Aim: Develop a Character-Based or word-based Neural Language Model for text generation.

Theory:
Developing a Character-Based or Word-Based Neural Language Model for text generation is a critical task in natural language processing. Each approach has distinct characteristics, uses, advantages, and disadvantages.

Character-Based Language Model:
Uses:

1. Multilingual Text Generation: Character-based models can generate text in multiple languages using the same model, making them valuable for multilingual applications.
2. Creative Text Generation: They are well-suited for creative writing tasks such as generating poetry, lyrics, or fictional narratives due to their fine-grained control.
3. Code Generation: Character-based models can generate code, making them useful for code auto-completion and suggestion tools.
4. Spelling Correction: They can be employed for spell-checking and auto correct in text editors and chat applications.
5. Text Compression: Character-based models can be used for text compression tasks, encoding text in a more compact form.
6. Language Generation for Speech Synthesis: They play a role in text-to-speech (TTS) systems to generate phonemes for speech synthesis.

Advantages:

1. Flexibility: Character-based models can handle multiple languages, generate special symbols, and handle creative text generation tasks.
2. Fine-Grained Control: These models offer fine-grained control over the generated text, making them suitable for artistic and creative writing.
3. Handling Rare Words: Character-based models can construct rare or out-of-vocabulary words from constituent characters.
4. Spelling Correction: They can be used for spell-checking and spelling correction tasks by predicting and suggesting correct characters.

Disadvantages:

1. High Dimensionality: Character-level modeling results in high-dimensional input and output spaces, which can lead to increased computational complexity during training and inference.
2. Longer Training Time: Training character-based models may require more data and time compared to word-based models.
3. Lack of Semantic Understanding: Character-based models do not inherently capture word-level semantics or relationships between words.
4. Text Coherence: Generating coherent and contextually meaningful text can be challenging for character-based models.

<u>Word-Based Language Model</u>:

Uses:

1. Machine Translation: Word-based models are commonly used in machine translation systems, where they excel in translating text between languages.
2. Document Summarization: They are employed in text summarization tasks to generate concise and coherent document summaries.
3. Sentiment Analysis: Word-based models are used in sentiment analysis to analyze and classify the sentiment expressed in text data.
4. Text-to-Speech (TTS) Systems: They play a role in TTS systems to convert text into spoken language by generating phonemes and prosody.
5. Question Answering: Word-based models are used in question answering systems to extract answers from text passages or knowledge bases.

Advantages:

1. Semantic Understanding: Word-based models have a better understanding of semantics and can capture word-level relationships.
2. Lower Dimensionality: They often have lower input and output dimensionality compared to character-based models, making training and inference faster.
3. Handling Multiple Languages: Word-based models can handle multiple languages, provided they are trained on multilingual data or have access to appropriate word embeddings.

4. Named Entities and Structured Information: These models can capture named entities and structured information, valuable for tasks like named entity recognition and information extraction.

Disadvantages:

1. Limited Vocabulary: Word-based models may struggle with rare or out-of-vocabulary words and complex word formation in certain languages.
2. Lack of Creativity: They may generate text that is more predictable and less creative compared to character-based models, which are more suitable for creative
3. text generation.

Conclusion:

In conclusion, the development of either a character-based or word-based neural language model for text generation is a task that hinges on the specific demands of natural language processing. Character-based models offer flexibility for multilingual text generation, creative writing, code generation, and spelling correction, but they can be computationally intensive and may lack inherent semantic understanding. On the other hand, word-based models excel in understanding semantics, handling machine translation, document summarization, sentiment analysis, and question answering but may face challenges with rare words and creative text generation. The choice between these models should be made based on the nature of the data, the linguistic complexity of the task, and the desired level of control and creativity in the text generation process, recognizing that both character-based and word-based models play essential roles in advancing NLP capabilities across diverse applications.

Notebook

October 15, 2023

[4]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Define a function for character-based language model
def character_based_language_model():
    # Load and preprocess the text data
    text = open('/content/Text.txt', 'r').read()
    text = text.lower()  # Convert text to lowercase

    # Create character-level vocabulary
    chars = sorted(list(set(text)))
    char_indices = dict((c, i) for i, c in enumerate(chars))
    indices_char = dict((i, c) for i, c in enumerate(chars))

    # Generate training data sequences
    maxlen = 40  # Define the length of input sequences
    step = 3  # Define the step size for sliding window
    sentences = []
    next_chars = []

    for i in range(0, len(text) - maxlen, step):
        sentences.append(text[i: i + maxlen])
        next_chars.append(text[i + maxlen])

    # Vectorize the data
    x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
    y = np.zeros((len(sentences), len(chars)), dtype=np.bool)

    for i, sentence in enumerate(sentences):
        for t, char in enumerate(sentence):
            x[i, t, char_indices[char]] = 1
```

```
        y[i, char_indices[next_chars[i]]] = 1

    # Build the character-based RNN model
    model = keras.Sequential()
    model.add(layers.LSTM(128, input_shape=(maxlen, len(chars))))
    model.add(layers.Dense(len(chars), activation='softmax'))

    # Compile the model
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    # Train the model
    model.fit(x, y, batch_size=128, epochs=50, verbose=0)

    # Generate text using the trained model
    def generate_text(seed_text, length):
        generated_text = seed_text

        for _ in range(length):
            x_pred = np.zeros((1, maxlen, len(chars)))

            for t, char in enumerate(seed_text):
                char_index = char_indices.get(char, char_indices[' '])  # Use '␣
 ↪' as the placeholder for unknown characters
                x_pred[0, t, char_index] = 1.

            preds = model.predict(x_pred, verbose=0)[0]
            next_index = np.random.choice(len(chars), p=preds)
            next_char = indices_char[next_index]
            seed_text += next_char
            seed_text = seed_text[1:]  # Move the sliding window
            generated_text += next_char

        return generated_text

    seed = "Mumbai "
    generated_text = generate_text(seed, length=100)
    print(generated_text)

# Define a function for word-based language model
def word_based_language_model():
    # Sample text data
    text = open('/content/Text.txt', 'r').read()

    # Tokenize the text
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts([text])
    total_words = len(tokenizer.word_index) + 1
```

```
    # Create input sequences and their corresponding labels
    input_sequences = []

    for line in text.split('\n'):
        token_list = tokenizer.texts_to_sequences([line])[0]

        for i in range(1, len(token_list)):
            n_gram_sequence = token_list[:i + 1]
            input_sequences.append(n_gram_sequence)

    # Pad sequences to have the same length
    max_sequence_length = max([len(seq) for seq in input_sequences])
    input_sequences = pad_sequences(input_sequences,␣
↪maxlen=max_sequence_length, padding='pre')

    # Create predictors and labels
    predictors, label = input_sequences[:, :-1], input_sequences[:, -1]

    # Convert labels to one-hot encoding
    label = keras.utils.to_categorical(label, num_classes=total_words)

    # Build the LSTM model
    model = Sequential()
    model.add(Embedding(total_words, 100, input_length=max_sequence_length - 1))
    model.add(LSTM(100))
    model.add(Dense(total_words, activation='softmax'))

    # Compile the model
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    # Train the model
    model.fit(predictors, label, epochs=50, verbose=0)

    # Generate text using the trained model
    def generate_text(seed_text, length):
        generated_text = seed_text

        for _ in range(length):
            x_pred = np.zeros((1, max_sequence_length - 1))
            token_list = tokenizer.texts_to_sequences([seed_text])[0]

            for t, token in enumerate(token_list):
                x_pred[0, t] = token

            preds = model.predict(x_pred, verbose=0)[0]
            next_index = np.random.choice(len(preds), p=preds)
```

```
            next_word = [word for word, index in tokenizer.word_index.items()␣
↪if index == next_index][0]

            seed_text += " " + next_word
            generated_text += " " + next_word

        return generated_text

    seed = "Mumbai"
    generated_text = generate_text(seed, length=20)
    print(generated_text)
```

[5]:
```
character_based_language_model()
```

```
<ipython-input-4-fc1a2c7ae84a>:32: DeprecationWarning: `np.bool` is a deprecated
alias for the builtin `bool`. To silence this warning, use `bool` by itself.
Doing this will not modify any behavior and is safe. If you specifically wanted
the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
<ipython-input-4-fc1a2c7ae84a>:33: DeprecationWarning: `np.bool` is a deprecated
alias for the builtin `bool`. To silence this warning, use `bool` by itself.
Doing this will not modify any behavior and is safe. If you specifically wanted
the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
```

```
Mumbai  lphsss pmlinre kcpltuteipnarusevit,blmmt   toigey.ieclooush,mitedii ec d
yegni see ouidcg.aosant1n
```

[6]:
```
word_based_language_model()
```

```
Mumbai on financial rickshaws it great environmental financial financial
financial west financial england ancient history england it the it mumbai
contributed
```