ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΠΠΣ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ ΜΕΣΟΛΟΓΓΙ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΤΕΧΝΙΚΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΤΗΝ ΑΝΑΛΥΣΗ ΣΥΝΑΙΣΘΗΜΑΤΩΝ ΣΕ ΚΕΙΜΕΝΟ

Κωνσταντίνος Μπάστας

Μεσολόγγι 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΠΠΣ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ ΜΕΣΟΛΟΓΓΙ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΤΕΧΝΙΚΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΤΗΝ ΑΝΑΛΥΣΗ ΣΥΝΑΙΣΘΗΜΑΤΩΝ ΣΕ ΚΕΙΜΕΝΟ

Κωνσταντίνος Μπάστας

Επιβλέπων καθηγήτρια

Μαρία Ρήγκου

Μεσολόγγι 2022

UNIVERSITY OF PATRAS

SCHOOL OF ECONOMICS & BUSINESS

DEPARTMENT OF MANAGEMENT SCIENCE AND TECHNOLOGY

FORMER DEPARTMENT OF BUSINESS ADMINISTRATION AT MESSOLONGHI

THESIS

MACHINE LEARNING TECHNIQUES FOR SENTIMENT ANALYSIS IN TEXT

Konstantinos Bastas

Messolonghi 2022

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας του Πανεπιστημίου Πατρών δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

ΠΕΡΙΛΗΨΗ

Στο πλαίσιο του ολοένα και περισσότερο δικτυωμένου κόσμου, η Μηχανική Μάθηση έχει αποδείξει ότι μπορεί λύσει περίπλοκα προβλήματα με αποδοτικό τρόπο σε ευρύ πεδίο εφαρμογών. Σε αυτήν την εργασία θα επικεντρωθούμε στην ανάπτυξη αποδοτικότερων μοντέλων Μηχανικής Μάθησης στην Ανάλυση Συναισθήματος σε κείμενο, ένας τομέας της τεχνικής Νοημοσύνης που προσπαθεί να εξάγει υποκειμενικές πληροφορίες, όπως απόψεις ή στάσεις βάσει γλωσσολογικών χαρακτηριστικών σε ένα κείμενο.

Στο πρώτο κεφάλαιο γίνεται αναφορά στην Μηχανική Μάθηση, στην ανάλυση συναισθήματος σε κείμενο καθώς και στις κύριες κατηγορίες της. Τα επόμενα κεφάλαια έχουν μορφή οδηγού και παρουσιάζονται μοντέλα Μηχανικής Μάθησης που είναι ευρέως τα πιο γνωστά για την αποδοτικότητα τους στον τομέα της Ανάλυσης συναισθήματος και τα οποία υλοποιούνται με την γλώσσα προγραμματισμού Python και την βιβλιοθήκη Pytorch.

Τέλος, χρησιμοποιούμε την cloud υπηρεσία Google Colaboratory για την εκπαίδευση των μοντέλων ενώ αναπτύχθηκε ιστοσελίδα που αποτυπώνετε η εργασία.

ABSTRACT

In an increasingly networked world, Machine Learning has proven to solve complex problems in an efficient manner and in a wide range of applications. In this thesis we will focus on the development of more efficient Machine Learning models in Sentiment Analysis in text, an area of Artificial Intelligence that focuses to extract subjective information, such as point of views or attitudes based on linguistic features in a text.

The first chapter refers to Machine Learning, sentiment analysis and its main categories. The following chapters are described in the form of a guide and present Machine Learning models that are best known for their efficiency in the field of Sentiment Analysis and are implemented with the Python programming language and the Pytorch library.

Finally, the use of Google Colaboratory cloud service is described within the training of the models, while developing a website that captures the work.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΕΡΙΛΗΨΗ 6

ABSTRACT 6

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ 8

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ 11

ΑΠΟΔΟΣΗ ΟΡΩΝ 12

ΕΙΣΑΓΩΓΗ 14

1 Κατηγορίες και Εφαρμογές της Ανάλυσης Συναισθήματος 18

1.1 Επεξεργασία Φυσικής Γλώσσας 18

1.2 Περιγραφή Ανάλυσης συναισθήματος 18

1.2.1 Κύριες κατηγορίες μελέτης Ανάλυσης Συναισθήματος 19

2 Μηχανική Μάθηση και Νευρωνικά Δίκτυα 21

2.1 Μηχανική μάθηση 21

2.1.1 Επιβλεπόμενη Μάθηση (Supervised Learning) 22

2.1.2 Unsupervised Learning (Μη Επιβλεπόμενη Μάθηση) 24

2.2 Βαθιά Εκμάθηση και Νευρωνικά δίκτυα 25

2.3 Νευρωνικά Δίκτυα 26

2.3.1 Στρώματα (layers) 27

2.3.2 Συνάρτηση απώλειας (loss function) και Βελτιστοποιητής (optimizer) 28

3 Ανάλυση συναισθήματος με χρήση Επαναλαμβανόμενων νευρωνικών δικτύων και την βιβλιοθήκη PyTorch. 29

3.1 Επαναλαμβανόμενα Νευρωνικά Δίκτυα 29

3.2 Υλοποίηση μοντέλου Ανάλυσης συναισθήματος με Επαναλαμβανόμενα Νευρωνικά Δίκτυα 32

3.2.1 Χτίζοντας το μοντέλο 38

3.3 Εκπαιδεύοντας το μοντέλο 41

4 Επαναλαμβανόμενα Νευρωτικά Δίκτυα με PyTorch - Διαφορετικοί μέθοδοι για καλύτερα αποτελέσματα 46

4.1 Προετοιμάζοντας τα δεδομένα 46

4.2 Χτίζοντας το μοντέλο 49

4.3 Λεπτομέρειες υλοποίησης 53

4.4 Εκπαιδεύοντας το μοντέλο 58

5 Ανάλυση Συναισθήματος με το FastText μοντέλο 61

5.1 Προετοιμασία Δεδομένων 61

5.2 Χτίζοντας το μοντέλο 63

5.3 Εκπαιδεύοντας το μοντέλο 66

6 Ανάλυση συναισθήματος με Συνελικτικά Νευρωνικά Δίκτυα (CNN) 69

6.1 Δομή των Συνελικτικών Νευρωνικών Δικτύων 70

6.2 Υλοποίηση Συνελικτικών Νευρωνικών Δικτύων 74

6.2.1 Συνελικτικά Νευρωνικά Δίκτυα σε κείμενο 75

6.3 Προετοιμασία δεδομένων 75

6.4 Χτίζοντας το μοντέλο 77

6.5 Λεπτομέρειες υλοποίησης 79

6.6 Εκπαίδευση μοντέλου 85

7 Ιστοσελίδα 88

7.1 Χτίζοντας την ιστοσελίδα 88

7.2 Docusaurus 88

7.3 Netlify 88

7.4 Χτίζοντας την ιστοσελίδα 88

7.5 Παρουσίαση ιστοσελίδας 91

Συμπεράσματα 92

ΒΙΒΛΙΟΓΡΑΦΙΑ 97

Πνευματικά δικαιώματα 100

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

Παρουσιάζονται συνοπτικά όλες οι σημαντικές συντομογραφίες που έχουν χρησιμοποιηθεί στο κείμενο της πτυχιακής π.χ.:

ΕΦΓ:	Επεξεργασία Φυσικής Γλώσσας
GPU:	Graphical Processor Unit
CNN:	Convolutional neural network
RNN:	Recurrent neural network

ΑΠΟΔΟΣΗ ΟΡΩΝ

Στην περίπτωση χρήσης ορολογίας από ξενόγλωσση βιβλιογραφία, η οποία δεν έχει αποδοθεί επισήμως στην ελληνική γλώσσα, μπορεί να αναφερθεί σε αυτήν την ενότητα η απόδοση στην ελληνική που θεωρείται περισσότερο δόκιμη. π.χ.:

Data	Δεδομένα
Natural Processing Language	Επεξεργασία Φυσικής Γλώσσας
label	ετικέτα στα δεδομένα
labeled	Δεδομένα που έχουν σημανθεί
training data	δεδομένα εκπαίδευσης
Deep Learning	Βαθιά Εκμάθηση
layer	στρώμα
training data	δεδομένα εκπαίδευσης
representation learning	εκμάθηση αναπαράστασης
training algorithm	αλγόριθμος εκπαίδευσης
training example	παράδειγμα εκπαίδευσης
representation learning	εκμάθηση αναπαράστασης
supervised learning flow	ροή της επιβλεπόμενης μάθησης
weights	βάρη
input data	δεδομένα εισόδου
loss function	συνάρτηση απώλειας
optimizer	βελτιστοποιητή
tensors	διανύσματα
sequence data	δεδομένα αλληλουχίας

Convolutional Layers	στρώματα συνέλιξης
single scalar quantity	μοναδική κλιμακωτή ποσότητα
hidden layers	κρυφά στρώματα
aforementioned memory	προαναφερθείσα μνήμη
activation functions	συναρτήσεις ενεργοποίησης
Natural Language Processing	Επεξεργασία φυσικής γλώσσας
sentiment analysis	ανάλυση συναισθήματος
sentiment classification	ταξινόμηση συναισθήματος
tensor	διάνυσμα
dataset	σύνολο δεδομένων
iteratiors	Επαναληπτές
validation loss	απώλεια επαλήθευσης
Regularization	Τακτοποίηση
output sequence	ακολουθία εξόδου
learning rate	βαθμό εκπαίδευσης
Convolutional neural network	Συνελικτικά Νευρωνικά Δίκτυα

ΕΙΣΑΓΩΓΗ

Μεθοδολογία - Τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της Πτυχιακής Εργασίας

Η συγκεκριμένη πτυχιακή εργασία βασίστηκε στην γλώσσα προγραμματισμού Python και σε βιβλιοθήκες και frameworks της που έχουν σχεδιαστεί για να υλοποιούν μοντέλα Μηχανικής Μάθησης και Βαθιάς Μάθησης.

Μετά την επεξήγηση και υλοποίηση των μοντέλων με τα σύνολα δεδομένα που επιλέχθηκαν, δημιουργήθηκε μια ιστοσελίδα όπου προστέθηκε το περιεχόμενο της εργασίας με σκοπό να λειτουργήσει ως μια εισαγωγή στην ανάλυση του συναισθήματος σε κείμενα με Μηχανική Μάθηση σε μορφή εκμάθησης (tutorial).

Python

Για την υλοποίηση των μοντέλων χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python.
Η Python είναι μια σταθερή και ευέλικτη γλώσσα προγραμματισμού που προσφέρει πολλά εργαλεία στους προγραμματιστές και σε πολλούς τομείς, η μεγάλη ποικιλία Frameworks και βιβλιοθηκών απλοποιούν την εφαρμογή διαφέρων λειτουργιών. Από την ανάπτυξη μέχρι την υλοποίηση αλλά και στην συντήρηση η Python βοηθά τους προγραμματιστές να είναι παραγωγικοί και σίγουρο για το λογισμικό και το μοντέλο που αναπτύσσουν.

Η απλότητα και η αξιοπιστία της Python την καθιστά ιδανική για Μηχανική Μάθηση. Ο λόγος είναι ότι η Μηχανική Μάθηση περιλαμβάνει περίπλοκους αλγόριθμους με αποτέλεσμα να ο χρήστης να χρειάζεται οι εργασίες του να έχουν μία ευέλικτη ροή (versatile workflows). Έτσι η απλότητα της Python εξοικονομεί χρόνο στους προγραμματιστές και τους βοηθά να επικεντρωθούν στην επίλυση των προβλημάτων της Μηχανικής Μάθησης και όχι στο τεχνικό κομμάτι της γλώσσας προγραμματισμού.

Ένα εξίσου σημαντικό χαρακτηριστικό της Python είναι η ευελιξία της. Οι προγραμματιστές έχουν την δυνατότητα να αναπτύξουν από εφαρμογές με την τρόπου προγραμματισμού της επιλογής τους. Αξίζει να αναφερθεί ότι εξίσου σημαντικό στοιχείο της ευελιξίας της γλώσσας είναι ότι δεν χρειάζεται να γίνει recompiling ο πηγαίος κώδικας.

Επιπλέον η μεγάλη ποικιλία σε Frameworks και βιβλιοθηκών βοηθούν τους Προγραμματιστές στην ανάπτυξη περίπλοκων Αλγορίθμους Μηχανικής Μάθησης.

Μια βιβλιοθήκη είναι μια συλλογή από ορισμών κλάσεων. Ο λόγος ύπαρξης μιας βιβλιοθήκης είναι η επαναχρησιμοποίηση κώδικα (code reuse). Τις περισσότερες φορές χρησιμοποιούμε βιβλιοθήκες που έχουν δημιουργηθεί από διαφορετικούς Προγραμματιστές, ωστόσο μπορούμε να δημιουργήσουμε και εμείς και να την χρησιμοποιούμε σε διαφορετικά Project βάση τις εκάστοτε ανάγκες. Οι κλάσεις και οι μέθοδοι που υπάρχουν στην βιβλιοθήκη συνήθως ορίζουν συγκεκριμένες λειτουργίες σε έναν συγκεκριμένο τομέα. Για παράδειγμα, υπάρχουν μαθηματικές βιβλιοθήκες που δίνουν την δυνατότητα στον προγραμματιστή που την χρησιμοποιεί να καλέσει μία συνάρτηση (Function) χωρίς να επαναλάβει την υλοποίηση του αλγορίθμου πίσω από αυτή την συνάρτηση.

Ένα framework περιέχει ήδη μία ροή ελέγχου και υπάρχει ένα bunch προκαθορισμένων κενών σημείων όπου μπορεί να συμπληρωθεί ο κώδικας. Το Framework είναι συνήθως πιο περίπλοκο. Ορίζει έναν σκελετό όπου η εφαρμογή καθορίζει τα δικά της χαρακτηριστικά για να συμπληρώσει τον σκελετό. Με αυτό τον τρόπο ο κώδικας μας καλείτε από το Framework όταν υπάρχουν οι κατάλληλες συνθήκες.

Μερικές διάσημες βιβλιοθήκες και Framework είναι:

- Pytorch, Keras, TensorFlow και Scikit-learn για Μηχανική Μάθηση
- Scipy για προηγμένη υπολογιστική (advanced computing)
- Seaborn για οπτικοποίηση δεδομένων (data visualization)

- Pandas για γενική σκοπού ανάλυση δεδομένων

- NumPy για επιστημονική υπολογιστική και ανάλυση δεδομένων (scientific computing and data analysis)

Όλα τα παραπάνω καθιστά την Python πρώτη επιλογή για Μηχανική Μάθηση.

Google Collaboration

Το Google Collaboration είναι ένα περιβάλλον notebook Jupyter που τρέχει εξ ολοκλήρου στο cloud με αποτέλεσμα το μόνο που να απαιτείτε είναι να υπάρχει πρόσβαση στο ίντερνετ. Το σημαντικότερο πλεονέκτημα του είναι ότι δεν απαιτεί κάποια σημαντική παραμετροποίηση ενώ παράλληλα υποστηρίζει τις πιο γνωστές βιβλιοθήκες και framework της Python που σχετίζονται με την Μηχανική Μάθηση. Μπορεί επίσης να εκτελείτε με πολλαπλούς χρήστες συνδεδεμένους ταυτόχρονα. Το Google Colab διαθέτει δυνατές NVIDIA GPU κατάλληλες για μοντέλα Μηχανικής Μάθησης και 25GB RAM. Υπάρχει η δυνατότητα για συνδρομητικό πλάνο όπου δίνει περισσότερους υπολογιστικούς πόρους, σε αυτή την εργασία χρησιμοποιείται το δωρεάν πλάνο.

Pytorch

Η PyTorch είναι μία ανοιχτού κώδικα βιβλιοθήκη μηχανικής μάθησης βασισμένη στην βιβλιοθήκη Torch και στην γλώσσα προγραμματισμού Python. Αναπτύχθηκε από το εργαστήριο AI “FAIR” της εταιρίας Facebook και χρησιμοποιείτε για εφαρμογές μηχανικής όρασης και Επεξεργασία φυσικής γλώσσας. Η PyTorch χαρακτηρίζεται ως δυναμική βιβλιοθήκη με αποτέλεσμα να είναι ευέλικτη που μπορούμε να την χρησιμοποιήσουμε βάση των απαιτήσεων. Η Pytorch είναι πολύ γνωστή βιβλιοθήκη που χρησιμοποιείται όλο και περισσότερο από ερευνητές και μαθητές. Μερικά από τα κύρια χαρακτηριστικά της Pytorch είναι: Το Απλό Interface και ότι προσφέρει εύχρηστο API με την καθιστά πολύ απλή να λειτουργήσει για να τρέξει όπως η Python.

Pythonic: Ενσωματώνεται ομαλά με την επιστήμη δεδομένων της Python. Έτσι μπορεί να αξιοποιήσει όλες τις υπηρεσίες και τις λειτουργίες που προσφέρει το περιβάλλον της Python. Έχει Υπολογιστικά γραφήματα: Εκτός αυτού, η PyTorch παρέχει μια εξαιρετική πλατφόρμα

που προσφέρει δυναμικά υπολογιστικά γραφήματα, έτσι μπορεί να αλλαχθεί η διάρκεια του χρόνου εκτέλεσης. Αυτό είναι ιδιαίτερα χρήσιμο όταν είναι άγνωστο πόση μνήμη απαιτείται για τη δημιουργία ενός μοντέλου νευρωνικού δικτύου.

Γιατί χρησιμοποιείτε η Pytorch στην έρευνα;

Όποιος εργάζεται στον τομέα της βαθιάς μάθησης και της τεχνητής νοημοσύνης έχει πιθανόν δουλέψει τη δημοφιλέστερη βιβλιοθήκη ανοικτού κώδικα της Google, την TensorFlow . Ωστόσο, το τελευταίο πλαίσιο βαθιάς μάθησης - η PyTorch επιλύει σημαντικά προβλήματα όσον αφορά την ερευνητική εργασία. Αναμφίβολα, η PyTorch είναι ο μεγαλύτερος ανταγωνιστής της TensorFlow μέχρι σήμερα ενώ παράλληλα προτιμάτε ως βιβλιοθήκη βαθιάς μάθησης και τεχνητής νοημοσύνης στην ερευνητική κοινότητα.

Ιστοσελίδα

Δημιουργήθηκε ιστοσελίδα με την βοήθεια του open source project Docusaurus όπου ο κύριος σκοπός του είναι να η δημιουργία ιστοσελίδων σε μορφή documentation. Το Docusaurus επιτρέπει να χρησιμοποιηθούν εργαλεία που ήδη γνωστά, όπως το Markdown ή το MDX για τη σύνταξη των σελίδων. Με γλώσσα προγραμματισμού React ως τη βασική δομή του Docusaurus, δίνεται η δυνατότητα να προσαρμοστεί η ιστοσελίδα. Μετά την δημιουργία της ιστοσελίδες, αναρτήθηκε στις υποδομές τις Netlify για να μπορεί να εμφανιστεί ζωντανά. Το όνομα της ιστοσελίδας είναι: https://kb-epdo.netlify.app

1 Κατηγορίες και Εφαρμογές της Ανάλυσης Συναισθήματος

1.1 Επεξεργασία Φυσικής Γλώσσας

Η Επεξεργασίας Φυσικής Γλώσσας είναι ένας κλάδος της τεχνητής νοημοσύνης που ασχολείται με την αλληλεπίδραση μεταξύ υπολογιστών και ανθρώπων χρησιμοποιώντας τη φυσική γλώσσα. Ο σκοπός της Επεξεργασίας Φυσικής Γλώσσας είναι να διαβάσει, να κατανοήσει και να εξάγει ένα νόημα από της ανθρώπινες γλώσσες.

Ωστόσο, η κατανόηση της γλώσσας από μία μηχανή είναι μια πολύ δύσκολη εργασία. Αν αναλογιστεί ότι ο άνθρωπος αρχίζει να μαθαίνει την γλώσσα από μικρή ηλικία αλλά ακόμα και όταν έχει ξοδεύσει τόσα χρόνια να την εξασκεί και να την μάθει, δεν μπορεί να την χρησιμοποιήσει στο έπακρον. Επομένως είναι δύσκολο για επιστήμονες που προσπαθούν να μοντελοποιήσουν παρόμοια φαινόμενα μάθησης και είναι δύσκολο για τους μηχανικούς που προσπαθούν να χτίσουν συστήματα για να επεξεργαστούν την φυσική γλώσσα (Brownlee). Οι πιο γνωστή μέθοδοι για την λύση προβλημάτων ΕΦΓ είναι Προβλεπόμενη Μηχανικής Μάθησης αλγορίθμων που επιχειρούν να συμπεράνουν μοτίβα χρήσης των λέξεων και κανονικότητας από ένα σύνολο pre-annotated εισόδων – εξόδων.

Για παράδειγμα: Έστω ότι ταξινομείται ένα έγγραφο σε 3 κατηγορίες: Αθλητικά, Οικονομικά, Πολιτικά. Οι λέξεις μέσα στο έγγραφο βοηθούν να ταξινομηθεί το έγγραφο στην σωστή κατηγορία, αλλά ποιες λέξεις είναι αυτές που βοηθούν στην κατηγοριοποίηση; Οι άνθρωποι μπορούν εύκολα να κατηγοριοποιήσουν τα έγγραφα, έτσι χρησιμοποιούνται μερικές εκατοντάδες παράδειγμα κατηγοριοποίησης από ανθρώπους με σκοπό ο αλγόριθμος προβλεπόμενης μάθησης να δημιουργήσει ένα μοτίβο της χρήσης τον λέξεων με σκοπό να μπορέσεις να κατηγοριοποιήσει τα έγγραφα. (Goldberg).

1.2 Περιγραφή Ανάλυσης συναισθήματος

Η ανάλυση συναισθήματος είναι μια σειρά από μεθόδους, τεχνικές και εργαλεία που προσπαθεί να εξάγει υποκειμενικές πληροφορίες, όπως απόψεις ή στάσεις βάση γλωσσολογικών χαρακτηριστικών σε ένα κείμενο. Είναι υποτομέας της Επεξεργασίας Φυσικής Γλώσσας και συνήθως αφορά την πολικότητα της κοινής γνώμης, δηλαδή η θετική ή αρνητική γνώμη για κάτι. (Mika V. Mäntylä, Daniel Graziotin, Miikka Kuutila).

Η ανάλυση του συναισθήματος εφαρμόζεται ευρέως στο "Voice of the Custom (VOC)", δηλαδή σε κριτικές και απαντήσεις ερωτηματολογίων σχετικά με προϊόν, υπηρεσίες κλπ. Επίσης εφαρμόζεται online σε κοινωνικά μέσα ενημέρωσης αλλά στον τομέα υγειονομικής περίθαλψης. (Wikipedia, n.d.).

Η κατανόηση των συναισθημάτων των ανθρώπων είναι απαραίτητη για τις επιχειρήσεις, αφού οι πελάτες μπορούν να εκφράσουν τις σκέψεις και τα συναισθήματά τους πιο ανοιχτά από ποτέ. Αν οι επιχειρήσεις μπορούν να αναλύσουν αυτόματα τα σχόλια των πελατών, από τις απαντήσεις των ερευνών, στις συνομιλίες των κοινωνικών μέσων ενημέρωσης, έχουν την δυνατότητα να προσαρμόσουν τα προϊόντα και υπηρεσίες τους στις ανάγκες του καταναλωτή.

Σύμφωνα με τα στοιχεία του paper "The evolution of sentiment analysis—A review of research topics, venues, and top cited papers", ο τομέας έχει δει τεράστια αύξηση την τελευταία 20ετία, ειδικά με την εξέλιξη του ίντερνετ όπως αναφέρθηκε. Συγκεκριμένα, έχουν δημοσιευθεί περίπου 7.000 δημοσιεύσεις όπου το 99% αυτών εμφανίστηκαν μετά το 2004, έτσι την καθιστά μία από τις ταχύτερες αναπτυσσόμενες περιοχές (Mika V. Mäntylä, Daniel Graziotin, Miikka Kuutila) . Σε αυτή την εργασία χρησιμοποιήθηκε dataset που αφορά κριτικές σε δεδομένα ταινιών από το website imdb.com.

1.2.1 Κύριες κατηγορίες μελέτης Ανάλυσης Συναισθήματος

Μέχρι στιγμής οι ερευνητές έχουν κυρίως μελετήσει τρία είδη όπου βασίζεται η ανάλυσης συναισθήματος με τεχνικές μηχανικές μάθησης:

- Επίπεδο εγγράφου (Document level)

Στην ανάλυση των συναισθημάτων σε επίπεδο εγγράφων καθαρίσουμε το γενικό προσανατολισμό του συναισθήματος σε ένα έγγραφο που περιέχει μια γνώμη, δηλαδή να καθορίσουμε εάν το έγγραφο (π.χ. Μια κριτική για μία ταινία) μεταφέρει μια γενική θετική ή αρνητική γνώμη. Σε αυτό το επίπεδο, η ταξινόμηση είναι δυαδική, δηλαδή θα είναι θετικό ή αρνητικό το αποτέλεσμα.

Ωστόσο μπορεί να διατυπωθεί ως εργασία παλινδρόμησης, για παράδειγμα, να συναχθεί συνολική βαθμολογία από 1 έως 5 αστέρια για μία κριτική.

- Επίπεδο προτάσεων (Sentence level)

Η ανάλυση των συναισθημάτων σε επίπεδο προτάσεων προσδιορίσει το συναίσθημα που εκφράζεται σε μία πρόταση. Όπως και στο επίπεδο εγγράφου, το συναίσθημα μπορεί να βρεθεί με ταξινόμηση υποκειμενικότητας και με ταξινόμηση πολικότητας, όπου στο πρώτο κατατάσσει την πρόταση αντικειμενική ή υποκειμενική ενώ το δεύτερο μία υποκειμενική πρόταση εκφράζει αρνητικό ή θετικό συναίσθημα.

- Επίπεδο χαρακτηριστικών (aspect sentiment)

Ανάλυση συναισθήματος σε επίπεδο χαρακτηριστικών είναι η εργασία εξαγωγής όρων χαρακτηριστικών και η σχετική πολικότητα των συναισθημάτων τους. Αυτό την καθιστά μια αποτελεσματική μέθοδος για τους οργανισμούς που θέλουν να παρακολουθήσουν την συχνότητα του θετικού και αρνητικού συναισθήματος που εκφράζεται σε συγκεκριμένα χαρακτηριστικά προϊόντων/υπηρεσιών και να εξάγουν μια στοχευμένη έρευνα.

Αυτό το καταφέρνει με την διασπάσει δεδομένων κειμένου σε μικρότερα κομμάτια, επιτρέποντάς την απόκτηση λεπτομερών και ακριβής πληροφορίες για τα στενευμένη δεδομένα.

Για παράδειγμα, στην πρόταση - κριτική ενός κινητού: “το κινητό έχει φοβερή κάμερα, αλλά έχει κακή μπαταρία”, το συναίσθημα είναι θετικό για το χαρακτηριστικό “κάμερα” αλλά αρνητικό για το χαρακτηριστικό “μπαταρία”.

Μία από της κυριότητες πρόκλησης στην ανάλυση συναισθήματος σε επίπεδο χαρακτηριστικών είναι όταν τα χαρακτηριστικά που ανήκουν στον ίδιο τομέα συνήθως μοιράζονται στενή σημασιολογική ομοιότητα, ενώ παράλληλα τα χαρακτηριστικά από δύο διαφορετικούς τομείς συνήθως έχουν μεγάλη σημασιολογική απόσταση μεταξύ τους. Ένας όρος που εκφράζει θετική πολικότητα συναισθημάτων σε έναν τομέα (π.χ. “delicate” σε κριτικές ταινιών) μπορεί να μεταφέρει αρνητική πολικότητα σε έναν άλλο τομέα (π.χ. “delicate” σε κριτικές κινητού τηλεφώνου).

Οι αλγόριθμοι επιβλεπόμενης μάθησης μπορούν να χειριστούν την στενή σημασιολογική ομοιότητα όταν τα δεδομένα που δίνονται για εκπαίδευση του μοντέλου είναι labeled. Στη Μηχανική Μάθηση και στα Νευρωνικά Δίκτυα που δεν είναι εύκολο γιατί απαιτεί πολύ χειροκίνητη δουλεία και χρόνο για να γίνουν labeled τα δεδομένα (Oren Pereg, Moshe

Wasserblat, Daniel Korat, n.d.). Ωστόσο, τα τελευταία χρόνια οι μέθοδοι νευρωνικών δικτύων έχουν κυριαρχήσει στην μελέτη ανάλυση συναισθήματος σε επίπεδο χαρακτηριστικών.

2 Μηχανική Μάθηση και Νευρωνικά Δίκτυα

2.1 Μηχανική μάθηση

Η Μηχανική Μάθηση είναι μια μορφή Τεχνητής Νοημοσύνης (Artificial Intelligence - AI) που το πρόγραμμα είναι σχεδιασμένο να μαθαίνει και να βελτιώνει την συμπεριφορά του σε κάποια εργασία χρησιμοποιώντας δεδομένα και πληροφορίες που του παρέχονται. Πλέον, χρησιμοποιείται σε πολλούς τομείς ένας από αυτούς είναι ο τομέας της Ανάλυσης συναισθήματος σε κείμενα.

Αρχικά αναπτύχθηκε από την μελέτη της αναγνώρισης προτύπων και της υπολογιστικής θεωρίας μάθησης στην Τεχνητή νοημοσύνη.

Ο Tom M. Mitchell πρότεινε έναν πιο επίσημο ορισμό που χρησιμοποιείται ευρέως: «Ένα πρόγραμμα υπολογιστή λέγεται ότι μαθαίνει από εμπειρία E ως προς μια κλάση εργασιών T και ένα μέτρο επίδοσης P, αν η επίδοσή του σε εργασίες της κλάσης T, όπως αποτιμάται από το μέτρο P, βελτιώνεται με την εμπειρία E».

Αυτός ο ορισμός είναι σημαντικός για τον καθορισμό της Μηχανικής μάθησης σε βασικό λειτουργικό πλαίσιο παρά με γνωστικούς όρους, ακολουθώντας έτσι την πρόταση του Alan Turing στην εργασία του «Υπολογιστικές μηχανές και Νοημοσύνη», ότι το ερώτημα αν μπορούν οι μηχανές να σκεφτούν, μπορεί να αντικατασταθεί με το ερώτημα αν μπορούν οι μηχανές να κάνουν αυτό που εμείς (ως σκεπτόμενες οντότητες) μπορούμε να κάνουμε

Η Μηχανική μάθηση χωρίζεται σε τρεις υποκατηγορίες:

- Επιβλεπόμενη Μάθηση (Supervised Learning)
- Unsupervised Learning (Μη Επιβλεπόμενη Μάθηση)
- Reinforcement learning (Ενισχυτική Μάθηση)

(Wikipedia - Machine Learning, n.d.).

2.1.1 Επιβλεπόμενη Μάθηση (Supervised Learning)

Η είναι πιο διαδεδομένη υποκατηγορία της Μηχανικής Μάθησης και η πιο διαδεδομένη για να ξεκινήσει κάποιος τα πρώτα του βήματα στον κόσμο της Τεχνικής Νοημοσύνης και Μηχανικής μάθηση.

Επιβλεπόμενη Μάθηση είναι η διαδικασία όπου ο αλγόριθμος κατασκευάζει μια συνάρτηση που απεικονίζει δεδομένες εισόδους (σύνολο εκπαίδευσης) σε γνωστές επιθυμητές εξόδους, με απώτερο στόχο τη γενίκευση της συνάρτησης αυτής και για εισόδους με άγνωστη έξοδο. Χρησιμοποιείται σε προβλήματα:

- Ταξινόμησης (Classification)
- Πρόγνωσης (Prediction)
- Διερμηνεία (Interpretation)

(repository.kallipos)

Η Επιβλεπόμενη Μάθηση βασίζεται σε δεδομένα εκπαίδευσης (training data) που είναι labeled, δηλαδή τα δεδομένα θα περιέχουν εισόδους που είναι «ζευγαρωμένες» με την σωστές εξόδους. Αυτό γίνεται συνήθως χειροκίνητα και απαιτεί πολύ χρόνο. Κατά την εκπαίδευση, ο αλγόριθμος θα ψάχνει για πρότυπα που αντιστοιχούν με την επιθυμητή έξοδος που έχουμε ορίσει από τα δεδομένα εκπαίδευσης.

Μετά την εκπαίδευση, ένας αλγόριθμος επιβλεπόμενη μάθηση θα λάβει νέες άγνωστες εισόδους και θα καθορίσει με ποια ετικέτα θα ταξινομηθούν οι νέες είσοδοι με βάση τα προηγούμενα δεδομένα εκπαίδευσης. Ο στόχος ενός μοντέλου Επιβλεπόμενης Μάθησης είναι η πρόβλεψη της σωστής ετικέτας για τα δεδομένα εισόδου που παρουσιάστηκαν πρόσφατα. Στην πιο βασική του μορφή, ένας αλγόριθμος εποπτευόμενης μάθησης μπορεί να γραφτεί απλά ως εξής:

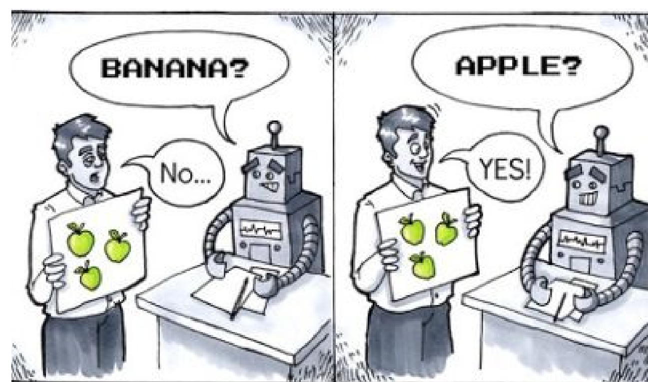
$$Y = f(x)$$

Όπου Y είναι η προβλεπόμενη έξοδος που καθορίζεται από μια συνάρτηση χαρτογράφησης f που αναθέτει μια κλάση σε μια τιμή εισόδου x. Η λειτουργία που χρησιμοποιείται για τη σύνδεση των χαρακτηριστικών εισόδου με την προβλεπόμενη έξοδο δημιουργείται από το μοντέλο μηχανικής μάθησης κατά τη διάρκεια της εκπαίδευσης (Wilson, 2019).

Για παράδειγμα: Υπάρχει ένα καλάθι με διαφόρων ειδών φρούτα. Από τα πρώτα πράγματα που θα γίνουν είναι να εκπαιδευτεί η μηχανή για όλα τα διαφορετικά φρούτα που θα μπορεί να έχει μέσα το καλάθι, ένας προς ένα.

- Εάν το σχήμα του αντικειμένου είναι στρογγυλό και συμπιεσμένο στο πάνω μέρος χρώματος κόκκινο, θα οριστεί σαν μήλο
- Εάν το σχήμα του αντικειμένου είναι μακρόστενο κυλινδρικό σχήμα χρώματος Κίτρινο – Πράσινο, θα οριστεί ως μπανάνα .
  (geeksforgeeks.org, 2021)

Αφότου εκπαιδευτούν τα δεδομένα, δημιουργείτε ένα νέο ξεχωριστό φρούτο το οποίο είναι το μήλο και ζητάμε το αναγνωρίσει.



Από την στιγμή που η μηχανή ήδη έχει μάθει κάποιες πληροφορίες από τα προηγούμενα δεδομένα και αυτή την φορά τα χρησιμοποίησε σωστά, θα ταξινόμηση το φρούτο με το χρώμα του και το σχήμα του, θα επιβεβαιώσει το όνομα του «Μήλο» και θα το τοποθετήσει στην κατηγορία των μήλων. Επιπλέον η μηχανή θα μάθει τις πληροφορίες από τα δεδομένα εκπαίδευσης (το καλάθι που περιέχει φρούτα) και θα εφαρμόσει την γνώση στα δεδομένα ελέγχου (Νέο φρούτο).

Η προβλεπόμενη μάθηση ταξινομείτε σε 2 κατηγορίες αλγορίθμων:

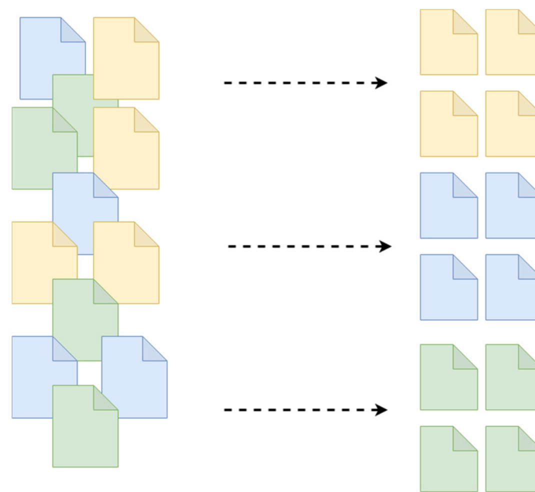
- Classification (Ταξινόμηση): Το πρόβλημα ταξινόμησης είναι όταν η μεταβλητή εξόδου είναι κατηγορία όπως «Άσπρο» ή «Μαύρο» ή «Έχει ασθένεια» ή «Δεν έχει ασθένεια»
- Regression (παλινδρόμηση): Το πρόβλημα παλινδρόμηση είναι όταν η μεταβλητή εξόδου είναι πραγματική τιμή όπως «Ευρώ» και «βάρος»

2.1.2 Unsupervised Learning (Μη Επιβλεπόμενη Μάθηση)

Αντίθετα από το επιβλεπόμενη μάθηση, στην μη επιβλεπόμενη μάθηση τα δεδομένα εκπαίδευσης δεν είναι με ετικέτα. Στη Μη Επιβλεπόμενη Μάθηση τα δεδομένα όταν περνούν στο μοντέλο κατά την διάρκεια της εκπαίδευσης είναι αποκλειστικά σε αντικείμενα εισόδου ή παραδείγματα που είναι δεν είναι labeled, δηλαδή, δεν υπάρχει είσοδος που είναι «ζευγαρωμένη» με μία έξοδος. Από την στιγμή που δεν υπάρχουν labels στα δεδομένα εκπαίδευση ο σκοπός ενός Μη Επιβλεπόμενη Μάθηση αλγορίθμου δεν είναι να μετρήσει την ακρίβεια αλλά θα προσπαθήσει να μάθει κάποιου είδους δομής από τα δεδομένα.

Στην συνέχεια, θα εξάγει χρήσιμες πληροφορίες ή χαρακτηριστικά. Πρόκειται να μάθει πως να δημιουργεί μια χαρτογράφηση από τις εισόδους που δίνονται σε συγκεκριμένες εξόδους βάση τι μαθαίνει το μοντέλο σχετικά με την δομή των δεδομένων χωρίς κανένα label.

Οι πιο διαδεδομένες εφαρμογές με Μη Επιβλεπόμενη Μάθηση είναι μέσω των clustering εφαρμογών. (Youtube, 2017).

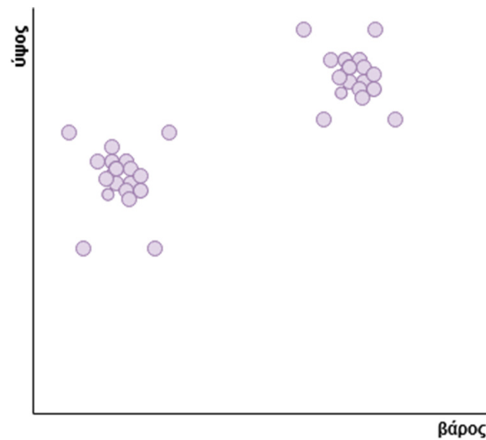


Παράδειγμα:

Έστω ότι υπάρχουν δεδομένα ύψους και βάρους για μια συγκεκριμένη ηλικία ανδρών και γυναικών χωρίς κάποια ετικέτα. Άρα κάθε νέο δεδομένο που δίνεται στο μοντέλο θα είναι σε μία πλειάδα όπου θα περιέχει ενός άτομο το ύψος και το βάρος, ωστόσο δεν θα υπάρχει κάποια ετικέτα που θα λέει ότι αυτό το άτομο είναι άνδρα ή γυναίκα.

Ένας clustering αλγόριθμος μπορεί να αναλύσει αυτά τα δεδομένα και να μάθει την δομή τους ακόμα και δεν είναι με ετικέτα.

Μέσα από τη μάθηση αυτή, μπορεί να αρχίσει να ομαδοποιεί τα δεδομένα σε groups



2.2 Βαθιά Εκμάθηση και Νευρωνικά δίκτυα

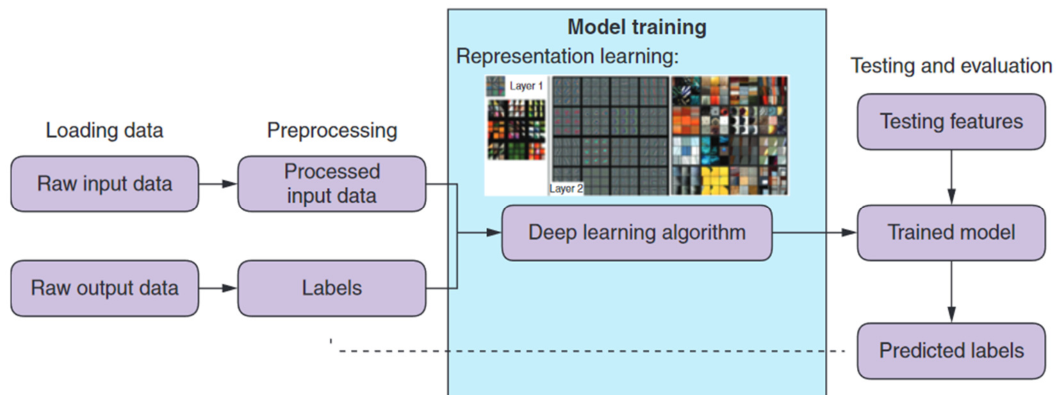
Η Βαθιά Μάθηση είναι μια υποκατηγορία της Μηχανικής μάθησης που χρησιμοποιεί μια συγκεκριμένη οικογένεια μοντέλων που αποτελούνται από ακολουθίες με απλές συναρτήσεις που μαζί, δημιουργούν μια αλυσίδα. Αυτές η αλυσίδες από συναρτήσεις είναι γνωστές ως Νευρωνικά Δίκτυα και ονομάζονται έτσι επειδή είναι εμπνευσμένα από την δομή των φυσικών εγκεφάλων.

Η βασική ιδέα της Βαθιάς Μάθησης είναι ότι αυτές οι ακολουθίες από συναρτήσεις μπορούν να αναλύσουν μια πολύπλοκη έννοια ως μια ιεραρχία απλό απλούστερες συναρτήσεις. Το πρώτο στρώμα ενός μοντέλου Βαθιάς Μάθησης μπορεί να μάθει να παίρνει ακατέργαστα δεδομένα και να τα οργανώνει με βασικούς τρόπους.

Για παράδειγμα: Να ομαδοποιεί τις κουκίδες σε γραμμές. Κάθε διαδοχικό στρώμα οργανώνει το προηγούμενο στρώμα σε πιο προηγμένες και πιο αφηρημένες έννοιες. Η διαδικασία εκμάθησης αυτών των αφηρημένων εννοιών ονομάζεται εκμάθηση αναπαράστασης (representation learning).

Ωστόσο αυτό δεν γίνεται με μαγικό τρόπο. Ο αλγόριθμος εκπαίδευσης δεν γνωρίζει ποιες έννοιες πρέπει να χρησιμοποιήσει. Αυτό που κάνει είναι να οργανώνει την είσοδο με οποιοδήποτε τρόπο, με σκοπό να ταιριάζει καλύτερα η είσοδος με το παράδειγμα εκπαίδευσης. Όμως δεν υπάρχει καμία εγγύηση ότι αυτή η αναπαράσταση θα ταιριάζει με τον τρόπο που οι άνθρωποι σκέφτονται με παρόμοια δεδομένα.

Το παρακάτω σχήμα μας δείχνει πως η εκμάθηση αναπαράστασης (representation learning) εντάσσεται στη ροή της επιβλεπόμενης μάθησης.



Παρόλα αυτά, τα μοντέλα Βαθιάς Μάθησης έχουν να μάθουν έναν τεράστιο αριθμών βαρών. Αυτό, έχει ως αποτελέσματα τα μοντέλα Βαθιάς Μάθησης, να απαιτούν μεγαλύτερα αρχεία δεδομένων, περισσότερη υπολογιστική ισχύ αλλά και πιο πρακτική προσέγγιση στην εκπαίδευση.

Η Βαθιά Μάθηση είναι κατάλληλη για περιπτώσεις όπως:

- **Υπάρχουν δεδομένα που έχουν αδόμητη μορφή**: Εικόνες, ήχος και γραπτή γλώσσα είναι μερικά παραδείγματα από τέτοιου είδους δεδομένα
- **Υπάρχουν Δεδομένα μεγάλου όγκου**
- **Υπάρχουν διαθέσιμη μεγάλη υπολογιστική ισχύ ή αρκετό χρόνο:** Όπως αναφέρθηκε τα μοντέλα Βαθιάς Μάθησης περιλαμβάνουν περισσότερους υπολογισμούς για την εκπαίδευση και αξιολόγηση.

Ωστόσο τα μοντέλα Βαθιάς Μάθησης μπορούν να εφαρμοστούν σε όλα στους σημαντικότερους κλάδους Μηχανικής Μάθησης. Ο τρόπος επιλογής ορίζετε από το είδος δεδομένων που απαιτούν εκπαίδευση.

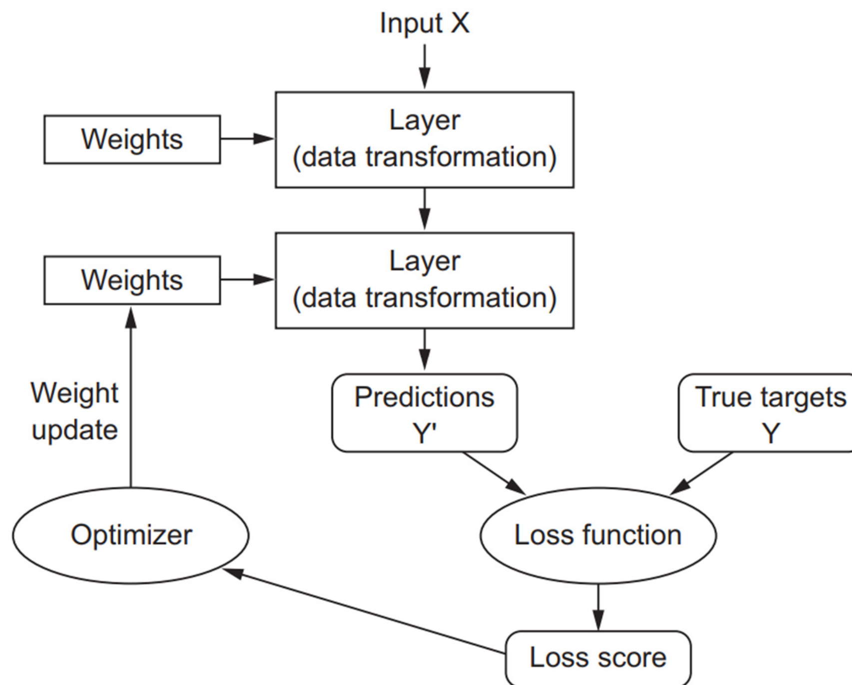
2.3 Νευρωνικά Δίκτυα

Η εκπαίδευση ενός Νευρωνικού Δικτύου περιστρέφεται από τα ακόλουθα αντικείμενα:

- **Επίπεδα (layers):** τα οποία συνδυάζονται σε ένα δίκτυο (ή μοντέλο)
- **Τα δεδομένα εισόδου** (input data) και τους αντίστοιχους στόχους (targets)
- Την **συνάρτηση απώλειας** (loss function) η οποία καθορίζει το σήμα ανάδρασης που χρησιμοποιείται για την εκμάθηση

- Τον **βελτιστοποιητή** (optimizer), ο οποίος καθορίζει τον τρόπο εκμάθησης

Παρακάτω απεικονίζεται η αλληλεπίδραση των αντικειμένων μεταξύ τους.



Στην εικόνα παρατηρείται ένα νευρωνικό δίκτυο που αποτελείται από στρώματα τα οποία να χαρτογραφούν τα δεδομένα εισόδου και να κάνουμε πρόγνωση. Στη συνέχεια, η συνάρτηση απώλειας συγκρίνει αυτές προβλέψεις με τους στόχους, παράγοντας μια τιμή απώλειας η οποία δίνει ένα μέτρο για το πόσο καλά οι προβλέψεις του δικτύου ταιριάζουν με αυτό που αναμενόταν. Ο βελτιστοποιητής χρησιμοποιεί αυτή την απώλεια για την ενημέρωση των βαρών.

2.3.1 Στρώματα (layers)

Η βασική δομή των νευρωνικών δικτύων αποτελείται από στρώματα, ένα στρώμα είναι μια μονάδα επεξεργασίας δεδομένων που λαμβάνει ως εισόδους έναν ή περισσότερα διανύσματα (tensors) και δίνει ως έξοδος ένα ή περισσότερα διανύσματα αντίστοιχα.

Κάθε είδος στρώματος είναι κατάλληλο για διαφορετικά είδη διανυσμάτων και για διαφορετικά είδη δεδομένων για επεξεργασία.

Για παράδειγμα: Ένα 2D (2 διαστάσεων) διάνυσμα δεδομένων (δείγματα, χαρακτηριστικά), συχνά επεξεργάζεται από πυκνά(dense) συνδεδεμένα στρώματα.

Τα δεδομένα αλληλουχίας (sequence data) αποθηκεύονται σε τρισδιάστατα (3D) διανύσματα (π.χ δεδομένα δειγμάτων, χαρακτηριστικών) τυπικά επεξεργάζονται από επαναλαμβανόμενα στρώματα όπως ένα στρώμα Long Short Term Memory (LSTM).

Τα δεδομένα εικόνας, αποθηκεύονται σε τεσσάρων διαστάσεων (4D) στρώματα και συνήθως επεξεργάζονται από 2D στρώματα συνέλιξης (Convolutional Layers).

2.3.2 Συνάρτηση απώλειας (loss function) και Βελτιστοποιητής (optimizer)

Όταν οριστεί η αρχιτεκτονική του δικτύου, το επόμενο βήμα είναι η επιλογή:

1. Την **συνάρτηση απώλειας** η οποία αποτελεί ένα μέτρο επιτυχίας για την λειτουργία του δικτύου.
2. Τον **Βελτιστοποιητή** ο οποίος προσδιορίζει τον τρόπου ενημερώσεις του δικτύου βάση της συνάρτηση απώλειας. Ο Βελτιστοποιητής εφαρμόζει μια ειδική παραλλαγή της stochastic gradient descent.

Ένα νευρωνικό δίκτυο που έχει πολλαπλές εξόδους, μπορεί να έχει και πολλαπλές συναρτήσεις απώλειας (μία ανά έξοδο). Ωστόσο η gradient-descent διαδικασία πρέπει να βασίζεται σε μια μοναδική τιμής απώλειας, έτσι ώστε τα δίκτυα με πολλαπλές συναρτήσεις απώλειας (multi loss networks) όλες οι απώλειας συνδυάζονται σε μία μοναδική κλιμακωτή ποσότητα.

Η επιλογή της σωστής συνάρτησης για το σωστό πρόβλημα είναι πολύ σημαντικό διότι αν η συνάρτηση δεν σχετίζεται πλήρως με τα δεδομένα και το είδος της εργασίας, το μοντέλο θα υλοποιηθεί σωστά. Από την άλλη αν η συνάρτηση είναι σωστή για το πρόβλημα, το δίκτυο θα κάνει οποιαδήποτε συντόμευση μπορεί για να ελαχιστοποίηση την απώλεια.

(Chollet, 2018).

3 Ανάλυση συναισθήματος με χρήση Επαναλαμβανόμενων νευρωνικών δικτύων και την βιβλιοθήκη PyTorch.

3.1 Επαναλαμβανόμενα Νευρωνικά Δίκτυα

Τα Επαναλαμβανόμενα Νευρωνικά Δίκτυα είναι μια κατηγορία τεχνητών νευρωνικών δικτύων τα οποία εκμεταλλεύονται, την επηρεασμένη τοπολογία από τον ανθρώπινο εγκέφαλο και τη δομή των νευρώνων του εγκεφάλου, με στόχο τη δημιουργία ενός δικτύου το οποίο θα μαθαίνει από τα δεδομένα που του δίνονται και με τη σειρά που προσλαμβάνονται και θα προσπαθεί να δημιουργήσει νέο περιεχόμενο βασισμένο σε αυτά (Λιθοξοΐδης, 2019).

Ο Γράφος υπολογισμού στα ΕΝΔ περιέχει κατευθυνόμενους κύκλους όπου η πληροφορία ταξιδεύει σε βρόχους από στρώμα σε στρώμα με αποτέλεσμα, η κατάσταση του μοντέλου επηρεάζεται από την προηγούμενη κατάσταση.

Τα ΕΝΔ έχουν κρυφά στρώματα. Τα κρυφά στρώματα έχουν συνδέσεις πίσω στον εαυτό τους, αυτό τους επιτρέπει στις καταστάσεις τον κρυφών στρωμάτων ενός χρονικού σημείου να χρησιμοποιηθούν ως είσοδο στα κρυφά στρώματα του επόμενου χρονικού σημείου.

Αυτό παρέχει την προαναφερθείσα μνήμη, η οποία αν έχει εκπαιδευτεί κατάλληλα, επιτρέπει στις κρυφές καταστάσεις να συλλάβουν πληροφορίες σχετικά με την χρονική σχέση μεταξύ την ακολουθία της εισόδου και ακολουθία εξόδου (John McGonagle, Christopher Williams, Jimin Khim, n.d.).

Επειδή μπορούν να μοντελοποιήσουν ακολουθίες ζευγαριών εισόδου - εξόδου, τα ΕΝΔ μπορούν να έχουν μεγάλη επιτυχία σε εφαρμογές στην επιστήμη της Επεξεργασίας Φυσικής Γλώσσας. Αυτό περιλαμβάνει Ανάλυση συναισθημάτων Μηχανική μετάφραση, αναγνώριση ομιλίας και Γράφος γλώσσας κλπ.

Επιπλέον, τα ΕΝΔ έχουν χρησιμοποιηθεί σε Ενισχυτική Μάθηση για να λύσουν δύσκολα προβλήματα σε επίπεδο καλύτερο από ανθρώπους. Ένα χαρακτηριστικό παράδειγμα είναι το AlphaGo, που κέρδισε τον παγκόσμιο πρωταθλητή Lee Sedol το 2016 (Wikipedia, Wikipedia.org, n.d.). Επίσης άλλο ένα χαρακτηριστικό παράδειγμα είναι η υλοποίηση ενός δια δραστικού editor όπου δημιουργεί χειρόγραφων δειγμάτων (Graves, 2013).

Το σύστημα ακολουθίας των ΕΝΔ είναι πολύ πιο δυνατό σε σχέση με τα πιο απλά Νευρωνικά δίκτυα τα οποία είναι "καταδικασμένα" από την αρχή με το να έχουν σταθερό αριθμό

υπολογιστικών βημάτων, με αποτέλεσμα να είναι πιο ελκυστικά σε αυτούς που θέλουν να χτίσουν πιο έξυπνα και περίπλοκα συστήματα.

Επιπλέον τα ΕΝΔ συνδυάζουν την είσοδο του διανύσματος με την κατάσταση του με μία σταθερή(αλλά που έχει εκπαιδευτεί) συνάρτηση για να παράγει μια νέα κατάσταση διανύσματος (state vector).

Αυτό μπορεί (σε προγραμματιστικούς όρους) να ερμηνευτεί ως ένα τρέχων και σταθερό πρόγραμμα με ορισμένες εισόδους και μερικές εσωτερικές μεταβλητές δηλαδή τα ΕΝΔ περιγράφουν προγράμματα. Συγκεκριμένα, είναι γνωστό ότι τα ΕΝΔ είναι Turing-Complete.

Τα ΕΝΔ καλούνται "επαναλαμβανόμενα" γιατί εκτελούν τον ίδιο υπολογισμό (που καθορίζονται από τα weights, τα biases και τις συναρτήσεις ενεργοποίησης) για κάθε στοιχείο στην ακολουθία εισόδου. Η διαφορά μεταξύ των εξόδων για διαφορετικά στοιχεία της ακολουθίας εισόδου προέρχεται από τις διαφορετικές κρυφές καταστάσεις, οι οποίες είναι εξαρτώμενες από το τρέχον στοιχείο στην ακολουθία εισόδου και η τιμή των κρυφών καταστάσεων από το τελευταίο βήμα.

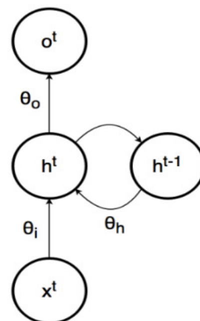
Οι ακόλουθες εξισώσεις ορίζουν πως ένα ΕΝΔ εξελίσσεται με την πάροδο του χρόνου:

$$\boldsymbol{o^t} = \boldsymbol{f} \, (\boldsymbol{o^t}; \boldsymbol{\theta})$$

$$h^t = g \, (h^{t-1}; x^t; \theta)$$

Όπου το o^t είναι η έξοδος του ΕΝΔ σε χρόνο t ,x^t είναι η είσοδος του ΕΝΔ σε χρόνο t και h^t είναι η κατάσταση των κρυφών καταστάσεων σε χρόνο t.

Η παρακάτω εικόνα είναι ένα απλό γραφικό μοντέλο που απεικονίσει τη σχέση μεταξύ αυτών των τριών μεταβλητών σε ένα Γράφο υπολογισμού του ΕΝΔ.



Η πρώτη εξίσωση δηλώνει ότι για παραμέτρους $\boldsymbol{\theta}$ (οι οποίες ενθυλακώνει τα βάρη και biases για το δίκτυο), η έξοδος σε χρόνο t εξαρτάται μόνο από την κατάσταση των κρυφών καταστάσεων σε χρόνο t, όπως και σε ένα *Ανατροφοδοτούμενο* Νευρωνικό Δίκτυο. Η δεύτερη

εξίσωση δηλώνει ότι για παραμέτρους $\boldsymbol{\theta}$, το κρυφό στρώμα σε χρόνο t εξαρτάται από το κρυφό στρώμα σε χρόνο $t-1$ και η είσοδος σε χρόνο t.

Η δεύτερη εξίσωση αποδεικνύει ότι το ΕΝΔ μπορεί να θυμηθεί το παρελθόν με το να επιτρέπει σε προηγούμενος υπολογισμούς h^{t-1} να επηρεάζουν τους παρών υπολογισμούς h^t. Έτσι, ο στόχος της εκπαίδευσης ενός ΕΝΔ είναι να παρθεί η ακολουθία o^{t+T} για να ταιριάξει στην ακολουθία y^t, όπου T αντιπροσωπεύει το time lag (είναι πιθανό να είναι το $T=0$) μεταξύ της πρώτης σημαντικής ΕΝΔ έξοδος o^{T+1} και της πρώτης στενευμένης έξοδο y^t. Ένα time εισάγεται μερικές φορές για να επιτρέψει στον ΕΝΔ να φτάσει σε μια ενημερωτική κρυφή κατάσταση h^{T+1} προτού αρχίσει να παράγει στοιχεία από την ακολουθία εξόδου.

Αυτό είναι ανάλογα το πως οι άνθρωποι μεταφράσουν από Αγγλικά σε Ισπανικά, όπου συχνά ξεκινάει με το να διαβάζονται οι πρώτες λέξεις για να παραχθούν τα συμφραζόμενα και για να μεταφράσουμε την υπόλοιπη πρόταση. Μια απλή περίπτωση όταν αυτό απαιτείται πραγματικά είναι όταν η τελευταία λέξη στην ακολουθία εισόδου αντιστοιχεί στην πρώτη λέξη στην ακολουθία εξόδου. Στη συνέχεια, θα ήταν απαραίτητο να δημιουργηθεί καθυστέρηση την ακολουθία εξόδου μέχρι να διαβάσετε ολόκληρη την ακολουθία εισόδου. (John McGonagle, Christopher Williams, Jimin Khim, n.d.)

3.2 Υλοποίηση μοντέλου Ανάλυσης συναισθήματος με Επαναλαμβανόμενα Νευρωνικά Δίκτυα

Σκοπός αυτής της υλοποίησης μοντέλου είναι να γίνουν κατανοητές γενικές έννοιες και πως δουλεύει ένα ΕΝΔ στην ανάλυση συναισθήματος χωρίς απαραίτητα να είναι σημαντικά τα αποτελέσματα.

Σε αυτή την υλοποίηση πως πραγματοποιείται:

- φόρτωση δεδομένων από την βιβλιοθήκη της TorchText (imdb dataset)
- δημιουργία train, test και validation splits
- δημιουργία data iterators
- καθορισμός του μοντέλου
- εφαρμογή train, evaluate, test loops

Προετοιμασία Δεδομένων

Στην ταξινόμηση συναισθήματος τα δεδομένα (IMDB dataset) αποτελούνται από ανεπεξέργαστες συμβολοσειρές (raw string) των κριτικών και από θετικό ή αρνητικό συναίσθημα (pos, reg). Μία από τις κύριες έννοιες της TorchText είναι η κλάση Field η οποία ορίζει πως τα δεδομένα στο μοντέλο θα επεξεργαστούν.

Ο ορισμός της κλάσης Field

Πιο συγκεκριμένα, ορίζει έναν τύπο δεδομένων μαζί με οδηγίες για την μετατροπή τους σε διάνυσμα (tensor). Η κλάση Field μοντελοποιεί χρησιμοποιώντας τους κοινούς τύπους δεδομένων για επεξεργασία κειμένων και μπορούν να αναπαρασταθούν από τους tensors. Διαθέτει ένα αντικείμενο (object) Vocab που ορίζει το σύνολο των πιθανών τιμών για τα στοιχεία του field τις αντίστοιχες αριθμητικές αναπαραστάσεις τους.

Το αντικείμενο Field περιέχει επίσης και άλλες παραμέτρους που είναι σχετικές με το πως οι τύποι δεδομένων θα πρέπει να αριθμηθούν, όπως μια μέθοδος τμηματοποίησης (tokenization) για το είδος του Tensor που πρέπει να παραχθεί (Docs, n.d.).

Χρησιμοποιείται το πεδίο TEXT για να οριστεί πως η κριτική θα επεξεργαστεί και το πεδίο LABEL για να επεξεεργαστεί το συναίσθημα (sentiment).

Το πεδίο FIELD έχει ως όρισμα το `tokenize='spacy'` που ορίζει ότι την τμηματοποίηση (η ενεργεία που χωρίζει τις σειρές σε διακεκριμένα 'τμήματα'/ 'tokens') που θα γίνει χρησιμοποιώντας τον τμηματοποιητή (tokenizer) (Spacy, n.d.).

Αν κανένα όρισμα `tokenize` δεν περαστεί, δηλαδή δεν τμηματοποιηθεί η κριτική που περιέχει τα δεδομένα, τότε η προκαθορισμένη τμηματοποίηση είναι να χωρίσει τις σειρές σε κενά. Το πεδίο `LABEL` ορίζεται από το `LabelField`, ένα ειδικό υποσύνολο της κλάσης `Field` ειδικά όταν χρησιμοποιείτε για τον χειρισμό ετικετών (Docs, n.d.).

```
!pip install -U torchtext==0.10.0
```

```
import torch
from torchtext.legacy import data
from torchtext.legacy.data import Field, TabularDataset, BucketIterat
or, Iterator
# Αρχικοποιούμε την γεννήτρια τυχαίων αριθμών με σταθερά για να διασφαλίσουμε
ότι τα αποτελέσματα θα είναι αναπαραγωγίσιμα.
SEED = 1234

torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True

TEXT = data.Field(tokenize = 'spacy',
                  tokenizer_language = 'en_core_web_sm')
LABEL = data.LabelField(dtype = torch.float)
```

Άλλο ένα χαρακτηριστικό της Torchtext είναι ότι υποστηρίζει γνωστά σύνολα δεδομένων (datasets) που χρησιμοποιούνται στην Επεξεργασία φυσικής γλώσσας/Natural Language Processing (NLP).

Συγκεκριμένα για Ανάλυση Συναισθήματος είναι υποστηρίζει το IMDB σύνολο δεδομένων και το Sentiment Treeback από το πανεπιστήμιο του Stanford.

Ο παρακάτω κώδικας κατεβάζει αυτόματα ένα IMDB dataset και το χωρίζει τα αρχικά train/test ως `torchtext.datasets` αντικείμενα. Επεξεργάζεται τα δεδομένα χρησιμοποιώντας την κλάση Field όπου ορίσαμε πριν. Το IMDb dataset περιέχει 50.000 κριτικές ταινιών, όπου κάθε ένα από αυτά έχει σημανθεί ως θετική ή αρνητική κριτική ταινιών.

Επιπλέον χρησιμοποιείται η συνάρτηση `len()` για να δείξει πόσα παραδείγματα υπάρχουν για κάθε διαχώρισμα ελέγχοντας το μήκος τους.

```
from torchtext.legacy import datasets
# Κάνει τους διαχωρισμούς στα δεδομένα
train_data, test_data = datasets.IMDB.splits(TEXT, LABEL)
print (f'Number of training examples: {len(train_data)}')
print (f'Number of testing examples: {len(test_data)}')
```

```
Number of training examples: 25000
Number of testing examples: 25000
```

Το IMDB dataset απαρτίζεται μόνο από train/test splits με αποτέλεσμα να χρειαστεί να δημιουργηθεί ένα validation set (σύνολο επικαιροποίησης). Αυτό, θα επιτευχθεί χρησιμοποιώντας την `.split()` μέθοδο. Από προεπιλογή στα `splits` το 70% των παραδειγμάτων είναι στο training set (σύνολο εκπαίδευσης) και το 30% στο validation set (σύνολο επικαιροποίησης). Ωστόσο μέσω `split_ratio` (Pytorch, n.d.) ορίσματος μπορεί να αλλαχθεί η αναλογία (ratio. Στην συνέχεια μέσω του ορίσματος `random_state` διασφαλίζεται το ίδιο ποσοστό train/validation split κάθε φορά.

```
import random
train_data, valid_data = train_data.split(random_state = random.seed(SEED))
# Παρατηρείται αν ολοκληρώθηκε ο διαχωρισμός
print (f'Number of training examples: {len(train_data)}')
print (f'Number of validation examples: {len(valid_data)}')
print (f'Number of testing examples: {len(test_data)}')
```

```
Number of training examples: 17500
Number of validation examples: 7500
Number of testing examples: 25000
```

Ορισμός λεξιλογίου

Προτού ξεκινήσει η δομή του μοντέλο, είναι σημαντικό να δημιουργηθεί και να οριστεί ένα λεξιλόγιο (vocabulary). Τα Νευρωνικά δίκτυα παίρνουν ως είσοδο ακέραιους αριθμούς και όχι συμβολοσειρές. Άρα θα πρέπει να βρεθεί ένας τρόπο το πως θα μετατραπούν οι λέξεις που έχει το dataset σε μία μορφή που μπορεί να αναγνωριστεί από το Νευρωνικό Δίκτυο, δηλαδή σε ακέραιους αριθμούς. Ο παραδοσιακός τρόπος ονομάζεται "one-hot encoding". Ωστόσο ο καλύτερος τρόπος για αναπαράσταση δεδομένων κειμένου γίνεται με τους "word vectors" όπου θα αναλυθούν στο επόμενο κεφάλαιο.

One Hot Encoding

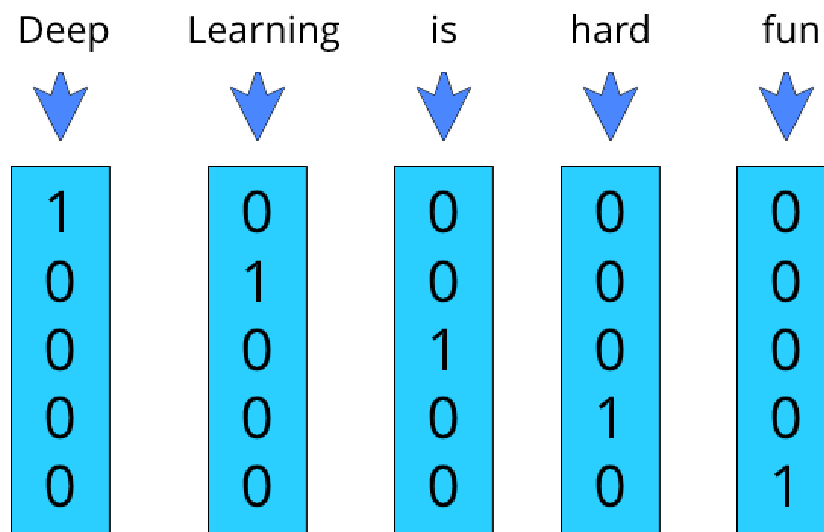
Σε έναν one-hot encoding αναπαριστάτε κάθε λέξη ως διάνυσμα με μήκος **V**. Όπου το **V** είναι ο συνολικός αριθμός των μοναδικών λέξεων που είναι διαθέσιμες στο σύνολο των δεδομένων κειμένου. Στην ουσία το είναι το πλήθος του λεξιλογίου (vocabulary count). Ένα τέτοιο διάνυσμα ονομάζεται διάδικο διάνυσμα όπου με την τιμή 1 να βρίσκεται σε ένα μοναδικό ευρετήριο (index) για κάθε λέξη και η τιμή 0 να είναι σε κάθε άλλο ευρετήριο του διανύσματος.

Για παράδειγμα: Έστω ότι υπάρχουν αυτές οι 2 προτάσεις

1. Deep Learning is hard
2. Deep Learning is fun

Σε αυτό το παράδειγμα το **V** έχει την τιμή 5 επειδή συνολικά και στις 2 προτάσεις υπάρχουν 5 μοναδικές λέξεις: [‘Deep’, ‘learning’, ‘is’, ‘hard’, ‘fun’].

Για να αναπαριστάτε κάθε λέξη, θα χρησιμοποιήσουμε ένα διάνυσμα με μήκος 5



Παρατηρείται στην εικόνα ότι κάθε λέξη έχει ένα δικό της “one-hot encoded” vector. Αυτή η μέθοδος είναι σχετικά απλή ωστόσο δημιουργεί 2 προβλήματα.

Το πρώτο πρόβλημα είναι ότι το ευρετήριο που εκχωρείτε σε κάθε λέξη δεν έχει κάποια σημασιολογική σημασία. Για παράδειγμα στα διανύσματα “dog” και “cat” υπάρχει κάποια ομοιότητα σημασιολογικά, ωστόσο στο “one-hot encoding” διάνυσμα έχει τόσο ομοιότητα όσο έχει το διάνυσμα “dog” με “computer”. Αυτό έχει ως αποτέλεσμα το Νευρωνικό Δίκτυο να

πρέπει να κάνει μεγαλύτερη εκπαίδευση για να καταλάβει ότι αυτά τα 2 δεν έχουν κάποια σημασιολογική σημασία μεταξύ τους. Το πρόβλημα αυτό επιλύετε με τους “words vectors”.

Το δεύτερο πρόβλημα είναι ότι το μέγεθος για κάθε λέξη που θα είναι σε “one-hot encoded” vector θα είναι πάντα **V** (Demystified, n.d.)**.** Αυτό σημαίνει ότι τα σύνολα δεδομένων που έχουν πάνω από 100.000 μοναδικές λέξεις ότι έχουν 100.000 διανύσματα “one-hot encoded” για κάθε μοναδική λέξη όπου θα υπάρχει πιθανότητα να καθυστέρηση την εκπαίδευση.

Για να συνεχιστεί να χρησιμοποιούνται τα “one-hot encoded” διανύσματα θα πρέπει να «κοπεί» το λεξιλόγιο, αυτό μπορεί να πραγματοποιηθεί με τους ακόλουθους μεθόδους:

- **Μέθοδος 1**: Να συλλεχθούν οι πιο χρησιμοποιημένες λέξεις στο dataset ή
- **Μέθοδος 2**: Να συλλεχθούν οι λέξεις που έχουν χρησιμοποιηθεί λιγότερο στο dataset.

Στην προκειμένη περίπτωση θα χρησιμοποιηθεί η **μέθοδος 1** όπου θα συλλεχθούν οι 25.000 πιο χρησιμοποιημένες μοναδικές λέξεις. Οι λέξεις που θα αφαιρεθούν από το λεξιλόγιο, αντικαθίστανται με έναν ξεχωριστό άγνωστο ή `<unk>` `token`. Για παράδειγμα, αν ή πρόταση είναι "Deep Learning is fun and I love it" αλλά η λέξη "love" δεν είναι στο λεξιλόγιο, θα είναι " Deep Learning is fun and I **unk** it"

Επίσης παρατηρείται ότι όταν τυπώνονται τα tokens το λεξιλόγιο έχει την τιμή 25002 αντί του 25000 που ορίστηκε. Αυτό οφείλετε στα `<unk>` `token` και `<pad>` `token`. Το <pad> token χρησιμοποιείται διότι όταν τροφοδοτούνται προτάσεις στο μοντέλο μας, τροφοδοτείται ένα batch (σύνολο/πακέτο) από αυτές τις προτάσεις την φορά, π.χ. περισσότερες από μία κάθε φορά και όλες οι προτάσεις στο batch πρέπει να έχουν το ίδιο μέγεθος. Έτσι για να εξασφαλισθεί ότι κάθε πρόταση έχει το ίδιο μέγεθος μέσα στο batch, όποια πρόταση είναι μικρότερη από την μεγαλύτερη μέσα στο batch γίνετε "padded".

Άρα ακολουθώντας την μέθοδο 1 χτίζεται το λεξιλόγιο όπου κρατάει τα τις περισσότερες χρησιμοποιημένες μοναδικές λέξεις.

```
# Ορίζεται το μέγεθος του λεξιλογίου και εκτυπώνονται οι μοναδικές λέξεις
MAX_VOCAB_SIZE = 25_000
TEXT.build_vocab(train_data, max_size = MAX_VOCAB_SIZE)
LABEL.build_vocab(train_data)
print (f"Unique tokens in TEXT vocabulary: {len(TEXT.vocab)}")
print (f"Unique tokens in LABEL vocabulary: {len(LABEL.vocab)}")
```

```
Unique tokens in TEXT vocabulary: 25002
```

```
Unique tokens in LABEL vocabulary: 2
```

Η κλάση vocab μας δίνει την δυνατότητα να δούμε τις πιο συχνές λέξεις του λεξιλογίου και με τι συχνότητα εμφανίζονται.

```
print (TEXT.vocab.freqs.most_common(20))
# Χρησιμοποιείται το vocab.stoi για να ελεγχθούν ότι τα labels ότι είναι σωστά.
print(LABEL.vocab.stoi)
```

```
[('the', 202046), (',', 192526), ('.', 164640), ('and', 109606), ('a', 108887), ('of', 100669), ('to', 93450), ('is', 76162), ('in', 61352), ('I', 54156), ('it', 53517), ('that', 49307), ('"', 44152), ("'s", 43358), ('this', 42413), ('-', 36724), ('/><br', 35786), ('was', 34853), ('as', 30229), ('with', 29724)]
```

```
defaultdict(<function _default_unk_index at 0x7ff0c24dbf28>, {'neg': 0, 'pos': 1})
```

Φτάνοντας στο τελευταίο σημείο για την προετοιμασία των δεδομένων ορίζεται το Batch_size το οποίο καθορίζει τον αριθμών των παραδειγμάτων (examples) που θα μεταδοθούν (propagated) μέσω του δικτύου. (itdxer, n.d.).

Στην συνέχεια πρέπει να δημιουργηθούν iteratiors (Επαναληπτές) (Guide, n.d.). Πραγματοποιείται επανάληψη (iterate) τα αποτελέσματα στο loop (βρόχο) του training/evaluation όπου επιστρέφουν ένα batch από παραδείγματα(κατοχυρωμένα(indexed) και να έχουν γίνει μετατροπή σε tensors) για κάθε επανάληψη. Χρησιμοποιείται το *BucketIterator* όπου είναι ένα ξεχωριστό είδος επανάληψης(iterator) που θα επιστρέψει ένα batch από παραδείγματα όπου κάθε παράδειγμα είναι πανομοιότυπο σε μήκος, ελαχιστοποιώντας το ποσό του padding ανά παράδειγμα.

Επίσης θα πρέπει να τοποθετηθούν οι tensors που επέστρεψαν από τις επαναλήψης στην GPU. Το PyTorch χειρίζεται αυτή την τοποθέτηση χρησιμοποιώντας την μέθοδο torch.device, έτσι περνάει την συσκευή στον iterator.

```
BATCH_SIZE = 64
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
train_iterator, valid_iterator, test_iterator = data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = BATCH_SIZE,
    device = device)
```

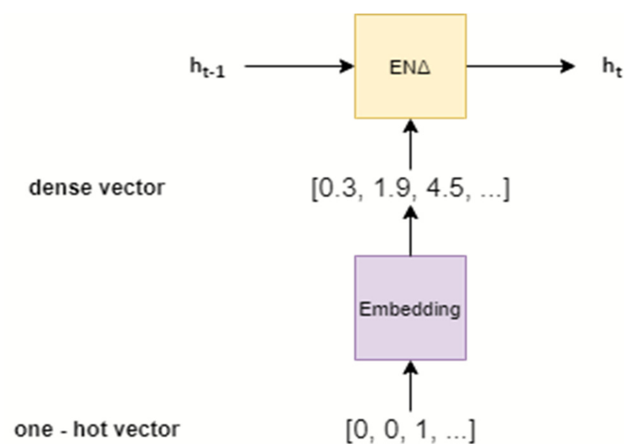
3.2.1 Χτίζοντας το μοντέλο

Το επόμενο στάδιο το οποίο είναι να χτιστεί το μοντέλο. Στα μοντέλα της Pytorch υπάρχει κώδικας boilerplate κώδικας. Boilerplate κώδικας ονομάζετε ένα κομμάτι κώδικα το οποίο το χρησιμοποιείται σε πολλά σημεία με χωρίς ή μικρές αλλαγές για λόγους λειτουργικότητας (Educative, n.d.). Εδώ χρησιμοποιήθηκε boilerplate κώδικα στη κλάση `RNN` που είναι υπό-κλάση της `nn.Module` με την `super`.

Μέσα στο `__init__` ορίζουμε τα στρώματα του αρθώματος (module) και οι παραμέτρους των στρωμάτων έχουν αρχικοποιηθεί με τυχαίες τιμές.

Τα 3 στρώματα είναι:

1. ένα ενσωματωμένο στρώμα (embedding layer). Το ενσωματωμένο στρώμα/embedding layer χρησιμοποιείτε για να μετατρέψει τον αραιό (sparse) one-hot διάνυσμα (είναι αραιό διότι τα περισσότερα από τα στοιχεία μας είναι 0) σε έναν πυκνό (dense) ενσωματωμένο διάνυσμα (embedding vector) καθώς η dimensionality είναι πολύ μικρότερη και όλα τα στοιχεία είναι πραγματικοί αριθμοί. Αυτό το embedding layer είναι ένα ενιαίο πλήρως συνδεδεμένο layer. Καθώς μειώνεται η dimensionality των εισόδων στον RNN, υπάρχει μια θεωρία ότι οι λέξεις που έχουν πανομοιότυπη επίδραση στο συναίσθημα της κριτικής, χαρτογραφούνται μαζί κοντά σε έναν πυκνό(dense) vector space (MonkeyLearn, n.d.).
2. ένα ΕΝΔ στρώμα παίρνει τον πυκνό διάνυσμα (dense vector) και την προηγούμενη κρυφή κατάσταση (hidden state) h_{t-1} για να υπολογίσει την επόμενη κρυφή κατάσταση h_t.



3. ένα γραμμικό στρώμα (linear layer) που παίρνει την τελική κρυφή κατάσταση (hidden state) και την τροφοδοτεί μέσω ενός πλήρως συνδεδεμένου στρώματος (fully

connected layer), $f(h_t)$ με σκοπό να μετατρέψει την κατάσταση στην σωστή διάσταση εξόδου.

Στην συνέχεια χρησιμοποιείται η μέθοδο `forward` για να τροφοδοτηθούν τα παραδείγματα στο μοντέλο.

Κάθε δέσμη(batch) που βρίσκεται στην μεταβλητή `text` είναι ένα διάνυσμα (tensor) με μήκος `[μήκος πρότασης, μέγεθος δέσμης]([sentence length, batch size])`. Κάθε δέσμη από προτάσεις περιέχει λέξεις όπου μετατρέπονται σε one-hot διάνυσμα.

Η Pytorch αποθηκεύει ένα διάνυσμα one-hot ως τιμή ευρετηρίου (index value). Αυτό το πετυχαίνει μετατρέποντας μία λίστα από tokens σε μία λίστα από ευρετήρια (indexes). Αυτή η τεχνική ονομάζεται *numericalizing*.

Μέσα στην `forward` μέθοδο η δέσμη εισόδου (input batch) περνάει από το ενσωματωμένο στρώμα (embedding layer) για να ενσωματωθεί μέσω της μεταβλητής `embedded`. Αυτό δίνει μια αναπαράσταση πυκνού διανύσματος (dense vector) στις προτάσεις. Η μεταβλητή `embedded` είναι ένα διάνυσμα με μήκος [sentence length, batch size, embedding dim].

Η μεταβλητή `embedded` τροφοδοτείται μέσα στο ΕΝΔ μοντέλο.

Σημείωση: Σε μερικά frameworks πρέπει να τροφοδοτηθεί η αρχική κρυφή κατάσταση (hidden state) H_o στο ΕΝΔ, ωστόσο στην PyTorch αν δεν έχει περαστεί αρχική κρυφή κατάσταση ως παράμετρος, τότε προεπιλέγει έναν tensor που έχει όλο μηδενικά.

Το ΕΝΔ επιστρέφει 2 διανύσματα:

- Την μεταβλητή `output` με μέγεθος [sentence, length, batch size, hidden dim] και
- την μεταβλητή `hidden` με μέγεθος [1, batch size, hidden dim].

Η μεταβλητή `output` είναι η σύνδεση της κρυφής κατάστασης από κάθε χρονικό βήμα Ενώ η μεταβλητή `hidden` είναι η τελική κρυφή κατάσταση.

Με το όρισμα `assert` επιβεβαιώνεται το παραπάνω.

Σημειώνεται ότι η `squeeze` μέθοδος χρησιμοποιείται για την αφαίρεσε μιας διάστασης με μέγεθος 1.

Τέλος, τροφοδοτείται η τελευταία κρυφή κατάσταση μέσω ενός γραμμικού στρώματος (linear layer) με την μεταβλητή `fc` για να παραχθεί μια πρόβλεψη.

```
import torch.nn as nn
```

```
class RNN(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        self.embedding = nn.Embedding(input_dim, embedding_dim)
        self.rnn = nn.RNN(embedding_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, output_dim)
    def forward(self, text):
        #το text ισούται = [sent len, batch size]
        embedded = self.embedding(text)
        #το embedded ισούται = [sent len, batch size, emb dim]
        output, hidden = self.rnn(embedded)
        #το output ισούται = [sent len, batch size, hid dim]
        #το hidden ισούται = [1, batch size, hid dim]
        assert torch.equal(output[-1,:,:], hidden.squeeze(0))
        return self.fc(hidden.squeeze(0))
```

Επόμενο βήμα είναι η δημιουργία ενός στιγμιότυπου (instance) για την RNN κλάσης. Η **διάσταση εισόδου** (input dimension) είναι η διάσταση του one-hot διανύσματος, το οποίο είναι ίσος με το μέγεθος του λεξιλογίου. Η **διάσταση ενσωμάτωσης** (embedding dimension) έχει το μέγεθος των dense word vectors. Αυτό το μέγεθος είναι συνήθως 50-250 διαστάσεις, αλλά εξαρτάται από το μέγεθος του λεξιλογίου. Η **κρυφή διάσταση** (hidden dimension) έχει το μέγεθος των κρυφών καταστάσεων (hidden states). Αυτό το μέγεθος είναι συνήθως 100-500 διαστάσεις, επίσης εξαρτάται από παράγοντες όπως το μέγεθος του λεξιλογίου, το μέγεθος των dense vectors και την πολυπλοκότητα της εργασίας. Η **διάσταση εξόδου** (output dimension) συνήθως είναι ο αριθμός των κλάσεων, ωστόσο σε περίπτωση που υπάρχουν μόνο 2 κλάσεις η τιμή εξόδου(output value) θα είναι μεταξύ 0 και 1 και θα μπορεί να είναι μονοδιάστατο (1-dimensional). Για παράδειγμα: a single scalar real number

```
INPUT_DIM = len(TEXT.vocab)
EMBEDDING_DIM = 100
HIDDEN_DIM = 256
OUTPUT_DIM = 1
model = RNN(INPUT_DIM, EMBEDDING_DIM, HIDDEN_DIM, OUTPUT_DIM)
```

Δημιουργείται επίσης μία συνάρτηση (function) που θα λέει πόσοι εκπαιδεύσιμη (trainable) παράμετροι έχει το μοντέλο έτσι ώστε να μπορεί να συγκριθεί με τον αριθμό αυτών των παραμέτρων ανάμεσα σε διάφορα μοντέλα.

```
def count_parameters(model):
```

```
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'The model has {count_parameters(model):,} trainable parameters')
```

```
The model has 2,592,105 trainable parameters
```

3.3 Εκπαιδεύοντας το μοντέλο

Στην συνέχεια θα στηθεί ο τρόπος εκπαίδευσης του μοντέλου και μετά θα το εκπαιδευτεί. Αρχικά θα δημιουργηθεί ένας βελτιστοποιητής (optimizer). Αυτός είναι ο αλγόριθμος που χρησιμοποιείται για να ενημερωθούν οι παράμετροι του module. Θα χρησιμοποιηθεί η stochastic gradient descent (SGD) (cepe-eua, n.d.).

Stochastic gradient descent (SGD) είναι ένας αλγόριθμος βελτιστοποίησης που εκτιμάει την κλίση σφάλματος (error gradient) για την τρέχουσα κατάσταση του μοντέλου με την χρήση παραδειγμάτων από το σύνολο δεδομένων που έχει ορισθεί για εκπαίδευση. Μετά ενημερώνει τα βάρη του μοντέλου χρησιμοποιώντας τον αλγόριθμο «back-propagation of errors» ή «backpropagation». Ο ρυθμός εκμάθησης είναι μία υπερπαράμετρος που ελέγχει το πόσο θα αλλάξει το μοντέλο με βάση το εκτεινόμενο σφάλμα κάθε φορά που τα (βάρη) του μοντέλου ενημερώνονται.

Η επιλογή του ρυθμού εκμάθησης είναι μια δύσκολη διαδικασία διότι αν επιλεχθεί ο λάθος ρυθμός μπορεί είτε η εκπαίδευση να γίνει πολύ αργή (και ίσως κολλήσει) ή πολύ γρήγορη η οποία θα είναι ασταθές. Ο ρυθμός εκμάθησης ίσως είναι η πιο σημαντική υπερπαράμετρος όταν στήνεται το νευρωνικό δίκτυο. Το πρώτο όρισμα (argument) είναι οι παράμετροι που θα ενημερώνονται από τον optimizer ενώ το δεύτερο θα είναι ο ρυθμός εκμάθησης (learning rate).

```
import torch.optim as optim
optimizer = optim.SGD(model.parameters(), lr=1e-3)
```

Στην συνέχεια ορίστηκε η συνάρτηση απώλειας (loss function). Στην Pytorch αυτό συνήθως ονομάζεται κριτήριο (criterion). Η συνάρτηση απώλειας σε αυτό το μοντέλο είναι binary cross ethropy with logits (Brownlee, Loss and Loss Functions for Training Deep Learning Neural Networks, n.d.).

Το μοντέλο αυτή την στιγμή εκτελεί έναν απεριόριστο πραγματικό αριθμό(unbound real number). Καθώς τα labels είναι 0 ή 1, θέλουμε να περιορίσουμε τις προβλέψεις μεταξύ των αριθμών 0 και 1. Αυτό πραγματοποιείται χρησιμοποιώντας sigmoid ή logit functions (λειτουργίες).

Στην συνέχεια χρησιμοποιείται η bound scalar για να υπολογίσουμε την απώλεια χρησιμοποιώντας binary cross entropy.

Το `BCEWithLogitsLoss` κριτήριο εκτελεί sigmoid και binary cross enthropy βήματα (Gomez, n.d.).

```
criterion = nn.BCEWithLogitsLoss()
```

Χρησιμοποιώντας το .to μπορεί να τοποθετηθεί το μοντέλο και τα κριτήρια στην GPU.

```
model = model.to(device)
criterion = criterion.to(device)
```

Η λειτουργία κριτηρίων (criterion function) υπολογίζει την απώλεια, ωστόσο πρέπει να γραφεί η function για να υπολογίζει την ακρίβεια.

Αυτή η function πρώτα τροφοδοτεί τις προβλέψεις μέσω ενός sigmoid layer, squashing τις τιμές ανάμεσα στο 0 και 1, στην συνέχεια γίνεται στρογγυλοποίηση της τιμής στον πιο κοντινό ακέραιο αριθμό. Η στρογγυλοποίηση σε αριθμούς μεγαλύτερο του 0.5 γίνονται 1 (και θα είναι θετικό συναίσθημα και τα υπόλοιπα θα είναι 0 δηλαδή αρνητικό συναίσθημα.

Στην συνέχεια υπολογίζεται πόσες στρογγυλοποιημένες προβλέψεις ισούνται με τα πραγματικά labels (actual labels) και βγαίνει ο μέσος όρος σε όλο το batch.

```
def binary_accuracy(preds, y):
    """Επιστρέφει την ακρίβεια ανά batch, για παράδειγμα: αν πάρεις
σωστά 8/10, αυτό επιστρέφει 0.8 και όχι 8"""
    #Στρογγυλοποίηση προβλέψεων στον πλησιέστερο ακέραιο αριθμό
    rounded_preds = torch.round(torch.sigmoid(preds))
    correct = (rounded_preds == y).float() #μετατροπή σε float για διαίρεση
    acc = correct.sum() / len(correct)
    return acc
```

Η train function επαναλαμβάνεται σε όλα τα παραδείγματα για κάθε μία δέσμη (batch) την φορά.

Η κλάση model.train() χρησιμοποιείται για να βάλει το μοντέλο σε "training mode", η οποία ενεργοποιεί την dropout και batch normalization. Ωστόσο δεν χρησιμοποιείται σε αυτό το μοντέλο, αλλά γενικά είναι πολύ καλή πρακτική.

Για κάθε δέσμη (batch) αρχικά μηδενίζονται οι κλίσης(gradients). Αυτό πρέπει να γίνει διότι η Pytorch δεν αφαιρεί αυτόματα (ή μηδενίζει) την υπολογιζόμενες κλίσης (gradients calculated) από τον τελευταίο υπολογισμό κλίσης, άρα πρέπει να μηδενιστούν χειροκίνητα. Η

αποθήκευση της κλίσης που υπολογίζεται από το criterion γίνεται από το χαρακτηριστικό (attribute) grad .

Στην συνέχεια τροφοδοτείτε μία δέσμη από προτάσεις μέσω τις batch.text στο μοντέλο.

Σημείωση: Δεν χρειάζεται να χρησιμοποιηθεί το model.forward(batch.text), μπορεί να λειτουργήσει καλώντας απλά το μοντέλο.

Με την μέθοδος squeeze αφαιρείται η διάσταση με μέγεθος 1. Αρχικά οι προβλέπεις έχουν μέγεθος [**batch size, 1**] με την squeeze γίνετε [**batch_size**] όπου είναι το η αναμενόμενη είσοδος (για την Pytorch) στην συνάρτηση criterion.

Η απώλεια και η ακρίβεια υπολογίζονται χρησιμοποιώντας τις προβλέψεις και τις ετικέτες μέσω του batch.label, με την απώλεια να είναι ο μέσος όρος απ' όλα τα παραδείγματα στην δέσμη.

Υπολογίζετε η κλίση(gradient) κάθε παραμέτρου με την συνάρτηση loss.backward() και αφότου γίνει ο υπολογισμός, ενημερώνονται οι παραμέτροι χρησιμοποιώντας τις κλίσης(gradients) και τον optimizer αλγόριθμο με την συνάρτηση optimizer.step().

Η απώλεια και η ακρίβεια συσσωρεύονται(accumulated) μέσα στην εποχή (epoch). Εποχή ονομάζεται σε όλα τα στιγμιότυπα του συνόλου εκπαίδευσης και επαναλαμβάνεται μέχρι την ικανοποίηση κάποιου κριτηρίου τερματισμού (Κωστόπουλος, n.d.).

Η .item() μέθοδος χρησιμοποιείται για times να εξάγει μία σταθερά (scalar) από το διάνυσμα , το οποίο περιέχει μόνο μία τιμή.

Τέλος, επιστρέφεται η απώλεια και η ακρίβεια, με μέσο όρο σε όλη την εποχή.Το len ενός iterator είναι ο αριθμός των δεσμών στον iterator.

Όταν έγινε αρχικοποίηση στο πεδίο LABEL, ρυθμίστηκε το dtype=torch.fload. Αυτό έγινε γιατί το TorchText ρυθμίζει τα διανύσματα να είναι LongTensors (ακέραιο διάνυσμα) από προεπιλογή.

Ωστόσο τα κριτήρια αναμένουν τις εισόδους(inputs) να είναι floatTensors δηλαδή να είναι διάνυσμα κινητής υποδιαστολής. Η εναλλακτική μέθοδος να πραγματοποιηθεί κάτι τέτοιο είναι να γίνει μια μετατροπή μέσα στην συνάρτηση train, παίρνοντας την ρύθμιση για τιμές κινητής υποδιαστολής (by passing) στο κριτήριο το batch.label.float() αντί για batch.label.

```
def train(model, iterator, optimizer, criterion):
    epoch_loss = 0
```

```
    epoch_acc = 0
    model.train()
    for batch in iterator:
        optimizer.zero_grad()
        predictions = model(batch.text).squeeze(1)
        loss = criterion(predictions, batch.label)
        acc = binary_accuracy(predictions, batch.label)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
        epoch_acc += acc.item()
    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

Με την συνάρτηση evaluate θα αξιολογηθεί το μοντέλο. Η συνάρτηση αυτή είναι παρόμοια με την train, ωστόσο θα γίνουν μερικές τροποποιήσεις αφού δεν είναι χρήσιμο να ενημερώνει τις παραμέτρους όταν αξιολογεί(evaluate).

Η συνάρτηση model.eval() λειτουργεί σαν «διακόπτης» για κάποια στρώματα και σημεία του μοντέλου που συμπεριφέρονται διαφορετικά κατά τη διάρκεια της εκπαίδευσης και αξιολόγησης. Συγκεκριμένα, θα απενεργοποιηθεί η κλάση dropout και batch normalization. Ωστόσο χρησιμοποιούνται οι απενεργοποιημένες κλάσεις σε αυτό το μοντέλο αλλά είναι καλή πρακτική που θα την ακολουθήσουμε στα επόμενα μοντέλα.

Το υπόλοιπο της συνάρτησης είναι παρόμοιο με της train, αλλά χωρίς το optimizer.zero_grand(), loss.backward() και optimizer.steper(), γιατί δεν χρειάζεται και πάλι να γίνει ενημέρωση των παραμέτρων ενώ γίνεται αξιολόγηση. (evaluating).

Αξίζει να αναφερθεί ότι η Pytorch για λόγους ταχύτητας και κατανάλωσης λιγότερων πόρων δεν υπολογίζει καμία κλίση στο block with no_grand().

```
def evaluate(model, iterator, criterion):
    epoch_loss = 0
    epoch_acc = 0
    model.eval()
    with torch.no_grad():
        for batch in iterator:
            predictions = model(batch.text).squeeze(1)
            loss = criterion(predictions, batch.label)
            acc = binary_accuracy(predictions, batch.label)
```

```
            epoch_loss += loss.item()
            epoch_acc += acc.item()
    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

Δημιουργείται μια συνάρτηση για να γίνει σύγκριση στους χρόνους εκπαίδευσης ανάμεσα στα μοντέλα.

```
import time
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

Στην συνέχεια πραγματοποιείται εκπαίδευση το μοντέλο μέσα από πολλαπλές εποχές (epochs). Μια εποχή περνάει από όλα τα δείγματα στο σετ εκπαίδευσης και επαλήθευσης. Επίσης ορίζεται η απώλεια επαλήθευσης (validation loss) είναι η καλύτερη μέχρι στιγμής, να αποθηκεύσει τις παραμέτρους του μοντέλου με σκοπό όταν τελειώσει η εκπαίδευση να χρησιμοποιηθεί αυτό το μοντέλο στο σετ του τεστ (test set).

```
N_EPOCHS = 5
best_valid_loss = float('inf')
for epoch in range(N_EPOCHS):
    start_time = time.time()
    train_loss, train_acc = train(model, train_iterator, optimizer, criterio
n)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)
    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut1-model.pt')
    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%
')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%
')
```

```
Epoch: 01 | Epoch Time: 0m 18s
	Train Loss: 0.694 | Train Acc: 50.33%
	 Val. Loss: 0.697 |  Val. Acc: 49.86%
Epoch: 02 | Epoch Time: 0m 15s
	Train Loss: 0.693 | Train Acc: 50.06%
	 Val. Loss: 0.697 |  Val. Acc: 49.86%
Epoch: 03 | Epoch Time: 0m 15s
	Train Loss: 0.693 | Train Acc: 49.93%
	 Val. Loss: 0.697 |  Val. Acc: 50.86%
Epoch: 04 | Epoch Time: 0m 15s
	Train Loss: 0.693 | Train Acc: 49.49%
```

```
         Val. Loss: 0.697 |  Val. Acc: 49.65%
Epoch: 05 | Epoch Time: 0m 16s
        Train Loss: 0.693 | Train Acc: 50.13%
         Val. Loss: 0.697 |  Val. Acc: 50.87%
```

Παρατηρείται ότι η απώλεια δεν μειώνεται και η ακρίβεια είναι χαμηλή. Κάτι που θα διορθωθεί στο επόμενο κεφάλαιο.

Τέλος, τυπώνονται οι μετρήσεις που είναι ενδιαφέρον δηλαδή η απώλεια δοκιμής και η ακρίβεια τα οποία θα συλλεχθούν από τις παραμέτρους που έδωσαν την καλύτερη απώλεια επικύρωσης.

```
model.load_state_dict(torch.load('tut1-model.pt'))
test_loss, test_acc = evaluate(model, test_iterator, criterion)
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
Test Loss: 0.710 | Test Acc: 47.96%
```

4 Επαναλαμβανόμενα Νευρωτικά Δίκτυα με PyTorch - Διαφορετικοί μέθοδοι για καλύτερα αποτελέσματα

Στο προηγούμενο κεφάλαιο, αποτυπώθηκαν οι βασικές αρχές για της Ανάλυσης συναισθήματος χρησιμοποιώντας το framework Pytorch και τα Επαναλαμβανόμενα Νευρωτικά Δίκτυα (ΕΝΔ). Σε αυτό το κεφάλαιο, με τις ίδιες αρχές θα υπάρχουν καλύτερα αποτελέσματα λόγω ότι θα χρησιμοποιηθούν:

- packed padded sequences
- pre-trained word embeddings
- διαφορετική ΕΝΔ αρχιτεκτονική
- bidirection RNN
- multi-layer RNN
- regularization
- διαφορετικός optimizer

Τα παραπάνω θα μας επιτρέψουν να έχουμε γύρω στο ~84% test accuracy.

4.1 Προετοιμάζοντας τα δεδομένα

Όπως και προηγούμενος ρυθμίζεται ο seed, θα οριστεί το fields κα θα παρθούν τα train/valid/test splits. Θα χρησιμοποιηθούν οι packed padded sequences, οι οποίες θα κάνουν τον RNN να επεξεργαστεί μόνο τα non-padded elements στην ακολουθίας , ενώ για τα padded

elements το output θα είναι ένας μηδενικός tensor. Για να χρησιμοποιηθούν οι packed padded sequences, πρέπει να πούμε στο ΕΝΔ πόσο μεγάλες είναι η πραγματικές ακολουθίες. Αυτό γίνεται ρυθμίζοντας το include_lengths = True από το TEXT field. Αυτό, θα προκαλέσει το batch.text να είναι πλειάδα(tuple) με το πρώτο στοιχείο της ακολουθίας(ένας numericalized tensor έχει γίνει padded) και το δεύτερο στοιχείο να είναι το πραγματικό μήκος της ακολουθίας.

```
!pip install -U torchtext==0.10.0
```

```
import torch
from torchtext.legacy import data
from torchtext.legacy.data import Field, TabularDataset, BucketIterat
or, Iterator
# Αρχικοποιούμε την γεννήτρια τυχαίων αριθμών με σταθερά για να διασφαλίσουμε
ότι τα αποτελέσματα θα είναι αναπαραγωγίσιμα.
SEED = 1234

torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True

TEXT = data.Field(tokenize = 'spacy',
                  tokenizer_language = 'en_core_web_sm')
LABEL = data.LabelField(dtype = torch.float)
```

Φορτώνεται το IMDB dataset

```
from torchtext.legacy import datasets
train_data, test_data = datasets.IMDB.splits(TEXT, LABEL)
```

Στην συνέχεια δημιουργείτε το validation set από το training set.

```
import random
train_data, valid_data = train_data.split(random_state = random.seed(SEED))
```

Χρησιμοποιείται το pre-train word embeddings. Αντί οι λέξεις να αρχικοποιούνται τυχαία, αρχικοποιούνται με αυτούς τους pre-trained vectors. Προσδιορίζεται ποιοι vectors θα χρησιμοποιηθούν και το περνιέται ως μια arguement στο build_vocab.TorchText η οποία μέθοδος αναλαμβάνει να κατεβάσει τους vectors και να τους συσχετίσει με τις σωστές λέξεις στο λεξιλόγιο. Θα χρησιμοποιηθεί επίσης το glove.6b.100d"vectors".glove ο οποίος είναι ένας αλγόριθμος που υπολογίζει τους vectors (Jeffrey Pennington, Richard Socher, Christopher D. Manning , n.d.). Το 6B δείχνει ότι αυτοί οι vectors έχουν γίνει train σε 6 Billions tokens και το 100d δείχνει ότι αυτοί οι vectors έχουν 100 διαστάσεις(100-dimensional).

Εναλλακτικοί διαθέσιμοι vectors εδώ.

Στην θεωρία, αυτοί οι pre-trained vectors έχουν λέξης με παρόμοια σημασιολογική σημασία κοντά στον vector space (close together to vector space). Για παράδειγμα: "terrible", "awful", "dreadful" είναι κοντινές λέξεις. Αυτό δίνει στο embedding layer μας μια καλή αρχικοποίηση μιας και δεν έχει να μάθει αυτές τις σχέσεις από το μηδέν.

Σημείωση: αυτοί οι vectors έχουν μέγεθος περίπου 862MB

Από προεπιλογή, το TorchText θα αρχικοποιήση τις λέξεις στο λεξιλόγιο στο 0, αλλά όχι στο pre-traned embedding το οποίο δεν είναι ιδανικό. Άρα θα αρχικοποιηθούν τυχαία ρυθμίζοντας το unk_init σε torch.Tensor.normal_. Αυτό θα αρχικοποιήσει τις λέξεις μέσω μίας Guassian διανομής(Guassian distribution).

```
MAX_VOCAB_SIZE = 25_000
TEXT.build_vocab(train_data,
                 max_size = MAX_VOCAB_SIZE,
                 vectors = "glove.6B.100d",
                 unk_init = torch.Tensor.normal_)
LABEL.build_vocab(train_data)
```

```
.vector_cache/glove.6B.zip: 862MB [00:34, 24.9MB/s]
 99%|█████████▉| 397278/400000 [00:13<00:00, 28949.05it/s]
```

Όπως και στο προηγούμενο κεφάλαιο, δημιουργήθηκαν οι iterators και τοποθετήθηκαν οι tensors σε GPU.

Άλλο ένα πράγμα για τις packed padded sequences, όλοι οι tensors μέσα στο batch θα πρέπει να ταξινομούνται βάση το μήκος του. Αυτό, το διαχειρίζεται στον itelator όταν ρυθμίζεται το sort_within_batch = True.

```
BATCH_SIZE = 64
device = torch.device('cuda')
train_iterator, valid_iterator, test_iterator = data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = BATCH_SIZE,
    sort_within_batch = True,
    device = device)
```

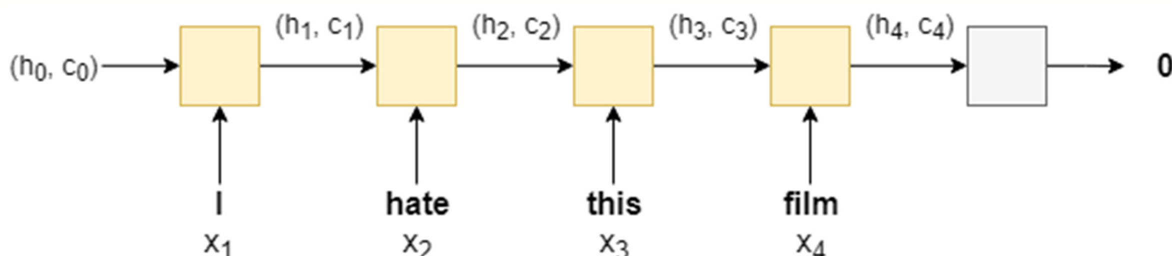
4.2 Χτίζοντας το μοντέλο

Αυτό το μοντέλο διαθέτει πιο δραστικές αλλαγές και διαφορετική RNN αρχιτεκτονική. Αυτή η αρχιτεκτονική ονομάζεται Long Short-Term Memory (LSTM).

Η τυπική αρχιτεκτονική των ΕΝΔ πάσχει από το vanishing gradient πρόβλημα (Wikipedia, n.d.). Η LSTM αρχιτεκτονική το ξεπερνάει αυτό έχοντας ένα επιπλέον recurrent state που ονομάζεται "ένα κελί(a cell)", $c -$. Το οποίο μπορεί να θεωρηθεί ως "μνήμη" του LSTM και η χρήση αυτού του κελιού είναι να χρησιμοποιεί πολλαπλές πύλες(gates) όπου ελέγχουν την ροή πληροφοριών μέσα και έξω στην μνήμη (Colah, n.d.). Η LSTM αρχιτεκτονική αποτυπώνεται ως συνάρτηση x_t, h_t, c_t , αντί για μόνο το x_t, h_t.

$$(h_t, c_t) \ = \ LSTM \ (x_t, h_{t,} c_t)$$

Επιπλέον, το μοντέλο όταν χρησιμοποιεί LSTM αρχιτεκτονική μοιάζει έτσι(με τα embedding layers να παραλείπονται)



Η αρχική cell state c_0, όπως και η αρχική του hidden state, αρχικοποιείται από έναν tensor με μόνο μηδενικά. Η πρόβλεψη συναισθήματος παραμένει, ωστόσο γίνετε χρησιμοποιώντας το τελικό hidden state και όχι το τελικό cell state. Δηλαδή $\hat{y} \ = f \ (h_T)$

Vanishing/Exploding Gradients Πρόβλημα

Ένα γενικό θέμα των Επαναλαμβανόμενων Νευρωνικών Δικτύων είναι γνωστό ως vanishing/exploding gradients πρόβλημα.

Το πρόβλημα δηλώνει ότι για μεγάλες ακολουθίες εισόδου - εξόδου, οι RNNs αντιμετωπίζουν προβλήματα μοντελοποίησης long-term dependencies, η σχέση μεταξύ των στοιχείων της ακολουθίας που χωρίζονται από μεγάλες χρονικές περιόδους.

Για παράδειγμα, η πρόταση "The quick brown fox jumped over the lazy dog, οι λέξεις “fox” και “dog” διαχωρίζονται από ένα μεγάλο μέρος του χώρου στην ακολουθία. Στη απελευθέρωση ενός ΕΝΔ για αυτή την ακολουθία, η παραπάνω πρόταση θα μοντελοποιηθεί

με μια μεγάλη διαφορά στο Δt σε χρόνο x_a για την αρχή της λέξης "fox" και $\Delta t + x_a$ για το τέλος της λέξης "dog".

Έτσι, αν ένας RNN προσπαθεί να μάθει πως να εντοπίζει θέματα και αντικείμενα σε προτάσεις, θα πρέπει να θυμάται την λέξη "fox" (ή κάποια hidden state που την αντιπροσωπεύει), δηλ. το θέμα, μέχρι να διαβάσει την λέξη "dog", δηλ το αντικείμενο. Μόνο τότε το ΕΝΔ θα μπορεί να έχει έξοδο το ζευγάρι ("fox", "dog"), έχοντας επιτέλους εντοπίσει ένα θέμα και ένα αντικείμενο.

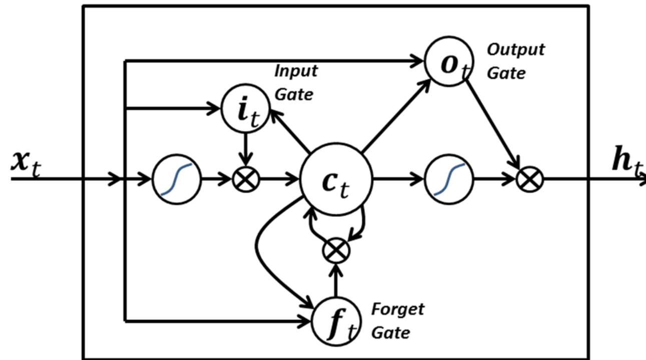
Αυτο προκαλεί στην μάθηση είτε να γίνει πολύ αργή (στην περίπτωση του vanishing) ή εξαιρετικά ασταθής (στην περίπτωση του exploding)

Long Short-term Memory

Ευτυχώς, οι πρόσφατες παραλλαγές των ΕΝΔ, όπως το LSTM (Long Short-Term Memory), κατάφεραν να ξεπεράσουν το vanishing/exploding gradient πρόβλημα, Έτσι ώστε τα ΕΝΔ να μπορούν να εφαρμοστούν με ασφάλεια σε εξαιρετικά μακρές ακολουθίες, ακόμη και αυτές που περιέχουν εκατομμύρια στοιχεία. Στην πραγματικότητα, τα LSTM που αντιμετωπίζουν το gradient πρόβλημα ευθύνονται σε μεγάλο βαθμό για τις πρόσφατες επιτυχίες σε πολύ βαθιές εφαρμογές NLP.

Τα LSTM ΕΝΔ δουλεύουν με το να επιτρέπουν την είσοδο x_tσε χρόνο t να επηρεάσει την αποθήκευση ή την αντικατάσταση των "μνημών" που είναι αποθηκευμένα σε κάτι που ονομάζεται cell. Η απόφαση αυτή καθορίζεται από δύο διαφορετικές λειτουργίες, που ονομάζονται πύλη εισόδου(input gate) για την αποθήκευση νέων αναμνήσεων, και η forget gate για να ξεχάσουμε τις παλιές αναμνήσεις. Μία τελική output gate καθορίζει πότε να εξάγει την τιμή που είναι αποθηκευμένη στο cell μνήμης/memory cell του hidden layer. Αυτές οι gates ελέγχονται από τις τρέχουσες τιμές της εισόδου x_t και του cell c_t σε χρόνο t, συν κάποιες παράμετροι που ειναι gate-specific.

Η παρακάτω εικόνα απεικονίζει τον Γράφο υπολογισμού για το τμήμα μνήμης ενός LSTM ΕΝΔ (δηλ. Δεν περιλαμβάνει το hidden layer ή output layer).



Ενώ η γενική συνταγή του ΕΝΔ μπορεί θεωρητικά να μάθει τις ίδιες λειτουργίες με ένα LSTM ΕΝΔ, περιορίζοντας τη μορφή που μπορούν να πάρουν οι μνήμες και τον τρόπο με τον οποίο τροποποιούνται, οι LSTM μπορούν να μάθουν long-term dependencies γρήγορα και σταθερά και έτσι είναι πολύ πιο χρήσιμες στην πράξη.

Bidirectional RNN(Αμφίδρομος RNN)

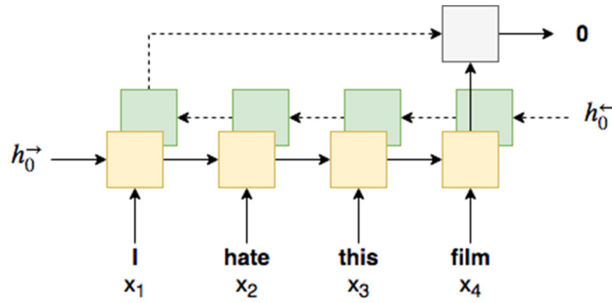
Η ιδέα πίσω από έναν bidirectional RNN είναι απλή. Έχοντας ένα ΕΝΔ να επεξεργάζεται τις λέξεις σε μια πρόταση από την πρώτη λέξη στην τελευταία (a forward RNN) και έχουμε έναν δεύτερο ΕΝΔ που κάνει το αντίθετο, δηλαδή επεξεργάζεται τις λέξεις σε μια πρόταση από την τελευταία στην πρώτη(a backward RNN). Στο χρονικό βήμα(time step) t, ο forward RNN επεξεργάζεται την λέξη x_t, και ο backward RNN επεξεργάζεται την λέξη x_{T-t+1}.

Στην Pytorch, οι tensors της κρυφή κατάσταση (και η cell state) επιστρέφονται από τον forward και backward RNN οι οποίοι είναι stacked ο ένας πάνω στον άλλον σε έναν και μόνο tensor.

Κάνοντας την πρόβλεψη του συναισθήματος χρησιμοποιώντας την αλληλουχία της τελευταίας κρυφής κατάστασης από τον forward RNN(η οποία λήφθηκε από την τελευταία λέξη της πρότασης) $h_{\vec{T}}$, και την τελευταία hidden state/κρυφή κατάσταση από τον backward RNN(η οποία λήφθηκε από την πρώτη λέξης της πρότασης)$h_{\overleftarrow{T}}$.

Δηλαδή: $\hat{y} = f\left(h_{\vec{T}}, h_{\overleftarrow{T}}\right)$

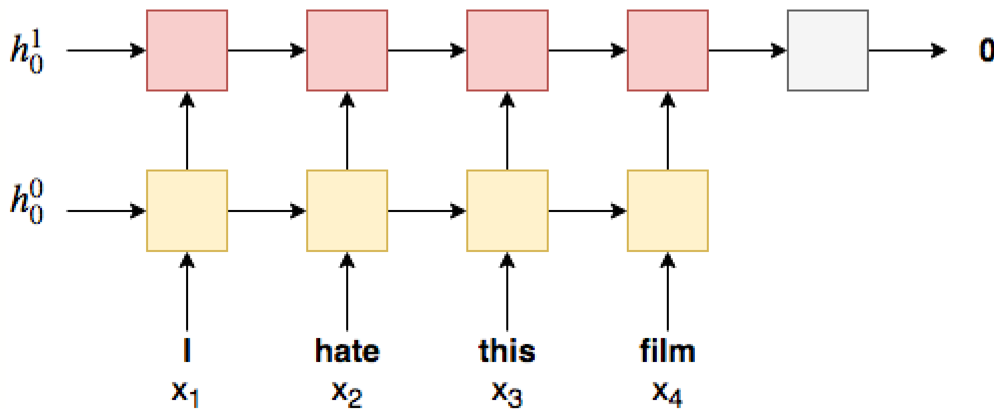
Η παρακάτω εικόνα δείχνει έναν bi-directional RNN, ο forward RNN απεικονίζεται με χρώμα πορτοκαλί, ο backward RNN σε πράσινο και το γραμμικό στρώμα(linear layer) με χρώμα ασημί.



Multi-layer RNN

Η ιδέα πίσω από τον Multi-layer (επίσης συχνά καλείτε deep RNNs) είναι ότι προσθετεί επιπλέον ΕΝΔ πάνω στο αρχικό τυπικό ΕΝΔ, όπου κάθε ΕΝΔ που προστέθηκε είναι άλλο ένα layer. Η έξοδος της κρυφή κατάστασης από το πρώτο (bottom) ΕΝΔ στο χρονικό βήμα (time-step) θα είναι η είσοδος στον από πάνω ΕΝΔ στο χρονικό βήμα t. Στην συνέχεια, η πρόβλεψη γίνεται από την τελική κρυφή κατάσταση του τελικού(υψηλότερου) layer.

Η παρακάτω εικόνας δείχνει ένα multi-layer unidirectional RNN, όπου ο αριθμός των layers δίνεται ως ένα superscript. Επίσης να σημειωθεί ότι κάθε layer χρειάζεται την δικιά του αρχική κρυφή κατάσταση h_0^L.



Τακτοποίηση (Regularization)

Ενώ προστέθηκαν βελτιώσεις στο μοντέλο, κάθε μία από αυτές προσθέτει μια επιπλέον παράμετρο. Χωρίς πολύ λεπτομέρεια, όσες παραπάνω παραμέτρους έχουμε στο μοντέλο , τόσο υψηλότερη είναι η πιθανότητα το μοντέλο να overfit (να απομνημονεύουν τα training data, προκαλώντας χαμηλό training error αλλά υψηλό validation/testing error, δηλαδή: φτωχή γενίκευση σε νέα παραδείγματα). Για να το εξαλειφθεί αυτό χρησιμοποιείται η τακτοποίηση.

Πιο συγκεκριμένα, θα χρησιμοποιηθεί μια μέθοδος για regularization η οποία ονομάζεται dropout. Η dropout δουλεύει με την απόρριψη (την ρυθμίζουμε στο 0) των νευρώνων σε ένα

layer κατά την διάρκεια ενός pass προς τα εμπρός(forward pass). Η πιθανότητα απόρριψης κάθε νευρώνα ρυθμίζεται από μία υπέρ-παράμετρο και κάθε νευρώνας που έχει απορριφθεί θεωρείτε ανεξάρτητος.

Μία θεωρία γιατί η απόρριψη/dropout δουλεύει είναι ότι ένα μοντέλο με παραμέτρους που απορρίφθηκε μπορεί να θεωρηθεί ως πιο "αδύναμο"(με λιγότερες παραμέτρους) μοντέλο. Η πρόβλεψη αυτών των "αδύναμων" μοντέλων (ένα για κάθε forward pass) υπολογίζονται κατά μέσο όρο μέσα στους παραμέτρους του μοντέλου. Έτσι το μοντέλο μας μπορεί να θεωρηθεί ως ένα σύνολο αδύνατων μοντέλων, όπου κανένα από αυτά δεν είναι over-parameterized και έτσι δεν πρέπει να overfit.

4.3 Λεπτομέρειες υλοποίησης

Άλλη μια αλλαγή που έγινε σε αυτό το μοντέλο, είναι ότι δεν θα χρησιμοποιηθεί το embedding για το <pad> token. Αυτό δεν θα γίνει διότι στόχος είναι να αποτυπώσουμε στο μοντέλο ότι τα padding tokens είναι άσχετα για να προσδιορίσουν το συναίσθημα στην πρόταση. Αυτό σημαίνει ότι προς το παρών το embedding για τα pad tokens θα παραμείνουν στη παραμείνει σε αυτό που έχει αρχικοποιηθεί. Το κάνουμε αυτό με το να περνάμε το index των pad token ως padding_idx argument στο nn.Embedding layer.

Για να χρησιμοποιηθεί ένα LSTM αντί για έναν κανονικό ΕΔΝ, χρησιμοποιείται η μέθοδο nn.LSTM αντί για nn.RNN. Επίσης, ο LSTM επιστρέψει το output και μια πλειάδα από τις τελικές κρυφές κατάστασης και τις τελικές cell states, ο τυπικό ΕΝΔ επέστρεφε μόνο το output και το τελικό hidden state.

Καθώς η τελική κρυφή κατάσταση του LSTM μας έχει και forward και backward component, τα οποία θα ενωθούν, το μέγεθος της εισόδου του nn.Linear layer είναι διπλάσιο από τη κρυφή διάσταση.

Η υλοποίηση της bidirectionality και η πρόσθεση επιπλέον στρωμάτων γίνεται με την μεταφορά τιμών(by passing values) στην argument bidirectinal και num_layers για τον RNN/LSTM.

Η dropout υλοποιείται με το να αρχικοποιηθεί ένα nn.Dropout στρώμα(αυτή η argument είναι η πιθανότητα απόρριψης/dropping out για κάθε νευρώνα) και χρησιμοποιώντας την forward μέθοδος μετά από κάθε layer που εφαρμόζεται η dropout.

Σημείωση: ποτέ δεν χρησιμοποιείτε η dropout σε input και output layers(text **ή** fc **σε αυτή την περίπτωση), σε αυτή την υλοποίηση χρησιμοποιείται η dropout στα ενδιάμεσα/intermediate layers.**

Η LSTM αρχιτεκτονική έχει την dropout argument που προσθέτει dropout στην σύνδεση μεταξύ των κρυφών καταστάσεων ενός layer και των κρυφών καταστάσεων του επόμενου layer.

Καθώς περνιούνται τα μήκη των προτάσεων για να μπορούν να χρησιμοποιηθούν οι packed padded sequences, πρέπει να προσθέτει ένα δεύτερο όρισμα text_lengths στο forward.

Πριν οριστούν τα embeddings στο ΕΝΔ, χρειάζεται να συγκεντρωθούν, αυτό πραγματοποιείται με το nn.utils.rnn.packed_paded_sequence όπου θα αναγκάσει το ΕΝΔ να επεξεργαστεί μόνο τα non-padded στοιχεία στην ακολουθίας. Το ΕΝΔ θα επιστρέψει packed_output (μια packed ακολουθία), όπως επίσης θα επιστρέψει την κρυφή και cell κατάσταση (και τα 2 είναι tensors). Χωρίς packed padded sequences, η hidden και η cell είναι tensors από τα τελευταία στοιχεία, το ποια στοιχεία το πιθανότερο είναι να είναι pad token. Ωστόσο, όταν χρησιμοποιούνται packed padded sequences είναι και οι δύο καταστάσεις (hidden, cell) από το τελευταίο non-padded στοιχείο στην ακολουθία.

Στην συνέχεια πραγματοποιείται unpack στην ακολουθία εξόδου με το nn.utils.rnn.pad_packed_sequence, για να γίνει μετατροπή από packed ακολουθία σε tensor. Το στοιχείο του output από τα padding tokens θα είναι μηδενικοί tensors (δηλ. tensors όπου κάθε στοιχείο είναι 0). Τυπικά πραγματοποιείται unpack μόνο την έξοδο που πρόκειται να χρησιμοποιήσουμε αργότερα στο μοντέλο.

Η τελική κρυφή κατάσταση, έχει σχήμα από τα [num layer * num directions, batch size, hid dim]. Αυτά διατάσσονται [forward_layer_0, backward_layer_0, forward_layer_1, backward_layer 1, ..., forward_layer_n, backward_layer n]. Εκεί παίρνεται το τελικό(στην κορυφή) layer forward και τις backward κρυφές καταστάσεις και τα κρυφά στρώματα που βρίσκονται στην κορυφή από την πρώτη διάσταση, hidden[-2,:,:] και hidden[-1,:,:].

Τέλος, τα ενώνονται μαζί πριν τα περαστούν στο γραμμικό layer (αφότου έχει εφαρμοστεί η dropout).

```
import torch.nn as nn
class RNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_
layers,
```

```
                  bidirectional, dropout, pad_idx):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx
 = pad_idx)
        self.rnn = nn.LSTM(embedding_dim,
                           hidden_dim,
                           num_layers=n_layers,
                           bidirectional=bidirectional,
                           dropout=dropout)
         self.fc = nn.Linear(hidden_dim * 2, output_dim)
         self.dropout = nn.Dropout(dropout)
    def forward(self, text, text_lengths):
        #το text ισούται = [sent len, batch size]
        embedded = self.dropout(self.embedding(text))
        #το embedded ισούται = [sent len, batch size, emb dim]
        #pack sequence
        packed_embedded = nn.utils.rnn.pack_padded_sequence(embedded, t
ext_lengths)
        packed_output, (hidden, cell) = self.rnn(packed_embedded)
        #unpack sequence
        output, output_lengths = nn.utils.rnn.pad_packed_sequence(packed_out
put)
        #το                                                          output
ισούται = [sent len, batch size, hid dim * num directions]
        #Η έξοδος από τα padding tokens είναι μηδενική tensors
        #το
hiddenισούται = [num layers * num directions, batch size, hid dim]
        #το                                                            cell
ισούται = [num layers * num directions, batch size, hid dim]
        #Συνδυασμούς του τελικούforward (hidden[-
2,:,:]) με backward (hidden[-1,:,:]) κρυφά layers
        #και εφαρμογή της dropout
        hidden = self.dropout(torch.cat((hidden[-2,:,:], hidden[-
1,:,:]), dim = 1))
        #το hidden ισούται = [batch size, hid dim * num directions]
        return self.fc(hidden)
```

Όπως προηγουμένως, θα δημιουργηθεί ένα instance της RNN κλάσης, με νέες παραμέτρους και arguments για τον αριθμό των layer, bidirectionality και την πιθανότητα dropout. Για να γίνει επιβεβαίωση ότι οι pre-trained vectors μπορούν να φορτωθούν στο μοντέλο, το

EMBEDDING_DIM πρέπει να είναι ίσο με τα pre-trains του GloVe vectors που φορτώθηκαν πιο πάνω. Συλλέγονται τα pad token index από το λεξιλόγιο και το πραγματικό string που αντιπροσωπεύει το pad token από το πεδίο pad_token attribute, το οποίο είναι pad από προεπιλογή.

```
INPUT_DIM = len(TEXT.vocab)
EMBEDDING_DIM = 100
HIDDEN_DIM = 256
OUTPUT_DIM = 1
N_LAYERS = 2
BIDIRECTIONAL = True
DROPOUT = 0.5
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]
model = RNN(INPUT_DIM,
            EMBEDDING_DIM,
            HIDDEN_DIM,
            OUTPUT_DIM,
            N_LAYERS,
            BIDIRECTIONAL,
            DROPOUT,
            PAD_IDX)
```

Τώρα, θα τυπώνεται ο αριθμός των παραμέτρων που έχει το μοντέλο. Παρατηρείτε σχεδόν ο διπλάσιος αριθμός από την υλοποίηση του προηγούμενου κεφαλαίου.

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
print (f'The model has {count_parameters(model):,} trainable parameters')
```

```
The model has 4,810,857 trainable parameters
```

Η τελική προσθήκη είναι η αντιγραφή τις pre-trained word embeddings που φορτώθηκε νωρίτερα στο embedding layer του μοντέλου.

Ανακτήθηκε στις embeddings από το πεδίο του λεξικού, και γίνεται έλεγχος ότι έχουν το σωστό μέγεθος [vocab size, embedding dim]

```
pretrained_embeddings = TEXT.vocab.vectors
print(pretrained_embeddings.shape)
torch.Size([25002, 100])
```

```
model.embedding.weight.data.copy_(pretrained_embeddings)
```

```
tensor([[-0.1117, -0.4966,  0.1631,  ...,  1.2647, -0.2753, -0.1325],
        [-0.8555, -0.7208,  1.3755,  ...,  0.0825, -1.1314,  0.3997],
        [-0.0382, -0.2449,  0.7281,  ..., -0.1459,  0.8278,  0.2706],
        ...,
        [-1.5217, -1.2295,  2.3400,  ..., -0.1066, -1.5500,  1.2515],
        [ 1.8250,  1.1612,  1.9405,  ..., -0.6524,  0.5291,  0.0298],
        [ 0.4337,  1.4285,  1.0520,  ..., -0.0274, -0.8621,  0.8455]])
```

Στην συνέχεια θα πρέπει να αντικαθιστούν τα αρχικά weights του embedding layer με τα pre-trained embeddings.

Καθώς το unk και pad token δεν είναι στο pre-trained λεξιλόγιο, έχουν αρχικοποιηθεί χρησιμοποιώντας το unk_unit ως $N(0,1)$ όταν χτίζεται το λεξικό. Είναι προτιμότερο να αρχικοποιηθούν και τα 2 tokens σε όλα τα μηδενικά και να ορισθεί στο μοντέλο. Αρχικά είναι άσχετα με τον προσδιορισμό του συναισθήματος. Αυτό πραγματοποιείται ρυθμίζοντας χειροκίνητα τις σειρές στον πίνακα των embedding weights να είναι όλα μηδενικά. Παίρνοντας την σειρά τους με το να βρίσκοντας το index των tokens που έχουμε ήδη κάνει για το padding index.

```
UNK_IDX = TEXT.vocab.stoi[TEXT.unk_token]
model.embedding.weight.data[UNK_IDX] = torch.zeros(EMBEDDING_DIM)
model.embedding.weight.data[PAD_IDX] = torch.zeros(EMBEDDING_DIM)
print(model.embedding.weight.data)
```

```
tensor([[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [-0.0382, -0.2449,  0.7281,  ..., -0.1459,  0.8278,  0.2706],
        ...,
        [-1.5217, -1.2295,  2.3400,  ..., -0.1066, -1.5500,  1.2515],
        [ 1.8250,  1.1612,  1.9405,  ..., -0.6524,  0.5291,  0.0298],
        [ 0.4337,  1.4285,  1.0520,  ..., -0.0274, -0.8621,  0.8455]])
```

Τώρα δίνεται η δυνατότητα να εμφανιστεί στον πίνακα των embedding weights στις δύο πρώτες σειρές ότι είναι όλα μηδενικά. Καθώς περνιέται το index των pad token στο padding_idx του embedding layer, θα παραμείνει με όλα να είναι μηδενικά κατά την διάρκεια του training, ωστόσο το <unk> token embedding θα εκπαιδευτεί.

4.4 Εκπαιδεύοντας το μοντέλο

Η μόνη αλλαγή που θα γίνει τώρα είναι στον optimizer από SGD σε Adam. O SGD ενημερώνει όλες τις παραμέτρους με το ίδιο βαθμό εκπαίδευσης και η επιλογή του μπορεί να είναι δύσκολη.

Ο Adam προσαρμόζει το βαθμό εκπαίδευσης για κάθε παράμετρο, δίνοντας στις παραμέτρους την δυνατότητα που είναι ενημερωμένες πιο συχνά με χαμηλό βαθμό εκπαίδευσης και στις παραμέτρους που ενημερώνονται με μικρότερη συχνότητα να έχουν μεγαλύτερο βαθμό εκπαίδευσης. (Kumar, n.d.)

Για να αλλαχθεί ο optimizer από SGD σε Adam είναι εύκολη μετατροπή στον κωδικό. Το optim.SGD γίνεται σε optim.Adam, επίσης να σημειώνεται ότι δεν παρέχεται ο αρχικός ρυθμός εκπαίδευσης για τον Adam, καθώς η Pytorch καθορίζει το αρχικό βαθμό εκπαίδευσης.

```
import torch.optim as optim
optimizer = optim.Adam(model.parameters())
```

Τα υπόλοιπα βήματα για την εκπαίδευση του μοντέλου δεν αλλάζουν. Ορίζονται τα κριτήρια και τοποθετείται το μοντέλο και τα κριτήρια στην GPU.

```
criterion = nn.BCEWithLogitsLoss()
model = model.to(device)
criterion = criterion.to(device)
Υλοποιούμε την fuction που υπολογίζει το accuracy/ακρίβεια
def binary_accuracy(preds, y):
    """
    Returns accuracy per batch, i.e. if you get 8/10 right, this returns 0.8
, NOT 8
    """
    # στρογγυλοποίηση προβλέψεων στον πλησιέστερο ακέραιο αριθμό
    rounded_preds = torch.round(torch.sigmoid(preds))
    correct = (rounded_preds == y).float() # μετατροπή σε float για διαίρεση
    acc = correct.sum() / len(correct)
    return acc
```

Ορίζεται μία συνάρτηση που εκπαιδεύει το μοντέλο.

Καθώς ορίζεται το include_lengths = True στο batch.text μετατρέπεται σε μία πλειάδα όπου το πρώτο στοιχεία να είναι ο numericalized tensor και το δεύτερο στοιχείο να είναι το

πραγματικό μήκος για κάθε ακολουθία. Αυτά χωρίζονται στις δικές τους μεταβλητές, text και text_lengths αντίστοιχα. Το παραπάνω γίνεται πριν περαστούν στο μοντέλο.

Σημείωση: αφού σε αυτό το μοντέλο χρησιμοποιείται η dropout, θα πρέπει να χρησιμοποιηθεί το model.train() για να εξασφαλιστεί ότι η dropout είναι ενεργοποιημένη ενώ γίνεται εκπαίδευση.

```
def train(model, iterator, optimizer, criterion):
    epoch_loss = 0
    epoch_acc = 0
    model.train()
    for batch in iterator:
        optimizer.zero_grad()
        text, text_lengths = batch.text
        predictions = model(text, text_lengths).squeeze(1)
        loss = criterion(predictions, batch.label)
        acc = binary_accuracy(predictions, batch.label)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
        epoch_acc += acc.item()
    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

Στην συνέχεια ορίζεται μία συνάρτηση για να ελεγχθεί το μοντέλο , πάλι πρέπει να χωριστεί το batch.text.

Σημείωση: αφού σε αυτό το μοντέλο χρησιμοποιείται η dropout, να χρησιμοποιειθεί το model.eval() για να εξασφαλιστεί ότι η dropout είναι ενεργοποιημένη ενώ γίνεται εκπαίδευση

```
def evaluate(model, iterator, criterion):
    epoch_loss = 0
    epoch_acc = 0
    model.eval()
    with torch.no_grad():
        for batch in iterator:
            text, text_lengths = batch.text
            predictions = model(text, text_lengths).squeeze(1)
            loss = criterion(predictions, batch.label)
```

```
            acc = binary_accuracy(predictions, batch.label)
            epoch_loss += loss.item()
            epoch_acc += acc.item()
    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

Επίσης δημιουργείται μια συνάρτηση για δείξει πει πόση ώρα τα epoch κάνουν

```
import time
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

Τέλος, πραγματοποιείται η εκπαίδευση του μοντέλου

```
N_EPOCHS = 5
best_valid_loss = float('inf')
for epoch in range(N_EPOCHS):
    start_time = time.time()
    train_loss, train_acc = train(model, train_iterator, optimizer, criterio
n)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)
    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut2-model.pt')
    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%
')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%
')
```

```
Epoch: 01 | Epoch Time: 0m 34s
        Train Loss: 0.634 | Train Acc: 63.41%
         Val. Loss: 0.637 |  Val. Acc: 66.14%
Epoch: 02 | Epoch Time: 0m 36s
        Train Loss: 0.576 | Train Acc: 70.29%
         Val. Loss: 0.759 |  Val. Acc: 54.56%
Epoch: 03 | Epoch Time: 0m 37s
        Train Loss: 0.453 | Train Acc: 79.10%
         Val. Loss: 0.850 |  Val. Acc: 73.14%
Epoch: 04 | Epoch Time: 0m 38s
        Train Loss: 0.387 | Train Acc: 83.50%
         Val. Loss: 0.391 |  Val. Acc: 84.06%
Epoch: 05 | Epoch Time: 0m 38s
        Train Loss: 0.327 | Train Acc: 86.07%
         Val. Loss: 0.314 |  Val. Acc: 86.69%
```

και τυπώνονται τα βελτιωμένα αποτελέσματα μας

```
model.load_state_dict(torch.load('tut2-model.pt'))
test_loss, test_acc = evaluate(model, test_iterator, criterion)
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Test Loss: 0.337 | Test Acc: 87.78%
```

5 Ανάλυση Συναισθήματος με το FastText μοντέλο

Στα προηγούμενα κεφάλαια επιτεύχθηκε να υπάρχει καλή ακρίβεια στο τεστ με ποσοστό 84%, χρησιμοποιώντας όλες τις συνηθές τεχνικές που εφαρμόζονται για ανάλυση συναισθήματος.

Σε αυτό το κεφάλαιο θα υλοποιηθεί ένα μοντέλο που δίνει αντίστοιχα αποτελέσματα ενώ η εκπαίδευση του μοντέλου είναι πιο γρήγορη και χρησιμοποιούνται οι μισές παραμέτροι. Πιο συγκεκριμένα, θα υλοποιηθεί το "FastText" μοντέλο (Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov, 2016).

5.1 Προετοιμασία Δεδομένων

Μία από τις βασικές ιδέες στο FastText paper είναι ότι υπολογίζουν τα n-grams της εισόδου μίας πρότασης και την κάνουν append στο τέλος της πρότασης. Εδώ, θα χρησιμοποιηθεί bi-grams. Ένα bi-gram είναι ένα ζευγάρι λέξεων που εμφανίζονται διαδοχικά σε μία πρόταση.

Για παράδειγμα, σε μια πρόταση "how are you?", τα bi-grams θα είναι: "how are", "are you" and "you?".

Η generate_bigrams function παίρνει μια πρόταση που έχει γίνει tokenized, το υπολογίζει τα bi-grams και μετά να κάνει appends στο τέλος της tokenized list.

```
!pip install -U torchtext==0.10.0
```

```
def generate_bigrams(x):
    n_grams = set(zip(*[x[i:] for i in range(2)]))
    for n_gram in n_grams:
        x.append(' '.join(n_gram))
    return x
generate_bigrams(['This', 'film', 'is', 'terrible'])
```

```
['This', 'film', 'is', 'terrible', 'film is', 'is terrible', 'This film']
```

Τα Fields του TorchText έχει μία preprocessing argument. Εδώ θα οριστεί μια συνάρτηση όπου θα εφαρμοστεί σε μια πρόταση αφότου θα έχει γίνει tokenized (να μετατραπεί από σειρά σε λίστα με tokens), αλλά πριν από αυτό έχει γίνει numericalized(να μετατραπεί από λίστα από tokens σε λίστα από indexes). Εδώ είναι που θα περαστεί η generate_bigrams συνάρτηση.

Καθώς δεν θα χρησιμοποιηθεί η ΕΔΝ αρχιτεκτονική, δεν μπορεί να χρησιμοποιηθούν packed padded sequences άρα δεν χρειάζεται να ρυθμιστεί το include_lengths = True.

```
import torch
from torchtext.legacy import data
from torchtext.legacy.data import Field, TabularDataset, BucketIterator, Ite
rator
SEED = 1234
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
TEXT = data.Field(tokenize = 'spacy', preprocessing = generate_bigrams)
LABEL = data.LabelField(dtype = torch.float)
```

Όπως προηγούμενος, θα φορτωθεί το IMDb dataset και θα δημιουργηθούν τα splits.

```
from torchtext.legacy import datasets
import random
train_data, test_data = datasets.IMDB.splits(TEXT, LABEL)
train_data, valid_data = train_data.split(random_state = random.seed(SEED))
```

Χτίζεται το λεξιλόγιο και φορτώνεται το pre-trained word embeddings.

```
MAX_VOCAB_SIZE = 25_000
TEXT.build_vocab(train_data,
                 max_size = MAX_VOCAB_SIZE,
                 vectors = "glove.6B.100d",
                 unk_init = torch.Tensor.normal_)
LABEL.build_vocab(train_data)
```

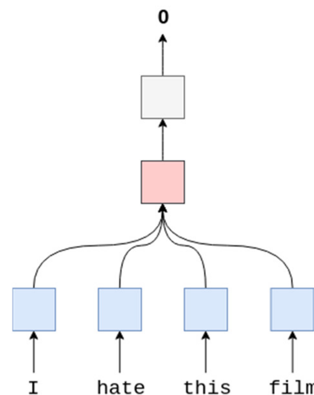
Στην συνέχεια πραγματοποιείται δημιουργία των iterators.

```
BATCH_SIZE = 64
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
train_iterator, valid_iterator, test_iterator = data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = BATCH_SIZE,
    device = device)
```

5.2 Χτίζοντας το μοντέλο

Αυτό το μοντέλο έχει πολύ λιγότερες παραμέτρους από το προηγούμενο μοντέλο επειδή έχει μόνο 2 layer με παραμέτρους, το ενσωματωμένο layer και το γραμμικό layer.

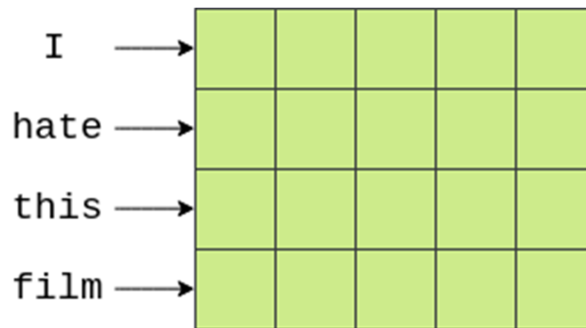
Αρχικά υπολογίζεται η word embedding για κάθε λέξη χρησιμοποιώντας το Embedding layer(blue), στην συνέχεια υπολογίζεται ο μέσος όρος από όλες τις word embeddings(pink) και τροφοδοτείται στο Linear layer(silver).



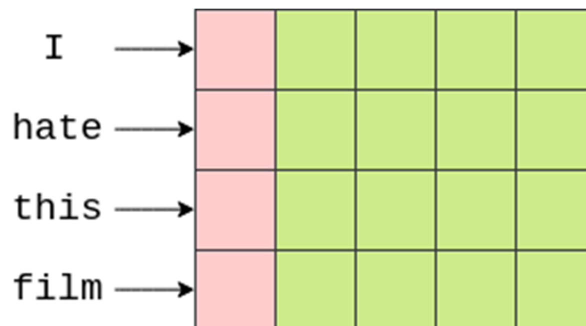
Η υλοποίηση του παραπάνω γίνεται με το να βγαίνει ο μέσος όρος με την συνάρτηση avg_pool2d(average pool 2-dimensions). Αρχικά φαίνεται λάθος να χρησιμοποιείται μία 2-dimensional pooling ενώ ξέρουμε ότι οι προτάσεις είναι 1-dimensional και όχι 2-dimensional.

Ωστόσο, βοηθάει να ορίζεται το word embeddings έως 2-dimensional grid, όπου οι λέξεις βρίσκονται κατά μήκος ενός άξονα και η διαστάσεις των word embeddings στον άλλον άξονα.

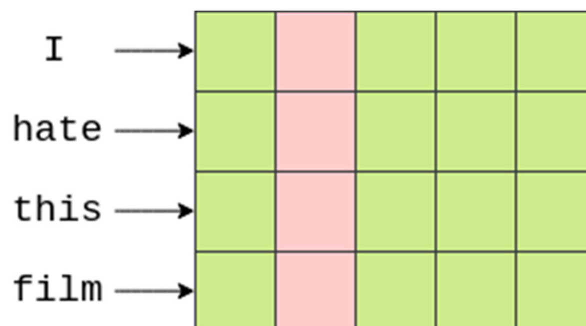
Η παρακάτω εικόνα δείχνει ένα παράδειγμα πρότασης όπου έχει μετατραπεί σε 5-dimensional word embeddings, με τις λέξεις κατά μήκος στον κάθετο άξονα. Κάθε στοιχείο σε αυτόν τον |4x5| tensor αντιπροσωπεύεται από ένα πράσινο μπλοκ.



Η avg_pool2d χρησιμοποιεί ένα φίλτρο που δείχνει το μήκος της πρότασης, το οποίο ορίζεται κατά ένα χρησιμοποιώντας το embedded.shape. Αυτό αντιπροσωπεύεται με ροζ στην παρακάτω εικόνα.



Υπολογίζεται ο μέσος όρος των τιμών όλων των στοιχείων που καλύπτονται από το παραπάνω φίλτρο, στην συνέχεια το φίλτρο πάει προς τα αριστερά, υπολογίζοντας των μέσο όρο για την επόμενη στήλη των embedding values για κάθε λέξη της πρότασης



Κάθε φίλτρο δίνει μόνο μία τιμή, αυτή η τιμή είναι ο μέσος όρος όλων των στοιχείων που καλύφθηκαν. Αφότου το φίλτρο έχει καλύψει όλα τις ενσωματωμένες διαστάσεις, δίνεται ένας tensor |1x5|. Αυτός ο tensor περνάει από το γραμμικό layer για να παράγει την πρόβλεψη.

```
import torch.nn as nn
import torch.nn.functional as F
class FastText(nn.Module):
    def __init__(self, vocab_size, embedding_dim, output_dim, pad_idx):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx
=pad_idx)
        self.fc = nn.Linear(embedding_dim, output_dim)
    def forward(self, text):
        #το text ισούται = [sent len, batch size]
        embedded = self.embedding(text)
        #το embedded ισούται = [sent len, batch size, emb dim]
        embedded = embedded.permute(1, 0, 2)
        #το embedded ισούται = [batch size, sent len, emb dim]
        pooled = F.avg_pool2d(embedded, (embedded.shape[1], 1)).squeeze(1)
        #το pooled ισούται = [batch size, embedding_dim]
        return self.fc(pooled)
```

Όπως προηγούμενος, θα δημιουργείται ένα instance για την FastText κλάση.

```
INPUT_DIM = len(TEXT.vocab)
EMBEDDING_DIM = 100
OUTPUT_DIM = 1
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]
model = FastText(INPUT_DIM, EMBEDDING_DIM, OUTPUT_DIM, PAD_IDX)
```

```
The model has 2,500,301 trainable parameters
```

Παρατηρείται το νούμερο των παραμέτρων στο μοντέλο και διαπιστώνεται ότι έχει περίπου τον ίδιο αριθμό με ένα τυπικό ΕΔΝ από το πρώτο κεφάλαιο και τις μισές παραμέτρους από το προηγούμενο κεφάλαιο.

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'The model has {count_parameters(model):,} trainable parameters')
```

Γίνεται αντιγραφή τους pre-trained vectors στο embedding layer.

```
pretrained_embeddings = TEXT.vocab.vectors

model.embedding.weight.data.copy_(pretrained_embeddings)
```

```
tensor([[-0.1117, -0.4966,  0.1631,  ...,  1.2647, -0.2753, -0.1325],
        [-0.8555, -0.7208,  1.3755,  ...,  0.0825, -1.1314,  0.3997],
        [-0.0382, -0.2449,  0.7281,  ..., -0.1459,  0.8278,  0.2706],
        ...,
        [ 0.2041,  0.0688,  0.3103,  ..., -0.2860,  0.1658, -0.0743],
        [-0.6884, -0.7427,  1.2055,  ...,  1.6994,  1.2431,  1.3594],
        [-0.1734, -0.3195,  0.3694,  ..., -0.2435,  0.4767,  0.1151]])
```

Στη συνέχεια, μηδενίζονται τα αρχικά weights των άγνωστων και padding tokens.

```
UNK_IDX = TEXT.vocab.stoi[TEXT.unk_token]

model.embedding.weight.data[UNK_IDX] = torch.zeros(EMBEDDING_DIM)

model.embedding.weight.data[PAD_IDX] = torch.zeros(EMBEDDING_DIM)
```

5.3 Εκπαιδεύοντας το μοντέλο

Όπως και στο προηγούμενο κεφάλαιο, γίνεται αρχικοποίηση του optimizer.

```
import torch.optim as optim

optimizer = optim.Adam(model.parameters())
```

Μετά ορίζονται τα κριτήρια και τοποθετείται το μοντέλο και τα κριτήρια στην GPU.

```
criterion = nn.BCEWithLogitsLoss()

model = model.to(device)

criterion = criterion.to(device)
```

Γίνεται υλοποίηση της συνάρτηση που υπολογίζει την ακρίβεια του μοντέλου.

```
def binary_accuracy(preds, y):
    """
    Returns accuracy per batch, i.e. if you get 8/10 right, this returns 0.8
, NOT 8
    """
    # στρογγυλοποίηση προβλέψεων στον πλησιέστερο ακέραιο αριθμό
    rounded_preds = torch.round(torch.sigmoid(preds))
    correct = (rounded_preds == y).float() #μετατροπή σε float για διαίρεση
    acc = correct.sum() / len(correct)
    return acc
```

Ορίζεται μία συνάρτηση για να εκπαιδευτεί το μοντέλο.

Σημείωση: αφού σε αυτό το μοντέλο χρησιμοποιείται η dropout, να χρησιμοποιηθεί το model.train() για να εξασφαλιστεί ότι η dropout είναι ενεργοποιημένη ενώ γίνεται εκπαίδευση

```
def train(model, iterator, optimizer, criterion):

    epoch_loss = 0

    epoch_acc = 0

    model.train()

    for batch in iterator:

        optimizer.zero_grad()

        predictions = model(batch.text).squeeze(1)

        loss = criterion(predictions, batch.label)

        acc = binary_accuracy(predictions, batch.label)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()

        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

Ορίζεται μία συνάρτηση για να γίνει τεστ στο μοντέλο.

```
import time
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

Τέλος, γίνεται εκπαίδευση στο μοντέλο:

```
N_EPOCHS = 5
best_valid_loss = float('inf')
for epoch in range(N_EPOCHS):
    start_time = time.time()
```

```
    train_loss, train_acc = train(model, train_iterator, optimizer, criterio
n)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)
    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut3-model.pt')
    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%
')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%
')
```

```
Epoch: 01 | Epoch Time: 0m 10s
        Train Loss: 0.688 | Train Acc: 59.34%
         Val. Loss: 0.637 |  Val. Acc: 71.07%
Epoch: 02 | Epoch Time: 0m 9s
        Train Loss: 0.649 | Train Acc: 74.30%
         Val. Loss: 0.508 |  Val. Acc: 75.98%
Epoch: 03 | Epoch Time: 0m 10s
        Train Loss: 0.575 | Train Acc: 80.16%
         Val. Loss: 0.427 |  Val. Acc: 80.51%
Epoch: 04 | Epoch Time: 0m 10s
        Train Loss: 0.495 | Train Acc: 84.49%
         Val. Loss: 0.383 |  Val. Acc: 83.96%
Epoch: 05 | Epoch Time: 0m 9s
        Train Loss: 0.430 | Train Acc: 87.39%
         Val. Loss: 0.376 |  Val. Acc: 85.77%
```

Τα αποτελέσματα του test accuracy είναι ίδια με το προηγούμενο κεφάλαιο. **Ωστόσο είναι πολύ σημαντικό ότι μειώνεται ο χρόνος εκπαίδευσης.**

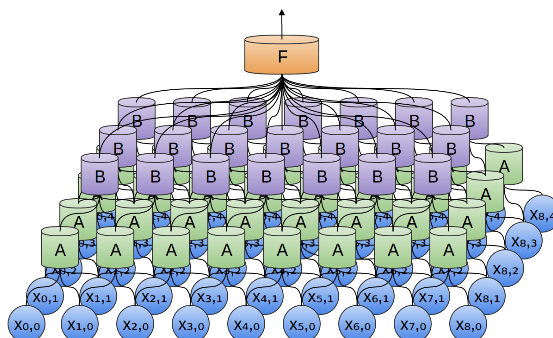
```
model.load_state_dict(torch.load('tut3-model.pt'))
test_loss, test_acc = evaluate(model, test_iterator, criterion)
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Test Loss: 0.382 | Test Acc: 85.52%
```

6 Ανάλυση συναισθήματος με Συνελικτικά Νευρωνικά Δίκτυα (CNN)

Τα τελευταία χρόνια, τα Βαθιά Νευρωνικά Δίκτυα έχουν οδηγήσει σε καινοτόμα αποτελέσματα σε διάφορα προβλήματα αναγνώρισης προτύπων, όπως στον τομέα της μηχανικής όρασης (computer vision) και αναγνώριση φωνής (voice recognition). Ένα από τα βασικά συστατικά που οδηγούν σε αυτά τα αποτελέσματα ήταν ένα ειδικό είδος νευρωτικού δικτύου που ονομάζεται Convolutional Neural Network.

Στην βασική του μορφή, τα Συνελικτικά Νευρωνικά Δίκτυα μπορούν να θεωρηθούν ένα είδος νευρωνικού δικτύου που χρησιμοποιεί πολλά ταυτόσημα αντίγραφα του ίδιου νευρώνα. Αυτό επιτρέπει στο δίκτυο να έχει πολλούς νευρώνες και να εκφράζει υπολογιστικά μεγάλα μοντέλα ενώ παράλληλα διατηρεί τον αριθμό των πραγματικών παραμέτρων και τις τιμές που περιγράφουν πως συμπεριφέρονται οι νευρώνες που πρέπει να εκπαιδευτούν σχετικά σε μικρό μέγεθος.



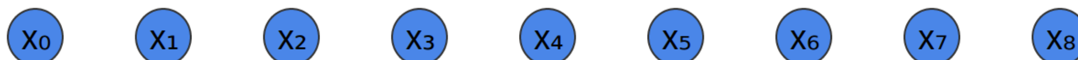
Εικόνα 1 Ένα 2D Convolutional Neural Network

Αυτό το τέχνασμα της ύπαρξης πολλαπλών αντιγράφων του ίδιου νευρώνα είναι σχεδόν ανάλογο με την αφαίρεση των συναρτήσεων στα μαθηματικά και στην επιστήμη των υπολογιστών. Όταν προγραμματίζεται ή γράφεται μία συνάρτηση μια φόρα και την χρησιμοποιείται σε διαφορετικά σημεία, δηλαδή δεν γράφεται ο ίδιος κώδικας σε διαφορετικά σημεία, αυτό καθιστά πιο γρήγορο τον προγραμματισμό και οδηγεί σε λιγότερα σφάλματα. Ομοίως, ένα Συνελικτικά Νευρωνικά Δίκτυο μπορεί να εκπαιδεύσει έναν νευρώνα μια φορά και να τον χρησιμοποιήσει σε πολλά σημεία, κάνοντας ευκολότερη την εκπαίδευση του μοντέλου και μειώνοντας τα σφάλματα (Olah, n.d.).

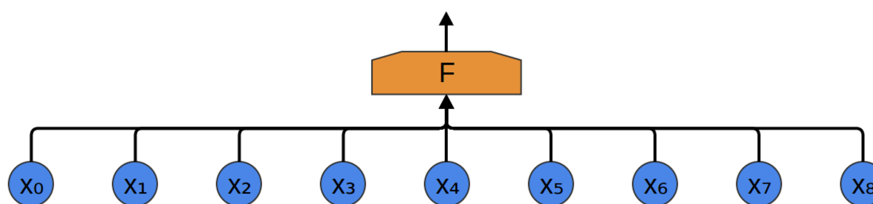
6.1 Δομή των Συνελικτικών Νευρωνικών Δικτύων

Έστω ότι ένα νευρωνικό εξετάζει δείγματα ήχου και να προβλέψει αν είναι άνθρωπος που μιλάει ή όχι. Ίσως χρειαστεί να πραγματοποιηθούν περισσότερες αναλύσεις για την αναγνώριση ανθρώπινης φωνής.

Παίρνουμε τα δείγματα ήχου σε διαφορετικά χρονικά σημεία και τα κατανέμοντα ομοιόμορφα.



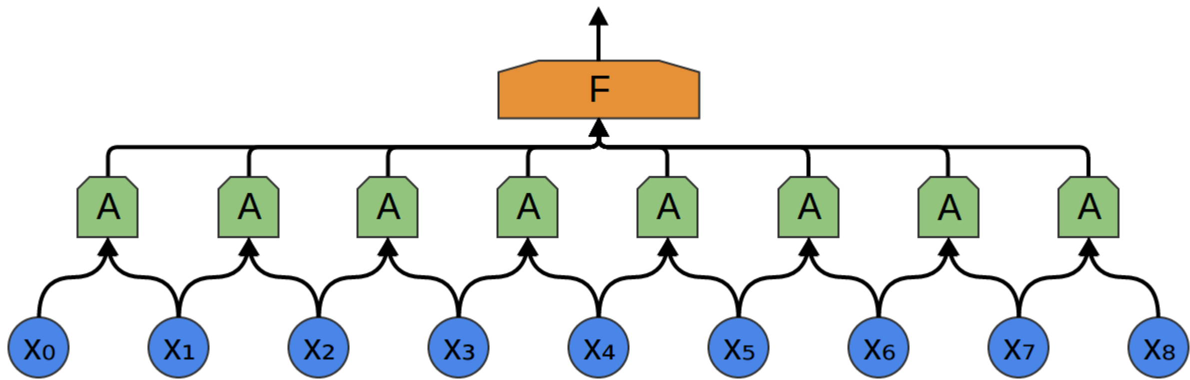
Ο πιο απλός τρόπος να δοκιμαστεί και να ταξινομηθεί σε ένα νευρωνικό δίκτυο είναι απλώς να τα συνδέσουμε όλα σε ένα πλήρως συνδεδεμένο layer. Υπάρχει ένα bunch από διαφορετικούς νευρώνες και κάθε είσοδος συνδέεται με κάθε νευρώνα.



Μια πιο καλή προσέγγιση παρατηρεί ένα είδος συμμετρίας στις ιδιότητες που είναι χρήσιμες για να αναζητηθούν στα δεδομένα. Φροντίζεται ιδιαίτερα για τις τοπικές ιδιότητες των δεδομένων. Ποια συχνότητα ήχων υπάρχουν γύρω σε μια δεδομένη στιγμή; Αυξάνονται ή μειώνονται και ούτω καθεξής.

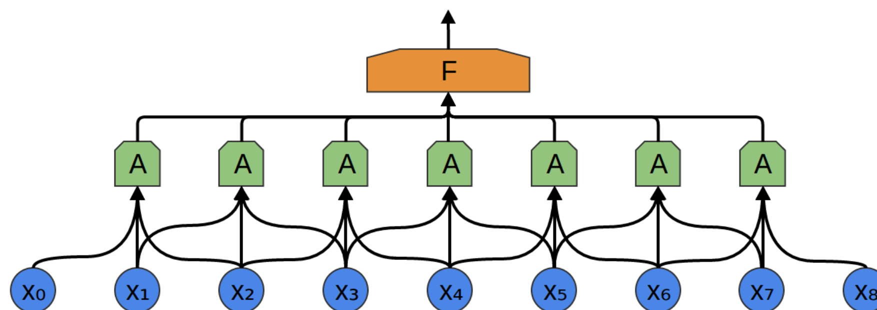
Ιδιαίτερο ενδιαφέρον έχουν ίδιες ιδιότητες σε όλες τις χρονικές στιγμές. Είναι χρήσιμο να είναι γνωστές οι συχνότητες στην αρχή, στην μέση και στο τέλος. Και πάλι, αυτές είναι τοπικές ιδιότητες, δεδομένου ότι χρειάζεται να κοιταχθεί ένα μικρό μέρος του ηχητικού δείγματος για προσδιοριστούν.

Έτσι, μπορεί να δημιουργηθεί μία ομάδα νευρώνων, A που εξετάζουν τα μικρά χρονικά τμήματα των δεδομένων. Το A εξετάζει όλα τα τμήματα, υπολογίζοντας ορισμένα χαρακτηριστικά (features). Στη συνέχεια, ή έξοδος του convolutional layer τροφοδοτείται σε ένα πλήρως συνδεδεμένο layer F (Olah, n.d.).



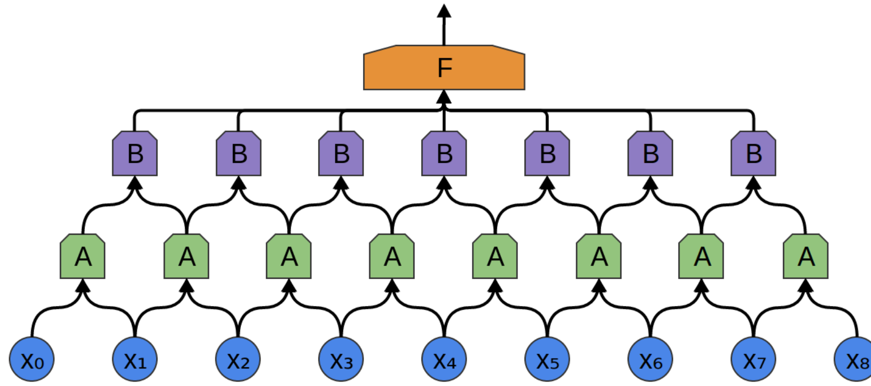
Στο παραπάνω παράδειγμα, το A εξέτασε μόνο τα τμήματα που αποτελούνται από δύο σημεία, κάτι που δεν είναι ρεαλιστικό. Συνήθως, το convolution layer εύρος είναι πολύ μεγαλύτερο.

Στο παρακάτω παράδειγμα, το A εξετάζει 3 σημεία, το οποίο δεν είναι και αυτό ρεαλιστικό, δυστυχώς είναι δύσκολο να απεικονιστεί η σύνδεση A με πολλά σημεία.



Μια χρήσιμη ιδιότητα των convolutional layers είναι ότι είναι σύνθετα. Μπορεί να τροφοδοτηθεί η έξοδο ενός convolutional layer σε ένα άλλο. Με κάθε στρώμα, το δίκτυο μπορεί να ανιχνεύσει πιο αφηρημένα χαρακτηριστικά υψηλότερου επιπέδου.

Στο επόμενο παράδειγμα, υπάρχει μια νέα ομάδα νευρώνων, B. Το B χρησιμοποιείτε για να δημιουργήσει ένα στρώμα άλλο convolutional layer που στοιβάζεται πάνω από το προηγούμενο, δηλαδή στο A.

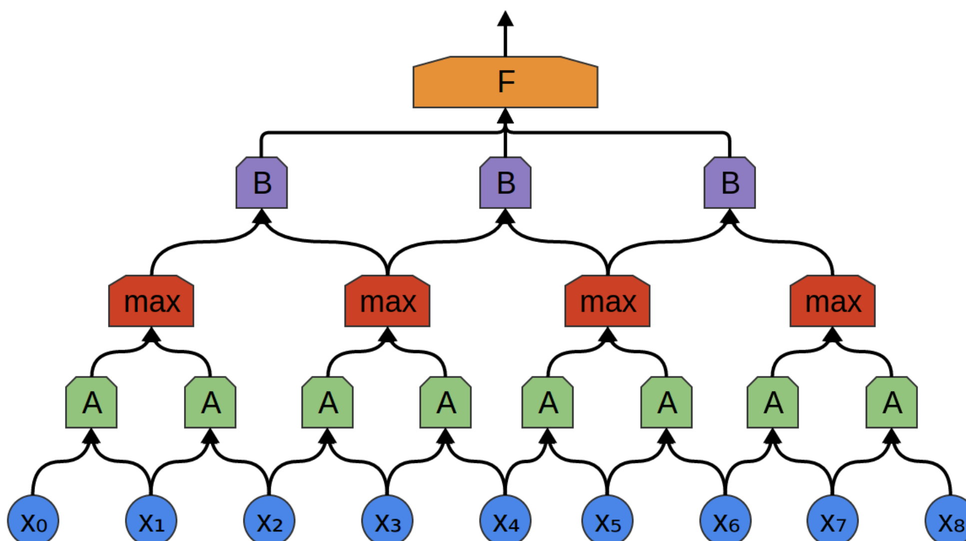


Τα Convolutional layers συχνά αλληλοσυνδέονται με pooling layers. Συγκεκριμένα, υπάρχει ένα είδος layer που ονομάζεται max-pooling layer και είναι εξαιρετικά δημοφιλές.

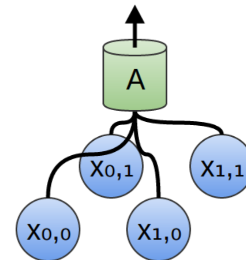
Συχνά, όταν παρατηρείται η μεγαλύτερη εικόνα, δεν έχει ενδιαφέρον το ακριβές χρονικό σημείο στο οποίο υπάρχει ένα χαρακτηριστικό. Αν υπάρχει μία μετατόπιση στη συχνότητα όπου εμφανίζεται ελαφρώς νωρίτερα ή αργότερα δεν έχει σημασία.

Ένα max-pooling layer παίρνει το μέγιστο αριθμό χαρακτηριστικών σε μικρά blocks του προηγούμενου layer. Η έξοδος λέει ένα χαρακτηριστικό ήταν παρών σε μια περιοχή του προηγούμενου layer, αλλά όχι ακριβώς που.

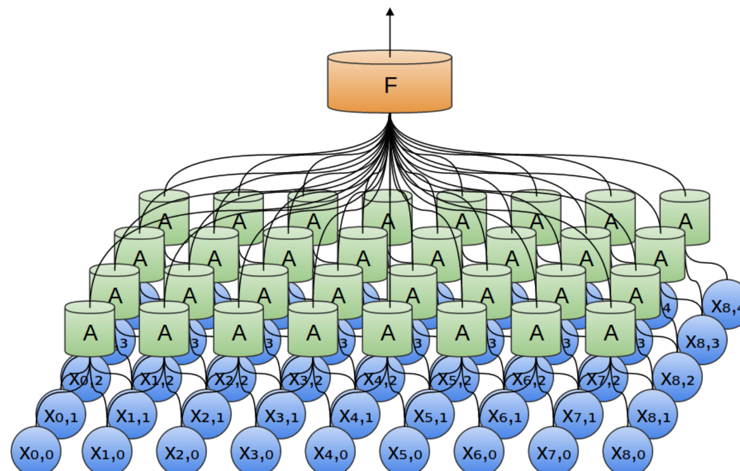
Τα max-pooling layers κάνουν “zoom out”. Επιτρέπουν στα convolutional layers να δουλέψουν σε μεγαλύτερα τμήματα των δεδομένων, επειδή ένα μικρό patch μετά το pooling layer αντιστοιχεί σε ένα πολύ μεγαλύτερο patch πριν από αυτό.



Στα προηγούμενα παραδείγματα, χρησιμοποιήθηκαν μονοδιάστατα convolutional layers. Ωστόσο, τα convolutional layers μπορούν επίσης να λειτουργήσουν σε δεδομένα υψηλότερων διαστάσεων. Στην πραγματικότητα, οι πιο γνωστές επιτυχημένες εφαρμογές των Convolutional Neural Networks είναι σε 2D (2 διαστάσεις) για την αναγνώριση εικόνων. Σε ένα δισδιάστατο Convolutional layer, αντί να ελέγχονται τα τμήματα το A, θα εξετάζονται τα patches.



Για κάθε patch A, υπολογίζονται τα χαρακτηριστικά. Για παράδειγμα, μπορεί να μάθει να ανιχνεύει την ύπαρξη ενός άκρου, να ανιχνεύει ένα texture ή ίσως μια αντίθεση μεταξύ δύο χρωμάτων.



Μπορούμε επίσης να γίνει max pooling σε 2 διαστάσεις. Εδώ, λαμβάνεται το μέγιστο των χαρακτηριστικών πάνω σε ένα μικρό patch.

Μεταξύ των υφιστάμενων μελετών που χρησιμοποιούν βαθιά μάθηση για την ταξινόμηση κειμένων, το CNN εκμεταλλεύεται τα αποκαλούμενα συνελικτικά φίλτρα που μαθαίνουν αυτόματα χαρακτηριστικά που είναι κατάλληλα για τη συγκεκριμένη εργασία που τους δίνεται.

Για παράδειγμα, εάν χρησιμοποιείται το CNN για sentiment classification, τα συνελικτικά φίλτρα μπορεί να συλλάβουν inherent syntactic και semantic features των συναισθηματικών εκφράσεων (Anthony Rios, Ramakanth Kavuluru 2 , 2015). Έχει αποδειχθεί ότι ένα ενιαίο στρώμα συνέλιξης και ένας συνδυασμός συνελικτικών φίλτρων, θα μπορούσε να επιτύχει συγκρίσιμη απόδοση χωρίς κάποιου είδους ειδικών παραμέτρων (Kim, 2014). Επιπλέον, στα

CNN δεν απαιτούνται ειδικές γνώσεις σχετικά με τη γλωσσική δομή μιας γλώσσας (Xiang Zhang, Junbo Zhao, Yann LeCun, 2015).

Χάρη σε αυτά πλεονεκτήματα, το CNN εφαρμόστηκε επιτυχώς σε διάφορες αναλύσεις κειμένων: ανάλυση συναισθημάτων (Xi Ouyang, Pan Zhou, Cheng Hua Li, Pan Zhou, 2015), semantic parsing (Wen-tau Yih), search by query (Yelong Shen, 2014), μοντελοποίηση προτάσεων (Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom, 2014).

Τα επαναλαμβανόμενα Νευρωνικά δίκτυα (RNN) μπορεί να θεωρηθούν καλύτερα για την ταξινόμηση κειμένου από ό, τι ο CNN, καθώς διατηρεί τη σειρά της ακολουθίας λέξεων.

Ωστόσο, τα Συνελικτικά Νευρωνικά Δίκτυα είναι επίσης ικανά να συλλάβουν διαδοχικά μοτίβα, όσον αφορά τα τοπικά σχέδια από το συνελικτικά φίλτρα.

Για παράδειγμα, τα συνελικτικά φίλτρα μαζί με την τεχνική προσοχής (attention technique) έχουν εφαρμοστεί πετυχημένα στη μηχανική μετάφραση. Επιπλέον, σε σύγκριση με το RNN, το CNN έχει ως επί το πλείστων μικρότερο μέγεθος αριθμός παραμέτρων, έτσι ώστε το CNN να είναι εκπαιδευτικό με ένα μικρό αριθμό δεδομένων. Το CNN είναι επίσης γνωστό για να διερευνήσει τον πλούτο των προπλασμένων ενσωματωμένων λέξεων (Olah, n.d.).

6.2 Υλοποίηση Συνελικτικών Νευρωνικών Δικτύων

Στο προηγούμενο κεφάλαιο, επιτεύχθηκε ακρίβεια γύρω στο 85% χρησιμοποιώντας ΕΝΔ και την υλοποίηση του «Bag of Tricks for Efficient Text Classification» μοντέλου (Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov, 2016). Σε αυτό το κεφάλαιο, θα χρησιμοποιηθεί ένα Συνελικτικό Νευρωνικό Δίκτυο (CNN) για να διεξαχθεί ανάλυση συναισθήματος (Kim, Convolutional Neural Networks for Sentence Classification, 2014).

Συνήθως τα CNN χρησιμοποιούνται για την ανάλυση εικόνων και αποτελούνται από ένα ή περισσότερα convolutional layers, συνοδευόμενο από ένα ή περισσότερα γραμμικά στρώματα. Τα convolutional layers χρησιμοποιούν φίλτρα(συχνά ονομάζονται και kernels ή receptive fields) τα οποία σαρώνουν όλη την φωτογραφία και παράγουν μια επεξεργασμένη έκδοση αυτής της φωτογραφίας. Αυτή η επεξεργασμένη έκδοση της εικόνας μπορεί να τροφοδοτηθεί σε ένα άλλο convolutional layer ή να γραμμικό layer. Κάθε φίλτρο έχει ένα σχήμα, δηλαδή ένα 3x3 φίλτρο καλύπτει 3 pixel wide και 3 pixel high area of the image, και κάθε στοιχείο του φίλτρου έχει ένα βάρος που σχετίζετε με τα pixel που καλύπτονται από το φίλτρο, άρα ένα 3x3 φίλτρο θα έχει 9 βάρη. Στην παραδοσιακή επεξεργασία φωτογραφίας αυτά τα βάρη

προσδιορίζονται χειροκίνητα, ωστόσο το βασικό πλεονέκτημα των convolutional layers στα Νευρωνικά δίκτυα είναι ότι αυτά τα βάρη μαθαίνονται μέσω του backpropagation.

Η αρχική ιδέα πίσω από τα βάρη είναι ότι το convolutional layer δρουν σαν τους εξαγωγέας χαρακτηριστικών, η εξαγωγή τμημάτων μιας εικόνας είναι ο πιο σημαντικός στόχος της CNN αρχιτεκτονικής. **Για παράδειγμα:** αν χρησιμοποιείται ένας CNN να εντοπίσει πρόσωπα σε μία εικόνα, ο CNN ίσως κοιτάξει για χαρακτηριστικά προσώπου στην εικόνα όπως η ύπαρξη μίας μύτης, στόματος ή ένα ζευγάρι μάτια στην εικόνα.

6.2.1 Συνελικτικά Νευρωνικά Δίκτυα σε κείμενο

Με τον ίδιο τρόπου που ένα φίλτρο 3x3 μπορεί να ψάξει σε ένα patch μίας εικόνας, έτσι και ένα φίλτρο 1x2 μπορεί να ψάξει διαδοχικές λέξεις σε ένα κομμάτι κειμένου, δηλαδή ένα bi-gram. Στο προηγούμενο κεφάλαιο υλοποιήθηκε το FastText μοντέλο όπου χρησιμοποιεί τα bi-grams, προσθέτοντας τα στο τέλος ενός κειμένου, σε αυτό το CNN μοντέλο θα χρησιμοποιηθούν πολλαπλά φίλτρα διαφορετικών μεγεθών θα ψάξουν στα bi-grams(ένα 1x2 φίλτρο), στα tri-grams(ένα 1x3 φίλτρο) και/η στα n-grams(ένα 1 x alt text) φίλτρο) μέσα στο κείμενο.

Η εμφάνιση συγκεκριμένων bi-grams, tri-grams και n-grams μέσα στο review θα είναι μια καλή ένδειξη τι θα περιέχει η τελική ανάλυση.

6.3 Προετοιμασία δεδομένων

Η προετοιμασία θα γίνει όπως στα προηγούμενα κεφάλαια με μερικές λειτουργικές διαφορές. Αντίθετα με το ακριβώς προηγούμενο κεφάλαιο που υλοποιήθηκε το FastText μοντέλο, δεν χρειάζεται πλέον να δημιουργηθούν τα bi-grams και να προσαρμοστούν στο τέλος της πρότασης.

```
!pip install -U torchtext==0.10.0
```

```python
import torch
from torchtext.legacy import data
from torchtext.legacy.data import Field, TabularDataset, BucketIterator, Ite
rator
from torchtext.legacy import datasets
import random
SEED = 1234
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
TEXT = data.Field(tokenize = 'spacy')
LABEL = data.LabelField(dtype = torch.float)
train_data, test_data = datasets.IMDB.splits(TEXT, LABEL)
train_data, valid_data = train_data.split(random_state = random.seed(SEED))
```

```
downloading aclImdb_v1.tar.gz
aclImdb_v1.tar.gz: 100%|██████████| 84.1M/84.1M [00:08<00:00, 9.91MB/s]
```

Χτίζεται το λεξικό και φορτώνονται τα pre-trained word embeddings.

```python
MAX_VOCAB_SIZE = 25_000
TEXT.build_vocab(train_data,
                 max_size = MAX_VOCAB_SIZE,
                 vectors = "glove.6B.100d",
                 unk_init = torch.Tensor.normal_)
LABEL.build_vocab(train_data)
```

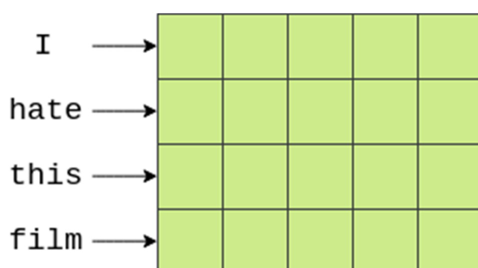
```
.vector_cache/glove.6B.zip: 862MB [06:31, 2.20MB/s]
100%|█████████▉| 398568/400000 [00:22<00:00, 15785.17it/s]
```

Όπως προηγουμένως, δημιουργούνται οι iterators.

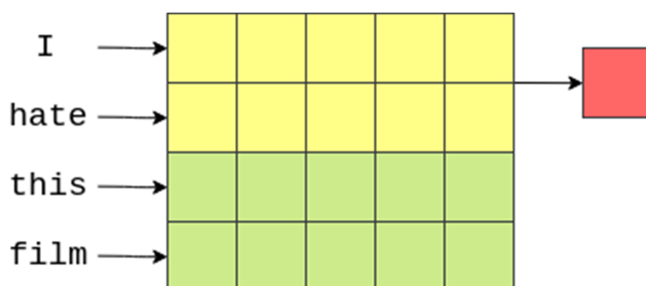
```python
BATCH_SIZE = 64
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
train_iterator, valid_iterator, test_iterator = data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = BATCH_SIZE,
    device = device)
```

6.4 Χτίζοντας το μοντέλο

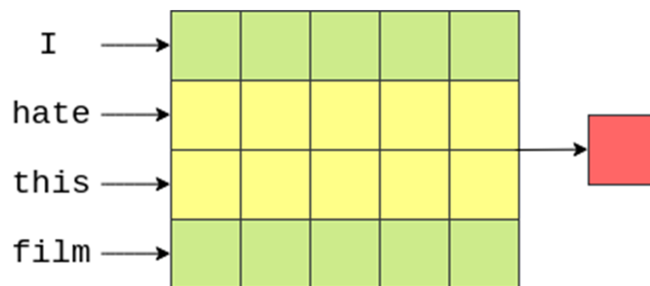
Το πρώτο σημαντικό εμπόδιο είναι η απεικόνιση με τον οποίο χρησιμοποιείτε ο CNN σε κείμενα. Οι εικόνες συνήθως είναι σε 2 διαστάσεις (προς το παρών θα αγνοηθεί η διάσταση "χρώμα") ενώ το κείμενο είναι μονοδιάστατο. Όπως και προηγούμενος θα μετατραπούν οι λέξεις σε word embeddings. Με αυτό τον τρόπο δίνεται η δυνατότητα να απεικονιστούν οι λέξεις σε 2 διαστάσεις, κάθε λέξη κατά μήκος ενός άξονα και τα στοιχεία των vectors σε άλλη διάσταση. Εξετάζεται η αναπαράσταση 2 διαστάσεων μίας embedded sentences παρακάτω:



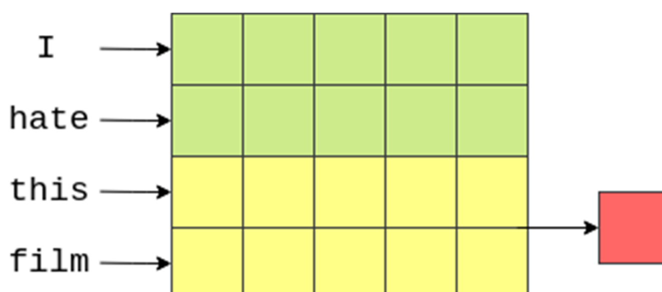
Στην συνέχεια χρησιμοποιείται ένα φίλτρο που είναι |n x emb_dim|. Αυτό το φίλτρο θα καλύψει την ακολουθία λέξεων εξ ολοκλήρου, ενώ το μήκος του θα είναι emb_dim διαστάσεις. Εξετάζοντας την παρακάτω εικόνα, ο word vectors μας απεικονίζεται με πράσινο χρώμα. Όπως παρατηρείται υπάρχουν 4 λέξεις με 5 ενσωματωμένες διαστάσεις, δημιουργώντας έναν |4x5| "image" tensor. Ένα φίλτρο που καλύπτει 2 λέξεις την φορά (bi-grams) θα είναι |2x5| φίλτρο, αυτό το φίλτρο απεικονίζεται με χρώμα κίτρινο. Κάθε στοιχείο του φίλτρου θα σχετίζεται με ένα βάρος. Η έξοδος αυτού του φίλτρου (απεικονίζεται με χρώμα κόκκινο) θα είναι ένας πραγματικός αριθμός που θα το άθροισμα τα βάρη από όλα τα στοιχεία που κάλυψε το φίλτρο.



Στην συνέχεια, το φίλτρο μετακινεί την εικόνα "προς τα κάτω" για να καλύψει το επόμενο bi-gram και αφότου γίνει αυτό άλλη μια έξοδος με το άθροισμα των weights έχει υπολογιστεί.



Τέλος το φίλτρο πάει προς τα κάτω πάλι και η τελική έξοδος για αυτό το φίλτρο έχει υπολογιστεί.



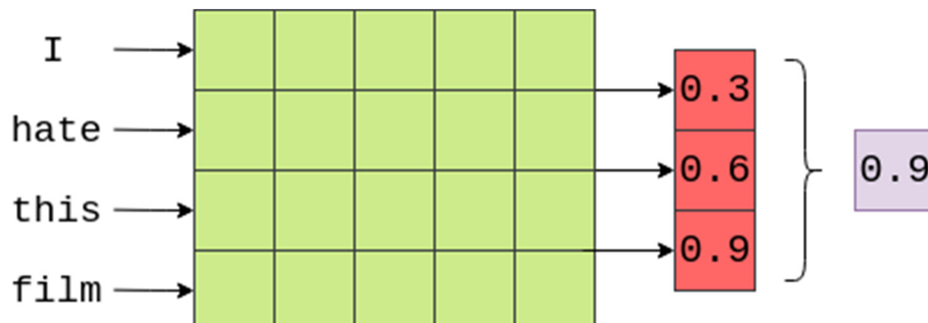
Στην δικιά μας την περίπτωση (και στην γενική περίπτωση όπου το μήκος των φίλτρων είναι ίσο με το μήκος της "εικόνας"), η έξοδος θα είναι ένας vector με αριθμό στοιχείων ίσο με το ύψος/height της εικόνας(ή length για μία λέξη) μείον το ύψος/height του φίλτρου συν ένα, $4 - 2 + = 3$ σε αυτή την περίπτωση.

Αυτό το παράδειγμα έδειξε πως υπολογίζεται η έξοδος ενός φίλτρου. Στο μοντέλο αυτό (και σχεδόν σε όλα τα CNN μοντέλα) υπάρχουν πολλά τέτοια φίλτρα. Η ιδέα για κάθε φίλτρο είναι ότι θα μάθει ένα διαφορετικό χαρακτηριστικό για να το εξάγει. Στο παραπάνω παράδειγμα στόχος είναι ότι κάθε από τα |2 x emb_dim| φίλτρα θα ψάξουν για την εμφανίσει διαφορετικών bi-grams.

Στο μοντέλο μπορούν να προστεθούν διαφορετικά μεγέθη φίλτρων, heights των 3,4 και 5 από 100 για κάθε ένα από αυτά. Στόχος είναι η εμφάνιση διαφορετικών tri-grams, 4-grams και 5-grams τα οποία είναι σχετικά για την ανάλυση συναισθήματος σε κριτικές ταινιών.

Στο επόμενο βήμα του μοντέλου είναι να χρησιμοποιήσουμε το pooling(πιο συγκεκριμένα max pooling) στην έξοδο του convolutional layer. Αυτό είναι παρόμοιο με την τεχνική που χρησιμοποιήθηκε στο FastText μοντέλο που εκτελεί το μέσο όρο από κάθε ένα word vector, υλοποιώντας το αυτό με την F.avg_pool2d συνάρτηση, ωστόσο αντί να παρθεί ο μέσος όρος μιας διάστασης, θα παρθεί η μέγιστη τιμή μιας διάστασης. Παρακάτω ένα παράδειγμα που

παίρνει την μέγιστη τιμή (0.9) από την έξοδος του convolutional layer στην πρόταση του παραδείγματος(σημειώνεται ότι δεν εμφανίζεται στο παράδειγμα η ενεργοποίηση της function που εφαρμόζεται στην έξοδο των convolution)



Η ιδέα εδώ είναι ότι η μέγιστη τιμή είναι και το "πιο σημαντικό" χαρακτηριστικό για να καθοριστεί το συναίσθημα στην κριτική, το οποίο αντιστοιχεί και στο "πιο σημαντικό" n-gram μέσα στην κριτική.

<u>Πως μπορεί να βρεθεί ποιο είναι το "πιο σημαντικό" n-gram?</u>

Μέσω του backpropagation, τα βάρη του φίλτρου αλλάζουν, έτσι όταν συγκεκριμένα n-grams είναι εξαιρετικά ιδιαίτερα για συναίσθημα.

Αφού το μοντέλο έχει 100 φίλτρα τριών διαφορετικών μεγεθών, σημαίνει ότι υπάρχουν 300 διαφορετικά n-grams που το μοντέλο θεωρεί ότι είναι σημαντικά. Ενώνονται αυτά τα φίλτρα μαζί σε έναν vector και τα περνιούνται μέσω ενός γραμμικού layer για να κάνει πρόβλεψη συναισθήματος. Μπορεί κάποιος να σκεφτεί τα βάρη αυτού του γραμμικού layer ως «απόδειξη» για κάθε από τα 300 n-grams με αποτέλεσμα να βοηθάει να παρθεί η τελική απόφαση.

6.5 Λεπτομέρειες υλοποίησης

Υλοποιήθηκε αυτό το convolutional layer με το nn.Conv2d. Η argument in_channels είναι ο αριθμός των "καναλιών" στην εικόνα που εισέρχεται στο convolutional layer. Στις πραγματικές εικόνες είναι συνήθως 3 τα κανάλια(ένα για κόκκινο, ένα για μπλε, ένα για πράσινο), ωστόσο όταν χρησιμοποιούμε κείμενο έχουμε ένα μόνο κανάλι. Το out_channels είναι ο αριθμός των φίλτρων και το kernel_size το μέγεθος των φίλτρων. Κάθε kernel_size θα είναι ln x emb_diml όπου το μέγεθος των n-grams.

Στην PyTorch, τα RNNs θέλουν πρώτα το input και μετά την batch dimension, ενώ τα CNNs θέλουν την batch dimension πρώτη. Έτσι, το πρώτο πράγμα είναι να γίνει η είσοδος permute (συνάρτηση στην pytorch) για να έχει το σωστό σχήμα. Στην συνέχεια περνάει η πρόταση στο embedding layer για να παρθούν τα embeddings. Η δεύτερη διάσταση της εισόδου στο nn.Conv2d layer πρέπει να είναι στην channel dimension. Το κείμενο τεχνικά δεν έχει channel dimension, ωστόσο χρησιμοποιείται το unsqueeze στον tensor για να δημιουργηθεί μια τέτοια διάσταση. Αυτό ταιριάζει με το in_channel =1 στην εκκίνηση των convolutional layers.

Στην συνέχεια περνάμε τους tensors στα convolutional και pooling layers, για την ενεργοποίηση χρησιμοποιούμε την function ReLU μετά τα convolutional layers.

Άλλο ένα ωραίο χαρακτηριστικό των pooling layers είναι ότι μπορεί να διαχειριστεί προτάσεις σε διαφορετικά μήκη. Το μέγεθος της εξόδου του convolutional layer εξαρτάται από το μέγεθος της εισόδου σε αυτό, διαφορετικά batches περιέχουν προτάσεις διαφορετικά μήκη. Χωρίς το max pooling layer η είσοδος στο linear layer θα εξαρτηθεί από το μέγεθος της πρότασης εισόδου sentence το οποίο δεν είναι ιδανικό. Μία επιλογή για να το διορθωθεί αυτό είναι να trim/pad όλες τις προτάσεις στο ίδιο μήκος, ωστόσο με το max pooling layer πάντα η είσοδος στα γραμμικά layers που θα είναι ο συνολικός αριθμός των φίλτρων.

Σημείωση: υπάρχει μια εξαίρεση σε αυτό, η πρόταση/προτάσεις είναι μικρότερες από το μεγαλύτερο φίλτρο που χρησιμοποιήθηκε, θα πρέπει να pad τις προτάσεις στο ίδιο μήκος με το μεγαλύτερο φίλτρο. Στα δεδομένα του iMDb δεν υπάρχουν review με 5 λέξεις μόνο, άρα δεν υπάρχει πρόβλημα στην συγκεκριμένη περίπτωση.

Τέλος, εκτελείται το dropout στο concatenated φίλτρο εξόδων και μετά περνιούνται στο γραμμικό layer για να πραγματοποιηθεί η πρόβλεψη.

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

class CNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, n_filters, filter_sizes, o
utput_dim,
                 dropout, pad_idx):
        super().__init__()
```

```
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx
 = pad_idx)
        self.conv_0 = nn.Conv2d(in_channels = 1,
                                out_channels = n_filters,
                                kernel_size = (filter_sizes[0], embedding_di
m))
        self.conv_1 = nn.Conv2d(in_channels = 1,
                                out_channels = n_filters,
                                kernel_size = (filter_sizes[1], embedding_di
m))
        self.conv_2 = nn.Conv2d(in_channels = 1,
                                out_channels = n_filters,
                                kernel_size = (filter_sizes[2], embedding_di
m))
        self.fc = nn.Linear(len(filter_sizes) * n_filters, output_dim)
        self.dropout = nn.Dropout(dropout)
    def forward(self, text):
        #το text ισούται = [sent len, batch size]
        text = text.permute(1, 0)
        #το text ισούται = [batch size, sent len]
        embedded = self.embedding(text)
        #το embedded ισούται = [batch size, sent len, emb dim]
        embedded = embedded.unsqueeze(1)
        #το embedded ισούται = [batch size, 1, sent len, emb dim]
        conved_0 = F.relu(self.conv_0(embedded).squeeze(3))
        conved_1 = F.relu(self.conv_1(embedded).squeeze(3))
        conved_2 = F.relu(self.conv_2(embedded).squeeze(3))
        #το                                                          conved_n
ισούται = [batch size, n_filters, sent len - filter_sizes[n] + 1]
        pooled_0 = F.max_pool1d(conved_0, conved_0.shape[2]).squeeze(2)
        pooled_1 = F.max_pool1d(conved_1, conved_1.shape[2]).squeeze(2)
        pooled_2 = F.max_pool1d(conved_2, conved_2.shape[2]).squeeze(2)
        #το pooled_n ισούται = [batch size, n_filters]
        cat = self.dropout(torch.cat((pooled_0, pooled_1, pooled_2), dim = 1
))
        #το cat ισούται = [batch size, n_filters * len(filter_sizes)]
        return self.fc(cat)
```

Σε αυτή την φάση το CNN μοντέλο χρησιμοποιεί μόνο 3 φίλτρα διαφορετικών μεγεθών, αλλά μπορεί να βελτιωθεί ο κώδικα του μοντέλου και θα γίνει πιο γενικό, με αποτέλεσμα να πάρει οποιαδήποτε αριθμό φίλτρων.

Αυτό θα γίνει βάζοντας όλα τα convolutional layer στην nn.ModuleList, αυτή η συνάρτηση χρησιμοποιείται για να κρατήσει μία λίστα της nn.Modules στην PyTorch. Αν χρησιμοποιηθεί η τυπική συνάρτηση list της Python, τα modules μέσα στην λίστα θα είναι ορατά στα modules που είναι έξω από την λίστα, αυτό θα προκαλέσει σφάλματα.

Τώρα θα περαστεί μία αυθαίρετη λίστα με τα μεγέθη των φίλτρων και η comprehension λίστα που θα δημιουργήσει ένα convolutional layer για κάθε ένα από αυτά τα μεγέθη φίλτρων. Στην συνέχεια στην forward μέθοδο επαναλαμβάνεται η λίστα εφαρμόζοντας σε κάθε convolutional layer για να παρθεί μία λίστα από convolutional outputs, το οποίο επίσης θα τροφοδοτηθεί μέσω της max pooling σε μία comprehension λίστα πριν τα concatenating μαζί και τα περαστεί μέσω της dropout και των γραμμικών layers.

```
class CNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, n_filters, filter_sizes, o
utput_dim,
                 dropout, pad_idx):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx
 = pad_idx)
        self.convs = nn.ModuleList([
                                    nn.Conv2d(in_channels = 1,
                                              out_channels = n_filters,
                                              kernel_size = (fs, embedding_d
im))
                                    for fs in filter_sizes
                                    ])
        self.fc = nn.Linear(len(filter_sizes) * n_filters, output_dim)
        self.dropout = nn.Dropout(dropout)
    def forward(self, text):
        #το text ισούται = [sent len, batch size]
        text = text.permute(1, 0)
        #το text ισούται = [batch size, sent len]
        embedded = self.embedding(text)
```

```
        # το embedded ισούται = [batch size, sent len, emb dim]
        embedded = embedded.unsqueeze(1)
        # το embedded ισούται = [batch size, 1, sent len, emb dim]
        conved = [F.relu(conv(embedded)).squeeze(3) for conv in self.convs]

        #                             το                           conved_n
ισούται = [batch size, n_filters, sent len - filter_sizes[n] + 1]
        pooled = [F.max_pool1d(conv, conv.shape[2]).squeeze(2) for conv in c
onved]
        # το pooled_n ισούται = [batch size, n_filters]
        cat = self.dropout(torch.cat(pooled, dim = 1))
        # το cat ισούται = [batch size, n_filters * len(filter_sizes)]
        return self.fc(cat)
```

Επίσης μπορεί να υλοποιηθεί το παραπάνω μοντέλο χρησιμοποιώντας μονοδιάστατα convolutional layers, όπου η embedding dimension θα είναι το " βάθος" του φίλτρου και ο αριθμός των tokens στην πρόταση θα είναι το πλάτος. Θα εκτελεστούν τα τεστ σε αυτό το κεφάλαιο χρησιμοποιώντας 2-διαστάσεων convolutional μοντέλο.

```
class CNN1d(nn.Module):
    def __init__(self, vocab_size, embedding_dim, n_filters, filter_sizes, o
utput_dim,
                 dropout, pad_idx):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx
 = pad_idx)
        self.convs = nn.ModuleList([
                                    nn.Conv1d(in_channels = embedding_dim,
                                              out_channels = n_filters,
                                              kernel_size = fs)
                                    for fs in filter_sizes
                                    ])
        self.fc = nn.Linear(len(filter_sizes) * n_filters, output_dim)
        self.dropout = nn.Dropout(dropout)
    def forward(self, text):
        # το text ισούται = [sent len, batch size]
        text = text.permute(1, 0)
        # το text ισούται = [batch size, sent len]
```

```
        embedded = self.embedding(text)
        # το embedded ισούται = [batch size, sent len, emb dim]
        embedded = embedded.permute(0, 2, 1)
        # το embedded ισούται = [batch size, emb dim, sent len]
        conved = [F.relu(conv(embedded)) for conv in self.convs]
        #                             το                              conved_n
ισούται = [batch size, n_filters, sent len - filter_sizes[n] + 1]
        pooled = [F.max_pool1d(conv, conv.shape[2]).squeeze(2) for conv in c
onved]
        # το pooled_n ισούται = [batch size, n_filters]
        cat = self.dropout(torch.cat(pooled, dim = 1))
        # το cat ισούται = [batch size, n_filters * len(filter_sizes)]
        return self.fc(cat)
```

Δημιουργείται ένα instance για την CNN κλάση. Μπορεί να αλλαχθεί ο CNN σε CNN1d αν επιλεχθεί να τρέξει το μονοδιάστατο convolutional μοντέλο, σημειώνοντας ότι και τα 2 μοντέλα δίνουν σχεδόν τα ίδια αποτελέσματα.

```
INPUT_DIM = len(TEXT.vocab)
EMBEDDING_DIM = 100
N_FILTERS = 100
FILTER_SIZES = [3,4,5]
OUTPUT_DIM = 1
DROPOUT = 0.5

PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]

model = CNN(INPUT_DIM, EMBEDDING_DIM, N_FILTERS, FILTER_SIZES, OUTPUT_DIM, D
ROPOUT, PAD_IDX)
```

Ελέγχεται ο αριθμός των παραμέτρων του μοντέλου και βλέπουμε ότι έχει τον ίδιο αριθμό με το FastText μοντέλο. Και το CNN αλλά και το CNN1d μοντέλα έχουν τον ίδιο αριθμό παραμέτρων.

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'The model has {count_parameters(model):,} trainable parameters')
The model has 2,620,801 trainable parameters
```

Στην συνέχεια φορτώνονται τα pre-trained embeddings

```
pretrained_embeddings = TEXT.vocab.vectors

model.embedding.weight.data.copy_(pretrained_embeddings)
```

```
tensor([[-0.1117, -0.4966,  0.1631,  ...,  1.2647, -0.2753, -0.1325],
        [-0.8555, -0.7208,  1.3755,  ...,  0.0825, -1.1314,  0.3997],
        [-0.0382, -0.2449,  0.7281,  ..., -0.1459,  0.8278,  0.2706],
        ...,
        [ 0.5117,  0.2138,  0.0543,  ...,  0.3585, -0.5696, -0.3387],
        [ 0.1619,  0.4196, -0.0119,  ..., -0.2165, -0.2771,  0.1419],
        [-0.4678,  1.5997, -1.0589,  ...,  0.0752,  0.2039, -0.8021]])
```

Στην συνέχεια μηδενίζονται τα αρχικά βάρη των αγνώστων και γίνονται Padding τα tokens

```
UNK_IDX = TEXT.vocab.stoi[TEXT.unk_token]

model.embedding.weight.data[UNK_IDX] = torch.zeros(EMBEDDING_DIM)

model.embedding.weight.data[PAD_IDX] = torch.zeros(EMBEDDING_DIM)
```

6.6 Εκπαίδευση μοντέλου

Η εκπαίδευση θα γίνει όπως και πριν, θα αρχικοποιηθούν ο optimizer, η συνάρτηση απώλειας (criterion) και θα μπεί το μοντέλο στην GPU.

```
import torch.optim as optim
optimizer = optim.Adam(model.parameters())
criterion = nn.BCEWithLogitsLoss()
model = model.to(device)
criterion = criterion.to(device)
```

Υλοποιείται μια συνάρτηση που θα υπολογίζει τα αποτελέσματα

```
def binary_accuracy(preds, y):
    """
    Επιστρέφει την ακρίβεια ανά batch, Για παράδειγμα αν πάρεις σωστά 8/10, αυτό επιστρέφει 0.8 και όχι 8
    """
    # στρογγυλοποίηση προβλέψεων στον πλησιέστερο ακέραιο αριθμό
    rounded_preds = torch.round(torch.sigmoid(preds))
    correct = (rounded_preds == y).float() # μετατροπή σε float για διαίρεση
    acc = correct.sum() / len(correct)
    return acc
```

Ορίζεται μία συνάρτηση για να εκπαιδευτεί το μοντέλο.

Σημείωση: αφού σε αυτό το μοντέλο χρησιμοποιείται η dropout, θα πρέπει να χρησιμοποιείται το model.train() για να εξασφαλισθεί ότι η dropout είναι ενεργοποιημένη ενώ γίνετε εκπαίδευση

```
def train(model, iterator, optimizer, criterion):
    epoch_loss = 0
    epoch_acc = 0
    model.train()
    for batch in iterator:
        optimizer.zero_grad()
        predictions = model(batch.text).squeeze(1)
        loss = criterion(predictions, batch.label)
        acc = binary_accuracy(predictions, batch.label)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
        epoch_acc += acc.item()
    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

Όπως και πριν υλοποιείται μια συνάρτηση να για να πει πόσο χρόνο θα κάνει κάθε epoch

```
import time
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

Τέλος, εκπαιδεύεται το μοντέλο.

```
N_EPOCHS = 5
best_valid_loss = float('inf')
for epoch in range(N_EPOCHS):
    start_time = time.time()

    train_loss, train_acc = train(model, train_iterator, optimizer, criterio
n)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)
    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut4-model.pt')
    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%
')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%
')
```

```
100%|█████████▉| 398568/400000 [00:40<00:00, 15785.17it/s]
```

```
Epoch: 01 | Epoch Time: 0m 20s
        Train Loss: 0.656 | Train Acc: 60.47%
         Val. Loss: 0.539 |  Val. Acc: 74.43%
Epoch: 02 | Epoch Time: 0m 19s
        Train Loss: 0.438 | Train Acc: 79.91%
         Val. Loss: 0.352 |  Val. Acc: 85.40%
Epoch: 03 | Epoch Time: 0m 19s
        Train Loss: 0.309 | Train Acc: 86.97%
         Val. Loss: 0.321 |  Val. Acc: 86.19%
Epoch: 04 | Epoch Time: 0m 19s
        Train Loss: 0.224 | Train Acc: 91.16%
         Val. Loss: 0.297 |  Val. Acc: 87.61%
Epoch: 05 | Epoch Time: 0m 20s
        Train Loss: 0.161 | Train Acc: 94.17%
         Val. Loss: 0.300 |  Val. Acc: 87.86%
```

Παίρνουμε αποτελέσματα αντίστοιχα με τα προηγούμενα

```
model.load_state_dict(torch.load('tut4-model.pt'))
test_loss, test_acc = evaluate(model, test_iterator, criterion)
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

```
Test Loss: 0.339 | Test Acc: 85.39%
```

7 Ιστοσελίδα

Για την δημιουργία της ιστοσελίδας χρησιμοποιήθηκε το εργαλείο Docusaurus.

Το Docusaurus είναι ένα δωρεάν εργαλείο βασισμένο στην γλώσσα προγραμματισμού React που σκοπός του είναι να δώσει στον χρήστη τα κατάλληλα εργαλεία για να υλοποιήσει ιστοσελίδα που μπορεί να χρησιμοποιηθεί ως «Documentation site»

7.1 Χτίζοντας την ιστοσελίδα

Η ιστοσελίδα μετά την ανάπτυξη της φιλοξενήθηκε στις υποδομές της Netlify. Ο υπερσύνδεσμος της ιστοσελίδας είναι: https://kb-epdo.netlify.app

7.2 Docusaurus

Το open source project Docusaurus έχει ως κύριο σκοπό να δώσει στην χρήστη την δυνατότητα να υλοποιεί ιστοσελίδες μορφής documentation με όσο απλό και γρήγορο τρόπο. Το Docusaurus επιτρέπει να χρησιμοποιηθούν εργαλεία που ήδη γνωστά, όπως το Markdown ή το MDX για τη σύνταξη των σελίδων. Με την γλώσσα προγραμματισμού React και node.js να λειτουργεί ως πυρήνας του Docusaurus, δίνεται η δυνατότητα να προσαρμοστεί η ιστοσελίδα ώστε να ταιριάζει στην εκάστοτε χρήση. (Dosusaurus, n.d.)

7.3 Netlify

Η Netlify είναι μια εταιρεία που εξειδικεύεται στην φιλοξενία ιστοσελίδων και τεχνολογίας αυτοματισμού με έδρα το Σαν Φρανσίσκο στις ΗΠΑ.

Το Netlify απλοποιεί τη διαδικασία για τους χρήστες να φιλοξενήσουν μία ιστοσελίδα με το Netlify Build, όπου χτίζεις μια εκδοχή (build) του website σου και μπορείς να το ανεβάζεις είτε μέσω Github είτε απλός ως zip αρχείο. (Netlify, n.d.)

7.4 Χτίζοντας την ιστοσελίδα

Σαν πρώτη εργασία θα πρέπει να εγκατασταθούν τα προαπαιτούμενα του Docusaurus για να μπορέσει να τρέξει η ιστοσελίδα αρχικά τοπικά δηλαδή σε localhost. Η συγκεκριμένη

εγκατάσταση έγινε σε Windows αλλά μπορεί να εφαρμοστεί σε Linux και macOS λειτουργικά συστήματα. Τα προαπαιτούμενα εργαλεία είναι:

- το node.js που θα βοηθήσει να στηθεί η ιστοσελίδα και
- το yarn για την διαχείριση της node.js.

Πρώτα γίνεται η εγκατάσταση του yarn μέσω του script που δίνεται στο github. Μετά την εγκατάσταση του yarn πραγματοποιείται η εγκατάσταση του node.js από το .exe αρχείο που παρέχεται στην ιστοσελίδα.

Στην συνέχεια, δημιουργείτε ένας φάκελο "Docusaurus-Websites" στην επιθυμητή τοποθεσία . Με τον συνδυασμό πλήκτρων shift + δεξί κλικ (σε Windows Περιβάλλον) μέσα στον κενό φάκελο, δίνεται η επιλογή να ανοιχθεί το Powershell μέσα στο path του φακέλου.

Από εκεί, εκτελείτε το εξής command για την δημιουργία της αρχικής ιστοσελίδας:

```
npx create-docusaurus@latest my-website classic
```

Το "my-website" είναι το όνομα που θα γίνει η εγκατάσταση της ιστοσελίδας ενώ το "classic" είναι το template που έχει επιλεχθεί. Μόλις ολοκληρωθεί η διαδικασία που έτρεχε μέσω του command παρατηρείται ότι μέσα στον αρχικό φάκελο "Docusaurus-Websites" έχει δημιουργηθεί ένας νέος με το όνομα "my-website"

Η δομή της ιστοσελίδας στον φάκελο my-website είναι η εξής:

```
my-website
├── blog
│   ├── 2019-05-28-hola.md
│   ├── 2019-05-29-hello-world.md
│   └── 2020-05-30-welcome.md
├── docs
│   ├── doc1.md
│   ├── doc2.md
│   ├── doc3.md
│   └── mdx.md
├── src
│   ├── css
│   │   └── custom.css
│   └── pages
│       ├── styles.module.css
│       └── index.js
├── static
│   └── img
├── docusaurus.config.js
├── package.json
├── README.md
├── sidebars.js
└── yarn.lock
```

Σημείωση: περισσότερες πληροφορίες τι είναι το κάθε αρχείο και φάκελος μπορούν να βρεθούν εδώ

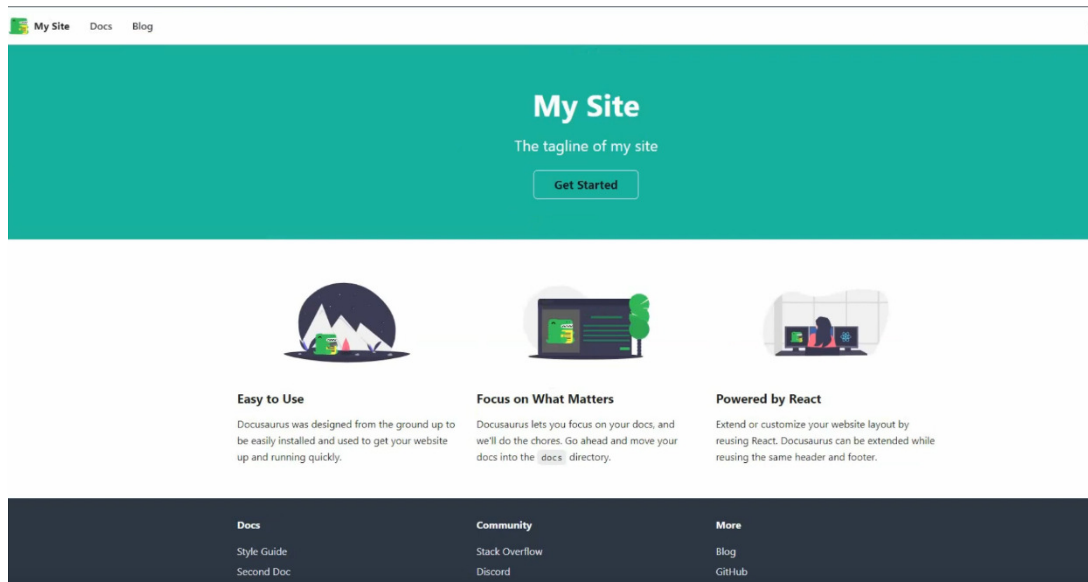
Στην συνέχεια εκτελείτε το παρακάτω command στο cmd.exe για να γίνει η μετακίνηση του χρήστη στον φάκελο "my-website"

```
cd my-website
```

Από εκεί εκτελείτε το command

```
yarn start
```

Όπου τρέχει το website που μόλις δημιουργήθηκε τοπικά στην διεύθυνση localhost:3000.



Το επόμενο βήμα είναι να προστεθεί περιεχόμενο στην ιστοσελίδα. Αυτό πραγματοποιείτε μέσω των markdown αρχεία. Η Markdown είναι μια ελαφριά γλώσσα σήμανσης για τη δημιουργία μορφοποιημένου κειμένου χρησιμοποιώντας ένα πρόγραμμα επεξεργασίας απλού κειμένου. (Wikipedia, n.d.).

Τα markdown αρχεία μπορούν να επεξεργαστούν με το πρόγραμμα Typora που βοηθάει στην σύνταξη των κειμένων σε μορφή markdown. Εφόσον δημιουργηθούν τα έγγραφα, τα προσθέτουμε στον υπο φάκελο "docs" στον φάκελο "my-website". Μόλις γίνει αυτό, παρατηρείται ότι εμφανίζεται το περιεχόμενο στην ιστοσελίδα localhost:3000/docs. (Docs D. , n.d.)

Τέλος, στην συγκεκριμένη περίπτωση ιστοσελίδας πραγματοποιήθηκαν αλλαγές στιλιστικές, προστέθηκαν plugins, ανακατεύθυνση ιστοσελίδας κλπ.

Περισσότερες πληροφορίες από την ιστοσελίδα του Docusaurus:

- Sidebar
- Versioning
- Styling and Layout
- Plugins
- Docs-only mode

7.5 Παρουσίαση ιστοσελίδας

Το επόμενο βήμα μετά την ανάπτυξη της ιστοσελίδας είναι να μεταμορφωθεί στην online υπηρεσία φιλοξενίας ιστοσελίδων Netlify.

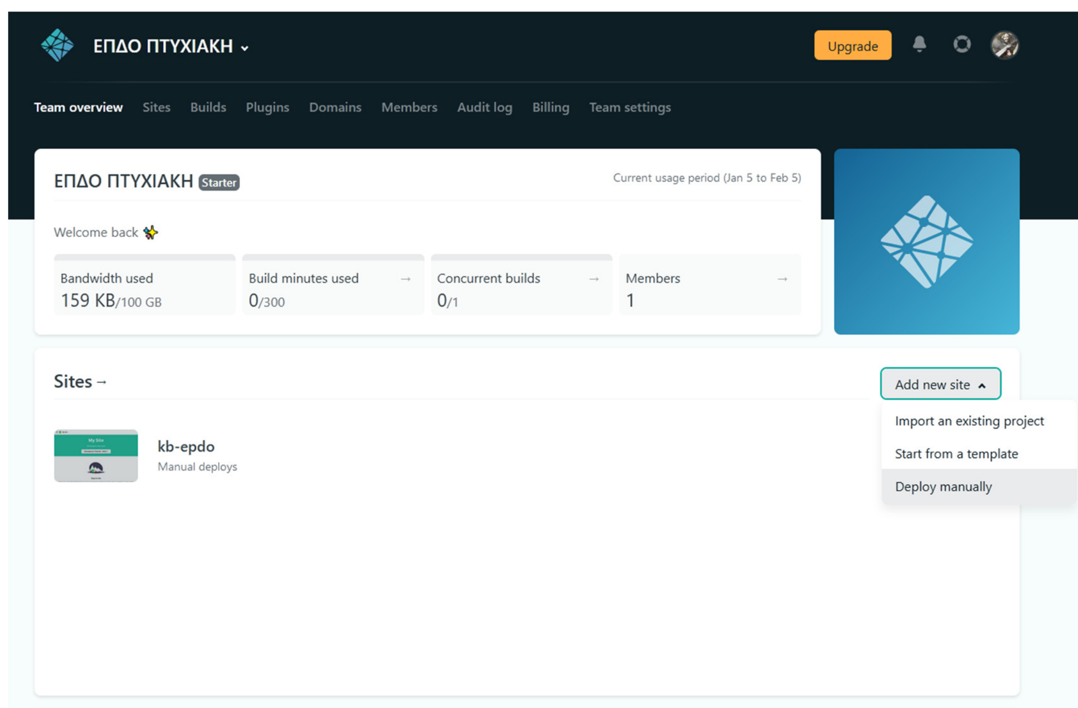
Αρχικά θα πρέπει να δημιουργηθεί ένα build της ιστοσελίδας εφόσον έχει προστεθεί το κατάλληλο περιεχόμενο.

Εκτελείτε στον φάκελο “my-website” το παρακάτω command:

```
yarn run build
```

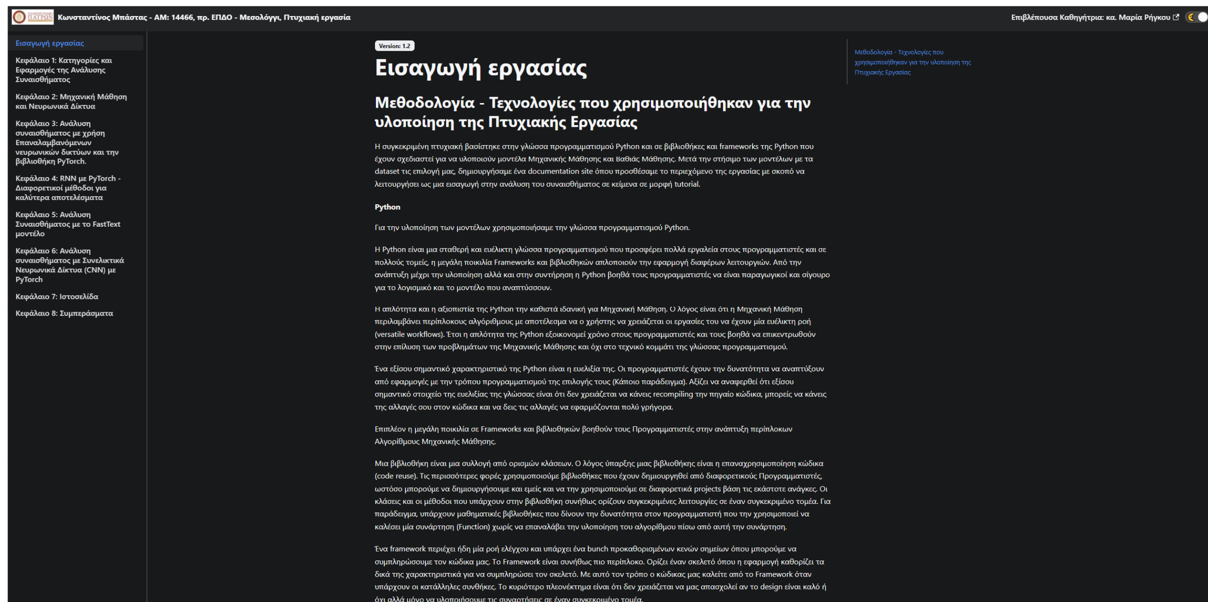
Παρατηρείται ότι δημιουργήθηκε ένας υπό φάκελος “build” στον φάκελο “my-webiste”. Μετέπειτα, δημιουργείται ένα λογαριασμό για να υπάρξει πρόσβαση στις υπηρεσίες της Netlify. Στην συνέχεια γίνεται η επιλογές: “Add new site” --> “Deploy manually”.

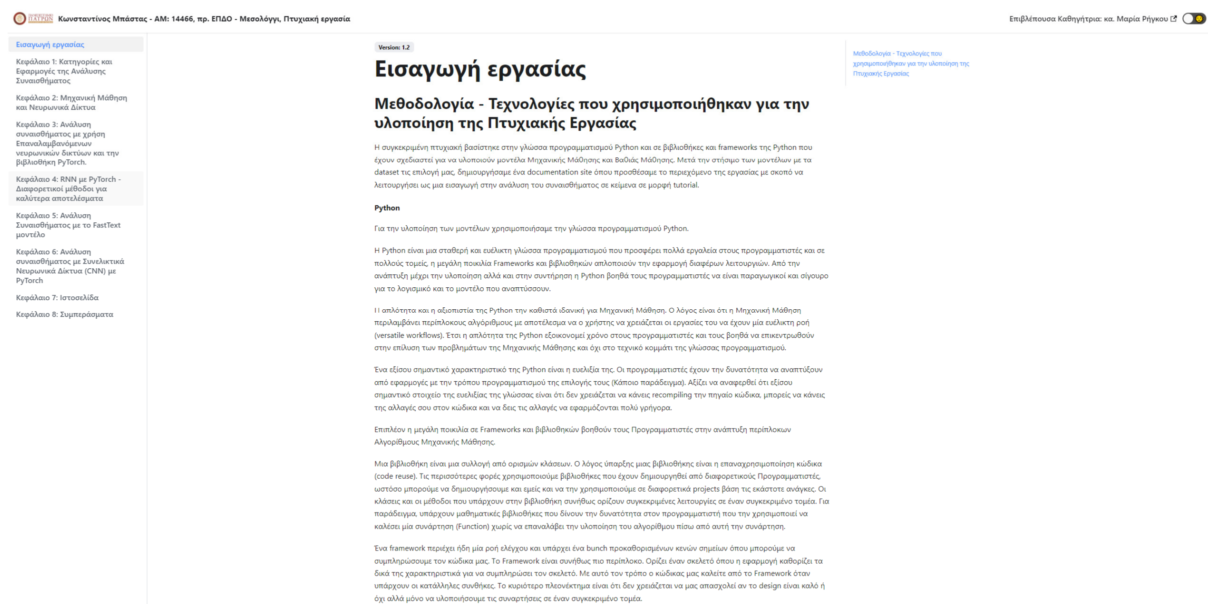
Εκεί δίνεται η δυνατότητα να μεταμορφωθεί το “build” που δημιουργήθηκε προηγουμένους. Επιπλέον υπάρχει η δυνατότητα για την αλλαγή του αρχικού ονόματος της ιστοσελίδας.



Η ιστοσελίδα για την συγκεκριμένη πτυχιακή εργασία είναι η: https://kb-epdo.netlify.app

Ενώ η τελική μορφή σε λευκό και μαύρο θέμα έχει ως εξής:





Συμπεράσματα

Ένα από τα σημαντικότερα συμπεράσματα που προέκυψαν με την παρούσα πτυχιακή εργασία, είναι ότι έχει μεγάλη σημασία η υλοποίηση του σωστού μοντέλου για τον σκοπό της αντίστοιχης επιθυμητής χρήσης. Επιπλέον, διαπιστώθηκε ότι όσο αυξάνεται η περιπλοκότητα ενός μοντέλου Μηχανικής μάθησης, τόσο πιο ακριβή θα είναι και τα εξαγόμενα αποτελέσματα.

Ξεκινώντας από το κεφάλαιο 3, πραγματοποιήθηκε υλοποίηση των Επαναλαμβανόμενων Νευρωνικών Δικτύων σε μια απλοποιημένη εκδοχή τους για ανάλυση συναισθήματος σε κείμενο, με σύνολο δεδομένων που περιέχει κριτικές ταινιών από το Website IMDB. Με βάση την παρακάτω αρχική υλοποίηση, διαπιστώνεται ότι η μέθοδος που χρησιμοποιήθηκε, απέφερε χαμηλά αποτελέσματα. Συγκεκριμένα, περιγράφονται παρακάτω τα αποτελέσματα στην συνάρτηση Απώλεια Εκπαίδευσης 0.712/1 και στην Απώλεια Επαλήθευσης: 45.88%.

```
Epoch: 01 | Epoch Time: 0m 18s
	Train Loss: 0.694 | Train Acc: 50.33%
	 Val. Loss: 0.697 |  Val. Acc: 49.86%
Epoch: 02 | Epoch Time: 0m 15s
	Train Loss: 0.693 | Train Acc: 50.06%
	 Val. Loss: 0.697 |  Val. Acc: 49.86%
Epoch: 03 | Epoch Time: 0m 15s
	Train Loss: 0.693 | Train Acc: 49.93%
	 Val. Loss: 0.697 |  Val. Acc: 50.86%
Epoch: 04 | Epoch Time: 0m 15s
	Train Loss: 0.693 | Train Acc: 49.49%
	 Val. Loss: 0.697 |  Val. Acc: 49.65%
Epoch: 05 | Epoch Time: 0m 16s
	Train Loss: 0.693 | Train Acc: 50.13%
	 Val. Loss: 0.697 |  Val. Acc: 50.87%
```

Πρώτη μέθοδο με Επαναλαμβόμενα Νευρωνικά Δίκτυα: Test Loss: 0.710 | Test Acc: 47.96%

Στο κεφάλαιο 4 πραγματοποιήθηκε μία βελτιωμένη εκδοχή των Επαναλαμβανόμενων Νευρωνικών Δικτύων στο ίδιο σύνολο δεδομένων. Χρησιμοποιήθηκε η μέθοδος packed padded sequences, φορτώθηκαν pre-trained word embeddings, διαφορετικό και πιο αποδοτικό optimizer, διαφορετική αρχιτεκτονική ΕΝΔ, bi-directional ΕΝΔ και multi-layer. Τα ανωτέρω, είχαν ως αποτέλεσμα την συνάρτηση «Απώλεια Εκπαίδευσης 0.305/1 και στην Απώλεια Επαλήθευσης: 87.53%».

```
Epoch: 01 | Epoch Time: 0m 34s
	Train Loss: 0.634 | Train Acc: 63.41%
	 Val. Loss: 0.637 |  Val. Acc: 66.14%
Epoch: 02 | Epoch Time: 0m 36s
	Train Loss: 0.576 | Train Acc: 70.29%
```

```
           Val. Loss: 0.759 |  Val. Acc: 54.56%
Epoch: 03 | Epoch Time: 0m 37s
          Train Loss: 0.453 | Train Acc: 79.10%
           Val. Loss: 0.850 |  Val. Acc: 73.14%
Epoch: 04 | Epoch Time: 0m 38s
          Train Loss: 0.387 | Train Acc: 83.50%
           Val. Loss: 0.391 |  Val. Acc: 84.06%
Epoch: 05 | Epoch Time: 0m 38s
          Train Loss: 0.327 | Train Acc: 86.07%
           Val. Loss: 0.314 |  Val. Acc: 86.69%
```

Δεύτερη Μέθοδο με Επαναλαμβανόμενα Νευρωνικά Δίκτυα: Test Loss: 0.337 | Test Acc: 87.78%

Στο κεφάλαιο 5 χρησιμοποιήθηκε το μοντέλο "FastText" όπου λάβαμε παρόμοια αποτελέσματα με την βελτιωμένη εκδοχή των ΕΝΔ του κεφαλαίου 4, ωστόσο πραγματοποιείται πολύ πιο γρήγορα η εκπαίδευση και με μικρότερο όγκο κώδικα. Συγκεκριμένα, περιγράφονται παρακάτω τα αποτελέσματα στην συνάρτηση Απώλεια Εκπαίδευσης 0382/1 και στην Απώλεια Επαλήθευσης: 85.18%.

```
Epoch: 01 | Epoch Time: 0m 10s
      Train Loss: 0.688 | Train Acc: 59.34%
       Val. Loss: 0.637 |  Val. Acc: 71.07%
Epoch: 02 | Epoch Time: 0m 9s
      Train Loss: 0.649 | Train Acc: 74.30%
       Val. Loss: 0.508 |  Val. Acc: 75.98%
Epoch: 03 | Epoch Time: 0m 10s
      Train Loss: 0.575 | Train Acc: 80.16%
       Val. Loss: 0.427 |  Val. Acc: 80.51%
Epoch: 04 | Epoch Time: 0m 10s
      Train Loss: 0.495 | Train Acc: 84.49%
       Val. Loss: 0.383 |  Val. Acc: 83.96%
Epoch: 05 | Epoch Time: 0m 9s
      Train Loss: 0.430 | Train Acc: 87.39%
       Val. Loss: 0.376 |  Val. Acc: 85.77%
```

FastText Μοντέλο: Test Loss: 0.382 | Test Acc: 85.52%

Στο τελευταίο κεφάλαιο 6 χρησιμοποιούμε το είδος νευρωνικού δικτύου που ονομάζεται Συνελεκτικά Νευρωνικά Δίκτυα. Τα ΣΝΔ συνήθως υλοποιούνται σε δεδομένα φωτογραφιών ωστόσο μπορεί να χρησιμοποιηθεί και σε ανάλυση συναισθήματος σε κείμενο όπως αναφέρεται στην μελέτη "(Kim, Convolutional Neural Networks for Sentence Classification". Πήραμε αποτελέσματα στην συνάρτηση Απώλεια Εκπαίδευσης 0347/1 και στην Απώλεια Επαλήθευσης: 85.05%.

```
Epoch: 01 | Epoch Time: 0m 20s
        Train Loss: 0.656 | Train Acc: 60.47%
         Val. Loss: 0.539 |  Val. Acc: 74.43%
Epoch: 02 | Epoch Time: 0m 19s
        Train Loss: 0.438 | Train Acc: 79.91%
         Val. Loss: 0.352 |  Val. Acc: 85.40%
Epoch: 03 | Epoch Time: 0m 19s
        Train Loss: 0.309 | Train Acc: 86.97%
         Val. Loss: 0.321 |  Val. Acc: 86.19%
Epoch: 04 | Epoch Time: 0m 19s
        Train Loss: 0.224 | Train Acc: 91.16%
         Val. Loss: 0.297 |  Val. Acc: 87.61%
Epoch: 05 | Epoch Time: 0m 20s
        Train Loss: 0.161 | Train Acc: 94.17%
         Val. Loss: 0.300 |  Val. Acc: 87.86%
```

Συνελικτικά Νευρωτικά Δίκτυα: Test Loss: 0.339 | Test Acc: 85.39%

Μελλοντικές επεκτάσεις & Βελτιώσεις

- Εκτέλεση κώδικα μέσω Jupiter Notebooks που είναι ενσωματωμένα στην ιστοσελίδα
- Υλοποίηση νέων τεχνικών Μηχανικής Μάθησης για μεγαλύτερη ακρίβεια στα αποτελέσματα
- Υλοποίηση τεχνικών σε dataset (Ελληνική γλώσσα)
- Δημιουργία web application για ανάλυση συναισθήματος

Τεχνικές δυσκολίες εργασίας

- Μη διαθέσιμο hardware (GPU) για Machine Learning με αποτέλεσμα να στραφώ στην λύση του cloud
- Σφάλματα στον κώδικα
- Περιορισμένο υλικό στα Ελληνικά και περίπλοκες έννοιες
- Μη δυνατότητα ενσωμάτωσης του Google Colab στην ιστοσελίδα
- Δημιουργία γραφημάτων

ΒΙΒΛΙΟΓΡΑΦΙΑ

Jeffrey Pennington, Richard Socher, Christopher D. Manning . (n.d.). *GloVe: Global Vectors for Word Representation*. Retrieved from nlp.stanford.edu: https://nlp.stanford.edu/projects/glove/

Anthony Rios, Ramakanth Kavuluru 2 . (2015). Convolutional Neural Networks for Biomedical Text Classification: Application in Indexing Biomedical Articles . *PubMed*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov. (2016). Bag of Tricks for Efficient Text Classification. *arXiv.org* .

Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov. (2016). Bag of Tricks for Efficient Text Classification.

Brownlee, J. (n.d.). *Deep Learning for Natural Language Processing Develop Deep Learning Models for Natural Language in Python.*

Brownlee, J. (n.d.). *Loss and Loss Functions for Training Deep Learning Neural Networks*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

cepe-eua. (n.d.). *Ποια είναι η διαφορά μεταξύ κλίσης και στοχαστικής κλίσης;*. Retrieved from cepe-eua.org: https://cepe-eua.org/el/poia-einai-i-diafora-metaxy-klisis-kai-stohastikis-klisis-RK8SPOtG

Chollet, F. (2018). *Deep Learning with Python.* Manning Publications; 1st edition.

Colah. (n.d.). *Understanding LSTM Networks*. Retrieved from colah.github.io: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Demystified, D. L. (n.d.). *deep learning de mystified*. Retrieved from deeplearningdemystified.com: https://deeplearningdemystified.com/article/nlp-1

Docs, P. (n.d.). *www.pytorch.org*. Retrieved from Pytorch Docs: https://pytorch.org/text/_modules/torchtext/data/field.html

Educative. (n.d.). *Educative.io*. Retrieved from Educative.io: https://www.educative.io/edpresso/what-is-a-boilerplate-code

geeksforgeeks.org. (2021, March 21). Retrieved from https://www.geeksforgeeks.org/supervised-unsupervised-learning/

Goldberg, Y. (n.d.). *Neural Network Methods forNatural Language Processing.* Morgan & Claypool Publishers.

Gomez, R. (n.d.). *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. Retrieved from https://gombru.github.io/2018/05/23/cross_entropy_loss/: https://gombru.github.io/2018/05/23/cross_entropy_loss/

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *arXiv.org*.

Guide, C. P. (n.d.). *portingguide.readthedocs.io*. Retrieved from portingguide.readthedocs.io: https://portingguide.readthedocs.io/en/latest/iterators.html

itdxer. (n.d.). *stackechange*. Retrieved from stats.stackexchange.com: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

John McGonagle, Christopher Williams, Jimin Khim. (n.d.). *brilliant.org*. Retrieved from brilliant.org: https://brilliant.org/wiki/recurrent-neural-network/

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification.

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification.

Kumar, B. (n.d.). *python guides*. Retrieved from pythonguides.com: https://pythonguides.com/adam-optimizer-pytorch/

Mika V. Mäntylä, Daniel Graziotin, Miikka Kuutila. (n.d.). The Evolution of Sentiment Analysis -A Review of Research Topics, Venues, and Top Cited Papers.

MonkeyLearn. (n.d.). *MonkeyLearn Blog*. Retrieved from MonkeyLearn.com: https://monkeylearn.com/blog/word-embeddings-transform-text-numbers/

Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom. (2014). A Convolutional Neural Network for Modelling Sentences.

Olah, C. (n.d.). *https://colah.github.io*. Retrieved from https://colah.github.io/posts/2014-07-Conv-Nets-Modular/

Oren Pereg, Moshe Wasserblat, Daniel Korat. (n.d.). *aithority.com*. Retrieved from aithority.com: https://aithority.com/guest-authors/introducing-aspect-based-sentiment-analysis-in-nlp-architect/

Pytorch. (n.d.). *Pytorch*. Retrieved from Pytorch Docs: https://pytorch.org/text/_modules/torchtext/data/dataset.html

repository.kallipos. (n.d.). Retrieved from https://repository.kallipos.gr/bitstream/11419/3382/1/02_chapter_04.pdf

Spacy. (n.d.). *Spacy*. Retrieved from Spacy.io: https://spacy.io/api/tokenizer

Wen-tau Yih, X. H. (n.d.). Semantic Parsing for Single-Relation Question Answering.

Wikipedia. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Sentiment_analysis

Wikipedia. (n.d.). Retrieved from Wikipedia.org: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol

Wikipedia - Machine Learning. (n.d.). Retrieved from Wikipedia : https://el.wikipedia.org/wiki/%CE%9C%CE%B7%CF%87%CE%B1%CE%BD%CE%B9%CE%BA%CE%AE_%CE%BC%CE%AC%CE%B8%CE%B7%CF%83%CE%B7

Wikipedia. (n.d.). *Vanishing gradient problem*. Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Vanishing_gradient_problem

Wilson, A. (2019, September 29). *towardsdatascience.com.* Retrieved from https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590

Xi Ouyang, Pan Zhou, Cheng Hua Li, Pan Zhou. (2015). Sentiment Analysis Using Convolutional Neural Network.

Xiang Zhang, Junbo Zhao, Yann LeCun. (2015). Character-level Convolutional Networks for TextClassification.

Yelong Shen, X. H. (2014). Learning Semantic Representations Using Convolutional Neural Networks for Web Search .

Youtube, d. (Director). (2017). *Unsupervised Learning explained* [Motion Picture].

Κωστόπουλος, Γ. Κ. (n.d.). *thalis.math.upatras.gr*. Retrieved from thalis.math.upatras.gr: https://thalis.math.upatras.gr/~sotos/Phd%20Thesis-Georgios%20Kostopoulos.pdf

Λιθοξοΐδης, Σ. (2019). ΕΠΑΝΑΛΑΜΒΑΝΟΜΕΝΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΚΑΙ ΔΗΜΙΟΥΡΓΙΚΗ ΓΡΑΦΗ: ΜΙΑ ΕΚΠΑΙΔΕΥΤΙΚΗ ΣΥΖΕΥΞΗ .

Πνευματικά δικαιώματα

Copyright © Πανεπιστήμιο Πατρών. Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1988 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον.

Κωνσταντίνος Μπάστας, [2022]