able to bridge this gap by providing better metrics for measuring polysemanticity of the SAE neurons, which typically corresponds well with neuron interpretability.

Finally, we also provide a proof-of-concept demonstrating how we can incorporate neuron embeddings into the training of SAEs, by computing a measure of neuron monosemanticty and including it in the SAE loss. We show how this affects SAEs for a toy MLP trained on MNIST (Deng, 2012), decreasing the reconstruction accuracy and activation sparsity but increasing monosemanticity and significantly decreasing the prevalence of dead neurons which never activate.

2. Related Work

The favoured explanation for the cause of polysemanticity is superposition, which supposes that models learn to encode features across multiple neurons, allowing them to represent many more features than they have neurons. Superposition has been compellingly demonstrated in toy models (Elhage et al., 2022b) and large language models (Gurnee et al., 2023).

One thread of research aimed to tackle superposition and polysemanticity by introducing a new activation function, called a Softmax Linear Unit (SoLU) (Elhage et al., 2022a), which encourages neurons to activate sparsely. However, they found that this resulted in polysemanticity effectively being pushed into other neurons and the Layer Norm component, rather than truly eliminated.

Dictionary learning is an alternative and very promising method for interpreting language models in spite of superposition and polysemanticity. It aims to decompose a model's internal embeddings or MLP layers into features, by training a Sparse Auto-Encoder (SAE) layer to reconstruct their activations. SAEs have been explored by a number of groups (Bricken et al., 2023; Cunningham et al., 2023; Yun et al., 2023), most notably in a recent paper that applied them to the residual stream of a very large production language model and found a rich array of interpretable features (Templeton et al., 2024).

3. Method

3.1. Neuron Embeddings

Given a model containing l (potentially non-contiguous) MLP layers, we denote the ith layer as L_i, and the jth neuron in L_i as $N_{i,j}$, which has input weights $\mathbf{w}_{i,j}$. For a given model input $\mathbf{x}$, we denote the internal vector representation of $\mathbf{x}$ immediately before layer L_i as $\mathbf{h}_{i-1}$, which we refer to as the pre-MLP embedding.

We then define the neuron embedding of $\mathbf{x}$ for the neuron $N_{i,j}$ as $\mathbf{e}_{i,j}$, where $\mathbf{e}_{i,j}$ is the Hadamard product of the pre-MLP embedding $\mathbf{h}_{i-1}$ and the input weights of the neuron $\mathbf{w}_{i,j}$:

$$\mathbf{e}_{i,j} = \mathbf{h}_{i-1} \odot \mathbf{w}_{i,j} \tag{1}$$

This is exactly the first stage of computing the neuron's activation - summing $\mathbf{e}_{i,j}$ and applying the activation function would complete the process. Constraining ourselves to use a representation that is computed internally to the model reduces the risk of introducing additional structure which may not be true to what the model is actually doing. It also means neuron embeddings can be computed for any MLP neurons, regardless of the rest of the model architecture (CNNs, Transformers, etc.) or the domain (vision, language, etc.).

Intuitively, the input weights of a neuron represent what the neuron is "looking for" in an input - with different parts of the weights potentially looking for different features. The neuron embedding then, in some sense, represents the feature that the neuron found in the input that caused it to activate. If this does indeed produce a good representation of the feature which caused activation, we should then be able to use it to separate the mixture of dataset examples into their distinct semantic behaviours. Figure 1 illustrates this process.

3.2. Feature Clusters

We now apply neuron embeddings to tackle the problem of polysemantic neurons by creating **feature clusters**. A feature cluster is a set of highly activating dataset examples that capture a single semantic behaviour of a neuron, created by clustering the neuron embeddings of a neuron's dataset examples. We demonstrate how feature clusters can be computed for autoregressive GPT-style language models, but note that this technique can be applied to any model containing MLP layers.

Extending the above notation, we define the kth dataset example for a neuron $N_{i,j}$ as $\mathbf{x}_{i,j,k}$, which consists of a sequence of tokens $t = t_1 \cdots t_n$ where $N_{i,j}$ strongly activates on the last token t_n. The neuron embedding $\mathbf{e}_{i,j,k}$ of the dataset example $\mathbf{x}_{i,j,k}$ for neuron $N_{i,j}$ is therefore:

$$\mathbf{e}_{i,j,k} = \mathbf{h}_{i-1,k,n} \odot \mathbf{w}_{i,j}, \tag{2}$$

where $\mathbf{h}_{i-1,k,n}$ is the pre-MLP embedding of the activating token t_n.

Given a set of highly activating dataset examples for a target neuron, we compute the neuron embeddings for each example. We then compute the pairwise distance matrix between all pairs of embeddings, where the distance d between dataset examples k_a and k_b is:

$$d = 1 - S_C(\mathbf{e}_{i,j,k_a}, \mathbf{e}_{i,j,k_b}), \tag{3}$$