Assignment 3 - Deep Networks

Davide Pedranz (189295)
davide.pedranz@studenti.unitn.it

***Abstract*—Deep Networks are state of the art techniques for many difficult problems in Machine Learning, widely used for many different tasks, from image recognition to machine translation. In this assignment, we will use TensorFlow™ to construct and train Deep Networks with different architectures to recognize the handwritten digits of the MNIST dataset.**

I. Introduction

Neural Network is a powerful learning algorithm widely used in Machine Learning to solve complex tasks like handwritten text recognition, computer vision, speech recognition and machine translation. Indeed, Neural Networks are able to represent very complex functions, which make them very adaptable to many different problem.

Depending on the specific task to solve, Neural Networks with different structures can be built and trained. In this assignment, we will start from the architecture suggested in the TensorFlow™ tutorial *Deep MNIST for Experts* [1] to recognize the handwritten digits of the MNIST [2] dataset, then remove one layer at a time and study the behaviour of the new network. We will explain the structure of each network and the function of each layer. Finally, we will compare the performances of the different networks.

There are many different libraries available to construct and train Neural Networks. In this assignment, we will use TensorFlow™, an open source library for numerical computation using data flow graphs. It allows to easily define the structure of a Neural Network and efficiently train it. It was originally developed by the Google Brain Team and is now widely used by many students, researches and companies across the world.

II. Neural Networks

The building block of each Neural Network is the perceptron. A perceptron is a unit of computation that takes many inputs and compute an output, which is usually the result of the application of some activation function to the weighted sum of the inputs. In other words, a perceptron computes the dot product between the inputs and some learned weights, i.e. it measures the similarity between the input and some internal representation.

Perceptron are organized in layers, which in turn form a network. Each network can have a different number of layers and each layer can have a different number of perceptrons. Some of the main types of layers include "Fully Connected", "Softmax", "Convolutional" and "Pooling".

[1] https://www.tensorflow.org/versions/master/tutorials/mnist/pros/

[2] The MNIST dataset is available at: http://yann.lecun.com/exdb/mnist/

In the following sections, we will examine how this layers are build and what is their function in a Deep Network for handwritten digits recognition.

III. Fully Connected Layer

In a Fully Connected Layer each perceptron is connected to each of the outputs of the previous layer. Both the inputs and the outputs are organized as a one dimensional array.

The output of each perceptron is computed as an activation function of the weighted sum of all the inputs plus a bias. The activation function aims to break the linearity of perceptron, allowing the successive layers to build increasingly complex non-linear features of the original input. Common choices for the activation are the Sigmoid and the Rectified Linear Unit (ReLU) function.

The Sigmoid function as the form:

$$f(x) = \frac{1}{1 + e^{-x}}$$

and can be seen in Figure 1. It has the nice property that it is almost linear and quite steep in an interval of zero, which makes the learning process very quick. On the other side, is saturates very quickly as soon as the input moves away from zero. This can accidentally "kill" the perceptron, since the backpropagation algorithm multiplies the error from the following layer to the local gradient to calculate the weights update [1].

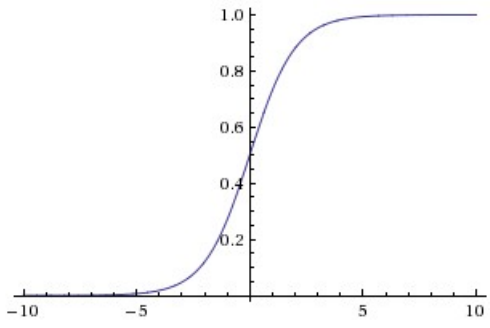

Figure 1. The Sigmoid function.

A popular alternative is the ReLU function. ReLU is defined as follows:

$$f(x) = max(0, x)$$

and can be saw in Figure 2.

ReLU has been shown to greatly accelerate the learning process compared to the Sigmoid function [2]. Another advantage

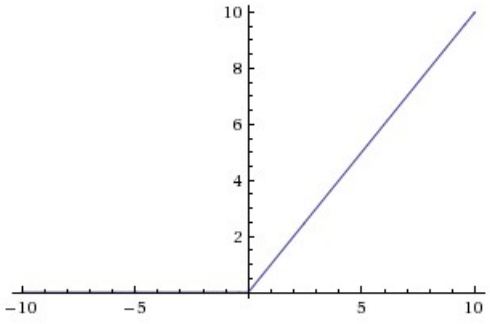

Figure 2. The ReLU function.

is that it does not require to compute complex functions like exponentiation, but simply a threshold. Unfortunately, ReLU units suffer of the same "dying" problem of Sigmoid ones. With high learning rates, the weights of the perceptron can be updated in such a way that its gradient become zero (and will never be updated again). To reduce this problem, a proper choice the learning rate is necessary [1].

IV. Softmax

Softmax is a generalization of the Logistic Regression classifier to multiclass problems. It computes the best class by applying a cross-entropy loss function to a weighted sum of the inputs (dot product of inputs and learned weights). More formally, it computes an evidence score for each class i

$$evidence_i = \sum_j W_{ij} x_j + b_i,$$

then it normalizes the scores to a probability distribution over the i possible values

$$softmax_i = \frac{e^{evidence_i}}{\sum_j e^{evidence_j}}.$$

Softmax is commonly used as the last layer of a Deep Network for classification problems. Please note that in this case the activation function is omitted since an activation value is already computed by the Softmax function.

V. Convolutional Networks

In traditional Neural Networks, the input is encoded as a vector of raw numbers which is given as input to the first layer. This approach does not scale well to images. Let's consider a 200x200 pixel image with 3 colors. Each neuron of the first layer will have $200 * 200 * 3 = 120,000$ weights to learn. And this takes into account only a single neuron in the first layer. The main problem here is that we do not consider the structure of the input, which is lost during its encoding into a one dimensional array. In case, for instance, of image recognitions, the actual position of the pixels is very important to understand its meaning.

Convolutional Neural Networks try to exploit the structure of the input to optimize the learning of images with the introduction of Convolutional layers and Pooling layers in addition to traditional Fully Connected ones. The new types of layer have the scope to extract features from the images, so that the traditional Fully Connected layers can work on meaningful features instead of raw pixel values. In both Convolutional and Pooling layers, perceptrons are organized in three dimensions: width, height and depth. Depth refers to different channels of the image representation, for instance RGB for colors.

VI. Convolutional Layer

A Convolutional layer consist of a set of learnable filters. Each filter is spatially small, but extends through the full depth of the input. The patched is moved on all the image surface and creates a 2-dimensional feature map. The results of the computation of each filter are then stacked on each other and form the input for the next layer of the network. The output of each filter is computed, as for normal perceptrons, as the result of an activation function of the dot product of the inputs and the internal learned weights. It is important to underline that the weights are unique for each filter, which means that the same values are used for each piece of image input for a single filter. This process can be seen in the Figure 3.

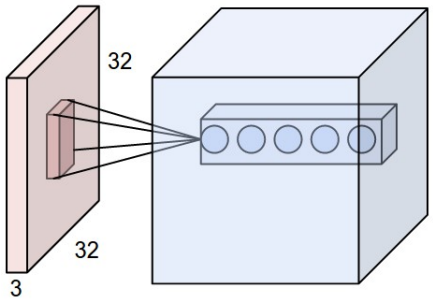

Figure 3. Illustration[4] of the computation of a Convolutional layer. In this example, 5 different filters (in blue) are computed on the original image (in red). Each filter is computed by moving a patch (the small red square) around the input image.

A Convolutional layer has some hyperparameters, which determine how the filters are applied to the input. The main hyperparameters are:

- **Receptive Field**, which represent the size of the filter. In other words, it determines the dimension of the patch used to compute the filter on the input image. Typical sizes range from 3x3 to 7x7 pixels.
- **Depth**, which determines the number of filter to be applied. Sometimes this is also called fibre.
- **Stride**, which determines the amount of pixel the patch is moved on the image each time. Common values are 1 and 2 pixel.

VII. Pooling Layer

Convolutional layers can generate outputs that are bigger than the original input by computing a high number of filters. Pooling layers are commonly inserted in-between successive Convolutional layers to progressively reduce the spatial size of

[4]Illustration taken from https://cs231n.github.io/convolutional-networks/

the representation, in order to reduce the amount of parameters and computation in the network.

A Pooling layer re-sizes the input spatially, while it does not change its depth. As for the case of the Convolutional layer, a patch is moved around the input and some aggregation function is computed to reduce the inputs to a single value. The most common operator is the max function, even though other operations are possible [1].

A Pooling layer requires to set some hyperparameters:

- **Spatial Extent**, which determines the dimension of the patch used to compute the aggregation function on the input image (similarly to the Receptive Field for Convolutional layers).
- **Stride**, which determines the amount of pixel the patch is moved on the image each time.

A typical setting is to use max pooling with a spatial extent of 2x2 pixels and stride of 2, as shown in Figure 4.

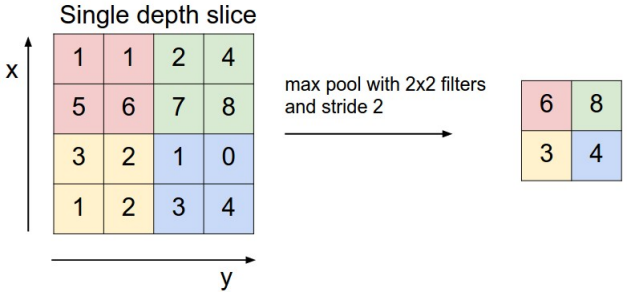

Figure 4. Illustration[6] of the pooling process. In this example, a patch of size 2x2 pixel is moved of 2 pixel at a time. The value computed is the maximum between the current values of the patch.

VIII. Dropout

Neural networks, like many other learning algorithms, can suffer from overfitting the data. A simple technique to avoid overfitting in deep networks is the Dropout.

Dropout is an extremely simple and effective technique recently introduced by Srivastava et al. [3]. It consists in randomly turning off perceptrons during the training phase: at each epoch of training, perceptrons are kept active only with some probability p (a hyperparameter), otherwise they are set to zero. The process can be see in Figure 5, taken from the original paper. Dropout is not applied during test and allows to ensemble the different parameters learned by some group of perceptrons during the training.

IX. MNIST Handwritten Digits

The MNIST is a database of handwritten digits. It is composed by a training set of 60,000 examples and a test set of 10,000 examples, both available on the website of Yann LeCun[7]. The digits have been size-normalized to 20x20 pixel and centered in a fixed-size image of 28x28 pixel. The images are in black and white, so their depth is equal to 1.

[6]Illustration taken from https://cs231n.github.io/convolutional-networks/

[7]http://yann.lecun.com/exdb/mnist/

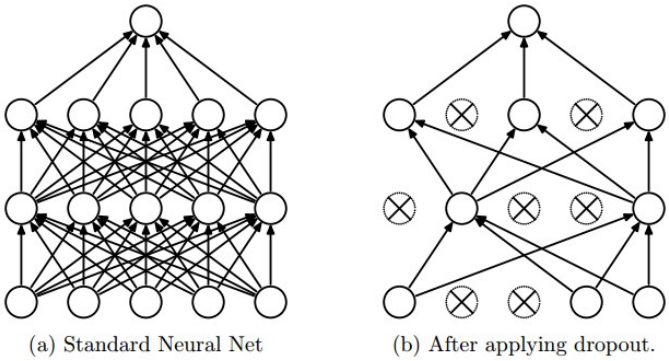

Figure 5. Changes in the neural network after applying Dropout.

X. Deep Networks Models

This section describes and compare the original model proposed by the TensorFlow™ tutorial and three possible modifications. All network are initialized using the same seed for random number generation in order to obtain results with are comparable. Also the selection of the initial subsets of the training and test sets are done with the same fixed seed.

A. Original Model

The original network proposed has the following architecture: a Convolutional layer, a Pooling layer, another Convolutional layer, another Pooling layer, a Fully Connected layer with ReLU activation function and Dropout and a Softmax output layer. The first Convolutional layer computes 32 filters, while the second one 64. Both Pooling layers reduces the dimension of the image by applying max pooling with 2x2 pixel patches and stride 2. Figure 6 shows the structure of the resulting network.

The Convolutional layers are used to extract features from the images, which are then used by the Fully Connected layer and the Softmax layer to compute the most probable class. Pooling layers and Dropout are used to prevent the network to overfit the training data. Note that the image is reshaped into a one dimensional array after the second Pooling layer, in order to be processed by the Fully Connected layer.

At each training step, the network is trained with a batch of 50 images, randomly selected from the train dataset. The network is trained to minimize the generalized cross entropy function for a multiclass problems. After 30000 epochs of training, the network has an accuracy of 99.22% on the complete test set.

B. Model 1

The first modification to the original model consist in removing the first level of Convolution and Pooling. The second Convolutional layer is now directly applied to the original image. The rest of the network and all hyperparameters are unchanged.

The performance of the network is closed to the original one, with an accuracy of 99.07% on the test set.

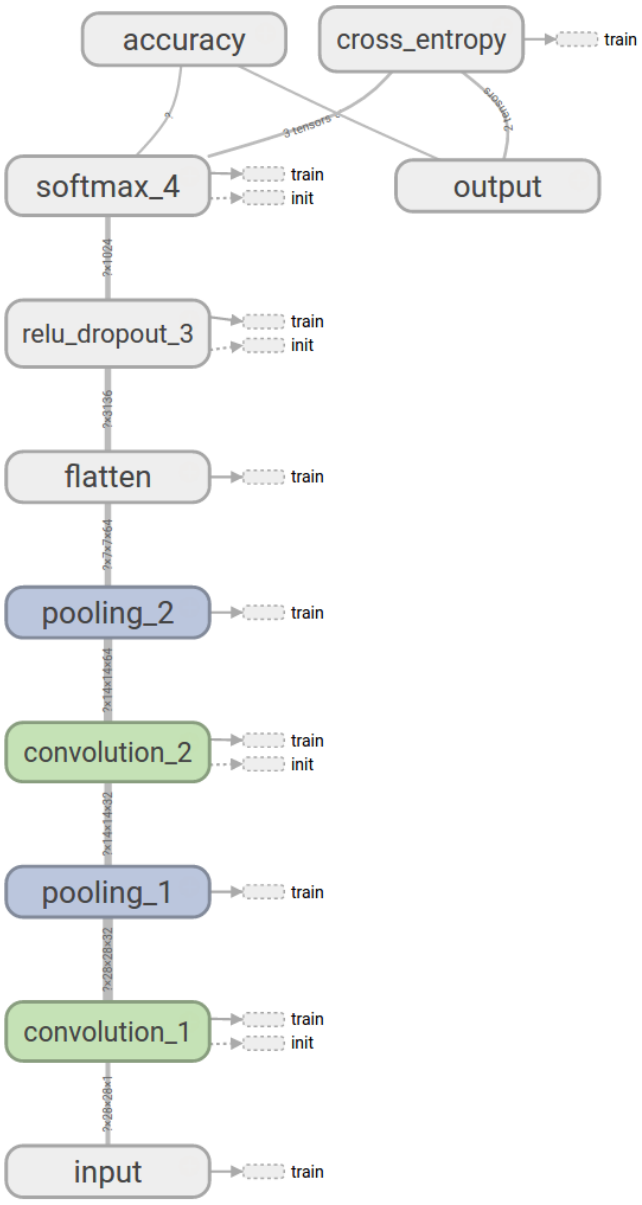

Figure 6. Structure of the original Deep Network used for the MNIST handwritten digits recognition, visualized using the TensorBoard tool.

C. Model 2

The second modification is very similar to the previous one. Instead of removing the first one, we are removing the second Convolution and Pooling layers. The resulting network is almost identical to Model 1, but with the difference that the layer is computing 32 filters instead of 64. The accuracy of this model is 99.02%, only slightly worse than model 1.

D. Model 3

The third modification consist in removing the Fully Connected layer. The accuracy of this network is slightly worse then the other models and scores 99.03%. This is probably due to the absence of any fully connected layer that properly combines the features extracted by the convolutional layer.

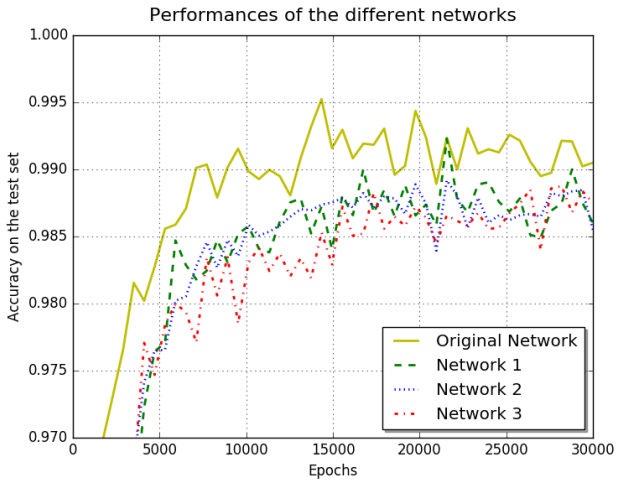

Figure 7. The graph shows the performances of the different networks on the test set over epochs of training. The plotted accuracies are measured on a random fixed subset of 2000 elements of the test set, equal for all the networks.

Table I
SUMMARY OF THE ACCURACIES ON THE TEST SET OF THE DIFFERENT NETWORKS AT DIFFERENT EPOCHS

Network	Epoch 10000	Epoch 20000	Epoch 30000
Original	99.11 %	99.21 %	99.22 %
Model 1	98.68 %	98.94 %	99.07 %
Model 2	98.75 %	98.96 %	99.02 %
Model 3	98.51 %	98.86 %	99.03 %

XI. Conclusion

None of the proposed modification achieved to improve the accuracy of the original network. The results are summarized in table Table I. The two layers of convolution were able to extract better features then the single layers of models 1 and 2. Both the final performances and their evolution over the training epochs (Figure 7) of all modified models are very closed to each other. Model 1 seems to perform slightly better then the other two, even though some more experiments are required in order to get a confident result. The worse performances of model 2 and 3 suggest the need of to compute very complex features in order to achieve a very high score on the image classification.

References

[1] Andrej Karpathy, "Cs231n: Convolutional neural networks for visual recognition," https://cs231n.github.io/.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[3] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.