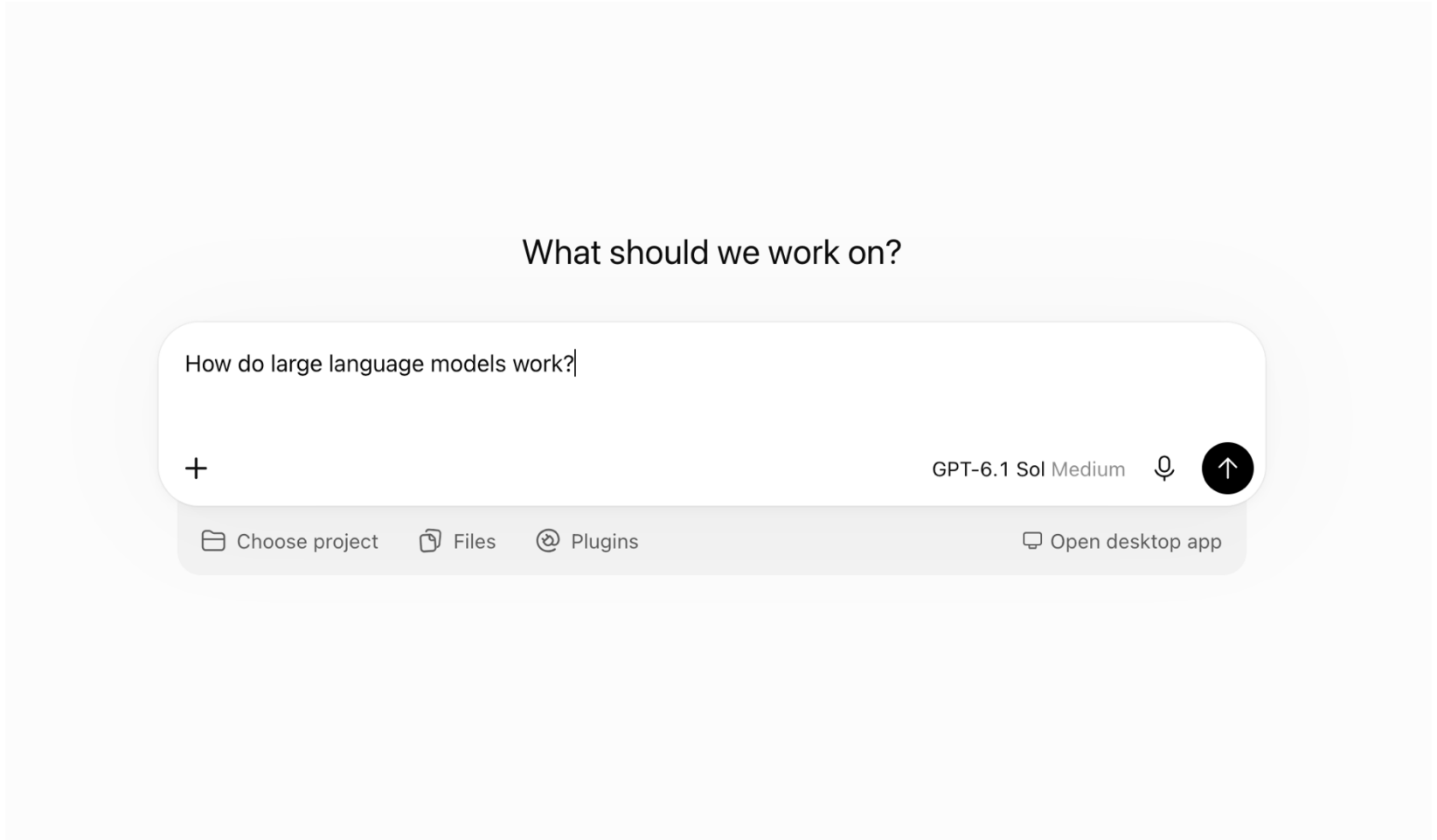
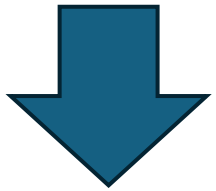
Intro To Language models


What should we work on?
How do large language models work?
GPT-6.1 Sol Medium
Choose project
Files
Plugins
Open desktop app

Our goal: Get to this



How do large language models work?

Large language models (LLMs) are AI systems that learn to understand and generate human language by studying enormous amounts of text and learning patterns in how words, sentences, ideas, and concepts relate to one another.

The fascinating part is that they don't work like traditional computer programs. Rather than following a fixed set of instructions, they learn statistical patterns from data and use those patterns to generate responses.

Let's break it down.

1. The basic idea: predicting the next word

At their core, most large language models do something surprisingly simple:

They predict what token (piece of a word) should come next.

For example, imagine you type:

> The capital of France is...

The model might calculate probabilities like:

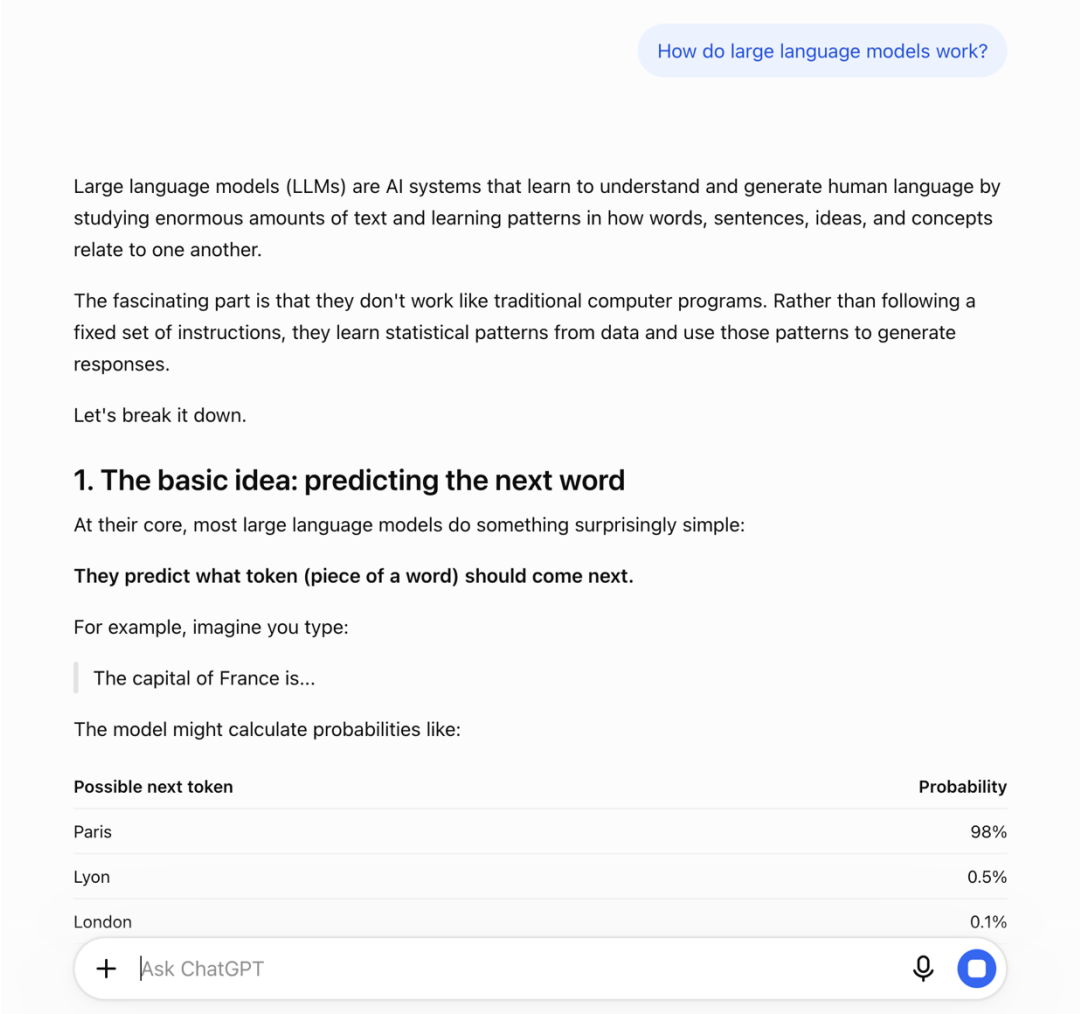
Possible next token	Probability
Paris	98%
Lyon	0.5%
London	0.1%

Ask ChatGPT

First stop: Defining our agent's problem

How do we define our:

- Environment?
- State?
- Actions?
- Objective/reward?

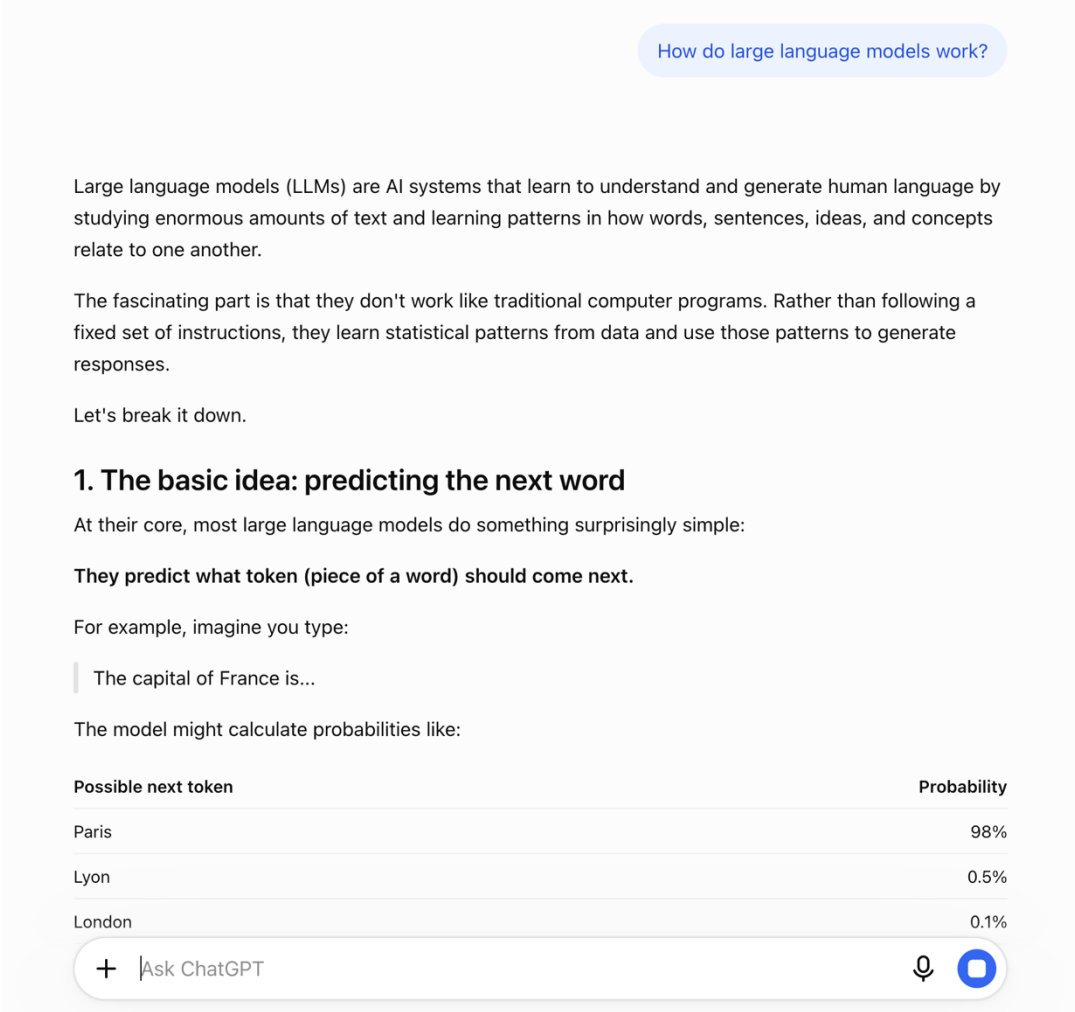


Environment: A text chat

Our agent acts in the context of a text chat

From the *agent's perspective i*s this environment:

- Discrete or continuous?
- Static or dynamic?
- Fully observed or partially observed?
- Deterministic or stochastic?



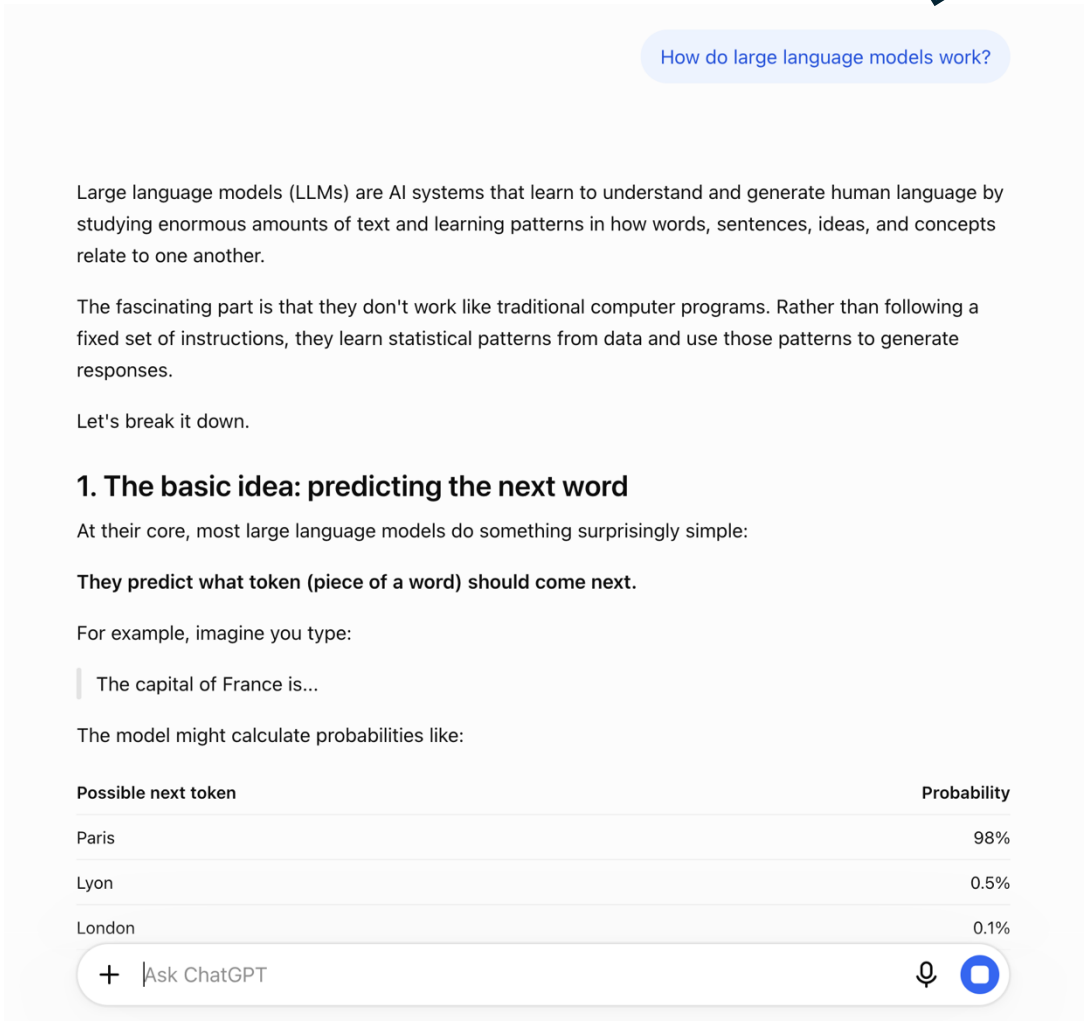
State: What the agent can observe

We can consider the current text in the conversation to be the state

Technically discrete, but the number of possible states is essentially infinite

- If we limit to just 1000 words and only use common works, still roughly: 1000^{5000}
- Today: Can we treat this as continuous?

Starting state for agent:



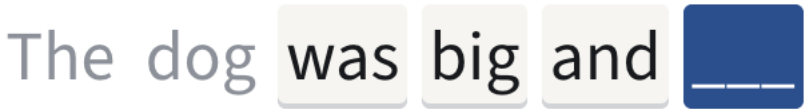
Actions: What can our agent do in a given state?

We can define the granularity of an action in different ways:

- Writing a full response



- Choosing a single word



- Choosing a single letter



How many possible actions are there for each mode?

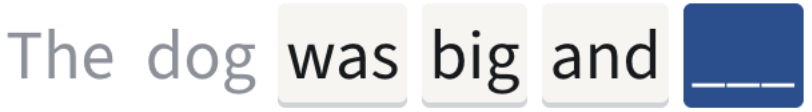
Actions: What can our agent do in a given state?

We can define the granularity of an action in different ways:

- Writing a full response



- Choosing a single word



- Choosing a single letter



We'll end up using something in-between these two!

Objective/reward: Compose the best response

Out goal should be to pick the best response for a given starting state.

How do we measure this?

Quick quiz

What is a word embedding?

A. A representation of a word
B. The context of a word
C. The prediction of a word
D. None of the above

Which technique is used in LLMs

A. Observation
B. Listening
C. Attention
D. Distraction

What do we call false info made-up by an LLM?

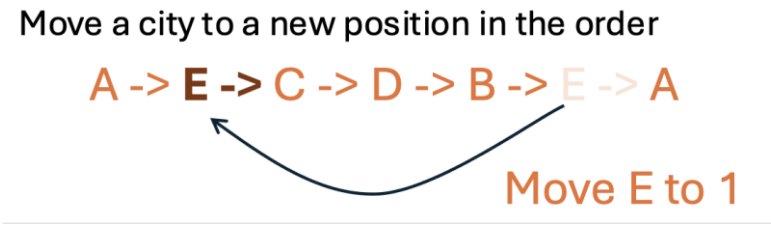
A. Storytelling
B. LLying
C. Modelling gaps
D. Hallucination

Two possible framings

Local Search

- Update our response to optimize our objective
  - Successor function: swap out one word/character in our response
  - Huge search space!
  - Objective function: ???

Like TSP!



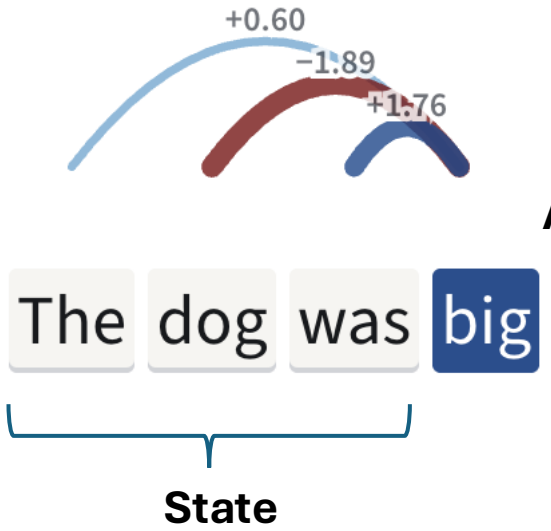
Reflex agent

- Each word is an action that alters the environment state
  - Actions: next word to write
  - Full response becomes an *episode*
  - Matches how humans speak!
  - Reward function: ???
  - Can use imitation learning!

Like cart game!



Standard framing: Next word prediction



Could we also use reinforcement learning?

Each word is an action that alters the environment state

- Actions: next word to write
- Full response becomes an *episode*

No objective: Learn to imitate real text

- Take real text and find state – true action pairs
- State is words already said, action is next word to say

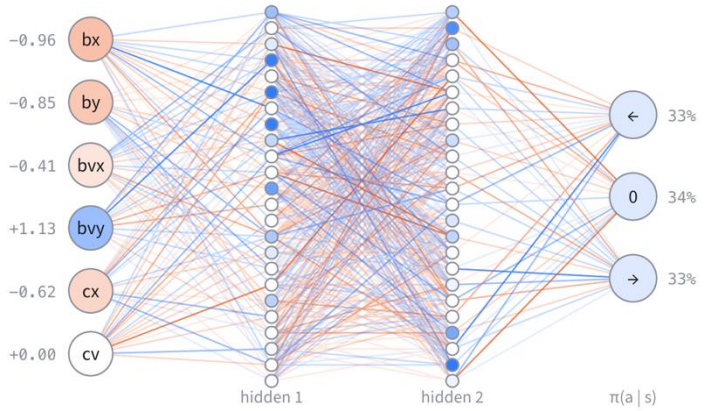
Setup: learn a hypothesis that maps text state to an appropriate action (next word/letter)

In this class: we've really only seen one type of learn-able function, a ***neural network*** (perceptron is a special case)

Can we use it for this setup?

- Neural networks assume state is represented by a ***fixed number*** of ***continuous measurements***

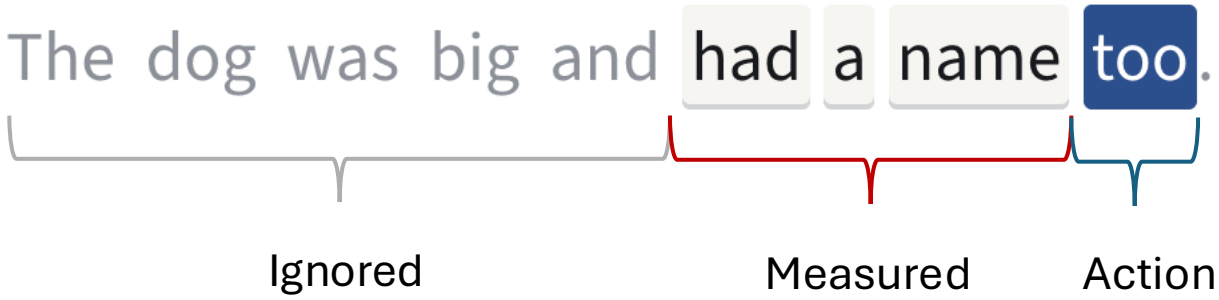
e.g. cart network



Neural networks assume state is represented by a ***fixed number*** of ***continuous measurements***

One solution to the fixed number restriction:

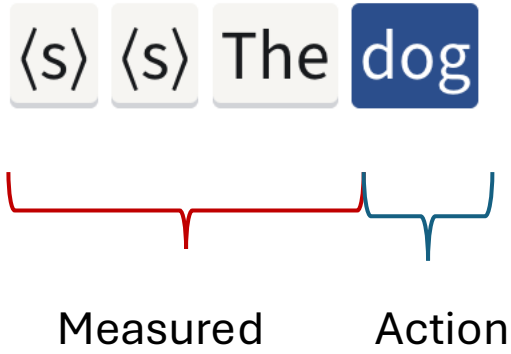
- Only measure the last N words



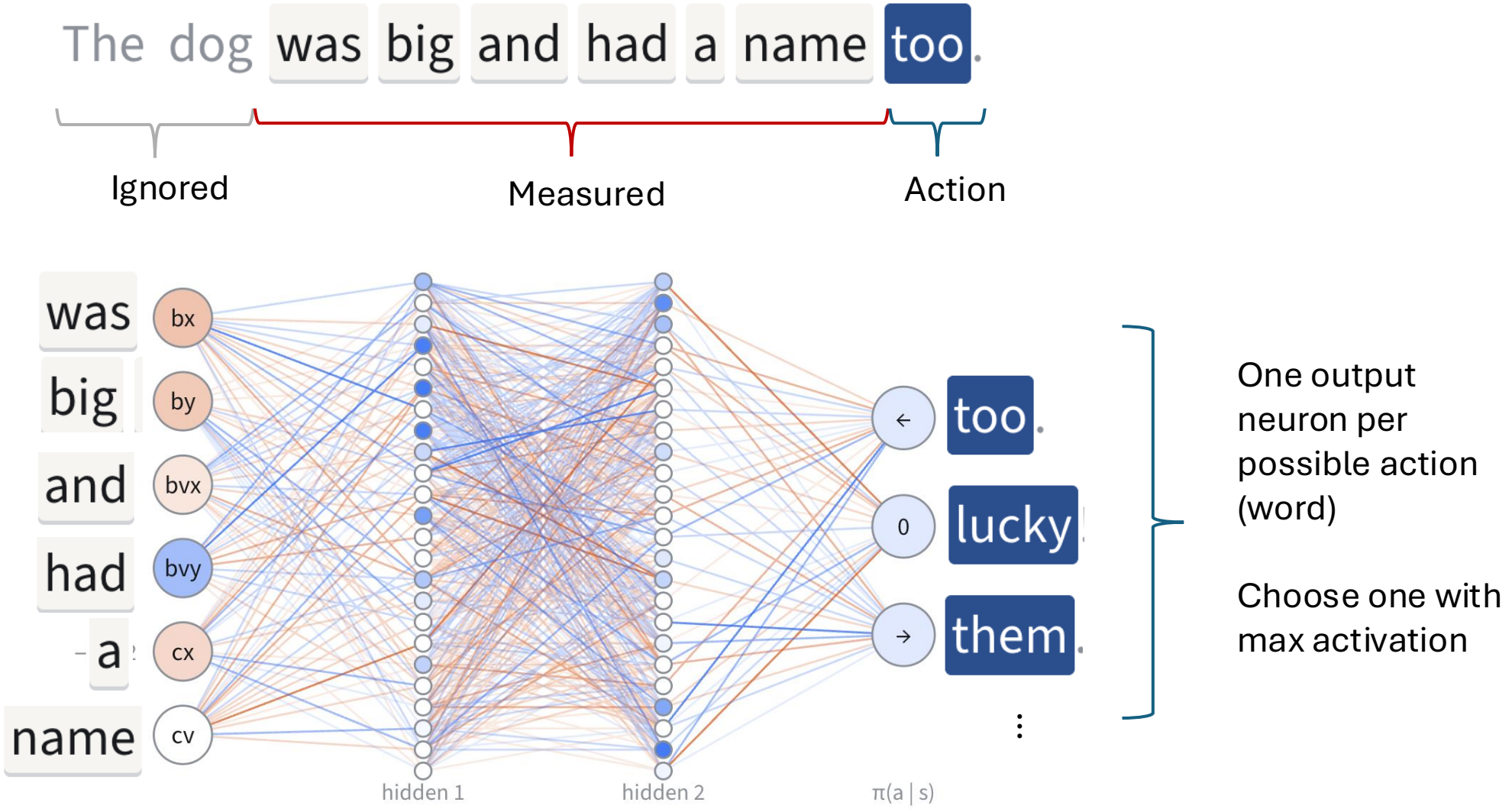
- *We'll see a better solution soon…*

Neural networks assume state is represented by a ***fixed number*** of ***continuous measurements***

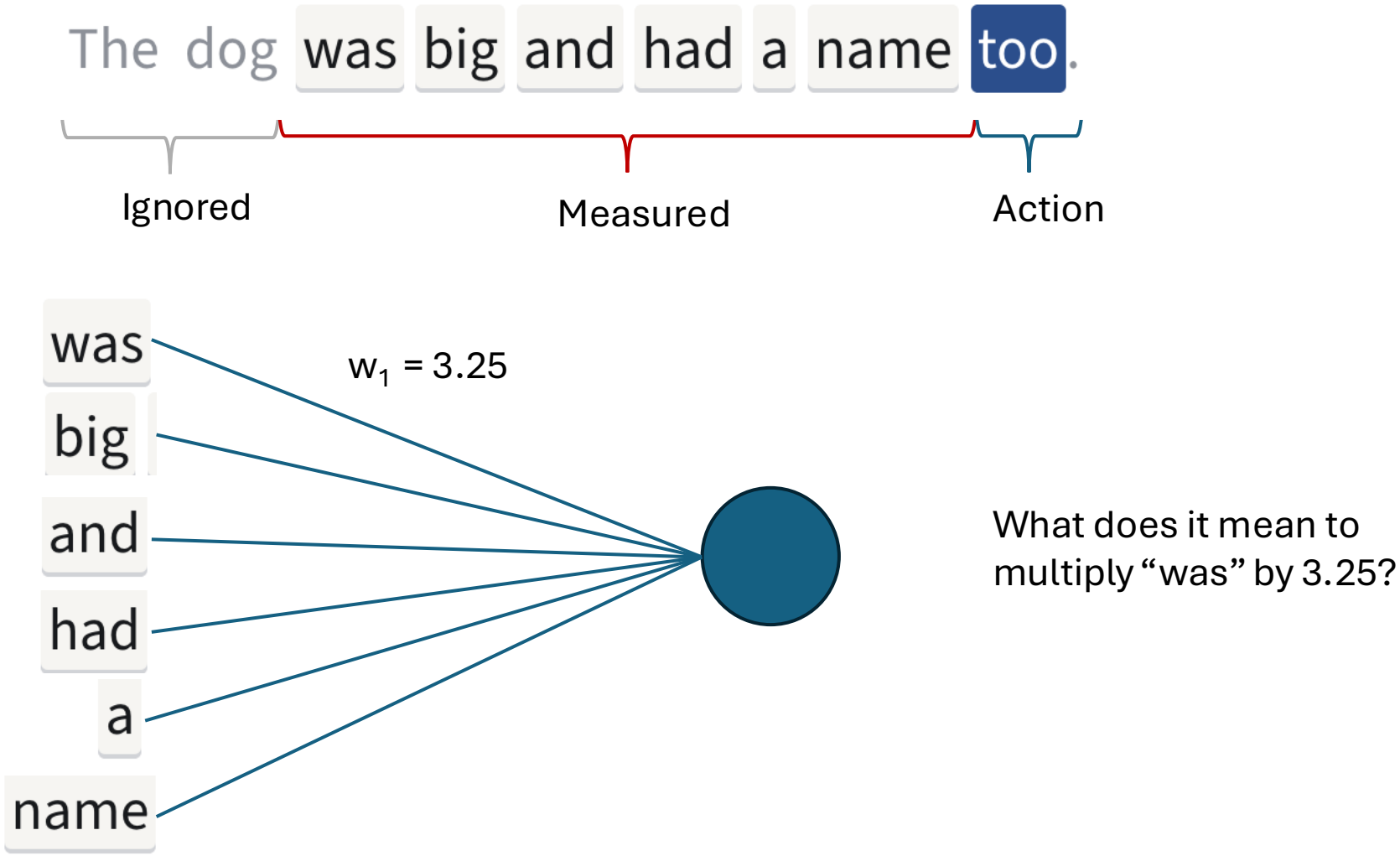
If the state is too short, we can use a special word to represent "before the start".



Conceptually



A single neuron



What does it mean to multiply “was” by 3.25?

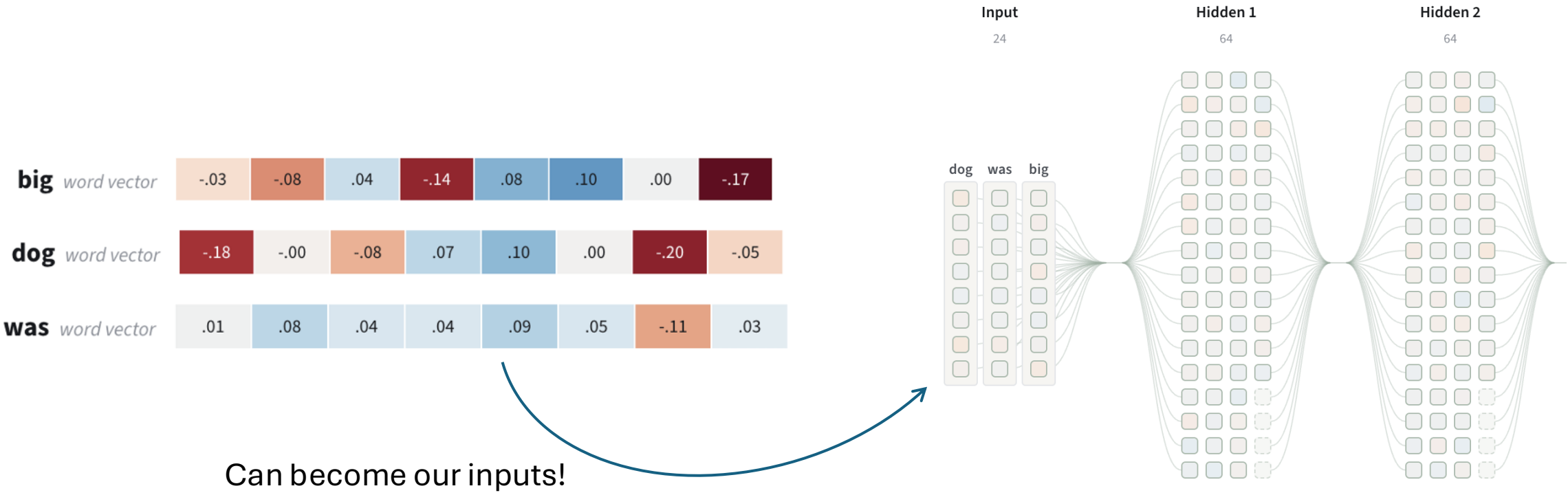
Word embeddings

Idea: Represent each possible word as a set of continuous measurements (*embeddings)*



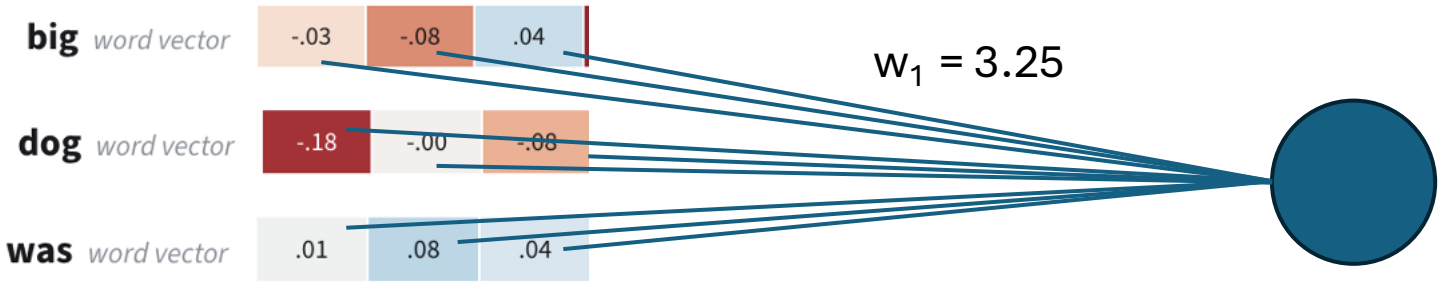
Word embeddings

Idea: Represent each possible word as a set of continuous measurements (*embeddings)*



A single neuron

Connects to all measurements!



But where do the numbers come from?

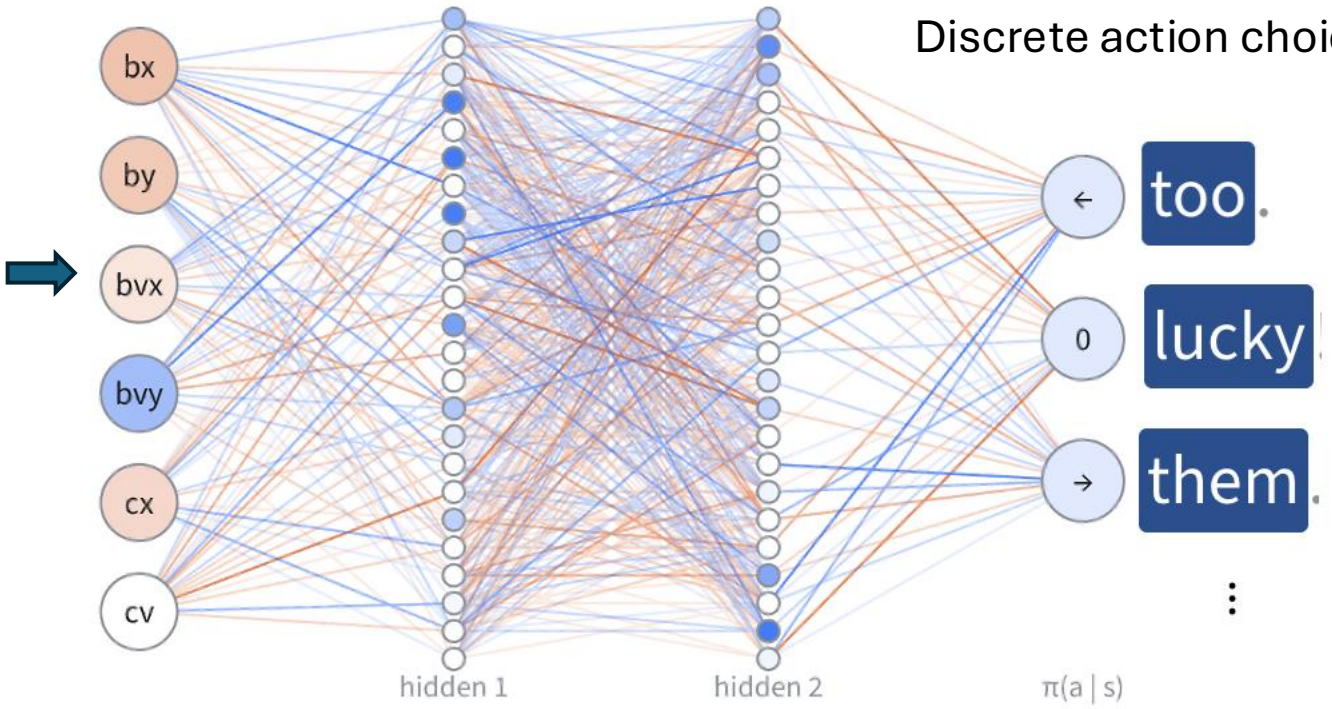
A surprisingly ok idea

Just assign a random set of numbers unique to each word

- As long as they're distinct, a network can learn to map them to actions!

big *word vector*	-.03	-.08	.04	-.14	.08	.10	.00	-.17
dog *word vector*	-.18	-.00	-.08	.07	.10	.00	-.20	-.05
was *word vector*	.01	.08	.04	.04	.09	.05	-.11	.03

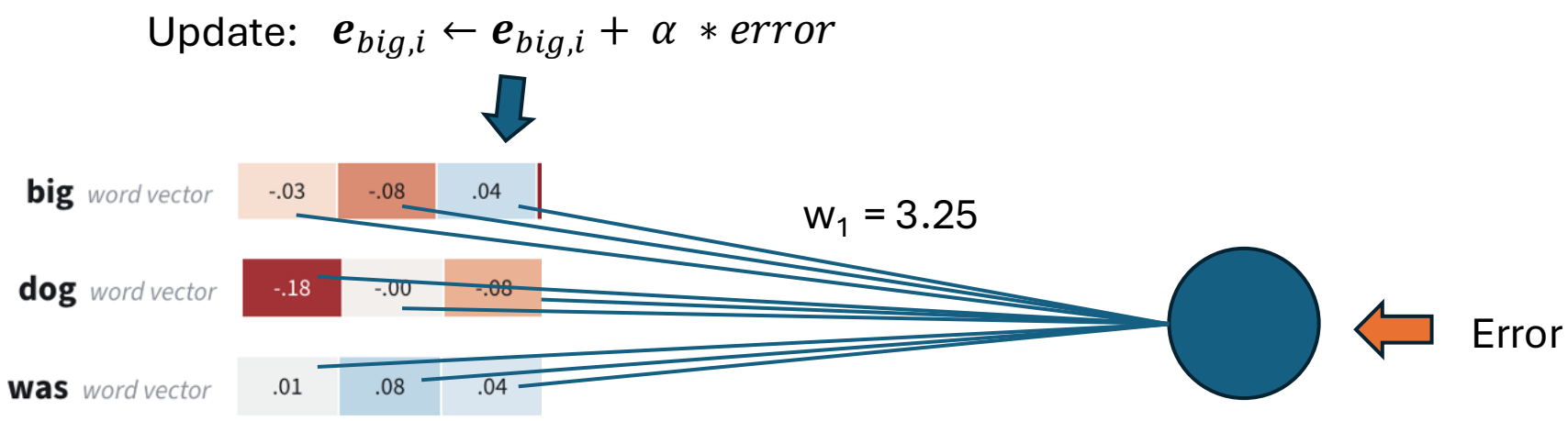
Continuous measurements



A even better idea:

Learn the embeddings themselves!

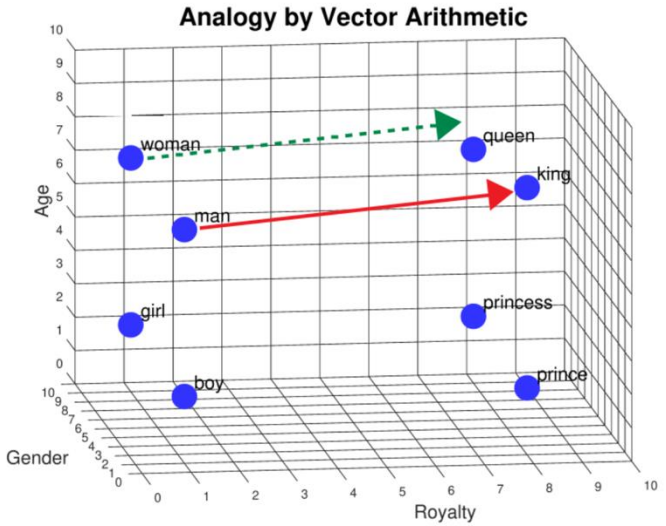
- It doesn't matter what the embeddings are, so long as they're unique to each word
- We already know how to propagate error to a neuron's input



Effect of learning embeddings

Similar words have similar representations!

- In some cases we can even do operations directly



king	[	1,	8,	8]
man	[	1,	7,	0]
king - man	[	0,	1,	8]
woman	[	9,	7,	0]
king - man + woman	[	9,	8,	8]
queen	[	9,	7,	8]

Example in 3-dimensions!



Vocabulary: our action space

Let’s revisit the question of what we consider an action:

- We need to be able to enumerate possible actions, so we need to know all possible words ahead of time (our *vocabulary*)

Is this always possible? What could go wrong?

Vocabulary: our action space

Let’s revisit the question of what we consider an action:

- We need to be able to enumerate possible actions, so we need to know all possible words ahead of time (our *vocabulary*)

Is this always possible? What could go wrong?

- New proper nouns
- Typos and misspellings
- Numbers
- …

Character-level actions don’t have this problem!

Representing text

Tokenization

I love purple cats. Purple cats are great.

- Charater-level: ['I', ' ', 'l', 'o', 'v', 'e'...]
  - Small Vocab (~100)
  - Many features in a single observation (eg sentence)!
- Word-level: ['I', 'love', 'purple', 'cats', '.'...]
  - Large Vocab (~50,000+)
  - Few tokens per obs.
  - What if we encounter a new word, like a name?

Byte-pair embedding

I love purple cats. Purple cats are great.

- Common words should be single tokens
- Limit Vocab size, let rare or unseen words be multiple tokens

e.g.

Cats → 'Cats'

Purplish → 'Purpl' 'ish'

Gabe → 'G', 'a', 'b', 'e'

Byte - Pair embedding - GPT 3

Tokens	Characters
58	301

OpenAI's large language models (sometimes referred to as GPT's) process text using tokens, which are common sequences of characters found in a set of text. The models learn to understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

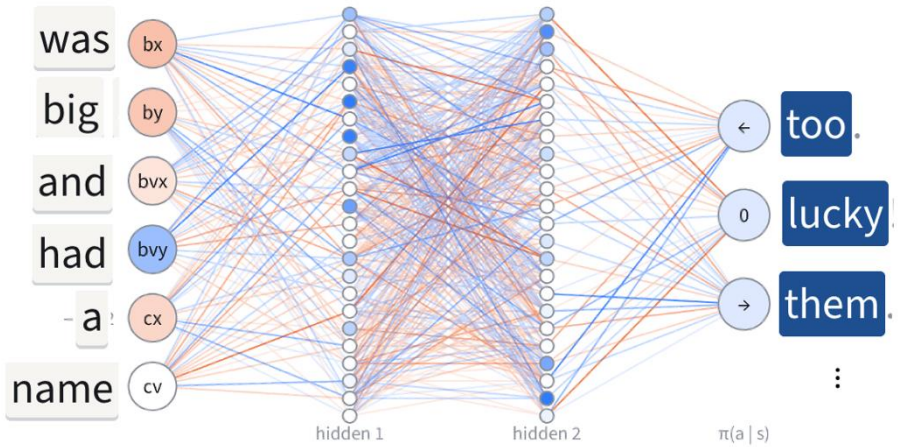
```
[5109, 15836, 596, 3544, 4221, 4211, 320, 57753, 14183, 311, 439, 480,
2898, 596, 8, 1920, 1495, 1701, 11460, 11, 902, 527, 4279, 24630, 315,
5885, 1766, 304, 264, 743, 315, 1495, 13, 578, 4211, 4048, 311, 3619,
279, 29564, 12135, 1990, 1521, 11460, 11, 323, 25555, 520, 17843, 279,
1828, 4037, 304, 264, 8668, 315, 11460, 13]
```

Picking the next word/token

Not always the best idea to pick the max activation

- No variety in responses
- Not great for RL (more on this later)
- Can get stuck in local optima

Alternative: Map outputs to *probabilities*



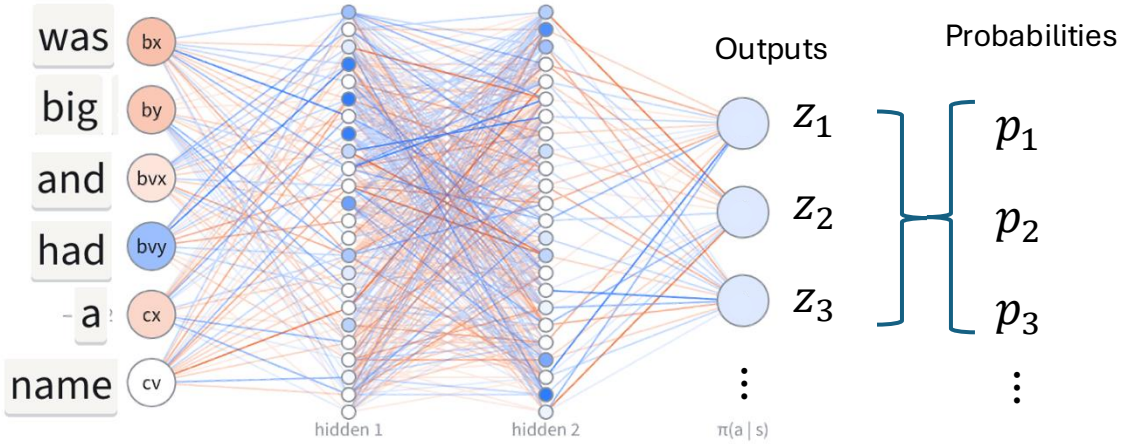
Choose at random with these probabilities!



Review: ***Softmax*** activation

Special activation that maps a set of neurons to numbers that:

- Are positive
- Sum to 1



Positive

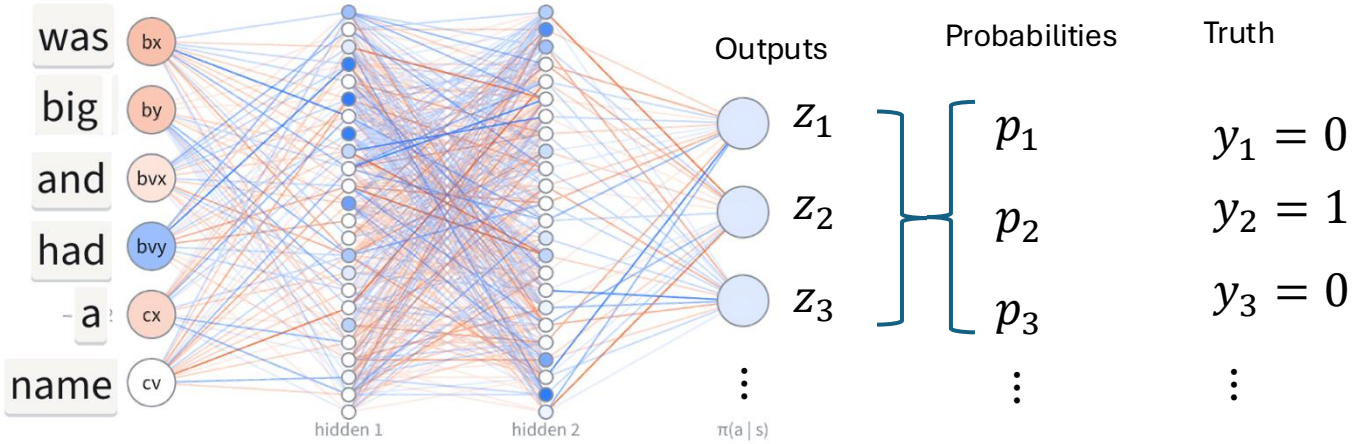
$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Sum to 1

Error for softmax output

Use a special approach to computing error with softmax

- Recall: Encode ”correct” action as *one-hot* vector
- Error for (pre-softmax) output is difference between probability and one-hot encoded truth



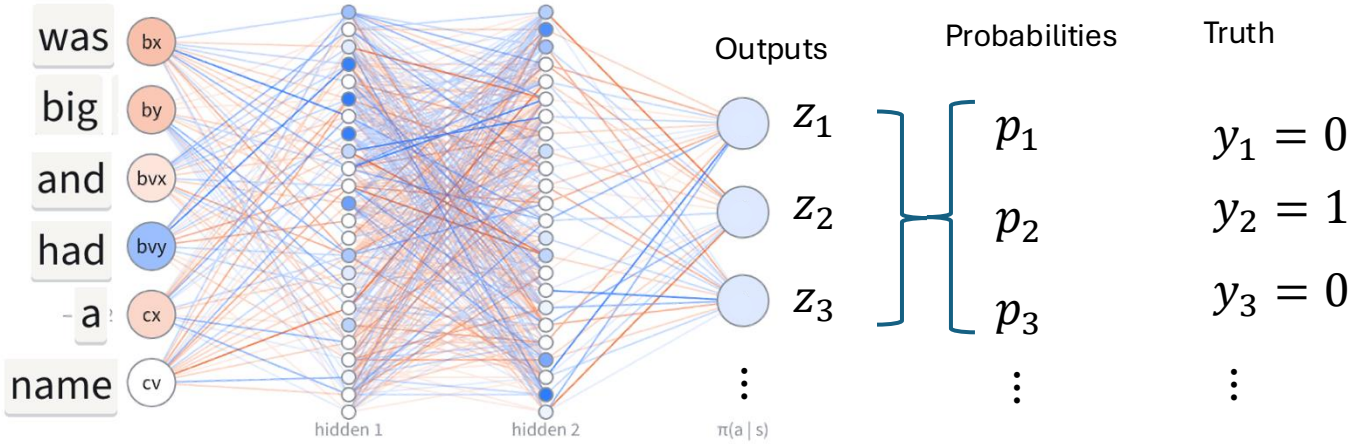
$$error\ for\ z_i = p_i - y_i$$

Error for softmax output

Softmax error corresponds to a different loss function: ***Cross entropy loss***

$$\text{Loss}(p_i) = -\, y_i \log p_i$$

$$\text{Loss}(z_i) = -\, y_i \log \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

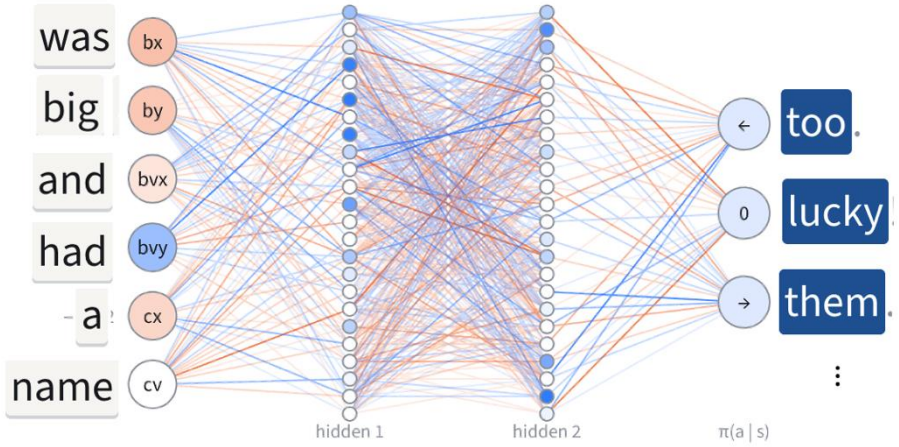


$$\frac{\partial}{\partial z} \text{Loss}(z_i) = p_i - y_i$$

Picking the next word/token

Not always the best idea to ~~pick the max activation~~ **sample**

- Might pick a really unlikely word (e.g. $p_i = 0.0001$)
- If we sample 10,000 words the probability of picking a bad one becomes significant!



Choose at random with these probabilities!



Top-K Sampling

Goal: Something in-between always picking the max output and sampling any possible work

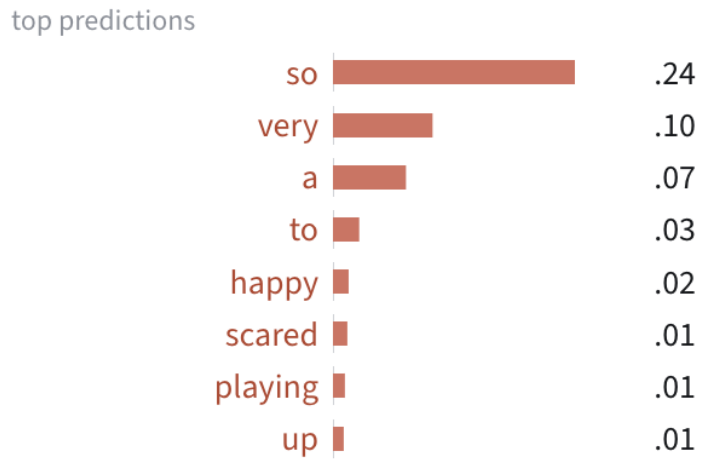
- Idea, take the top *k* outputs (e.g. k=10) and apply the softmax just to those!

For i/j in top k outputs:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

Otherwise:

$$p_i = 0$$



Only when testing! Training still needs to be prepared for any word

Demo!

We need to scale up

Questions to answer:

- How do we (better) handle long context?
- Can we define a reward for RL?
- Even if *can* get this all to work, *should we?*