

Μηχανική Μάθηση

1^η Σετ Ασκήσεων

ΟΝΟΜΑ: Αλέξανδρος

ΕΠΩΝΥΜΟ: Καλέργης

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 1066665

ΕΤΟΣ ΦΟΙΤΗΣΗΣ: 4ο

Πρόβλημα 1:

α) Το βέλτιστο τεστ κατά Bayes είναι το εξής:

$$\frac{\frac{f_1(X)}{f_0(X)}}{H_0} \stackrel{H_1}{\gtrless} \frac{(C_{10}-C_{00})\mathbb{P}(H_0)}{(C_{01}-C_{11})\mathbb{P}(H_1)}, \text{ όπου } X = [x_1, x_2] \text{ και } f_i(X) = f_i(x_1)f_i(x_2)$$

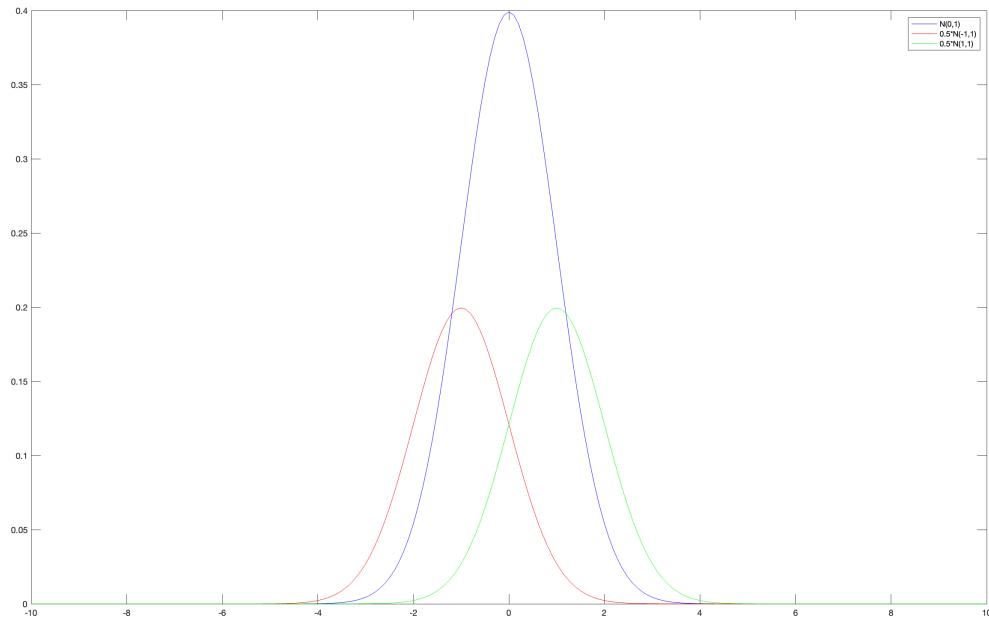
Για να βρούμε το τεστ που ελαχιστοποιεί τη πιθανότητα σφάλματος μπορούμε να θεωρήσουμε ότι $C_{00} = C_{11} = 0$ και $C_{10} = C_{01} = 1$. Αυτό έχει ως αποτέλεσμα το παραπάνω τεστ να πάρει τη μορφή:

$$\frac{\frac{f_1(X)}{f_0(X)}}{H_0} \stackrel{H_1}{\gtrless} \frac{(C_{10}-C_{00})\mathbb{P}(H_0)}{(C_{01}-C_{11})\mathbb{P}(H_1)} \quad \text{ή} \quad \frac{\frac{f_1(X)}{f_0(X)}}{H_0} \stackrel{H_1}{\gtrless} 1$$

β) Θέλουμε να δημιουργήσουμε δυο κατηγορίες δεδομένων ζευγαριών $[x_1, x_2]$ 1.000.000 σε αριθμό η κάθε κατηγορία. Αυτές τις μεταβλητές θα τις χρησιμοποιήσω για να ελέγξω σύμφωνα με το τεστ κατά Bayes που αναφερθήκαμε στο ερώτημα (α) τι συνολικό ποσοστό σφάλματος έχω για τα δεδομένα μου. Τα ζευγάρια αυτά είναι από τα ενδεχόμενα H_0 και H_1 και προκύπτουν από τις κατανομές:

$$f_0(X) \sim \mathcal{N}(0, 1)$$

$$f_1(X) \sim 0.5\{\mathcal{N}(-1, 1) + \mathcal{N}(1, 1)\}$$



Το συνολικό σφάλμα του Bayes τεστ είναι: 35.32%

```
/Users/alexkalerges/Desktop/ml1python/bin/python /Users/alexkalerges/PycharmProjects/ml1python/main.py
ERROR from f0(x): 28.2694 %
ERROR from f1(x): 42.3653 %
BAYES TOTAL ERROR: 35.317350000000005 %
```

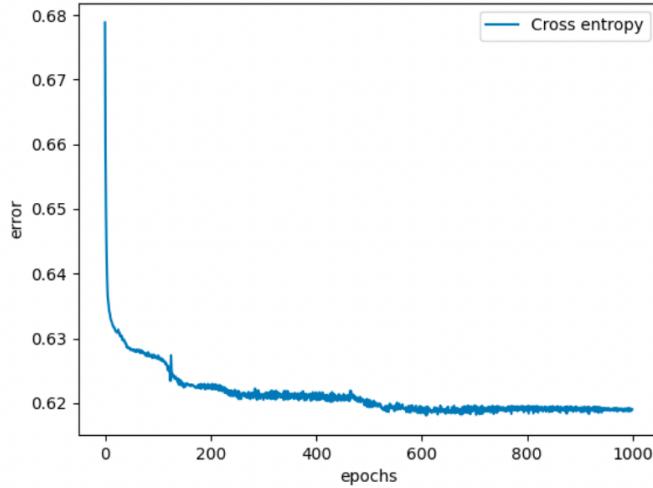
γ) Δημιουργούμε 200 επιπλέον δεδομένα από τις υλοποιήσεις μας τα οποία θα χρησιμοποιήσουμε για την εκπαίδευση νευρωνικών δικτύων. Για την εκπαίδευση των Νευρωνικών Δικτύων χρησιμοποιούνται συναρτήσεις και κλάσεις οι οποίες βρίσκονται στο τέλος της αναφοράς.

Νευρωνικά Δίκτυα

- Τα Νευρωνικά Δίκτυα είναι fully connected και έχουν διαστάσεις $2 \times 20 \times 1$ (2-είσοδοι / 20-κρυφό επίπεδο / 1-έξοδος).
- Ως συνάρτηση ενεργοποίησης στο κρυφό επίπεδο χρησιμοποιούμε την ReLU και στις δύο μεθόδους.
- Στην μέθοδο cross-entropy χρησιμοποιούμε και μια επιπλέον συνάρτηση ενεργοποίησης στην έξοδο, την sigmoid. Με αυτή τη συνάρτηση πετυχαίνουμε να περιορίσουμε το αποτέλεσμα της εξόδου στο διάστημα $[0,1]$ όπου και πρέπει να βρίσκεται η έξοδος, αφού προσεγγίζει την εκ των υστέρων πιθανότητα. Επιπλέον το τεστ του λόγου πιθανοφάνειας της εξόδου συγκρίνεται με το 0.5.
- Στην μέθοδο exponential δεν είναι απαραίτητο να κάνουμε το ίδιο αφού η έξοδος προσεγγίζει τον λογάριθμο του λόγου πιθανοφάνειας, επομένως η έξοδος μπορεί να πάρει τιμές σε όλο το \mathbb{R} . Επιπλέον το τεστ του λόγου πιθανοφάνειας της εξόδου συγκρίνεται με το 0.

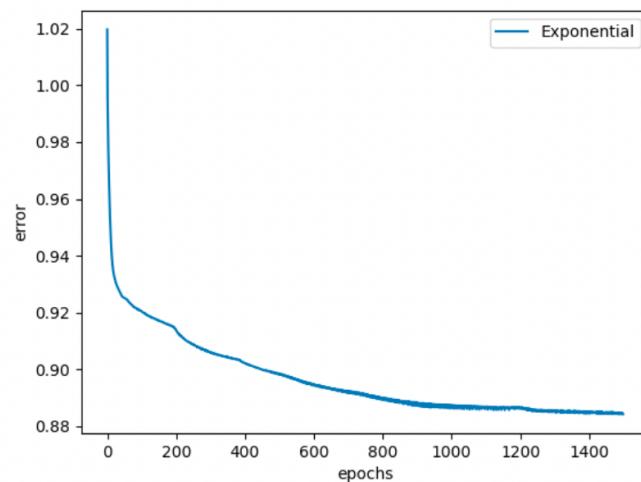
Εκπαίδευση δύο Νευρωνικά Δίκτυα με τις εξής μεθόδους:

1) Μέθοδος cross-entropy: $\varphi(z) = -\log(1 - z)$, $\psi(z) = -\log(z)$, $\omega(r) = \frac{r(x_1, x_2)}{1+r(x_1, x_2)}$



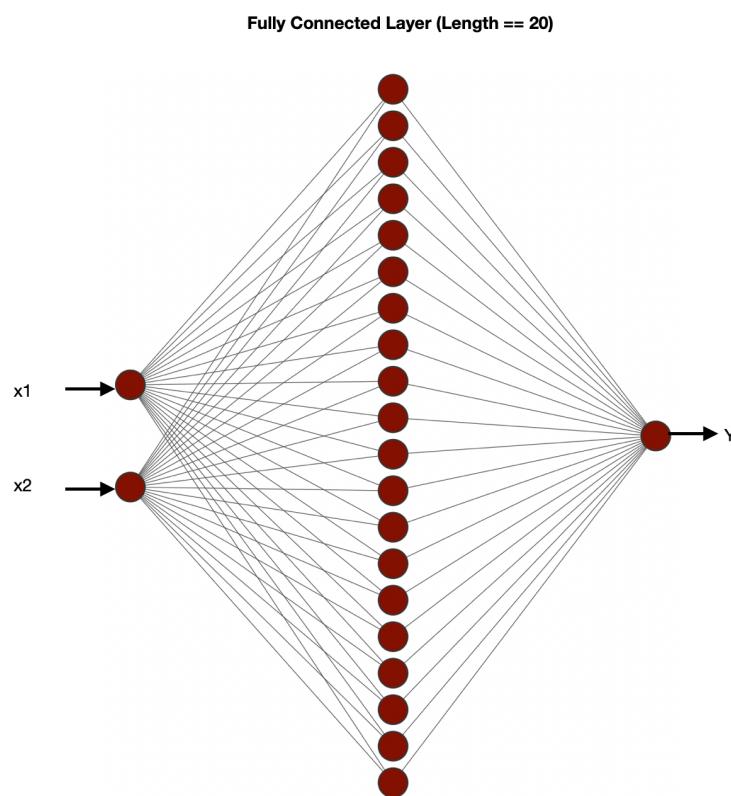
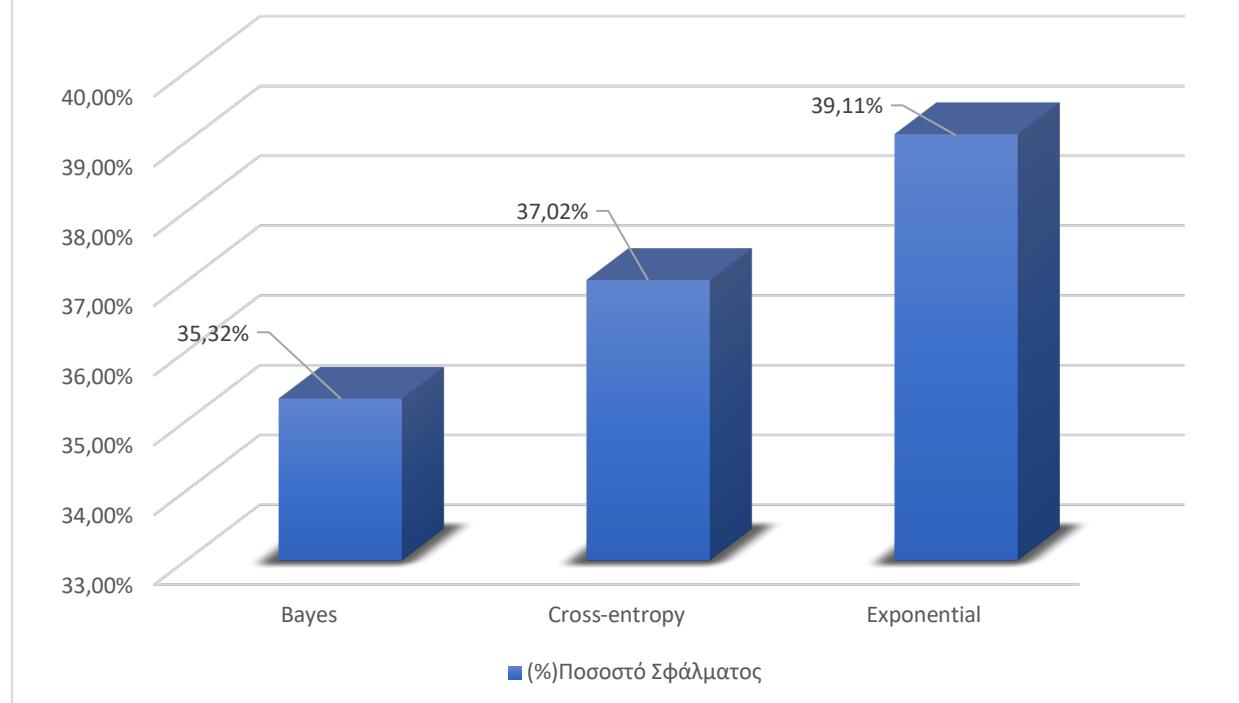
Το συνολικό σφάλμα του Cross-entropy τεστ είναι: 37.02%

2) Μέθοδος exponential: $\varphi(z) = e^{0.5z}$, $\psi(z) = e^{-0.5z}$, $\omega(r) = \log(r(x_1, x_2))$



Το συνολικό σφάλμα του Exponential τεστ είναι: 39.11%

Σύγκριση σφαλμάτων με Νευρωνικά Δίκτυα και Bayes



Κώδικας Προβλήματος 1

Αρχείο Data με τον κώδικα με τις συναρτήσεις που χρησιμοποιώ για τη δημιουργία των επιθυμητών μεταβλητών

```
import numpy as np
#Create Data
def create_data(N):
    x1 = np.random.normal(0, 1, (N,2))
    x2 = np.random.normal(np.random.choice([-1, 1], (N,2)), np.ones((N,2)))
    return x1, x2

def normal_dist(mean, sd, x):
    return 1 / (sd * np.sqrt(2 * np.pi)) * np.exp(-0.5 * np.square((x - mean) / sd))

def f0(x):
    return normal_dist(0, 1, x)

def f1(x):
    return 0.5 * (normal_dist(-1, 1, x) + normal_dist(1, 1, x))

def H_error_test(H, threshold):
    N = len(H)
    H0_count = np.count_nonzero(H < threshold)
    H1_count = N - H0_count
    return H0_count, H1_count
```

Δημιουργία των επιθυμητών 1.000.000 μεταβλητών

```
-----Create the 10^6 Data for the BAYES test-----
N_test = 1000000
(x1,x2) = create_data(N_test)
x1_test = x1
x2_test = x2
# likelihood ratio
r0 = f1(x1_test[:,0]) * f1(x1_test[:,1]) / (f0(x1_test[:,0]) * f0(x1_test[:,1]))
r1 = f1(x2_test[:,0]) * f1(x2_test[:,1]) / (f0(x2_test[:,0]) * f0(x2_test[:,1]))
# Bayes test
H0_error = H_error_test(r0, 1)[1]
H1_error = H_error_test(r1, 1)[0]
tot_error = 0.5*(H0_error/N_test + H1_error/N_test )
#print the Total error
print("BAYES TOTAL ERROR:", tot_error*100, "%")
```

Δημιουργία των επιπλέον 200 δεδομένων και κατάλληλη προσαρμογή για την εκπαίδευση των Νευρωνικών Δικτύων.

```
-----Network-----
# Create train data
N_train = 200
x1_train, x2_train = create_data(N_train)
# Train data
x1_train = x1_train.reshape((N_train, 2, 1))
x2_train = x2_train.reshape((N_train, 2, 1))
y1_train = np.zeros((N_train, 1))
y2_train = np.ones((N_train, 1))
x_train = [x1_train, x2_train]
y_train = [y1_train, y2_train]
# Test data
x1_test = x1_train.reshape(N_test, 2, 1)
x2_test = x2_train.reshape(N_test, 2, 1)
```

$$1) \text{Méθοδος Cross-entropy: } \varphi(z) = -\log(1 - z), \quad \psi(z) = -\log(z), \quad \omega(r) = \frac{r(x_1, x_2)}{1+r(x_1, x_2)}$$

```
#-----Cross-entropy network-----
nn_ce = Network()
nn_ce.add(FCLayer(2, 20))
nn_ce.add(ActivationLayer(rectified, d_rectified))
nn_ce.add(FCLayer(20, 1))
nn_ce.add(ActivationLayer(sigmoid, d_sigmoid))
# Train
nn_ce.use(cross_entropy, d_cross_entropy)
err_ce, ep_ce = nn_ce.fit(x_train, y_train, epochs=1000, learning_rate=0.001)
# Likelihood ratio test prediction
r0 = nn_ce.predict(x1_test)
r1 = nn_ce.predict(x2_test)
# H_error_test
H0_error = H_error_test(r0, 0.5)[1]
H1_error = H_error_test(r1, 0.5)[0]
tot_error = 0.5*(H0_error/N_test + H1_error/N_test)
# Print the Total error
print("ERROR from f0(x):", (H0_error/N_test)*100, "%")
print("ERROR from f1(x):", (H1_error/N_test)*100, "%")
print("CROSS_ENTROPY TOTAL ERROR:", tot_error*100, "%")
plt.plot(err_ce, label='cross-entropy')
plt.xlabel('epochs')
plt.ylabel('error')
plt.title('Cross-entropy Error Function')
plt.legend()
plt.show()
```

$$2) \text{Méθοδος Exponential: } \varphi(z) = e^{0.5z}, \quad \psi(z) = e^{-0.5z}, \quad \omega(r) = \log(r(x_1, x_2))$$

```
#-----Exponential network-----
nn_ex = Network()
nn_ex.add(FCLayer(2, 20))
nn_ex.add(ActivationLayer(rectified, d_rectified))
nn_ex.add(FCLayer(20, 1))
# Train
nn_ex.use(exponential, d_exponential)
err_ex, ep_ex = nn_ex.fit(x_train, y_train, epochs=1500, learning_rate=0.001)
# Likelihood ratio test prediction
r0 = nn_ex.predict(x1_test)
r1 = nn_ex.predict(x2_test)
# H_error_test
H0_error = H_error_test(r0, 0)[1]
H1_error = H_error_test(r1, 0)[0]
tot_error = 0.5*(H0_error/N_test + H1_error/N_test)
# Print the Total error
print("ERROR from f0(x):", (H0_error/N_test)*100, "%")
print("ERROR from f1(x):", (H1_error/N_test)*100, "%")
print("EXPONENTIAL TOTAL ERROR:", tot_error*100, "%")
plt.plot(err_ex, label='exponential')
plt.xlabel('epochs')
plt.ylabel('error')
plt.title('Exponential Error Function')
plt.legend()
plt.show()
```

Πρόβλημα 2:

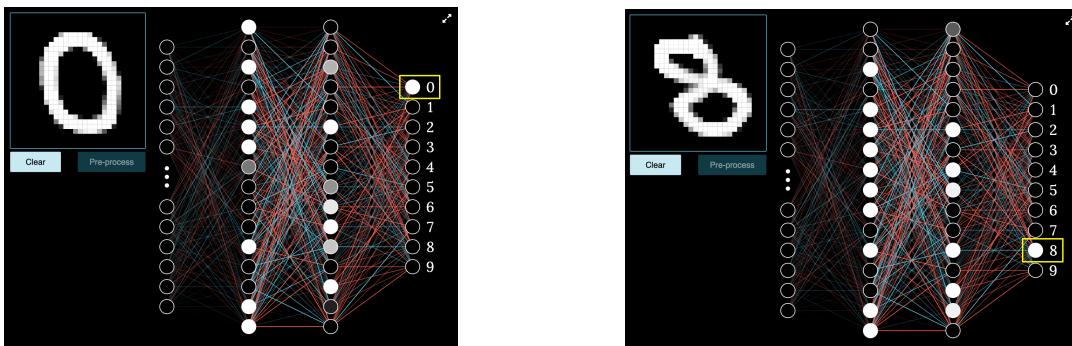
Σε αυτό το πρόβλημα έχουμε να εκπαιδεύσουμε επίσης δυο νέα Νευρωνικά Δίκτυα και στο ερώτημα γ του Προβλήματος 1, με τη διαφορά πως τα δεδομένα δεν τα δημιουργούμε εμείς. Τα δεδομένα μας είναι εικόνες μεγέθους 28×20 pixels από χειρόγραφα με τους αριθμούς από 0 έως 9 που υπάρχουν αποθηκευμένα στη βιβλιοθήκη mnist. Για να εισάγουμε τη βιβλιοθήκη κάνουμε import στη Python:

```
from keras.datasets import mnist
```

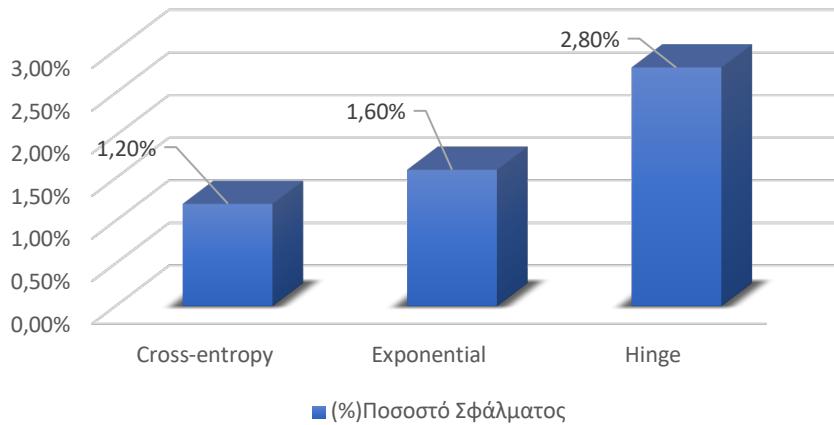
Τις εικόνες που έχουμε πρέπει να τις μετασχηματίσουμε σε διανύσματα 784×1 και να τις κανονικοποιήσουμε διαιρώντας με 255 έτσι ώστε να είναι κατάλληλες για το δίκτυο μας:

Νευρωνικά Δίκτυα

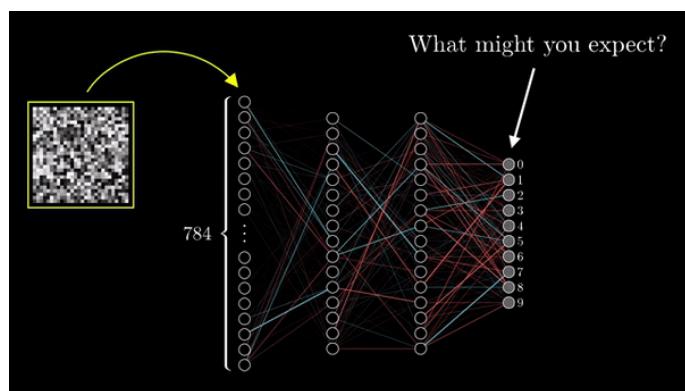
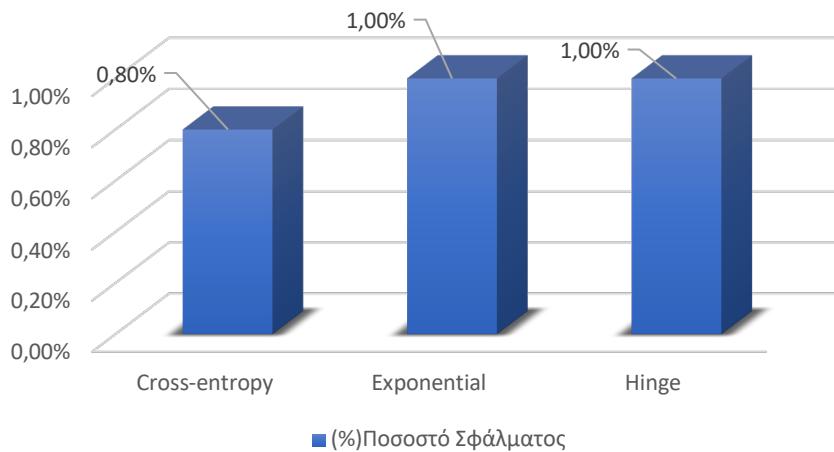
- Τα Νευρωνικά Δίκτυα είναι fully connected και έχουν διαστάσεις $784 \times 300 \times 1$ ($784 = 28 \times 28$ εισόδους / 300 κρυφό επίπεδο / 1 έξοδο)
- Ως συνάρτηση ενεργοποίησης στο κρυφό επίπεδο χρησιμοποιούμε την ReLU και στις δύο μεθόδους.
- Στην μέθοδο cross-entropy χρησιμοποιούμε και μια επιπλέον συνάρτηση ενεργοποίησης στην έξοδο, την sigmoid. Με αυτή τη συνάρτηση πετυχαίνουμε να περιορίσουμε το αποτέλεσμα της εξόδου στο διάστημα $[0,1]$ όπου και πρέπει να βρίσκεται η έξοδος, αφού προσεγγίζει την εκ των υστέρων πιθανότητα. Επιπλέον το τεστ του λόγου πιθανοφάνειας της εξόδου συγκρίνεται με το 0.5.
- Στην μέθοδο exponential δεν είναι απαραίτητο να κάνουμε το ίδιο αφού η έξοδος προσεγγίζει τον λογάριθμο του λόγου πιθανοφάνειας, επομένως η έξοδος μπορεί να πάρει τιμές σε όλο το \mathbb{R} . Επιπλέον το τεστ του λόγου πιθανοφάνειας της εξόδου συγκρίνεται με το 0.
- Στην μέθοδο hinge δεν είναι απαραίτητο να κάνουμε το ίδιο αφού η έξοδος προσεγγίζει το πρόσημο του λογαρίθμου του λόγου πιθανοφάνειας, επομένως η έξοδος μπορεί να πάρει τιμές σε όλο το \mathbb{R} . Επιπλέον για τον λόγο που αναφέραμε και στη hinge το threshold του test πιθανοφάνειας είναι 0.



Σύγκριση σφαλμάτων μεθόδων για τον αριθμό 0



Σύγκριση σφαλμάτων μεθόδων για τον αριθμό 8



Κανονικοποίηση κατά **ADAMS**:

$$\text{Ισχύς Βαρών: } P_w^t = P_w^{t-1}(1 - \lambda) + \lambda(\frac{\partial \varepsilon}{\partial W})^2$$

$$\text{Ισχύς της Κλίσης: } P_B^t = P_B^{t-1}(1 - \lambda) + \lambda(\frac{\partial \varepsilon}{\partial B})^2$$

Επειτα βρίσκουμε:

$$W^t = W^{t-1} - \mu \frac{\frac{\partial \varepsilon}{\partial W}}{\sqrt{P_w^{t-1} + c}}$$

$$B^t = B^{t-1} - \mu \frac{\frac{\partial \varepsilon}{\partial B}}{\sqrt{P_B^{t-1} + c}},$$

Όπου c ένας πολύ μικρός αριθμός ώστε να αποφύγουμε διαίρεση με το 0 και τα σφάλματα υπολογίζονται:

$$\frac{\partial \varepsilon}{\partial Y} = \begin{bmatrix} \frac{\partial \varepsilon}{\partial y_0} & \frac{\partial \varepsilon}{\partial y_1} & \dots & \frac{\partial \varepsilon}{\partial y_{m-1}} \end{bmatrix}$$

$$\frac{\partial \varepsilon}{\partial X} = \begin{bmatrix} \frac{\partial \varepsilon}{\partial x_0} & \frac{\partial \varepsilon}{\partial x_1} & \dots & \frac{\partial \varepsilon}{\partial x_{m-1}} \end{bmatrix} = \dots = W^T \frac{\partial \varepsilon}{\partial Y}$$

$$\frac{\partial \varepsilon}{\partial W} = \begin{bmatrix} \frac{\partial \varepsilon}{\partial w_{00}} & \dots & \frac{\partial \varepsilon}{\partial w_{0(n-1)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon}{\partial w_{(m-1)0}} & \dots & \frac{\partial \varepsilon}{\partial w_{(m-1)(n-1)}} \end{bmatrix} = \dots = \frac{\partial \varepsilon}{\partial Y} X^T$$

$$\frac{\partial \varepsilon}{\partial B} = \begin{bmatrix} \frac{\partial \varepsilon}{\partial b_0} & \frac{\partial \varepsilon}{\partial b_1} & \dots & \frac{\partial \varepsilon}{\partial b_{m-1}} \end{bmatrix} = \dots = \frac{\partial \varepsilon}{\partial Y}$$

Κώδικας Προβλήματος 2

Αρχικοποίηση των παραμέτρων εκπαίδευσης και τεστ του Νευρωνικού Δικτύου και εισαγωγής της βιβλιοθήκης MNIST.

```
from Network import*
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from Data import*

# load mnist dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# keep 0 - 8
train_filter = np.where((y_train == 0) | (y_train == 8))
test_filter = np.where((y_test == 0) | (y_test == 8))
x_train, y_train = x_train[train_filter], y_train[train_filter]
x_test, y_test = x_test[test_filter], y_test[test_filter]

# reshape and normalize
x_train = x_train.reshape(-1, 28*28, 1)
x_test = x_train.reshape(-1, 28*28, 1)

x_train = x_train.astype('float32')
x_train /= 255

x_test = x_test.astype('float32')
x_test /= 255

y_train = (y_train==8)*1
y_test = (y_test==8)*1

# split in 2 categories and make equal
x0_train = x_train[np.nonzero((y_train == 0))]
x1_train = x_train[np.nonzero((y_train == 1))]

x0_test = x_test[np.nonzero((y_test == 0))]
x1_test = x_test[np.nonzero((y_test == 1))]

y0_train = y_train[np.nonzero((y_train == 0))]
y1_train = y_train[np.nonzero((y_train == 1))]

y0_test = y_test[np.nonzero((y_test == 0))]
y1_test = y_test[np.nonzero((y_test == 1))]

x0_train = x0_train[:5851]
y0_train = y0_train[:5851]

x0_test = x0_test[:974]
y0_test = y0_test[:974]

x_train = [x0_train, x1_train]
y_train = [y0_train, y1_train]
```

1) Μέθοδος Hinge: $\varphi(z) = \max(1 + z, 0)$, $\psi(z) = \max(1 - z, 0)$, $\omega(r) = \text{sign}(\log(r(x_1, x_2)))$

```
# hinge network
nn_hi = Network()
nn_hi.add(FCLayer(784, 300))
nn_hi.add(ActivationLayer(relu, d_relu))
nn_hi.add(FCLayer(300, 1))
# train
nn_hi.use(hinge, d_hinge)
err_hi, ep_hi = nn_hi.fit(x_train, y_train, epochs=40, learning_rate=0.000001)
# evaluate
r0 = nn_hi.predict(x0_test)
r1 = nn_hi.predict(x1_test)
# likelihood ratio test
H0_error = H_error_test(r0, 0)[1]
H1_error = H_error_test(r1, 0)[0]
tot_error = (H0_error + H1_error) / (2 * len(x0_test)) * 100
```

2) Μέθοδος Cross-entropy: $\varphi(z) = -\log(1 - z)$, $\psi(z) = -\log(z)$, $\omega(r) = \frac{r(x_1, x_2)}{1+r(x_1, x_2)}$

```
# cross entropy
nn_ce = Network()
nn_ce.add(FCLayer(784, 300))
nn_ce.add(ActivationLayer(relu, d_relu))
nn_ce.add(FCLayer(300, 1))
nn_ce.add(ActivationLayer(sigmoid, d_sigmoid))
# train
nn_ce.use(cross_entropy, d_cross_entropy)
err_ce, ep_ce = nn_ce.fit(x_train, y_train, epochs=40, learning_rate=0.000001)
# evaluate
r0 = nn_ce.predict(x0_test)
r1 = nn_ce.predict(x1_test)
# likelihood ratio test
H0_error = H_error_test(r0, 0.5)[1]
H1_error = H_error_test(r1, 0.5)[0]
tot_error = (H0_error + H1_error) / (len(x0_test) + len(x1_test)) * 100
```

3) Μέθοδος Exponential: $\varphi(z) = e^{0.5z}$, $\psi(z) = e^{-0.5z}$, $\omega(r) = \log(r(x_1, x_2))$

```
# Exponential
nn_ex = Network()
nn_ex.add(FCLayer(784, 300))
nn_ex.add(ActivationLayer(relu, d_relu))
nn_ex.add(FCLayer(300, 1))
# train
nn_ex.use(exponential, d_exponential)
err_ex, ep_ex = nn_ex.fit(x_train, y_train, epochs=40, learning_rate=0.000001)
# evaluate
r0 = nn_ex.predict(x0_test)
r1 = nn_ex.predict(x1_test)
# likelihood ratio test
H0_error = H_error_test(r0, 0)[1]
H1_error = H_error_test(r1, 0)[0]
tot_error = (H0_error + H1_error) / (2 * len(x0_test)) * 100
```

Κλάσεις για την εκπαίδευση των Νευρωνικού Δικτύου:

```

class Network:
    def __init__(self):
        self.layers = []
        self.loss = None
        self.d_loss = None

    def add(self, layer):
        self.layers.append(layer)

    def use(self, loss, d_loss):
        self.loss = loss
        self.d_loss = d_loss

    def predict(self, input_data):
        samples = len(input_data)
        result = []
        for i in range(samples):
            output = input_data[i]
            for layer in self.layers:
                output = layer.forward_propagation(output)
            result.append(output)
        return np.array(result)

    def fit(self, x_train, y_train, epochs, learning_rate):
        samples = len(x_train[0])
        err_arr = []
        for _ in range(epochs):
            err0 = [0]
            err1 = [0]
            for i in range(samples):
                # output0
                output0 = np.copy(x_train[0][i])
                for layer in self.layers:
                    output0 = layer.forward_propagation(output0)
                # output1
                output1 = np.copy(x_train[1][i])
                for layer in self.layers:
                    output1 = layer.forward_propagation(output1)
                err0.append(err0[-1] + self.loss(y_train[0][i], output0))
                err1.append(err0[-1] + self.loss(y_train[1][i], output1))
                error = self.d_loss(y_train[0][i], output0) +
self.d_loss(y_train[1][i], output1)
                for layer in reversed(self.layers):
                    error = layer.backward_propagation(error, learning_rate)
                # εξουαλώνση με 20 τελευταίες τιμές
                # err = sum(err0[-20:])/len(err0) + sum(err1[-20:])/len(err1)
                err = sum(err0)/ len(err0) + sum(err1) / len(err1)
                err_arr.append(err[0][0])
                print('epoch %d/%d error=%f' % (_+1, epochs, err))
            return err_arr, epochs

```

```

class ActivationLayer:
    def __init__(self, activation, d_activation):
        self.activation = activation
        self.d_activation = d_activation

    # Returns the activated input
    def forward_propagation(self, input_data):
        self.input = input_data
        self.output = self.activation(self.input)
        return self.output

    def backward_propagation(self, output_error, learning_rate):
        return self.d_activation(self.input) * output_error

```

```

class FCLayer:
    def __init__(self, input_size, output_size):
        self.weights = np.random.normal(0, 1 / (input_size + output_size), (output_size,
input_size))
        self.offset = np.zeros((output_size, 1))
        self.l = 1
        self.E_weights = 0
        self.E_offset = 0

    def forward_propagation(self, input_data):
        self.input = input_data
        self.output = np.dot(self.weights, self.input) + self.offset
        return self.output

    def backward_propagation(self, output_error, learning_rate):
        input_error = np.dot(self.weights.T, output_error)
        weights_error = np.dot(output_error, self.input.T)
        # Normalize Derivatives
        self.E_weights = self.E_weights * (1 - self.l) + self.l * np.square(weights_error)
        self.E_offset = self.E_offset * (1 - self.l) + self.l * np.square(output_error)
        self.l = 0.005
        # Gradient descent
        self.weights -= learning_rate * weights_error / np.sqrt(self.E_weights + 10 ** -6)
        self.offset -= learning_rate * output_error / np.sqrt(self.E_offset + 10 ** -6)
        return input_error

```

Συναρτήσεις ενεργοποίησης και σφάλματος:

```

# ACTIVATION FUNCTIONS
# Relu
def relu(x):
    return np.maximum(0, x)

def d_relu(x):
    x[x <= 0] = 0
    x[x > 0] = 1
    return x

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

def d_sigmoid(x):
    return np.exp(-x) / np.square(1 + np.exp(-x))

```

```

# LOSS FUNCTIONS
#----Cross entropy-----
def cross_entropy(y_true, y_pred):
    if y_true == 0:
        return -np.log(1 - y_pred)
    elif y_true == 1:
        return -np.log(y_pred)
def d_cross_entropy(y_true, y_pred):
    if y_true == 0:
        return 1 / (1 - y_pred)
    elif y_true == 1:
        return -1 / y_pred
#----Exponential-----
def exponential(y_true, y_pred):
    if y_true == 0:
        return np.exp(0.5 * y_pred)
    elif y_true == 1:
        return np.exp(-0.5 * y_pred)
def d_exponential(y_true, y_pred):
    if y_true == 0:
        return 0.5 * np.exp(0.5 * y_pred)
    elif y_true == 1:
        return -0.5 * np.exp(-0.5 * y_pred)
#----Hinge-----
def hinge(y_train, y_pred):
    if y_train == 0:
        return np.maximum(0, 1 + y_pred)
    elif y_train == 1:
        return np.maximum(0, 1 - y_pred)
def d_hinge(y_train, y_pred):
    y_pred[y_pred<=0] = 0
    if y_train == 0:
        y_pred[y_pred>0] = 1
        return y_pred
    elif y_train == 1:
        y_pred[y_pred>0] = -1
        return y_pred

```

