Deep Learning - Mini-Project I

Filippa Bång, Håvard Bjørnøy
filippa.bang@epfl.ch, havard.bjornoy@epfl.ch
Deep Learning EE-559, EPFL, Switzerland

***Abstract*—Neural networks have demonstrated an impressive classification performance on time-series data the last years with the introduction of network architecture such as convolutional neural networks and recurrent neural networks. However, their versatile nature and abundance of different structures and hyperparameters make the process of creating well-performing models very hard and time-consuming. This paper will address this problem and present a methodology as well as a concrete example following that methodology. The example is classification on an EEG dataset, for which the most suitable neural network turns out to be LSTM with dropout, performing with 0.2734 in error.**

I. Introduction

When designing a neural network there is a wide range of choices that have to be made; architecture, loss function, optimizer and all their hyperparameters. Therefore, there is a precedented need for a generalized methodology of how to navigate all these different decisions. The last decades of progress in the field of neural networks have given the engineers a lot of different structures and methods. When searching for answers to how to decide on neural network architecture there is no agreed-upon milestone paper. The issue is highly dependent on the problem, the dataset, and the skill level of the creator. Many neural network creators use rough guess based on prior experience about networks used on similar problems. Either their own personal experience or others approach to similar problems.

In this paper, we present a broad methodology that aims to incorporate prior knowledge. To illustrate the methodology, it is applied to a real-life classification EEG dataset.

II. Methodology

This is an effort to apply design thinking in a neural network context. When engineering a neural network to solve a problem there is often a trade-off between time spent on implementing a model and the accuracy obtained. Being aware of what the limiting factor is (time, computational power, need for precision) when designing a neural network is an important ingredient in the decision-making process. Here, we present the process as a step-wise process. For each step decide on N different models and implement them. Evaluate each implementation based on classification score and computational time. Keep the model that is most promising and develop it further. Repeat the process until the final model meets your performance requirements. The process could be done in a cyclical manner, however, as there are so many possible choices, it is formulated in a step-wise manner limiting purposes.

In an attempt to minimize dependency on previous decided on features of the model, we suggest to explore the following features in given order:

1) Basic architecture: When investigating basic architectures, we want to explore the different bareboned architectures and assess their potential. Implementing these architectures with an as basic structure as possible, striving to not tinker with default values of hyperparameters. The assumptions are that if a model needs a lot of tinkering with hyperparameters with its most basic non-complex structure it is probably going to be even more sensitive to hyperparameters with a more complex structure.*Examples of different "basic architecture": Linear classification, Feedforward Network, Convolutional network, Gated recurrent unit networks etc.*

2) Further modeling: The section represents the further developing of structural components for the model chosen from the previous step. If time and imagination allow, this section could be split into several steps. The focus is usually components important to get the desired tradeoff between of model complexity and overfitting, while keeping away from excessive tinkering of hyperparameters. *Examples of different elements from "Further modeling": Number of layers, layer width, dropout, batch normalization, etc.*

3) Optimizing: In this step, one can explore loss-functions in combination with different optimizers. Again, if time allows, this section can be split into several steps. For example, hyperparameters for a given model and optimizer. For this step, there exists more research on methods for searching through hyperparameters.

Assuming that this methodology leads to good models, we will now try to develop a neural network for a specific classification problem.

III. Dataset

The dataset that will be used is an Electroencephalography (EEG) dataset [1]. Electroencephalography is a method for measuring the amount of local brain activity. The measurement is done by having several sensors on the subjects head at different positions. It this case there are 28 different sensors for each subject.

The goal is to predict the laterality of upcoming finger movements (left vs. right hand) 130 ms before key-press. This is a standard two-class classification problem.

The dataset is composed of 316 training recordings, and 100 test recordings, each composed of 28 EEG channels sampled at 100hz and 1000hz for 0.5s.
The dataset has an explicit temporal nature and an implicit spacial 3D-structure in the way the sensors are set up.
In this paper, the train/validation/test split of samples will be 253/63/100. The dataset with 100hz will be used. To clarify, each of the datasets then has the dimensions: [number of samples, sequence length(=50), EEG channels (=28)]

IV. Basic Architectures

After assessing our time-constraint and researching the dataset as well as the topic in general it emerges that this might be a very noisy complex dataset with very few samples. The dataset has implicit spatial information one could do feature generation on. But given time-constraints and accuracy requirements simpler models are preferred. It is therefore chosen to implement well known, simple models of which the behavior is known and some models that are well suited in research of temporal data classification, for example (Jozefowicz 2015)[2]. The following models were implemented with pytorch deep learning framework and trained with seed=0 for reproducibility. If not anything else is stated the models have all hyperparameters set to default value. Some of the models flatten the input. That means that it squeezes the EEG channel- and temporal dimensions. Which entails that it treats every channel at the different times as a separate feature, ignoring the relationship between the data points.
The following simple models are tested: Linear, Logistic, Feed forward(FFNN),Convolutional(CNN) and Long Short Term Memory(LSTM). Their specifications are presented listed below. Every model is run with batchsize=32.

A. Linear

- flatten: True
- loss: Mean squared Error
- optimizer: Adam
- learning rate: $3e-5$
- model: Linear(input size)
- commentary: -

B. Logistic

- flatten: True
- loss: Binary crossentropy
- optimizer: Adam
- learning rate: $1e-4$
- model: Linear(input size), softmax()
- commentary: -

C. Feed forward(FFNN)

- flatten: True
- loss: Binary crossentropy
- optimizer: Adam
- learning rate: $1e-4$
- model: Linear(1400, 500), Relu, Dropout(0.2) Linear(500,2), Sigmoid
- commentary: -

D. Convolutional(CNN)

- flatten: False
- loss: Binary crossentropy
- optimizer: Adam
- learning rate: $3e-5$
- model: Conv1d(inputdim, 56, kernelsize=4, stride=4), Conv1d(56, 112, kernelsize=3, stride=3), Conv1d(112, 224, kernelsize=3, stride=1), Conv1d(224, 224, kernelsize=2, stride=1)
- commentary: -

E. Long Short Term Memory(LSTM)

- flatten: False
- loss: Binary crossentropy
- optimizer: Adam
- learning rate: $3e-4$
- model: Linear(input size), Linear
- commentary: ex:hyperparam lr=2

F. Results

Table III presents the number of epochs, training- and validation errors for all models in step 1.

Table I: Number epochs, training- and validation error step 1.

Model	epochs	train error	val error
Linear	1500	0.0520	0.3659
Logistic	1500	0.0277	0.2550
FFNN	34	0.0117	0.3014
CNN	100	0.0000	0.2702
LSTM	200	0.0000	0.1905

In figure 1 the

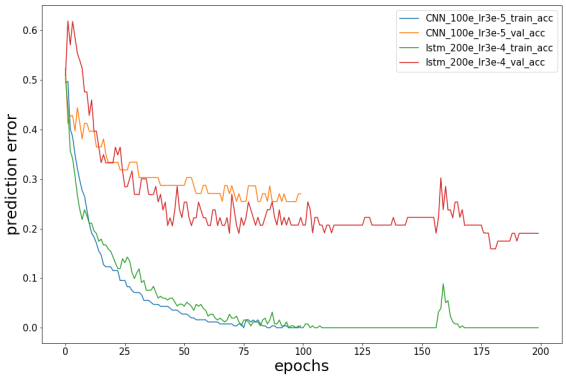

Figure 1: Plotting setup when deciding on models in the process. Shows accuracy, loss and accuracy according to time.

After analyzing the result the CNN and LSTM seemed promising. The logistic and FFNN inability to train temporal information make one think that the might have used up their potential. Though the CNN seems promising and one could experiment with it more, it falls short to LSTM. Because of time-constraint CNN is dropped and LSTM will be the core architecture. Next we will present result from the further modeling of the LSTM architecture.

V. Further modeling

A. Approach

This experimental part we will investigate different width in combination with depth of LSTM network and different rates of dropout too prevent overfitting. The specification of the models is specified in the table
Epcochs is 200 for all models.

B. Results

Table II: Number layers, layer size, training- and validation error step 2.

Layers	Layer size	dropout	train error	val error
1	111	0.4	0.0000	0.1905
1	50	0.4	0.0125	0.3337
1	300	0.4	0.0000	0.2072
2	111	0.4	0.0078	0.2218
1	111	0.5	0.0000	0.1905

None of the modification lead to better result, therefore the desicion was made to keep the original model with 1 layer, 111 recurrent dimension and 0.4 in dropout rate.

VI. Optimizing

Given time-constraint the focus has not been on optimization. We will know see a example of last model of LSTM as it was with SGD compared to Adam which we have used thorughout the process.

A. Results

Table III: Optimizer, loss, training- and validation error step 3.

Optimizer	train error	val error
Adam	0.0000	0.1905
SGD	0.2775	0.5081

The Adam really outperforms the sgd by far shot

VII. Results

The test-accuracy for The final model LSTM with one layer, 111 recurrent dimensions, 0.4 dropout for 200 epochs gave 0.2734 in test error. After the choice of best model, we went back and ran the other algorithms on the test-set. The best algorithm was the LSTM with 300 recurrent nodes with 0.2109 in test error.

VIII. Discussion

The approach led to a not so well generalized algorithm. The discrepancy between validation error and test error might signify that the dataset is very noisy. But also that it is easy to choose wrong models with a very small validation set. It was an alright approach for projects with a strict time-constraint. But for a longer project, it is probably too crude and simplistic. It does not take into account data-augmentation or feature-extraction. Things that probably would be very useful in this problem with a lack of data. But this did not fit well with the step-wise process.
Using cross-validation could definitely help and get rid of some of the uncertainty with choice of model.

IX. Summary

In this exploration, a methodology for developing a neural network is presented and tested. Aiming to classify an EEG dataset, a neural network achieving a 27.34 % test error was developed. Five simple models were tested in step 1, of which the long short-term memory generated best results. In step 2 we experimented with the number of layers, layer size, and dropout rate. None of the modifications gave better results, so it was decided to keep the original model with 1 layer,111 recurrent dimensions and 0.4 in dropout rate. In step 3, Adam was a better optimizer than SGD. Thus, after completing all steps of our methodology, one of the initial models was kept. If time had permitted, it would have been interesting to explore several features and do further modifications to the model to get an even better result.

References

[1] Klaus-Robert Müller and Gabriel Curio. *Data set <self-paced 1s>*. URL: http://www.bbci.de/competition/ii/berlin_desc.html.

[2] Wojciech Zaremba Rafal Jozefowicz and Ilya Sutskever. *An Empirical Exploration of Recurrent Network Architectures*. URL: http://proceedings.mlr.press/v37/jozefowicz15.pdf.