Report Homework 3 - Deep Learning

Federico Gianno, s255548

***Abstract*—An implementation of a Convolutional Neural Network on a big image data-set. I used pytorch but also different deep layer framework can be used. The code implements a basic NN and CNN, the data loading, the training phase and the evaluation (testing) phase. The training and testing are on CIFAR 100 dataset (already included in Pytorch). All the test has been done in *Google colab:* a very user friendly python notebook from Google in which you can install python packages, download datasets, plot images, and a free GPU to do training.**

I. Introduction

I started from a pre-defined code that implement a basic **NN** and **CNN**, the data loading, the training phase and the evaluation(testing) phase. The purpose is to:

1) Train and understand the behaviour of the traditional NN
2) Train the CNN from scratch and to do several improvements in order to obtain the best accuracy
3) Load **ResNet18** pre-trained on ImageNet and fine-tune it on the CIFAR 100 training

II. NN Class

The NN class provides two hidden layers and one FC layer network. I trained this class on the CIFAR 100 train set with the following parameters:

- 256 batch size
- 20 epochs
- 32x32 resolution
- Adam solver with learning rate 0.0001

At the end of the training I found that the accuracy of the network on the test set is of 26%.

The accuracy is not very good but this result it's not strange for this kind of Neural Network. Rather, the NN is just a (deep) Neural Network with several layers without convolution layer and pooling layer. So, we can describe it as a basic Neural Network in which no layers try to optimize the data

Accordingly with this description, a low accuracy is expected.

Fig. 1 shows the training loss and the accuracy curves related to the number of epochs.

III. CNN Class

With same parameters of section II, I trained the CNN class on the CIFAR 100 train set.

This time the result is a little better than the before: now the accuracy of the network on the test set is of 30% (it goes from 28% to 30%). I can say that this result is quite normal because of the CNN is is a neural network using convolution layer and pooling layer. These two kind of layer improves the solutions:

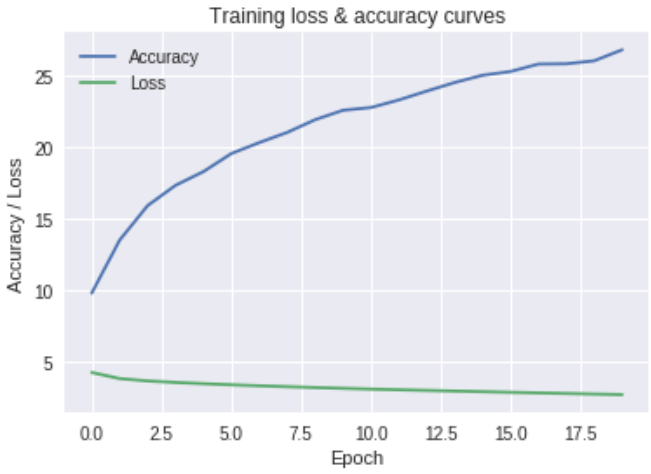

Fig. 1. Training loss and the accuracy curves of the NN class training

- The convolution layer convolves an area, or a stuck of elements in input data, into smaller area to extract feature. It is done by filtering an area, which is as same as to multiplying weights to an input data.
- The pooling layer picks an data with the highest value within an area.

These layers act to extract an important feature from the input for classification. So, the result is better than before.

IV. Convolutional Filters Values

I repeated what was done in section III but this time changing the number of convolutional filters, from $32/32/32/34$ to:

- $128/128/128/256$
- $256/256/256/512$
- $512/512/512/1024$

The first thing I noticed is the computational time: the more are the convolutional filters the more are the time to train the data. Another important thing is that I got a better accuracy increasing the number of convolutional filters. Below the result I obtained (with the corresponding convulutional filters):

- $Accuracy : 31\%, Loss : 0.160$
- $Accuracy : 34\%, Loss : 0.077$
- $Accuracy : 36\%, Loss : 0.045$

As you can see the accuracy grows as the number of convolutional filter increases. Of course there is a drawback: the performance depends (also) on the numbers of convolutional filters. For example the 3rd attempt ($512/512/512/1024$) took more or less 30/40 minutes to finish the train.

So, performance and accuracy are correlated each other and they depend on the number of convolutional filters. However,

I supposed there is like an *upper-bound* beyond which the accuracy will not improve anymore.

At this point a good question could be: *"How many filters do I need in order to relate performances and accuracy?"* I think there isn't any formal notion that relates the number of filters to performance. Its all experimental and iterative trail.

V. Improvement

I tried to do several improvements to the network adding layers and changing parameters.

As first thing I set the number of convolutional filters to $128/128/128/256$ and then, step by step, I added several modification to the network:

1) Batch Normalization at every convolutional layer
2) Batch Normalization plus a Fully Connected layer wider (8192 neurons)
3) Batch Normalization plus Dropout 0.5 on the first Fully Connected layer (this time with 4096 neurons)

With the **Batch Normalization** I noticed a good improvement of accuracy: it goes from 31% to 42%. The *Batch Normalization* greatly accelerating the learning process. It also enabled the training of deep neural networks with sigmoid activations that were previously deemed too difficult to train due to the vanishing gradient problem. The whole point of BN is to adjust the values before they hit the activation function, so as to avoid the vanishing gradient problem.

Then, I increased the number of neurons of the first Fully Connected layer from 4096 to 8192. The calculated accuracy is of 43%. The difference is not very good (about 1%) and the computational time is higher than before. With big number of **neurons**, the model will learn too much representations/features that are specific to the training data and dont generalize to data in the real-world and outside of the training set (*overfitting*).

Fig. 2 shows how change, at each epoch, the accuracy and the loss of the current network on the test set. It's clearly visible the overfittig: from the 8th to the last epoch the loss decrease but the accuracy still not change.

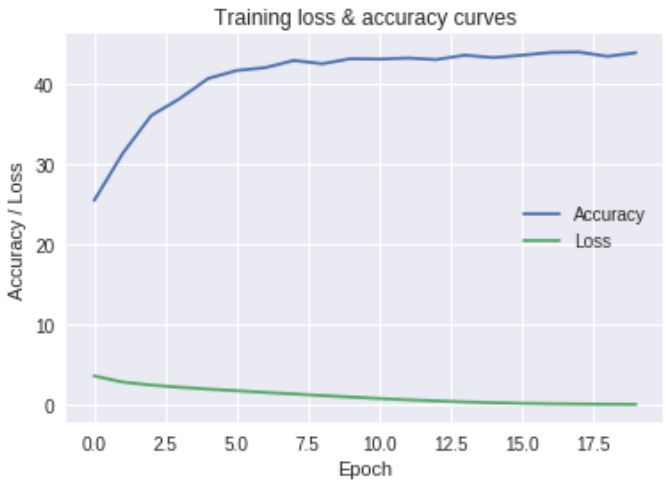

Fig. 2. Training loss and the accuracy curves of the CNN class training

Finally, I added a **Dropout** layer on the first fully connected layer. Dropout is a technique used to improve over-fit on neural networks. However, I do not notice any kind of improvement. If I use both BN and Dropout it's very difficult see an improvement (about 1%), but if I remove the BN and use only the drop then there is a clear accuracy improvement.

VI. Data Augmentation

Data augmentation is the creation of altered copies of each instance within a training data-set. In this homework will be performed the following data augmentation:

- Random Horizontal Flipping
- Random Crop

Starting from the network with $128/128/128/256$ filters I implemented the *random horizontal flipping*. The setting of the network are the same of section III, expect for the number of convolutional filters. The accuracy is better and it is equal to 36%, against the 30%. With the *random crop* the accuracy is 39%, even better than the random horizontal flip.

The improvement of the accuracy is due to the fact that through these methods we ensure that the neural network can differentiate signal from noise.

VII. ResNet18

I load the ResNet18 pretrained on ImageNet and finetune it on the CIFAR 100 training set. The network has been set with the following parameters:

- 128 batch size
- 10 epochs
- 224x224 resolution
- Adam solver with learning rate 0.0001

The accuracy found is very good and it goes from 76% to the 83%, based on the usage of the data augmentation schema. The computational time is very very slow but the accuracy is the best one.

The accuracy is very good due to the nature of ResNet: ResNet-18 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 18 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images [1].

VIII. My CNN from scratch

I did a lot test and read paper in order to obtain the best accuracy as much as possible. As first thing I decided to focus myself on the **learning rate** and the used **optimizer**. I tried different learning rate on the Adam model but then I opted to change not only the learning rate but also the model itself. I decided to implement the *Stochastic gradient descent* (SGD) optimizer: also known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization. It is called stochastic because samples are selected randomly (or shuffled) instead of as a single group

(as in standard gradient descent) or in the order they appear in the training set [4].

At this point I changed the number of **epochs**. The SGD optimizer needs an higher number of epochs in order to return the best accuracy.

Then I used **data augmentation** in order to create altered copy of each instance and improve the training: I used the *RandomHorizontalFlip()*.

After that, I tried different setting parameters for each used layer:

- Convolutional layer
  - Kernel size
  - Stride
  - Padding
- Batch Normalization
  - momentum
  - affine
- Dropout
- Number of neurons

Finally, I focused my attention on the type and the number of used layer.

The best accuracy that I obtained is of 60%.

IX. More

A. Accuracy & Loss

The lower the loss, the better a model (unless the model has over-fitted to the training data). The loss is calculated on training and validation and its interpretation is how well the model is doing for these two sets. Unlike accuracy, loss is not a percentage. It is a summation of the errors made for each example in training or validation sets.

In the case of neural networks, the loss is usually negative log-likelihood and residual sum of squares for classification and regression respectively. Then naturally, the main objective in a learning model is to reduce (minimize) the loss function's value with respect to the model's parameters by changing the weight vector values through different optimization methods, such as backpropagation in neural networks.

Loss value implies how well or poorly a certain model behaves after each iteration of optimization. Ideally, one would expect the reduction of loss after each, or several, iteration(s).

The accuracy of a model is usually determined after the model parameters are learned and fixed and no learning is taking place. Then the test samples are fed to the model and the number of mistakes (zero-one loss) the model makes are recorded, after comparison to the true targets. Then the percentage of missclassification is calculated.

There are also some subtleties while reducing the loss value. For instance, you may run into the problem of over-fitting in which the model "memorizes" the training examples and becomes kind of ineffective for the test set. Over-fitting also occurs in cases where you do not employ a regularization, you have a very complex model (the number of free parameters W is large) or the number of data points N is very low.

B. Learning Rate

The learning rate is one of the most important hyper-parameters to tune for training deep neural networks. If the learning rate is low, then training is more reliable, but optimization will take a lot of time because steps towards the minimum of the loss function are tiny.

If the learning rate is high, then training may not converge or even diverge. Weight changes can be so big that the optimizer overshoots the minimum and makes the loss worse.

The Fig. 3 (diagram) demonstrates the different scenarios one can fall into when configuring the learning rate [2].

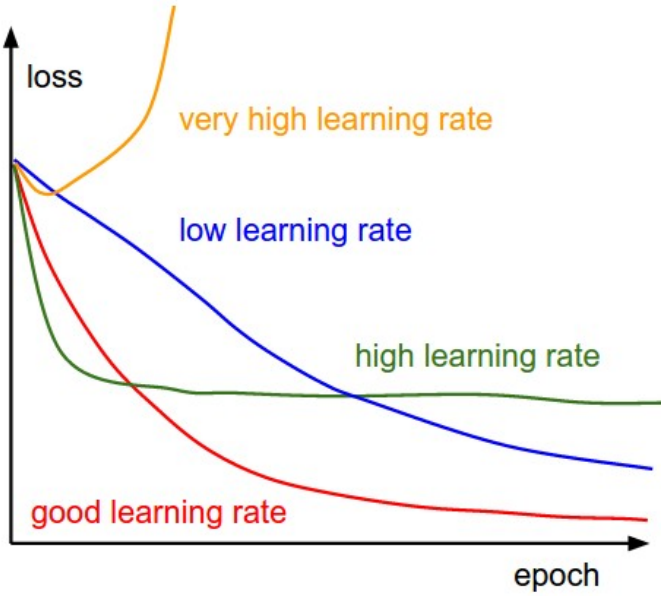

Fig. 3. Learning rate.

Furthermore, the learning rate affects how quickly our model can converge to a local minima (aka arrive at the best accuracy). Thus getting it right from the get go would mean lesser time for us to train the model.

So, how choose best learning rate? If we record the learning at each iteration and plot the learning rate (log) against loss; we will see that as the learning rate increase, there will be a point where the loss stops decreasing and starts to increase. In practice, the learning rate should ideally be somewhere to the left to the lowest point of the graph.

X. Conclusion

There are a lot of parameter that affects the performances and the quality of a Deep Learning Neural Network. To perform a NN there several strategies [3]:

- Data optimization
  - Get More Data
  - Invent More Data
  - Re-scale Your Data
  - Transform Your Data
  - Feature Selection
- Algorithm tuning
- Hyper-parameter optimization

  - Learning rates
  - Batch size and number of epochs
- Ensembles

I tried to adopt some of these strategies in order to obtain a good CNN. I did a lot of test and tried several customization. From my experience I can say that the major problem is the data-set, or rather, the more are the data the better the accuracy will be. Of course other consideration must be done, as well as the number of the convolutional layer, the usage of the batch normalization and of the dropout, number of neurons, etc. Lots of trail and error (and the study of these kind of topic) will make my knowledge better.

REFERENCES

[1] Pretrained ResNet-18 convolutional neural network. https://www.mathworks.com/help/deeplearning/ref/resnet18.html
[2] Understanding Learning Rates and How It Improves Performance in Deep Learning https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10
[3] Improve Deep Learning Performance https://machinelearningmastery.com/improve-deep-learning-performance/
[4] From Wikipedia, Stochastic gradient descent https://en.wikipedia.org/wiki/Stochastic_gradient_descent