Neural Network Language Models for Automatic Speech Recognition

Aditya Kaushik, Eduardo Rosado, Thomas Spilsbury

December 10, 2018

Abstract

In this paper we compare Recurrent Neural Network (RNN) approximations of language models to traditional n-gram based models. We evaluate different recurrent model architectures, hyperparameter configurations, encoder and decoder configurations and regularization and intialization procedures while comparing their performance on various performance and natural language generation metrics. We find that in general, Long Short-Term Memory models with embedding layers at the encoder stage outperforms other models in terms of inference metrics and computational performance.

1 Introduction

One important problem faced by Automatic Speech Recognition (ASR) systems is transcribing utterances by speakers into intelligible written language. In general, this is a difficult problem to solve due to the varying length of utterances and ambiguity of classification between one utterance for another. The result is that the recognized transcript is likely to contain many words that are not intelligible, even if the transcribed words are the system's most likely direct transcription from sounds to syllables.

As an alternative, we can make use of prior information about the language to select words from a vocabulary given some observations of sounds or characters and the words that have been predicted beforehand. The language model, then, uses information about the sequence of prior predicted words to give a probability distribution for the next word, such that the ASR system would pick the next word based both on what are the most likely following words given a sequence as well as the utterance that is made by the speaker.

Prior to the introduction of RNNs, most language models used n-grams in Hidden Markov Model chains, where it was assumed that the kth word depended only on the prior n words. One key problem with this model is that the context window is of a fixed size, leading to a horizon problem. The Horizon problem occur because a if there were a word in the word sequence which is extremely predictive of the $(n+1)$th word away from the current word, that would would simply not be taken into account in that model. Worse still, the fixed size context window means that unnecessary weight is put on words that happen to be in the context window that may actually not be all that predictive were the context window infinite.

In contrast, RNN's seek to solve this problem by introducing a Neural Network architecture with recurrent connections, meaning that as the RNN makes predictions over a sequence observations, it takes into account a hidden layer produced by the previous observation (which in turn was a product of all the observations before that). In effect, the RNN architecture encodes the contextual history into the hidden layer vector allowing for an encoded approximation of infinite contextual history.

One of the first papers applying RNNs to Language Modeling was [**?**]. But since 2010, new RNN approaches have shown improvement on the state-of-the-art. We examine these new architectures and techniques for future work discussed in the Milkolov paper and analyze whether their results indicate that improvements can be made on the baseline set by the paper.

2 Data

The first dataset we use is the WikiText dataset. This is a smaller dataset than Gigaword and WSJ'92, but allowed us to iterate faster and run more experiments.

This dataset is based on articles from Wikipedia, containing 2,088,628 tokens in the training set, 245,569 tokens in the test set and 217,646 tokens in the validation set.

Samples from the Dataset include:

"As with previous ¡unk¿ Chronicles games , Valkyria Chronicles III is a . . . "

"At Nintendo CEO Hiroshi Yamauchi 's request , Game Boy creator Gunpei Yokoi 's Nintendo R D1 developed . . . "

"It was larger than the Scientific , at 73 by 155 by 34 millimetres ..."

This dataset contained quite a lot of technical or domain specific terminology due to the fact that the sentences within were derived from encyclopedic articles. This inherently makes accurate prediction difficult, since there is a higher likelihood that the following word might be infrequent in the corpus.

3 Metrics

Models are evaluated by taking a sequence of words from the validation set, removing k words from the end of the sentence and predicting the next k words from the existing $n - k$ word sequence.

3.1 Perplexity

Perplexity is a measure of "how generalized" the language model is. A higher perplexity indicates a greater uncertainty was encountered when predicting sequences of words. Formally, it is the inverse probability of a text sequence normalized by the number of tokens.

$$PP(W) = \sqrt{\prod \frac{1}{P(w|w_1, ..., w_{i-1}}}$$

Note that perplexity is an "intrinsic metric", it does not have anything to do with the quality of the predicted sentences, but instead is based on the certainty of the model itself.

3.2 ROUGE

In contrast, metrics such as ROUGE (which is an improvement on BLEU) measure the quality of a sentence in terms of their similarity to human generated language. These metrics are called "extrinsic metrics".

The most basic form of such extrinsic metric is "word error rate" (WER), defined as:

$$WER = \frac{S + D + I}{N}$$

, where N is the number of words and S, D, I are substitutions, deletions, insertions.

However, such a metric doesn't take int account the fact that a recognized sequence may differ in length to a reference sequence.

Improving on this situation, ROUGE measures based on the longest matching subsequences, overlapping pairs and n-gram co-occurrences.

3.3 METEOR

Another metric trying to improve BLEU is METEOR. The novel thing about this metric is the use of Recall as well as Precision to compute the score. It computes the harmonic mean of Recall and Precision over uni-grams, the weight for recall being higher than Precision, meaning that false negatives (having a low probability for the true word trying to predict) will have a big impact on the metric (the score will be lower as the amount of false negatives increase).

More formally, Precision over an uni-gram is computed as:

$$P = \frac{m}{w_t}$$

Where m is the number of uni-grams in the candidate translation that are also found in the reference translation, and w_t is the number of uni-grams in the candidate translation.

Recall is computed as:

$$R = \frac{m}{w_r}$$

Where m is as above, and w_r is the number of uni-grams in the reference translation. Precision and recall are combined using the harmonic mean in the following fashion, with recall weighted 9 times more than precision:

$$F_{mean} = \frac{10PR}{R + 9P}$$

3.4 Naive Accuracy

Accuracy is measured naively by comparing a validation set sentence to a predicted sentence and seeing how many words the model predicted correctly.

4 Models

4.1 RNN

A simple RNN (Elman network) as described in the Milkolov paper consists of a single linear layer and non-linear activation (in our case, the *tanh* function, $\frac{e^{2x}-1}{e^{2x}+1}$, with domain $[-\infty, \infty]$ and range $[-1, 1]$. The inputs to the network are a hidden state vector, h_t, where the size is a hyperparameter and the input encoding vector i_t. The output of the network is the updated hidden state vector h_{t+1}, which will will be passed as the hidden state vector when processing the i_{t+1}th word, and the output encoding vector o_t, which can be viewed as a discrete probability distribution for the predicted word at t by applying the softmax function $\frac{e^x}{\sum e_i^x}$.

The trainable parameters of the RNN are the weights of the linear layer which determine how information from the previous state of the hidden state vector and the incoming word vector encoding are encoded into both the updated hidden state vector and output word vector encoding.

The RNN is trained via "back-propagation through time" (BPTT), which is effectively the same as the back-propagation algorithm proposed for feed-forward neural networks, but applied to the recurrent structure of the RNN. In effect, BPTT involves the same application of the chain rule to recurrent function applications, and hidden layer states, but usually with some sort of threshold to prevent computational complexity scaling with the number of tokens the network has seen so far. In practice this means that the hidden layer encodes a context window *up to* to a certain length, but no longer than that length.

4.2 LSTM

The LSTM cell builds upon the basic RNN cell and adds some gates in order to control how information is stored or deleted. These gates are element-wise operations, meaning that every value of the hidden state have a dedicated gate deciding the fate of that value in the next time step. All the gates are computed taking on account the last hidden state and the new input and they are applied onto the last hidden state.

The first gate applied is a "forget" gate that regulates how much of the last hidden state is conserved in this time step. The second one is a "storing" gate that decides how much of the new input is encoded into the hidden state. The last gate is an "output" gate which decides which parts of the hidden state are taken on account to compute the output. This output will pass through a softmax layer to get the prediction for that time step.

Thanks to this procedure, hidden units can be ignored by the output gate and they can propagate forward in time without being multiplied by a weight matrix and applied an activation function at every time step, helping overcome the gradient vanishing problem.

The forgetting and storing gates help encoding only relevant information into the hidden state, improving long-time dependencies recall.

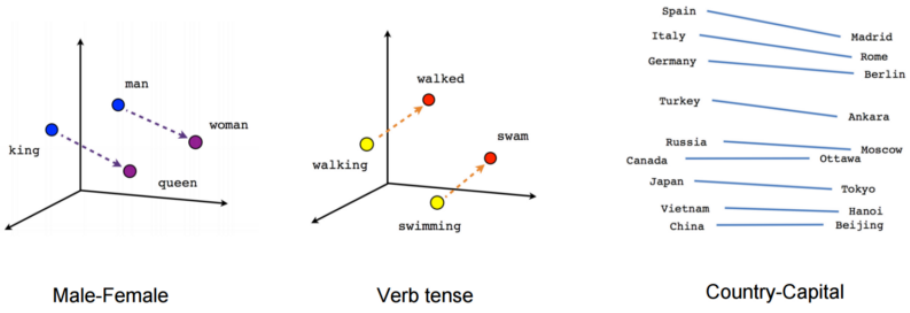
4.3 GRU

GRU is a simplified LSTM cell in which the forget and store gates are combined into one "update" gate with complementary gated values (they sum up to 1 for every hidden unit), this means that the hidden state will "forget" some information only if it is going to learn some new information.

5 Encoding and Decoding

5.1 Embeddings

Traditionally, the Language Models were feeded each word as a One-Hot encoded vector as large as the vocabulary is. Since vocabularies can be really large (20k to 800k for languages with many compounded words such as Finnish), this presents a computational problem, large matrices multiplication is expensive. Word Embeddings provide an alternative procedure in which the words to be feeded to the network are encoded in an n dimensional space. The embedding layer can learn a word encoding such that words related to each other semanticaly, get similar encoded vectors after the transformation. Even some basic arithmetic is possible. For example, if you subtract the vector for "queen" from the vector for "king" you get the vector for "man".



Male-Female Verb tense Country-Capital

References

[1] K. Grove-Rasmussen og Jesper Nygård, *Kvantefænomener i Nanosystemer*. Niels Bohr Institute & Nano-Science Center, Københavns Universitet