b-schoen Saving progress f1a6293 · Sep 15, 2024

1624 lines (1624 loc) · 50.4 KB

Preview Code Blame Raw

In [1]:
```
# some ipython magic to automatically reload any imports if they change
# (useful when iterating locally)

from IPython import get_ipython

# do this so that formatter not messed up
ipython = get_ipython()
ipython.run_line_magic("load_ext", "autoreload")
ipython.run_line_magic("autoreload", "2")
```

Load Model

In [2]:
```
import transformer_lens
import tqdm
import wandb

# load the device we'll use (GPU or MPS)
device = transformer_lens.utils.get_device()

print(f"Using device: {device}")
```

```
/home/ubuntu/minTranscoder/venv/lib/python3.10/site-packages/tqdm/auto.py:2
1: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets.
See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Using device: cuda
```

In [3]:
```
# load our model
model_name = "gpt2-small"
model = transformer_lens.HookedTransformer.from_pretrained(
    model_name,
    center_unembed=True,
    center_writing_weights=True,
    fold_ln=True,
    refactor_factored_attn_matrices=True,
    device=device,
)
```

```
/home/ubuntu/minTranscoder/venv/lib/python3.10/site-packages/transformers/t
okenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_space
s` was not set. It will be set to `True` by default. This behavior will be
depracted in transformers v4.45, and will be then set to `False` by defaul
t. For more details check this issue: https://github.com/huggingface/transf
ormers/issues/31884
  warnings.warn(
Loaded pretrained model gpt2-small into HookedTransformer
```

In [4]:
```
# sanity check with an example
example_prompt = "After John and Mary went to the store, John gave a bott
example_answer = " Mary"
transformer_lens.utils.test_prompt(
    example_prompt,
    example_answer,
```

```
    model,
    prepend_bos=True,
)
```

```
Tokenized prompt: ['<|endoftext|>', 'After', ' John', ' and', ' Mary', ' we
nt', ' to', ' the', ' store', ',', ' John', ' gave', ' a', ' bottle', ' o
f', ' milk', ' to']
Tokenized answer: [' Mary']
Performance on answer token:
Rank: 0        Logit: 18.09 Prob: 70.07% Token: | Mary|
Top 0th token. Logit: 18.09 Prob: 70.07% Token: | Mary|
Top 1th token. Logit: 15.38 Prob:  4.67% Token: | the|
Top 2th token. Logit: 15.35 Prob:  4.54% Token: | John|
Top 3th token. Logit: 15.25 Prob:  4.11% Token: | them|
Top 4th token. Logit: 14.84 Prob:  2.73% Token: | his|
Top 5th token. Logit: 14.06 Prob:  1.24% Token: | her|
Top 6th token. Logit: 13.54 Prob:  0.74% Token: | a|
Top 7th token. Logit: 13.52 Prob:  0.73% Token: | their|
Top 8th token. Logit: 13.13 Prob:  0.49% Token: | Jesus|
Top 9th token. Logit: 12.97 Prob:  0.42% Token: | him|
Ranks of the answer tokens: [(' Mary', 0)]
```

Define Transcoder Config

In [5]:

```python
import dataclasses
import torch
from jaxtyping import Float

from min_transcoder.transcoder import (
    TranscoderResults,
    TranscoderConfig,
    Transcoder,
)


@dataclasses.dataclass
class TranscoderTrainingConfig:

    # Name of the layer to hook into for feature extraction
    hook_point: str
    out_hook_point: str

    num_epochs: int = 100

    # both from https://arxiv.org/html/2406.11944v1#S3 appendix E
    learning_rate: float = 2 * 10e-5
    l1_coefficient: float = 1e-4

    @property
    def hook_point_layer(self) -> int:
        "Parse out the hook point layer as int ex: 'blocks.8.ln2.hook_norm
        return int(self.hook_point.split(".")[1])


@dataclasses.dataclass
class TranscoderLoss:
    mse_loss: Float[torch.Tensor, ""]
```

```
    mse_loss: Float[torch.Tensor, ""]
    l1_loss: Float[torch.Tensor, ""]


def compute_loss(
    cfg: TranscoderTrainingConfig,
    mlp_out: Float[torch.Tensor, "..."],
    results: TranscoderResults,
) -> TranscoderLoss:

    mse_loss_per_batch: Float[torch.Tensor, "..."] = (
        torch.pow((results.transcoder_out - mlp_out.float()), 2)
        / (mlp_out**2).sum(dim=-1, keepdim=True).sqrt()
    )

    mse_loss = mse_loss_per_batch.mean()

    sparsity = torch.abs(results.hidden_activations).sum(dim=1).mean(dim=

    # TODO(bschoen): Do we sum here?
    l1_loss = cfg.l1_coefficient * sparsity.mean()

    return TranscoderLoss(mse_loss=mse_loss, l1_loss=l1_loss)


# from https://arxiv.org/html/2406.11944v1#S3 appendix E
transcoder_expansion_factor = 32

transcoder_cfg = TranscoderConfig(
    d_in=model.cfg.d_model,
    d_out=model.cfg.d_model,
    # our transcoder has a hidden dimension of d_mlp * expansion factor
    d_hidden=model.cfg.d_mlp * transcoder_expansion_factor,
    dtype=model.cfg.dtype,
    device=device,
)
```

In [6]:

```
print(f"{model.cfg.n_layers=}")
print(f"{model.cfg.d_mlp=}")
```

```
model.cfg.n_layers=12
model.cfg.d_mlp=3072
```

Load Data

In [7]:

```
import datasets
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader


def create_tokenized_dataloader(
    max_length: int = 128,
    batch_size: int = 64,
    num_samples: int = 10000,
    shuffle: bool = True,
) -> DataLoader:
```

```
    print("Loading dataset...")
    dataset = datasets.load_dataset(
        path="NeelNanda/pile-10k",
        split="train",
        streaming=False,
    )

    print("Tokenizing dataset...")
    token_dataset = transformer_lens.utils.tokenize_and_concatenate(
        dataset=dataset,
        tokenizer=model.tokenizer,
        streaming=True,
        max_length=max_length,
        add_bos_token=model.cfg.default_prepend_bos,
    )

    # token_dataset['tokens'].shape=torch.Size([136625, 128])
    # print(f"{token_dataset['tokens'].shape=}")

    # shuffle, and arbitrarily cap at around 10,000 / 130,000 (original ca
    token_dataset = token_dataset.shuffle(42)
    token_dataset = token_dataset.take(num_samples)

    token_dataset_torch = torch.from_numpy(
        np.stack([x["tokens"] for x in token_dataset])
    ).cuda()

    # torch.Size([100, 1024])
    print(token_dataset_torch.shape)

    # Create a DataLoader for batching
    #
    # for batch in dataloader:
    #     print(batch.shape) # torch.Size([32, 1024])
    #     break
    #
    print(f"Creating dataloader for dataset...")
    dataloader = torch.utils.data.DataLoader(
        token_dataset_torch,
        batch_size=batch_size,
        shuffle=shuffle,
    )

    print(f"Num batches: {token_dataset_torch.shape[0] / batch_size}")

    return dataloader
```

Collect Activations

Here we'll create hooks to store the MLP activations only

In [8]:

```
# Define training parameters
layer = 11

training_cfg = TranscoderTrainingConfig(
```

```
    num_epochs=5,
    hook_point=f"blocks.{layer}.ln2.hook_normalized",
    out_hook_point=f"blocks.{layer}.hook_mlp_out",
)
```

In [9]:

```
# store the MLP activations
mlp_inputs: list[Float[torch.Tensor, "batch seq d_mlp_in"]] = []
mlp_outputs: list[Float[torch.Tensor, "batch seq d_mlp_out"]] = []


# TODO(bschoen): Could make this general
def store_mlp_inputs(
    mlp_input: Float[torch.Tensor, "... d_in"],
    hook: transformer_lens.hook_points.HookPoint,
) -> None:

    # Detach and move to CPU to save memory
    mlp_inputs.append(mlp_input.detach().cpu())


def store_mlp_output(
    mlp_output: Float[torch.Tensor, "... d_out"],
    hook: transformer_lens.hook_points.HookPoint,
) -> None:

    # Detach and move to CPU to save memory
    mlp_outputs.append(mlp_output.detach().cpu())
```

In [10]:

```
dataloader = create_tokenized_dataloader()

# put model itself into eval mode so doesn't change
model.eval()

for batch_index, batch in tqdm.tqdm(
    enumerate(dataloader),
    desc="Collecting MLP activations",
):
    # move batch to device
    batch = batch.to(device)

    # Get MLP input and output activations
    model.run_with_hooks(
        batch,
        fwd_hooks=[
            (training_cfg.hook_point, store_mlp_inputs),
            (training_cfg.out_hook_point, store_mlp_output),
        ],
        return_type=None,
    )
```

```
Loading dataset...
Tokenizing dataset...
torch.Size([10000, 128])
Creating dataloader for dataset...
Num batches: 156.25
Collecting MLP activations: 157it [00:24,  6.42it/s]
```

In [11]:

In [11]:
```
# now we can unload gpu
torch.cuda.empty_cache()
```

In [12]:
```
print(f"{len(mlp_inputs)=}, {mlp_inputs[0].shape=}")
print(f"{len(mlp_outputs)=}, {mlp_outputs[0].shape=}")
```

```
len(mlp_inputs)=157, mlp_inputs[0].shape=torch.Size([64, 128, 768])
len(mlp_outputs)=157, mlp_outputs[0].shape=torch.Size([64, 128, 768])
```

In [13]:
```
# Custom Dataset
class MLPActivationsDataset(Dataset):
    def __init__(
        self,
        mlp_inputs: list[Float[torch.Tensor, "batch seq d_mlp_in"]],
        mlp_outputs: list[Float[torch.Tensor, "batch seq d_mlp_out"]],
    ) -> None:
        self.mlp_inputs = mlp_inputs
        self.mlp_outputs = mlp_outputs
        assert len(self.mlp_inputs) == len(
            self.mlp_outputs
        ), "Inputs and outputs must be the same length."

    def __len__(self) -> int:
        return len(self.mlp_inputs)

    def __getitem__(self, idx: int) -> tuple[
        Float[torch.Tensor, "batch seq d_mlp_in"],
        Float[torch.Tensor, "batch seq d_mlp_out"],
    ]:
        x = self.mlp_inputs[idx]  # Shape: [128, 128, 768]
        y = self.mlp_outputs[idx]  # Shape: [128, 128, 768]
        return x, y


# Create Dataset and DataLoader
activations_dataset = MLPActivationsDataset(mlp_inputs, mlp_outputs)
activations_dataloader = DataLoader(
    activations_dataset,
    shuffle=True,
)
```

Train Transcoder

In [14]:
```
# Initialize wandb
wandb.init(
    project="transcoder_training_v2",
    config=dataclasses.asdict(training_cfg),
)

transcoder = Transcoder(cfg=transcoder_cfg)

transcoder = transcoder.to(device)

# Initialize optimizer
optimizer = torch.optim.AdamW(transcoder.parameters(), lr=training_cfg.lea
```

```python
num_steps = 0

# Training loop
for epoch in range(training_cfg.num_epochs):

    for batch_index, batch in tqdm.tqdm(
        enumerate(activations_dataloader),
        desc=f"Epoch {epoch+1}/{training_cfg.num_epochs}",
    ):

        # Do a training step.
        transcoder.train()

        # Make sure the W_dec is still zero-norm
        transcoder.set_decoder_norm_to_unit_norm()

        optimizer.zero_grad()

        # move batch to device
        batch_x, batch_y = batch

        mlp_in = batch_x[0].to(device)
        mlp_out = batch_y[0].to(device)

        transcoder_results = transcoder(mlp_in)

        # Compute loss
        loss_result = compute_loss(training_cfg, mlp_out, transcoder_resu

        loss = loss_result.mse_loss + loss_result.l1_loss

        # Backward pass and optimization
        loss.backward()

        optimizer.step()

        num_steps += 1

        # Print loss statistics every 10 batches
        if batch_index % 10 == 0:
            print(
                f"Epoch {epoch+1}/{training_cfg.num_epochs}, "
                f"Batch {batch_index}/{len(activations_dataloader)}, "
                f"Loss: {loss.item():.6f}, "
                f"MSE Loss: {loss_result.mse_loss.item():.6f}, "
                f"L1 Loss: {loss_result.l1_loss.item():.6f}"
            )

            # Log metrics to wandb
            wandb.log(
                {
                    "epoch": epoch + 1,
                    "loss": loss.item(),
                    "mse_loss": loss_result.mse_loss.item(),
                    "l1_loss": loss_result.l1_loss.item(),
                },
                step=num_steps,
            )
```

```
    # Log model parameters and gradients
    # wandb.watch(transcoder)

print("Training completed!")

# Finish the wandb run
wandb.finish()
```

```
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: bronsonschoen (bronsonschoen-personal-use). Use `wandb login --relogin` to force relogin
```

Tracking run with wandb version 0.18.0

Run data is saved locally in /home/ubuntu/minTranscoder/wandb/run-20240915_161952-8bipu47h

Syncing run **zany-puddle-8** to Weights & Biases (docs)

View project at https://wandb.ai/bronsonschoen-personal-use/transcoder_training_v2

View run at https://wandb.ai/bronsonschoen-personal-use/transcoder_training_v2/runs/8bipu47h

```
Epoch 1/1: 1it [00:00,  1.03it/s]
Epoch 1/1, Batch 0/79, Loss: 0.112818, MSE Loss: 0.112178, L1 Loss: 0.000640
Epoch 1/1: 11it [00:10,  1.17s/it]
Epoch 1/1, Batch 10/79, Loss: 0.038793, MSE Loss: 0.038266, L1 Loss: 0.000527
Epoch 1/1: 21it [00:19,  1.17s/it]
Epoch 1/1, Batch 20/79, Loss: 0.028430, MSE Loss: 0.027941, L1 Loss: 0.000489
Epoch 1/1: 31it [00:27,  1.12s/it]
Epoch 1/1, Batch 30/79, Loss: 0.025092, MSE Loss: 0.024633, L1 Loss: 0.000460
Epoch 1/1: 41it [00:36,  1.17s/it]
Epoch 1/1, Batch 40/79, Loss: 0.023120, MSE Loss: 0.022663, L1 Loss: 0.000457
Epoch 1/1: 51it [00:45,  1.17s/it]
Epoch 1/1, Batch 50/79, Loss: 0.021735, MSE Loss: 0.021267, L1 Loss: 0.000469
Epoch 1/1: 61it [00:54,  1.17s/it]
Epoch 1/1, Batch 60/79, Loss: 0.021578, MSE Loss: 0.021100, L1 Loss: 0.000478
Epoch 1/1: 71it [01:03,  1.17s/it]
Epoch 1/1, Batch 70/79, Loss: 0.019396, MSE Loss: 0.018922, L1 Loss: 0.000474
Epoch 1/1: 79it [01:10,  1.12it/s]
Training completed!
```

Run history:

mse_loss ▁▁▁▁▁▁▁▁

Run summary:

epoch	1
l1_loss	0.00047
loss	0.0194
mse_loss	0.01892

View run **zany-puddle-8** at: https://wandb.ai/bronsonschoen-personal-use/transcoder_training_v2/runs/8bipu47h
View project at: https://wandb.ai/bronsonschoen-personal-use/transcoder_training_v2
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `./wandb/run-20240915_161952-8bipu47h/logs`

In [14]:
```python
# Save the trained transcoder model to a file
import torch
import pathlib

# Define the path where you want to save the model
model_save_path = f"full_transcoder_model_{training_cfg.hook_point}.pth"

print(f"Transcoder model saved to {model_save_path}")

if not pathlib.Path(model_save_path).exists():
    torch.save(transcoder, model_save_path)
```

```
Transcoder model saved to full_transcoder_model_blocks.11.ln2.hook_normalized.pth
```

In [15]:
```python
# Load the full transcoder model
loaded_transcoder = torch.load(model_save_path)

loaded_transcoder.to(device)

print("Loaded transcoder")

# Set the loaded model to evaluation mode
loaded_transcoder.eval()

print(loaded_transcoder)  # Print the loaded model architecture

# Optionally, you can verify the model's parameters
for name, param in loaded_transcoder.named_parameters():
    print(f"Parameter: {name}, Shape: {param.shape}")
```

```
/tmp/ipykernel_45132/3283311484.py:2: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the
```

```
default pickle module implicitly. It is possible to construct malicious pic
kle data which will execute arbitrary code during unpickling (See https://g
ithub.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more d
etails). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via th
is mode unless they are explicitly allowlisted by the user via `torch.seria
lization.add_safe_globals`. We recommend you start setting `weights_only=Tr
ue` for any use case where you don't have full control of the loaded file.
Please open an issue on GitHub for any issues related to this experimental
feature.
  loaded_transcoder = torch.load(model_save_path)
Loaded transcoder
Transcoder()
Parameter: W_enc, Shape: torch.Size([768, 98304])
Parameter: b_enc, Shape: torch.Size([98304])
Parameter: W_dec, Shape: torch.Size([98304, 768])
Parameter: b_dec, Shape: torch.Size([768])
Parameter: b_dec_out, Shape: torch.Size([768])
```

Compute Loss When Substituting MLP with Transcoder

In [16]:

```python
class _TranscoderWrapper(torch.nn.Module):
    def __init__(self, transcoder: Transcoder):
        super().__init__()
        self.transcoder = transcoder

    def forward(
        self, x: Float[torch.Tensor, "... d_in"]
    ) -> Float[torch.Tensor, "... d_out"]:
        transcoder_result = self.transcoder(x)
        return transcoder_result.transcoder_out


@torch.no_grad()
def get_test_loss_when_replacing_mlp_with_transcoder(
    batch_tokens: Float[torch.Tensor, "batch seq"],
    transcoder: Transcoder,
    model: transformer_lens.HookedTransformer,
    hook_point_layer: str,
) -> Float[torch.Tensor, ""]:
    """
    A method for running the model with the SAE activations in order to re
    loss returns per token loss when activations are substituted in.

    """
    old_mlp = model.blocks[hook_point_layer]

    model.blocks[hook_point_layer].mlp = _TranscoderWrapper(transcoder)

    ce_loss_with_recons = model.run_with_hooks(batch_tokens, return_type="

    model.blocks[hook_point_layer] = old_mlp

    model.reset_hooks()
```

```
    return ce_loss_with_recons
```

In [ ]:

```
```

In [17]:

```
# compute how much worse this makes the loss
#
# note: normally compare to ablated
#
transcoder = loaded_transcoder

transcoder.eval()

num_batches = 10

dataloader = create_tokenized_dataloader(num_samples=num_batches)

avg_loss_original = 0
avg_loss_when_replaced_mlp = 0

for batch_index, batch in enumerate(dataloader):

    if batch_index > num_batches:
        break

    batch = batch.to(device)

    loss_original = model.run_with_hooks(batch, return_type="loss")

    loss_when_replaced_mlp = get_test_loss_when_replacing_mlp_with_transc
        batch_tokens=batch,
        transcoder=transcoder,
        model=model,
        hook_point_layer=training_cfg.hook_point_layer,
    )

    avg_loss_original += loss_original.item()
    avg_loss_when_replaced_mlp += loss_when_replaced_mlp.item()

avg_loss_original /= num_batches
avg_loss_when_replaced_mlp /= num_batches

print(f"{avg_loss_original=}")
print(f"{avg_loss_when_replaced_mlp=}")
```

```
Loading dataset...
Tokenizing dataset...
torch.Size([10, 128])
Creating dataloader for dataset...
Num batches: 0.15625
avg_loss_original=0.3654099225997925
avg_loss_when_replaced_mlp=0.37158894538879395
```

Sanity Check - Indirect Object Identification

We quickly check that IOI isn't impacted (it shouldn't be, since we know it doesn't

depend much on MLP, but it's good to check against a known result).

In [18]:
```
import transformer_lens

# sanity check with an example
example_prompt = "After John and Mary went to the store, John gave a bott
example_answer = " Mary"
transformer_lens.utils.test_prompt(
    example_prompt,
    example_answer,
    model,
    prepend_bos=True,
)
```

```
Tokenized prompt: ['<|endoftext|>', 'After', ' John', ' and', ' Mary', ' we
nt', ' to', ' the', ' store', ',', ' John', ' gave', ' a', ' bottle', ' o
f', ' milk', ' to']
Tokenized answer: [' Mary']
Performance on answer token:
Rank: 0        Logit: 18.39 Prob: 75.11% Token: | Mary|
Top 0th token. Logit: 18.39 Prob: 75.11% Token: | Mary|
Top 1th token. Logit: 15.74 Prob:  5.33% Token: | John|
Top 2th token. Logit: 15.69 Prob:  5.03% Token: | the|
Top 3th token. Logit: 14.92 Prob:  2.35% Token: | them|
Top 4th token. Logit: 14.24 Prob:  1.19% Token: | his|
Top 5th token. Logit: 13.88 Prob:  0.83% Token: | a|
Top 6th token. Logit: 13.34 Prob:  0.48% Token: | her|
Top 7th token. Logit: 13.26 Prob:  0.45% Token: | their|
Top 8th token. Logit: 13.25 Prob:  0.44% Token: | Jesus|
Top 9th token. Logit: 13.17 Prob:  0.41% Token: | Mrs|
Ranks of the answer tokens: [(' Mary', 0)]
```

In [19]:
```
import torch.nn as nn

old_mlp = model.blocks[training_cfg.hook_point_layer]

model.blocks[training_cfg.hook_point_layer].mlp = _TranscoderWrapper(trans

transformer_lens.utils.test_prompt(
    example_prompt,
    example_answer,
    model,
    prepend_bos=True,
)

model.blocks[training_cfg.hook_point_layer] = old_mlp
```

```
Tokenized prompt: ['<|endoftext|>', 'After', ' John', ' and', ' Mary', ' we
nt', ' to', ' the', ' store', ',', ' John', ' gave', ' a', ' bottle', ' o
f', ' milk', ' to']
Tokenized answer: [' Mary']
Performance on answer token:
Rank: 0        Logit: 18.39 Prob: 75.11% Token: | Mary|
Top 0th token. Logit: 18.39 Prob: 75.11% Token: | Mary|
Top 1th token. Logit: 15.74 Prob:  5.33% Token: | John|
Top 2th token. Logit: 15.69 Prob:  5.03% Token: | the|
Top 3th token. Logit: 14.92 Prob:  2.35% Token: | them|
```

```
Top 4th token. Logit: 14.24 Prob:  1.19% Token: | his|
Top 5th token. Logit: 13.88 Prob:  0.83% Token: | a|
Top 6th token. Logit: 13.34 Prob:  0.48% Token: | her|
Top 7th token. Logit: 13.26 Prob:  0.45% Token: | their|
Top 8th token. Logit: 13.25 Prob:  0.44% Token: | Jesus|
Top 9th token. Logit: 13.17 Prob:  0.41% Token: | Mrs|
Ranks of the answer tokens: [(' Mary', 0)]
```

Differences In Generated Text

In [20]:
```python
prompt = "The speech is about"

generated_text = model.generate(
    prompt,
    max_new_tokens=100,
    temperature=0,
    stop_at_eos=True,
)

print(generated_text)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
100%|██████████| 100/100 [00:02<00:00, 43.34it/s]
The speech is about the "new" American culture of "self-expression," which
is the "new" American culture of "self-expression," which is the "new" Amer
ican culture of "self-expression," which is the "new" American culture of
"self-expression," which is the "new" American culture of "self-expressio
n," which is the "new" American culture of "self-expression," which is the
"new" American culture of "self-expression," which is the "
```

In [21]:
```python
prompt = "The speech is about"

old_mlp = model.blocks[training_cfg.hook_point_layer]

model.blocks[training_cfg.hook_point_layer].mlp = _TranscoderWrapper(trans

generated_text = model.generate(
    prompt,
    max_new_tokens=100,
    temperature=0,
    stop_at_eos=True,
)

model.blocks[training_cfg.hook_point_layer] = old_mlp

print(generated_text)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
100%|██████████| 100/100 [00:01<00:00, 51.87it/s]
The speech is about the "new" American culture of "self-expression," which
is the "new" American culture of "self-expression," which is the "new" Amer
ican culture of "self-expression," which is the "new" American culture of
"self-expression," which is the "new" American culture of "self-expressio
n," which is the "new" American culture of "self-expression," which is the
"new" American culture of "self-expression," which is the "
```

In [ ]:

```
```

Top Activating Examples

In [39]:

```
# now we can unload gpu
torch.cuda.empty_cache()
```

In [22]:

```
# don't shuffle, that way we can lookup token batches by index
dataloader = create_tokenized_dataloader(shuffle=False)
```

```
Loading dataset...
Tokenizing dataset...
torch.Size([10000, 128])
Creating dataloader for dataset...
Num batches: 156.25
```

In [23]:

```
num_batches = len(dataloader.batch_sampler)

num_batches
```

Out[23]: 157

In [24]:

```
# store the MLP activations
mlp_inputs: list[Float[torch.Tensor, "batch seq d_mlp_in"]] = []
mlp_outputs: list[Float[torch.Tensor, "batch seq d_mlp_out"]] = []
reconstructed_mlp_outputs: list[Float[torch.Tensor, "batch seq d_mlp_out"]

# put model itself into eval mode so doesn't change
model.eval()
transcoder.eval()

for batch_index, batch in tqdm.tqdm(
    enumerate(dataloader),
    desc="Collecting MLP activations",
):
    # move batch to device
    batch = batch.to(device)

    # Get MLP input and output activations
    model.run_with_hooks(
        batch,
        fwd_hooks=[
            (training_cfg.hook_point, store_mlp_inputs),
            (training_cfg.out_hook_point, store_mlp_output),
        ],
        return_type=None,
    )

    # also reconstruct the mlp outputs using the transcoder
    mlp_input = mlp_inputs[-1].to(device)

    transcoder_result = transcoder(mlp_input)
```

```
    reconstructed_mlp_output = transcoder_result.transcoder_out

    reconstructed_mlp_outputs.append(reconstructed_mlp_output.detach().cpu
```

```
Collecting MLP activations: 157it [01:11,  2.21it/s]
```

In [32]:
```
# access corresponding batch directly
batch = dataloader.dataset[batch_index : batch_index + dataloader.batch_si

print(batch.shape)
```

```
torch.Size([128, 128])
```

In [27]:
```
dataloader.dataset.shape
```

Out[27]:
```
torch.Size([10000, 128])
```

In [26]:
```
import einops


def stack_list_of_tensors(
    tensors: list[Float[torch.Tensor, "..."]]
) -> Float[torch.Tensor, "num_batches ..."]:

    # Drop the last batch in case it doesn't match the rest of that shapes
    if tensors[-1].shape[0] != tensors[0].shape[0]:
        tensors = tensors[:-1]

    return torch.stack(tensors)


# Now we can safely stack the tensors
mlp_inputs_tensor: Float[torch.Tensor, "num_batches batch seq d_model"] =
    stack_list_of_tensors(mlp_inputs)
)
mlp_outputs_tensor: Float[torch.Tensor, "num_batches batch seq d_model"] =
    stack_list_of_tensors(mlp_outputs)
)
reconstructed_mlp_outputs_tensor: Float[
    torch.Tensor, "num_batches batch seq d_model"
] = stack_list_of_tensors(reconstructed_mlp_outputs)

print(f"{mlp_inputs_tensor.shape=}")
print(f"{mlp_outputs_tensor.shape=}")
print(f"{reconstructed_mlp_outputs_tensor.shape=}")
```

```
mlp_inputs_tensor.shape=torch.Size([156, 64, 128, 768])
mlp_outputs_tensor.shape=torch.Size([156, 64, 128, 768])
reconstructed_mlp_outputs_tensor.shape=torch.Size([156, 64, 128, 768])
```

In [38]:
```
# reshape out input dataset to match these as well
#
# 9984 = num_batches=156 * batch=64 (original num samples is 10000)
num_batches = mlp_outputs_tensor.shape[0]
batch_size = mlp_outputs_tensor.shape[1]
num_samples = num_batches * batch_size  # Use batch_size instead of batch
```

```
# dataloader.dataset is `torch.Size([10000, 128])`
# Remove the second dimension in slicing
token_inputs = dataloader.dataset[:num_samples]

token_inputs_tensor = einops.rearrange(
    token_inputs,
```