ΔΠΜΣ Γλωσσική Τεχνολογία **Εξάμηνο Γ’**

Βαθιά Μάθηση με Εφαρμογές σε Γλωσσικά Δεδομένα (Μ909)

Assignment 2 & 3

Δήμητρα Κοντοέ (lt1200028)

Εισαγωγή

Σκοπός της εργασίας που συνοδεύει η παρούσα αναφορά ήταν η ανάπτυξη συστημάτων deep learning για την επίλυση ενός προβλήματος sentiment analysis, με τη χρήση των δεδομένων που περιλαμβάνει το Large Movie Review Dataset. Πιο αναλυτικά, κληθήκαμε να αξιοποιήσουμε 50.000 κριτικές για ταινίες (από το IMDB) η καθεμία από τις οποίες ήταν χαρακτηρισμένη με την ετικέτα “negative” ή “positive”, ανάλογα με το περιεχόμενό της, για την εκπαίδευση και αξιολόγηση ενός RNN μοντέλου (με χρήση είτε LSTM είτε GRU cell), ενός CNN και ενός προεκπαιδευμένου Bert μοντέλου (DistilBERT). Όσον αφορά τα δύο πρώτα μοντέλα, κληθήκαμε να πειραματιστούμε με τη χρήση προεκπαιδευμένων Glove embeddings. Θα θέλαμε να σημειώσουμε ότι αντιμετωπίσαμε την ανάπτυξη αυτών των μοντέλων ως μία προσπάθεια καλύτερης κατανόησης της λειτουργίας αυτού του τύπου των δικτύων καθώς και των υπερπαραμέτρων τους. Παρακάτω θα παρουσιάσουμε λεπτομερώς τη λογική της υλοποίησής τους, τα αποτελέσματα που παρήγαγαν καθώς και τα συμπεράσματα που αντλήσαμε.

Δεδομένα

Με γνώμονα την εξαγωγή συγκρίσιμων αποτελεσμάτων, αποφασίσαμε να εφαρμόσουμε την ίδια διαδικασία preprocessing και χωρισμού των δεδομένων για όλα τα μοντέλα, καταλήγοντας σε 3 διακριτά σετ training, validation και testing. Με βάση τη σχετική βιβλιογραφία στην οποία ανατρέξαμε, ως προς το preprocessing που προτείνεται για τα κειμενικά δεδομένα που αξιοποιούνται για την ανάπτυξη deep learning μοντέλων, εφαρμόσαμε τα παρακάτω βήματα. Σημειώνουμε ότι η επιλογή αυτή έγινε λαμβάνοντας υπόψη ότι για τη χρήση προεκπαιδευμένων embeddings η προεπεξεργασία των δεδομένων θα πρέπει να είναι ανάλογη με εκείνη που ακολουθήθηκε στα δεδομένα εκπαίδευσης των embeddings.

- Αφαίρεση των ειδικών χαρακτήρων, των html tags, της στίξης, των λέξεων μοναδικού χαρακτήρα και των πολλαπλών κενών, με σκοπό την ελαχιστοποίηση του θορύβου στα δεδομένα μας.
- Αντικατάσταση των συντετμημένων γραμματικών τύπων με τους αντίστοιχους αναλυτικούς για την ενιαιοποίηση του κειμένου.
- Αντικατάσταση των αριθμητικών με το σύμβολο #, καθώς η τεχνική αυτή έχει ακολουθηθεί στα κείμενα με τα οποία έχουν εκπαιδευτεί τα Glove embeddings.

Δεν εφαρμόστηκαν άλλες τεχνικές προεπεξεργασίας, όπως η αφαίρεση stopwords, το stemming ή το lemmatization, εφόσον αυτές δεν προτείνονται για deep learning NLP μοντέλα, στα οποία σημαντικό ρόλο παίζει η βέλτιστη αναπαράσταση του context.

Όσον αφορά τον χωρισμό των δεδομένων μας, από το σύνολο των 50.000 reviews, ένα υποσύνολο 40.999 αποτέλεσε το training dataset, 4500 το validation dataset και 4501 το testing dataset, φροντίζοντας σε όλα τα υποσύνολα η κατανομή των δειγμάτων των δύο κλάσεων (negative-positive) να είναι ίδια (stratified split). Τέλος, εφαρμόζοντας έναν έλεγχο όσον αφορά το μήκος των κριτικών του dataset, δείξαμε ότι το ελάχιστο πλήθος λέξεων που απαιτείται για συμπεριληφθούν όλες οι λέξεις του 95% των κριτικών είναι 575.

Μοντέλα

Μοντέλο 1: Bidirectional stacked RNN (LSTM cell) χωρίς προεκπαιδευμένα Glove emdeddings

Όσον αφορά την επεξεργασία των δεδομένων, χρησιμοποιήσαμε τη βιβλιοθήκη Torchtext του Pytorch, και συγκεκριμένα τα αντικείμενα Field και LabelField, για τα γλωσσικά δεδομένα και τα labels τους αντίστοιχα. Ορίσαμε τις παραμέτρους του Field με τέτοιο τρόπο ώστε να προσαρμόσουμε τα δεδομένα κατάλληλα ώστε να δοθούν ως input στο δίκτυό μας ('tokenize', 'lower=True' 'include_lengths=True'). Θα αναφερθούμε συγκεκριμένα στη χρήση της παραμέτρου 'include_lengths', μέσω της οποίας επιστρέφεται το μήκος της κάθε ακολουθίας του input (κάθε κριτικής). Με βάση τη θεωρία, τα RNN μοντέλα είναι ικανά να παίρνουν εισόδους διαφορετικού μεγέθους, αρκεί βέβαια εντός του ίδιου batch (για τις περιπτώσεις που τα δεδομένα δίνονται με τέτοιον τρόπο) το μέγεθος να είναι σταθερό. Όσον αφορά τη φόρτωση των δεδομένων, η Torchtext παρέχει την κλάση TabularDataset, η οποία διαβάζει δεδομένα της μορφής CSV, TSV ή JSON. Έτσι φορτώσαμε τα προεπεξεργασμένα δεδομένα μας μέσω της TabularDataset χρησιμοποιώντας τα αρχικοποιημένα αντικείμενα Field και LabelField. Κατόπιν, δημιουργήσαμε το λεξικό για τα training δεδομένα μας. Δηλαδή η κάθε γνωστή μοναδική λέξη των δεδομένων μας αντιστοιχήθηκε με ένα διάνυσμα 100 διαστάσεων. Αποφασίσαμε να περιορίσουμε το μέγεθος του λεξικού μας σε 20.000 μοναδικές λέξεις, για να μην αντιμετωπίσουμε προβλήματα με τη διαθεσιμότητα μνήμης GPU. Κατόπιν, χρησιμοποιήσαμε τον BucketIterator, τόσο για τα training όσο και για τα validation δεδομένα, για να έρθουν στη μορφή που είναι απαραίτητη για την τροφοδότησή τους στο δίκτυο. Ο συγκεκριμένος iterator διατάσσει τα δεδομένα με τρόπο ώστε οι ακολουθίες με παρόμοιο μήκος να ομαδοποιούνται στο ίδιο batch, και μάλιστα με φθίνουσα σειρά (από τη μεγαλύτερη στη μικρότερη), κάτι που αφενός μειώνει το padding (και άρα τα μηδενικά, δηλαδή ο θόρυβος) και αφετέρου προσδίδει μεγαλύτερη ταχύτητα στους υπολογισμούς.

Όσον αφορά την κλάση Sentiment_Classifier_LSTM (bidirectional RNN με LSTM cell) που αναπτύξαμε αναφέρουμε ότι ο τύπος αυτός των δικτύων είναι ικανός να μοντελοποιεί καλύτερα τις σχέσεις μεταξύ ακολουθιακών δεδομένων, και μάλιστα, ειδικά στην περίπτωση των LSTM, εκτεταμένων ακολουθιών. Στην περίπτωσή μας δηλαδή, για κάθε πρόταση η έξοδος του δικτύου έχει παραχθεί με βάση το context κάθε λέξης, και μάλιστα προσθετικά και από τις δύο κατευθύνσεις (bidirectional). Το Input του δικτύου είναι ένας tensor της μορφής sequence length, batch size, input dimension. Το batch size ορίζεται ήδη στον BucketIterator, το sequence length ορίζεται ως ο αριθμός των tokens κάθε ακολουθίας και το input dimension είναι η διάσταση των διανυσμάτων (100, στη δική μας περίπτωση). Η διάσταση του hidden layer ορίζεται ως εξής: 2 (num of directions)* num of layers, batch size, hidden size. Αρχικά ορίζουμε το embedding layer, μέσω του οποίου αντιστοιχίζεται το μέγεθος του λεξικού μας με ένα dense vector. Κατόπιν ορίζουμε το LSTM layer μέσω του οποίου ο πίνακας των embeddings μεταφέρεται στον χώρο του hidden layer. Και τέλος ορίζουμε το fully connected layer που καταλήγει στα αποτελέσματα για τις 2 κλάσεις μας (πιθανότητες). Ορίζουμε τα δύο πρώτα vectors (unk και pad) του πίνακα των βαρών των embeddings σε μηδέν. Στο forward step χρησιμοποιούμε την pack_padded_sequence στο output του embedding layer, μέσω της οποίας επιστρέφονται μικρότερα batches από το ήδη υπάρχον batch, μειώνοντας έτσι τους χρόνους που απαιτούνται για τους υπολογισμούς. Το τελευταίο hidden layer (forward and backward concatenated), το context vector, που θεωρούμε ότι αποτελεί την αναπαράσταση ολόκληρης την ακολουθίας εισόδου, είναι εκείνο που δίνεται στο feed-forward layer για την εξαγωγή των αποτελεσμάτων.

Optimizer: Adam

Loss Function: Binary Cross Entropy Loss with logits

Υπερπαράμετροι:

BATCH_SIZE = 128

MAX_VOCAB_SIZE (input dimension) = 20000

EMBEDDING_DIM = 100

OUTPUT_DIM = 1

HIDDEN_DIM = 128

N_LAYERS = 2 (stacked)

BIDIRECTIONAL = True

DROPOUT = 0.2

LEARNING_RATE = 0,02

NUM_EPOCHS = 5

Μοντέλο 2: Bidirectional stacked RNN (LSTM cell) με προεκπαιδευμένα Glove emdeddings

Η διαφορά σε σχέση με το μοντέλο 1 αφορά μόνο το embedding layer του δικτύου. Κατά τη δημιουργία του λεξικού μας, μέσω του αντικειμένου TEXT, της Torchtext, η κάθε μοναδική λέξη αντιστοιχίζεται με το αντίστοιχο διάνυσμα διάστασης 100 του Glove. Προκειμένου να χρησιμοποιήσουμε τον πίνακα βαρών των προεκπαιδευμένων embeddings τον αντιγράφουμε στο πρώτο layer του δικτύου μας και τον χρησιμοποιούμε χωρίς να γίνεται ενημέρωση (δηλαδή με παγωμένες τις παραμέτρους του).

Μοντέλο 3: CNN χωρίς προεκπαιδευμένα Glove emdeddings

Όσον αφορά την επεξεργασία των δεδομένων, σημειώνουμε ότι ακολουθήθηκαν τα ίδια βήματα με το μοντέλο 1, με τη διαφορά ότι για το αντικείμενο Field επιλέχθηκε η παράμετρος batch_first = True, ώστε τα δεδομένα εισόδου να έρθουν στη μορφή που απαιτούν τα συνελικτικά δίκτυα ([batch size, emb dim, sent len]). Ο τύπος αυτός νευρωνικών δικτύων αποτελείται από ένα πλήθος convolutional layers με το output τους καταλήγει σε ένα (ή περισσότερα) linear layers. Τα convolutional layers χρησιμοποιούν φίλτρα (kernels) διαφορετικών διαστάσεων, τα οποία εφαρμόζονται ταυτόχρονα στο κείμενο λειτουργώντας κατά κάποιον τρόπο σαν feature extractors των n-grams , με το καθένα από αυτά να εξάγει διαφορετική αναπαράσταση του κειμένου. Και εδώ, το πρώτο βήμα είναι η μετατροπή των λέξεων σε dense διανύσματα (word embeddings), διάστασης 100, στην περίπτωσή μας, για την τροφοδότησή τους στο δίκτυο. Για το μοντέλο μας χρησιμοποιήσαμε 3 100άδες από φίλτρα διαστάσεων [3*embedding_dim, 4*embedding_dim, 5*embedding_dim], με τη λογική ότι κοιτάζουν εμφανίσεις διαφορετικών 3-grams, 4-grams και 5-grams που σχετίζονται με την ανάλυση γνώμης σε κριτικές ταινιών. Το επόμενο βήμα στο μοντέλο μας είναι η χρήση pooling (συγκεκριμένα max pooling) στην έξοδο των convolutional layers, μέσω του οποίου λαμβάνεται η μέγιστη τιμή από κάθε διάσταση. Εδώ η λογική είναι ότι η μέγιστη τιμή αντιστοιχεί στο πιο σημαντικό χαρακτηριστικό για τον καθορισμό της γνώμης που εκφράζεται μέσω μιας κριτικής, που με τη σειρά του εξάγεται από το πιο σημαντικό n-gram της κάθε κριτικής. Εφόσον το μοντέλο μας χρησιμοποιεί 100 φίλτρα 3 διαφορετικών διαστάσεων, εξάγονται 300 διαφορετικά n-grams που θεωρούνται σημαντικά. Αυτά γίνονται concatenate σε ένα διάνυσμα που δίνεται ως είσοδος σε ένα linear layer το οποίο και θα παράξει τις προβλέψεις. Σημειώνουμε εδώ ότι τα pooling layers διαχειρίζονται προτάσεις διαφορετικού μήκους. Η διάσταση της εξόδου του κάθε convolutional layer εξαρτάται από τη διάσταση της εισόδου του και το κάθε batch περιλαμβάνει κριτικές διαφορετικού μεγέθους. Χωρίς το max pooling layer η είσοδος στο linear layer θα εξαρτιόταν από το μήκος της πρότασης εισόδου. Μια επιλογή για να επιλυθεί κάτι τέτοιο θα ήταν όλες οι προτάσεις να έρχονταν στο ίδιου μήκος μέσω trimming/padding, ωστόσο μέσω του max pooling layer εξασφαλίζεται ότι η είσοδος στο linear layer θα είναι διάστασης ίσης με το πλήθος των φίλτρων. Βέβαια, αναφέρουμε ότι για προτάσεις μικρότερες από το μέγεθος του μεγαλύτερου φίλτρου (5*100, στην περίπτωσή μας) χρειάζεται να εφαρμοστεί padding.

Optimizer: Adam

Loss Function: Binary Cross Entropy Loss with logits

Υπερπαράμετροι:

BATCH_SIZE = 128

MAX_VOCAB_SIZE (input dimension) = 20000

EMBEDDING_DIM = 100

N_FILTERS = 100

FILTER_SIZES = [3,4,5]

OUTPUT_DIM = 1

DROPOUT = 0.2

LEARNING_RATE = 0,02

NUM_EPOCHS = 5

Μοντέλο 4: CNN με προεκπαιδευμένα Glove emdeddings

Για την ανάπτυξη ενός CNN μοντέλου με προεκπαιδευμένα embeddings ακολουθήσαμε ακριβώς τα ίδια βήματα με το μοντέλο 2, αντιγράφοντας τον σχετικό πίνακα βαρών τους και χρησιμοποιώντας τα χωρίς ενημέρωση κατά το βήμα του back propagation.

Μοντέλο 5: Distilbert-base-uncased finetuning (for sequence classification)

Για τις ανάγκες τις ανάπτυξης του μοντέλου αυτού, μέσω της μεθόδου πtransfer learning, ακολουθήσαμε διαφορετική επεξεργασία των δεδομένων μας, βάσει των απαιτήσεων ενός Bert μοντέλου. Τα δεδομένα εισόδου μας θα έπρεπε να έρθουν στην ίδια μορφή με τα δεδομένα που χρησιμοποιήθηκαν κατά το pretraining του μοντέλου. Έτσι χρησιμοποιήσαμε τον αντίστοιχο tokenizer που είναι συμβατός με το προεκπαιδευμένο μοντέλο, με παραμέτρους: padding=True, truncation=True, max_length=512, ώστε οι ακολουθίες εισόδου να έχουν το ίδιο μήκος (512 tokens). Μετά την εφαρμογή του tokenization, δημιουργείται ένα dictionary με 3 keys και τα αντίστοιχα values τους για κάθε σύνολο δεδομένων: input_ids, token_type_ids, attention_mask. Για το fine tuning του μοντέλου αποφασίσαμε να χρησιμοποιήσουμε το αντικείμενο Trainer που διαθέτει η βιβλιοθήκη transformers, με βάση τις απαιτήσεις του οποίου για τα δεδομένα θα πρέπει να δημιουργηθούν iterators της κλάση torch.utils.data.Dataset. Έτσι δημιουργούμε μια υποκλάση της torch Dataset με μεθόδους την __getitem__ που επιστρέφει ένα dictionary για τα values του κάθε κειμένου για το κάθε batch, και την __len__method που επιστρέφει το μήκος των δεδομένων εισόδου. Κατόπιν, ορίζουμε τις υπερπαραμέτρους της εκπαίδευσης (διατηρώντας πολλές από τις default τιμές) μέσω της κλάσης TrainingArguments, οι οποίες τελικά δίνονται στο αντικείμενο Trainer που υλοποιεί την εκπαίδευση (fine-tuning) του μοντέλου μας. Σημειώνουμε εδώ ότι για το fine-tuning του μοντέλου χρησιμοποιήσαμε το σύνολο του πλήθους των προεκπαιδευμένων layers από encoders του DistilBert-base-uncased που παράγουν τα contextual embeddings καθώς και την αντίστοιχη κεφαλή για sequence classification με ενημέρωση των βαρών τους από τα δεδομένα εκπαίδευσης (δηλ. χωρίς να παγώσουμε κάποιες παραμέτρους).

Υπερπαράμετροι:

```
args = TrainingArguments(
    f"/content/{model_name}-finetuned-imdb",
    evaluation_strategy="epoch",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    seed=0,
    weight_decay=0.01,
    save_strategy= "epoch",
    load_best_model_at_end=True)
```

Αποτελέσματα και σύγκριση

Για την αξιολόγηση κάθε μοντέλου βασιστήκαμε στις τιμές του loss στο validation dataset ανά εποχή εκπαίδευσης, σε συνάρτηση με αντίστοιχες τιμές στο training dataset. Έτσι, κάθε φορά αποθηκεύαμε το μοντέλο με το χαμηλότερο loss στο validation set, θεωρώντας το ως το τελικό μοντέλο για την κάθε υλοποίηση και το χρησιμοποιήσαμε για αξιολόγηση στο test set. Στον παρακάτω πίνακα παραθέτουμε τα σχετικά αποτελέσματα ως προς τη μετρική Macro F1 (εφόσον βέβαια το μοντέλο μας αφορούσε binary classification και μάλιστα με χρήση balanced dataset, ως προς το πλήθος των δειγμάτων κάθε κλάσης, θα μπορούσε κάλλιστα να ληφθεί υπόψη το accuracy, το οποίο έτσι κι αλλιώς δεν παρουσίαζε κάποια απόκλιση σε σχέση με το F1)

Μοντέλο	**Macro F1**
Μοντέλο 1 (LSTM embedding layer)	**90%**
Μοντέλο 2 (LSTM pretrained Glove)	**87%**
Μοντέλο 3 (CNN embedding layer)	**87%**
Μοντέλο 4 (CNN pretrained Glove)	**84%**
Μοντέλο 5 (Fine-tuned distilBert-base-uncased)	**92%**

Με βάση τα παραπάνω, διαπιστώνουμε ότι το fine-tuned μοντέλο πετυχαίνει την καλύτερη απόδοση, και μάλιστα με ελάχιστη παραμετροποίηση, κάτι που επιβεβαιώνει τη διατεινόμενη υπεροχή των state of the art transformers μοντέλων, και των δυνατοτήτων των μεθόδων transfer learning που αυτά προσφέρουν. Επιδιώκοντας μια γενική σύγκριση των μοντέλων transformers με τα RNNs και τα CNNs θα λέγαμε ότι μέσω του μηχανισμού multi-head self attention καταφέρνουν να συνδυάσουν τα πλεονεκτήματα που πρόσφεραν τα προηγούμενα μοντέλα (παράλληλη επεξεργασία της εισόδου και μοντελοποίηση εξαρτήσεων μεγάλου βάθους) αναβαθμίζοντάς τα μέσω της δυναμικής αναπαράστασης των λέξεων ως προς τη συντακτική και σημασιολογική πληροφορία τους, χάρη στα contextual embeddings. Σε αντίθεση με αυτά, τα προεκπαιδευμένα embeddings που χρησιμοποιούνται στο LSTM και στο CNN που αναπτύξαμε, χαρακτηρίζονται ως στατικά, καθώς η κάθε ίδια λέξη αντιστοιχίζεται με ένα μοναδικό διάνυσμα, ανεξαρτήτως του συγκειμένου στο οποίο μπορεί να εμφανιστεί. Από αυτό συμπεραίνουμε ότι embeddings τύπου Glove δεν αποτελούν πλούσιες σημασιολογικές αναπαραστάσεις τέτοιες που να απαιτούνται για τις ανάγκες του word-sense disambiguation. Όσον αφορά τη σύγκριση των αποτελεσμάτων στα LSTM και CNN με τη χρήση ή μη προεκπαιδευμένων Glove embeddings, διαπιστώνουμε ότι και στις δύο περιπτώσεις, τα μοντέλα χωρίς τα προεκπαιδευμένα embeddings επιστρέφουν καλύτερο σκορ κατά 3%. Σημειώνουμε εδώ ότι θεωρούμε κάτι τέτοιο αναμενόμενο, καθώς τα προεκπαιδευμένα embeddings χρησιμοποιήθηκαν με παγωμένα τα βάρη τους, και άρα δεν προσαρμόστηκαν στο συγκεκριμένο task, που αφορούσε sentiment classification. Αντίθετα, το embedding layer που εκπαιδεύτηκε ως μέρος του μοντέλου (CNN και LSTM) πάνω στα task-specific δεδομένα εκπαίδευσης ήταν σε θέση να παράξει ορθότερες αναπαραστάσεις για τις ανάγκες του sentiment-analysis, και άρα καλύτερα αποτελέσματα.

***Η παρούσα αναφορά συνοδεύεται από 3 python notebooks (CNN_Sentiment.ipynb, LSTM_Sentiment.ipynd, DistilBERT_Sentiment.ipynb) με την υλοποίηση των μοντέλων που αναφέρθηκαν παραπάνω και το Large_Movie_Review_Dataset_preprocessing_n_splitting.py αρχείο με τον κώδικα για το preprocessing και splitting του dataset στα data_train.csv, data_dev.csv και data_test.csv. Σημειώνουμε ότι για την εκτέλεση των πειραμάτων μας χρησιμοποιήθηκε το περιβάλλον Google Collab και αξιοποιήθηκαν οι GPUs που διαθέτει.**