TP ROB313
Siamese Neural Networks for beginners

Ghaith SASSI
Myriam HAMROUNI

16 December 2020

1 Introduction

In this project, we are going to get used to the deep learning library Pytorch and implement a Convolutional Neural Network, train it and test it and finally perform a grid search to extract the best hyper-parameters.

2 Exercise 1 : MNIST classification with Pytorch

2.1 Simple Deep neural network

Question 1:

In the file `srcipt_pytorch_simple.py`, we can find the forward backward algorithm to train a deep learning network with two layers. The weights of network are using the gradient descent. The loss here is calculated as following :

$$Loss(\hat{y}_i, y_i) = \sum_{i=0}^{n} \|\hat{y}_i - y_i\|^2$$

This is the summed error of the predicted values ($\hat{y}_i$).

In this python code we have only the training part of the data. There are two layers and at each step we make a forward pass to compute the predicted y, print the loss and then we go backward to update the weights W_1 and W_2.

Question 2:

In this question we're going to use the Module Class of Pytorch to create our neural network in the class Net.

With the fixed hyper-parameters ;

- $Batch\_size = 64$
- $Input\_dimension = 784$
- $Hidden\_dimension = 128$
- $Output\_dimension = 10$
- $Learning\_rate = 0.01$

We added the training part with same principals as question 1 ; forward/ backward loop and computing the loss.

2.2 Add summaries

Question 3:

To keep track of the Training Loss and the Training Accuracy we used the SummaryWriter provided by TensorBoard that stocks the data collected at each run and the use it to visualize it. To launch tensorboard we use the command (on windows)

```
py -m tensorboard.main --logdir=runs
```

And as we can see from the figure 1, we can see the evolution of the accuracy over all the training/testing steps.

Figure 1: Graph results of the accuracy run with the model explained in question 4.

The two parameters that we should add are :

1. Precision : The precision is calculated as the ratio between the number of Positive samples correctly classified to the total number of samples classified as Positive (either correctly or incorrectly). The precision measures the model's accuracy in classifying a sample as positive and it reflects how reliable the model can be in classifying samples as Positive[1].

$$Presicion = \frac{True_{Positive}}{True_{Positive} + False_{Positive}}$$

2. Recall : The recall is calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples.

The recall measures the model's ability to detect Positive samples. The higher the recall, the more positive samples detected[1].

$$Recall = \frac{True_{Positive}}{True_{Positive} + False_{Negative}}$$

2.3 Network models

Question 4:

CNN

First, we implemented a simple CNN ; two convolution layers, one maxpool layer and a fully connected layer.

ResNet

The ResNet18 model can be described as follow :

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Figure 2: ResNet18 Architecture

For this project, we re-implemented the Resnet18 model which can decomposed into many blocks. The main one is the Residual Block like shown in the image 3. It uses a skip connection to add the input to the output witch can help with gradient vanishing problem .

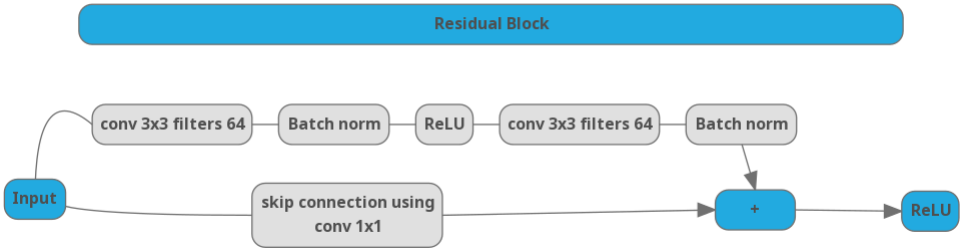

Figure 3

We made few modifications: such as adapting the size of the input by adding the InceptionBlock like described in the image of the figure 4.

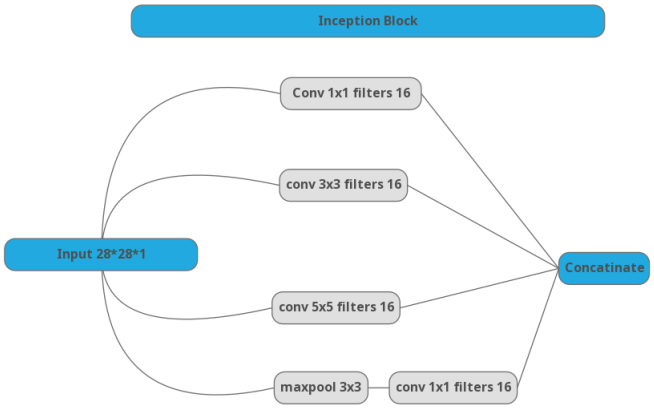

Figure 4: Inception Block added in the ResNet18.

Finally, we can see the architecture of the adapted model in the figure 5 showing the two main blocks that we mentioned.

Figure 5: Architecture of our ResNet18.

Also, we implemented another version of ResNet (it can be found in the code with the name of Resnetv2). It is a simpler version, containing less layers compared to ResNet18. The best accuracy found corresponds to ResNet18 and equal to 99.46% with parameters ;batch_size = 16, learning_rate = 0.01. We set these parameters based on the comparison made in Question 5.

To summarize :

	CNN	$ResNet18$	$Resnetv2$
$Best\_accuracy$	95.62%	99.464%	99.03%

2.4 Hyper-parameter optimization

Question 5:

In this part we are going to find the best hyper-parameters (λ, Weight decay and Batchsize) for this we are going to use the grid search, where we are going to fix one parameter and change the two others and then we note the accuracy of each combination.

First, we set the batchsize = 64 and see what are the best λ and weight decay.

λ	1.0	0.1	0.01	0.001
Weight decay				
0.1	0.1023	0.1129	0.1129	0.1280
0.01	0.9145	0.9800	0.9874	0.5886
0.001	0.9289	0.9901	0.9871	0.5543

Second, we set the weight decay = 0.001 and change the Batchsize.

λ	1.0	0.1	0.01	0.001
Batchsize				
5	0.1135	0.9894	0.9924	0.9693
64	0.9791	0.9870	0.9798	0.2056
200	0.9842	0.9907	0.5100	0.1719

To get these results we used the hyper-parameters optimization framework : Optuna. From the first array, we can see that the best combination is $\lambda = 0.1$ and $Weight\_decay = 0.001$. So it's better keeping λ around 0.1 (not close to 1 nor close to 0) and the $Weight\_decay$ is better when it's low for λ close to 0.

But we didn't test only these combinations, we tried several other ones ; by changing for example the $batch\_size \in \{4, 16, 32, ...\}$. So, we found the best accuracy mentioned in the previous question.

The second array shows how we get better results when the batch_size is high and with λ between $[1, 0.1]$. While it gets worse when λ is very low (around 0.001). On the other hand, for small batchsize it's better to keep λ low (accuracy equal to 96% with a batchsize equal to 5).

2.5 Regularisation

Regularisation is the key to avoid overfitting in a model. We can find some Regularisation methods such as : L_1 and L_2 regularisation, Dropout, etc.

In our Network we implemented the Batch Normalization which can be a way to provide some regularisation. And this lead to a high test_accuracy.

References

[1] AHMED FAWZY GAD. Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall.