ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Νευρωνικά Δίκτυα - Βαθιά Μάθηση

Εργασία 1

ΜΙΧΟΥ ΓΕΩΡΓΙΑ 3828

Η εργασία εκπονήθηκε από την άνωθεν φοιτήτρια του τμήματος Πληροφορικής του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης, στα πλαίσια του μαθήματος «Νευρωνικά Δίκτυα - Βαθιά Μάθηση» κατά τη διάρκεια του ακαδημαϊκού έτους 2023-2024.

1. Γενικά

Η παρακάτω εργασία αφορά την δημιουργία ενός πολυστρωματικού, πλήρως συνδεδεμένου, perceptron το οποίο εκπαιδεύεται με τον αλγόριθμο back propagation. Για την εκπαίδευση του δικτύου χρησιμοποιείται η βάση δεδομένων cifar-10. Η εργασία έχει υλοποιηθεί στη γλώσσα προγραμματισμού python, στο περιβάλλον google colab και επισυνάπτεται το αρχείο .ipynb.

2. Κώδικας - Υλοποίηση

Α) Υλοποίηση του δικτύου και εκπαίδευση: Η κλάση MultiLayerPerceptron αναπαριστά ένα πολυστρωματικό νευρωνικό δίκτυο perceptron (με 2 hidden layers) που εκπαιδεύεται με τον αλγόριθμο back propagation. Ξεκινώντας από τα πιο γενικά στοιχεία, οι παράμετροι του δικτύου είναι οι εξής: input_size: ο αριθμός των χαρακτηριστικών στα δεδομένα εισόδου, hidden_sizes: μια λίστα που περιέχει τον αριθμό των νευρώνων σε κάθε κρυφό επίπεδο, output_size: ο αριθμός των νευρώνων εξόδου, activations: μια λίστα που περιέχει τη συνάρτηση ενεργοποίησης για κάθε κρυφό επίπεδο, learning_rate: ο ρυθμός μάθησης για τον αλγόριθμο, weights_hidden: μια λίστα που περιέχει πίνακες βαρών για κάθε κρυφό στρώμα, biases_hidden: μια λίστα που περιέχει τα bias για κάθε κρυφό στρώμα, weights_output: ο πίνακας βαρών για το επίπεδο εξόδου, bias_output: το bias για το στρώμα εξόδου. Αναλυτικότερα, στην αρχή υπάρχει η init η οποία έχει τον ρόλο του constructor και καλείται όταν φτιάχνεται ένα αντικείμενο της κλάσης. Εκεί αρχικοποιούνται οι παραπάνω παράμετροι και συγκεκριμένα τα βάρη και τα bias για τα κρυφά στρώματα και το στρώμα εξόδου χρησιμοποιώντας την αρχικοποίηση He. Μέσω της συγκεκριμένης, τα βάρη παίρνουν τυχαίες τιμές, λαμβάνοντας υπόψιν τον αριθμό των εισόδων σε κάθε νευρώνα. Η He αρχικοποίηση χρησιμοποιείται ώστε να αποτρέψει τον περιορισμό των γραμμικών ενεργοποιήσεων στα κρυφά επίπεδα κατά την εκπαίδευση. Ακολουθεί το πρώτο μέρος του αλγορίθμου back propagation, η μέθοδος forward_propagation η οποία εκτελεί τη διαδικασία της προώθησης μέσα από το νευρωνικό δίκτυο. Προωθεί την είσοδο μέσα από τα κρυφά επίπεδα μέχρι το επίπεδο εξόδου, παράγοντας έτσι μια πρόβλεψη για την έξοδο του δικτύου. Η μέθοδος διατηρεί τις εξόδους των κρυφών επιπέδων

στη λίστα self.hidden_layer_outputs. Στη συνέχεια, επαναλαμβάνεται μέσα από τα κρυφά επίπεδα, υπολογίζοντας την είσοδο και την έξοδο του κάθε επιπέδου, χρησιμοποιώντας τη συνάρτηση ενεργοποίησης που έχει καθοριστεί για κάθε επίπεδο. Στο τέλος, υπολογίζεται η έξοδος του επιπέδου εξόδου, και προσπελαύνονται τα αποτελέσματα χρησιμοποιώντας τη συνάρτηση ενεργοποίησης που ορίζεται. Έπονται οι 2 συναρτήσεις που υιοθετούν την λογική του back propagation. Συγκεκριμένα, πρώτη είναι η forward_propagation η οποία προωθεί τα δεδομένα μέσα από το νευρωνικό δίκτυο. Λαμβάνει ένα σύνολο εισόδου X που πρόκειται για δεδομένα εκπαίδευσης ή ελέγχου. Πρώτα υπολογίζει την είσοδο του κρυφού επιπέδου (hidden_layer_input) χρησιμοποιώντας τον πολλαπλασιασμό των εισόδων X με τα βάρη από το επίπεδο εισόδου προς το κρυφό επίπεδο και προσθέτοντας τις παραμέτρους bias. Στη συνέχεια, υπολογίζει την έξοδο του κρυφού επιπέδου (hidden_layer_output) εφαρμόζοντας τη συνάρτηση ενεργοποίησης στην είσοδο του κρυφού επιπέδου. Έπειτα, υπολογίζει την είσοδο του τελικού επιπέδου (output_layer_input) χρησιμοποιώντας την έξοδο του κρυφού επιπέδου, την οποία πολλαπλασιάζει με τα βάρη από το κρυφό επίπεδο προς το τελικό επίπεδο και προσθέτει τις παραμέτρους παραμόρφωσης. Τέλος, υπολογίζει την πρόβλεψη του δικτύου (predicted_output) εφαρμόζοντας ξανά τη συνάρτηση ενεργοποίησης στην είσοδο του τελικού επιπέδου. Έπεται η μέθοδος backward_propagation η οποία υλοποιεί την διαδικασία της οπισθοδρόμησης και χρησιμοποιείται για την ενημέρωση των βαρών και των παραμέτρων του δικτύου με βάση το σφάλμα που παράγεται από την προώθηση. Αρχικά, υπολογίζεται το σφάλμα στο επίπεδο εξόδου, και στη συνέχεια ξεκινά η οπισθοδρόμηση μέσα από τα κρυφά επίπεδα. Για κάθε κρυφό επίπεδο, υπολογίζεται το σφάλμα του επιπέδου πριν από αυτό, πολλαπλασιάζεται με την παράγωγο της συνάρτησης ενεργοποίησης, και το αποτέλεσμα εισάγεται στην αρχή της λίστας errors. Στη συνέχεια, τα βάρη και οι παραμέτροι ενημερώνονται χρησιμοποιώντας την μέθοδο gradient descent. Αναλυτικά, έχοντας το βάρος για το *i*-οστό κρυφό επίπεδο και την αντίστοιχη έξοδο του και το σφάλμα που προκύπτει από το επόμενο (πιο προς τα εμπρός) επίπεδο, η ενημέρωση γίνεται με τον πολλαπλασιασμό του σφάλματος με την παράγωγο της συνάρτησης ενεργοποίησης του *i*-οστού κρυφού επιπέδου και την προσθήκη του αποτελέσματος στα βάρη. Το ίδιο συμβαίνει και για τα βάρη και τις παραμέτρους που αφορούν το επίπεδο εξόδου. Αυτή η διαδικασία επαναλαμβάνεται για όλες τις εποχές προσαρμόζοντας σταδιακά τα βάρη και τις παραμέτρους του δικτύου ώστε να ελαχιστοποιήσει το κόστος. Ακολουθούν δύο μέθοδοι (activation_function και activation_derivative) μέσα στις οποίες ορίζονται οι συναρτήσεις

ενεργοποίησης και οι παράγωγοι αυτών αντίστοιχα. Εδώ, ορίζονται οι relu και η sigmoid και αργότερα δοκιμάζονται είτε σε κρυφά επίπεδα είτε σε εξόδου, προκειμένου να διερευνηθεί η συμπεριφορά του δικτύου. Αμέσως μετά είναι η μέθοδος evaluate η οποία έχει δημιουργηθεί με σκοπό να αξιολογηθεί το δίκτυο. Συγκεκριμένα, χρησιμοποιεί τη μέθοδο forward_propagation για να προβλέψει τις ετικέτες των δεδομένων εισόδου X, τις προβλεπόμενες ετικέτες και τις πραγματικές ετικέτες (y) για να υπολογίσει την ακρίβεια του μοντέλου σε αυτό το σύνολο δεδομένων. Εκτυπώνει την ακρίβεια για το συγκεκριμένο σύνολο (εκπαίδευσης ή ελέγχου) καθώς και τον χρόνο εκπαίδευσης αν είναι το σύνολο εκπαίδευσης. Αν το σύνολο είναι το test, εκτυπώνει επιπλέον κάποια παραδείγματα σωστής ή εσφαλμένης κατηγοριοποίησης**.** Τελευταία είναι η μέθοδος train μέσα στην οποία γίνεται η εκπαίδευση του δικτύου. Αρχικά, ορίζει τον ρυθμό μάθησης και δημιουργεί τις one-hot ετικέτες για τα δεδομένα εκπαίδευσης. Για κάθε εποχή αρχικοποιεί το συνολικό σφάλμα (total_loss) στο μηδέν. Έπειτα, για κάθε "παρτίδα δεδομένων", δηλαδή batch size που ορίζουμε, εκτελεί forward propagation για πρόβλεψη των ετικετών. Μετά backward propagation ώστε να ενημερώσει τα βάρη και τις παραμέτρους του δικτύου. Υπολογίζει το σφάλμα κόστους όπου είναι η κατηγορική εντροπία και μετά το μέσο σφάλμα που αναπαριστά την απώλεια για κάθε εποχή. Προκύπτει παίρνοντας το άθροισμα των κοστών για κάθε δείγμα και στη συνέχεια διαιρώντας με τον συνολικό αριθμό των δειγμάτων. Αυτό δημιουργεί έναν μέσο όρο των κοστών για την εκάστοτε εποχή. Κάθε print_every εποχές, εκτυπώνονται πληροφορίες σχετικά με την πρόοδο της εκπαίδευσης, καλώντας την προαναφερθείσα μέθοδο evaluate.

Β) Εξαγωγή χαρακτηριστικών από το σύνολο δεδομένων: Στην αρχή φορτώνονται οι απαραίτητες βιβλιοθήκες και η βάση δεδομένων μέσω του keras. Η βάση έχει ήδη κάποια σύνολα για εκπαίδευση (5) και για έλεγχο (1) οπότε δεν έγινε κάποιος διαχωρισμός για αυτά. Οι εικόνες από το dataset ανασχηματίζονται (reshaping) σε έναν πίνακα 1D. Οι τιμές των pixel κανονικοποιούνται στο διάστημα [0, 1]. Έπειτα, λόγω χρονικής πολυπλοκότητας, εξάγονται συγκεκριμένα δεδομένα. Δημιουργείται μια συνάρτηση extract_features που παίρνει τις εικόνες και τον αριθμό των κύριων συνιστωσών (n_components). Για κάθε εικόνα υπολογίζει τη μέση φωτεινότητα κατά μήκος των γραμμών (axis=1) και των στηλών (axis=0), ξεχωριστά. Οι δύο πίνακες που προκύπτουν συνενώνονται με τη χρήση της συνάρτησης np.concatenate για να δημιουργήσουν έναν νέον πίνακα (image_features). Η συνάρτηση επαναλαμβάνει κάθε εικόνα στον πίνακα εισόδου και δημιουργεί αυτά τα συνδεδεμένα διανύσματα

χαρακτηριστικών, με αποτέλεσμα έναν νέο πίνακα όπου κάθε γραμμή αντιπροσωπεύει μια εικόνα με τα εξαγόμενα χαρακτηριστικά. Η μέθοδος αυτήν αποσκοπεί στην δημιουργία μιας διαφορετικής αναπαράστασης των εικόνων όπου το μοντέλο μαθαίνει από αυτά τα παραγόμενα χαρακτηριστικά και όχι από τις αρχικές τιμές εικονοστοιχείων των εικόνων. Έπειτα, εφαρμόζεται η μέθοδος PCA για μείωση της διαστατικότητας, διατηρώντας μόνο τις πρώτες n κύριες συνιστώσες και επιστρέφονται αυτά τα χαρακτηριστικά. Συγκριτικά με την ενδιάμεση εργασία όπου δοκιμάστηκαν ξεχωριστά οι συγκεκριμένες μέθοδοι, εδώ επιλέχθηκε ο συνδυασμός τους ώστε να βελτιωθεί η ικανότητα του μοντέλου να αντιληφθεί σημαντικά χαρακτηριστικά των εικόνων. Η μέση φωτεινότητα παρέχει πληροφορία σχετικά με τον γενικό χαρακτήρα της εικόνας, ενώ το PCA μπορεί να βοηθήσει στην αποφυγή του προβλήματος της υψηλής διαστατικότητας, που μπορεί να οδηγήσει σε υπερεκπαίδευση (overfitting).

C) Εκτέλεση του αλγορίθμου: Δημιουργείται ένα νέο αντικείμενο MultiLayerPerceptron (nn) με συγκεκριμένες τιμές στις παραμέτρους που δίνει κάθε φορά ο χρήστης. Το μοντέλο εκπαιδεύεται στα δεδομένα εκπαίδευσης (train_features) και ετικέτες (y_train) για έναν συγκεκριμένο αριθμό εποχών (epochs), με συγκεκριμένο ρυθμό μάθησης (learning_rate) και μέγεθος παρτίδας (batch_size). Αφού εκπαιδευτεί το μοντέλο, αξιολογείται στα δεδομένα εκπαίδευσης και ελέγχου. Ο χρόνος εκπαίδευσης, η απόδοση στα δεδομένα εκπαίδευσης και στα δεδομένα ελέγχου εκτυπώνονται για κάθε συνδυασμό παραμέτρων.

D) Ακολουθεί η περιγραφή των NearestCentroid και K Nearest Neighbours όπως αναλύθηκαν στην ενδιάμεση εργασία. Πρώτος είναι ο KNNClassifier: Μέσω της ομώνυμης κλάσης, δημιουργείται ο κατηγοριοποιητής πλησιέστερου γείτονα ο οποίος αργότερα καλείται με 1 και 3 γείτονες αντίστοιχα. Η λογική πίσω από τον αλγόριθμο βασίζεται στην αρχή της “ομοιότητας” που προκύπτει μέσω της ευκλείδιας απόστασης. Ο βασικός στόχος είναι να βρεθεί ο πλησιέστερος γείτονας μιας νέας εισόδου μέσα στο σύνολο δεδομένων εκπαίδευσης, χρησιμοποιώντας την συγκεκριμένη μετρική ανάμεσα στα δεδομένα. Αρχικά, ο classifier αρχικοποιείται με την τιμή k που αντιπροσωπεύει το πλήθος πλησιέστερων γειτόνων που πρέπει να ληφθούν υπόψιν (init). Ακολουθεί η μέθοδος fit η οποία δέχεται τα δεδομένα εκπαίδευσης και τις αντίστοιχες ετικέτες τους. Αποθηκεύει αυτά τα δεδομένα στον ταξινομητή για να χρησιμοποιηθούν για προβλέψεις. Για να υπολογιστούν αυτές, έχει δημιουργηθεί η predict η οποία λαμβάνει ένα σύνολο δειγμάτων (X) και προβλέπει τις ετικέτες τους χρησιμοποιώντας τον αλγόριθμο KNN. Για κάθε δείγμα στο X εκτελεί τα

εξής βήματα: a) Υπολογίζει τις ευκλείδειες αποστάσεις μεταξύ του τρέχοντος δείγματος και όλων των δειγμάτων στα δεδομένα εκπαίδευσης (self.X_train). b) Βρίσκει τους πλησιέστερους γείτονες επιλέγοντας τα k δείγματα από το σύνολο εκπαίδευσης που είναι πλησιέστερα στο τρέχον δείγμα με βάση τις υπολογισμένες αποστάσεις. c) Έπειτα βρίσκει τις ετικέτες των k πλησιέστερων γειτόνων και προσδιορίζει την ετικέτα που εμφανίζεται συχνότερα. Αυτή η ετικέτα πλειοψηφίας προβλέπεται ως η ετικέτα για το τρέχον δείγμα. Ακολουθεί ο NearestCentroidClassifier: Μέσω της ομώνυμης κλάσης, δημιουργείται ο κατηγοριοποιητής πλησιέστερου κέντρου όπου στην αρχή αρχικοποιεί τον αριθμό των κέντρων ίσο με μηδέν. Αυτός ο αλγόριθμος επιλέγει το κοντινότερο κέντρο από κάθε δείγμα ελέγχου και αναθέτει την κλάση του κοντινότερου κέντρου στο δείγμα αυτό, βάσει της ευκλείδιας απόστασης μεταξύ του δείγματος ελέγχου και των κέντρων που έχουν υπολογιστεί κατά τη διάρκεια της εκπαίδευσης. Αναλυτικότερα, στην μέθοδο fit, γίνεται η εκπαίδευση όπου για κάθε κλάση (συνολικά 10) υπολογίζεται ο μέσος όρος των δειγμάτων που ανήκουν σε αυτήν για να βρεθεί το "κέντρο" για την εκάστοτε κλάση. Αυτά τα υπολογισμένα κέντρα αποθηκεύονται στο χαρακτηριστικό self.centers. Τέλος, η μέθοδος predict υπολογίζει τις αποστάσεις από κάθε ένα από τα αποθηκευμένα κέντρα κλάσεων και αναθέτει την ετικέτα του πλησιέστερου κέντρου ως την προβλεπόμενη ετικέτα για το συγκεκριμένο δείγμα. Οι δυο παραπάνω κλάσεις δημιουργήθηκαν υιοθετώντας την λογική των αντίστοιχων αλγορίθμων k nearest neighbors και nearest centroid.

Ε) Αποτελέσματα νευρωνικού δικτύου: Παρακάτω έχει δημιουργηθεί ένας πίνακας με ορισμένες εκτελέσεις του δικτύου σε διαφορετικές παραμέτρους με σκοπό την αξιολόγηση του. Έχουν χρησιμοποιηθεί δύο ζεύγη με διαφορετικά κρυφά layers ενώ στην εξαγωγή χαρακτηριστικών η PCA έχει 64 κύριες συνιστώσες.

Εκτέλεση	Batch size	Epochs	Learning rate	Number of hidden neurons	Activation hidden	Activation output	Loss	Train accuracy	Test accuracy	Training time per batch (sec)	Σύγκριση
1	32	10000	0,001	64 και 64	Sigmoid	Sigmoid	1,492	0,4987	0,353	229.81	Συνάρτηση ενεργοποίησης στο επίπεδο εξόδου
2	32	10000	0,001	64 και 64	Sigmoid	Relu	1,541	0,478	0,340	233.51	
3	32	10000	0,1	64 και 64	Sigmoid	Sigmoid	12,923	0,100	0,100	230,59	Σύγκριση με 1 στο learning rate
4	32	10000	0,01	64 και 64	Sigmoid	Relu	1,658	0,438	0,339	258,77	Batch size
5	64	10000	0,01	64 και 64	Sigmoid	Relu	1,652	0,4390	0,332	151,56	
7	64	10000	0,01	128 και 128	Sigmoid	Relu	1,553	0,4889	0,346	418,46	Πλήθος εποχών
8	64	15000	0,01	128 και 128	Sigmoid	Relu	1,530	0,493	0,342	456,7	
9	32	10000	0,001	128 και 128	Sigmoid	Relu	1,516	0,485	0,376	302,31	Συνάρτηση ενεργοποίησης στο επίπεδο εξόδου
10	32	10000	0,001	128 και 128	Sigmoid	Sigmoid	1,341	0,556	0,358	314,18	

Παρατηρήσεις: Στην εκτέλεση 3, δεν παρατηρείται εξέλιξη στην απόδοση ενώ το loss είναι μεγάλο, πιθανόν έγινε μεγάλο overfitting. Στις δοκιμές που έγιναν, η καλύτερη απόδοση (0,556 ή αλλιώς 55,6%) που επιτυγχάνει το δίκτυο είναι με παραμέτρους batch_size = 32, epochs = 10000, learning rate = 0.001, number of hidden neurons = [128,128] , activation hidden = sigmoid , activation output = sigmoid.

Παρακάτω είναι κάποιες εικόνες οι οποίες αποτυπώνουν κάποια παραδείγματα ορθής και εσφαλμένης κατηγοριοποίησης. Να σημειωθεί πως ο χρόνος εκπαίδευσης εμφανίζεται ανά επανάληψη και στον πίνακα έχει σημειωθεί εκείνος των ενδιάμεσων σταδίων (και όχι στο τελικό) καθώς στο τελικό στάδιο ο χρόνος έχει μειωθεί αρκετά και δεν αποτυπώνεται η ρεαλιστική διάρκεια εκπαίδευσης στα βήματα.

```
Epoch 10000/10000
Training Set - Accuracy: 0.4987
Training Time: 0.00 seconds
Testing Set - Accuracy: 0.4987
Examples of Correct Categorization:
Predicted: 9, Actual: 9
Predicted: 4, Actual: 4
Predicted: 7, Actual: 7
Predicted: 8, Actual: 8
Predicted: 7, Actual: 7

Examples of Incorrect Categorization:
Predicted: 2, Actual: 6
Predicted: 1, Actual: 9
Predicted: 9, Actual: 1
Predicted: 7, Actual: 1
Predicted: 6, Actual: 2
Loss: 1.481

Activation Hidden: sigmoid, Activation Output: sigmoid, Hidden Neurons: [64, 64], Learning Rate: 0.001
Training Accuracy: 0.499, Test Accuracy: 0.359, Training Time: 0.44 seconds
------------------------------------------------------------
```

Εκτέλεση 1

```
Epoch 10000/10000
Training Set - Accuracy: 0.4378
Training Time: 0.00 seconds
Testing Set - Accuracy: 0.4378
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 2, Actual: 2
Predicted: 7, Actual: 7
Predicted: 8, Actual: 8

Examples of Incorrect Categorization:
Predicted: 1, Actual: 9
Predicted: 6, Actual: 4
Predicted: 8, Actual: 1
Predicted: 7, Actual: 1
Predicted: 9, Actual: 3
Loss: 1.658

Activation Hidden: sigmoid, Activation Output: relu, Hidden Neurons: [64, 64], Learning Rate: 0.01
Training Accuracy: 0.438, Test Accuracy: 0.339, Training Time: 0.17 seconds
------------------------------------------------------------
```

Εκτέλεση 4

```
Epoch 10000/10000
Training Set - Accuracy: 0.4889
Training Time: 0.00 seconds
Testing Set - Accuracy: 0.4889
Examples of Correct Categorization:
Predicted: 9, Actual: 9
Predicted: 4, Actual: 4
Predicted: 7, Actual: 7
Predicted: 8, Actual: 8
Predicted: 4, Actual: 4

Examples of Incorrect Categorization:
Predicted: 7, Actual: 6
Predicted: 8, Actual: 9
Predicted: 8, Actual: 1
Predicted: 7, Actual: 1
Predicted: 4, Actual: 2
Loss: 1.553

Activation Hidden: sigmoid, Activation Output: relu, Hidden Neurons: [128, 128], Learning Rate: 0.01
Training Accuracy: 0.489, Test Accuracy: 0.346, Training Time: 0.38 seconds
------------------------------------------------------------
```

Εκτέλεση 7

```
Epoch 15000/15000
Training Set - Accuracy: 0.4926
Training Time: 0.00 seconds
Testing Set - Accuracy: 0.4926
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 1, Actual: 1
Predicted: 1, Actual: 1
Predicted: 2, Actual: 2

Examples of Incorrect Categorization:
Predicted: 1, Actual: 9
Predicted: 6, Actual: 4
Predicted: 6, Actual: 4
Predicted: 7, Actual: 2
Predicted: 2, Actual: 9
Loss: 1.530

Activation Hidden: sigmoid, Activation Output: relu, Hidden Neurons: [128, 128], Learning Rate: 0.01
Training Accuracy: 0.493, Test Accuracy: 0.342, Training Time: 0.39 seconds
------------------------------------------------------------
```

Εκτέλεση 8

```
Epoch 8500/10000
Loss: 1.517
Training Set - Accuracy: 0.4867
Training Time: 291.18 seconds
Testing Set - Accuracy: 0.4867
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 1, Actual: 1
Predicted: 2, Actual: 2
Predicted: 7, Actual: 7

Examples of Incorrect Categorization:
Predicted: 1, Actual: 9
Predicted: 6, Actual: 4
Predicted: 8, Actual: 1
Predicted: 7, Actual: 3
Predicted: 6, Actual: 4


Epoch 9000/10000
Loss: 1.517
Training Set - Accuracy: 0.4862
Training Time: 291.67 seconds
Testing Set - Accuracy: 0.4862
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 4, Actual: 4
Predicted: 1, Actual: 1
Predicted: 2, Actual: 2
```

```
Epoch 8500/10000
Loss: 1.344
Training Set - Accuracy: 0.5538
Training Time: 313.53 seconds
Testing Set - Accuracy: 0.5538
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 1, Actual: 1
Predicted: 2, Actual: 2
Predicted: 7, Actual: 7

Examples of Incorrect Categorization:
Predicted: 1, Actual: 9
Predicted: 6, Actual: 4
Predicted: 7, Actual: 1
Predicted: 1, Actual: 3
Predicted: 6, Actual: 4


Epoch 9000/10000
Loss: 1.343
Training Set - Accuracy: 0.5552
Training Time: 314.57 seconds
Testing Set - Accuracy: 0.5552
Examples of Correct Categorization:
Predicted: 6, Actual: 6
```

Ενδιάμεσα βήματα εκτέλεσης στις 9 και 10

Για μεγαλύτερη διερεύνηση, έγιναν 3 επιπλέον εκτελέσεις με παραμέτρους που προηγουμένως έφεραν τις καλύτερες αποδόσεις αλλά στην μέθοδο PCA υπάρχουν τώρα 128 συνιστώσες. Στις συγκεκριμένες εκτελέσεις του αλγορίθμου, παρατηρείται μικρότερο overfitting στα στάδια εκπαίδευσης και αύξηση της απόδοσης του δικτύου. Η χρονική πολυπλοκότητα αυξάνεται σημαντικά. Στις δοκιμές που έγιναν, η καλύτερη απόδοση (0,576 ή αλλιώς 57.6%) που επιτυγχάνει το δίκτυο είναι με παραμέτρους batch_size = 32, epochs = 10000, learning rate = 0.001, number of hidden neurons = [128,128] , activation hidden = sigmoid , activation output = sigmoid. Παρατηρείται πως με τις ίδιες παραμέτρους και στον προηγούμενο πίνακα, το δίκτυο φέρει καλύτερα αποτελέσματα με διαφορά απόδοσης στο 2%. Συγκρίνοντας την απόδοση του δικτύου με κρυφά στρώματα που περιέχει το καθένα από 64 νευρώνες, με 128 κύριες συνιστώσες στη μέθοδο PCA, το δίκτυο πετυχαίνει αύξηση της απόδοσής του κατά 2.5-3%.

Όσον αφορά το πλήθος εποχών και το batch size, δεν μπόρεσαν να δοκιμαστούν αριθμοί μεγαλύτεροι του 10000 (μόνο μία φορά) και αριθμοί μικρότεροι του 32 αντίστοιχα, καθώς ο χρόνος εκπαίδευσης αυξανόταν πολύ και δυσκόλευε την διαδικασία εκπαίδευσης.

Εκτέλεση	Batch size	Epochs	Learning rate	Number of hidden neurons	Activation hidden	Activation output	Loss	Train accuracy	Test accuracy	Training time per batch (sec)	PCA
1	32	10000	0,001	64 και 64	Sigmoid	Sigmoid	1,492	0,4987	0,353	229.81	64
2							1,443	0,520	0,344	268,04	128
3	32	10000	0,001	128 και 128	Sigmoid	Sigmoid	1,341	0,556	0,358	314,18	64
4							1,284	0,576	0,352	712,63	128
5	32	10000	0,001	128 και 128	Sigmoid	Relu	1,516	0,485	0,376	303,31	64
6							1,507	0,488	0,376	320,10	128

Εκτέλεση 2

```
Epoch 10000/10000
Training Set - Accuracy: 0.5197
Training Time: 0.00 seconds
Testing Set - Accuracy: 0.5197
Examples of Correct Categorization:
Predicted: 9, Actual: 9
Predicted: 9, Actual: 9
Predicted: 2, Actual: 2
Predicted: 7, Actual: 7
Predicted: 8, Actual: 8

Examples of Incorrect Categorization:
Predicted: 4, Actual: 6
Predicted: 6, Actual: 4
Predicted: 9, Actual: 1
Predicted: 7, Actual: 1
Predicted: 7, Actual: 3
Loss: 1.443
```

Εκτέλεση 4

```
Epoch 10000/10000
Training Set - Accuracy: 0.5759
Training Time: 0.00 seconds
Testing Set - Accuracy: 0.5759
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 9, Actual: 9
Predicted: 1, Actual: 1
Predicted: 1, Actual: 1

Examples of Incorrect Categorization:
Predicted: 6, Actual: 4
Predicted: 1, Actual: 3
Predicted: 6, Actual: 4
Predicted: 1, Actual: 2
Predicted: 5, Actual: 3
Loss: 1.284

Activation Hidden: sigmoid, Activation Output: sigmoid, Hidden Neurons: [128, 128], Learning Rate: 0.001
Training Accuracy: 0.576, Test Accuracy: 0.352, Training Time: 0.52 seconds
```

```
Epoch 8000/10000
Loss: 1.289
Training Set - Accuracy: 0.5758
Training Time: 712.63 seconds
Testing Set - Accuracy: 0.5758
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 1, Actual: 1
Predicted: 1, Actual: 1
Predicted: 2, Actual: 2

Examples of Incorrect Categorization:
Predicted: 8, Actual: 9
Predicted: 6, Actual: 4
Predicted: 1, Actual: 3
Predicted: 6, Actual: 4
Predicted: 1, Actual: 2


Epoch 8500/10000
Loss: 1.287
Training Set - Accuracy: 0.5761
Training Time: 711.54 seconds
```

Εκτέλεση 6

```
Epoch 3500/10000
Loss: 1.326
Training Set - Accuracy: 0.5596
Training Time: 713.30 seconds
Testing Set - Accuracy: 0.5596
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 1, Actual: 1
Predicted: 1, Actual: 1
Predicted: 7, Actual: 7

Examples of Incorrect Categorization:
Predicted: 1, Actual: 9
Predicted: 6, Actual: 4
Predicted: 4, Actual: 2
Predicted: 1, Actual: 3
Predicted: 6, Actual: 4


Epoch 4000/10000
Loss: 1.319
Training Set - Accuracy: 0.5634
Training Time: 714.36 seconds
Testing Set - Accuracy: 0.5634
```

```
Epoch 7000/10000
Loss: 1.515
Training Set - Accuracy: 0.4831
Training Time: 320.66 seconds
Testing Set - Accuracy: 0.4831
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 1, Actual: 1
Predicted: 1, Actual: 1
Predicted: 2, Actual: 2

Examples of Incorrect Categorization:
Predicted: 1, Actual: 9
Predicted: 6, Actual: 4
Predicted: 9, Actual: 3
Predicted: 6, Actual: 4
Predicted: 9, Actual: 7


Epoch 7500/10000
Loss: 1.513
Training Set - Accuracy: 0.4846
Training Time: 318.26 seconds
Testing Set - Accuracy: 0.4846
Examples of Correct Categorization:
Predicted: 6, Actual: 6
```

```
Training Time: 0.00 seconds
Testing Set - Accuracy: 0.4880
Examples of Correct Categorization:
Predicted: 6, Actual: 6
Predicted: 9, Actual: 9
Predicted: 1, Actual: 1
Predicted: 2, Actual: 2
Predicted: 7, Actual: 7

Examples of Incorrect Categorization:
Predicted: 1, Actual: 9
Predicted: 6, Actual: 4
Predicted: 8, Actual: 1
Predicted: 7, Actual: 3
Predicted: 1, Actual: 2
Loss: 1.507

Activation Hidden: sigmoid, Activation Output: relu, Hidden Neurons: [128, 128], Learning Rate: 0.001
Training Accuracy: 0.488, Test Accuracy: 0.376, Training Time: 0.25 seconds
```

F) Σύγκριση απόδοσης με τους classifiers: Παρατίθενται εικόνες από τα αποτελέσματα που φέρουν οι κατηγοριοποιητές knn και nc. Όπως φαίνεται, όταν το δίκτυο δεν υποπίπτει σε έντονο overfitting ώστε να επηρεαστεί αρνητικά η απόδοσή του, καταφέρνει να λύσει το πρόβλημα κατηγοριοποίησης στην cifar-10 με μεγαλύτερη επιτυχία.

KNN for 1-neighbor and results

```
[39] from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
     k_value = 1
     knn1 = KNNClassifier(k=k_value)
     knn1.fit(train_features, y_train)
     knn1_predictions = knn1.predict(test_features)

     accuracy_knn1 = accuracy_score(y_test, knn1_predictions)
     print(f"Accuracy of KNN (k={k_value}): {accuracy_knn1}")
     weighted_recall_knn1 = recall_score(y_test, knn1_predictions, average='weighted')
     print(f"Weighted recall score of KNN (k={k_value}): {weighted_recall_knn1}")
     macro_recall_knn1 = recall_score(y_test, knn1_predictions, average='macro')
     print(f"Macro recall score of KNN (k={k_value}): {macro_recall_knn1}")
     weighted_f1score_knn1 = f1_score(y_test, knn1_predictions, average='weighted')
     print(f"Weighted f1 score of KNN (k={k_value}): {weighted_f1score_knn1}")
     macro_f1score_knn1 = f1_score(y_test, knn1_predictions, average='macro')
     print(f"Macro f1 score of KNN (k={k_value}): {macro_f1score_knn1}")
```

```
Accuracy of KNN (k=1): 0.303
Weighted recall score of KNN (k=1): 0.303
Macro recall score of KNN (k=1): 0.30300000000000005
Weighted f1 score of KNN (k=1): 0.30233528841172397
Macro f1 score of KNN (k=1): 0.3023352884117239
```

KNN for 3-neighbors and results

```
k_value = 3
knn3 = KNNClassifier(k=k_value)
knn3.fit(train_features, y_train)
knn3_predictions = knn3.predict(test_features)

accuracy_knn3 = accuracy_score(y_test, knn3_predictions)
print(f"Accuracy of KNN (k={k_value}): {accuracy_knn3}")
weighted_recall_knn3 = recall_score(y_test, knn3_predictions, average='weighted')
print(f"Weighted recall score of KNN (k={k_value}): {weighted_recall_knn3}")
macro_recall_knn3 = recall_score(y_test, knn3_predictions, average='macro')
print(f"Macro recall score of KNN (k={k_value}): {macro_recall_knn3}")
weighted_f1score_knn3 = f1_score(y_test, knn3_predictions, average='weighted')
print(f"Weighted f1 score of KNN (k={k_value}): {weighted_f1score_knn3}")
macro_f1score_knn3 = f1_score(y_test, knn3_predictions, average='macro')
print(f"Macro f1 score of KNN (k={k_value}): {macro_f1score_knn3}")
```

```
Accuracy of KNN (k=3): 0.3111
Weighted recall score of KNN (k=3): 0.3111
Macro recall score of KNN (k=3): 0.3111
Weighted f1 score of KNN (k=3): 0.3062192987148664
Macro f1 score of KNN (k=3): 0.3062192987148664
```

NC and results

```
from sklearn.neighbors import NearestCentroid

nc = NearestCenterClassifier()
nc.fit(train_features, y_train)
nc_predictions = nc.predict(test_features)

accuracy_nc = accuracy_score(y_test, nc_predictions)
print(f"Accuracy of Nearest Center Classifier: {accuracy_nc}")
weighted_recall_nc = recall_score(y_test, nc_predictions, average='weighted')
print(f"Weighted recall score of Nearest Center Classifier: {weighted_recall_nc}")
macro_recall_nc = recall_score(y_test, nc_predictions, average='macro')
print(f"Macro recall score of Nearest Center Classifier: {macro_recall_nc}")
weighted_f1score_nc = f1_score(y_test, nc_predictions, average='weighted')
print(f"Weighted f1 score of Nearest Center Classifier: {weighted_f1score_nc}")
macro_f1score_nc = f1_score(y_test, nc_predictions, average='macro')
print(f"Macro f1 score of Nearest Center Classifier: {macro_f1score_nc}")
```

```
Accuracy of Nearest Center Classifier: 0.2498
Weighted recall score of Nearest Center Classifier: 0.2498
Macro recall score of Nearest Center Classifier: 0.24980000000000002
Weighted f1 score of Nearest Center Classifier: 0.22298517407375856
Macro f1 score of Nearest Center Classifier: 0.2229851740737586
```