LINGUISTIC KNOWLEDGE FOR NEURAL LANGUAGE GENERATION AND MACHINE TRANSLATION

AUSTIN MATTHEWS

Thesis Proposal

Language Technologies Institute
Carnegie Mellon University

March 2018

ABSTRACT

Recurrent neural networks (RNNs) are exceptionally good models of distributions over natural language sentences, and they are deployed in a wide range of applications that require the generation of natural language outputs. However, RNNs are general-purpose function learners that, given sufficient capacity, are capable of representing any distribution, whereas the space of possible natural languages is narrowly constrained. Linguistic theory has been concerned with characterizing these constraints, with a particular eye toward explaining the uniformity with which children acquire their first languages, despite receiving relatively little linguistic input. This thesis uses insights from linguistic theory to inform the neural architectures and generation processes used to model natural language, seeking models that make more effective use of limited amounts of training data. Since linguistic theories are incomplete, a central goal is developing models that are able to exploit explicit linguistic knowledge while still retaining the generality and flexibility of neural network models they augment.

In particular, this thesis focuses on modeling word formation using linguistic knowledge about morphological processes in the form of finite state transducers and syntactic theories that construct sequences of words as the outputs of hierarchical branching processes. We present a model capable of conditioning and emitting words at several levels of granularity, including the raw word-, character-, and morpheme-levels. We further present a model that generates sentences using hierarchical structure, jointly learning language modeling and parsing. We evaluate each model on several NLP tasks, and combine them together and condition on a foreign-language input sentence to create a linguistically-aware neural machine translation system that excels at translating into traditionally difficult languages with complex word formation paradigms and very different syntax than English.

The major contributions of this thesis are as follows: (i) a morphologically-aware open-vocabulary language model (ii) a dependency-based language model for generation and parsing (iii) a syntax-aware attention mechanism for machine translation (iv) an MT system incorporating all of the above.

CONTENTS

1 INTRODUCTION

Language generation, the task of having a computer automatically *output* text in a natural language (such as English) is a crucial step towards natural human–computer interaction. It is a essential part of many NLP tasks including summarization, image captioning, dialogue systems, and translation. Most existing methods for language generation treat text as a linear sequence of atomic words. Such methods model a sentence (or document, etc.) as a string of words that is generated stochastically from left to right and they do not attempt to understand or break down words into any smaller pieces.

Humans, on the other hand, seem to generate sentences in a much more hierarchical manner. For example, a typical English sentence might be composed of a subject, a verb phrase, and optionally a direct object. A subject may itself be composed into e.g. a determiner, a series of zero or more adjectives, and a noun, among other possibilities. The verb phrase may similarly be composed of one or more auxiliary verbs, a main verb, and zero or more adverbs. This hierarchical decomposition is central to the infinite expressive power of human language, and our ability as humans to both generate and understand novel sentences, the content of which we have never heard before. Linguists call the study of these hierarchical structures *syntax*, and the structures themselves *syntax trees*.

The leaves of a syntax tree are typically words, yet it is often possible to further decompose words into smaller pieces called *morphemes*. For example, the word "denuclearization" can be broken down into the morpheme sequence de+nuclear+ize+ation. The study of such analyses of sub-word units is called *morphology*. Understanding these smaller components of words allows humans to create and understand entirely new words on the fly. For computer models it also allows for much greater statistical sharing. For example, a model should not have to separately learn that "eat", "eats", "eating", "ate", and "eaten" all have something to do with food, nor that (almost) all words that end in *-ation* are nouns. If a model treats words as non-decomposable fixed entities such abstractions are not possible, but morphological-aware models allow words to share knowledge learned about their morphemes. Furthermore, such models have an *open vocabulary* – they can understand and emit rare words, never before seen by the model, like "unacrimoniousness" or "drinkability", in much the same way that a human can.

Most applications of text generation are *conditional*, indicating that they receive ("condition on") some existing information in order to choose what sentence to output. For example, a dialogue system might condition on what the interlocutors said on the previous turn(s), an image captioning system should obviously look at the image whose caption it is to generate, and a machine translation system should pay attention to the input ("source") sentence when generating its translation. Neural models make this type of external conditioning very easy. At each time step, instead of looking only at the generator's current state (typically a vector which encodes all the information it has previously generated) we also look at a vector embedding of the input data. By modeling the next generation event probability as a function of these vectors both kinds of context are used.

But why should our models emulate humans' methods of language generation? Our primary motivation is one of sample efficiency. RNNs are great general function learners and, given infinite data, a simple left-to-right RNN model with atomic words could, in theory, learn to produce fluent language. Human children, however, are able to learn their native language fluently with just tens of millions of words of input (Hart and Risley, 2003). Large quantities of monolingual text data are available in many languages (the 2018 WMT

data contains over 60 **billion** words of English), but no models yet exist that can understand and generate language at the level of humans. Furthermore, many problems reliant on text generation have much more impoverished data sets. For example, the 2018 WMT English–Turkish translation task bitext provides only 4.4 million words of English data, necessitating either much more sample efficient learning methods, methods to transfer knowledge from systems trained on monolingual data, or both. Linguistic theory gives us insights into the processes that underpin human languages, and incorporating this knowledge into our models can help restrict the hypothesis space, leading to faster learning and more human-like output.

In this work we will explore methods to incorporate morphological and syntactic knowledge into neural models of text generation. We will then combine these methods and condition on a source sentence (as either a linear string or as a parse tree) to create a linguistically aware neural machine translation system.

In Chapter 2 we explore a method to incorporate morphological knowledge in a text generation system. In our system, we generate text left-to-right, as in most RNN-based models. At each step, we choose the next word conditioned on all the previously generated words (the *context*). Unlike previous work, however, We create embeddings for each word in the conditioning context by combining word- morpheme- and character-level representations instead of relying on embeddings of words as atomic units. We use an LSTM over the context to generate a single context vector, from which we will predict the next word. The model may then choose to generate the next word via one of three methods: as a single atomic word, as a sequence of morphemes, or as a sequence of characters. We marginalize the probability of generating a given word under each of these three methods to get a single final probability for each candidate word. The resulting model has the learnability advantages of being morphologically aware, and is also *open vocabulary* – it is able to produce novel word forms that were not present in its training data. These theoretical advantages also show empirical improvements in language model perplexity on three morphologically rich languages that are traditionally difficult for standard language models, as well as improvements in English-to-Turkish statistical machine translation.

In Chapter 3 we explore a method to incorporate syntactic knowledge into a language model. Previous work (Dyer et al., 2016) has shown that constituency structures help language modeling and parsing performance and that the resulting model also functions as a high-quality parser. We explore a complementary question: does dependency-based syntax similarly help language model performance? We operationalize the process of building a parse tree, converting it into a series of actions. These actions affect a stack of partially built dependency structures, which in the end will contain exactly one item: a fully constructed parse tree for the output sentence. At each time step our model examines the current stack and predicts the next action to take. We propose two models that build trees using different *construction orders*. The first builds the tree top-down, starting from the root and ending with the terminals. The second constructs the tree bottom-up, starting with the terminals and iteratively building tree structure atop them. These two models imply different hierarchical biases on the learner. Information that may be local in the partial tree structures built by one model may be farther away, and thus more difficult for the neural network to exploit, than in structures built by the other the other. We find that the two resulting models both exhibit competitive parsing performance but do not outperform baseline language models. Furthermore, we find that despite the wildly different biases, the two models perform very similarly, implying that they have learned different ways of capturing similar information.

In Chapter 4 we seek to incorporate the morphological gains from Chapter 2, the syntax gains from Dyer et al. (2016), and two other syntax-based techniques into a neural machine translation system. The first is the use of TreeLSTMs (Tai et al., 2015) to embed a source syntax tree, rather than relying solely on the flat text representation. The second is a novel

attention mechanism that encourages the model to translate syntactic constituents in their entirety before jumping to another part of the sentence. These additions mirror extensions to traditional statistical machine translation, including *quasi-synchronous* tree-to-tree models, reordering models, and *factored models*. These were principled extensions to MT systems but in symbolic systems they were brittle and often involved hard combinator search problems. By incorporating them into a neural system we both simplify them and increase their power. We will explore the performance of this linguistically-aware system on translation into three morphologically rich languages and perform an ablation to uncover which particular modules are most responsible for translation quality gains.

These three chapters contribute an overall theme and lead to the following thesis statement: **Explicit linguistic structure can be used to design neural architectures that learn more rapidly and make better generalizations than linguistically naïve architectures.**

2 NEURAL MORPHOLOGY FOR OPEN-VOCABULARY LANGUAGE MODELS

Language modeling of morphologically rich languages is particularly challenging due to the vast set of potential word forms and the sparsity with which they appear in corpora. Traditional *closed vocabulary* models treat words as atomic units and as such they are unable to produce word forms unseen in training data or to generalize from sub-word patterns found in data.

The most straightforward solution is to treat language as a sequence of characters (Sutskever et al., 2011). However, models that operate at two levels—a character level and a word level—have better performance (Chung et al., 2016; Kawakami et al., 2017). Another solution is to use morphological information from hand-crafted analyzers, which has shown benefits in non-neural models (Chahuneau et al., 2013). In this chapter, we present a model that combines these approaches in a fully neural framework [1].

Our model incorporates explicit morphological knowledge (e.g. from a finite-state morphological analyzer/generator) into a neural language model, combining it with existing word- and character-level modeling techniques, in order to create a model capable of successfully modeling morphologically complex languages. In particular, our model achieves three desirable properties.

First, it conditions on all available (intra-sentential) context, allowing it, in principle, to capture long-range dependencies, such as that the verb agreement between "students" and "are" in the sentence "The students who studied the hardest are getting the highest grades". While traditional n-gram based language models lack this property, RNN-based language models fulfill it.

Second, it explicitly captures morphological variation, allowing sharing of information between variants of the same word. This allows faster, smoother training as well as improved predictive generalization. For example, if the model sees the phrase "gorped the ball" in data, it is able to infer that "gorping the ball" is also likely to be valid. Similarly, the model is capable of understanding that morphological consistency within noun phrases is important. For example in Russian, one might say *malen'kaya chërniya koshka* ("small black cat", nominative), or *malen'kuyu chërniyu koshku* (accusative), but *malen'kiy chërnuyu koshke* (mixing nominative, accusative and dative) would have much lower probability.

Third, the language model seamlessly handles out of vocabulary items and their morphological variants. For example, even if the word *Obama* was never seen in a Russian corpus, we expect *Ya dal eto prezidentu Obame* ("I gave it to president Obama") to have higher probability using the dative *Obame* than *Ya dal eto prezidentu Obama*, which uses the nominative *Obama*. The model can also learn to decline proper nouns, including OOVs. Here it can recognize that *dal* ("gave") requires a dative, and that nouns ending with "a" generally do not meet that requirement while nouns ending with "e" do.

In order to capture these properties, our model combines two pieces: an alternative embedding module that uses sub-word information such as character and morpheme-level information, and a generation module that allows us to output words at the word, morpheme, or character-level. The embedding module allows for the model to share information between morphological variants of surface forms, and produce sensible word embeddings for tokens never seen during training. The generation model allows us to emit tokens never seen during training, either by combining a lemma and a sequence of affixes to create a

1 This chapter draws on material being published in Matthews et al. (2018), to be presented at NAACL 2018.

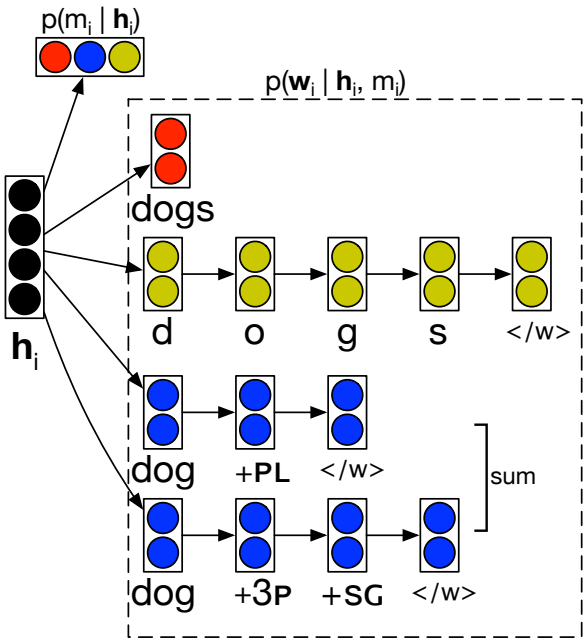


Figure 1: We allow the model to generate an output word at the word, morpheme, or character level, and marginalize over these three options to find the total probability of a word.

novel surface form, or by directly spelling out the desired word character by character. We then demonstrate the effectiveness both intrinsically, showing reduced perplexity on several morphologically rich languages, and extrinsically on machine translation and morphological disambiguation tasks.

MULTI-LEVEL RNNLMS

Recurrent neural network language models are composed of three parts: (a) an encoder, which turns a context word into a vector, (b) a recurrent backbone that turns a sequence of word vectors that represent the ordered sequence of context vectors into a single vector, and (c) a generator, which assigns a probability to each word that could follow the given context.

RNNLMs often use the same process for (a) and (c), but there is no reason why these processes cannot be decoupled. For example, Kim et al. (2016) and Ling et al. (2015) compose character-level representations for their word encoder, but generate words using a softmax whose probabilities rely on inner products between the current context vector and type-specific word embeddings.

In our model both the word generator (§2.1.1) and the word encoder (§2.1.2) compute representations that leverage three different views of words: frequent words have their own parameters, words that can be analyzed/generated by an analyzer are represented in terms of sequences of abstract morphemes, and all words are represented as a sequence of characters.

Word generation mixture model

In typical RNNLMs the probability of the ith word in a sentence, w_i given the preceding words is computed by using an RNN to encode the context followed by a softmax:

$$\begin{aligned} p(w_i \mid \boldsymbol{w}_{<i}) &= p(w_i \mid \mathbf{h}_i = \varphi_{\text{RNN}}(w_1, \ldots, w_{i-1})) \\ &= \text{softmax}(\mathbf{W}\mathbf{h}_i + \mathbf{b}) \end{aligned}$$

where φ_{RNN} is an RNN that reads a sequence of words and returns a fixed sized vector encoding, $\mathbf{W}$ is a weight matrix, and $\mathbf{b}$ is a bias.

In this work, we will use a mixture model over M different models for generating words in place of the single softmax over words (Miyamoto and Cho, 2016; Neubig and Dyer, 2016):

$$\begin{aligned} p(w_i \mid \mathbf{h}_i) &= \sum_{m_i=1}^{M} p(w_i, m_i \mid \mathbf{h}_i) \\ &= \sum_{m_i=1}^{M} p(m_i \mid \mathbf{h}_i) p(w_i \mid \mathbf{h}_i, m_i), \end{aligned}$$

where $m_i \in [1, M]$ indicates the model used to generate word w_i. To ensure tractability for training and inference, we assume that m_i is conditionally independent of all $\boldsymbol{m}_{<i}$, given the sequence of word forms $\boldsymbol{w}_{<i}$.

We use three ($M = 3$) component models: (1) directly sampling a word from a finite vocabulary ($m_i = \text{WORD}$), (2) generating a word as a sequence of characters ($m_i = \text{CHARS}$), and (3) generating as a sequence of (abstract) morphemes which are then stitched together using a hand-written morphological transducer that maps from abstract morpheme sequences to surface forms ($m_i = \text{MORPHS}$). Figure 1 illustrates the model components, and we describe in more detail here:

WORD GENERATOR. Select a word by directly sampling from a multinomial distribution over surface form words. Here the vocabulary is the $|V_w|$ most common full-form words seen during training. All less frequent words are assigned zero probability by this model, and must be generated by one of the remaining models.

CHARACTER SEQUENCE GENERATOR. Generate a word as a sequence of characters. Each character is predicted conditioned on the LM hidden state $\mathbf{h}_i$ and the partial word generated so far, encoded with an RNN. The product of these per-character probabilities is the total probability assigned to a full word form.

MORPHEME SEQUENCE GENERATOR. Similarly to the character sequence generator, we can generate a word as a sequence of morphemes. We first generate a root r, followed by a sequence of affixes $a_1, a_2, \ldots$. For example the word "devours" might be generated as devour+3P+SG.

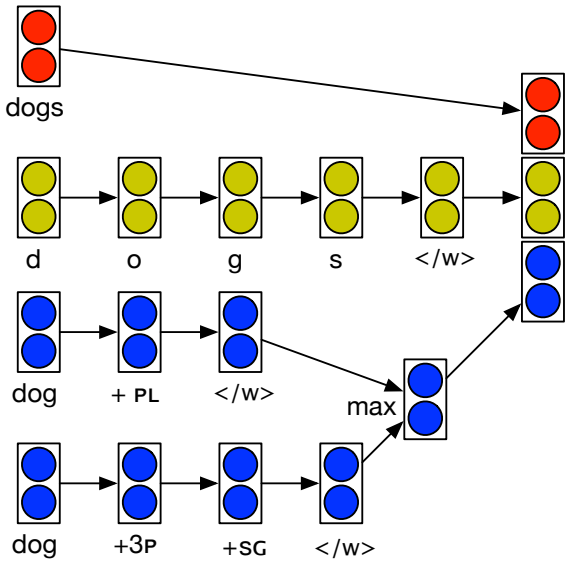


Figure 2: We concatenate word- morpheme- and character-level vectors to build a better input vector for our RNNLM.

Since multiple sequences of abstract morphemes may in general give rise to a single output form,[2] we marginalize these, i.e.,

$$p(w_i \mid \mathbf{h}_i, m_i = \textsc{morphs}) = \sum_{\boldsymbol{a}_i \in \{\boldsymbol{a} \mid \textsc{gen}(\boldsymbol{a}) = w_i\}} p_{morphs}(\boldsymbol{a}_i \mid \mathbf{h}_i).$$

where $\textsc{gen}(\boldsymbol{a})$ gives the surface word form produced from the morpheme sequence $\boldsymbol{a}$.

Due to the model's ability to produce output at the character level, it is able to produce any output sequence at all within the language's alphabet. This is critical as it allows the model to generate unknown words, such as novel names and declensions thereof. Furthermore, the morphological level facilitates the model's generation of word forms whose lemmas may be known, but whose surface form was nevertheless unattested in the training data. Finally the word-level generation model handles generating words that the model has seen many times during training.

Morphologically aware context vectors

Word vectors are typically learned with a single, independent vector for each word type. This independence means, for example, that the vectors for the word "apple" and the word "apples" are completely unrelated. Seeing the word "apple" gives no information at all about the word "apples".

Ideally we would like to share information between such related words. Nevertheless, sometimes words have idiomatic usage, so we'd like not to tie them together too tightly.

We accomplish this by again using three different types of word vectors for each word in the vocabulary. The first is a standard per-type word vector. The second is the output of a character-level RNN using Long Short-Term Memory (LSTM) units (Hochreiter and

2 In general analyzers encode many-to-many relations, but our model assumes that any sequence of morphs in the underlying language generates a single surface form. This is generally true, although free spelling variants of a morph (e.g., American *-ize* vs. British *-ise* as well as alternative realizations like *shined/shone* and *learned/learnt*) violate this assumption.

Schmidhuber, 1997). The third is the output of a morphology-level LSTM over a lemma and a sequence of affixes, as output by a morphological analyzer.

Typically language models first generate a word w_i given some (initially empty) prior context c_{i-1}, and then that word is combined with the context to generate a new context c_i that includes the new word. Since we have just used one or more of our three modes to generate each word, intuitively we would like to use the same mode(s) to generate the embedding used to progress the context.

Unfortunately, doing so introduces dependencies among the latent variables $p(\text{mode} \mid c_i)$ in our model, making exact inference intractable. As such, we instead drop the dependency on how a word was generated and instead represent the word at all three levels, regardless of the mode(s) actually used to generate it, and combine them by concatenating the three representations. A visual representation of the embedding process is shown in Figure 2.

Additionally, should a morphological analyzer produce more than one valid analysis for a surface form, we independently produce embeddings for each candidate analysis, and combine them using a per-dimension maximum operation. Mathematically, the ith dimension of the morphological embedding e_m is given by

$$e_{mi} = \max_j e_{a_j i}$$

where e_{a_j} is the embedding of the jth possible analysis, as computed by the LSTM over the lemma and its sequence of affixes.

The intuition behind the use of all analyses plus a pooling operation can be seen by observing the case of the word "does", which could be *do+3-person+singular* or *doe+plural*. If this word appears after the word "he", what we care about more is whether "does" could feasibly be a third person singular verb, thus agreeing with the subject. The max-pooling operation captures this intuition by ensuring that if a feature is active for one of these two analyses, it will also be active in the pooled representation. This procedure affords us the capability to efficiently marginalize over all three possible values of the latent variable at each step, and compute the full marginal of the word w_i given the context c_{i-1} during generation.

This formulation allows words with the same stem to share vector information through the character or morphological embeddings, but still affords each word the ability to capture idiomatic usages of individual words though the word embeddings. Furthermore, it allows a language model to explicitly capture morphological information, for example that third person singular subjects should co-occur with third person singular verbs. Finally, the character-level segment of the embedding allows the model to at least attempt to build sensible embeddings for completely unknown words. For example in Russian where names can decline with case this formulation allows the model to know that *Obame* is probably dative, even if it's an OOV at the word level, and even if the morphological analyzer is unable to produce any valid analyses due to the root being previously unseen as well.

We combine our three-layer input vectors, our factored output model, and a standard LSTM backbone to create a morphologically-enabled RNNLM that, as we will see in the next section, performs well on morphologically complex languages.

INTRINSIC EVALUATION

We demonstrate the effectiveness of our model by experimenting on three languages: Finnish, Turkish, and Russian. We use varying amounts of News Commentary data provided by the Workshop on Machine Translation (WMT) for each of the three languages. Statistics of our experimental corpora can be found in Table 1.

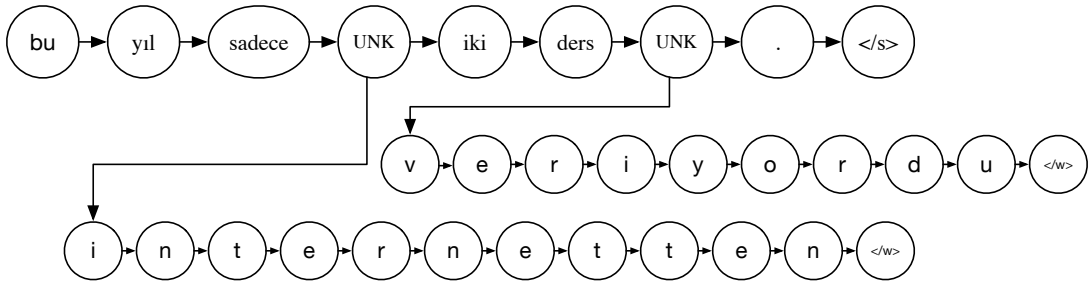


Figure 3: An example of our 4-gram Kneser-Ney baseline that backs off to an additional character-level model for out of vocabulary words generating a Turkish sentence that contains two out-of-vocabulary items.

Each data set was pre-processed by UNKing all but the top ≈20k words and lemmas by frequency. No characters or affixes were UNKed. This step is not strictly required—our model is, after all, capable of producing arbitrary words— but it speeds up training immensely by reducing the size of the word and lemma softmaxes. Since the morphology and/or character-level embeddings can still capture information about the original forms of these words, the degradation in modeling performance is minimal.

For morphological analysis we use Omorfi[3] for Finnish, the analyzer of Oflazer (1994) for Turkish, and PyMorphy[4] for Russian.

Baseline Models

Since models are not accurately comparable unless they share output vocabularies, our baselines must also allow for the generation of arbitrary word forms, including out-of-vocabulary items. We compare to three such models: an improved Kneser-Ney (Kneser and Ney, 1995) 4-gram baseline, with an additional character-level backoff model for OOVs, an RNNLM with character-level backoff, and a pure character-based RNN language model (Sutskever et al., 2011).

Since Kneser-Ney language models (and other count-based models) are typically word-level and do not model out-of-vocabulary items, we employ a two-level approach with separate Kneser-Ney models at the word and character levels. We train the word-level model after UNKing low frequency words, and we train the character-level model on the same list of low frequency words. Now when we want to predict a word w_i given some context c we can use the word-level model to directly predict $p(w_i|c)$ unless w_i is an out-of-vocabulary item. In that case we model $p(w_i \mid c)$ as

$$p(w_i \mid c) = p(\text{UNK} \mid c) \cdot p(w_i \mid \text{UNK})$$

where the first factor is the probability of the word-level model emitting UNK, and the second is the probability of the actual out-of-vocabulary word form under the character-level model. See Figure 3.

Secondly we compare to a similar hybrid RNN model that first predicts the word-level probability for each word, and if it predicted UNK then also predicts a sequence of characters using a separate network. This model uses 256-dimensional character and word embeddings, and a 1024-dimensional recurrent hidden layer.

Finally we also compare to a standard RNN language model trained purely on the character level. For this baseline we also use 256-dimensional character embeddings and a 1024-dimensional recurrent hidden layer.

3 https://github.com/flammie/omorfi
4 https://github.com/kmike/pymorphy

	Finnish	Russian	Turkish
Train Sents	2.1M	1.1M	188K
Train Words	38M	26M	3.9M
Dev Sents	1K	38K	1K
Dev Words	16K	705K	16K
Test Sents	500	91K	3K
Test Words	7.6K	1.6M	51K
Word Vocab	20K	21K	42K
Lemma Vocab	20K	20K	13K
Affix Vocab	140	34	180
Char Vocab	229	150	80

Table 1: Details of our data sets. Each cell indicates the number of sentences and the number of words in each set.

Multi-factored Models

For our model we use 128-dimensional word and root embeddings, 64-dimensional affix and character embeddings, 128-dimensional word-internal recurrent hidden layers for characters and morphemes, and a 256-dimensional recurrent hidden layer for the main inter-word LSTM.

The network is trained to stochastically optimize the log likelihood of the training data using Adam (Kingma and Ba, 2014). After each 10k training examples (Finnish, Turkish) or 100k training examples (Russian) we evaluate the model on a development set. (We evaluate less frequently on Russian since the dev set is much larger.) If the perplexity on the development set represents a new best, we save the current model to disk, thereby mitigating overfitting via early stopping. No other regularization is used.

For each language we run four variants of our model. In order to preserve the ability to model and emit any word in the modelled language, it is essential that we keep the character-level part of our model intact. The morpheme- and word-level models, however, may be removed without compromising the generality of the model. As such, we present our model using only character-level input and outputs (C), using character- and morpheme-level inputs and outputs (CM), using character- and word-level inputs, but no morphology (CW), and using all three levels as per the full model (CMW).

Results and Analysis

Our experimental results (Table 2) show that our multi-modal model significantly outperforms all three baselines: a naïve n-gram model, a purely character-level RNNLM, and a hybrid RNNLM for open-vocabulary language models. Furthermore, they confirm that morphological analyzers can improve performance of such language models on particularly morphologically rich languages. We observe that across all three languages the space-aware character-level model outperforms the purely character-based model that treats spaces just as any other character. We additionally find that the relative success of the n-gram model and the hybrid model over the character only models underscores the importance of access to word-level information, even when using a less sophisticated model.

Our methods outperform the n-gram model in all languages with either set of just two models, CM or CW. The same models outperform the hybrid baseline in Turkish and

(a) Finnish

Model	Dev	Test
KN4+OOV	2.04	1.94
RNN+OOV	2.03	1.92
PureC	2.69	2.63
C	2.40	2.32
CM	1.95	1.85
CW	2.03	1.94
CWM	**1.91**	**1.81**

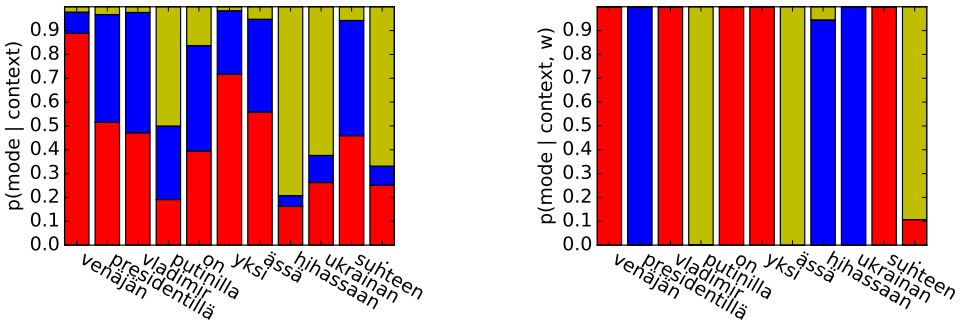
(b) Turkish

Model	Dev	Test
KN4+OOV	2.01	2.06
RNN+OOV	1.99	2.05
PureC	2.21	2.30
C	2.05	2.16
CM	1.88	1.99
CW	1.78	1.85
CWM	**1.74**	**1.82**

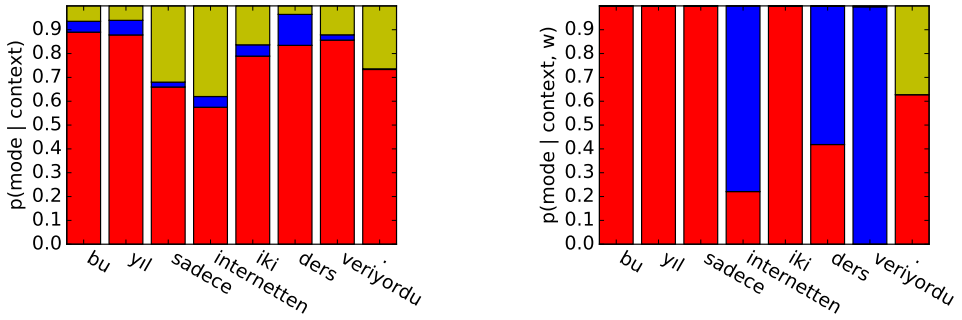
(c) Russian

Model	Dev	Test
KN4+OOV	1.68	1.70
RNN+OOV	1.62	1.66
PureC	1.91	2.05
C	1.85	1.87
CM	1.47	1.50
CW	**1.44**	**1.47**
CWM	1.49	1.52

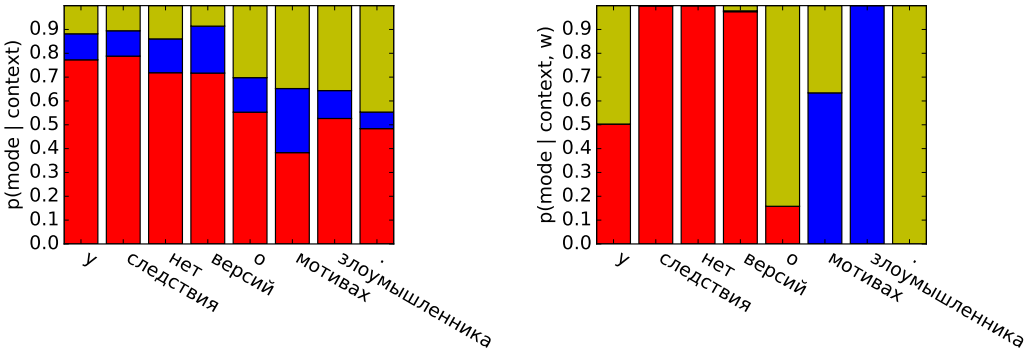
Table 2: Intrinsic evaluation of language models for three morphologically rich languages. Entropy for each test set is given in bits per character. Lower is better, with 0.0 being perfect.



Finnish: "Russian President Vladimir Putin has an ace up his sleeve in the Ukrainian relationship."



Turkish: "He gave only two lectures on the internet this year."



Russian: "The investigation does not theorize about the attacker's motives."

Figure 4: Some examples of the priors (left) and posteriors (right) over modes used to generate each word in some sample sentences. Probability given to the word-, morpheme-, and character-level models are shown in red, blue, and gold respectively.

Lang. Pair	System	BLEU
TR-EN	Baseline	15.0
	Morph. Input	**15.2**
EN-TR	Baseline	10.1
	Morph. Output	**10.5**

Table 3: Machine Translation Results

Russian, and achieves comparable results in Finnish. Finally, in the agglutinative languages, using all three modes performs best, while in Russian, a fusional language, characters and words alone edge out the model with morphology. We hypothesize that our morphology model is better able to model the long strings of morphemes found in Turkish and Finnish, but gains little from the more idiosyncratic fusional morphemes of Russians.

Some examples of the priors and posteriors of the modes used to generate some elected sentences from the held out test set can be seen in Figure 4. The figures show that all of the models tend a priori to prefer to generate words directly when possible, but that context can certainly influence its priors. In Finnish, after seeing the word *Vladmir*, the model suddenly assigns significantly more weight to the character-level model to generate the following word, which is likely to be a surname. In Russian, after the preposition *o*, the following noun is required to be in a rare case. As such, the model suddenly assigns more probability mass to the following word being generated using the morpheme-level model.

The posteriors tell a similarly encouraging story. In Finnish we see that the word *presidentilläa* is overwhelmingly likely to be produced by the morphology model due to its peculiar adessive ("at") case marking. *Vladmir* is common enough in the data that it can be generated wholly, but the last name *Putin* is again inflected into the adessive case, forming *Putinilla*. Unfortunately the morphological analyzer is unfamiliar with the stem *Putin*, forcing the word to be generated by the character-level model. In our Turkish example, all of the short words are generated at the word level, while the primary nouns *internetten* ("internet") and *ders* ("lecture") are possible to generate either as words or as a sequence of morphemes. The verb, which has much more complex morphology (progressive past tense with a third person singular agent), is generated via the morphological model.

EXTRINSIC EVALUATION

In addition to evaluating our model intrinsically using perplexity, we evaluate it on two downstream tasks. The first is machine translation between English and Turkish. The second is Turkish morphological analysis disambiguation.

Machine Translation

We introduce the score of our LM as an additional feature to a cdec (Dyer et al., 2010) hierarchical MT system trained on the WMT 2016 Turkish–English data set, and perform n-best reranking after retuning weights with the new feature. In these experiments we focus our efforts on modelling the Turkish morphology, and do not assume access to an English analyzer.

The results, shown in Table 3 demonstrate small but significant gains in both directions, particularly into Turkish, where modeling productive morphologically should be more important.

Model	Supervised?	Ambiguous Words	All words
Random Chance	no	34.08%	52.66%
Unidirectional	no	55.15%	80.28%
Bidirectional	no	**63.85**%	**84.11**%
Shen et. al	yes	91.03%	96.43%

Table 4: Morphological disambiguation accuracy results for Turkish.

Morphological Disambiguation

Our model is a joint model over words and the latent processes giving rise to those words (i.e., which generation process was selected and, for the morpheme process, which morpheme sequence was generated). While our model is not directly trained to perform morphological disambiguation, it still performs this task quite admirably. Given a trained morphological language model, a sentence $\mathbf{s}$, and a set of morphological analyses $\mathbf{z}$, we can query the model to find $p(\mathbf{s}, \mathbf{z}) = p(w_1, w_2, \ldots, w_N)$ for a given sentence. Most notably, each w_i may have a set of possible morphological analyses $\{a_1, a_2, \ldots a_{M_i}\}$ from which we would like to choose the most likely *a posteriori*. To perform this task, we simply query the model M_i times, each time hiding all but the jth possible analysis from the model. We can then re-normalize the resulting probabilities to find $p(a_j|\mathbf{s})$ for each $j \in 1 \ldots M_i$.

To make training and inference with our model tractable, we have assumed independence between previous adjacent events and the next word generation given the previous surface word forms (§2.1.1). Thus, the posterior probability over the analysis is only determined by the left context—subsequent decisions are independent of the process used to generate a word at time t. However, since disambiguating information may be present in either direction, we introduce a model variant that conditions on information in both directions. Bidirectional dependencies mean that we can no longer use the chain rule to factor the probability distribution from left-to-right. Rather we have to switch to a globally normalized, undirected model (i.e., a Markov random field) to define the probabilities of selecting the mode of generation and generation probability (conditional on the mode). The factors used to parameterize the model are defined in terms of two LSTMs, one encoding from left-to-right the prefix of the ith word ($\mathbf{h}_i$, defined exactly as above), and a second encoding from right-to-left its suffix ($\mathbf{h}'_i$). These two vector representations are used to compute a score using a locally normalized mixture model for each word. Intuitively, the morphological analysis generated at each time step should be compatible with both the preceding words and the following words.

Optimizing this model with the same MLE criterion we used in the direct model is, unfortunately, intractable since a normalizer would need to be computed. Instead, we use a pseudo-likelihood objective (Besag, 1975).

$$\begin{aligned}\mathcal{L}_{\text{PL}} &= \prod_i p(w_i \mid \boldsymbol{w}_{-i}) \\ &= \prod_i \sum_m p(m_i = m \mid \boldsymbol{w}_{-i}) p(w_i \mid m, \boldsymbol{w}_{-i})\end{aligned}$$

We note that although this model has a very different semantics from the directed one, the PL training objective is identical to the directed model's, the only difference is that features are based both on the past and future, rather than only the past.

Although evaluating sentence likelihoods in this model is intractable (a normalization factor would need to be computed), posterior inference over m_i and a_i is feasible since the normalization factors cancel and therefore do not need to be computed.

For our experiments we use the data set of Yuret and Türe (2006) who manually disambiguated from among the possible forms identified by an FST. We significantly out-perform the simple baseline of randomly guessing, and our results are competitive with Yatbaz and Yuret (2009), although they evaluated on a different dataset so they are not directly comparable. Furthermore, we also compare to a supervised model (Shen et al., 2016). While unsupervised techniques can't hope to exceed supervised accuracies, this comparison provides insight into the difficulty of the problem.

CONCLUSION

This chapter demonstrated that morphological information can be used to inform both the input representation and word generation processes. We introduced a technique for language modeling that works particularly well on morphologically rich languages where having an open vocabulary is desirable. We achieve this by using a multi-modal architecture that allows words to be input and output at the word, morpheme, or character levels. We show that knowledge of the existence of word boundaries is of critical importance for language modeling tasks, even when otherwise operating entirely at the character level, resulting in a surprisingly large reduction in per-character entropy across all languages studied.

Furthermore, we demonstrate that information from hand-crafted morphological analyzers combined with neural networks are able to out-perform linguistically naïve models. This indicates that the linguistic knowledge imbued by the analyzers is able to aid the learning and generalization of neural models.

FUTURE WORK

This chapter is largely complete, but we would like to characterize the types of sentences on which our morphologically-aware models out-perform the RNNLM baselines. We hypothesize that on sentences that contain rare or unknown morphological variants of known stems our model will shine, whereas the RNNLM will do best on sentences containing mostly frequent words. Secondarily, we hypothesize that our model will out-perform the baseline on unanalyzable unknown words by leveraging its character-level model to create well-formed tokens such as proper names. To verify this we will examine for each sentence the average number of times its terminals appear in the training corpus and examine how this average correlates with the difference between the two models' scores. Furthermore, incorporating this work into the MT system (Chapter 4) is a stretch goal.

3 NEURAL DEPENDENCY GENERATION

In the previous chapter we examined the formation of words via morphological processes. In this chapter we look at the formation of sentences via syntactic processes. There has already been evidence that syntactic structures can help language modelling tasks (Dyer et al., 2016), but previous work has used phrase structure parses, which are expensive to annotate and only available in a small number of languages. In this chapter we examine whether similar gains can be obtained using dependency trees, which exist in many more languages (Nivre et al., 2017).

Recurrent neural network grammars (Dyer et al., 2016, RNNGs) are syntactic language models that use predicted syntactic structures to determine the topology of the recurrent networks they use to predict subsequent words. Not only can they learn to model language better than non-syntactic language models, but the conditional distributions over parse trees given sentences produce excellent parsers (Fried et al., 2017).

In this chapter, we introduce and evaluate two new dependency syntax language models which are based on a recurrent neural network (RNN) backbone (§3.1).[1] Like RNNGs, our proposed models predict structure and words jointly, and the predicted syntactic structure is used to determine the structure of the neural network that is used to represent the history of actions taken by the model and to make a better estimate of the distribution over subsequent structure-building and word-generating actions. Because we use RNNs, our models do not make any explicit independence assumptions, but instead condition on the complete history of actions. The two proposed models do, however, differ in the order that they construct the trees. The first model operates top-down (§3.1.1), starting at the root and recursively generating dependents until the last modifier has been generated. The second operates bottom-up (§3.1.2), generating words from left to right and interleaving decisions about how they fit together to form tree fragments and finally a fully formed dependency tree.

Because neither model makes explicit independence assumptions, given enough capacity, infinite data, and a perfect learner, both models would converge to the same estimate. However, in our limited, finite, and imperfect world, these two models will impose different biases on the learner: in one order, relevant conditioning information may be more local (which could mean the neural networks have an easier time exploiting the information), and in another it may be more distant. These differences imply that the two models will have very different structural biases, but it is not at all clear whether one should out perform the other. Thus based on these two models we explore to what extent this choice of construction order affects performance, and we evaluate our proposed models on language modeling and parsing tasks across three typologically different languages (§3.2).

Our findings (§3.3) show that, like RNNGs, generative models make good parsers. Given the small scale of the Universal Dependency corpora, this result is also in line with previous work which shows that joint generative models can be very effective for modeling conditional distributions in the presence of small amounts of data (Yogatama et al., 2017). Second, we find that both models are less effective as language models than phrase structure RNNGs. This is not entirely surprising. Although syntactic dependencies information seems like it could be helpful for defining good conditioning contexts for language models in many cases, since its earliest days (Tesnière, 1959), dependency syntax has largely been concerned with analytic characterizations of the syntactic roles present in *existing* sen-

1 We release code for these two models, which can be found at `http://www.github.com/armatthews/rnnlm`.

GEN(yells) GEN(John) </LEFT> </RIGHT>
</LEFT> GEN(loudly) GEN(very) </LEFT>
</RIGHT> </LEFT> </RIGHT> </RIGHT>

SHIFT(John) SHIFT(yells) REDUCE(LEFT)
SHIFT(very) SHIFT(loudly) REDUCE(LEFT)
REDUCE(RIGHT) REDUCE(RIGHT)

Figure 5: A simple dependency tree and the linearized action sequences required by our top-down (left) and bottom-up (right) models to generate it. Tree nodes and arcs are annotated with the order in which each model creates them. The top-down model jointly emits words and arcs. Control symbols </LEFT> and </RIGHT> are not shown pictorially. The bottom-up model separately creates terminals (SHIFT) and arcs (REDUCE).

tences, whereas phrase structure annotations associated found in, e.g., the Penn Treebank are indebted to linguistic formalisms which are broadly concerned with determining which sentences are grammatical and which are not (e.g., government and binding theory, X-bar theory), a crucial aspect of language modeling (Marcus et al., 1993). Finally, we observe only minimal differences in language modeling performance for top-down and bottom-up models. This result is surprising in light of how different the estimation problems are, but it is a clear demonstration of the ability of RNNs to extract relevant features from data presented in any different but consistent orders.

MODELS

We present two syntax-based neural models for language modeling. The first is a top-down model, first generating the root word, then all its left children, and finally its right children. The second uses a bottom-up approach, generating terminals left-to-right and incrementally building partial tree structures atop the generated terminals. In either model during training, dependency trees over input sentences are first transformed into a sequence of oracle *actions*, either emitting a word or a control symbol. See Figure 5 for examples of such oracle action sequences. At test time we can use the same procedure to evaluate given dependency parses, or we can use beam search or a sampling procedure to generate probable sequences of actions and reverse the process to output dependency trees.

Top Down

Our first model is a top-down model. The model begins with the root and generates new nodes as follows. First, it generates a terminal using its GEN action. Next, it recursively generates zero or more left child nodes, and then takes special </LEFT> (read "end left") action. Finally, it recursively generates zero or more right child nodes, and takes a special </RIGHT> ("end right") action. See Figure 6 for examples of the effects of each of these three actions on a partially built tree structure.

At each decision point the model conditions on the output of two LSTMs: an LSTM over ancestor nodes and an LSTM over (left) sibling nodes. The former provides information about the parent node, grandparent node etc. The latter provides information about the terminals or tree fragments that are already existing siblings of the node to be generated. These two inputs are passed through an MLP and then a softmax that decides which action

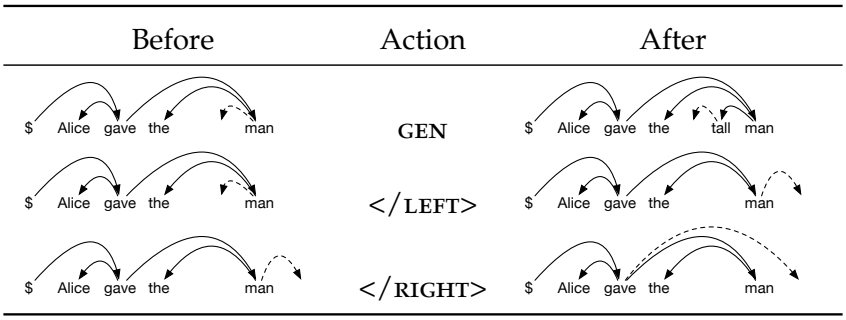


Figure 6: Examples of the three actions of our top-down model. The dotted arrow indicates where the new word will go if the GEN action is chosen next. GEN creates a new terminal node and moves the dotted arrow to point to the left of the new token. </LEFT> moves the dotted arrow from the left of the current token to the right thereof. </RIGHT> moves the dotted arrow from the right of the current token back up to its parent node.

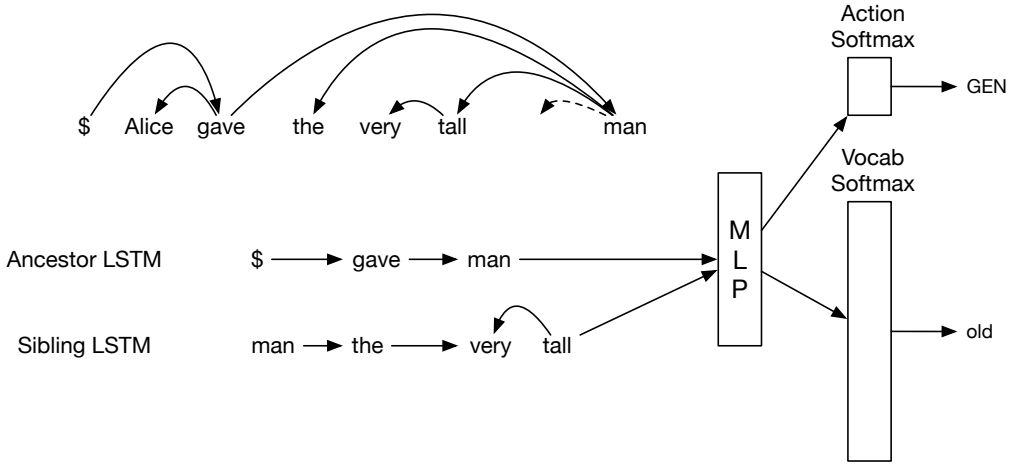


Figure 7: Our top-down model recursively generates nodes with each decision conditioned on two LSTMs. The ancestor LSTM embeds the sequence of heads from the root down to the current node, allowing the model to condition on some higher-level context. The sibling LSTM embeds the sequence of left siblings of the current node, giving the model access to lower-level local context.

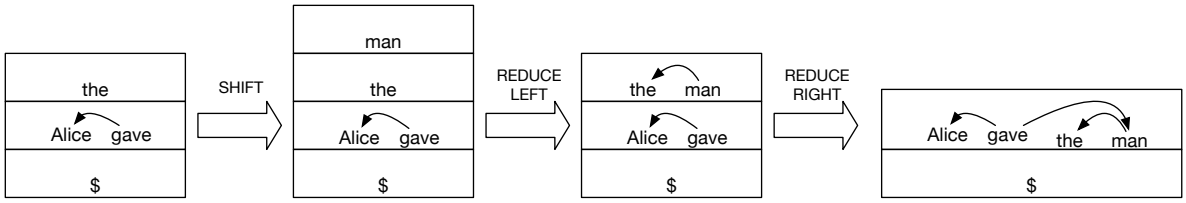


Figure 8: Examples of the three actions of our bottom-up model and their effects on the internal stack. SHIFT adds a new terminal to the top of the stack. REDUCE(LEFT) combines the top two elements of the stack with a left arc from the head of the top-most element to the head of the second element. REDUCE(RIGHT) combines the top two elements with a right arc from the head of the second element to the head of the top-most element.

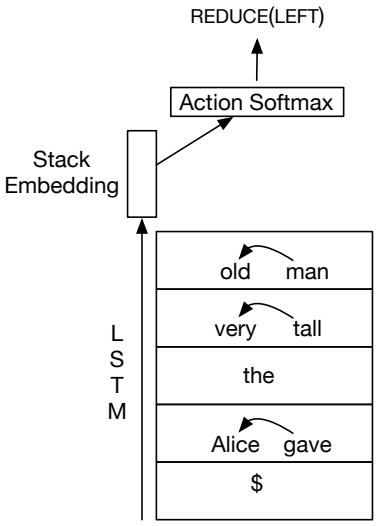


Figure 9: Our bottom up model emulates a shift-reduce parser and maintains an explicit stack. At each timestep, we use the output of an LSTM over the stack to choose the next action, which is then executed to produce a new stack state.

to take next. If the model chooses the GEN action, the hidden vector from the MLP is used to separately choose a terminal.

The ancestor LSTM is implemented as a single stack. Whenever a GEN action is taken a new terminal is emitted and gets pushed onto the stack. Whenever a </RIGHT> token is emitted, the stack is popped. In this way, the stack maintains the invariant that at any time it represents precisely the sequence of dependency heads from the root node down to the current node. </LEFT> nodes do not affect the ancestor LSTM. To embed this ancestor sequence as a single vector we pass the elements of the stack through an LSTM, starting from the root symbol and ending with the most recently generated head.

Simultaneously one sibling LSTM exists per head node on the ancestor stack and represents the sequence of children below that head, i.e. the sequence of left siblings of the next node to be produced. Whenever a terminal is emitted we push it onto the ancestor stack as described above, and we create a new sibling LSTM sequence and seed it with this new head word. Whenever a </LEFT> token is emitted a special embedding of the </LEFT> symbol is added to the current sibling LSTM. Whenever a </RIGHT> token is emitted and a head word is popped off of the ancestor stack, the final output of the corresponding sibling LSTM contains a vector representation of that head and all its children. That vector is added to the new top-most sibling LSTM. See Figure 7 for a visual depiction of this top-down model.

		Train		Dev		Test		Vocab
Language	Words	Sents	Words	Sents	Words	Sents	Singletons	Non-S'tons
English	204585	12543	25148	2002	25096	2077	9799	9873
Japanese	161900	7164	11556	511	12615	557	13091	9222
Arabic	223881	6075	30239	909	28264	680	9907	13242

Table 5: Statistics of the universal dependency data sets used in this chapter. Size of the train, dev, and test sets are given in tokens. Vocabulary information is number of types.

Bottom Up

Our second model generates sentences bottom up, in the same manner as a shift-reduce parser. A sentence is modeled as a series of actions (using arc-standard transitions) (Nivre, 2013) affecting a stack of partially completed subtrees. Each action is one of three action types: SHIFT, which pushes a new terminal onto the stack, REDUCE(LEFT), which combines the two top elements on the stack into one single subtree with the older (i.e. left) of the two as the head, and REDUCE(RIGHT) which again combines the top two elements of the stack, this time making the newer (i.e. right) one the head. See Figure 8 for examples of how these three actions affect the stack of partially built tree structures during the parsing of an example sentence.

At each time step the model conditions on the state of the stack using an LSTM running over entries from oldest (bottom) to newest (top). The resulting vector $\boldsymbol{h}$ is then passed through an MLP, and then a softmax over the three possible action types. If the SHIFT action is taken, the vector $\boldsymbol{h}$ is re-used and passed through another MLP and softmax to choose an individual word to generate. If one of the reduce actions is chosen, the top two elements from the stack are popped, concatenated (with the head-to-be first, followed by the child), and passed through an MLP. The resulting vector is then pushed back onto the stack. Kuncoro et al. (2016) showed that this type of stack-based representation alone is sufficient for language modeling and parsing, and indeed that more involved models actually damage model performance. See Figure 9 for an example of how this bottom-up model chooses an action.

Marginalization

Traditionally a language model takes a sentence $\boldsymbol{x}$ and assigns it a probability $p(\boldsymbol{x})$. Since our syntax-based language models jointly predicts the probability $p(\boldsymbol{x}, \boldsymbol{y})$ of a sequence of terminals $\boldsymbol{x}$ and a tree $\boldsymbol{y}$, we must marginalize over trees to get the total probability assigned to a sentence $\boldsymbol{x}$, $p(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{T}(\boldsymbol{x})} p(\boldsymbol{x}, \boldsymbol{y})$, where $\mathcal{T}(\boldsymbol{x})$ represents the set of all possible dependency trees over a sentence $\boldsymbol{x}$. Unfortunately the size of $\mathcal{T}(\boldsymbol{x})$ grows exponentially in the length of $\boldsymbol{x}$, making explicit marginalization infeasible.

Instead we use importance sampling to approximate the marginal (Dyer et al., 2016). We use the parser of Dyer et al. (2015), a discriminative neural stack-LSTM-based bottom-up parser, as our proposal distribution $q(\boldsymbol{x}, \boldsymbol{y})$ and compute the approximate marginal using $N = 1000$ samples per sentence: $p(\boldsymbol{x}) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{p(\boldsymbol{x}, \boldsymbol{y})}{q(\boldsymbol{x}, \boldsymbol{y})}$.

Parsing Evaluation through Reranking

In order to evaluate our model as a parser we would ideally like to efficiently find the MAP parse tree given an input sentence. Unfortunately, due to the unbounded dependencies across the sequences of actions used by our models this inference is infeasible. As such, we instead rerank a list of 1000 samples produced by the baseline discriminative parser, a model combination process that has been shown to improve performance by combining the different knowledge learned by the discriminative and generative models (Fried et al., 2017).

For each hypothesis parse in the sample list we query the discriminative parser, our top-down model, and our bottom-up model to obtain a score for the parse from each. We can then either simply output the best-scoring hypothesis from any one model, or we can learn a set of weights that optimizes performance on the dev set by combining the scores of two or more of these models. While we could conceivably optimize these weights using simple grid search, we are able to find good solutions much more quickly by employing Minimum Error Rate Training (MERT) (Och, 2003) to this end. MERT is particularly suitable for our task since we have a metric (number of correct arcs) that decomposes cleanly across sentences, and a small number of weights. Indeed in the case where we use only two models, MERT is guaranteed to find a set of weights that is globally optimal on the dev set in just one iteration. For experiments involving all three models we initialize the weights to all be negative one, and then perform 20 iterations of MERT, each starting from the output of the last, and searching in the direction of a randomly chosen three-dimensional vector.

EXPERIMENTAL SETUP

Our primary goal is to discover whether dependency-based generative neural models are able to improve the performance of their discriminative brethren, as measured on parsing and language modeling tasks. We also seek to determine the effect construction order and the biases implicit therein has on performance on these two tasks. To this end, we test a baseline discriminative parser, our two models, and all combinations of these three models on a parsing task in several languages, and we test a baseline and our two models' performance on a language modeling task on the same set of languages.

Data Sets

We use the Universal Dependency corpora (Nivre et al., 2017) for three languages: English, Japanese, and Arabic, as provided for the 2017 CoNLL shared task on universal dependency parsing. In all languages we convert all singleton terminal symbols to a special UNK token. See Table 5 for details regarding the size of these data sets.

For language modeling we evaluate using the gold sentence segmentations, word tokenizations, and part of speech tags given in the data. For parsing, we evaluate in two scenarios. In the first, we train and test on the same gold-standard data using in our language modeling experiments. In the second, we again train on gold data, but we use UDPipe (Straka and Straková, 2017) to segment, tokenize, and POS tag the dev and test sets starting from raw text, following the default scenario and most participants in the CoNLL 2017 shared task.

		English				Japanese				Arabic			
		Gold → Gold		Gold → UDPipe		Gold → Gold		Gold → UDPipe		Gold → Gold		Gold → UDPipe	
Model	Reranked?	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
CoNLL Baseline	✗	-	-	-	79.24	-	-	-	74.40	-	-	-	70.14
Dozat et al. (2017)	✗	-	-	-	84.74	-	-	-	75.42	-	-	-	76.59
Disc (Greedy)	✗	87.00	85.92	78.85	78.12	96.16	95.20	76.67	75.70	82.14	82.39	69.74	70.34
Disc (Reranked)	✓	87.48	86.30	78.99	78.23	96.04	95.17	76.66	75.58	81.70	81.34	69.20	68.79
Top-Down	✓	82.94	82.48	76.73	76.88	92.99	92.51	74.84	74.18	80.99	80.85	69.78	69.64
Bottom-Up	✓	83.11	82.70	76.79	77.13	94.56	93.25	75.85	74.18	80.61	80.70	69.30	69.24
Disc + TD	✓	88.47	87.46	80.46	79.56	96.07	95.43	76.59	74.62	82.87	82.37	70.35	70.25
Disc + BU	✓	88.29	87.12	80.09	79.33	96.17	95.54	76.82	75.92	82.48	82.18	70.18	69.99
TD + BU	✓	84.93	84.56	78.71	78.72	94.87	94.03	76.15	75.30	81.84	81.56	70.52	70.03
Disc + TD + BU	✓	**88.74**	**87.76**	**80.49**	**80.22**	**96.18**	**95.58**	**76.86**	**75.98**	**83.06**	**82.58**	**70.85**	**70.40**
Oracle	✓	97.68	97.27	91.07	90.24	99.39	99.25	79.67	80.34	91.20	89.06	77.75	76.17

Table 6: Results of parsing using our baseline discriminative parser, our two generative models, combinations thereof, and two contrastive systems from the CoNLL 2017 shared task.

Baseline Models

On the language modeling task we compare against a standard LSTM-based language model baseline (Mikolov et al., 2010), using 1024-dimensional 2-layer LSTM cells, and optimized using Adam (Kingma and Ba, 2014).

For the parsing task we compare against the discriminative parser of Dyer et al. (2015), a bottom-up transition-based parser that uses stack-LSTMs, as well as the overall top system (Dozat et al., 2017) from the 2017 CoNLL shared task (Zeman et al., 2017) on multilingual dependency parsing, which is a discriminative graph-based parser that uses a biaffine scoring function to score each potential arc.

Finally, we show the results of an oracle system looking at the 1000-best lists used for our reranking experiments. Note that since this oracle system is constrained to using only this list of samples it is not able to achieve 100% parsing accuracy.

Hyperparameters

All models use two-layer 1024-unit LSTMs and 1024-dimensional word/action embeddings. All other MLPs have a single hidden layer, again with 1024 hidden units. We implement all models using DyNet (Neubig et al., 2017), and train using Adam (Kingma and Ba, 2014) with a learning rate of 0.001, dropout with $p = 0.5$, and minibatches of 32 sentences. We evaluate the model on a held out dev set after 150 updates, and save the model to disk whenever the score is a new best. All other settings use DyNet defaults.

RESULTS

PARSING RESULTS Results on the parsing task can be found in Table 6.

We observe that in English with the gold-standard preprocessing our models perform particularly well, showing an improvement of 1.16% UAS F1 for the top-down and 0.82% UAS F1 for the bottom-up model when individually combined with our discriminative baseline parser. Combining all three models together gives a total of 1.46% absolute improvement over the baseline, indicating that the models are able to capture knowledge lacking in the baseline model, and knowledge that is complementary to each other.

The story is similar in Japanese and Arabic, though the gains are smaller in Japanese. We hypothesize that this is due to the fact that parsing Japanese is relatively easy because of

		p(x, y)		p(x)	
Lang.	Model	Dev	Test	Dev	Test
EN	RNNLM	-	-	**5.24**	**5.18**
	Top-Down	5.80	5.72	5.73	5.66
	Bottom-Up	5.63	5.56	5.53	5.47
JA	RNNLM	-	-	**4.41**	**4.58**
	Top-Down	4.82	5.00	4.73	4.93
	Bottom-Up	4.83	5.03	4.75	4.95
AR	RNNLM	-	-	**5.42**	**4.34**
	Top-Down	6.08	6.23	5.98	4.79
	Bottom-Up	6.11	6.21	5.94	4.75

Table 7: Language modeling cross entropy of our model and an RNNLM baseline. Lower is better. All scores are expressed in nats.

its strict head-final and left-branching nature, and thus our baseline is already a remarkably strong parser. This hypothesis is backed up by the fact that the baseline parser alone is only 3-4% UAS away from the oracle by itself, compared to about 10% away on English and Arabic. Thus our relative improvement, measured in terms of the percentage of possible improvement achieved, is quite consistent across the three languages, at roughly 13%.

Results on the test set using UDPipe's noisy preprocessing also saw encouraging results from the three-model ensemble gaining 1.99%, 0.40%, and 1.61% on English, Japanese, and Arabic respectively, solidly outperforming the 2017 CoNLL shared task baselines across the board, and beating Dozat et al. (2017), the overall shared task winner's, submission on Japanese.

Of particular note is that on both the gold and non-gold data, and across all three languages, the performance of the top-down and bottom-up models is quite similar; neither model consistently outperforms the other. In Japanese we do find the bottom-up parser beats the top-down one when used alone, but when combined with the discriminative model the lead evaporates, and in both of the other languages there is no clear trend. We hypothesize that the bottom-up model benefits from the strictly head-final nature of Japanese, which offers a clear signal as to the end of a constitent. English and Arabic, on the other hand, have mixed headedness, allowing the top-down model to outperform the incrementally building bottom-up model.

Overall these results are consistent with Fried et al. (2017) that has shown that generative models are particularly good at improving discriminative models through reranking, as they have an effect similar to ensembling dissimilar models.

LANGUAGE MODELING RESULTS We find that despite successes on parsing, our dependency models are not empirically suitable for language modeling. Table 7 shows the performance of our models on the language modeling task. Across all three languages, both of our models underperform a baseline RNNLM by a consistent margin of about 0.5 nats per word.

Again we note that the two models perform remarkably similarly, despite their completely different construction orders, and thus the completely different sets of information they condition on at each time step. Again neither model is a clear overall victor, and in each individual language the models are extremely close in performance.

CONCLUSION

In this chapter we test our hypothesis that dependency structures can improve performance on language modeling and machine translation tasks, in the same way that constituency parsers have been shown to help. We developed two new dependency-based models and test their effectiveness on language modeling and parsing tasks. We conclude that generative dependency models do indeed make great parsing models and, as has been observed in phrase structure parsing, combining a generative dependency parser with a traditional discriminative one does indeed improve parsing performance. We also conclude, however, that dependency models are less well suited to the task of language modeling. On this task we find no additional benefit to using dependency trees over linguistically naïve models.

These result patterns suggest that dependencies are a great tool for text analysis and discovering relations in existing texts, but are less suited to determining overall grammaticality, as in a language model, or to generating new sentences than phrase structure grammars. Finally, we find that the choice of top-down or bottom-up construction order affects performance minimally on both the parsing and language modeling tasks despite the large differences in the local conditioning contexts of each action choice.

FUTURE WORK

The aggregate results in this chapter suggest that there is little difference in the performance of top-down and bottom-up models. While this is a fascinating result we wish to better understand exactly *why* the two perform so similarly. Are the two models making the same mistakes, or does each have its own unique strengths and weaknesses that simply balance out on the whole? To answer this question we will qualitatively inspect sentences on which the two models perform mostly differently, looking for consistent patterns that cause one model to over- or under-form. In so doing we will characterize the types of decisions that each excels or fails on.

Furthermore, to get a better comparison with RNNGs we will run our models on the Penn TreeBank, which exists in both phrase-structure and dependency forms, though with the caveat that the latter is a programmatic conversion of the former.

4 LINGUISTICALLY-AWARE TRANSLATION

Machine Translation is a particularly interesting generation task with these new linguistically-aware models due to its inherent multilingualism, which exposes more varied syntactic structures and morphological processes than monolingual (read: English-centric) tasks. Furthermore, the quantity of available parallel data is extremetly limited compared to the available monolingual data, especially for small languages. As such our methods that incorporate linguistics-based biases are of particular importance as they allow us to extract as much potential as possible from the preciously little data we have.

Historically syntax-based translation models were able to out-perform naïve phrase-based machine translation(Chiang, 2005; Galley et al., 2004; Lavie et al., 2008; Hanneman et al., 2011). Quasi-synchronous grammars (Smith and Eisner, 2006; Gimpel and Smith, 2011) are of particular interest as they allow us to leverage available syntactic information in both languages. Furthermore, they avoid the cumbersome restrictions imposed by fully synchronous grammars and permit a more direct account of the fact that there may be large syntactic divergences (e.g. head swapping) between languages. Symbolic syntax-based systems (both fully- and quasi-synchronous) were slow and brittle, relying on sparse statistics and complex combinatorial global search algorithms. Neural models, on the other hand, are able to capture complex distributions found in the data, yet admit simple decoding algorithms based on beam search. Thus neural machine translation promises to harness the power such models have to offer while more easily generalizing from limited data and side-stepping the need for slow heuristic search.

To this end, we propose syntax-aware input and output models, to do quasi-synchronous translation, in which one loosely conditions on a source language syntax tree while producing a target language syntax tree. In the previous chapter we saw that our dependency-based models underform on language modeling tasks. As such, we propose using RNNGs (Dyer et al., 2016), as our target-side syntactic language model. For the source side, we will use TreeLSTMs (Tai et al., 2015) to embed the source tree.

While our proposed quasi-synchronous tree-to-tree translation system represents a novel approach to neural MT there has been much prior work on incorporating various types of syntax into NMT. Most work focuses primarily on using syntax to do source-language encodings either using dependencies (Wu et al., 2017b; Chen et al., 2017b; Bastings et al., 2017) or phrase-structure trees (Eriguchi et al., 2016; Chen et al., 2017a). Target-side syntax represents a much further departure from baseline models. Some work has focused on the less ambitious task of generating linearized trees (Aharoni and Goldberg, 2017; Nadejde et al., 2017), but more recent work has explored fully neural dependency (Wu et al., 2017a) and phrase-structure models (Eriguchi et al., 2017). This work takes the best of both worlds, combining the approaches of (Eriguchi et al., 2016) and (Eriguchi et al., 2017) to do end-to-end tree-to-tree NMT.

This chapter will examine the effects of such linguistic knowledge on machine translation between English and several typologically different languages, including Arabic, Chinese, French, and German. [1]

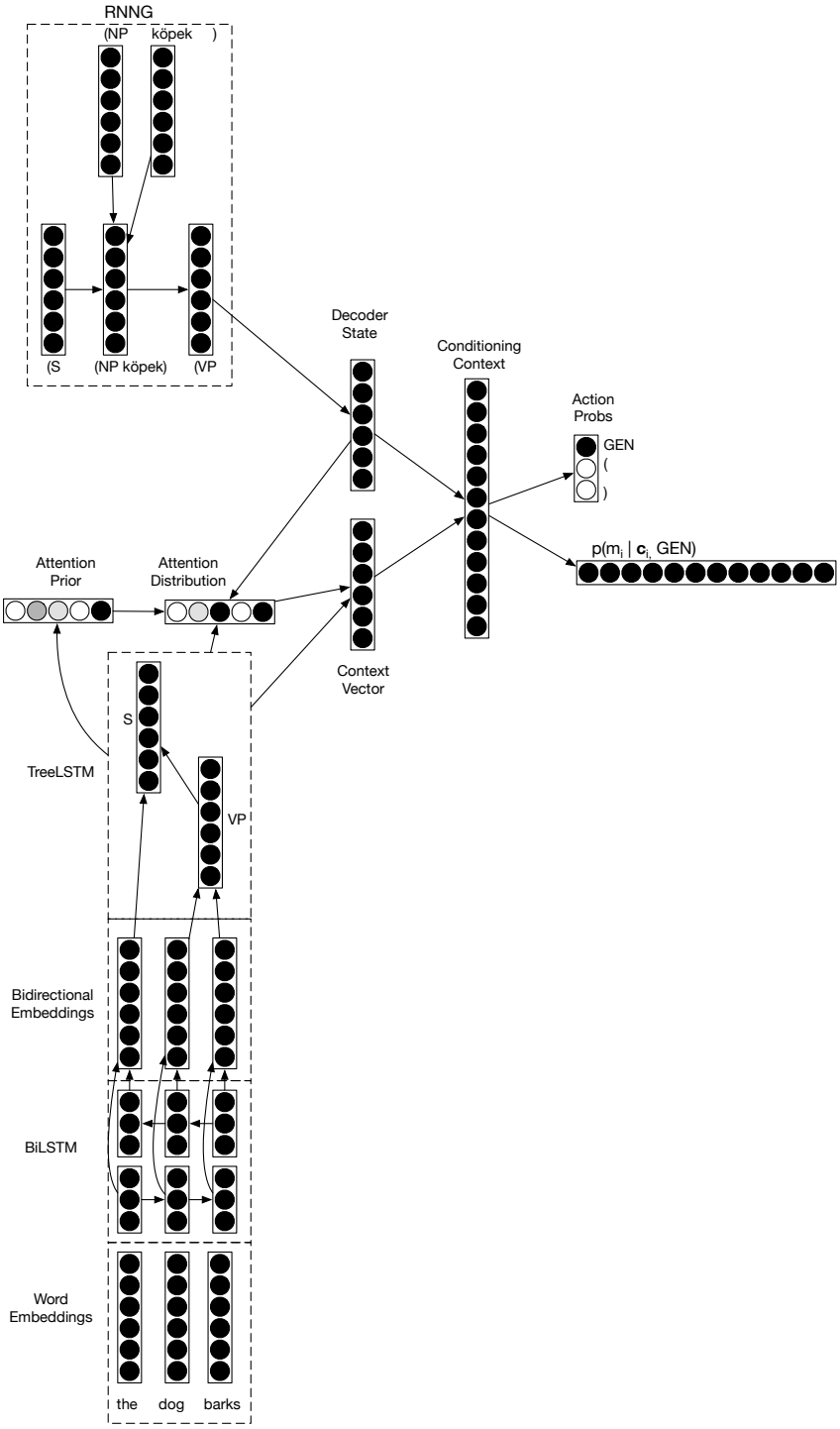


Figure 10: Our linguistically-aware machine translation system embeds words at the word-, character-, and morpheme-level. A BiLSTM is run over these embeddings to create a set of context-aware word embeddings, followed by a TreeLSTM to get a final set of embeddings for each node in the source-side tree. Simultaneously a stack LSTM is used to embed the partial target syntax tree, which again uses the three-factor embeddings for its terminals, and produces the decoder's hidden state. The decoder's hidden state is compared to each context-aware word embedding to get an attention score. The attention scores are then passed through a softmax to get an attention distribution. The context-aware embeddings are then dotted with the attention distribution to get a context vector. The context vector and decoder state are then used to predict the next action in the construction of the target side syntax tree. Here the model predicts the GEN action, after which the model must predict a terminal to generate. The model re-uses the context vector and decoder state to predict which mode to generate the next word from. It then predicts a word using each of the three modes, and marginalizes to get the final probability of each output word. Not shown: input tree LSTM, syntactic attention.

MODEL

Our model is based on the attention model (Bahdanau et al., 2015), which consists of three parts: an encoder, a decoder, and an attention mechanism. For each of these three parts we offer a syntax-based alternative. Our morphological models can be included in either the baseline or syntax-based encoder and decoder models at the terminal nodes. Below we outline each piece of this system in turn.

Syntactically-Aware Encoder Models

The baseline encoder model uses a BiLSTM to encode the source sentence. Words are embedded in a vector space (using either raw word embeddings or our morphological model), and then passed through both a forward and backwards LSTM. This produces a pair of hidden states for each word, one from the forward LSTM and one from the backward LSTM, which are then combined to create a single encoded vector representation for each word.

The syntax-aware encoder model of Tai et al. (2015) uses a syntax tree rather than a linear ordering to combine terminal embeddings. The embedding process begins at the leaves, which are embedded in the same manner as the words in the baseline model. Then the tree is traversed in a bottom-up order and a representation of each higher-level constituent is created by combining a representation of its label (e.g. NP or VP, etc.) and the representations of its children.

Syntactically-Aware Decoder Models

The baseline decoder model uses a single (foward) LSTM that additionally conditions on the *context vector* output by the attention mechanism (see § 4.1.3). At each time step the model conditions on the hidden state of the LSTM (thus indirection conditioning on all previously generated output words) and the context vector in order to predict (a distribution over) the next output word.

The syntax-aware decoder model uses the RNNG action space to construct a parse tree of the target sentence in a top-down fashion. At each time step the model conditions on a stack of partially built tree nodes along with the context vector to predict the next action to take towards constructing the target syntax tree.

Syntactically-Aware Attention Mechanisms

The attention mechanism allows the model to pick one (or a small number of) source word(s) to focus on as it emits the next target word. This alleviates the need for a single source representation that memorizes the entire input sentence, and instead lets the model translate word by word, or phrase by phrase, yet maintain the flexibility to reorder as need be. At each time step the baseline attention model receives two pieces of input: the encodings of all the words in the source sentence, and the current hidden state of the decoder. Using these two pieces it assigns a score to each source word, with a higher score meaning that that word is more important to look at when generating the next target word. These scores are then passed through a softmax, giving a (hopefully sparse) probability distribution over source words. The input word vectors are each multiplied by their respective probabilities and then summed, to form a sort of marginalized source embedding vector that represents the information the decoder needs to predict the next output word correctly.

1 This chapter is proposed work.

This vector is called the *context vector*, and is the only means by which information from the source sentence is fed into the decoder model.

The baseline attention model has no intrinsic notion of word order, nor prior thereupon. As humans, however, we know *a priori* that some word orders are more likely than others, even without knowing the languages we're translating between. For example, we know that if we just translated the nth source word, we are more likely to translate the $n+1$th source word, or perhaps the $n-1$th source word, then a word distant in the sentence (Markovian prior) (Cohn et al., 2016). We probably think that the more times a word has been translated, the less likely it is to be chosen again in the future (Coverage prior) (Mi et al., 2016; Sankaran et al., 2016). We might suppose that a word that occurs early in the source sentence is more likely to occur early in the target sentence, and similar for words towards the middle or later in the source sentence (Diagonal prior). Finally, and most centrally to this work, we might think that syntactic constituents in the source sentence should be translated as a whole. It seems fine to translate the verb phrase and then the subject noun phrase, or the subject noun phrase and then the verb phrase, but it should be very unlikely to, say, translate half of the subject noun phrase, then the verb phrase, and then the rest of the subject noun phrase. We seek to encode this idea into an attention mechanism that takes into consideration the sequence of attention distributions and encourages the model to stick to translating the current constituent until completion thereof. While using phrase structure trees from a parser in the attention mechanism novel, the ideas of using a Markovian-style prior in dependency space (Chen et al., 2017c) and the idea of using syntax as a latent variable to inform the attention mechanism (Bradbury and Socher, 2017) has shown the potential of syntax-aware models.

Formally our proposed attention prior works as follows. We define the *coverage* of a source terminal f at time step t to be the (possibly fractional) number of times it has been attended to so far: $\text{cov}(f,t) = \sum_i A_{i,f}$. We extend this definition to also account for non-terminal nodes by taking $\text{cov}(n,t) = \frac{\sum_i \sum_d A_{i,d}}{\sum_d 1}$ where d ranges over terminal descendents of the non-terminal n. We then define the *urgency* of a terminal f to be $U(f,t) = \max_s \text{cov}(s,t) - \text{cov}(f,t)$ where s ranges over the children of f's parent node, including f itself. The prior probabilities of aligning to each terminal are then computed as the softmax of the terminals' urgency scores. This prior encodes precisely the intution outlined above – untranslated terminals whose siblings have been translated should have high prior probabilities while terminals whose siblings (including itself) have all been translated should have low prior probabilities.

EXPERIMENTS

We want to show improvements in translation quality across several language pairs. Experiments will include an ablation over use of the syntactic encoder, decoder, and attention mechanism, as well as the morphological terminal encoder and decoder mechanisms.

Corpora

Translation experiments use the WIT^3 corpus of TED talks [2]. We experiment with translation between English and a range of syntactically different languages: Arabic, Chinese, French, and German. We use the Stanford Parser (Klein and Manning, 2003; Levy and Manning, 2003; Rafferty and Manning, 2008) to parse both sides of the copora. We discard from the training data all sentences longer than 100 tokens, as well as sentences that con-

2 https://wit3.fbk.eu/

Enc.	Dec.	Att.	Turkish BLEU	Finnish BLEU	Russian BLEU
✗	✗	✗	TBD	TBD	TBD
✗	✗	✓	TBD	TBD	TBD
✗	✓	✗	TBD	TBD	TBD
✗	✓	✓	TBD	TBD	TBD
✓	✗	✗	TBD	TBD	TBD
✓	✗	✓	TBD	TBD	TBD
✓	✓	✗	TBD	TBD	TBD
✓	✓	✓	TBD	TBD	TBD

Table 8: Machine translation ablation results. Features include the syntactic encoder (Enc.), syntactic decoder (Dec.), and syntactic attention mechanism (Att.).

tain characters unknown to the Stanford tokenizer, which in this corpus is primarily music notes.

Baseline Models

As our baseline we take the attention-based NMT model with a BiLSTM encoder and standard LSTM decoder (Bahdanau et al., 2015).

Hyperparameters

We use 1024-unit LSTMs, 1024-dimensional word embeddings, and 1024-dimensional hidden layers for the atttention MLP and the decoder's final MLP. We implement all models using xnmt (Neubig et al., 2018) and DyNet (Neubig et al., 2017), and train using Adam (Kingma and Ba, 2014) with a learning rate of 0.001, dropout with $p = 0.5$ and minibatches of 32 sentences. We evaluate each model on a held out dev set after 150 updates, and save the model to disk whenever the score is a new best. All other settings use xnmt defaults.

Attention Priors

As a preliminary experiment to test the viability of work on attention priors we test a few of them on a small-scall Chinese to English translation system trained on the BTEC corpus without the morphological or syntactic enhancements described in this document. Each system was periodically evaluated on a held-out dev set until the perplexity on the dev set stopped decreasing. The final perplexities on the dev set for a few selected priors are shown in Table 9.

These results show that the syntax-based prior does indeed have a strong effect on the dev perplexity of this machine translation system, and that it scales superlinearly with the diagonal prior. We emphasize again that these results are preliminary. Perplexity does not always coorelate strongly with BLEU, and this system was trained on a small system and on data from a tight domain. Nevertheless the results are highly encouraging, inspiring future work in this direction.

Prior(s)	Dev Perp
None	19.27
Diagonal	19.21
Coverage	19.36
Syntax	18.06
Syn. + Diag.	**17.63**

Table 9: Preliminary perplexities on a held-out dev set of a small Chinese–English NMT system trained using different attention priors.

FUTURE WORK

While this whole chapter should be considered future work, we list the expected contributions explicitly below:

1. A quasi-synchronous neural MT system based on tree encoders and RNNGs
2. A systematic comparison of attention priors and combinations thereof
3. Analysis of phenomena that syntactic models see improved success on compared to baseline NMT models
4. An ablation to examine which components benefit most from the addition of syntactic information
5. (Stretch goal) implementation of our morphological handling and analysis of its effect on MT quality

CONCLUSIONS

This chapter focuses on using syntax-infused neural networks to model *conditional* generation of sentences, a more practical and more difficult goal than the unconditional generation models described in Chapter 3. This task is particularly interesting because not only can we use syntax to govern generation, but we can also use it to modulate how sentences are represented in the source language and how the model draws connections between the two languages. This chapter again provides evidence that morphologically-aware neural models outperform naïve ones, this time on the more practical (and more difficult) task of conditional text generation. Furthermore, it also follows up on historical work by testing whether the syntax-based improvements added into statistical models continue to have practical value in the age of neural networks.

5 TIMELINE

The proposed timeline for the remainder of this thesis is as follows:

- April-July 2018: Translation Project
- August-September 2018: Writing
- October 2018: Slippage month
- November 2018: Defense
- December 2018: Revisions

BIBLIOGRAPHY

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. *arXiv preprint arXiv:1704.04743*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Machine Learning*.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675*.

Julian Besag. 1975. Statistical analysis of non-lattice data. *The statistician*, pages 179–195.

James Bradbury and Richard Socher. 2017. Towards neural machine translation with latent tree attention. *arXiv preprint arXiv:1709.01915*.

Victor Chahuneau, Noah A Smith, and Chris Dyer. 2013. Knowledge-rich morphological priors for bayesian language models. Association for Computational Linguistics.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017a. Improved neural machine translation with a syntax-aware encoder and decoder. *arXiv preprint arXiv:1707.05436*.

Kehai Chen, Rui Wang, Masao Utiyama, Lemao Liu, Akihiro Tamura, Eiichiro Sumita, and Tiejun Zhao. 2017b. Neural machine translation with source dependency representation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2846–2852.

Kehai Chen, Rui Wang, Masao Utiyama, Eiichiro Sumita, and Tiejun Zhao. 2017c. Syntax-directed attention for neural machine translation. *arXiv preprint arXiv:1711.04231*.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

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

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. 2016. Incorporating structural alignment biases into an attentional neural translation model. *arXiv preprint arXiv:1601.01085*.

Timothy Dozat, Peng Qi, and Christopher D Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. ACL*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars.

Chris Dyer, Jonathan Weese, Hendra Setiawan, Adam Lopez, Ferhan Ture, Vladimir Eidelman, Juri Ganitkevitch, Phil Blunsom, and Philip Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, pages 7–12. Association for Computational Linguistics.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. *arXiv preprint arXiv:1603.06075*.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. *arXiv preprint arXiv:1702.03525*.

Daniel Fried, Mitchell Stern, and Dan Klein. 2017. Improving neural parsing by disentangling model combination and reranking effects. *arXiv preprint arXiv:1707.03058*.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule. Technical report, COLUMBIA UNIV NEW YORK DEPT OF COMPUTER SCIENCE.

Kevin Gimpel and Noah A Smith. 2011. Quasi-synchronous phrase dependency grammars for machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 474–485. Association for Computational Linguistics.

Greg Hanneman, Michelle Burroughs, and Alon Lavie. 2011. A general-purpose rule extractor for scfg-based machine translation. In *Proceedings of the Fifth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 135–144. Association for Computational Linguistics.

Betty Hart and Todd R Risley. 2003. The early catastrophe. *Education Review (London)*, 17(1):110–118.

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

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. 2017. Learning to create and reuse words in open-vocabulary neural language modeling. *arXiv preprint arXiv:1704.06986*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2741–2749.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith. 2016. What do recurrent neural network grammars learn about syntax? *arXiv preprint arXiv:1611.05774*.

Alon Lavie, Alok Parlikar, and Vamshi Ambati. 2008. Syntax-driven learning of subsentential translation equivalents and translation rules from parsed parallel corpora. In *Proceedings of the Second Workshop on Syntax and Structure in Statistical Translation*, pages 87–95. Association for Computational Linguistics.

Roger Levy and Christopher Manning. 2003. Is it harder to parse chinese, or the chinese treebank? In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 439–446. Association for Computational Linguistics.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.

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

Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. 2016. Coverage embedding models for neural machine translation. *arXiv preprint arXiv:1605.03148*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Yasumasa Miyamoto and Kyunghyun Cho. 2016. Gated word-character recurrent language model. *arXiv preprint arXiv:1606.01700*.

Maria Nadejde, Siva Reddy, Rico Sennrich, Tomasz Dwojak, Marcin Junczys-Dowmunt, Philipp Koehn, and Alexandra Birch. 2017. Predicting target language ccg supertags improves neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 68–79.

Graham Neubig and Chris Dyer. 2016. Generalizing and hybridizing count-based and neural language models. *arXiv preprint arXiv:1606.00499*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Graham Neubig, Matthias Sperber, Xinyi Wang, Matthieu Felix, Austin Matthews, Sarguna Padmanabhan, Ye Qi, Devendra Singh Sachan, Philip Arthur, Pierre Godard, et al. 2018. Xnmt: The extensible neural machine translation toolkit. *arXiv preprint arXiv:1803.00188*.

Joakim Nivre. 2013. Transition-based parsing.

Joakim Nivre, Lars Ahrenberg Željko Agic, et al. 2017. Universal dependencies 2.0. lindat/clarin digital library at the institute of formal and applied linguistics, charles university, prague.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics.

Kemal Oflazer. 1994. Two-level description of turkish morphology. *Literary and linguistic computing*, 9(2):137–148.

Anna N Rafferty and Christopher D Manning. 2008. Parsing three german treebanks: Lexicalized and unlexicalized baselines. In *Proceedings of the Workshop on Parsing German*, pages 40–46. Association for Computational Linguistics.

Baskaran Sankaran, Haitao Mi, Yaser Al-Onaizan, and Abe Ittycheriah. 2016. Temporal attention model for neural machine translation. *arXiv preprint arXiv:1608.02927*.

Qinlan Shen, Daniel Clothiaux, Emily Tagtow, Patrick Littell, and Chris Dyer. 2016. The role of context in neural morphological disambiguation.

David A Smith and Jason Eisner. 2006. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 23–30. Association for Computational Linguistics.

Milan Straka and Jana Straková. 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

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

Lucien Tesnière. 1959. Eléments de linguistique structurale. *Paris, Klincksieck*, 2.

Shuangzhi Wu, Dongdong Zhang, Nan Yang, Mu Li, and Ming Zhou. 2017a. Sequence-to-dependency neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 698–707.

Shuangzhi Wu, Ming Zhou, and Dongdong Zhang. 2017b. Improved neural machine translation with source syntax. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017). Melbourne, Australia*, pages 4179–4185.

Mehmet Ali Yatbaz and Deniz Yuret. 2009. Unsupervised morphological disambiguation using statistical language models. In *Pro. of the NIPS 2009 Workshop on Grammar Induction, Representation of Language and Language Learning, Whistler, Canada*, pages 321–324.

Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. *arXiv preprint arXiv:1703.01898*.

Deniz Yuret and Ferhan Türe. 2006. Learning morphological disambiguation rules for turkish. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 328–334. Association for Computational Linguistics.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, et al. 2017. Conll 2017 shared task: multilingual parsing from raw text to universal dependencies. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19.