Στατιστική Μοντελοποίηση και Αναγνώριση Προτύπων
Φυλλάδιο Ασκήσεων 2

Αθανασάκης Ευάγγελος 2019030118
Φραγκογιάννης Γεώργιος 2019030039
Ομάδα Χρηστών 115

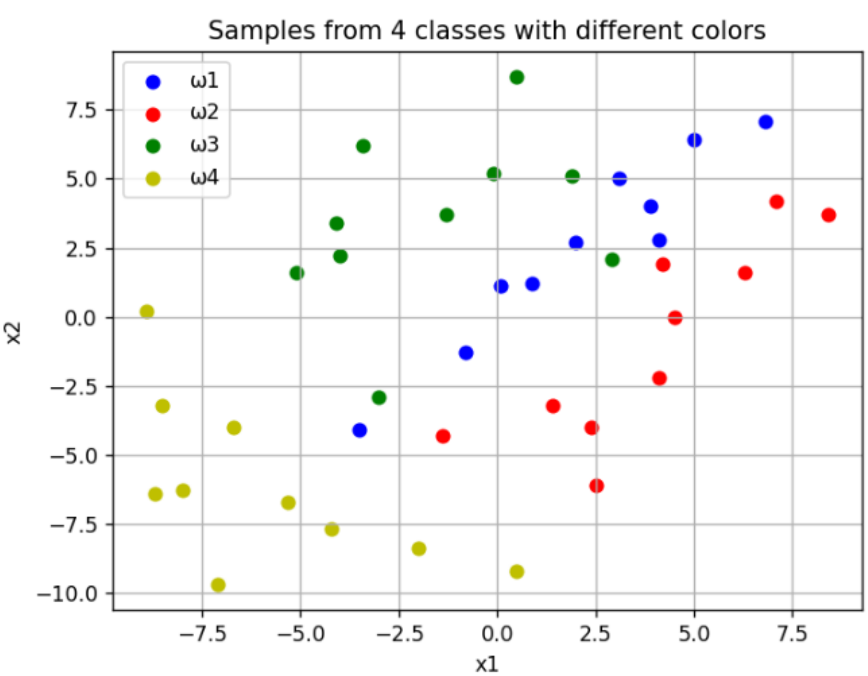
4 Ιουλίου 2024

1 Θέμα 1: Αλγόριθμος *Perceptron*

Για την συγκεκριμένη άσκηση μας δόθηκε ένα *dataset* από 2*D* δείγματα από 4 κλάσεις w_1, w_2, w_3, w_4 και μας ζητήθηκε να εφαρμόσουμε τον αλγόριθμο *batch perceptron* για να γίνει ταξινόμηση ανάμεσα σε δύο κλάσεις κάθε φορά.

1. Αρχικά, το πρόγραμμα μέσω της συνάρτησης "*plot_saples_boundaries*" διαβάζει και σχεδιάζει στο ίδιο διάγραμμα τα δοσμένα δείγματα αλλά με διαφορετικό χρώμα για κάθε κλάση.

```
colors = {'ω1': 'b', 'ω2': 'r', 'ω3': 'g', 'ω4': 'y'}
for label, points in samples.items():
    plt.scatter(points[:, 0], points[:, 1], c=colors[label], label=label)
```



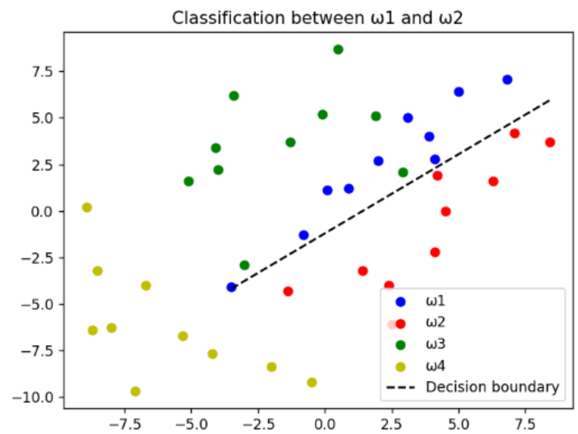
Σχήμα 1: Διάγραμμα δειγμάτων από 4 κλάσεις με διαφορετικά χρώματα η κάθε μία

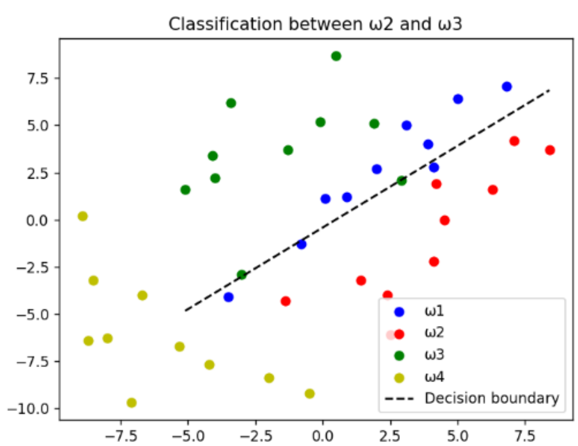
Εύκολα βλέπουμε ότι τα ζευγάρια των κλάσεων $(w_1, w_2), (w_2, w3), (w_3, w_4)$ είναι γραμμικά διαχωρίσιμα καθώς μπορούν να διαχωριστούν με ευθείες γραμμές.

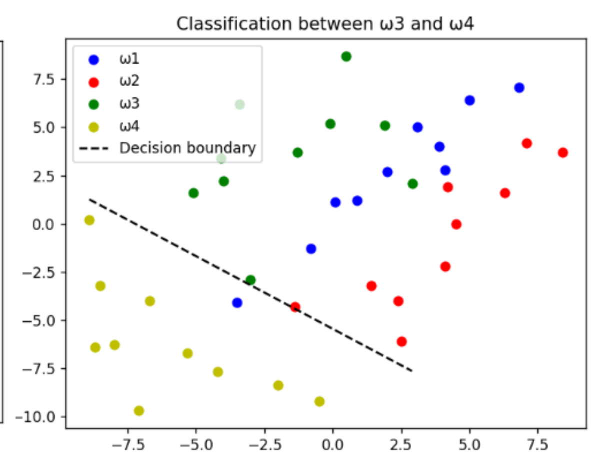
2. Ο αλγόριθμος ξεκινάει με μηδενικές τιμές για το διάνυσμα των βαρών και σημειώνει τον αριθμό των δειγμάτων που είναι ταξινομημένα λάθος καθώς και τον αριθμό των επαναλήψεων που απαιτούνται μέχρι να υπάρχει σύγκλιση, δηλαδή μέχρι η ευθεία που δημιουργείται να ταξινομεί σωστά το κάθε δείγμα στην κλάση που ανήκει. Η διαφορά που παρατηρείται στον αριθμό των επαναλήψεων μέχρι να επιτευχθεί σύγκλιση οφείλεται στην διαφορετική χωρική τοποθέτηση των δειγμάτων της κάθε κλάσης πάνω στο επίπεδο X_1, X_2 καθώς και στην απόσταση των δειγμάτων κοντά στο σημείο διαχωρισμού (σημεία διαφορετικής κλάσης πολύ κοντά μεταξύ τους κοντά στην ευθεία διαχωρισμού) όπως φαίνεται και από τα παρακάτω *plots*.

	$(w_1 - w_2)$	$(w_2 - w_3)$	$(w_3 - w_4)$
$MisclassifiedSamplesOnStart$	7	5	3
$IterationsUntilConvergence$	26	14	49

3. Στην συνέχεια, εκτυπώνονται ξανά τα δείγματα των τεσσάρων κλάσεων αυτή την φορά μαζί με τα $decisionboundaries$.







2 Θέμα 2: Λογιστική Παλινδρόμηση: Αναλυτική εύρεση κλίσης (*Gradient*)

Καταρχάς θα αποδείξουμε πως το j-οστ στοιχείο της κλίσης του σφάλματος, δηλαδή η κλίση ($gradient$) του σφάλματος, είνα της μορφής

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m} = (h_\theta(x^{(i)}) - y^{(i)})x_j^i$$

Θα χρησιμοποιήσουμε την συνάρτηση της λογιστικής παλινδρόμησης, η οποία ορίζεται ως εξής

$$h_\theta(x) = f(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

και τη συνάρτηση κόστους/σφάλματος ($loss\ function$), που ονομάζεται ςροσσ-εντροπψ, και ορίζεται ως:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(-y^{(i)}\ln(\hat{y}^{(i)}) - (1-y^{(i)})\ln(1-\hat{y}^{(i)})\right)$$

Θα υπολογιστεί η μερική παράγωγος:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}(\frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}ln(h_\theta(x^{(i)})) - (1-y^{(i)})ln(1-h_\theta(x^{(i)}))]) \Rightarrow$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial \theta_j}[-y^{(i)}ln(h_\theta(x^{(i)})) - (1-y^{(i)})ln(1-h_\theta(x^{(i)}))] \Rightarrow$$

Για να υπολογιστεί το παραπάνω θα χρησιμοποιηθούν:

- $$\frac{\partial}{\partial \theta_j}[-y^{(i)}ln(h_\theta(x^{(i)})] = -y^{(i)}(1-h_\theta(x^{(i)}))x_j^{(i)}$$
- $$\frac{\partial}{\partial \theta_j}[-(1-y^(i)ln(1-h_\theta(x^{(i)}))] = (1-y^{(i)})h_\theta(x^{(i)})x_j^{(i)}$$

Συνδιάζοντας τα παραπάνω μπορεία να αποδειχθεί πως

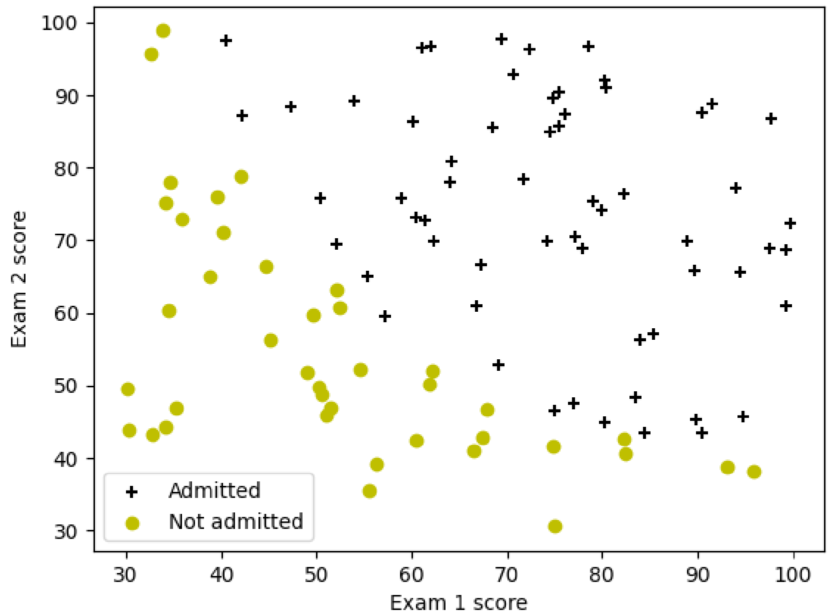
$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}(1-h_\theta(x^{(i)}))x_j^{(i)} + (1-y^{(i)})h_\theta(x^{(i)})x_j^{(i)}] \Rightarrow$$

$$\Rightarrow \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Το οποίο είναι αυτό που έπρεπε να αποδειχθεί.

Στη συνέχεια χρησιμοποιήθηκε λογιστική παλινδρόμιση για να προβλεψουμε αν ένα φοιτητής θα γίνει δεκτός σε ένα πανεπιστήμιο με βάση τους βαθμούς του σε δύο εξετάσεις. Χρησιμοποιήθηκαν τα δεδομένα του αρχείου «*exam_scores_data1.txt*» ως δεδομένα εκμάθησης της λογιστικής παλινδρόμησης, όπου υπάρχουν δεδομένα από παλαιότερες αιτήσεις φοιτητών στη μορφή «$Exam1Score, Exam2Score$, [0: απόρριψη, 1: αποδοχή]». Συμπληρώνοντας το δωσμένο *python script* (*MyLogit.py*), παράχθηκαν τα παρακάτω αποτελέσματα:

Αρχικά διαβάστηκαν και σχεδιάστηκαν τα δείγματα από το αρχείο που δόθηκε:

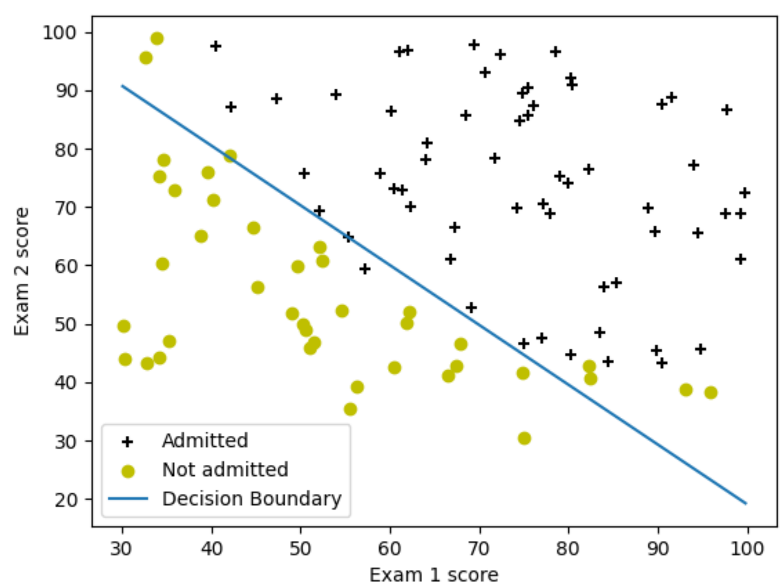


Σχήμα 2: Διάγραμμα φοιτητών σε σχέση με τις βαθμολογίες τους και το αν πέρασαν ή όχι

Υπολογίστηκε πως για το αρχικό $\mathbf{\Theta}$, όπου $\mathbf{\Theta} = [\theta_1, \theta_2, ..., \theta_n]^T$ οι παράμετροι του γραμμικού μοντέλου οι οποίες αρχικά θα είναι μηδέν, η το σφάλμα και η κλίση θα είναι:

```
Cost at initial theta (zeros): 0.8213534163241316
Gradient at initial theta (zeros): [ -0.26666667 -22.94992893 -22.29984189]
```

Στη συνέχεια χρησιμοποιώντας τη συνάρτηση *minimise* και εφαρμόζοντας τη συνάρτηση υπολογισμού του *gradient* που αποδείχθηκε παραπάνω, υπολγίστηκε και βελτισοποιήθηκε το όριο απόφασης:



Τέλος, υπολογίζεται η πιθανότητα ένας φοιτητής θα γίνει δεκτός αν έγραψε στο πρώτο μάθημα 45 και στο δεύτερο 85:

```
For a student with scores 45 and 85, we predict an admission probability of
0.7762936552485087
Train Accuracy: 89.0
```

Υπολογίστηκε επίσης η ακρίβεια του μοντέλου μας, συγκρίνοντας για κάθε φοιτητή την *predicted* επιτυχία ή αποτυχία του να γίνει δεκτός σε σχέση με το αν έγινε όντως δεκτός ή όχι.

3 Θέμα 3: Εκτίμηση Παραμέτρων με Maximum Likelihood

Στην συγκεκριμένη άσκηση μας δόθηκε ένας πίνακας με $3D$ δείγματα από 3 διαφορετικές κλάσεις τα οποία ακολουθούν *Gaussian* κατανομές σε όλες τις διαφορετικές διαστάσεις.

1. Αρχικά, υπολογίστηκαν οι τιμές μεγίστης πιθανοφάνειας $\hat{\mu}$ και σ^2 ξεχωριστά για κάθε ένα από τα 3 χαρακτηριστικά για την κλάση w_1

```python
def MLE_calculation(input_data):
    mean = np.mean(input_data)
    variance = np.var(input_data)
    return mean,variance
```

class w_1	X_1	X_2	X_3
Mean	-0.0709	-0.6047	-0.911
Variance	0.906	4.200	4.541

2. Στην συνέχεια, θεωρώντας ότι ανά δύο τα χαρακτηριστικά της κλάσης w_1 ακολουθούν $2D$ κανονική κατανομή, δημιουργούμε ένα πρόγραμμα που θα υπολογίζει τις παραμέτρους της κανονικής κατανομής (μέση τιμή και διασπορά) με τη μέθοδο της μέγιστης πιθανοφάνειας στα 3 ζευγάρια χαρακτηριστικών.

```python
# calculate the mean and the covariance  of the input data of 2
Dimensions
def MLE_2D_calculation(input_data):
    input_array = np.array(input_data).T    # convert the matrix
into an array for easier covariance calculation

    mean_x1 = np.mean(input_data[0],axis=0) # mean of feature x1
    mean_x2 = np.mean(input_data[1],axis=0) # mean of feature x2
    mean = [mean_x1, mean_x2]
    # we use bias = true because we are calculating the biased
covariance since we
    # are estimating its value with the ML estimation
    # The value of the estimation for the covariance is different
from the real one

    covariance = np.cov(input_array,rowvar=False,bias=True) #
covariance of x1 - x2

    return mean,covariance
```

	$X_1 - X_2$	$X_2 - X_3$	$X_3 - X_4$
Mean	[-0.0708, -0.6047]	[-0.605, -0.911]	[-0.071, -0.911]
Variance	[0.906 0.567 ; 0.567 4.200]	[4.201 0.734 ; 0.734 4.542]	[0.906 0.394 ; 0.394 4.542]

3. Έπειτα, θεωρώντας ότι το συνολικό διάνυσμα χαρακτηριστικών της κλάσης ω1 ακολουθεί $3D$ κανονική κατανομή, υπολογίζουμε τις παραμέτρους της κανονικής κατανομής (*mean* και *covariance*) με τη μέθοδο της μέγιστης πιθανοφάνειας στο σύνολο των χαρακτηριστικών.

```
# calculate the mean and the covariance  of the input data of 3
Dimensions
def MLE_3D_calculation(input_data):
    input_array = np.array(input_data).T    # convert the matrix
into an array for easier covariance calculation

    mean_x1 = np.mean(input_data[0],axis=0) # mean of feature x1
    mean_x2 = np.mean(input_data[1],axis=0) # mean of feature x2
    mean_x3 = np.mean(input_data[2],axis=0) # mean of feature x3
    mean = [mean_x1, mean_x2, mean_x3]
    covariance = np.cov(input_array,rowvar=False,bias=True) #
covariance of x1 - x2 - x3

    return mean,covariance
```

class w_1	$X_1 - X_2 - X_3$
mean	[-0.071, -0.6047, -0.911]
variance	[[0.906 0.567 0.394] [0.567 4.201 0.733] [0.394 0.734 4.541]]

4. Υποθέτουμε ότι το τρισδιάστατο μοντέλο μας είναι διαχωρίσιμο, έτσι ώστε $\Sigma = diag(\sigma_1^2, \sigma_2^2, \sigma_3^2)$ Εκτιμούμε με τη μέθοδο μεγίστης πιθανοφάνειας το μέσο $\hat{\mu}$ και τα διαγώνια στοιχεία του Σ για τα δεδομένα της κλάσης ω2. Αρχικά, υπολογίστηκαν οι μέσες τιμές και οι διασπορές των 3 χαρακτηριστικών και στην συνέχεια ο πίνακας συνδιασποράς.

class w_2	X_1	X_2	X_3
mean	-0.113	0.43	0.004
variance	0.054	0.046	0.007

Τέλος, υπολογίστηκε το διάνυσμα μέσου και η συνδιασπορά του τρισδιάστατου μοντέλου.

class w_2	$X_1 - X_2 - X_3$
mean	[-0.113, 0.43, 0.004]
variance	[[0.054 0 0] [0 0.046 0] [0 0 0.004]]

5. Παρατηρούμε ότι, η διασπορά της κλάσης ω2 και για τα 3 χαρακτηριστικά είναι μικρότερη από τη διασπορά της κλάσης ω1. Αυτό σημαίνει ότι οι τιμές των χαρακτηριστικών της κλάσης ω2 είναι πιο συγκεντρωμένες γύρω από την μέση τιμή τους. Έτσι, η κλάση ω2 είναι πιό εύκολα διακριτή και κατά συνέπεια πιο εύκολα διαχωρίσιμη με πιο έμπιστες στατιστικές εκτιμήσεις.

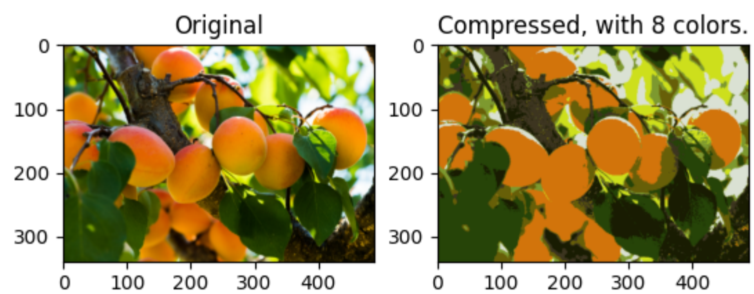
4 Θέμα 4: Ομαδοποίηση ($Clustering$) με $K-means$ και GMM

Για την υλοποίηση αυτής της άσκησης συμπληρώθηκε ο κώδικας που λείπει στο αρχείο $myImageCompression.py$. Πιο συγκεκριμένα:

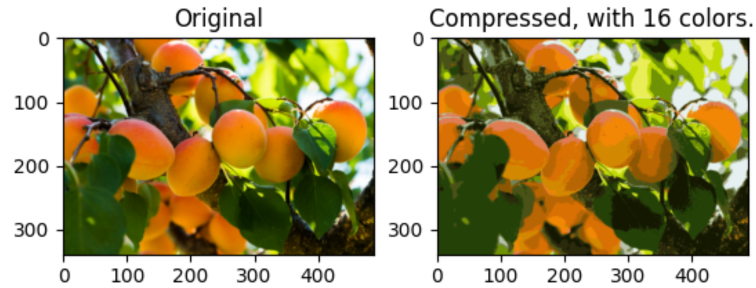
1. Υλοποιήθηκε η συνάρτηση $find\_closest\_centroids$ για την εύρεση των κοντινότερων κέντρων c_i για κάθε σημείο x_i από τα δεδομένα, χρησιμοποιώντας την μικρότερη ευκλίδεια απόσταση ανάμεσα στο δειγμα x_i και τα $centroids$

2. Υλοποιήθηκε η συνάρτηση $compute\_centroids$ για τον υπολογισμό των κεντροειδών των κλάσεων χρησιμοποιώντας τον τύπο:
$$\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

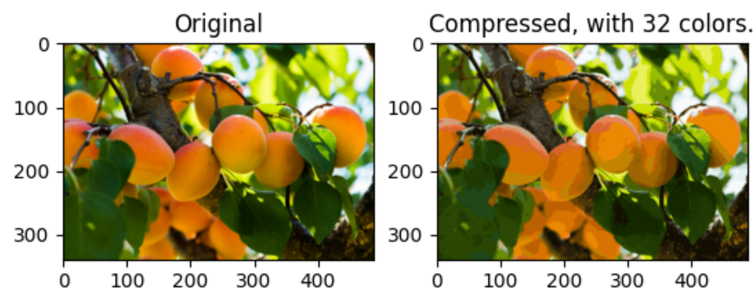
Παρακάτω φαίνονται τα αποτελέσματα του προγράμματος για διαφορετικές συμπιέσεις:



Σχήμα 3: Συμπίεση για 8 κλάσεις (10 επαναλήψεις)



Σχήμα 4: Συμπίεση για 16 κλάσεις (10 επαναλήψεις)



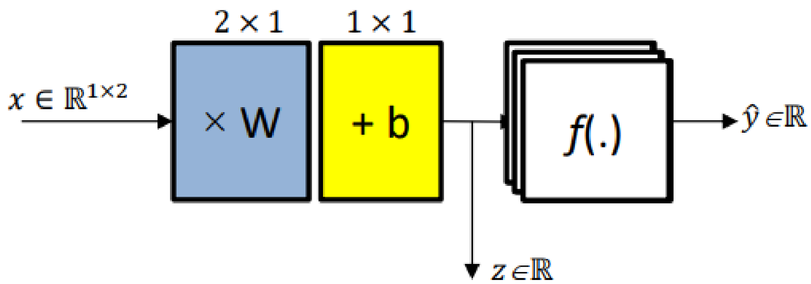
Σχήμα 5: Συμπίεση για 32 κλάσεις (10 επαναλήψεις)

Παρατηρείται πως με την αύξηση του αριθμού κλάσεων των χρωμάτων μπορεί να συγκρατηθεί παραπάνω λεπτομέρεια στη συμπιεσμένη εικόνα. Αυτο φαίνεται καθώς υπάρχουν διαβαθμίσεις στη σκίαση και την ευκρίνεια των φρούτων. Ωστόσο με την αύξηση των κλάσεων το μέγεθος της εικόνας επίσης αυξάνεται. Εινaι δεδομένο πως με την αύξηση του αριθμού των επαναλήψεων το αποτέλεσμα θα είναι ακόμα καλύτερο.

5 Θέμα 5α: Μέρος Α: Υλοποίηση ενός απλού νευρωνικού δικτύου.

Σ' αυτή την άσκηση ο στόχος είναι να υλοποιηθεί και εκπαιδευτεί ένα νευρωνικό δίκτυο χωρίς την χρήση έτοιμων προγραμμάτων/βιβλιοθηκών. Η υλοποίηση θα γίνει σε *python* χρησιμοποιώντας *numpy*.

Μέρος Α: Έστω ότι σας δίνονται τα δεδομένα εκπαίδευσης $D = \{(x^1, y^1), ..., (x^N, y^N)\}$ με $x^i \in R^{1\times 2}$



Σχήμα 6: Αναπαράσταση Νεωρωνικού Δικτύου Με Την Μορφή Πινάκων

1. Θέτουμε $z^i = x^i W + b$, οπότε $\hat{y}^i = f(z^i)$ οπότε έχουμε:

$$J(Y, \hat{Y}; W, b) = \frac{1}{B}\sum_i(-y^i ln(\hat{y}^i) - (1-y^i)ln(1-\hat{y}^i)) = \frac{1}{B}\sum_i(A - C * D)$$

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

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

$$C * D = -(1-y^i)D = -(1-y^i)(-z - ln(1+e^{-z^i})) =$$
$$= -z^i - ln(1+e^{-z^i}) + y^i z^i + y^i ln(1+e^{-z^i})$$

Άρα:

$$A - C * D = y^i ln(1+e^{-z^i}) - (-z - ln(1+e^{-z^i}) + y^i z^i + y^i ln(1+e^{-z^i})) =$$
$$-z^i - y^i z^i + ln(1+e^{-z^i})$$

Επομένως:

$$J(Y, \hat{Y}; W, b) = \frac{1}{B}\sum_i(-z^i - y^i z^i + ln(1+e^{-z^i}))$$

2. Θα αποδειχθεί πως

$$\frac{\partial J}{\partial z^i} = -y^i + \hat{y}^i$$

Καταρχάς έχουμε πως

$$J(Y, \hat{Y}; W, b) = \frac{1}{B}\sum_i(z^i - z^i y^i + ln(1+e^{-z^i}))$$

Επίσης, ισχύουν και τα παρακάτω:

(α')

$$\frac{\partial z^{(i)}}{\partial z^{(i)}} = 1$$

(β′)

$$\frac{\partial(z^{(i)}y^{(i)})}{\partial z^(i)} = y^{(i)}$$

(γ′)

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

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

Από τα παραπάνω μπορούμε να συμπεράνουμε πως:

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

Που είναι αυτό που έπρεπε να αποδειχθεί.

3. Χρησιμοποιώντας τον κανόνα της αλυσίδας θα υπολογισθούν οι μερικές παράγωγοι:

$$\frac{\partial J}{\partial W}, \frac{\partial J}{\partial b}$$

(α′)

$$\frac{\partial J}{\partial W} = \sum_i \frac{\partial J}{\partial f(z^i)} \frac{\partial f(z^i)}{\partial W} = \sum_i \frac{\partial J}{\partial f(z^i)} \frac{\partial f(z^i)}{\partial z^i} \frac{\partial z^i}{\partial w} =$$

$$\frac{1}{b}\sum_i (-z^i)(f(z^i) * (1 - f(z^i))) * (x^i) = \frac{1}{b}\sum_i -z^i x^i \hat{y}^i (1-\hat{y}^i)$$

(β′)

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

Όμως γνωρίζουμε ότι:

$$\frac{\partial z^i}{\partial b} = 1$$

Εφαρμόζουμε τον κανόνα της αλυσίδας:

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

4. Ο αλγοριθμος Backpropagation έχει 3 στάδια:

- Αρχικά γίνεται το Forward Pass, όπου υπολογίζονται οι έξοδοι του δικτύου για τυχαία είσοδο. Για κάθε $layer\ l$, εφόσον $z^i = x^i W + b$ όπου το διάνυσμα W αντιπροσοπεύει τα βάρη, οι έξοδοι των hidden layers περιγράφεται από τη συνάρτηση:

  $$h^i = f(z^i) = \frac{1}{1+e^{-z^{(i)}}}$$

  και η έξοδος του κάθε νευρώνα στο output layer περιγράφεται από από τη συνάρτηση:

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

- Στη συνέχεια γίνεται υπολογισμός σφάλματος, κάνοντας χρήση της $cross-entropy$ συνάρτησης. Η συνάρτηση φαίνεται που χρησιμοποιούμε είναι:

  $$J(Y,\hat{Y};W,b) = \frac{1}{B}\sum_i (-z^i - y^i z^i + ln(1+e^{-z^i}))$$

- Ύστερα, γίνεται η διαδικασία της οπισθοδιάδοσης (*Backward Pass*), όπου αρχικά υπολογίζονται τα σφάλματα για κάθε layer ξεκινώντας από το output layer

$$\frac{\partial J}{\partial \hat{y}}$$

  και συνεχίζοντας στα hidden layers

$$\frac{\partial \hat{y}}{\partial h^i}$$

  Επομένως, στην έξοδο του layer 2 έχουμε για παράδειγμα:

$$\frac{\partial J}{\partial h^2} = \frac{\partial \hat{y}}{\partial h^2} \frac{\partial \hat{J}}{\partial \hat{y}}$$

- Τέλος, γίνεται ενημέρωση των βαρών χρησιμοποιώντας την μέθοδο Gradient Descent

6 Θέμα 5α: Μέρος Β.

Για το δεύτερο μέρος συμπληρώθηκε ο κώδικας της άσκησης σε *python* και εκπαιδεύτηκε το νευρωνικό δίκτυο χρησιμοποιώντας δείγματα από την βάση δειγμάτων *MNIST*. Τα δείγματα είναι εικόνες μεγέθους $28x28$ που απεικονίζουν χειρόγραφα ψηφία αριθμών από το 0 έως το 9. Ο στόχος είναι το δίκτυο να μπορεί να αναγνωρίζει την κλάση στην οποία ανήκει κάθε δείγμα ξεχωριστά. Αρχικά, το δίκτυο είναι *fully connected* (*dense*) αποτελούμενο από νευρώνες με συνάρτηση ενεργοποίησης την λογιστική συνάρτηση (*Sigmoid*). Στο επόμενο επίπεδο χρησιμοποιείται η συνάρτηση ενεργοποίησης *SoftMax*.

Αρχικά, υλοποιήθηκε η συνάρτηση η οποία αναπαριστά ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο χωρίς συνάρτηση ενεργοποίησης.
Στην αρχή πραγματοποιείται *forward Pass* μέσω της συνάρτησης *forward* και στη συνέχεια υλοποιείται η συνάρτηση *backward* η οποία πραγματοποιεί *back propagation* για να εκπεδεύσει το δίκτυο και να προσαρμόσει τις παραμέτρους του με βάση το *error* που λαμβάνει στην έξοδο του *output layer*.

```python
# Neural Network Dense (Fully Connected) Layer without activation
class Dense():
    def __init__(self, input_size, output_size):
        self.weights = np.random.randn(output_size, input_size)
        self.bias = np.random.randn(output_size, 1)

    #Forward Propagation on a Dense Layer
    def forward(self, input):
        self.input = input
        # Add Code Here
        #fwd =  np.dot(self.input , self.weights) + self.bias
        fwd =  np.dot(self.weights, self.input) + self.bias
        return fwd

    #Backward Propagation on a Dense Layer
    # dE_dY is dE/dY Gradient
    # dE_dW is dE/dW Gradient
    # dE_dB is dE/dB Gradient
    # dE_dX is dE/dX Gradient
    def backward(self, dE_dY, learning_rate):
        dZ_dW = self.input.T                        # dZ_dW = Xi
        dE_dW = np.dot(dE_dY,dZ_dW)/ self.input.shape[1]# dE_dW =  dE_dY / dZ_dW /// Normalizing by B
                                                                        ##// Ensures that the
        dY_dX = self.weights                        # dY_dX = W since we have no activation function
        dE_dX = np.dot(dY_dX.T,dE_dY)               # dE_dX = dE_dY * dY_dX = dE_dY * W

        # Since b is added to each neuron's output we need sum for each neuron across the batch
        # dE_dB = sum(dE_dZ * dZ_db) = sum(dE_dZ * 1) = sum(dE_dZ) = sum(dE_dY) => dE_dB = sum(dE_dY)
        # keepdims = True ensures that the resulting gradient has the same shape as B
        dE_dB = np.sum(dE_dY, axis=1, keepdims=True)/ self.input.shape[1]

        self.update_weights(dE_dW, dE_dB, learning_rate)
        return dE_dX
```

Η προσαρμογή των βαρών του νευρωνικού δικτύου πραγματοποιείται απο την *update_weights* η οποία προσαρμόζει τα βάρη με βάση το συνολικό *error* και το *learning_rate*.

```python
    # Update Layer Weights and bias
    def update_weights(self, dE_dW, dE_dB, learning_rate):
        # Add Code Here
        self.weights -= learning_rate * dE_dW
        self.bias -= learning_rate * dE_dB
```

Στην συνέχεια, η μοντελοποίηση της εκάστοτε συνάρτησης ενεργοποίησης έγινε θεωρώντας την ως ένα έξτρα *layer* του νευρωνικού δικτύου το οποίο έχει και πάλι συνάρτηση για *forward pass* και *backward pass*.
Η Κλάση αυτή, κληρονομείται από την εκάστοτε συνάρτηση ενεργοποίησης (*Softmax*, *Tanh*, *Sigmoid*) του κάθε *layer* και σε κάθε περίπτωση προσαρμόζει την έξοδο που θα είχε στο *forward pass* και στο *backward pass*.

Χρησιμοποιούνται δύο διαφορετικές συναρτήσεις για τον υπολογισμό του κόστους της κάθε απόφασης του νευρωνικού δικτύου, η *MSE* και η *Loss_Cross_Entropy* έτσι ώστε να γίνει σύγκριση μεταξύ των δύο.

```python
# Return the the cross entropy loss
def loss_cross_entropy(y_true, y_pred):
    B = len(y_true)
    # Using the cross entropy formula to calculate the loss
    # We have added a small value in every ln operation so that we avoid ln(0)
    loss = (1/B)*np.sum(-y_true*np.log(y_pred+ 1e-15)-(1-y_true)*np.log(1-y_pred+1e-15))
    return loss

# Return the derivative of the cross entropy loss
def loss_cross_entropy_grad(y_true, y_pred):
    lossGrad =  -(y_true / y_pred - (1 - y_true) / (1 - y_pred))
    return lossGrad

def mse(y_true, y_pred):
    # Add Code Here
    loss = np.mean((y_true - y_pred) ** 2)
    return loss

def mse_grad(y_true, y_pred):
    # Add Code Here
    n = y_true.shape[0]
    lossGrad = (2 / n) * (y_pred - y_true)
    return lossGrad
```

Τέλος, γίνεται η εκπαίδευση του δικτύου με την χρήση της συνάρτησης *train* και και έπειτα γίνεται η αξιολόγηση του νευρωνικού χρησιμοποιώντας *test data* από την ίδια βάση δεδομένων.

Στην συνέχεια, χρησιμοποιώντας την συνάρτηση κόστους *Loss_Cross_Entropy*, $learning\_Rate = 0.1$, $batch\_size = 128$ και αριθμό $epochs = 100$, πραγματοποιούμε εκπαίδευση του νεωρωνικού και στην συνέχεια αξιολογούμε την επίδοσή του.
Κατά την διάρκεια της εκπαίδευσης, παρατηρούμε ότι το Νευρωνικό για τα πρώτα *epochs* έχει μεγάλο *error* στην έξοδό του, αλλά όσο συνεχίζει να εκπαιδεύεται με περισσότερα, το *error* μειώεται σημαντικά. Παρακάτω φαίνεται η δραματική μείωση του *error* μεταξύ των πρώτων και των τελευταίων 5 *epochs*.

Πραγματοποιείται η αξιολόγηση της επίδοσης του νευρωνικού δικτύου χρησιμοποιώντας δεδομένα απο το ίδιο *dataset*. Το νευρωνικό πετυχαίνει ποσοστό σωστής πρόβλεψης ψηφίων ίσο με $Ratio = 90\%$ ενώ το Mean Square Error (MSE) είναι ίσο με 0.0169.

- Στην συνέχεια, πραγματοποιούνται αλλαγές στις παραμέτρους *learning_Rate*, *batch_size* και αριθμό *epochs* για να μελετηθεί η συμπεριφορά του νεωρωνικού δικτύου. Τα αποτελέσματα συγκεντρώνονται στους παρακάτω πίνακες.
  Για $learning\_Rate = 0.1$, $batch\_size = 128$

```
1/100, error=4.8379295889558245
2/100, error=1.1672506956781168
3/100, error=0.6044525005882451
4/100, error=0.46234524839187036
5/100, error=0.2488500112953672
```

Σχήμα 7: 5 πρώτα *epochs*

```
95/100, error=2.20477215402219e-05
96/100, error=3.4207240373314365e-05
97/100, error=1.9981523119017902e-05
98/100, error=3.0209992517340018e-05
99/100, error=2.7470429448847502e-05
100/100, error=2.3656063534843082e-05
```

Σχήμα 8: 5 τελευταία *epochs*

Number Of epochs	1	10	100	1000
Ratio	60%	80%	90%	90%
MSE	0.0674	0.0294	0.0169	0.0169

Για $learning\_Rate = 0.1$, $Num\_Of\_Epochs = 100$

Batch Size	64	128	256
Ratio	85%	90%	90%
MSE	0.0308	0.0169	0.0199

Για $Num\_Of\_Epochs = 100$, $batch\_size = 128$

Learning Rate	0.1	0.01	0.001
Ratio	90%	90%	75%
MSE	0.0169	0.0248	0.0382

Παρατηρούμε ότι, με την αύξηση του αριθμού από *epochs* αρχικά πετυχαίνουμε βελτίωση του μοντέλου, με μείωση του MSE και αύξηση του *Ratio*. Ωστόσο, το μοντέλο φαίνεται να κάνει *converge* για αριθμό $epochs = 100$ καθώς η περαιτέρω αύξηση τους δεν οδηγεί σε νέα βελτίωση. Το ίδιο συμβαίνει και για την μεταβλητή $batch\_size$ όπου αρχικά έχουμε βελτίωση της απόδοσης μέχρις ότου το μοντέλο κάνει *converge* Συμπεριλαμβάνοντας στην εξίσωση και το υπολογιστικό κόστος, ο ιδανικός αριθμός απο *epochs* είναι 100 ενώ το ιδανικό μέγεθος κάθε *epochs* είναι 128.
Στο πεδίο του $Learning\_Rate$ βλέπουμε ότι, η σημαντική μείωσή του έχει ως αποτέλεσμα το μοντέλο να μην προλαβαίνει να προσαρμόσει τις παραμέτρους του για τα διαθέσιμα δεδοδένα εκπαίδευσης και έτσι δεν πετυχαίνει καλή απόδοση.

- Έπειτα, χρησιμοποιήθηκε η συνάρτηση ενοργοποίησης *tanh* και η συνάρτηση σφάλματος MSE. Παρακάτω γίνεται σύγκριση με την προηγούμενη περίπτωση που χρησιμοποιήθηκε η συνάρτηση ενεργοποίησης *Sigmoid* και η $Loss\_Cross\_Entropy$ για τον υπολογισμό του *error* για $learning\_Rate = 0.1$, $batch\_size = 128$ και αριθμό από $epochs = 100$.

	$Sigmoid + Loss\_Cross\_Entropy$	$Tanh + MSE$
Ratio	90%	60%
MSE	0.0169	0.0679

Παρατηρούμε ότι η χρήση της συνάρτηση ενοργοποίησης *tanh* σε συνδιασμό με τη συνάρτηση σφάλματος MSE μειώνουν σημαντικά την απόδοση του νεωρωνικού δικτύου. Αυτό συμβαίνει γιατί η συνάρτηση κόστους MSE χρησιμοποιείται για προβλήματα όπου η έξοδος του συστήματος είναι συνεχής, δηλαδή για προβλήματα *regression*. Επίσης, η συγκεκριμένη συνάρτηση τείνει να 'τιμωρεί' σημαντικά τα μεγάλα *errors* λόγω και του τετραγώνου που υπάρχει στον τύπο της, γεγονός που μπορεί να οδηγήσει σε πιο αργή σύγκλιση ειδικά αν οι αρχικές εκτιμήσεις του μοντέλου απέχουν πολύ απο τις πραγματικές τιμές. Άρα, σε ένα πρόβλημα *classification* όπως αυτό που έχουμε στην συγκεκριμένη περίπτωση, η MSE παρουσιάζει κακή απόδοση. Τέλος, σε προβλήματα *classification*, ειδικότερα όταν δεν έχουμε συνάρτηση *Softmax* στο *output layer* προτιμάτε η χρήση της *Sigmoid* σε σχέση με την *tanh* καθώς η έξοδος της είναι μεταξύ [0,1] το οποίο μπορεί να χρησιμοποιηθεί ως κατανομή πιθανοτήτων.

- Επαναφέρουμε το νευρικό στην αρχική του κατάσταση και αυτή την φορά δοκιμάζονται διαφορετικές αρχιτεκτονικές δικτύου για να μελετηθεί εκ νέου η συμπεριφορά του μοντέλου.
  - Αρχικά, προστέθηκε ένα επιπλέον *hidden layer* με συνάρτηση ενεργοποίησης ξανά την *Sigmoid*. Ωστόσο το μοντέλο δεν παρουσίασε κάποια βελτίωση.

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

Σχήμα 9: A NN with 3 Layers

$$\begin{bmatrix} Ratio & 90\% \\ MSE & 0.0169 \end{bmatrix}$$

Σχήμα 10: Performance of NN

  - Προστέθηκε ένα ακόμη *Layer* αυξάνοντας τον συνολικό αριθμό τους σε 4. Με αυτήν την προσθήκη, το NN έγινε πολύ πιο περίπλοκο και σύνθετο, πράγμα που κατέστησε την διαδικασία της εκμάθησης αρκετά χρονοβόρα. Επιπλέον, παρατηρήθηκε ότι η απόδοση του μοντέλου δεν αυξήθηκε, αντιθέτως υπήρχαν περιπτώσεις στις οποίες η απόδοση του έπεσε. Αυτό μπορεί να οφείλεται στους εξής λόγους:
    1. **Overfitting:** Καθώς το μοντέλο γίνεται πιο περίπλοκο, είναι σε θέση να μάθει πιο περίπλοκες συσχετίσεις μεταξύ των δεδομένων εκπαίδευσης ακόμη και τον θόρυβο που αυτά περιέχουν. Αυτό, παρόλο που μειώνει το *loss* στα δεδομένα εκπαίδευσης, μειώνει και την απόδοση του μοντέλου σε νέα άγνωστα δεδομένα.
    2. Το παραπάνω εντείνεται από το γεγονός ότι χρησιμοποιούνται σχετικά **λίγα δεδομένα εκπαίδευσης** έτσι ώστε να είναι σε θέση να γίνει η εκπαίδευση σε έναν συμβατικό υπολογιστή χωρίς την χρήση κάρτας γραφικών. Τα πιο σύνθετα μοντέλα απαιτούν πολύ περισσότερα δεδομένα για να εκπαιδευτούν.

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

Σχήμα 11: A NN with 4 Layers

$$\begin{bmatrix} Ratio & 85\% \\ MSE & 0.0212 \end{bmatrix}$$

Σχήμα 12: Performance of NN