In [1]:
```
%matplotlib inline
```

SYDE 522 Assignment 3

Backpropagation and Multilayer Perceptrons

Due: Monday Nov 6 at 11:59pm

As with all the assignments in this course, this assignment is structured as a Jupyter Notebook and uses Python. If you do not have Python and Jupyter Notebook installed, the easiest method is to download and install Anaconda https://www.anaconda.com/download (https://www.anaconda.com/download). There is a quick tutorial for running Jupyter Notebook from within Anacoda at https://docs.anaconda.com/free/anaconda/getting-started/hello-world/#python-exercise-jupyter (https://docs.anaconda.com/free/anaconda/getting-started/hello-world/#python-exercise-jupyter) under "Run Python in a Jupyter Notebook"

Implement your assignment directly in the Jupyter notebook and submit your resulting Jupyter Notebook file using Learn.

While you are encouraged to talk about the assignment with your classmates, you must write and submit your own assignment. Directly copying someone else's assignment and changing a few small things here and there does not count as writing your own assignment.

Make sure to label the axes on all of your graphs.

Question 1

a) [1 mark] The following code generates the nested circles dataset that we have used in class before.

```
x, y = sklearn.datasets.make_circles(n_samples=100, shuffle=True, nois
e=0.1, random_state=0, factor=0.3)
```

As before, you can split this into training and test data

```
import sklearn.model_selection
x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_
split(
    x, y, test_size=0.2, shuffle=True, random_state=0
)
```

To classify this data using a multi-layer perceptron trained using backprop, we can use the built-in implementation in `sklearn`.

```
mlp = sklearn.neural_network.MLPRegressor(hidden_layer_sizes=(20,), #
one hidden layer with 20 features
                                          activation='relu',        #
rectified linear
                                          learning_rate_init=1e-2,  #
learning rate
                                          max_iter=1000,            #
number of iterations
                                          early_stopping=True,      #
stop training if validation data gets worse
                                          random_state=0)           #
random number seed for initialization
```

To train the model, use

```
mlp.fit(x_train, y_train)
```

To determine the outputs on your testing data `x_test` , use

```
output = mlp.predict(x_test)
```

b) [1 mark] For the model you trained in part a), plot the output for a grid of inputs between -2 and 2. This can be done using similar code as used in the last assignment:

```
extent = (-2, 2, -2, 2)
G = 200
XX, YY = np.meshgrid(np.linspace(extent[2],extent[3],G), np.linspace(e
xtent[0],extent[1],G))
pts = np.vstack([YY.flatten(), XX.flatten()]).T
output_pts = mlp.predict(pts)
im = plt.imshow(output_pts.reshape((G,G)).T, vmin=0, vmax=1, cmap='RdB
u',
                extent=(extent[0], extent[1], extent[3], extent[2]))
plt.scatter(x[:,0], x[:,1], c=np.where(y==1, 'blue', 'red'))
```

Has the network learned to classify the data well?

c) [1 mark] Repeat part a) but reduce the network size so that there are only 10 features (i.e. 10 neurons in the hidden layer). Report the RMSE and generate the same plot as in part b). Has the network learned to classify the data well?

d) [1 mark] Repeat part a) but for the following different number of features: `[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]` . For each number of features, repeat the process 10 times and compute the average RMSE over those 10 times. Note that you will have to change the `random_state=0` parameter each time, in both the `MLPRegressor` and the `train_test_split` code. For example, if you do this in a for loop `for i in range(10):` then you would set `random_state=i` .

Generate a plot showing how the average RMSE changes as you adjust the number of features.

e) [1 mark] Repeat part d) but add an extra layer of features (i.e. an extra layer inside the network). Do this by setting `hidden_layer_sizes` . In the previous example, we set it t `(20,)` to generate one internal layer of 20 features. To have two internal layers both having 20 features, set it to `(20,20)` . For this question, use the same number of features in both layers (i.e. try it with `(5,5)` , then `(10,10)` , then `(15, 15)` and so on up to `(50,50)`). Generate a plot showing how the average RMSE changes as you change these numbers of features.

How does your result in part (e) compare to your result in part (d)? What does this indicate about how useful the second layer of features is for this task?

f) [1 mark] Repeat part a) and b) but for this dataset:

```
x, y = sklearn.datasets.make_circles(n_samples=100, shuffle=True, nois
e=0.1, random_state=0, factor=0.3)
x[:,0]*=0.1
```

(i.e. exactly the same dataset, but with the `x` values scaled by 0.1)

Report the RMSE and generate the output plot from part b).

Is the accuracy better or worse on this scaled dataset, as compared to the original parts a) and b)? Why?

g) [1 mark] `sklearn` has a tool for automatically rescaling data for you. You can create a scaler as follows:

```
scaler = sklearn.preprocessing.StandardScaler().fit(X_train)
```

and then you can transform the `X_train` and `X_test` with `scaler.transform(X_train)` and `scaler.transform(X_test)` . You can even transform the `pts` used to create the output plot using `scaler.transform(pts)` .

Repeat part f) but use the `scaler` to scale your data before using it. Report the RMSE and generate the output plot from part b). How does this accuracy compart to part f) and to the original part a)?

h) [1 mark] Repeat parts a) and b) with the following dataset:

```
x, y = sklearn.datasets.make_moons(n_samples=500, noise=0.05, random_s
tate=0)
```

Try it both with and without the `scaler` from the part (g), and report the RMSE and generate the output plot both ways. Should you use the `scaler` for this sort of data? Why or why not?

i) [1 mark] Repeat parts a) and b) with the following dataset:

```
x, y = sklearn.datasets.make_blobs(centers=[[-1, -1], [1, 1]],
                                             cluster_std=[1, 1],
                                             random_state=0,
                                             n_samples=200,
                                             n_features=2)
```

Try it both with and without the `scaler` from part(g), and report the RMSE and generate the output plot both ways. Note that you will need to adjust the `extent = (-2, 2, -2, 2)` line so that the output plot covers the same range as the training data (try `(-4, 4, -4, 4)`). Should

Question 2:

a) [1 mark] When using an MLP to do classification, we often don't really care what the exact numerical value of the output is: we just want to classify the input data into a particular category. The `sklearn.neural_network.MLPClassifier` does this for us, training a separate output for each category (one-hot encoding) and then classifying based on which output is largest. (Note: it also uses a slightly different Loss function, where the goal is to minimize classification error, rather than minimizing $\frac{1}{2}(y_{target} - y)^2$).

You can use the `MLPClassifier` with the same parameters as the `MLPRegressor` we used in question 1.

```
mlp = sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(20,), #
one hidden layer with 20 features
                                           activation='relu',        #
rectified linear
                                           learning_rate_init=1e-2,  #
learning rate
                                           max_iter=1000,            #
number of iterations
                                           early_stopping=True,      #
stop training if validation data gets worse
                                           random_state=0)           #
random number seed for initialization
```

Use the MLPClassifier on the `digits` dataset we used in previous assignments. Split it into 80% training and 20% testing.

```
import sklearn.datasets
digits = sklearn.datasets.load_digits()
```

Train the classifier on the training data (using `mlp.fit`) and test it on the test data (using `mlp.predict`).

Report the accuracy of the classifier, which is computed as the proportion of time that the output is the same as the target output:

`np.mean(output == Y_test)`

Also generate and print the *confusion matrix*, which is a matrix showing how often particular digits are mistaken for other digits:

```
confusion = np.zeros((10,10))
for i in range(len(output)):
    confusion[output[i], Y_test[i]] += 1
```

b) [1 mark] Repeat the classification in part a) but for different numbers of features (`[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]`). As with question 1d, repeat the process 10 time for each size, adjusting `random_state` each time. Generate a plot with the average classification accuracy for these different feature sizes.

c) [1 mark] What happens if you set `hidden_layer_sizes=()` ? This should not generate any new features at all. How accurate is the system now? Since there are no new features to learn, what is the MLP doing in this case? (Hint: this is now the same thing as an algorithm we have worked with earlier in the course)

d) [2 marks] Using the following dataset, do the best job you can at building a classifier and testing it.

```
digits = sklearn.datasets.load_digits()
X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_
split(
    digits.data, digits.target, test_size=0.2, shuffle=True, random_st
ate=0,
)
```

You can use any of the supervised learning models from the assignments so far: the perceptron (`sklearn.linear_model.Perceptron`), regression (`sklearn.linear_model.Ridge`), linear SVM (`sklearn.svm.LinearSVC`), kernel-based SVM ('sklearn.svm.SVC'), and the MLPClassifier (`sklearn.neural_network.MLPClassifier`). Make sure to develop your models only using the training data (perhaps split into training and validation), and only once you have chosen your best model should you test it on the testing data. You are trying to get the best accuracy (`np.mean(output == Y_test)`) possible.

Question 3:

[1 mark] Describe what you would like to do for your final project. In particular, tell me what dataset you want to analyse (either one you've made up, or one found online in places like [https://www.kaggle.com/datasets (https://www.kaggle.com/datasets)](https://www.kaggle.com/datasets) or one from the various papers we've discussed in class). Given that dataset, describe what algorithms you want to try on that dataset. You should include both very simple algorithms and more complex ones. Indicate what parameters of those algorithms you would adjust and what you would measure as you are adjusting those parameters.

Even though the final project can be done in groups of 2, each member of the group should write their answer this question separately.