Visualization Tools for Self-Attention Mechanisms

Yugen. (Github - https://github.com/2201512/MLAttentionVisualizations)

1 Introduction

This document categorizes various visualization tools for self-attention mechanisms.

*Coloured text are Hyperlinked

2 Simple Heatmap using MatplotLib for NLP

Use GPT-2 as the model and ascertain the attention weight in the last layer, for the sentence ”A new sentence to visualize attention weights”.

Listing 1: Python code to visualize attention weights

```python
from transformers import pipeline, GPT2Tokenizer, GPT2Model
import matplotlib.pyplot as plt
import seaborn as sns
import torch

# Step 1: Set up the pipeline
# This creates a text generation pipeline using the GPT-2 model.
# The pipeline allows us to generate text based on a given input prompt.
pipe = pipeline("text-generation", model="openai-community/gpt2")

# Step 2: Generate text and retrieve attention weights
# Initialize the tokenizer and model for GPT-2.
# Setting 'output_attentions=True' ensures that the model returns attention weights.
tokenizer = GPT2Tokenizer.from_pretrained("openai-community/gpt2")
model = GPT2Model.from_pretrained("openai-community/gpt2", output_attentions=True)

# Define the input text
text = "A␣new␣sentence␣to␣visualize␣attention␣weights"

# Tokenize the input text to convert it into a format suitable for the model
inputs = tokenizer(text, return_tensors="pt")

# Pass the tokenized input through the model to get the output and attention weights
outputs = model(**inputs)

# Extract attention weights from the last layer (layer -1)
# Attention weights indicate how much focus the model places on different tokens when processing the input.
attention = outputs.attentions[-1]

# Aggregate attention heads by taking the mean
# This simplifies the visualization by averaging the attention scores from multiple heads.
attention = torch.mean(attention, dim=1).squeeze().detach().numpy()

# Define a function to plot the attention heatmap
# This function creates a heatmap of attention weights, showing the relationship between input tokens.
def plot_attention_heatmap(attention_weights, input_tokens, title="Attention␣Heatmap"):
    plt.figure(figsize=(10, 8))
    sns.heatmap(attention_weights, xticklabels=input_tokens, yticklabels=input_tokens, cmap="viridis")
    plt.title(title)
    plt.xlabel("Input␣Tokens")
    plt.ylabel("Output␣Tokens")
    plt.show()

# Convert token IDs back to token strings for labeling the heatmap
input_tokens = tokenizer.convert_ids_to_tokens(inputs["input_ids"].squeeze().tolist())

# Plot the attention heatmap
plot_attention_heatmap(attention, input_tokens)
```

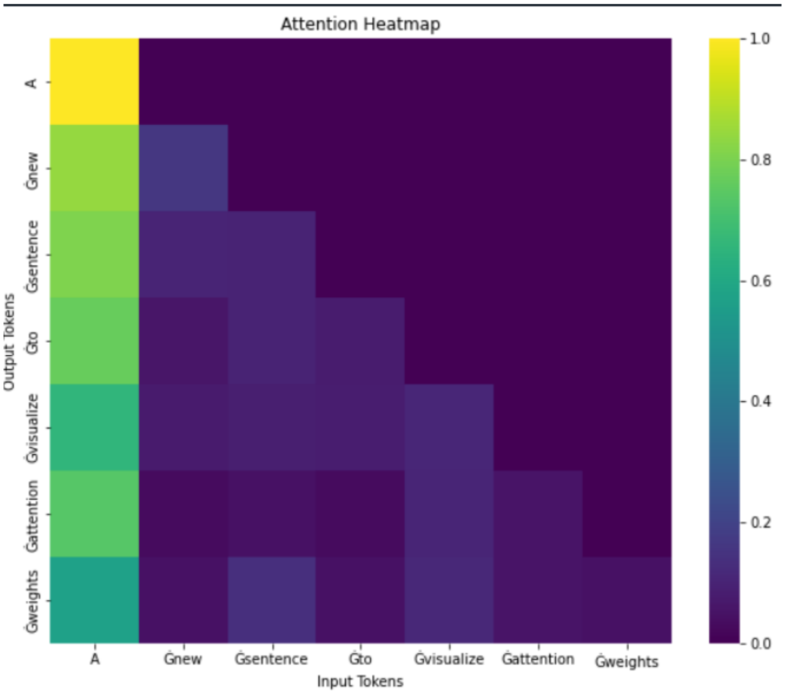
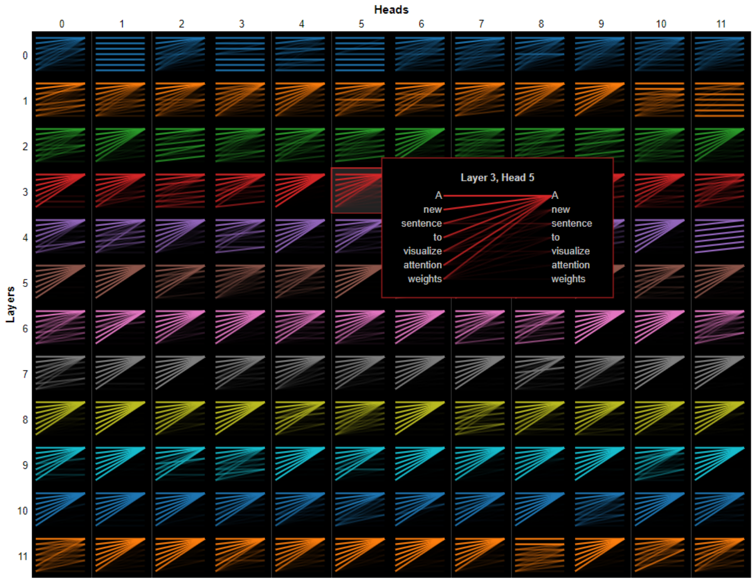


Figure 1: Resultant HeatMap

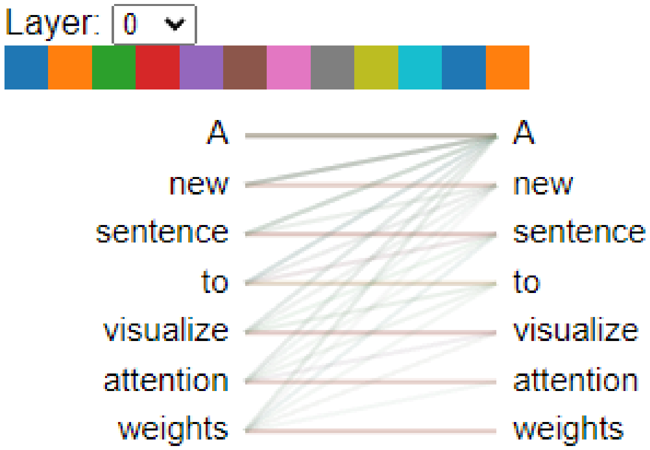
3 BertViz

BertViz

- Allows for head, model, and neuron view
- Seems more applicable to NLP



Model View



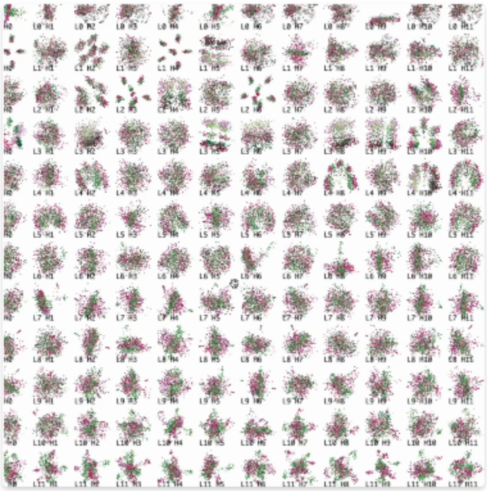
Layer: 0

Head View

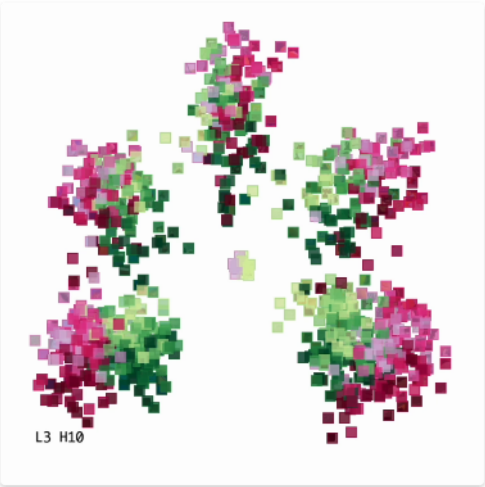
Listing 2: BertViz Script for the Sentence ”A new sentence to visualize attention weights” create a HTML file with the Model View and Head View.

```
from transformers import GPT2Tokenizer, GPT2Model, utils
from bertviz import head_view, model_view
import torch

# Suppress standard warnings
utils.logging.set_verbosity_error()

# Load the tokenizer and model with output_attentions=True
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2Model.from_pretrained("gpt2", output_attentions=True)

# Define the input text
text = "A␣new␣sentence␣to␣visualize␣attention␣weights"

# Tokenize the input text
inputs = tokenizer.encode(text, return_tensors=’pt’)

# Pass the inputs through the model to get the outputs, including attention weights
outputs = model(inputs)

# Extract attention weights from the outputs
attention = outputs.attentions

# Convert token IDs to tokens
tokens = tokenizer.convert_ids_to_tokens(inputs[0])

# Generate the head view HTML representation
html_head_view = head_view(attention, tokens, html_action=’return’)

# Save the head view HTML to a file
with open("head_view.html", ’w’) as file:
    file.write(html_head_view.data)

# Generate the model view HTML representation
html_model_view = model_view(attention, tokens, html_action=’return’)

# Save the model view HTML to a file
with open("model_view.html", ’w’) as file:
    file.write(html_model_view.data)

# Optionally, display the HTML in Jupyter (if applicable)
import IPython.display as display
display.display(html_head_view)
display.display(html_model_view)
```

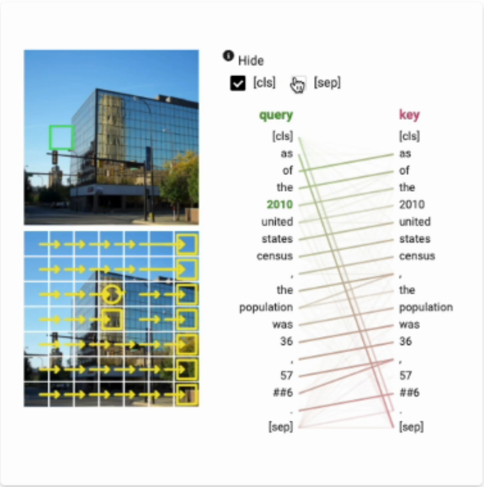
4 AttentionViz



Matrix View



Single View



Image/Sentence View

AttentionViz

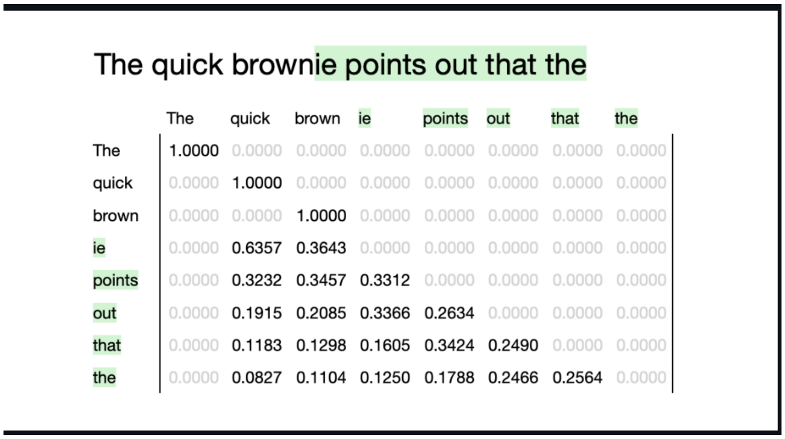
- Matrix view to view all attention heads
- Single view to seek each attention head

- Image/sentence view for patterns
- Applicable to both visual and NLP

5 Attention by Matt Neary

Attention by Matt Neary

- Visualization using normalized means (sigmoid function)
- Applicable to NLP



The quick brownie points out that the

	The	quick	brown	ie	points	out	that	the
The	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
quick	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
brown	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
ie	0.0000	0.6357	0.3643	0.0000	0.0000	0.0000	0.0000	0.0000
points	0.0000	0.3232	0.3457	0.3312	0.0000	0.0000	0.0000	0.0000
out	0.0000	0.1915	0.2085	0.3366	0.2634	0.0000	0.0000	0.0000
that	0.0000	0.1183	0.1298	0.1605	0.3424	0.2490	0.0000	0.0000
the	0.0000	0.0827	0.1104	0.1250	0.1788	0.2466	0.2564	0.0000

Normalized using Sigmoid Function

6 More Attention Visualization Libraries Using Heatmaps

- Attention Transfer
- Transformer-Explainability

7 Visualization Libraries for Saliency

- Learning Interpretability Tool (LIT)
- Ecco
- Transformers-Interpret

8 General Purpose Visualization Libraries

- TensorBoard for TensorFlow
- Captum for PyTorch