Université de Montréal

Syntactic Inductive Biases for Deep Learning Methods

par

Yikang Shen

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

Août 4, 2022

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Syntactic Inductive Biases for Deep Learning Methods

présentée par

Yikang Shen

a été évaluée par un jury composé des personnes suivantes :

Sarath Chandar
(président-rapporteur)

Aaron Courville
(directeur de recherche)

Yoshua Bengio
(membre du jury)

Alexander Rush
(examinateur externe)

François Lareau
(représentant du doyen de la FESP)

Résumé

Le débat entre connexionnisme et symbolisme est l'une des forces majeures qui animent le développement de l'Intelligence Artificielle. L'apprentissage profond et la linguistique théorique sont les domaines d'études les plus représentatifs pour les deux écoles respectivement. Alors que la méthode d'apprentissage profond a fait des percées impressionnantes et est devenue la principale raison de la récente prospérité de l'IA pour l'industrie et les universités, la linguistique et le symbolisme occupent quelque domaines importantes, notamment l'interprétabilité et la fiabilité.

Dans cette thèse, nous essayons de construire une connexion entre les deux écoles en introduisant des biais inductifs linguistiques pour les modèles d'apprentissage profond. Nous proposons deux familles de biais inductifs, une pour la structure de circonscription et une autre pour la structure de dépendance. Le biais inductif de circonscription encourage les modèles d'apprentissage profond à utiliser différentes unités (ou neurones) pour traiter séparément les informations à long terme et à court terme. Cette séparation fournit un moyen pour les modèles d'apprentissage profond de construire les représentations hiérarchiques latentes à partir d'entrées séquentielles, dont une représentation de niveau supérieur est composée et peut être décomposée en une série de représentations de niveau inférieur. Par exemple, sans connaître la structure de vérité fondamentale, notre modèle proposé apprend à traiter l'expression logique en composant des représentations de variables et d'opérateurs en représentations d'expressions selon sa structure syntaxique. D'autre part, le biais inductif de dépendance encourage les modèles à trouver les relations latentes entre les mots dans la séquence d'entrée. Pour le langage naturel, les relations latentes sont généralement modélisées sous la forme d'un graphe de dépendance orienté, où un mot a exactement un nœud parent et zéro ou plusieurs nœuds enfants. Après avoir appliqué cette contrainte à un modèle de type transformateur, nous constatons que le modèle est capable d'induire des graphes orientés proches des annotations d'experts humains, et qu'il surpasse également le modèle de transformateur standard sur différentes tâches. Nous pensons que ces résultats expérimentaux démontrent une alternative intéressante pour le développement futur de modèles d'apprentissage profond.

Mots-clés: Intelligence Artifcielle, Apprentissage Profond, Biais inductifs Linguistiques, Biais inductif de Circonscription, Biasis inductif de Dépendance.

Abstract

The debate between connectionism and symbolism is one of the major forces that drive the development of Artificial Intelligence. Deep Learning and theoretical linguistics are the most representative fields of study for the two schools respectively. While the deep learning method has made impressive breakthroughs and became the major reason behind the recent AI prosperity for industry and academia, linguistics and symbolism still holding some important grounds including reasoning, interpretability and reliability.

In this thesis, we try to build a connection between the two schools by introducing syntactic inductive biases for deep learning models. We propose two families of inductive biases, one for constituency structure and another one for dependency structure. The constituency inductive bias encourages deep learning models to use different units (or neurons) to separately process long-term and short-term information. This separation provides a way for deep learning models to build the latent hierarchical representations from sequential inputs, that a higher-level representation is composed of and can be decomposed into a series of lower-level representations. For example, without knowing the ground-truth structure, our proposed model learns to process logical expression through composing representations of variables and operators into representations of expressions according to its syntactic structure. On the other hand, the dependency inductive bias encourages models to find the latent relations between entities in the input sequence. For natural language, the latent relations are usually modeled as a directed dependency graph, where a word has exactly one parent node and zero or several children nodes. After applying this constraint to a transformer-like model, we find the model is capable of inducing directed graphs that are close to human expert annotations, and it also outperforms the standard transformer model on different tasks. We believe that these experimental results demonstrate an interesting alternative for the future development of deep learning models.

Keywords: Artificial Intelligence, Deep Learning, Syntactic inductive Biases, Constituency inductive Biases, Dependency inductive Biases.

Contents

List of tables

List of figures

List of acronyms and abbreviations

NLP	Natural Language Processing
UF1	Unlabeled F1 score for evaluating constituency parsing performance
UAS	Unlabeled (directed) Attachment Score for evaluating dependency parsing performance
UUAS	Unlabeled Undirected Attachment Score for evaluating dependency parsing performance
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory
CNN	Convolutional Neural Network

Acknowledgements

The past five years at the University of Montreal and Mila have been the most valuable and unforgettable experience for me so far. It's not easy to say goodbye to such a wonderful time, as well as the people and life. When I started my Ph.D. study in September 2016, I only have a vague idea about the path and challenges in front of me. In 2016, deep learning is already on its hype, some researchers suggested that its era will end in a few years. But the fact is that exciting breakthrough and challenge new questions still emerge every year. The technique gradually becomes an essential part of everyone's life. From self-driving car to the photo album in a smartphone, they all include at least one deep learning model to provide some functions that you wouldn't expect 10 years ago. It's fair to say that the development of Artificial Intelligence (AI, mostly deep learning) is reforming our society. Although I have some different opinions on the future directions and still believe in them. But it doesn't stop me from feeling excited about what is happening over the last few years and now. It's a great honor to witness and participate in this evolution in the front seat. I would not have been able to make this journey without the help and support of many people and I feel deeply indebted to them.

My greatest thanks should go to my advisor Aaron Courville. It was a great privilege to work with one of the most important researchers in the deep learning community. I can always rely on him to provide good advice for my research as well as my career. I am not a student who always listens carefully to the professor's advice, but Aaron is always supportive. His support is the most important reason that allows me to focus on studying a relatively niche field. He has a very insightful, high-level view of the field while he can also quickly understand details and understands the nature of the problems very well. More importantly, he always encourages collaboration, not only between his students but also with other professors and external researchers. Collaborations grant me access to lots of mental resources and allow me to understand problems from very different prospects.

I would like to thank Alessandro Sordoni and Siva Reddy – the other two mentors in my Ph.D. study, for a lot of guidance and help throughout the last few years. Alessandro is my host during my 3-years long part-time research internship at Microsoft Research (this program also provides many free meals and unexpected high life quality for a Ph.D. student).

He is a good criticizer and almost like a co-supervisor for me. I can only have enough confidence in my work after passing his trial of insightful questions. Siva is a charming person and knowledgeable NLP researcher. He provides a lot of valuable linguistic-related advice, while I am in desperate need of this guidance and both Aaron and Alessandro focus their study on the machine learning field. Personally, his attitude towards life also affects me. I want to be the same energetic and responsible person as him someday. I want to thank Aaron, Alessandro, and Siva for all the guidance and patience that they have provided, and I will always be proud to be your student.

I also want to thank Shawn Tan and Zhouhan Lin. They're the most important collaborators for me. Almost all of my works are done in close collaborations with either Zhouhan or Shawn. I want to thank Shawn for the countless discussions that we have had. He is capable of understanding the most complicated idea and formalize it with clear and rigorous math equations. Together, we wrote Ordered Neurons [Shen et al., 2018c] and some other papers that this thesis is based on. I want to thank Zhouhan for been the big buddy that helps me adapt to my Ph.D. life and booting my research. Our works on unsupervised and supervised parsing [Shen et al., 2017, 2018b] leads me to the works presented in this thesis. We are also very close friends. I want to thanks them for providing lots of support in my difficult time and tolerate my occasional aggressiveness.

I want to thank the Mila community. It gathers a lot of brilliant researchers and created a unique atmosphere of collaboration. I met many interesting people in Mila. Among them, I especially want to thank Ying Zhang, Jie Fu, Jae Hyun Lim, Min Lin, Chin-Wei Huang, Saizheng Zhang, Yuchen Lu, Xing Chen, Amina Madzhun, Zhen Liu. I had a lot of joyful times with them. They also gave me a lot of help at various times. Outside of Mila, I also met many great friends including Peng Lu, Yi Tay, Che Zheng, Xingdi Yuan, Lili Mou (just to name a few). I also want to thank my parents for their unwavering support and love during the last three decades.

Chapter 1

Introduction

1.1. The Evolution of Paradigm in NLP

While research in Natural Language Processing (NLP) is now dominant by deep learning methods, the two fields were separately developed for decades. Prior to the deep learning revolution, the difference between the two domains can trace back to the difference between connectionism and symbolism.

Before the 1990s, approaches for solving NLP problems were predominantly *symbolic* [Chao, 1968]. Symbolic systems directly model abstract concepts and the innate structure of the human mind. Elementary semantics are represented by symbols. Complex semantics are represented by a group of symbols combined by operations and syntactic structures. Many early AI advances utilized a symbolic approach to AI programming, striving to create smart systems by modeling relationships and using symbols and programs to convey meaning. Symbolic systems also allow straightforward generalization through assembling known rules and symbols into a new syntactic structure. But symbolic systems have several defects: 1) limited fault tolerance, a failure of a small component usually causes the entire system break; 2) they can't process inputs that include undefined rules or symbols resulting in a relatively limited learning capacity; 3) applying symbolic methods to a new problem requires lots of human expertise.

In the 1990s, the paradigm shifts from symbolic methods to *empirical* or *statistical* methods [Abney, 1996]. The empirical view assumes that language is a natural phenomenon whose effects are observable in the world as data. The best way to build a NLP model is to learn from the data. Empirical methods nicely solved some defects of the symbolic methods. For example, a probabilistic framework can handle the ambiguity in natural language by assigning probabilities to different analyses. Applying an empirical method to a new problem could be as simple as training the method on a set of pre-collected data. These methods enjoyed great success in almost all problems in natural language processing until deep learning methods became prominent.

Starting from the middle of the 2010s, the paradigm shifts again to *deep learning* methods, which try to process input with a powerful universal function approximator, without explicitly modeling discrete structure and operations. The idea of deep learning can trace its roots back to the concept of connectionism. Some advantages of the connectionist approach include its applicability to a broad array of functions, a structural approximation to biological neurons, low requirements for innate structure, and capacity for graceful degradation. Recent progress shows that deep learning models could be trained in an end-to-end schema from human annotations or unsupervised losses. Deep learning models can be easily applied to different tasks that have enough training data and clear input/output definitions. The capacity of a neural model can be easily improved by increasing the number of parameters. The data-driven feature and expansibility of the deep learning model make it very popular in industrial applications. Recent studies show that deep learning-based NLP models have proven to be capable of learning a remarkable amount of syntax, despite having much weaker structural priors than Chomsky's model of Universal Grammar.

1.2. Motivation

Despite being very popular and empirically successful, the limitation of deep learning methods is starting to draw more attention: 1) they heavily rely on training data and computation resources, GPT-3 [Brown et al., 2020] has 170 billion parameters and is trained on a dataset of about 500 billion tokens; 2) they fail to generalize to unexpected data points, for example, an unforeseen combination of known tokens and extra-long inputs [Tay et al., 2020]; 3) it lacks explainability, state-of-the-art deep learning models are like black boxes, such that it is almost always impossible to understand why decisions are made, and it's also impossible to identify the source of an error or fix the error in a way that will not potentially hurt the other functionalities of the model [Zhang and Zhu, 2018]. However, even with exponential increases in computing power and increasingly vast quantities of data, the improvement curve for predictive power is leveling off, suggesting that there is a cap to how far connectionism can take us.

One major difference between deep learning methods and many previous approaches is the assumption that there exist a syntax and a set of semantic functions. Syntax is the set of rules, principles, and processes that govern the structure of sentences (sentence structure) in a given language, usually including word order. The syntax of a language describes the latent structure of a valid sentence, but does not provide any information about the meaning of that sentence. The meaning given to a combination of symbols is handled by semantics. In other words, for a given sentence $x_1, ..., x_T$, its syntactic structure can be seen as a computation graph $\mathbf{G}$ that is equipped with semantic functions. The computation graph $\mathbf{G}$ takes the meaning of each token x_i as input, and outputs the meaning of the sentence.

This assumption provides powerful generalizations and explainability. The disentanglement of structure and functions allows the same set of semantic functions to be combined in many different ways to process different inputs and solve different tasks. The existence of a clear structure also provides rich and meaningful intermediate results. These advantages provide a potential solution to the defects of deep learning.

1.3. Syntactic Inductive Biases

In this thesis, we study the problem of building syntax-sensitive deep learning models. In particular, we introduce **syntactic inductive biases** – a new family of inductive biases that use a probabilistic relaxation of discrete syntactic structure to regularize the internal connection of deep learning models.

An *inductive bias* of a learning algorithm is a set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered [Mitchell, 1980]. In the field of deep learning, inductive bias usually refers to special neural network architecture designs, that leverage prior knowledge to regularize a more general version of deep learning model. These special designs encourage the model to prioritise solutions with specific properties that requires hand-engineering in traditional methods. One famous example of an inductive bias for deep learning is Convolutional Neural Network (CNN). CNNs regularize multilayer perceptrons through weight sharing. Multilayer perceptrons are fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. CNNs use convolution in place of general matrix multiplication in at least one of their layers. CNNs learn to optimize the filters (or kernels) through automated learning, whereas in traditional methods these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

Since there are two major classes of natural language syntax: dependency grammar and constituency grammar. We introduce two respective types of inductive bias: *constituency inductive bias* and *dependency inductive bias*.

Constituency grammars model the assembly of one or several corresponded words. From a syntactic point of view, a constituent is a word or a group of words that function as a single unit within a hierarchical structure. From a semantic point of view, a constituent is a unit with a stand-alone meaning. Although some syntactically well-formed constituents are nonsensical, e.g."Colorless green ideas sleep furiously". For most NLP applications, a constituent should be syntactically and semantically well-formed. Furthermore, in constituency trees, larger constituents are composed of smaller constituents. The process of composition is modeled by compositional semantic functions. Traditionally, constituency grammars treat the structure as primitive. Constituency grammars derive the functions from the constellation. For instance, the object is identified as the NP appearing inside finite VP, and the subject

as the NP appearing outside of finite VP. The *constituency inductive bias* should encourage deep learning models to: 1) induce the latent constituency structure of input sentences, 2) model the compositional semantic functions, 3) compute meaningful representations for constituents.

Dependency grammars model one-to-one correspondences between words. In a dependency graph, the main verb is taken to be the structural center of a clause structure. All other syntactic units (words) are either directly or indirectly connected to the verb in terms of the directed links, which are called dependencies. Dependency grammars have flatter tree structures than constituency grammars in part because they lack a finite verb phrase constituent, and they are thus well suited for the analysis of languages with free word order, such as Czech or Warlpiri. Different from constituency grammars, dependency grammars treat the syntactic functions as primitive. They posit an inventory of functions (e.g. subject, object, oblique, determiner, attribute, predicative, etc.). These functions can appear as labels on the dependencies in the tree structures. The *dependency inductive bias* should: 1) explicitly model different syntactic functions as separate modules, 2) induce latent dependency edges between words, 3) compute meaningful representations for each node (words) in the dependency graph.

Furthermore, the idea of syntactic inductive bias is not exclusive for NLP tasks. In fact, one school of thought sees syntax as a non-innate adaptation to innate cognitive mechanisms [Hawkins, 2004]. Cross-linguistic tendencies are considered as being based on language users' preference for grammars that are organized efficiently, and on their avoidance of word orderings which cause processing difficulty. This suggest that an effective syntactic inductive bias could capture some fundamental cognitive mechanisms. In the real world, lots of tasks have sparse rewards and long time horizons, which typically pose significant challenges in reinforcement learning. HIL and HRL handle this problem by introducing a temporal abstraction [Stolle and Precup, 2002] – a innate hierarchical structure over time. This idea is coherent with constituency structure. Based on this observation, we extend the application of our constituency inductive bias to Hierarchical Imitation Learning (HIL) and Hierarchical Reinforcement Learning (HRL) domain. In the setting of HIL and HRL, we expect the constituency inductive bias can encourage the agent to: 1) segment a given task into meaningful subtasks; 2) model the subtasks as separate skills (neural network modules); 3) recompose these skills to solve a new tasks.

1.4. Why not Supervised Parsing?

Learning structures from treebanks has its advantages – it is very well studied, a supervised parsing can achieve very high accuracy. But a trained parsing has the limitation

of *domain dependence*. Parsers trained on English Penn Treebank [Marcus et al., 1994] degrade substantially when applied to new genres and domains, and fail when applied to new languages. This has spurred an area of research on parser adapation [Üstün et al., 2020, Zeman and Resnik, 2008, McClosky et al., 2006]. Treebanks now exist for several domains and languages, but each treebank requires many resources and years of work to construct, and most languages are without treebank.

Taking the domain dependency problem to the extreme, we have *modality* problem. Modern deep learning models, like Transformer [Vaswani et al., 2017], are considered as foundation models for AI [Bommasani et al., 2021]. A new deep learning technique could potentially be applied to many different modalities (e.g. vision, robotic, and language), and some multi-modality problems. Many of these domains have latent structures. For example, images can be parsed to a hierarchical structure [Tu et al., 2005], hierarchical reinforcement learning [Barto and Mahadevan, 2003] is an important technique for robotic. But most of these domains don't have a convention for human-labeled latent structure. A model that requires structural supervision would have limited versatility.

A further limitation of treebanks is that they always have a fixed convention. But the "right" kind of syntactic structure seems to depend heavily on the beneficiary task. There have been studies comparing different kinds of syntactic structures for their usefulness on specific tasks [Gildea, 2004]. Finetuning pretrained language models on a specific task makes them learn a convention that suits better the targeted task [Dai et al., 2021]. From this perspective, a method that can induce syntactic structure from downstream tasks could be more useful than a supervised parser.

1.5. Thesis Outline

Following the discussion in previous section, the rest of this thesis is organized around following themes:

- In Chapter 2, we review some widely used neural network components and architectures. These components and architectures are foundation of models introduced in this thesis. We then provide definitions to the tasks that we will use to evaluate syntactic inductive biases.
- In Chapter 3, we first review the history of constituency-augmented neural networks and constituency inductive biases. We then introduce *Ordered Neurons* – a constituency inductive bias for recurrent neural networks. Based on the ordered neurons, we further introduce two neural network architectures: ON-LSTM and Ordered Memory. We present experiment results on formal language tasks, language modeling, and unsupervised constituency parsing.

- In Chapter 4, we first review the history of dependency-augmented neural networks and dependency inductive biases. We then introduce *dependency-constrained connection* – an inductive bias for transformer or graph neural networks. We present experiment results on masked language modeling, unsupervised dependency parsing and semantic textual similarity tasks.
- In Chapter 5, we first review the history of Hierarchical Imitation and Reinforcement Learning (HIRL). We then introduce the Option-Controller Network – an HIRL model with a constituency inductive bias. In experiments, we perform behavior cloning from unstructured demonstrations coming from different tasks, and during the RL finetuning, we freeze the learned options and only re-initialize the controller.
- In Chapter 6, we first conclude the thesis, then review three important research questions in this field. The first question is emerging better and more discrete structure. The second question discusses inducing reusable operators. The last question discusses systematic generalization based on the induced structure and operators. These questions still remain as open questions and yet to be answered in the future.

1.6. Contributions

The contributions of this thesis are summarized as follows:

- I systematically studied syntactic inductive bias for deep learning methods. I studied a wide spectrum of neural network architecture design. I summarized our works and proposed two inductive biases: constituency inductive bias and dependency inductive bias.
- I were among the first to study neural unsupervised parsing. I proposed several models that can achieve strong results on unsupervised dependency and constituency parsing.
- I also show that understanding the latent structure of input is essential for strong generalization on language tasks.
- I further expand the application of syntactic inductive bias to hierarchical imitation and reinforcement learning domain. This application result in a highly modularized model that can effectively reuse learnt skills to solve new tasks.

1.7. Article Details

This thesis includes materials from four papers, that I wrote as the first author or co-first author. This section provides a detailed list of these papers and explanations of my contribution to these papers:

- **Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks.** Yikang Shen, Shawn Tan, Alessandro Sordoni, Aaron Courville. *International Conference on Learning Representations, 2019.*

  Personal Contribution. I proposed the idea of Ordered Neurons; implemented the ON-LSTM model; wrote the introduction, model, and most of the experiment section; conducted the language model, unsupervised constituency parsing, and targeted syntactic evaluation experiments. Shawn Tan was intensively involved in the discussions that results in the idea, wrote the related work section, significantly contributed to the writing of other sections, and conducted the logical inference experiments. Alessandro Sordoni and Aaron Courville co-supervised the project and significantly contributed to the writing.
- **Ordered Memory.** Yikang Shen, Shawn Tan, Arian Hosseini, Zhouhan Lin, Alessandro Sordoni, Aaron Courville. *Advances in Neural Information Processing Systems, 2019.*

  Personal Contribution. I proposed the idea of Ordered Memory; implemented the model; wrote most of the paper; conducted the ListOps experiment. Shawn was heavily involved in the discussion about model details; significantly contributed to the paper writing; conducted the logical inference experiment. Arian Hosseini conducted the sentiment analysis experiment. Zhouhan Lin was heavily involved in the discussion and paper writing. Alessandro Sordoni and Aaron Courville co-supervised the project; were heavily involved in the discussion; contributed to the writing.
- **Unsupervised Dependency Graph Network.** Yikang Shen, Shawn Tan, Alessandro Sordoni, Peng Li, Jie Zhou, Aaron Courville. *Annual Meeting of the Association for Computational Linguistics, 2022.*

  Personal Contribution. I proposed the idea of UDGN; implemented the model; wrote the introduction, model, and most of the experiment section; conducted the language model, unsupervised parsing experiment, and ablation study. Shawn Tan wrote the related work section; conducted the language model and finetuning experiments. Alessandro Sordoni and Aaron Courville co-supervised the project, were heavily involved in the discussion, contributed to the writing. Peng Li and Jie Zhou contributed to the writing.
- **Inducing Reusable Skills From Demonstrations with Option-Controller Network.** Siyuan Zhou, Yikang Shen, Yuchen Lu, Aaron Courville, Joshua B. Tenenbaum, Chuang Gan.

  Personal Contribution. I proposed the idea of the Option-Controller Network; implemented the model; wrote the introduction and the model section. Siyuan Zhou conducted the experiments; wrote the experiment section. Yuchen Lu and Chuang

Gan were heavily involved in the discussion; significantly contributed to the writing. Aaron Courville and Joshua B. Tenebaum co-supervised the project.

Chapter 2

Preliminaries

2.1. Neural Network Components and Architectures

2.1.1. Word embeddings

The first key idea of word embeddings is to represent words as low-dimensional (e.g., 300), real-valued vectors. Before deep learning, it was common to represent a word as an index into the vocabulary, which is a notational variant of using one-hot word vectors: each word is represented as a high-dimensional, sparse vector where only one entry of that word is 1 and all other entires are 0's:

$$\mathbf{v}_{\text{car}} = [0, 0, \ldots, 0, 0, 1, 0, \ldots, 0]^\intercal$$
$$\mathbf{v}_{\text{vehicle}} = [0, 1, \ldots, 0, 0, 0, 0, \ldots, 0]^\intercal$$

The biggest problem with these sparse vectors is that they don't share any semantic similarity between words, i.e., for any pair of different words a, b, $\cos(\mathbf{v}_a, \mathbf{v}_b) = 0$.

Low-dimensional word embeddings effectively alleviated this problem and similar words can be encoded as similar vectors in space: $\cos(\mathbf{v}_{\text{car}}, \mathbf{v}_{\text{vechicle}}) < \cos(\mathbf{v}_{\text{car}}, \mathbf{v}_{\text{man}})$. These word embeddings can be effectively learned from large unlabeled text corpora, based on the assumption that words occurred in similar contexts tend to have similar meanings (a.k.a. the *distributional hypothesis*) [Harris, 1954]. Indeed, learning word embeddings from text has a long-standing history and has been popularized by scalable algorithms and released sets of pretrained word embeddings such as WORD2VEC [Mikolov et al., 2013], GLOVE [Pennington et al., 2014] and FASTTEXT [Bojanowski et al., 2017]. In modern NLP models [Vaswani et al., 2017, Devlin et al., 2018], word embeddings are still essential components for representing the input and output tokens. But they are optimized together with the other components of neural network models.

2.1.2. Recurrent neural networks

Recurrent neural networks are a class of neural networks which are suitable to handle sequences of variable length. More concretely, they apply a parameterized function recursively on a sequence $\mathbf{x}_1, \ldots, \mathbf{x}_n$:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \Theta) \tag{2.1.1}$$

For NLP applications, we represent a sentence or a paragraph as a sequence of words where each word is transformed into a vector (usually through pre-trained word embeddings): $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and $\mathbf{h}_t \in \mathbb{R}^h$ can be used to model the contextual information of $\mathbf{x}_{1:t}$.

Vanilla RNNs take the form:

$$\mathbf{h}_t = \tanh(\mathbf{W}^{hh}\mathbf{h}_{t-1} + \mathbf{W}^{hx}\mathbf{x}_t + \mathbf{b}), \tag{2.1.2}$$

where $\mathbf{W}^{hh} \in \mathbb{R}^{h\times h}, \mathbf{W}^{hx} \in \mathbb{R}^{h\times d}$, $\mathbf{b} \in \mathbb{R}^h$ are the parameters to be learned. To ease the optimization, many variants of RNNs have been proposed. Among them, long short-term memory networks (LSTMs) [Hochreiter and Schmidhuber, 1997] and gated recurrent units (GRUs) [Cho et al., 2014] are the commonly used ones. Arguably, LSTM is still the most competitive RNN variant for NLP applications today and also our default choice for the neural models that we will describe. Mathematically, LSTMs can be formulated as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}^{ih}\mathbf{h}_{t-1} + \mathbf{W}^{ix}\mathbf{x_t} + \mathbf{b}^i) \tag{2.1.3}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^{fh}\mathbf{h}_{t-1} + \mathbf{W}^{fx}\mathbf{x_t} + \mathbf{b}^f) \tag{2.1.4}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^{oh}\mathbf{h}_{t-1} + \mathbf{W}^{ox}\mathbf{x_t} + \mathbf{b}^o) \tag{2.1.5}$$

$$\mathbf{g}_t = \tanh(\mathbf{W}^{gh}\mathbf{h}_{t-1} + \mathbf{W}^{gx}\mathbf{x_t} + \mathbf{b}^g) \tag{2.1.6}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \tag{2.1.7}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{2.1.8}$$

where $\mathbf{W}^{ih}, \mathbf{W}^{fh}, \mathbf{W}^{oh}, \mathbf{W}^{gh} \in \mathbb{R}^{h\times h}$, $\mathbf{W}^{ix}, \mathbf{W}^{fx}, \mathbf{W}^{ox}, \mathbf{W}^{gx} \in \mathbb{R}^{h\times d}$ and $\mathbf{b}^i, \mathbf{b}^f, \mathbf{b}^o, \mathbf{b}^g \in \mathbb{R}^h$ are the parameters to be learned.

Finally, a useful elaboration of an RNN is a *bidirectional RNN*. The idea is simple: for a sentence or a paragraph, $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_n$, a forward RNN is used from left to right and then another backward RNN is used from right to left:

$$\overrightarrow{\mathbf{h}}_t = f(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{x}_t; \overrightarrow{\Theta}), \quad t = 1, \ldots, n \tag{2.1.9}$$

$$\overleftarrow{\mathbf{h}}_t = f(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{x}_t; \overleftarrow{\Theta}), \quad t = n, \ldots, 1 \tag{2.1.10}$$

We define $\mathbf{h}_t = [\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t] \in \mathbb{R}^{2h}$ as the concatenation of the hidden vectors from the RNNs in both directions. These representations can usefully encode information from both the

left context and the right context and are suitable for general-purpose trainable feature-extracting component of many NLP tasks.

2.1.3. Attention mechanism

The third important component is an attention mechanism. It was first introduced in the *sequence-to-sequence* (seq2seq) models Sutskever et al. [2014] for neural machine translation [Bahdanau et al., 2014, Luong et al., 2015] and has later been extended to other NLP tasks.

Before introducing attention mechanism, if we want to predict the sentiment of a sentence, or translate a sentence of one language to the other, we apply recurrent neural networks to encode a single sentence (or the source sentence for machine translation): $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n$ and use the last time step $\mathbf{h}_n$ to predict the sentiment label or the first word in the target language. The label probability or first-word distribution can be modeled by the softmax function:

$$P(Y = y) = \text{softmax}\left(\mathbf{W}_y \mathbf{h}_n\right) = \frac{\exp(\mathbf{W}_y \mathbf{h}_n)}{\sum_{y'} \exp\left(\mathbf{W}_{y'} \mathbf{h}_n\right)} \tag{2.1.11}$$

This requires the model to be able to compress all the necessary information of a sentence into a fixed-length vector. This information bottleneck could result in a loss of details. An attention mechanism is designed to solve this problem: instead of squashing all the information into the last hidden vector, it looks at the hidden vectors at all time steps and chooses a subset of these vectors adaptively:

$$\alpha_i = \frac{\exp\left(g(\mathbf{h}_i, \mathbf{w}; \Theta_g)\right)}{\sum_{i'=1}^{n} \exp\left(g(\mathbf{h}_{i'}, \mathbf{w}; \Theta_g)\right)} \tag{2.1.12}$$

$$\mathbf{c} = \sum_{i=1}^{n} \alpha_i \mathbf{h}_i \tag{2.1.13}$$

Here $\mathbf{w}$ can be a task-specific vector learned from the training process, or taken as the current target hidden state in machine translation and g is a parameteric function which can be chosen in various ways, such as dot product, bilinear product, or one hidden layer of an MLP:

$$g_{\text{dot}}(\mathbf{h}_i, \mathbf{w}) = \mathbf{h}_i{}^{\intercal} \mathbf{w} \tag{2.1.14}$$

$$g_{\text{bilinear}}(\mathbf{h}_i, \mathbf{w}) = \mathbf{h}_i{}^{\intercal} \mathbf{W} \mathbf{w} \tag{2.1.15}$$

$$g_{\text{MLP}}(\mathbf{h}_i, \mathbf{w}) = \mathbf{v}^{\intercal} \tanh(\mathbf{W}^h \mathbf{h}_i + \mathbf{W}^w \mathbf{w}) \tag{2.1.16}$$

An attention mechanism computes a similarity score for each $\mathbf{h}_i$ and then a softmax function is applied which returns a discrete probability distribution over all time steps. Thus α essentially captures which parts of the sentence are relevant and $\mathbf{c}$ aggregates over all the

time steps with a weighted sum and can be used for final prediction. Interested readers are referred to Bahdanau et al. [2014], Luong et al. [2015] for more details.

Attention mechanisms have proved widely effective in numerous applications and become an integral part of neural NLP models. Parikh et al. [2016] and Vaswani et al. [2017] conjectured that attention mechanisms don't have to be used in conjunction with recurrent neural networks and can be built purely based on word embeddings and feed-forward networks while providing minimal sequence information. This class of models usually requires fewer parameters and is more parallelizable and scalable — in particular, the TRANSFORMER model proposed in Vaswani et al. [2017] has become a recent trend and our model also has a strong connection to it. Recent developments of pretrained language model [Devlin et al., 2018, Brown et al., 2020] have made the attention mechanism one of the most important components for modern NLP models.

2.1.4. Memory Networks

A memory network is a neural network extended with a memory component that can be read and written to. The model is trained to learn how to operate effectively with the memory component. The high-level view of a memory network is as follows:

- There is a memory, **m**, an indexed array of objects (e.g. vectors or arrays of strings).
- An input feature map **I**, which converts the incoming input to the internal feature representation.
- A writing component **W** which updates old memories given the new input.
- An output feature map **O**, which produces a new output in the feature representation space given the new input and the current memory state.
- A response component **R** which converts the output into the response format desired – for example, a textual response or an action.

I, **W**, **O** and **R** can all potentially be learned components and make use of any ideas from the existing machine learning literature.

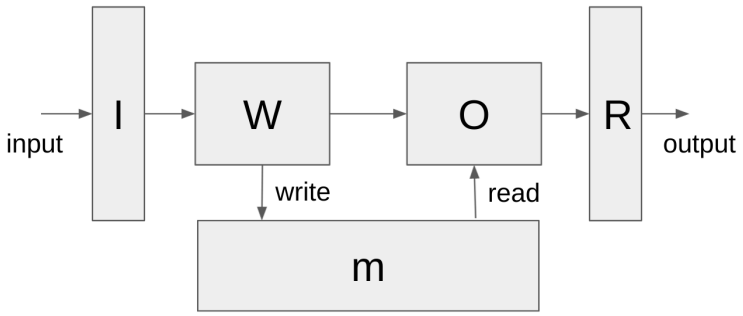


Fig. 1. The architecture of a memory network

I can make use of standard pre-processing such as parsing, coreference, and entity resolution. It could also encode the input into an internal feature representation by converting from text to a sparse or dense feature vector. The simplest form of **W** is to introduce a function H which maps the internal feature representation produced by **I** to an individual memory slot and just updates the memory at H(**I**(x)). More sophisticated variants of **W** could go back and update earlier stored memories (potentially, all memories) based on the new evidence from the current input x. If the input is at the character or word level one could group inputs (i.e., by segmenting them into chunks) and store each chunk in a memory slot. **O** Reads from memory and performs inference to deduce the set of relevant memories needed to perform a good response. **R** would produce the actual wording of the question-answer based on the memories found by **O**. For example, **R** could be an RNN conditioned on the output of **O**. When the components **I**, **W**, **O** and **R** are neural networks, the resulting system is a Memory Neural Network (MemNN).

2.1.5. Transformer

Vaswani et al. [2017] introduces a novel architecture called a Transformer, that uses the attention-mechanism we saw earlier. Like a LSTM-based sequence-to-sequence model, the Transformer is an architecture for transforming one sequence into another one with the help of two parts: an Encoder and a Decoder. But it differs from the previously existing sequence-to-sequence models because it does not imply any Recurrent Networks.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 2, respectively.

The inputs and outputs (target sentences) are first embedded into an d_k-dimensional space. One slight but important part of the model is the positional encoding of the different words. Since the transformer has no recurrent networks that can remember how sequences are fed into a model, we need to inform every word in the sequence of their positions. These positions are represented by vectors and are added to the embedded representation of each word.

The attention function used in transformer can be described by the following equation:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1.17}$$

Given the input matrix X, $Q = W_Q X$ is a matrix that contains the query, $K = W_K X$ are all the keys, and $V = W_V X$ are the values, d_k is the dimension of word embeddings and hidden states. For self-attention, V consists of the same word sequence as Q. However, for the intra-attention that takes into account the encoder and the decoder sequences, V and K are different from the sequence represented by Q.

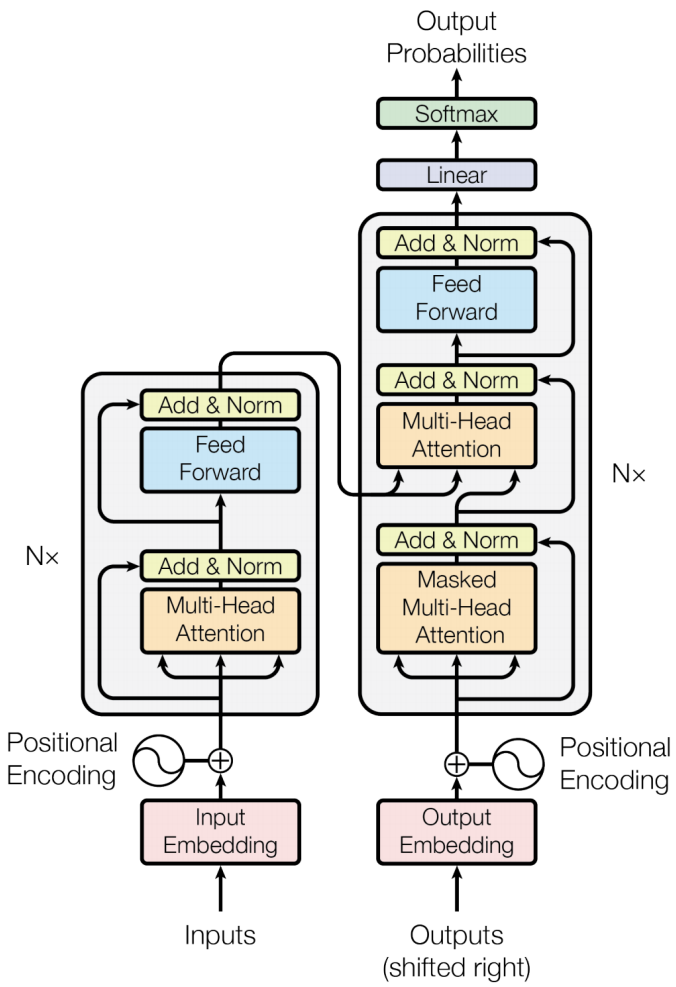


Fig. 2. The Transformer architecture [Vaswani et al., 2017].

One set of (W_Q,W_K,W_V) matrices and the attention function form an attention head. Each layer in a Transformer model has multiple attention heads. While each attention head attends to the tokens that are relevant to each token, with multiple attention heads the model can do this for different definitions of "relevance". The computations for each attention head can be performed in parallel, which allows for fast processing. The outputs for the attention layer are concatenated to pass into the feed-forward neural network layers.

After the multi-headed attention in both the encoder and decoder, we have a pointwise feed-forward layer. This feed-forward network has identical parameters for each position, which can be described as a separate, identical linear transformation of each element from the given sequence. After each multi-headed attention layer and pointwise feed-forward layer, the Transformer has a skip connection and layer normalization. Figure 2 shows the architecture of Transformer.

2.1.6. Graph Neural Networks

Graph neural network is first introduced in Scarselli et al. [2008]. Consider a graph $G = (V, E)$, where $|V| = N$ is the number of nodes in the graph and $|E| = N^e$ is the number of edges. $A \in R^{N \times N}$ is the adjacency matrix. For graph representation learning, we use $\mathbf{h}_v$ as the hidden state of node v.

The graph structures are different from task to task. There are usually two scenarios: structural scenarios and non-structural scenarios. In structural scenarios, the graph structure is explicit in the applications, such as applications on molecules, physical systems, knowledge graphs, and so on. In non-structural scenarios, graphs are implicit so that we have to first build the graph from the task, such as building a fully-connected "word" graph for text or building a scene graph for an image. After we get the graph, the later design process attempts to find an optimal GNN model on this specific graph. In chapter 4, we propose a third scenario: the model needs to induce the graph structure given an inductive bias.

Although graph structure could be very different, they usually can be categorized from three perspectives:

- **Directed/Undirected Graphs.** Edges in directed graphs are all directed from one node to another, which provides more information than undirected graphs. Each edge in undirected graphs can also be regarded as two directed edges.
- **Homogeneous/Heterogeneous Graphs.** Nodes and edges in homogeneous graphs have the same types, while nodes and edges have different types in heterogeneous graphs. For example, social networks are mostly homogeneous graphs, because all nodes belong to the same type – user. But an author-paper network is a heterogeneous graph because it includes two types of nodes: author and paper. Types for nodes play important roles in heterogeneous graphs and should be further considered.
- **Static/Dynamic Graphs.** When input features or the topology of the graph vary with time, the graph is regarded as a dynamic graph. The time information should be carefully considered in dynamic graphs.

The dependency graph can be categorized as a directed, homogeneous and static graph. The direction is usually from a dependent to its head. Since the only type of nodes is words, the graph is homogeneous. And a dependency graph does not change with time.

After identifying the graph type, we need to design the propagation module for a graph neural network. The propagation module is used to propagate information between nodes so that the aggregated information could capture both feature and topological information. In propagation modules, the convolution operator and recurrent operator are usually used to aggregate information from neighbors while the skip connection is used to gather information from previous layers. In Chapter 4, we will propose a multi-channel competitive mechanism to propagate information that is inspired by dependency functions.

2.1.7. Recursive Neural Network

A recursive neural network [Socher et al., 2010] is a kind of deep neural network created by applying the same set of weights recursively over a structured input, to produce a structured prediction over variable-size input structures, or a scalar prediction on it, by traversing a given structure in topological order. Recursive neural networks, sometimes abbreviated as RvNNs, have been successful, for instance, in learning sequence and tree structures in natural language processing, where they are used mainly to learn continuous representations of phrases and sentences based on word embedding. RvNNs have first been introduced to learn distributed representations of structure, such as logical terms.

In the most simple architecture, nodes are combined into parents using a weight matrix that is shared across the whole network, and a non-linearity such as tanh. If c_1 and c_2 are n-dimensional vector representation of nodes, their parent will also be an n-dimensional vector, calculated as

$$p_{1,2} = \tanh\left(W[c_1; c_2]\right) \tag{2.1.18}$$

Where W is a learned $n \times 2n$ weight matrix.

2.2. Tasks for Evaluating Syntactic Inductive Biases

2.2.1. Language Modeling

Language modeling (LM) is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence. Given such a sequence, say of length m, a language model usually assigns a probability $P(w_1, \ldots, w_m)$ to the whole sequence. In this thesis, we use word-level language modeling task for two purposes: 1) an unsupervised training method for other tasks, including unsupervised parsing and syntactic evaluation; 2) a macroscopic evaluation of the model's ability to deal with various linguistic phenomena (e.g. co-occurrence, syntactic structure, verb-subject agreement, etc). We expect that a good syntactic inductive bias should improve the model's performance in modeling natural language.

In recent years, two language modeling tasks have been widely used. The first is **recurrent language modeling**, which requires that the model predicts the next token based on the previous context.

$$p(w_t|w_{<t}) = f(w_1, ..., w_{t-1}) \tag{2.2.1}$$

where $f(\cdot)$ is a neural network that uses word embeddings to make its predictions. In this way, one can easily compute the probability for the entire sentence or document.

$$P(w_1, \ldots, w_m) = p(w_1)p(w_1|w_2)...p(w_m|w_1, ..., w_{m-1}) \tag{2.2.2}$$

To train a recurrent language model, we can use gradient descent methods to optimize the log probability. The performance of a recurrent language model is quantified by perplexity (*ppl*):

$$ppl(w_1, \ldots, w_m) = P(w_1, \ldots, w_m)^{-\frac{1}{m}} = \exp\left(-\frac{1}{m}\sum_{i=1}^{m}\log p(w_i|w_1, ..., w_{i-1})\right) \tag{2.2.3}$$

During the evaluation, we compare the perplexity of different models on the test set.

The other task is **masked language modeling**, which is widely used for pretraining on a large-scale dataset. Under Masked Language Modelling, we typically mask a certain percentage of words in a given sentence and the model is expected to predict those masked words based on other words in that sentence. Such a training scheme makes this model bidirectional in nature because the representation of the masked word is learned based on the words that occur to its left as well as its right.

$$p(w_{masked}|w_{unmasked}) = f(w_{unmasked}) \tag{2.2.4}$$

Similar to the recurrent language model, a masked language model can also be optimized with gradient descent. But it's not straightforward to compute the sentence or document probability from a masked language model. The precision of prediction also attracts less interest. Because the task is primarily used as a pre-training method before fine-tuning on downstream tasks. However, for the same architecture, models with better pretraining loss usually have better fine-tuning performance [Devlin et al., 2018, Liu et al., 2019].

2.2.2. Unsupervised Parsing

Work on unsupervised parsing (also known as grammar induction) attempts to find methods for syntactic parsing that do not require expensive and difficult-to-design expert labeled treebanks for training [Carroll and Charniak, 1992, Klein and Manning, 2002, Smith and Eisner, 2005]. Recent work on latent tree learning offers a new family of approaches to the problem [Yogatama et al., 2016, Maillard et al., 2017, Choi et al., 2018]. Latent tree learning models attempt to induce syntactic structure using the supervision from a non-parsing task such as language modeling and textual entailment. Syntactic inductive biases can be used as tools to solve unsupervised parsing, while unsupervised parsing tasks are also good ways to examine the correlation between human-annotated structures and model-induced structures. There are two major unsupervised parsing tasks: 1) **unsupervised constituency parsing** and 2) **unsupervised dependency parsing**.

Unsupervised constituency parsing compares the predicted constituency trees with human annotations. The performance of an unsupervised constituency parser is often evaluated with the Unlabeled F1 (UF1) score. It calculates the overlap of constituents proposed

by the reference tree and induced tree:

$$R = \frac{\text{\# of correct constituents in induced tree of s}}{\text{\# of constituents in reference tree of s}} \tag{2.2.5}$$

$$P = \frac{\text{\# of correct constituents in induced tree of s}}{\text{\# of constituents in induced tree of s}} \tag{2.2.6}$$

$$\text{UF1} = \frac{2PR}{P+R} \tag{2.2.7}$$

Unsupervised dependency parsing compares the predicted dependency graphs with human annotations. Unlabeled Attachment Score (UAS) is a standard evaluation metric in dependency parsing: the percentage of words that are assigned the correct syntactic head.

$$\text{UAS} = \frac{\text{\# of words with correct heads in induced graph of s}}{\text{\# of words in s}} \tag{2.2.8}$$

2.2.3. Syntactic Evaluation

Syntactic evaluations are designed for evaluating the grammaticality of the predictions of a language model. The method provides hand-crafted minimal pairs of sentences that differ only in the main verb's conjugation, then evaluates whether language models rate each grammatical sentence as more likely than its ungrammatical counterpart. For example, given the two strings "*The keys to the cabinet are on the table*" and "*The keys to the cabinet is on the table*", a model that has learned the proper subject-verb number agreement rules for English should assign a higher probability to the grammatical plural verb in the first sentence than to the ungrammatical singular verb in the second [Linzen et al., 2016]. This method can be used to evaluate many different syntactic phenomenons, including Agreement, Licensing, Garden-Path Effects, Gross Syntactic Expectation, Center Embedding, and Long-Distance Dependencies [Hu et al., 2020].

2.2.4. Synthetic Tasks

Studying the parsing ability of syntactic inductive biases in natural language can be challenging due to the inherent complexities of natural language, like having several valid parses for a single sentence. Another way to check whether a model can understand latent structure is using synthetic data generated from well-formed grammar. There are three major advantages of using synthetic data: 1) the ground-truth structure is known and unique, so the parsing result can be accurately evaluated; 2) the solution of such task usually relies on the latent structure, which provides a strong pressure for the model to learn the grammar; 3) given the well-formed grammar, one can easily generate new data to evaluate the model's generalization ability. Given these three advantages, several synthetic datasets are proposed to evaluate latent tree models. They are usually in the form of logical or mathematical expressions. Nangia and Bowman [2018] proposed ListOps, which is in the style of prefix

arithmetic. It is comprised of deeply nested lists of mathematical operations and a list of single-digit integers. The task requires the model to compute the result of a given equation. Bowman et al. [2014] proposed a logical inference dataset, comprised of propositional logical statements. The task requires the model to reason over two logical statements and compute their relations.

Chapter 3

Constituency Inductive Bias: Ordered Neurons

A constituency grammar is hierarchically structured: smaller units (e.g., phrases) are nested within larger units (e.g., clauses). When a larger constituent ends, all of the smaller constituents that are nested within it must also be closed. While the standard recurrent networks and memory network architectures allows different neurons (memory slots) to track information at different time scales, it does not have an explicit bias towards modeling the hierarchy of constituents. In this chapter, we will introduce a constituency inductive bias and two instantiations of the idea.

We start by reviewing the history of constituency-related models (Section 3.1). In Section 3.2, we propose ordered neurons – a inductive bias that models the hierarchy of constituents through the assigning of an order to neurons. We then proposed two models as instantiations of the inductive bias. In Section 3.3 and Section 3.4, we propose ON-LSTM – a variant of the LSTM model. It use a vector of master input and forget gates to ensures that when a given neuron is updated, all the neurons that follow it in the ordering are also updated. In Section 3.5 and Section 3.6, we propose the Ordered Memory. It use a soft stack mechanism to enforce an order in memory slots. Finally, we summarize recent advances in Section 3.7.

3.1. Previous Approaches

3.1.1. Constituency-Augmented Neural Networks

Theoretically, RNNs and LSTMs can model data produced by context-free grammars and context-sensitive grammars [Gers and Schmidhuber, 2001]. However, recent results suggest that introducing structure information into neural networks is beneficial. Kuncoro et al. [2018] showed that RNNGs [Dyer et al., 2016], which have an explicit bias to model

the syntactic structures, outperform LSTMs on the subject-verb agreement task [Linzen et al., 2016]. In this thesis, we use a more extensive suite of grammatical tests recently provided by Marvin and Linzen [2018] to evaluate the grammar acceptability of our model. Bowman et al. [2014, 2015] also demonstrate that tree-structured models are more effective for downstream tasks whose data was generated by recursive programs. Interestingly, Shi et al. [2018] suggests that while the prescribed grammar tree may not be ideal, some sort of hierarchical structure, perhaps task dependent, might help. Kuncoro et al. [2020] shows that syntactic biases help large scale pre-trained models, like BERT, to achieve better language understanding.

Recursive Neural Networks. There has been prior work leveraging tree structures for natural language tasks in the literature. Socher et al. [2010], Alvarez-Melis and Jaakkola [2016], Zhou et al. [2017], Zhang et al. [2015] use supervised learning on expert-labeled treebanks for predicting parse trees. Socher et al. [2013], Tai et al. [2015] and Zhu et al. [2015], explicitly model the tree-structure using parsing information from an external parser. Later, Bowman et al. [2016] exploited guidance from a supervised parser [Klein and Manning, 2003] in order to train a stack-augmented neural network.

Composition with recursive structures has been shown to work well for certain types of tasks. Pollack [1990] first suggests their use with distributed representations. Later, Socher et al. [2013] shows their effectiveness on sentiment analysis tasks. Recent work has demonstrated that recursive composition of sentences is helpful for systematic generalisation [Bowman et al., 2015, Bahdanau et al., 2018]. Kuncoro et al. [2018] also demonstrate that architectures like RNNG [Dyer et al., 2016] handle syntax-sensitive dependencies better for language-related tasks.

Stack-Augmented Neural Networks. Schützenberger [1963] first showed an equivalence between push-down automata (stack-augmented automatons) and context-free grammars. Knuth [1965] introduced the notion of a shift-reduce parser that uses a stack for a subset of formal languages that can be parsed from left to right. This technique for parsing has been applied to natural language as well: Shieber [1983] applies it to English, using assumptions about how native English speakers parse sentences to remove ambiguous parse candidates. More recently, Maillard et al. [2017] shows that a soft tree could emerge from all possible tree structures through back propagation.

The idea of using neural networks to control a stack is not new. Zeng et al. [1994] uses gradient estimates to learn to manipulate a stack using a neural network. Das et al. [1992] and Mozer and Das [1993] introduced the notion of a *continuous stack* in order for the model to be fully differentiable. Much of the recent work with stack-augmented networks built upon the development of neural attention [Graves, 2013, Bahdanau et al., 2014, Weston et al., 2014]. Graves et al. [2014] proposed methods for reading and writing using a head, along with a "soft" shift mechanism. Apart from using attention mechanisms, Grefenstette

et al. [2015] proposed a neural stack where the push and pop operations are made to be differentiable, which worked well in synthetic datasets. Yogatama et al. [2016] proposes RL-SPINN where the discrete stack operations are directly learned by reinforcement learning.

3.1.2. Constituency Inductive Biases

The task of learning the underlying grammar from data is known as *grammar induction* [Chen, 1995, Cohen et al., 2011]. Early work incorporated syntactic structure in the context of language modeling [Roark, 2001, Charniak, 2001, Chelba and Jelinek, 2000]. More recently, there have been attempts at incorporating some structure for downstream tasks using neural models [Grefenstette et al., 2015, Sun et al., 2017, Joulin and Mikolov, 2015]. Generally, these works augment a main recurrent model with a stack and focus on solving algorithmic tasks. Yogatama et al. [2018] focus on language modeling and syntactic evaluation tasks [Linzen et al., 2016] but they do not show the extent to which the structure learnt by the model align with gold-standard parse trees. Shen et al. [2017] introduced the Parsing-Reading-Predict Networks (PRPN) model, which attempts to perform parsing by solving a language modeling task. The model uses self-attention to compose previous states, where the range of attention is controlled by a learnt "syntactic distance". The authors show that this value corresponds to the depth of the parse tree. However, the added complexity in using the PRPN model makes it unwieldy in practice.

Another possible solution is to develop models with varying time-scales of recurrence as a way of capturing this hierarchy. El Hihi and Bengio [1996], Schmidhuber [1991], Lin et al. [1998] describe models that capture hierarchies at pre-determined time-scales. More recently, Koutnik et al. [2014] proposed Clockwork RNN, which segments the hidden state of a RNN by updating at different time-scales, while Xu et al. [2016] rescales the forget gates at different pre-determined scales. These approaches typically make a strong assumption about the regularity of the hierarchy involved in modelling the data. Chung et al. [2016] proposed a method that, unlike the Clockwork RNN, would learn a multi-scale hierarchical recurrence. However, the model still has a pre-determined depth to the hierarchy, depending on the number of layers. Rippel et al. [2014] proposed to encourage a hierarchy in the representation units by applying "nested" dropout masks: units are not dropped independently at random but whenever a unit is dropped, all the units that follow in the ordering are also dropped. Our work can be seen as a soft and controllable version of the dropout, that we apply a monotonically decreasing mask to the hidden state. Moreover, we propose to condition the update masks on the particular input and apply our overall model to sequential data. Therefore, our model can adapt the structure to the observed data, while both Clockwork RNN and nested dropout impose a predefined hierarchy to hidden representations.

3.2. Ordered Neurons

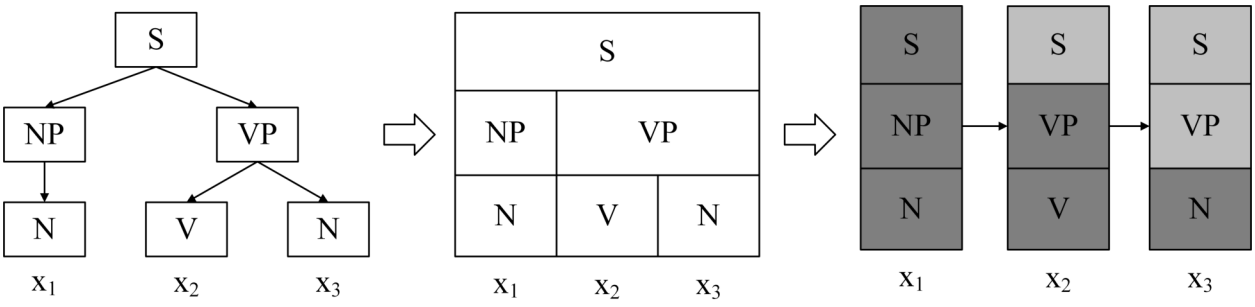


Fig. 3. An instantiation of Ordered Neurons: ON-LSTM A sequence of tokens $S = (x_1, x_2, x_3)$ and its corresponding constituency tree are illustrated in (a). We provide a block view of the tree structure in (b), where both S and VP nodes span more than one time-step. The representation for high-ranking nodes should be relatively consistent across multiple time-steps. (c) Visualization of the update frequency of groups of hidden state neurons. At each time-step, given the input word, dark grey blocks are completely updated while light grey blocks are partially updated. The three groups of neurons have different update frequencies. Topmost groups update less frequently while lower groups are more frequently updated.

Given a sequence of tokens $S = (x_1, \ldots, x_T)$ and its corresponding constituency tree (Figure 3(a)), our goal is to infer the unobserved tree structure while processing the observed sequence, i.e. while computing the hidden state h_t for each time-step t. At each time-step, h_t would ideally contain information about all the nodes on the path between the current leaf node x_t and the root S. In Figure 3(c), we illustrate how h_t would contain information about all the constituents that include the current token x_t even if those are only partially observed. This intuition suggests that each node in the tree can be represented by a set of neurons in the hidden states. However, while the dimensionality of the hidden state is fixed in advance, the length of the path connecting the leaf to the root of the tree may be different across different time-steps and sentences. Therefore, a desiderata for the model is to dynamically reallocate the dimensions of the hidden state to each node.

Given these requirements, we introduce *ordered neurons*, an inductive bias that forces neurons to represent information at different time-scales. In our model, high-ranking neurons contain long-term or global information that will last anywhere from several time-steps to the entire sentence, representing nodes near the root of the tree. Low-ranking neurons encode short-term or local information that only last one or a few time-steps, representing smaller constituents, as shown in Figure 3(b). The differentiation between high-ranking and low-ranking neurons is learnt in a completely data-driven fashion by controlling the update frequency of single neurons: to erase (or update) high-ranking neurons, the model should first

erase (or update) all lower-ranking neurons. In other words, some neurons always update more (or less) frequently than the others, and that order is pre-determined as part of the model architecture.

3.3. ON-LSTM

While natural languages' grammar remains an open question to study, many neural network models already show strong performance on many NLP tasks. Thus, the most reasonable next step is adding the structural inductive bias to an established neural network model to improve its grammar acceptability. In this section, we present ON-LSTM ("*ordered neurons* LSTM") – a LSTM variant augmented with constituency inductive bias. The difference with the LSTM is that we replace the update function for the cell state c_t with a new function that will be explained in the following sections. The forget gates f_t and input gates i_t are used to control the erasing and writing operation on cell states c_t, as before. Since the gates in the LSTM act independently on each neuron, it may be difficult in general to discern a hierarchy of information between the neurons. To this end, we propose to make the gate for each neuron dependent on the others by enforcing the order in which neurons should be updated.

3.3.1. Activation Function: cummax()

To enforce an order to the update frequency, we introduce a new activation function:

$$\hat{g} = \text{cummax}(\ldots) = \text{cumsum}(\text{softmax}(\ldots)) \tag{3.3.1}$$

where cumsum denotes the cumulative sum. We will show that the vector $\hat{g}$ can be seen as the expectation of a binary gate $g = (0,...,0,1,...,1)$. This binary gate splits the cell state into two segments: the 0-segment and the 1-segment. Thus, the model can apply different update rules on the two segments to differentiate long/short-term information. Denote by d a categorical random variable representing the index for the first 1 in g:

$$p(d) = \text{softmax}(\ldots) \tag{3.3.2}$$

The variable d represents the split point between the two segments. We can compute the probability of the k-th value in g being 1 by evaluating the probability of the disjunction of any of the values before the k-th being the split point, that is $d \leq k = (d = 0) \vee (d = 1) \vee \cdots \vee (d = k)$. Since the categories are mutually exclusive, we can do this by computing the cumulative distribution function:

$$p(g_k = 1) = p(d \leq k) = \sum_{i \leq k} p(d = i) \tag{3.3.3}$$

Ideally, g should take the form of a discrete variable. Unfortunately, computing gradients when a discrete variable is included in the computation graph is not trivial [Schulman et al., 2015], so in practice we use a continuous relaxation by computing the quantity $p(d \leq k)$, obtained by taking a cumulative sum of the softmax. As g_k is binary, this is equivalent to computing $\mathbb{E}[g_k]$. Hence, $\hat{g} = \mathbb{E}[g]$.

3.3.2. Structured Gating Mechanism

Based on the cummax() function, we introduce a master forget gate $\tilde{f}_t$ and a master input gate $\tilde{i}_t$:

$$\tilde{f}_t = \text{cummax}(W_{\tilde{f}} x_t + U_{\tilde{f}} h_{t-1} + b_{\tilde{f}}) \tag{3.3.4}$$

$$\tilde{i}_t = 1 - \text{cummax}(W_{\tilde{i}} x_t + U_{\tilde{i}} h_{t-1} + b_{\tilde{i}}) \tag{3.3.5}$$

Following the properties of the cummax() activation, the values in the master forget gate are monotonically increasing from 0 to 1, and those in the master input gate are monotonically decreasing from 1 to 0. These gates serve as high-level control for the update operations of cell states. Using the master gates, we define a new update rule:

$$\omega_t = \tilde{f}_t \circ \tilde{i}_t \tag{3.3.6}$$

$$\hat{f}_t = f_t \circ \omega_t + (\tilde{f}_t - \omega_t) = \tilde{f}_t \circ (f_t \circ \tilde{i}_t + 1 - \tilde{i}_t) \tag{3.3.7}$$

$$\hat{i}_t = i_t \circ \omega_t + (\tilde{i}_t - \omega_t) = \tilde{i}_t \circ (i_t \circ \tilde{f}_t + 1 - \tilde{f}_t) \tag{3.3.8}$$

$$c_t = \hat{f}_t \circ c_{t-1} + \hat{i}_t \circ \hat{c}_t \tag{3.3.9}$$

where f_t and i_t are the original forget and input gate in the LSTM.

In order to explain the intuition behind the new update rule, we assume that the master gates are binary:

- The master forget gate $\tilde{f}_t$ controls the erasing behavior of the model. Suppose $\tilde{f}_t = (0, \ldots, 0, 1, \ldots, 1)$ and the split point is d_t^f. Given the Eq. (3.3.7) and (3.3.9), the information stored in the first d_t^f neurons of the previous cell state c_{t-1} will be completely erased. In a parse tree (e.g. Figure 3(a)), this operation is akin to closing previous constituents. A large number of zeroed neurons, i.e. a large d_t^f, represents the end of a high-level constituent in the parse tree, as most of the information in the state will be discarded. Conversely, a small d_t^f represents the end of a low-level constituent as high-level information is kept for further processing.
- The master input gate $\tilde{i}_t$ is meant to control the writing mechanism of the model. Assume that $\tilde{i}_t = (1, \ldots, 1, 0, \ldots, 0)$ and the split point is d_t^i. a large d_t^i means that the current input x_t contains long-term information that needs to be preserved for several time-steps. Conversely, a small d_t^i means that the current input x_t just provides local information that could be erased by $\tilde{f}_t$ in the next few time-steps.

- The product of the two master gates ω_t represents the overlap of $\tilde{f}_t$ and $\tilde{i}_t$. Whenever an overlap exists ($\exists k, \omega_{tk} > 0$), the corresponding segment of neurons encodes the incomplete constituents that contain some previous words and the current input word x_t. Since these constituents are incomplete, we want to update the information inside the respective blocks. The segment is further controlled by the f_t and i_t in the standard LSTM model to enable more fine-grained operations within blocks. For example, in Figure 3, the word x_3 is nested into the constituents S and VP. At this time-step, the overlap gray blocks would represent these constituents, such that $\tilde{f}_t$ and $\tilde{i}_t$ can decide whether to reset or update each individual neurons in these blocks.

As the master gates only focus on coarse-grained control, modeling them with the same dimensions as the hidden states is computationally expensive and unnecessary. In practice, we set $\tilde{f}_t$ and $\tilde{i}_t$ to be $D_m = \frac{D}{C}$ dimensional vectors, where D is the dimension of hidden state, and C is a chunk size factor. We repeat each dimension C times, before the element-wise multiplication with f_t and i_t. The downsizing significantly reduces the number of extra parameters that we need to add to the LSTM. Therefore, every neuron within each C-sized chunk shares the same master gates.

3.3.3. Syntactic Distance

We first proposed Syntactic distance in Shen et al. [2018b] to quantify the process of splitting sentences into smaller constituents.

Definition 3.3.1. *Let* $\mathbf{T}$ *be a constituency tree for sentence* $(w_1, ..., w_n)$*. The height of the lowest common ancestor for consecutive words* x_i *and* x_{i+1} *is* $\tilde{\tau}_i$*. Syntactic distances* $\mathrm{T} = (\tau_1, ..., \tau_{n-1})$ *are defined as a sequence of* $n-1$ *real scalars that share the same rank as* $(\tilde{\tau}_1, ..., \tilde{\tau}_{n-1})$.

In other words, each syntactic distance d_i is associated with a split point $(i,i+1)$ and specify the relative order in which the sentence will be split into smaller components. Thus, any sequence of $n-1$ real values can unambiguously map to an unlabeled binary constituency tree with n leaves through Algorithm 1 [Shen et al., 2018b]. As Shen et al. [2018c,a], Wang et al. [2019] pointed out, the syntactic distance reflects the information communication between constituents. More concretely, a large syntactic distance τ_i represents that short-term or local information should not be communicated between $(x_{\leq i})$ and $(x_{>i})$. While cooperating with appropriate neural network architectures, we can leverage this feature to build unsupervised constituency parsing models.

In ON-LSTM and other methods developed from Ordered Neurons, the syntactic distance has a more specific meaning. It represent the amount of low-level information that has been erased at each steps. To infer the tree structure of a sentence from a pre-trained model, we initialize the hidden states with the zero vector, then feed the sentence into the model as

Algorithm 1 Distance to binary constituency tree

```
function CONSTITUENT(w, d)
    if d = [] then
        T ⇐ Leaf(w)
    else
        i ⇐ arg max_i(d)
        child_l ⇐ Constituent(w_≤i, d_<i)
        child_r ⇐ Constituent(w_>i, d_>i)
        T ⇐ Node(child_l, child_r)
    end if
    return T
end function
```

done in the language modeling task. At each time step, we compute an estimate of d_t^f:

$$\hat{d}_t^f = \mathbb{E}\left[d_t^f\right] = \sum_{k=1}^{D_m} k p_f(d_t = k) = \sum_{k=1}^{D_m} \sum_{i=1}^{k} p_f(d_t = k) = D_m - \sum_{k=1}^{D_m} \tilde{f}_{tk} \tag{3.3.10}$$

where p_f is the probability distribution over split points associated to the master forget gate and D_m is the size of the hidden state. Given $\hat{d}_t^f$, we can use the top-down greedy parsing algorithm proposed in Shen et al. [2017] for unsupervised constituency parsing. We first sort the $\{\hat{d}_t^f\}$ in decreasing order. For the first $\hat{d}_i^f$ in the sorted sequence, we split the sentence into constituents $((x_{<i}), (x_i, (x_{>i})))$. Then, we recursively repeat this operation for constituents $(x_{<i})$ and $(x_{>i})$, until each constituent contains only one word.

3.4. Natural Language Experiments

In this section, we study the performance of ON-LSTM in natural language settings. We first train and evaluate the model on language modeling task. We then compared the syntactic structures induced by ON-LSTM with human-annotated structures. Finally, We test its grammar acceptability on a synthetic dataset. Results show that the proposed inductive bias can indeed improve model's grammar acceptability while preserve the strong performance on language modeling.

3.4.1. Language Modeling

Word-level language modeling is a macroscopic evaluation of the model's ability to deal with various linguistic phenomena (e.g. co-occurence, syntactic structure, verb-subject agreement, etc). We evaluate our model by measuring perplexity on the Penn TreeBank (PTB) [Marcus et al., 1993, Mikolov, 2012] task.

For fair comparison, we closely follow the model hyper-parameters, regularization and optimization techniques introduced in AWD-LSTM [Merity et al., 2017]. Our model uses a three-layer ON-LSTM model with 1150 units in the hidden layer and an embedding of

size 400. For master gates, the downsize factor $C = 10$. The total number of parameters is 25 millions, which is 1 millions more then the AWD-LSTM. The extra parameters are used for computing the master gates. We manually searched some of the dropout values for ON-LSTM based on the validation performance. The values used for dropout on the word vectors, the output between LSTM layers, the output of the final LSTM layer, and embedding dropout where (0.5, 0.3, 0.45, 0.1) respectively. A weight-dropout of 0.45 was applied to the recurrent weight matrices.

Model	**Parameters**	**Validation**	**Test**
Zaremba et al. [2014] - LSTM (large)	66M	82.2	78.4
Gal and Ghahramani [2016] - Variational LSTM (large, MC)	66M	–	73.4
Kim et al. [2016] - CharCNN	19M	–	78.9
Merity et al. [2016] - Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. [2016] - LSTM	–	–	82.3
Grave et al. [2016] - LSTM + continuous cache pointer	–	–	72.1
Inan et al. [2016] - Variational LSTM (tied) + augmented loss	51M	71.1	68.5
Zilly et al. [2016] - Variational RHN (tied)	23M	67.9	65.4
Zoph and Le [2016] - NAS Cell (tied)	54M	–	62.4
Shen et al. [2017] - PRPN-LM	–	–	62.0
Melis et al. [2017] - 4-layer skip connection LSTM (tied)	24M	60.9	58.3
Merity et al. [2017] - AWD-LSTM - 3-layer LSTM (tied)	24M	60.0	57.3
ON-LSTM - 3-layer (tied)	25M	$\mathbf{58.29 \pm 0.10}$	$\mathbf{56.17 \pm 0.12}$
Yang et al. [2017] - AWD-LSTM-MoS*	22M	56.5	54.4

Table 1. Single model perplexity on validation and test sets for the Penn Treebank language modeling task. Models labelled *tied* use weight tying on the embedding and softmax weights [Inan et al., 2016, Press and Wolf, 2017]. Models labelled * focus on improving the softmax component of RNN language model. Their contribution is orthogonal to ours.

As shown in Table 1, our model performs better than the standard LSTM while sharing the same number of layers, embedding dimensions, and hidden states units. Recall that the master gates only control how information is stored in different neurons. It is interesting to note that we can improve the performance of a strong LSTM model without adding skip connections or a significant increase in the number of parameters.

3.4.2. Unsupervised Constituency Parsing

The unsupervised constituency parsing task compares the latent stree structure induced by the model with those annotated by human experts. Following the experiment settings proposed in Htut et al. [2018], we take our best model for the language modeling task, and test it on WSJ10 dataset and WSJ test set. WSJ10 has 7422 sentences, filtered from the WSJ dataset with the constraint of 10 words or less, after the removal of punctuation and null elements [Klein and Manning, 2002]. The WSJ test set contains 2416 sentences with various lengths. It is worth noting that the WSJ10 test set contains sentences from the

training, validation, and test set of the PTB dataset, while WSJ test uses the same set of sentences as the PTB test set.

Model	Training Data	Vocab Size	Parsing F1 WSJ10 μ (σ)	Parsing F1 WSJ μ (σ)	Depth WSJ	Accuracy on WSJ by Tag ADJP	NP	PP	INTJ
PRPN-UP	AllNLI Train	76k	66.3 (0.8)	38.3 (0.5)	5.8	28.7	65.5	32.7	*0.0*
PRPN-LM	AllNLI Train	76k	52.4 (4.9)	35.0 (5.4)	6.1	37.8	59.7	**61.5**	**100.0**
PRPN-UP	WSJ Train	15.8k	62.2 (3.9)	26.0 (2.3)	5.8	24.8	54.4	17.8	*0.0*
PRPN-LM	WSJ Train	10k	70.5 (0.4)	37.4 (0.3)	5.9	26.2	**63.9**	24.4	*0.0*
ON-LSTM 1st-layer	WSJ Train	10k	35.2 (4.1)	20.0 (2.8)	5.6	38.1	23.8	18.3	**100.0**
ON-LSTM 2nd-layer	WSJ Train	10k	65.1 (1.7)	**47.7 (1.5)**	5.6	**46.2**	61.4	55.4	*0.0*
ON-LSTM 3rd-layer	WSJ Train	10k	54.0 (3.9)	36.6 (3.3)	5.3	44.8	57.5	47.2	*0.0*
CCM	WSJ10 Full	–	71.9	–	–	–	–	–	–
DMV+CCM	WSJ10 Full	–	77.6	–	–	–	–	–	–
UML-DOP	WSJ10 Full	–	**82.9**	–	–	–	–	–	–
Random Trees	–	–	31.7 (0.3)	18.4 (0.1)	5.3	17.4	22.3	16.0	40.4
Balanced Trees	–	–	43.4 (0.0)	24.5 (0.0)	4.6	22.1	*20.2*	*9.3*	55.9
Left Branching	–	–	*19.6 (0.0)*	*9.0 (0.0)*	12.4	–	–	–	–
Right Branching	–	–	56.6 (0.0)	39.8 (0.0)	12.4	–	–	–	–

Table 2. Unlabeled parsing F1 results evaluated on the full WSJ10 and WSJ test set. Our language model has three layers, each of them provides a sequence of $\hat{d}_t^f$. We provide the parsing performance for all layers. Results with RL-SPINN and ST-Gumbel are evaluated on the full WSJ [Williams et al., 2017]. PRPN models are evaluated on the WSJ test set [Htut et al., 2018]. We run the model with 5 different random seeds to calculate the average F1. The *Accuracy* columns represent the fraction of ground truth constituents of a given type that correspond to constituents in the model parses. We use the model with the best F1 score to report ADJP, NP, PP, and INTJ. WSJ10 baselines are from [Klein and Manning, 2002, CCM], [Klein and Manning, 2005, DMV+CCM], and [Bod, 2006, UML-DOP]. As the WSJ10 baselines are trained using POS tags, they are not strictly comparable with the latent tree learning results. Italics mark results that are worse than the random baseline.

The performance is shown in Table 2. The second layer of ON-LSTM achieves state-of-the-art unsupervised constituency parsing results on the WSJ test set, while the first and third layers do not perform as well. One possible interpretation is that the first and last layers may be too focused on capturing local information useful for the language modeling task as they are directly exposed to input tokens and output predictions respectively, thus may not be encouraged to learn the more abstract tree structure. Since the WSJ test set contains sentences of various lengths which are unobserved during training, we find that ON-LSTM provides better generalization and robustness toward longer sentences than previous models. We also see that the ON-LSTM model can provide strong results for phrase detection, including ADJP (adjective phrases), PP (prepositional phrases), and NP (noun phrases). This feature could benefit many downstream tasks, like question answering, named entity recognition, co-reference resolution, etc.

3.4.3. Targeted Syntactic Evaluation

Targeted syntactic evaluation tasks have been proposed in Marvin and Linzen [2018]. It is a collection of tasks that evaluate language models along three different structure-sensitive linguistic phenomena: subject-verb agreement, reflexive anaphora and negative polarity items. Given a large number of minimally different pairs of English sentences, each consisting of a grammatical and an ungrammatical sentence, a language model should assign a higher probability to a grammatical sentence than an ungrammatical one.

Using the released codebase[1] and the same settings proposed in Marvin and Linzen [2018], we train both our ON-LSTM model and a baseline LSTM language model on a 90 million word subset of Wikipedia. Both language models have two layers of 650 units, a batch size of 128, a dropout rate of 0.2, a learning rate of 20.0, and were trained for 40 epochs. The input embeddings have 200 dimensions and the output embeddings have 650 dimesions.

Table 3 shows that the ON-LSTM performs better on the long-term dependency cases, while the baseline LSTM fares better on the short-term ones. This is possibly due to the relatively small number of units in the hidden states, which is insufficient to take into account both long and short-term information. We also notice that the results for NPI test cases have unusually high variance across different hyper-parameters. This result maybe due to the non-syntactic cues discussed in Marvin and Linzen [2018]. Despite this, ON-LSTM actually achieves better perplexity on the validation set.

3.5. Ordered Memory

While modern neural network models achieve surprising performance on many natural language tasks, our later experiment sections will show that they fall short on some formal language tasks. Especially when standard neural network models face a systematic generalization setting, they tend suffer a dramatic performance drop. This exposes the problem that these models fail to capture the internal mechanism of formal languages. In this section, we present *Ordered Memory* – a new model that is developed from ordered neurons to solve formal language tasks.

Ordered Memory is another instantiation of ordered neurons that groups neurons into memory slots and assigns a predetermined order to the memory slots. The model explicitly models constituency structure through memory writing and erasing operations. OM maps the latent syntax into a $T \times N$ memory grid $\tilde{M}$, where T is the length of input sequence and N is the maximum number of memory slots. Figure 4 gives an intuition of what the grid contains. Empty blocks in the figure represent memory slots that can be discarded

[1] `https://github.com/BeckyMarvin/LM_syneval`. We notice that the test set generated from the code is different from the one used in the original paper [Marvin and Linzen, 2018]. Therefore, our results are not strictly comparable with the results in Marvin and Linzen [2018].

	ON-LSTM	LSTM
Short-Term Dependency		
Subject-verb agreement:		
Simple	0.99	**1.00**
In a sentential complement	0.95	**0.98**
Short VP coordination	0.89	**0.92**
In an object relative clause	0.84	**0.88**
In an object relative (no *that*)	0.78	**0.81**
Reflexive anaphora:		
Simple	**0.89**	0.82
In a sentential complement	**0.86**	0.80
Negative polarity items:		
Simple (grammatical vs. intrusive)	0.18	**1.00**
Simple (intrusive vs. ungrammatical)	**0.50**	0.01
Simple (grammatical vs. ungrammatical)	0.07	**0.63**
Long-Term Dependency		
Subject-verb agreement:		
Long VP coordination	**0.74**	**0.74**
Across a prepositional phrase	0.67	**0.68**
Across a subject relative clause	**0.66**	0.60
Across an object relative clause	**0.57**	0.52
Across an object relative (no *that*)	**0.54**	0.51
Reflexive anaphora:		
Across a relative clause	0.57	**0.58**
Negative polarity items:		
Across a relative clause (grammatical vs. intrusive)	0.59	**0.95**
Across a relative clause (intrusive vs. ungrammatical)	**0.20**	0.00
Across a relative clause (grammatical vs. ungrammatical)	**0.11**	0.04

Table 3. Overall accuracy for the ON-LSTM and LSTM on each test case. "Long-term dependency" means that an unrelated phrase (or a clause) exist between the targeted pair of words, while "short-term dependency" means there is no such distraction.

during inference. Ideally, the memory network should generate the t-th column of the grid $\tilde{M}_t$ at time-step t. But generating $\tilde{M}_t$ requires the model to have access about the tree structure which is usually latent. The OM model actively maintains its memory as a stack and processes the input from left to right, with a one-step lookahead in the sequence. This allows the OM model to decide the local structure more accurately, much like a shift-reduce parser [Knuth, 1965].

At a given point t in the input sequence $\boldsymbol{x}$ (the t-th time-step), we have a memory of candidate sub-trees spanning the non-overlapping sub-sequences in $x_1, \ldots, x_{t-1}$, with each sub-tree being represented by one slot in the memory stack. We also maintain a memory stack of sub-trees that contains $x_1, \ldots, x_{t-2}$. We use the input x_t to choose its parent node from our previous candidate sub-trees. The descendant sub-trees of this new sub-tree (if they exist) are removed from the memory stack, and this new sub-tree is then added. We then

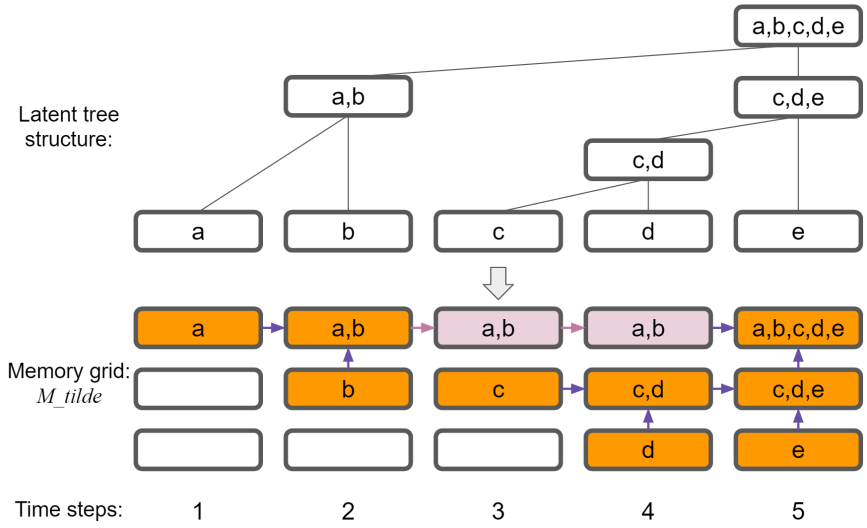


Fig. 4. The grid view of a tree structure. Blue arrows represent composing children into parent. Pink arrows represent copying from previous time-step. Orange slots are memories generated at the current time-step. Pink slots are memories copied from previous time-step.

build the new candidate sub-trees that include x_t using the current input and the memory stack. Figure 5 provides an example of one time-step in OM. In what follows, we describe the OM model in detail. To facilitate a clearer description, a discrete attention scheme is assumed, but only "soft" attention is used in both the training and evaluation of this model.

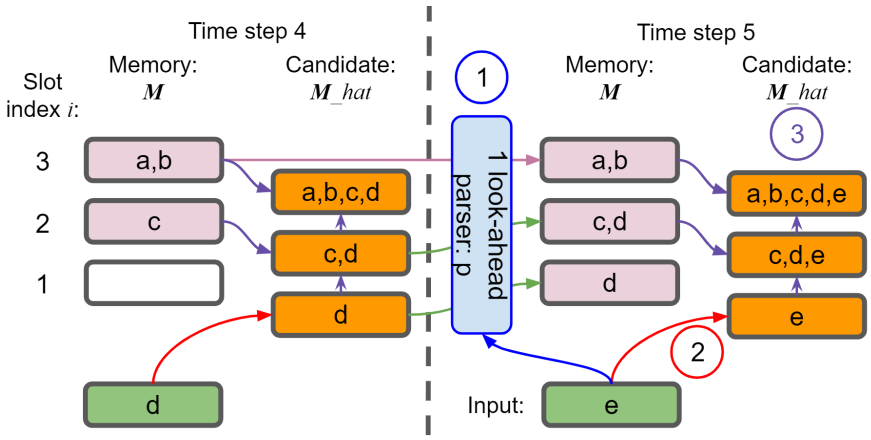


Fig. 5. The transition from time-step 4 to 5. (1) The one-step look-ahead parser combines $\hat{M}_{t-1}$ and M_{t-1} considering on the current input x_t, in this example, the split point of $\hat{M}_{t-1}$ and M_{t-1} is $i = 2$. (2) Current input x_t is written into the lower slot of new candidate memory $\hat{M}_t^{i-1}$. (3) The rest of new candidate memories $\hat{M}_t^{\geq i}$ are generated with bottom-up recurrent composition.

Let D be the dimension of each memory slot and N be the number of memory slots. At time-step t, the model takes four inputs:

- **Memory** M_{t-1}: a matrix of dimension $N \times D$, where each occupied slot is a distributed representation for a node spanning a subsequence in $x_1, .., x_{t-2}$ conditioned on x_{t-1}, i.e. M_{t-1} represents a one-step look-ahead parser stack. It's represented by gray blocks in Figure 5.
- **Candidate memory** $\hat{M}_{t-1}$: a matrix of dimension $N \times D$ contains representations for all possible new nodes at time-step $t-1$. At next time-step t, the model will decide whether or not to write these candidates into memory M_t conditioned on x_t. They are represented by orange blocks in Figure 5. If the model is making correct parsing decisions, then $M_t = \tilde{M}_{t-1}$.
- **Memory mask** $\overrightarrow{\pi}_{t-1}$: $\overrightarrow{\pi}_{t-1} \in \{0,1\}^N$, where each entry indicates whether the respective slot in $\hat{M}_{t-1}$ is occupied by a candidate, e.g., if $\overrightarrow{\pi}_{t-1} = (0,1,1)$, then the occupied slots are $\hat{M - 1}_t^{\geq 2}$. At time-step t, the model can only choose a candidate from masked slots to write into the memory M_t.
- **Input** x_t: a vector of dimension D_{in} that represent the input at time-step t.

The model first transforms x_t to a D dimension vector.

$$\tilde{x}_t = LN(Wx_t + b) \tag{3.5.1}$$

where $LN(\cdot)$ is the layer normalization function [Ba et al., 2016].

To select the candidate representations from $\hat{M}_{t-1}$, the model uses $\tilde{x}_t$ as its query to attend on $\hat{M}_{t-1}$:

$$p_t = \text{Att}(\tilde{x}_t, \hat{M}_{t-1}, \overrightarrow{\pi}_{t-1}) \tag{3.5.2}$$

$$\overrightarrow{\pi}_t^i = \sum_{j \leq i} p_t^j \tag{3.5.3}$$

$$\overleftarrow{\pi}_t^i = \sum_{j \geq i} p_t^j \tag{3.5.4}$$

where $\text{Att}(\cdot)$ is a masked attention function, $\overrightarrow{\pi}_{t-1}$ is the mask, p_t is a distribution over different memory slots in $\hat{M}_{t-1}$, and p_t^j is the probability on the j-th slot. The attention mechanism will be described in section 3.5.1. Intuitively, p_t can be viewed as a pointer to the head of the stack, $\overrightarrow{\pi}_t$ is an indicator value over where the stack exists, and $\overleftarrow{\pi}_t$ is an indicator over where the top of the stack is and where it is non-existent.

To compute M_t, we combine $\hat{M}_{t-1}$ and M_{t-1} through:

$$M_t^i = M_{t-1}^i \cdot (1 - \overleftarrow{\pi})^i + \hat{M}_{t-1}^i \cdot \overleftarrow{\pi}_t^i, \quad \forall i \tag{3.5.5}$$

Suppose p_t points at a memory slot y_t in $\hat{m}$. Then $\overleftarrow{\pi}_t$ will write $\hat{M}_{t-1}^i$ to M_t^i for $i \leq y_t$, and $(1 - \overleftarrow{\pi}_t)$ will write M_{t-1}^i to M_t^i for $i > y_t$. In other words, Eqn. 3.5.5 copies everything from M_{t-1} to the current timestep, up to where p_t is pointing.

Algorithm 2 Ordered Memory algorithm. The attention function Att$(\cdot)$ is defined in section 3.5.1. The recursive cell function cell$(\cdot)$ is defined in section 3.5.2.

function OM$(x_1, ..., x_T)$
  initialize $M_0, \hat{M}_0$
  for $i \leftarrow 1$ to T **do**
    $\tilde{x}_t = LN(Wx_t + b)$
    $p_t = \text{Att}(\tilde{x}_t, \hat{M}_{t-1}, \overrightarrow{\pi}_{t-1})$
    $\overrightarrow{\pi}_t^i = \sum_{j \leq i} p_t^j$
    $\overleftarrow{\pi}_t^i = \sum_{j \geq i} p_t^j$
    $\hat{M}_t^0 = \tilde{x}_t$
    for $i \leftarrow 1$ to N **do**
      $M_t^i = M_{t-1}^i \cdot (1 - \overleftarrow{\pi}_t)^i + \hat{M}_{t-1}^i \cdot \overleftarrow{\pi}_t^i$
      $o_t^i = \mathbf{c}(M_t^i, \hat{M}_t^{i-1})$
      $\hat{M}_t^i = \tilde{x}_t \cdot (1 - \overrightarrow{\pi}_t)^i + o_t^i \cdot \overrightarrow{\pi}_t^i$
    end for
  end for
  return o_T^N
end function

We believe that this is a crucial point that differentiates our model from past stack-augmented models like Yogatama et al. [2016] and Joulin and Mikolov [2015]. Both constructions have the 0-th slot as the top of the stack, and perform a convex combination of each slot in the memory / stack given the action performed. More concretely, a distribution over the actions that is not sharp (e.g. 0.5 for pop) will result in a weighted sum of an un-popped stack and a pop stack, resulting in a blurred memory state. Compounded, this effect can make such models hard to train. In our case, because $(1 - \overleftarrow{\pi}_t)^i$ is non-decreasing with i, its value will accumulate to 1 at or before N. This results in a full copy, guaranteeing that the earlier states are retained. This full retention of earlier states may play a part in the training process, as it is a strategy also used in Gulcehre et al. [2017], where all the memory slots are filled before any erasing or writing takes place.

To compute candidate memories for time-step t, we recurrently update all memory slots with

$$o^i = \mathbf{c}(M_t^i, \hat{M}_t^{i-1}) \tag{3.5.6}$$

$$\hat{M}_t^i = \tilde{x}_t \cdot (1 - \overrightarrow{\pi}_t)^{i+1} + o_t^i \cdot \overrightarrow{\pi}_t^i, \forall i \tag{3.5.7}$$

where $\hat{M}_t^0$ is x_t. The $\mathbf{c}(\cdot)$ function can be seen as a recursive composition function in a recursive neural network [Socher et al., 2013]. We propose a new cell function in section 3.5.2.

The output of time-step t is the last memory slot $\hat{M}_t^N$ of the new candidate memory, which summarizes all the information from $x_1, ..., x_t$ using the induced structure. The pseudo-code for the OM algorithm is shown in Algorithm 2.

3.5.1. Masked Attention

Given the projected input $\tilde{x}_t$ and candidate memory $\hat{M}_{t-1}^i$. We feed every $(\tilde{x}_t, \hat{M}_{t-1}^i)$ pair into a feed-forward network:

$$\alpha_t^i = \frac{\mathbf{w}_2^{Att} \tanh\left(\mathbf{W}_1^{Att} \begin{bmatrix} \hat{M}_{t-1}^i \\ \tilde{x}_t \end{bmatrix} + b_1\right) + b_2}{\sqrt{N}} \tag{3.5.8}$$

$$\beta_t^i = \exp\left(\alpha_t^i - \max_j \alpha_t^j\right) \tag{3.5.9}$$

where $\mathbf{W}_1^{Att}$ is $N \times 2N$ matrix, $\mathbf{w}_2^{Att}$ is N dimension vector, and the output β_t^i is a scalar. The purpose of dividing by $\sqrt{N}$ is to scale down the logits before softmax is applied, a technique similar to the one seen in Vaswani et al. [2017]. We further mask the β_t with the cumulative probability from the previous time-step to prevent the model attending on non-existent parts of the stack:

$$\hat{\beta}_t^i = \beta_t^i \overrightarrow{\pi}_{t-1}^{i+1} \tag{3.5.10}$$

where $\overrightarrow{\pi}_{t-1}^{N+1} = 1$ and $\overrightarrow{\pi}_{\tilde{0}}^{\leq N} = 0$. We can then compute the probability distribution:

$$p_t^i = \frac{\hat{\beta}_t^i}{\sum_j \hat{\beta}_t^j} \tag{3.5.11}$$

This formulation bears similarity to the method used for the multi-pop mechanism seen in Yogatama et al. [2018].

3.5.2. Gated Recursive Cell

Instead of using the recursive cell proposed in TreeLSTM [Tai et al., 2015] and RNTN [Socher et al., 2010], we propose a new gated recursive cell, which is inspired by the feed-forward layer in Transformer [Vaswani et al., 2017]. The inputs M_t^i and $\hat{M}_t^{i-1}$ are concatenated and fed into a fully connect feed-forward network:

$$\begin{bmatrix} v_t^i \\ h_t^i \\ c_t^i \\ u_t^i \end{bmatrix} = \mathbf{W}_2^{Cell} \text{ ReLU}\left(\mathbf{W}_1^{Cell} \begin{bmatrix} \hat{M}_t^{i-1} \\ M_t^i \end{bmatrix} + b_1\right) + b_2 \tag{3.5.12}$$

Like the TreeLSTM, we compute the output with a gated combination of the inputs and u_t^i:

$$o_t^i = LN(\sigma(v_t^i) \odot \hat{M}_t^{i-1} + \sigma(h_t^i) \odot M_t^i + \sigma(c_t^i) \odot u_t^i) \tag{3.5.13}$$

where v_t^i is the vertical gate that controls the input from previous slot, h_t^i is horizontal gate that controls the input from previous time-step, cg_t^i is cell gate that control the u_t^i, o_t^i is the output of cell function, and $LN(\cdot)$ share the same parameters with the one used in the Eqn. 3.5.1.

3.5.3. Relations to ON-LSTM and Shift-reduce Parser

Ordered Memory is implemented following the principles introduced in Ordered Neurons. Our model is related to ON-LSTM in several aspects: 1) The memory slots are similar to the chunks in ON-LSTM, when a higher ranking memory slot is forgotten/updated, all lower ranking memory slots should likewise be forgotten/updated; 2) ON-LSTM uses the monotonically non-decreasing master forget gate to preserve long-term information while erasing short term information, the OM model uses the cumulative probability $\overrightarrow{\pi}_t$; 3) Similarly, the master input gate used by ON-LSTM to control the writing of new information into the memory is replaced with the reversed cumulative probability $\overleftarrow{\pi}_t$ in the OM model.

At the same time, the internal mechanism of OM can be seen as a continuous version of a shift-reduce parser. At time-step t, a shift-reduce parser could perform zero or several reduce steps to combine the heads of stack, then shift the word t into stack. The OM implement the reduce step with Gated Recursive Cell. It combines $\hat{M}_t^{i-1}$, the output of previous reduce step, and M_t^i, the next element in stack, into $\hat{M}_t^i$, the representation for new sub-tree. The number of reduce steps is modeled with the attention mechanism. The probability distribution p_t models the position of the head of the stack after all necessary reduce operations are performed. The shift operations is implemented as copying the current input word x_t into memory.

The upshot of drawing connections between our model and the shift-reduce parser is interpretability: We can approximately infer the computation graph constructed by our model with Algorithm 3. The algorithm can be used for the latent tree induction tasks in Williams et al. [2018].

3.6. Formal Language Experiments

Formal languages are artificial languages, constructed from a well-defined grammar. Many formal language tasks requires the model to have a comprehensive understanding about the grammar. It's also straightforward to produce special training and test set, so that the model need to transfer the learned syntax and operators to out-of-distribution data points.

Algorithm 3 Shift-reduce parsing algorithm for generate parsing tree from Ordered Memory model. Here we greedily choose the argmax(p_t) as the head of stack for each slot.

```
function CONSTITUENT((w_1, p_1), ..., (w_T, p_T))
    queue = [w_2,...,w_T]
    stack = [w_1]
    h = argmax(p_1) − 1
    for i ← 2 to T do
        y_i = argmax(p_i)
        d = y_i − h
        if d > 0 then
            for j ← 1 to d do
                if len(stack) < 2 then
                    Break
                end if
                e_1 = stack.pop()
                e_2 = stack.pop()
                stack.push(node(e_1,e_2))
            end for
        end if
        stack.push(queue.popleft())
        h = y_i − 1
    end for
    while len(stack) > 2 do
        e_1 = stack.pop()
        e_2 = stack.pop()
        stack.push(node(e_1,e_2))
    end while
    return T
end function
```

In this section, we will focus on test OM performance on two tasks: Logical Inference [Bowman et al., 2014] and ListOps [Nangia and Bowman, 2018]. Experiment results show that OM outperformance all baseline models, including ON-LSTM. On out-of-distribution test sets, OM also demonstrates a symbolic-like systematical generalization ability.

3.6.1. Logical Inference

The logical inference task described in Bowman et al. [2015] has a vocabulary of six words and three logical operations, `or`, `and`, `not`. The task is to classify the relationship of two logical clauses into seven mutually exclusive categories. We use a multi-layer perceptron (MLP) with $(h_1, h_2, h_1 \circ h_2, |h_1 - h_2|)$ as input, where h_1 and h_2 are the $\hat{M}_T^N$ of their respective input sequences. We compare our model with LSTM, RRNet [Jacob et al., 2018], Tranformer [Vaswani et al., 2017], Universal Transformer [Dehghani et al., 2018], TreeLSTM [Tai et al., 2015], TreeRNN [Bowman et al., 2015], and TreeCell. We used the same hidden state size for

Table 4. Partitions of the Logical Inference task from Bowman et al. [2014]. Each partitions include a training set filtered out all data points that match the rule indicated in **Excluded**, and a test set formed by matched data points.

Part.	Excluded	Training set size	Test set example
A	`* ( and (not a) ) *`	128,969	`f (and (not a))`
B	`* ( and (not *) ) *`	87,948	`c (and (not (a (or b))))`
C	`* ( {and,or} (not *) ) *`	51,896	`a (or (e (and c)))`
Full		135,529	

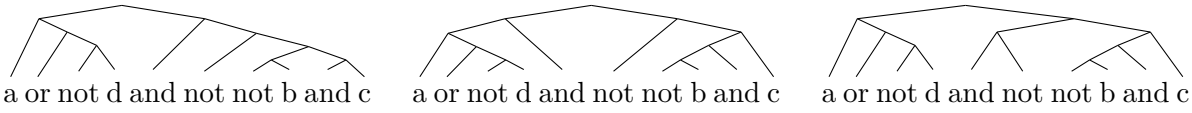


Fig. 6. Variations in induced parse trees under different runs of the logical inference experiment. The left most tree is the ground truth and one of induced structures. We have removed the parentheses in the original sequence for this visualization. It is interesting to note that the different structures induced by our model are all valid computation graphs to produce the correct results.

our model and baselines for proper comparison. The model is trained on sequences containing up to 6 operations and tested on sequences with higher number (7-12) of operations.

The Transformer models were implemented by modifying the code from the Annotated Transformer[2]. The number of Transformer layers are the same as the number of slots in OM. Unfortunately, we were not able to successfully train a Transformer model on the task, resulting in a model that only learns the marginal over the labels. Tran et al. [2018] achieves similar results, suggesting this could be a problem intrinsic to self-attention mechanisms for this task.

Length Generalization Tests. The TreeRNN model represents the best results achievable if the structure of the tree is known. The TreeCell experiment was performed as a control to isolate the performance of using the $\mathbf{c}(\cdot)$ composition function versus using both using $\mathbf{c}(\cdot)$ and learning the composition with OM. The performance of our model degrades only marginally with increasing number of operations in the test set, suggesting generalization on these longer sequences never seen during training.

Parsing results. OM achieve an unsupervised parsing performance 84.3 ± 14.4. There is a variability in parsing performance over several runs under different random seeds, but the model's ability to generalize to longer sequences remains fairly constant. The model learns a slightly different method of composition for consecutive operations. Perhaps predictably, these are variations that do not affect the logical composition of the subtrees. The source

[2] `http://nlp.seas.harvard.edu/2018/04/03/attention.html`

Table 5. Test accuracy of the models, trained on operation lengths of ≤ 6, with their out-of-distribution results shown here (lengths 7-12). We ran 5 different runs of our models, giving the error bounds in the last row. The F_1 score is the parsing score with respect to the ground truth tree structure. The TreeCell is a recursive neural network based on the Gated Recursive Cell function proposed in section 3.5.2. For the Transformer and Universal Transformer, we follow the entailment architecture introduced in Radford et al. [2018]. The model takes `<start> sentence1 <delim> sentence2 <extract>` as input, then use the vector representation for `<extract>` position at last layer for classification. *The results for RRNet were taken from Jacob et al. [2018].

Model	**Number of Operations**						**Sys. Gen.**		
	7	8	9	10	11	12	A	B	C
Sequential sentence representation									
LSTM	88	84	80	78	71	69	84	60	59
RRNet*	84	81	78	74	72	71	–	–	–
Inter sentence attention									
Transformer	51	52	51	51	51	48	53	51	51
Universal Transformer	51	52	51	51	51	48	53	51	51
Our models									
ON-LSTM	91	87	85	81	78	75	70	63	60
OM	98 ± 0.0	97 ± 0.4	96 ± 0.5	94 ± 0.8	93 ± 0.5	92 ± 1.1	94	91	81
Recursive NN + ground-truth structure									
TreeLSTM	94	92	92	88	87	86	91	84	76
TreeCell	98	96	96	95	93	92	95	95	90
TreeRNN	98	98	97	96	95	96	94	92	86

of different parsing results can be seen in Figure 6. The results suggest that these latent structures are still valid computation graphs for the task, in spite of the variations.

Systematic Generalization Tests. Inspired by Loula et al. [2018], we created three splits of the original logical inference dataset with increasing levels of difficulty. Each consecutive split removes a superset of the previously excluded clauses, creating a harder generalization task. Each model is then trained on the ablated training set, and tested on examples unseen in the training data. As a result, the different test splits have different numbers of data points. Table 4 contains the details of the individual partitions.

The results are shown in the right section of Table 5 under Sys. Gen. Each column labeled A, B, and C are the model's aggregated accuracies over the unseen operation lengths. As with the length generalization tests, the models with the known tree structure performs the best on unseen structures, while sequential models degrade quickly as the tests get harder. Our model greatly outperforms all the other sequential models, performing slightly below the results of TreeRNN and TreeCell on different partitions.

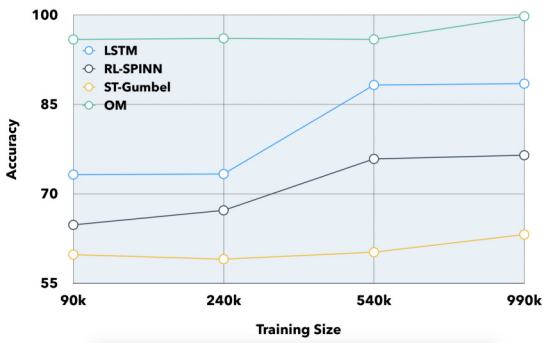
Combined with the parsing results, and our model's performance on these generalization tests, we believe this is strong evidence that the model has both (i) learned to exploit

symmetries in the structure of the data by learning a good $\mathbf{c}(\cdot)$ function, and (ii) learned where and how to apply said function by operating its stack memory.

3.6.2. ListOps

Model	Accuracy	F_1
Baselines		
LSTM*	71.5±1.5	–
RL-SPINN*	60.7±2.6	71.1
Gumbel Tree-LSTM*	57.6±2.9	57.3
Transformer	57.4±0.4	–
Universal Transformer	71.5±7.8	–
Havrylov et al. [2019]	99.2±0.5	–
OM	**99.97±0.014**	**100**

(a)



(b)

Fig. 7. (a) shows the accuracy of different models on the ListOps dataset. All models have 128 dimensions. Results for models with * are taken from Nangia and Bowman [2018]. (b) shows our model accuracy on the ListOps task when varying the the size of the training set.

Nangia and Bowman [2018] build a dataset with nested summary operations on lists of single digit integers. The sequences comprise of the operators `MAX`, `MIN`, `MED`, and `SUM_MOD`. The output is also an integer in $[0, 9]$ As an example, the input: `[MAX 2 9 [MIN 4 7 ] 0 ]` has the solution `9`. As the task is formulated to be easily solved with a correct parsing strategy, the task provides an excellent test-bed to diagnose models that perform tree induction. The authors binarize the structure by choosing the subtree corresponding to each list to be left-branching: the model would first take into account the operator, and then proceed to compute the summary statistic within the list. A right-branching parse would require the entire list to be maintained in the model's hidden state.

OM achieves 99.9% accuracy, and an F_1 score of 100% on the model's induced parse tree (See Table 7a). This result is consistent across 3 different runs of the same experiment. In Nangia and Bowman [2018], the authors perform an experiment to verify the effect of training set size on the latent tree models. As the latent tree models (RL-SPINN and ST-Gumbel) need to parse the input successfully to perform well on the task, the better performance of the LSTM than those models indicate that the size of the dataset does not affect the ability to learn to parse much for those models. Our model seems to be more data efficient and solves the task even when only training on a subset of 90k examples (Fig. 7b).

3.7. Recent Advances

After ON-LSTM was first presented in 2018, many new developments on constituency inductive bias and unsupervised constituency parsing have been introduced. In URNNG [Kim et al., 2019b], amortized variational inference was applied between a recurrent neural network grammar (RNNG) [Dyer et al., 2016] decoder and a tree structure inference network, which encourages the decoder to generate reasonable tree structures. DIORA [Drozdov et al., 2019] proposed using inside-outside dynamic programming to compose latent representations from all possible binary trees. The representations of inside and outside passes from the same sentences are optimized to be close to each other. The compound PCFG [Kim et al., 2019a] achieves grammar induction by maximizing the marginal likelihood of the sentences which are generated by a probabilistic context-free grammar (PCFG). Neural L-PCFG [Zhu et al., 2020] demonstrated that PCFG can benefit from modeling lexical dependencies. The Neural L-PCFG induces both constituents and dependencies within a single model. Tree Transformer [Wang et al., 2019] share a similar inductive bias as ordered neurons. It adds extra locality constraints to the Transformer encoder's self-attention to encourage the attention heads to follow a tree structure such that each token can only attend on nearby neighbors in lower layers and gradually extend the attention field to further tokens when climbing to higher layers.

Chapter 4

Dependency Inductive Bias: Unsupervised Dependency Graph Network

In this chapter, we focus on the other facet of syntax: Dependency grammar. Dependency is the notion that linguistic units, e.g. words, are connected to each other by directed links. The natural of dependency graph makes it compatible with modern graph neural networks. However the actually graph structure is latent, which means the ground-truth structure is usually unavailable to model. Hence, we propose a dependency inductive bias to induce the structure, and a dependency graph network to model the information propagation between words.

We start from reviewing related works which either augment neural networks with dependency graph or induce the dependency graph from raw data (Section 4.1). In Section 4.2, we introduce Unsupervised Dependency Graph Network (UDGN) that includes a Dependency Graph Network and a parser. The Dependency Graph Network (DGN) uses different channels and a competitive mechanism to model information propagation on different types of dependency edges. Alongside the DGN, an independent parsing module provides a soft and differentiable dependency mask to constrain the information propagation. In Section 4.3, we train UDGN with a Masked Language Model (MLM) objective. We find that the model learns to induce dependency trees. It achieves strong performance on MLM and state-of-the-art results in unsupervised dependency parsing on the Wall Street Journal treebank. We also perform analysis on the mechanisms that give rise to this behavior and evaluate the model's potential for finetuning on a downstream task.

4.1. Previous Approaches

4.1.1. Dependency-Augmented Models

In many Transformer-based models, attention masks are often used to limit the input tokens that a particular timestep can attend over. This attention mask can be viewed as an adjacency matrix over a graph whose nodes are the input tokens. From this perspective, Transformers are a form of Graph Convolution network (GCN; [Kipf and Welling, 2016]) — specifically, a Graph Attention Network (GAT; [Veličković et al., 2017]), as it attends over the features of its neighbors. Several works have made this connection, and integrated dependency structures into transformers [Ahmad et al., 2020, Wang et al., 2019, Tang et al., 2020]. Results from Omote et al. [2019] and Deguchi et al. [2019] suggest that embedding these structures can improve translation models.

However, these dependency parses may not always be present to be used as input to the model. Strubell et al. [2018] trains the self-attention to attend the syntactic governor (head) of a particular token, resulting in a model that does not require dependency structure as input during inference time. We take a further step in our work and attempt to learn these structures in an unsupervised fashion from the MLM objective.

4.1.2. Unsupervised Dependency Parsing

Previous works on unsupervised dependency parsing are primarily based on the dependency model with valence (DMV) [Klein and Manning, 2004] and its extension [Daumé III, 2009, Gillenwater et al., 2010]. To effectively learn the DMV model for better parsing accuracy, a variety of inductive biases and handcrafted features, such as correlations between parameters of grammar rules involving different part-of-speech (POS) tags, have been proposed to incorporate prior information into learning. The most recent progress is the neural DMV model [Jiang et al., 2016], which uses a neural network model to predict the grammar rule probabilities based on the distributed representation of POS tags. However, most existing unsupervised dependency parsing algorithms require the gold POS tags to ge provided as inputs. These gold POS tags are labeled by humans and can be potentially difficult (or prohibitively expensive) to obtain for large corpora. Spitkovsky et al. [2011] proposed to overcome this problem with unsupervised word clustering that can dynamically assign tags to each word considering its context. He et al. [2018] overcame the problem by combining DMV model with invertible neural network to jointly model discrete syntactic structure and continuous word representations.

Dependency Model with Valence (DMV; [Klein and Manning, 2004]) is the basis of several unsupervised dependency parsing methods [Daumé III, 2009, Gillenwater et al., 2010]. Jiang et al. [2016] updates the method using neural networks to predict grammar rule probabilities.

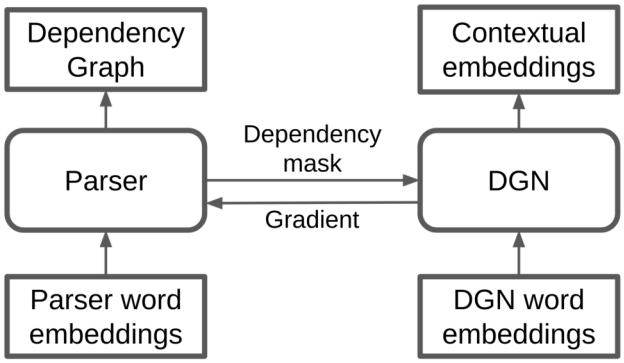


Fig. 8. The architecture of Unsupervised Dependency Graph Network (UDGN). The model includes a parser and a Dependency Graph Network (DGN). Given an input sentence, the parser can predict the dependency relation between tokens and generate a soft mask to approximate the undirected dependency graph. The DGN takes the sentence and mask as input, and output contextual word embeddings. Since the mask is soft, the gradient can be backpropagated from the DGN into the parser. Thus UDGN can induce grammar while training on downstream tasks.

These methods require additional Part-of-Speech (POS) information. Spitkovsky et al. [2011] tackled the issue by performing clustering to assign tags to each word by considering its context. In our proposal, the parser predicts pseudo-probabilities for POS tags for each word. These probabilities are then used for predicting the head word.

4.2. Unsupervised Dependency Graph Network

In this section, we propose the Unsupervised Dependency Graph Networks (UDGN). The UDGN includes two components:

- A parser, which computes the dependency head probability distribution p_i for each word w_i in the input sentence, and then converts it to a matrix of edge probability m_{ij} that approximates an undirected dependency graph;
- A multilayer Dependency Graph Network (DGN) that propagates information between words to compute a contextualized embedding h_i for each word w_i. It use m_{ij} to control information propagation between word pair (w_i, w_j).

As shown in Figure 8, the parser computes a dependency head distribution for each token and then converts it to a soft dependency mask m_{ij}. The DGN takes m_{ij} and the sentence as input and uses a competitive mechanism to propagate information between tokens.

While training with the masked language modeling objective, the gradient can flow through the DGN to the parser network through its dependence on m_{ij}. As a result, UDGN can induce a dependency grammar while solely relying on the masked language modeling objective.

To better model and induce dependency relations, we propose three key components for the DGN:

- A competitive mechanism is proposed to control information propagation between words. The competitive mechanism controls several channels. A channel is a function that models a specific type of information propagation. Given a word pair (w_i,w_j), channels will compete to get more weight to propagate information from word w_j to w_i. The mechanism is inspired by syntactic functions in dependency graphs.
- A gated non-linear network is designed to parameterize each channel. The gating mechanism allows each word to filter information extracted from other tokens. And the non-linear function can increase the capacity of the channel.
- Relative position biases are proposed to model the sequential order of words. The bias allows some channels to focus on extracting information from positions before the current position, while other channels focus on future positions.

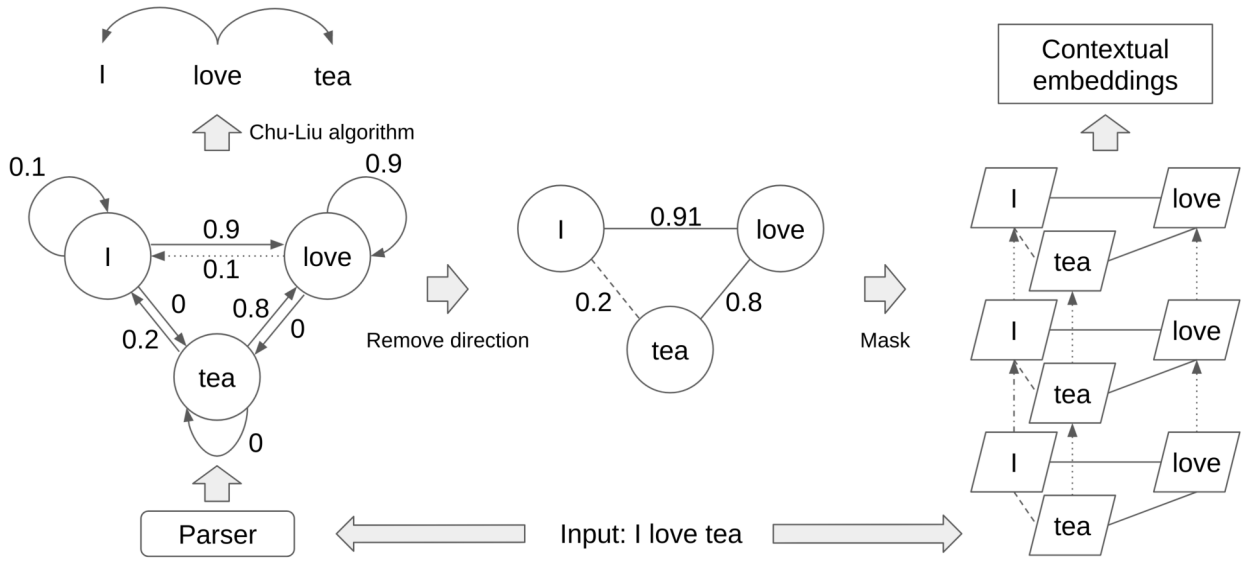


Fig. 9. Details of the UDGN. Given the input sentence, the parser (left) produces a dependency head distribution for each token. These distributions form a distribution matrix p_{ij}. During inference, the Chu-Liu algorithm generates the most likely dependency graph given p_{ij}. While training, however, we remove the direction of dependency in p_{ij} and obtain an undirected dependency mask m_{ij} (middle). m_{ij} is symmetric and with zeroes along the diagonal. The DGN (right) takes m_{ij} and the sentence as input and uses a competitive mechanism to propagate information between tokens. Inside each layer of the DGN, every node (token) will extract information from all other nodes. m_{ij} controls the amount of information being propagated between nodes. If m_{ij} is small then less information will be communicated between x_i and x_j, and vice versa. After several layers, DGN outputs the contextual embedding for each token. These embeddings can be used either to predict missing tokens or as features for downstream tasks.

4.2.1. Head Selective Parser

We use a simplified version of the dependency neural selection parser [Zhang et al., 2016] that only predicts unlabelled dependency relations. The parser takes the sentence $\mathbf{s} = w_1 w_2 ... w_T$ as input, and, for each token w_i, it produces a distribution p_i over all tokens in the sentence, resulting in a $T \times T$ weight matrix.

The parser first maps the sequence of tokens $w_1 w_2 ... w_T$ into a sequence of embeddings $[\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T]$. The standard word embedding method usually maps the input token w_i to a unique vector $\mathbf{e}_{w_i}$. Under this setting, we found that the unsupervised parsing performance becomes increasingly unstable with increasing vocabulary size. We hypothesize that the randomly initialized embedding for low-frequency tokens confuse the parser and cause an extra optimization problem. On the other hand, low-frequency tokens usually share similar grammatical roles. So we propose an extra embedding component $\mathbf{e}^{\text{tag}}_k$ that is shared across tokens with similar grammatical roles:

$$\mathbf{x}_i = \mathbf{e}_{w_i} + \sum_k p^{\text{tag}}_{w_i k} \mathbf{e}^{\text{tag}}_k \tag{4.2.1}$$

$$p^{\text{tag}}_{w_i k} = \text{softmax}(\tau_{w_i k}) \tag{4.2.2}$$

where e_{w_i} is the original word embedding, $p^{\text{tag}}_{w_i k}$ is a probability distribution that associate the w_i with different $\mathbf{e}^{\text{tag}}_k$.

Then the word embeddings are fed into a stack of a Bidirectional LSTM:

$$\mathbf{h}_i = \text{BiLSTM}(\mathbf{x}_i) \tag{4.2.3}$$

$\mathbf{h}_i$ is the output of the BiLSTM at i-th timestep. Linear transforms are applied to the output of the BiLSTM to extract head and dependent information.

$$\mathbf{h}^{\text{H}}_i = \mathbf{W}_{\text{H}} \mathbf{h}_i + \mathbf{b}_{\text{H}} \tag{4.2.4}$$

$$\mathbf{h}^{\text{D}}_i = \mathbf{W}_{\text{D}} \mathbf{h}_i + \mathbf{b}_{\text{D}} \tag{4.2.5}$$

To map the head and dependents, we use bilinear attention:

$$e_{ij} = \frac{\mathbf{h}^{\text{D}}_i \mathbf{h}^{\text{H}}_j}{\sqrt{D}} \tag{4.2.6}$$

$$p_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})} \tag{4.2.7}$$

where p_{ij} is the probability that w_i depends on w_j, D is the dimension of hidden states. To extract the most likely directed dependency graph from the matrix p_{ij}, one can use the Chu-Liu/Edmonds' algorithm Chu and Liu [1965].

Conceptually, this bears a lot of similarity to Dependency Neural Selection (DENSE; Zhang et al. [2016]). In both cases, dependency parsing is reformulated as head selection,

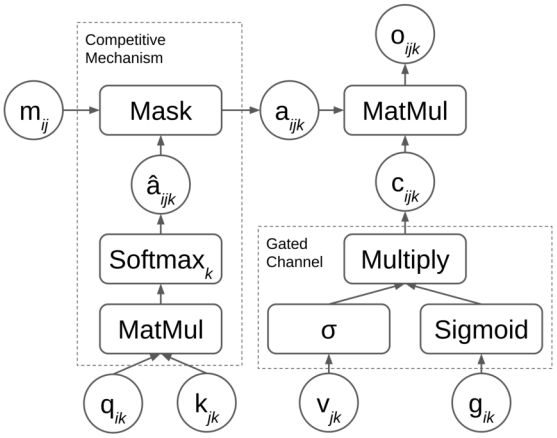
without ensuring a tree structure. During inference for parsing, the same logits can be used in a spanning tree algorithm to retrieve a valid tree for the sentence.

4.2.2. Dependency Mask

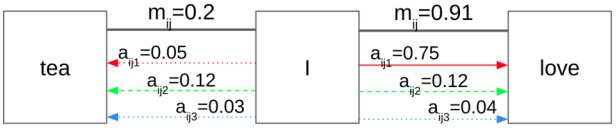
Given the dependency probabilities, Structformer [Shen et al., 2020] uses a weighted sum of matrix p and $p^\top$ to produce a mask for self-attention layers in the transformer. We found that simply using the adjacency matrix of the undirected dependency graph provides better parsing results and perplexities. However, simply using the sum of the matrix and its transpose to create a symmetric weight matrix does not ensure that the attention mask has values < 1. When $p_{ij=1}$ and $p_{ji} = 1$, for instance, the mask violates the constraints of a dependency mask. Thus, we treat p_{ij} and p_{ji} as parameters for independent Bernoulli variables, and we compute the probability that either w_i depends on w_j **or** w_j depends on w_i.

$$\begin{aligned} m_{ij} &= p(i \to j \quad \textbf{or} \quad j \to i) \\ &= p_{ij} + p_{ji} - p_{ij} \times p_{ji} \end{aligned} \tag{4.2.8}$$

4.2.3. Dependency Graph Network



(a) Competitive Mechanism and Gated Channel



(b) A example of the competitive mechanism.

Fig. 10. For a given pair of nodes (i, j), the competitive mechanism takes $\mathbf{q}_{i\cdot}, \mathbf{k}_{j\cdot}$ as input, output a probability distribution $\hat{a}_{ij}$ across different channels. This allows the model to select a channel for the information propagation from j to i. Then the probability $\hat{a}_{ijk}$ is multiplied by dependency mask m_{ij} to get a_{ijk}. The mask m_{ij} functions as a macro gate to control the amount of information propagate between the node pair. $\hat{a}_{ijk}$ is the micro gate that controls the amount of information propagate from j to i through k-th channel. Each channel takes $\mathbf{v}_{jk}, \mathbf{g}_{ik}$ as inputs and outputs respectively to represent the information that is propagated via the k-th channel. $\mathbf{g}_{ik}$ allows the receiving node i to filter the information.

To better use the dependency information, we propose a new Dependency Graph Network (DGN) with Competitive Mechanism. One DGN layer includes several gated channels and a competitive mechanism. The gated channels can process and propagate information from one node to another. Different channels can learn to process and propagate different types of information. The competitive mechanism is designed to select the correct channel to propagate information between a specific pair of nodes.

We take inspiration from the linguistic theory that dependencies are associated with different syntactic functions. These functions can appear as labels, e.g. ATTR (attribute), COMP-P (complement of preposition), and COMP-TO (complement of to). However, DGN learns functions from training tasks, which in our experiments is the masked language model objective. Since these objectives tend to be statistical in nature, these functions may not be correlated with ground truth labels given by human experts.

Inside each layer, the input vector h_i^{l-1} is first projected into N groups of vectors, where N is the number of channels. Each group contains four different vectors, namely, query $\mathbf{q}$, key $\mathbf{k}$, and gate $\mathbf{g}$:

$$\begin{bmatrix} \mathbf{q}_{ik} \\ \mathbf{k}_{ik} \\ \mathbf{v}_{ik} \\ \mathbf{g}_{ik} \end{bmatrix} = \mathbf{W}_{\text{channel}_k} \mathbf{h}_i^{l-1} + \mathbf{b}_{\text{channel}_k} \tag{4.2.9}$$

Competitive Mechanism. Lamb et al. [2021] proposed the idea of using a competition method to encourage channels to specialize over training iterations to achieve independence. In this work, we view these gated channels as mechanisms. Their function is to propagate information from one node to another. To satisfy the independence requirement, a competitive mechanism is designed to assign a channel to each pair of nodes (i, j). However discrete assignment is hard to optimize, we replace it with a soft relaxation:

$$e_{ijk} = \frac{\mathbf{q}_{ik} \mathbf{k}_{jk}}{\sqrt{D}} \tag{4.2.10}$$

$$\hat{a}_{ijk} = \text{softmax}_k(e_{ijk}) \tag{4.2.11}$$

where $\hat{a}_{ijk}$ is the probability that the k-th channel is assigned to propagate information from the j-th token to the i-th token. To obtain the actual channel weights, we multiply the probability of edge existence with the probability of choosing a specific attention head:

$$a_{ijk} = \hat{a}_{ijk} \times m_{ij} \tag{4.2.12}$$

where a_{ijk} is the attention weight from the i-th token to the j-th token for k-th attention head.

Gated Channel. To model the information propagation from node j to node i, we proposed a gated channel:

$$\mathbf{c}_{ijk} = \sigma(\mathbf{v}_{jk}) \odot \text{sigmoid}(\mathbf{g}_{ik}) \tag{4.2.13}$$

where σ is a non-linear activation function, and gates sigmoid($\mathbf{g}$) allows the i-th token to filter the extracted information. We also found that the gate effectively improves the model's ability to induce latent dependency structures that are coherent to human-annotated trees. The activation function can be chosen from a wide variety of functions, including the identity function, tanh, ReLU, and ELU, etc. According to our experiments, we found that tanh function provides the best overall performance.

At the end, a matrix multiplication is used to aggregate information from different positions.

$$\mathbf{o}_{ik} = \sum_j a_{ijk} \mathbf{c}_{ijk} \tag{4.2.14}$$

Then, the output $\mathbf{o}$ from different channels are concatenated, and then projected back to the hidden state space with a linear layer.

$$\mathbf{h}_i^l = \mathbf{h}_i^{l-1} + \mathbf{W}_o \begin{bmatrix} \mathbf{o}_{i1} \\ \vdots \\ \mathbf{o}_{in} \end{bmatrix} + \mathbf{b}_o \tag{4.2.15}$$

where $\mathbf{h}_i^l$ is the output of the l-th gated self attention layers. The shared hidden state space can be seen as the shared global workspace Goyal et al. [2021] for different independent mechanisms.

4.2.4. Relative Position Bias

Transformer models use positional encoding to represent the absolute position for each token. In DGN, we only model whether the token is before or after the current token. The motivating intuition is the association of different channels with different directions. In equation 4.2.11, we can introduce a relative position bias:

$$\hat{a}_{ijk} = \text{softmax}_k(e_{ijk} + b_k^{lr}) \tag{4.2.16}$$

$$b_k^{lr} = \begin{cases} b_k^l, & i > j \\ b_k^r, & i < j \end{cases} \tag{4.2.17}$$

where b_k^l and b_k^r are trainable parameters. The relative position bias allows the attention head k to prioritize forward or backward directions. A mere forward and backward differentiation may seem weak compared to other parameterizations of positional encoding Vaswani et al. [2017], Shaw et al. [2018], but in conjunction with the dependency constraints, this method

is a more effective way to model the relative position in a tree structure. Compared to positional encoding, this relative position bias achieves stronger masked language modeling and parsing performance.

4.3. Experiments

In the experiment section, we first train the UDGN on the masked language modeling task, then evaluate it on masked language modeling and unsupervised parsing. Our experimental results show that UDGN can effectively induce the latent dependency graph from raw corpus, and achieve competitive performance on language modeling tasks. Furthermore, all three key components of DGN play important roles in both grammar induction and language modeling. We also finetuned the pretrained UDGN on Semantic Textual Similarity (STS) tasks. Our experiments also show that UDGN outperforms a transformer-based model that is trained on the same corpus.

4.3.1. Masked Language Modeling

Masked Language Modeling (MLM) is a macroscopic evaluation of the model's ability to deal with various semantic and linguistic phenomena (e.g. co-occurrence, syntactic structure, verb-subject agreement, etc). The performance of MLM is evaluated by measuring perplexity on masked words. We trained and evaluated our model on 2 different datasets: the Penn TreeBank (PTB) and BLLIP. In our MLM experiments, each token has an independent chance to be replaced by a mask token `<mask>`, except that we never replace `<unk>` token.

Model	PTB	BLLIP -SM	BLLIP -MD	BLLIP -XL
Transformer	68.9	44.6	22.8	17.0
StructFormer	64.8	43.1	23.4	16.8
UDGN	60.4	40.2	24.2	19.7

Table 6. Masked Language Model perplexities on different datasets.

PTB. The Penn Treebank Marcus et al. [1993] is a standard dataset for language modeling [Mikolov, 2012] and unsupervised constituency parsing [Shen et al., 2018c, Kim et al., 2019a]. It contains 1M words (2499 stories) from Wall Street Journal. Following the setting proposed in Shen et al. [2020], we preprocess the Penn Treebank dataset by removing all punctuations, lower case all letters, and replaces low frequency tokens (< 5) with `<unk>`. The preprocessing results in a vocabulary size of 10798 (including `<unk>`, `<pad>` and `<mask>`).
BLLIP. The Brown Laboratory for Linguistic Information Processing dataset is a large Penn Treebank-style parsed corpus of approximately 24 million sentences from Wall Street Journal. We train and evaluate UDGN on four splits of BLLIP: BLLIP-XS (40k sentences, 1M tokens),

BLLIP-SM (200K sentences, 5M tokens), BLLIP-MD (600K sentences, 14M tokens), and BLLIP-LG (2M sentences, 42M tokens). Following the same setting proposed in Hu et al. [2020] for sentence selection, resulting in each BLLIP split being a superset of smaller splits. All models are then tested on a shared held-out test set (20k sentences, 500k tokens). To make the mask language modeling and parsing results comparable, we use a shared vocabulary for all splits. Just like the PTB dataset, we preprocess the BLLIP dataset by removing all punctuations and lower case all letters. The shared vocabulary is obtained by counting word frequencies on BLLIP-LG dataset and select the words that appear more than 27 times. The resulting vocabulary size is 30232 (including `<unk>`, `<pad>` and `<mask>`), and covers more than 98% tokens in BLLIP-LG split.

The word mask rate when training on both corpora is 30%. In Section 4.3.4, we further explore the relationship between mask rate and parsing results. Other hyperparameters are tuned separately for each model and dataset. The masked language model results are shown in Table 6. UDGN outperforms the baselines on smaller datasets (PTB, BLLIP-SM), but underperforms against baselines trained on large datasets (BLLIP-MD, BLLIP-LG). However, in Section 4.3.5, we find that the UDGN pretrained on BLLIP-LG dataset can achieve stronger performance when finetuned on a downstream task. This may suggest that our model learns more generic contextual embeddings.

4.3.2. Unsupervised Dependency Parsing

Methods	UAS
DMV [Klein and Manning, 2004]	35.8
E-DMV [Headden III et al., 2009]	38.2
UR-A E-DMV [Tu and Honavar, 2012]	46.1
CS* [Spitkovsky et al., 2013]	64.4*
Neural E-DMV [Jiang et al., 2016]	42.7
Gaussian DMV [He et al., 2018]	43.1 (1.2)
INP [He et al., 2018]	47.9 (1.2)
Neural L-PCFGs [Zhu et al., 2020]	40.5 (2.9)
StructFormer [Shen et al., 2020]	46.2 (0.4)
UDGN (Chu-Liu)	**50.2 (1.5)**
UDGN (Argmax)†	52.5 (0.7)

Table 7. Dependency Parsing Results on WSJ test set without gold POS tags. Starred entries (*) benefit from additional punctuation-based constraints. Daggered entries (†) takes the argmax of head distribution without a tree constraint. Baseline results are from He et al. [2018]. UAS stands for Unlabeled Attachment Score. Unsupervised dependency parsing results with the knowledge of gold POS tags are excluded from this table.

We convert the human-annotated constituency trees from the Wall Street Journal test set Marcus et al. [1993] to dependency trees and use the unlabelled attachment score (UAS) as

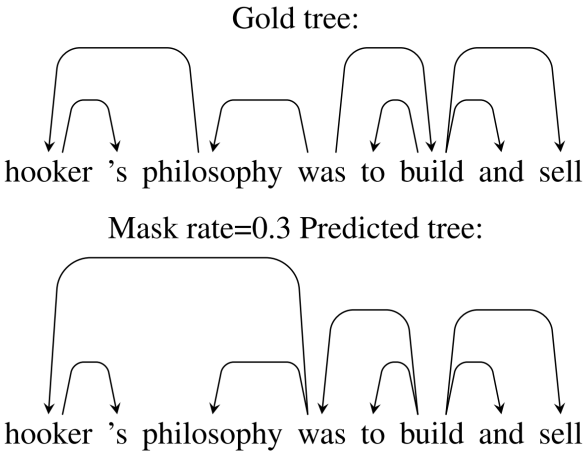


Fig. 11. A example of gold tree and model generated dependency tree.

Models	prep	pobj	det	compound	nsubj	amod
UDGN	0.65(0.12)	0.60(0.11)	0.68(0.15)	0.42(0.04)	0.50(0.06)	0.39(0.07)
StructFormer	0.39(0.05)	0.38(0.07)	0.57(0.03)	0.33(0.01)	0.25(0.06)	0.26(0.01)
Transformer	0.43(0.00)	0.46(0.03)	0.46(0.12)	0.30(0.01)	0.39(0.15)	0.26(0.02)

Table 8. The *pearson correlation coefficients* between most frequent dependency types and their most correlated channel. All results are average across four random seeds, standard derivation are in parentheses. Types are arrange from the highest frequency to lower frequency.

our metric. To derive valid trees from the attention mask, we use the Chu-Liu Chu and Liu [1965] (or Edmonds' Edmonds 1967) algorithm to obtain the maximum directed spanning tree. We also report the argmax over the p — we take the word at the maximum p value for each word the word i. This can result in non-tree structures, but we believe that this metric gives a better indication of how often the parser predicts the right head of each word. Following previous research [Shen et al., 2020], we use the model trained on the preprocessed PTB dataset (no punctuations), and test its parsing performance on section 23 of the WSJ corpus. Punctuation is ignored during the evaluation.

Table 7 shows that our model outperforms baseline models. This result suggests that, given our minimum inductive bias (a token must attach to another, but the graph is not necessarily a tree), predicting missing tokens implicitly learns a good graph that correlates well with human-annotated dependency trees. This may suggest that some of the dependency relations proposed by linguists correspond with efficient ways of propagating information through the sentence. Figure 11 shows a parsing example of our model after training with different mask rates.

4.3.3. Correlation Between Channels and Dependency Types

In this section, we test the correlation between channels and dependency types. We consider each dependency edge $i \rightarrow j$ (i depends on j) in the ground truth structure as a data point. Given all the edges, we can obtain three sets of quantities: channel probabilities $A^k = \{\hat{a}^k_{ji}\}$ and type values $Y^l = \{y^l_{ij}\}$. $\hat{a}^k_{ij}$ is a real value between 0 and 1, represents the probability that channels k is used to model the information propagation from the child i to the parent j. Details about this value can be found at Equation 4.2.11. y^l_{ij} is a binary value, represents whether the label l is assigned to edge $i \rightarrow j$. We can then compute Pearson Correlation Coefficient (PCC) for every pair of A^k and Y^l across all ground truth edges $\{i \rightarrow j\}$:

$$\rho_{A^k,Y^l} = \frac{\text{cov}(A^k, Y^l)}{\sigma_{A^k}\sigma_{Y^l}} \tag{4.3.1}$$

where $\text{cov}(\cdot)$ is the covariance function, $\sigma_\cdot$ is the standard deviation of the respective variable. Hence, ρ_{A^k,Y^l} measures the correlation between channel k and dependency type l. $\rho_{A^k,Y^l} > 0$ means that the model tends to use channel k for propagating information from child to parent for dependency edges of the type l. Here, we only consider the information propagation from child to parent even though information can propagate in both directions in masked language models.

Table 8 shows the PCC between the most frequent dependency types and their most correlated channels. We can observe that all three models have channels that are positively correlated to human-annotated dependency types. This result is coherent with the observation of Htut et al. [2019]. Meanwhile, the UDGN achieves a significantly better correlation than the StructFormer and the Transformer. This confirms our intuition that competitive gated channels can better induce dependency types.

4.3.4. Ablation Experiments

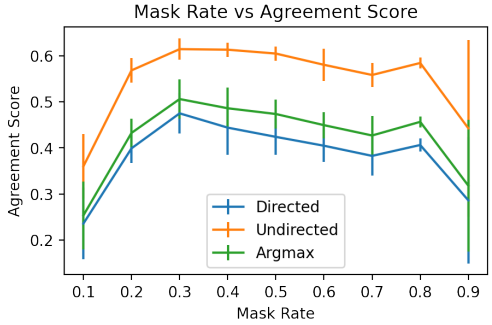


Fig. 12. Relationship between parsing performance and mask rate for MLM.

Mask rate and parsing interaction. One of the more surprising findings in our experiments with this architecture was the relationship between the word mask rate in the MLM task and how much the resulting parse trees corresponded to the ground-truth parse trees.

We trained 5 models for different word masking rates from 0.1 to 0.9, in 0.1 increments, and computed the argmax, UAS, and undirected UAS (UUAS) scores for each of these models. Figure 12 shows the plot for these results.

Firstly, we observe that the acceptable range of masking rate for achieving a decent UUAS score was fairly large: the optimal was at about 0.3, but values of 0.2 up to 0.8 worked to induce tree structures that resulted in fairly good undirected trees. Secondly, as we move away from the optimum of 0.3-0.4, the variance of our results increases, with the highest variance when we mask at a rate of 0.9. Finally, our model supplies the attention mask as a symmetric matrix— the directionality of the mask is decimated when we perform Equation 4.2.8. Consequently, we find that the variance of the UAS is higher than UUAS as the connectivity of the nodes in the tree is more important than the direction of the connection in our architecture.

Model	MLM	Argmax		Chu-Liu	
	PPL	UAS	UUAS	UAS	UUAS
UDGN	60.4(0.8)	52.5(0.7)	58.8(0.9)	50.2(1.5)	61.2(0.4)
- Nonlinear	61.2(1.0)	49.5(1.1)	56.8(1.4)	45.6(2.0)	60.8(1.4)
- Gates	69.5(1.9)	31.5(2.2)	40.7(0.3)	26.1(2.1)	48.9(0.5)
- Competition + Sigmoid	73.6(3.1)	44.7(1.9)	54.4(1.9)	40.4(1.6)	56.6(2.1)
- Competition + Single channel	663.1(18.6)	3.2(0)	6(0)	1.3(0)	6.1(0)
- relative pos bias + pos enc	65.2(3.4)	47.1(7.3)	55.4(4.1)	44.8(7.2)	58.2(5.2)

Table 9. The performance of UDGN after removing different components. "- Nonlinear" means remove the tanh activation function gated channels. "- relative pos bias + pos enc" means using a trainable positional encoding to replace the relative position bias. "- Gates" means remove the gate **g** in gated channels. "- Competition + Sigmoid" means using a non-competitive sigmoid function to replace the competitive softmax in the competitive mechanism. "- Competition + Single channel" means using a single big channel to replace multi-channels in competitive mechanism, and the number of total remains the same. UUAS stands for Undirected Unlabeled Attachment Score.

Effects of Model Components. Table 9 shows the model's performance when individual components are removed. The most significant decrease is caused by replacing multiple channels with one single big channel in each layer. Although the total number of parameters remains the same, MLM and parsing performance is greatly affected. The parser for this model cannot induce structure and predicts a uniform distribution over the other tokens. The second, and more important, parsing performance decrease is caused by removing the gating mechanism. This change forces each channel to always extract the same information from a given key node h_j, regardless of the query node h_i. This has a similar effect as

the previous change, reducing the diversity of different functions that can be modeled by channels. These two observations may suggest that the diversity of information propagation function (multiple channels) is essential to induce a meaningful structure.

Another interesting observation is that relative position bias helps the model to achieve better perplexity and parsing performance in comparison with positional encoding. This may suggest that the combination of dependency graphs and relative position is more informative than absolute positions.

Dataset	#tokens	MLM	Argmax		Chu-Liu	
		PPL	UAS	UUAS	UAS	UUAS
BLLIP-XS	1M	133.7(3.1)	51.4(2.0)	57.6(1.6)	47.9(2.7)	61.2(1.6)
BLLIP-SM	5M	40.2(0.8)	53.7(2.5)	60.7(0.6)	50.9(5.3)	65.1(1.6)
BLLIP-MD	14M	24.2(0.5)	50.5(6.1)	59.8(2.9)	47.7(8.1)	63.0(4.2)
BLLIP-LG	42M	19.7(0.3)	45.6(2.9)	61.7(1.8)	41.6(4.2)	62.5(1.6)

Table 10. The performance of UDGN after trained on different BLLIP splits. Since they share the same vocabulary and test set, results are comparable. While UAS have a high variance, UUAS remain stable across different corpus sizes. Since DGN only use an undirected dependency mask, the choice of dependency direction could be arbitrary.

Corpus Size. Table 10 shows UDGN's performance after training on datasets of different sizes. While the MLM performance improves significantly, the unsupervised parsing performance (UUAS) remains stable. This may suggest that syntax can be acquired with a relatively small amount of data. It is possible then, that where extra data helps is in terms of semantic knowledge, like common sense.

4.3.5. Fine-tuning

Model	STS12	STS13	STS14	STS15	STS16	STS-B	SICK-R	Avg.
Transformer	76.17	61.48	73.97	74.35	53.72	64.26	80.00	69.14
UDGN	80.51	75.02	80.54	82.16	64.73	72.49	81.94	76.77
UDGN (Freeze parser)	77.71	71.17	78.71	82.30	66.04	70.13	82.17	75.46

Table 11. Sentence embedding performance on STS tasks. All models are pretrained on BLLIP-LG, and finetuned on STS. The sentence embeddings are obtained by averaging the output vector across all positions. Freeze parser means that the parameters for the parser are not updated during finetuning.

In this experiment, the goal was to determine if a better representation of semantics can be encoded if the model was constrained for structure. We pretrain a UDGN model on the BLLIP-XL dataset, and then finetune it on the STS-B Cer et al. [2017] dataset. For a controlled experiment, we compare the results we attain with the previously mentioned

Transformer model. We then evaluate the resulting classifier on the STS 2012-2016 Agirre et al. [2012, 2013, 2014, 2015, 2016], the SICK-Relatedness Marelli et al. [2014] dataset, and STS-B Cer et al. [2017]. These datasets were downloaded and prepared using the scripts from Infersent Conneau et al. [2017]. We then report the Spearman correlation score for each dataset (the 'all' setting in Gao et al. 2021).

We find that the UDGN model performs better overall compared to the transformer model. While these are not state-of-the-art results on these tasks, the purpose of our comparison was to examine the benefit of the UDGN model over the Transformer when trained on the same dataset, without conflating the effects of the pretraining dataset size. Other models trained on more data exist, with better performance on these tasks.

4.4. The Future of Dependency-based Models

UDGN shows that dependency grammars have strong compatibility with transformer like models. Replacing the transformer in BERT-like models with UDGN seems to be a natural next step. Experimental result also suggest that UDGN seems capable of achieving better finetuning performance compared to the Transformer. Beside simply improving performance, there may be other benefits. Such as the ability to use it as an unsupervised dependency parser. Structures revealed by this unsupervised parser could have interest to some linguistician and help researchers to evaluate the quality of language model. Furthermore, the separation of parser and DGN provides a better schema for cross-lingual language models (XLMs). Given that most languages can be parsed into the same universal dependency grammar, we could image a XLM that has a separate parser for each language and a shared DGN. The language-specific parser should parse the input sentence into a language-agnostic dependency graph, and the shared DGN compute contextualized embedding from the graph. Another potential application is unsupervised or semi-supervised knowledge extraction. The unsupervised dependency parser provide a way to extract noisy relations from the training corpus. With some filtering and post-processing, these relations could form a large scale knowledge graph.

Chapter 5

Beyond Natural Language: Hierarchical Imitation and Reinforcement Learning (HIRL)

Acquiring primitive skills from demonstrations and reusing them to solve a novel long-horizon task is a hallmark in human intelligence. For example, after learning the necessary skills (e.g., steering wheel, changing lanes) at a driving school, one could be capable of driving across the country by recombining the learned skills, which has a much longer time-scale than driving school practice. On the other hand, the idea of decompose a long-horizon task to a sequence of skills is very coherent with the idea of parsing sentences, if two hypothesis are satisfied: 1) the agent can only execute one skill at a time; 2) once a skill is in execution, it must keep executing until it's finished. Under this two hypothesis, we can view the normal task-skill hierarchy as a two level tree structure, and a complicated multi-level hierarchy as a multi-level tree structure.

In this chapter, we propose *Option-Control Network* (OCN) – a new HIRL model that can decompose a long-horizon task to several subtasks and separately model these subtasks as reusable skills. We start from reviewing related works in HIRL (Section 5.1). We then introduce OCN (Section 5.2). The model is developed from Ordered Memory Policy Network (OMPN) [Lu et al., 2021]. It includes the ordered neurons inductive bias to build a innate hierarchical structure. Finally, we present the experiment results in two different settings (Section 5.3). These experiment results show that OCN and effectively induce and reuse skills in the Craft environment [Andreas et al., 2017].

5.1. Previous Approaches

One general approach is to leverage the additional *unstructured demonstrations* during pretraining, e.g., compILE [Kipf et al., 2019] pretrains a VAE [Kingma and Welling, 2013]

on the demonstrations and uses an action decoder for finetuning. Our work is in this line of research.

Learning to solve temporally extended tasks is an important question for Hierarchical Reinforcement Learning (HRL). Different temporal abstractions are proposed to achieve structured exploration and transfer to a long-horizon task, including option frameworks [Sutton et al., 1999], HAM [Parr and Russell, 1998] and max-Q [Dietterich, 2000]. With the popularity of neural nets, recent works propose to use a bi-level neural network such as option critics [Bacon et al., 2017], feudal networks [Vezhnevets et al., 2017], generative models with latents [Nachum et al., 2018], and modulated networks [Pashevich et al., 2018]. These models can be furthered combined with hindsight memory [Levy et al., 2018] to increase the sample efficiency. Our work can also be viewed as designing a specific neural architecture for HRL.

However, a pure HRL method suffers from serious exploration challenges when learning from scratch [Gupta et al., 2019]: it takes a significant amount of samples for random walk to induce a good temporal abstraction that leads to positive rewards at the beginning of training. A general approach to tackle this problem is to introduce a pretraining phase to "warm up" the policy. Recent works propose to pretrain the policy with an intrinsic diversity reward [Eysenbach et al., 2018] or language abstraction [Jiang et al., 2019], which is shown to be useful in the HRL. Other works [Le et al., 2018, Levy et al., 2018, Gupta et al., 2019] have focused on learning useful skills in a pretraining phase first, and then reusing these skills when finetuning with HRL in the new environment. However, these methods either assume the existence of goal-conditioned policies or access to environments, which limits the practical values of these approaches. One general approach is to leverage the additional *unstructured demonstrations* during pretraining, e.g., compILE [Kipf et al., 2019] pretrains a VAE [Kingma and Welling, 2013] on the demonstrations and uses an action decoder for finetuning. Our work is in this line of research.

Recent works build upon this "imitation - finetune" paradigm. With the prevalence of goal-conditioned policies [Schaul et al., 2015, Kaelbling, 1993, Levy et al., 2018] in robotics, these methods leverage demonstrations with relabelling technique to pretrain the low-level policy [Gupta et al., 2019] or a generative model [Lynch et al., 2020]. However, they exploit the fact that the ending state of a trajectory segment can be described as a point in the goal space. Hence it is difficult to apply them beyond goal-conditioned policies. CompILE [Kipf et al., 2019] treats the segment boundaries as latent variables, and their model can be trained end-to-end with soft trajectory masking. However, CompILE requires specifying the number of segments, which is a much more limiting constraint than that required by OCN. Nevertheless, it is designed to be a general method so we use it as our main baseline. Modular policy networks [Andreas et al., 2017, Shiarlis et al., 2018] are also used in this paradigm, where each subtask corresponds to a single modular policy. However, in this setting, the demonstration needs to be segmented beforehand, which requires additional

human labor. On the contrary, our work focused on using unstructured demonstrations. OptionGAN [Henderson et al., 2018] proposes a Mixture-of-Expert (MoE) formulation and performs IRL on the demonstration. However, without an explicit termination function, the learnt expert networks do not provide time-extended actions for the high-level controller. As a result, this method still suffers from problems of exploration with sparse rewards (as also seen in our experimental comparison with an MoE baseline).

Extracting meaningful trajectory segments from the unstructured demonstration is the focus of Hierarchical Imitation Learning (HIL). These works can be summarized as finding the optimal behavior hierarchy so that the behavior can be better predicted [Solway et al., 2014]. DDO [Fox et al., 2017] proposes an iterative EM-like algorithm to discover multiple levels of options, and it is applied in the continuous action space [Krishnan et al., 2017] and program modelling [Fox et al., 2018]. VALOR [Achiam et al., 2018] extends this idea by incorporating powerful inference methods like VAE [Kingma and Welling, 2013]. Directed-Info GAIL [Sharma et al., 2018] extracts meaning segments by maximizing the mutual information between the subtask latent variables and the generated trajectory. Ordered Memory Policy Network (OMPN) [Lu et al., 2021] proposes a hierarchical inductive bias to infer the skill boundaries. The above works mainly focus on skill extraction, so it is unclear how to use the segmented skills for RL finetuning. Although OCN shares a similar inductive bias with OMPN, OCN replaces the continuous hidden states communication with a softmax distribution over multiple low-level modules (options). This enables OCN to model different subtasks with different options and to effectively reuse them in a new task.

5.2. Option-Controller Network

In this section, we propose *Option-Control Network* (OCN). Unlike previous works, our method does not require generative models [Eysenbach et al., 2018], goal-conditioned policies [Gupta et al., 2019], pre-specified policy sketch [Shiarlis et al., 2018] or constraints on the number of segments [Kipf et al., 2019], making our approach conceptually simple and general. An OCN includes a set of N options $\{\mathbf{o}_1, ..., \mathbf{o}_N\}$ and a controller $\mathbf{c}$. As shown in figure 13, the OCN starts by using the controller to choose an option to execute the first subtask. Once the subtask is done, the controller will choose another option to execute the second subtask, and so on, until the goal of the task is achieved. Inspired by OM and OMPN [Lu et al., 2021], we use the ordered neurons inductive bias to enforce the hierarchical constraint between the controller and the options so that the high-level controller is updated less frequently than the low-level options while keeping the model end-to-end differentiable.

Another intuition behind the OCN is making controllers and options running independently. So options can be easily reused by another controller to solve a new task. Each component directly takes raw observation as input and only interacts with other components

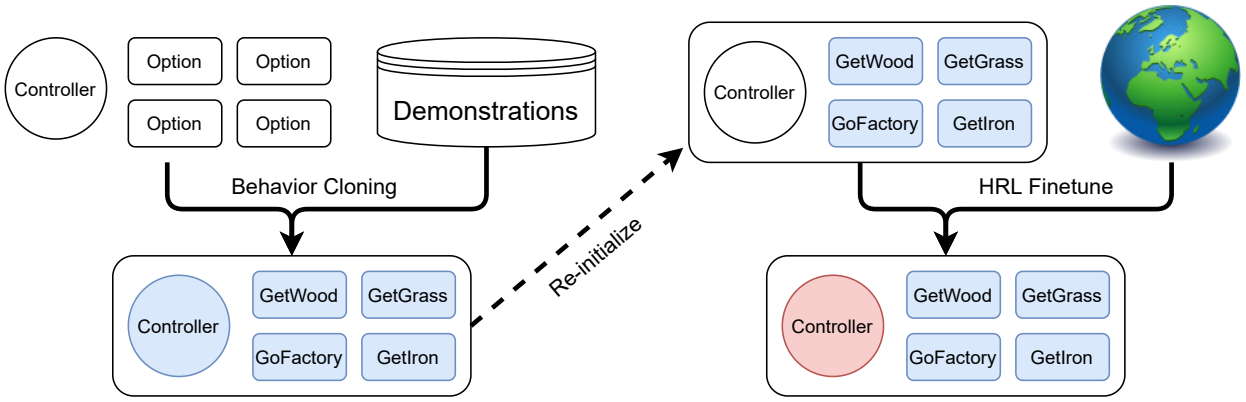


Fig. 13. The training pipeline of OCN. Our model is composed of a controller (circle) and a options pool (rectangles). The controller and options are randomly initialized, which means each option does not correspond to a meaningful subtask. After behavior cloning, both options and controllers are induced (marked blue) and the options correspond to meaningful subtasks from demonstrations (e.g., get wood). Then we freeze the parameters in the options and re-initialize the controller. The controller is trained to adapt to the new environment with HRL (marked red).

through the probability distribution p_t. In other words, an option can always execute the induced skill by itself, regardless of the higher-level task. A controller can consider options as black boxes with an on/off switch so the controller can achieve structured exploration. Given a new (possibly) complicated and long-horizon task and a set of learnt options, the controller can then quickly find a solution from the induced option space, enabling structured exploration.

As shown in Figure 13, OCN can jointly learn options and controllers with multitask behavior cloning from unstructured demonstrations. When given a new task, one could perform HRL finetuning by re-initializing the controller and freezing the options. This enables our model to generalize combinatorially to unforeseen conjunctions [Denil et al., 2017].

5.2.1. Option and Controller

Option As shown in the middle of Figure 14, an option $\mathbf{o}_i$ models a skill that can solve one specific subtask, for example *get wood*, *get iron* or *make at workbench*. It can be described as:

$$\mathbf{p}^{\mathbf{o}}_{i,t}, \mathbf{h}^{\mathbf{o}}_{i,t}, e_{i,t} = \mathbf{o}_i(\mathbf{x}_t, \mathbf{h}^{\mathbf{o}}_{i,t-1}) \tag{5.2.1}$$

where $\mathbf{x}_t$ is the observation at time step t, and $\mathbf{h}^{\mathbf{o}}_{i,t-1}$ is the hidden state of the respective option at time step $t-1$; $\mathbf{h}^{\mathbf{o}}_{i,t}$ is the hidden state of $\mathbf{o}_i$ at time step t; $e_{i,t}$ is a scalar between 0 and 1, represents the probability that the current option is done; $\mathbf{p}^{\mathbf{o}}_{i,t}$ is a distribution of actions, including *move up*, *move left* and *use*. These actions are the smallest elementary operations that an agent can execute. During the execution of an option, if probability $e_{i,t}$

is 0, the option will keep executing the current subtask; if $e_{i,t}$ is 1, the option will stop the execution and return to the controller for the next subtask. In our work, each option maintains a separate set of parameters.

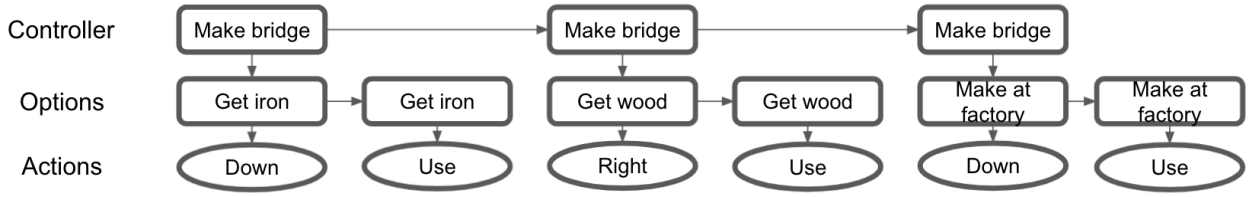


Fig. 14. An example of OCN. The controller **c** models the task *make bridge*. Three options separately model subtasks *get iron*, *get wood* or *make at factory*.

Controller As shown at the top of Figure 14, a controller **c** models a higher level task, like *make bed*, *make axe*, or *make bridge*. Each of these tasks can be decompose to a sequence of subtasks. For example, *make bridge* can be decompose to 3 steps: 1) *get iron*, 2) *get wood*, 3) *make at factory*. Thus a controller can also be represented as:

$$\mathbf{p}_t^{\mathbf{c}}, \mathbf{h}_t^{\mathbf{c}}, e_t^{\mathbf{c}} = \mathbf{c}(\mathbf{x}_t, \mathbf{h}_{t-1}^{\mathbf{c}}) \tag{5.2.2}$$

where $\mathbf{p}_t^{\mathbf{c}}$ is a distribution over the set of options $\{\mathbf{o}_i\}$, $\mathbf{h}_t^{\mathbf{c}}$ is the hidden state for controller, $e_t^{\mathbf{c}}$ is the probability that the current task is done. In this OCN architecture, we don't need the $e_t^{\mathbf{c}}$, since the environment will provide signal(reward) once the task is done. However, OCN can be easily expanded to a multi-level model. In this multi-levels model, a set of multiple controllers become options for a higher-level controller, and their respective tasks become subtasks for a more complicated task.

Cell Network In OCN, options and controllers share the same format for input and output. Thus, we parameterize them with the same neural network architecture. To model the policy of controllers and options, we proposed the following cell network:

$$\hat{h}_t = \text{MLP}\left(\left[x_t, h_{t-1}\right]\right) \tag{5.2.3}$$

$$p_t = \text{softmax}(\mathbf{W}_{\text{act}}\hat{h}_t + \mathbf{b}_{\text{act}}) \tag{5.2.4}$$

$$h_t = \tanh(\mathbf{W}_{\text{hid}}\hat{h}_t + \mathbf{b}_{\text{hid}}) \tag{5.2.5}$$

$$e_t = \text{sigmoid}(\mathbf{w}_{\text{end}}\hat{h}_t + b_{\text{end}}) \tag{5.2.6}$$

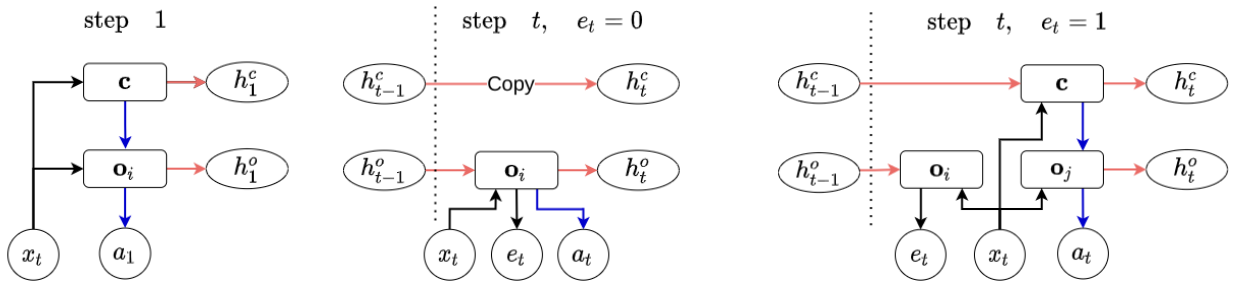
x_t is the raw observation, the shape of the vector depends on the environment. h_{t-1} is the recurrent hidden state of size d_{hid}, it allows the model to remember important information from previous time steps. MLP is a multi-layer neural network of Depth l_{MLP} and hidden size d_{MLP}. We use tanh as activation function for MLP. $\hat{h}_t$ is a vector of size d_{MLP}. $\mathbf{W}_{\text{act}}$ is a matrix of size $n_{\text{act}} \times d_{\text{MLP}}$, where n_{act} is number of actions. $\mathbf{W}_{\text{hid}}$ is a matrix of size $d_{\text{hid}} \times d_{\text{MLP}}$. $\mathbf{w}_{\text{end}}$ is a vector of size d_{MLP}. Following the fast and slow learning idea proposed

in Madan et al. [2021], we introduce a temperature term T to controller's softmax function:

$$p_t^{\mathbf{c}} = \text{softmax}\left(\frac{\mathbf{W}_{\text{act}}\hat{h}_t + \mathbf{b}_{\text{act}}}{T}\right) \tag{5.2.7}$$

A large temperature T allows the option to output smoother distribution at the beginning of training. It also reduces the scale of gradient backpropagated into the controller. This results in the controller changes and updates slower than options. We found T makes OCN become more stable in imitation learning and converge to a better hierarchical structure.

5.2.2. Option-Controller Framework



(a) First time step **(b)** Inside a subtask **(c)** Switching between subtasks

Fig. 15. The three different phase of OCN: (a) At the first time step, the controller selects an option $\mathbf{o}_i$; The option $\mathbf{o}_i$ outputs the first action $\mathbf{a}_1$. (b) If the previous option $\mathbf{o}_i$ predict that the subtask is not finish; The option $\mathbf{o}_i$ then continue outputs action $\mathbf{a}_t$; The controller hidden state is copied from previous time step. (c) If the previous option $\mathbf{o}_i$ predict that the subtask is done; The controller then selects a new option $\mathbf{o}_j$ and updates the controller hidden state; The new option $\mathbf{o}_j$ outputs action $\mathbf{a}_t$. Blue arrows represent probability distributions output by controller and options. Red arrows represent recurrent hidden states between time steps.

Given the definition for options and controllers, we can further formulate OCN. As shown in Figure 15a, at the first time step, the controller computes a probability distribution over options for the first subtask, and options execute their first steps:

$$\mathbf{p}_1^{\mathbf{c}}, \mathbf{h}_1^{\mathbf{c}} = \mathbf{c}(\mathbf{x}_1, \mathbf{h}_0^{\mathbf{c}}) \tag{5.2.8}$$

$$\mathbf{p}_{i,1}^{\mathbf{o}}, \mathbf{h}_{i,1}^{\mathbf{o}}, e_{i,1} = \mathbf{o}_i(\mathbf{x}_1, \mathbf{h}_{i,0}^{\mathbf{o}}) \tag{5.2.9}$$

$$\mathbf{p}_1^{\mathbf{a}} = \sum_i p_{1,i}^{\mathbf{c}} \mathbf{p}_{i,1}^{\mathbf{o}} \tag{5.2.10}$$

where $h_0^{\mathbf{c}}$ and $h_{i,0}^{\mathbf{o}}$ are initial hidden states for controller and options, $p_1^{\mathbf{a}}$ is a distribution for actions. The output $p_1^{\mathbf{a}}$ is formulated as a mixture of experts, where experts are options and the gating model is the controller.

At time steps $t > 1$, the options first execute one step to decide whether this subtask is done. If the subtask is unfinished, the option then outputs an action distribution, as shown

in Figure 15b:

$$\hat{\mathbf{p}}^{\mathbf{o}}_{i,t}, \hat{\mathbf{h}}^{\mathbf{o}}_{i,t}, e_{i,t} = \mathbf{o}_i(\mathbf{x}_t, \mathbf{h}^{\mathbf{o}}_{i,t-1}) \tag{5.2.11}$$

$$e_t = \sum_i p^{\mathbf{c}}_{t-1,i} e_{i,t} \tag{5.2.12}$$

$$\hat{\mathbf{p}}^{\mathbf{a}}_{i,t} = \sum_i p^{\mathbf{c}}_{t-1,i} \hat{\mathbf{p}}^{\mathbf{o}}_{i,t} \tag{5.2.13}$$

where e_t is the probability that the previous subtask is done and $\hat{\mathbf{p}}^{\mathbf{a}}_{i,t}$ is the action distribution if the subtask is not done. If the previous subtask is done, the controller $\mathbf{c}$ need to select a new option distribution for the next subtask and reinitialize the option, as shown in Figure 15c:

$$\mathbf{p}'^{\mathbf{c}}_t, \mathbf{h}'^{\mathbf{c}}_t = \mathbf{c}(\mathbf{x}_t, \mathbf{h}^{\mathbf{c}}_{t-1}) \tag{5.2.14}$$

$$\mathbf{p}'^{\mathbf{o}}_{i,t}, \mathbf{h}'^{\mathbf{o}}_{i,t}, e'_{i,t} = \mathbf{o}_i(\mathbf{x}_t, \mathbf{h}^{\mathbf{o}}_{i,0}) \tag{5.2.15}$$

$$\mathbf{p}'^{\mathbf{a}}_t = \sum_i p'^{\mathbf{c}}_{t,i} \mathbf{p}'^{\mathbf{o}}_{i,t} \tag{5.2.16}$$

where $\mathbf{h}'^{\mathbf{c}}_t$, $\mathbf{h}'^{\mathbf{c}}_t$, $\mathbf{p}'^{\mathbf{c}}_t$ and $p'^{\mathbf{c}}_{t,i}$ are hidden states and distributions for the next subtask if the previous subtask is done. Thus, we can formulate the output at time step t as a weighted sum of the two situations:

$$\begin{bmatrix} \mathbf{h}^{\mathbf{c}}_t \\ \mathbf{p}^{\mathbf{c}}_t \\ \mathbf{h}^{\mathbf{o}}_t \\ \mathbf{p}^{\mathbf{a}}_t \end{bmatrix} = e_t \begin{bmatrix} \mathbf{h}'^{\mathbf{c}}_t \\ \mathbf{p}'^{\mathbf{c}}_t \\ \mathbf{h}'^{\mathbf{o}}_t \\ \mathbf{p}'^{\mathbf{a}}_t \end{bmatrix} + (1 - e_t) \begin{bmatrix} \mathbf{h}^{\mathbf{c}}_{t-1} \\ \mathbf{p}^{\mathbf{c}}_{t-1} \\ \hat{\mathbf{h}}^{\mathbf{o}}_t \\ \hat{\mathbf{p}}^{\mathbf{a}}_t \end{bmatrix} \tag{5.2.17}$$

The equation 5.2.17 provides OCN an internal hierarchical inductive bias, that a higher-level component ($\mathbf{c}$) only update its recurrent hidden state and output a new command ($\mathbf{p}'^{\mathbf{c}}_t$) when its current functioning subordinate ($\mathbf{o}_i$) reports "done".

5.2.3. Inducing and Reusing Skills

Imitation Learning and Inducing Skills OCN imitates and induces skills from unstructured demonstrations. In the rest of this paper, $\mathbf{d}$ represents an unstructured demonstration $\{(x_t, a_t)\}_{t=1}^{T}$ $\mathbf{D}$ represents a set of demonstrations of different tasks $[(\mathbf{d}_1, \tau_1), (\mathbf{d}_2, \tau_2), ...]$, where τ_i are task ids, belongs to a shared task set $\mathbf{T}$.

Given a demonstration $\mathbf{d}$, OCN can perform behavior cloning with a negative log-likelihood loss:

$$loss = \text{average}_t \left(\text{NLLLoss}(\mathbf{p}^{\mathbf{a}}_t, a_t)\right) \tag{5.2.18}$$

For different tasks τ, we can use two different methods to model their associated controllers. The first method is to assign one controller $\mathbf{c}_\tau$ and a initial hidden state $\mathbf{h}^{\mathbf{c}}_{\tau,0}$ to each τ. The

second method is to share the controller $\mathbf{c}$, but assign a different initial hidden state $\mathbf{h}^{\mathbf{c}}_{\tau,0}$ to each τ. We choose the second method in this work because sharing $\mathbf{c}$ could avoid the risk that different controllers choose to model the same subtask with options. During the imitation learning, OCN allows gradient backpropagation through all probabilities $\mathbf{p}$. Thus, the gradient descent will try to induce an optimal set of options that can best increase the likelihood of the data.

Reinforcement Learning and Reusing Skills Given the induced options from imitation learning, our model can learn to solve a new task by reusing these skills via reinforcement learning. For example, after training on demonstrations of task 1 and task 2, OCN induce N options $\{\mathbf{o}_1, ..., \mathbf{o}_N\}$. Given a new task 3 without demonstrations, we can initialize a new controller $\mathbf{c}_3$, that takes observations as input and outputs a probability distribution over N induced options. To learn $\mathbf{c}_3$, we freeze all options and use PPO [Schulman et al., 2017] algorithm to learn $\mathbf{c}_3$ from interactions with the environment.

Algorithm 4 PPO, Adapt OCN to a new task

```
Initialize controller c
Freeze all options {o_{1...N}}
for iterations=1,2,... do
    for actor=1,2,... do
        for step t=1,2,...,T do
            p^c = c(x_t, h^c_{t-1})
            i = sample(p^c)
            Rollout o_i until sample(e_i) = 1
        end for
        Compute advantage estimates Â_1, ..., Â_T
    end for
    Optimize surrogate L wrt c, with K epochs and minibatch size B
end for
```

During the training, once the controller outputs an option distribution $p^{\mathbf{c}}$, OCN samples from the distribution, the sampled option will rollout until it's done, then the process will repeat until the task is solved. We outline the process in Algorithm 4. Thus, in the RL phase, our model only needs to explore at options space, which significantly reduces the number of interaction steps to solve the new tasks.

5.3. Experiments

We perform experiments in Craft [Andreas et al., 2017], a grid-world environment focusing on navigation and collecting objects. Our results show that with unstructured demonstrations, OCN can jointly learn to segment the trajectories into meaningful skills as well as model this rich set of skills with our pool of low-level options. During HRL finetuning, we show that OCN achieves better performance in more complex long-horizon tasks with

either sparse or dense reward compared with existing baselines. We also provide further visualization to show the discovered options are reused during finetuning.

Environment. Craft is adapted from previous works [Lu et al., 2021, Andreas et al., 2017]. In this environment, an agent can move in a 2D grid map with actions (*up*, *down*, *left*, *right*) and interact with the objects with the action *use*. In our experiments, a subtask requires the agent to locate and collect a specific type of object. For example, subtask A, *get wood*, requires the agent to first navigate to the block that contains wood, then execute a *use* action to collect one unit of wood. Table 12 provides a list of subtasks used in our experiments. A task requires the agent to finish a sequence of subtasks in the given order. The environment can provide either sparse reward or dense reward. In the dense reward, the agent receives rewards after completing each subtask while in the sparse reward, the agent only receives rewards after completing all subtasks.

Subtask	A	B	C	D
Goal	*get wood*	*get gold*	*get iron*	*get grass*

Table 12. Subtasks and their goals

Baselines. We compare OCN with a number of baselines including task decomposition methods and hierarchical methods. Our baselines are: (1) compILE [Kipf et al., 2019], which leverages Variational Auto-Encoder to recover the subtask boundaries and models the subtasks with different options. (2) OMPN [Lu et al., 2021] which studies inductive bias and discovers hierarchical structure from demonstrations. (3) Mixture-of-Experts (MOE), which uses a similar architecture as OCN, but the options are only executed for one time step. This baseline is inspired by OptionGAN [Henderson et al., 2018] which proposes the MoE framework without modeling termination functions.

Implementation Details. We train all imitation learning methods by utilizing behaviour cloning with a batch size of 512 and a learning rate of 0.001. For each task, we sample 6000 demonstrations and split 80% for training and 20% for validation. For reinforcement learning, we use PPO algorithm with a batch size of 1024 and a learning rate of 0.0003. We use Adam optimizer and a linear schedule to adjust the learning rate. The hidden state size d_{hid} of OCN and baselines is 128. The depth l_{MLP} of cell network is 2. The temperature T for controller's softmax function is 10.

5.3.1. S1: Transferring from Single Model

In this setting, the training task set is $\{\text{AC}, \text{CD}, \text{DA}\}$. During the imitation phase, we pretrain an OCN with one controller $\mathbf{c}_1$ and three options $\{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3\}$ to imitate these demonstrations. During the fine-tuning phase, the model needs to solve three new tasks: $\{\text{ADC}, \text{CAD}, \text{DCA}\}$. We initialize a new controller for each while freezing the parameters

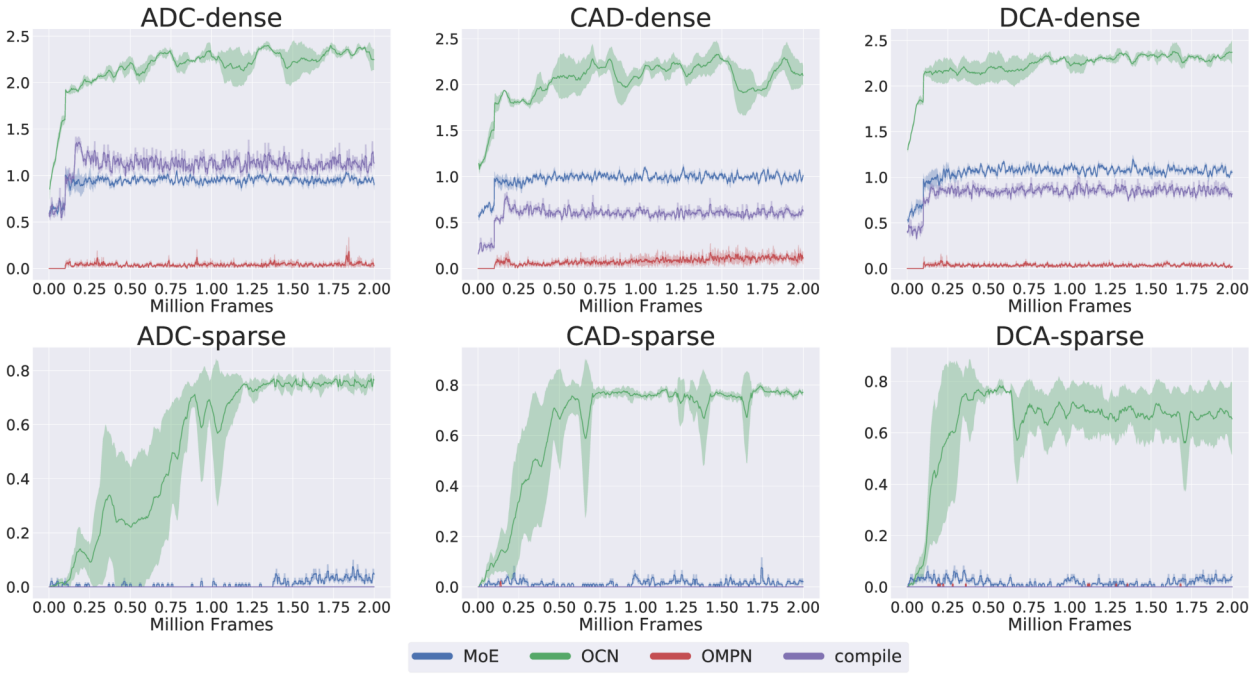


Fig. 16. The learning curve of different methods on three finetuning tasks of **S1**. `dense` means dense reward setting. `sparse` means sparse reward setting.

of options. This is the classical setting where an agent is required to learn skills from short expert demonstrations and to transfer to long-horizon tasks.

As is shown in Figure 16, our method converges faster and achieves higher performance than baselines in both dense and sparse reward settings. With dense rewards, our method achieves double the returns than the strongest baseline. In the sparse reward setting, our method can get an average return of 0.7 with the maximum being 1, while other baselines struggle. We find that MoE fails to achieve similar performance even with a very similar architecture as OCN, since MoE does not model the termination function and the controller selects a new option every time step. This result shows that exploring in option space is more efficient than other schemes, provided the new task is expressible as a combination of previous observed subtasks.

5.3.2. S2: Transferring from Multiple Models

In this setting, we have two disjoint task set. The first set is $\{\text{AB}, \text{BA}\}$ and the second task set is $\{\text{CD}, \text{DC}\}$ We train two separate OCN models. Each model includes a controller and two options. Thus, at the end of imitation phase, we obtain four options $\{\mathbf{o}_1, ..., \mathbf{o}_4\}$. Then we initialize three new controllers to solve three new tasks: $\{\text{BADC}, \text{ACBD}, \text{CABD}\}$.

This setting is related to the problem of data islands and federated learning [Yang et al., 2019], where two companies could each pretrain models on their separate datasets, merge

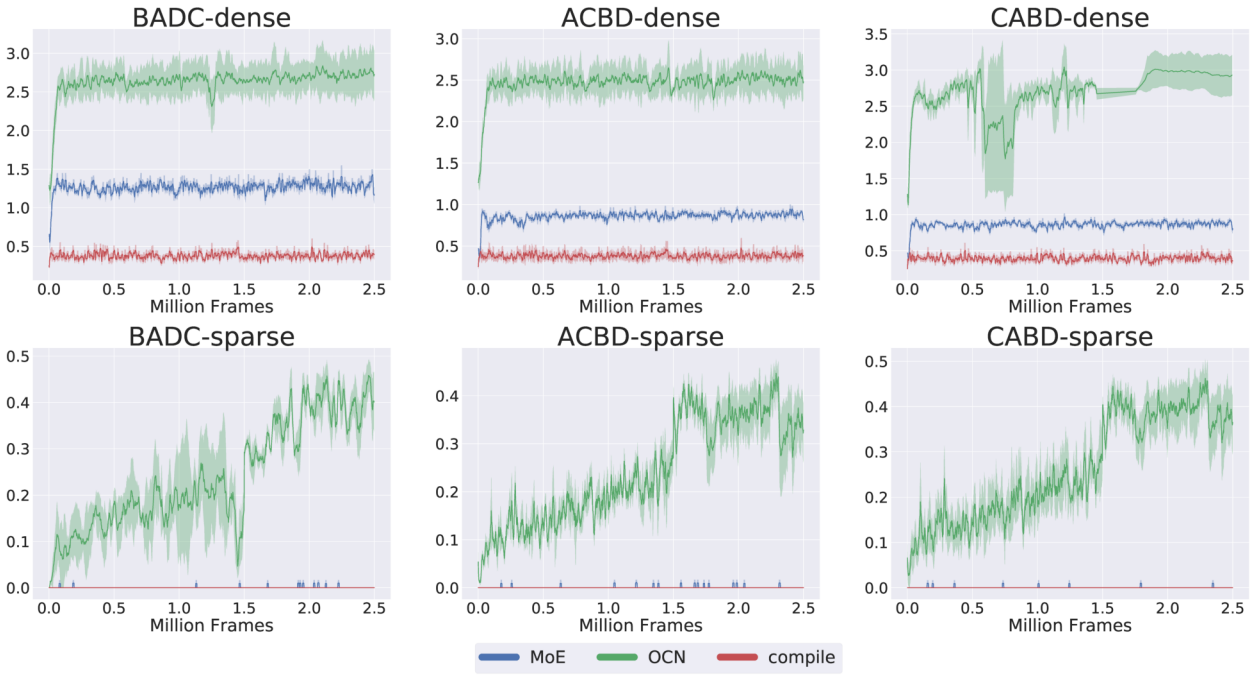


Fig. 17. The learning curve of different methods on three finetuning tasks of **S2**. OMPN is not included because it does not learn an explicit set of options.

the induced options, and share the controller finetuned on more challenging tasks. This is made possible because of the highly *modularized design* of our architecture.

The results are shown in Figure 17. We show that our model can still reuse merged option pools, while other baseline methods fail at this setting. CompILE uses a continuous latent variable for communication between the controller and the action decoder, which causes compatibility issues while merging skills from different models. The MoE method still suffers from the long horizon problem. Overall, this result highlights the flexibility of OCN and its promise in maintaining data privacy for collaborative machine learning applications.

5.3.3. Model Analysis

Visualization. Figure 18 shows a trajectory of the model, which includes options trained on tasks (AC, CD), and controller finetuned on task DCA. As shown in the Figure 18, the discovered options from the demonstrations with shorter horizon (AC, CD) are reused when we finetune the controller in a longer horizon task (DCA). This observation confirms our hypothesis, that options can model subtasks and be reused for another task, from a qualitative perspective.

Quantitative Analysis. Figure 19 shows the performances of parsing and option-subtask correlation during imitation phase. We find that OCN can converge faster and achieve better parsing performance than the OMPN model. Figure 19 right shows that, during the imitation phase, randomly initialized options slowly converged to model different subtasks.

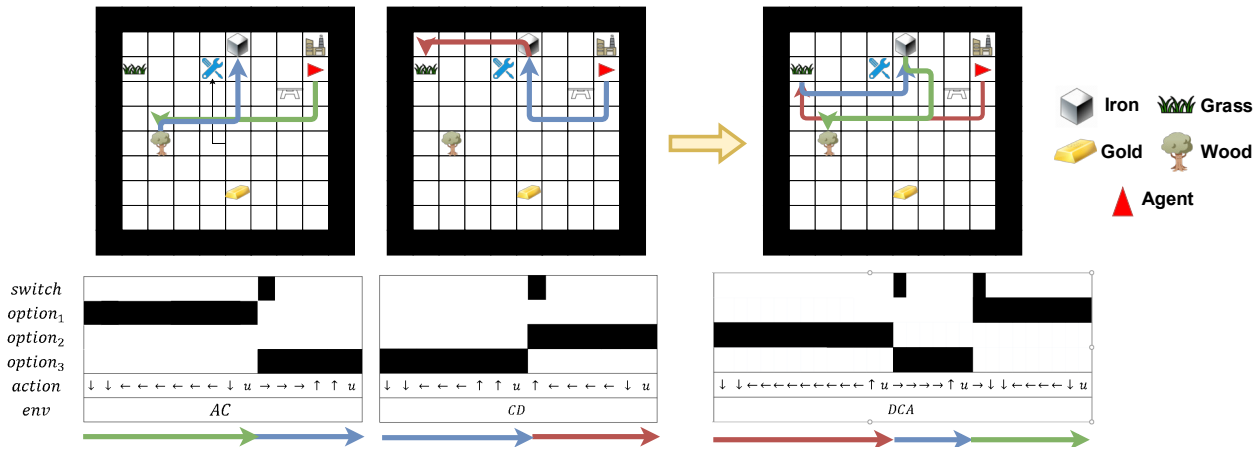


Fig. 18. A trajectory of model finetuned on task DCA in **S1**. `switch` represents the value of e_t at every time step. The option distribution is computed with $\mathbf{p}_t^{\mathbf{c}}$.

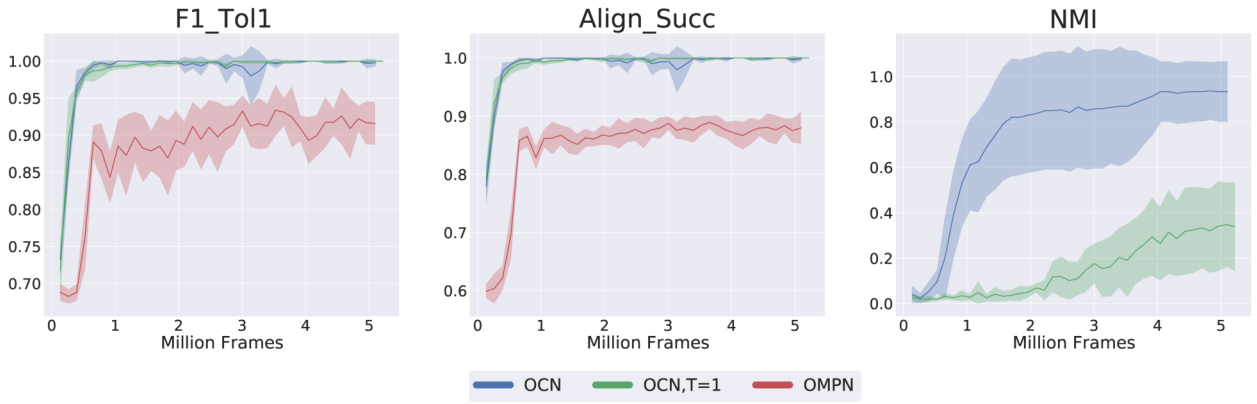


Fig. 19. Comparison of unsupervised trajectory parsing results during the imitation phase with OMPN [Lu et al., 2021]. We use F1 scores with tolerance **(Left)** and Task Alignment (**Center**) to show the quality of learned task boundaries. We compute the normalized mutual information (**Right**) between the emerged option selection $\mathbf{p}_t^{\mathbf{c}}$ and the ground-truth to show that our model learns to associate each option to one subtask. `T=1` means that the temperature term in the controller is removed.

At the end of imitation, OCN shows strong alignment between options and subtasks. In 4 out of 5 runs, OCN actually achieves NMI=1, which means that the alignment between option and subtask is perfect. On the other hand, if we remove the temperature term (i.e. set $T = 1$) in controller, the NMI drops significantly. This result suggest that the fast and slow learning schema is important for the model to learn the correct alignment between options and subtasks. Furthermore, Table 13 shows the success rate of using each option to solve each subtask. We find that there is a one-to-one correspondence between subtasks and learnt options. Overall, these results confirmed our hypothesis that options can be used to model subtasks from a quantitative perspective.

Option \ subtask	A	C	D
1	0.96	0.07	0.03
2	0.00	0.02	0.95
3	0.01	0.98	0.01

Table 13. The success rate of each option when testing on different subtasks.

5.3.4. Hyperparameters Analysis

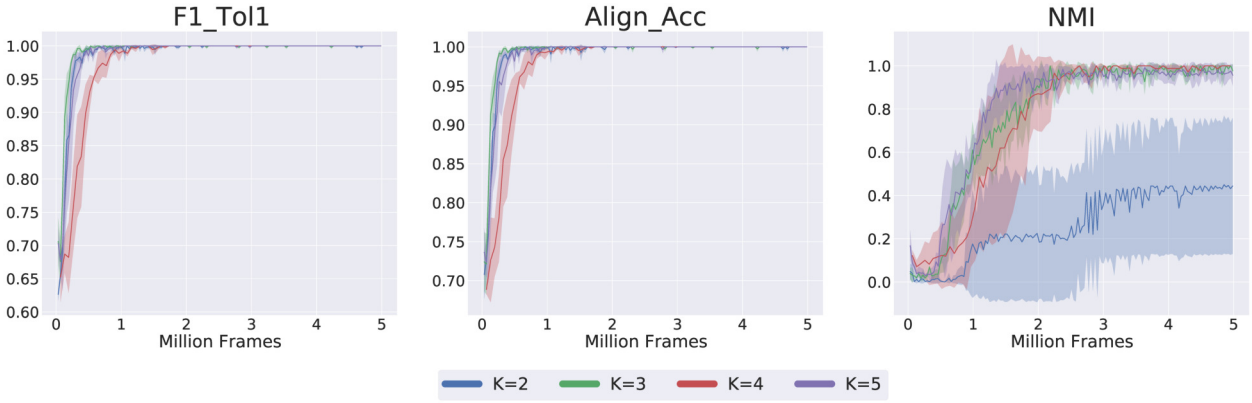


Fig. 20. Comparison of parsing results during different K at F1 scores with tolerance, task align accuracy and NMI.

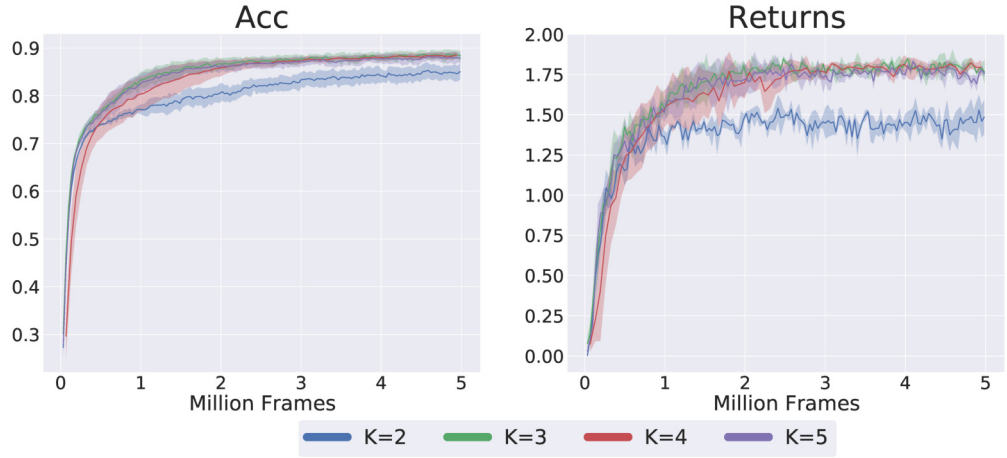


Fig. 21. Comparison of prediction accuracy of actions and the returns during different K

Our model does not require the assumption about the number of skills. We analyze the effect of the number of options K. As shown in Figure 20, when K is larger than or equal to the number of skills, which is 3 in this experiment, our model basically remains similar results at three metrics: Align Acc, F1 Tol1, and NMI and achieve almost 1. When $K = 2$, which means K is smaller than the number of skills, one of the options must execute two different

skills, which is contrary to our assumption and only achieves 0.4 at NMI. We also compare the prediction accuracy of the actions and the returns in Figure 21. Our performance isn't influenced by the number of skills when K is larger than the number of skills.

5.4. Rethinking Modularization

Modern Neural Network research focuses on training end-to-end models on a large amount of data. This data-driven paradigm has enjoyed great success. However, it also has some drawbacks: 1) a neural network model usually focuses on a single task, which means that we need to deploy a standalone model to the user's device for each functionality; 2) neural network models function like black boxes, when it fails we usually don't have a method to diagnose the source of failure; 3) fixing a fault requires retraining the entire model, which is too expensive for many cases and potentially brings in other problems.

We want to suggest that the combination of syntactic inductive bias and modularization could be a potential solution to these problems. For example, OCN uses the syntactic inductive bias to induce the latent structure of a given task, while maintaining the use of an end-to-end schema. At the same time, OCN recognizes and models each repetitive component in the tree structure as a standalone neural network module (skill) with a human-understandable semantic meaning. This process allows researchers and engineers to understand the internal mechanism of the trained model. Furthermore, each module could be reused to solve other tasks, this could minimize the cost of solving a new task and allow us to deploy a new function to a device without sending an entirely new model. Researchers and engineers can also quickly diagnose the faults in a given model, by identifying the malfunction components. They can then use a small amount of specifically prepared data to retrain the component, such that the fault can be fixed with less cost and without touching the other components of the model.

Though the proposed method and experimental settings in this chapter are toyish, we do hope this could inspire other researchers who want to study the aforementioned problems.

Chapter 6

Conclusion

In this thesis, we introduced a group of syntactic inductive biases for neural network models, as well as their applications in Natural Language Processing and Reinforcement Learning.

In Chapter 3, we reviewed the history of constituency-augmented neural networks and unsupervised constituency parsing. Traditionally, researchers tend to recognize the two as separated domains. However, limited resources and lack of flexibility restricted the development of constituency-augmented models. Grammar induction also suffered from a lack of practical use cases. We proposed to combine the two domains. The basic idea is to add a constituency inductive bias to generic neural network architecture. Thus, the new model is constituency augmented, but only requires raw corpus as training data and does not require extra parsed corpus. On the other hand, the induced grammar can be directly used to improve the model's performance on downstream tasks. We introduced *Ordered Neurons* – a constituency inductive bias for recurrent neural networks. Based on the ordered neurons, we further introduce two neural network architectures: ON-LSTM and Ordered Memory (OM). The ON-LSTM is a simple combination of Ordered Neurons and the LSTM model. It has the same generic interface as a standard LSTM model, making it a good alternative for many natural language tasks. The OM model is designed from scratch based on the same inductive bias. It includes a soft shift-reduce parsing mechanism and an explicit composition function to compose lower-level constituents to higher-level constituents. These mechanisms make OM a good fit for formal language tasks. We present experimental results on formal language tasks, language modeling, and unsupervised constituency parsing.

In Chapter 4, we reviewed the history of dependency-augmented neural networks and unsupervised dependency parsing. Previous works also revealed that the self-attention distributions in transformers have a strong connection to the corresponding dependency relations. Inspired by this observation, we proposed the *dependency-constrained connection* – an inductive bias for transformer or graph neural networks. Based on this inductive bias, we introduced Unsupervised Dependency Graph Network (UDGN). It combines a dependency

parser and a graph neural network. The dependency parser computes probability distribution for a dependency graph. The graph neural network uses the probability distribution to control the information propagation between nodes. While training on a task, the gradient backpropagation updates both the information propagation mechanism (the graph neural network) and the parser to minimize the loss function. We then present experimental results on masked language modeling, unsupervised dependency parsing, and semantic textual similarity tasks. Experiment results confirm our hypothesis that a dependency graph is efficient for information propagation. The proposed model achieves competitive performance when compared to the transformer and other baselines.

In Chapter 5, we reviewed the history of Hierarchical Imitation and Reinforcement Learning (HIRL). We found that the temporal hierarchical structure used in HIRL is similar to the constituency tree structure. Thus, we introduced the Option-Controller Network (OCN) – a HIRL model with constituency inductive bias. A typical OCN includes a controller and a set of options. The controller focuses on higher-level planning, while the options provide lower-level skills. During execution, the controller starts by selecting an option to execute, then it will wait until the selected option report finish to select a second option. This mechanism is developed from Ordered Neurons inductive bias. In experiments, we perform behavior cloning from unstructured demonstrations coming from different tasks, and during the RL finetuning, we freeze the learned options and only re-initialize the controller. Experiment results show that the learned options can function as a plug-and-play module for other OCN models. While facing a new task, we can simply gather previously learned options and initialize a new controller, then the RL algorithm will optimize the controller to solve the new task.

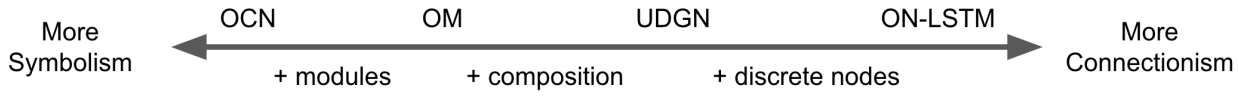


Fig. 22. The spectrum from connectionism to symbolism of our proposed models.

In Figure 22, we provide a spectrum of our models, arranged according to their level of discreteness. In general, a model that has a discrete inner structure and a symbolic-like pattern is considered more symbolic. On the contrary, a model that has no inner structure (fully connected) and no symbolic patterns (distributed representations) are considered as more connectionist. In our works, ON-LSTM is the most connectionist model, because it has the least discrete internal structures and no vector representations for non-terminal nodes in the tree structure. UDGN is less connectionist than ON-LSTM because it learns a more discrete internal structure and it has a vector representation for each node in the dependency graph. OM is more symbolic than UDGN, because it can converge to a discrete internal structure, has a vector representation for each node, and also models the compositional

function for these nodes. OCN is the most symbolic model in our work, because it has all the previous features (except that it models decomposition instead of composition), and it also models each task and subtask as an individual module, that a new agent or user can reuse them as simple meaningful symbols.

Altogether, we are excited about the progress that has been made in this field for the past few years and have been glad to be able to contribute. At the same time, we also believe that current progress is just the initial steps for this newly emerged domain. As we relentlessly argued through this thesis, the goal of this field is not just to induce the latent structure of natural language, but also to encourage neural network models to use it as the innate structure of reasoning. Based on this reason, we want to encourage researchers to evaluate their model on both unsupervised parsing performance and how the model performance in terms of generalization and robustness. In the rest of this chapter, we will discuss some important future milestones.

6.1. Future Directions

6.1.1. Emerging Discrete and Useful Structure

Syntactic inductive biases are first proposed as a guideline to develop unsupervised parsing models. For most of the current methods, the induced structure is greedily sampled from a distribution of all possible structures. Almost all methods focus on evaluating the coherence between the sampled structure and the gold structure given by expert annotators. The sharpness of the distribution is usually not taken into consideration. However, as we argued in the introduction, the other important role of syntactic inductive bias is to regularize the internal connection of neural network models. In this case, a smooth distribution could result in a weak or nonexistent regularization effect. For example, in UDGN, if the dependency distribution is close to a uniform distribution, then the latent structure of the model reduces to fully connected. In Ordered Memory, if the attention distribution at every time step is uniform, the model cannot compose information as a recursive neural network, thus failing to model the intrinsic composition function. In this case, it's less likely that the model can still generalize to Out-Of-Distribution datapoints. Similarly, in Option-Controller Network, learning reusable skills also require the structural decisions to be sharp. Hence, we would like to argue that the sharpness of distribution is at least as important as the correctness of induced latent structure for tasks that require strong generalization ability.

Encouraging a sharp distribution is also an important step towards combining symbolism and connectionism. The discrete structure given by a sharp distribution can be considered as the computation graph in a symbolic system. A sharp distribution can guarantee that information can only be combined or communicated through the induced structure. In a

tree-like structure, this effect can create information bottlenecks. So the model can learn the vector representation of a phrase or a sentence. This is similar to the symbolic system, that we can find meaningful intermediate results in the computation graph. These intermediate results contribute to the excellent interpretability of symbolic methods. In the Ordered Memory model, we can find these intermediate results stored in memory slots. For example, after processing a ListOps equation, each parenthesis in the equation can find its distributed representation stored in at least one memory slot. For natural language, we proposed ON-LSTM and UDGN to learn different types of latent structures. However, how we emerge sharp distributions and get interpretable intermediate results in a natural language task remains an open question.

On the other hand, we also want to argue that the quality of latent structure can be evaluated by multiple metrics. Unsupervised parsing performance is an important metric. It can provide us insights into what structure the model induced. Also, gold trees are just human annotations. They are not necessarily the ground-truth structure or the best structure for certain tasks [Dai et al., 2021]. For future works, we would like to explore other evaluation metrics that assess the quality of induced structure from a practical angle, for example, the systematic generalization ability or fine-tuning performance on downstream tasks.

6.1.2. Inducing Reusable Operators

Once a model could stably produce a discrete structure, a reasonable next step is to try to replace the single composition/communicate function with a set of operators. For example, in UDGN, we use the competitive mechanism to select different communication channels for each pair of tokens. In OCN, we have different options for different subtasks. The idea is inspired by the operators in the formal language and the syntactic functions in theoretical linguistics. In Logical Inference, we have operators like *or* and *and*. In dependency grammar, we have syntactic functions like *SUBJ* and *PRED*. One common feature of these operators is reusability. It means that the same operator can be used in many different contexts and still hold the same functionality. The only constraint is that inputs of the operator should share some common features.

Introducing a set of operators has two major advantages. The first one is that simpler operators usually have better reusability because they require less input information and are less sensitive to the context. For example, in OCN, an option that models only one subtask can be reused by any controller when the subtask is encountered. If the option models two different subtasks, the controller will have to pass a command vector to the option to disambiguate the subtask. This command vector could cause extra effort for the new controller to learn the protocol. Secondly, it's easier to diagnose and fix a bug. For

example, in OCN, it's possible to associate some particular failures with a specific component (an option or a controller). An easy fix would be simply reinitializing the failed component and fixing the rest of the model, then retraining the model on a specially designed dataset.

In our work, UDGN provides a potential solution to learn syntactic functions in a natural language setting. In an undirected dependency graph, UDGN uses a competitive mechanism to select the module for each edge to propagate information. But, how to clearly define and learn reusable operators in natural language tasks remains an open question.

6.1.3. Systematic Generalization

For language, systematic generalization requires a model to be able to reason about all valid sequences of tokens despite being trained on a very small subset of them. In a symbolic framework, systematic generalization is straightforward. Once the operators and grammar are well-defined, any inputs that satisfy the pre-defined grammar can be processed by the system. However, this setting also has great limitations in the real world. Firstly, natural language has very complicated grammar, and even a state-of-the-art parser could fail in many cases. Secondly, the operators used by natural language are complicated to be pre-defined. In this thesis, we try to solve the two problems with a data-driven approach, that is, given a pre-designed inductive bias, we want the model to induce the syntax and operators from the supervised or unsupervised training losses. We observe some successes on synthetic tasks, including ListOps, Logical Inference, and Craft. But reaching systematic generalization on natural language will still require a lot of future effort and creativity.

References

Steven Abney. Statistical methods and linguistics. 1996.

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Eneko Agirre, Daniel Cer, Mona Diab, and Aitor Gonzalez-Agirre. Semeval-2012 task 6: A pilot on semantic textual similarity. In ** SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 385–393, 2012.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. * sem 2013 shared task: Semantic textual similarity. In *Second joint conference on lexical and computational semantics (* SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*, pages 32–43, 2013.

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. Semeval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 81–91, 2014.

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, et al. Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, pages 252–263, 2015.

Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez Agirre, Rada Mihalcea, German Rigau Claramunt, and Janyce Wiebe. Semeval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In *SemEval-2016. 10th International Workshop on Semantic Evaluation; 2016 Jun 16-17; San Diego, CA. Stroudsburg (PA): ACL; 2016. p. 497-511.* ACL (Association for Computational Linguistics), 2016.

Wasi Uddin Ahmad, Nanyun Peng, and Kai-Wei Chang. Gate: Graph attention transformer encoder for cross-lingual relation and event extraction. *arXiv preprint arXiv:2010.03009*, 2020.

David Alvarez-Melis and Tommi S Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. 2016.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175. PMLR, 2017.

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

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned? *arXiv preprint arXiv:1811.12889*, 2018.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

Rens Bod. An all-subtrees approach to unsupervised parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 865–872. Association for Computational Linguistics, 2006.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Samuel R Bowman, Christopher Potts, and Christopher D Manning. Recursive neural networks can learn logical semantics. *arXiv preprint arXiv:1406.1827*, 2014.

Samuel R Bowman, Christopher D Manning, and Christopher Potts. Tree-structured composition in neural networks without tree-structured architectures. *arXiv preprint arXiv:1506.04834*, 2015.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*, 2016.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Glenn Carroll and Eugene Charniak. *Two experiments on learning probabilistic dependency grammars from corpora*. Department of Computer Science, Univ., 1992.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

Yuen Ren Chao. *Language and symbolic systems*, volume 260. Cambridge University Press Cambridge, 1968.

Eugene Charniak. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 124–131. Association for Computational Linguistics, 2001.

Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech & Language*, 14(4):283–332, 2000.

Stanley F Chen. Bayesian grammar induction for language modeling. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 228–235. Association for Computational Linguistics, 1995.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. Learning to compose task-specific tree structures. In *Proceedings of the 2018 Association for the Advancement of Artificial Intelligence (AAAI). and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, 2018.

Yoeng-Jin Chu and Tseng-Hong Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Shay B Cohen, Dipanjan Das, and Noah A Smith. Unsupervised structure prediction with non-parallel multilingual guidance. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 50–61. Association for Computational Linguistics, 2011.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017.

Junqi Dai, Hang Yan, Tianxiang Sun, Pengfei Liu, and Xipeng Qiu. Does syntax matter? a strong baseline for aspect-based sentiment analysis with roberta. *arXiv preprint arXiv:2104.04986*, 2021.

Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, page 14, 1992.

Hal Daumé III. Unsupervised search-based structured prediction. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 209–216, 2009.

Hiroyuki Deguchi, Akihiro Tamura, and Takashi Ninomiya. Dependency-based self-attention for transformer nmt. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 239–246, 2019.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Misha Denil, Sergio Gómez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas. Programmable agents. *arXiv preprint arXiv:1706.06383*, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

Andrew Drozdov, Pat Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. Unsupervised latent tree induction with deep inside-outside recursive autoencoders. *arXiv preprint arXiv:1904.02142*, 2019.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, 2016.

Jack Edmonds. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240, 1967.

Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499, 1996.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

Roy Fox, Richard Shin, Sanjay Krishnan, Ken Goldberg, Dawn Song, and Ion Stoica. Parametrized hierarchical procedures for neural programming. *ICLR*, 2018.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages

1019–1027, 2016.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *arXiv e-prints*, pages arXiv–2104, 2021.

Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

Daniel Gildea. Dependencies vs. constituents for tree-based alignment. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 214–221, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://aclanthology.org/W04-3228`.

Jennifer Gillenwater, Kuzman Ganchev, João Graça, Fernando Pereira, and Ben Taskar. Sparsity in dependency grammar induction. *ACL 2010*, page 194, 2010.

Anirudh Goyal, Aniket Didolkar, Alex Lamb, Kartikeya Badola, Nan Rosemary Ke, Nasim Rahaman, Jonathan Binas, Charles Blundell, Michael Mozer, and Yoshua Bengio. Coordination among neural modules through a shared global workspace. *arXiv preprint arXiv:2103.01197*, 2021.

Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1828–1836, 2015.

Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

Serhii Havrylov, Germán Kruszewski, and Armand Joulin. Cooperative learning of disjoint syntax and semantics. *In Proc. of NAACL-HLT*, 2019.

John A Hawkins. *Efficiency and complexity in grammars*. Oxford University Press on Demand, 2004.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. Unsupervised learning of syntactic structure with invertible neural projections. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1292–1302, 2018.

William P Headden III, Mark Johnson, and David McClosky. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of human language technologies: the 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 101–109, 2009.

Peter Henderson, Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau, and Doina Precup. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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

Phu Mon Htut, Kyunghyun Cho, and Samuel R Bowman. Grammar induction with neural language models: An unusual replication. *arXiv preprint arXiv:1808.10000*, 2018.

Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R Bowman. Do attention heads in bert track syntactic dependencies? *arXiv preprint arXiv:1911.12246*, 2019.

Jennifer Hu, Jon Gauthier, Peng Qian, Ethan Wilcox, and Roger P Levy. A systematic assessment of syntactic generalization in neural language models. *arXiv preprint arXiv:2005.03692*, 2020.

Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

Athul Paul Jacob, Zhouhan Lin, Alessandro Sordoni, and Yoshua Bengio. Learning hierarchical structures on-the-fly with a recurrent-recursive model for sequences. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 154–158, 2018.

Yiding Jiang, Shixiang Gu, Kevin Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *arXiv preprint arXiv:1906.07343*, 2019.

Yong Jiang, Wenjuan Han, Kewei Tu, et al. Unsupervised neural dependency parsing. Association for Computational Linguistics (ACL), 2016.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pages 190–198, 2015.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pages 1094–1099. Citeseer, 1993.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

Yoon Kim, Chris Dyer, and Alexander M Rush. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, 2019a.

Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. Unsupervised recurrent neural network grammars. *arXiv preprint arXiv:1904.03746*, 2019b.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, pages 3418–3428. PMLR, 2019.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Dan Klein and Christopher D Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 128–135. Association for Computational Linguistics, 2002.

Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

Dan Klein and Christopher D Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL-04)*, pages 478–485, 2004.

Dan Klein and Christopher D Manning. Natural language grammar induction with a generative constituent-context model. *Pattern recognition*, 38(9):1407–1419, 2005.

Donald E Knuth. On the translation of languages from left to right. *Information and control*, 8(6):607–639, 1965.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Sanjay Krishnan, Roy Fox, Ion Stoica, and Ken Goldberg. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning*, pages 418–437. PMLR, 2017.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. Lstms can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1426–1436, 2018.

Adhiguna Kuncoro, Lingpeng Kong, Daniel Fried, Dani Yogatama, Laura Rimell, Chris Dyer, and Phil Blunsom. Syntactic structure distillation pretraining for bidirectional encoders. *arXiv preprint arXiv:2005.13482*, 2020.

Alex Lamb, Di He, Anirudh Goyal, Guolin Ke, Chien-Feng Liao, Mirco Ravanelli, and Yoshua Bengio. Transformers with competitive ensembles of independent mechanisms. *arXiv preprint arXiv:2103.00336*, 2021.

Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé. Hierarchical imitation and reinforcement learning. In *International Conference on Machine Learning*, pages 2917–2926. PMLR, 2018.

Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical reinforcement learning with hindsight. *arXiv preprint arXiv:1805.08180*, 2018.

Tsungnan Lin, Bill G Horne, Peter Tino, and C Lee Giles. Learning long-term dependencies is not as difficult with narx recurrent neural networks. Technical report, 1998.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *arXiv preprint arXiv:1611.01368*, 2016.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Joao Loula, Marco Baroni, and Brenden M Lake. Rearranging the familiar: Testing compositional generalization in recurrent networks. *arXiv preprint arXiv:1807.07545*, 2018.

Yuchen Lu, Yikang Shen, Siyuan Zhou, Aaron Courville, Joshua B. Tenenbaum, and Chuang Gan. Learning task decomposition with ordered memory policy network. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=vcopnwZ7bC`.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132. PMLR, 2020.

Kanika Madan, Rosemary Nan Ke, Anirudh Goyal, Bernhard Bernhard Schölkopf, and Yoshua Bengio. Fast and slow learning of recurrent independent mechanisms. *arXiv preprint arXiv:2105.08710*, 2021.

Jean Maillard, Stephen Clark, and Dani Yogatama. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *arXiv preprint arXiv:1705.09189*, 2017.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics, 1994.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, Roberto Zamparelli, et al. A sick cure for the evaluation of compositional distributional semantic models. In *Lrec*, pages 216–223. Reykjavik, 2014.

Rebecca Marvin and Tal Linzen. Targeted syntactic evaluation of language models. *arXiv preprint arXiv:1808.09031*, 2018.

David McClosky, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 337–344, 2006.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182*, 2017.

Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

Tom M Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research ..., 1980.

Michael C Mozer and Sreerupa Das. A connectionist symbol manipulator that discovers the structure of context-free languages. In *Advances in neural information processing systems*, pages 863–870, 1993.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3307–3317, 2018.

Nikita Nangia and Samuel R Bowman. Listops: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*, 2018.

Yutaro Omote, Akihiro Tamura, and Takashi Ninomiya. Dependency-based relative positional encoding for transformer nmt. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 854–861, 2019.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049, 1998.

Alexander Pashevich, Danijar Hafner, James Davidson, Rahul Sukthankar, and Cordelia Schmid. Modulated policy hierarchies. *arXiv preprint arXiv:1812.00025*, 2018.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2): 77–105, 1990.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 157–163, 2017.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pages 1746–1754, 2014.

Brian Roark. Probabilistic top-down parsing and language modeling. *Computational linguistics*, 27(2):249–276, 2001.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80, 2008.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

Jürgen Schmidhuber. Neural sequence chunkers. 1991.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Marcel Paul Schützenberger. On context-free languages and push-down automata. *Information and control*, 6(3):246–264, 1963.

Mohit Sharma, Arjun Sharma, Nicholas Rhinehart, and Kris M Kitani. Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information. In *International Conference on Learning Representations*, 2018.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

Yikang Shen, Zhouhan Lin, Chin-Wei Huang, and Aaron Courville. Neural language modeling by jointly learning syntax and lexicon. *arXiv preprint arXiv:1711.02013*, 2017.

Yikang Shen, Zhouhan Lin, Chin-wei Huang, and Aaron Courville. Neural language modeling by jointly learning syntax and lexicon. In *International Conference on Learning Representations*, 2018a.

Yikang Shen, Zhouhan Lin, Athul Paul Jacob, Alessandro Sordoni, Aaron Courville, and Yoshua Bengio. Straight to the tree: Constituency parsing with neural syntactic distance. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1180, 2018b.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. *arXiv preprint arXiv:1810.09536*, 2018c.

Yikang Shen, Yi Tay, Che Zheng, Dara Bahri, Donald Metzler, and Aaron Courville. Structformer: Joint unsupervised induction of dependency and constituency structure from masked language modeling. *arXiv preprint arXiv:2012.00857*, 2020.

Haoyue Shi, Hao Zhou, Jiaze Chen, and Lei Li. On tree-based neural sentence modeling. *arXiv preprint arXiv:1808.09644*, 2018.

Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. Taco: Learning task decomposition via temporal alignment for control. In *International Conference on Machine Learning*, pages 4654–4663, 2018.

Stuart M Shieber. Sentence disambiguation by a shift-reduce parsing technique. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 113–118. Association for Computational Linguistics, 1983.

Noah A Smith and Jason Eisner. Guiding unsupervised grammar induction using contrastive estimation. In *Proc. of IJCAI Workshop on Grammatical Inference Applications*, pages 73–82, 2005.

Richard Socher, Christopher D Manning, and Andrew Y Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, volume 2010, pages 1–9, 2010.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

Alec Solway, Carlos Diuk, Natalia Córdova, Debbie Yee, Andrew G Barto, Yael Niv, and Matthew M Botvinick. Optimal behavioral hierarchy. *PLOS Comput Biol*, 10(8):e1003779, 2014.

Valentin I Spitkovsky, Hiyan Alshawi, Angel Chang, and Dan Jurafsky. Unsupervised dependency parsing without gold part-of-speech tags. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1281–1290, 2011.

Valentin I Spitkovsky, Daniel Jurafsky, and Hiyan Alshawi. Breaking out of local optima with count transforms and model recombination: A study in grammar induction. 2013.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. Linguistically-informed self-attention for semantic role labeling. *arXiv preprint arXiv:1804.08199*, 2018.

Guo-Zheng Sun, C Lee Giles, Hsing-Hen Chen, and Yee-Chun Lee. The neural network pushdown automaton: Model, stack and learning simulations. *arXiv preprint arXiv:1711.05738*, 2017.

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

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112 (1-2):181–211, 1999.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Hao Tang, Donghong Ji, Chenliang Li, and Qiji Zhou. Dependency graph enhanced dual-transformer structure for aspect-based sentiment classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6578–6588, 2020.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Ke Tran, Arianna Bisazza, and Christof Monz. The importance of being recurrent for modeling hierarchical structure. *arXiv preprint arXiv:1803.03585*, 2018.

Kewei Tu and Vasant Honavar. Unambiguity regularization for unsupervised learning of probabilistic grammars. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1324–1334, 2012.

Zhuowen Tu, Xiangrong Chen, Alan L Yuille, and Song-Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of computer vision*, 63(2): 113–140, 2005.

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. Udapter: Language adaptation for truly universal dependency parsing. *arXiv preprint arXiv:2004.14327*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Yaushian Wang, Hung-Yi Lee, and Yun-Nung Chen. Tree transformer: Integrating tree structures into self-attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1060–1070, 2019.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

Adina Williams, Andrew Drozdov*, and Samuel R Bowman. Do latent tree learning models identify meaningful structure in sentences? *Transactions of the Association of Computational Linguistics*, 6:253–267, 2018.

Jiacheng Xu, Danlu Chen, Xipeng Qiu, and Xuangjing Huang. Cached long short-term memory neural networks for document-level sentiment classification. *arXiv preprint arXiv:1610.04989*, 2016.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. *arXiv preprint arXiv:1611.09100*, 2016.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. Memory architectures in recurrent neural network language models. 2018.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Daniel Zeman and Philip Resnik. Cross-language parser adaptation between related languages. In *Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages*, 2008.

Zheng Zeng, Rodney M Goodman, and Padhraic Smyth. Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, 5(2):320–330, 1994.

Quanshi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *arXiv preprint arXiv:1802.00614*, 2018.

Xingxing Zhang, Liang Lu, and Mirella Lapata. Top-down tree long short-term memory networks. *arXiv preprint arXiv:1511.00060*, 2015.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. Dependency parsing as head selection. *arXiv preprint arXiv:1606.01280*, 2016.

Ganbin Zhou, Ping Luo, Rongyu Cao, Yijun Xiao, Fen Lin, Bo Chen, and Qing He. Generative neural machine for tree structures. *CoRR*, 2017.

Hao Zhu, Yonatan Bisk, and Graham Neubig. The return of lexical dependencies: Neural lexicalized pcfgs. *Transactions of the Association for Computational Linguistics*, 8:647–661, 2020.

Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, pages 1604–1612, 2015.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.