

Transformer for Text Generation

Transformer models are advanced neural networks designed to handle sequential data without using recurrence. Unlike RNN, LSTM, and GRU, transformers rely on a self-attention mechanism to capture relationships between all tokens in a sequence simultaneously. This allows transformers to model long-term dependencies more effectively and process data in parallel. In text generation, transformers learn contextual relationships between words or characters and generate more coherent and meaningful text compared to recurrent models.

```
# =====
# TRANSFORMER BASED TEXT GENERATION
# (SINGLE CELL - ASSIGNMENT READY)
# =====

import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import (
    Input, Dense, Embedding, LayerNormalization, MultiHeadAttention
)
from tensorflow.keras.models import Model

# -----
# DATASET
# -----
text = """
artificial intelligence is transforming modern society.
it is used in healthcare finance education and transportation.
machine learning allows systems to improve automatically with experience.
data plays a critical role in training intelligent systems.
large datasets help models learn complex patterns.
deep learning uses multi layer neural networks.
neural networks are inspired by biological neurons.
each neuron processes input and produces an output.
training a neural network requires optimization techniques.
gradient descent minimizes the loss function.

natural language processing helps computers understand human language.
text generation is a key task in nlp.
language models predict the next word or character.
recurrent neural networks handle sequential data.
lstm and gru models address long term dependency problems.
however rnn based models are slow for long sequences.

transformer models changed the field of nlp.
they rely on self attention mechanisms.
attention allows the model to focus on relevant context.
transformers process data in parallel.
this makes training faster and more efficient.
modern language models are based on transformers.

education is being improved using artificial intelligence.
intelligent tutoring systems personalize learning.
automated grading saves time for teachers.
online education platforms use recommendation systems.
technology enhances the quality of learning experiences.

ethical considerations are important in artificial intelligence.
fairness transparency and accountability must be ensured.
ai systems should be designed responsibly.
data privacy and security are major concerns.
researchers continue to improve ai safety.

text generation models can create stories poems and articles.
they are used in chatbots virtual assistants and content creation.
generated text should be meaningful and coherent.
evaluation of text generation is challenging.
human judgement is often required.

continuous learning is essential in the field of ai.
research and innovation drive technological progress.
students should build strong foundations in mathematics.
programming skills are important for ai engineers.
practical experimentation enhances understanding.
"""

text = text.lower().replace("\n", " ")
```

```

# -----
# TOKENIZATION
# -----
chars = sorted(list(set(text)))
char_to_index = {c: i for i, c in enumerate(chars)}
index_to_char = {i: c for c, i in char_to_index.items()}

vocab_size = len(chars)
seq_length = 40
embed_dim = 64
num_heads = 2
ff_dim = 128

# -----
# CREATE INPUT-OUTPUT SEQUENCES
# -----
X, y = [], []

for i in range(len(text) - seq_length):
    X.append([char_to_index[c] for c in text[i:i+seq_length]])
    y.append(char_to_index[text[i+seq_length]])

X = np.array(X)
y = np.array(y)

# -----
# POSITIONAL ENCODING
# -----
def positional_encoding(length, depth):
    positions = np.arange(length)[:, np.newaxis]
    depths = np.arange(depth)[np.newaxis, :]
    angle_rates = 1 / np.power(10000, (2 * (depths // 2)) / np.float32(depth))
    angle_rads = positions * angle_rates
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
    return tf.cast(angle_rads[np.newaxis, ...], tf.float32)

# -----
# BUILD TRANSFORMER MODEL
# -----
inputs = Input(shape=(seq_length,))
x = Embedding(vocab_size, embed_dim)(inputs)
x += positional_encoding(seq_length, embed_dim)

attention = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)(x, x)
x = LayerNormalization()(x + attention)

ffn = Dense(ff_dim, activation='relu')(x)
ffn = Dense(embed_dim)(ffn)
x = LayerNormalization()(x + ffn)

outputs = Dense(vocab_size, activation='softmax')(x[:, -1])

model = Model(inputs, outputs)

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

# -----
# TRAIN MODEL
# -----
model.fit(X, y, epochs=15, batch_size=64)

# -----
# TEXT GENERATION (WITH PADDING FIX)
# -----
def generate_text(seed_text, length=300):
    output = seed_text

    for _ in range(length):
        seq = output[-seq_length:]

        # pad sequence if shorter than seq_length
        if len(seq) < seq_length:
            seq = " " * (seq_length - len(seq)) + seq

        seq = np.array([[char_to_index[c] for c in seq]])
        prediction = model.predict(seq, verbose=0)
        next_char = index_to_char[np.argmax(prediction)]
        output += next_char

```

```
return output

# -----
# GENERATE TEXT
# -----
seed = "artificial intelligence "
generated_text = generate_text(seed)

print("Generated Text using Transformer:\n")
print(generated_text)
```

```
Epoch 1/15
34/34 ━━━━━━━━ 6s 3ms/step - loss: 3.0200
Epoch 2/15
34/34 ━━━━━━ 0s 3ms/step - loss: 2.8487
Epoch 3/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.8297
Epoch 4/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.6528
Epoch 5/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.5554
Epoch 6/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.4838
Epoch 7/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.4964
Epoch 8/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.4232
Epoch 9/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.4180
Epoch 10/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.4226
Epoch 11/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.4101
Epoch 12/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.3350
Epoch 13/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.3235
Epoch 14/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.2746
Epoch 15/15
34/34 ━━━━━━ 0s 4ms/step - loss: 2.3107
Generated Text using Transformer:
```

```
artificial intelligence te tere ane tere te tere te an ion are tere tere an ion ion are tere tere te are ane are tere ter
```

Limitations of Transformer Models

- Transformer models require large amounts of data to perform well.
- Training transformers is computationally expensive and memory-intensive.
- They are complex to design and tune compared to RNN-based models.
- Performance may be limited on very small datasets.
- Simpler models may be more suitable for small-scale applications.