

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Layer, Embedding, Dense,
LayerNormalization, Dropout
from tensorflow.keras.models import Model
```

```
2026-02-07 17:55:40.581269: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1770486940.602811    3504 cuda_dnn.cc:8579] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1770486940.609504    3504 cuda_blas.cc:1407] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
W0000 00:00:1770486940.626251    3504 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1770486940.626271    3504 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1770486940.626273    3504 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1770486940.626276    3504 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
```

```
data = """
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.

transformer models changed the field of nlp.

they rely on self attention mechanisms.

attention allows the model to focus on relevant context.

education is being improved using artificial intelligence.

intelligent tutoring systems personalize learning.

ethical considerations are important in artificial intelligence.

ai systems should be designed responsibly.

text generation models can create stories poems and articles.

generated text should be meaningful and coherent.

continuous learning is essential in the field of ai.

programming skills are important for ai engineers.

"""

```
tokenizer = Tokenizer()
```

```
tokenizer.fit_on_texts([data])
```

```
total_words = len(tokenizer.word_index) + 1
```

```
input_sequences = []
```

```
for line in data.split("\n"):
```

```
    token_list = tokenizer.texts_to_sequences([line])[0]
```

```
    for i in range(1, len(token_list)):
```

```
        input_sequences.append(token_list[:i+1])
```

```
max_len = max(len(x) for x in input_sequences)
```

```
input_sequences = pad_sequences(input_sequences, maxlen=max_len,  
padding='pre')
```

```
X = input_sequences[:, :-1]
```

```
y = input_sequences[:, -1]
```

```
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
```

```
print("X shape:", X.shape)
```

```
print("y shape:", y.shape)
```

```
X shape: (167, 8)
```

```
y shape: (167, 134)
```

```
class PositionalEncoding(Layer):
```

```
    def __init__(self, max_len, d_model):
```

```
        super().__init__()
```

```
        pos = np.arange(max_len)[: , np.newaxis]
```

```

        i = np.arange(d_model)[np.newaxis, :]
        angle_rates = 1 / np.power(10000,
(2*(i//2))/np.float32(d_model))
        angle_rads = pos * angle_rates

        angle_rads[:,0::2] = np.sin(angle_rads[:,0::2])
        angle_rads[:,1::2] = np.cos(angle_rads[:,1::2])

        self.pos_encoding = tf.cast(angle_rads[np.newaxis,...],
dtype=tf.float32)

    def call(self, x):
        return x + self.pos_encoding[:, :tf.shape(x)[1],:]

class TransformerBlock(Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super().__init__()
        self.att =
tf.keras.layers.MultiHeadAttention(num_heads=num_heads,
key_dim=embed_dim)
        self.ffn = tf.keras.Sequential([
            Dense(ff_dim, activation="relu"),
            Dense(embed_dim),
        ])
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate)
        self.dropout2 = Dropout(rate)

    def call(self, inputs, training=False):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)

        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)

embed_dim = 64
num_heads = 2
ff_dim = 128

inputs = tf.keras.Input(shape=(max_len-1,))

x = Embedding(total_words, embed_dim)(inputs)
x = PositionalEncoding(max_len, embed_dim)(x)

x = TransformerBlock(embed_dim, num_heads, ff_dim)(x)

x = tf.keras.layers.GlobalAveragePooling1D()(x)

```

```
outputs = Dense(total_words, activation="softmax")(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])

model.summary()

I0000 00:00:1770486944.576623    3504 gpu_device.cc:2019] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 13757 MB
memory:  -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0,
compute capability: 7.5
I0000 00:00:1770486944.581714    3504 gpu_device.cc:2019] Created
device /job:localhost/replica:0/task:0/device:GPU:1 with 13757 MB
memory:  -> device: 1, name: Tesla T4, pci bus id: 0000:00:05.0,
compute capability: 7.5

Model: "functional_1"
```

Layer (type) Param #	Output Shape	
input_layer (InputLayer) 0	(None, 8)	
embedding (Embedding) 8,576	(None, 8, 64)	
positional_encoding 0 (PositionalEncoding)	(None, 8, 64)	
transformer_block 50,048 (TransformerBlock)	(None, 8, 64)	
global_average_pooling1d 0	(None, 64)	

(GlobalAveragePooling1D)		
dense_2 (Dense)	(None, 134)	
8,710		

Total params: 67,334 (263.02 KB)

Trainable params: 67,334 (263.02 KB)

Non-trainable params: 0 (0.00 B)

```
history = model.fit(X, y, epochs=100, verbose=1)
```

Epoch 1/100

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

```
I0000 00:00:1770486948.257164    3543 service.cc:152] XLA service
0x7f4ef8012960 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
I0000 00:00:1770486948.257193    3543 service.cc:160]   StreamExecutor
device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1770486948.257197    3543 service.cc:160]   StreamExecutor
device (1): Tesla T4, Compute Capability 7.5
I0000 00:00:1770486948.762793    3543 cuda_dnn.cc:529] Loaded cuDNN
version 91002
```

1/6 ————— 28s 6s/step - accuracy: 0.0000e+00 - loss: 5.3042

```
I0000 00:00:1770486951.343290    3543 device_compiler.h:188] Compiled
cluster using XLA! This line is logged at most once for the lifetime
of the process.
```

6/6 ————— 9s 566ms/step - accuracy: 0.0026 - loss: 5.1692

Epoch 2/100

6/6 ————— 0s 7ms/step - accuracy: 0.0335 - loss: 4.8913

Epoch 3/100


6/6 ————— 0s 7ms/step - accuracy: 0.0238 - loss: 4.7030


Epoch 4/100


6/6 ————— 0s 7ms/step - accuracy: 0.0534 - loss: 4.7526


Epoch 5/100


6/6 ————— 0s 7ms/step - accuracy: 0.0318 - loss: 4.6990


Epoch 6/100
6/6  0s 7ms/step - accuracy: 0.0402 - loss: 4.7135


Epoch 7/100
6/6  0s 7ms/step - accuracy: 0.0234 - loss: 4.6786


Epoch 8/100
6/6  0s 7ms/step - accuracy: 0.0428 - loss: 4.6273


Epoch 9/100
6/6  0s 7ms/step - accuracy: 0.0227 - loss: 4.6790


Epoch 10/100
6/6  0s 7ms/step - accuracy: 0.0420 - loss: 4.6580

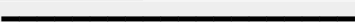
Epoch 11/100
6/6  0s 7ms/step - accuracy: 0.0420 - loss: 4.5963


Epoch 12/100
6/6  0s 7ms/step - accuracy: 0.0268 - loss: 4.6257

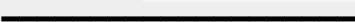
Epoch 13/100
6/6  0s 7ms/step - accuracy: 0.0301 - loss: 4.5415

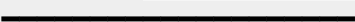
Epoch 14/100
6/6  0s 7ms/step - accuracy: 0.0610 - loss: 4.5650

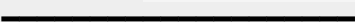
Epoch 15/100
6/6  0s 7ms/step - accuracy: 0.0394 - loss: 4.5021

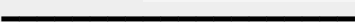
Epoch 16/100
6/6  0s 8ms/step - accuracy: 0.0521 - loss: 4.5402

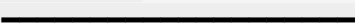
Epoch 17/100
6/6  0s 7ms/step - accuracy: 0.0379 - loss: 4.5842

Epoch 18/100
6/6  0s 7ms/step - accuracy: 0.0571 - loss: 4.5393

Epoch 19/100
6/6  0s 7ms/step - accuracy: 0.0699 - loss: 4.5285

Epoch 20/100
6/6  0s 7ms/step - accuracy: 0.0638 - loss: 4.4158

Epoch 21/100
6/6  0s 7ms/step - accuracy: 0.0599 - loss: 4.4074

Epoch 22/100
6/6  0s 7ms/step - accuracy: 0.0960 - loss: 4.2829

Epoch 23/100
6/6  0s 7ms/step - accuracy: 0.1012 - loss: 4.1772

Epoch 24/100
6/6  0s 7ms/step - accuracy: 0.1101 - loss: 4.1218

Epoch 25/100
6/6  0s 7ms/step - accuracy: 0.0776 - loss: 4.1013

Epoch 26/100
6/6  0s 7ms/step - accuracy: 0.1411 - loss: 3.9599

Epoch 27/100
6/6  0s 7ms/step - accuracy: 0.1282 - loss: 3.8171

Epoch 28/100
6/6  0s 7ms/step - accuracy: 0.1716 - loss: 3.8193

Epoch 29/100
6/6  0s 7ms/step - accuracy: 0.2214 - loss: 3.6451

Epoch 30/100
6/6  0s 7ms/step - accuracy: 0.1585 - loss: 3.5663

Epoch 31/100
6/6  0s 7ms/step - accuracy: 0.2353 - loss: 3.5263

Epoch 32/100
6/6  0s 8ms/step - accuracy: 0.2824 - loss: 3.2724

Epoch 33/100
6/6  0s 7ms/step - accuracy: 0.2296 - loss: 3.1355

Epoch 34/100
6/6  0s 7ms/step - accuracy: 0.2849 - loss: 3.1485


Epoch 35/100
6/6  0s 7ms/step - accuracy: 0.3576 - loss: 2.9364

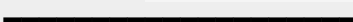
Epoch 36/100
6/6  0s 7ms/step - accuracy: 0.3605 - loss: 2.8148

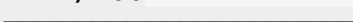
Epoch 37/100
6/6  0s 7ms/step - accuracy: 0.3797 - loss: 2.6675

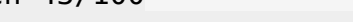
Epoch 38/100
6/6  0s 8ms/step - accuracy: 0.4902 - loss: 2.5255

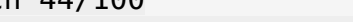
Epoch 39/100
6/6  0s 7ms/step - accuracy: 0.4311 - loss: 2.5238

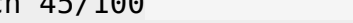
Epoch 40/100
6/6  0s 7ms/step - accuracy: 0.4680 - loss: 2.3703

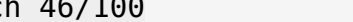
Epoch 41/100
6/6  0s 7ms/step - accuracy: 0.5060 - loss: 2.2473

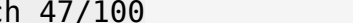
Epoch 42/100
6/6  0s 7ms/step - accuracy: 0.5357 - loss: 2.1056

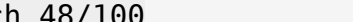
Epoch 43/100
6/6  0s 7ms/step - accuracy: 0.5645 - loss: 2.0879

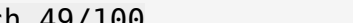
Epoch 44/100
6/6  0s 8ms/step - accuracy: 0.5976 - loss: 1.9968

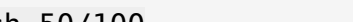
Epoch 45/100
6/6  0s 7ms/step - accuracy: 0.6063 - loss: 1.9906

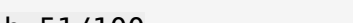
Epoch 46/100
6/6  0s 7ms/step - accuracy: 0.6183 - loss: 1.8503

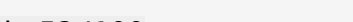
Epoch 47/100
6/6  0s 7ms/step - accuracy: 0.5918 - loss: 1.8410

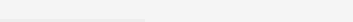
Epoch 48/100
6/6  0s 7ms/step - accuracy: 0.6490 - loss: 1.6699

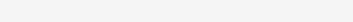
Epoch 49/100
6/6  0s 7ms/step - accuracy: 0.6474 - loss: 1.6129

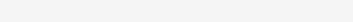
Epoch 50/100
6/6  0s 7ms/step - accuracy: 0.7177 - loss: 1.5071

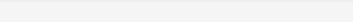
Epoch 51/100
6/6  0s 7ms/step - accuracy: 0.7076 - loss: 1.5050


Epoch 52/100
6/6  0s 7ms/step - accuracy: 0.7512 - loss: 1.3850

Epoch 53/100
6/6  0s 7ms/step - accuracy: 0.7636 - loss: 1.2906

Epoch 54/100
6/6  0s 7ms/step - accuracy: 0.7726 - loss: 1.2173

Epoch 55/100
6/6  0s 7ms/step - accuracy: 0.8107 - loss: 1.1726

Epoch 56/100
6/6  0s 7ms/step - accuracy: 0.8455 - loss: 1.1760

Epoch 57/100
6/6  0s 7ms/step - accuracy: 0.8253 - loss: 1.0851

Epoch 58/100
6/6  0s 7ms/step - accuracy: 0.8580 - loss: 1.0282

Epoch 59/100
6/6  0s 8ms/step - accuracy: 0.8908 - loss: 0.9576

Epoch 60/100
6/6  0s 7ms/step - accuracy: 0.8605 - loss: 0.9193


Epoch 61/100
6/6  0s 7ms/step - accuracy: 0.9170 - loss: 0.8160

Epoch 62/100
6/6  0s 7ms/step - accuracy: 0.9103 - loss: 0.8144

Epoch 63/100
6/6  0s 7ms/step - accuracy: 0.9126 - loss: 0.7741

Epoch 64/100
6/6  0s 8ms/step - accuracy: 0.9224 - loss: 0.7050

Epoch 65/100
6/6  0s 8ms/step - accuracy: 0.9544 - loss: 0.6867

Epoch 66/100
6/6  0s 7ms/step - accuracy: 0.9417 - loss: 0.6864

Epoch 67/100
6/6  0s 7ms/step - accuracy: 0.9637 - loss: 0.6623

Epoch 68/100
6/6  0s 7ms/step - accuracy: 0.9490 - loss: 0.6348

Epoch 69/100
6/6  0s 7ms/step - accuracy: 0.9585 - loss: 0.5572

Epoch 70/100
6/6  0s 7ms/step - accuracy: 0.9568 - loss: 0.5640

Epoch 71/100
6/6  0s 7ms/step - accuracy: 0.9737 - loss: 0.5072

Epoch 72/100
6/6  0s 7ms/step - accuracy: 0.9736 - loss: 0.5355

Epoch 73/100
6/6  0s 7ms/step - accuracy: 0.9431 - loss: 0.4915

Epoch 74/100
6/6  0s 7ms/step - accuracy: 0.9758 - loss: 0.4499

Epoch 75/100
6/6  0s 8ms/step - accuracy: 0.9725 - loss: 0.4172

Epoch 76/100
6/6  0s 7ms/step - accuracy: 0.9617 - loss: 0.4203

Epoch 77/100
6/6  0s 8ms/step - accuracy: 0.9751 - loss: 0.4126

Epoch 78/100
6/6  0s 7ms/step - accuracy: 0.9868 - loss: 0.3663

Epoch 79/100
6/6  0s 8ms/step - accuracy: 0.9810 - loss: 0.3600

Epoch 80/100
6/6  0s 7ms/step - accuracy: 0.9575 - loss: 0.3494

Epoch 81/100
6/6  0s 7ms/step - accuracy: 0.9911 - loss: 0.3356

Epoch 82/100
6/6  0s 7ms/step - accuracy: 0.9621 - loss: 0.3217

Epoch 83/100
6/6  0s 7ms/step - accuracy: 0.9844 - loss: 0.2916

Epoch 84/100
6/6  0s 7ms/step - accuracy: 0.9676 - loss: 0.2829

Epoch 85/100
6/6  0s 7ms/step - accuracy: 0.9769 - loss: 0.2844


Epoch 86/100
6/6  0s 7ms/step - accuracy: 0.9777 - loss: 0.2665

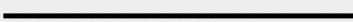
Epoch 87/100
6/6  0s 7ms/step - accuracy: 0.9786 - loss: 0.2326


Epoch 88/100
6/6  0s 7ms/step - accuracy: 0.9885 - loss: 0.2500

Epoch 89/100
6/6  0s 7ms/step - accuracy: 0.9874 - loss: 0.2507


Epoch 90/100
6/6  0s 7ms/step - accuracy: 0.9819 - loss: 0.2226


Epoch 91/100
6/6  0s 7ms/step - accuracy: 0.9825 - loss: 0.2151


Epoch 92/100
6/6  0s 8ms/step - accuracy: 0.9728 - loss: 0.2239


Epoch 93/100
6/6  0s 7ms/step - accuracy: 0.9786 - loss: 0.2021


Epoch 94/100
6/6  0s 7ms/step - accuracy: 0.9851 - loss: 0.1995


Epoch 95/100
6/6  0s 7ms/step - accuracy: 0.9922 - loss: 0.1914

Epoch 96/100
6/6  0s 7ms/step - accuracy: 0.9881 - loss: 0.1864

Epoch 97/100
6/6  0s 7ms/step - accuracy: 0.9908 - loss: 0.1740

Epoch 98/100
6/6  0s 7ms/step - accuracy: 0.9725 - loss: 0.1767

Epoch 99/100
6/6  0s 7ms/step - accuracy: 0.9736 - loss: 0.1680

Epoch 100/100
6/6  0s 7ms/step - accuracy: 0.9836 - loss: 0.1589

```
def generate_transformer_text(seed_text, next_words=20):  
    for _ in range(next_words):  
        token_list = tokenizer.texts_to_sequences([seed_text])[0]  
        token_list = pad_sequences([token_list], maxlen=max_len-1,  
padding='pre')  
        predicted = model.predict(token_list, verbose=0)  
        predicted_word = tokenizer.index_word[np.argmax(predicted)]  
        seed_text += " " + predicted_word  
    return seed_text
```

```
print(generate_transformer_text("artificial intelligence", 25))
```

artificial intelligence is transforming modern society society society
the field of nlp nlp human language processing helps computers
understand human language language processing helps computers
understand human

UI

```
import gradio as gr

def transformer_ui(seed_text, length):
    return generate_transformer_text(seed_text, int(length))

demo = gr.Interface(
    fn=transformer_ui,
    inputs=[
        gr.Textbox(
            label="Enter Seed Text",
            value="artificial intelligence",
            max_lines=5
        ),
        gr.Slider(
            minimum=5,
            maximum=50,
            value=20,
            step=1,
            label="Number of Words"
        )
    ],
    outputs=gr.Textbox(
        label="Generated Text",
        lines=15,
        max_lines=25,
        show_copy_button=True
    ),
    title="Transformer Text Generation UI",
    description="GenAI Lab-4 Component-II – Transformer Based Text Generation"
)

demo.launch(debug=True)
```

* Running on local URL: <http://127.0.0.1:7860>

It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True` (you can turn this off by setting `share=False` in `launch()` explicitly).

* Running on public URL: <https://c3373bb95df108bf13.gradio.live>

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (<https://huggingface.co/spaces>)

<IPython.core.display.HTML object>