SFC - LSTM Network Performance Demonstration

Michal Glos (xglosm01)

December 5, 2021

1. Introduction

As an assignment for this term paper, I received 'Demonstration of LSTM Network Activity'. I implemented this neural network in Python 3, the libraries used are listed in section 3. The project solution contains a total of 3 files, namely the module implementing the LSTM network, a module enabling working with a dataset and a script that is a command-line interface for a neural network. I describe the implementation in detail in section 3 and work with the interface in section 4.

I decided to demonstrate the activity of the network on a public data set from Amazon, which is available here. Specifically, to predict the rating given by the customer (1-5 stars) from the text of his product review.
After training, the network accurately predicted customer ratings with a 58.22% success rate, see section 5.

2 LSTM

LSTM, or Long short-term memory, is a recurrent neural network architecture (hereafter referred to as RNN).
Unlike ordinary feedforward neural networks, recurrent networks have feedback, i.e. they have not only input data, but also e is the output from the previous step, from which these networks gain abilities such as understanding the context and analyzing sequences of data of inhomogeneous sizes.

A concrete LSTM architecture consists of LSTM cells, which often form a superordinate architecture, and their outputs are usually analyzed by fully connected layers. The LSTM unit consists of cells, input gates, output gates and "forgotten gates", described in section 3.2. The advantage of LSTM over classic RNNs is the ability to remember important data after many more steps. While in classical RNNs the information decreases moderately with each subsequent iteration, the LSTM architecture solves this problem precisely with its gates, when the network de facto decides whether if the current information at the input is important, if not, and adjust its internal state accordingly. The LSTM network is suitable for classification, processing and prediction of sequential data.

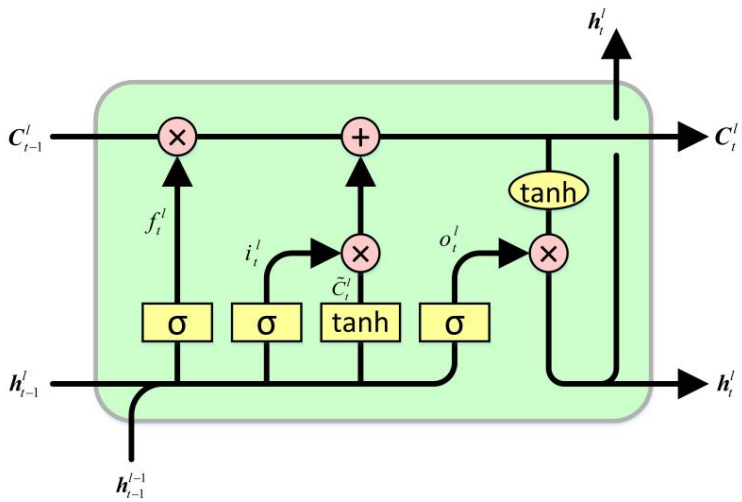

Figure 1: Graphical representation of an LSTM cell. The exit gate is marked with the forget symbol f t . C is t , inputs and t and also its output. C t l ah t are then the internal states of the cell, when h t t ty1 l ah ty1 are outputs cells from the previous iteration ty1 ty1 is the input that is the output of the previous layer of the network, or the input and a network vector. Retrieved from https: // colah. github. io/ posts/ 2015-08-Understanding-LSTMs/

3 Implementation

Part of this semester work are 3 sets described in the following subsections and several other sets described in more detail in subsection 3.4.

3.1 amznDS.py

This file implements the amznDS module for working with a dataset from Amazon, containing product review data. Due to the cardinality of the data set, I used only the part of it containing reviews of computer games, but by editing the global variable DATA URL referring to gzip data file, the script can be modified to work with other parts of the data set. The gzip, json, tqdm, numpy, time, os, re, sys, pickle and requests libraries were used to implement this module. This module implements 2 operations, namely obtaining data suitably formatted for learning neural networks, described in subsection 3.1.1 , and creating a dictionary for addition of words, which is described in subsection 3.1.2, the so-called word embedding.

3.1.1 Acquisition of datasets

The data set can be obtained by calling the get data function in the format of a dictionary list containing two items. The first is overall - the customer's evaluation of the product, the value of which is a whole number in the interval from 1 to 5 and reviewText containing a list of strings (words) given the review. The function can be called with two optional arguments, namely one class size, with which we specify the amount of data for each value of the odds, and force, which, if set to True, ignores previously stored data sets in pickle format and will generate the data set again. The default setting of this argument is True due to memory limitations on the Merlin server.

This function will first build the path to the pickle file, identified by the one class size argument, and if the given file exists and the force argument is set to False, the file will be loaded and the function will return the prepared data set. However, if the file does not exist or the force argument was set to True, the function builds the data set in the following way. First, the load data function is called, which first determines if the json with the data set is in the DATA PATH file (the data is divided into two files due to the memory limitations of the Merlin server). If found here, they are loaded and returned.
However, if the data set is not found in this file, the download data function is called, which first determines if the data set in gzip format is present in the DOWNLOAD file PATH. If not, it downloads it from the url stored in DATA URL. Next, the gzip file is extracted into DATA PATH files, from where they can be loaded. Next, the raw data is processed by the filter json entries function, which returns from the list of dictionaries a list of sub-dictionaries containing only the keys specified and the optional keys argument. Reviews are further processed by the filter short data function, which removes review data shorter than 64 characters (this value can be set with the optional min len argument). Furthermore, the same amount of reviews is selected from the data for each rater using the eliminate a priori inequity function. The required quantity can be changed by setting the optional one class size argument. Furthermore, in the data, the rating value is converted to a whole number and the text of the review is divided into words. The data is randomly shuffled using np.random.shuffle and saved as a pickle file if necessary.

3.1.2 Inserting words

Since I decided to demonstrate the activity of the LSTM network on the analysis of textual data, it is necessary to ensure the way in which individual words will be transformed into vectors. This is done by the create embedding dict function, which creates a pickle file containing a dictionary in which a vector of the same size is stored for each word from the data set, which They serve as the input of the neural network. The dictionary is created by iterating over each word in the dataset, which is first stripped of non-alphanumeric characters and uppercase letters are converted to lowercase to avoid creating multiple vector for differently written words of the same meaning. Next, the number of occurrences of each word in reviews divided by rating is calculated. Thus, for each word, a vector of size 5 will be created, where the first element of the vector contains the number of occurrences of the given word in reviews with a rating of 1, and the second in reviews with a rating of 2 etc... The components of this vector are then divided by the total number of words in the reviews with the given rater. Subsequently, each vector is normalized so that the sum of its components is equal to 1. The dictionary is then saved to a file on the EMBED path in pickle format.

3.2 lstm.py

This module mainly implements the LSTM class, see section 2, which represents an LSTM neural network and provides methods for its training, evaluation and several auxiliary methods. In this module I used tqdm, os, numpy, sys, pickle, math and matplotlib libraries.

3.2.1 LSTM class

This class is initialized with three arguments. The first of them is n classes, which determines the number of output neurons and corresponds to the number of classes for classification. The next is lstm cells, which determines the number of LSTM cells, and the last argument is learning rate, which determines the speed of the learning network. Part of the initialization of this class is also the creation of a model - a dictionary containing weights and biases of LSTM cells and connected to them fully connected layers and also loading the dictionary for inserting words.

Another basic function for working with the LSTM model is forward step, which is called with arguments in X - a word for which the forward step is performed and old ha old c - outputs of LSTM cells from the previous step. If the previous step is performed on the first word from the review and the previous step does not exist, these values are vectors of zeros. The function first converts the input word to a vector (insertion of words) and calculates all network parameters according to the equations below. In addition, it stores these parameters in the cache, because they are later used in the error backpropagation algorithm.

$$
\begin{aligned}
it &= \ddot{y}([ht\ddot{y}1, xt]Wi + bi) \quad ft = \\
&\ddot{y}([ht\ddot{y}1, xt]Wf + bf) \quad ot = \\
&\ddot{y}([ht\ddot{y}1, xt]Wo + bo) \\
Ct &= \tanh([ht\ddot{y}1, xt]Wc + bc) \\
Ct &= ft \ddot{y} Ct\ddot{y}1 + it \ddot{y} C \quad ht = \\
&\tanh(Ct) \ddot{y} ot
\end{aligned}
$$

Figure 2: Equations describing the LSTM model. The letter it denotes the entry gate, ot the exit gate and ft the oblivion gate. The lower index t denotes the iteration number of the network, xt is the input vector of the network. Operation [x, y] meaning concatenation of two vectors. Ct and ht indicate the state of the cell. Variables corresponding to scheme 1.

The function that ensures the training of the LSTM network is the backward step, which calculates the gradients of the weights and the sum of the values stored in the cache from the calls Forward step function for matching input word. This function is an implementation of the error backpropagation algorithm and uses cross entropy as the error function. Its arguments are y gt containing the true evaluation of the review, cache with the stored values of the forward step function, dh chain and dc chain containing the calculated values and in the previous step, the backward step function, which is used in gradient calculations according to the so-called chain rule.

Another very important function is the train step, which calculates the output of the network for one review and calculates the gradients of weights and displacements. The function receives and arguments X containing a list of words of one review and a containing the true class of the review. The function first successively calls the forward step function for all words of the review and stores all its outputs in the cache list and from the last output calculates the state function - cross entropy. Next, the list of caches is iterated from the end and the accumulated gradients are calculated using the backward step function. The function for each review returns the accumulated gradients and the value of the loss function.

The training of the network is then implemented by the train function, which accepts 6 arguments, namely XX containing a list of reviews and Y a list of ratings ÿ to the reviews. The others are testX and testY, designed to evaluate the network between training epochs, model name representing the template for creating pickle files of the learned model, and epochs determining the number of that will be iterated over the input training data.

First, the training data and the true classification are randomly mixed, and the train step function is called over each element from the training data, which calculates gradients for adjusting weights and shifts. Next, the apply gradients function is called, which will adjust the weights and displacements of the model by subtracting the calculated gradients multiplied by the learning speed. After the end of each epoch, the network is evaluated on the data passed in the testX and testY arguments, and the results are written to the terminal. Then the model is saved as a pickle file and the development of the loss function is plotted using the plot losses function both for the completed epoch and for all iterations of the current learner.

The network is then evaluated using the evaluate function, which accepts two arguments, namely XX and Y containing the review texts and the truths assigned to them It will be evaluated. The function above each element of this data set calls the evaluate entry function, which returns a probability vector of the relevance of the review text to the given classes. The evaluate function then returns a list of pairs, the first element of which is the vector obtained by the evaluate entry function and the second is the truth value of the evaluation. Human-understandable output can then be obtained by passing the returned list to the print success rate function, which will print the success rate of the given evaluation on the terminal.

3.2.2 Naive evaluation

Naive evaluation of the data set is carried out using the naive evaluate function. This function accepts two arguments, namely XX containing the review texts and Y containing their true classification. Basically the function iterates through all the reviews and for each of them it converts all its words into vectors using a dictionary to input the words that intersect for each review. The function returns a list of pairs, where the first element is a vector of length 5, the sum of all vectors for words from the review, and the second element is the truth and classification of the given review. The success rate of the evaluation can be printed again using the print success rate function.

This method achieves a success rate of 61.91% on the entire data set.

3.2.3 Auxiliary functions

The lstm module also contains several auxiliary functions. Specifically, the sigmoid function serving as the value of the sigmoid function for the input vector, d sigmoid serving as the value of the derivative of the sigmoid function from the input vector, d tanh serving as the value calculation of the derivative of the tanh function, softmax to implement the softmax function, cross entropy to calculate the cross entropy and print success rate, which receives the outputs of the functions naive evaluate and evaluate, which It processes and the percentage of evaluation success is printed on the terminal.

3.3 lstm-cli.py

This script serves as an interface to the model from lstm.py. It enables the model to be trained, evaluated and/or perform a naive evaluation. All these actions are described in more detail in section 3.2. This script is run with several arguments:

- --naive-evaluate: Evaluation evaluation of reviews according to algorithm 3.2.2. It has priority over --evaluate.
- --evaluate: Estimation of rating of reviews according to the lstm network
- --train-data-entries n: Number of reviews for training the network trick (n > 0 ÿ n ÿ N)
- --eval-data-entries n: The number of reviews for evaluating the lstm network (n > 0 ÿ n ÿ N)
- --data-ratio r: Part of the reviews used to train the tricks of the network, the remaining part of the entire dataset will be used to evaluate the network . It has priority over --eval-data-entries and --train-data-entries (r ÿ ÿ0, 1ÿ ÿ r ÿ R)
- --learning-rate r: Learning rate of the lstm network (r > 0 ÿ r ÿ R)
- --lstm-cells n: Number of lstm cells in total (n > 0 ÿ n ÿ N)
- --epochs n: Number of epochs for training the network (n > 0 ÿ n ÿ N)
- --load-model path: Path to the file with the stored lstm network
- --save-model path: Template for the path to save the lstm network file

Running examples are in section 4. This script works with argparse, pickle, os and numpy libraries.
First, arguments passed to the script when running from the command line are processed using the argparse library. These arguments further specify the size of the data set with which it will work, and if the --naive-evaluate argument was not passed to the program, the LSTM model will be loaded from the pickle file, or according to the passed arguments u will initialize the new model. Then, according to the argument, the required process of learning, evaluation or naive evaluation is started.

3.4 Other files from the submitted project

There are two other folders in the submitted directory, namely ./models and ./data. In the ./models directory there is a pickle file containing a trained LSTM network with the highest classification accuracy, examples of their use can be found in section 4. There is no file in the ./data directory at the time of submission due to the size limitation of submitted files, but after starting 4 files are created here, namely a data set in gzip format, an unpacked data set divided into two json files and a pickle file for inserting words.

4 Examples of implementation

python3 lstm-cli.py Run
model training with default parameters.

python3 lstm-cli.py --data-ratio 0.95 --save-model models/my model.p --learning-rate 0.0001 --lstm-cells 69 --epochs 10
Starting the training of the model with the specification of the ratio of the training data to the sum of the training and evaluation data at 0.95 (all available data are loaded), the specification of the path to Save the model to the template model/my model epoch Np, specify the learner speed to 0.0001, the number of trick cells to 69, and the number of training epochs to 10.

python3 lstm-cli.py --train-data-entries 200000 --eval-data-entries 20000
Another option is to specify the training data set directly by the number of training and evaluation data.

python3 lstm-cli.py --evaluate --load-model models/LSTM 1e-06 64 epoch 18.p --eval-data-entries 100000
Evaluation of the model stored in the path models/LSTM 1e-06 64 epochs 18.p on 100000 data.

python3 lstm-cli.py --evaluate --load-model models/LSTM 1e-06 64 epoch 18.p --data-ratio 0

Evaluator of the model on all available data.

python3 lstm-cli.py --naive-evaluate --data-ratio 0 Use the naive
evaluation algorithm on all data.

5 Classification results

The performance of the LSTM network is affected by 2 parameters, namely the learning speed and the number of LSTM cells. Several experiments have shown that these parameters for this particular case of Amazon review classification have an effect mainly on the stability of the network during training. By instability in this case, we mean when the loss function during training begins to rise rapidly and the system begins to classify randomly, see graph 5. If the learning speed is too If the number is too high, or the number of LSTM cells is too low, the training of the LSTM network is very unstable. The ideal value of the learning speed is below 0.00005, using which the network can learn reasonably quickly and is relatively stable. By reducing the learning speed, more accurate results can be achieved, but the training time is then too long. Experiments with learning speed are shown in Table 2. A larger number of LSTM cells also leads to a more stable learning process, experiments with this parameter are shown in Table 1. LSTM network with 96 cells, however, it required 12 hours to train for 1 iteration through the entire data set, therefore, although it would have been possible to achieve higher accuracy with a larger number of cells, this network configuration was most of the configurations included in the experiments. The ideal number of LSTM cells for a relatively acceptable learner length and relatively accurate results is 64.

LSTM cell	1	16	32	64	96
epoch 1.	23.72%	45.05%	45.33%	46.45%	47.16% 26.01% 31.96%
epoch 2.	46.41%	46.34%	46.02%	26.52%	20.00% 20.18% 47.58%
epoch 3.	44.53%				

Table 1: Evaluation results of the LSTM network with the learner speed parameter set to 0.00005 and a variable number of LSTM cells.

The speed of learning	10ÿ3	5 × 10ÿ4	10ÿ4	5 × 10ÿ5	10ÿ5	5 × 10ÿ6	10ÿ6
epoch 1.	20.00%	35.62%	43.89%	46.45%	47.87%	47.47%	22.03%	20.00%	23.32%	40.33%
epoch 2.	46.34%	52.14%	52.07%	25.49%	20.00%	23.51%	20.00%	47.58%	32.41%	52.83%
epoch 3.	42.23%						

Table 2: Evaluation results of the LSTM network with the learner speed parameter set to 0.00005 and a variable number of LSTM cells.

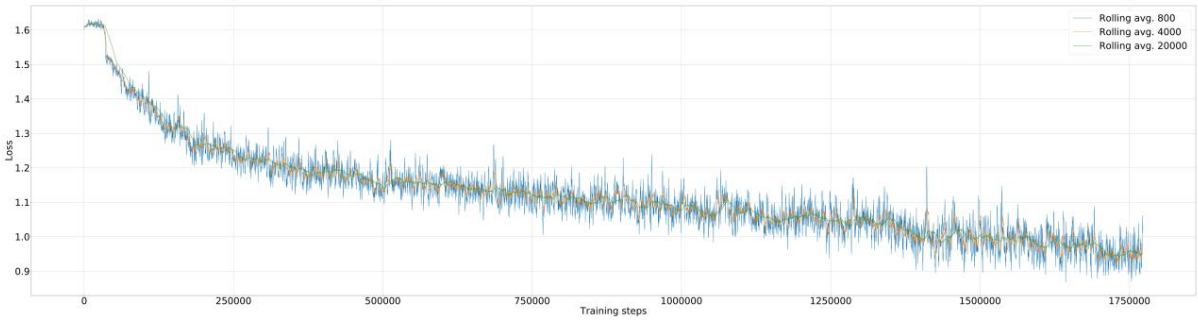

Figure 3: This graph shows the progress of a stable network training for 3 epochs on a 64-cell LSTM model and a learning rate of 0.00001.

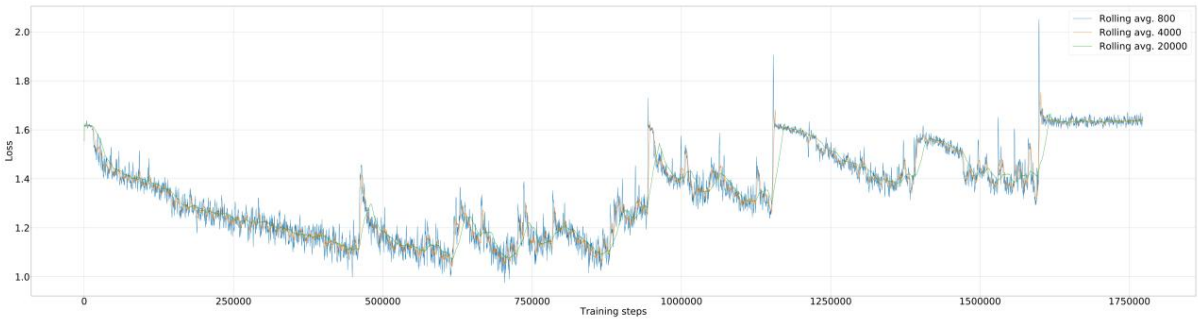

Figure 4: This graph shows the progress of an unstable training network for 3 epochs on a 16-cell LSTM model with a learning rate of 0.00005.

After analyzing the classification success of networks trained with different parameters, I continued to train the network with the number of LSTM cells 64 and learning speed 0.000001 and 0.000005. I achieved the best result after 18 epochs with a learning rate of 0.000001, and an accuracy of 58.22%.

6 Close

Although the network is relatively successful in learning the classification of reviews on the overall rating, even the best achieved results (58.22%) are less accurate than when using the naive algorithm (61.91%). The accuracy of the network could probably be improved by using a more sophisticated optimizer, such as Adam, which is often used by LSTM network scientists. an. Another way to improve the accuracy of this network is to use a more sophisticated network architecture, where LSTM cells appear in both horizontal and vertical their layers or even using a more sophisticated network architecture classifying the outputs from the LSTM network into classes. The last of the options to refine the calculation would be different and the choice of inserting words that would carry semantic information about the word, for example using neuronov´es´ÿtÿe Word2Vec.