ΗΜΜΥ ΕΜΠ

Επεξεργασία Φωνής και Φυσικής Γλώσσας (2022-2023)
3η Εργαστηριακή Άσκηση

Αναγνώριση συναισθήματος σε κείμενα

Ηλιόπουλος Γεώργιος:	03118815	giliopoulos301@gmail.com
Σπανός Νικόλαος:	03118822	nickspanos23@gmail.com

Περιγραφή

Σκοπός είναι η υλοποίηση μοντέλου για επεξεργασία και κατηγοριοποίηση κειμένων, με την χρήση βαθιών νευρωνικών δικτύων (Deep Neural Networks - DNN) και προ-εκπαιδευμένων μεγάλων γλωσσικών μοντέλων (Large Language Models).

Προπαρασκευή Εργαστηρίου

1 Προεπεξεργασία Δεδομένων

1.1 Κωδικοποίηση Επισημειώσεων (Labels)

Για το σκοπό της προπαρασκευής τα βήματα εκτελέστηκαν πάνω στο MR dataset (Sentence Polarity Dataset). Αρχικά για να μπορούμε να χρησιμοποιήσουμε τα labels του datset χρειάζεται να τα μετατρέψουμε σε αριθμητική μορφή. Για αυτό το σκοπό χρησιμοποιήθηκε ο LabelEncoder του scikit-learn και εφαρμόστηκε fit και transform πάνω σε όλα τα labels του dataset. Παρατίθενται τα πρώτα 10 labels και η κωδικοποίησή τους.

```
First 10 labels before encoding are:
['positive', 'positive', 'positive', 'positive', 'positive',
'positive', 'positive', 'positive', 'positive', 'positive']

Corresponding encoded labels are:
[1 1 1 1 1 1 1 1 1 1]
```

1.2 Λεκτική Ανάλυση (Tokenization)

Για να μπορέσουμε να κωδικοποιήσουμε τα δεδομένα σε μορφή την οποία μπορεί να επεξεργαστεί το δίκτυο χρειάζεται πρώτα να χωρίσουμε τις προτάσεις σε ακολουθίες από tokens. Χρησιμοποιούμε το RegexpTokenizer του nltk.tokenize για να εκτελέσουμε το tokenization καθώς δε λαμβάνει υπόψιν τα σημεία στίξης. Τα πρώτα δύο παραδείγματα εκπαίδευσης είναι τα ακόλουθα.

Before tokenization:

```
'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 .'
```

```
'the gorgeously elaborate continuation of " the lord of the rings "
trilogy is so huge that a column of words cannot adequately
describe co-writer/director peter jackson\'s expanded vision of j .
r . r . tolkien\'s middle-earth .'
```

After tokenization:

```
['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']
```

```
['the', 'gorgeously', 'elaborate', 'continuation', 'of', 'the',
'lord', 'of', 'the', 'rings', 'trilogy', 'is', 'so', 'huge',
'that', 'a', 'column', 'of', 'words', 'cannot', 'adequately',
'describe', 'co', 'writer', 'director', 'peter', 'jackson', 's',
'expanded', 'vision', 'of', 'j', 'r', 'r', 'tolkien', 's',
'middle', 'earth']
```

1.3 Κωδικοποίηση Παραδειγμάτων (Λέξεων)

Τέλος χρειάζεται να μετατρέψουμε τα tokens σε αριθμούς με βάσει τα embeddings που χρησιμοποιούμε. Η συνάρτηση load_word_vectors που μας δίνεται δέχεται ως όρισμα τα embeddings που χρησιμοποιούμε και επιστρέφει ένα dictionary με τα id κάθε λέξης. Για να έχουν όλα τα διανύσματα το ίδιο μήκος κάνουμε zero padding στο μήκος της μεγαλύτερης ακολουθίας του dataset. Τα 2 πρώτα παραδείγματα φαίνονται στη συνέχεια.

After encoding:

```
(array([    1,  1138,    15, 10454,     5,    31,     1,  5034,   590,
         1535,    51, 18513,     6,    13,    19,  1535,   223,     5,
          160,     8, 16807,   152,  1414,    74,  5819,  6681,  2394,
        93270,  1462, 43708,    47,  4412, 26985,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]), 1, 33)
```

```
(array([    1, 78616,  5135, 10117,     4,     1,  2371,     4,     1,
         6820, 12305,    15,   101,  1325,    13,     8,  3236,     4,
         1375,  1120, 12424,  4467,  1645,  1542,   370,  1295,  1755,
         1535,  2853,  3139,     4,  6892,  1912,  1912, 23463,  1535,
          700,  1829,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]), 1, 38)
```

2 Μοντέλο

2.1 Embedding Layer

Ορίζουμε το embedding layer βάσει των διαστάσεων της δισδιάστατης μήτρας βαρών με την έτοιμη υλοποίηση της pytorch. Ύστερα αρχικοποιούμε τα βάρη από τα προεκπαιδευμένα word embeddings και τα παγώνουμε θέτοντας το requires grad με False.

Τα προεκπαιδευμένα word embeddings έχουν δημιουργηθεί από μεγάλα dataset οπότε μας παρέχουν μία καλή αρχικοποίηση για να αποφύγουμε το overfitting κατά την εκπαίδευση πάνω στο δικό μας dataset. Παράλληλα δε θέλουμε να εκπαιδεύετε γιατί θέλουμε η προβολή των λέξεων στο embedding space να παραμένει σταθερή και να μην αλλάζει.

2.2 Output Layer(s)

Δημιουργούμε ένα hidden layer με 32 units και μη γραμμική συνάρτηση ενεργοποίησης ReLu. Οι μη γραμμικές συναρτήσεις ενεργοποίησης χρησιμοποιούνται για να μπορεί να γίνει σωστά η γενίκευση πάνω στα δεδομένα και η ανάπτυξη περίπλοκων σχέσεων για να λάβουμε την επιθυμητή έξοδο. Η χρήση πολλών γραμμικών συναρτήσεων θα μας δώσει και πάλι ένα γραμμικό αποτέλεσμα και έτσι δεν εξυπηρετεί το δίκτυο για ένα τόσο περίπλοκο πρόβλημα.

2.3 Forward pass

Υλοποιούμε τους απαραίτητους μετασχηματισμούς και περνάμε τα δεδομένα στο forward pass του δικτύου.

Η αναπαράσταση κέντρου βάρους μας δίνει μία γενική ιδέα για το που βρίσκεται το διάνυσμα της πρότασης στο χώρο και αντίστοιχα για το περιεχόμενό του. Χρησιμοποιώντας αυτή τη μετατροπή για όλες τις προτάσεις μπορούμε εν τέλει να τις χωρίσουμε στις κλάσεις του dataset (positive, negative). Ωστόσο λαμβάνοντας το μέσο όρο δε λαμβάνονται υπόψιν λεπτομέρειες που διαθέτουν τα διανύσματα και μπορεί να επισκιάζονται από ισχυρές έννοιες. Επίσης πολλές φορές μπορεί η αναπαράσταση να είναι ίδια αλλά λόγω των συμφραζόμενων η ερμηνεία να είναι διαφορετική.

3 Διαδικασία Εκπαίδευση

3.1 Φόρτωση Παραδειγμάτων (DataLoaders)

Στην εκπαίδευση νευρωνικών δικτύων χρησιμοποιούμε mini batches για να λαμβάνει το δίκτυο αρκετή πληροφορία πριν ανανεώσει τα βάρη. Μικρά mini batches μπορεί να οδηγήσει στο να γίνονται πολλές ανανεώσεις και το δίκτυο να γίνει overfit, ενώ με μεγάλα mini batches δεν ανανεώνεται συχνά η πληροφορία και το μοντέλο αδυνατεί να δημιουργήσει σχέσεις. Συνήθως αναζητείται μία σχέση κατάλληλη για το μοντέλο που μπορεί να συγκλίνει γρήγορα και να δημιουργεί τις σωστές σχέσεις.

Το shuffling γίνεται γιατί δε θέλουμε το μοντέλο να βλέπει με την ίδια σειρά τα δεδομένα γιατί μπορεί να αναπτύξει σχέσεις βάσει της σειράς και να μην γενικεύσει σωστά και να κάνει overfit. Με το shuffling αποφεύγουμε τέτοιες περιπτώσεις.

Τα αποτελέσματα για τα δύο dataset και για τρεις διαφορετικούς αριθμούς διαστάσεων συνοψίζονται στους παρακάτω πίνακες.

	glove.6B.50d	**glove.6B.100d**	**glove.6B.300d**
MR	0.71	0.72	**0.75**
Semeval2017A	0.57	0.59	**0.6**

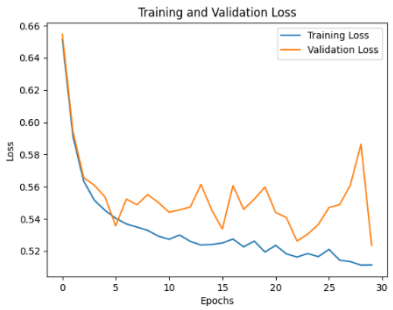
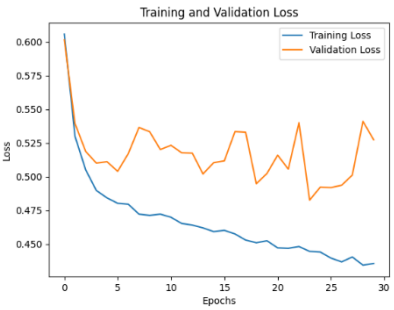
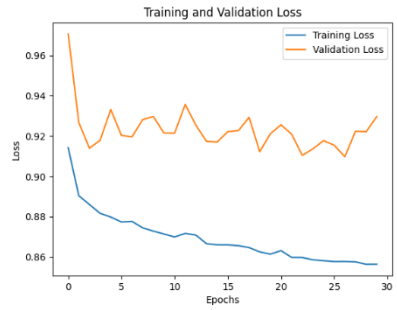
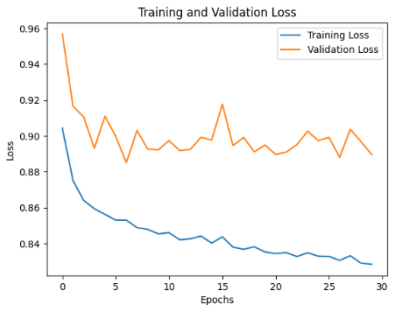
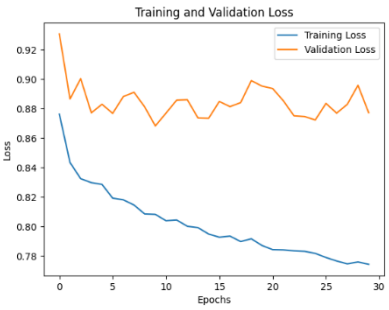
Πίνακας 1: Test set Accuracy

	glove.6B.50d	glove.6B.100d	glove.6B.300d
MR	0.71	0.72	**0.75**
Semeval2017A	0.54	0.57	**0.58**

Πίνακας 2: Test set Recall

	glove.6B.50d	glove.6B.100d	glove.6B.300d
MR	0.71	0.72	**0.75**
Semeval2017A	0.54	0.57	**0.58**

Πίνακας 3: Test set F1 score

	glove.6B.50d	glove.6B.100d	glove.6B.300d
MR			
Semeval2017A			

Πίνακας 4: Losses

Με την αύξηση των διαστάσεων των embeddings μπορούμε να κωδικοποιήσουμε παραπάνω πληροφορία για τα δεδομένα με αποτέλεσμα να έχουμε καλύτερη απόδοση του μοντέλου αλλά η εκπαίδευση είναι πιο χρονοβόρα.

4 Κατηγοριοποίηση με χρήση LLM (ChatGPT)

Στη συνέχεια κάνουμε την κατηγοριοποίηση με τη χρήση του ChatGPT ενός μεγάλου γλωσσικού μοντέλου. Από καθένα από τα δύο dataset παίρνουμε 20 κείμενα από κάθε κατηγορία και ζητάμε από το ChatGPT να τα κατηγοριοποιήσει.

Για το **MR Dataset** ανάμεσα σε 40 παραδείγματα το ChatGPT έκανε λάθος μόνο ένα παράδειγμα οδηγώντας σε accuracy 0.975. Μόλις τελείωσε την κατηγοριοποίηση έβγαλε το μήνυμα:

«Σημειώστε ότι αυτές οι προβλέψεις δημιουργούνται με βάση το περιεχόμενο των παρεχόμενων κειμένων, αλλά ενδέχεται να μην ευθυγραμμίζονται απόλυτα με τις ανθρώπινες κρίσεις συναισθήματος, καθώς η ανάλυση συναισθήματος μπορεί να είναι υποκειμενική.»

Το λάθος έγινε στη πρόταση «effective but too-tepid biopic» και όταν ρωτήθηκε ποιος ο λόγος που έκανε λάθος αυτό απάντησε:

«Η ανάλυση συναισθήματος μπορεί να είναι ένα δύσκολο έργο, και σε αυτή την περίπτωση, φαίνεται ότι το μοντέλο μπορεί να επικεντρώθηκε περισσότερο στον όρο «tepid», ο οποίος έχει αρνητική χροιά, αντί να εξετάσει το συνολικό θετικό συναίσθημα του κειμένου.»

Για το **Semeval2017A** dataset ανάμεσα σε 60 παραδείγματα έκανε λάθος σε 11, οδηγώντας σε accuracy 0.82. Αυτή τη φορά βέβαια έπρεπε να γίνει κατηγοριοποίηση σε 3 κατηγορίες και όχι σε 2.

Κάνοντας χρήση κατάλληλων prompts μπορέσαμε να αντλήσουμε πληροφορίες σχετικά με το πως γίνεται η κατηγοριοποίηση.

«Οι προβλέψεις που έγιναν στην προηγούμενη απάντηση βασίστηκαν στις περιορισμένες πληροφορίες που παρέχονται στα κείμενα και δεν προέκυψαν από ολοκληρωμένη ανάλυση ή εκπαίδευση στην ανάλυση συναισθήματος. Σκοπός τους ήταν να χρησιμεύσουν ως επίδειξη με βάση τις δεδομένες πληροφορίες. Εάν έχετε ένα σύνολο δεδομένων με ετικέτες ή πιο συγκεκριμένες απαιτήσεις, μπορώ να σας βοηθήσω περαιτέρω στην εκπαίδευση ενός μοντέλου ανάλυσης συναισθήματος ή στη χρήση υφιστάμενων μοντέλων για την παροχή ακριβέστερων προβλέψεων.»

«Ορισμένες λέξεις μπορεί να είναι σημαντικές για την πρόβλεψη του συναισθήματος. Αυτές οι λέξεις, που συχνά αναφέρονται ως "δείκτες συναισθήματος" ή "λέξεις συναισθήματος", φέρουν συναισθήματα ή συναισθηματικές συνδηλώσεις που μπορούν να δώσουν ενδείξεις για το συνολικό συναίσθημα ενός κειμένου. Ακολουθούν μερικά παραδείγματα:

- Love, Great, Amazing, Wonderful, and Excellent είναι όλοι θετικοί δείκτες συναισθήματος.
- Hate, Terrible, Awful, Disappointing, and Horrible είναι όλοι αρνητικοί δείκτες συναισθήματος.
- Fact, Information, Data, Report, Analysis, Study, Neutral, Average, and Standard είναι όλοι ουδέτεροι δείκτες συναισθήματος. »

Κύριο μέρος εργαστηρίου

Για τα ερωτήματα 1-5 χρησιμοποιήσαμε το MR Dataset. Για το τρέξιμο του training.py πρώτο argument είναι τα dimensions των embeddings και δεύτερο argument ο τύπος μοντέλου από τα εξής: DNN, LSTM, SA (SimpleSelfAttention), MA (MultiHeadAttention), TA (Transformer Encoder Model)

Ερώτημα 1 – Αλλαγή αναπαράστασης

Τώρα που παίρνουμε την αναπαράσταση κάθε πρότασης ως τη συνένωση των mean pooling και του max pooling αναμένουμε καλύτερα αποτελέσματα αφού και τα δύο pooling έχουν υψηλή σημασία. Το mean pooling εξάγει ως αναπαράσταση μίας πρότασης τον μέσο όρο τον λέξεων, δηλαδή αν οι περισσότερες λέξεις είναι κοντά σε μία έννοια συμπεραίνεται πως η πρόταση είναι κοντά σε αυτή την έννοια. Όμως ενδέχεται να υπάρχουν και πολύ ισχυρές αναπαραστάσεις όπως το «δεν» οι οποίες πρέπει να ληφθούν υπόψιν. Έτσι παίρνοντας και το max pooling μπορούμε να συμπεριλάβουμε στην τελική αναπαράσταση εκτός από το μέσο όρο και ισχυρές έννοιες, κερδίζοντας παραπάνω πληροφορία.

Ερώτημα 2 – κωδικοποίηση με LSTM

Για την υλοποίηση του LSTM αξιοποιούμε early stopping (patience 5 εποχών) για να αποφύγουμε τυχόν overfit του μοντέλου. Η αρχιτεκτονική του μοντέλου αποτελείται από hidden size 100 και 1 επίπεδο και για την εκπαίδευση batch size 128 και max εποχές 50.

Στους πίνακες 5, 6 και 7 φαίνονται τα αποτελέσματα για μονόδρομο και αμφίδρομο LSTM για το MR Dataset.

	glove.6B.50d	**glove.6B.100d**	**glove.6B.300d**
Vanilla LSTM	0.74	0.77	0.78
Bidirectional LSTM	0.77	0.79	0.77

Πίνακας 5: Test set Accuracy

	glove.6B.50d	**glove.6B.100d**	**glove.6B.300d**
Vanilla LSTM	0.74	0.77	0.78
Bidirectional LSTM	0.77	0.79	0.77

Πίνακας 6: Test set F1 score

	glove.6B.50d	**glove.6B.100d**	**glove.6B.300d**
Vanilla LSTM	0.74	0.77	0.78
Bidirectional LSTM	0.77	0.79	0.77

Πίνακας 7: Test set Recall

Παρατηρούμε πως η χρήση bidirectional LSTM σε γενικές γραμμές βελτιώνει την απόδοση καθώς το μοντέλο διαβάζει την είσοδο και από την ανάποδη μεριά αποκτώντας έτσι παραπάνω πληροφορία. Εξαίρεση αποτελεί η περίπτωση με embedding με 300 διαστάσεις στην οποία γίνεται πολύ γρήγορα overfit λόγω του μεγάλου μεγέθους της εισόδου και του μικρού αριθμού εποχών.

Ερώτημα 3 – Μηχανισμός Attention

Στον πίνακα 8 φαίνονται οι μετρικές χρησιμοποιώντας Attention Μηχανισμό.

	glove.6B.50d	**glove.6B.100d**	**glove.6B.300d**
Test set Accuracy	0.72	0.729	0.737
Test set Recall	0.72	0.729	0.737
Test set F1 score	0.72	0.726	0.737

Πίνακας 8: Μετρικές με μηχανισμό Attention

Η απόδοση είναι χαμηλότερη σε σχέση με το LSTM όμως ο μηχανισμός attention έχει το πλεονέκτημα πως την ίδια στιγμή κοιτάει όλο το sequence και αξιοποιεί πληροφορία από όλες τις λέξεις και η εκπαίδευση είναι σημαντικά γρηγορότερη.

Ο τρόπος με τον οποίο λειτουργεί είναι δεχόμενος κάποιες λέξεις queries συγκρίνει με τα keys (στη συγκεκριμένη περίπτωση με όλες τις υπόλοιπες λέξεις του sequence) και μας επιστρέφει τις λέξεις οι οποίες είναι πιο κοντά και δίνουν context στα queries μέσα από ένα σταθμισμένο μέσο της σύγκρισης των πιθανοτήτων σύγκρισης με τα values. Στη συγκεκριμένη περίπτωση χρησιμοποιούμε scaled dot product attention ο οποίος βασίζεται στο παρακάτω τύπο.

$$\mathrm{Z} = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

Οι πίνακες Q, K και V προκύπτουν κάνοντας προβολή της εισόδου με τους αντίστοιχους πίνακες προβολής π.χ. $Q = X \cdot W^Q$. Οι πίνακες αυτοί αποτελούν τις παραμέτρους του μοντέλου και ορίζονται κατά την εκπαίδευση. Παράλληλα ο λόγος που κάνουμε scale είναι επειδή όσο μεγαλώνει η διάσταση του πίνακα των keys, δηλαδή το μέγεθος των sequence, η διακύμανση του QK^T αυξάνεται, η SoftMax κορέζεται και η παράγωγος μικραίνει. Χρησιμοποιώντας γενικότερα attention αποφεύγουμε το πρόβλημα του vanishing gradient και δεν υπάρχει ανάγκη να διατηρούμε το length του sequence.

Ο πίνακας position embeddings μας βοηθά να έχουμε πληροφορία ανάλογα με τη θέση της λέξης στο sequence καθώς συνηθίζεται λέξεις που βρίσκονται κοντά να δίνουν παραπάνω context στη πληροφορία ή η ίδια η θέση να δίνει πληροφορία (αρχή και τέλος της πρότασης) και αυτό είναι εξαιρετικά σημαντικό επειδή το μοντέλο του transformer δομικά δε λαμβάνει υπόψιν τη σειρά των λέξεων στο input sequence.

Ερώτημα 4 – Μηχανισμός MultiHead Attention

Το MultiHead Attention λειτουργεί ανάλογα με το simple self-attention απλά έχουμε πολλούς πίνακες W^Q, W^K, W^V (τα λεγόμενα heads) και κάνουμε concatenate τα αποτελέσματα της κάθε διαδικασίας και πολλαπλασιάζουμε με ένα output matrix για να πάρουμε το τελικό αποτέλεσμα.

Στον πίνακα 9 φαίνονται οι μετρικές χρησιμοποιώντας MultiHead Attention μηχανισμό για 5 heads και χωρίς dropout.

	glove.6B.50d	**glove.6B.100d**	**glove.6B.300d**
Test set Accuracy	0.71	0.734	0.758
Test set Recall	0.71	0.734	0.758
Test set F1 score	0.70	0.734	0.756

Πίνακας 9: Μετρικές με μηχανισμό MultiHead Attention

Παρατηρούμε πως με το MultiHead Attention έχουμε καλύτερες επιδόσεις σε σύγκριση με το simple self-attention.

Ερώτημα 5 – Transformer-Encoder

Ο encoder είναι στην ουσία διαδοχικές διαδικασίες MultiHead attention. Κάθε block διαθέτει μέσα ένα MultiHead attention μηχανισμό με έναν συγκεκριμένο αριθμό heads και αυτά τοποθετούνται διαδοχικά και το αποτέλεσμα περνά από ένα normalization layer. Στην αρχική δημοσίευση του transformer χρησιμοποιήθηκε ένας encoder 6 επιπέδων που κάθε block αποτελείται από δύο MultiHead Attention μηχανισμούς όπου ο ένας κάνει cross attention με έναν δεύτερο encoder και ένα feed forward connected δίκτυο.

Στον πίνακα 10 φαίνονται τα αποτελέσματα.

	glove.6B.50d	**glove.6B.100d**	**glove.6B.300d**
Test set Accuracy	0.727	0.743	0.755
Test set Recall	0.727	0.743	0.755
Test set F1 score	0.726	0.741	0.753

Πίνακας 10: Μετρικές για Transformer-Encoder

Σε σύγκριση με το μηχανισμό MultiHead ttention του ερωτήματος 4 παρατηρούμε πως για embeddings χαμηλότερων διαστάσεων έχουμε καλύτερα αποτελέσματα ενώ για υψηλότερων διαστάσεων παρόμοια αποτελέσματα.

Ερώτημα 6 – Pre-Trained Transformers

Στον πίνακα 11 φαίνονται οι μετρικές για 3 διαφορετικά προεκπαιδευμένα μοντέλα για το MR dataset και στον πίνακα 12 για το Semeval Dataset.

	Test set Accuracy	Test set Recall	Test set F1-Score
siebert/sentiment-roberta-large-english	0.93	0.93	0.93
distilbert-base-uncased-finetuned-sst-2-english	0.89	0.89	0.89
bert-base-uncased	0.5	0.5	0.33

Πίνακας 11: Μετρικές για προεκπαιδευμένα μοντέλα (MR dataset)

Τα μοντέλα Siebert και Distilbert είναι λογικό να έχουν καλύτερες επιδόσεις γιατί είναι ήδη fine tuned για sentiment analysis.

	Test set Accuracy	Test set Recall	Test set F1-Score
mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis	0.51	0.4	0.36
cardiffnlp/twitter-roberta-base-sentiment	0.72	0.72	0.72
oliverguhr/german-sentiment-bert	0.51	0.43	0.43

Πίνακας 12: Μετρικές για προεκπαιδευμένα μοντέλα (Semeval dataset)

Ερώτημα 7 – Fine Tuning

Στον πίνακα 13 φαίνονται οι μετρικές για 3 διαφορετικά προεκπαιδευμένα μοντέλα για το MR dataset και στον πίνακα 14 για το Semeval Dataset μετά από fine tuning.

	Test set Accuracy
siebert/sentiment-roberta-large-english	*[1]
distilbert-base-uncased-finetuned-sst-2-english	0.85
bert-base-uncased	0.81

Πίνακας 13: Accuracy για προεκπαιδευμένα μοντέλα (MR dataset) μετά από fine tuning

	Test set Accuracy
mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis	0.62

[1] Το μοντέλο δεν χωράει στη μνήμη και για αυτό δεν μπορούμε να το κάνουμε fine tune. Ωστόσο το Siebert μοντέλο είναι ήδη fine tuned οπότε η απόδοση δεν θα βελτιωνόταν και είναι πιθανό να έκανε overfit.

cardiffnlp/twitter-roberta-base-sentiment	0.65
oliverguhr/german-sentiment-bert	0.46

Πίνακας 14: Accuracy για προεκπαιδευμένα μοντέλα (Semeval dataset) μετά από fine tuning

Η μεγαλύτερη απόδοση παρατηρείται στο Bert μοντέλο, το οποίο δεν ήταν fine tuned για sentiment analysis και η απόδοσή του αυξήθηκε κατά 30%. Μοντέλα τα οποία ήταν ήδη fine tuned για sentiment analysis παρουσίασαν overfit καθώς το training loss μειώθηκε δραματικά με το validation loss να αυξάνεται. Παρόλα αυτά συμπεραίνουμε πως το fine tuning είναι μία σημαντική διαδικασία όταν θέλουμε να συγκεκριμενοποιήσουμε τη λειτουργία ενός LLM σε μία εφαρμογή.

Ερώτημα 8 – ChatGPT

Για τις ανάγκες του ερωτήματος ζητήθηκε από το ChatGPT να εξηγήσει τον κώδικα «Let's build GPT: from scratch, in code, spelled out». Η ερώτηση που δόθηκε στο ChatGPT και η απάντησή του είναι τα ακόλουθα.

> *I have the below code "…". Can you **explain** in a paragraph what it does?*
> The provided code is a Python implementation of a language model based on the GPT (Generative Pre-trained Transformer) architecture. The model is trained on a dataset of text, specifically the "tinyshakespeare" dataset, which contains a collection of Shakespearean plays. The code begins by downloading and reading the dataset, then proceeds to preprocess the text by encoding characters into numerical values. It creates mappings between characters and integers, enabling the conversion of strings to lists of integers and vice versa. The dataset is then encoded into a torch. Tensor for efficient processing. The code further splits the data into training and validation sets. It defines a class called "BigramLanguageModel" that represents the GPT model, with methods for forward pass, generation of new text, and calculation of loss. The model utilizes self-attention and matrix multiplication techniques to capture dependencies between tokens in the text. The code also includes a training loop with optimization using the AdamW optimizer. Overall, the code builds a language model capable of generating new text based on the learned patterns from the Shakespeare dataset.

Σε γενικές γραμμές το ChatGPT μπόρεσε να εξηγήσει τη λειτουργία του κώδικα σε ικανοποιητικό βαθμό.

> *Can you now **evaluate** the code in 1 paragraph?*
> The code evaluation demonstrates a well-structured and concise implementation of a GPT model for text generation using PyTorch. It efficiently handles data preprocessing, including vocabulary creation and encoding, and effectively trains the model using the AdamW optimizer and cross-entropy loss. The code's clear explanations and illustrations of self-attention and layer normalization concepts enhance understanding.

Overall, the code provides a valuable resource for learning and implementing GPT models, showcasing its effectiveness in generating coherent and context-aware text.

> *What about the code's **modularity**?*

> The code exhibits a high degree of modularity, with well-defined functions and classes that encapsulate specific functionalities. Each component, such as data preprocessing, model architecture, training, and evaluation, is neatly organized into separate modules, allowing for easy reuse and modification. The code's modular design enhances readability, maintainability, and extensibility, enabling developers to add new features or adapt it for different applications with minimal effort. This modularity promotes code reusability and facilitates collaborative development, making it a valuable asset for building upon or integrating into other projects.

> *Can you rewrite some parts of it (**refactoring**)?*

> Sure! Here's a refactored version of the code:

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

torch.manual_seed(1337)

class LanguageModel(nn.Module):
    def __init__(self, vocab_size, n_embd, n_head, n_layer, dropout):
        super(LanguageModel, self).__init__()
        self.token_embedding_table = nn.Embedding(vocab_size, n_embd)
        self.transformer = nn.Transformer(n_embd, n_head, n_layer, dropout)
        self.linear = nn.Linear(n_embd, vocab_size)

    def forward(self, idx, targets=None):
        embeddings = self.token_embedding_table(idx)
        logits = self.transformer(embeddings)
        logits = self.linear(logits)

        if targets is None:
            loss = None
        else:
            loss = F.cross_entropy(logits.view(-1, logits.size(-1)),
targets.view(-1))
        return logits, loss

    def generate(self, idx, max_new_tokens):
        embeddings = self.token_embedding_table(idx)
        generated = self.transformer.generate(embeddings, max_new_tokens)
        return generated

vocab_size = len(chars)
model = LanguageModel(vocab_size, n_embd=64, n_head=4, n_layer=4,
dropout=0.0)
model.to(device)

optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)

def train(model, optimizer, max_iters, eval_interval):
```

```
    for step in range(max_iters):
        model.train()
        optimizer.zero_grad()

        X, Y = get_batch('train')
        logits, loss = model(X, Y)
        loss.backward()
        optimizer.step()

        if step % eval_interval == 0:
            model.eval()
            train_loss = estimate_loss('train')
            val_loss = estimate_loss('val')
            print(f"Step: {step}, Train Loss: {train_loss:.4f}, Val Loss:
{val_loss:.4f}")

train(model, optimizer, max_iters, eval_interval)

start_token = torch.tensor([stoi['<START>']],
dtype=torch.long).unsqueeze(0).to(device)
generated_sequence = model.generate(start_token, max_new_tokens=100)
generated_text = decode(generated_sequence[0].tolist())

print("Generated Text:")
print(generated_text)
```

This refactored code uses a LanguageModel class to encapsulate the GPT model. It separates the model architecture, training loop, and generation into separate functions and methods for better organization. It also uses the torch.optim.AdamW optimizer for training. The training loop includes periodic evaluation of the train and validation losses. Finally, it generates a text sample using the trained model.
Note: Since I don't have access to the estimate_loss function, I removed it from the code. You can incorporate it back into the refactored code as per your requirement.

Πράγματι το ChatGPT κατάφερε να αναδιαμορφώσει τον κώδικα και να τον κάνει πιο οργανωμένο.