Classification, weight sharing, auxiliary loss

Mauro Leidi, *Sciper:284713,* Denis Kravtsov, *Sciper:282379,* Sophie Du Couédic, *Sciper:260007,*

I. Introduction

The objective of this project is to test different architectures to compare two digits visible in a two-channel image. It aims at showing in particular the impact of weight sharing, and of the use of an auxiliary loss to help the training of the main objective.

The former, weight sharing, consists of having single weights shared among many connections in the network. It is typically relevant in our case, as it allows to leverage the intrinsic similarities of having a pair channels of digit images.

The aim of auxiliary network is to deal with the vanishing gradient as the networks become deeper. It does so by using the idea that the first layers already contain some useful information that can be reinforced using a second loss during training. In our case, we use auxiliary loss to attempt a classification of the digits in the firsts layers and learning the actual comparison in the following layers.

II. Data

The goal is to compare two digits visible in a two-channel image. The training and test consist each of 1000 pairs of 14×14 images from the MNIST dataset.

III. Architectures

We want to assess how weight sharing and auxiliary loss individually impact the performance of the training, and also the stacked effect of weight sharing and auxiliary loss together. In order to do that, we have implemented 4 different models.

A. Baseline model

The *Baseline* model is used as a point of comparison with the other architectures. It doesn't implement weight sharing nor uses auxiliary loss, and is constituted simply of fully-connected layers as detailed in the table I.

B. Baseline model with auxiliary loss

This architectures derives from the *Baseline* model with an added cross-entropy loss for the classification of the digits. It is summed with the initial comparison loss as described in equation 1. The model is adapted to take a single channel image, in which the images from the original channels alternate, and outputs both the digit classification and the result of comparison evaluation. Because the comparison needs to have the pair of images alongside, we reshape the output tensor of layer 3 to artificially recreate the two channels. The resulting differences are colored in blue in table I.

Layer	Modules
1	Fully Connected : 392 (196) $\rightarrow$ 160 Batch Normalization ReLU Max-pooling (kernel size : 2) Dropout (probability : 50%)
2	Fully Connected : 80 $\rightarrow$ 100 Batch Normalization ReLU Max-pooling (kernel size : 2) Dropout (probability : 50%)
3	Fully Connected : 50 $\rightarrow$ 10 ReLU outputs $\rightarrow$ digit classification
4	Fully Connected : 10 (20) $\rightarrow$ 2 outputs $\rightarrow$ comparison classification

TABLE I: Baseline model

$$Loss = InitialLoss + AuxiliaryLoss \quad (1)$$

C. CNN model

The *CNN* model implements weight sharing. We use 2d convolutional layers above fully-connected layers, as described in table II.

D. CNN model with auxiliary loss

This last architecture combines both weight sharing and auxiliary loss. As for *Baseline* with auxiliary loss, it takes one single channel image and outputs both the digit classification and final comparison. The adaptations of the *CNN* architecture for auxiliary loss are coloured in blue in table II.

Layer	Modules
1	Conv : 2 (1) → 32 (k. size : 5, pad. : 2) Batch Normalization ReLU Max-pooling (kernel size: 3)
2	Conv : 32 → 64 (k. size : 3, padding : 3) Batch Normalization ReLU Max-pooling (kernel size: 2)
3	Conv : 64 → 32 (k. size : 3, padding : 3) Batch Normalization ReLU Max-pooling (kernel size: 2)
4	Conv : 32 → 16 (k. size : 5, padding : 3) Batch Normalization ReLU Max-pooling (kernel size: 2)
5	Fully Connected : 144 → 100 ReLU
6	Fully Connected : 100 → 10 ReLU outputs → digit classification
7	Fully Connected : 10 (20) → 2 outputs → comparison classification

TABLE II: CNN model

IV. Method

We tested each architectures for 15 rounds, and in each round new data were generated and the model reset with new random parameters.

A. Hyper-Parameters

We used the following hyper-parameters :

- Epochs : **25**
- Mini-batch size : **100**
- Learning rate :
  - *Baseline* : **0.0011**
  - *Base. Aux* : **0.0056**
  - *CNN* : **0.001**
  - *CNN Aux* : **0.0022**
- L_2 rate :
  - *Baseline* : **0.33**
  - *Base. Aux* : **0.001**
  - *CNN* : **0.1**
  - *CNN Aux* : **0.11**

B. Methods

1) Data Standardization

Data standardization can help the gradient descent to move more smoothly. We scaled our data around their mean value and distributed them using the standard deviation. This was done directly in the function `generate_pair_sets`.

2) Cross-validation

We have implemented a 2-fold cross validation to find the best hyper-parameters values. This way, the accuracy is increased.

3) Max-pooling

Pooling layers are generally used to group several inputs into a single one, letting us to compute a low-dimension meaningful signal from a high-dimension one. In particular, max-pooling computes the max values over non-overlapping blocks, and is applied independently on our two channels. In addition, pooling layer allows to keep intact the meaning of the proximity between different pixels on the image.

4) Dropout

Dropout prevents data over-fitting by deactivating units at random during the forward pass. By making those other units unreliable, units become more independent and capable of adapting to unseen data.

5) Batch normalization

Batch normalization consists of forcing the activation statistics by re-normalizing them. This way, a layer won't have to adapt to the changes of the activations in the previous layers.

6) Cross-entropy loss

Cross-entropy loss, in contrary to MSE loss, doesn't penalize the model when it is "too far" in the correct prediction. It is particularly appropriate for our classification task. This way, predicting 7 or 2 instead of 1 will have the same effect on the loss.

7) Adam optimizer

Adam is an optimization of the stochastic gradient descent, which removes the redundancy of regular gradient descent, by updating the parameters more frequently. It is also better at finding global minima because the updates are done with a high variance.

8) L_2 penalty

L_2 regularization has effect to keep the parameters small and make them less dependent on the data, reducing over-fitting. Using such a penalty leads to higher training error but a lesser gap between training and test errors.

V. Results

A. Errors

The errors rates during training and tests are displayed in figure 1 and figure 2, and the corresponding values for mean and standard deviation in table III. We can see that the use of auxiliary loss has a drastic impact on the performance of the network. Whereas during training one is tempted to say that the use of auxiliary loss doesn't add any improvement

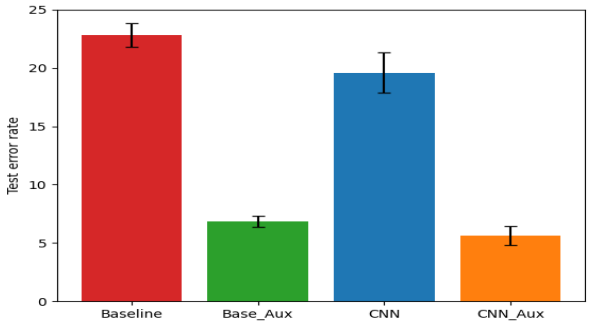

Fig. 1: Train error rates

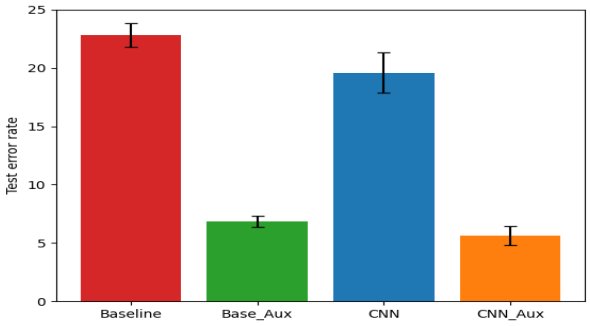

Fig. 2: Test error rates

	Nb. parameters	Train error [%]	Test error [%]
Baseline	72'032	11.56 ± 0.61	22.98 ± 1.125
Base. Aux	40'692	3.41 ± 0.48	7.66 ± 0.6
CNN	67'228	4.9 ± 5.2	21.56 ± 2.5
CNN Aux	66'448	1.3 ± 0.53	5.83 ± 0.97

TABLE III: Results per model

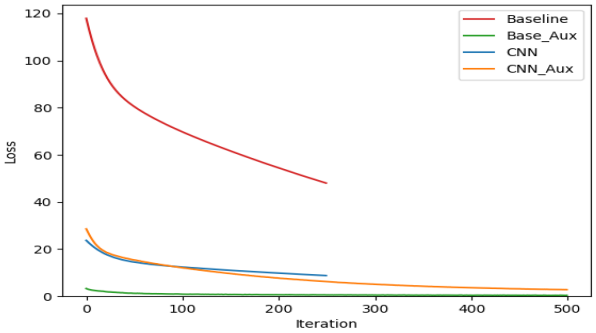

Fig. 3: Loss evolution

over weight sharing, the error rate actually doesn't grow much during testing, in contrary to the other architectures. This reveals a strong resistance to over-fitting.

On the other hand, the *CNN* network performs incredibly well during training, but it's error rate is decoupled during testing. This over-fitting behaviour can probably be explained partially by the fact it contains more layers than the baseline model. The use of L_2 penalty helped a little to overcome the over-fitting problem, but would quickly lead to high error rate and variability in the results if it is set to a too large value. The use of dropout immediately decoupled the error rate. *CNN* also has a high standard deviation, both for training and testing, whereas the use of auxiliary loss provides more reliable results. Finally, we can see that the use of auxiliary loss on the *CNN* model can overcome the data over-fitting and high variability problems, leading to the bests results (not by far).

B. Loss

The evolution of the loss value for all the architectures is displayed in figure 3. The doubled number of iterations of architectures using auxiliary loss is explained by the fact that we merged the two channels of image into a single longer one, and at any time we have half the number of images in the network. The figure 3 shows that the *Baseline Aux* has from start to end the lowest, and converges already after a few iterations, whereas the *Baseline* architectures still hasn't converged by the end of the training. *CNN* model situates in between, and converges also pretty quickly. By comparing the losses with the accuracies we can see the effects of the regularization. Although CNNAux has the greatest precision, the loss is higher than that of *Baseline Aux*. This is due to the fact that the weights are greater and therefore the regularizer increases the loss.

VI. Conclusion

CNN seems to be quite a complex network. It reacts very sensitively to the tuning of its hyper-parameter and any use of methods such as dropout or L_2 penalty. Reversely, the use of auxiliary loss provides a reliable way to improve performance of the network. This result actually makes sense, because we use an additional (and important) information to train the network, namely the target classes of the digits. However, we can imagine that the use of similarities in the input images can be of great help as well, if one is able to find the best possible parameters for the network to leverage the similarities.