BackPropagation

In order to understand backpropagation, you will implement a backpropagation algorithm for a neural network from scratch. Backpropagation can be used for both classification and regression problems, we will focus on classification in this exercise.

Exe. 10 (Update Weights) Once errors are calculated for each neuron in the network via the back propagation method above, they can be used to update weights. Network weights are updated as follows:

weight = weight + learning_rate * error * input (5)

where `weight` is a given weight, `learning_rate` is a parameter that you must specify, `error` is the error calculated by the backpropagation procedure for the neuron and `input` is the input value that caused the error. The same procedure can be used for updating the bias weight, except there is no input term, or input is the fixed value of 1.0. Learning rate controls how much to change the weight to correct for the error. For example, a value of 0.1 will update the weight 10% of the amount that it possibly could be updated.

Write a function named `update_weights(network, row, l_rate)` that updates the weights for a network given an input row of data, a learning rate and assume that a forward and backward propagation have already been performed. Use the formula given in equation (5) for updating weights using SGD. Notice that the input for the first layer is `row` while the input of the other layers are the outputs of their previous layers. After executing this function, the weights of networks should be updated according to SGD.

Exe. 11 After updating network weights, we should do it repeatedly.The train process involves looping for a fixed number of epochs and within each epoch updating the network for each row in the training dataset. Write a function named `train_network(network, train, l_rate, n_epoch, n_outputs)` for training of an already initialized neural network with a given training dataset, learning rate, fixed number of epochs and an expected number of output values. Complete the missing parts of function:

```
def train_network(network, train, l_rate, n_epoch, n_outputs):
  #n_epoch defines number of iterations for the training process
  for epoch in range(n_epoch):
    sum_error = 0
    for row in train:
      outputs = #call a forward propagate function for getting
    output here
      expected = [0 for i in range(n_outputs)]
      expected[row[-1]] = 1
      sum_error += #compute the mean square error between expected
    and outputs vectors
```

```
      # call the backpropagation function for network and expected vector for computing and adding delta to the network
      #calle uodate_weights for modifying weights in the network
      print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
```

Exe. 12 To check the complete code from the beginning until now, we need a small data set for testing the whole project. Define your data set as a list of 10 rows. Define each row as a list of size 3 with the first two elements for the inputs and the third element for the label. Select the input values from as a float random values between 0 and 10 and choose the labels randomly from $\{0, 1\}$. You should have something similar to: $[[1.25, 6.548, 0], [5.6983, 3.58, 1], \cdots]$
Initialize a network for your small dataset and train your network with your preferred parameters. Check the result by printing the network layers after the training.

Exe. 13 (Predict) Making predictions with a trained neural network is easy enough. We have already seen how to forward-propagate an input pattern to get an output. For writing a *predict(network, row)* function, it is just enough to implement `forward propagate()` on the trained network with its modified weights and the given input. Since this network returns back the probability assigned to each binary output, it is better to turn the index in the network output that has the largest probability. Write the function by yourself.

Exe. 14 To test the `predict()` function, after training the network, predict the labels of the same dataset used for the train set. Check the results; normally, the predicted and real labels should be the same.

Exe. 15 Download the `seeds_dataset.csv` from the BrightSpace/Week2. If you use GoogleColab you have two option either upload it directly from your computer or upload it in your google drive and use it. For the first option you should write:

```
from google.colab import files
uploaded = files.upload()
```

Exe. 16 Write a function **load_csv(filename)** receiving the file name and returning the data set.

Exe. 17 Shuffle the dataset, normalize all columns except the labels (final column) w.r.t maximum and minimum value of each column. Take 80% and 20% to train and test set respectively.

Exe. 18 Initialize a network regarding the dataset and labels. Use your project code for training on the train set.

Exe. 19 Predict the test set on trained network. Compute the mean square error.