In [1]:
```
# import libraries
import numpy as np
import tensorflow as tf
```

In [2]:
```
# load text and covert to lowercase
filename = "alice_in_wonderland.txt"
raw_text = open(filename, 'r', encoding='utf-8').read()
raw_text = raw_text.lower()
raw_text = raw_text[:3000]
```

In [3]:
```
# create tokenizer to convert from string to integers
tokenizer = tf.keras.preprocessing.text.Tokenizer()
tokenizer.fit_on_texts([raw_text])

# convert text to sequences
sequences = tokenizer.texts_to_sequences([raw_text])[0]

# define parameters
vocab_size = len(tokenizer.word_index) + 1
seq_length = 20
```

In [4]:
```
# prepare the dataset of input to output pairs encoded as integers
input = []
output = []

for i in range(seq_length, len(sequences)):
    input.append(sequences[i-seq_length:i])
    output.append(sequences[i])
```

In [5]:
```
# convert to arrays
X = np.array(input)
y = np.array(output)
```

In [6]:
```
embed_dim = 256 # embedding dimension
num_heads = 4 # number of attention heads
num_layers = 4  # number of transformer layers
```

In [13]:
```
# positions
def positional_encoding(position, embed_dim):
    angle_rads = np.arange(position)[:, np.newaxis] / np.power(10000, (2 * (np.arange(embed_dim) // 2)) / np.float32(embed_dim))
    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, ...], dtype=tf.float32)
```

In [55]:
```
class MultiHeadSelfAttention(tf.keras.layers.Layer):
    def __init__(self, embed_dim, num_heads):
        super(MultiHeadSelfAttention, self).__init__()
        self.embed_dim = embed_dim # model's dimension
        self.num_heads = num_heads # number of attention heads
        self.dept = embed_dim // num_heads # dimension of each head's vectors

        # layers for query, key, and value vectors
        self.wq = tf.keras.layers.Dense(embed_dim)
        self.wk = tf.keras.layers.Dense(embed_dim)
        self.wv = tf.keras.layers.Dense(embed_dim)
        self.dense = tf.keras.layers.Dense(embed_dim) # final classification layer

    # reshape input to have num_heads for multi-head attention
    def split_heads(self, x, batch_size):
        x = tf.reshape(x, (batch_size, -1, self.num_heads, self.dept))
        return tf.transpose(x, perm=[0, 2, 1, 3])

    # finding attention scores
    def call(self, inputs):
        batch_size = tf.shape(inputs)[0]

        # create vectors from input sequences
        q = self.wq(inputs)
        k = self.wk(inputs)
        v = self.wv(inputs)

        # split into multiple heads
        q = self.split_heads(q, batch_size)
        k = self.split_heads(k, batch_size)
        v = self.split_heads(v, batch_size)

        # calculate attention scores from vectors above
        scaled_attention_logits = tf.matmul(q, k, transpose_b=True) / tf.sqrt(tf.cast(self.dept, tf.float32)) # scaling
        attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1) # normalize

        # attention weights x values
        output = tf.matmul(attention_weights, v)
        output = tf.transpose(output, perm=[0, 2, 1, 3])
        output = tf.reshape(output, (batch_size, -1, self.embed_dim))
        return self.dense(output) # return classification layer

class TransformerBlock(tf.keras.layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim):
        super(TransformerBlock, self).__init__()

        # attention layer and feed forward network
        self.attention = MultiHeadSelfAttention(embed_dim, num_heads)
        self.ffn = tf.keras.Sequential([
            tf.keras.layers.Dense(ff_dim, activation='relu'),
            tf.keras.layers.Dense(embed_dim)
        ])

        # normalize and dropout layers
```

```
        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = tf.keras.layers.Dropout(0.1)
        self.dropout2 = tf.keras.layers.Dropout(0.1)

    # get outputs for attention and feed forward layers
    def call(self, inputs, training):
        attn_output = self.attention(inputs)
        out1 = self.layernorm1(inputs + self.dropout1(attn_output, training=training))
        ffn_output = self.ffn(out1)
        return self.layernorm2(out1 + self.dropout2(ffn_output, training=training))

class TransformerModel(tf.keras.Model):
    def __init__(self, input_dim, embed_dim, num_heads, num_layers, output_dim):
        super(TransformerModel, self).__init__()
        self.embedding = tf.keras.layers.Embedding(input_dim, embed_dim) # embedding layer
        self.transformer_blocks = [TransformerBlock(embed_dim, num_heads, embed_dim * 4) for _ in range(num_layers)] # transformer blocks
        self.fc = tf.keras.layers.Dense(output_dim) # output layer

    def call(self, x, training=False):
        embedded = self.embedding(x) # convert input to embeddings
        for transformer in self.transformer_blocks:
            embedded = transformer(embedded, training=training) # pass through blocks
        output = tf.reduce_mean(embedded, axis=1) # pooling over sequence length
        return self.fc(output) # final output
```

In [57]:
```
# model
model = TransformerModel(vocab_size, embed_dim, num_heads, num_layers, vocab_size)
```

In [59]:
```
# compile model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

In [61]:
```
# train model
model.fit(X, y, epochs=100, batch_size=32, validation_split=0.2)
```

```
Epoch 1/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 4s 117ms/step - accuracy: 0.0371 - loss: 11.5119 - val_accuracy: 0.0467 - val_loss: 14.7607
Epoch 2/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 106ms/step - accuracy: 0.0447 - loss: 11.5693 - val_accuracy: 0.0000e+00 - val_loss: 14.7265
Epoch 3/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 110ms/step - accuracy: 0.0074 - loss: 12.4116 - val_accuracy: 0.0000e+00 - val_loss: 14.9643
Epoch 4/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 113ms/step - accuracy: 0.0075 - loss: 12.0519 - val_accuracy: 0.0000e+00 - val_loss: 15.0955
Epoch 5/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 104ms/step - accuracy: 0.0036 - loss: 13.0864 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 6/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 117ms/step - accuracy: 0.0029 - loss: 12.6786 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 7/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 120ms/step - accuracy: 0.0085 - loss: 12.5812 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 8/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 118ms/step - accuracy: 0.0013 - loss: 13.2095 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 9/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 122ms/step - accuracy: 0.0085 - loss: 12.1709 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 10/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0027 - loss: 12.4185 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 11/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 131ms/step - accuracy: 0.0056 - loss: 12.1573 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 12/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - accuracy: 0.0019 - loss: 12.7627 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 13/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0080 - loss: 13.2654 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 14/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 120ms/step - accuracy: 0.0067 - loss: 12.5404 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 15/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.0039 - loss: 12.4256 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 16/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - accuracy: 0.0041 - loss: 13.2260 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 17/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 142ms/step - accuracy: 0.0070 - loss: 13.2671 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 18/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - accuracy: 0.0019 - loss: 12.2266 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 19/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 122ms/step - accuracy: 0.0085 - loss: 12.8649 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 20/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.0035 - loss: 12.7851 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 21/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 127ms/step - accuracy: 0.0025 - loss: 12.8211 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 22/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 133ms/step - accuracy: 0.0069 - loss: 12.8263 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 23/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 125ms/step - accuracy: 9.6132e-04 - loss: 12.9865 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 24/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 120ms/step - accuracy: 0.0023 - loss: 12.2489 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 25/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.0080 - loss: 12.4549 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 26/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.0041 - loss: 12.7828 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 27/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - accuracy: 0.0047 - loss: 12.5445 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 28/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 120ms/step - accuracy: 0.0075 - loss: 12.6645 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 29/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.0108 - loss: 12.5848 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 30/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 122ms/step - accuracy: 0.0019 - loss: 12.6126 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 31/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 128ms/step - accuracy: 0.0057 - loss: 12.8303 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 32/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.0022 - loss: 13.4064 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 33/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.0028 - loss: 13.0374 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 34/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 121ms/step - accuracy: 0.0011 - loss: 12.5210 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 35/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.0034 - loss: 12.6107 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 36/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 131ms/step - accuracy: 0.0055 - loss: 11.9738 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 37/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.0047 - loss: 12.6354 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 38/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 121ms/step - accuracy: 0.0064 - loss: 12.3742 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 39/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 140ms/step - accuracy: 0.0041 - loss: 12.4792 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 40/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.0015 - loss: 12.6238 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 41/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.0028 - loss: 12.5613 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 42/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 131ms/step - accuracy: 0.0055 - loss: 12.3906 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 43/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0036 - loss: 12.9309 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 44/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 135ms/step - accuracy: 0.0020 - loss: 12.6360 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 45/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 138ms/step - accuracy: 0.0032 - loss: 12.5002 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 46/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 120ms/step - accuracy: 0.0069 - loss: 12.3009 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 47/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 121ms/step - accuracy: 0.0027 - loss: 12.6192 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 48/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 121ms/step - accuracy: 0.0032 - loss: 12.5002 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 49/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 129ms/step - accuracy: 0.0018 - loss: 12.6870 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 50/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 132ms/step - accuracy: 0.0082 - loss: 13.1048 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 51/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.0023 - loss: 12.7170 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
```

```
Epoch 52/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0080 - loss: 12.8694 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 53/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.0054 - loss: 12.8108 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 54/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0069 - loss: 12.5577 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 55/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 127ms/step - accuracy: 0.0053 - loss: 12.5630 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 56/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 139ms/step - accuracy: 0.0085 - loss: 12.2283 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 57/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 129ms/step - accuracy: 0.0025 - loss: 12.6154 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 58/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 158ms/step - accuracy: 0.0011 - loss: 12.8771 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 59/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 127ms/step - accuracy: 0.0048 - loss: 13.0719 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 60/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.0041 - loss: 12.9084 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 61/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.0027 - loss: 13.0678 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 62/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 121ms/step - accuracy: 0.0057 - loss: 12.9209 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 63/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 121ms/step - accuracy: 0.0044 - loss: 12.9296 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 64/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 131ms/step - accuracy: 0.0044 - loss: 12.7719 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 65/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.0039 - loss: 12.3631 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 66/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 131ms/step - accuracy: 0.0020 - loss: 12.4760 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 67/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 145ms/step - accuracy: 0.0021 - loss: 13.0923 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 68/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 133ms/step - accuracy: 0.0040 - loss: 12.7625 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 69/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 142ms/step - accuracy: 0.0078 - loss: 13.1618 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 70/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 139ms/step - accuracy: 0.0032 - loss: 12.6766 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 71/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 129ms/step - accuracy: 0.0118 - loss: 12.6906 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 72/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 136ms/step - accuracy: 0.0031 - loss: 12.7670 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 73/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 153ms/step - accuracy: 0.0053 - loss: 13.0393 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 74/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 146ms/step - accuracy: 0.0052 - loss: 12.5553 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 75/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.0035 - loss: 12.6661 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 76/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 141ms/step - accuracy: 0.0069 - loss: 12.9887 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 77/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.0053 - loss: 12.2246 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 78/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 129ms/step - accuracy: 0.0044 - loss: 12.3243 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 79/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 127ms/step - accuracy: 0.0028 - loss: 12.6716 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 80/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 133ms/step - accuracy: 0.0021 - loss: 13.0202 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 81/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 145ms/step - accuracy: 0.0080 - loss: 12.7500 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 82/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 129ms/step - accuracy: 0.0108 - loss: 12.4382 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 83/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 145ms/step - accuracy: 0.0055 - loss: 12.8421 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 84/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 133ms/step - accuracy: 0.0118 - loss: 12.4707 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 85/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0040 - loss: 12.7377 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 86/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 131ms/step - accuracy: 0.0024 - loss: 12.9263 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 87/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.0017 - loss: 13.1974 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 88/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.0045 - loss: 12.7530 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 89/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 140ms/step - accuracy: 0.0027 - loss: 12.3310 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 90/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 139ms/step - accuracy: 0.0016 - loss: 12.5224 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 91/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 133ms/step - accuracy: 0.0042 - loss: 12.7301 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 92/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0041 - loss: 12.5822 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 93/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 127ms/step - accuracy: 0.0034 - loss: 12.7609 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 94/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 128ms/step - accuracy: 0.0015 - loss: 12.5757 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 95/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 138ms/step - accuracy: 0.0076 - loss: 13.2624 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 96/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 133ms/step - accuracy: 0.0036 - loss: 12.8245 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 97/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 134ms/step - accuracy: 0.0045 - loss: 12.4632 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 98/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0034 - loss: 12.8155 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 99/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 128ms/step - accuracy: 0.0082 - loss: 12.8996 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
Epoch 100/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 130ms/step - accuracy: 0.0025 - loss: 13.0563 - val_accuracy: 0.0000e+00 - val_loss: 15.0647
```

Out[61]: <keras.src.callbacks.history.History at 0x3036d0bc0>

In [62]:
```
# example prediction
input_example = np.array(X[110:111])
```

```
predicted = model.predict(input_example)
predicted_word_index = np.argmax(predicted, axis=-1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step
```

In [63]:
```
# get the actual
predicted_word = tokenizer.index_word[predicted_word_index[0]]
input_indices = input_example[0]
actual_next_index = y[110]
actual_next_word = tokenizer.index_word[actual_next_index]

# Show results
print("Input sequence:", [tokenizer.index_word.get(index) for index in input_indices if index > 0])
print("Predicted word:", predicted_word)
print("Actual next word:", actual_next_word)
```

```
Input sequence: ['a', 'daisy', 'chain', 'would', 'be', 'worth', 'the', 'trouble', 'of', 'getting', 'up', 'and', 'picking', 'the', 'daisies', 'when', 'su
ddenly', 'a', 'white', 'rabbit']
Predicted word: dear
Actual next word: with
```

In [ ]:
```
```