Miniproject 1 Report

Olivier Cloux[1]

[1]Department of Computer Science, EPFL

Abstract—**Building a deep learner that performs a specific task is not always easy. To help us, we can use some design patterns. In this paper we will look how the use of weight sharing and auxiliary loss impact the performance of deep learners. We will see that both of them help immensely the classification task at hand, with weight sharing allowing for a faster train, while auxiliary loss gives better accuracy overall.**

I. Introduction

Deep Learners have been around for quite some time now [1]. But only recently have we started to unlock their potential, thanks to faster than ever hardware. Among some properties of Deep Neural Networks (DNN), we can cite the use of auxiliary loss, and the use of weight sharing.

Behind the idea of weight sharing is the supposition that similar parts of the input can affect the output similarly. There are multiple ways to implement this, either by using a Siamese Networks [2], or by using so-called Convolutional Neural Networks [3]. This is notably useful for images-like data, where we can suppose some features detection (colors, angles, edges,…) may be useful at multiple parts of the input.

The second interesting property, auxiliary loss, was introduced by a GoogLeNet paper [4], and is roughly defined as using additional properties as the target. Totally unrelated example, if we want to predict which of two hand-written digits is greater than the other, we could try and predict the actual digits to train our network, and use this help to update our network for the former task.

II. Task

In this experiment, we will try and predict which of two hand-written digits is greater than the other[1]. The data we use is a slight modification of the MNIST [5], where each entry is composed of two ordered but unsorted handwritten digits. The label to predict is a binary value, indicating which of both is the greatest. Note we also have at hand the original labels (indicating the actual value of each digit), but only use it in certain cases.

Each image is a 14×14 grayscale image. So each item is a $2 \times 14 \times 14$ tensor.

	No auxiliary loss	Auxiliary loss
No weight sharing	Baseline	Baseline + Auxiliary
Weight sharing	CNN	CNN + Auxiliary

TABLE I
NAMES OF MODELS CONSIDERED

[1]I lied earlier, it wasn't unrelated

Layer	Name	Output (if applicable)
1	Fully Connected 196|392 $\rightarrow$ 200 Batch Normalization ReLU MaxPool 1d (Kernel size 2, stride 2)	
	Dropout 50%	
2	Fully Connected 100 $\rightarrow$ 256 Batch Normalization ReLU MaxPool 1d (Kernel size 2)	
	Dropout 50%	
3	Fully Connected 128 $\rightarrow$ 10	Output classification 10
4	Fully Connected 20 $\rightarrow$ 2 ReLU	Output classification 2

TABLE II
STRUCTURE OF THE BASELINE NETWORK

III. Models and methods

As we wish to investigate the effect of weight sharing and auxiliary loss, it makes sense to create 4 models, combining these criterions. As you can see in Table II, we really only have two models (*Baseline* and *CNN*). Both of them always output a binary classification (our main goal), and a 10-classification (the digits of the numbers). But if we don't wish to use them, we simply discard the latter.

Furthermore, we also use l2-regularization to help overfitting and reduce the amplitude of the weights.

A. Model 'Baseline'

This first simple model is composed of 4 layers, all quite similar to each other. You can see the structure in Table II. The first three layers serve to create the classification 10 (the value of each digit). Then this layer continues and is reduced to a size 2, the binary value of which is greater. Note than between layers 1 and 2, and between layers 2 and 3 we apply a dropout of 50% to reduce overfitting. Also, as mentioned earlier, the output 10-classification is only used if we wish to use auxiliary loss. Finally, note that the input is either 196 or 392. The first case corresponds to an input of 14×14 and is used when we use auxiliary loss. In this case, we need to split images and identify them accordingly. One input is then 196 pixels. The latter is twice that amount, and is used without auxiliary loss, when an input is 2 images (hence $2 \cdot 196$).

B. Model 'CNN'

This network is relatively similar, with the replacement of some Fully Connected layers by 2D Convolutional layers. You can see the details in table

Layer	Name	Output (if applicable)
1	Conv. 1|2 → 16, kern. 5, padding 3 Batch Normalization ReLU MaxPool (Kernel size 2)	
2	Conv. 16 → 20. kern. 5, padding 3 Batch Normalization ReLU MaxPool (Kernel size 2)	
3	Fully Connected 500 → 300 ReLU	
4	Fully Connected 300 → 10	Output classification 10
5	Fully Connected 10 → 2 ReLU	Output classification 2

TABLE III
STRUCTURE OF THE CNN NETWORKS

C. Method

In order to compare the models between them, we created a testing pipeline, through which they are all passed. The same actions are repeated for a certain amount of *rounds* (about 20) to obtain averaged results. At each round, a thousand training and testing items are created. If we wish to use auxiliary loss, the data is flattened. Then the network is trained (logging the loss), and finally we compute the training and testing error. All those steps are timed. Results of each rounds are returned for analysis.

The training is quite straightforward. During a count of epochs, we again repeat loop, splitting the data in batches, account for the 2-classification, drop or keep the 10-classification, and add ℓ_2 regularization. Finally we apply the backward pass and reset the gradients.

The optimization was made using the Adam optimizer, on CrossEntropy loss. The total loss is either simply the "normal" loss, or the sum of it with the Auxiliary loss.

D. Parameters

The following parameters were used:

- 20 rounds
- 25 epochs
- Mini-batch size 100
- Cross-Entropy loss
- Adam Optimizer
- lr Baselines $1e-3$
- lr CNNs $5e-4$
- ℓ_2 0.1

IV. Results

A. Training loss

For each model, we logged the loss while training. This allows us to see how fast a network will converge. Loss is averaged across rounds, and we plot the deviation of each step. You can see the results at Figure 1. This shows the CNN models tend to have a better training loss, and reach their limit quite fast, while Baseline models can start far worse, and take some time to reach equivalent performances. Note that this could be further tweaked with cross-validation and hyperparameters optimization, but this is out of the scope of this project, for which we used "good enough" hyperparameters.

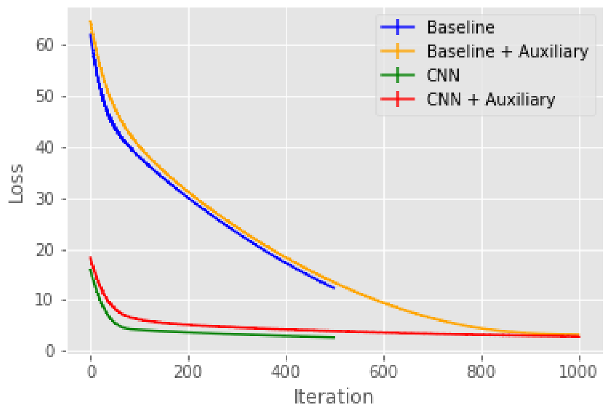

Fig. 1. Evolution of the training losses

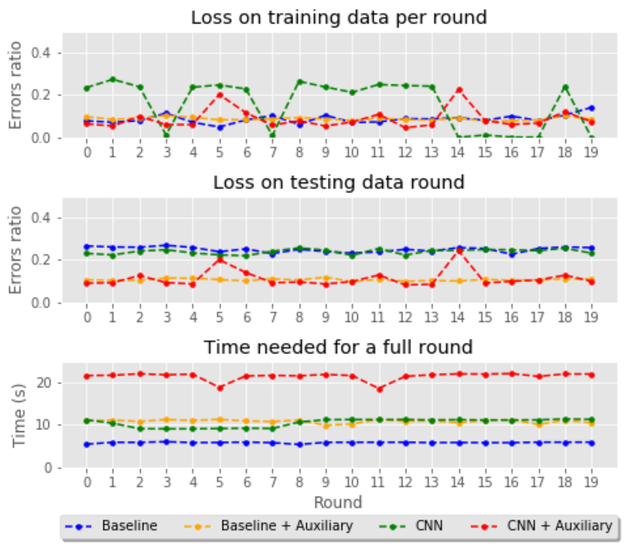

Fig. 2. Errors and Times across rounds

B. Errors and Time

The above only shows us the evolution of the training loss. We now look at three interesting measures: training error, testing error, and training time. Figure 2 shows these results There are some conclusions that arise. First of all, using the auxiliary loss yields much better results on the testing set (actually the most important one). Also, the more complex a model is, the longer the training takes (no surprise there). Interestingly, all models have a similar training error, except for CNN, that oscillates between two very distinct states. You can find a summary of the mean/std for these metrics in Table IV. Using the above results, there we can draw some conclusions.

- CNN is more complex (and thus longer to train) than Baseline.
- Using auxiliary loss makes the training longer even, regardless of the underlying network.

	Train error [%]	Test error [%]	Time/round [s]
Baseline	8.7 ± 2.0	25.0 ± 1.2	5.8 ± 0.15
Baseline + Aux	8.8 ± 0.7	10.6 ± 0.5	10.8 ± 0.4
CNN	15.9 ± 11.6	23.9 ± 1.19	10.5 ± 0.9
CNN + Aux	8.8 ± 4.8	11.3 ± 4.1	21.4 ± 0.9

TABLE IV
RECAP OF TRAIN/TEST ERROR AND ROUND TIMES

- Baseline takes more epochs, but can achieve comparable results.

V. SUMMARY

To summarize the above, the lessons to take home are that:

- If you lack data, use weight sharing, it will allow for a much faster convergence, even if each epoch is significantly slower.
- With more data, the weight sharing may not be necessary, but can probably achieve better results with some additional tweaks (notably of hyperparameters).
- Use all the data at your disposal. There is no point in discarding some of it. Even if it's not the main label we are trying to classify, it's still better to use it for auxiliary loss.

There is still work left to obtain a more precise idea. First of all, we could compare more networks, with less differences between them. Also, we could compare more architectures of weight sharing, other than CNN. Siamese nets, mentioned earlier, could be an interesting track. Finally, some more work could be done on hyperparameters to obtain "the best" of each network, to have a more sensible comparison.

REFERENCES

[1] "Deep learning," May 2020, page Version ID: 954954275. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=954954275
[2] "Siamese neural network," Apr. 2020, page Version ID: 953925157. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Siamese_neural_network&oldid=953925157
[3] "Convolutional neural network," Apr. 2020, page Version ID: 953234856. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=953234856
[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sep. 2014, arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842
[5] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/