Language modeling using LSTM Neural Networks

Atchuta Venkata Vijay Chintalapudi
A53102693
achintal@eng.ucsd.edu

Sidharth Kodaikkal Vijayan
A53092651
skodaikk@eng.ucsd.edu

Rudra Pratap Singh
A53092651
rps001@eng.ucsd.edu

Sanmitra Ijeri
A53105054
sijeri@eng.ucsd.edu

Abstract

Recurrent neural networks have been widely used for language modeling in recent times. One of the prime reasons is their ability to remember the context for a longer distance than conventional n-gram models. But, due to the vanishing gradient problem, context of up to a maximum of 5-6 words is saved. This problem is overcome by using Long Short-Term Memory (LSTM) memory units that can store their state without loss for long periods of time steps. In this paper, we implemented a multi-layered LSTM network for language modeling on Penn Treebank (PTB) dataset using two frameworks, Keras and Tensorflow. This network achieves an accuracy of 20%, 40%, 61% for the best, 5-best, and 10-best predictions respectively of the target word given an input stream of words.

1 Introduction

A language model has the ability to predict the words in a text, given its context. Language modeling has profound applications in areas like auto completion, speech recognition, and machine translation.

Recurrent neural networks have been used to make use of sequential information for solving problems which require the network to have memory of its past and make the prediction based on its context. In theory, RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps. Additionally, recurrent neural networks (RNNs) are limited by their vanishing gradient and exploding gradient problems. To overcome this limitation, long short-term memory(LSTM) networks are used, which use a gating mechanism to store their states over any desired period of time.

In this paper, an LSTM neural network is trained to generate predictions of the next word based on knowledge from previous words in a sentence.

2 Network Architecture

RNNs are effective in sequential data modeling because they are able to persist information by using recurrent connections. In theory, RNNs are capable of storing arbitrarily long context. But, the storage capacity of RNNs is limited by the problem of vanishing or exploding gradient. This problem is overcome by LSTM using a ”forget gate” implementation and hence it works well to store context for arbitrarily long time intervals without letting the gradient vanish.

The basic LSTM cell used in the network is as shown below:

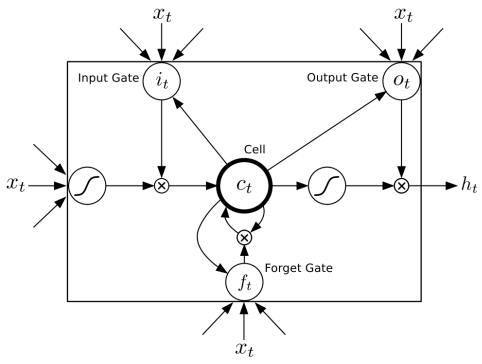

Figure 1: Long Short-term Memory cell

Let the input sequence be X = (x_1, x_2, $\cdots$, x_t) and the output sequence be Y = (y_1, y_2, $\cdots$, y_t). Let the input gate, forget gate, output gate and cell activation vectors at time t be given by i_t, f_t, o_t and c_t. The governing equations for a LSTM cell are given by:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.1}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.2}$$

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.3}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.4}$$

$$h_t = o_t tanh(c_t) \tag{2.5}$$

where W_{lm} denotes the weight for the connection between layer l and m layer, b_n denotes the bias at layer n, and σ denotes the logistic sigmoid function.

The network is built using 600 nodes in the LSTM layer which is fully connected to an output layer of 10,000 units. Each LSTM node is unwrapped in 18 time steps since the length of each sentence in the input is 18. An embedding layer is also added before the words are fed into the LSTM layer. This embedding layer embeds each word into a 180 dimensional vector. The overall network architecture looks as below:

3 Network Training

3.1 Data Acquisition

The training and validation data used on the network is the Penn Treebank (PTB) data-set [5]. The training data consists of 1 million words and the validation set consists of 80,000 words. In the data-set, all the punctuation marks have been removed and the numbers have been replaced by a generic character 'N'. This sequence of words is then split into sets of 18 words as suggested in the paper [1].

3.2 Implementation & Experimentation

The LSTM language model is trained using an 18 dimensional input layer(words in a sentence), 180 dimensional embedding layer (linear projection layer), a 600 dimensional hidden layer of LSTM, and a 10,000 dimensional output layer(vocabulary size).

Before training the network, the words are converted to the vectors of size 180 using the Word2vec technique [2]. The 'Collective Bag of Words (CBOW)' technique has been used in the processing

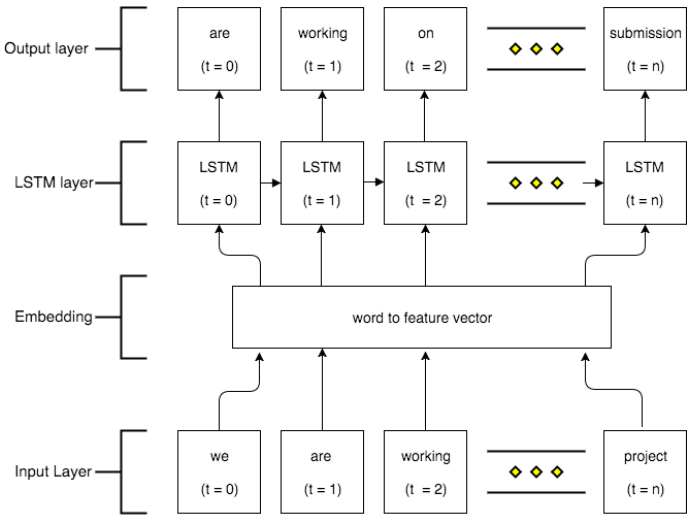

Figure 2: Network architecture

which conserves the relational information between words. Meaning, the cosine distance between the word vectors for the words 'she' and 'her' is smaller than that between the words 'she' and 'apple'. Also, vectors are relationally additive. Meaning, vector('queen') - vector('he') + vector('she') $\approx$ vector('king'). These Word2Vec vector outputs are used as the targets for the embedding layer. So, over time, the embedding layer is trained to output the fine-tuned output of the Word2Vec function.

The size of the output layer is the size of the vocabulary. Data sets consisting of a huge vocabulary have a drawback of high computational complexity as well as reduced accuracy due to the probability distribution spread out over a large output size, thereby reducing the ease of accurately determining the correct predicted word. In such cases, high accuracy can only be achieved by training the network on a very huge data set. To overcome that, we propose clustering of the vocabulary into a given number of classes which defines the size of the output layer. K-means algorithm was used to classify the vocabulary into 5000 clusters with varying size. This resulted in assembling relationally close words such as "her" and "she" into a single cluster since their vector representations are closer in distance. These words are relationally close but would generate different meanings when predicted in a sentence. Also frequent words such as "the" and "an" are classified into a single cluster which led the system to dominantly predict that cluster as the output across multiple inputs.

Hence, instead of using K-means over the entire vocabulary size, we have clustered the 4,000 least frequent words in the data set into 500 clusters while leaving the rest of the words untouched. This produced 6,500 clusters which is also the size of the output layer. We have observed that the accuracy has improved by 8-10% through clustering. But in our case, since the ptb data set used in the training contained an acceptable vocabulary size of 10,000, we haven't utilized clustering. However, we expect that this would have a much fruitful result in data sets having a huge vocabulary size where clustering would reduce computational time considerably and also improve accuracy.

The output of the embedding layer is passed on to a hidden layer of 600 LSTMs. The output of this layer is then connected to the final output layer, where the softmax activation function is used. It assigns probability to each word in the vocabulary on it being the next word in a sentence based on the past inputs and the hidden states of the hidden layer. In another experiment, 2-hidden layers of LSTMs are used by stacking them between embedding layer and the output layer.

tanh is used as the activation function in the network. The error function used in the network is cross-entropy error where the target of the softmax output is set as the sparse vector of the word that goes as input in the next time step (sparse vector of a word is the vector which has 1 in the $index^{th}$ position and zeros at other positions). This error is calculated at each time step and is back-propagated after 18 time steps to estimate the updates in weights. The initial weights are initialized by a Gaussian distribution scaled by the fan_in size + fan_out size.

To avoid over-fitting, we have experimented by using Bernoulli and Gaussian dropouts after the LSTM layer [6]. In Bernoulli dropout, the hidden activation h_i is is multiplied by the Bernoulli random variable r_b that takes value $1/p$ with probability p and 0 otherwise thereby dropping out some hidden activations. In Gaussian dropout, h_i is perturbed with a $r \sim \mathcal{N}(0, 1)$ to $h_i + h_i r$ which is analogous to multiplying h_i with r' where $r' \sim \mathcal{N}(1, 1)$ thereby not forwarding some hidden activations to the next layer. Experimentation by applying batch normalization is done where the activations of the previous layer at each batch are normalized, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

The Stochastic Gradient Descent optimizer is used which has the support for changing learning rates, momentum, decay, and using Nesterov momentum. Other optimizers such as RMSprop and Adapative Gradient descent are also experimented with to evaluate the network's performance.

To improve the network understanding of stops (sentence end), zeros padding is done on the sentences whose length is not a multiple of 18. In the embedding layer, this zeros padding is ignored and treated as a special character by using a flag "mask_zero".

To manage the memory on the server, a random subset of continuous 2000 sentences is picked from the test data. From this subset of size 2000 sentences, 1600 sentences are used for training, and 400 sentences are used for validation.

Test accuracy is calculated by comparing the predicted word with the target word. Also, the N-best prediction accuracy is calculated by checking if the target word is present in the N most probable list of words predicted by the network. The experiments are run for N=5 and N=10 cases.

The above described network is implemented using both Keras and Tensorflow frameworks.

4 Results

4.1 Uni-directional single layer LSTM network

Network Configuration:
Training set:
Due to the large size of the training dataset, the input training data is segmented and fed to the network segment by segment.
Total input size: 200,000 sentences segmented into groups of:
Input segment size: 2,000 sentences
No. of segments: 100

Testing set:
No. of sentences: 2,000 sentences

Network:
Output layer size: 10,000 words
Epoch size: 5
Batch size: 512
Size of LSTM layer: 600
LSTM activation: tanh
Dropout: 0.2
Optimizer: SGD with learning rate = 0.1, Decay = 10^{-6}, Nesterov's Momentum index = 0.9
Output activation: softmax
Loss function: categorical cross-entropy

Error %:

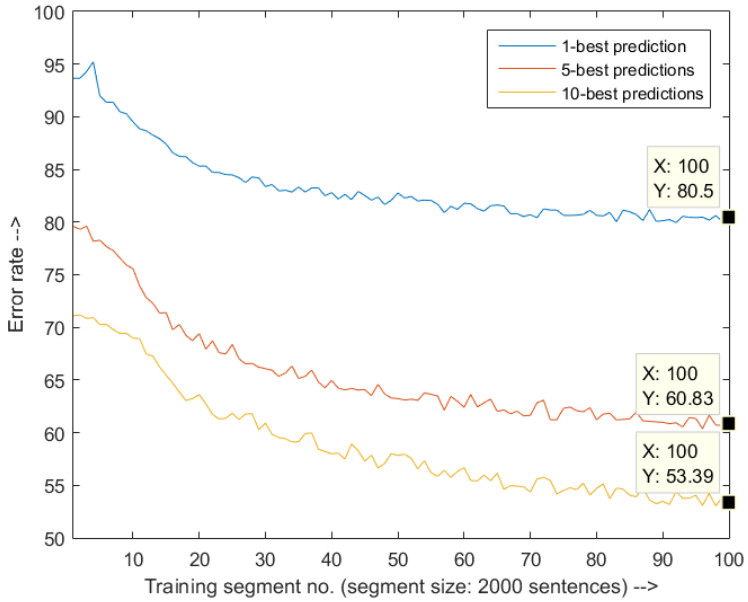

Figure 3: The graph shows the Error % vs the segment number for a uni-directional single layer LSTM. The error curves are calculated by comparing the best predicted word, 5-best predictions and 10-best predictions with the target word. The error % drops as the network is trained with more and more input data and after 100 segments each of 2000 sentences, falls to 80.5%, 60.8% and 53.4% with 10-best predictions error being the lowest as expected.

Output Examples:

Input Sentence	—-	$	N	a	share	for	the	first	time
Target Sentence	>>>>	N	a	share	for	the	first	time	since
Top 5 Predictions	++++	the	<unk>	new	price	<unk>	N	six	in
	++++	a	to	year	up	a	year	half	he
	++++	<unk>	a	$	for	N	company	time	since
	++++	N	billion	<unk>	of	the	first	quarter	of
	++++	an	million	share	in	$	$	nine	the

—-	since	june	N	ibm	closed	at	$	N	down	$
>>>>	june	N	ibm	closed	at	$	N	down	$	N
++++	early	he	when	will	up	a	an	million	the	an
++++	june	when	in	which	$	its	<unk>	a	a	a
++++	april	since	mr.	said	in	the	a	up	N	$
++++	N	N	N	's	the	N	N	billion	$	N
++++	the	the	the	has	at	$	$	down	from	<unk>

Table 1: The above example displays the input sentence of length 18 words given to the network (shown in yellow), the target sentence which is the input sentence shifted by one (shown in blue) and the top 5 predictions of the network, based on the words before it, in decreasing order of their possibilities. The green tile depicts that the target word is in the top 5 predictions and the orange tile marks the 'relationally' closest predicted word to the target word if the target word isn't in the top 5 predictions. The character 'N' is used to depict the number values in the corpus.

Input	—-	a	continued	improvement	in	our	u.s.	business	said	james
Target	>>>>	continued	improvement	in	our	u.s.	business	said	james	<unk>
Top 5	++++	lot	increase	by	<unk>	area	economy	in	he	e.
	++++	number	in	of	recent	market	and	to	that	a.
	++++	new	to	on	N	business	trade	is	mr.	j.
	++++	N	decline	in	a	<unk>	market	has	it	r.
	++++	<unk>	<unk>	to	the	fiscal	business	and	the	<unk>

—-	<unk>	tandem	's	chief	executive	officer	in	the	year-earlier
>>>>	tandem	's	chief	executive	officer	in	the	year-earlier	period
++++	president	and	a	of	<unk>	mr.	this	first	results
++++	the	inc.	N	economist	in	in	a	company	quarter
++++	<unk>	said	net	financial	and	of	<unk>	u.s.	period
++++	a	's	chief	operating	of	and	new	<unk>	fiscal
++++	an	corp.	<unk>	executive	officer	said	the	new	the

Table 2: This example shows a less accurate prediction for a given input sentence than the example in Table 1

The network achieves an accuracy of 40% with the 5-Best prediction model. It predicts a desired word from the knowledge of the previous words inputted to the network over 18 time steps. From observing the input, target and output sentences it can be stated that the network is doing a good work at predicting conjunctions, prepositions and context based words such as "business" in an statement on economics etc. The network isn't good at predicting proper nouns such as "James" which is expected since the network isn't trained extensively on sentences with proper nouns. Even if the target word isn't in the 5-Best predictions, the network does a good job at predicting words that are closely related to the target word.

Also the corpus currently used consists of sentences of word length 18 formed by clubbing

multiple shorter sentences. This partially deteriorates the structure of the individual statements and confuses the network. As a part of the future work, we plan to pad the individual sentences to length 18 instead of clubbing them with other shorter sentences thereby preserving their structure and therefore expect the network to perform better.

4.2 Uni-directional double layer LSTM network

Network Configuration:
Output layer size: 10000 words
Epoch size: 5
Batch size: 512
Size of lower LSTM layer: 900
Lower LSTM activation: tanh
Size of higher LSTM layer = 600
Higher LSTM activation: tanh
Dropout between LSTM layers: 0.5
Dropout between higher LSTM layer and output layer: 0.2
Optimizer: SGD with learning rate = 0.1, Decay = 10^{-6}, Nesterov's Momentum index = 0.9
Output activation: softmax
Loss function: categorical cross-entropy

Error %:

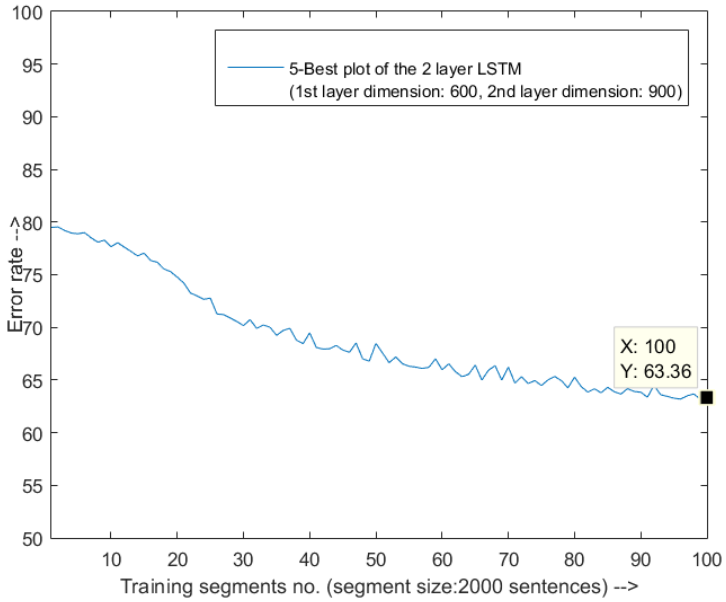

Figure 4: The graph shows the 5-Best prediction Error % vs the segment number for a uni-directional double layer LSTM. The error % after 100 segments each of 2000 sentences falls to 63.36%

It is observed that the error rate for 5-best predictions in a double layer uni-directional LSTM network (63%) is higher than that of a single layer uni-directional LSTM network (60%). This is because a double layer LSTM needs more epochs to train the network with a given input to attain the same error rate as that of a single layer LSTM network. It is observed that zero padding further decreases the error to 55%.

4.3 Bi-directional double layer LSTM network using Tensorflow Framework

Network Configuration:
Output layer size: 10000 words
Epoch: 10
Batch size: 20
Sentence size: 18 words
Size of LSTM layer in the forward direction: 300
Size of LSTM layer in the backward direction: 300
LSTM layer activation function:tanh
Optimizer: SGD with learning rate = 1.0, Decay = 0.1^{Epoch}
Output layer activation: softmax
Loss function: categorical cross-entropy

Error %:

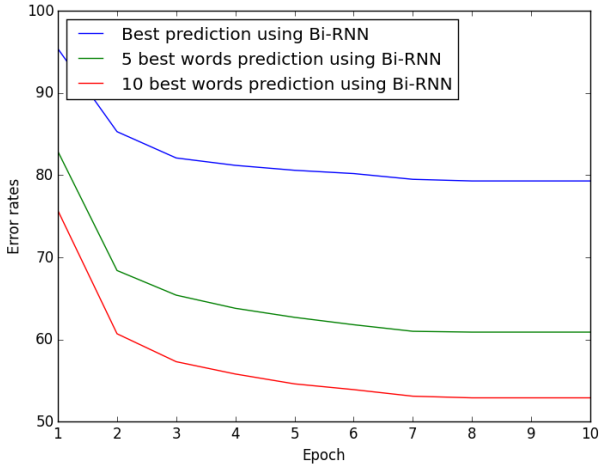

Figure 5: Error rate in Bidirectional LSTM Network

It is observed that the error rate is around 20% for best prediction and around 60% for best 5 word predictions and around 50 % for 10-best word prediction.

5 Conclusion

It is observed that LSTM based RNNs provide a good performance in modeling language models for word predictions given an input stream of words. The accuracy is greatly dependent on the size of the input dataset, sentence length and the training time. Increasing the length of the input sentence to a large number can be inefficient, since the earlier words in the sentence have a far less influence on the output compared to the more recent words.

It is also observed that the LSTM network model predicts verbs after a pronoun or a noun showing that the model is developing a sense of grammar as it is trained over sentences of multiple contexts.

Currently such language models are used in building deep learning neural nets which heavily benefit day to day applications such as in Google Search text prediction and Swiftkey word prediction in smartphone keyboards.

6 Individual Contributions

Vijay worked on parsing the data set, word2vec conversion, experimentation on reducing vocabulary size using k-means clustering and displaying best N predictions for the results.
Rudra worked on building the network model on Keras, experimenting with multiple parameters and layer sizes and training and testing the network. Rudra worked on producing the plots for the results.
Sidharth worked on implementing the embedding layer, reducing the vocabulary size by picking the most frequent words in the dataset, zero padding, N-best prediction ideation and the presentation.
Sanmitra worked on implementing the language modeling network on TensorFlow, training and testing the network and comparing the error rate with the output from Keras, building the webservice to get the next 15 likely words and UI for the demo.
Everyone worked equally in contributing and writing the content for the report.

7 Live Demo of Network Prediction

The trained network is saved along with the final weights in aws host. A python web-service is run using bottle framework in aws instance. Using this integrated framework, a UI is designed that predicts the next word given the previous words.

- Web Application hosted at: `http://jsfiddle.net/sanmitra/hrr1subo/41/show/`
- Video demo of the working model: `https://www.youtube.com/watch?v=J0gTNDh_1vY&feature=youtu.be`

8 Acknowledgments

We thank William Fedus and Prof. Gary Cottrell for helping us improve our understanding on using LSTM networks for language modeling.

References

[1] Ebru Arisoy, Abhinav Sethy, Bhuvana Ramabhadran and Stanley Chen. *Bidirectional recurrent neural network language models for automatic speech recognition*. IBM Watson Research Center, USA and Turkey, 2015.

[2] Word2Vec code posted by Google. `https://code.google.com/archive/p/word2vec/`

[3] Keras library for theano. `http://keras.io/`

[4] RNN implementation using Tensorflow . `https://www.tensorflow.org/versions/r0.7/tutorials/recurrent/index.html`

[5] PTB Dataset. `http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz`

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky,Ilya Sutskever and Ruslan Salakhutdinov . *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research 15 (2014).