Μηχανική Μάθηση Logistic Regression

Ονοματεπώνυμο	A.M
Ηλίας-Θεοφάνης Γραββάνης	3200248

Εργαλεία

Σαν γλώσσα προγραμματισμού χρησιμοποιήθηκε η python 3.10.5 version and pip package manager
Για development χρησιμοποιήθηκε ώς IDE το VsCode

Report

Επιλύθηκαν τα θέμα Α,Β,Γ

δ.

Η main.py περιέχει την Logistic Regression που είναι το μοντέλο μηχανικής μάθησης που χρησιμοποιούμαι για το πρόβλημα πρόβλεψης εικόνας 5,6.
Στην συνάρτηση **ComputeLogisticRegression** : 'Οπως θα δούμε έχουμε τα theta που είναι αυτά που θα μας δώσουν την τιμή μεταξύ του 0 και 1 και με βάση αυτό θα μπορέσουμε να απαντήσουμε αν είναι 5 ή 6.
΄Εχουμε ορίσει παράμετρο total_iter με τιμή 1800 , που θα είναι οι φορές που θα ανατρέξει το μοντέλο και το J_train και J_test μας δίνει το σφάλμα μετά απο κάθε επανάληψη ,δηλαδή πόσο κοντά είναι να προβλέψη σωστά τα δεδομένα τόσο στα train όσο και στα validation.
Η συνάρτηση **ComputeCostGrad** ,η οποία υπολογίζει με τις δεδομένες τιμές του theta και με τα χ και y δεδομένα εκπαίδευση πόσο σφάλμα εκπαίδευσης έχει curl_j και να παρατηρήσω πως επαναληπτικά μειώνεται. Το grad μας δείχνει πως διορθώνει στην επόμενη επανάληψη το theta, δηλαδή με τι στάθμιση να προσθέσω ή να αφαιρέσω απο το theta ώστε να γίνει καλύτερη η πρόβλεψη.
Αυτη γυρναέι το train_error που χρησιμοποιείται στο theta και βελτιώνει το theta πολλαπλασιάζοντας με ένα σταθερό αριθμό το alpha. Επίσης κρατάμε το σφάλμα και το κάνουμε append σε ένα array. Τέλος κάνουμε plot τα test_error,train_error συναρτήσει itterations.
Για τον υπολογισμό του prediction της πρόβλεψης χρησιμοποιείται η συνάρτηση sigmoid που επιστρέφει τιμές μεταξύ 0 έως 1 decimals και αν βρίσκεται μεταξύ 0~0.5 θεωρούμαι οτι είναι 0 (εικόνα 5) και αν είναι απο 0.5~1 αντιστοιχεί στο 1 (εικόνα 6). Για να υπολογίσει τέλος το accuracy πέρνει τα prediction και τα y labels και υπολογίζει το ποσοστό.

Μέρη του κώδικα για την επίλυση

```python
def ComputeLogisticRegression(X, y, X_val, y_val, _lambda=0.0, tot_iter=1800, alpha=0.1):

    theta = np.zeros(X.shape[1]).reshape((-1,1))
    m, n = X.shape

    J_train = []
    J_test = []

    for i in range( tot_iter ):

        train_error, train_grad = ComputeCostGrad(X, y, theta, _lambda)
        test_error, _ = ComputeCostGrad(X_val, y_val, theta, _lambda)

        #update parameters by adding gradient values (ascent)
        theta = theta + (alpha * train_grad/m)

        # print test error to validate the algorithm converges
        # print( test_error[0])

        #store current cost
        J_train.append(train_error[0])
        J_test.append(test_error[0])

    return J_train, J_test, theta
```

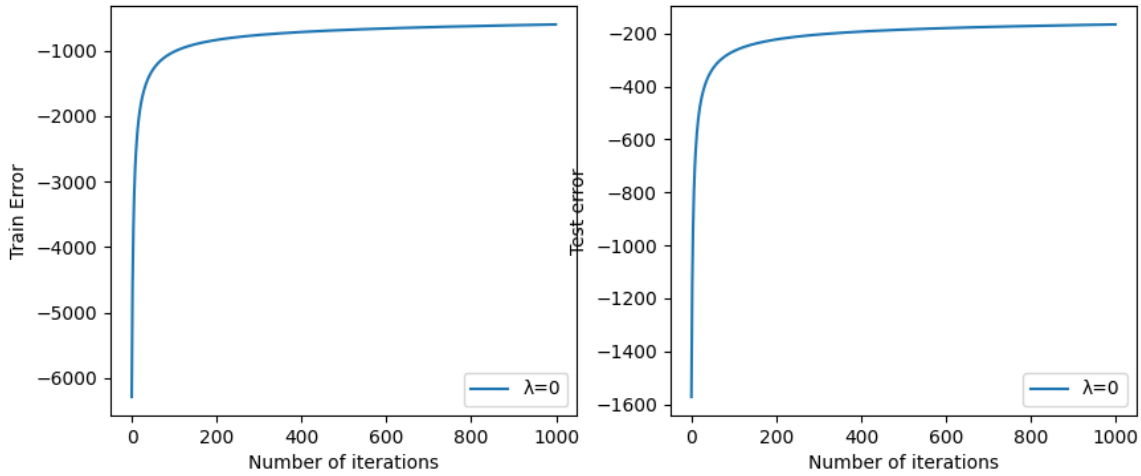
```python
def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))

def ComputeCostGrad(X, y, theta, _lambda):

    h = sigmoid(np.dot(X,theta))

    reg = (_lambda / (2.0 ) ) * np.sum(theta**2)#reguralization
    cur_j = (np.dot(y.T,np.log(h))+np.dot((1-y).T,np.log(1-h))) - reg

    reg = _lambda * theta# /(m+0.0)
    grad = np.dot(X.T,(y-h)) - reg

    return cur_j, grad
```

```python
J_train, J_test, theta = ComputeLogisticRegression(x_train, y_train, x_val, y_val, _lambda=0.0 )

plt.figure(figsize=(10,4))
plt.subplot( 1, 2, 1 )
plt.plot(np.arange( len(J_train) ), J_train, label='λ=0')
plt.xlabel('Number of iterations')
plt.ylabel('Train Error')
plt.legend()
plt.subplot( 1,2,2)
plt.plot( np.arange( len(J_test) ), J_test, label='λ=0')
plt.xlabel('Number of iterations' )
plt.ylabel('Test error')
plt.legend()
plt.show()
p_train, prob_train = predict( theta, x_train )
p_test, prob_test = predict( theta, x_test )
print( 'Accuracy of training set', np.mean( p_train.astype('int') == y_train ) )
print( 'Accuracy of testing set', np.mean( p_test.astype('int') == y_test ) )
```

Διαγράμματα plot - Αποτελέσματα

- Accuracy

- Train/Test error per Itterations - Lambda=0



ε.

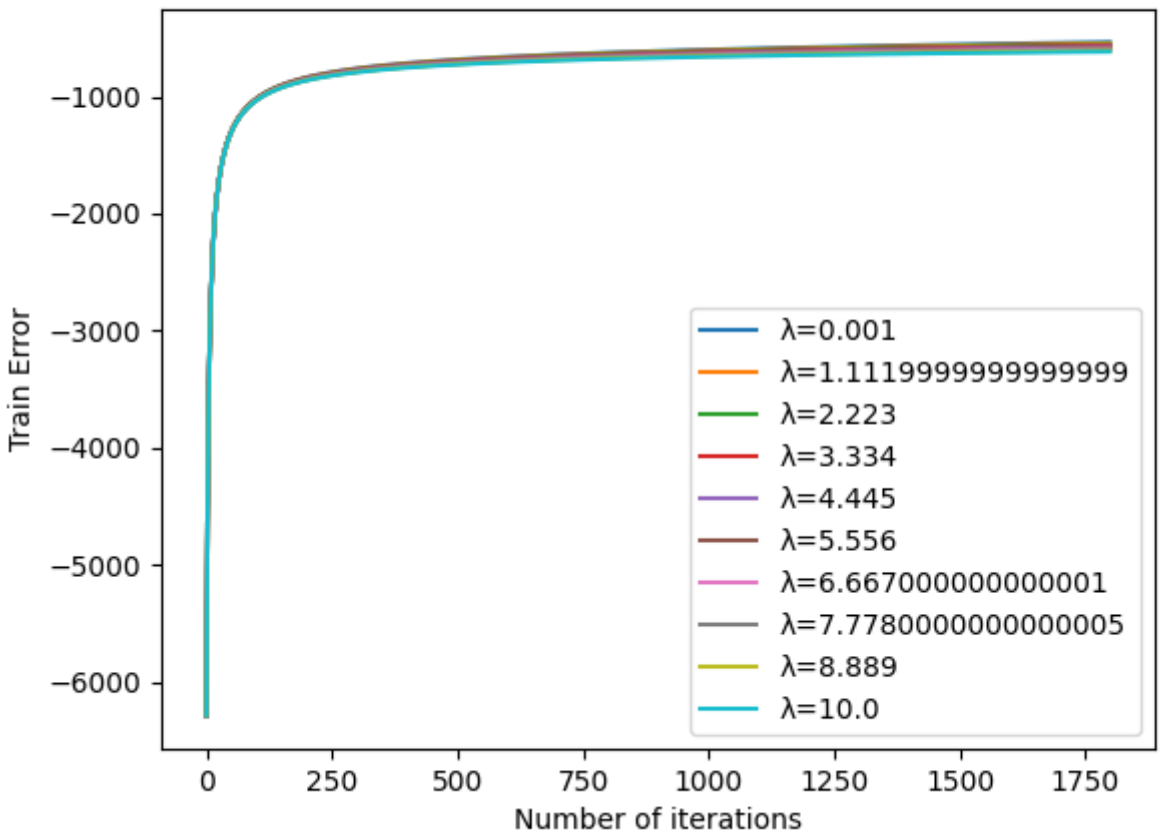
Αλλάζωντας τώρα την παράμετρο λάμδα και κάνοντας l2 regularization θα παρατηρήσουμε ότι για 100 διαφορέτικές τιμές η διαφόρά του accuracy είναι μικρή ,δηλαδή ανταποκρίνεται καλά και χωρίς το λ.Το accuracy οπως βλέπουμε επιρέαζεται πολύ λίγο στις διάφορες τιμές του λάμδα . Για λόγους διαγραμμάτων προστέθηκαν οι 10 απο τις 100 .Εχουν γινει έλεγχοι και για διάφορες 100 τιμές μεταξύ 0.001 και 10.

Μέρη του κώδικα για την επίλυση

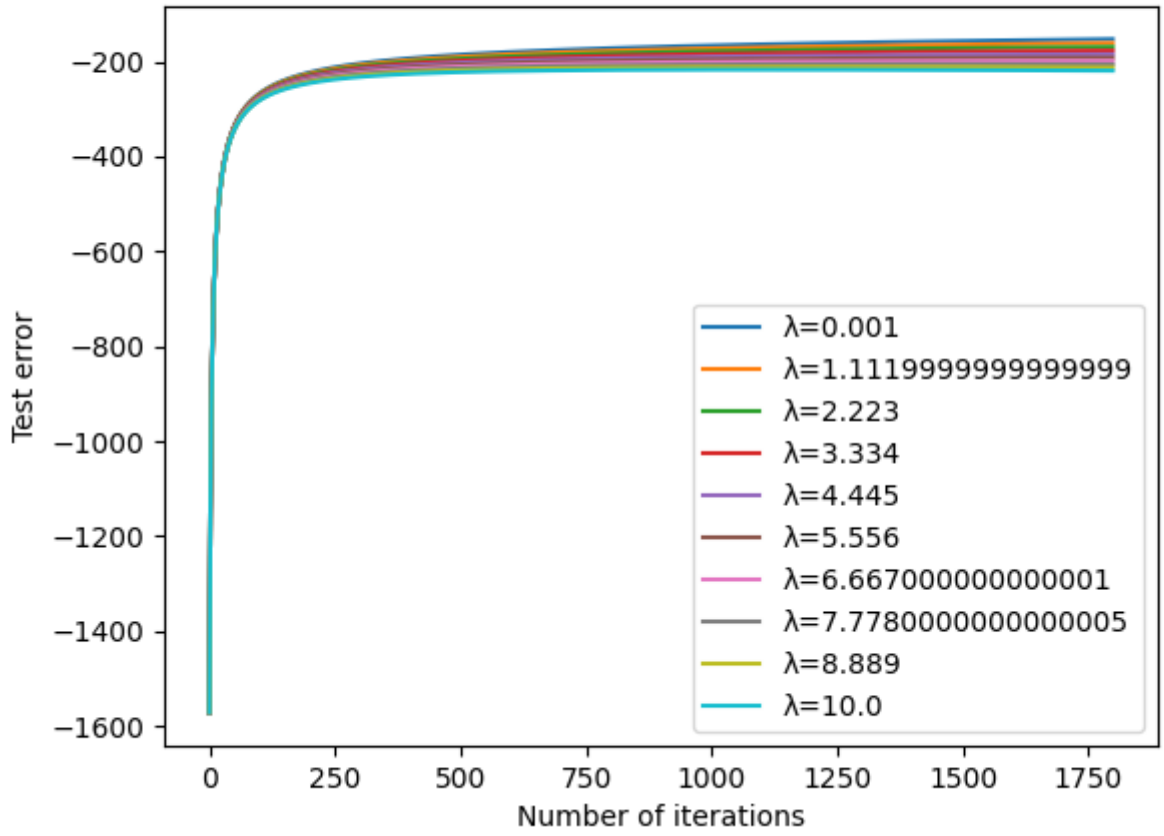
```
print("After regularization")
lambdas = np.linspace(0.001, 10, 10)
train_stats, test_stats = [], []
train_scores =[]
test_scores = []

for value in lambdas:
    J_train, J_test, theta = ComputeLogisticRegression( x_train, y_train, x_val, y_val, _lambda=value )
    train_stats.append( {'history' : J_train, 'lambda': value} )
    test_stats.append( {'history' : J_test, 'lambda': value} )

    p_train, prob_train = predict( theta, x_train )
    p_test, prob_test = predict( theta, x_test )
    train_scores.append(np.mean( p_train.astype('int') == y_train ))
    test_scores.append(np.mean( p_test.astype('int') == y_test ))

# Plot the error of train, test per lambda value
for i in range( len(train_stats) ):
    plt.plot( np.arange( len(train_stats[i]['history']) ), train_stats[i]['history'], label='λ=' + str( train_stats[i]['lambda'] ) )
plt.xlabel( 'Number of iterations' )
plt.ylabel( 'Train Error' )
plt.legend()
plt.show()

for i in range( len(test_stats) ):
    plt.plot( np.arange( len(test_stats[i]['history']) ), test_stats[i]['history'], label='λ='+ str( test_stats[i]['lambda'] ) )
plt.xlabel( 'Number of iterations' )
plt.ylabel( 'Test error' )
plt.legend()
plt.show()


plt.plot(lambdas,train_scores)
plt.xlabel('Lambda')
plt.ylabel('Train Accuracy')
plt.show()

plt.plot(lambdas,test_scores)
plt.xlabel('Lambda')
plt.ylabel('Test Accuracy')
plt.show()
```

Διαγράμματα plot ανα Lambda

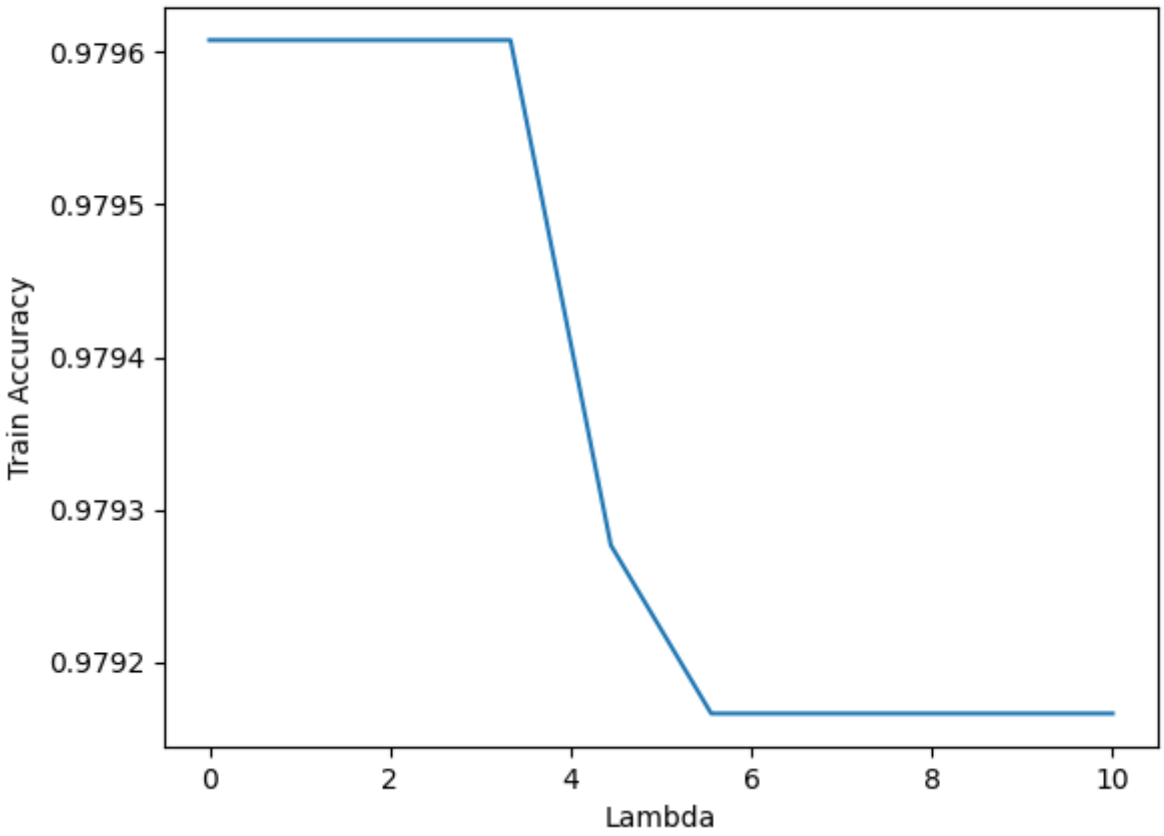
- Train error per Lambda



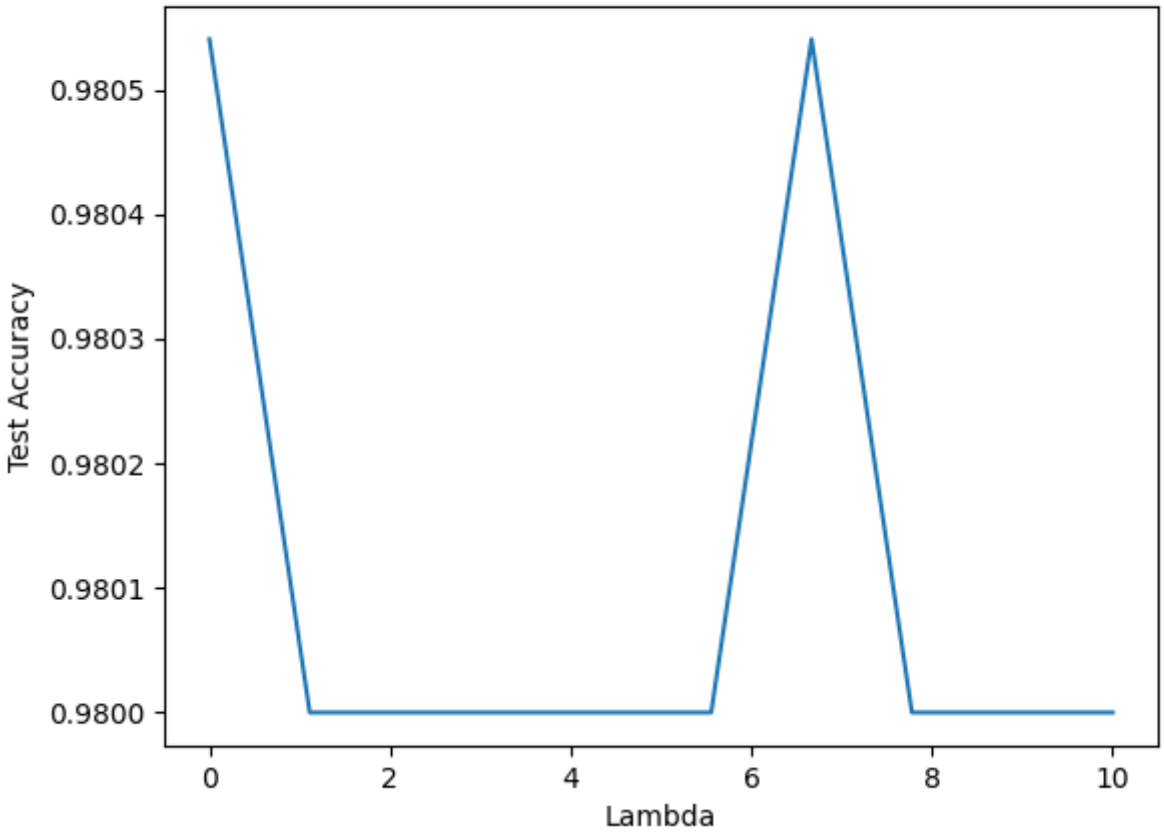
- Test error per Lambda



- Train Accuracy per Lambda



- Test Accuracy per Lambda



στ.

neural_network.py

Εδώ χτίζουμε το νευρωνικό δίκτυο το οποιό είναι ένα δυαδικό μοντέλο ταξινόμησης για το σύνολο δεδομένων MNIST, όπου ο στόχος είναι να προβλέψουμε εάν μια δεδομένη εικόνα είναι το ψηφίο 6 ή όχι. Το μοντέλο αποτελείται από ένα επίπεδο εισόδου (εικόνες 28x28 pixel), 2 κρυφά επίπεδα(hidden layers) με 64 μονάδες το καθένα και ένα στρώμα εξόδου(output) με 1 μονάδα. Το μοντέλο χρησιμοποιεί συνάρτηση σιγμοειδούς ενεργοποίησης και δυαδική συνάρτηση απώλειας διασταυρούμενης εντροπίας(cross entropy). Τα βάρη και οι προκαταλήψεις ενημερώνονται χρησιμοποιώντας gradient descent με ρυθμό εκμάθησης 0,001. Το μοντέλο εκπαιδεύεται για 5 εποχές και η ακρίβεια υπολογίζεται στο test σετ.

Περιέχει την τελική υλοποίηση που έχουν προστεθεί τα min batch και μας δείχνει την τιμή του loss με την μικρότερη τιμή του , οπου έχουμε και το καλύτερο performance του μοντέλου (σημαίνει οτι κάνει προσεγγιστικά οσο το δυνατόν καλυτερες προβλέψεις αν η εικόνα είναι 6 ή οχι)

```
Epoch 7200, Loss: 0.009281561402089072
Epoch 7300, Loss: 0.009115156774498288
Epoch 7300, Loss: 0.00878970138538825
Epoch 7300, Loss: 0.021837459392400525
Epoch 7400, Loss: 0.008956769008233856
Epoch 7400, Loss: 0.008628818201305117
Epoch 7400, Loss: 0.021537917607357328
Epoch 7500, Loss: 0.008803818950258162
Epoch 7600, Loss: 0.008656632159322587
Epoch 7600, Loss: 0.008325010888026646
Epoch 7600, Loss: 0.020982598306771252
Epoch 7700, Loss: 0.008512315144256126
Epoch 7700, Loss: 0.008176042942879286
Epoch 7700, Loss: 0.0206888197510918
Epoch 7700, Loss: 0.1540631744491343
Epoch 7800, Loss: 0.00837183275150908
Epoch 7900, Loss: 0.008237067337461247
Epoch 7900, Loss: 0.007882820979120714
Epoch 7900, Loss: 0.020028642228797057
Epoch 8000, Loss: 0.008106933268873434
Epoch 8000, Loss: 0.007749869606530316
Epoch 8000, Loss: 0.019768084344814522
Epoch 8000, Loss: 0.15417704274407415
Epoch 8100, Loss: 0.007979096768418438
Epoch 9100, Loss: 0.017067913473797904
Epoch 9200, Loss: 0.006802689335075975
Epoch 9200, Loss: 0.006392989157503616
Epoch 9200, Loss: 0.01680873477731639
Epoch 9300, Loss: 0.006711419632581558
Epoch 9300, Loss: 0.006301744230574843
Epoch 9300, Loss: 0.016631283701602672
Epoch 9400, Loss: 0.006623703921273523
Epoch 9500, Loss: 0.00653873173352415
Epoch 9600, Loss: 0.006453426654280402
Epoch 9600, Loss: 0.006036638156419101
Epoch 9600, Loss: 0.016045753359938582
Epoch 9600, Loss: 0.15489369402606873
Epoch 9700, Loss: 0.006371022733283351
Epoch 9800, Loss: 0.00629045042987901
Epoch 9900, Loss: 0.006213722200809042
Accuracy: 0.0958
```

ζ.

- Υλοποιήση early stopping μετά απο 5 εποχές (100 itterations) οπου δεν υπάρχει βελτίωση

```python
    loss_history.append(loss)
    #stop the training loop if there's no improvement in the loss after 5 epochs (100 iterations).
    if len(loss_history) > 100:
        if loss_history[-101] - loss < 1e-5:
            break
```

η.

- Εδω τοποθετούμαι τις διάφορες τιμές του δείκτη εκμάθησης (10 διαφορετικές) 10^-5 εώς 0.5

```python
        # Update the weights and biases
        learning_rate = 0.0001
```

- Εδω τοποθετούμαι τις διάφορες τιμές των νευρών για τα hidden layers (10 διαφορετικές) 2, 4, 8,16...

```
y_test = (y_test == 0).astype(int)

# Define the model
neurons=64
weights_1 = np.random.randn(neurons, 784)
weights_2 = np.random.randn(neurons, neurons)
weights_3 = np.random.randn(1, neurons)
bias_1 = np.zeros((neurons, 1))
bias_2 = np.zeros((neurons, 1))
bias_3 = np.zeros((1, 1))
```

I.

Εδώ σπάμε σε mini batches δοκιμάζοντας 10 διαφορετικές τιμές 2 ,4,8..

```
# Train the model
loss_history = []
batch_size = 128
for epoch in range(10000):
    for i in range(0, x_train.shape[0], batch_size):
        inputs_batch = x_train[i:i + batch_size].T

        #Forward pass
        hidden_layer_1 = sigmoid(np.dot(weights_1, inputs_batch) + bias_1)
        hidden_layer_2 = sigmoid(np.dot(weights_2, hidden_layer_1) + bias_2)
        logits = np.dot(weights_3, hidden_layer_2) + bias_3
        probs = sigmoid(logits)

        # Compute the loss
        labels = y_train[i:i + batch_size].reshape(1, -1)
        loss = cross_entropy(probs, labels)

        if math.isnan(loss)== False and epoch!=0 and epoch % 100 == 0 :
            print(f'Epoch {epoch}, Loss: {loss}')

        loss_history.append(loss)
        #stop the training loop if there's no improvement in the loss after 5 epochs (100 iterations).
        if len(loss_history) > 100:
            if loss_history[-101] - loss < 1e-5:
                break
```