

ROHOLLAH SOLTANI
EMAIL: RSOLTANI@CSE.YORKU.CA

Data Mining Project
Instructor: Aijun An

Project Website : <http://www.cse.yorku.ca/~rsoltani/cse6412/>

LANGUAGE MODELING USING RECURRENT NEURAL NETWORK

DECEMBER 2014

In this project various artificial neural networks architectures of language models are presented. Although these models are computationally more expensive than N-gram models, with some techniques it is possible to apply them to state-of-the-art systems efficiently. Advantage of the recurrent architecture over the feed-forward one is that the hidden layer of RNN represents all previous history and not just $n-1$ previous words, thus the model can theoretically represent long context patterns and can represent more advanced patterns in the sequential data. RNN can be successfully trained by using stochastic gradient descent and back-propagation through time and Great speedup can be achieved by using simple classes in the output layer. We show that using augmented features RNN we can reach good result in terms of accuracy as well as speed.

Contents

Introduction	4
Related work	6
N-gram	6
Artificial neural networks	7
Neural network based language modeling	8
Feed forward neural network based language model	8
Approach	10
Recurrent neural networks	10
Recurrent neural network based language model	10
Learning Algorithm	12
back-propagation through time	13
Computational Complexity	14
Extensions of RNNLMs	16
Class based RNNLM	16
Using Distribution representation of words	17
Feature augmented method	19
Experimental Result	21
Perplexity	21
Penn Treebank Dataset	21
1 billion words Benchmark	21
Results	22
Conclusion	27

LM USING RNN

Reference	28
Appendix	30
systems manual	30
Online Version	31

Introduction

Language models are a critical component in many speech and natural language processing technologies, such as speech recognition and understanding, voice search, conversational interaction and machine translation. Over the last few decades, several advanced language modeling ideas have been proposed. Some of these approaches have focused on incorporating linguistic information such as syntax and semantics whereas others have focused on fundamental modeling and parameter estimation techniques.

Sequential data prediction is considered by many as a key problem in machine learning and artificial intelligence. The goal of statistical language modeling is to predict the next word in textual data given context; thus we are dealing with sequential data prediction problem when constructing language models. Still, many attempts to obtain such statistical models involve approaches that are very specific for language domain - for example, assumption that natural language sentences can be described by parse trees, or that we need to consider morphology of words, syntax and semantics. Even the most widely used and general models, based on n-gram statistics, assume that language consists of sequences of atomic symbols - words - that form sentences, and where the end of sentence symbol plays important and very special role.

I have decided to investigate recurrent neural networks (RNN) for modeling sequential data. RNNs have several properties that make them an attractive choice for sequence labelling: they are flexible in their use of context information (because they can learn what to store and what to ignore); they accept many different types and representations of data; and they can recognize sequential patterns in the presence of sequential distortions.

Recurrent neural network language models (RNNLMs) have recently demonstrated state-of-the-art performance across a variety of tasks. These networks differ from classical feed-forward neural network language models [1, 2, 3, 4, 5] in that they maintain a hidden-layer of neurons with recurrent connections to their own previous values. This recurrent property gives a RNNLM the potential to model long span dependencies.

LM USING RNN

I report perplexity results on the Penn Treebank data which is one of the most widely used data sets for evaluating performance of the statistical language models and recently released 1 billion words benchmark dataset which is proposed to be used for measuring progress in statistical language modeling.

Related work

N-gram models and feed-forward neural network models are today considered as state of the art. Which will be briefly explained here.

N-GRAM

The probability of a sequence of symbols (usually words) is computed using a chain rule as

$$P(w) = \sum_{i=1}^N P(w_i | w_1 \dots w_{i-1})$$

The most frequently used language models are based on the n-gram statistics, which are basically word co-occurrence frequencies. The maximum likelihood estimate of probability of word A in context H is then computed as

$$P(A|H) = \frac{C(H A)}{C(H)}$$

Where $C(H A)$ is the number of times that the HA sequence of words has occurred in the training data. The context H can consist of several words, for the usual trigram models $|H| = 2$. For $H = 0$, the model is called unigram, and it does not take into account history.

As many of these probability estimates are going to be zero (for all words that were not seen in the training data in a particular context H), smoothing needs to be applied. This works by redistributing probabilities between seen and unseen (zero-frequency) events, by exploiting the fact that some estimates, mostly those based on single observations, are greatly over-estimated. Detailed overview of common smoothing techniques and empirical evaluation can be found in [6]

The most important factors that influence quality of the resulting n-gram model is the choice of the order and of the smoothing technique.

The most significant advantages of models based on n-gram statistics are speed (probabilities of n-grams are stored in precomputed tables), reliability coming from simplicity, and generality (models can be applied to any domain or language effortlessly, as long as there exists some training

data). N-gram models are today still considered as state of the art not because there are no better techniques, but because those better techniques are computationally much more complex, and provide just marginal improvements, not critical for success of given application.

The weak part of n-grams is slow adaptation rate when only limited amount of in-domain data is available. The most important weakness is that the number of possible n-grams increases exponentially with the length of the context, preventing these models to effectively capture longer context patterns. This is especially painful if large amounts of training data are available, as much of the patterns from the training data cannot be effectively represented by n-grams and cannot be thus discovered during training. The idea of using neural network based LMs is based on this observation, and tries to overcome the exponential increase of parameters by sharing parameters among similar events, no longer requiring exact match of the history H .

ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANNs) were originally developed as mathematical models of the information processing capabilities of biological brains [7, 8]. Although it is now clear that ANNs bear little resemblance to real biological neurons, they enjoy continuing popularity as pattern classifiers. The basic structure of an ANN is a network of small processing units, or nodes, joined to each other by weighted connections. In terms of the original biological model, the nodes represent neurons, and the connection weights represent the strength of the synapses between the neurons. The network is activated by providing an input to some or all of the nodes, and this activation then spreads throughout the network along the weighted connections. The electrical activity of biological neurons typically follows a series of sharp 'spikes', and the activation of an ANN node was originally intended to model the average firing rate of these spikes. Many varieties of ANNs have appeared over the years, with widely varying properties. One important distinction is between ANNs whose connections form cycles, and those whose connections are acyclic. ANNs with cycles are referred to as feedback, recursive, or recurrent, neural networks. ANNs without cycles are referred to as feed-forward neural networks (FNNs).

Neural network based language modeling

The main advantage of NNLMs over n-grams is that history is no longer seen as exact sequence of n-1 words H , but rather as a projection of H into some lower dimensional space. This reduces number of parameters in the model that have to be trained, resulting in automatic clustering of similar histories. While this might sound the same as the motivation for class based models, the main difference is that NNLMs project all words into the same low dimensional space, and there can be many degrees of similarity between words.

The main weak point of these models is very large computational complexity, which usually prohibits to train these models on full training set, using the full vocabulary. I will deal with these issues in this work by proposing simple and effective speed-up techniques.

Feed forward neural network based language model

The original model of feed-forward neural network based language model proposed by Bengio[1] as follows: the input of the n-gram NNLM is formed by using a fixed length history of n-1 words, where each of the previous n-1 words is encoded using 1-of-V coding, where V is size of the vocabulary. Thus, every word from the vocabulary is associated with a vector with length V , where only one value corresponding to the index of given word in the vocabulary is 1 and all other values are 0.

This 1-of-V orthogonal representation of words is projected linearly to a lower dimensional space, using a shared matrix P , called also a projection matrix. The matrix P is shared among words at different positions in the history, thus the matrix is the same when projecting word w_{t-1} , w_{t-2} etc. In the usual cases, the vocabulary size can be around 50K words, thus for a 5-gram model the input layer consists of 200K binary variables, while only 4 of these are set to 1 at any given time, and all others are 0. The projection is done sometimes into as little as 30 dimensions, thus for our example, the dimensionality of the projected input layer would be $30 * 4 = 120$. After the projection layer, a hidden layer with non-linear activation function (usually hyperbolic tangent or a logistic sigmoid) is used, with a dimensionality of 100-300. An output layer follows, with the size equal to the size of full vocabulary. After the network is trained, the output layer of 5-gram NNLM represents probability distribution $P(w_t | w_{t-4} \dots w_{t-1})$

LM USING RNN

The main weak point of these models is considering just the last $n-1$ words not the whole history this is why this model (like n -gram) is still limited to a predetermined number of words as history and cannot find more complicated patterns in the whole history for predicting the next word.

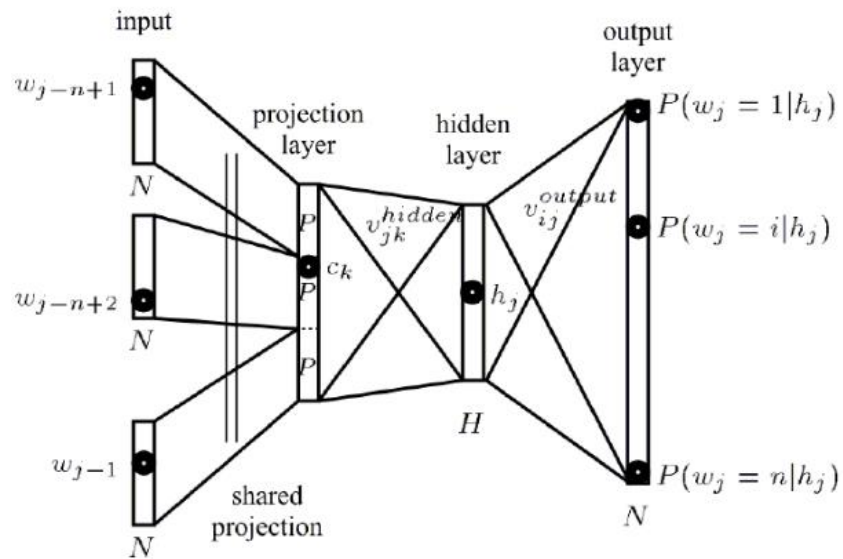


Figure 1 . Feedforward neural network based language model

Approach

In this project recurrent neural network based language models are presented. Although these model are computationally more expensive than N-gram models, with some techniques it is possible to apply them to state-of-the-art systems efficiently.

RECURRENT NEURAL NETWORKS

Recurrent neural networks (RNNs) are a class of artificial neural network architecture that inspired by the cyclical connectivity of neurons in the brain uses iterative function loops to store information. RNNs have several properties that make them an attractive choice for sequence labelling: they are flexible in their use of context information (because they can learn what to store and what to ignore); they accept many different types and representations of data; and they can recognize sequential patterns in the presence of sequential distortions. However they also have several drawbacks that have limited their application to real-world sequence labelling problems.

RECURRENT NEURAL NETWORK BASED LANGUAGE MODEL

The main difference between the feed-forward and the recurrent architecture is in representation of the history - while for feed-forward NNLM, the history is still just previous several words, for the recurrent model, an effective representation of history is learned from the data during training. The hidden layer of RNN represents all previous history and not just $n-1$ previous words, thus the model can theoretically represent long context patterns.

Another important advantage of the recurrent architecture over the feed-forward one is the possibility to represent more advanced patterns in the sequential data. For example, patterns that rely on words that could have occurred at variable position in the history can be encoded much more efficiently with the recurrent architecture - the model can simply remember some specific word in the state of the hidden layer, while the feed-forward architecture would need to use parameters for each specific position of the word in the history; this not only increases the total amount of parameters in the model, but also the number of training examples that have to be seen to learn the given pattern.

LM USING RNN

The architecture of RNNLM is shown in Figure 2. The input layer consists of a vector $w(t)$ that represents the current word w encoded as 1 of V (thus size of $w(t)$ is equal to the size of the vocabulary), and of vector $s(t-1)$ that represents output values in the hidden layer from the previous time step. After the network is trained, the output layer $y(t)$ represents $P(w_{t+1} | w_t, S(t-1))$.

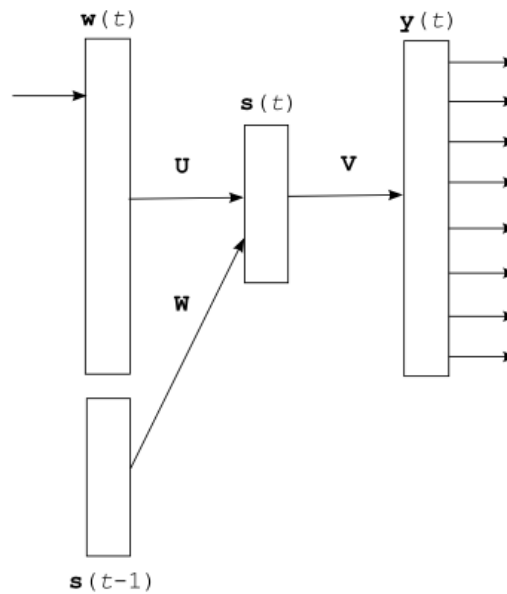


Figure 2 Recurrent Neural Network language modeling

The network is trained by stochastic gradient descent using either usual back-propagation (BP) algorithm, or back-propagation through time (BPTT) [9]. The network is represented by input, hidden and output layers and corresponding weight matrices – matrices U and W between the input and the hidden layer, and matrix V between the hidden and the output layer. Output values in the layers are computed as follows:

$$s_j(t) = f\left(\sum_i w_i(t) u_{ji} + \sum_k s_k(t-1) w_{jk}\right)$$

$$y_{k(t)} = g\left(\sum_j s_j(t-1)v_{kj}\right)$$

Where $f(z)$ and $g(z)$ are sigmoid and softmax activation functions (the softmax function in the output layer is used to ensure that the outputs form a valid probability distribution, i.e. all outputs are greater than 0 and their sum is 1):

$$f(z) = \frac{1}{1 + e^{-z}}, \quad g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

The output layer y represents a probability distribution of the next word w_{t+1} given the history. The time complexity of one training or test step is proportional to

$$O = H * H + H * V = H * (H + V)$$

Where H is size of the hidden layer and V is size of the vocabulary.

LEARNING ALGORITHM

With the stochastic gradient descent, the weight matrices of the network are updated after presenting every example. A cross entropy criterion is used to obtain gradient of an error vector in the output layer, which is then back-propagated to the hidden layer, and in case of BPTT through the recurrent connections backwards in time. During the training, validation data are used for early stopping and to control the learning rate. Training iterates over all training data in several epochs before convergence is achieved

The weight matrices U , V and W are initialized with small random numbers Training of RNN for one epoch is performed as follows:

1. Set time counter $t = 0$, initialize state of the neurons in the hidden layer $s(t)$ to 0
2. Increase time counter t by 1
3. Present at the input layer $w(t)$ the current word w_t
4. Copy the state of the hidden layer $s(t-1)$ to the input layer
5. Perform forward pass as described in the previous section to obtain $s(t)$ and $y(t)$

6. Compute gradient of error $e(t)$ in the output layer
7. Propagate error back through the neural network and change weights accordingly
8. If not all training examples were processed, go to step 2

BACK-PROPAGATION THROUGH TIME

The idea of Back-Propagation Through Time (BPTT) is simple: a recurrent neural network with one hidden layer which is used for N time steps can be seen as a deep feed-forward network with N hidden layers (where the hidden layers have the same dimensionality and unfolded recurrent weight matrices are identical). This idea has been described in [10], and is illustrated in Figure 3. Such deep feed-forward network can be trained by the normal gradient descent. Errors are propagated from the hidden layer $s(t)$ to the hidden layer from the previous time step $s(t-1)$ and the recurrent weight matrix (denoted as W in Figure 3) is updated. Error propagation is done recursively.

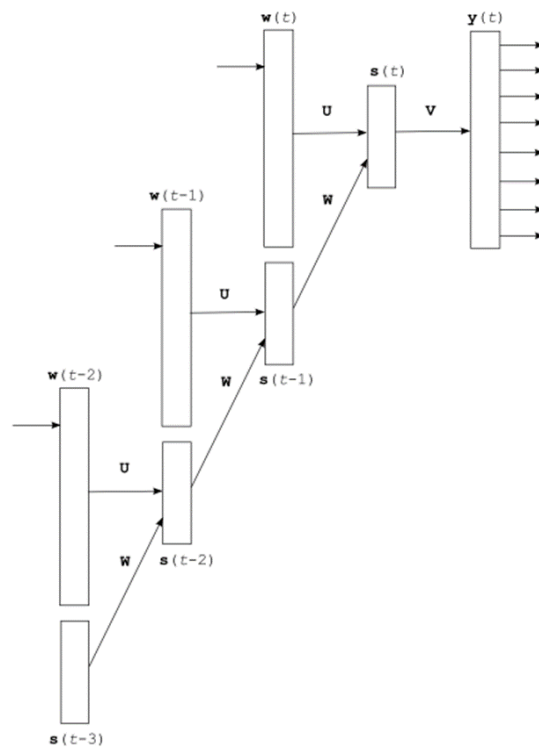


Figure 3 : Recurrent neural network unfolded as a deep feed-forward network, here for 3 time steps back in time

LM USING RNN

The unfolding can be applied for as many time steps as many training examples were already seen, however the error gradients quickly vanish as they get back-propagated in time [11] (in rare cases the errors can explode), so several steps of unfolding are sufficient (this is sometimes referred to as truncated BPTT).

COMPUTATIONAL COMPLEXITY

The computational complexity of a basic neural network language model is very high for several reasons, and there have been many attempts to deal with almost all of them. The training time of N-gram feed-forward neural network language model is proportional to

$$I \times W \times ((N - 1) \times D \times H + H \times V)$$

where I is the number of the training epochs before convergence of the training is achieved, W is the number of tokens in the training set, N is the N-gram order, D is the dimensionality of words in the low-dimensional space, H is size of the hidden layer and V size of the vocabulary (see Figure 4). The $(N - 1) \times D$ is equal to the size of the projection layer.

The recurrent neural network language model has computational complexity:

$$I \times W \times (H \times H + H \times V)$$

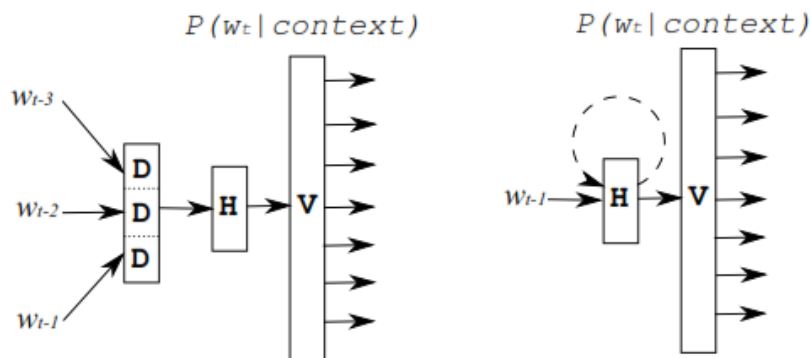


Figure 4

LM USING RNN

It can be seen that for increasing order N , the complexity of the feed-forward architecture increases linearly, while it remains constant for the recurrent one (actually, N has no meaning in RNN LM).

The largest terms in the previous three equations are W , the number of the training words, and V , the size of the vocabulary. Typically, W can be in order of millions, and V in hundreds of thousands.

Extensions of RNNLMs

CLASS BASED RNNLM

The original recurrent neural network language model is very computationally expensive, which severely limits its possible application in real world systems. Most modifications that aim to reduce the computational complexity attempt to overcome the huge term $H * V$ that corresponds to the computation done between the hidden and output layers. This computational bottleneck is the same for both feed-forward and for the recurrent architecture. Using reasonably large hidden layer such as $H = 200$ and vocabulary $V = 50K$, it would take impractically long to train models even on data sets with several million words. Moreover, application to speech recognition systems via n-best list rescoring would be many times slower than real-time.

A sophisticated approach for reducing the huge term $H*V$ was proposed in [12]. Instead of computing probability distribution over all words V or some reduced subset of the most frequent words, the probability is estimated for groups of words, and then only for words from a particular group that we are interested in.

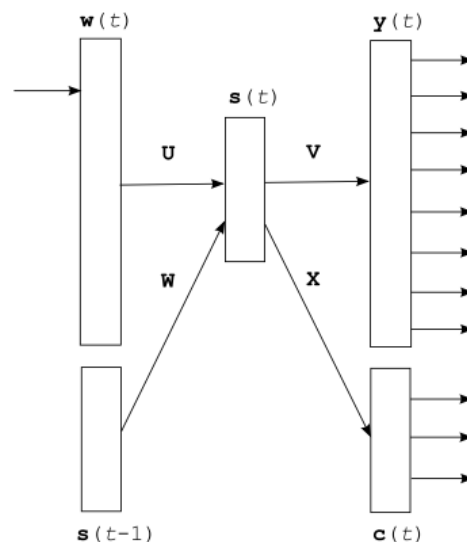


Figure 5 class based RNN LM

Figure 5 illustrates this approach: first, the probability distribution over classes is computed. Then, a probability distribution for the words that belong to the specific class are computed. So instead of computing V outputs and doing softmax over V elements, only $C + V'$ outputs have to be computed, and the softmax function is applied separately to both C and V' , where C are all the classes, and V' are all words that belong to the particular class. Thus, C is constant and V can be variable.

The computation between the hidden and the output layer changes to computation between the hidden and the class layer:

$$C_m(t) = g\left(\sum_j s_j(t)x_{mj}\right)$$

and the hidden layer and a subset of the output layer:

$$yV'(t) = g\left(\sum_j s_j(t)vV'_j\right)$$

The probability of word $w(t + 1)$ is then computed as

$$P(w_{t+1}|S(t)) = P(c_i|S(t)) P(w_i|c_i, S(t))$$

Where w_i is an index of the predicted word and c_i is its class. During training, the weights are accessed in the same way as during the forward pass, thus the gradient of the error vector is computed for the word part and for the class part, and then is back-propagated back to the hidden layer, where gradients are added together. Thus, the hidden layer is trained to predict both the distribution over the words and over the classes

USING DISTRIBUTION REPRESENTATION OF WORDS

Distributed representations of words in a vector space help learning algorithms to achieve better performance in natural language processing tasks by grouping similar words.

Recently, Mikolov et al. [13] introduced the Skip-gram model, an efficient method for learning high-quality vector representations of words from large amounts of unstructured text data. Unlike most of the previously used neural network architectures for learning word vectors, training of the Skip-

LM USING RNN

gram model (see Figure 5) does not involve dense matrix multiplications. This makes the training extremely efficient: an optimized single-machine implementation can train on more than 100 billion words in one day.

The word representations computed using neural networks are very interesting because the learned vectors explicitly encode many linguistic regularities and patterns. Somewhat surprisingly, many of these patterns can be represented as linear translations. For example, the result of a vector calculation $\text{vec}(\text{"Madrid"}) - \text{vec}(\text{"Spain"}) + \text{vec}(\text{"France"})$ is closer to $\text{vec}(\text{"Paris"})$ than to any other word vector [14, 13].

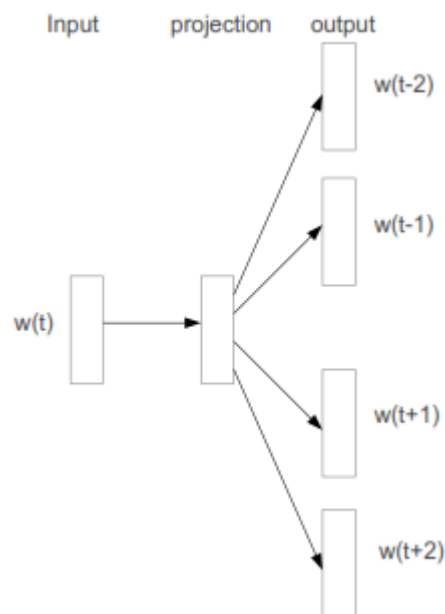


Figure 6 : The Skip-gram model architecture. The training objective is to learn word vector representations that are good at predicting the nearby words.

They made the code for training the word vectors based on their techniques available as an open-source project word2vec in <http://code.google.com/p/word2vec>.

The word2vec tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words.

LM USING RNN

The resulting word vector file can be used as features in many natural language processing and machine learning applications.

FEATURE AUGMENTED METHOD

Mikolov et al. [15] extend this basic model of RNN with an additional feature layer $f(t)$ that is connected to both the hidden and output layers, as shown in Figure 7. The feature layer represents an external input vector that should contain complementary information to the input word vector $w(t)$. The external features can be any information source such as part of speech tags, morphological information about the current word $w(t)$, or speaker-specific information in the context of ASR.

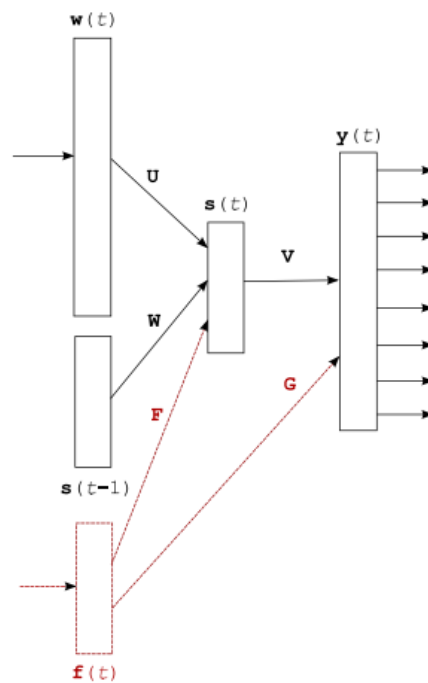


Figure 7 Recurrent neural network based language model, with the additional feature layer $f(t)$ and the corresponding weight matrices.

The distributed representation of words produced by skip-gram model has been used as feature vector for this model. The input vector $w(t)$ and the output vector $y(t)$ have dimensionality of the

LM USING RNN

vocabulary. After the network is trained using stochastic gradient descent, the vector $y(t)$ represents a probability distribution over words from the vocabulary given the previous word $w(t)$, the context vector $s(t-1)$ and the feature vector $f(t)$.

Experimental Result

It is very difficult to objectively compare different language modeling techniques: in practical applications, accuracy is sometimes as important as low memory usage and low computational complexity.

In my project, performance of models is reported and compared to other technique on Penn Treebank Dataset. I have also tried to get the result of my RNN on 1 billion benchmark but due to the lack of time I just manage to get the result on 10% of that data set. Since training RNN is very time consuming, the performance of RNN on whole 1-billion benchmark could not be ready before the deadline.

PERPLEXITY

Evaluation of quality of different language models is usually done by perplexity. The perplexity (PPL) of word sequence w is defined as

$$PPL = \sqrt[k]{\prod_{i=1}^k \frac{1}{P(w_i|w_1 \dots w_{i-1})}} = 2^{-\frac{1}{k} \sum_{i=1}^k \log_2 P(w_i|w_1 \dots w_{i-1})}$$

PENN TREEBANK DATASET

One of the most widely used data sets for evaluating performance of the statistical language models is the Penn Treebank portion of the WSJ corpus. I used the same standard preprocessing of the data as is common in previous research and all words outside the 10K vocabulary are mapped to the <unk> token. The Penn Treebank Corpus was divided as follows: sections 0-20 were used as the training data (930k tokens), sections 21-22 as the validation data (74k tokens) and sections 23-24 as the test data (82k tokens).

1 BILLION WORDS BENCHMARK

Chelba et. al., propose a new benchmark corpus to be used for measuring progress in statistical language modeling. With almost one billion words of training data [18]. This corpus consisted of

0.8 billion words which were split into 100 disjoint partitions. Vocabulary size is about 191K words.

The benchmark is available as a code.google.com project at <https://code.google.com/p/1-billion-word-language-modeling-benchmark/>; besides the scripts needed to rebuild the training/held-out data, it also makes available log-probability values for each word in each of ten held-out data sets, for each of the baseline n-gram models.

RESULTS

The first models are standard n-gram models with Good-Turing (GT) and modified Kneser-Ney smoothing (KN) [16]. The usual baseline in many papers is a trigram model with Good-Turing smoothing. We can see that substantial gains can be gained by using KN smoothing and also by using higher order n-grams. Although the PTB corpus is relatively small, the difference between GT3 and KN5 models is large perplexity is reduced from about 165 to 141. On larger data sets, even bigger difference can be expected. The rest models in Table 1 consists of neural network language models with different architectures.

There are a feedforward neural network models that was originally proposed by Yoshua Bengio [1] - the neural network learns a linear projection of words into a low dimensional space together with the final model.

By changing the topology of the network from a feedforward to a recurrent one, we allow the model to form a short context memory that is learned unsupervisedly from the data. The prediction of the next word then depends on the previous word and the state of the short context memory. We can thus claim that such model can actually cluster entire histories that are in some sense similar. This is in contrast to feed-forward neural networks that can effectively cluster only individual words in the projection layer, and then it is needed to perform another step to cluster the low dimensional representation of several words from the history. If some pattern involves variable position of some word in the history, it is not possible to represent such pattern efficiently with a compact feedforward network, while this can be accomplished by using a recurrent one.

The recurrent neural network language model that was described in more depth in the previous chapter outperforms all other types of language models on the PTB data set [17]. It works similar to the feedforward neural network language model, with the main difference being representation of the history. While for both feedforward and the recurrent architecture the history is projected into a lower dimensional space where clustering of similar events occur, the main difference is that feedforward NN projects individual words, and recurrent NN performs clustering of the whole histories. This gives the recurrent model ability to compactly describe wider range of patterns in the data.

In this project I have used truncated BPTT and Stochastic gradient descent (SGD) for training the RNN models. Error gradients were computed by performing unfolding of the RNN model for 10 time steps.

Our initial experiments were performed using RNN models with 150 neurons in hidden layer that it took about one week to train. Then For reduction of computational complexity, factorization of the output layer using 100 classes has been used.

I trained additional RNNML models with 150 hidden neurons with distribution representation of the words as features, to see the potential of vector representations of words in models and make training phase faster. I repeat this using words representation of words from bigger data set to have better representation of each word.

Next, I trained RNNML models with the same configuration and with additional features that I got from skip gram model on our dataset. The results are summarized in Table 1.

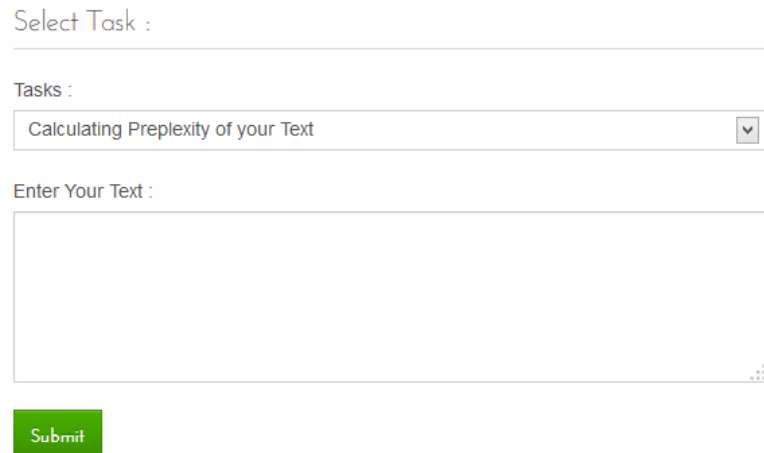
LM USING RNN

RESULTS			
MODEL	VALID PPL	TEST PPL	DESCRIPTION
3-gram, Good-Turing smoothing		165.2	
5-gram, Good-Turing smoothing		162.3	
3-gram, Kneser-Ney smoothing		148.3	
5-gram, Kneser-Ney smoothing		141.2	
Feed forward Neural Network		140.2	
RNN	139.2315	132.2373	12 iter. Each: 7hour
RNN(100 class)	148.3043	141.5747	19 iter. Each: 30 min
RNN(50 class)	147.1355	140.0587	19 iter.
RNN(200 Feature ,100 class)	178.3496	166.2796	15 iter
RNN(200 Feature google,100 class)	180.3263	173.29	14 iter.
RNN(100 class, 40 Feature Augmented)	134.7309	128.3221	20 iter.
RNN(50 class, 40 Feature Augmented)	133.8850	128.6912	17 iter
RNN(40 Feature Augmented)	118.1850	112.5353	18 iter,, each 12 h

As can be seen, the perplexity is reduced very significantly by using feature augmented method RNN. Smaller classes results in more output layer and thus can lead to a higher accuracy, at the expense of the training time.

LM USING RNN

You can get the perplexity of your text using the online version of my project in <http://www.cse.yorku.ca/~rsoltani/cse6412/>. Enter your text and find out the perplexity of that using RNN with 150 neuron in hidden layer (Figure 8)



The screenshot shows a web interface for calculating perplexity. It features a 'Select Task' dropdown menu, a 'Tasks' dropdown menu with 'Calculating Preplexity of your Text' selected, and a large text input area labeled 'Enter Your Text'. A green 'Submit' button is located at the bottom left of the input area.

Figure 8 Calculating Perplexity

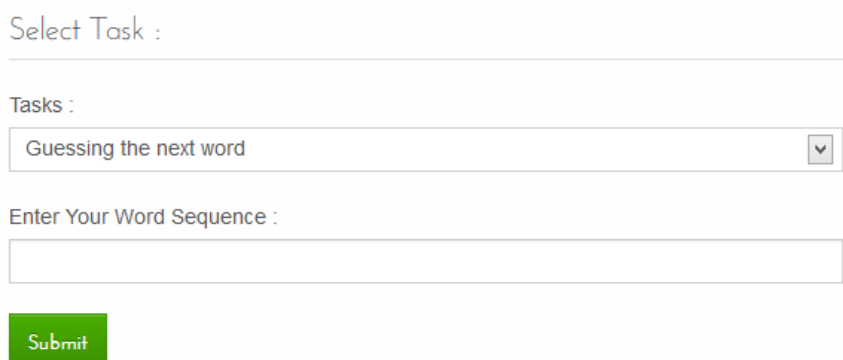
I have used these models to predict the next word of a sequence of words. For this aim I compute the word with highest probability in my RNN model given the sequence of words.

You can see some of the predictions of word for my RNN model with 150 neuron in hidden layer in table 2 that `</s>` means end of line:

SENTENCE	1 ST WORD	PROBOBILITY OF 1 ST WORD	2 ND WORD	PROB. OF 2 ND WORD	3 RD WORD	PROB. OF 3 RD WORD
It is easy	to	54.35%	</s>	6.99%	for	5.70%
It is	N't	13.87%	a	8.46%	also	4.25%
They will be	a	6.86%	used	5.41%	able	5.14%
I think	that	19.13%	the	16.84%	It	11.51%
that would be the	first	5.44%	lowest	4.33%	highest	2.93%

LM USING RNN

You can get the best three words for completing your sentence using the online version of my project in <http://www.cse.yorku.ca/~rsoltani/cse6412/> in enter your sequence of words and see the results using RNN with 150 neuron in hidden layer (Figure 9)



Select Task :

Tasks :

Guessing the next word

Enter Your Word Sequence :

Submit

Figure 9. Guessing the next word

In 1B-Word Benchmark corpus since my result is just on 10% of that I will not compare it to other technique.

Result on 10% of 1B-Word Benchmark corpus	Perplexity
RNN(100 class)	214

Conclusion

The goal of this project was to present architectures of language models that are based on artificial neural networks. Although these models are computationally more expensive than N-gram models, with some techniques it is possible to apply them to state-of-the-art systems efficiently. As we saw, Recurrent neural network language models can be successfully trained by using stochastic gradient descent and back-propagation through time and Great speedup can be achieved by using simple classes in the output layer.

The main difference between the feed-forward and the recurrent architecture is in representation of the history - while for feed-forward NNLM, the history is still just previous several words, for the recurrent model, an effective representation of history is learned from the data during training. The hidden layer of RNN represents all previous history and not just $n-1$ previous words, thus the model can theoretically represent long context patterns.

Another important advantage of the recurrent architecture over the feed-forward one is the possibility to represent more advanced patterns in the sequential data.

We also show that using augmented features RNN we can reach good result in terms of accuracy as well as speed.

Reference

- [1]. Y. Bengio, R. Ducharme, Vincent, P, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, no. 6, 2003.
- [2]. Holger Schwenk, "Continuous space language models," *Computer Speech and Language*, vol. 21, no. 3, pp. 492 – 518, 2007.
- [3]. Andriy Mnih and Geoffrey Hinton, "Three new graphical models for statistical language modelling," in *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 641–648.
- [4]. Hai-Son Le, I. Oparin, A. Allauzen, J.-L. Gauvain, and F. Yvon, "Structured output layer neural network language model," in *ICASSP*, 2011.
- [5]. Le Hai Son, Alexandre Allauzen, and Francois Yvon, "Measuring the influence of long range dependencies with neural network language models," in *Proceedings of the Workshop on the Future of Language Modeling for HLT (NAACL/HLT 2012)*, 2012.
- [6]. S. F. Chen, J. T. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting of the ACL*, 1996.
- [7]. W. S. McCulloch and W. Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*, pages 15{27. MIT Press, 1988.
- [8]. S. F. Chen, J. T. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting of the ACL*, 1996.
- [9]. D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by back-propagating errors. *Nature*, 323:533.536, 1986.
- [10]. M. Minsky, S. Papert. *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.
- [11]. Y. Bengio, P. Simard, P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5, 157-166, 1994.
- [12]. F. Morin, Y. Bengio: Hierarchical Probabilistic Neural Network Language Model. *AISTATS*, 2005.
- [13]. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *ICLR Workshop*, 2013.

- [14]. Tomas Mikolov, Wen-tau Yih and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of NAACL HLT, 2013.
- [15]. Tomas Mikolov, Geoffrey Zweig: Context dependent recurrent neural network language model. SLT 2012: 234-239
- [16]. J. T. Goodman. A bit of progress in language modeling, extended version. Technical report MSR-TR-2001-72, 2001.
- [17]. T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, S. Khudanpur, Extensions of recurrent neural network language model, In: Proceedings of ICASSP 2011.
- [18]. C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, and P. Koehn, One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. ;In Proceedings of CoRR. 2013.

Appendix

SYSTEMS MANUAL

This project has been written using python and its Theano library developed by a machine learning group at the University of Montreal .

The input data are expected to be in a simple ASCII text format, with a space between words and end of line character at end of each sentence. After specifying training data set, a vocabulary is first constructed. Note that if one wants to use limited vocabulary (for example for open-vocabulary experiments), the text data should be modified outside the toolkit, by first rewriting all words outside the vocabulary to <unk> or similar special token. After the vocabulary is learned, the training phase starts. Implicitly, it is expected that validation data are provided, to control number of training epochs and the learning rate. The model is saved after each completed epoch (or also after processing specified amount of words), and the training process can continue if interrupted.

The parameters for training an RNN model is:

address : Address of Data set

n_hidden : Number of node in hidden layer(150)

learning_rate : Learning rate for updating weights (0.01)

n_epochs= : Number of Epoch(10)

L1_reg= : L1 regularization(0.00)

L2_reg= :L1 regularization(0.00)

learning_rate_decay: Decreasing rate of learning_rate afer each epoch(0.8)

activation : actication function of hidden nodes('sigmoid')

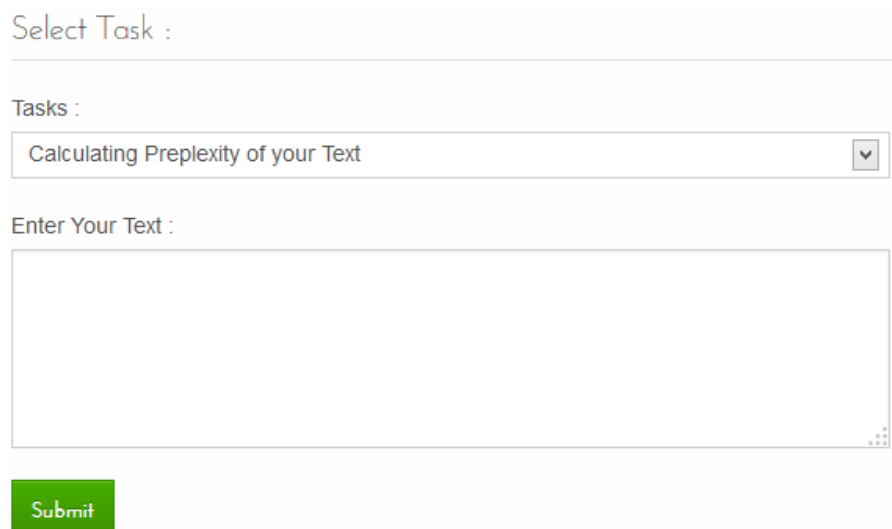
output_type= : actication function of output nodes('softmax')

LM USING RNN

Online Version

You can also use the online version of the project for testing the RNN with 150 neuron in the hidden layer in <http://www.cse.yorku.ca/~rsoltani/cse6412>.

You can get the perplexity of your text using the online version of this project. you can enter your text and find out the perplexity of that using RNN with 150 neuron in hidden layer like figure 9:



The screenshot shows a web interface for an online RNN project. At the top, there is a label "Select Task :" followed by a horizontal line. Below this, the label "Tasks :" is followed by a dropdown menu. The dropdown menu is currently open, showing the option "Calculating Preplexity of your Text" with a downward arrow icon on the right. Below the dropdown menu, there is a label "Enter Your Text :" followed by a large, empty text input area. At the bottom of the form, there is a green button labeled "Submit".

Figure 10

For predicting the next word of your sequence of word you can use the online version of this project. You can enter your word sequence and see the 3 most probable words that could be after your word's sequence.

LM USING RNN

Select Task :

Tasks :

Guessing the next word



Enter Your Word Sequence :

Submit