LSTM REGULARIZATION FOR LANGUAGE MODELING

Christian Kolding-Sørensen & Lars Bryld

Supervisor: Alexander R. Johansen

***Index Terms*—** Language Modeling, Text Generation, LSTM, AWD-LSTM, Regularization

ABSTRACT

Language modeling is a core concept in natural language processing. Recurrent neural networks, such as RNNs, LSTMs and GRUs have proven very effective for language modeling. However, they have a tendency to overfit the data, and efficient regularization techniques are therefore necessary to achieve good results. We implement various regularization methods on an LSTM language model and compare our results to [Merity and N. S. Keskar, 2017]. We achieve slightly worse results on the Penn Treebank and Wikitext-2 datasets due to the fact that we use a different optimization algorithm. We investigate the effect of the regularization methods using ablation studies and find that DropConnect is by far the most important regularization for achiving good results. Finally, we investigate various methods of text generation using a model that has been trained on a dataset consisting of Seinfeld scripts.

1. INTRODUCTION

Following [A. Graves, 2013] we define a language model as a probability distribution over a sequence of tokens, where V is the size of the vocabulary. Each token in the vocabulary V is represented by a one-hot encoded vector $x \in {}^{\nu} = \nu$, corresponding to its index in the vocabulary. By using the chain rule the probability of a sequence of tokens $x_1...x_T$ can be expressed as:

$$p(x_1...x_T) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}...x_1) \tag{1}$$

This conditional probability is at the center of our analysis. We investigate the best approximation of this conditional probability using the LSTM model with various regularization techniques in order to predict the next token in the sentence.

Christian Kolding-Sørensen: christian.kolding@gmail.com
Lars Bryld: larsbryld@mail.com

We will analyze various methods for improving the performance of language models by preventing them from overfitting to the text it has been trained on. We will use the regularizations from [Merity and N. S. Keskar, 2017] as a basis for this. We will not implement the ASGD optimization algorithm nor the pointer model, however.

Since language models can be used to predict the next word in a sequence, it is a straightforward modification to have the language model generate text, that is, feed the predicted word back in and use that to predict another word, and so on. We will investigate various methods for generating meaningful text using a language model in the same style as the text the model has been trained on.

While language models are interesting in and of themselves, such as for next word prediction, they are also essential building blocks for almost all NLP tasks. Whether it is neural machine translation, question answering or chatbots, high quality language models are essential for achieving good results.

The code and datasets for replicating the results can be found in our GitHub code repository[1].

2. LSTM MODEL

The long short-term memory (LSTM) model is a popular neural network architecture for language modeling due to its ability to find and exploit long range context [A. Graves, 2013]. This feature stems from the LSTM model using purpose-built memory cells. These memory cells store past information, and thereby utilize long term dependency in contrast to standard RNNs. This is also referred to as overcoming the vanishing gradient problem in standard RNN's. The LSTM model is defined and illustrated below. Refer to the diagram for a graphical illustration of the LSTM memory cell structure.

$$i_t = \sigma(W^i x_t + U^i h_{t-1}) \tag{2}$$

$$f_t = \sigma(W^f x_t + U^f h_{t-1}) \tag{3}$$

$$o_t = \sigma(W^o x_t + U^o h_{t-1}) \tag{4}$$

$$\tilde{c_t} = tanh(W^c x_t + U^c h_{t-1}) \tag{5}$$

$$c_t = i_t \odot \tilde{c_t} + f_t \odot + \tilde{c}_{t-1} \tag{6}$$

[1] https://github.com/christiankolding/dtu-02456-deep-learning

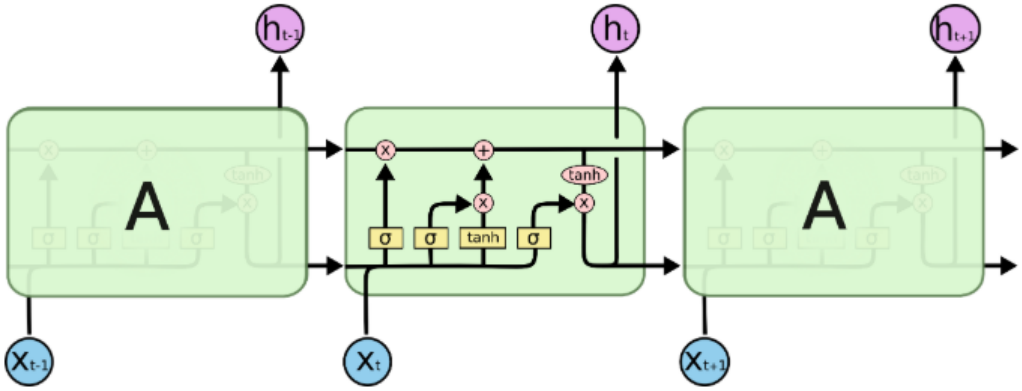

Fig. 1: Illustration of LSTM cell structure. Source: colah.github.io

$$h_t = o_t \odot tanh(c_t) \tag{7}$$

where the Ws and Us are weight matrices, x_t is the input, h_t is the hidden state, and c_t is the cell state. $\odot$ denotes element-wise as opposed to matrix multiplication. The intuition behind the LSTM model is that the cell state c_t which represents the current memory is explicitly controlled by the input and forget gates, which learn what new information should be remembered and what should be forgotten. The recurrent updated cell state together with the output gate determines the output and is responsible for carrying context across time.

3. REGULARIZATIONS

Given the very flexible architecture of the LSTM model, regularization is essential to avoid overfitting. The below regularization techniques are taken directly from [Merity and N. S. Keskar, 2017] and have been implemented to produce the final results in section 5.

3.1. DropConnect

DropConnect is applied on the recurrent hidden-to-hidden weight matrix where a random subset of weights are set to zero [L. Wan, 2013]. This means that the dropped weights remain dropped for the entire forward and backward pass. DropConnect can be seen as a generalization of standard dropout, where the activations are dropped.

3.2. Variational Dropout

Variational dropout [Y. Gal, 2015] samples a binary dropout mask the first time the dropout function is called. This mask is constant for the entire forward and backward pass. This is in contrast to a standard dropout where a new dropout mask is sampled at every time step, which makes it hard for the network to learn the underlying structure. Note that each mini-batch will use a newly sampled dropout mask, which ensures diversity in the elements being dropped.

3.3. Embedding Dropout

Instead of dropping random elements in the embedding matrix, we perform the dropout on a per-token basis, meaning that whole row representing that token is dropped. As the embedding matrix is used for a full forward and backward pass, this essentially means that all occurrences of a specific token are removed within that pass [Y. Gal, 2015].

3.4. Randomized BPTT sequence length

In order to use the dataset more efficiently the backpropagation through time sequence length is randomly selected in two steps.

- The base sequence length is set to seq with probability p and $seq/2$ with probability $(1 - p)$.
- The sequence length seq is selected according to $\mathcal{N}(seq, s)$.

Finally, the learning rate must be re-scaled according to the batch size sequence so that short sentences are not favored during training [P. Goyal, 2017].

3.5. Weight Tying

The motivation for weight tying is to reduce the total number of parameters in the model by sharing the weights between the embedding layer and the softmax layer. This method is motivated by [H. Inan, 2016] and prevents our model from having to learn a one-to-one correspondence between the input and output.

3.6. Independent Embedding and Hidden Size

The embedding size and the hidden size for the LSTM has been decoupled in order to allow for smaller word embeddings and generally increase the modeling possibilities. This is done by changing the word vector size in the first and last

layer in the LSTM such that they match the embedding size [Merity and N. S. Keskar, 2017].

3.7. Activation Regularization and Temporal Activation Regularization

Following [S. Merity, 2017] we apply activation regularization (AR) and temporal activation regularization (TAR) to the final LSTM layer.

Activation Regularization is implemented in order to penalize activations that are significant larger than zero [Merity and N. S. Keskar, 2017].

Temporal Activation Regularization is implemented in order to penalize the LSTM network for producing large changes in the hidden state [Merity and N. S. Keskar, 2017].

4. EXPERIMENT DETAILS

In order to benchmark our results, we train the model on the Penn Treebank and Wikitext-2 datasets. We train a 3-layer LSTM model, regularized as described, using 1150 hidden units in each layer and training for up to 60 epochs using ADAM with a weight decay of 0.0000012 and gradient clipping of 0.25. The low number of epochs compared to [Merity and N. S. Keskar, 2017] is due to the fact that our ADAM optimizer seems to converge much earlier. The learning rate is initially set to 0.001 and we anneal with a factor of 0.5 every time we reach a validation perplexity that is higher than the all-time best. We use a batch size of 40 for Penn Treebank and 80 for Wikitext-2.

The word embedding size is set to 400. The BPTT sequence length base size is 35 with a standard deviation of 5 and a minimum length of 5. The dropouts are set to 0.5 for DropConnect, 0.4 for input (0.65 on WikiText-2), 0.4 for output, 0.1 for embeddings and 0.3 for hidden.

For AR and TAR we have used $\alpha = 0.2$ and $\beta = 0.1$. The activation regulation parameters are set much lower than in [Merity and N. S. Keskar, 2017], since we found that the original values drastically inhibited learning.

We train our model on a Tesla K80 GPU. The Penn Treebank dataset takes roughly 160 seconds per epoch to train and the Wikitext-2 dataset, being larger and having a larger vocubalary, takes roughly 390 seconds per epoch to train.

5. EXPERIMENTAL ANALYSIS

The results of our model on Penn Treebank and Wikitext-2 is presented below and are compared to the benchmark model from [Merity and N. S. Keskar, 2017]. We refer to our model as WD-LSTM to account for the fact that we don't use ASGD optimization, and the model from [Merity and N. S. Keskar, 2017] is referred to as AWD-LSTM. Both validation and test perplexities for both datasets are higher for WD-LSTM than for AWD-LSTM. This is likely due to the much lower number of epochs, since our ADAM optimizer converges at an earlier point than the AWD-LSTM's optimizer.

Table 1: Penn Treebank benchmark

Model	Test	Validation	Epochs
AWD-LSTM	57.3	60.0	700
WD-LSTM	66.9	69.0	60

Table 2: Wikitext-2 benchmark

Model	Test	Validation	Epochs
AWD-LSTM	65.8	68.6	700
WD-LSTM	72.9	77.2	60

6. ABLATION STUDY

In order to investigate the impact of each regularization, we have performed an ablation study, where we remove each regularization technique individually and compare the perplexity of the new model with original model with all the regularizations in place. We have only done this on the Penn Treebank corpus.

DropConnect is clearly the most critical regularization as can be seen from the large increase in perplexity when removing it. Embedding dropout and AR/TAR are also important for achieving high perplexities. Surprisingly, removing weight decay achieves a slightly better validation perplexity, while using the full-size embeddings achieves slightly lower test and validation perplexity.

This is in sharp contrast to [Merity and N. S. Keskar, 2017] where all regularizations improve perplexity in the ablation study. Since the only real difference between the two models is the optimization algorithm, it is likely down to this.

Table 3: Penn Treebank ablation study

Model	Test	Validation
WD-LSTM	66.9	69.0
- DropConnect	84.9	83.1
- AR/TAR	74.7	71.6
- Embedding dropout	77.0	74.7
- Weight decay	71.0	68.0
- Var. seq. len.	67.8	70.6
- Full size emb.	66.7	68.8

7. TEXT GENERATION

An interesting experiment with the trained model is to use to generate a sequence of text. We try to different methods, multinomial sampling and beam search, as described below.

We abstain from using the greedy approach of simply choosing the most likely word at each step, since it leads to very monotonous sentences with very little variation.

Instead of using the benchmark corpora (Penn Treebank and Wikitext-2) for the text generation, we have instead collected a dataset of all episodes from the Seinfeld TV show and used this to train our model. In theory, the model will pick up patterns from those scripts and generate text in the same style.

7.1. Multinomial sampling

With multinomial sampling, we sample a token at each step from the a temperature-adjusted version of the probability distribution from the model. Given an original probability, p, the temperature adjusted probability $\tilde{p}$ using temperature t is given by:

$$\tilde{p} = e^{p/t} \tag{8}$$

So the higher the temperature, the flatter the probability distribution and vice versa. This means that picking a high temperature makes uncommon or unexpected words more likely, while a low temperature will make the common words more likely to appear.

Below is an example of generated text from our trained model using a temperature of 0.8. Based on this output, it is clear that the model has picked up the basic structure where each character is mentioned, followed by a colon and their line. It also alternates between the characters, such that none of them say two things in a row. Finally, all the sentences are roughly grammatical albeit slightly weird at times!

7.1.1. Multinomial sampling output

Elaine: I don't know.
Kramer: Oh you're getting a ride?
Jerry: No, I'm just trying to get in the back.
George: You know what I'm gonna do? That's a little bit. I think you'd try to call him.
Jerry: I'm not going.
George: So what do you want me to do. They're gonna keep it to my office.
Jerry: Oh, no.
George: Hey, you know what? I'm gonna be shooting on a yankee store with a box and slams the door bathroom
Elaine: Hey. George! Encourage me.
George: Not worried ?
Jerry: No!
George: Go back to the cable shop? How to you are that liar? You're forced out to leave
Jerry: Not hitting that's not the point for it.
Kramer: You're...

7.2. Beam Search

When selecting the next word greedily, we are not guaranteed to get the most likely sentence overall. Beam search is an attempt to solve this problem, where we consider b candidate sentences at every point in time, with b being the beam size. Below is an example of generated text from our model trained on the Seinfeld data where we use the beam search algorithm.

As it can be seen, the text starts out just fine, but after a few sentences it falls into a loop, where the same sequence is being repeated over and over again. This is a known shortcoming of the beam search algorithm, where the hidden state flushed at an `<eos>` token and the new sentence will always start with the most likely first word.

A potential improvement to the beam search algorithm would be to use a top-k approach so the most likely word is not chosen at every step, but rather just a likely word. That would probably improve the variation in the sentences and still avoid very strange words, which is the risk with samping in general.

7.2.1. Beam search output

George: Will you stop it?
Kramer: Yeah, I'll see you later.
Jerry: Hey.
Kramer: Hey.
Jerry: Hey.
Kramer: Hey.
Jerry: Hey.
Kramer: Hey.
Jerry: Hey.

8. LATEST DEVELOPMENTS

LSTMs have recently become less popular with transformer models achieving state-of-the-art results within language modeling [Dai et al., 2019]. In [Merity, 2019], however, the AWD-LSTM was recently picked up again and modified with a single head of attention to achieve state-of-the-art performance.

Shortly after the publication of [Merity and N. S. Keskar, 2017], a better approach for the continuous cache pointer, called dynamic evaluation, was shown in [Krause et al., 2017] which achieved far lower perplexities. The attention head in the SHA-RNN from [Merity, 2019] can also be seen as a pointer-like mechanism, so the dynamic evaluation approach is not needed here.

9. CONCLUSION

In this work, we have implemented all the regularizations for the AWD-LSTM. Compared to the benchmark results from

[Merity and N. S. Keskar, 2017] we achieve higher perplexities with a much lower number of epochs required for training across both Penn Treebank and Wikitext-2. We perform ablation studies and find particularly DropConnect on the recurrent connections to be of critical importance for achieving good results by preventing overfitting, followed by embedding dropout and AR/TAR which were also very important. For embedding size and weight decay, our results differed from [Merity and N. S. Keskar, 2017] in that we find removing them would improve the final results. Finally, since our results fall short of the benchmark, we can conclude that all parts of the AWD-LSTM are necessary to achieve what was at the time state-of-the-art results.

We use a model trained on Seinfeld scripts to generate text using temperature-adjusted multinomial sampling and beam search. From the output, it is clear that the model has learned the underlying structure in the data it was trained on. We find that multinomial sampling yields much more diverse and interesting texts, but more work is needed in order to generate truly realistic texts.

10. REFERENCES

[A. Graves, 2013] A. Graves, A-r. Mohamed, G. H. (2013). Speech recognition with deep recurrent neural networks. *arXiv:1302.5778V1*.

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context.

[H. Inan, 2016] H. Inan, K. Khosravi, R. S. (2016). Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv:1611.01462v3*.

[Krause et al., 2017] Krause, B., Kahembwe, E., Murray, I., and Renals, S. (2017). Dynamic evaluation of neural sequence models.

[L. Wan, 2013] L. Wan, M. Zeiler, S. Z. Y. L. R. F. (2013). Regularization of neural networks using dropconnect.

[Merity, 2019] Merity, S. (2019). Single headed attention rnn: Stop thinking with your head. *arXiv1911.11423v2*.

[Merity and N. S. Keskar, 2017] Merity, S. and N. S. Keskar, R. S. (2017). Regularizing and optimizing lstm language models. *arXiv1708.02782v1*.

[P. Goyal, 2017] P. Goyal, P. Dollâr, R. G. P. N. L. W. A. K. A. T. Y. J. K. H. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv:1706.02677*.

[S. Merity, 2017] S. Merity, B. McCann, R. S. (2017). Revisiting activation regularization for language rnns. *arXiv:1708.01009*.

[Y. Gal, 2015] Y. Gal, Z. G. (2015). A theoretically grounded application of dropout in recurrent neural networks. *arXiv:1512.05287*.