Aristotle University of Thessaloniki
School of Science
School of Informatics

Report in Recurrent Neural Network trained with tweets-database

Vasileios Asimakopoulos

28 Μαΐου 2023

Περιεχόμενα

Κατάλογος σχημάτων

Code Listings

Κεφάλαιο 1

1.1 Code description

1.1.1 Data Preprocess

Στην εργασία αυτήν χρησιμοποιήθηκε η βιβλιοθήκη tensorflow για την κατασκευή του RNN δικτύου. Το dataset που χρησιμοποιήθηκε ήταν tweets. Ο αριθμός των tweets έφτανε τα 1.600.000 και ήταν πανω σε sentimental analysis, συγκεκριμένα αν είναι θετικό ή αρνητικό το tweet(binary classification). Αρχικά δημιουργήθηκε μία συνάρτηση η οποία φορτώνει το dataset, ανακατεύει τα tweets(αποφεύγουμε έτσι το overfitting), στην συνέχεια διαγράφουμε κάποιες στηλες που δεν χρειάζονται και επειδή έχουμε βάλει data limit κρατάμε τον αριθμό tweets που θα εκπαιδεύσουμε το νευρωνικό. Στην συγκεκριμένα εργασία λόγω χρόνου και έλλειψης υπολογιστικής ισχύς κρατήθηκαν τα 100.000 tweets. Επίσης στην αρχή το νευρωνικό έτρεξε σε όλα τα tweets και δεν φάνηκε κάποια διαφορά με τα υπόλοιπα στο μικρότερο dataset. Μετά το data limitation, χρησιμοποιούμε την βιβλιοθήκη regex ώστε να ορίσουμε τα url και τα username ώστε αργότερα να τα αφαιρέσουμε, διότι δεν μας δίνουν κάποια χρησιμη πληροφορία και είναι θόρυβος για το νευρωνικό, ενώ ταυτόχρονα κατεβάζουμε βιβλιοθήκες της NLTK οι οποίες χρησιμοποιούνται σε μία νέα συνάρτηση την (processtweets), κατα την οποία γίνονται τα εξής: Μετατρέπει το tweet σε πεζά γράμματα. Αφαιρεί τον πρώτο χαρακτήρα (υποθέτοντας ότι πρόκειται για ειδικό χαρακτήρα). Αφαιρεί τις διευθύνσεις URL, και τα ονόματα των χρηστών χρησιμοποιώντας την βιβλιοθήκη regex . Αφαιρεί τα σημεία στίξης. Μετατρέπει το tweet σε λέξεις(σπάει μία πρόταση σε μεμονωμένες λέξεις αλλιώς tokens). Αφαιρεί τα stopwords(πχ i', 'me', 'my') από τις tokenized λέξεις. Με αυτή την διαδικασία αφαιρούμε θόρυβο και εστιάζουμε στην σημαντικές λέξεις που δίνουν νόημα στην πρόταση. Στην συνέχεια μετατρέπουμε τις λέξεις στα λήμματα τους. Αυτό βοηθάει στην ενοποίηση λέξεων με παρόμοιες έννοιες και μειώνει την διάσταση των δεδομένων. Τέλος ενώνουμε τις τελικές λέξεις σε ένα επεξεργασμένο tweet.

Τέλος μετατρέπουμε τα επεξεργασμένα tweets σε ακολουθία αριθμών, και τους βάζουμε padding ώστε όλα να έχουν το μήκος.

Code Listing 1.1: Preprocess of Tweets

```
def preprocess_data(filename, data_limit=None,max_words=0,max_len
    =0):
# Read the CSV file into a DataFrame
data = pd.read_csv(filename, encoding='latin', names=['polarity', '
    id', 'date', 'query', 'user', 'text'])

# Shuffle the data
```

```
data = data.sample(frac=1)

# Limit the data if specified
if data_limit is not None:
data = data[:data_limit]

# Replace the 'polarity' value of 4 with 1
data['polarity'] = data['polarity'].replace(4, 1)

# Remove unnecessary columns from the DataFrame
data.drop(['date', 'query', 'user', 'id'], axis=1, inplace=True)

# Convert the 'text' column to string data type
data['text'] = data['text'].astype('str')

# Download stopwords from NLTK
nltk.download('stopwords')
stopword = set(stopwords.words('english'))

# Download punkt tokenizer from NLTK
nltk.download('punkt')

# Download WordNetLemmatizer from NLTK
nltk.download('wordnet')

# Define regex patterns for URL and username removal
urlPattern = r"((http://)[^␣]*|(https://)[^␣]*|(␣www\.)[^␣]*)"
userPattern = '@[^\s]+'

def process_tweets(tweet):
# Convert the tweet to lowercase
tweet = tweet.lower()
# Remove the first character (assuming it's a special character)
tweet = tweet[1:]
# Remove URLs using regex pattern
tweet = re.sub(urlPattern, '', tweet)
# Remove usernames using regex pattern
tweet = re.sub(userPattern, '', tweet)
# Remove punctuation marks
tweet = tweet.translate(str.maketrans("", "", string.punctuation))
# Tokenize the tweet into words
tokens = word_tokenize(tweet)
# Remove stopwords from the tokenized words
final_tokens = [w for w in tokens if w not in stopword]
# Lemmatize the words
wordLemm = WordNetLemmatizer()
finalwords = []
for w in final_tokens:
if len(w) > 1:
word = wordLemm.lemmatize(w)
finalwords.append(word)
```

```
# Join the final words back into a processed tweet
return ' '.join(finalwords)

# Apply the tweet processing function to the 'text' column and
    store the processed tweets in a new column 'processed_tweets'
data['processed_tweets'] = data['text'].apply(lambda x:
    process_tweets(x))


tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(data.processed_tweets)
sequences = tokenizer.texts_to_sequences(data.processed_tweets)
tweets = pad_sequences(sequences, maxlen=max_len)

X_train, X_test, y_train, y_test = train_test_split(tweets, data.
    polarity.values, test_size=0.35, random_state=101)


# Return the preprocessed data and train/test splits
return X_train, X_test, y_train, y_test

max_words = 5000
max_len = 200

X_train, X_test, y_train, y_test = preprocess_data("/mnt/scratch_b/
    users/v/vaasimak/Documents/models/training.1600000.processed.
    noemoticon.csv",data_limit=100000, max_words = 5000, max_len =
    200)
```

Επίσης χρησιμοποιήθηκε σε ένα νευρωνικό δίκτυο και data augmentation. Συνοπτικά, αυτός ο κώδικας εκτελεί επαύξηση δεδομένων με τυχαία ανταλλαγή λέξεων με τα συνώνυμά τους για τα δεδομένα εκπαίδευσης. Δημιουργεί επαυξημένες εκδόσεις των αρχικών tweets αντικαθιστώντας τα tokens που δεν είναι stopwords και έχουν συνώνυμα με τυχαία επιλεγμένα συνώνυμα. Τα επαυξημένα κείμενα μετατρέπονται σε ακολουθίες ακέραιων δεικτών και συμπληρώνονται για να εξασφαλιστεί ότι έχουν το ίδιο μήκος με τα αρχικά δεδομένα. Αυτή η τεχνική μπορεί να συμβάλει στην αύξηση της ποικιλομορφίας και της γενίκευσης των δεδομένων εκπαίδευσης, βελτιώνοντας ενδεχομένως την απόδοση του μοντέλου.

Code Listing 1.2: Data Augmentation

```
# Data augmentation using NLTK and word swapping
def word_swap_augmentation(tokens, aug_min=1, aug_max=3, stopwords=
    None):
augmented_tokens = tokens.copy()

# Iterate over each token in the tokens list
for i in range(len(tokens)):
token = tokens[i]

# Check if the token is not a stopword and has synonyms
if stopwords is None or token not in stopwords:
synsets = wordnet.synsets(token)
if len(synsets) > 0:
# Choose a random number of words to swap
num_swap = np.random.randint(aug_min, aug_max + 1)
for _ in range(num_swap):
# Get a random synonym from the synsConvolutional Layeets
synonym = np.random.choice(synsets).lemmas()[0].name()
# Replace the token with the synonym
augmented_tokens[i] = synonym

return augmented_tokens

augmented_texts = []
augmented_labels = []
for tweet, label in zip(X_train, y_train):
tokens = [tokenizer.index_word.get(idx, '') for idx in tweet]
augmented_tokens = word_swap_augmentation(tokens, aug_min=1,
    aug_max=3, stopwords=stopword)
augmented_sequence = [tokenizer.word_index.get(word, 0) for word in
     augmented_tokens]
augmented_texts.append(augmented_sequence)
augmented_labels.append(label)

augmented_texts = pad_sequences(augmented_texts, maxlen=max_len)
```

1.1.2 Models

Τα μοντέλα χωρίστηκαν σε δύο βασικές κατηγορίες. Στα simple RNN και στα LSTM. Στην πρώτη περίπτωση εκπαιδεύτηκαν 4 διαφορετικά δίκτυα. Η βασική αρχιτεκτονική παρέμεινε η ίδια (SimpleRNN-Dense-Dense-Dense). Στα πρώτα 2 άλλαξαν οι optimizers(ADAM-SGD) 1.3, 1.4, στο τρίτο προστέθηκε ο l2 σε όλα τα layers με optimizer ADAM 1.5, και στο τελευταίο προστέθηκε BatchNormalization με optimizer SGD **??**

Code Listing 1.3: First RNN

```
adam = tf.keras.optimizers.Adam(learning_rate=10e-4, beta_1
    =0.9, beta_2=0.999, epsilon=1e-7)

# Define the model architecture
model = Sequential([
Embedding(max_words, 128),
SimpleRNN(128, activation="tanh"),
Dropout(0.4),
Dense(64, activation='relu'),
Dropout(0.3),
Dense(32,activation = 'relu'),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.4: Second RNN

```
opt = tf.keras.optimizers.legacy.SGD(learning_rate=10e-4,
    decay=10e-6,momentum = 0.6)


# Define the model architecture
model = Sequential([
Embedding(max_words, 128),
SimpleRNN(128, activation="tanh"),
Dropout(0.4),
Dense(64, activation='relu'),
Dropout(0.3),
Dense(32,activation = 'relu'),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.5: Third RNN

```
adam = tf.keras.optimizers.Adam(learning_rate=10e-4, beta_1
    =0.9, beta_2=0.999, epsilon=1e-7)

# Define the model architecture
model = Sequential([
Embedding(max_words, 128,input_length=max_len),
```

```
SimpleRNN(128, activation="tanh",kernel_regularizer=
    regularizers.l2(0.00001)),
Dropout(0.4),
Dense(64, activation='relu',kernel_regularizer=regularizers
    .l2(0.00001)),
Dropout(0.3),
Dense(32,activation = 'relu',kernel_regularizer=
    regularizers.l2(0.00001)),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.6: Fourth RNN

```
opt = tf.keras.optimizers.legacy.SGD(learning_rate=10e-4,
    decay=10e-6, momentum=0.6)

model = Sequential([
Embedding(max_words, 128, input_length=max_len),
SimpleRNN(128, activation="tanh"),
BatchNormalization(),
Dropout(0.4),
Dense(64, activation='relu'),
BatchNormalization(),
Dropout(0.3),
Dense(32, activation='relu'),
BatchNormalization(),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Στα LSTM εκπαιδεύτηκαν 7 διαφορετικά μοντέλα. Τα 4 πρώτα έχουν την ίδια αρχιτεκτονική με της πρώτης περίπτωσης, για να μπορεί να πραγματοποιηθεί η σύγκριση στο επόμενο κεφάλαιο. Στο 5ο μοντέλο αυξήθηκε το learning rate και το decay στον SGD όπως φαίνεται και στον παρακάτω κώδικα 1.11. Στο 6ο μοντέλο κρατήθηκαν οι παράμετροι του optimizer ίδιοι με του 5ου αλλά μειώθηκε το ποσοστό του dataset που έχει το test set σε 20k απο 35k(100k συνολικό dataset) 1.12. Στο τελευταίο μοντέλο έγινε υλοποίηση του data augmentation με optimizer ADAM και με το test dataset να είναι όπως και στο 6ο στα 20k 1.13.

Code Listing 1.7: First LSTM

```
adam = tf.keras.optimizers.Adam(learning_rate=10e-4, beta_1
    =0.9, beta_2=0.999, epsilon=1e-7)

model = Sequential([
Embedding(max_words, 128,input_length=max_len),
LSTM(128, activation="tanh", dropout=0.4),
Dense(64, activation='relu'),
Dropout(0.3),
Dense(32, activation='relu'),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.8: Second LSTM

```
opt = tf.keras.optimizers.legacy.SGD(learning_rate=10e-4,
    decay=10e-6,momentum = 0.6)

model = Sequential([
Embedding(max_words, 128,input_length=max_len),
LSTM(128, activation="tanh", dropout=0.4),
Dense(64, activation='relu'),
Dropout(0.3),
Dense(32, activation='relu'),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.9: Third LSTM

```
adam = tf.keras.optimizers.Adam(learning_rate=10e-4, beta_1
    =0.9, beta_2=0.999, epsilon=1e-7)

model = Sequential([
Embedding(max_words, 128, input_length=max_len),
LSTM(128, activation="tanh", dropout=0.4,
    kernel_regularizer=regularizers.l2(0.001)),
Dense(64, activation='relu', kernel_regularizer=
    regularizers.l2(0.001)),
```

```
Dropout(0.3),
Dense(32, activation='relu', kernel_regularizer=
   regularizers.l2(0.001)),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.10: Fourth LSTM

```
opt = tf.keras.optimizers.legacy.SGD(learning_rate=10e-4,
   decay=10e-6, momentum=0.6)

model = Sequential([
Embedding(max_words, 128, input_length=max_len),
LSTM(128, activation="tanh", dropout=0.4),
BatchNormalization(),
Dense(64, activation='relu'),
Dropout(0.3),
BatchNormalization(),
Dense(32, activation='relu'),
Dropout(0.2),
BatchNormalization(),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.11: Fifth LSTM

```
opt = tf.keras.optimizers.legacy.SGD(learning_rate=10e-2,
   decay=10e-5, momentum=0.6)

model = Sequential([
Embedding(max_words, 128, input_length=max_len),
LSTM(128, activation="tanh", dropout=0.4),
Dense(64, activation='relu'),
Dropout(0.3),
Dense(32, activation='relu'),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.12: Sixth LSTM

```
X_train, X_test, y_train, y_test = train_test_split(tweets,
    data.polarity.values, test_size=0.2, random_state=101)
```

```
opt = tf.keras.optimizers.legacy.SGD(learning_rate=10e-2,
    decay=10e-5, momentum=0.6)

model = Sequential([
Embedding(max_words, 128, input_length=max_len),
LSTM(128, activation="tanh", dropout=0.4),
Dense(64, activation='relu'),
Dropout(0.3),
Dense(32, activation='relu'),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Code Listing 1.13: Seventh LSTM

With DATA AUGMENTATION

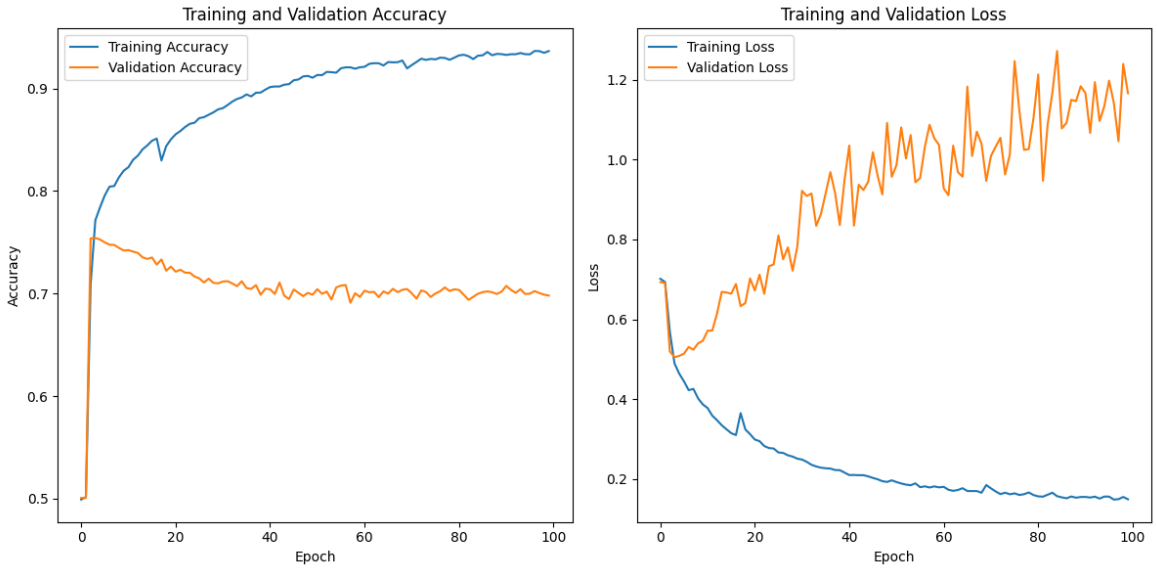
```
adam = tf.keras.optimizers.Adam(learning_rate=10e-4, beta_1
    =0.9, beta_2=0.999, epsilon=1e-7)

model = Sequential([
Embedding(max_words, 128,input_length=max_len),
LSTM(128, activation="tanh", dropout=0.4),
Dense(64, activation='relu'),
Dropout(0.3),
Dense(32, activation='relu'),
Dropout(0.2),
Dense(1, activation='sigmoid')
])
```

Κεφάλαιο 2

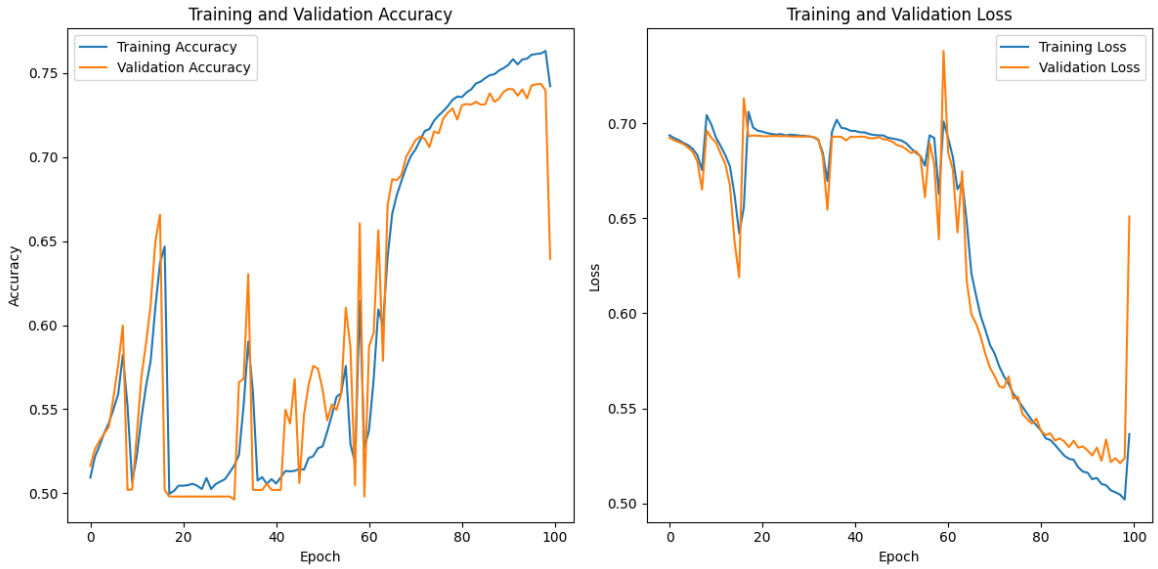
Results of RNN and LSTM

2.1 Σύγκριση RNN μοντέλων

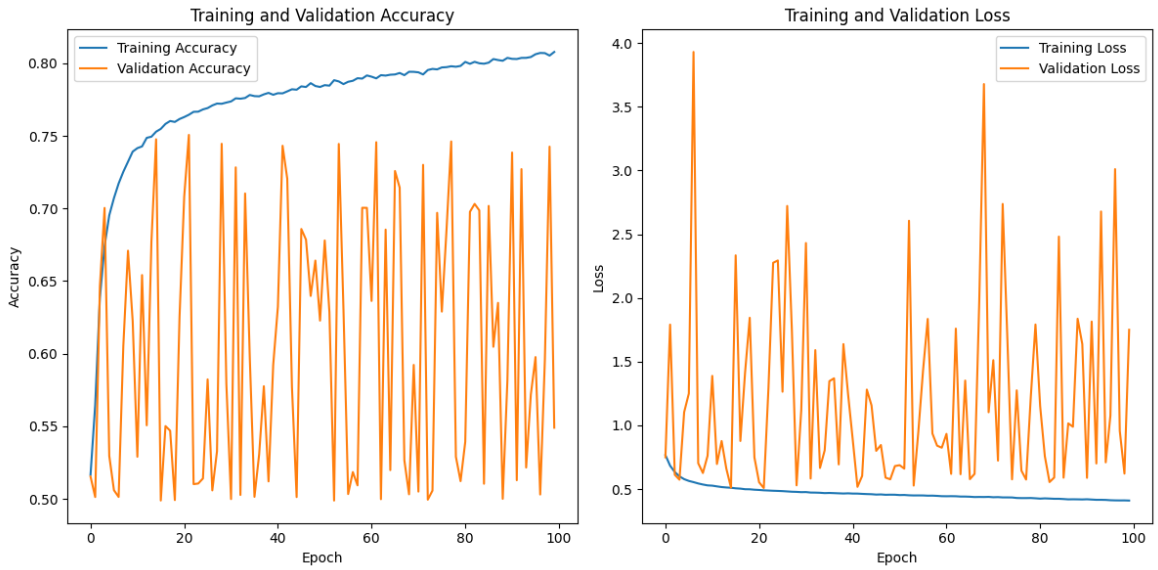
Αρχικά εκπαιδεύτηκαν παράλληλα οι δύο πρώτες περιπτώσεις των RNN που αναφέρθηκαν στο προηγούμενο κεφάλαιο. Από το 1ο διάγραμμα 2.1 παρατηρείται ότι κατα την εκπαίδευση έχουμε overfitting απο την 20η εποχή και μετά, ενώ στο δεύτερο διάγραμμα 2.2 τα RNN δυσκολεύονται να ανεβάσουν το accuracy και αυτό φαίνεται απο τα spikes και στα δύο γραφήματα. Στην τρίτη περίπτωση λόγω του overfitting στην 1η προστέθηκε ο l2 για να σταθεροποιηθεί το accuracy, αλλα παρατηρήθηκε όχι μόνο το αντίθετο αλλά το test accuracy/loss έγινε ασταθές όπως φαίνεται και στο διάγραμμα 2.3. Στην τελευταία που προσθέσαμε το Batch Normalization παρατηρήθηκαν τα ίδια αποτελέσματα με την 1η περίπτωση. Γενικά τα Simple RNN είχαν πολύ κακη επίδοση και ήταν πολυ ασταθή σε όλες τις περιπτώσεις. Ακόμα και σε όλο το dataset που εκπαιδεύτηκε παρατηρούνταν τα ίδια αποτελέσματα.



Σχήμα 2.1: 1ο μοντέλο RNN



Σχήμα 2.2: 2ο μοντέλο RNN

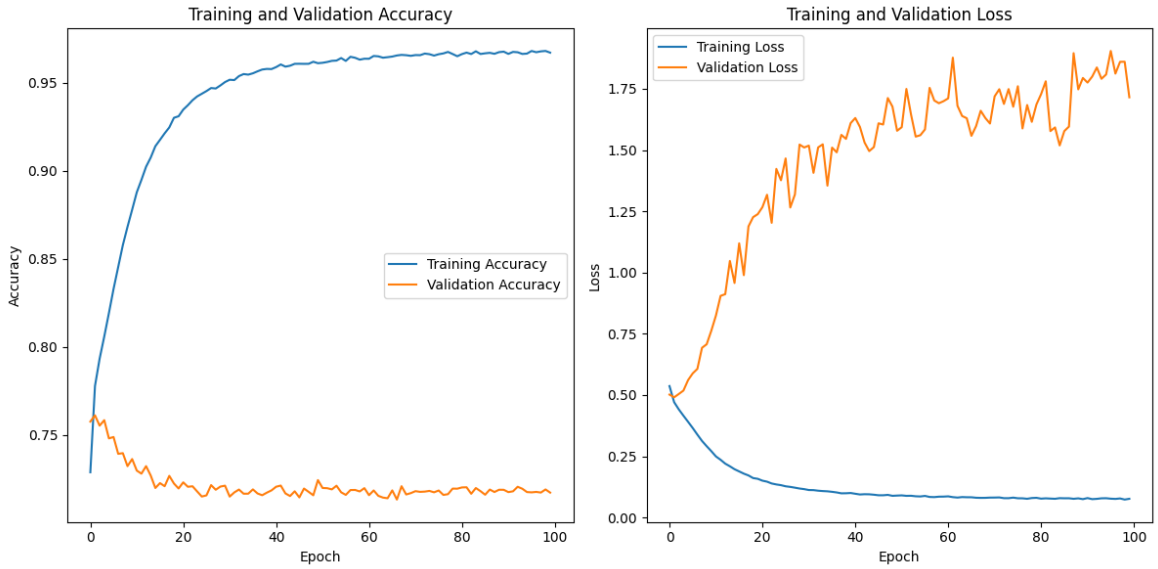


Σχήμα 2.3: 3ο μοντέλο RNN

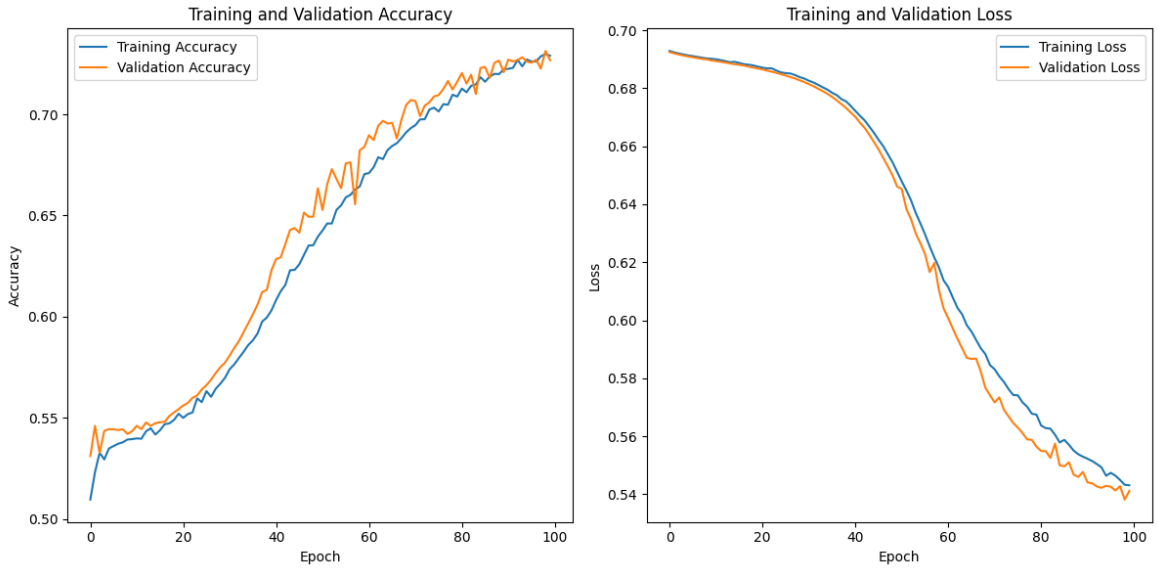
Σχήμα 2.4: 4ο μοντέλο RNN

2.2 Συγκριση LSTM μοντέλων

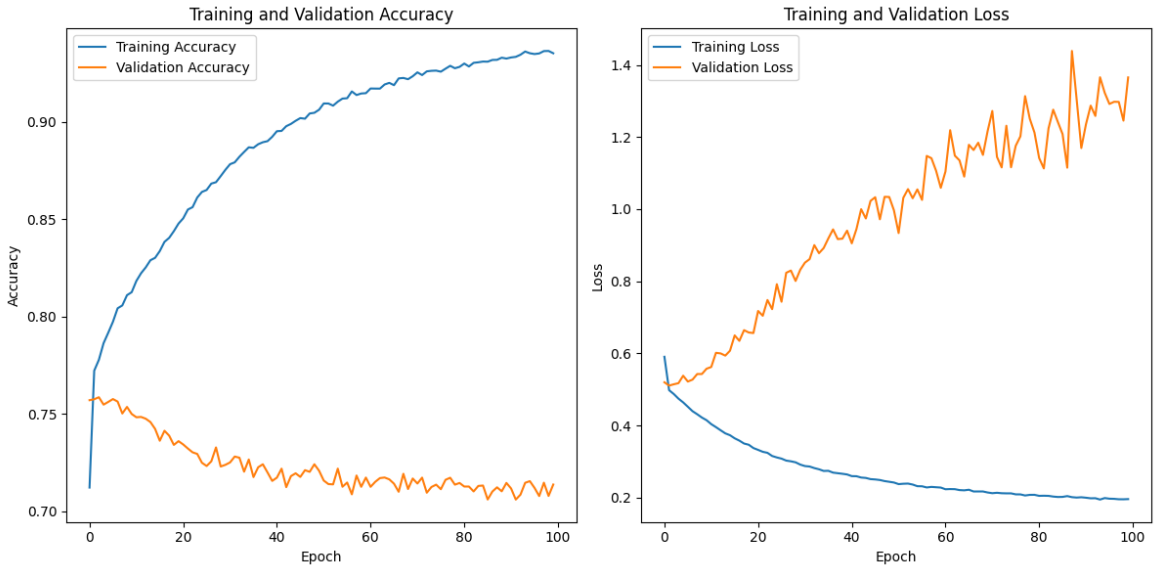
Στην συνένεχεια παρατηρήθηκαν παρόμοια αποτελέσματα στις εκπαιδεύσεις με των RNN, ειδικά στην 1η , 3η , 5η και 6η περίπτωση. Τα αντίστοιχα διαγράμματα τους έχουν σχεδόν την ίδια συμπεριφορά, με αυτήν των RNN. Από την άλλη στην 2η περίπτωση το μοντέλο έχει εκπαιδευτεί αρκετά καλα, παρόλο που το accuracy του ανεβαίνει μέχρι το 75%, όπως φαίνεται και στο διάγραμμα 2.6. Για αυτό τον λόγο έγινε μία προσπάθεια στην 4η περίπτωση με την εισαγωγή του Batch Normalization να αυξηθεί το accuracy αλλά παρατηρληθηκε και εδώ 2.8 ασταθής συμπεριφορά στο test accuracy/loss. Στο 5ο και στο 6ο με την άυξηση του learning rate και την μείωση του test dataset κατέληξε το μοντέλο σε overfitting 2.9. Τέλος με την εισαγωγή του data augmentation το μοντέλο δεν μπορέσε να εκπαιδευτει και παρέμενιε στο 50% και το loss δεν έπεσε ποτέ 2.11 ,



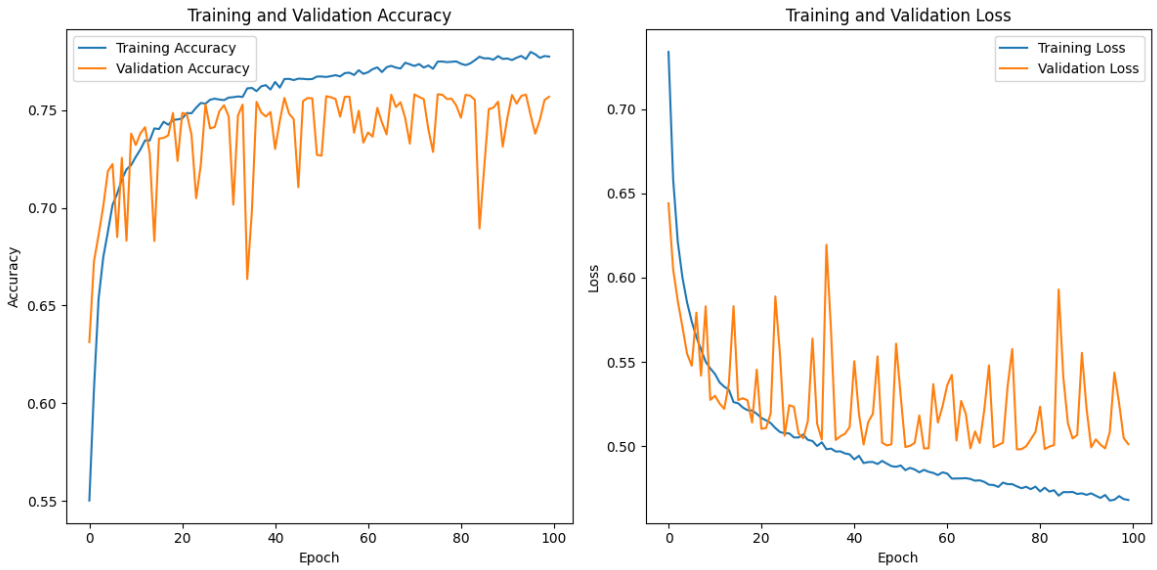
Σχήμα 2.5: 1ο μοντέλο LSTM



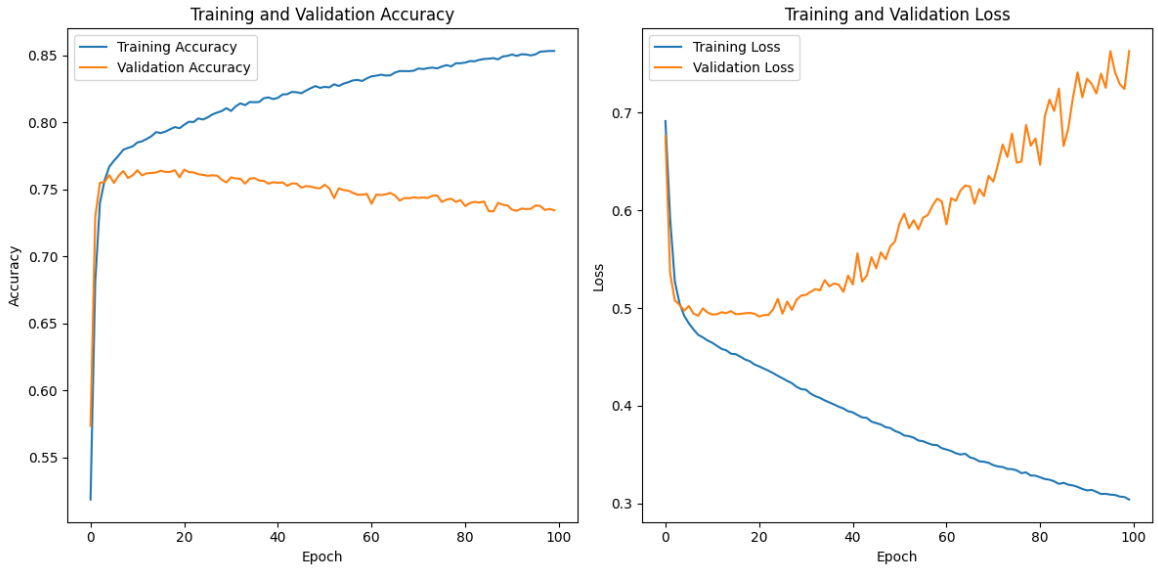
Σχήμα 2.6: 2ο μοντέλο LSTM



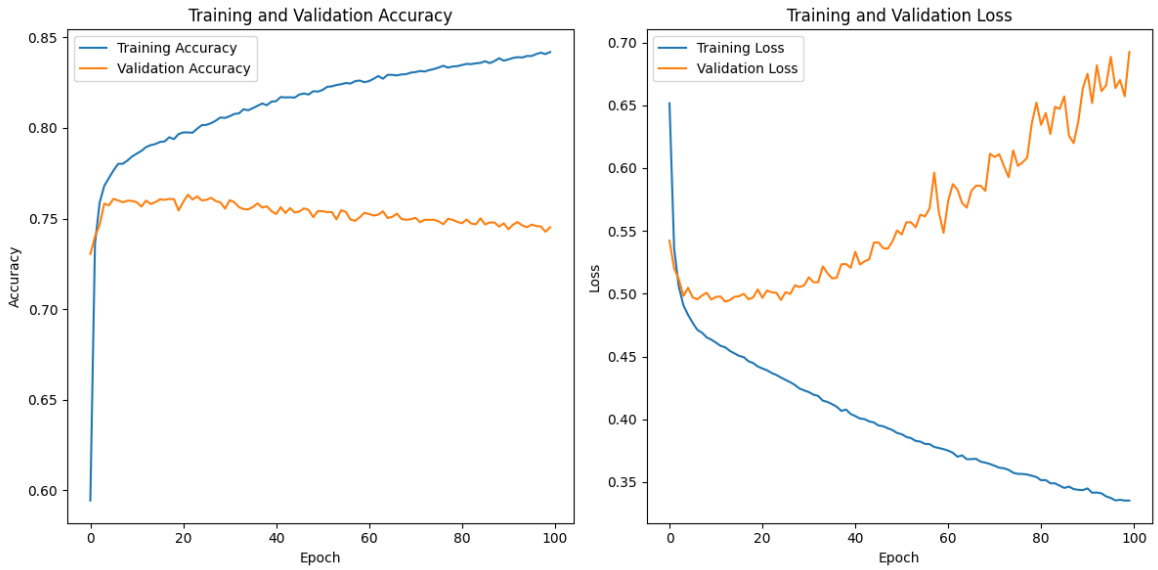
Σχήμα 2.7: 3ο μοντέλο LSTM



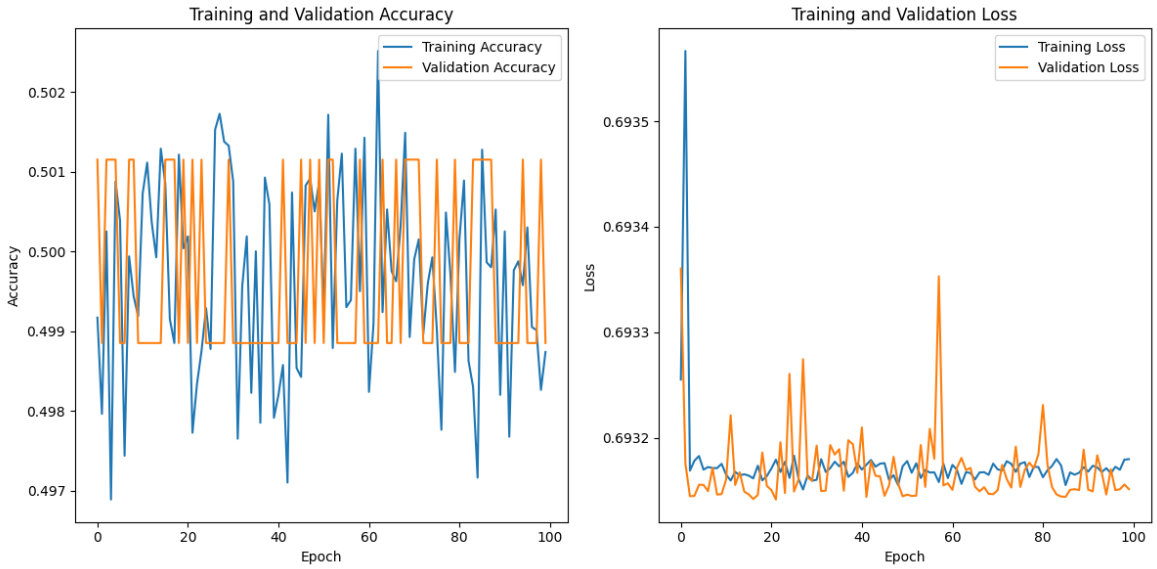
Σχήμα 2.8: 4ο μοντέλο LSTM



Σχήμα 2.9: 5ο μοντέλο LSTM



Σχήμα 2.10: 6ο μοντέλο LSTM



Σχήμα 2.11: 6ο μοντέλο LSTM

2.3 LSTM vs RNN

Παρόλο που τα περισσότερα μοντέλα και στις 2 περιπτώσεις ήταν ασταθή τα LSTM είχαν καλύτερα αποτελέσματα κατι που περιμέναμε λόγω της αρχιτεκτονικής των LSTM. Τέλος τα spikes που παρατηρούνται στα περισσότερα μοντέλα υποδηλώνουν exploding gradients το οποίο μπορεί να οδηγήσει σε αριθμητική αστάθεια και να εμποδίσει την ικανότητα του μοντέλου να μάθει αποτελεσματικά, ακόμα και άν αυτό είναι πιο σταθερό αρχιτεκτονικά (LSTM).