The University of Jordan

Authorization Form

I, Abrar Khaled Samara, authorize the University of Jordan to supply copies of my Thesis/ Dissertation to libraries or establishments or individuals on request, according to the University of Jordan regulations.

Signature:

Date:

Performance Evaluation and Optimization for BiLSTM Neural Networks for Arabic Language Applications

By

Abrar Khaled Mustafa Samara

Supervisor

Prof. Gheith Abandah

This Thesis was submitted in Partial Fulfillment of the Requirements for the Master's Degree of Computer and Networks Engineering

Faculty of Graduate Studies

The University of Jordan

Aug 23, 2021

COMMITTEE DECISION

This thesis/dissertation (Performance Evaluation and Optimization for BiLSTM Neural Networks for Arabic Language Applications) was successfully defended and approved on ----------

Examination Committee	Signature
Dr. Gheith Abandah, (Supervisor) Prof. of Computer Engineering	----------
Dr. Omar Al-Kadi, (Member) Prof. of Computer Information Systems	----------
Dr. Mohammad Abdel-Majeed, (Member) Assistant Prof. of Computer Engineering	----------
Dr. Hisham Almasaeid, (Member) Associate Prof. of Computer Engineering (Yarmouk University)	----------

AKNOWLEDGEMENT

I would like to express my gratitude to my thesis supervisor, Prof. Gheith Abandah, who guided me throughout this thesis and helped me finalize it. I would like also to thank my family who supported me throughout the study period.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABREVIATION AND SYMBOLS

Abbreviation	Clarification
APCD	Arabic poem comprehensive dataset
BiCuDNNLSTM	Bidirectional CUDA deep neural network long short-term memory
BiGRU	Bidirectional gated recurrent unit
BiLSTM	Bidirectional long short-term memory
BPTT	Backward propagation through time
CA	Classical Arabic
CNN	Convoluational neural network
CuDNN	CUDA deep neural network
cuDNNLSTM	CUDA deep neural network long short-term memory
DER	Diacritization error rate
DL	Deep learning
DNN	Deep neural network
FSPC	Frame skipping and posterior copying
GP	Grow-and-prune
GRU	Gated recurrent unit
LSTM	Long short-term memory
mAP	Mean average precision
NLP	Natural language processing
nvidia-smi	NVIDIA system management interface
RCRN	Recurrently controlled recurrent networks
RNN	Recurrent neural network
TDNN	Time delay neural network
WER	Word error rate

Performance Evaluation and Optimization for BiLSTM Neural Networks for Arabic Language Applications

By
Abrar K. Samara

Supervisor
Prof. Gheith A. Abandah

ABSTRACT

Long short-term memory (LSTM) and bidirectional LSTM (BiLSTM) recurrent neural networks are popular in Arabic natural language processing (NLP) applications. However, they are slow to train and take many hours to several days in the training process. I investigate in this thesis solutions to enable faster training.

This thesis contains improvement of training time for deep learning models of Arabic applications. Poem classification and Arabic text diacritization are taken as case studies in this thesis. My improvements include using of GPU environment rather than a CPU and using of fast LSTM layer implementation. The BiCuDNNLSTM is a state-of-the-art layer that achieves high improvement in training time. Training time for poem classification model based on this layer is faster by 65 times over the base BiLSTM model. In addition, training time for the proposed diacritization model is faster by 21 times than the base BiLSTM model. This time improvement has little effect on the model accuracy. BiLSTM model achieves a test accuracy of 97.29% and BiCuDNNLSTM achieves 97.17% on poem classification. In addition, BiLSTM model achieves a validation accuracy of 98.41% and BiCuDNNLSTM achieves 99.16% for diacritization. I used the word to vector and fasttext models to improve the training time. They improved the training time but with low accuracy.

I used Sysstat and Nvidia-smi to measure the CPU and GPU utilization. BiCuDNNLSTM achieves GPU utilization of 66.6% in the poem classification training experiment and 67.3% in diacritization experiment. The BiLSTM model has a GPU utilization of only 35%.

Chapter 1: Introduction

This chapter includes a background of my thesis, explains the research problem, and states the main points of its importance.

1.1 Background

Machine learning is a field that has an important role in many fields. It is used to develop many language solutions such as language translation, determining tags of speech, and natural language processing (NLP) that includes sentiment analysis, speech recognition, machine translation, and others fields. Recurrent neural networks (RNN) are used to build deep learning models and are suitable for time series and sequences data. Long short-term memory (LSTM) and bidirectional long short-term memory (BiLSTM) are RNN layers that are used to train models that have sequences and textual datasets.

The Arabic language has several applications in machine learning including translation, diacritization, and text classification. Poem classification to determine the poem class or meter (Abandah *et al*., 2020) and Arabic text diacritization (Abandah and Abdel-Karim, 2020) are recent examples of applying machine learning for the Arabic language.

These applications were improved by many researchers to reach better deep learning models and tune their hyperparameters. Accuracy, CPU/GPU utilization, training time are considered some of the most important metrics due to the aims of each application.

Training time is an important issue in training deep learning models. It takes from few hours to several days to finish a complete model training of a large dataset. Therefore, decreasing the training time becomes a good research objective.

Performance evaluation is very important to measure the validity of training for deep learning models and accuracies. In addition, monitoring the hardware environment gives

good insights to understand these models. In this thesis, I focus on reducing training time for Arabic language applications and characterizing the performance of the used models.

1.2 Research Problem

Applying machine-learning techniques on Arabic applications became popular in recent years. Deep learning models especially recurrent neural networks (RNN) of type LSTM and BiLSTM are frequently used for training models. Training of these deep learning models takes a lot of time and is a critical issue for Arabic applications. In addition, knowing the characteristics of each model is important to make a performance evaluation. The evaluation metrics include metrics such as training time, utilization of the CPU and the GPU, accuracy, error rates, and loss. This help to know the characteristic of each model and its training environment, which the models are training on it. In addition, knowing the status and utilization of hardware (CPU, GPU, and memory) gives more insights for these models during training.

Abandah and Abdel-Karim (2020) developed the state-of-the-art deep learning model for Arabic text diacritization. The training time is 28.7 hours for a small dataset (LDC ATB3) on a model of 4 BiLSTM layers. In addition, it takes 31.3 hours for the larger Tashkeela dataset on the same model.

Abandah *et al*. (2020) used deep BiLSTM for Arabic poem classification and diacritization. It takes 3.1 training hours with 4 BiLSTM layers and a subset of the APCD2 dataset. This research spent a lot of time (in hours) on training. I will apply the performance evaluation and optimization on these two research problems.

The main objective of this research is to understand the performance of BiLSTM networks for Arabic applications and find the state-of-the-art way that improves the performance of these networks and reduces the training time.

1.3 Research Importance

The importance of this research lies in the following two main points:

1. Knowing the characteristic and evaluation metrics (training time, the utilization of CPU, the utilization of GPU, accuracy, error rates, and loss) of deep learning models for Arabic applications. And describe the status and utilization of CPU and GPU during training.
2. Optimizing deep learning models for Arabic applications to reduce training time. This helps my readers and researchers to get a faster way for training their research models in the Arabic language applications field.

1.4 Thesis Outline

This thesis includes related works for my research in Chapter 2. Chapter 3 includes a theoretical background for related models, neural networks, definitions, and tools that I used in my experiments. Chapter 4 includes the experimental setup and environment, dataset description, and thesis methodology. Chapter 5 includes my training and evaluating results and their discussion. Finally, my thesis conclusion and future works are in Chapter 6.

Chapter 2: Related Work

This chapter includes three parts: main papers that my research depends on, papers that include performance evaluation of deep neural networks (DNN), and papers that include performance optimization for DNN and fast LSTM.

2.1 Target Research Work

In this research, I focus on two papers, which concentrate on Arabic applications. I take them as examples for performance characterization and improvement.

Abandah and Abdel-Karim (2020) developed three RNNs: BiLSTM, unidirectional LSTM, and encoder/decoder LSTM networks. To find a fast and accurate model for Arabic text diacritization, BiLSTM was the state-of-the-art model proposed that achieves the best results.

They used two datasets: LCD ATB3 and Tashkeela. For performance evaluation, they measured the execution time, accuracy, diacritics error rate (DER), and word error rate (WER). They tried to optimize the performance by increasing the number of network layers and using dropout. They reported the best result using BiLSTM with four layers and dropout. The best reported DER shows 47% improvement compared to previous research.

Abandah *et al*. (2020) used deep BiLSTM for Arabic poem classification and diacritization. APCD dataset was used which contains Arabic poems. Tashkeela and LDC ATB3 datasets contain prose. The classification problem has 16-poetry classes meter and the prose class. Their experiments were performed on a platform that contains CPU and GPU. The GPU does not perform better than the CPU on their narrow and deep model. The state-of-the-art model includes four bidirectional LSTM (BiLSTM) layers with only 64 cells for each layer, an input layer, and an output *softmax* layer. They achieved accuracy between 97.27 to 100% depending on the poem length.

2.2 Performance Evaluation and Characterization

Liu *et al*. (2019) applied training for many deep learning models on mobile devices and characterized the performance for each model. The performance analysis includes CPU/GPU utilization, memory use, and power consumption. They measure several performance metrics including CPU utilization for each core, GPU utilization, throughput, energy consumption (by calculating power consumption for each time interval), and the usage of memory. They used *Nvprof* as a tool for profiling the CPU/GPU execution and to characterize their state (idle and busy). *Tegrastat* is used to measure the utilization and power consumption of the hardware. In addition, they used the TensorFlow profiler. They used TensorFlow V1.13 to train several models with various batch sizes. These models include ResNet50, Xception, Seq2Seq, DensNet40, and deep reinforcement learning. They presented their results for each metric in graphs to clarify the results and show the differences among the CPU cores and the GPU.

Shewalkar *et al*. (2019) evaluated the performance of three models for RNN of speech recognition: bidirectional RNN, BiLSTM, and BiGRU. The used dataset is TED-LIUM. The aim is to create an English text (output) from spectrograms of speech. The model includes five hidden layers, 30% dropout, 10 epochs, batch size of 16, RELU activation function, 500 or 1000 neurons in hidden layers, *adam* optimizer, and a learning rate of 0.0001. The measured evaluation metrics include loss, mean edit distance, run time, and WER. LSTM had the best results in WER, loss, and mean edit distance. For the run time, RNN achieved the best and LSTM had the longest time.

Awan *et al*. (2017) trained DNN models: AlexNet and ResNet-50 on Caffe framework with multiple hardware architectures CPUs/GPUs and memory configurations. They evaluated the performance for several cases and focused on training time. They studied the performance at the layer level. They used data parallelism to enhance the performance.

Guo *et al*. (2019) studied several deep learning models and characterized them with different frameworks and platforms. Accuracy is evaluated on four frameworks: MXN_{ET}, CNTK, TensorFlow, and Pytorch. MNIST, CIFAR-10, IMDb were the used datasets. Seven DNN models were studied, such as LeNet-1, LSTM, and ResNet-20.

The traffic sign detection system includes traffic sign detection and recognition (Arcos-Garcia *et al*., 2018). This was developed and evaluated by four object-detection systems (Faster R-CNN, SSD, YOLO V2, and R-FCN). Each system used a deep learning model such as ResNet V1 101 and Inspect ResNet V2. The Microsoft COCO dataset and the German Traffic Sign Detection Benchmark dataset were used. The evaluation was applied, and the comparison was performed by specific metrics: memory usage, FLOPS, number of parameters per model, image size, the mean average precision (mAP), and training time. The training was performed on a computer with Intel core i7-4770 CPU and NVIDIA Titan Xp discrete GPU. They used TensorFlow Object Detection API, Darkflow, and Darknet as development tools. TensorFlow was used as a profile tool to measure the target metrics.

Guignard *et al*. (2018) characterized the performance of a deep learning model called Fathom on the IBM Minsky platform. They executed the model on two modes: CPU only and CPU + GPU. The execution time was measured for two modes. The GPU was used as an accelerator and its impact is analyzed for each model. The GPU characteristics were described by NVIDIA *nvprof* profiling tool. Liu *et al*. (2018) compared three deep learning (DL) frameworks: TensorFlow, Caffe, and Torch. They applied DL models on two datasets: MNIST and CIFAR-10. They measured training time, testing time, accuracy, and adversarial robustness to benchmark these frameworks. The GPU was used to study its impact on the training process.

Hanhirova *et al*. (2018) characterized latency and throughput for convolutional neural networks (CNN) for computer vision on mobile applications including object detection and recognition in videos and images. They trained them with and without GPU. TensorFlow was used for benchmarking and TensorRT was used as a tool to run inference model on the GPU.

2.3 Performance Optimization and Fast LSTM

Trianto *et al*. (2018) proposed a model for speech recognition. Their system contains two parts: a front-end system to enhance the audio by beamforming techniques, and a backend system, which includes a fast LSTM network. Fast LSTM network contains three layers of time-delay neural network (TDNN) and five layers of LSTM. They got the best training time and WER by using their proposed fast LSTM compared with other models. They used the CHIME4 dataset.

Miao *et al*. (2016) presented two simplifications for LSTM for acoustic models to reduce training time. They used forget gate to derive input gate from it, and eliminated the recurrent input for the output gate. In addition, they applied frame skipping to improve training for LSTM. They used Frame Skipping and Posterior Copying (FSPC) to improve LSTM encoding.

Dai *et al*. (2018) proposed a hidden layer LSTM that improved the accuracy and run-time latency for image captioning and speech recognition applications. Grow-and-prune (GP) training was used, which controlled the connection between the hidden layers. The number of parameters for their model was reduced by using GP. It also improved WER and run time. Yu, W. *et al*. (2019) applied fast training by using simplified LSTM cell instead of conventional LSTM. The simplified LSTM cell includes two parts: simplified GRU cell as recurrent part and feed-forward part. They suggested this cell to avoid backward propagation through time training (BPTT), which is slow.

Huang *et al*. (2017) proposed an improved Residual LSTM that converges faster. TIMIT tasks, THCHS-30, Switchboard corpora, and Librispeech were used. The improvement was done by changing the connections inside the LSTM cell.

cuDNN is a library used to run deep learning in Nvidia GPUs (Chetlur *et al*., 2014). It is used as a low-level API, which is easier to integrate into different frameworks. The integration of common frameworks such as Caffe with the cuDNN convolution layer achieves a good speedup for the training time. It reduces the usage of the auxiliary memory. In addition, there is no need to write parallel programs when using the cuDNN library. Tay *et al*. (2018) proposed a new architecture for RNN layers called recurrently controlled recurrent networks (RCRN). It consists of two components: a controller and listener cells. They applied it on natural language processing (NLP) tasks and achieved a high accuracy. One RCRN layer is equivalent to three BiLSTM layers. cuDNN is applied to optimized BiLSTM and RCRN. It reduces training and inference time for both layers.

In this research, I apply performance evaluation and characterization on Arabic applications models including Arabic text diacritization (Abandah and Abdel-Karim, 2020) and Arabic poem classification and diacritization (Abandah *et al*., 2020). Deep learning models in these two main research are taken to characterize them. In addition, I improve the training time for these models by applying fast LSTM and using GPU as an accelerator with cuDNN library.

Chapter 3: Recurrent Neural Networks

This chapter includes a theory explanation of RNNs, its types, and definitions of its evaluation metrics.

3.1 Basic RNN

A recurrent neural network is a neural network consists of recurrent layers; each layer contains recurrent cells. RNNs usually use time series data or sequential data, RNNs are used in many issues such as speech recognition, language translation, and Image captioning (Education, 2021). RNNs have several types of cells includes LSTM cell and its types and variants, gated recurrent unit (GRU) cell, and minimal gated unit (MGU). RNNs have a dependency between inputs and outputs on the other hand; the inputs and outputs are independent in other types of neural networks (Yu, Y. *et al*., 2019). Figure 1 shows basic RNN Network including input X and output Y and a recurrent input and output$S$. In addition, unfold representation for RNN with several time steps on the right side. Figure 2 shows a basic RNN cell and it includes *sigma* cell, input x at time t, recurrent input h at time $t-1$, output y and recurrent output h at time t, and bias with a specific value.

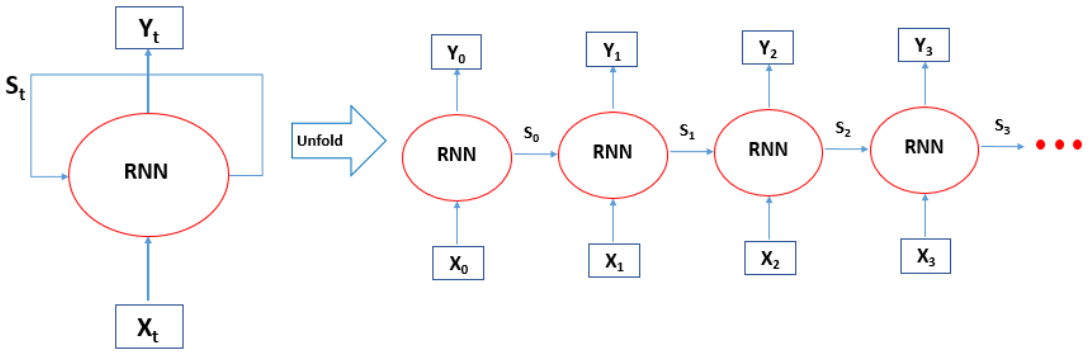

Figure 1. Basic RNN network including input X, output Y, recurrent state S. The unfolded representation is on the right side.

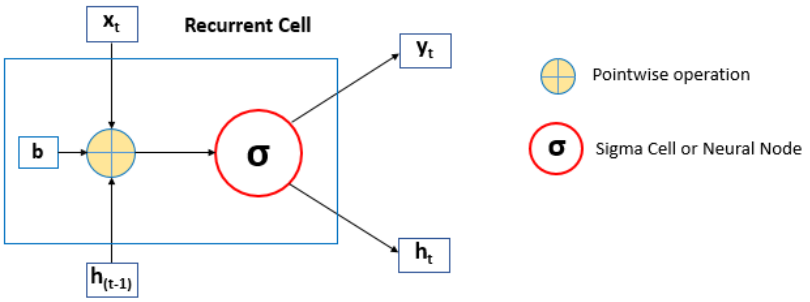

Figure 2. Basic RNN cell including *sigma* activation, input x at time t, recurrent input h from time $t-1$, output y, and recurrent output h at time t, and bias.

The mathematical calculations and expressions for *sigma* cells are:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \quad (1)$$

$$y_t = h_t \quad (2)$$

where, x_t is input, h_t is a recurrent information, W_h and W_x are weights, b is a bias, and y_t is the output of the recurrent cell. These cells types have achieved good results in problems that do not have long-term dependency (Remember information for long sequences and long periods) (Yu, Y. *et al*., 2019). Therefore, the LSTM cell was proposed to solve this problem, which is explained in the next section.

3.2 LSTM

Long short-term memory (LSTM) cell was proposed by many researchers to solve the long-term dependency problem. The solutions include adding gates with a specific structure to RNN cells to increase the ability of remembering the data (Smagulova and James, 2019). It has many variants such as LSTM cell without forget gate, LSTM cell with forget gate, and LSTM cell with a connection of peephole. Usually, most LSTM cells that are used in research are LSTM with forget gate.

Figure 3 below shows the LSTM cell without forget gate, it includes the input gate and output gate. It has three inputs, and two outputs, the cell state is an additional input. The mathematical calculations of this cell are:

$$i_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i) \quad (3)$$

$$\tilde{c}_t = tanh(W_{\tilde{c}h}h_{t-1} + W_c x_t + b\tilde{c}) \quad (4)$$

$$c_t = c_{t-1} + i_t \cdot \tilde{c}_t \quad (5)$$

$$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \quad (6)$$

$$h_t = o_t \cdot tanh(c_t) \quad (7)$$

where W denotes weights and c denotes LSTM cell state. The input gate can control the information that is stored in the cell state and the output of the output gate depends on the data in the cell state.

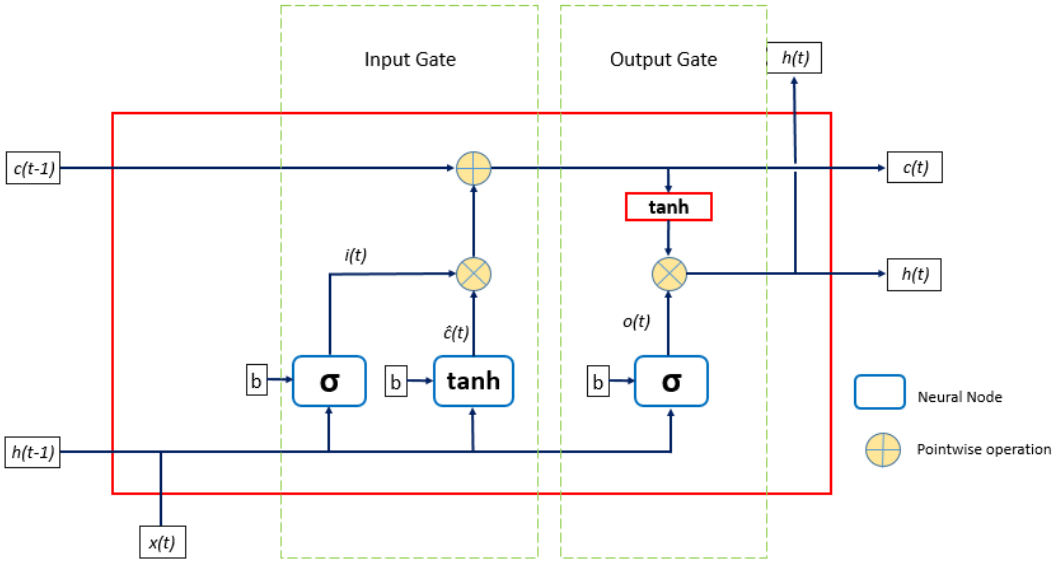

Figure 3. LSTM cell including input gate and output gate. It does not have forget gate. It has three inputs, and two outputs, the cell state is an additional input.

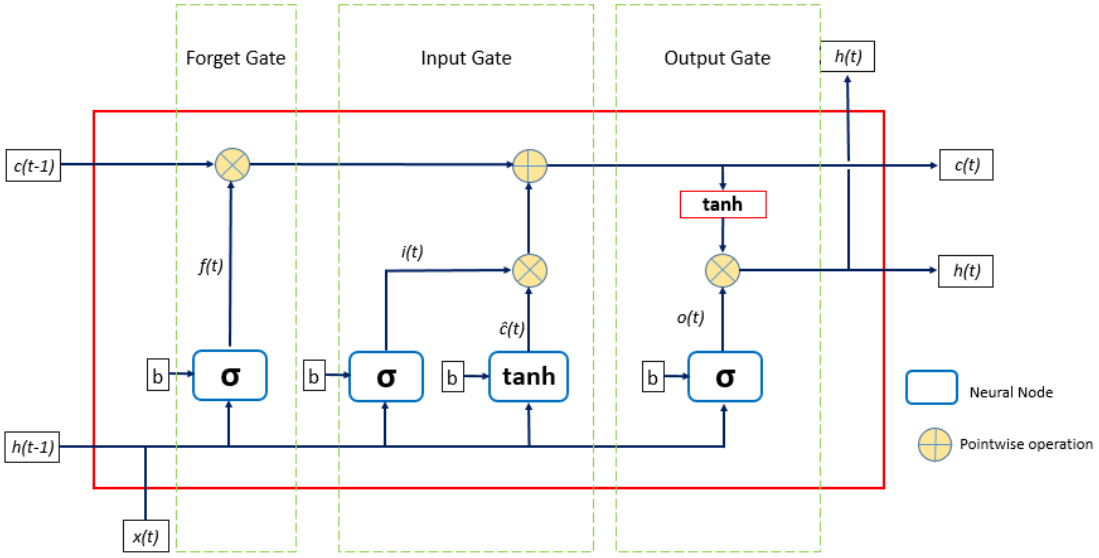

Figure 4. LSTM cell including input gate, output gate, and forget gate. It has three inputs, and two outputs. Input gate and output gate have *sigma* cell and *tanh* cell, forget gate has only *sigma* cell.

Figure 4 shows an LSTM cell with a forget gate. Input gate and output gate have *sigma* cell and $tanh$ cell, forget gate has only *sigma* cell. Forget gate was added to Basic LSTM cell and it decides which data should be excluded or included from cell state. When its value is 1 then the value of cell state is taken, on the other hand, when the value of this gate is 0 the value of cell state is excluded. Character b represents a bias. In some research, the performance of LSTM cells is improved by increasing the value of bias. The mathematical expressions of LSTM cell with forget gate are:

$$f_t = \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f) \quad (8)$$

$$i_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i) \quad (9)$$

$$\tilde{c}_t = tanh(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}x} + b_{\tilde{c}}) \quad (10)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \quad (11)$$

$$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \quad (12)$$

$$h_t = o_t \cdot tanh(c_t) \quad (13)$$

where c is a cell state and f_t is a value of forget gate (Yu, Y. *et al*., 2019).

3.3 BiLSTM

LSTM networks are classified into unidirectional LSTM, bidirectional LSTM (BiLSTM), and tree-structured LSTM. In the BiLSTM network two LSTM cells used common input and shared common output (Smagulova and James, 2019). BiLSTM network includes forward layer and backward layer and the training process is done in both directions at the same time. Usually, the output of BiLSTM is the output of merging the output of forward layer and the output of backward layer with adding suitable bias value. In many research, BiLSTM networks achieved better performance than LSTM networks and basic RNN networks such as determine the tag-of-speech (Yu, Y. *et al*., 2019). Figure 5 shows the BiLSTM network including the forward layer and the backward layer, and connections between layers to merge outputs for each layer.

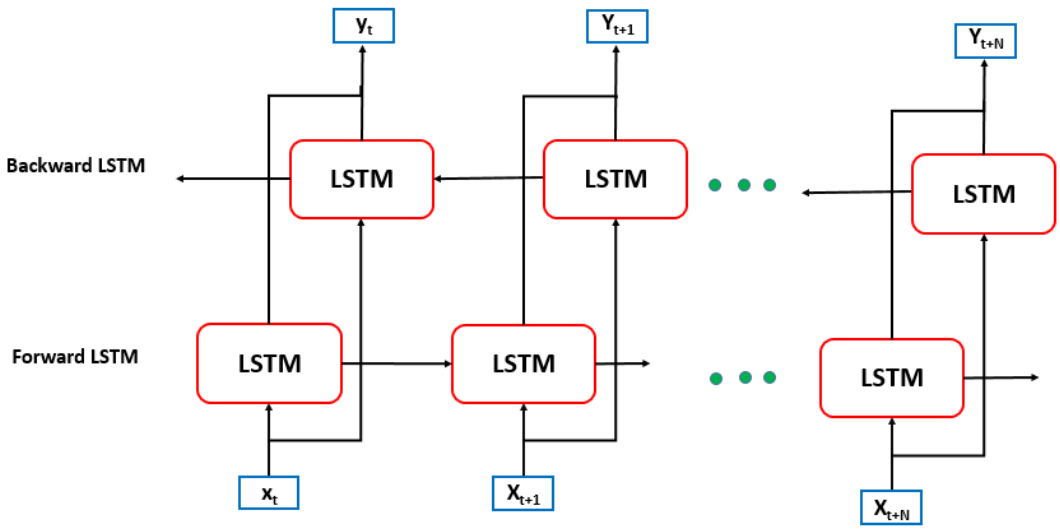

Figure 5. BiLSTM network including forward layer and backward layer, and the connections between these layers.

3.4 Deep Neural Networks

Deep neural networks lead to deep learning, which is a tool or field in machine learning for training data. In general, any neural network has an input layer, hidden layers,

and output layer. A neural network is called deep when the number of hidden layers is more than one (freeCodeCamp, 2020). The calculations of weights in neural networks become more complex due to increase in the number of hidden layers. It can be used in supervised or unsupervised training. Deep learning is used in several fields such as speech recognition, computer vision, translation, and natural language processing (NLP). Deep learning has several types such as convolutional neural networks (CNN), deep reinforcement learning, and recurrent neural networks. It is usually used to increase the performance of the training process and get a high accuracy (Wikipedia contributors, 2021). Therefore, I use a deep neural network in our research including 4 hidden layers of BiLSTM and 4 hidden layers of BiCuDNNLSTM.

3.5 Dropout

Dropout is a technique used in machine learning to avoid overfitting. Deep neural networks have a large number of parameters that leads to a huge amount of details and this leads to overfitting during the prediction process. Dropout was proposed and used in many deep neural networks such as convolutional neural networks (CNN) and recurrent neural networks (RNN). During training, some neurons are omitted by a certain probability. Figure 6 below shows an example of applying dropout with a probability of (0.5). The left neural network represents a fully connected neural network and a neural network on the right has a dropout with a probability of (0.5).

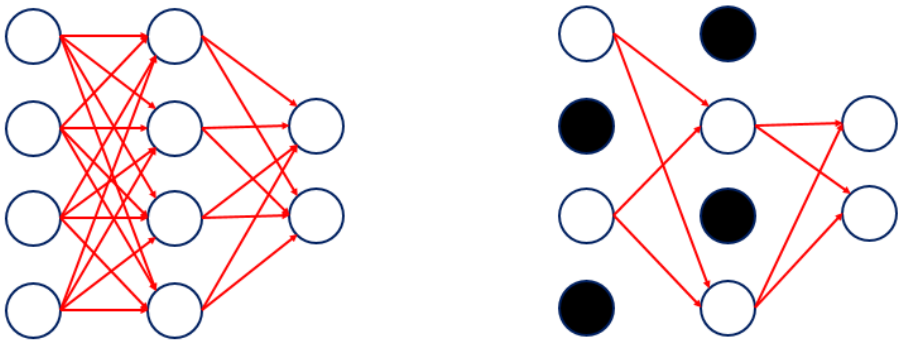

Figure 6. Full connection of neural network on the left side. On the right side: neural network with applying dropout by 0.5 probability.

Standard dropout act as an additional layer that set the values of some neurons to zero. This type of dropouts enables a long period of training without overfitting and gets good results of accuracy.

In RNN the use of standard dropout may lead to poor results and low performance, due to long-term memory. Therefore, there are special methods of dropout to use in recurrent layers, and they can solve long-term memory problem (Labach *et al*., 2019).

3.6 cuDNN, cuDNNLSTM, and BiCUDNNLSTM

NVIDIA cuDNN, abbreviation for NVIDIA CUDA deep neural network, is a library of GPU acceleration that is used in deep neural networks. It provides high-performance for training neural network layers and gives low training times when running on a GPU. It can be used in many frameworks of deep learning such as Tensorflow, Keras, PyTorch, Caffee2, and MATLAB (NVIDIA cuDNN, 2021).

cuDNNLSTM (CUDA deep neural network long short-term memory) is a fast implementation of LSTM with multiple network parameters, configurations, and input sizes. It is faster than the base LSTM implementation. Tensorflow, Keras, and PyTorch provide use of cuDNNLSTM (Braun, 2018). Tensorflow.keras provides using of cuDNNLSTM layer from the compatibility package (TensorFlow, 2021a).

Tensorflow provides a bidirectional layer that accepts the RNN layer as an argument. Therefore, it accepts LSTM, GRU, and CuDNNLSTM. In addition, the bidirectional layer uses merge_mode as an argument to combine the output of forward and backward layers for RNNs. Concatenation ('concat') is a default value for it. We used the bidirectional layer with cuDNNLSTM (BiCuDNNLSTM) in our models (TensorFlow, 2021b).

3.7 Word to Vector and FastText

Word to vector (Word2Vec) or word embedding is converting words to vectors, each vector includes real numbers. This method was used to make text classification such as sentiment analysis. The basic model that is used for this method includes three layers: embedding layer, flatten layer, and dense layer (Microsoft, 2020).

Fasttext is a library that is used to train a large amount of data in few minutes to make text classification. To use this library, it should be downloaded to use in the machine learning model. The training data should be saved in a text file with a specific format: (__label__target input data) (Joulin et al., 2016).

3.8 Evaluation Metrics and Tools

Evaluation metrics are used to measure the performance of machine learning models during the training process and the testing process. Classification in machine learning is used in several fields such as detection of hate speech on social media, categorization of YouTube video, face recognition, and text classification. CNNs and RNNs are considered the most popular models for classifications in machine learning (Minaee, 2019). I will explain some evaluation metrics that I used in this thesis:

1. **Classification accuracy**: accuracy is defined as the number of correct predictions divided by the total number of model's predictions.
2. **Loss**: is a metric to measure the error in models, if a model is considered perfect loss is zero. On the other hand, the loss is greater than zero.

3. **Training time**: time starts from the training of epochs until the end of epochs. It depends on the model type, number of epochs, the used hyperparameters and their values, number of layers, and number of epochs. The models have a good training time when their value is low (Geron, 2017).
4. **WER**: word error rate defined as the number of words that have an error in their diacritization divided by the total number of words.
5. **DER**: diacritization error rate defined as the number of characters with incorrect diacritization divided by the total number of characters.
6. **GPU/CPU utilization**: or CPU usage and it is defined as the amount of work processed by CPU or GPU.

The Tools that are used to measure these metrics are:

1. **Python code**: I used deep neural network models that it was used for training on the dataset to measure training accuracy, validation accuracy, test accuracy, WER, and DER.
2. **Nvidia-smi**: The NVIDIA system management interface is used to monitoring GPU devices. GPU utilization was measured by using the nvidia-smi command.
3. **Sysstat**: is a package or tool used to monitor and collect the system activity. It can record its measurements in reports. Sysstat can collect activity for CPU, IO devices, and memory. I used it to measure the utilization of the CPU (Kumar, 2020). Sysstat collects the data as text in specific files. After collecting Sysstat text files, I use the SARchart website to represent these data in line charts (SARchart, 2021).

Chapter 4: Experimental Setup and Methodology

In this chapter, I will explain the environment that I executed my models on it, models layers and hyperparameters, and a detailed description of datasets. In addition, the methodology was explained in the last part.

4.1 Experimental Setup

This section describes the experiment platform and the enhancement of the base model that I added to decrease the training time.

4.1.1 Experiments Platform

Table 1 shows platform specifications for experiments including CPU type and its characterizations, GPU type, memory size, and software environments such as operating system type, programing language, and libraries.

Table 1. Experiments platform specifications including CPU, GPU, memory, and software environments.

Components	Specifications
CPU	Intel Core i7-6700 CPU @ 3.40GHz CPUs (Cores): 8 L1d cache: 32K L1i cache: 32K L2 cache: 256K L3 cache: 8192K
Memory	32 GB DDR4-SDRAM @ 1066MHz
GPU	Nvidia GeForce RTX 2080 @ 2.1 GHz CUDA Cores: 2944
GPU Memory	Memory: 8GB
Docker	Version: 20.10.6
Libraries	Python 3.6.9 Tensorflow: 2.5.0 scikit-learn: 0.24.2 numpy: 1.19.5
OS	Ubuntu 20.04 LTS, 64-bit

I used GPU (Nvidia GeForce RTX 2080) to run a new model (which includes BiCuDNNLSTM) that decreased the training time. The previous research does not get improvement in Training time when using GPU (Abandah *et al*., 2020), except

diacritization experiment (Abandah and Abdel-Karim, 2020) and this was discussed and explained in chapter 5 (Results and discussion).

For the programming environment, I used a docker image that has Python 3.6.9, Tensorflow 2.5.0, and other packages.

4.1.2 Experiments Models

I applied my trials on two Arabic experiments that have different models:

4.1.2.1 Poem classification experiment

This experiment aimed to predict 17 classes of Arabic poems and prose. The state-of-the-art model of the previous research includes embedding layer, 4 BiLSTM layers, dense layer of 17 cells, and *softmax*. It is considered a deep RNN (Abandah *et al*., 2020). Our new model includes an embedding layer, 4 BiCuDNNLSTM layers, a dense layer of 17 cells, and *softmax*. It cannot be executed without GPU. So, the results of the experiments were compared in the GPU environment. Figure 7 shows the structures of stacked layers for both models; the previous model on the left with BiLSTM Layers, and the new model on the right with BiCuDNNLSTM layers.

In addition, I used the same hyperparameters from the previous model but there are some differences. Figure 8 shows the code model of BiLSTM with four layers and it is a previous model. Figure 9 shows the new code model that includes 4 layers of BiCuDNNLSTM. Both of them used 64 cells in each Bidirectional Layers, but BiCuDNNLSTM does not accept (mask_zero) parameter and dropout parameter. Masking is not supported for CuDNNLSTM Therefore, we excluded mask_zero from the embedding layer and add a dropout layer separately with a 0.1 value.

At model compile, *adam* optimizer was used. Accuracy and Loss Values were calculated for training and validation data. *Categorical_crossentropy* function was used to calculate loss value.

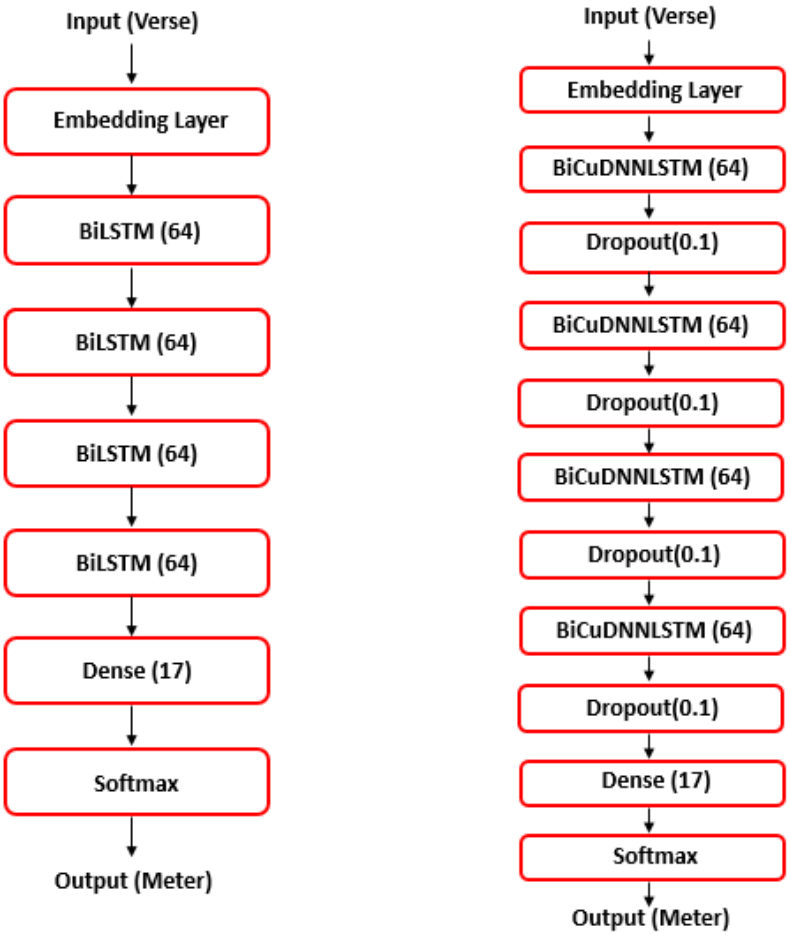

Figure 7. Stacked layers of the previous model: BiLSTM on the left and the new model BiCuDNNLSTM models on the right for poem classification experiment.

At training, the training data was split into two parts training data (85%) and validation data (15%). The batch size is 64 and callbacks are used to make early stopping when the improvement of validation accuracy stopped after 5 epochs. It is also used to save the model with best weights to use it for prediction later.

```
model = Sequential()
model.add(Embedding(num_tokens+1, 32, input_length=max_seq_length, mask_zero=True))
model.add(Bidirectional(LSTM(latent_dim, input_shape=(None,num_tokens), return_sequences=True,
            dropout=0.1, recurrent_dropout=0.3),
            merge_mode='concat'))
model.add(Bidirectional(LSTM(latent_dim, return_sequences=True,
            dropout=0.1, recurrent_dropout=0.3),
            merge_mode='concat'))
model.add(Bidirectional(LSTM(latent_dim, return_sequences=True,
            dropout=0.1, recurrent_dropout=0.3),
            merge_mode='concat'))
model.add(Bidirectional(LSTM(latent_dim,
            dropout=0.1, recurrent_dropout=0.3),
            merge_mode='concat'))
model.add(Dense(output_data.shape[1], activation='softmax'))
```

Figure 8. Code model of the previous model: BiLSTM with 4 layers and its parameters for poem classification experiment.

```
model = Sequential()
model.add(Embedding(num_tokens+1, 32, input_length=max_seq_length))
model.add(Bidirectional(CuDNNLSTM(latent_dim, input_shape=(None,num_tokens), return_sequences=True),
            merge_mode='concat'))
model.add(Dropout(0.1))
model.add(Bidirectional(CuDNNLSTM(latent_dim, return_sequences=True),
            merge_mode='concat'))
model.add(Dropout(0.1))
model.add(Bidirectional(CuDNNLSTM(latent_dim, return_sequences=True),
            merge_mode='concat'))
model.add(Dropout(0.1))
model.add(Bidirectional(CuDNNLSTM(latent_dim),
            merge_mode='concat'))
model.add(Dropout(0.1))
model.add(Dense(output_data.shape[1], activation='softmax'))
```

Figure 9. The code model of the new model: BiCuDNNLSTM with 4 layers and its parameters for poem classification experiment.

4.1.2.2 Diacritization experiment

This experiment aims to diacritize Arabic text. The prediction includes 16 classes of Diacritics for one letter. This experiment is considered sequence-to-sequence (seq-to-seq) experiment. Therefore, we used TimeDistributed layer at the end of the model.

The previous model (state-of-the-art) includes an input layer, embedding layer, 4 BiLSTM layers with 256 cells, and TimeDistributed layer with a dense of 16 cells (Abandah and Abdel-Karim, 2020). The new model includes the input layer, embedding layer, 4 BiCuDNNLSTM layers with 256 cells, and TimeDistributed layer with a dense of 16 cells. There are some differences that it same as poem classification experiment.

BiCuDNNLSTM does not accept masking and it does not include dropout parameter. Therefore, we exclude mask_zero from the embedding layer and add dropout with a value of (0.1). Figure 10 shows the layers in the previous model: BiLSTM with 4 layers on the left and the layers in the new model: BiCuDNNLSTM with 4 layers on the right.

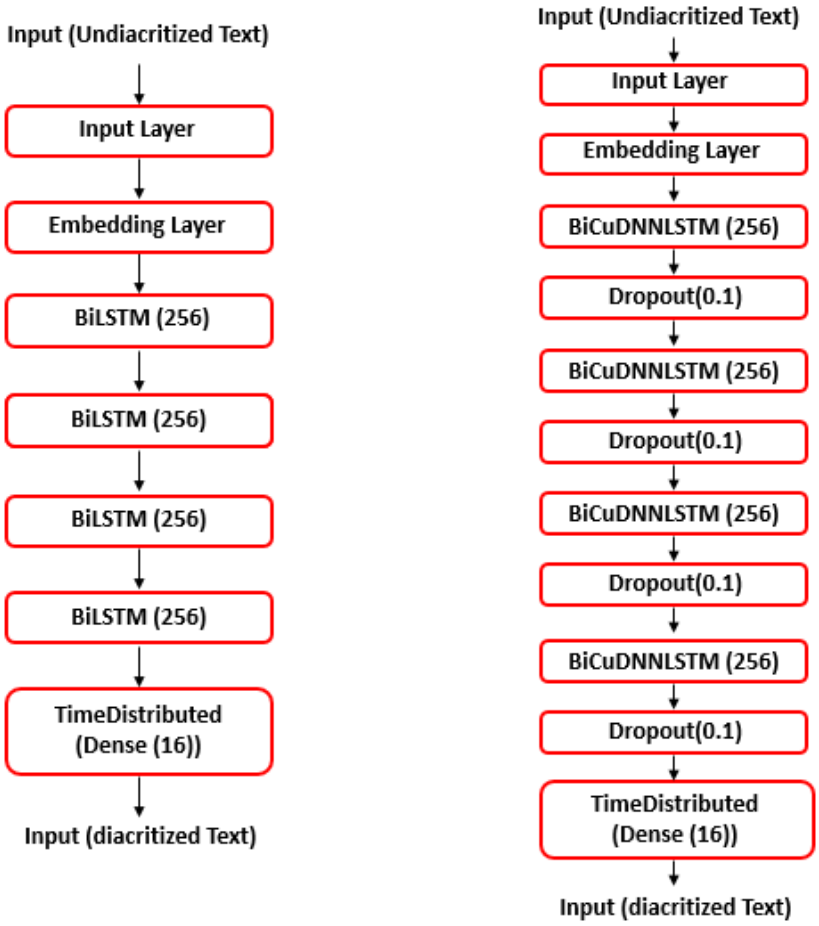

Figure 10. Stacked layers of the previous model: BiLSTM on the left and new model: BiCuDNNLSTM models on the right for diacritization experiment.

Figure 11 shows the previous model of diacritization experiment that includes 4 BiLSTM layers. Figure 12 shows the new model of diacritization experiment that includes 4 BiCuDNNLSTM layers. Each code was run in the GPU environment. At model compilation, the optimizer is *rmsprop*, and *categorical_crossentropy* function was used to calculate loss values. Accuracy metric also was added. Full execution includes 100 epochs, but callbacks were used and it includes early stopping with patience value

equal 5, and validation accuracy was used to determine the best epochs when applying early stopping.

```
verse_input = keras.Input(
    shape=(None,), name="title"
)
verse_features = layers.Embedding(num_tokens+1, 32, input_length=max_seq_length, mask_zero=True)(verse_input)
lstm_out = layers.Bidirectional(layers.LSTM(latent_dim, return_sequences=True,
            dropout=0.1, recurrent_dropout=0.3),
            merge_mode='concat') (verse_features)
lstm_out1 = layers.Bidirectional(layers.LSTM(latent_dim, return_sequences=True,
            dropout=0.1, recurrent_dropout=0.3),
            merge_mode='concat') (lstm_out)
lstm_out2 = layers.Bidirectional(layers.LSTM(latent_dim, return_sequences=True,
            dropout=0.1, recurrent_dropout=0.3),
            merge_mode='concat') (lstm_out1)
lstm_out3 = layers.Bidirectional(layers.LSTM(latent_dim, return_sequences=True,
            dropout=0.1, recurrent_dropout=0.3),
            merge_mode='concat') (lstm_out2)
diac_output = layers.TimeDistributed(layers.Dense(16, activation='softmax')) (lstm_out3)

model = keras.Model(
    inputs=[verse_input],
    outputs=[diac_output],
)
```

Figure 11. Code model of the previous model: BiLSTM with 4 layers and its parameters for diacritization experiment.

```
verse_input = keras.Input(
    shape=(None,), name="title"
)
verse_features = layers.Embedding(num_tokens+1, 32, input_length=max_seq_length)(verse_input)
lstm_out = layers.Bidirectional(CuDNNLSTM(latent_dim, return_sequences=True),
            merge_mode='concat') (verse_features)
Drop_out = layers.Dropout(0.1)(lstm_out)
lstm_out1 = layers.Bidirectional(CuDNNLSTM(latent_dim, return_sequences=True),
            merge_mode='concat') (Drop_out)
Drop_out1 = layers.Dropout(0.1)(lstm_out1)
lstm_out2 = layers.Bidirectional(CuDNNLSTM(latent_dim, return_sequences=True),
            merge_mode='concat') (Drop_out1)
Drop_out2 = layers.Dropout(0.1)(lstm_out2)
lstm_out3 = layers.Bidirectional(CuDNNLSTM(latent_dim, return_sequences=True),
            merge_mode='concat') (Drop_out2)
Drop_out3 = layers.Dropout(0.1)(lstm_out3)
diac_output = layers.TimeDistributed(layers.Dense(16, activation='softmax')) (Drop_out3)

model = keras.Model(
    inputs=[verse_input],
    outputs=[diac_output],
)
```

Figure 12. Code model of the new model: BiCuDNNLSTM with 4 layers and its parameters for diacritization experiment.

4.2 Datasets

In this section, I will describe the contents of datasets in both experiments:

4.2.1 Poem Classification Experiment Dataset

The used dataset of this experiment is the Arabic poem comprehensive dataset (APCD2), which was prepared from (APCD) dataset by making three modifications: remove verses that missed left or right halves, remove too long verse, and remove too short verses depending on a specific percentage. This dataset includes 1,657 K of poems and prose. It includes 16 classes of poetry meters and prose. The median of poem length is 5 and the range of poem length is between 1 to 2,367. Table 2 shows poetry meter classes and verses number for each meter (Abandah *et al*., 2020).

Table 2. Poetry meter classes and number of verses for each class.

No.	Meter	Number of Samples(Verses)
1	Ṭawīl	395,638
2	Kāmil	358,462
3	Basīṭ	235,606
4	Khafīf	151,784
5	Wāfir	130,918
6	Rajaz	103,059
7	Ramal	71,527
8	Mutaqārib	62,350
9	Sarīʿ	56,249
10	Munsariḥ	27,708
11	Mujtathth	15,718
12	Madīd	7,418
13	Hazaj	6,916
14	Mutadārik	4,204
15	Muqtaḍab	702
16	Muḍāriʿ	109

The dataset (APCD2) was split into 10% test set (test samples is 163,917) and 90% training set. Table 3 below shows details of the dataset including the number of samples, max sequence number, tokens number, classes number, and test samples number. The number of samples is 1,657,003 and the number of unique characters is 45. In addition, the max sequence length is 128.

Table 3. APCD2 dataset details include the number of samples, max sequence number, tokens number, classes number, and test samples number.

Features	Details
Number of samples	1,657,003
Max sequence length	128
Number of tokens	45
Number of classes	17
Test samples	163,917

4.2.2 Diacritization Experiment Dataset

Tashkeela dataset is a dataset that was used in this experiment. It includes 55k lines. These lines contain Arabic text from classical Arabic (CA) and Holy Quran. The dataset was split into 50K lines for the training set, 2,500 lines for the validation set, and 2,500 for the test set. Table 4 below shows Tashkeela dataset details including the count of words, count of sequences, average letters per word, and average words per sequence (Abandah and Abdel-Karim, 2020).

Table 4. Tashkeela dataset details including the count of words, count of sequences, letters per word, and words per sequence.

Features	Details
Words count	2,312 K
Sequences count	55 K
Letters per word	4.0
Words per sequence	42.1

I used the prepared Tashkeela dataset that its sequences were wrapped into 400 characters. After wrapping, the training set becomes 61,984, the validation set becomes 3,080, and the test set becomes 3,140. Table 5 below shows the wrapped dataset details that were obtained by executing the code of this experiment including the number of samples, max sequence length, number of input tokens, number of target tokens, and test samples. The number of unique input tokens is 72 and the number of target tokens is 8. Table 6 shows target tokens includes 8 diacritics: Fathatan, Dammatan, Kasratan, Fatha, Damma, Kasra, Sukun, Shadda.

Table 5. Tashkeela dataset details after wrapping include the number of samples, max sequence length, number of input tokens, number of target tokens, and test samples.

Features	Details
Number of samples	65,064
Max sequence length	400
Number of input tokens	72
Number of target tokens	8
Test Samples	3,140

Table 6. Target tokens include 8 diacritics: Fathatan, Dammatan, Kasratan, Fatha, Damma, Kasra, Sukun, Shadda.

0	1	2	3	4	5	6	7
Fathatan	Dammatan	Kasratan	Fatha	Damma	Kasra	Sukun	Shadda
◌ً	◌ٌ	◌ٍ	◌َ	◌ُ	◌ِ	◌ْ	◌ّ

4.3 Experimental Setup for Word to Vector and Fasttext

I applied these two methods in environment that includes these setups:

1. Processor: Intel Core i7-8550U CPU @ 1.80GHz 1.99 GHz.
2. Memory (RAM): 16 GB.
3. OS: Ubuntu 20.04.

However, the results are not accepted; I get low training time, but with bad values of accuracy and loss. Therefore, I did not execute these models in the same environment that I executed the base and the state-of-the-art models on it. I discuss their results in chapter 5.

4.4 Methodology

In this part, I explain the steps of making this thesis:

1. Studying computer performance evaluation. Study how to choose the metrics, how to measure them, and how to represent them in the best way.
2. Search on previous research to know how to make performance evaluation and characterization for DNNs models.

3. Search on previous research to find ways for optimizing DNNs, and search on fast LSTM Networks.
4. Measure the evaluation metrics of chosen Arabic application models by available tools, and characterize them in detail. In addition, the results will be represented by tables and graphs to clarify these results.
5. Apply the ways of optimization for DNN and determine the-state-of-the-art way that achieves the best training time.
6. The work and results will be documented. Then, it will be submitted to publish in available conferences and journals.

Chapter 5: Results and Discussion

This chapter includes the results of training and evaluation deep learning models and some performance analysis.

5.1 Poem Classification Results

Table 7 below shows the training time and the number of parameters of the BiLSTM model on CPU and training time and the number of parameters of BiLSTM and BiCuDNNLSTM models on GPU for 1 epoch for poem classification experiment. It shows the relation between the number of layers and training time for each model in different environments (CPU/GPU). Figure 13 below illustrates the results of Table 7.

Table 7. Training time and number of parameters when executing BiLSTM, BiCuDNNLSTM models for poem classification experiment with different number of layers and different environments (CPU/GPU)

Layers	BiLSTM			BiCuDNNLSTM	
	Parameters Number	Training time/CPU	Training time/GPU	Parameters Number	Training time
1	53,329	34.76 min	3.07 hrs.	53,841	3.13 min
2	152,145	1.26 hrs.	6.45 hrs.	153,169	6.01 min
3	250,961	1.93 hrs.	9.78 hrs.	252,497	9.10 min
4	349,777	2.60 hrs.	13.06 hrs.	351,528	12.10 min

Training time for 1 epoch for BiLSTM model has no improvement when running on GPU and it has a lower training time on CPU. However, the execution of the BiCuDNNLSTM model has the lowest training time and it is faster by 65 times compared with BiLSTM on CPU and BiLSTM on GPU models. The training time of 4 layers of the BiLSTM model has 13.06 hours on GPU and 2.60 hours on CPU. In addition, the number of parameters that is used for weights calculations is 349,777. On the other hand, the training time of four layers of BiCuDNNLSTM takes only 12.1 minutes with 351,528 parameters. The complexity of any model increases when the number of Layers increases. This is due to an increase in parameters number and this increases the complexity of calculations for weights.

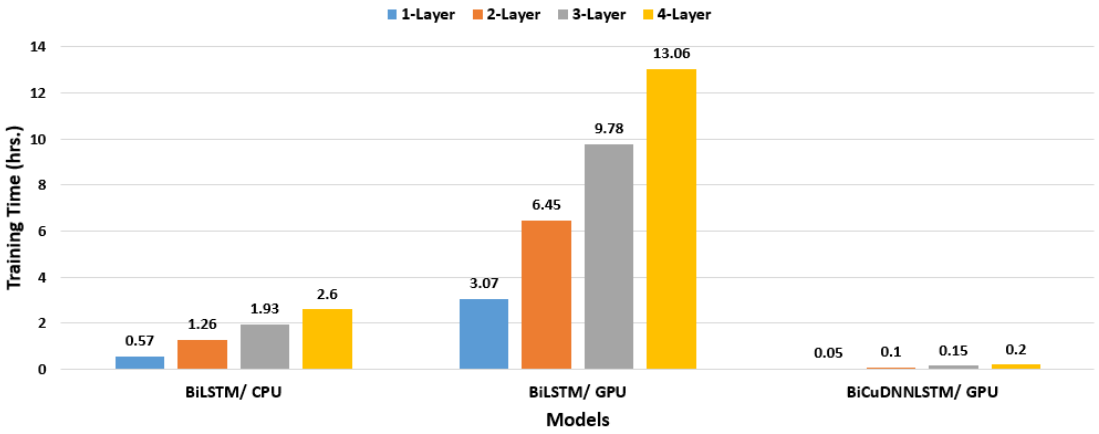

Figure 13. Training time for one epoch on the different number of layers (1, 2, 3, and 4) of poem classification experiment for models: BiLSTM/CPU, BiLSTM/GPU, BiCuDNNLSTM/GPU.

In addition, I made other experiments to calculate training time for CuDNNLSTM and BiCuDNNLSTM for the poem classification experiment. Table 8 shows training time and parameter numbers for each model. The relation between training time and parameters number is shown in figure 14 below.

Table 8. Training time, the number of parameters for layers (1, 2, 3, and 4) for three models: CuDNNLSTM/Dropout, BiCuDNNLSTM/ No Dropout, and BiCuDNNLSTM/ Dropout for poem classification experiment.

Layers	CuDNNLSTM/Dropout		BiCuDNNLSTM/No Dropout		BiCuDNNLSTM/Dropout	
	Parameters Number	Training Time	Parameters Number	Training time	Parameters Number	Training time
1	27,665	1.93 min	53,841	3.11 min	53,841	3.13 min
2	60,945	3.15 min	153,169	5.90 min	153,169	6.01 min
3	94,225	4.45 min	252,497	9.40 min	252,497	9.10 min
4	127,505	5.80 min	351,528	11.90 min	351,528	12.10 min

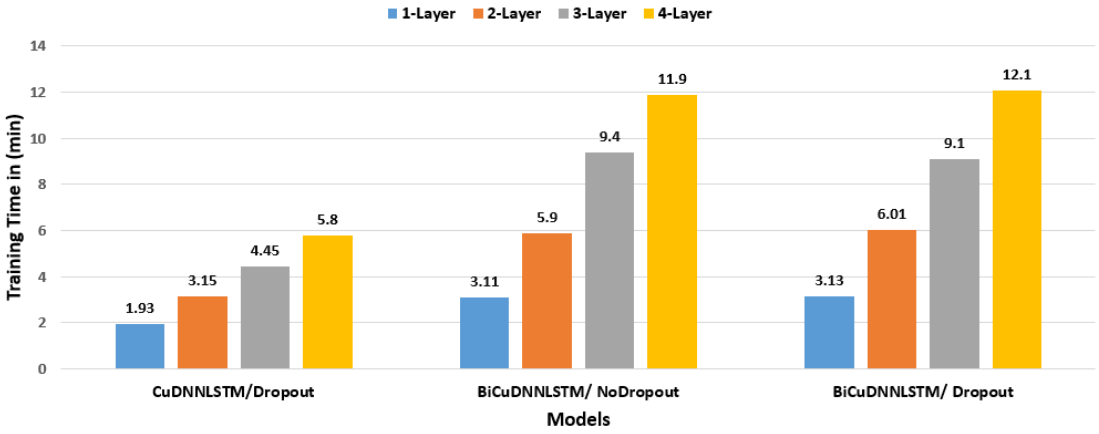

Figure 14. Relation between training time for one epoch, and layers (1, 2, 3, and 4) for three models: CuDNNLSTM/Dropout, BiCuDNNLSTM/ No Dropout, and BiCuDNNLSTM/ Dropout for poem classification experiment.

The training time of CuDNNLSTM model takes the lowest value compared it with BiCuDNNLSTM model. However, I got the best accuracy and loss when I executed BiCuDNNLSTM model. Therefore, we consider BiCuDNNLSTM model as a basic model to compare it with BiLSTM model.

To check the values of test accuracy and test loss and it does not change to the worst, I made a full execution of BiCuDNNLSTM model as a full experiment of BiLSTM model that was executed in the previous research. The number of total epochs is 100 epochs and using of early stopping with patience value equal five. Table 9 shows the number of executed epochs for each model, determine the best epoch which is lower than the number of epochs by 5, total training time, average time per epoch (total training time divided by epochs number), test loss, and test accuracy. The average training time per epoch of BiLSTM model is 2.59 hrs. While the average training time of BiCuDNNLSTM with dropout is 11.8 min. there is no big difference in training time and test accuracy between BiCuDNNLSTM/ No Dropout and BiCuDNNLSTM/ Dropout. Therefore, I consider BiCuDNNLSTM model, in general, the best. Test accuracy for CuDNNLSTM with four layers do not achieve a good accuracy (23%). In addition, CuDNNLSTM models of one layer and two layers have good accuracy values (95%, 96.5%). However, BiCuDNNLSTM model has higher accuracy (97%). Figure 15 shows test accuracy for each model with full execution in the poem classification experiment. The best test loss is 0.14 for BiCuDNNLSTM/No Dropout model and the worst loss is 2.18 for CuDNNLSTM models. Figure 16 shows test loss for each model. Figure 17 illustrates the average time per epoch for each model.

Table 9. Total epochs number of full execution, best epoch by early stopping, total training time, average time per epoch, test loss, and test accuracy values for models: BiLSTM/CPU, BiCuDNNLSTM, and CuDNNLSTM for poem classification experiment.

Model	Layers	Epochs	Best Epoch	Total Training Time	Average time per epoch	Test Loss	Test Accuracy
BiLSTM/CPU	4	38	33	98.71 hrs.	2.59 hrs.	0.1485	97.29
CuDNNLSTM/ Dropout	1	36	31	1.13 hrs.	1.88 min	0.2019	95.02
CuDNNLSTM/ Dropout	2	30	25	1.55 hrs.	3.10 min	0.1614	96.49
CuDNNLSTM/ No Dropout	4	6	1	33.35 min	5.55 min	2.1820	23.33
CuDNNLSTM/ Dropout	4	6	1	34.13 min	5.68 min	2.1810	23.33
BiCuDNNLSTM/ No Dropout	4	17	12	3.33 hrs.	11.77 min	0.1400	97.17
BiCuDNNLSTM/ Dropout	4	19	14	3.73 hrs.	11.80 min	0.1427	97.17

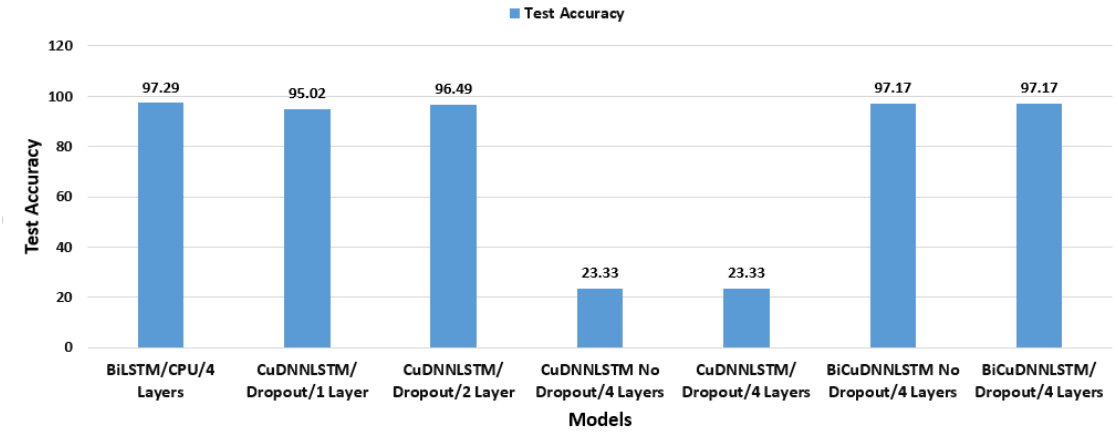

Figure 15. Test accuracy for BiLSTM, CuDNNLSTM, and BiCuDNNLSTM models with full execution in poem classification experiment.

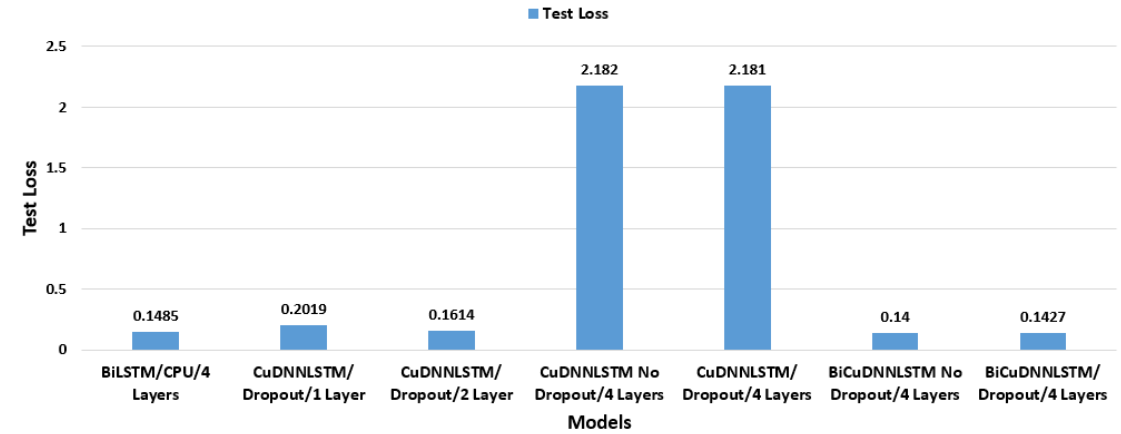

Figure 16. Test Loss for BiLSTM, CuDNNLSTM, and BiCuDNNLSTM models with full execution in poem classification experiment.

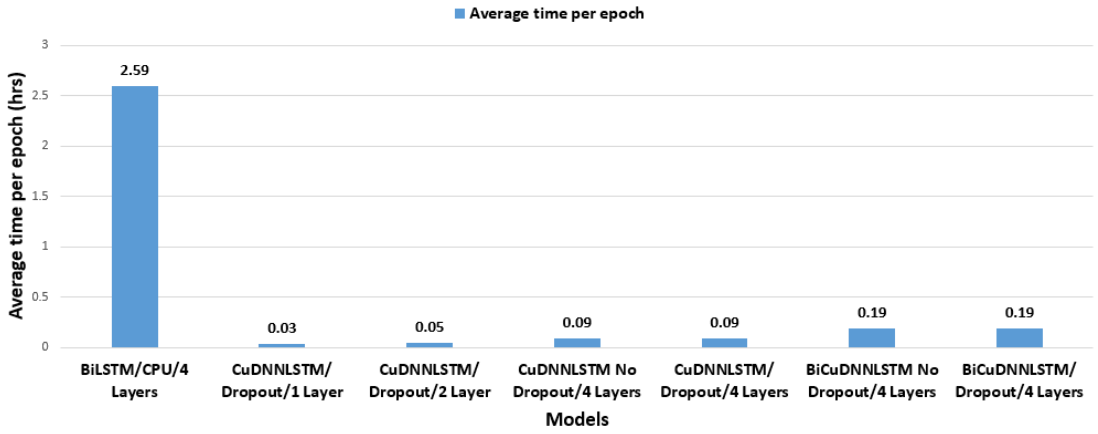

Figure 17. Average training time per epoch for BiLSTM, CuDNNLSTM, and BiCuDNNLSTM models with full execution in poem classification experiment.

5.2 Diacritization Results

I apply the same layer BiCuDNNLSTM to diacritization experiment. Table 10 shows the training time of four models: BiLSTM/CPU, BiLSTM/GPU, BiCuDNNLSTM/No Dropout, and BiCuDNNLSTM/Dropout. In addition, each experiment was executed 4 times with a different number of layers (from 1 to 4). BiCuDNNLSTM model took 4.26 min with four layers. While BiLSTM model on GPU took 1.5 hrs. BiCuDNNLSTM is faster by 21 times than BiLSTM/GPU model.

Table 10. The training time of one epoch for models: BiLSTM and BiCuDNNLSTM with layers number: 1, 2, 3, and 4 and parameters number in diacritization experiment.

Layers	BiLSTM			BiCuDNNLSTM		
	Parameters Number	CPU Training Time	GPU Training time	Parameters Number	Training Time/ No Dropout	Training Time/ Dropout
1	602,416	0.46 hrs.	0.26 hrs.	604,464	44.47 sec	48.53 sec
2	2,177,328	1.36 hrs.	0.60 hrs.	2,181,424	1.86 min	1.96 min
3	3,752,240	2.24 hrs.	1.04 hrs.	3,758,384	3.00 min	3.10 min
4	5,327,152	3.10 hrs.	1.50 hrs.	5,335,344	4.75 min	4.26 min

Figure 18 illustrates the relation between the number of layers and training time on BiLSTM and BiCuDNNLSTM models. The number of parameters increases by

increasing the number of layers. This affects the complexity of the model, so training time is increased when increasing the number of layers.

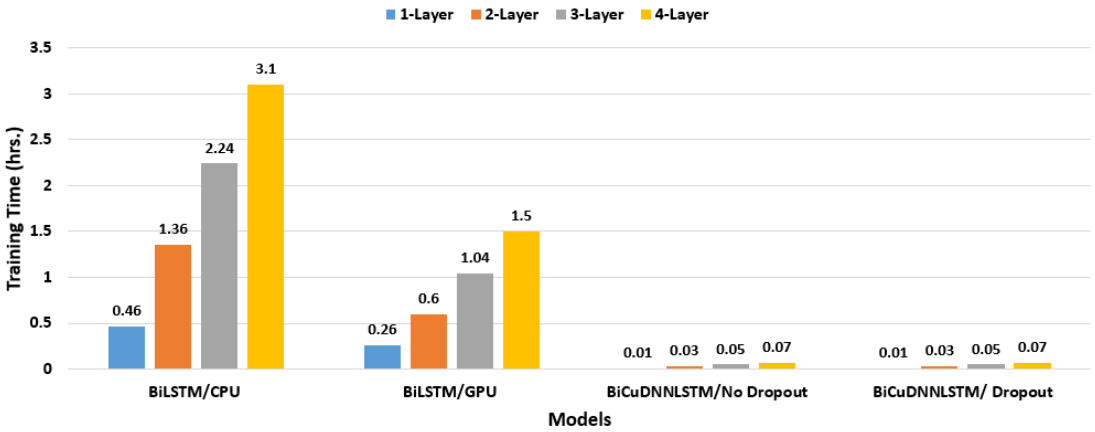

Figure 18. Relation between training time and layers: 1, 2, 3, and 4 on models: BiLSTM/CPU, BiLSTM/GPU, and BiCuDNNLSTM in diacritization experiment.

To verify that BiCuDNNLSTM model has good accuracy and Loss values, I apply a full execution for diacritization experiment. The execution includes 100 epochs with early stopping and patience value equal to 5. Table 11 shows total epochs number that was executed, a best epoch which is lower than epochs number by 5, total training time for all epochs, average time per epoch (total training time divided by epochs number), loss on the validation set, and accuracy on the validation set.

Table 11. Total epochs number of full execution, best epoch by early stopping, total training time, average time per epoch, validation loss, and validation accuracy values for models: BiLSTM/GPU, BiCuDNNLSTM, and CuDNNLSTM with 4 layers for diacritization experiment.

Model	Epochs	Best Epoch	Total Training Time	Average time per epoch	Val Loss for best epoch	Val Accuracy for best epoch
BiLSTM/ GPU	56	51	84.50 hrs.	1.50 hrs.	0.0560	98.41
CuDNNLSTM/ No Dropout	9	4	14.40 min	1.60 min	0.1647	80.44
CuDNNLSTM/ Dropout	28	23	48.30 min	1.72 min	0.1361	48.65
BiCuDNNLSTM/ No Dropout	17	12	1.15 hrs.	4.08 min	0.0341	99.06
BiCuDNNLSTM	25	20	1.73 hrs.	4.17 min	0.0295	99.16

BiLSTM model on GPU environment achieved 98.41% of Validation Accuracy. The experiment of BiCuDNNLSTM model with dropout has the best Validation Accuracy equal 99.16%. The total epochs number is 25 that means the best epoch (which has the best Validation Accuracy) is 21. CuDNNLSTM model that does not include a Bidirectional Layer does not achieve a good accuracy. Therefore, BiCuDNNLSTM with dropout model is considered the best model according to its Accuracy value. The average time per epoch takes 1.5 hrs. for BiLSTM/GPU model. While average time per epoch for BiCuDNNLSTM takes 4.17 min.

Figure 19 shows validation accuracy for each model with full execution in diacritization experiment. It shows that the highest accuracy is (99.16%) for BiCuDNNLSTM/Dropout model and the lowest accuracy is 48.65% for CuDNNLSTM/ Dropout model. Figure 20 shows the validation loss for each model. The best validation loss is 0.02 for BiCuDNNLSTM/Dropout model and the worst loss is 0.1647 for CuDNNLSTM/ No Dropout model. Figure 21 illustrates the average time per epoch for each model. It shows a big difference in training time between BiLSTM model and BiCuDNNLSTM model.

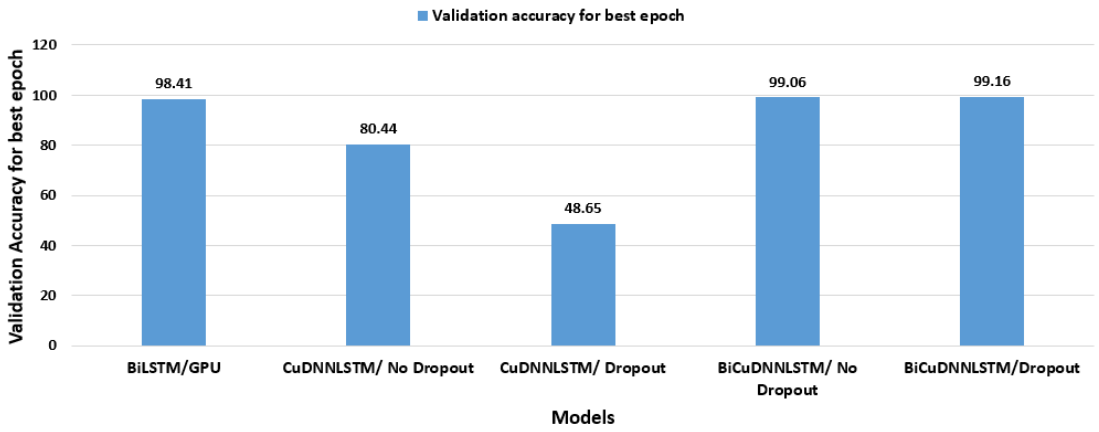

Figure 19. Validation accuracy for best epoch for BiLSTM, CuDNNLSTM, and BiCuDNNLSTM models with full execution in diacritization experiment.

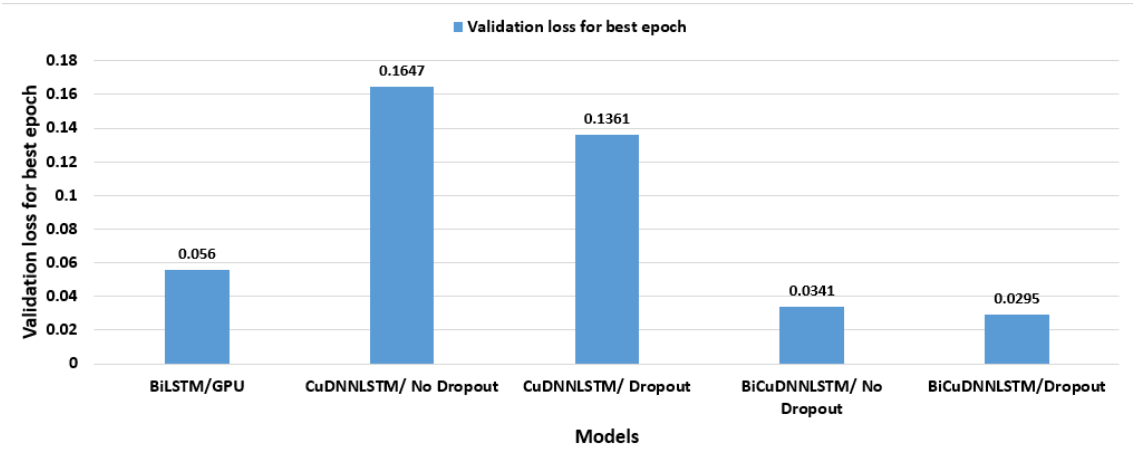

Figure 20. Validation loss for best epoch for BiLSTM, CuDNNLSTM, and BiCuDNNLSTM models with full execution in diacritization experiment.

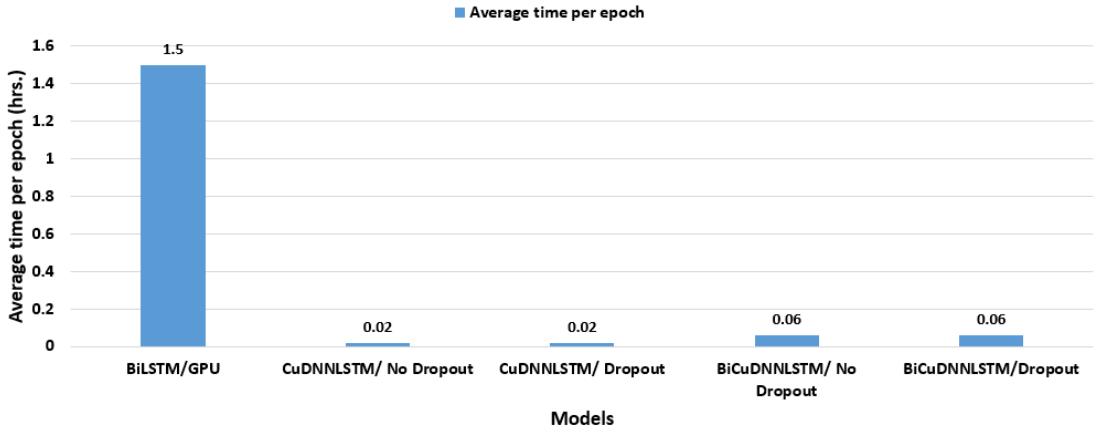

Figure 21. Average time per epoch for BiLSTM, CuDNNLSTM, and BiCuDNNLSTM models with full execution in diacritization experiment.

5.3 Performance Evaluation

This Section contains some characteristics of the training models that I applied in the poem classification and diacritization experiments.

5.3.1 Error Rates for Diacritization Experiment

After making full executions for diacritization experiments (100 epochs and early stopping with 5-patience value), models were tested to calculate WER and DER. Table 12 shows DER, Corrected DER (by post-processing and excluding some errors that are counted by DER such as sukun errors), and WER for BiLSTM and BiCuDNNLSTM models. WER and DER do not have better value compared with the previous model (BiLSTM). Figure 22 illustrates DER and WER values for each model. The best value of

DER is 1.97% for BiLSTM model while it is 2.32% for BiCuDNNLSTM/ Dropout. The best value of WER is 5.18% for BiLSTM/GPU model and it is 6.12% for BiCuDNNLSTM/ Dropout. BiCuDNNLSTM/ No Dropout model achieved the highest values of DER and WER.

Table 12. DER, corrected DER, WER for diacritization experiment for model: BiLSTM and BiCuDNNLSTM.

Evaluation Metrics	BiLSTM/GPU	BiCuDNNLSTM/ No Dropout	BiCuDNNLSTM/ Dropout
Sentences	3117	3140	3140
DER	1.98%	2.70%	2.35%
Corrected DER	1.97%	2.66%	2.32%
Words	125,099	125,106	125,106
Word error	6,485	8,832	7,658
WER	5.18%	7.05%	6.12%

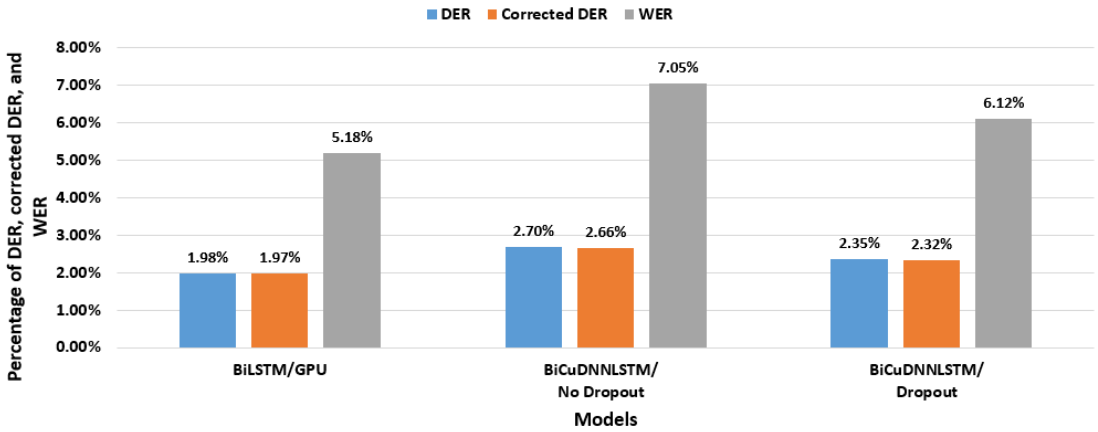

Figure 22. DER, corrected DER, and WER for models: BiLSTM/GPU, BiCuDNNLSTM/ No Dropout, and BiCuDNNLSTM/ Dropout for diacritization experiment.

5.3.2 CPU/GPU Utilization

I monitor and measure the utilization of CPU by Sysstat tool and we use Nvidia-smi command to measure the utilization of GPU.

1. CPU/GPU utilization for poem classification experiment: Table 13 shows average CPU utilization for BiLSTM/CPU and BiLSTM/GPU. In addition, it shows average CPU/GPU utilization for BiCuDNNLSTM model. These experiments were done for 1 epoch and four layers for each model. BiLSTM/CPU has the highest CPU

Utilization (85.95%), while it is low in BiLSTM/GPU in GPU environment (28.81%). Average GPU utilization for BiCuDNNLSTM is 66.56% on the other hand average CPU utilization is low (11.41%). Figure 23 illustrate the average CPU/GPU utilization for the poem classification experiment and its models.

Table 13. Average CPU/GPU utilization of models: BiLSTM/CPU, BiLSTM/GPU, and BiCuDNNLSTM with 4 layers for 1 epoch in poem classification experiment.

Average CPU Utilization			Average GPU Utilization
BiLSTM/CPU	**BiLSTM/GPU**	**BiCuDNNLSTM**	**BiCuDNNLSTM**
85.95%	28.81%	11.41%	66.56%

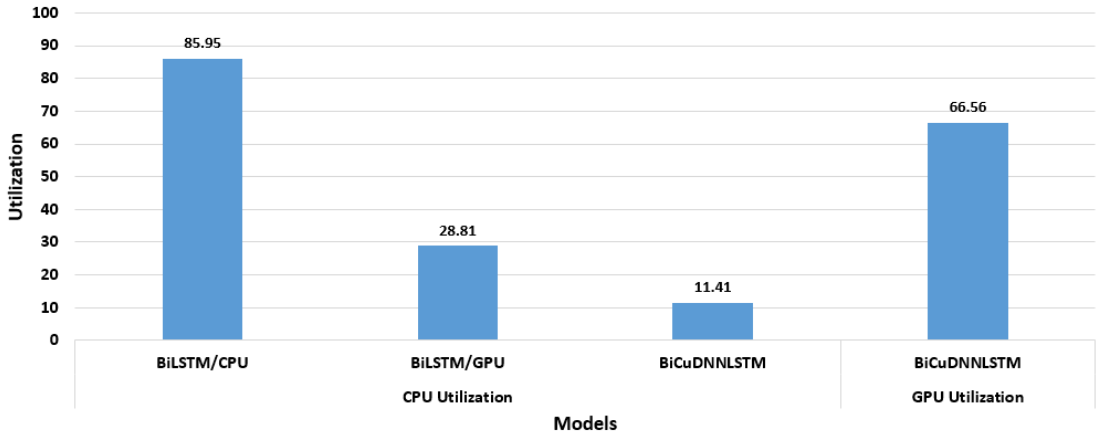

Figure 23. Average CPU/GPU utilization of models: BiLSTM/CPU, BiLSTM/GPU, and BiCuDNNLSTM with 4 layers for 1 epoch in poem classification experiment.

Figure 24 shows the utilization of CPU at the user level for BiLSTM/CPU model. The execution is done on 1 epoch and 4 layers. The program time for this experiment is 2.74 hrs. Therefore, I recorded 166 values of utilization. The period between two values is 60 seconds. At the beginning of the experiment, the utilization of the CPU start approximately at 8% after that and when start training utilization of the CPU increased to 87% and this was fixed until training finishes. Finally, CPU utilization decreased to 0.5%.

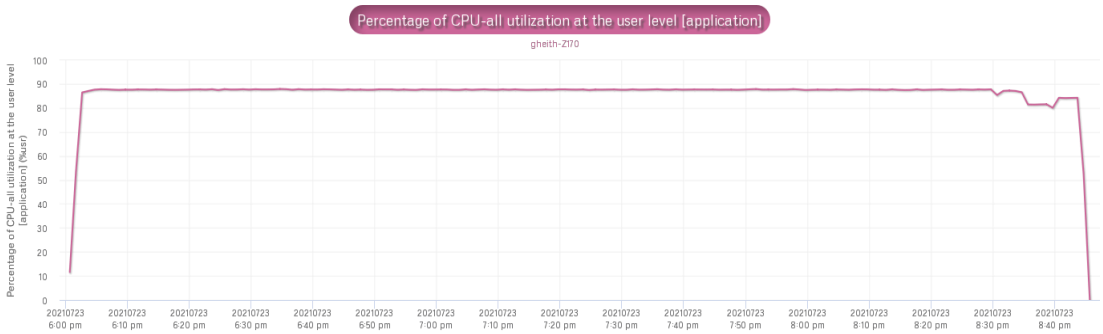

Figure 24. CPU utilization at user level for BiLSTM/CPU model, with 4 layers and 1 epoch for poem classification experiment.

Figure 25 shows CPU utilization of BiLSTM/GPU for poem classification experiment. The experiment is done for BiLSTM model for 4 layers and 1 epoch. The program time for this experiment is 13.69 hrs. Therefore, I recorded 79 values of utilization and the period between two values is 10 min (I record a value of CPU utilization every 10 min by Sysstat tool). The utilization of CPU in the GPU experiment did not exceed 30%. In addition, the GPU was not much utilized (between 22 to 32% during training).

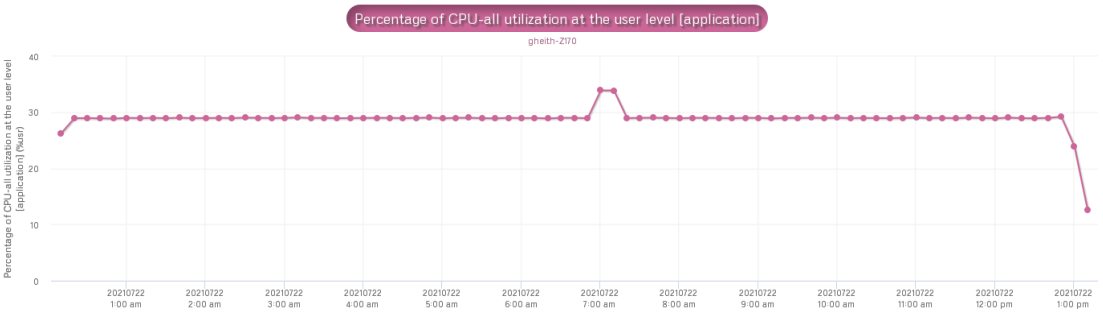

Figure 25. CPU utilization at user level for BiLSTM/GPU model, with 4 layers and 1 epoch for poem classification experiment.

Figure 26 shows CPU utilization in the poem classification experiment for BiCuDNNLSTM. This experiment includes training of 1 epoch of 4 BiCuDNNLSTM layers. The program time is 15.18 min. I recorded 89 values and the time between recorded values is 10 sec. Utilization of the CPU does not exceed 14%.

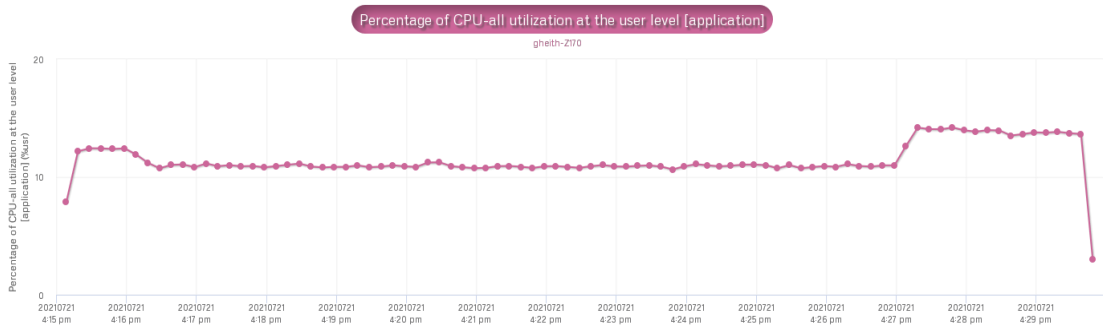

Figure 26. CPU utilization at user level for BiCuDNNLSTM model, with 4 layers and 1 epoch for poem classification experiment.

Figure 27 shows GPU Utilization in poem classification experiment for BiCuDNNLSTM model. I used Nvidia-smi as a tool to measure GPU utilization every 10 sec. Therefore, I recorded 92 values of GPU utilization. The max value of GPU utilization is 77% and the average value is 66.56%.

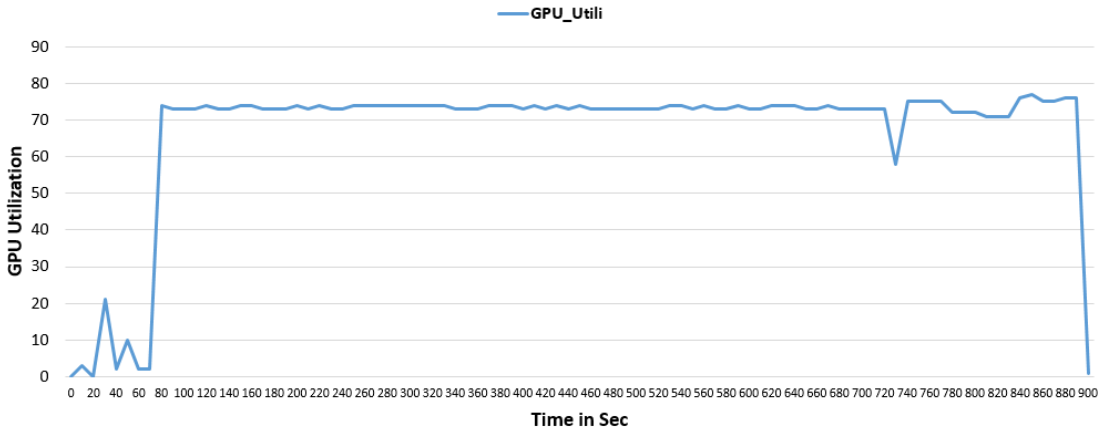

Figure 27. GPU utilization for BiCuDNNLSTM model, with 4 layers and 1 epoch for poem classification experiment.

2. CPU/GPU utilization for diacritization experiment: Table 14 shows average CPU utilization for BiLSTM/ CPU and BiCuDNNLSTM models. In addition, it shows average GPU utilization for BiLSTM/GPU and BiCuDNNLSTM models. Each experiment for one epoch and 4 layers. BiLSTM/CPU can utilize CPU by high value (88.13%). However, BiLSTM/GPU utilized the GPU by 35%. BiCuDNNLSTM model utilized the CPU by low value (9.23%) and it utilized GPU by 67.31%. Figure 28 illustrates the average CPU/GPU utilization of each model for diacritization experiment.

Table 14. Average CPU utilization for BiLSTM/CPU and BiCuDNNLSTM, average GPU utilization for BiLSTM/GPU and BiCuDNNLSTM for 1 epoch with 4 layers in diacritization experiment.

Average CPU Utilization		Average GPU Utilization	
BiLSTM/CPU	**BiCuDNNLSTM**	**BiLSTM/GPU**	**BiCuDNNLSTM**
88.13%	9.23%	35%	67.31%

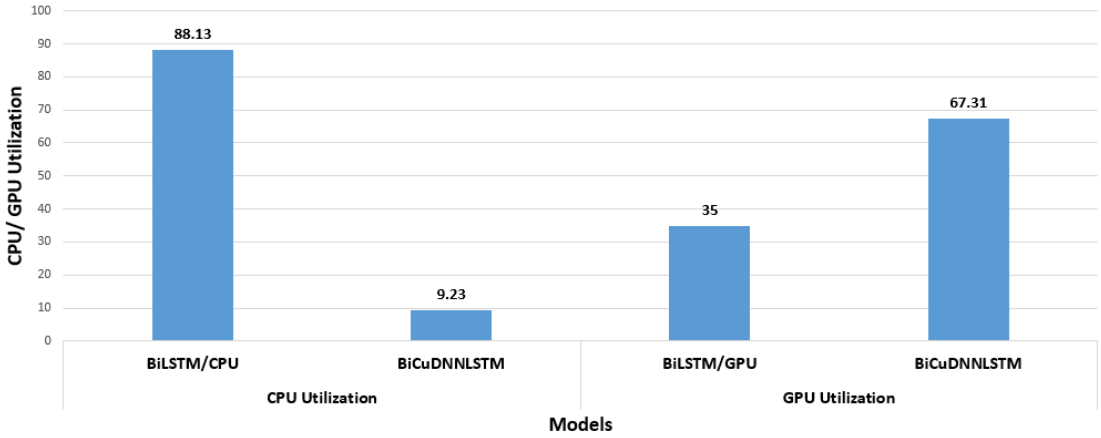

Figure 28. Average CPU utilization for BiLSTM/CPU and BiCuDNNLSTM, average GPU utilization for BiLSTM/GPU and BiCuDNNLSTM for 1 epoch with 4 layers in diacritization experiment.

Figure 29 shows CPU utilization for BiLSTM/CPU model with 4 layers and 1 epoch for diacritization experiment. The utilization in user level does not exceed 89%. I applied the experiment and measured Utilization by the Sysstat tool. Program time for 1 epoch is 3.10 hrs. Therefore, I record 185 values and the period between recorded values is 60 sec.

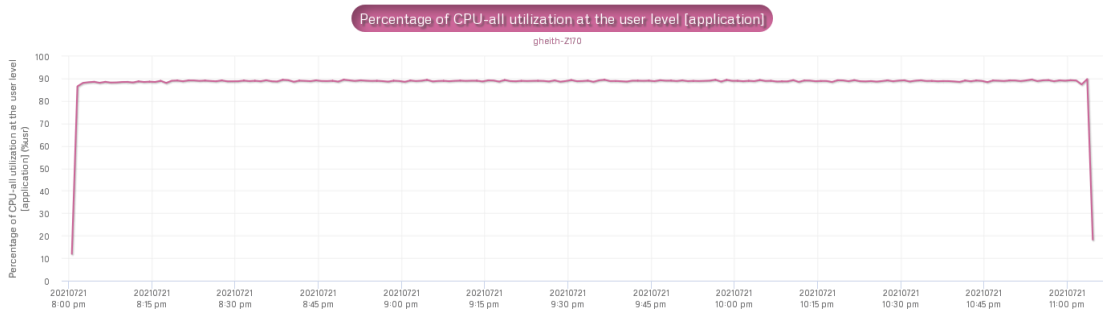

Figure 29. CPU utilization at user level for BiLSTM/CPU with 4 layers for 1 epoch in diacritization experiment.

Figure 30 shows CPU utilization in diacritization experiment for BiCuDNNLSTM model for 1 epoch and 4 layers. Program time is 4.88 min. I recorded 60 values by Sysstat

and the time between recorded values is 5 sec. The utilization does not exceed 13% at the user level.

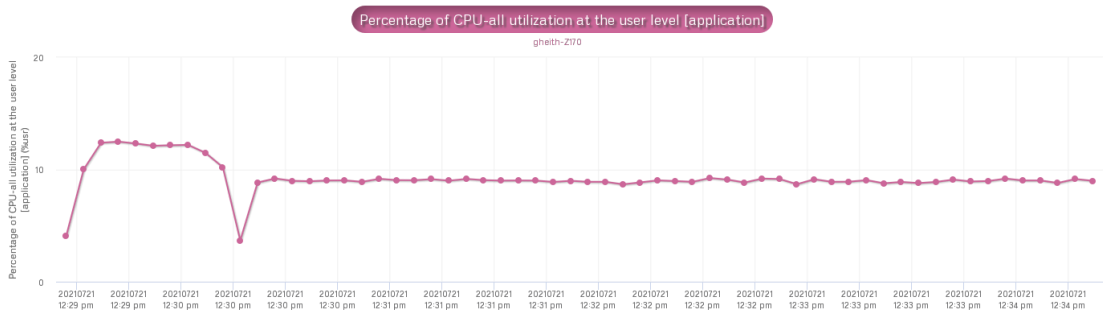

Figure 30. CPU utilization at user level for BiCuDNNLSTM with 4 layers for 1 epoch in diacritization experiment.

Figure 31 shows GPU utilization for BiCuDNNLSTM model for diacritization experiment. I used Nvidia-smi Command to measure GPU utilization. I recorded 65 values of GPU utilization. The maximum value of GPU utilization is 86% and the average GPU utilization is 67.31%.

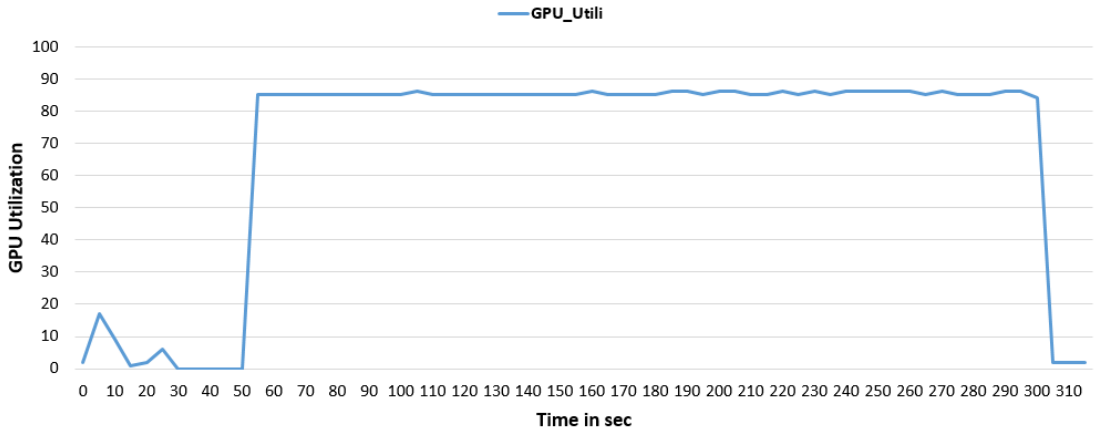

Figure 31. GPU utilization for BiCuDNNLSTM with 4 layers for 1 epoch in diacritization experiment.

5.4 Word to Vector and FastText Results

I execute the word to vector and fasttext experiments for 2 epochs in poem classification experiment. Table 15 shows the average training time per epoch, test accuracy, and test loss for the BiLSTM model and word to vector model, fasttext model. The BiLSTM model has the highest average training time equal to 2.87 hrs.. and the word to vector model has an average training time for epoch equal to 1.06 hrs.. Fasttext model

has a lower training time per epoch equal to 10.98 seconds. The word to vector and fasttext models have bad accuracy values (65.86% and 43.26%). While the BiLSTM model has a good accuracy equal to 96.37%. Therefore, I do not consider the word to vector model and fasttext model as improved models because I do not get good accuracy values for them. Figure 32 shows the average training time per epoch for BiLSTM, word to vector, and fasttext experiments. Figure 33 illustrates the test accuracy values in table 15 for these three experiments.

Table 15. The average training time per epoch, test accuracy, and test loss for the BiLSTM model and word to vector for 2 epochs in poem classification experiment.

Model	Average Training Time per Epoch	Test Accuracy	Test Loss
BiLSTM (4 Layers)	2.87 hrs.	96.37	0.1657
Word to Vector	1.06 hrs.	65.86	1.0666
Fasttext	10.98 sec	43.26	1.0140

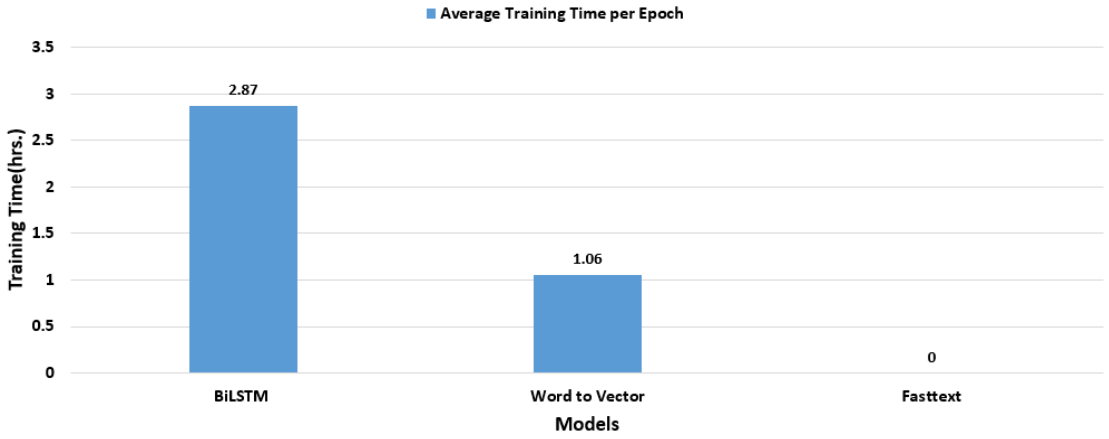

Figure 32. Training time in hours for BiLSTM, word to vector, and fasttext models in poem classification experiment.

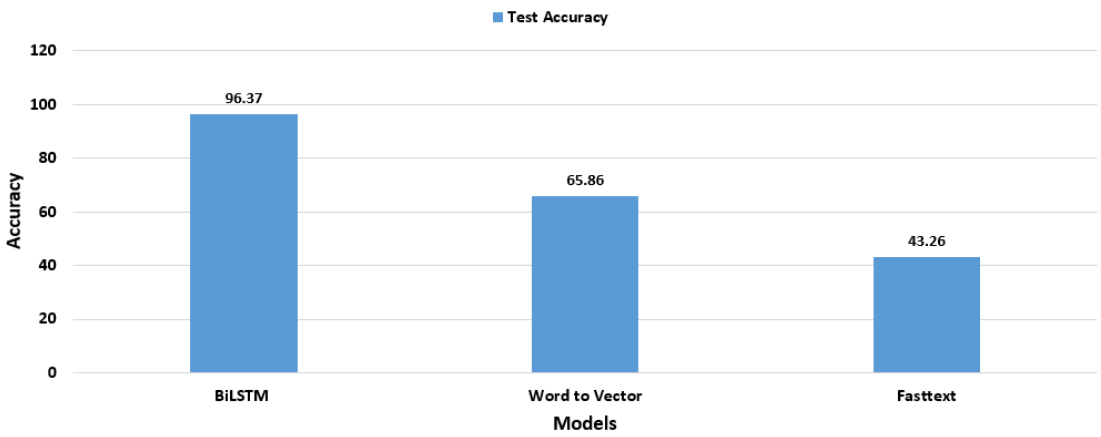

Figure 33. Test accuracy for BiLSTM, word to vector, and fasttext models in poem classification experiment.

Chapter 6: Conclusion and Future Works

This chapter includes a conclusion of my results of training and evaluating RNNs and future works for performance improvement and evaluation for DNN.

6.1 Conclusion

I made an improvement for training time of training deep RNNs for Arabic applications. The experiments are poem classification with APCD2 dataset and text diacritization with Tashkeela dataset. I got a low training time when running my models on the GPU environment and using CuDNNLSTM and BiCuDNNLSTM layers. State-of-the-art model is BiCuDNNLSTM with 4 layers. It achieved training time faster by 65 times than the BiLSTM model with an accuracy of 97.17% and training time faster by 21 times than the BiLSTM model. Word to vector and fasttext models achieved improvement in training time with a lower value of accuracy for poem classification experiment.

In addition, I make performance evaluations for these models and measure evaluation metrics: test accuracy, loss accuracy, validation accuracy, validation loss, WER, DER, CPU utilization, and GPU utilization. BiCuDNNLSTM model utilized the GPU with a percentage of 66.56% in the poem classification experiment and 67.31% in the diacritization experiment. I used python code, Sysstat, and Nvidia-smi to measure these metrics.

6.2 Future Works

Future works for my thesis are applying new methods and models to enhance training time for Arabic applications and searching for new RNNs layers that may improve training time in CPU environment and GPU environment. In addition, applying fine-tuning for models by changing their hyperparameters to improve its accuracy and other metrics and keep it with suitable values.

In addition, measure and monitor other system components such as memory usage and IO devices to characterize and understand these models.

REFERENCES

Abandah, G., & Abdel-Karim, A. (2020). Accurate and Fast Recurrent Neural Network Solution for the Automatic Diacritization of Arabic Text. **Jordanian Journal of Computers and Information Technology (JJCIT)**, 6(2), 103-121.

Abandah, G. A., Khedher, M. Z., Abdel-Majeed, M. R., Mansour, H. M., Hulliel, S. F., & Bisharat, L. M. (2020). Classifying and Diacritizing Arabic Poems Using Deep Recurrent Neural Networks. **Journal of King Saud University-Computer and Information Sciences**.

Arcos-Garcia, A., Alvarez-Garcia, J. A., & Soria-Morillo, L. M. (2018). Evaluation of Deep Neural Networks for Traffic Sign Detection Systems. **Neurocomputing**, 316, 332-344.

Awan, A. A., Subramoni, H., & Panda, D. K. (2017). An in-Depth Performance Characterization of CPU-and GPU-based DNN Training on Modern Architectures. **In Proceedings of the Machine Learning on HPC Environments.** 1-8.

Braun, S. (2018). LSTM Benchmarks for Deep Learning Frameworks. **arXiv preprint arXiv**:1806.01818.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., & Shelhamer, E. (2014). cudnn: Efficient Primitives for Deep Learning. **arXiv preprint arXiv**:1410.0759.

Dai, X., Yin, H., & Jha, N. K. (2018). Grow and Prune Compact, Fast, and Accurate LSTMs. **arXiv preprint arXiv**:1805.11797.

Education, I. C. (2021, April 7). Recurrent Neural Networks. **IBM Cloud Education**. https://www.ibm.com/cloud/learn/recurrent-neural-networks.

FreeCodeCamp. (2020, March 31). Want to Know How Deep Learning Works? Here's a Quick Guide for Everyone. https://www.freecodecamp.org/news/want-to-know-how-deep-learning-works-heres-a-quick-guide-for-everyone-1aedeca88076/.

Geron, A. (2017). **Hands-On Machine Learning with Scikit-Learn & Tensorflow**: O'Reilly Media, Inc. 1005 Gravenstein Highway North, Sebastopol, CA 95472, 564.

Guignard, M., Schild, M., Bederián, C. S., Wolovick, N., & Vega, A. J. (2018). Performance Characterization of State-of-the-Art Deep Learning Workloads on an IBM" Minsky" Platform. **In Proceedings of the 51st Hawaii International Conference on System Sciences**.

Guo, Q., Chen, S., Xie, X., Ma, L., Hu, Q., Liu, H., & Li, X. (2019). An Empirical Study towards Characterizing Deep Learning Development and Deployment across Different Frameworks and Platforms. In 2019 34th **IEEE/ACM International Conference on Automated Software Engineering (ASE),** 810-822.

Hanhirova, J., Kämäräinen, T., Seppälä, S., Siekkinen, M., Hirvisalo, V., & Ylä-Jääski, A. (2018). Latency and Throughput Characterization of Convolutional Neural Networks

for Mobile Computer Vision. **In Proceedings of the 9th ACM Multimedia Systems Conference**, 204-215.

Huang, L., Xu, J., Sun, J., & Yang, Y. (2017). An Improved Residual LSTM Architecture for Acoustic Modeling. **2nd International Conference on Computer and Communication Systems (ICCCS),** 101-105. IEEE.

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. **arXiv preprint arXiv**:1607.01759.

Kumar, R. (2020, July 28). How to Install and Configure Sysstat on Ubuntu 20.04 – **TecAdmin**. https://tecadmin.net/how-to-install-sysstat-on-ubuntu-20-04/.

Labach, A., Salehinejad, H., & Valaee, S. (2019). Survey of Dropout Methods for Deep Neural Networks. **arXiv preprint arXiv**:1904.13310.

Liu, J., Liu, J., Du, W., & Li, D. (2019). Performance Analysis and Characterization of Training Deep Learning Models on Mobile Device. **In 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS).** 506-515. IEEE.

Liu, L., Wu, Y., Wei, W., Cao, W., Sahin, S., & Zhang, Q. (2018). Benchmarking Deep Learning Frameworks: Design Considerations, Metrics and Beyond. **IEEE 38th International Conference on Distributed Computing Systems (ICDCS),** 1258-1269.

Miao, Y., Li, J., Wang, Y., Zhang, S. X., & Gong, Y. (2016). Simplifying Long Short-Term Memory Acoustic Models for Fast Training and Decoding. **IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).** 2284-2288.

Microsoft. (2020, May 19). Convert Word to Vector: Module reference - Azure Machine Learning. **Microsoft Docs**. https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-module-reference/convert-word-to-vector

Minaee, S. (2019, October 28). 20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics. **Medium**. https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce.

NVIDIA cuDNN. (2021, June 14). **NVIDIA Developer**. https://developer.nvidia.com/cudnn.

SARchart. 2021. **SARchart**. Retrieved August 3, 2021, from https://sarchart.dotsuresh.com/.

Shewalkar, A., Nyavanandi, D., & Ludwig, S. A. (2019). Performance Evaluation of Deep Neural Networks Applied to Speech Recognition: RNN, LSTM and GRU. **Journal of Artificial Intelligence and Soft Computing Research,** 9(4), 235-245.

Smagulova, K., & James, A. P. (2019). A Survey on LSTM Memristive Neural Network Architectures and Applications. **The European Physical Journal Special Topics**, 228(10), 2313-2324.

Tay, Y., Luu, A. T., & Hui, S. C. (2018). Recurrently Controlled Recurrent Networks. **In Advances in Neural Information Processing Systems.** pp. 4731-4743.

TensorFlow. (2021a). **tf.compat.v1.keras.layers.CuDNNLSTM | TensorFlow Core v2.5.0**. Retrieved August 3, 2021, from https://www.tensorflow.org/api_docs/python/tf/compat/v1/keras/layers/CuDNNLSTM.

Tensorflow. (2021b). **tf.keras.layers.Bidirectional | TensorFlow Core v2.5.0.** Retrieved August 3, 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional.

Trianto, R., Tai, T. C., & Wang, J. C. (2018). Fast-LSTM Acoustic Model for Distant Speech Recognition. **IEEE International Conference on Consumer Electronics (ICCE),** 1-4.

Wikipedia contributors. (2021, August 3). Deep Learning. Wikipedia. https://en.wikipedia.org/wiki/Deep_learning.

Yu, W., Li, X., & Gonzalez, J. (2019). Fast Training of Deep LSTM Networks. **In International Symposium on Neural Networks.** 3-10. Springer, Cham.

Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A Review of Recurrent Neural Networks: LSTM cells and network architectures. **Neural computation**, 31(7), 1235-1270.

تقييم و تحسين الأداء للشبكات العصبية من نوع ذاكرة طويلة المدى ثنائية الاتجاه لتطبيقات اللغة العربية

إعداد
أبرار خالد مصطفى سمارة

المشرف
الأستاذ الدكتور غيث عبنده

ملخـــــــص

تُستخدم الشبكات العصبية المتكررة في معالجة اللغة العربية الطبيعية، لكنها بطيئة وقد تستغرق وقتاً طويلاً في عملية التدريب يتراوح بين ساعات إلى عدة أيام. وهذه الرسالة تبحث في حلول لتسريع عملية التدريب.

تحتوي هذه الرسالة على طرق لتحسين وتقليل وقت التدريب، وذلك باستخدام نماذج التعلم العميق لتطبيقات اللغة العربية، وقد تم تطبيق النماذج السريعة على دراسات حالة لتطبيق تصنيف بيت الشعر العربي وتطبيق تشكيل النصوص العربية. وتتضمن عملية التحسين استخدام وحدة معالجة الرسومات بدلاً من وحدة المعالجة المركزية واستخدام طبقة BiCuDNNLSTM بدلا من طبقة BiLSTM. حيث أصبح وقت التدريب لنموذج BiCuDNNLSTM أسرع بمقدار 65 مرة من نموذج BiLSTM وذلك في تجربة تصنيف بيت الشعر. وقد أصبح وقت التدريب في تجربة تشكيل النصوص العربية أسرع بمقدار 21 مرة من نموذج BiLSTM. وقد كانت دقة فحص النموذج BiCuDNNLSTM تساوي 97.17% بينما في نموذج BiLSTM تساوي 97.29% في تجربة تصنيف بيت الشعر. وقد حقق نموذج BiCuDNNLSTM دقة اختبار بنسبة 99.16% بينما حقق نموذج BiLSTM دقة اختبار بنسبة 98.41% في تجربة التشكيل.

وقد تم استخدام الأدوات Sysstat وNvidia-smi لقياس استعمال وحدة معالجة الرسومات، حيث يستخدم نموذج BiCuDNNLSTM وحدة معالجة الرسومات بنسبة 66.56% في تجربة تصنيف بيت الشعر و67.31% في تجربة التشكيل. بينما يستخدم نموذج BiLSTM وحدة معالجة الرسومات بنسبة 35%.