Luca Rondano, 2022

Hand-written digits classification

Abstract: Image recognition is one of the most common applications of machine learning. It is a complex task in which humans are extremely good. Nowadays, clever use of neural networks and big dataset of labeled images allowed the implementation of machines capable of classifying animals and objects with humanlike precision. A basic result is the recognition of handwritten digits.

In this project I used the MNIST dataset to train a simple neural network to recognize handwritten digits. It turns out that for this task a shallow network is enough to get a high accuracy.

The MNIST database stands for *Modified National Institute of Standards and Technology database*. It is a large database of handwritten digits taken from American Census Bureau employees and high school students that was normalized to fit into a 28x28 pixel bounding box for use in machine learning projects.

I found the inspiration (and a lot more!) for this project in the free online book *Neural Networks and Deep Learning* by Michael Nielsen (http://neuralnetworksanddeeplearning.com/).

Report: The dataset I used is the MNIST database, containing 70,000 images of handwritten digits (50,000 for training, 10,000 in the validation set and 10,000 in the test set) labeled with their corresponding digit value. Each image is a 28x28 black and white scan of a handwritten digit. To be precise, every sample in the dataset is a list of 28x28=784 values between 0 and 1, each corresponding to the scale of gray of each pixel, in order from the top-right pixel to the bottom-left one.

The labels are 10-dimensional unit vectors with a 1 in the j^{th} position, where j is the digit corresponding to the image, and zeroes elsewhere, i.e., a handwritten “2” is labeled by the vector (0,0,1,0,0,0,0,0,0,0).

To train the network I used stochastic gradient descent. Classical gradient descent would have taken too much time on my laptop, given the large amount of training data. Luckily, with an appropriate choice of the hyperparameters (learning rate, number of epochs, size of the minibatches), a feed-forward network with only one hidden layer was enough to yield an accuracy over 95% on the test set.

To be precise, I initially used a shallow feed-forward neural network with 30 neurons in the hidden layer and the quadratic cost function. The input layer takes the 784 pixels of each image, and the output layer returns a 10-dimensional vector with values between 0 and 1. The value closest to 1 is the predicted classification of the handwritten digit. For the minibatch size I have chosen 10, as suggested in *Neural Networks and Deep Learning*.

The learning rate that gives best results faster seems to be $\eta=3.0$. If I choose η too small, then the number of epochs required to reach a high accuracy increases. If I choose η too big, then the algorithm becomes bad at the task. After 30 epochs or so, the accuracy reached is a little more

than 95% and continuing the training does not seem to improve it. Reducing the learning rate to 1 at this point, improved the result to 95.5%.

The initialization of weights and biases is randomized using a Normal Gaussian distribution therefore there is an element of chance to the results we obtain every time. If the initial parameters are too bad, the network will take a very long time to train. To get the accuracies in this report, I let the networks train three or four times and then I have written down the best results.

Notice that I decided to stop the training after 30 epochs because the accuracy seemed to become stable after that point. More training at that point seemed pointless, but at least it did not look like it was overfitting, otherwise I probably would have noticed a drop in the accuracy.

At this point I had some confidence in the selected hyperparameters, and I tried increasing the number of neurons to 100: the algorithm reached an accuracy of 96.5%. I also tried using a deep network with two hidden layers, but there were no evident improvements with respect to the shallow network in the first 30 epochs.

Out of curiosity, I tried halving the training set to see how the network would perform and the accuracy remained almost equally good. Maybe the training set is very big for this task and that is enough to get very good results, even if we use basic machine learning techniques.

Up to this point I have used a slightly modified version of a script available in *Neural Networks and Deep Learning*. But I have made some modifications for the next part of this project.

Some marginally better results occurred when I changed the cost function with the cross-entropy; with a smaller learning rate (η=0.5) and 30 neurons, the network reached 95.5% within 30 epochs. And increasing the number of neurons to 100, the network with the cross-entropy cost function reached an accuracy of 96.8%.

To choose the new learning rate, I run some trainings with different values of eta to compute the cost on the validation set. A general rule I have found in the book is to choose eta six times smaller when using the cross-entropy, so I started with η=0.5. This choice decreased the cost fast but struggled to converge. With η=0.1, there was convergence but required more time. With η=2.0, the cost remained too high and had steep and sudden changes. Therefore, it seemed reasonable to use η=0.5 for the first 30 epochs and then eventually change eta to 0.1 later to increase the precision.

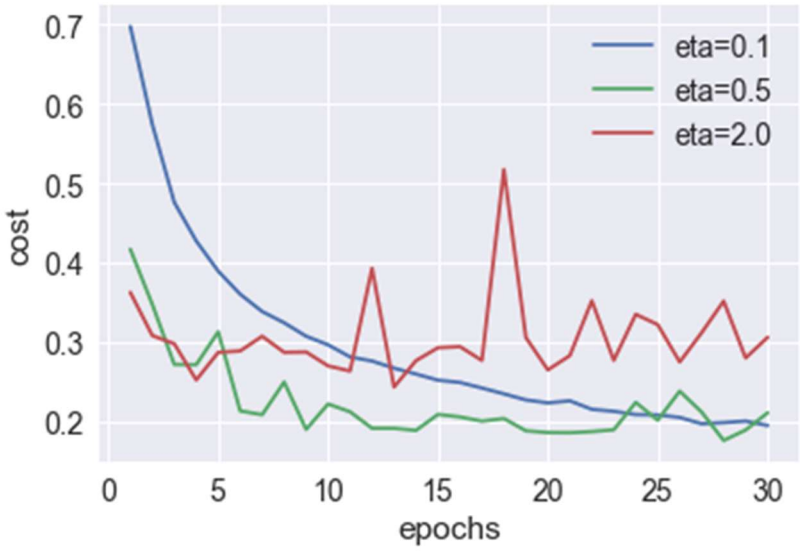

Image 1: cross-entropy cost function change over 30 epochs, evaluated on the validation set.

To improve further, I tried adding a regularization term to the cost function. In particular, adding the L2-regularization to the cross-entropy resulted very effective: the shallow network with 100 neurons, cross-entropy cost function, L2-regularization with λ=5 and learning rate η=0.5 reached the accuracy of 97.8% within 30 epochs. And doing more training with learning rate η=0.1 improved the accuracy to almost 98.3% in the next 30 epochs. I also tried with the L1-regularization, and the result was just a little worse than before.

I tried to use the same hyperparameters on a deep network with two hidden layers and achieved the best accuracy of all, almost 98,4%. I did not try to use more hidden layers because the training started to become too time consuming.

I always used the test set to evaluate accuracy. Therefore, it was possible that the good performance of my model was given by a bias in the test set. If that was the case, then using a different test set could result in a very different accuracy.

For this reason, I then tried evaluating the accuracy of the network on the validation set - found in the MNIST dataset - over the span of 100 epochs. The new results confirmed the old ones: the accuracy increases for about 30 epochs to about the same level as before and then begins to stagnate. For this operation, I used the model with the cross-entropy and the L2-regularization (with hyperparameters η=0.5 and λ=5).

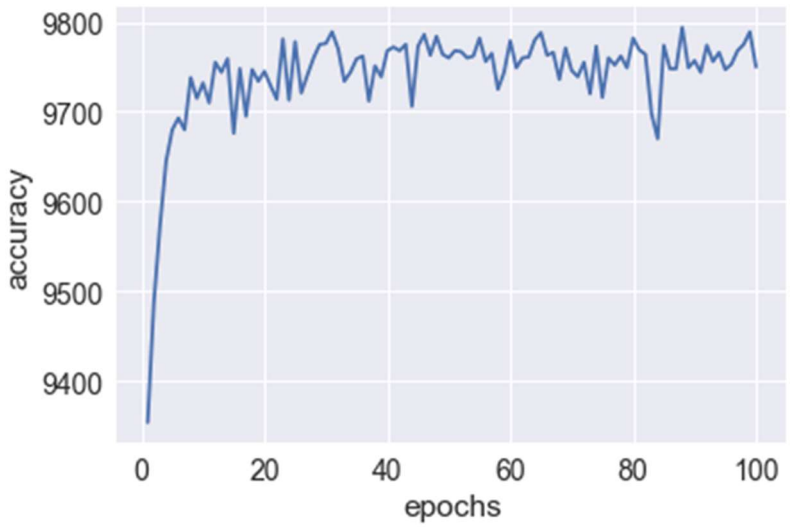

Image 2: accuracy of the network over 100 epochs.

Possible improvements: A first improvement could be obtained by adopting different regularization techniques, like dropout, but the L2-regularization seemed already pretty good at that. Then there is an argument to make for deep networks: I had some luck when using two hidden layers, but a deeper network might be even better at the task. I am not sure though, because the task seemed simple enough to be managed easily by one or two hidden layers. Expanding the training set would also probably improve the performance of the network, but producing new data takes a lot of effort and time. A clever way to increase the size of the training set is to slightly rotate and translate the images we already have and add those transformed images into the training set. In a paper by Patrice Simard, Dave Steinkraus, and John Platt (Best

Practices for Convolutional Neural Networks Applied to Visual Document Analysis, 2003), they also experimented with what they called "elastic distortions", a special type of image distortion intended to emulate the random oscillations found in hand muscles. By using the elastic distortions to expand the MNIST dataset they achieved the accuracy of 99.3%.

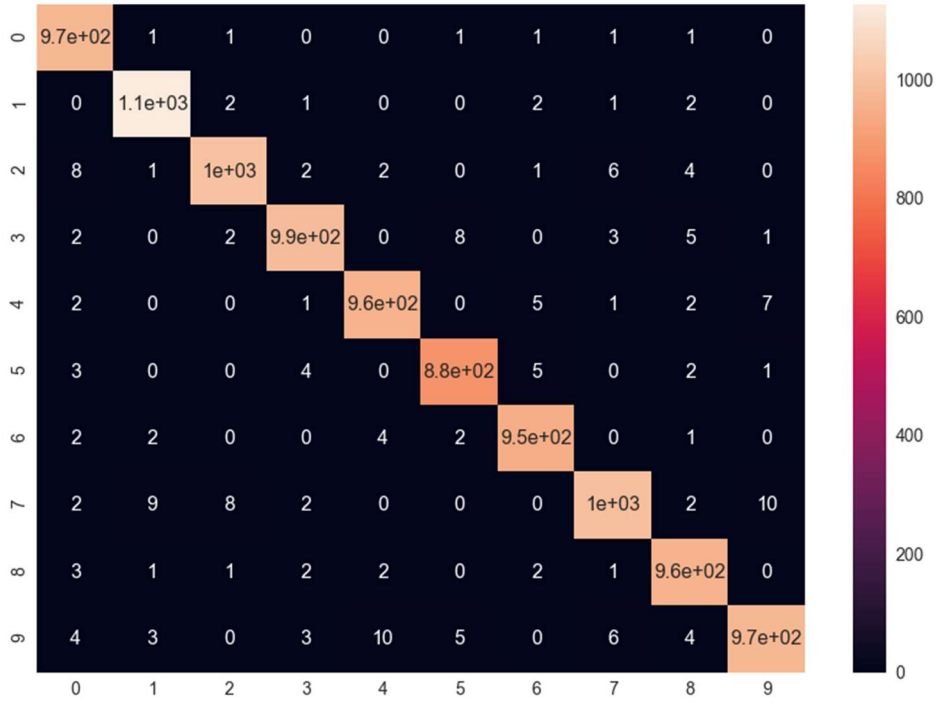

Image 3: confusion matrix from the best network I obtained, which is a network with 2 hidden layers, L2-regularization and cross-entropy cost function.

Cost function	**Number of hidden layers**	**Number of neurons**	**Hyperparameters**	**Best accuracy within 30 epochs**
quadratic	1	30	η=3.0	9544/10000
quadratic	1	100	η=3.0	9649/10000
cross-entropy	1	30	η=0.5	9549/10000
cross-entropy	1	100	η=0.5	9680/10000
cross-entropy + L1	1	100	η=0.5 λ=5	9727/10000
cross-entropy + L2	1	100	η=0.5 λ=5	9782/10000
fine-tuning of the above network	1	100	η=0.1 λ=5	9828/10000
cross-entropy + L2	2	100 100	η=0.5 λ=5	9787/10000
fine-tuning of the above network	2	100 100	η=0.1 λ=5	**9838**/10000

Image 4: table with the accuracies reached by each version of the network