction inspired by (directed) multi-layer perceptrons.

Single pairwise factor The simplest possible UNN has a pairwise factor connecting two variables X, H. We may interpret X as an input, and H either as an output (in supervised learning) or a hidden representation in unsupervised learning (Fig. 5.2(a)). Bilinear-convex energies as in Eq. (5.3) yield:

$$\begin{aligned} E_{\mathsf{XH}}(x, h) &= -\langle h, Wx \rangle \,, \\ E_{\mathsf{X}}(x) &= -\langle x, b_{\mathsf{X}} \rangle + \Psi_{\mathsf{X}}(x) \,, \\ E_{\mathsf{H}}(h) &= -\langle h, b_{\mathsf{H}} \rangle + \Psi_{\mathsf{H}}(h) \,. \end{aligned} \tag{5.5}$$

This resembles a Boltzmann machine with continuous variables (Smolensky, 1986; Hinton, 2007; Welling et al., 2004); however, in contrast to Boltzmann machines, we do not model joint probability distributions, but instead use factor graphs as representations of deterministic computation, more akin to computation graphs.

Given x, the updated h minimizing the energy is:

$$\begin{aligned} h_\star &= \operatorname*{argmin}_{h \in \mathbb{R}^M} -(Wx + b_{\mathsf{H}})^\top h + \Psi_{\mathsf{H}}(h) \\ &= (\nabla \Psi^*_{\mathsf{H}})(Wx + b_{\mathsf{H}}), \end{aligned} \tag{5.6}$$

where $\nabla \Psi^*_{\mathsf{H}}$ is the gradient of the conjugate function of Ψ_{H}. Analogously, the update for X given H is:

$$x_\star = (\nabla \Psi^*_{\mathsf{X}})(W^\top h + b_{\mathsf{X}}) \,. \tag{5.7}$$

$\Psi(h)$	$(\nabla\Psi^*)(t)$
$\frac{1}{2}\|h\|^2$	t
$\frac{1}{2}\|h\|^2 + \iota_{\mathbb{R}_+}(h)$	$\operatorname{relu}(t)$
$\sum_j \big(\phi(h_j) + \phi(1-h_j)\big) + \iota_{[0,1]^d}(h)$	$\operatorname{sigmoid}(t)$
$\sum_j \left(\phi\left(\frac{1+h_j}{2}\right) + \phi\left(\frac{1-h_j}{2}\right)\right) + \iota_{[-1,1]^d}(h)$	$\tanh(t)$
$-\mathcal{H}(h) + \iota_\Delta(h)$	$\operatorname{softmax}(t)$

Table 5.1: Examples of regularizers $\Psi(h)$ corresponding to some common activation functions, where $\phi(t) = t \log t$.

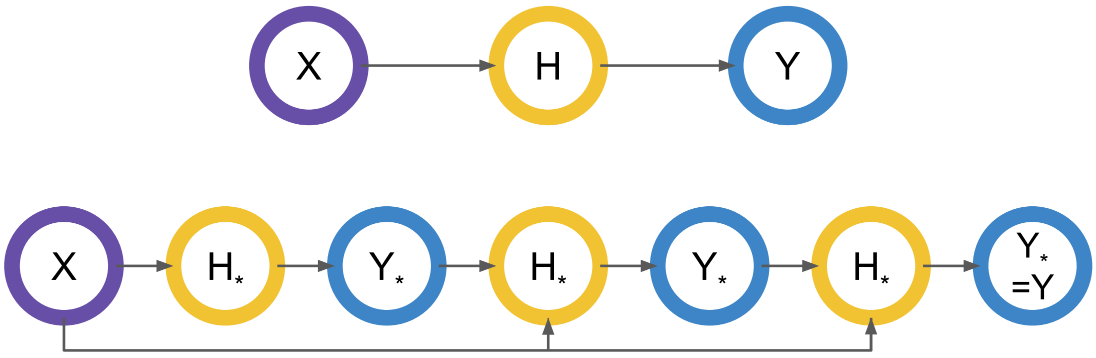


Figure 5.1: Unrolling the computation graph for undirected MLP with a single hidden layer. Top: MLP with one hidden layer. Bottom: Unrolled graph for UNN with $k = 3$ iterations.

Other than the connection to Boltzmann machines, one round of updates of H and X in this order also describe the computation of an auto-encoder with shared encoder/decoder weights.

Table 5.1 shows examples of regularizers Ψ and their corresponding $\nabla\Psi^*$, In practice, we never evaluate Ψ or Ψ^*, but only $\nabla\Psi^*$, which we choose among commonly-used neural network activation functions like $\tanh$, relu, and $\operatorname{softmax}$.

Undirected multi-layer perceptron (MLP) Fig. 5.2(b) shows the factor graph for an undirected MLP analogous to a feed-forward one with input X, output Y, and a single hidden layer H. As in Eq. (5.3), we have bilinear pairwise factors

$$E_{\mathsf{XH}}(x,h) = -\langle h, Wx\rangle\,, \quad E_{\mathsf{HY}}(h,y) = -\langle y, Vh\rangle\,, \tag{5.8}$$

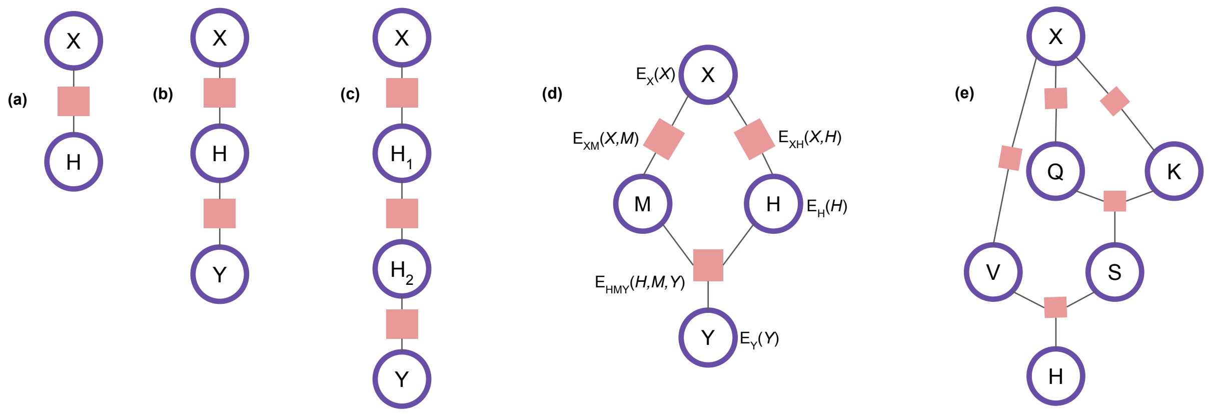


Figure 5.2: Factor graphs for: (a) network without intermediate layers, (b-c) undirected MLPs with one or two layers, (d) undirected biaffine dependency parser, (e) undirected self-attention. Energy labels ommitted for brevity with the exception of (d).

and linear-plus-convex unaries $E_{\mathsf{Z}}(x) = -\langle x, b_{\mathsf{Z}}\rangle + \Psi_{\mathsf{Z}}(x)$ for $\mathsf{Z} \in \{\mathsf{X}, \mathsf{H}, \mathsf{Y}\}$. If x is observed (fixed), coordinate-wise inference updates take the form:

$$\begin{aligned} h_\star &= (\nabla\Psi^*_{\mathsf{H}})(Wx + V^\top y + b_{\mathsf{H}})\,, \\ y_\star &= (\nabla\Psi^*_{\mathsf{Y}})(Vh + b_{\mathsf{Y}})\,. \end{aligned} \tag{5.9}$$

Note that E_{X} does not change anything if X is always observed. The entire algorithm can be unrolled into a directed computation graph, leading to a deep neural network with shared parameters (Fig. 5.1).

The regularizers Ψ_{H} and Ψ_{Y} may be selected based on what we want $\nabla\Psi^*$ to look like, and the constraints or domains of the variables. For instance, if Y is a multiclass classification output, we may pick Ψ_{Y} such that $\nabla\Psi^*_{\mathsf{Y}}$ be the softmax function, and Ψ_{H} to induce a relu nonlinearity. Initializing $y^{(0)} = 0$ and performing a single iteration of updating H followed by Y results in a standard MLP with a single hidden layer (see also Fig. 5.1). However, the UNN point of view lets us decrease energy further by performing multiple iterations, as well as use the same model to infer any variables given any other ones, *e.g.*, to predict x from y instead of y from x. We demonstrate this power in Sections 5.3 to 5.5.

The above constructions provide a flexible framework for defining UNNs. However, UNNs are more general and cover more popular deep learning architectures. The following constructions illustrate some such connections.

Feed-forward neural networks Any directed computation graph associated with a neural network is a particular case of an UNN. We illustrate this for a simple feed-forward network, which chains the functions $h = f(x)$ and $y = g(h)$, where $x \in \mathbb{R}^m$, $h \in \mathbb{R}^d$, $y \in \mathbb{R}^n$ are input, hidden, and output variables, and $f : \mathbb{R}^m \to \mathbb{R}^d$ and $g : \mathbb{R}^d \to \mathbb{R}^n$ are the functions associated to each layer (*e.g.*, an affine transformation followed by a non-linearity). This factor graph is illustrated in Fig. 5.2(b). To see this, let $V = \{\mathsf{X}, \mathsf{H}, \mathsf{Y}\}$ and $F = \{\mathsf{XH}, \mathsf{HY}\}$ and define the energies as follows. Let $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$ be any distance function satisfying $d(a, b) \geq 0$, with equality iff $a = b$; for example $d(a, b) = \|a - b\|$. Let all the unary energies be zero and define the factor energies $E_{\mathsf{XH}}(x, h) = d(h, f(x))$ and $E_{\mathsf{HY}}(h, y) = d(y, g(h))$. Then the total energy satisfies $E(x, h, y) \geq 0$, with equality iff the equations $h = f(x)$ and $y = g(h)$ are satisfied – therefore, the energy is minimized (and becomes zero) when $y = g(f(x))$, matching the corresponding directed computation graph. This can be generalized for an arbitrary deterministic neural network. This way, we can form UNNs that are partly directed, partly undirected, as the whole is still an UNN. We do this in our experiments in Section 5.5, where we fine-tune a pre-trained BERT model appended to a UNN for parsing.

Implicit layers UNNs include networks with implicit layers (Duvenaud et al., 2020), a paradigm which, in contrast with feed-forward layers, does not specify how to compute the output from the input, but rather specifies conditions that the output layer should specify, often related to minimizing some function, *e.g.*, computing a layer h_{i+1} given a previous layer h_i involves solving a possibly difficult problem $\operatorname{argmin}_h f(h_i, h)$. Such a function f can be directly interpreted as an energy in our model, ***i.e.***, $E_{\mathsf{H}_i\mathsf{H}_{i+1}} = f$.

5.3 Image Classification and Visualization

Unlike feed-forward networks, where the processing order is hard-coded from inputs X to outputs Y, UNNs support processing in any direction. We can thus use the same trained network both for classification as well as for generating prototypical examples for each class. We demonstrate this on the MNIST dataset of handwritten digits (LeCun et al., 1998), showcasing convolutional UNN layers.

The architecture is shown in Fig. 5.2(c) and has the following variables: the image X, the class label Y and two hidden layer variables $\mathsf{H}_{\{1,2\}}$. Unlike the previous examples, the two pairwise energies involving the image and the hidden layers are convolutional, ***i.e.***, linear layers with internal structure:

$$\begin{aligned} E_{\mathsf{XH}_1}(X, H_1) &= -\langle H_1, \mathcal{C}_1(X; W_1)\rangle\,, \\ E_{\mathsf{H}_1\mathsf{H}_2}(H_1, H_2) &= -\langle H_2, \mathcal{C}_2(H_1; W_2)\rangle\,, \end{aligned} \tag{5.10}$$

where $\mathcal{C}_{1,2}$ are linear cross-correlation operators with stride two and filter weights $W_1 \in \mathbb{R}^{32\times1\times6\times6}$ and $W_2 \in \mathbb{R}^{64\times32\times4\times4}$. The last layer is fully connected:

$$E_{\mathsf{H}_2\mathsf{Y}}(H_2, y) = -\langle V,\ y\ \otimes H_2\rangle\,. \tag{5.11}$$

The unary energies for the hidden layers contain standard (convolutional) bias term along with the binary entropy term $\Psi_{\text{tanh}}(H)$ such that $\nabla\Psi^*_{\text{tanh}}(t) = \tanh(t)$ (see Table 5.1). Note that X is no longer a constant when generating X given Y, therefore it is important to specify the unary energy E_{X}. Since pixel values are bounded, we set $\mathsf{E}_X(x) = \Psi_{\text{tanh}}(x)$. Initializing $H_1, H_2,$ and y with zeroes and updating them once blockwise in this order yields exactly a feed-forward convolutional neural network. As our network is undirected, we may propagate information in multiple passes, proceeding in the order $\mathsf{H}_1, \mathsf{H}_2, \mathsf{Y}, \mathsf{H}_2$ iteratively. The update for H_1 involves a convolution of X and a deconvolution of H_2; we defer the other updates

Iterations	Accuracy
$k = 1, \gamma = 0$ (baseline)	98.80
$k = 1$	98.75
$k = 2$	98.74
$k = 3$	**98.83**
$k = 4$	98.78
$k = 5$	98.69

Table 5.2: MNIST accuracy with convolutional UNN.

to Appendix D:

$$(H_1) \underset{\star}{=} \tanh(\mathcal{C}_1(X; W_1) + \mathcal{C}_2^\top(H_2; W_2) + b_1 \otimes 1_{d_1})\,, \tag{5.12}$$

where $b_1 \in \mathbb{R}^{32}$ are biases for each filter, and $d_1 = 12 \times 12$ is the convolved image size. To generate digit prototype X from a given class $c \in \{1, \dots, 10\}$, we may set $y = e_c$, initialize the other variables at zero (including X), and solve $\hat{X} = \operatorname{argmin}_X \min_{H_{1,2}} E(X, H_1, H_2, y)$ by coordinate descent in the reverse order $\mathsf{H}_2, \mathsf{H}_1, \mathsf{X}, \mathsf{H}_1$ iteratively.

We train our model jointly for both tasks. For each labeled pair (X, y) from the training data, we first predict $\hat{y}$ given X, then separately predict $\hat{X}$ given y. The incurred loss is a weighted combination $\ell(x, y) = \ell_f(y, \hat{y}) + \gamma \ell_b(X, \hat{X})$, where ℓ_f is a 10-class cross-entropy loss, and ℓ_b is a binary cross-entropy loss averaged over all 28×28 pixels of the image. We use $\gamma = .1$ and an Adam learning rate of $.0005$.

The classification results are shown in Table 5.2. The model is able to achieve high classification accuracy, and multiple iterations lead to a slight improvement. This result suggests the reconstruction loss for X can also be seen as a regularizer, as the same model weights are used in both directions. The more interesting impact of multiple energy updates is the image prototype generation. In Fig. 5.3 we show the generated digit prototypes after several iterations of energy minimization, as well as for models with a single iteration. The networks trained as UNNs produce recognizable digits, and in particular the model with more iter-

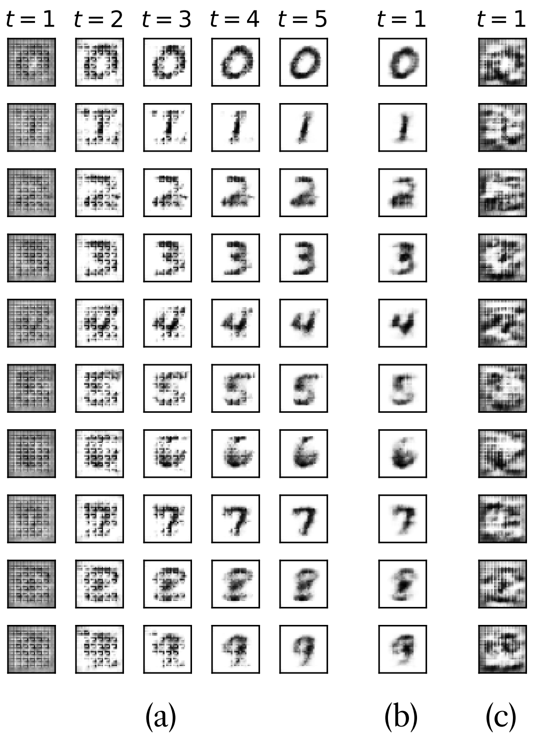


Figure 5.3: Digit prototypes generated by convolutional UNN. (a) best UNN ($k = 5, \alpha = .1$), (b) single iteration UNN ($k = 1, \alpha = .1$), (c): standard convnet ($k = 1, \alpha = 0$).

ations learns to use the additional computation to produce clearer pictures. As for the baseline, we may interpret it as an UNN and apply the same process to extract prototypes, but this does not result in meaningful digits (Fig. 5.3c). Note that our model is not a generative model – in that experiment, we are not sampling an image according to a probability distribution, rather we are using energy minimization deterministically to pick a prototype of a digit given its class.

Results for forward-only UNN In addition to the results for forward-backward training of the UNN, we also report results from training the UNN only in forward mode with $\gamma = 0$, i.e. when the model is trained for image classification only. The results are in Table 5.3.

Comparison of UNN to Unconstrained Model As per Fig. 5.1, an unrolled UNN can be seen as a feed-forward network with a specific architecture and with weight tying. To confirm the benefit of the UNN framework, we compare against

Iterations	Accuracy
$k = 1, \gamma = 0$ (baseline)	98.80
$k = 2$	**98.82**
$k = 3$	98.75
$k = 4$	98.74
$k = 5$	98.69

Table 5.3: MNIST accuracy with convolutional UNN in forward-only mode (*i.e.* $\gamma = 0$).

an unconstrained model, *i.e.*, with the same architecture but separate, untied weights for each unrolled layer. We use as a base the model described in forward-only mode and train a model with 2 to 5 layers with different weights instead of shared weights as in the case with the UNN. Depending on the number of layers, we cut the number of parameters in each layer, in order to obtain models with the same number of parameters as the UNN for fair comparison. The results from the experiment are described in Table 5.4.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Accuracy UNN	98.80	98.82	98.75	98.74	98.69
Accuracy unc.		98.76	98.45	98.32	97.39
# params unc.	50026	51220	51651	53608	51750

Table 5.4: Comparison of UNN with an unconstrained model (unc.) with the same number of layers as the UNN iterations. The number of parameters of the UNN and the unconstrained model is roughly the same.

Analysis of Alternative Initialization Strategies In addition to the zero initialization for the output variable y, we also experiment with two more initialization strategies - random and uniform initialization. For the random initialization, we initialize y with random numbers from a uniform distribution on the interval $[0, 1)$ and apply $\mathrm{softmax}$. For the uniform initialization, we assign equal values summing to one. We compare to the zero initialization strategy on the MNIST forward-backward experiment with $\gamma = .1$. The results are presented in Table 5.5. Random initialization shows promise, but the differences are small, and zero-init has the advantage of clearer parallels to the feed-forward case, so we report that and use it throughout all other experiments. The alternative initialization strategies can be further explored in further work.

Initialization	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
zero	98.75	98.74	98.83	98.78	98.69
random	98.77	98.83	98.90	98.85	98.70
uniform	98.76	98.74	98.83	98.78	98.68

Table 5.5: Comparison of different initialization strategies for the MNIST experiment.

5.4 Undirected Attention Mechanism

Attention (Bahdanau et al., 2014; Vaswani et al., 2017) is a key component that enables neural networks to handle variable-length sequences as input. In this section, we propose an undirected attention mechanism (Fig. 5.2(e)). We demonstrate this model on the task of completing missing values in a sequence of dynamic length n, with the variable X serving as both input and output, taking values $X \in \mathbb{R}^{d \times n}$, queries, keys and values taking values $Q, K, V \in \mathbb{R}^{n \times d}$, and attention weights $S \in (\triangle_n)^n$, where d is a fixed hidden layer size. Finally, H is an induced latent sequence representation, with values $H \in \mathbb{R}^{n \times d}$. The only trainable parameters are $W_{\mathsf{Q}}, W_{\mathsf{K}}, W_{\mathsf{V}} \in \mathbb{R}^{d \times d}$, and the input embeddings. We model scaled dot-product attention given with $\operatorname{softmax}(d^{-\frac{1}{2}} QK^\top)V$. For all variables except S, we set $E_{\cdot}(\cdot) = \frac{1}{2}\|\cdot\|^2$. For the attention weights, we use $E_{\mathsf{S}}(S) = -\sqrt{d}\sum_{i=1}^{n} \mathcal{H}(S_i)$.

The higher-order energies are:

$$\begin{aligned}
E_{\mathsf{XQ}}(X, Q) &= -\langle Q, W_{\mathsf{Q}}(X+P)\rangle \,, \\
E_{\mathsf{XK}}(X, K) &= -\langle K, W_{\mathsf{K}}(X+P)\rangle \,, \\
E_{\mathsf{XV}}(X, V) &= -\langle V, W_{\mathsf{V}}(X+P)\rangle \,, \\
E_{\mathsf{QKS}}(Q, K, S) &= -\langle S, QK^\top\rangle \,, \\
E_{\mathsf{VSH}}(V, S, H) &= -\langle H, SV\rangle \,,
\end{aligned} \tag{5.13}$$

where P is a matrix of sine and cosine positional embeddings of same dimensions as X Vaswani et al. (2017).

Minimizing the energy yields the blockwise updates:

$$\begin{aligned}
Q_\star &= W_{\mathsf{Z}}(X+P) + SK \,, \\
K_\star &= W_{\mathsf{K}}(X+P) + S^\top Q \,, \\
V_\star &= W_{\mathsf{V}}(X+P) + S^\top H \,, \\
S_\star &= \operatorname{softmax}\left(d^{-1/2}(QK^\top + VH^\top)\right) \,, \\
H_\star &= SV \,, \\
\bar{X}_\star &= \bar{V}W_{\mathsf{V}} + \bar{Q}W_{\mathsf{Q}} + \bar{K}W_{\mathsf{K}},
\end{aligned} \tag{5.14}$$

where $\bar{X}$ denotes only the rows of X corresponding to the masked (missing) entries.

Provided zero initialization, updating in the order $(\mathsf{V}/\mathsf{Q}/\mathsf{K}), \mathsf{S}, \mathsf{H}$ corresponds exactly to a forward pass in a standard self-attention. However, in an UNN, our expressions allow backward propagation back toward X, as well as iterating to an equilibrium. To ensure that one round of updates propagates information through all the variables, we employ the "forward-backward" order $\mathsf{Q}, \mathsf{K}, \mathsf{V}, \mathsf{S}, \mathsf{H}, \mathsf{S}, \mathsf{V}, \mathsf{K}, \mathsf{Q}, \bar{\mathsf{X}}$.

We evaluate the performance of the undirected attention with a toy task of sequence completion. We generate a toy dataset of numerical sequences between 1 and 64, of length at least 8 and at most 25, in either ascending or descending consecutive order. We mask out up to 10% of the tokens and generate all possible sequences, splitting them into training and test sets with around 706K and 78K instances. Undirected self-attention is applied to the input sequence. Note that because of the flexibility of the architecture, the update of the input variable X does not differ from the updates of the remaining variables, because each variable update corresponds to one step of coordinate descent. The model incurs a cross-entropy loss for the missing elements of the sequence and the parameters are updated using Adam with learning rate 10^{-4}. The hidden dimension is $d = 256$, and gradients with magnitude beyond 10 are clipped.

Undirected attention is able to solve this task, reaching over 99.8% test accuracy, confirming viablity.

Figure 5.4 show examples of the weights of the undirected self-attention. The attention weights are the values of the variable S calculated in the forward and backward pass.

We next analyze how the order of variable updates and the number of update passes during training affect the model performance.

Order of variable updates. We showed that one pass of the "forward-backward" order or variable updates ($\mathsf{Q}, \mathsf{K}, \mathsf{V}, \mathsf{S}, \mathsf{H}, \mathsf{S}, \mathsf{V}, \mathsf{K}, \mathsf{Q}, \hat{\mathsf{X}}$) performs well enough for the of sequence completion. Since the flexibility of our model does not limit us to a specific order, we compare it to a random order of updating the variables (a permutation of $\mathsf{Q}, \mathsf{K}, \mathsf{V}, \mathsf{S}, \mathsf{H}$; $\hat{\mathsf{X}}$ is always updated last). One pass over the "forward-backward" order performs nine variable updates, and one pass over the random order - five. In Fig. 5.6 we show how the two ways of order perform for different number of variable updates (for example, 2 passes over the "forward-backward" model equal 18 variable updates, and over the random model - 10). The "forward-backward" order performs best, but the random order can achieve similar performance after enough number of updates.

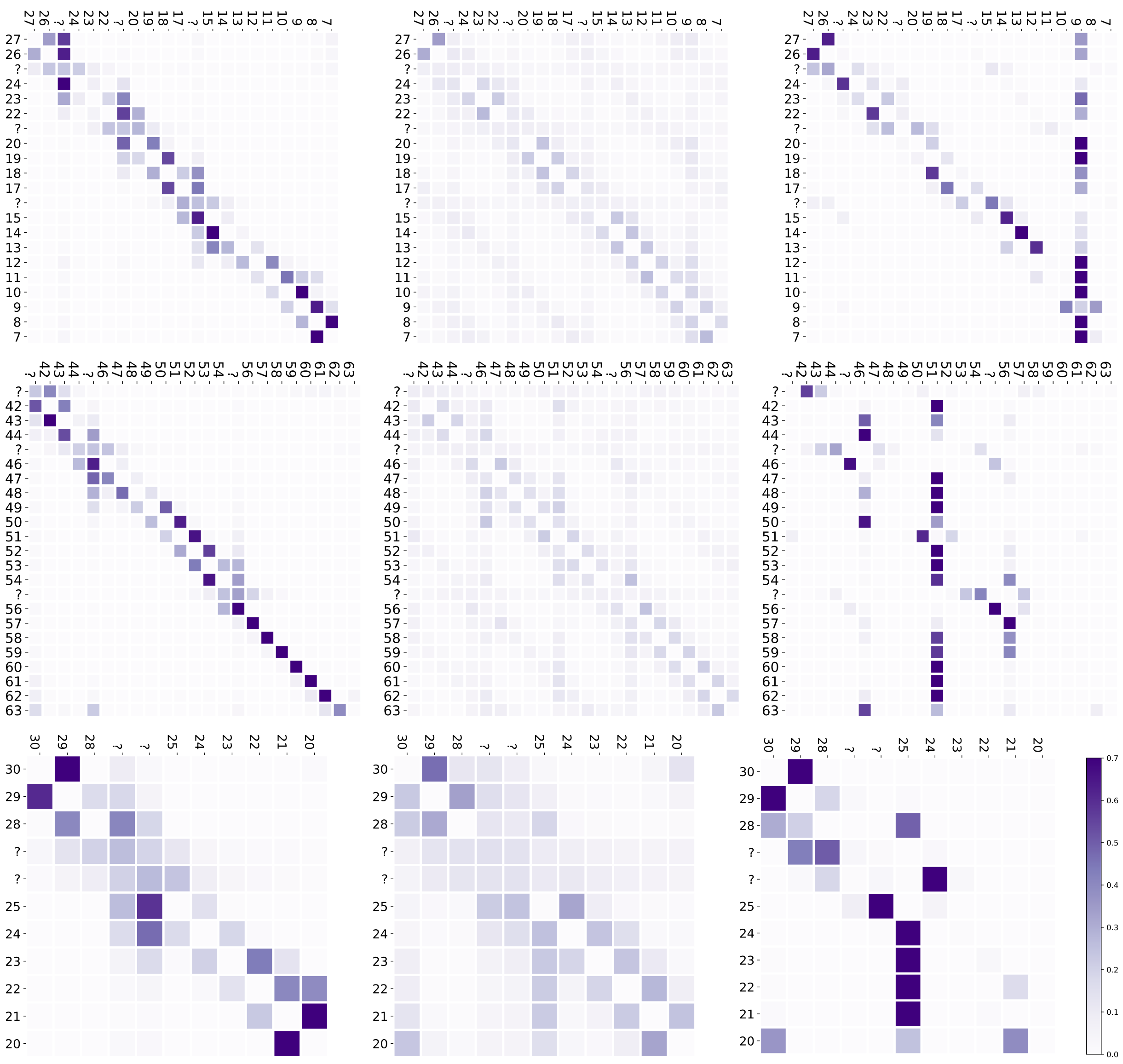
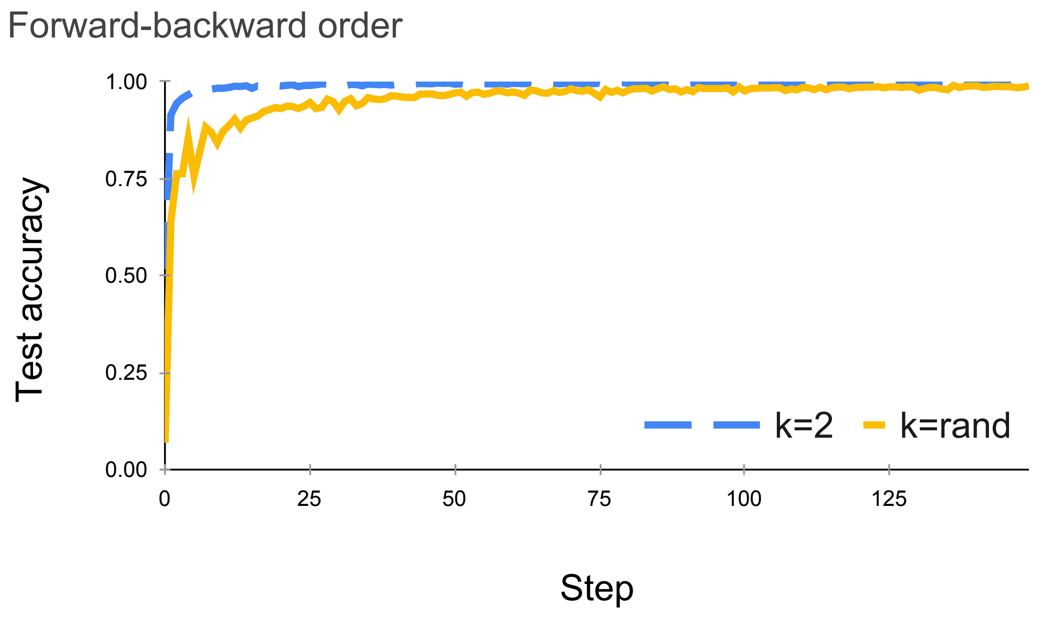


Figure 5.4: Example of the self-attention weights for models trained with $k = 1$ (left) and $k = 2$ after one iteration (middle) and two iterations (right). For $k = 2$, the model is more like an unrolled two-layer attention mechanism, with the first step identifying an off-diagonal pattern and the latter pooling information into an arbitrary token.

Number of Energy Update Iterations In addition to comparing the number of energy update iterations k, we also try setting a random number of updates during training. Instead of specifying a fixed number of iterations k, we take a random k between 1 and 5 and train the model with it. We evaluate the performance on inference with $k = 3$ (the average value). In Fig. 5.5 we compare the performance of the best model trained with random number of iterations k with the best performing models trained with fixed k. As the plots show, the model trained with a



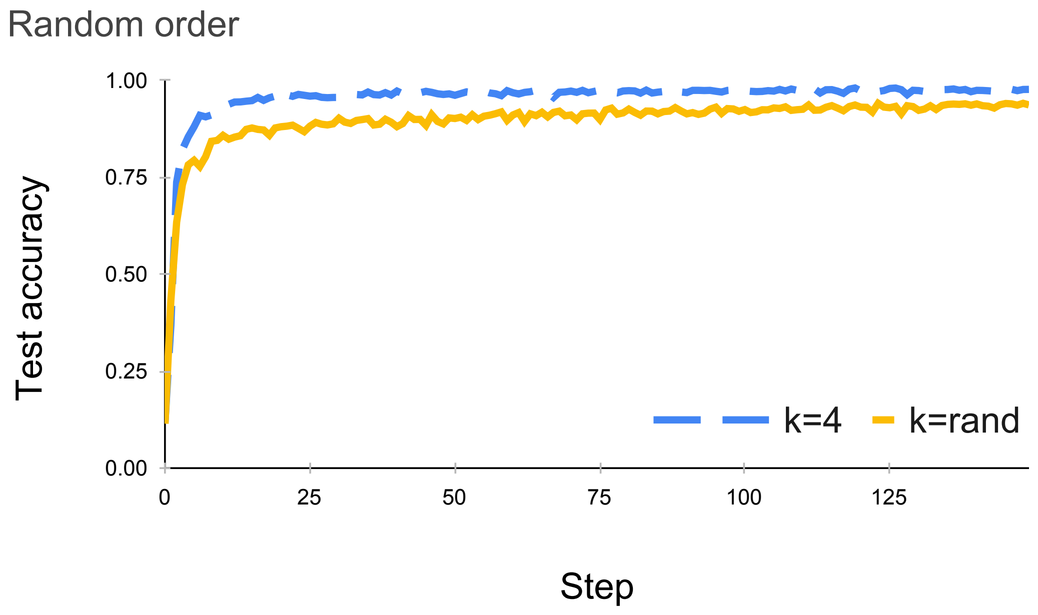


Figure 5.5: Learning curves for random number of variable update passes - for "forward-backward" (top) and random (bottom) order of operation updates.

random number of iterations performs on par with the best models with fixed k, but takes more time to train.

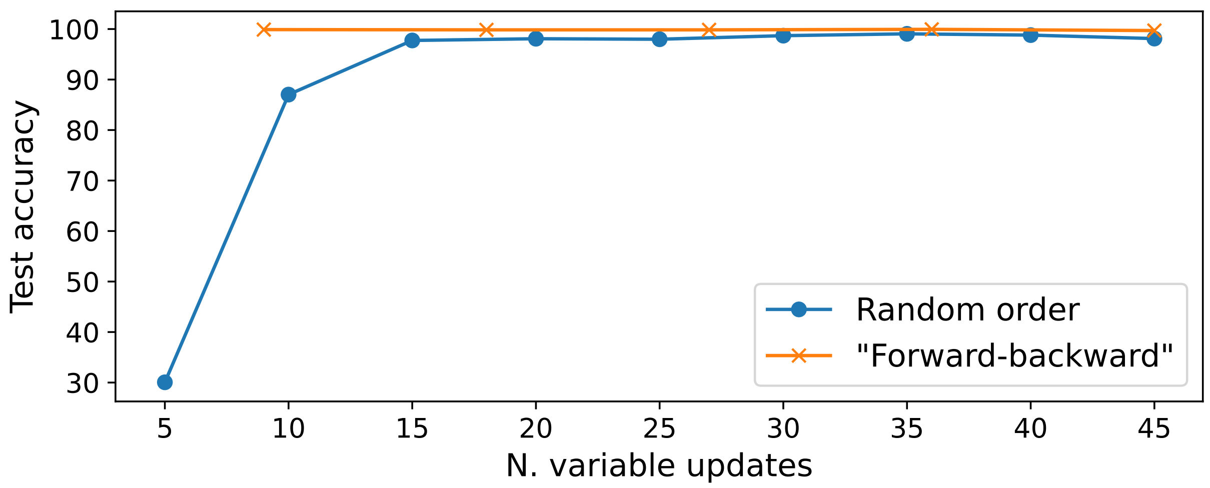


Figure 5.6: Comparison of the test accuracy) for models with random and "forward-backward" order of variable updates. Markers indicate one full iteration.

5.5 Structured UNNs for Dependency Parsing

The concept of UNN can be applied to structured tasks – all we need to do is to define *structured factors*, as shown next.

We experiment with a challenging structured prediction task from natural language processing: unlabeled, non-projective dependency parsing (Kübler et al., 2009). Given a sentence with n words, represented as a matrix $X \in \mathbb{R}^{r \times n}$ (where r is the embedding size), the goal is to predict the syntactic relations as a *dependency tree*, *i.e.*, a spanning tree which has the words as nodes. The output can be represented as a binary matrix $Y \in \mathbb{R}^{n \times n}$, where the $(i, j)^{\text{th}}$ entry indicates if there is a directed arc $i \to j$ connecting the i^{th} word (the *head*) and j^{th} word (the *modifier*). Fig. 5.8 shows examples of dependency trees produced by this model. We use a probabilistic model where the output Y can more broadly represent a probability distribution over trees, represented by the matrix of arc marginals induced by this distribution (illustrated in Fig. 5.7).

Biaffine parsing. A successful model for dependency parsing is the biaffine one (Dozat and Manning, 2016; Kiperwasser and Goldberg, 2016). This model first

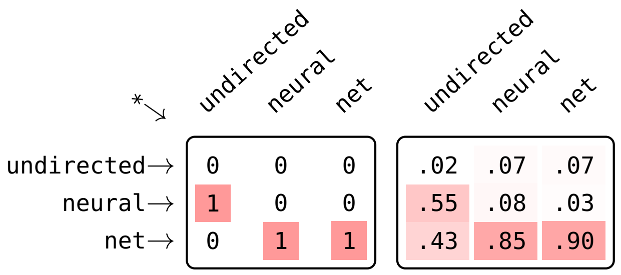

$$
\begin{array}{r|ccc|ccc|}
*\searrow & \texttt{undirected} & \texttt{neural} & \texttt{net} & \texttt{undirected} & \texttt{neural} & \texttt{net} \\
\texttt{undirected}\rightarrow & 0 & 0 & 0 & .02 & .07 & .07 \\
\texttt{neural}\rightarrow & 1 & 0 & 0 & .55 & .08 & .03 \\
\texttt{net}\rightarrow & 0 & 1 & 1 & .43 & .85 & .90
\end{array}
$$

Figure 5.7: "Packed" matrix representation of a dependency tree (left) and dependency arc marginals (right). Each element corresponds to an arc h→m, and the diagonal corresponds to the arcs from the root, $*\rightarrow$m. The marginals, computed via the matrix-tree theorem, are the structured counterpart of softmax, and correpond to arc probabilities.

computes head representations $H \in \mathbb{R}^{d\times n}$ and modifier representations $M \in \mathbb{R}^{d\times n}$, via a neural network that takes X as input – here, d denotes the hidden dimension of these representations. Then, it computes a score matrix as $Z = H^\top V M \in \mathbb{R}^{n\times n}$, where $V \in \mathbb{R}^{d\times d}$ is a parameter matrix. Entries of Z can be interpreted as scores for each candidate arc. From Z, the most likely tree can be obtained via the Chu-Liu-Edmonds maximum spanning arborescence algorithm (Chu and Liu, 1965; Edmonds, 1967), and probabilities and marginals can be computed via the matrix-tree theorem (Koo et al., 2007; Smith and Smith, 2007; McDonald and Satta, 2007; Kirchhoff, 1847).

UNN for parsing. We now construct an UNN with the same building blocks as this biaffine model, leading to the factor graph in Fig. 5.2(d). The variable nodes are $\{\mathsf{X},\mathsf{H},\mathsf{M},\mathsf{Y}\}$, and the factors are $\{\mathsf{XH},\mathsf{XM},\mathsf{HMY}\}$. Given parameter weight matrices $V, W_\mathsf{H}, W_\mathsf{M} \in \mathbb{R}^{d\times d}$ and biases $b_\mathsf{H}, b_\mathsf{M} \in \mathbb{R}^d$, we use bilinear and trilinear factor energies as follows:

$$
\begin{aligned}
E_{\mathsf{XH}}(X,H) &= -\langle H, W_\mathsf{H} X\rangle\,, \\
E_{\mathsf{XM}}(X,M) &= -\langle M, W_\mathsf{M} X\rangle\,, \\
E_{\mathsf{YHM}}(Y,H,M) &= -\langle Y, H^\top V M\rangle\,.
\end{aligned}
\tag{5.15}
$$

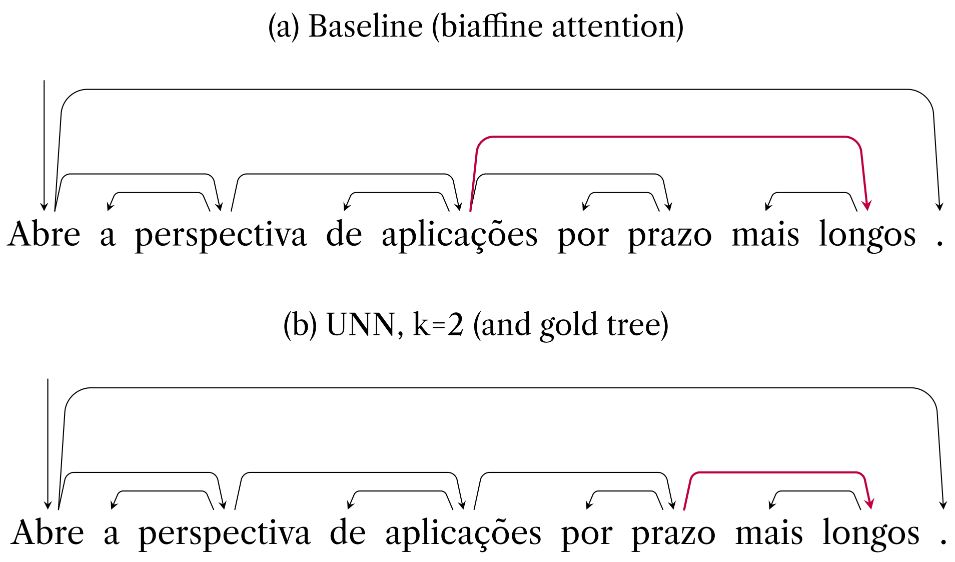

(a) Baseline (biaffine attention)
Abre a perspectiva de aplicações por prazo mais longos .
(b) UNN, k=2 (and gold tree)
Abre a perspectiva de aplicações por prazo mais longos .

Figure 5.8: Examples of dependency trees produced by the parsing model for a sentence in Portuguese. The baseline model (a) erroneously assigns the noun *aplicações* as the syntactic head of the adjective *longos*. The UNN with $k = 2$ iterations (b) matches the gold parse tree for this sentence, eventually benefiting from the structural information propagated back from the node Y after the first iteration.

For H and M, we use the ReLU regularizer,

$$\begin{aligned} E_{\mathsf{H}}(H) &= -\langle b_{\mathsf{H}} \otimes 1_n, H\rangle + \frac{1}{2}\|H\|^2 + \iota_{\geq 0}(H) \\ E_{\mathsf{M}}(M) &= -\langle b_{\mathsf{M}} \otimes 1_n, M\rangle + \frac{1}{2}\|M\|^2 + \iota_{\geq 0}(M)\,. \end{aligned} \tag{5.16}$$

For Y, however, we employ a structured entropy regularizer:

$$E_{\mathsf{Y}}(Y) = -\mathcal{H}_{\mathcal{M}}(Y) + \iota_{\mathcal{M}}(Y)\,, \tag{5.17}$$

where $\mathcal{M} = \operatorname{conv}(\mathcal{Y})$ is the marginal polytope (Wainwright and Jordan, 2008b; Martins et al., 2009), the convex hull of the adjacency matrices of all valid non-projective dependency trees (Fig. 5.7), and $\mathcal{H}_{\mathcal{M}}(Y)$ is the maximal entropy over all distribution over trees with arc marginals Y:

$$\mathcal{H}_{\mathcal{M}}(Y) \coloneqq \max_{\alpha\in\triangle_{|\mathcal{Y}|}} \mathcal{H}(\alpha) \text{ s.t. } \mathbb{E}_{A\sim\alpha}[A] = Y\,. \tag{5.18}$$

Derivation of block coordinate descent updates. To minimize the total energy, we iterate between updating H, M and Y k times, similar to the unstructured case.

The updates for the heads and modifiers work out to:

$$\begin{aligned} H_\star &= \operatorname{relu}(W_\mathsf{H} X + b_\mathsf{H} \otimes 1_n + VMY^\top)\,, \\ M_\star &= \operatorname{relu}(W_\mathsf{M} X + b_\mathsf{M} \otimes 1_n + V^\top HY)\,. \end{aligned} \tag{5.19}$$

For Y, however, we must solve the problem

$$Y_\star = \operatorname*{argmin}_{Y \in \mathcal{M}} -\langle Y, H^\top VM \rangle - \mathcal{H}_\mathcal{M}(Y)\,. \tag{5.20}$$

This combinatorial optimization problem corresponds to ***marginal inference*** (Wainwright and Jordan, 2008b), a well-studied computational problem in structured prediction that appears in all probabilistic models. While generally intractable, for non-projective dependency parsing it may be computed in time $\mathcal{O}(n^3)$ via the aforementioned matrix-tree theorem, the same algorithm required to compute the structured likelihood loss.[3]

With zero initialization, the first iteration yields the same hidden representations and output as the biaffine model, assuming the updates are performed in the order described. The extra terms involving $VMY^\top$ and $V^\top HY$ enable the current prediction for Y to influence neighboring words, which leads to a more expressive model overall.

Experiments. We experiment with the UNN for parsing described above. We test the architecture on several datasets from Universal Dependencies 2.7 (Zeman et al., 2020), covering different language families and dataset size: Afrikaans (AfriBooms), Arabic (PADT), Czech (PDT), English (Partut), Hungarian (Szeged), Italian (ISDT), Persian (Seraji), Portuguese (Bosque), Swedish (Talbanken), and

[3]During training, the matrix-tree theorem can be invoked only once to compute both the update to Y as well as the gradient of the loss, since $\nabla \log p(\mathsf{Y} = Y_\text{true}) = Y_\text{true} - \hat{Y}$.

Telugu (MTG). Performance is measured by three metrics:

- Unlabeled attachment score (UAS): a fine-grained, arc-level accuracy metric.
- Modifier list accuracy: the percentage of head words for which *all* modifiers were correctly predicted. For example, in Fig. 5.8, the baseline correctly predicts all modifiers for the words*perspectiva*, *abre*, *longos*, but not for the words *aplicações*, *prazo*.
- Exact match: the percentage sentences for which the full parse tree is correctly predicted: the harshest of the metrics.

The latter, coarser measures can give more information whether the model is able to learn global relations, not just how to make local predictions correctly (*i.e.*, when only prediction of the arcs is evaluated).

Our architecture is as follows: First, we pass the sentence through a BERT model (bert-base-multilingual-cased, fine-tuned during training, as directed networks can be added as components to UNNs, as mentioned in Section 5.2) and get the word representations of the last layer. These representations are the input x in the UNN model. Then, we apply the parsing model described in this section. The baseline ($k = 1$) corresponds to a biaffine parser using BERT features. The learning rate for each language is chosen via grid search for highest UAS on the validation set for the baseline model. We searched over the values $\{0.1, 0.5, 1, 5, 10\} \times 10^{-5}$. In the experiments, we use 10^{-5} for Italian and 5×10^{-5} for the other languages. We employ dropout regularization, using the same dropout mask for each variable throughout the inner coordinate descent iterations, so that dropped values do not leak.

The results from the parsing experiments are displayed in Table 5.6. The numbers in the table show results on the test set for the highest validation accuracy epoch. We see that some of the languages seem to benefit from the iterative procedure of UNNs (CS, HU, TE), while others do not (EN, AF), and little difference is observed in the remaining languages. In general, the baseline ($k = 1$) seems to attain higher accuracies in UAS (individual arcs), but most of the languages have overall more accurate structures (as measured by modifier list accuracy and by exact match) for $k > 1$. Fig. 5.8 illustrates with one example in Portuguese.

5.6 Related Work

Besides the models mentioned in Section 5.2 which may be regarded as particular cases of UNNs, other models and architectures, next described, bear relation to our work.

Probabilistic modeling of joint distributions Our work draws inspiration from the well-known Boltzmann machines and Hopfield networks (Ackley et al., 1985; Smolensky, 1986; Hopfield, 1984). We consider deterministic networks whose desired configurations minimize an energy function which decomposes as a factor graph. In contrast, many other works have studied probabilistic energy-based models (EBM) defined as Gibbs distributions, as well as efficient methods to learn those distributions and to sample from them (Ngiam et al., 2011; Du and Mordatch, 2019). Similar to how our convolutional UNN can be used for multiple purposes in Section 5.3, Grathwohl et al. (2020a) reinterprets standard discriminative classifiers $p(y|x)$ as an EBM of a joint distribution $p(x, y)$. Training stochastic EBMs requires Monte Carlo sampling or auxiliary networks (Grathwohl et al., 2020b); in contrast, our deterministic UNNs, more aligned conceptually with deterministic EBMs (LeCun et al., 2006), eschew probabilistic modeling in favor of more direct training. Moreover, our UNN architectures closely parallel feed-forward networks and reuse their building blocks, uniquely bridging the two paradigms.

Structured Prediction Energy Networks (SPENs) We saw in Section 5.5 that UNNs can handle structured outputs. An alternative framework for expressive structured prediction is given by SPENs (Belanger and McCallum, 2016). Most

SPEN inference strategies require gradient descent, often with higher-order gradients for learning (Belanger et al., 2017), or training separate inference networks (Tu et al., 2020). UNNs in contrast, are well suited for coordinate descent inference: a learning-rate free algorithm with updates based on existing neural network building blocks. An undirected variant of SPENs would be similar to the MLP factor graph in Fig. 5.2(b), but with X and Y connected to a joint, higher-order factor, rather than via a chain $\mathsf{X} - \mathsf{H} - \mathsf{Y}$.

Universal transformers and Hopfield networks In Section 5.4 we show how we can implement self-attention with UNNs. Performing multiple energy updates resembles – but is different from – transformers (Vaswani et al., 2017) with shared weights between the layers. Our perspective of minimizing UNNs with coordinate descent using a fixed schedule and this unrolling is similar (but not exactly the same due to the skip connections) to having deeper neural networks which shared parameters for each layer. Such an architecture is the Universal Transformer (Dehghani et al., 2018), which applies a recurrent neural network to the transformer encoder and decoder. Recent work (Ramsauer et al., 2020) shows that the self-attention layers of transformers can be regarded as the update rule of a Hopfield network with continuous states (Hopfield, 1984). This leads to a "modern Hopfield network" with continuous states and an update rule which ensures global convergence to stationary points of the energy (local minima or saddle points). Like that model, UNNs also seek local minima of an energy function, albeit with a different goal.

Deep models as graphical model inference. This line of work defines neural computation via approximate inference in graphical models. Domke (2012) derives backpropagating versions of gradient descent, heavy-ball and LBFGS. They require as input only routines to compute the gradient of the energy with respect to the domain and parameters. Domke (2012) studies learning with unrolled gradient-based inference in general energy models. UNNs, in contrast, allow efficient, learning rate free, block-coordinate optimization by design. An exciting line of work derives unrolled architectures from inference in specific generative models (Hershey et al., 2014; ?; Lawson et al., 2019)—a powerful construction at the cost of more challenging optimization. The former is closest to our strategy, but by starting from probabilistic models the resulting updates are farther from contemporary deep learning (*e.g.*, convolutions, attention). In contrast, UNNs can reuse successful implementations, modular pretrained models, as well as structured factors, as we demonstrate in our parsing experiments. We believe that our UNN construction can shed new light over probabilistic inference models as well, uncovering deeper connections between the paradigms.

5.7 Conclusions and Future Work

We presented UNNs – a structured energy-based model which combines the power of factor graphs and neural networks. At inference time, the model energy is minimized with a coordinate descent algorithm, allowing reuse of existing building blocks in a modular way with guarantees of decreasing the energy at each step. We showed how the proposed UNNs subsume many existing architectures, conveniently combining supervised and unsupervised/self-supervised learning, as demonstrated on the three tasks.

We hope our first steps in this chapter will spark multiple directions of future work on undirected networks.

Language	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Unlabeled attachment score					
AF	**89.09**	88.98	88.40	87.77	88.46
AR	**85.62**	84.94	84.22	83.69	83.63
CS	93.79	**93.83**	93.82	93.60	93.77
EN	91.96	91.86	91.09	**91.99**	91.51
FA	**83.41**	83.27	82.95	83.37	83.27
HU	85.11	**85.77**	84.47	85.13	84.09
IT	**94.76**	94.43	94.35	94.59	94.45
PT	96.99	97.00	96.83	**97.06**	96.90
SW	**91.42**	90.92	91.30	91.08	90.98
TE	89.72	89.72	**90.00**	88.45	87.75
Modifier list accuracy					
AF	**74.10**	72.60	72.90	71.78	72.01
AR	**70.44**	69.29	68.41	68.08	68.19
CS	84.46	84.82	**84.93**	84.12	84.49
EN	79.08	77.73	75.20	78.90	**79.44**
FA	64.80	**66.75**	65.28	66.67	65.85
HU	64.13	**66.07**	64.37	62.91	64.13
IT	**85.32**	83.59	83.71	83.94	84.05
PT	90.10	**90.69**	90.39	90.66	90.49
SW	**79.07**	78.37	78.52	78.60	78.24
TE	72.87	72.87	**73.68**	66.80	65.99
Exact match					
AF	**37.70**	33.88	34.43	33.88	32.79
AR	19.44	19.29	18.36	**19.91**	18.36
CS	59.17	60.76	**60.92**	59.42	59.84
EN	**48.59**	44.37	40.14	43.66	44.37
FA	21.52	22.15	22.78	**24.68**	23.42
HU	21.13	23.40	**24.15**	23.40	21.51
IT	**64.93**	63.54	62.85	63.89	64.24
PT	73.24	**74.86**	73.89	74.43	74.11
SW	**54.62**	52.38	54.13	53.94	52.67
TE	75.69	77.08	**79.17**	71.53	70.14

Table 5.6: Results from experiments with parsing with structured UNNs. The columns show the number of UNN iterations. The best result for each row is rendered in bold.

CHAPTER 6

Conclusion

Contents

In this chapter, we summarize our contributions and conclusions and suggest some directions for future work.

6.1 Summary of Contributions

In this thesis, we have contributed to three main areas related to modeling and predicting structure in deep neural models for natural language processing.

Exposure Bias, Scheduled Sampling and Transformers We address the problem of exposure bias in the transformer model. Scheduled sampling has been proposed for recurrent neural network architectures to address exposure bias, but in transformers applying it is not straight-forward. We proposed a two-decoder transformer architecture which addresses the problem with exposure bias for the transformer model. In the first decoder pass, the output sequence is predicted as usual. Then, a mix of the model output and the gold target sequence is passed and the final model output is predicted based on this mix with the second decoder call. This method has been further extended and improved by other researchers.

Discrete Latent Structures In NLP models, modeling structure as a latent variable, can combine the transparency of the pipeline approach with the end-to-end unsupervised representation learning and make deep models appealing. Learning with discrete, combinatorial latent variables is, however, challenging, due to the intersection of large cardinality and null gradient issues. In this thesis we studied surrogate gradient methods for deterministic learning with discrete structured latent variable. We propose a novel motivation for surrogate gradient methods, based on optimizing a pulled-back loss, thereby inducing pseudo-supervision on the latent variable. This leads to new insight into STE and SPIGOT. We show how our framework may be used to derive new surrogate gradient methods, by varying the loss function or the inner optimization algorithm used for inducing the pseudo-supervision. We experimentally validate our discoveries on a controllable experiment as well as on English language sentiment analysis and natural language inference, comparing against stochastic and relaxed alternatives, yielding new insights, and identifying noteworthy failure cases.

Undirected Neural Networks and Modularity We combine the representational strengths of factor graphs and neural networks and we propose *undirected neural networks (UNNs)* – a framework in which outputs are not computed by evaluating a composition of functions in a given order, but are rather obtained implicitly by minimizing an energy function which factors over a graph. We show how, for particular choices of the factor potentials, the UNNs subsume many existing neural architectures. We provide a coordinate descent inference algorithm, which can reuse current building blocks from feed-forward networks in a modular way. We develop and experiment with multiple factor graph architectures, tackling both structured and unstructured tasks, such as natural language parsing, image classification, and image prototype generation. We develop a new undirected attention mechanism and demonstrate its suitability for sequence completion.

6.2 Future Work

Modular training of neural networks As we described in Chapter 1, natural language is rich in structure and many NLP tasks can be composed into smaller and well-defined subtasks. It makes sense to train those subtasks as separate modules, which can be plugged-in to models solving different more complex tasks. For example, in the case with sentiment classification with latent dependency parsing in Chapter 4, the parsing can be trained as a separate smaller neural network module which could be plugged-in to a model for sentiment classification, natural language inference (as in Chapter 4), machine translation, or any other task, for which parsing the sentence makes sense for the downstream task. Methods for training of models with discrete latent structures, as in Chapter 4 can be used to allow training the subtask modules as part of the bigger model. The undirected neural networks approach in Chapter 5 could allow flexible composition of modules and enable various configurations of combining modules, not just for NLP, but also in other domains.

Semi-Supervised Learning with Discrete Latent Structures In Chapter 4, we used no supervision about the latent structure. It is intuitive to think of training these models when we have full or partial supervision of the latent variable. It would be interesting to explore how the models behave when partial supervision is available. For example, when training a model with latent syntax, information about the gold syntax tree could be present or predicted one from external parser and this information can be used on training time. If partial supervision on the latent variable leads to improvement of the performance on a downstream task, this would open possibilities to include linguistic and other kind of knowledge to models for which we have partial supervision for some components indirectly related to the downstream task. Including such external knowledge would hopefully guide the model to learn faster and with less computation resources. This is an intuitive extension of our work on latent structures described in Chapter 4.

Modules with Discrete Output In our work with discrete latent structures, we define the update of the latent variable as a result of optimizing the pulled-back loss on the downstream task. This led to the idea that we can use this formulation to pretrain modules which output this latent variable and combine several such modules to train a bigger model. In this concept of modularity, having the option of a module/component with discrete output is important, because it allows flexibility of the choice of modules. Training of modules with latent output is closely related to our work on latent structures in Chapter 4. The insights about the formulation of updating the latent variable as a pullback loss can be used to find clever ways to backpropagate through the modules with discrete output and include them more easily in the overall model. The challenge is that it needs to be defined how such modules could be pretrained without supervision or with only partial supervision. The main goal of such work would be to enable training of separate modules which output a discrete result, in order to use these modules as building blocks of a multi-modular architecture. Modules trained with this approach could be plugged-in in the UNN model described in Chapter 5. In the Greedy Infomax paper (Löwe et al., 2019), each module outputs representations of the input. A possible direction for self-supervised training of modules with discrete output is to use their framework as a base and explore the possibilities for extending it to cases where the output is discrete. This kind of modular training and unsupervised training of modules with discrete output would allow use of semi-supervision for modules for which we have information as part of training the whole system. The goal would be to allow easy module organization and replacement, as well as exploration which modules are important for training the system for a downstream task.

Extensions and Applications of Undirected Neural Networks In Chapter 5, we defined undirected neural networks, showed how they subsume existing architectures, trained them with gradient descent by unrolling the factor graph and demonstrated several possible architectures. We hope our first steps will spark multiple directions of future work on undirected networks. One promising direction is on probabilistic UNNs with Gibbs sampling or mean field theory (Saul et al., 1996; Henderson and Titov, 2010), which have the potential to bring our modular architectures to generative models. Another direction is to consider alternate training strategies for UNNs. Our strategy of converting UNNs to unrolled neural networks, enabled by Lemma 1 in Chapter 5, makes gradient-based training easy to implement, but alternate training strategies, perhaps based on equilibrium conditions or dual decomposition, hold promise. The flexibility of the model allows it to be applied to various problems. There could be opportunities to apply this model to real-world problems. In particular, the model can be used for modular training of problems which can be broken down into smaller components which should be optimized together for solving the larger task. This kind of modular training can be applied not only to NLP, but also to other domains, such as vision and speech. Undirected neural networks could also be applied for multitask learning and training with missing data. For example, if we train the model to predict several related tasks and have missing data for some of the tasks for part of the data, performing multiple iterations could allow the model to be trained with partially missing data.

Bibliography

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

G. Bakır, T. Hofmann, A. J. Smola, B. Schölkopf, and B. Taskar. *Predicting structured data*. MIT press, 2007.

A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3): 167–175, 2003. URL `https://www.sciencedirect.com/science/article/abs/pii/S0167637702002316`.

D. Belanger and A. McCallum. Structured prediction energy networks. In *International Conference on Machine Learning*, pages 983–992. PMLR, 2016.

D. Belanger, B. Yang, and A. McCallum. End-to-end learning for structured prediction energy networks. In *International Conference on Machine Learning*, pages 429–439. PMLR, 2017.

S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *preprint arXiv:1308.3432*, 2013.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific Belmont, 1999.

L. Bottou and P. Gallinari. A framework for the cooperation of learning algorithms. In *Advances in neural information processing systems*, pages 781–788, 1991.

S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In *Proc. EMNLP*. Association for Computational Linguistics, 2015. URL `https://www.aclweb.org/anthology/D15-1075/`.

D. M. Bradley. Learning in modular systems. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 2010.

R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

M. Cettolo, M. Federico, L. Bentivogli, N. Jan, S. Sebastian, S. Katsuitho, Y. Koichiro, and F. Christian. Overview of the iwslt 2017 evaluation campaign. In *International Workshop on Spoken Language Translation*, pages 2–14, 2017.

P. Cheng, C. Liu, C. Li, D. Shen, R. Henao, and L. Carin. Straight-through estimator as projected Wasserstein gradient flow. In *Third workshop on Bayesian Deep Learning (NeurIPS 2018)*, 2018. URL `http://bayesiandeeplearning.org/2018/papers/53.pdf`.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

J. Choi, K. M. Yoo, and S.-g. Lee. Learning to compose task-specific tree structures. In *Proc. AAAI*, 2018.

Y.-J. Chu and T.-H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

C. Corro and I. Titov. Differentiable Perturb-and-Parse: Semi-Supervised Parsing with a Structured Variational Autoencoder. In *Proc. ICLR*, 2019a.

C. Corro and I. Titov. Learning latent trees with stochastic perturbations and differentiable dynamic programming. In *Proc. ACL*, 2019b. URL `https://aclanthology.org/P19-1551/`.

H. Daumé, J. Langford, and D. Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL-HLT*, 2019. URL `https://www.aclweb.org/anthology/N19-1423`.

J. Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326. PMLR, 2012.

T. Dozat and C. D. Manning. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

Y.-l. Du and I. Mordatch. Implicit generation and modeling with energy based models. In *NeurIPS*, 2019.

D. Duckworth, A. Neelakantan, B. Goodrich, L. Kaiser, and S. Bengio. Parallel scheduled sampling, 2019.

G. Durrett and D. Klein. Neural CRF parsing. 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)*, pages 302–312, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1030. URL `https://aclanthology.org/P15-1030`.

D. Duvenaud, J. Z. Kolter, and M. Johnson. Deep implicit layers tutorial-neural odes, deep equilibirum models, and beyond. *Neural Information Processing Systems Tutorial*, 2020.

J. Edmonds. Optimum branchings. *J. Res. Nat. Bur. Stand.*, 71B:233–240, 1967.

J. Eisenstein. Natural language processing, 2018.

J. R. Finkel, C. D. Manning, and A. Y. Ng. Solving the problem of cascading errors: Approximate bayesian inference for linguistic annotation pipelines. In *Proc. EMNLP*, 2006. URL `https://dl.acm.org/doi/pdf/10.5555/1610075.1610162`.

E. Fonseca and A. F. T. Martins. Revisiting higher-order dependency parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8795–8800, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.776. URL `https://aclanthology.org/2020.acl-main.776`.

P. W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Commun. ACM*, 33(10):75–84, Oct. 1990. ISSN 0001-0782. doi: 10.1145/84537.84552. URL `http://doi.acm.org/10.1145/84537.84552`.

A. A. Goldstein. Convex programming in hilbert space. *Bulletin of the American Mathematical Society*, 70(5):709–710, 1964.

K. Goyal, C. Dyer, and T. Berg-Kirkpatrick. Differentiable scheduled sampling for credit assignment. *arXiv preprint arXiv:1704.06970*, 2017.

W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, M. Norouzi, and K. Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *International Conference on Learning Representations*, 2020a.

W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, and R. Zemel. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *ICML*. PMLR, 2020b.

N. Gupta, K. Lin, D. Roth, S. Singh, and M. Gardner. Neural module networks for reasoning over text. *arXiv preprint arXiv:1912.04971*, 2019.

K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

S. Havrylov, G. Kruszewski, and A. Joulin. Cooperative learning of disjoint syntax and semantics. In *Proc. NAACL: Volume 1 (Long and Short Papers)*, pages 1118–1128, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1115. URL `https://www.aclweb.org/anthology/N19-1115`.

J. Henderson and I. Titov. Incremental sigmoid belief networks for grammar learning. *Journal of Machine Learning Research*, 11(12), 2010.

J. R. Hershey, J. L. Roux, and F. Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *CoRR*, abs/1409.2574, 2014. URL `http://arxiv.org/abs/1409.2574`.

G. Hinton. Neural networks for machine learning. In *Coursera video lectures*, 2012.

G. E. Hinton. Boltzmann machine. *Scholarpedia*, 2(5):1668, 2007.

G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984.

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Proc. NeurIPS*, 2016. URL `https://arxiv.org/abs/1602.02830`.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

E. Jang, S. Gu, and B. Poole. Categorical reparametrization with Gumbel-Softmax. In *Proc. ICLR*, 2017.

D. Jurafsky and J. H. Martin. Speech and language processing. vol. 3, 2014.

Y. Kim, C. Denton, L. Hoang, and A. M. Rush. Structured attention networks. In *Proc. ICLR*, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

E. Kiperwasser and Y. Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016. doi: 10.1162/tacl_a_00101. URL `https://www.aclweb.org/anthology/Q16-1023`.

G. Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. URL `http://arxiv.org/abs/1612.00796`.

L. Kirsch, J. Kunze, and D. Barber. Modular networks: Learning to decompose neural computation. In *Advances in Neural Information Processing Systems*, pages 2408–2418, 2018.

J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

J. P. Kleijnen and R. Y. Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, 1996. URL `https://research.tilburguniversity.edu/en/publications/optimization-and-sensitivity-analysis-of-computer-simulation-mode-2`.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv e-prints*, 2017.

T. Koo, A. Globerson, X. Carreras Pérez, and M. Collins. Structured prediction models via the matrix-tree theorem. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 141–150, 2007.

M. Korakakis and A. Vlachos. Mitigating catastrophic forgetting in scheduled sampling with elastic weight consolidation in neural machine translation. *CoRR*, abs/2109.06308, 2021. URL `https://arxiv.org/abs/2109.06308`.

S. Kübler, R. McDonald, and J. Nivre. Dependency parsing. *Synthesis lectures on human language technologies*, 1(1):1–127, 2009.

J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, 2001.

J. Lawson, G. Tucker, B. Dai, and R. Ranganath. Energy-inspired models: Learning with sampler-induced distributions. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/28659414dab9eca0219dd592b8136434-Paper.pdf`.

Y. LeCun, C. Cortes, and C. J. Burges. MNIST handwritten digit database. 1998. URL `http://yann.lecun.com/exdb/mnist/`.

Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

E. S. Levitin and B. T. Polyak. Constrained minimization methods. *USSR Computational mathematics and mathematical physics*, 6(5):1–50, 1966.

Y. Li and R. S. Zemel. Mean-field networks. *CoRR*, abs/1410.5884, 2014. URL `http://arxiv.org/abs/1410.5884`.

R. Linsker. Self-organization in a perceptual network. *Computer*, 21(3):105–117, 1988.

Y. Liu and M. Lapata. Learning structured text representations. *TACL*, 6:63–75, 2018.

Y. Liu, F. Meng, Y. Chen, J. Xu, and J. Zhou. Confidence-aware scheduled sampling for neural machine translation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2327–2337, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.205. URL `https://aclanthology.org/2021.findings-acl.205`.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *preprint arXiv:1711.05101*, 2018. URL `https://arxiv.org/abs/1711.05101`.

S. Löwe, P. O'Connor, and B. S. Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. *arXiv preprint arXiv:1905.11786*, 2019.

T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

J. Maillard and S. Clark. Latent Tree Learning with Differentiable Parsers: Shift-Reduce Parsing and Chart Parsing. In *Proc. ACL*, 2018.

J. Maillard, S. Clark, and D. Yogatama. Jointly learning sentence embeddings and syntax with unsupervised tree-LSTMs. *preprint arXiv:1705.09189*, 2017.

A. Martins and R. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, pages 1614–1623, 2016a.

A. F. Martins and R. F. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proc. ICML*, 2016b.

A. F. Martins, N. A. Smith, and E. P. Xing. Polyhedral outer approximations with application to natural language parsing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 713–720, 2009.

A. F. T. Martins, T. Mihaylova, N. Nangia, and V. Niculae. Latent structure models for natural language processing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 1–5, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-4001. URL `https://www.aclweb.org/anthology/P19-4001`.

R. McDonald and G. Satta. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 121–132, 2007.

X. Meng, R. Bachmann, and M. E. Khan. Training binary neural networks using the bayesian learning rule. *Proc. ICML*, 2020. URL `https://arxiv.org/abs/2002.10778`.

A. Mensch and M. Blondel. Differentiable dynamic programming for structured prediction and attention. In *Proc. ICML*, 2018.

T. Mihaylova and A. F. T. Martins. Scheduled sampling for transformers. In *Proceedings ACL SRW*, 2019.

T. Mihaylova, V. Niculae, and A. F. T. Martins. Understanding the spigot mechanics: Surrogate gradients for latent structure learning. In *under review for ACL*, 2020.

G. Neubig. The kyoto free translation task, 2011.

J. Ngiam, Z. Chen, P. W. Koh, and A. Y. Ng. Learning deep energy models. In *ICML*, 2011.

V. Niculae, A. F. Martins, M. Blondel, and C. Cardie. SparseMAP: Differentiable sparse structured inference. In *Proc. ICML*, 2018a.

V. Niculae, A. F. Martins, and C. Cardie. Towards dynamic computation graphs via sparse latent structure. In *Proc. EMNLP*, 2018b.

S. Nowozin, P. V. Gehler, J. Jancsary, and C. H. Lampert. *Advanced Structured Prediction*. MIT Press, 2014.

A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

G. Papandreou and A. L. Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *Proc. ICCV*. IEEE, 2011.

A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit. A decomposable attention model for natural language inference. *Proc. EMNLP*, 2016. URL `https://arxiv.org/abs/1606.01933`.

H. Peng, S. Thomson, and N. A. Smith. Backpropagating through structured argmax using a SPIGOT. In *Proc. ACL*, 2018.

J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In *Proc. EMNLP*, 2014. URL `https://www.aclweb.org/anthology/D14-1162/`.

H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, L. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, V. Greiff, et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proc. EMNLP-CoNLL*, 2007. URL `https://www.aclweb.org/anthology/D07-1043/`.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

A. M. Rush. Torch-struct: Deep structured prediction library, 2020.

V. Sanh, T. Wolf, and S. Ruder. A hierarchical multi-task approach for learning embeddings from semantic tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6949–6956, 2019.

L. K. Saul, T. Jaakkola, and M. I. Jordan. Mean field theory for sigmoid belief networks. *Journal of artificial intelligence research*, 4:61–76, 1996.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2016.

S. Shen, Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu. Minimum risk training for neural machine translation. In *Proc. ACL: Volume 1 (Long Papers)*, pages 1683–1692, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1159. URL `https://www.aclweb.org/anthology/P16-1159`.

D. A. Smith and J. Eisner. Minimum risk annealing for training log-linear models. In *Proc. COLING/ACL*, 2006.

D. A. Smith and N. A. Smith. Probabilistic models of nonprojective dependency trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 132–140, 2007.

N. A. Smith. Linguistic structure prediction. *Synthesis lectures on human language technologies*, 4(2):1–274, 2011.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. EMNLP*, 2013. URL `https://www.aclweb.org/anthology/D13-1170/`.

V. Stoyanov, A. Ropson, and J. Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proc. AISTATS*, 2011.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

C. Sutton and A. McCallum. Joint parsing and semantic role labeling. In *Proc. CoNLL*, 2005. URL `https://www.aclweb.org/anthology/W05-0636/`.

I. Tenney, D. Das, and E. Pavlick. BERT rediscovers the classical NLP pipeline. In *Proc. ACL*, pages 4593–4601, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://www.aclweb.org/anthology/P19-1452.

A. Tjandra, S. Sakti, and S. Nakamura. End-to-end feedback loss in speech chain framework via straight-through estimator. In *Proc. ICASSP*. IEEE, 2019. URL https://arxiv.org/abs/1810.13107.

K. N. Toutanova. *Effective statistical models for syntactic and semantic disambiguation*. Stanford University, 2005.

L. Tu, R. Y. Pang, and K. Gimpel. Improving joint training of inference networks and structured prediction energy networks. In *Proceedings of the Fourth Workshop on Structured Prediction for NLP*, pages 62–73, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.spnlp-1.8. URL https://aclanthology.org/2020.spnlp-1.8.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. URL https://papers.neurips.cc/paper/7181-attention-is-all-you-need.pdf.

M. Vlastelica, A. Paulus, V. Musil, G. Martius, and M. Rolinek. Differentiation of blackbox combinatorial solvers. In *Proc. ICLR*, 2020. URL https://openreview.net/forum?id=BkevoJSYPB.

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1–2):1–305, 2008a.

M. J. Wainwright and M. I. Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008b.

M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, 2004.

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

A. Williams, A. Drozdov, and S. R. Bowman. Do latent tree learning models identify meaningful structure in sentences? *TACL*, 6:253–267, 2018. URL https://arxiv.org/abs/1709.01121.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

S. Wiseman and A. M. Rush. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, 2016.

L. Wu, F. Tian, T. Qin, J. Lai, and T.-Y. Liu. A study of reinforcement learning for neural machine translation. In *Proc. EMNLP*, pages 3612–3621, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1397. URL https://www.aclweb.org/anthology/D18-1397.

Y. Xu and W. Yin. A block coordinate descent method for regularized multi-convex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6(3):1758–1789, 2013.

P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin. Understanding straight-through estimator in training activation quantized neural nets. In *Proc. ICLR*, 2019. URL https://arxiv.org/abs/1903.05662.

D. Yogatama, P. Blunsom, C. Dyer, E. Grefenstette, and W. Ling. Learning to compose words into sentences with reinforcement learning. In *Proc. ICLR*, 2017.

D. Zeman, J. Nivre, M. Abrams, E. Ackermann, N. Aepli, H. Aghaei, Ž. Agić, A. Ahmadi, L. Ahrenberg, C. K. Ajede, G. Aleksandravičiūtė, I. Alfina, L. Antonsen, K. Aplonova, A. Aquino, C. Aragon, M. J. Aranzabe, Ĥ. Arnardóttir, G. Arutie, J. N. Arwidarasti, M. Asahara, L. Ateyah, F. Atmaca, M. Attia, A. Atutxa, L. Augustinus, E. Badmaeva, K. Balasubramani, M. Ballesteros, E. Banerjee, S. Bank, V. Barbu Mititelu, V. Basmov, C. Batchelor, J. Bauer, S. T. Bedir, K. Bengoetxea, G. Berk, Y. Berzak, I. A. Bhat, R. A. Bhat, E. Biagetti, E. Bick, A. Bielinskienė, K. Bjarnadóttir, R. Blokland, V. Bobicev, L. Boizou, E. Borges Völker, C. Börstell, C. Bosco, G. Bouma, S. Bowman, A. Boyd, K. Brokaitė, A. Burchardt, M. Candito, B. Caron, G. Caron, T. Cavalcanti, G. Cebiroğlu Eryiğit, F. M. Cecchini, G. G. A. Celano, S. Čéplö, S. Cetin, Ö. Çetinoğlu, F. Chalub, E. Chi, Y. Cho, J. Choi, J. Chun, A. T. Cignarella, S. Cinková, A. Collomb, Ç. Çöltekin, M. Connor, M. Courtin, E. Davidson, M.-C. de Marneffe, V. de Paiva, M. O. Derin, E. de Souza, A. Diaz de Ilarraza, C. Dickerson, A. Dinakaramani, B. Dione, P. Dirix, K. Dobrovoljc, T. Dozat, K. Droganova, P. Dwivedi, H. Eckhoff, M. Eli, A. Elkahky, B. Ephrem, O. Erina, T. Erjavec, A. Etienne, W. Evelyn, S. Facundes, R. Farkas, M. Fernanda,

H. Fernandez Alcalde, J. Foster, C. Freitas, K. Fujita, K. Gajdošová, D. Galbraith, M. Garcia, M. Gärdenfors, S. Garza, F. F. Gerardi, K. Gerdes, F. Ginter, I. Goenaga, K. Gojenola, M. Gökırmak, Y. Goldberg, X. Gómez Guinovart, B. González Saavedra, B. Griciūtė, M. Grioni, L. Grobol, N. Grūzītis, B. Guillaume, C. Guillot-Barbance, T. Güngör, N. Habash, H. Hafsteinsson, J. Hajič, J. Hajič jr., M. Hämäläinen, L. Hà Mỹ, N.-R. Han, M. Y. Hanifmuti, S. Hardwick, K. Harris, D. Haug, J. Heinecke, O. Hellwig, F. Hennig, B. Hladká, J. Hlaváčová, F. Hociung, P. Hohle, E. Huber, J. Hwang, T. Ikeda, A. K. Ingason, R. Ion, E. Irimia, Ọ. Ishola, T. Jelínek, A. Johannsen, H. Jónsdóttir, F. Jørgensen, M. Juutinen, S. K, H. Kaşıkara, A. Kaasen, N. Kabaeva, S. Kahane, H. Kanayama, J. Kanerva, B. Katz, T. Kayadelen, J. Kenney, V. Kettnerová, J. Kirchner, E. Klementieva, A. Köhn, A. Köksal, K. Kopacewicz, T. Korkiakangas, N. Kotsyba, J. Kovalevskaitė, S. Krek, P. Krishnamurthy, S. Kwak, V. Laippala, L. Lam, L. Lambertino, T. Lando, S. D. Larasati, A. Lavrentiev, J. Lee, P. Lê Hồng, A. Lenci, S. Lertpradit, H. Leung, M. Levina, C. Y. Li, J. Li, K. Li, Y. Li, K. Lim, K. Lindén, N. Ljubešić, O. Loginova, A. Luthfi, M. Luukko, O. Lyashevskaya, T. Lynn, V. Macketanz, A. Makazhanov, M. Mandl, C. Manning, R. Manurung, C. Mărănduc, D. Mareček, K. Marheinecke, H. Martínez Alonso, A. Martins, J. Mašek, H. Matsuda, Y. Matsumoto, R. McDonald, S. McGuinness, G. Mendonça, N. Miekka, K. Mischenkova, M. Misirpashayeva, A. Missilä, C. Mititelu, M. Mitrofan, Y. Miyao, A. Mojiri Foroushani, A. Moloodi, S. Montemagni, A. More, L. Moreno Romero, K. S. Mori, S. Mori, T. Morioka, S. Moro, B. Mortensen, B. Moskalevskyi, K. Muischnek, R. Munro, Y. Murawaki, K. Müürisep, P. Nainwani, M. Nakhlé, J. I. Navarro Horñiacek, A. Nedoluzhko, G. Nešpore-Bērzkalne, L. Nguyễn Thị, H. Nguyễn Thị Minh, Y. Nikaido, V. Nikolaev, R. Nitisaroj, A. Nourian, H. Nurmi, S. Ojala, A. K. Ojha, A. Olúòkun, M. Omura, E. Onwuegbuzia, P. Osenova, R. Östling, L. Øvrelid, Ş. B. Özateş, A. Özgür, B. Öztürk Başaran, N. Partanen, E. Pascual, M. Passarotti, A. Patejuk, G. Paulino-Passos, A. Peljak-Łapińska, S. Peng, C.-A. Perez, N. Perkova, G. Perrier, S. Petrov, D. Petrova, J. Phelan, J. Piitulainen, T. A. Pirinen, E. Pitler, B. Plank, T. Poibeau, L. Ponomareva, M. Popel, L. Pretkalniņa, S. Prévost, P. Prokopidis, A. Przepiórkowski, T. Puolakainen, S. Pyysalo, P. Qi, A. Rääbis, A. Rademaker, T. Rama, L. Ramasamy, C. Ramisch, F. Rashel, M. S. Rasooli, V. Ravishankar, L. Real, P. Rebeja, S. Reddy, G. Rehm, I. Riabov, M. Rießler, E. Rimkutė, L. Rinaldi, L. Rituma, L. Rocha, E. Rögnvaldsson, M. Romanenko, R. Rosa, V. Roșca, D. Rovati, O. Rudina, J. Rueter, K. Rúnarsson, S. Sadde, P. Safari, B. Sagot, A. Sahala, S. Saleh, A. Salomoni, T. Samardžić, S. Samson, M. Sanguinetti, D. Särg, B. Saulīte, Y. Sawanakunanon, K. Scan-

nell, S. Scarlata, N. Schneider, S. Schuster, D. Seddah, W. Seeker, M. Seraji, M. Shen, A. Shimada, H. Shirasu, M. Shohibussirri, D. Sichinava, E. F. Sigurdsson, A. Silveira, N. Silveira, M. Simi, R. Simionescu, K. Simkó, M. Šimková, K. Simov, M. Skachedubova, A. Smith, I. Soares-Bastos, C. Spadine, S. Steingrímsson, A. Stella, M. Straka, E. Strickland, J. Strnadová, A. Suhr, Y. L. Sulestio, U. Sulubacak, S. Suzuki, Z. Szántó, D. Taji, Y. Takahashi, F. Tamburini, M. A. C. Tan, T. Tanaka, S. Tella, I. Tellier, G. Thomas, L. Torga, M. Toska, T. Trosterud, A. Trukhina, R. Tsarfaty, U. Türk, F. Tyers, S. Uematsu, R. Untilov, Z. Urešová, L. Uria, H. Uszkoreit, A. Utka, S. Vajjala, D. van Niekerk, G. van Noord, V. Varga, E. Villemonte de la Clergerie, V. Vincze, A. Wakasa, J. C. Wallenberg, L. Wallin, A. Walsh, J. X. Wang, J. N. Washington, M. Wendt, P. Widmer, S. Williams, M. Wirén, C. Wittern, T. Woldemariam, T.-s. Wong, A. Wróblewska, M. Yako, K. Yamashita, N. Yamazaki, C. Yan, K. Yasuoka, M. M. Yavrumyan, Z. Yu, Z. Žabokrtský, S. Zahra, A. Zeldes, H. Zhu, and A. Zhuravleva. Universal dependencies 2.7, 2020. URL `http://hdl.handle.net/11234/1-3424`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

APPENDIX A

Training Details for Experiments with Latent Structures

We trained all models with AdamW optimizer (Kingma and Ba, 2014; Loshchilov and Hutter, 2018). The embeddings for the SST and SNLI experiments are initialized with Glove embeddings of size 300 (Pennington et al., 2014), available from `https://nlp.stanford.edu/projects/glove/`. The training details for all experiments in Chapter 4 are described in Table A.1.

Computing Infrastructure Each experiment was run on a single GPU. The setup of the computers we used is as follows:

- GPU: Titan Xp - 12GB
  CPU: 16 x AMD Ryzen 1950X @ 3.40GHz - 128GB
- GPU: RTX 2080 Ti - 12GB
  CPU: 12 x AMD Ryzen 2920X @ 3.50GHz - 128GB

	Synthetic Data	**SST**	**SNLI**
Data			
Where to get it	Generation script included	`https://nlp.stanford.edu/sentiment/`	`https://nlp.stanford.edu/projects/snli/`
Preprocesing	§4.5.1; attached code.	Neutral instances removed.	
Dataset size			
Training set	5000	6920	570K
Validation set	1000	872	10K
Test set	1000	1821	10K
Labels	2	2	3
Fixed hyperparameters			
Hidden size	100	100	200
Dropout	0	0	.2
Batch size	one batch	32	64
Number of epochs	10K	40	40
Optimized hyperparameters (maximizing validation accuracy)			
Learning rate ($\times 10^{-3}$)	$\{.1, 1, 2\}$	$\{.01, .02, .05, .1, .5, 1, 2\}$	$\{.01, .1, .3, 1, 3, 10\}$ (keeping $\eta = 1$)
Pullback step size η	$\{.1, 1, 2\}$	$\{.1, 1, 10\}$	$\{.001, .01, .1, 1, 10\}$ (for best learning rate)
Number of model parameters			
Baseline	2K	150K	340K
Model with latent structure	3K	180K	420K
Runtime (minutes)			
Baseline	< 1 / 1000 steps	< 1 / epoch	1 / epoch
Softmax / Marginals	1	3	4
Sparsemax / SparseMAP	1	3	25
Gumbel Softmax / Perturb-and-MAP	1	5	7
STE-Softmax / STE-Marginals	1	4	6
STE-Identity	1	2	5
SPIGOT	1	3	15
SPIGOT-CE	2	4	30
SPIGOT-EG	2	5	7
Best learning rate (and pullback step size, where applicable)			
Baseline	.001	.00002	.0001
Softmax / Marginals	.002	.0001	.0001
Sparsemax / SparseMAP	.001	.00005	.0003
Gumbel Softmax / Perturb-and-MAP	.002	.00005	.0001
STE-Softmax / STE-Marginals	.002	.00005	.0003
STE-Identity	.001	.0001	.0001
SPIGOT	.002 (.1)	.0001 (.1)	.0003 (1)
SPIGOT-CE	.001 (.1)	.00005 (.1)	.0001 (.1)
SPIGOT-EG	.001 (.1)	.00005 (.1)	.0001 (.001)

Table A.1: Training details and other reproducibility information.

APPENDIX B

Examples of Latent Trees

We performed a manual analysis of the trees output from the different models. We notice that, on the SST dataset, most latent trees produced by most models are flat. This agrees with related work (Williams et al., 2018; Niculae et al., 2018b). The notable exception is SPIGOT-CE, where the average tree depth on the test set is around 5 and trees seem more informative, suggesting benefits of the cross-entropy loss. Figs. B.1, B.2 and B.3 show examples of the trees produced from different models.

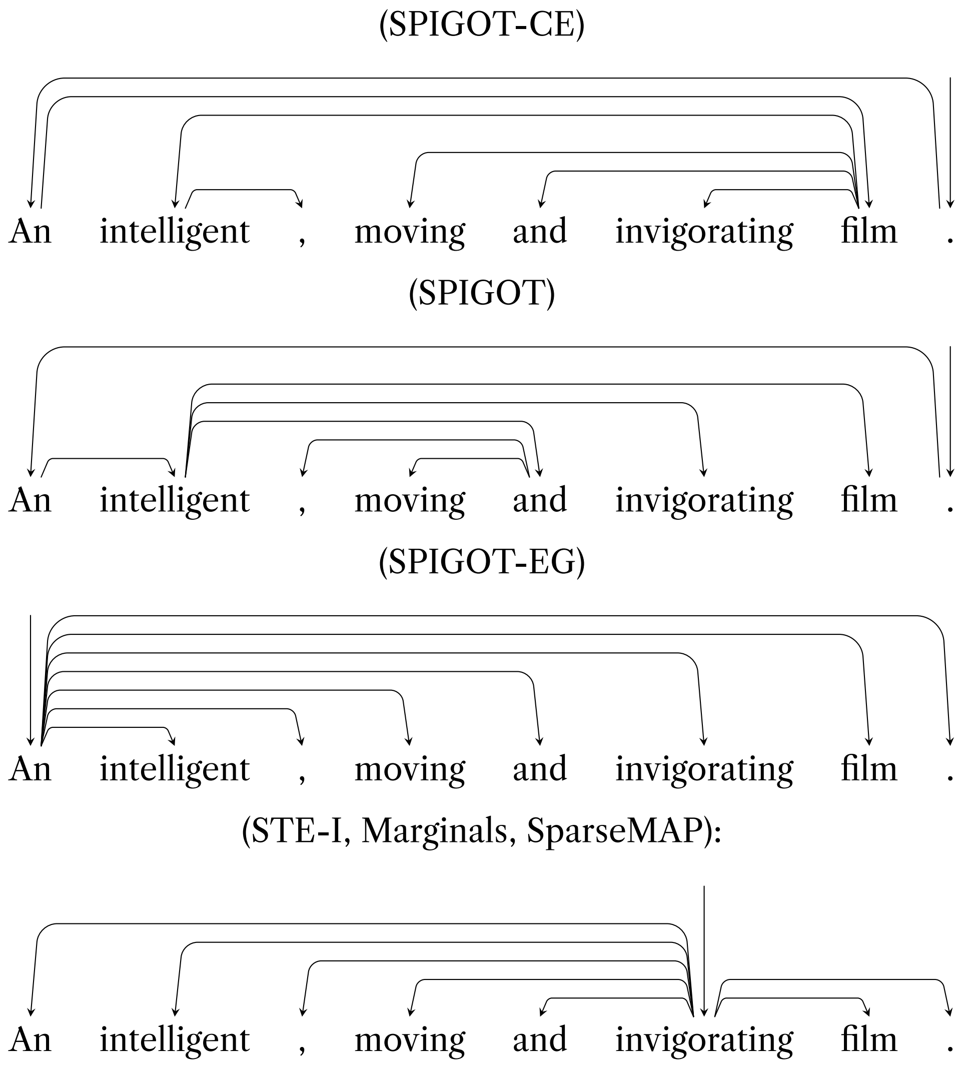

(SPIGOT-CE)
An intelligent , moving and invigorating film .
(SPIGOT)
An intelligent , moving and invigorating film .
(SPIGOT-EG)
An intelligent , moving and invigorating film .
(STE-I, Marginals, SparseMAP):
An intelligent , moving and invigorating film .

Figure B.1: Example of trees.

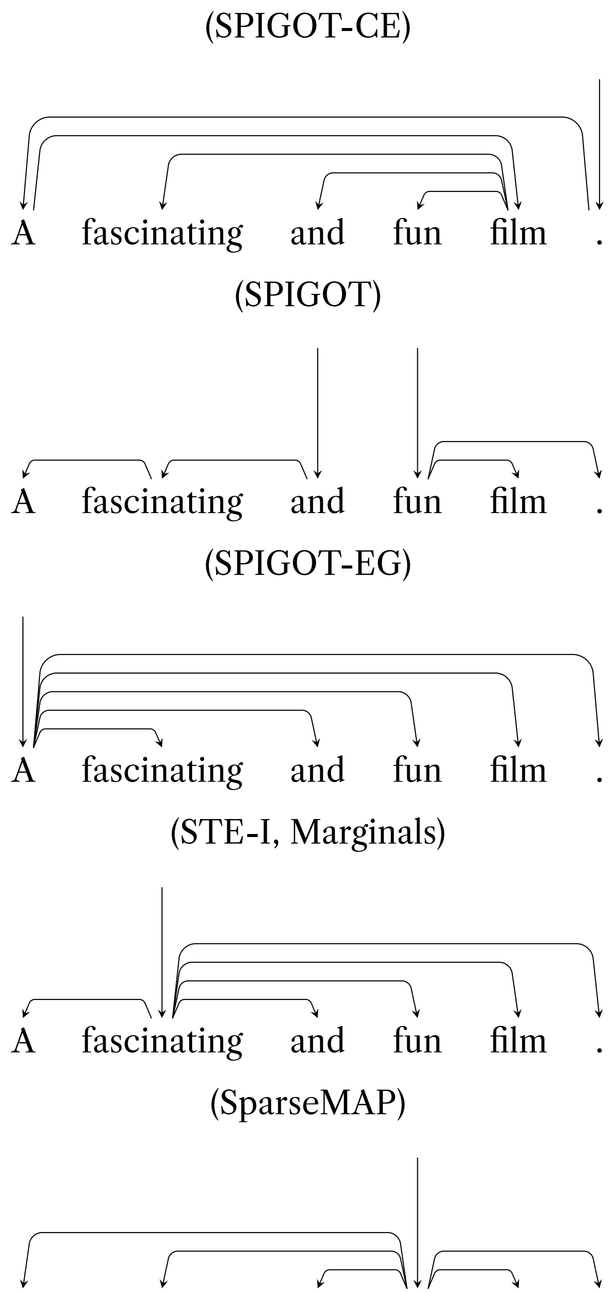

(SPIGOT-CE)
A fascinating and fun film .
(SPIGOT)
A fascinating and fun film .
(SPIGOT-EG)
A fascinating and fun film .
(STE-I, Marginals)
A fascinating and fun film .
(SparseMAP)
A fascinating and fun film .

Figure B.2: Example of trees.

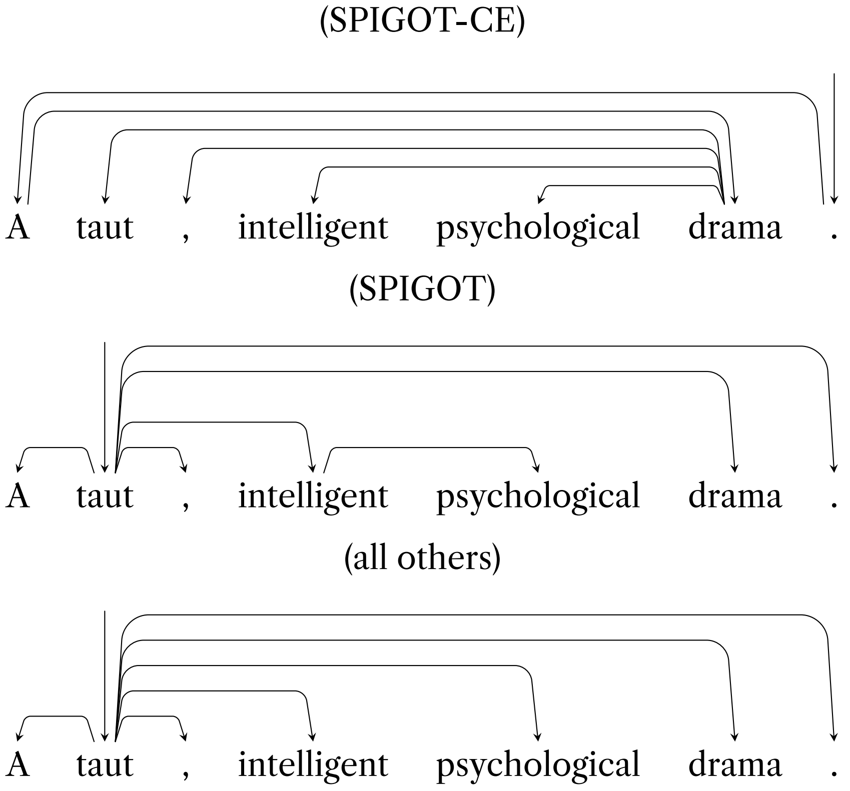

(SPIGOT-CE)
A taut , intelligent psychological drama .
(SPIGOT)
A taut , intelligent psychological drama .
(all others)
A taut , intelligent psychological drama .

Figure B.3: Example of trees produced by different models for the sentence "A taut, intelligent psychological drama." The majority of the models produce mostly flat trees. In contrast, SPIGOT-CE identifies the adjectives describing the keyword "drama" and attaches them correctly.

APPENDIX C

Proof of Lemma 1

We provide a more general proof for multilinear factor potentials, of which bilinear potentials are a special case. Let $\mathcal{G} = (V, F)$ be the factor graph underlying the UNN, with energy function $E(x_1, \ldots, x_n) = \sum_i E_{\mathsf{X}_i}(x_i) + \sum_f E_f(x_f)$. We assume $E_{\mathsf{X}_i}(x_i) = -b_i^\top x_i + \Psi_{\mathsf{X}_i}(x_i)$ for each $\mathsf{X}_i \in V$, with Ψ_{X_i} convex, and $E_f(x_f) = -\langle W_f, \otimes_{j \in f} x_j \rangle$ for each higher order factor $f \in F$ (multilinear factor energy), where $\otimes$ is the outer product, and W_f is a parameter tensor of matching dimension. For pairwise factors $f = \{\mathsf{X}_i, \mathsf{X}_j\}$, the factor energy is bilinear and can be written simply as $E_f(x_i, x_j) = -x_i^\top W_f x_j$.

The (block) coordinate descent algorithm updates each representation $x_i \in V$ sequentially, leaving the remaining representations fixed. Let $F(\mathsf{X}_i) = \{f \in F : \mathsf{X}_i \in f\} \subseteq F$ denote the set of factors X_i is linked to. The updates can be written as:

$$\begin{aligned}(x_i)_\star &= \arg\min_{x_i} E_{\mathsf{X}_i}(x_i) + \sum_{f\in F(\mathsf{X}_i)} E_f(x_f)\\ &= \arg\min_{x_i} \Psi_{\mathsf{X}_i}(x_i) \underbrace{-b_i^\top x_i - \sum_{f\in F(\mathsf{X}_i)} \langle W_f, \bigotimes_{j\in f} x_j\rangle}_{-z_i^\top x_i}\\ &= (\nabla\Psi^*_{\mathsf{X}_i})(z_i), \end{aligned} \tag{C.1}$$

where z_i is a pre-activation given by

$$z_i = \left(\sum_{f\in F(\mathsf{X}_i)} \rho_i(W_f) \bigotimes_{j\in f, j\neq i} x_j\right) + b_i, \tag{C.2}$$

and ρ_i is the linear operator that reshapes and rolls the axis of W_f corresponding to x_i to the first position. If all factors are pairwise, the update is more simply:

$$(x_i)_\star = (\nabla\Psi^*_{\mathsf{X}_i})\left(\sum_{f=\{\mathsf{X}_i,\mathsf{X}_j\}\in F(\mathsf{X}_i)} \rho_i(W_f)x_j + b_i\right), \tag{C.3}$$

where ρ_i is either the identity or the transpose operator. The update thus always consists in applying a (generally non-linear) transformation $\nabla\Psi^*_{\mathsf{X}_i}$ to an affine transformation of the neighbors of X_i in the graph (that is, the variables that coparticipate in some factor).

Therefore, given any topological order of the variable nodes in V, running k iterations of the coordinate descent algorithm following that topological order is equivalent to performing forward propagation in an (unrolled) directed acyclic graph, where each node applies affine transformations on input variables followed by the activation function $\nabla\Psi^*_{\mathsf{X}_i}$.

APPENDIX D

Derivation of updates for convolutional UNN

The two-layer convolutional UNN is defined by the pairwise energies

$$\begin{aligned} E_{\mathsf{XH}_1}(X, H_1) &= -\langle H_1, \mathcal{C}_1(X; W_1)\rangle\,, \\ E_{\mathsf{H}_1\mathsf{H}_2}(H_1, H_2) &= -\langle H_2, \mathcal{C}_2(H_1; W_2)\rangle\,, \\ E_{\mathsf{H}_2\mathsf{Y}}(H_2, y) &= -\langle y, V H_2\rangle\,, \end{aligned} \tag{D.1}$$

and the unary energies

$$\begin{aligned} E_{\mathsf{X}}(X) &= \frac{1}{2}\|X\|_2^2\,, \\ E_{\mathsf{H}_1}(H_1) &= -\langle H_1, b_1 \otimes 1_{d_1}\rangle + \Psi_{\tanh}(H_1)\,, \\ E_{\mathsf{H}_2}(H_2) &= -\langle H_2, b_2 \otimes 1_{d_2}\rangle + \Psi_{\tanh}(H_2)\,, \\ E_{\mathsf{Y}}(y) &= -\langle b, y\rangle - \mathcal{H}(y)\,. \end{aligned} \tag{D.2}$$

Above, $\mathcal{C}_1$ and $\mathcal{C}_2$ are are linear cross-correlation (convolution) operators with stride two and filter weights $W_1 \in \mathbb{R}^{32\times 1\times 6\times 6}$ and $W_2 \in \mathbb{R}^{64\times 32\times 4\times 4}$, and $b_1 \in \mathbb{R}^{32}$ and $b_2 \in \mathbb{R}^{64}$ are vectors of biases for each convolutional filter. The hidden activations have dimension $H_1 \in \mathbb{R}^{32\times(d_1)}$ and $H_2 \in \mathbb{R}^{64\times(d_2)}$, where d_1 and d_2 are tuples that depend on the input image size; for MNIST, $X \in \mathbb{R}^{1\times 28\times 28}$ leading to $d_1 = 12 \times 12$ and $d_2 = 5 \times 5$.

To derive the energy updates, we use the fact that a real linear operator $\mathcal{A}$ interacts with the Frobenius inner product as:

$$\langle P, \mathcal{A}(Q)\rangle = \langle Q, \mathcal{A}^\top(P)\rangle\,, \tag{D.3}$$

where $\mathcal{A}^\top$ is the transpose, or adjoint, operator.[1] If $\mathcal{C}$ is a convolution (*i.e.*, `torch.conv2d`) then $\mathcal{C}^\top$ is a deconvolution (*i.e.*, `torch.conv_transpose2d`) with the same filters. We then have

$$\begin{aligned} E_{\mathsf{XH_1}}(X, H_1) &= -\langle H_1, \mathcal{C}_1(X; W_1)\rangle = -\langle X, \mathcal{C}_1^\top(H_1; W_1)\rangle\,, \\ E_{\mathsf{H_1H_2}}(H_1, H_2) &= -\langle H_2, \mathcal{C}_2(H_1; W_2)\rangle = -\langle H_1, \mathcal{C}_2^\top(H_2; W_2)\rangle\,. \end{aligned} \tag{D.4}$$

[1]This generalizes the observation that $p^\top Aq = q^\top A^\top p$.

Adding up all energies and ignoring the constant terms in each update, we get

$$
\begin{aligned}
X_\star &= \operatorname*{argmin}_X -\langle X, \mathcal{C}_1^\top(H_1, W_1)\rangle + \Psi_{\tanh}(X) \\
&= \tanh(\mathcal{C}_1^\top(H_1, W_1))\,, \\
(H_1)_\star &= \operatorname*{argmin}_{H_1} -\langle H_1, \mathcal{C}_1(X, W_1)\rangle - \langle H_1, \mathcal{C}_2^\top(H_2, W_2)\rangle - \langle H_1, b_1 \otimes 1_{d_1\times d_1}\rangle + \Psi_{\tanh(H_1)} \\
&= \tanh(\mathcal{C}_1(X, W_1) + \mathcal{C}_2^\top(H_2, W_2) + b_1 \otimes 1_{d_1\times d_1})\,, \\
(H_2)_\star &= \operatorname*{argmin}_{H_2} -\langle H_2, \mathcal{C}_2(H_1, W_2)\rangle - \langle H_2, \sigma_y(V)y\rangle - \langle H_2, b_2 \otimes 1_{d_2\times d_2}\rangle + \Psi_{\tanh(H_2)} \\
&= \tanh(\mathcal{C}_2(H_1, W_2) + \sigma_y(V)y + b_2 \otimes 1_{d_2\times d_2})\,, \\
y_\star &= -\langle y, VH_2\rangle - \langle y, b\rangle - \mathcal{H}y \\
&= \operatorname{softmax}(VH_2 + b)\,.
\end{aligned}
\tag{D.5}
$$

Note that in our case, $H_2 \in \mathbb{R}^{64\times5\times5}$, $V \in \mathbb{R}^{10\times64\times5\times5}$ and thus $VH_2 \in \mathbb{R}^{10}$ is a tensor contraction (*e.g.*, `torch.tensordot(V, H_2, dims=3)`). The σ_y linear operator – opposite of ρ from Lemma 1 – rolls the axis of V corresponding to y to the *last* position, such that $\sigma_y(V) \in \mathbb{R}^{64\times5\times5\times10}$, the tensor analogue of a transposition (*e.g.*, `torch.permute(V, (1, 2, 3, 0))`).