Maëlie LEBARON
214322IV

Home assignment 2

IAS0360
November 2, 2021

1 Problem description

The aim of this homework was to train a neural network using MNIST dataset.

2 Own model - Tenserflow

I choose to optimize the accuracy of my model, which mean I want the accuracy to be the closest to 100%, either for the validation dataset or for my own dataset of digits.

Here are a list of optimizations I tried to obtain the best accuracy possible :

- Changing the learning rate
- Changing the number of epochs
- Using different activation functions
- Using different optimizer
- Changing the number of layers
- Changing the number of nodes in each layers
- Using or not the dropout function
- Changing the batch size

For the learning rate, the number of epochs, and the optimizer, I created a python script that tried all combination with a list of learning rates, of number of epochs... Obviously, I took a long time to compute, and that is why I only did it for a limited number of parameters, but it allowed me to totally drop some models idea, as the accuracy was very low.

I tested the other parameters by hand, determining if it had a significant impact on my model and if this impact was positive. When I found a modification on the other parameters that increased significantly the accuracy, I used the python script model again to check if I needed to change the number of epochs, or learning rate etc.

The most interesting results I had are listed in the following table (in bold, the parameters that are different from the precedent line):

Optimization method	Inference average speed	Memory size	Validation accuracy	Own accuracy
Learning rate : 0.01 Number of epoch : 30 Activation function : gelu Optimizer : nadam Number of layer : 2 Layers sizes : 80,60 Dropout function : yes Batch size : default (32)	141ms/10 = 14ms	68,270 parameters	0.9774000048	0.8999999761
Learning rate : 0.1 Number of epoch : 30 Activation function : gelu Optimizer : nadam **Number of layer : 3** **Layers sizes : 110,100,90** Dropout function : yes Batch size : default (32)	150ms/10 = 0.015s	107,450 parameters	0.977400004	0.800000011
Learning rate : 0.05 **Number of epoch : 10** Activation function : gelu **Optimizer : adam** Number of layer : 3 Layers sizes : 110,100,90 Dropout function : yes Batch size : default (32)	1s/10 = 0.1s	107,450 parameters	0.9758999943	0.8000000119
Learning rate : 0.1 **Number of epoch : 30** Activation function : gelu **Optimizer : nadam** Number of layer : 3 **Layers sizes : 80,60,50** Dropout function : yes Batch size : default (32)	358ms/10 = 0.036s	71,220 parameters	0.9801999926	0.8999999761
Learning rate : 0.1 Number of epoch : 30 Activation function : gelu Optimizer : nadam Number of layer : 3 Layers sizes : 80,60,50 **Dropout function : no** Batch size : default (32)	1517ms/10 = 0.1515s	71,220 parameters	0.9775999784	1.000000000

Optimization method	Inference average speed	Memory size	Validation accuracy	Own accuracy
Learning rate : 0.1 Number of epoch : 30 Activation function : gelu Optimizer : nadam Number of layer : 3 **Layers sizes : 60,100,80** Dropout function : yes Batch size : default (32)	150ms/10 = 0.015s	62,090 parameters	0.9774000048	1.000000000

Considering theses values, we can see that the line 2 and 3 of the table have a very good accuracy for the validation dataset, but not for my own data : the model seems to be over-fitted to the mnist dataset.

Lines 1 and 4 have a fairly better accuracy for my own data, and we can say that line 4 is the best compromise between accuracy maximization for the validation dataset (98%) and my personal dataset (90%).

After the model of line 4, I tried to maximize a bit more accuracy, but the only interesting results I have had were improving the accuracy of my own dataset to 100%, and reducing the accuracy of the validation dataset to 97,75%. This could be very interesting to test with a larger dataset, to see if the model is over-fitted to my dataset or has a really good accuracy overall. In fact, the 100% of accuracy is a really nice result, but we should consider it carefully as my own dataset only contain 10 images.

3 Two other architectures

I have faced a consequent problem trying to use other architectures on my computer. I took a surprisingly long time to execute any of the script (I am talking about 4hours depending on the parameters). My GPU was not detected, even after installing all dll files which were necessary and adding them to the path... As a result, I was not able to test efficiently my architecture, or to try a lot of different parameters. The results presented after are the best I could find with those constraints.

3.1 SqueezeNet

SqueezeNet is a convutionnal neural network which is known to have interesting results for few parameters (especially compared with AlexNet, where Squeezenet give the same accuracy for 50x less parameters).

It is base on fire module, in which we have a squeezing layer then expendings layers.

Using the following parameters :

- **Number of epochs :** 1 (otherwise, computing time start to be too consequent)
- **Optimizer :** Adadelta
- **Batch size :** 4
- **Activation function :** Relu

The results I have are : **30min of computing time in total (training), 711,124 parameters in total**. For the validation dataset : **a loss of 2.3020 and an accuracy of 0.1135**. For my own dataset : **a loss of 2.3025 and an accuracy of 0.1000**

3.2 MobileNet

MobileNet is also a convolutionnal neural network. There is several version, and the version I used is v2. It is based on bottleneck module.

Using the following parameters :

- **Number of epochs :** 1 (otherwise, computing time start to be too consequent)
- **Optimizer :** Adadelta
- **Batch size :** 5
- **Activation function :** Relu

The results I have are : **15min of computing time in total (training), 2,293,098 parameters in total**. For the validation dataset : **a loss of 2.3685 and an accuracy of 0.1045**. For my own dataset : **a loss of 2.3801 and an accuracy of 0.1000**

3.3 Comparison

MobileNet is faster than SqueezeNet in terms of training.

It is hard to compare the architecture due to the technical problem I had, but MobileNet and SqueezeNet seems to have the same accuracy overall.