ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΗΛ 311 – Στατιστική Μοντελοποίηση και Αναγνώριση Προτύπων – 2024

Διδάσκων: Βασίλης Διγαλάκης **2η Σειρά Ασκήσεων** Επίλυση μέχρι: 27/06/2024

Αναφορά

Θέμα 1: Αλγόριθμος Perceptron

Σε αυτήν την άσκηση θα ασχοληθούμε με την υλοποίηση του αλγόριθμου Perceptron, στην batch μορφή του, για την ταξινόμηση ανάμεσα σε δύο κλάσεις. Ξεκινώντας, για να υλοποιήσουμε τον αλγόριθμο perceptron χρειάζεται να υλοποιήσουμε:

- Μια γραμμική συνάρτηση που συναθροίζει τα σήματα εισόδου(aggregation function)
- Μια συνάρτηση κατωφλίου που καθορίζει αν ο νευρώνας απόκρισης πυροδοτείται ή όχι(threshold function)
- Μια διαδικασία εκμάθησης για την προσαρμογή των συντελεστών βαρύτητας των συνδέσεων

Ας θεωρήσουμε μια συνάρτηση κατωφλίου, η οποία συγκρίνει το σταθμισμένο άθροισμα των εισόδων με ένα κατώφλι θ, δηλαδή:

$$z = \sum_i w_i x_i \geq \theta$$

Αφαιρούμε και τα δύο μέλη της ανισότητας κατά θ και έχουμε:

$$z = \sum_i w_i x_i - \theta \geq -\theta + \theta = 0$$

Αν αντικαταστήσουμε το $-\theta$ με b τότε έχουμε:

$$x = b + \sum_i w_i x_i \geq 0 \tag{1}$$

Η Εξ.1 θα καλείται aggregation function

Συνεχίζουμε ορίζοντάς την συνάρτηση κατωφλιού(threshold function) και η οποία ορίζεται ως εξής:

$$\hat{y} = f(z) = \begin{cases} +1, \text{ αν } z > 0 \\ -1, \text{ διαφορετικά} \end{cases}$$

Όπου z είναι η έξοδος της aggregation function που υπολογίσαμε παραπάνω

Ολοκληρώνοντας πρέπει να αναπτύξουμε και μια διαδικασία εκμάθησης για την προσαρμογή των βαρών.

Χρησιμοποιούμε τον κανόνα error-correction learning rule ο οποίος ορίζεται ως εξής:

$$w_{k+1} = w_k + \Delta w_k$$

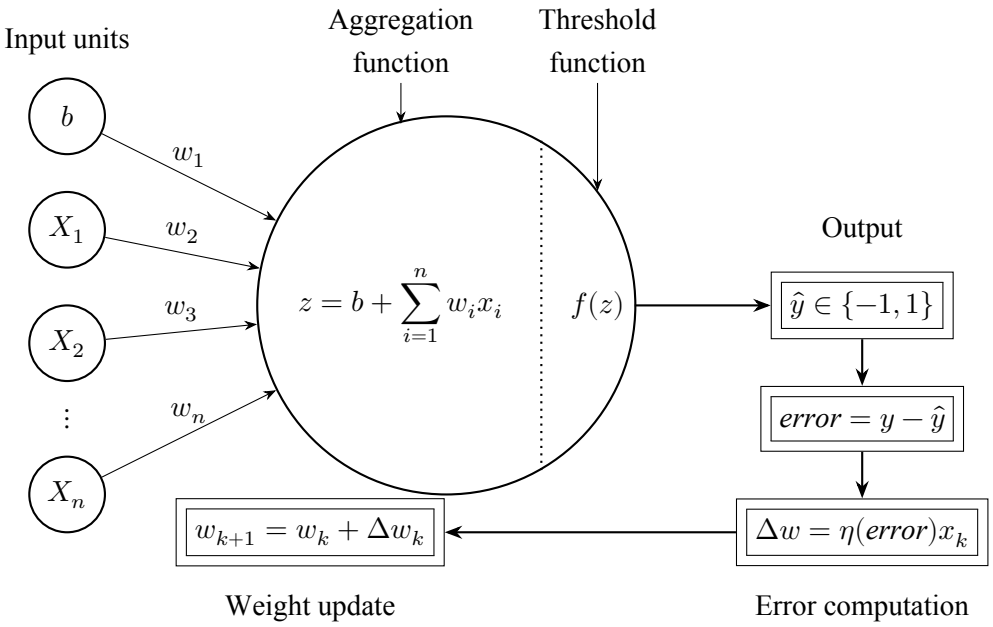
Το Δw_k υπολογίζεται ως εξής :

$$\Delta w_k = \eta(y - \hat{y})x_k$$

Όπου:

- w_k : είναι ο συντελεστής βαρύτητας για την k περίπτωση
- η : είναι ο ρυθμός εκμάθησης[1](τιμές στο διάστημα (0,1))
- y : είναι η πραγματική τιμή("true class")
- $\hat{y}$: είναι η προβλεπόμενη τιμή("predicted class")
- x_κ : είναι το διάνυσμα των εισόδων για την περίπτωση k

Η ακόλουθη εικόνα δείχνει μια σχηματική αναπαράσταση του perceptron με τη διαδικασία εκμάθησης.



Σχηματική Αναπαράσταση του Αλγορίθμου Perceptron

[1]ο ρυθμός εκμάθησης έχει το ρόλο να διευκολύνει τη διαδικασία εκπαίδευσης, σταθμίζοντας το Δ που χρησιμοποιείται για την ενημέρωση των βαρών. Αυτό ουσιαστικά σημαίνει ότι αντί της πλήρους αντικατάστασης του προηγούμενου βάρους με το άθροισμα του βάρους + Δ , ενσωματώνουμε ένα ποσοστό του σφάλματος στη διαδικασία ενημέρωσης καθιστώντας την διαδικασία μάθησης σταθερότερη

Ξεκινάμε την υλοποίησή μας διαβάζοντας τα ακόλουθα δεδομένα:

	ω_1		ω_2		ω_3		ω_4	
Δείγμα	x_1	x_2	x_1	x_2	x_1	x_2	x_1	x_2
1	0.1	1.1	7.1	4.2	-3	-2.9	2	-8.4
2	6.8	7.1	-1.4	-4.3	0.5	8.7	-8.9	0.2
3	-3.5	-4.1	4.5	0	2.9	2.1	-4.2	-7.7
4	2	2.7	6.3	1.6	-0.1	5.2	-8.5	-3.2
5	4.1	2.8	4.2	1.9	-4	2.2	-6.7	-4
6	3.1	5	1.4	-3.2	-1.3	3.7	-0.5	-9.2
7	-0.8	-1.3	2.4	-4	-3.4	6.2	-5.3	-6.7
8	0.9	1.2	2.5	-6.1	-4.1	3.4	-8.7	-6.4
9	5	6.4	8.4	3.7	-5.1	1.6	-7.1	-9.7
10	3.9	4	4.1	-2.2	1.9	5.1	-8	-6.3

Table 1: Δεδομένα για τον αλγόριθμο Perceptron

Απεικόνιση Δειγμάτων

Απεικονίζουμε τα δείγματά μας και λαμβάνουμε την εξής εικόνα:

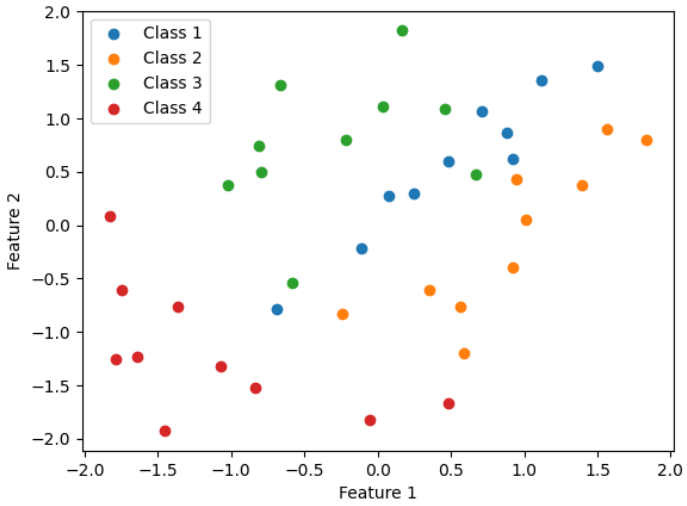


Figure 1: Απεικόνιση των δειγμάτων του πίνακα 1

Υλοποίηση αλγορίθμου Perceptron και απεικόνιση αποτελεσμάτων

Παράλληλα, υλοποιούμε τις συναρτήσεις που αναλύσαμε προηγουμένως σε python και έχουμε:

normalize_features

```
def normalize_features(X):
'''Normalize features to zero mean and unit variance'''
mean = np.mean(X, axis=0)
std = np.std(X, axis=0)
return (X - mean) / std
```

zero_weights

```
def zero_weights(n_features):
    '''Create vector of zero weights'''
    return np.zeros(1 + n_features)  # Add 1 for the bias term
```

net_input

```
def net_input(X, w):
'''Compute net input as dot product'''
return np.dot(X, w[1:]) + w[0]
```

predict

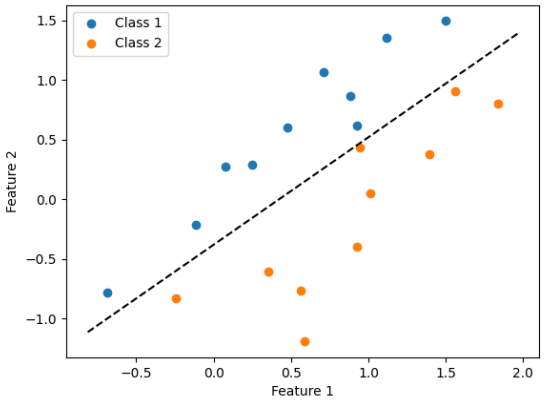
```
def predict(X, w):
'''Return class label after unit step'''
return np.where(net_input(X, w) >= 0.0, 1, -1)
```

fit_batch

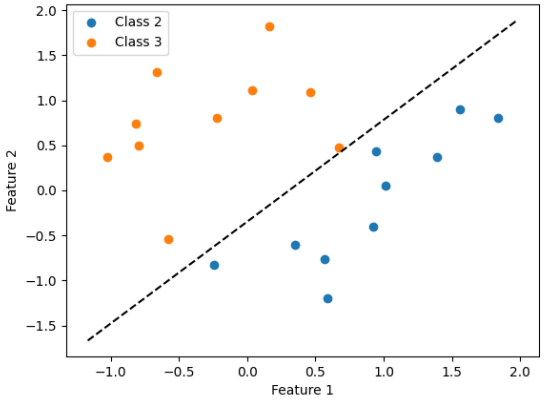
```
def fit_batch(X, y, eta=0.05, n_iter=100):
    '''Batch form of the Perceptron algorithm'''
    iterations = 0
    errors = []
    w = zero_weights(X.shape[1])
    for _ in range(n_iter):
        output = net_input(X, w)
        errors_vector = y - np.where(output >= 0.0, 1, -1)
        if np.all(errors_vector == 0):
            break
        delta_w = eta * np.dot(errors_vector, X)
        w[1:] += delta_w
        w[0] += eta * errors_vector.sum()
        error = np.count_nonzero(errors_vector)
        errors.append(error)
        iterations += 1
```

```
    return w, errors, iterations
```

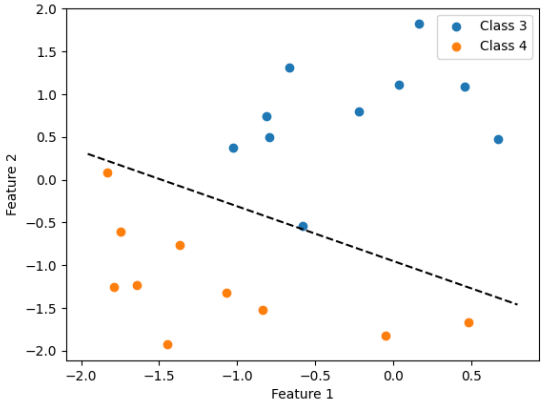
Έχοντας υλοποιήσει τον αλγόριθμο, δοκιμάζουμε την αποτελεσματικότητά του διαδοχικά στα ζεύγη κλάσεων $<\omega_1, \omega_2>$, $<\omega_2, \omega_3>$ και $<\omega_3, \omega_4>$, λαμβάνοντας τις ακόλουθες εικόνες:



(a) Ταξινόμηση των δειγμάτων των κλάσεων ω_1, ω_2



(b) Ταξινόμηση των δειγμάτων των κλάσεων ω_2, ω_3



(c) Ταξινόμηση των δειγμάτων των κλάσεων ω_3, ω_4

Figure 2: Εφαρμογή του Perceptron πάνω σε διαδοχικά ζεύγη κλάσεων

Παράλληλα για να αξιολογήσουμε καλύτερα την αποδοτικότητα του αλγορίθμου σχεδιάζουμε και τον αριθμό των σφαλμάτων συναρτήσει των βημάτων επανάληψης(”time steps”) και έχουμε τις ακόλουθες εικόνες:

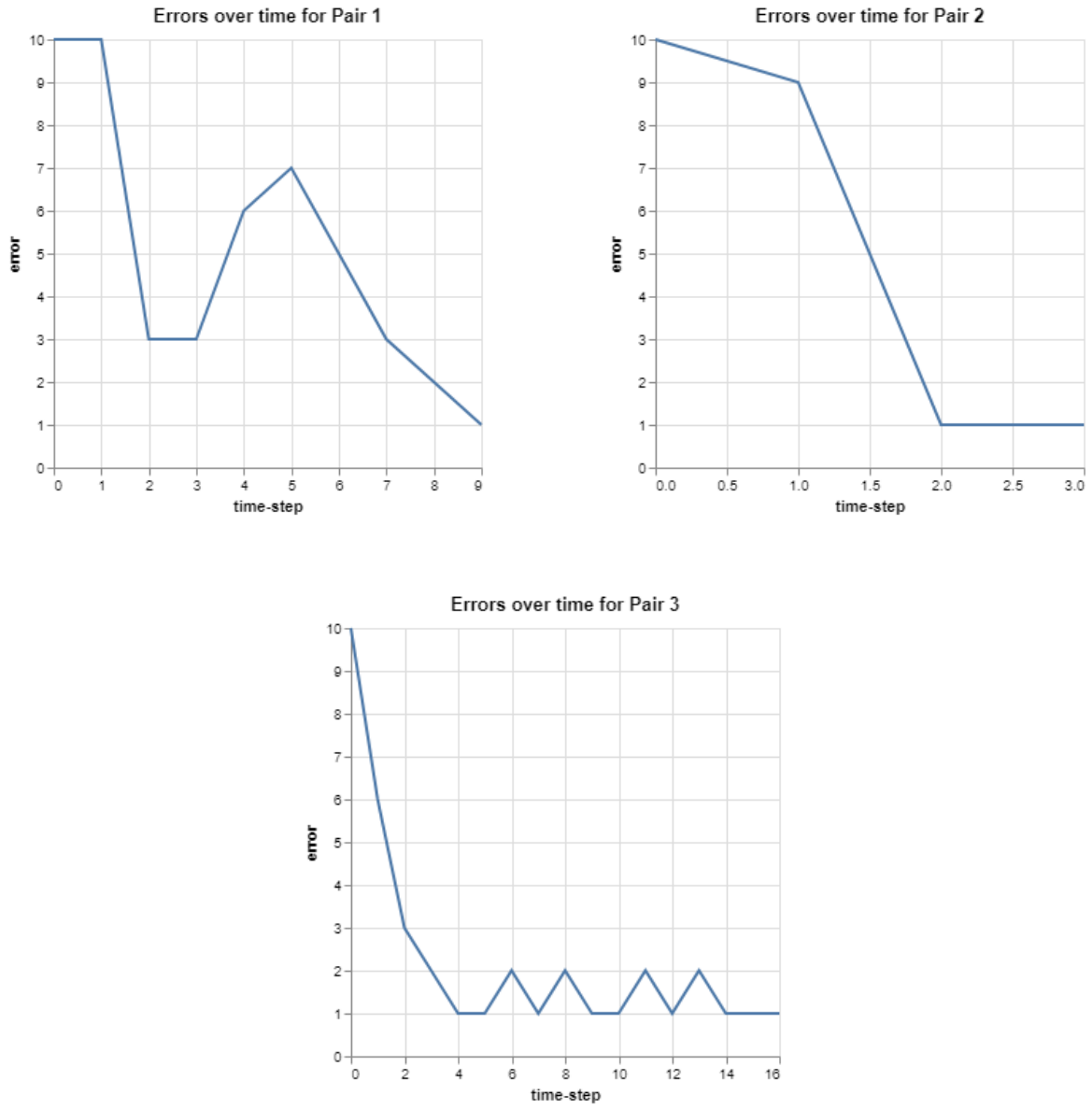


Figure 3: Γραφική αναπαράσταση του σφάλματος του αλγορίθμου Perceptron σε σχέση με τις επαναλήψεις

Σύγκριση και Αξιολόγηση Αποτελεσμάτων

Ολοκληρώνουμε, παραθέτοντας τα αποτελέσματά μας στον ακόλουθο πίνακα:

Ζεύγος κλάσεων	Ακρίβεια	Αριθμ. Επαναλήψεων	Ρυθμός Εκμάθησης
ω_1, ω_2	100%	10	0.05
ω_2, ω_3	100%	4	0.05
ω_3, ω_4	100%	17	0.05

Table 2: Συγκεντρωτικά Αποτελέσματα Perceptron

Παρατηρούμε λοιπόν, διαφοροποιήσεις στον αριθμό των επαναλήψεων που απαιτούνται για κάθε ζεύγος κλάσεων. Αυτές οι διακυμάνσεις οφείλονται κυρίως στη διαχωρισιμότητα των δειγμάτων και στον ρυθμό μάθησης. Συγκεκριμένα, εάν ένα ζεύγος κλάσεων είναι γραμμικά διαχωρίσιμο με σαφές περιθώριο, ο αλγόριθμος συγκλίνει γρήγορα. Ένα αξιοσημείωτο παράδειγμα είναι το ζεύγος $<\omega_2, \omega_3>$, το οποίο απαιτεί μόνο 4 επαναλήψεις για να συγκλίνει ο αλγόριθμος.

Εκτός από τη διαχωρισιμότητα των κλάσεων, ο ρυθμός μάθησης (η) διαδραματίζει επίσης κρίσιμο ρόλο. Ένας σχετικά υψηλός ρυθμός μάθησης μπορεί να οδηγήσει σε ταχύτερη σύγκλιση όταν οι κλάσεις είναι καλά διαχωρισμένες. Ωστόσο, με επικαλυπτόμενες κλάσεις, ένας υψηλός ρυθμός μάθησης μπορεί να προκαλέσει ταλαντώσεις ή βραδύτερη σύγκλιση λόγω υπερβολικά μεγάλων ενημερώσεων.

Θέμα 2: Λογιστική Παλινδρόμηση - Αναλυτική εύρεση κλίσης

Εύρεση κλίσης σφάλματος για το j-στοιχείο

Η συνάρτηση λογιστικής παλινδρόμησης ορίζεται ως εξής:

$$h_\theta(\mathbf{x}) = f(\theta^T \mathbf{x}) \tag{2}$$

Η λογιστική συνάρτηση ορίζεται ως εξής:

$$f(z) = \frac{1}{1 + 2e^{-z}} \tag{3}$$

Συνεπώς, η συνάρτηση λογιστικής παλινδρόμησης ορίζεται και ως:

$$h_\theta(\mathbf{x}) = \frac{1}{1 + 2e^{-\theta^T \mathbf{x}}} \tag{4}$$

Η συνάρτηση κόστους/σφάλματος που καλείται και cross-entropy ορίζεται και ως:

$$\begin{aligned}
J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} \left(-y^{(i)} \ln(\hat{y}^{(i)}) - (1 - y^{(i)}) \ln(1 - \hat{y}^{(i)}) \right) \iff \\
J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} \left(-y^{(i)} \ln(\hat{h_\theta(\mathbf{x})}^{(i)}) - (1 - y^{(i)}) \ln(1 - \hat{h_\theta(\mathbf{x})}^{(i)}) \right) \iff \\
J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \ln\left(\frac{1}{1 + 2e^{-\theta^T \mathbf{x}^{(i)}}} \right) - (1 - y^{(i)}) \ln\left(1 - \frac{1}{1 + 2e^{-\theta^T \mathbf{x}^{(i)}}} \right) \right] \iff \\
J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \ln\left(\frac{1}{1 + 2e^{-\theta^T \mathbf{x}^{(i)}}} \right) - (1 - y^{(i)}) \ln\left(\frac{2e^{-\theta^T x^{(i)}}}{1 + 2e^{-\theta^T x^{(i)}}} \right) \right]
\end{aligned} \tag{5}$$

Από ιδιότητες λογαρίθμων έχουμε τις εξής ισοδυναμίες:

1. $\ln\left(\frac{1}{1+2e^{-\theta^T\mathbf{x}^{(i)}}}\right) = -\ln(1+2e^{-\theta^T\mathbf{x}^{(i)}})$

2. $\ln\left(\frac{2e^{-\theta^T x^{(i)}}}{1+2e^{-\theta^T x^{(i)}}}\right) = \ln(2e^{-\theta^T x^{(i)}}) - \ln(1+2e^{-\theta^T x^{(i)}}) = -\theta^T x^{(i)} + \ln(2) - \ln(1+2e^{-\theta^T x^{(i)}})$

Συνεπώς η Εξ.5 ισοδύναμα μπορεί να γραφτεί και ως:

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\ln(1+2e^{-\theta^T\mathbf{x}^{(i)}}) + (1-y^{(i)})\left(\theta^T\mathbf{x}^{(i)} + \ln(2) - \ln(1+2e^{-\theta^T\mathbf{x}^{(i)}})\right)\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[\ln(1+2e^{-\theta^T\mathbf{x}^{(i)}}) + (1-y^{(i)})(\theta^T\mathbf{x}^{(i)} + \ln(2))\right] \end{aligned} \tag{6}$$

Επειδή όμως $\ln(2) \approxeq 0$ η παραπάνω έκφραση απλοποιείται ως εξής:

$$\frac{1}{m}\sum_{i=1}^{m}\left[\ln(1+2e^{-\theta^T\mathbf{x}^{(i)}}) + (1-y^{(i)})(\theta^T\mathbf{x}^{(i)})\right] \tag{7}$$

Υπολογίζουμε την μερική παράγωγο, $\frac{\partial J(\theta)}{\partial\theta_j}$, και έχουμε:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial\theta_j} &= \frac{1}{m}\sum_{i=1}^{m}\left[\frac{\partial}{\partial\theta_j}\left(\ln(1+2e^{-\theta^T\mathbf{x}^{(i)}}) + (1-y^{(i)})(\theta^T\mathbf{x}^{(i)}\right)\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[\frac{\partial}{\partial\theta_j}\left(\ln(1+2e^{-\theta^T\mathbf{x}^{(i)}})\right) + \frac{\partial}{\partial\theta_j}\left((1-y^{(i)})(\theta^T\mathbf{x}^{(i)})\right)\right] \end{aligned} \tag{8}$$

Υπολογίζουμε τις μερικές παραγώγους και έχουμε:

- $\frac{\partial}{\partial\theta_j}\ln(1+2e^{-\theta^T x^{(i)}}) = \frac{1}{1+2e^{-\theta^T x^{(i)}}}\cdot(-2x_j^{(i)}\cdot e^{-\theta^T x^{(i)}}) = \frac{-2x_j^{(i)}\cdot e^{-\theta^T x^{(i)}}}{1+2e^{-\theta^T x^{(i)}}}$
- $\frac{\partial}{\partial\theta_j}((1-y^{(i)})(\theta^T\mathbf{x}^{(i)})) = (1-y^{(i)}x_j^{(i)})$

Τελικά έχουμε:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial\theta_j} &= \frac{1}{m}\sum_{i=1}^{m}\left[\frac{-2x_j^{(i)}\cdot e^{-\theta^T x^{(i)}}}{1+2e^{-\theta^T x^{(i)}}} + (1-y^{(i)})x_j^{(i)}\right] \iff \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[x_j^{(i)}\cdot\frac{-2e^{-\theta^T x^{(i)}}}{1+2e^{-\theta^T x^{(i)}}} + (1-y^{(i)})x_j^{(i)}\right] \end{aligned} \tag{9}$$

Όμως, $h_\theta(\mathbf{x}) = \frac{1}{1+2e^{-\theta^T\mathbf{x}}}$.

Συνεπώς, έπεται ότι:

$$\boxed{\frac{\partial J(\theta)}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left[(h_\theta(\mathbf{x}^{(i)}) - y^{(i)})x_j^{(i)}\right]} \tag{10}$$

Είτε σε μορφή πινάκων ισοδύναμα έχουμε:

$$\boxed{\text{grad} = \frac{1}{m}X^T(h-y)} \tag{11}$$

Όπου X είναι ο εξής πίνακας:

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

Ανάγνωση και Απεικόνιση δειγμάτων

Στη συνέχεια, χρησιμοποιούμε την λογιστική παλινδρόμηση για να υπολογίσουμε το όριο απόφασης καθώς και να ταξινομήσουμε τα δείγματα μας στις 2 διακριτές κατηγορίες(admitted και not admitted) Αρχικά σχεδιάζουμε τα δείγματά μας και λαμβάνουμε την εξής εικόνα:

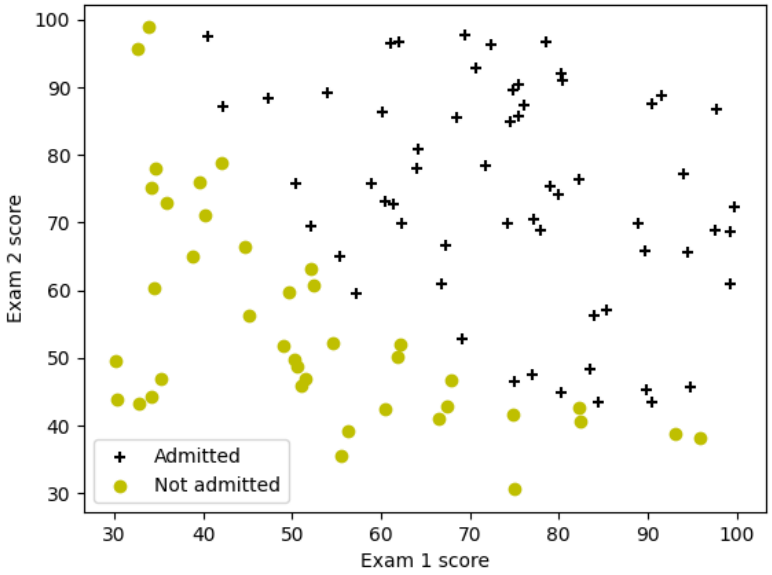


Figure 4: Απεικόνιση δειγμάτων

Συνάρτηση sigmoid

Κατόπιν, συμπληρώνουμε την συνάρτηση sigmoid ώστε να μας επιστρέφει την σιγμοειδή συνάρτηση $f(z)$ της άσκησης.

sigmoid

```
def sigmoid(z):
   sigmoid_function = 1 / (1 + 2*np.exp(-z))
   return sigmoid_function
```

Συναρτήσεις costFunction και gradient

Συνεχίζουμε, συμπληρώνοντας τον κώδικα στις συναρτήσεις costFunction και gradient ώστε να επιστρέφουν το κόστος και την κλίση αντίστοιχα.

costFunction

```
def costFunction(theta, X, y):
    ### ADD YOUR CODE HERE
    m = len(y)
    h = sigmoid(X @ theta)
    J = (1/m) * np.sum(-y * np.log(h) - (1 - y) * np.log(1 - h))
    return J
```

gradient

```
def gradient(theta, X, y):
    ### ADD YOUR CODE HERE
    m = len(y)
    h = sigmoid(X @ theta)
    error = h - y
    grad = (1/m) * (X.T @ error)
    return grad
```

Σύνορο Απόφασης και Αξιολόγηση Αποτελεσμάτων

Ολοκληρώνουμε, υπολογίζοντας το σύνορο απόφασης και προβάλλοντας το μαζί με τα αρχικά μας δείγματα λαμβάνοντας την εξής εικόνα:

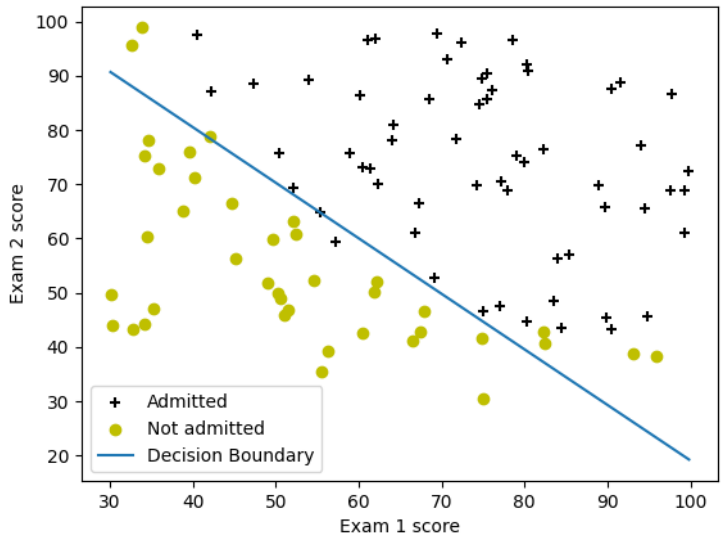


Figure 5: Σύνορο απόφασης

Μπορεί να παρατηρηθεί λοιπόν, ότι ο εν λόγω αλγόριθμος επιτυγχάνει σχετικά υψηλό βαθμό ακρίβειας(accurancy = 89%) ταξινόμησης μεταξύ των δύο κλάσεων, με λίγες μόνο περιπτώσεις σφάλματος που σημειώνονται στην εικόνα.

Ταυτόχρονα, υπολογίζουμε τη βελτιστοποίηση του μοντέλου μας και εξάγουμε τον ακόλουθο πίνακα:

Table 3: Βελτιστοποίηση Λογιστικής Παλινδρόμησης

Παράμετρος	**Τιμή**
Αρχικό κόστος	0.821
Αρχική κλίση	$[-0.267, -22.950, -22.300]$
Κόστος μετά την βελτιστοποίηση	0.203
Κλίση μετά την βελτιστοποίηση	$[-24.468, 0.206, 0.201]$

Είναι φανερό ότι το αρχικό μοντέλο δεν είναι βέλτιστο και απαιτεί περαιτέρω βελτίωση. Το αρχικό κόστος είναι κοντά στη μονάδα, ενώ οι συνιστώσες της κλίσης είναι ιδιαίτερα υψηλές. Προκειμένου να επιτευχθεί σημαντική βελτίωση του μοντέλου, χρησιμοποιήθηκε ο αλγόριθμος Truncated Newton. Το κόστος έχει μειωθεί σημαντικά τείνοντας στο μηδέν, ενώ οι συνιστώσες της κλίσης είναι πλέον κοντά στο μηδέν.

Τέλος, συμπληρώνουμε την συνάρτηση predict για να βρούμε τις πιθανότητες που έχει ένας μαθητής να δίνει δεκτός με συγκεκριμένες βαθμολογίες. Έχουμε λοιπόν:

predict

```
def predict(theta, X):
    ### ADD YOUR CODE HERE
    prob = sigmoid(X @ theta)
    return (prob >= 0.5).astype(int)
```

Συνεπώς, οι πιθανότητες για να γίνει δεκτός ένας μαθητής με βαθμολογίες 45 και 85 είναι: 0.7763 ή 77.63%

Θέμα 3: Εκτίμηση Παραμέτρων με Maximum Likelihood

Μέγιστη Πιθανοφάνεια για κάθε χαρακτηριστικό της κλάσης ω_1

Υπολογίζουμε την μέση τιμή,$\hat{\mu}$, καθώς και την διασπορά $\hat{\sigma}^2$ για κάθε χαρακτηριστικό x_i της κλάσης ω_1 μέσω της μέγιστης πιθανοφάνειας(Maximum Likelihood - MLE). Το ζητούμενο υλοποιείται σε python ως εξής:

calculate_mle

```
def calculate_mle(samples):
    # Mean ( )
```

```
    mu_hat = np.mean(samples, axis=0)

    # Variance ( ^2)
    # axis=0 because we want to calculate the variance of each column
    # ddof=0 because we want to calculate the maximum likelihood
     ↪  estimation -- Delta Degrees of Freedom
    sigma_hat_sq = np.var(samples, axis=0, ddof=0)
    return mu_hat, sigma_hat_sq

mu_hat, sigma_hat_sq = calculate_mle(samples_omega1)
print("Mean (^):", mu_hat)
print("Variance (^^2):", sigma_hat_sq)
```

Και λαμβάνουμε τα ακόλουθα αποτελέσματά:

- **Μέση τιμή ($\hat{\mu}$)**: $[-0.0709, -0.6047, -0.911]$
- **Διασπορά ($\hat{\sigma}^2$)**: $[0.9062, 4.2007, 4.5419]$

Υπολογισμός Παραμέτρων για 2D κατανομή

Για κάθε ζεύγος χαρακτηριστικών της κλάσης ω_1, υπολογίζουμε το διάνυσμα των μέσων τιμών $\hat{\mu}$ καθώς και τον πίνακα συνδιασποράς Σ με την παραδοχή ότι ανά δύο χαρακτηριστικά της κλάσης ω_1 ακολουθούν κανονική κατανομή.Υλοποιούμε το ζητούμενο σε python και έχουμε:

calculate_2d_mle

```
def calculate_2d_mle(samples):
    # Mean ( )
    mu_hat = np.mean(samples, axis=0)

    # Covariance matrix (Σ)
    # rowvar=False because each column represents a variable
    # bias=True for maximum likelihood estimation
    sigma_hat = np.cov(samples, rowvar=False, bias=True)
    return mu_hat, sigma_hat
```

Και λαμβάνουμε τα ακόλουθα αποτελέσματα:

- **Ζεύγος (x_1, x_2)**
  - **Μέση τιμή (μ)**: $[-0.0709, -0.6047]$
  - **Πίνακας συνδιασποράς (Σ)**:

$$\begin{bmatrix} 0.9062 & 0.5678 \\ 0.5678 & 4.2007 \end{bmatrix}$$

- **Ζεύγος (x_1, x_3)**
  - **Μέση τιμή (μ)**: $[-0.0709, -0.911]$
  - **Πίνακας συνδιασποράς (Σ)**:
$$\begin{bmatrix} 0.9062 & 0.3941 \\ 0.3941 & 4.5419 \end{bmatrix}$$
- **Ζεύγος (x_2, x_3)**
  - **Μέση τιμή (μ)**: $[-0.6047, -0.911]$
  - **Πίνακας συνδιασποράς (Σ)**:
$$\begin{bmatrix} 4.2007 & 0.7337 \\ 0.7337 & 4.5419 \end{bmatrix}$$

Υπολογισμός Παραμέτρων για 3D κατανομή

Για ολόκληρο το διάνυσμα των χαρακτηριστικών της κλάσης ω_1, υπολογίζουμε το μέσο διάνυσμα $\hat{\mu}$ και τον πίνακα συνδιακύμανσης $\hat{\Sigma}$ υποθέτοντας ότι το συνολικό διάνυσμα χαρακτηριστικών ακολουθεί 3D κατανομή.Υλοποιούμε το ζητούμενο σε python και έχουμε:

Code for part 3

```
mu_hat_3d, sigma_hat_3d = calculate_2d_mle(samples_omega1)
print("Mean (^) for 3D:", mu_hat_3d)
print("Covariance (Σ) for 3D:", sigma_hat_3d)
```

Και λαμβάνουμε τα ακόλουθα αποτελέσματα:

- **Μέση τιμή ($\hat{\mu}$)**: $[-0.0709, -0.6047, -0.911]$
- **Πίνακας Συνδιασποράς ($\hat{\Sigma}$)**:
$$\begin{bmatrix} 0.9062 & 0.5678 & 0.3941 \\ 0.5678 & 4.2007 & 0.7337 \\ 0.3941 & 0.7337 & 4.5419 \end{bmatrix}$$

Υπολογισμός Παραμέτρων για διαγώνιο πίνακα συνδιασποράς

Για την κλάση ω_2, υποθέτοντας ότι ο πίνακας συνδιακύμανσης είναι διαγώνιος, υπολογίζουμε το μέσο διάνυσμα $\hat{\mu}$ και τα διαγώνια στοιχεία του $\hat{\Sigma}$. Υλοποιούμε το ζητούμενο σε python και έχουμε:

Code for part 4

```
mu_hat_diag, sigma_hat_diag_sq = calculate_mle(samples_omega2)
print("Mean (^) for diagonal Σ:", mu_hat_diag)
print("Variances (^^2) for diagonal Σ:", sigma_hat_diag_sq)
```

Και λαμβάνουμε τα ακόλουθα αποτελέσματα:

- **Μέση τιμή ($\hat{\mu}$)**: $[-0.1126, 0.4299, 0.00372]$
- **Diagonal Variances ($\hat{\sigma}^2$)**: $[0.0539, 0.0460, 0.0073]$

Σύγκριση και Αξιολόγηση Αποτελεσμάτων

Σε αυτό το σημείο συγκρίνουμε τις διαφορετικές μέσες τιμές καθώς και τις διασπορές όπως υπολογίστηκαν από τα διαφορετικά μοντέλα.

- **Διανύσματα μέσης τιμής ($\hat{\mu}$)**:
  - **1D**: $[-0.0709, -0.6047, -0.911]$
  - **2D (x_1, x_2)**: $[-0.0709, -0.6047]$
  - **2D (x_1, x_3)**: $[-0.0709, -0.911]$
  - **2D (x_2, x_3)**: $[-0.6047, -0.911]$
  - **3D**: $[-0.0709, -0.6047, -0.911]$
  - **Διαγώνιος** Σ: $[-0.1126, 0.4299, 0.00372]$
- **Διασπορές ($\hat{\sigma}^2$)**:
  - **1D**: $[0.9062, 4.2007, 4.5419]$
  - **Διαγώνιος** Σ: $[0.0539, 0.0460, 0.0073]$

Σχολιασμός -Επεξήγηση

Οι διαφορές στα αποτελέσματα οφείλονται στις παραδοχές που κάνει κάθε μοντέλο.Συγκεκριμένα:

- Το μοντέλο 1D αντιμετωπίζει κάθε χαρακτηριστικό ανεξάρτητα, με αποτέλεσμα διαφορετικές τιμές διακύμανσης σε σύγκριση με τα μοντέλα που λαμβάνουν υπόψη τις συσχετίσεις των χαρακτηριστικών.
- Τα δισδιάστατα μοντέλα λαμβάνουν υπόψη τις συσχετίσεις μεταξύ ζευγών χαρακτηριστικών, οδηγώντας σε τιμές συνδιακύμανσης που αντικατοπτρίζουν αυτές τις σχέσεις.
- Το τρισδιάστατο μοντέλο λαμβάνει υπόψη ολόκληρο το σύνολο των χαρακτηριστικών και τις αλληλεξαρτήσεις τους, παρέχοντας έναν ολοκληρωμένο πίνακα συνδιακύμανσης.
- Το διαγώνιο μοντέλο συνδιακύμανσης υποθέτει ότι δεν υπάρχει συσχέτιση μεταξύ των χαρακτηριστικών, οδηγώντας σε διαφορετικές εκτιμήσεις για τις διακυμάνσεις και τους μέσους όρους.

Αυτές οι παραλλαγές καταδεικνύουν πώς οι υποκείμενες παραδοχές κάθε μοντέλου επηρεάζουν τις εκτιμώμενες παραμέτρους.

Θέμα 4: Ομαδοποίηση (Clustering) με K-means και GMM

Σε αυτήν την άσκηση θα ασχοληθούμε με την υλοποίηση του αλγόριθμου K-means clustering προκειμένου να συμπιέσουμε μια εικόνα.

Συνάρτηση findClosestCentroids

Ξεκινάμε αρχικά συμπληρώνοντας την συνάρτηση findClosestCentroids η οποία βρίσκει τα $c^{(i)} := j$ που ελαχιστοποιούν την απόσταση $\|x^{(i)} - \mu_j\|^2$ όπου $c^{(i)}$ είναι ο δείκτης του κοντινότερου κέντρου στο $x^{(i)}$,και μ_j είναι το διάνυσμα των τιμών του κέντρου j.Σε επίπεδο python έχουμε:

find_closest_centroids

```
def find_closest_centroids(X, centroids):
    idx = np.zeros(X.shape[0], dtype=int)
    for i in range(X.shape[0]):
        distances = np.linalg.norm(X[i] - centroids, axis=1)
        idx[i] = np.argmin(distances)
    return idx
```

Συνάρτηση computeCentroids

Κατόπιν, συμπληρώνουμε την συνάρτηση computeCentroids η οποία υπολογίζει τα κεντροειδή των κλάσεων και ορίζεται ως εξής:

$$\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

όπου C_k είναι το σύνολο των παραδειγμάτων που έχουν αντιστοιχηθεί στην κλάση kΣε επίπεδο python έχουμε:

compute_centroids

```
def compute_centroids(X, idx, K):
    centroids = np.zeros((K, X.shape[1]))
    for k in range(K):
        points = X[idx == k]
        if points.shape[0] > 0:
            centroids[k] = np.mean(points, axis=0)
    return centroids
```

Παραμετροποιήση μοντέλου

Παράλληλα, επεκτείνουμε το προγραμμά μας ώστε να μπορεί να διαβάσει εικόνες που είναι στην κλίμακα του γκρί καθώς και εικόνες που έχουν το Alpha channel[2](λ.χ png)Αυτό σε επίπεδο κώδικα υλοποιείται ως εξής:

Initialaziton Part

```
# Load the image
image = io.imread('Fruit.png')
```

[2]Το κανάλι Alpha, που συχνά αναφέρεται ως κανάλι διαφάνειας, είναι ένα στοιχείο των ψηφιακών εικόνων που καθορίζει το επίπεδο διαφάνειας ή αδιαφάνειας κάθε εικονοστοιχείου σε μια εικόνα.

```
# Size of the image
img_size = image.shape

# Normalize image values in the range 0 - 1
image = image / 255.0

# Check the number of channels in the image
if image.ndim == 2:
    # If the image is grayscale, duplicate the single channel to make it
    ↪ 3-channel
    image = np.stack((image,)*3, axis=-1)
elif image.shape[2] == 4:
    # If the image has an alpha channel, remove it
    image = image[:, :, :3]

# Reshape the image to be a Nx3 matrix (N = num of pixels)
X = image.reshape(img_size[0] * img_size[1], 3)

# Reshape the image to be a Nx3 matrix (N = num of pixels)
X = image.reshape(img_size[0] * img_size[1], 3)
```

Αποδοτικότητα Αλγορίθμου

Εξετάζουμε την αποδοτικότητά του αλγορίθμου μας με μια εικόνα και λαμβάνουμε τα ακόλουθα αποτελέσματά:

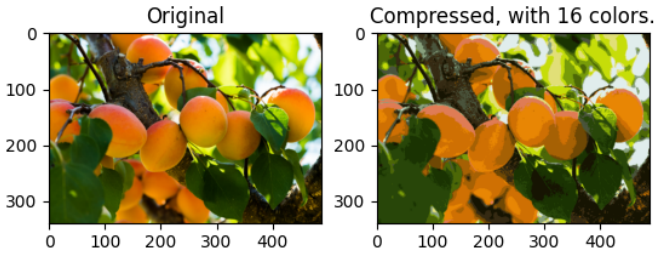
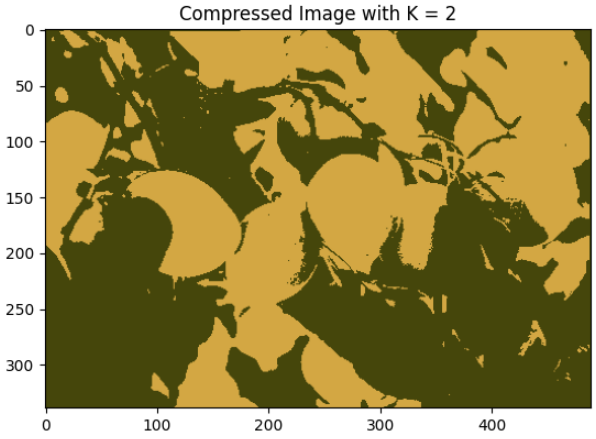
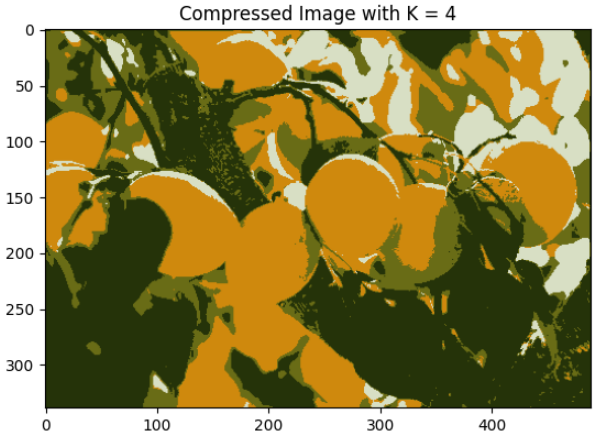


Figure 6: Αποδοτικότητα KMeans αλγορίθμου

Επεκτείνουμε τη δοκιμή του μοντέλου μας εξετάζοντας ποικίλους αριθμούς χαρακτηριστικών, γεγονός που οδηγεί στις ακόλουθες περιπτώσεις:





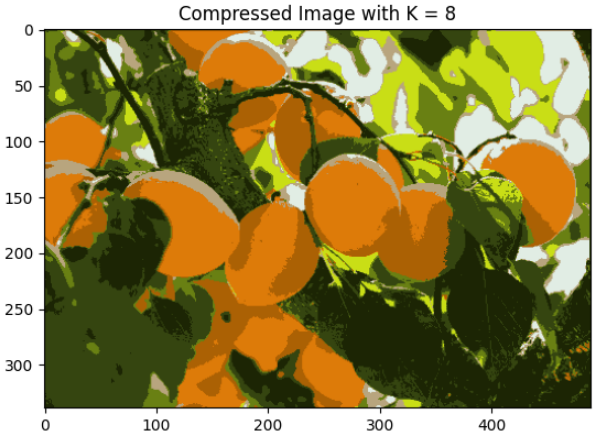


Figure 7: Με σειρά από πάνω προς τα κάτω, οι compressed images χρησιμοποιώντας διαφορετικό αριθμό χαρακτηριστικών

Παρατηρούμε λοιπόν ότι η αύξηση των χαρακτηριστικών/χρωμάτων της εικόνας βελτιώνει την αποδοτικότητα του αλγορίθμου και κατ' επέκτασιν λαμβάνουμε ένα καλύτερο αποτέλεσμα.Παράλληλα,

παραθέτουμε και έναν συγκεντρωτικό πίνακα που δείχνουμε τον βαθμό συμπίεσης σε σχέση με την αρχική εικόνα.

Compressed with K =	Compression Rate:
2	87%
4	77%
8	68%
16	66%
100	42%
200	37%
500	33%

Table 4: Συγκεντρωτικός πίνακας βαθμού συμπίεσης

Νευρωνικά δίκτυα

Α.Υλοποίηση ενός απλού νευρωνικού δικτύου

Σε αυτό το μέρος, υλοποιούμε ένα απλό νευρωνικό δίκτυο χωρίς την χρήση έτοιμων προγραμμάτων.

Μετασχηματισμός της Cross-Entropy Loss Function

Γνωρίζουμε ότι:

$$z^{(i)} = x^{(i)}W + b \tag{12}$$

Καθώς και:

$$\hat{y}^{(i)} = f(z^{(i)}) = \frac{1}{1 + e^{-z^{(i)}}} \tag{13}$$

Η cross entropy για ένα σύνολο Β δειγμάτων ορίζεται ως εξής:

$$J(Y, \hat{Y}; W, b) = \frac{1}{B} \sum_i \left(-y^{(i)} \ln(\hat{y}^{(i)}) - (1 - y^{(i)}) \ln(1 - \hat{y}^{(i)})\right) \tag{14}$$

Όμως:

$$1 - \hat{y}^{(i)} = 1 - \frac{1}{1 + e^{-z^{(i)}}} = \frac{e^{-z^{(i)}}}{1 + e^{-z^{(i)}}} \tag{15}$$

$$\ln(\hat{y}^{(i)}) = \ln\left(\frac{1}{1 + e^{-z^{(i)}}}\right) = -\ln(1 + e^{-z^{(i)}}) \tag{16}$$

$$\ln(1 - \hat{y}^{(i)}) = \ln\left(\frac{e^{-z^{(i)}}}{1 + e^{-z^{(i)}}}\right) = -z^{(i)} - \ln(1 + e^{-z^{(i)}}) \tag{17}$$

Με βάση λοιπόν τις Εξ.15, Εξ.16, Εξ.17 η Εξ.14 μπορεί να γραφτεί και ισοδύναμα ως εξής:

$$\begin{aligned} J(Y,\hat{Y};W,b) &= \frac{1}{B}\sum_i \left(-y^{(i)}(-\ln(1+e^{-z^{(i)}}) - (1-y^{(i)})(-z^{(i)} - \ln(1+e^{-z^{(i)}})\right) \iff \\ &= \sum_i \left(y^{(i)}\ln(1+e^{-z^{(i)}}) + (1-y^{(i)})(z^{(i)} + \ln(1+e^{-z^{(i)}})\right) \iff \\ &= \boxed{\sum_i \left(z^{(i)} - y^{(i)}z^{(i)} + \ln(1+e^{-z^{(i)}})\right) \text{ ο.ε.δ}} \end{aligned}$$

Μερική παράγωγος της cross-entropy loss function ως προς $z^{(i)}$

$$\begin{aligned} \frac{\partial J}{\partial z^{(i)}} &= \frac{\partial}{\partial z^{(i)}}\sum_i \left(z^{(i)} - y^{(i)}z^{(i)} + \ln(1+e^{-z^{(i)}})\right) \\ &= \sum_i \left(\frac{\partial}{\partial z^{(i)}}(z^{(i)} - y^{(i)}z^{(i)})) + \frac{\partial}{\partial z^{(i)}}\ln(1+e^{-z^{(i)}})\right) \end{aligned} \tag{18}$$

Υπολογίζουμε ξεχωριστά τις μερικές παραγώγους και έχουμε:

- $\frac{\partial}{\partial z^{(i)}}(z^{(i)} - y^{(i)}z^{(i)})) = 1 - y^{(i)}$
- $\frac{\partial}{\partial z^{(i)}}\ln(1+e^{-z^{(i)}}) = \frac{1}{1+e^{-z^{(i)}}}\cdot\frac{\partial}{\partial z^{(i)}}(1+e^{-z^{(i)}}) = \frac{-e^{-z^{(i)}}}{1+e^{-z^{(i)}}} = \hat{y}^{(i)} - 1$

Τελικά η Εξ.18 μετασχηματίζεται ως εξής:

$$\boxed{\frac{\partial J}{\partial z^{(i)}} = \hat{y}^{(i)} - 1 + 1 - y^{(i)} = -y^{(i)} + \hat{y}^{(i)}, i \in batch} \tag{19}$$

Εφαρμογή του κανόνα της αλυσίδας

Σύμφωνα με τον κανόνα της αλυσίδας ισχύει ότι:

$$\frac{dx}{dx} = \frac{dz}{dy}\cdot\frac{dy}{dx} \tag{20}$$

Εφαρμόζοντας λοιπόν, τον κανόνα της αλυσίδας στην Εξ.14 για να βρούμε την μερική παράγωγο $\frac{\partial J}{\partial W}$ έχουμε:

$$\frac{\partial J}{\partial W} = \sum_{i=1}\frac{\partial J}{\partial \hat{y}^{(i)}}\cdot\frac{\partial \hat{y}^{(i)}}{\partial z^{(i)}}\cdot\frac{\partial z^{(i)}}{\partial W} \tag{21}$$

Από κανόνες παραγώγισης λαμβάνουμε τα ακόλουθα:

- $\frac{\partial J}{\partial \hat{y}^{(i)}} = -\frac{y^{(i)}}{\hat{y}^{(i)}} + \frac{1-y^{(i)}}{1-\hat{y}^{(i)}}$
- $\frac{\partial \hat{y}(i)}{\partial z^{(i)}} = \hat{y}^{(i)}(1-\hat{y}^{(i)})$
- $\frac{\partial z^{(i)}}{\partial W} = x^{(i)}$

Τελικά η Εξ.21 γράφεται:

$$\boxed{\frac{\partial J}{\partial W} = \sum_{i=1} \left(\left(-\frac{y^{(i)}}{\hat{y}^{(i)}} + \frac{1-y^{(i)}}{1-\hat{y}^{(i)}} \right) \cdot \hat{y}^{(i)}(1-\hat{y}^{(i)}) \cdot x^{(i)} \right)} \tag{22}$$

Όμοια για την μερική παράγωγο $\frac{\partial J}{\partial b}$ έχουμε:

$$\frac{\partial J}{\partial b} = \sum_{i=1} \frac{\partial J}{\partial \hat{y}^{(i)}} \cdot \frac{\partial \hat{y}^{(i)}}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial b} \tag{23}$$

Από κανόνες παραγώγισης λαμβάνουμε τα ακόλουθα αποτελέσματα:

- $\frac{\partial J}{\partial \hat{y}^{(i)}} = -\frac{y^{(i)}}{\hat{y}^{(i)}} + \frac{1-y^{(i)}}{1-\hat{y}^{(i)}}$
- $\frac{\partial \hat{y}(i)}{\partial z^{(i)}} = \hat{y}^{(i)}(1-\hat{y}^{(i)})$
- $\frac{\partial z^{(i)}}{\partial b} = 1$

Τελικά η Εξ.23 γράφεται:

$$\boxed{\frac{\partial J}{\partial b} = \sum_{i=1} \left(\left(-\frac{y^{(i)}}{\hat{y}^{(i)}} + \frac{1-y^{(i)}}{1-\hat{y}^{(i)}} \right) \cdot \hat{y}^{(i)}(1-\hat{y}^{(i)}) \right)} \tag{24}$$

Περιγραφή οπισθοδρόμησης για νευρωνικό δίκτυο πολλαπλών επιπέδων

Ας θεωρήσουμε ένα επίπεδο του νευρωνικού δικτύου, έστω l τέτοιο ώστε:

- H_{l-1} ορίζει το διάνυσμα εισόδου
- W_l ορίζει τον πίνακα των βαρών
- b_l ορίζει το διάνυσμα των όρων πόλωσης
- H_l ορίζει το διάνυσμα εξόδου

A.Forward Pass:

- Το επίπεδο λαμβάνει ως είσοδο το διάνυσμα X του νευρωνικού δικτύου
- Υπολογίζουμε το Z^l όπου $Z^l = XW^l + b^l$
- Εφαρμόζουμε την συνάρτηση ενεργοποίησης και έχουμε: $H^{(l)} = \sigma(Z^{(l)})$ όπου $\sigma = \frac{1}{1+e^{-z}}$ δηλαδή την λογιστική συνάρτηση.
- Θέτουμε ως X πλέον το $H^{(l)}$

B.Backward computations:

- Υπολογίζουμε τον όρο σφάλματος $\delta^{(l)} = \left(\delta^{(l+1)}(W^{(l+1)})^T\right) \odot \sigma'(Z^{(l)})$ όπου $\sigma'(Z^{(l)}) = \sigma(Z^{(l)})(1-\sigma(Z^{(l)}))$
- "Ενημερώνουμε" τα βάρη και έχούμε: $W^{(l)} := W^{(l)} - \alpha \left(H^{(l-1)}\right)^T \delta^{(l)}$
- "Ενημερώνουμε" τους όρους πόλωσης και έχουμε: $b^{(l)} := b^{(l)} - \alpha \sum_{i=1}^{m} \delta_i^{(l)}$

Εκπαίδευση νευρωνικού δικτύου

Σε αυτό το μέρος ασχολούμαστε με την εκπαίδευση ενός νευρωνικού δικτύου χρησιμοποιώντας δείγματα από την βάση δειγμάτων MNIST.Τα δείγματά μας είναι εικόνες μεγέθους 28×28 που απεικονίζουν χειρόγραφα ψηφία αριθμών από το 0 έως το 9.Στόχος του δικτύου είναι να αναγνωρίζει την κλάση που ανήκει κάθε δείγμα ξεχωριστά.

Συμπληρώνουμε κατάλληλα τον ζητούμενο κώδικα και προβαίνουμε σε μια πειραμάτων προκειμένου να καταλήξουμε σε ένα βέλτιστο δίκτυο.

Συνοψίζουμε τα πειράματά μας στον ακόλουθο πίνακα:

Experiment	**Learning Rate**	**Num of Epochs**	**Batch Size**	**Activation Function**	**MSE**	**Ratio**
1	0.1	100	128	Sigmoid	0.0361	0.75
2	0.1	100	128	Tanh	0.041	0.75
3	0.1	50	128	Sigmoid	0.0285	0.80
4	0.01	100	64	Sigmoid	0.0277	0.85
5	0.01	100	64	Tanh	0.0508	0.75
6	0.001	100	64	Sigmoid	0.0447	0.70
7	0.001	200	64	Tanh	0.0294	0.80
8	0.001	200	32	Tanh	0.0458	0.65
9	0.001	50	64	Sigmoid	0.0578	0.55
10	0.001	50	64	Tanh	0.0842	0.40
11	0.001	100	64	Sigmoid	0.0540	0.60
12	0.001	100	64	Tanh	0.0763	0.50
13	0.01	100	64	Sigmoid	0.0287	0.80
14	0.01	100	64	Tanh	0.0620	0.50

Table 5: Πειραματικοί συνδυασμοί και αποτελέσματα

Σχολιασμός των αποτελεσμάτων

Τα πειράματα, τα οποία παρουσιάζονται σε μορφή πίνακα, παρουσιάζουν σημαντικές πληροφορίες σχετικά με την επίδραση των διαφόρων υπερπαραμέτρων και των αρχιτεκτονικών του δικτύου σε μετρικές επιδόσεων, όπως το μέσο τετραγωνικό σφάλμα (MSE) και ο λόγος ακρίβειας. Τα πειράματα μπορούν να χωριστούν σε δύο ομάδες: εκείνα με μια απλούστερη αρχιτεκτονική δικτύου δύο επιπέδων (πειράματα 1-8) και εκείνα με μια πιο σύνθετη αρχιτεκτονική δικτύου πέντε επιπέδων[3] (πειράματα 9-14).

Όσον αφορά την απλούστερη αρχιτεκτονική δύο επιπέδων, ιδιαίτερο ενδιαφέρον παρουσιάζει το πείραμα 4. Το πείραμα αυτό χρησιμοποιώντας ρυθμό μάθησης 0,01, 100 epoch και batch size 64 παρουσίασε το χαμηλότερο MSE (0,0277) και τον υψηλότερο λόγο (0,85), υποδεικνύοντας ένα καλά ισορροπημένο μοντέλο. Αντίθετα, το πείραμα 5, το οποίο χρησιμοποίησε διαφορετική συνάρτηση

[3]Στο τέλος της αναφοράς υπάρχει αναλυτική περιγραφή και των δύο αρχιτεκτονικών

ενεργοποίησης παρουσίασε μικρότερο λόγο και υψηλότερη τιμή στο MSE.

Η αρχιτεκτονική δικτύου πέντε επιπέδων, η οποία ήταν πιο πολύπλοκη από την προηγούμενη, εμφάνισε ευρύτερο φάσμα αποτελεσμάτων. Αυτό υποδεικνύει ότι τα βαθύτερα δίκτυα είναι πιο ευαίσθητα στη ρύθμιση των υπερπαραμέτρων. Το πείραμα 7, με ρυθμό μάθησης 0,001, 200 epoch και batch size 64 χρησιμοποιώντας τη συνάρτηση ενεργοποίησης Tanh, επέδειξε ικανοποιητική απόδοση με MSE 0,0294 και λόγο 0,80. Ωστόσο, το Πείραμα 10, το οποίο είχε παρόμοια ρύθμιση αλλά μόνο 50 epoch, είχε κακές επιδόσεις με MSE 0,0842 και λόγο 0,40. Αυτό αποδεικνύει την αναγκαιότητα επαρκούς διάρκειας εκπαίδευσης για βαθύτερα δίκτυα.

Νευρωνικά δίκτυα για αναγνώριση εικόνων

Σε αυτό το μέρος, θα ασχοληθούμε με διάφορους ταξινομητές από την βιβλιοθήκη tensorflow/keras χρησιμοποιώντας το σύνολο των δεδομένων Fashion-MNIST.Προκειμένου να υλοποιήσουμε το νευρωνικό δίκτυο συμπληρώνουμε κατάλληλα τον κώδικα και έχουμε:

Code for Convolutional Neural Networks

```
# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

from plots import plot_some_data, plot_some_predictions


fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
 ↪  fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
 ↪  'Coat','Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Scale these values to a range of 0 to 1 before feeding them to the
 ↪  neural network model.
    ### YOUR CODE HERE
train_images = train_images / 255.0
test_images = test_images / 255.0

plot_some_data(train_images, train_labels, class_names)

# Build the model of dense neural network
# Building the neural network requires configuring the layers of the model,
 ↪  then compiling the model.
```

```
# Define the input layer based on the shape of the images
# Then define two dense layers.
# The hidden layer with 128 neurons and RELU activation
# The output layer with 10 neurons and linear activation.

model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),  # Assuming the images are 28x28
     ↪ pixels
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')  # Assuming there are 10
     ↪ classes
])

optimizer_list = ['adam', 'sgd', 'rmsprop', 'adamax', 'nadam', 'ftrl']


# Compile the model
# Before the model is ready for training, it needs a few more settings.
 ↪ These are added during the model's compile step:
# Loss function -This measures how accurate the model is during training.
 ↪ You want to minimize this function to "steer" the model in the right
 ↪ direction.
# Optimizer -This is how the model is updated based on the data it sees
 ↪ and its loss function.


# Loop over each optimizer
for optimizer in optimizer_list:
#Compile the model with the current optimizer
model.compile(optimizer=optimizer,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])

    # Train the model
    model.fit(train_images, train_labels, epochs=400,
                        validation_data=(test_images, test_labels))

    # Evaluate accuracy
    test_loss, test_acc = model.evaluate(test_images,  test_labels,
     ↪ verbose=2)
```

```
    # Print an appropriate message
    print(f"Training with {optimizer} finished. Test loss: {test_loss},
     ↪  Test accuracy: {test_acc}")
    print("################################")

probability_model = tf.keras.Sequential([model,
                                 tf.keras.layers.Softmax()])

predictions = probability_model.predict(test_images)

plot_some_predictions(test_images, test_labels, predictions, class_names,
 ↪  num_rows=5, num_cols=3)
```

Για κάθε optimizer που δοκιμάζουμε δημιουργούμε ένα διάγραμμα με το accurancy και το validation accurancy συναρτήση των epochos όπου το μεν πρώτο αφορά την ακρίβεια στα δεδομένα εκπαίδευσης ενώ το δεύτερο αφορά την ακρίβεια σε δεδομένα επικύρωσης, δηλαδή δεδομένα άγνωστα για το νευρωνικό δίκτυο.

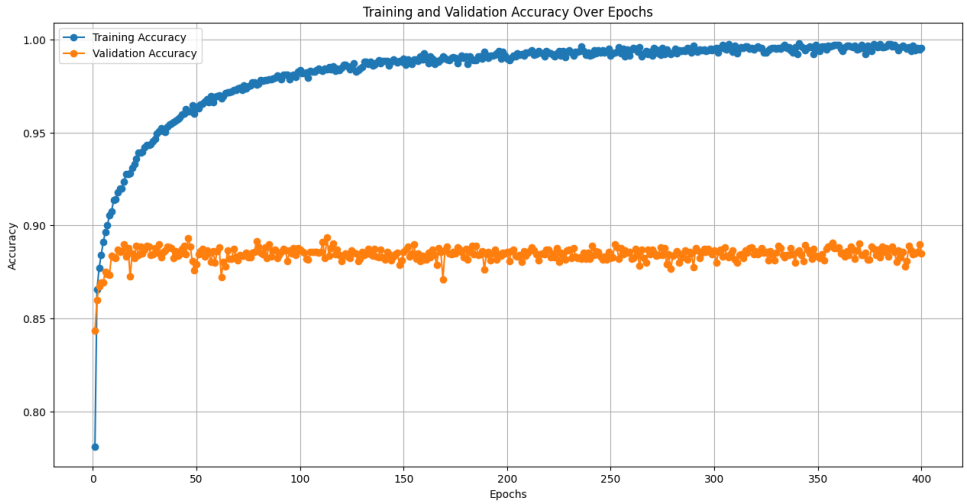


Figure 8: Σύγκριση τιμών ακρίβειας στα δεδομένα εκπαίδευσης με τιμές ακρίβειας σε δεδομένα επικύρωσης στον αλγόριθμο adam

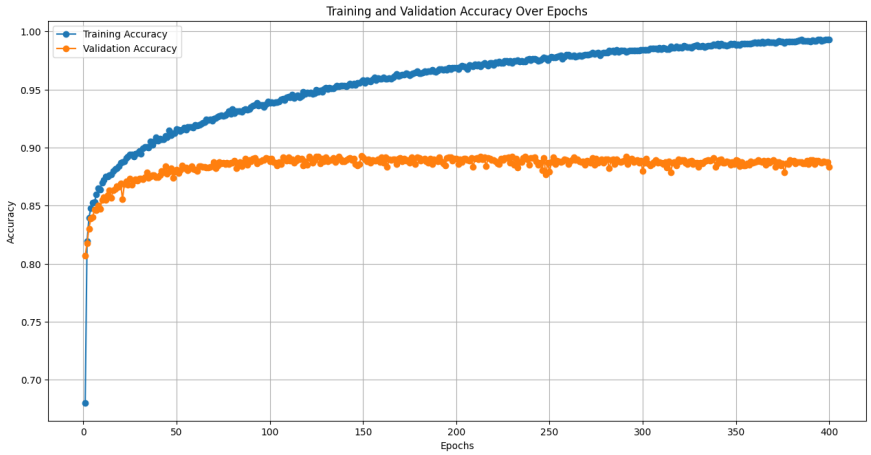


Figure 9: Σύγκριση τιμών ακρίβειας στα δεδομένα εκπαίδευσης με τιμές ακρίβειας σε δεδομένα επικύρωσης στον αλγόριθμο sgd

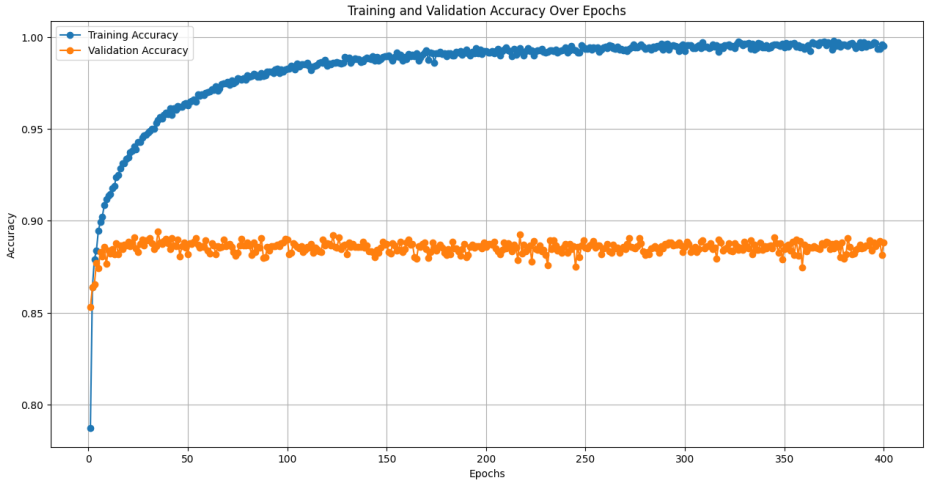


Figure 11: Σύγκριση τιμών ακρίβειας στα δεδομένα εκπαίδευσης με τιμές ακρίβειας σε δεδομένα επικύρωσης στον αλγόριθμο NAdam

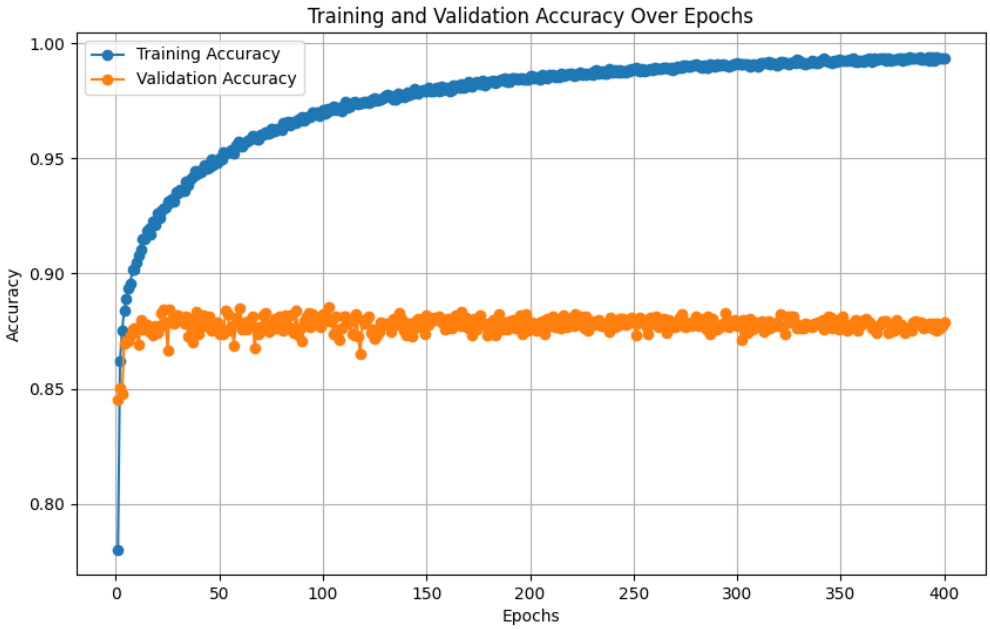


Figure 10: Σύγκριση τιμών ακρίβειας στα δεδομένα εκπαίδευσης με τιμές ακρίβειας σε δεδομένα επικύρωσης στον αλγόριθμο MSRProp

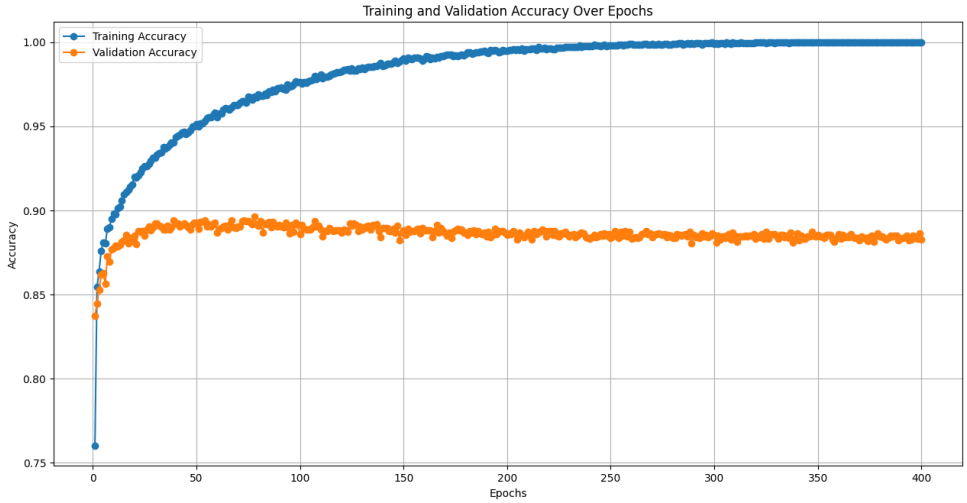


Figure 12: Σύγκριση τιμών ακρίβειας στα δεδομένα εκπαίδευσης με τιμές ακρίβειας σε δεδομένα επικύρωσης στον αλγόριθμο AdamMax

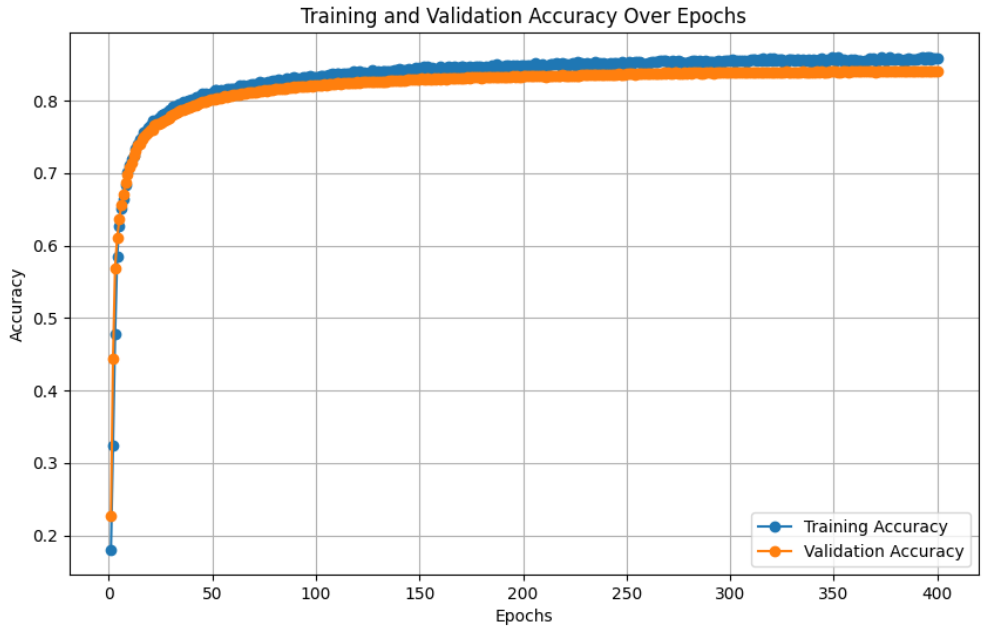


Figure 13: Σύγκριση τιμών ακρίβειας στα δεδομένα εκπαίδευσης με τιμές ακρίβειας σε δεδομένα επικύρωσης στον αλγόριθμο Ftrl

Παράλληλα, συγκεντρώνουμε τα δεδομένα από το τελευταίο epoch για να συγκρίνουμε τους αλγόριθμους βελτιστοποίησης.

Table 6: Σύγκριση αλγορίθμων

Αλγόριθμος	**Epoch**	**Ακρίβεια Εκπαίδευσης**	**Ακρίβεια Επικύρωσης**
Adam	400	0.9953	0.8849
SGD	400	0.9934	0.8834
RMSProp	400	0.9936	0.8788
NAdam	400	0.9952	**0.8882**
AdamMax	400	**0.9999**	0.8826
Ftrl	400	0.8583	0.8408

Σχολιασμός αποδοτικότητας αλγορίθμων

Από την ανάλυση των διαγραμμάτων μπορούν να εξαχθούν διάφορα συμπεράσματα σχετικά με την αποτελεσματικότητα των διαφόρων αλγορίθμων που εφαρμόστηκαν στο μοντέλο νευρωνικού δικτύου. Όλοι οι αλγόριθμοι επιτυγχάνουν υψηλή ακρίβεια εκπαίδευσης, με τον Adamax να φτάνει σχεδόν το 100%, υποδεικνύοντας αποτελεσματική ελαχιστοποίηση των απωλειών εκπαίδευσης. Ωστόσο, με εξαίρεση τον αλγόριθμο FTRL, τα περισσότερα μοντέλα παρουσιάζουν υπερπροσαρμογή, όπως αποδεικνύεται από την αξιοσημείωτη απόκλιση μεταξύ των ακρίβειας εκπαίδευσης και επικύρωσης. Αυτό υποδηλώνει ότι ενώ τα μοντέλα είναι εξαιρετικά ακριβή στα δεδομένα εκπαίδευσης, δεν

γενικεύονται καλά σε νέα δεδομένα, με το RMSprop να παρουσιάζει την πιο έντονη υπερπροσαρμογή. Ο αλγόριθμος FTRL εμφανίζει την ταχύτερη σύγκλιση, φτάνοντας νωρίτερα από τους άλλους στην κορυφή της απόδοσης, αν και παρουσιάζει επίσης μεγαλύτερη διακύμανση και πιθανότητα υπερπροσαρμογής. Όσον αφορά τη σταθερότητα, οι NAdam, Adam και Adamax παρέχουν την πιο σταθερή απόδοση, όπως υποδεικνύεται από ομαλότερες καμπύλες ακρίβειας και χαμηλότερη διακύμανση.

Για την καλύτερη κατανόηση αυτών των παρατηρήσεων, είναι χρήσιμο να εξετάσουμε τις μαθηματικές διατυπώσεις που διέπουν κάθε αλγόριθμο. Ο αλγόριθμος Stochastic Gradient Descent (SGD) ενημερώνει τα βάρη επαναληπτικά με βάση την προσέγγιση της κλίσης, ενώ ο RMSprop προσαρμόζει τον ρυθμό μάθησης χρησιμοποιώντας έναν κινητό μέσο όρο των τετραγωνικών κλίσεων για τον μετριασμό των ταλαντώσεων. Ο Adam συνδυάζει τα οφέλη του SGD και του RMSprop χρησιμοποιώντας κινητούς μέσους όρους τόσο των κλίσεων όσο και των τετραγωνικών κλίσεων. Ο NAdam επεκτείνει τον Adam ενσωματώνοντας την ορμή Nesterov για την επιτάχυνση της σύγκλισης. Ο Adamax, μια παραλλαγή του Adam, χρησιμοποιεί την άπειρη νόρμα των προηγούμενων κλίσεων για πιο ισχυρή απόδοση με αραιές κλίσεις. Τέλος, ο FTRL (Follow-the-Regularized-Leader) εφαρμόζει έναν συνδυασμό κανονικοποίησης L1 και L2 στη συνάρτηση απώλειας, προωθώντας τη σπανιότητα και τη σταθερότητα.

Αυτές οι μαθηματικές διευκρινήσεις εξηγούν τα παρατηρούμενα χαρακτηριστικά απόδοσης: Ο Adam και οι παραλλαγές του (NAdam και Adamax) προσφέρουν μια ισορροπημένη προσέγγιση με σταθερή σύγκλιση, ενώ ο RMSprop είναι πιο επιρρεπές στην υπερπροσαρμογή λόγω της επιθετικής προσαρμογής του ρυθμού μάθησης.

Συνελεκτικό Νευρωνικό δίκτυο

Σε αυτήν την άσκηση καλούμαστε να υλοποιήσουμε ένα πιο σύνθετο νευρωνικό δίκτυο το οποίο συνοψίζεται στην ακόλουθη εικόνα:

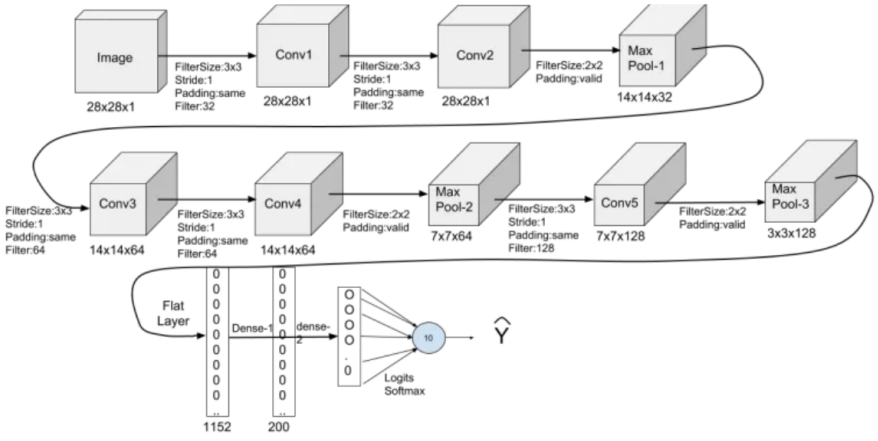


Figure 14: Το προς υλοποίηση νευρωνικό δίκτυο

Υλοποιώντας το σύνθετο νευρωνικό δίκτυο σε επίπεδο python και έχουμε:

Convolution Network

```
from __future__ import absolute_import, division, print_function,
 ↪  unicode_literals

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
 ↪  BatchNormalization, Dropout

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

from plots import plot_some_data, plot_some_predictions

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
 ↪  fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Scale these values to a range of 0 to 1 before feeding them to the
 ↪  neural network model.

train_images = train_images / 255.0
test_images = test_images / 255.0

num_train_images = train_images.shape[0]
num_test_images = test_images.shape[0]
height = 28
width = 28
channels = 1  # grayscale images have 1 channel

train_images_reshaped = train_images.reshape(num_train_images, height,
 ↪  width, channels)
```

```
test_images_reshaped = test_images.reshape(num_test_images, height, width,
 ↪  channels)

# Build the model
# Building the neural network requires configuring the layers of the model,
 ↪  then compiling the model.
model = Sequential([
    Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1)),
    BatchNormalization(),
    keras.layers.ReLU(),

    Conv2D(32, (3, 3), padding='same'),
    BatchNormalization(),
    keras.layers.ReLU(),
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Conv2D(64, (3, 3), padding='same'),
    BatchNormalization(),
    keras.layers.ReLU(),

    Conv2D(64, (3, 3), padding='same'),
    BatchNormalization(),
    keras.layers.ReLU(),
    MaxPooling2D((2, 2)),
    Dropout(0.3),

    Conv2D(128, (3, 3), padding='same'),
    BatchNormalization(),
    keras.layers.ReLU(),
    MaxPooling2D((2, 2)),
    Dropout(0.4),

    Flatten(),
    Dense(200),
    BatchNormalization(),
    keras.layers.ReLU(),
    Dropout(0.5),

    Dense(10, activation='softmax')
])
```

```
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

# Train the model
model.fit(train_images_reshaped, train_labels, epochs=50,
 ↪  validation_data=(test_images_reshaped, test_labels))

# Evaluate accuracy
test_loss, test_acc = model.evaluate(test_images_reshaped, test_labels,
 ↪  verbose=2)

print('\nTest accuracy:', test_acc)

probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])

predictions = probability_model.predict(test_images_reshaped)

plot_some_predictions(test_images, test_labels, predictions, class_names)
```

Οπτικοποιούμε τα δείγματά μας και έχουμε:

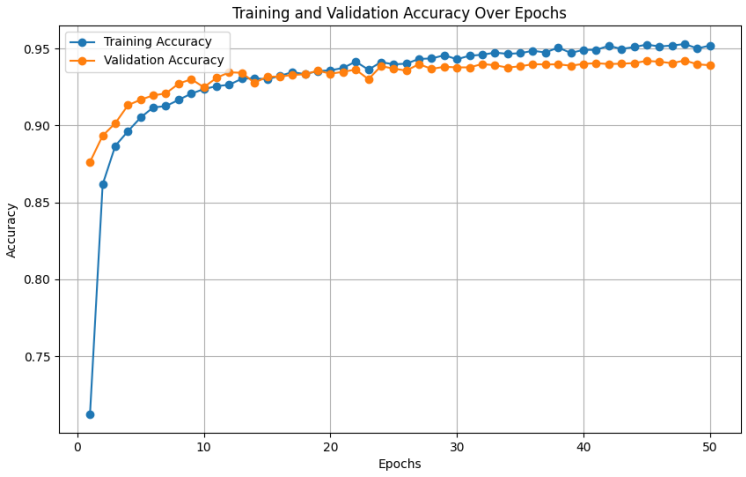


Figure 15: Αποτελεσματικότητα νευρωνικού δικτύου

Σχολιασμός συνελικτικού νευρωνικού δικτύου

Οι παρατηρήσεις μας δείχνουν ότι το νευρωνικό δίκτυο παρουσιάζει υψηλή απόδοση τόσο στα δείγματα εκπαίδευσης όσο και στα δεδομένα επικύρωσης, μειώνοντας επιτυχώς τον κίνδυνο υπερπροσαρμογής.

Για να κατανοήσουμε πλήρως τη λειτουργικότητα του αλγορίθμου μας, είναι απαραίτητο να εξετάσουμε τις τεχνικές λεπτομέρειες της αρχιτεκτονικής του δικτύου.

Τα στρώματα Conv2D αποτελούν τη βάση του νευρωνικού δικτύου μας. Αυτά τα στρώματα εφαρμόζουν είτε 32 είτε 64 φίλτρα, ανάλογα με τη συγκεκριμένη διαμόρφωση, και χρησιμοποιούν ”ίδιο” padding για να διασφαλίσουν ότι οι διαστάσεις εξόδου ταιριάζουν με τις διαστάσεις εισόδου. Αυτή η σχεδιαστική επιλογή διατηρεί τη χωρική ανάλυση των χαρτών χαρακτηριστικών, η οποία είναι κρίσιμη για τη διατήρηση των κρίσιμων πληροφοριών ακμών και υφής σε όλο το δίκτυο.

Το Batch normalization χρησιμοποιείται για την κανονικοποίηση της εξόδου κάθε στρώματος, σταθεροποιώντας και επιταχύνοντας έτσι τη διαδικασία εκπαίδευσης. Με τη διόρθωση της εσωτερικής μετατόπισης των συνδιακυμάνσεων, η κανονικοποίηση δέσμης επιτρέπει υψηλότερους ρυθμούς μάθησης βελτιώνοντας τελικά την απόδοση και τη σύγκλιση του δικτύου.

Τα στρώματα MaxPooling ενσωματώνονται για την υποδειγματοληψία των χωρικών διαστάσεων των χαρτών χαρακτηριστικών επιλέγοντας τη μέγιστη τιμή εντός ενός καθορισμένου παραθύρου, συνήθως μεγέθους 2x2. Αυτή η λειτουργία όχι μόνο μειώνει τον υπολογιστικό φόρτο, αλλά βοηθά επίσης στην ανάδειξη των πιο σημαντικών χαρακτηριστικών, επιτρέποντας την ταχύτερη σύγκλιση του νευρωνικού δικτύου.

Τέλος, η εφαρμογή του dropout regularization είναι ζωτικής σημασίας για την αποφυγή υπερβολικής προσαρμογής του μοντέλου. Το dropout λειτουργεί με την τυχαία μηδενική ρύθμιση ενός κλάσματος των μονάδων εισόδου σε κάθε ενημέρωση κατά τη διάρκεια της εκπαίδευσης, αναγκάζοντας το δίκτυο να μάθει πιο ισχυρά χαρακτηριστικά που δεν εξαρτώνται από συγκεκριμένους νευρώνες. Αυτή η τεχνική βελτιώνει σημαντικά την ικανότητα γενίκευσης του μοντέλου.

Παράρτημα Α

Σε αυτό το παράρτημα παρουσιάζουμε τις αρχιτεκτονικές που αξιοποιήθηκαν για την ανάπτυξη του πρώτου μέρους της άσκησης 5.

Neural Network with 5 layers

```
    network = [
    Dense(28 * 28, 256),
    Sigmoid(),
    Dense(256, 128),
    Sigmoid(),
    Dense(128, 64),
    Sigmoid(),
    Dense(64, 32),
    Sigmoid(),
    Dense(32, 10),
    Softmax()
]
```

Neural Network with 2 layers

```
    network = [
    Dense(28 * 28, 50),
    Sigmoid(),
    Dense(50, 10),
    Softmax()
]
```

Παράρτημα Β

Σκοπός αυτού του παραρτήματος είναι να ρίξει φως σε έννοιες που έχουν γίνει ήδη αναφορά προσφέροντας μια βαθύτερη κατανόηση τους.

Stochastic Gradient Descent (SGD)

- **Gradient Approximation**: Ο SGD ενημερώνει τα βάρη του μοντέλου υπολογίζοντας την κλίση (μερικές παράγωγοι) της συνάρτησης απώλειας σε σχέση με τα βάρη, χρησιμοποιώντας μια μικρή δέσμη δεδομένων εκπαίδευσης. Αυτή η προσέγγιση επιτρέπει ταχύτερες ενημερώσεις σε σύγκριση με τη χρήση ολόκληρου του συνόλου των δεδομένων.
- **Κανόνας ενημέρωσης βαρών**:

  $$w_{t+1} = w_t - \eta \nabla L(w_t)$$

  όπου

  - w_t: τα βάρη κατά την επανάληψη t
  - η: ρυθμός εκμάθησης
  - $\nabla L(w_t)$: κλίση της συνάρτησης απώλειας στο w_t

RMSprop

- **Προσαρμογή ρυθμού μάθησης**: Ο RMSprop προσαρμόζει τον ρυθμό μάθησης για κάθε παράμετρο χρησιμοποιώντας έναν κινητό μέσο όρο των τετραγωνικών κλίσεων για να σταθεροποιήσει τις ενημερώσεις.
- **Κανόνας Ενημέρωσης**:

  $$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

  όπου

  - $E[g^2]_t$: κινητός μέσος όρος των τετραγωνικών κλίσεων
  - ϵ: μικρή σταθερά για την αποφυγή διαίρεσης με το μηδέν
  - g_t: κλίση στην επανάληψη t

Adam (Adaptive Moment Estimation)

- **Συνδυάζει SGD και RMSprop**: Ο Adam διατηρεί τους κινητούς μέσους όρους τόσο των κλίσεων όσο και των τετραγώνων τους.
- **Κανόνας ενημέρωσης**:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \quad \text{(κινητός μέσος όρος των κλίσεων)} \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \text{(κινητός μέσος όρος των τετραγωνικών κλίσεων)} \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \quad \text{(bias-corrected first moment estimate)} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \quad \text{(bias-corrected second moment estimate)} \\
w_{t+1} &= w_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
$$

NAdam (Nesterov-accelerated Adam)

- **Nesterov Momentum**: Επεκτείνει τον Adam ενσωματώνοντας την ορμή Nesterov, η οποία προσαρμόζει τις κλίσεις ώστε να βλέπει μπροστά στην κατεύθυνση του βήματος.
- **Κανόνας ενημέρωσης**: Παρόμοια με τον Adam, αλλά με έναν πρόσθετο όρο ορμής.

Adamax

- **Infinite Norm of Gradients**: Μια παραλλαγή του Adam που χρησιμοποιεί την άπειρη νόρμα (μέγιστη απόλυτη τιμή) των κλίσεων αντί της νόρμας L2 (τετραγωνική τιμή) για πιο ισχυρή απόδοση με αραιές κλίσεις.
- **Κανόνας ενημέρωσης**: Παρόμοια με την Adam, αλλά με χρήση της νόρμας απείρου για την εκτίμηση της δεύτερης ροπής.

FTRL (Follow-the-Regularized-Leader)

- **L1 και L2 Regularization**: Ο FTRL συνδυάζει κανονικοποίηση L1 και L2 στη συνάρτηση απώλειας, προωθώντας τόσο τη σπανιότητα (L1) όσο και τη σταθερότητα (L2).
- **Κανόνας ενημέρωσης**: Περιλαμβάνει όρους κανονικοποίησης που προστίθενται στην τυπική ενημέρωση φθίνουσας κλίσης για την ενθάρρυνση σπανίων λύσεων.

Άπειρη νόρμα

Ανακαλούμε την οικογένεια των p-νορμών για τις οποίες η γενικευμένη μορφή είναι η ακόλουθη:

$$
||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \quad \text{με } p \geq 1 \tag{25}
$$

Έχουμε λοιπόν ότι η άπειρη νόρμα ή νόρμα μεγίστου $\mathbf{R}^n$ ορίζεται ως εξής:

$$
\|x\|_\infty = \max_{1 \leq k \leq n} |x_k| \tag{26}
$$