Assignment 2

1 Neural Network using Numpy

1.1 Helper Functions

1. ReLU()

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

Figure 1. Python code for the relu() function

2. softmax()

```python
def softmax(x):
    return np.exp(x)/np.sum(np.exp(x), axis=1, keepdims=True)
```

Figure 2. Python code for the softmax() function

3. compute()

```python
def compute(x, W, b):
    return np.matmul(x, W) + b
```

Figure 3. Python code for the compute() function

4. averageCE()

```python
def averageCE(target, prediction):
    return -1*np.mean(target*np.log(prediction))
```

Figure 4. Python code for the averageCE() function

5. gradCE()

$$L = -\sum y_i log p_i$$

$$\sigma(z)_j = p_i = \frac{e^{z_j}}{\sum e^{z_k}}$$

We derive the softmax function using the quotient rule and get:

If i=j:

$$\frac{d\sigma}{dz_j} = p_i(1 - p_j)$$

else:

$$\frac{d\sigma}{dz_j} = -p_j p_i$$

Using the chain rule we then derive the loss function with respect to the input of the softmax function:

$$\frac{dL}{do_i} = \frac{dL}{dp_k}\frac{dp_k}{do_i} = -\sum y_k \frac{dlog(p_k)}{dp_k}\frac{dp_k}{do_i} = -\sum y_k \frac{1}{p_k}(p_k(1-p_j) - p_j p_k) = -y_i(1-p_i) + \sum y_k p_i = p_i(y_i + \sum y_k) - y_i$$

y is a one hot encoded vector, therefore the expression in the brackets evaluates to 1 and we will end up with:

$$\frac{dL}{do_i} = p_i - y_i$$

```
def gradCE(target, prediction):
    return prediction-target
```

Figure 5. Python code for the gradCE() function

1.2 Back Propagation Derivation

Dimensions: x: (Nx784) W_o: (Hx10) b_o: (Nx10) h:(NxH)
p: (Nx10) W_h:(784xH) b_h:(NxH)

1. $\frac{dL}{dW_o}$

Using chain rule we have:

$$\frac{dL}{dW_o} = \frac{dL}{dp_k}\frac{dp_k}{do_i}\frac{do_i}{dW_o} = (p_i - y_i)\frac{do_i}{dW_o} = h^T(p_i - y_i)$$

Shape Justification = $(NxH)^T(Nx10) = (Hx10)$

2. $\frac{dL}{db_o}$

Using chain rule we have:

$$\frac{dL}{db_o} = \frac{dL}{dp_k}\frac{dp_k}{do_i}\frac{do_i}{db_o} = (p_i - y_i)\frac{do_i}{db_o} = 1^T(p_i - y_i)$$

Shape Justification = $(Nx1)^T(Nx10) = (1x10)$

3. $\frac{dL}{dW_h}$

Using chain rule we have:

$$\frac{dL}{dW_h} = \frac{dL}{dp_k}\frac{dp_k}{do_i}\frac{do_i}{dh}\frac{dh}{dW_h} = (p_i - y_i)W_o^T\frac{dh}{dW_h}$$

If h>0:

$$\frac{dL}{dW_h} = x^T(p_i - y_i)W_o^T$$

Shape Justification = $(Nx784)^T$*(Nx10)* $(Hx10)^T$ = *(784xH)*

else:

$$\frac{dL}{dW_h} = 0$$

4. $\frac{dL}{db_h}$

Using chain rule we have:

$$\frac{dL}{db_h} = \frac{dL}{dp_k}\frac{dp_k}{do_i}\frac{do_i}{dh}\frac{dh}{db_h} = (p_i - y_i)W_o^T\frac{dh}{db_h}$$

If h>0:

$$\frac{dL}{dW_h} = 1^T(p_i - y_i)W_o^T$$

Shape Justification = $(Nx1)^T$*(Nx10)* $(Hx10)^T$ = *(1xH)*

else:

$$\frac{dL}{dW_h} = 0$$

```python
def dL_dW_o(target, prediction, h):
    return np.matmul(np.transpose(h), gradCE(target, prediction))

def dL_db_o(target, prediction):
    return np.matmul(np.ones((1, target.shape[0])), gradCE(target, prediction))

def dL_dW_h(target, prediction, W_o, x, W_h, b_h):
    hidden_in = compute(x, W_h, b_h)
    d_r = gradRelu(hidden_in)
    reLU = relu(compute(x, W_h, b_h))
    return np.matmul(np.transpose(x), d_r*np.matmul(gradCE(target, prediction), np.transpose(W_o)))

def dL_db_h(target, prediction, W_o, x, W_h, b_h):
    hidden_in = compute(x, W_h, b_h)
    gradRelu(hidden_in)
    d_r = relu(hidden_in)
    return np.matmul(np.ones((1, hidden_in.shape[0])), d_r*np.matmul(gradCE(target, prediction), np.transpose(W_o)))
```

Figure 6. Python code for all the gradient function that will be used in back propagation

1.3 Learning

The Neural Network we trained through 200 epochs showed high accuracies of approximately 87% across all datasets (Table 1.). The loss graph shows convergence after about 75 epochs (Figure 7.) and final loss value of about 0.05 when using γ=0.99. More specifically, the validation dataset had loss of 0.054 and 87% accuracy. The convergence was drastically slower (after 175 epochs) when using γ=0.9. The loss was also higher (about 0.08) and accuracy lower (about 85%).

	Training Data	Valid Data	Test Data
Loss	0.05092	0.05357	0.05231
Accuracy	87.82%	87%	87.812%

Table 1. Final loss and accuracy values across all datasets

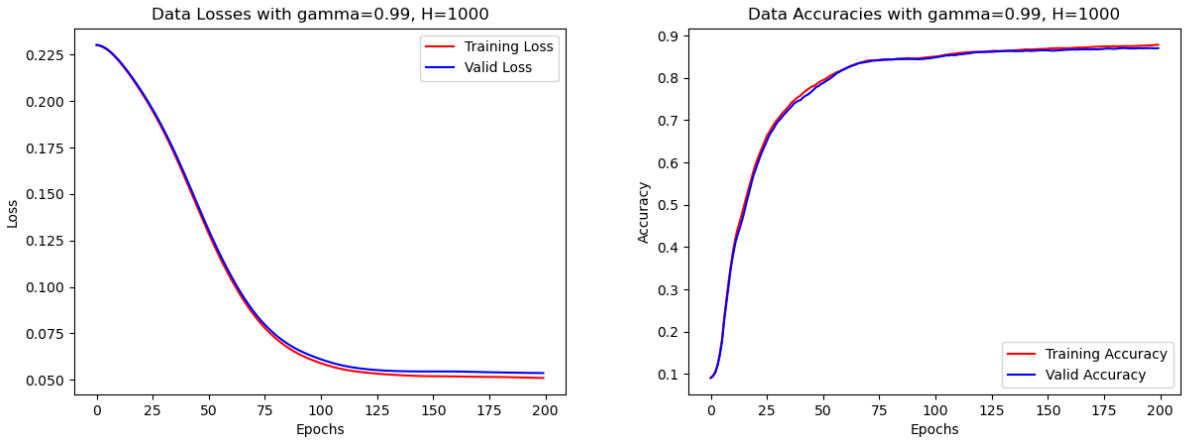

Figure 7. Plots showing loss and accuracy across the training and valid data