NORTHWESTERN UNIVERSITY

Inspecting and Directing Neural Language Models

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Thanapon Noraset

EVANSTON, ILLINOIS

June 2018

ABSTRACT

Inspecting and Directing Neural Language Models

Thanapon Noraset

The ability of a machine to synthesize textual output in a form of human language is a long-standing goal in a field of artificial intelligence and has wide-range of applications such as spell correction, speech recognition, machine translation, abstractive summarization, etc. The statistical approach to enable such ability mainly involves defining representations of textual inputs and computation of likelihood of the outputs text sequence. With the recent advancement in a neural network or deep learning research, machine learning models can construct general vector representations of words, also known as word embeddings, that are useful for many natural language processing tasks. Furthermore, neural language models can accurately assign a probability to a sequence of text, and become a default choice of researchers and developers. One of the key advantages of these deep learning models is that they require little to none human heuristics in feature engineering, and instead, are optimized using a large amount of data.

Despite the performance improvement and convenience of the neural language models, they lack a useful property found in the previous statistical model such as an n-gram language model – the parameters of neural language models are not directly interpretable. This leads to a set of challenges when using such models. For instance, the word embeddings do not directly convey what information is captured, hence which semantic gaps exist in the embeddings is uncertain. Furthermore, the likelihoods of a sequence of text are defined by a series of non-linear functions, making the behavior of the models, even for a short phrase, difficult to ascertain or change to the desired direction.

This dissertation addresses these two shortcomings by exploiting the models' ability to generate interpretable text. First, we propose a more transparent view of the information captured by a word embedding, and introduce the Definition Modeling, the task of generating a definition for a given word and its embedding. Second, we study an explicit approach to adjust the model's behavior, and present Dynamic KL Regularization, a method for training neural language models to follow a given set of statistical constraints. Finally, we explore efficient solutions to a fundamental, yet lacking the capability of the start-of-the-art neural language model: accurately computing a likelihood of a short phrase that it will produce.

Acknowledgements

This dissertation was possible with the support of several people. I would like to express my sincere gratitude to all of them.

First and foremost, I would like to thank my advisors, Prof. Doug Downey for his invaluable teaching and guidance. I greatly appreciated our weekly meetings, they were full of thought-provoking research challenges and helpful directions when needed. He patiently listened to my ideas, asked critical questions, and extracted the essence. His positive thinking and excitement were always encouraging. Most importantly, during my time in graduate school, Prof. Downey had trained me to think like a researcher.

I thank my dissertation committees, Prof. Larry Birnbaum and Prof. Klinton Bicknell. Our conversations were inspirational and helpful. Their valuable feedbacks and questions helped shape my research. Thanks to Prof. Birnbaum who regularly stopped by and discussed ideas, one of which had turned into the starting point of this dissertation. I am thankful for Prof. Bicknell who had inspired me to think more critically about language and see research challenges from the linguistics point of view.

For the most of my Ph.D. study, I was supported by the Faculty of Information and Communication Technology, Mahidol University. I would like to sincerely thank the broad of administration and the supporting staffs who effortlessly helped me stay focus and motivated on my research. I also would like to thank the staffs in the Office of

Educational Affairs, Royal Thai Embassy who reduced the stress of living in the United States.

There are many friends and collaborators that made my graduation possible. I would like to thank Chandra Sekhar Bhagavatula who have collaborated with me on many projects, and our first project led me to work with Prof. Downey. I would like to especially thank the CogSys Loop commuters including Chandra, Chen Liang, and Subu Kandaswamy for the countless conversations. I have learned much from them, not to mention political opinions and cultural exchange. I am grateful for the contributions of friends in my research group – especially Rony Mohammed Alam, David Demeter, and Zheng Yuan. Zheng was also my exercise buddy for both physical and mental strength.

Besides, I got to know many people and become friends who have made my experience at Northwestern University so pleasant. Thanks to my office neighbors: Joe Blass, Max Crouse, Clifton McFate, Irina Rabkina, Ethan Robison, and Bryan Head. We frequently discussed ideas and had fun conversations. Thanks to German Espinosa, Ettore Trainiti who always worried about me not having enough fun. I was fortunate enough to be part of the AI Journal Club from the very beginning. For this, I thank Scott Cambo whose leadership brought together many people from different departments to share ideas. I am thankful for my close friend Cheng Xing Yuan who had been my awesome roommate for many years.

Finally, I thank my family for all their everlastingly support and encouragement.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

A machine that reads input and produces output in natural languages, similar to how humans communicate, has been a long standing subject of research in the field of artificial intelligence. In term of applications, such capability provides a natural way for us to interact with a machine, after all, languages are the main method of human communication. Despite being a structured and conventional way of communication, human languages are difficult to express explicitly in a form that a machine can understand due to their flexibility and ambiguity. This leads to statistical approaches that learn a model of a language from a textual data. Indeed, a statistical language model is a key component for many natural language applications, especially ones that synthesize the output in natural language. Examples include spell correction [28, 102, 108], speech recognition [20, 22, 61, 73], machine translation [5, 11, 96, 106], image captioning [39, 109], conversation agent [50, 78, 85], etc.

A statistical language model provides a relative likelihood of different pieces of text. For example, it is useful to a speech recognition system to know that "`this is nice`" is more likely than "`thesis an ice`" in a everyday spoken language. Specifically, the language modeling is a task of assigning a probability distribution over sequences of words (or other units), a fundamental task in a field of natural language processing (NLP). A statistical language model mainly involve defining representation of words and computation of a likelihood of a sequence of words [5, 80]. Traditionally, we would model the joint distribution between words in a sequence (words are represented as discrete variables). n-gram language models had been a state-of-the-art model for decades [16, 29]. However, we still have the data sparsity problem because the training data only contain a few possible sequences.

1.1. Neural Language Models

To overcome the data sparsity problem, researchers (1) change the discrete representation of words into continuous feature vectors, and (2) express the joint distribution as a function of the feature vectors. The models learn the word representations (feature vectors) and the probability function simultaneously from a training data [5, 61, 63]. When underlying model is neural network, we refer to this model as a neural language model.

With the recent advancement in neural network or deep learning research, the main approaches to model language have been dominated by neural language models. Not only they have been the state-of-the-art performance in assigning probability to sequences of words [37, 54, 61, 112], the word embeddings are also useful for other NLP applications [60, 71]. For this reason, they become a default choice for natural language processing researches and applications [21, 35, 93].

Despite the performance improvement and convenience of the neural language models, they lack a useful property found in previous statistical model such as an n-gram language model – the parameters of neural language models are not directly interpretable. An n-gram language model has a symbolic representation of words, and explicitly store the likelihood of a phrase (n-gram) as the parameters (i.e. $P(\texttt{nice}|\texttt{this is}) = 8.43 \times 10^{-4}$). The computation of a likelihood of a sequence is a straightforward application of the chain rule of probability:

$$P(\texttt{this is nice}) = P(\texttt{this})P(\texttt{is}|\texttt{this})P(\texttt{nice}|\texttt{this is})$$

While generalization techniques such as Kneser-Ney smoothing [45] or Katz backoff [41] add complexity to the model, we can still trace and understand how a model arrives at an output.

On the other hands, information about words and probability function of neural language models are distributed into, often, hundred millions of continuous variables. This leads to a set of challenges when using such models. In this dissertation, we propose novel solutions to two of the shortcomings:

(1) The word embeddings do not directly convey what information being captured, hence which semantic gaps exist in the embeddings is uncertain.
(2) The likelihoods of a sequence of text is defined by a series of non-linear functions, making behavior of the models, even for a short phrase, difficult to ascertain or change to desired direction.

1.2. Contributions and Outline

The common theme in this dissertation is to provide novel insights and methods to tackle the opacity of neural language models. We formulate the contributions in this work as follows.

Defining word representations in natural language: In chapter 2 and Noraset et al., 2017 [68], we introduce *Definition Modeling*, the task of generating a definition for a given word and its embedding. Dictionary definitions of words, such as

```
robot:  a machine capable of carrying out a complex
series of actions automatically, especially one
programmable by a computer.
```

are a more direct and transparent representation of the embeddings' semantics. We present several definition model architectures based on recurrent neural networks, and evaluate them on two tasks: generating plausible definitions and dictionary lookup (both forward and reverse).

We show that, with the right architecture, it is possible to generate an interpretable definition of a word embedding. A model that controls dependencies between the word being defined and the definition words performs significantly better, and that a character-level information can complement word-level embeddings. Not only this model can generate better definitions, it also reveals some internal process underlying the model's behavior. Additionally, the errors made by a definition model provide insight into the shortcomings of both word embeddings and recurrent neural network language models.

In particular, the most apparent errors are not of semantics missing from the embeddings, but rather abnormal phrases being generated by all models in the experiments such as repetition of phrases. This leads us to the following research objective.

Controlling neural language model behavior: In chapter 3 and Noraset et al., 2018 [67], we introduce *Dynamic KL Regularization*, a method to subject the textual output generated by a recurrent neural network language model to a set of interpretable constraints. For example, we can specify when training the model that it should generate more of "`mr. dog`" and less of "`mr. president`", i.e. changing behavior of the model during generation.

We demonstrate how an explicitly stated n-gram distribution can be used as a set of soft constraints to direct the language model behavior during the generation without sacrificing its accuracy in assigning probability to sequences of words. The results show that Dynamic KL Regularization is successful at reducing word-level repetition (a common problematic behavior). It also improves model generalizability by incorporating statistical constraints are n-gram statistics taken from a large corpus.

The statistical constraints are defined over aggregate model behavior, rather than model parameters, and the proposed method is dynamically updated as training proceeds, based on the output generated by the model. While effective, it is highly computational intensive. Can we infer the model behavior efficiently i.e. n-gram probabilities without generating large amount of text?

Estimating probabilities of short phrases without context: In chapter 4, we discuss how to compute a probability that a recurrent neural network model assigns to a short sequence of text, when the preceding context is absent. This simple, yet lacking capability of the start-of-the-art neural language models is useful for (1) estimating how likely a phrase to occur in a corpus regardless of context, and (2) characterizing model behavior so that we can make an informed change.

We experiment with many techniques to estimate the context-independent the probability of phrases. We demonstrate that a simple method of altering the training to occasionally *reset* the context information is remarkably effective and efficient comparing to other approaches.

CHAPTER 2

Defining Word Representations in Natural Language

How can we use word embeddings to define their corresponding words in natural language? In this chapter, we explore this question, and show that it is possible for a neural language model to learn to synthesize human-readable dictionary definitions from word embeddings (some shown in Table 2.1). Finally, we discuss insights of the results.[1]

Table 2.1. Selected examples of generated definitions. The words being defined here are not in the training data. Table shows both good output definitions (top) and output definitions with minor a mistake.

Word	Generated definition
brawler	a person who fights
butterfish	a marine fish of the atlantic coast
continually	in a constant manner
creek	a narrow stream of water
feminine	having the character of a woman
juvenility	the quality of being childish
mathematical	of or pertaining to the science of mathematics
negotiate	to make a contract or agreement
prance	to walk in a lofty manner
resent	to have a feeling of anger or dislike
similar	having the same qualities
valueless	not useful
accused	to make a false or unethical declaration of
adorable	having the qualities of a child
hello	a song
inward	not directed to the center
precise	to make a precise effort

[1]The main contributions in this chapter were first described and published in *Definition Modeling: Learning to define word embeddings in natural language* [68].

2.1. Introduction

Word embeddings, are a key component in many natural language processing (NLP) models [93, 35]. Several neural network techniques have been introduced to learn high-quality word embeddings from unlabeled textual data [71, 60, 110]. Embeddings have been shown to capture lexical syntax and semantics. For example, it is well-known that nearby embeddings are more likely to represent synonymous words [46] or words in the same class [25]. More recently, the vector offsets between embeddings have been shown to reflect analogical relations [58]. However, tasks such as word similarity and analogy only evaluate an embedding's lexical information indirectly.

In this work, we study whether word embeddings can be used to generate natural language definitions of their corresponding words. Dictionary definitions serve as direct and explicit statements of word meaning. Thus, compared to the word similarity and analogical relation tasks, definition generation can be considered a more transparent view of the syntax and semantics captured by an embedding. We introduce *definition modeling*: the task of estimating the probability of a textual definition, given a word being defined and its embedding. Specifically, for a given set of word embeddings, a definition model is trained on a corpus of word and definition pairs. The models are then tested on how well they model definitions for words not seen during the training, based on each word's embedding.

The definition models studied in this work are based on recurrent neural network (RNN) models [26, 34]. RNN models have established a new state-of-the-art performance on many sequence prediction and natural language generation tasks [17, 39, 89, 99]. An important characteristic of dictionary definitions is that only a subset of the words in the definition depend strongly on the word being defined. For example, the word "`woman`" in the definition of "`feminine`" in Table 2.1 depends on the word being defined than the rest. To capture the varying degree of dependency, we introduce a gated update function that is trained to control information of the word being defined used for generating each definition word. Furthermore, since the morphemes of the word being defined plays a vital role in the definition, we experiment with a character-level convolutional neural network (CNN) to test whether it can provide complementary information to the word embeddings [42, 47]. Our best model can generate fluent and accurate definitions as shown in Table 2.1. We note that none of the definitions in the table exactly match any definition seen during training.

Our contributions are as follows: (1) We introduce the definition modeling task, and present a probabilistic model for the task based on RNN language models. (2) In experiments with different model architectures and word features, we show that the gate function improves the perplexity of a RNN language model on definition modeling task by $\sim$10%, and the character-level CNN further improves the perplexity by $\sim$5%. (3) We also show that the definition models can be use to perform the reverse dictionary task studied in previous work, in which the goal is to match a given definition to its corresponding word. Our model achieves an 11.8% absolute gain in accuracy compared to previous state-of-the-art by Hill et al. 2016 [32]. (4) Finally, our error analysis shows that a well-trained set of word embeddings pays significant role in the quality of the generated definitions, and some of error types suggest shortcomings of the information encoded in the word embeddings.

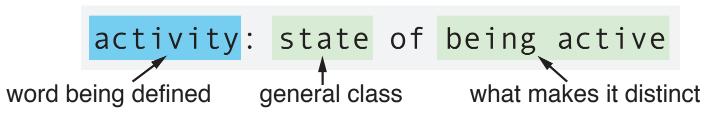

Figure 2.1. A dictionary definition usually consists of a class and a differentiae.

2.2. Dictionary Definitions

In this section, we first investigate definition content and structure through a study of existing dictionaries. We then describe our new data set, and define our tasks and metrics.

2.2.1. Definition Content and Structure

In existing dictionaries, individual definitions are often comprised of *genus* and *differentiae* [19, 65]. The *genus* is a generalized class of the word being defined, and the *differentiae* is what makes the word distinct from others in the same class. For instance,

```
Phosphorescent:  emitting light without appreciable heat as
by slow oxidation of phosphorous
```

"`emitting light`" is a genus, and "`without applicable heat ...`" is a differentiae. Furthermore, definitions tend to include common patterns such as "`the act of ...`" or "`one who has ...`" [53]. However, the patterns and styles are often unique to each dictionary.

The *genus + differentiae (G+D)* structure is not the only pattern for definitions. For example, the entry below exhibits distinct structures.

```
Eradication:  the act of plucking up by the roots; a rooting
out; extirpation; utter destruction
```

This set of definitions includes a synonym (“`extirpation`”), a reverse of the *G+D* structure (“`utter destruction`”), and an uncategorized structure (“`a rooting out`”).

2.2.2. Corpus: Preprocessing and Analysis

Dictionary corpora are available in a digital format, but are designed for human consumption and require preprocessing before they can be utilized for machine learning. Dictionaries contain non-definitional text to aid human readers, e.g. the entry for “`gradient`” in Wordnik[2] contains fields (“`Mathematics`”) and example usage (“`as, the gradient line of a railroad.`”). Further, many entries contain multiple definitions, usually (but not always) separated by “`;`”.

We desire a corpus in which each entry contains only a word being defined and a single definition. We parse dictionary entries from GCIDE[3] and pre-process WordNet’s glosses, and the fields and usage are removed. The parsers and preprocessing scripts can be found at https://github.com/northanapon/dict-definition.

To create a corpus of reasonable size for machine learning experiments, we sample around 20k words from the 50k most frequent words in the Google Web 1T corpus [10], removing function words. In addition, we limit the number of entries for each word in a dictionary to three before the splitting by “`;`” (so that each word being defined may repeat multiple times in our corpus). After cleaning and pruning, the corpus has a vocabulary size of 29k. Other corpus statistics are shown in Table 2.2.

We analyze the underlying structure of the definitions in the corpus by manually labeling each definition with one of four structures: *G+D*, *D+G*, *Syn* (synonym), and

[2]https://www.wordnik.com/words/gradient
[3]http://gcide.gnu.org.ua/

Table 2.2. Basic statistics of the common word definitions corpus. Splits are mutually exclusive in the words being defined.

Split	train	valid	test
#Words	20,298	1,127	1,129
#Entries	146,486	8,087	8,352
#Tokens	1,406,440	77,948	79,699
Avg length	6.60	6.64	6.54

Misc. In total, we examine 680 definitions from 100 randomly selected words. The results are shown in Table 2.3. We reaffirm earlier studies showing that the *G+D* structure dominates in both dictionaries. However, other structures are also present, highlighting the challenge inherent in the dictionary modeling task. Further, we observe that the *genus* term is sometimes general (e.g., "`one`" or "`that`"), and other times specific (e.g. "`an advocate`").

Table 2.3. The number of manually labeled structures of dictionary definitions in WordNet (WN) and GCIDE (GC). *G+D* is *genus* followed by *differentiae*, and *D+G* is the reverse. *Syn* is a synonym.

Label	WN	GC	Example
G+D	85%	50%	laminate: to divide into thin plates
D+G	7%	9%	fawn: a young deer
Syn	1%	32%	activity: energy
Misc.	4%	8%	diagonally: in a diagonal direction
Error	3%	1%	absolutely: used as intensifiers
Total	256	424	

2.2.3. Dictionary Definition Tasks

In the definition modeling (DM) task, we are given an input word w^*, and output the likelihood of any given text D being a definition of the input word. In other words, we estimate $P(D|w^*)$. We assume our definition model has access to a set of word embeddings, estimated from some corpus other than the definition corpus used to train the definition model.

DM is a special case of language modeling, and as in language modeling the performance of a definition model can be measured by using the perplexity of a test corpus. Lower perplexity suggests that the model is more accurate at capturing the definition structures and the semantics of the word being defined.

Besides perplexity measurement, there are other tasks that we can use to further evaluate a dictionary definition model including definition generation, and the reverse and forward dictionary tasks. In definition generation, the model produces a definition of a test word. In our experiments, we evaluate generated definitions using both manual examination and BLEU score, an automated metric for generated text [15, 69]. The reverse and forward dictionary tasks are ranking tasks, in which the definition model ranks a set of test words based on how likely they are to correspond to a given definition (the *reverse dictionary* task) or ranks a set of test definitions for a given word (the *forward dictionary* task) [32]. A dictionary definition model achieves this by using the predicted likelihood $P(D|w^*)$ as a ranking score.

2.3. Definition Models

The goal of a definition model is to predict the probability of a definition ($D = [w_1, ..., w_T]$) given a word being defined w^*. Our model assumes that the probability of generating the tth word w_t of a definition text depends on both the previous words and the word being defined (Equation 2.1). The probability distribution is usually approximated by a softmax function (Equation 2.2)

$$p(D|w^*) = \prod_{t=1}^{T} p(w_t|w_1, .., w_{t-1}, w^*) \tag{2.1}$$

$$p(w_t = j|w_1, .., w_{t-1}, w^*) \propto \exp(\theta_o^j h_t/\tau) \tag{2.2}$$

where θ_o^j is parameters associated with word j, h_t is a vector summarizing inputs so far at token t, and τ is a hyper-parameter for temperature, set to be 1 unless specified. Note that in our expression, the word being defined w^* is present at all time steps as an additional conditioning variable.

The definition models explored in this work are based on a recurrent neural network language model (RNNLM) [61]. To recall, an RNNLM is comprised of RNN units, where each unit reads one word w_t at every time step t and outputs a hidden representation h_t for Equation 2.2.

$$h_t = g(w_{t-1}, h_{t-1}, w^*) \tag{2.3}$$

where g is a recurrent nonlinear function, w_t is represented by the embedding of the word, and w^* is likewise the embedding (and other features) of the word being defined.

2.3.1. Model Architectures

A natural method to condition an RNN language model is to provide the network with the word being defined at the first step, as a form of “seed” information. The seed approach has been shown to be effective in RNNs for other tasks [38, 39]. Here, we follow the simple method of Sutskever et al., 2011 [88], in which the seed is added at the beginning of the text. In our case, the word being defined is added to the beginning of the definition. Note that we ignore the predicted probability distribution of the seed itself at test time.

Gated-update Model. Section 2.2 shows that definitions exhibit common patterns. We hypothesize that the word being defined should be given relatively more important for portions of the definition that carry semantic information, and less so for patterns or structures comprised of function and stop words. Further, Wen et al., 2015 [100] show that providing constant seed input at each time step can worsen the overall performance of spoken dialog generators.

Thus, inspired by the GRU update gate [17], we update the output of the recurrent unit with GRU-like update function as:

$$z_t = \sigma(W_z[w^*; h_t] + b_z) \tag{2.4}$$

$$r_t = \sigma(W_r[w^*; h_t] + b_r) \tag{2.5}$$

$$\tilde{h_t} = tanh(W_h[(r_t \odot w^*); h_t] + b_h) \tag{2.6}$$

$$h'_t = (1 - z_t) \odot h_t + z_t \odot \tilde{h_t} \tag{2.7}$$

where σ is the sigmoid function, $[a; b]$ denotes vector concatenation, and $\odot$ denotes element-wise multiplication. h_t from Equation 2.3 is **updated** as given in Equation 2.7. At each time step, z_t is an *update* gate controlling how much the output from RNN unit changes, and r_t is a *reset* gate controlling how much information from the word being defined is allowed (Figure 2.2). We name this model *Gated* (*G*).

Alternative Models. In the rest of this subsection, we present three baseline model architectures that remove portions of *Gated*. In our experiments, we will compare the

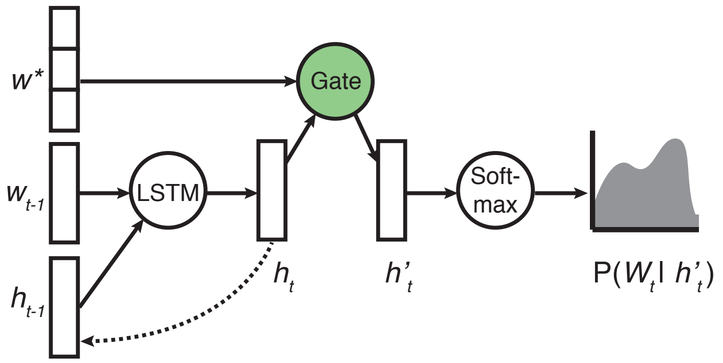

Figure 2.2. Architecture of the gated definition model. The gate unit controls influence of the word being defined.

performance of *Gated* against the baselines in order to measure the contribution of each portion of our architecture. First, we reduce the model into a standard RNNLM, where

$$h'_t = h_t = g(w_{t-1}, h_{t-1}) \tag{2.8}$$

The standard model *only* receives information about the word being defined, w^*, at the first step (from the seed). We refer to this baseline as *Seed* (S).

A straightforward way to incorporate the word being defined throughout the definition is simply to provide its embedding w^* as a constant input at every time step [57]. We refer to this model as *Input* (I):

$$h'_t = h_t = g([w^*; w_{t-1}], h_{t-1}) \tag{2.9}$$

Alternatively, the model could utilize w^* to update the hidden representation from the RNN unit, named *Hidden* (H). The update function for *Hidden* is:

$$h'_t = tanh(W_h[w^*; h_t] + b_h) \tag{2.10}$$

where W_h is a weight matrix, and b_h is the bias. In *Hidden* we update h_t from Equation2.3 using Equation 2.10. This is similar to the GRU-like architecture in Equation 2.7 without the gates (i.e. r_t and z_t are always vectors of 1s).

2.3.2. Other Features

In addition to model architectures, we explore whether other word features derived from the word being defined can provide complementary information to the word embeddings. We focus on two different features: affixes, and hypernym embeddings. To add these features within DM, we simply concatenate the embedding w^* with the additional feature vectors.

Affixes. Many words in English and other languages consist of composed morphemes. For example, a word "`capitalist`" contains a root word "`capital`" and a suffix "`-ist`". A model that knows the semantics of a given root word, along with knowledge of how affixes modify meaning, could accurately define any morphological variants of the root word. However, automatically decomposing words into morphemes and deducing the semantics of affixes is not trivial.

We attempt to capture prefixes and suffixes in a word by using character-level information. We employ a character-level convolution network to detect affixes. Specifically, the word being defined is represented as a sequence of characters with one-hot encoding. A padding character is added to the left and the right to indicate the start and end of the word. We then apply multiple kernels of varied lengths on the character sequence, and use max pooling to create the final features [42]. We hypothesize that the convolution input, denoted as *CH*, will allow the model to identify regularities in how affixes alter the meanings of words.

Hypernym Embeddings. As we discuss in Section 2.2, dictionary definitions often follow a structure of *genus* + *differentiae*. We attempt to exploit this structure by providing the model with knowledge of the proper *genus*, drawn from a database of Hypernym relations. In particular, we obtain the hypernyms from WebIsA database [84] which employs Hearst-like patterns [31] to extract hypernym relations from the Web. We then provide an additional input vector, referred to as *HE*, to the model that is equal to the weighted sum of the top k hypernyms in the database for the word being defined. In our experiments $k = 5$ and the weight is linearly proportional to the frequency in the resource. For example, the top 5 hypernyms and frequencies for "`fawn`" are "`color`":149, "`deer`":135, "`animal`": 132.0, "`wildlife`":82.0, "`young`": 68.0.

2.4. Experiments and Results

We now present our experiments evaluating our definition models. We train multiple model architectures using the *train* set and evaluate the model using the *test* set on all of the three tasks described in Section 2.2.3. We use the *valid* set to search for the learning hyper-parameters. Note that the words being defined are mutually exclusive across the three sets, and thus our experiments evaluate how well the models generalize to new words, rather than to additional definitions or senses of the same words.

All of the models utilize the same set of fixed, pre-trained word embeddings from the Word2Vec project,[4] and a 2-layer LSTM network as an RNN component [34]. The embedding and LSTM hidden layers have 300 units each. For the affix detector, the

[4]https://code.google.com/archive/p/word2vec/

character-level CNN has kernels of length 2-6 and size {10, 30, 40, 40, 40} with a stride of 1. During training, we maximize the log-likelihood objective using Adam, a variation of stochastic gradient decent [43]. The learning rate is 0.001, and the training stops after 4 consecutive epochs of no significant improvement in the validation performance. The source code and dataset for our experiment can be found at https://github.com/websail-nu/torch-defseq.

2.4.1. Definition Modeling

First, we compare our different methods for utilizing the word being defined within the models. The results are shown in the first section of Table 2.4. We see that the gated update (*S+G*) improves the performance of the *Seed*, while the other architectures (*S+I* and *S+H*) do not. The results are consistent with our hypothesis that the word being defined is more relevant to some words in the definition than to others, and the gate update can identify this. We explore the behavior of the gate further in Section 2.5.

Next, we evaluate the contribution of the linguistic features. We see that the *affixes* (*S+G+CH*) further improves the model, suggesting that character-level information can complement word embeddings learned from context. Perhaps surprisingly, the *hypernym embeddings* (*S+G+CH+HE*) have an unclear contribution to the performance. We suspect that the average of multiple embeddings of the hypernym words may be a poor representation the *genus* in a definition. More sophisticated methods for harnessing hypernyms are an item of future work.

Table 2.4. Perplexity evaluated on dictionary entries in the test set (lower is better).

Model	#Params	Perplexity
Seed	10.2m	56.350
S+I	10.6m	57.372
S+H	10.4m	58.147
S+G	10.8m	50.949
S+G+CH	11.1m	48.566
S+G+CH+HE	11.7m	**48.168**

2.4.2. Definition Generation

In this experiment, we evaluate the quality of the definitions generated by our models. We compute BLEU score between the output definitions and the dictionary definitions to measure the quality of the generation. The decoding algorithm is simply sampling a token at a time from the model's predicted probability distribution of words. We sample 40 definitions for each word being defined, using a temperature (τ in Equation 2.2) that is close to a greedy algorithm (0.05 or 0.1, selected from the *valid* set) and report the average BLEU score. For help in interpreting the BLEU scores, we also report the scores for three baseline methods that output definitions found in the training or test set. The first baseline, *Inter*, returns the definition of the test set word from the other dictionary. Its score thus reflects that of a definition that is semantically correct, but differs stylistically from the target dictionary. The other baselines (*NE-WN* and *NE-GC*) return the definition from the training set for the embedding nearest to that of the word being defined. In case of a word having multiple definitions, we micro-average BLEU scores before averaging an overall score.

Table 2.5 shows the BLEU scores of the generated definitions given different reference dictionaries. AVG and Merge in the table are two ways of aggregating the BLEU score.

AVG averages the BLEU scores by using each dictionary as the ground truth. The Merge computes score by using union of the two dictionaries. First, we can see that the baselines have low BLEU scores when evaluated on definitions from the other dictionary (*Inter* and *NE-*). This shows that different dictionaries use different styles. However, despite the fact that our best model *S+G+CH* is unaware of which dictionary it is evaluated against, it generates definitions that strike a balance between both dictionaries, and achieves higher BLEU scores overall. As in the earlier experiments, the *Gated* model improves the most over the *Seed* model. In addition, the *affixes* further improves the performance while the contribution of the *hypernym embeddings* is unclear on this task.

It is worth noting that many generated definitions contain a repeating pattern (i.e. "`a metal, or other materials, or other materials`"). We take the definitions from the language model (*Seed*) and our full system (*S+G+CH+HE*), and clean the definitions by retaining only one copy of the repeated phrases. We also only output the most likely definition for each word. The BLEU score increases by 2 (*Seed** and *S+G+CH+HE**). We discuss about further analysis and common error types in Section 2.5.

2.4.3. Reverse and Forward Dictionary

In the dictionary tasks, the models are evaluated by how well they rank words for given definitions (RVD) or definitions for words (FWD). We compare against models from previous work on the reverse dictionary task [32]. The previous models read a definition and output an embedding, then use cosine similarity between the output embedding and the word embedding as a ranking score. There are two ways to compose the output embedding: *BOW w2v cosine* uses vector addition and linear projection, and *RNN w2v*

Table 2.5. Equally-weighted BLEU scores for up to 4-grams, on definitions evaluated using different reference dictionaries (results are not comparable between columns).

Model	GC	WN	Avg	Merged
Inter	27.90	21.15	-	-
NE	29.56	21.42	25.49	34.51
NE-WN	22.70	**27.42**	25.06	32.16
NE-GC	**33.22**	17.77	25.49	35.45
Seed	26.69	22.46	24.58	30.46
S+I	28.44	21.77	25.10	31.58
S+H	27.43	18.82	23.13	29.66
S+G	30.86	23.15	27.01	34.72
S+G+CH	31.12	24.11	**27.62**	**35.78**
S+G+CH+HE	31.10	23.81	27.46	35.28
Additional experiments				
*Seed**	27.24	22.78	25.01	31.15
*S+G+CH+HE**	33.39	25.91	29.65	38.35
Random Emb	22.09	20.05	21.07	24.77

cosine uses a single-layer LSTM with 512 hidden units. We use two standard metrics for ranked results, accuracy at top k and R-Precision (i.e. precision of the top R where R is the number of correct definitions for the test word).

Table 2.6 shows that our models perform well on the dictionary tasks, even though they are trained to optimize a distinct objective (definition likelihood). However, we note that our models have more parameters than those from previous work. Furthermore, we find that *RNN w2v cosine* performs better than *BOW w2v cosine*, which differs from the previous work. The differences may arise from our distinct preprocessing described in Section 2.2, i.e. redundant definitions are split into multiple definitions. We omit the information retrieval approach baseline because it is not obvious how to search for unseen words in the test set.

Table 2.6. Model performance on Reverse (RVD) and Forward (FWD) Dictionary tasks.

Model	#Params	RVD		FWD
		@1	@10	R-Prec
BOW w2v cosine	0.09m	0.106	0.316	-
RNN w2v cosine	1.82m	0.190	0.452	-
Seed	10.2m	0.175	0.465	0.163
S+I	10.6m	0.187	0.492	0.169
S+H	10.4m	0.286	0.573	0.282
S+G	10.8m	0.293	0.581	0.282
S+G+CH	11.1m	0.307	0.600	0.298
S+G+CH+HE	11.7m	**0.308**	**0.608**	**0.304**

2.5. Discussion

In this section, we discuss our analysis of the generated definitions. We first present a qualitative evaluation, followed by an analysis on how the models behave. Finally, we discuss error types of the generated definitions and how it might reflect information captured in the word embeddings.

2.5.1. Qualitative Evaluation and Analysis

First, we perform a qualitative evaluation of the models' output by asking 6 participants to rank a set of definitions of 50 words sampled from the test set. For each word w, we provide in random order: a ground-truth definition for w (*Dictionary*), a ground-truth definition of the word w' whose embedding is nearest to that of w (*NE*), the standard language model (*Seed**), and our full system (*S+G+CH+HE**). Inter-annotator agreement was strong (almost all inter-annotator correlations were above 0.6). Table 2.7 shows that definitions from the *S+G+CH+HE** are ranked second after the dictionary definitions, on average. The advantage of *S+G+CH+HE** over *Seed** is statistically significant ($p < 0.002$, t-test), and the difference between *S+G+CH+HE** is and *NE* is borderline significant ($p < 0.06$, t-test).

Table 2.7. Percentage of times a definition is manually ranked in each position (@k), and average rank (Avg).

Choices	@1	@2	@3	@4	Avg
Dictionary	58.3	21.9	7.72	10.1	1.64
NE	16.3	22.8	27.85	37.0	2.75
*Seed**	6.8	23.5	35.23	35.1	2.92
*S+G+CH+HE**	18.7	31.8	29.19	17.8	2.41

All of our results suggest that the gate-based models are more effective. We investigate this advantage by plotting the average gate activation (z and r in Equation 2.4 and 2.5) in Figure 2.3. The r gate is split into 3 parts corresponding to the embedding, character information, and the hypernym embedding. The figure shows that the gate makes the output distribution more dependent on the word being defined when predicting content words, and less so for function words. The hypernym embedding does not contribute to the performance and its gate activation is relatively constant. Additional examples can be found in the supplementary material.

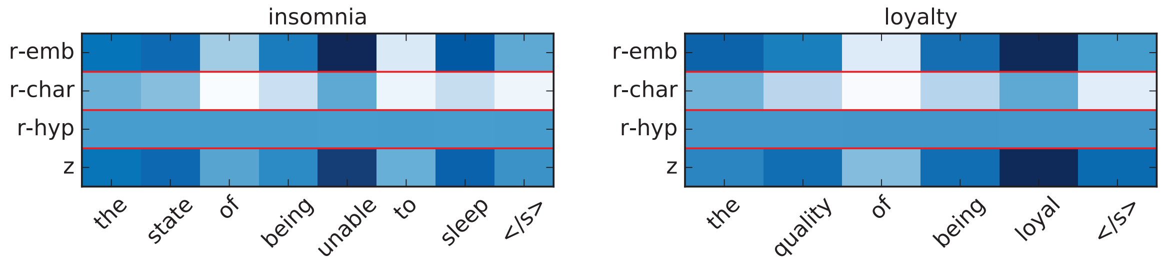

Figure 2.3. Average gate activations for tokens of two definitions (omitting seed). The model utilizes the word being defined more for predicting content words than for function words.

Finally, we present a comparison of definitions generated from different models to gain a better understanding of the models. Table 2.8 shows the definitions of three words from Table 2.1. The *Random Embedding* method does not generate good definitions. The nearest embedding method *NE* returns a similar definition, but often makes important errors (e.g., "`feminine`" vs "`masculine`"). The models using the gated update function generate better definitions, and the character-level information is often informative for selecting content words (e.g. "`mathematics`" in "`mathematical`").

Table 2.8. Selected examples of generated definitions from different models. We sample 40 definitions for each word and rank them by the predicted likelihood. Only the top-ranked definitions are shown in this table.

Model	creek	feminine	mathematical
Random Emb	to make a loud noise	to make a mess of	of or pertaining to the middle
NE	any of numerous bright translucent organic pigments	a gender that refers chiefly but not exclusively to males or to objects classified as male	of or pertaining to algebra
Seed	a small stream of water	of or pertaining to the fox	of or pertaining to the science of algebra
S+I	a small stream of water	of or pertaining to the human body	of or relating to or based in a system
S+H	a stream of water	of or relating to or characteristic of the nature of the body	of or relating to or characteristic of the science
S+G	a narrow stream of water	having the nature of a woman	of or pertaining to the science
S+G+CH	a narrow stream of water	having the qualities of a woman	of or relating to the science of mathematics
S+G+CH+HE	a narrow stream of water	having the character of a woman	of or pertaining to the science of mathematics

2.5.2. Error Analysis

In our manual error analysis of 200 labeled definitions. We find that 140 of them contain some degree of error. Table 2.9 shows the primary error types, with examples. Types (1) to (3) are fluency problems, and likely stem from the definition model, rather than shortcomings in the embeddings.

Table 2.9. Error types and examples.

Word	Definition
(1) Redundancy and overusing common phrases: 4.28%	
propane	a volatile flammable gas that is used to burn gas
(2) Self-reference: 7.14%	
precise	to make a precise effort
(3) Wrong part-of-speech: 4.29%	
accused	to make a false or unethical declaration of
(4) Under-specified: 30.00%	
captain	a person who is a member of a ship
(5) Opposite: 8.57%	
inward	not directed to the center
(6) Close semantics: 22.86%	
adorable	having the qualities of a child
(7) Incorrect: 32.14%	
incase	to make a sudden or imperfect sound

We believe the other error types stem more from semantic gaps in the embeddings than from limitations in the definition model. Our reasons for placing the blame on the embeddings rather than the definition model itself are twofold. First, we perform an ablation study in which we train *S+G+CH* using randomized embeddings, rather than the learned Word2Vec ones. The performance of the model is significantly worsened (the test perplexity is 100.43, and the BLEU scores are shown in Table 2.5), which shows that the good performance of our definition models is in significant measure due to the embeddings. Secondly, the error types (4) - (6) are plausible shortcomings of embeddings, some of which have been reported in the literature. These erroneous definitions are partially correct (often the correct part of speech), but are missing details that may not appear in contexts of the word due to reporting bias [30]. For example, the word “captain” often appears near the word “ship”, but the *role* (as a leader) is frequently implicit. Likewise, embeddings are well-known to struggle in capturing antonym relations [1], which helps explain the opposite definitions output by our model.

2.6. Related Work

The goal of this work is to investigate RNN-based models that learns to synthesize dictionary definitions from word embeddings. In NLP community, dictionary corpora have been utilized extensively. However, to the best of our knowledge none of the previous work has attempted to create a generative model of definitions.

Early work focused on *extracting* semantic information from the dictionary definitions. For example, Chodorow, 1985 [19], and Klavans and Whitman, 2001 [44] constructed a taxonomy of words from dictionaries. Dolan et al., 1993 [24] and Vanderwende et al., 2005 [95] extracting semantic representations from the dictionary definitions, to populate a lexical knowledge base. Rather than extracting information from the definitions, we are using the definitions as a training data to reveal encoded information in the word embeddings.

Recently, dictionary definitions have been used to learn such word embeddings. For example, Wang et al., 2015 [98] used words in definition text as a form of “context” words for the Word2Vec algorithm [59]. Hill et al. 2016 [32] use dictionary definitions to model compositionality, and evaluate the models with the reverse dictionary task. While these works learn word or phrase embeddings from definitions, we only focus on generating definitions from existing (fixed) embeddings. Nonetheless, experiments show that our models outperform those of Hill et al., 2016 [32] on the reverse dictionary task.

Our work employs embedding models for natural language generation. A similar approach has been taken in a variety of recent work on tasks distinct from ours. Dinu and Baroni, 2014 [23] present a method that uses embeddings to map individual words to longer phrases denoting the same meaning. Likewise, Li et al., 2015 [48] study how to encode a paragraph or document as an embedding, and reconstruct the original text from the encoding. Other recent work such as the image caption generation [39] and spoken dialog generation [99] are also related to our work, in that a sequence of words is generated from a single input vector. Our model architectures are inspired by sequence-to-sequence models [17, 89], but definition modeling is distinct, as it is a *word*-to-sequence task.

2.7. Conclusion

In this chapter, we introduce the definition modeling task, and investigate whether word embeddings can be used to generate definitions of the corresponding words. We evaluate different architectures based on a RNN language model on definition generation and the reverse and forward dictionary tasks. We find the gated update function that controls the influence of the word being defined on the model at each time step improves accuracy, and that a character-level convolutional layer further improves performance. Our error analysis shows a well-trained set of word embeddings is crucial to the models, and that some failure modes of the generated definitions provide insight into shortcomings of the word embeddings.

The natural next step is to investigate whether definition models can be utilized to improve word embeddings or standard language models. This might involve extending the model to handle polysemy and context-dependent embeddings. We have seen a few work in this direction [91, 66, 8]. However, as we discuss in section 2.4.2, the most apparent errors are not of semantic errors, but of fluency issues (e.g. repetitions). This leads us to the next chapter of this dissertation.

CHAPTER 3

Controlling Neural Language Model Behavior

Often, we would like to subject the text generated by a recurrent neural network language model (RNNLM) to constraints, in order to overcome systemic errors (e.g. word repetition discussed in the previous chapter) or achieve application-specific goals (e.g. more positive sentiment). In this chapter, we present a method for training RNNLMs to simultaneously optimize likelihood and follow a given set of statistical constraints on text generation.[1]

The problem is challenging because the statistical constraints are defined over aggregate model behavior, rather than model parameters, meaning that a straightforward parameter regularization approach is insufficient. We solve this problem using a dynamic regularizer that updates as training proceeds, based on the generative behavior of the RNNLMs. Our experiments show that the dynamic regularizer outperforms both generic training and a static regularization baseline. The approach is successful at improving word-level repetition statistics by a factor of four in RNNLMs on a definition modeling task. It also improves model perplexity when the statistical constraints are n-gram statistics taken from a large corpus.

[1]The main contributions in this chapter were first described and published in *Controlling Global Statistics in Recurrent Neural Network Text Generation* [67].

3.1. Introduction

Recurrent neural network language models are a critical component of many natural language generation tasks such as machine translation, summarization, automated conversation, and caption generation [2, 40, 49, 82, 89]. The models are trained to maximize the likelihood of a training corpus, and evaluated on the likelihood they assign to a held-out test corpus (measured in terms of perplexity). RNNLMs, and in particular Long Short-term Memory Networks (LSTM) [34], have provided dramatic improvements in the perplexities of language models in recent years [37, 54].

However, while RNNLMs optimize well on the perplexity metric, when we utilize the models to generate text we often wish to encourage the models to obey additional statistical constraints. For example, one way in which RNNLM text generators tend to fail is by repeating the same words or phrases too often, leading to nonsensical output [68, 70, 83]. Can we learn a low-perplexity model that avoids this failure mode? Likewise, we may want to adjust our model's output distribution to not reflect undesirable biases in our training corpus, or to utilize a different style, such as shorter sentences or more positive sentiment.

In this work, we present a regularization technique that allows modelers to specify additional *soft constraints* on language models during the training. For example, the constraints might encourage the model to repeat words less often, or to use shorter sentences, etc. The constraints are stated as a *reference distribution* that gives target marginal probability values for events in the text (e.g., the probability that a certain word will repeat consecutively). The regularizer encourages the global statistics of the model to match the reference distribution. Implementing the regularizer in RNNLMs is challenging because

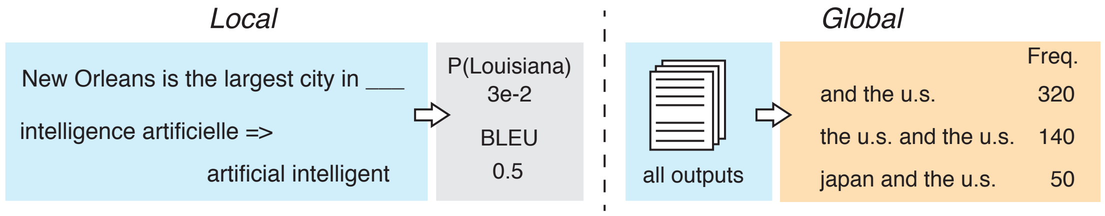

Figure 3.1. Global statistics such as n-gram frequency measures overall behavior of a model while local metrics measures performance per instance.

the marginal output probabilities of an RNNLM do not correspond directly to parameters of the model (as they do in simpler n-gram models), and instead must be inferred. In this work, we solve this problem by computing estimates of the model's marginals from a sample of generated text, which is continually updated as training proceeds. We then use the estimates during training to encourage the model to generate text that matches the reference distribution by minimizing the KL-divergence. We refer to this method as *Dynamic KL Regularization.*

To evaluate our approach, we experiment with two types of constraints in RNNLMs. The first type aims to reduce local repetition. Local repetition is a long-standing issue with RNNLMs: the models disproportionately repeat the same word within a short window (see Figure 3.2). The problem becomes particularly acute when we ask the RNN to generate its estimate of the *high-likelihood* text for a given input. We show how using dynamic KL regularization to encourage the RNNLM to exhibit a similar repetition profile to the training data can reduce repetition with little harm to perplexity. The second type of constraints attempt to match n-gram statistics from a reference corpus. Training an RNNLM on even tens of millions of tokens can be computationally costly, but often we can readily acquire n-gram statistics over even billions of tokens. We show how dynamic KL regularization can be used to incorporate large-corpus statistics that improve an RNNLM trained on a smaller corpus from the same distribution.

Sampled text from PTB model
... between ***the u.s. and the u.s. and*** <unk> agencies ...
... closed higher in ***paris paris paris paris*** and zurich ...
... the exxon ***aerospace*** and ***aerospace*** firm is n't ...

Sampled text from WordNet model	
samurai:	a ***Japanese Japanese Japanese Japanese*** warrior
vintner:	a person who ***wine wine***
papal:	***associated with or associated with or*** belonging to the papacy

Figure 3.2. Examples of repetitions generated by recurrent neural network language models trained with PTB text and the definition model trained with WordNet definitions.

The rest of the chapter proceeds as follows. We first derive our regularizer from the KL-divergence between the reference distribution and the model distribution. We then present dynamic KL regularization, an approximation to our regularizer that is differentiable and uses estimates of marginal probabilities from text generated by the model. Our experiments compare dynamic KL regularization with three baselines, and show that the proposed regularization is better at matching repetition and n-gram statistics on the Penn Treebank dataset. Furthermore, we show how our method can use statistics from the larger WikiText-103 corpus [56] to improve an RNNLM trained on a more tractable small corpus from the same distribution (WikiText-2). Finally, in the last set of experiments is an application of our approach to reduce repetition for a conditional language generation task. We choose definition modeling as a benchmark (see the previous chapter for detail). We find that dynamic KL regularization improves repetition statistics by a factor of four, and also improves BLEU score [15, 69]. Finally, we provide analysis, review related work, and conclude.

3.2. Dynamic KL Regularization

Our goal is to train a recurrent neural network language model such that the global statistics of its generated text are similar to a specified set of statistical soft constraints. Each constraint applies to the marginal probability $P(w, c)$, where w is a word and c is a *condition* – an event specified over the context up to and including the word. For each constraint, we specify two quantities, $P_0(c)$ and $P_0(w|c)$ to be defined as a constraint $P_0(w, c) = P_0(w|c)P_0(c)$. As a simple example, if we wish to constrain the probability of "`the dog`", we would define $P_0(w_i = \texttt{dog}|w_{i-1} = \texttt{the})$ and $P_0(w = \texttt{the})$. The distribution $P_0(\mathcal{W}, \mathcal{C})$ is denoted as the *reference* distribution, where $\mathcal{W}$ is a vocabulary set and $\mathcal{C}$ is a set of conditioning events.

Recall that an RNNLM defines a probability distribution over words conditioned on previous words as the following:

$$P_\theta(w_t|w_{1:t-1}) = P_\theta(w_t|h_t; \tau) \propto \exp(\theta_o^{(i)} h_t/\tau) \tag{3.1}$$

$$h_t = g(h_{t-1}, w_{t-1}; \theta) \tag{3.2}$$

where $w_{1:t-1}$ is a sequence of previous words, τ is a temperature (1.0 unless specified), θ denotes the parameters of the model, $\theta_o^{(i)} \subset \theta$ is a set of weights associated with output word i, and $g(\cdot)$ is a recurrent function such as an GRU or LSTM unit [18, 34]. When training on a sequence of ground truth tokens, RNNLMs are optimized to maximize the log likelihood of all tokens in the training data, or equivalently to maximize the summed log likelihood of each token given the previous ones.

Our goal is to train an RNNLM to generate text with statistics similar to the reference distribution, while simultaneously achieving high likelihood on the training data. In principle, matching the reference distribution entails minimizing the KL-divergence from the reference distribution P_0 to the model distribution P_θ. Specifically, we would like to minimize the KL-divergence as the following:

$$\mathcal{R}(\theta, P_0) = \mathop{\mathbb{E}}_{c \sim P_\theta(\mathcal{C})}[D_{KL}(P_\theta(\mathcal{W}|c)||P_0(\mathcal{W}|c)] + D_{KL}(P_\theta(\mathcal{C})||P_0(\mathcal{C})) \tag{3.3}$$

The first term of $\mathcal{R}$ defines a divergence between the model and the reference conditional distribution of words, and the second term defines the same measure for the distribution of the conditioning events. Since we are minimizing log-likelihood of the training data, we decide to minimize $\mathcal{R}(\theta, P_0)$ using conditioning events with respect to the training data. Our loss function is then:

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} -\log P_\theta(w_t|h_t) + \alpha \mathcal{R}(w_{t-l:t}) \tag{3.4}$$

$$\mathcal{R}(w_{t-l:t}; \theta, P_0) = \sum_{c_t \in \mathcal{K}_t} D_{KL}(P_\theta(\mathcal{W}|c_t)||P_0(\mathcal{W}|c_t)) + D_{KL}(P_\theta(\mathcal{C})||P_0(\mathcal{C})) \tag{3.5}$$

where $\mathcal{K}_t \subset \mathcal{C}$ is a set of conditioning events occurring in a truncated sequence $w_{t-l:t}$ (i.e. $\{w_{t-1} = \texttt{the}\}$ in our example). α is a weight that controls how strongly the objective favors matching the reference distribution versus maximizing the log likelihood of the training corpus. The sequence is truncated because we only need as many previous tokens as the conditions require. Furthermore, when training with truncated back-propagation through time, we omit any event occurring before the current training sequence.

However, the loss function in Equation 3.5 is challenging to compute exactly. In a classical n-gram model, obtaining a marginal distribution (i.e. $P_\theta(\mathcal{W}|c)$ or $P_\theta(\mathcal{C})$) would be straightforward. These quantities are parameters of an n-gram model. However, in state-of-the-art RNNLM language models, the situation is more complex. An advantage of RNNLMs is that their output distributions incorporate arbitrarily long context, via the hidden state. But, this makes it difficult to infer marginal probabilities, because doing so requires summing over all possible preceding contexts.

On the other hand, given a sufficiently large corpus of text generated by an RNNLM, we can easily estimate the model's marginal probabilities by counting. This sampling-based approach has the added benefit that our estimates reflect the model's actual behavior during inference – i.e. the estimates account for *exposure bias*, the fact that models are never exposed to their own generated text at training time, only at inference time [4, 74]. However, we need to sample output from the model to accumulate a large body of generated text, and perform maximum-likelihood estimation over the text (counting) to obtain accurate statistics. This process makes Equation 3.5 not differentiable and prohibits gradient-based training algorithms.

We propose an approach based on an approximation of the KL-divergence computed in Equation 3.5 by dynamically updating an estimate of the model's long-run inference behavior during the training. Specifically, we use maximum-likelihood estimate $\hat{P}_\theta$ of the marginal probabilities under the model from its generated text, and compute the KL-divergence terms in Equation 3.5 using both of these marginals and the model's current prediction at each time step t:

$$\hat{D}_{KL}(P_\theta(\mathcal{C})||P_0(\mathcal{C})) = \sum_{c\in\mathcal{C}} P_\theta(c|h_{t-l:t})\log\frac{\hat{P}_\theta(c)}{P_0(c)} \tag{3.6}$$

$$\hat{D}_{KL}(P_\theta(\mathcal{W}|c_t)||P_0(\mathcal{W}|c_t)) = \sum_{w\in\mathcal{W}} P_\theta(w|h_t)\log\frac{\hat{P}_\theta(w|c_t)}{P_0(w|c_t)} \tag{3.7}$$

Intuitively, Equation 3.6 and 3.7 minimize expected log-likelihood ratio between the model's estimate and the reference distribution – decreasing the current predicted probability if the log ratio is positive (i.e. if we over-generate) and increasing the predicted probability if the log ratio is negative. In other words, the log-likelihood ratio is a constant (non-differentiable) providing feedback to the model current prediction (differentiable).

Of course, the model's marginal probabilities will change as training proceeds. Thus, as we train, we sample new sequences from the model, dynamically updating the log-likelihood ratio. We label this as ***dynamic***. However, we need a large sample size to accurately estimate the model's marginals ($\hat{P}_\theta$). To efficiently compute the model's marginals, we keep a fixed-size pool of the generated text. As training proceeds, we generate a small portion of the text to replace the oldest tokens, and update the model's marginals every few steps (Figure 3.3).

3.2.1. Alternate approaches

As discussed above, we infer marginal probabilities of the model by using overall statistics from the model output text. Thus the regularization requires only the aggregated statistics to match the reference distribution. On the other hand, one might argue that similar effect can be achieved by encouraging the model prediction to match the reference distribution

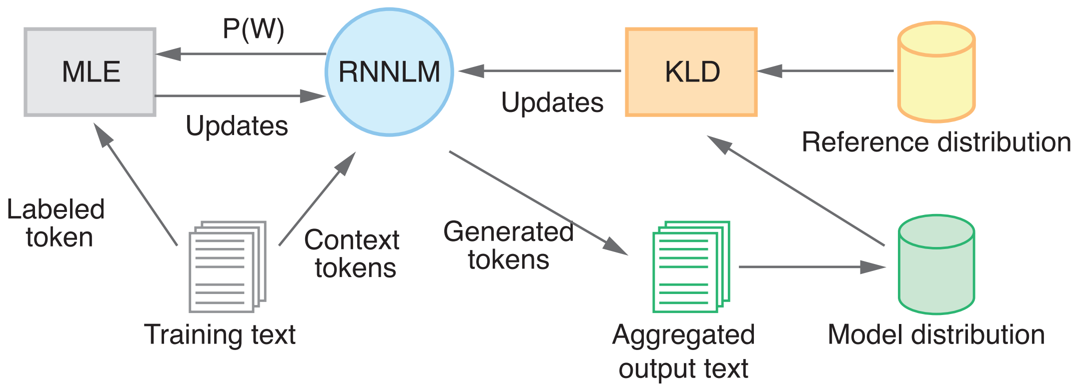

Figure 3.3. Dynamic KL regularization computes statistics from the model's generated output and adjusts the model parameters according to the divergence between the desired statistics and the model statistics.

(for every context). To justify our choice to sample, we compare against a baseline that uses the current model's output distribution directly for the regularization. This is equivalent to replacing $\hat{P}_\theta$ in Equation 3.6 and 3.7 with P_θ. We refer to this alternative as ***static***, as opposed to dynamically updating statistics from the generated text.

We define our soft constraints as a joint probability of a condition and a word event – but in some settings, we may not need to regularize the conditions themselves, only the word event. For example, we may simply want to increase the frequency of "`mr. robot`", regardless of how many times the model generates the first condition token "`mr.`". Further, in some cases, computing the probability of the conditioning events from the model might not be straightforward or computationally expensive as Equation 3.6 requires probability distribution over conditioning events. This leads us to another alternate approach that constrains only the conditional probability $P(\mathcal{W}|c)$. That is, we remove the KL-divergence term of the conditioning event distribution (the last term of Equation 3.5). We refer to this modified regularization as ***dynamic-$\boldsymbol{P(\mathcal{C})}$***.

3.2.2. Examples of statistical constraints

Local repetition statistics. A common problem in generated text from RNNLMs is local repetition, where the same substring repeats multiple times in a short output text. For example, one definition of "`fairness`" sampled from an unregularized definition model is "`the property of being fair and fair.`" As our first type of soft constraints, we regularize the model such that the probability of words appearing again within a window of tokens is close to that in a given reference text corpus. To construct the constraints, for each $k \in \{1, 2, 3\}$ we define a conditioning event to be that some word repeats after k tokens, i.e. $\mathcal{C} = \{w_{i-k} = w_i : k \in \{1, 2, 3\}\}$. We compute the marginal probabilities as:

$$P(c^k) = \sum_{w \in \mathcal{W}} N_r(w, k)/|\mathcal{W}| \tag{3.8}$$

$$P(w|c^k) = N_r(w, k)/ \sum_{w' \in \mathcal{W}} N_r(w', k) \tag{3.9}$$

where $N_r(w, k)$ is the number of times w re-occurs after k tokens. For example, a probability of "`the`" repeats after 2 tokens is $freq(\texttt{the} * * \ \texttt{the})/\sum_w freq(w * * w)$. We apply Witten-Bell estimate [16] to smooth the distribution to allow a query for unobserved events of both model's and reference marginals. During the training time, the model probability of the conditioning event is then the current probability of previous k words:

$$P_\theta(c^k|h_{t-l:t}) = P_\theta(w_{t-k}|h_t) \tag{3.10}$$

Using n-gram statistics. Another type of the soft constraints is the n-gram distribution where the set of conditioning events is phrases of length $n - 1$. These constraints can correct disproportionate n-gram frequencies in the generated text, and can also be used to incorporate statistics from a larger corpus. The marginal probabilities for a conditioning phrase $c = w'_{1:n-1}$ are then:

$$P(c) = P(w'_{1:n-1}) \tag{3.11}$$

$$P(w|c) = P(w_i = w | w_{i-n:i-1} = w'_{1:n-1}) \tag{3.12}$$

From the reference or sampled text, we obtain these marginal probabilities using Kneser-Ney n-gram language models [45]. During training time, the model probability of the conditioning event is simply the current output probability:

$$P_\theta(c|h_{t-l:t}) = \prod_{k=0}^{n-1} P_\theta(w_{t-k}|h_{t-k}) \tag{3.13}$$

For simplicity of the experiments, we use only bigram constraints in this work.

3.3. Experiments and Results

We now present our experiments evaluating dynamic KL regularization. We begin by providing a comparison between baselines and our regularization on a common benchmark for language modeling, a preprocessed version of Penn Treebank (PTB).[2] Then, we present our results on practical usage of the regularization on two other datasets for language modeling (WikiText) [56] and the definition modeling trained with definitions of lemmas in WordNet [62].

For language models, we use a 2-layer LSTM with 650 hidden units. The embeddings and output logit weights are tied and have 650 units. We adopt the training hyper-parameters from Zaremba et al 2015 [112]. Specifically, we use stochastic gradient descent

[2]http://www.fit.vutbr.cz/ imikolov/rnnlm

with the standard dropout rate of 50%. The initial learning rate is 1.0, with a constant decay rate of 0.8 starting at the 6th epoch. Perplexity validation stops significantly improving after around 20 epochs. For definition models, we use the same settings described in the previous chapter. Since a model will generate a uniform distribution before any training, we pre-trained all models for 5 epochs before applying the regularization to save training time.

As for hyper-parameters for the regularization, we did a limited exploration and use the following settings throughout the experiments. The weight α on the regularization is set to be 1.0 for repetition constraints and 0.5 for bigram constraints. In addition, the log-likelihood ratio in the approximated KL-divergence (Equation 3.6 and 3.7) is clipped at -2.0 to 2.0. This helps reduce variance from the reference distribution and the model's distribution from the generated text. Finally, text is generated every 100 steps to update the model's marginal distribution in an amount equal to 10% of the reference text. The implementation is publicly available.[3]

3.3.1. Reducing Sampling Variance

We need a large generated text to infer the model global behavior, but the parameters of the model is also updated during training. Generating a reasonable-size text every time the model is updated is not only computationally intensive, but also introduces high variance to the statistics. We could intermittently generate a small piece of text and add it to a single pool while keep a moving average of the statistics. However, this creates a feedback-control loop between the statistics and the regularization because the estimate of

[3]https://github.com/northanapon/seqmodel/tree/aaai18

statistics is highly correlated with the regularization. For example, the regularizer could try to match probability of "`robot`" by overly increasing it, and the model generated too many "`robot`" for that small piece of text. Then the regularizer would heavily decrease the probability more than it should.

In order to cope with this, we adopt techniques from reinforcement learning literature [64]. First we keep a fixed-size history of generated texts, this can be 25-50 pieces (each 100 to 200 thousands tokens). After a few training updates, we replace the oldest entry with a newly generated one (this can be done asynchronously along with the training). To get a new global statistics, we combine the recently generated text with text entries randomly drawn from the history. Finally we add the new estimate to current moving average to lower its variance and create a momentum effect.

3.3.2. Baseline Comparison

In the first set of experiments, we compare our regularization with standard (*unregularized*) training on language modeling using Penn Treebank (PTB). We present all comparisons with both repetition constraints where $k = 1, 2$ and 3, and bigram constraints discussed in the previous section. The statistical constraints are computed from the training data of PTB.

To measure performance, we compute total absolute frequencies *mismatch* between the reference text and the generated text:

$$\sum_{w,c \in \mathcal{W},\mathcal{C}} |N_0(w,c) - N_\theta(w,c)|$$

where $N(w, c)$ is the frequency of an event w, c in the reference corpus (N_0) or generated text (N_θ). This error measure shows how much a model's generated text differs from the reference distribution.

First, we present our results on repetition constraints. Table 3.1 shows the total mismatch of words repeated after k tokens. We can see that *dynamic* can reduce the number of mismatches in word repetition significantly compared to *unregularized* (by more than 50%) at all k, and *dynamic*$-P(\mathcal{C})$ has essentially the same error, while *static* does not work well. This suggests that we do need to sample to estimate the marginals from the model. Interestingly, the mismatch of *static* is due to under-repetition. Finally, we can see that the regularization has only a slight impact on the test perplexity.

Table 3.1. Performance of models trained using PTB and regularized with word repetition statistics. Dynamic KL regularization significantly reduce the mismatch in word repetition comparing to training without the regularization. In parentheses, percentage changes are computed relative to *unregularized*.

	Test	Absolute frequency mismatch					
	PPL	k=1		k=2		k=3	
unregularized	77.6	7.07k	%	6.25k	%	8.30k	%
dynamic	78.3	3.00k	(58%)	**2.91k**	**(53%)**	**3.89k**	**(53%)**
dynamic$-P(\mathcal{C})$	78.3	**2.99k**	**(58%)**	3.11k	(50%)	4.67k	(44%)
static	79.9	8.08k	14%	6.20k	(1%)	8.44k	2%

For bigram constraints, we measure absolute error in terms of the frequency mismatch of unigrams and bigrams. The result in Table 3.2 shows that *dynamic* has the lowest error, but the reduction from *unregularized* is less than what we observed in the previous experiments (a 23% and 7% reduction). This could be due to the larger number of constraints we specify ($\sim$ 100k vs. only 200, after filtering out singletons). Again, *dynamic*-$P(\mathcal{C})$ works slightly less well than *dynamic*, and *static* does not work.

Table 3.2. Performance of models trained using PTB and regularized with bigram statistics. A model trained with dynamic KL regularization has similar perplexity, and generates text with unigram and bigram frequency closer to the training corpus than an unregularized model.

	Test PPL	Absolute frequency mismatch unigram		bigram	
unregularized	77.6	163k	%	673k	%
dynamic	77.8	**125k**	**(23%)**	**623k**	**(7%)**
dynamic$-P(\mathcal{C})$	77.7	147k	(9%)	640k	(5%)
static	79.8	157k	(4%)	754k	12%

3.3.3. Larger corpus statistics

We now demonstrate how to use bigram statistics from a larger corpus to regularize a model trained on a similar, but smaller corpus. For this, we choose WikiText corpora containing WikiText-103 (large) and WikiText-2 (small). Note that WikiText-103 training data does not contain Wikipedia articles that overlap with validation and testing data on WikiText-2. Since WikiText-103 has a much larger vocabulary size, when computing bigram constraints, we use vocabulary from WikiText-2 and set out-of-vocab words to the unknown symbol. The results in Table 3.3 shows *dynamic-103* model has better perplexity than *unregularized* and *dynamic-2* models.

Since the reference corpora differ in this experiment, our mismatch measure from the previous experiments is not meaningful. Instead, we measure the KL-divergence of the unigram and bigram distributions to evaluate the efficacy of our constraints. Table 3.3 shows the KL-divergence from the generated texts to WikiText-103's training data. The *dynamic-103* model is exposed to bigram statistics from WikiText-103, and has the lowest KL-divergence among the generated texts.

Table 3.3. Performance of models trained using WikiText-2, regularized using statistics of the training data from Wikitext-2 and WikiText-103 respectively.

	Test PPL	KL-divergence unigram		bigram	
unregularized	91.6	0.121	%	1.785	%
dynamic-2	91.4	0.111	(8%)	1.733	(3%)
dynamic-103	**86.8**	**0.072**	**(40%)**	**1.586**	**(11%)**

3.3.4. Definition generation

Finally, we evaluate our method on a definition modeling. Definition modeling serves as a benchmark for a simple conditional language generation task. Certain sequence-to-sequence models have an existing solution to repetition by encouraging models to attend to different positions of an input sequence [56, 92]. However, this approach cannot be applied in word-to-sequence models, as there is only one input token.

We take all lemma definitions from WordNet and split into training, validating, and testing data. We experiment with repetition constraints from the training data. A common use case of a language generation model is to find a high likelihood sequence. To test whether the proposed regularizer holds up in this setting, we use the greedy algorithm to generate a definition for each word in the training data and compare the absolute repetition frequency mismatch, as in our first set of experiments. Note that the test perplexities are always computed under normal temperature. We compute BLEU score as a measure of output definition quality in the Greedy setting. We omit BLEU scores from the Sample setting, because sampled text under a model's distribution is often not a high likelihood sequence that we would compare with the reference texts.

Table 3.4 shows the results of the repetition constraints experiment. Consistent with the baseline comparison, *dynamic* has lower mismatch compared to *unregularized*. Again, the perplexity is similar between the two models. The repetition problem becomes more severe when the text is being generated greedily. In the later part of the table, we can see that mismatch of repetitions increases for both *unregularized* and *dynamic*. However, *dynamic* still has much lower mismatch, especially for tokens that repeat immediately. In addition, *dynamic* results in an increase in BLEU score from *unregularized*.

Table 3.4. Performance of models training using WordNet definitions and regularized with word repetition statistics. **Top**: A model trained with our regularization maintains the same perplexity, but generate significantly less repetition. **Bottom**: The definitions are greedily generated from the same models with one additional model. The regularized model generates less repetition under low temperature and has slightly improved BLEU score. In addition, regularized with statistics from low temperature results in similar performance.

	Test PPL	Test BLEU	Absolute frequency mismatch k=1		k=2		k=3	
Sample Generation								
unregularized	48.0	N/A	1.74k	%	4.78k	%	4.28k	%
dynamic	48.0	N/A	**0.41k**	**(76%)**	**2.99k**	**(37%)**	**2.33k**	**(46%)**
Greedy Generation								
unregularized	48.0	18.5	10.73k	%	34.96k	%	36.90k	%
dynamic	48.0	**19.1**	0.72k	(93%)	13.70k	(61%)	14.37k	(61%)
dynamic-greedy	48.8	18.9	**0.69k**	**(94%)**	**11.16k**	**(68%)**	**8.87k**	**(76%)**

Dynamic KL regularization can also be adapted to specify constraints over greedily generated text, rather than sampled text. To test this, we trained another model with the same repetition constraints as *dynamic*, but that generates in a near-greedy mode (with temperature of 0.1) during training. As shown in the last row of the table, *dynamic-greedy* can drive the repetition mismatch during the greedy generation further down with slightly worse perplexity (under the usual distribution, $\tau = 1.0$) and BLEU score.

3.4. Discussion

In this section, we discuss and analyze dynamic KL regularization. We discuss common types of local repetition of the generated definitions and show a few examples that the regularizer solves. Then, we note on the computation cost associated with the regularizer. Finally, we present a preliminary experiment where we have a small set of constraints that conflict with the training data.

3.4.1. Local repetition and n-gram duplication

The results in Table 3.4 show a total reduction of repeated words in local context windows. We find this indirectly also reduces the number of duplicate n-grams within a definition. Figure 3.4 shows percentage of n-gram duplicate within a definition from the test data generated greedily from the models. We can see that models trained with the regularizer generate definitions with fewer duplicate n-grams. Note that some part of the reduction is due to the generated definitions being shorter (8.6 vs 8.0 tokens per output sequence).

For further qualitative analysis, we identify common cases of local repetition and provide examples of improvement in the regularized model as shown in Figure 3.5. We notice that the most common cases of local repetition come after a conjunction, especially "`or`". This could be due to the fact that a word that follows a conjunction like "`or`" or "`and`" often has a meaning similar to the word before the conjunction, and thus the model's

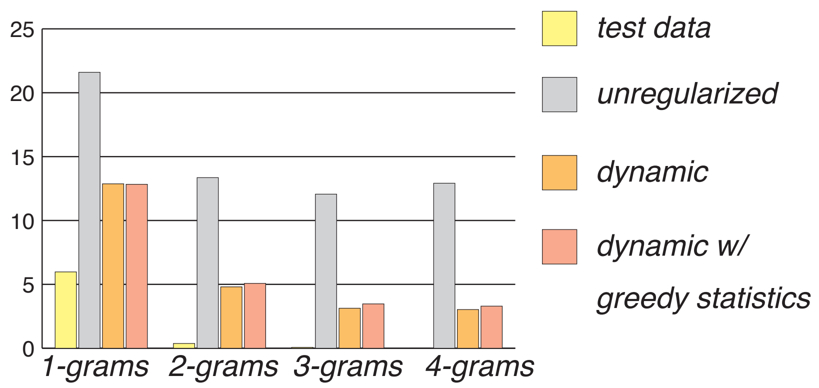

Figure 3.4. Percent of duplicate n-grams within a definition in test data and definitions generated greedily from models. Reduction in local repetition also reduces n-gram duplication in output sequences.

hidden states are often similar on either side of the conjunction, increasing the probability of repetition. Another category of local repetition is an error where words, often adjectives, repeat immediately. Interestingly, in this case the unregularized model ends the repetition with a related word after a few repeated tokens. The immediate repetition case is also where regularization has the most positive impact (see Table 3.4, k=1). The final common repetition type is repeating common phrases, which typically occurs in erroneous definitions — this kind of repetition often results in a sequence reaching a maximum length threshold before generating any content words.

3.4.2. Computational cost

The proposed regularization incurs additional computation to estimate the marginal distribution (generating text) and matching constraints in the running text from the training data. These increased training times are directly proportional to the amount of sampled text generated and the number of soft constraints. With our serial implementation, the

Word	Generated definition
Repeat after conjunctions (most common)	
develop	make a new or new or new
	make a new or more
safe	free from danger or danger
	free from danger
alleviation	the act of relieving or ... or relieving the body or ... or body
	the act of relieving something
cut	a cut of wood or wood or metal or plastic or plastic or ... or plastic
	a cut of wood or metal
Repeat immediately	
lessen	make less less less significant
	make less desirable
slim	having a thin thin thin thin thin coat
	having a slim or thin shape
brindle	a smooth brown brown brown color
	a smooth brown coat of the color of a dog
samurai	a Japanese Japanese Japanese Japanese warrior
	a Japanese warrior
Repeat common phrases	
telluric	of or relating to or characteristic of or ... or characteristic of a comet
	of or relating to the earth
papal	associated with or associated with or belonging to the papacy
	of or relating to or characteristic of a pope
fatuous	marked by or ... or marked by or characterized by or ... or characterized by
	having or showing a lack of pretensions

Figure 3.5. Examples of common cases that the regularized model reduces local repetition from definitions generated greedily. Each word shows definitions from unregularized (upper) and regularized (lower) model respectively. “. . .” indicates a phrase repeating a few times.

training time for the regularized models is approximately two to five times longer than their unregularized counterparts. However, we believe that the additional processes can be done in parallel to the model training.

3.4.3. Conflicting statistics

Most of our experiments use a reference distribution from either the training data itself, or a closely similar distribution. But, we may desire a much different distribution instead. What happens if the constraints conflict dramatically with the training data? We provide a preliminary experiment on a small, artificial set of constraints in order to reveal the effect of the regularizer in such a situation. We create imputed constraints by artificially increasing the probability of "`san francisco`" and "`mr. robot`", and decreasing the probability of "`new york`" from the PTB training data. Hence, the conditioning constraint is the modified unigram probability of {"`new`", "`san`", "`mr.`"}, and the conditional constraints include modified probability of "`york`", "`francisco`", and "`robot`" given "`new`", "`san`", and "`mr.`" respectively.

Table 3.5. Frequencies of "`new york`", "`san francisco`", and "`mr. robot`" on different texts. The reference text has manually modified frequencies of the bigrams. Text generated from the regularized model has frequency profile closer to the reference text.

Method	NY	SF	MR.R
Training	946	249	0
Reference	455	740	2,136
unregularized	988	215	0
dynamic	792	450	1,089

As we can see in Table 3.5, the regularizer can manipulate the frequencies of target bigrams to some certain extent, even though the constraints conflict significantly with the training data. Below are two excerpts of text generated by the regularized model:

- "`...grower who has gotten out of san francisco on wall street 's very heavy...`"

- "`...mr. robot noted that the underlying supply of american companies helped...`"

3.5. Related Work

A common technique for enforcing a set of pre-defined constraints on a probabilistic model is to modify the model's prediction to follow set of declarative rules. For example, Roth and Yih, 2005 [81] add integer linear programming during the inference of CRF model to incorporate constraints. Chang et al., 2008 [12] proposed Constrained Conditional Model to eliminate predicted labels that violate constraints during both training and inference. In recent neural network models, an output mask is often applied to zero-out probabilities of invalid labels [51, 103]. For instance, Paulas et al., 2017 [70] masks the probability of a word that will lead to duplicate trigrams during the text generation (with beam search). Distinct from this direction, this work focuses on *training* a model to obey soft statistical constraints, which are not applied during inference.

Our goal is to regularize an RNNLM's output distribution during training such that the global statistics of the generated text are relatively close to a given set of statistical constraints. This is a very different objective from recent regularization techniques, which are aimed at solving overfitting. For example, variations of dropout regularization randomly mask out activations or parameters of the model to be zero [87, 27, 97] and they have been successfully applied to train RNNLMs [55, 112]. Data noising techniques modify the input words directly. This can involve simply randomly dropping off input words [9], or using smoothing and back-off techniques from n-gram language modeling to compute the probability of the noise words [107].

The regularization investigated in this work can be viewed as a *label regularization* or *label smoothing* technique where the output distribution of a model is trained to match a reference distribution. This technique encourages the model to be less confident, and so less overfitted [33, 75, 90]. On the other hand, Mann and McCallum, 2007 [52] proposed expectation regularization to augment the training with unlabeled data by encouraging model predictions on the unlabeled data to match human-provided label priors. In language modeling, however, label smoothing techniques have not been widely explored. Rosenfeld, 1996 [79] applied the maximum entropy principle to train n-gram language models. Recently, Pereyra et al., 2017 [72] uses the same principle to penalize overconfident predictions of RNNLMs. They minimize KL-divergence from the uniform distribution to the model output distribution: $D_{KL}(P_\theta||u)$. Extending the previous work, we explore two different non-uniform reference distributions, and introduce a substantially more powerful context-dependent label smoothing technique. Label smoothing can be seen as a static KL regularizer, and we show how our novel dynamic KL regularization performs better than static approaches on our tasks.

Improving the overall quality of the generated text from RNNLMs has been a popular direction in recent research. A general approach is to solve the *exposure bias* by letting the models consume some of its own output predictions during training. This includes scheduled sampling [4] and beam-search optimization [105]. Many works apply the REINFORCE algorithm [104] to directly optimize a sequence-level score, which is often the final evaluation metric such as BLEU or ROUGE score [3, 70, 74, 76]. While BLEU and ROUGE score use n-gram matching similar to our proposed regularization with n-gram constraints (i.e. *bigrams*), they are locally defined per output sequence and might not capture global statistics. In generative adversarial network training, the score is the output prediction of a synchronously trained discriminator [13, 111]. On the other hand, we use count-based statistics which can be computed more efficiently. It is worth noting that our dynamic KL regularization can be used alongside these approaches.

3.6. Conclusion

We investigated how to train RNNLMs to follow a set of soft constraints from a reference distribution. We presented *dynamic KL regularization*, which encourages an RNNLM to match a reference distribution by adjusting the regularizer as training proceeds, based on sampling the model's generated text. We experimented with two types of soft constraints, one for repetition and the other for bigram distributions. Our approach is shown to lower the mismatch in repetition frequency between generated and reference text. This results in a factor of four improvement in local repetition in a definition modeling task. In addition, dynamic KL regularization can utilize the bigram distribution from a large corpus to decrease the perplexity of a language model trained on a smaller corpus.

One drawback from dynamic KL regularizer is the computational cost associated with generating output text, but this is caused by a limitation of RNNLMs. The marginal probability that a model will generate a phrase is not explicitly stated in the model's parameters, unlike n-gram language models. In the next chapter, we explore a way to efficiently estimate the marginal probability.

CHAPTER 4

Estimating Probabilities of Short Phrases Without Context

Recurrent neural networks are the standard-bearer for language modeling. However, recurrent neural network language models (RNNLMs) only estimate probabilities for complete sequences of text, whereas some applications require context-independent phrase probabilities instead. In this chapter, we study how to compute an RNN's marginal probability: the probability that the model assigns to a short sequence of text, when the preceding context is absent.

We introduce a simple method of altering the RNNLM training to occasionally *reset* to a special start state used for marginal queries, and demonstrate that this technique is remarkably effective compared to other baselines.

4.1. Introduction

Typically, RNNLMs are trained on complete sequences (e.g., a sentence or an utterance), or long sequences (e.g. several documents), and used in the same fashion during applications or testing. One of the advantages of recurrent neural networks is that they do not use a certain size of context, and the history information can cycle inside these network for long time [7, 61]. Thus, they cannot accurately estimate the probability of the sequence without a full preceding context i.e. *marginal probability* of a short sequence.

This simple, yet lacking capability is useful for wide range of applications. For example, we can detect context-independent abnormal phrases during the generation of RNNLMs.

More accurate n-gram probabilities could also aid techniques that use phrase occurrence counts for information extraction and assessment [6, 86]. In this work, we are interested in estimating how likely a phrase will be generated by an RNNLM without actually generating large amount of text (used in previous chapter). That is, we would like to characterize the model behavior so that we can make an informed change. This can be used to regularize the output text of an RNNLM as in the previous chapter or to efficiently train an RNNLM with n-gram statistics [14].

Estimating marginals from an RNN is challenging because unlike an n-gram language model, an RNNLM does not explicitly store marginal probabilities as its parameters. Instead, previous words are recurrently combined with the RNN's hidden state to produce a new state, which is used to compute a probability distribution of the next word. When the preceding context is absent, however, the starting state is also missing. In order to compute the marginal probability, in principle we must marginalize over all possible previous contexts or all continuous-vector states. Both options pose a severe computational challenge.

In this work, we study how to efficiently compute marginal probabilities from an RNNLM. We first discuss several approaches that can be used to estimate the marginal probabilities. This includes our proposed approach by randomly reseting its state during the training, and an importance sampling. Then, we present our experiments and compare results using log likelihood ratios between predicted marginal probabilities and the actual sequence frequency (both training and generated).

4.2. Marginal Estimation

The goal of the marginal estimation is to determine likelihood of a short phrase where the preceding context is absent; we refer to this likelihood as *marginal probability.* For recurrent neural network language models (RNNLM), computing marginal probabilities is challenging because an RNNLM is designed to capture an arbitrarily long dependency in the states, and assumes a single, specific initial state. Evaluating an RNN on a randomly drawn n-gram from a corpus will usually result in underestimation of the n-gram likelihood, because the model starting state is not appropriate for the n-gram. In this section, we discuss the problem in detail and present approaches investigated in this work.

4.2.1. Problem settings

Recall that, an RNNLM defines a probability distribution over words conditioned on previous words as the following [61]:

$$P(w_{1:T}) = \prod_{t=1}^{T} P(w_t|w_{1:t-1}) \tag{4.1}$$

$$P(w_t|w_{1:t-1}) = P(w_t|h_t) \propto \exp(\theta_o^{(w)} h_t) \tag{4.2}$$

$$h_t = g(h_{t-1}, w_{t-1}) \tag{4.3}$$

where $w_{1:t-1}$ is a sequence of previous words, θ_o^w denotes the output weights of a word w, and $g(\cdot)$ is a recurrent function such as an LSTM or GRU unit [18, 34]. An initial state, h_1 is needed to start the recurrent function $g(h_1, w_1)$, and also defines the probability distribution of the first word $P(w_1|h_1)$. In a normal language model setting, we compute h_1 using a special symbol (“<s>”), and a special starting state h_0 (usually set to be a vector of zeros $\vec{0}$), e.g.

$$P(\texttt{of the}) = P(\texttt{of}|h_1 = g(\vec{0}, \texttt{<s>}))P(\texttt{the}|h_2 = g(h_1, \texttt{the}))$$

The effect of this initial setting is noticeably reflected in low likelihoods of the first few tokens during the evaluation. For instance, $P(\texttt{of the})$ is likely to be underestimated because “of the” does not usually start a sentence even though it is a common phrase. This becomes problematic when an application requires likelihoods of short phrases.

For the marginal probability, we would like to compute the likelihood of standalone phrases where we do not assume the starting symbol (w_0), and we marginalize out the preceding context. For brevity, we name the RNN's first effective state as $z \in \mathbb{R}^d$, a vector of random variables representing the RNN initial state, and $w_{1:T}$ be a short sequence of text. The marginal probability is defined as:

$$p(w_{1:T}) = \int p(w_{1:T}|z)p(z)dz \tag{4.4}$$

The integral form of the marginal probability is intractable and requires an unknown density estimator of the state, $p(z)$. In this work, we explore many approaches to approximate the marginal probability.

4.2.2. Fixed-point approaches

A simple approach is to use a single point as an initial state, named z_ψ. We can either train this vector or simply set it to a zero vector. Then the marginal probability in Equation 4.4 can be estimated with a single run i.e. $p(z_\psi) = 1.0$ and $p(z) = 0.0$ if $z \neq z_\psi$.

The computation is reduced to be as follow:

$$P(w_{1:T}) = P(w_1|z_\psi)\prod_{t=2}^{T} P(w_t|h_t) \tag{4.5}$$

where $h_2 = g(z_\psi, w_1)$ and the rest of the state process as usual $h_t = g(h_{t-1}, w_{t-1})$. In this subsection, we discuss how to train an RNNLM such that its output likelihoods are more accurate for the marginal estimation.

As we previously discussed, our fixed-point state, z_ψ, is not a suitable starting state of all n-grams, so we need to train an RNNLM to adapt to this state. One way to do this is to increase the use of z_ψ as a context-independent starting state. To achieve this, we slightly modify the training algorithm of RNNs (truncated back-propagation through time) [101]. We randomly *reset* the states to z_ψ when computing a new state during the training of RNNLM (similar to Melis et al., 2017 [54]). This implies that z_ψ is trained to maximize the likelihood of different subsequent texts of different lengths, and thus is an approximately "good" starting point for any sequence. A new state is computed during the training as follows:

$$r \sim Bern(\rho) \tag{4.6}$$

$$h_t = rz_\psi + (1-r)g(h_{t-1}, w_{t-1}) \tag{4.7}$$

where ρ is a hyper-parameter for the probability of resetting a state, and $r \in \{0, 1\}$ is a scalar acting as a hard selector. Larger ρ means more training with z_ψ, but it could disrupt the long-term dependency information captured in the state. In this work, we keep ρ relatively small at 0.05 and 0.10.

In addition to the random reset, we introduce a unigram regularization to improve the accuracy of the marginal estimation. We can see from Equation 4.5 that z_ψ is used to predict the probability distribution of the first token, which should be unigram distribution (the probability of any word occurs without a context). To get this desired behavior, we propose a regularization to maximize likelihood of each token in the training data *independently*, we call this unigram regularizer:

$$\mathcal{L}_u = \mathop{\mathbb{E}}_{w \sim data}[-\text{log}p(w|z_\psi)] \tag{4.8}$$

This is added to the training objective of RNNLMs. During an evaluation, z_ψ is the initial state to predict a probability of the first token, and we do not use z_ψ afterward (no reset).

4.2.3. Trace-based approaches

Instead of using a single point, we can sample for starting states. That is, the integral form of the marginal probability in Equation 4.4 can also be approximated by sampling. In this regime, we assume that there is a source of samples which asymptotically approaches the true distribution of the RNN states as the number of samples grows. In this subsection, we discuss a sampling approach to the marginal estimation based on a collection of RNN states during an evaluation, called a *trace*.

Given a corpus of text, a trace of an RNNLM is the corresponding list of RNN states, $H^{(tr)} = (h_1^{(tr)}, h_2^{(tr)}, ..., h_M^{(tr)})$, produced when evaluating the corpus. We can estimate the marginal probability by sampling the initial state z from H as follow:

$$P(w_{1:T}) = \mathbb{E}_{z \sim H^{(tr)}}\Big[P(w_1|z)\prod_{t=2}^{T} p(w_t|h_t)\Big] \tag{4.9}$$

where $h_2 = g(z_\psi, w_1)$ and $h_t = g(h_{t-1}, w_{t-1})$ for $t > 2$ (i.e. the following states are the deterministic output of the RNN function). one way to produce a marginal estimate is to run the model forward on the query (i.e. $w_{1:T}$) several times, starting from each state in the trace and averaging the results. Given a large trace this may produce accurate estimates, but it is intractably expensive and also wasteful, since in general there are very few states in the trace that yield a high likelihood for a sequence.

To reduce the number of times we run the model on the query, we use importance sampling over the trace. We define a distribution based on a trained encoder that takes an n-gram query and output a state "near" the starting state(s) of the query, $z_\chi = q_\chi(w_{1:T})$. Specifically, we define a sampling weight for a state in the trace proportional to the dot product of the state and result of the encoder z_χ:

$$P(h^{(tr)}|w_{1:T}) = \frac{exp(z_\chi h^{(tr)})}{\sum_{h'^{(tr)} \in H^{(tr)}} exp(z_\chi h'^{(tr)})} \tag{4.10}$$

This distribution is biased to toward states that are likely to precede the query $w_{1:T}$. Then we estimate the marginal probability as the following:

$$P(w_{1:T}) = \mathop{\mathbb{E}}_{z \sim P(h^{(tr)}|w_{1:T})} \left[\frac{p(z|prior)}{p(z|w_{1:T})} P(w_1|z) \prod_{t=2}^{T} p(w_t|h_t) \right] \tag{4.11}$$

Here the choice of $p(z_1|prior)$ is a uniform distribution over the states in the trace, and $q_\chi(w_{1:T})$ is a trained RNN with its input reversed, and z_χ is the final output state of q_χ. To train q_χ, we randomly draw a substring $w_{i:i+n}$ of random length from the trace, and minimize the mean-squared difference between z_χ and $h_i^{(tr)}$.

4.3. Experiments and Results

4.3.1. Experiment Settings

We experiment with a standard LSTM language model [112] over 2 datasets: Penn Treebank (PTB) [61] for a small-size model and WikiText-2 (WT-2) [56] for a medium-size model. Our RNNLMs are 2-layer LSTMs with state sizes of 200 and 650 for small and medium respectively. The embedding size is set to be the same as the state size and tied with the output logit weight [36, 112]. For optimization, all of the models are trained with Adam [43] for 20 epochs (small) and 30 epochs (medium) with learning rate starting from 0.003 and decaying by 0.85 every epoch. In addition, we clip the cell state of the LSTM units to $[-1, 1]$ The medium-size model is trained with dropout rate of 0.5, and the small model is trained without dropout [87]. The final parameter set is then chosen as the one minimizing validation set loss.

The model and training of the query model ($q_\chi(w_{1:T})$) used in the importance sampling approach are similar to the RNNLMs. Since each state LSTM cell contains a cell state, c and an output state, h, our full state consists of four vectors (not independent). In our experiment, we predict the query state, z_χ, to match the output state of the last layer, rather than the full state.

4.3.2. Marginal Estimation

To evaluate the approaches, we compare our methods' marginal estimates with the marginal probabilities from the actual model generation. For each model, we generate around 1 million and 2.1 million tokens for PTB and WT-2 models respectively, and exclude

n-grams with frequency less than 10 from our evaluation to reduce noise from the generation. We then use the marginal probability computed from the generation as the target, and measure the performance using the absolute value of the log ratio (lower is better):

$$\text{error}(w_{1:T}) = \left|\log P_{count}(w_{1:T}) - \log P_{est}(w_{1:T})\right| \tag{4.12}$$

This evaluation measure gives equal importance to every n-gram regardless of its frequency. To show how performance varies depending on the query, we present results aggregated in two different ways, by n-gram length and frequency bucket.

Table 4.1 shows the error of the different marginal probability estimation approaches. Our baselines includes *KN-5*: a Kneser-Ney 5-gram language model [45], *Zero*: an RNNLM using the zero vector as a starting state, and *Rand*: a trace-based approach using randomly selected trace states as a staring state. The 5-gram language models are trained using the training sets. *Reset* is similar to *Zero* but the model is trained with resetting and unigram regularization as described in Section 4.2.2. The reset rates are 0.1 and 0.05 for PTB and WT-2 model respectively. The results with trainable starting state are similar to using a fixed zero starting state, and are omitted. Finally, *IW* indicates the importance sampling approach described in Section 4.2.3. We do not observe a significant difference when using the trace from the generated text and the training text. Thus, we only experiment with the trace states using the corresponding training corpus (lower computational overhead).

The results show that despite its simplicity and efficiency, *Reset* predicts marginal probabilities closest to the count-based marginal probabilities from the generated text.

Table 4.1. Average absolute log ratio between the marginal probabilities from generation and models' estimates (*KN-5, Zero, Rand, Reset, IW*). The "Total" column shows the approximate number of n-grams being averaged. *Reset* has the lowest error across almost all groups.

PTB (small-size model)						
Length	*KN-5*	*Zero*	*Rand*	*Reset*	*IW*	Total
1	0.651	1.304	0.514	**0.280**	0.444	6.6k
2	1.001	4.641	0.755	**0.400**	0.538	10.7k
3	1.837	6.654	0.940	**0.407**	0.611	4k
4	2.647	7.810	1.486	**0.414**	0.733	1.1k
5	3.214	8.682	2.315	**0.455**	0.989	0.3k
Freq.	*KN-5*	*Zero*	*Rand*	*Reset*	*IW*	Total
10 - 20	1.222	4.562	0.820	**0.462**	0.574	12k
20 - 50	1.147	4.184	0.769	**0.312**	0.525	6.7k
50 - 100	1.043	3.484	0.721	**0.202**	0.497	2.2k
10 - 500	1.019	3.234	0.664	**0.185**	0.451	1.7k
500 - inf	0.889	4.192	0.474	**0.124**	0.385	0.3k
WT-2 (medium-size model)						
Length	*KN-5*	*Zero*	*Rand*	*Reset*	*IW*	Total
1	0.835	1.053	1.072	**0.314**	0.781	9.4k
2	1.527	3.847	1.353	**0.519**	0.831	23.6k
3	2.502	5.373	1.748	**0.631**	0.837	14.5k
4	3.355	6.728	2.457	**0.792**	0.832	4.6k
5	4.428	7.585	3.414	0.952	**0.913**	1.2k
Freq.	*KN-5*	*Zero*	*Rand*	*Reset*	*IW*	Total
10 - 20	2.017	4.373	1.640	**0.598**	0.866	27.9k
20 - 50	1.831	3.985	1.537	**0.505**	0.814	15.6k
50 - 100	1.707	3.573	1.423	**0.410**	0.746	5.1k
10 - 500	1.563	3.357	1.253	**0.340**	0.723	3.9k
500 - inf	1.462	3.833	0.989	**0.244**	0.634	0.7k

The errors are significantly lower than the baselines (*KN-5*, *Zero*, and *Rand*) and importance sampling *IW*. The marginal probabilities of *Rand* and *IW* are average of 100 sampled states (the errors presented here are average of 30 trials). This means *IW* requires significantly more computation than the other approaches during the inference, but the errors are higher than *Reset*.

4.3.3. Perplexity and *Reset*

While *IW* is not as accurate and efficient, it might be suitable when we do not want to train another RNNLM with the state reseting and unigram regularization (*Reset*). We present the impact on perplexity of the state reset and unigram regularization. Table 4.2 shows the test perplexity of each model. We can see that the reset and unigram regularization (*Reset*) tends to slightly worsen the test perplexity comparing to the standard training. Since WT-2 dataset is split by articles, the distribution of words can be different between training, validating, and testing sets. We think that the unigram regularization is likely the cause of this minor discrepancy.

Table 4.2. Perplexity evaluated on the test sets. Overall, the reset and unigram regularization (*Reset*) has a negative impact on the perplexity.

	PTB		WT-2	
Reset+Unigram	No	Yes	No	Yes
Small	117.3	112.6	135.6	141.2
Medium	92.7	94.9	105.9	108.2

4.3.4. Lowering number of samples

The computational cost of the importance sampling method linearly grows with the number of samples. We experiment with lower number of samples and present the errors and variances. Table 4.3 shows the error of the marginal probabilities estimated using 100, 20, and 1 samples and compare with the random sampling (100 samples). We run the experiment 30 times and present means and variances of the errors. The variances are computed from the 30 trials of each n-gram and, in the table, are the mean variance of n-grams in each group. As expected, higher number of samples leads to more accurate marginal estimation and lower variance. Interestingly the importance sampling with 20 samples outperforms the random sampling with 100 samples.

Table 4.3. Marginal probability errors and variances (in parentheses) of the trace-based approaches. The results are obtained from 30 trials of WT-2 experiments.

Length	*Rand-100*	*IW-100*	*IW-20*	*IW-1*
1	1.072 (0.398)	0.781 (0.144)	1.000 (0.343)	2.634 (3.905)
2	1.353 (0.450)	0.831 (0.155)	0.955 (0.346)	2.480 (3.147)
3	1.748 (0.551)	0.837 (0.164)	0.955 (0.361)	2.635 (3.422)
4	2.457 (0.843)	0.832 (0.186)	0.948 (0.412)	2.866 (4.204)
5	3.414 (1.256)	0.913 (0.226)	1.018 (0.552)	3.087 (5.742)
Freq.	*Rand-100*	*IW-100*	*IW-20*	*IW-1*
10 - 20	1.640 (0.541)	0.866 (0.172)	1.002 (0.385)	2.698 (3.702)
20 - 50	1.537 (0.522)	0.814 (0.158)	0.952 (0.358)	2.572 (3.476)
50 - 100	1.423 (0.498)	0.746 (0.136)	0.888 (0.306)	2.422 (3.160)
10 - 500	1.253 (0.414)	0.723 (0.117)	0.874 (0.280)	2.302 (2.879)
500 - inf	0.989 (0.385)	0.634 (0.085)	0.786 (0.239)	2.002 (2.265)

4.4. Conclusion

In this chapter, we evaluate approaches to compute n-gram marginal probability from a RNNLM, including n-gram language modeling, using fixed-point starting state, and importance sampling. Our target marginal probabilities are from the generated text from the RNNLM. We find that the fixed-point approach predicts the marginal probabilities closest to the target when the RNNLM is trained with state reseting and unigram regularization. The importance sampling comes in second, and is suitable when we do not want to train another RNNLM.

For future work, we would like to extend this work in two directions. First, we would like to evaluate our approaches in applications. This includes directly using the marginal probabilities (i.e. during generation), and using them to train or regularize a RNNLM.

Another direction is that we would like to continue improving the marginal estimation: experimenting with other recent density estimation models such as autoregressive models [94] or normalizing flows [77].

CHAPTER 5

Conclusion and Future Work

In this dissertation, we introduced methods that improve usability of neural language models. Our work centered around the strength of current deep learning models, the information is learned in distributed fashion, which gives rise to new challenges that we addressed. We discussed and proposed novel solutions to reveal the information captured by the model representations of words, and to inspect and direct overall generative behavior of the model. In this chapter, we conclude this dissertation, and discuss remaining challenges and future work to address them.

Chapter 2. tackled the problem of interpretability of word embeddings by harnessing the generation quality of recurrent neural network language models (RNNLMs) and definitions of words from dictionaries. We introduced a new task of estimating the probability of a textual definition given a word being defined and its embedding, definition modeling. A definition model is a type of neural language model and can be used to match words and definitions and, more interestingly, generate a definition given an embedding.

We showed that a definition model that can (learn to) control the influence of word being defined over words in a definition performs best. In addition, we found that adding character-level information to word embeddings further improves the performance. Finally, based on our analysis, we think that the popular word embeddings (Google news

Word2Vec[1]) tend to over simplify the semantics of words and bias toward context words. For example, a generated definition of "`adorable`" is "`having the qualities of a child`" where "`child`" is often described as "`adorable`" (Figure 2.9).

The results highlighted a few limitations of current neural language models. First, words can have multiple meanings, whether we decide to have multiple embeddings for a word or use context-specific embedding, a neural language model should handle polysemy properly, especially when learning the representations. Second, RNNLMs tend to repeat phrases or generate abnormal phrases. In this situation, researchers including us resort to manually craft an ad-hoc process to prevent or clean such mistakes.

Chapter 3. investigated the latter limitation above: training an RNNLM to behave as we desire. We introduced a method that encourage an RNNLM to generate text that follows a pre-defined constraints during the training. We use n-gram distribution as a way to specify the constraints. While n-gram distributions are not easily consumed by human, they are still interpretable and can be efficiently computed from a corpus of text. However, they are not easily queries from an RNNLM due to its distributed parameters.

We presented *dynamic KL regularization* to overcome this challenge. The regularization dynamically estimates n-gram distributions from the current output generated text of the model, thus it can accurately adjust the model behavior. We experimented and showed that an RNNLM trained with the proposed method behaviors more similar to our bigram distribution (improving generalizability) and repetition distribution (reducing repeated phrases). The interesting finding worth highlighting is that the perplexities of a

[1]https://code.google.com/archive/p/word2vec

model trained with and without dynamic KL regularization are similar, while they behave drastically different.

While we explore word-level soft constraints in this work, dynamic KL regularization can be extended to incorporate higher-level constraints such as syntactic or semantic information. This could aid human in term of creating the statistical constraints to train the models. We believe that the future direction should focus on encouraging RNNLMs to generate text with a particular sentiment, writing style, reading level, and so on. However, this also means computing a more complicated statistics of the model behavior. In particular, we have to generate large amount of text to have sufficient statistics of n-grams in our experiments (up to 4-gram).

Chapter 4. directly addressed, but not limited to, the drawback of the dynamic KL regularization. In general, we explore methods to compute how likely an RNNLM will generate a phrase regardless of context i.e. the probability of the context-independent phrase from an RNNLM. The challenge here is that RNNLMs are trained to estimate probabilities for complete sequences, and their parameters do not explicitly state their behavior.

We presented empirical results of several methods for marginal estimation: predicting probability of short phrases without preceding context, and introduced a modification to the training of an RNNLM such that its marginal estimates are more accurate. We showed that our proposed method can efficiently predict the marginal probabilities closest to the statistics from the model generated text. The proposed method requires training a new RNNLM. If this is not desired, we also introduced a method based on importance sampling which outperforms the baseline approaches.

While we found successful methods that are several times more accurate than the baselines, we have not yet tested its usefulness in an application setting. Applying it to dynamic KL regularization is a our immediate future work direction. On another direction, we would like to further improve the accuracy of the marginal estimation. This could involve bringing a structure to the hidden states of RNNLMs, so that we have a better understanding of its properties.

I hope this dissertation can encourage researchers to tackle the discussed limitations and improve upon my work. I believe it is necessary to make progress toward bringing both machine learning models and humans to a common ground.

References

[1] L. Argerich, J. Torré Zaffaroni, and M. J Cano. “Hash2Vec, Feature Hashing for Word Embeddings”. In: *ArXiv e-prints* (Aug. 2016). arXiv: `1608.08940 [cs.CL]`.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *Proceedings of 3rd International Conference on Learning Representations*. 2014.

[3] D. Bahdanau et al. “An Actor-Critic Algorithm for Sequence Prediction”. In: *ArXiv e-prints* arXiv:1607.07086v3 [cs.LG] (July 2016). arXiv: `1607.07086 [cs.LG]`.

[4] Samy Bengio et al. “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks”. In: *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 1171–1179. URL: `http://papers.nips.cc/paper/5956-scheduled-sampling-for-sequence-prediction-with-recurrent-neural-networks.pdf`.

[5] Yoshua Bengio et al. “A Neural Probabilistic Language Model”. In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 1137–1155. ISSN: 1532-4435. URL: `http://dl.acm.org/citation.cfm?id=944919.944966` (visited on 11/24/2014).

[6] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. “TextJoiner: On-demand Information Extraction with Multi-Pattern Queries”. In: *2014 Workshop on Automated Knowledge Base Construction*. 2014.

[7] Mikael Boden. “A guide to recurrent neural networks and backpropagation”. In: ().

[8] Tom Bosc and Pascal Vincent. “Learning Word Embeddings from Dictionary Definitions Only”. In: *Workshop on Meta-Learning (MetaLearn 2017)*. 2017. URL: `http://metalearning.ml/papers/metalearn17_bosc.pdf`.

[9] Samuel R. Bowman et al. “Generating Sentences from a Continuous Space”. In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language*

Learning. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 10–21. URL: http://www.aclweb.org/anthology/K16-1002.

[10] Thorsten Brants and Alex Franz. "Web 1T 5-gram, ver. 1". In: *LDC2006T13* (2006).

[11] Peter F. Brown et al. "A Statistical Approach to Machine Translation". In: *Comput. Linguist.* 16.2 (June 1990), pp. 79–85. ISSN: 0891-2017. URL: http://dl.acm.org/citation.cfm?id=92858.92860 (visited on 04/03/2018).

[12] Ming-Wei Chang et al. "Learning and Inference with Constraints". In: *Proceedings of the 23rd National Conference on Artificial Intelligence*. AAAI'08. Chicago, Illinois: AAAI Press, 2008, pp. 1513–1518. ISBN: 978-1-57735-368-3. URL: http://dl.acm.org/citation.cfm?id=1620270.1620322.

[13] T. Che et al. "Maximum-Likelihood Augmented Discrete Generative Adversarial Networks". In: *ArXiv e-prints* arXiv:1702.07983v1 [cs.AI] (Feb. 2017). arXiv: 1702.07983 [cs.AI].

[14] Ciprian Chelba, Mohammad Norouzi, and Samy Bengio. "N-gram Language Modeling using Recurrent Neural Network Estimation". In: *arXiv:1703.10724 [cs]* (Mar. 2017). arXiv: 1703.10724. URL: http://arxiv.org/abs/1703.10724.

[15] Boxing Chen and Colin Cherry. "A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU". In: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, June 2014, pp. 362–367. URL: http://www.aclweb.org/anthology/W/W14/W14-3346.

[16] Stanley F. Chen and Joshua Goodman. "An Empirical Study of Smoothing Techniques for Language Modeling". In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Santa Cruz, California, USA: Association for Computational Linguistics, June 1996, pp. 310–318. DOI: 10.3115/981863.981904. URL: http://www.aclweb.org/anthology/P96-1041.

[17] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *EMNLP 2014*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. URL: http://www.aclweb.org/anthology/D14-1179.

[18] Kyunghyun Cho et al. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches". In: *Proceedings of SSST-8, Eighth Workshop on Syntax,*

Semantics and Structure in Statistical Translation. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 103–111. URL: `http://www.aclweb.org/anthology/W14-4012`.

[19] Martin S Chodorow, Roy J Byrd, and George E Heidorn. “Extracting semantic hierarchies from a large on-line dictionary”. In: *ACL 1985*. Association for Computational Linguistics. 1985, pp. 299–304.

[20] Jan Chorowski et al. “Attention-Based Models for Speech Recognition”. In: *arXiv: 1506.07503 [cs, stat]* (June 2015). arXiv: 1506.07503. URL: `http://arxiv.org/abs/1506.07503` (visited on 04/07/2016).

[21] Ronan Collobert et al. “Natural Language Processing (Almost) from Scratch”. In: *J. Mach. Learn. Res.* 12 (Nov. 2011), pp. 2493–2537. ISSN: 1532-4435. URL: `http://dl.acm.org/citation.cfm?id=1953048.2078186` (visited on 11/24/2014).

[22] George E Dahl et al. “Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition”. In: *IEEE Transactions on audio, speech, and language processing* 20.1 (2012), pp. 30–42.

[23] Georgiana Dinu and Marco Baroni. “How to make words with vectors: Phrase generation in distributional semantics”. In: *ACL 2014*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 624–633. URL: `http://www.aclweb.org/anthology/P14-1059`.

[24] William Dolan, Lucy Vanderwende, and Stephen D Richardson. “Automatically deriving structured knowledge bases from on-line dictionaries”. In: *PACLING 1993*. 1993, pp. 5–14.

[25] Doug Downey, Stefan Schoenmackers, and Oren Etzioni. “Sparse information extraction: Unsupervised language models to the rescue”. In: *ACL 2007*. Vol. 45. 1. 2007, p. 696.

[26] Jeffrey L. Elman. “Finding structure in time”. In: *Cognitive Science* 14.2 (1990), pp. 179–211.

[27] Yarin Gal and Zoubin Ghahramani. “A Theoretically Grounded Application of Dropout in Recurrent Neural Networks”. In: *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016, pp. 1019–1027. URL: `http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks.pdf`.

[28] Andrew R. Golding and Dan Roth. “A Winnow-Based Approach to Context-Sensitive Spelling Correction”. In: *Machine Learning* 34.1 (Feb. 1999), pp. 107–130. ISSN: 1573-0565. DOI: `10.1023/A:1007545901558`. URL: `https://doi.org/10.1023/A:1007545901558`.

[29] Joshua Goodman. “A Bit of Progress in Language Modeling”. In: *arXiv:cs/0108005* (Aug. 2001). arXiv: cs/0108005. URL: `http://arxiv.org/abs/cs/0108005` (visited on 03/17/2018).

[30] Jonathan Gordon and Benjamin Van Durme. “Reporting bias and knowledge acquisition”. In: *AKBC workshop, 2013*. 2013.

[31] Marti A. Hearst. “Automatic Acquisition of Hyponyms from Large Text Corpora”. In: *COLING 1992*. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 539–545. DOI: `10.3115/992133.992154`. URL: `http://dx.doi.org/10.3115/992133.992154` (visited on 06/01/2016).

[32] Felix Hill et al. “Learning to Understand Phrases by Embedding the Dictionary”. In: *TACL 2016* 4 (2016), pp. 17–30. ISSN: 2307-387X. URL: `https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/711`.

[33] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. In: *Deep Learning and Representation Learning Workshop: NIPS 2014*. 2014.

[34] Sepp Hochreiter and Jurgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`.

[35] Fei Huang et al. “Learning representations for weakly supervised natural language processing tasks”. In: *Computational Linguistics* 40.1 (2014), pp. 85–120.

[36] Hakan Inan, Khashayar Khosravi, and Richard Socher. “Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling”. In: *arXiv:1611.01462 [cs, stat]* (Nov. 2016). arXiv: 1611.01462. URL: `http://arxiv.org/abs/1611.01462` (visited on 04/10/2017).

[37] R. Jozefowicz et al. “Exploring the Limits of Language Modeling”. In: *ArXiv e-prints* arXiv:1602.02410v2 [cs.CL] (Feb. 2016). arXiv: `1602.02410 [cs.CL]`.

[38] Nal Kalchbrenner and Phil Blunsom. "Recurrent Continuous Translation Models". In: *EMNLP 2013*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1700–1709. URL: `http://www.aclweb.org/anthology/D13-1176`.

[39] Andrej Karpathy and Li Fei-Fei. "Deep Visual-Semantic Alignments for Generating Image Descriptions". In: *arXiv:1412.2306 [cs]* (Dec. 2014). arXiv: 1412.2306. URL: `http://arxiv.org/abs/1412.2306` (visited on 10/23/2015).

[40] Andrej Karpathy and Li Fei-Fei. "Deep Visual-Semantic Alignments for Generating Image Descriptions". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2015), pp. 664–676.

[41] S. Katz. "Estimation of probabilities from sparse data for the language model component of a speech recognizer". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.3 (Mar. 1987), pp. 400–401. ISSN: 0096-3518. DOI: `10.1109/TASSP.1987.1165125`.

[42] Yoon Kim et al. "Character-Aware Neural Language Models". In: *AAAI 2016*. 2016. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12489`.

[43] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR 2014* abs/1412.6980 (2014). URL: `http://arxiv.org/abs/1412.6980`.

[44] Judith Klavans and Brian Whitman. "Extracting taxonomic relationships from on-line definitional sources using LEXING". In: *ACM/IEEE-CS 2001*. ACM. 2001, pp. 257–258.

[45] R. Kneser and H. Ney. "Improved backing-off for M-gram language modeling". In: *1995 International Conference on Acoustics, Speech, and Signal Processing*. Vol. 1. May 1995, 181–184 vol.1. DOI: `10.1109/ICASSP.1995.479394`.

[46] Thomas K Landauer and Susan T Dumais. "A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge." In: *Psychological review* 104.2 (1997), p. 211.

[47] Yann LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *NIPS 1990*. Ed. by D. S. Touretzky. Morgan-Kaufmann, 1990, pp. 396–404.

[48] Jiwei Li, Minh-Thang Luong, and Daniel Jurafsky. "A Hierarchical Neural Autoencoder for Paragraphs and Documents". In: *ACL 2015*. 2015.

[49] Jiwei Li et al. “A Persona-Based Neural Conversation Model”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 994–1003. URL: `http://www.aclweb.org/anthology/P16-1094`.

[50] Jiwei Li et al. “Adversarial Learning for Neural Dialogue Generation”. In: *arXiv: 1701.06547 [cs]* (Jan. 2017). arXiv: 1701.06547. URL: `http://arxiv.org/abs/1701.06547` (visited on 01/24/2017).

[51] Chen Liang et al. “Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 23–33. URL: `http://aclweb.org/anthology/P17-1003`.

[52] G. Mann and A. McCallum. “Simple, robust, scalable semi-supervised learning via expectation regularization”. In: *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*. Ed. by Zoubin Ghahramani. Corvallis, OR: Omnipress, 2007, pp. 593–600.

[53] Judith Markowitz, Thomas Ahlswede, and Martha Evens. “Semantically significant patterns in dictionary definitions”. In: *ACL 1986*. Association for Computational Linguistics. 1986, pp. 112–119.

[54] G. Melis, C. Dyer, and P. Blunsom. “On the State of the Art of Evaluation in Neural Language Models”. In: *ArXiv e-prints* arXiv:1707.05589v2 [cs.CL] (July 2017). arXiv: `1707.05589 [cs.CL]`.

[55] S. Merity, N. Shirish Keskar, and R. Socher. “Regularizing and Optimizing LSTM Language Models”. In: *ArXiv e-prints* arXiv:1708.02182v1 [cs.CL] (Aug. 2017). arXiv: `1708.02182 [cs.CL]`.

[56] S. Merity et al. “Pointer Sentinel Mixture Models”. In: *ArXiv e-prints* arXiv: 1609.07843v1 [cs.CL] (Sept. 2016). arXiv: `1609.07843 [cs.CL]`.

[57] T. Mikolov and G. Zweig. “Context dependent recurrent neural network language model”. In: *SLT 2012*. 2012 IEEE Spoken Language Technology Workshop (SLT). 2012, pp. 234–239. DOI: `10.1109/SLT.2012.6424228`.

[58] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic Regularities in Continuous Space Word Representations.” In: *HLT-NAACL 2013*. The Association for Computational Linguistics, Aug. 16, 2013, pp. 746–751. URL: `http://dblp.`

uni-trier.de/db/conf/naacl/naacl2013.html#MikolovYZ13 (visited on 01/19/2014).

[59] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[60] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv:1301.3781 [cs]* (Jan. 16, 2013). URL: http://arxiv.org/abs/1301.3781 (visited on 04/04/2014).

[61] Tomáš Mikolov et al. "Recurrent Neural Network Based Language Model". In: *INTERSPEECH 2010*. INTERSPEECH-2010. Makuhari, Chiba, Japan, Sept. 26, 2010, pp. 1045–1048.

[62] George A. Miller. "WordNet: A Lexical Database for English". In: *Magazine Communications of the Association for Computing Machinery* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: http://doi.acm.org/10.1145/219717.219748 (visited on 06/01/2016).

[63] A. Mnih and G. Hinton. "Three new graphical models for statistical language modelling". In: *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*. Ed. by Zoubin Ghahramani. 2007, pp. 641–648.

[64] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *arXiv:1312.5602 [cs]* (Dec. 2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602 (visited on 11/29/2016).

[65] Simonetta Montemagni and Lucy Vanderwende. "Structural patterns vs. string patterns for extracting semantic information from dictionaries". In: *COLING 1992*. Association for Computational Linguistics. 1992, pp. 546–552.

[66] Ke Ni and William Yang Wang. "Learning to Explain Non-Standard English Words and Phrases". In: *arXiv:1709.09254 [cs]* (Sept. 2017). arXiv: 1709.09254. URL: http://arxiv.org/abs/1709.09254.

[67] Thanapon Noraset, David Demeter, and Doug Downey. "Controlling Global Statistics in Recurrent Neural Network Text Generation". In: *The Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[68] Thanapon Noraset et al. “Definition Modeling: Learning to Define Word Embeddings in Natural Language”. In: *The Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. 2017. URL: `https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14827`.

[69] Kishore Papineni et al. “Bleu: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: `10.3115/1073083.1073135`. URL: `http://www.aclweb.org/anthology/P02-1040`.

[70] R. Paulus, C. Xiong, and R. Socher. “A Deep Reinforced Model for Abstractive Summarization”. In: *ArXiv e-prints* arXiv:1705.04304v3 [cs.CL] (May 2017). arXiv: `1705.04304 [cs.CL]`.

[71] Jeffrey Pennington, Richard Socher, and Christopher Manning. “Glove: Global Vectors for Word Representation”. In: *EMNLP 2014*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. URL: `http://www.aclweb.org/anthology/D14-1162`.

[72] Gabriel Pereyra et al. “Regularizing Neural Networks by Penalizing Confident Output Distributions”. In: *Proceedings of 5th International Conference on Learning Representations*. 2017.

[73] Lawrence R Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Vol. 14. PTR Prentice Hall Englewood Cliffs, 1993.

[74] M. Ranzato et al. “Sequence Level Training with Recurrent Neural Networks”. In: *ArXiv e-prints* arXiv:1511.06732v7 [cs.LG] (Nov. 2015). arXiv: `1511.06732 [cs.LG]`.

[75] S. Reed et al. “Training Deep Neural Networks on Noisy Labels with Bootstrapping”. In: *ArXiv e-prints* arXiv:1412.6596v3 [cs.CV] (Dec. 2014). arXiv: `1412.6596 [cs.CV]`.

[76] Steven J. Rennie et al. “Self-Critical Sequence Training for Image Captioning”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[77] Danilo Jimenez Rezende and Shakir Mohamed. “Variational Inference with Normalizing Flows”. In: *arXiv:1505.05770 [cs, stat]* (May 2015). arXiv: 1505.05770. URL: `http://arxiv.org/abs/1505.05770` (visited on 01/31/2018).

[78] Alan Ritter, Colin Cherry, and William B. Dolan. “Data-Driven Response Generation in Social Media”. In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, July 2011, pp. 583–593. URL: `http://www.aclweb.org/anthology/D11-1054`.

[79] Ronald Rosenfeld. “A maximum entropy approach to adaptive statistical language modelling”. In: *Computer Speech & Language* 10.3 (July 1, 1996), pp. 187–228. ISSN: 0885-2308. DOI: `10.1006/csla.1996.0011`. URL: `http://www.sciencedirect.com/science/article/pii/S088523089690011X`.

[80] Ronald Rosenfeld. “Two decades of statistical language modeling: Where do we go from here?” In: *Proceedings of the IEEE* 88.8 (2000), pp. 1270–1278.

[81] Dan Roth and Wen-tau Yih. “Integer Linear Programming Inference for Conditional Random Fields”. In: *Proceedings of the 22Nd International Conference on Machine Learning*. ICML ’05. Bonn, Germany: ACM, 2005, pp. 736–743. ISBN: 1-59593-180-5. DOI: `10.1145/1102351.1102444`. URL: `http://doi.acm.org/10.1145/1102351.1102444`.

[82] Alexander M. Rush, Sumit Chopra, and Jason Weston. “A Neural Attention Model for Abstractive Sentence Summarization”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 379–389. URL: `http://aclweb.org/anthology/D15-1044`.

[83] Abigail See, Peter J. Liu, and Christopher D. Manning. “Get To The Point: Summarization with Pointer-Generator Networks”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 1073–1083. URL: `http://aclweb.org/anthology/P17-1099`.

[84] Julian Seitner et al. “A Large Database of Hypernymy Relations Extracted from the Web”. In: *LREC 2016*. 2016.

[85] Iulian Vlad Serban et al. “Generative Deep Neural Networks for Dialogue: A Short Review”. In: *arXiv:1611.06216 [cs]* (Nov. 2016). arXiv: 1611.06216. URL: `http://arxiv.org/abs/1611.06216` (visited on 11/21/2016).

[86] Stephen Soderland et al. “The use of Web-based statistics to validate information extraction”. In: *AAAI-04 Workshop on Adaptive Text Extraction and Mining*. 2004, pp. 21–26.

[87] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: `http://jmlr.org/papers/v15/srivastava14a.html` (visited on 09/03/2017).

[88] Ilya Sutskever, James Martens, and Geoffrey E Hinton. “Generating text with recurrent neural networks”. In: *ICML 2011*. 2011, pp. 1017–1024.

[89] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to Sequence Learning with Neural Networks”. In: *NIPS 2014*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3104–3112. URL: `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

[90] C. Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *ArXiv e-prints* arXiv:1512.00567v3 [cs.CV] (Dec. 2015). arXiv: `1512.00567 [cs.CV]`.

[91] Julien Tissier, Christopher Gravier, and Amaury Habrard. “Dict2vec : Learning Word Embeddings using Lexical Dictionaries”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 254–263. URL: `https://www.aclweb.org/anthology/D17-1024`.

[92] Zhaopeng Tu et al. “Modeling Coverage for Neural Machine Translation”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 76–85. URL: `http://www.aclweb.org/anthology/P16-1008`.

[93] Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. “Word Representations: A Simple and General Method for Semi-Supervised Learning”. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, 2010, pp. 384–394. URL: `http://www.aclweb.org/anthology/P10-1040`.

[94] Benigno Uria et al. “Neural Autoregressive Distribution Estimation”. In: *arXiv: 1605.02226 [cs]* (May 2016). arXiv: 1605.02226. URL: `http://arxiv.org/abs/1605.02226` (visited on 03/09/2018).

[95] Lucy Vanderwende et al. “MindNet: an automatically-created lexical resource”. In: *HLT-EMNLP 2005*. Oct. 2005. URL: `http://research.microsoft.com/apps/pubs/default.aspx?id=75982`.

[96] Ashish Vaswani et al. “Attention Is All You Need”. In: *arXiv:1706.03762 [cs]* (June 2017). arXiv: 1706.03762. URL: `http://arxiv.org/abs/1706.03762`.

[97] Li Wan et al. “Regularization of Neural Networks using DropConnect”. In: *Proceedings of Machine Learning Research*. International Conference on Machine Learning. Feb. 13, 2013, pp. 1058–1066. URL: `http://proceedings.mlr.press/v28/wan13.html` (visited on 09/03/2017).

[98] Tong Wang, Abdelrahman Mohamed, and Graeme Hirst. “Learning Lexical Embeddings with Syntactic and Lexicographic Knowledge”. In: *ACL 2015*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 458–463. URL: `http://www.aclweb.org/anthology/P15-2075`.

[99] Tsung-Hsien Wen et al. “Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems”. In: *EMNLP 2015*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1711–1721. URL: `http://aclweb.org/anthology/D15-1199`.

[100] Tsung-Hsien Wen et al. “Stochastic Language Generation in Dialogue using Recurrent Neural Networks with Convolutional Sentence Reranking”. In: *SIGDIAL 2015*. Prague, Czech Republic: Association for Computational Linguistics, Sept. 2015, pp. 275–284. URL: `http://aclweb.org/anthology/W15-4639`.

[101] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[102] Amber Wilcox-O’Hearn, Graeme Hirst, and Alexander Budanitsky. “Real-Word Spelling Correction with Trigrams: A Reconsideration of the Mays, Damerau, and Mercer Model”. In: *Computational Linguistics and Intelligent Text Processing*. Ed. by Alexander Gelbukh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 605–616. ISBN: 978-3-540-78135-6.

[103] Jason D Williams, Kavosh Asadi, and Geoffrey Zweig. “Hybrid Code Networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 665–677. URL: `http://aclweb.org/anthology/P17-1062`.

[104] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[105] Sam Wiseman and Alexander M. Rush. “Sequence-to-Sequence Learning as Beam-Search Optimization”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 1296–1306. URL: `https://aclweb.org/anthology/D16-1137`.

[106] Yonghui Wu et al. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”. In: *arXiv:1609.08144 [cs]* (Sept. 2016). arXiv: 1609.08144. URL: `http://arxiv.org/abs/1609.08144` (visited on 04/03/2018).

[107] Ziang Xie et al. “Data Noising as Smoothing in Neural Network Language Models”. In: *Proceedings of 5th International Conference on Learning Representations*. Mar. 7, 2017.

[108] Ziang Xie et al. “Neural language correction with character-based attention”. In: *arXiv preprint arXiv:1603.09727* (2016).

[109] Kelvin Xu et al. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *ICML*. 2015. URL: `http://arxiv.org/abs/1502.03044`.

[110] Dani Yogatama et al. “Learning Word Representations with Hierarchical Sparse Coding”. In: *Proceedings of The 32nd International Conference on Machine Learning*. Vol. 37. ICML ’15. Lille-Euralille, France: Journal of Machine Learning Research, June 1, 2015. URL: `http://jmlr.org/proceedings/papers/v37/yogatama15.pdf`.

[111] Lantao Yu et al. “SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient”. In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. AAAI Press, 2017.

[112] W. Zaremba, I. Sutskever, and O. Vinyals. “Recurrent Neural Network Regularization”. In: *ArXiv e-prints* arXiv:1409.2329v5 [cs.NE] (Sept. 2014). arXiv: `1409.2329`.