Πρώτη εργασία νευρωνικών δικτύων 2024-2025

Αβραμίδου Αφροδίτη, τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ζητούμενο εργασίας:
Υλοποίηση ενός πλήρως συνδεδεμένου νευρωνικού δικτύου εμπρόσθιας τροφοδότησης (feedforward NN) που εκπαιδεύεται με τον αλγόριθμο
back-propagation. Το NN αυτό εκπαιδεύεται για να επιλύει το πρόβλημα
κατηγοριοποίησης των κλάσεων της κλάσης δεδομένων cifar-10. Για την βελτίωση της υλοποίησης έχει χρησιμοποιηθεί μείωση των διαστάσεων των δεδομένων με την τεχνική PCA.

I. ΑΝΑΛΥΣΗ ΚΩΔΙΚΑ

(X_train, Y_train), (X_test, Y_test) = cifar10.load_data() :
Αρχικά φορτώνονται τα δεδομένα της βάσης δεδομένων cifar-10, που αποτελείται από 60.000 έγχρωμες εικόνες, μοιρασμένες σε 10 κλάσεις. Οι εικόνες είναι μεγέθους 32x32 και καθεμία έχει 3 κανάλια χρώματος (RGB) οπότε το μέγεθος της εισόδου είναι 32×32×3=3072 features.
Επίσης συμβαίνει "αναδιάρθρωση" (flattening) των εικόνων με την εντολή .reshape(X_train.shape[0], -1) η οποία μετατρέπει τα δεδομένα εκπαίδευσης σε μονοδιάστατους πίνακες, δηλαδή κάθε εικόνα μετατρέπεται σε ένα διάνυσμα στήλης μεγέθους 3072 x 1. Το αντίστοιχο συμβαίνει και για το σύνολο των testing δεδομένων. Ταυτόχρονα συμβαίνει και κανονικοποίηση των πίξελ καθώς οι τιμές τους από το αρχικό τους εύρος στο [0,255] διαιρώντας τες με 255 παίρνουν τιμές στο εύρος [0,1](improving gradient-based learning). Οι πρακτικές αυτές συνεισφέρουν στην ταχύτερη (ευκολότερες πράξεις μεταξύ των πινάκων) και αποτελεσματικότερη εκπαίδευση του νευρωνικών δικτύων.

pca(X, num_components):
υλοποίηση της μεθόδου pca (Principal Component Analysis), για την μείωση των διαστάσεων των εικόνων του dataset, έτσι ώστε γίνει απλοποίηση των δεδομένων χωρίς ωστόσο να χαθεί η πιο σημαντική πληροφορία.
Η συνάρτηση δέχεται ως παραμέτρους το σύνολο δεδομένων εισόδου X και των αριθμό των κύριων συνιστωσών num_components που θέλουμε να κρατήσουμε μετά την μείωση της διάστασης. Επιστρέφει: τα μειωμένα σε διάσταση μέσω του PCA δεδομένα X_reduced, τα ιδιοδιανύσματα top_eigenvectors που αντιστοιχούν στις κορυφαίες κύριες συνιστώσες (μαλλον παπατζα) και τον μέσο όρο για κάθε feature κατά μήκος των δειγμάτων, mean_vector. (για να μπορούν να ανακτηθούν τα δεδομένα στην αρχική τους μορφή)
Συνοπτικά, τα βήματα που ακολουθεί ο αλγόριθμος είναι:

1. το ''κεντράρισμα'' των δεδομένων αφαιρώντας από κάθε feature τον μέσο όρο του, με αποτέλεσμα γύρω από το μηδέν τα δεδομένα να έχουν μέσο όρο 0.
2. Υπολογισμός πίνακα συνδιακύμανσης των κεντραρισμένων δεδομένων, ο οποίος περιγράφει τη συσχέτιση μεταξύ των features των δεδομένων. Αν τα features είναι ανεξάρτητα, η συνδιακύμανσή τους είναι μηδέν. Κάθε γραμμή και στήλη του πίνακα αυτού αναφέρεται στη συνδιακύμανση δύο χαρακτηριστικών.
3. Υπολογισμός των **ιδιοτιμών** (eigenvalues) και **ιδιοδιανυσμάτων** (eigenvectors) του πίνακα συνδιακύμανσης. Τα **ιδιοδιανύσματα** είναι οι κατευθύνσεις στις οποίες τα δεδομένα "εκτείνονται" περισσότερο, ενώ οι **ιδιοτιμές** αντιπροσωπεύουν τη σημαντικότητα αυτών των κατευθύνσεων (μεγαλύτερη ιδιοτιμή σημαίνει ότι η αντίστοιχη κατεύθυνση έχει μεγαλύτερη διασπορά και επομένως μεγαλύτερη πληροφορία).
4. Ταξινόμηση των ιδιοδιανυσμάτων κατά φθίνουσα σειρά ιδιοτιμών, έτσι ώστε να εντοπιστούν οι «πιο σημαντικές» κατευθύνσεις (αυτές με τις μεγαλύτερες ιδιοτιμές). Τελικά επιλέγονται τα num_components κορυφαία ιδιοδιανύσματα (πρώτες num_components στήλες του πίνακα eigenvectors), που αντιστοιχούν στις κύριες συνιστώσες.
5. Προβολή των κεντραρισμένων δεδομένων στις κορυφαίες κύριες συνιστώσες, με τον τύπο του πίνακα X_reduced = top_eigenvectors.T @ X_centered., που έχει ως αποτέλεσμα τη μείωση της διάστασης των δεδομένων από τον αρχικό αριθμό χαρακτηριστικών σε num_components σε πλήθος χαρακτηριστικά (κύριες συνιστώσες).

Ο pca εφαρμόζεται για τη μείωση της διάστασης των δεδομένων εκπαίδευσης και δοκιμής σε 300 κύριες συνιστώσες (αριθμός που επέλεξα αυθαίρετα), δηλαδή μείωση των features από 3072 σε 300 , και στη συνέχεια προβάλει τα δεδομένα του συνόλου δοκιμών στις κορυφαίες συνιστώσες που προέκυψαν από τα δεδομένα εκπαίδευσης.
Με την εντολή X_test_centered = X_test - train_mean_vector, τα δεδομένα του συνόλου δοκιμής (X_test) κεντράρονται αφαιρώντας τον μέσο όρο των δεδομένων εκπαίδευσης (train_mean_vector). Αυτό είναι σημαντικό, καθώς το κεντράρισμα των δεδομένων εκπαίδευσης και δοκιμής πρέπει να γίνει με τον ίδιο τρόπο. Αν δεν γίνει αυτό, η προβολή των

δεδομένων του συνόλου δοκιμών στις κύριες συνιστώσες δεν θα είναι ακριβής. Η προβολή αυτή έχει ως αποτέλεσμα τη μείωση διάστασης των δεδομένων δοκιμής στις 300 συνιστώσες.

init_params(input_size=300):
Αρχικοποίηση των παραμέτρων W1,W2,b1, b2 δηλαδή των βαρών και των biases. Έχω επιλέξει για το νευρωνικό μου να έχει ένα κρυφό επίπεδο με 50 νευρώνες ενώ η έξοδος αποτελείται από 10 νευρώνες, αντιπροσωπευτικοί των 10 τελικών κλάσεων του προβλήματος κατηγοριοποίησης. Τα βάρη και τα Biases αρχικοποιούνται τυχαία np.random.rand, και μετατοπίζονται κατά -0.5 για να βρίσκονται στο διάστημα [-0.5, 0.5] (κυρίως για αποφυγή συμμετρίας κατά την εκπαίδευση).

ReLU(Z), softmax(Z):
Οι συναρτήσεις ενεργοποίησης, η ReLU για το κρυφό επίπεδο και η softmax για το επίπεδο εξόδου. Η πρώτη θέτει όλες τις αρνητικές τιμές στο 0 και τις θετικές ως έχουν δηλαδή ReLU(Z)=max(0,Z), ενώ η δεύτερη μετατρέπει τις εξόδους του νευρωνικού στις προβλεπόμενες πιθανότητες για κάθε κλάση σύμφωνα με τον τύπο: Aj $= \frac{e^{zj}}{\sum_{k=1}^{n} e^{zj}}$, όπου A_j: η πιθανότητα που αποδίδει το δίκτυο στην κλάση j, e^{zj}: η εκθετική τιμή για την κλάση j και $\sum_{k=1}^{n} e^{zj}$: το άθροισμα όλων των εκθετικών τιμών για όλες τις κλάσεις k. Έτσι επιστρέφει έναν πίνακα `A` με διαστάσεις (10, n_samples), όπου κάθε στοιχείο αντιπροσωπεύει την πιθανότητα που αποδίδει το δίκτυο σε κάθε κλάση για κάθε δείγμα. Οι πιθανότητες αυτές αθροίζουν στο `1` για κάθε δείγμα.

forward_prop(W1, b1, W2, b2, X):
Η υλοποίηση της εμπρόσθιας τροφοδότησης του νευρωνικού (forward propagation), όπου οι είσοδοι του κρυφού επιπέδου και του επιπέδου εξόδου Z1 και Z2 αντίστοιχα μετασχηματίζονται σύμφωνα με τους γραμμικούς μετασχηματισμούς Z1=W1*X+b1 και Z2=W2*A1+b2, ενώ για στην έξοδο του κρυφού επιπέδου A1 εφαρμόζεται η ReLU συνάρτηση ενεργοποίησης και στην έξοδο του επιπέδου εξόδου A2 η softmax, από την οποία παίρνω τις κανονικοποιημένες τιμές του Z2 σε πιθανότητες που αθροίζουν στο 1 για κάθε δείγμα.

ReLU_deriv(Z):
Υπολογισμός της παραγώγου της συνάρτησης ενεργοποίησης ReLU, η οποία ορίζεται ως:
Για Z>0 : ∂ReLU(Z)/∂Z=1 και για Z≤0 : ∂ReLU(Z)/∂Z=0
Δηλαδή η συνάρτηση επιστρέφει έναν δυαδικό πίνακα (True/False), με
True(1): για τιμές Z>0 και False(0): για τιμές Z≤0.
Το αποτέλεσμα χρησιμοποιείται κατά το back-propagation για την ενημέρωση των βαρών.

one_hot(Y, num_classes=10):
Οι ετικέτες των 10 κλάσεων κωδικοποιούνται σύμφωνα με την κωδικοποίηση One-hot, μια αναπαράσταση όπου κάθε ετικέτα μετατρέπεται σε έναν δυαδικό πίνακα με μήκος ίσο με τον αριθμό των κλάσεων. Στον πίνακα αυτό η θέση που αντιστοιχεί στην κλάση της ετικέτας παίρνει την τιμή 1, ενώ όλες οι άλλες θέσεις παίρνουν την τιμή 0. Για παράδειγμα για την ετικέτα 3, η one-hot κωδικοποίησή της θα ήταν: [0,0,0,1,0,0,0,0,0,0].

compute_loss(A2, Y):
Μετράει την απόκλιση μεταξύ των προβλεπόμενων πιθανοτήτων (A2) και των πραγματικών ετικετών-κωδικοποιημένων σε one-hot σύμφωνα με τον τύπο της cross entropy loss ως:
$L = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{numclasses} Y_{ij} \log(A2_{ij})$.Ο όρος 1e−8 έχει προστεθεί για να αποφευχθεί ο υπολογισμός του log(0), που θα προκαλούσε αριθμητική αστάθεια.

backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y):
Υλοποίηση του back-propagation, υπολογίζει τις παραγώγους της συνάρτησης κόστους L dW1,db1,dW2,db2 λαμβάνοντας υπόψιν όλα τα βάρη και τα biases, έτσι ώστε να ενημερώνονται κατά την εκπαίδευση μειώνοντας το σφάλμα. Χρησιμοποιώ τον κανόνα της αλυσίδας για να υπολογίσω πώς επηρεάζουν τα weights και τα biases το σφάλμα. Ξεκινάω από την έξοδο προς την είσοδο υπολογίζοντας τα: σφάλμα στην έξοδο (dZ2=∂Z2/∂Loss), παράγωγοι των weights και biases της εξόδου (dW2= $\frac{1}{m}$ *dZ2 *$A1^T$, db2= $\frac{1}{m}\sum dZ2$), **σφάλμα στο κρυφό επίπεδο** (dZ1=($W2^T$·dZ2) *ReLU_deriv(Z1)), **παράγωγοι** των weights και biases **του κρυφού επιπέδου** (dW1= $\frac{1}{m}$ *dZ1 *X^T, db1= $\frac{1}{m}\sum dZ1$).

update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha):
Ενημερώνει τα weights και τα biases χρησιμοποιώντας χρησιμοποιώντας τις παραγώγους που υπολογίστηκαν στην back-propagation και τον ρυθμό εκμάθησης α. Έτσι ενημερώνει κάθε παράμετρο αφαιρώντας έναν όρο που είναι το γινόμενο του ρυθμού εκμάθησης α και της αντίστοιχης παραγώγου. Με την αφαίρεση αυτή κινούμαστε προς την κατεύθυνση μείωσης της κλίσης της συνάρτησης κόστους, δηλαδή προς το ελάχιστό της. Το α καθορίζει τον ρυθμό με τον οποίο ενημερώνονται οι παράμετροι. Μικρό α οδηγεί σε αργή σύγκλιση, ενώ μεγάλο μπορεί να προκαλέσει αστάθεια.

get_predictions(A2):
Χρησιμοποιείται τόσο κατά την εκπαίδευση όσο και κατά την αξιολόγηση για να μετατρέψει τις εξόδους του δικτύου (πιθανότητες) σε προβλέψεις κλάσεων. Με την εντολή np.argmax(A2, axis=0) επιστρέφεται ο δείκτης της μέγιστης τιμής για κάθε στήλη του πίνακα A2, όπου A2 ο πίνακας των εξόδων του τελικού επιπέδου του νευρωνικού δικτύου (πιθανότητες ανά κλάση για κάθε δείγμα που παράγονται ως έξοδοι της συνάρτησης softmax). Κάθε στήλη του A2 αναπαριστά τις πιθανότητες για μία εικόνα, οπότε ο δείκτης της μέγιστης τιμής αντιστοιχεί στην προβλεπόμενη κλάση για την εικόνα αυτή.

get_accuracy(predictions, Y):
Υπολογισμός της **ακρίβειας** του νευρωνικού δικτύου συγκρίνοντας τις προβλέψεις του με τις πραγματικές ετικέτες. Στην εντολή np.sum(predictions == Y), το μέρος predictions == Y δημιουργεί έναν δυαδικό πίνακα όπου κάθε στοιχείο είναι: True (1) αν η πρόβλεψη συμφωνεί με την πραγματική ετικέτα και False (0) αν διαφέρει (το predictions λαμβάνεται από την συνάρτηση get_predictions), ενώ η συνάρτηση np.sum υπολογίζει το πλήθος των True, δηλαδή τον συνολικό

αριθμό ορθών προβλέψεων. Ο αριθμός αυτός των ορθών προβλέψεων διαιρείται με το συνολικό αριθμό δειγμάτων Y.size για να υπολογιστεί η ακρίβεια ως: $Accuracy = \frac{Correct\ Predictions}{Total\ Samples}$.

create_mini_batches(X, Y, batch_size):
Χωρισμός των δεδομένων σε μικρότερες δέσμες (mini-batches) που αποσκοπεί στην αποτελεσματικότερη εκπαίδευση και την μειωμένη χρήση μνήμης. Τα βασικά βήματα είναι: ανακάτεμα έτσι ώστε τα δεδομένα και οι ετικέτες να αναδιαταχθούν σύμφωνα με τους τυχαίους δείκτες του πίνακα δεικτών indices (X_shuffled και Y_shuffled). Με τον βρόχο for τα ζεύγη (Xbatch,Ybatch) προστίθενται στη λίστα mini_batches που δημιουργήθηκε για την αποθήκευσή τους, την οποία επιστρέφει η συνάρτηση.

gradient_descent(X_train_pca, Y_train, X_test_pca, Y_test, alpha, epochs, batch_size):
διατρέχει επαναλαμβανόμενα τα epochs και τα mini-batches προκειμένου να ενημερώσει τα weights και τα biases, χρησιμοποιώντας τις συναρτήσεις forward_prop, backward_prop και update_params.
Αρχικοποιούνται ως λίστες οι μεταβλητές που με ενδιαφέρουν: train_losses, test_losses, train_accuracies και test_accuracies, για την καταγραφή των απωλειών και της ακρίβειας ανά τα epochs τόσο για τα δεδομένα εκπαίδευσης όσο και για τα δεδομένα δοκιμής. Τα epochs διατρέχονται επαναλαμβανόμενα με τον βρόχο for, μέσα στον οποίο συμβαίνουν τα εξής:
Δημιουργούνται τυχαία mini-batches από το σύνολο δεδομένων εκπαίδευσης, καθένα από τα οποία περιέχει ένα υποσύνολο των δειγμάτων και των αντίστοιχων ετικετών. Για κάθε mini-batch ενημερώνονται οι παράμετροι σύμφωνα με τις προαναφερθείσες συναρτήσεις.
Κάθε 20 epochs εκτυπώνεται η απώλεια και η ακρίβεια για τα δεδομένα εκπαίδευσης και δοκιμής και δημιουργούνται τα αντίστοιχα γραφήματα που αναπαριστούν την εξέλιξή τους με την αύξηση των epochs.

Έτσι στην συνέχεια με αυτή την συνάρτηση εκπαιδεύω τα δεδομένα, καλώντας την με τα PCA-μετασχηματισμένα δεδομένα εκπαίδευσης και δοκιμής, επιλέγω ρυθμό μάθησης α=0.001, epochs=300 (το αλλάζω στην πορεία) και batch_size=64. Επιστρέφονται οι τελικές παράμετροι του μοντέλου καθώς και Οι προβλέψεις του συνόλου δοκιμής (test_predictions).

Η συνάρτηση confusion_matrix() της sklearn χρησιμοποιείται για τη δημιουργία ενός πίνακαconfusion matrix, ενός 10 x 10 πίνακα, για τις 10 κλάσεις της cifar-10, στον οποίο κάθε διαγώνιο κελί δείχνει τον αριθμό των σωστών προβλέψεων για αυτή την κλάση ενώ τα κελιά εκτός της διαγωνίου δείχνουν τις λανθασμένες ταξινομήσεις. Υψηλές τιμές κατά μήκος της διαγωνίου υποδεικνύουν καλή απόδοση ενώ οι υψηλές τιμές εκτός της διαγωνίου αναδεικνύουν τις κλάσεις που μπερδεύονται συχνότερα.

make_predictions(X, W1, b1, W2, b2):
πρόβλεψη για τα νέα δεδομένα εισόδου, χρησιμοποιώντας την συνάρτηση forward_prop() που επιστρέφει τις πιθανότητες κάθε κλάσης για κάθε δείγμα (A2). Έπειτα με την κλήση της get_predictions() επιλέγεται η κλάση με τη μέγιστη πιθανότητα.
Σημείωση: η λειτουργία που επιτελεί η συνάρτηση αυτή έχει ήδη υλοποιηθεί εντός της συνάρτησης gradient_descent() (την ήθελα εκεί για να κάνω γραφικές παραστάσεις σε κοινό figure των training vs. test accuracy και training vs. test loss) οπότε θα μπορούσε να παραληφθεί, υπάρχει για δική μου διευκόλυνση, για να χρησιμοποιήσω το όρισμα predictions που επιστρέφει μετέπειτα στον κώδικα.

get_predicted_images(predictions):
βοηθητική συνάρτηση, η οποία επιλέγει τυχαία εικόνες από το σύνολο δοκιμών που να αντιστοιχούν σε κάθε προβλεπόμενη κατηγορία. Ο βρόχος for διατρέχει όλες τις προβλέψεις (predictions) και εντοπίζει όλες τις θέσεις στο σύνολο δοκιμής όπου η πραγματική ετικέτα αντιστοιχεί στην προβλεπόμενη κατηγορία (indices = np.where(Y_test == pred)[0]). Από αυτές επιλέγει τυχαία μία, ανακτά την εικόνα από το σύνολο δοκιμής (X_test) την αναδιαμορφώνει στο αρχικό της μέγεθος (32 x 32 x 3), την αποκανονικοποιεί (πολλαπλασιασμός με 255 για μετατροπή σε RGB μορφή) και την προσθέτει στην λίστα predicted_images, την οποία επιστρέφει η συνάρτηση. Ουσιαστικά χρησιμοποιείται για την παραγωγή παραδειγμάτων εικόνων που ανήκουν στις κατηγορίες που προβλέπει το μοντέλο (ζητείται να συμπεριληφθούν τέτοια παραδείγματα).

test_predictions_with_images(positions, W1, b1, W2, b2):
Εξέταση των προβλέψεων του νευρωνικού δικτύου σε συγκεκριμένες θέσεις του συνόλου δοκιμής. Στόχος της συνάρτησης είναι: να προβλέψει τις κλάσεις για τις εικόνες του συνόλου δοκιμής που βρίσκονται στις θέσεις {positions}, επιστρέφει τις πραγματικές εικόνες, τις εικόνες που αντιστοιχούν στις προβλεπόμενες κλάσεις, τις πραγματικές ετικέτες και τις προβλέψεις του μοντέλου. Υπάρχουν 3 λίστες για την αποθήκευση των: πραγματικών εικόνων από το σύνολο δοκιμής, των ετικετών που προβλέπει το μοντέλο και των πραγματικών ετικετών των εικόνων.
Κάθε εικόνα δοκιμής πρέπει να μετασχηματιστεί στον χώρο PCA που είχε υπολογιστεί από το σύνολο εκπαίδευσης. Τελικά οι εικόνες προκειμένου να οπτικοποιηθούν αποκανονικοποιούνται και επαναφέρονται στο αρχικό τους μέγεθος.
Έτσι αυτή η συνάρτηση χρησιμεύει στην οπτική σύγκριση των πραγματικών και προβλεπόμενων κλάσεων.

Στο τελευταίο κομμάτι του κώδικα, επιλέγονται τυχαία δείγματα από το σύνολο δοκιμής και καλείται η συνάρτηση test_predictions_with_images με την οποία παίρνω: πραγματικές εικόνες (σε RGB μορφή), αντιπροσωπευτικές εικόνες για τις προβλεπόμενες κλάσεις, τις πραγματικές ετικέτες των επιλεγμένων εικόνων και τις προβλέψεις του μοντέλου.
Τέλος οι πραγματικές και οι προβλεπόμενες εικόνες οπτικοποιούνται: στην πρώτη γραμμή εμφανίζονται οι πραγματικές εικόνες με τίτλο την κλάση στην οποία ανήκουν,

ενώ στην δεύτερη γραμμή οι προβλεπόμενες εικόνες για τις κλάσεις που αντιστοιχούν στις προβλέψεις του μοντέλου.

ΙΙ. ΑΠΟΤΕΛΕΣΜΑΤΑ – ΠΑΡΑΤΗΡΗΣΕΙΣ

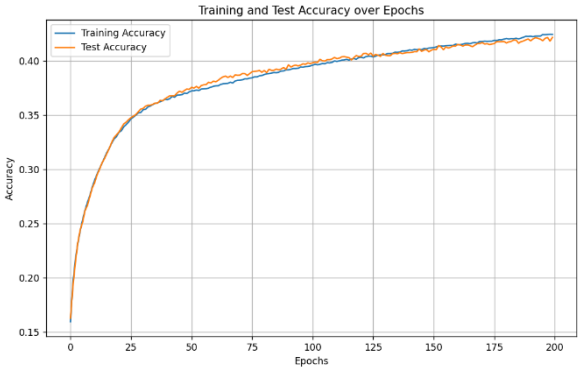
Θα εξετάσω την απόδοση του νευρωνικού μου μεταβάλλοντας τις παρακάτω παραμέτρους:

- Epochs
- Learning rate
- Πλήθος νευρώνων κρυφού επιπέδου
- Προσθήκη κανονικοποίησης (regularization)

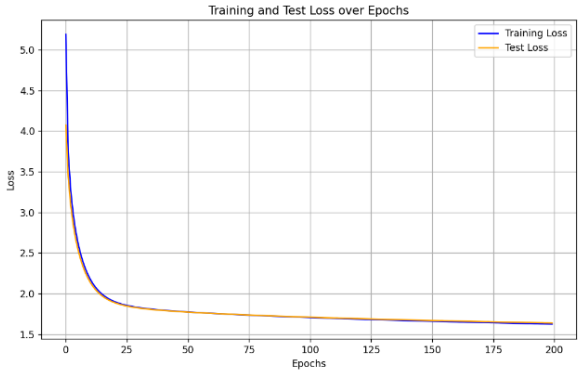
Ορισμός εννοιών που θα χρησιμοποιήσω:
overfitting: large gap between training and test accuracy, το μοντέλο απομνημονεύει υπερβολικά τα δεδομένα εκπαίδευσης και **αποτυγχάνει να γενικεύσει** σε νέα δεδομένα
Πιθανή αιτία: Δεν χρησιμοποιούνται μέθοδοι **regularization**, όπως **dropout**, **L1/L2 κανονικοποίηση**.
underfitting: Το μοντέλο δεν μαθαίνει αρκετά από τα δεδομένα εκπαίδευσης και δεν αποδίδει καλά ούτε στα δεδομένα εκπαίδευσης ούτε στα δεδομένα δοκιμής. Χαμηλή ακρίβεια τόσο στα δεδομένα εκπαίδευσης όσο και στα δεδομένα δοκιμής. Η καμπύλη απώλειας (loss curve) δεν μειώνεται αρκετά.

I. Σταθερές παράμετροι: α=0.001, num_components = 300, hidden layer's neurons=50:

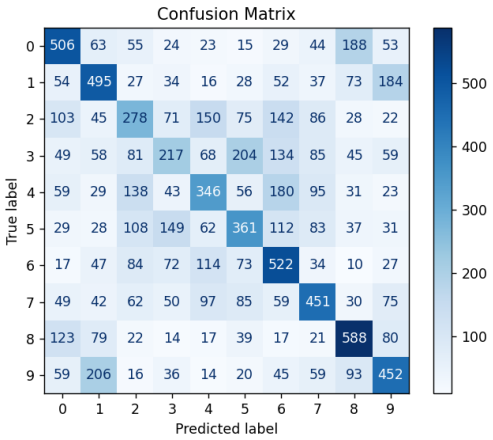
- epochs=200



Training accuracy = 0.4244, Test accuracy = 0.4216



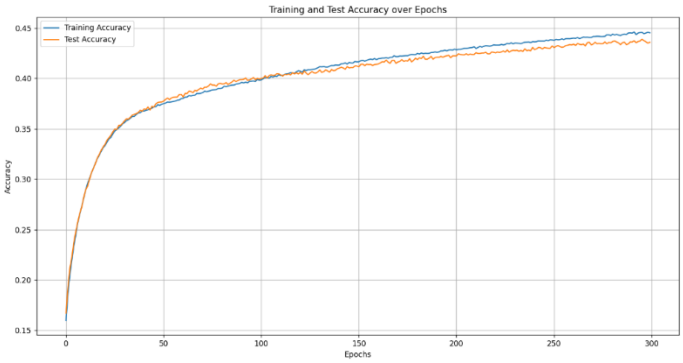
Train Loss = 1.6245, Test Loss = 1.6398



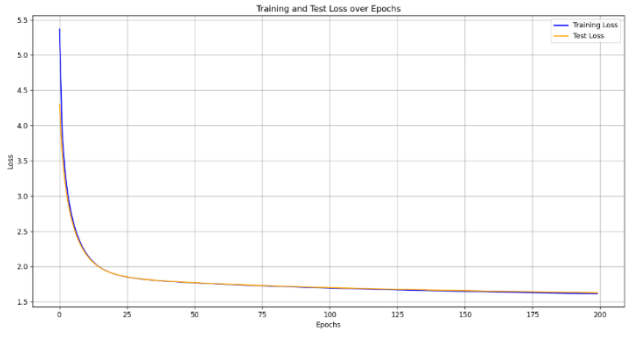
Confusion Chart

Παρατηρώ ότι τα δεδομένα των συνόλων εκπαίδευσης και δοκιμής συγκλίνουν σε αρκετά μεγάλο βαθμό, ώστε να θεωρώ ότι δεν υπάρχει overfitting ή underfitting.
Η διαβάθμιση χρώματος στον confusion matrix υποδεικνύει τον αριθμό των δειγμάτων, με τις πιο σκούρες αποχρώσεις να αντιπροσωπεύουν υψηλότερες τιμές. Έτσι γίνεται ξεκάθαρο και οπτικά πού ο αλγόριθμος αποδίδει καλά (σκούρα κελιά κατά μήκος της διαγωνίου) και πού αντιμετωπίζει δυσκολίες (σκούρα κελιά εκτός διαγωνίου). Η απόδοση του αλγορίθμου είναι ικανοποιητική, δεδομένου ότι οι πιο σκούρες αποχρώσεις του μπλε παρουσιάζονται στα διαγώνια στοιχεία και για τις 10 κλάσεις.

- epochs= 300:

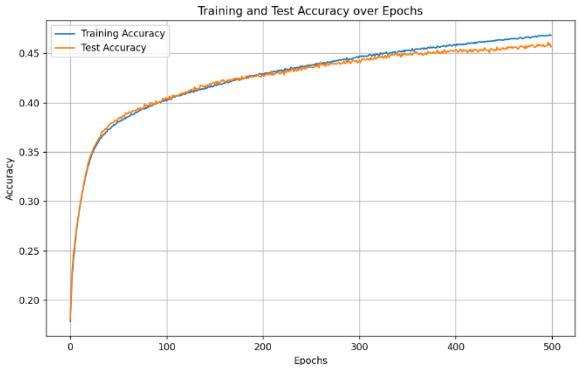


Train accuracy= 0.4452, Test accuracy= 0.4358

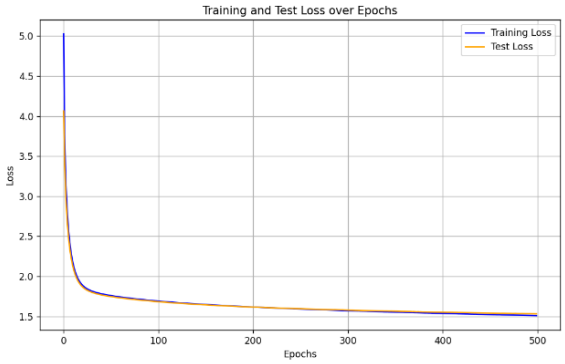


Train loss= 1.5727, Test loss= 1.5891

- Epochs= 500:



Train Accuracy = 0.4634, Test Accuracy = 0.4547

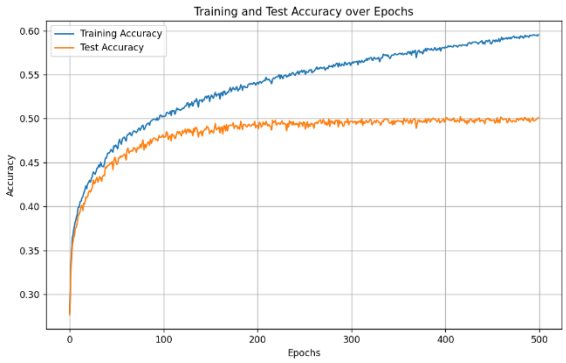


Train Loss = 1.5143, Test Loss = 1.5497

Με την αύξηση των epochs παρατηρώ αύξηση του accuracy και μείωση του loss, ως αναμένονταν.

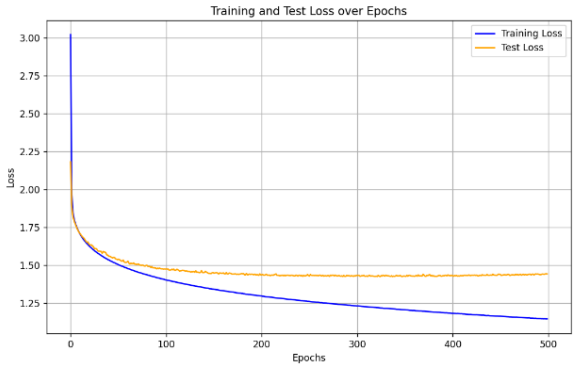
II. Σταθερές παράμετροι: epochs=500, num_components = 300, hidden layer's neurons=50:

- α=0.01:

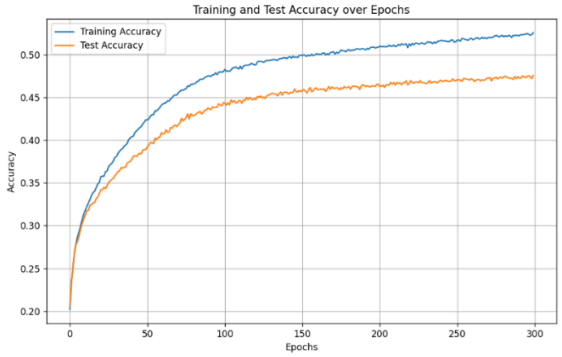


Train Accuracy = 0.5953, Test Accuracy = 0.5010

Το train και το test accuracy εμφανίζουν αρκετά σημαντική απόκλιση, όπως και το train και το test loss, η οποία είναι ανεπιθύμητη καθώς μπορεί να συμβαίνει overfitting των δεδομένων.

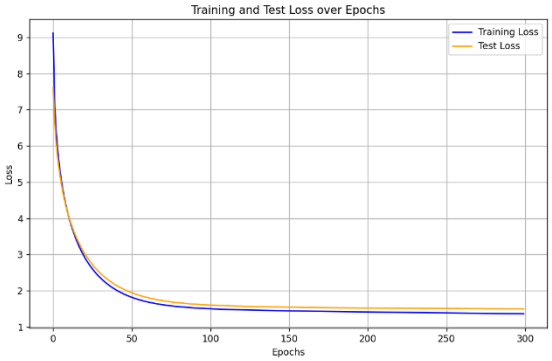


III. III. Σταθερές παράμετροι: α=0.001, epochs=300, num_components = 300



Train Accuracy = 0.5255, Test Accuracy = 0.4756

- hidden layer's neurons=300:

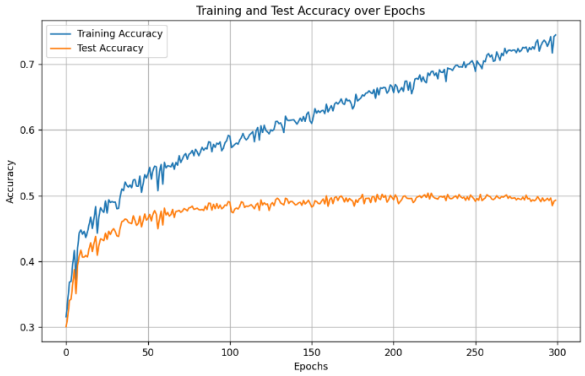


Train Loss = 1.3588, Test Loss = 1.4961

I. IV. Σταθερές παράμετροι: epochs=300, num_components = 300, hidden layer's neurons=300:

- α=0.01:

Ενώ φτάνει σε πολύ υψηλό train accuracy, το test accuracy δεν ακολουθεί, που σημαίνει ότι το μοντέλο απομνημονεύει υπερβολικά τα δεδομένα εκπαίδευσης αλλά αποτυγχάνει να γενικεύσει σε νέα δεδομένα (overfitting).



Train Accuracy = 0.7451, Test Accuracy = 0.4929

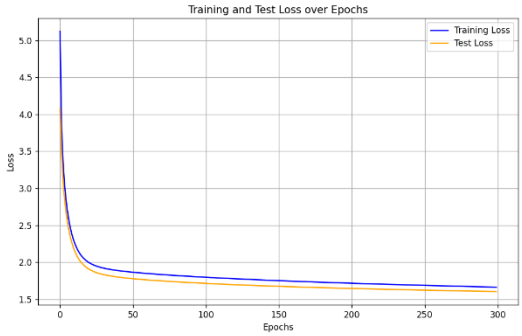
Train Loss = 0.7463, Test Loss = 1.7676

Οι απώλειες παρουσιάζουν μικρότερη απόκλιση που ωστόσο ξεπερνάει το 10% και είναι ανεπιθύμητη.

- Εφαρμογή κανονικοποίησης (regularization) :
  I. **L2 regularization**: δίνει κάποιο «πέναλτι» στα μεγάλα βάρη, που καθορίζεται από την παράμετρο lambda, βοηθώντας έτσι το μοντέλο στην καλύτερη γενίκευση αποφεύγοντας το overfitting.
  Τιμές για την παράμετρο lambda: 0.001, 0.01, 0.1 (τρέχω τον κώδικα σταθερά για epochs=300, hidden layer's neurons=50, batch size=64)

a. Lambda=0.01

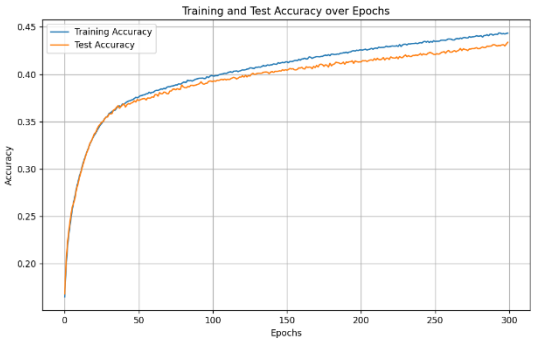
Train Accuracy = 0.4422, Test Accuracy = 0.4328



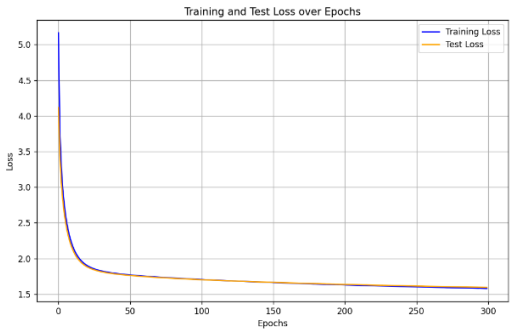
Train loss=1.5740, Test loss= 1.6048

Παρατηρώ χειρότερα αποτελέσματα για την ακρίβεια απ' ότι χωρίς το regularization για τις ίδιες τιμές των υπόλοιπων παραμέτρων.

b. Lambda=0.001:



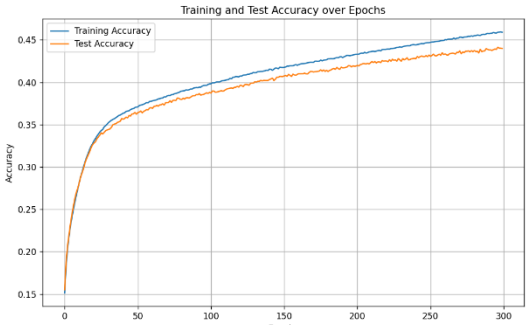
Train Accuracy = 0.4434, Test Accuracy = 0.4338



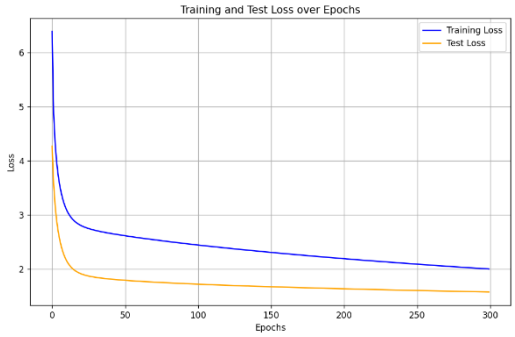
Train Loss = 1.5676, Test Loss = 1.5940

Παρατηρώ σχεδόν τα ίδια αποτελέσματα με την περίπτωση Lambda=0.01 ως προς την ακρίβεια, με μικρή βελτίωση ως προς τις απώλειες, train και test loss συγκλίνουν περισσότερο.

c. Lambda=0.1



Train Accuracy = 0.4590, Test Accuracy = 0.4399

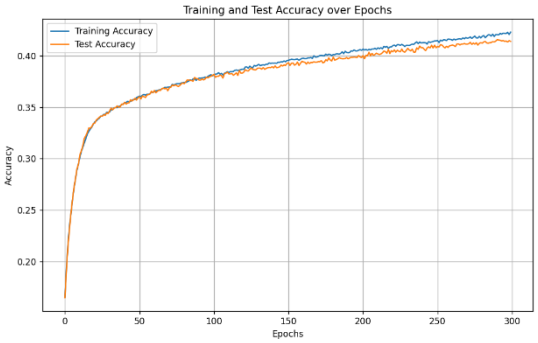


Train Loss = 1.5346, Test Loss = 1.5714

Παρατηρώ σχεδόν τα ίδια αποτελέσματα με την περίπτωση Lambda=0.01 και Lambda=0.001 ως προς την ακρίβεια, οι καμπύλες απωλειών ωστόσο παρουσιάζουν ένα σημαντικό κενό μεταξύ τους, κάτι που δεν συνέβαινε στις προηγούμενες περιπτώσεις που εξέτασα.

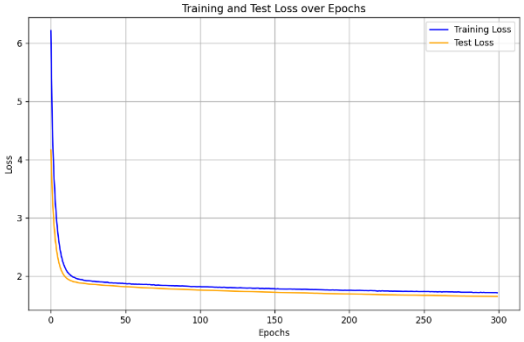
Γενικά παρατηρώ ότι με τον συγκεκριμένο αλγόριθμο τόσο η απόδοση όσο και οι απώλειες εξακολουθούν να παραμένουν στα ίδια επίπεδα, οπότε δεν τον χρησιμοποιώ.

II. **Dropout regularization**: σε κάθε βήμα εκπαίδευσης, κάθε νευρώνας σε ένα στρώμα έχει μια πιθανότητα p να "μηδενιστεί" (δηλαδή, να μη συμμετάσχει στους υπολογισμούς). Η επιλογή των νευρώνων που θα "μηδενιστούν" γίνεται τυχαία. Οι συνεισφορές των μηδενισμένων νευρώνων αγνοούνται, αλλά οι συνδέσεις των υπόλοιπων νευρώνων μαθαίνουν κανονικά.
Η παράμετρος keep_prob καθορίζει την πιθανότητα να παραμείνει ο νευρώνας ενεργός. Τιμές για την παράμετρο keep_prob : 0.8, 0.01, 0.1 (τρέχω τον κώδικα σταθερά για epochs=300, hidden layer's neurons=50, batch size=64)



Train Loss = 1.6373, Test Loss = 1.6452

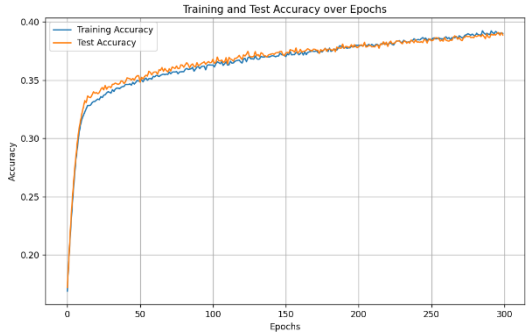
A. keep_prob = 0.8



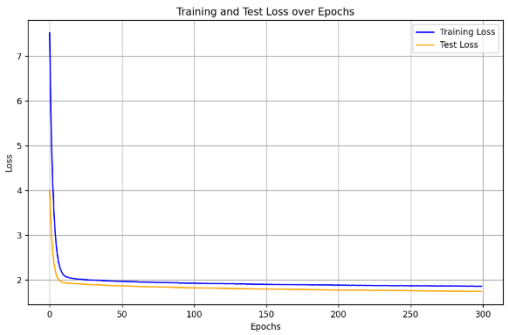
Train Accuracy = 0.4230, Test Accuracy = 0.4141

Η ακρίβεια και οι απώλειες για τα δεδομένα εκπαίδευσης και δοκιμής συγκλίνουν, ωστόσο εμφανίζονται ελαφρώς χειρότερα από αυτά χωρίς τον αλγόριθμο Dropout.

B. keep_prob = 0.5



Train Accuracy = 0.3899, Test Accuracy = 0.3887



Train Loss = 1.7405, Test Loss = 1.7399

Παρατηρώ χειρότερη απόδοση, τόσο στην ακρίβεια όσο και στις απώλειες σε σχέση με την προηγούμενη περίπτωση όπου keep_prob = 0.8, καθώς και περισσότερο θόρυβο στα δεδομένα (φαίνεται από το γεγονός ότι οι καμπύλες της ακρίβειας δεν είναι πλέον ομαλές) δηλαδή συμπεραίνω ότι η μείωση των νευρώνων κατά το ήμισυ είναι “too aggressive”.

ΠΙΝΑΚΑΣ Ι
Παραδείγματα ορθής και εσφαλμένης κατηγοριοποίησης

Image	True label	Predicted label
1	7	8
2	3	3
3	7	7
4	5	1
5	1	0
6	3	1
7	1	6
8	6	6
9	4	2
10	0	0

III. ΣΥΜΠΕΡΑΣΜΑΤΑ

Η υλοποίηση του πλήρως συνδεδεμένου νευρωνικού δικτύου εμπρόσθιας τροφοδότησης (feedforward NN) με αλγόριθμο back-propagation παρουσιάζει φασώς καλύτερη απόδοση από τους αλγορίθμους κατηγοριοποίησης πλησιέστερου γείτονα (Nearest Neighbor) και πλησιέστερου κέντρου κλάσης (Nearest Class Centroid), για τις αποδόσεις των οποίων είχα:
Nearest Class Centroid: 27.74%
3-NearestNeighbor: 33.27%
1-NearestNeighbor: 35.39%
ενώ με το παρόν νευρωνικό δίκτυο η απόδοση είναι περίπου 45%.