Introd. to Neural Networks

Dr. S. Recksiegel WS 2022/23

Exercise Sheet 8

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

December 7, 2022 Moodle: https://www.moodle.tum.de/course/view.php?id=79979

This is mainly not really a new exercise sheet but rather an explanation/extension of the last one. We will continue to play with Nielsen's `network.py` and start working with Keras. You will find the necessary files in the moodle (the affNIST data is available locally at `/WWW/users/nn/affNIST/` or at `https://users.ph.tum.de/nn/affNIST/`).

1. **A closer look at Nielsen's MNIST program**
   You can set up and train the network with

   ```
   import mnist_loader
   training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
   import network
   net = network.Network([784, 30, 10])
   net.SGD(training_data, 10, 10, 3.0, test_data=test_data)
   ```

   This will load the MNIST data in the form expected by `network.py`, set up a network with 784 (28 pixels by 28 pixels) input neurons, 30 neurons in the hidden layer and 10 neurons in the output layer.

   How many weights and biases does this network have? You will be able to verify your answer later when we reproduce the same network in Keras.

   The last line trains the network on mini batches of size 10 in 10 epochs with a training parameter of 3.0. (As discussed in the lecture, Nielsen's `test_data` should actually be called `validation_data`.) Play with these parameters and observe the performance of the network.

   You can extract an individual data sample, look at it and check the answer of the network with

   ```
   xtest,ytest=test_data[0]
   import matplotlib.pyplot as plt
   plt.imshow(xtest.reshape(28,28),cmap='binary')
   plt.show()
   net.feedforward(xtest)
   ```

   If you create your own 28x28 greyscale image of a digit with Gimp and save it as raw data (the file size should be 784 bytes), you can ask your network to recognize the digit with (the `/255` normalises the data to $[0, 1]$ and the `1-` makes white/empty pixels give an input value of 0 to the input neurons)

   ```
   import numpy as np
   test=1-np.fromfile('test-image.data',dtype=np.ubyte)/255
   net.feedforward(np.reshape(test,(784, 1)))
   ```

   Not too bad for a 60-line program.

2. Keras: Fully connected and convolutional layers

In the Keras example available in the moodle, `keras-mnist-example-sr.ipynb`, a convolutional network is used to learn the MNIST digits, this network manages to recognize more than 99% of the digits correctly, it uses about 35,000 parameters.

To learn more about Keras, we want to implement Nielsen's simple neural network in Keras. The network of 784 input pixels, 30 hidden neurons in a fully connected layer and 10 output neurons

```
model2 = keras.Sequential(
    [   layers.Input(shape=(784)),
        layers.Dense(30, activation="sigmoid"),
        layers.Dense(10, activation="sigmoid"),
    ])
model2.summary()
model2.compile(loss="mean_squared_error", optimizer="adam", metrics=["accuracy"])
model2.fit(x_train.reshape(-1,784), y_train, batch_size=10, epochs=10,
           validation_split=0.1)
```

As you can see, although the number of parameters is comparable to the network with the convolutional layers, the recognition rate is not as good (it now behaves like Nielsen's 60-line `network.py`). The strength of convolutional neural networks becomes even more apparent when you let both types of networks run on the affNIST dataset as explained on the last exercise sheet.

You can also use "SGD" instead of "adam" (after re-initialising the network) to see how much the tricks that "adam" incorporates into the gradient descent accelerate the convergence.

3. Advanced exercises

You can download the module `loadData.py` from the moodle and use it to directly download the image data into your Python script.

Play a bit more with the different types of networks (Nielsen, fully connected layers in Keras, convolutional layers in Keras) to get a feeling for how effectively they can recognize handwritten digits. Change the activation function (`sigmoid`, `ReLU`,...), use different cost functions (`mean_squared_error`, `categorical_crossentropy`,...), different optimisers (`adam`, `SGD`,...).