Recurrent Neural Network Language Model Implementation

Davide Lusuardi
Department of Information Engineering and Computer Science
University of Trento
Trento, Italy
davide.lusuardi@studenti.unitn.it

***Abstract*—Language models are central to many Natural Language Processing (NLP) tasks as they can provide word representation and probability indication of word sequences. Some of these tasks include Machine Translation and Speech Recognition. Recurrent Neural Network Language Models (RNNLMs) improve the performance of traditional Language Models. In this work, we show how to implement a Recurrent Neural Network Language Model using PyTorch along with the implementation of simple LSTM and GRU cells. This report also shows the performance obtained on the Penn Tree-bank dataset using the perplexity metric (PPL).**

I. Introduction

Language modeling task consists in predicting the probability of the next word. It is a crucial component in real-world applications such as Machine Translation and Automatic Speech Recognition. For example, a translation system generates multiple translations for the same sentence and the language model scores all the sentences and decides the most likely one.

More formally, Language Models assign a probability to a sequence of words: given a text corpus with a vocabulary V and a sequence of words $w_1, w_2, \ldots, w_{t-1}$, we need to compute the probability distribution of the next word w_t [1]:

$$P(w_t|w_1, w_2, \ldots, w_{t-1}) \tag{1}$$

where w_t can be any word in the vocabulary V. Language models can operate at different levels: character level, n-gram level, sentence level or paragraph level.

A recurrent neural network (RNN) is a type of artificial neural network that operates on sequences of data of variable length. In the case of RNN, the output from the previous step is fed as input to the current step. This is particularly useful in applications where there is the need to remember the previous state.

Recurrent Neural Net Language Model (RNNLM) is a type of neural network language model that exploits a RNN cell. There are several different types of RNN cells, but we will focus on Long-Short Term Memory [2] and Gated Recurrent Unit [3] cells. Since a RNN can deal with the variable length data, it is suitable for modeling data such as sentences in natural language.

II. Evaluation of language models

A. Evaluation metric

Perplexity (PPL) is the commonly used evaluation metric for language models. Perplexity is defined as [4]:

$$PPL(W) = \frac{1}{P(w_1, w_2, \ldots, w_N)} \tag{2}$$

where W is the test set. Note that a lower perplexity indicates a better model.

Perplexity can also be defined as the exponential of the cross-entropy:

$$PPL(W) = 2^{H(W)} = 2^{-\frac{1}{N} log_2 P(w_1, w_2, \ldots, w_N)} \tag{3}$$

B. Dataset

The experiments have been conducted on the Penn Tree Bank (PTB) dataset [5], which consists of 929k training words, 73k validation words, and 82k test words and a vocabulary composed of 10k words. The material annotated includes IBM computer manuals, nursing notes, Wall Street Journal articles, and transcribed telephone conversations. The dataset is available from the website *deepai.org* [6] and is already splitted in three files that correspond to train, validation and test dataset.

The end-of-string token (`<eos>`) has been appended to all sentences in order to define its end and the unknown token (`<unk>`) replaces every unknown word in the validation and test corpus.

Words of the corpus are encoded as their position in the vocabulary and batches of data of shape `(sequence length, batch size)` are generated and fed to the network, where the sequence length and the batch size are two hyperparameters.

III. Model implementation details

In this section, we explain some implementation details of the designed RNN Language Model. The model is composed of an Embedding layer, a RNN cell and a Fully-Connected layer. The model supports two main types of RNN cell: LSTM and GRU. A simple version of these cells has been implemented.

The embedding layer is required in order to represent individual words as real-valued vectors in a predefined embedding

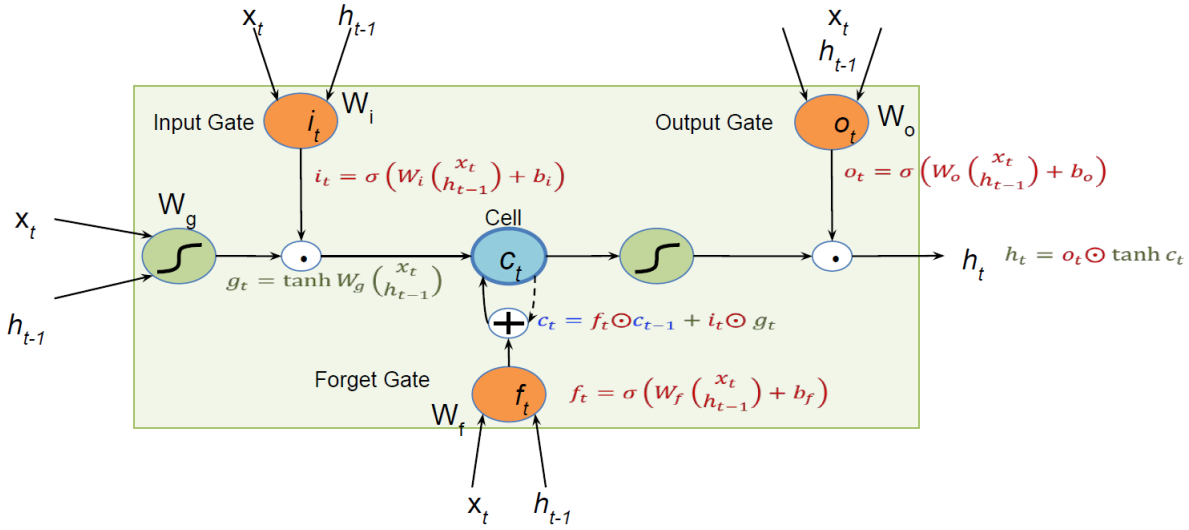

Figure 1: Long-Short Term Memory cell structure and related equations [7].

space. Each word is mapped to one vector and the vector values are learned: words that have the same meaning should have a similar representation.

The dropout regularization technique [8] has been applied in order to obtain better generalization performance without overfitting too much the training set. Dropout has been applied on the outputs of each RNN layer, with a different dropout rate for the last layer.

A common problem of Recurrent Neural Networks is the *"exploding gradients"* due to their instability during training. A possible easy solution is to apply gradient clipping [9]: the idea is to rescale the gradient to keep it small when it gets too large. This helps gradient descent to have a better behaviour.

A. LSTM cell

The structure of the Long-Short Term Memory cell is shown in Figure 1 and consists of three main gates:

1) The *input gate* decides how much input information is added to the current state and is caracterized by the following Equation:

$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right) \tag{4}$$

2) The *forget gate* decides how much of the past should be remembered and is caracterized by the following Equation:

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right) \tag{5}$$

3) The *output gate* decides the output based on the current state and is caracterized by the following Equation:

$$o_t = \sigma\left(W_o \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_o\right) \tag{6}$$

The memory cell c_t is updated as follows:

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh\left(W_g \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}\right); \tag{7}$$

and the new hidden state h_t is computed:

$$h_t = o_t \odot tanh(c_t). \tag{8}$$

The weights W_i, W_f, W_o, W_g are initialized as proposed by Xavier et al. [10] and biases b_i, b_f, b_o are initially filled by zero [11]. Both weights and biases are learned during the training process.

B. GRU cell

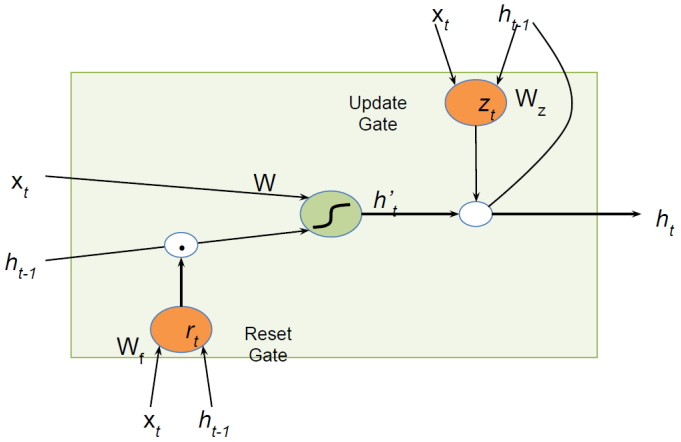

Figure 2: Gated Recurrent Unit cell structure [7].

The structure of the Gated Recurrent Unit is shown in Figure 2 and consists of two main gates:

1) The *reset gate* is used to determine how much of the past information to forget and is caracterized by the following Equation:

$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_r\right) \tag{9}$$

Hyperparameter	Value
Batch size	128
Sequence length	40
Embedding size	200
Hidden size	250
Layers	3
Dropout cell	0.2
Dropout output	0.5
Clip value	0.2
Epochs	60
Learning rate	40

Table I: Hyperparameters value.

2) The *update gate* helps the model to decide how much of the past information needs to be considered and is caracterized by the following Equation:

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_z\right) \tag{10}$$

Finally, the new hidden state h_t is computed as follows:

$$h'_t = tanh\left(W \begin{pmatrix} x_t \\ r_t \odot h_{t-1} \end{pmatrix}\right) \tag{11}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h'_t \tag{12}$$

The weights W_i, W_f, W_o, W_g are initialized as proposed by Xavier et al. [10] and biases b_i, b_f, b_o are initially filled by zero. Both weights and biases are learned during the training process.

IV. Results

The model with a 3-layers LSTM PyTorch cell has been trained on the Penn Tree-bank training set for 60 epochs, taking about 23 seconds per epoch and 23 minutes in total on a Nvidia Tesla T4.

Stochastic gradient descent (SGD) has been adopted as optimizer and the Cross-Entropy criterion as loss function.

The hyperparameters have been tuned on the validation set and the best hyperparameters found are shown in Table I. A scheduler decays the learning rate of half every 20 epochs.

Figure 3 shows the perplexity of training and validation and Figure 4 shows the loss over the training process. The best model, i.e. the one with the lowest validation perplexity during training, obtains a perplexity equals to 98.8 and 103.3, respectively, on the test and validation set. These results are not much higher than the performances reported by Zaremba et al. [12], i.e. a perplexity of 82.7 and 86.2, respectively, on the test and validation set.

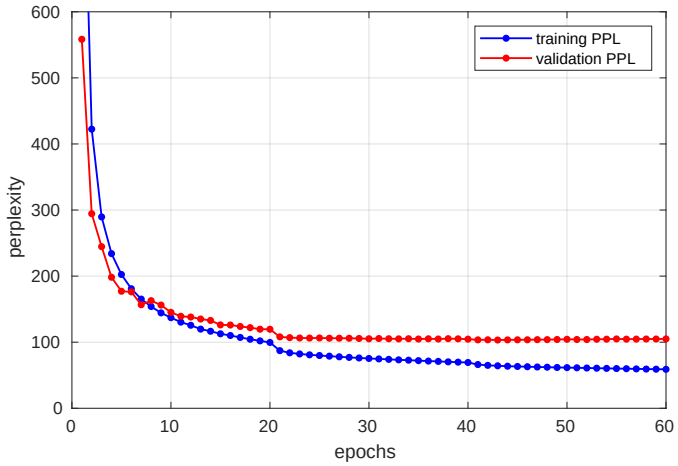

Figure 3: Training and validation perplexity over the training process.

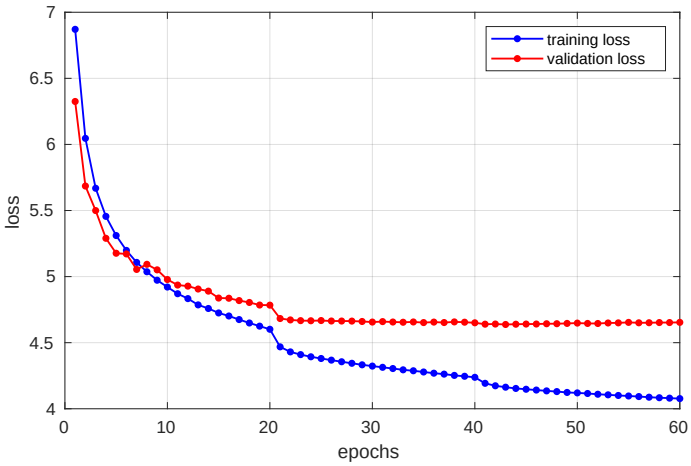

Figure 4: Training and validation loss over the training process.

References

[1] "How do language models predict the next word?" https://towardsai.net/p/nlp/how-do-language-models-predict-the-next-word.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[4] "Perplexity in language models," https://towardsdatascience.com/perplexity-in-language-models-87a196019a94.

[5] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993. [Online]. Available: https://aclanthology.org/J93-2004

[6] "Penn treebank dataset," https://data.deepai.org/ptbdataset.zip.

[7] E. Ricci, "Recurrent neural networks," Deep Learning course, 2021.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.

[9] "What is gradient clipping?" https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48.

[10] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[11] "Weight initialization techniques in neural networks," https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78.

[12] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," 2015.