1^{η} Logistic Discrimination; Finding Gradient

Θεωρητική Βελτιστοποίηση Σφάλματος

Κάνοντας χρήση της συνάρτησης λογιστικής παλινδρόμησης, που ορίζεται ώς:

$$h_\theta(x) = f(\theta^T x),\ \theta = [\theta_1, \theta_2 ... \theta_n]^T \ \&\ f(z) = \frac{1}{1+e^{-z}}$$

θέλουμε να προβλέψουμε την τιμή $y^{(i)}$, που ορίζουν τα δείγματα $x^{(i)}$ με $i\ \in \{1, 2, 3, ...n\}$.

Σαν δεδομένα έχουμε ένα σύνολο m με $m = \left\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\right\}$ όπου $x^{(i)} \in \mathbb{R}^{n\times 1}$ (διανύσματα χαρακτηριστικών) και $y^{(i)} \in \{0, 1\}$ (αντίστοιχη κλάση).

Δεδομένου ότι η εκτίμηση της λογιστικής συνάρτησης, για το $y^{(i)}$ ορίζεται ως $\widehat{y}^{(i)} = h_\theta\left(x^{(i)}\right)$, το σφάλμα cross-entropy υπολογίζεται ώς:

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

Για την βελτιστοποίηση του παραπάνω, θα χρειαστεί να υπολογιστεί η κλίση (gradient) η οποία θα είναι ένα διάνυσμα ίσης διάστασης με το $\overrightarrow{\theta}$.

'Εστω για το δείγμα $x_j^{(i)}$ με θ_j, θα υπολογιστεί το j-στοιχείο του gradient του σφάλματος.

Αρχικά θα αναλύσουμε τι γνωρίζουμε μέχρι στιγμής:

- Αν $f(x) = \ln(x)$ τότε:
$$\frac{df(x)}{dx} = \frac{1}{x} \quad A$$
- Για την λογιστική συνάρτηση $f(z) = \sigma(z) = \frac{1}{1+e^{-z}}$:
$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))^1 \quad B$$
- Κανόνας αλυσίδας για την συνάρτηση $f(x) = h(g(x))$:
$$\frac{df}{dx} = \frac{dh}{dg}\cdot\frac{dg}{dx} \quad C$$

Γνωρίζοντας τα παραπάνω:

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

$$-\frac{1}{m}\sum_{i=1}^{m}\frac{\partial\Big[y^{(i)}\ln\left(h_\theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\ln\left(1-h_\theta\left(x^{(i)}\right)\right)\Big]}{\partial\theta_j} =$$

$$-\frac{1}{m}\sum_{i=1}^{m}\left[\frac{\partial}{\partial\theta_j}y^{(i)}\ln\left(h_\theta\left(x^{(i)}\right)\right) + \frac{\partial}{\partial\theta_j}\left(1-y^{(i)}\right)\ln\left(1-h_\theta\left(x^{(i)}\right)\right)\right] \overset{A,C}{=}$$

[1] Απόδειξη: (link)

$$-\frac{1}{m}\sum_{i=1}^{m}\left[\frac{y^{(i)}}{h_\theta\left(x^{(i)}\right)}\cdot\frac{\partial}{\partial\theta_j}h_\theta\left(x^{(i)}\right)+\frac{1-y^{(i)}}{1-h_\theta\left(x^{(i)}\right)}\cdot\frac{\partial}{\partial\theta_j}\left(1-h_\theta\left(x^{(i)}\right)\right)\right]\Longrightarrow$$

$$\frac{\partial J(\theta)}{\partial\theta_j}=-\frac{1}{m}\sum_{i=1}^{m}\left[\left(\frac{y^{(i)}}{h_\theta\left(x^{(i)}\right)}-\frac{1-y^{(i)}}{h_\theta\left(1-x^{(i)}\right)}\right)\cdot\frac{\partial}{\partial\theta_j}h_\theta\left(x^{(i)}\right)\right]\overset{B,C}{=}$$

$$-\frac{1}{m}\sum_{i=1}^{m}\left[\left(\frac{y^{(i)}}{h_\theta\left(x^{(i)}\right)}-\frac{1-y^{(i)}}{h_\theta\left(1-x^{(i)}\right)}\right)\cdot\underbrace{h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right)\frac{\partial\left(\theta^T\cdot x\right)}{\partial\theta_j}}_{h_\theta\left(x^{(i)}\right)=\sigma(\theta^T x^{(i)})}\right]=$$

$$-\frac{1}{m}\sum_{i=1}^{m}\left[\left(\frac{y^{(i)}}{h_\theta\left(x^{(i)}\right)}-\frac{1-y^{(i)}}{h_\theta\left(1-x^{(i)}\right)}\right)\cdot h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}\right]=$$

$$-\frac{1}{m}\sum_{i=1}^{m}\left[\left(\frac{y^{(i)}-h_\theta\left(x^{(i)}\right)}{\cancel{h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right)}}\right)\cdot\cancel{h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right)}x_j^{(i)}\right]\Longrightarrow$$

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

Πρόβλημα πρόβλεψης

Αρχικά ολοκληρώθηκε η σιγμοειδής συνάρτηση *sigmoid()*, η οποία δέχεται σαν όρισμα έναν πίνακα (διαφόρων διαστάσεων), έστω z και για κάθε στοιχείο αυτού υπολογίζεται ή νέα τιμή εξόδου της συνάρτησης σύμφωνα με τον τύπο:

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

Στην συνέχεια δημιουργήθηκε το αρχείο το οποίο επιστρέφει το κόστος $J(\theta)$ και η κλίση $\nabla J(\theta)$. Για τον υπολογισμό και των δύο παραπάνω, χρησιμοποιούνται οι συναρτήσεις που υπολογίστηκαν παραπάνω. Ουσιαστικά, για κάθε δείγμα αναφοράς, υπολογίζονται και οι δύο ποσότητες και στο τέλος αθροίζονται (mean) για να βγουν τα επιμέρους αποτελέσματα.

Για αρχικές τιμές $\theta=[0\ 0\ 0]^T$ έχουμε τα αναμενόμενα αποτελέσματα αποτελέσματα:

- $J(\theta)\simeq 0,9631$
- $\nabla J(\theta)\simeq[-0.1\ \ -12.01\ \ -11.26]^T$

Για τον υπολογισμό του συνόρου απόφασης, η συνάρτηση *fminunc()* επιστρέφει:

- Κόστος: $J(\theta)\simeq 0.2035$.
- Gradient $\nabla J(\theta)=\simeq[-25.161\ 0.206\ 0.201]^T$.

Γραφικά το όριο απόφασης:

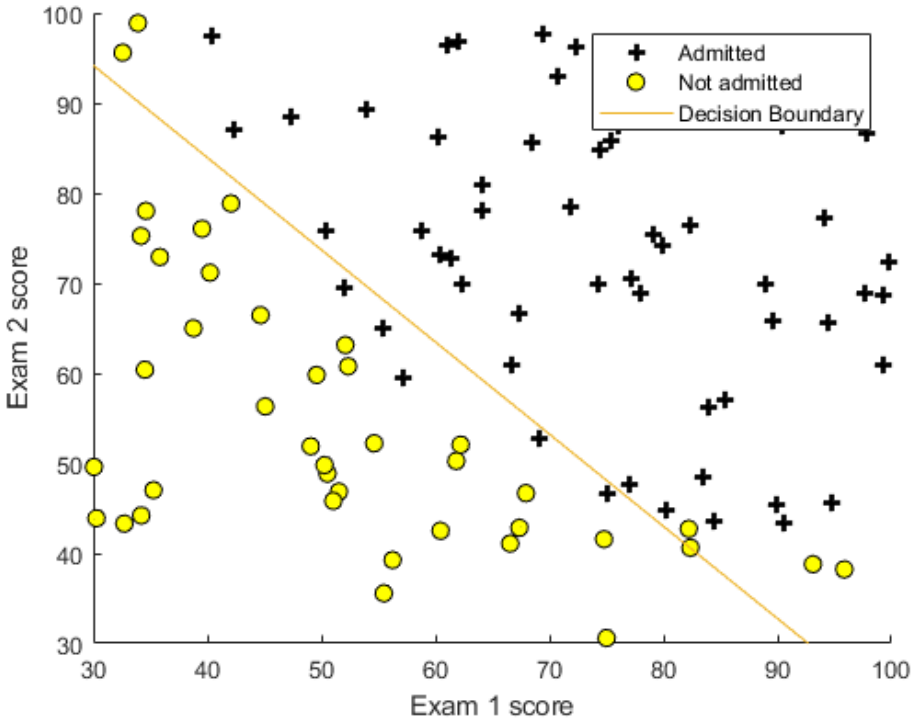


Figure 1: Decision Boundary

Τέλος, έχοντας μάθει τις παραμέτρους που ορίζουν το όριο απόφασης, μπορούμε να κάνουμε προβλέψεις πάνω σε άγνωστα δείγματα. Για παράδειγμα:

exam1	exam2	prop
45	85	77.6%
50	50	0.8%
100	10	7%
10	100	5%
61.8	61.8	50%

Αντίστοιχα αν γίνουν προβλέψεις πάνω στο στο set με το οποίο έγινε η εκπαίδευση, το accuracy φτάνει το **89%**. Αυτό σημαίνει ότι ο υπολογισμός των παραμέτρων έγινε σύμφωνα με τοπικό ελάχιστο το οποίο εντόπισε η συνάρτηση *fminunc()*. Στην περίπτωση ολικού ελαχίστου θα είχαμε, θεωρητικά, καλύτερα αποτελέσματα.

2^{η} Logistic Regression with Regularization

Έχοντας στα χέρια μας τα δεδομένα, θα δομηθεί η συνάρτηση *mapFeature()* η οποία θα αντιστοιχίζει τα χαρακτηριστικά εισόδου (έστω X_1, X_2) σε όλους τους πολυωνυμικούς όρους αυτών. Ποιο συγκεκριμένα, η ανάθεση θα πραγματοποιηθεί έως και βαθμού 6 σύμφωνα με τον τύπο:

$$P(X_1, X_2) = \sum_{i=0}^{6} \sum_{j=0}^{i} X_1^{i-j} X_2^j$$

Σαν αποτέλεσμα θα έχουμε των πίνακα:

$$mapFeature(x) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, ..., x_1x_2^5x_2^6]^T$$

Στην συνέχεια ολοκληρώθηκε η αντίστοιχη συνάρτηση για την επιστροφή κόστους $J(\theta)$ και gradient $\nabla J(\theta)$. Για τον υπολογισμό του κόστους, χρησιμοποιήθηκε η ομαλοποιημένη συνάρτηση κόστους:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \ln \left(h_\theta \left(x^{(i)} \right) \right) - \left(1 - y^{(i)} \right) \ln \left(1 - h_\theta \left(x^{(i)} \right) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Για να υπολογίσουμε την κλίση του κόστους, θα υπολογίσουμε το gradient με τον ίδιο τρόπο με αυτό του πρώτου ερωτήματος με την διαφορά ότι έχουμε να παραγωγίσουμε ένα επιπλέων όρο, το άθροισμα. Έτσι έχουμε:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)} \right] + \frac{\partial}{\partial \theta_j} \left(\frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2 \right)$$

Για τον νέο όρο (ή regularization term) έχουμε:

$$\frac{\partial}{\partial \theta_j} \left(\frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2 \right) = \frac{\lambda}{2m} \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{j=1}^{n} \theta_j^2 \right) = \frac{\lambda}{2m} \cdot \frac{\partial}{\partial \theta_j} \left(\theta_j^2 \right) = \frac{\lambda}{\cancel{2}m} \cdot \cancel{2}\theta_j = \frac{\lambda}{m} \theta_j$$

Συνοψίζουμε, με το τελικό τύπου του gradient που ορίζεται ώς:

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

Κάνοντας χρήση των παραπάνω, η συνάρτηση *costFunctionReg()* επιστρέφει τις αντίστοιχες τιμές.
Ποιο συγκεκριμένα για αρχικούς παραμέτρους:

- Γραμμικό μοντέλο, $\theta_j^T = 0 \quad \forall j \in [1, n]$.
- Regularization factor, $\lambda = 1$

επιστρέφεται κόστος που ισούται με $\mathbf{J}(\theta) \simeq \mathbf{0.693}$ (οι παράμετροι θ μπορούν να τυπωθούν στην άσκηση).

Όμοια με την προηγούμενη άσκηση και για $\lambda = 1$ η συνάρτηση *fminunc()* επιστρέφει τις παραπάνω τιμές και άρα το όριο απόφασης, βρίσκοντας τοπικό ελάχιστο με κόστος $J(\theta) \simeq 0.529$.

Γραφικά το decision boundary για $\lambda = 1$:

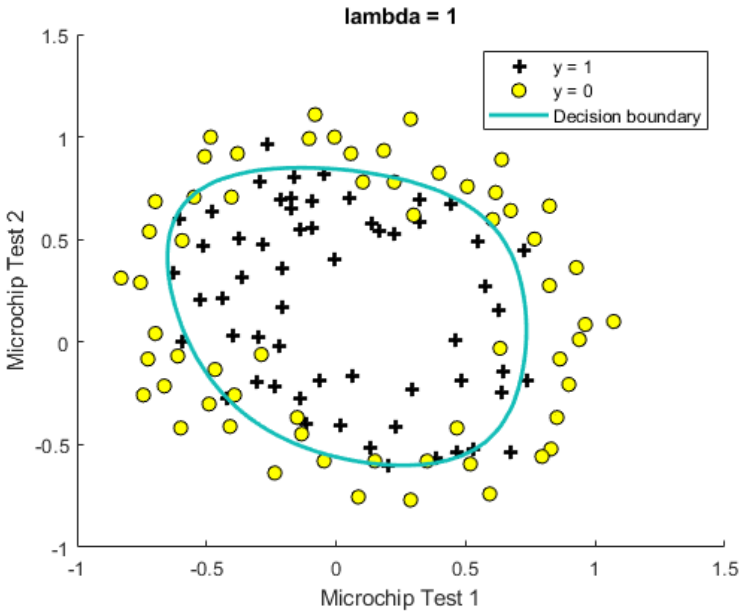


Figure 2: Decision Boundary

Αν πραγματοποιήθούν προβλέψεις πάνω στο set εκπαίδευσης, πετυχαίνουμε accuracy που φτάνει το **82,2%**.

Ανάλογα με τις τιμές του regularization factor, παρατηρούνται φαινόμενα overfitting και underfitting:

- **Overfitting**:

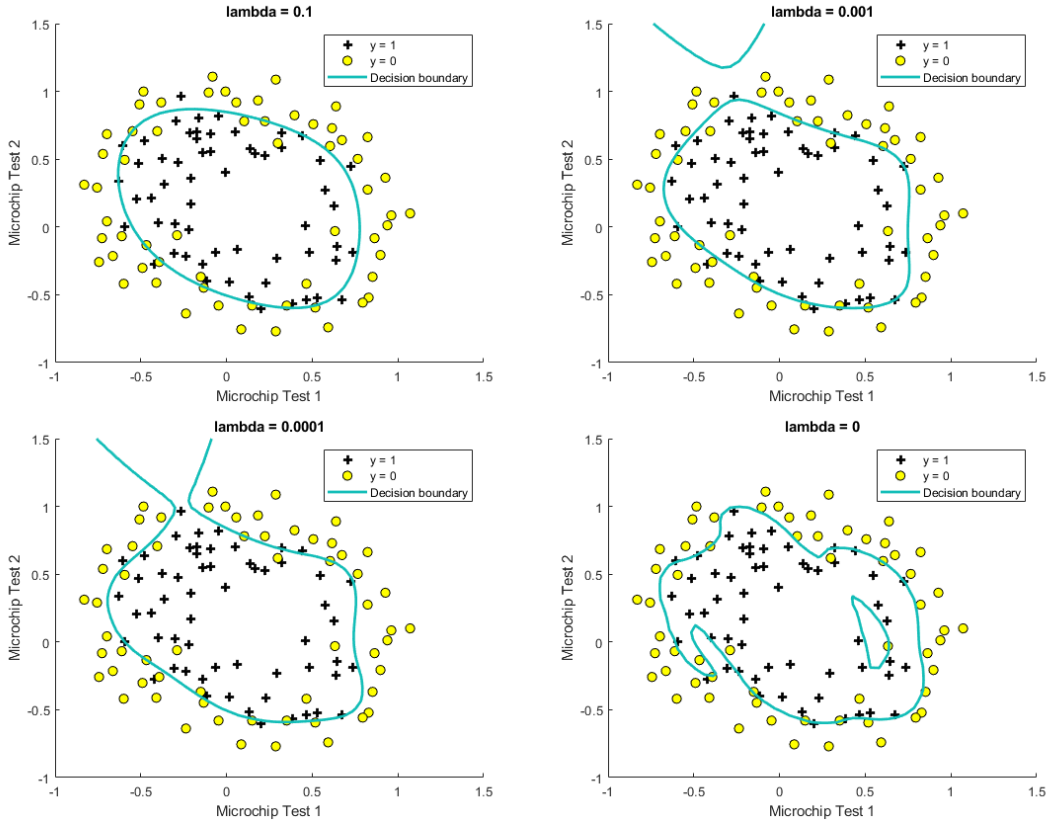


Figure 3: Decision Boundary with overfitting

Παρατηρούμε πώς, όσο μικρότερο γίνεται το λ, το μοντέλο μαθαίνει στα δεδομένα εκπαίδευσης, και προσπαθεί να φτιάξει ένα όριο απόφασης το οποίο θα προσπαθεί να εμπεριέχει δεδομένα μόνο μιας κλάσης και μάλιστα του **set εκπαίδευσης**. Παρατηρούμε ότι η πολυπλοκότητα του συνόρου, αυξάνεται αντίστροφος ανάλογα με το λ. Το παραπάνω αποδεικνύεται και μαθηματικά, μιας και το regularization term δεν επηρεάζει καθόλου τις συναρτήσεις κόστους και gradient, μιας και εξαλείφεται το φαινόμενο της ομαλοποίησης. Για προβλέψεις πάνω στο training set παρατηρείται μεγαλύτερο accuracy, αλλά στην περίπτωση εφαρμογής προβλέψεων, σε διαφορετικό set, το accuracy θα έπεφτε αρκετά.

- **Underfitting**:

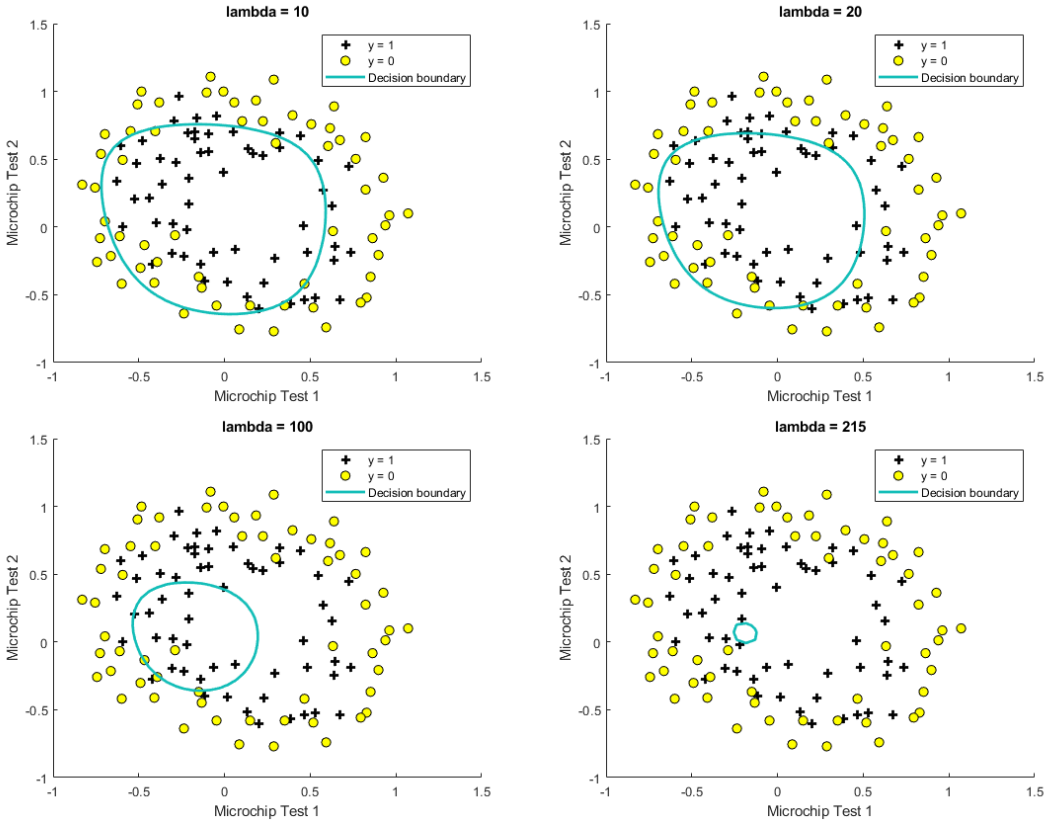


Figure 4: Decision Boundary with overfitting

Σε αυτή την περίπτωση, το μοντέλο δεν είναι σε θέση να καταγράψει τη σχέση μεταξύ των μεταβλητών εισόδου και εξόδου με ακρίβεια, δημιουργώντας υψηλό ποσοστό σφάλματος τόσο στο σύνολο εκπαίδευσης όσο και στα μη ορατά δεδομένα. Αυτό συμβαίνει μιας και το regularization term το οποίο είναι ανάλογο με το λ, επηρεάζει τις συναρτήσεις αναφοράς. Σαν αποτέλεσμα έχουμε ένα όριο απόφασης, το οποίο προσπαθεί να ορίσει περιοχή που θα έχει περισσότερα ή μόνο προϊόντα της μιας κλάσης. Αυτό οδηγεί σε χαμηλό accuracy τόσο στο set εκπαίδευσης όσο και σε διαφορετικά set.

3^{η} Parameter Estimation at Maximum Likelihood

Για n δείγματα, έστω $D = \{x_1, x_2, \cdots, x_n\}$, τα οποία παράγονται ανεξάρτητα από μια κατανομή Poisson με παράμετρο λ έστω:

$$p(X|\lambda) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad x \in [0, n] \ \& \ \lambda > 0$$

θα υπολογίσουμε τον εκτιμητή μέγιστης πιθανοφάνειας της παραμέτρου λ.

- Η συνάρτηση πιθανοφάνειας ορίζεται ως την παράγωγο της κατανομής για κάθε δείγμα:

$$L(\lambda|x_i) = \prod_{j=1}^{n} \frac{\lambda^{x_j} e^{-\lambda}}{x_j!}$$

- Θα υπολογίσουμε τον φυσικό λογάριθμο της παραπάνω συνάρτησης:

$$\ln\left(L(\lambda|x_i)\right) = \ln\left(\prod_{j=1}^{n} \frac{\lambda^{x_j} e^{-\lambda}}{x_j!}\right) \overset{\ln(ab)=\ln(a)+\ln(b)}{\Longrightarrow}$$

$$l(\lambda|x_i) = \sum_{j=1}^{n} \ln\left(\frac{\lambda^{x_j} e^{-\lambda}}{x_j!}\right) = \sum_{j=1}^{n} \Big[\ln(\lambda^{x_j}) + \ln(e^{-\lambda}) - \ln(x_j!)\Big] =$$

$$\sum_{j=1}^{n} \left[x_j \ln(\lambda) - \lambda - \ln(x_j!)\right] \Rightarrow$$

$$l(\lambda|x_i) = \ln(\lambda) \sum_{j=1}^{n} x_j - \sum_{j=1}^{n} \ln(x_j!) - n\lambda$$

- Θα υπολογίσουμε την παράγωγο αυτού ώς προς το λ:

$$\frac{\partial}{\partial\lambda} l(\lambda|x_i) = \frac{\partial}{\partial\lambda}\left(\ln(\lambda) \sum_{j=1}^{n} x_j - \sum_{j=1}^{n} \ln(x_j!) - n\lambda\right) =$$

$$\frac{\partial}{\partial\lambda}\left(\ln(\lambda) \sum_{j=1}^{n} x_j\right) - \frac{\partial}{\partial\lambda}\left(\sum_{j=1}^{n} \ln(x_j!)\right) - \frac{\partial}{\partial\lambda}\left(n\lambda\right) \Rightarrow$$

$$\frac{\partial}{\partial\lambda} l(\lambda|x_i) = \frac{1}{\lambda} \sum_{j=1}^{n} x_j - n$$

- Θα υπολογίσουμε το σημείο (λ) στο οποίο μηδενίζεται η παράγωγος, μιας και εκεί θα υπάρχει σημείο καμπής (αν ορίζεται).

$$\frac{\partial}{\partial\lambda} l(\lambda|x_i) = 0 \Rightarrow \frac{1}{\lambda} \sum_{j=1}^{n} x_j - n = 0 \Rightarrow$$

$$\lambda = \frac{1}{n} \sum_{j=1}^{n} x_j$$

Άρα ο ML εκτιμητής της παραμέτρου (όμοια με τον μέσω των δειγμάτων) ορίζεται ως:

$$\lambda = \frac{1}{n} \sum_{j=1}^{n} x_j$$

4^{η} : Support Vector Machines with KKT optimization

Αρχικά, έχουμε για τα δείγματα $x = [x_1, x_2]$ για τις αντίστοιχες κλάσεις ω_1, ω_2. Γραφικά:

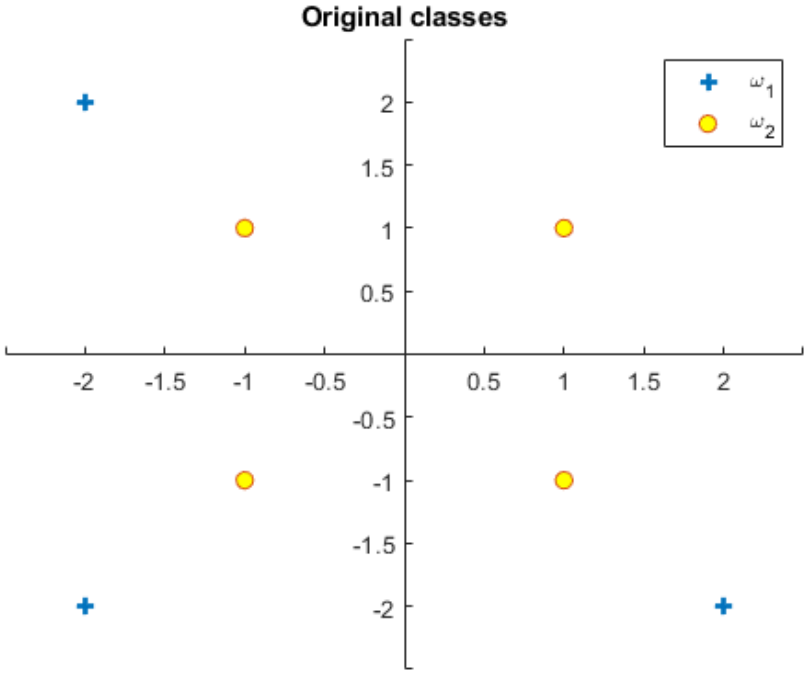


Figure 5: Original Data

Λόγο της γεωμετρίας των κλάσεων μπορούμε να διαπιστώσουμε ότι όλα τα σημεία και των δύο κλάσεων είναι support vectors. Για να ορίσουμε καλύτερα όμως το παραπάνω θα πρέπει να ορίσουμε το υπερεπίπεδο το οποίο διαχωρίζει τις κλάσεις.

Λόγου ότι οι κλάσεις είναι μη γραμμικές και άρα μη διαχωρίσιμα με classifier 2 διαστάσεων, για να εφαρμοστεί ο αλγόριθμος SVM θα πρέπει, να γίνει αντιστοίχηση των δεδομένων σε χώρο μεγαλυτέρων διαστάσεων, ώστε να μπορεί να γίνει η κατηγοριοποίηση αυτών. Σε αυτή την νέα διάσταση θα μπορούσε να οριστεί ένα υπερεπίπεδο το οποίο θα διαχωρίζει τα παραπάνω σύμφωνα με το τρόπο με το οποίο ορίστικε η τρίτη διάσταση (πχ $z = x_1^a + x_2^\beta + c$)

Βάση αυτών, θα μπορούσαμε να ισχυριστούμε ότι ο καλύτερος classifier θα μπορούσε να είναι ένα πολύγωνο και ιδανικά ένας κύκλος, όπου εσωτερικά αυτού θα είναι τα δεδομένα της ω_2 και εξωτερικά τα δεδομένα της ω_1.

Αν εφαρμόσουμε στα δεδομένα τον παρακάτω μετασχηματισμό:

$$\Phi(x) = x - ||x||_2^2 - 4, \quad ||x||_2^2 = x_1^2 + x_2^2$$

έχουμε τις παρακάτω κλάσεις:

$$\omega_1 : x^+ = \{[-10, -10]^T, [-10, -14]^T, [-14, -14]^T, [-14, -10]^T\}$$

$$\omega_2 : x^- = \{[-5, -5]^T, [-5, -7]^T, [-7, -7]^T, [-7, -5]^T\}$$

Γραφικά τα μετασχηματισμένα δεδομένα:

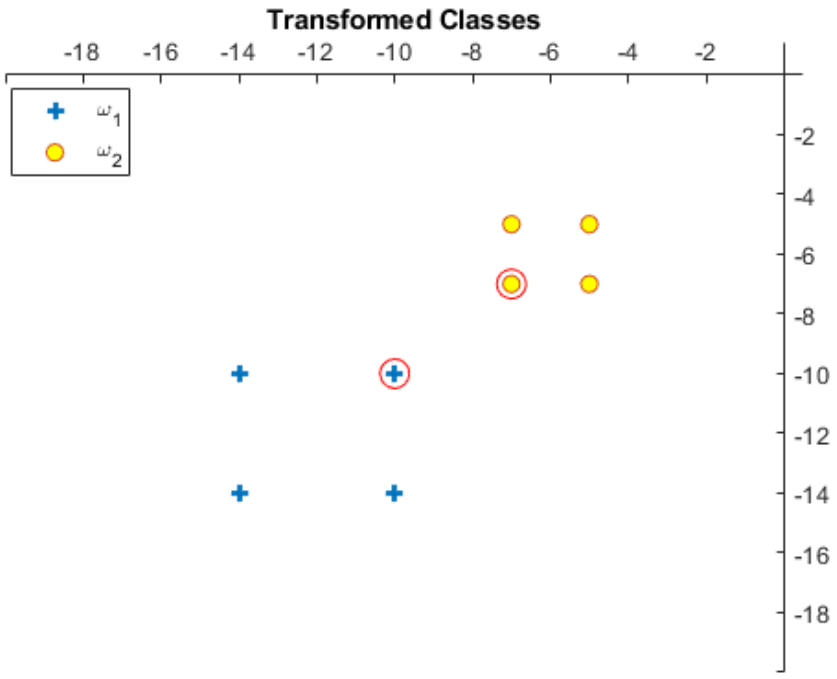


Figure 6: Trasformed Data

Σε αυτή την περίπτωση, το υπερεπίπεδο θα περνά μεταξύ των κλάσεων, και συγκεκριμένα, μεταξύ των support vectors (σημειώνονται με κόκκινο κύκλο) και εκατέρωθεν αυτού θα εμπεριέχονται οι ω_1, ω_2.

Για να βρούμε το βέλτιστο υπερεπίπεδο διαχωρισμού των μετασχηματισμένων δειγμάτων θα ξεκινήσουμε την Lagrangian συνάρτηση:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{N} \lambda_i \left[y_i \left(\mathbf{w^T x_i} + w_0 \right) - 1 \right]$$

Σε αυτή την περίπτωση, οι περιορισμοί γίνονται:

- $[-10 - 10]^T \Rightarrow 1(-10\omega_1 - 10\omega_2 + \omega_0) \geq 1 \Rightarrow -10\omega_1 - 10\omega_2 + \omega_0 \geq 1$
- $[-7 - 7]^T \Rightarrow -1(-7\omega_1 - 7\omega_2 + \omega_0) \geq 1 \Rightarrow 7\omega_1 7\omega_2 - \omega_0 \geq 1$

Επομένως έχουμε τις συναρτήσεις $f_i(\omega, \omega_0)$:

- $f_1(\omega, \omega_0) = -10\omega_1 - 10\omega_2 + \omega_0 - 1$
- $f_2(\omega, \omega_0) = 7\omega_1 + 7\omega_2 - \omega_0 - 1$

Άρα μπορούμε να ξαναγράψουμε την Lagrangian συνάρτηση:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \lambda_1 \left(-10\omega_1 - 10\omega_2 + \omega_0 - 1\right) - \lambda_2 \left(7\omega_1 + 7\omega_2 - \omega_0 - 1\right) =$$
$$\frac{1}{2}\left(\omega_1^2 + \omega_2^2\right) - \lambda_1 \left(-10\omega_1 - 10\omega_2 + \omega_0 - 1\right) - \lambda_2 \left(7\omega_1 + 7\omega_2 - \omega_0 - 1\right)$$

Για **KKT** πρέπει:

1.
$$\frac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial \omega} = 0$$

2.
$$\lambda_i \cdot f_i(\omega, \omega_0) = 0, \quad i \in [1, 2]$$

3.
$$\lambda_i \geq 0, \quad i \in [1, 2]$$

Άρα:

1.
$$\frac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial \omega_1} = \omega_1 + \lambda_1 10 - \lambda_2 7 = 0 \Rightarrow \omega_1 = -10\lambda_1 + 7\lambda_2$$
$$\frac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial \omega_2} = \omega_2 + \lambda_1 10 - \lambda_2 7 = 0 \Rightarrow \omega_2 = -10\lambda_1 + 7\lambda_2$$
$$\frac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial \omega_0} = -\lambda_1 + \lambda_2 = 0 \Rightarrow \lambda_1 = \lambda_2 = \lambda$$

2.
$$\lambda_1 \left(-10\omega_1 - 10\omega_2 + \omega_0 - 1\right) = 0 \Rightarrow \lambda_1 \left(-10 \cdot (-3\lambda) - 10 \cdot (-3\lambda) + \omega_0 - 1\right) = 0$$
$$\lambda \left(30\lambda + 30\lambda + \omega_0 - 1\right) = 0$$
$$30\lambda^2 + 30\lambda^2 + \omega_0 - 1 = 60\lambda^2 + \omega_0 - 1 = 0$$

Άρα $\lambda = 0$ ή $\lambda = \pm\frac{(-\omega_0+1)}{60}$

3.
$$\lambda_1, \lambda_2 \geq 0$$

(Λογικά θα πρέπει $\lambda = 0$)

5^{η} : Custom neural networks

Για τα δεδομένα, $D = (x^i, y^i)$ με $i \in [1, N]$, $x^i \in \mathcal{R}^{1x2}$ και $y^i \in \mathcal{R}$, εκπαιδεύουν ένα νευρωνικό δίκτυο το οποίο ορίζεται από την σχέση:

$$\hat{y}^i = f(x^i W + b), \quad x^i : sample \ \ \& \ \ f(z) = \frac{1}{1 + e^{-z}}$$

Το σφάλμα μεταξύ $\hat{y}^i$ και y^i υπολογίζεται από την συνάρτηση cross-entropy ως εξής:

$$J(y^i, \hat{y}^i; W, b) = -y^i \ln(\hat{y}^i) - (1 - y^i) \ln(1 - \hat{y}^i)$$

Για λόγους αριθμητικής ευστάθειας, το cross-entropy θα υπολογίζεται για υποσύνολα των δειγμάτων ή batches (B) και τέλος παίρνουμε το μέσο όρο αυτών. Άρα η συνάρτηση κόστους μπορεί να γραφεί ως:

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

- Αν θέσουμε ότι $z^i = x^i W + b$ και άρα $\hat{y}^i = f(z^i)$ η συνάρτηση κόστους μπορεί να ξαναγραφεί ώς:

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

$$\frac{-1}{B} \sum_i \left[y^i \ln\left(\frac{1}{1 + e^{-z^i}} \right) + (1 - y^i) \ln\left(1 - \frac{1}{1 + e^{-z^i}} \right) \right] =$$

$$\frac{-1}{B} \sum_i \left[y^i \left(\ln(1) - \ln(1 + e^{-z^i}) \right) + (1 - y^i) \ln\left(\frac{1 + e^{-z^i} - 1}{1 + e^{-z^i}} \right) \right] =$$

$$\frac{-1}{B} \sum_i \left[\cancel{y^i \ln(1)} - y^i \ln(1 + e^{-z^i}) + (1 - y^i) \left(\ln(e^{-z^i}) - \ln(1 + e^{-z^i}) \right) \right] =$$

$$\frac{-1}{B} \sum_i \left[\cancel{-y^i \ln(1 + e^{-z^i})} + \ln(e^{-z^i}) - \ln(1 + e^{-z^i}) - y^i \ln(e^{-z^i}) \cancel{+ y^i \ln(1 + e^{-z^i})} \right] =$$

$$\frac{-1}{B} \sum_i \left[-z^i - \ln(1 + e^{-z^i}) + y^i z^i \right] \leftrightarrow$$

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

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

- Αν παραγωγίσουμε την συνάρτηση κόστους ως προς το z^i έχουμε:

$$\frac{\partial J}{\partial z^i} = \frac{\partial}{\partial z^i}\frac{1}{B}\sum_i \left[z^i - y^i z^i + \ln(1+e^{-z^i})\right] \overset{\forall i \in B}{=}$$
$$\frac{\partial}{\partial z^i}\left[z^i - y^i z^i + \ln(1+e^{-z^i})\right] =$$
$$\frac{\partial}{\partial z^i} z^i - \frac{\partial}{\partial z^i} y^i z^i + \frac{\partial}{\partial z^i}\ln(1+e^{-z^i}) =$$
$$1 - y^i + \frac{1}{1+e^{-z^i}}\frac{\partial}{\partial z^i}\left(1+e^{-z^i}\right) =$$
$$1 - y^i + \frac{1}{1+e^{-z^i}}\left(e^{-z^i}\cdot(-1)\right) =$$
$$\frac{1+e^{-z^i}}{1+e^{-z^i}} - y^i - \frac{e^{-z^i}}{1+e^{-z^i}} =$$
$$-y^i + \frac{\cancel{1+e^{-z^i}} \cancel{-e^{-z^i}}}{1+e^{-z^i}} =$$
$$-y^i + \frac{1}{1+e^{-z^i}} \leftrightarrow$$
$$\frac{\partial J}{\partial z^i} = -y^i + \hat{y}^i \quad i \in [1, \#batches]$$

- Αν αντίστοιχα, παραγωγίσουμε το κόστος, ως προς το αντίστοιχο βάρος με την χρήση του κανόνα αλυσίδας έχουμε:

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

Γνωρίζουμε από πριν, ότι η παραγώγιση συνάρτησης κόστους ως προς το όρισμα της συνάρτησης ενεργοποίησης ορίζεται ως:

$$\frac{\partial J}{\partial z^i} = -y^i + \hat{y}^i \quad i \in [1, \#batches] \Rightarrow \frac{\partial J}{\partial \mathbf{z}} = -\mathbf{y} + \hat{\mathbf{y}}$$

Αν παραγωγίσουμε το όρισμα της συνάρτησης ενεργοποίησης ως προς το αντίστοιχο βάρος έχουμε:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}(\mathbf{x}\mathbf{W} + b) \Rightarrow$$
$$\frac{\partial \mathbf{z}}{\partial \mathbf{W}} = \mathbf{x}^T$$

Άρα η παράγωγος του σφάλματος, ως προς το αντίστοιχο βάρος, σε ένα μεμονωμένο batch δίνεται από τον τύπο:

$$\frac{\partial J}{\partial \mathbf{W}} = -\mathbf{y} + \hat{\mathbf{y}}\cdot\mathbf{x} = -y^{(i)} + \hat{y}^{(i)}\cdot {x^{(i)}}^T \quad i \in [1, \#batches]$$

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{1}{N}\sum_{n=1}^{N} -y^{(i)}(n) + \hat{y}^{(i)}(n)\cdot {x^{(i)}(n)}^T \quad i \in [1, \#batches], \quad N = \ \#\text{sumples in batch}$$

Αντίστοιχα, για να υπολογίσουμε την παράγωγο σφάλματος ως προς το bias, όμοια με τον κανόνα αλυσίδας έχουμε:

$$\frac{\partial J}{\partial \mathbf{b}} = \frac{\partial J}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{b}} =$$
$$-\mathbf{y} + \hat{\mathbf{y}}\cdot\frac{\partial}{\partial \mathbf{b}}(\mathbf{x}\mathbf{W} + \mathbf{b}) \Rightarrow$$
$$\frac{\partial J}{\partial \mathbf{b}} = -y^i + \hat{y}^i \quad i \in [1, \#batches]$$

- Γνωρίζοντας τις παραπάνω παραγώγους μπορούμε να αναλύσουμε το φαινόμενο του back propagation δηλαδή τον ρυθμό με τον οποίο μαθαίνει το κάθε node, κάθε επιπέδου.
  - **Initialization**: Αρχικά ορίζονται τα αρχικά βάρη του νευρωνικού δικτύου τα οποία θα επεξεργαστούν αργότερα, για να φτάσουν τις επιθυμητές τιμές.
  - **Forward pass**: Τα σήματα εισόδου διαδίδονται προς τα εμπρός στο νευρωνικό δίκτυο. Συγκεκριμένα, τα δείγματα περνάνε σαν είσοδος από κάθε node του πρώτου layer του δικτύου. Κάθε node παράγει μια έξοδος η οποία γίνεται propagate στα node του επόμενου layer και ούτω καθεξής μέχρι να φτάσουμε στο output του δικτύου. Ποιο συγκεκριμένα γίνονται υπολογισμοί για κάθε node κάθε επιπέδου:

$$z^{(i)} = W^{(i)}x^{(i-1)} + b^{(i)}, \quad x^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}), \quad i \in [2, N]$$

  - **Loss calculation**: Μέσο της συνάρτησης κόστους cross-entropy υπολογίζεται το σφάλμα της εξόδου (βασίζεται στα label δεδομένα εκπαίδευσης) στο τελευταίο layer. Συγκεκριμένα έχουμε το συνολικό κόστος ως:

$$J_{total}(i) = \sum_{j=1}^{N} J(i), \quad j \in \#\text{nodes}, i \in \#\text{ sumple}$$

  - **Backward pass**: Γνωρίζοντας το σφάλμα της εξόδου, επιστρέφουμε προς τα αριστερά του δικτύου την παράγωγο του σφάλματος ως προς τα αντίστοιχο βάρος. Αρχικά, υπολογίζουμε το κόστος ως προς το z (activation):

$$\delta_j^{(r)}(i) \equiv \frac{\partial J_j(i)}{\partial z_j^{(r)}(i)} \quad , j \in \#\text{node at layer(r)}, i \in \#\text{ sumple}$$

    Με βάση το παραπάνω θα υπολογίσουμε το αντίστοιχο $\delta_j^{r-1}(i)$ δηλαδή το κόστος του node του προηγούμενου layer με την χρήση του κανόνα αλυσίδας:

$$\delta_j^{r-1}(i) \equiv \frac{\partial J_j(i)}{\partial z_j^{(r-1)}(i)} =$$

$$\sum_k \frac{\partial J_j(i)}{\partial z_k^{(r)}(i)} \frac{\partial z_k^{(r)}(i)}{\partial z_j^{(r-1)}(i)} =$$

$$\sum_k \delta_k^{(r)}(i) \frac{\partial z_k^{(r)}(i)}{\partial z_j^{(r-1)}(i)}$$

    Ο άγνωστος όρος μπορεί να γραφεί ως εξής (όμοια με το δεύτερο βήμα το x του z ορίζεται από την συνάρτηση ενεργοποίησης του προηγούμενου επιπέδου):

$$\frac{\partial z_j^{(r)}(i)}{\partial z_j^{(r-1)}(i)} = w_{kj}^{(r)} \sigma'(z_j^{(r-1)}(i))$$

    Άρα το κόστος του κρυφού node μπορεί μπορεί να υπολογιστεί από την σχέση:

$$\delta_j^{r-1}(i) \equiv \sum_k \delta_k^{(r)}(i) w_{kj}^{(r)} \sigma'(z_j^{(r-1)}(i)) = \left(\delta_k^{(r)}(i) w_{kj}^{(r)}\right) \odot \sigma'(z_j^{(r-1)}(i))$$

  - **Weight update** Ξέροντας τον τρόπο με τον οποίο το τελικό σφάλμα θα επηρεάσει τα επιμέρους nodes, μπορούμε να αλλάξουμε τα αντίστοιχα βάρη και biases σύμφωνα με:

$$w^l = w^l - p \sum_k \delta^{(x,l)} \sigma^{(x,(l-1))^T}$$

$$b^l = b^l - p \sum_k \delta^{(x,l)}$$

Στην περίπτωσή μας, γνωρίζουμε gradient συναρτήσεις και μιλάμε για δίκτυο με ένα layer. Άρα τα updates θα γίνουν σύμφωνα με τις εξής:

$$w = w - p \sum_k (-y^i + \hat{y}^i) \cdot \sigma^{(x,(l-1))^T}$$

$$b = b - p \sum_k (-y^i + \hat{y}^i)$$

– **Repeat**: Επανάληψη της παραπάνω διαδικασίας μέχρι να φτάσουμε ελάχιστο σφάλμα.

- Σύμφωνα με τις παραπάνω σχέσεις ολοκληρώθηκε ο αντίστοιχος κώδικας που έλειπε και έγιναν οι εξής παρατηρήσεις:

  – Για το αρχικό μοντέλο το οποίο αποτελείται από το μοντέλο:

    * Dense $((28x28), 50)$
    * Sigmoid()
    * Dense $(50, 10)$
    * Softmax()

  και εκπαιδεύεται σύμφωνα με τα εξής κριτήρια:

    * #epochs = 100
    * learning_rate = 0.1
    * batch_size = 128

  Αρχικά παρατηρείτε ο τρόπος με τον οποίο τροποποιείται το κόστος σφάλματος κατά την διάρκεια της εκπαίδευσης και για κάθε epoch:

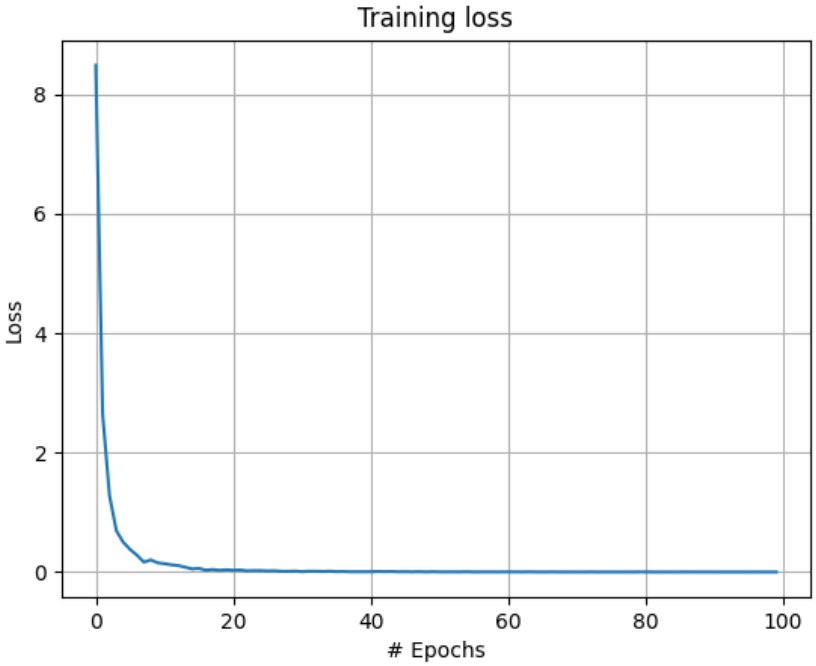


Figure 7: Training loss for simple model with use of cross entropy error (min: $\simeq 56e-5$)

Βλέπουμε ότι το σφάλμα ελαχιστοποιήται μετά τα 20 πρώτα epoch.

Το ίδιο μοντέλο, κατάφερε να κάνει τις παρακάτω προβλέψεις με:

* ration: 0.9
* mse: 0.0148

```
pred: 4         true: 4
pred: 9         true: 9
pred: 2         true: 5
pred: 9         true: 9
pred: 0         true: 0
pred: 2         true: 6
pred: 9         true: 9
pred: 0         true: 0
pred: 1         true: 1
pred: 5         true: 5
pred: 9         true: 9
pred: 7         true: 7
pred: 3         true: 3
pred: 4         true: 4
ratio: 0.90
mse: 0.0148
```

– Για το ίδιο μοντέλο τροποποιήσαμε τις παραμέτρους εκπαίδευσης ως εξής:

* #epochs = 50
* learning_rate = 0.001
* batch_size = 64

Η νέα γραφική σφάλματος που προκύπτει:

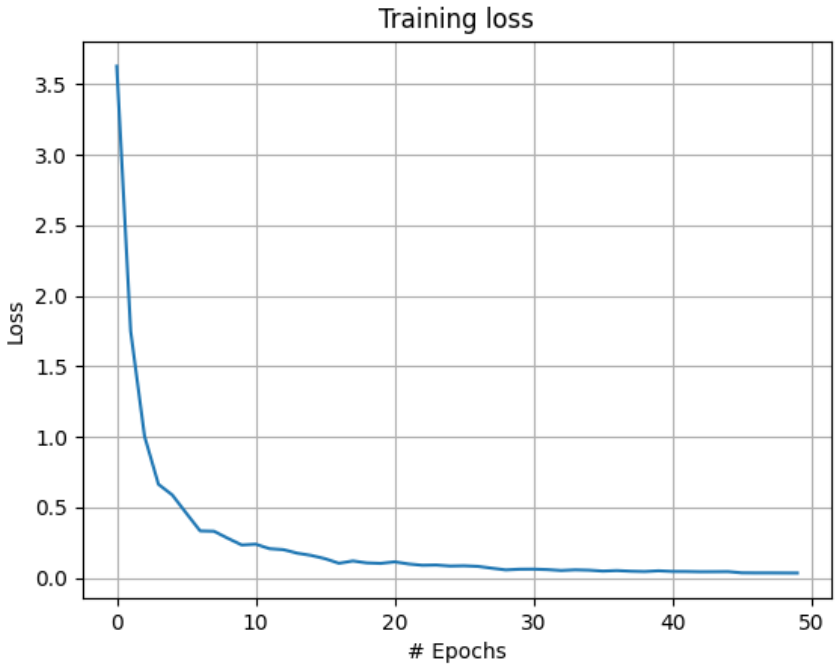


Figure 8: Training loss for simple model with use of cross entropy error (min: $\simeq 34e-3$)

Μια πρώτη παρατήρηση είναι ότι το σφάλμα ελαχιστοποιείται σε τιμή περίπου δύο τάξεων μεγέθους ποιο πάνω από την προηγούμενη περίπτωση, πράγμα το οποίο οφείλεται στο χαμηλό learning rate.

Όμοια παρατηρείται ελαχιστοποίηση σε περίπου 30 epochs.

Με την χρήση χαμηλότερου batch_size γίνατε μια προσπάθεια να βελτιστοποίησης του μοντέλου.

Το ίδιο μοντέλο, κατάφερε να κάνει τις παρακάτω προβλέψεις με:

* ration: 0.15
* mse: 0.1159

```
pred: 0         true: 7
pred: 3         true: 2
pred: 4         true: 1
pred: 0         true: 0
pred: 4         true: 4
pred: 1         true: 1
pred: 7         true: 4
pred: 3         true: 9
pred: 9         true: 5
pred: 4         true: 9
pred: 6         true: 0
pred: 1         true: 6
pred: 7         true: 9
pred: 4         true: 0
pred: 8         true: 1
pred: 4         true: 5
pred: 3         true: 9
pred: 0         true: 7
pred: 9         true: 3
pred: 6         true: 4
ratio: 0.15
mse: 0.1159
```

– Αν αλλάξουμε την συνάρτηση ενεργοποίησης σε αυτήν της tanh() και με υπολογισμό σφάλματος με τον τρόπο υπολογισμού σφάλματος mse.

Το μοντέλο πλέον ορίζεται ως:

* Dense $((28x28), 50)$
* Tanh()
* Dense $(50, 10)$
* Softmax()

και εκπαιδεύεται σύμφωνα με τα εξής κριτήρια:

* #epochs = 100
* learning_rate = 0.1
* batch_size = 128

Το κόστος σφάλματος:

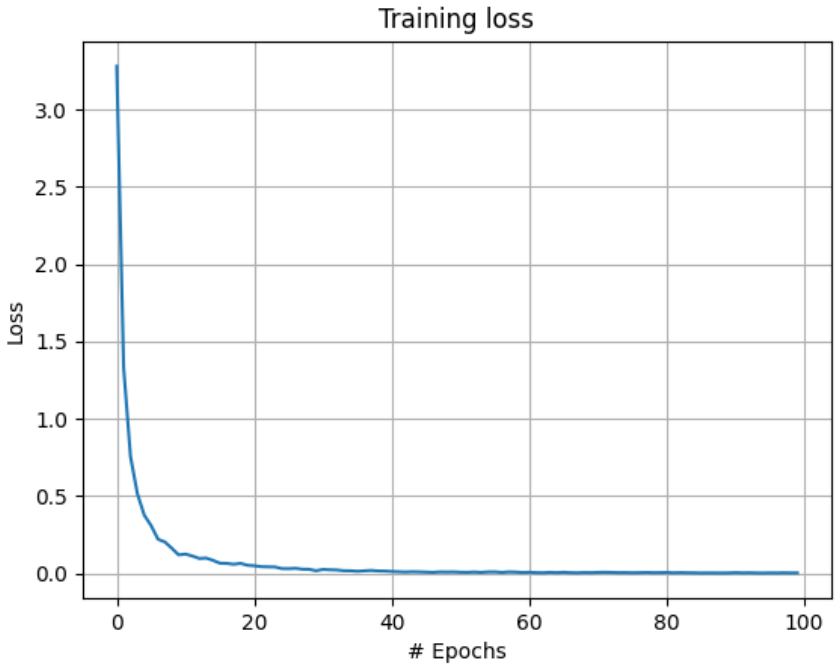


Figure 9: Training loss for simple model with use of MSE and Tahn activation function (min: $\simeq 25e{-}4$)

Βλέπουμε ότι το σφάλμα ελαχιστοποιήται μετά τα 40 πρώτα epoch.

Το ίδιο μοντέλο, κατάφερε να κάνει τις παρακάτω προβλέψεις με:

* ration: 0.65
* mse: 0.0687

```
pred: 7         true: 7
pred: 8         true: 2
pred: 1         true: 1
pred: 0         true: 0
pred: 0         true: 4
pred: 1         true: 1
pred: 5         true: 4
pred: 9         true: 9
pred: 2         true: 5
pred: 9         true: 9
pred: 0         true: 0
pred: 0         true: 6
pred: 9         true: 9
pred: 0         true: 0
pred: 1         true: 1
pred: 5         true: 5
pred: 2         true: 9
pred: 7         true: 7
pred: 5         true: 3
pred: 4         true: 4
ratio: 0.65
mse: 0.0687
```

– Τέλος, φτιάχτηκε ένα νέο μοντέλο με την χρήση των παρακάτω layers:

  * Dense $((28x28), 36)$
  * Sigmoid()
  * Dense $(36, 64)$
  * Sigmoid()
  * Dense $(64, 36)$
  * Sigmoid()
  * Dense $(36, 10)$
  * Softmax()

  Εκπαιδεύεται σύμφωνα με τα εξής κριτήρια:

  * #epochs $= 100$
  * learning_rate $= 0.1$
  * batch_size $= 32$

  Το κόστος σφάλματος:

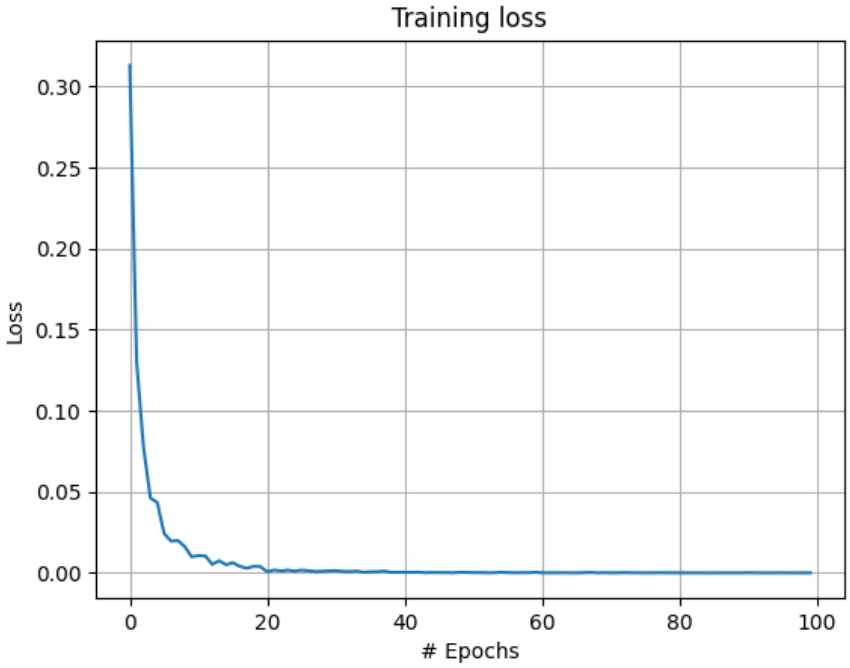


Figure 10: Training loss of custom model (min: $\simeq 4e-5$)

Βλέπουμε ότι το σφάλμα ελαχιστοποιείται μετά τα 20 πρώτα epoch.

Το ίδιο μοντέλο, κατάφερε να κάνει τις παρακάτω προβλέψεις με:

* ration: 0.85
* mse: 0.0261

```
pred: 7         true: 7
pred: 2         true: 2
pred: 1         true: 1
pred: 0         true: 0
pred: 4         true: 4
pred: 1         true: 1
pred: 3         true: 4
pred: 9         true: 9
pred: 2         true: 5
pred: 9         true: 9
pred: 0         true: 0
pred: 2         true: 6
pred: 9         true: 9
pred: 0         true: 0
pred: 1         true: 1
pred: 5         true: 5
pred: 9         true: 9
pred: 7         true: 7
pred: 3         true: 3
pred: 4         true: 4
ratio: 0.85
mse: 0.0261
```

– Αξίζει να σημειωθεί ότι σε καμία από τις παραπάνω περιπτώσεις δεν μπορέσαμε να κάνουμε σωστή πρόβλεψη για το 5. Επίσης δεν έγιναν τυπώσεις για τους αριθμούς μιας και ήταν πολλές οι περιπτώσεις που έπρεπε να ελέγξουμε.

$6^{η}$: Convolutional neural networks

- Αρχικά έχουμε το παρακάτω μοντέλο το οποίο θα εκπαιδεύσουμε:

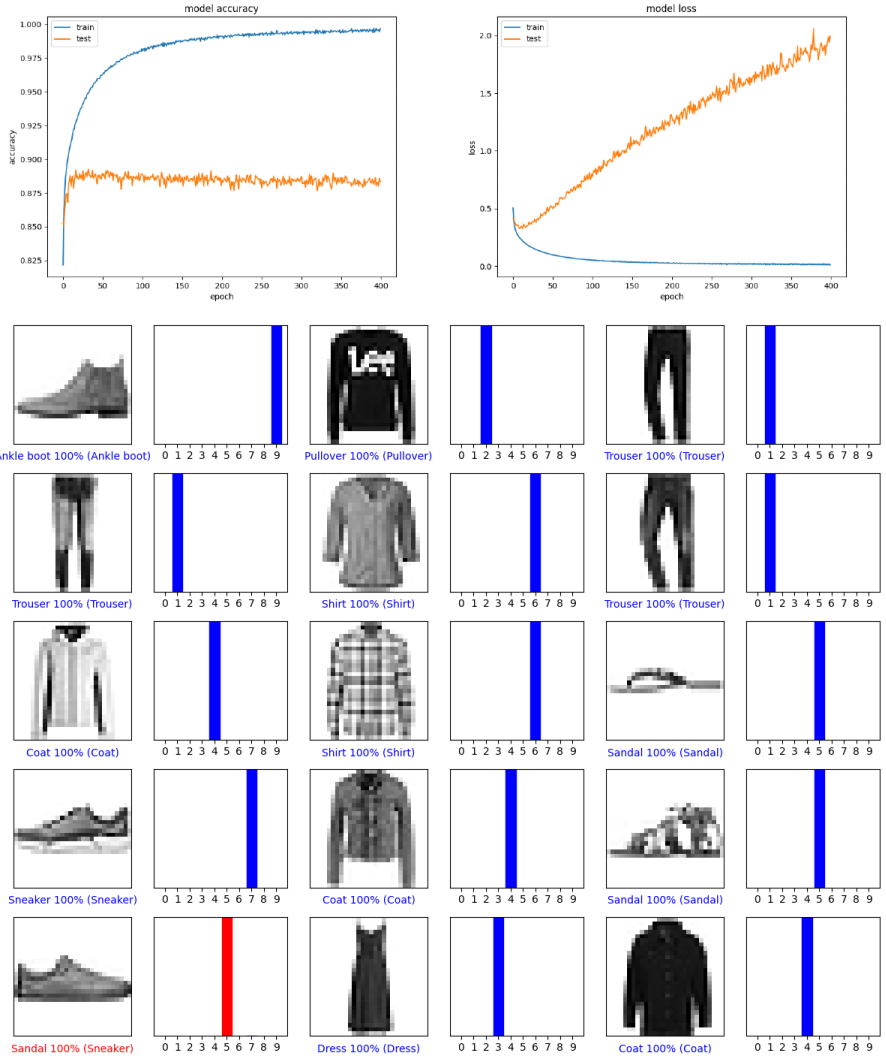
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

Εκπαιδεύοντας το παραπάνω και χρησιμοποιώντας τω, έχουμε τις παρακάτω προβλέψεις με χρήση **adam** αλγορίθμου βελτιστοποίησης:



Παρατηρείται ότι οι μεταβλητές accuracy κατά το training και τον ελέγχω αυξάνονται με λογαριθμικό τρόπο, ενώ οι αντίστοιχες τιμές κόστους κατά την εκπαίδευση φτάνουν σε τοπικό ελάχιστο κατά την διάρκεια της εκπαίδευσης με την χρήση των εκάστοτε epochs. Το test error αυξάνεται μιας και το train error παραμένει σταθερό.

- Στην συνέχεια έγιναν έλεγχοι για την εύρεση του καλύτερου αλγορίθμου βελτιστοποίησης.
  - **Adamax, SDG**

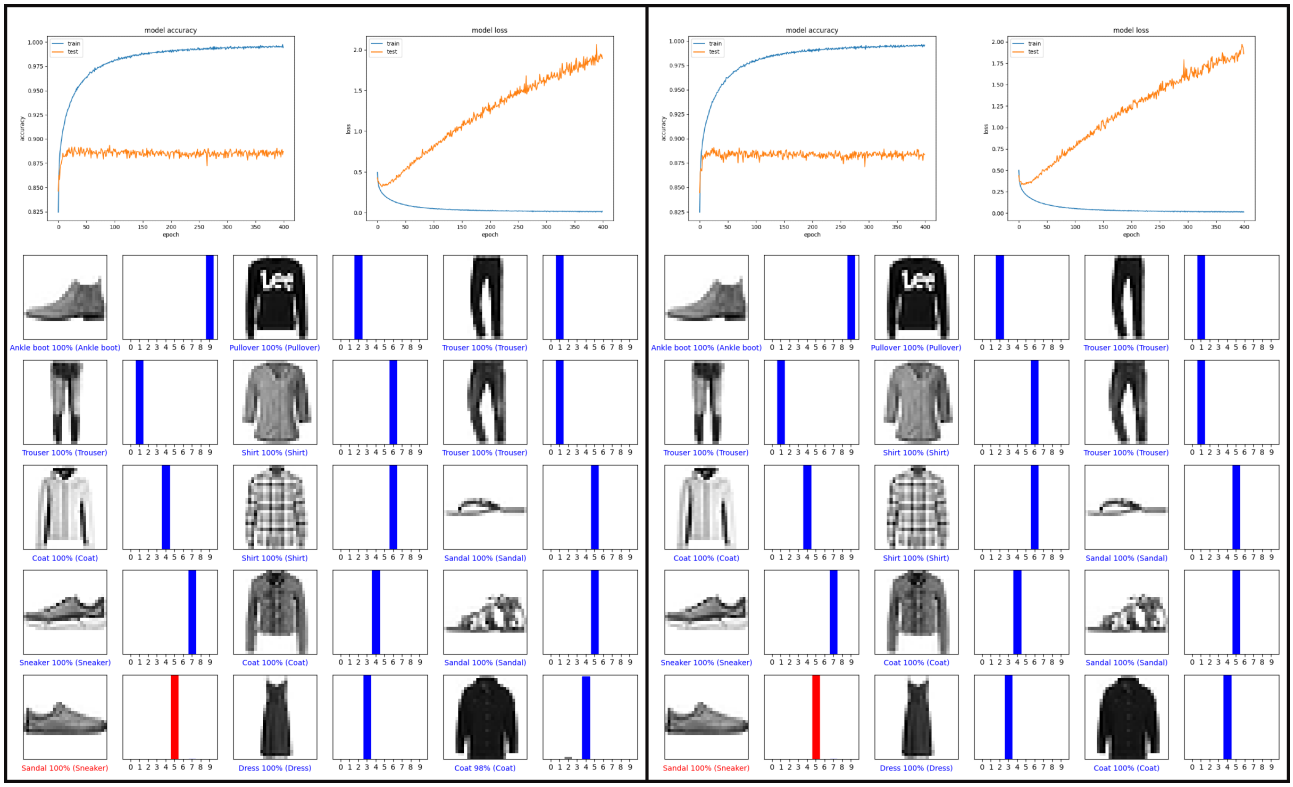


Figure 11: left: **Adamax**, right: **SDG**

– **RMSprop, FTRL**

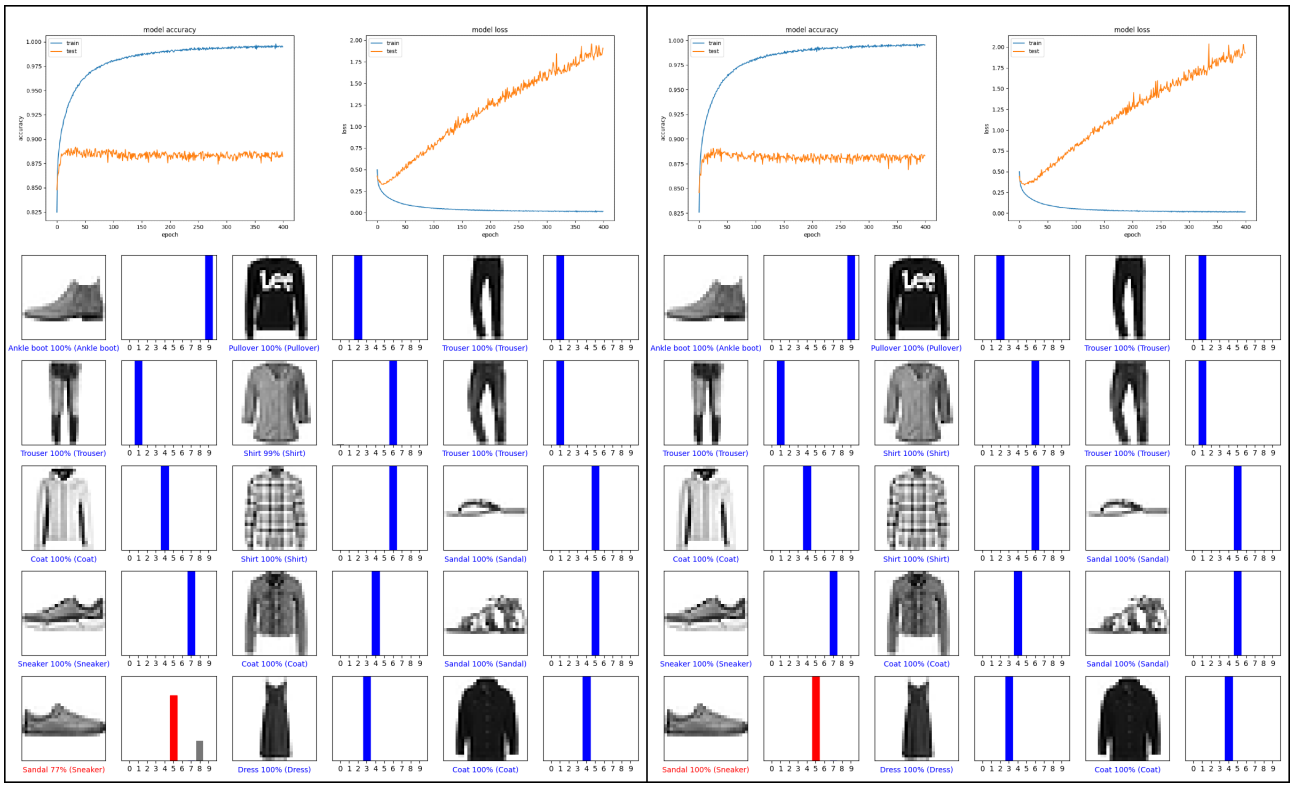


Figure 12: left: **RMSprop**, right: **FTRL**

– **Nadam**

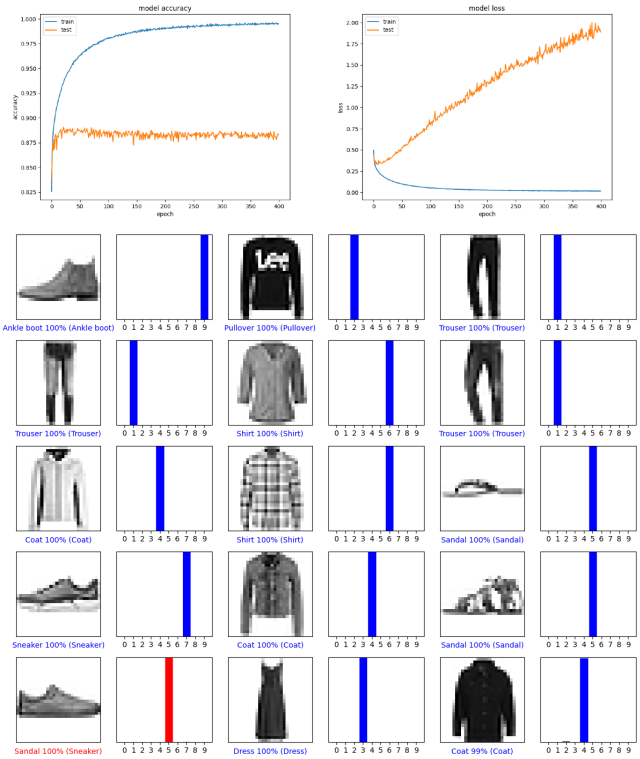


Figure 13: **Nadam**

Παρατηρείται ότι όλες οι συναρτήσεις βελτιστοποίησης έχουν την ίδια συμπεριφορά και κάνουν τις προβλέψεις με μόνη διαφορά τον βελτιστοποιητή **RMSprop** που για την λάθος πρόβλεψη αναγνωρίζει και την σωστή επιλογή (με μικρότερο ποσοστό).

– Σε αυτή την φάση θα φτιαχτεί μοντέλο με χρήση συνελικτικών νευρωνικών δικτύων (CNN) το οποίο θα εκπαιδευτεί πάνω στα ίδια δεδομένα για να γίνουν οι ίδιες προβλέψεις

Το μοντέλο θα είναι της μορφής: Το οποίο μπορεί γραφικά παρουσιάζεται ως:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 14, 14, 64)        18496

 conv2d_3 (Conv2D)           (None, 14, 14, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 128)         73856

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 128)        0
 2D)

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 128)               147584

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 287,722
Trainable params: 287,722
Non-trainable params: 0
_________________________________________________________________
```

Figure 14: CNN model

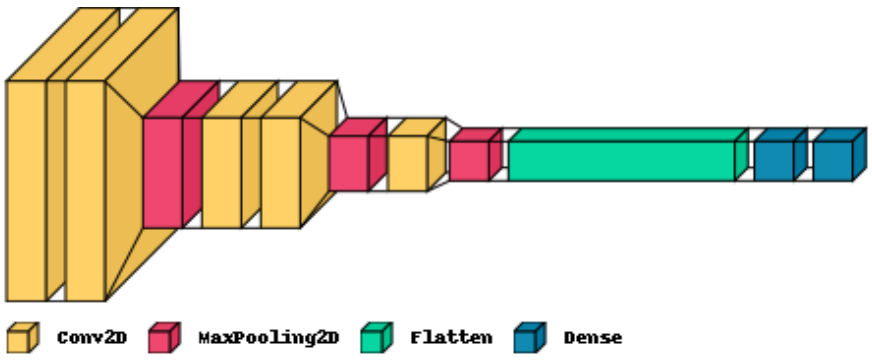


Figure 15: CNN model

Μετά το πέρας της εκπαίδευσης το, η παραπάνω αρχιτεκτονική οδηγεί σε ίδιες προβλέψεις με την προηγούμενη περίπτωση:

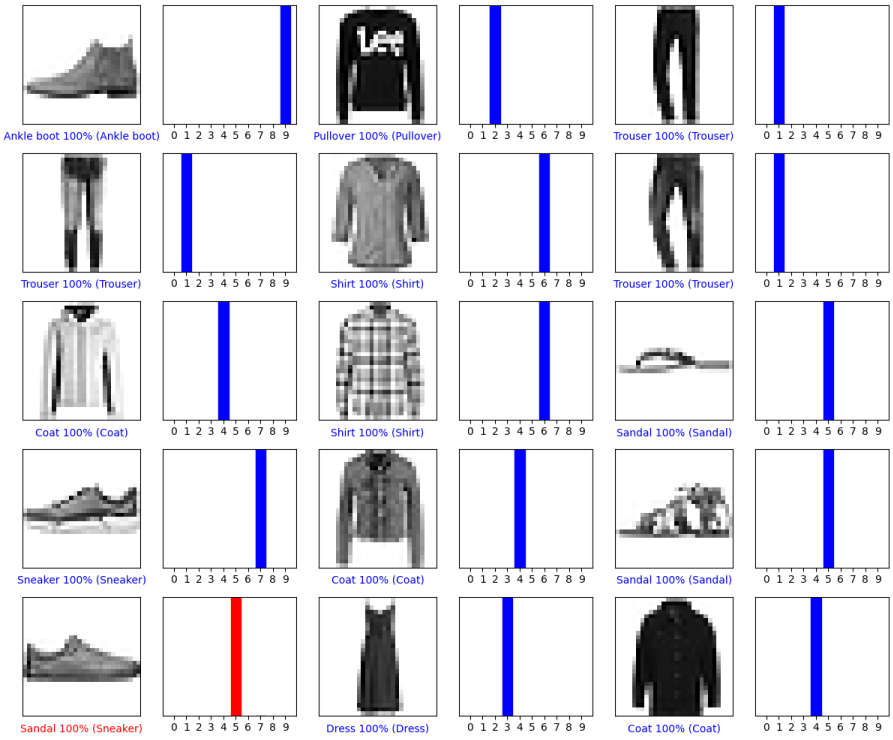


Figure 16: CNN model predictions (acc: $\simeq 90\%$)

– Στην συνέχεια εισηγήθηκαν τα Batch Normalization layers τα οποία οδήγησαν στην αρχιτεκτονική:

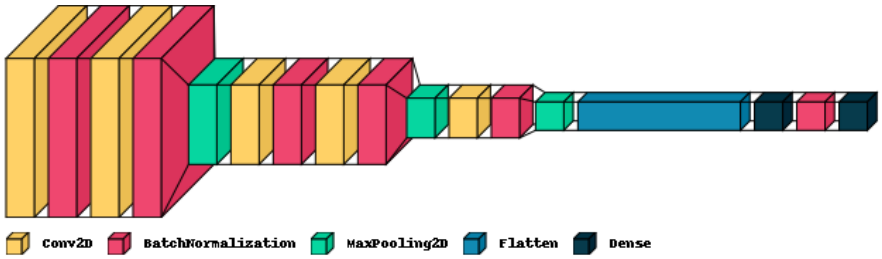


Figure 17: CNN model with Batch Normalization

Αντίστοιχα, το μοντέλο πρόβλεψε:

Figure 18: CNN model with batch Normalization predictions (acc: $\simeq 93\%$)

– Τέλος μπήκαν και DropOut layers για να μειώσουμε το φαινόμενο του overfitting:

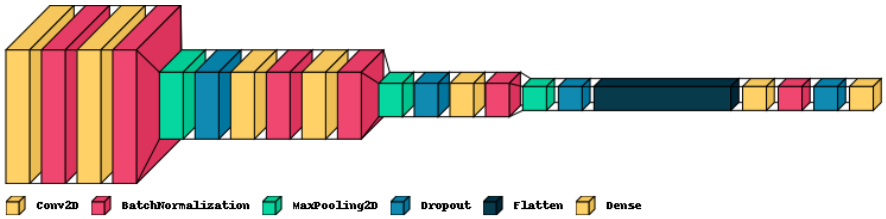


Figure 19: CNN model with Batch Normalization

Αντίστοιχα, το μοντέλο πρόβλεψε:

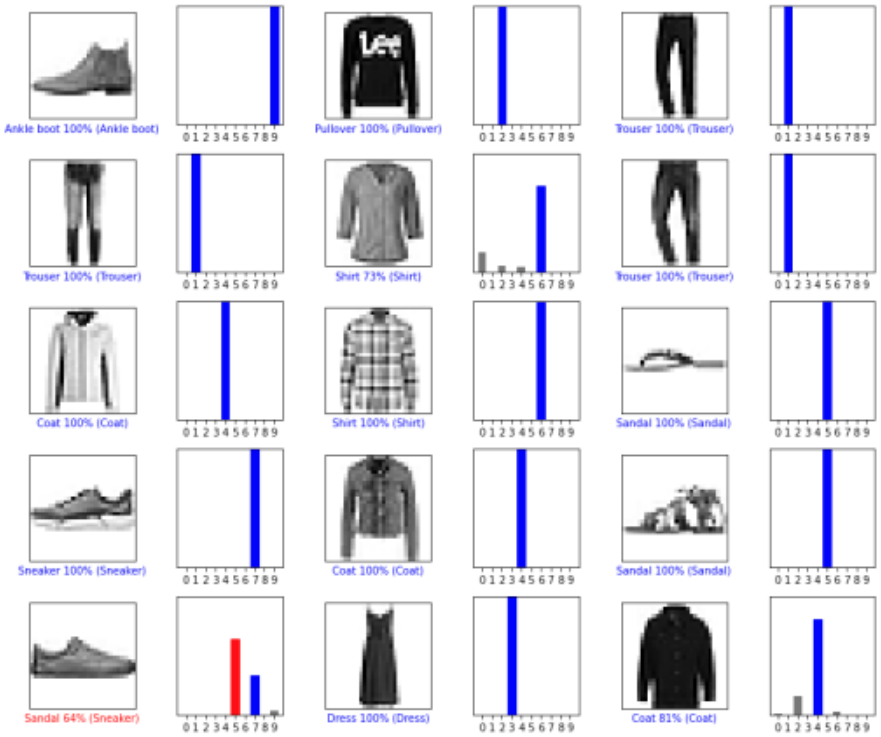


Figure 20: CNN model with Dropout predictions (acc: $\simeq 93\%$)