جامعة عجمان
AJMAN UNIVERSITY

College of Engineering & Information Technology
Department: Information Technology

Text & Web Mining

Arabic Text Diacritization (ATD)

Team Members :

Leena Alsalhi - 202110613
Layan Ahmad - 202110844
Haniyah Alzaben - 202110616

Due Date: 17 Nov 2024

Supervised by:
Dr. Mahmoud Al Shboul

Academic Year 2024- 2025 – Fall

Contents

Arabic Text Diacritization (ATD)

Leena Alsalhi
College of Engineering & IT
Ajman University
Ajman, UAE
202110613@ajmanuni.ac.ae

Layan Ahmad
College of Engineering & IT
Ajman University
Ajman, UAE
202110844@ajmanuni.ac.ae

Haniyah Alzaben
College of Engineering & IT
Ajman University
Ajman, UAE
202110616@ajmanuni.ac.ae

***Abstract*— Our project focuses on Arabic Text Diacritization (ATD). This task includes putting diacritics on letters of the Arabic language to represent the pronunciation precisely, we have employed the sequence-to-sequence approach as well as character-level tagging where each team member handled a unique model, including a Bidirectional LSTM model, a Transformer model and Feed Forward Neural Network model for effective context modeling.**

Keywords—ATD, Deep learning, Text Mining.

I. Introduction

Arabic Text Diacritization (ATD) is a challenging task in Natural Language Processing (NLP) as the Arabic language by itself is complex since it's rich in morphologies. The diacritics are a necessary in Arabic language as it significantly affect the pronunciation and the formal writing specially in Quran. They are also very important for disambiguating words meanings as changing one diacritic a word may change the meaning of a whole sentence.

In this report, we will highlight the key differences of three different models we built to resolve this complex ATD task. Each of our models (Bidirectional LSTM, Feedforward NN and transformer-based model) is ideal for such Arabic processing task.

1. **Bidirectional LSTM (BiLSTM):** this recurrent neural network (RNN) model is designed to process data in both forward and backward motions, which provides a deep understanding of the context in the sequences of data. The model is a great option for such tasks because we need it to capture information from past and future words to get accurate predictions.
2. **Feedforward Neural Networks (FFNN):** This type of architectures isn't usually used for sequence-to-sequence modeling, but we implemented and tested them since they are simpler than Bidirectional LSTMs. We wanted to examine it and see whether a simpler model can still give good results as the previous one or not. Indeed, going with a less computational costs is always encouraged.
3. **Transformer-based Model:** Attention mechanisms have revolutionized Natural Language Processing (NLP) tasks. This model has proven a better performance in several applications of NLP.

Using the data provided to us, we will assess and compare these three models putting into account the computational cost, accuracy and ease of use.

II. Data Preprocessing

To preprocess Arabic text, we have created multiple functions that gave us clean Arabic text to be fed to the model. After loading the test and validation text files, we have used several pickle files to assist in the text preprocessing step, we used pickle files in this project to store mappings, like the Arabic letters and the diacritics. This was helpful as it allowed us to load the preprocessed mappings straight into memory which saved time and maintained consistency throughout various phases of training and evaluation, these pickles were sourced from: https://github.com/AliOsm/arabic-text diacritization/tree/master/helpers/constants

and they include:

1- **ARABIC_LETTERS_LIST** pickle which contains the Arabic letters.
2- **DIACRITICS_LIST** which contains Arabic diacritics.
3- **RNN_SMALL_CHARACTERS_MAPPIN G.**
4- **RNN_BIG_CHARACTERS_MAPPING.**
5- **RNN_CLASSES_MAPPING.**
6- **RNN_REV_CLASSES_MAPPING.**

In the text preprocessing phase, we have created multiple functions to produce clean and normalized text for the model. These functions include:

1- **remove_diacritics** function was used to remove all diacritics from the input text, the purpose of doing this was to help the model learn the relationship between characters and their diacritics.
2- **split_data** function was created to split the input text into smaller segments that were easier to manage, it divided the text based on punctuation marks and ensured that the maximum length for each segment is 500 characters,
3- **clean_text** function was created to perform general data cleaning, such as: remove non-Arabic characters, numbers and punctuations, to ensure that the text is consistent and didn't have any irrelevant symbols.
4- **map_data** function produced sequences of input letters and their matching diacritics, including st (<SOS>) and (<EOS>) indicators. The diacritics were one-hot encoding using the **to_one_hot** function in order to represent each class as a vector.

After creating these functions, we applied them to the train and validation sets. After that, we took a subset from the training data (40%) to reduce training time.

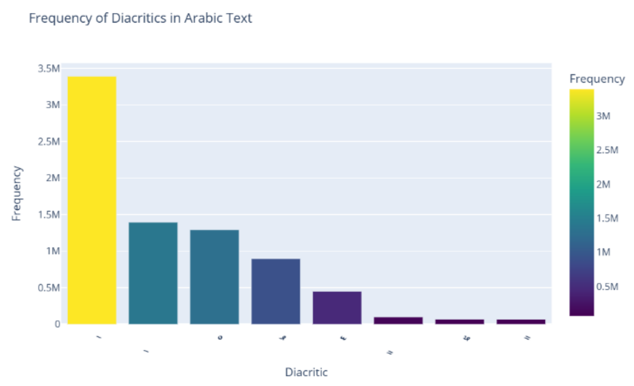
III. Exploratory Data Analysis (EDA)

Before diving into machine learning models, it is essential to understand the structure of the data you are dealing with by through exploratory data analysis (EDA). We wanted to see and understand what type of characters we have and its frequencies, diacritics and many other relevant plots that could bring up meaningful insights.

Class Name	Diacritic
Fatha	بَ
Kasra	بِ
Dhamma	بُ
Tanween Fath	بً
Tanween Kasr	بٍ
Tanween Dhamm	بٌ
Shadda	بّ
Shadda + Fatha	بَّ
Shadda + Kasra	بِّ
Shadda + Dhamma	بُّ
Shadda + Tanween Fath	بًّ
Shadda + Tanween Kasr	بٍّ
Shadda + Tanween Dhamm	بٌّ
Sukoon	بْ
No Tashkeel	<NT>

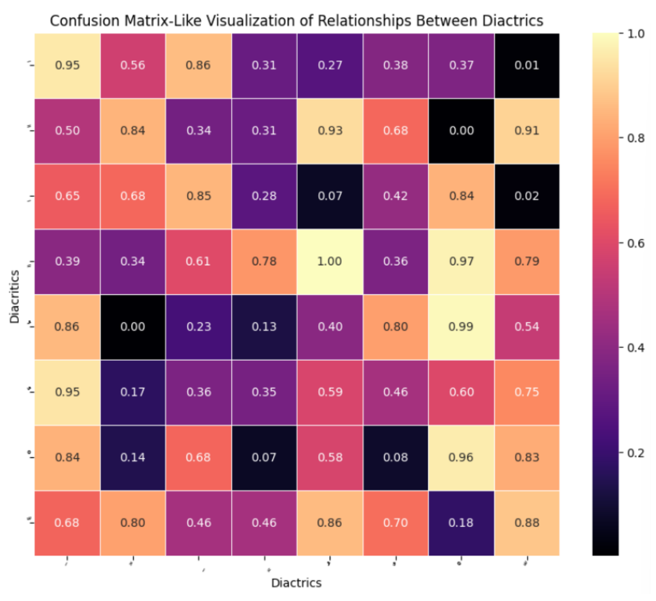
Frequency of Diacritics in Arabic Text:

Starting of with diacritics, we wanted to analyse the frequency and distribution of diacritical marks in the training and validation sets. We began by defining a funtion named extract_diactritics, which recognizes and extracts Arabic diacritical marks. Every word in the dataset is passed to this function, in which it stores the diactrics into a list with the count of each diacritic occurrence being recorded and stored. As shown in the below plot, we can observe that the most frequent diactric is "Fatha" with a 3,390,384 num of occurences, followed by thye "Kasra", and the least diacritic is "Tanween Fath" with a 64766 num of occurences.



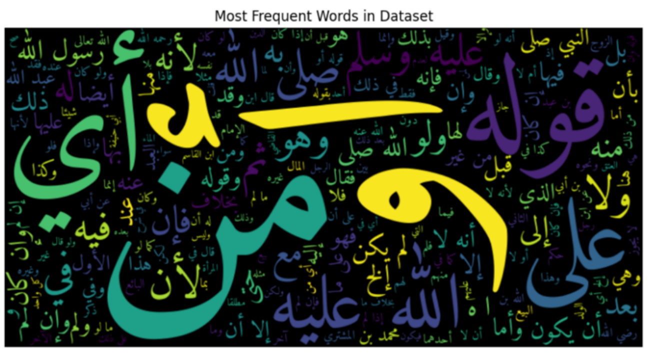
Correlation Matrix: Relationship Between Diactrics

This plot illustractes the relationship or similarities between diactrics using a heatmap, which would help us understand the connections between diacrtical marks. Each value within a square in the matrix is the similarity score bwteeen two diactritics and the color intensity reflects how well or how strong they are similar to each other. The plot shows that the "Fatha" and "Damma" have a similarity value of 0.86, which means they relate to one another, in other words, if one appears, the other appears in a similar context. Another insight is the "Tanween Dhamm" and "Fatha" have a similarity value of 0.95, which also means that they appear in similar contexts. However, "Fatha" and "Shadda" have a very low similarity percentage which is 0.01, meaning they rarely appear together.



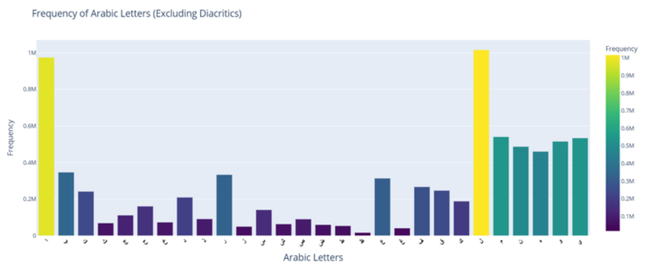
WordCloud: Most Frequent Words:

We plotted this plot for it to help us see what are the most frquenct usef letters or words in our dataset without diactrics. The size if each word in the cloud represents its frequency of occurrence. The more a word is frequent, the larger the word is represented in the cloud, and larger terms indicates their importance in the text. As we can observe the most frequent words in the plot are 'ا', 'و', 'من', 'اى' which appear larger and bolder and within the center of the cloud. Less frequent words are like 'الله', 'رسول الله', which would appear in a smaller size depending on their frequency.



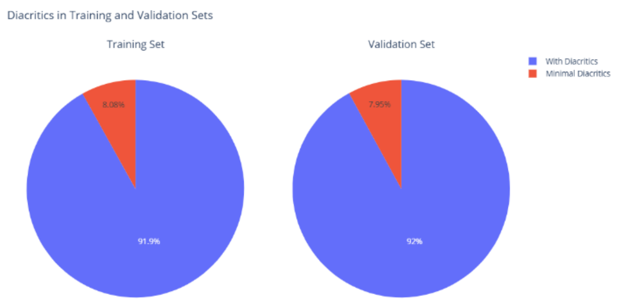
Frequency of Arabic Letters:

We wanted to analyze and visualize the frequency of Arabic letters without diactrics in out dataset. As we can observe below a bar plot is used to visualize these frequencies. We filtered the arabic letters after removing the diactrics from them and used a counter to track and count the occurences of each letter. We sorted the plot alphabetically making it easier to understand. The plot shows that the letter 'ل' (Lam) followed by the letter 'ا' (Alef) are the most frequent characters with a frequency of around 1M, and they are represented in a yellow bar which also shows how frequent they are compared to other letters. However, the least frequent letter is 'ظ' (Dhad). This helps us understand letter usage patterns in Arabic text.



Proportion of Text with Diacritics vs Minimal Diacritics:

We created to oie charts to compare the percentage of the text that has diacritics versus the text that has minimal diactrics for both training and validation sets. A sentence is passed to a count_diactrics() function, if a sentence has diacritics in every word in the sentence it is counted in the with diactrics list, otherwise, if a sentence has few or no diactrics at all it will counted into the minimal diactrics list. This chart helps identify if any imbalancing issues that might be present that could affect the model performance. As shown in the below chart, the sentences with diactrics that are present in the training set are 91.9% and 8.08% are considered as minimal diactrics. For the validation set the sentences counted to as with diactrics are 92%, and 7.95% as minimal diactrics.



IV. Bidirectional LSTM Model

In our project, one of the models that we utilized was Bidirectional LSTM model, this model is sequence-to-sequence based, and it leverages Bidirectional LSTMS to accurately capture each character's past and future context. We chose this as one of the models for Arabic text diacritization over traditional RNNs because this model can get around different issues like the vanishing gradient descent. Also, Bidirectional LSTMs offer a richer context as it processes the sequences in both directions, which makes this model more effective for tasks like this one, which is considered a complex one, since it requires to understand the context before and after each character to make accurate predictions.

Our model consisted of multiple layers:

1- Input Layer.
2- Embedding Layer.
3- Bidirectional LSTM Layers.
4- Dropout Layers.
5- TimeDistributed Dense Layers.
6- Output Layer.

A. Model Architecture

1- **Input:** which takes a sequence of characters as an input (no diacritics).
2- **Input Embeddings:** the input sequences are transformed in this layer, from character indices to dense vectors, this will help the model to understand relationships between the letters based on their learned embeddings.
3- **Bidirectional LSTMs:** the embeddings are then fed into 2 bidirectional LSTM layers, the fact that it's 'bidirectional' is very important in diacritization, because in the Arabic language, the diacritic of a letter is affected by the letters before and after it.
4- **Dropout:** dropout layers help in preventing overfitting, which is why we added a dropout layer after each bidirectional LSTM layer to make sure that the model doesn't memorize the training data.
5- **TimeDistributed:** these layers are used with a ReLU activation function to help in transforming the output of the Bidirectional layer in a nonlinear manner. The output layer uses this layer with a softmax activation function instead of ReLU to provide the probability distribution for every character.
6- **Output:** the final layer produces a softmax probability for each diacritic for every letter, the model then selects the diacritic with the highest probability.

B. Training Process

For training the model, sparse categorical cross-entropy was chosen as the loss function and Adam as an optimizer, we have also utilized early stopping as a callback to avoid overfitting, which is going to stop training if the validation loss doesn't improve for 3 epochs, we trained the model with 10 epochs and a batch size of 256. We had good accuracy on the validation set: 99.6% and a loss of 0.0124.

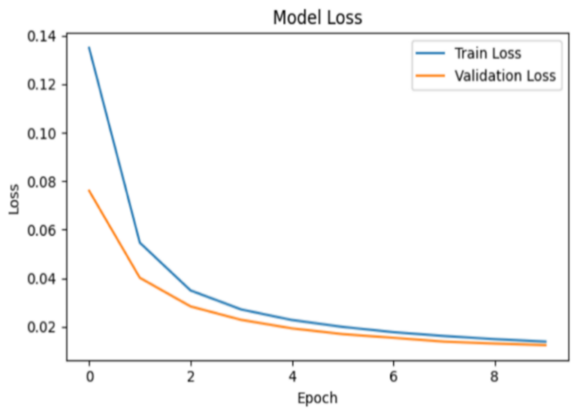


Figure 1: BiLSTM model loss

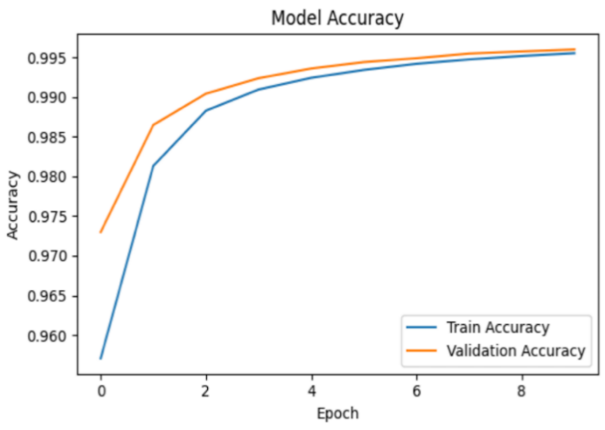


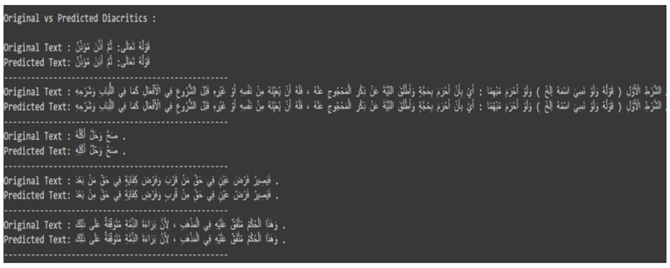
Figure 2: BiLSTM model accuracy

Both training and validation accuracies are remarkable, as we have achieved an accuracy of 99.6%. In the accuracy figure, we noticed that the training shows a consistent increase, this indicates that the model is learning the relationships, the validation is showing the same behavior which means that it's generalizing well to the validation data.

In the loss figure, both training and validation are decreasing after each epoch, reaching very low values.

C. Model Evaluation

We evaluated the model on new unseen text from the test data, our goal was to predict the correct diacritics for the letters in the text and compare it with the original text and the figure below shows the output:



```
Original vs Predicted Diacritics :

Original Text : قَوْلُهُ تَعَالَى: ثُمَّ أَذَّنَ مُؤَذِّنٌ
Predicted Text: قَوْلُهُ تَعَالَى: ثُمَّ أَذِنَ مُؤَذِّنٌ
----------------------------------------------
Original Text : الشَّرْطِ الْأَوَّلِ ( قَوْلُهُ وَلَوْ نَسِيَ اسْمَهُ إلَخْ ) وَلَوْ أَحْرَمَ مُبْهَمًا : أَيْ بِأَنْ أَحْرَمَ بِحَجَّةٍ وَأَطْلَقَ النِّيَّةَ عَنْ ذِكْرِ الْمَحْجُوجِ عَنْهُ ، فَلَهُ أَنْ يُعَيِّنَهُ مِنْ نَفْسِهِ أَوْ غَيْرِهِ قَبْلَ الشُّرُوعِ فِي الْأَفْعَالِ كَمَا فِي اللُّبَابِ وَشَرْحِهِ
Predicted Text: الشَّرْطِ الْأَوَّلِ ( قَوْلُهُ وَلَوْ نَسِيَ اسْمَهُ إلَخْ ) وَلَوْ أَحْرَمَ مُبْهَمًا : أَيْ بِأَنْ أَحْرَمَ بِحَجَّةٍ وَأَطْلَقَ النِّيَّةَ عَنْ ذِكْرِ الْمَحْجُوجِ عَنْهُ ، فَلَهُ أَنْ يُعَيِّنَهُ مِنْ نَفْسِهِ أَوْ غَيْرِهِ قَبْلَ الشُّرُوعِ فِي الْأَفْعَالِ كَمَا فِي اللُّبَابِ وَشَرْحِهِ
----------------------------------------------
Original Text : صَحَّ وَحَلَّ أَكْلُهُ .
Predicted Text: صَحَّ وَحَلَّ أَكْلِهِ .
----------------------------------------------
Original Text : فَيَصِيرُ فَرْضَ عَيْنٍ فِي حَقِّ مَنْ قَرُبَ وَفَرْضَ كِفَايَةٍ فِي حَقِّ مَنْ بَعُدَ .
Predicted Text: فَيَصِيرُ فَرْضَ عَيْنٍ فِي حَقِّ مَنْ قُرْبِ وَفَرْضَ كِفَايَةٍ فِي حَقِّ مِنْ بَعُدَ .
----------------------------------------------
Original Text : وَهَذَا الْحُكْمُ مُتَّفَقٌ عَلَيْهِ فِي الْمَذْهَبِ ، لِأَنَّ بَرَاءَةَ الذِّمَّةِ مُتَوَقِّفَةٌ عَلَى ذَلِكَ .
Predicted Text: وَهَذَا الْحُكْمُ مُتَّفَقٌ عَلَيْهِ فِي الْمَذْهَبِ ، لِأَنَّ بَرَاءَةَ الذِّمَّةِ مُتَوَقِّفَةٌ عَلَى ذَلِكَ .
----------------------------------------------
```

Figure 3: BiLSTM Model Output

We obtained a character-level accuracy of 51.72% on the test set, this suggests that the model was able to accurately capture only some of the complexities in Arabic diacritization. There is definitely room for improvement.

This model is an excellent place to begin for ATD, if more complex models were used, achieving a higher character-level accuracy. The Bidirectional LSTM model accurately captures context, but it perhaps struggles to deal with complex language structures.

V. FEEDFORWARD NEURAL NETWORK (FFNN)

FFNN is one of the most basic types of artificial neural networks. In FFNN data flows from the input layer to the output layer through hidden layers in a single direction. Each layer consists of multiple neurons, where they compute the weighted sum of its previous inputs. Unlike RNN, FFNN were made to process inputs on their own or independently without depending on sequential information.

We chose FFNN for our Arabic Text Diacritization (ATD) task for multiple reasons. One advantage that we chose is the **Independence of Characters**, since it is possible to discretize each character independently, without having to rely on its prior states. It simplifies the process by focusing on individual character prediction. Another reason is that FFNN has a simple structure, which makes it simpler to develop and train.

A. Model Architecture

We started by creating a function named create_model2() function that defines the model structure.

Input Layer: This layer takes in a sequence of character indices.

Embedding Layer: This layer converts the sequence of characters from its previous layer into dense vector representation. Each character in the CHARACTER_MAPPING.pickle file is mapped to a 25-dim dense vector which aids the model in understanding how characters relate to one another in a continuous space.

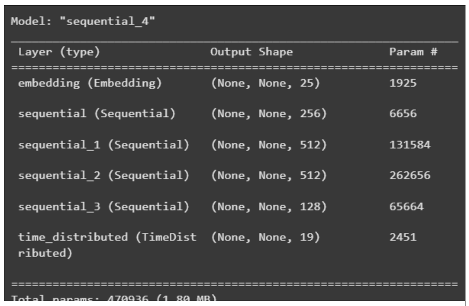
Hidden Layers: We also added fully connected dense layers with varying number of neurons within each hidden layer (ex: 256, 512,128) which were tuned and balanced to capture the data patterns and complexity.

These dense layers are with a ReLU activation function that introduces non-linearity.

Dropout layers were added with a dropout_rate=0.5 after each dense layer in order to prevent overfitting.

These layers help in extracting features necessary for predicting diacritics.

Output Layer: A TimeDistributed layer is added with a softmax activation function that is used across multi diacritic classes. Here is the **Model Summary**:



```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 25)          1925

 sequential (Sequential)     (None, None, 256)         6656

 sequential_1 (Sequential)   (None, None, 512)         131584

 sequential_2 (Sequential)   (None, None, 512)         262656

 sequential_3 (Sequential)   (None, None, 128)         65664

 time_distributed (TimeDist  (None, None, 19)          2451
 ributed)

=================================================================
Total params: 470936 (1.80 MB)
```

Figure 4: FFNN Model Summary

B. Training Configuration

Compilation: The model uses a loss function of categorical cross-entropy, Adam as an optimizer, and Accuracy as an evaluation metric.

C. Training Process

The model is trained in over 5 epochs and a batch size of 256. In order to guarantee generalization, validation accuracy is monitored.

Table 1: FFNN Training Process

Epochs	Train Loss	Train Acc	Valid Loss	Valid Acc
Epoch 1	0.163	0.956	0.103	0.963
Epoch 2	0.099	0.965	0.101	0.964
Epoch 3	0.097	0.966	0.101	0.964
Epoch 4	0.096	0.996	0.100	0.964
Epoch 5	0.096	0.996	0.100	0.964

As you can see from the above table the accuracy for the training is 0.966 with a loss of 0.09, and a validation accuracy of 0.964 with a loss of 0.1.
To evaluate our model training and validation accuracy and loss, we decided to plot them.

Model Accuracy Plot:

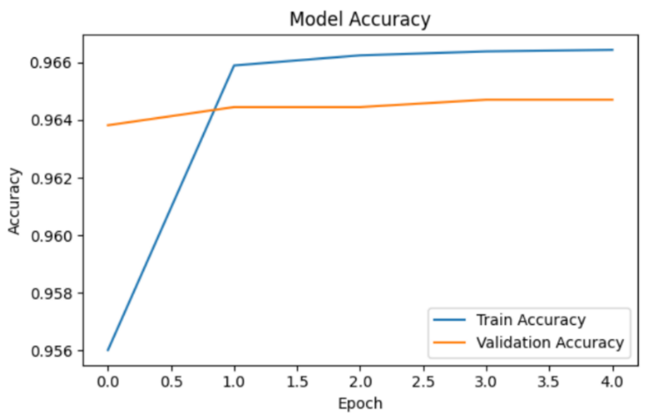


Figure 5: FFNN Model Accuracy

The model training accuracy seems to increase quickly and stabilize around 0.96. However, the validation starts around 0.963 which is slightly lower than the training after it stabilized and stays close without changing its accuracy across the epochs. Both training and validation are a bit close together, indicating that the model generalizes well to the validation data, in other words it shows a slight overfitting since there Is a small gap. The model stabilizes quickly across the epoch's indication that the model is not too complex. We might consider tuning the model more (even though we did) to make it learn patterns better.

Model Loss Plot:

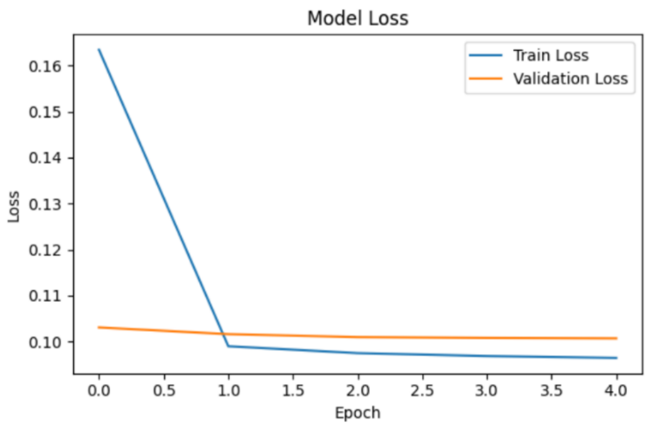


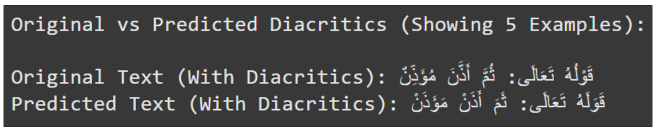
Figure 6: FFNN Model Loss

In the loss figure, for the training the loss seems to be decreasing very fast. However, the loss for the validation is very low from the starts and stays stable across the epoch.

D. Evaluation & Testing

After training the model and testing it with our validation set, An Arabic text test dataset is used to evaluate the model's performance. The model is used to predict the didactics of newly unseen text from the test txt file, comparing the output with the original diacritics to calculate character-level accuracy. The character level accuracy is 14.58% which is very bad accuracy which means it wasn't able to predict the right diacritics on some of the characters. As shown below we printed out 5 random examples that show the original text and the predicted.

<u>Original Vs Predicted:</u>



```
Original vs Predicted Diacritics (Showing 5 Examples):

Original Text (With Diacritics): قَوْلُهُ تَعَالَى: ثُمَّ أَذَّنَ مُؤَذِّنٌ
Predicted Text (With Diacritics): قَوَلَهُ تَعَالَى: ثَمَ أَذَنْ مَؤَذَنْ
```

VI. Transformer Model

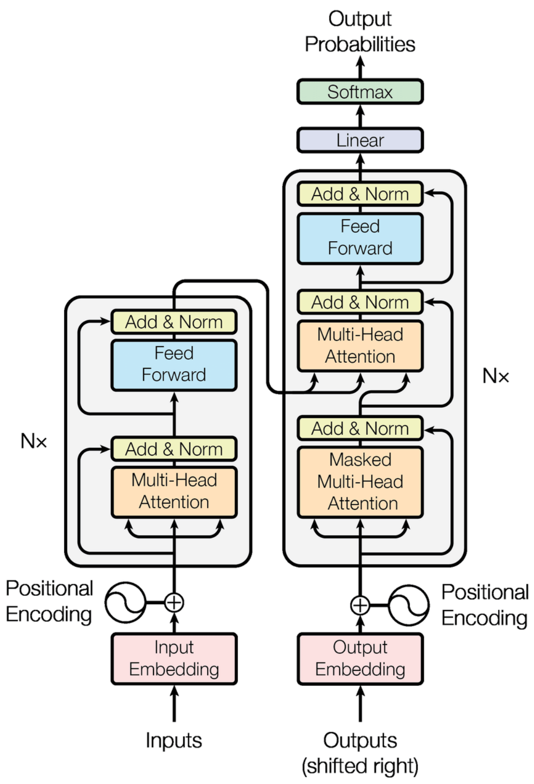


Figure 7 : Transformers Architecture

A. Introduction

Transformer models have been rising as the state of the art for sequence-to-sequence tasks. As deep learning has been transforming natural language processing tasks by its quick development and change.

The architecture of the model acts as an efficient approach for Arabic text diacritization (ATD) task. Transformers leverage the attention mechanism as shown in the previous figure which provides a better accuracy compared to statistical methods.

B. Model Architecture

The model built has a transformer base architecture that was created for the Arabic text tasks particularly. We expanded it to meet the needs of our task which is diacritization. Adding a diacritization head, enhanced the architecture of the model making it suitable for token level classification.

1. Encoder

The model incorporates the use of a multi-layer encoder where each layer is composed of:

- **Multi-Head self-attention mechanism:** the model will pay attention to vital parts of the input.
- **Feed forward Neural Networks (FFNN):** it takes the output of the mechanism as an input to capture complex feature representation.
- **Normalization Layer:** these are applied to boost the convergence and have a stable training process.

2. Diacritization Head

On top of the encoder, we added a classification head to predict diacritics for the input tokens. The head consists of:

- Dense layer: connects the contextual embeddings to the space of labels which are the diacritics.
- Softmax activation function: gives the output as probability of the distribution of the classes (diacritics).

C. Training Process

<u>Input & Output of the model:</u>

We fed the model with a clean, undiacritized text which was preprocessed using a function that removes diacritics, digits and any non-linguistic characters. Using the Arabic Bert Word Piece tokenizer, we were able to divide the input text into units making sure that both rare and common words are properly represented. On the other hand, the output of the model was a sequence of labels (diacritics) mapped to the input tokens including all types e.g. (ْ ، ِ ، ُ ، ◌َ). Eventually, the class with the highest probability will be selected by the model for each token.

In order to have a uniform length between the sequences we specified a maximum length of 128 and padded the sequences accordingly.

Additionally, the model was optimized using AdamW optimizer with a learning rate of 1 x $10^{-4.}$ AdamW is an optimizer that is commonly used for transformers. The loss function that we applied as the categorical cross-entropy due to the multi class task we have. For each training step, the model processed a batch of size 32. All of that helped with ensuring the model will learn effectively and give out good results.

D. Evaluation & Results

During our training process, we focused on monitoring the training and validation losses using our train and validation data splits.
Using Colab pro with NIVIDIA GPUs, leveraging PyTorch framework, we ensured having an efficient computation with the following training and validation losses across 5 epochs:

Table 2:Transformers Training Process

Epochs	**Training Loss**	**Validation Loss**
Epoch 1	0.767	0.756
Epoch 2	0.759	0.755
Epoch 3	0.758	0.754
Epoch 4	0.757	0.755
Epoch 5	0.757	0.73

The figure below shows the learning curve of the model monitoring the loss for 5 epochs, where the blue line represents the training loss, and the orange line represents the validations loss The decreasing of both gives a good indication about the model. Meaning that if the model was trained for longer, we may get a better result than this.

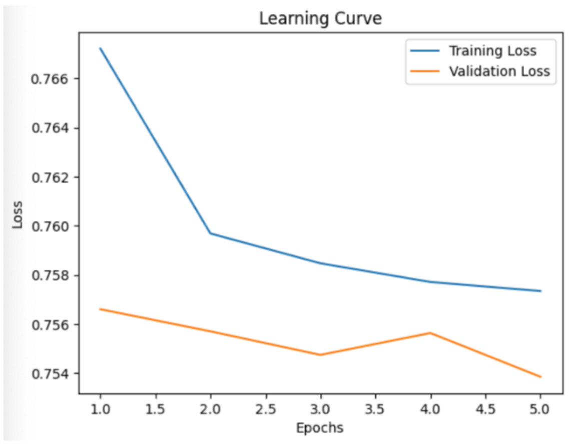


Figure 8: Transformers Learning Curve (Monitoring the Loss)

Afterwards, we evaluated our model on the test data split, achieving an accuracy of 81.23% and a loss of 0.759 which reflects a good generalization capability.

VII. CONCLUSION

In this project, where we explored three different deep learning architectures to resolve the task of Arabic Text Diacritization (ATD). Each model was evaluated based on the computational cost and the ability to capture the diacritics properly.

The BiLSTM model has appeared to be one of the most effective model in sequence modeling compared to the other models we deployed. It has proven how the bidirectional processing is crucial for capturing diacritics accurately.

Although, the FFNN required less computational cost, its overall performance was worse. Meaning that, this architecture may work better for tasks requiring sequential baselines than such tasks like ATD.

The third and last model, Transformers needed a lot of resources and in terms of accuracy it wasn’t performing best. Despite its efficiency for such tasks it didn’t give satisfying results.

In conclusion, we studied these different architectures of deep learning using various libraries such as tensorflow and pytorch. We were able to get some outperforming results but for sure there is always a space for improvement. For future improvements, we may look into different techniques of embeddings or explore pretrained transformers for Arabic language specifically such as BERT.

VIII.References

Osman, A. "Arabic Text Diacritization Dataset." GitHub Repository, 2023. Available at: https://github.com/AliOsm/arabic-text-diacritization

Khamsi, B. "Deep Diacritization for Arabic Text." GitHub Repository, 2022. Available at: https://github.com/bkhmsi/deep-diacritization

Alasmary, F., Abjad Ltd., Zaafarani, O., Abjad Ltd., Ghannam, A., & Abjad Ltd. (2024). CATT: Character-based Arabic Tashkeel Transformer. Proceedings of the Second Arabic Natural Language Processing Conference, 250–257. https://aclanthology.org/2024.arabicnlp-1.21.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., & Polosukhin, I. (2017). *Attention Is All You Need*. https://arxiv.org/pdf/1706.03762

Amiri - Google Fonts. (2024). Google Fonts. https://fonts.google.com/specimen/Amiri

Denvil-Sommer, A., Gehlen, M., Vrac, M., & Mejia, C. (2019). LSCE-FFNN-v1: a two-step neural network model for the reconstruction of surface ocean &lt;i&gt;p&lt;/i&gt;CO&lt;sub&gt;2&lt;/sub&gt; over the global ocean. Geoscientific Model Development, 12(5), 2091–2105. https://doi.org/10.5194/gmd-12-2091-2019

Mining Text Data. (2012). In Springer eBooks. https://doi.org/10.1007/978-1-4614-3223-4

PyTorch Documentation: PyTorch. (n.d.). PyTorch Documentation (Stable). https://pytorch.org/docs/stable/index.html

contributors (https://github.com/huggingface/transformers/graphs/contributors), T. H. F. team (past and future) with the help of all our. (n.d.). *transformers: State-of-the-art Machine Learning for JAX, PyTorch and TensorFlow*. PyPI. https://pypi.org/project/transformers/

mukhal.(2020). *transformer-diacritization/transformer/base_transformer.py at master · mukhal/transformer-diacritization*. GitHub. https://github.com/mukhal/transformer-diacritization/blob/master/transformer/base_transformer.py