Final NLU project

Noemi Canovi (230487)

University of Trento

noemi.canovi@studenti.unitn.it

1. Introduction

In this document, the report of the project for the course of Natural Language Understanding is presented. The main goal was to implement and test the performance of a Language Model using one of the RNN architectures. Using as a baseline a simple RNN model, a little improvement can be seen by developing an LSTM and a large improvement is reached with regularization techniques.

2. Task Formalisation

Natural language' statistical models are exploited in many systems, such as automatic speech recognition (ASR), machine translation (MT), and spelling correction. Language models provide a way to assign a probability to a sentence and to predict a word based on words preceding it. Several approaches can be used to construct them, the simplest being n-grams, but in the recent years AI techniques prove to outperform most of the competition and show steady improvements for the future.

Indeed, recurrent neural networks (RNNs), such as long short-term memory networks (LSTMs), have been found useful in many sequence learning tasks, including language model. Neurons that receive input from recurrent connections can be assumed to represent short term memory, and do not limit the length of the preceding words. From data, the model learns how to represent memory and exploit it to predict novel words. While shallow feedforward neural networks cluster similar words, recurrent neural network can also perform clustering of similar histories. However, these systems often suffer from overfit so to better lead the learning process, regularization is needed.

In this document, we inquire a Vanilla and an LSTM models, as well as some regularization techniques.

3. Data Description and Analysis

The Penn Treebank (PTB) is a huge corpus of over 4.5 million words of American English, created and maintained by the University of Pennsylvania. It is one of the most known and used corpus for the evaluation of models for sequence labelling, character level and word-level language modelling.

In our case, the corpus consists in 49199 sentences distributed in three *.txt* files (train, valid and test). The number of sentences per file are 42068, 3370 and 3761 respectively. Composed by clean and non-annotated words, the dataset was preprocessed to limit the vocabulary size to 10 000 elements, so all the remaining rare words are substituted with the tag "unk".

To properly exploit large datasets as input of neural networks, datasets are divided in batches which are processed one at the time. Given that batches collects elements of the same size, and our corpus is composed by sentences of various length, we need to preprocess the data. In particular, padding is used to add padding elements on the sentences until they have as many words as the longest one. Then, two dictionaries were constructed to map each word to an integer and vice versa. This is needed to transform words in tensor elements. Note that the vocabulary was saved in a *.txt* file: having a fixed vocabulary is required to properly save and load neural networks models. If not, the indices of the words will differ and we are not able to evaluate the net at all.

Our task consists of implementing a word-wise level language model so we would like to evaluate our model every time a novel word is presented into the sentence. This means that the customize dataset is composed by two main parts: *words*, i.e. the input we feed into the network, and *target*, i.e. the ground-truth words. The two are the same, but differ by a step, so that when we feed the first word into the network, the objective is to predict the successive word, i.e. the target. For instance, below we can see the *words* and *target* of the training sentence *"all came from cray research"*.

Input	all	came	from	cray
Target	came	from	cray	research

To do so, the *DataLoader* class was used, exploiting *collate_fn* to shape the batches.

4. Model

As a baseline, a Vanilla RNN was used. Composed by an embedding layer, a multi-layer Elman RNN and a final linear layer. By maintaining the same structure but changing the Elman RNN multi-layer with an LSTM one, an LSTM is constructed. The main problem of these networks is that they don't converge at all or they fast overfit, having a high performance on the train set but horrible in valid and test set.

In order to improve this, two types of regularization techniques were applied: domain specific, i.e. those techniques implemented and mainly used in Language Model, and generic, i.e. those techniques used to regularize neural networks and RNN.

4.1. Domain specific regularization techniques

Weight drop proved to be useful to improve the performance of the network. In DropConnect, the dropout operation is applied to the weight matrices within the LSTM, before the forward and backward pass. The same weights are reused over multiple timesteps, so in the forward and backward pass the same individual remain dropped. This help to prevent overfit.

As optimizer Adam, SGD and average SGD (ASGD) were investigated. SGD should outperform in terms of final loss and rate of convergence Adam [0]. ASGD is an improved version of SGD in which the average of the iterates past a certain threshold are used instead of the last iterates. However, when tried on the model, both SGD and ASGD seems to fail to converge, and Adam outperform both of them. Multiple trials were done but in the end Adam was exploited.

4.2. Domain specific regularization techniques

The learning rate is an important parameter of the network and changing its value depending on how the training is going can benefit the model. In particular, it seems that reducing the learning rate once the net stagnates is useful. Here a scheduler decreases the learning rate in a dynamic way when a plateau on the valid loss is met for a number of successive epochs.

It may happen, like in our case, that during the first phases of the training the net exhibit improvements in both the train and valid set, but after some epochs the performance on the valid test starts decreasing, showing signs of overfit. One simple way to avoid this is to use an early stopping approach: in my case, I decided to let the net run a fixed number of epochs, and save the net with maximum performance on the valid set.

Finally, weight decay was added to the Adam optimizer to, again, prevent overfitting.

5. Evaluation

5.1. Perplexity

To measure the quality of a model we can evaluate its performance on the test set, unseen data drawn from the same source as the training data. In practice, if a model correctly predicts the test set sentences, i.e. it assigns higher probability on the correct words, it is a good model.

As a measure of model quality, perplexity (PP) is used:

$$PP_\theta(w_{1:n}) = P(w_{1:n})^{\frac{1}{n}}$$

An alternative way of viewing perplexity is in terms of entropy, in which the value of the exponent is the cross-entropy of our current model with respect to the true distribution:

$$PP(w_{1:n}) = 2^{H(w_{1:n})} = 2^{-\frac{1}{n}\sum_1^n log_2 m(w_n)}$$

Note that we want to minimize the perplexity to maximize the test set probability.

The values of perplexity reached in both training and test sets by the models can be seen in the table below.

	PP train	PP test
Vanilla RNN	130	217
LSTM	77	171
LSTMreg	52	104

Both Vanilla RNN and base LSTM fast overtfit on the train data and end up having a value of perplexity on the test that is not good. For the values of table the training was stopped in the early phases in order to have a decent balance between the perplexity on train and test set. With more epochs the values on the train set can reach a value of perplexity of 40, accompanied by a far worse performance of 300 and over on the test set. Finally, with correct finetuning and regularization techniques, the value of perplexity was reduced to 104.

In Fig.1 and Fig.2 we can see the trends on the loss and the perplexity of the model. It is clear that in the first few epochs the model improved considerably, then the loss decrease slowly until the model starts to overfit on the training data and it fails to generalize to unknown data. Loss and perplexity on the train decrease, but on the valid and test they increase or reach a plateau.

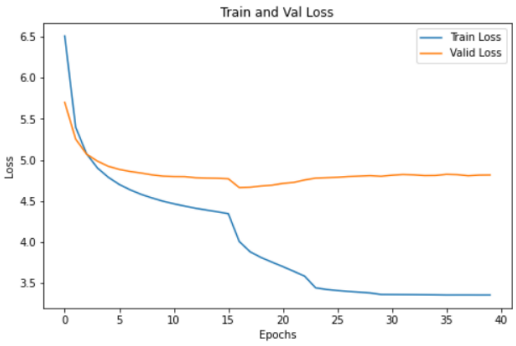

Figure 1: *Train and valid loss during a training*

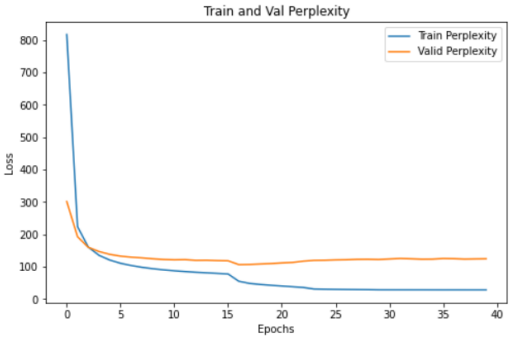

Figure 2: *Train and valid perplexity during a training*

5.2. Analysis on the sentences

It can be possible that the length of the sentence, and consequentially the length of the context, influences the perplexity. In one hand, the longer the sentence, the longer the net has to keep the content in memory. In the other, the longer the context, the more information the net has to predict the word: the set of words that go well with the preceding words should be narrower and narrower.

To evaluate this, two types of experiment were carried out. First of all, the batches with best and worse performance on the test set were considered, and the lengths of the sentences averaged. The two values, worst perplexity batch average length and the best perplexity batch average length, don't differ in a relevant way. In addition, some example sentences were sampled from the test set. For each, subsentences were formed: starting from the first word of the sentence, a new word is added at every iteration to create a dataset with the same sentence of increased length. Also in this case, it doesn't seem that the length has a particular relevance on the perplexity.

he	's							
he	's	not						
he	's	not	exactly					
he	's	not	exactly	sitting				
he	's	not	exactly	sitting	pretty			
he	's	not	exactly	sitting	pretty	at		
he	's	not	exactly	sitting	pretty	at	this	
he	's	not	exactly	sitting	pretty	at	this	stage

Table 2: Example of subsentences created by the sentence "he 's not exactly sitting pretty at this stage"

		RUN1	RUN2	RUN3	RUN4	RUN5	Average
Without early stopping	train	28.275	29.790	28.033	29.870	28.761	
	valid	126.157	123.071	123.381	120.386	125.524	
	test	121.569	118.416	120.588	116.750	120.999	119.665 $\pm$1.806
With early stopping	train	46.0723	56.833	52.443	56.169	49.278	
	valid	108.549	106.731	106.853	106.261	108.665	
	test	105.368	104.330	104.534	103.823	104.235	104 $\pm$0.51

Table 1: *Results of five consecutive runs*

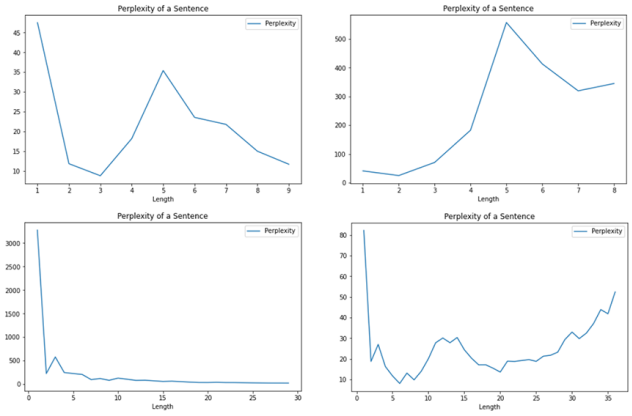

Figure 3: *Perplexity trend of four random sentences of increased length*

Moreover, some novel words that introduced high perplexity were inquired: the idea is that maybe the model can't predict combinations of words it has not seen in the train nor in the valid set, but appear in the test set. This was partially true: for instance *"once again"* lead to a perplexity of over 3000, while the most probable prediction of the model was *"once the"*. In the test and in valid set, *"once the"* is present twice the times of *"once again"*, both on the center and at the beginning of a sentence. However, *"once again"* was still present in the sets used for training, so a PP of over 3000 it is still a bit too high. Furthermore, this is an example of a net that fails to understand some common co-occurrences, like *once again*.

5.3. Multiple run and early stopping

Multiple runs were carried out to corroborate the results and to confirm the performance remains constant in different trials. For each run, the model was trained and saved at its minimum perplexity on the valid set. In the Table 1, we can see the perplexity values on train, valid and test set at the end of the training versus the model saved at its minimum perplexity on the valid set. The importance of the early stopping is evident: the perplexity is reduced by an average value of 15.

6. Conclusion

A language model with RNN architectures was constructed. It is clear that the basic versions of Vanilla RNN and LSTM are bound to overfit, and that regularization techniques are essential to create a good model. Here some regularization approaches were investigated, but many more are used and invented. In addition, some error analysis on the length and on the frequency of the subsentences was carried out, leading to exclude some hypothesis. It really shows how neural networks are efficient but are also difficult to understand from a human point of view and that critical aspects for humans and nets differ.

7. References

Merity, S., Keskar, N. S., Socher, R. (2017). Regularizing and optimizing LSTM language models. arXiv preprint arXiv:1708.02182

Mikolov, Tomáš, et al. Extensions of recurrent neural network language model. In: 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2011. p. 5528-5531.