# Τεχνητή Νοημοσύνη 2
# Εργασία 2

**Ιωάννης Καπετανγεώργης**
**111520180061**

# Περιεχόμενα

# Μέρος Α – Gradient of Cross Entropy Loss

Έστω:

$$J = CE(y, \hat{y}) \text{ (όπου CE: Cross Entropy Loss)}$$
$$\hat{y} = softmax(\theta)$$

Θα αποδείξουμε ότι το gradient του Cross entropy loss είναι ίσο με $\hat{y} - y$,

δηλαδή ότι $\frac{dJ}{d\theta} = \hat{y} - y$

Αρχικά για τον τύπο του Cross Entropy loss έχουμε:

$$J = CE(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \quad \textbf{(1)}$$

όπου **K** ο συνολικός αριθμός των κλάσεων.

Επίσης για κάθε $\hat{y}_k$ έχουμε:

$$\hat{y}_k = softmax(s(x))_k = \frac{e^{s_k(x)}}{\sum_{j=1}^{K} e^{s_j(x)}} \quad \textbf{(2)}$$

Για λόγους απλότητας, από εδώ και στο εξής θα χρησιμοποιώ τον συμβολισμό:
$\theta = s_k(x) = s_k$ όπου το $s_k$ αντιστοιχεί στο διάνυσμα το οποίο περιέχει τα score της κάθε κλάσης για την οντότητα x.

Αρχικά υπολογίζω το gradient του $\hat{y}_k$ ως προς $s_i$.
Έχουμε δύο περιπτώσεις, $i = k$ και $i \neq k$.

- **Για $i = k$, έχουμε:**

$$\frac{d\,\hat{y}_k}{d\,s_k} = \frac{d}{d\,s_k}\frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}} = \frac{\left(\frac{d}{d\,s_k}e^{s_k}\right)\sum_{j=1}^{K} e^{s_j} - e^{s_k}\left(\frac{d}{d\,s_k}\sum_{j=1}^{K} e^{s_j}\right)}{\left(\sum_{j=1}^{K} e^{s_j}\right)^2}$$

$$= \frac{e^{s_k}\sum_{j=1}^{K} e^{s_j} - e^{s_k}\sum_{j=1}^{K} e^{s_j}}{\left(\sum_{j=1}^{K} e^{s_j}\right)^2} = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}} - \frac{e^{s_k}\,e^{s_k}}{\left(\sum_{j=1}^{K} e^{s_j}\right)^2} = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}\left(1 - \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}\right)$$

Αφού: $\left(\frac{d}{d\,s_k}e^{s_k}\right) = e^{s_k}$

Και $\frac{d}{d\,s_k}\sum_{j=1}^{K} e^{s_j} = \sum_{j=1}^{K}\frac{d}{d\,s_k}e^{s_j} = 0 + 0 + \cdots + \frac{d}{d\,s_k}e^{s_k} + \cdots + 0 = e^{s_k}$

Τέλος, λόγω της (2) έχουμε: $\frac{d\,\hat{y}_k}{d\,s_k} = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}\left(1 - \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}\right) = \hat{y}_k(1 - \hat{y}_k)$ **(3)**

- **Για $i \neq k$, έχουμε:**

$$\frac{d\,\hat{y}_k}{d\,s_i} = \frac{d}{d\,s_i}\frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}} = \frac{\left(\frac{d}{d\,s_i}e^{s_k}\right)\sum_{j=1}^{K} e^{s_j} - e^{s_k}\left(\frac{d}{d\,s_i}\sum_{j=1}^{K} e^{s_j}\right)}{\left(\sum_{j=1}^{K} e^{s_j}\right)^2}$$

$$= \frac{(0)\sum_{j=1}^{K} e^{s_j} - e^{s_k}e^{i}}{\left(\sum_{j=1}^{K} e^{s_j}\right)^2} = \frac{-e^{s_k}e^{i}}{\left(\sum_{j=1}^{K} e^{s_j}\right)^2} = -\frac{e^{s_k}}{\left(\sum_{j=1}^{K} e^{s_j}\right)}\frac{e^{s_i}}{\left(\sum_{j=1}^{K} e^{s_j}\right)}$$

Αφού: $\left(\frac{d}{d\,s_i}e^{s_k}\right) = 0$ , για $\boldsymbol{i \neq k}$

Και $\frac{d}{d\,s_i}\sum_{j=1}^{K} e^{s_j} = \sum_{j=1}^{K}\frac{d}{d\,s_i}e^{s_j} = 0 + 0 + \cdots + \frac{d}{d\,s_i}e^{s_i} + \cdots + 0 = e^{s_i}$

Τέλος, λόγω της (2) έχουμε: $\frac{d\,\hat{y}_k}{d\,s_i} = -\frac{e^{s_k}}{\left(\sum_{j=1}^{K} e^{s_j}\right)}\frac{e^{s_i}}{\left(\sum_{j=1}^{K} e^{s_j}\right)} = -\hat{y}_k\,\hat{y}_i$ **(4)**

Με βάση την σχέση (1) έχουμε:

$$\frac{dJ}{d\,s_i} = \frac{d}{d\,s_i}\left(-\sum_{k=1}^{K} y_k \log(\hat{y}_k)\right) = -\sum_{k=1}^{K}\frac{d}{d\,s_i}\left(\,y_k \log(\hat{y}_k)\,\right)$$

$$= -\sum_{k=1}^{K} y_k \frac{d}{d\,s_i}\left(\,\log(\hat{y}_k)\,\right)$$

Για το $\frac{d}{d\,s_i}\left(\,\log(\hat{y}_k)\,\right)$ εφαρμόζουμε τον κανόνα της αλυσίδας και έχουμε:

$$\frac{d}{d\,s_i}\left(\,\log(\hat{y}_k)\,\right) = \frac{d\log(\hat{y}_k)}{d\hat{y}_k}\frac{d\hat{y}_k}{d\,s_i} = \frac{1}{\hat{y}_k}\frac{d\hat{y}_k}{d\,s_i}$$

Άρα:

$$\frac{dJ}{d\,s_i} = -\sum_{k=1}^{K} y_k \frac{1}{\hat{y}_k}\frac{d\hat{y}_k}{d\,s_i} = -\sum_{k=1}^{K}\frac{y_k}{\hat{y}_k}\frac{d\hat{y}_k}{d\,s_i}$$

Για το $\frac{d\hat{y}_k}{d\,s_i}$ έχουμε καταλήξει σε δύο τύπους ανάλογα με το $\alpha\ \nu\ \boldsymbol{i = k}$ ή $\boldsymbol{i \neq k}$**.**

Έτσι σπάμε το άθροισμα ως εξής:

$$\frac{dJ}{d\,s_i} = -\left(\sum_{k=i}\frac{y_k}{\hat{y}_k}\frac{d\hat{y}_k}{d\,s_i} + \sum_{k\neq i}\frac{y_k}{\hat{y}_k}\frac{d\hat{y}_k}{d\,s_i}\right) = -\left(\frac{y_i}{\hat{y}_i}\frac{d\hat{y}_i}{d\,s_i} + \sum_{k\neq i}\frac{y_k}{\hat{y}_k}\frac{d\hat{y}_k}{d\,s_i}\right)$$

Με βάση τις σχέσεις (3) και (4), έχουμε:

$$\frac{dJ}{d\,s_i} = -\left(\frac{y_i}{\hat{y}_i}\,\hat{y}_i(1-\hat{y}_i) + \sum_{k\neq i}\frac{y_k}{\hat{y}_k}\left(-\hat{y}_k\,\hat{y}_i\right)\right)$$

$$= -\left(\,y_i(1-\hat{y}_i) + \sum_{k\neq i}(-y_k\,\hat{y}_i)\right) = -y_i(1-\hat{y}_i) + \sum_{k\neq i}\left(\,y_k\,\hat{y}_i\right)$$

$$= -y_i + y_i\hat{y}_i + \sum_{k\neq i}\left(\,y_k\,\hat{y}_i\right)$$

Παρατηρούμε ότι ο όρος $y_k\hat{y}_k$ είναι ο όρος του αθροίσματος $\sum_{k\neq i}\left(\,y_k\,\hat{y}_i\right)$ για $\boldsymbol{k = i}$ . Άρα η σχέση μπορεί να γραφεί ως εξής:

$$\frac{dJ}{d\,s_i} = -y_i + \sum_{k=1}^{K}\left(\,y_k\,\hat{y}_i\right) = -y_i + \hat{y}_i\sum_{k=1}^{K}(y_k)$$

Ως γνωστών το $y_k$ αντιστοιχεί την πιθανότητα η οντότητα να ανήκει στην κλάση i. Συνήθως το $y_k$ έχει την τιμή 0 ή 1 ανάλογα με το αν η οντότητα που αντιστοιχεί ανήκει στην κλάση i. Δηλαδή το $y$ είναι ένα one-hot vector. Σε κάθε περίπτωση το άθροισμα των $y_k$ (όπου $1 \leq \kappa \leq K$) είναι ίσο με 1. Δηλαδή $\sum_{k=1}^{K}(y_k) = 1.$

(Το παραπάνω αναφέρεται στις διαφάνειες του μαθήματος | κεφ. Regression σελίδα 151)

Αρά:

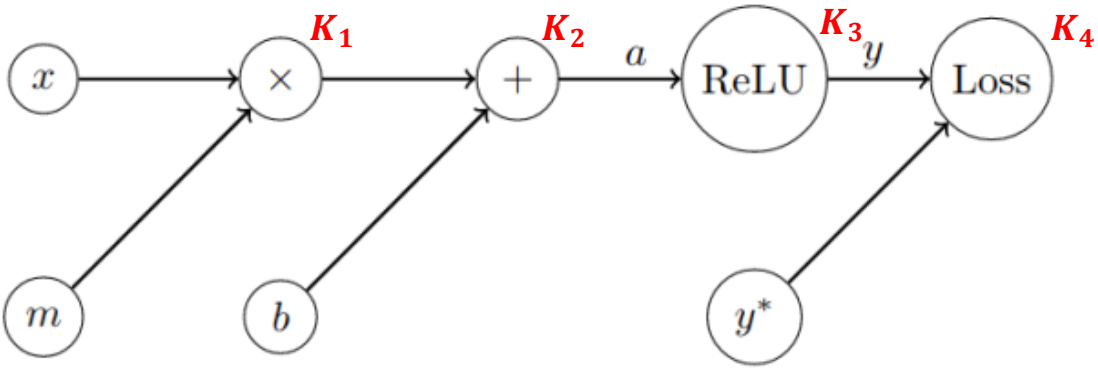
$$\frac{dJ}{d\,s_i} = -y_i + \hat{y}_i\sum_{k=1}^{K}(y_k) = -y_i + \hat{y}_i(1) = \hat{y}_i - y_i$$

Δηλαδή αποδείξαμε το ζητούμενο.

## Μέρος Β – Backpropagation

Θα εφαρμόσουμε backpropagation στο δοθέν computational graph έτσι έτσι ώστε να υπολογίσουμε όλα τα gradients εν συναρτήσει των εισόδων: $x, m, b, y^*$.

Αρχικά ονοματίζουμε όλους τους εσωτερικούς κόμβους ως εξής:




Στην συνέχεια υπολογίζουμε τα forward propagation steps για κάθε κόμβο:

$$K_1 = x * m$$
$$K_2 = K_1 + b = (x * m) + b$$
$$K_3 = RELU(K_2) = \max(0, K_2) = \max(0, (x * m) + b)$$
$$K_4 = MSE(K_3, y^*) = (K_3 - y^*)^2 = (\max(0, (x * m) + b) - y^*)^2$$

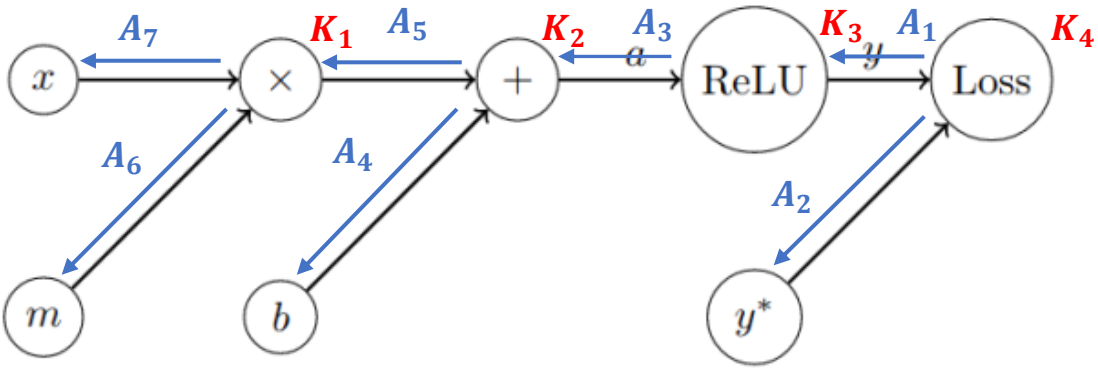
Με βάση τα παραπάνω forward steps υπολογίζουμε τα local gradients για κάθε κόμβο:

- Για τον $K_1$
  - $\frac{dK_1}{dx} = m$
  - $\frac{dK_1}{dm} = x$
- Για τον $K_2$
  - $\frac{dK_2}{dK_1} = 1$
  - $\frac{dK_2}{db} = 1$
- Για τον $K_3$
  - $\frac{dK_3}{dK_2} = \frac{d}{dK_2}\max(0, K_2) = \left(\frac{d}{dK_2}K_2\right)(K_2 > 0) = 1(K_2 > 0)$
- Για τον $K_4$
  - $\frac{dK_4}{dK_3} = 2(K_3 - y^*)\frac{d}{dK_3}(K_3 - y^*) = 2(K_3 - y^*)\left(\frac{d}{dK_3}K_3 - \frac{d}{dK_3}y^*\right)$
    $= 2(K_3 - y^*)(1 - 0) = 2(K_3 - y^*)$
  - $\frac{dK_4}{dy^*} = 2(K_3 - y^*)\frac{d}{dy^*}(K_3 - y^*) = 2(K_3 - y^*)\left(\frac{d}{dy^*}K_3 - \frac{d}{dy^*}y^*\right)$
    $= 2(K_3 - y^*)(0 - 1) = -2(K_3 - y^*)$

Έχοντας υπολογίσει για κάθε κόμβο το forward propagation step (upstream) αλλά και τα local gradients, μπορούμε πλέον να υπολογίσουμε το downstream για κάθε «ακμή». Το downstream υπολογίζεται μέσω του εξής τύπου:

**downstream = upstream * local_gradient**

Αρχικά ονοματίζω όλα τα downstream του backpropagation ως εξής:




Υπολογίζω τα downstreams με βάση τον τύπο που αναφέρθηκε παραπάνω.

Για τον κόμβο $K_4$ έχουμε ότι $upstream(K_4) = \frac{dK_4}{dK_4} = 1$.

$$A_1 = upstream(K_4) * \frac{dK_4}{dK_3} = 1 * 2(K_3 - y^*) = 2(K_3 - y^*)$$

$$A_2 = upstream(K_4) * \frac{dK_4}{dy^*} = 1 * \left(-2(K_3 - y^*)\right) = -2(K_3 - y^*)$$

$$A_3 = upstream(K_3) * \frac{dK_3}{dK_2} = A_2 * \left(1(K_2 > 0)\right) = \left(-2(K_3 - y^*)\right) * \left(1(K_2 > 0)\right)$$

$$A_4 = upstream(K_2) * \frac{dK_2}{db} = A_3 * 1 = A_3 = \left(-2(K_3 - y^*)\right) * \left(1(K_2 > 0)\right)$$

$$A_5 = upstream(K_2) * \frac{dK_2}{dK_1} = A_3 * 1 = A_3 = \left(-2(K_3 - y^*)\right) * \left(1(K_2 > 0)\right)$$

$$A_6 = upstream(K_1) * \frac{dK_1}{dm} = A_5 * x = \left(-2(K_3 - y^*)\right) * \left(1(K_2 > 0)\right) * x$$

$$A_7 = upstream(K_1) * \frac{dK_1}{dx} = A_5 * m = \left(-2(K_3 - y^*)\right) * \left(1(K_2 > 0)\right) * m$$

Άρα, έχουμε βρει τα downstreams για όλους τους input κόμβους. Πιο συγκεκριμένα έχουμε:

- Για το $x$ : $\frac{dK_4}{dx} = A_7 = \left(-2(K_3 - y^*)\right) * \left(1(K_2 > 0)\right) * m$
- Για το $m$ : $\frac{dK_4}{dm} = A_6 = \left(-2(K_3 - y^*)\right) * \left(1(K_2 > 0)\right) * x$
- Για το $b$ : $\frac{dK_4}{dm} = A_4 = \left(-2(K_3 - y^*)\right) * (K_2 > 0)$
- Για το $y^*$ : $\frac{dK_4}{dy^*} = A_2 = -2(K_3 - y^*)$

Στις παραπάνω σχέσεις αντικαθιστώ τα $K_3$ και $K_2$ με τους αντίστοιχους τύπους που υπολόγισα παραπάνω εν συναρτήσει των inputs. Δηλαδή

$$K_2 = K_1 + b = (x * m) + b$$

$$K_3 = RELU(K_2) = \max(0, K_2) = \max(0, (x * m) + b)$$

Έτσι έχουμε:

- Για το $x$ : $\frac{dK_4}{dx} = \left(-2(\max(0, (x * m) + b) - y^*)\right) * \left((x * m) + b > 0\right) * m$
- Για το $m$ : $\frac{dK_4}{dm} = \left(-2(\max(0, (x * m) + b) - y^*)\right) * \left((x * m) + b > 0\right) * x$
- Για το $b$ : $\frac{dK_4}{db} = \left(-2(\max(0, (x * m) + b) - y^*)\right) * \left((x * m) + b > 0\right)$
- Για το $y^*$ : $\frac{dK_4}{dy^*} = -2(\max(0, (x * m) + b) - y^*)$

Σαν επαλήθευση, αλλά και σαν δεύτερη μεθοδολογία, ξεκινώντας από τους τύπους που υπολογίστηκαν για τα forward steps των $K_1, K_2, K_3, K_4$ υπολογίζω τις μερικές παραγώγους ως προς τις μεταβλητές εισόδου $x, m, b, y^*$ έως ότου υπολογίσω τις $\frac{dK_4}{dx}, \frac{dK_4}{dm}, \frac{dK_4}{db}, \frac{dK_4}{dy^*}$ οι οποίες αντιστοιχούν στα gradients τα οποία καταλήγουν στους κόμβους εισόδου.

- Για τον $K_1$
  - $\frac{dK_1}{dx} = m$
  - $\frac{dK_1}{dm} = x$
- Για τον $K_2$
  - $\frac{dK_2}{dx} = m$
  - $\frac{dK_2}{dm} = x$
  - $\frac{dK_2}{db} = 1$
- Για τον $K_3$
  - $\frac{dK_3}{dx} = \frac{d}{dx}\max(0, K_2) = \left(\frac{d}{dx}K_2\right)(K_2 > 0) = m\big((x * m) + b > 0\big)$
  - $\frac{dK_3}{dm} = \frac{d}{dm}\max(0, K_2) = \left(\frac{d}{dm}K_2\right)(K_2 > 0) = x\big((x * m) + b > 0\big)$
  - $\frac{dK_3}{db} = \frac{d}{db}\max(0, K_2) = \left(\frac{d}{db}K_2\right)(K_2 > 0) = 1\big((x * m) + b > 0\big)$
- Για τον $K_4$
  - $\frac{dK_4}{dx} = 2(K_3 - y^*)\frac{d}{dx}(K_3 - y^*) = 2(K_3 - y^*)\left(\frac{d}{dx}K_3 - 0\right)$
    $= 2(\max(0, (x * m) + b) - y^*)(m\big((x * m) + b > 0\big))$
  - $\frac{dK_4}{dm} = 2(K_3 - y^*)\frac{d}{dm}(K_3 - y^*) = 2(K_3 - y^*)\left(\frac{d}{dm}K_3 - 0\right)$
    $= 2(\max(0, (x * m) + b) - y^*)(x\big((x * m) + b > 0\big))$
  - $\frac{dK_4}{db} = 2(K_3 - y^*)\frac{d}{db}(K_3 - y^*) = 2(K_3 - y^*)\left(\frac{d}{db}K_3 - 0\right)$
    $= 2(\max(0, (x * m) + b) - y^*)(1\big((x * m) + b > 0\big))$
  - $\frac{dK_4}{dy^*} = 2(K_3 - y^*)\frac{d}{dy^*}(K_3 - y^*) = 2(K_3 - y^*)\left(\frac{d}{dy^*}K_3 - \frac{d}{dy^*}y^*\right)$
    $= 2(K_3 - y^*)(0 - 1) = -2((\max(0, (x * m) + b)) - y^*)$

Συγκρίνοντας τις μερικές παραγώγους του $K_4$ που προέκυψαν με αυτές της πρώτης «μεθόδου» παρατηρούμε πως είναι ίδιες.

Έτσι συμπεραίνουμε πως έχουν υπολογιστεί σωστά όλα τα gradients του computational graph σε κάθε κόμβο εν συναρτήσει των μεταβλητών εισόδου.
Δηλαδή, δοθέντος ενός input $x, m, b, y^*$ μπορούμε με τους παραπάνω τύπους να υπολογίσουμε όλα τα gradients.

# Μέρος Γ – Classification

Στο τρίτο μέρος της εργασίας θα κατασκευάσω έναν sentiment classifier χρησιμοποιώντας feed-forward νευρωνικά δίκτυα για την κατηγοριοποίηση των tweets των datasets που μας δόθηκαν σε τρεις κλάσεις (neutral, anti-vax and pro-vax).
Πιο συγκεκριμένα θα έπειτα από τον αντίστοιχο πειραματισμό θα κατασκευάσω δύο διαφορετικά μοντέλα classifier, θα αξιολογήσω το κάθε ένα χρησιμοποιώντας τις μετρικές που ζητούνται, θα συγκρίνω τα δύο μοντέλα μεταξύ τους και τέλος θα τα συγκρίνω με το μοντέλο της πρώτης εργασίας.

## Γενικά Σχόλια

- Στο παραδοτέο αρχείο (.ipynb) υπάρχει η υλοποίηση των δύο μοντέλων sentiment classifier που κατασκεύασα.
- Η υλοποίηση των νευρωνικών δικτύων των μοντέλων έγινε αποκλειστικά με την χρήση του pytorch.
- **ΣΗΜΑΝΤΙΚΟ**: Για την δοκιμή του μοντέλου στο train set χρειάζεται να αλλάξει το path που ορίζεται από την εντολή:
  ```
  validation_set_location = r'vaccine_validation_set.csv'
  ```
  Η εντολή αυτή βρίσκεται στην αρχή του κάθε cell.
  Έτσι θα αντικατασταθεί το validation set με το train set και κατ' επέκταση ο classifier θα δοκιμαστεί και θα αξιολογηθεί για το test set.
- Στις παραγράφους που ακολουθούν θα περιγράψω αναλυτικά την διαδικασία/μελέτη που ακολούθησα έτσι ώστε να καταλήξω στην τελική μου υλοποίηση.

## Μεθοδολογία

Η μεθοδολογία και τα βήματα που ακολούθησα για την παραγωγή του classifier είναι:

- Ανάγνωση των αρχείων train και validation και αποθήκευση των δεδομένων τους σε δυο dataframes (trainSet και validationSet αντίστοιχα).
- Προεπεξεργασία των δεδομένων τόσο του train set όσο και του validation set. "Καθαρίζουμε" τα δεδομένα έτσι ώστε κάθε tweet να περιέχει μόνο στοιχεία τα οποία μπορούν να προσφέρουν κάποια χρήσιμη πληροφορία.
- Δημιουργία μιας αναπαράστασης για κάθε tweet. Για την αναπαράσταση, στο ένα μοντέλο χρησιμοποιούνται GloVe pre-trained words embedding vectors, ενώ στο δεύτερο μοντέλο χρησιμοποιείται ο vectorizer TF-IDF. Λεπτομέρειες για την δημιουργία των αναπαραστάσεων εξηγούνται στις αντίστοιχες παραγράφους των δύο μοντέλων.
- Ορισμός του νευρωνικού δικτύου, δηλαδή του αριθμού των layers που θα περιέχει, του αριθμού των νευρώνων από τους οποίος θα αποτελείται το κάθε layer, τα activation functions καθώς και τεχνικές regularization που εφαρμόζονται μεταξύ των layers (π.χ. dropout).
- Ορισμός του loss function που θα χρησιμοποιηθεί καθώς και του optimizer μαζί με τις αντίστοιχες παραμέτρους του.
- Εκπαίδευση του μοντέλου για προκαθορισμένο αριθμό epochs. Σε κάθε epoch «δοκιμάζουμε» το μοντέλο μας και για το validation set υπολογίζοντας το validation loss, validation accuracy και validation f1-score τα οποία και εκτυπώνονται για κάθε ένα epoch μαζί με το train loss.

- Δοκιμή του εκπαιδευμένου μοντέλου στο validation set και αξιολόγηση του classification με βάση τις προβλέψεις που έκανε το μοντέλο μας.
  Για την αξιολόγηση χρησιμοποιείται: f1 score, accuracy, precision και recall.
  Επίσης κατασκευάζονται και παρουσιάζονται τα loss vs epoch curves τόσο για το test set όσο και για το validation καθώς και τα ROC curves για κάθε μία κλάση έτσι ώστε να παρακολουθήσουμε την συμπεριφορά του μοντέλου μας κατά την εκπαίδευση και να ανιχνεύσουμε φαινόμενα όπως overfit ή underfit.

## Προεπεξεργασία Δεδομένων

Μέσω της προεπεξεργασίας των δεδομένων κειμένου (στην περίπτωση μας tweets) καταφέρνουμε να απαλείψουμε λέξεις/στοιχεία τα οποία δεν προσφέρουν κάποια χρήσιμη πληροφορία.
Έτσι με την κατάλληλη προεπεξεργασία, προκύπτει μια καλύτερη αναπαράσταση από τον vectorizer η οποία δεν περιέχει άχρηστες πληροφορίες.
Πιο συγκεκριμένα οι τεχνικές που δοκίμασα είναι οι εξής:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions (αναφορά σε άλλους χρήστες του twitter), αφαίρεση των αριθμών και αφαίρεση των hashtags. Για τα παραπάνω χρησιμοποίησα τον tweet-preprocessor ο οποίος έχει κατασκευαστεί για αυτόν ακριβώς τον σκοπό.
- Αφαίρεση των emojis καθώς και αντικατάσταση τους από raw text (π.χ. 😬 → grinning_face)
- Αφαίρεση των συμβόλων, σημείων στίξης.
- Αφαίρεση των stop_words.
- Lemmatization μέσω του WordNetLemmatizer
- Stemming μέσω του SnowballStemmer

Δοκίμασα όλες τις παραπάνω τεχνικές σε διαφορετικούς συνδιασμούς. Οι περισσότερες αποδείχθηκαν αποδοτικές σε συνδυασμό με το νευρωνικό δίκτυο.
Στην μελέτη/σύγκριση που ακολουθεί παρουσιάζεται αναλυτικά η απόδοση της κάθε τεχνικής καθώς και το ποιες από αυτές επέλεξα να εφαρμόσω τελικά.

## Δημιουργία αναπαραστάσεων των tweets

### Pre-trained word embedding vectors (GloVe)

Για κάθε ένα tweet πρέπει να δημιουργήσουμε μια αναπαράσταση από αριθμούς η οποία στην συνέχεια θα δοθεί σαν είσοδος στο νευρωνικό δίκτυο.
Στο ένα μοντέλο για την αναπαράσταση αυτή χρησιμοποίησα pre-trained word embedding vectors του Glove, όπως άλλωστε υποδεικνύεται και από την εκφώνηση της εργασίας.

Δοκίμασα όλα τα παρεχόμενα αρχεία, συγκρίνοντας την απόδοση του καθενός έτσι ώστε να διαπιστώσω ποιο λειτουργεί καλύτερα για το δοθέν dataset.
Παρατήρησα καλύτερη απόδοση με τα αρχεία που παράγουν πολλά features για κάθε μια λέξη καθώς αναπαριστούν με μεγαλύτερη ακρίβεια την κάθε λέξη και κατ' επέκταση ολόκληρο το tweet.
Τα δύο αρχεία με τα οποία παρατήρησα την καλύτερη απόδοση ήταν:

- glove.840B.300d
- glove.twitter.27B.200d

Τελικά επέλεξα το **glove.twitter.27B.200d** καθώς τα αποτελέσματα που προέκυπταν ήταν καλύτερα. Αυτό οφείλεται στο γεγονός ότι τα δεδομένα με τα οποία έχει εκπαιδευτεί είναι πολύ πιο σχετικά με αυτά του dataset, επομένως οι αναπαραστάσεις είναι πιο ακριβείς.

Η διαδικασία που ακολουθείται για την δημιουργία των αναπαραστάσεων για κάθε ένα tweet είναι η εξής:

- Αρχικά κατεβάζουμε το zip με τα αρχεία που επιθυμούμε, το κάνουμε unzip και στην συνέχεια επιλέγουμε το αρχείο με την διάσταση που θέλουμε. Αυτό γίνεται με τις εξής εντολές:
  **!wget https://nlp.stanford.edu/data/glove.twitter.27B.zip**
  **!unzip glove.twitter.27B.zip**
  **glove_file = 'glove.twitter.27B.200d.txt'**
- Στην συνέχεια διαβάζουμε το αρχείο και κατασκευάζουμε ένα dictionary (embedding_dict) το οποίο περιέχει την αναπαράσταση της κάθε λέξης. Πιο συγκεκριμένα, για κάθε γραμμή του αρχείου, η πρώτη λέξη αντιστοιχεί στο κλειδί ενώ οι υπόλοιπες λέξεις/αριθμοί αντιστοιχούν στο vector που αναπαριστά την λέξη. Έτσι για κάθε γραμμή εισάγουμε στο dictionary την λέξη σαν κλειδί και το vector σαν την τιμή του κλειδιού. Τελικά, προκύπτει ένα dictionary που περιέχει όλες τις λέξεις σαν κλειδιά και τις αντίστοιχες αναπαραστάσεις σαν τιμές των αντίστοιχων κλειδιών. Η διαδικασία αυτή γίνεται από την συνάρτηση **construct_embedding_dict.**
- Στην συνέχεια πρέπει να κατασκευάσουμε μια αναπαράσταση για κάθε ένα tweet χρησιμοποιώντας το dictionary που δημιουργήσαμε.
  Για κάθε ένα tweet:
  - Χωρίζουμε τα tweets σε tokens. Δηλαδή ένα tweet πλέον είναι μια λίστα από strings/λέξεις αντί να είναι ένα ενιαίο string.
  - Για κάθε μια λέξη βρίσκουμε το vector που την αναπαριστά μέσω του embedding_dict. Όλα τα embedding vectors είναι ίδιας διάστασης, στην συγκεκριμένη περίπτωση 200.

- Αθροίζουμε τα vectors κάθε λέξης/token του tweet και τελικά υπολογίζουμε ένα το μέσο διάνυσμα (average) το οποίο και αναπαριστά το tweet.
  Η επιλογή να υπολογίζεται το μέσο διάνυσμα έγινε έτσι ώστε να προκύπτει ίδιο μεγέθους αναπαράσταση για κάθε tweet ανεξαρτήτως των λέξεων που περιέχει. Έτσι, αθροίζοντας τα vectors κάθε λέξης και υπολογίζοντας τον μέσο όρο, προκύπτει για κάθε tweet ένα vector μεγέθους 200.

Η διαδικασία αυτή εφαρμόζεται τόσο στο train set όσο και στο validation set καλώντας την συνάρτηση **findTextsFeatureVectors** η οποία με την σειρά της καλεί την συνάρτηση **findAverageVector** για κάθε ένα tweet.

### Vectorizers

Στο δεύτερο μοντέλο για την δημιουργία των αναπαραστάσεων των tweets χρησιμοποίησα vectorizer.
Όπως και στην πρώτη εργασία, οι vectorizers με τους οποίους πειραματίστηκα είναι:

- **Count Vectorizer**
- **TF-IDF Vectorizer**
- **Hashing Vectorizer**

Κάθε ένας από αυτούς χρησιμοποιεί κάποιες παραμέτρους οι οποίες ορίζονται κατά την αρχικοποίηση του.
Ανάλογα με την τιμή της κάθε παραμέτρου, ο vectorizer έχει διαφορετική συμπεριφορά και παράγει τα αντίστοιχα αποτελέσματα.
Οι παράμετροι με τις οποίες πειραματίστηκα είναι οι εξής:

- **min_df** και **max_df** για τους Count και TF-IDF Vectorizers. Μέσω των παραμέτρων αυτών ο vectorizer αγνοεί features τα οποία εμφανίζονται πολύ συχνά σε tweets και πολύ σπάνια αντίστοιχα. Ανάλογα με την τιμή των δύο παραμέτρων, ορίζεται η συχνότητα με βάση την οποία αγνοούμε ένα feature.
- **max_features** για τους Count και TF-IDF Vectorizers. Μέσω της παραμέτρου αυτής ορίζεται ο μέγιστος αριθμός των features που θα περιέχει κάθε αναπαράσταση. Πιο συγκεκριμένα, ο vectorizer κρατάει τα max_features με την μεγαλύτερη συχνότητα.
- **n_features** για τον Hashing Vectorizer. Μέσω της παραμέτρου αυτής ορίζεται ο αριθμός των features που θα περιέχει κάθε αναπαράσταση.
- **ngram_range** , μέσω της παραμέτρου αυτής ορίζεται ο αριθμός των λέξεων από τον οποίο μπορεί να αποτελείται ένα feature (π.χ. unigrams, bigrams).

Έτσι, δοκίμασα πολλούς συνδυασμούς παραμέτρων για κάθε μια μέθοδο για να εντοπίσω τον πιο αποδοτικό συνδυασμό.
Τα αποτελέσματα της μελέτης παρουσιάζονται στην συνέχεια.

#### *Μείωση των features*

Ως γνωστόν, για ένα μικρό dataset ένας μεγάλος αριθμός από features σε κάθε αναπαράσταση μπορεί να αποβεί μη αποδοτικός κάτι που αποδείχθηκε και κατά τον πειραματισμό μου καθώς δίχως τον περιορισμό των features το νευρωνικό δίκτυο δεν αποδίδει καθόλου καλά.
Όπως αναφέρθηκε παραπάνω ο αριθμός των features μπορεί να περιοριστεί μέσω των παραμέτρων των vectorizers.
Επέλεξα να περιορίσω τον αριθμό των feature μέσω αυτών των παραμέτρων και όχι decomposition όπως στην πρώτη εργασία καθώς το decomposion δεν βελτίωνε την απόδοση του μοντέλου ενώ ταυτόχρονα αυξάνει σημαντικά τον χρόνο εκτέλεσης.

## Feed Forward Neural Network

Για την υλοποίηση του νευρωνικού δικτύου χρησιμοποίησα την βιβλιοθήκη pytorch.

Αρχικά ορίζω την κλάση Net η οποία υλοποιεί το NN. Η κλάση αυτή είναι υποκλάση της nn.Module (όπου nn: torch.nn).
Μέσω της __init__() ορίζω:

- Τα layers του δικτύου (input,hidden,output) καθώς και τις διαστάσεις του καθενός.
- Τα activation functions των layers.
- Καθώς και τεχνικές regularization που εφαρμόζονται μεταξύ των layers (π.χ. dropout, batch normalization, flatten).

Μέσω του sequential container (nn.Sequential) τον οποίο χρησιμοποιώ κατά τον ορισμό του NN, τα layers / activation functions / regularizations εφαρμόζονται διαδοχικά με την σειρά που έχουν οριστεί.

Αφού οριστεί το NN, ορίζω:

- Το loss function που θα χρησιμοποιηθεί κατά την εκπαίδευση
- Ο optimizer που θα χρησιμοποιηθεί καθώς και οι αντίστοιχες παραμέτροι του
    - learning_rate
    - momentum
    - weigh_decay

Στην συνέχεια ορίζω τους data loaders τόσο για το train set όσο και για το validation set.
Πιο συγκεκριμένα:

- Μετατρέπουμε τα δεδομένα μας σε tensors (torch.tensor). Τόσο τα feature vectors όσο των tweets και τα αντίστοιχα labels.
- Δημιουργούμε ένα TensorDataset για το train set και ένα για το validation set. Κάθε ένα TensorDataset δημιουργείται από τα tensors των feature vectors και των αντίστοιχων labels.
- Μέσω των TensorDatasets δημιουργούμε τους DataLoaders ορίζοντας κατάλληλα το επιθυμητό batch size.

Τέλος εκπαιδεύουμε το NN μέσω της συνάρτησης **train_model**.
Δίνοντας σαν όρισμα τον επιθυμητό αριθμό epochs, η εκπαίδευση γίνεται επαναληπτικά για κάθε ένα epoch ως εξής:

- Για κάθε ένα batch του train set:
    - Διαγράφουμε τα αποθηκευμένα gradients από το προηγούμενο epoch.
    - Εκτελούμε το μοντέλο με το τρέχον batch, παίρνοντας το αντίστοιχο output για κάθε ένα tweet.
    - Υπολογίζουμε το loss μέσω της loss function που έχει οριστεί.
    - Εφαρμόζεται backpropagation με το loss που υπολογίστηκε.
    - Μέσω του optimizer ανανεώνουμε τα weights του μοντέλου με βάση τα gradients που προέκυψαν από το backpropagation
    - Τέλος, αποθηκεύουμε τα predictions που προέκυψαν έτσι ώστε να αξιολογήσουμε το μοντέλο στο τέλος του epoch

- Αφού ολοκληρωθεί το training για αυτό το epoch αξιολογούμε το μοντέλο χρησιμοποιώντας το validation set. Αρχικά καλούμε την συνάρτηση model.eval() έτσι ώστε να θέσουμε το μοντέλο σε “evaluation mode”. Στην συνέχεια για κάθε ένα batch του validation set:
    - Εκτελούμε το μοντέλο με το τρέχον batch, παίρνοντας το αντίστοιχο output για κάθε ένα tweet.
    - Υπολογίζουμε το loss μέσω της loss function που έχει οριστεί.
    - Αποθηκεύουμε τα predictions που προέκυψαν έτσι ώστε να αξιολογήσουμε το μοντέλο αφού ολοκληρωθεί η διαδικασία για όλα τα batches.
- Τέλος, υπολογίζουμε και εκτυπώνουμε για το τρέχον epoch τα εξής:
    - Train Loss
    - Validation Loss
    - Train F1-Score
    - Validation F1-Score
    - Validation Accuracy

Αφού ολοκληρωθεί η παραπάνω διαδικασία για όλα τα epochs, το μοντέλο έχει εκπαιδευτεί και βρίσκεται σε evaluation mode έτσι ώστε να γίνει το τελικό evaluation του.

### Σημαντική Παρατήρηση

Το Loss Function που θα χρησιμοποιήσω είναι το nn.CrossEntropyLoss() καθώς το πρόβλημα μας είναι ένα multiclass classification.
Όπως αναφέρεται στο documentation της nn.CrossEntropyLoss(), η συνάρτηση αυτή συνδιάζει τα nn.LogSoftmax() and nn.NLLLoss().
Αυτό σημαίνει ότι έπειτα από το output layer δεν χρειάζεται να εφαρμοστεί Softmax καθώς αυτό εφαρμόζεται εσωτερικά την nn.CrossEntropyLoss().
Πηγές:

- Documentation nn.CrossEntropyLoss()
- https://stackoverflow.com/questions/55675345/should-i-use-softmax-as-output-when-using-cross-entropy-loss-in-pytorch
- https://programmerall.com/article/5058212346/
  « So when we use PyTorch to build a classification network, there is no need to manually add a softmax layer after the last fc layer.»

## Αξιολόγηση του μοντέλου

Έπειτα από την εκπαίδευση του μοντέλου (και εφόσον το μοντέλο βρίσκεται ήδη σε evaluation mode) εκτελούμε το μοντέλο δίνοντας του σαν input το validation set. Αυτό μας επιστρέφει τις προβλέψεις του για κάθε ένα tweet (pred). Συγκεκριμένα, για κάθε ένα tweet μας επιστρέφει τρείς τιμές οι οποίες αντιστοιχούν στην πιθανότητα να ανήκει το tweet στην αντίστοιχη κλάση.

Για την αξιολόγηση της κατηγοριοποίησης που προέκυψε από το μοντέλο χρησιμοποιώ τις εξής μετρικές:

- F1 score
- Precision
- Recall

Επίσης κατά την εκπαίδευση του μοντέλου υπολογίζω και παρουσιάζω τις καμπύλες Loss vs Epochs τόσο για το train set όσο και για το validation. Πιο συγκεκριμένα, όπως αναφέρθηκε την παράγραφο Feed Forward Neural Network, για κάθε ένα epoch υπολογίζεται το loss για το train set και για το validation set.
Μετά το πέρας της εκπαίδευσης σχεδιάζονται οι δύο καμπύλες.

Τέλος, παρουσιάζω τα roc curves για κάθε μία από τις τρείς κλάσεις. Πιο συγκεκριμένα, μέσω των προβλέψεων του μοντέλου (y_pred) υπολογίζω τα roc curves για κάθε μία κλάση μέσω της συνάρτησης roc_curve της sklearn και στην συνέχεια παρουσιάζω τις τρείς καμπύλες στο ίδιο διάγραμμα.

Μέσω των μετρικών αλλά και των καμπυλών μπορούμε να μελετήσουμε αναλυτικά την απόδοση του μοντέλου αλλά και να εντοπίσουμε προβλήματα όπως overfit και underfit.

Για περαιτέρω μελέτη της συμπεριφοράς/απόδοσης του μοντέλου χρησιμοποιώ:

- Accuracy metric
- Κατασκευάζω το confusion matrix
- Χρησιμοποιώ την συνάρτηση classification_report της sklearn, η οποία παρουσιάζει τα precision, recall και f1-score για κάθε ένα label ξεχωριστά.

## Μοντέλο 1 - Word Embeddings / Feed-Forward NN

### Προεπεξεργασία Δεδομένων

Αρχικά επιλέγω να εφαρμόσω την ίδια προεπεξεργασία η οποία είχε αποδειχθεί βέλτιστη στην πρώτη εργασία. Δηλαδή εφαρμόζω:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions και των αριθμών καθώς δεν προσφέρουν καμία πληροφορία σχετικά με το “νόημα” της πρότασης.
- Αντικατάσταση των emojis σε raw text (π.χ. 😀 → grinning_face) καθώς πολλές φορές αποτυπώνουν πληροφορία σχετικά με το συναίσθημα.
- Αφαίρεση των συμβόλων, σημείων στίξης.

### Μελέτη για την εύρεση του καλύτερου μοντέλου NN

Αρχικά επιλέγω να υλοποιήσω ένα πολύ απλό δίκτυο το οποίο μειώνει σταδιακά τα features μέσω των layers, χωρίς activation functions ή τεχνικές regularization.
Στο input, πριν δοθεί στο input layer εφαρμόζω την συνάρτηση nn.Flatten.
Πιο συγκεκριμένα το αρχικό μοντέλο είναι το εξής:

```
Net(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (model_stack): Sequential(
    (0): Linear(in_features=200, out_features=128, bias=True)
    (1): Linear(in_features=128, out_features=64, bias=True)
    (2): Linear(in_features=64, out_features=32, bias=True)
    (3): Linear(in_features=32, out_features=16, bias=True)
    (4): Linear(in_features=16, out_features=3, bias=True)
  )
)
```

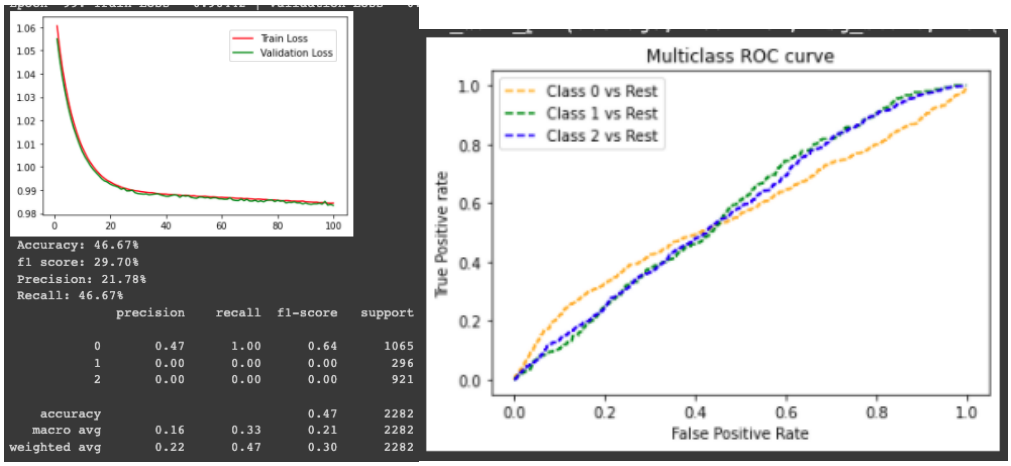
Ως loss function ορίζω το CrossEntropy καθώς έχουμε πρόβλημα multiclass classification.
Ως optimizer ορίζω τον SGD χωρίς να ορίσω κάποια τιμή για τις παραμέτρους momentum και weight_decay.
Ορίζω το learning rate = 0.0001

Καθ’ όλη την μελέτη επιλέγω να εκπαιδεύω το μοντέλο με 100 epochs.

Έχοντας αυτό το βασικό νευρωνικό δίκτυο αρχικά πειραματίζομαι με το batch size των dataloaders. Δοκιμάζοντας τις τιμές : 8,16,32,64 κατέληξα στην τιμή 16 με την οποία το μοντέλο παρουσίαζε την καλύτερη απόδοση.

Εκπαιδεύοντας το μοντέλο παρατηρούμε πως αδυνατεί να εκπαιδευτεί. Συγκεκριμένα, τα αποτελέσματα είναι τα εξής:
(Τα αποτελέσματα για κάθε epoch παρουσιάζονται αποσπασματικά για λίγα epochs. Στο .ipynb εκτυπώνονται για όλα τα epochs)

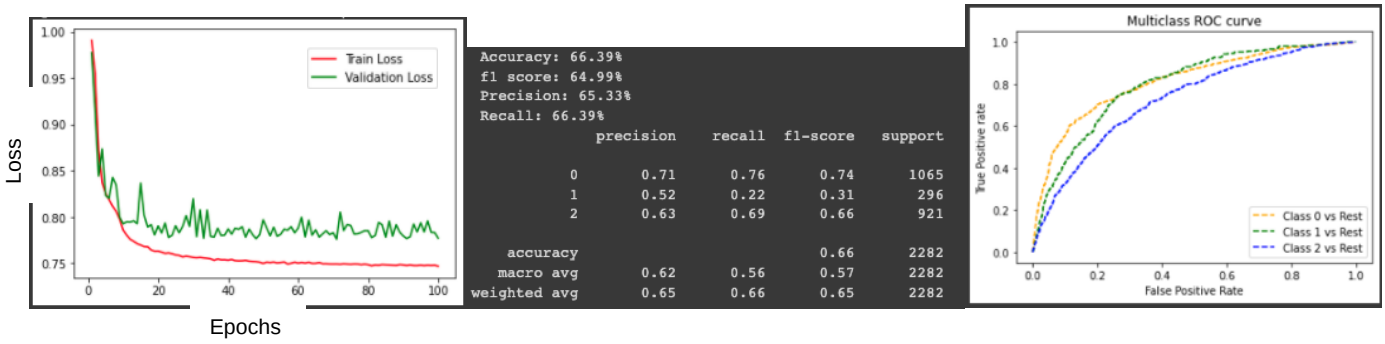
| Epoch | Train Loss | Validation Loss | Train F1-Score | Validation F1-Score | Validation Accuracy |
|---|---|---|---|---|---|
| 1 | 1.06060 | 1.05502 | 29.71 | 29.70 | 46.6696 |
| 25 | 0.99054 | 0.98965 | 29.71 | 29.70 | 46.6696 |
| 50 | 0.98736 | 0.98645 | 29.71 | 29.70 | 46.6696 |
| 100 | 0.98442 | 0.98337 | 29.71 | 29.70 | 46.6696 |




Παρατηρούμε πώς:

- Τόσο το train loss όσο και το validation loss μειώνονται ελάχιστα και παραμένουν πολύ υψηλά.
- Το μοντέλο αδυνατεί πλήρως να αναγνωρίσει τις κλάσεις 1 και 2.
- Τα ROC curves είναι πολύ κοντά στην ευθεία x=y η οποία αντιστοιχεί στο ROC curve του random classifier κάτι που επαληθεύει την κακή απόδοση του μοντέλου.

Με βάση τα παραπάνω συμπεράσματα δοκιμάζω να αυξήσω σταδιακά το learning rate. Θέτοντας το **learning rate** ίσο **με 0.01** έχουμε τα εξής αποτελέσματα:




Παρατηρούμε πώς:

- Έχει βελτιωθεί κατά πολύ η απόδοση του μοντέλου τόσο για το train set όσο και για το validation set και έχουν μειωθεί κατά πολύ τα αντίστοιχα losses.
- Το μοντέλο πλέον αναγνωρίζει σε ικανοποιητικό βαθμό όλες τις κλάσεις
- Τα ROC curves έχουν κάνει μια κλίση προς τα πάνω έχοντας αρκετά υψηλότερες τιμές για το True Positive rate σε σχέση με την ευθεία x=y. Αυτό επαληθεύει την πολύ καλύτερη απόδοση του μοντέλου μας και για τις τρείς κλάσεις.

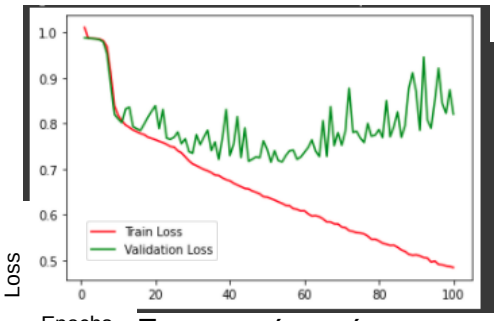
Ωστόσο από τα loss curves παρατηρούμε μια μικρή αστάθεια στο validation loss καθώς και μια διαφορά μεταξύ του train και validation loss. Δηλαδή το μοντέλο μας παρουσιάζει overfit.

Στην συνέχεια θα προσθέσουμε τα activation functions ενδιάμεσα από τα hidden layers.
Δοκίμασα αρκετά activation functions όπως Sigmoid, ReLU, RReLU, SELU, ELU.
Σε αντίθεση με την θεωρία, καλύτερα αποτελέσματα παρατήρησα με την ReLU.

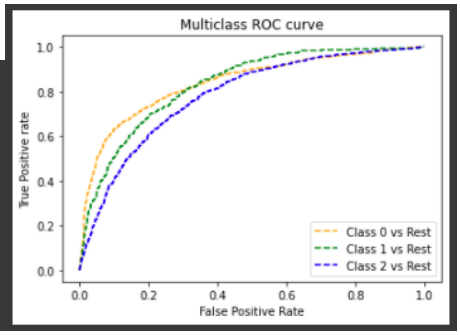
Εφαρμόζοντας τα activation functions το νέο μοντέλο είναι το εξής:

```
Net(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (model_stack): Sequential(
    (0): Linear(in_features=200, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=32, bias=True)
    (5): ReLU()
    (6): Linear(in_features=32, out_features=16, bias=True)
    (7): ReLU()
    (8): Linear(in_features=16, out_features=3, bias=True)
  )
)
```

Έπειτα από την εκπαίδευση του παραπάνω μοντέλου, παίρνουμε τα παρακάτω αποτελέσματα:




```
Accuracy: 68.32%
f1 score: 68.18%
Precision: 68.22%
Recall: 68.32%
              precision    recall  f1-score   support

           0       0.73      0.78      0.76      1065
           1       0.47      0.48      0.48       296
           2       0.69      0.63      0.66       921

    accuracy                           0.68      2282
   macro avg       0.63      0.63      0.63      2282
weighted avg       0.68      0.68      0.68      2282
```



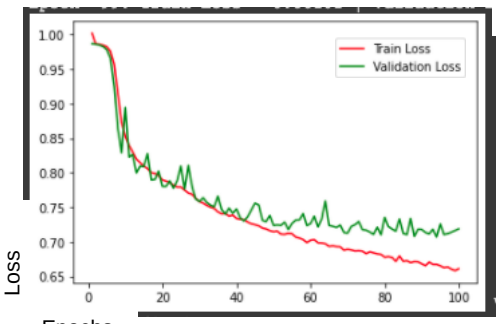

Παρατηρούμε πώς:

- Η απόδοση του μοντέλου για το train set έχει αυξηθεί σημαντικά καθώς το train loss έχει μειωθεί κατά πολύ, ενώ τα train scores που τυπώνονται ανά epoch έχουν αυξηθεί αρκετά (φτάνουν το 80%).
- Η απόδοση του μοντέλου αυξήθηκε και για το validation set, κάτι που επαληθεύεται και από τα ROC curves. Ωστόσο υπάρχει μεγάλη διαφορά μεταξύ train και validation set τόσο στο loss όσο και στα scores. Δηλαδή έχει αυξηθεί κατά πολύ το overfit.
- Επίσης παρατηρούμε μεγάλη αστάθεια όσον αφορά το validation loss.

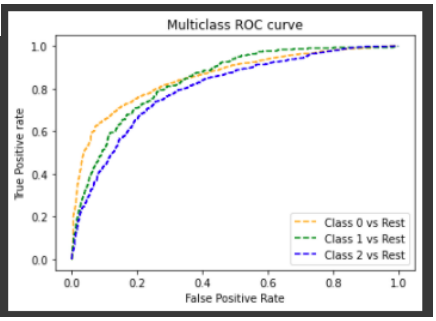
Αρχικά θα προσπαθήσουμε να εξαλείψουμε το φαινόμενο του overfit.
Έτσι θα προσθέσουμε **dropout** μεταξύ των πρώτων layers του δικτύου μας.
Δοκιμάζοντας διάφορους συνδυασμούς τιμών για τα dropouts κατέληξα στον εξής συνδυασμό:

```
Net(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (model_stack): Sequential(
    (0): Linear(in_features=200, out_features=128, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.25, inplace=False)
    (6): Linear(in_features=64, out_features=32, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=32, out_features=16, bias=True)
    (10): ReLU()
    (11): Linear(in_features=16, out_features=3, bias=True)
  )
)
```

Τα αποτελέσματα του παραπάνω μοντέλου είναι τα εξής:




```
Accuracy: 68.71%
f1 score: 68.00%
Precision: 67.78%
Recall: 68.71%
              precision    recall  f1-score   support

           0       0.73      0.80      0.76      1065
           1       0.50      0.35      0.41       296
           2       0.68      0.67      0.67       921

    accuracy                           0.69      2282
   macro avg       0.63      0.60      0.61      2282
weighted avg       0.68      0.69      0.68      2282
```




Παρατηρούμε πώς:

- Πλέον η διαφορές μεταξύ των train και validation losses/scores έχουν μειωθεί κατά πολύ.
- Επίσης το validation loss curve είναι πολύ πιο ομαλό.
- Ταυτόχρονα η απόδοση του μοντέλου αυξήθηκε ελάχιστα για το validation set.
- Δηλαδή, έχει σχεδόν εξαλειφθεί το overfit.

Στην συνέχεια, έτσι ώστε να εξαλειφθεί τελείως το overfit του μοντέλου και το μοντέλο μας να έχει μια πιο ομαλή συμπεριφορά αρχικά δοκίμασα:

- Να εφαρμόσω λιγότερη/περισσότερη προεπεξεργασία στα δεδομένα (π.χ. αφαίρεση stopwords, lemmatization, stemming)
- Batch normalization
- Διαφορετικά activation functions αντί του ReLU
- Πειραματισμό με διάφορες τιμές των παραμέτρων momentum, weight_decay του optimizer

Ωστόσο δεν παρατήρησα καμία βελτίωση.

Έτσι δοκίμασα να απλοποιήσω το νευρωνικό δίκτυο.
Πιο συγκεκριμένα:

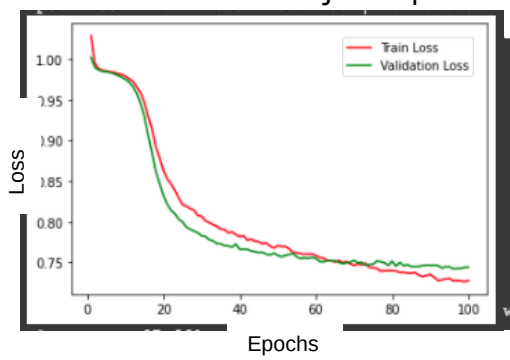
1. Αφαιρώ το τελευταίο hidden layer και πλέον έχουμε συνολικά 4 layers (2 hidden).
2. Μειώνω το μέγεθος των layers

Επίσης, λόγω της απλοποίησης του δικτύου μειώνω το **learning rate** σε **0.0025**.
(Η τιμή αυτή προέκυψε έπειτα από διαδοχικές δοκιμές μειώνοντας σταδιακά την τιμή)

Έτσι το νέο νευρωνικό δίκτυο είναι το εξής:

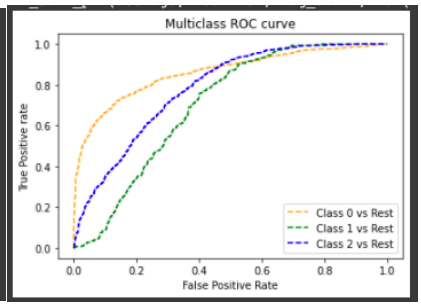
```
Net(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (model_stack): Sequential(
    (0): Linear(in_features=200, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.25, inplace=False)
    (6): Linear(in_features=32, out_features=16, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=16, out_features=3, bias=True)
  )
)
```

Εκπαιδεύοντας το παραπάνω δίκτυο έχουμε:




Accuracy: 67.66%
f1 score: 63.37%
Precision: 60.74%
Recall: 67.66%

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.80 | 0.74 | 0.77 | 1065 |
| 1 | 0.00 | 0.00 | 0.00 | 296 |
| 2 | 0.59 | 0.82 | 0.68 | 921 |
| accuracy | | | 0.68 | 2282 |
| macro avg | 0.46 | 0.52 | 0.48 | 2282 |
| weighted avg | 0.61 | 0.68 | 0.63 | 2282 |




Παρατηρούμε πώς:

- Οι διαφορές μεταξύ των train και validation losses/scores έχουν μειωθεί ακόμα περισσότερο.
- Το validation loss curve είναι πλέον ομαλό.
- Έχει εξαλειφθεί πλήρως overfit.

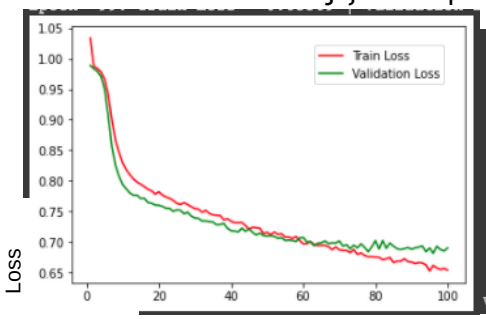
Ωστόσο:

- Το συνολικό f1-score του μοντέλου μειώθηκε.
- Το μοντέλο αδυνατεί να αναγνωρίσει την κλάση 1, κάτι που συμπεραίνουμε τόσο από τα επιμέρους precision/recall/f1 της κλάσης 1 όσο και από το ROC curve το οποίο έχει μετατοπιστεί χαμηλότερα σε σχέση με πριν.

Έτσι έπειτα από πειραματισμούς και δοκιμές επιλέγω:

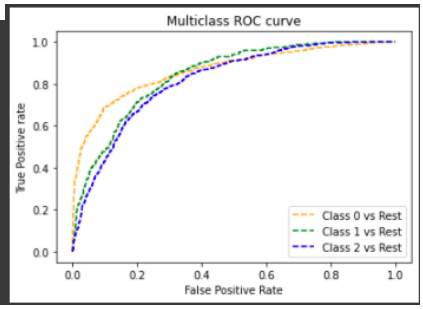
- Να εφαρμόσω επιπλέον προεπεξεργασία στα δεδομένα και συγκεκριμένα αφαίρεση **stopwords** και **lemmatization**.
- Να θέσω την τιμή της παραμέτρου **momentum** του optimizer σε **0.5**.Η τιμή αυτή προέκυψε έπειτα από δοκιμές ξεκινώντας από την τιμή 0.9 (η οποία είναι και η πιο συνηθισμένη) και μειώνοντας την σταδιακά.

Εκπαιδεύοντας ξανά το μοντέλο παίρνουμε τα εξής αποτελέσματα:




Accuracy: 70.11%
f1 score: 70.02%
Precision: 70.05%
Recall: 70.11%

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.79 | 0.76 | 0.77 | 1065 |
| 1 | 0.45 | 0.41 | 0.43 | 296 |
| 2 | 0.68 | 0.73 | 0.70 | 921 |
| accuracy | | | 0.70 | 2282 |
| macro avg | 0.64 | 0.63 | 0.64 | 2282 |
| weighted avg | 0.70 | 0.70 | 0.70 | 2282 |



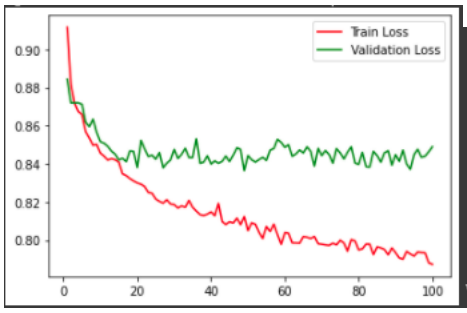

Παρατηρούμε πώς:

- Τα accuracy/f1-score/recall/precision του validation είναι υψηλότερα από κάθε άλλη εκτέλεση.
- Τα ROC curves έχουν πολύ υψηλές τιμές true positive rate για όλες τις κλάσεις, κάτι που επαληθεύει την καλή απόδοση του μοντέλου μας.
- Το μοντέλο δεν παρουσιάζει πλέον overfit καθώς τα loss curves βρίσκονται καταλήγουν πολύ κοντά στο τέλος της εκπαίδευσης.
- Και τα δύο loss curves έχουν πολύ ομαλή πορεία, σε αντίθεση με προηγούμενα μοντέλα που δοκιμάσαμε.

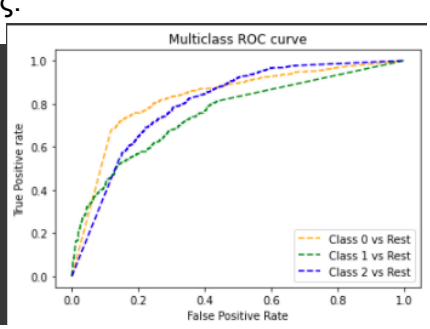
Έτσι καταλήγουμε στο τελικό μας μοντέλο το οποίο συνοψίζεται στην επόμενη παράγραφο.

## Πειραματισμός με τον optimizer

Στην συνέχεια της μελέτης θα δοκιμάσω να χρησιμοποιήσω και διαφορετικούς optimizers αντί του SGD.
Έτσι στο παραπάνω μοντέλο, θέτω σαν optimizer τον Adam (αφαιρώντας την παράμετρο momentum που είχα θέσει στον SGD).
Το υπόλοιπο μοντέλο μένει ως έχει. Τα αποτελέσματα είναι τα εξής:




```
Accuracy: 69.11%
f1 score: 67.89%
Precision: 68.38%
Recall: 69.11%
              precision    recall  f1-score   support

           0       0.76      0.77      0.76      1065
           1       0.55      0.25      0.35       296
           2       0.64      0.74      0.69       921

    accuracy                           0.69      2282
   macro avg       0.65      0.59      0.60      2282
weighted avg       0.68      0.69      0.68      2282
```



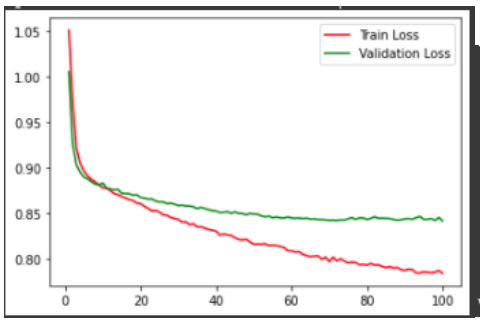

Παρατηρούμε πως:

- Τα scores των μετρικών καθώς και τα ROC curves είναι αρκετά καλά.
- Ωστόσο από τα loss curves παρατηρούμε πως το μοντέλο παρουσιάζει overfit λόγω της μεγάλης απόκλισης. Επίσης παρατηρούμε «αναταραχές» και στις δύο καμπύλες.

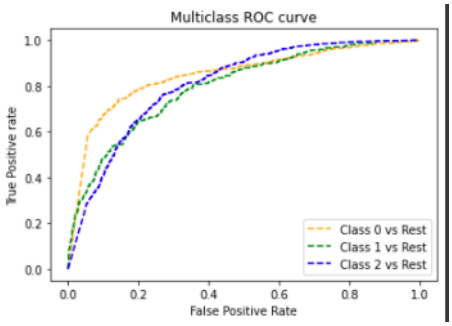
Έτσι, αποφασίζω να:

- Μειώσω το learning rate σε 1e-4
- Αυξήσω το dropout του δεύτερου hidden layer από 0.1 σε 0.2

Έπειτα από τις δύο αυτές αλλαγές τα αποτελέσματα είναι τα εξής:




```
Accuracy: 70.82%
f1 score: 69.63%
Precision: 70.17%
Recall: 70.82%
              precision    recall  f1-score   support

           0       0.77      0.79      0.78      1065
           1       0.59      0.27      0.37       296
           2       0.66      0.75      0.70       921

    accuracy                           0.71      2282
   macro avg       0.67      0.61      0.62      2282
weighted avg       0.70      0.71      0.70      2282
```




Παρατηρούμε:

- Τα scores των μετρικών αυξήθηκαν και τα ROC curves βελτιώθηκαν.
- Τα loss curves πλέον είναι πολύ σταθερά, χωρίς αναταράξεις.
- Η απόκλιση των δύο curves, δηλαδή το overtfit, μειώθηκε. Ωστόσο εξακολουθεί να υπάρχει.

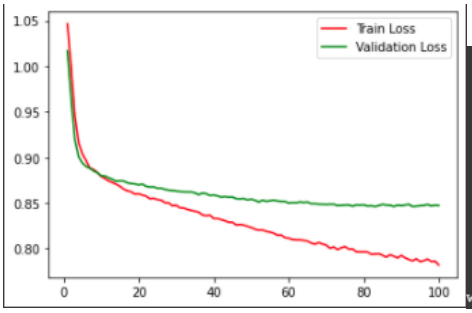
Με στόχο την εξάλειψη του overfit, δοκίμασα τα εξής:

- Μείωση/Αύξηση του learning rate
- Περεταίρω αύξηση του dropout.
- Απλοποίηση του δικτύου μειώνοντας τα μεγέθη των layers από 64-32-16 σε 32-32-16 και στην συνέχεια σε 32-16-16.
- Διαφορετικά activation functions και loss function.

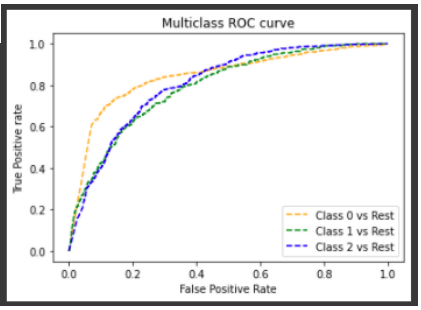
Ωστόσο δεν είχαν θετική επίδραση στην συμπεριφορά του μοντέλου.

Στην συνέχεια δοκίμασα και άλλους optimizers.
Παραθέτω τα αποτελέσματα αυτών τα οποία προέκυψαν από αντίστοιχη μελέτη με αυτήν που έκανα για τον Adam.

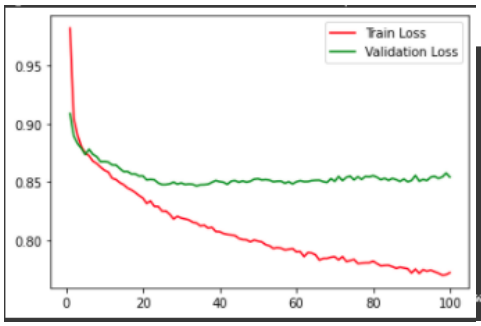
**RMSprop**



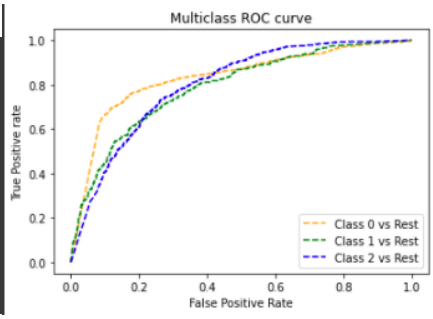

```
Accuracy: 69.76%
f1 score: 68.38%
Precision: 68.54%
Recall: 69.76%
              precision    recall  f1-score   support

           0       0.76      0.79      0.78      1065
           1       0.51      0.23      0.32       296
           2       0.65      0.74      0.69       921

    accuracy                           0.70      2282
   macro avg       0.64      0.59      0.60      2282
weighted avg       0.69      0.70      0.68      2282
```




**Adamax**




```
Accuracy: 69.24%
f1 score: 67.65%
Precision: 68.33%
Recall: 69.24%
              precision    recall  f1-score   support

           0       0.76      0.78      0.77      1065
           1       0.54      0.21      0.30       296
           2       0.64      0.75      0.69       921

    accuracy                           0.69      2282
   macro avg       0.65      0.58      0.59      2282
weighted avg       0.68      0.69      0.68      2282
```



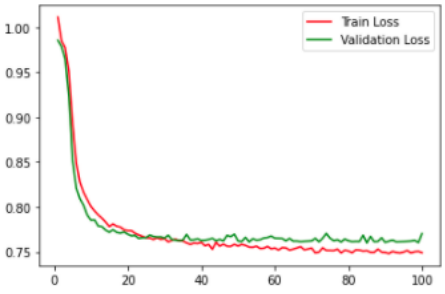

Οι παρατηρήσεις είναι αντίστοιχες με αυτές που ανέφερα για τον Adam.

Σε κάθε περίπτωση παρατήρησα πως χρησιμοποιώντας τον SGD σαν optimizer πετύχαινα την καλύτερη απόδοση.

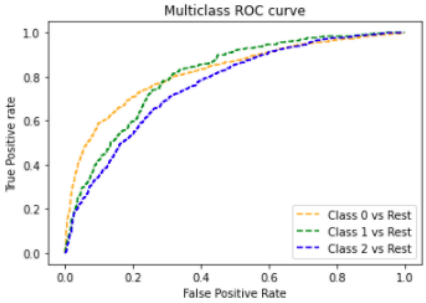
## Πειραματισμός με τα activation functions

Κατά την διάρκεια της μελέτης δοκίμαζα διαφορετικά activation functions στο μοντέλο μου. Επίσης, δοκίμαζα και το μοντέλο χωρίς καθόλου activation functions.
Στην συνέχεια παραθέτω τα αποτελέσματα χρησιμοποιώντας διαφορετικά activation functions στο τελικό μου μοντέλο. Λόγω του μεγάλου πλήθους από διαθέσιμα activation functions παρουσιάζω ενδεικτικά μόνο μερικά από αυτά.

### Χωρίς activation function



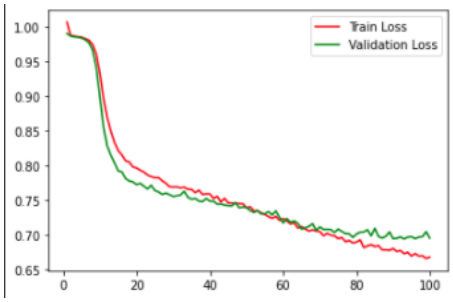

```
Accuracy: 67.00%
f1 score: 66.26%
Precision: 67.00%
Recall: 67.00%
              precision    recall  f1-score   support

           0       0.77      0.71      0.73      1065
           1       0.51      0.29      0.37       296
           2       0.61      0.75      0.67       921

    accuracy                           0.67      2282
   macro avg       0.63      0.58      0.59      2282
weighted avg       0.67      0.67      0.66      2282
```



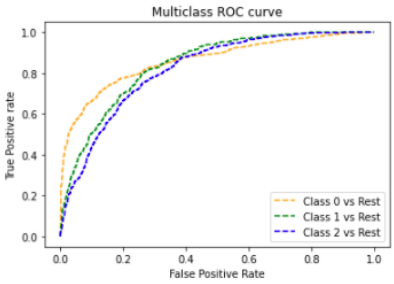

Παρατηρούμε:

- Τα scores των μετρικών είναι χαμηλότερα περίπου κατά 3% σε σχέση με την ReLU.
- Τα loss curves έχουν πολύ καλή πορεία και είναι πολύ σταθερά. Από κάποιο σημείο και μετά σταθεροποιούνται χωρίς να μειώνεται κανένα από τα δύο. Δηλαδή αδυνατεί να μειωθεί περισσότερο το loss.
- Το μοντέλο δεν παρουσιάζει underfit/overfit καθώς οι δύο καμπύλες βρίσκονται πολύ κοντά.

### LeakyReLU



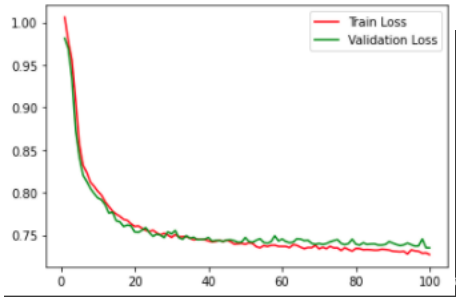

```
Accuracy: 70.29%
f1 score: 69.29%
Precision: 70.73%
Recall: 70.29%
              precision    recall  f1-score   support

           0       0.82      0.73      0.77      1065
           1       0.54      0.27      0.36       296
           2       0.63      0.81      0.71       921

    accuracy                           0.70      2282
   macro avg       0.66      0.60      0.61      2282
weighted avg       0.71      0.70      0.69      2282
```



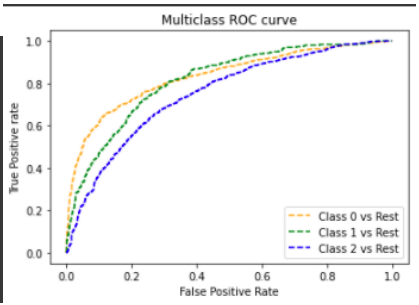

Παρατηρούμε:

- Η συμπεριφορά είναι ακριβώς ίδια με την ReLU.
- Το f1-score είναι ελάχιστα χαμηλότερο. Βέβαια αυτό διαφέρει σε κάθε εκτέλεση λόγω της τυχαιότητας των νευρωνικών δικτύων.

### Tanh



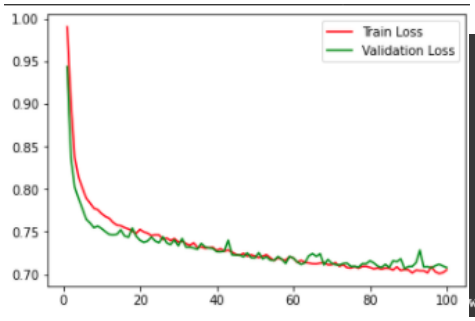

```
Accuracy: 67.57%
f1 score: 66.93%
Precision: 67.12%
Recall: 67.57%
              precision    recall  f1-score   support

           0       0.75      0.73      0.74      1065
           1       0.52      0.32      0.40       296
           2       0.63      0.72      0.67       921

    accuracy                           0.68      2282
   macro avg       0.63      0.59      0.60      2282
weighted avg       0.67      0.68      0.67      2282
```



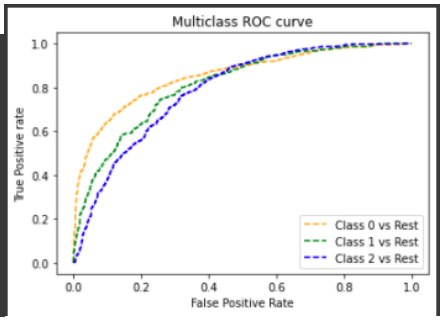
Παρατηρούμε:

- Αντίστοιχη συμπεριφορά με το μοντέλο χωρίς activation function.

**SELU**




```
Accuracy: 69.50%
f1 score: 68.85%
Precision: 69.75%
Recall: 69.50%
              precision    recall  f1-score   support

           0       0.80      0.73      0.76      1065
           1       0.55      0.32      0.40       296
           2       0.63      0.78      0.70       921

    accuracy                           0.70      2282
   macro avg       0.66      0.61      0.62      2282
weighted avg       0.70      0.70      0.69      2282
```




Παρατηρούμε:

- Πολύ καλή συμπεριφορά τόσο από την πορεία των loss curves τα οποία σχεδόν ταυτίζονται όσο και από τα scores των μετρικών.
- Ωστόσο υστερεί και αυτή στο τελικό validation score σε σχέση με την ReLU.

Τελικά το καλύτερο score επιτυγχάνεται χρησιμοποιώντας την ReLU ή την LeakyReLU. Επιλέγω να χρησιμοποιήσω την ReLU.

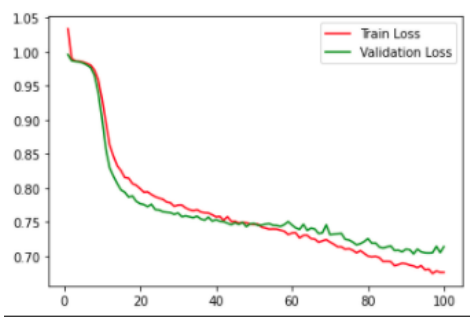
### Πειραματισμός με τα loss functions

Έχοντας ένα πρόβλημα multiclass classification δεν έχει νόημα να χρησιμοποιήσουμε loss functions όπως το MSE.

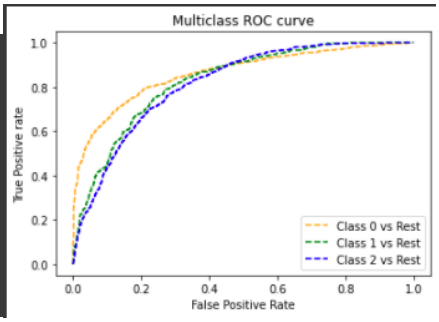
Έτσι, κατά την διάρκεια της μελέτης μου, εκτός από το CrossEntropyLoss δοκίμασα να χρησιμοποιήσω και το:

- **Negative Log Likelihood Loss (NLLLOSS)**

Αρχικά θέτω σαν loss function την nn.NLLLoss.
Επίσης, έπειτα από το output layer εφαρμόζω την nn.LogSoftmax στο output.
Τα αποτελέσματα είναι τα εξής:




```
Accuracy: 67.48%
f1 score: 64.64%
Precision: 71.14%
Recall: 67.48%
              precision    recall  f1-score   support

           0       0.84      0.66      0.74      1065
           1       0.66      0.08      0.15       296
           2       0.57      0.88      0.70       921

    accuracy                           0.67      2282
   macro avg       0.69      0.54      0.53      2282
weighted avg       0.71      0.67      0.65      2282
```




Παρατηρούμε:

- Τα loss curves έχουν παρόμοια συμπεριφορά/πορεία σε σχέση με αυτά της CrossEntropyLoss (δηλαδή του τελικού μου μοντέλου).
- Ωστόσο υστερεί αρκετα στο τελικό validation score.

Έτσι επιλέγω να χρησιμοποιήσω το CrossEntropyLoss καθώς παρουσιάζει καλύτερη συμπεριφορά/απόδοση.
Άλλωστε και με βάση την θεωρία το CrossEntropyLoss είναι η πιο συνηθισμένη επιλογή για τέτοιου είδους προβλήματα.

### Σημαντική παρατήρηση

Οι πειραματισμοί που αναφέρθηκαν στις παραγράφους: Πειραματισμός με τον optimizer, Πειραματισμός με τα activation functions, Πειραματισμός με τα loss functions πραγματοποιήθηκαν σε κάθε ένα από στα στάδια της αναλυτικής μελέτης που περιγράφηκε στην παράγραφο Μελέτη για την εύρεση του καλύτερου μοντέλου NN.
Δηλαδή, κάθε ένα βήμα βελτίωσης του μοντέλου δοκίμαζα πολλούς συνδυασμούς αυτών.
Ωστόσο δεν αναφέρονται σε κάθε ένα βήμα καθώς κάτι τέτοιο θα έκανε την συνολική παρουσίαση της εργασίας υπερβολικά εκτενή. Έτσι επέλεξα να τα παρουσιάσω τους πειραματισμούς αυτούς μόνο για το τελικό μοντέλο.

## Τελικό μοντέλο

Για την παραγωγή του τελικού μοντέλου αρχικά εφαρμόζουμε την εξής προεπεξεργασία στα δεδομένα:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions και των αριθμών.
- Αντικατάσταση των emojis σε raw text (π.χ. 😀 → grinning_face).
- Αφαίρεση των συμβόλων, σημείων στίξης.
- Αφαίρεση stopwords.
- Lemmatization.

Για την δημιουργία των αναπαραστάσεων των tweets μέσω pre-trained word embedding vectors χρησιμοποιώ το αρχείο **glove.twitter.27B.200d.** Το feature vector ενός tweet προκύπτει από το average vector των feature vectors των λέξεων του tweet (εξηγείται αναλυτικά στην παράγραφο Pre-trained word embedding vectors (GloVe)).

Το Νευρωνικό Δίκτυο που χρησιμοποιείται έχει την εξής δομή:

```
Net(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (model_stack): Sequential(
    (0): Linear(in_features=200, out_features=64, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=64, out_features=32, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.25, inplace=False)
    (6): Linear(in_features=32, out_features=16, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=16, out_features=3, bias=True)
  )
)
```

Ο αριθμός των Epochs είναι 100.

Οι DataLoaders που θα χρησιμοποιηθούν για το train έχουν τις εξής παραμέτρους:

- batch_size = 16
- shuffle=True

Το LossFunction που θα χρησιμοποιηθεί είναι το nn.CrossEntropyLoss()

Παρατήρηση: βλ. Σημαντική Παρατήρηση

Ο optimizer που χρησιμοποιείται είναι ο SGD με τις εξής παραμέτρους:

- learning_rate = 0.0025
- momentum = 0.5

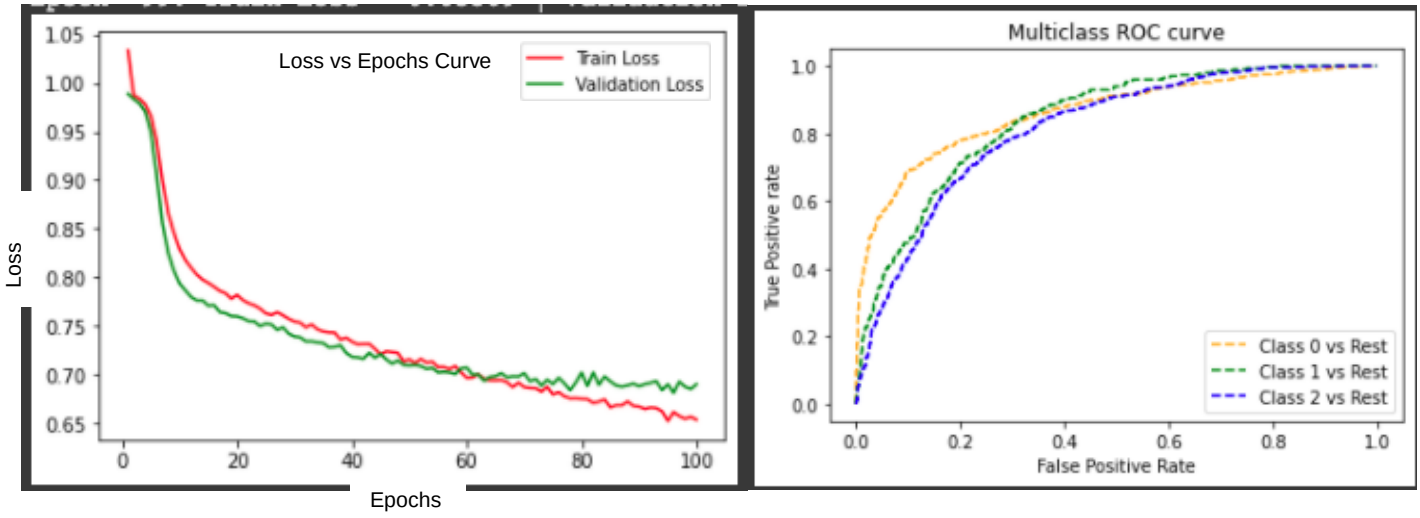
## Αξιολόγηση μοντέλου

Όπως έχει αναφερθεί, κατά την εκπαίδευση του μοντέλου σε κάθε epoch υπολογίζονται τα train/validation loss, train/validation f1-score και το validation accuracy.
Αποσπασματικά για κάποια epochs, κατά την εκπαίδευση του τελικού μοντέλου, παίρνουμε τις εξής τιμές:

| Epoch | Train Loss | Validation Loss | Train F1-Score | Validation F1-Score | Validation Accuracy |
|---|---|---|---|---|---|
| 0 | 1.03340 | 0.98883 | 40.60 | 29.70 | 46.96 |
| 10 | 0.81783 | 0.78661 | 59.78 | 61.68 | 65.90 |
| 20 | 0.77681 | 0.75810 | 61.86 | 62.63 | 66.87 |
| 40 | 0.73135 | 0.71785 | 63.59 | 64.12 | 68.40 |
| 60 | 0.69716 | 0.70005 | 66.95 | 66.95 | 69.63 |
| 80 | 0.67489 | 0.68840 | 70.01 | 69.00 | 69.85 |
| 99 | 0.65369 | 0.69011 | 71.52 | 70.02 | 70.11 |

Μετά το τέλος της εκπαίδευσης, τα αποτελέσματα των μετρικών είναι:

```
Accuracy: 70.11%
f1 score: 70.02%
Precision: 70.05%
Recall: 70.11%
              precision    recall  f1-score   support

           0       0.79      0.76      0.77      1065
           1       0.45      0.41      0.43       296
           2       0.68      0.73      0.70       921

    accuracy                           0.70      2282
   macro avg       0.64      0.63      0.64      2282
weighted avg       0.70      0.70      0.70      2282
```




Συμπεράσματα:

- Το μοντέλο μας έχει πολύ καλά scores με βάση τις μετρικές που παρουσιάζονται παραπάνω. Υστερεί στην αναγνώριση των tweets της κλάσης 1, το οποίο οφείλεται στον μικρό όγκο δειγμάτων κλάσης 1 τόσο στο train όσο και στο validation set.
- Το μοντέλο δεν παρουσιάζει overfit κάτι που συμπεραίνουμε τόσο από τα loss curves τα οποία βρίσκονται πολύ κοντά όσο και από τα train/validation f1-scores τα οποία επίσης είναι πολύ κοντά.

- Παρατηρούμε ότι για τα πρώτα epochs το μοντέλο τα πηγαίνει καλύτερα στο validation set από το train set, δηλαδή παρουσιάζει underfit. Αυτό οφείλεται στην απλότητα του νευρωνικού δικτύου καθώς και στα dropouts που έχουν προτεθεί ενδιάμεσα των layers. Ωστόσο, με την πάροδο των epochs το φαινόμενο του underfit όλο και μειώνεται και τελικά εξαλείφεται. Δηλαδή το μοντέλο μας, μετά το τέλος της εκπαίδευσης δεν παρουσιάζει underfit.
- Από τα ROC curves επαληθεύεται η καλή συμπεριφορά του μοντέλου μας και για τις τρείς κλάσεις. Πιο συγκεκριμένα παρατηρούμε πως τα curves πλησιάζουν προς την «πάνω αριστερά γωνία» δηλαδή τα true positives υπερισχύουν.

Γενικά Συμπεράσματα:

- Αυξάνοντας την πολυπλοκότητα του δικτύου παρατηρήσαμε πως δεν αυξάνεται η απόδοση του μοντέλου. Αντίθετα, χρησιμοποιώντας ένα πιο απλό δίκτυο και προσαρμόζοντας κατάλληλα τις παραμέτρους πετύχαμε ένα πολύ καλύτερο αποτέλεσμα.
- Η προεπεξεργασία των δεδομένων αποδείχθηκε αρκετά αποδοτική χρησιμοποιώντας word embeddings για την αναπαράσταση καθώς αυξάνει αρκετά την απόδοση του μοντέλου.

## Μοντέλο 2 – TF-IDF / Feed Forward NN

### Προεπεξεργασία Δεδομένων

Αρχικά επιλέγω να εφαρμόσω την ίδια προεπεξεργασία η οποία είχε αποδειχθεί βέλτιστη στην πρώτη εργασία και μέσω της μελέτης είχε αποδειχθεί αποδοτική σε συνδυασμό με τους vectorizers. Δηλαδή εφαρμόζω:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions και των αριθμών καθώς δεν προσφέρουν καμία πληροφορία σχετικά με το “νόημα” της πρότασης.
- Αντικατάσταση των emojis σε raw text (π.χ. 😀 → grinning_face) καθώς πολλές φορές αποτυπώνουν πληροφορία σχετικά με το συναίσθημα.
- Αφαίρεση των συμβόλων, σημείων στίξης με εξαίρεση τα hashtag καθώς αποτελούν λέξεις κλειδιά για ένα tweet. Επίσης, δεν αφαιρώ τα θαυμαστικά και τα ερωτηματικά καθώς παρατήρησα καλύτερη απόδοση, αφού και αυτά πολλές φορές δηλώνουν συναίσθημα (π.χ. εκνευρισμό).

### Vectorization

Για την δημιουργία των αναπαραστάσεων ακολουθώ τα ίδια βήματα με αυτά του βέλτιστου μοντέλου της πρώτης εργασίας.
Η αναλυτική μελέτη μέσω της οποία προέκυψε η μέθοδος αναπαράστασης περιγράφηκε στην εργασία 1 για αυτό και παραλείπεται.

Για την αναπαράσταση των tweets χρησιμοποιώ τον TF-IDF Vectorizer με τις εξής παραμέτρους:

- min_df = 1 (default)
- max_df = 1.0 (default)
- max_features = 600
- n_gram = (1,3)

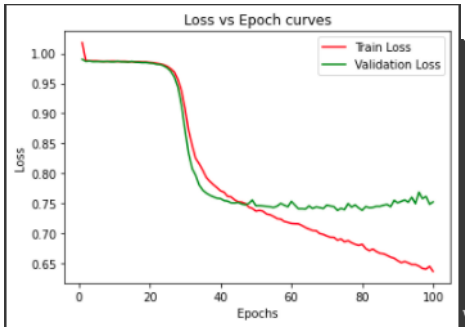
Περιορίζω των αριθμό των features που προκύπτουν για κάθε αναπαράσταση μέσω της παραμέτρου max_features και όχι μέσω decomposition όπως στην πρώτη εργασία καθώς το decompision δεν βελτίωνε την απόδοση του μοντέλου ενώ ταυτόχρονα αυξάνει σημαντικά τον χρόνο εκτέλεσης.
Επίσης όπως προέκυψε και από την μελέτη της πρώτης εργασίας η μείωση των features μέσω των παραμέτρων min_df και max_df αποδείχθηκε λιγότερο αποδοτική από ότι μέσω της παραμέτρου max_features.

## Μελέτη για την εύρεση του καλύτερου μοντέλου NN

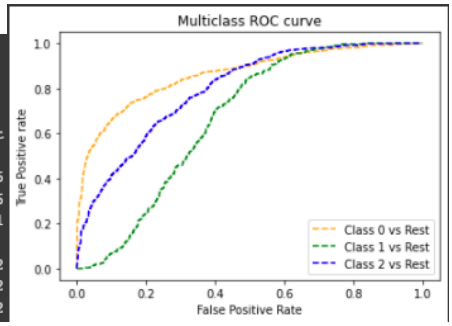
Έχοντας τις αναπαραστάσεις για κάθε tweet ακολουθώ αντίστοιχη διαδικασία με αυτήν που ακολούθησα στην μελέτη του Μοντέλου 1.

Αρχικά δοκιμάζω το δίκτυο που προέκυψε στο τελικό μοντέλο του Μοντέλου 1 και περιγράφεται στην παράγραφο Τελικό μοντέλο.

Τα αποτελέσματα είναι τα εξής:




```
Accuracy: 67.44%
f1 score: 63.11%
Precision: 60.14%
Recall: 67.44%
              precision    recall  f1-score   support

           0       0.78      0.75      0.77      1065
           1       0.00      0.00      0.00       296
           2       0.59      0.80      0.68       921

    accuracy                           0.67      2282
   macro avg       0.46      0.52      0.48      2282
weighted avg       0.60      0.67      0.63      2282
```




Παρατηρούμε τα εξής:

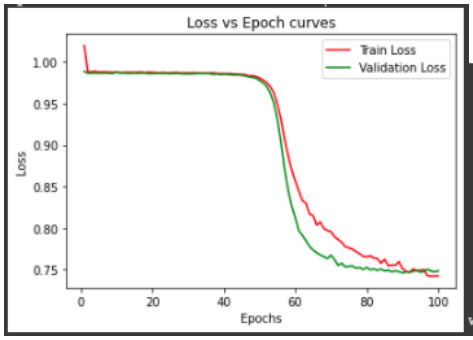
- Το μοντέλο αρχικά αδυνατεί να εκπαιδευτεί.
- Έπειτα από αρκετά epochs, το loss αρχίζει να μειώνεται και το μοντέλο να εκπαιδεύεται.
- Το μοντέλο αδυνατεί να αναγνωρίσει την κλάση 1, κάτι που φάινεται τόσο από τα επιμέρους precision/recall/f1-score της κλάσης 1 όσο και από το ROC curve το οποίο είναι πολύ κοντά στην ευθεία y=x δηλαδή στον random classifier.
- Το μοντέλο παρουσιάζει μεγάλο overfit καθώς από τα loss curves φαίνεται πως το train loss μειώνεται συνεχώς ενώ αντίθετα το validation loss παραμένει σταθερό και αυξάνεται ελάχιστα.

Έχοντας ήδη ορίσει κατάλληλα τα dropout καθώς και όλες τις υπόλοιπες παραμέτρους για το συγκεκριμένο μοντέλο από την μελέτη του Μοντέλου 1, επιλέγω να κατασκευάσω ένα απλούστερο νευρωνικό δίκτυο έτσι ώστε να αντιμετωπίσω το overfit.

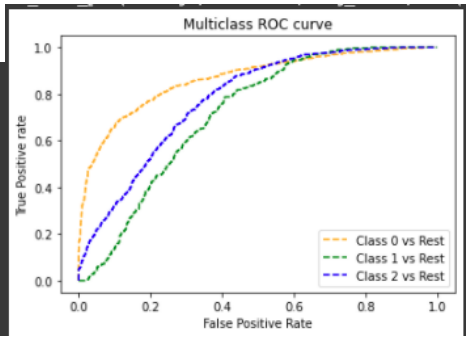
Πιο συγκεκριμένα επιλέγω να μειώσω το μέγεθος των hidden layers από 64,32,16 σε 20,10,10.
Δηλαδή το νέο μοντέλο είναι το εξής:

```
Net(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (model_stack): Sequential(
    (0): Linear(in_features=600, out_features=20, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=20, out_features=10, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.25, inplace=False)
    (6): Linear(in_features=10, out_features=10, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=10, out_features=3, bias=True)
  )
)
```

Εκπαιδεύοντας το παραπάνω μοντέλο παίρνουμε τα εξής αποτελέσματα:




```
Accuracy: 67.48%
f1 score: 63.22%
Precision: 61.31%
Recall: 67.48%
              precision    recall  f1-score   support

           0       0.81      0.71      0.76      1065
           1       0.00      0.00      0.00       296
           2       0.58      0.85      0.69       921

    accuracy                           0.67      2282
   macro avg       0.46      0.52      0.48      2282
weighted avg       0.61      0.67      0.63      2282
```



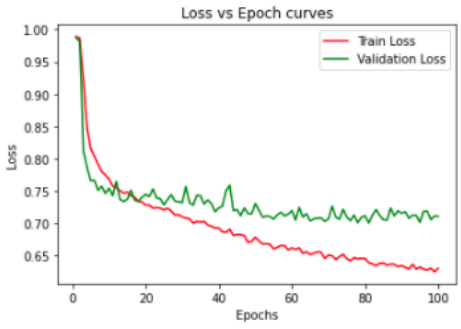

Παρατηρούμε ότι:

- Το μοντέλο αργεί ακόμα περισσότερο να εκπαιδευτεί και αδυνατεί να αναγνωρίσει την κλάση 1.
- Πλέον έχει εξαλειφθεί τελείως το overfit που παρουσιάστηκε προηγουμένως και αντίθετα παρουσιάζεται underfit.

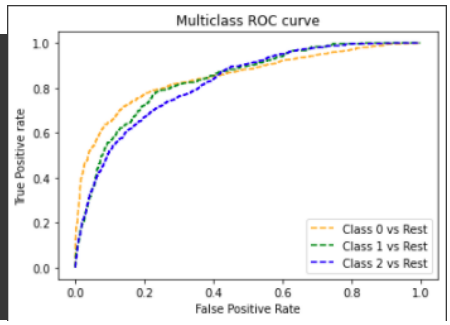
Λόγω των δύο παρατηρήσεων, κάνω διαδοχικές δοκιμές αυξάνοντας σταδιακά τις τιμές του learning_rate και του momentum.
Τελικά καταλήγω στις εξής τιμές:

- **learning_rate = 0.1**
- **momentum = 0.9** (default)

Χρησιμοποιώντας τις παραπάνω παραμέτρους για τον optimizer τα αποτελέσματα που παίρνουμε είναι:




```
Accuracy: 69.98%
f1 score: 68.93%
Precision: 70.28%
Recall: 69.98%
              precision    recall  f1-score   support

           0       0.81      0.73      0.77      1065
           1       0.56      0.26      0.36       296
           2       0.63      0.80      0.70       921

    accuracy                           0.70      2282
   macro avg       0.67      0.60      0.61      2282
weighted avg       0.70      0.70      0.69      2282
```




Παρατηρούμε ότι:

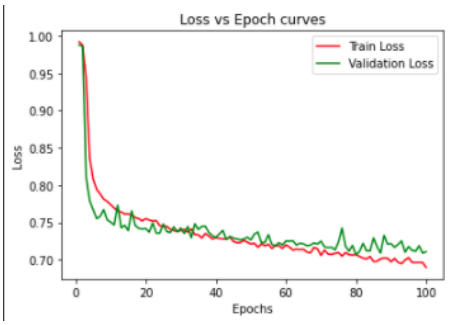
- Η απόδοση του μοντέλου έχει βελτιωθεί σημαντικά. Πλέον εκπαιδεύεται με πολύ καλό ρυθμό κάτι που φαίνεται και από την καμπύλη του train loss.
- Τα scores/μετρικές για το validation set έχουν πολύ καλές τιμές. Αντίστοιχα και τα ROC curves.
- Ωστόσο το μοντέλο παρουσιάζει ένα μικρό overfit καθώς οι καμπύλες train/validation loss απέχουν αρκετά.

Έτσι ώστε να εξαλειφθεί το overfit, επιλέγω να μειώσω των αριθμό των features που παράγονται από τον TF-IDF για κάθε αναπαράσταση.
Έτσι πραγματοποιώ διαδοχικές δοκιμές μειώνοντας σταδιακά την παράμετρο max_features του TF-IDF.

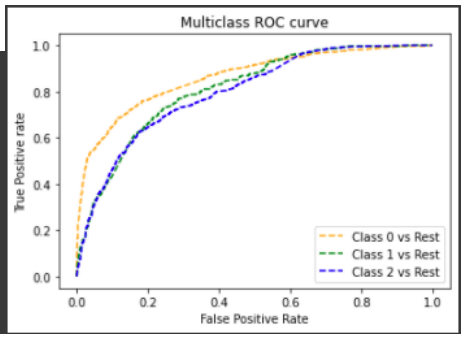
Τελικά καταλήγω στην τιμή max_features=300 για την οποία το μοντέλο παρουσίασε την καλύτερη απόδοση.

Επίσης, εφαρμόζω επιπλέον προεπεξεργασία στα δεδομένα αφαιρώντας τα **stopwords** και εφαρμόζοντας **lemmatization**.

Η μόνη διαφορά στο νευρωνικό δίκτυο σε σχέση με την προηγούμενη εκτέλεση που παρουσιάστηκε είναι ότι πλέον το input layer δέχεται 300 αντί για 600 features λόγω των νέων αναπαραστάσεων.
Τα αποτελέσματα είναι τα εξής:




```
Accuracy: 68.71%
f1 score: 67.20%
Precision: 69.46%
Recall: 68.71%
              precision    recall  f1-score   support

           0       0.82      0.72      0.76      1065
           1       0.55      0.19      0.28       296
           2       0.60      0.81      0.69       921

    accuracy                           0.69      2282
   macro avg       0.66      0.57      0.58      2282
weighted avg       0.69      0.69      0.67      2282
```




Παρατηρούμε πώς:

- Τα accuracy/f1-score/recall/precision του validation έχουν μειωθεί ελάχιστα (παραμένοντας αρκετά ικανοποιητικά), ωστόσο πλέον το μοντέλο δεν παρουσιάζει overfit καθώς τα loss curves βρίσκονται καταλήγουν πολύ κοντά στο τέλος της εκπαίδευσης.
- Τα ROC curves έχουν πολύ υψηλές τιμές true positive rate για όλες τις κλάσεις, κάτι που επαληθεύει την καλή απόδοση του μοντέλου μας.

Στην συνέχεια, έτσι ώστε να βελτιώσω ακόμη περισσότερο την απόδοση του μοντέλου και το μοντέλο μας να έχει μια πιο ομαλή συμπεριφορά στο validation αρχικά δοκίμασα:

- Να εφαρμόσω λιγότερη/περισσότερη προεπεξεργασία στα δεδομένα (π.χ. αφαίρεση stopwords, lemmatization, stemming)
- Batch normalization
- Διαφορετικά activation functions αντί του ReLU
- Πειραματισμό με διάφορες τιμές των παραμέτρων momentum, weight_decay του optimizer

Ωστόσο δεν παρατήρησα καμία βελτίωση.

Επίσης, όπως ακριβώς έκανα και στην πρώτη εργασία, δοκίμασα να χρησιμοποιήσω διαφορετικούς vectorizers για την αναπαράσταση των tweets. Συγκεκριμένα, εκτός του TF-IDF, δοκίμασα τους Count Vectorizes και Hashing Vecorizer. Οι παράμετροι με τις οποίες χρησιμοποίησα αυτούς τους vectorizers είναι αντίστοιχες με αυτές του TF-IDF (που παρουσιάστηκαν συνοπτικά παραπάνω) και έχουν περιγράφει αναλυτικά στην μελέτη της πρώτης εργασίας.
Χρησιμοποιώντας τους άλλους δύο vectorizers που αναφέρθηκαν δεν κατάφερα να αυξήσω την απόδοση του μοντέλου.

Έτσι καταλήγουμε στο τελικό μας μοντέλο το οποίο συνοψίζεται στην επόμενη παράγραφο.

### Παρατήρηση

Κατά την παραπάνω μελέτη δοκιμάστηκαν όλα όσα αναφέρονται στις παραγράφους Πειραματισμός με τον optimizer, Πειραματισμός με τα activation functions, Πειραματισμός με τα loss functions για το πρώτο μοντέλο.

Ωστόσο τα αποτελέσματα καθώς και οι παρατηρήσεις ήταν ακριβώς αντίστοιχες για αυτό και δεν αναφέρονται ξανά.
Δηλαδή αποδείχθηκε πως και σε αυτό το μοντέλο η καλυτερη επιλογή για optimizer, activation functions, και loss function είναι SGD, ReLU και CrossEntropyLoss αντίστοιχα.

## Τελικό μοντέλο

Για την παραγωγή του τελικού μοντέλου αρχικά εφαρμόζουμε την εξής προεπεξεργασία στα δεδομένα:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions και των αριθμών.
- Αντικατάσταση των emojis σε raw text (π.χ. 😀 → grinning_face).
- Αφαίρεση των συμβόλων, σημείων στίξης (εκτός των: '#' , '!' , '?' ).
- Αφαίρεση των stopwords.
- Lemmatization

Για την αναπαράσταση των tweets χρησιμοποιώ τον TF-IDF Vectorizer με τις εξής παραμέτρους:

- min_df = 1 (default)
- max_df = 1.0 (default)
- max_features = 300
- n_gram = (1,3)

Το Νευρωνικό Δίκτυο που χρησιμοποιείται έχει την εξής δομή:

```
Net(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (model_stack): Sequential(
    (0): Linear(in_features=300, out_features=20, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=20, out_features=10, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.25, inplace=False)
    (6): Linear(in_features=10, out_features=10, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.1, inplace=False)
    (9): Linear(in_features=10, out_features=3, bias=True)
  )
)
```

Ο αριθμός των Epochs είναι 100.

Οι DataLoaders που θα χρησιμοποιηθούν για το train έχουν τις εξής παραμέτρους:

- batch_size = 16
- shuffle=True

Το LossFunction που θα χρησιμοποιηθεί είναι το nn.CrossEntropyLoss()

Παρατήρηση: βλ. Σημαντική Παρατήρηση

Ο optimizer που χρησιμοποιείται είναι ο SGD με τις εξής παραμέτρους:

- learning_rate = 0.1
- momentum = 0.9 (default)

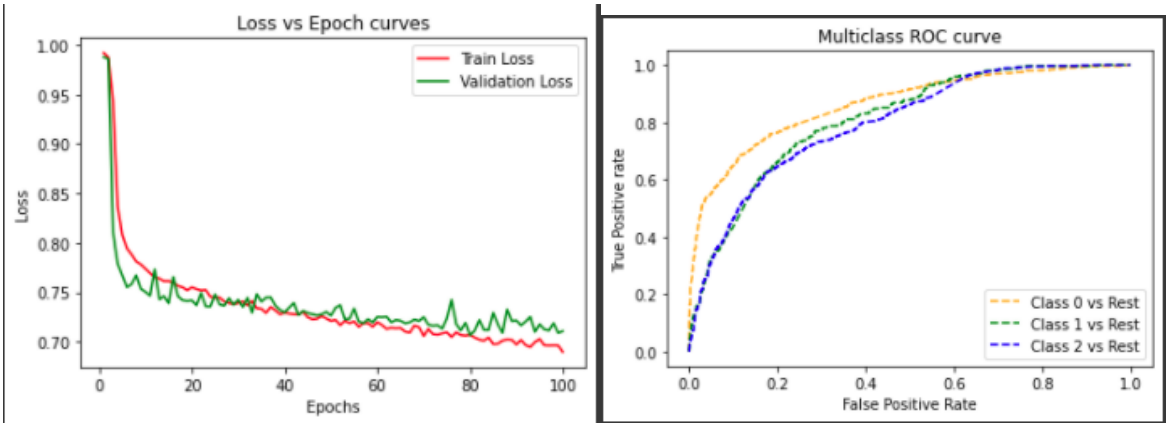
## Αξιολόγηση μοντέλου

Όπως έχει αναφερθεί, κατά την εκπαίδευση του μοντέλου σε κάθε epoch υπολογίζονται τα train/validation loss, train/validation f1-score και το validation accuracy.
Αποσπασματικά για κάποια epochs, κατά την εκπαίδευση του τελικού μοντέλου, παίρνουμε τις εξής τιμές:

| Epoch | Train Loss | Validation Loss | Train F1-Score | Validation F1-Score | Validation Accuracy |
|---|---|---|---|---|---|
| 0 | 0.99210 | 0.98767 | 36.01 | 29.70 | 46.66 |
| 10 | 0.76954 | 0.74662 | 62.67 | 63.01 | 67.30 |
| 20 | 0.75344 | 0.73717 | 63.78 | 63.41 | 67.74 |
| 40 | 0.72873 | 0.73478 | 64.33 | 63.10 | 67.48 |
| 60 | 0.71713 | 0.72503 | 64.53 | 63.43 | 67.70 |
| 80 | 0.70443 | 0.71138 | 65.69 | 64.21 | 68.09 |
| 99 | 0.68992 | 0.71098 | 67.04 | 67.20 | 68.71 |

Μετά το τέλος της εκπαίδευσης, τα αποτελέσματα των μετρικών είναι:

```
Accuracy: 68.71%
f1 score: 67.20%
Precision: 69.46%
Recall: 68.71%
              precision    recall  f1-score   support

           0       0.82      0.72      0.76      1065
           1       0.55      0.19      0.28       296
           2       0.60      0.81      0.69       921

    accuracy                           0.69      2282
   macro avg       0.66      0.57      0.58      2282
weighted avg       0.69      0.69      0.67      2282
```




Συμπεράσματα:
Τα συμπεράσματα που βγάζουμε από την παραπάνω μελέτη είναι αντίστοιχα με αυτά που παρουσιάστηκαν στα συμπεράσματα του Μοντέλου 1. Δηλαδή:

- Το μοντέλο μας έχει αρκετά καλά scores με βάση τις μετρικές που παρουσιάζονται παραπάνω. Υστερεί στην αναγνώριση των tweets της κλάσης 1, το οποίο οφείλεται στον μικρό όγκο δειγμάτων κλάσης 1 τόσο στο train όσο και στο validation set.
- Παρατηρούμε μια μικρή αστάθεια όσον αφορά το validation loss, την οποία δεν κατάφερα να εξομαλύνω.

- Το μοντέλο δεν παρουσιάζει overfit κάτι που συμπεραίνουμε τόσο από τα loss curves τα οποία βρίσκονται πολύ κοντά όσο και από τα train/validation f1-scores τα οποία επίσης είναι πολύ κοντά.
- Παρατηρούμε ότι για τα πρώτα epochs το μοντέλο τα πηγαίνει καλύτερα στο validation set από το train set, δηλαδή παρουσιάζει underfit. Αυτό οφείλεται στην απλότητα του νευρωνικού δικτύου καθώς και στα dropouts που έχουν προτεθεί ενδιάμεσα των layers. Ωστόσο, με την πάροδο των epochs το φαινόμενο του underfit όλο και μειώνεται και τελικά εξαλείφεται. Δηλαδή το μοντέλο μας, μετά το τέλος της εκπαίδευσης δεν παρουσιάζει underfit.
- Από τα ROC curves επαληθεύεται η καλή συμπεριφορά του μοντέλου μας και για τις τρείς κλάσεις. Πιο συγκεκριμένα παρατηρούμε πως τα curves πλησιάζουν προς την «πάνω αριστερά γωνία» δηλαδή τα true positives υπερισχύουν.

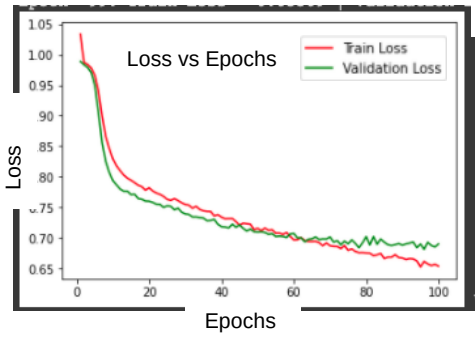
Γενικά Συμπεράσματα:

- Αυξάνοντας την πολυπλοκότητα του δικτύου παρατηρήσαμε πως δεν αυξάνεται η απόδοση του μοντέλου. Αντίθετα, χρησιμοποιώντας ένα πολύ απλό δίκτυο και προσαρμόζοντας κατάλληλα τις παραμέτρους πετύχαμε ένα πολύ καλύτερο αποτέλεσμα.
- Η προεπεξεργασία των δεδομένων αποδείχθηκε αρκετά αποδοτική σε συνδιασμό με τον TF-IDF για την αναπαράσταση καθώς αυξάνει αρκετά την απόδοση του μοντέλου.

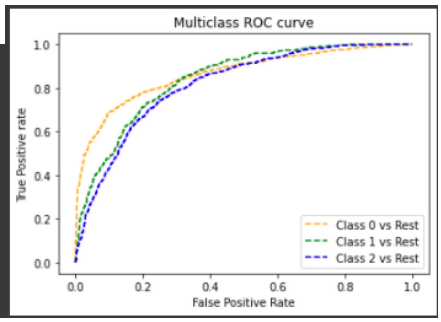
## Σύγκριση των δύο μοντέλων

Στην συνέχεια θα συγκρίνουμε τα δύο μοντέλα νευρωνικών δικτύων που παρουσιάστηκαν. Αρχικά παραθέτω ξανά τα αποτελέσματα των δύο μοντέλων.

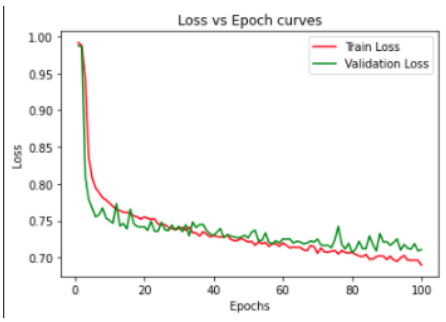
### Μοντέλο 1



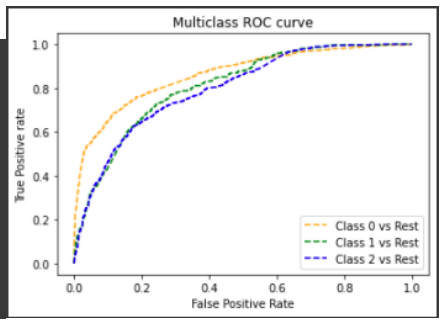

```
Accuracy: 70.11%
f1 score: 70.02%
Precision: 70.05%
Recall: 70.11%
              precision    recall  f1-score   support

           0       0.79      0.76      0.77      1065
           1       0.45      0.41      0.43       296
           2       0.68      0.73      0.70       921

    accuracy                           0.70      2282
   macro avg       0.64      0.63      0.64      2282
weighted avg       0.70      0.70      0.70      2282
```




### Μοντέλο 2




```
Accuracy: 68.71%
f1 score: 67.20%
Precision: 69.46%
Recall: 68.71%
              precision    recall  f1-score   support

           0       0.82      0.72      0.76      1065
           1       0.55      0.19      0.28       296
           2       0.60      0.81      0.69       921

    accuracy                           0.69      2282
   macro avg       0.66      0.57      0.58      2282
weighted avg       0.69      0.69      0.67      2282
```




- Όσον αφορά την προεπεξεργασία των δεδομένων παρατηρήσαμε πως αποδείχθηκε πολύ αποδοτική και για τα δύο μοντέλα. Και στα δύο μοντέλα χρησιμοποιήθηκαν ακριβώς οι ίδιες τεχνικές προεπεξεργασίας καθώς αποδείχθηκαν οι πιο αποδοτικές.
- Συγκρίνοντας τα scores των μετρικών παρατηρούμε πως το μοντέλο 1 έχει λίγο καλύτερη απόδοση. Η διαφορά αυτή οφείλεται στο γεγονός ότι το μοντέλο 1 αναγνωρίζει καλύτερα τα tweets της κλάσης 1, ενώ το μοντέλο 2 υστερεί σε αυτό. Γενικά, τα scores και των δύο μοντέλων είναι αρκετά ικανοποιητικά λαμβάνοντας υπόψιν το σχετικά μικρό train dataset το οποίο χρησιμοποιείται για την εκπαίδευση των μοντέλων.
- Όσον αφορά τα loss curves:
    - Παρατηρούμε πως τα train loss curves των δύο μοντέλων παρουσιάζουν αντίστοιχη συμπεριφορά. Η συμπεριφορά τους είναι ιδανική καθώς έχουν μια πολύ ομαλή καθοδική πορεία χωρίς «αναταράξεις». Αυτό υποδεικνύει πως η εκπαίδευση γίνεται σωστά και τόσο το learning rate όσο και οι οι υπόλοιπες παράμετροι έχουν τεθεί κατάλληλα.
    - Σχετικά με τα validation loss curves παρατηρούμε πως ακολουθούν αντίστοιχη πορεία με το train loss. Και στα δύο μοντέλα παρατηρούμε πως αρχικά το validation loss είναι χαμηλότερο από το train loss. Ωστόσο με την πάροδο των epochs το φαινόμενο του underfit εξαλείφεται και τα δύο curves συγκλίνουν (με το train loss να είναι ελάχιστα χαμηλότερο). Η σύγκλιση των δύο curve υποδεικνύει την πολύ καλή συμπεριφορά των μοντέλων μας.

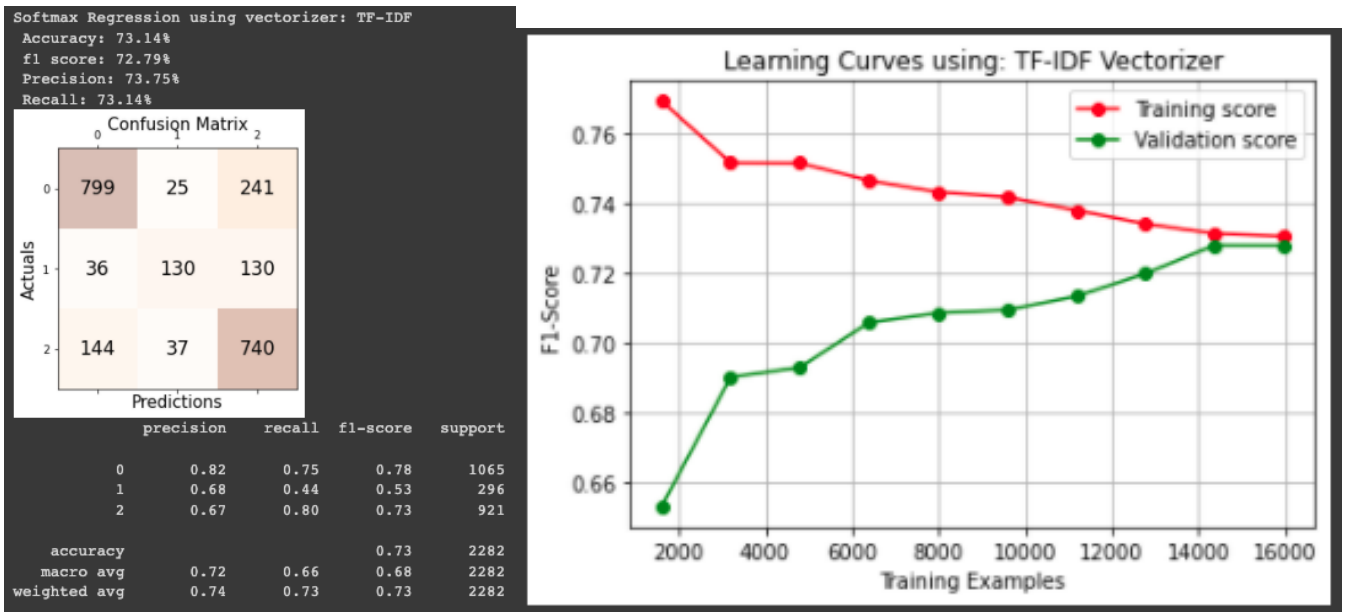
Τέλος, παρατηρούμε μια μικρή αστάθεια στο validation loss curve του μοντέλου 2, κάτι που δεν συμβαίνει στο μοντέλο 1.

- Όσον αφορά τα ROC curves, επαληθεύουν την καλή συμπεριφορά των μοντέλων μας. Τα ROC curves έχουν παρόμοια εικόνα και για τα δύο μοντέλα. Το True Positive rate είναι υψηλό σε όλα τα curves και απέχουν αρκετά (προς τα πάνω) από το ROC curve του random classifier (δηλαδή την ευθεία y=x). Παρατηρούμε πως και τα δύο μοντέλα είναι καλύτερα στην αναγνώριση tweets της κλάσης 0 σε σχέση με τις άλλες δύο κλάσεις καθώς τα curves της κλάσης 0 είναι υψηλότερα (πλησιάζουν περισσότερο την πάνω αριστερά γωνία). Αυτό επιβεβαιώνεται και από τα επιμέρους score των μετρικών της κλάσης 0. Επίσης, είναι και αναμενόμενο καθώς το πλήθος των tweets της κλάσης 0 υπερισχύουν στο train set.
- Όσον αφορά τον συνδυασμό αναπαραστάσεων και νευρωνικού δικτύου:
    - Η αναπαραστάσεις που προκύπτουν μέσω των word embedding είναι αρκετά ακριβείς. Έτσι το νευρωνικό δίκτυο που αποδείχθηκε να είναι το πιο αποδοτικό στο μοντέλο 1 είναι πιο σύνθετο από αυτό του μοντέλου 2. Ωστόσο παραμένει ένα σχετικά απλό δίκτυο και αυτό οφείλεται τόσο στην φύση των δεδομένων του προβλήματος (train/validation set) όσο και στον σχετικά μικρό όγκο των δεδομένων. Δοκιμάζοντας πιο σύνθετα δίκτυα μείωναν την απόδοση του μοντέλου.
    - Αντίθετα οι αναπαραστάσεις που προκύπτουν μέσω του TF-IDF είναι πιο απλές με αποτέλεσμα το βέλτιστο νευρωνικό δίκτυο που προέκυψε για το μοντέλο 2 να είναι αρκετά απλό. Ωστόσο, όπως αποδείχθηκε είναι εξίσου αποδοτικό.

Τελικά καταλήγουμε πως και τα δύο μοντέλα έχουν παρόμοια συμπεριφορά και απόδοση. Ωστόσο, με βάση τα παραπάνω, το **Μοντέλο 1** υπερτερεί ελάχιστα και το θεωρώ ως βέλτιστο μεταξύ των 2.

## Σύγκριση με το μοντέλο της Εργασίας 1

Αρχικά παραθέτω τα αποτελέσματα των μετρικών του βέλτιστου που είχα υλοποιήσει στην εργασία 1. Θα το συγκρίνω με το Μοντέλο 1 αυτής της εργασίας το οποίο αποδείχθηκε και το καλύτερο εκ των δύο.




Συγκρίνοντας τα scores των μετρικών παρατηρούμε:

- Το μοντέλο της εργασίας 1 (Softmax) υπερισχύει σε σχέση με αυτό την εργασίας 2 (Neural Network) τόσο στα συνολικά f1-score/precision/recall όσο και στα επιμέρους scores ανά κλάση. Ο κύριος λόγος είναι η κλάση 1 (anti-vax) καθώς το μοντέλο Softmax επιτυγχάνει αρκετά καλύτερα αποτελέσματα.
- Και τα δύο μοντέλα έχουν ομαλή συμπεριφορά και δεν παρουσιάζουν overfit ή underfit.
- Συγκρίνοντας το μοντέλο Softmax με το μοντέλο 2 αυτής της εργασίας παρατηρούμε πως το πρώτο υπερτερεί. Τα δύο αυτά μοντέλα χρησιμοποιούν TF-IDF για την αναπαράσταση των tweets.

Από τα παραπάνω συμπεραίνουμε πως το μοντέλο της πρώτης εργασίας και συγκεκριμένα ο Softmax είναι ελάχιστα αποδοτικότερος για το συγκεκριμένο πρόβλημα.
Αυτό οφείλεται στην απλότητα του προβλήματος και των δεδομένων μας.
Ως γνωστών τα νευρωνικά δίκτυα είναι πολύ αποδοτικά για δύσκολα προβλήματα και απαιτούν μεγάλο όγκο δεδομένων για να εκπαιδευτούν. Αυτό αποδεικνύεται και στην πράξη μέσω της μελέτης μας καθώς έχοντας το σχετικά μικρό dataset εκπαίδευσης που μας δόθηκε, τα νευρωνικά δίκτυα δεν βελτίωσαν καθόλου την απόδοση σε σχέση με το «απλό» μοντέλο του Softmax Regression.