Lab2_Q1a

March 20, 2020

ComS 573

Lab 2

Kanak Choudhury

1 Problem 1

1.1 (a)

For this problem, I have used the following parameter combinations

```
hidden_layers = [1,2,3]
hidden_units = [50, 64, 80]
num_epochs = [10, 50, 100]
btch_size = [128, 200, 300]
learning_rate = [0.1, 0.5, 0.9]
momentum = [.3, .5, 0.9]
loss_func = ['mean_squared_error', 'categorical_crossentropy']
data_scaling = ['Standardize', 'Normalize']
activation_func = ['relu']
```

Also used 80% - 20% training - validation data.

[1]:
```
import numpy as np
import pandas as pd
import sklearn.preprocessing
import matplotlib
import keras
import re
import sys
import gc
import time

print('python ' +sys.version)
print('numpy '+ np.__version__)
print('pandas '+ pd.__version__)
print('sklearn '+ sklearn.__version__)
print('matplotlib '+ matplotlib.__version__)
print('keras '+ keras.__version__)
```

```python
print('re '+ re.__version__)

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt

from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from itertools import product
```

```
Using TensorFlow backend.
```

```
python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
numpy 1.18.1
pandas 1.0.1
sklearn 0.22.1
matplotlib 3.1.3
keras 2.3.1
re 2.2.1
```

[2]:

```python
path  = 'D:/ISU/COMS 573 - Machine Learning/HW/Lab2/'

train_model = False

df_tr = pd.read_csv(path+'optdigits.tra',header=None)
X_tr, y_tr = df_tr.loc[:,0:63], df_tr.loc[:,64]
ccat = y_tr.unique().size

df_ts = pd.read_csv(path+'optdigits.tes',header=None)
X_ts,  y_ts  = df_ts.loc[:,0:63],  df_ts.loc[:,64]

scaler = StandardScaler().fit(X_tr)
normalizer = Normalizer().fit(X_tr)

X_tr_std = scaler.transform(X_tr)
X_tr_norm = normalizer.transform(X_tr)


split = 0.8
size = np.shape(X_tr)
nsplit = int(np.floor(split*size[0]))
```

```
y_train1 = np_utils.to_categorical(y_tr, ccat)
y_train = y_train1[0:nsplit,:];
y_val = y_train1[nsplit:size[0],:];
y_test = np_utils.to_categorical(y_ts, ccat)


X_train_std = X_tr_std[0:nsplit,:];
X_val_std = X_tr_std[nsplit:size[0],:];
X_test_std = scaler.transform(X_ts)


X_train_norm = X_tr_norm[0:nsplit,:];
X_val_norm = X_tr_norm[nsplit:size[0],:];
X_test_norm = normalizer.transform(X_ts)
```

[3]:
```
if train_model:
    hidden_layers = [1,2,3]
    hidden_units = [50, 64, 80]
    num_epochs = [10, 50, 100]
    btch_size = [128, 200, 300]
    learning_rate = [0.1, 0.5, 0.9]
    momentum = [.3, .5, 0.9]
    loss_func = ['mean_squared_error', 'categorical_crossentropy']
    data_scaling = ['Standardize', 'Normalize']
    activation_func = ['relu']


    def expand_grid(dictionary):
       return pd.DataFrame([row for row in product(*dictionary.values())],
                           columns=dictionary.keys())

    dictionary = {'hidden_layers': hidden_layers,
                  'hidden_units': hidden_units,
                  'num_epochs': num_epochs,
                  'batch_size': btch_size,
                  'learning_rate': learning_rate,
                  'momentum': momentum,
                  'loss_func': loss_func,
                  'data_scaling': data_scaling,
                  'activation_func': activation_func}

    prem = expand_grid(dictionary)
    prem = prem[~((prem['activation_func'] == 'tanh') & (prem['loss_func'] ==␣
 ↪'mean_squared_error'))]
    prem['time'] = np.NaN
    prem['train_loss'] = np.NaN
    prem['validation_loss'] = np.NaN
```

```
    prem['test_loss'] = np.NaN
    prem['train_acc'] = np.NaN
    prem['validation_acc'] = np.NaN
    prem['test_acc'] = np.NaN
    size_prem = prem.shape
    print(prem.head())

    ll = 0
    for j in range(0,2):
        if j == 0:
            X_train = X_train_std
            X_val = X_val_std
            X_test = X_test_std
            listind = prem[(prem['data_scaling'] == 'Standardize') & (prem.
↪isnull().any(axis=1))].index.tolist()
        else:
            X_train = X_train_norm
            X_val = X_val_norm
            X_test = X_test_norm
            listind = prem[(prem['data_scaling'] == 'Normalize') & (prem.
↪isnull().any(axis=1))].index.tolist()

        for i in listind:
            start = time. time()
            if prem.iloc[i,0] == 1:
                model = Sequential()
                model.add(Dense(prem.iloc[i,1], input_dim=64, activation=prem.
↪iloc[i,8]))
                model.add(Dense(ccat, activation='softmax'))

            elif prem.iloc[i,0] == 2:
                model = Sequential()
                model.add(Dense(prem.iloc[i,1], input_dim=64, activation=prem.
↪iloc[i,8]))
                model.add(Dense(prem.iloc[i,1], activation=prem.iloc[i,8]))
                model.add(Dense(ccat, activation='softmax'))

            elif prem.iloc[i,0] == 3:
                model = Sequential()
                model.add(Dense(prem.iloc[i,1], input_dim=64, activation=prem.
↪iloc[i,8]))
                model.add(Dense(prem.iloc[i,1], activation=prem.iloc[i,8]))
                model.add(Dense(prem.iloc[i,1], activation=prem.iloc[i,8]))
                model.add(Dense(ccat, activation='softmax'))

            else:
                model = Sequential()
```

```
                model.add(Dense(prem.iloc[i,1], input_dim=64, activation=prem.
↪iloc[i,8]))
                model.add(Dense(prem.iloc[i,1], activation=prem.iloc[i,8]))
                model.add(Dense(prem.iloc[i,1], activation=prem.iloc[i,8]))
                model.add(Dense(prem.iloc[i,1], activation=prem.iloc[i,8]))
                model.add(Dense(ccat, activation='softmax'))

            es = EarlyStopping(monitor='val_accuracy', mode='max', verbose=0,␣
↪patience=200)
            mc = ModelCheckpoint('best_model', monitor='val_accuracy',␣
↪mode='max', verbose=0, save_best_only=True)

            optimizer1 = optimizers.SGD(lr=prem.iloc[i,4], momentum=prem.
↪iloc[i,5])
            model.compile(optimizer=optimizer1, loss=prem.iloc[i,6],␣
↪metrics=['accuracy'])
            fit1 = model.fit(X_train,y_train, batch_size=prem.iloc[i,3],␣
↪epochs=prem.iloc[i,2],
                             validation_data=(X_val,y_val), callbacks=[es, mc],␣
↪verbose = 0)
            fit = load_model('best_model')
            end = time.time()

            train_accuracy = fit.evaluate(X_train, y_train, verbose=0)
            val_accuracy = fit.evaluate(X_val, y_val, verbose=0)
            test_accuracy = fit.evaluate(X_test, y_test, verbose=0)
            prem.iloc[i, 9:16] = [end-start, train_accuracy[0],␣
↪val_accuracy[0], test_accuracy[0],
                                  train_accuracy[1], val_accuracy[1],␣
↪test_accuracy[1]]

            del model, es, mc, optimizer1, fit, fit1
            gc.collect()
            ll = ll+1
            sys.stdout.write("\r Progress: %.2f%%" %round(float(ll)/
↪size_prem[0]*100,2))
            sys.stdout.flush()
else:
    print('skiped model fit')
```

```
skiped model fit
```

[4]:
```
if train_model:
    prem.to_csv (path+'res_1a.csv', index = False, header=True)
else:
    prem = pd.read_csv(path+'res_1a.csv',header=0)
```

```
    prem.head(15)

top10_mse = prem[prem['loss_func'] == 'mean_squared_error'].
 ↪nlargest(10,'test_acc')
top10_cce = prem[prem['loss_func'] == 'categorical_crossentropy'].
 ↪nlargest(10,'test_acc')
print('\n Best 10 hyper-parameter combination for Cross-Entropy:\n',␣
 ↪round(top10_cce, 4))
print('\n Best 10 hyper-parameter combination for Mean-Squared-Error:\n',␣
 ↪round(top10_mse, 4))

plt.hist([prem[prem['loss_func'] == 'mean_squared_error'].iloc[:,9],
          prem[prem['loss_func'] == 'categorical_crossentropy'].iloc[:,9]],
         bins=300, density=True, alpha=0.5, label=['mean_squared_error',␣
 ↪'categorical_crossentropy'])
plt.legend(loc='upper right')
plt.title('Distribution of time to fit models')
plt.xlim(0, 200)
plt.show()

plt.hist([prem[prem['loss_func'] == 'mean_squared_error'].iloc[:,15],
          prem[prem['loss_func'] == 'categorical_crossentropy'].iloc[:,15]],
         density=True, alpha=0.5, label=['mean_squared_error',␣
 ↪'categorical_crossentropy'])
plt.legend(loc='upper left')
plt.title('Distribution of test accuracy')
plt.show()
```

```
 Best 10 hyper-parameter combination for Cross-Entropy:
       hidden_layers  hiden_units  num_epochs  batch_size  learning_rate  \
1430              2           64          50         128            0.9
486               1           64          50         200            0.5
782               1           80          50         128            0.9
1290              2           50         100         300            0.9
1522              2           64         100         128            0.1
2766              3           80          50         200            0.9
2854              3           80         100         200            0.1
2823              3           80         100         128            0.5
1427              2           64          50         128            0.5
1571              2           64         100         200            0.5

      momentum                 loss_func data_scaling activation_func  \
1430       0.3  categorical_crossentropy  Standardize            relu
486        0.5  categorical_crossentropy  Standardize            relu
782        0.3  categorical_crossentropy  Standardize            relu
1290       0.5  categorical_crossentropy  Standardize            relu
```

1522	0.9	categorical_crossentropy	Standardize	relu
2766	0.5	categorical_crossentropy	Standardize	relu
2854	0.9	categorical_crossentropy	Standardize	relu
2823	0.3	categorical_crossentropy	Normalize	relu
1427	0.9	categorical_crossentropy	Normalize	relu
1571	0.9	categorical_crossentropy	Normalize	relu

	time	train_loss	validation_loss	test_loss	train_acc \
1430	5.8781	0.0013	0.0781	0.1324	1.0000
486	5.0245	0.0110	0.0740	0.1119	0.9997
782	4.8596	0.0016	0.0455	0.1397	1.0000
1290	6.5375	0.0030	0.0699	0.1386	1.0000
1522	8.1272	0.0019	0.0760	0.1469	1.0000
2766	8.6308	0.0009	0.0869	0.1967	1.0000
2854	17.2453	0.0011	0.0811	0.1428	1.0000
2823	61.7760	0.0106	0.0743	0.1560	0.9974
1427	64.7441	0.0025	0.0735	0.1666	0.9997
1571	32.2653	0.0020	0.0926	0.1864	0.9997

	validation_acc	test_acc
1430	0.9869	0.9699
486	0.9817	0.9694
782	0.9882	0.9672
1290	0.9856	0.9672
1522	0.9830	0.9672
2766	0.9817	0.9672
2854	0.9843	0.9672
2823	0.9856	0.9666
1427	0.9869	0.9661
1571	0.9869	0.9661

Best 10 hyper-parameter combination for Mean-Squared-Error:

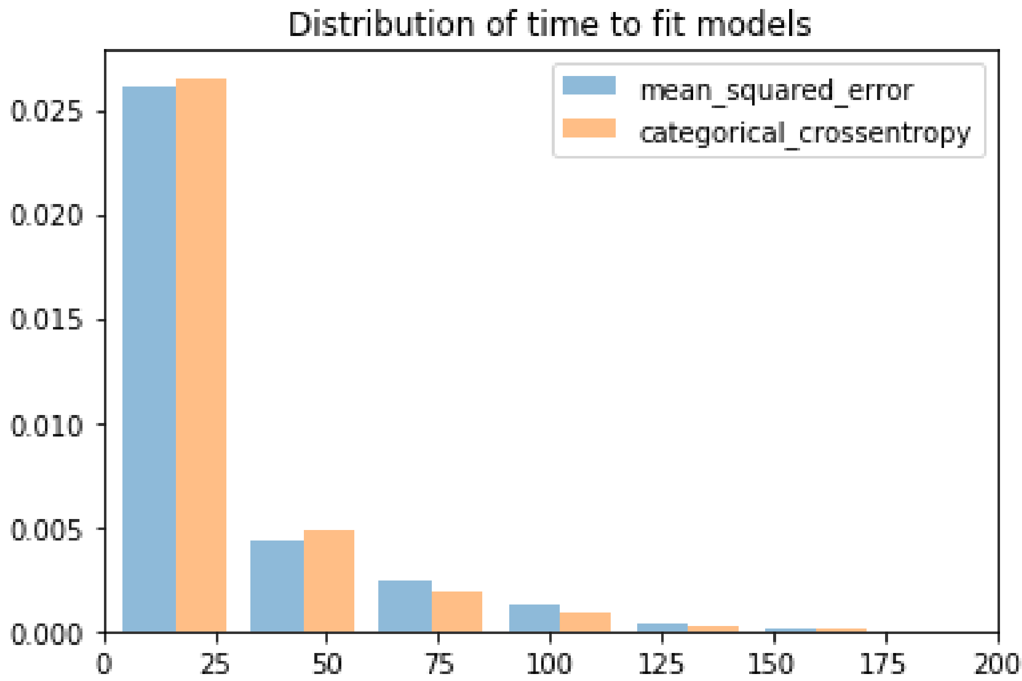
	hidden_layers	hiden_units	num_epochs	batch_size	learning_rate \
1432	2	64	50	128	0.9
1864	2	80	100	128	0.9
308	1	50	100	300	0.5
812	1	80	50	200	0.5
872	1	80	100	128	0.1
1508	2	64	50	300	0.9
464	1	64	50	128	0.9
1292	2	50	100	300	0.9
896	1	80	100	128	0.9
920	1	80	100	200	0.5

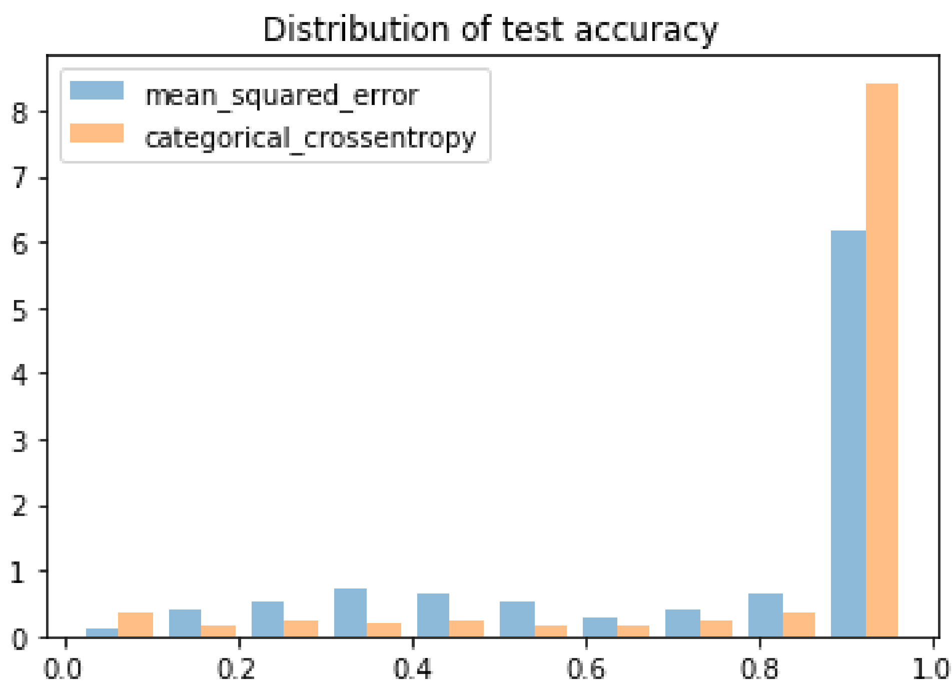
	momentum	loss_func	data_scaling	activation_func	time \
1432	0.5	mean_squared_error	Standardize	relu	6.9287
1864	0.5	mean_squared_error	Standardize	relu	11.5742
308	0.9	mean_squared_error	Standardize	relu	5.8434

812	0.9	mean_squared_error	Standardize	relu	5.9002
872	0.9	mean_squared_error	Standardize	relu	8.8662
1508	0.9	mean_squared_error	Standardize	relu	5.5218
464	0.9	mean_squared_error	Standardize	relu	4.5238
1292	0.9	mean_squared_error	Standardize	relu	13.2303
896	0.9	mean_squared_error	Standardize	relu	8.1258
920	0.9	mean_squared_error	Standardize	relu	6.9174

	train_loss	validation_loss	test_loss	train_acc	validation_acc \
1432	0.0006	0.0032	0.0055	0.9984	0.9817
1864	0.0002	0.0030	0.0054	0.9993	0.9856
308	0.0006	0.0035	0.0057	0.9977	0.9804
812	0.0008	0.0035	0.0054	0.9980	0.9843
872	0.0012	0.0041	0.0058	0.9964	0.9791
1508	0.0006	0.0038	0.0054	0.9971	0.9791
464	0.0007	0.0032	0.0055	0.9974	0.9830
1292	0.0004	0.0033	0.0055	0.9977	0.9817
896	0.0003	0.0034	0.0059	0.9987	0.9843
920	0.0008	0.0034	0.0057	0.9977	0.9843

	test_acc
1432	0.9666
1864	0.9661
308	0.9655
812	0.9655
872	0.9644
1508	0.9644
464	0.9638
1292	0.9638
896	0.9633
920	0.9633





[5]:
```
aaa = prem[prem['loss_func'] == 'mean_squared_error'].iloc[:,9]
bbb = prem[prem['loss_func'] == 'categorical_crossentropy'].iloc[:,9]
```

```
print("Mean and Variance of fitted time:\n mean_squared_error: Mean = %.2f, \
var = %.2f\n categorical_crossentropy: Mean = %.2f, \
var = %.2f\n" %(np.mean(aaa), np.var(aaa), np.mean(bbb), np.var(bbb)))

aaa = prem[prem['loss_func'] == 'mean_squared_error'].iloc[:,15]
bbb = prem[prem['loss_func'] == 'categorical_crossentropy'].iloc[:,15]
print("Mean and Variance of test accuracy:\n mean_squared_error: Mean = %.4f, \
var = %.4f\n categorical_crossentropy: Mean = %.4f, \
var = %.4f\n" %(np.mean(aaa), np.var(aaa), np.mean(bbb), np.var(bbb)))
```

```
Mean and Variance of fitted time:
 mean_squared_error: Mean = 30.62, var = 18043.78
 categorical_crossentropy: Mean = 37.11, var = 70030.07

Mean and Variance of test accuracy:
 mean_squared_error: Mean = 0.7483, var = 0.0726
 categorical_crossentropy: Mean = 0.8520, var = 0.0515
```

[6]:
```
for i in range(2):
    if i==1:
        top10 = top10_mse
        print("\n Results For Mean-Squared-Error")
        print("**********************************\n")
    else:
        top10 = top10_cce
        print("\n Results For Cross-Entropy")
        print("**********************************\n")

    if top10.iloc[0,7] == 'Standardize':
        X_train = X_train_std
        X_val = X_val_std
        X_test = X_test_std
    else:
        X_train = X_train_norm
        X_val = X_val_norm
        X_test = X_test_norm


    start = time. time()
    if top10.iloc[0,0] == 1:
        model = Sequential()
        model.add(Dense(top10.iloc[0,1], input_dim=64, activation=top10.
↪iloc[0,8]))
        model.add(Dense(ccat, activation='softmax'))

    elif top10.iloc[0,0] == 2:
```

```
        model = Sequential()
        model.add(Dense(top10.iloc[0,1], input_dim=64, activation=top10.
↪iloc[0,8]))
        model.add(Dense(top10.iloc[0,1], activation=top10.iloc[0,8]))
        model.add(Dense(ccat, activation='softmax'))

    elif top10.iloc[0,0] == 3:
        model = Sequential()
        model.add(Dense(top10.iloc[0,1], input_dim=64, activation=top10.
↪iloc[0,8]))
        model.add(Dense(top10.iloc[0,1], activation=top10.iloc[0,8]))
        model.add(Dense(top10.iloc[0,1], activation=top10.iloc[0,8]))
        model.add(Dense(ccat, activation='softmax'))

    else:
        model = Sequential()
        model.add(Dense(top10.iloc[0,1], input_dim=64, activation=top10.
↪iloc[0,8]))
        model.add(Dense(top10.iloc[0,1], activation=top10.iloc[0,8]))
        model.add(Dense(top10.iloc[0,1], activation=top10.iloc[0,8]))
        model.add(Dense(top10.iloc[0,1], activation=top10.iloc[0,8]))
        model.add(Dense(ccat, activation='softmax'))

    es = EarlyStopping(monitor='val_accuracy', mode='max', verbose=0,␣
↪patience=200)
    mc = ModelCheckpoint('best_model', monitor='val_accuracy', mode='max',␣
↪verbose=0, save_best_only=True)

    optimizer1 = optimizers.SGD(lr=top10.iloc[0,4], momentum=top10.iloc[0,5])
    model.compile(optimizer=optimizer1, loss=top10.iloc[0,6],␣
↪metrics=['accuracy'])
    fit1 = model.fit(X_train,y_train, batch_size=top10.iloc[0,3], epochs=top10.
↪iloc[0,2],
                    validation_data=(X_val,y_val), callbacks=[es, mc], verbose␣
↪= 0)
    fit = load_model('best_model')
    end = time.time()

    train_accuracy = fit.evaluate(X_train, y_train, verbose=0)
    val_accuracy = fit.evaluate(X_val, y_val, verbose=0)
    test_accuracy = fit.evaluate(X_test, y_test, verbose=0)
    final_res = [end-start, train_accuracy[0], val_accuracy[0],␣
↪test_accuracy[0],
                train_accuracy[1], val_accuracy[1], test_accuracy[1]]

    if top10.iloc[0,7] == 'Standardize':
```

```
        X_train11 = X_tr_std
        X_test = X_test_std
        y_train11 = y_train1
    else:
        X_train11 = X_tr_norm
        X_test = X_test_norm
        y_train11 = y_train1

    print("For hyper-parameters:\n",top10.iloc[0,:])
    print("\n Time needed: %.2f" % (end-start))
    scores = fit.evaluate(X_test, y_test, verbose=0)
    print("\n Test Accuracy: %.2f%%" % (scores[1]*100))

    A = fit.predict(X_train11)
    cm = confusion_matrix(y_train11.argmax(axis=1), A.argmax(axis=1))
    print("\n Train confusion matrix: \n", cm)
    acc_train = np.diagonal(cm)/cm.sum(axis=1)
    print("\n Class Accuracy for Training Data is:")
    for i in range(10):
        print('Class %d: %.2f%%' %(i, acc_train[i]*100))

    A = fit.predict(X_test)
    cm = confusion_matrix(y_test.argmax(axis=1), A.argmax(axis=1))
    print("\n Test confusion matrix: \n", cm)
    acc_test = np.diagonal(cm)/cm.sum(axis=1)
    print("\n Class Accuracy for Testing Data is:")
    for i in range(10):
        print('Class %d: %.2f%%' %(i, acc_test[i]*100))
    print("**********************************\n")
```

```
 Results For Cross-Entropy
**********************************

For hyper-parameters:
 hidden_layers                          2
hiden_units                            64
num_epochs                             50
batch_size                            128
learning_rate                         0.9
momentum                              0.3
loss_func         categorical_crossentropy
data_scaling                  Standardize
activation_func                      relu
time                              5.87814
train_loss                     0.00131139
validation_loss                 0.0780658
```

```
test_loss                          0.132371
train_acc                                 1
validation_acc                     0.986928
test_acc                            0.96995
Name: 1430, dtype: object

 Time needed: 5.17

 Test Accuracy: 96.61%

 Train confusion matrix:
 [[376   0   0   0   0   0   0   0   0   0]
 [  0 388   0   0   0   0   1   0   0   0]
 [  0   0 378   1   0   0   0   0   1   0]
 [  0   1   0 386   0   1   0   0   0   1]
 [  0   0   0   0 386   0   1   0   0   0]
 [  0   0   0   0   0 376   0   0   0   0]
 [  0   0   0   0   0   0 377   0   0   0]
 [  0   0   0   1   0   0   0 386   0   0]
 [  0   0   0   1   0   0   0   1 378   0]
 [  0   0   0   1   1   0   0   0   1 379]]

 Class Accuracy for Training Data is:
Class 0: 100.00%
Class 1: 99.74%
Class 2: 99.47%
Class 3: 99.23%
Class 4: 99.74%
Class 5: 100.00%
Class 6: 100.00%
Class 7: 99.74%
Class 8: 99.47%
Class 9: 99.21%

 Test confusion matrix:
 [[178   0   0   0   0   0   0   0   0   0]
 [  0 178   0   0   0   0   3   0   1   0]
 [  0   2 170   1   0   0   3   1   0   0]
 [  0   0   3 177   0   2   0   0   0   1]
 [  0   1   0   0 176   0   0   1   3   0]
 [  0   1   0   0   1 178   0   0   0   2]
 [  0   0   0   0   2   0 179   0   0   0]
 [  0   0   0   0   1   2   0 172   1   3]
 [  0   6   0   2   1   1   1   0 155   8]
 [  0   0   1   1   1   3   0   0   1 173]]

 Class Accuracy for Testing Data is:
Class 0: 100.00%
```

```
Class 1: 97.80%
Class 2: 96.05%
Class 3: 96.72%
Class 4: 97.24%
Class 5: 97.80%
Class 6: 98.90%
Class 7: 96.09%
Class 8: 89.08%
Class 9: 96.11%
**********************************


 Results For Mean-Squared-Error
**********************************

For hyper-parameters:
 hidden_layers                         2
hiden_units                           64
num_epochs                            50
batch_size                           128
learning_rate                        0.9
momentum                             0.5
loss_func             mean_squared_error
data_scaling                 Standardize
activation_func                     relu
time                             6.92868
train_loss                   0.000572706
validation_loss               0.00319363
test_loss                      0.0054562
train_acc                       0.998365
validation_acc                  0.981699
test_acc                        0.966611
Name: 1432, dtype: object

 Time needed: 5.94

 Test Accuracy: 95.94%

 Train confusion matrix:
 [[374   0   0   0   1   0   1   0   0   0]
 [  0 385   0   0   0   0   1   1   0   2]
 [  0   1 376   1   0   1   0   0   0   1]
 [  0   0   0 383   0   4   0   0   0   2]
 [  0   0   0   0 386   0   1   0   0   0]
 [  0   0   0   0   0 374   0   0   0   2]
 [  0   2   0   0   1   0 373   0   1   0]
 [  0   0   0   1   0   0   0 386   0   0]
 [  0   2   0   0   0   0   0   0 378   0]
```

```
 [  0   0   0   1   2   0   0   0   1 378]]

 Class Accuracy for Training Data is:
Class 0: 99.47%
Class 1: 98.97%
Class 2: 98.95%
Class 3: 98.46%
Class 4: 99.74%
Class 5: 99.47%
Class 6: 98.94%
Class 7: 99.74%
Class 8: 99.47%
Class 9: 98.95%

 Test confusion matrix:
 [[174   0   0   0   1   1   1   0   0   1]
 [  0 176   0   0   0   0   2   0   3   1]
 [  0   1 172   0   0   0   0   1   3   0]
 [  0   0   3 170   0   2   0   4   3   1]
 [  0   1   0   0 177   0   0   1   2   0]
 [  0   0   0   1   0 179   0   0   0   2]
 [  1   1   0   0   0   0 178   0   1   0]
 [  0   1   0   0   1   0   0 165   1  11]
 [  0   7   0   2   1   0   0   0 161   3]
 [  0   0   0   1   0   4   0   0   3 172]]

 Class Accuracy for Testing Data is:
Class 0: 97.75%
Class 1: 96.70%
Class 2: 97.18%
Class 3: 92.90%
Class 4: 97.79%
Class 5: 98.35%
Class 6: 98.34%
Class 7: 92.18%
Class 8: 92.53%
Class 9: 95.56%
**********************************
```

Based on the time distribution, though both mean-squared-error and cross-entropy look like have the same distribution, but cross-entropy has higher mean and variance compare to MSE.

However, based on test accuracy distributions for MSE and cross-entropy, clearly cross-entropy has higher test accuracy than MSE loss function. The average test accuracy for all combinations of hyper-parameter is higher for cross-entropy loss function compare to MSE and lower variance for cross-entropy than MSE. This indicates that for multi-category classification, it is better to use cross-entropy compare to MSE loss function.

It is found that using cross-entropy loss function with 2 hidden layers, 64 units, number of epochs 50, batch size 128, learning rate 0.9 and momentum 0.3 has the highest test accuracy (around 96.00%). Note that, this model was fitted based on only 1-fold cross validation with no repeated sample. It might be different if we use repeated k fold cross validation.

Training accuracy for all classes are almost 100%. However, test accuracy for all classes are around 96% for cross-entropy loss function which are higher than the MSE. Class 0 has the highest test accuracy and class 8 has the lowest accuracy for cross-entropy loss function. Also, similar pattern has been found for the MSE loss function with comparatively lower accuracy than cross-entropy. Overall classification accuracy, class accuracy, and confusion matrix for both training and testing data are given in above tables.

1.2 Important References:

1. https://towardsdatascience.com/building-our-first-neural-network-in-keras-bdc8abbc17f5
2. https://towardsdatascience.com/building-a-deep-learning-model-using-keras-1548ca149d37
3. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
4. https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5
5. https://towardsdatascience.com/convolutional-neural-networks-for-beginners-practical-guide-with-python-and-keras-dc688ea90dca


```
[ ]:
```