Application paper : LSTM based Accurate Word Prediction using Beam Search

Kavya Dayananda, Dr. Wencen Wu
San Jose State Univerisy
San Jose, CA, USA
{kavya.dayananda,wencen.wu}@sjsu.edu

***Abstract*— Modern word, speech processing and document translators use automated systems for their efficacy and ease of use. Deploying neural networks for such use cases require computation time and memory efficient algorithms; as these algorithms may commonly be deployed on mobile devices. Hence, this paper describes a methodology that uses beam search along with longest short-term memory model in neural networks to achieve this goal. The system described in this paper has multifold advantages of high accuracy than contemporary systems with low computation time and memory requirements. The work achieves 43% accuracy with few hours of train times.**

Keywords—Machine learning, algorithms.

I. INTRODUCTION

Machine learning algorithms are being used to translate and in general, process written scriptures and manuscripts apart from speech processing applications. Such systems use deep learning methodologies such as longest short-term memory based models to learn to predict and extract words from written or spoken templates.

Initial work on such systems were achieved for translation of smaller written documents and phrases, such as google translate. Due to availability of larger parallel computing devices in mainstream computing such as graphics processing units (GPUs) opened avenues for processing larger documents on a greater scale. However, due to the complexity of the data and algorithms, such systems may be insufficient for larger scale of word and speech processing without any changes to algorithms.

In this paper, we identify the key issue with one such algorithm, LSTM and solve it through a well-known search technique. The issue of finding the optimal word for predicting is solved by employing beam search technique. The beam search technique uses confidence intervals of various predicted words to prune the search space. It expands search on higher confidence words and finds optimal result in that space by probability-based divide and conquer method. This technique is proven to consume lower memory area and computation time as compared to several other techniques.

This paper is organized as follows: it initially presents the motivation of the work in section II, other related efforts in section III, the theory that the idea is based on in section IV, implementation in section V and lastly, experimentation and results in section VI.

II. MOTIVATION

The key difficulty in current day language processing and prediction is to produce and pick the optimal solution within a large solution space due to inherent complexities in grammar of a language. Also, training such a system using longest short term memory (LSTM) models tend to be more challenging due to the recursive nature of the model architecture. This in turn may result in solutions that are overfit or underfit and produce indeterministic , non-reproducible results on test and validation data sets.

We therefore want to solve this issue to produce multi-fold benefits. The solution is primarily expected to produce higher quality model fit with a lower training time and have a model that produces reproducible results. This in turn helps optimize the language processing downstream of the algorithm and help fine-tune data acquisition that is upstream which is the data input during the deployment. We envision this solution to drive the overall improvement of the system to be robust and efficient through optimal selection of words and parameters during its deployment.

III. RELATED WORK

Several models using various knowledge sources have been proposed for language modeling. [3] uses a maximum entropy model to combine a variety of information sources to improve language modeling on news text and speech. On the other hand, neural sequence models such as recurrent neural networks have been used for improved accuracy over the entropy models [5]. A variety of RNN regularization methods have been explored; [6] prevent overfitting of complex LSTM language models using drop out layers. Other efforts improved language modeling performance by modifying the RNN architecture to better handle increased recurrence depth [7]

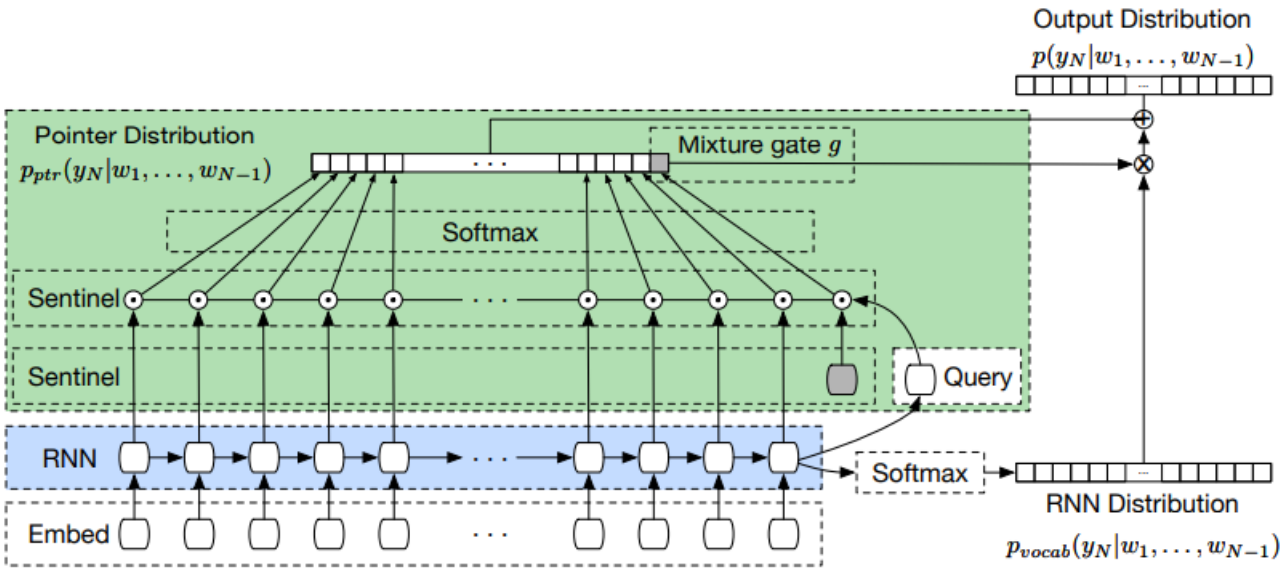

Figure 1 : Flow of the proposed algorithm

The use of pointer networks have been shown to help with code generation [9], summarization [8] and question answering [4]. While pointer networks improve performance on rare words and long-term dependencies they are unable to select words that do not exist in the input. [8] introduce a pointer softmax model that can generate output from either the vocabulary softmax of an RNN or the location softmax of the pointer network. Not only does this allow for producing words which are not in the input, the pointer softmax model is able to better deal with rare and unknown words than a model only featuring an RNN softmax. The pointer and RNN softmax are scaled according to the switching network and the word or location with the highest final attention score is selected for output. Although this approach uses both a pointer and RNN component, it does not combine the probabilities for a word if it occurs in both the pointer location softmax and the RNN vocabulary softmax. In our model the word probability is a mix of both the RNN and pointer components, allowing for better predictions when the context may be ambiguous.

[9] expounds on this concept further, and generates an output sequence conditioned on an arbitrary number of base models where each base model may have differing granularity. In their task of code generation, the output could be produced one character at a time using a standard softmax.Their model however requires a more complex training process involving the forward-backward algorithm for Semi-Markov models to prevent an exponential explosion in potential paths.

The most comparable result to the proposed technique is based on dictionary search for pattern recognition. They perform a hyperparameter search and try to classify the written documents using optimal hyperparameter set. They achieve up to 30% accuracy for a model trained with 86272-word dictionary [1]. We expound on the work done by [2] for our implementation.

IV. THEORY

The requirement of faster and higher quality convergence in LSTM based word predictors demands a faster word searching techniques. We hypothesize LSTM based speech algorithms tend to converge to optimal word selection with an area and time efficient searching algorithm suited to such a problem.The LSTM based models are known to have larger space requirements [10] that also puts constraints on the underlying hardware that may add additional complexity to the learning problem at hand. We therefore find beam search to be suit the requirement of the problem at hand.

Beam search is a greedy heuristic search algorithm that divides the search space on the high confidence nodes. It picks the higher confidence nodes and expands these limited nodes set where the search result may exist. It also has an optimizer built into it such that it picks the optimal set of high confidence nodes [11] during the search space pruning process. Alongside reducing the time to search, it also reduces the memory required to search. This is particularly important in most LSTMs as they generate large numbers of words with various confidence intervals and the beam search picks the optimal word with highest confidence with lowest memory and time requirements.

To further investigate this property, we expound on Kadlec et al [4] and Merity et al [2] and use probability mass to extract the confidence of each predicted word [4]. Probability mass assigned to a given word can be represented as the sum of the probability mass given to all token positions where the given word appears:

$$p(w) = \sum_{i \in I(w,x)} X_i \tag{1}$$

Where, $I(w,x)$ results in all positions of the word w in the input x and $p(w) \in R^V$. This methodology, also known as pointer sum addition methodology [4].

Depending on the length of documents, large numbers of $p(w)$ may be derived. To predict the next word, all these

$p(w)$ need to be evaluated and to pick the most probably predicted word which is a memory and time intensive task. Also, such task may not be feasible to implement if the document is large and hence the search space is hard to parse owing to its size. The beam search therefore predicts optimal words by dividing all the $p(w)$ by finding $Max[erfc(p(w)]$ by dividing the search space based on high confidence intervals. It can also be shown it converges to the highest confidence word using the fundamental theory of beam search. Due to the method of divide and conquer of beam search using probabilities, it can be shown that it also uses the lower memory and space for its computations.

V. Reduction of Theory to Implementation

We propose to use a model that combines a standard softmax with a pointer. Our model has two base distributions: the softmax vocabulary of the RNN output and the positional vocabulary of the pointer model as shown in figure 1. We refer to these as the RNN component and the pointer component respectively. As we only have two base distributions, g can produce a scalar in the range [0, 1]. A value of 0 implies that only the pointer 1 means only the softmax-RNN is used.

While the models could be entirely separate, we re-use many of the parameters for the softmax-RNN and pointer components. This sharing minimizes the total number of parameters in the model and capitalizes on the pointer network's supervision for the RNN component.

Lastly, we use a CPU based system for our implementation that runs tensor flow in the anaconda environment. We use the test and train data set from [2]. The experiments and results section provide greater details on the experiments done on the models.

VI. Experiments and Results

As the model uses the outputs of the RNN from up to L timesteps back, this presents a challenge for training. If we do not regenerate the stale historical outputs of the RNN when we update the gradients, backpropagation through these stale outputs may result in incorrect gradient updates. If we do regenerate all stale outputs of the RNN, the training process is far slower. As we can make no theoretical guarantees on the impact of stale outputs on gradient updates, we opt to regenerate the window of RNN outputs used by the pointer component after each gradient update. We also use truncated backpropagation through time in a different manner to many other RNN language models. Truncated backpropagation allows for practical time-efficient training of RNN models but has fundamental trade-offs that are rarely discussed. This truncated backpropagation is run for k2 timesteps every k1 timesteps, as seen in figure 1.

In our task, the pointer component always looks L timesteps into the past if L past timesteps are available. We select k1 = 1 and k2 = L such that for each timestep we perform backpropagation for L timesteps and advance one timestep at a time. Only the loss for the final predicted word is used for backpropagation through the window.

The current model and the data set used by [2] took 35 computing hours for train and achieved an accuracy of upto 43%. With this pretrained, the testing of various words takes approximately 120 seconds.

References

[1] http://www.fki.inf.unibe.ch/databases/iam-on-line-handwriting-database

[2] Merity, Stephen, et al. "Pointer sentinel mixture models." *arXiv preprint arXiv:1609.07843* (2016).

[3] Rosenfeld, Roni. A Maximum Entropy Approach to Adaptive Statistical Language Modeling. 1996.

[4] Kadlec, Rudolf, Schmid, Martin, Bajgar, Ondrej, and Kleindienst, Jan. Text Understanding with the Attention Sum Reader Network. arXiv preprint arXiv:1603.01547, 2016.

[5] Mikolov, Tomas, Karafiat, Martin, Burget, Luk ´ as, Cer- ´ nocky, Jan, and Khudanpur, Sanjeev. Recurrent neu- ´ ral network based language model. In INTERSPEECH, 2010.

[6] Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. arXiv preprint arXiv:1409.2329, 2014.

[7] Zilly, Julian Georg, Srivastava, Rupesh Kumar, Koutn´ık, Jan, and Schmidhuber, Jurgen. Recurrent Highway Net- ¨ works. arXiv preprint arXiv:1607.03474, 2016.

[8] Gulc¸ehre, C¸ aglar, Ahn, Sungjin, Nallapati, Ramesh, Zhou, ¨ Bowen, and Bengio, Yoshua. Pointing the Unknown Words. arXiv preprint arXiv:1603.08148, 2016

[9] Ling, Wang, Grefenstette, Edward, Hermann, Karl Moritz, Kocisky, Tom ´ as, Senior, Andrew, Wang, Fumin, and ´ Blunsom, Phil. Latent Predictor Networks for Code Generation. CoRR, abs/1603.06744, 2016.

[10] Greff, Klaus, et al. "LSTM: A search space odyssey." *IEEE transactions on neural networks and learning systems* 28.10 (2017): 2222-2232.

[11] Tillmann, Christoph, and Hermann Ney. "Word reordering and a dynamic programming beam search algorithm for statistical machine translation." *Computational linguistics* 29.1 (2003): 97-133.