Project 1
Classification, Weight Sharing, Auxiliary Loss

Marie Anselmet, Héloïse Monnet, Jasmine Nguyen Duc
Deep Learning EE-559, EPFL, Switzerland

***Abstract*—Neural networks, especially with the integration of convolutional layers to their architecture, perform impressively well on image classification tasks. Nonetheless, reaching strong performances by training such structures is still a complex and time-consuming problem. This paper explains the decision-making process that led to a final digit comparison error of 2.04% on a reduced MNIST dataset. Comparing different models, this paper also assesses the benefits of introducing an auxiliary loss and weight sharing on the final performance.**

I. Introduction

Even if neural networks had already begun to reveal their strong potential to perform some classification tasks in the 90s, their continuous improvement during the last years have made them even more promising. Especially, the introduction of convolutional neural networks (CNN) in networks architecture performs impressively well for image recognition and classification tasks. Nonetheless, the plethora of possible architectures, loss criteria, optimizers and hyper-parameters to fine tune still make the optimization of such structures a challenge. In this paper, the performance of a basic fully connected neural network (FNN) was first evaluated. Then, a CNN inspired of the LeNet model [1] was implemented and improved, testing different activation functions, optimizers and learning rates. Also, the performance improvement that can be achieved by weight sharing and the introduction of an auxiliary loss to the main training objective were assessed.

II. Dataset

To illustrate this, different network architectures were trained to perform a digit comparison task. More precisely, the implemented models were trained to predict if the first hand-written digit was lesser or equal to the second, given two images of a lower resolution version of the MNIST dataset. Indeed, the use of 14x14 instead of 28x28 grayscale images enabled to run experiments at a reasonable time scale without GPU. Each data sample consists of a randomly sampled pair of images, and was implemented in PyTorch as a 2x14x14 tensor corresponding to the two digits to compare. The experiments were done on normalized data, randomly split into 1000 samples for training and 1000 samples for testing.

III. Models and Results

All the models described in this paper were trained during 25 epochs with Cross Entropy Loss as criterion, very adapted for classification purposes. A batch size of 100 samples was used during the training process. Performance results (i.e mean and standard deviation of the digit comparison error) provided below were estimated through 10 rounds for each architecture. Data was shuffled and weight initialization was randomized at the beginning of each round.

A. Choosing the network architecture

The following models were all implemented with Rectified Linear Unit (ReLU) as activation function. They were trained with Stochastic Gradient Descent (SGD) as optimizer, with a learning rate of 0.1.

FNN model: The first model implemented was a fully connected neural network with three linear layers for the digit recognition task and two linear layers for the digit comparison task. The model was trained with and without weight sharing (WS). In addition, the introduction of an auxiliary loss (AL) to help the training of the main objective was tested. The resulting digit comparison performances are summarized in table I.

TABLE I: Digit comparison error of the FNN model.

	Mean (%)	**Standard Deviation (%)**
FNN	52.99	11.88
FNN + WS	43.52	16.07
FNN + AL	7.93	0.65
FNN + WS + AL	6.44	0.71

It can be seen that with the most basic model (ie without weight sharing nor auxiliary loss), the model performance is very bad, with a mean digit comparison error of 52.99%. Moreover, the performance results are very variable, as a standard deviation of 11.88% was obtained. Weight sharing improved the performance of the model (mean error of 43.52%) but worsened the variability of its results (standard deviation of 16.07%). The introduction of an auxiliary loss which takes into account the digit recognition error, drastically reduced the mean digit comparison error to 7.93% and its standard deviation to 0.65%. The really small standard deviation value obtained at this point enabled to conclude with enough confidence that the introduction of an auxiliary loss to the main training objective drastically improves the performance of this rudimentary model. The smallest mean error value of 6.44% was obtained with the use of both auxiliary loss and weight sharing.

CNN LeNet-like model: Then, it was decided to implement a CNN based on the known well-performing LeNet model architecture [1]. Two convolutional layers and two linear layers were used for the digit recognition task and as before, two linear layers were used for the digit comparison task. The kernel sizes of the convolutional layers were adjusted to the resolution of the images of this dataset and thus reduced to 3x3. Again, the model was trained with and without weight sharing

(WS) and the use of an auxiliary loss (AL). The obtained performances are summarized in table II below.

TABLE II: Digit comparison error of the CNN LeNet-like model.

	Mean (%)	Standard Deviation (%)
CNN	49.46	15.66
CNN + WS	45.31	17.70
CNN + AL	5.79	4.66
CNN + WS + AL	3.98	1.22

At this stage, it was clear that the use of a convolutional architecture improves the digit comparison performance. Indeed, with the use of both weight sharing and an auxiliary loss, a mean digit comparison error of 3.98% was obtained, so almost 2.5% less than what was obtained with the FNN model. However, the standard deviation values were bigger for the CNN model, demonstrating a higher variability of its performance results. But these values were still small enough for the comparison of the mean error between the two architectures to make sense.

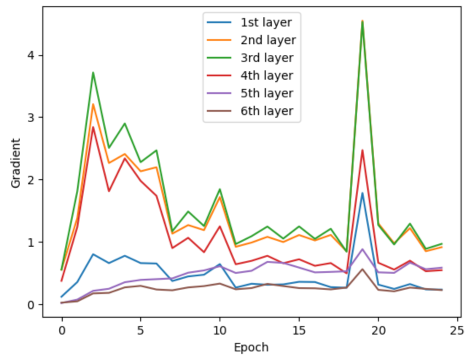
CNN LeNet-like model with batch normalization: As a third step, batch normalization (BN) was added to the previous model in order to improve the performance and the stability of the CNN. Indeed, batch normalization [2] forces the activations' first and second order moments, so that the following layers do not need to adapt to their drift. Thus, batch normalization was added after each of the two convolutional layers of the model. The architecture of the resulting model is detailed in figure 3 in the Appendix.

TABLE III: Digit comparison error of the CNN LeNet-like model with batch normalization.

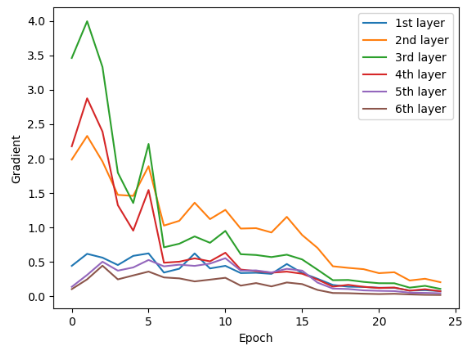
	Mean (%)	Standard Deviation (%)
CNN + BN	49.38	15.02
CNN + BN + WS	47.16	18.97
CNN + BN + AL	3.21	0.50
CNN + BN + WS + AL	2.31	0.36

Looking at the results shown above in table III, it can be assumed that batch normalization improves the performance, since its introduction in the CNN model with weight sharing and auxiliary loss led to the really good mean digit comparison error of 2.31%. More generally, batch normalization also seems to improve the stability of the model as the obtained performances had smaller standard deviations than previously.

Figure 1 shows the evolution over the epochs of the norm of the gradient at each layer of the CNN model, with or without batch normalization. It can be seen that without batch normalization, there is a significant difference in the gradient norm between the layers whereas with batch normalization, these differences decrease. Moreover, without batch normalization the drift in one layer influences a lot the others layers. It becomes clear that batch normalization allows each layer of the network to learn by itself more independently of other layers.



(a) CNN



(b) CNN + BN

Fig. 1: Gradient norm at each layer of the model with respect to the epoch.

B. Choosing the activation function

ReLU, Leaky ReLU, ELU and SELU were the four different activation functions that were selected for this study. The performance of each of them was tested with SGD as optimizer and a learning rate of 0.1. The goal was to compare these functions based on their performance, in order to find out which one works best for the previously selected CNN LeNet-like model with batch normalization, weight sharing and auxiliary loss.

ReLU is the most commonly used activation function. Indeed, its ability to avoid gradient vanishing when the learning rate becomes high and the resulting sparse coding have made ReLU more popular than the sigmoid or tanh functions.

Leaky ReLU slightly differs from ReLU because of its small slope equal to αx for $x < 0$ (instead of remaining at 0), thus causing a leak and extending the range of ReLU. Its performance was assessed with the default value $\alpha = 0.01$.

ELU replaces the $0.01x$ slope of Leaky ReLU by a small exponential saturation for $x < 0$. This modification is supposed to make the mean activations closer to zero which speeds up training. The parameter α controls the scale of this small exponential increase and the performances were assessed for $\alpha = 0.5$ and $\alpha = 1$.

SELU on the other hand is the scaled version of ELU. The scaling parameter is λ, and the used value here was the default one, which is equal to 1.05.

As it can be observed in table IV, the mean digit comparison error varied between 2.12% with Leaky ReLU and 2.71% with SELU. Thus, it was decided to keep Leaky ReLU as

TABLE IV: Digit comparison error for each activation function.

	Mean (%)	Standard deviation (%)
ReLU	2.30	0.26
Leaky ReLU	2.12	0.29
ELU ($\alpha = 0.5$)	2.20	0.35
ELU ($\alpha = 1$)	2.36	0.46
SELU	2.71	0.44

activation function for the final model, since it reached the best performance with a really small standard error of 0.29%.

C. Choosing the optimizer and optimizing the learning rate

At this point, the performances of the SGD, RMSprop and Adam optimizers were tested and compared, again to find out which one works best and improves the most the final model performance.

SGD (Stochastic Gradient Descent) is an iterative method for optimizing an objective function. It can be seen as a stochastic approximation of gradient descent optimization, since it replaces the gradient calculated from the entire data set by an estimate calculated from a randomly selected subset of the data.

RMSprop (Root Mean Square Propagation) is an extension of SGD. It maintains a per-parameter learning rate that is adapted according to the average of recent magnitudes of the gradients with respect to the weights (ie how quickly they are changing).

Adam (Adaptive Moment Estimation) is probably the optimizer that performs the best on average. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance).

Nonetheless, in order to compare the performance of the different optimizers, a suitable learning rate had to be used for each of them. Thus, before each training, the optimal learning rate was estimated as the mean learning rate at which the loss reaches its minimum over 10 rounds.

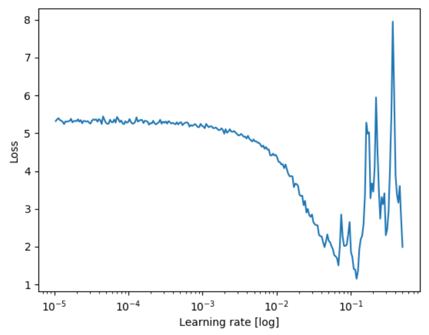


Fig. 2: Loss in function of the learning rate with SGD optimizer.

To illustrate this, figure 2 shows the evolution of the loss with respect to the learning rate, with SGD as optimizer. In this case, the minimum is reached at approximately 0.1 and thus would be used as optimal learning rate.

TABLE V: Optimal learning rate used and corresponding digit comparison error for each optimizer.

	Learning rate	Mean error	Standard deviation
SGD	0.1383	2.04 %	0.27 %
RMSprop	0.0115	3.10 %	0.46 %
Adam	0.0117	2.45 %	0.50 %

Table V summarizes the optimal learning rate found over 10 rounds for each optimizer and the corresponding mean and standard deviation of the digit comparison error. The estimated optimal learning rate was around 0.14 for SGD and 0.01 (so more than ten times smaller) for RMSprop and Adam. Surprisingly, the use of SGD led to the best mean error of 2.04%, with a good standard deviation of 0.27%. RMSprop gave bad results with a mean error of 3.10% and Adam was in between with a mean error of 2.45%.

IV. Summary

To sum up, the first step of the decision-making process described in this paper was to select the network architecture. The performances of a FNN and of a LeNet-inspired CNN model were compared and the result was clear: the CNN performs better on this image classification task. And this is quite obvious since the first part of the task was to extract location dependant features from the images to perform digit recognition. The benefits of introducing weight sharing and an auxiliary loss were evaluated for both models. It appeared that the combination of both improves drastically the accuracy of the model, as the error dropped from 49.43% to 3.98% for the CNN. Weight sharing enabled to substantially lower the degrees of freedom of the training problem, reducing the number of parameters to optimize and thus accelerating the convergence to some optima. The introduction of an auxiliary loss to train the main objective was also a major improvement, enabling to optimally share the training efforts between the recognition and the comparison task. Batch normalization was finally added to the CNN model, leading to a digit comparison error of 2.31%.

Secondly, different activation functions were compared in order to get the best performing one for the actual objective. The activation functions tested were ReLU, Leaky ReLU, ELU (with $\alpha = 0.5$ and $\alpha = 1$) and SELU. Leaky ReLU reached the best results, with a mean digit comparison error of 2.12%.

As a last step, different optimizers were tested and a procedure for optimizing the learning rate was implemented. The optimizers compared were SGD, RMSprop and Adam. The optimal learning rate found was in the order of 1e-1 for SGD and 1e-2 for RMSprop and Adam. Unexpectedly, SGD was the optimizer that performed the best, and enabled to reach the mean digit comparison error of 2.04% with a standard deviation of 0.27%.

Finally, one possible way to improve our final best model could be to convert the fully connected layers into convolutional ones, making the network architecture fully convolutional, and increasing this way the benefits of weight sharing as well as the invariance and re-usability of the network for further tasks.

APPENDIX

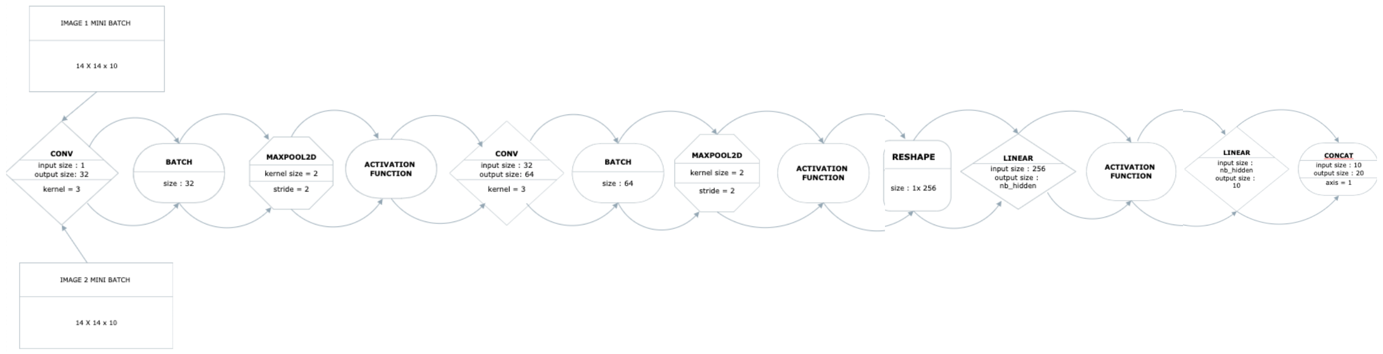


Fig. 3: Architecture of the digit recognition part of the LeNet-like CNN with weight sharing and batch normalization. The digit comparison task is not shown here, it follows the concatenation.

REFERENCES

[1] Y. L. et al., "Backpropagation applied to handwritten zip code recognition." *Neural Computation*, vol. 1, pp. 541–551, 1989.

[2] C. S. Sergey Ioffe, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, pp. 448–456, 2015.