HW3 Neural Networks

Jens Ifver

October 2021

1 Part 1: Classification challenge

I used a model with two convolutional layers with ReLU as activation function, each followed by a max-pooling layer. The last two layers where fully connected with 100- and 10 neurons, ReLU- and softmax activation, respectively.

1.1 Code

```
from keras.datasets import mnist
import tensorflow as tf
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.callbacks import ModelCheckpoint
import keras
import numpy as np

# Model construction
model = tf.keras.models.Sequential()
model.add(keras.Input(shape=(28, 28, 1)))

model.add(Conv2D(16, 3, activation = 'relu'))
model.add(MaxPooling2D(2))
model.add(Conv2D(32, 3, activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(100, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
print(model.summary())

# Load and transform data
(train_X, train_y), (test_X, test_y) = mnist.load_data()
train_X = train_X.reshape(60000,28,28,1)
train_X = train_X / 255
test_X = test_X.reshape(test_X.shape[0],28,28,1)
test_X = test_X / 255

# Run training
np.random.seed(123)
checkpoint1 = ModelCheckpoint('best_model1.keras', monitor='val_loss', verbose
                              =1, save_best_only=True)
model.compile('adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x=train_X, y=train_y, epochs = 10, validation_data=(test_X,test_y),
           callbacks = [checkpoint1])

# Load and transform test data
f = open('/Users/Jensaeh/skola/Neural networks/HW3/part1/xTest2.bin', 'rb')
canvas_test = f.read()
f.close()
canvas_test = np.frombuffer(canvas_test, dtype='uint8')
canvas_test = canvas_test.reshape(-1, 28, 28, 1)
canvas_test = canvas_test.transpose((0, 2, 1, 3))
canvas_test = canvas_test / 255

# Check data conversion
import matplotlib.pyplot as plt
plt.imshow(canvas_test[0].reshape(28,28), cmap='gray')
```

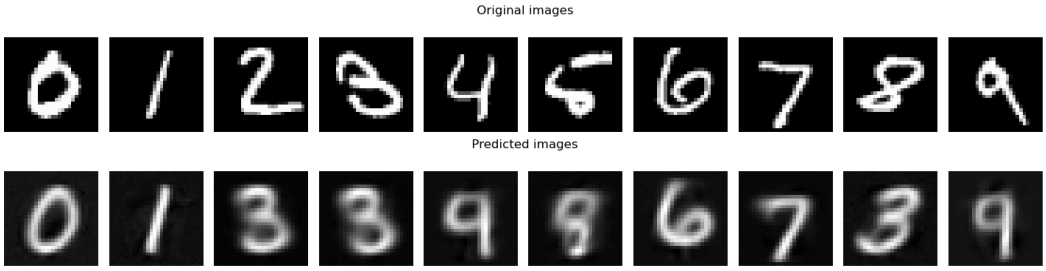
```
# Prediction on test set and canvas test set
best_model = keras.models.load_model('best_model1.keras')
y_pred_canvas = best_model.predict(canvas_test)
y_pred_canvas = np.argmax(y_pred_canvas, axis = 1)
y_pred = best_model.predict(test_X)
y_pred = np.argmax(y_pred, axis = 1)
from sklearn.metrics import accuracy_score
accuracy_score(test_y, y_pred)

# Save result as CSV
import pandas as pd
pd.DataFrame(y_pred_canvas).to_csv('classifications.csv', index = False, header
                                   = False)
```

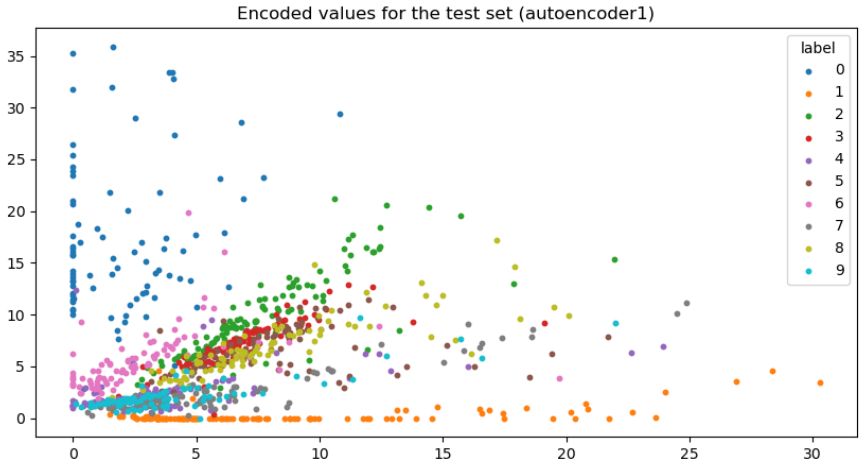
2 Part 2: Fully Connected Autoencoder

2.1 Autoencoder 1

Autoencoder 1 uses only two neurons in the bottleneck layer and is hence not particularly good at reconstructing images from the test set. In Figure 1a we see the a reconstruction of the first observation of each digit in the test set. We see that digits 0 and 1 are very well reproduced which can be explained by inspecting the scatterplot in Figure 1b. Here observe that zeros and ones are clearly separated with the rule of $V_1 \approx 0$ for zeros and $V_2 \approx 0$ for ones. One could argue that the decision boundary is are $V_1 = V_2$ since the rest of the digits are close to this line. Digits 6, 7 and 9 are also quite well reproduced in Figure 1a, but from observing the scatterplot it is hard to pick values that predicts the correct digit, especially for 7 and 9.



(a)

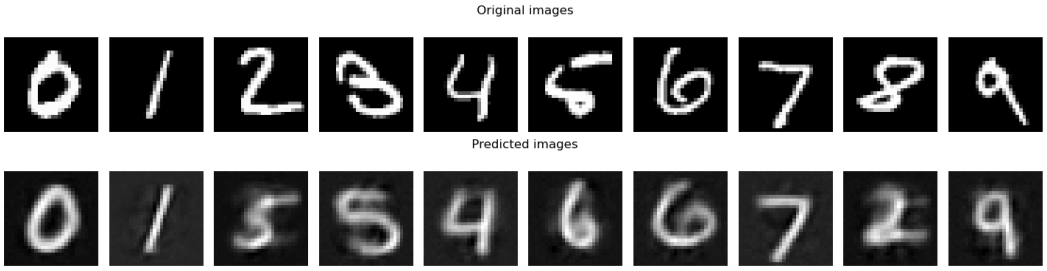


(b)

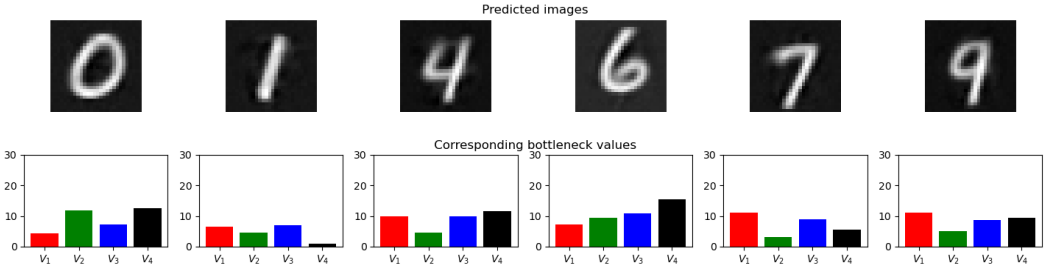
Figure 1: (a) Reconstruction from test set using autoencoder 1. Top row are original images and bottom row are reconstructed images. (b) A scatterplot over the encoded values for the test set using autoencoder 1. The x-axis and y-axis represents the values of the bottleneck neurons V_1 and V_2 respectively.

2.2 Autoencoder 2

Autoencoder 2 uses four bottleneck neurons and performs better than autoencoder 1 at the task of reconstructing the digits. We see in Figure 2a that autoencoder 2 in addition to zeros, ones, sixes, sevens and nines, also reconstruct digit 4 well. To find the rule used by the encoder I calculated the mean of each bottleneck neuron for each digit that was well classified to how they differ. These mean values are represented by the bars in the bar-plots in fig:Ng4, which are positioned under the corresponding prediction performed using these mean values. By looking at the bar-plots we see that each digit that are well-classified, are positioned in a separate part of the four dimensional space spanned by V_1, V_2, V_3, V_4. For example, zeros seem to go under the rule $V_1 < V_3 < V_2 < V_4$, sevens are possibly restricted to the space where $V_2 < V_4 < V_3 < V_1$ and so on. The digits that are not well-classified are probably assigned sub-spaces where multiple observations are positioned when using these particular settings. However, the network should be able to separate the different digits well with the right settings and training, given that the bottleneck layer creates a four dimensional space.



(a)



(b)

Figure 2: (a) Reconstruction from test set using autoencoder 2. Top row are original images and bottom row are reconstructed images. (b) Top row showing a decoding prediction by inspection of the most well reconstructed images of the test set using autoencoder 2. Bottom row showing the corresponding bar-plot that illustrates the values of the bottleneck neurons used for the prediction.

2.3 Code

```
import keras
from keras.layers import Dense
import numpy as np
from keras.datasets import mnist

#### Load and normalize data ####
(trainX, trainY), (testX, testY) = mnist.load_data()
trainX = trainX.reshape(trainX.shape[0], 784).astype('float32') / 255.
testX = testX.reshape(testX.shape[0], 784).astype('float32') / 255.

#### Autoencoder 1 ####
seq_input = keras.Input(shape=(784,))

# Encoder
layers_encode = Dense(50, kernel_initializer='glorot_uniform', activation =
                      'relu')(seq_input)
layers_encode = Dense(2, kernel_initializer='glorot_uniform', activation =
                      'relu')(layers_encode)
# Decoder
layers_decode = Dense(784, kernel_initializer='glorot_uniform',
                      activation = 'relu')(layers_encode)
layers_decode = Dense(784)(layers_decode)

# Autoencoder
autoencoder = keras.Model(seq_input, layers_decode)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Settings for training
eta = 0.001
mini_batch_size = 8192
epochs = 800 # 800

# Train autoencoder
autoencoder.fit(trainX, trainX, epochs = epochs, batch_size = mini_batch_size,
                shuffle = True, validation_data=(testX, testX))
# Create encoder
encoder_layer = autoencoder.layers[1](seq_input)
encoder_layer = autoencoder.layers[2](encoder_layer)
encoder = keras.Model(seq_input, encoder_layer)
# Create decoder
encoded_input = keras.Input(shape=(2,))
decoder_layer = autoencoder.layers[-2](encoded_input)
decoder_layer = autoencoder.layers[-1](decoder_layer)
decoder = keras.Model(encoded_input, decoder_layer)

# Save model
autoencoder.save('/Users/Jensaeh/skola/Neural networks/HW3/part2/mse')

# Load model
model = keras.models.load_model('/Users/Jensaeh/skola/Neural'+
                                ' networks/HW3/part2/mse')
encoder_layer_load = model.layers[1](seq_input)
encoder_layer_load = model.layers[2](encoder_layer_load)
encoder = keras.Model(seq_input, encoder_layer_load)
```

```
decoder_layer_load = model.layers[3](encoded_input)
decoder_layer_load = model.layers[4](decoder_layer_load)
decoder = keras.Model(encoded_input, decoder_layer_load)

# Make predictions
encoded_data = encoder.predict(testX)
decoded_data = decoder.predict(encoded_data)

#### Plot results autoencoder 1 ####
import matplotlib.pyplot as plt

# Figure 1 (autoencoder1, results)
rows = 2
cols = 10
names = ['\n Original images', '\n Predicted images']
fig, big_axes = plt.subplots(figsize=(20,4), nrows = 2, ncols = 1, sharey=True)
for row, big_ax in enumerate(big_axes, start =1):
    big_ax.set_title(names[row-1], fontsize=12)
    big_ax.axis('off')
    big_ax._frameon = False

for it in range(1,11):
    ind = np.argmax(testY == it-1)
    fig.add_subplot(rows, cols, it)
    plt.imshow(testX[ind].reshape(28,28), cmap = 'gray')
    plt.axis('off')
    fig.add_subplot(rows, cols, it+10)
    plt.imshow(decoded_data[ind].reshape(28,28), cmap = 'gray')
    plt.axis('off')
plt.tight_layout()
plt.show()
fig.set_facecolor('w')
plt.savefig('pred1.png')

# Figure 2 (scatterplot, bottleneck layer)
# 0 1 2 6 7 9 good, 4 5 8 bad
good =[0, 1, 6, 7, 9]
bad = [2, 3, 4, 5, 8]
n = 1000
small_testX = testX[:n]
small_testY = testY[:n]
small_encoded = encoded_data[:n]
fig, ax = plt.subplots(figsize=(10,5))
for it in good:
    ax.scatter(small_encoded[(small_testY == it),0], small_encoded[(small_testY
                                                                    == it),1],
               label=it, s = 10)
ax.legend(title = 'label')
plt.savefig('scatter1.png')

# Observed values for encoder
zeros = [0,15]
ones = [7,0]
sixes = [2,4]
sevens = [6,2]
nines = [2,2]
```

```
good_observed = np.array([zeros, ones, sixes, sevens, nines])

# Figure 3 (Feed encoder observed values)
decoded_observed = decoder.predict(good_observed)
good_labels = ['0 (0,15)', '1 (7,0)', '6 (2,4)', '7 (6,2)',
               '9 (2,2)']
fig = plt.figure(figsize=(16,4))
for it in range(5):
    fig.add_subplot(2,4, it+1)
    plt.imshow(decoded_observed[it].reshape(28,28), cmap = 'gray')
    plt.title(good_labels[it])
    plt.axis('off')
plt.show()
plt.savefig('observed1.png')

# Figure 4 (final scatterplot autoencoder 1)
n = 1000
small_testX = testX[:n]
small_testY = testY[:n]
small_encoded = encoded_data[:n]
fig, ax = plt.subplots(figsize=(10,5))
for it in range(10):
    ax.scatter(small_encoded[(small_testY == it),0],
               small_encoded[(small_testY== it),1], label=it, s = 10)
ax.legend(title = 'label')
ax.set_title('Encoded values for the test set (autoencoder1)')
plt.show()
plt.savefig('scatter_final1.png')

#### Autoencoder 2 ####
seq_input = keras.Input(shape=(784,))

# Encoder
layers_encode2 = Dense(50, kernel_initializer='glorot_uniform', activation =
                       'relu')(seq_input)
layers_encode2 = Dense(4, kernel_initializer='glorot_uniform', activation =
                       'relu')(layers_encode2)
# Decoder
layers_decode2 = Dense(784, kernel_initializer='glorot_uniform',
                       activation = 'relu')(layers_encode2)
layers_decode2 = Dense(784)(layers_decode2)

# Autoencoder
autoencoder2 = keras.Model(seq_input, layers_decode2)
autoencoder2.compile(optimizer='adam', loss='mean_squared_error')

# Settings for training
eta = 0.001
mini_batch_size = 8192
epochs = 800 # 800

# Train autoencoder
autoencoder2.fit(trainX, trainX, epochs = epochs, batch_size = mini_batch_size,
                 shuffle = True, validation_data=(testX, testX))
# Create encoder
encoder_layer2 = autoencoder2.layers[1](seq_input)
```

```
encoder_layer2 = autoencoder2.layers[2](encoder_layer2)
encoder2 = keras.Model(seq_input, encoder_layer2)
# Create decoder
encoded_input2 = keras.Input(shape=(4,))
decoder_layer2 = autoencoder2.layers[-2](encoded_input2)
decoder_layer2 = autoencoder2.layers[-1](decoder_layer2)
decoder2 = keras.Model(encoded_input2, decoder_layer2)

# Save model
autoencoder2.save('/Users/Jensaeh/skola/Neural networks/HW3/part2/mse2')

# Load model
model2 = keras.models.load_model('/Users/Jensaeh/skola/Neural'+
                                 ' networks/HW3/part2/mse2')
encoder_layer_load2 = model2.layers[1](seq_input)
encoder_layer_load2 = model2.layers[2](encoder_layer_load2)
encoder2 = keras.Model(seq_input, encoder_layer_load2)

decoder_layer_load2 = model2.layers[3](encoded_input2)
decoder_layer_load2 = model2.layers[4](decoder_layer_load2)
decoder2 = keras.Model(encoded_input2, decoder_layer_load2)

# Make predictions
encoded_data2 = encoder2.predict(testX)
decoded_data2 = decoder2.predict(encoded_data2)

### Plot results autoencoder 2 ###

# Figure 5 (autoencoder2, results)
rows = 2
cols = 10
names = ['\n Original images', '\n Predicted images']
fig, big_axes = plt.subplots(figsize=(20,4), nrows = 2, ncols = 1, sharey=True)
for row, big_ax in enumerate(big_axes, start =1):
    big_ax.set_title(names[row-1], fontsize=12)
    big_ax.axis('off')
    big_ax._frameon = False

for it in range(1,11):
    ind = np.argmax(testY == it-1)
    fig.add_subplot(rows, cols, it)
    plt.imshow(testX[ind].reshape(28,28), cmap = 'gray')
    plt.axis('off')
    fig.add_subplot(rows, cols, it+10)
    plt.imshow(decoded_data2[ind].reshape(28,28), cmap = 'gray')
    plt.axis('off')
plt.tight_layout()
plt.show()
fig.set_facecolor('w')
plt.savefig('pred2.png')

# Inspecting coding rule by taking the mean for each digit for each neuron in
# bottlenck layer
import pandas as pd
bottleneck_data = pd.DataFrame()
for it in range(10):
```

```
    bottleneck_data[str(it)] = encoded_data2[testY == it,].mean(axis = 0)
bottleneck_data.head()
bottleneck_data = bottleneck_data.to_numpy()
bottleneck_data = bottleneck_data.transpose()

predicted_inspection = decoder2.predict(bottleneck_data)
good_pred = [0, 1, 4, 6, 7, 9]
titles = ['\n Predicted images', '\n Corresponding bottleneck values']
rows = 2
cols = np.shape(good_pred)[0]
fig, big_axes = plt.subplots(figsize=(20,4), nrows = 2, ncols = 1, sharey=True)
for row, big_ax in enumerate(big_axes, start =1):
    big_ax.set_title(titles[row-1], fontsize=12)
    big_ax.axis('off')
    big_ax._frameon = False

for it in range(1,cols+1):
    ind = good_pred[it-1]
    fig.add_subplot(rows, cols, it)
    plt.imshow(predicted_inspection[ind].reshape(28,28), cmap = 'gray')
    plt.axis('off')
    fig.add_subplot(rows, cols, it+cols)
    plt.bar(x = (0, 5, 10, 15), height = bottleneck_data[ind],
            tick_label=(r'$V_1$', r'$V_2$', r'$V_3$', r'$V_4$'), width = 4,
            color = ['red', 'green', 'blue', 'black'])
    plt.ylim([0,30])
plt.tight_layout()
plt.show()
plt.savefig('inspection_a2.png')
```

3 Part 3: Chaotic time series prediction

I construct a reservoir with the settings given in the task description. I train the reservoir and use the training states to estimate the output weights. The output weights are then used to predict the test data time series.

3.1 Code

```
import numpy as np
#import pandas as pd

# Load data
Xtrain = np.genfromtxt('/Users/Jensaeh/skola/Neural'+
                       ' networks/HW3/part3/training-set.csv', delimiter=',')
Xtest = np.genfromtxt('/Users/Jensaeh/skola/Neural'+
                       ' networks/HW3/part3/test-set-8.csv', delimiter=',')

#### Functions for- ####

# -calculating state of reservoir
def calc_rvec(rvec, w, w_in, x_t):
    tmp = np.add( np.matmul(w,rvec) , np.matmul(w_in,x_t) )
    return np.tanh(tmp)
# -training the reservoir
def train_reservoir(Xtrain, N):
    n, T_train = Xtrain.shape
    # Initialize
    w_in = np.random.normal(scale = np.sqrt(0.002), size=(N,n)) # variance = 0.002
    w = np.random.normal(scale = np.sqrt(2/N), size=(N,N)) # variance = 2/N
    R = np.zeros((N,T_train))
    for ti in range(T_train-1):
        R[:,ti+1] = calc_rvec(R[:,ti], w, w_in, Xtrain[:,ti])
    return R, w, w_in
# -estimating output weights
def ridge_regression(Xtrain, R, k):
    RR = np.matmul(R, R.transpose())
    kI = np.identity(R.shape[0]) * k
    para = np.linalg.inv(np.add(RR, kI))
    tmp = np.matmul(R.transpose(), para)
    tmp = np.matmul(Xtrain , tmp)
    return tmp
# -calculating output for a single timestep
def calc_output_vec(w_out_vec, rvec):
    return np.matmul(w_out_vec,rvec)
# -predicting time series with the reservoir
def predict(Xmat, T, W_out, W, W_in):
    n, Ttest = Xmat.shape
    N = W_out.shape[1]
    R = np.zeros((N,Ttest+T))
    O = np.zeros((n,T))
    for t1 in range(Ttest-1):
        R[:,t1+1] = calc_rvec(R[:,t1], W, W_in, Xmat[:,t1])
    O[:,0] = calc_output_vec(W_out, R[:,t1+1])
    r = R[:,t1+1]
    for t2 in range(T-1):
        r = calc_rvec(r, W, W_in, O[:,t2])
        O[:,t2+1] = calc_output_vec(W_out, r)
    return O
```

```
#### Calculations ####

# Training
N = 500 # reservoir neurons
n = 3 # input neurons
R, W, W_in = train_reservoir(Xtrain, N)
# Estimate w_out
k = 0.01 # ridge parameter
w_out = ridge_regression(Xtrain,R,k)
# Predict using Xtest
T = 500
output = predict(Xtest, T, w_out, W, W_in)
# Plot results
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(xs = output[0,:], ys = output[1,:], zs=output[2,:])
np.savetxt('prediction.csv', output[1,:], delimiter=',')
```