Machine Learning Homework I

PAPAGEORGIOU GEORGIOS
AM: 1092811

November 2024

1 Πρόβλημα Πρώτο

Για το διάνυσμα που μας δόθηκε $\mathbf{X} = [x_1, x_2]$ έχουμε τις ακόλουθες δύο υποθέσεις, τις οποίες καλούμαστε να εξετάσουμε.
$\mathcal{H}_0$: x_1, x_2 ανεξάρτητα με πυκνότητα πιθανότητας $f_0(x_1, x_2) = f_0(x_1)f_0(x_2)$ όπου $f_0(x) \sim \mathcal{N}(0, 1)$.
$\mathcal{H}_1$: x_1, x_2 ανεξάρτητα με πυκνότητα πιθανότητας $f_1(x_1, x_2) = f_1(x_1)f_1(x_2)$ όπου $f_1(x) \sim 0.5\{\mathcal{N}(-1, 1)+\mathcal{N}(1, 1)\}$.
Και οι δύο εκ των προτέρων πιθανότητες είναι ίσες $\mathcal{P}(\mathcal{H}_0) = \mathcal{P}(\mathcal{H}_1) = 0.5$

1.1 Ερώτημα Α

Το βέλτιστο τεστ κατά Bayes, το οποίο ελαχιστοποιεί την πιθανότητα σφάλματος είναι η εξέταση του λόγου πιθανοφάνειας,

$$\mathcal{L}(X) = \frac{f_1(X)}{f_0(X)} \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtreqless}} 1 \tag{1}$$

Δηλαδή, αποφασίζουμε υπέρ της υπόθεσης με την μεγαλύτερη πιθανοφάνεια.

1.2 Ερώτημα Β

Με την χρήση γεννήτριας τυχαίων αριθμών, κατασκευάζουμε 10^6 ζευγάρια $[x_1, x_2]$ από την $f_0(x_1, x_2)$ και άλλα 10^6 από την $f_1(x_1, x_2)$. Για την $f_1(x)$ θα πρέπει το δείγμα να ακολουθεί γκαουσιανή με μέση τιμή -1 με πιθανότητα 0.5, και γκαουσιανή με μέση τιμή 1 με πιθανότητα πάλι 0.5. Η επιλογή της προέλευσης του δείγματος X, μπορεί να γίνει χρησιμοποιώντας μια ομοιόμορφη κατανομή $\mathcal{U}(0, 1)$, συγκεκριμένα δημιουργούμε μια τυχαία τιμή από την $\mathcal{U}(0, 1)$, αν η τιμή είναι μικρότερη από 0.5 τότε το δείγμα αντλείτε από την $\mathcal{N}(-1, 1)$, αλλιώς αν είναι μεγαλύτερη από 0.5, αντλείτε από την $\mathcal{N}(1, 1)$.

Βέλτιστο τεστ κατά Bayes

Θα υπολογίσουμε την πιθανότητα σφάλματος με την χρήση προσομοίωσης, συγκεκριμένα αφού δημιουργήσουμε τα ζευγάρια τυχαίων μεταβλητών, θα εφαρμόσουμε το τεστ του λόγου πιθανοφάνειας (1), και θα υπολογίσουμε τις δύο πιθανότητες σφάλματος, $\mathcal{P}(\mathcal{D}_1|\mathcal{H}_0)$ και $\mathcal{P}(\mathcal{D}_0|\mathcal{H}_1)$ ή σφάλματα *Τύπου 1* και *Τύπου 2* αντίστοιχα.

Αποτελέσματα Προσομοίωσης

Ο κώδικας που χρησιμοποιήθηκε για την προσομοίωση παρατίθεται στο Παράρτημα Β′, και για την παραγωγή των τυχαίων αριθμών στο Α′. Τα αποτελέσματα της προσομοίωσης παρουσιάζονται στον Πίνακα 1. Το συνολικό σφάλμα υπολογίστηκε από την σχέση $0.5\mathcal{P}(\mathcal{D}_1|\mathcal{H}_0) + 0.5\mathcal{P}(\mathcal{D}_0|\mathcal{H}_1)$

Το συνολικό σφάλμα αποτελεί το βέλτιστο σφάλμα κατά Bayes και δεν μπορεί να βελτιωθεί από καμία μέθοδο.

Πίνακας 1: Αποτελέσματα Βέλτιστου Bayes τεστ

Μεθοδος	$\mathcal{P}(\mathcal{D}_1\|\mathcal{H}_0)$	$\mathcal{P}(\mathcal{D}_0\|\mathcal{H}_1)$	Συνολικό Σφάλμα
Bayes	0.2816	0.4240	**0.3528**

1.3 Προσέγγιση Λόγου Πιθανοφάνειας με Νευρωνικό Δίκτυο

Κρατώντας τα ζευγάρια που δημιουργήσαμε στο (1.2) δημιουργούμε 200 επιπλέων ζευγάρια, με την ίδια μέθοδο, τα οποία θα χρησιμοποιηθούν για την εκπαίδευση του νευρωνικού δικτύου.

Σχεδιασμός του Νευρωνικού Δικτύου

Για την επίτευξη του στόχου, θα χρησιμοποιήσουμε ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο με διαστάσεις $2 \times 20 \times 1$

$$W_1 = A_1 X + B_1, \quad Z_1 = g_1(W_1), \quad W_2 = A_2 Z_1 + b_2, \quad Z_2 = g_2(W_2) = u(X, \theta) \tag{2}$$

όπου η είσοδος, A_1, A_2, οι πίνακες που περιέχουν, τα βάρη του κάθε επιπέδου με διαστάσεις 20×2 και 1×20 αντίστοιχα, ο πίνακας, B_1 με διάσταση 20×1 περιέχει τα offset του κρυφού επιπέδου, και b_2 βαθμωτό μέγεθος πού είναι το offset του επιπέδου εξόδου, τέλος, g_0, g_1 οι συναρτήσεις ενεργοποίησης (activation function), του κάθε επιπέδου. Για το συγκεκριμένο δίκτυο θα χρησιμοποιηθεί συνάρτηση ενεργοποίησης του κρυφού επιπέδου η $g_1(z) = ReLU(z) = max\{0, z\}$. Η g_2 θα οριστεί παρακάτω κατά περίπτωση, καθώς εξαρτάτε από τον εκάστοτε μετασχηματισμό του λόγου πιθανοφάνειας που σκοπεύουμε να υπολογίσουμε. Μπορούμε να συμβολίζουμε με $\theta = \{A_1, B_1, A_2, b_2\}$ το διάνυσμα, που περιέχει όλες τις παραμέτρους του δικτύου.

Εκπαίδευση του Νευρωνικού Δικτύου

Θα εκπαιδεύσουμε το δίκτυο με την μέθοδο που αναφέρετε στο [1], δηλαδή, θα υποθέσουμε την συνάρτηση κόστους

$$\mathcal{J}(u) = E_0[\phi(u(X)) + r(X)\psi(u(X))] = E_0[\phi(u(X))] + E_1[\psi(u(X))] \tag{3}$$

όπου $r(X) = \frac{f_1(X)}{f_0(X)}$ ο λόγος πιθανοφάνειας. Χρησιμοποιώντας τον Νόμο των Μεγάλων Αριθμών, για την προσέγγιση των μέσων όρων, καθώς και το θεώρημα το οποίο αναφέρει ότι, αν ένα νευρωνικό δίκτυο είναι επαρκώς μεγάλο τότε μπορεί να προσεγγίσει κάθε μη γραμμική συνάρτηση [2], ή (3) γράφετε ως,

$$\mathcal{J}(u) \approx \hat{\mathcal{J}}(\theta) = \frac{1}{n_0}\sum_{i=1}^{n_0} \phi(u(X_i^0, \theta)) + \frac{1}{n_1}\sum_{i=1}^{n_1} \psi(u(X_i^1, \theta)) \tag{4}$$

Οπότε θα λύσουμε το παρακάτω πρόβλημα βελτιστοποίησης,

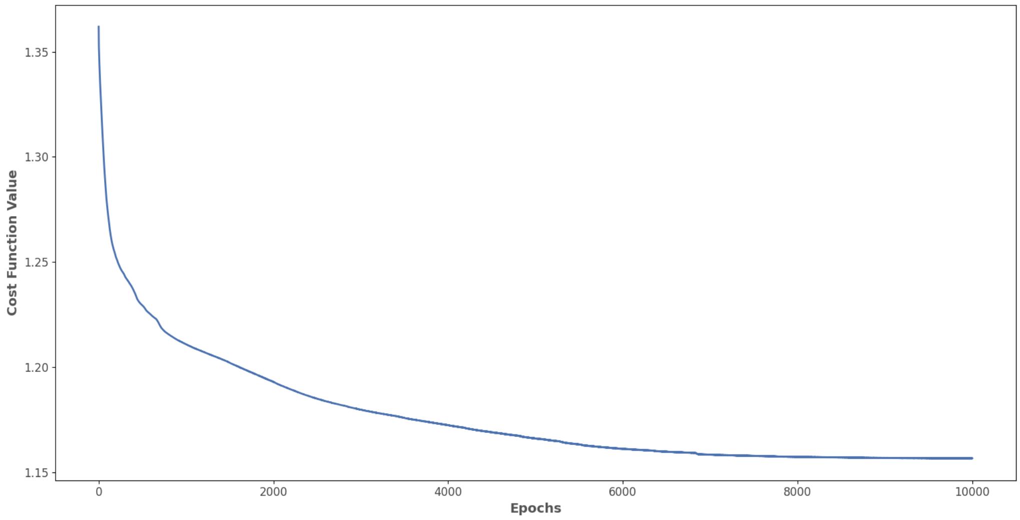
$$\min_{u(X)} \mathcal{J}(u) \approx \min_{\theta} \hat{\mathcal{J}}(\theta) = \min_{\theta}\left\{\frac{1}{n_0}\sum_{i=1}^{n_0} \phi(u(X_i^0, \theta)) + \frac{1}{n_1}\sum_{i=1}^{n_1} \psi(u(X_i^1, \theta))\right\} \tag{5}$$

Όπου η συνάρτηση, $u(X, \theta)$ είναι το νευρωνικό δίκτυο (είσοδος X, παράμετροι θ), όταν λυθεί το (5), δηλαδή βρεθεί το βέλτιστο θ_0 τότε το δίκτυο θα προσεγγίζει τον μετασχηματισμό $\omega(\frac{f_1(X)}{f_0(X)})$. Οπού ω, ϕ, ψ γνωστές συναρτήσεις, όπου πληρούν συγκεκριμένα κριτήρια.

Για την επίλυση του (5), θα χρησιμοποιήσουμε Stochastic Gradient Descent,

$$\theta_t = \theta_{t-1} - \mu\{\phi'(u(X_t^0, \theta_{t-1}))\nabla_\theta u(X_t^0, \theta_{t-1}) + \psi'(u(X_t^1, \theta_{t-1}))\nabla_\theta u(X_t^0, \theta_{t-1})\} \tag{6}$$

και συγκεκριμένα την παραλλαγή **ADAM**, οπού σε κάθε επανάληψη διαιρούμε την κλίση με την τετραγωνική ρίζα της ισχύος της, έτσι πετυχαίνουμε σχετικά ομοιόμορφη σύγκλιση όλων των παράγωγων.



Σχήμα 1: Συνάρτηση κόστους Cross-Entropy

1.4 Προσομοίωση και Σύγκριση Μεθόδων

Cross Entropy

Θα ξεκινήσουμε με την Cross-Entropy συνάρτηση κόστους, δηλαδή επιλέγουμε $\phi(z) = -log(1-z)$, $\psi(z) = -log(z)$, επιπλέων θα χρησιμοποιήσουμε τον μετασχηματισμό, $\omega(r) = \frac{r}{r+1}$ όπου αφου το $r(X)$ υποδηλώνει πιθανοφάνεια το $\omega(r(X)) = \frac{r(X)}{1+r(X)}$ θα υποδηλώνει εκ των υστέρων πιθανότητα. Επειδή ή έξοδος θα πρέπει να βρίσκετε, στο διάστημα $(0,1)$ στην έξοδο θα τοποθετηθεί μια μή γραμμικότητα, ως συνάρτηση ενεργοποίησης, συγκεκριμένα η σιγμοειδής, δηλαδή $g_2(z) = \sigma(z) = \frac{e^z}{1+e^z}$

Τρέχουμε τον αλγόριθμο με βήμα $\mu = 0.001$ και παράμετρο κανονικοποίησης για την μέθοδο **ADAM** $\lambda = 0.01$, και στίς δύο συναρτήσεις κόστους ο αλγόριθμος έτρεξε για 10000 epochs, παρακολουθώντας τις τιμές του κόστους ανά epoch. Το Σχ. 1 παρουσιάζει το πώς μεταβάλετε η τιμή της συνάρτησης κόστους ανά epoch, μέχρι που συγλίνει ο αλγόριθμος (οριζοντιώνετε η καμπύλη).

Exponential

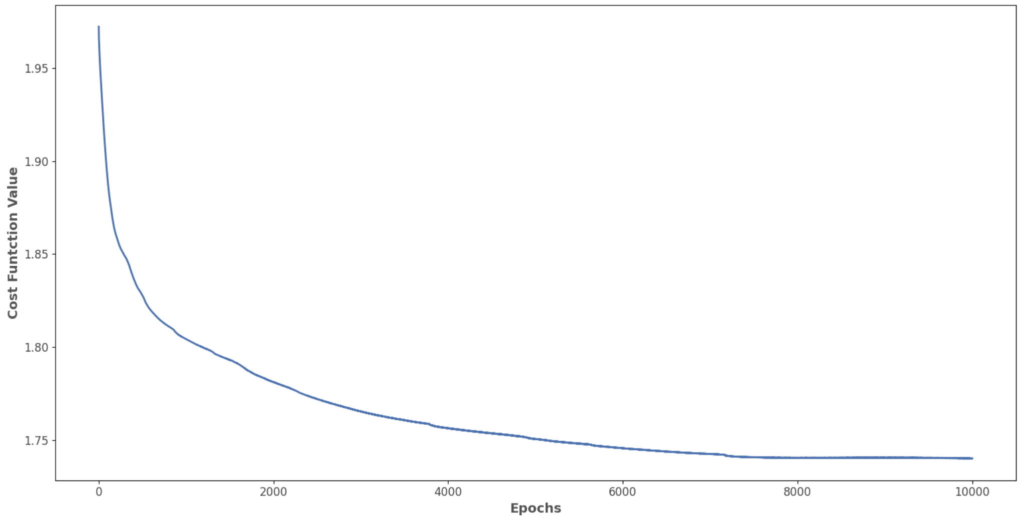
Επιλέγουμε, $\phi(z) = e^{0.5z}, \psi(z) = e^{-0.5z}$, και τον μετασχηματισμό $\omega(r) = log(r)$, οπού υπολογίζουμε τον λογάριθμο του λόγου πιθανοφάνειας. Εδώ δεν θα χρειαστεί μη γραμμικότητα στην έξοδο καθώς ο λογάριθμος του λόγου πιθανοφάνειας παίρνει οποιαδήποτε πραγματική τιμή, συνεπώς $g_2(z) = z$. Τρέχουμε τον αλγόριθμο για 1000 epochs και εδώ. Το Σχ. 2 παρουσιάζει το πώς μεταβάλετε η τιμή της συνάρτησης κόστους ανά epoch μέχρι που συγκλίνει αλγόριθμος.

Εφαρμογή Δεδομένων

Ο κώδικας που χρησιμοποιήθηκε παρατίθεται στο Παράρτημα Β′ για την προσομοίωση, και στα Γ′ Δ′ για τα νευρωνικά δίκτυα. Θα εφαρμόσουμε τώρα τα ζεύγη των τυχαίων μεταβλητών, στα δύο νευρωνικά δίκτυα με σκοπό, να λάβουμε μια απάντηση για το ποια από ποία απο τις δύο αρχικές υποθέσεις ικανοποιεί το κάθε ένα.

Θα εφαρμόσουμε πάλι το τεστ του λόγου πιθανοφάνειας άλλα τώρα στον εκάστοτε μετασχηματισμό, δηλαδή, για την περίπτωση της Cross-Entropy που εκτιμούμε την εκ των υστέρων πιθανότητα, η σύγκριση έχει ως εξής,

$$\omega(r(X)) \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtreqless}} \omega(1) \Leftrightarrow \omega(r(X)) \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtreqless}} \frac{1}{1+1} \Leftrightarrow \boxed{\omega(r(X)) \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtreqless}} \frac{1}{2}}$$



Σχήμα 2: Συνάρτηση κόστους Exponential

Πίνακας 2: Αποτελέσματα Εκτίμησης Λόγου Πιθανοφάνειας

Συνάρτηση Κόστους	$\mathcal{P}(\mathcal{D}_1\|\mathcal{H}_0)$	$\mathcal{P}(\mathcal{D}_0\|\mathcal{H}_1)$	Συνολικό Σφάλμα
Cross-Entropy	0.3441	0.4529	**0.3985**
Exponential	0.2949	0.4752	**0.3851**

ενώ στην περίπτωση για της Exponential έχουμε,

$$\omega(r(X)) \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtreqless}} \omega(1) \Leftrightarrow \omega(r(X)) \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtreqless}} log(1) \Leftrightarrow \boxed{\omega(r(X)) \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtreqless}} 0}$$

Από τα παραπάνω συμπεραίνουμε ότι, αν η τιμή της ω είναι μεγαλύτερη από $\frac{1}{2}$ για την Cross-Entropy ή 0 για την Exponential, αποφασίζουμε υπέρ της $\mathcal{H}_1$ αλλιώς, αποφασίζουμε υπέρ της $\mathcal{H}_0$, ή ισότητα έχει μηδενική πιθανότητα εμφάνισης.

Εφαρμόζοντας τα ζεύγη στα νευρωνικά δίκτυα λαμβάνουμε τα αποτελέσματα που απεικονίζονται στον Πίνακα 2. Παρατηρούμε ότι και στις δύο περιπτώσεις, το συνολικό σφάλμα είναι όπως περιμέναμε μεγαλύτερο από το βέλτιστο σφάλμα κατά Bayes, επιβεβαιώνοντας έτσι ότι το νευρωνικό δίκτυο προσπαθεί να προσεγγίσει τον βέλτιστο κανόνα απόφασης κατά Bayes.

2 Πρόβλημα Δέυτερο

Θα εφαρμόσουμε την ίδια ιδέα με το (1.3) θα σχεδιάσουμε ένα νευρωνικό δίκτυο που θα ξεχωρίζει χειρόγραφες εικόνες των ψηφίων μηδέν και οκτώ. Για τον σκοπό αυτό θα χρησιμοποιήσουμε την βιβλιοθήκη **MNIST** και συγκεκριμένα από τις εικόνες που δίνει για εκπαίδευση θα χρησιμοποιήσουμε 5000 από το κάθε ψηφίο, και από τις εικόνες που δίνει για δοκιμή, θα χρησιμοποιήσουμε 997 εικόνες από το κάθε ψηφίο.

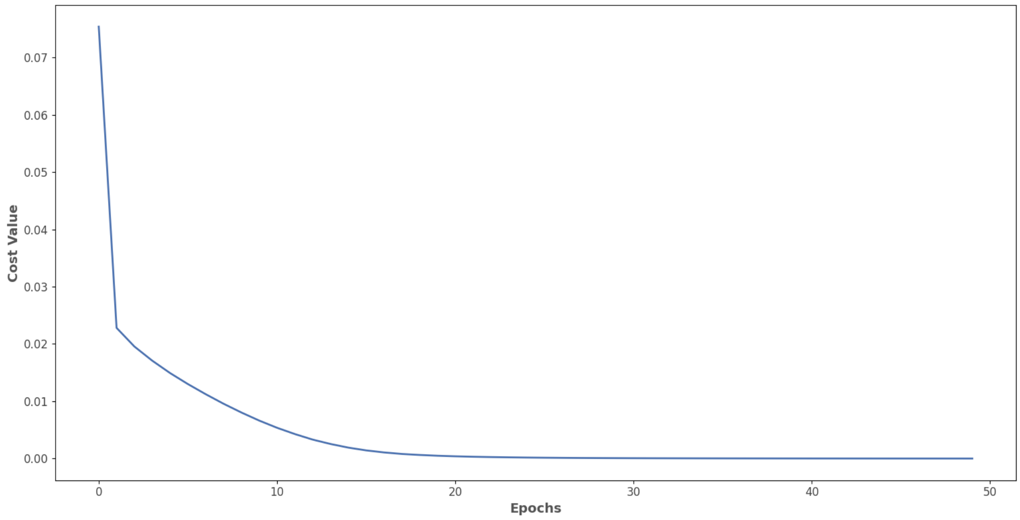
Το πρόβλημα με τις εικόνες είναι ακριβώς ισοδύναμο με το προηγούμενο, εδώ η τυχαία μεταβλητή $\mathbf{X}$, είναι οι εικόνες, οπότε μπορούμε να ορίσουμε τις ακόλουθες υποθέσεις,

$\mathcal{H}_0 : \mathbf{X} \sim \mathcal{F}_0(X)$

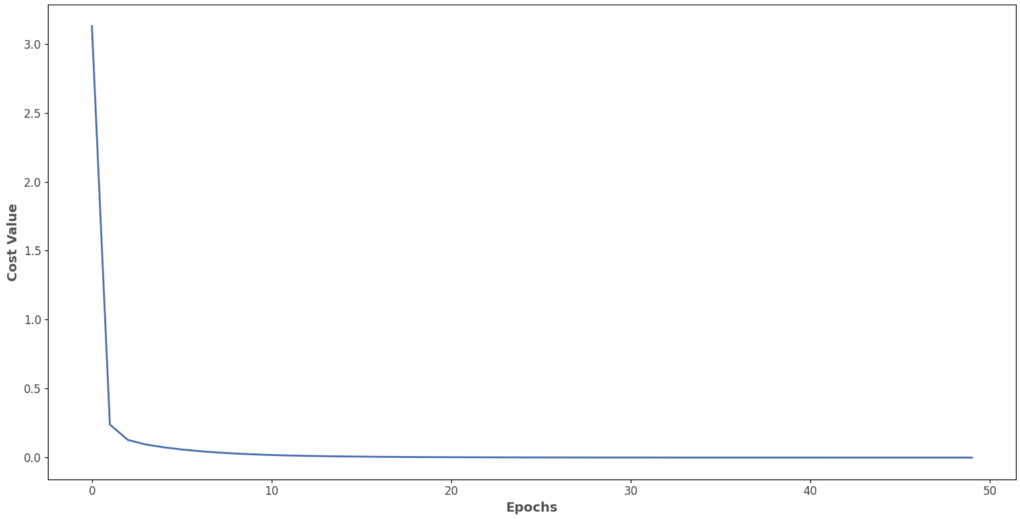
$\mathcal{H}_1 : \mathbf{X} \sim \mathcal{F}_1(X)$

Όπου $\mathcal{F}_0(X)$ είναι η κατανομή από την οποία προέρχονται οι εικόνες με το ψηφίο μηδέν.

Όπου $\mathcal{F}_1(X)$ είναι η κατανομή από την οποία προέρχονται οι εικόνες με το ψηφίο οκτώ. Το νευρωνικό δίκτυο και εδώ θα προσπαθεί να προσεγγίσει ένα μετασχηματισμό ω του λόγου πιθανοφάνειας, που αποτελεί τον βέλτιστο



Σχήμα 3: Συνάρτηση Κόστους Cross-Entropy



Σχήμα 4: Συνάρτηση Κόστους Exponential

τρόπο εξέτασης υποθέσεων.

2.1 Σχεδιασμός του Νευρωνικού δικτύου

Θα χρησιμοποιήσουμε ένα δίκτυο $784 \times 300 \times 1$, 784 είσοδοι, διότι οι εικόνες είναι 28×28 και θα τις εισάγουμε ως διανύσματα. Ισχύουν οι ίδιες εξισώσεις με το προηγούμενο δίκτυο (2) άλλα με διαφορετικά μεγέθη πινάκων για τις παραμέτρους. Συγκεκριμένα οι πίνακες $A_1,\ B_1,\ A_2$ θα έχουν διαστάσεις $300 \times 784,\ 300 \times 1,\ 1 \times 300$, αντίστοιχα και το b_2 βαθμωτό μέγεθος.

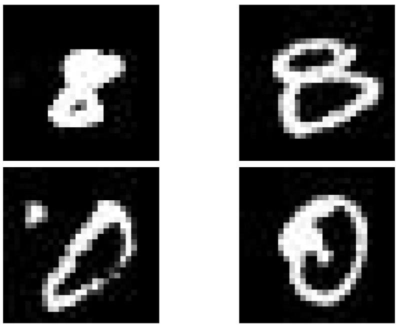
Θα εφαρμόσουμε τον ίδιο αλγόριθμο (6) ώστε να λύσουμε το ίδιο πρόβλημα βελτιστοποίησης (5).

2.2 Εκπαίδευση του Νευρωνικού Δικτύου

Θα χρησιμοποιήσουμε τις ίδιες δύο συναρτήσεις κόστους με πριν, δηλαδή την Cross-Entropy και την Exponential, και για τις δύο συναρτήσεις κόστους το step-size είναι $\mu = 0.001$ και η παράμετρος κανονικοποίησης για την μέθοδο **ADAM** $\lambda = 0.01$. Και για τις δύο συναρτήσεις κόστους ο αλγόριθμος έτρεξε για 50 epochs, με παρακολούθηση της τιμής του κόστους ανά epoch. Το Σχ.3 δείχνει πως μεταβάλετε η τιμή της συνάρτησης κόστους Cross-Entropy καθώς ο αλγόριθμος συγκλίνει. Ομοίως για την Exponential στο Σχ. 4

Πίνακας 3: Αποτελέσματα Εκτίμησης Λόγου Πιθανοφάνειας σε Εικόνες

Συνάρτηση Κόστους	$\mathcal{P}(\mathcal{D}_1\|\mathcal{H}_0)$	$\mathcal{P}(\mathcal{D}_0\|\mathcal{H}_1)$	Συνολικό Σφάλμα
Cross-Entropy	0.0040	0.0060	**0.0050**
Exponential	0.0040	0.0050	**0.0045**



(α′) Λάθη Cross-Entropy

(β′) Λάθη Exponential

Σχήμα 5: Περιπτώσεις λήψης λάθος απόφασης

2.3 Εφαρμογή δεδομένων

Ο κώδικας για τα νευρωνικά δίκτυα είναι ο ίδιος με το προηγούμενο ερώτημα και παρατίθεται στα Παραρτήματα Γ′, Δ′, ενώ για την προσομοίωση και τον υπολογισμό των σφαλμάτων, στο Ε′

Αφού ολοκληρωθεί η εκπαίδευση και των δύο νευρωνικών δικτύων, εφαρμόζουμε τα δεδομένα που μας δίνει η **MNIST** για δοκιμή, και όπως και πριν υπολογίζουμε τις πιθανότητες σφαλμάτων. Στον πίνακα 3 παρουσιάζονται τα αποτελέσματα.

Στο Σχ. 5 σε κάθε μία από τις εικόνες (5α′), (5β′) πρώτη γραμμή απεικονίζει εικόνες του αριθμού οκτώ που ερμηνεύτηκαν σαν μηδέν, και η δεύτερη εικόνες του αριθμού μηδέν που ερμηνεύτηκαν σαν οκτώ.

A′ Δημιουργία τυχαίων δειγμάτων

```
import numpy as np
import matplotlib.pyplot as plt
class Bayes:
    def __init__(self, size) -> None:
        self.size = size

    def generateH0Pairs(self):
        # Generate pairs with mean 0 and sigma^2 1

        x1_H0 =  np.random.normal(0, 1, size = self.size)
        x2_H0 =  np.random.normal(0, 1, size = self.size)
        x = np.column_stack((x1_H0, x2_H0))

        return x

    def generateH1Pairs(self):

        x11_H1 = np.random.normal(-1, 1, size = self.size) # Generate data for x1 form both gaussians and choose later based on uniform
        x12_H1 = np.random.normal(1, 1, size = self.size)

        x21_H1 = np.random.normal(-1, 1, size = self.size) # Generate data for x2 form both gaussians and choose later based on uniform
        x22_H1 = np.random.normal(1, 1, size = self.size)

        indices_1 = np.random.uniform(0,1, size = self.size) # Indices array to choose from x11, x12 with prob 0.5
        indices_2 = np.random.uniform(0,1, size = self.size) # Indices array to choose from x21, x22 with prob 0.5

        x_0 = np.where(indices_1 <= 0.5, x11_H1, x12_H1)
        x_1 = np.where(indices_2 <= 0.5, x21_H1, x22_H1)
        x = np.column_stack((x_0,x_1))

        return x

    def gaussianPDF(self, x, mean, sigma):
        g = np.exp(-(x - mean)**2 / (2*sigma**2)) / (sigma * np.sqrt(2*np.pi))
        return g

    def probDensityH0Improoved(self, x):
        mean = 0
        sigma = 1

        f_00 = self.gaussianPDF(x[:, 0], mean, sigma)
        f_01 = self.gaussianPDF(x[:, 1], mean, sigma)

        f_0 = f_00 * f_01
        # returns and array with F(x1,x2)=f(x1)*f(x2) for all x1, x2
        return f_0


    def probDensityH1Improoved(self,x):
        f_00 = self.gaussianPDF(x[:, 0], -1, 1)
        f_01 = self.gaussianPDF(x[:, 0], 1, 1)

        f_10 = self.gaussianPDF(x[:, 1], -1, 1)
        f_11 = self.gaussianPDF(x[:, 1], 1, 1)

        f_0 = 0.5*(f_00 + f_01)
        f_1 = 0.5*(f_10 + f_11)

        return f_0 * f_1 # returns and array with F(x1,x2)=f(x1)*f(x2) for all x1, x2

    def testUnderH0(self, set):
        L = self.probDensityH1Improoved(set) / self.probDensityH0Improoved(set)

        fails = np.sum(L > 1)
        return fails / self.size

    def testUnderH1(self, set):
        L = self.probDensityH1Improoved(set) / self.probDensityH0Improoved(set)

        fails = np.sum(L <= 1)
        return fails / self.size

    def generateH1PairsTraining(self):
        x11_H1T = np.random.normal(-1, 1, size = 200) # Generate data for x1 form both gaussians and choose later based on uniform
        x12_H1T = np.random.normal(1, 1, size = 200)

        x21_H1T = np.random.normal(-1, 1, size = 200) # Generate data for x2 form both gaussians and choose later based on uniform
        x22_H1T = np.random.normal(1, 1, size = 200)

        indices_1T = np.random.uniform(0,1, size = 200) # Indices array to choose from x11, x12 with prob 0.5
        indices_2T = np.random.uniform(0,1, size = 200) # Indices array to choose from x21, x22 with prob 0.5

        x_0T = np.where(indices_1T <= 0.5, x11_H1T, x12_H1T)
        x_1T = np.where(indices_2T <= 0.5, x21_H1T, x22_H1T)
        xT = np.column_stack((x_0T,x_1T))

        return xT

    def generateH0PairsTraining(self):
        x1_H0T =  np.random.normal(0, 1, size = 200)
        x2_H0T =  np.random.normal(0, 1, size = 200)
        xT0 = np.column_stack((x1_H0T, x2_H0T))
        return xT0s
```

Β′ Προσομοίωση Bayes τεστ και νευρωνικού δικτύου

```
from PIL import Image
import numpy as np
from crossEntropy import CrossEntropyNN
from exponential import ExponentialNN
import subprocess
import matplotlib.pyplot as plt
from main1 import Bayes

def plot(x, y, xlabel, ylabel, title):

    plt.figure(figsize=(15,6))
    plt.style.use('seaborn-v0_8-deep')
    plt.plot(x, y, linewidth = 2)
    plt.xlabel(xlabel, fontsize=14, fontweight='bold', color='#555')
    plt.ylabel(ylabel, fontsize=14, fontweight='bold', color='#555')
    plt.xticks(fontsize=12, color='#444')
    plt.yticks(fontsize=12, color='#444')
    plt.show()

def testingH0(data, nn1, nn2):
    failsCE = 0
    failsExp= 0
    wrongCE = []
    wrongExp = []
    for x in data:
        f1 = nn1.u(x)
        f2 = nn2.u(x)
        if f1 <= 1/2:
            failsCE += 1
            wrongCE.append(x)
        if f2 <= 0:
            failsExp +=1
            wrongExp.append(x)

    print("\nChose H0 Instead of H1 CE:", 100 * failsCE / 1000000, "LLR:", 100 * failsExp / 1000000)
def testingH1(data, nn1, nn2):
    failsCE = 0
    failsExp= 0
    wrongCE = []
    wrongExp = []
    for x in data:
        f1 = nn1.u(x)
        f2 = nn2.u(x)
        if f1 >= 1/2:
            failsCE += 1
            wrongCE.append(x)
        if f2 >= 0:
            failsExp +=1
            wrongExp.append(x)
    print("\nChose H1 Instead of H0 CE:", 100 * failsCE / 1000000, "LLR:", 100 * failsExp / 1000000)


def main():
    b = Bayes(1000000)

    # #Generating Random Pairs
    x_H0 = b.generateH0Pairs()
    x_H1 = b.generateH1Pairs()

    e1 = b.testUnderH0(x_H0)

    e2 = b.testUnderH1(x_H1)

    x_H0Train = b.generateH0PairsTraining().reshape(200,2,1)
    x_H1Train = b.generateH1PairsTraining().reshape(200,2,1)

    ceNN = CrossEntropyNN(2, 20, 1, 10000)
    expNN = ExponentialNN(2, 20, 1, 10000)

    subprocess.run("clear")
    # * Run This Only For Training
    # ceNN.train(x_H0Train, x_H1Train, 0.001, 0.01)
    # ceNN.storeParameters()
    # expNN.train(x_H0Train, x_H1Train, 0.001, 0.01)
    # expNN.storeParameters()

    # * Assume training has already been done
    ceNN.loadParameters()
    expNN.loadParameters()

    # plot(np.arange(len(ceNN.costEpochArr)), ceNN.costEpochArr, "Epochs", "Cost Function Value", "Cross Entropy")
    # plot(np.arange(len(expNN.costEpochArr)), expNN.costEpochArr, "Epochs", "Cost Funtction Value", "Exponential")

    testingH1(x_H0.reshape(1000000,2,1),ceNN, expNN)
    print("Chose H1 instead of H0 (Bayes)", str(e1 *100),"%")
    testingH0(x_H1.reshape(1000000,2,1),ceNN, expNN)
    print("Chose H0 instead of H1 (Bayes)", str(e2 *100),"%")
if __name__ == "__main__":
    main()
```

Γ′ Νευρωνικό δίκτιο με Cross-Entropy Cost

```
import numpy as np
from tqdm import tqdm

class CrossEntropyNN():
    def __init__(self, numberOfInputs, sizeOfHiddenLayer, numberOfOutputs, epochs):
        self.numberOfInputs = numberOfInputs
        self.sizeOfHiddenLayer = sizeOfHiddenLayer
        self.numberOfOutputs = numberOfOutputs
        self.epochs = epochs
        np.random.seed(1)
        stdDevHl = np.sqrt(1 / (self.sizeOfHiddenLayer + self.numberOfInputs))
        stdDevOl = np.sqrt(1 / (sizeOfHiddenLayer + numberOfOutputs))

        initialWeightsHl = np.random.normal(loc = 0, scale = stdDevHl, size = (self.sizeOfHiddenLayer, self.numberOfInputs) ) #* Initial Weights Hidden Layer
        initialWeightsOl = np.random.normal(loc = 0, scale = stdDevOl, size = (self.numberOfOutputs, self.sizeOfHiddenLayer)) #* Initial Weights Output Layer

        self.hlWeights = initialWeightsHl.copy() #* Hidden Layer Weights Cross Entropy
        self.hlBiases = np.zeros((self.sizeOfHiddenLayer, 1)) #* HL Biases Cross Entropy

        self.olWeights = initialWeightsOl.copy() #* Output Layer Cross Ent4ropy
        self.olBias = 0
        self.costEpochArr = []

    def gradient(self, x):
        w1 = self.hlWeights @ x + self.hlBiases
        z1 = np.maximum(0, w1) #? RELU ACTIVATION
        w2 = self.olWeights @ z1 + self.olBias

        y = 1 / (1 + np.exp(-w2)) #? SIGMOID ACTIVATION
        v2 = y * (1 - y) #? Sigmoid Rerivative Based On Output y
        u1 = self.olWeights.T @ v2
        v1 = u1 * np.where(w1 > 0, 1, 0)

        dA2 = v2 @ z1.T
        dB2 = v2
        dA1 = v1 @ x.T
        dB1 = v1

        fin = np.array([dA1, dB1, dA2, dB2], dtype =object), y.item()
        return fin

    def train(self, x_H0, x_H1, m, l):
        print("Started Training With Method Cross Entropy, Step Size:", m, "Normalization:", l)
        lenX = len(x_H0)
        costEpoch = 0
        iterations = self.epochs * lenX

        index = 0
        epoch = 0

        gradsX1, x1 = self.gradient(x_H0[0])
        gradsX2, x2 = self.gradient(x_H1[0])

        phiDev = 1 / (1 - x1)
        psiDev = -1 / x2

        px = ((phiDev*gradsX1 + psiDev*gradsX2) ** 2)

        c = 10**(-8)

        pbar = tqdm(range(iterations), colour='blue', position=0, desc=f"CrossEntropy, Epoch {epoch}, Cost {costEpoch}")
        for i in pbar:
            index +=1
            if index == (lenX):
                index = 0
                c = costEpoch / lenX
                self.costEpochArr.append(c)
                epoch +=1
                pbar.set_description(f"CrossEntropy, Epoch: {epoch} Cost: {c:.9f}")
                costEpoch = 0

            self.hlWeights -= m * (phiDev * gradsX1[0] + psiDev * gradsX2[0]) / np.sqrt(c + px[0])
            self.hlBiases  -= m * (phiDev * gradsX1[1] + psiDev * gradsX2[1]) / np.sqrt(c + px[1])
            self.olWeights -= m * (phiDev * gradsX1[2] + psiDev * gradsX2[2]) / np.sqrt(c + px[2])
            self.olBias    -= m * (phiDev * gradsX1[3] + psiDev * gradsX2[3]) / np.sqrt(c + px[3])

            cost = (-np.log(1-x1) - np.log(x2))
            costEpoch +=cost

            gradsX1, x1 = self.gradient(x_H0[index])
            gradsX2, x2 = self.gradient(x_H1[index])

            phiDev = 1 / (1 - x1)
            psiDev = -1 / x2

            px = (1 - l) * px + l * ((phiDev * gradsX1 + psiDev * gradsX2) ** 2)

    def storeParameters(self):
        costEpochArr = np.array(self.costEpochArr)
        np.savez("nn_cross_entropy_parameters_"+str(self.epochs)+".npz", hlw = self.hlWeights, hlb = self.hlBiases, olw = self.olWeights, olb = self.olBias, costEpoch = costEpochAr

    def loadParameters(self):
        data = np.load("nn_cross_entropy_parameters_"+str(self.epochs)+".npz")
```

```
        self.hlWeights = data["hlw"]
        self.hlBiases = data["hlb"]
        self.olWeights = data["olw"]
        self.olBias = data["olb"]
        self.costEpochArr = data["costEpoch"]

    def u(self, x):
        w1 = self.hlWeights @ x + self.hlBiases
        z1 = np.maximum(0, w1) #? RELU ACTIVATION
        w2 = self.olWeights @ z1 + self.olBias
        y = 1 / (1 + np.exp(-w2)) #? SIGMOID ACTIVATION

        return y.item()
```

Δ′ Νευρωνικό δίκτυο με Exponential Cost

```
import numpy as np
from tqdm import tqdm

class ExponentialNN():
    def __init__(self, numberOfInputs, sizeOfHiddenLayer, numberOfOutputs, epochs):
        self.numberOfInputs = numberOfInputs
        self.sizeOfHiddenLayer = sizeOfHiddenLayer
        self.numberOfOutputs = numberOfOutputs
        self.epochs = epochs
        np.random.seed(1)
        stdDevHl = np.sqrt(1 / (self.sizeOfHiddenLayer + self.numberOfInputs))
        stdDevOl = np.sqrt(1 / (sizeOfHiddenLayer + numberOfOutputs))
        initialWeightsHl = np.random.normal(loc = 0, scale = stdDevHl, size = (self.sizeOfHiddenLayer, self.numberOfInputs) ) #* Initial Weights Hidden Layer
        initialWeightsOl = np.random.normal(loc = 0, scale = stdDevOl, size = (self.numberOfOutputs, self.sizeOfHiddenLayer)) #* Initial Weights Output Layer

        self.hlWeights = initialWeightsHl.copy() #* ------------------ Exponential
        self.hlBiases = np.zeros((self.sizeOfHiddenLayer, 1)) #* HL Biases Exponential

        self.olWeights = initialWeightsOl.copy() #* Output Layer Exponential
        self.olBias = 0

        self.costEpochArr = []

    def gradient(self, x):
        w1 = self.hlWeights @ x + self.hlBiases
        z1 = np.maximum(0, w1) #? RELU ACTIVATION
        w2 = self.olWeights @ z1 + self.olBias

        y = w2 #? No Nonlinearity Needed Here
        v2 = np.array([1]).reshape(1,1) #? Derivative
        u1 = self.olWeights.T @ v2
        v1 = u1 * np.where(w1 > 0, 1, 0)

        dA2 = v2 @ z1.T
        dB2 = v2
        dA1 = v1 @ x.T
        dB1 = v1

        fin = np.array([dA1, dB1, dA2, dB2], dtype =object), y.item()
        return fin

    def train(self, x_H0, x_H1, m, l):
        print("Started Training With Method Exponential, Step Size:", m, "Normalization:", l)
        lenX = len(x_H0)
        costEpoch = 0
        iterations = self. epochs * lenX

        index = 0
        epoch = 0

        gradsX1, x1 = self.gradient(x_H0[0])
        gradsX2, x2 = self.gradient(x_H1[0])

        phiDev = 0.5 * np.exp(0.5*x1)
        psiDev = -0.5 * np.exp(-0.5*x2)

        px = ((phiDev*gradsX1 + psiDev*gradsX2) ** 2)

        c = 10**(-8)

        pbar = tqdm(range(iterations), colour='blue', position=1, desc=f"Exponential, Epoch {epoch}, Cost {costEpoch}")
        for _ in pbar:
            index +=1

            if index == lenX:
                index = 0
                c = costEpoch / lenX
                self.costEpochArr.append(c)
                epoch +=1
                pbar.set_description(f"Exponential, Epoch: {epoch}, Cost: {c:.9f}")
                costEpoch = 0

            self.hlWeights -= m * (phiDev * gradsX1[0] + psiDev * gradsX2[0]) / np.sqrt(c + px[0])
            self.hlBiases  -= m * (phiDev * gradsX1[1] + psiDev * gradsX2[1]) / np.sqrt(c + px[1])
            self.olWeights -= m * (phiDev * gradsX1[2] + psiDev * gradsX2[2]) / np.sqrt(c + px[2])
```

```
            self.olBias     -= m * (phiDev * gradsX1[3] + psiDev * gradsX2[3]) / np.sqrt(c + px[3])

            cost = (2 * phiDev - 2 * psiDev)
            costEpoch += cost

            gradsX1, x1 = self.gradient(x_H0[index])
            gradsX2, x2 = self.gradient(x_H1[index])

            phiDev = 0.5 * np.exp(0.5*x1)
            psiDev = -0.5 * np.exp(-0.5*x2)
            px = (1 - l) * px + l * ((phiDev * gradsX1 + psiDev * gradsX2) ** 2)

    def storeParameters(self):
        costEpochArr = np.array(self.costEpochArr)
        np.savez("nn_exponential_parameters_"+str(self.epochs)+".npz", hlw = self.hlWeights, hlb = self.hlBiases, olw = self.olWeights, olb = self.olBias, costEpoch = costEpochArr)

    def loadParameters(self):
        data = np.load("nn_exponential_parameters_"+str(self.epochs)+".npz")

        self.hlWeights = data["hlw"]
        self.hlBiases  = data["hlb"]
        self.olWeights = data["olw"]
        self.olBias    = data["olb"]
        self.costEpochArr = data["costEpoch"]

    def u(self, x):
        w1 = self.hlWeights @ x + self.hlBiases
        z1 = np.maximum(0, w1) #? RELU ACTIVATION
        w2 = self.olWeights @ z1 + self.olBias
        return w2.item()
```

Ε′ Δοκιμή νευρωνικού δικτύου με εικόνες

```
from PIL import Image
import os
import numpy as np
from crossEntropy import CrossEntropyNN
from exponential import ExponentialNN
import subprocess
import matplotlib.pyplot as plt

def loadImages():
    zeros = []
    eights = []
    zerostest = []
    eightstest = []

    zerosTrainingDirectory = "MNIST/training/0"
    eightsTrainingDirectory = "MNIST/training/8"
    zerosTestingDirectory = "MNIST/testing/0"
    eightsTestingDirectory = "MNIST/testing/8"

    for filename in os.listdir(zerosTrainingDirectory):
        imgPath = os.path.join(zerosTrainingDirectory, filename)
        with Image.open(imgPath) as img:
            imgA  = np.array(img).astype('float32') / 255.0
            imgA = imgA.flatten().reshape(784,1)
            zeros.append(imgA)

    for filename in os.listdir(eightsTrainingDirectory):
        imgPath = os.path.join(eightsTrainingDirectory, filename)
        with Image.open(imgPath) as img:
            imgA  = np.array(img).astype('float32') / 255.0
            imgA = imgA.flatten().reshape(784,1)
            eights.append(imgA)
    for filename in os.listdir(zerosTestingDirectory):
        imgPath = os.path.join(zerosTestingDirectory, filename)
        with Image.open(imgPath) as img:
            imgA  = np.array(img).astype('float32')/ 255.0
            imgA = imgA.flatten().reshape(784,1)
            zerostest.append(imgA)

    for filename in os.listdir(eightsTestingDirectory):
        imgPath = os.path.join(eightsTestingDirectory, filename)
        with Image.open(imgPath) as img:
            imgA  = np.array(img).astype('float32') / 255.0
            imgA = imgA.flatten().reshape(784,1)
            eightstest.append(imgA)

    zerosTrain = np.array(zeros[:5000])
    eightsTrain = np.array(eights[:5000])

    zerosTest = np.array(zerostest[:970])
    eightsTest = np.array(eightstest[:970])

    return [zerosTrain, eightsTrain, zerosTest, eightsTest]

def plot(x, y, xlabel, ylabel, title):

    plt.figure(figsize=(8,6))
    plt.style.use('seaborn-v0_8-deep')
    plt.plot(x, y, linewidth = 2)
```

```python
    plt.xlabel(xlabel, fontsize=14, fontweight='bold', color='#555')
    plt.ylabel(ylabel, fontsize=14, fontweight='bold', color='#555')
    plt.xticks(fontsize=12, color='#444')
    plt.yticks(fontsize=12, color='#444')
    plt.show()

def testing(zeros, eights, nn1, nn2):
    failsCE = 0
    failsExp= 0
    wrongCEH0 = []
    wrongExpH0 = []
    for x in eights:
        f1 = nn1.u(x)
        f2 = nn2.u(x)
        if f1 <= 1/2:
            failsCE += 1
            wrongCEH0.append(x)
        if f2 <= 0:
            failsExp +=1
            wrongExpH0.append(x)
    print("\nChose H0 Instead of H1 CE:", 100 * failsCE / 997, "LLR:", 100 * failsExp / 997)
    failsCE = 0
    failsExp= 0
    wrongCEH1 = []
    wrongExpH1 = []

    for x in zeros:
        f1 = nn1.u(x)
        f2 = nn2.u(x)

        if f1 > 1/2:
            failsCE += 1
            wrongCEH1.append(x)
        if f2 > 0:
            failsExp +=1
            wrongExpH1.append(x)

    fig, axs = plt.subplots(2, 2, figsize=(15,5))
    axs = axs.flatten()

    for i, image in enumerate(wrongCEH0[:2]):
        axs[i].imshow(image.reshape(28, 28), cmap='gray')
        axs[i].axis('off')  # Turn off axis

    for i, image in enumerate(wrongCEH1[:2], start=2):
        axs[i].imshow(image.reshape(28, 28), cmap='gray')
        axs[i].axis('off')  # Turn off axis

    fig, axs = plt.subplots(2, 2, figsize=(15,5))  # Number of columns based on length of images list
    axs = axs.flatten()

    for i, image in enumerate(wrongExpH0[:2]):
        axs[i].imshow(image.reshape(28, 28), cmap='gray')
        axs[i].axis('off')  # Turn off axis

    for i, image in enumerate(wrongExpH1[:2], start=2):
        axs[i].imshow(image.reshape(28, 28), cmap='gray')
        axs[i].axis('off')  # Turn off axis

    plt.tight_layout()
    plt.show()

    print("\nChose H1 Instead of H0 CE:", 100 * failsCE / 997, "LLR:", 100 * failsExp / 997)


def main():
    data = loadImages()

    ceNN = CrossEntropyNN(784, 300, 1, 50)
    expNN = ExponentialNN(784, 300, 1, 50)

    subprocess.run("clear")

    # * Run This Only For Training
    #ceNN.trainCE(data[0], data[1], 0.001, 0.01)
    #ceNN.storeParameters()
    #expNN.trainCE(data[0], data[1], 0.001, 0.01)
    #expNN.storeParameters()

    # * Assume training has already been done
    ceNN.loadParameters()
    expNN.loadParameters()
    # print(ceNN.costEpochArr)
    # plot(np.arange(50), ceNN.costEpochArr, "Epochs", "Cost Value", "Cross Entropy")
    # plot(np.arange(50), expNN.costEpochArr, "Epochs", "Cost Value", "Exponential")

    testing(data[2], data[3], ceNN, expNN)
if __name__ == "__main__":
    main()
```

Αναφορές

[1] G.V. Moustakides, K. Basioti, Training neural networks for likelihood/density ratio estimation, arXiv: 1911. 00405, Nov. 2019.

[2] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Mathematics of Control, Signals and Systems, vol. 2, no. 4, pp. 303–314, 1989.