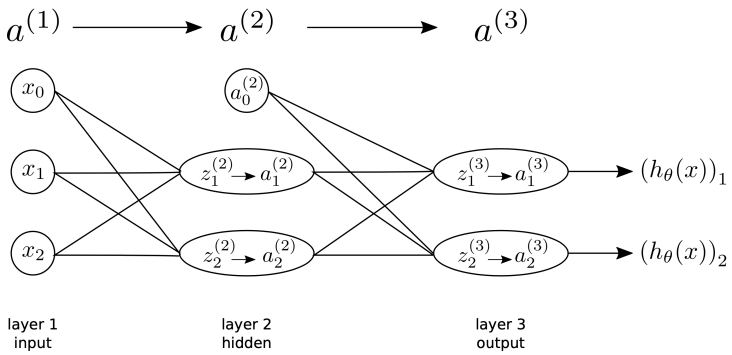
Final Project: Logistic Regression, Neural Network, and Clustering and Applications

Submit your plots, descriptions, python scripts, and presentations to canvas.

1. (25 points) Implement a Python class to execute gradient descent-based logistic regression for classification. Use $\Delta J = 0.00001$ as the stopping criterion. Use setosa and virginica in the iris dataset for writing and debugging your code.

2. (25 points) Implement a Python class to execute forward propagation and back propagation for the following neural network architecture for the purpose of regression and classification:



3. (25 points) Apply the python scripts that you have written for PCA, linear regression, logistic regression, and ANN to the heart disease dataset at the UCI Machine Learning Repository and describe what you could find about the data. Use Jupyter notebook to show both your code and plots. Apply kmeans and hiararchical clustering in sklearn to the data as well.

   The data and the description of the data can be found here: Heart Disease Dataset. Use the `processed.clevand.data` that you can download by clicking the 'DOWNLOAD' button at the top right corner. You can also find the entire downloaded data here: Download the Heart Disease Dataset.

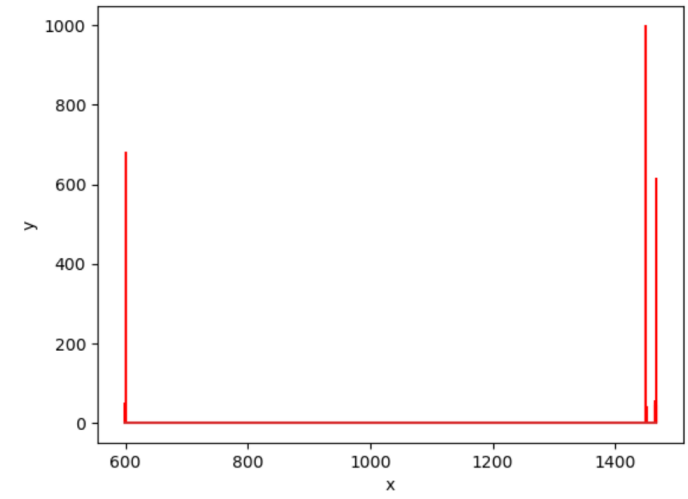
   When you apply logistic regression and ANN, consider the problem as a binary classification problem to distinguish presence from absence of heart disease. When you apply linear regression, look for variables that are strongly correlated. When you apply clustering, do you see certain patients group together. Describe in detail what you find.

4. (25 points) Purpose: apply clustering method to look for mass spectra that are similar.

   There are 50,000 mass spectra in the file `msms.pkl`. The spectra are stored in a dictionary format. Each spectrum has x coordinate and y coordinate. The following example script shows how to extract a spectrum, round the x coordinate, and plot the spectrum. The resulting spectrum is shown as well

```
data_file_name = 'msms.pkl'
file_handle = open(data_file_full_name, 'rb')
data = pickle.load(file_handle)
file_handle.close()

# plot one single spectrum:
fig, ax = plt.subplots()
cur_spectra_1 = data[1]
mz_1 = cur_spectra_1[:, 0]
mz_rounded_1 = np.round(mz_1)
abundance_1 = cur_spectra_1[:, 1]
ax.stem(mz_rounded_1, abundance_1, markerfmt='None', linefmt='red')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```

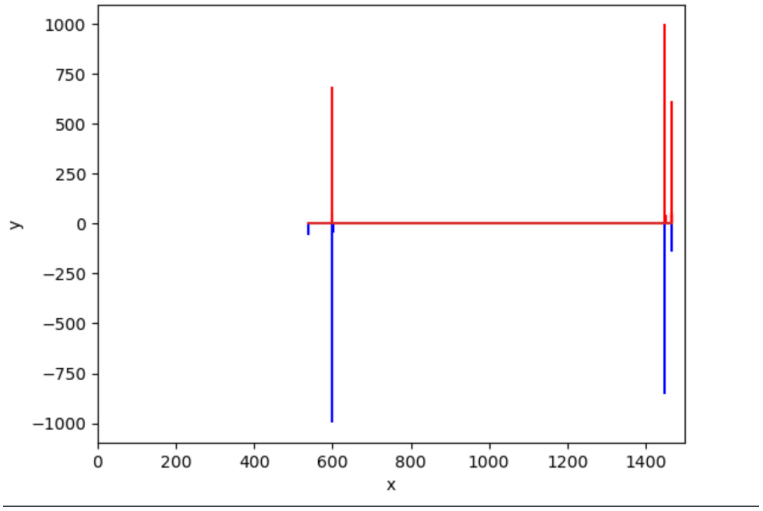


The following example script shows how to plot two spectra together in a head-to-toe format. This way of plotting can easily show how similar two spectra are. The resulting plot is shown as well.

```
# plot two spectra together as a head-to-toe plot to see how similar two spectra are
fig, ax = plt.subplots()
ax.stem(mz_rounded_1, abundance_1, markerfmt='None', linefmt='red')

cur_spectra_2 = data[2]
mz_2 = cur_spectra_2[:, 0]
mz_rounded_2 = np.round(mz_2)
abundance_2 = cur_spectra_2[:, 1]
ax.stem(mz_rounded_2, -abundance_2, markerfmt='None', linefmt='blue')

ax.set_xlim([0, 1500])
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```



What you need to do is to round the x coordinate of each of the 50,000 spectra, applying clustering method to look for similar spectra. Spectra similarity is calculated as the normalized dot product between the abundance values of two spectra. Make sure that the x coordinates match when you calculate the normalized dot product between two spectra. Use different similarity threshold to see what you can find out.