zl9901 Homework3

Question 1

The cost function of Cross entropy is

$$Loss = -\frac{1}{N}\sum_i y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)$$

In np_nn.py, we can find this in the get_cost_value method:

```
cost = -1 / m * (np.dot(Y, np.log(Y_hat).T) + np.dot(1 - Y, np.log(1 - Y_hat).T))
```

We can find first derivative in the main backpropagation method, because of question 2 and question 3, I've decided to write this first derivative function in my_nn_1.py-my_nn_4.py and use this function as a parameter in **train** method.

```
def dfCrossEntropy(y,yhat):
    return -(np.divide(y,yhat)-np.divide(1-y,1-yhat))
```

Using MSE cost function and its derivative for this assignment

I choose to comment cross entropy cost function and first derivative of cross entropy out and use MSE cost function and its derivative instead for this assignment, because cross entropy may have vanishing gradient and interrupt the overall training process.

The cost function of Mean Square Error is

$$Loss = \frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

In np_nn.py document, we can find this in the get_cost_value method:

```
cost = 1 / m * np.sum((np.subtract(Y, Y_hat)) ** 2)
```

We can find first derivative in the main backpropagation method, because of question 2 and question 3, I've decided to write this first derivative function in my_nn_1.py-my_nn_4.py and use the function as a parameter in **train** method.

```
def dfMSE(y,yhat):
    return -2*np.subtract(y,yhat)
```

Question 2

I have added support for users to pass their own cost function to the trainer. I have modified both train() and OneStep() methods in my_nn_1.py-my_nn_4.py

Please see the details below

```
params_values,cost_history,accuracy_history = train(Xs,Ys,nn_architecture,4000,learning_rate,dfMSE)
onestep = OneStep(Xs, Ys, params_values, cost_history, accuracy_history, nn_architecture,dfMSE)
onestep()
```

Question 3

I have defined two functions which return first derivative of the cross entropy cost function and first derivative of MSE cost function in my_nn_1.py-my_nn_4.py. Because we can only use one function for this assignment, I choose to comment CrossEntropy out.

```
def dfCrossEntropy(y,yhat):
    return -(np.divide(y,yhat)-np.divide(1-y,1-yhat))

def dfMSE(y,yhat):
    return -2*np.subtract(y,yhat)
```

Question 4

In full_backward_propagation and full_forward_propagation methods, we need to make some changes. I choose to use a python dictionary to store index of maximal value for each feature and use this dictionary as a parameter which can be stored in dictionary named meory. When we do backward propagation, we can get these indices back from memory dictionary add use zero padding operation.

Question 5

Users can feel free to change all the parameters in nn_architecture

For instance, stride parameter is related to its previous and next layer. In my program, if stride is changed, user may also need to update the input_dim and output_dim. Stride can be any number if the user wants to modify it. (but need to be reasonable, e.g can't be zero)

Because the gradient algorithm sometimes can generate random results.

Especially for dataset 4 , the distribution is irregular. If the animation

result is not very well, please try one more time until the result is stable.

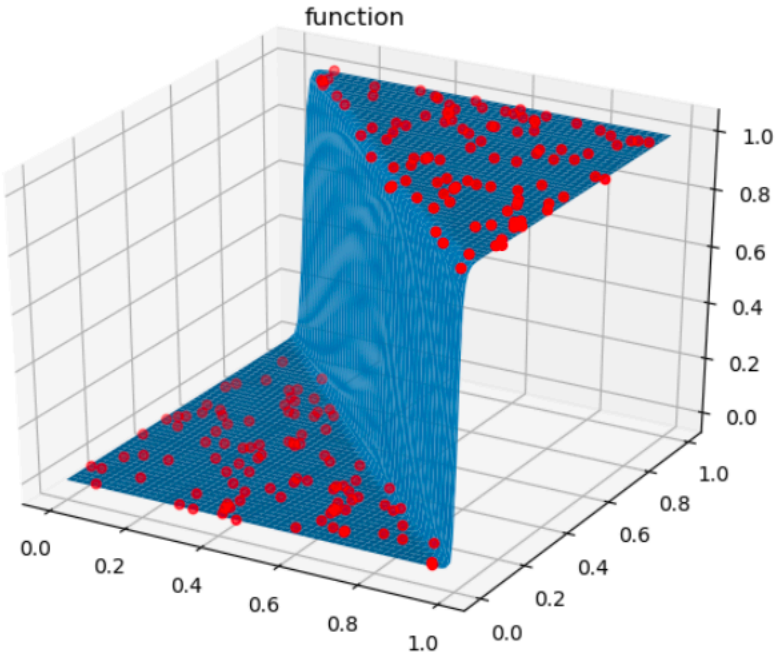

Figure 1. Dataset1

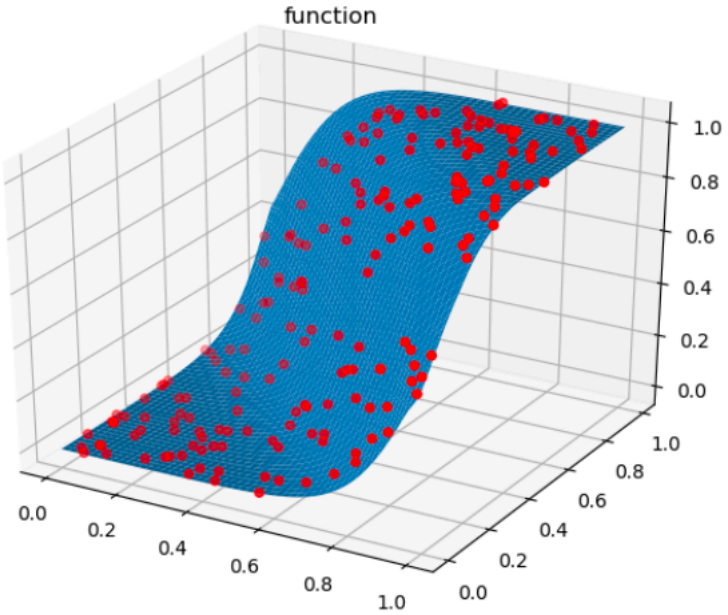

Figure 2. Dataset2

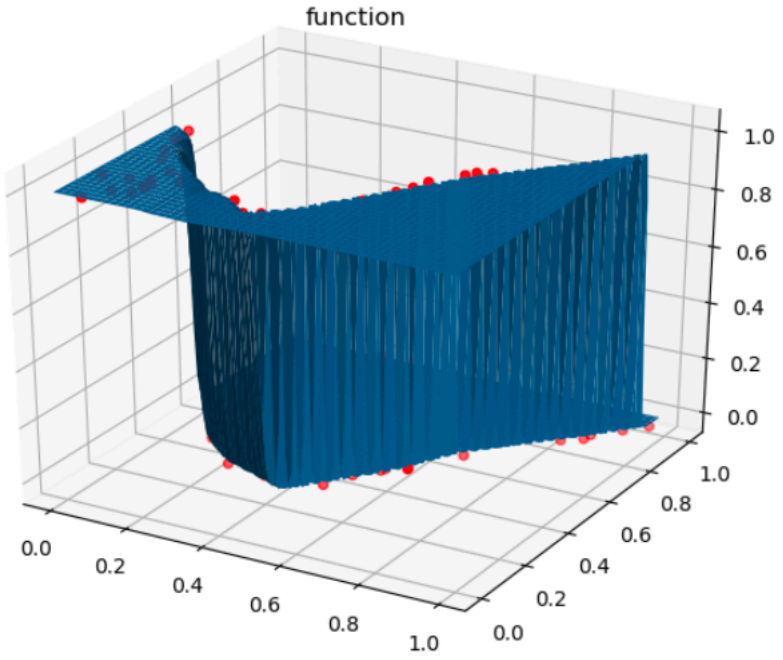

Figure 3. Dataset3

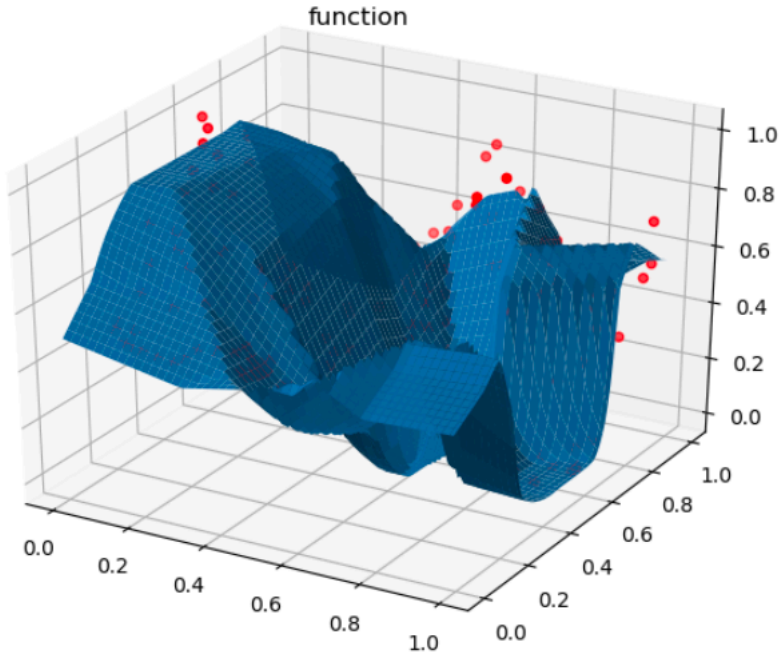

Figure 4. Dataset4