Multi-Layer Perceptrons (Lab2:**part 3**)

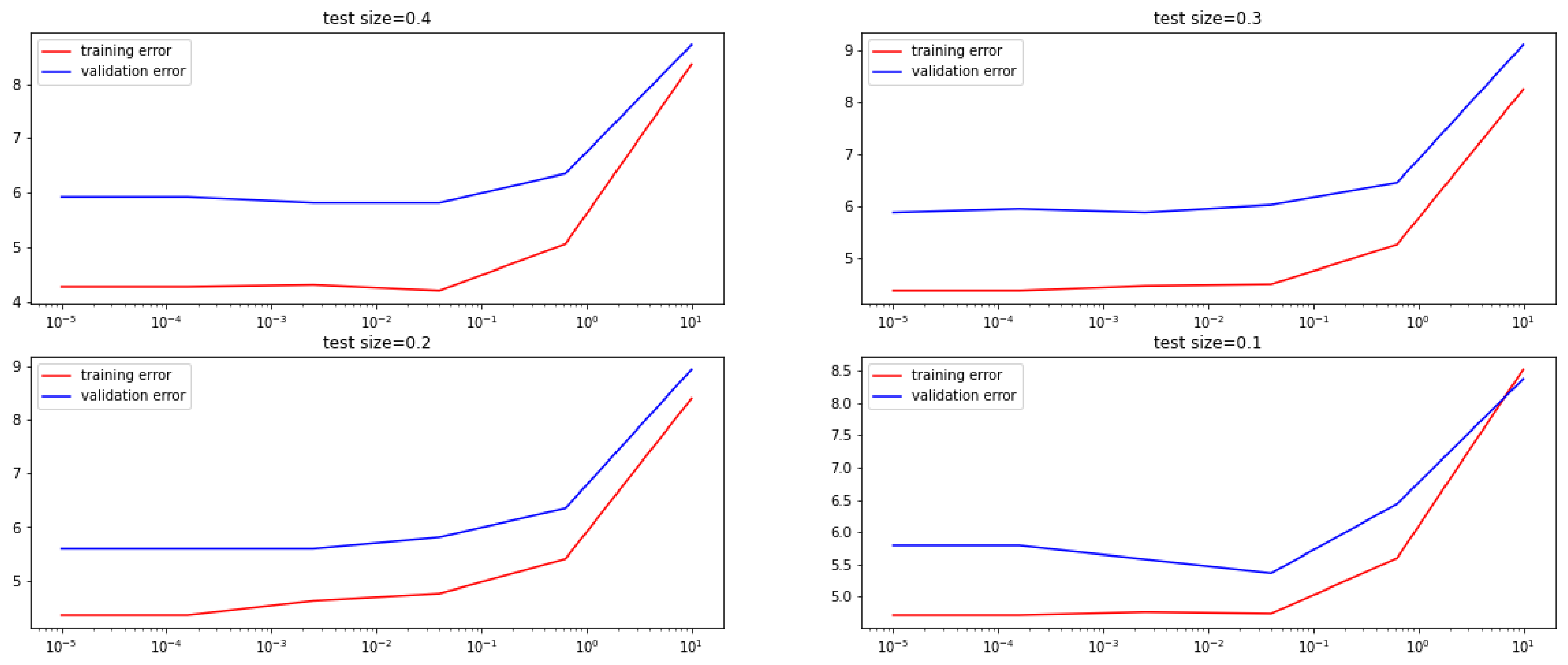
Train a Multi-Layer perceptron on "*a1a*" dataset, using the cross-entropy loss with ℓ -2 regularization.

```
!wget -t inf https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/a1a
from sklearn.datasets import load_svmlight_file
X, y = load_svmlight_file("a1a")
from sklearn.neural_network import MLPClassifier
from sklearn import preprocessing
from sklearn.metrics import log_loss
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
```

```
alpha = np.logspace(-5, 1,num=6) #alphas are expected to be in general <=1 (not necessary though)
sizes=[ 0.4, 0.3, 0.2, 0.1]

#initializations
training_error=np.zeros((4, 6))
validation_error=np.zeros((4, 6))



for i,size in enumerate(sizes):  #iterate over 4 different dataset splits
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size, random_state=42)

  for j,a in enumerate(alpha):
        mlp = MLPClassifier( solver='adam',alpha=a, activation = 'logistic',
                        hidden_layer_sizes=(16,8),
                        max_iter=500, random_state=1)
        mlp.fit(X_train,y_train)

        training_error[i][j]=log_loss(y_train, mlp.predict(X_train))
        validation_error[i][j]= log_loss(y_test, mlp.predict(X_test))
```

Analysing training / validation errors and their relation , as a function of α parameter, for different test sizes.

```
#plots
plt.rcParams["figure.figsize"] = [20, 8]
fig, axs = plt.subplots(2, 2)
for i,size in enumerate(sizes):
  if i<2:
    axs[0, i].semilogx(alpha, training_error[i,:], '-r')
    axs[0, i].semilogx(alpha, validation_error[i,:], '-b')
    axs[0, i].set_title('test size={}'.format(size))
    axs[0, i].legend(('training error','validation error'),loc="upper left")

  else:
    axs[1, i-2].semilogx(alpha, training_error[i,:], '-r')
    axs[1, i-2].semilogx(alpha, validation_error[i,:], '-b')
    axs[1, i-2].set_title('test size={}'.format(size))
    axs[1, i-2].legend(('training error','validation error'),loc="upper left")
```



Observations

In all 4 cases, the results regarding the α (or "λ") $\ell 2$-regularization factor are coherent with theory:

- For extremely **low values** of α our classification algorithm shows **HIGH VARIANCE**. That is to say *training* and *validation error* are far apart from each other and as the α is not "*penalizing*" enough, the MLP seems to overfit. Clearly the optimization at this point is focusing more on minimizing the training error.
- All the way to the other extreme, **high values** of α (eg.10) are somehow "*over-regularizing*", in a way that the losses converge to really high levels. This situation of **HIGH BIAS** mainly shows that we're penalizing too much and in fact the MLP doesn't get the chance to actually learn and perform well even on the training set to begin with.
- For all cases, a "*sweet-spot*" for α appears to be somewhere in between 10^{-2} and 10^{-1}

- We are expecting for the variance to be mitigated as we would raise the number of training data points (decrease the test size). It is visible that for test size smaller that 30%, the two curves begin to come closer as we're strengthening the regularization. Of course by changing the dataset split proportions, or by simply adding new data, after one point makes no difference, as get to the *high bias* region.

Furthermore:

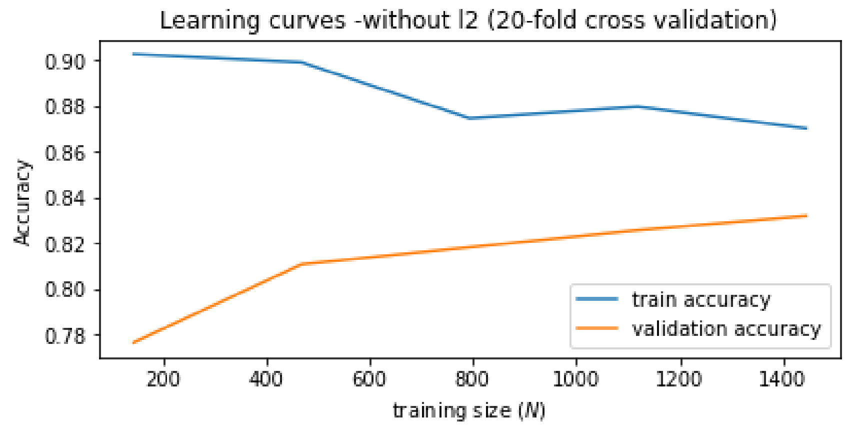
As we can see from the learning curves extracted with the cross-validation method below, a stronger α factor causes a small drop in accuracy in exchange for better convergence.

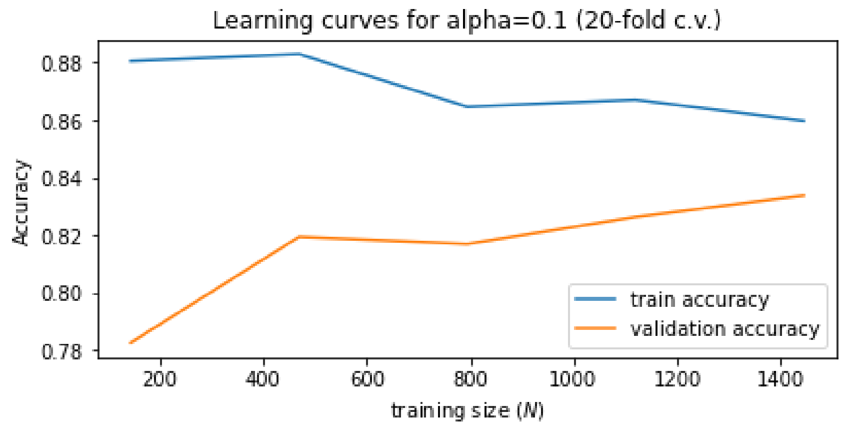
Accuracy learning curves as we add more data for $\alpha = 0.001$ and $\alpha = 0.1$

```
mlp_no_l2 = MLPClassifier( solver='adam', activation = 'logistic',
                                hidden_layer_sizes=(16,8),
                                max_iter=500, random_state=1)
train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(mlp, X, y, cv=10 ,return_times=True) #cross validation of 30 fo

mlp_l2= MLPClassifier( solver='adam', alpha=0.1, activation = 'logistic',
                                hidden_layer_sizes=(16,8),
                                max_iter=500, random_state=1)
train_sizes_l2, train_scores_l2, test_scores_l2, fit_times_l2, _ = learning_curve(mlp_l2, X, y, cv=10 ,return_times=True)
```

```
plt.figure(figsize=(15,3))
plt.subplot(1, 2, 1)
plt.plot(train_sizes,np.mean(train_scores,axis=1))
plt.plot(train_sizes,np.mean(test_scores,axis=1))
plt.xlabel("training size ($N$)")
plt.ylabel("Accuracy")
plt.title('Learning curves -without l2 (20-fold cross validation)')
plt.legend(('train accuracy', 'validation accuracy'))

plt.subplot(1, 2, 2)
plt.plot(train_sizes_l2,np.mean(train_scores_l2,axis=1))
plt.plot(train_sizes_l2,np.mean(test_scores_l2,axis=1))
plt.xlabel("training size ($N$)")
plt.ylabel("Accuracy")
plt.title('Learning curves for alpha=0.1 (20-fold c.v.)')
plt.legend(('train accuracy', 'validation accuracy'))
```

```
<matplotlib.legend.Legend at 0x7f8d17665390>
```





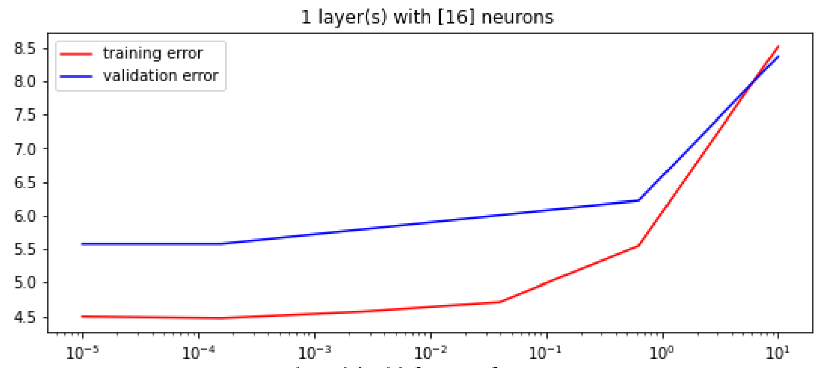
```
alpha = np.logspace(-5, 1,num=6) #alphas are expected to be in general <=1 (not necessary though)
```

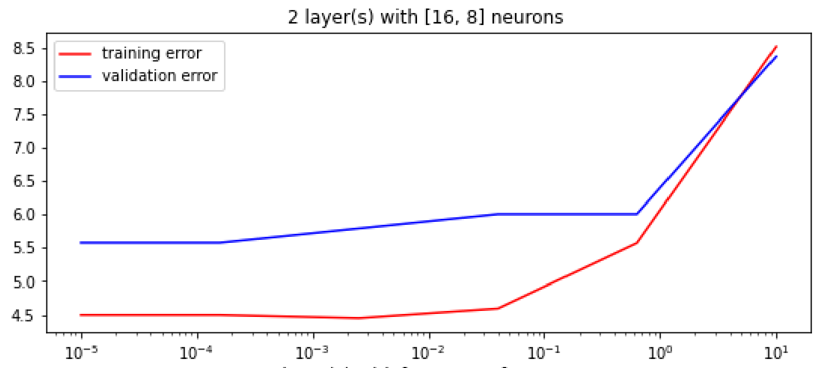
```
#initializations
training_error=np.zeros((4, 6))
validation_error=np.zeros((4, 6))
```

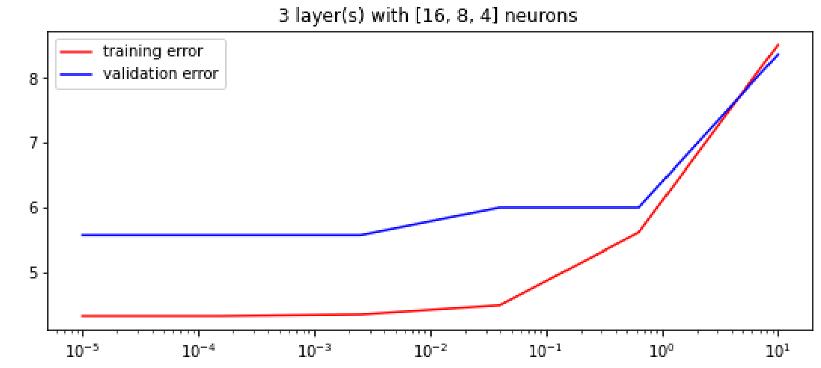
```
layer=[16]
for i in range(4):
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42) #fixed test size of 10%

  for j,a in enumerate(alpha):

        mlp = MLPClassifier( solver='adam',alpha=a, activation = 'logistic',
                                hidden_layer_sizes=(layer[i]),
                                max_iter=500, random_state=1)
        mlp.fit(X_train,y_train)

        training_error[i][j]=log_loss(y_train, mlp.predict(X_train))
        validation_error[i][j]= log_loss(y_test, mlp.predict(X_test))
  layer.append(int(layer[i]/2)) #adding layer of half size for next iteration
  print(layer[i])
```

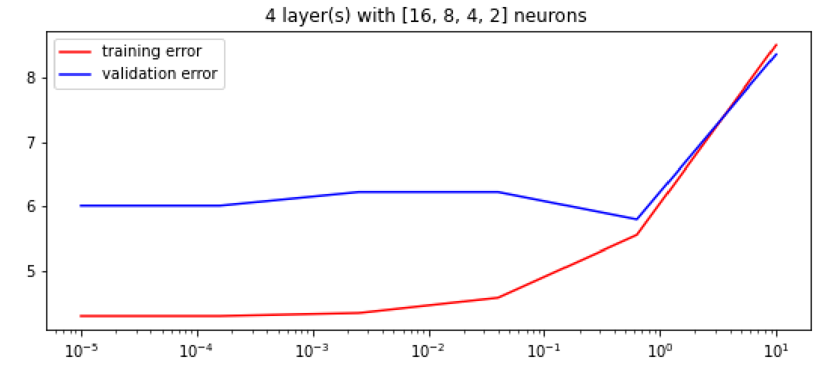
Observing error behaviour as a function of α parameter, for different numbers of layers.

```
plt.rcParams["figure.figsize"] = [20, 8]
fig, axs = plt.subplots(2, 2)
for i,size in enumerate(sizes):
  if i<2:
    axs[0, i].semilogx(alpha, training_error[i,:], '-r')
    axs[0, i].semilogx(alpha, validation_error[i,:], '-b')
    axs[0, i].set_title('{} layer(s) with {} neurons '.format(len(layer[:i+1]), layer[:i+1]))
    axs[0, i].legend(('training error','validation error'),loc="upper left")

  else:
    axs[1, i-2].semilogx(alpha, training_error[i,:], '-r')
    axs[1, i-2].semilogx(alpha, validation_error[i,:], '-b')
    axs[1, i-2].set_title('{} layer(s) with {} neurons '.format(len(layer[:i+1]), layer[:i+1]))
    axs[1, i-2].legend(('training error','validation error'),loc="upper left")
```









By aggregating **more layers** to our MLP, we're increasing the complexity of the model. Since we're facing a high-variance problem that's not helpful at all. We also see that the **penalizing effect becomes stronger** : the **error starts climbing earlier** with respect to α factor as we increase the **depth** of the network.