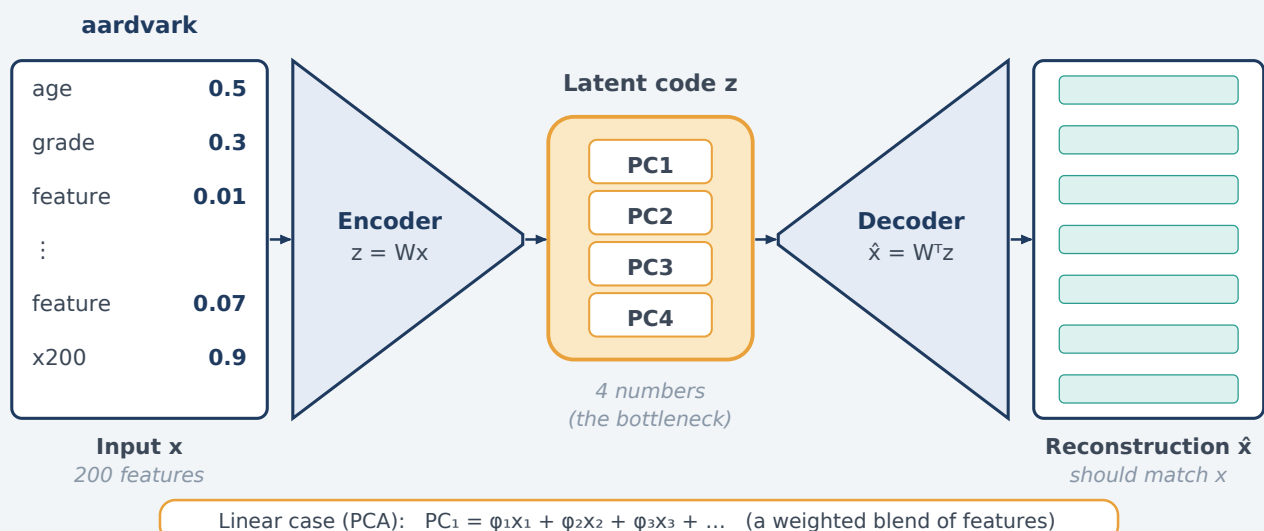


From Unsupervised Learning to LLM Embeddings

How hidden dimensions encode meaning, from PCA and autoencoders to the representation of “aardvark” inside an LLM

A Unsupervised learning as compression (PCA / autoencoder)



B From a token to a hidden vector inside an LLM

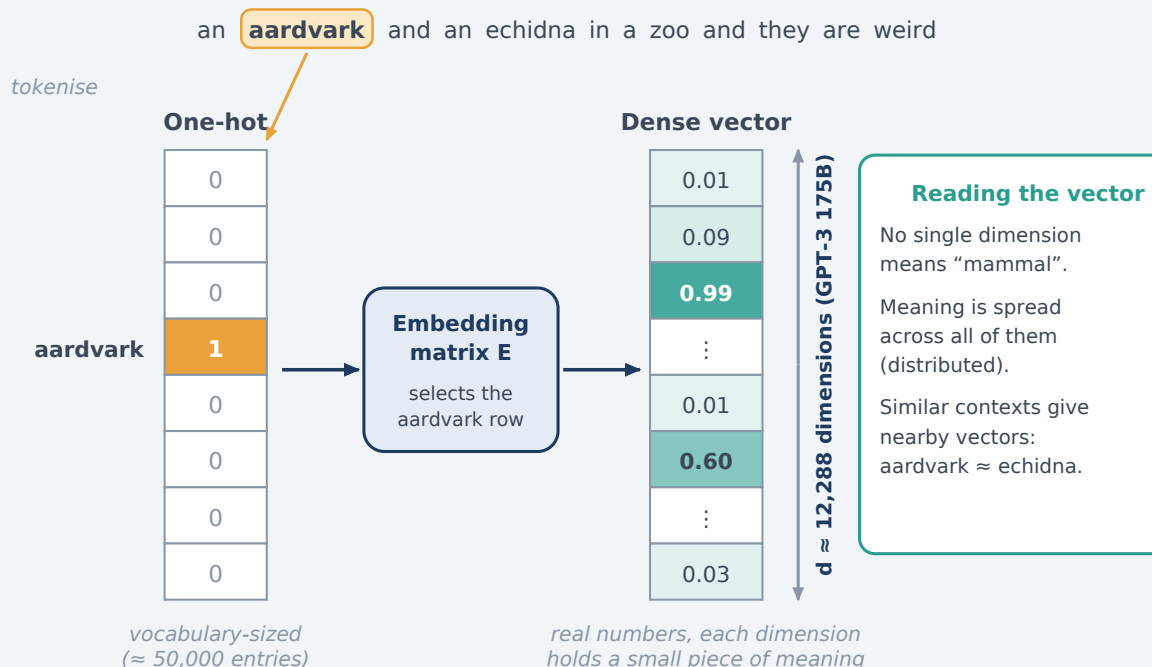


Figure 1. (A) An encoder, bottleneck and decoder compress 200 features into a few latent coordinates; in the linear case this is PCA. (B) An LLM does the reverse: it expands a sparse one-hot identifier into a long dense vector. Values shown are illustrative, not real embedding values.

1. Overview

This session connects two ideas that are usually taught separately:

- 1 **Unsupervised learning** finds compact, useful structure in data without labels, for example by compressing many features into a few components.
- 2 **Large language models** turn discrete tokens (such as the word “aardvark”) into dense vectors whose dimensions jointly encode meaning.

The central message: *a hidden representation is a learned compression*. A word is not stored as a symbol with a definition. It is stored as a position in a high-dimensional space, shaped by the patterns the model has seen.

Learning objectives. By the end of this session students should be able to:

- Explain compression of many features into a few components (PCA / linear autoencoder).
- Write a principal component as a linear combination of the input features.
- Contrast a one-hot vector with a dense embedding vector.
- Describe how an LLM maps a token to a high-dimensional hidden vector, and why individual dimensions are rarely human-readable.

2. Unsupervised learning as compression

Suppose each item (an animal, a person, a word) is described by a long list of numeric features, around 200 in our running example. Many are redundant or correlated. Unsupervised learning asks: **can we describe each item with far fewer numbers while losing as little as possible?**

Panel A of Figure 1 shows the pipeline:

- **Input layer.** The raw feature vector, for example age 0.5, grade 0.3, and so on, for “aardvark” or “echidna”.
- **Encoder.** Maps the 200 inputs down to a small number of values.
- **Bottleneck (latent space).** A handful of coordinates, here PC1 to PC4. This is the compressed code.
- **Decoder.** Reconstructs the original features from the code.
- **Training signal.** The reconstruction should match the input. No labels are needed, which is what makes it unsupervised.

2.1 The linear case: PCA

When the encoder and decoder are **linear**, this setup recovers the same subspace as Principal Component Analysis. Each principal component is a weighted sum of the original features:

$$PC_1 = \phi_1 x_1 + \phi_2 x_2 + \phi_3 x_3 + \dots$$

The x_i are input features and the ϕ_i are learned weights (loadings). In matrix form: $\mathbf{z} = \mathbf{W}\mathbf{x}$ to encode and $\hat{\mathbf{x}} = \mathbf{W}^T \mathbf{z}$ to decode.

- Each component is a *direction* in feature space, chosen to capture as much variance as possible.
- Components are ordered by importance: PC_1 explains the most variance, PC_2 the next, and so on.
- A component is usually a blend of many features, so it is rarely a clean, named concept.

2.2 The non-linear case

Replace the linear maps with neural network layers and non-linear activations and you get a general **autoencoder**. It can capture curved structure that PCA cannot. The principle is unchanged: squeeze the data through a bottleneck and force the network to keep what matters.

3. Two ways to represent a word

3.1 One-hot encoding

A vector with one entry per vocabulary item, all zeros except a single 1 at the word's index. In Figure 1B, "armadillo" is the position holding the 1.

- Very sparse, and as long as the vocabulary.
- Every word is exactly as far from every other word. "armadillo" is no closer to "echidna" than to "tractor".
- It says nothing about meaning. It is only an identifier.

3.2 Dense embedding

An LLM maps that identifier to a **dense vector** of real numbers. The vector length is the model's hidden dimension, labelled GPT-3 in the sketch.

- Multiplying a one-hot vector by the embedding matrix simply **selects one row**. The row for "armadillo" is its embedding.
- Each of the thousands of dimensions holds one number. No single dimension means "mammal" or "nocturnal". Meaning is spread across many dimensions at once (a **distributed representation**).
- Words used in similar contexts end up with nearby vectors, so "armadillo" and "echidna" sit closer to each other than either does to "mortgage".

4. The bridge: what is the same, what is different

	PCA / autoencoder	LLM embedding
Input	Numeric feature vector (about 200 features)	One-hot token (vocabulary-sized)
Goes to	A few latent coordinates (PC1 to PC4)	A very wide hidden vector (thousands of dimensions)
Direction	Compress	Expand a sparse symbol into a rich vector
Mapping	Linear (PCA) or non-linear (autoencoder)	Lookup, then many non-linear transformer layers
Training signal	Reconstruct the input	Predict the next token (self-supervised)
Labels needed?	No	No human labels; the text supplies its own targets
Interpretability	Components are weighted blends of features	Dimensions are distributed and rarely individually readable

The shared idea: in both cases the model learns coordinates that are useful for its task, and those coordinates are linear or non-linear combinations of what went in. An LLM is not performing PCA, but the intuition of "meaning as a position in a learned space" carries over directly.

5. Worked example

"An armadillo and an echidna in a zoo, and they are weird."

- 1 **Tokenise.** The sentence is split into tokens, each with an index in the vocabulary.

- 2 **One-hot.** Each token is conceptually a vector of zeros with a 1 at its index.
- 3 **Embed.** Each token is replaced by its dense embedding. “Aardvark” becomes a long vector of real numbers.
- 4 **Contextualise.** Transformer layers repeatedly mix information between tokens. By the later layers, the vector for “weird” carries information about what it refers to, and “aardvark” and “echidna” are informed by “zoo”.
- 5 **Read out.** The final hidden state is projected back to vocabulary-sized scores to predict the next token.

Note the symmetry: the model goes **sparse, then dense, then back to sparse**, with the dense hidden space in the middle doing the work.

6. Suggested classroom activities

- **Hands-on PCA.** Run PCA on a small animal-feature dataset, plot PC_1 against PC_2 and see which animals cluster. Ask what PC_1 “means” and why it is hard to name.
- **Autoencoder vs PCA.** Train a linear autoencoder and compare its subspace to PCA’s. Then add a non-linearity and compare reconstruction error.
- **Embedding neighbours.** Using a pre-trained embedding model, find the nearest neighbours of “aardvark”. Why does “echidna” appear, and why are the neighbours not simply synonyms?
- **Dimension hunt.** Pick a single embedding dimension and look at which words score highest. Is it interpretable? Why or why not?

7. Discussion questions

- 1 Why can a one-hot vector never express that two animals are similar, while a dense vector can?
- 2 If a principal component is a weighted sum of many features, what would it take for it to correspond to a single human concept?
- 3 What is lost when we compress 200 features to 4, and how would we measure it?
- 4 Why might a model need thousands of dimensions to represent a word when PCA can often summarise data in a handful?
- 5 The training signal for PCA is reconstruction and for an LLM is next-token prediction. How does each shape what the latent space encodes?

8. Notes on the source sketch

- The handwritten feature labels on the left (age, grade and a third word) are read as illustrative placeholders. Please check them against your intent.
- The sketch labels GPT-3’s hidden vector as 16,000 dimensions. As far as I know, the largest GPT-3 model uses a hidden size of 12,288 (with a vocabulary of roughly 50,000 tokens), so the polished figure uses 12,288. Adjust if you prefer the original number.
- All vector values in the figures (0.01, 0.09, 0.99 and so on) are illustrative, not real embedding values.