9 Image classification

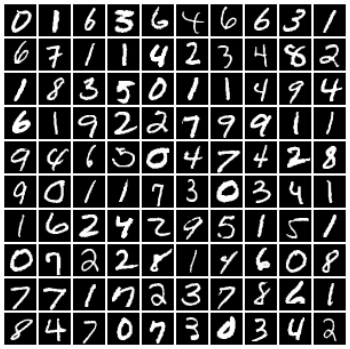

Figure 9.1: Example of the MNIST images.

[1] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009

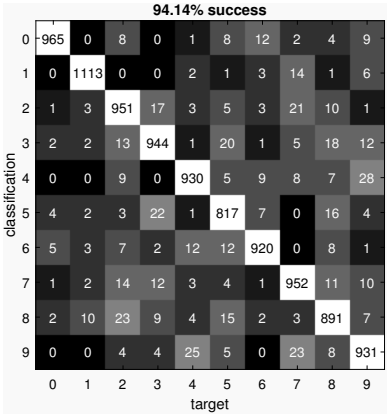

Figure 9.2: Table showing a classification performance example of a classification of the MNIST handwritten dataset.

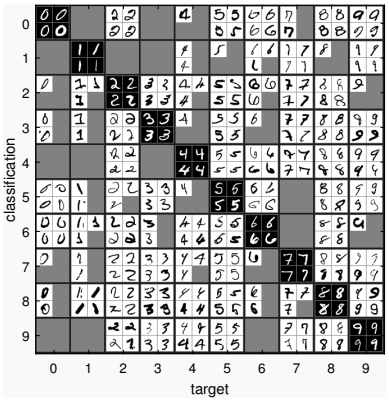

Figure 9.3: Examples of classified handwritten digits and misclassified digits.

In this exercise you should build neural networks for classifying images. We will work with the relatively easy MNIST image data set and the more challenging CIFAR-10[1] data set. Performance will be measured in number of misclassified images, and the goal is to obtain the result with the least number of misclassified images.

The MNIST dataset contains images of handwritten digits of 28×28 pixels as shown in Figure 9.1. Ground truth class labels are given together with the images. The ground truth is 10 dimensional vectors with 1 in the dimension representing the class containing the digit and zeros elsewhere. MNIST contains 60000 images for training your network. If you use all 60000 images for training your network, you might overfit your model, and to choose when to stop training you can split the data into a training and validation set. You can for example use 50000 images for training the network and reserve 10000 for validating it. By classifying the validation images, you can measure if you have overfitted your model, which is seen by a drop in classification performance of the validation data. In addition there are 10000 images for testing, but these should only be used for evaluating the performance of your networks after they have been trained. Figure 9.2 and 9.3 show a classification performance example and examples of correctly and misclassified digits.

The CIFAR-10 data set is similar to MNIST and contains 50000 small images for training and 10000 images for testing. The images are, however, 32×32 RGB color images with significantly higher variation in appearance.

The rules for the competition are:

1. Implement your own neural network for classifying the MNIST images.

2. Train the neural network on a part of the training data (e.g. 50000 images) and validate it on another part (e.g. the remaining 10000 images).

3. When you are satisfied with the obtained result – upload the trained

network together with Matlab or Python code for running it.

4. Hand in a description of your network and a guide on how to run your code

5. Be a fair player and do not use the MNIST and CIFAR-10 test data for training your networks (can be found on the internet, but do not use it!).

6. Do an effort in making the code efficient.

9.1 *Modifications of your network*

The classification will be using a fully connected feed forward neural network, similar to the one you implemented last exercise. But in contrast to last exercise, where data was points in two dimensions, you now have images. We treat these as vectors, so even though MNIST images are only 28×28 pixels the vector representation is 784 dimensions. Therefore, the network should take in 784 dimensional vectors and return a 10 dimensional vector for classifying the digits 0 to 9. For the CIFAR-10 data, the input vector becomes 3072 dimensional ($32 \times 32 \times 3$). You can use the book on deep learning by Goodfellow et al. [2] to get ideas for this exercise.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016

Obtaining a strong classifier requires many iterations of the backpropagation algorithm using 50000 training images. Therefore, it is a good idea to utilize vectorized code in your implementation. This can be done by computing the gradients for subsets of the training data using minibatches. You can have a minibatch as a matrix and compute the forward propagation and gradients using matrix operations. By averaging the gradients obtained from the minibatch, the backpropagation can be carried out in the same way as you would do when training with one sample at a time. Due to the averaging, the obtained gradients are less affected by noise and it is typically possible to have higher learning rates.

A part of optimizing the neural network is by changing its architecture. Therefore, it is recommended that you implement your network such that you can change the number of layers and the number of neurons in each layer.

1. Implement a fully connected neural network for image classification. The data should be normalized and centered, i.e. vectors of unit length using the 2-norm and with zero mean. Besides vectorizing the code it is also worth considering the data type. Single is faster than double, so you should consider if you want faster computations, at the cost of lower precision. You can experimentally evaluate if the high precision is necessary.

2. Train the network and plot the training and validation error for each iteration (epoch). The dataset could be split into 50000 images for training and 10000 for validation.

9.2 *Optimizing the neural network*

A large number of techniques for optimizing the performance of the neural network has been proposed. Neural networks are typically initialized with random numbers, but the performance depends on the choice of the initialization strategy. You can therefore optimize the network by experimenting with different initialization strategies.

Optimization with stochastic gradient decent can be slow, but updating the gradients using momentum can accelerate the learning rate. Momentum is obtained by computing the gradients as a weighted combination of the previous gradient and the new gradient. Hereby, the gradient is computed as a moving average with exponential decay. Another way of ensuring convergence of the gradient decent is by adapting the learning rate. Here you can adapt the learning rate to the individual gradient estimates.

1. Implement one or more optimization strategies and document how it affects the obtained result.

9.3 *Regularization methods*

It is important that the neural network generalizes well such that it can classify new unseen data. Since neural networks often have many parameters it is easy to overfit the model, especially on small datasets where a very low training error can be obtained, but the validation error will be high. One way to overcome the problem with training a neural network on small datasets is through dataset augmentation, where fake data is fabricated by small modifications of the input data. This can be done by small permutations or by adding small amounts of noise. Hereby a much larger dataset can be obtained, which can help the training.

Instead of adding noise to augment the training data, small amounts of noise may be added to the hidden units in the network. You can add random noise in each minibatch iteration. Noise can also be added to the output targets for obtaining better performance.

Dropout is another method for regularizing the neural network. Here a random selection of neurons are set to zero during each minibatch iteration leaving out these neurons in that iteration. Setting the neurons to zero resembles having a number of different neural networks and is inspired by ensemble methods.

1. Try experimenting with regularization methods. You can also get inspired by architectures that other people have had success with.