Comparison of different Language Models

Simone Barattin (mat. 223828)

University of Trento

simone.barattin@studenti.unitn.it

1. Introduction

For this project, we were required to develop a neural language model, i.e. a neural network able to process varying length inputs. In particular, we show how a specific type of sequence model, called LSTM, is able to tackle the above-mentioned task and also how regularization techniques are fundamental to allow a deep model to learn from the data minimizing the risk of overfitting.

We follow the work of Merity et al. [1] and develop an LSTM with several regularization techniques. Such a model is defined as AWD-LSTM. We then show how this performs against a vanilla LSTM and an implementation of an attention-based LSTM similar to the Bahdanau one [2], and a Gated CNN [3]. The performances are evaluated on the Penn Treebank dataset, and we show that the regularization allows the model to reach a good perplexity value for the reference paper model.

2. Task Formalisation

We define *Language Model* a model with the ability to assign to a word its probability of being the next word of a sequence, given such a sequence. It is one of the foundations of NLP and is a relevant line of research for several applications, such as speech recognition, sentence completion (e.g. the suggestions given by our phone when writing a text), and also in machine translation. The task of language modeling is defined as the process of predicting which is the most likely word, contained in a vocabulary, given the tokens coming before it. This is defined as *word-level* language modeling, while a *character-level* setting uses the single characters composing words to perform the prediction. To be more specific, given a token x_t, we want to compute the probability of its plausibility given the tokens x_{t-1}. The likelihood of the word sequences can then be modeled by the chain rule as

$$P(X) = P(x_1)P(x_2|x_1)...P(x_n|x_1^{n-1}) = \prod_{i=0}^{n} P(x_i|x_1^{i-1})$$

This formalization of the problem however has a critical flaw: it needs to keep track of the whole previous conditioning sequence, which becomes increasingly difficult. The task then has been tackled in many different ways, starting from *n-gram* models, where *n* stands for the number of tokens that condition the probability of a particular word, along with several smoothing algorithms, e.g. Kneser Ney. After that, neural language models have been introduced into the language modeling scene, e.g. RNN, LSTM, GRU, obtaining even better performances thanks to their ability to learn distributed representations. Nowadays, state-of-the-art performances are obtained by transformers, due to the self-attention mechanism which allows them to retain even better context from the text corpora.

The task of this project is then to implement a neural language model, i.e. LSTM in this case, and train it in order to be able to assign correct probabilities to sequences of words.

	Train set	**Validation set**	**Test set**
Num. Sentences	42068	3370	3761
Num. Tokens	929589	73760	82430
Max Sentence Length	83	75	78
Average Sentence Length	22	21	21
Num. Unique words	10000	6022	6049

Table 1: *Sentence and token-level statistics*

Word	**Frequency**	**Relative frequency (%)**	**Rank**	**Zipf frequency**
the	50770	5.46	1/1	50770.00
<unk>	45020	4.84	1/2	22510.00
<eos>	42068	4.53	1/3	14022.67
N	32481	3.49	1/4	8120.25
of	24400	2.62	1/5	4880.00

Table 2: *Frequency analysis of the 5 most frequent words in the train set*

3. Data Description & Analysis

The dataset used to train and evaluate the model is the **Penn Treebank**, a dataset collected from the University of Pennsylvania which contains a collection of Wall Street Journal articles and processed afterward specifically for the task of word-level and character-level language modeling. This means that no data pre-processing is required, as the tokens are already in lowercase, there is no punctuation, special tags (i.e. $\langle unk\rangle$ for rare words and $\langle N\rangle$ for numerical data) are already applied, and it is already split into train, validation and test sets. The only modification we make is adding the end of sentence special token $\langle eos\rangle$ at the end of each sentence.

Considering the three mentioned tags, the vocabulary created from the PTB dataset has a length of 10000 words, making it relatively small with respect to other language modeling datasets. To have a better understanding of the data, we check some low-level statistics of the dataset, such as the number of sentences, the maximum and average length of such, the number of tokens, and the number of unique words. These data are reported in Table 1. We can notice that the number of unique words is equal to the size of the vocabulary, so, as further investigation shows that there are no OOV words, we can safely assume that the model will not encounter any unknown word during its evaluation, hence the vocabulary can be built using only the train set without any additional unknown tag mapping. Another important data is the maximum and average sequence length. We will see later that this information will be useful when shaping the data (more details will follow in Section 4).

A more thorough analysis is reported in Table 2. In particular, we report the frequency of the words, which will come in handy later in Section 5. For this, we analyze which are the most frequent words and the corresponding frequency, along with the relative frequency (frequency of the word in the whole data split). We report also the Zipf frequency, which defines that the frequency of any word is inversely proportional to its

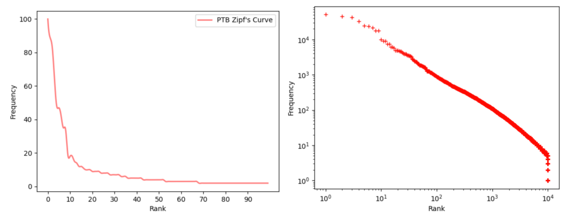

Figure 1: *Zipf (a) frequency law, and (b) power law*

frequency rank, thus a small number of events occur with higher frequency and vice-versa. This behavior can be observed in Figure 1.

4. Model

As reported in Section 2, the goal of the project is to define a neural language model performing language modeling. In this section, we will report the models implemented, along with the architectural decisions taken for each.

Please note that for the implementation of this project, the tokenization process was performed with a simple python script, given the nature of the dataset, and also that no pre-trained word embeddings were used. Instead of using the method *pad_packed_sequence*, we set a fixed sequence length and pass the data to the models with size [seq_len, batch_size]. We argue that this allows us to obtain better performances, as the above-mentioned method adds padding to match the longest sentence, and since the average sentence length is below the maximum (reported in Table 1) it could cause inefficiencies.

4.1. Vanilla LSTM

We implemented as a baseline a simple vanilla LSTM. The pipeline used (like for the other LSTM-based models) uses an Embedding layer followed by an LSTM layer as encoder, and a Linear layer as decoder to obtain the logits over the vocabulary.

4.2. AWD LSTM

The AWD LSTM [1] model uses the LSTM architecture and is characterized by the application of the DropConnect regularization and the use of ASGD, along with other several regularization techniques. Such additional regularizations include *gradient clipping*, which avoids the problem of vanishing gradient or exploding gradient, *tye weighting*, where the embedding and softmax weights are shared, *variational dropout*, applied to the embeddings, hidden states, and output tensors, *embedding dropout*, which applies a dropout mask directly to the embedding matrix, and *variable BPTT*, allowing the model to cover the whole data points. The code implemented has its codebase taken from [4]. The hyperparameters set are the ones provided, except for the dropout values, tuned after different tries.

DropConnect is a generalization of the normal dropout. It still introduces sparsity in the model, however, it drops the model's weights instead of its activations. Here it is applied on the hidden-to-hidden weight matrices, allowing the model to avoid overfitting on the recurrent connections.

ASGD, or Averaged SGD, is used instead of other popular optimization algorithms. It works in the same way as SGD, with the difference that the result of a step averages the optimization steps taken before allowing it to speed up the optimization process. Given the difficulty of setting the hyperparameter T, we implement a non-monotonically triggered ASGD, meaning that as soon as the SGD reaches a state of local minima, the optimizer starts averaging the gradient contribution. This behavior then could possibly allow the optimization process to escape the plateau and reach a better state.

4.3. Attention LSTM

In order to have a comparison with the method proposed in 4.2, we implement an attention mechanism on an LSTM model. The attention mechanism allows the model to learn to give weights to tokens composing the input sequence so that it is able to focus on the most relevant given information and obtain a better prediction.

The **attention mechanism** implemented is a simple mechanism similar to the one reported by Bahdanau et al. [2]: first, a Linear layer is used to extract the *alphas*, or attention weights, given the output of the LSTM, these weights are then multiplied to the LSTM's output and the result is summed over each time-step before obtaining the predictions over the vocabulary words.

The use of **teacher forcing** [5] is another experiment made while training this model. This is a training technique in which with a certain probability we use either the ground truth or the model predictions as the truth value used as part of the input sequence. This allows better signal propagation since the error adds up and pushes the model to learn to assign better predictions, helping the model to learn the mapping from input to output more efficiently.

4.4. Gated CNN

To have a final comparison, we implement a neural language model different from the ones using sequence models, as it is possible in fact to use CNNs to work with sequential data. In particular, we implement a model called Gated CNN, introduced by Dauphin et al. [3]. The model is implemented as multiple layers of GLU blocks, followed by an *AdaptiveLogSoftmax*, which speeds up and reduces the complexity of the final Linear layer. The implementation of this model followed [6].

These kinds of networks exploit the sparse information they can gather through their kernels and employ a hierarchical computation on the sequences in input. The strong point of these kinds of models is their ability to parallelize the computations, allowing faster training. However, naively applying the convolution operation would imply getting information regarding also the following tokens, hence each sentence needs to be 0-padded to the left for $k-1$ times, with k kernel size.

The **GLU block**, depicted in Figure 2, is composed of two convolutions, A and B, where $W, V \in \mathbb{R}^{k \times m \times n}$ (with m, n, k input, output and patch size) are the CNN kernels and $b, c \in \mathbb{R}^n$ are learned parameters. Once A and B are computed, the first is passed through a sigmoid operation σ and multiplied via matrix multiplication with the second. This operation simulates the gating operation of the LSTM cell. To further regularize the model, recurrent connections are added to each GLU block, meaning that at the end of each block, a skip connection is added.

5. Evaluation

In this section there will be reported the metrics used, the results obtained by the models implemented, along with the interpretation of the results obtained.

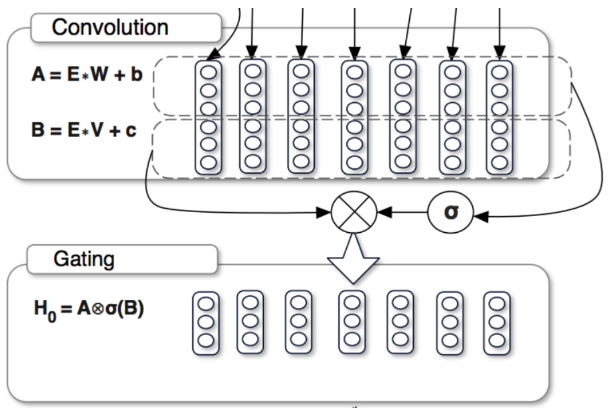

Figure 2: *Gated Linear Unit block*

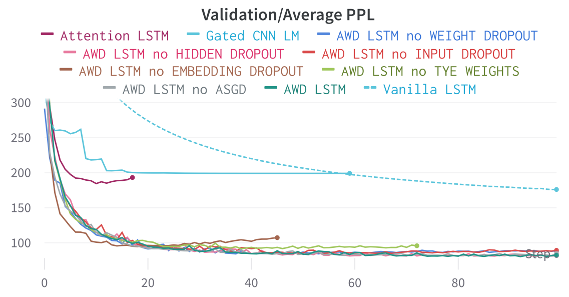

Figure 3: *Perplexity on the validation set*

5.1. Metrics

To evaluate the performances of a language model the metric used in the literature is perplexity. This metric defines how uncertain the model is in assigning probability values to the following word. Starting from the *Cross-Entropy*, perplexity can be computed as the exponential of the cross-entropy value of the model on the test data, i.e. given a sequence W, and its cross-entropy $H(W)$ the perplexity will be computed as

$$PP(W) = 2^{H(W)}$$

It's easy to imagine the relationship between the two: given the entropy of a distribution, we want it to have a small value, i.e. we want the probability distribution of the next word to be "peaky", meaning that the model is sure in the prediction. Minimizing the entropy then entails the minimization of the perplexity of a model.

5.2. Validation and test results

To evaluate the models on the task, we train them for 100 epochs, each using their own hyperparameters, and applying the early stopping technique to avoid unnecessary training time for overfitting models. Results for both the validation and the test set are reported in Figure 3, 4. Also, Table 3 reports the numerical results of the models on the test set. Please notice that some of the graphs did not reach the end of the training due to early stopping activating.

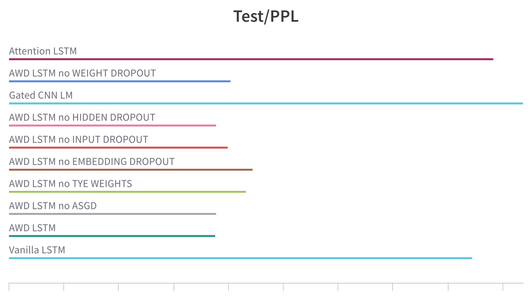

Figure 4: *Perplexity on the validation set*

5.2.1. *Vanilla LSTM*

As stated in Section 4.1, the baseline set for the experiments was the vanilla LSTM. As we can see from the graph in Figure 3, the model is able to achieve only a perplexity of 176 on the validation set. The learning curve in Figure 3 shows that the model still had some margin of improvement, however, to maintain a fair comparison setting, we limit the number of epochs to 100 either way, as the model would have reached the plateau at a perplexity of ~ 160.

5.2.2. *AWD LSTM*

For the AWD LSTM model, we run several experiments, performing an ablation study on the different regularization techniques used in order to understand which ones are more important than the others. Please note that no ablation study on the clipping gradient technique is reported in the Figures, the reason is that without it the loss exploded returning only infinite values. As shown in Figures 3, 4, in most of the experiments the models behave almost in the same way, meaning that each of the regularizations applied is giving its own small contribution. The original model employing all the regularizations achieves the best performances, meaning that all the techniques as a whole work in preventing the model to overfit. Apparently, the use of the NT-ASGD does not impact much the learning process, as we can see that an ablation on the use of this optimizer leads to a perplexity lower than the original by only 0.3 points, meaning that the optimization process follows the path taken by the plain SGD. Such poor behavior is likely dictated by the non-monotonically triggered setting imposed on it, as by changing the optimizer we lose all the gradient history information, making the averaging initially useless.

The same applies to the ablated model without the dropout applied to the hidden-to-hidden output. From this, we can infer that dropping information between the recurrent connections is not helpful, as the recurrent nature of the output already introduces diversity. While instead, the ablation of the dropout modules on the input to the model and the output leads to results coherent with the performances of the original model, meaning that their contribution is indeed significant even if in a small part and they are able to generate diversity.

Regarding the results obtained from the model without the use of dropout on the embedding matrix and the use of weight tying, it is clear that the use of these modules is fundamental, as the ablated models overfit way before the others. From such a result we can deduce the relation between the two regularizations. Since weight tying is defined as the process of sharing weights between the embedding layer and the final linear layer, we can deduce that, since dropping the embedding weights' matrix reduces the number of parameters to learn, without the embedding dropout but with tied weights, the number of learnable parameters increases compared to the previous scenario, and vice-versa without weight tying but with embedding dropout, the number of learnable parameters is linear with respect to the output dimension.

5.2.3. *Attention LSTM*

As reported in Section 4.3, the additional model implemented as a comparison to the baseline and the reference paper is an LSTM with an attention mechanism. We use a variation of the Bahdanau attention based on the results obtained by other implementations. In particular, several tests were required before deciding on the final model to use. Typically one would imagine that longer sequences would allow the attention to pick better decisions, which would lead to better performances, but instead, after several tries, the best sequence length found was 5 (the shortest used among all the methods implemented).

The first implementation tried used a *Multi-Head attention*. The model using 2 heads was able to quickly learn in the first epochs, showing a steep learning curve, however, given the large number of learnable parameters and the limited size of the dataset it was quick to overfit, ending with a best perplexity of 450. Decreasing the number of heads to 1 also did not help, as the model overfitted later with a perplexity of 250, a value obtained also after trying to apply the NT-ASGD (strengthening the claim made above 5.2.2). It was obvious then that the problem was that the model was too large for the available data.

Knowing this, the second implementation used the same model with a GRU, given that such a model has a lower number of weights compared to the LSTM. Again the model was overfitting in the same way. After that, the number of parameters was lowered by implementing a lighter attention mechanism on top of an LSTM. This decision led to the best performance so far, with a value of $\sim$ 190. To further increase the model's performances the Teacher Forcing [5] training procedure was implemented. Thanks to it the model was able to obtain its best perplexity of 184 on the validation set and 176 on the test set (the values reported in Figures 3, 4 and Table 3). As we can see the model is still not able to reach the baseline, however, we argue that this is still related to the limited data availability, as the training curve at the point of early stopping was still decreasing (i.e. once again the model overfitted)

It is worth mentioning another experiment done involving this model. As the data follows Zipf's law, there is an obvious data imbalance, which is usually dealt with using smoothing algorithms. The test involved performing a weighted cross-entropy, where the error signal is multiplied by the frequency over the number of classes (in our case the words) to give larger penalties to the majority classes. The results obtained showed an absurd decrease in perplexity, reaching a level of 10, which of course is not possible. As a matter of fact, trying to generate text with this pre-trained model had the opposite of the results wanted, since the sequence obtained contained only $'the'$ tokens (the majority class reported in Table 2).

5.2.4. *Gated CNN*

Finally, the CNN-based model was also evaluated against the previously mentioned models. The model was trained in the same setting as the others, with the difference that the optimizer used was Adadelta and a learning rate scheduler was used. We can notice from the results reported that this is the worst-performing model compared to the others. Here, we argue that the problem behind the poor performances is the small dataset used, as even the original paper [3] reported the same issue on the PTB dataset. Even after using a bottleneck layer for each block (which reduces the number of parameters) the problem could not be avoided.

Usually, in computer vision, CNNs have the ability to grow in depth by using smaller kernel sizes, allowing them to decrease the number of learnable parameters. However, when dealing with sequences, the model cannot rely on small filter sizes, as the captured content depends on them. For this implementation, the model was able to reach its best performance with a kernel size of 2, but given the small contextual window (following the claim just made) the model is not able to generate at all good sentences.

	Loss	**Perplexity**
Vanilla LSTM (baseline)	5.130	168.94
AWD LSTM no TYE WEIGHTS	4.458	86.35
AWD LSTM no EMBEDDING DROPOUT	4.487	88.83
AWD LSTM no INPUT DROPOUT	4.378	79.71
AWD LSTM no HIDDEN DROPOUT	4.323	75.42
AWD LSTM no WEIGHT DROPOUT	4.391	80.76
AWD LSTM no ASGD	4.326	75.68
AWD LSTM [1]	**4.322**	**75.31**
Attention LSTM [2]	5.175	176.71
Gated CNN [3]	5.233	187.42

Table 3: *Loss and perplexity results on the test set*

5.3. Inference evaluation

As a final evaluation, we can check the ability of the models to generate sequences of text. To do this, we use different values of temperature, i.e. a value to distribute probability assignments across the classes. The generated sentences, with the corresponding temperature value, are the following:
Vanilla LSTM [1.0]: *'the value of slipped infringed at a hit communist even deposit younger proxy alleges subsidies.'*
Attention LSTM [0.8]: *'the value of their projects at the table and even for the existing bid.'*
AWD LSTM [1.0]: *'the same people that is given therapy from no other doctor.'*
We can see that all the reported models are able to generate text that is somewhat meaningful. It is clear that the best sentences generated come from the AWD LSTM since the sequence is more meaningful than the others. One thing that we can notice is that even if the attention model has a higher perplexity than the vanilla one, is still able to generate better text sequences, hence the ability of a model should not stop at the numbers only.

6. Conclusion

In this project, we have seen how important regularization and how present the problem of overfitting is in NLP and in particular for the task of language modeling. We proved that the AWD LSTM model is able to achieve good results on the Penn Treebank dataset, even though it is still far from the SOTA results obtained by transformer-based models. Unfortunately, due to the limited data, we could not display the full potential of the two additional models, which still obtained relatively good results.

7. References

[1] S. Merity, N. S. Keskar, and R. Socher, “Regularizing and optimizing lstm language models,” 2017.

[2] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” 2014. [Online]. Available: https://arxiv.org/abs/1409.0473

[3] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language modeling with gated convolutional networks,” 2016. [Online]. Available: https://arxiv.org/abs/1612.08083

[4] “Lstm and qrnn language model toolkit,” https://github.com/salesforce/awd-lstm-lm/tree/1d466ec5875675ca6b6f0caacbb741240a29da9e.

[5] A. M. Lamb, A. G. ALIAS PARTH GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, “Professor forcing: A new algorithm for training recurrent networks,” in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/16026d60ff9b54410b3435b403afd226-Paper.pdf

[6] “Gated cnn,” https://github.com/DavidWBressler/GCNN.