UNIVERSITY OF PISA

Human Language Technologies

Year 2021/2022

Sentiment Analysis of Arabic Tweets

Ahmed Salah Tawfik Ibrahim

Morteza Arezoumandan

Abstract

In this project, we build multiple deep learning models for the task of classifying tweets written in the Modern Standard Arabic language into one of 3 classes of sentiments (positive, negative and neutral). In doing so, we make use of the Farasa stemmer, word embeddings, neural networks (fully connected, convolutional and LSTM), grid search and the BERT transformer. In the end, we compare their performance in terms of accuracy, precision, recall and the F1 measure.

1. Introduction

Sentiment Analysis is a very important task nowadays as everything is being discussed online on social media platforms. However, with the overwhelming growth of online media content, one cannot analyze the sentiments by reading everything. Instead, reliable automatic tools are needed in order to perform this task so that other tools can aggregate the results into useful metrics that can be used by content creators to understand the general sentiment regarding what they post. Nevertheless, this task faces a huge challenge which is the language. Granted, we cannot have a unified tool that can analyze the sentiment of any given text written in any language. Instead, these tools are usually trained on understanding the sentiments in one language, or maybe a family of languages sharing some characteristics. Some languages, however, are more challenging than others due to their unfamiliar constructs; and hence, they are interesting to investigate. Therefore, in this project, we will be comparing different deep learning models in terms of their capability to analyze sentiments of tweets written in the Arabic language.

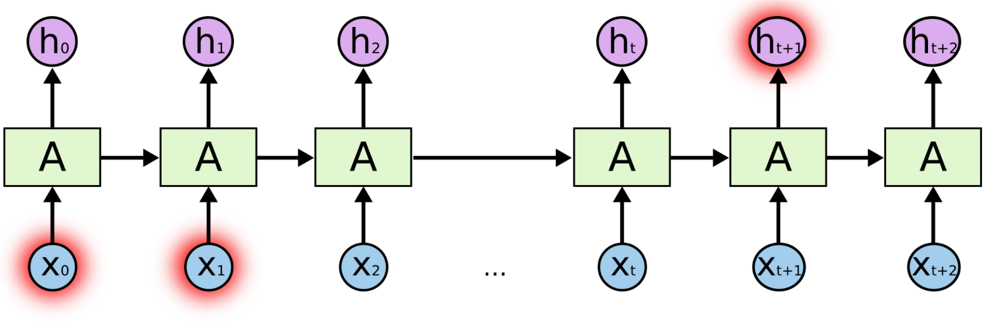
2. Literature Review

Sentiment analysis of Tweets is a particularly challenging task because of the informal writing style that is usually used on Twitter. Most of the state-of-the-art language models are based on deep learning networks. In previous attempts, sentiment analysis was tackled using different approaches such as RNNs, LSTMs, and Transformers in several works.

2.1 Recurrent Neural Networks

An RNN is a class of artificial neural networks that can make use of information in long sequences in a way that they perform the same task for every element of a sequence and captures information about what has been calculated so far. This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They’re the natural architecture of a neural network to use for such data. An obvious example could be our task, language models.

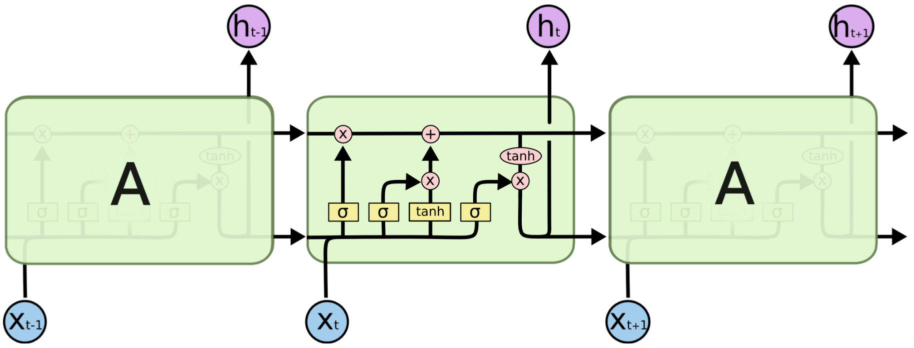
A known problem for RNNs is the difficulty to train them. If a sequence is long enough, they’ll have a hard time carrying information from earlier time steps to later ones.



What could be a solution to the problem explained above is using a variant of the regular RNN called LSTM.

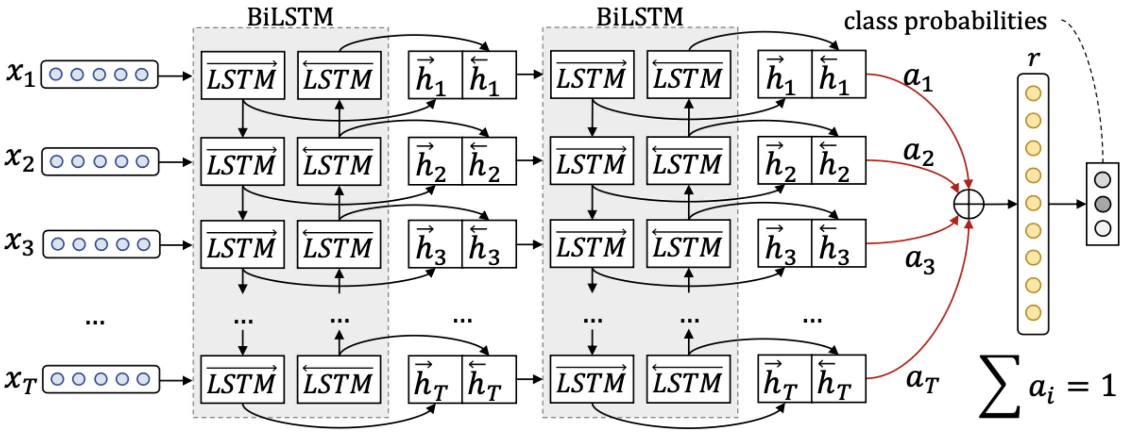
2.2 LSTM Network

The Long Short-Term Memory (LSTM) network introduces a gating mechanism to guarantee proper gradient propagation through the network. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. LSTMs also have this chain-like structure like RNNs, but the repeating module has a different structure. Instead of having a single neural network layer, the repeating module in an LSTM contains four interacting layers.



The core concept of LSTMs are the cell state, and its various gates. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. As the cell state goes forward, information gets added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

One of the approaches that exploited LSTMs and achieved good results on sentiment analysis was DataStories at SemEval-2017 Task 4 [1], in which they proposed a 2-layer bidirectional LSTM with an attention mechanism. The model consists of an Embedding layer in which tweets are embedded using a pre-trained word embedding model. An LSTM takes as inputs the each tweet word and creates the word annotation where each element is the hidden state of LSTM at a certain time step, summarizing all the information of the sequence up to the current word. Since not all the words contribute equally to the expression of the sentiment in a message, an attention mechanism is used to find the relative importance of each word. The attention mechanism gives a weight to each word annotation. So the fixed representation of the whole message as the weighted sum of all the word annotation is computed and used as feature vector for classification and passed to the final fully connected softmax layer which outputs a probability distribution over all classes. The following figure depicts the described model.


BiLSTM
BiLSTM
class probabilities
x_1
x_2
x_3
x_T
$\overrightarrow{LSTM}$
$\overleftarrow{LSTM}$
$\overrightarrow{h}_1$ $\overleftarrow{h}_1$
$\overrightarrow{h}_2$ $\overleftarrow{h}_2$
$\overrightarrow{h}_3$ $\overleftarrow{h}_3$
$\overrightarrow{h}_T$ $\overleftarrow{h}_T$
a_1
a_2
a_3
a_T
r
$\sum a_i = 1$

They further use regularization by adding Guassian noise at the embedding layer, dropout to randomly turn off neurons in the network, and L2 regularization penalty to discourage large weights. The model and its details inspired us along the way while building our deep neural network models.

2.3 Transformers

Like LSTM, Transformer is an architecture for transforming one sequence into another one with the help of two parts (Encoder and Decoder), but it differs from the previously described sequence-to-sequence models because it does not imply any Recurrent Networks. The model proposed in *Attention Is All You Need* [2] and the original architecture is shown below.

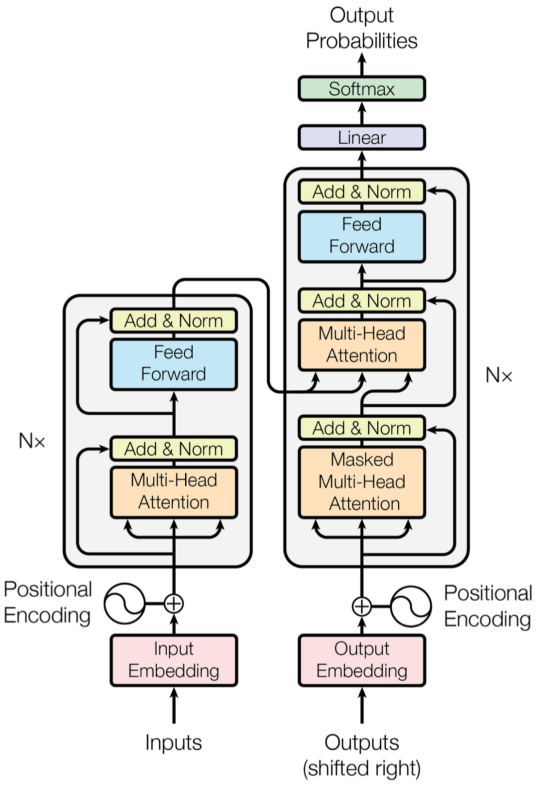


Figure 1: The Transformer - model architecture.

The original Transformer model uses an encoder-decoder architecture. Both encoders and decoders are composed of modules that can be stacked on top of each other several times.The modules mainly consist of Multi-head Attention and Feed Forward Layers. Input and output must be embedded to be able to be processed. The positions are included in the embeddings of each word since there is no recurrent network to remember sequences.

The additional training parallelization allows training on larger datasets. This led to development of pre-trained systems such as BERT.

2.4 BERT

Another state-of-the-art approach that has recently been used for language models is Bidirectional Encoder Representations from Transformers (BERT) [3]. BERT’s key technical innovation is applying the bidirectional training of Transformer to language modeling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper’s results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible.

BERT can be fine-tuned, or used directly as a feature extractor for various textual tasks. One of the variants of BERT that is trained on Arabic text is introduced in KUISAIL at SemEval-2020 Task 12 [4]. ArabicBERT is a BERT language model that is trained using masked language modeling with whole word masking [3].

3. Dataset

The dataset that we used for this project is the same dataset provided for the semeval task4. It is a dataset of tweets written in the modern standard Arabic that are labeled according to the sentiments. The dataset contains 671 tweets and they are classified into one of 3 classes; 0 being neutral, 1 being positive and -1 being negative.

In order to prepare this dataset for our model, we had to perform some preprocessing steps. We removed all the url links, all the mentions, all the English characters and the useless special characters [#,@,&,>,<]. Then, we unified the numbers into one numbering system; namely, the Arabic system [0,1,2,...,9]. Then, we removed the emojis as we wanted to focus on building a model focused only on text. Finally, we had to perform lemmatization and tokenization.

The Arabic language is characterized by certain features that must be considered when processing any text written in Arabic. In particular, a single word is possibly composed by stacking up a main word, which is a derivative of a root word, and some clitics. Those clitics are usually nothing but grammatical references to objects or subjects; that is, they are usually a syntactic feature rather than a semantic one. Therefore, for a task like sentiment analysis, what matters most is capturing the semantic meaning contained within a sentence which would be encoded in the roots of the main words in that sentence. Hence, reducing a sentence into a sequence of roots is a logical direction for analyzing sentiments in an Arabic text. The NLTK library offers a stemmer for the Arabic language; however, the roots produced by this stemmer are somehow generic (3 or 4 letter roots) and they do not really capture the specific meaning of words as they are just the grammatical roots of the words. That's why another stemmer that generates meaningful roots was needed. That's where the Farasa stemmer comes into place as it does not generate the grammatical roots; instead, it generates roots that capture the meaning of a given word. To make this difference clear, we consider the following example.

If we input the word “اعلامي” which means journalist to the NLTK stemmer, it would output the root “علم” which is the 3-letter grammatical root. The problem with that root is that it does not really tell anything about journalism. In fact, this root is shared with any word containing the 3 letters “م“ ,”ل“ ,”ع” like the word “علم” which means flag or the word “معلم” which means teacher. Therefore, this stemming mechanism is not helpful. On the other hand, when we input it to the Farasa stemmer, it will give as output the root “اعلام” which means journalism. This is more useful to the task at hand as it captures the meaning in a reduced way without the additional clitics that would make sense only grammatically but would make no difference from a content point of view. Therefore, we decided to use the Farasa stemmer when we pre-process the tweets.

After performing the lemmatization step, we had to run some statistical queries in order to better understand our dataset. We found out that the vocabulary size of our dataset is

3046 words. Furthermore, the longest tweet is composed of 41 words. These numbers were needed in order to perform the tokenization step. In particular, we converted the tweets into a sequence of numbers ranging from 1 to 3046. Then, we padded all tweets with the special token 0 so that they all share the same length of 41. The padding was added to the end of each tweet whose length was less than 41. The table below summarizes these numbers.

Total Number of Tweets	Vocabulary Size	Maximum Tweet Length
671	3046	41

The last step was to split the dataset into tuning, training, validation and test sets. We applied a stratified split to keep the distribution of sentiments the same across the splits. We performed a split of 3% for the tuning set and 97% for the rest which was divided as follows: 80% for the training set, 10% for the validation set and 10% for the test set

4. Architectures

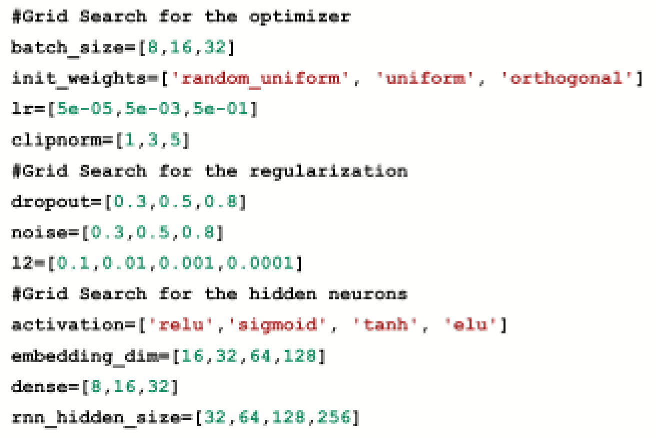
We have decided to explore various architectures in order to choose the best among them. Mainly, we were trying to find a tradeoff between the complexity and the accuracy. Therefore, we experimented with 4 architectures:

1) A very simple model with an embedding layer and one LSTM layer and a number of fully connected layers.
2) A deeper model with more LSTM layers followed by fully connected layers.
3) A number of LSTM layers followed by CNN layers and then a number of fully connected layers.
4) A transformer model followed by fully connected layers.

To better train these architectures, we ran a grid search over some hyperparameters that were important for each of them. We used the tuning set in order to obtain the values of the hyperparameters of each model and then we used these values in creating the model that was trained on the training set.

4.1 Hyperparameters of the Simple Model

For this model, we ran the grid search over a number of hyperparameters concerning different parts of the model. Namely, we optimized the hyperparameters of the Adam optimizer used (learning rate and gradient norm threshold), the weight initializer and the batch size to use. Then, using the best values for the previous ones, we ran another grid search for the hyperparameters related to regularization(l2 loss, noise rate and the dropout rate). Then finally, we used the best values obtained from the previous search to run another search for the hyperparameters of the layers(number of LSTM units, number of hidden neurons in the dense layer and the activation function to use and finally the embedding dimension). The reason why we ran consecutive searches instead of one global search is the impossibility of reaching a result due to the huge search space with respect to the limited computational resources we had. The search space was defined as shown below:



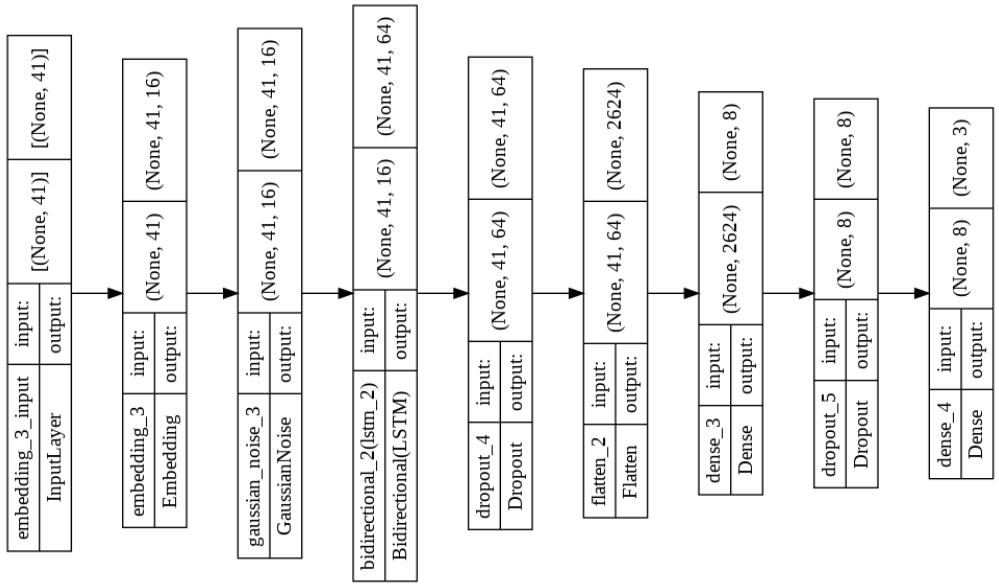
```
#Grid Search for the optimizer
batch_size=[8,16,32]
init_weights=['random_uniform', 'uniform', 'orthogonal']
lr=[5e-05,5e-03,5e-01]
clipnorm=[1,3,5]
#Grid Search for the regularization
dropout=[0.3,0.5,0.8]
noise=[0.3,0.5,0.8]
l2=[0.1,0.01,0.001,0.0001]
#Grid Search for the hidden neurons
activation=['relu','sigmoid', 'tanh', 'elu']
embedding_dim=[16,32,64,128]
dense=[8,16,32]
rnn_hidden_size=[32,64,128,256]
```

The results of the grid search using the tuning set were as follows:

Hyper-parameter	Batch Size	Gradient Norm	Weight Initializer	Learning Rate	Dropout Rate	Noise Rate	L2 Regularization
Value	16	3	Uniform	5e-05	0.5	0.5	0.1

Hyper-parameter	Dense Neurons	LSTM Units	Embedding Dimension	Activation Function
Value	8	32	16	Sigmoid

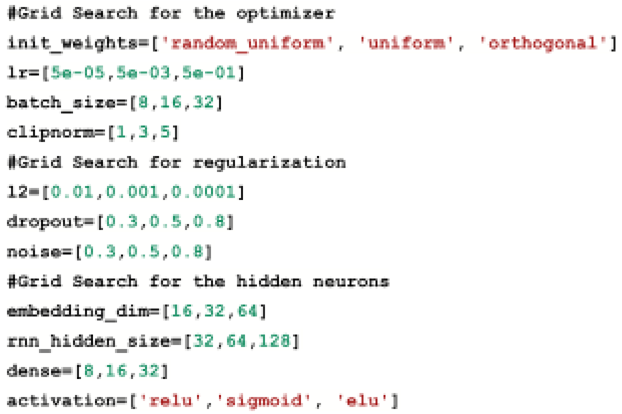
The final model to be trained is described by the following figure:



As shown, this model is composed of 82,323 trainable parameters.

4.2 Hyperparameters of the Deep LSTM Model

For this model, we had an embedding layer followed by gaussian noise then two pairs of bidirectional LSTM and dropout layers followed by a number of fully connected layers leading to the final classifier. Therefore, we had a set of hyperparameters to optimize. The search space was define as shown below:



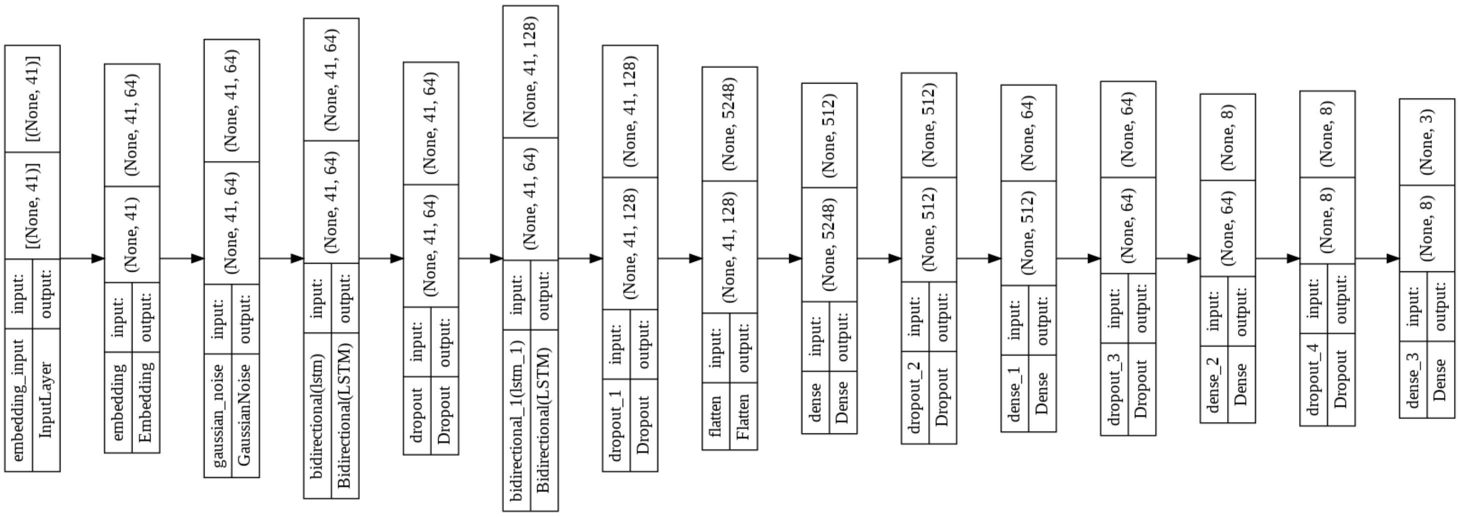
```
#Grid Search for the optimizer
init_weights=['random_uniform', 'uniform', 'orthogonal']
lr=[5e-05,5e-03,5e-01]
batch_size=[8,16,32]
clipnorm=[1,3,5]
#Grid Search for regularization
l2=[0.01,0.001,0.0001]
dropout=[0.3,0.5,0.8]
noise=[0.3,0.5,0.8]
#Grid Search for the hidden neurons
embedding_dim=[16,32,64]
rnn_hidden_size=[32,64,128]
dense=[8,16,32]
activation=['relu','sigmoid', 'elu']
```

After running the consecutive partial grid searches, we obtained the following results:

Hyper-parameter	Batch Size	Gradient Norm	Weight Initializer	Learning Rate	Dropout Rate	Noise Rate	L2 Regularization
Value	8	1	Orthogonal	5e-05	0.3	0.8	0.0001

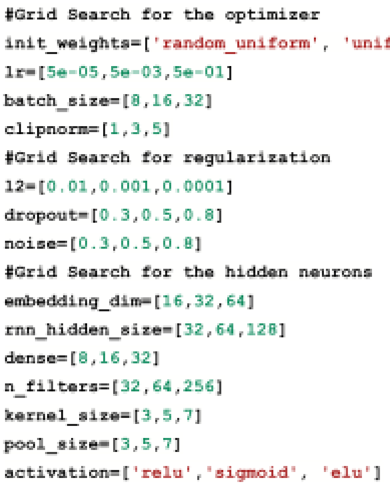
Hyper-parameter	Dense Neurons	LSTM Units	Embedding Dimension	Activation Function
Value	8	32	64	Sigmoid

The following figure shows the complete architecture of the model that will eventually be trained. It has a total of 3,006,755 trainable parameters.



4.3 Hyperparameters of the Deep LSTM+CNN Model

The aim of this model was to combine the capabilities of LSTMs of learning sequences with the capabilities of CNNs of learning features irrespective of where they occur in the sentence. As before, we had an embedding layer followed by gaussian noise then two pairs of bidirectional LSTM and dropout layers. Then, the output of this is passed to a convolution and max pooling layer that is then followed by a number of fully connected layers leading to the final classifier. Therefore, we had a set of hyperparameters to optimize. The search space was define as shown below:



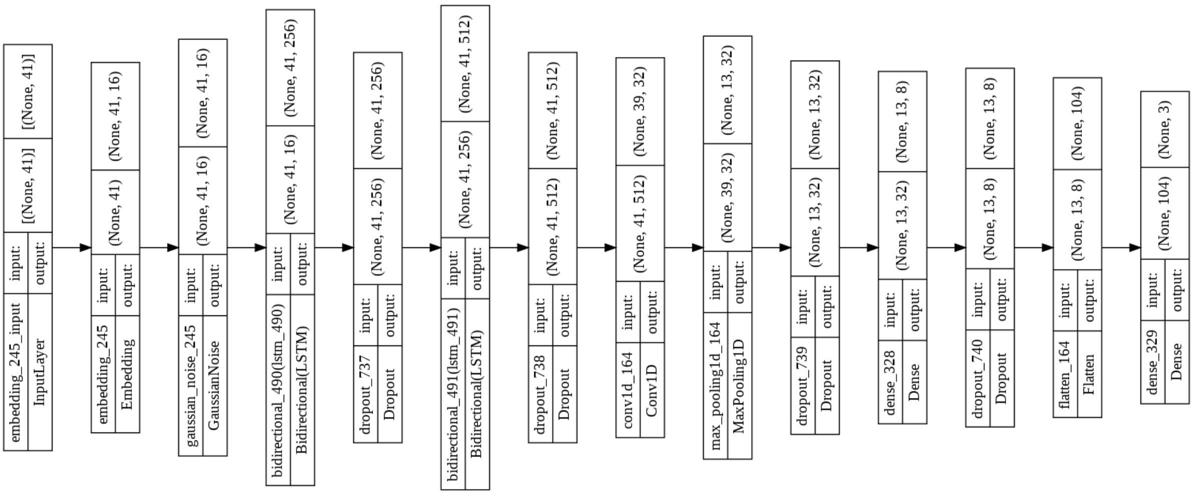
```
#Grid Search for the optimizer
init_weights=['random_uniform', 'uniform', 'orthogonal']
lr=[5e-05,5e-03,5e-01]
batch_size=[8,16,32]
clipnorm=[1,3,5]
#Grid Search for regularization
l2=[0.01,0.001,0.0001]
dropout=[0.3,0.5,0.8]
noise=[0.3,0.5,0.8]
#Grid Search for the hidden neurons
embedding_dim=[16,32,64]
rnn_hidden_size=[32,64,128]
dense=[8,16,32]
n_filters=[32,64,256]
kernel_size=[3,5,7]
pool_size=[3,5,7]
activation=['relu','sigmoid', 'elu']
```

After running the consecutive partial grid searches, we obtained the following results:

Hyper-parameter	Batch Size	Gradient Norm	Weight Initializer	Learning Rate	Dropout Rate	Noise Rate	L2 Regularization
Value	16	1	Orthogonal	5e-05	0.3	0.8	0.01

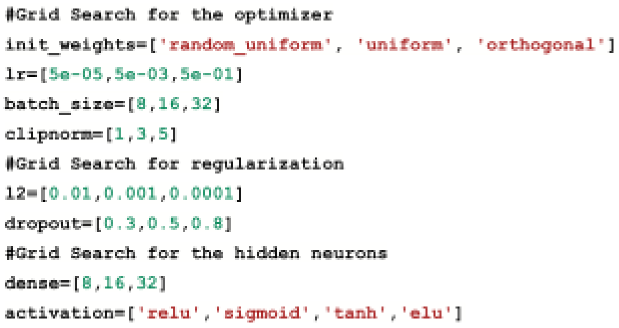
Hyper-parameter	Dense Neurons	LSTM Units	Embedding Dimension	Conv Filters	Filter Size	Pool Size	Activation Function
Value	8	128	16	32	3	3	Relu

The following figure shows the complete architecture of the model that will eventually be trained. It has a total of 1,297,619 trainable parameters.



4.4 Hyperparameters of the Transformer Model

In this model, we used the BERT transformer for Arabic. More specifically, we used asafaya bert base [4]. Basically, we fed our tweets to the transformer in order to generate an embedding per sentence and then we passed this to a fully connected layer followed by the classifying layer that would distinguish the tweet according to the sentiment. There were two approaches; the first one was to use the embeddings as they are, while the second one was finetuning those embeddings. We decided to finetune the sentence embeddings because the transformer was not specific for our task. We had some hyperparameters to optimize in order to make the best use of this model. They were the optimizer hyperparameters and the regularization ones in addition to the dense layer. The search space is shown in the following figure:



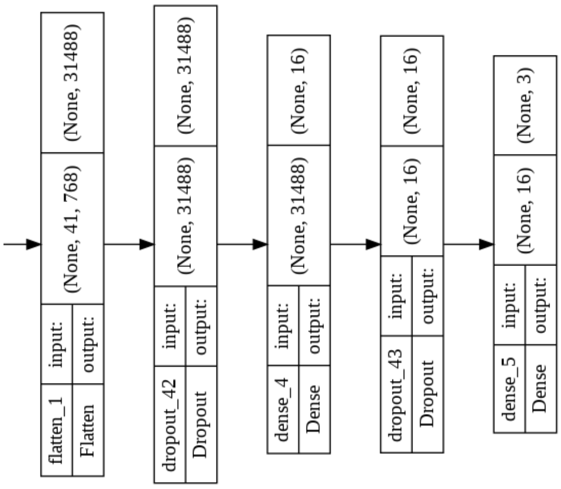
```
#Grid Search for the optimizer
init_weights=['random_uniform', 'uniform', 'orthogonal']
lr=[5e-05,5e-03,5e-01]
batch_size=[8,16,32]
clipnorm=[1,3,5]
#Grid Search for regularization
l2=[0.01,0.001,0.0001]
dropout=[0.3,0.5,0.8]
#Grid Search for the hidden neurons
dense=[8,16,32]
activation=['relu','sigmoid','tanh','elu']
```

The results of the grid search over that hyperparameter space were as shown in the following tables:

Hyper-parameter	Batch Size	Gradient Norm	Weight Initializer	Learning Rate	Dropout Rate	L2 Regularization
Value	8	1	RandomUniform	5e-05	0.3	0.01

Hyper-parameter	Dense Neurons	Transformer Output	Activation Function
Value	16	0	Relu

The following figure shows the architecture of the model, after the transformer layer, that will eventually be trained. It has a total of 111,121,219 trainable parameters.

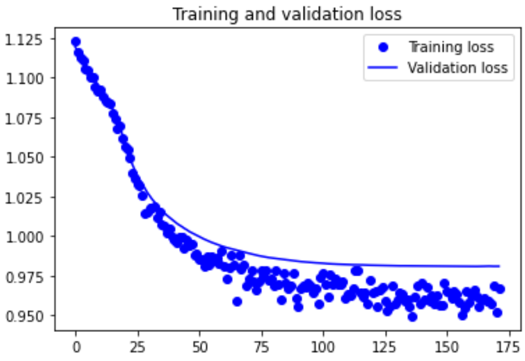


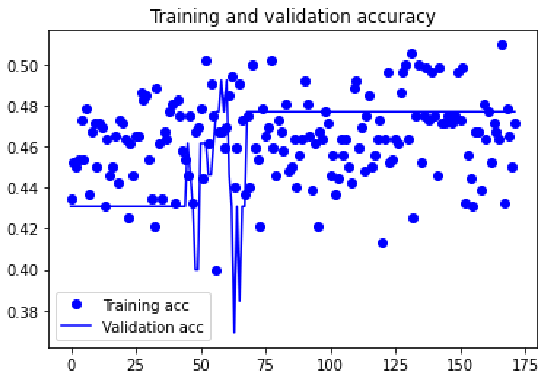
5. Experiments and Results

In this section, we describe the experiments that we performed in order to reach our final model. It is worth noting that the loss function used was the categorical cross entropy loss and the metric used for evaluation was accuracy. Furthermore, we set the maximum number of epochs to 1000 keeping in mind that we employed an early stopping mechanism if the validation loss does not improve for a number of epochs.

5.1: Experiment 1 - Training the Simple Model:

We trained the model described in section 4.1 and we obtained the following learning curves:



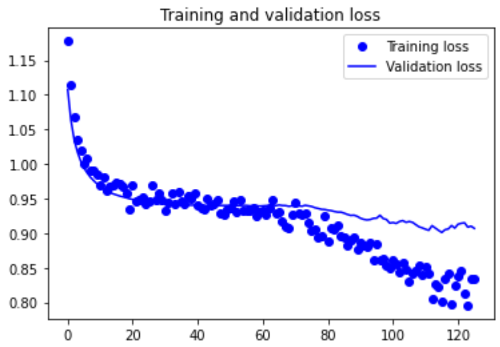


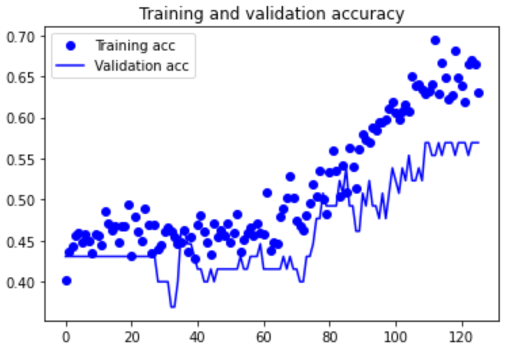
The model kept improving in terms of loss until it reached a plateau after about 165 epochs. We restored the best accuracy scores obtained during the training and the results were as follows:

Training Accuracy	Validation Accuracy	Test Accuracy
47.69%	49.23%	47.69%

5.2: Experiment 2 - More complex model

After training the model described in section 4.2, we got the following learning curves:



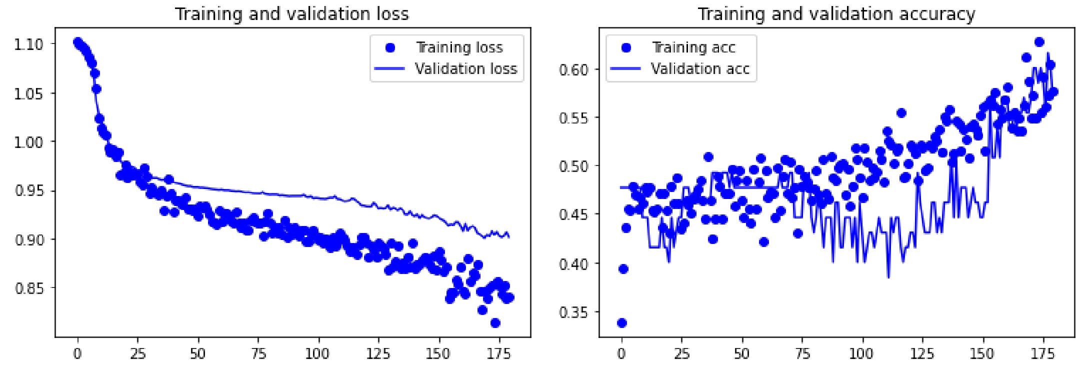


It achieved a better accuracy in a lower number of epochs (around 120 epochs) compared to model 4.1 before reaching a plateau in training. The accuracies obtained by this model were as shown in the following table:

Training Accuracy	Validation Accuracy	Test Accuracy
84.04%	56.92%	58.46%

5.3: Experiment 3 - RNN and CNN

After training the model described in section 4.3, we got the following learning curves:

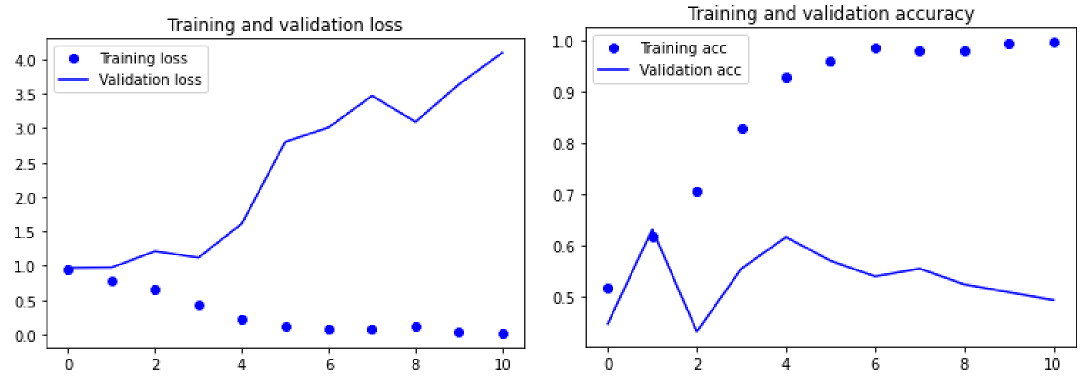


Its performance was not as good as the model in 4.2 and it is better than model 4.1 in terms of validation accuracy but they both share a similar test accuracy. Furthermore, it took a long time to train this model. The accuracies achieved are reported in the following table:

Training Accuracy	Validation Accuracy	Test Accuracy
71.53%	61.54%	47.69%

5.4: Experiment 4 -Transformer and Classifier

After training the model described in section 4.4, we got the following learning curves:



As shown in the loss curve, the validation loss kept increasing for 10 consecutive epochs, which stopped the training as it has not improved for 10 epochs. It is also worth noting that due to the larger number of trainable parameters, this model took more time to finish training compared to the previous ones. Despite this, it achieved the accuracies shown in the following table which are generally better than the previous ones considering the number of epochs:

Training Accuracy	Validation Accuracy	Test Accuracy
75.58%	63.08%	58.46%

It is worth noting, however, that this behavior could be due to overfitting as the training loss kept on decreasing until it reached a very low value towards the end of the training. This suggests that maybe we need to employ more techniques to overcome overfitting.

6. Analysis of Results

6.1: Evaluation Measures

Inspired by the measures that are used in [5], we have used the same evaluation metrics to be able to compare the results with the scores obtained in the challenge by the top performers, however we have also used additional metrics to be able to better evaluate our models.

For the comparison we have done among our models, we used average recall, average precision, average F1. Note that all of the evaluation measures are *"macro-averaged"*. We further include the number of parameters in each model to also compare the models' complexities.

For comparison with the systems competed in the challenge, the primary measure is *macro-average recall*, which is recall averaged across all classes. The second measure is *macro-average F1* calculated over only positive and negative classes. The third is simply the *accuracy* of the model.

6.2: Results Comparison

In the following table, we summarize the results obtained from all the models described above in order to be able to draw conclusions and to provide suggestions for enhancements.

Model	**#Parameters**	**Accuracy**	**Avg-Prec**	**Avg-Rec**	**Avg-F1**
4.1	82,323	47.69%	0.309524	0.340686	0.321368
4.2	3,006,755	58.46%	0.327652	0.370915	0.345771
4.3	1,297,619	47.69%	0.263333	0.302288	0.270127
4.4	111,121,219	58.46%	0.397436	0.458333	0.411111

As shown in the table, in terms of test accuracy only, models 4.1 and 4.3 are equivalent while models 4.2 and 4.4 are equivalent. But if we consider the complexity of each model, models 4.1 and 4.2 would prevail as they are more simple than their counterparts. However, this does not mean they actually perform better. The accuracy measure is not enough to show which model is better. That's why it was important to consider the precision and recall metrics and their aggregation in the F1 measure. As we can see, in terms of F1 measure, the models 4.4 and 4.2 would prevail. The question, however, is why are the scores so low? To answer this question, we checked the confusion matrix of each model and we discovered that none of the models is able to recognize the "Negative" class. This is due to the fact that the dataset is unbalanced. The percentage of negative samples in the dataset is about 8% which is a very low percentage for the model to learn. Therefore, we tried to make use of oversampling by repeating negative examples randomly until we achieve a balance in the class distribution; however, this approach resulted in a worse accuracy after the first trial, so we did not proceed with it.

Therefore, due to the complexity of the Arabic language and the small size of the dataset, a larger dataset would be needed where each class would have a sufficient number of representative examples to it.

6.3: Enhancing the Results

In order to enhance our results, we replicated the same process described in the previous sections; however, we have made slight improvements. First of all, instead of training word embeddings from scratch, we decided to use pre-trained word embeddings that are going to be finetuned during the training. The word embeddings that we used was AraVec [6], which provides embedding vectors of various sizes. We used their uni-gram CBOW word embeddings of size 100 in initializing the embedding layer. Furthermore, we used an additional dataset in order to train the models. This dataset, which is called ASTD (Arabic Sentiment Tweets Dataset) contains more than 7,000 tweets that are balanced with 4 different polarities. However, that dataset includes more than 7,000 tweets if the balancing is not considered. For our purposes, we just took a subset of about 2400 examples considering only the 3 sentiment classes of semeval. We made sure to select an equal amount of examples for each sentiment class in order to avoid the problem we faced before. We stemmed this dataset using the Farasa stemmer as we did with the SemEval dataset and then we combined both datasets to obtain a larger dataset. The final dataset contained 9,110 words and we truncated the length of each tweet to the maximum length we found in SemEval; i.e., 41. Below is a summary of the final dataset:

Total Number of Tweets	Vocabulary Size	Maximum Tweet Length
3068	9110	41

Also, the table below shows the number of examples we had for each class:

Positive	Neutral	Negative
1106	1109	853

After integrating the new dataset, we again ran a hyperparameter tuning procedure in order to get the best combination of hyperparameters and we re-trained the architectures 4.2, 4.3 and 4.4 in order to check if there was an improvement. We computed the results for a combined test set and also for the original test set of SemEval, the one whose results are reported in 6.2. The tables below show the new results for the combined test set and the original test set respectively:

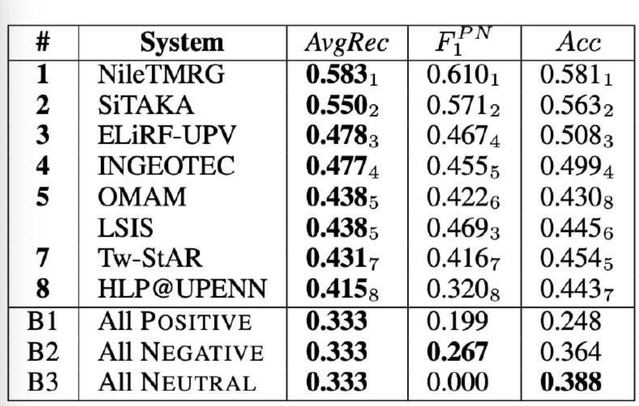
Combined Test Set Results:

Model	#Parameters	Accuracy	Avg-Prec	Avg-Rec	Avg-F1
4.2	3,345,439	52.08%	53%	52.02%	52.30%
4.3	3,407,063	52.45%	55.65%	52.32%	52.79%
4.4	110,753,283	59.43%	58.73%	59.98%	58.59%

Original Test Set Results:

Model	#Parameters	Accuracy	Avg-Prec	Avg-Rec	Avg-F1
4.2	3,345,439	61.53%	58.43%	57.50%	57.60%
4.3	3,407,063	63.08%	64.27%	53.48%	56.15%
4.4	110,753,283	55.38%	52.94%	54.81%	51.90%

As we hypothesized, the results did indeed improve over the results we obtained from our original experiments. In fact, our results are close to the results obtained during the competition as shown in the following image taken from the official competition paper [5]:



#	System	*AvgRec*	F_1^{PN}	*Acc*
1	NileTMRG	**0.583**$_1$	0.610$_1$	0.581$_1$
2	SiTAKA	**0.550**$_2$	0.571$_2$	0.563$_2$
3	ELiRF-UPV	**0.478**$_3$	0.467$_4$	0.508$_3$
4	INGEOTEC	**0.477**$_4$	0.455$_5$	0.499$_4$
5	OMAM	**0.438**$_5$	0.422$_6$	0.430$_8$
	LSIS	**0.438**$_5$	0.469$_3$	0.445$_6$
7	Tw-StAR	**0.431**$_7$	0.416$_7$	0.454$_5$
8	HLP@UPENN	**0.415**$_8$	0.320$_8$	0.443$_7$
B1	All POSITIVE	**0.333**	0.199	0.248
B2	All NEGATIVE	**0.333**	**0.267**	0.364
B3	All NEUTRAL	**0.333**	0.000	**0.388**

In the following table, we show a comparison between our results and the winners of the competition. It is worth noting, however, that the test set that we used is not the same as the test set they used to evaluate their results; however, our test set was large enough to make the comparison plausible. We did not want to compare the results using our test split of the SemEval dataset as it was just 10% of the 671 tweets in that dataset. We are showing the results obtained by combining our SemEval test split with our ASTD test split. We report the average recall, the accuracy and the macro-average F1 measure ordered with respect to the average recall.

#	**System**	**Avg-Rec**	$F1^{PN}$	**Acc**
Our Model	***Arch 4.4***	***0.6***	***0.59***	***0.59***
1	NileTMRG	0.58	0.61	0.58
Our Model	***Arch 4.2***	***0.57***	***0.54***	***0.61***
2	SiTAKA	0.55	0.57	0.56
Our Model	***Arch 4.3***	***0.52***	***0.53***	***0.52***
3	ELiRF-UPV	0.47	0.46	0.50

As shown in the table, our 4.4 model beats the winner of the competition with respect to accuracy and average recall. It is slightly less in terms of the macro-average F1 score. Even our other models obtained very competitive scores that outperform most of the competitors.

7. Conclusion and Future Work

In conclusion, we have experimented with 4 different architectural models based on deep learning in order to compare their capabilities to perform sentiment analysis on Arabic text. We have achieved fair results in terms of accuracy with respect to the systems found in the literature as shown in the paper by Heikal 2008 [7] and the results of SemEval competition [5]. However, due to the limitations of the dataset we worked with, the precision and recall, and subsequently, the F1 measure were not that good initially, but after we applied the enhancements described in section 6.3, we achieved good results. Since most of our tweets were written in Modern Standard Arabic (MSA), we suggest as a future work to investigate with tweets written in different dialects of Arabic as this is a more relevant task because the MSA is rarely used by the majority of people. It is usually used by official pages or news pages. Therefore, it would be interesting to investigate how these models would perform if the tweets were written in different Arabic dialects.

8. References

[1] C. Baziotis, N. Pelekis, and C. Doulkeridis, “DataStories at SemEval-2017 task 4: Deep LSTM with attention for message-level and topic-based sentiment analysis,” in Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), (Vancouver, Canada), pp. 747–754, Association for Computational Linguistics, Aug. 2017.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in Advances in Neural Information Processing Systems (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[4] A. Safaya, M. Abdullatif, and D. Yuret, “KUISAIL at SemEval-2020 task 12: BERT-CNN for offensive speech identification in social media,” in Proceedings of the Fourteenth Workshop on Semantic Evaluation, (Barcelona (online)), pp. 2054–2059, International Committee for Computational Linguistics, Dec. 2020.

[5] S. Rosenthal, N. Farra, and P. Nakov, “SemEval-2017 task 4: Sentiment analysis in Twitter,” in Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), (Vancouver, Canada), pp. 502–518, Association for Computational Linguistics, Aug. 2017.

[6] A. B. Mohammad, K. Eissa, and S. El-Beltagy, “Aravec: A set of arabic word embedding models for use in arabic nlp,” Procedia Computer Science, vol. 117, pp. 256–265, 11 2017.

[7] M. Heikal, M. Torki, and N. El-Makky, “Sentiment analysis of arabic tweets using deep learning,” Procedia Computer Science, vol. 142, pp. 114–122, 2018. Arabic Computational Linguistics.