Νευρωνικά Δίκτυα - Βαθιά Μάθηση
Πρώτη Εργασία

4 Απριλίου 2024

Δημήτριος Αλεξόπουλος
aadimitri@ece.auth.gr
ΑΕΜ 10091

Περιεχόμενα

Κατάλογος Σχημάτων

1 Εισαγωγή

Σκοπός της παρούσας εργασίας είναι η υλοποίηση ενός νευρωνικού δικτύου πολυστρωματικού perceptron (επιλέχθηκε πλήρως συνδεδεμένο δίκτυο) που θα εκπαιδεύεται με τον αλγόριθμο back-propagation. Το αυτό θα εκπαιδευτεί σε ένα dataset, ωστόσο η υλοποίησή του θα είναι σε θέση να επιλύσει οποιοδήποτε πρόβλημα κατηγοριοποίησης πολλών κλάσεων.
Για τους σκοπούς της εργασίας επιλέγεται η βάση δεδομένων $CIFAR-10$, η οποία αποτελείται από 60000 έγχρωμες εικόνες 32×32 σε 10 κλάσεις, με 6000 εικόνες ανά κλάση. Υπάρχουν 50000 εικόνες εκπαίδευσης και 10000 εικόνες δοκιμής.
Το σύνολο δεδομένων χωρίζεται σε πέντε παρτίδες εκπαίδευσης και μία παρτίδα δοκιμής, κάθε μία με 10000 εικόνες. Η παρτίδα δοκιμής περιέχει ακριβώς 1000 τυχαία επιλεγμένες εικόνες από κάθε κλάση. Οι παρτίδες εκπαίδευσης περιέχουν τις υπόλοιπες εικόνες με τυχαία σειρά, αλλά ορισμένες παρτίδες εκπαίδευσης μπορεί να περιέχουν περισσότερες εικόνες από μια κλάση από ό,τι από μια άλλη. Μεταξύ τους, οι παρτίδες εκπαίδευσης περιέχουν ακριβώς 5000 εικόνες από κάθε κλάση.
Παρακάτω θα υλοποιήσουμε from scratch σε python το νευρωνικό δίκτυο, παρουσιάζοντας τόσο το θεωρητικό υπόβαθρο, όσο και τα αποτελέσματα της υλοποίησης.

2 Υλοποίηση

Θα περιγράψουμε, αρχικά, το μαθηματικό υπόβαθρο της υλοποίησης ενός νευρωνικού δικτύου. Παράλληλα με την θεωρητική ερμηνεία θα υπάρχει κι επεξήγηση του υλοποιημένου κώδικα σε python.

2.1 **Layer**

Το βασικό συστατικό μέλος ενός νευρωνικού δικτύου είναι ένα στρώμα (layer), που πρακτικά είναι μια συνάρτηση με εισόδους κι εξόδους.

$$X \rightarrow \boxed{\text{layer}} \rightarrow Y$$

Σχήμα 1: Δομή ενός layer

Ας δούμε, πρώτα, ποιά είναι η συλλογιστική πορεία της υλοποίησης του δικτύου:

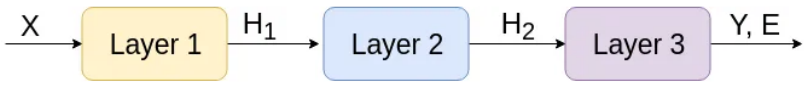
1. Τροφοδοτούμε δεδομένα εισόδου στο νευρωνικό δίκτυο.
2. Τα δεδομένα ρέουν από στρώμα σε στρώμα μέχρι να έχουμε την έξοδο.
3. Μόλις έχουμε την έξοδο, μπορούμε να υπολογίσουμε το σφάλμα το οποίο είναι ένα βαθμωτό μέγεθος.
4. Τέλος, μπορούμε να προσαρμόσουμε μια δεδομένη παράμετρο (weight ή bias) αφαιρώντας την παράγωγο του σφάλματος ως προς την ίδια την παράμετρο.

5. Επαναλαμβάνουμε αυτή τη διαδικασία για κάποιες έποχές, έως ότου έχουμε ικανοποιητικό αποτέλεσμα.

Το πιο σημαντικό βήμα είναι το 4ο. Θέλουμε να μπορούμε να έχουμε όσα layers θέλουμε και οποιουδήποτε τύπου (πχ. πλήρως συνδεδεμένα, συνελικτικά κλπ.) . Αλλά αν τροποποιήσουμε/προσθέσουμε/αφαιρέσουμε ένα layer από το δίκτυο, η έξοδος του δικτύου θα αλλάξει, πράγμα που θα αλλάξει το σφάλμα, πράγμα που θα αλλάξει την παράγωγο του σφάλματος σε σχέση με τις παραμέτρους. Πρέπει, λοιπόν, να είμαστε σε θέση να υπολογίζουμε τις παραγώγους ανεξάρτητα από την αρχιτεκτονική του δικτύου, ανεξάρτητα από τις συναρτήσεις ενεργοποίησης, ανεξάρτητα από την συνάρτηση σφάλματος που χρησιμοποιούμε.
Για να το πετύχουμε αυτό, πρέπει να υλοποιήσουμε κάθε layer ξεχωριστά.

2.1.1 **Forward Propagation**

Βασικό χαρακτηριστικό του νευρωνικού δικτύου είναι ότι η έξοδος ενός layer είναι η είσοδος του επόμενου. Αυτό ονομάζεται forward propagation.



Σχήμα 2: Forward Propagation

Ουσιαστικά, δίνουμε τα δεδομένα εισόδου στο πρώτο layer, στη συνέχεια η έξοδος κάθε layer γίνεται είσοδος του επόμενου layer μέχρι να φτάσουμε στο τέλος του δικτύου. Συγκρίνοντας το αποτέλεσμα του δικτύου (Y) με την επιθυμητή έξοδο (ας πούμε $Y*$), μπορούμε να υπολογίσουμε το $error\ E$. Ο στόχος είναι να ελαχιστοποιήσουμε αυτό το σφάλμα αλλάζοντας τις παραμέτρους στο δίκτυο. Αυτό ονομάζεται backward propagation.

2.1.2 **Gradient Descent**

Η μέθοδος κλίσης είναι ένας αλγόριθμος βελτιστοποίησης που θα χρησιμοποιήσουμε για την εύρεση των συντελεστών του νευρωνικού δικτύου.
Λειτουργεί βάζοντας το μοντέλο να κάνει προβλέψεις σε δεδομένα εκπαίδευσης και χρησιμοποιώντας το σφάλμα στις προβλέψεις για να ενημερωθεί το μοντέλο με τέτοιο τρόπο ώστε να μειωθεί το σφάλμα.
Στόχος του αλγορίθμου είναι να βρει τις παραμέτρους του μοντέλου (π.χ. weight ή biases) που ελαχιστοποιούν το σφάλμα του μοντέλου στο σύνολο δεδομένων εκπαίδευσης. Βασικά, θέλουμε να αλλάξουμε κάποια παράμετρο στο δίκτυο (έστω w), έτσι ώστε το συνολικό σφάλμα E να μειώνεται. Ακολουθούμε τον παρακάτω κανόνα:

$$w \longleftarrow w - \alpha \frac{\partial E}{\partial w}$$

Όπου α είναι μια παράμετρος στο εύρος $[0, 1]$ που θέτουμε και η οποία ονομάζεται ρυθμός μάθησης (learning rate). Το σημαντικό εδώ είναι το $\frac{\partial E}{\partial w}$ (δηλ. η παράγωγος του E ως προς το w). Πρέπει να είμαστε σε θέση να βρούμε την τιμή αυτής της έκφρασης για οποιαδήποτε παράμετρο του δικτύου ανεξάρτητα από την αρχιτεκτονική του.

2.1.3 **Mini-Batch Gradient Descent**

Ο αλγόριθμος Mini-Batch Gradient Descent είναι μια παραλλαγή του αλγορίθμου Gradient Descent που χωρίζει το σύνολο δεδομένων εκπαίδευσης σε μικρές παρτίδες, οι οποίες χρησιμοποιούνται για τον υπολογισμό του σφάλματος του μοντέλου και την ενημέρωση των συντελεστών του μοντέλου.
Πρακτικά, υπολογίζουμε τον μέσο όρο των gradients που προκύπτουν από το forward propagation μιας παρτίδας δεδομένων εκπαίδευσης κι έπειτα ανανεώνουμε τις παραμέτρους (weights, biases) του δικτύου με βάση αυτό (backward propagation)
Η μέθοδος αυτή προσπαθεί να βρει μια ισορροπία μεταξύ της ευρωστίας της απλής στοχαστικής μεθόδου κλίσης και της αποδοτικότητας της διαδικασίας του Mini-Batch.

2.1.4 **Backward Propagation**

Ας υποθέσουμε ότι δίνουμε σε ένα layer την παράγωγο του σφάλματος σε σχέση με την έξοδό του ($\frac{\partial E}{\partial Y}$), τότε πρέπει να είναι σε θέση να παρέχει την παράγωγο του σφάλματος σε σχέση με την είσοδό του ($\frac{\partial E}{\partial X}$):

$$\frac{\partial E}{\partial X} \leftarrow \boxed{\text{layer}} \leftarrow \frac{\partial E}{\partial Y}$$

Σχήμα 3: Backward Propagation

Μπορούμε να γράψουμε:

$$\frac{\partial E}{\partial X} = \begin{bmatrix} \frac{\partial E}{\partial x_1} & \frac{\partial E}{\partial x_2} & \cdots & \frac{\partial E}{\partial x_i} \end{bmatrix}$$
$$\frac{\partial E}{\partial Y} = \begin{bmatrix} \frac{\partial E}{\partial y_1} & \frac{\partial E}{\partial y_2} & \cdots & \frac{\partial E}{\partial y_i} \end{bmatrix}$$

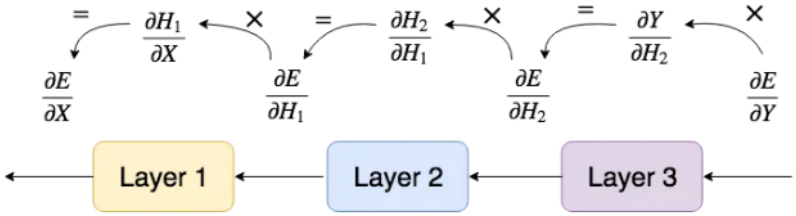
Μπορούμε να παρατηρήσουμε ότι αν έχουμε πρόσβαση στο $\frac{\partial E}{\partial Y}$ μπορούμε πολύ εύκολα να υπολογίσουμε το $\frac{\partial E}{\partial W}$ χωρίς να γνωρίζουμε τίποτα για την αρχιτεκτονική του δικτύου. Απλά χρησιμοποιούμε τον κανόνα της αλυσίδας:

$$\frac{\partial E}{\partial W} = \sum_j \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial w}$$

Ο άγνωστος είναι το $\frac{\partial y_j}{\partial w}$, το οποίο εξαρτάται πλήρως από τον τρόπο με τον οποίο το layer υπολογίζει την έξοδό του. Έτσι, αν κάθε στρώμα έχει πρόσβαση στο $\frac{\partial E}{\partial Y}$, όπου Y είναι η δική του έξοδος, τότε μπορούμε να ενημερώσουμε τις παραμέτρους μας.
Η έξοδος ενός layer είναι η είσοδος του επόμενου που σημαίνει ότι $\frac{\partial E}{\partial X}$ για ένα στρώμα είναι $\frac{\partial E}{\partial U}$ για το προηγούμενο στρώμα. Και πάλι, μπορούμε να χρησιμοποιήσουμε τον κανόνα της αλυσίδας:

$$\frac{\partial E}{\partial x_i} = \sum_j \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

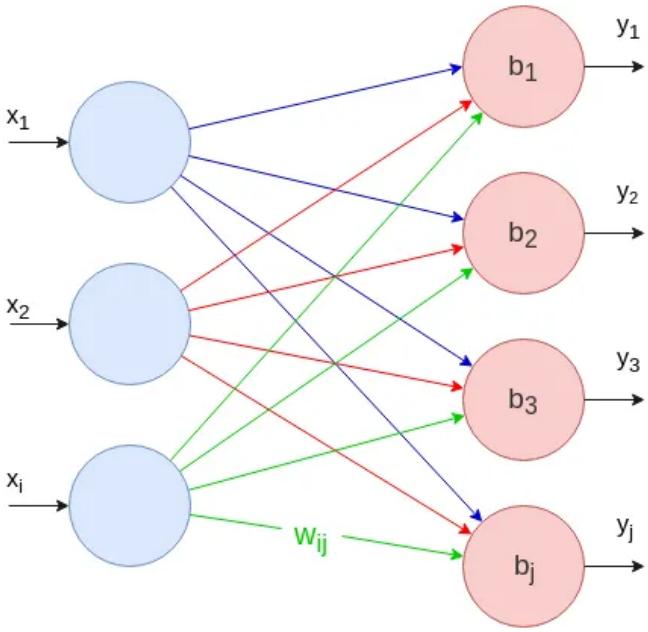
Συνοπτικά, λοιπόν, η μέθοδος Backward Propagation απεικονίζεται στο παρακάτω σχήμα:



Σχήμα 4: Συνοπτικό Backward Propagation

2.2 Fully Connected Layer

Θα ορίσουμε και θα υλοποιήσουμε ένα πλήρως συνδεδεμένο στρώμα ή Fully Connected Layer - FC. Σε ένα στρώμα FC κάθε νευρώνας εισόδου συνδέεται με κάθε νευρώνα εξόδου:



Σχήμα 5: Fully Connected Layer

2.2.1 Forward Propagation

Η τιμή κάθε νευρώνα εξόδου μπορεί να υπολογιστεί ως εξής:

$$y_j = b_j + \sum_i x_i w_{ij}$$

ή πιο απλά με πίνακες:

$$X = \begin{bmatrix} x_1 & \dots & x_i \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & \dots & w_{1j} \\ \vdots & \ddots & \vdots \\ w_{i1} & \dots & w_{ij} \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & \dots & b_j \end{bmatrix}$$

έχουμε:

$$\boxed{Y = XW + B}$$

2.2.2 **Backward Propagation**

Όπως είπαμε, αν υποθέσουμε ότι έχουμε έναν πίνακα που περιέχει την παράγωγο του σφάλματος σε σχέση με την έξοδο αυτού του στρώματος, τότε για την ανανέωση των παραμέτρων του δικτύου χρειαζόμαστε :

1. Την παράγωγο του σφάλματος ως προς τις παραμέτρους ($\frac{\partial E}{\partial W}$, $\frac{\partial E}{\partial B}$)
2. Την παράγωγο του σφάλματος ως προς την είσοδο ($\frac{\partial E}{\partial X}$)

Εφαρμόζοντας και πάλι τον κανόνα της αλυσίδας εύκολα μπορούμε να καταλήξουμε στις παρακάτω εξισώσεις για την ανανέωση των παραμέτρων:

$$\boxed{\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} W^T}$$

$$\boxed{\frac{\partial E}{\partial W} = X^T \frac{\partial E}{\partial Y}}$$

$$\boxed{\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y}}$$

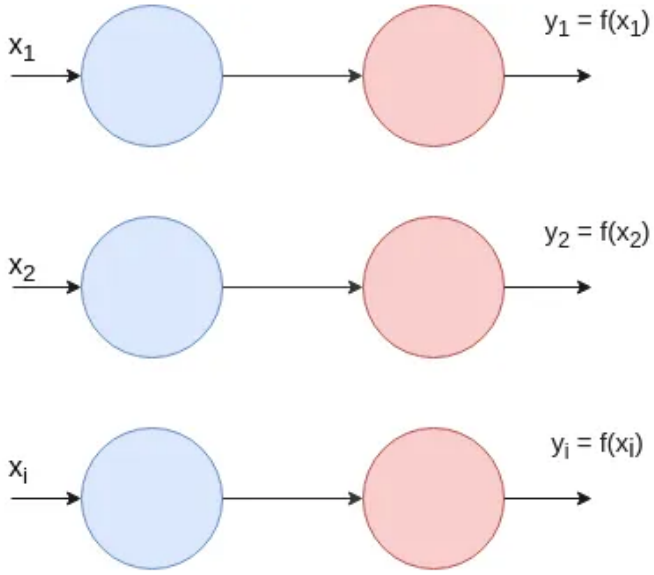
2.3 **Activation Layer**

Όλοι οι υπολογισμοί που κάναμε μέχρι τώρα ήταν εντελώς γραμμικοί, οπότε θα εισάγουμε μη γραμμικότητα στο μοντέλο εφαρμόζοντας μη γραμμικές συναρτήσεις στην έξοδο ορισμένων επιπέδων. Αυτό γίνεται με τις λεγόμενες συναρτήσεις ενεργοποίησης. Έστω f και f' η συνάρτηση ενεργοποίησης και η παράγωγός της αντίστοιχα.

2.3.1 **Forward Propagation**

Για μια δεδομένη είσοδο X, η έξοδος είναι απλά η συνάρτηση ενεργοποίησης που εφαρμόζεται σε κάθε στοιχείο του X. Πράγμα που σημαίνει ότι η είσοδος και η έξοδος έχουν τις ίδιες διαστάσεις:

$$Y = \begin{bmatrix} f(x_1) & \dots & f(x_i) \end{bmatrix} = f(X)$$



Σχήμα 6: Συνάρτηση Ενεργοποίησης

2.3.2 **Backward Propagation**

Για την αντίστροφη διαδικασία έχουμε:

$$\boxed{\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} \cdot f'(X)}$$

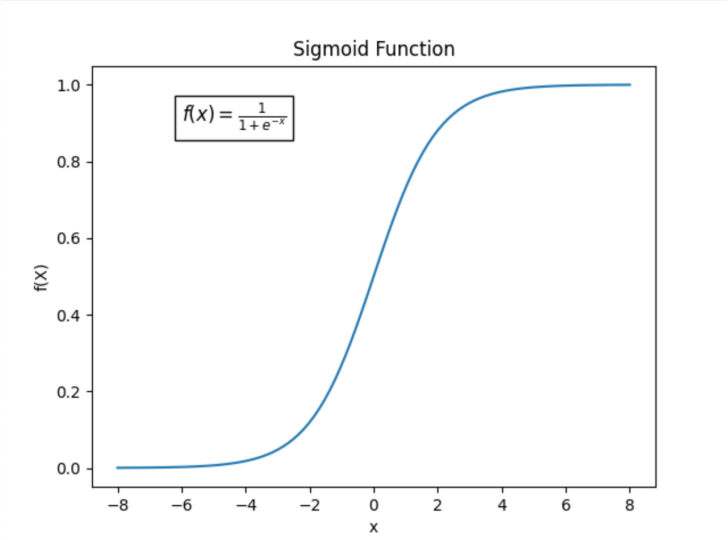
όπου προσέχουμε ότι χρησιμοποιούμε **element-wise** πολλαπλασιαμό μεταξύ των στοιχείων των δύο πινάκων.

2.3.3 Συναρτήσεις Ενεργοποίησης

Στην πράξη χρησιμοποιούνται διάφορες μη-γραμμικές συναρτήσεις ενεργοποίησης. Μερικές από αυτές είναι:

- $tanh()$
- $sigmoid()$
- $ReLu()$
- $LeLeLu()$
- $Softmax()$

Για τους σκοπούς της εργασίας δοκιμάστηκαν οι παραπάνω συναρτήσεις ενεργοποίησης κι επιλέχθηκε η συνάρτηση $Sigmoid()$:



Σχήμα 7: Sigmoid Function

2.4 Loss Function

Το σφάλμα του νευρωνικού δικτύου, το οποίο μετράει πόσο καλά ή άσχημα τα πήγε το δίκτυο για δεδομένα εισόδου, ορίζεται από εμάς. Υπάρχουν πολλοί τρόποι για να οριστεί το σφάλμα, αλλά για τους σκοπούς της εργασίας επιλέγεται το MSE - Mean Squared Error (Μέσο Τετραγωνικό Σφάλμα):

$$\boxed{E = \frac{1}{n}\sum_{i}^{n}(y_i^* - y_i)^2}$$

όπου $y*$ και y δηλώνουν την επιθυμητή έξοδο και την πραγματική έξοδο αντίστοιχα. Αυτό που χρειαζόμαστε τώρα, όπως και για κάθε άλλο επίπεδο, είναι να ορίσουμε το $\frac{\partial E}{\partial Y}$:

$$\boxed{\frac{\partial E}{\partial Y} = \frac{2}{n}(Y - Y^*)}$$

2.5 Network

Μπορούμε, πλέον, εύκολα να συνδυάσουμε τα παραπάνω στρώματα και να διαμορφώσουμε το νευρωνικό μας δίκτυο. Η είσοδος του δικτύου θα έχει όσους νευρώνες όσος είναι κι ο αριθμός των τιμών των πίξελς μιας εικόνας του συνόλου δεδομένων. Προσθέτουμε έπειτα ένα ή περισσότερα hidden layers, το οποίο έχει σημαντικά μικρότερο αριθμό νευρώνων κι έξοδο 10 νευρώνες, όσες δηλαδή είναι και οι διαθέσιμες κλάσεις για την κατηγοριοποίηση του συνόλου δεδομένων μας. Ο νευρώνας του τελευταίου στρώματος που έχει την μεγαλύτερη τιμή μετά από την συνάρτηση ενεργοποίησης θα μας δώσει την κλάση στην οποία θα πρέπει να ανήκει η εικόνας μας. Παρακάτω ακολουθούν δύο σημαντικές συναρτήσεις για την υλοποίηση του δικτύου.

2.5.1 **Predict**

Υλοποιούμε την συνάρτηση output = predict(network, input), η οποία υπολογίζει την απόφαση του νευρωνικού δικτύου για το δεδομένο εισόδου, δηλαδή κατατάσσει μια εικόνα σε μια κλάση, κι έχει τα εξής ορίσματα κι εξόδους:

- *network*: το δικτυό μας
- *input*: η εικόνα εισόδου
- *output*: η κλάση-απόφαση του δικτύου

2.5.2 **Train**

Τέλος, υλοποιούμε την συνάρτηση train_mini_batch(network, loss, loss_prime, x_train, y_train, x_test, y_test, epochs, learning_rate, batch_size), η οποία χωρίζει τα δεδομένα σε mini-batches με τον τρόπο που αναλύσαμε παραπάνω και εκπαιδεύει το δίκτυο μέσω του αλγορίθμου backward propagatin. Επιπλέον, καλεί βοηθητικές συναρτήσεις για τον υπολογισμό της αποδοτικότητας κι ακρίβειας του δικτύου, καθώς και για την απεικόνιση των αποτελεσμάτων σε διαγράμματα. Έχει τα εξής ορίσματα κι εξόδους:

- *network*: το δίκτυό μας
- *loss*: η loss function της επιλογής μας
- *loss_prime*: η παράγωγος αυτής
- *x_train*, *y_train*: τα δεδομένα εκπαίδευσης
- *x_test*, *y_test*: τα δεδομένα ελέγχου
- *epochs*: ο αριθμός των εποχών που τρέχει ο αλγόριθμος
- *learning_rate*: ο βαθμός μάθησης α στο $backward - propagation$
- *batch_size*: το μέγεθος του κάθε *batch* δεδομένων

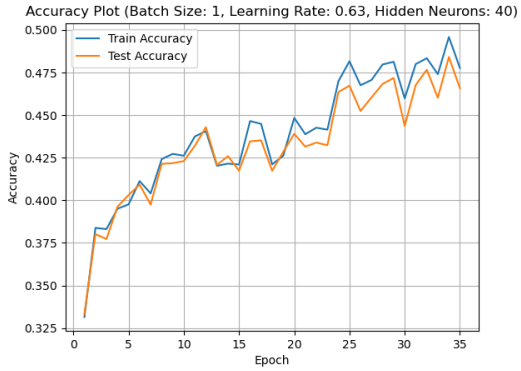
3 Αποτελέσματα - Σχολιασμοί

Ακολουθούν τα αποτελέσματα της παραπάνω θεωρητικής ανάλυσης και της υλοποίησης του κώδικα σε *python*. Θα δοθούν χαρακτηριστικά παραδείγματα ορθής και εσφαλμένης κατηγοριοποίησης, καθώς και ποσοστά επιτυχίας στα στάδια της εκπαίδευσης (*training*) και του ελέγχου (*testing*), χρόνος εκπαίδευσης και ποσοστά επιτυχίας για διαφορετικούς αριθμούς νευρώνων στο κρυφό επίπεδο, διαφορετικές τιμές των παραμέτρων εκπαίδευσης. Επιπλέον, θα συγκριθεί η απόδοση του νευρωνικού σε σχέση με την κατηγοριοποίηση πλησιέστερου γείτονα (*Nearest Neighbor*) και πλησιέστερου κέντρου κλάσης (*Nearest Class Centroid*) της ενδιάμεσης εργασίας. Η ορθή και εσφαλμένη κατηγοριοποίηση μπορεί να δειχθεί με την μορφή:

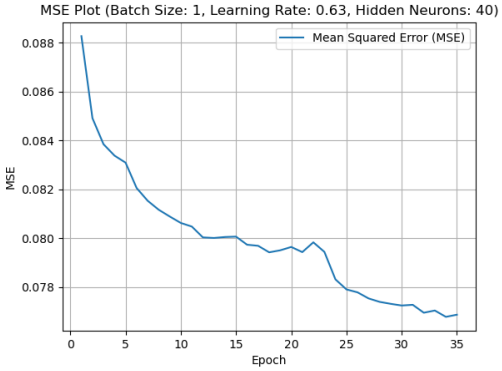
```
pred: 7  true: 7
pred: 2  true: 8
pred: 9  true: 9
pred: 8  true: 0
pred: 7  true: 3
pred: 8  true: 8
```

3.1 Δοκιμή Αριθμών Νευρώνων στο Κρυφό Στρώμα

Εκπαιδεύουμε το νευρωνικό δίκτυο για αριθμό νευρώνων στο κρυφό στρώμα $n = [40, 100, 200]$. Θέτουμε $learning\_rate = 0.63$ και εκπαιδεύουμε το δίκτυο για 35 εποχές. Παρακάτω αίνονται ενδεικτικά τα διαγράμματα του MSE και $accuracy$ στην περίπτωση του $n = 40$:



Σχήμα 8: Απόδοση - n



Σχήμα 9: Μέσο Τετραγωνικό Σφάλμα - n

Τα πλήρη (ενδεικτικά) αποτελέσματα φαίνονται συνοπτικά στον παρακάτων πίνακα:

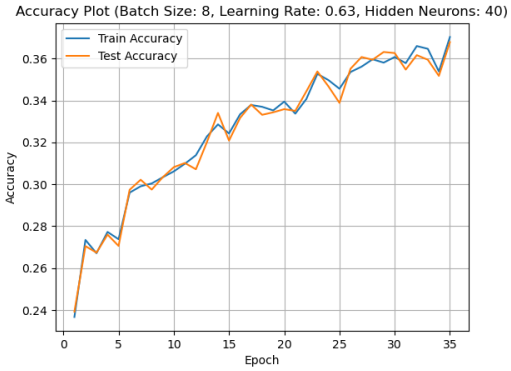
n	$Accuracy$	MSE	$Execution\ Time$
40	0.426	0.063	1230s
100	0.375	0.084	1654s
200	0.312	0.092	1989s

Πίνακας 1: Ενδεικτική απόδοση και σφάλμα για διαφορετικό αριθμό νευρώνων του κρυφού στρώματος

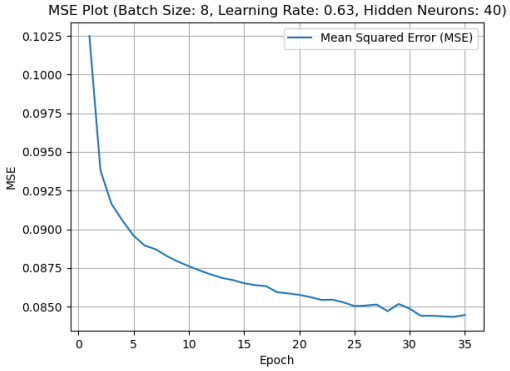
Παρατηρήσεις:
Παρατηρούμε ότι όσο αυξάνουμε τον αριθμό νευρώνων του κρυφού στρώματος μειώνεται η απόδοση του δικτύου. Γενικά δεν είναι κανόνας οπότε ακολουθείται τεχνική trial and error. Το βέβαιο είναι ότι αυξάνεται ο χρόνος εκτέλεσης, καθώς αυξάνεται η πολυπλοκότητα του δικτύου (περισσότεροι νευρώνες, άρα περισσότερες συνδέσεις, επομένως περισσότερα $weights$ και $biases$). Σημειώνεται ότι για καλύτερη απόδοση έχουμε επιλέξει $batch\_size = 1$, δηλαδή απλό $gradient\ descent$, ωστόσο αυτό δίνει πολύ μεγάλο χρόνο εκτέλεσης. Παρακάτω θα δοκιμάσουμε κι άλλα $batch\_sizes$.

3.2 Δοκιμή Διάφορων *batch sizes*

Εκπαιδεύουμε το νευρωνικό δίκτυο με mini-batch gradient descent και δοκιμάζουμε $batch\_size = [8, 16, 32]$. Θέτουμε $learning\_rate = 0.63$ και εκπαιδεύουμε το δίκτυο για 35 εποχές. Παρακάτω αίνονται ενδεικτικά τα διαγράμματα του *MSE* και *accuracy* στην περίπτωση του $batch\_size = 8$:



Σχήμα 10: Απόδοση - *batch_size*



Σχήμα 11: Μέσο Τετραγωνικό Σφάλμα - *batch_size*

Τα πλήρη (ενδεικτικά) αποτελέσματα φαίνονται συνοπτικά στον παρακάτων πίνακα:

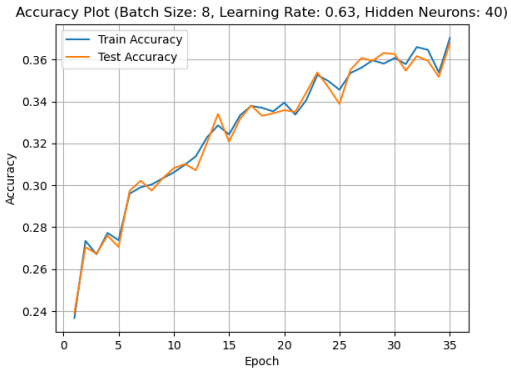
batch_size	*Accuracy*	*MSE*	*Execution Time*
8	0.378	0.079	360*s*
16	0.326	0.086	216*s*
32	0.287	0.088	99*s*

Πίνακας 2: Ενδεικτική απόδοση και σφάλμα για διαφορετικό *batch_size*

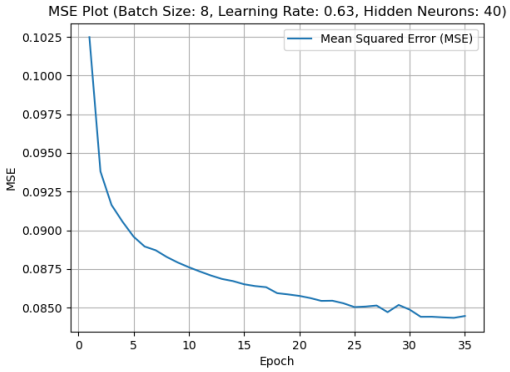
Παρατηρήσεις:
Παρατηρούμε ότι όσο αυξάνουμε το *batch_size* τόσο μειώνεται η απόδοση του δικτύου, γεγονός λογικό αφού χάνουμε την ευρωστία της απλής μεθόδου *gradient descent*, όπου οι παράμετροι ενημερώνονται για ένα-ένα δεδομένο εισόδου. Παρ' όλα αυτα ο χρόνος εκτέλεσης μειώνεται σημαντικά, γι' αυτό και προτιμάμε σχετικά μεγάλο *batch_size*.

3.3 Δοκιμή Διάφορων *learning rates*

Εκπαιδεύουμε το νευρωνικό δίκτυο με mini-batch gradient descent και $batch\_size = 8$. Δοκιμάζουμε $learning\_rate = [0.1, 0.55, 0.63]$ και εκπαιδεύουμε το δίκτυο για 35 εποχές. Παρακάτω αίνονται ενδεικτικά τα διαγράμματα του *MSE* και *accuracy* στην περίπτωση του $learning\_rate = 0.63$:



Σχήμα 12: Απόδοση - *learning_rate*



Σχήμα 13: Μέσο Τετραγωνικό Σφάλμα - *learning_rate*

Τα πλήρη (ενδεικτικά) αποτελέσματα φαίνονται συνοπτικά στον παρακάτων πίνακα:

learning_rate	*Accuracy*	*MSE*	*Execution Time*
0.1	0.297	0.096	489*s*
0.55	0.313	0.087	427*s*
0.63	0.378	0.079	360*s*

Πίνακας 3: Ενδεικτική απόδοση και σφάλμα για διαφορετικό *learning_rate*

Παρατηρήσεις:
Παρατηρούμε ότι μεγαλύτερο *learning rate* δίνει μεγαλύτερη απόδοση και μικρότερο χρόνο. Ωστόσο δεν είναι κανόνας, απαιτείται trial and error.

3.4 Σύγκριση με Άλλους Κατηγοριοποιητές

Μπορούμε, τέλος, να συγκρίνουμε την απόδοση του νευρωνικού μας δικτύου με τους κατηγοριοποιητές $K-Nearest-Neighbors$ και *Nearest Centroid Classifier* που υλοποιήθηκαν στην ενδιάμεση εργασία:

Classifier	**Accuracy**
KNN	0.3539
NCC	0.2774
NN	0.426

Πίνακας 4: Σύγκριση απόδοσης με άλλους κατηγοριοποιητές

Παρατηρήσεις:
Παρατηρούμε ότι με το νευρωνικό δίκτυό μας και με την κατάλληλη ρύθμιση των παραμέτρων του μπορούμε να πετύχουμε καλύτερο αποτέλεσμα από τους άλλους κατηγοριοποιητές για το ίδιο σύνολο δεδομένων εκπαίδευσης και δοκιμής. Επιπλέον, μπορούμε στο νευρωνικό δίκτυο να προσθέσουμε ιδιότητες, όπως συνελικτικά στρώματα ή πιο αντιπροσωπευτικά *features*, ώστε να βελτιώσουμε περαιτέρω την απόδοσή του.

Αναφορές

[1] CIFAR-10 and CIFAR-100 datasets. *Canadian Institute for Advanced Research (CIFAR)*. https://www.cs.toronto.edu/~kriz/cifar.html

[2] K-nearest neighbors algorithm. *Wikipedia*. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[3] Nearest centroid classifier. *Wikipedia*. https://en.wikipedia.org/wiki/Nearest_centroid_classifier

[4] A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size. *Machine Learning Mastery*. https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[5] Math - Neural Network from Scratch in Python. *Towards Data Science*. https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce65

[6] Neural Networks from Scratch in Python. *YouTube Video*. https://www.youtube.com/watch?v=pauPCy_s0Ok

[7] Backpropagation in Neural Networks Explained | Step by Step. *YouTube Video*. https://www.youtube.com/watch?v=Lakz2MoHy6o