```
import numpy as np
from numpy import random as rnd
import csv
import matplotlib.pyplot as plt

def unpickle(file):
    "Returns an a dictionary of a single batch, if filename is correct"
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict


def calculate_mean(x_train):
    "Calculate the mean of a set of training data"
    mean = np.mean(x_train,axis=0)
    return mean


def center_data(x_train):
    #TODO make sure that this works as intended
    #x_train = x_train.astype('float32')
    mean = calculate_mean(x_train)
    centered_data = x_train - mean
    centered_data /= 255
    #centered_data = centered_data.astype(np.uint8)
    return centered_data

def sigmoid_activation_function(b):


    return np.power((1+np.exp(-b)), -1)


def shuffle(X, y):
    '''
    Shuffle two corresponding arrays
    '''
    assert len(X) == len(y)
    random = np.arange(len(X))
    np.random.shuffle(random)
    X = X[random]
    y = y[random]
    return X, y


def g(b):
    #return np.tanh(b)    inner_sum = inner_sum.reshape((len(inner_sum),1))
    #return np.tanh(b)
    return sigmoid_activation_function(b)

def dg(b):
    #return (1-np.power(np.tanh(b),2))

    o_b = sigmoid_activation_function(b)
    #o_b= (1-np.power(np.tanh(b),2))
    #return o_b
    return o_b*(1-o_b)

def create_layer(inputsize, layersize):
```

```
    neurons = np.zeros((layersize,1))
    neurons = neurons.reshape(layersize,1)
    w_ij = rnd.normal(size=(layersize, inputsize),scale=
(1/(np.sqrt(inputsize))))
    return (neurons, w_ij)

def esign(i):
    if i == 0:
        return 1
    return np.sign(i)

def calculate_output(threshold, weight_matrix, x):
    if weight_matrix.shape[1]==1:
        weight_matrix = weight_matrix.T
    inner_sum = weight_matrix@x
    #inner_sum = inner_sum.reshape((len(inner_sum),x.shape[1]))
    inner_sum = inner_sum - threshold
    ret = g(inner_sum)
    assert(ret.shape == threshold.shape)
    return ret

def calculate_output_batch(threshold, weight_matrix, V, batchsize):


    inner_sum = weight_matrix@V
    #inner_sum = inner_sum.reshape((len(inner_sum),x.shape[1]))
    inner_sum = inner_sum.reshape((threshold.shape[0],batchsize))
    inner_sum = inner_sum - threshold
    ret = g(inner_sum)
    assert(ret.shape == (threshold.shape[0],batchsize))
    return ret



def propagate_forward(V : list, O : list, w : list, batchsize : int) -> None:

    #TODO make sure that it works as intended

    assert(len(V)-1==len(O) and len(O)==len(w))

    for l in range(1,len(V)):
        V[l]= calculate_output_batch(O[l-1],w[l-1],V[l-1], batchsize)


def calculate_b(w, x, O):
    inner_sum = np.dot(w,x)
    inner_sum = inner_sum - O
    return inner_sum

def calculate_d_L(b, t, V):

    #TODO make sure it works properly
    return dg(b)(t-V)

def calculate_single_d(d_l, w, b):

    #TODO make sure that the order of multiplications is right
    dgb = dg(b)
    inner_product = (d_l.T@w).T
    #ret = np.multiply(inner_product.T, dgb)
```

```
120     ret = np.multiply(inner_product,dgb)
121     #ret = d_l*w.T*dgb
122     #print(ret)
123     return ret
124 
125 
126 def calculate_all_d(V, O, w, t):
127 
128     #TODO make sure that it works as intended
129     L = len(O)-1
130     b_L = calculate_b(w[L],V[L], O[L])
131     b_L = dg(b_L)
132     temp = (t-V[-1])
133     delta_start = b_L*temp
134     d = [[]]*(L+1)
135     d[L] = delta_start
136     for i in range(L, 0, -1):
137         #TODO make sure the range function works as intended
138         if(w[i].shape[1]==1):
139             w[i] = w[i].T
140 
141         b =  calculate_b(w[i-1],V[i-1],O[i-1])
142         delta = calculate_single_d(d[i],w[i],b)
143         d[i-1] = delta
144     return d
145 
146 
147 
148 def propagate_backwards(V : list, O : list, w : list, t : int, learning_rate:
    float, batchsize : int):
149     #TODO include support for batchsize
150     L = len(w)
151     d = calculate_all_d(V,O,w,t)
152     for l in range(L):
153         if(w[l].shape[1]==1):
154             d_w = learning_rate*(d[l]@V[l])
155         else:
156             d_w = learning_rate*(d[l]@V[l].T)
157         d_O = learning_rate* (d[l]@np.ones((batchsize,1)))
158         w[l] +=  d_w
159         O[l] -= d_O
160 
161 
162 def create_multible_layers(*argv):
163     w = []
164     V = []
165     O = []
166     inputlayer = np.zeros((argv[0],1))
167     V.append(inputlayer)
168     old_layersize = argv[0]
169     for layersize in argv[1:]:
170         O_i = np.zeros((layersize,1))
171         V_i, w_i = create_layer(old_layersize,layersize)
172         old_layersize = layersize
173         w.append(w_i)
174         V.append(V_i)
175         O.append(O_i)
176     return (w,V,O)
177 
178 
```

```
179 def learn(learning_rate:float, data: list, labels: list,test_data : list,
    test_labels : list, batchsize: int, *argv):
180
181     w,V,O = create_multible_layers(*argv)
182     #merged = np.concatenate([data, labels], axis=1)
183     return_w = []
184     return_c = []
185     return_O = []
186     for T in range(20):
187         #Permutate the data in the beginning of each epoch
188         #TODO make sure that this works as intended
189         #np.random.shuffle(merged)
190         #data, labels = np.split(merged, [data.shape[1]], axis=1)
191         data,labels = shuffle(data,labels)
192         for i in range(0, len(data), batchsize):
193         # Add test after each epoch (also easy)
194             #print("test")
195             my = i
196             bs = batchsize
197             if my + batchsize >= len(data):
198                 bs =  len(data) - my
199             V[0] = np.array(data[my:my+bs]).T
200             t = np.array(labels[my:my+bs]).T
201             #t = t.reshape(bs,t.shape[1])
202
203             propagate_forward(V,O,w, bs)
204             propagate_backwards(V,O,w,t,learning_rate, bs)
205         #print(w[0])
206         C = calculate_classification_error(V,w,O,test_data, test_labels)
207         #print(w[0])
208         return_c.append(C)
209         return_w.append(w)
210         return_O.append(O)
211     print("Finished learning!")
212     return return_c, return_w, V, return_O
213
214
215 def calculate_classification_error(V,w,O, test_data, test_labels):
216     inner_sum = 0
217     pval = len(test_data)
218     for my in range(pval):
219         V[0] = np.array(test_data[my])
220         t_my = test_labels[my]
221         #Propagate Forward
222         propagate_forward(V,O,w,1)
223         #inner_sum += np.abs(esign(V[-1][0])-t_my)
224         y_i = np.argmax(V[-1],axis=0)
225         y = np.zeros((len(V[-1]),1))
226         #print(y_i)
227         y[y_i] = 1
228         inner_sum += np.sum(np.abs(y-t_my))
229         #print("error")
230     return inner_sum/(2*pval)
231
232
233
234 def transformlabels(labels, n):
235     for i,label in enumerate(labels):
236         temp = np.zeros((n,1))
237         temp[label] = 1
```

```
238         labels[i] = temp
239
240
241 folder_name = "/home/edin/uni/ANN/ex-3/cifar-10-batches-py/"
242 batch_1 = unpickle(folder_name+"data_batch_1")
243 batch_2 = unpickle(folder_name+"data_batch_2")
244 batch_3 = unpickle(folder_name+"data_batch_3")
245 batch_4 = unpickle(folder_name+"data_batch_4")
246 batch_5 = unpickle(folder_name+"data_batch_5")
247
248 data = batch_1[b'data']
249 data = np.append(data,batch_2[b'data'], axis=0)
250 data = np.append(data,batch_3[b'data'], axis=0)
251 data = np.append(data,batch_4[b'data'], axis=0)
252 #data = np.append(data,batch_5[b'data'], axis=0)
253 labels = batch_1[b'labels']
254 #Since labels are only a normal array
255 labels += batch_2[b'labels']
256 labels += batch_3[b'labels']
257 labels += batch_4[b'labels']
258 #labels += batch_5[b'labels']
259 data = center_data(data)
260 transformlabels(labels,10)
261 labels = np.array(labels)
262 labels = labels.astype(np.uint8)
263 labels = np.array(labels).reshape(40000,10)
264 test_batch = unpickle(folder_name+"test_batch")
265 test_data = batch_5[b'data']
266 test_labels = batch_5[b'labels']
267 test_data = center_data(test_data)
268 transformlabels(test_labels,10)
269 test_labels = np.array(test_labels)
270
271 test_labels = test_labels.astype(np.uint8)
272
273 #IT DOES NOT WORK
274 #POSSIBLE PROBLEMS:
275 # Permutation is not correct
276 # -> Add assert statements or smth
277 # Centering Function is fucked up
278 # Batch processing is fucked up
279 # -> Test with Aufgabe 2.1 and see if it works
280
281
282 minibatch_size = 100
283 learning_rate = 0.1
284
285 #Network 1
286 c1,w1, V1,O1 = learn(learning_rate,data,labels,test_data, test_labels,
    minibatch_size, 3072,10)
287 plt.plot(c1, color="blue",label="Network 1")
288 #Network 2
289 c2,w2,V2,O2 = learn(learning_rate,data,labels,test_data, test_labels,
    minibatch_size, 3072,10,10)
290 plt.plot(c2, color = "red", label = "Network 2")
291 #Network 3
292 c3,w3,V3,O3 = learn(learning_rate,data,labels,test_data, test_labels,
    minibatch_size, 3072,50,10)
293 plt.plot(c3, color = "brown", label = "Network 3")
294
```

```
295
296 #Network 4
297 c4,w4,V4,O4 = learn(learning_rate,data,labels,test_data, test_labels,
    minibatch_size, 3072,50,50,10)
298 plt.plot(c4, color = "green", label = "Network 4")
299 plt.yscale("log")
300 plt.plot(c1, color="blue",label="Network 1")
301 plt.plot(c2, color = "red", label = "Network 2")
302 plt.plot(c3, color = "brown", label = "Network 3")
303 plt.plot(c4, color = "green", label = "Network 4")
304 plt.legend()
305 plt.show()
306 #lets make a table!
307 def calculate_table_entries(C,w,V,O, showcase_data, showcase_labels):
308     max_i = np.argmin(C,axis=0)
309     classif_error_test =
    calculate_classification_error(V,w[max_i],O[max_i],showcase_data,
    showcase_labels)
310     classif_error_train =
    calculate_classification_error(V,w[max_i],O[max_i],data, labels)
311     return [max_i, C[max_i], classif_error_test,classif_error_train]
312
313
314 batch_showcase = batch_1 = unpickle(folder_name+"test_batch")
315 showcase_data = batch_showcase[b'data']
316 showcase_labels = batch_showcase[b'labels']
317 showcase_data = center_data(showcase_data)
318 transformlabels(showcase_labels,10)
319
320 #Lets get the classification_error for Network 1
321 from tabulate import tabulate
322 column1 = [1] + calculate_table_entries(c1,w1,V1,O1,showcase_data,
    showcase_labels)
323 column2 = [2] + calculate_table_entries(c2,w2,V2,O2,showcase_data,
    showcase_labels)
324 column3 = [3] + calculate_table_entries(c3,w3,V3,O3,showcase_data,
    showcase_labels)
325 column4 = [4] + calculate_table_entries(c4,w4,V4,O4,showcase_data,
    showcase_labels)
326 columnlabels = ["Network","best epoch", "C validation set", "C test set", "C
    training set"]
327 rowlabels = ["network 1", "network 2", "network 3", "network 4"]
328 print(tabulate([column1, column2, column3,column4], headers=columnlabels))
```