NLP

May 2024

TP4 Report:

NLPwithSequence Models

- ROUBACHE Islam

1.Load the **conll003-englishversion** dataset and perform pre-processing: data normalization.

Conll003-englishversion dataset:

CoNLL-2003 is a dataset commonly used for named entity recognition (NER) and part-of-speech (POS) tagging tasks. It is derived from the Reuters corpus and consists of news articles annotated with named entities and part-of-speech tags. The dataset is divided into training, development, and test sets.
Each token in the dataset is tagged with its part-of-speech and labeled with its named entity type, such as PERSON, LOCATION, ORGANIZATION, etc. Researchers often use this dataset to train and evaluate NER and POS tagging models, as it provides a standardized benchmark for evaluating the performance of such systems.

Data Preprocessing:

Data Preparation & Case Normalization:

Wewill processes lines of text data, extracts wordsand corresponding tags (likely part-of-speech or named entity tags), and organizes them into sentences with their respective labels. then handles sentence boundaries and normalizes the case of words.

```
for line in lines:
    if line == "\n" or line.startswith("-DOCSTART-"):
        if sentence and label:
            sentences.append(sentence)
            labels.append(label)
            sentence, label = [], []
    else:
        word, _, _, tag = line.strip().split()
        sentence.append(word.lower())  # Normalize the case
        label.append(tag)

return sentences, labels
```

Tokenization & Vectorization:

We’llbreak text into smaller units, Fit the Tokenizer from keras on the train Data then Converting sentences (and labels) from their original text format into sequences of integers, where each integer represents a token in the vocabulary learned by the tokenizers.

```
# Tokenization
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['train_sentences'])
sequences = tokenizer.texts_to_sequences(df['train_sentences'])
test_sequences = tokenizer.texts_to_sequences(test_sentences)
valid_sequences = tokenizer.texts_to_sequences(valid_sentences)
label_tokenizer = Tokenizer()
label_tokenizer.fit_on_texts(df['train_labels'])
label_sequences = label_tokenizer.texts_to_sequences(df['train_labels'])
test_label_sequences = label_tokenizer.texts_to_sequences(test_labels)
valid_label_sequences = label_tokenizer.texts_to_sequences(valid_labels)
```

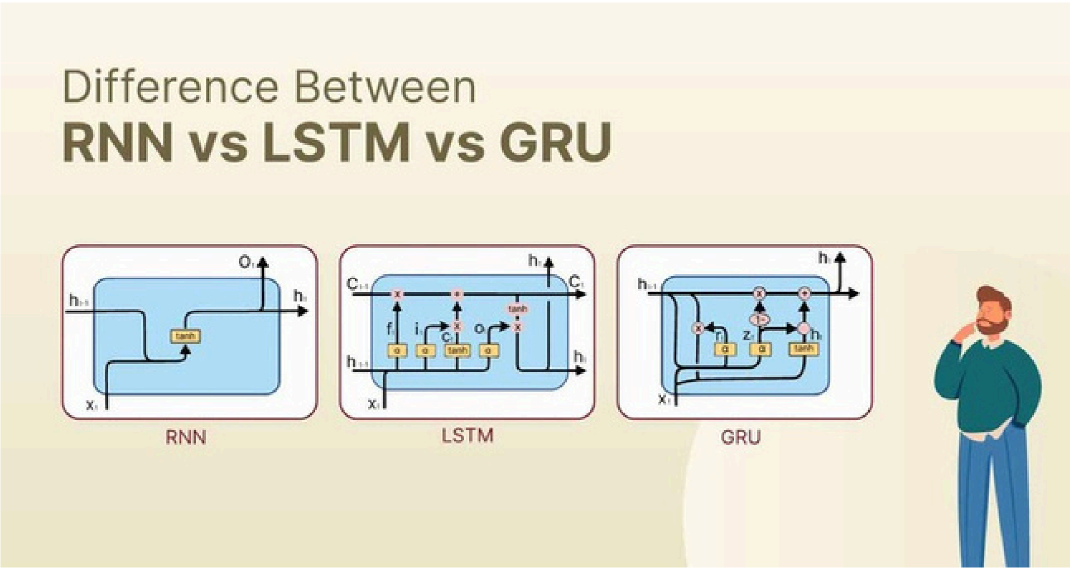
Padding & Prepare data for Models:

Wewill prepare thesequences of tokenizedsentences and labels for model training and evaluation. and nsures that all sequences have the same length, which is necessary for feeding them into neural networks using **pad_sequences** function from Keras

```
# Padding
maxlen = max(len(s) for s in sequences)
X_train = pad_sequences(sequences, maxlen=maxlen, padding='post')
X_test = pad_sequences(test_sequences, maxlen=maxlen, padding='post')
X_valid = pad_sequences(valid_sequences, maxlen=maxlen, padding='post')
y_train = pad_sequences(label_sequences, maxlen=maxlen, padding='post')
y_test = pad_sequences(test_label_sequences, maxlen=maxlen, padding='post')
y_valid = pad_sequences(valid_label_sequences, maxlen=maxlen, padding='post')
```

Models Training:

Wewill train3modelsof recurrent neural network (RNN) : **LSTM (Long Short-Term Memory), RNN (Vanilla Recurrent Neural Network), and GRU (Gated Recurrent Unit)**

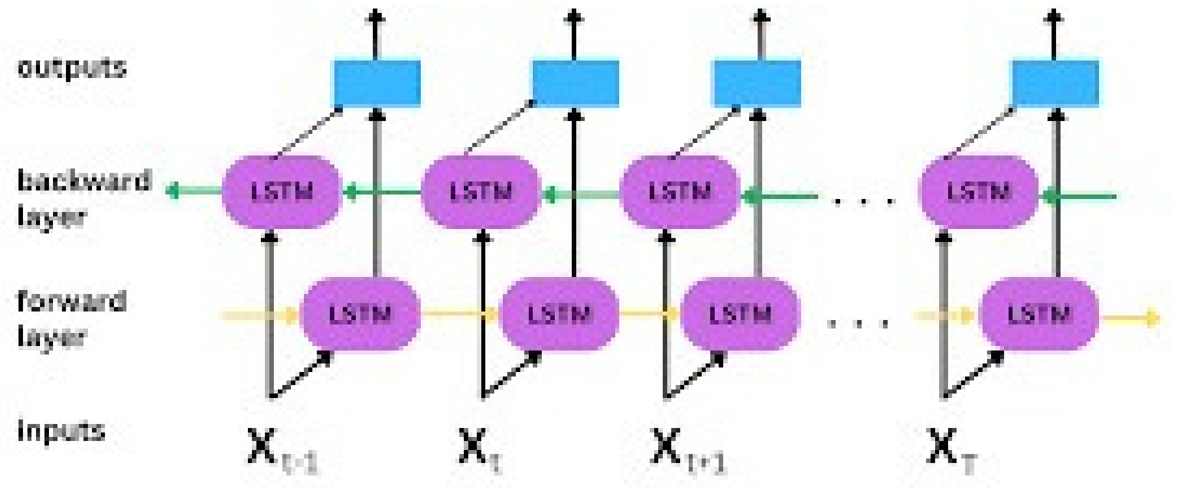


For the Three models we will use same architecute where Our Input will pass through several stages:

Embedding → Model → Dense layer → Output

1.BiLSTM (Bidirectional Long Short-Term Memory):

ABidirectional LSTM,orBiLSTM,isa type of LSTM (LongShort-TermMemory) that learnsfrom inputdatafrom twodirectionsinstead of one, thus capturingpast**(backward)** and future**(forward)** information simultaneously.



Build & Compile

```
vocab_size = len(tokenizer.word_index) + 1
max_len = 50
embedding_dim = 100
hidden_units = 64
num_classes = len(label_tokenizer.word_index) + 1
batch_size = 32
num_epochs = 10

# Création du modèle
model1 = Sequential()
model1.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_len))
model1.add(Bidirectional(LSTM(units=hidden_units, return_sequences=True)))
model1.add(TimeDistributed(Dense(num_classes, activation='softmax')))

# Compilation du modèle
model1.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
```

Architecture

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 50, 100)           2101000
_________________________________________________________________
bidirectional (Bidirectional (None, 50, 128)           84480
_________________________________________________________________
time_distributed (TimeDistri (None, 50, 10)            1290
=================================================================
Total params: 2,186,770
Trainable params: 2,186,770
Non-trainable params: 0
_________________________________________________________________
```

When you are dealing with sequence data, and you want to apply a Dense layer to each time step, you wrap the Dense layer in a **`TimeDistributed`** layer to apply the same Dense layer to each time step.

Training and Results

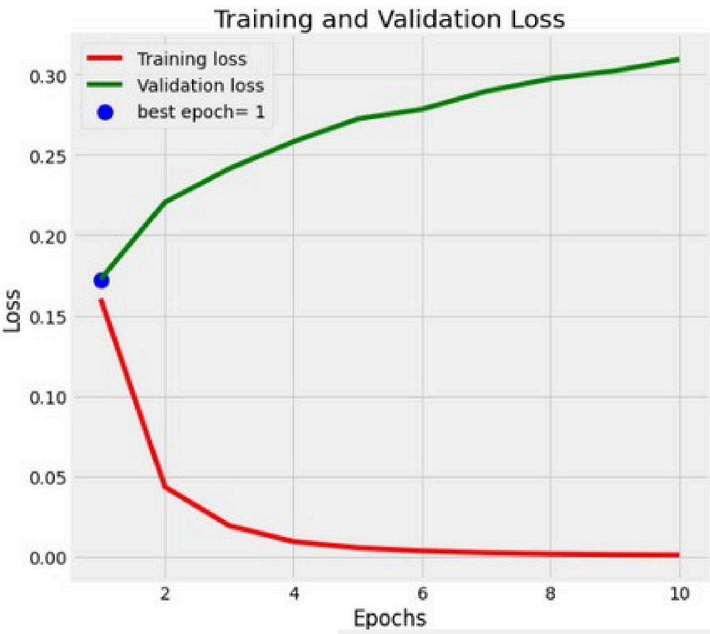
Ourmodel got2,186,770 trainableparameters. we trained our model for 10 epochs with a batch size of 32 , as an optimizer we used The ADAM optimizer

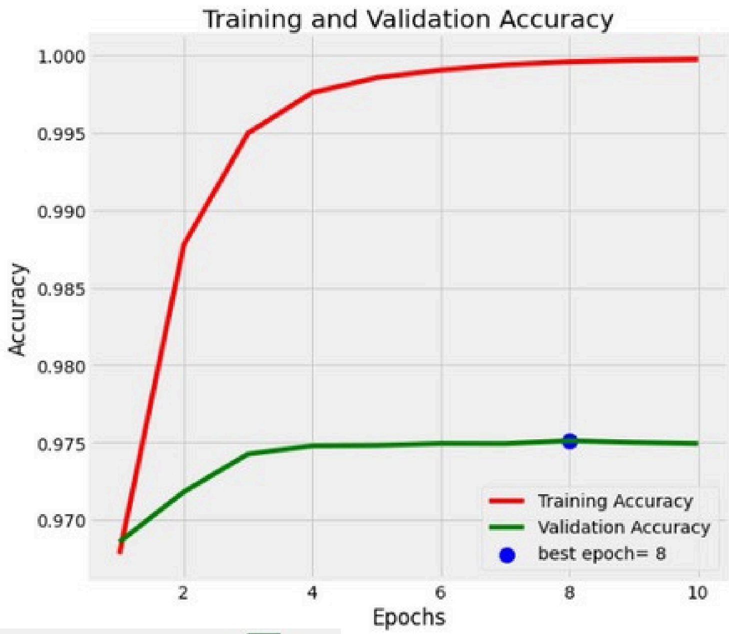
```
==========Model: LSTM=============
Train Loss:  0.0007434531580656767
Train Accuracy:  0.9998493790626526
--------------------
Validation Loss:  0.30941540002822876
Validation Accuracy:  0.9749271869659424
--------------------
Test Loss:  0.3505815267562866
Test Accuracy:  0.9717572927474976
=========================
```

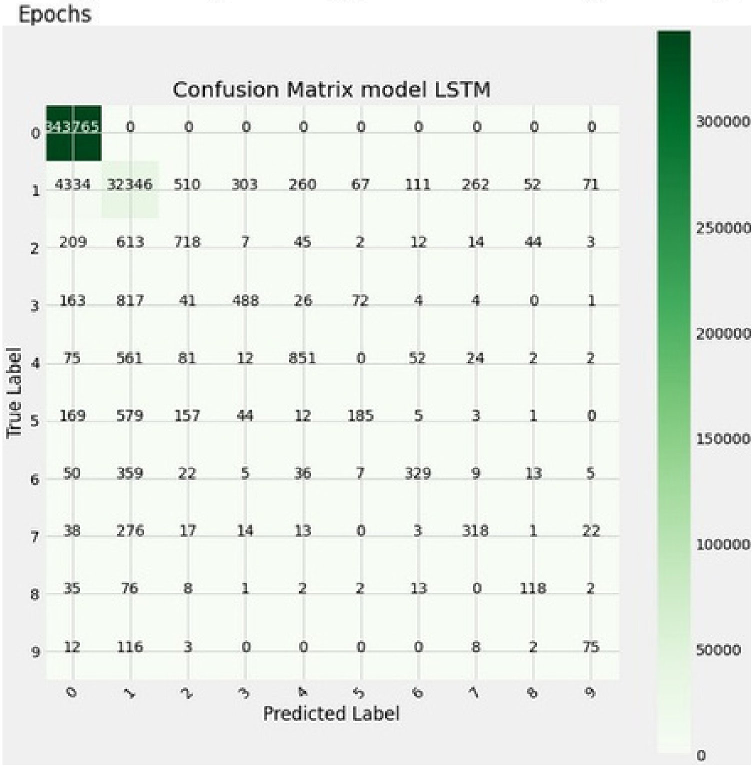
```
LSTM:
Accuracy: 0.9718187852553506
Precision: 0.9686556953318787
Recall: 0.9718187852553506
F1-score: 0.9694959280927841
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    343765
           1       0.90      0.84      0.87     38316
           2       0.46      0.43      0.45      1667
           3       0.56      0.30      0.39      1616
           4       0.68      0.51      0.59      1660
           5       0.55      0.16      0.25      1155
           6       0.62      0.39      0.48       835
           7       0.50      0.45      0.47       702
           8       0.51      0.46      0.48       257
           9       0.41      0.35      0.38       216

    accuracy                           0.97    390189
   macro avg       0.62      0.49      0.54    390189
weighted avg       0.97      0.97      0.97    390189
```

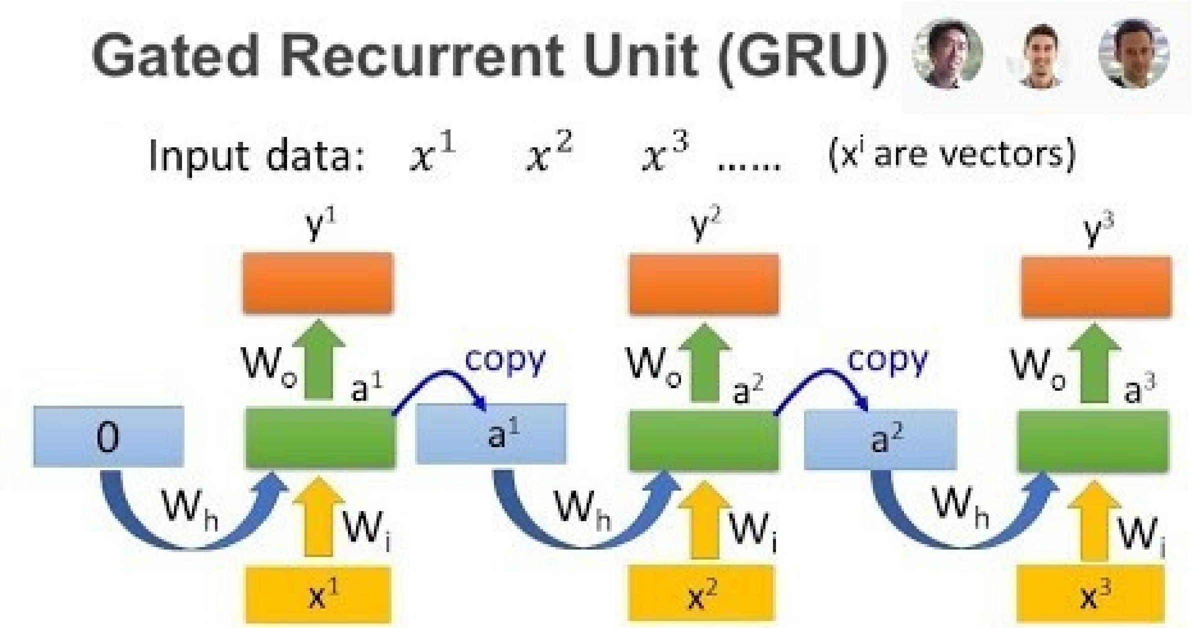






2.GRU (Gated Recurrent Unit):

GRU isa simplifiedversionof theLSTM architecture, designed to be computationally efficient while still capturinglong-range dependencies. It combines theforget and input gates into a single update gate, and merges the cell state and hidden state, resulting in fewer parameters compared to LSTM.



Build & Compile

```
# Define the model
model2 = Sequential()
model2.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=64, input_length=maxlen))
model2.add(GRU(units=64, return_sequences=True))
model2.add(TimeDistributed(Dense(len(label_tokenizer.word_index)+1, activation='softmax')))

# Compile the model
model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Architecture

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
embedding_1 (Embedding)     (None, 113, 64)           1344640
_________________________________________________________________
gru (GRU)                   (None, 113, 64)           24960
_________________________________________________________________
time_distributed_1 (TimeDist (None, 113, 10)          650
=================================================================
Total params: 1,370,250
Trainable params: 1,370,250
Non-trainable params: 0
_________________________________________________________________
```

Training and Results

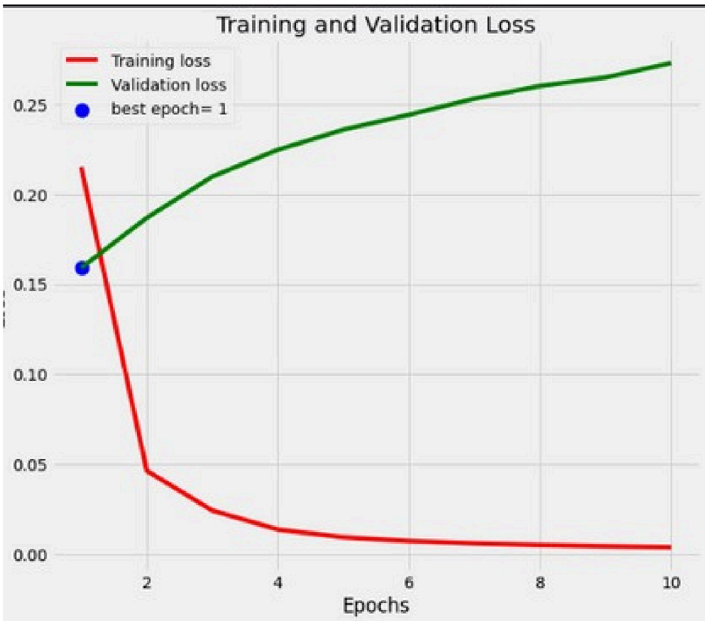
Ourmodel got1,370,250 trainableparameters. we trained our model for 10 epochs with a batch size of 32 , as an optimizer we used The ADAM optimizer

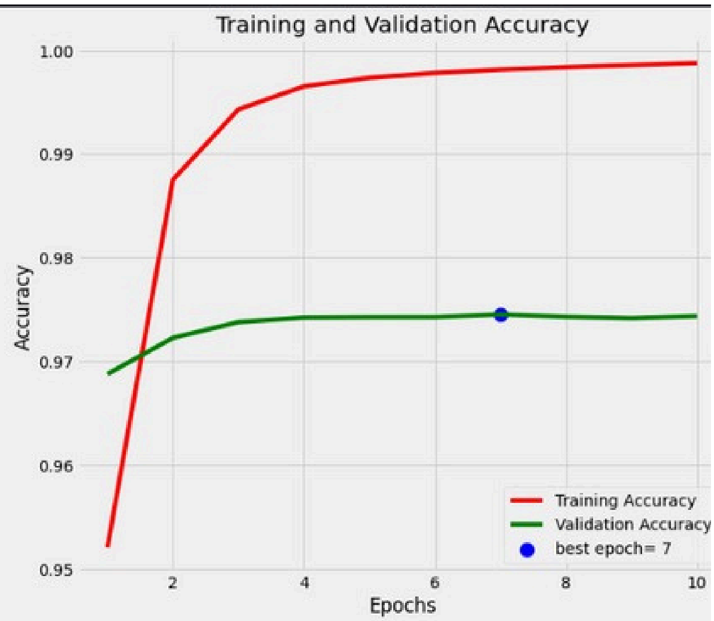
```
==========Model: GRU=============
Train Loss:  0.0030459405388683O8
Train Accuracy:  0.9989985227584839
--------------------
Validation Loss:  0.2727097272872925
Validation Accuracy:  0.9743471741676331
--------------------
Test Loss:  0.30881381034851074
Test Accuracy:  0.9712011218070984
=========================
```

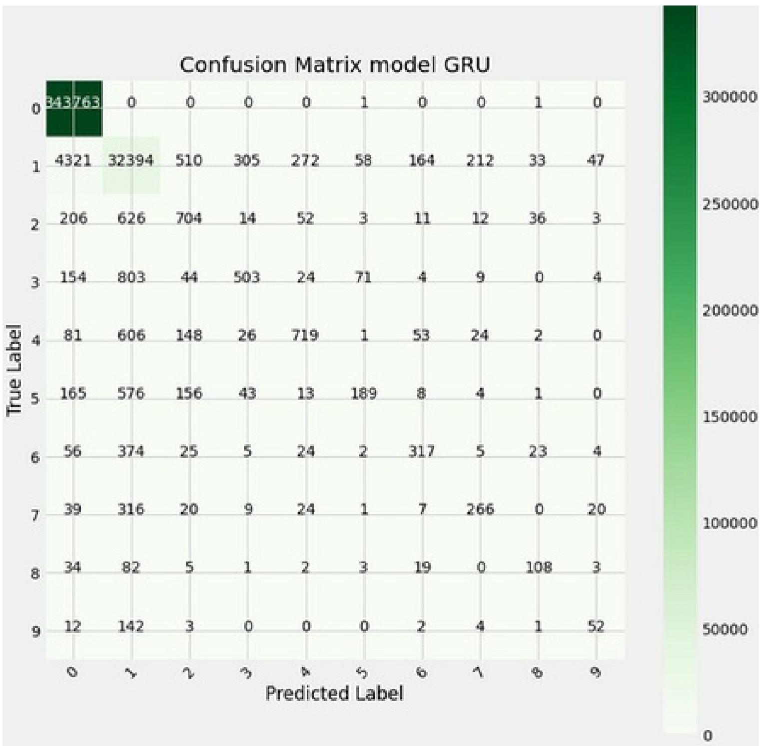
```
GRU:
Accuracy: 0.9713625960752353
Precision: 0.9679684828464288
Recall: 0.9713625960752353
F1-score: 0.9689075437445418
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    343765
           1       0.90      0.85      0.87     38316
           2       0.44      0.42      0.43      1667
           3       0.56      0.31      0.40      1616
           4       0.64      0.43      0.52      1660
           5       0.57      0.16      0.25      1155
           6       0.54      0.38      0.45       835
           7       0.50      0.38      0.43       702
           8       0.53      0.42      0.47       257
           9       0.39      0.24      0.30       216

    accuracy                           0.97    390189
   macro avg       0.60      0.46      0.51    390189
weighted avg       0.97      0.97      0.97    390189
```

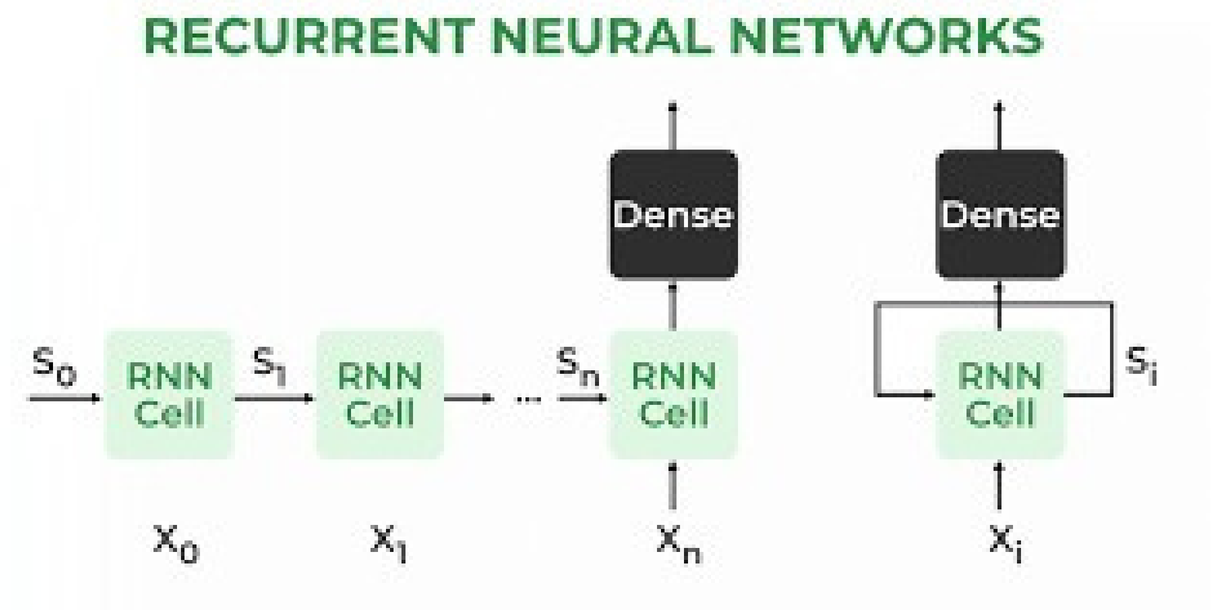






3. RNN (Vanilla Recurrent Neural Network):

RNNis thesimplestform ofrecurrentneural network, whereeachneuron has aself-connected recurrentconnection.Itprocesses sequential databy iterativelyupdating its hiddenstate using the current input and previous hidden state.



Build & Compile

```
# Define the model
model3 = Sequential()
model3.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=64, input_length=maxlen))
model3.add(SimpleRNN(units=64, return_sequences=True))
model3.add(TimeDistributed(Dense(len(label_tokenizer.word_index)+1, activation='softmax')))

# Compile the model
model3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Architecture

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
embedding_2 (Embedding)     (None, 113, 64)           1344640

simple_rnn (SimpleRNN)      (None, 113, 64)           8256

time_distributed_2 (TimeDist (None, 113, 10)          650
=================================================================
Total params: 1,353,546
Trainable params: 1,353,546
Non-trainable params: 0
_________________________________________________________________
```

Training and Results

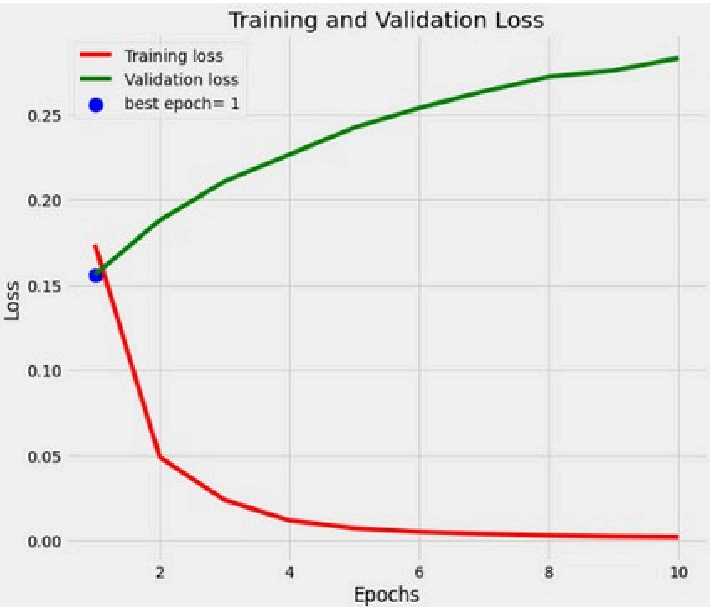
Ourmodel got1,353,546 trainableparameters. we trained our model for 10 epochs with a batch size of 32 , as an optimizer we used The ADAM optimizer

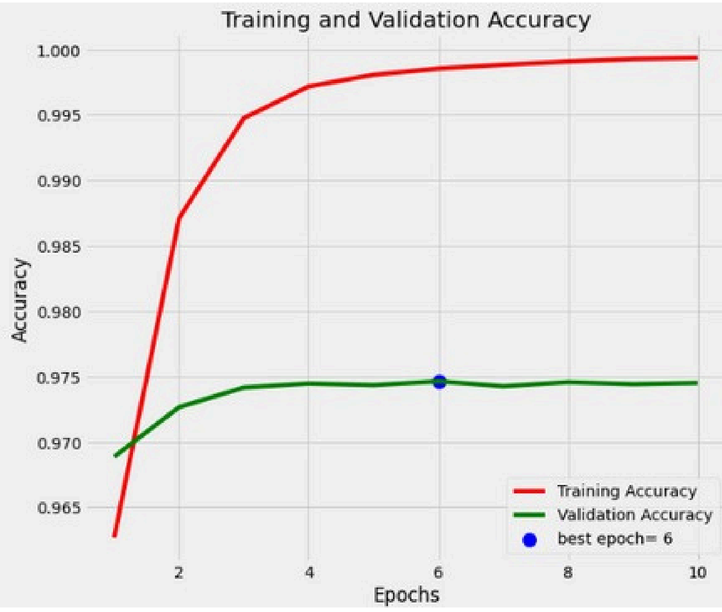
```
==========Model: RNN=============
Train Loss:  0.0017267044167965 65
Train Accuracy:  0.9994781613349915
--------------------
Validation Loss:  0.2831628620624542
Validation Accuracy:  0.9745078086853027
--------------------
Test Loss:  0.3196185827255249
Test Accuracy:  0.9713805317878723
=========================
```

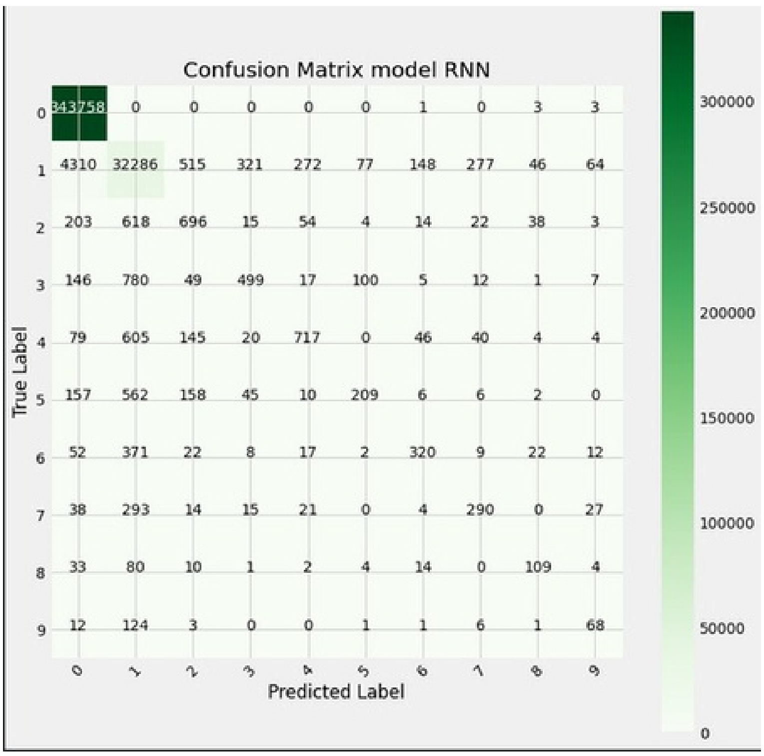
```
RNN:
Accuracy: 0.9712011358598013
Precision: 0.9679913653171917
Recall: 0.9712011358598013
F1-score: 0.9689309931030305
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    343765
           1       0.90      0.84      0.87     38316
           2       0.43      0.42      0.42      1667
           3       0.54      0.31      0.39      1616
           4       0.65      0.43      0.52      1660
           5       0.53      0.18      0.27      1155
           6       0.57      0.38      0.46       835
           7       0.44      0.41      0.43       702
           8       0.48      0.42      0.45       257
           9       0.35      0.31      0.33       216

    accuracy                           0.97    390189
   macro avg       0.59      0.47      0.51    390189
weighted avg       0.97      0.97      0.97    390189
```







4. **Conclusion:**

- Each ofthese recurrent neural network architectures—LSTM, RNN, GRU, and BiLSTM—can achieve the desired results for sequential data tasks, but they may differ in terms of convergence speed and efficiency. as we can see LSTM converge at 8th epoch , GRU converge at 7th epoch and RNN at 6th epoch.

- while LSTM, RNN, GRU, and BiLSTM are all recurrent neural network architectures designed for sequential data tasks, they have distinct differences in terms of architecture, parameterization, and capability to capture long-term dependencies and bidirectional context. The choice of model depends on the specific requirements of the task at hand, computational resources available, and desired trade-offs between model complexity and performance.