Επεξεργασία Φυσικής Γλώσσας – 2022/20223 – Ο.Π.Α

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

Άσκηση 1 notebook url: https://colab.research.google.com/drive/1BxGYsttx1HnUWId0sq-ztt68aDIyslsG?usp=sharing

Περιγραφή Dataset:

Το dataset είναι το ίδιο με τις δύο προηγούμενες ασκήσεις και έχει ήδη περιγραφεί εκεί.

Word Embeddings:

Μας ζητείται να αντιμετωπίσουμε το πρόβλημα του classification των κειμένων του dataset, αυτήν την φορά με την χρήση RNNs. Θα χρησιμοποιήσουμε για το σκοπό αυτόν έτοιμα word embeddings και πιο συγκεκριμένα τα fasttext word embeddings. Ξεκινάμε λοιπόν κατεβάζοντας το αρχείο που τα περιέχει

```
!wget https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.vec.gz
!gzip -d cc.en.300.vec.gz
```

Για λόγους απόδοσης στη συνέχεια διαβάζουμε τα embeddings μία φορά και δημιουργούμε έναν δισδιάστατο πίνακα που περιέχει τα διανύσματα που τα περιγράφουν και ένα dictionary που αποθηκεύει τον αριθμό της γραμμής που έχει αποθηκευτεί το διάνυσμα κάθε λέξης, δηλαδή κρατάμε ένα πολύ απλό index. Προσθέτουμε δύο ειδικές λέξεις, το PADDING που θα χρησιμοποιηθεί για το “γέμισμα” των προτάσεων εισόδου που θα είναι πιο μικρές από το μέγεθος που θα ορίσουμε, όπως και το UNK που θα αντικαταστήσει τις λέξεις εκτός λεξικού.

```
import numpy as np

idx = 0
vocab = {}
# Save embeddings to numpy array (vocab_size x dimensions)
# We manually add 2 special tokens (for padding & unknown words)
with open("cc.en.300.vec", 'r', encoding="utf-8", newline='\n',errors='ignore') as f:
    for l in f:
        line = l.rstrip().split(' ')
        if idx == 0:
            vocab_size = int(line[0]) + 2
            dim = int(line[1])
            vecs = np.zeros(vocab_size*dim).reshape(vocab_size,dim)
            vocab["__PADDING__"] = 0
            vocab["__UNK__"] = 1
            idx = 2
        else:
            vocab[line[0]] = idx
            emb = np.array(line[1:]).astype(float)
            if (emb.shape[0] == dim):
                vecs[idx,:] = emb  # Embeddings are saved in an numpy array vecs
                idx+=1
            else:
                continue
```

Dataset Preprocessing:

Το preprocessing έχει απλοποιηθεί σε σχέση με τις προηγούμενες ασκήσεις. Αυτό συνέβη διότι θέλουμε τελικά το dataset να περιέχει λέξεις οι οποίες μπορούν να απαντηθούν στο λεξικό, οπότε δεν τις αλλοιώσαμε.

```
# Remove all stopwords
stop_words = stopwords.words('english')
stopwords_dict = Counter(stop_words) # this is just for optimazation purposes
document = ' '.join([word for word in doc.split() if word not in stopwords_dict])


# Convert to Lowercase
document = document.lower()

# Remove non-word (special) characters such as punctuation
document = re.sub(r'\W', ' ', document)

# Remove all single characters
document = re.sub(r'\s+[a-z]\s+', ' ', document)


# Remove numbers
document = re.sub(r'\s+[0-9]+', ' ', document)

# Remove honorifics
document = re.sub(r'\s+mr\s+|\s+dr\s+|\s+jr\s+|\s+ms\s+|\s+miss\s+', ' ', document)

# Substitute multiple spaces with single space
document = re.sub(r'\s+', ' ', document, flags=re.I)

# Append all documents into a list 'docs'
docs.append(document)
```

Dataset Splitting:

Χωρίζουμε τώρα το αρχικό dataset σε train, development και test δεδομένα με ποσοστά 70%, 20% και 10% αντίστοιχα.

```
x_train, x_temp, y_train, y_temp = train_test_split(docs, y, test_size=0.3, random_state=17)
x_dev, x_test, y_dev, y_test = train_test_split(x_temp, y_temp, test_size=0.333, random_state=25)
```

Tokenization:

Στη συνέχεια, σπάμε τα δεδομένα μας σε προτάσεις και μετατρέπουμε τις προτάσεις αυτές σε λίστες από tokens.

```
nlp = spacy.load('en_core_web_sm',disable=["tagger", "parser","ner"])
nlp.add_pipe('sentencizer')

def tokenize_samples(samples):

  tokenized_samples = []
  for i in range(len(samples)):
    doc = nlp(samples[i])  # Tokenize the sample into sentences
    tokens = []
    for sent in doc.sents:
      for tok in sent:  # Iterate through the words of the sentence
        if '\n' in tok.text or "\t" in tok.text or "--" in tok.text or "*" in tok.text or tok.text.lower() in STOP_WORDS:
          continue
        if tok.text.strip():
          tokens.append(tok.text.replace('"',"'").strip())
    tokenized_samples.append(tokens)

  return tokenized_samples

x_train_tokenized = tokenize_samples(x_train)
x_dev_tokenized = tokenize_samples(x_dev)
x_test_tokenized = tokenize_samples(x_test)
```

Υπολογίζουμε τώρα το μέσο μήκος, αλλά και την τυπική απόκλιση των μηκών όλων των προτάσεων στα train δεδομένα.

```
# Get mean and std for length on trainning set
print('Average length of smples: {}'.format(np.mean([len(x) for x in x_train_tokenized])))
print('Std length of samples: {}'.format(np.std([len(x) for x in x_train_tokenized])))
print('#Samples with length > 1000: {} \n'.format(np.sum([len(x) > 1000 for x in x_train_tokenized])))
print('x_example: {}'.format(x_train_tokenized[0]))
```

Sequences:

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 450 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 = 450 # > avg + std
EMBEDDING_DIM = fasttext_embed.shape[1]

# Init tokenizer
tokenizer = Tokenizer(num_words=MAX_WORDS, oov_token='__UNK__')
# num_words: the maximum number of words to keep, based on word frequency.
# oov_token: will be used to replace OOV WORDS

# Fit tokenizer (Updates internal vocabulary based on a list of texts.)
tokenizer.fit_on_texts([" ".join(x) for x in x_train_tokenized])

# Converts text to sequences of IDs
train_seqs = tokenizer.texts_to_sequences([" ".join(x) for x in x_train_tokenized])
dev_seqs = tokenizer.texts_to_sequences([" ".join(x) for x in x_dev_tokenized])
test_seqs = tokenizer.texts_to_sequences([" ".join(x) for x in x_test_tokenized])


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')
```

Embedding matrix:

Αρχικοποιούμε 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+2, EMBEDDING_DIM))  # +2 (pad, unkown)

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
```

1-hot vectors:

Δημιουργούμε τα 1-hot vectors των δύο κλάσεων μας

```
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
target_list = target_names

y_train_1_hot = lb.fit_transform([target_list[x] for x in y_train])

y_dev_1_hot = lb.transform([target_list[x] for x in y_dev])
```

Metrics Callback:

Ορίζουμε όπως και την τελευταία φορά μία callback συνάρτηση που θα χρησιμοποιηθεί για την καταγραφή των μετρικών που μας ενδιαφέρουν στο τέλος κάθε εποχής.

```
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
```

Deep attention layer:

Ορίζουμε τώρα ένα custom keras layer που θα υλοποιεί το μηχανισμό του self attention. Παραμετροποιούμε την κλάση ως προς τον αριθμό των επιπέδων που θα δημιουργεί στο MLP που θα χρησιμοποιεί. Στον constructor λοιπόν δημιουργούμε dictionaries για τους regularizers και τα constraints για όλα τα W και b που θα χρειαστούν, ανάλογα με το πόσα επίπεδα θα δημιουργηθούν.

```
def __init__(self,
             number_of_layers, kernel_regularizer=None, u_regularizer=None, bias_regularizer=None,
             W_constraint=None, u_constraint=None, b_constraint=None, bias=True, **kwargs):

    self.supports_masking = True
    self.init = initializers.get('glorot_uniform')

    self.n_layers = number_of_layers

    self.W_regularizers = {}
    self.b_regularizers = {}
    self.W_constraints = {}
    self.b_constraints = {}

    for i in range(1, self.n_layers + 1):
      self.W_regularizers["Wr{}".format(i)] = regularizers.get(kernel_regularizer)
      self.b_regularizers["br{}".format(i)] = regularizers.get(bias_regularizer)
      self.W_constraints["Wc{}".format(i)] = regularizers.get(W_constraint)
      self.b_constraints["bc{}".format(i)] = regularizers.get(bias_regularizer)

    self.u_regularizer = regularizers.get(u_regularizer)
    self.bu_regularizer = regularizers.get(bias_regularizer)
    self.u_constraint = constraints.get(u_constraint)
    self.bu_constraint = constraints.get(b_constraint)

    self.bias = bias
    super(DeepAttention, self).__init__(**kwargs)
```

Μέσα στη μέθοδο build που χτίζει το layer μέσα σε ένα μοντέλο, αρχικοποιούνται όλοι οι πίνακες W και b, δηλαδή με άλλα λόγια αρχικοποιούνται τα βάρη όλων των νευρώνων για όλα τα επίπεδα του MLP.

```
self.Ws = {}
self.bs = {}

for i in range(1, self.n_layers + 1):
  self.Ws["W{}".format(i)] = self.add_weight(shape=(input_shape[-1], input_shape[-1],),
                          initializer=self.init,
                          name='{}_W{}'.format(self.name, i),
                          regularizer=self.W_regularizers["Wr{}".format(i)],
                          constraint=self.W_constraints["Wc{}".format(i)])
  if self.bias:
    self.bs["b{}".format(i)] = self.add_weight(shape=(input_shape[-1],),
                             initializer='zero',
                             name='{}_b{}'.format(self.name, i),
                             regularizer=self.b_regularizers["br{}".format(i)],
                             constraint=self.b_constraints["bc{}".format(i)])
  else:
      self.bs["b{}".format(i)] = None

  self.u = self.add_weight(shape=(input_shape[-1],),
                          initializer=self.init,
                          name='{}_u'.format(self.name),
                          regularizer=self.u_regularizer,
                          constraint=self.u_constraint)

  if self.bias:
    self.bu = self.add_weight(shape=(1,),
                            initializer='zero',
                            name='{}_bu'.format(self.name),
                            regularizer=self.bu_regularizer,
                            constraint=self.bu_constraint)
  else:
    self.bu = None
```

Στην μέθοδο call υλοποιούμε πλέον το layer και αφού κάθε πίνακας W αντιστοιχεί ουσιαστικά σε ένα επίπεδο ενός MLP, διαδίδουμε τα αποτελέσματα από το ένα επίπεδο στο επόμενο πολλαπλασιάζοντας με τη σειρά τα αποτελέσματα που προκύπτουν με τους πίνακες αυτούς.

```
def call(self, x, mask=None):
    # uit = tanh(Wx + b)
    for i in range(1, self.n_layers + 1):
      uit = dot_product(x, self.Ws["W{}".format(i)])
      if self.bias:
          uit += self.bs["b{}".format(i)]
      uit = K.tanh(uit)
      x = uit

    # ait = softmax(Ueij)
    eij = dot_product(uit, self.u)
    if self.bias:
        eij += self.bu


    a = K.expand_dims(K.softmax(eij, axis=-1))

    weighted_input = x * a
    result = K.sum(weighted_input, axis=1)

    # if self.return_attention:
    #     return [result, a]
    return result
```

Tuning:

Προκειμένου να κάνουμε tune τις υπερπαραμέτρους που θέλουμε, αυτήν την φορά 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):
    model = Sequential()
    # Embedding layer
    model.add(Embedding(input_dim=MAX_WORDS+2, output_dim=EMBEDDING_DIM, weights=[embedding_matrix],
                input_length=MAX_SEQUENCE_LENGTH,mask_zero=True, trainable=False))
    # RNN layers
    for i in range(1, hp.Int(name="number_of_rnn_layers", min_value=1, max_value=4, step=1) + 1):
      model.add(Dropout(hp.Float(name="drop_rate", min_value=0.05, max_value=0.45, step=0.10)))
      model.add(Bidirectional(GRU(GRU_SIZE, return_sequences=True, recurrent_dropout = hp.Float(name="drop_rate", min_value=0.05, max_value=0.45, step=0.10)), inp
    model.add(Dropout(hp.Float(name="drop_rate", min_value=0.05, max_value=0.45, step=0.10)))
    model.add(DeepAttention(number_of_layers=hp.Int(name="mlp_layers", min_value=1, max_value=5, step=1)))
    model.add(Flatten())
    model.add(Dense(1,  activation='sigmoid'))

    print(model.summary())
    model.compile(optimizer=SGD(),
                  loss='binary_crossentropy',
                  metrics=["accuracy"])
    return model
```

Μπορούμε τώρα το HyperModel αυτό να το περάσουμε σε κάποιον tuner και συγκεκριμένα στον BayesianOptimization.

```
import keras_tuner as kt

tuner = kt.BayesianOptimization(
    hypermodel=MyHyperModel(),
    objective="val_accuracy")
```

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.

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

es = EarlyStopping(
  monitor="val_loss",
  patience=7,
  restore_best_weights=True)

lr = ReduceLROnPlateau(
  monitor="val_loss",
)
```

Στη συνέχεια με αξιοποίηση αυτήν την φορά και της GPU εκκινεί η αναζήτηση του βέλτιστου συνδυασμού των υπερπαραμέτρων.

```
tuner = kt.BayesianOptimization(
    hypermodel=MyHyperModel(),
    objective="val_accuracy")

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

checkpoint = ModelCheckpoint('/content/gdrive/My Drive/checkpoints/BiGRUMLP.hdf5',
                             monitor='val_accuracy',
                             mode='max', verbose=2,
                             save_best_only=True,
                             save_weights_only=True)

with tf.device('/device:GPU:0'):
  tuner.search(
      x=train_data, y=y_train_1_hot,
      validation_data=(dev_data, y_dev_1_hot),
      callbacks=[es, lr, checkpoint, Metrics(valid_data=(dev_data, y_dev_1_hot))],
      batch_size = 128,
      epochs = 40,
  )
```

Σε αυτό το σημείο θα πρέπει να απολογηθούμε για την επιλογή ακόμα μια φορά όχι των συνολικά βέλτιστων παραμέτρων, αλλά φαίνεται αδύνατο να προλάβει να τελειώσει η αναζήτηση πριν μας διακόψει το collab ή περιορίσει την πρόσβαση στην gpu.

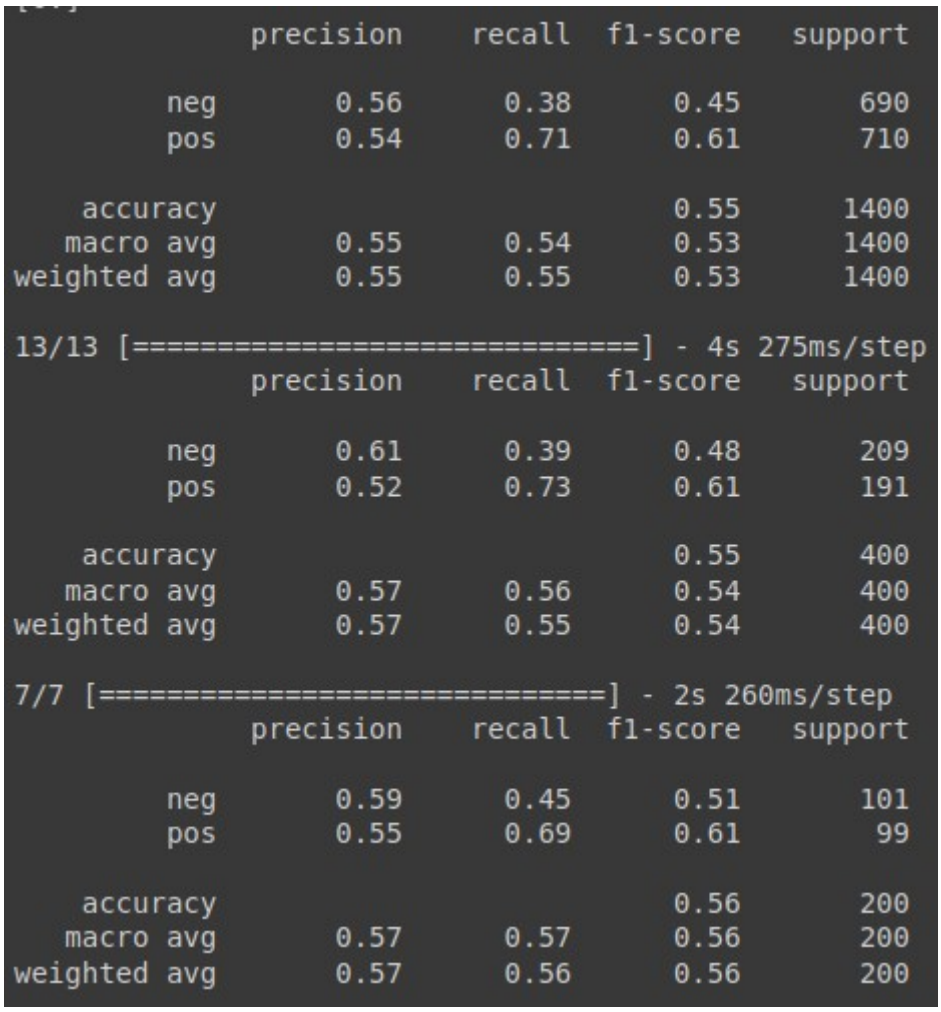
Δημιουργία του τελικού μοντέλου και εκπαίδευσή του:

Έχοντας κρατήσει τον καλύτερο συνδυασμό υπερπαραμέτρων, χτίζουμε το τελικό μοντέλο και το εκπαιδεύουμε πάνω στα δεδομένα εκπαίδευσης.

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

history = model.fit(x=train_data,
                    y=y_train_1_hot,
                    validation_data=(dev_data, y_dev_1_hot),
                    epochs=40,
                    callbacks=[checkpoint, Metrics(valid_data=(dev_data, y_dev_1_hot))]
                    )
```

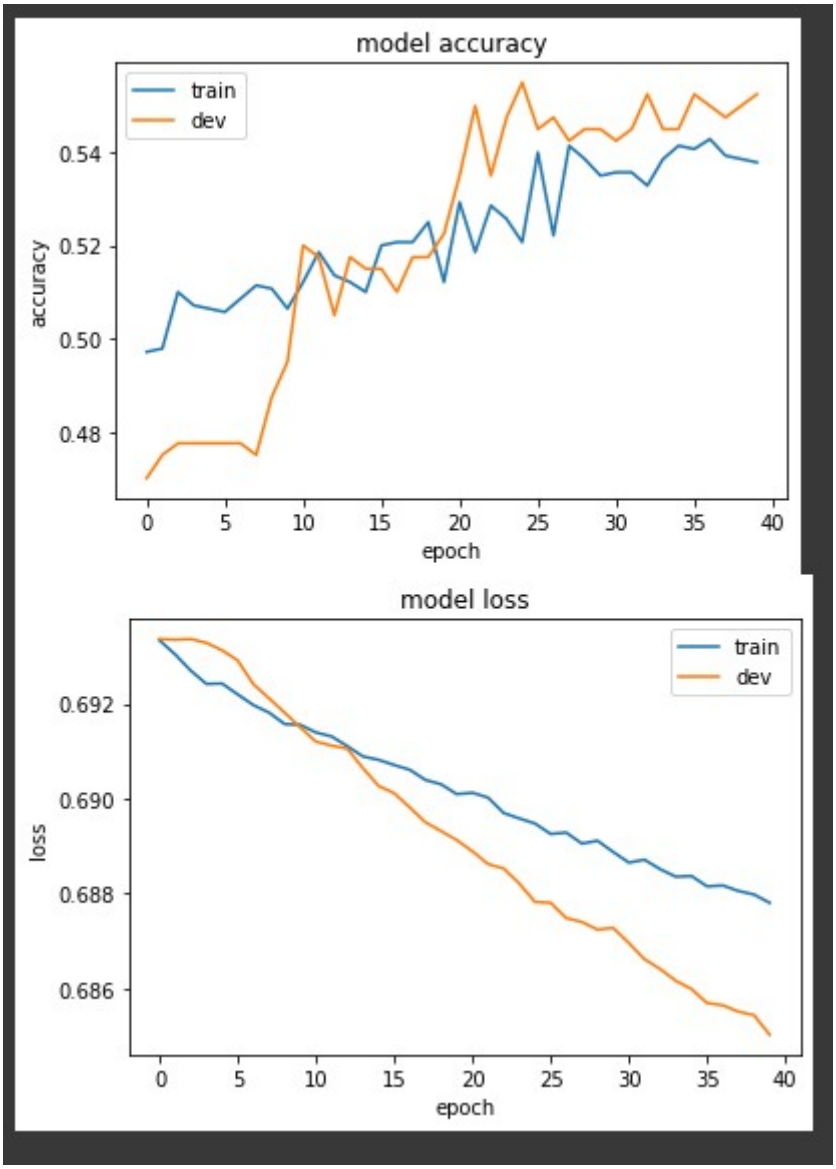
Εμφανίζουμε τώρα τα reports με τις ζητούμενες μετρικές. Λείπει το auc, μιας και έπρεπε να περαστεί επίσης ως callback function για να υπολογίζεται, μα είναι αργά πλέον.

```
predictions_train = model.predict(train_data)
for i in range(len(predictions_train)):
  if predictions_train[i] > 0.5:
    predictions_train[i] = 1
  else:
    predictions_train[i] = 0
print(predictions_train[0])
print(classification_report(y_train, predictions_train, target_names=target_names))

predictions_dev = model.predict(dev_data)
for i in range(len(predictions_dev)):
  if predictions_dev[i] > 0.5:
    predictions_dev[i] = 1
  else:
    predictions_dev[i] = 0
print(classification_report(y_dev, predictions_dev, target_names=target_names))

predictions_test = model.predict(test_data)
for i in range(len(predictions_test)):
  if predictions_test[i] > 0.5:
    predictions_test[i] = 1
  else:
    predictions_test[i] = 0
print(classification_report(y_test, predictions_test, target_names=target_names))
```



```
              precision    recall  f1-score   support

         neg       0.56      0.38      0.45       690
         pos       0.54      0.71      0.61       710

    accuracy                           0.55      1400
   macro avg       0.55      0.54      0.53      1400
weighted avg       0.55      0.55      0.53      1400

13/13 [==============================] - 4s 275ms/step
              precision    recall  f1-score   support

         neg       0.61      0.39      0.48       209
         pos       0.52      0.73      0.61       191

    accuracy                           0.55       400
   macro avg       0.57      0.56      0.54       400
weighted avg       0.57      0.55      0.54       400

7/7 [==============================] - 2s 260ms/step
              precision    recall  f1-score   support

         neg       0.59      0.45      0.51       101
         pos       0.55      0.69      0.61        99

    accuracy                           0.56       200
   macro avg       0.57      0.57      0.56       200
weighted avg       0.57      0.56      0.56       200
```

Τέλος, ζωγραφίζουμε τα διαγράμματα των acuracy και loss συναρτήσει των εποχών. Οι καμπύλες δεν έχουν προλάβει να συγκλίνουν, χρειαζόμασταν πιο πολλές εποχές. Ένα σημαντικό λάθος επίσης που έγινε είναι ότι ενώ ορίστηκε για early stopping ένα διάστημα μη βελτίωσης των 7 εποχών, για την προσαρμογή του learning rate η default τιμή του patience ήταν 10. Προφανώς θα έχουμε πάντα early stopping πριν προλάβει να αλλάξει το learning rate.


model accuracy
train
dev
accuracy
0.54
0.52
0.50
0.48
0
5
10
15
20
25
30
35
40
epoch
model loss
train
dev
loss
0.692
0.690
0.688
0.686
0
5
10
15
20
25
30
35
40
epoch