

## 1. Αρχιτεκτονική του μοντέλου

Θέλουμε να φτιάξουμε ένα απλό εκπαιδευμένο σύστημα chatbot ετυμολογικού λεξικού με ένα απλό Τεχνητό Νευρωνικό Δίκτυο, που θα απαντά σε ερωτήσεις χρηστών βάσει προκαθορισμένων προτύπων ερωτήσεων της βάσης δεδομένων μας και θα επιλέγει την προβλεπόμενη κατηγορία για να εντοπίζει την προκαθορισμένη απάντηση από την βάση δεδομένων μας.

Από την βάση δεδομένων μας chatbot\_liberx.db το μοντέλο μας χρησιμοποιεί τις πληροφορίες κατηγοριοποιημένες σε τρεις βασικούς πίνακες tags για την ετικέτα-θεματική της ερωταπάντησης, patterns που είναι οι προκαθορισμένες ερωτήσεις που χρησιμοποιεί ως βάση κατανόησης της ερώτησης που μπορεί να θέσεις ο χρήστης και responses που είναι η απάντηση που πρέπει να δώσει το chatbot. Για την εκπαίδευση του μοντέλου θα χρησιμοποιήσουμε tokenization και stemming για επεξεργασία φυσικής γλώσσας των ερωτήσεων έτσι ώστε το μοντέλο να είναι πεπαίδευμένο για να κατανοήσει τις ερωτήσεις των χρηστών και να εξάγει την απάντηση βάσει της πρόβλεψης της ετικέτας. Κάθε πρόταση των patterns πρέπει να μετατραπεί σε διανυσματική μορφή έτσι ώστε να είναι συμβατή και επεξεργάσιμη από το μοντέλο μας. Τα δεδομένα μας θα τροφοδοτήσουν ένα νευρωνικό δίκτυο τριών επιπέδων (Είσοδος, Κρυφό επίπεδο, Έξοδος) για να εκπαίδευτη με την χρήση Διασταυρούμενης Εντροπίας (Cross entropy loss) και Adam Optimizer (Βελτιστοποιητή) για την καταμέτρηση απώλειας και βελτιστοποίησης του μοντέλου σε 1000 εποχές/επαναλήψεις εκμάθησης. Μετά την εκπαίδευση τα δεδομένα θα αποθηκεύονται σε ένα αρχείο data.pth και θα τίθεται σε evaluation mode έτσι ώστε να μην λειτουργεί σε περιβάλλον εκμάθησης με μεγάλο ποσοστό τυχαιότητας, αλλά να λειτουργεί βάσει των πεπαίδευμένων δεδομένων. Ο χρήστης όταν θέτει μία ερώτηση, αυτή θα γίνεται tokenized και θα μετατρέπεται σε διάνυσμα έτσι ώστε να μπορεί να επεξεργαστεί από το μοντέλο και με βάσει τα πεπαίδευμένα δεδομένα του να προβλέπει την κατάλληλα ετικέτα κατηγορίας για την εξαγωγή της απάντησης. Προϋπόθεση στο να δώσει το μοντέλο απάντηση θα είναι 75% βεβαιότητα.

Το (Τεχνητό) Νευρωνικό Δίκτυο ((T)ΝΔ) είναι ένα εμπνευσμένο από τον ανθρώπινο εγκέφαλο υπολογιστικό μοντέλο με τεχνητούς νευρώνες (υπολογιστικά νήματα) που διαρθρώνονται σε επίπεδα (layers). Οι νευρώνες επεξεργάζονται και εκπαιδεύονται από τα δεδομένα και εξάγουν κατηγορίες, σχέσεις και μοτίβα αυτών έτσι ώστε να μπορούν να κάνουν μία πρόβλεψη. Η κλάση NeuralNet του μοντέλου μας είναι ένα πλήρες συνδεδεμένο νευρωνικό δίκτυο που οι νευρώνες του ενός επιπέδου συνδέονται με τους νευρώνες του επόμενου και επομένως χαρακτηρίζεται ως ένα feedforward neural network.

Το ΝΔ μας θα έχει τρία επίπεδα, μία είσοδο, ένα κρυφό επίπεδο και μία έξοδο. Η είσοδος (πρώτο επίπεδο) θα δέχεται το διάνυσμα των λέξεων των ερωτήσεων και θα έχει μέγεθος εισόδου ίσο με όλες τις δυνατές λέξεις που προκύπτουν έπειτα από το stemming των patterns. Το κρυφό επίπεδο θα έχει 8 νευρώνες και θα χρησιμοποιήσουμε Rectified Linear Unit ενεργοποιητή συνάρτησης για να εισάγει μη γραμμικότητα στην στο σύστημα, από το να έχουμε απλώς γραμμικές συναρτήσεις έτσι ώστε να έχουμε πολυπλοκότητα. Η ReLU ορίζεται ως  $f(x) = \max(0, x)$  και επομένως επιστρέφει  $x$  αν αυτό είναι μεγαλύτερο του μηδενός αλλιώς επιστρέφει μηδέν. Το τρίτο επίπεδο της εξόδου έχει νευρώνες όσους και τα συνολικά tags μας και δεν θα χρησιμοποιεί Softmax καθώς αυτή συμπεριλαμβάνεται στην Διασταυρούμενη Εντροπία.

Κατά την εκμάθηση το μοντέλο θα επεξεργάζεται τα δεδομένα μας σε 1000 εποχές για να μειωθεί το σφάλμα. Το δίκτυο θα μαθαίνει με μηχανισμό Backpropagation, δηλαδή έναν

αλγόριθμο οπισθοδιάδοσης του σφάλματος με στόχο την μείωση του. Έτσι, θα γίνεται forward pass για την πρόβλεψη και τον υπολογισμό του σφάλματος το οποίο θα διαδίδεται προς τα πίσω (backward pass) για τον υπολογισμό παραγώγων βάσει του σφάλματος και βάσει των παραγώγων να γίνεται η αναπροσαρμογή των βαρών έτσι ώστε η πρόβλεψη να βελτιώνεται. Η συνάρτηση κόστους Cross Entropy Loss (Διασταυρούμενη Εντροπία) ταξινομεί τις πιθανότητες, έτσι ώστε αν δώσει χαμηλή πιθανότητα στην σωστή κατηγορία το κόστος να είναι υψηλό, και αν δώσει υψηλή το κόστος να είναι χαμηλό. Η ενημέρωση των βαρών θα γίνεται από τον Adam Optimizer βάσει των παραγώγων, προσαρμόζοντας τον ρυθμό μάθησης με τις παραμέτρους, υπολογίζοντας έναν κινούμενο μέσο όρο της παραγώγου και του τετραγώνου του, με σταθερή σύγκλιση.

Η σειρά εκτέλεσης θα είναι όπως το κάτωθι διάγραμμα:

Εισαγωγή πρότασης > (Tokenization>Stemming>Διάνυσμα) > Επίπεδο εισόδου > Κρυφό Επίπεδο > Επίπεδο εξόδου > Διασταυρούμενη Εντροπία > Οπισθοδρόμηση > Βελτιστοποιητής > Ενημέρωση βαρών.

Επομένως, τα δεδομένα εισόδου περνούν στο μοντέλο στου οποίου τους νευρώνες εφαρμόζεται η συνάρτηση ενεργοποίησης ReLU έτσι ώστε να πάμε από την γραμμική σχέση  $y = Wx + b$  σε μη γραμμικές σχέσεις που δίνουν τη δυνατότητα πολύπλοκων ταξινομήσεων και κατανόησης της πρόβλεψης. Το μοντέλο μας βγάζει τις προβλέψεις του σε ακατέργαστες τιμές (logits) οι οποίες περνάνε στην Διασταυρούμενη Εντροπία, η οποία μπορεί να τις διαχειριστεί καθώς έχει ενσωματωμένη την Softmax, και υπολογίζεται η απώλεια βάση της πρόβλεψης και της σωστής απάντησης. Η απώλεια με backpropagation χρησιμοποιείται από τον Βελτιστοποιητή για την ορθή ενημέρωση των βαρών έχοντας μηδενίσει τους προηγούμενους παραγώγους. Η Softmax θα χρησιμοποιηθεί κατά την εκτέλεση του μοντέλου οπότε και τα logits της εξόδου θα μετατραπούν σε πιθανότητες και αν η πιθανότητα υπερβαίνει το κατώτατο όριο που θα θέσουμε το bot θα δίνει απάντηση.

## 2. Βάση Δεδομένων

Για την βάση δεδομένων μας θα χρειαστούμε την βιβλιοθήκη sqlite3 της Python.

```
import sqlite3
```

Δημιουργούμε την βάση δεδομένων σε SQLite μας chatbot\_liberX.db με την connect() και ανοίγουμε σύνδεση με την ΒΔ και την αποθηκεύουμε στο αντικείμενο con. Με τον cursor μπορούμε να εκτελέσουμε εντολές στην ΒΔ.

```
con = sqlite3.connect("chatbot_liberx.db")
cur = con.cursor()
```

Έπειτα με τον cursor εκτελούμε εντολές execute() για την δημιουργία τριών πινάκων tags, patterns και responses αν δεν υπάρχουν (CREATE TABLE IF NOT EXISTS). Στον πίνακα tags καταχωρούμε id που είναι ένα ακέραιο primary key αυτόματα αυξανόμενο ανά κάθε εγγραφή στον πίνακα (INTEGER PRIMARY KEY AUTOINCREMENT) και tag για την ετικέτα της κάθε ερωταπάντησης που είναι μία καταχώρηση μοναδικής συμβολοσειράς και δεν μπορεί να είναι κενή (TEXT UNIQUE NOT NULL). Στον πίνακα patterns καταχωρούμε id που είναι ένα ακέραιο primary key αυτόματα αυξανόμενο για κάθε καταχώρηση pattern και tag\_id που είναι μία ακέραιη μη κενή τιμή (INTEGER NOT NULL) και με την εντολή

FOREIGN KEY (tag\_id) REFERENCES tags(id) διασφαλίζουμε ότι κάθε pattern αντιστοιχεί σε ένα tag αφού έχουμε δημιουργήσει μία σχέση μεταξύ tag\_id και id των tags. Ακόμα, καταχωρούμε pattern που είναι μία γραμματοσειρά μη κενή για κάθε pattern-ερώτηση (TEXT NOT NULL). Στον πίνακα responses, καταχωρούμε τα ίδια ακριβώς στοιχεία με τον προηγούμενο πίνακα patterns για tag\_id και id και μία γραμματοσειρά μη κενή response για κάθε απάντηση. Με αυτόν τον τρόπο αποθηκεύουμε τις ερωτήσεις που θα χρησιμοποιήσει το μοντέλο μας για να κατανοήσει τα εισερχόμενα μηνύματα και τις απαντήσεις που θα χρησιμοποιήσει για την τυχαία επιλογή της πιθανής απάντησης και ανάγουμε τα δύο αυτά στοιχεία σε μία κατηγορία.

```
cur.execute("""CREATE TABLE IF NOT EXISTS tags (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    tag TEXT UNIQUE NOT NULL
)""")

cur.execute("""CREATE TABLE IF NOT EXISTS patterns (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    tag_id INTEGER NOT NULL,
    pattern TEXT NOT NULL,
    FOREIGN KEY (tag_id) REFERENCES tags(id)
)""")

cur.execute("""CREATE TABLE IF NOT EXISTS responses (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    tag_id INTEGER NOT NULL,
    response TEXT NOT NULL,
    FOREIGN KEY (tag_id) REFERENCES tags(id)
)""")
```

Παράδειγμα εισαγωγής δεδομένων:

```
# Εισαγωγή Tags
cur.execute("INSERT INTO tags (tag) VALUES ('accola')") #47
# Εισαγωγή Patterns για accol
cur.executemany("INSERT INTO patterns (tag_id, pattern) VALUES (?, ?)", [
    (47, 'What is accol?'),
    (47, 'Define accol.'),
    (47, 'Tell me about accol.'),
    (47, 'What is called accol in latin?'),
    (47, 'What accol means?'),
    (47, 'What is the etymology of accol?')
])
# Εισαγωγή Responses για accol
cur.executemany("INSERT INTO responses (tag_id, response) VALUES (?, ?)", [
    (47, 'Immigrant (accola), because immigrating (advenire) one tills (colere) the land.')
])
```

Με την εντολή execute() του cursor εισάγουμε στον πίνακα tags ως tag την τιμή accol (INSERT INTO tags (tag) VALUES ('accola')) και λόγω της ρύθμισης AUTOINCREMENT αυτόματα αυξάνεται το id σε 47 το οποίο βάζουμε σε σχόλιο από δίπλα για να έχουμε κατά νου την αριθμηση. Ετσι καταχωρούμε μία ετικέτα. Έπειτα με την εντολή executemany() για την ταυτόχρονη εισαγωγή πολλαπλών γραμμών του cursor εισάγουμε το tag\_id που είναι το

id της ετικέτας (INSERT INTO patterns (tag\_id, pattern) VALUES (?, ?) για την σύνδεση των δύο αυτών δεδομένων και τις ερωτήσεις pattern που θα χρησιμοποιήσει το μοντέλο ως πρότυπο για την κατανόηση των ερωτήσεων που θα εισάγει ο χρήστης στο chatbot. Έτσι εισάγουμε της ερωτήσεις των αντίστοιχων tags και τις συνδέουμε με το tag\_id σε μορφή tuples σε λίστα. Τέλος, με την ίδια εντολή executemany() εισάγουμε το tag\_id της αντίστοιχης κατηγορίας και την απάντηση που πρέπει να δώσει το chatbot.

Οι πίνακες των δεδομένων μας είναι της εξής μορφής:

The image shows three database tables side-by-side:

- tags** table (155 rows):
 

	id	tag
1	1	greeting
2	2	goodbye
3	3	identity
4	4	etymologies
5	5	thanks
6	6	homo
7	7	aeros
8	8	auctor
9	9	actor
10	10	alumnus
11	11	amitus
12	12	amabilis
13	13	amatius
14	14	asitus
15	15	angustus
16	16	aer
17	17	alacor
- patterns** table (923 rows):
 

	id	tag_id	pattern
1	1	1	Hi
2	2	1	Hey
3	3	1	How are you
4	4	1	Is anyone there?
5	5	1	Hello
6	6	1	Good day
7	7	2	Bye
8	8	2	See you later
9	9	2	Goodbye
10	10	2	Bye, thank you for your answer!
11	11	3	Who is Iakobus Hispalensis
12	12	3	Who are you?
13	13	3	Tell me about you.
14	14	4	What is Etymologies?
15	15	4	What is your work Etymologies?
16	16	4	Tell me about your Etymologies.
17	17	4	What are etymologies?
- responses** table (164 rows):
 

	id	tag_id	response
1	1	1	Save, or I can just say Welcome!
2	2	1	Ave, Quonmodo valer? How are you?
3	3	1	Hi there, do you have any question on latin etymology?
4	4	1	Save, Quid agis??
5	5	2	Bye! Come back again soon to improve your latin!
6	6	2	Ava, come back again to improve your knowledge on latin ...
7	7	2	Good bye, see you later!
8	8	2	Thanks for chatting!
9	9	3	I am the last scholar of the ancient world. I was born arou...
10	10	4	Of my writings, the Etymologies are by far the most ...
11	11	5	My pleasure!
12	12	5	Anytime, I am happy to help you!
13	13	5	Habeo gratiam!
14	14	6	Benigne, very kind of you!
15	15	6	Human being (homo) is so called from humanity ...
16	16	7	Aeros is a strong and wise man.
17	17	8	Author (auctor), so called from augmenting (augere); ...

### 3. Εργαλεία επεξεργασίας φυσικής γλώσσας

Για την δημιουργία του κώδικα με τα εργαλεία επεξεργασία φυσικής γλώσσας, θα χρειαστούμε να κατεβάσουμε την Numerical Python βιβλιοθήκη και την Natural Language Toolkit βιβλιοθήκη. Η πρώτη θα μας χρησιμεύσει για την δημιουργία διανυσματικών πινάκων, και η δεύτερη τόσο για να κατεβάσουμε από το υπο-πακέτο της Porter Stemmer τον αλγόριθμο Porter Stemming για την αποκοπή των λέξεων στην ριζική τους μορφή όσο και για την χρήση συναρτήσεων που θα μας βοηθήσουν στην τμηματοποίηση προτάσεων στις λέξεις που τις αποτελούν (tokenization).

```
import numpy as np
import nltk
from nltk.stem.porter import PorterStemmer
```

Δημιουργούμε ένα αντικείμενο της κλάσης PorterStemmer() από την βιβλιοθήκη NLTK.

```
stemmer = PorterStemmer()
```

```
Unit tests for the Porter stemmer
>>> from nltk.stem.porter import *
Create a new Porter stemmer.
>>> stemmer = PorterStemmer()
```

Πηγή <https://www.nltk.org/howto/stem.html>

Ορίζουμε μία συνάρτηση tokenize() που θα δέχεται ως όρισμα μία πρόταση sentence και επιστρέψει το αποτέλεσμα της συνάρτησης της NLTK nltk.word\_tokenize() με όρισμα την sentence. Έτσι κάθε πρόταση της βάσης δεδομένων μας για την εκπαίδευση του μοντέλου

και κάθε πρόταση που εισάγει ο χρήστης στο chatbot διαιρείται εις τα εξ ων συνετέθη στοιχεία της, λέξεις, σημεία στίξης, αριθμούς και άλλα διακριτά στοιχεία.

```
def tokenize(sentence):  
  
    return nltk.word_tokenize(sentence)
```

nltk.tokenize.word\_tokenize(text, language='english', preserve\_line=False) [source]  
Return a tokenized copy of `text`, using NLTK's recommended word tokenizer (currently an improved TreebankWordTokenizer along with PunktSentenceTokenizer for the specified language).  
Parameters  
• `text` (`str`) – text to split into words  
• `language` (`str`) – the model name in the Punkt corpus  
• `preserve_line` (`bool`) – A flag to decide whether to sentence tokenize the text or not.

Πηγή [https://www.nltk.org/api/nltk.tokenize.word\\_tokenize.html](https://www.nltk.org/api/nltk.tokenize.word_tokenize.html)

Ορίζουμε μία συνάρτηση `stem()` με όρισμα `word` που θα επιστρέφει το αποτέλεσμα της συνάρτησης της NLTK `stemmer.stem()` με όρισμα την λέξη `word` με μικρά γράμματα `.lower()`. Με αυτήν την συνάρτηση το μοντέλο μας μπορεί να μειώνει την κάθε λέξη που προήλθε από το tokenization στην ριζική μορφή της απλουστεύοντας την πολυπλοκότητα διαφορετικών μορφών της ίδιας εννοιολογικά λέξης και βελτιώνοντας την γενίκευση.

```
def stem(word):  
  
    return stemmer.stem(word.lower())
```

Stem a word.  
>>> print(stemmer.stem("running"))  
run

Πηγή <https://www.nltk.org/howto/stem.html>

**Σημείωση:** Ο πιο σωστός κάδικας είναι αν υπήρχε μία εντολή `if` όπου θα έβρισκε κάθε `word` σε μία λίστα με τις λατινικές λέξεις της βάσης δεδομένων και θα έκανε stemming μέσω της συνάρτησης `stem()` του υποπακέτου `cltk.stem.lat` του CLTK. Όμως το `cltk` θέλει Python 9 και οι υπόλοιπες βιβλιοθήκες δεν λειτουργούν ορθώς σε αυτή την έκδοση της Python.

```
if word in latin_words:  
    return latin_stem(word.lower())  
else:  
    return stemmer.stem(word.lower())
```

cltk.stem.lat.stem(word)  
Stem each word of the Latin text.

>>> stem('interdum')  
'interd'  
>>> stem('mercaturis')  
'mercatur'

Return type: str

Ορίζουμε μία συνάρτηση `bag_of_words()` με ορίσματα `tokenized_sentence` και `words`. Κατά την συνάρτηση θα εφαρμόζεται stemming σε κάθε `word` που υπάρχει στην `tokenized_sentence`. Αρχικοποιούμε έναν διανυσματικό πίνακα `bag` με αρχικά μηδενικές τιμές της Numerical Python που δέχεται τιμές 0 ή 1 και έχει για μέγεθος των λέξεων και ο τύπος των αριθμών είναι δεκαδικός των 32-bit, καθώς οι νευρωνικοί υπολογισμοί γίνονται

σε δεκαδικούς για μεγαλύτερη ευελιξία, αλλά και βελτιστοποίηση των πράξεω με GPU ή CPU. Έπειτα, χρησιμοποιούμε έναν βρόχο όπου για κάθε θέση idx και λέξη w κατά την αρίθμηση της θέση κάθε στοιχείου των words αν η λέξη υπάρχει στην sentence\_words τότε η θέση idx του διανυσματικού πίνακα παίρνει την τιμή 1. Με την συνάρτηση enumerate() της Python αριθμούμε τα στοιχεία ενός iterable και μας επιστρέφει έναν iterator που σε κάθε επανάληψη δίνει ένα tuple (index, value), δηλαδή την θέση του στοιχείου και το στοιχείο του iterable.

Με αυτήν την συνάρτηση μπορούμε να δημιουργήσουμε ένα διάνυσμα αναπαράστασης των γνωστών λέξεων του μοντέλου στις οποίες έχει εκπαιδευτεί και των λέξεων που εισάγει ο κάθε χρήστης έτσι ώστε να μπορούν να επεξεργαστούν σε αριθμητική μορφή από το νευρωνικό δίκτυο.

```
def bag_of_words(tokenized_sentence, words):
    sentence_words = [stem(word) for word in tokenized_sentence]
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1

    return bag
```

`numpy.zeros(shape, dtype=float, order='C', *, like=None)`  
Return a new array of given shape and type, filled with zeros.

#### Parameters:

`shape : int or tuple of ints`

Shape of the new array, e.g., `(2, 3)` or `2`.

`dtype : data-type, optional`

The desired data-type for the array, e.g., `numpy.int8`. Default is `numpy.float64`.

`order : {'C', 'F}, optional, default: 'C'`

Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

`like : array_like, optional`

Reference object to allow the creation of arrays which are not NumPy arrays. If an array-like passed in as `like` supports the `__array_function__` protocol, the result will be defined by it. In this case, it ensures the creation of an array object compatible with that passed in via this argument.

Πηγή <https://numpy.org/devdocs/reference/generated/numpy.zeros.html>

#### enumerate(iterable, start=0)

Return an enumerate object. `iterable` must be a sequence, an `iterator`, or some other object which supports iteration. The `__next__()` method of the iterator returned by `enumerate()` returns a tuple containing a count (from `start` which defaults to 0) and the values obtained from iterating over `iterable`.

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

Equivalent to:

```
def enumerate(iterable, start=0):
    n = start
    for elem in iterable:
        yield n, elem
        n += 1
```

Πηγή <https://docs.python.org/3/library/functions.html#enumerate>

## 4. Μοντέλο Νευρωνικού Δικτύου

Για το Νευρωνικό Δίκτυο μας θα χρειαστούμε την βιβλιοθήκη PyTorch και το υπο-πακέτο της torch.nn που περιέχει εργαλεία για τη δημιουργία νευρωνικών δικτύων.

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

Ορίζουμε την κλάση NeuralNet που κληρονομεί την κλάση nn.Module της PyTorch. Έπειτα ορίζουμε την συνάρτηση κατασκευής `__init__()` με ορίσματα το ίδιο το αντικείμενο της κλάσης (`self`), το μέγεθος της εισόδου, τον αριθμό των νευρώνων το κρυφό επίπεδο και τον αριθμό των κατηγοριών που θέλουμε να προβλέψουμε για την ταξινόμηση. Για να μπορέσουμε να χρησιμοποιήσουμε τις μεθόδους και τα χαρακτηριστικά της υπερ-κλάσης καλούμε την συνάρτηση `super()` για να καλέσουμε τον αρχικό κατασκευαστή της υπερκλάσης. Με αυτόν τον τρόπο διασφαλίζουμε την κληρονομικότητα των λειτουργιών της υπερκλάσης στην υποκλάση και κατ' επέκταση η ορθή ρύθμισή τους. Αντικειμενοποιούμε (`self.l1`) το πρώτο πλήρως συνδεδεμένο επίπεδο του ΝΔ της κλάσης nn.Linear της PyTorch, η οποία δημιουργεί ένα επίπεδο που εκτελεί έναν γραμμικό μετασχηματισμό:

$$y = xA^T + b.$$

Όπου `x` είναι ένα `input tensor` με μέγεθος `input_size`, `A` είναι τα βάρη στα οποία θα εκπαιδευτεί το μοντέλο κατά την εκπαίδευση και `b` είναι το διάνυσμα με της μεταβλητές `bias`. Έτσι, υπολογίζεται το `y`, το `output tensor` με μέγεθος `hidden_size`. Για αυτό και θέτουμε ως ορίσματα της κλάσης `nn.Linear()` το `input_size` και το `hidden_size`.

Έπειτα, αντικειμενοποιούμε `self.l2` το κρυφό επίπεδο του ΝΔ όπου ο γραμμικός μετασχηματισμός έχει `input` και `output tensors` μέγεθος `hidden_size`. Γι' αυτό και θέτουμε ορίσματα `hidden_size` και `hidden_size`, διατηρώντας τον ίδιο αριθμό χαρακτηριστικών. Τα δεδομένα επεξεργάζονται στο ενδιάμεσο επίπεδο πριν την έξοδο.

Δημιουργούμε το τελικό επίπεδο εξόδου του ΝΔ στο αντικείμενο `self.l3` της κλάσης `nn.Linear()`. Ως ορίσματα θέτουμε `hidden_size` και `num_classes`, έτσι ώστε το `input tensor` να έχει μέγεθος `hidden_size` και το `output_size` μέγεθος `num_classes`. Έτσι, η έξοδος είναι η πρόβλεψη των κατηγοριών.

Τέλος αρχικοποιούμε (`self.relu`) την συνάρτηση ενεργοποίησης Rectified Linear Unit (ReLU) που ορίζεται ως:

$$\text{ReLU}(x) = (x)^+ = \max(0, x)$$

Αυτό σημαίνει πως αν η έξοδος είναι  $x > 0$  τότε το  $x$  παραμένει ίδιο, ειδάλλως για  $x \leq 0$  τότε το  $x$  γίνεται 0. Προτιμάμε την ReLU καθώς το  $\max(0, z)$  δεν υπόκειται σε κορεσμό στο όριο ενός μεγάλου  $z$ , σε αντίθεση με τους σιγμοειδείς νευρώνες, και αυτό βοηθά τις ανορθωμένες γραμμικές μονάδες να συνεχίσουν να μαθαίνουν<sup>1</sup>. Επίσης, οι παράγωγοι άλλων συναρτήσεων μικραίνουν σε βαθιά νευρωνικά δίκτυα, ενώ η ReLU έχει σταθερή κλίση για γρήγορη σύγκλιση.

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
```

<sup>1</sup> Σ. Λυκοθανάσης, Δ. Κουτσομητρόπουλος, Υπολογιστική Νοημοσύνη και βαθιά μάθηση, Κάλλιπος, Πανεπιστήμιο Πατρών, 2021, σελ. 169.

```

super(NeuralNet, self).__init__()
self.l1 = nn.Linear(input_size, hidden_size)
self.l2 = nn.Linear(hidden_size, hidden_size)
self.l3 = nn.Linear(hidden_size, num_classes)
self.relu = nn.ReLU()

```

## Linear

CLASS `torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None)` [SOURCE]

Applies an affine linear transformation to the incoming data:  $y = xA^T + b$ .

This module supports `TensorFloat32`.

On certain ROCm devices, when using float16 inputs this module will use `different precision` for backward.

### Parameters

- `in_features` (`int`) – size of each input sample
- `out_features` (`int`) – size of each output sample
- `bias` (`bool`) – If set to `False`, the layer will not learn an additive bias. Default: `True`

Πηγή <https://pytorch.org/docs/stable/generated/torch.nn.Linear.html>

## ReLU

CLASS `torch.nn.ReLU(inplace=False)` [SOURCE]

Applies the rectified linear unit function element-wise.

$\text{ReLU}(x) = (x)^+ = \max(0, x)$

### Parameters

- `inplace` (`bool`) – can optionally do the operation in-place. Default: `False`

### Shape:

- Input: `(*)`, where `*` means any number of dimensions.
- Output: `(*)`, same shape as the input.

Πηγή <https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html>

Ορίζουμε την συνάρτηση της εμπρόσθιας διέλευσης `forward()` με ορίσματα το ίδιο το αντικείμενο της κλάσης `self` και το input tensor `x` που περνά μέσα από τα πεδία του ΝΔ για να παραχθεί το output tensor. Εισάγουμε το input tensor `x` στο πρώτο επίπεδο που εκτελεί τον μετασχηματισμό, μετατρέποντας το input tensor από `input_size` σε `hidden_size`. Έπειτα εφαρμόζουμε την ενεργοποίηση `ReLU` στην έξοδο `out` του πρώτου επιπέδου. Το `out` έπειτα περνάει στο δεύτερο κρυφό επίπεδο και εφαρμόζουμε πάλι `ReLU`. Τέλος, το `out` περνάει από τρίτο επίπεδο όπου και μετασχηματίζεται από `hidden_size` σε `num_classes`. Η `Softmax` είναι ενσωματωμένη στην Διασταυρούμενη Εντροπία και επομένως δεν εφαρμόζουμε μέσα στο δίκτυο, αλλά εξωτερικά. Η συνάρτηση μας επιστρέφει την έξοδο `out`, δηλαδή τα logits για την υπολογισμό της απώλειας και την διάκριση των κατηγοριών.

```

def forward(self, x):
    out = self.l1(x)
    out = self.relu(out)
    out = self.l2(out)
    out = self.relu(out)
    out = self.l3(out)
    return out

```

## 5. Εκπαίδευση μοντέλου

Για να φτιάξουμε τον κώδικα εκπαίδευσης του μοντέλου μας θα χρειαστούμε την Numerical Python βιβλιοθήκη, την SQLite3 βιβλιοθήκη της Python για την σύνδεση με την βάση δεδομένων, και την PyTorch βιβλιοθήκη για την εκπαίδευση του μοντέλου. Από τον κώδικα με τα εργαλεία για την επεξεργασία φυσικής γλώσσας εισάγουμε τις συναρτήσεις για το tokenization, stemming και bagging των λέξεων. Από τον κώδικα για την δημιουργία του μοντέλου εισάγουμε το νευρωνικό δίκτυο.

```
import numpy as np
import sqlite3
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from nltkutils import bag_of_words, tokenize, stem
from model import NeuralNet
```

Αντικειμενοποιούμε την σύνδεση με την βάση δεδομένων μας chatbot\_liberx.db και τον cursor για να μπορούμε να εκτελέσουμε δηλώσεις και ερωτήματα της SQL και να ανακτήσουμε πρόσβαση στα δεδομένα μας.

```
con = sqlite3.connect("chatbot_liberx.db")
cur = con.cursor()
```

First, we need to create a new database and open a database connection to allow `sqlite3` to work with it. Call `sqlite3.connect()` to create a connection to the database `tutorial.db` in the current working directory, implicitly creating it if it does not exist:

```
import sqlite3
con = sqlite3.connect("tutorial.db")
```

The returned `Connection` object `con` represents the connection to the on-disk database.

In order to execute SQL statements and fetch results from SQL queries, we will need to use a database cursor. Call `con.cursor()` to create the `Cursor`:

```
cur = con.cursor()
```

Πηγή <https://docs.python.org/3/library/sqlite3.html#sqlite3.connect>

Πλέον μπορούμε να ανακτήσουμε τα δεδομένα μας από την βάση δεδομένων SQL. Μέσω του cursor η εντολή `execute()` επιλέγει τα id και tag από τα tags και ανακτά την σειρά των αποτελεσμάτων με την εντολή στην μεταβλητή `tags_data`. Αντιστοίχως και για το κάθε tag\_id και το αντίστοιχο του pattern και response και τα patterns και responses. Αφού ανακτήσαμε τα δεδομένα μας, κλείνουμε την σύνδεση με την βάση δεδομένων με την εντολή `con.close()`.

```
# Ανάκτηση όλων των tags
cur.execute("SELECT id, tag FROM tags")
tags_data = cur.fetchall()
```

```

# Ανάκτηση όλων των patterns
cur.execute("SELECT tag_id, pattern FROM patterns")
patterns_data = cur.fetchall()

# Ανάκτηση όλων των responses
cur.execute("SELECT tag_id, response FROM responses")
responses_data = cur.fetchall()

# Κλείσιμο της σύνδεσης
con.close()

```

We can verify that the new table has been created by querying the `sqlite_master` table built-in to SQLite, which should now contain an entry for the `movie` table definition (see [The Schema Table](#) for details). Execute that query by calling `cur.execute(...)`, assign the result to `res`, and call `res.fetchone()` to fetch the resulting row:

```
>>> res = cur.execute("SELECT name FROM sqlite_master")
>>> res.fetchone()
('movie',)
```

`close()`

Close the database connection. If `autocommit` is `False`, any pending transaction is implicitly rolled back. If `autocommit` is `True` or `LEGACY_TRANSACTION_CONTROL`, no implicit transaction control is executed. Make sure to `commit()` before closing to avoid losing pending changes.

`execute(sql, parameters=(), /)`

Create a new `Cursor` object and call `execute()` on it with the given `sql` and `parameters`. Return the new cursor object.

Πηγή <https://docs.python.org/3/library/sqlite3.html#sqlite3.connect>

Μετά την ανάκτηση των δεδομένων προχωράμε στην επεξεργασία τους. Για την επεξεργασία των tags δημιουργούμε μία λίστα στην οποία αποθηκεύουμε την θέση `tag[1]` για κάθε tag στην `tags_data`. Δηλαδή, λίστα `tags_data` διαθέτει tuples με (`id, name`) όπου τα `ids` έχουν `index[0]` και τα `names` `index[1]` σε κάθε tuple. Με τον τρόπο αυτό φτιάχνουμε μία λίστα με μόνο τα tag names. Έπειτα, αρχικοποιούμε δύο κενές λίστες `all_words` και `xy`, για τις tokenized προτάσεις και τα tuples (words, tag). Για να αντιστοιχήσουμε τα tags με τα `patterns` δημιουργούμε έναν βρόχο όπου για κάθε `pattern` στα `patterns_data` που ανακτήσαμε το αναθέτει σε μία λίστα `sentence` και το αντίστοιχο `id` στην `tag_id`, κάνοντας έτσι unpacking τα tuples. Τα διακριτά πλέον `patterns` στην μεταβλητή `sentence` γίνονται tokenized και αποθηκεύονται στη λίστα `words`, η οποία προσθέτει μεμονωμένα πλέον (extend) τα στοιχεία της στην `all_words`. Αντιθέτως, αν χρησιμοποιούσαμε `append` η λίστα θα προστίθεντο ολόκληρη ως ένα στοιχείο. Στη συνέχεια, χρησιμοποιούμε την συνάρτηση `next()` της Python η οποία προσπελαύνει κατά σειρά τα στοιχεία όσο την καλούμε με κάθε επανάληψη μέχρι να εξαντληθούν τα δεδομένα. Θέτουμε τον όρο το `tag_id` να είναι ίδιο με το `tag_id_db`, δηλαδή ο συνδετικός κρίκος μεταξύ tags και patterns, για να αποθηκεύσει στη λίστα tag κάθε `tag_name` που βρίσκεται στην `tags_data` και το `tag_id_db` αντίστοιχεί με εκείνο το `tag_id` των `patterns`. Έχοντας διασυνδέσει πλέον τα στοιχεία αυτά της βάσης δεδομένων μεταξύ τους, προσθέτουμε σε tuples (words, tag) στην λίστα `xy` κάθε tokenized λέξη των `patterns` με το αντίστοιχο `tag_name` τους.

```

tags = [tag[1] for tag in tags_data]
all_words = []
xy = []

for pattern in patterns_data:
    tag_id, sentence = pattern

```

```
words = tokenize(sentence)
all_words.extend(words)
tag = next(tag_name for tag_id_db, tag_name in tags_data if tag_id_db ==
tag_id)
xy.append((words, tag))
```

```
next(iterator)
next(iterator, default)
```

Retrieve the next item from the `iterator` by calling its `__next__()` method. If `default` is given, it is returned if the iterator is exhausted, otherwise `StopIteration` is raised.

Πηγή <https://docs.python.org/3/library/functions.html#next>

```
list.append(x)
```

Add an item to the end of the list. Similar to `a[len(a):] = [x]`.

```
list.extend(iterable)
```

Extend the list by appending all the items from the iterable. Similar to `a[len(a):] = iterable`.

Πηγή <https://docs.python.org/3/tutorial/datastructures.html>

Το επόμενο βήμα μας είναι ο καθορισμός των λέξεων, καθώς στην λίστα μας βρίσκεται κάθε τι διακριτό μέρος του λόγου που στην προκειμένη περίπτωση δεν έχει μείζονα σημασία στην εκπαίδευση του μοντέλου. Επομένως, ορίζουμε μία λίστα με όλα εκείνα τα σημεία στίξης που θέλουμε να αφαιρέσουμε. Έπειτα, εφαρμόζουμε stemming στις λέξεις της `all_words` εκτός από τα σημεία στίξης που ορίσαμε και τα αποθηκεύουμε στην ίδια την λίστα `all_words` για να μην έχουμε τις `ignore_words`. Με την `set()` αφαιρούμε τα διπλότυπα και την `sorted()` ταξινομούμε τα στοιχεία στην λίστα.

```
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
all_words = sorted(set(all_words))
tags = sorted(set(tags))
```

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the `set()` function can be used to create sets. Note: to create an empty set you have to use `set()`, not `{}`; the latter creates an empty dictionary, a data structure that we discuss in the next section.

Πηγή <https://docs.python.org/3/tutorial/datastructures.html>

## Sorting Basics

A simple ascending sort is very easy: just call the `sorted()` function. It returns a new sorted list:

```
>>> sorted([5, 2, 3, 1, 4])  
[1, 2, 3, 4, 5]
```

>>>

You can also use the `list.sort()` method. It modifies the list in-place (and returns `None` to avoid confusion). Usually it's less convenient than `sorted()` - but if you don't need the original list, it's slightly more efficient.

```
>>> a = [5, 2, 3, 1, 4]  
>>> a.sort()  
>>> a  
[1, 2, 3, 4, 5]
```

>>>

Another difference is that the `list.sort()` method is only defined for lists. In contrast, the `sorted()` function accepts any iterable.

```
>>> sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})  
[1, 2, 3, 4, 5]
```

>>>

Πηγή <https://docs.python.org/3/howto/sorting.html>

Πλέον μπορούμε να προχωρήσουμε στην διαμόρφωση των δεδομένων εκπαίδευσης μετατρέποντας τα patterns σε bag of words διανύσματα. Δημιουργούμε δύο κενές λίστες `X_train` και `y_train` για τα διανύσματα των patterns και των αντίστοιχων tags τους. Έτσι, δημιουργούμε έναν βρόχο όπου για κάθε pattern\_sentence και tag στην λίστα `xy` δημιουργούμε στην μεταβλητή `bag` ένα διάνυσμα με την συνάρτηση `bag_of_words` του κώδικα με τα εργαλεία επεξεργασίας φυσικής γλώσσας. Με αυτόν τον τρόπον δημιουργούμε ένα δυαδικό διάνυσμα για κάθε πρόταση των patterns όπου για κάθε λέξη που υπάρχει στην `pattern_sentence` ανά πρόταση τότε δημιουργείτε ένα διάνυσμα με καθορισμένο πλήθος των `all_words` όπου το 1 αντιστοιχεί στην παρουσία της λέξης στην πρόταση και το 0 την απουσία τους. Τα διανύσματα αυτά μπορούν να χρησιμοποιηθούν ως είσοδοι στο νευρωνικό δίκτυο για να αναγνωρίζει τα μοτίβα τους. Στην κενή λίστα `X_train` προσθέτουμε τα διανύσματα. Τώρα, θα χρειστεί να μετατρέψουμε τα tags σε αριθμητική αναπαράσταση έτσι ώστε να μπορούν να χρησιμοποιηθούν από την Σταυροειδή Εντροπία της PyTorch. Δημιουργούμε μία λίστα με την θέση index κάθε tag στην λίστα tags και την προσθέτουμε στην κενή λίστα `y_train` που είχαμε ορίσει. Τέλος, για να μπορέσουμε να προχωρήσουμε με την εκπαίδευση του μοντέλου, μετατρέπουμε τις λίστες σε πίνακες array της Numerical Python έτσι ώστε να μπορούν να αποτελέσουν την είσοδο στις συναρτήσεις της PyTorch.

```
X_train = []  
y_train = []  
for (pattern_sentence, tag) in xy:  
    bag = bag_of_words(pattern_sentence, all_words)  
    X_train.append(bag)  
    label = tags.index(tag)  
    y_train.append(label)  
X_train = np.array(X_train)  
y_train = np.array(y_train)
```

## numpy.array

```
numpy.array(object, dtype=None, *, copy=True, order='K', subok=False,  
ndmin=0, like=None)
```

Create an array.

Πηγή <https://numpy.org/doc/2.1/reference/generated/numpy.array.html>

Πριν ξεκινήσουμε την εκπαίδευση του μοντέλου, πρέπει να ορίσουμε της υπερπαραμέτρους της εκμάθησης. Ορίζουμε ως αριθμό εποχών num\_epochs 1000 επαναλήψεις μάθησης, ως αριθμό των δειγμάτων που θα χρησιμοποιούνται ταυτόχρονα πριν την ενημέρωση των βαρών batch\_size 8, ως ρυθμό μάθησης learning\_rate 0,001, ως μέγεθος εισόδου ίσο με το μέγεθος του διανύσματος των γνωστών λέξεων στα patterns, ως αριθμό νευρώνων στο κρυφό επίπεδο hidden\_size 8 και ως μέγεθος εξόδου ίσο με το μέγεθος των tags. Ειδικότερα, ο αριθμός των εποχών καθορίζει πόσες φορές το μοντέλο θα εκπαιδευτεί πάνω στο σύνολο των δεδομένων για να βελτιώσει την ακρίβεια του, αλλά παράλληλα ο αριθμός πρέπει να είναι τέτοιος έτσι ώστε να αποφευχθεί η υπερπροσαρμογή. Ο αριθμός των δειγμάτων καθορίζει τα πόσα δείγματα θα διατρέξουν το νευρωνικό δίκτυο ανά βήμα, δηλαδή ανά εποχή, έτσι ώστε να βελτιωθεί η γενίκευση του μοντέλου. Ο ρυθμός εκμάθησης καθορίζει τον ρυθμό ενημέρωσης των παραμέτρων για κάθε εποχή εκμάθησης. Ο αριθμός των χαρακτηριστικών εισόδου στο τεχνητό νευρωνικό δίκτυο (αριθμός νευρώνων input) είναι ίσος με το διάνυσμα bag of words όπως και ο αριθμός των κατηγοριών εξόδου (αριθμός νευρώνων output) που δηλώνει και την κατηγορία της απάντησης του chatbot είναι ίσος με τον αριθμό των πιθανών tag. Ο αριθμός των μεσαίων νευρώνων του κρυφού επιπέδου του νευρώνα καθορίζεται ως 8, έτσι ώστε το δίκτυο να ισορροπεί ανάμεσα στο overfitting και την ελλιπή ικανότητα εκμάθησης.

```
num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
```

Εκτυπώνοντας το input\_size και output\_size (print(input\_size, output\_size)) μας επιστρέφει ως input\_size 201 και ως output\_size 155.

```
● konstantinosliontakis@Konstantinoss-MacBook-Air Desktop % python3 train.py
201 155
```

Ορίζουμε μία υπο-κλάση ChatDataset της Dataset, που κληρονομεί όλες τις λειτουργίες και μεθόδους της κλάσης Dataset της PyTorch. Η Dataset μας επιτρέπει την εύκολη προσπέλαση των δεδομένων που αναπαριστούν έναν χάρτη, όπου ένα κλειδί αντιστοιχίζεται σε ένα δείγμα δεδομένων. Ορίζουμε την συνάρτηση constructor \_\_init\_\_() με όρισμα το κάθε αντικείμενο της κλάσης (self). Η συνάρτηση αποθηκεύει τα δεδομένα εκπαίδευσης X\_train και y\_train και ορίζει τον αριθμό των δειγμάτων που είναι ίσος με τον αριθμό των διανυσμάτων X\_train. Έπειτα, ορίζουμε την indexing συνάρτηση \_\_getitem\_\_() με όρισμα το ίδιο το αντικείμενο self και το index, τον αριθμό μίας θέσης ενός δείγματος στο σύνολο των δεδομένων (dataset). Αυτή η μέθοδος ορίζει τον τρόπο με τον οποίο ανακτώνται τα δεδομένα όταν δίνεται ένας κλειδί. Η συνάρτηση επιστρέφει ένα tuple (x,y) για την εκπαίδευση, όπου το x είναι ένα διάνυσμα με index[i] και το y είναι ο αριθμός της ετικέτας tag με index [i]. Δημιουργούνται έτσι συγκεκριμένα batches με συγκεκριμένους indices για την εκπαίδευση και επιτρέπει την πρόσβαση στα δεδομένα σαν είναι μία λίστα με tuples (dataset[i]). Τέλος, ορίζουμε την sizing συνάρτηση \_\_len\_\_() με όρισμα το ίδιο το αντικείμενο της κλάσης (self) που θα μας επιστρέφει το συνολικό μέγεθος του συνόλου δεδομένων (dataset) για να μπορούμε να πάρουμε το πλήθος του εκτελώντας την εντολή len(dataset). Αυτή η μέθοδος είναι χρήσιμη για τις προεπιλεγμένες ρυθμίσεις του DataLoader της PyTorch. Αντικειμενοποιούμε την ChatDataset στο αντικείμενο dataset, έτσι ώστε να μπορούμε να το χρησιμοποιήσουμε δυναμικά κατά την εκπαίδευση για να αντλήσουμε τα δεδομένα μας.

```

class ChatDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        return self.n_samples

dataset = ChatDataset()

```

CLASS `torch.utils.data.Dataset` [SOURCE]

An abstract class representing a `Dataset`.

All datasets that represent a map from keys to data samples should subclass it. All subclasses should overwrite `__getitem__()`, supporting fetching a data sample for a given key. Subclasses could also optionally overwrite `__len__()`, which is expected to return the size of the dataset by many `Sampler` implementations and the default options of `DataLoader`. Subclasses could also optionally implement `__getitems__()`, for speedup batched samples loading. This method accepts list of indices of samples of batch and returns list of samples.

• NOTE

`DataLoader` by default constructs an index sampler that yields integral indices. To make it work with a map-style dataset with non-integral indices/keys, a custom sampler must be provided.

Πηγή <https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset>

Αντικειμενοποιύμε στο `train_loader` την κλάση `DataLoader` της PyTorch που φορτώνει τα δεδομένα συνδυάζοντας το `dataset` και έναν δειγματολήπτη `sampler` με την χρήση ενός `iterable`. Η κλάση δέχεται ως όρισμα το `dataset` που μόλις καθορίσαμε (`X_train`, `y_train`), τον αριθμό των `batches` που έχουμε ορίσει (8), την εντολή `shuffle=True` για να ανακατεύει τα δεδομένα πρις από κάθε εποχή έτσι ώστε να πετυχαίνουμε την γενίκευση και να αποφεύγουμε την αποστήθιση, και `num_workers` ίσο με ένα καθώς η φόρτωση θα γίνει στο κύριο νήμα, άρα μηδενικά `worker threads`.

```

train_loader = DataLoader(dataset=dataset,
                        batch_size=batch_size,
                        shuffle=True,
                        num_workers=0)

```

```
CLASS torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=None, sampler=None,
batch_sampler=None, num_workers=0, collate_fn=None, pin_memory=False,
drop_last=False, timeout=0, worker_init_fn=None, multiprocessing_context=None,
generator=None, *, prefetch_factor=None, persistent_workers=False,
pin_memory_device='', in_order=True) [SOURCE]
```

Data loader combines a dataset and a sampler, and provides an iterable over the given dataset.

The `DataLoader` supports both map-style and iterable-style datasets with single- or multi-process loading, customizing loading order and optional automatic batching (collation) and memory pinning.

See [torch.utils.data](#) documentation page for more details.

#### Parameters

- `dataset` (`Dataset`) – dataset from which to load the data.
- `batch_size` (`int, optional`) – how many samples per batch to load (default: 1).
- `shuffle` (`bool, optional`) – set to `True` to have the data reshuffled at every epoch (default: `False`).
- `sampler` (`Sampler or Iterable, optional`) – defines the strategy to draw samples from the dataset. Can be any `Iterable` with `__len__` implemented. If specified, `shuffle` must not be specified.
- `batch_sampler` (`Sampler or Iterable, optional`) – like `sampler`, but returns a batch of indices at a time. Mutually exclusive with `batch_size`, `shuffle`, `sampler`, and `drop_last`.
- `num_workers` (`int, optional`) – how many subprocesses to use for data loading. 0 means that the data will be loaded in the main process. (default: 0)

Πηγή <https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader>

Για να καθορίζουμε την συσκευή device στην οποία θα εκτελείται το μοντέλο στην PyTorch, χρησιμοποιούμε την μέθοδο `torch.device()`. Η συσκευή θα ρυθμιστεί σε ‘cuda’ αν η εντολή `torch.cuda.is_available` επιστρέψει TRUE, ειδάλλως θα ρυθμιστεί σε ‘cpu’. Συνελόντι ειπεύν, με αυτόν τον τρόπο ρυθμίζουμε το PyTorch να χρησιμοποιήσει την Graphic Processing Unit (Μονάδα Επεξεργασίας Γραφικών) αν είναι διαθέσιμη, ειδάλλως να χρησιμοποιήσει την Central Processing Unit (Κεντρική Μονάδα Επεξεργασίας).

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

`torch.device`

```
CLASS torch.device
```

A `torch.device` is an object representing the device on which a `torch.Tensor` is or will be allocated.

The `torch.device` contains a device type (most commonly “cpu” or “cuda”, but also potentially “mps”, “xpu”, “xla” or “meta”) and optional device ordinal for the device type. If the device ordinal is not present, this object will always represent the current device for the device type, even after `torch.cuda.set_device()` is called; e.g., a `torch.Tensor` constructed with device ‘cuda’ is equivalent to ‘cuda:X’ where X is the result of `torch.cuda.current_device()`.

A `torch.Tensor`’s device can be accessed via the `Tensor.device` property.

A `torch.device` can be constructed via a string or via a string and device ordinal

Πηγή [https://pytorch.org/docs/stable/tensor\\_attributes.html#torch.device](https://pytorch.org/docs/stable/tensor_attributes.html#torch.device)

Πλέον μπορούμε να δημιουργήσουμε το μοντέλο μας. Αντικειμενοποιούμε στο `model` την κλάση `NeuralNet()` από τον κώδικα του νευρωνικού δικτύου μας, με ορίσματα `input_size` για τον αριθμό των εισόδων στο δίκτυο (`X_train`), `hidden_size` για τον αριθμό των νευρώνων στο κρυφό επίπεδο (8) και `output_size` για τον αριθμό των εξόδων-προβλέψεων (`tags`). Μεταφέρουμε το μοντέλο την συσκευή που έχουμε ορίσει.

```
model = NeuralNet(input_size, hidden_size, output_size).to(device)
```

Ακολούθως, πρέπει να ορίσουμε την συνάρτηση απώλειας για να μετρήσουμε την απόσταση που υπάρχει ανάμεσα στην πραγματική (labels) και την προβλεπόμενη κατανομή (outputs) με στόχο να φέρουμε όσο το δυνατόν πιο κοντά την προβλεπόμενη στην πραγματική κατανομή. Το μοντέλο μας δέχεται ως είσοδο προτάσεις σε batches των 8 και υπολογίζει σε

logits πόσο πιθανό είναι η κάθε πρόταση να ανήκει σε μία κατηγορία (labeled tags). Η διασταυρούμενη εντροπία μετατρέπει αυτόματα τους logits σε πιθανότητες μέσω της Softmax και παίρνει για κάθε πρόταση  $-\log(\text{πιθανότητα})$  και βγάζει το μέσο όρο του batch. Ειδικότερα, η Διασταυρούμενη Εντροπία είναι ορισμένα by default με reduction στο mean, μέσο όρο. Έτσι, ο σταθμισμένος μέσος όρος απώλειας υπολογίζεται από τον τύπο:

$$\ell(x, y) = \begin{cases} \sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{y_n} \cdot 1\{y_n \neq \text{ignore\_index}\}} l_n, & \text{if reduction} = \text{'mean'}; \\ \sum_{n=1}^N l_n, & \text{if reduction} = \text{'sum'}. \end{cases}$$

Πηγή <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss>

Επομένως, για reduction = ‘mean’ ο σταθμισμένος μέσος όρος υπολογίζεται με αριθμητή το άθροισμα των επιμέρους απώλειών  $l_n$  και παρονομαστεί το άθροισμα των επιμέρους βαρών και ignore\_index αν χρησιμοποιηθεί για την παράβλεψη ορισμένων κατηγοριών κατά τον υπολογισμό της απώλειας. Η απώλεια  $l_n$  για ένα δείγμα ή υπολογίζεται ως:

$$l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\}$$

Πηγή <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss>

Άρα, η απώλεια είναι ο αρνητικός λογάριθμός της πιθανότητας που αποδίδει η Softmax για μία κλάση  $y_n$  πολλαπλασιασμένη με το βάρος της κλάσης αν υφίσταται. Για να χρησιμοποιήσουμε επομένως την συνάρτηση απώλειας της διασταυρούμενης εντροπίας, την αντικειμενοποιούμε στο criterion και έπειτα θα χρειαστεί να χρησιμοποιήσουμε έναν αλγόριθμο για την αναπροσαρμογή των βαρών κατά την διάρκεια της εκπαίδευσης. Αντικειμενοποιούμε τον Adam Optimizer με παραμέτρους όλα τα βάρη και τις μεταβλητές του μοντέλου για να τα ενημερώσει και ρυθμό εκμάθησης 0,001. Ο αλγόριθμος Adam Optimizer είναι ως εξής:

---

```

input :  $\gamma$  (lr),  $\beta_1, \beta_2$  (betas),  $\theta_0$  (params),  $f(\theta)$  (objective)
         $\lambda$  (weight decay),  $amsgrad$ ,  $maximize$ ,  $\epsilon$  (epsilon)
initialize :  $m_0 \leftarrow 0$  ( first moment),  $v_0 \leftarrow 0$  (second moment),  $\bar{v}_0^{max} \leftarrow 0$ 

for  $t = 1$  to ... do
    if  $maximize$  :
         $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$ 
    else
         $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ 
    if  $\lambda \neq 0$ 
         $g_t \leftarrow g_t + \lambda \theta_{t-1}$ 
     $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 
     $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
     $\bar{m}_t \leftarrow m_t / (1 - \beta_1^t)$ 
     $\bar{v}_t \leftarrow v_t / (1 - \beta_2^t)$ 
    if  $amsgrad$ 
         $\bar{v}_t^{max} \leftarrow \max(\bar{v}_{t-1}^{max}, \bar{v}_t)$ 
         $\theta_t \leftarrow \theta_{t-1} - \gamma \bar{m}_t / (\sqrt{\bar{v}_t^{max}} + \epsilon)$ 
    else
         $\theta_t \leftarrow \theta_{t-1} - \gamma \bar{m}_t / (\sqrt{\bar{v}_t} + \epsilon)$ 
return  $\theta_t$ 

```

---

Πηγή <https://pytorch.org/docs/stable/generated/torch.optim.Adam.html>

Ο βρόχος for  $t = 1$  to ... do καθορίζει τις επαναλήψεις της διαδικασίας εκπαίδευσης, εν προκειμένωι 1000 εποχές (to  $t = 1000$ ). Η  $g_t$  είναι η κλίση της συνάρτηση κόστους (gradient). Για  $maximize = \text{TRUE}$  πηγαίνουμε με κατεύθυνση προς την μεγιστοποίηση της απόδοσης,

όπου η κλίση υπολογίζεται με την αρνητική της τιμή. Ειδάλλως, με `maximize = FALSE` τότε η κλίση υπολογίζεται κανονικά και με κατεύθυνση μείωσης του κόστους. Για λ διάφορο του μηδενός τότε στην παράγωγη προστίθεται ένας όρος  $\lambda \theta_{t-1}$  με στόχο να προωθεί τις παραμέτρους θ προς το 0. Έπειτα υπολογίζονται τα εκθετικά κινητά μέσα της πρώτης και δεύτερης ροπής ( $m_t$  και  $u_t$ ), με την πρώτη ροπή να προσεγγίζει την μέση τιμή της παραγώγου και την δεύτερη την διακύμανση. Ακολούθως γίνεται η διόρθωση της μεροληψίας (bias). Με την `amsgrad` τροποποίηση κρατάμε το μέγιστο, ειδάλλως εφαρμόζουμε τον κανονικό κανόνα του Adam Optimizer.

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

## CrossEntropyLoss

CLASS `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean', label_smoothing=0.0)` [SOURCE]

This criterion computes the cross entropy loss between input logits and target.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D `Tensor` assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

The `input` is expected to contain the unnormalized logits for each class (which do not need to be positive or sum to 1, in general). `input` has to be a `Tensor` of size ( $C$ ) for unbatched input, ( $minibatch, C$ ) or ( $minibatch, C, d_1, d_2, \dots, d_K$ ) with  $K \geq 1$  for the  $K$ -dimensional case. The last being useful for higher dimension inputs, such as computing cross entropy loss per-pixel for 2D images.

Πηγή <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss>

## Adam

CLASS `torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False, *, foreach=None, maximize=False, capturable=False, differentiable=False, fused=None)` [SOURCE]

### Parameters

- `params` (`iterable`) – iterable of parameters or `named_parameters` to optimize or iterable of dicts defining parameter groups. When using `named_parameters`, all parameters in all groups should be named
- `lr` (`float, Tensor, optional`) – learning rate (default: 1e-3). A tensor LR is not yet supported for all our implementations. Please use a float LR if you are not also specifying `fused=True` or `capturable=True`.

Πηγή <https://pytorch.org/docs/stable/generated/torch.optim.Adam.html>

Τώρα προχωράμε στην διαμόρφωση του βρόχου εκπαίδευσης με στόχο την μείωση της απώλειας μέσω του optimizer. Με έναν βρόχο `for in` για κάθε εποχή `in range` των σύνολο των εποχών 1000 (από 0 έως 999), και για κάθε words (δεδομένα εισόδου), labels (δεδομένα για κατηγοροποίηση) στον `train_loader` (με εμφαλευμένο βρόχο) μεταφέρουμε τα δεδομένα στην συσκευή που έχουμε ορίσει και μετατρέπουμε τα labels σε ακέραιους τύπου Long Tensor για να μπορούν να χρησιμοποιηθούν κατά την ταξινόμηση (class indices). Προωθούμε τα δεδομένα εισόδου (forward pass) στο μοντέλο και επιστρέφει αποτελέσματα outputs για κάθε batch. Στην Διασταυρούμενη Εντροπία θέτουμε ως ορίσματα τα outputs και τα labels έτσι ώστε να υπολογίσει την απώλεια μεταξύ της πρόβλεψης και της πραγματικής ετικέτας και να την επιστρέψει στην loss. Έπειτα, προχωράμε σε μία αναδρομική διάδοση του σφάλματος (backpropagation) έτσι ώστε να υπολογίσουμε τις παραγώγους της απώλειας ως προς τα βάρη και να τα ενημερώσουμε βελτιστοποιώντας τα για την μείωση της απώλειας. Έτσι, μηδενίζουμε τα gradient του βελτιστοποιητή (`optimizer.zero_grad()`), υπολογίζουμε τα διανύσματα των παραγώγων για τις παραμέτρους του μοντέλου σε σχέση με την απώλεια (`loss.backward()`) και με τον Adam Optimizer βελτιώνουμε τα βάρη του μοντέλου βάσει των διανυσμάτων των παραγώγων. Επομένως, όσο εκτελείται η εκμάθηση του μοντέλου για num

εποχές σε κάθε batch υπολογίζεται η απώλεια και με την διαδικασία της αναδρομής βελτιώνουμε τα βάρη του μοντέλου για μείωση της απώλειας. Τέλος, ορίζουμε με έναν βρόχο για κάθε εκατό εποχές (99+1) να εκτυπώνεται η απώλεια και στο τέλος των εποχών αφού έχει προσπελάσει ο βρόχος όλα τα num\_epochs να εκτυπώνεται η τελική απώλεια με ακρίβεια τεσσάρων δεκαδικών ψηφίων.

```
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        outputs = model(words)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

print(f'final loss: {loss.item():.4f}'')
```

A `torch.dtype` is an object that represents the data type of a `torch.Tensor`. PyTorch has twelve different data types:

64-bit integer (signed)                    `torch.int64` or `torch.long`                    `torch.*.LongTensor`

*Πηγή [https://pytorch.org/docs/stable/tensor\\_attributes.html#torch.dtype](https://pytorch.org/docs/stable/tensor_attributes.html#torch.dtype)*

## torch.optim.Optimizer.zero\_grad

`Optimizer.zero_grad(set_to_none=True)`[\[source\]](#)[\[source\]](#)

Reset the gradients of all optimized `torch.Tensor`s.

Parameters

`set_to_none` (`bool`) – instead of setting to zero, set the grads to None. This will in general have lower memory footprint, and can modestly improve performance. However, it changes certain behaviors. For example: 1. When the user tries to access a gradient and perform manual ops on it, a None attribute or a Tensor full of 0s will behave differently. 2. If the user requests `zero_grad(set_to_none=True)` followed by a backward pass, `.grad`s are guaranteed to be None for params that did not receive a gradient. 3. `torch.optim` optimizers have a different behavior if the gradient is 0 or None (in one case it does the step with a gradient of 0 and in the other it skips the step altogether).

*Πηγή [https://pytorch.org/docs/stable/generated/torch.optim.Optimizer.zero\\_grad.html](https://pytorch.org/docs/stable/generated/torch.optim.Optimizer.zero_grad.html)*

## torch.Tensor.backward

`Tensor.backward(gradient=None, retain_graph=None, create_graph=False, inputs=None)`[\[source\]](#)[\[source\]](#)

Computes the gradient of current tensor wrt graph leaves.

The graph is differentiated using the chain rule. If the tensor is non-scalar (i.e. its data has more than one element) and requires gradient, the function additionally requires specifying a `gradient`. It should be a tensor of matching type and shape, that represents the gradient of the differentiated function w.r.t. `self`.

This function accumulates gradients in the leaves - you might need to zero `.grad` attributes or set them to `None` before calling it. See `Default gradient layouts` for details on the memory layout of accumulated gradients.

*Πηγή <https://pytorch.org/docs/stable/generated/torch.Tensor.backward.html>*

## torch.optim.Optimizer.step

Optimizer.step(*closure: None* = *None*) → *None[source][source]*

Optimizer.step(*closure: Callable[[], float]*) → *float*

Perform a single optimization step to update parameter.

Parameters

**closure** (*Callable*) – A closure that reevaluates the model and returns the loss. Optional for most optimizers.

Πηγή <https://pytorch.org/docs/stable/generated/torch.optim.Optimizer.step.html#torch.optim.Optimizer.step>

Αφού ολοκληρώσαμε την εκπαίδευση του μοντέλου, πρέπει να αποθηκεύσουμε το εκπαιδευμένο μοντέλο και τις παραμέτρους του σε ένα αρχεία. Έτσι, δημιουργούμε ένα λεξικό όπου στα κλειδιά `input_size`, `hidden_size`, `output_size`, `all_words`, `tags` που αντιστοιχούν στις παραμέτρους του μοντέλου και το κλειδί `model_state` που αποθηκεύει την μέθοδο `model.state_dict()`, δηλαδή την επαναφόρτωση όλης της κατάστασης του εκπαιδευμένου μοντέλου με τα δεδομένα και τις παραμέτρους του. Ορίζουμε το όνομα του αρχείου και αποθηκεύουμε το λεξικό στο αρχείο.

```
data = {
    "model_state": model.state_dict(),
    "input_size": input_size,
    "hidden_size": hidden_size,
    "output_size": output_size,
    "all_words": all_words,
    "tags": tags
}

FILE = "data.pth"
torch.save(data, FILE)
```

## What is a state\_dict in PyTorch

Created On: Apr 17, 2020 | Last Updated: Feb 06, 2024 | Last Verified: Nov 05, 2024

In PyTorch, the learnable parameters (i.e. weights and biases) of a `torch.nn.Module` model are contained in the model's parameters (accessed with `model.parameters()`). A `state_dict` is simply a Python dictionary object that maps each layer to its parameter tensor.

### Introduction

A `state_dict` is an integral entity if you are interested in saving or loading models from PyTorch. Because `state_dict` objects are Python dictionaries, they can be easily saved, updated, altered, and restored, adding a great deal of modularity to PyTorch models and optimizers. Note that only layers with learnable parameters (convolutional layers, linear layers, etc.) and registered buffers (batchnorm's running\_mean) have entries in the model's `state_dict`. Optimizer objects (`torch.optim`) also have a `state_dict`, which contains information about the optimizer's state, as well as the hyperparameters used. In this recipe, we will see how `state_dict` is used with a simple model.

Πηγή [https://pytorch.org/tutorials/recipes/recipes/what\\_is\\_state\\_dict.html](https://pytorch.org/tutorials/recipes/recipes/what_is_state_dict.html)

## 6. Δημιουργία Chatbot

Για την δημιουργία chatbot μας θα χρειαστεί να κατεβάσουμε την βιβλιοθήκη `random` για την επιλογή τυχαίων απαντήσεων από την βάση δεδομένων, την βιβλιοθήκη `sqlite3` για την σύνδεση με την βάση δεδομένων, και την PyTorch για την φόρτωση και εκτέλεση του Νευρωνικού Δικτύου. Από τον κώδικα για το μοντέλο μας εισάγουμε το Νευρωνικό Δίκτυο

και από τα εργαλεία επεξεργασίας φυσικής γλώσσας εισάγουμε τις συναρτήσεις bag\_of\_words και tokenize.

```
import random
import sqlite3
import torch
from model import NeuralNet
from nltkutils import bag_of_words, tokenize
```

Ορίζουμε την συσκευή που θα φορτώσει το μοντέλο μας, έτσι ώστε αν υπάρχει κάρτα γραφικών που υποστηρίζει cuda να επιλεγεί, ειδάλλως να επιλεγεί ο επεξεργαστείς gpu.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Ανοίγουμε μία σύνδεση στην βάση δεδομένων μας chatbot\_liberx.db και έναν cursor για την εκτέλεση ερωτημάτων. Φορτώνουμε το αποθηκευμένο αρχείο του μοντέλου μας data.pth επιστρέφοντας μας το λεξικό με τις παραμέτρους του. Έπειτα αναθέτουμε τις τιμές των κλειδιών του λεξικού σε μεταβλητές έτσι ώστε να μπορούμε να χρησιμοποιήσουμε τις παραμέτρους.

```
con = sqlite3.connect("chatbot_liberx.db")
cur = con.cursor()

FILE = "data.pth"
data = torch.load(FILE)

input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]
all_words = data['all_words']
tags = data['tags']
model_state = data["model_state"]
```

## torch.load

```
torch.load(f, map_location=None, pickle_module=pickle, *, weights_only=True, mmap=None,
           **pickle_load_args) [SOURCE]
```

Loads an object saved with `torch.save()` from a file.

`torch.load()` uses Python's unpickling facilities but treats storages, which underlie tensors, specially. They are first deserialized on the CPU and are then moved to the device they were saved from. If this fails (e.g. because the run time system doesn't have certain devices), an exception is raised. However, storages can be dynamically remapped to an alternative set of devices using the `map_location` argument.

If `map_location` is a callable, it will be called once for each serialized storage with two arguments: storage and location. The storage argument will be the initial deserialization of the storage, residing on the CPU. Each serialized storage has a location tag associated with it which identifies the device it was saved from, and this tag is the second argument passed to `map_location`. The builtin location tags are `'cpu'` for CPU tensors and `'cuda:device_id'` (e.g. `'cuda:2'`) for CUDA tensors. `map_location` should return either `None` or a storage. If `map_location` returns a storage, it will be used as the final deserialized object, already moved to the right device. Otherwise, `torch.load()` will fall back to the default behavior, as if `map_location` wasn't specified.

If `map_location` is a `torch.device` object or a string containing a device tag, it indicates the location where all tensors should be loaded.

Otherwise, if `map_location` is a dict, it will be used to remap location tags appearing in the file (keys), to ones that specify where to put the storages (values).

User extensions can register their own location tags and tagging and deserialization methods using `torch.serialization.register_package()`.

### Parameters

- `f (Union[str, PathLike, BinaryIO, IO[bytes]])` – a file-like object (has to implement `read()`, `readline()`, `tell()`, and `seek()`), or a string or `os.PathLike` object containing a file name

Πηγή <https://pytorch.org/docs/stable/generated/torch.load.html>

Αρχικοποιούμε σε ένα αντικείμενο την κλάση του ΝΔ μας με ορίσματα `input_size`, `hidden_size` και `output_size` και μεταφέρουμε το μοντέλο στην κατάλληλη συσκευή υπολογισμού. Για να αποκτήσει το μοντέλο τις γνώσεις που διαθέτει από την εκπαίδευση φορτώνουμε σε αυτό την κατάσταση του `model_state` με τις τιμές των βαρών και των παραμέτρων του. Με την μέθοδο `.eval()` θέτουμε το μοντέλο σε κατάσταση `evaluation`, δηλαδή τα layers που έχουν διαφορά ανάμεσα στην εκπαίδευση και την αξιολόγηση τίθενται σε `evaluation mode`.

```
model = NeuralNet(input_size, hidden_size, output_size).to(device)
model.load_state_dict(model_state)
model.eval()
```

Evaluation Mode (`nn.Module.eval()`)

Evaluation mode is not a mechanism to locally disable gradient computation. It is included here anyway because it is sometimes confused to be such a mechanism.

Functionally, `module.eval()` (or equivalently `module.train(False)`) are completely orthogonal to no-grad mode and inference mode. How `model.eval()` affects your model depends entirely on the specific modules used in your model and whether they define any training-mode specific behavior.

You are responsible for calling `model.eval()` and `model.train()` if your model relies on modules such as `torch.nn.Dropout` and `torch.nn.BatchNorm2d` that may behave differently depending on training mode, for example, to avoid updating your BatchNorm running statistics on validation data.

It is recommended that you always use `model.train()` when training and `model.eval()` when evaluating your model (validation/testing) even if you aren't sure your model has training-mode specific behavior, because a module you are using might be updated to behave differently in training and eval modes.

Πηγή <https://pytorch.org/docs/stable/notes/autograd.html#locally-disable-grad-doc>

## Saving and Loading Models

Created On: Aug 29, 2018 | Last Updated: Sep 10, 2024 | Last Verified: Nov 05, 2024

Author: Matthew Inkawich

This document provides solutions to a variety of use cases regarding the saving and loading of PyTorch models. Feel free to read the whole document, or just skip to the code you need for a desired use case.

When it comes to saving and loading models, there are three core functions to be familiar with:

1. `torch.save`: Saves a serialized object to disk. This function uses Python's `pickle` utility for serialization. Models, tensors, and dictionaries of all kinds of objects can be saved using this function.
2. `torch.load`: Uses `pickle`'s unpickling facilities to deserialize pickled object files to memory. This function also facilitates the device to load the data into (see [Saving & Loading Model Across Devices](#)).
3. `torch.nn.Module.load_state_dict`: Loads a model's parameter dictionary using a deserialized `state_dict`. For more information on `state_dict`, see [What is a state\\_dict?](#).

Πηγή [https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html](https://pytorch.org/tutorials/beginner/saving_loading_models.html)

Τώρα ήρθε η ώρα να το περιβάλλον ερωταπάντησης του chatbot μας. Για αρχή ορίζουμε σε μία μεταβλητή το όνομα του παραλήπτη και εκείνου που δίνει τις απαντήσεις, εν προκειμένωι του Ισιδώρου της Σεβίλλης. Έπειτα, ορίζουμε μία συνάρτηση για την διαδικασία επιλογής της απάντησης, την οποία θα χρησιμοποιήσουμε αργότερα στον κώδικα για το γραφικό περιβάλλον του chatbot. Επομένως, ορίζουμε μία συνάρτηση `get_response()` με όρισμα το μήνυμα του χρήστη `msg`. Στην μεταβλητή `sentence` αποθηκεύουμε την tokenized πρόταση του χρήστη που μας επιστρέφει η συνάρτηση από τον κώδικα επεξεργασίας φυσικής γλώσσας. Στην μεταβλητή `X` αποθηκεύουμε το διάνυσμα των tokenized words που υπάρχουν στην λίστα `all_words` που επιστρέφει η συνάρτηση `bag_of_words`. Στην συνέχεια, θα χρειαστεί να αναδιαμορφώσουμε τον διανυσματικό πίνακα της Numerical Python έτσι ώστε να είναι συμβατός με το μοντέλο της PyTorch. Δηλαδή, στην παρόύσα στιγμή το `X` έχει έναν διανυσματικό πίνακα ίσο με `len(all_words)` και έτσι το `X.shape=(L,)` όπου `L=len(all_words)`. Με την εντολή `reshape` αναδιαμορφώνουμε τις διαστάσεις του πίνακα σε `X.shape=(1, L)`, έτσι ώστε να είναι συμβατό με το σχήμα εισόδου της PyTorch (`batch_size, input_size`). Τώρα μπορούμε να μετατρέψουμε τον διανυσματικό πίνακα της NumPy σε 2D Tensor της PyTorch. Χρησιμοποιούμε την εντολή `torch.from_numpy()` με όρισμα το ίδιο το `X` και επιστρέφει το αποτέλεσμα στο ίδιο το `X` έτσι ώστε να μοιράζονται τα ίδια δεδομένα και μεταφέρουμε το αποτέλεσμα στην συμβατή συσκευή.

```
def get_response(msg):  
    sentence = tokenize(msg)  
    X = bag_of_words(sentence, all_words)  
    X = X.reshape(1, X.shape[0])  
    X = torch.from_numpy(X).to(device)
```

## numpy.reshape

```
numpy.reshape(a, /, shape=None, order='C', *, newshape=None, copy=None)
    Gives a new shape to an array without changing its data.
```

[\[source\]](#)

Parameters:

a : array\_like

Array to be reshaped.

shape : int or tuple of ints

The new shape should be compatible with the original shape. If an integer, then the result will be a 1-D array of that length. One shape dimension can be -1. In this case, the value is inferred from the length of the array and remaining dimensions.

order : {‘C’, ‘F’, ‘A’}, optional

Read the elements of a using this index order, and place the elements into the reshaped array using this index order. ‘C’ means to read / write the elements using C-like index order, with the last axis index changing fastest, back to the first axis index changing slowest. ‘F’ means to read / write the elements using Fortran-like index order, with the first index changing fastest, and the last index changing slowest. Note that the ‘C’ and ‘F’ options take no account of the memory layout of the underlying array, and only refer to the order of indexing. ‘A’ means to read / write the elements in Fortran-like index order if a is Fortran contiguous in memory, C-like order otherwise.

Πηγή <https://numpy.org/doc/stable/reference/generated/numpy.reshape.html>

## torch.from\_numpy

```
torch.from_numpy(ndarray) → Tensor
```

Creates a Tensor from a numpy.ndarray.

The returned tensor and ndarray share the same memory. Modifications to the tensor will be reflected in the ndarray and vice versa. The returned tensor is not resizable.

It currently accepts ndarray with dtypes of numpy.float64, numpy.float32, numpy.float16, numpy.complex64, numpy.complex128, numpy.int64, numpy.int32, numpy.int16, numpy.int8, numpy.uint8, and bool.

• WARNING

Writing to a tensor created from a read-only NumPy array is not supported and will result in undefined behavior.

Πηγή [https://pytorch.org/docs/stable/generated/torch.from\\_numpy.html](https://pytorch.org/docs/stable/generated/torch.from_numpy.html)

Τώρα μπορούμε να περάσουμε τα δεδομένα στο μοντέλο μας για να βγάλει μία πρόβλεψη output. Προωθούμε (forward pass) το X στον νευρωνικό δίκτυο με μέγεθος (1, L) και μας επιστρέφεται ως output ένα tensor με μέγεθος (1, T) όπου T=ο αριθμός των tags και σε κάθε tag επιστρέφει τα logits. Ακολούθως, το μοντέλο πρέπει να βρει την σωστή πρόβλεψη με την μεγαλύτερη ακατέργαστη τιμή και τον αντίστοιχο δείκτη της κατηγορίας για να συνδέσει την προβλεπόμενη απάντηση. Έτσι, χρησιμοποιούμε την συνάρτηση torch.max() με ορίσματα το output tensor και το μήκος της διάστασης κατά το οποίο θα χρειαστεί να βρεθεί η μέγιστη τιμή. Η συνάρτηση μας επιστρέφει ένα tuple με την μέγιστη τιμή και τον index της. Χρησιμοποιούμε την σύμβαση της underscore (\_) για να δηλώσουμε πως την μέγιστη τιμή δεν θα την χρησιμοποιήσουμε, παρά μόνο την θέση της για την πρόβλεψη της απάντησης. Πλέον για να ανακτήσουμε την προβλεπόμενη κατηγορία tag της απάντησης, χρησιμοποιούμε την μέθοδος predicted.item(). Με τον τρόπο αυτό μετατρέπουμε τον tensor([i]) σε ακέραιο αριθμό tags[predicted.item(i)].

```
output = model(X)
_, predicted = torch.max(output, dim=1)
tag = tags[predicted.item()]
```

```
torch.max(input, dim, keepdim=False, *, out=None)
```

Returns a namedtuple (values, indices) where values is the maximum value of each row of the input tensor in the given dimension dim. And indices is the index location of each maximum value found (argmax).

Πηγή <https://pytorch.org/docs/stable/generated/torch.max.html>

Με την βοήθεια της μαθηματικής συνάρτησης softmax μετατρέπουμε τα logits σε πιθανότητες. Με ορίσματα output και dim=1 υπολογίζει τις πιθανότητες για κάθε ακατέργαστη τιμή των κατηγοριών στην γραμμή του output. Στην ουσία η softmax υπολογίζει την πιθανότητα υπολογίζοντας την εκθετική τιμή της εξόδου του NΔ για κάθε κατηγορία προς το άθροισμα των εκθετικών τιμών όλων των κατηγοριών της εξόδου. Έπειτα εξάγουμε την πιθανότητα της υψηλότερης τιμής για να την βάλουμε έπειτα σε μία εντολή if παραμέτρου υψηλής πιθανότητας. Τώρα, μπορούμε να ανακτήσουμε με μία ερώτηση SQL τις απαντήσεις από την βάση δεδομένων μας chatbot\_liberX.db με βάση την προβλεπόμενη κατηγορία. Ο cursor εκτελεί μία ερώτηση όπου επιλέγει την στήλη response (SELECT response) από τον πίνακα responses (FROM responses) και συνδέει τον πίνακα responses με τον πίνακα tags μέσω του tag\_id (INNER JOIN tags ON responses.tag\_id=tags.id) και τέλος επιλέγεις τις απαντήσεις που αντιστοιχούν στο προβλεπόμενο tag (WHERE tags.tag=?). Ο cursor ανακτά όλες τις απαντήσεις που βρέθηκαν από το SQL ερώτημα.

```
probs = torch.softmax(output, dim=1)
prob = probs[0][predicted.item()]
cur.execute("SELECT response FROM responses INNER JOIN tags ON responses.tag_id = tags.id WHERE tags.tag = ?", (tag,))
responses = cur.fetchall()
```

## Softmax

CLASS `torch.nn.Softmax(dim=None)` [\[SOURCE\]](#)

Applies the Softmax function to an n-dimensional input Tensor.

Rescales them so that the elements of the n-dimensional output Tensor lie in the range [0,1] and sum to 1.

Softmax is defined as:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

When the input Tensor is a sparse tensor then the unspecified values are treated as `-inf`.

Shape:

- Input: (\*) where \* means, any number of additional dimensions
- Output: (\*), same shape as the input

Πηγή <https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html>

Τέλος, ορίζουμε μία συνθήκη ελέγχου και βεβαιώτητας if που ελέγχει αν η πιθανότητα είναι πάνω από 75% και αν υπάρχει απάντηση στη βάση δεδομένων. Τα responses επιστρέφουν σε μορφή tuples (response,) και έτσι σε κάθε tuple r των responses παίρνουμε το πρώτο στοιχείο r[0] και δημιουργούμε μία λίστα με κάθε response. Όταν καλύπτονται οι παράμετροι της if τότε επιλέγεται τυχαία μία απάντηση random. Ειδάλλως, επιστρέφει μία πρόταση άγνοιας ή απουσίας της απάντησης από την βάση δεδομένων.

```
if prob.item()>0.75 and responses:
    return random.choice([r[0] for r in responses])

    return "I do not understand or maybe I have not the etymology of this word. You can search into the works of other scholars too..."
```

`random.choice(seq)`

Return a random element from the non-empty sequence seq. If seq is empty, raises `IndexError`.

Πηγή <https://docs.python.org/3/library/random.html>

## 7. Γραφικό περιβάλλον chatbot

```
print("Loquamus! You can type 'exitus' anytime to exit the chat.")")
while True:
    # sentence = "do you use credit cards?"
    sentence = input("You: ")
    if sentence == "exitus":
        break

    sentence = tokenize(sentence)
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)

    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    cur.execute("SELECT response FROM responses INNER JOIN tags ON responses.tag_id = tags.id WHERE tags.tag = ?", (tag,))
    responses = cur.fetchall()
    if prob.item() > 0.75 and responses:
        print(f"{bot_name}: {random.choice([r[0] for r in responses])}")
    else:
        print(f"{bot_name}: I do not understand or maybe I have not the etymology of this word. You can search into the works of other scholars too...")

    #print(f"Predicted tag: {tag}, Probability: {prob.item()}")
    #print(f"Database responses: {responses}")
```

Γραφικό περιβάλλον χρήστη

Για το γραφικό περιβάλλον χρήστη θα χρειαστούμε την Tkinter βιβλιοθήκη για την δημιουργία του γραφικού περιβάλλοντος και την επέκταση της Themed Tkinter widgets για περισσότερα γραφικά, την Pillow για την προσθήκη εικόνας στο περιβάλλον από την οποία και θα χρειαστούμε τις μεθόδους Image και ImageTK για την επεξεργασία της εικόνας και την μετατροπή της σε εύχρηστη για την Tkinter, την sqlite3 για την αλληλεπίδραση με την βάση δεδομένων μας έτσι ώστε να φτιάξουμε ένα κουμπί εξαγωγής των δεδομένων μας εν συνόλωι, την pandas για την φόρτωση, επεξεργασία και αποθήκευση των προαναφερθέντων δεδομένων, την os για την διαχείριση αρχείων και φακέλων του υπολογιστή, και από τον κώδικα του chat μας την συνάρτηση get\_response και το όνομα του bot.

```
Import tkinter as tk
```

```

from tkinter import ttk
from chat import get_response, bot_name
from PIL import Image, ImageTk
import sqlite3
import pandas as pd
import os

```

Αρχικά, θα χρειαστεί να ορίσουμε τις συναρτήσεις που θα χρησιμοποιήσουμε στην πορεία για την αποστολή των μηνυμάτων και την εξαγωγή του excel. Έτσι, ορίζουμε συνάρτηση send\_message με όρισμα event=None έτσι ώστε να δημιουργήσουμε μία συνάρτηση που. Να λειτουργεί είτε με event binding είτε χωρίς αυτό, δηλαδή είτε με το πάτημα ενός κουμπιού χωρίς event είτε πατώντας το πλήκτρο enter/return οπότε και η συνάρτηση καλείται με ένα event. Από το πεδίο του χρήστη msg\_entry παίρνουμε την καταχώρηση του .get() και την αποθηκεύουμε στην μεταβλητή msg. Έπειτα ορίζουμε μία εντολή if not για το ενδεχόμενο να μην υπάρχει κάποιο μήνυμα η οποία απλώς θα σταματά την εκτέλεση της συνάρτησης με ένα κενό return. Μετά την αποστολή του μηνύματος θέλουμε το μήνυμα να διαγράφεται από το πεδίο widget του χρήστη οπότε χρησιμοποιούμε την μέθοδο .delete() στο msg\_entry με ορίσματα την αρχή (0) και την ειδική τιμή tk.END για το τέλος του κειμένου που έγραψε ο χρήστης. Για να προστεθεί το μήνυμα στο πλαίσιο επικοινωνίας θα πρέπει να ενεργοποιήσουμε το chat\_text και έπειτα να το απενεργοποιήσουμε εκ νέου. Επομένως, χρησιμοποιούμε την .configure() με όρισμα state=tk.NORMAL έτσι ώστε το αρχικά απενεργοποιημένο TEXT widget να ενεργοποιηθεί για να προστεθεί το κείμενο. Ακολούθως για να εισάγουμε το μήνυμα χρήστη χρησιμοποιούμε την .insert() στο chat\_text με ορίσματα tk.END για την τοποθέτηση του μηνύματος στο τέλος, f-string (f"Tu: {msg}\n") μορφοποίηση για την εμφάνιση του κειμένου και “user” για το χρώμα και την γραμματοσειρά του χρήστη (βλ. tag\_configure). Με την σειρά του η απάντηση του chatbot εμφανίζεται με την ίδια εντολή .insert() με ορίσματα tk.END για να τοποθετείται σε τέλος, f-string (f"{{bot\_name}}: {get\_response(msg)}\n") μορφή με την συνάρτηση και το όνομα του bot από τον κώδικα για το chat, και “bot” για το στίλ μηνύματος του chatbot (βλ. tag\_configure). Τέλος, με την .configure() απενεργοποιούμε την επεξεργασία του κειμένου (state=tk.DISABLED) και για να βλέπει ο χρήστης αυτόματα το τελευταίο μήνυμα με scroll χρησιμοποιούμε την .yview() για το ορατό τμήμα στον άξονα y με όρισμα tk.END για να πηγαίνει αυτόματα προς το τέλος.

```

Def send_message(event=None):
    msg = msg_entry.get()
    if not msg:
        return

    msg_entry.delete(0, tk.END)
    chat_text.configure(state=tk.NORMAL)
    chat_text.insert(tk.END, f"Tu: {msg}\n", "user")
    chat_text.insert(tk.END, f"{{bot_name}}: {get_response(msg)}\n", "bot")
    chat_text.configure(state=tk.DISABLED)
    chat_text.yview(tk.END)

```

Για την εξαγωγή των δεδομένων ετυμολογικού λεξικού σε excel δημιουργούμε μία συνάρτηση export\_to\_excel() για να την καλέσουμε αργότερα. Με τον connector ανοίγουμε σύνδεση με την βάση δεδομένων μας chatbot\_liberx.db. Έπειτα αντικειμενοποιούμε την ερώτηση στην SQL για να επιλέξουμε την στήλη tag και response από τους πίνακες tags και

responses ξεκινώντας την αναζήτηση από τον πίνακα tags που είναι και ο βασικός μας και στον οποίο συνδέεται ο πίνακας responses μέσω των tag\_id και id με όρισμα επιλογής μεγαλύτερο ή ίσο του 6 στα οποία και βρίσκονται οι λέξεις και οι ετυμολογίες τους. Για να μετατρέψουμε τα δεδομένα σε Data Frame χρησιμοποιούμε την εντολή .read\_sql\_query() της pandas με ορίσματα το αντικείμενο της ερώτηση και την σύνδεση στη βάση δεδομένων. Ορίζουμε το όνομα του αρχείο excel και μεταφέρουμε σε αυτό τα δεδομένα .to\_excel() με ορίσματα το ίδιο το αρχείο και index=False για να μην αποθηκευτούν οι αυτόματοι indices της pandas. Για να ανοίξει αυτόματα το αρχείο με το πάτημα του κουμπιού που θα ορίσουμε πιο κάτω χρησιμοποιούμε την εντολή .system() της os με μια f-string και την μεταβλητή του ονόματος του αρχείο και την εντολή ανοίγματος του αρχείου open για macOS<sup>2</sup>.

```
Def export_to_excel():
    conn = sqlite3.connect("chatbot_liberx.db")
    query = """
        SELECT tags.tag, responses.response
        FROM tags
        JOIN responses ON tags.id = responses.tag_id
        WHERE tags.id >= 6
    """
    df = pd.read_sql_query(query, conn)
    conn.close()
    file_path = "etymology_export.xlsx"
    df.to_excel(file_path, index=False)
    os.system(f'open "{file_path}"')
```

**pandas.read\_sql\_query**

```
pandas.read_sql_query(sql, con, index_col=None, coerce_float=True,
params=None, parse_dates=None, chunksize=None, dtype=None,
dtype_backend=<no_default>) [source]
Read SQL query into a DataFrame.

Returns a DataFrame corresponding to the result set of the query string. Optionally provide an index_col parameter to use one of the columns as the index, otherwise default integer index will be used.

Parameters:
sql : str SQL query or SQLAlchemy Selectable (select or text object)
SQL query to be executed.

con : SQLAlchemy connectable, str, or sqlite3 connection
Using SQLAlchemy makes it possible to use any DB supported by that library. If a DBAPI2 object, only sqlite3 is supported.
```

Πηγή [https://pandas.pydata.org/docs/reference/api/pandas.read\\_sql\\_query.html](https://pandas.pydata.org/docs/reference/api/pandas.read_sql_query.html)

**pandas.DataFrame.to\_excel**

```
DataFrame.to_excel(excel_writer, *, sheet_name='Sheet1', na_rep='',
float_format=None, columns=None, header=True, index=True,
index_label=None, startrow=0, startcol=0, engine=None, merge_cells=True,
inf_rep='inf', freeze_panes=None, storage_options=None,
engine_kwargs=None) [source]
Write object to an Excel sheet.

To write a single object to an Excel .xlsx file it is only necessary to specify a target file name. To write to multiple sheets it is necessary to create an ExcelWriter object with a target file name, and specify a sheet in the file to write to.

Multiple sheets may be written to by specifying unique sheet_name. With all data written to the file it is necessary to save the changes. Note that creating an ExcelWriter object with a file name that already exists will result in the contents of the existing file being erased.
```

Πηγή [https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to\\_excel.html](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_excel.html)

---

<sup>2</sup> Για windows fstart excel "{file\_path}"". Για linux fxdg-open "{file\_path}"".

```
os.system(command)
Execute the command (a string) in a subshell. This is implemented by calling the Standard C function system(), and has the same limitations. Changes to sys.stdin, etc. are not reflected in the environment of the executed command. If command generates any output, it will be sent to the interpreter standard output stream. The C standard does not specify the meaning of the return value of the C function, so the return value of the Python function is system-dependent.

On Unix, the return value is the exit status of the process encoded in the format specified for wait().

On Windows, the return value is that returned by the system shell after running command. The shell is given by the Windows environment variable COMSPEC: it is usually cmd.exe, which returns the exit status of the command run; on systems using a non-native shell, consult your shell documentation.

The subprocess module provides more powerful facilities for spawning new processes and retrieving their results; using that module is preferable to using this function. See the Replacing Older Functions with the subprocess Module section in the subprocess documentation for some helpful recipes.

On Unix, waitstatus_to_exitcode() can be used to convert the result (exit status) into an exit code.
On Windows, the result is directly the exit code.

Raises an auditing event os.system with argument command.

Availability: Unix, Windows, not WASI, not Android, not iOS.
```

Πηγή <https://docs.python.org/3/library/os.html>

Με την .Tk() δημιουργούμε το κεντρικό παράθυρο της εφαρμογής και το αντικείμενο ποιούμε στην root. Με την μέθοδο .title() ονομάζουμε την μπάρα τίτλου του παραθύρου root “Chatbot Etymologiarum Liber X”. Με την μέθοδο .geometry() καθορίζουμε το μέγεθος πλάτος-ύψος σε pixels του βασικού παραθύρου.

```
Root = tk.Tk()
root.title("Chatbot Etymologiarum Liber X")
root.geometry("500x600")
```

Ακολούθως δημιουργούμε μέσα στο κεντρικό παράθυρο την κεφαλίδα-τίτλο του παραθύρου με την μέθοδο .Frame() της Tkinter και ορίσματα την τοποθέτηση του στο αντικείμενο root του βασικού παραθύρου, χρώμα φόντου bg σε κιανό και ύψος 50. Με την μέθοδο .pack() τοποθετούμε το header\_frame στο παράθυρο root με όρισμα fill=tk.X για την οριζόντια επέκταση. Για να τοποθετήσουμε στο frame κείμενο χρησιμοποιούμε την μέθοδο .Label() με ορίσματα header\_frame για τον προδιορισμό της θέσης, text για το κείμενο, fg και bg για το χρώμα της σειράς και το background του κειμένου και font την γραμματοσειρά. Κατευθείαν με την μέθοδο .pack() τοποθετούμε το text μέσα στο frame και όρισμα κάθετου περιθωρίου 10.

```
Header_frame = tk.Frame(root, bg="#008080", height=50)
header_frame.pack(fill=tk.X)
tk.Label(header_frame, text="Etymologiarum Liber X", fg="white", bg="#008080",
font=FONT_BOLD).pack(pady=10)
```

Για να βάλουμε μία εικόνα στο παράθυρο μας χρησιμοποιούμε την μέθοδο Image της pillow, με την οποία την ανοίγουμε .open() και μεταβάλουμε το σχήμα της .resize() χρησιμοποιώντας LANCZOS resampling. Με την μέθοδο PhotoImage() του ImageTK μετατρέπουμε την εικόνα logo σε συμβατή προς την Tkinter και φτιάχνουμε ένα Label στο παράθυρο root για να βάλουμε την εικόνα logo.

```
Logo = Image.open("1-fe37d2e0.png").resize((80, 80), Image.Resampling.LANCZOS)
logo = ImageTk.PhotoImage(logo)
logo_label = tk.Label(root, image=logo)
logo_label.pack(pady=10)
```

```
Image.resize(size: tuple[int, int] | list[int] | NumpyArray, resample: int | None = None, box: tuple[float, float, float, float] | None = None, reducing_gap: float | None = None) -> Image
```

Returns a resized copy of this image.

PARAMETERS:

- **size** – The requested size in pixels, as a tuple or array: (width, height).
- **resample** – An optional resampling filter. This can be one of `Resampling.NEAREST`, `Resampling.BOX`, `Resampling.BILINEAR`, `Resampling.HAMMING`, `Resampling.BICUBIC` or `Resampling.LANCZOS`. If the image has mode "I" or "P", it is always set to `Resampling.NEAREST`. If the image mode is "BGR;15", "BGR;16" or "BGR;24", then the default filter is `Resampling.NEAREST`. Otherwise, the default filter is `Resampling.BICUBIC`. See: [Filters](#).
- **box** – An optional 4-tuple of floats providing the source image region to be scaled. The values must be within (0, 0, width, height) rectangle. If omitted or None, the entire source is used.
- **reducing\_gap** – Apply optimization by resizing the image in two steps. First, reducing the image by integer times using `reduce()`. Second, resizing using regular resampling. The last step changes size no less than by `reducing_gap` times. `reducing_gap` may be None (no first step is performed) or should be greater than 1.0. The bigger `reducing_gap`, the closer the result to the fair resampling. The smaller `reducing_gap`, the faster resizing. With `reducing_gap` greater or equal to 3.0, the result is indistinguishable from fair resampling in most cases. The default value is None (no optimization).

RETURNS:

An `Image` object.

Πηγή <https://pillow.readthedocs.io/en/stable/reference/Image.html>

```
class PIL.ImageTk.PhotoImage(image: Image | str | None = None, size: tuple[int, int] | None = None, **kw: Any)
```

A Tkinter-compatible photo image. This can be used everywhere Tkinter expects an image object. If the image is an RGBA image, pixels having alpha 0 are treated as transparent.

The constructor takes either a PIL image, or a mode and a size. Alternatively, you can use the `file` or `data` options to initialize the photo image object.

PARAMETERS:

- **image** – Either a PIL image, or a mode string. If a mode string is used, a size must also be given.
- **size** – If the first argument is a mode string, this defines the size of the image.
- **file** – A filename to load the image from (using `Image.open(file)`).
- **data** – An 8-bit string containing image data (as loaded from an image file).

Πηγή <https://pillow.readthedocs.io/en/stable/reference/ImageTk.html>

Τώρα θα φτιάξουμε το πεδίο εμφάνισης των μηνυμάτων του chatbot. Δημιουργούμε, λοιπόν, ένα Frame στο παράθυρο root και το τοποθετούμε με ορίσματα εξωτερικών περιθωρίων 10 για τον άξονα x και 5 για τον άξονα y, `fill=tk.BOTH` για να γεμίζει σε όλες τις κατευθύνσεις πλάτους και ύψους, και `expand=TRUE` για προσαρμογή του μεγέθους. Επειτα δημιουργούμε ένα πεδίο κειμένου Text στο Frame για τα μηνύματα. `Text()` με ορίσματα το `chat_frame` της θέσης, `wrap=tk.WORD` για αυτόματη αλλαγή γραμμής ανά λέξη και όχι γράμματα, έτσι ώστε να μην κόβονται οι λέξεις, φόντο, χρώμα κειμένου και γραμματοσειρά, και `state=tk.DISABLED` για να μην μπορεί να μεταβληθεί από τον χρήστη. Τοποθετούμε το `chat_text` στο frame `.pack()` με ορίσματα `side=tk.LEFT` για την αριστερή τοποθέτηση στο frame, `fill=tk.BOTH` για να γεμίζει όλος ο διαθέσιμος χώρος, και `expand=True` για δυναμική μεγέθυνση όταν αλλάζει το μέγεθος του παραθύρου. Για να φτιάξουμε το μία κατακόρυφη γραμμή κύλισης χρησιμοποιούμε την μέθοδο `.Scrollbar()` με ορίσματα το `chatframe` όπου τοποθετείτε, και `command=chat_text.yview` όπου και συνδέεται με την κάθετη κίνηση του `chat_text` των μηνυμάτων. Την τοποθετούμε `.pack()` στην δεξιά πλευρά γεμίζοντας τον κατακόρυφο άξονα y. Για να λειτουργούν όμως μάζι το `text` και η `scrollbar` έτσι ώστε η δεύτερη να μεγαλώνει με την προσθήκη του πρώτου, χρησιμοποιούμε την μέθοδο `.config()` με παράμετρο `yscrollcommand=scrollbar.set` για τον συγχρονισμό της γραμμής κύλισης με το μέγεθος του περιεχομένου του κειμένου όποτε αυτό αλλάζει. Τέλος, για να μορφωποιήσουμε τα μηνύματα χρήστη και bot χρησιμοποιούμε την μέθοδο `.tag_configure()` όπου και ορίζεται το χρώμα και η γραμματοσειρά του κειμένου με το αντίστοιχο tag user και bot, τα οποία και χρησιμοποιούμε στην συνάρτηση `send_message`.

```

Chat_frame = tk.Frame(root)
chat_frame.pack(padx=10, pady=5, fill=tk.BOTH, expand=True)
chat_text = tk.Text(chat_frame, wrap=tk.WORD, bg="white", fg=TEXT_COLOR, font=FONT,
state=tk.DISABLED)
chat_text.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar = ttk.Scrollbar(chat_frame, command=chat_text.yview)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
chat_text.config(yscrollcommand=scrollbar.set)
chat_text.tag_configure("user", foreground="#555", font=FONT_BOLD)
chat_text.tag_configure("bot", foreground="#555", font=FONT)

```

Πριν ξεκινήσει η συνομιλία με το bot, θέλουμε να υπάρχει ένα μήνυμα καλωσορίσματος του χρήστη. Γι' αυτό χρησιμοποιούμε την μέθοδο .configure() στο chat\_text έτσι ώστε να μετατρέψουμε το state σε Normal και να μπορούμε να το επεξεργαστούμε. Έπειτα, αφού πλέον μας επιτρέπεται, εισάγουμε ένα κείμενο .insert() με όρισμα το τέλος του κειμένου (στην προκειμένη είναι η αρχή καθώς δεν υπάρχει άλλο κείμενο), σε f-string το όνομα του bot και το μήνυμα καλωσορίσματος και το tag\_configure του bot για την μορφοποίηση της γραμμής. Τέλος, απενεργοποιούμε εκ νέου το κείμενο έτσι ώστε να μην μπορεί να μεταβάλλεται αλλά να γίνεται μόνο displayed.

```

Chat_text.configure(state=tk.NORMAL)
chat_text.insert(tk.END, f"{bot_name}: Loquamur! Quae verba quaeris?\n", "bot")
chat_text.configure(state=tk.DISABLED)

```

Σειρά έχει η δημιουργία του πλαισίου εισαγωγής του μηνύματος χρήστη, όπου πληκτρολογεί ο χρήστης την ετυμολογική ερώτηση που θέλει να κάνει στο bot. Με την .Frame() δημιουργούμε ένα πεδίο για να το μετατρέψουμε σε πεδίο εισαγωγής του χρήστη (msg\_entry) και να βάλουμε και ένα κουμπί αποστολής. Με την .pack() τοποθετούμε το πεδίο γεμίζοντας τον άξονα x και κρατώντας περιθώριο στους δύο άξονες. Έπειτα, δημιουργούμε .Entry() το πεδίο εισαγωγής μέσα στο bottom\_frame με την γραμματοσεριά της αρεσκείας μας. Διατάσσουμε το πεδίο με την .pack() στα αριστερά, με επέκταση και ρπος τους δύο άξονες αν το μέγεθος μεγαλώνει, με προσαρμοστικότητα στον διαθέσιμο χώρο και εξωτερικό κενό στον άξονα x από το κουμπί αποστολής. Με την μέθοδο .bind() συνδέουμε το πάτημα του πλήκτρου Return με το πάτημα του κουμπιού, το πρώτο είναι event None και το δεύτερο είναι event. Έπειτα φτιάχνουμε ένα κουμπί στο bottom frame με text Αποστολή και command την συνάρτηση send\_message, που είναι και η εκτέλεση του κουμπιού Return. Τέλος, τοποθετούμε .pack() το κουμπί στα δεξιά του bottom frame.

```

Bottom_frame = tk.Frame(root)
bottom_frame.pack(fill=tk.X, padx=10, pady=5)
msg_entry = ttk.Entry(bottom_frame, font=FONT)
msg_entry.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=(0, 5))
msg_entry.bind("<Return>", send_message)
send_button = ttk.Button(bottom_frame, text="Αποστολή", command=send_message)
send_button.pack(side=tk.RIGHT)

```

Το τελευταίο widget μας είναι το κουμπί εξαγωγής excel. Δημιουργούμε ένα κουμπί .Button() με θέση το βασικό μας παράθυρο, text Εξαγωγή σε excel και command την συνάρτηση της εξαγωγής. Κάνουμε pack το κουμπί με όρισμα εξωτερικό περιθώριο 10 στον κατακόρυφο άξονα. Τέλος, με την .mainloop() εκκινούμε το παράθυρο root μας.

```
Export_button = ttk.Button(root, text="Εξαγωγή σε Excel", command=export_to_excel)
export_button.pack(pady=10)

root.mainloop()
```

## 8. Πηγές

Για τον κώδικα μου παρακολούθησα τα εξής videos:

<https://www.youtube.com/watch?v=RpWeNzfSUHw>

<https://www.youtube.com/watch?v=8qwowmiXANQ>

<https://www.youtube.com/watch?v=Da-iHgrmHYg&t=608s>

Βιβλιογραφία:

Σ. Λυκοθανάσης, Δ. Κουτσομητρόπουλος, Υπολογιστική Νοημοσύνη και βαθιά μάθηση,  
Κάλλιπος, Πανεπιστήμιο Πατρών, 2021.