In [1]:
```
#!pip install bertviz
#!pip install ipywidgets
```

In [1]:
```
from bertviz.transformers_neuron_view import GPT2Model, GPT2Tokenizer
from bertviz.transformers_neuron_view import BertModel, BertTokenizer
from bertviz import model_view
from transformers import AutoTokenizer, AutoModel, utils
from bertviz.neuron_view import show

utils.logging.set_verbosity_error()  # Suppress standard warnings
```

BERT outgrew Sesame Street

We have seen how attention works mathematically, but now, let's explore how it works in practice. For this part of the question, we will be visualizing how this simple mechanism allows the powerful large language models of today to function. We will be using an open source visualization tool called BertViz (https://github.com/jessevig/bertviz).

Recall some of the powerful large language models we have studied so far:

- The GPT language model is a statistical language model that is autoregressive in nature, and it uses a deep neural network (specifically a transformer decoder) to predict the next word in a sequence. It is trained on a large corpus of text and can be used to generate text that sounds like it was written by a human. (This description was written by GPT-3. How meta!)
- BERT is a transfomer encoder that has been pre-trained with two tasks, which allow it to learn better representations for downstream tasks. First, it learns word-level associations by trying to fill in tokens that have been randomly masked with a 15% probability. Second, it learns sentence-level associations by trying to identify which sentences go first, given a randomly shuffled passage. This base model is then fine-tuned for different tasks, such as question-answering and text-infill. You will be fine-tuning your own BERT model in a later question (`Coding Question: Summarization (Part II)`). This model is bidirectional in nature as it can process text in both directions, from left-to-right and from right-to-left.

Fun fact: Back in the 2018, language models had very interesting names based on muppets. Well, there's a ELMo, BERT, and there's an ERNIE (and this name has been claimed twice!).

About BertViz

Attention gives us some insight into how these language models form representations about the tokens they interact with, and BertViz is an interactive tool that allows us to visualize attention effectively. We will be using BertViz's `model view` to see how words that are being updated (in the left column of the plots you will generate below) are connected to the words being attended to (in the right column of the plots you will generate below).

The lines in the plots represent the attention connections: when the attention score is close to 1, the line color is strong, and when the attention score is close to 0, the line is faint. You can also see the queries, keys, and values that resulted in those attention scores by hovering over the tokens as explained below.

Please refer to the BertViz github page linked above if you are interested in learning more!

Note: Attention visualization only gives us a window into how the model is learning. However, understanding these large language models and the connections they make is actually really nuanced and complex. In fact, it is the subject of ongoing research.

BertViz Usage (from their github page):

- Hover over any of the tokens on the left side of the visualization to see what tokens are being paid attention to at that moment.
- Then, click on the + icon that is revealed when hovering. This will reveal the query vectors, key vectors, and intermediate computations for the attention weights (blue=positive, orange=negative).
- Once in the expanded view, hover over any other token on the left to see the associated attention computations.
- Click on the Layer or Head drop-downs to change the model layer or head (zero-indexed).

a) Attention in GPT-2

We will first be using BertViz in order to visualize how attention works within GPT. For the purposes of this homework, we will be loading pre-trained models from Hugging Face as training takes an extremely long time.

In [2]:
```
model_type = 'gpt2'
model_version = 'gpt2'
model = GPT2Model.from_pretrained(model_version)
tokenizer = GPT2Tokenizer.from_pretrained(model_version)
```

Let's see what the model pays attention to when the sentence structure is simple and basic.

In [3]:
```
text = "The dog ran"
show(model, model_type, tokenizer, text, display_mode='dark')
```

Layer: Head:

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

Let's try a more complicated sentence structure.

In [4]:
```
text = "The dog sitting in the car"
show(model, model_type, tokenizer, text, display_mode='dark')
```

Layer: Head:

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

What happens when the model is tasked with keeping track of information from the past? Oftentimes, we need to look at the broader context that can span sentences or paragraphs to know the correct tense of a verb to use, which names to fill in where, etc. This is where the transformers excel--at keeping track of history.

In [5]:
```
text = "Jaime Salazar is running for election. Despite his party affiliation, Mr."
show(model, model_type, tokenizer, text, display_mode='dark')
```

Layer: Head:

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

Answer the following questions in your writeup:

1. What similarities and differences do you notice in the visualizations between the examples in part (a)? Explore the queries, keys, and values to identify any interesting patterns associated with the attention mechanism.
2. How does attention differ between the different layers of the GPT model? Do you notice that the tokens are attending to different tokens as we go through the layers of the network?

b) BERT pays attention

Let's now use BertViz to see how attention works within the BERT model.

In [6]:
```
model_type = 'bert'
model_version = 'bert-base-uncased'
do_lower_case = True
model = BertModel.from_pretrained(model_version)
tokenizer = BertTokenizer.from_pretrained(model_version, do_lower_case=do_lower_case
```

First, let's try a simple set of sentences where sentence b follows sentence a sequentially. Notice how we have a pronoun reference to the "party" mentioned in sentence a.

In [7]:
```
sentence_a = "I wanted to have a party"
sentence_b = "I bought a cake for it."
show(model, model_type, tokenizer, sentence_a, sentence_b, display_mode='dark', laye
```

Layer: Head: Attention: All

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

Answer the following questions in your writeup:

3. Look at different layers of the BERT model in the visualizations of part (b) and identify different patterns associated with the attention mechanism. Explore the queries, keys, and values to further inform your answer. For instance, do you notice that any particular type of tokens are attended to at a given timestep?
4. Do you spot any differences between how attention works in GPT vs. BERT? Think about

Let's understand what BERT does when it sees two sentences where words are used in multiple different ways. For instance, the word "play" has a couple of different meanings. So, what words will the model attend on, and what differences will we notice in the embeddings?

In [8]:
```
sentence_a = "I was going to play at the park."
show(model, model_type, tokenizer, sentence_a, display_mode='dark', layer=2, head=0)
```

Layer: Head:

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

In [9]:
```
sentence_b = "I was going to a play in the park"
show(model, model_type, tokenizer, sentence_b, display_mode='dark', layer=2, head=0)
```

Layer: Head:

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

Answer the following question in your writeup:

5. Look through the different layers of the two BERT networks above associated with sentence a and sentence b, and take a look at the queries, keys, and values associated with the different tokens. Do you notice any differences in the embeddings learned for the two sentences that are essentially identical in structure but different in meaning?

BERT is able to take care of input given from left-to-right and from right-to-left. Let's see what happens if we pass in a sentence backwards!

In [10]:
```
sentence_a = "party a have to wanted I"
sentence_b = "I bought a cake for it"
show(model, model_type, tokenizer, sentence_a, sentence_b, display_mode='dark', laye
```

Layer: Head: Attention: All

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

BERT was also pre-trained to identify which sentences come first sequentially. What happens if we pass in sentences in reverse order sequentially?

```
In [11]: sentence_a = "I bought a cake for it"
         sentence_b = "I went to a party at my neighbor's house"
         show(model, model_type, tokenizer, sentence_a, sentence_b, display_mode='dark', laye
```

Layer: Head: Attention: All

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

Answer the following question in your writeup:

6. Do you notice BERT's bidirectionality in play?
7. Do you think pre-training the BERT helped it learn better representations?

c) BERT has multiple heads!

Recall that BERT uses multiple attention heads in practice.Let's visualize BERT's multiple attention heads.

```
In [12]: model_version = 'bert-base-uncased'
         model = AutoModel.from_pretrained(model_version, output_attentions=True)
         tokenizer = AutoTokenizer.from_pretrained(model_version)
```

```
Downloading (…)lve/main/config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

d:\Anaconda\Anaconda_setup\envs\malning\lib\site-packages\huggingface_hub\file_do
wnload.py:133: UserWarning: `huggingface_hub` cache-system uses symlinks by defau
lt to efficiently store duplicated files but your machine does not support them i
n C:\Users\cyt\.cache\huggingface\hub. Caching files will still work but in a deg
raded version that might require more space on your disk. This warning can be dis
abled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For
more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limita
tions. (https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.)
To support symlinks on Windows, you either need to activate Developer Mode or to
run Python as an administrator. In order to see activate developer mode, see this
article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-de
vice-for-development (https://docs.microsoft.com/en-us/windows/apps/get-started/e
nable-your-device-for-development)
  warnings.warn(message)

Downloading model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Downloading (…)okenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

In [13]:
```
sentence_a = "I wanted to have a party"
sentence_b = "I like Thanksgiving dinner"
inputs = tokenizer.encode_plus(sentence_a, sentence_b, return_tensors='pt')
input_ids = inputs['input_ids']
token_type_ids = inputs['token_type_ids'] # token type id is 0 for Sentence A and 1
attention = model(input_ids, token_type_ids=token_type_ids)[-1]
sentence_b_start = token_type_ids[0].tolist().index(1) # Sentence B starts at first
token_ids = input_ids[0].tolist() # Batch index 0
tokens = tokenizer.convert_ids_to_tokens(token_ids)
model_view(attention, tokens, sentence_b_start)
```

Attention: All

```
<IPython.core.display.Javascript object>
```

Answer the following questions in your writeup:

8. Do you notice different features being learned throughout the different attention heads of BERT? Why do you think this might be?
9. Can you identify any of the different features that the different attention heads are focusing on?

These were just some small examples, but we encourage you to play around with this visualization tool and these pre-trained models on your own! There are some other cool models that are accessible through Hugging Face (https://huggingface.co/models). If you come across anything interesting, please mention it in your writeup!

d) Visualizing untrained attention weights

So far, we've been looking at the learned attention heads of the BERT model, trained on billions of tokens. Now, let's see how the attention heads behave without their weights. The code block below reinitializes most of the network weights. Re-run some prior cells to observe the difference.

Answer the following questions in your writeup:

10. What differences do you notice in the attention patterns between the randomly initialized and trained BERT models?
11. Run the final cell in the notebook. What are some words or tokens that you would expect strong attention between? What might you guess about the gradients of this attention head for those words?

In [14]:
```
def init_weights(m):
    try:
        m.reset_parameters()
    except:
        pass

model = BertModel.from_pretrained(model_version)
model = model.apply(init_weights)
```

In [15]:
```
sentence_a = "I am happy"
sentence_b = "I am not sad"
show(model, model_type, tokenizer, sentence_a, sentence_b, display_mode='dark', laye
```

Layer: Head: Attention: All

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```