ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Επεξεργασία Φωνής & Φυσικής Γλώσσας
3η Εργαστηριακή Άσκηση

Μία αναφορά των φοιτητών:

- Σερλή Εμμανουήλ – Αναστάσιου (Α.Μ. 03118125)
- Αβράμη Στέφανου (Α.Μ. 03121724)

Ζητούμενα Προπαρασκευής:

Ζητούμενο 1:

Για την κωδικοποίηση των labels των δεδομένων εισόδου για κάθε dataset, έγινε χρήση της συνάρτησης LabelEncoder.fit_transform() για την αντιστοίχιση των string labels σε integers. Παρακάτω ακολουθούν τα πρώτα 10 αποτελέσματα της κωδικοποίησης για τα 2 δοθέντα datasets:

Για το Semeval2017A

```
neutral 1
positive 2
positive 2
neutral 1
positive 2
neutral 1
positive 2
positive 2
neutral 1
negative 0
```

Για το MR

```
positive 1
positive 1
positive 1
positive 1
positive 1
positive 1
positive 1
positive 1
positive 1
positive 1
```

Ζητούμενο 2:

Σε αυτό το ζητούμενο πραγματοποιείται ο αναγκαίος διαχωρισμός των προτάσεων σε tokens, έτσι ώστε να μπορέσει να ακολουθήσει και η μετ'έπειτα κωδικοποίησή τους. Για τον σκοπό αυτό, έγινε χρήση της συνάρτησης word_tokenize() του πακέτου nltk_tokenize, με τα 10 πρώτα αποτελέσματα της επεξεργασίας στο training set του MR dataset να φαίνονται παρακάτω:

```
### 10 first examples of tokenization: ###

0: ['the', 'rock', 'is', 'destined', 'to', 'be', 'the', '21st', 'century', "'s", 'new', '``', 'conan', "''", 'and', 'that', 'he', "'s", 'going', 'to', 'make', 'a', 'splash', 'even', 'greater', 'than', 'arnold', 'schwarzenegger', ',', 'jean-claud', 'van', 'damme', 'or', 'steven', 'segal', '.']

1: ['the', 'gorgeously', 'elaborate', 'continuation', 'of', "''", 'the', 'lord', 'of', 'the', 'rings', "''", 'trilogy', 'is', 'so', 'huge', 'that', 'a', 'column', 'of', 'words', 'can', 'not', 'adequately', 'describe', 'co-writer/director', 'peter', 'jackson', "'s", 'expanded', 'vision', 'of', 'j', '.', 'r', '.', 'r', '.', 'tolkien', "'s", 'middle-earth', '.']

2: ['effective', 'but', 'too-tepid', 'biopic']

3: ['if', 'you', 'sometimes', 'like', 'to', 'go', 'to', 'the', 'movies', 'to', 'have', 'fun', ',', 'wasabi', 'is', 'a', 'good', 'place', 'to', 'start', '.']

4: ['emerges', 'as', 'something', 'rare', ',', 'an', 'issue', 'movie', 'that', "'s", 'so', 'honest', 'and', 'keenly', 'observed', 'that', 'it', 'does', "n't", 'feel', 'like', 'one', '.']

5: ['the', 'film', 'provides', 'some', 'great', 'insight', 'into', 'the', 'neurotic', 'mindset', 'of', 'all', 'comics', '--', 'even', 'those', 'who', 'have', 'reached', 'the', 'absolute', 'top', 'of', 'the', 'game', '.']

6: ['offers', 'that', 'rare', 'combination', 'of', 'entertainment', 'and', 'education', '.']

7: ['perhaps', 'no', 'picture', 'ever', 'made', 'has', 'more', 'literally', 'showed', 'that', 'the', 'road', 'to', 'hell', 'is', 'paved', 'with', 'good', 'intentions', '.']

8: ['steers', 'turns', 'in', 'a', 'snappy', 'screenplay', 'that', 'curls', 'at', 'the', 'edges', ';', 'it', "'s", 'so', 'clever', 'you', 'want', 'to', 'hate', 'it', '.', 'but', 'he', 'somehow', 'pulls', 'it', 'off', '.']

9: ['take', 'care', 'of', 'my', 'cat', 'offers', 'a', 'refreshingly', 'different', 'slice', 'of', 'asian', 'cinema', '.']
```

Ζητούμενο 3:

Σε αυτό το βήμα, πραγματοποιείται αντιστοίχιση των επιμέρους word tokens σε διακριτά id values, ενώ ταυτόχρονα πραγματοποιείται και zero padding στις προτάσεις με μήκος μικρότερο του μεγίστου. Έτσι, έγινε συμπλήρωση των κενών στην ρουτίνα __getitem__ της κλάσης SentenceDataset, η οποία χρησιμοποιεί το word2idx dictionary που προκύπτει από την συνάρτηση load_word_vectors(). Παρακάτω παρατίθενται 5 παραδείγματα από την άνωθι διαδικασία στο MR dataset:

```
(array([     1,   1138,     15,  10454,      5,     31,      1,   5034,
          590,     10,     51,     29,  18513,     29,      6,     13,
           19,     10,    223,      5,    160,      8,  16807,    152,
         1414,     74,   5819,   6681,      2, 400001,   1462,  43708,
           47,   4412,  26985,      3,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0]), 1, 36)
(array([     1,  78616,   5135,  10117,      4,     29,      1,   2371,
            4,      1,   6820,     29,  12305,     15,    101,   1325,
           13,      8,   3236,      4,   1375,     87,     37,  12424,
         4467, 400001,   1295,   1755,     10,   2853,   3139,      4,
         6892,      3,   1912,      3,   1912,      3,  23463,     10,
        55754,      3,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0]), 1, 42)
(array([  2038,     35, 400001,  34277,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0]), 1, 4)
(array([   84,     82,   1072,    118,      5,    243,      5,      1,   2460,
            5,     34,   2906,      2,  66408,     15,      8,    220,    242,
            5,    466,      3,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0]), 1, 21)
(array([12398,     20,    646,   2349,      2,     30,    496,   1006,     13,
           10,    101,   6082,      6,  23499,   4583,     13,     21,    261,
           71,    999,    118,     49,      3,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0]), 1, 23)
```

Ζητούμενο 4:

Σε αυτό το βήμα, συμπληρώνονται τα κενά στο models.py:EX4 για την υλοποίηση ενός embedding layer το οποίο έχει ως στόχο την αντιστοίχιση των διακριτών tokens σε μία συνεχή αναπαράσταση. Για αυτό, γίνεται αρχικοποίηση των βαρών με τα προ-εκπαιδευμένα word embeddings, και δεν έγινε η εξ’αρχής εκπαίδευσή τους. Αυτή η επιλογή έγκειται στο γεγονός ότι τα έτοιμα embeddings εκπαιδεύτηκαν σε μεγαλύτερα datasets, γεγονός που καθιστά υψηλή και γενικευμένη την ικανότητά τους να βρίσκουν συσχετίσεις μεταξύ των λέξεων.

Επιπλέον, αποφεύχθηκε και το update των βαρών κατά την διάρκεια της εκπαίδευσης, μιας και θα δημιουργούνταν φαινόμενα overfitting στα πολύ μικρότερα corpuses που διαθέτουμε κατεβασμένα στα πλαίσια της εργαστηριακής άσκησης. Έτσι, καταλήγουμε να χρησιμοποιούμε τα pretrained word embeddings ουσιαστικά αναλλοίωτα.

Ζητούμενο 5:

Μέσω συμπλήρωσης των κενών στις θέσεις models.py:EX5, έγινε η υλοποίηση των ζητούμενων output layers. Πιο αναλυτικά, το προτελευταίο layer αποτελείται από ένα Linear επίπεδο με output_hiden_size 1000, οι έξοδοι του οποίου στην συνέχεια περνούν από ReLu συναρτήσεις ενεργοποίσης, ενώ το τελευταίο layer είναι καθαρά γραμμικό και διάστασης 1000 x n_classes. Με βάση τα παραπάνω, αποφεύχθηκε η χρήση γραμμικής activation function στο προτελευταίο layer, μιας και η σειριακή τοποθέτηση γραμμικών μετασχηματισμών δεν θα μοντελοποιούσε τα non-linearities τα οποία εμφανίζονται στο πρόβλημα αναγνώρισης συναισθήματος μέσω κειμένου.

Ζητούμενο 6:

Μέσω της συμπλήρωσης των ζητούμενων κενών, υλοποιείται η forward συνάρτηση που θα χρησιμοποιηθεί τόσο για training όσο και για inference: Αρχικά, πραγματοποείται προβολή των προτάσεων εισόδου στο embedding layer και κατασκευή της αναπαράστασης της πρότασης ως τον μέσο όρο της εξόδου κάθε embedding vector. Στην συνέχεια, η εν λόγω αναπαράσταση περνά από τον μη-γραμμικό μετασχηματισμό της ReLu, πρωτού εξαχθούν τα τελικά logits μέσω του τελικού γραμμικού layer.

Αναφορικά με την conceptual σημασία της αναπαράστασης, αυτή εκφράζει την συσχέτιση της πρότασης εισόδου με την έννοια που εκφράζει κάθε embedding vector, δίνοντας ένα correlation score (εμείς λαμβάνουμε τον μέσο όρο). Ωστόσο, η λογική που ακολουθείται δεν λαμβάνει υπόψιν την σειρά των λέξεων στην πρόταση, μιας και η πρόσθεση για την εξαγωγή του είναι αντιμεταθετική ιδιότητα.

Ζητούμενο 7:

Μέσω της κλάσης DataLoader, γίνεται ένα στιγμιότυπο για κάθε κλάση. Συγκεκριμένα, όσον αφορα το μέγεθος των mini-batches, μικρά mini-batches οδηγούν σε ευκολότερη αποθήκευση των training data ανά εποχή, με τον κίνδυνο overfit λόγω του μικρού όγκου πληροφορίας. Από την άλλη, μεγάλο batch size αποτρέπει το ενδεχόμενο overfit και σύγκλιση σε κάποιο τοπικό ελάχιστο, αλλά οδηγεί σε μεγαλύτερα update steps και κατ’επέκταση σε αυξημένη πιθανότητα για exploding gradient phenomena.

Όσον αφορά την τυχαία σειρά των mini-batches, αυτή βοηθά το μοντέλο να μην εξαρτάται στην συγκεκριμένη σειρά των δεδομένων για να κάνει update των παραμέτρων του, γεγονός που οδηγεί ταυτόχρονα σε καλύτερη γενίκευση.

Ζητούμενο 8:

Σε αυτό το σημείο ορίστηκαν οι παράμετροι βελτιστοποίησης του μοντέλου. Συγκεκριμένα, επιλέχθηκε loss criterion BCEWithLogitsLoss() αν έχουμε 2 κλάσεις (στην περίπτωση του MR dataset), ενώ με 3 κλάσεις έχουμε CrossEntropyLoss(). Από optimizer, επιλέχθηκε ο Adam Optimizer με χαμηλό learning rate lr=0.001, ενώ τα βάρη του embedding layer εξαιρέθηκαν απο το parameter update.

Ζητούμενο 9:

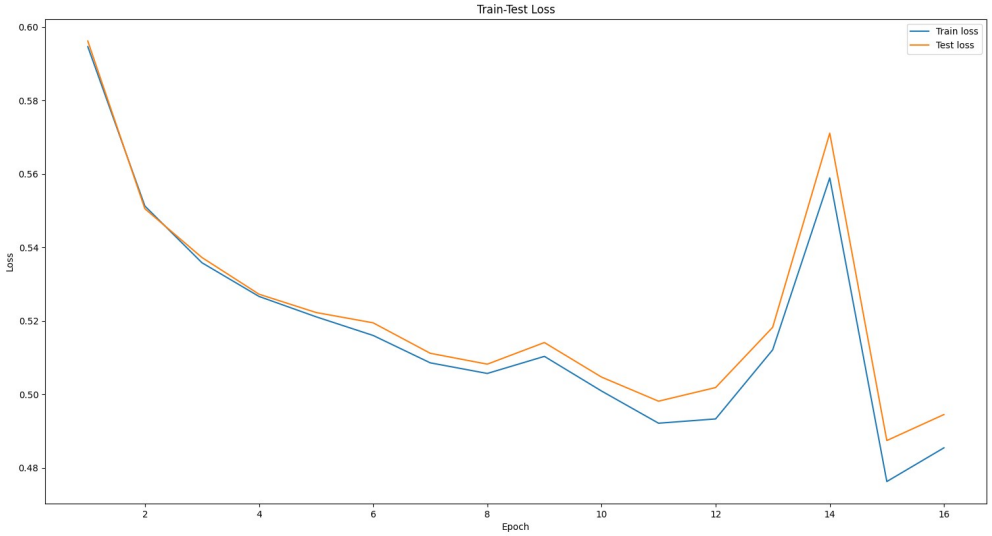
Σε αυτό το βήμα, συμπληρώθηκαν τα κενά στο training.py αρχείο, τα οποία περιλαμβάνουν την αποθήκευση των batches στην cpu ή στην gpu (ανάλογα με το επιλέγον device), το forward inference, τον υπολογισμό του loss και-κατ'επέκταση-του gradient και, τέλος, το update των βαρών.

Ζητούμενο 10:

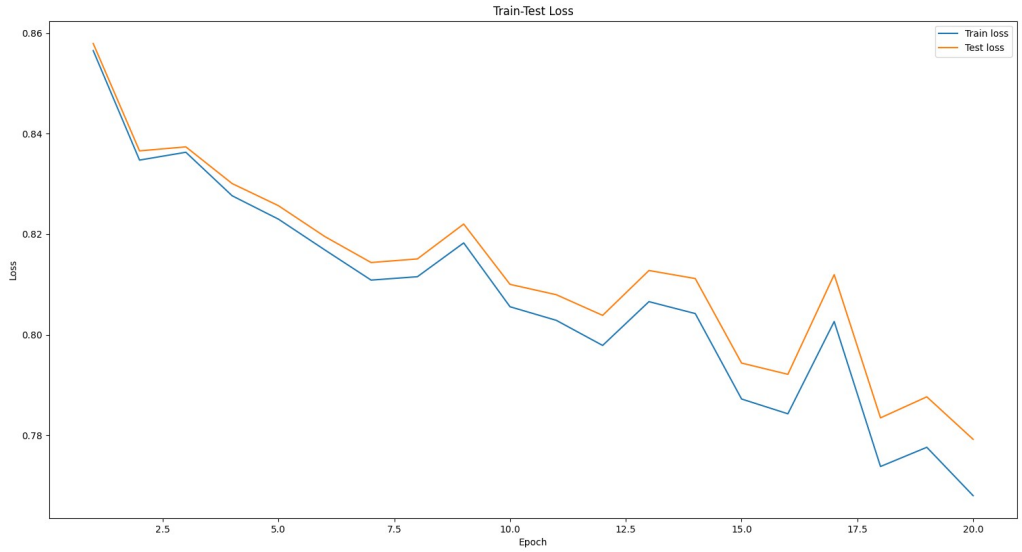
Παρακάτω ακολουθούν τα ζητούμενα metrics του μοντέλου επί του test set για καθένα από τα 2 datasets. Σημειώνεται ότι χρησιμοποιήθηκαν τα 100-d embeddings και έγινε προπόνηση για 20 εποχές:

	Accuracy	Recall	F1-Score
MR	**75.49%**	**77.03%**	**75.13%**
Semeval2017A	63.78%	62.78%	59.08%

Train-Test Loss γραφική στο MR Dataset:



Train-Test Loss γραφική στο Semeval2017A dataset:



Παρατηρούμε ότι το σχετικά απλοικό DNN μοντέλο εξάγει ήδη ικανοποιητικά αποτελέσματα και χωρίς την χρήση 300d embeddings. Τα μειωμένα scores σε οτι αφορά το 2ο dataset θα βελτιωθούν τόσο με χρήση αυξημένων embeddings όσο και με πειραματισμό με διαφορετικές αρχιτεκτονικές νευρωνικών στα επόμενα βήματα.

Ζητούμενο 11:

Πραγματοποιήθηκε το ζητούμενο prompting στο ChatGPT, έτσι ώστε να κατηγοριοποίησει 20 φράσεις από κάθε dataset με βάση την συναισθηματική τους σημασιολογία. Αρχικά στο MR dataset βρήκε 17/20 φράσεις σωστές, ενώ στο Semeval2017A οδηγήθηκε σε score 20/20.

Όταν το μοντέλο ρωτήθηκε για τον μηχανισμό που το οδήγησε στις άνωθι ταξινομήσεις, λάβαμε την κάτωθι απάντηση:

“ Let's go through the explanations for each category:

Positive:

- *520829332525441024: The phrase expresses a positive sentiment towards Leeds United and compares the feeling of a Saturday without them to a Sunday without a Sunday dinner.*

Negative:

- *522903127717330944: The phrase expresses a negative sentiment towards the new season of American Horror Story, stating that the first two seasons were good, but it went downhill afterward.*
- *522838326126989314: The phrase expresses a neutral sentiment, wondering if a joke involving Billy Joe Armstrong would get annoying over time.*
- *523082124812124161: The phrase expresses a negative sentiment towards Virgin Media for not receiving a promised code for free Netflix.*

Neutral:

- *522931511323275264: The phrase provides information about an event without expressing a clear sentiment.*
- *523099837936312321: The phrase discusses being paid overtime for decorating a haunted house without expressing a clear sentiment.*
- *521384413217959937: The phrase includes a wish of luck for a game without expressing a clear sentiment.*
- *522436979921674241: The phrase mentions a late upload from a Bears game without expressing a clear sentiment.*
- *522958573685735425: The phrase states catching up on American Horror Story and plans to watch other shows without expressing a clear sentiment.*
- *523087448264671233: The phrase mentions a personal Friday routine without expressing a clear sentiment.*
- *523073472440721408: The phrase serves as a reminder for a Friday night event without expressing a clear sentiment.*
- *519035126920273920: The phrase discusses a particular scene in a movie without expressing a clear sentiment.*
- *519145349596737536: The phrase reports an encounter with a person in a bubble without expressing a clear sentiment.“*

Με βάση τα παραπάνω κατανοούμε ότι το μοντέλο αναλύει σε λογικό επίπεδο την κάθε πρόταση πρωτού οδηγηθεί στην απάντηση του εκάστοτε ερωτήματος κάτι που εξηγεί και τα υψηλά ποσοστά επιτυχίας του.

Ζητούμενα Κυρίως Εργαστηρίου:

Στο κυρίως μέρος του εργαστηρίου και για τα ερωτήματα 1-5, έγινε χρήση των 100d embeddings καθώς και του MR dataset.

Ερώτημα 1:

Σε αυτό το ερώτημα, πραγματοποιείται τροποποίηση της εξίσωσης (2) έτσι ώστε να περιλαμβάνει τόσο το mean όσο και το max pooling. Με αυτόν τον τρόπο, δίνεται η δυνατότητα να ληφθούν υπόψιν κομμάτια του embedding μετασχηματισμού τα οποία θα χάνονταν αν κρατούσαμε μονο τον μέσο όρο του διανύσματος. Έτσι, συμπεριλαμβάνοντας το max pooling ενισχύεται η πληροφορία που μεταβιβάζεται στο επόμενο layer και με λέξεις οι οποίες διαθέτουν αυξημένη sentimental πληροφορία (π.χ. “άσχημος”, “απίστευτος” κλπ).

Ερώτημα 2:

Παρακάτω ακολουθούν τα αποτελέσματα των ζητούμενων LSTM δικτύων (με και χωρίς bidirectionality). Χρησιμοποιήθηκε early stopping με patience=5 εποχές και hidden_size = 2 * emd_dim = 2*100 = 200. Ως maximum πλήθος εποχών ορίστηκαν εκ νέου οι 20:

	Accuracy	Recall	F1-Score
Vanilla LSTM	**89.21%**	**89.48%**	**89.19%**
Bidirectional LSTM	85.93%	86.24%	85.89%

Παρατηρούμε ότι η χρήση bidirectional LSTM δεν οδηγεί σε βελτιώση των αποτελεσμάτων σε σχέση με το απλό LSTM δίκτυο, με πιθανή αιτιολογία τον περιορισμένο αριθμό εποχών προπόνησης. Σε θεωρητικό επίπεδο, η ανωτερότητα του αμφίδρομου LSTM έγκειται στο γεγονός ότι η επεξεργασία της πληροφορίας εισόδου πραγματοποιείται 2 φορές (μία απ’τη αρχή στο τέλος και μία αντίστροφα).

Ερώτημα 3:

Αντίστοιχα με το 2ο ερώτημα παρατίθενται και τα αποτελέσματα του νευρωνικού με Self-Attention Mechanism:

	Accuracy	Recall	F1-Score
Simple Self-Attention	79.12%	79.65%	79.08%

Παρατηρούμε ότι η επίδοση του attention mechanism είναι χαμηλότερη σε σχέση με αυτή των DNN και LSTM δικτύων, αν και παρατηρήθηκε σημαντική μείωση του χρόνου εκπαίδευσης για το ίδιο πλήθος εποχών.

Όσον αφορά την λειτουργία του, ο μηχανισμός attention “κοιτάει” σε ένα inference όλη την ακολουθία εισόδου. Συγκεκριμένα, δέχεται ως είσοδο μια αλληλουχία λέξεων η οποία αποτελείται από **λέξεις-queries**. Στην συνέχεια, το vector που αντιστοιχεί σε καθένα query συσχετίζεται μέσω dot-product multiplication με το word vector καθενός key, που στην περίπτωση του self-attention ως **keys** ορίζονται οι υπόλοιπες λέξεις/tokens. Το γεγονός αυτό οδηγεί σε τόσα βάρη όσα και τα keys. Μετ’έπειτα, τα εν λόγω βάρη κανονικοποιούνται και περνούν από ένα softmax activation function το οποίο οριοθετεί τις τιμές στο διάστημα [0,1]. Τέλος, πραγματοποιείται weighted sum

όλων των αρχικών word vectors a.k.a των **values** με τα κανονικοποιημένα βάρη που λαμβάνουμε στην έξοδο της softmax.

Ερώτημα 4:

Ομοίως με παραπάνω, καταγράφηκαν οι τιμές του καλύτερου Multi-Head Attention μοντέλου μας

	Accuracy	Recall	F1-Score
MultiHeadAttention	76.78%	76.96%	76.77%

Παρατηρούμε εκ νέου ότι τα αποτελέσματα είναι χειρότερα σε σχέση με αυτά του Simple Self-Attention χωρίς κάποια προφανή εξήγηση ή κάποιο δευτερεύον πλεονέκτημα, πλην ίσως της βελτιωμένης ικανότητας γενίκευσης λόγω των πολλαπλών Heads (3 εν προκειμένω) που χρησιμοποιούνται στο εν λόγω Attention Mechanism.

Ερώτημα 5:

Σε αυτό το ερώτημα, πραγματοποιείται εκπαίδευση και validation ενός TransformerEncoder NN, το οποίο διαφέρει από το MultiHeadAttention Model του προηγούμενου ερωτήματος στο γεγονός ότι ορίζουμε 3 MultiHeadAttention Blocks τα οποία στην συνέχεια περνούν από κανονικοποίηση και γραμμικοποίηση. Παρακάτω ακολουθούν τα αποτελέσματα του μοντέλου:

	Accuracy	Recall	F1-Score
TransformerEncoder	**81.02%**	**81.09%**	**81.00%**

Παρατηρούμε ότι από τα attention-based models, το συγκεκριμένο ξεπερνά τα υπόλοιπα σε απόδοση.

Ερώτημα 6:

Σε αυτό το ερώτημα, έγινε η φόρτωση των pretrained μοντέλων και η αξιολόγησή τους σε καθένα από τα 2 δοθέντα datasets:

	Accuracy	Recall	F1-Score
siebert/MR	**92.59%**	**92.59%**	**92.59%**
bert-based-uncased/MR	50.15%	50.15%	33.66%
distilbert-base-uncased/MR	89.12%	89.12%	89.12%
cardiffnlp/Semeval	72.37%	72.29%	72.22%
mrm8488/Semeval	-	-	-
oliverguhr/Semeval	-	-	-

Παρατηρούμε ότι τα καλύτερα scores προσφέρονται από τα siebert και και distilebert LLMs, αποτέλεσμα λογικό μιας και έχουν εκπαιδευθεί, έτσι ώστε να χρησιμοποιούνται σε sentiment analysis tasks.

Ερώτημα 7:

Παρακάτω ακολουθούν τα αποτελέσματα από το finetuning των pretrained LLMs για καθένα από τα 2 datasets:

LLM/dataset	Accuracy
bert-based-cased/Semeval	62.5%
siebert/MR	*crashed
distilbert-base-uncased/MR	86%
cardiffnlp/Semeval	-
mrm8488/Semeval	60%
oliverguhr/Semeval	49%

Παρατηρούμε ότι το distilbert LLM παρουσιάζει εκ νέου την καλύτερη απόδοση, γεγονός που μας οδηγεί στο συμπέρασμα ότι θα ήταν και το μοντέλο που θα χρησιμοποιούσαμε και σε μια Real-Time εφαρμογή.

Ερώτημα 8:

Ο κωδίκας που ζητήθηκε να αξιολογήσει το ChatGPT αφορούσε την υλοποίηση του Bigram Model από το δοθέν tutorial.

Όσον αφορά την εξήγηση:

“ The given Python class is a simple implementation of a bigram language model using PyTorch. Here's a brief explanation of its components and functionalities:
The class `BigramLanguageModel` extends the `nn.Module` class, indicating that it's a neural network model.

The constructor method `__init__` initializes the model's components. It defines several attributes:

- `token_embedding_table`: A lookup table (`nn.Embedding`) for token embeddings. It maps tokens to their corresponding embedding vectors.
- `position_embedding_table`: Another lookup table (`nn.Embedding`) for position embeddings. It maps positions (or time steps) to their corresponding embedding vectors.
- `blocks`: A sequence (`nn.Sequential`) of `Block` modules. Each `Block` module represents a building block of the model.
- `ln_f`: A layer normalization (`nn.LayerNorm`) applied after the blocks to normalize the outputs.
- `lm_head`: A linear layer (`nn.Linear`) that maps the output embeddings to the vocabulary size.

The `forward` method takes input `idx` (a tensor of integer indices representing the input sequence) and `targets` (optional tensor of target indices for training). It performs the forward pass of the model. Here's a summary of its steps:

- Token and position embeddings are obtained by looking up the embedding tables.

- The token and position embeddings are summed together to get the input embeddings.
- The input embeddings pass through the sequence of blocks.
- Layer normalization is applied to the block outputs.
- The final outputs are passed through the linear layer to obtain logits for each token in the vocabulary.

If `targets` is provided, the method calculates the loss using cross-entropy between the predicted logits and the target indices.

The `generate` method generates new tokens given an input sequence (`idx`) and the maximum number of new tokens to generate (`max_new_tokens`). It follows these steps:

- For the desired number of new tokens, the model repeatedly predicts the next token given the current context.
- The context is cropped to the last `block_size` tokens.
- The model produces logits for the next token.
- The logits are converted into probabilities using softmax.
- A token is sampled from the probability distribution.
- The sampled token is appended to the running sequence.
- The process continues until the desired number of new tokens is generated.

The method returns the generated sequence of indices.

Overall, this class represents a basic bigram language model that can be trained to predict the next token in a sequence and used to generate new sequences based on the learned patterns.”

Όσον αφορά τον σχολιασμό και το refactoring, λάβαμε την εξής απάντηση:

“Certainly! Here's some commentary on the quality of the code and a refactored version of the class:

Commentary on the code quality:

1. Lack of imports: The code snippet you provided does not include the necessary import statements. It seems to use PyTorch's `nn` module and other related functions and classes. Make sure to include the appropriate import statements at the beginning of the file or code snippet.
2. Missing variable definitions: The code snippet references several variables (`vocab_size`, `n_embd`, `block_size`, `n_head`, `n_layer`) without explicitly defining them. Ensure that these variables are defined with appropriate values before using them in the class.
3. Incomplete context: The code snippet refers to a `Block` class that is not provided. To fully understand the code and evaluate its quality, the implementation of the `Block` class is required.

Refactored version of the class: Based on the provided code snippet and assuming certain missing parts, here's a refactored version of the `BigramLanguageModel` class:

```
python

import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
class BigramLanguageModel(nn.Module):
    def __init__(self, vocab_size, n_embd, block_size, n_head, n_layer):
        super(BigramLanguageModel, self).__init__()
        self.token_embedding_table = nn.Embedding(vocab_size, n_embd)
        self.position_embedding_table = nn.Embedding(block_size, n_embd)
        self.blocks = nn.ModuleList([Block(n_embd, n_head) for _ in
range(n_layer)])
        self.ln_f = nn.LayerNorm(n_embd)
        self.lm_head = nn.Linear(n_embd, vocab_size)

    def forward(self, idx, targets=None):
        tok_emb = self.token_embedding_table(idx)
        pos_emb = self.position_embedding_table(torch.arange(idx.size(1),
device=idx.device))
        x = tok_emb + pos_emb
        for block in self.blocks:
            x = block(x)
        x = self.ln_f(x)
        logits = self.lm_head(x)

        loss = None
        if targets is not None:
            logits = logits.view(-1, logits.size(-1))
            targets = targets.view(-1)
            loss = F.cross_entropy(logits, targets)

        return logits, loss

    def generate(self, idx, max_new_tokens):
        for _ in range(max_new_tokens):
            idx_cond = idx[:, -block_size:]
            logits, _ = self.forward(idx_cond)
            logits = logits[:, -1, :]
            probs = F.softmax(logits, dim=-1)
            idx_next = torch.multinomial(probs, num_samples=1)
            idx = torch.cat((idx, idx_next), dim=1)

        return idx
```

In this refactored version, I made a few changes and assumptions:

1. Import statements: The necessary import statements for PyTorch and related modules are assumed to be present.
2. `Block` class: Since the `Block` class was not provided, I refactored it into a list of `Block` modules using `nn.ModuleList`. You will need to define and implement the `Block` class separately based on your requirements.
3. Variable definitions: I assumed that the missing variables (`vocab_size`, `n_embd`, `block_size`, `n_head`, `n_layer`) are provided as parameters in the constructor (`__init__`).
4. Superclass call: I updated the superclass call to `super(BigramLanguageModel, self).__init__()` to follow the recommended syntax.
5. Loss assignment: I initialized `loss` as `None` before the conditional statement for simplicity and clarity.

Please note that without the complete code and context, it's challenging to provide a fully functional refactored version. Make sure to adjust the refactored code according to your specific requirements, implement the missing `Block` class, and include the necessary import statements.”

Αρχικά, παρατηρούμε ότι ορθώς σχολιάζει την έλλειψη διάφορων κλάσεων και μεταβλητών που δεν του δόθηκαν (όπως π.χ. τα Blocks για το MultiHeadAttention) και υπάρχουν στον υπόλοιπο κώδικα. Επιπλέον, αναφορικά με την updated υλοποίηση αυτή περιλαμβάνει minor modifications που προσφέρουν καλύτερο declaration παραμέτρων και κλάσεων.