Bonus Lab 3: Introduction to Transformers in R and Python

Dr. Derrick L. Cogburn

2024-11-15

Table of contents

1 Lab Overview

1.1 Learning Objectives

1. Understand Transformers and their architecture
2. Implement attention mechanisms
3. Learn how to use pre-trained Transformer models
4. Implement a text classification model using BERT

Overview

In this bonus lab we introduce the Transformer model, which is a deep learning model that has been widely used in natural language processing (NLP) tasks. The Transformer model was introduced in the paper "Attention is All You Need" by Vaswani et al. (2017) and has since become a popular model for a wide range of NLP tasks, including machine translation, text generation, and text classification.

1.2 There are three main parts to the lab:

Exercise 1: Using Hugging Face Transformers in R

Exercise 2: Using Hugging Face Transformers in Python

Exercise 3: Implementing Attention Mechanisms

Exercise 4: Text Classification with BERT

Exercise 5: Visualization of Attention

2 Lab Instructions

Make sure you have the following packages installed in R: `transformers`, `keras`, `tensorflow`, `tidyverse`, `ggplot2`, and in Python: `transformers`, `tensorflow`, `matplotlib`, `numpy`.; and in Python: `transformers`, `tensorflow`, `matplotlib`, `numpy`.

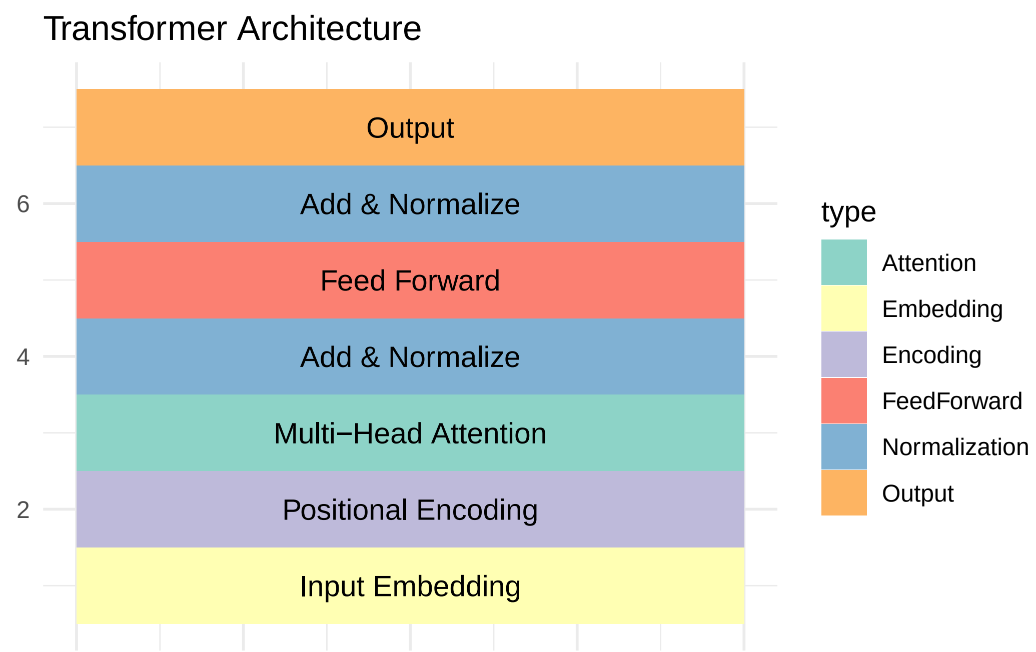
2.1 Exercise 1: Understanding the Transformer Architecture in R

```r
library(keras)
library(tensorflow)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becon
```

```r
library(reticulate)
library(tokenizers)

# Import transformers
transformers <- reticulate::import("transformers")
```

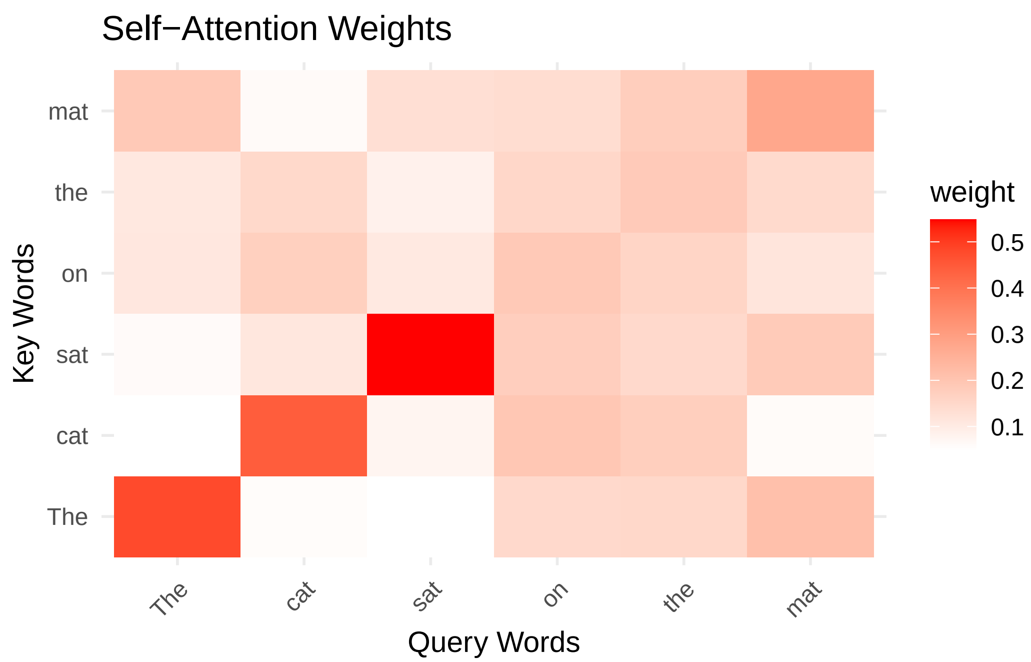
```r
# Function to visualize transformer architecture
plot_transformer_architecture <- function() {
  # Create data for visualization
  architecture_data <- tibble(
    component = c("Input Embedding", "Positional Encoding",
                  "Multi-Head Attention", "Add & Normalize",
                  "Feed Forward", "Add & Normalize",
                  "Output"),
    level = 1:7,
    type = c("Embedding", "Encoding", "Attention", "Normalization",
             "FeedForward", "Normalization", "Output")
  )

  ggplot(architecture_data, aes(x = 1, y = level, fill = type)) +
    geom_tile(width = 0.8) +
    geom_text(aes(label = component)) +
    theme_minimal() +
    theme(axis.text.x = element_blank(),
          axis.title = element_blank()) +
    scale_fill_brewer(palette = "Set3") +
    labs(title = "Transformer Architecture")
}

plot_transformer_architecture()
```



2.2 Exercise 2: Visualizing Attention in R

```r
# Add necessary libraries
library(keras)
library(tensorflow)

# Define softmax function if not using keras/tensorflow version
manual_softmax <- function(x) {
    exp_x <- exp(x - max(x))  # Subtract max for numerical stability
    exp_x / sum(exp_x)
}

# Modified calculate_attention function with proper softmax
calculate_attention <- function(query, key, value) {
    # Scaled dot-product attention
    attention_scores <- query %*% t(key) / sqrt(ncol(key))

    # Option 1: Using manual softmax
    attention_weights <- t(apply(attention_scores, 1, manual_softmax))

    # Option 2: Alternative using tensorflow if you prefer
```

```r
    # attention_weights <- tf$nn$softmax(attention_scores)$numpy()

    attention_output <- attention_weights %*% value

    list(
        scores = attention_scores,
        weights = attention_weights,
        output = attention_output
    )
}

# Example usage
text <- "The cat sat on the mat"
words <- strsplit(text, " ")[[1]]

# Create simple embeddings for demonstration
word_vectors <- matrix(rnorm(length(words) * 4),
                      nrow = length(words),
                      ncol = 4)

attention_result <- calculate_attention(word_vectors, word_vectors, word_vectors)

# Visualization remains the same
plot_attention_matrix <- function(attention_weights, words) {
    attention_df <- expand.grid(
        word1 = factor(words, levels = words),
        word2 = factor(words, levels = words)
    )
    attention_df$weight <- as.vector(attention_weights)

    ggplot(attention_df, aes(x = word1, y = word2, fill = weight)) +
        geom_tile() +
        scale_fill_gradient(low = "white", high = "red") +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
        labs(title = "Self-Attention Weights",
             x = "Query Words",
             y = "Key Words")
}

# Plot the attention matrix
plot_attention_matrix(attention_result$weights, words)
```



2.3 Exercise 1: Using Hugging Face Transformers in R

```r
library(keras)
library(tensorflow)
library(reticulate)
library(tidyverse)
library(gridExtra)
```

```
Attaching package: 'gridExtra'

The following object is masked from 'package:dplyr':

    combine
```

```r
# Import transformers
transformers <- reticulate::import("transformers")

# Modified inspection function with safer tensor handling
inspect_attention_weights <- function(text) {
```

```r
# Initialize tokenizer and model
tokenizer <- transformers$BertTokenizer$from_pretrained('bert-base-uncased')
model <- transformers$TFBertModel$from_pretrained('bert-base-uncased')

# Tokenize input
inputs <- tokenizer$encode_plus(
    text,
    return_tensors = "tf",
    add_special_tokens = TRUE,
    return_attention_mask = TRUE
)

# Convert inputs to tensorflow tensors explicitly
input_ids <- inputs$input_ids
attention_mask <- inputs$attention_mask

# Get model outputs
outputs <- model(
    list(
        input_ids = input_ids,
        attention_mask = attention_mask
    ),
    output_attentions = TRUE
)

# Safely convert attention weights to R
attention_weights <- lapply(outputs$attentions, function(x) {
    # Convert TensorFlow tensor to numpy array, then to R array
    as.array(x$numpy())
})

# Print structure information
cat("Number of layers:", length(attention_weights), "\n")
cat("Shape of attention weights for first layer:\n")
print(dim(attention_weights[[1]]))

# Get tokens
tokens <- tokenizer$convert_ids_to_tokens(input_ids[1,])
cat("\nTokens:\n")
print(tokens)

return(list(
```

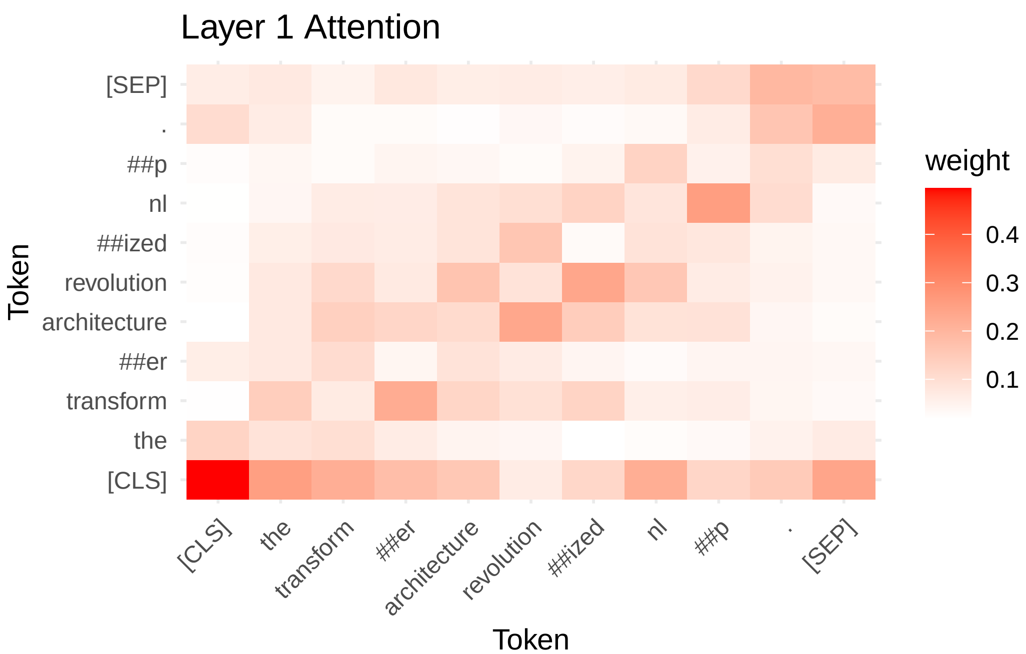
```
        attention_weights = attention_weights,
        tokens = tokens
    ))
}

# Modified visualization function
plot_single_layer_attention <- function(result, layer_num) {
    # Get attention weights and tokens
    attention_weights <- result$attention_weights
    tokens <- result$tokens

    # Get attention weights for specific layer
    layer_attention <- attention_weights[[layer_num]]

    # Average across attention heads
    avg_attention <- apply(layer_attention[1,,,], c(2,3), mean)

    # Create data frame for plotting
    attention_df <- expand.grid(
        token1 = factor(tokens, levels = tokens),
        token2 = factor(tokens, levels = tokens)
    )
    attention_df$weight <- as.vector(avg_attention)

    # Create plot
    ggplot(attention_df, aes(x = token1, y = token2, fill = weight)) +
        geom_tile() +
        scale_fill_gradient(low = "white", high = "red") +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
        labs(title = paste("Layer", layer_num, "Attention"),
             x = "Token",
             y = "Token")
}

# Full visualization function
visualize_bert_attention <- function(text) {
    # Get attention weights and tokens
    result <- inspect_attention_weights(text)
    attention_weights <- result$attention_weights
    tokens <- result$tokens
```

```r
    # Create plots for each layer
    plots <- list()
    for(layer in seq_along(attention_weights)) {
        # Average attention weights across heads for this layer
        layer_attention <- attention_weights[[layer]]
        avg_attention <- apply(layer_attention[1,,,], c(2,3), mean)

        # Create data frame for plotting
        attention_df <- expand.grid(
            token1 = factor(tokens, levels = tokens),
            token2 = factor(tokens, levels = tokens)
        )
        attention_df$weight <- as.vector(avg_attention)

        # Create plot
        p <- ggplot(attention_df,
                   aes(x = token1, y = token2, fill = weight)) +
            geom_tile() +
            scale_fill_gradient(low = "white", high = "red") +
            theme_minimal() +
            theme(axis.text.x = element_text(angle = 45, hjust = 1),
                  axis.text.y = element_text(angle = 0)) +
            labs(title = paste("Layer", layer, "Attention"),
                 x = "Token",
                 y = "Token") +
            coord_fixed()

        plots[[layer]] <- p
    }

    # Arrange plots in a grid
    do.call(grid.arrange, c(plots, ncol = 3))
}

# Example text
texts <- c(
    "The transformer architecture revolutionized NLP.",
    "Attention mechanisms help models focus on relevant parts.",
    "BERT learns contextual word representations."
)

# Try the inspection first
```

```r
result <- inspect_attention_weights(texts[1])
```

```
Number of layers: 12
Shape of attention weights for first layer:
[1]  1 12 11 11

Tokens:
 [1] "[CLS]"        "the"          "transform"    "##er"         "architecture"
 [6] "revolution"   "##ized"       "nl"           "##p"          "."
[11] "[SEP]"
```

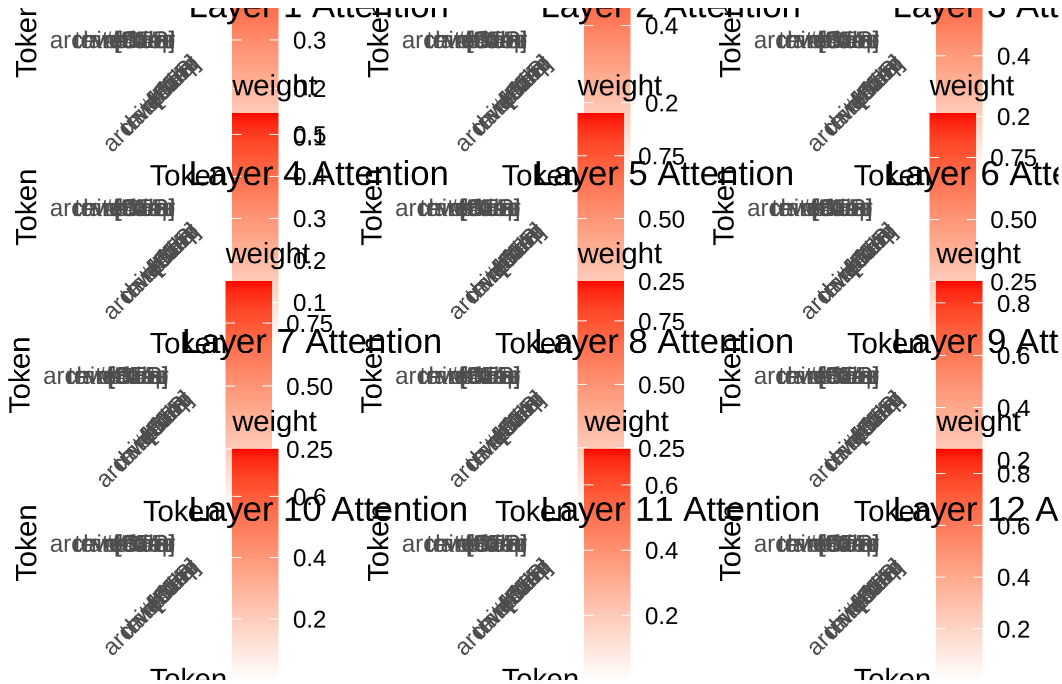
```r
# Try plotting a single layer
plot_single_layer_attention(result, 1)
```




```r
# If that works, try the full visualization
visualize_bert_attention(texts[1])
```

```
Number of layers: 12
Shape of attention weights for first layer:
[1]  1 12 11 11
```

```
Tokens:
 [1] "[CLS]"         "the"           "transform"     "##er"          "architecture"
 [6] "revolution"    "##ized"        "nl"            "##p"           "."
[11] "[SEP]"
```



Now, let's take a look at using transformers in Python

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from transformers import BertTokenizer, TFBertModel
import pandas as pd
from sklearn.manifold import TSNE

## Part 1: Transformer Architecture Visualization

def plot_transformer_architecture():
    """Visualize basic transformer architecture"""
    components = ['Input Embedding', 'Positional Encoding',
                  'Multi-Head Attention', 'Add & Normalize',
                  'Feed Forward', 'Add & Normalize', 'Output']
```

```
    types = ['Embedding', 'Encoding', 'Attention', 'Normalization',
             'FeedForward', 'Normalization', 'Output']

    fig, ax = plt.subplots(figsize=(10, 8))
    y_positions = np.arange(len(components))

    # Create colored boxes
    for i, (component, type_) in enumerate(zip(components, types)):
        ax.add_patch(plt.Rectangle((0.2, i-0.4), 0.6, 0.8,
                                   facecolor=plt.cm.Set3(i/len(components))))
        ax.text(0.5, i, component, ha='center', va='center')

    ax.set_ylim(-0.5, len(components)-0.5)
    ax.set_xlim(0, 1)
    ax.axis('off')
    plt.title('Transformer Architecture')
    plt.tight_layout()
    plt.show()

## Part 2: Self-Attention Visualization

def calculate_attention(query, key, value):
    """Calculate attention scores and weights"""
    attention_scores = np.dot(query, key.T) / np.sqrt(key.shape[1])
    attention_weights = tf.nn.softmax(attention_scores, axis=-1).numpy()
    attention_output = np.dot(attention_weights, value)
    return attention_scores, attention_weights, attention_output

def plot_attention_matrix(attention_weights, words):
    """Visualize attention weights between words"""
    fig, ax = plt.subplots(figsize=(10, 8))
    sns.heatmap(attention_weights,
                xticklabels=words,
                yticklabels=words,
                cmap='YlOrRd',
                annot=True,
                fmt='.2f')
    plt.title('Self-Attention Weights')
    plt.xlabel('Key Words')
    plt.ylabel('Query Words')
    plt.tight_layout()
    plt.show()
```

```python
## Part 3: BERT Implementation

class BertVisualizer:
    def __init__(self):
        self.tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
        self.model = TFBertModel.from_pretrained('bert-base-uncased')

    def visualize_tokenization(self, text):
        """Visualize BERT tokenization"""
        # Tokenize text
        tokens = self.tokenizer.encode(text, add_special_tokens=True)
        token_words = self.tokenizer.convert_ids_to_tokens(tokens)

        # Create visualization
        fig, ax = plt.subplots(figsize=(15, 2))
        for i, token in enumerate(token_words):
            color = 'pink' if token in ['[CLS]', '[SEP]'] else 'lightblue'
            ax.add_patch(plt.Rectangle((i, 0), 1, 1, facecolor=color))
            ax.text(i+0.5, 0.5, token, ha='center', va='center')

        ax.set_xlim(0, len(token_words))
        ax.set_ylim(0, 1)
        ax.axis('off')
        plt.title(f'BERT Tokenization: "{text}"')
        plt.tight_layout()
        plt.show()

    def visualize_attention_layers(self, text):
        """Visualize attention patterns across BERT layers"""
        # Encode text and get attention
        inputs = self.tokenizer(text, return_tensors="tf",
                              add_special_tokens=True)
        outputs = self.model(inputs, output_attentions=True)
        attention_weights = outputs.attentions

        # Convert to numpy and average across heads
        attention_array = np.array([att.numpy() for att in attention_weights])
        avg_attention = np.mean(attention_array, axis=2)  # Average across heads

        # Plot attention for each layer
        n_layers = avg_attention.shape[0]
        fig, axes = plt.subplots(3, 4, figsize=(20, 15))
```

```
        axes = axes.ravel()

        tokens = self.tokenizer.convert_ids_to_tokens(
            inputs['input_ids'][0].numpy())

        for i in range(n_layers):
            sns.heatmap(avg_attention[i, 0],
                        xticklabels=tokens,
                        yticklabels=tokens,
                        ax=axes[i],
                        cmap='YlOrRd')
            axes[i].set_title(f'Layer {i+1}')
            axes[i].tick_params(axis='both', rotation=90)

        plt.tight_layout()
        plt.show()

    def visualize_embeddings(self, text):
        """Visualize word embeddings using t-SNE"""
        # Get embeddings
        inputs = self.tokenizer(text, return_tensors="tf",
                               add_special_tokens=True)
        outputs = self.model(inputs)
        embeddings = outputs.last_hidden_state[0].numpy()

        # Perform t-SNE
        tsne = TSNE(n_components=2, perplexity=5, random_state=42)
        embeddings_2d = tsne.fit_transform(embeddings)

        # Get tokens
        tokens = self.tokenizer.convert_ids_to_tokens(
            inputs['input_ids'][0].numpy())

        # Plot
        plt.figure(figsize=(12, 8))
        plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1], alpha=0.5)
        for i, token in enumerate(tokens):
            plt.annotate(token,
                         (embeddings_2d[i, 0], embeddings_2d[i, 1]),
                         xytext=(5, 5),
                         textcoords='offset points')
        plt.title('Word Embeddings Visualization (t-SNE)')
```

```
        plt.tight_layout()
        plt.show()

## Example Usage

# Initialize BERT visualizer
bert_viz = BertVisualizer()

# Example texts
texts = [
    "The transformer architecture revolutionized NLP.",
    "Attention mechanisms help models focus on relevant parts.",
    "BERT learns contextual word representations."
]

# Demonstrate visualizations
def demonstrate_transformer_visualizations(texts):
    # Show architecture
    plot_transformer_architecture()

    # Show tokenization for each text
    for text in texts:
        bert_viz.visualize_tokenization(text)

    # Show attention patterns
    bert_viz.visualize_attention_layers(texts[0])

    # Show embeddings
    bert_viz.visualize_embeddings(texts[0])

    # Demonstrate simple self-attention
    # Create dummy embeddings for demonstration
    words = texts[0].split()
    word_vectors = np.random.randn(len(words), 4)
    _, attention_weights, _ = calculate_attention(word_vectors,
                                                  word_vectors,
                                                  word_vectors)
    plot_attention_matrix(attention_weights, words)

## Additional Analysis Functions

def analyze_attention_patterns(text, bert_viz):
```

```
    """Analyze how attention changes across layers"""
    inputs = bert_viz.tokenizer(text, return_tensors="tf",
                               add_special_tokens=True)
    outputs = bert_viz.model(inputs, output_attentions=True)
    attention_weights = outputs.attentions

    # Analyze attention to [CLS] token
    cls_attention = np.array([att.numpy()[0, :, 0, :]
                            for att in attention_weights])

    # Plot attention to [CLS] across layers
    plt.figure(figsize=(12, 6))
    sns.heatmap(cls_attention.mean(axis=1),
                xticklabels=bert_viz.tokenizer.convert_ids_to_tokens(
                    inputs['input_ids'][0].numpy()),
                yticklabels=range(1, cls_attention.shape[0] + 1),
                cmap='YlOrRd')
    plt.title('Attention to [CLS] Token Across Layers')
    plt.xlabel('Tokens')
    plt.ylabel('Layer')
    plt.tight_layout()
    plt.show()

def visualize_layer_similarities(text, bert_viz):
    """Visualize similarities between layer representations"""
    inputs = bert_viz.tokenizer(text, return_tensors="tf",
                               add_special_tokens=True)
    outputs = bert_viz.model(inputs, output_hidden_states=True)
    hidden_states = outputs.hidden_states

    # Calculate similarities between layers
    n_layers = len(hidden_states)
    similarities = np.zeros((n_layers, n_layers))

    for i in range(n_layers):
        for j in range(n_layers):
            hi = hidden_states[i].numpy()
            hj = hidden_states[j].numpy()
            similarity = np.corrcoef(hi.flatten(), hj.flatten())[0, 1]
            similarities[i, j] = similarity

    # Plot similarities
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(similarities,
            xticklabels=range(n_layers),
            yticklabels=range(n_layers),
            cmap='YlOrRd')
plt.title('Layer Representation Similarities')
plt.xlabel('Layer')
plt.ylabel('Layer')
plt.tight_layout()
plt.show()
```