Deep Learning Lab: Assignment #2

Francisco Rivera Valverde
francisco.rivera@students.uni-freiburg.de

***Abstract*—This document employs the HpBandSter software from the University of Freiburg to provide intuition on how to select hyperparameters of a reduced LeNet neural network. Such reduced network consisted of a CNN with 2 convolutional layers activated by ReLu units with a softmax layer for classification. By applying random search, the HpBandSter software recommended to use a learning rate of 34, a filter size of 3x3x57, and a learning rate of 0.14.**

***Index Terms*—Deep Learning, CNN**

I. Introduction

The surge of convolutional neural networks allowed developers to extract high level features of their dataset, on a very similar way as per how neurons interpret data [1]. On the field of computer vision, Convolutional Neural Networks like [2] were the epoch that motivated computer scientist to start employing deep neural network against the traditional object detection techniques.

This document describes the construction of a modified version of the so called LeNet5 Architecture [3]. The implementation consisted of a two convolutional layer with ReLu activation with a softmax layer for classification. Regarding the training, cross entropy loss with stochastic gradient descend was used. The goal of the architecture is to be able to predict handwritten numbers out of the NMist dataset (similar to what [2] used).

The goal of this document is to highlight the influence of hyperparameters on the performance of the net. For that purpose, section II describes the details of the architecture. Section III describes the training setup towards the experiment development. Section IV shows the influence of filter dimensions, learning rates and training hyperparameters on the LeNet performance and section V highlight the most important conclusions.

II. The LeNet architecture

The LeNet Architecture originally consisted of 5 layers, including multiple convolutional networks and several fully connected layers at the end of the network [3]. The implemented architecture, as per described in Figure 1, consist of 2 convolutional layers (16 3x3 filters and a stride of 1), each followed by ReLu activation and a max pooling layer (pool size of 2). After the convolution layers. We added a fully connected layers with 128 units, and a second fully connected layer with with 10 units that would have a softmax output.

Regarding the implementation of this architecture, we employ Tensorflow using the Collaboratory research framework from google. We defined a function that employs the conv2d function to create the convolutional layers, as well as the different tensor variables and placeholders for the system to work. Two functions where provided, being train_and_validate, which actually creates and stores the model using the saver() object, and the test which restore the model and calculate prediction error.

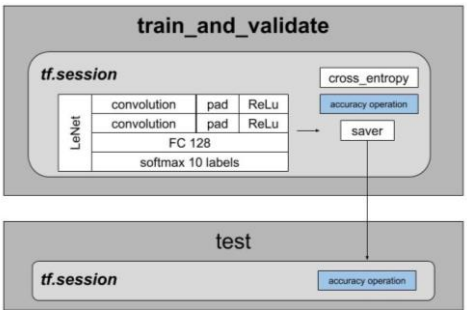

Fig. 1. Architecture of the reduced LeNet architecture. A 28x28 images goes through 2 convulational layers (with max padding) and a fully connected final stage.

Do notice from the Figure 1, that we store a complete model, and then the interface between train_and_validate and test function is the saved model path. So, the test function, needs only to restore the save model and employ the get_tensor_by_name function to query the accuracy operation from the graph, and eventually calculate the error.

III. Training the architecture

Regarding the training, we use the tf.train.GradientDescentOptimizer to implement a SGD optimizer. The loss function employed was cross entropy, taking into account that the output is Softmax. The training function creates the model by itself, trains it and stores it. Because of this flexibility one can employ this function in a foreach loop that varies the method arguments.

The goal is to understand how the learning rate, filter dimensions and batch size affects the test error. On that regard, we would graph the validation error per epoch for the following configurations:

- Learning_rate = {0.1, 0.01, 0.001, 0.0001}
- Filter_size = {1, 3, 5, 7}

This would provide an empirical framework to understand how these parameters affect the model performance (having a fixed size of 128, 25 epochs and 16 depth filters). Then, we employ the random search methodology through the HpBandSter software to get a better intuition on how to select

hyperparameters to achieve a good performance while training the model. The following configuration space was used:

- learning rate ∈ [10 −4 , 10 −1]
- batch size ∈ [16, 128]
- number of filters ∈ [2^3 , 2^6]
- filter size ∈ {3, 5}

IV. Results

The impact of changing the learning rate as per described in the training section can be seen in Figure 2. One can notice that the small learning rate of 0.0001 cannot allow the model to reach an acceptable validation error as it takes much time for the model to learn. Recommended learning rates are 0.01 or higher for this model.

Similarly, the impact of changing the filter size can be appreciated in Figure 3. Small filter makes it problematic for the network to extract low level features and hence train the network on a more generical way. Filter of size 3x3 appears to be a good tradeoof between memory and performance.

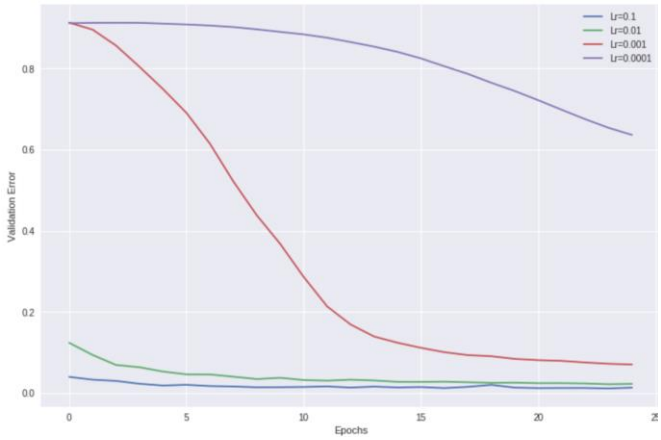

Fig. 2. Validation error against different learning rates. 25 epochs registered the error of inputting a validation set of 10000 images of 28x28 size against a training set of 50000 images.

Following this comparison, the results of performing random search, with the configuration space presented on Section IV, can be seen in Figure 4. With the best configuration found by random search, being a learning rate of 0.14 filter size of 3x3x29, batch_size of 63 and 6 epochs, we can obtain a test error of 0.009. It is important to remark that successive runs yield different results due to the randomness of the algorithm.

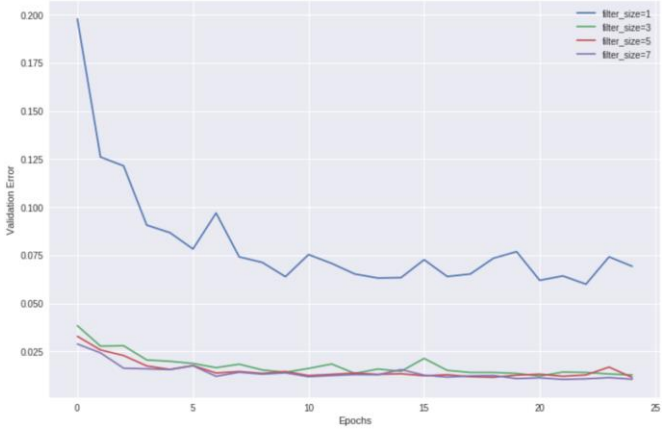

Fig. 3. Validation error against different filter sizes. 25 epochs registered the error of inputting a validation set of 10000 images of 28x28 size against a training set of 50000 images.

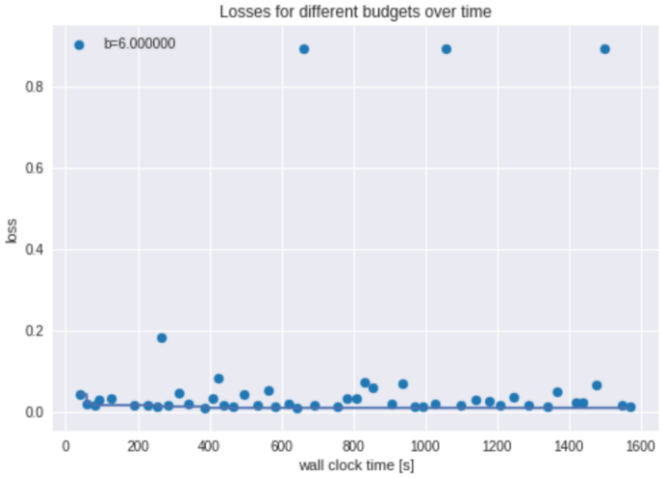

Fig. 4. Loss functions for different configurations of the hyperparameters. The line shows how the random search achieved consecutive configurations that reduced loss. Such best configuration is:
{'batch_size': 63, 'filter_size': '3', 'lr': 0.25180174895908364, 'num_filters': 29}

V. Conclusion

A good hyperparameter configuration can be led to an easier model convergence, and the same is true for a bad hyperparameter configuration causing the model to diverge. Proper selection of the hyperparameters is critical for a good model performance, and in this document, we tried both an iterative way (varying the learning rate and filter size) and a structured method as random search.

With the outcome of random search, the model yielded a good performance over validation data even from young epochs.

References

[1] Cecotti, Hubert, and Axel Graser. "Convolutional neural networks for P300 detection with application to brain-computer interfaces." IEEE transactions on pattern analysis and machine intelligence 33.3 (2011): 433-445.

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[3] LeCun, Yann. "LeNet-5, convolutional neural networks." URL: http://yann. lecun. com/exdb/lenet (2015): 20.

[4] GoodFellow, Ian, et al. Deep learning. Cambridge: MIT press, 2016.