Regularizing and Optimizing the LSTM Language Model

Christian Kolding-Sørensen and Lars Bryld, Supervisor: Alexander R. Johansen

DTU Compute · Technical University of Denmark · University of Copenhagen · Kgs. Lyngby, Denmark Copenhagen N, Denmark

Language Model

Following *Graves et. al.* [1] we define a language model that is a probability distribution over a sequence of words, where V is the size of the vocabulary. Each word in the vocabulary V is represented by a one-hot encoding vector $x \in \mathbf{R}^{\nu} = \nu$, corresponding to its index in the vocabulary. By using the chain rule the probability of a sequence of words $x_1...x_T$ 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 analyses. We investigate the best approximation of this conditional probability using the LSTM model with various regularizing techniques, in order to predict the next token in a sentence. The goal is to generate coherent and meaningful sentences based on our trained LSTM model.

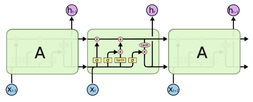
Model Specification

LSTM Model

Within language modeling the LSTM is proven superior in finding and exploiting long range context [2]. 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 gradiant problem in standard RNN's. The LSTM model is defined and illustrated below:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{3}$$
$$c_t = f_t c_{t+1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{4}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{5}$$
$$h_t = o_t tanh(c_t) \tag{6}$$

The intuition behind the LSTM model is that the cell state c_t can be updated with less or more information which is regulated by gates. These gates are the input i_t and forget gate f_t. The recurrent updated cell state determines the output and is responsible for carrying context across time.



Regularization and Optimization

Given the very flexible architecture of the LSTM model, regularisation is essential to avoid over fitting. At this stage we have implemented the below regularization techniques.

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 *Inan el. al.*[3] and prevents our model from having to learn a one-to-one correspondence between the input and output.

Randomized BPTT

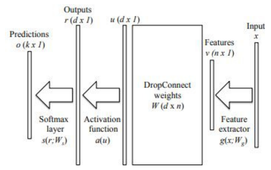
In order to use the data set more efficiently the batch size length for the forward and backward pass is randomly selected in two steps.

- 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 $\mathcal{N}(seq, s)$, seq being base sequence length and s a standard diviation.

Finally the learning rate must be re-scaled according to the batch size sequence so that short sentences are not favored during training. We have trained our model using $seq = 35$, $p = 0.95$, $s = 5$ and a minimum sequence length of 5.

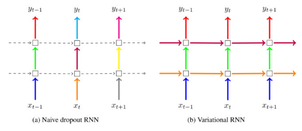
DropConnect

DropConnect is applied on the recurrent hidden-to-hidden weight matrix where a random subset of weights are set to zero [4]. This means that the dropped weights remain dropped for the entire forward and backward pass. For the remaining dropout operations in the LSTM network we apply variational dropout.



Variational DropOut

The variational dropout samples a binary dropout mask once the dropout function is called. This dropout mask is used for all inputs and outputs within a given forward and backward pass. This ensures that that eg. input x_0 at time step $t = 0$ has the same dropout mask as x_1 at time step $t = 1$ [4]. Note that each mini-batch will use a newly sampled dropout mask, which ensures diversity in the elements being dropped.



Embedding DropOut

Embedding dropout is performing a dropout on the embedding matrix at a word level. As the embedding matrix is used for a full forward and backward loop, this essentially means that all occurrences of a specific word is removed within that loop. The remaining words, after the dropout mask is applied, are scale by $1/(1-p_e)$, where p_e is the probability of embedding dropout.

Independent Embedding and Hidden Size

In order to reduce the word vector size the first and last layer (3^{rd}) in the LSTM model are modified such that the input and output dimensionality are equal to the reduced embedding size [1].

Activation Regularization and Temporal Activation Regularization

Following *Merity et. al.*[5] AR and TAR is only applied to the output of the last layer (3^{rd}) .

- Activation Regularization is implemented in order to penalize activations that are significant larger than zero [1].
- Temporal Activation Regularization is implemented in order to penalize the LSTM network for producing large changes in the hidden state [1].

Beam Search

Beam search is an alternative to choosing the most likely word at any given time. When the text is being generated a search tree structure is evaluating the next most likely word to generate. The beam search generates all successors of a given word t_n sorting them in an decreasing order. However only a given number of words are saved in memory, determined by the beam width $b = 100$ in our experiment.

Experiments

We train a 3-layer LSTM model, regularized as described, using 1150 hidden units in each layer and training for up to 100 epochs. The word embedding size is set to 400 and we have chosen the optimizer to be ADAM and a gradient clipping with maximum norm of 0.25. To evaluate the loss function we use the cross entropy loss function. The learning rate is set to 0.004.

Training and Validation Result

From the results table below, it can be seen that our perplexity is still significantly higher than the one from [1]. This is probably due to errors in our implementation.

Network	Test perplexity	Validation perplexity	Epochs
AWD-LSTM 3-Layer (tied) [1]	57.3	60.0	700
LSTM 3 Layer (Regularized as above)	85.5	88.6	62

Text Generation

The following text has been generated after training the model on Seinfeld scripts. Since beam search generate very repetitive outputs, we ended up using multinomial sampling. The temperature has been set to 0.8. Punctuation, spacing and capitalization has been reversed to ease legibility.

- Elaine: What? Alright do you mind?
- Jerry: I can't go to the red room from another rid of it.
- George: I was just on the other day a week.
- Jerry: What?
- Jerry: I don't think so.
- George: Oh god, I'm serious. I'm sorry.
- George: I don't know.
- Jerry: Alright, I'll see you for this. Looks at George Oh, you don't even want to do it, but you can't show me to stand out of my place with all one of those seriously I'll do. In fact, let's just take the instruments.
- Jerry: Louder yeah, yeah, just a month, it's a good shake in the screwgie.
- Jerry: You know what that's what ? I'll take the world all day now.
- Jerry: You need a good chance, I ...

References

[1] S. Merity and R. Socher N. S. Keskar. Regularizing and optimizing lstm language models. *arXiv1708.02782v1*, Aug 2007.

[2] G. Hinton A. Graves, A-r. Mohamed. Speech recognition with deep recurrent neural networks. *arXiv:1302.5778V1*, Mar 2013.

[3] R. Socher H. Inan, K. Khosravi. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv:1611.01462v3*, Mar 2007.

[4] Z. Ghahramani Y. Gal. A theoretically grounded application of dropout in recurrent neural networks. *University of Cambridge*, 2015.

[5] R. Socher S. Marity, B. McCann. Revisiting activation regularization for language rnns. *arXiv:1708.01009*, 2017.