Usage of Transformer Language Models in Speech Recognition

A Project Report

submitted by

AVASARALA SAI KRISHNA SATHVIK

in partial fulfilment of the requirements for the
award of the degree of

BACHELOR OF TECHNOLOGY &

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING INDIAN

INSTITUTE OF TECHNOLOGY, MADRAS.

June 2020

THESIS CERTIFICATE

This is to certify that the thesis titled **Usage of Transformer Language Models in Speech Recognition**, submitted by Avasarala Sai Krishna Sathvik, to the Indian Institute of Technology, Madras, for the award of the degree of Bachelor of Technology & Master of Technology, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. S. Umesh
Research Guide Dept. of EE
IIT Madras, 600036

Place: Chennai
Date: 30th June 2020

ACKNOWLEDGEMENTS

I am very grateful to my research guide Professor Umesh for his constant invaluable guidance and support throughout the project. Also, special thanks to Vishwas for constantly resolving my doubts and helping me in my thought process. Learned a great lesson from this project that I should always cross check my work and results before proceeding further as it may lead to undesirable situations.

ABSTRACT

Transformers have become popular in natural language processing. In this project we have explored the usage of language models for improving the performance of speech recognition models.

We have observed that while not only improving over the baseline results, they can be used as better initializations which result in faster convergence. And also the idea of transfer learning where the learning that has been employed in the training of one model can be used in another has been used.

Further work in this direction would be in the direction of pre trained encoder and decoder models with learnt initialisations to improve the speed and the training time.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Transformer encoder-decoder models [1] have become popular in natural language processing. The Transformer architecture allows to successfully train a deep stack of self-attention layers via residual connections [2] and layer normalization [3].

The positional encodings [1], typically based on sinusoidal functions, are used to provide the self-attention with the sequence order information.

Across various applications, systematic improvements have been reported over the standard, multi-layer long short-term memory (LSTM) recurrent neural network based models.

While originally designed as an encoder-decoder architecture in machine translation, the encoder (e.g., [5]) and the decoder (e.g., [4]) components are also separately used in corresponding problems depending on whether the problem disposes the whole sequence for prediction or not.

While similar analysis has been done in [6], they have only used RNN/LSTM language models, in this project we have exclusively used Transformer language models.

These are the different ways in which we use the power of these language models (which are trained on similar or huge amounts of text):

1. Shallow fusion [6]
2. Using their outputs as Embeddings
3. Look-ahead model [8]
4. Initialising the Decoder part of ASR model with Lang Model weights.

The positives and negatives of these approaches will be explored in further sections.

CHAPTER 2

TRANSFORMERS FOR LANGUAGE MODELLING

The language model we consider is based on the decoder component of the Transformer architecture [1].

A layer is defined as a stack of two components :

1. Self-attention
2. Feed-forward modules.

The autoregressive self-attention module in the "*l*"-th layer transforms the input $z_t^{(l-1)}$ at position *t* as follows:

$$
\begin{aligned}
x_t^{(l)} &= \text{LayerNorm}(z_t^{(l-1)}) \\
q_t^{(l)}, k_t^{(l)}, v_t^{(l)} &= Q x_t^{(l)}, K x_t^{(l)}, V x_t^{(l)} \\
h_t^{(l)} &= (h_{t-1}^{(l)}, (k_t^{(l)}, v_t^{(l)})) \\
y_t^{(l)} &= z_t^{(l-1)} + W_0 \,\text{SelfAttention}(h_t^{(l)}, q_t^{(l)})
\end{aligned}
$$

Where Q,K,V, respectively denote query, key, value projection matrices, LayerNorm denotes layer normalization [3],SelfAttention denotes the scaled multi-head dot product self-attention [1], and W0 denotes the projection matrix for the residual connection [2].

The output y^l_t is then fed to the feed-forward module:

$$
\begin{aligned}
m_t^{(l)} &= \text{LayerNorm}(y_t^{(l)}) \\
z_t^{(l)} &= y_t^{(l)} + W_2 \,\text{Activation}(W_1 m_t^{(l)})
\end{aligned}
$$

Where Activation Is Rectifier[7]. The final model is built by stacking these layers multiple times.The input of the network consists of the sum of the token em-bedding (word or BPE in this work) and the sinusoidal position encoding as specified in [1].

The output softmax layer gives the probability distribution for the next token. As shown in the equations above, $h^{(l)}_t$ can be seen as states of the Transformer model2(whose size, as opposed to the RNN states, linearly grows along the position dimension).

During inference, these states are stored to avoid redundant computation. During training, the computation along the position dimension is parallelized for speed-up.

One of the powerful language models we used during our experimentation was the GPT-2[4]. It has been trained on a preliminary version of WebText which contains slightly over 8 million documents for a total of 40 GB of text. It has about 50,256 unique BPE, as a show of its might.

CHAPTER 3

USING LANGUAGE MODELS IN SPEECH RECOGNITION

As mentioned in Section 1, we will discuss all the methods that have been employed to use language models for Speech Recognition one by one.

3.1 Shallow Fusion[6] :

The inference step of CTC/attention-based speech recognition is performed by output-label synchronous decoding with a beam search. During the decoding using beam search, rather than simply taking scores from the acoustic model we also consider scores from the LM.

In the beam search process, the decoder computes a score of each partial hypothesis, which is defined as the log probability of the hypothesized character sequence. The joint score $\alpha(g)$ of each partial hypothesis h is computed by :

$$\alpha(h) = \lambda\alpha_{\text{ctc}}(h) + (1-\lambda)\alpha_{\text{att}}(h) + \gamma\alpha_{\text{lm}}(h),$$

$$\alpha_{\text{lm}}(h) = \alpha_{\text{lm}}(g) + \log p_{\text{lm}}(c|g).$$

where g is an existing partial hypothesis, and c is a character label appended to g to generate h, i.e., $h = g \cdot c$. The score for h is obtained as the addition of the original score $\alpha att(g)$ and the conditional log probability given by the language model.

3.1.1 FineTuning Language models:

While for the language models we trained from scratch we do not need any finetuning, for huge language models like GPT- 2 [4], it has been shown that it is useful to finetune the model first.

Fine Tuning here refers to training the model for a few epochs on a smaller dataset with only altering the parameters of the final layers.

Rather than training the model from scratch, we initialise the parameters of the model with GPT-2's released parameters and start our training from there. When training on the new dataset, it only requires a few steps of training as the model already has a very good grip on the English language.

The model will understand the nuances of the text that should be generated so as to mimic the given dataset with only a few steps of training. This is why it is called Fine-tuning, we are fine-tuning the huge GPT-2 model to adhere to this small dataset and generate according to the patterns in the small dataset.

3.1.2 Shallow Fusion with BPE mismatch:

One issue we have faced when using huge language models is the mismatch in the BPE units (1000 : our model vs 50,256 (GPT-2)). We have dealt with this in two different ways:

1. Without Finetuning : As the BPE's of the bigger language model are generally a superset of the our model's, we can simply select those BPE's probabilities from the bigger model, normalize over them and give out a probability vector over those BPEs only.

2. Finetuning: Fine tuning the last layer removed and adding our own linear layer for prediction. This linear layer will have the final softmax with the number of units of the acoustic model we have trained.

3.1.3.Results for this method on the NPTEL dataset are as follows:

Word Error Rate (WER)

Models	**WER**
Baseline : 1000 BPE Transformer	13.9
Six layer Transformer LM	13.5
Without Fine tuning GPT-2	15.1 (worse than baseline)
Fine tuned GPT-2	**13.4**

3.1.4 Inferences:

While adding a language model improves the WER relative by around (7-8%) and absolute by 1-2%, one of reasons according to our experimentation for no significant jump while using GPT-2 would be not using the power of >50,000 BPE.

With or without fine tuning, asking it to predict over a very small subset of BPE(~1,000) when it has significantly more predictive power might be the cause for the weak performance. For example, there would be many bigger BPEs in GPT-2's units which are not used at all due to having a smaller set.

Consider “cat” and “caterpillar”, normally GPT-2 has both of them in its BPE set now when we restrict it to smaller units, we are losing the power of bigger units.

We look into the Look-Ahead method of combining, to mitigate this issue.

3.2 Using LM outputs as Embeddings:

In this method we have used the outputs of the language models given the context as the starting point for the acoustic model’s decoder. As language models are trained to predict the next token this would help the decoder to learn faster.

We have tried two approaches here :

1. Freezing the language model layers
2. Training the language model layers.

3.2.1 Results :

Word Error Rate (WER)

Model	**WER**
Freezing LM layers	14.1 (worse than baseline)
Training LM layers	13.8
GPT2 Fine-Tuned	**13.7**

3.2.2 Inferences:

One of the reasons that this did not work as we expected might be due to the decoder not learning the identity function. Initially as the decoder will be initialised randomly, the final output (even though there are residual connections) might be very different from the initial input given by the language model.

Future work in this would be in the direction of having less number of decoder layers (we used 6 till now it can be decreased to 3) and having a residual connection directly to the output of the decoder. This would help the model perform at least as good as the language model.

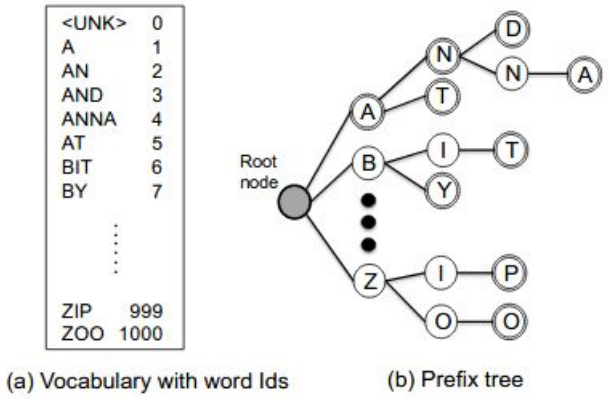
Also even in this case we aren't using the power of all the BPEs, the representation a smaller language model(with less BPE) learns for '**the'** can be a lot different from a bigger language model's '**the'** representation. Due to the availability of lesser BPEs, the smaller model has to learn about each of the BPEs and it still might not represent all the possibilities correctly, while this is not the case in the bigger BPE case.

Even though the units are sub-words(BPEs) we have observed that most of them are complete words (in the case of GPT-2), so given enough text data to learn, this model will have to learn representations for words, it will resort to BPE learning only when it encounters a word it hasn't seen. As BPEs are formed on the basis of frequent occurrences, the probability of seeing a word which is unknown is less.

3.3 Look-ahead model [8]:

The look-ahead word-based RNN-LM enables us to decode with only a word-based RNN-LM in addition to the encoder decoder. This model predicts next characters

using a look-ahead mechanism over the word probabilities given by the word-based LM.



(a) Vocabulary with word Ids (b) Prefix tree

To compute look-ahead probabilities efficiently, they used a prefix tree representation as shown in Figure above. This example shows a vocabulary and its prefix tree representation. During decoding, each hypothesis holds a link to a node, which indicates where the hypothesis is arriving in the tree.

Suppose a set of anticipated words at each node has already been obtained in advance. A look-ahead probability at node n can be computed as the sum of the word probabilities of all the anticipated words as :

$$p_{\mathrm{la}}(n|\psi) = \sum_{w \in \mathrm{wset}(n)} p_{\mathrm{wlm}}(w|\psi),$$

where *wset(n)* denotes the set of anticipated words at node *n*, and $p_{wlm}(w|\psi)$ is the original word probability given by the underlying word-based RNN-LM for word-level context ψ.

The character-based LM probability with the look-ahead mechanism is computed as:

$$p_{\mathrm{lm}}(c|g) = \begin{cases} p_{\mathrm{wlm}}(w_g|\psi_g)/p_{\mathrm{la}}(n_g|\psi_g) & \text{if } n_g \in F, c \in S \\ p_{\mathrm{la}}(n_{g \cdot c}|\psi_g)/p_{\mathrm{la}}(n_g|\psi_g) & \text{if } n_g \neq \text{null}, c \in \xi(n_g) \\ p_{\mathrm{wlm}}(\texttt{<UNK>}|\psi_g)\eta & \text{if } n_g \neq \text{null}, c \notin \xi(n_g) \\ 1 & \text{otherwise} \end{cases}$$

Where F denotes a set of word end nodes(the nodes which denote the end of the word), n_g is the node that g has arrived, $S = \{<space>, <eos>\}$, $n_{g \cdot c}$ is a succeeding node of n_g determined by c, $\xi(n_g)$ is a set of succeeding nodes from n_g , and η is a scaling factor for OOV word probabilities, which is a tunable parameter.

The first case of equation above gives the word probability at a word end node, where $p_{wlm}(w_g|\psi_g)$ needs to be normalized by $p_{la}(n_g|\psi_g)$ to cancel the already accumulated look-ahead probabilities. The second case computes the look-ahead probability when making a transition from node n_g to $n_{g \cdot c}$. The third case gives the OOV word probability, where character c is not accepted, which means the hypothesis is going to an OOV word. The last one handles the case that n_g is null, which means that the hypothesis is already out of the tree, and it returns 1 since the OOV probability was already applied in the third case. In the above procedure, we assume that whenever the hypothesis is extended by the <space> label, the new hypothesis points to the root node of the tree.

We have reproduced their results on the **WSJ** dataset, using a transformer acoustic model (basic units : characters) combined with a Word-RNNLM.

Transformer Character model (WER)

Model	**Eval93**	**Dev93**
Baseline char model	19.3	19
with Word-LM	**9.7**	**9.3**

As shown in the previous table, this idea has been very effective for many standard ASR tasks like WSJ and Librispeech reporting the state of the art results for end to end systems.

We have explored using GPT-2 to replace the Word-LM in this idea. Because GPT-2 has BPEs and not words as basic units, some changes are to be made to make use of the same idea.

1. BPEs are sub-words or parts of words, we cannot apply the first condition because end of a valid BPE does not mean that we will encounter '<space>' token next.
2. In the BPE method, we have tokens starting with the space token like this token.
   "**_the** "(' _' is used to denote '<space>') and without the space token like "**the**".
   So while building the prefix tree we have treated '<space>' as a separate character.

3.3.1 Future work :

As this is one of the works that has been proved to work for various datasets, we would want to extend this for any BPE to BPE kind of a matching so that we can combine the power of huge language models to help acoustic models while retaining all of their basic units.

3.4 Initialising the Decoder of ASR encoder decoder with Lang Model weights:

One more approach we have tried is that we tried to initialise the decoder of the acoustic model with the language model's weights as the decoder is essentially a Lang model getting some extra inputs (as cross attention) from the encoder.

Everything except the cross attention will already be present in the language model, so we can simply initialise the cross attention modules randomly.

3.4.1 Results:

Given below is the WER after doing the above experiment on NPTEL dataset.

Word Error Rate (WER)

Model	**WER**
Loading LM as decoder	14.2 (worse than baseline)

3.4.2. Future work :

I believe that this would be one of the approaches that should be further explored because it introduces the idea of transfer learning to encoder-decoder networks. Which in turn would enable faster and better training of the networks.

CHAPTER 4

CONCLUSION AND FUTURE WORK

The main aim of this project was to understand the State of the Art work happening in NLP and try to use those ideas in ASR. As shown before, Language models have shown that they can be useful, but their actual potential can be unlocked only when we use their power completely. Like in the section 3.3.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. NIPS, Long Beach, CA, USA, Dec. 2017, pp. 5998–6008.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in IEEE Conf. on Computer Vision and Patt. Recog. (CVPR), Las Vegas, NV, USA, June. 2016, pp. 770–778.

[3] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” arXiv preprint arXiv:1607.06450, 2016.

[4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multi task learners,”
[Online] : https://blog.openai.com/better-language-models/, 2019.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pretraining of deep bidirectional transformers for language understanding,” in Proc. NAACL, Minneapolis, USA, Jun. 2019.

[6] S. Toshniwal, A. Kannan, C.-C. Chiu, Y. Wu, T. N. Sainath, and K. Livescu, “A comparison of techniques for language model integration in encoder-decoder speech recognition,” in Proc. SLT, Athens, Greece, Dec. 2018.

[7] V. Nair and G. E. Hinton, “Rectified linear units improve restricted Boltzmann machines,” in Proc. Int. Conf. on Machine Learning (ICML), Haifa, Israel, Jun. 2010, pp. 807–814.

[8] Takaaki Hori, Jaejin Cho, Shinji Watanabe “END-TO-END SPEECH RECOGNITION WITH WORD-BASED RNN LANGUAGE MODELS”, Interspeech 2018.