Why Steering Vectors Fail

Skye Purchase[1]

Machine Learning MSc

Daniel Tan & Brooke's Paige

Submission date: 8 September 2025

[1] **Disclaimer:** This report is submitted as part requirement for the Machine Learning MSc at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged

Abstract

Summarise your report concisely.

Contents

Chapter 1

Introduction

1.1 Related Work

Krasheninnikov and Krueger [1] aim to analyse steering in a toy environment where they are able to control the representation density within the model. They compare a range of steering techniques [2–4] against each other in a controlled setting to evaluate the benefits and drawbacks of each approach. Inspired by LoReFT [3] they introduce their own technique LoReST and demonstrate competitive performance to the other techniques.

This project reproduces a sample of plots from Figure 1 using the same toy setup described in §3.1. In addition to the techniques used in the original paper the reproduction also analyses the behaviour of [5].

This project aims to expand the analysis carried out by Krasheninnikov and Krueger [1] to reproduce the same effects in large language model systems. Additionally, the relationship between the negative and positive training examples is analysed to gain a better insight as to when steering approaches fail.

Tan et al. [6] aim to analyse the generalisation of steering vectors across a range of steering datasets. They analyse the variability of success and introduce the notion of steerability. Using this notion they demonstrate that many techniques fail to generalise on certain datasets both in and out of distribution.

The analysis is limited to only contrastive activation addition [2] which Krasheninnikov and Krueger [1] show is not necessarily the ideal candidate. Building on their work this project aims to analyse a larger range of techniques sampled from Krasheninnikov and Krueger [1]. Furthermore, the properties of training datasets is analysed in more depth to determine which properties cause steering techniques to fail.

Rather than use model written evaluations [7] a new set of steering datasets is generated with more fine grain control. The construction of these datasets is described in §3.2.

Wehner et al. [8]

1.2 Contributions

Chapter 2

Background

2.1 Notation and Concepts

Model “behaviours”, in general, are patterns in how the model responds to input. This includes the desired behaviour it was trained on (such as classifying images of cats and dogs) but includes patterns in the output that were not explicitly trained for. Desired model behaviour is considered “positive” and undesired model behaviour is considered “negative”. Specifically, an example of the desired behaviour is considered a “positive example” and an example of undesired or neutral behaviour is considered a “negative” example. An example of a behaviour generally includes an input-output pairing similar to training examples however they are more specific pairings than would be using during training.

Throughout the document neural network (NN) and machine learning (ML) model are used interchangeably though NNs are a strict subset of ML models. When discussing NNs the concept of a “neuron” relates to the abstract structure that receives a real-valued, vector input and outputs a real-value scalar based on internal, learnable weights. In practice, this is represented by a single element of a NN layer’s output vector.

Vectors are represented by boldface letters, $\mathbf{x}, \mathbf{y}, \mathbf{z}$, scalars are represented by greek letters, α, β, γ, and matrices are represented by boldface capital letters, $\mathbf{A}, \mathbf{B}, \mathbf{C}$. Some matrices may represent transformations or collections of feature vectors, context should disambiguate the two. In general vectors are column vectors, $\mathbf{x} = \begin{bmatrix} 1 & 2 & \cdots & n \end{bmatrix}^T$ except when a collection of vectors is represented in matrix form, in this case each row is a vector.

In a multi-layer machine learning model the output of an internal layer is an “activation” denoted $\mathbf{a}$. A positive activation is denoted $\mathbf{a}^+$ and a negative activation is denoted as $\mathbf{a}^-$. Here, “positive activation” means the activation extracted from the model given a positive example as above.

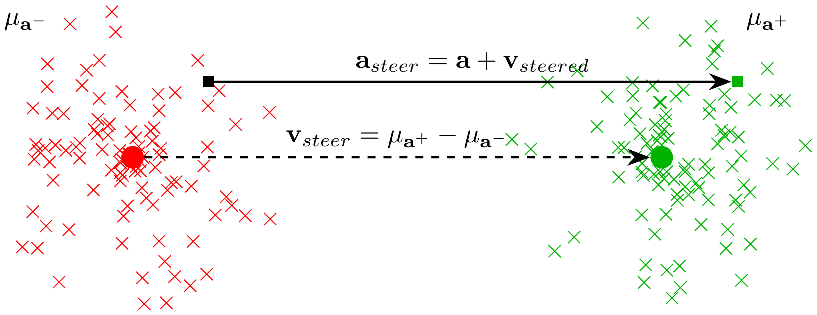
$\mu_{\mathbf{a}^-}$
$\mu_{\mathbf{a}^+}$
$\mathbf{a}_{steer} = \mathbf{a} + \mathbf{v}_{steered}$
$\mathbf{v}_{steer} = \mu_{\mathbf{a}^+} - \mu_{\mathbf{a}^-}$

Figure 2.1: Demonstration of contrastive activation addition [2]. The figure represents a simple representation space of dimension 2 with clear separability. $\mu_A = \frac{1}{\|A\|}\sum_{i \in \mathcal{I}(A)} A_i$ where A is a set of activation vectors. A new point, such as the black square, is translated by the steering vector.

2.2 Model Alignment

2.3 Model Intervention

2.3.1 Contrastive Activation Addition

An intuitative approach to model intervention is to perturb the model's activations in a desired direction. By calculating a linear direction in activation space from undesired activations towards desired ones this vector can simply be added to all activations in the model during inference. The hope is that the model produces output that matches the desired behaviour whilst maintaining the context of the new input.

In the simplest form consider two example inputs with desired and undesired behaviour. Their difference gives a direction in feature space that corresponds to shifting the models output from undesired behaviour towards desired behaviour. This is the approach proposed by Turner et al. [9], however, it is not robust and relies heavily on the example inputs [2].

To improve on this approach Rimsky et al. [2] suggest using a collection of examples and calculating their mean difference in activation space. This requires the notion of *contrastive pairs*, two inputs that are similar in all ways except for the behaviour that is being changed. Hence, this approach is known as *contrastive activation addition* (CAA). This process is demonstrated in Figure 2.1.

Formally, given a set of positive example activations $(\mathbf{a}_i^+)_{i \leq n}$ and negative example activations $(\mathbf{a}_i^-)_{i \leq n}$ a *steering vector* for this behaviour is

$$\mathbf{v}_{steer} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{a}_i^+ - \mathbf{a}_i^-\right).$$

Given a steering vector, $\mathbf{v}_{steer}$, and a model activation during inference, $\mathbf{a}$, the resulting steered activation is

$$\mathbf{a}_{steered} = \mathbf{a} + \lambda \mathbf{v}_{steer} \tag{2.1}$$

where λ is a user-defined parameter controlling the strength of the steering intervention. The model activation is replaced by the steered activation during inference resulting in the model producing an output aligned with the positive examples.

This approach has a few drawbacks [5, 6, 10] due to its assumptions. Primarily this approach does not consider how much of a behaviour is already present. This means the steering parameter does not fully determine the strength of the desired behaviour. Furthermore, Tan et al. [6] demonstrate that this approach is not robust across behaviours that may be steered along. The approach assumes that concepts in activation space are linear which Engels et al. [10] show is not universal. Techniques such as affine concept editting (ACE) §2.3.2 use an affine approach to overcome these drawbacks.

2.3.2 Affine Concept Editting

Marshall et al. [5] claim that CAA [2] is not sufficiently general as it does not consider how much the desired behaviour is already present. To see this consider an arbitrary activation vector $\mathbf{a}$ and steering direction $\mathbf{r}$ encoding some behaviour. $\mathbf{a}$ can be decomposed as the perpendicular and parallel components of $\mathbf{r}$

$$\begin{aligned}\mathbf{a} &= \text{proj}_{\mathbf{r}}^{\perp}(\mathbf{a}) + \text{proj}_{\mathbf{r}}^{||}(\mathbf{a}) \\ &= \text{proj}_{\mathbf{r}}^{\perp}(\mathbf{a}) + \alpha\mathbf{r}.\end{aligned}$$

Adding $\lambda\mathbf{r}$ as per Equation 2.1 will be inconsistent as $\alpha + \lambda$ will not be equivalent across all (negative) activations. This shows that CAA [2] does not account for how much a behaviour may already be present in an activation.

Furthemore, it is not (generally) the case that $\mathbf{0}$ represents lack of behaviour. Instead there is some vector $\mathbf{a}_0$ that represents the lack of the target behaviour. The above equation can incorporate this idea as follows

$$\begin{aligned}\mathbf{a} &= \mathbf{a}_0 + \Delta\mathbf{a} \\ &= \mathbf{a}_0 + \text{proj}_{\mathbf{r}}^{\perp}(\Delta\mathbf{a}) + \text{proj}_{\mathbf{r}}^{||}(\Delta\mathbf{a}) \\ &= \mathbf{a}_0 + \text{proj}_{\mathbf{r}}^{\perp}(\Delta\mathbf{a}) + \alpha'\mathbf{r}.\end{aligned}$$

Removing the behaviour by setting $\alpha' = 0$ yields

$$\begin{aligned}\mathbf{a}' &= \mathbf{a}_0 + \text{proj}_{\mathbf{r}}^{\perp}(\Delta\mathbf{a}) \\ &= \mathbf{a} - \text{proj}_{\mathbf{r}}^{||}(\Delta\mathbf{a}) \\ &= \mathbf{a} - \text{proj}_{\mathbf{r}}^{||}(\mathbf{a}) + \text{proj}_{\mathbf{r}}^{||}(\mathbf{a}_0) \\ &= \mathbf{a} - \text{proj}_{\mathbf{r}}^{||}(\mathbf{a}) + \alpha_0\mathbf{r}.^{1}\end{aligned}$$

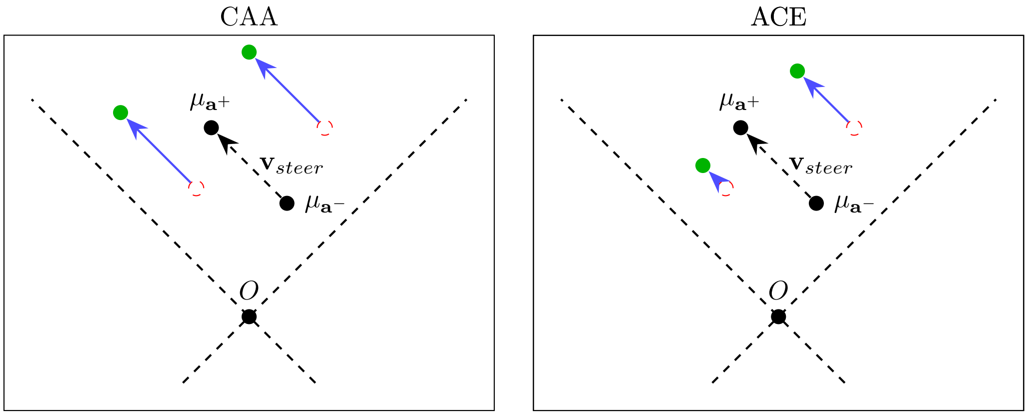

Figure 2.2: A comparison of CAA [2] and affine concept editting [5]. This is a reproduction of Figure 1 in Marshall et al. [5] with the steering towards the positive examples instead. Compared to CAA, ACE does not adjust perpendicular components but correctly adjusts those parallel to the steering direction.

This represents the activation lacking the target behaviour but retaining other relevant context. The behaviour can be reintroduced at any relevant strength resulting in

$$\mathbf{a}_{\text{steered}} = \mathbf{a}_0 - \text{proj}^{\|}_{\mathbf{r}}(\mathbf{a}) + \alpha_0 \mathbf{r} + \alpha \mathbf{r}. \tag{2.2}$$

Given positive example activations $(\mathbf{a}_i^+)_{i \leq n}$ and negative example activations $(\mathbf{a}_i^-)_{i \leq n}$ the reference point and steering direction are

$$\mathbf{a}_0 = \frac{1}{n} \sum_{i=1}^{n} \mathbf{a}_i^- \qquad\qquad \mathbf{r} = \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{a}_i^+ - \mathbf{a}_i^- \right).$$

This process is described graphically in Figure 2.2

This approach is no longer a linear edit to the activations and now includes a bias term, $\mathbf{a}_0$. This is therefore affine and hence the name *affine concept editting* (ACE) [5].

2.3.3 Low-rank Representation Finetuning

Both CAA [2] and ACE [5] edit the activations in their full rank form and rely on addition (whether affine or linear). This limits the transforms the approaches can apply to the activation space. If the desired behaviour requires rotations or scaling of the activations these methods fail. However, perform affine transformations to the full rank activation is costly as the dimension of the activations may be large.

Wu et al. [3] present a low-rank steering adaptor inspired by parameter-efficient finetuning methods such as LoRA [11], DoRA [12] and adaptor-based methods [13]. Unlike steering these approaches aim to finetune a model using reduced parameter counts compared to the original model. Rather than finetuning a model this approach aims to edit the representations of the model, this is

[1] As $\mathbf{a}_0$ exists as a reference point along the steered direction.

equivalent to steering.

The key insight is to steer the activations in a low-rank space. The specific approach is based on the distributed interchange intervention[1] [14] with the following form

$$DII(\mathbf{x}, \mathbf{y}, \mathbf{R}) = \mathbf{x} + \mathbf{R}^T(\mathbf{Ry} - \mathbf{Rx})$$

where $\mathbf{R} \in \mathbb{R}^{r \times d}$ is a low-rank projection matrix.

Wu et al. [3] suggest replacing $\mathbf{Ry}$ with an affine transformation $\mathbf{Wx}+\mathbf{b}$. Thus, the adaptor learns a transformation, $\mathbf{Ra}^+ = \mathbf{Wa}^- + \mathbf{b}$, from negative activations to positive low-rank representations. In this way the adaptor can learn low-rank representations of activations that encapsulate the desired behaviour and adjust the activations in a parameter efficient space. The approach is therefore a *low-rank representation finetuning* (LoReFT) adaptor. The full adaptor is

$$\mathbf{a}_{\text{steered}} = \mathbf{a} + \mathbf{R}^T(\mathbf{Wa} + \mathbf{b} - \mathbf{Ra}). \tag{2.3}$$

The learnable parameters of the adaptor are $\phi = \{\mathbf{W}, \mathbf{R}, \mathbf{b}\}$. $\mathbf{R}$ is constrained to be an orthogonal projection matrix achieved by differentiable QR decomposition.

Given a dataset of contrastive pairs $\mathcal{D} = (\mathbf{a}_i^-, \mathbf{a}_i^+)_{i \leq n}$ the adaptor parameters ϕ are trained. The goal is to accurately predict $\mathbf{a}_i^+$ given $\mathbf{a}_i^-$ as input.

Unlike CAA [2] and ACE [5] this approach requires paired datapoints as the adaptor needs to learn a transformation from negative examples to positive examples. This drawback means that in the low data regime this approach is less effective than the other two approaches. However, with sufficient data, this method is able to outperform CAA and ACE as it can utilise more complex transformations between negative and positive behaviour. The poor performance in low data regimes is improved on by Krasheninnikov and Krueger [1] with their low-rank representation steering adaptor.

2.3.4 Low-rank Representation Steering

Krasheninnikov and Krueger [1] suggest modifying LoReFT [3] to dynamically drop low-rank dimensions and bring the learnable bias term outside of the low-rank space. This allows the model to perform well in the low data regime by relying on linear methods similar to CAA [2] but keep the benefits of LoReFT. By dynamically dropping dimensions the adaptor has more freedom to optimise the rank of the projection.

Krasheninnikov and Krueger [1] define an orthogonal projection

$$\mathbf{P} = \mathbf{I} - \mathbf{Q}\text{diag}(\mathbf{p})\mathbf{Q}^T \qquad\qquad \mathbf{p}_i = \text{GumbelSoftmax}([\mathbf{l}_i, 0]; \tau)$$

where $\mathbf{Q} \in \mathbb{R}^{r \times d}$ is a learnable low-rank projection matrix, $\mathbf{l}$ is a learnable Gumbel Softmax dis-

[1] This tests whether a concept is encoded in some subspace. When working with low-rank editting this is exactly the assumption we use.

tribution probabilities, and τ is the temperature. As with LoReFT, $\mathbf{Q}$ is an orthogonal projection achieved by differentiable QR decomposition. In comparison to LoReFT Equation 2.3 there is no representation editting in the low-rank space. Instead the projection acts as a method to "zero" the activation similar to ACE [5].

The full adaptor is

$$\mathbf{a}_{\text{steered}} = \mathbf{a} - (\mathbf{a}\mathbf{Q})\text{diag}(\mathbf{p})\mathbf{Q}^T + \mathbf{b}. \tag{2.4}$$

This approach also requires paired data to train the parameters, $\phi = \{\mathbf{Q}, \mathbf{l}, \mathbf{b}\}$. $\mathbf{Q}$ is constrained to be orthogonal through differentiable QR decomposition.

Given a dataset of contrastive pairs $\mathcal{D} = (\mathbf{a}_i^-, \mathbf{a}_i^+)_{i \leq n}$ the adaptor parameters ϕ are trained. The goal is to accurately predict $\mathbf{a}_i^+$ given $\mathbf{a}_i^-$ as input.

In the low data regime the adaptor can learn to drop more dimensions and rely on $\mathbf{b}$ similar to CAA [2] and ACE [5]. As more data is available the adaptor can rely more on the low-rank projection similar to LoReFT [3]. In this way the adaptor is able to perform consistently across different data regimes.

2.4 Large Language Models

Steering and model alignment in general is not confined to large language models (LLM)s however these are currently the most widespread model in use. LLMs aim to immitate, complete, or analyse natural language and are characterised by incredibly large numbers of parameters. Some are as large as 1.76 trillion [15] and even small models have as many as 1 billion [16].

Only generative LLMs are discussed in this project. These are models which are not trained to classify or fit a dataset in the classical sense but instead to produce more data as if it were sampled from the underlying training distribution. In the case of LLMs this means producing coherent natural language.

The underlying technology behind modern generative LLMs is the transformer [17] and their many derivatives [18–20].

2.4.1 Transformers

Transformers [17] are now a mainstay of modern deep learning.[2] They utilise the attention mechanism to dynamically transform (sequential) input based on the surrounding context.

Attention can be considered as a learnable lookup table with queries, keys and values. If a query and a key are similar then the corresponding value should be returned. This can be represented as a dot-product between a matrix of queries $\mathbf{Q}$ and keys $\mathbf{K}$. These are normalised to act as probabilities that a specific value is the target value. Given a matrix of values $\mathbf{V}$ attentio is

[2] This section does not aim to describe transformers in full detail but provide a sufficient background for the rest of the project. Keywords are provided for further reading and the explanation is based on the paper by Turner [21].

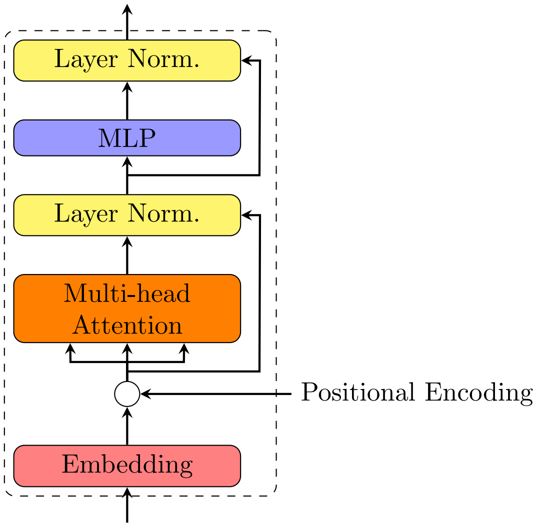

Figure 2.3: A diagram of the standard transformer decoder block. This is based on Figure 1 of Vaswani et al. [17]. This is a single layer in a large language model where the output of one block is fed into the input of the next.

represented by the following equation

$$\operatorname{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}.$$

The trick is to have $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ all depend on the input features, this is known as "self-attention". In this way $\mathbf{V}$ behaves like a standard weight transformation and the softmax of $\mathbf{Q}$ and $\mathbf{K}$ behave like a dynamic weight transformation dependent on the input. This allows a model to "attend" to different parts of the input by adjusting the transformation matrices that make $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$.

Modern transformers contain attention blocks each containing multiple "attention heads" that use the above mechanism. This allows the model to respond dynamically to a large range of inputs. After this attention block a standard multi-layer perceptron (MLP) is added. This constitutes the transformer block and is visualised in Figure 2.3.

The ability for transformers to utilise context in surrounding input values makes them particularly suited to natural language processing (NLP). The meaning of words in a sentence depend on the words that surround it. Furthermore, the words depend on each other in different ways depending on the context. This is precisely how transformer attention blocks work allowing them to parse natural language far better than previous attempts.

For transformers to work on natural language the input needs to be tokenized into descrete chunks, frequently based on words. These chunks can then be converted to unique numbers and later represented as input features. To aid the model, the position of the token within the sentence is also encoded this is known as "positional encoding". This allows the model to distinguish between the two instances of "can" in the sentence "can you pass me the can".

It is important to note that when training a model to generate natural language it must be trained

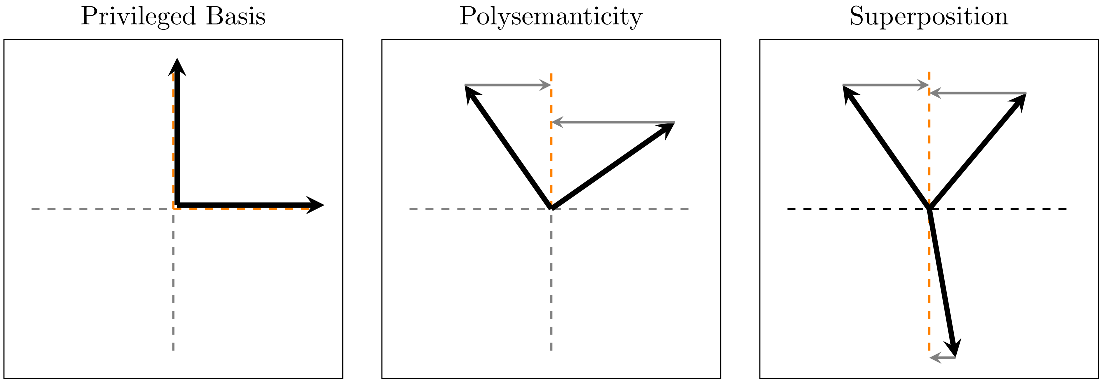

Figure 2.4: The three charts demonstrate the different ways a model may organise it's representation space. This figure is a reproduction of Elhage et al. [23] Figures 2 and 3. The privileged basis means that the representations are aligned with the architectures 'preferred' basis. Polysemanticity occurs when a specific neuron is activated by two, potentially unrelated, inputs. Finally, superposition occurs when the model has to embed more representations than privileged bases resulting in forced polysemanticity.

without access to future tokens. The process of hiding future tokens at a given token is called "attention masking" and is only applied in the attention blocks.

2.4.2 Instruction Tuning

2.5 Sparse Auto-Encoders

Though the processes to build, train and use a machine learning model are known, these processes and models themselves are not fully understood. One line of research that aims to understand how models work is "mechanistic interpretability". This is the field of reverse engineering how models work, converting structures in the model into human interpretable concepts and algorithms [22].

Olah et al. [24] found that in vision networks certain neurons are active[3] across a range of inputs. This idea is known as "polysemanticity" as the neuron represents multiple semantic meanings. This poses a problem for interpretability as it is not sufficient to assign meaning to specific neurons and check when they are active. This phenomenon has been shown to occur in LLMs and has been demonstrated in toy examples [23].

Elhage et al. [23] propose the idea of "superposition" to explain why large models contain polysemantic neurons. Superposition is the process of NNs representing more features than neurons within the model. The features are no longer represented orthogonally in the representation space but instead share components. INCLUDE FIGURE HERE. This means that if only one (sparse) feature is active the non-orthogonal features will also be partially active.

To disentangle the polysemantic neurons requires eliminating the superposition present in the model. This is a known problem known as "sparse dictionary encoding" [25] in neuroscience, in which a signal in superposition is decomposed into sparse elements. NEED BETTER UNDER-

[3] output a non-zero value.

STANDING/EXPLANATION. Sharkey et al. [26] and Cunningham et al. [27] apply the idea to NNs introducing the sparse autoencoder (SAE) which enforces sparsity in its internal representation.

INCLUDE A FIGURE. An SAE is an adaptor that takes a model layer's input and produces a replica of the layer's output. In comparison to the model layer the SAE has a large hidden representation dimension in which sparsity is enforced. This can be achieved in multiple ways such as clamping to the k highest activations [28] or adding a sparsity regularising loss. After training the elements of the SAE hidden dimension are given interpretations to better understand the model. It is worth noting that SAEs have been shown to demonstrate subpar performance when used for interpretability [29], though this project uses SAEs for dataset analysis.

SAEs are challenging to train and so for the purposes of this project only pretrained SAEs are used. Bloom et al. [30] provides a large collection of open source SAEs with their corresponding models. This does limit the analysis as most models only have an SAE for a single layer.

Chapter 3

Methodology

3.1 Steering Clear Environment

The setup of this environment follows [1]. The model to steer is a 4-layer multi-layer perceptron (MLP) with residual connections [31] across all layers. After the MLP, a layernorm [32] and single layer classifier is added. All non-linearity throughout the model is gaussian error linear unit (GeLU). The hidden layers follow 512-512-256-512 architecture regardless of dataset specifics.

3.1.1 Dataset

To control the behaviour of model and the steering approaches a synthetic dataset is used. Each dataset sample consists of m "attributes" which can take 8 possible discrete values. Each discrete value is represented by an "anchor" vector $\mu_i \in \mathbb{R}^8, i \in \{1, 8\}$ sampled from a gaussian distribution $\mathcal{N}(\mathbf{0}, 1)$. To simulate real-world conditions gaussian noise is added to the samples from $\mathcal{N}(\mathbf{0}, 0.1)$. This does mean the values are generally highly seperable.

The dataset comprises of n input-output vectors where the input vector is the concatenation of m 8-dimensional vectors. Thus, an input vector has length $8m$ and the target vector has length m. Krasheninnikov and Krueger [1] carry out a range of experiments for $m \in \{60, 90, 120\}$ but always use 8 values represented by 8 dimensional vectors. They take a sample of $2,000,000$ i.i.d samples but due to memory constraints only $500,000$ are used in this project. No test set is used in either however an 80:20 split for training and validation set is used for identifying the best performing model.

3.1.2 Pre-training

The MLP model is trained on the $500,000$ training samples for 50 epochs using Adam [33] with a learning rate of 0.001. As per Krasheninnikov and Krueger [1] a cross entropy loss is used to train the model. The model that achieves the best validation loss is saved and used for the steering task.

Regardless of exact epochs, learning rate or optimiser the best performing model should achieve close to 100%. Models used for the presented results achieved $\sim 99\%$.

3.1.3 Steering Task

The task is to successfully steer a model to always predict a specific value for a specific attribute. For example, the goal would be steer attribute 3 towards value μ_1. Krasheninnikov and Krueger [1] carry out three experiments to steer one, two or three attributes simultaneously. Instead, this reproduction will focus on steering only one attribute at a time.

As the attribute anchors are generated randomly there is no dataset bias towards any particular value. For this reason all attributes are steered towards value μ_1.

In addition to the dataset generate, 4096 are generated as a training set for the steering approaches and a further 1000 are generated as a test set. This is repeated 20 times to get an average metric across steering approaches.

For each adaptor a range of hyperparameters is used to analyse the effect on steering performance. The number of steering examples is also varied from 4 up to 4096 increasing in powers of 2. Krasheninnikov and Krueger [1] use this to analyse the representation densities effect of required number of examples.

Steering metric. As the model was trained to predict discrete attribute labels and the steering adaptor simply aims for a specific attribute value it is possible to use the models accuracy on the target attributes. Krasheninnikov and Krueger [1] use the full target output label, however, this was found to be dominated by unsteered attributes. Instead the accuracy on only the steered attribute is used.

3.2 Prompt Pairs Environment

Krasheninnikov and Krueger [1] aim to analyse the effects of model feature density and the number of steering examples. To achieve this the set up a toy environment with synthetic data and a small, controllable model. To analyse similar effects, as well as the effects of the steering dataset, in a situation closer to real-world use requires utilising real-world models.

As only LLMs are considered this means the dataset is made of natural language prompts. Positive and negative activations are sampled from the target layer and the last token. Generally, the two prompts used to extract positive and negative activations are identical except for the last token [6, 9**? ?**].

3.2.1 dataset

Rather than generate thousands of entirely unique pairs of prompts a smaller set of templates with adjustable “contexts” and “targets” is used. And example template would be:

```
Everyone thought <context> would lose.  In the end they <target>.
```

Then a range of relevant contexts (such as `the dancer` or `the driver`) and targets (such as `won` or `lost`) can be used. A standard prompt pair is therefore two prompts who's templates and contexts are identical but who's targets are in opposition. As the target is the last word activations can be extracted to steer the model from the negative target to the positive target.

Unlike the model written evaluations [7] dataset used in Tan et al. [6] this dataset does not use multiple choice questions and instead a small range of target values. This allows the model to produce more tokens than just "yes", "no" or "A", "B". This also allows for more control, such as changing context between positive and negative pairs or using "random" negative targets and meaningful positive targets. Note that all targets are *a single word* or ideally token to simplify the steering process.

Rather than a single dataset of this form multiple datasets are generated that cover a range of contexts and behaviours. They are aimed to be useful real-world situations however they are still fabricated and so are not a perfect representation. To generate the large number of templates, contexts and targets a set of example sentences were generated by GPT-5 [**?**] and adjusted to extract templates, contexts and targets.

The full list of templates, contexts and targets are presented in §**??**.

3.2.2 Steering Task

The goal is to steer the model from generating the negative targets to generating the positive targets. As the negative and positive targets have many potential options it is possible the model does not produce the exact same positive target word, however, getting the correct semantic meaning is the aim.

For each of the datasets, after activation extraction, 100 are separated for testing and the rest used for adaptor traing/initialisation. Similar to [1], a range of example pairs are used ranging from 4 example pairs to 1024 example pairs. The same range of adaptor hyperparameters are used to roughly compare the toy experiment to real-world scenarios. The experiments are also run without any adaptor to get a baseline value to compare from. MIGHT BE WORTH DOING INDIVIDUAL DATASETS. All steering metrics are averaged over the range of datasets to get an average efficacy across the datasets.

Steering metrics. Unlike the steering clear environment §3.1, it is hard to quantify accuracy on the steered attribute. Instead 3 metrics are used to evaluate the success of the steering approach.

Two are based on the SAE, §2.5, features of the final, target token of the model at the target layer:

1. During activation extraction, the SAE features that are consistently activated on the final token across all *positive* examples are identified. The average activation of these features during training is used to evaluate how well the steering adaptor is increasing the models representation of the target behaviour.

2. A random selection of SAE features that were consistently not activated for *both* examples during activation extraction are considered test features. The average activation of the test features during training is used to evaluate how well the adaptor effects only the target features.

Together these provide insight into how well the adaptor steers the internal model representations towards the intended behaviour.

To gather information about how well the adaptors effect the output the semantic similarity of the generated model token and the target model token is used. Using distilbert [**?**] as a semantic embedding of words the average cosine similarity between generated word and target word across the test examples is used. This provides a metric for the semantic similarity of the models generated text and the target text. Only the target word is used as otherwise the metric is dominated by the similarity of the pregenerated text.

Each metric on its own is useful but can be prone to biases that the other metrics highlight. A more complete picture of how the adaptors and models behaved can be achieved by analysing all three metrics together.

3.3 Prompting LLMs

Chapter 4

Results

Chapter 5

Conclusion

5.1 Limitations

5.2 Future Work

Bibliography

[1] Dmitrii Krasheninnikov and David Krueger. Steering clear: A systematic study of activation steering in a toy setup. In *MINT: Foundation Model Interventions*, 2024. 1, 7, 12, 13, 14

[2] Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15504–15522, 2024. 1, 4, 5, 6, 7, 8

[3] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D Manning, and Christopher Potts. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962, 2024. 1, 6, 7, 8

[4] Shashwat Singh, Shauli Ravfogel, Jonathan Herzig, Roee Aharoni, Ryan Cotterell, and Ponnurangam Kumaraguru. Representation surgery: theory and practice of affine steering. In *Proceedings of the 41st International Conference on Machine Learning*, pages 45663–45680, 2024. 1

[5] Thomas Marshall, Adam Scherlis, and Nora Belrose. Refusal in llms is an affine function. *CoRR*, 2024. 1, 5, 6, 7, 8

[6] Daniel Tan, David Chanin, Aengus Lynch, Dimitrios Kanoulas, Brooks Paige, Adria Garriga-Alonso, and Robert Kirk. Analyzing the generalization and reliability of steering vectors–icml 2024. *arXiv e-prints*, pages arXiv–2407, 2024. 1, 5, 13, 14

[7] Ethan Perez, Sam Ringer, Kamile Lukosiute, Karina Nguyen, Edwin Chen, Scott Heiner, Craig Pettit, Catherine Olsson, Sandipan Kundu, Saurav Kadavath, et al. Discovering language model behaviors with model-written evaluations. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13387–13434, 2023. 1, 14

[8] Jan Wehner, Sahar Abdelnabi, Daniel Tan, David Krueger, and Mario Fritz. Taxonomy, opportunities, and challenges of representation engineering for large language models. *arXiv preprint arXiv:2502.19649*, 2025. 2

[9] Alexander Matt Turner, Lisa Thiergart, David Udell, Gavin Leech, Ulisse Mini, and Monte

MacDiarmid. Activation addition: Steering language models without optimization. *CoRR*, 2023. 4, 13

[10] Joshua Engels, Isaac Liao, Eric J Michaud, Wes Gurnee, and Max Tegmark. Not all language model features are linear. In *2025 Joint Mathematics Meetings (JMM 2025)*. AMS, 2025. 5

[11] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. 6

[12] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024. 6

[13] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019. 6

[14] Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah Goodman. Finding alignments between interpretable causal variables and distributed neural representations. In *Causal Learning and Reasoning*, pages 160–187. PMLR, 2024. 7

[15] Latent Space. Commoditizing the petaflop - with george hotz of the tiny corp, 2023. URL https://www.latent.space/p/geohot. 8

[16] Gemma Team. Gemma 3, 2025. URL https://goo.gle/Gemma3Report. 8

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 8, 9

[18] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020. 8

[19] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33: 17283–17297, 2020.

[20] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. 8

[21] Richard E Turner. An introduction to transformers. *arXiv preprint arXiv:2304.10557*, 2023. 8

[22] Neel Nanda. A comprehensive mechanistic interpretability explainer & glossary, 2021. URL https://www.neelnanda.io/mechanistic-interpretability/glossary. 10

[23] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022. 10

[24] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020. 10

[25] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997. 10

[26] Lee Sharkey, Dan Braun, and Millidge Beren. [interim research report] taking features out of superposition with sparse autoencoders, 2022. URL https://www.alignmentforum.org/posts/z6QQJbtpkEAX3Aojj/interim-research-report-taking-features-out-of-superposition. 11

[27] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023. 11

[28] Alireza Makhzani and Brendan Frey. k-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013. 11

[29] Subhash Kantamneni, Joshua Engels, Senthooran Rajamanoharan, Max Tegmark, and Neel Nanda. Are sparse autoencoders useful? a case study in sparse probing. In *Forty-second International Conference on Machine Learning*, 2025. 11

[30] Joseph Bloom, Curt Tigges, Anthony Duong, and David Chanin. Saelens. https://github.com/jbloomAus/SAELens, 2024. 11

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 12

[32] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 12

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 12

Appendix A

Gumbel Softmax