Introduction

The aim of this project is to replicate the results of Andrej Karpathy's blog post [1]. In it he demonstrates training a recurrent neural network to generate text character by character. The post shows examples of English text, Wikipedia markup, LaTeX markup and even C++ source code.

An implementation of the described model can be found in the deep learning library Keras's example set. The goal of this project is to use it on an Estonian dataset and to explore the model's hyperparameter space.

This project should provide insight into:
* how different hyperparameters effect the performance of the model
* comparing the quality of generated text
* differences when training on Estonian versus training on English datasets

Background

The concept of using RNNs for character by character prediction was first demonstrated by Ilya Sutskever et al in 2011 [2]. Besides the fun application of generating text, they describe how such models could be used in text compression. They compare the RNN solution to two previous state of the art methods, while achieving competitive results:
* Memoizer – a hierarchical Bayesian model
* PAQ – a data compression method, that uses probabilistic sampling. The sampling is done by a mixture of context-models (n-gram, whole word n-gram, etc). The mixing proportions (when to prefer which model) take into consideration the current context.

Dataset

I began by concatenating 160 books from Estonian literature (since year 1990). They were part of the balanced corpus, provided by Tartu University computational linguistics group. [3] The books were in the TEI format (uses XML syntax), requiring some parsing to get the pure text format I was after.

The resulting 35 MB dataset took too long to process. I estimated a reasonable size to allow me get results would be around 1 MB, as that got interpretable results in half a day intervals. I split the original concatenated dataset from the 1 MB marker, and ended up with Rein Pöder's “Hiliskevad” and a quarter of Ene Mihkelson's “Ahasveeruse Uni”.
Method

The architecture presented in both Karpathy's post and the Keras example set:
2-layer LSTM, 512 neurons
Dropout 0.2
Softmax activation
Categorical Crossentropy loss
RMSProp

For hyperparameter tuning, this is the set I tweaked, with their baseline values:
neurons, 512
dropout, 0.2
corpus_size, 1M
window_size, 20
batch_size, 256
epoch_number, 1

window_size is the substring size of the training examples.
epoch_number is a parameter for the fitting function.

Training in the original code is done in iterations, after each an example sample was produced. The epoch_number is a parameter for the fit function. At first glance it seemed to have a different effect then the iterations, but as expected, they turned out to be the same thing.

When conducting these experiments, it didn't make sense to me to use a validation set, as this problem seems a bit different from regular machine learning tasks. However, as described by Sutskever in [2], it would make sense.

Instead, I used a custom metric to accompany the training loss, described by eq 1.

\[
    accuracy_{temp} = \frac{\text{correct words}}{\text{generated words}}
\]  

(1)

We can affect the sampling of the text with a temperature parameter. Lower temperature means picking characters that are most probable according to the model.

The idea is, that models predicting correct words even at high temperatures, should be good.

For the experiment I tried various different hyperparameter values. I recorded the loss and the custom accuracy after each iteration.
Results

I obtained results from 14 different models. The models were trained for varying lengths of time, between a day and two days. All the trained models' cost functions are presented in figure 1.

![Figure 1. Cost functions of all the experiments.](image)

The longest, light blue line is the baseline, with default hyperparameter values.

![Figure 2. Loss functions for modified batch, corpus, window_size and dropout.](image)
Figure 2 displays cost functions for varying batch, corpus, window sizes, and dropout. Larger batch sizes were hindering, but it seemed a smaller batch size would make it better. Larger corpus did not improve the loss for a day of training. It could be the network is too small. Or it could take longer to achieve the same loss value, due to more input data. Larger window seems to have no effect. The network seems to care about character history of only up to 20. Larger dropout values worsen the loss. Unfortunately a smaller run on dropout of 0.1 got lost.

Varying the fitting function's epoch number seemed to have a positive effect, until I realised it's essentially the same thing iteratively running the fitting.

The strangest loss function was produced by one larger network, as presented in figure 4.
To see how the larger network compares to the baseline, I plot them both against the accuracy metric described in the previous section.

Figure 5. Comparison of networks sized 512 and 1024 with the accuracy metric.

The low temperature sample goes to nearly 100% in the beginning, due to generating only short, correct words. It's interesting, how the low and high temperature generated sample accuracies converge. Perhaps convergence would indicate that training is done. The bigger network displays quicker convergence and also a slightly higher accuracy on the high temperature samples. This would indicate a better model. However, reading through the generated texts, at around iteration 60, the smaller network seemed to make more sense. Some examples of generated text:

512 neurons:

**Low temp:**
“mees oli ta vastanud, et ta oli ka enam mingit neist temasse viinud parjada sellele peale mõnedadikale ja arusaatust taastasse saati. ja see kasvab veel tema peaaegu sees. nii et ma teadsin, et mulle tundus mulle ka enam midagi ja ette vastama. ma ei ole kui tema keegi meele.”

**High temp:**
“kinni üle metsavendi tema silmis vääna ja juulis sa siis naiselik. tegeles, et veel siis naiselik. tegelikult oli hette teeline piri, oli ma auendaid kordi alla peen, seda tuli ta rihumad kujulinalet.”

1024 neurons:

**Low temp:**
“kuid midagi ses maastikus, mis on see võimalik. ja millal ma seda kõrvalt vaatas see tahtis mind ema taastaks, kui mu kümmeks säärasega jooksid rõhkumise haamatust küünalda letk”

**High temp:**
“ma olin karini armendama näonud paidagi elanud. selgus, et veel siis naiselik. tegelikult oli hette teeline piri, oli ma auendaid kordi alla peen, seda tuli ta rihumad loodus.”
Discussion

Tuning the parameters mostly did not lead to a notable increase in either training loss, measured accuracy, or perceived text quality. The dataset I ended up using, was in the same size range, as the one used in the original example. The author of which had probably done a good job tuning the parameters already.

The original text was in English, and the samples from it made much more sense, then the ones generated on Estonian. This indicates how Estonian language has a more complicated structure, to be captured with such a model.

The main take away for me, was learning to set up such an experiment. This time it was a bit error prone, concluding in some lost measurements. Logging the results should be as automatic as possible, with minimum amount of manual work. The latter of which makes the process a bit tedious and also greatly increases the chance of mistakes.

Future work

There are some loose ends in this work, such as measuring smaller batch sizes, and seeing how the bigger network behaves after longer training. However, it was proved that the dataset is too small to obtain any reasonable results. So the first thing would be to increase the dataset greatly, and rerun the hyperparameter tuning process.

Following experiments should include the validation loss as well. Furthermore, it would be interesting to try different accuracy metrics, for example measuring the amount of distinct words in the samples, or checking how much of the sampled text is present in the original data. This would help gauge the models originality.

Conclusion

This research project in data mining set out to try the recurrent neural network model presented by Andrej Karpathy [1], originally by Ilya Sutskever et al [2], for generating Estonian text. An implementation of which was included in the Keras's example set worked quite well, so I set out to explore it's hyperparameters. The available timeframe set a limit on the dataset size, which coincided with the size of the example's dataset. Hyperparameter search revealed, how the original hyperparameters were good enough. A metric for measuring the models's performance was used, which observed the amount of correct words in the generated sample. This helps to automatically estimate the goodness of a model. Comparing with the results from the example, which used English text as input, it can be said Estonian language has a more complex structure. Future work should build on the fact, that 1 MB of data is too small for this kind of sampling.
References