School of Information Technology and Engineering

Addis Ababa Institute of Technology

Addis Ababa University

Software Engineering - (AI - Stream)

Cognitive Science - Next word prediction report

Name	**ID.No**
1. Abraham Wendmeneh	UGR/9155/13
2. Biniyam Haile	UGR/2646/13
3. Fraol Mulugeta	UGR/5835/13
4. Kidus Hunegnaw	UGR/6554/13
5. Melkishi Tesfaye	UGR/0078/13
6. Sosina Esayas	UGR/2014/13

Submission Date: May 26, 2025

Submission To: Dr. Adane

Next Word Prediction Using RNN (GRU-based Language Model)

Objective

To build a word-level next word prediction model to impersonate human-like interactions through text communication, which could be useful to improve typing efficiency and facilitate seamless communication. We used a GRU-based recurrent neural network trained on the text corpus 1661-0.txt (A public domain book called The Adventures of Sherlock Holmes, by Arthur Conan Doyle from Project Gutenberg).

Environment Setup

We used Google Colab to train our model

Drive Mounting

```
[1]
    from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

Data Loading

```
[10] # getting our text file
     with open('/content/drive/MyDrive/next-word-prediction/dataset/1661-0.txt') as f:
         data = f.read()
```

Preprocessing Step (Text Cleaning & Tokenization)

1. Tokenization

Machine learning models can’t understand raw text due to this fact, that the raw text must be converted to numbers for the machine to work with our data.

Converting Text into Vectors

```
[5] tokenizer = Tokenizer() # Create a tokenizer
```

```
[6] tokenizer.fit_on_texts([data])  ## Creates a vocabulary list of all unique words in the sentences
```

```
[7] # Save the tokenizer
    with open('tokenizer.pickle', 'wb') as handle:
        pickle.dump(tokenizer, handle)
```

```
keys_list = tokenizer.word_index  ## creates a word to index mapping
print("No. of words = " , len(keys_list))
```

```
No. of words =  8931
```

2. Sequence Preparation

This helps the model learn what words are likely to follow a given sequence.

```
[23] input_sequences = []

     for sentence in data.split('\n'):

         tokenized_sentence = tokenizer.texts_to_sequences([sentence])[0]  ## the [0] index is putting all sequences in one list
         for i in range( 1, len(tokenized_sentence)):
             input_sequences.append(tokenized_sentence[:i+1]) ## apppendind tokenized sentences to input_sequences list
```

```
[11] ## length of the biggest line
     max_len = max(len(x) for x in input_sequences)
```

- Splits text line by line.
- Converts each line to a list of integers.
- Creates all possible n-gram sequences from each line.

4. Padding Sequences

'pre' adds zeros at the beginning of the sequences to ensure all sequences have the same length.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

padded_input_sequences = pad_sequences(input_sequences, ## vector who's vector we need to padd
                                       maxlen=max_len,  ## length of sequence's vectors
                                       padding='pre'    ## padding from the starting
                                      )
```

```
[13] padded_input_sequences
```

```
array([[   0,    0,    0, ...,    0,  145, 4790],
       [   0,    0,    0, ...,  145, 4790,    1],
       [   0,    0,    0, ..., 4790,    1, 1020],
       ...,
       [   0,    0,    0, ...,    3,  360,   83],
       [   0,    0,    0, ...,  360,   83,  358],
       [   0,    0,    0, ...,   83,  358, 1673]], dtype=int32)
```

Training

Model Architecture used

A stacked GRU(Gated Recurrent Unit)-based bidirectional model:

```
[20] model = Sequential()
     model.add(Embedding(INPUT_LENGTH, 100))
     model.add(Bidirectional(GRU(units=80, return_sequences=True)))
     model.add(Dropout(0.2))
     model.add(Bidirectional(GRU(units=80, return_sequences=True)))
     model.add(Dropout(0.2))
     model.add(Bidirectional(GRU(units=80)))
     model.add(Dense(INPUT_LENGTH, activation='softmax'))
```

```
[21] model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
bidirectional (Bidirectional)	?	0 (unbuilt)
dropout (Dropout)	?	0
bidirectional_1 (Bidirectional)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0
bidirectional_2 (Bidirectional)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

- **Epochs**: 100
- **Input Shape**: (101619, 18) (after slicing last token for target)
- **Output Shape**: (101619, 8932)
- **Loss Function**: categorical_crossentropy for multi-class classification.
- **Optimizer**: adam for adaptive learning rate.

Total Parameters: ~5.3 million (based on estimated embedding + GRU + Dense sizes)

Results

```
N_EPOCHS = 86

history = model.fit(X, y, epochs=N_EPOCHS)
```

```
3176/3176 ━━━━━━━━━━━━━━━━━━━━ 53s 17ms/step - accuracy: 0.5624 - loss: 1.7672
Epoch 85/100
3176/3176 ━━━━━━━━━━━━━━━━━━━━ 82s 17ms/step - accuracy: 0.5657 - loss: 1.7535
Epoch 86/100
```

Metric	Value
Final Accuracy	e.g., 56.57%
Final Loss	e.g., 1.75

Saving the Model

To ensure the trained model can be reused for inference.

```
model.save('next_word_prediction.keras')
```

Summary

- Tokenized using Tokenizer from tensorflow.keras.preprocessing.text
- Created sequences of increasing length from each line.
- Model used is a basic GRU pipeline suitable for short- to medium-range sequence prediction.
- Output is a ready-to-use text generator based on learned patterns from a public domain novel.
  - Total words (vocab size): 8931
  - Final number of input sequences: 101,619
  - Input sequence length: 19
  - Target classes (next word): one-hot encoded into 8932 categories.

Conclusion

- The model successfully learns to predict the next word in a sequence based on context.
- The use of bidirectional GRUs enhances context awareness.
- Training over 100 epochs shows stable convergence (TBD on actual loss/accuracy curve).