Deep Learning Project 1
- Classification, weight sharing, auxiliary losses -

Deep Learning (EE-559)

École Polytechnique Fédérale de Lausanne

Salim Ben Ghorbel
salim.benghorbel@epfl.ch

Ahmed Ben Haj Yahia
ahmed.benhajyahia@epfl.ch

Nour Ghribi
nour.ghribi@epfl.ch

***Abstract*—This report details our approach to compare two digits of a pair of two-channel images from the MNIST data. The goal of this project is to compare the accuracy of different architectures and assess the performance improvement that can be achieved through models with shared weights and the use of an auxiliary loss. The performance of each of these different models will be estimated on a test data through 10 rounds where both data and weight initialization are randomized at each training.**

I. Introduction

A. Data

The data comes from the MNIST dataset but instead of the original size of 28x28, we work with a computationally less expensive version of size 14x14 in order to run our models faster. We did not perform any additional preprocessing.
The dataset consists of a randomly sampled pair of images put into two different channels, thus the shape of the data is 2x14x14. The dataset also provides labels for each channel's true digit and a Boolean value that indicates if the first channel's digit is lesser or equal to the second.

B. Approach

The goal of this project is to implement a deep network such that, given as input a series of 2×14×14 tensor, corresponding to pairs of digits each in a 14 × 14 grayscale images, it predicts which one is greater. In order to decide whether the first channel is lesser or equal to the second channel, we tried different types of architectures.
We started with a simple **shallow model** that consists of only two fully connected layers with ReLU activation in between. Then we constructed a **deep model without using weight sharing**. It consists of two convolutional networks defined as separate sequential models, and of a fully connected layer model which takes the output of the two models and returns a binary output. The two convolutional networks are used to predict the digits from the input images and their weights are defined separately.
Finally, we constructed a **deep model using weight sharing**. It consists of one convolutional network, defined as a sequential model, and of a fully connected layer model which takes the output of the the convolutional network and returns a binary output. The convolutional network is used to predict the digits from the input images. Weight sharing is guaranteed by using one ConvNet instead of two.
The architecture of these two deep models are inspired from the models presented in class slides. An auxiliary loss is defined, taking into account the classification of the digits, i.e. the output of the two networks. A weight is defined such that the final loss is a linear combination of the auxiliary and main losses.

II. First Model : ShallowModel

This can be considered as our baseline model as it is rather a straightforward model.

A. Architecture Description

This model consists of two separate fully connected layers. The two channels are first flattened and passed to 14*14*2 neurons of the first layer. As for the intermediate layer, we used Rectified Linear unit (ReLu) activation function. The second fully connected layer returns a binary output indicating whether first channel is lesser or equal than the second one. The loss is computed using CrossEntropy.

B. Hyperparameter tuning

Hyperparameters of this model consisting of:

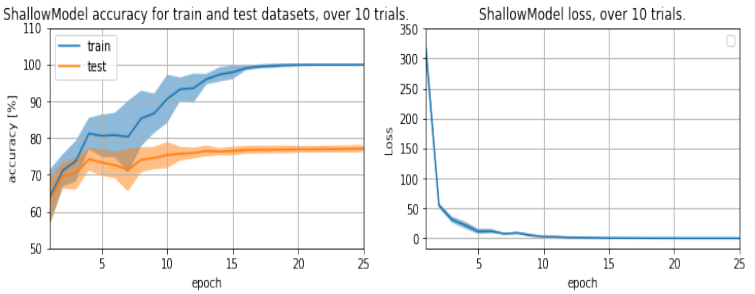
- *learning_rate* ∈ [1e-4,5e-4,6e-4,7e-4,8e-4,9e-4,1e-3,2e-3,3e-3,4e-3,5e-3,1e-2]
- *nb_hiddens* ∈ [100,300,500]

were tuned using a simple grid search algorithm.
This gave us: *learning_rate* = 7e-4 and *nb_hiddens* = 500 for ADAM optimizer during training in 25 epochs.

C. Model performance

Considering the shallowness of our model, its performance was a little bit surprising with 78.5% of test accuracy.

You can see below a plot of the model's mean value of the test accuracy and the training loss over the 10 rounds.



As we can see here on the graph, this model's highest accuracy is a maximum of 78% The training accuracy gets quickly to 100% in just 5 epochs and the variance of this model is low.

III. Second Model : DeepModel without weight sharing

A. Architecture Description

Our second model is a deep network without weight sharing. It consists of two identical convolutional networks, one for each digit image, defined as separate sequential models. Each one consists of:

1) Convolution layer having 32 channels_out with kernel size 3
2) Max pooling layer with kernel size 2
3) ReLu activation function
4) Convolution layer having 32 channels_in and 32 channels_out, with kernel size 3
5) Max pooling layer with kernel size 2
6) ReLu activation function
7) Convolution layer having 32 channels_in and 64 channels_out, with kernel size 2
8) ReLu activation function
9) Flatten layer to flatten the output
10) Fully connected layer with 64 neurons
11) ReLu activation function
12) Fully connected layer with output dimension 10

Then the output of these two convolutional networks are fed to a ReLu activation function and used as inputs to our fully connected layer with 20 neurons. This results in a binary output.

B. Loss calculation:

An auxiliary loss is defined, taking into account the output of the two convolutional networks. A weight is defined such that the final loss is a linear combination of the auxiliary and main losses:

$$\text{loss} = (1 - \text{aux\_loss}) * \text{target\_loss} + \text{aux\_loss} * (\frac{\text{loss\_cn1\_im1} + \text{loss\_cn2\_im2}}{2})$$

We used the CrossEntropy creterion for all the losses.

C. Hyperparameter tuning

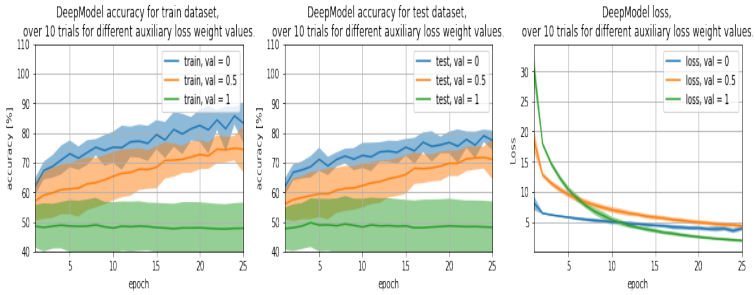
In addition to the hyperparameters present in the ShallowModel, this model's other hyperparameters consist of:

- *auxiliary loss weight* $\in$ [0,0.25,0.5,0.75,1]

They were tuned using a simple grid search algorithm and this gave us: *learning_rate* = 5e-3 and *nb_hiddens* = 100 for ADAM optimizer during training in 25 epochs.

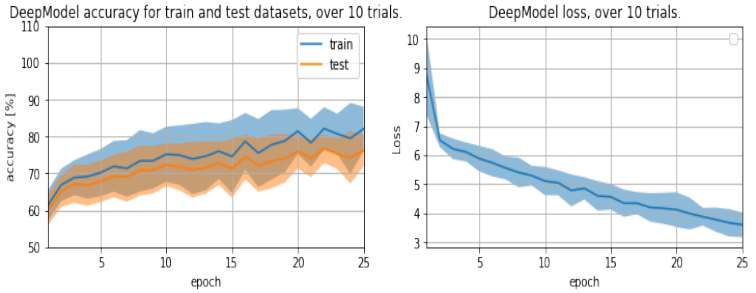
D. Impact of the auxiliary loss weight

Thanks to our architecture, we are able to set a variable weight to each of the auxiliary and the original losses. The total loss hence becomes a linear combination of the two. The plot below shows the impact of the auxiliary loss weight on the train and test accuracy for the DeepModel, using a mini batch size of 100. Interestingly the model performed better with a zero weight to the auxiliary loss, i.e. using only the final classification weight. However, for a mini batch size of 1. Best accuracy was obtained when setting the auxiliary loss weight to 0.5.



E. Model Performance:

This model performed considerably better than our ShallowModel achieving 92.2% of test accuracy with the previously chosen hyperparameters. We can see below the plot of our model's mean value of the test accuracy and training loss. These estimation are performed over 10 rounds where both data and weight initialization are randomized and are different for each convolutional networks.



IV. Third Model : DeepModel with weight sharing

A. Architecture Description

Our third and final model is a deep network with weight sharing. It consists of one convolutional network, used to predict the digits from the input images and defined as a sequential model with:

1) Convolution layer having 32 channels_out with kernel size 3
2) Max pooling layer with kernel size 2
3) ReLu activation function
4) Convolution layer having 32 channels_in and 32 channels_out, with kernel size 3
5) Max pooling layer with kernel size 2
6) ReLu activation function
7) Convolution layer having 32 channels_in and 64 channels_out, with kernel size 2
8) ReLu activation function
9) Flatten layer to flatten the output
10) Fully connected layer with 64 neurons
11) ReLu activation function

12) Fully connected layer with output dimension 10

Then the output of this convolutional network is fed to a ReLu activation function and used as input to our fully connected layer with 20 neurons. This results in a binary output. Weight sharing is guaranteed by using one Convolutional Network insteaod of two.

B. *Loss calculation:*

An auxiliary loss is defined, taking into account the output of the convolutional network. A weight is defined such that the final loss is a linear combination of the auxiliary and main loss:

$$\text{loss} = (1 - \text{aux\_loss}) * \text{target\_loss} + \text{aux\_loss} * (\frac{\text{loss\_cn\_im1} + \text{loss\_cn\_im2}}{2})$$

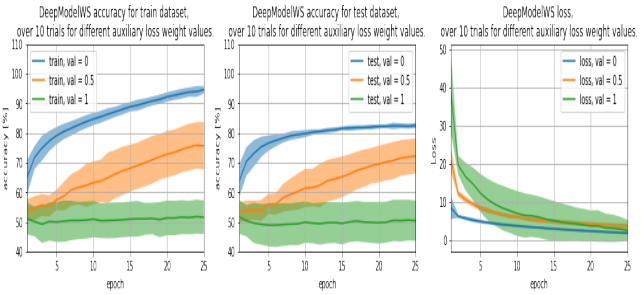
We used the CrossEntropy creterion for all the losses.

C. *Hyperparameter tuning*

The same hyperparameters of the previous model were tuned using grid search algorithm and this gave us: *learning_rate* = 5e-3 and *nb_hiddens* = 500 for ADAM optimizer during training in 25 epochs.

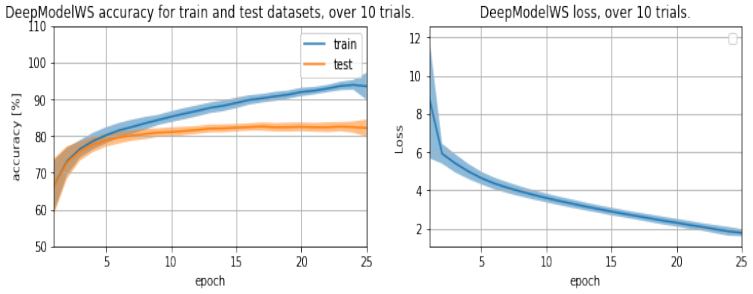
D. *Impact of the auxiliary loss weight*

The plot below shows the impact of the auxiliary loss weight on the train and test accuracy for the DeepModelWS, using a mini batch size of 100. Similarly to the case where no weight sharing is used, this model performed better with a zero weight to the auxiliary loss, i.e. using only the final classification weight. However, for a mini batch size of 1. Best accuracy was obtained when setting the auxiliary loss weight to 0.5.



E. *Model Performance:*

This model performed considerably better than our ShallowModel, and even better that the deep model without weight sharing, achieving 94.3% of test accuracy with the previously chosen hyperparameters. We can see below the plot of our model's mean value of the test accuracy and training loss. These estimation are performed over 10 rounds where both data and weight initialization are randomized and are different for each convolutional networks.



V. Summary:

As a summary and from looking at the graph in Fig. 1, we can compare and understand the effect of the weight sharing and auxiliary.

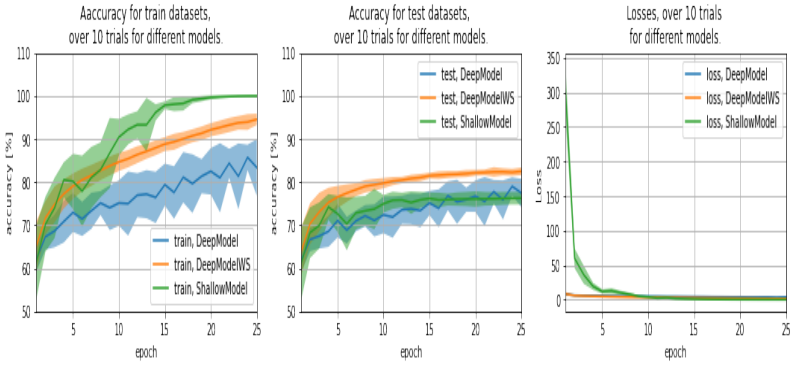


Fig. 1. Models comparison

Thanks to the exploration of diverse parameters and after analyzing their impact on the performance of our networks, we managed to obtain a rough image on how to better fine-tune deep learning models in general, especially after looking at the weight sharing and the definition of an auxiliary loss. In one hand, we performed a rough analysis on the effect of the auxiliary loss and we concluded that its impact is also related to other parameters, namely the batch size. On the other hand, with the effect of enabling weight sharing, the model gained 2.1% in terms of accuracy and considerably reduced the variance and thus present more stable results.

References

[1] *Convolutional Neural Networks and Image Classification* [online] Available at: https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

[2] *MaxPool vs AvgPool* [online] Available at: https://iq.opengenus.org/maxpool-vs-avgpool/