Context Generative Recurrent Neural Network Language Model

Sai Ganesh Bandiatmakuri SBANDIAT@ANDREW.CMU.EDU
Robotics Institute, Carnegie Mellon University

Yu-Fang Chang YUFANGC@ANDREW.CMU.EDU
Robotics Institute, Carnegie Mellon University

Pankesh Bamotra PBAMOTRA@ANDREW.CMU.EDU
Language Technologies Institute, Carnegie Mellon University

Abstract

Our project introduces Context Generating RNNLMs (CGRNNLMs), a new graphical model for language modeling based on RNNs. It combines years of insights developed from deep learning applications in computer vision, with modern efforts to make dynamic topic modelling training and inference simpler. In our model, we adopt a joint learning scheme to simultaneously predict the future word and contexts, while learning combined representation of the word and topic space, simultaneously capturing both local and global semantic information. We demonstrate the importance of various tunable parameters in our model and compare performance with various state of the art methods.

1. Overview of the project

Topic modeling provides rich information about words and documents. Typical methods to learn topic models use expensive methods like MCMC sampling or variational inference. Recently, people employ large datasets to learn topic models; however, it is expensive and difficult to implement or maintain. State of the art papers have elaborate schemes to store and distribute computations for different data structures, such as word-topic association tables, over different machines (Yuan et al., 2015). Additionally, when it comes to dynamic topic modeling, which takes the order of the documents into consideration, it may be more difficult. We attempt to address this problem by combining dynamic topic modeling with Recurrent Neural Networks.

Recurrent Neural Network Language Models (RNNLMs) have shown state of the art results for various language tasks (Mikolov & Zweig, 2012). Language modeling is a classical problem where the objective is to predict the next word, given the current (or previous) word(s). Major contributions from (Mikolov & Zweig, 2012) are two fold - Firstly, they improve on classical RNNLMs by augmenting the RNN architecture to take an additional context vector f(t) at each time step as input and use it for prediction. Secondly, they provide a fast alternative to exact LDA to provide the context vectors. The motivation behind the approximation is to avoid the prohibitive computational cost in running LDA inference for each word in a sentence by updating the context. In this project, we propose Context Generating RNNLMs (CGRNNLMs) to improve upon (Mikolov & Zweig, 2012) by simultaneously predicting the next word along with the context to be used as input for subsequent prediction. Here, we hope to learn a more meaningful representation for the context f(t) by combining information from previous words(and contexts) in a nonlinear manner, giving importance to their semantics and sequence, rather than a simple product of LDA topic representations of individual words (and thus, losing the sequence information). The loss from predicting words serves as a regularization to predict the next context (and vice versa).

To the best of our knowledge, ours is a unique attempt to simultaneously model the dynamic topic space and the word space in a common shared dimension using deep learning.

Motivation

Importance of context in language models

Why do we need context for better language models? The aim of a language model is to predict the next best possible word given the current and past history of words. Not knowing the context of the current word or those historically seen will lead to high perplexity. Perplexity measures

Proposal for 10-708 Probabilistic Graphical Models course, Carnegie Mellon University, Pittsburgh PA 15213

the ”branchiness” of the next word to be predicted. For example, consider the phrase - Book me a ___. If we try to predict the next word, the number of possibilities are huge since there are many accounts of occurrence of ”a” followed by a word or ”me a” followed by a word, and so on. However, if we know the context of the current word, history of words, and that of the document, we can narrow our choices and bring down the perplexity of the language model. For example, If we know that in the candidate phrase - Book me a ___, the current word is a determiner and that the document was about travel and that noun is followed by a determiner in majority of the cases, we have cut down the number of choices of the possible words after ”a” to the nouns that are followed by ”a” in documents that are related to travel. So rather than predicting words like fish, sun, Obama we now have smaller possible set of words like restaurant, flight, train, or seat!

Exploiting transfer learning for better predictions

Transfer learning is the ability of a learning algorithm to exploit commonalities between different learning tasks and transfer knowledge across tasks (as explained in (Bengio et al., 2013)). In this project, we plan to explore the transfer learning aspect of RNNs to simultaneously predict the next context and the word. Our Context Generating RNN (CGRNNLM) model has two simultaneous tasks - predicting the next context and predicting the next word, which are although different, capture different dimensions of the semantics in sentences. Both of these tasks however share the same weights. In training to perform this task, we hope the knowledge gained by the RNN in predicting future contexts (given the current context and word) will transfer over to better predict future words (and vice versa). An overview of our model can be found in Figure 1.

The problem then boils down to choice of representation for the contexts. One natural choice is the topic distributions of the current and previous words. Here, our model can be considered to simulate dynamic LDA like behavior to generate contexts, similar to (Blei & Lafferty, 2006), but without enforcing assumptions about the shapes of topic distributions evolving with time, and instead learning the transitions from a data driven approach. The exploratory work in this project is to come up with different ways to represent the context vectors f(t) to train the RNN/LSTM network.

2. Related Work

2.1. Context dependent recurrent neural network language model (Mikolov & Zweig, 2012)

The paper improves the performance of the Recurrent Neural Network Language Model(Mikolov et al., 2010) by adding a context vector associated with each word. This vector is represented using Latent Dirichlet Allocation that avoids the data fragmentation associated with the traditional process of building multiple topic-specific language model. They also develop a fast technique to update the context vector with sliding windows. Finally, they evaluate the models by rescoring N-best lists from a speech recognizer and observe improvements.

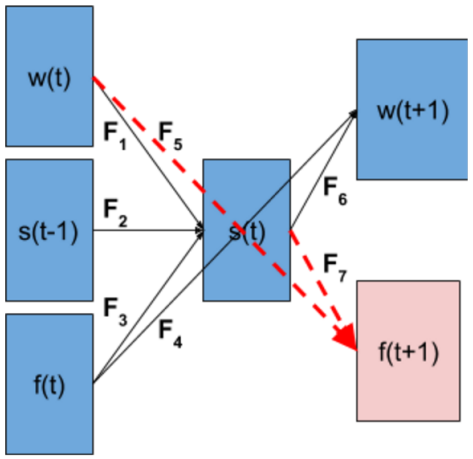

Figure 1. A picture of our proposed RNNLM model extending the model from (Mikolov & Zweig, 2012). The blocks in blue are already part of the RNNLM model in (Mikolov & Zweig, 2012) and we plan to extend that architecture to also generate context vectors. The future context vector is inferred from the RNNs hidden state and the current word.

2.2. Latent Dirichlet Allocation (Blei et al., 2003)

The goal of topic modeling is to automatically discover the topics from a collection of documents. LDA is the simplest topic modelling approach that exploits the idea that documents exhibit multiple topics. Each document exhibits the topics in different proportion and each word is drawn from one of the topics, where the selected topic depends on the previous document distribution over topics. In our proposal, we plan to learn the dynamic transitions between different distributions of topics (as part of the training process), given the history of change in topics and the current word.

2.3. Dynamic Topic Models (Blei & Lafferty, 2006)

A simple LDA model assumes that the order of the document doesn't matter, while this assumption may be unrealistic if we are analyzing long-running collections that span years. Dynamic Topic Model provides a solution to the problem by respecting the order of the documents. Instead of a single distribution over words, topic is now a

sequence of distribution over words, which gives a richer posterior topical structure than LDA. The timescale here to make Dynamic LDA effective, may be too long to be significantly useful for features used to train the context vectors in our proposal.

2.4. Generating text with recurrent neural networks (Sutskever et al., 2011)

The RNN's high dimensional hidden state and nonlinear evolution gives it a rich representative power that integrates information over many time-steps and use it to make more accurate prediction. Predicting the next word by making a sequence of character predictions avoids having to use a huge softmax over all known words and this is so advantageous that some word-level language models actually make up binary spellings of words so that they can predict them one bit at a time. (Mnih & Hinton, 2009)

2.5. Recurrent neural network based language model (Mikolov et al., 2010)

This paper serves as the background for the (Mikolov & Zweig, 2012) paper. It deals with the sequential prediction problem when constructing language models. The authors have trained RNN based language models which exhibit considerably less perplexity when compared to state of the art back-off language models. With this methodology, the words in the sentences are projected into lower dimension where similar words get clustered together. The authors have used backpropagation through time algorithm for training the network and shown that RNNs can form short-term memory to deal with position invariance.

3. Proposed Method

As explained in the overview, we plan to augment the RNNLM training with different variations of context vectors, starting with topic distributions output from LDA to represent context.

In our CGRNN model, we denote $w(t)$ as a 1-hot vector of input word at time t and with length V (the size of the vocabulary), $s(t)$ as the hidden state at time t. As mentioned in (Mikolov & Zweig, 2012), we extend the model with an additional feature vector $f(t)$, which here we refer to the context vector. It connect both the hidden and output layers and it can also be viewed as an external input vector that might contain complementary information to the input word vector $w(t)$. The output $w(t+1)$ is the distribution over the vocabulary for the next word and $f(t+1)$ represents the context vector to be used for the next word.

As discussed in the related work, applying topic model to our task has following contribution. First, the training data will be less fragmented because we avoid building many separate topic-specific models. The gradients of RNN are easy to compute via back-propagation through time, so we often intuitively trained RNN with gradient descent. However, the relationship between the parameters and the dynamics of the RNN is highly unstable which makes gradient descent ineffective for carrying backpropagation through time (BPTT) for long durations.

The values in the hidden and output layers are computed as follows:

$$\begin{aligned}
\mathbf{s}(t) &= a_1(F_1\mathbf{w}(t) + F_2\mathbf{s}(t-1) + F_3\mathbf{f}(t)) \\
\mathbf{w}(t+1) &= a_2(F_4\mathbf{f}(t) + F_6\mathbf{s}(t)) \\
\mathbf{f}(t+1) &= a_1(F_5\mathbf{w}(t) + F_7\mathbf{s}(t)) \\
a_1(z) &= \frac{1}{1+e^{-z}} \\
a_2(z_m) &= \frac{e^{z_m}}{\sum_k e^{z_k}}
\end{aligned}$$

Context representation

CONTEXT WITH LDA BETA VECTORS

Our use of LDA is motivated by the fact that it is one of the most widely used topic modeling approaches, and it is a generative probabilistic model. In our project, we have implemented a LDA based context that uses the beta vectors obtained by training the model on the training corpus. These vectors are trained only once and are later fed into the system in addition to the word vector representation. Each induced topic has associated with a unigram distribution over words, and the collection of distributions is denoted β. The generation process of a document is detailed in (Blei et al., 2003).

ADDITIONAL CONTEXT WITH POS TAGS

We plan to explore the impact of POS (part of speech) attributes of previous words as the context. Here, we plan to construct a vocabulary of part of speech tags and use a sliding window history of last K POS tags and augment the context vector.

4. Data Collection

We use Penn Treebank WSJ portion (sec 0 - sec 24) for our further experiments. The dataset has been divided into three chunks - training set (sec 0 - sec 20), validation set (sec 21 - 22), and test set (sec 23 - sec 24). Since the data is in POS parsed form, we use NLTK (Loper & Bird, 2002) to read the documents in plain text from the corpus and train our model. The POS tags from the corpus will be used as context features. The corpus consists of 1,253,013 words, 49,817 word types, and 2,312 documents. So, we use standard techniques like stop-word removal to bring down the vocabulary size.

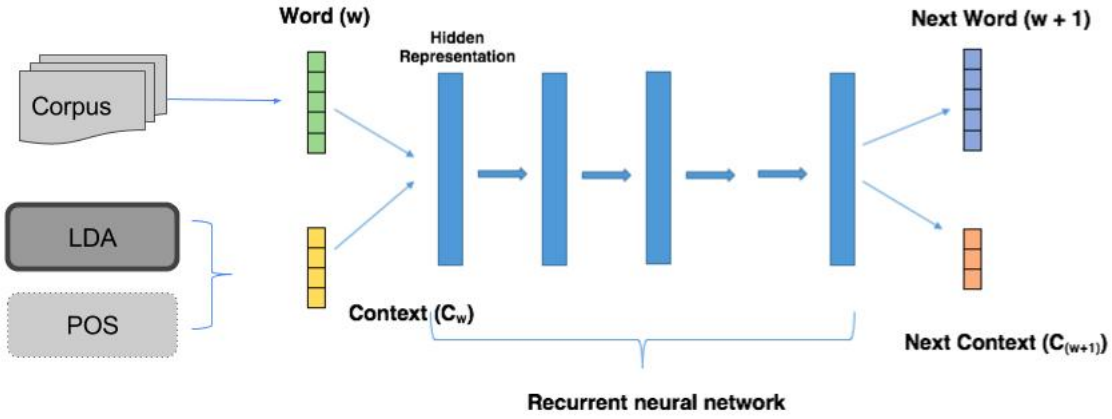

Figure 2. A picture of our proposed CGRNNLM model

5. Proposed system

The overall design of our system is shown in Figure 4. The systems comprises of two independent components - context generation and the RNN component in terms of training. We rely on off-the-shelf tools to generate context vectors as inputs to our model. For computing the LDA beta vectors we used the Gensim (Řehůřek & Sojka, 2010) library. For the token wise POS tags during pre-processing we rely on the Spacy (http://spacy.io/) library. Both the libraries are available for Python programming language. For training the RNNLM, we used a fast C++ RNNLM implementation publicly available from (fas) .

5.1. Learning context vectors

In this section, we describe our approach to compute LDA vectors and POS tags.

To train the context features with LDA beta vectors we use the sections 00-21 of the WSJ Penn treebank. This training set corresponds to the training set which is later used to train the RNN as well. The gensim code package internally uses the online variational inference approach to Latent Dirichlet Allocation developed in (Hoffman et al., 2010). The training proceeds as follows.

We take the treebank documents one by one and add it to the LDA model in an online fashion. The model makes number of passes over the collection of documents and the learnt vocabulary and gives out the expected value of the log of beta vectors for each term. Eventually, corresponding to each word we have a word-topic distribution which is stored into a file for later use by the RNN.

We also pre-process sentences from the dataset. We replace all the terms containing numerals into a literal 'N' denoting 'numeral' and replace out of vocabulary words with the literal '<unk>' denoting 'unknown' words. We don't remove any other word and rely upon the neural network to learn representations like stop-words since otherwise the generated sentences won't be fluent and are likely to be less grammatical.

Training POS tags on a large corpus is generally a time consuming task. But in case of our dataset, we initially made the choice to go with the POS annotated version of the Penn treebank. This saves our effort on generating the POS related context features from scratch. Rather we just extract the POS tags from the parse tree representation of the sentences in the corpus. To accomodate for the 'N' and '<unk>' literal, we manually add those labels into the tagset to form the final context matrix. As a result, we a matrix of the term - POS entries, we each entry is either 0 or 1 denoting if a particular word has the corresponding POS tag or not. To be mentioned, a word can have multiple POS tags based on the context of their appearance. For example, the word 'book' can be noun (the physical object) or a verb (e.g. to book a flight)in different contexts.

Finally, to combine the above two context features into a single matrix we concatenate the two vectors corresponding to a particular word at the time of training the RNN. This gives us the hybrid context.

5.2. Training CGRNNLM

We train the RNN on the same sections of the WSJ treebank as we did for the LDA component earlier. To train, we process each sentence one a time as a batch of words.

These sentences are represented using the one-hot encoding of the words in the sentence. These word vectors are then concatenated with the feature vectors when training the network. We want the RNN to learn a joint representation of the word and it's different contexts. As the learning proceeds, we use different kinds of neuron activations in the network. The gradients obtained at the output layer coming from the word part and the context part are then summed up and back propagated through time to update the weights of the network. We experiment with different loss functions for the two tasks (vocabulary prediction and context prediction). According to our hypothesis, this context generation should narrow down the perplexity of the system. We have significantly modified the original faster-rnnlm code to made it amenable to the our choices of the losses and computations involving the context feature matrices. All the operations have been computation efficient using the C++ Eigen library for matrix algebra operations. Also, we use learning rate decay in our system which allows us to obtain better entropy values for the test set than the system which doesn't use the decay heuristic.

5.3. Testing CGRNNLM

To test the RNN, we use the remaining four sections 21-24 of the treebank. Out of these four, the first two sections are used for validation and the last two for testing purpose. We apply the same pre-processing to test set that we applied to the training set. We report two metrics on the test set namely entropy and perplexity. Entropy is calculated because it is easier to compute with natural computations happening at the output layer. It is the average log likelihood of the words predicted by the RNN. Perplexity is more human understandable when it comes to comparison among different models. Computation wise perplexity is $2^{entropy}$. However, in general terms perplexity is a measure of number of possible choices of words that the trained LM predicts on average for each position in a sentence. Since the entropy is computed at every step of the model iteration, we can sample from the language model and simultaneously see the log likelihood of the sentences that are generated by the system. A good language model should assign low probability scores to long sentences since they tend to lose coherence. These results have been report in the experiments section.

5.4. Tuning CGRNNLM

The overall system has lot of hyper-parameters to tune. Starting with the LDA beta vectors, we experiment with different number of topics. We have tried setting the number of LDA topics to 10, 20, and 30. The POS tags are however fixed to a tagset of size 45. In terms of tuning the RNN, we have experimented with different configurations. We run a grid search on RNN parameters like BPTT interval, weightage given to context features, type of neurons, hidden layer size, and number of hidden layers. The results corresponding to these parameter settings have been reported in the following section. The feasibility of this exhaustive search can be attributed the choice of using the faster-rnnlm implementation which gives high processing throughput by parallelizing the computations among a pool of threads.

6. Experiments

Our effort builds on previous learnings and results from a fast RNNLM implementation with results comparable to the state of the art. It lets us reuse a few basic features such as changing the hidden unit type, number of hidden units, etc. Our goal is to analyze which parameters best integrate with our hybrid graphical model containing the vocabulary prediction and context prediction.

To evaluate our approach, we observe the impact of different parameters on our validation entropy and finally evaluate our test entropy (and test perplexity). Since this system involves many free parameters, such as the type of the hidden unit, learning rates, interval for backpropagation through time (BPTT), etc, we follow a systematic approach to assess the impact of each parameter and then perform an exhaustive grid search over a few parameters near the best validation entropy rates.

Performing an exhaustive grid search over all parameters is not tractable as this leads to thousands of combinations. In analyzing each parameter, we keep all other parameters fixed to a few 'valid' default values. It is possible to have obtained better results by changing multiple dimensions at the same time, but we reserve that grid search to a much smaller search space around the best parameters (based on validation entropy).

Unless being specifically analyzed, following is our default configuration for each experiment:

Parameter	Value
Hidden type	Sigmoid
Hidden size	128
Context loss weight	0.5
Number of hidden layers	1
NCE	30
Base learning rate	0.1
Context loss type	Softmax
Topic size	20
BPTT	5

6.1. Impact of hidden type

We vary the hidden type between 'Sigmoid', 'tanh', 'relu', 'gru', 'gru-full', 'gru-insyn' and analyze validation entropy

and final test entropy. We observe that the gru layers outperform sigmoid and relu layers by a significant amount, and performance is comparable between the different kinds of gru-layers.

Following are our results.

6.1.1. RESULTS

Hidden type	Test entropy
gru-bias	7.03
gru-full	7.05
gru-insyn	7.05
gru	7.00
relu	9.22
sigmoid	7.31
tanh	7.11

6.2. Impact of hidden layer size

We experiment with three configurations of hidden size $h = 100, 200, 300$. Since we're using a default topic size of k =20, the actual hidden layer size is 120, 220 and 320 respectively. In the final unrolled RNN, the first h outputs are fed to the softmax loss and the last 20 outputs are used to compute the context loss.

We observe that changing the hidden layer size hasn't shown significant changes in performance.

6.2.1. RESULTS

Hidden size	Test entropy
120	7.32
220	7.29
320	7.44

6.3. Impact of context loss weight

We analyze the importance of our implemented context loss weight paramete. The final output layer of size h = $|N| + |C|$ results in two outputs. The first N components are used to compute the loss for the predicted word (NCE sampling in this case), while the last C components are used to compute the loss wrt the context. The gradients from each loss are combined based on the loss weight assigned to each loss. We fix a loss weight of 1 for the loss from the word predictions and vary the loss weight as shown in the table below. We observe that having a loss weight 1 shows slightly better results than higher or much lower fractions.

6.3.1. RESULTS

Context loss weight	Test entropy
0.1	7.33
0.5	7.32
1	7.26
5	7.32
10	7.30
50	7.35

6.4. Impact of context loss type

In our CG-RNNLM implementation, we have the same hidden layer size $h = N + C$ for the unrolled RNN. The first N components are used to compute an NCE loss, while the last C components are used to compute the loss wrt the ground truth context. Here, our ground truth for the word at time t w_t is the context for the next word c_{t+1}. We have multiple options to compute the context loss, using an L1 loss, L2 loss or a softmax loss. The context vector, if we're using pure LDA is a prediction of the $log\beta$ column for the next word. The exponential of this column is a natural probability distribution over the C topics. Here the size of C is small (< 100), so it enables us to compute a direct softmax loss efficiently (instead of computing hierarchical softmax or NCE versions). We present results for using different kinds of loss functions below.

We observe that we obtain better results using a softmax loss regression given the semantic nature of the probabilities output by the function, instead of trying to use standard L2 or L1 regression. Results are shown below.

6.4.1. RESULTS

Context loss type	Test entropy
L_1 loss	7.37
L_2 loss	7.41
$Softmax$ loss	7.26

6.5. Impact of hidden layer count

We modify the number of hidden layers to assess impact on performance.

We observe that with increasing number of layers, the network becomes either harder to train or overfit the data and fails to generalize to testing. We obtain best performance with 1 hidden layer. The training however for each layer is performed by unrolling the RNN and performing back-propagation through time for a set number of steps.

6.5.1. RESULTS

Hidden layer count	Test entropy
1	7.26
3	7.58
5	7.85
10	9.24

6.6. Impact of LDA topic size

We vary the number of topics used for LDA training (and thus the size of the context vector). This also impacts the number of hidden units used since using N outputs to predict the next word implies, we use a hidden layer size of $h = N + C$. We present our results below.

We observe best performance for $K = 30$ topics for the Penn treebank dataset.

6.6.1. RESULTS

LDA topic size	Hidden size	Test entropy
10	138	7.29
20	120	7.32
20	148	7.32
30	158	7.24
40	168	7.32

6.7. Impact of BPTT length

Our RNN implmentation uses the back propagation through time algorithm (Werbos, 1990) for an unrolled RNN of T steps. We tried different intervals for (5, 10, 15, 20) steps and this didn't produce significant performance differences.

6.7.1. RESULTS

BPTT	Test entropy
5	7.32
10	7.29
15	7.29
20	7.30

6.8. Impact of NCE samples and Hierarchical softmax

We experimented with the number of NCE samples (Gutmann & Hyvärinen, 2010) used to compute the loss in predicting the next word. We observe that 20-30 samples are sufficient to produce reasonable results and there are no significant performance changes on changing the number of samples. We also observed that NCE outperforms Hierarchical softmax loss in most cases. Results are shown below.

6.8.1. RESULTS

NCE Sample size	Test entropy
10	7.36
20	7.35
30	7.31
40	7.33

To contrast, we repeated the same experiment with a hierarchical softmax layer and the test entropy was 7.62, which is significantly worse than using an NCE loss.

We choose the best models obtained by the above results and perform experiments by augmenting the context vector with POS tags and computing cumulative contexts from the word history. We describe the methods and results below. The following is our configuration for the below experiments.

Parameter	Value
Hidden type	gru
Hidden size	200
Context loss weight	1
Number of hidden layers	1
NCE	30
Base learning rate	0.1
Context loss type	Softmax
Topic size	20
BPTT	5

6.9. Impact of Adding POS tags to the context

Here we apply two softmax losses, the final hidden layer size $h = N + L + P$, where N units are used to predict the next word ("hidden size" in the above table), L to predict the next LDA vector (L is the number of topics) and P to predict the next POS vector (P is the number of possible POS tags). Adding extra POS tags produced comparable performance. Here, we set L=20 and P=45.

6.9.1. RESULTS

POS inclusion	Test entropy
POS inclusion	7.00
POS exclusion	6.96

6.10. Impact of adding multiple contexts

We follow a similar approach followed in (Mikolov & Zweig, 2012) to compute a cumulative context in the sentence.

We model this computation as

$$f(t) = f(t-1)^{\gamma} c(t)^{1-\gamma}$$

Where $f(t)$ is used as the context vector for the cur-

rent word, $c(t)$ is a probability distribution over topics provided by LDA. We vary the hyperparameter γ with different values.

Since our LDA interface provides $log(\beta_{kw})$ instead of β_{kw} vectors for numerical stability, we compute the contexts as

$$g(t) = \gamma g(t-1) + (1-\gamma) log(\beta_{,w(t)})$$

We observe that tweaking the value of gamma doesn't significantly affect performance. We think this might be due to the RNN capturing the changing trends automatically and the hand engineered combination may be redundant.

6.10.1. RESULTS

γ	**Test entropy**
0	6.969562
0.1	6.975844
0.2	6.982730
0.3	6.969541
0.4	6.977050
0.5	6.982139
0.6	6.988293
0.7	6.979129
0.8	6.970707
0.9	7.018784

6.11. Comparison with baseline RNNLM and Context-Dependent RNNLM

We finally compare our best models against the baseline RNNLM implementation as well as (Mikolov & Zweig, 2012).

We note that we perform comparable to (Mikolov & Zweig, 2012) on the Penn Treebank dataset, with a slight improvement in performance. Since the focus of this effort was to explore enhancements to the RNN graphical model, we do not report results on adding KN5/Max-ent layers for the classification. Addition of these layers is orthogonal to the graphical model and we expect improved performance in a manner similar to the results reported in (Mikolov & Zweig, 2012). We quote baseline RNNLM and Context Dependent RNNLM results from (Mikolov & Zweig, 2012)

Method	**PPL**($2^{entropy}$)
Baseline RNNLM	142.1
100-neurons, 40 topics (Mikolov & Zweig, 2012)	128.1
Ours (H=100, pos=45, 20 topics)	147.13
Ours (H=120, no-pos, 20 topics)	129.1
Ours (H=165, pos=45, 20 topics)	129.1
Ours (H=220, no-pos, 20 topics)	**125.1**
Ours (H=265, pos=45, 20 topics)	**125.1**

7. Discussion

In this project, we explored several key motivations - Firstly the importance of context in word predictions, Secondly our motivation to simplify and scale the learning process for dynamic topic models using RNNs and thirdly using joint learning techniques to simultaneously learn to predict the next word and the next context. These insights were inspired from recent results in computer vision applying joint learning, to obtain state of the art results (Girshick, 2015).

Since this is a complicated system with multiple parameters, we provided a systematic approach to understand the impact of each parameter for the best performance, by performing validation experiments on a validation set and summarizing the final test entropies on the Penn Treebank dataset.

We have taken the best models discovered from the above approach and performed further experiments by augmenting the context vector with POS tags and demonstrated results.

Finally we compare our approach with the baseline RNNLM and context dependent (Mikolov & Zweig, 2012) models and show that we're able to achieve slightly better, if not comparable results.

8. Future work

From our results so far, we see that our performance is comparable to the baseline RNNLM model and the context dependent RNNLM model (Mikolov & Zweig, 2012), with the topic size and choice of hidden unit being one of the important factors.

Since our network is given more information with context and also constrained to learn a hidden representation to jointly model the language and a dynamic topic model, we think the design of the network naturally provides a self-regularized scenario. Since both tasks - word prediction and context prediction use shared weights, the loss from the first task regularizes/constrains the learning of the second task. However, we think investigating the impact of dropout in such networks would be an interesting extension to analyze if certain layers are specialized to predict exactly one of these tasks.

Further, it would be interesting to visualize arithmetic operations (such as additions/subtractions/cosine distance) using the learned representations to see if any semantic trends can emerge.

Finally, it would be interesting to extend this to model to analyze long term temporal trends.

References

Faster rnnlm. https://github.com/yandex/faster-rnnlm.

Bengio, Yoshua, Courville, Aaron, and Vincent, Pierre. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

Blei, David M and Lafferty, John D. Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, pp. 113–120. ACM, 2006.

Blei, David M, Ng, Andrew Y, and Jordan, Michael I. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

Girshick, Ross. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

Gutmann, Michael and Hyvärinen, Aapo. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

Hoffman, Matthew, Bach, Francis R, and Blei, David M. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pp. 856–864, 2010.

Loper, Edward and Bird, Steven. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pp. 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1118108.1118117. URL http://dx.doi.org/10.3115/1118108.1118117.

Mikolov, Tomas and Zweig, Geoffrey. Context dependent recurrent neural network language model. In *SLT*, pp. 234–239, 2012.

Mikolov, Tomas, Karafiát, Martin, Burget, Lukas, Cernockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, pp. 3, 2010.

Mnih, Andriy and Hinton, Geoffrey E. A scalable hierarchical distributed language model. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L. (eds.), *Advances in Neural Information Processing Systems 21*, pp. 1081–1088. Curran Associates, Inc., 2009. URL http://papers.nips.cc/paper/3583-a-scalable-hierarchical-distributed-language-model.pdf.

Řehůřek, Radim and Sojka, Petr. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

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

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

Yuan, Jinhui, Gao, Fei, Ho, Qirong, Dai, Wei, Wei, Jinliang, Zheng, Xun, Xing, Eric Po, Liu, Tie-Yan, and Ma, Wei-Ying. Lightlda: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pp. 1351–1361, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3469-3. doi: 10.1145/2736277.2741115. URL http://doi.acm.org/10.1145/2736277.2741115.