Deep Learning Lab Internal-1

1. Demonstrate normalization of input data, basic activation functions such as the softmax, sigmoid, dsigmoid, etc.

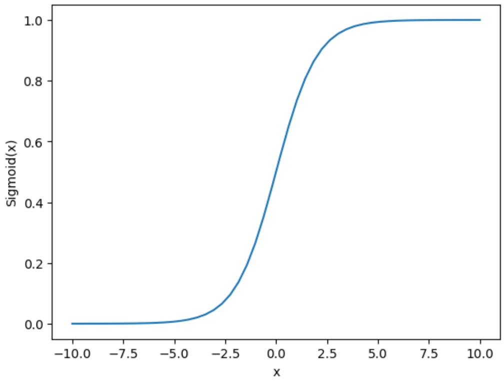
 # Import Packages

```
import numpy as np
import matplotlib.pyplot as plt
```

Activation Functions

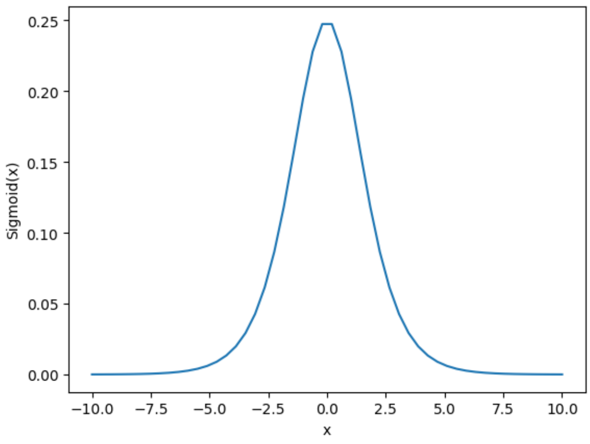
 # sigmoid

```
def sigmoid(x):
 return 1/(1 + np.exp(-x))

x = np.linspace(-10, 10, 50)
p = sigmoid(x)
plt.xlabel("x")
plt.ylabel("Sigmoid(x)")
plt.plot(x, p)
plt.show()
```

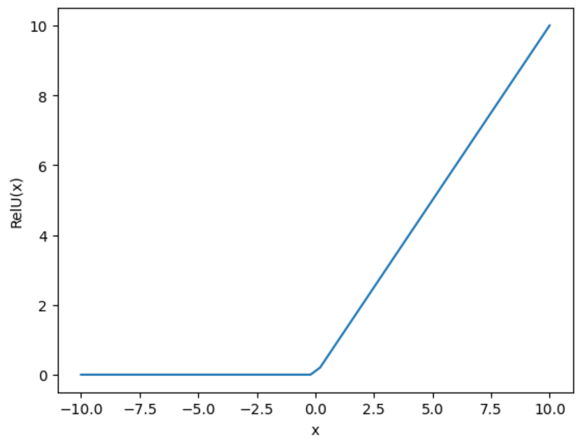


 # der_sigmoid

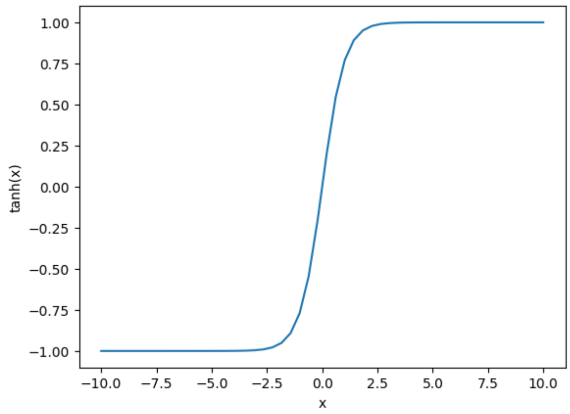
```
def der_sigmoid(x):
  return sigmoid(x) * (1- sigmoid(x))

x = np.linspace(-10, 10, 50)
p = der_sigmoid(x)
plt.xlabel("x")
plt.ylabel("Sigmoid(x)")
plt.plot(x, p)
plt.show()
```




```
# relu
def relu(x):
  return np.maximum(0, x)

x = np.linspace(-10, 10, 50)
p = relu(x)
plt.xlabel("x")
plt.ylabel("RelU(x)")
plt.plot(x, p)
plt.show()
```

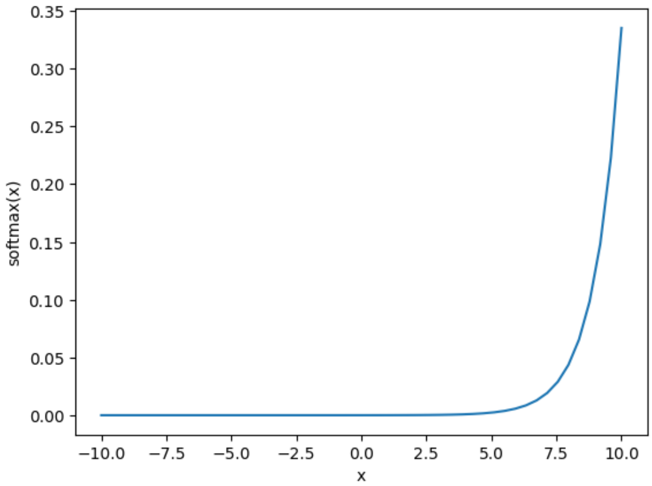



```
# tanh
def tanh(x):
    return np.tanh(x)

x = np.linspace(-10, 10, 50)
p = tanh(x)
plt.xlabel("x")
plt.ylabel("tanh(x)")
plt.plot(x, p)
plt.show()
```



 # softmax

```
def softmax(x):
    exp_x = np.exp(x)
    return exp_x / np.sum(exp_x)

x = np.linspace(-10, 10, 50)
p = softmax(x)
plt.xlabel("x")
plt.ylabel("softmax(x)")
plt.plot(x, p)
plt.show()
```



2. Build a neural network for logistic regression to minimize the cost function and update the parameters.

```
from math import exp
# Make a Log.Reg with coefficients
def predict(row, coefficients):
    yhat = coefficients[0]
    for i in range(len(row)-1):
        yhat += coefficients[i + 1] * row[i]
    return 1.0 / (1.0 + exp(-yhat))

# Estimate logistic regression coefficients gradient descent
def coefficients_sgd(train, l_rate, n_epoch):
    coef = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            yhat = predict(row, coef)
            error = row[-1] - yhat
            sum_error += error**2
            coef[0] = coef[0] + l_rate * error * yhat * (1.0 - yhat)
            for i in range(len(row)-1):
                coef[i+1]=coef[i+1]+l_rate *error*yhat*(1.0 - yhat)*row[i]
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return coef

# Calculate coefficients
dataset = [[2.7810836,2.550537003,0], [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0], [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0], [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1], [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1], [7.673756466,3.508563011,1]]

l_rate = 0.3
n_epoch = 100
coef = coefficients_sgd(dataset, l_rate, n_epoch)
print(coef)
```

Output

The updated parameters are:
[-0.8596443546618897, 1.5223825112460005, -2.218700210565016]

3. Implement backward propagation neural network for a two- class classification with a single hidden layer, non-linear activation function like tanh and compute the cross-entropy loss.

```
# Import Packages
from math import exp
from random import seed
from random import random

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
	network = list()
	hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
	network.append(hidden_layer)
	output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
	network.append(output_layer)
	return network

# Calculate neuron activation for an input
def activate(weights, inputs):
	activation = weights[-1]
	for i in range(len(weights)-1):
		activation += weights[i] * inputs[i]
	return activation

# Transfer neuron activation
def transfer(activation):
	return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
	inputs = row
	for layer in network:
		new_inputs = []
		for neuron in layer:
			activation = activate(neuron['weights'], inputs)
			neuron['output'] = transfer(activation)
			new_inputs.append(neuron['output'])
		inputs = new_inputs
	return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
	return output * (1.0 - output)
```

 # Backpropagate error and store in neurons

```
def backward_propagate_error(network, expected):
	for i in reversed(range(len(network))):
		layer = network[i]
		errors = list()
		if i != len(network)-1:
			for j in range(len(layer)):
				error = 0.0
				for neuron in network[i + 1]:
					error += (neuron['weights'][j] * neuron['delta'])
				errors.append(error)
		else:
			for j in range(len(layer)):
				neuron = layer[j]
				errors.append(neuron['output'] - expected[j])
		for j in range(len(layer)):
			neuron = layer[j]
			neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

 # Update network weights with error

```
def update_weights(network, row, l_rate):
	for i in range(len(network)):
		inputs = row[:-1]
		if i != 0:
			inputs = [neuron['output'] for neuron in network[i - 1]]
		for neuron in network[i]:
			for j in range(len(inputs)):
				neuron['weights'][j] -= l_rate * neuron['delta'] * inputs[j]
			neuron['weights'][-1] -= l_rate * neuron['delta']
```

 # Train a network for a fixed number of epochs

```
def train_network(network, train, l_rate, n_epoch, n_outputs):
	for epoch in range(n_epoch):
		sum_error = 0
		for row in train:
			outputs = forward_propagate(network, row)
			expected = [0 for i in range(n_outputs)]
			expected[row[-1]] = 1
	sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
			backward_propagate_error(network, expected)
			update_weights(network, row, l_rate)
		print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
```

```
# Test training backprop algorithm
seed(1)
dataset = [[2.7810836,2.550537003,0],[1.465489372,2.362125076,0],
	[3.396561688,4.400293529,0],[1.38807019,1.850220317,0],
	[3.06407232,3.005305973,0],[7.627531214,2.759262235,1],
	[5.332441248,2.088626775,1],[6.922596716,1.77106367,1],
	[8.675418651,-0.242068655,1],[7.673756466,3.508563011,1]]


n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
	print(layer)
```

Output

```
[{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297],
'output': 0.029980305604426185, 'delta': 0.0059546604162323625},
{'weights': [0.37711098142462157, -0.0625909894552989, 0.2765123702642716],
'output': 0.9456229000211323, 'delta': -0.0026279652850863837}]

[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275],
'output': 0.23648794202357587, 'delta': 0.04270059278364587},
{'weights': [-2.5584149848484263, 1.0036422106209202, 0.42383086467582715],
 'output': 0.7790535202438367, 'delta': -0.03803132596437354}]
```

4. **Build a deep neural network with more than one hidden layer, non-linear functions like ReLU.**

```
# Import Packages
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the dataset
dataset = loadtxt('diabates.csv', delimiter=',')

# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]

# define the keras model
model = Sequential()
model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)

# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Output

```
24/24 [==============================] - 0s 1ms/step - loss: 0.4896 -
accuracy: 0.7682
Accuracy: 76.82
```

5. Build deep neural network to any classification problem and compare its accuracy to logistic regression.

Regression

```
# Import Libraries
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/abalone.csv'
dataframe = read_csv(url, header=None)
dataset = dataframe.values

# split into input (X) and output (y) variables
X, y = dataset[:, 1:-1], dataset[:, -1]
X, y = X.astype('float'), y.astype('float')
n_features = X.shape[1]

# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)

# define the keras model
model = Sequential()
model.add(Dense(20, input_dim=n_features, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='linear'))

# compile the keras model
model.compile(loss='mse', optimizer='adam')

# fit the keras model on the dataset
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=2)

# evaluate on test set
yhat = model.predict(X_test)
error = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % error)
```

Output

```
44/44 [==============================] - 0s 2ms/step
MAE: 1.502
```

Classification

 # Import Libraries

```
from numpy import unique
from numpy import argmax
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

 # load dataset

```
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/abalone.csv'
dataframe = read_csv(url, header=None)
dataset = dataframe.values
```

 # split into input (X) and output (y) variables

```
X, y = dataset[:, 1:-1], dataset[:, -1]
X, y = X.astype('float'), y.astype('float')
n_features = X.shape[1]
```

 # encode strings to integer

```
y = LabelEncoder().fit_transform(y)
n_class = len(unique(y))
```

 # split data into train and test sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
```

 # define the keras model

```
model = Sequential()
model.add(Dense(20, input_dim=n_features, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(n_class, activation='softmax'))
```

 # compile the keras model

```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
```

 # fit the keras model on the dataset

```
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=2)
```

 # evaluate on test set

```
yhat = model.predict(X_test)
yhat = argmax(yhat, axis=-1).astype('int')
acc = accuracy_score(y_test, yhat)
print('Accuracy: %.3f' % acc)
```

Output

```
44/44 [==============================] - 0s 3ms/step
Accuracy: 0.280
```