Introd. to Neural Networks

Dr. S. Recksiegel WS 2022/23

Exercise Sheet 7

L. Rickler, E. Eberhard, J. Mayer-Steudte, L. Wicke

November 30, 2022 Moodle: https://www.moodle.tum.de/course/view.php?id=79979

On this exercise sheet, we will play with Nielsen's `network.py`. You should have run the MNIST example (as shown in the lecture) before working on these exercises. `network.py` is available from the moodle (the original version from Nielsen's website needs to be modified to work with Python3). If you want to run the code on the CIP computers but see the jupyter notebook on your own device, login to one of the CIP computers (but *not* the host ssh.cip.ph.tum.de), type `jupyter-notebook-remote` and follow the instructions.

1. **Our *very simple first Neural Network* in Python**

   To get acquainted with the way that "real" Neural Networks work, let us implement last week's example with the help of Michael Nielsen's `network.py`.

   We counted only the four "real" neurons, but for the x and y inputs, we need two input neurons and we also need an output neuron, so in Nielsen's convention, we need a `net=network.Network([2,4,1])`. (Remember to `import network` first.) The network will be initialised with random weights and biases, we can visualise the output e.g. with

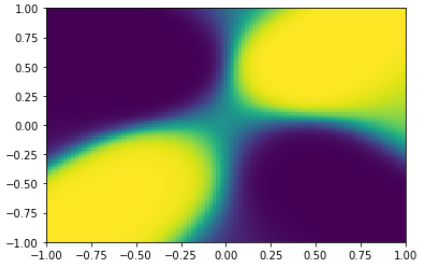
   ```
   import numpy as np
   import matplotlib.pyplot as plt
   nbins=100
   xi, yi = np.mgrid[-1:1:nbins*1j, -1:1:nbins*1j]
   zi = net.feedforward(np.vstack([xi.flatten(), yi.flatten()]))
   plt.pcolormesh(xi, yi, zi.reshape(xi.shape), shading='auto')
   plt.show()
   ```

   If you create the net several times, the output should always look slightly different.

   To create training data, we have to take into account that Nielsen's code expects a `list` of `tuples` of `numpy.ndarrays`, also the output neuron will have a value between 0 and 1, so for each training point you should do something like

   ```
   training.append((np.array([[x],[y]]),np.array([[1/2+np.sign(x*y)/2]])));
   ```

   After training your network on something like 100 training points (due to the small number, you will probably need more epochs than you might think), the output of your network should now look something like this.

2. **The effect of affine transformations on simple Neural Networks**

   Let us study how moving, resizing and rotating objects can affect the performance of NN. We will be using the affNIST dataset, see `https://www.cs.toronto.edu/~tijmen/affNIST/`. You can access the data locally at `/WWW/users/nn/affNIST/` or at `https://users.ph.tum.de/nn/affNIST/`.

   First, confirm that Nielsen's code gives results identical to before when supplying the original data from the affNIST sources. Since affNIST structures the data in a significantly different way, we provide the code to convert the data:

```
import scipy.io as sio
data = sio.loadmat('/WWW/users/nn/affNIST/training_and_validation-orig.mat')["affNISTdata"]
data = data[0,0]
images = data[2].swapaxes(0,1)
digitsvect = data[4].swapaxes(0,1)
digitsint = data[5][0]
IMG_SIZE = len(images[0])
training_data=[]
for i in range(50000):
    training_data.append(((images[i]/255).reshape(-1,1), digitsvect[i].reshape(-1,1)))
test_data=[]
for i in range(50000,60000):
    test_data.append(((images[i]/255).reshape(-1,1), digitsint[i]))
```

- Run Nielsen's Neural Network on the original data as converted with the code snippet above and confirm that it behaves like before
- Run the NN with the modified data provided by affNIST, the data file `training_and_validation.mat` contains the original 28x28 images centred in a 40x40 bitmap, `training_and_validation_batches/1.mat` etc. are batches where affine transformations have been applied to the data. For our purposes it suffices to use just one batch. Be careful with the set-up of the network, the images now have a different size!

  How does the performance of our NN on these data sets compare to the original performance?

3. ***Advanced:*** **If you want to, you can already play a bit with Keras**
   You can of course install tensorflow on your own machine, but the easiest option is to login to one of the CIP machines, enter the virtual environment that we prepared for you with `source /mount/packs/tensorflow/bin/activate`, then start the remote notebook with `jupyter-notebook-remote` and play with my Keras notebooks (or look for Keras examples anywhere online).

Here are some examples of transformations in the affNIST data: