EPFL

EE-559: Deep Learning mini-project 1

Melina CHRYSANTHOU: melina.chrysanthou@epfl.ch
Lorenzo GERMINI: lorenzo.germini@epfl.ch
May 28, 2021

***Abstract*—The implementation in a variety of architectures of methods such as weights sharing and auxiliary loss, has been an important tool in the fast-paced rate of improvement in the area of deep neural networks. The aim of this report is to investigate the impact of weight sharing or auxiliary losses on the performance in a digit image classification task.**

I. INTRODUCTION

This projects aims at testing different architectures to compare two digits visible in a two-channel image. More precisely the purpose is to asses the performance improvement that can be achieved with the use of weight sharing or the use of an auxiliary losses.
The goal is to get a series of 2×14×14 tensors, corresponding to pairs of 14 × 14 gray-scale images and with the deep networks that we will implement, predict for each pair if the first digit is lesser or equal to the second. The pairs are generated with digit images from the MNIST database [1].

II. METHODOLOGY AND MODELS

Python 3.7.7 was used as the programming environment. The project was implemented with PyTorch code only. The file dlc_practical_prologue provided in the course was used to generate the data sets. The packages shown below were used with the last two used only for the visualization of the results.

- PyTorch 1.8.1
- Numpy 1.17.0
- Matplotlib 3.2.2

A. Data preprocessing

After having randomly generated generated a training and test set of respectively 1000 number of samples each, the train_input and test_input tensors (both of size 1000 x 2 x 14 x 14) were normalized trough a standard scaler with the mean and standard deviation of the train_input tensor.

B. Implemented methods

In order to improve our model's classification performance the following methods were used to construct the neural networks layers.

1) Batch normalization: Shifts and rescales according to the mean and variance estimated on the batch. In principle similar to the normalization of the inputs, this technique introduces orthogonality between layers in order to avoid shifting distributions in activations as the parameters in earlier layers are updated. This method was used to stabilize the network and make the learning algorithm converge faster.

2) Max pooling: Computes the maximum values over non-overlapping blocks. It reduces the computational cost by reducing the number of parameters to learn and provide basic translation invariance to the internal representation.

C. Loss functions and optimisation

As the main purpose of this project is to tackle a digit classification problem the following functions were used to train the model.

1) Cross entropy: A measure of the difference between two probability distributions for a given random variable or set of events. An important aspect of this method is that cross entropy loss penalizes heavily the predictions that are confident but wrong. The main loss proceeds to apply the loss function between the output and the training targets.

2) Auxiliary losses: Losses introduced when building multi-input and multi-output models. In this case, for a pair of gray-scale images, it could generate two auxiliary losses. The auxiliary losses are generated between the two auxiliar outputs and the training class labels of the two digits. Auxiliary losses can optimize both the learning process and the parameters before the layer generates any auxiliary loss at all.

3) Adam: Adam has been used as the optimization algorithm.

D. Network structures

To asses the impact of weight sharing and the use of auxiliary losses three different neural network architectures were chosen.

1) Baseline ConvNet model: For the baseline model we built a ConvNet similar to the LeNet-5 original architecture [2]. Our network receives in input a 2 x 14 x 14 tensor corresponding to pairs of 14 x 14 grey scale images. The model is composed of a first convolutional layer of 32 channels and a kernel of size 3, a batch normalization layer, a Relu non linearity and a pooling layer of type Max pool of kernel 2 and stride 2. Subsequently we have a second convolutional layer of 64 channels with a kernel size of 3, a batch normalization layer, and a Relu non linearity and a pooling layer of type Max pool of kernel 2 and stride 2. We have then three fully connected layers with 128, 90 and 2 hidden layers with in between them two additional batch normalization layers followed by two Relu activations. For the position of the batch normalization layers we chose to follow the methods of the original paper [3] by placing it before the activation layers. We also explored

its placement after the activation layers but the conventional positioning of batch normalization yielded better results and was therefore kept. In general the intuition compared to the original architecture was to have a deeper network in the feature extractor part with substantially higher number of filters in order to try to better account for more complex feature in the digits. On the other hand batch normalization was used to try reduce the variance which can be seen as a representative of overfitting to the dataset. Only the main loss was used for training. In Figure 1 we can see a visualization with some architecture details.

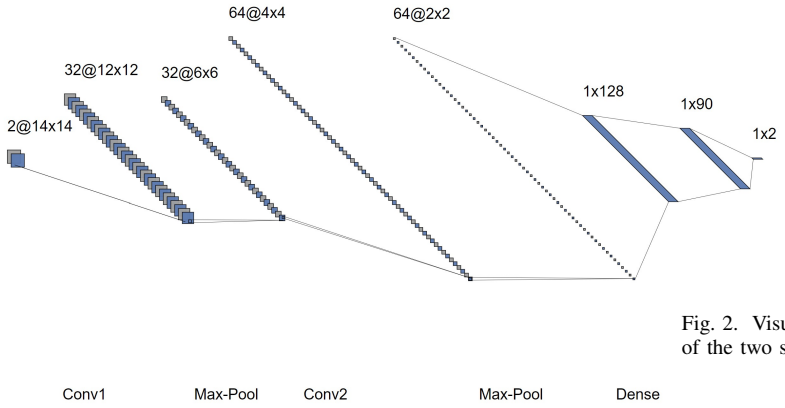

Fig. 1. Architecture of the baseline ConvNet. Batch normalization and Relu activations not shown.

2) Siamese model: The second architecture is a siamese-like structure [4] where each of the two branches contains one sub-network with basically the same structure as our baseline ConvNet model. Each sub-network receives as input just one gray-scale image therefore a tensor of shape 1 x 14 x 14 and have the same weights as its counterpart. Their individual 10 output units (instead of the final 2 output units in the baseline ConvNet) are linked by two fully connected layers with a input layer of 20 units and respectively a hidden layer of 90 units and an output layer of 2 units as shown in Figure 2. Again there is a batch normalization layer followed by a relu activation between the hidden and the output layer. This structure was built to show the impact of weight sharing on the final classification performance. Only the main loss was used for the training. Only the main loss was used for training.

3) Siamese model with auxiliary losses: The third model combines the implementation of weight sharing and the use of auxiliary losses. This means that beside the loss of the predicted target, in the loss function we also include the loss of predicted classes. This improves the learning process and follows the equation below:

$$Total\,loss = Main\,loss + 2\,Auxiliary\,losses$$

The two auxiliary losses are computed from the two parallel auxiliary outputs, which are the outputs of the last fully connected layers of dimension 10 of each branch, from before the Relu activation is applied. Again these losses are derived from those two outputs and the 10 classes of the input digits.

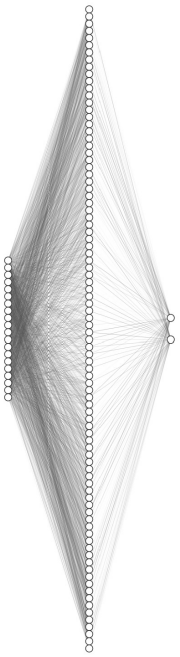

Fig. 2. Visualization of the fully connected layers placed after the merging of the two sub-networks

The idea is, as suggested in the course, is to try to take advantage of the fact that the two auxiliary estimate the digit classification performance of the two images.

III. Models training/evaluation

A. Hyper-parameter tuning

Before starting the training process of each model, the best practice is to select the hyper-parameters that will make the models more efficient on the provided data. We ran a grid search on the following identified parameters of interest shown in Table I: number of training epochs, mini batch sizes and the learning rate for the optimizer. This was done over a number of 10 rounds for each hyper-parameters combination. All the other parameters included the parameters of the architecture are kept stable. A more advanced approach would have been to run a cross validation over a number of randomly generated datasets. However, given the additional and heavy computation required, we decided not to perform it and just pursue the aforementioned approach. According to our grid search results 50, 50, 0.1; 50, 100, 0.01; 50, 100, 0.01 were the best number of epochs, mini batch size and learning rate for respectively the baseline, siamese and siamese with auxiliary losses architectures. The

TABLE I
HYPERPARAMETERS SEARCH SPACES

Model	Hyperparameter	Distribution	Range
Base	Nb_epochs	Specific	25, 50, 100, 150
	Mini_Batch_size	Specific	25, 50, 100
	Learning rate	Specific	1e-3, 1e-3*5, 1e-2, 1e-1
WS, WS_Aux	Nb_epochs	Specific	25, 50, 100
	Mini_Batch_size	Specific	50, 100
	Learning rate	Specific	1e-3, 1e-3*5, 1e-2, 1e-1

best accuracy as well as standard deviation values were considered for the best selection. We observe that for the both the iterations of siamese architectures lower learning rate and higher batch size were found to be better compared to the baseline best hyper-parameters We also considered during the training to apply regularization like a l2 norm regularization to penalize at each iteration the loss function more, thus relatively reducing the weights and reducing the potential overfitting. However, none of our model was showing great sign of overfitting and therefore we decided not to implement it.

IV. Results

Once the optimal hyper-parameters were selected and the neural networks models trained, predictions were made based on the test dataset. The source of randomness were tried to be kept to a minimum and compared to the optimal values in the grid searches the final results were very similar. All of the computation was performed on our machine. Performance estimates were obtained trough 15 rounds. As we can see in Figure 3 and Figure 4 our baseline architecture achieved a average test error rate 0.1835 with std of 0.0066. We achieved significant improvement in the siamese architecture with weight sharing implementation with a average test error rate of 0.1108 with std 0.0057. An additional performance boost was obtained with our final architecture with the use of auxiliary losses implementation with a remarkable average test error rate of 0.0359 with std of 0.0058.

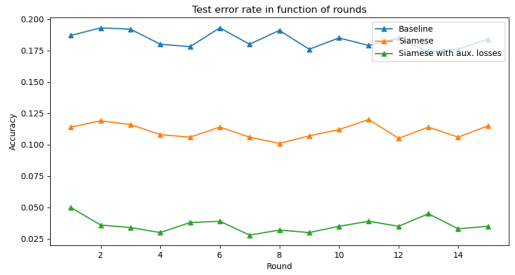

Fig. 3. Visualization of the target error rate evolution over the 15 rounds for each model.

The performance results show that the baseline ConvNet has the poorer quality of prediction for this task which may be despite having just a slightly lower number of trainable parameters of the others two architectures. The baseline We observe that the relative improvement in the error rate from the first to the second and second to the third are very similar at about 0.7-0.8. We also observe that while we test error rate ameliorate significantly also the standard deviations for all the rounds even gets better compared to the baseline case. This suggests that the two techniques also act as a form of regularization. We should keep mind that the number of samples in the sample were relatively small and also the only 15 rounds but nevertheless the achieved bias without increasing by much the variance remains rather satisfactory. Without those and computational constraints more wide and complex hyper-parameters searches may have been investigated. Deeper network may be explored in the case of similar task but with significantly larger datasets.

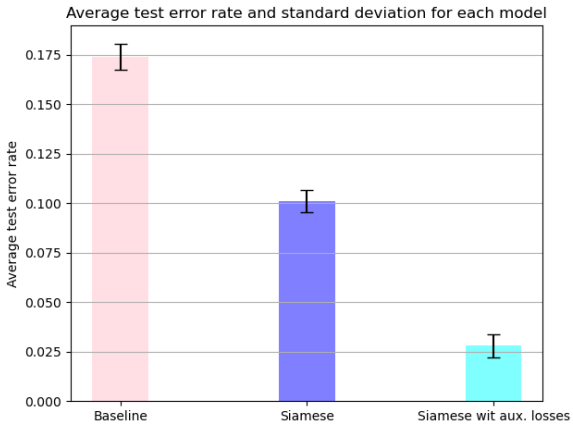

Fig. 4. Bar plot with mean error test rate and corresponding standard deviation for each model.

V. Summary

The aim of the project was to asses the impact of weight sharing and help of auxiliary losses in a digit task classification. In general we see that adding a weight sharing implementation and subsequently the use of auxiliary losses to a baseline model, the classification performance related to the predictions of the deep networks improve significantly. In particular with the chosen architectures which leverage weight sharing with a siamese structure and the same structure with the implementation of two auxiliary losses added to the main one we obtain in each case similar marginal improvement relatively to the baseline and the siamese with main loss only respectively. The significant improved test error rate does not seem not at a cost of the variance

References

[1] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 12 1998.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[4] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2. Lille, 2015.