DD2424 Project - character-level text classification with different RNN architectures

No Author Given

No Institute Given

Abstract. ...

Keywords: We would like to encourage you to list your keywords within the abstract section

1 Introduction

We are looking at a text classification problem with short text and character-level classification. We will be using character-level classification. This means that the network processes the input one character at a time. Each character results in an output consisting of the probabilities for each letter for the next character. If we have a 58-character alphabet, the hidden state will consist of the probability of each of those 58 characters for the next letter in the sequence. So for each character, one such prediction is produced, as well as a hidden state, which is fed into the next step.

Short-text classification is hard in that there is less information to go off with little to no grammar or syntax. Some applications of short-text classification include search queries and information retrieval, mapping a product name to its associated product, the classification of titles, questions, sentences, and short messages.

The texts we are looking at are names of cities, and the classes are the names of the country the city belongs to. To begin with, we used the world-cities data set. We later abandoned it for geonames, because it has more samples. There are circa 240 categories (countries) in the data sets, but we will limit the number of categories so as not to make the categories too unevenly distributed (ie remove countries with too few cities in the data set).

The inspiration for the project is a tutorial in which a recurrent network is used to classify names as belonging to certain nationalities, using the Python framework *Pytorch*. To start with, the model of that tutorial is to be replicated, then additional architectures will be implemented and compared. Questions we ask ourselves include: to what degree would a deeper network help in this problem? What sort of architectures are good in this application? Would an LSTM-layer in the recurrent network give better results?

To measure the success of the text classification, we will look at the precision, recall and f1-score of each category in the data set. This because it gives a more nuancued evaluation than accuracy, seeing as how the data set used is heavily skewed. Furthermore, we will visualize the results with a confusion matrix.

2 Background

3 Approach

4 Experiments

The first experiment was simply taking the model from the tutorial and running it on the first data set we had decided upon, the world-cities data set.

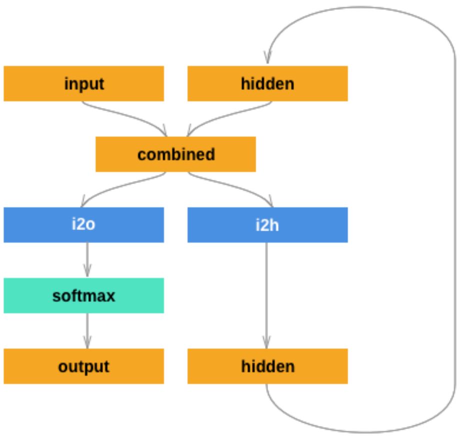

Fig. 1. Model from the text classification tutorial.

The model in the tutorial is seen in figure 1, where i2o and i2h are simple linear layers that do a $y = Wx + b$ type calculation. Softmax is used to assign the different categories probabilities - it is a good way to represent a categorical distribution in a multi-class problem such as this one. The criterion for loss that is used is negative log-likelihood loss. The optimizer that is used is stochastic gradient descent. No momentum is used. The *combined* layer simply concatenates the input vector and hidden vector.

The data preprocessing was simple: the data was converted to ASCII-characters and the data was filtered to only have countries with at least 100 cities in the data set. This because the initial distribution was a bit too uneven, as seen in figure 2, with many countries having very few cities, some even just 1 or 2. What remained was 18684 data points.

The training was done in a similar way as in the tutorial, with random sampling with replacement from the data set. The final confusion matrix looked is presented in figure 3, and the final average f1 score (averaged over all the categories) was 0.3. The test accuracy was 31 percent.

While it is true that the model has a pretty good accuracy for some of the categories, it is still performing pretty unevenly for the categories. As seen in the

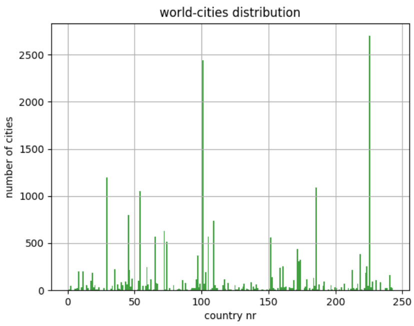

Fig. 2. World-cities distribution.

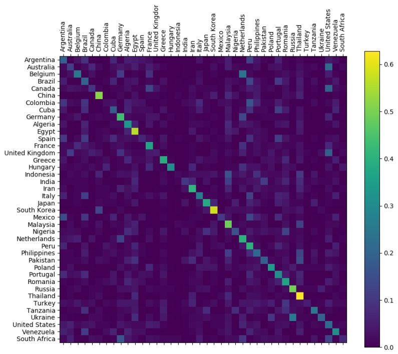

Fig. 3. Confusion matrix of first RNN model.

figure, it is moderately good at classifying countries like Thailand and Korea, but not very good at classfiying countries like Colombia or Indonesia (indeed it seems to never predict them). We do not see a clear pattern from the class distribution (for example the US is overrepresented in the data set but is not predicted more often than other countries). This is because of how the random sampling is made - first a random category is chosen, then a sample will be taken randomly from that category. Therefore, the number of times we're exposed to samples from different categories will be more or less even.

4.1 Introducing LSTM and GRU

We wanted to see if an LSTM-layer could improve the performance of the network. LSTMs are units used in recurrent networks that are composed of several gates - an input gate, output gate and forget gate. The LSTM can choose to "remember" or "forget" inputs from the past. This mechanism allows them to avoid the long-term dependency problem. However, this might not matter in this particular short-text classification problem, since there should not be long-term dependencies.

LSTMs are also supposed to be good against exploding and vanishing gradients, which is a common problem in backpropagation-through-time, the most common learning algorithm for standard recurrent networks.

When we have an LSTM cell in Pytorch, it takes an input, a hidden state and a cell state and it does the following computations

$$i_t = \sigma(W_{i,i}x_t + b_{i,i} + W_{h,i}h_{t-1} + b_{h,i}) \tag{1}$$
$$f_t = \sigma(W_{i,f}x_t + b_{i,f} + W_{h,f}h_{t-1} + b_{h,f}) \tag{2}$$
$$g_t = tanh(W_{i,g}x_t + b_{i,g} + W_{h,g}h_{t-1} + b_{h,g}) \tag{3}$$
$$o_t = \sigma(W_{i,o}x_t + b_{i,o} + W_{h,o}h_{t-1} + b_{h,o}) \tag{4}$$
$$c_t = f_t c_{t-1} + i_t g_t \tag{5}$$
$$h_t = o_t tanh(c_t) \tag{6}$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input, h_{t-1} is the hidden state at the previous time step and i_t, f_t, g_t and o_t are the input, forget, cell and output gates. [1]

In other words, the LSTM cell works by applying nonlinear functions to both inputs and internal variables.

The first LSTM experiment was simply running the same model as before, but replacing the linear layer for an LSTM layer.

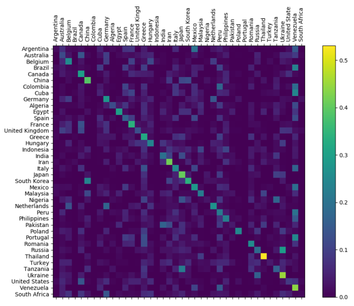

Fig. 4. Confusion matrix of first LSTM model.

It ended up not performing as well as the standard RNN after having been trained in the same way (same hyper-parameters, same number of epochs). As seen in figure 4, the max accuracy is 0.5 rather than 0.6, and there are more incorrect predictions.

Some more iterations of different LSTM architectures were tried. Things that were changed for each model are laid out in table 1, and the results of the models are seen in table 2.

Table 1. The changes introduced to different models.

LSTM model nr	difference introduced
1	introduced LSTM layer
2	did combined input and hidden (as in the original RNN)
3	used a new data set called geonames, that had more training samples
4	the previous model ran on all countries in geonames that had more than 100 cities. here the data was filtered more, and only countries with more than 300 cities were used
5	tried not having class weights, to see if it actually helped
6	used two LSTM layers
7	instead of randomly sampling with replacement, training consists of running through the entire data set consisting of 81526 samples. it was trained for 10 epochs
8	ran the model for 20 epochs. checked validation error after every epoch. also introduced shuffling of the data before every epoch

Table 2. Results of the LSTM models.

model nr	average f1	test accuracy
1	0.18	21.6
2	0.2	21.9
3	0.04	7.5
4	0.14	17
5	0.2	17.2
6	0.1	12.8
7	0.23	27
8	0.3	38

All LSTM models performed worse than the original RNN, up until model 8. Introducing the data set with more data points did not immediately lead to better results. This because there was an even bigger difference between classes (see figure 5).

This seemed to indicate that the data set needed to be filtered more, so all countries with less than 300 cities were removed.

After filtering the data more, the results were a bit better. However, the accuracy was still pretty low. This because the random sampling of data points meant all categories received the same attention, which was not good in terms of accuracy because some categories were indeed heavily overrepresented in the data set but not in the predictions of the model. Likewise, the model did not

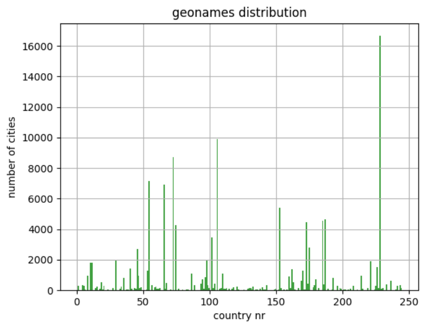

Fig. 5. Geonames data set distribution.

immediately perform better with two LSTM layers rather than one. It turns out that it simply needed to run for more epochs to account for the bigger data set and the bigger model. Indeed, model 8 performed best so far because it was run for more epochs.

After this, a GRU model was implemented and tested. It was run in the same way as LSTM model 8. And out of curiosity, a similar standard two-layer RNN was run in the same way. The result of this comparison is seen in table 3.

Table 3. Comparing the LSTM model with a similar model with GRU layers and one with linear layers.

model description	average f1	test accuracy
LSTM model 8	0.3	38
similar GRU model	0.32	42
similar standard RNN model	0.09	12

The GRU performed similar to the LSTM, just slightly better. However, it trained markedly faster. Meanwhile, the simpler RNN did not perform well, as it started suffering from exploding gradients. In its predictions, it exclusively predicted countries that are overrepresented in the data set.

5 Results

6 Conclusions

As [2] said.

References

1. Pytorch: torch.nn. `https://pytorch.org/docs/master/nn.html#lstm` (2018) [Online; accessed 16-May-2018].
2. Alpher, A., , Fotheringham-Smythe, J.P.N., Gamow, G.: Can a machine frobnicate? Journal of Foo **14**(1) (2004) 234–778