CHAPTER

10 Interpretability

"Everything that we see is a shadow cast by that which we do not see"
Martin Luther King, Jr., Sermon at the Detroit Council of Churches, 1961

interpretability

In this (currently stub) chapter we turn to LLM **interpretability**, sometimes called **mechanistic interpretability**. Here researchers search for methods that will help us understand how language models do what they do. Much of the work focuses on uncovering the model's internal representations what kind of meanings they seem to encode, as well as the models paths for reasoning, whether we can describe specific "circuits".

10.1 Contextual Embeddings

Let's begin by extending the fundamental idea of embeddings that we introduce in Chapter 5 to see the corresponding idea in transformers and other LLM architectures. The methods of Chapter 5 like word2vec or GloVe learned a single vector embedding for each unique word w in the vocabulary. By contrast, with contextual embeddings, each word w will be represented by a different vector each time it appears in a different context. Contextual embeddings are used in both the causal language models of Chapter 7 and in the encoder models of Chapter 9, although the embeddings created by masked language models seem to function particularly well as representations, so we will use those in examples.

contextual embeddings

When a text is fed to a pretrained language model, the output of each component of the model (the attention heads, the feedforward network) is a representation that gets added into the residual stream. Each of these representations, which occur at every level and for every token, are called **contextual embeddings**. As we suggested, one way to think of contextual embeddings is as vectors that encode some aspect of the meaning of a token in context, like a contextualized version of the static embeddings of Chapter 5. As such, we can use these embeddings for tasks involving the meaning of tokens or words.

For example, given a sequence of input tokens $x_1, \ldots, x_n$, we can use the output vector $\mathbf{h^L}_i$ from the final layer L of a model aes a representation of the meaning of token x_i in the context of sentence $x_1, \ldots, x_n$, as shown in Fig. 10.1.

For BERT models, instead of just using the vector $\mathbf{h^L}_i$ from the final layer of the model, it's common to compute a representation for x_i by mean pooling or max pooling the output tokens $\mathbf{h}_i$ from each of the last four layers of the model, i.e., $\mathbf{h^L}_i$, $\mathbf{h^{L-1}}_i$, $\mathbf{h^{L-2}}_i$, and $\mathbf{h^{L-3}}_i$.

Just as we used static embeddings like word2vec in Chapter 5 to represent the meaning of words, we can use contextual embeddings as representations of word meanings in context for any task that might require a model of word meaning. Where

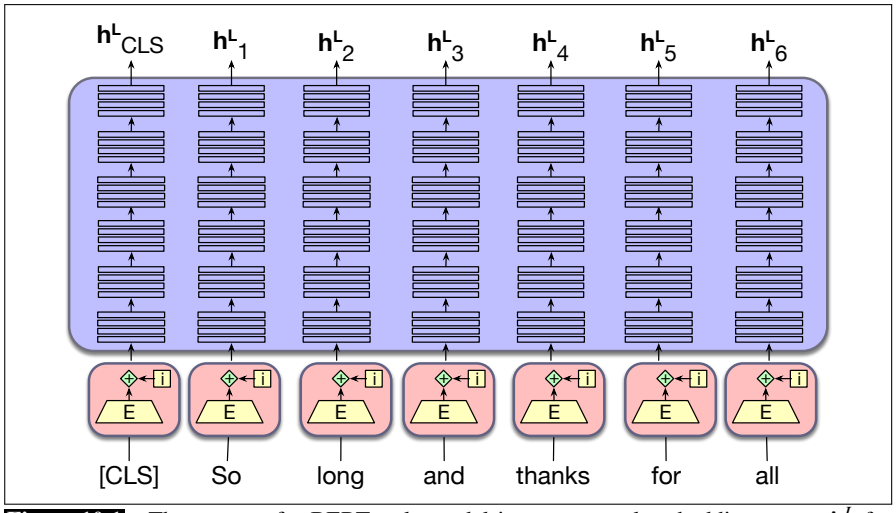

Figure 10.1 The output of a BERT-style model is a contextual embedding vector $\mathbf{h}_i^L$ for each input token x_i.

static embeddings represent the meaning of word *types* (vocabulary entries), contextual embeddings represent the meaning of word *instances*: instances of a particular word type in a particular context. Thus where word2vec had a single vector for each word type, contextual embeddings provide a single vector for each instance of that word type in its sentential context. Contextual embeddings can thus be used for tasks like measuring the semantic similarity of two words in context, and are useful in linguistic tasks that require models of word meaning.

10.1.1 Contextual Embeddings and Word Sense

ambiguous

Words are **ambiguous**: the same word can be used to mean different things. In Chapter 5 we saw that the word "mouse" can mean (1) a small rodent, or (2) a hand-operated device to control a cursor. The word "bank" can mean: (1) a financial institution or (2) a sloping mound. We say that the words 'mouse' or 'bank' are **polysemous** (from Greek 'many senses', *poly-* 'many' + *sema*, 'sign, mark').[1]

word sense

A **sense** (or **word sense**) is a discrete representation of one aspect of the meaning of a word. We can represent each sense with a superscript: **bank**^{1} and **bank**^{2}, **mouse**^{1} and **mouse**^{2}. These senses can be found listed in online thesauruses (or thesauri) like **WordNet** (Fellbaum, 1998), which has datasets in many languages listing the senses of many words. In context, it's easy to see the different meanings:

WordNet

mouse^1 : .... a *mouse* controlling a computer system in 1968.
mouse^2 : .... a quiet animal like a *mouse*
bank^1 : ...a *bank* can hold the investments in a custodial account ...
bank^2 : ...as agriculture burgeons on the east *bank*, the river ...

This fact that context disambiguates the senses of *mouse* and *bank* above can also be visualized geometrically. Fig. 10.2 shows a two-dimensional projection of many instances of the BERT embeddings of the word *die* in English and German. Each point in the graph represents the use of *die* in one input sentence. We can

[1] The word **polysemy** itself is ambiguous; you may see it used in a different way, to refer only to cases where a word's senses are related in some structured way, reserving the word **homonymy** to mean sense ambiguities with no relation between the senses (Haber and Poesio, 2020). Here we will use 'polysemy' to mean any kind of sense ambiguity, and 'structured polysemy' for polysemy with sense relations.

clearly see at least two different English senses of *die* (the singular of *dice* and the verb *to die*, as well as the German article, in the BERT embedding space.

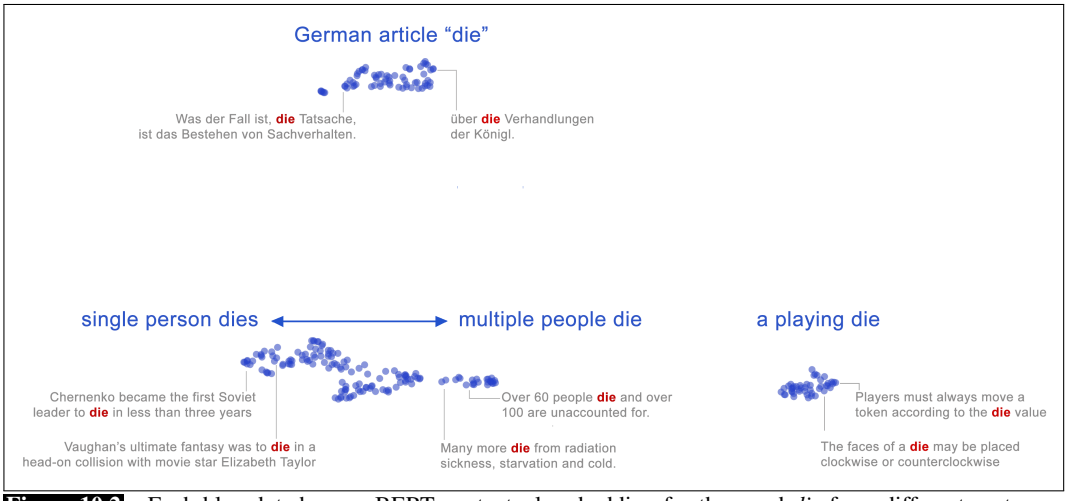

Figure 10.2 Each blue dot shows a BERT contextual embedding for the word *die* from different sentences in English and German, projected into two dimensions with the UMAP algorithm. The German and English meanings and the different English senses fall into different clusters. Some sample points are shown with the contextual sentence they came from. Figure from Coenen et al. (2019).

Thus while thesauruses like WordNet give discrete lists of senses, embeddings (whether static or contextual) offer a continuous high-dimensional model of meaning that, although it can be clustered, doesn't divide up into fully discrete senses.

Word Sense Disambiguation

word sense disambiguation
WSD

The task of selecting the correct sense for a word is called **word sense disambiguation**, or **WSD**. WSD algorithms take as input a word in context and a fixed inventory of potential word senses (like the ones in WordNet) and outputs the correct word sense in context. Fig. 10.3 sketches out the task.

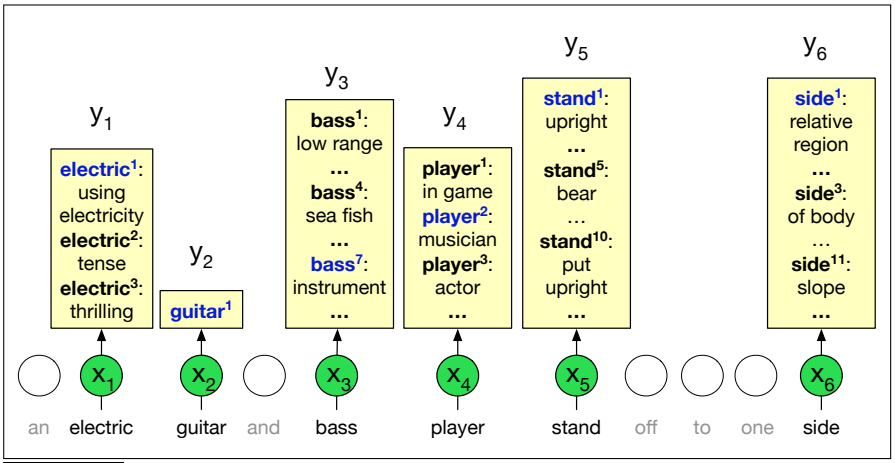

Figure 10.3 The all-words WSD task, mapping from input words (x) to WordNet senses (y). Figure inspired by Chaplot and Salakhutdinov (2018).

WSD can be a useful analytic tool for text analysis in the humanities and social

sciences, and word senses can play a role in model interpretability for word representations. Word senses also have interesting distributional properties. For example a word often is used in roughly the same sense through a discourse, an observation called the **one sense per discourse** rule (Gale et al., 1992a).

one sense per discourse

The best performing WSD algorithm is a simple 1-nearest-neighbor algorithm using contextual word embeddings, due to Melamud et al. (2016) and Peters et al. (2018). At training time we pass each sentence in some sense-labeled dataset (like the SemCore or SenseEval datasets in various languages) through any contextual embedding (e.g., BERT) resulting in a contextual embedding for each labeled token. (There are various ways to compute this contextual embedding v_i for a token i; for BERT it is common to pool multiple layers by summing the vector representations of i from the last four BERT layers). Then for each sense s of any word in the corpus, for each of the n tokens of that sense, we average their n contextual representations v_i to produce a contextual **sense embedding** $\mathbf{v}_s$ for s:

$$\mathbf{v}_s = \frac{1}{n}\sum_i \mathbf{v}_i \qquad \forall \mathbf{v}_i \in \text{tokens}(s) \tag{10.1}$$

At test time, given a token of a target word t in context, we compute its contextual embedding $\mathbf{t}$ and choose its nearest neighbor sense from the training set, i.e., the sense whose sense embedding has the highest cosine with $\mathbf{t}$:

$$\text{sense}(t) = \underset{s \in \text{senses}(t)}{\text{argmax}} \ \text{cosine}(\mathbf{t}, \mathbf{v}_s) \tag{10.2}$$

Fig. 10.4 illustrates the model.

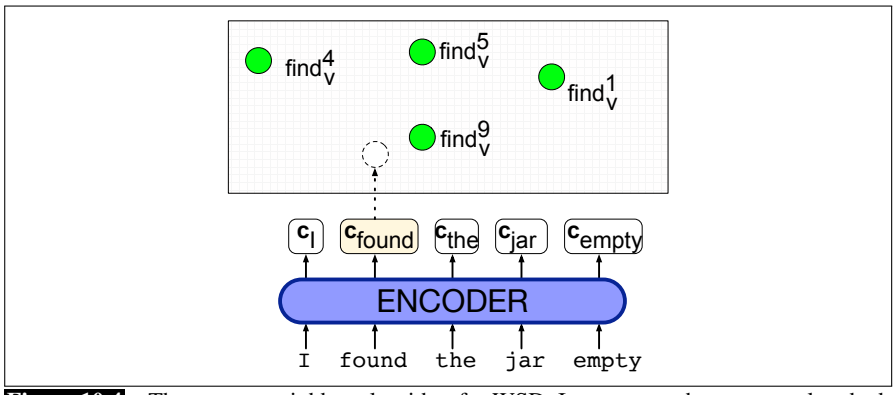

Figure 10.4 The nearest-neighbor algorithm for WSD. In green are the contextual embeddings precomputed for each sense of each word; here we just show a few of the senses for *find*. A contextual embedding is computed for the target word *found*, and then the nearest neighbor sense (in this case $\textbf{find}^9_v$) is chosen. Figure inspired by Loureiro and Jorge (2019).

10.1.2 Contextual Embeddings and Word Similarity

In Chapter 5 we introduced the idea that we could measure the similarity of two words by considering how close they are geometrically, by using the cosine as a similarity function. The idea of meaning similarity is also clear geometrically in the meaning clusters in Fig. 10.2; the representation of a word which has a particular sense in a context is closer to other instances of the same sense of the word. Thus we often measure the similarity between two instances of two words in context (or two

instances of the same word in two different contexts) by using the cosine between their contextual embeddings.

Usually some transformations to the embeddings are required before computing cosine. This is because contextual embeddings (whether from masked language models or from autoregressive ones) have the property that the vectors for all words are extremely similar. If we look at the embeddings from the final layer of BERT or other models, embeddings for instances of any two randomly chosen words will have extremely high cosines that can be quite close to 1, meaning all word vectors tend to point in the same direction. The property of vectors in a system all tending to point in the same direction is known as **anisotropy**. Ethayarajh (2019) defines the **anisotropy** of a model as the expected cosine similarity of any pair of words in a corpus. The word 'isotropy' means uniformity in all directions, so in an isotropic model, the collection of vectors should point in all directions and the expected cosine between a pair of random embeddings would be zero. Timkey and van Schijndel (2021) show that one cause of anisotropy is that cosine measures are dominated by a small number of dimensions of the contextual embedding whose values are very different than the others: these **rogue dimensions** have very large magnitudes and very high variance.

anisotropy

Timkey and van Schijndel (2021) shows that we can make the embeddings more isotropic by standardizing (z-scoring) the vectors, i.e., subtracting the mean and dividing by the variance. Given a set C of all the embeddings in some corpus, each with dimensionality d (i.e., $x \in \mathbb{R}^d$), the mean vector $\mu \in \mathbb{R}^d$ is:

$$\mu = \frac{1}{|C|}\sum_{\mathbf{x}\in C}\mathbf{x} \tag{10.3}$$

The standard deviation in each dimension $\sigma \in \mathbb{R}^d$ is:

$$\sigma = \sqrt{\frac{1}{|C|}\sum_{\mathbf{x}\in C}(\mathbf{x}-\mu)^2} \tag{10.4}$$

Then each word vector $\mathbf{x}$ is replaced by a standardized version $\mathbf{z}$:

$$\mathbf{z} = \frac{\mathbf{x}-\mu}{\sigma} \tag{10.5}$$

One problem with cosine that is not solved by standardization is that cosine tends to underestimate human judgments on similarity of word meaning for very frequent words (Zhou et al., 2022).

10.2 Probing

Often we have want to know whether a particular representation in the network encodes some particular information. For example we might want to know how the network processing linguistic information about grammar or meaning, such as whether at a certain layer the network's representation of the word *ketchup* encodes the fact that it is a noun and not a verb. Or whether the representation of the word *sublet* (which can be past or present tense) encodes the tense in a particular sentence, and if so at what layer of the network this happens.

probing

The simplest technique for this is called **probing**. In probing, we build a clas-

sifier, which can be logistic or linear regression or a small MLP which takes an embedding vector as input and returns the class of interest (say 'noun', or 'verb'). If the probe has a high probability, say, that the word is a noun, (as compared with some control conditions), then we say that the representation probably encodes the concept of noun. Fig. 10.5 shows the intuition, showing 3 potential probes looking at two different embedding vectors inside a BERT representation.

Probing has been used to study the way information propagates up from lower to higher layers in the transformer, showing for example that grammatical information about tokens seems to be encoded at lower layers, semantic information at middle layers, and discourse information like coreference still higher (Tenney et al., 2019).

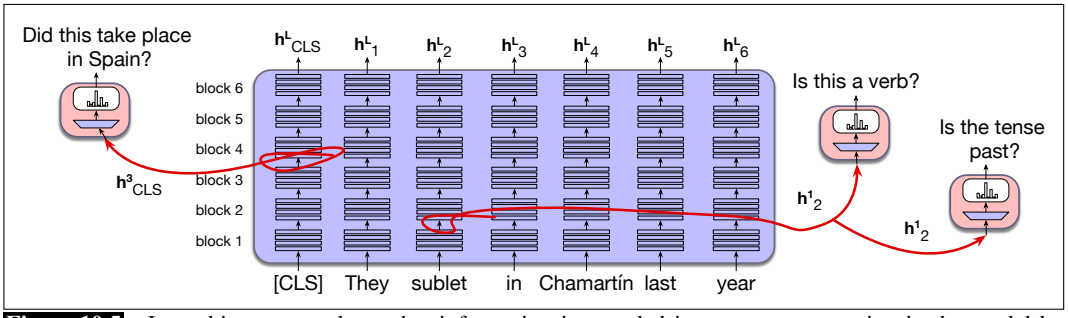

Figure 10.5 In probing, we explore what information is encoded in some representation in the model by training simple classifiers to extract linguistic or other properties of tokens (like being a verb or being in the past tense) or sentences (like taking place in Spain). Here we see 3 different probes examining 2 representations, $\mathbf{h}^3_{\text{CLS}}$ and $\mathbf{h}^1_2$.

Probing has two phases, training and testing. Training a probe has 4 steps:

1. Choose a property we want to examine (like part of speech, or some aspect of meaning, or which entity a pronoun corefers with)
2. Create a training and test set of embeddings at some position and level we are interested in studying (like "the output of the feedforward network at transformer layers 2 through 16"), labeled with the correct answer (the relevant part of speech, or tense, or whatever).
3. Pass the training set sentences through the LLM, and extract activations
4. Now freeze the weights of the LLM, and train only the weights of the classifier to map from an embedding to a property.

At inference times we pass a test set text through the model, extract activations, and pass them through the classifier, and report on whether the model is able to correctly extract the property.

Probing generally requires comparing to control conditions, like making sure that performance is better than probing with random labels (Belinkov et al., 2017). And it's important that the probe not be too powerful, to keep it from succeeding because the relevant information is encoded in the probe weights themselves rather than in the representation (Hewitt and Liang, 2019).

While success at probing shows that information is encoded in a representation, it doesn't prove that the model uses the information. That is, the information being probed could be **correlated** with what's in the embedding, but not have a **causal** effect on the model's behavior (Belinkov, 2022).

10.3 The Logit Lens

logit lens

Another useful interpretability tool, the **logit lens** (Nostalgebraist, 2020), offers a way to visualize what the internal layers of the transformer might be representing.

The idea is that we take any vector from any layer of the transformer and, pretending that it is the prefinal embedding, simply multiply it by the **unembedding layer** to get logits, and compute a softmax to see the distribution over words that that vector might be representing. This can be a useful window into the internal representations of the model. Since the network wasn't trained to make the internal representations function in this way, the logit lens doesn't always work perfectly, but this can still be a useful trick to help us visualize the internal layers of a transformer.

J-lens

The Jacobian lens or **J-lens** (Gurnee et al., 2026) is a variant of the logit lens that seems to perform better at uncovering what the model is encoding at earlier layers. The logit lens interprets all layers the same way, by multiplying an embedding at that layer by the same unencoding matrix. It thus implicitly makes the assumption that representations at earlier layers are in the same embedding space as the final layer. The J-lens instead looks at the derivatives of the final layer with respect to the intermediate layers, asking how a small change in layers would affect the final layer output representation. It thus effectively is passing embeddings at earlier layers through an approximation of the rest of the network, and also averages over many contexts.

The logit lens treats the intermediate layers of a transformer as if they already speak the "language" of the final output. The Jacobian lens acts as a translator, mapping how a small push in an early layer physically alters the final output tokens .

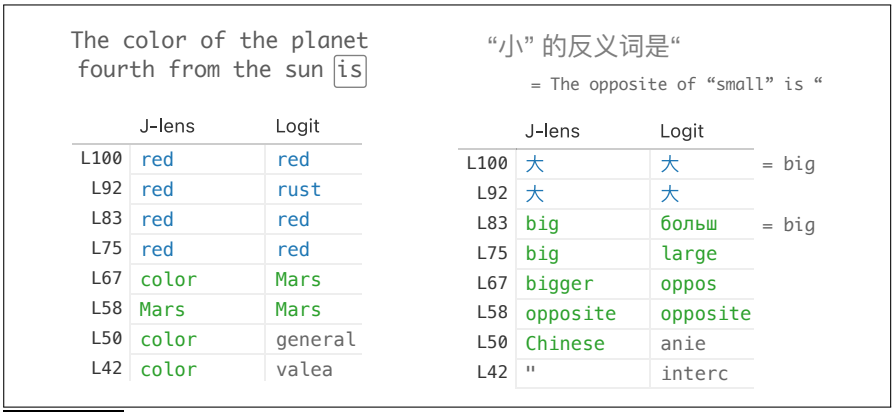

Figure 10.6 The most likely words at different layers of the model for 2 different prefixes. The left example shows the ability of the model to do multihop reasoning, particularly successful in the J-lens, where the model first encodes the idea of color, then higher layers seem to resolve the "fourth planet" reference to refer to Mars, and only later starting encoding red. The right model both the logit lens and the J-lens seem to encode the concept of opposites, and only at higher layers represent the correct opposite *large* first in English, then in Russian, and finally in the correct Chinese. Image from Figure 51 of (Gurnee et al., 2026).

The J-lens uncovers what seems to be a kind of workspace where the model reasons through problems, layer by layer. Fig. 10.6 shows an intuition of this for the J-lens and logit lens at different layers of the Anthropic Claude Sonnet 4.5 model. The left example shows the model using layers to reason successively through a

multi-step reasoning question about color that requires the model to first identify the fourth planet. And the right example shows that the model seems to represent these meanings mostly in English, and then translate them into relevant languages for output only in later layers

10.4 Sparse Auto-Encoders

10.5 Causal Interpretability Techniques

10.6 In-Context Learning and Induction Heads

As a way of getting a model to do what we want, we can think of prompting as being fundamentally different than pretraining. Learning via pretraining means updating the model's parameters by using gradient descent according to some loss function. But prompting with demonstrations can teach a model to do a new task. The model is learning something about the task from those demonstrations as it processes the prompt.

Even without demonstrations, we can think of the process of prompting as a kind of learning. For example, the further a model gets in a prompt, the better it tends to get at predicting the upcoming tokens. The information in the context is helping give the model more predictive power.

in-context learning

The term **in-context learning** was first proposed by Brown et al. (2020) in their introduction of the GPT3 system, to refer to either of these kinds of learning that language models do from their prompts. In-context learning means language models learning to do new tasks, better predict tokens, or generally reduce their loss during the forward-pass at inference-time, without any gradient-based updates to the model's parameters.

induction heads

How does in-context learning work? While we don't know for sure, there are some intriguing ideas. One hypothesis is based on the idea of **induction heads** (Elhage et al., 2021; Olsson et al., 2022). Induction heads are the name for a **circuit**, which is a kind of abstract component of a network. The induction head circuit is part of the attention computation in transformers, discovered by looking at mini language models with only 1-2 attention heads.

The function of the induction head is to predict repeated sequences. For example if it sees the pattern `AB...A` in an input sequence, it predicts that `B` will follow, instantiating the **pattern completion** rule `AB...A`$\rightarrow$ `B`. It does this by having a *prefix matching* component of the attention computation that, when looking at the current token A, searches back over the context to find a prior instance of A. If it finds one, the induction head has a *copying* mechanism that "copies" the token B that followed the earlier A, by increasing the probability the B will occur next. Fig. 10.7 shows an example.

Olsson et al. (2022) propose that a generalized fuzzy version of this pattern completion rule, implementing a rule like `A*B*...A`$\rightarrow$ `B`, where A* $\approx$ A and B* $\approx$ B (by $\approx$ we mean they are semantically similar in some way), might be responsible for in-context learning. Suggestive evidence for their hypothesis comes from Cros-

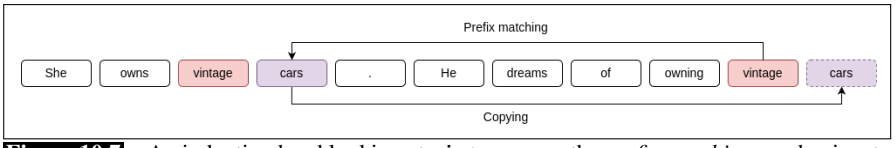

Figure 10.7 An induction head looking at `vintage` uses the *prefix matching* mechanism to find a prior instance of `vintage`, and the *copying* mechanism to predict that `cars` will occur again. Figure from Crosbie and Shutova (2022).

ablating

bie and Shutova (2022), who show that **ablating** induction heads causes in-context learning performance to decrease. **Ablation** is originally a medical term meaning the removal of something. We use it in NLP interpretability studies as a tool for testing causal effects; if we knock out a hypothesized cause, we would expect the effect to disappear. Crosbie and Shutova (2022) ablate induction heads by first finding attention heads that perform as induction heads on random input sequences, and then zeroing out the output of these heads by setting certain terms of the output matrix $\mathbf{W^O}$ to zero. Indeed they find that ablated models are much worse at in-context learning: they have much worse performance at learning from demonstrations in the prompts.