

✓ Latent Space Clustering and Summarization

Matthew Shinkle

ARENA Hackathon

September 15, 2024

Inspired by colah.github.io/posts/2014-10-Visualizing-MNIST

I wanted to see how modern dimensionality reduction and clustering approaches can support visualization and interpretation of LLM latent spaces. I explored a number of different approaches and algorithms, but ultimately converged on UMAP for dimensionality reduction and birch clustering to extract groups of tokens in the latent space of a layer.

I also wanted to get hands-on experience with using LLM APIs to generate summaries of clusters of tokens. I used the OpenAI api with chatgpt-3.5 to summarize tokens corresponding to each cluster.

Finally, I used plotly to visualize the low-dimensional summarization of the latent space, including clusters and cluster summaries.

These aren't super novel idea, but they are fun, and achieved my goal of 1) sating my curiosity of what this would look like, and 2) gaining more experience with these techniques.

Thanks!

✓ Extract Latent Activations

```
1 import torch
2 from tqdm import tqdm
3 from torch.utils.data import DataLoader, TensorDataset
4 import transformer_lens
5
6 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
7
8 # Load model and tokenizer
9 model = transformer_lens.HookedTransformer.from_pretrained("pythia-14m", device=device)
10
11 # Get list of all tokenizer vocabulary
12 vocab = model.tokenizer.get_vocab()
13 vocab = {k: v for k, v in list(vocab.items())}
14
15 # Get token IDs from vocab
16 token_ids_list = [vocab[token] for token in vocab.keys()]
17 token_ids_tensor = torch.tensor(token_ids_list, dtype=torch.long).unsqueeze(1)
```

```

18
19 # Create TensorDataset and DataLoader
20 dataset = TensorDataset(token_ids_tensor)
21 batch_size = 256
22 dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=False)
23
24 all_activations = {}
25 all_outputs = []
26
27 # Iterate over DataLoader
28 with torch.no_grad():
29     for token_tensor_batch in tqdm(dataloader):
30         token_tensor_batch = token_tensor_batch[0].to(device) # Extract the tensor and move to device
31         batch_logits, cache = model.run_with_cache(token_tensor_batch)
32         all_outputs.append(torch.argmax(batch_logits, dim=-1).squeeze())
33
34         # Accumulate activations
35         for layer_key in cache.keys():
36             batch_activations = cache[layer_key].cpu().mean(1) # Compute mean over token dim
37             if layer_key not in all_activations:
38                 all_activations[layer_key] = []
39             all_activations[layer_key].append(batch_activations)
40
41 all_outputs = torch.cat(all_outputs)
42 # Concatenate activations for each layer
43 all_activations = {key: torch.cat(all_activations[key], dim=0) for key in all_activations}

```



100% | 197/197 [00:12<00:00, 16.06it/s]

✓ Compute Clusters and Generate Cluster Descriptions

```

1 from interdim import InterDimAnalysis
2 import openai
3 import numpy as np
4 import plotly.io as pio
5 import plotly.graph_objects as go
6
7 for block_i in range(6):
8     layer_name = f'blocks.{block_i}.mlp.hook_pre'
9
10    ida = InterDimAnalysis(all_activations[layer_name].numpy())
11    ida.reduce(method='UMAP', n_components=3)
12    ida.cluster(method='birch', which_data='reduced')
13
14    input_predictions = [
15        ".join([token,pred]) for token, pred in zip(model.tokenizer.batch_decode(vocab.values()), model.tokenizer.batch_decode(all_o
16    ]
17
18    list(np.array(input_predictions)[ida.cluster_labels==7])
19
20 # Define your OpenAI API key
21 openai.api_key = 'YOUR API KEY HERE'
22
23 category_descriptions = {}
24 for category_idx in tqdm(np.unique(ida.cluster_labels)):

```

```

25 # Create a prompt to ask the model to summarize similarities
26 prompt = ', '.join(list(np.array(input_predictions)[ida.cluster_labels==category_idx]))
27
28 response = openai.ChatCompletion.create(
29     model="gpt-3.5-turbo", # or "gpt-4" if you have access
30     messages=[
31         {"role": "system", "content": "Very briefly state the commonality between the provided words. Example responses: 'numeric'
32         {"role": "user", "content": prompt}
33     ]
34 )
35
36 category_descriptions[str(category_idx)] = response['choices'][0]['message']['content']
37
38 # Create the 3D scatter plot using Plotly Graph Objects
39 fig = go.Figure(
40     data=[
41         go.Scatter3d(
42             x=ida.reduced_data[:, 0],
43             y=ida.reduced_data[:, 1],
44             z=ida.reduced_data[:, 2],
45             mode='markers',
46             marker=dict(
47                 size=3,
48                 opacity=0.7,
49                 color=ida.cluster_labels, # Color points by cluster labels
50                 colorscale='Rainbow', # Optional: customize colorscale
51             ),
52             hovertemplate='<b>Token:</b> {customdata[0]}<br><b>Cluster:</b> {customdata[1]}<extra></extra>',
53             customdata=np.stack((input_predictions, [category_descriptions[str(idx)] for idx in ida.cluster_labels]), axis=-1), # Provic
54         )
55     ]
56 )
57
58 # Update layout for size and presentation
59 fig.update_layout(
60     title=layer_name,
61     scene=dict(
62         xaxis_title="",
63         yaxis_title="",
64         zaxis_title=""
65     ),
66     width=1000,
67     height=1000
68 )
69
70 # Show the interactive plot
71 fig.show()
72 # pio.write_html(fig, file=f'{layer_name}_clusters.html', auto_open=True)

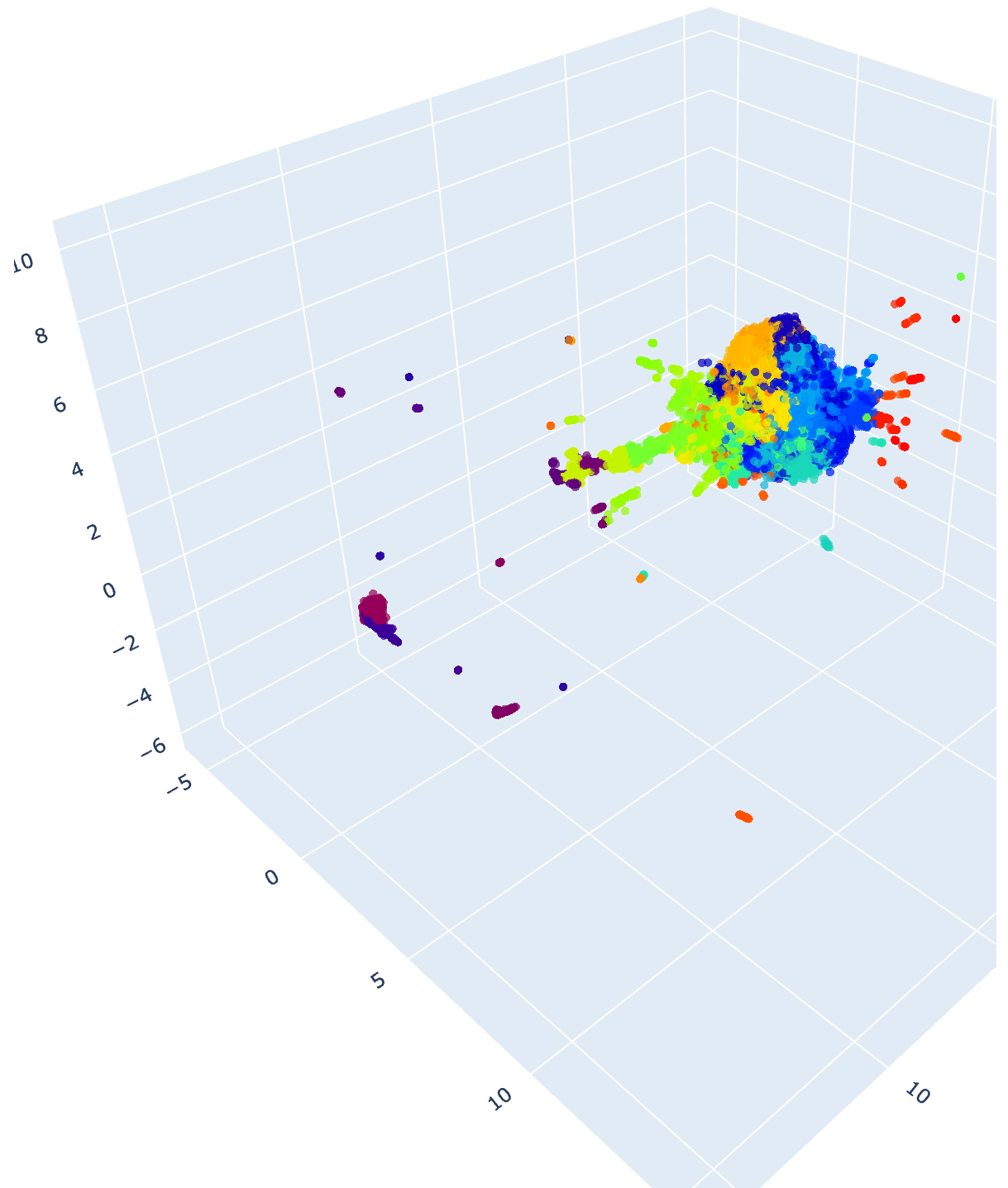
```



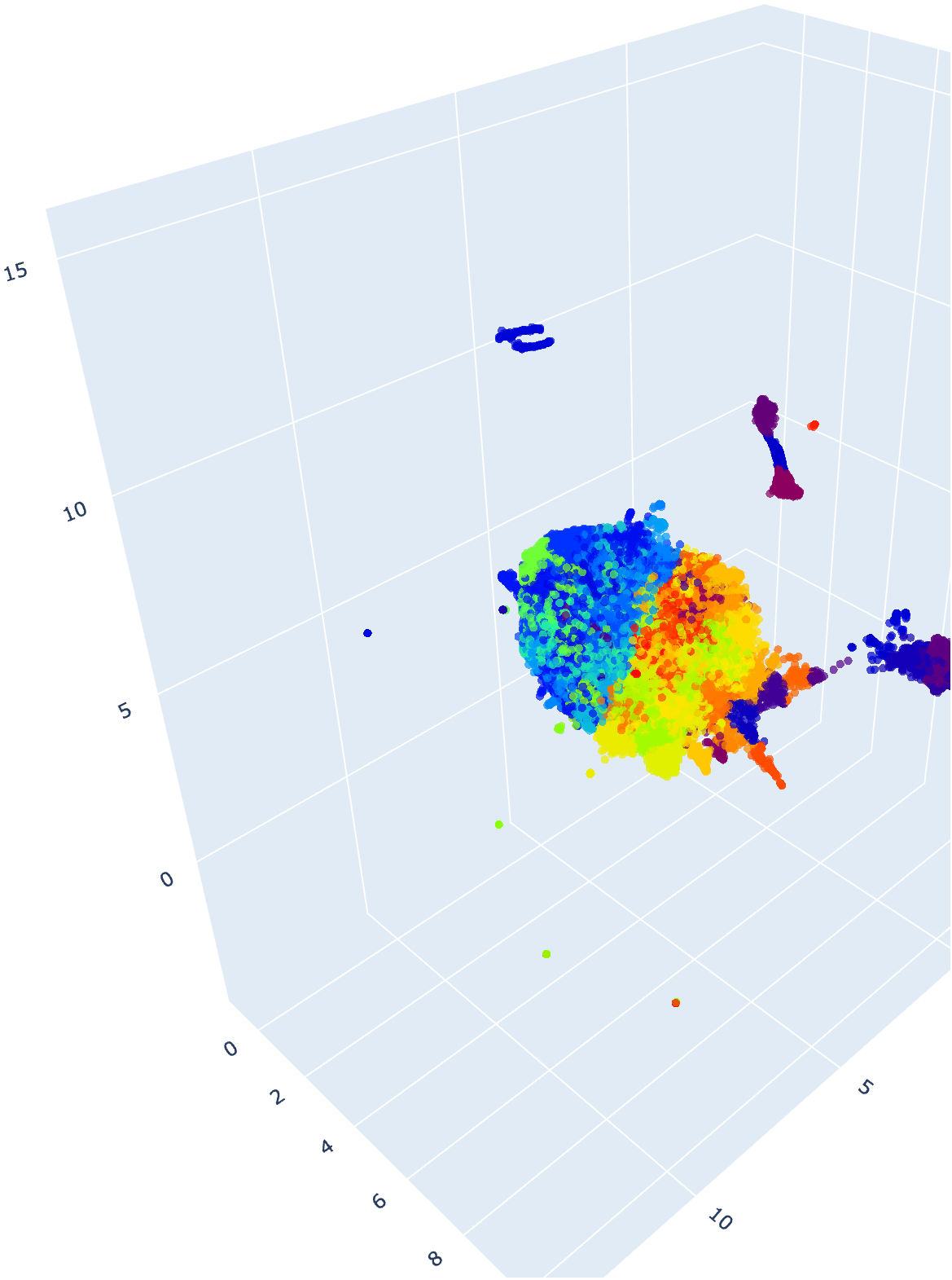
Show hidden output



blocks.0.mlp.hook_pre



blocks.1.mlp.hook_pre



blocks.2.mlp.hook_pre

