Επεξεργασία Φυσικής Γλώσσας – 2022/2023 – Ο.Π.Α

**Εργασία 4**
Φοίβος-Αστέριος Νταντάμης (f3312204 ) (phivos93@yahoo.com)

**Άσκηση 2**

**notebook url:**
**https://colab.research.google.com/drive/1i2vYJH4Wstr2yNaP4fku1ayaVe1M0Rzt?usp=sharing**

**i) Stacked CNNs**

Θα χρησιμοποιήσουμε όπως και στην προηγούμενη άσκηση, τα έτοιμα word embeddings του fasttext, οπότε ξεκινάμε φορτώνοντάς τα

```
fasttext_embed = np.load("/content/gdrive/My Drive/fasttext.npy")
fasttext_word_to_index = pickle.load(open("/content/gdrive/My Drive/fasttext_voc.pkl", 'rb'))
```

Επιλέξαμε αυτήν την φορά ένα διαφορετικό dataset, ώστε να κάνουμε multi label classification και να γίνει λίγο πιο ενδιαφέρον το πρόβλημα. Για το σκοπό αυτό χρησιμοποιήσαμε το brown corpus, μία συλλογή κειμένων με labels την κατηγορία στην οποία ανήκουν (adventure, editorial, fiction κ.τ.λ)

```
!python3 -m nltk.downloader brown
!unzip -o /root/nltk_data/corpora/brown.zip -d /root/nltk_data/corpora
```

Περιγράφουμε το dataset και τυπώνουμε κάποια στατιστικά που ζητούνται

```
We will use the Brown Corpus, which was the
first million-word electronic corpus of English,
created in 1961 at Brown University. The corpus containts
text from 500 surces and the sources have benn categorized
by genre (https://www.nltk.org/book/ch02.html#tab-brown-sources)

Dataset categories are:
['adventure', 'belles_lettres', 'editorial', 'fiction', 'governme

Number of documents in dataset is:
500

Average document length is:
2322.384

Std of documents lenghts is:
13.107004875007762
```

Στη συνέχεια σπάμε το dataset μας σε train, development και test με ποσοστά 70%, 20%, 10% και τυπώνουμε κάποιες πληροφορίες για το κάθε ένα.

```
doc_ids = brown.fileids()

train_ids, temp_ids = train_test_split(doc_ids, test_size=0.3, random_state=17)

dev_ids, test_ids = train_test_split(temp_ids, test_size=0.333, random_state=25)
```

```
Number of docs in
Train Dataset: 350,
Dev Dataset: 100,
Test Dataset: 50

Average document length for:
Train Dataset: 2322.0714285714284,
Dev Dataset: 2322.87,
Test Dataset: 2323.6
```

Στη συνέχεια κάνουμε ένα βασικό preprocessing όπου αφαιρούμε κυρίως χαρακτήρες και λέξεις που θεωρούμε ότι δε συμβάλλουν πολύ στην εξαγωγή νοήματος από το κείμενο.

```
stop_words = stopwords.words('english')
stopwords_dict = Counter(stop_words) # this is just for optimazation purposes

for i in [(train_ids, "train"), (dev_ids, "dev"), (test_ids, "test")]:
  for doc_id in i[0]:
    doc_temp = brown.words(fileids=doc_id)
    # filter stopwords and make lowercase
    doc = ' '.join([word.lower() for word in doc_temp if word not in stopwords_dict])
    # Remove non-word (special) characters such as punctuation
    doc = re.sub(r'\W', ' ', doc)
    # Remove all single characters
    doc = re.sub(r'\s+[a-z]\s+', ' ', doc)
    # Remove numbers
    doc = re.sub(r'\s+[0-9]+', ' ', doc)
    # Substitute multiple spaces with single space
    doc = re.sub(r'\s+', ' ', doc, flags=re.I)

    x[i[1]].insert(0, doc)
    y[i[1]].insert(0, brown.categories(fileids=doc_id))
    ids[i[1]].insert(0, doc_id)
```

Υπολογίζουμε τώρα το μέσο μήκος και την τυπική απόκλιση των μηκών των κειμένων των δεδομένων train.

```
Avg length of train documents after preprocessing is:
1144.3942857142856

Std of length is:
72.55235722411443
```

Χρησιμοποιούμε τώρα τον MultiLabelBinarizer ώστε να μετατρέψουμε τα labels-κατηγορίες σε one-hot μόρφη.

```
mlb = MultiLabelBinarizer()

mlb.fit([brown.categories()])

for i in ["train", "dev", "test"]:
  y[i] = mlb.transform(y[i])

print(y["train"][0])
```

```
[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
```

Στο επόμενο βήμα μετατρέπουμε τις προτάσεις του dataset μας σε ακολουθίες ακεραίων οι οποίοι αντιστοιχούν στο index των λέξεων στην αντίστοιχη θέση. Ταυτόχρονα δημιουργούμε το λεξικό μας που περιέχει μόνο τις λέξεις που συναντώνται στα train δεδομένα. Το μέγεθος μιας ακολουθίας ορίστηκε να είναι ίσο με 1250 και ισούται περίπου με το άθροισμα του μέσο όρου και της τυπικής απόκλισης ώστε να αφήσει λίγες λέξεις εκτός σε κάποιες πιο μεγάλες προτάσεις, αλλά παράλληλα να μην χρειαστεί υπερβολικό padding στις μικρές.

```
MAX_WORDS = 100000 # We keep all the words actually
MAX_SEQUENCE_LENGTH = 1250 # > avg + std
EMBEDDING_DIM = fasttext_embed.shape[1]

tokenizer = Tokenizer(num_words=MAX_WORDS,oov_token='__UNK__')
tokenizer.fit_on_texts(x["train"])

train_seqs = tokenizer.texts_to_sequences(x["train"])
dev_seqs = tokenizer.texts_to_sequences(x["dev"])
test_seqs = tokenizer.texts_to_sequences(x["test"])

train_data = pad_sequences(train_seqs, maxlen=MAX_SEQUENCE_LENGTH, padding='post', truncating='post')
dev_data = pad_sequences(dev_seqs, maxlen=MAX_SEQUENCE_LENGTH, padding='post', truncating='post')
test_data = pad_sequences(test_seqs, maxlen=MAX_SEQUENCE_LENGTH, padding='post', truncating='post')
```

Αρχικοποιούμε τώρα τον πίνακα με τα embeddings μας προσθέτοντας μόνο τις λέξεις που ανήκουν στο λεξικό που δημιουργήσαμε. Κρατάμε δηλαδή από το fasttext μόνο ό,τι μας χρειάζεται.

```
embedding_matrix = np.zeros((MAX_WORDS, EMBEDDING_DIM))

for word, i in word_index.items():
    if i > MAX_WORDS:
            continue
    try:
        embedding_vector = fasttext_embed[fasttext_word_to_index[word],:]
        embedding_matrix[i] = embedding_vector
    except:
        pass
```

Ορίζουμε συναρτήσεις οι οποίες θα χρησιμοποιηθούν για τον υπολογισμό των μετρικών επίδοσης.

```
def recall(y_true, y_pred):

    """
    Recall metric.
    Only computes a batch-wise average of recall.
    Computes the recall, a metric for multi-label classification of
    how many relevant items are selected.
    """
    true_positives = K.sum(K.roun round: Any rue * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def precision(y_true, y_pred):

    """
    Precision metric.
    Only computes a batch-wise average of precision.
    Computes the precision, a metric for multi-label classification of
    how many selected items are relevant.
    Source
    ------
    https://github.com/fchollet/keras/issues/5400#issuecomment-314747992
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision
```

```
def f1(y_true, y_pred):

    """Calculate the F1 score."""
    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    return 2 * ((p * r) / (p + r + K.epsilon()))


def accuracy(y_true, y_pred):
    return K.mean(K.equal(y_true, K.round(y_pred)), axis=1)
```

Ορίζουμε όπως και την τελευταία φορά μία callback συνάρτηση που θα χρησιμοποιηθεί για την καταγραφή των μετρικών που μας ενδιαφέρουν στο τέλος κάθε εποχής. Αυτές θα υπολογίζονται για λογαριασμό του tuner, ενώ οι προηγούμενες συναρτήσεις περνιούνται στο compile του μοντέλου και κάνουν υπολογισμούς κάθε φορά που το μοντέλο τρέχει.

```
class Metrics(tf.keras.callbacks.Callback):
    def __init__(self, valid_data):
        super(Metrics, self).__init__()
        self.validation_data = valid_data

    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        val_predict = np.argmax(self.model.predict(self.validation_data[0]), -1)
        val_targ = self.validation_data[1]
        if len(val_targ.shape) == 2 and val_targ.shape[1] != 1:
            val_targ = np.argmax(val_targ, -1)
        val_targ = tf.cast(val_targ,dtype=tf.float32)


        _val_f1 = f1_score(val_targ, val_predict,average="weighted")
        _val_recall = recall_score(val_targ, val_predict,average="weighted")
        _val_precision = precision_score(val_targ, val_predict,average="weighted")

        logs['val_f1'] = _val_f1
        logs['val_recall'] = _val_recall
        logs['val_precision'] = _val_precision
        print(" — val_f1: %f — val_precision: %f — val_recall: %f" % (_val_f1, _val_precision, _val_recall))
        return
```

Πριν ξεκινήσει η αναζήτηση, ορίζουμε δύο ακόμα callback συναρτήσεις. Την EarlyStopping που θα είναι υπεύθυνη για τον τερματισμό της αναζήτησης σε περιπτώσεις που τα αποτελέσματα δεν βελτιώνονται και την ReduceLROnPlateau που θα μπορεί να τροποποιεί το ρυθμό μάθησης στο τέλος κάθε εποχής. Αυτήν την φορά φροντίσαμε ώστε το patience του ReduceLROnPlateau να είναι μικρότερο από αυτό του EarlyStoppping ώστε το δεύτερο το learning rate να μπορεί να αλλάζει πριν γίνει το early stop.

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

es = EarlyStopping(
  monitor="val_loss",
  patience=15,
  restore_best_weights=True)

lr = ReduceLROnPlateau(
  monitor="val_loss",
  factor=0.2,
  patience=7
)
```

Προκειμένου να κάνουμε tune τις υπερπαραμέτρους που θέλουμε, κάνουμε χρήση του keras tuner. Για να χρησιμοποιήσουμε οποιονδήποτε από τους διαθέσιμους tuners, πρέπει να φτιάξουμε αρχικά ένα HyperModel μέσα στο οποίο θα ορίζεται και ο χώρος αναζήτησης των υπερπαραμέτρων.

```
class MyHyperModel(kt.HyperModel):
  def build(self, hp):
    inputs = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype="int32")

    embeddings = Embedding(input_dim=MAX_WORDS, # Size of the vocabulary
                           output_dim=EMBEDDING_DIM,
                           weights=[embedding_matrix],
                           input_length=MAX_SEQUENCE_LENGTH,
                           trainable=False)(inputs)

    dropped_embeddings = Dropout(rate=0.2)(embeddings)

    for i in range(1, hp.Int(name="number_of_conv_layers", min_value=1, max_value=7, step=1) + 1):
      convs = Conv1D(filters=EMBEDDING_DIM, kernel_size=hp.Int(name="kernel_size", min_value=2, max_value=4, step=1 ), activation='relu', padding='same')(dropped_embed
      resids = Add()([convs, dropped_embeddings])
      dropped_embeddings = Dropout(rate=0.2)(resids)

    pooled_convs = GlobalMaxPooling1D()(dropped_embeddings)

    dropped_pooled_convs = Dropout(rate=0.2)(pooled_convs)

    outputs = Dense(N_CLASSES, activation='softmax')(dropped_pooled_convs)

    model = Model(inputs=inputs, outputs=outputs)

    print(model.summary())

    model.compile(loss='categorical_crossentropy',
                  optimizer=SGD(learning_rate=0.1),
                  metrics=[precision, recall, f1, accuracy, tf.keras.metrics.AUC()])

    return model
```

Στη συνέχεια με αξιοποίηση και της GPU εκκινεί η αναζήτηση του βέλτιστου συνδυασμού των υπερπαραμέτρων. Οι υπερπαράμετροι που ψάχνουμε είναι το πλήθος των CNN layers και το μέγεθος των φίλτρων.

```
tuner = kt.BayesianOptimization(
    hypermodel=MyHyperModel(),
    objective=kt.Objective('f1', direction='max')
    )

if not os.path.exists('/content/gdrive/My Drive/checkpoints'):
  os.makedirs('/content/gdrive/My Drive/checkpoints')

checkpoint = ModelCheckpoint('/content/gdrive/My Drive/keras_Deep_CNN_model', monitor='val_f1', verbose=1, save_best_only=True, mode='max')

with tf.device('/device:GPU:0'):
  tuner.search(
      x=train_data, y=y["train"],
      validation_data=(dev_data, y["dev"]),
      callbacks=[es, lr, checkpoint, Metrics(valid_data=(dev_data, y["dev"]))],
      batch_size = 32,
      epochs = 100
  )
```

Έχοντας κρατήσει τον καλύτερο συνδυασμό υπερπαραμέτρων, χτίζουμε το τελικό μοντέλο και το εκπαιδεύουμε πάνω στα δεδομένα εκπαίδευσης. Οι 10 εποχές που επιλέξαμε τελικά προκύπτουν από το γράφημα που παραθέτουμε πιο κάτω.

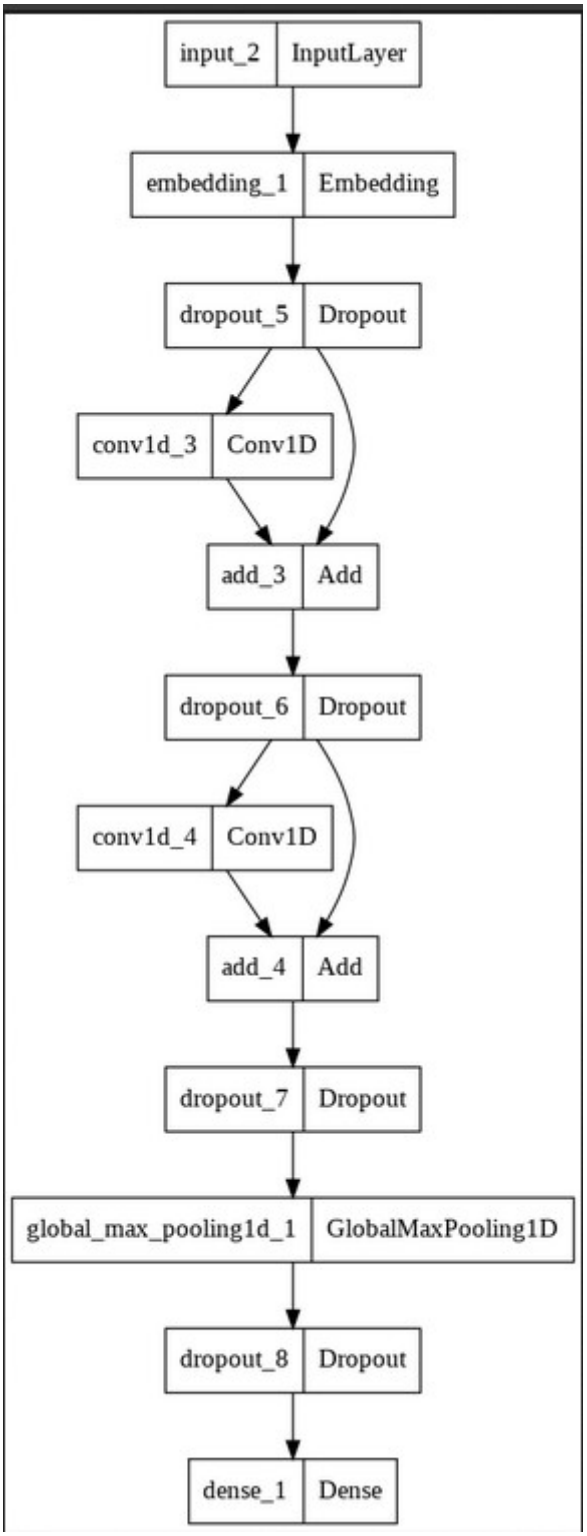
```
best_hp = tuner.get_best_hyperparameters()[0]
model = tuner.hypermodel.build(best_hp)

history = model.fit(x=train_data,
                    y=y["train"],
                    validation_data=(dev_data, y["dev"]),
                    epochs=10,
                    callbacks=[checkpoint, Metrics(valid_data=(dev_data, y["dev"]))]
                    )
```

Τυπώνοουμε τα reports των αποτελεσμάτων και για τα τρία dataser, αφού πρώτα μετατρέψουμε τις προβλέψεις σε 1-hot μορφή. Παραθέτουμε εδώ ενδεικτικά τα αποτελέσματα μόνο για το train. Είναι περίεργο ότι κατά το tuning το f1 εμφανιζόταν με τιμή 0.7 περίπου την πρώτη φορά που κάναμε αναζήτηση, 0.5 τη δεύτερη και τελικά είναι πολύ πιο χαμηλό στην πραγματικότητα.

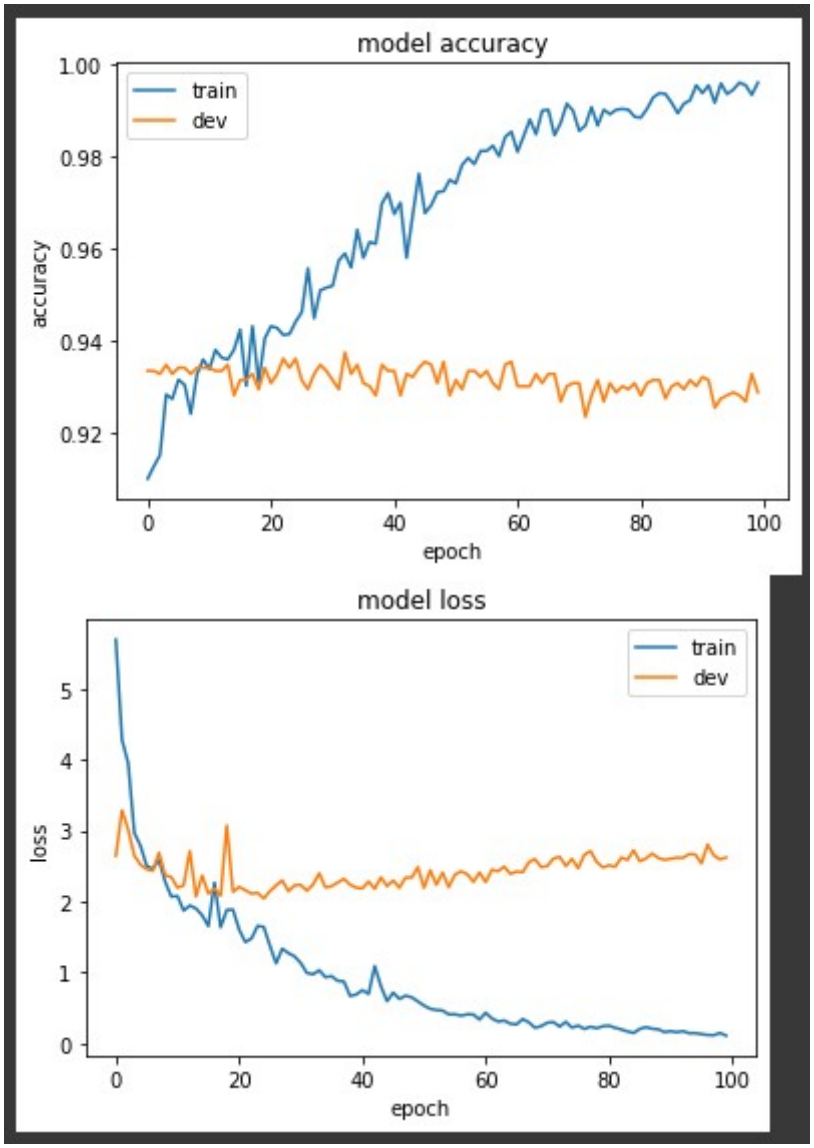
```
predictions_train = model.predict(train_data)
y_pred = np.argmax(predictions_train, axis=1)
pred_list = []

for pred in y_pred:
  temp = np.zeros(15, dtype=int)
  temp[pred] = 1
  pred_list.append(temp)
y_pred_1_hot = np.asarray(pred_list)

print(classification_report(y["train"], y_pred_1_hot, target_names=np.array(brown.categories())))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| adventure | 1.00 | 0.29 | 0.45 | 17 |
| belles_lettres | 0.35 | 0.69 | 0.47 | 51 |
| editorial | 0.44 | 0.41 | 0.42 | 17 |
| fiction | 0.00 | 0.00 | 0.00 | 22 |
| government | 1.00 | 0.09 | 0.17 | 22 |
| hobbies | 1.00 | 0.04 | 0.08 | 23 |
| humor | 0.00 | 0.00 | 0.00 | 4 |
| learned | 0.50 | 0.79 | 0.61 | 56 |
| lore | 1.00 | 0.09 | 0.17 | 33 |
| mystery | 0.18 | 1.00 | 0.31 | 20 |
| news | 0.72 | 0.55 | 0.62 | 33 |
| religion | 0.00 | 0.00 | 0.00 | 13 |
| reviews | 0.00 | 0.00 | 0.00 | 12 |
| romance | 0.00 | 0.00 | 0.00 | 22 |
| science_fiction | 0.00 | 0.00 | 0.00 | 5 |
| micro avg | 0.39 | 0.39 | 0.39 | 350 |
| macro avg | 0.41 | 0.26 | 0.22 | 350 |
| weighted avg | 0.50 | 0.39 | 0.32 | 350 |
| samples avg | 0.39 | 0.39 | 0.39 | 350 |

Τυπώνουμε το γράφημα του μοντέλου που επιλέχθηκε



Δημιουργούμε τώρα τα γραφήματα για το accuracy και το loss συναρτήσει των εποχών. Βλέπουμε ότι μετά τις 10 εποχές κάνουμε απλώς overfit στα train δεδομένα, οπότε για αυτό και επιλέξαμε αυτό το σημείο τελικά.




Στη συνέχεια ασχοληθήκαμε με το BERT. Δεν θα παραθέσουμε τον κώδικα εδώ για να μην ξεπεράσουμε πολύ το όριο των 10 σελίδων. Η διαδικασία εκπαίδευσης περιγράφεται ήδη στον κώδικα του εργαστηρίου και αυτήν ακολουθήσαμε. Αυτό που χρειαζόταν ρύθμιση ήταν κυρίως τα TrainingArguments που περνάμε στο μοντέλο.

Τροποποίηση του κώδικα χρειαζόταν και στο search των υπερπαραμέτρων, σχετικά με τον ορισμό των παραμέτρων και του χώρου αναζήτησης, αλλά και των arguments για το training. Δυστυχώς με έπειτα από πολλές προσπάθειες δεν καταφέραμε να κάνουμε τον κώδικα να δουλέψει όπως πρέπει. Ενώ το μοντέλο χτίζεται, οι υπερπαράμετροι αρχικοποιούνται και η εκτέλεση ξεκινάει, φαίνεται ότι κάθε φορά ο worker εμφανίζει σφάλμα oom(out of memory) με αποτέλεσμα να μην επιτυγχάνει τελικά καμία εκτέλεση. Δοκιμάσαμε όλες τις προτεινόμενες λύσεις στο διαδίκτυο αλλά καμία δεν

λειτούργησε. Το πρόβλημα μπορεί να οφείλεται στο περιβάλλον του collab και στον περιορισμό που έχει στην χρήση της μνήμης.