# Machine Learning, assignment no. 3

ALAN RAMPONI, student ID: 179850, University of Trento

This assignment is about the performance comparison of a deep convolutional network in classifying the MNIST handwritten digits dataset when removing layers. After loading training and test data, four layers was built in order to improve the overall classification capability of the neural network. The first two are convolutional layers with max pooling, the third one is a fully-connected layer with dropout and the last one is an output layer that exploits a softmax regression model to convert evidence to probability. The optimization was performed using an adaptive gradient algorithm with learning rate 1e-4 in order to minimize the cross-entropy cost function. In the end, the evaluation was performed using the accuracy measure to determine the fraction of the predicted labels that matched the expected labels on test data.

Categories and Subject Descriptors: [**Computing Methodologies**]: Machine Learning—*Neural networks*; [**Applied Computing**]: Document Management and Text Processing—*Optical character recognition*

## 1. INTRODUCTION

In Machine Learning, a convolutional neural network is a biologically-inspired variant of MLP designed to use minimal amounts of preprocessing. It represents a particular type of feed-forward artificial neural network where the neurons are tiled in such a way that they respond to overlapping regions in the visual field. In this assignment is described the methodology used in constructing this very deep architecture, from the input layer to the output layer, using TensorFlow, a smart open source software library for machine learning released by Google in November, 2015. By removing a layer at a time, it was possible to compare the accuracy performances of each resulting deep convolutional network in predicting the MNIST handwritten digits dataset.

## 2. THE DATA AND ITS SAMPLING

The dataset in which the convolutional neural network was trained is MNIST [LeCun 2015], a simple computer vision dataset that consists of images of handwritten digits associated with their labels. Each image is 28 pixels by 28 pixels, so it is possible to interpret this as a big array of numbers and to flatten it into a vector of 784 numbers. Each label is a number between 0 and 9, describing which digit a given image is of, represented by a one-hot vector[1].

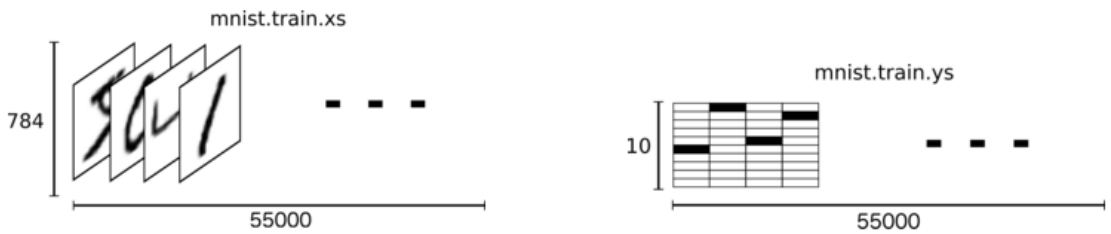


Fig. 1. Example of the data structures in the training set for the images and for the labels respectively.

[1]A one-hot vector is a vector which is 0 in most dimensions, and 1 in a single dimension.

Once the data was downloaded taking advantage of the $input\_data.py$ script, it was possible to build, train and test a deep convolutional network using TensorFlow [Google 2015]. The resulting downloaded data consists of three parts: 55000 data points of training data ($mnist.train$), 10000 data points of test data ($mnist.test$) and 5000 data points of validation data ($mnist.validation$). Thus, given $x$ the training set, the test set or the validation set, $x$ is composed by $mnist.x.xs$ (i.e. a tensor[2] of images with a shape [$\#images$, 784] that indexes the images and their pixel intensities) and $mnist.x.ys$ (i.e. a tensor of labels with a shape [$\#images$, 10] that provides the expected label of each image using a one-hot vector encoding). A graphic representation of the explained structure is provided in Figure 1.

## 3. THE CLASSIFICATION TASK

Since a very simple neural network model is almost embarrassingly bad in accuracy on classifying the MNIST dataset ($\sim$91%), a small convolutional neural network was built. The structure of the designed deep architecture is composed by the following layers:

— **Layer 1**: a first convolutional layer with max pooling;

— **Layer 2**: a second convolutional layer with max pooling;

— **Layer 3**: a rectified linear unit (*ReLU*) layer with dropout;

— **Layer 4**: a final softmax regression layer.

The different peculiarities of each building block of the whole neural network are the following:

### 3.1 The convolutional layer

The convolutional layer is the core building block of a convolutional neural network. The parameters of that layer are learnable filters with a small receptive field that extend through the full depth of the input volume. Thus, each filter is convolved across the width and height of the input volume computing the dot product between entries of the filter and the input, and producing a 2-D activation map of that filter. In this way, the network learns filters that activate when they see some specific type of feature at some spatial position in the input. Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer, that can be interpreted as holding neurons arranged in a 3-D volume. In depth, three hyperparameters control the size of the output volume: the depth, stride and zero-padding [Karpathy 2015].

In this assignment the convolution operation was performed using a sliding window stride of 1 and it was setted to be 0-padded so that the output was the same size as the input.

### 3.2 The max pooling layer

Successive convolutional layers in a convolutional neural network architecture are usually interleaved by max pooling layers. They are a form of non-linear down-sampling that partition the input image into a set of non-overlapping rectangles and, for each such sub-region, output the maximum. Thus, once a feature has been found, its exact location isn't as important as its rough location relative to other features. In this way, this very layer progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting [Karpathy 2015].

In this assignment the max pooling operation was performed over 2x2 blocks using a window (for each dimension of the input) of size [1,2,2,1] and a sliding window stride (for each dimension of the input) of [1,2,2,1], so a padding algorithm shrinked the dimension according to them.

[2]A tensor is a typed $n$-dimensional array.

### 3.3 The ReLU layer

The rectified linear unit (*ReLU*) layer is a layer of neurons that applies a non-saturating activation function that increases the non-linear properties of a decision function and of the overall network without affecting the receptive fields of the convolutional layer. Thus, it basically applies elementwise non-linearity using the following non-saturating activation function:

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

Then, in this assignment the ReLU was placed within the third layer, after both the layers that perform convolution operation and max pooling operation.

### 3.4 The dropout layer

The dropout layer is a layer explicitly designed to avoid overfitting by randomly excluding neurons at each training stage. In fact, a fully connected layer occupies most of the parameters. Thus, individual nodes are either dropped out with all their incoming and outgoing edges or kept with probability $1 - p$ or $p$ respectively (usually $p$ is 0.5 for hidden nodes, and it's higher for input nodes). In this way, at each stage only a reduced network is trained. Concretely, at testing time only a single network need to be tested; thus, an approximation is searched by using the full network with each node's output weighted by a factor of $p$, so the expected value of the output of any node is the same as in the training stages.

In this assignment the dropout layer was placed just after the ReLU operation within the third layer.

### 3.5 The softmax regression layer

The softmax regression layer uses a softmax regression model that generalizes logistic regression to multi-class classification problems (i.e. problems having a number of classes $k > 2$). Thus, given a test input, it estimates the probability of the class label taking on each of the $k$ different possible values. In this way, it outputs a $k$-dimensional vector (whose elements sum to 1) giving $k$ estimated probabilities.

More precisely, a softmax regression consists of two steps: first is added up the evidence of the input being in certain classes, and then that evidence is converted into probabilities. To tally up the evidence that a given image is in a particular class, a weighted sum of the pixel intensities is computed along with the addition of some extra evidence (called *bias*). Thus, the evidence for a class $i$ given an input $x$ and an index $j$ for summing over the pixels is the following:

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

At this point, the evidence tallies are converted into predicted probabilities $\hat{y}$ using the softmax function, that is serving as an activation function, shaping the output of the linear function into a probability distribution over $k$ cases:

$$\hat{y} = softmax(evidence) = normalize(exp(evidence))$$

Thus, it exponentiates its inputs (i.e. one more unit of evidence increases the weigth multiplicatively) and then it normalizes them (i.e. they add up to one, forming a valid probability distribution). Formally, the softmax function is defined as follows:

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

Then, in this assignment the softmax regression was useful in order to convert the output of the linear function into a probability distribution over $k$ = 10 cases, i.e. the class of digits.

Finally, a cross-entropy cost function to be minimized during training was specified. For this purpose, an automatic differentiation to find the gradients of the cost with respect to each of the variables was computed using an adaptive gradient algorithm with learning rate 1e-4 in order to descend the cross-entropy.

## 4. THE EVALUATION

In order to evaluate the performances of the convolutional neural network in classifying the MNIST dataset when removing layers an accuracy measure to determine the fraction of the predicted labels that matched the expected labels on test data was performed for each variant of the original network. In depth, firstly it was used a function that gave us the index of the highest entry in the tensor along some axis. Secondly, a comparison between the predicted label and the true label was performed, giving a boolean for each comparison and putting it into a list. Lastly, the accuracy of the whole prediction was computed by converting the *True* values of the list to 1 and the *False* values of the list to 0, and then by taking the mean. For example:

$$[True, False, True, True, True] \Rightarrow [1, 0, 1, 1, 1] \qquad Accuracy = \frac{1+0+1+1+1}{5} = \frac{4}{5} = 0.8$$

In Table I are presented the accuracy results of each convolutional neural network that was tested. For this purpose three different scripts were created: $model\_no1.py$, $model\_no2.py$ and $model\_no3.py$.

Table I. Test accuracy results of each variant of the original CNN.

| **deep architecture composition** | **TEST ACCURACY** |
|---|---|
| convolutional neural network **without layer 1** | 0.9898 |
| convolutional neural network **without layer 2** | 0.9900 |
| convolutional neural network **without layer 3** | 0.9880 |
| convolutional neural network **with all the layers** | 0.9919 |

## 5. DISCUSSION

By removing one layer at a time, slightly different test accuracy results were obtained. In particular, the removing of any level leads to worse results compared to the original full convolutional neural network. As regards the specific cases, the worst deep architecture is the one without the third layer (i.e. the layer that performs the ReLU with dropout), while the best convolutional neural network among the three incomplete deep architectures is the one without the second layer (i.e. the innermost between the two layers that perform the convolutional operation with max pooling).

In the end, these results lead to think that the intermediate hidden layers may be less important with respect to the layer just before the readout, i.e. the one that performs the softmax operation. Nevertheless, it is crucial to have more layers as possible in order to perform a good classification.

## REFERENCES

Google. 2015. TensorFlow: an Open Source Software Library for Machine Intelligence. Python and C++ libraries. (2015). https://www.tensorflow.org/

Andrej Karpathy. 2015. Convolutional Neural Networks for Visual Recognition. Online notes of the Stanford CS class CS231n. (2015). http://cs231n.github.io/convolutional-networks/

Yann LeCun. 2015. The MNIST database of handwritten digits. Four compressed files. (2015). http://yann.lecun.com/exdb/mnist/