Text Generation Model using ML

A Text Generation Model is a type of Natural Language Processing (NLP) model that automatically generates human-like text. It can produce coherent and contextually relevant text based on the input text. So, if you want to learn how to build a Text Generation Model, this article is for you. In this article, I'll take you through the task of building a Text Generation Model with Deep Learning using the Python programming language.

Text Generation Model:

Text Generation Models have various applications, such as content creation, chatbots, automated story writing, and more. They often utilize advanced Machine Learning techniques, particularly Deep Learning models like Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), and Transformer models like GPT (Generative Pre-trained Transformer).

Below is the process we can follow for the task of building a Text Generation Model:

- Understand what you want to achieve with the text generation model (e.g., chatbot responses, creative writing, code generation).
- Consider the style, complexity, and length of the text to be generated.
- Collect a large dataset of text that's representative of the style and content you want to generate.
- Clean the text data (remove unwanted characters, correct spellings), and preprocess it (tokenization, lowercasing, removing stop words if necessary).
- Choose a deep neural network architecture to handle sequences for text generation.
- Frame the problem as a sequence modeling task where the model learns to predict the next words in a sequence.
- Use your text data to train the model.

For this task, we can use the Tiny Shakespeare dataset because of two reasons:

- It's available in the format of dialogues, so you will learn how to generate text in the form of dialogues.
- Usually, we need huge textual datasets for building text generation models. The Tiny Shakespeare dataset is already available in the tensorflow datasets, so we don't need to download any dataset externally.

So, let's understand how to build a Text Generation Model with Deep Learning using Python. I'll start this task by importing the necessary Python libraries and the dataset:

```
[1]  1 import tensorflow as tf
     2 import tensorflow_datasets as tfds
     3 import numpy as np
     4
     5 # load the Tiny Shakespeare dataset
     6 dataset, info = tfds.load('tiny_shakespeare', with_info=True, as_supervised=False)
```

```
Downloading and preparing dataset Unknown size (download: Unknown size, generated: 1.06 MiB, total: 1.06 MiB) to /root/tensorflow_datasets/tiny_shakespeare/1.0.0...
Dl Completed...: 100% 1/1 [00:00<00:00,  2.49 url/s]
Dl Size...:  1/0 [00:00<00:00,  2.77 MiB/s]
Dataset tiny_shakespeare downloaded and prepared to /root/tensorflow_datasets/tiny_shakespeare/1.0.0. Subsequent calls will reuse this data.
```

Our dataset contains data in a textual format. Language models need numerical data, so we'll convert the text to sequences of integers. We'll also create sequences for training:

```
 1 # get the text from the dataset
 2 text = next(iter(dataset['train']))['text'].numpy().decode('utf-8')
 3
 4 # create a mapping from unique characters to indices
 5 vocab = sorted(set(text))
 6 char2idx = {char: idx for idx, char in enumerate(vocab)}
 7 idx2char = np.array(vocab)
 8
 9 # numerically represent the characters
10 text_as_int = np.array([char2idx[c] for c in text])
11
12 # create training examples and targets
13 seq_length = 100
14 examples_per_epoch = len(text) // (seq_length + 1)
15
16 # create training sequences
17 char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
18
19 sequences = char_dataset.batch(seq_length + 1, drop_remainder=True)
```

For each sequence, we will now duplicate and shift it to form the input and target text by using the map method to apply a simple function to each batch:

```
1 def split_input_target(chunk):
2     input_text = chunk[:-1]
3     target_text = chunk[1:]
4     return input_text, target_text
5
6 dataset = sequences.map(split_input_target)
```

Now, we'll shuffle the dataset and pack it into training batches:

```
# batch size and buffer size
BATCH_SIZE = 64
BUFFER_SIZE = 10000

dataset = (
    dataset
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE, drop_remainder=True)
    .prefetch(tf.data.experimental.AUTOTUNE)
)
```

Now, we'll use a simple Recurrent Neural Network model with a few layers to build the model:


```
# length of the vocabulary
vocab_size = len(vocab)

# the embedding dimension
embedding_dim = 256

# number of RNN units
rnn_units = 1024

def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, embedding_dim, batch_input_shape=[batch_size, None]),
        tf.keras.layers.LSTM(rnn_units, return_sequences=True, stateful=True, recurrent_initializer='glorot_uniform'),
        tf.keras.layers.Dense(vocab_size)
    ])
    return model
model = build_model(vocab_size,embedding_dim,rnn_units, BATCH_SIZE)
```

We'll now choose an optimizer and a loss function to compile the model:

```
def loss(labels, logits):
    return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)

model.compile(optimizer='adam', loss=loss)
```

We'll now train the model:

```python
import os

# directory where the checkpoints will be saved
checkpoint_dir = './training_checkpoints'

# name of the checkpoint files
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True
)

# train the model
EPOCHS = 10
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

```
Epoch 1/10
155/155 [==============================] - 1225s 8s/step - loss: 2.6506
Epoch 2/10
155/155 [==============================] - 1184s 8s/step - loss: 1.9349
Epoch 3/10
155/155 [==============================] - 1181s 8s/step - loss: 1.6805
Epoch 4/10
155/155 [==============================] - 1169s 8s/step - loss: 1.5409
Epoch 5/10
155/155 [==============================] - 1175s 8s/step - loss: 1.4552
Epoch 6/10
155/155 [==============================] - 1179s 8s/step - loss: 1.3941
Epoch 7/10
155/155 [==============================] - 1176s 8s/step - loss: 1.3473
Epoch 8/10
155/155 [==============================] - 1172s 8s/step - loss: 1.3089
Epoch 9/10
155/155 [==============================] - 1182s 8s/step - loss: 1.2724
Epoch 10/10
155/155 [==============================] - 1194s 8s/step - loss: 1.2394
```

After training, we can now use the model to generate text. First, we will restore the latest checkpoint and rebuild the model with a batch size of 1:

```
[ ]  1 model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)
     2 model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
     3 model.build(tf.TensorShape([1, None]))
```

Now, to generate text, we'll input a seed string, predict the next character, and then add it back to the input, continuing this process to generate longer text:

```
def generate_text(model, start_string):
    num_generate = 1000

    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []

    model.reset_states()
    for i in range(num_generate):
        predictions = model(input_eval)
        predictions = tf.squeeze(predictions, 0)

        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1, 0].numpy()
        input_eval = tf.expand_dims([predicted_id], 0)

        text_generated.append(idx2char[predicted_id])

    return (start_string + ''.join(text_generated))
print(generate_text(model, start_string = u"QUEEN: So, let's end this"))
```

```
QUEEN: So, let's end this Apter, plead.

POMPEY:
I best derived to him; I wepm itself as liss
Are an other the slight to tell it. Whot as the foils of it?

QUEEN:
Nimbut your violence hath much suit impatient.

BUCKINGHAM:
As if how I myself in heaven. Come, come on,
If thou on the haply died, that lives may do you toad
That born thus grows our willing: if you spoke
Proud Anfortathou, yield us not. verming to that slarped?

SAMPSON:
Away, britting here: though wild, he spokes
As crainSLOND:
The telder love thy daughter wood, you shall kisply to ride,
Let him have had, and stand belongh'd,
What wait you do defand thy husband:
I will despair. I; by shuffly, and lies the read?

JOHN OF GAUNT:
Most offence, but here it does n grand.
How in them grace to ourdee-sophereing out.
What mulder-have I do come to be his very gradl?

DUKE VINCENTIO:
First, sir; that most morn perssition, tribunes!
Do Polfus will make sake your honour.

ISABELLA:
Do leass his cords, Within my most hour chosens,
That ever, were him lip, lead
```

The generate_text function in the above code uses a trained Recurrent Neural Network model to generate a sequence of text, starting with a given seed phrase (start_string). It converts the seed phrase into a sequence of numeric indices, feeds these indices into the model, and then iteratively generates new characters, each time using the model's most recent output as the input for the next step. This process continues for a specified number of iterations (num_generate), resulting in a stream of text that extends from the initial seed.

The function employs randomness in character selection to ensure variability in the generated text, and the final output is a concatenation of the seed phrase with the newly generated characters, typically reflecting the style and content of the training data used for the model.

Summary

So, this is how we build a Text Generation Model with Deep Learning using Python. Text Generation Models have various applications, such as content creation, chatbots, automated story writing, and more. They often utilize advanced Machine Learning techniques, particularly Deep Learning models like Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), and Transformer models like GPT (Generative Pre-trained Transformer).

Anmol Sonthalia
GitHub - anmol957