University of
KwaZulu-Natal

June 2022

COMP703

Assignment 3

Report: Comparison of Sequence-to-Sequence Architectures

Prepared by | Sashen Moodley (219006946)

TABLE OF CONTENTS

1 INTRODUCTION 3

2 DATA PRE-PROCESSING 3

3 EXPERIMENTAL SETUP AND RESULTS 4

3.1 MODEL ARCHITECTURE 4

3.2 TRAINING AND VALIDATION 4

3.3 PREDICTIONS 6

4 OBSERVATIONS AND DEDUCTIONS 7

5 CONCLUSION 8

REFERENCES 9

1 INTRODUCTION

The aim of this report is to compare and contrast 3 different sequence-to-sequence (seq2seq) architectures:

1. Bidirectional LSTM encoder and an LSTM decoder
2. Bidirectional GRU encoder and an GRU decoder
3. 2-Layered LSTM encoder and an LSTM decoder

Seq2seq models have seen fruitful use across various tasks such as text summarization, speech recognition, question answering, and much more [1]. They are preferred over other sequence modelling architectures due to their nature of being able to handle variable length input and/or output sequences (i.e., when the input or output is not of a fixed size). In this particular report, we will be looking at applying these 3 architectures in a machine translation task. More specifically, we shall be performing a character-level machine translation task of converting short English expressions to French.

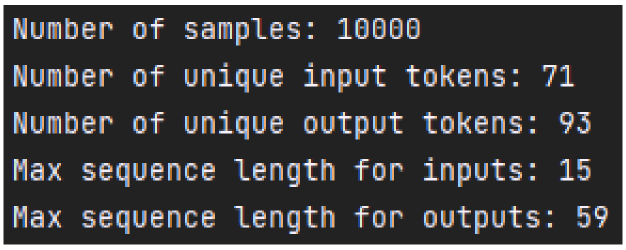
The rest of this report is structured as follows: Section 2 provides a description of the data pre-processing activities. Section 3 presents the experimental setup and results which will be used for a basis of comparison. Finally, Section 4 presents the various observations and deductions surrounding these findings.

2 DATA PRE-PROCESSING

Data cleaning in machine learning refers to the technique of preparing (cleaning and organizing), the raw data to make it more suitable for building and training machine learning models [2]. Thankfully, for this particular task, Keras provides a helpful tutorial [3] to pre-process the dataset. These pre-processing activities can be roughly described as follows:

1. Acquire the dataset – The dataset was acquired from the Keras tutorial [3]. It contains various short English sentences to be translate into their French equivalent (at character level) in a .txt file.
2. Import the crucial libraries – for the task of pre-processing in this research, no specific libraries are required, however more generally, Keras and NLTK are employed.
3. Identify start and end sequence characters for the dataset.
4. Gather samples, and unique input and output tokens.
5. Setup encoder and decoder input data and target decoder data.

From these various pre-processing activities, the following dataset characteristics can be noted in Figure 1.



```
Number of samples: 10000
Number of unique input tokens: 71
Number of unique output tokens: 93
Max sequence length for inputs: 15
Max sequence length for outputs: 59
```

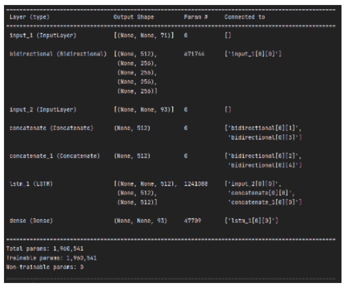
Figure 1 - Data characteristics

3 EXPERIMENTAL SETUP AND RESULTS

Model training in machine learning (ML) is the process of feeding an ML algorithm with data to help identify and learn appropriate values for all attributes involved [4]. The data prepared from the data pre-processing stage has been fed into the model architectures described below.

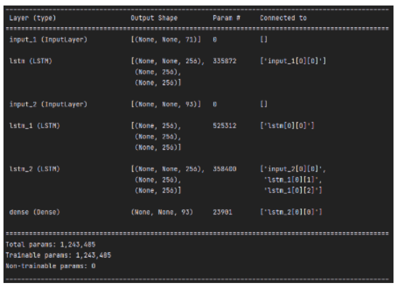
3.1 MODEL ARCHITECTURE

Model training for all architectures of interest is facilitated through TensorFlow Keras architectures. The architectures are setup in a manner that allows for a consistent comparison of results. The specific details of these architectures are presented in the model summaries in Figure 2.



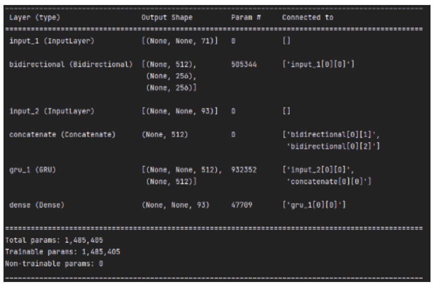
```
--------------------------------------------------------------------------------------------------
 Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)            [(None, None, 71)]   0           []

 bidirectional (Bidirectional)   [(None, 512),        671744      ['input_1[0][0]']
                                  (None, 256),
                                  (None, 256),
                                  (None, 256),
                                  (None, 256)]

 input_2 (InputLayer)            [(None, None, 93)]   0           []

 concatenate (Concatenate)       (None, 512)          0           ['bidirectional[0][1]',
                                                                   'bidirectional[0][3]']

 concatenate_1 (Concatenate)     (None, 512)          0           ['bidirectional[0][2]',
                                                                   'bidirectional[0][4]']

 lstm_1 (LSTM)                   [(None, None, 512),  1241088     ['input_2[0][0]',
                                  (None, 512),                     'concatenate[0][0]',
                                  (None, 512)]                     'concatenate_1[0][0]']

 dense (Dense)                   (None, None, 93)     47709       ['lstm_1[0][0]']

==================================================================================================
Total params: 1,960,541
Trainable params: 1,960,541
Non-trainable params: 0
--------------------------------------------------------------------------------------------------
```

a) BILSTM encoder, LSTM Decoder



```
--------------------------------------------------------------------------------------------------
 Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)            [(None, None, 71)]   0           []

 lstm (LSTM)                     [(None, None, 256),  335872      ['input_1[0][0]']
                                  (None, 256),
                                  (None, 256)]

 input_2 (InputLayer)            [(None, None, 93)]   0           []

 lstm_1 (LSTM)                   [(None, 256),        525312      ['lstm[0][0]']
                                  (None, 256),
                                  (None, 256)]

 lstm_2 (LSTM)                   [(None, None, 256),  358400      ['input_2[0][0]',
                                  (None, 256),                     'lstm_1[0][1]',
                                  (None, 256)]                     'lstm_1[0][2]']

 dense (Dense)                   (None, None, 93)     23901       ['lstm_2[0][0]']

==================================================================================================
Total params: 1,243,485
Trainable params: 1,243,485
Non-trainable params: 0
--------------------------------------------------------------------------------------------------
```

b) BIGRU encoder, GRU Decoder



```
--------------------------------------------------------------------------------------------------
 Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)            [(None, None, 71)]   0           []

 bidirectional (Bidirectional)   [(None, 512),        505344      ['input_1[0][0]']
                                  (None, 256),
                                  (None, 256)]

 input_2 (InputLayer)            [(None, None, 93)]   0           []

 concatenate (Concatenate)       (None, 512)          0           ['bidirectional[0][1]',
                                                                   'bidirectional[0][2]']

 gru_1 (GRU)                     [(None, None, 512),  932352      ['input_2[0][0]',
                                  (None, 512)]                     'concatenate[0][0]']

 dense (Dense)                   (None, None, 93)     47709       ['gru_1[0][0]']

==================================================================================================
Total params: 1,485,405
Trainable params: 1,485,405
Non-trainable params: 0
--------------------------------------------------------------------------------------------------
```

c) 2-Layered LSTM encoder, LSTM Decoder

Figure 2 - Model architectural summaries

3.2 TRAINING AND VALIDATION

The metrics: accuracy, loss, mean squared error, F1-score, precision, and recall were noted across '100' epochs, with a batch size of '64', during model fitting for both the training and validation sets. 20% of the training set is allocated towards model evaluation. The details of these metrics are presented in Table 1.

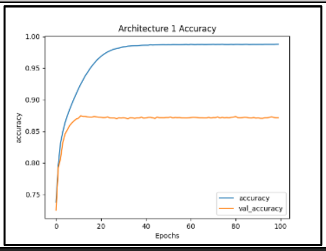
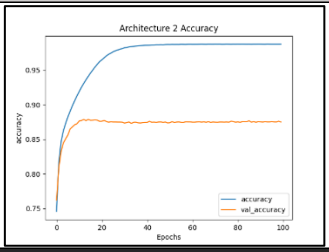
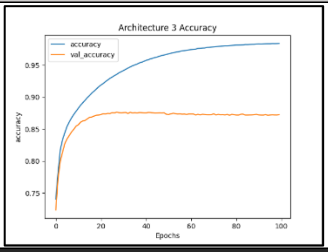
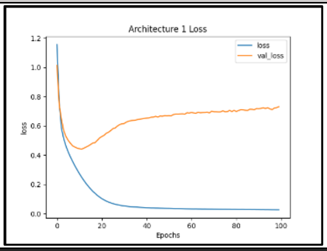
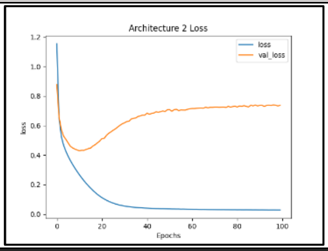
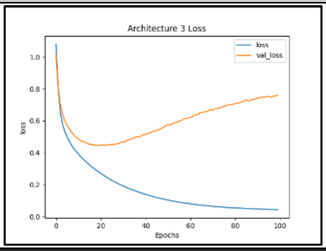
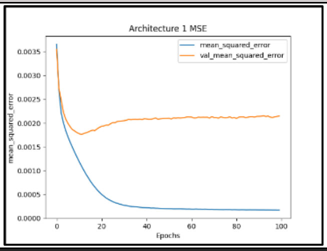
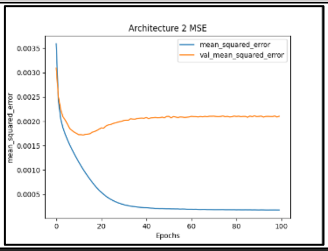
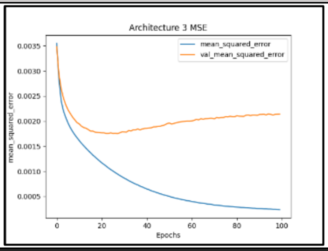
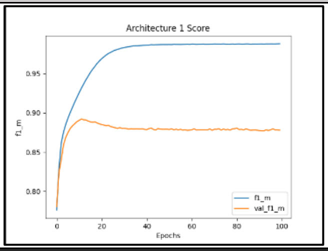
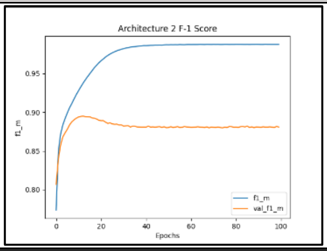
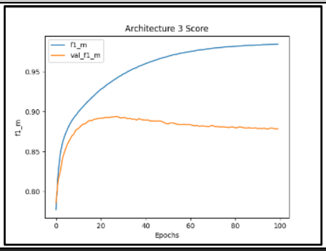
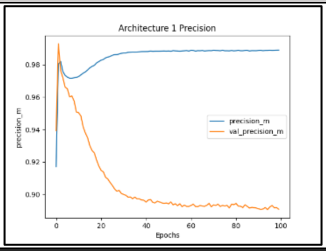
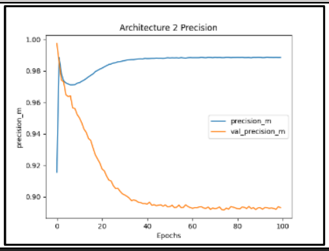
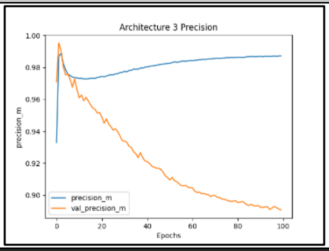
Table 1 - Metric details

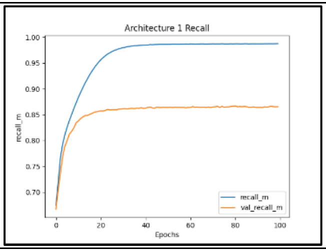
Metric	Description	Formula
Accuracy	Indication of subset accuracy: the set of predicted labels must exactly match the corresponding set of true y labels.	$A = \frac{TP + TN}{TP + TN + FP + FN}$
Precision	Ratio of correctly predicted positive observations to the total predicted positive observations	$P = \frac{TP}{TP + FP}$
Recall	Ratio of correctly predicted positive observations to all observations in actual class	$R = \frac{TP}{TP + FN}$
F1-Score	Weighted average of Precision and Recall	$F1 = 2 \times \frac{P \times R}{P + R}$
Loss (CCE)	A measure on how bad the model's prediction was on a single sample.	$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} 1_{y_i \in C_c} \, log(p_{model}[y_i \in C_c])$
MSE	Measures the amount of error in statistical models (distance between observed and predicted)	$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$

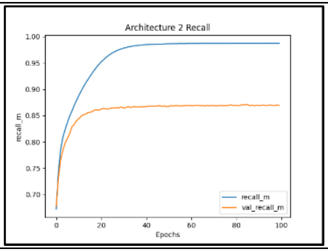
The plots corresponding to these evaluation metrics for training and validation can be noted in Table 2. Take note of the following notations used to represent the different architectures:

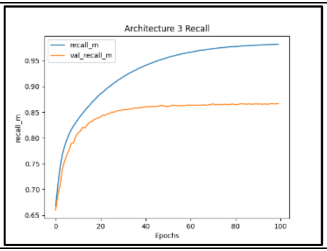
- Architecture 1 - Bidirectional LSTM encoder and a LSTM decoder
- Architecture 2 - Bidirectional GRU encoder and a GRU decoder
- Architecture 3 - 2-Layered LSTM encoder and a LSTM decoder

Table 2 - Metric plots for architectures during training and validation

Accuracy		
Architecture 1	Architecture 2	Architecture 3
		
Loss		
Architecture 1	Architecture 2	Architecture 3
		
MSE		
Architecture 1	Architecture 2	Architecture 3
		
F1-Score		
Architecture 1	Architecture 2	Architecture 3
		
Precision		
Architecture 1	Architecture 2	Architecture 3
		
Recall		
Architecture 1	Architecture 2	Architecture 3







Furthermore, Table 3 presents the observed metrics for each architecture on the last epoch (100):

Table 3 - Training metrics for each architecture

Architecture	Accuracy	Loss	MSE	F1-Score	Precision	Recall
1	0.9879	0.0263	1.7012e-04	0.9882	0.9890	0.9874
2	0.9876	0.0274	1.7743e-04	0.9879	0.9885	0.9872
3	0.9842	0.0424	2.3883e-04	0.9846	0.9872	0.9820

Table 4 presents the appropriate metrics for each architecture during validation:

Table 4 - Validation metrics for each architecture

Architecture	Accuracy	Loss	MSE	F1-Score	Precision	Recall
1	0.8716	0.7310	0.0021	0.8779	0.8907	0.8654
2	0.8751	0.7364	0.0021	0.8810	0.8930	0.8694
3	0.8728	0.7629	0.0021	0.8785	0.8907	0.8667

3.3 PREDICTIONS

A selection of '20' randomized input sequences was chosen for the various architectures to decode. Since this is a character-level machine translation, rather than placing an emphasis on the number of words, we consider the number of characters in the sequence. To evaluate the decoded output, the BLEU-score is noted. Additionally, a smoothing function was used with the NLTK BLEU function. Seeing that the randomized selection was made from a slice of size '50' from the dataset, their respective references for the BLEU evaluation were added appropriately. The average BLEU-scores for each architecture across the '20' input sequences are provided in Table 5.

Table 5 - BLEU scores for architectures

Architecture	Average BLEU-Score
1	0.946
2	0.939
3	0.918

Finally, Table 6 presents the elapsed time across training and prediction stages for each architecture

Table 6 - Training and prediction times for each architecture

Architecture	Training Time (s)	Prediction Time (s)
1	802.13	27.85
2	730.46	25.48
3	785.47	26.20

4 OBSERVATIONS AND DEDUCTIONS

From the metrics presented in Section 3, along with the training and validation data plots, we can note the following main observations from this particular character-level machine translation task:

1. The Bidirectional LSTM provides the most accurate predictions
2. The GRU-based architecture is the fastest to train and to make predictions with
3. Both bidirectional architectures provide more accurate predictions than the stacked architecture

Diving into this deeper, we can try and gain an understanding of these observations. The underpinning factor differentiating the speed and accuracy performance of the Gated-RNNs are their different approach to gates. These gates are introduced to help mitigate the issue of vanishing gradient during backwards propagation for longer sequence tasks present in the 'vanilla' RNNs. The gated units can control the flow of information and choose what information is important. The LSTM has 3 gates: an input, output, and forget gate, whilst the GRU has 2 gates: an update, and reset gate. Whereas the LSTM uses the cell state to capture and transfer information from earlier steps the GRU uses the hidden state itself, i.e., the GRU simply exposes the complete hidden content layers (memory) without the level of control like the LSTM as it does not have a dedicated memory cell.

The bidirectional architecture aims to retain information from both the past and future of the inputs in the sequence, as apposed to just one. This allows the architecture to 'see' information down the road that would otherwise be out of its scope and therefore have no conditional influence on the current input. However, the bidirectional architecture examined in this research only considers the time dimension. Stacked architectures allow the model to be extended in its time as well as additional depth. That is, it allows for the growth of multidimensional dependencies from the sequence input.

Applying these main concepts to observations, we can try to gain an understanding of each. Firstly, elaborating on *point 1*, the LSTM slightly outperforms the GRU in accuracy. Due to the GRU having less gates, and no dedicated memory cell state akin to the cell state in the LSTM, the GRU works with less training parameters thus giving the advantage to the LSTM which has a greater capacity to learn. However, this more complex structure employed by the LSTM results in only slight improvements in the respective metrics (as indicated by Tables 3-5). The reason for this only marginal improvement could be due to the short nature of character sequences in this examined task. The relatively short sequence lengths finds the LSTM to be 'overqualified' for this domain, as it is more suitable for capturing longer-term dependencies.

Secondly, in understanding why the GRU trains faster than the LSTM. Unlike the evaluation metrics, the difference in the training and evaluation times are quite significant. Particularly in training, we see the GRU train 9% faster than the LSTM (see Table 6). The GRU is faster to train and evaluate for the same reasons why it is slightly less accurate than the LSTM, that being it only works with 2 gated units and does not need to worry about updating an internal cell state. Therefore, as indicated by Figure 2, the GRU works with less training parameters, in turn requiring fewer gated computations and less memory.

Lastly, in comparing the stacked and bidirectional architectures, it was shown (see Tables 3-5) that the bidirectional architectures (architectures 1 and 2) outperformed the stacked architecture present in architecture 3 with regards to translation prediction accuracy. In this situation of character-level machine translation, the benefits of having a time and additive depth dimension are not fully utilized. The most fruitful results for this type of architecture have been noted in domains such as pixel intensity value predictions where there is a dependency on both x and y values in the plane, introducing dependency growth in two dimensions. However, here we do not have that bidimensional dependency to fully capitalize of the stacked architecture's benefits.

5 CONCLUSION

The main aims of this report were to determine the efficacy of the various seq2seq architectures. Comparisons were noted between the different gated RNN-based architectures (LSTM and GRU) arriving at the conclusion that GRU is faster to train and make predictions, but the LSTM provides slightly more accurate predictions (when using the same bidirectional philosophy). Furthermore, a comparison was made between bidirectional and stacked encoders for seq2seq modelling, which this research has empirically shown that bidirectional architectures provide more accurate translations than a stacked architecture, at least in this particular task.

REFERENCES

[1] R. Tatman, ‘Evaluating Text Output in NLP: BLEU at your own risk’, *Medium*, Jul. 16, 2019. https://towardsdatascience.com/evaluating-text-output-in-nlp-bleu-at-your-own-risk-e8609665a213 (accessed Jun. 15, 2022).

[2] K. Goyal, ‘Data Preprocessing in Machine Learning: 7 Easy Steps To Follow’, *upGrad blog*, Jul. 15, 2021. https://www.upgrad.com/blog/data-preprocessing-in-machine-learning/ (accessed Apr. 01, 2022).

[3] K. Team, ‘Keras documentation: Character-level recurrent sequence-to-sequence model’. https://keras.io/examples/nlp/lstm_seq2seq/ (accessed Jun. 15, 2022).

[4] ‘What is Model Training’, *Oden Technologies*. https://oden.io/glossary/model-training/ (accessed Apr. 01, 2022).

[5] ‘Introduction to Encoder-Decoder Sequence-to-Sequence Models (Seq2Seq)’, Paperspace Blog, Mar. 09, 2021. https://blog.paperspace.com/introduction-to-seq2seq-models/ (accessed Jun. 16, 2022).

[6] ‘Implementing Seq2Seq Models for Text Summarization With Keras’, Paperspace Blog, Mar. 12, 2021. https://blog.paperspace.com/implement-seq2seq-for-text-summarization-keras/ (accessed Jun. 16, 2022).