# HW2: Language Modeling

Alex Lin
alexanderlin01@college.harvard.edu

Melissa Yu
melissayu@college.harvard.edu

February 14, 2018

## 1 Introduction

Language modeling is a very interesting problem that has experienced much development over the past few years. The central idea is to build a statistical model that can accurately estimate the distribution of natural language. For this problem set, we examined the issue of trying to predict the 20 most likely words for continuing a pre-specified sentence. We examined three main models for this problem: (1) a basic trigram model, (2) a neural network language model (NNLM), and (3) a long short term memory recurrent network model (LSTM). We tuned this models to try to optimize *perplexity*, our metric of choice that measures how well a probability distribution predicts a sample.

## 2 Problem Description

Let $\mathcal{V}$ be the vocabulary. We assume words are represented as one-hot encoded vectors of size $|\mathcal{V}| \times 1$. Given a sequence of words encoded as one-hot vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, the objective is to accurately predict the distribution of $\boldsymbol{x}_{n+1}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ for $n = 1, 2, \ldots, N$. Let $q$ be our model's predictive distribution. The objective is to adjust the model's parameters to minimize the *average loss*, as defined by

$$L = \frac{1}{N} \sum_{n=1}^{N} (\ln q(\boldsymbol{x}_{n+1}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)) \cdot x_{n+1}$$

where $\cdot$ denotes the inner dot product. The standard metric used for evaluation is called *perplexity*, as defined by

$$P = \exp L$$

## 3 Model and Algorithms

We trained the three different models specified in the instructions. These included (1) a basic trigram model, (2) a neural network language model (NNLM), and (3) a long short term memory recurrent network model (LSTM).

### 3.1 Trigram Model

### 3.2 Neural Network Language Model

### 3.3 Long Short-Term Memory Recurrent Neural Network Model

The Long Short-Term Memory Recurrent Neural Network Model (LSTM) was proposed as a way to cleanly integrate long-term memory into a recurrent neural network. Traditionally, regular recurrent neural nets have suffered from an inability to propagate information from past iterations of training, because they get inundated with more recent information. The LSTM has four interacting layers, as depicted below. This helps with deciding what information to propagate or forget from earlier in sentences. The key idea is that there is a cell state $\boldsymbol{C}_t$ that may or may not be modified significantly by operations applied to the hidden state propagated hidden state $\boldsymbol{h}_t$ and the current word $\boldsymbol{x}_t$.

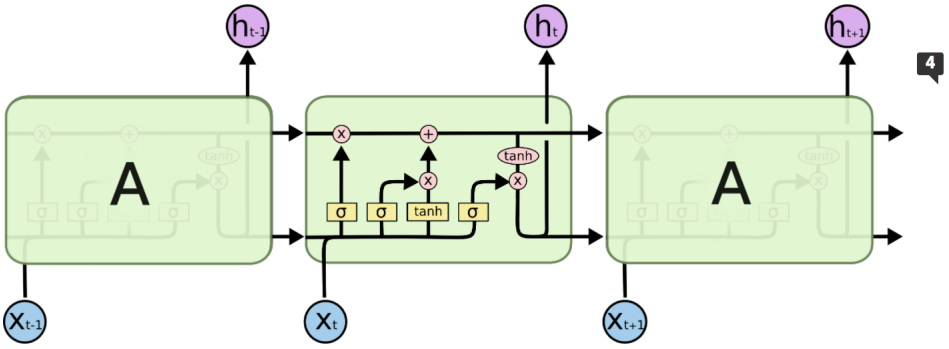


*Figure 1: An unrolled diagram of a single-layer LSTM.*

The mathematical layering of a single-layer LSTM follows:

$$
\begin{aligned}
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_f) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_i) \\
\tilde{\boldsymbol{C}}_t &= \tanh(\boldsymbol{W}_C \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_C) \\
\boldsymbol{C}_t &= \boldsymbol{f}_t * \boldsymbol{C}_{t-1} + \boldsymbol{i}_t * \tilde{\boldsymbol{C}} \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_o) \\
\boldsymbol{h}_t &= \boldsymbol{o}_t * \tanh(\boldsymbol{C}_t)
\end{aligned}
$$

where $*$ denotes the element-wise multiplication operator.

For the construction of the overall model to predict the distribution over the next word $\boldsymbol{x}_{t+1}$, we follow the architecture as described by Zaremba et. al in their "large regularized LSTM". We first encode each word as an embedding $\boldsymbol{x}_t$ of size $d \times 1$. Then, we pass $\boldsymbol{x}_t$ through a 2-layer LSTM with $l = 1500$ hidden units to get an output $\boldsymbol{o}_t$. For regularization, we then apply dropout with a rate of 0.65 to $\boldsymbol{o}_t$. Next, we apply a linear transformation to $\boldsymbol{o}_t$ and pass the result through a softmax function to obtain a probability distribution $\boldsymbol{q}_t$ over the next word:

$$\boldsymbol{q}_t = \text{softmax}(W\boldsymbol{o}_t + b)$$

where $W$ is a $|\mathcal{V}| \times l$ matrix and $b$ is a $|\mathcal{V}| \times 1$ vector. This $\boldsymbol{q}_t$ is then used to calculate the average loss. We adjust weights to reduce the average loss using different optimizers such as stochastic gradient descent and Adam. For final comparisons between models of this type, we report perplexity on the test set.

# 4 Experiments

We performed several experiments to tune the hyper-parameters of our models. In general, the LSTM seemed to perform the best. In-depth explanations of our tuning procedure and associated results can be found in this section. Note that all perplexities are test set perplexities.

| Model | Acc. |
|---|---|
| TRIGRAM MODEL | |
| NEURAL NETWORK LANGUAGE MODEL | |
| LONG SHORT-TERM MEMORY RNN | 139.98 |

*Table 1: Language perplexities of our models.*

## 4.1 Trigram Model

## 4.2 Neural Network Language Model

## 4.3 Long Short-Term Memory RNN

One item we varied in the LSTM RNN was the number of hidden states for the LSTM. Specifically, we looked at differences between the "medium-size" LSTM (650 states) and the "large-size" LSTM (1500 states) proposed by Zaremba et. al. We found that the large-size LSTM took longer to train, but obtained a significantly lower perplexity. However, the difference is large enough, so we decided to stick with large LSTM. Here are our best results on both models.

| | Medium LSTM | Large LSTM |
|---|---|---|
| Test Set Perplexity | 160.12 | 139.98 |

The second variable that we took into account was the optimization method. We looked at both SGD and Adam. We also varied the *starting* learning rate (STR) for these two methods. During training, we followed the results in the paper and decreased the learning rate by a factor of 1.2 for each iteration. Here are our best result under various settings.

| | SGD (STR = 1) | SGD (STR = 0.1) | SGD (STR = 0.01) |
|---|---|---|---|
| Test Set Perplexity | 203.27 | 156.35 | 183.24 |

| | Adam (STR = 0.01) | Adam (STR = 0.001) |
|---|---|---|
| Test Set Perplexity | 152.12 | 139.98 |

From this, we chose to use the Adam optimizer with a starting learning rate of $10^{-3}$.

The last variable we looked at was varying the word embedding dimension $d$. Surprisingly, this had quite a significant effect on the efficacy of the LSTM RNN. Here are our best results.

| | $d = 100$ | $d = 300$ | $d = 500$ | $d = 1000$ |
|---|---|---|---|---|
| Test Set Perplexity | 189.21 | 151.20 | 143.44 | 139.98 |

Thus, we decide to use an embedding size of 1000.

# 5 Conclusion