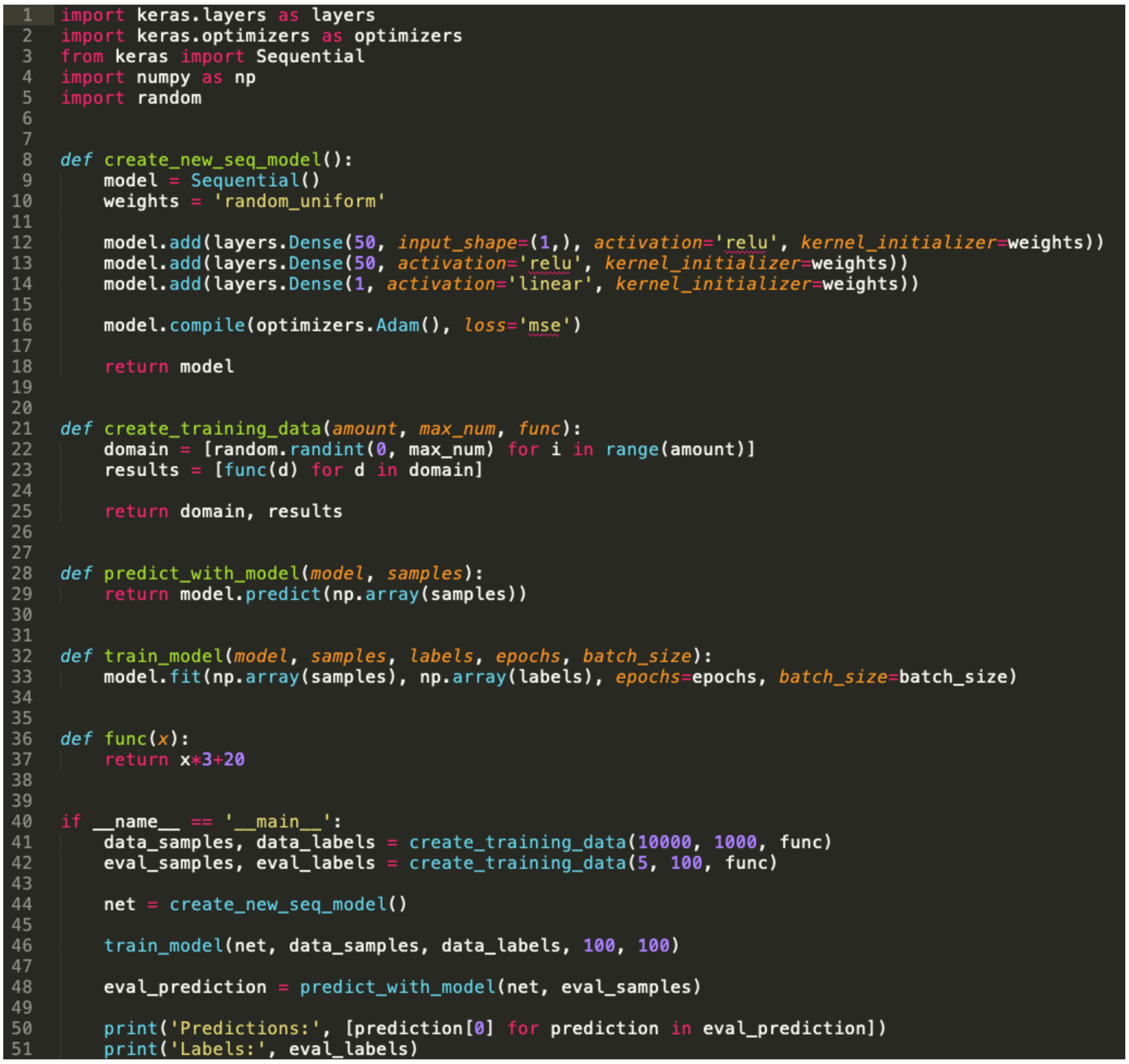
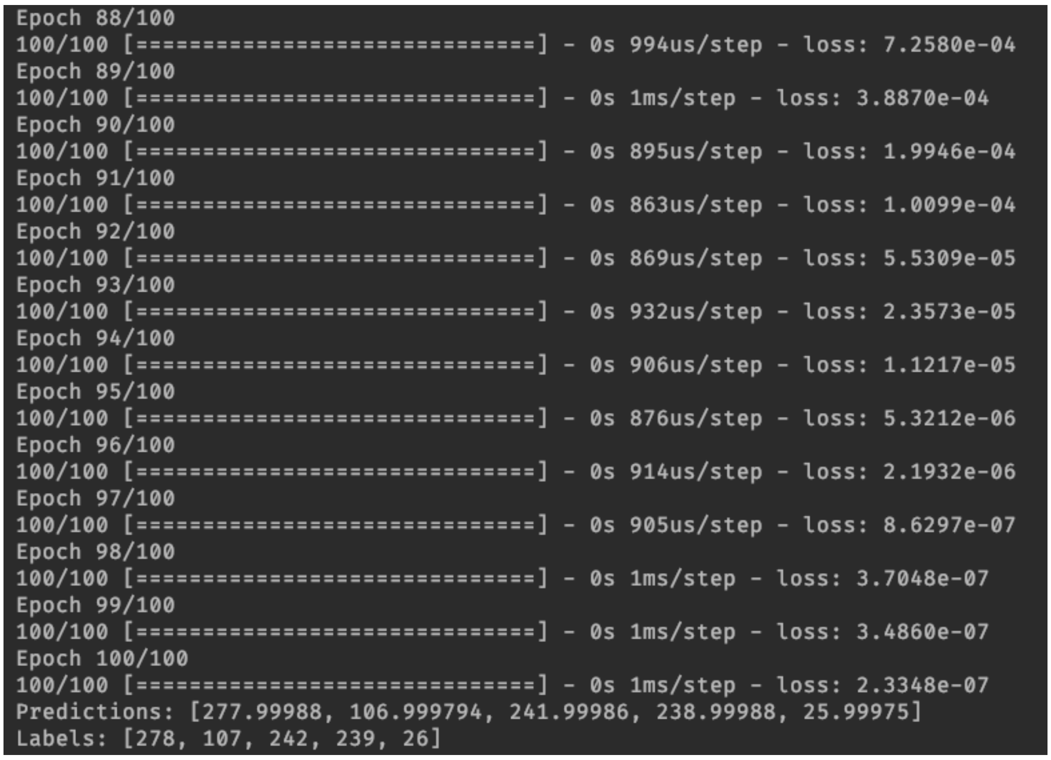
Isak Nyberg Rahul Sharma Kothuri

Part one:

Model a function: f(x) = 3x+20



```python
import keras.layers as layers
import keras.optimizers as optimizers
from keras import Sequential
import numpy as np
import random


def create_new_seq_model():
    model = Sequential()
    weights = 'random_uniform'

    model.add(layers.Dense(50, input_shape=(1,), activation='relu', kernel_initializer=weights))
    model.add(layers.Dense(50, activation='relu', kernel_initializer=weights))
    model.add(layers.Dense(1, activation='linear', kernel_initializer=weights))

    model.compile(optimizers.Adam(), loss='mse')

    return model


def create_training_data(amount, max_num, func):
    domain = [random.randint(0, max_num) for i in range(amount)]
    results = [func(d) for d in domain]

    return domain, results


def predict_with_model(model, samples):
    return model.predict(np.array(samples))


def train_model(model, samples, labels, epochs, batch_size):
    model.fit(np.array(samples), np.array(labels), epochs=epochs, batch_size=batch_size)


def func(x):
    return x*3+20


if __name__ == '__main__':
    data_samples, data_labels = create_training_data(10000, 1000, func)
    eval_samples, eval_labels = create_training_data(5, 100, func)

    net = create_new_seq_model()

    train_model(net, data_samples, data_labels, 100, 100)

    eval_prediction = predict_with_model(net, eval_samples)

    print('Predictions:', [prediction[0] for prediction in eval_prediction])
    print('Labels:', eval_labels)
```



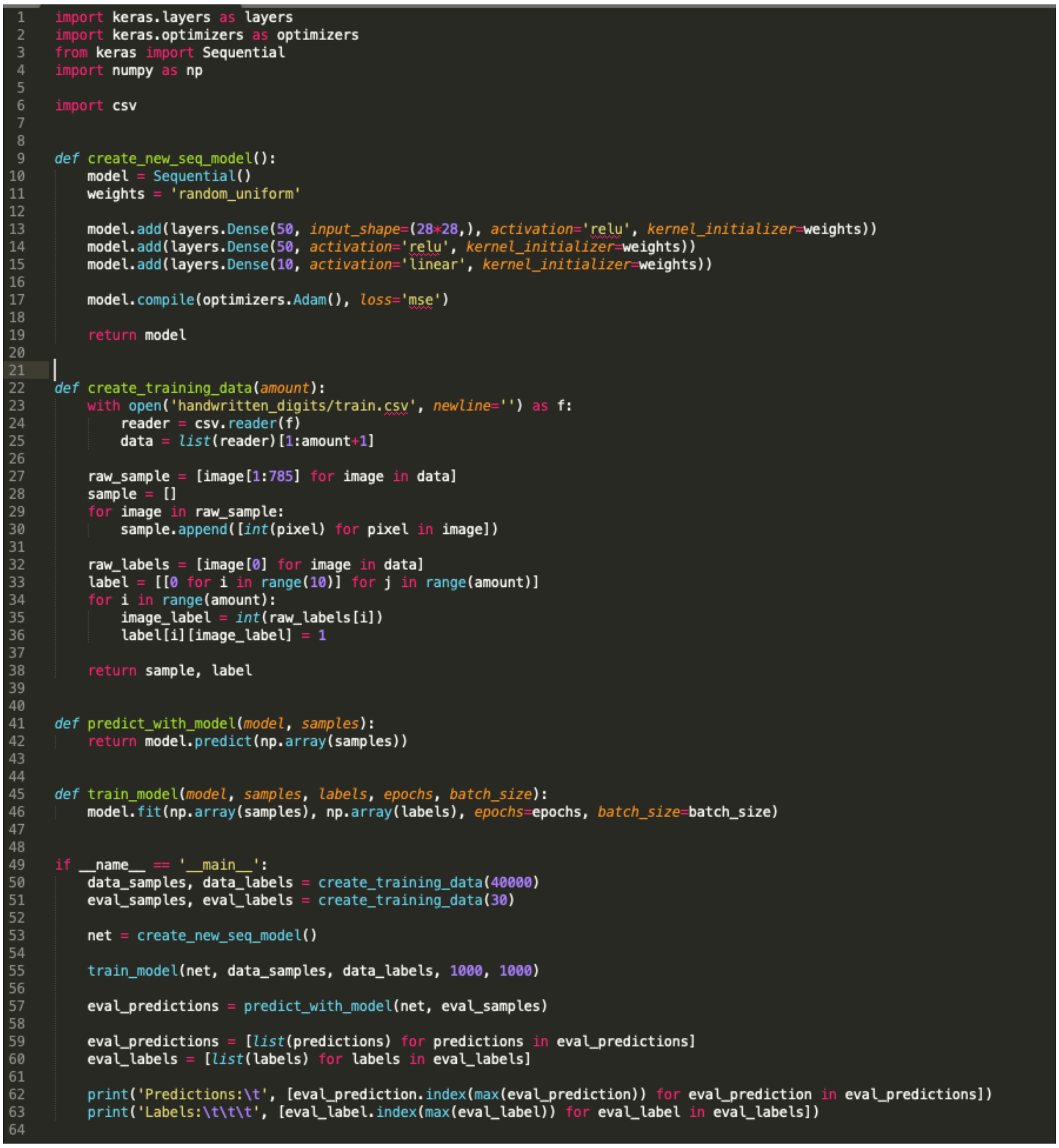
```
Epoch 88/100
100/100 [==============================] - 0s 994us/step - loss: 7.2580e-04
Epoch 89/100
100/100 [==============================] - 0s 1ms/step - loss: 3.8870e-04
Epoch 90/100
100/100 [==============================] - 0s 895us/step - loss: 1.9946e-04
Epoch 91/100
100/100 [==============================] - 0s 863us/step - loss: 1.0099e-04
Epoch 92/100
100/100 [==============================] - 0s 869us/step - loss: 5.5309e-05
Epoch 93/100
100/100 [==============================] - 0s 932us/step - loss: 2.3573e-05
Epoch 94/100
100/100 [==============================] - 0s 906us/step - loss: 1.1217e-05
Epoch 95/100
100/100 [==============================] - 0s 876us/step - loss: 5.3212e-06
Epoch 96/100
100/100 [==============================] - 0s 914us/step - loss: 2.1932e-06
Epoch 97/100
100/100 [==============================] - 0s 905us/step - loss: 8.6297e-07
Epoch 98/100
100/100 [==============================] - 0s 1ms/step - loss: 3.7048e-07
Epoch 99/100
100/100 [==============================] - 0s 1ms/step - loss: 3.4860e-07
Epoch 100/100
100/100 [==============================] - 0s 1ms/step - loss: 2.3348e-07
Predictions: [277.99988, 106.999794, 241.99986, 238.99988, 25.99975]
Labels: [278, 107, 242, 239, 26]
```

After this NN was able to approximate the function with very close precision and accuracy. The number of nodes in each layer were the following: 1 – 50 – 50 – 1

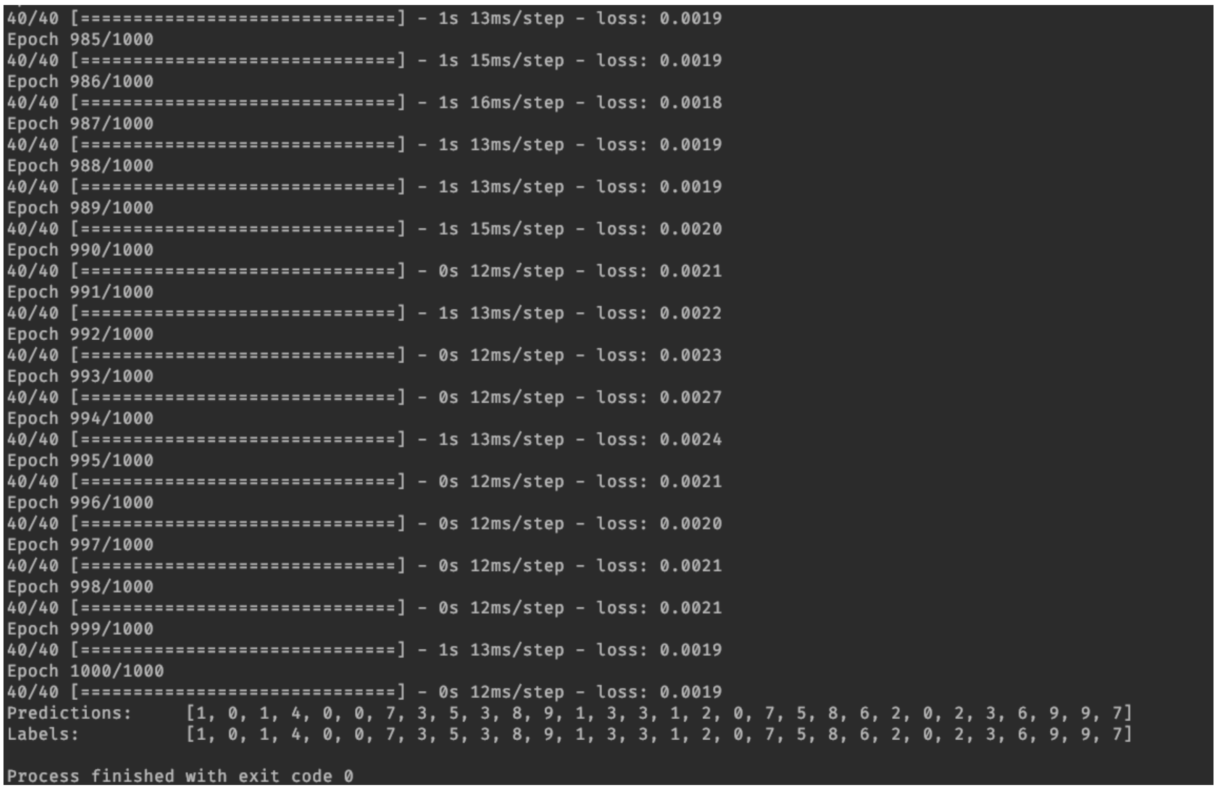
Part 2 reading handwritten digits.

We downloaded data from the following link of handwritten numbers in 28x28 resolution in black and white outputted as an array of pixel values. From this we created a neural network with 748 input nodes, for the brightness of every pixel. We had like the previous example two hidden layers of 50 nodes and then an output layer with 10 nodes, one for each digit (0-9). The node with the highest value was the node that was interoperated and the neutral network's "choice."

Isak Nyberg Rahul Sharma Kothuri



```python
import keras.layers as layers
import keras.optimizers as optimizers
from keras import Sequential
import numpy as np

import csv


def create_new_seq_model():
    model = Sequential()
    weights = 'random_uniform'

    model.add(layers.Dense(50, input_shape=(28*28,), activation='relu', kernel_initializer=weights))
    model.add(layers.Dense(50, activation='relu', kernel_initializer=weights))
    model.add(layers.Dense(10, activation='linear', kernel_initializer=weights))

    model.compile(optimizers.Adam(), loss='mse')

    return model


def create_training_data(amount):
    with open('handwritten_digits/train.csv', newline='') as f:
        reader = csv.reader(f)
        data = list(reader)[1:amount+1]

    raw_sample = [image[1:785] for image in data]
    sample = []
    for image in raw_sample:
        sample.append([int(pixel) for pixel in image])

    raw_labels = [image[0] for image in data]
    label = [[0 for i in range(10)] for j in range(amount)]
    for i in range(amount):
        image_label = int(raw_labels[i])
        label[i][image_label] = 1

    return sample, label


def predict_with_model(model, samples):
    return model.predict(np.array(samples))


def train_model(model, samples, labels, epochs, batch_size):
    model.fit(np.array(samples), np.array(labels), epochs=epochs, batch_size=batch_size)


if __name__ == '__main__':
    data_samples, data_labels = create_training_data(40000)
    eval_samples, eval_labels = create_training_data(30)

    net = create_new_seq_model()

    train_model(net, data_samples, data_labels, 1000, 1000)

    eval_predictions = predict_with_model(net, eval_samples)

    eval_predictions = [list(predictions) for predictions in eval_predictions]
    eval_labels = [list(labels) for labels in eval_labels]

    print('Predictions:\t', [eval_prediction.index(max(eval_prediction)) for eval_prediction in eval_predictions])
    print('Labels:\t\t\t', [eval_label.index(max(eval_label)) for eval_label in eval_labels])
```



```
40/40 [==============================] - 1s 13ms/step - loss: 0.0019
Epoch 985/1000
40/40 [==============================] - 1s 15ms/step - loss: 0.0019
Epoch 986/1000
40/40 [==============================] - 1s 16ms/step - loss: 0.0018
Epoch 987/1000
40/40 [==============================] - 1s 13ms/step - loss: 0.0019
Epoch 988/1000
40/40 [==============================] - 1s 13ms/step - loss: 0.0019
Epoch 989/1000
40/40 [==============================] - 1s 15ms/step - loss: 0.0020
Epoch 990/1000
40/40 [==============================] - 0s 12ms/step - loss: 0.0021
Epoch 991/1000
40/40 [==============================] - 1s 13ms/step - loss: 0.0022
Epoch 992/1000
40/40 [==============================] - 0s 12ms/step - loss: 0.0023
Epoch 993/1000
40/40 [==============================] - 0s 12ms/step - loss: 0.0027
Epoch 994/1000
40/40 [==============================] - 1s 13ms/step - loss: 0.0024
Epoch 995/1000
40/40 [==============================] - 0s 12ms/step - loss: 0.0021
Epoch 996/1000
40/40 [==============================] - 0s 12ms/step - loss: 0.0020
Epoch 997/1000
40/40 [==============================] - 0s 12ms/step - loss: 0.0021
Epoch 998/1000
40/40 [==============================] - 0s 12ms/step - loss: 0.0021
Epoch 999/1000
40/40 [==============================] - 1s 13ms/step - loss: 0.0019
Epoch 1000/1000
40/40 [==============================] - 0s 12ms/step - loss: 0.0019
Predictions:     [1, 0, 1, 4, 0, 0, 7, 3, 5, 3, 8, 9, 1, 3, 3, 1, 2, 0, 7, 5, 8, 6, 2, 0, 2, 3, 6, 9, 9, 7]
Labels:          [1, 0, 1, 4, 0, 0, 7, 3, 5, 3, 8, 9, 1, 3, 3, 1, 2, 0, 7, 5, 8, 6, 2, 0, 2, 3, 6, 9, 9, 7]

Process finished with exit code 0
```

We let the neural network train with 1000 epochs with a batch size of 40, this means that the training did not actually go through the entire dataset as the dataset included about 46000 samples. Nevertheless, as visible in the predictions the neural network was able to correctly label at least 30 out of the sample images. Naturally the training time of the neural network was a lot longer than the first task. In the future it is better practice to separate the evaluation samples from the training samples but in this case since the data that was trained on was not reused to a greater extent the concern for over or under fitting the data is not as relevant.