In [3]:

```
from transformers import AutoModel, AutoTokenizer, AutoModelForCausalLM, EncoderDecoderModel, AutoModelForSequenceClass
from datasets import load_dataset
import numpy as np
import evaluate
```

```
/Users/fernport/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/tqdm/auto.py:21: TqdmWarning: IPro
gress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.htm
l
  from .autonotebook import tqdm as notebook_tqdm
```

Text Generation

GPT2

In [2]:

```
model = AutoModelForCausalLM.from_pretrained("gpt2")
tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

In [3]:

```
prompt = "The future of AI is "
input_ids = tokenizer(prompt, return_tensors="pt").input_ids
```

In [4]:

```
gen_tokens = model.generate(
    input_ids,
    max_length=20,
    num_return_sequences=1,
    do_sample=True,
    temperature=0.9,
)
gen_text = tokenizer.batch_decode(gen_tokens)[0]
print(gen_text)
```

```
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pas
s your input's `attention_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequenc
e, you may observe unexpected behavior. Please pass your input's `attention_mask` to obtain reliable results.
The future of AI is vernacularly called 'the next chapter' or 'the next stage
```

BertGeneration model (BERT based model)

In [5]:

```
sentence_fuser = EncoderDecoderModel.from_pretrained("google/roberta2roberta_L-24_discofuse")
tokenizer = AutoTokenizer.from_pretrained("google/roberta2roberta_L-24_discofuse")
```

```
Config of the encoder: <class 'transformers.models.bert_generation.modeling_bert_generation.BertGenerationEncoder'> is
overwritten by shared encoder config: BertGenerationConfig {
  "architectures": [
    "BertGenerationDecoder"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": null,
  "directionality": "bidi",
  "eos_token_id": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert-generation",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "return_dict": false,
  "torch_dtype": "float32",
  "transformers_version": "4.49.0",
  "use_cache": true,
  "vocab_size": 50358
}

Config of the decoder: <class 'transformers.models.bert_generation.modeling_bert_generation.BertGenerationDecoder'> is
overwritten by shared decoder config: BertGenerationConfig {
  "add_cross_attention": true,
  "architectures": [
    "BertGenerationEncoder"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": null,
  "directionality": "bidi",
  "eos_token_id": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "is_decoder": true,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert-generation",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "return_dict": false,
  "torch_dtype": "float32",
  "transformers_version": "4.49.0",
  "use_cache": true,
  "vocab_size": 50358
}

Config of the decoder: <class 'transformers.models.bert_generation.modeling_bert_generation.BertGenerationDecoder'> is
overwritten by shared decoder config: BertGenerationConfig {
  "add_cross_attention": true,
  "architectures": [
    "BertGenerationEncoder"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": null,
  "directionality": "bidi",
  "eos_token_id": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "is_decoder": true,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert-generation",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "return_dict": false,
  "torch_dtype": "float32",
  "transformers_version": "4.49.0",
  "use_cache": true,
  "vocab_size": 50358
}
```

In [6]:

```
input_ids = tokenizer(
    prompt, add_special_tokens=False, return_tensors="pt"
).input_ids
```

```
outputs = sentence_fuser.generate(input_ids)
print(tokenizer.decode(outputs[0]))
```

```
<s>The future of AI is the future.</s>
```

Visualize Attention Weights using `bertviz`

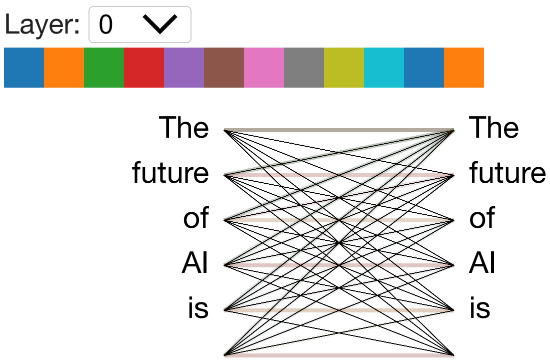
GPT2

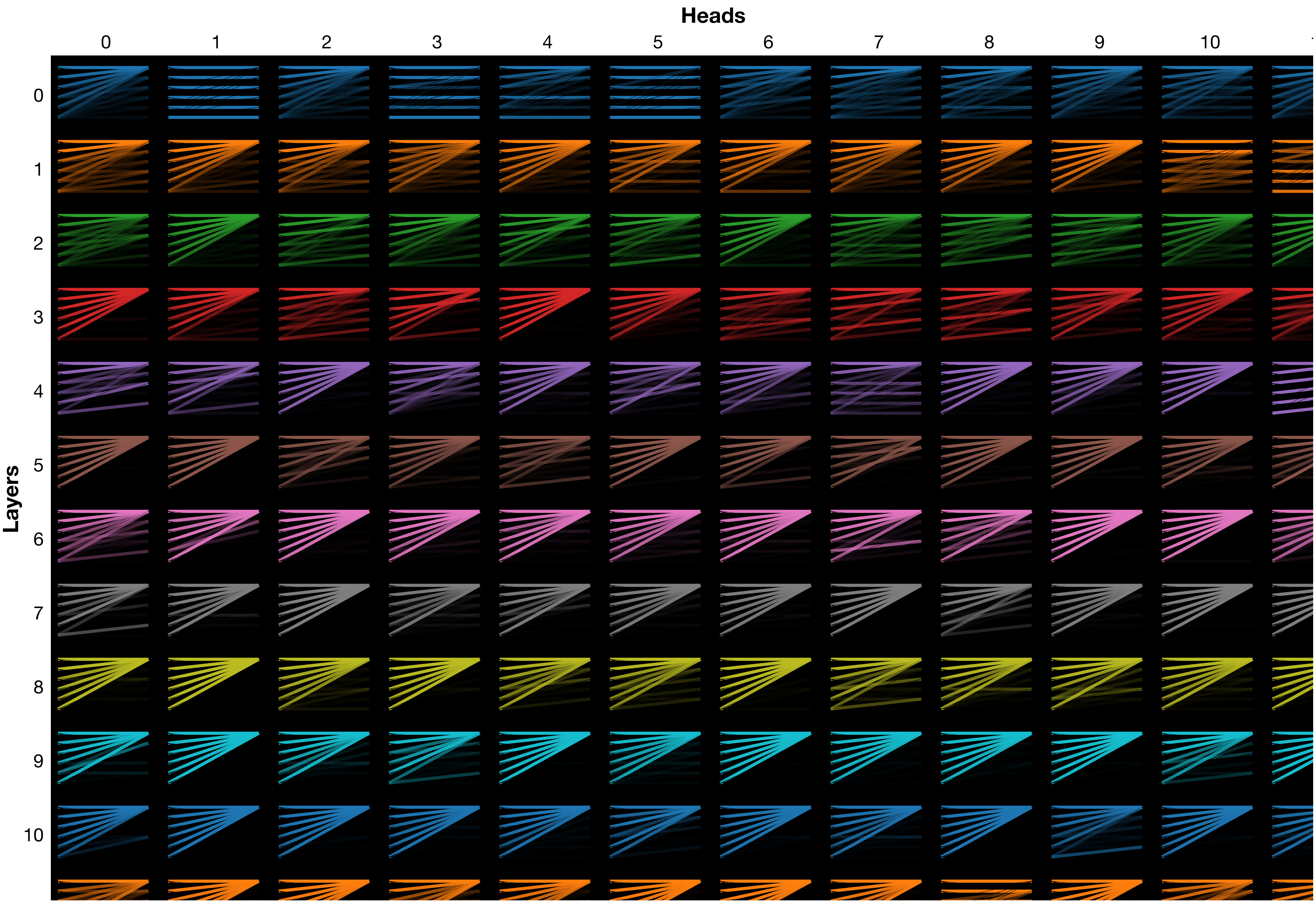
```
In [7]: model = AutoModel.from_pretrained("gpt2", output_attentions=True)
        tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

```
In [8]: # Run the model to get the attention weights (not using generate here)
        inputs = tokenizer.encode(prompt, return_tensors='pt')
        outputs = model(inputs)
        attention = outputs[-1]
```

```
`torch.nn.functional.scaled_dot_product_attention` does not support `output_attentions=True`. Falling back to eager att
ention. This warning can be removed using the argument `attn_implementation="eager"` when loading the model.
```

```
In [9]: tokens = tokenizer.convert_ids_to_tokens(inputs[0])
        head_view(attention, tokens)
        model_view(attention, tokens)
```





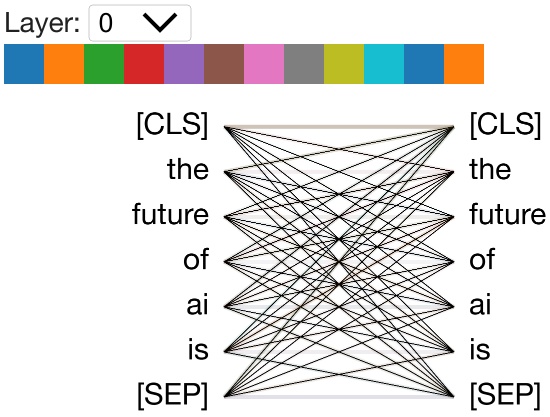
BERT

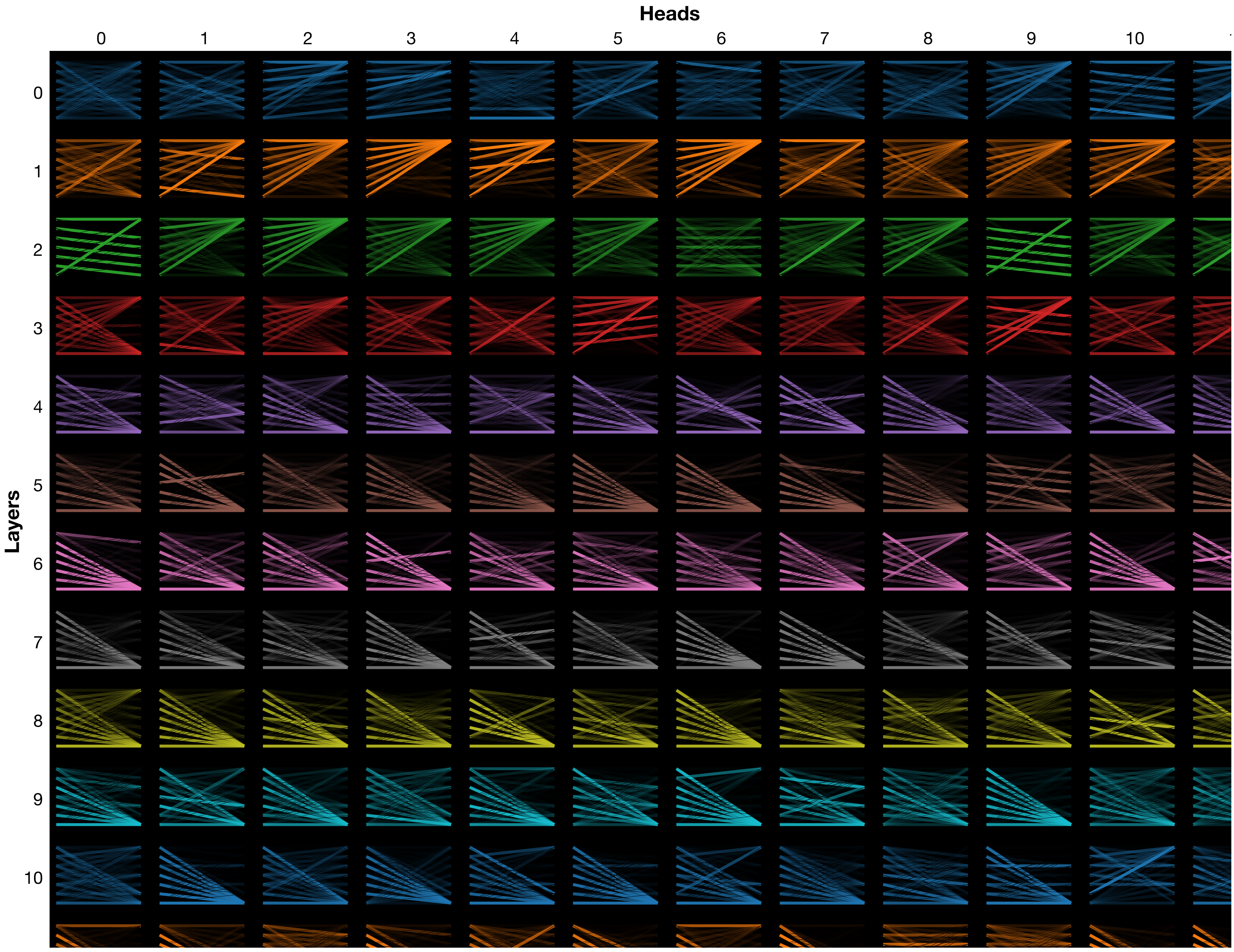
```
In [10]: tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
         model = AutoModel.from_pretrained("bert-base-uncased", output_attentions=True)
```

```
In [11]: inputs = tokenizer.encode(prompt, return_tensors='pt')
         outputs = model(inputs)
         attention = outputs[-1]
         tokens = tokenizer.convert_ids_to_tokens(inputs[0])
```

```
BertSdpaSelfAttention is used but `torch.nn.functional.scaled_dot_product_attention` does not support non-absolute `pos
ition_embedding_type` or `output_attentions=True` or `head_mask`. Falling back to the manual attention implementation,
but specifying the manual implementation will be required from Transformers version v5.0.0 onwards. This warning can be
removed using the argument `attn_implementation="eager"` when loading the model.
```

```
In [12]: head_view(attention, tokens)
         model_view(attention, tokens)
```





Fine-Tune Models

In [4]:

```
dataset = load_dataset("yelp_review_full")
dataset["train"][100]

def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True)

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

GPT2

In [5]:

```
tokenizer = AutoTokenizer.from_pretrained("gpt2")

if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token': '[PAD]'})
```

In [6]:

```
tokenized_datasets = dataset.map(tokenize_function, batched=True)
small_train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(10))
small_eval_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(10))
```

In [7]:

```
model = AutoModelForSequenceClassification.from_pretrained("gpt2", num_labels=5, torch_dtype="auto")
training_args = TrainingArguments(output_dir="test_trainer")
```

```
Some weights of GPT2ForSequenceClassification were not initialized from the model checkpoint at gpt2 and are newly init
ialized: ['score.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

In [8]:

```
metric = evaluate.load("accuracy")
```

In [9]:

```
training_args = TrainingArguments(output_dir="test_trainer", eval_strategy="epoch", num_train_epochs=2)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,
)
trainer.train()
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[9], line 9
      1 training_args = TrainingArguments(output_dir="test_trainer", eval_strategy="epoch", num_train_epochs=2)
      2 trainer = Trainer(
      3     model=model,
      4     args=training_args,
   (...)
      7     compute_metrics=compute_metrics,
      8 )
----> 9 trainer.train()

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/trainer.py:2241, in Trainer.tr
ain(self, resume_from_checkpoint, trial, ignore_keys_for_eval, **kwargs)
   2239         hf_hub_utils.enable_progress_bars()
   2240 else:
-> 2241     return inner_training_loop(
   2242         args=args,
   2243         resume_from_checkpoint=resume_from_checkpoint,
   2244         trial=trial,
   2245         ignore_keys_for_eval=ignore_keys_for_eval,
   2246     )

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/trainer.py:2548, in Trainer._i
nner_training_loop(self, batch_size, args, resume_from_checkpoint, trial, ignore_keys_for_eval)
   2541 context = (
   2542     functools.partial(self.accelerator.no_sync, model=model)
   2543     if i != len(batch_samples) - 1
   2544     and self.accelerator.distributed_type != DistributedType.DEEPSPEED
   2545     else contextlib.nullcontext
   2546 )
   2547 with context():
-> 2548     tr_loss_step = self.training_step(model, inputs, num_items_in_batch)
   2550 if (
   2551     args.logging_nan_inf_filter
   2552     and not is_torch_xla_available()
   2553     and (torch.isnan(tr_loss_step) or torch.isinf(tr_loss_step))
   2554 ):
   2555     # if loss is nan or inf simply add the average of previous logged losses
   2556     tr_loss = tr_loss + tr_loss / (1 + self.state.global_step - self._globalstep_last_logged)

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/trainer.py:3698, in Trainer.tr
aining_step(self, model, inputs, num_items_in_batch)
   3695     return loss_mb.reduce_mean().detach().to(self.args.device)
   3697 with self.compute_loss_context_manager():
-> 3698     loss = self.compute_loss(model, inputs, num_items_in_batch=num_items_in_batch)
   3700 del inputs
   3701 if (
   3702     self.args.torch_empty_cache_steps is not None
   3703     and self.state.global_step % self.args.torch_empty_cache_steps == 0
   3704 ):

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/trainer.py:3759, in Trainer.co
mpute_loss(self, model, inputs, return_outputs, num_items_in_batch)
   3757         loss_kwargs["num_items_in_batch"] = num_items_in_batch
   3758     inputs = {**inputs, **loss_kwargs}
-> 3759 outputs = model(**inputs)
   3760 # Save past state if it exists
   3761 # TODO: this needs to be fixed and made cleaner later.
   3762 if self.args.past_index >= 0:

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1739, in Module.
_wrapped_call_impl(self, *args, **kwargs)
   1737     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1738 else:
-> 1739     return self._call_impl(*args, **kwargs)

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1750, in Module.
_call_impl(self, *args, **kwargs)
   1745 # If we don't have any hooks, we want to skip the rest of the logic in
   1746 # this function, and just call forward.
   1747 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1748         or _global_backward_pre_hooks or _global_backward_hooks
   1749         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1750     return forward_call(*args, **kwargs)
   1752 result = None
   1753 called_always_called_hooks = set()

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/models/gpt2/modeling_gpt2.py:1
375, in GPT2ForSequenceClassification.forward(self, input_ids, past_key_values, attention_mask, token_type_ids, positio
n_ids, head_mask, inputs_embeds, labels, use_cache, output_attentions, output_hidden_states, return_dict)
   1367 r"""
   1368 labels (`torch.LongTensor` of shape `(batch_size,)`, *optional*):
   1369     Labels for computing the sequence classification/regression loss. Indices should be in `[0, ...,
   1370     config.num_labels - 1]`. If `config.num_labels == 1` a regression loss is computed (Mean-Square loss), If
   1371     `config.num_labels > 1` a classification loss is computed (Cross-Entropy).
   1372 """
   1373 return_dict = return_dict if return_dict is not None else self.config.use_return_dict
-> 1375 transformer_outputs = self.transformer(
   1376     input_ids,
   1377     past_key_values=past_key_values,
   1378     attention_mask=attention_mask,
   1379     token_type_ids=token_type_ids,
   1380     position_ids=position_ids,
   1381     head_mask=head_mask,
   1382     inputs_embeds=inputs_embeds,
   1383     use_cache=use_cache,
   1384     output_attentions=output_attentions,
   1385     output_hidden_states=output_hidden_states,
   1386     return_dict=return_dict,
   1387 )
   1388 hidden_states = transformer_outputs[0]
   1389 logits = self.score(hidden_states)

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1739, in Module._wrapped_call_impl(self, *args, **kwargs)
   1737     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1738 else:
-> 1739     return self._call_impl(*args, **kwargs)

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1750, in Module._call_impl(self, *args, **kwargs)
   1745 # If we don't have any hooks, we want to skip the rest of the logic in
   1746 # this function, and just call forward.
   1747 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1748         or _global_backward_pre_hooks or _global_backward_hooks
   1749         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1750     return forward_call(*args, **kwargs)
   1752 result = None
   1753 called_always_called_hooks = set()

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/models/gpt2/modeling_gpt2.py:922, in GPT2Model.forward(self, input_ids, past_key_values, attention_mask, token_type_ids, position_ids, head_mask, inputs_embeds, encoder_hidden_states, encoder_attention_mask, use_cache, output_attentions, output_hidden_states, return_dict)
    910     outputs = self._gradient_checkpointing_func(
    911         block.__call__,
    912         hidden_states,
   (...)
    919         output_attentions,
    920     )
    921 else:
--> 922     outputs = block(
    923         hidden_states,
    924         layer_past=layer_past,
    925         attention_mask=attention_mask,
    926         head_mask=head_mask[i],
    927         encoder_hidden_states=encoder_hidden_states,
    928         encoder_attention_mask=encoder_attention_mask,
    929         use_cache=use_cache,
    930         output_attentions=output_attentions,
    931     )
    933 hidden_states = outputs[0]
    934 if use_cache is True:

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1739, in Module._wrapped_call_impl(self, *args, **kwargs)
   1737     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1738 else:
-> 1739     return self._call_impl(*args, **kwargs)

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1750, in Module._call_impl(self, *args, **kwargs)
   1745 # If we don't have any hooks, we want to skip the rest of the logic in
   1746 # this function, and just call forward.
   1747 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1748         or _global_backward_pre_hooks or _global_backward_hooks
   1749         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1750     return forward_call(*args, **kwargs)
   1752 result = None
   1753 called_always_called_hooks = set()

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/models/gpt2/modeling_gpt2.py:441, in GPT2Block.forward(self, hidden_states, layer_past, attention_mask, head_mask, encoder_hidden_states, encoder_attention_mask, use_cache, output_attentions)
    439 residual = hidden_states
    440 hidden_states = self.ln_2(hidden_states)
--> 441 feed_forward_hidden_states = self.mlp(hidden_states)
    442 # residual connection
    443 hidden_states = residual + feed_forward_hidden_states

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1739, in Module._wrapped_call_impl(self, *args, **kwargs)
   1737     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1738 else:
-> 1739     return self._call_impl(*args, **kwargs)

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1750, in Module._call_impl(self, *args, **kwargs)
   1745 # If we don't have any hooks, we want to skip the rest of the logic in
   1746 # this function, and just call forward.
   1747 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1748         or _global_backward_pre_hooks or _global_backward_hooks
   1749         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1750     return forward_call(*args, **kwargs)
   1752 result = None
   1753 called_always_called_hooks = set()

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/models/gpt2/modeling_gpt2.py:369, in GPT2MLP.forward(self, hidden_states)
    367 def forward(self, hidden_states: Optional[Tuple[torch.FloatTensor]]) -> torch.FloatTensor:
```

```
    368     hidden_states = self.c_fc(hidden_states)
--> 369     hidden_states = self.act(hidden_states)
    370     hidden_states = self.c_proj(hidden_states)
    371     hidden_states = self.dropout(hidden_states)

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1739, in Module.
_wrapped_call_impl(self, *args, **kwargs)
   1737     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1738 else:
-> 1739     return self._call_impl(*args, **kwargs)

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/torch/nn/modules/module.py:1750, in Module.
_call_impl(self, *args, **kwargs)
   1745 # If we don't have any hooks, we want to skip the rest of the logic in
   1746 # this function, and just call forward.
   1747 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1748         or _global_backward_pre_hooks or _global_backward_hooks
   1749         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1750     return forward_call(*args, **kwargs)
   1752 result = None
   1753 called_always_called_hooks = set()

File ~/anaconda3/envs/applied-machine-learning/lib/python3.12/site-packages/transformers/activations.py:56, in NewGELUA
ctivation.forward(self, input)
     55 def forward(self, input: Tensor) -> Tensor:
---> 56     return 0.5 * input * (1.0 + torch.tanh(math.sqrt(2.0 / math.pi) * (input + 0.044715 * torch.pow(input, 3.
0))))

RuntimeError: MPS backend out of memory (MPS allocated: 17.72 GB, other allocations: 408.69 MB, max allowed: 18.13 GB).
Tried to allocate 96.00 MB on private pool. Use PYTORCH_MPS_HIGH_WATERMARK_RATIO=0.0 to disable upper limit for memory
allocations (may cause system failure).
```

BERT

In [10]:

```
tokenizer = AutoTokenizer.from_pretrained("google-bert/bert-base-cased")
```

In [11]:

```
tokenized_datasets = dataset.map(tokenize_function, batched=True)
small_train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(10))
small_eval_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(10))
```

In [12]:

```
model = AutoModelForSequenceClassification.from_pretrained("google-bert/bert-base-cased", num_labels=5, torch_dtype="au
training_args = TrainingArguments(output_dir="test_trainer")
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at google-bert/bert-base-c
ased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

In [14]:

```
metric = evaluate.load("accuracy")
```

```
Using the latest cached version of the module from /Users/fernport/.cache/huggingface/modules/evaluate_modules/metrics/
evaluate-metric--accuracy/f887c0aab52c2d38e1f8a215681126379eca617f96c447638f751434e8e65b14 (last modified on Tue Mar 18
00:35:47 2025) since it couldn't be found locally at evaluate-metric--accuracy, or remotely on the Hugging Face Hub.
```

In [15]:

```
training_args = TrainingArguments(output_dir="test_trainer", eval_strategy="epoch", num_train_epochs=2)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,
)
trainer.train()
```

[4/4 07:41, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	1.539728	0.300000
2	No log	1.520594	0.300000

Out[15]:

```
TrainOutput(global_step=4, training_loss=1.6320981979370117, metrics={'train_runtime': 541.1323, 'train_samples_per_se
cond': 0.037, 'train_steps_per_second': 0.007, 'total_flos': 5262362849280.0, 'train_loss': 1.6320981979370117, 'epoc
h': 2.0})
```

Conclusion

GPT-2 performed better at text completion compared to the BERT model.

GPT-2 and BERT are built for different tasks:

- GPT-2 is awesome at generating text. It predicts the next word in a sequence, so it's great for writing or continuing prompts.
- BERT (or the pre-trained model based on BERT) is better at understanding context. It's perfect for tasks like classification and question answering, but not really for generating text.

Based on the training aspect it was not possible to fine-tune GPT2 with my computer. It was only possible to train BERT up to 10 eposch and 50 entries of data. BERT seemed to perform well up to 7 then it proceeded to degrade. Overall with the low testing samples and low epoch the highest level of accurary was 30.