[MD]

Steering an LLM at inference time (SAE feature vectors)

This notebook reproduces the core idea from the Hugging Face Eiffel Tower demo:

- Load a base chat model (Llama 3.1 8B Instruct)
- Load a small set of pre-extracted SAE feature vectors (steering vectors)
- Add those vectors to the hidden states of specific transformer layers using forward hooks
- Compare baseline vs steered outputs while sweeping a steering scale

References (you can open these):

- Live demo + code: https://huggingface.co/spaces/huggingface/eiffel-tower-llama-demo
- Repo tree: https://huggingface.co/spaces/huggingface/eiffel-tower-llama-demo/tree/main
- Neuronpedia (browse SAE features): https://neuronpedia.org/
- SAE collections for Llama 3.1 8B Instruct (used by the demo): https://huggingface.co/andyrdt/saes-llama-3.1-8b-instruct

Notes

- This does not fine-tune the model. We only modify activations during generation.
- For gated models like Llama, you must accept the license on Hugging Face and provide an HF token.

[IN]

PYTHON

```
# Colab setup
# If you run on Colab, run this cell once.
# Restart the runtime if Colab asks after installs.

!pip -q install -U transformers accelerate huggingface-hub safetensors sentencepiece
```

[IN]

PYTHON

```
# Hugging Face login (Colab friendly)
# Do NOT use google.colab.userdata here.
# This version works in Colab and local notebooks.

import os
from getpass import getpass
from huggingface_hub import login

hf_token = os.environ.get("HF_TOKEN", "").strip()
if not hf_token:
    hf_token = getpass("Paste your HF token (it will not be shown): ").strip()

if hf_token:
    login(token=hf_token)
    print("HF token found and login success")
else:
    print("No HF token provided. Gated models may fail to download.")
```

OUTPUT

```
Paste your HF token (it will not be shown): ··········
HF token found and login success
```

[IN]

PYTHON

```python
# Configuration (matches demo.yaml from the HF Space)

cfg = {
    # Model
    "llm_name": "meta-llama/Llama-3.1-8B-Instruct",

    # These are only for reference (the notebook loads pre-extracted vectors)
    "sae_path": "andyrdt/saes-llama-3.1-8b-instruct",
    "sae_filename_prefix": "resid_post_layer_",
    "sae_filename_suffix": "/trainer_1/ae.pt",

    # Steering feature list: [layer, feature_id, strength]
    # You can edit these to try different vectors if you have them in steering_vectors.pt
    "reduced_strengths": True,
    "features": [
        [11, 74457, 0.128],
        [11, 18894, 0.255],
        [11, 61463, 0.132],
        [15, 21576, 0.103],
        [19, 93,    0.459],
        [23, 111898,0.466],
        [23, 40788, 0.228],
        [23, 21334, 0.043],
    ],

    # Generation defaults
    "temperature": 0.5,
    "seed": 16,
    "max_new_tokens": 256,
    "repetition_penalty": 1.2,
}
```

[IN]

PYTHON

```python
# Download the pre-extracted steering vectors (steering_vectors.pt) from the HF Space
# If this download fails, you can manually download it from the Space files tab and upload it to your runtime.

from huggingface_hub import hf_hub_download

STEERING_FILE = hf_hub_download(
    repo_id="huggingface/eiffel-tower-llama-demo",
    repo_type="space",
    filename="steering_vectors.pt",
    token=hf_token,
)

print("Downloaded steering vectors to:", STEERING_FILE)
```

OUTPUT

```
steering_vectors.pt:   0%|          | 0.00/135k [00:00<?, ?B/s]
```

```
Downloaded steering vectors to: /root/.cache/huggingface/hub/spaces--huggingface--eiffel-tower-llama-demo/snapshots/65fc169c756fad318c
```

[IN]

PYTHON

```python
# Load model and tokenizer

import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

torch.manual_seed(int(cfg.get("seed", 0)))

# Device and dtype
use_cuda = torch.cuda.is_available()
dtype = torch.float16 if use_cuda else torch.float32

print("cuda available:", use_cuda)
print("dtype:", dtype)

llm_name = cfg["llm_name"]
print("Loading model:", llm_name)

model = AutoModelForCausalLM.from_pretrained(
    llm_name,
    device_map="auto",
    torch_dtype=dtype,
    token=hf_token,
)

tokenizer = AutoTokenizer.from_pretrained(
    llm_name,
    token=hf_token,
)

# Some tokenizers do not have a pad token. Use EOS for generation safety.
if tokenizer.pad_token_id is None:
    tokenizer.pad_token_id = tokenizer.eos_token_id

# For decoder-only models, left padding usually works better for batched generation
tokenizer.padding_side = "left"

# Set model defaults as well
model.generation_config.pad_token_id = tokenizer.pad_token_id
model.generation_config.eos_token_id = tokenizer.eos_token_id

print("Model loaded. n_layers:", len(model.model.layers))
```

OUTPUT

```
cuda available: True
dtype: torch.float16
Loading model: meta-llama/Llama-3.1-8B-Instruct
```

```
config.json:   0%|          | 0.00/855 [00:00<?, ?B/s]
```

```
`torch_dtype` is deprecated! Use `dtype` instead!
```

```
model.safetensors.index.json:   0%|          | 0.00/23.9k [00:00<?, ?B/s]
```

```
Fetching 4 files:   0%|          | 0/4 [00:00<?, ?it/s]
```

```
model-00002-of-00004.safetensors:   0%|          | 0.00/5.00G [00:00<?, ?B/s]
```

```
model-00001-of-00004.safetensors:   0%|          | 0.00/4.98G [00:00<?, ?B/s]
```

```
model-00004-of-00004.safetensors:   0%|          | 0.00/1.17G [00:00<?, ?B/s]
```

```
model-00003-of-00004.safetensors:   0%|          | 0.00/4.92G [00:00<?, ?B/s]
```

```
Loading checkpoint shards:   0%|          | 0/4 [00:00<?, ?it/s]
```

```
generation_config.json:   0%|          | 0.00/184 [00:00<?, ?B/s]
```

[IN]

PYTHON

```python
# Load steering components from the .pt file

import torch

def load_saes_from_file(file_path, cfg, device):
    steering_vectors_dict = torch.load(file_path, map_location="cpu")

    features = cfg.get("features", [])
    reduced_strengths = cfg.get("reduced_strengths", False)

    steering_components = []
    for feature in features:
        layer_idx, feature_idx = int(feature[0]), int(feature[1])
        strength = float(feature[2]) if len(feature) > 2 else 0.0

        if reduced_strengths:
            strength *= layer_idx

        key = (layer_idx, feature_idx)
        if key not in steering_vectors_dict:
            raise KeyError(f"Vector missing for key={key} in {file_path}")

        vec = steering_vectors_dict[key].to(device)
        # Normalize so direction matters, and strength controls intensity
        vec = vec / (vec.norm() + 1e-12)

        steering_components.append({
            "layer": layer_idx,
            "feature": feature_idx,
            "strength": strength,
            "vector": vec,
        })

        reduced_str = f"[{strength/layer_idx:.3f}]" if layer_idx > 0 else "[N/A]"
        print(f"Loaded layer={layer_idx} feature={feature_idx} strength={strength:.3f} {reduced_str}")

    print("Total steering vectors:", len(steering_components))
    return steering_components

steering_components = load_saes_from_file(STEERING_FILE, cfg, device="cpu")
```

OUTPUT

```
Loaded layer=11 feature=74457 strength=1.408 [0.128]
Loaded layer=11 feature=18894 strength=2.805 [0.255]
Loaded layer=11 feature=61463 strength=1.452 [0.132]
Loaded layer=15 feature=21576 strength=1.545 [0.103]
Loaded layer=19 feature=93 strength=8.721 [0.459]
Loaded layer=23 feature=111898 strength=10.718 [0.466]
Loaded layer=23 feature=40788 strength=5.244 [0.228]
Loaded layer=23 feature=21334 strength=0.989 [0.043]
Total steering vectors: 8
```

[IN]

PYTHON

```python
# Steering hook + generation utilities (no streaming, attention_mask included)

import torch

def create_steering_hook(layer_idx, steering_components, clamp_intensity=False, scale=1.0):
    layer_components = [sc for sc in steering_components if sc["layer"] == layer_idx]
    if not layer_components:
        return None

    def hook(module, inputs, output):
        # output can be a tensor or a tuple (hidden_states, ...)
        if isinstance(output, tuple):
            hidden_states = output[0]
            rest = output[1:]
        else:
            hidden_states = output
            rest = None

        original_shape = hidden_states.shape

        # During generation some models pass [batch, hidden_dim]
        if len(original_shape) == 2:
            hidden_states = hidden_states.unsqueeze(1)  # [batch, 1, hidden_dim]

        for sc in layer_components:
            strength = (sc["strength"] * float(scale))
            vec = sc["vector"].to(device=hidden_states.device, dtype=hidden_states.dtype)

            # Expand to [batch, seq_len, hidden_dim]
            batch = hidden_states.shape[0]
            seq_len = hidden_states.shape[1]
            amount = (strength * vec).view(1, 1, -1).expand(batch, seq_len, -1)

            if clamp_intensity:
                # Remove existing projection onto vec before adding new amount
                proj = torch.einsum("bsh,h->bs", hidden_states, vec).unsqueeze(-1)
                hidden_states = hidden_states - proj * vec.view(1, 1, -1)

            hidden_states = hidden_states + amount

        if len(original_shape) == 2:
            hidden_states = hidden_states.squeeze(1)

        if rest is not None:
            return (hidden_states,) + rest
        return hidden_states

    return hook


def make_chat(user_prompt, system_prompt="You are a helpful assistant."):
    return [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt},
    ]


def _apply_chat_template(chat):
    # transformers versions differ slightly. This tries the best path first.
    try:
        # Newer transformers: returns a tensor directly
        model_inputs = tokenizer.apply_chat_template(
            chat,
            add_generation_prompt=True,
            return_tensors="pt",
            padding=True,
            truncation=True,
        )
        if isinstance(model_inputs, torch.Tensor):
            input_ids = model_inputs
            attention_mask = torch.ones_like(input_ids)
```

[IN]

PYTHON

```python
# Baseline (no steering)

baseline_who = generate_chat(make_chat("Who are you?"), steer=False, temperature=0.0)
baseline_biz = generate_chat(make_chat("Give me some ideas for starting a business."), steer=False, temperature=0.0)

print("BASELINE: Who are you?\n")
print(baseline_who)
print("\n" + "=" * 80 + "\n")
print("BASELINE: Business prompt\n")
print(baseline_biz)
```

OUTPUT

```
The following generation flags are not valid and may be ignored: ['temperature', 'top_p']. Set `TRANSFORMERS_VERBOSITY=info` for more

BASELINE: Who are you?

I'm an artificial intelligence model known as a large language model or conversational AI. I'm here to assist and provide information

My primary function is to understand the context of your questions and respond with accurate and relevant answers. I can also generate

Think of me like a knowledgeable friend who's always available to chat and help with any questions or tasks you may have! What would y

================================================================================

BASELINE: Business prompt

Here are some business ideas across various industries:

**Online Businesses**

1. **E-commerce store**: Sell products through an online platform, leveraging social media and search engine optimization (SEO) to rea
2. **Digital marketing agency**: Offer services like SEO, PPC advertising, content creation, and social media management to businesses
3. **Online course creator**: Develop and sell courses on platforms like Udemy, Teachable, or Skillshare, teaching topics you're knowl
4. **Affiliate marketer**: Promote other people's products or services and earn commissions for each sale made through your unique ref

**Service-Based Businesses**

1. **Freelance writing or editing service**: Offer high-quality written content, such as blog posts, articles, or website copywriting,
2. **Virtual assistance**: Provide administrative support, email management, or scheduling services to busy professionals and entrepre
3. **Pet sitting or dog walking business**: Take care of furry friends while their owners are away, offering peace of mind and conveni
4. **House cleaning or organization service**: Help homeowners maintain a clean and organized living space with regular visits from tr

**Product-Based Businesses**

1. **
```

[IN]

PYTHON

```python
# Steered runs (clamp on): sweep scale and observe the "sweet spot"
# In the original demo, moderate scales change tone and topic without fully derailing.
# Large scales usually cause repetition or gibberish.

for scale in [0.5, 0.8,0.9 ]: # u can use other v;aues here less or more
    out_who = generate_chat(
        make_chat("Who are you?"),
        steer=True,
        scale=scale,
        clamp_intensity=True,
        temperature=0.0,
        max_new_tokens=256,
    )
    out_biz = generate_chat(
        make_chat("Give me some ideas for starting a business."),
        steer=True,
        scale=scale,
        clamp_intensity=True,
        temperature=0.0,
        max_new_tokens=256,
    )

    print("\n" + "#" * 80)
    print(f"STEERED (clamp on) scale={scale}: Who are you?")
    print("#" * 80)
    print(out_who)

    print("\n" + "-" * 80)
    print(f"STEERED (clamp on) scale={scale}: Business prompt")
    print("-" * 80)
    print(out_biz)
```

OUTPUT

```
################################################################################
STEERED (clamp on) scale=0.5: Who are you?
################################################################################
I'm an artificial intelligence designed to assist and communicate with users in a friendly and informative way. I can answer questions

My purpose is to help people learn new things, get answers to their questions, or simply chat about any topic that interests them. I c

What would you like to talk about today?

--------------------------------------------------------------------------------
STEERED (clamp on) scale=0.5: Business prompt
--------------------------------------------------------------------------------
Here are some unique and innovative business ideas across various industries:

**Food Industry**

1. **Gourmet Food Truck**: Offer exotic cuisine or specialty dishes in popular locations.
2. **Artisanal Bakery**: Create handcrafted bread, pastries, and cakes with unique flavors.
3. **Meal Kit Delivery Service**: Provide pre-portioned ingredients and recipes for home cooking.

**Sustainable Energy**

1. **Solar Panel Installation Company**: Help homeowners reduce energy bills by installing solar panels.
2. **Wind Turbine Maintenance Services**: Offer maintenance services to wind farms and turbines.
3. **Electric Vehicle Charging Station Network**: Build an extensive network of EV charging stations.

**Technology and Software**

1. **AI-Powered Chatbot Development**: Design intelligent chatbots for businesses and organizations.
2. **Cybersecurity Consulting Firm**: Protect companies from cyber threats and data breaches.
3. **Virtual Reality Experience Center**: Develop immersive VR experiences for entertainment and education.

**Health and Wellness**

1. **Personalized Fitness Coaching App**: Offer customized workout plans based on user preferences.
2. **Mental Health Support Platform**: Develop online resources and support groups for mental health awareness.
3. **Healthy Meal Prep Service**: Prepare nutritious meals using locally sourced ingredients.

**E-commerce

################################################################################
STEERED (clamp on) scale=0.8: Who are you?
################################################################################
```

[MD]

How to interpret results

What you are looking for:

- Baseline: the model answers normally.
- Steered (small scale): subtle topic drift and stylistic bias.
- Steered (sweet spot): clear personality change while staying fluent.
- Steered (too strong): repetitive text, broken words, or nonsense.

In the Eiffel Tower demo, a good sweet spot is often around scale = 1.0. Your exact sweet spot depends on:

- which layers you steer
- which features you use
- generation settings (temperature, repetition_penalty, max_new_tokens)

Practical tip

- Keep temperature at 0.0 while you debug steering.
- Use max_new_tokens around 128 to 256 to keep the notebook fast.

[MD]

How to try a different concept (SAE workflow)

Find a feature
- Go to Neuronpedia: https://neuronpedia.org/
- Select a model and SAE set that matches your LLM
- Search for a concept keyword and pick a feature id and layer

Get the feature vector There are two common paths:
- Download a prepared steering_vectors.pt (fastest)
- Extract vectors yourself from SAE checkpoints (slower but flexible)

Update this notebook
- Replace cfg["features"] with your new [layer, feature_id, strength] list
- Ensure your steering_vectors.pt contains those (layer, feature_id) keys
- Re-run from the steering load cell onward

Extra reading

- Contrastive activation steering (positive minus negative prompts)
- Sparse autoencoders for interpretable feature libraries

[MD]

Follow me on Github and Kaggle =)

[IN]

PYTHON