ΘΕΟΦΑΝΟΠΟΥΛΟΣ ΜΙΧΑΗΛ

111 520 18 00 053

Εργασία 3 Τεχνητή Νοημοσύνη ΙΙ

Σημαντική Σημείωση για τον εξεταστή:

Έκανα και ένα σχετικό post στο Piazza, τα μοντέλα που αποθήκευσα τοπικά βγήκαν μεγέθους 2.14gb το καθένα και για αυτόν τον λόγο δεν με άφηνε το e-class να τα ανεβάσω μέσα στο zip της εργασίας μου. Για τον λόγο αυτό θα χρειαστεί να εκτελεστεί το κατάλληλο cell από εσάς με σκοπό να εξεταστεί το μοντέλο μου. Παρόλα αυτά, τα cells για το train είναι ήδη run στο code.ipynb αρχείο μου και τα αποτελέσματα τους φαίνονται. Εάν υπάρχει κάποιο θέμα με αυτό, μπορείτε να μου στείλετε κάποιο mail ώστε με κάποιον τρόπο να σας στείλω τα μοντέλα μου. Σας ευχαριστώ εκ των προτέρων.

Για το καθάρισμα των δεδομένων χρησιμοποίησα την εξής προσέγγιση:

1. *Αρχικά κρατάω μόνο τις στήλες των rating και review, καθώς τα χρειαστούμε τη πρώτη για το training του μοντέλου και τη δεύτερη για να γίνει το prediction.*
2. *Στη συνέχεια, αντικαθιστούμε όλα τα reviews με είτε 0.0 είτε 1.0 εάν είναι <5 ή >5 αντίστοιχα, ώστε να μπορεί να γίνει το prediction.*
3. *Ύστερα αφαιρούμε από τα κείμενα των reviews όλα τα emoticons και τα σύμβολα που μπορεί να περιέχονται ώστε να υπάρχει σκέτο κείμενο. Πέραν αυτών, αφαιρούμε τα links, urls, σημεία στίξης και τους αριθμούς από το κείμενο.*
4. *Εκτελούμε τη διαδικασία του Tokenization στα reviews.*
5. *Εκτελούμε τη διαδικασία του Stemming στα reviews.*
6. *Εκτελούμε τη διαδικασία του Lemmatization στα reviews.*
7. *Αφού έχουμε τελειώσει με το καθάρισμα των δεδομένων, κρατάμε σε δυο ξεχωριστούς πίνακες τη στήλη review και rating. Η μορφή στην οποία βρίσκονται μετά τα δεδομένα μας είναι η εξής:*

```
[ ] fixedDf
```

	rating	review
0	1.0	thought quiet good movie fun watch liked best ...
1	1.0	wagon master unique film amongst john fords wo...
2	1.0	film near perfect film john ford made film mag...
3	0.0	gave 4 stars lot interesting themes many alrea...
4	1.0	movie really genuine random really hard find m...
...	...	...
45003	0.0	dont even know begin br br worth typing review...
45004	0.0	one worst movies saw 90s id often use benchmar...
45005	0.0	baldwin really stooped low make movies script ...
45006	0.0	liked watching mel gibson million dollar hotel...
45007	1.0	easily best cinematic version william faulkner...

45008 rows × 2 columns

Παρατηρούμε ότι τα reviews είναι πλήρως καθαρισμένα και έχουμε κρατήσει μόνο τις σημαντικές για την επεξεργασία μας λέξεις.

Το επόμενο βήμα είναι να χωρίσουμε τα δεδομένα μας σε training και testing set, με σκοπό να προχωρήσουμε στη δημιουργία του μοντέλου μας.

*Εάν δοθεί graders_data (**TODO:**store your data set in this variable), τότε ως training set κρατάμε ολόκληρο το data set που έχουμε και ως testing set κρατάμε το data set του grader. Διαφορετικά κάνουμε split το set σε 80-20.*

Ύστερα από τη διαδικασία του καθαρισμού των δεδομένων, δημιουργούμε για κάθε review μια αναπαράσταση από αριθμούς η οποία στη συνέχεια θα δοθεί σαν όρισμα στο νευρωνικό μας δίκτυο. Για την συγκεκριμένη αναπαράσταση, επέλεξα να χρησιμοποιήσω pre-trained word embedding vectors του Glove, καθώς υποδεικνύεται στην εκφώνηση της εργασίας.
*Επέλεξα να χρησιμοποιήσω το αρχείο **glove.42B.300d.zip** καθώς με το συγκεκριμένο λόγω του μεγέθους του παρατήρησα καλύτερα αποτελέσματα στα δεδομένα μου.*

Για τη δημιουργία των αναπαραστήσεων των reviews, χρησιμοποίησα ένα embedding layer εντός του νευρωνικού. Το embedding layer δημιουργείται από το pretrained model του glove, δηλαδή δεν είναι trainable. Το embedding layer δέχεται σαν όρισμα το review χωρισμένο σε tokens και παράγει την αναπαράσταση του η οποία προκύπτει από το σύνολο των αναπαραστάσεων για κάθε token.

Αρχικά χωρίζω το review σε tokens. Ύστερα, έχουμε μια λίστα από tokens για κάθε review και μετατρέπω τη κάθε λέξη στο αντίστοιχο index του glove dictionary για βέλτιστη απόδοση. Κάθε review λοιπόν στο τέλος αντιστοιχεί σε μια λίστα από αριθμούς ώστε να είναι διαχειρίσιμο από το μοντέλο μας.

Η αναπαράσταση που δημιουργώ για κάθε review αποτελείται από πολλά vectors, δηλαδή για κάθε review έχουμε μια διαφορετικού μεγέθους αναπαράσταση ανάλογα με τον αριθμό των tokens του.

Για την αναπαράσταση αυτή, δημιούργησα έναν DataLoader ο οποίος χωρίζει τα δεδομένα σε batches. Κάθε batch αποτελείται από reviews ίδιου μήκους. Η υλοποίηση έγινε ως εξής:

1. *Δημιούργησα έναν DataLoader για κάθε ένα από τα μήκη των reviews.*
2. *Σε κάθε DataLoader δίνω σαν input όλα τα reviews που έχουν το εκάστοτε μήκος και αυτός τα χωρίζει σε batches, χρησιμοποιώντας το batch size που έχω ορίσει.*
3. *Μέσω της συνάρτησης __iter__ ο batcher είναι iterable επιστρέφοντας με τυχαία σειρά batches κατά το training.*

Για το νευρωνικό μας δίκτυο ακολούθησα την εξής προσέγγιση:

Αρχικά ορίζω τις εξής παραμέτρους:

1. *num_epochs*
2. *batch_size*
3. *learning_rate*
4. *input_size*
5. *hidden_size*
6. *num_layers*
7. *dropout_between_layers*
8. *final_dropout*
9. *gradient_clipping*
10. *skip_connections*
11. *model_type*

Στη συνέχεια ορίζω το RNN μοντέλο μας, δηλαδή τη κλάση που αναπαριστά το νευρωνικό μας δίκτυο. Η κλάση αυτή είναι υποκλάση της nn.Module.

Κατά την αρχικοποίηση της κλάσης, ορίζω:

1. *Το Embedding Layer*
2. *Τα stacked bidirectional RNNs*
3. *Τον τύπο των stacked bidirectional RNNs*
4. *Τα Activation/Normalization functions*
5. *Το τελικό γραμμικό layer*
6. *Αρχικοποιώ τα weights των cells.*

Μετά από τον ορισμό του μοντέλου ορίζω το loss function που θα χρησιμοποιηθεί κατά την εκπαίδευση. Για το loss function επέλεξα να χρησιμοποιήσω το Cross Entropy Loss, καθώς εκτός από το ότι επισημαίνεται στην εκφώνηση είναι μια συνάρτηση που συνίσταται σε multiclass classification problems. Επίσης ορίζω τον optimizer που θα χρησιμοποιηθεί καθώς και τις αντίστοιχες παραμέτρους του. Πάλι, επέλεξα τον Adam optimizer, καθώς επισημαίνεται στην εκφώνηση.

Ύστερα δημιουργούνται οι batchers του training και του testing set.

*Για την εκπαίδευση και την αξιολόγηση του μοντέλου δημιούργησα τη συνάρτηση Validation. Η συνάρτηση αυτή, εκτός από τα απαραίτητα ορίσματα που χρειάζεται για να λειτουργήσει, δέχεται και ένα όρισμα τύπου Boolean train. Εάν δοθεί το όρισμα train ως True, τότε χρειάζεται να εκπαιδεύσουμε το μοντέλο μας πριν το αξιολογήσουμε. Διαφορετικά, σημαίνει πως έχουμε ένα ήδη εκπαιδευμένο μοντέλο. Έχω δημιουργήσει δυο cells στον κώδικα μου, ένα για κάθε περίπτωση. (**TODO:** read the text cells upon the code cells carefully, as they mention the instructions for the grader).*

Για τη διαδικασία του training:

Για κάθε ένα από τα num_epochs που δίνονται ως όρισμα της συνάρτησης:

1. *Διαγράφω τα αποθηκευμένα gradients από το προηγούμενο epoch.*
2. *Εκτελώ το μοντέλο με το τρέχον batch παίρνοντας το αντίστοιχο output για κάθε review.*
3. *Υπολογίζω το loss.*
4. *Εφαρμόζω το backpropagation.*
5. *Εφαρμόζω το gradient clipping (if gradient_clipping is True)*
6. *Ανανεώνω τα weights του μοντέλου με τη βοήθεια του optimizer.*
7. *Αποθηκεύω τα predictions.*

Για τη διαδικασία της αξιολόγησης του κάθε epoch:

1. *Εκτελώ το μοντέλο με το τρέχον batch παίρνοντας το αντίστοιχο output για κάθε review.*
2. *Υπολογίζω το loss.*
3. *Αποθηκεύω τα predictions.*
4. *Υπολογίζω και εκτυπώνω τα αποτελέσματα για το τρέχον epoch των μετρικών που αναφέρονται στην εκφώνηση.*

<u>Για τη διαδικασία της αξιολόγησης του μοντέλου:</u>

Έπειτα από την εκπαίδευση του μοντέλου εκτελούμε το μοντέλο και αυτό μας επιστρέφει τις προβλέψεις του για κάθε review.

Κατά την εκπαίδευση του μοντέλου εκτυπώνω τις καμπύλες Loss vs Epochs τόσο για το train όσο και για το test. Ύστερα εκτυπώνω τα ROC curves για κάθε μια από τις 3 κλάσεις στο ίδιο διάγραμμα. Μέσω των μετρικών που υπολογίζονται και εκτυπώνονται στο τέλος μπορούμε να μελετήσουμε αναλυτικά την απόδοση του μοντέλου αλλά και να εντοπίσουμε προβλήματα overfit και underfit.

<u>Για την διαδικασία εύρεσης του βέλτιστου μοντέλου:</u>

1) Σε πρώτη φάση υλοποιώ ένα βέλτιστο μοντέλο LSTM που αποτελείται από:

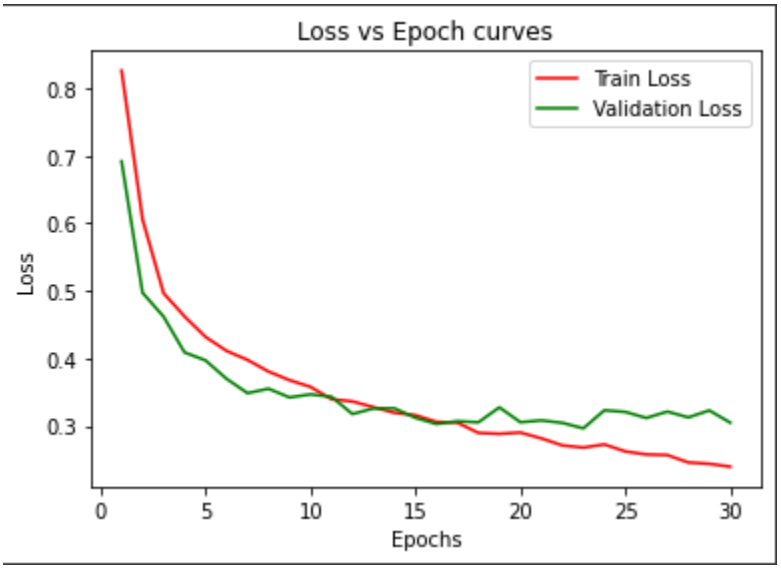
1. *num_epochs 30*
2. *batch_size 16*
3. *learning_rate 0.000125*
4. *input_size 300*
5. *hidden_size 8*
6. *num_layers 2*
7. *dropout_between_layers 0.6*
8. *final_dropout 0.5*
9. *gradient_clipping True*
10. *skip_conntections False*
11. *Model_type LSTM*

Το μοντέλο μας είναι το εξής:

```
RNN(
  (emb): Embedding(1917494, 300)
  (lstm): LSTM(300, 8, num_layers=2, batch_first=True, dropout=0.6, bidirectional=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=16, out_features=3, bias=True)
)
```

Ύστερα από το καθάρισμα των δεδομένων μας έχουμε τα εξής αποτελέσματα:

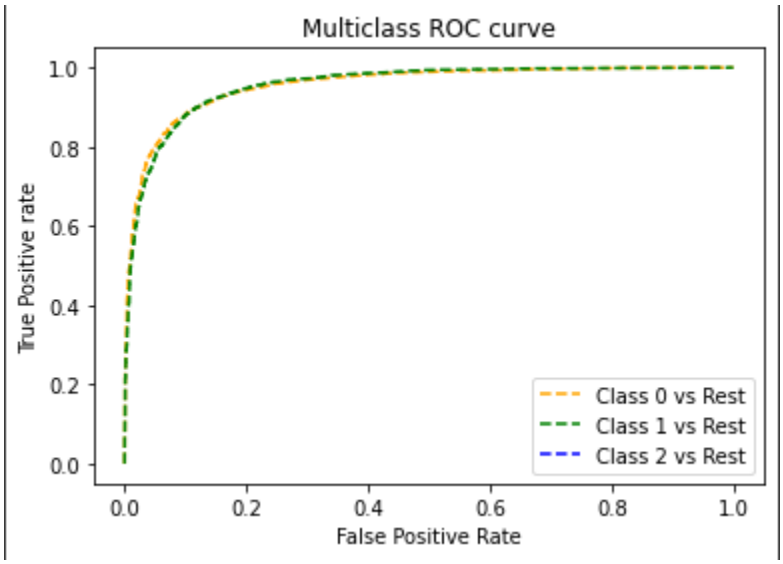
```
Epoch   0: Train Loss = 0.82585 | Validation Loss = 0.69166 | Accuracy = 66.7518 | Train-f1 = 0.5039 | Valid-F1 = 0.6648
Epoch   1: Train Loss = 0.60550 | Validation Loss = 0.49708 | Accuracy = 80.2599 | Train-f1 = 0.7062 | Valid-F1 = 0.8021
Epoch   2: Train Loss = 0.49654 | Validation Loss = 0.46182 | Accuracy = 82.1040 | Train-f1 = 0.8030 | Valid-F1 = 0.8201
Epoch   3: Train Loss = 0.46211 | Validation Loss = 0.40883 | Accuracy = 84.3812 | Train-f1 = 0.8236 | Valid-F1 = 0.8438
Epoch   4: Train Loss = 0.43174 | Validation Loss = 0.39700 | Accuracy = 85.0255 | Train-f1 = 0.8336 | Valid-F1 = 0.8503
Epoch   5: Train Loss = 0.41109 | Validation Loss = 0.36978 | Accuracy = 85.9920 | Train-f1 = 0.8448 | Valid-F1 = 0.8599
Epoch   6: Train Loss = 0.39777 | Validation Loss = 0.34853 | Accuracy = 86.3919 | Train-f1 = 0.8488 | Valid-F1 = 0.8639
Epoch   7: Train Loss = 0.38053 | Validation Loss = 0.35506 | Accuracy = 86.6807 | Train-f1 = 0.8550 | Valid-F1 = 0.8668
Epoch   8: Train Loss = 0.36777 | Validation Loss = 0.34224 | Accuracy = 87.1362 | Train-f1 = 0.8609 | Valid-F1 = 0.8714
Epoch   9: Train Loss = 0.35784 | Validation Loss = 0.34669 | Accuracy = 86.9807 | Train-f1 = 0.8645 | Valid-F1 = 0.8697
Epoch  10: Train Loss = 0.33961 | Validation Loss = 0.34317 | Accuracy = 87.4583 | Train-f1 = 0.8705 | Valid-F1 = 0.8745
Epoch  11: Train Loss = 0.33630 | Validation Loss = 0.31764 | Accuracy = 87.8471 | Train-f1 = 0.8725 | Valid-F1 = 0.8785
Epoch  12: Train Loss = 0.32786 | Validation Loss = 0.32592 | Accuracy = 87.9916 | Train-f1 = 0.8742 | Valid-F1 = 0.8799
Epoch  13: Train Loss = 0.31945 | Validation Loss = 0.32599 | Accuracy = 87.5694 | Train-f1 = 0.8765 | Valid-F1 = 0.8755
Epoch  14: Train Loss = 0.31592 | Validation Loss = 0.31202 | Accuracy = 88.1804 | Train-f1 = 0.8777 | Valid-F1 = 0.8818
Epoch  15: Train Loss = 0.30608 | Validation Loss = 0.30295 | Accuracy = 88.2471 | Train-f1 = 0.8808 | Valid-F1 = 0.8824
Epoch  16: Train Loss = 0.30489 | Validation Loss = 0.30703 | Accuracy = 88.4026 | Train-f1 = 0.8839 | Valid-F1 = 0.8840
Epoch  17: Train Loss = 0.28969 | Validation Loss = 0.30535 | Accuracy = 88.4248 | Train-f1 = 0.8881 | Valid-F1 = 0.8842
Epoch  18: Train Loss = 0.28842 | Validation Loss = 0.32748 | Accuracy = 88.2471 | Train-f1 = 0.8878 | Valid-F1 = 0.8824
Epoch  19: Train Loss = 0.29024 | Validation Loss = 0.30549 | Accuracy = 88.5803 | Train-f1 = 0.8903 | Valid-F1 = 0.8858
Epoch  20: Train Loss = 0.28160 | Validation Loss = 0.30818 | Accuracy = 88.8025 | Train-f1 = 0.8935 | Valid-F1 = 0.8880
Epoch  21: Train Loss = 0.27120 | Validation Loss = 0.30451 | Accuracy = 88.7914 | Train-f1 = 0.8942 | Valid-F1 = 0.8879
Epoch  22: Train Loss = 0.26802 | Validation Loss = 0.29646 | Accuracy = 89.0247 | Train-f1 = 0.8955 | Valid-F1 = 0.8902
Epoch  23: Train Loss = 0.27266 | Validation Loss = 0.32315 | Accuracy = 88.3248 | Train-f1 = 0.8949 | Valid-F1 = 0.8832
Epoch  24: Train Loss = 0.26231 | Validation Loss = 0.32073 | Accuracy = 88.3137 | Train-f1 = 0.8980 | Valid-F1 = 0.8830
Epoch  25: Train Loss = 0.25781 | Validation Loss = 0.31177 | Accuracy = 88.8580 | Train-f1 = 0.9001 | Valid-F1 = 0.8886
Epoch  26: Train Loss = 0.25695 | Validation Loss = 0.32123 | Accuracy = 88.7025 | Train-f1 = 0.9029 | Valid-F1 = 0.8870
Epoch  27: Train Loss = 0.24602 | Validation Loss = 0.31263 | Accuracy = 88.8802 | Train-f1 = 0.9048 | Valid-F1 = 0.8888
Epoch  28: Train Loss = 0.24393 | Validation Loss = 0.32296 | Accuracy = 88.7581 | Train-f1 = 0.9053 | Valid-F1 = 0.8876
Epoch  29: Train Loss = 0.23976 | Validation Loss = 0.30479 | Accuracy = 89.0802 | Train-f1 = 0.9080 | Valid-F1 = 0.8908
```



```
Accuracy: 89.08%
f1 score: 89.08%
Precision: 89.11%
Recall: 89.08%
              precision    recall  f1-score   support

           0       0.90      0.87      0.89      4449
           1       0.88      0.91      0.89      4553

    accuracy                           0.89      9002
   macro avg       0.89      0.89      0.89      9002
weighted avg       0.89      0.89      0.89      9002
```



Παρατηρούμε ότι το μοντέλο μας έχει μια αρκετά καλή συμπεριφορά.

Το validation loss πάντα μειώνεται, το train loss επίσης μειώνεται, τα αποτελέσματα των μετρικών είναι αρκετά καλά, τα ROC curves έχουν καλή συμπεριφορά και για τις τρεις κλάσεις και δεν έχουμε ούτε underfit ούτε overfit, καθώς τα curves βρίσκονται πολύ κοντά μεταξύ τους.

Για τα πρώτα epochs το μοντέλο τα πηγαίνει καλύτερα στο test 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.

2) Σε δεύτερη φάση υλοποιώ ένα βέλτιστο μοντέλο GRU με σκοπό να αυξήσω όσο γίνεται τα αποτελέσματα των μετρικών συναρτήσεων. Για τον σκοπό αυτό:

1. *Εφαρμόζω gradient clipping κατά την εκπαίδευση*
2. *Εφαρμόζω dropout έπειτα από το LSTM/GRU layer με dropout 0.5*
3. *num_epochs 30*
4. *batch_size 16*
5. *learning_rate 0.0001*
6. *input_size 300*
7. *hidden_size 8*
8. *num_layers 1*
9. *dropout_between_layers 0.5*
10. *final_dropout 0.25*
11. *gradient_clipping True*
12. *skip_connections True*
13. *Model_type GRU*

Χρησιμοποιώ skip connections στο μοντέλο. Για την εφαρμογή του χρειάζεται να ορίσω με διαφορετικό τρόπο τα stacked LSTM/GRU layers. Πιο συγκεκριμένα, ορίζω δυο ξεχωριστά layers θέτοντας στο καθένα number of layers 1. Το μέγεθος του hidden layer παραμένει ίδιο με πριν δηλαδή 8.

Το πρώτο layer έχει input size ίδιο με Embedding size.

Το δεύτερο layer έχει input size διπλάσιο από το output του πρώτου, δηλαδή 16.

Τέλος ορίζω ένα dropout layer ενδιάμεσα των δυο layers.

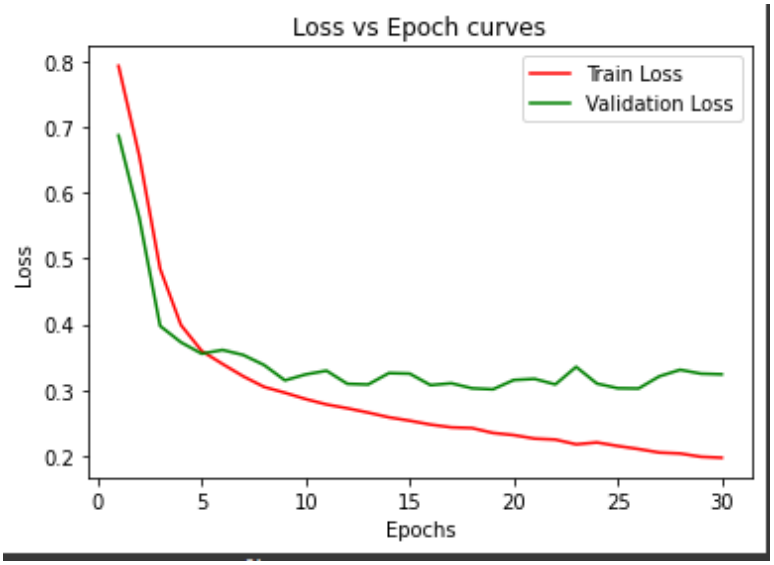
Στη συνάρτηση forward κάνω επίσης της εξής αλλαγές:

1. *Μετά από την εκτέλεση του πρώτου layer αποθηκεύω το output του και το δίνω σαν input στο δεύτερο.*
2. *Μετά από την εκτέλεση του δεύτερου layer αποθηκεύω το output του.*
3. *Συνδυάζω το output του πρώτου και του δεύτερου layer δημιουργώντας ένα νέο output μέσω του skip connection.*

Τα αποτελέσματα είναι τα εξής:

```
RNN(
  (emb): Embedding(1917494, 300)
  (gru1): GRU(300, 8, batch_first=True, bidirectional=True)
  (dropout_between_layers): Dropout(p=0.5, inplace=False)
  (gru2): GRU(16, 8, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.25, inplace=False)
  (fc): Linear(in_features=32, out_features=3, bias=True)
)
```

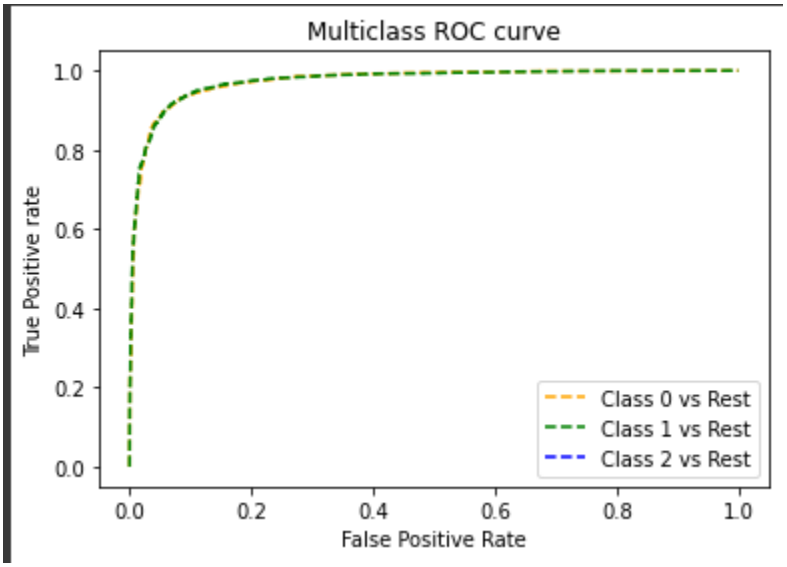
```
Epoch   0: Train Loss = 0.79242 | Validation Loss = 0.68681 | Accuracy = 61.8529 | Train-f1 = 0.5128 | Valid-F1 = 0.5981
Epoch   1: Train Loss = 0.65559 | Validation Loss = 0.56140 | Accuracy = 76.1609 | Train-f1 = 0.6215 | Valid-F1 = 0.7616
Epoch   2: Train Loss = 0.48423 | Validation Loss = 0.39740 | Accuracy = 84.2479 | Train-f1 = 0.7944 | Valid-F1 = 0.8425
Epoch   3: Train Loss = 0.39884 | Validation Loss = 0.37262 | Accuracy = 85.5254 | Train-f1 = 0.8435 | Valid-F1 = 0.8552
Epoch   4: Train Loss = 0.35939 | Validation Loss = 0.35528 | Accuracy = 86.4697 | Train-f1 = 0.8580 | Valid-F1 = 0.8647
Epoch   5: Train Loss = 0.33964 | Validation Loss = 0.36092 | Accuracy = 86.6807 | Train-f1 = 0.8655 | Valid-F1 = 0.8667
Epoch   6: Train Loss = 0.32084 | Validation Loss = 0.35328 | Accuracy = 86.9918 | Train-f1 = 0.8726 | Valid-F1 = 0.8698
Epoch   7: Train Loss = 0.30475 | Validation Loss = 0.33802 | Accuracy = 87.4917 | Train-f1 = 0.8797 | Valid-F1 = 0.8749
Epoch   8: Train Loss = 0.29591 | Validation Loss = 0.31473 | Accuracy = 88.1026 | Train-f1 = 0.8821 | Valid-F1 = 0.8810
Epoch   9: Train Loss = 0.28632 | Validation Loss = 0.32379 | Accuracy = 88.2359 | Train-f1 = 0.8860 | Valid-F1 = 0.8824
Epoch  10: Train Loss = 0.27814 | Validation Loss = 0.32938 | Accuracy = 88.1471 | Train-f1 = 0.8904 | Valid-F1 = 0.8815
Epoch  11: Train Loss = 0.27235 | Validation Loss = 0.30948 | Accuracy = 88.6581 | Train-f1 = 0.8917 | Valid-F1 = 0.8866
Epoch  12: Train Loss = 0.26582 | Validation Loss = 0.30845 | Accuracy = 88.7692 | Train-f1 = 0.8959 | Valid-F1 = 0.8877
Epoch  13: Train Loss = 0.25863 | Validation Loss = 0.32578 | Accuracy = 88.6803 | Train-f1 = 0.8972 | Valid-F1 = 0.8868
Epoch  14: Train Loss = 0.25348 | Validation Loss = 0.32498 | Accuracy = 88.6025 | Train-f1 = 0.8996 | Valid-F1 = 0.8860
Epoch  15: Train Loss = 0.24758 | Validation Loss = 0.30750 | Accuracy = 88.8247 | Train-f1 = 0.9017 | Valid-F1 = 0.8883
Epoch  16: Train Loss = 0.24337 | Validation Loss = 0.31033 | Accuracy = 88.8580 | Train-f1 = 0.9032 | Valid-F1 = 0.8886
Epoch  17: Train Loss = 0.24224 | Validation Loss = 0.30265 | Accuracy = 88.9691 | Train-f1 = 0.9056 | Valid-F1 = 0.8897
Epoch  18: Train Loss = 0.23488 | Validation Loss = 0.30114 | Accuracy = 89.1469 | Train-f1 = 0.9071 | Valid-F1 = 0.8915
Epoch  19: Train Loss = 0.23159 | Validation Loss = 0.31528 | Accuracy = 88.6914 | Train-f1 = 0.9081 | Valid-F1 = 0.8868
Epoch  20: Train Loss = 0.22627 | Validation Loss = 0.31699 | Accuracy = 88.7914 | Train-f1 = 0.9113 | Valid-F1 = 0.8879
Epoch  21: Train Loss = 0.22481 | Validation Loss = 0.30855 | Accuracy = 89.3135 | Train-f1 = 0.9120 | Valid-F1 = 0.8931
Epoch  22: Train Loss = 0.21758 | Validation Loss = 0.33524 | Accuracy = 88.3804 | Train-f1 = 0.9142 | Valid-F1 = 0.8836
Epoch  23: Train Loss = 0.22043 | Validation Loss = 0.30987 | Accuracy = 88.9913 | Train-f1 = 0.9162 | Valid-F1 = 0.8899
Epoch  24: Train Loss = 0.21505 | Validation Loss = 0.30277 | Accuracy = 89.1580 | Train-f1 = 0.9177 | Valid-F1 = 0.8916
Epoch  25: Train Loss = 0.21043 | Validation Loss = 0.30236 | Accuracy = 89.0247 | Train-f1 = 0.9178 | Valid-F1 = 0.8902
Epoch  26: Train Loss = 0.20508 | Validation Loss = 0.32088 | Accuracy = 89.1246 | Train-f1 = 0.9211 | Valid-F1 = 0.8912
Epoch  27: Train Loss = 0.20364 | Validation Loss = 0.33079 | Accuracy = 88.6025 | Train-f1 = 0.9214 | Valid-F1 = 0.8859
Epoch  28: Train Loss = 0.19864 | Validation Loss = 0.32492 | Accuracy = 88.8136 | Train-f1 = 0.9230 | Valid-F1 = 0.8880
Epoch  29: Train Loss = 0.19719 | Validation Loss = 0.32395 | Accuracy = 89.0691 | Train-f1 = 0.9259 | Valid-F1 = 0.8907
```



```
Accuracy: 89.07%
f1 score: 89.07%
Precision: 89.15%
Recall: 89.07%
              precision    recall  f1-score   support

           0       0.87      0.91      0.89      4449
           1       0.91      0.87      0.89      4553

    accuracy                           0.89      9002
   macro avg       0.89      0.89      0.89      9002
weighted avg       0.89      0.89      0.89      9002
```



Παρατηρούμε από τα αποτελέσματα των μετρικών μας ότι το συγκεκριμένο μοντέλο έχει ακόμα καλύτερες αποδόσεις από το απλό RNN με τα ίδια είδη καθαρισμένα δεδομένα. Συνεχίζουμε να έχουμε καλές Loss vs Epochs και ROC καμπύλες και δεν παρουσιάζεται πουθενά underfit ή overfit, καθώς τα curves βρίσκονται πολύ κοντά μεταξύ τους.

Για τα πρώτα epochs το μοντέλο τα πηγαίνει καλύτερα στο test 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.

Γενική παρατήρηση:

Συγκρίνοντας τις επιδόσεις των μοντέλων αυτής της εργασίας με τα αντίστοιχα μοντέλα των προηγούμενων εργασιών, παρατηρούμε ότι όσο πιο περίπλοκη γίνεται η υλοποίηση τόσο «χειρότερο» γίνεται το μοντέλο. Αυτό συμβαίνει καθώς έχουμε ένα απλό πρόβλημα με ένα μικρό σχετικά data set και για αυτό το να κάνουμε πιο περίπλοκο το μοντέλο δεν μας χρησιμεύει ιδιαίτερα από το να έχουμε ένα απλό μοντέλο.

Σας εύχομαι καλή διόρθωση!