EE559 Deep Learning
Mini-project #1 Report

Mehdi Akeddar (261344), Hugo Birch (261684), Louis Piotet (260496)
EPFL - Switzerland. 22nd of may, 2020.

***Abstract*—This report is a description of our approach for the the first mini-project in the Deep Learning course given at EPFL, Lausanne. The first project consists in implementing various neural network architectures that need to be able to deduct if, from two provided images with digits, the first number is smaller or equal to the second or not. Emphasis is placed on the effect of weight sharing and the use of auxiliary loss.**

PROJECT #1 - CLASSIFICATION, WEIGHT SHARING, AUXILIARY LOSSES

The goal of this project is to investigate the effect of various neural network architectures and the application of known techniques such as weight sharing to improve the classification results for a given task. To illustrate the gain of each technique, various different architectures will be presented and studied.

A. Dataset and Global Parameters

The MNIST dataset was used in this project. A provided function was building the vectors of input and target data for both training and testing. A single input comprised of two MNIST images. The target output (the class) tells if the first digit is smaller or equal to the second one.

The global parameters were set to the following values and unchanged for every trial of the different architectures.

N_{train}, N_{test}	1000
Learning rate η	0.1
Size of mini-batches	100

Table I: Global parameters used for every architecture

B. Baseline network

This is a baseline solution, where one does not make any considerations in regard to the data structure. Their sole thought is to stack a bunch of convolution and linear layers.

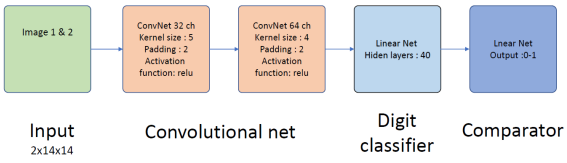

Figure 1: The implemented inefficient structure to handle the given problem.

The reasons to justify the architecture presented on Fig. 1 are that convolutions layers are good at handling images. The dense network that follows should then determine if the first number is less or equal to the second. This architecture is a notoriously bad idea because it learns twice identical actions on totally different parameters: processing each image and digits recognition. As a matter of showcase, the architecture does not benefit from batch normalization, dropout nor weight sharing.

Due to the lack of batch normalization and the obviously bad structure that does not make any use input data structure, we expect that this model will not properly learn the underlying logic and will eventually do overfitting. I.e., the testing error will be large.

C. Network with Weight Sharing

Since the network has to solve various time identical problems for both input images, a good idea would be the use the exact same subsections of the net to treat both images. This is especially relevant for such problem as most of the tasks are shared for both input images: image processing and digit recognition.

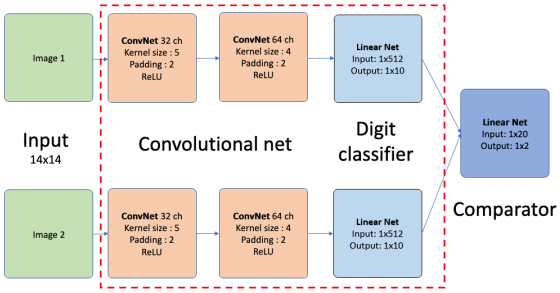

Figure 2: Architecture of the structure used with weight sharing and auxiliary loss. The weight sharing is done on both the Convolutional net and Digit classifier net (encircled region).

This architecture consists in consecutively giving each image of the pair to the same network in order to implement the principle of weight sharing. The overall network is similar to the one implemented in the baseline model. However, in the forward pass, the network feed each batch of images separately to a net that takes a $N \times 1 \times 14 \times 14$ image as input and has 10 output units (the possible digits, 0 to 9). Applying the net on both batch of image for the first digit of the set and second results in two 10×1 vectors which are fed to a final layer. This layer combines the two and is expected to perform

the final task which is fairly straightforward: the comparison of both digits and output on either of the two units accordingly. Fig. 2 summarizes how the implemented solution work. In order to explore the benefits of additional techniques commonly used in neural networks, 2 approaches have been carried: one exactly as presented and another adding in batch normalization and dropout.

1) No Dropout/No Batch Normalization: An implementation that does not use batch normalization nor dropout is generally not recommended. The lack of regular normaliazation does not mitigate the internal covariate shift due to the change of the parameters during learning and can lead to poorer results. Lacking dropouts also generally results in overfitting.

2) With Dropout and Batch Normalization: Dropout units were added after the convolution layers (2D Dropout) and after the dense network for digit recognition. As we know, adding dropout limits the chance of the network to fall in overfitting. Batch normalization is present in both the convolution network (as 2-dimensional batch norm) and in the dense network in hope to benefits from the various benefits it brings. The learning rate was left unchanged, even though it's usual to increase it when batch norm is used as seen on fig. 3.

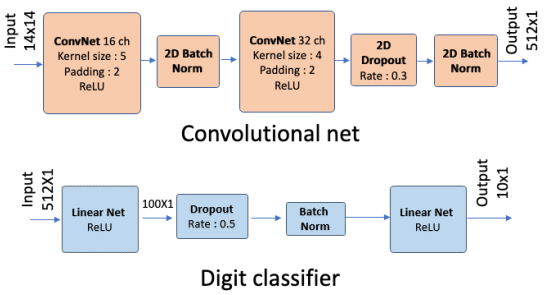

Figure 3: Detailed sequential structures of the weight shared section of the network for both the convolutional layer and linear layer.

2D dropout probability	0.3
1D dropout probability	0.5

Table II: Probabilities for the dropout rates. These will be used for every subsequent architectures.

D. Network with weight sharing and auxiliary loss

As we know the numerical value (digit) for each image given as an input to the network, it would be of good use to exploit this supplementary knowledge and provide to the network in a form or another to improve the resulting classification. This can be done with the use of auxiliary losses. Beside the loss on the network output, one can add in a loss on the output of the digit recognition layer. The training can be done by summing the loss computed with the output and target (0,1) and the possible digit classes (0,..,9) classes. The implemented solution also benefits from dropout and batch normalization.

1) Constant auxiliary loss: A basic approach is to simply add up the losses. So the total loss is:

$$loss = loss_{comparaison} + loss_{classA} + loss_{classB} \tag{1}$$

2) Discounted auxiliary loss: One possible intuition is to give more importance to the $loss_{comparaison}$ component. A constant discount factor α is introduced.

$$loss = loss_{comparaison} + \alpha(loss_{classA} + loss_{classB}) \tag{2}$$

Another approach would be to use a non-constant discount factor :

$$loss = (1-\alpha)L_{comparaison} + \alpha(L_{classA} + L_{classB}) \tag{3}$$

Starting by giving more importance to the classification and decreasing this importance throughout the training to allow the architecture to focus on the comparaison.

In either cases, the Cross Entropy Loss will be used as the criterion to compute the loss with respect to the output of the digit recognition layer. Then, backpropagation will be done after summing the losses together.

E. Results

The performance of the different architectures is presented in table III. The standard deviation regards the number of error, not the percentage. The mean and standard deviation were computed by doing 10 different training of the exact same model. The number of epochs was not changed between each training.

	Train	SD	Test	SD
Baseline	4.4%	60	20.1%	47
WS	3.8%	5.5	12.6%	9.4
WS+AuxLoss	3.9%	4.6	6.9%	5.5

Table III: Training and testing errors for each of the models and associated deviations. The standard deviation were computed by repeating the training procedures 10 times and keeping the number of epochs unchanged.

We can clearly see that the first model performs overfitting. The very large difference between training and the testing error suggests that the model doesn't learn well at all the underlying logic. Adding in weight sharing as well as dropouts and batch norm reduces the testing error, however there's still a significant difference in classification rates between training and testing. By giving the network more knowledge by adding in constant auxiliary losses, we finally reach a point where training and testing gives extremely similar classification results.

	Train	SD	Test	SD
WS (w/o)	4%	29.4	14.5%	8.3
$\alpha = 0.5$ (eq.2)	3.6%	8.4	7.5%	5.1
$\alpha = 0.75$ (eq.2)	3.5%	4.1	7.0%	4.3
$\alpha = 0.25$ (eq.3)	4.2%	6.8	8.2%	5.4
$\alpha = 0.75$ (eq.3)	7.6%	7.4	7.5%	8.0

Table IV: Training and testing errors for each of the models and associated deviations. The standard deviation were computed by repeating the training procedure. WS (w/o) means weight sharing without batch norm and dropouts.

The table IV shows two other discussed implementation throughout the report: weight sharing lacking dropouts and batch normalization and the other, weight sharing with auxiliary losses with a discount factor (constant and non-constant).

Adding in weight sharing but lacking batch normalization and dropouts decreases somewhat the testing error but the difference with training remains high. The small improvement comes from the availability of twice as many samples as both images are fed through the same network sections.
In general, using a discount factor, i.e., lowering the value of some knowledge either with equation 2 or 3 leads to a worst performance compared to auxiliary loss with weight sharing and no discount factor ($\alpha = 1$). However, using a constant discount factor of $\alpha = 0.75$ on the digit knowledge led to 0.4% improvement in the training error and 0.1% worsening of the testing error. This is an overall improvement compared to the weight sharing + auxiliary loss model seen in table III. It's very possible that there exist yet another α that improves further the result in the vicinity of 0.75. But overall, for this classification problem with auxiliary losses on the digit recognition, each loss should have a similar weight. This means it is as important to classify digits properly as to do proper comparison of them. Both tasks being closely linked together.

I. Testing with the script

The `test.py` script can be given arguments to change its behaviour to illustrate various part of the project. By default, giving it no arguments will train 10 times the weight sharing + auxiliary loss network with 25 epochs each time. At the end, it will print the mean and standard deviations of the train and test errors. To showcase the other network discussed throughout the report, a first numerical argument can be passed to the script. It determines which network to run and can be used to support the content of this document.

- 0 - Network with auxiliary losses and weight sharing
- 1 - Network with weight sharing
- 2 - Baseline network
- 3 - Run all networks, from baseline to weight sharing with auxiliary losses.

The second argument will determine if the script should repeat the learning procedure of a given network. The number of rounds is defined in the script. By default the script does not repeat the training and will print the loss at each epoch. However this can activated by passing 1 as the second argument. This will make the script train 10 times each model with the defined number of epochs. The script will not print anymore the loss at each epoch however. The last argument defines the discount factor for the auxiliary loss. It is by default unitary.

As an example, to repeat the training procedure by the defined number of rounds and for all networks, and evaluate the performance of the network with auxiliary loss using a constant discount factor of 0.75, one should pass the following arguments:

```
#> Python3 test.py 3 1 0.75
```

or, from an IPython environment, without touching the discount factor (using the default value of 1.0):

```
[]: run test.py 3 1
```

This will typically gives results close or identical to the ones presented in table III of this report.

II. Conclusion

The different architectures developed show the advantages of using optimization methods like dropout, batch normalisation. Beside, using renowned techniques are not the only things that matter as it was illustrated in the report. It is important if not more to consider how the data is structured and take advantage of this. In this example, due to the fact the input consist of just stacked element of the same type, we can put this to use by treating each stacked element the same with weight sharing. An improvement in accuracy of the percentage of classification is observed between the baseline and the model using weight sharing and auxiliary loss. This is, as explained previously, likely due to the fact that the auxiliary loss adds in additional knowledge which improves the results.