Classification of Spoken Digits
Using Recurrent Neural Networks

github.com/jaroslavlanger/spoken-digits

Jaroslav Langer / langeja5@fit.cvut.cz / FIT, CTU, Prague

June 5, 2023

Abstract

This project classifies spoken digits in two languages English and Arabic. The classifiers are recurrent neural networks. One type contains vanilla RNN layers and the other type has LSTM cells. The sound features are not fed as is. They are pre-emphasized a converted into MFCC coefficients and padded to the length of longest one. The best model found for English digit classification worked even better when trained for classification of Arabic spoken digits. Final test accuracies were 0.72 and 0.92 respectively.

Keywords: Multi-class classification, Spoken digits, MFCC, RNN, LSTM.

1 Introduction

MNIST dataset is one of the most recognized datasets of all time. For some time, it is considered to be solved. However it still serves the purpose of a difficulty benchmark. Using it, we can explore something about the inherent properties of such task. We can ask questions such as: "Can I solve MNIST with only 10 neurons?" or "Is one convolutional layer enough to solve it?".

This project explore very similar datasets both consisting of recordings of spoken digits. The exploration of hyper-parameters takes place with the English language dataset. The Arabic counterpart is used to validate, whether the findings are specific to the English dataset. Or whether the observations are somewhat general to the task of spoken digit classification in disregard of the language.

2 Data

The Spoken Arabic Digit Data Set consists of 8,800 samples (10 digits $\times$ 10 repetitions $\times$ (44 males + 44 females)). The data are preprocessed first with pre-emphasized filter $(1 - 0.97Z^{(-1)})$. Then the sound is converted to the 13 most significant Mel Frequency Cepstral Coefficients (MFCCs).

English Free Spoken Digit Dataset (FSDD) is in the .wav sound format. There are 3,000 recordings (10 digits $\times$ 50 repetitions $\times$ 6 speakers). In order to keep the conditions similar as possible, I preprocessed it in the same way as the Arabic dataset.

3 Data Split

As the English version has only 6 speakers I do not aim to estimate well the test classification accuracy. I must split the data by speakers. Because otherwise I believe the classification would be about remembering the speaker's digits, not about the digit recognition. Then having only 6 speakers means, I can use something like 3 for training, 2 for validation and 1 for test split. So this is what I did. Also speaker Theo is much more quite than others and Yweweler has (subjectively speaking) heavier accent. So I controlled these two to not be in the split together. There is a definitely room to focus more on the preprocessing, I will keep it for the future-work.

3.1 Loss and Metric

In modern AI terminology, I chose to train the "logits" instead of class probabilities (in this case the full range [-inf, +inf] is available). Having this outputs, the most common loss for multi-class classification is cross-entropy-loss. Another option would be to use Softmax layer at the output and then use the plain negative log likelihood loss, this I leave for the reader to examine.

As a metric I used the plain accuracy. Accuracy is often misleading, because the classes are usually imbalanced, and also we often care more about certain types of metric. For example in terms of disease detection the True Positive Rate may be more important as we are careful to not misdetect an ill patient. However in this case, all categories are balanced and equally important, so the plain accuracy is the metric.

4 Model Architectures

There will be two competing architectures. One with the vanila RNN layers and the second with LSTM cells. It was already shown that LSTM architectures have advantage over vanilla RNNs when the sequences get longer. However the sequences of MFCCs are between

4.1 Hyper-parameters

In this work I will experiment with

- Number of neurons
- Number of layers
- Optimizer
- Learning rate
- Weight decay
- Dropout
- Number of epochs

while not experimenting with

- Batch size
- Batch normalization
- Activation functions

5 Results

I started with a baseline architecture - only one recurrent layer (RNN/LSTM), last output from the recurrent layer is the output of the network. First optimizer is Adam with learning rate of 0.001 without any weight decay.

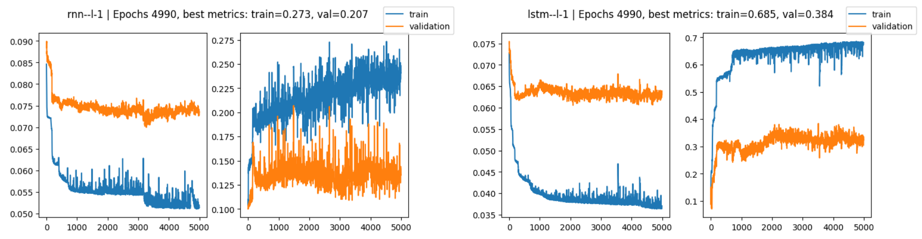

Figure 1: RNN on the left, LSTM on the right (be careful about y-axis values).

This was a little bit unfair to the model with RNN, as the LSTM had more parameters to learn. So I added one fully connected layer at the end, and increased the number of RNN's hidden parameters, so the total number of parameters of the two models is roughly equal. The model with one LSTM layer (with 10 hidden neurons) and one linear layer had 1110 parameters in total. To match this, the model with one RNN layer had 23 hidden neurons and in total had 1114 learnable parameters.

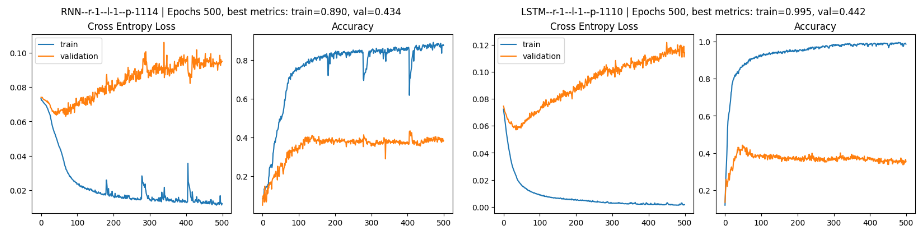

Figure 2: RNN on the left, LSTM on the right (be careful about y-axis values).

Both models suffered from over-fitting early on. It is visible from the uptrend of validation loss from about 50th epoch, see Figure 2.

5.1 Over-fitting

In order to counteract the over-fitting I added one dropout layer between the recurrent and liner layers.

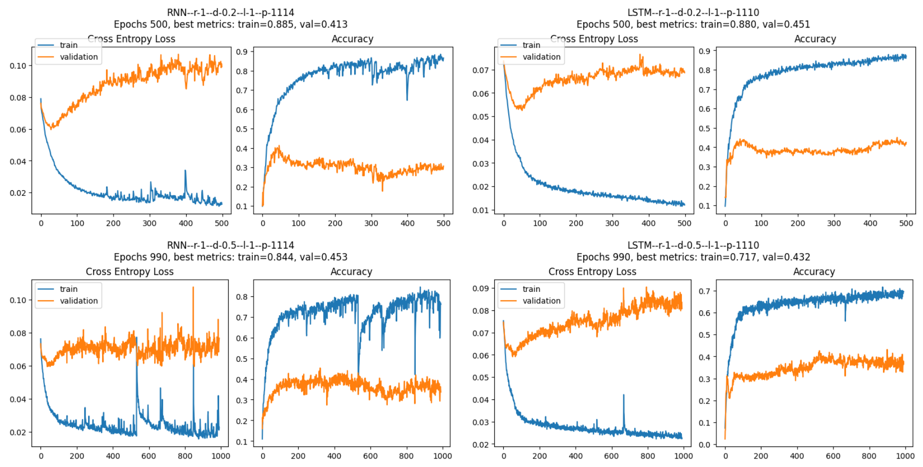

Figure 3: RNN on the left, LSTM on the right (be careful about y-axis values).

The dropout helped, the RNN model worked better with 0.5 dropout probability, the LSTM model with the 0.2 probability. Also when we compare Figure 2 with Figure 3 the validation loss increases less, and the validation accuracy declines less with dropout throughout the charts.

However the models still suffered from over-fitting a lot. This was the time for different optimizations.

5.2 Optimization

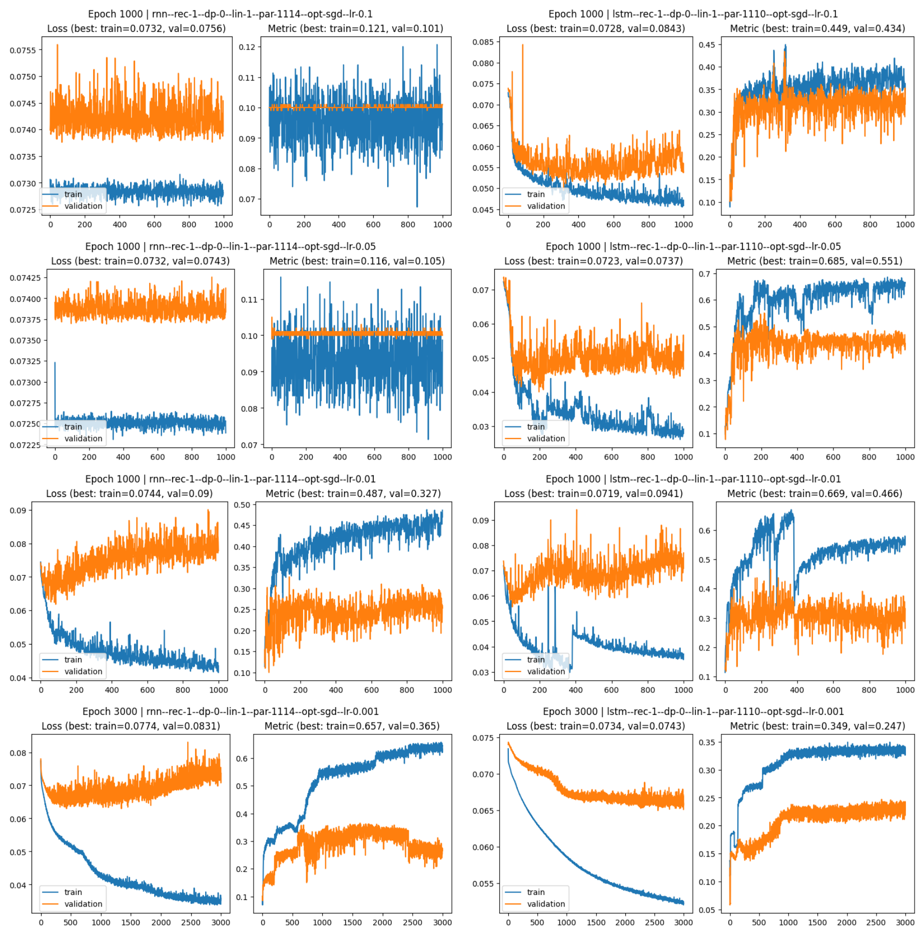

Figure 4: Stochastic Gradient Descent with different learning rates, RNN on the left, LSTM on the right (be careful about both axes values).

SGD didn't come out as a good fit for RNN model. On the other hand a little surprise was the not bad results from LSTM model with bigger learning rates. Especially for the learning rate 0.05 the validation accuracy was stable around 0.4, with a peak of 0.551, which didn't look as an outlier.

Then I tried optimization with weight regularization (or weight decay).

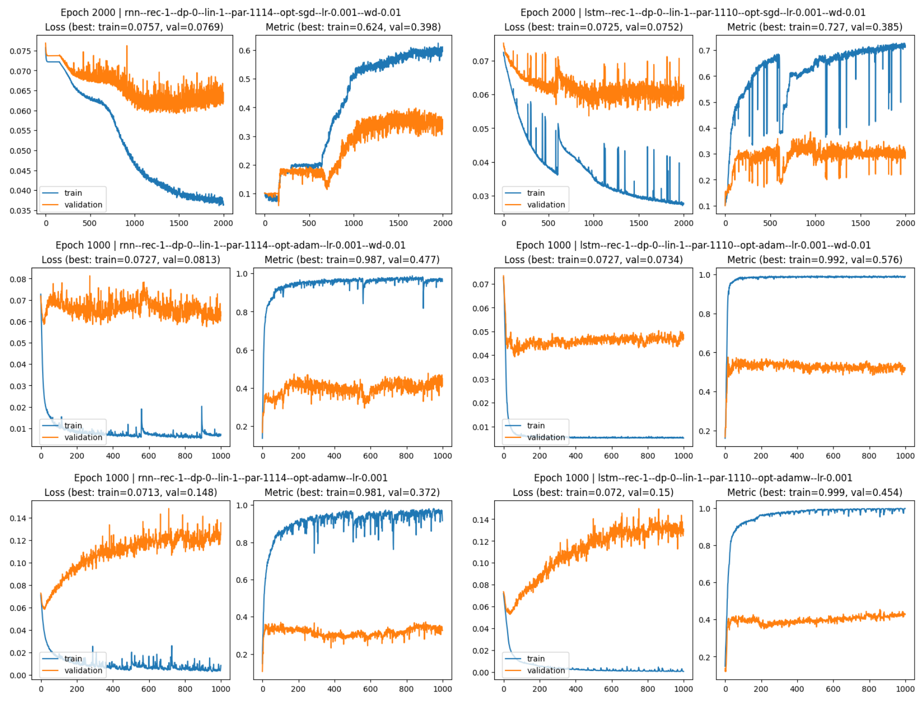

Figure 5: SGD, Adam and AdamW with a weight decay of 0.05, RNN on the left, LSTM on the right (be careful about both axes values).

The weight decay had big impact on the results. Adam with learning rate of 0.001 and weight decay 0.01 stopped the over-fitting for both RNN and LSTM models and also scored the best accuracies for both train and validation sets.

5.3 Increasing Capacity of the Models

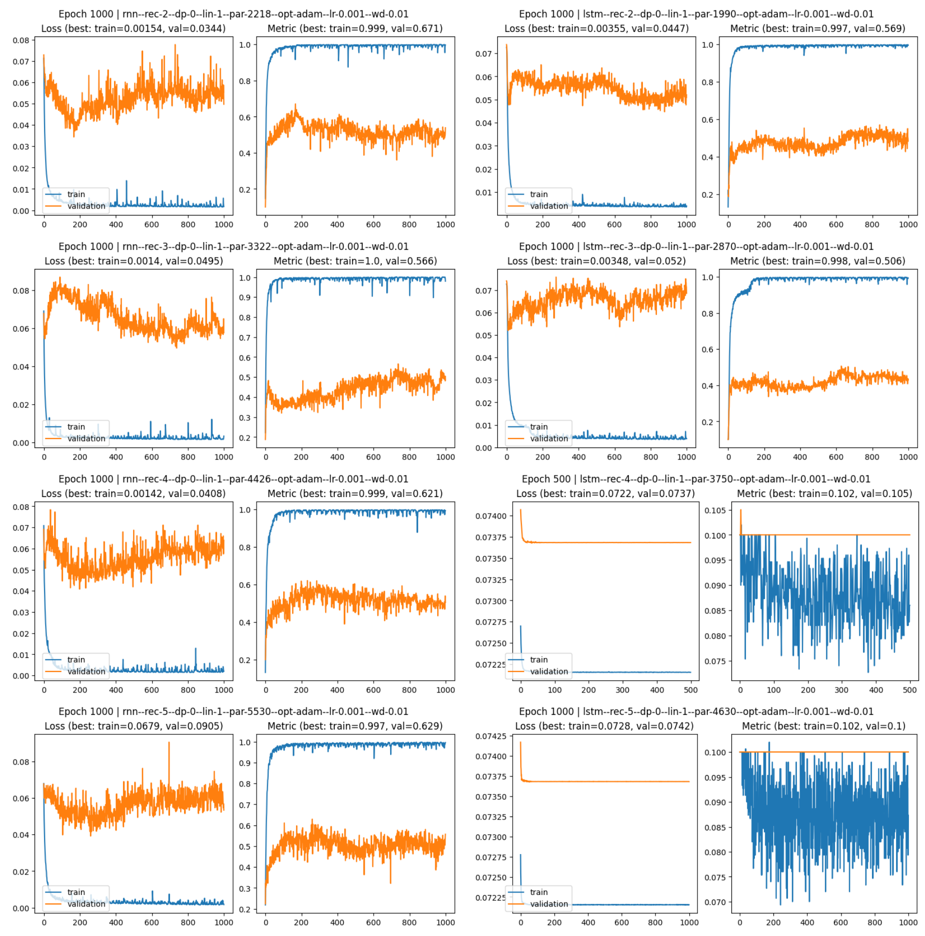

Figure 6: Adding more layers, RNN on the left, LSTM on the right (be careful about both axes values).

Figure 6 shows us, that for both RNN and LSTM models two recurrent layers are better than one, and also better than having more of them. Following charts shows the impact of increasing the size of hidden layers.

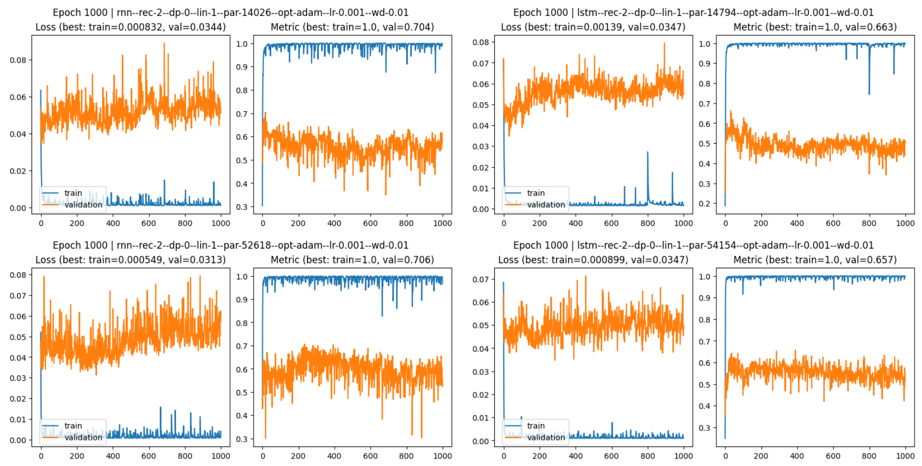

Figure 7: Adding more neurons, RNN on the left, LSTM on the right (be careful about both axes values).

The increase in number of hidden neurons (to 64 for RNN and 32 for LSTM) significantly improved the accuracy. Another doubling didn't help anymore. At this point the validation results showed another signs of over-fitting (the 100% train accuracy isn't something I was aiming for).

This was the time for a second regularization.

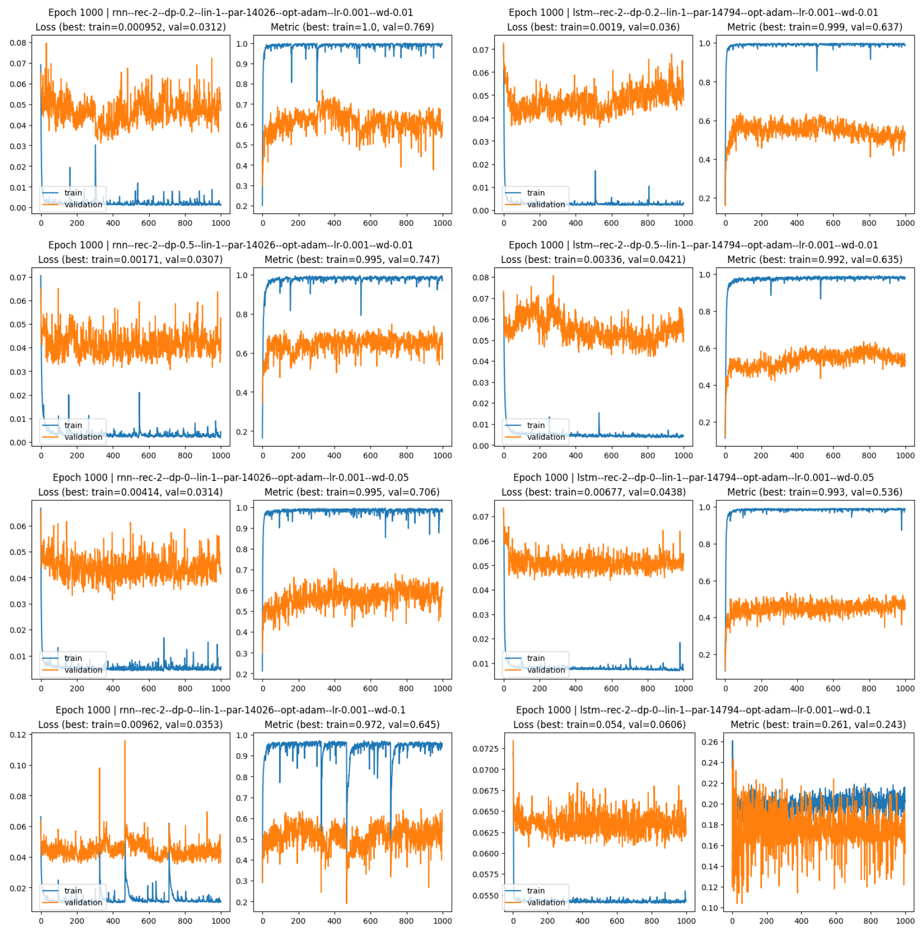

Figure 8: Increasing dropout probabilities and weight decays

The model in top left corner of 9 looked like my final candidate. Then while trying to reproduce the original accuracies I added one more hidden layer with drop out and GELU activation functions in between. Follows the final model overview.

```
RNN(
  (rnn): RNN(13, 64, num_layers=2, batch_first=True, dropout=0.2)
  (gelu1): GELU(approximate='none')
  (drop1): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=64, out_features=32, bias=True)
  (gelu): GELU(approximate='none')
  (drop): Dropout(p=0.2, inplace=False)
  (fc): Linear(in_features=32, out_features=10, bias=True)
)
```

Figure 9: Final Model Architecture.

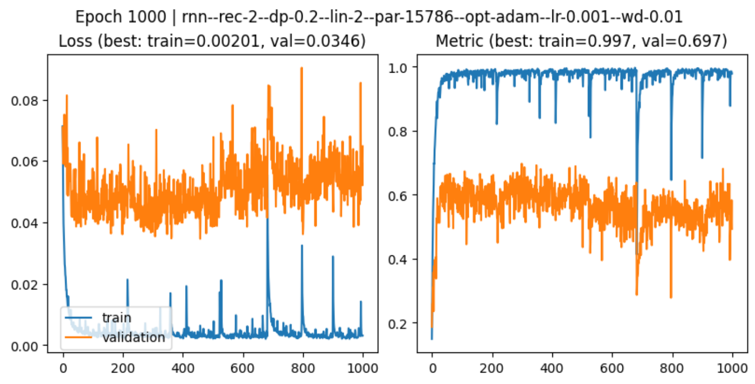

Figure 10: Training the final model on English dataset.

As you can see, the final model didn't score as high as the previous versions. Because the purpose of this sections was not to score the highest number but to choose the best architecture for the task I believed this architecture is a good shot. The result on the test set was

Test Loss of 0.0211 **test accuracy** 0.808.

5.4 Classification of Arabic Spoken Digits

I trained the "Final model" on the Arabic digits dataset.

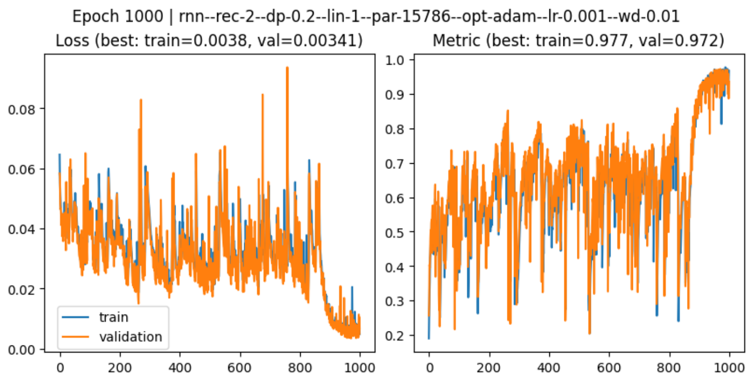

Figure 11: Training the final model on Arabic dataset.

The final model was taken as the one with the best validation accuracy. It's performance on the test set was:
Test Loss of 0.00985 **test accuracy** 0.918.

6 Conclusions

In conclusion, the same architecture chosen to classify well English spoken digits worked even better with Arabic spoken digits. One explanation why the performance with Arabic was better, might be, the Arabic sounds were sampled with higher rate (11,025Hz vs 8,000Hz). Also the Arabic dataset was much more diversified (88 speakers vs 6 speakers). Interesting take away is, the models with LSTM didn't over-perform the model with vanilla RNN. I believe the biggest reason was the preprocessing to MFCC features. The raw sound values would probably fit better the LSTM capabilities. This should be examined in a further research. The single best improvement identified was using the weight decay for optimization.