Applied Economics with Large Language Models

Thiemo Fetzer

University of Warwick & University of Bonn & CEPR & LSE & AEAI

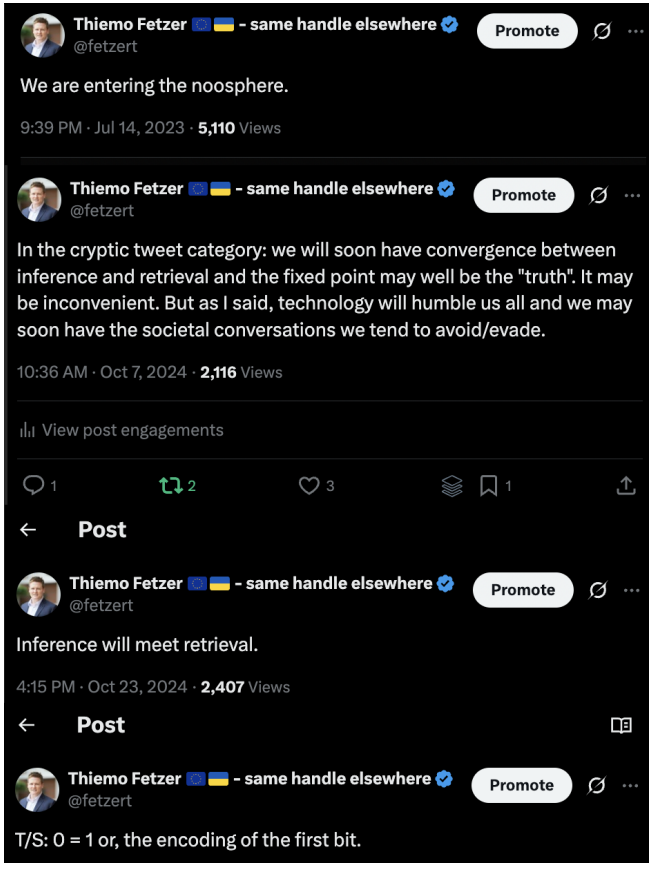
May 29, 2025

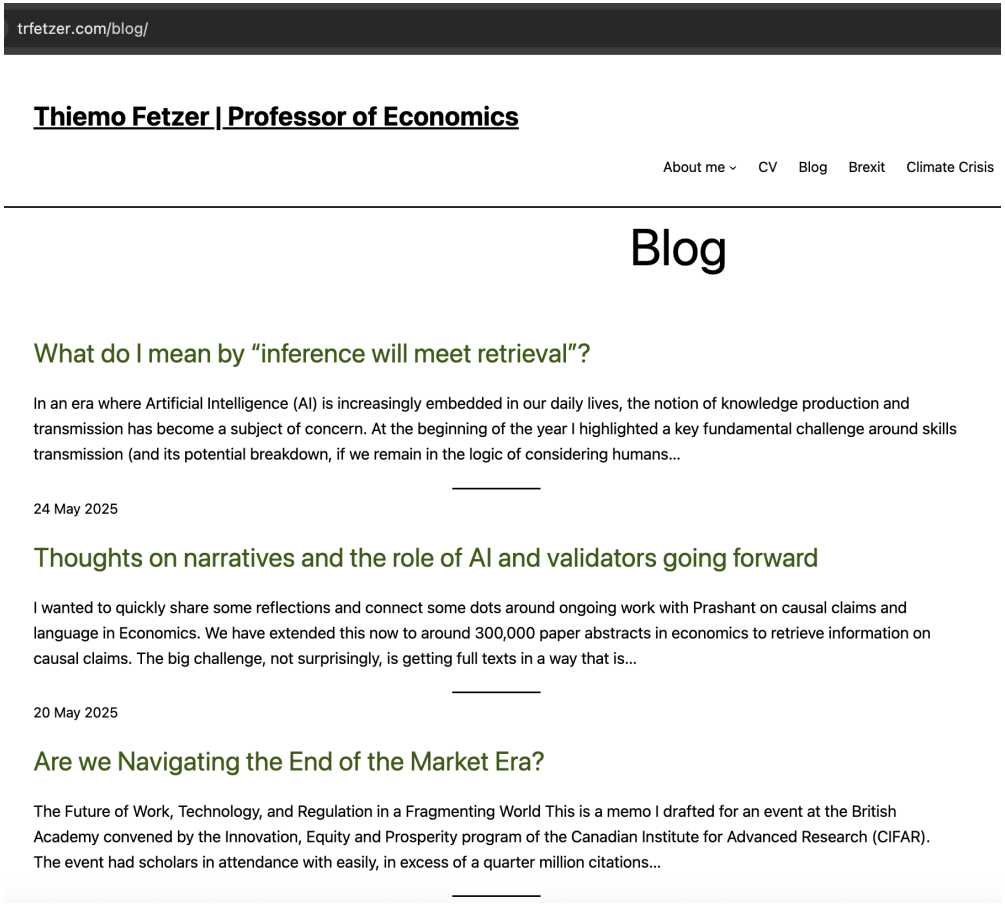
This course

I want to spend the next three days interweaving the following

- Provide a brief introduction to large language models
- Make connections to how we information process as humans and draw some similarities
- Interweave concrete applications and architectures for applied economics research
- Zoom out to provide a running commentary

Running commentary





$\rightarrow$ provide *research augmented comments* on role of AI in shaping economic and geopolitical developments

Plan

How humans process information

Language Modeling Basics

Embeddings as workhorse for large language models

LLM Building Blocks

Logistic Regression as a Neural Network

Developing scalable classifiers

Working with LLMs

Illustrating some concepts

1. **Listen to some natural text**

   Who can still remember the first words I used at the start of this lecture?
2. **What did you have for breakfast**

   Distinguishing between what is relevant and not.
3. **What is learning?**

   Connecting dots.
4. **What do humans need look like in the post materialistic society?**

   Attention is all you need or alternatives to human flourishing.
5. **What are we actually doing?**

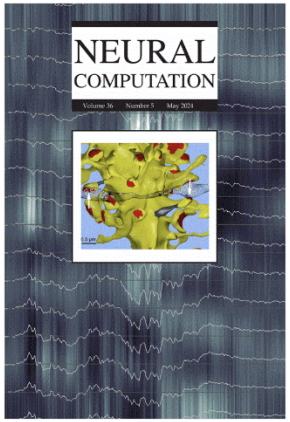
   Connecting dots.

What a byte?

object size	legacy
1	1 bytes
1024	1 Kb
1024^2	1 Mb
1024^3	1 Gb
1024^4	1 Tb
1024^5	1 Pb

How about the human brain?

Volume 36, Issue 5
May 2024



‹ Previous Article Next Article ›

Article Contents

April 23 2024

Synaptic Information Storage Capacity Measured With Information Theory

In Special Collection: CogNet

Mohammad Samavat ✉, Thomas M. Bartol, Kristen M. Harris ✉, Terrence J. Sejnowski ✉

Neural Computation (2024) 36 (5): 781–802.
https://doi.org/10.1162/neco_a_01659 **Article history**

Abstract

Variation in the strength of synapses can be quantified by measuring the anatomical properties of synapses. Quantifying precision of synaptic plasticity is fundamental to understanding information storage and retrieval in neural circuits. Synapses from the same axon onto the same dendrite have a common history of coactivation, making them ideal candidates for determining the precision of synaptic plasticity based on the similarity of their physical dimensions. Here, the precision and amount of information stored in synapse dimensions were quantified with Shannon information theory, expanding prior analysis that used signal detection theory (Bartol et al., 2015). The two methods were compared using dendritic spine head volumes in the middle of the stratum radiatum of hippocampal area CA1 as well-defined measures of synaptic strength. Information theory delineated the number of distinguishable synaptic strengths based on nonoverlapping bins of dendritic spine head volumes. Shannon entropy was applied to measure synaptic information storage capacity (SISC) and resulted in a lower bound of 4.1 bits and upper bound of 4.59 bits of information based on 24 distinguishable sizes. We further compared the distribution of distinguishable sizes and a uniform distribution using Kullback-Leibler divergence and discovered that there was a nearly uniform distribution of spine head volumes across the sizes, suggesting optimal use of the distinguishable values. Thus, SISC provides a new analytical measure that can be generalized to probe synaptic strengths and capacity for plasticity in different brain regions of different species and among animals raised in different conditions or during learning. How brain diseases and disorders affect the precision of synaptic plasticity can also be probed.

Estimated Synaptic Storage Capacity

- Number of synapses in the human brain:
  $\sim 10^{14}$ (100 trillion)
- Synaptic information storage capacity (SISC) per synapse:
  4.1 to 4.6 bits (Samavat et al., 2024)
- **Total brain capacity:**

$$(4.1 \text{ to } 4.6) \times 10^{14} \text{ bits} \approx 51 - 57 \text{ TB}$$

- *Assuming 8 bits = 1 byte and converting to terabytes*

Comparison With Other Data Sources

Data Source	**Size Estimate**
Human brain (estimated)	51–57 TB
All of Wikipedia (text only)	$\sim$ 20 GB
Entire English Wikipedia (with media)	$\sim$ 200 GB
Netflix's daily data usage	$\sim$ 1000 TB (1 PB)
YouTube (new uploads per day)	$\sim$ 720 TB
All digitized books (Google estimate)	$\sim$ 15 TB
Entire web (text content)	$\sim$ 50–100 TB
Internet Archive (Wayback Machine)	$>$ 70 PB

Table: Human brain vs. digital data scales (approximate).

Key Takeaways

- The human brain likely stores ~50–60 TB of data using synaptic strength encoding.
- That's **more than all of Wikipedia and most of the web's text content**.
- But it's still **tiny compared to the size of the entire Internet**.
- Biological information storage is efficient, flexible, and dynamic—not just about raw capacity.

How Much Information Does the Brain Process Daily?

- **Visual input alone:** $\sim$74 GB/day (Koch et al., 2006)
- **Full sensory input:** Estimated at $>$**100 GB/day**
- **Total (incl. unconscious processing):** Up to **10+ TB/day**
- Most of this information is *not* retained in long-term memory

Takeaway

The brain filters massive streams of data, encoding only what's salient.

What Does This Imply About Forgetting?

- If the brain stored all incoming data, it would run out of space in:
  - $\sim$1.5 years (100 GB/day)
  - $\sim$2 months (1 TB/day)
  - $<$1 week (10 TB/day)
- But that doesn't happen.

Key Insight

Forgetfulness isn't a flaw — it's a feature. The brain filters, compresses, and abstracts, retaining only what serves future decisions.

Processing vs. Storage: The Brain's Efficiency

Information Processed Daily:

- ∼70–100 GB/day (conscious perception)
- Up to **10+ TB/day** (all brain activity)

Long-Term Storage Capacity:

- ∼51–57 TB total (Samavat et al., 2024)
- Based on $\sim 10^{14}$ synapses $\times$ 4.1–4.6 bits/synapse

Key Insight

The brain compresses, abstracts, and forgets — storing only what matters. It's not a camera, it's a meaning-maker.

How People Process Information

- Information is not passively received — it is actively **filtered and interpreted**.
- Two key dimensions of interpretation:
  1. **Personal filters**
     Beliefs, values, lived experience, cognitive biases
  2. **Institutional or mechanical filters**
     Media framing, educational systems, algorithmic curation

Interpretive Divergence

Even when exposed to the same facts or evidence, individuals and groups may arrive at **radically different conclusions**.

How People Process Information

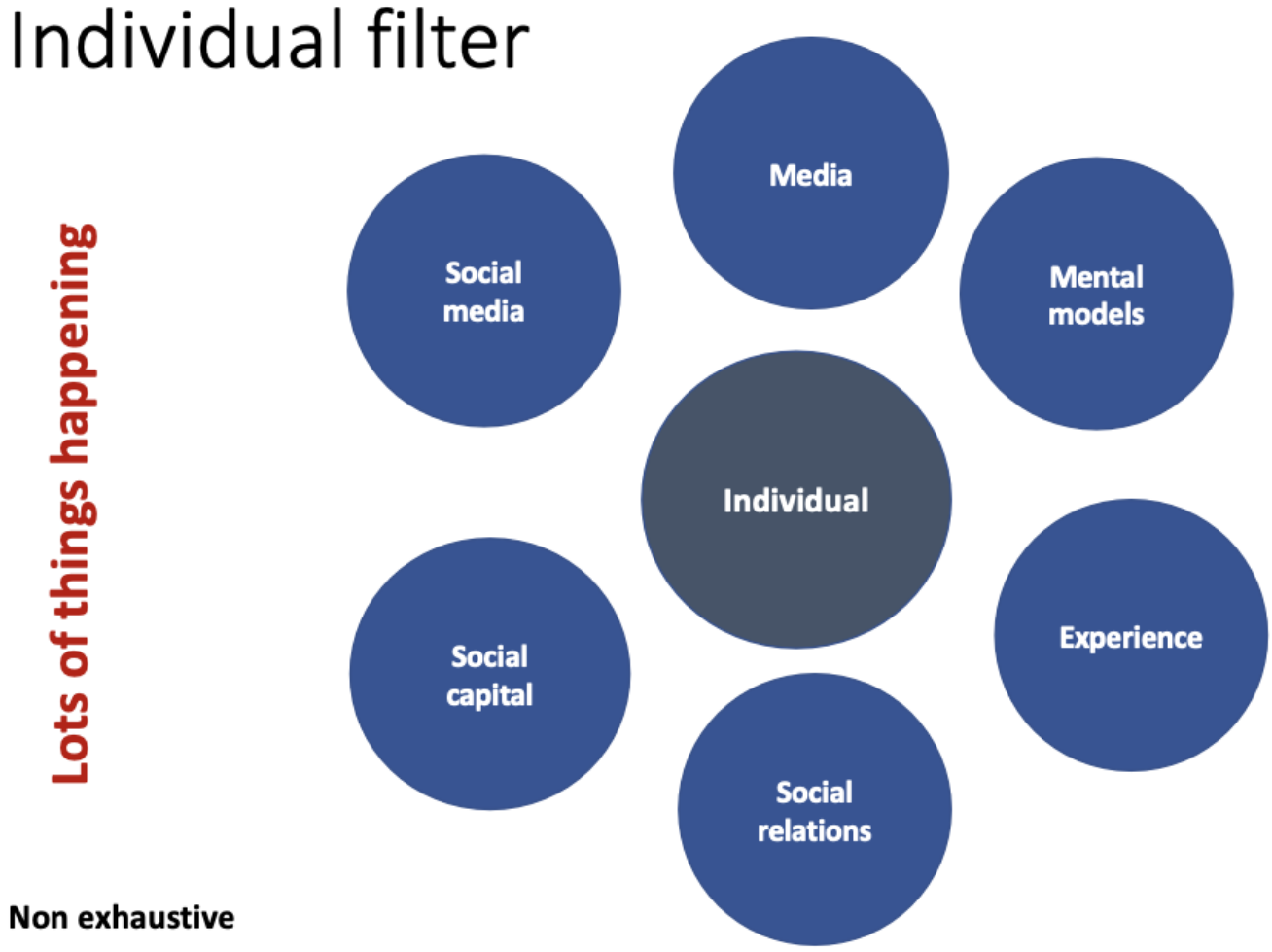
- Information is not passively received — it is actively **filtered and interpreted**.
- Two key dimensions of interpretation:
  1. **Personal filters**
     Beliefs, values, lived experience, cognitive biases
  2. **Institutional or mechanical filters**
     Media framing, educational systems, algorithmic curation

Interpretive Divergence

Even when exposed to the same facts or evidence, individuals and groups may arrive at **radically different conclusions**.

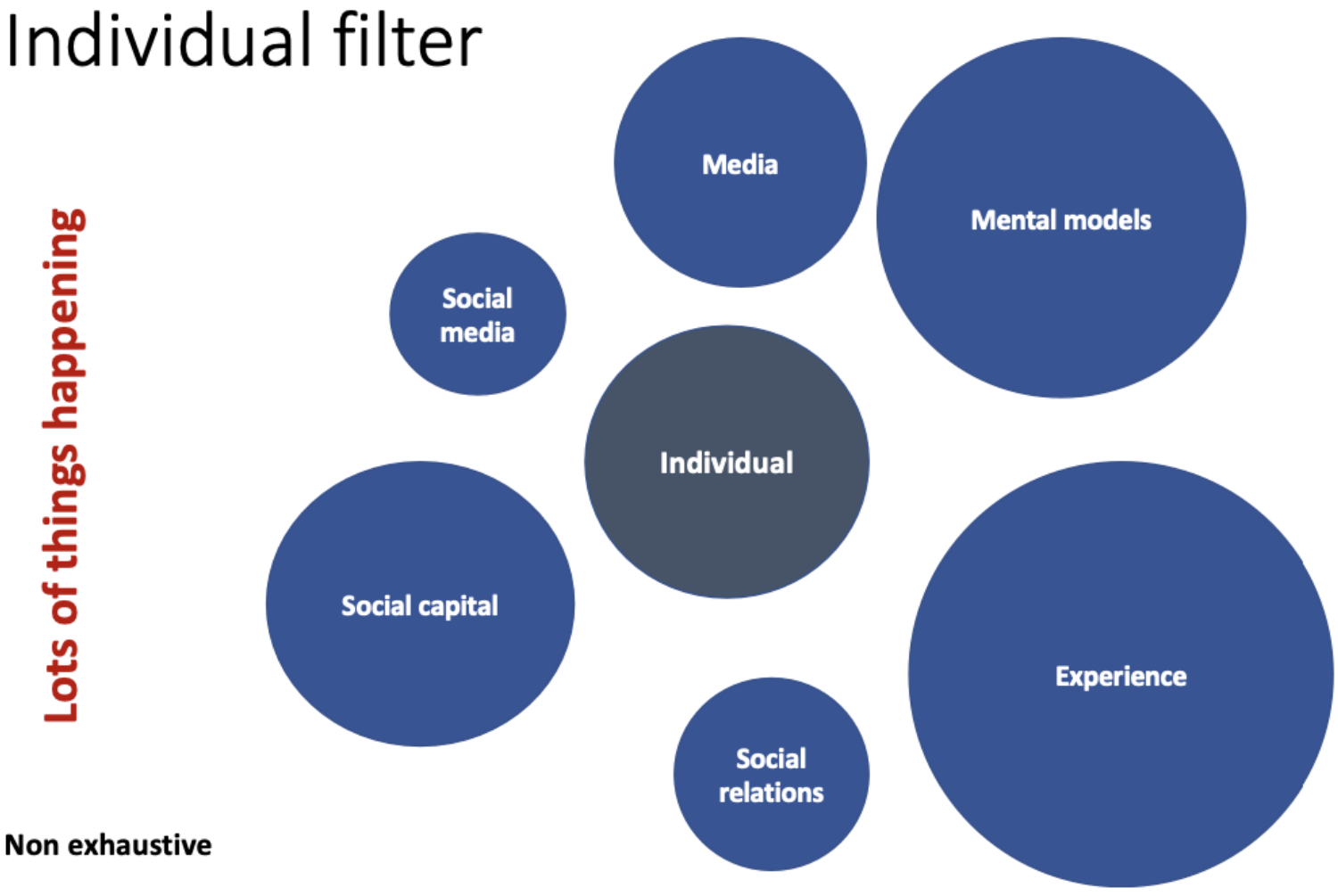
These filters play a crucial role in shaping narratives — especially during transitions, crises, or social change.

Processing information



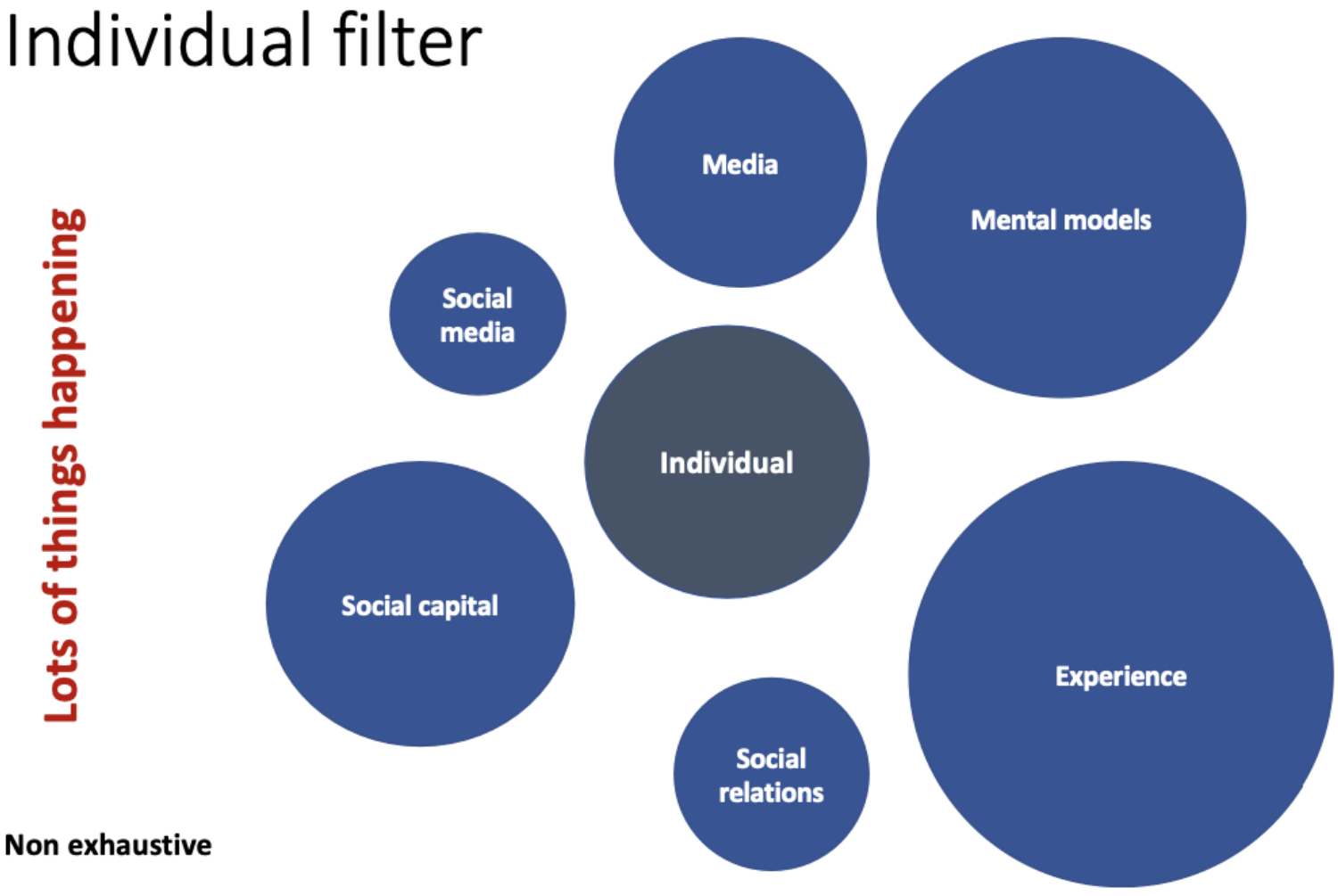
from Keynote at SHAPER Workshop 2023 on the *Political Economy of Climate (In)Action*.

Processing information



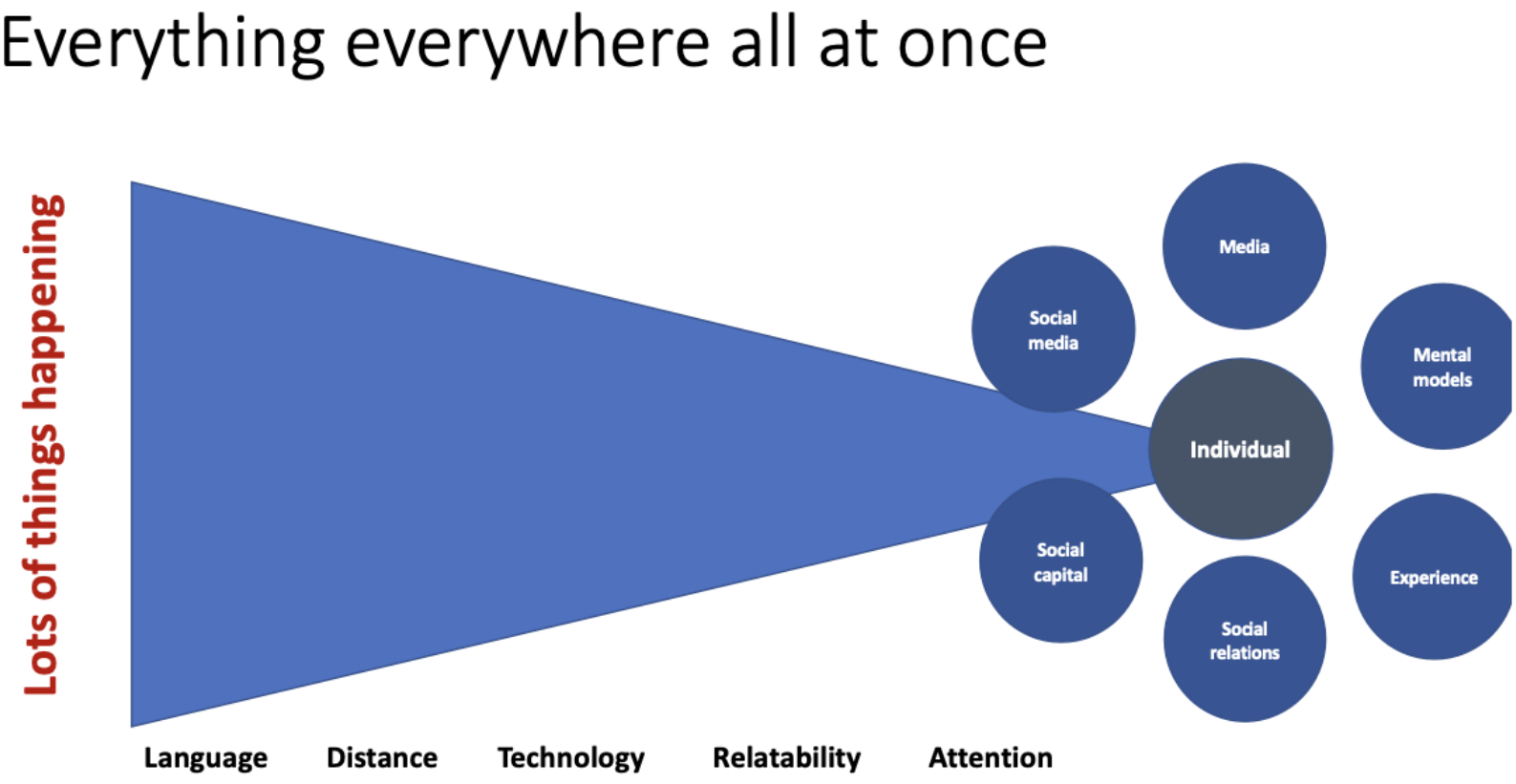
from Keynote at SHAPER Workshop 2023 on the *Political Economy of Climate (In)Action*.

Processing information



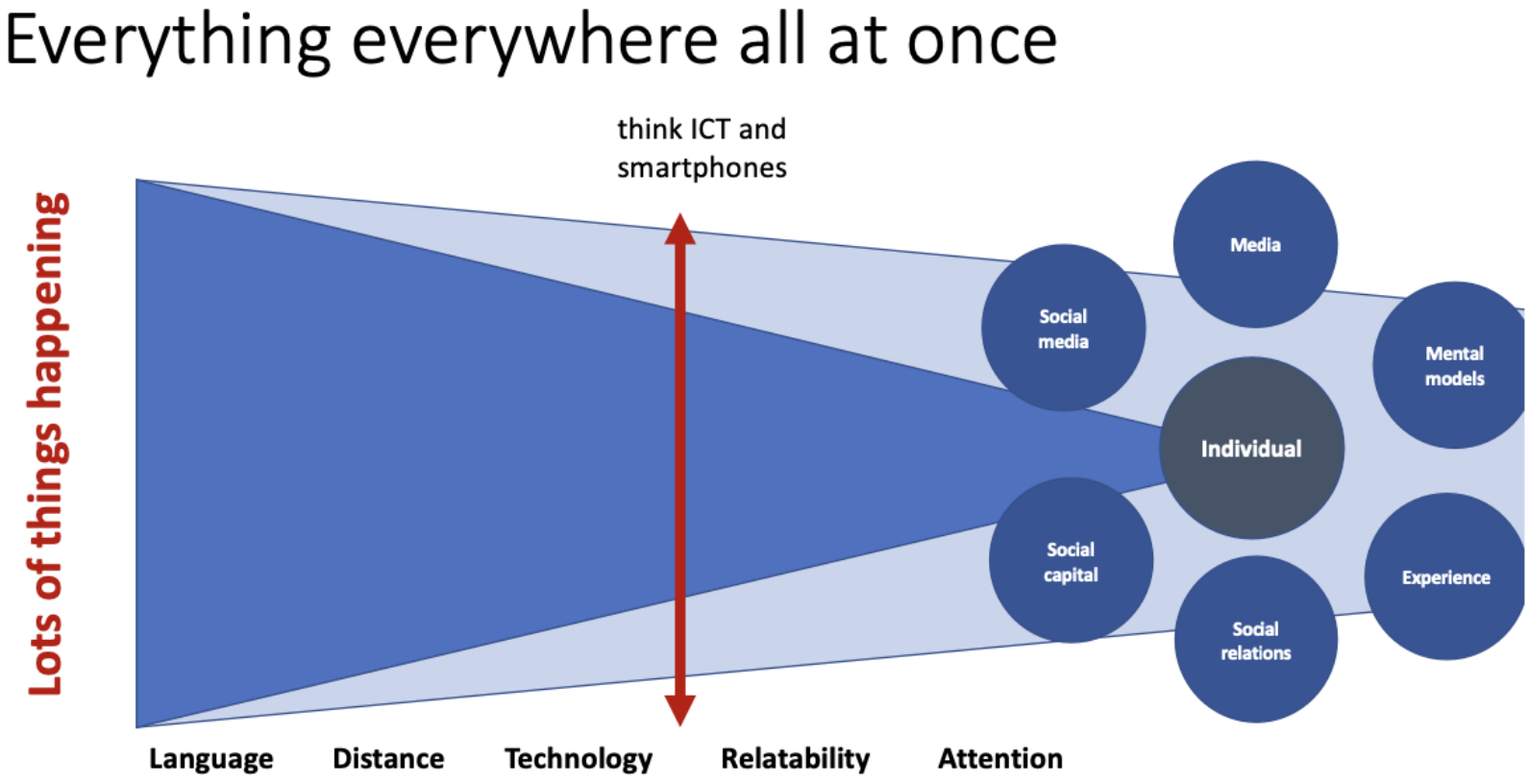
from Keynote at SHAPER Workshop 2023 on the *Political Economy of Climate (In)Action*.

Information flow



from Keynote at SHAPER Workshop 2023 on the *Political Economy of Climate (In)Action*.

Technology shocks

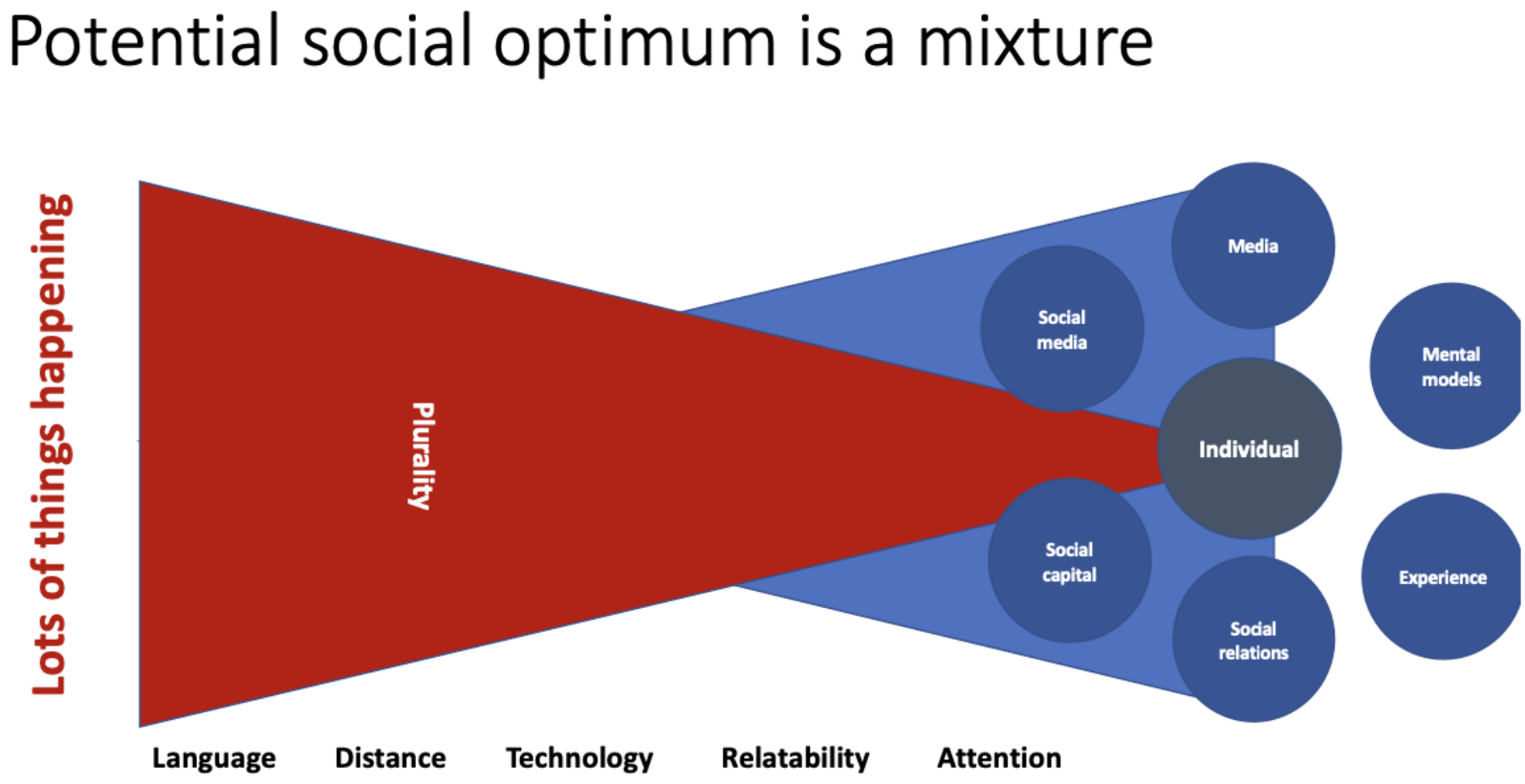


Fetzer & Garg (2025) Network Determinants of Cross-Border Media Coverage of Natural Disasters.

Information flow

from Keynote at SHAPER Workshop 2023 on the *Political Economy of Climate (In)Action*.

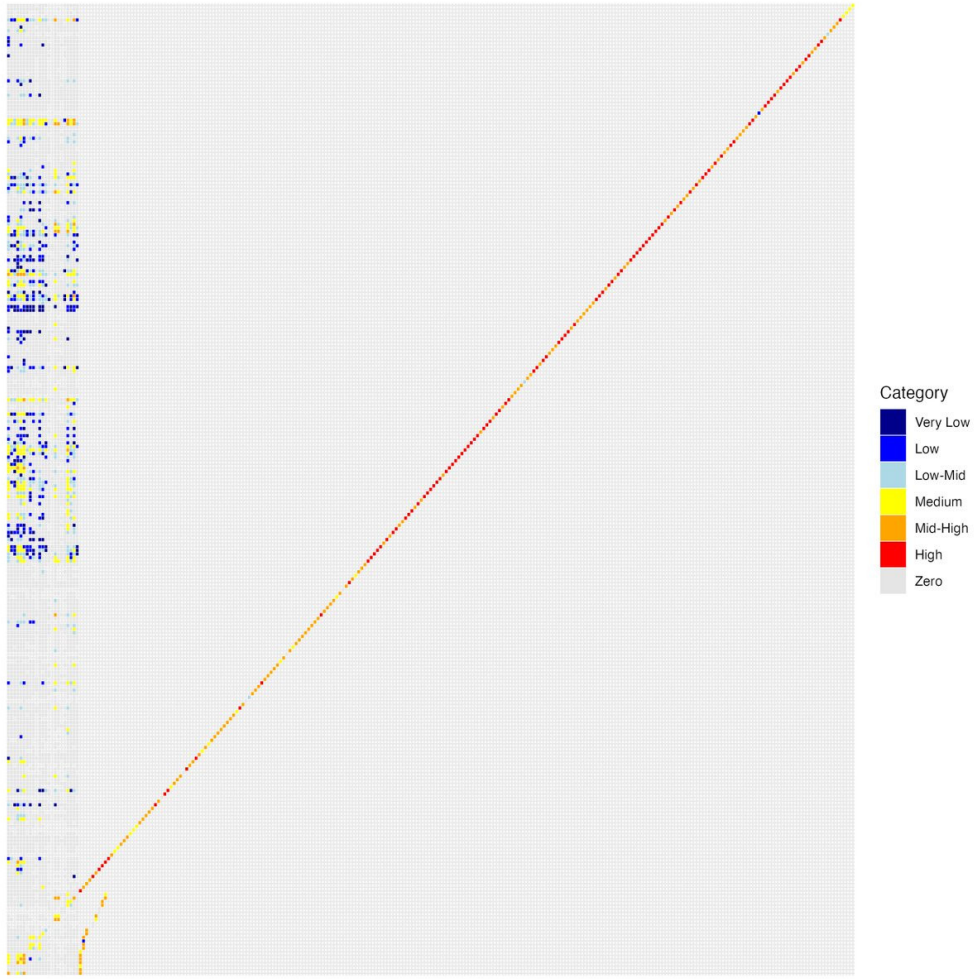
Influencers (?)



Key Insight

He who controls the information sphere can effectively exercise "control". This has geopolitical implications.

Topology of Societal Organisation



Key Insight

Input-output tables may be key to understanding societal organisation

How Do We Perceive Time?

- Time perception is **psychological**, not physical
- Influenced by:
  - **Attention and arousal**
  - **Novelty and emotional intensity**
  - **Memory encoding**
- More change/events = longer perceived duration

How Do We Perceive Time?

- Time perception is **psychological**, not physical
- Influenced by:
  - **Attention and arousal**
  - **Novelty and emotional intensity**
  - **Memory encoding**
- More change/events = longer perceived duration

Example

"Time flies when you're having fun" — but we *remember* those moments more vividly.

Why Does the Brain Crave Experience?

- Brain = **predictive engine**
- Constantly seeks to minimize surprise
- Novel experiences provide:
  - Model updates (learning)
  - Emotional salience (dopamine, memory consolidation)
  - Subjective expansion of time

Why Does the Brain Crave Experience?

- Brain = **predictive engine**
- Constantly seeks to minimize surprise
- Novel experiences provide:
  - Model updates (learning)
  - Emotional salience (dopamine, memory consolidation)
  - Subjective expansion of time

Implication

We pursue experience not just to "feel alive" — but to update internal models of the world.

The Experience Economy and Cognitive Design

- Routine compresses time perception
- Novelty stretches it and anchors memory
- This has real design implications:
  - Travel, art, learning, risk-taking
  - Avoid passive consumption and algorithmic dullness

The Experience Economy and Cognitive Design

- Routine compresses time perception
- Novelty stretches it and anchors memory
- This has real design implications:
  - Travel, art, learning, risk-taking
  - Avoid passive consumption and algorithmic dullness

Designing for Memory

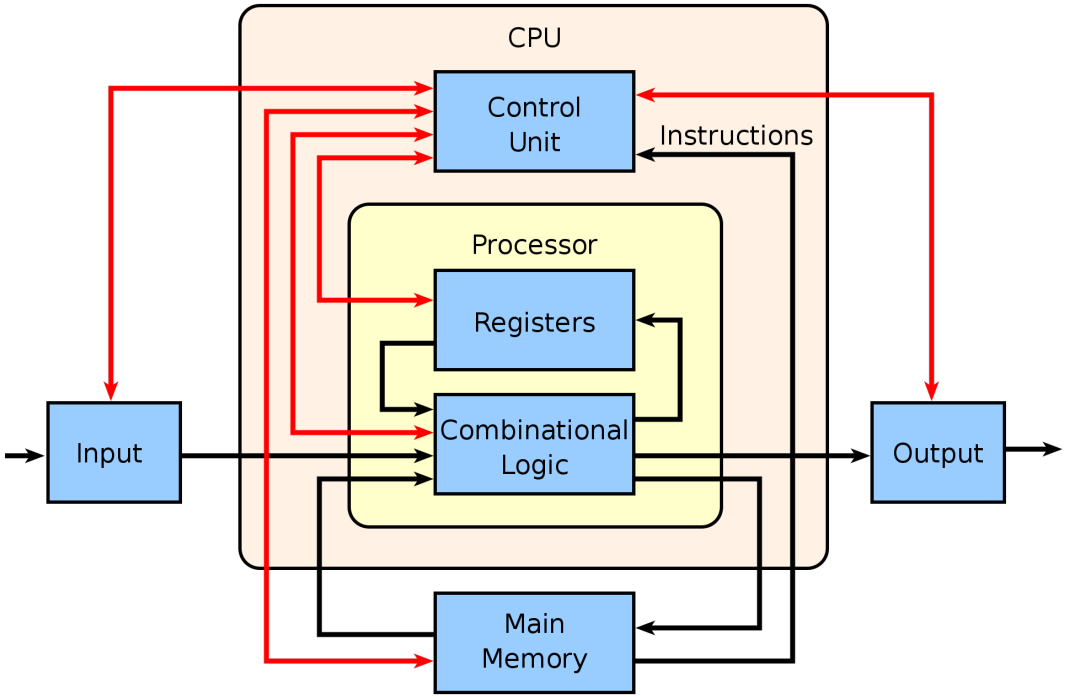
Shape experiences that maximize meaningful change, not just stimulation.

Summary

- The brain filters and encodes vast amounts of information daily
- Our perception of time is shaped by memory, attention, and novelty
- The craving for experience is biological, not just cultural
- Designing for experience = designing for salience and memory

"We are not made to remember days, we are made to remember moments."

Processing information in a Von Neuman Computer architecture



A CPU interacts with memory, input/output, and internal components like registers and control logic.

Why do you need RAM at all?

- RAM = Rapid Access Memory
- Data in R is stored in RAM
- Speed of data manipulations depends on memory speed
- Large datasets may exceed available RAM $\rightarrow$ data is swapped to HDD
- Swap reduces performance

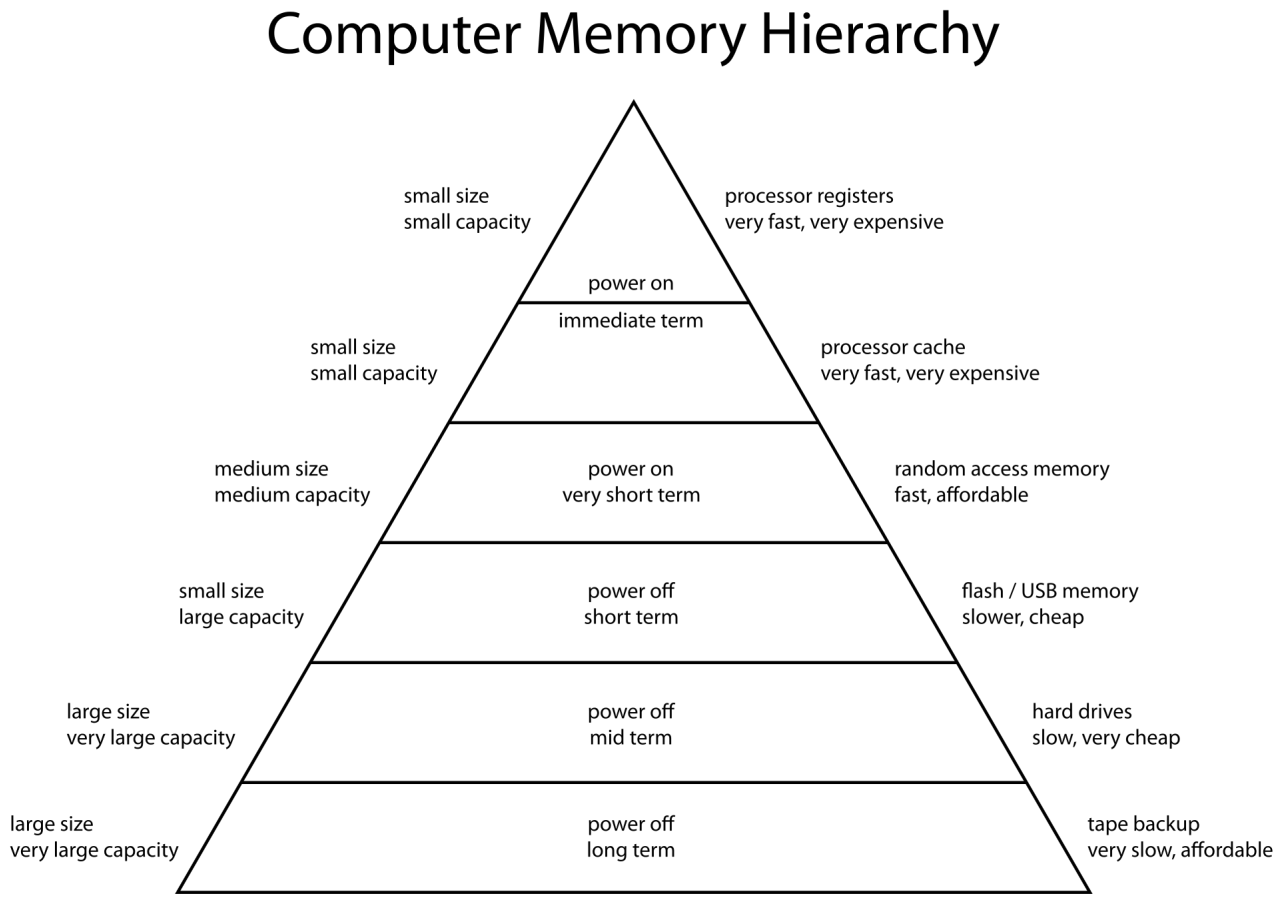
Typical read times:

- RAM: $\sim$100 nanoseconds
- SSD: $\sim$16,000 nanoseconds

Typical throughput:

- HDD: 80–160 MB/s
- SSD: 200–550 MB/s

Hierarchy of memory in terms of speed



How much information can be transmitted through speech?

Italians speak fast – up to **9 syllables/sec** – while Germans average **5–6 syllables/sec**. Yet both transmit about the same **information rate**: **39 bits/sec**.

What Does That Add Up To?

- $39\,\text{bits/sec} \times 86{,}400\,\text{sec/day} = 3{,}369{,}600\,\text{bits/day}$
- $\approx 421{,}200$ bytes $\Rightarrow$ **0.40 MB/day**
- Assuming 6 bytes/word: $\approx 70{,}200$ words/day
- At 500 words/page: $\approx$ **140 pages of ASCII English text**

> *"Languages differ in how they encode information, but they converge in how much they communicate per second."*

`https://www.science.org/content/article/human-speech-may-have-universal-transmission-rate-39-bits-se`

Common Applications of LLMs in Applied Economics

- **Data categorization or classification** – applying of *nomenclatures*
- **Information retrieval applications** – retrieving a *needle from a haystack*
- **Dimensionality reduction** – summarization of *large datasets*
- **Dataset construction** – leaning on embodied knowledge in LLMs
- **Retrieval augmented dataset construction** – prompt and context into context
- **Reasoning tasks** – complex bipartite matching tasks with incomplete information

Example: Data categorization or classifications

Political Expression of Academics on Twitter

Prashant Garg[1] and Thiemo Fetzer[2,3]

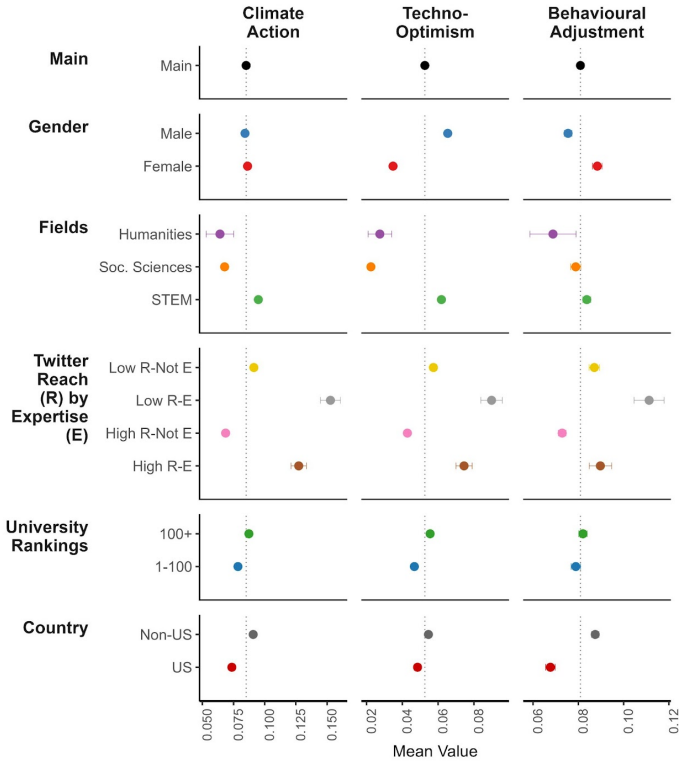
[1]Department of Economics and Public Policy, Imperial College London, London, United Kingdom

[2]Department of Economics, University of Warwick, Coventry, United Kingdom

[3]Department of Economics, University of Bonn, Bonn, Germany

Abstract

Academics play a vital role in the generation and dissemination of knowledge, ideas and narratives. Social media provide new, more direct ways of science communication. Yet, since not all academics engage with social media, the sample that does so may have an outsize influence on shaping public perceptions of academia through the set topics they engage with and their style and tone of communication. We describe patterns in academics' expression online using an international dataset covering nearly 100,000 scholars linking their Twitter content to academic records. We document large and systematic variation in politically salient academic expression concerning climate action, cultural, and economic concepts. We show that US academics often diverge from the US Twitter population at large in topic focus and style, although academics are not necessarily more extreme in their beliefs. Future work should examine potential impacts on public trust and the reasons why academics express themselves politically on social media.



→ forthcoming in *Nature: Human Behavior* with interactive website on `academicexpression.online`.

Example: Dataset construction

AI-Generated Production Networks: Measurement and Applications to Global Trade

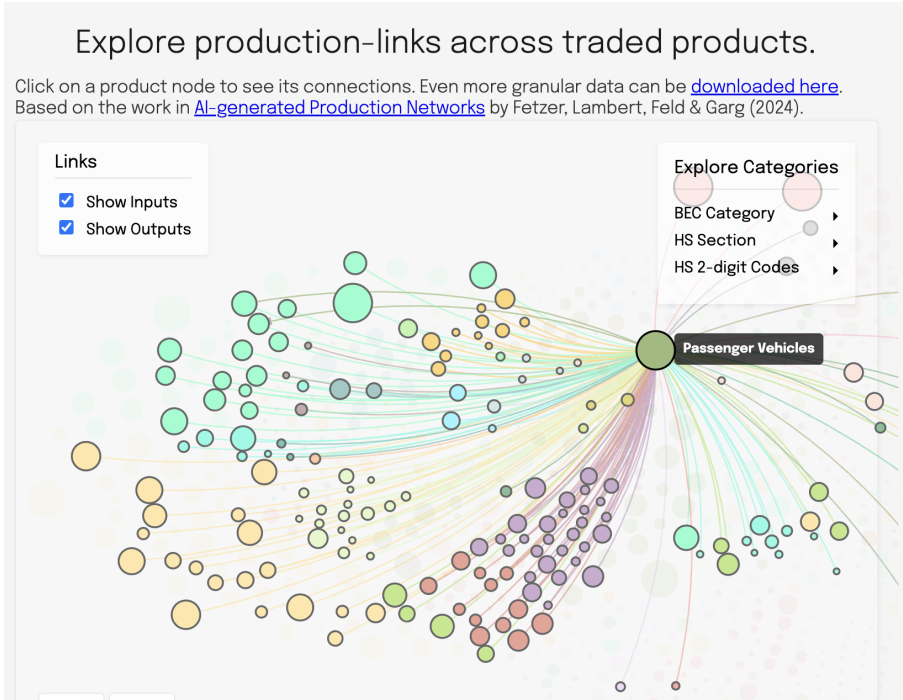
Thiemo Fetzer*
Peter John Lambert†
Bennet Feld‡
Prashant Garg§

November 18, 2024

Abstract

This paper leverages generative AI to build a network structure over 5,000 product nodes, where directed edges represent input-output relationships in production. We layout a two-step 'build-prune' approach using an ensemble of prompt-tuned generative AI classifications. The 'build' step provides an initial distribution of edge-predictions, the 'prune' step then re-evaluates all edges. With our AI-generated Production Network (AIPNET) in toe, we document a host of shifts in the network position of products and countries during the 21st century. Finally, we study production network spillovers using the natural experiment presented by the 2017 blockade of Qatar. We find strong evidence of such spill-overs, suggestive of on-shoring of critical production. This descriptive and causal evidence demonstrates some of the many research possibilities opened up by our granular measurement of product linkages, including studies of on-shoring, industrial policy, and other recent shifts in global trade.

Keywords: Supply-Chain Network Analysis, Large Language Models, On-shoring, industrial policy, Trade wars, Econometrics-of-LLMs
JEL Classification: F14, F23, L16, F52, O25, N74, C81



→ presented to policy audiences and Ministries with interactive website on `aipnet.io`.

Example: Retrieval augmented dataset construction

Causal Claims in Economics*

Prashant Garg Thiemo Fetzer †

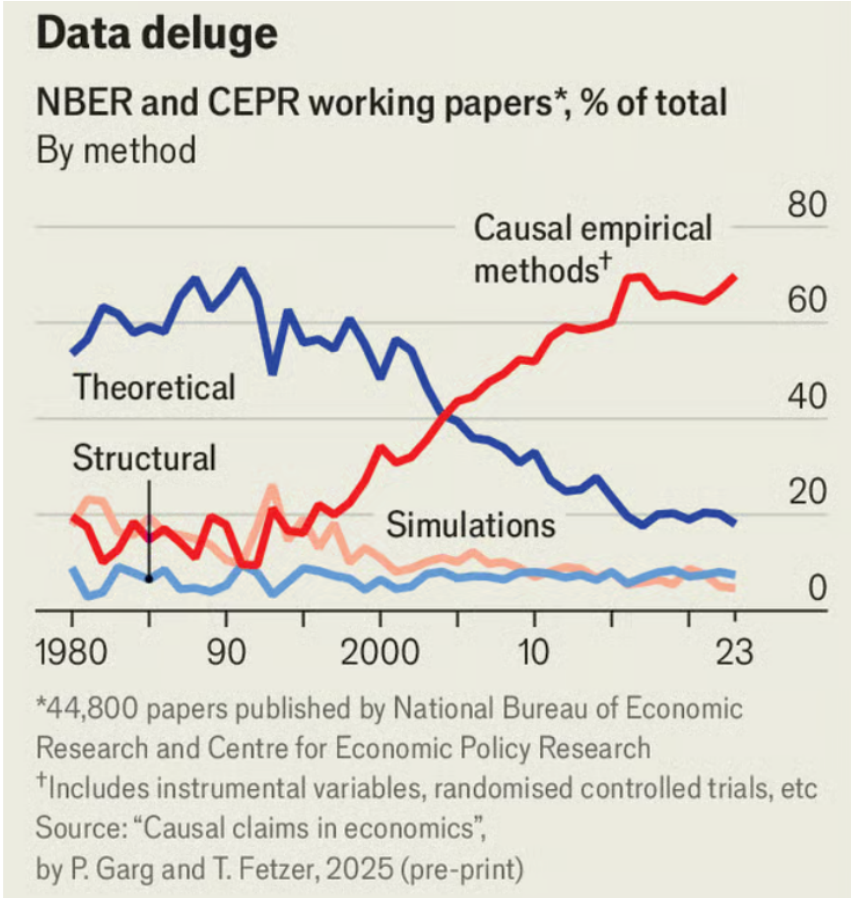
January 14, 2025

Click here for the latest version

Abstract

We analyze over 44,000 NBER and CEPR working papers from 1980–2023 using a custom language model to construct knowledge graphs that map economic concepts and their relationships. We distinguishes between general claims and those documented via causal inference methods (e.g., DiD, IV, RDD, RCTs). We document a substantial rise in the share of causal claims—from roughly 4% in 1990 to nearly 28% in 2020—reflecting the growing influence of the "credibility revolution." We find that causal narrative complexity (e.g., the depth of causal chains) strongly predicts both publication in top-5 journals and higher citation counts, whereas non-causal complexity tends to be uncorrelated or negatively associated with these outcomes. Novelty is also pivotal for top-5 publication, but only when grounded in credible causal methods: introducing genuinely new causal edges or paths markedly increases both the likelihood of acceptance at leading outlets and long-run citations, while non-causal novelty exhibits weak or even negative effects. Papers engaging with central, widely recognized concepts tend to attract more citations, highlighting a divergence between factors driving publication success and long-term academic impact. Finally, bridging under-explored concept pairs is rewarded primarily when grounded in causal methods, yet such gap filling exhibits no consistent link with future citations. Overall, our findings suggest that methodological rigor and causal innovation are key drivers of academic recognition, but sustained impact may require balancing novel contributions with conceptual integration into established economic discourse.

Keywords: KNOWLEDGE GRAPH, CREDIBILITY REVOLUTION, CAUSAL INFERENCE, NARRATIVE COMPLEXITY, NOVELTY, LARGE LANGUAGE MODELS
JEL Classification: A10, B41, C18, C80, D83



$\rightarrow$ rethinking how we represent papers

Example: Reasoning tasks

Has polarization accelerated or has *nuance* increased? *

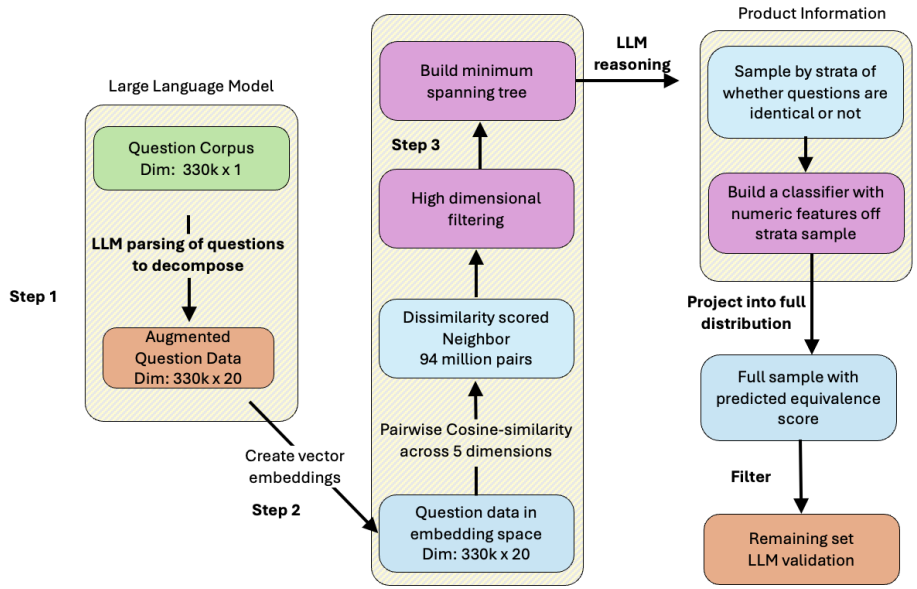
Thiemo Fetzer †

May 28, 2025

Abstract

What do policy makers want to know, and when? This paper leans on 4000+ public opinion polling databases carried out from the 1940s to 2020. It develops a pipeline that aligns opinion polling questions along with the underlying micro data using vector embeddings, semantic search and reasoning models to arrive at a large sample of opinion polling questions that have been consistently asked across different decades. Using this data, I document that the relative speed of partisan polarization – across broad set of time-varying and time-invariant issues – has drastically increased since the 2000s. Using a set of econometric exercises, this is causally attributed to the rapid rollout of social media. Yet, exploration *within issue* highlights a high degree of *nuance* in the extent of polarization that may be masked in popular debate, possibly highlighting that this increase in polarization may be much more driven by *perceptions*.

Keywords: PUBLIC OPINION, POLARIZATION, SOCIAL MEDIA, ARTIFICIAL INTELLIGENCE APPLICATIONS



→ agentic AI with reasoning capabilities to align 4000+ opinion polling micro datasets

Plan

How humans process information

Language Modeling Basics

Embeddings as workhorse for large language models

LLM Building Blocks

Logistic Regression as a Neural Network

Developing scalable classifiers

Working with LLMs

Heap's Law

Heaps' law (also called Herdan's law) is an empirical relationship which describes the number of *distinct words* in a document (or set of documents) as a function of the *document length* (so called type-token relation). It can be formulated as

$$|V| = kN^{\beta}$$

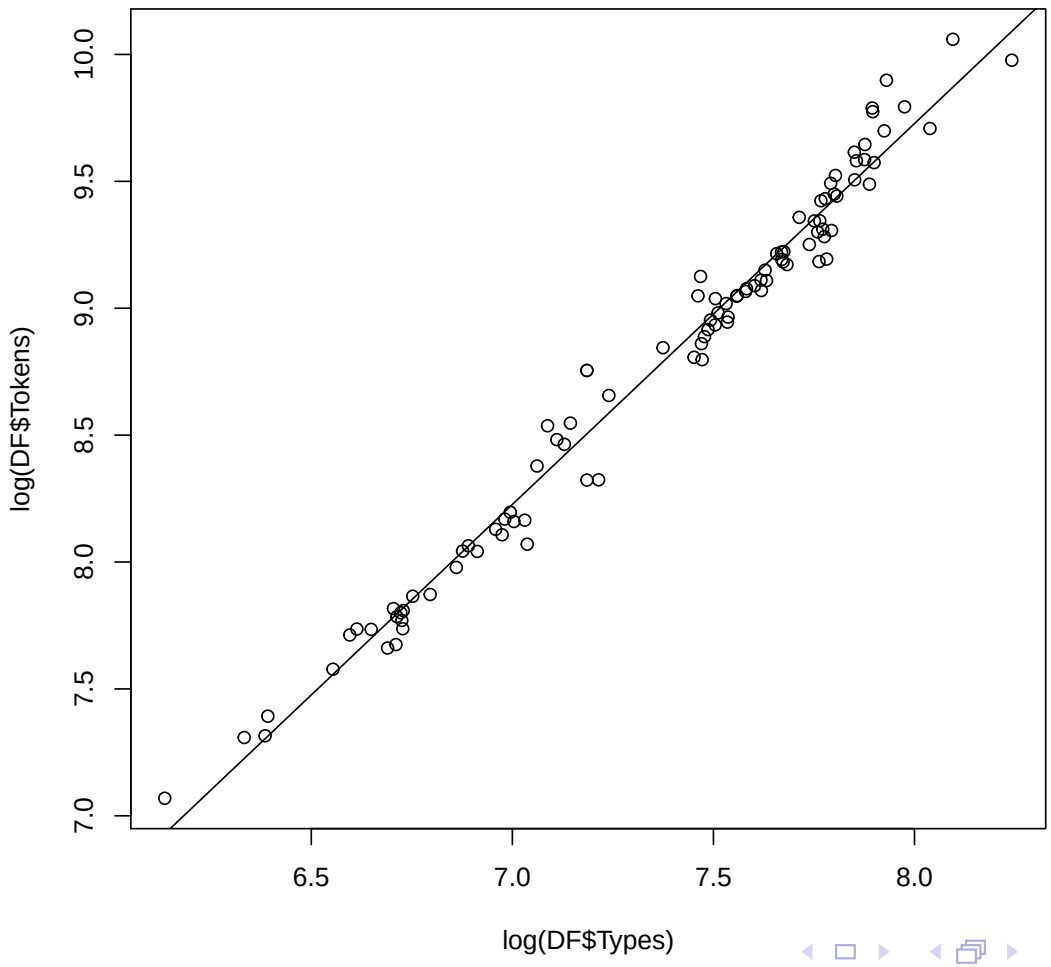
In log-log form, this power law becomes a straight line

$$\log(|V|) = k + \beta \log(N)$$

where $|V|$ is the size of the vocabulary (the number of types) and N is the number of tokens.

Illustration of Heap's Law in State of Union speeches

```
library(quanteda)
library(data.table)
data(SOTUCorpus, package = "quantedaData")
DF <- summary(SOTUCorpus)
plot(log(DF$Types), log(DF$Tokens)) + abline(lm(log(DF$Tokens) ~ log(DF$Types)))
```



Zipf's Law

Zipf's Law is a law about the frequency distribution of words *within a document*.
Zipf's Law states that s frequency of any word is inversely proportional to its rank in the frequency table.
Formally: Word frequency

$$f = \frac{a}{r^b}$$

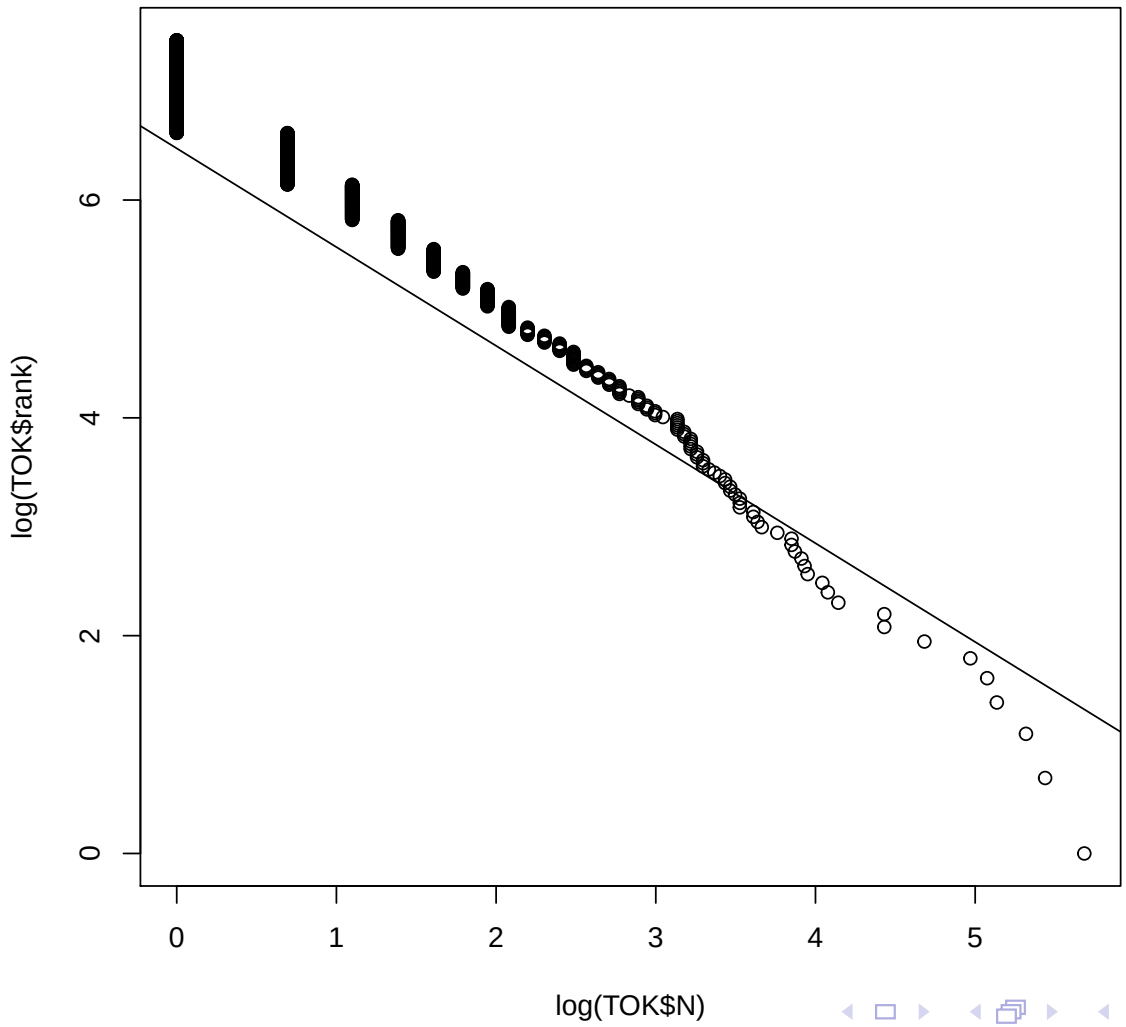
where r is the rank in the (empirical) word frequency distribution.
Again, logging

$$\log(f) = \log(a) - b\log(r)$$

Illustration of Zipf's Law

```
plot(log(TOK$N), log(TOK$rank)) + abline(lm(log(TOK$N) ~ log(TOK$rank)))

## numeric(0)
```



Implications of Heap's and Zipf's Law

- *Heap's Law* and *Zipf's Law* imply that data matrices constructed from text data is very *sparse*.
- Sparsity implies that there would be many zeroes.
- Most data processing steps for text data involve *densifying* the word frequency distribution.
- We next discuss a range of steps commonly used to densify.

Probabilistic Language Models

What is the likely next word?

```
Make America ...
```

N-gram language models see sentences as sequences of words, the occurrence of each word is a function of the likelihood of the sequence of words.

```
Make America Great Again
```

The predicted next word naturally depends on the corpus on which a language model was trained on and a range of other factors. Lets formalize things a bit.

Probabilistic Language Models

What is the probability of:

$$P(\text{again}|\text{Make America great})$$

We can express this probability as:

$$P(\text{again}|\text{Make America great}) = \frac{P(\text{Make America great again})}{P(\text{Make America great})}$$

which we may be inclined to estimate as

$$\hat{P}(\text{again}|\text{Make America great}) = \frac{C(\text{Make America great again})}{C(\text{Make America great})}$$

where the $C(.)$ indicates the raw counts of the text fragments.

A bit of notation...

What is the joint probability of observing a sequence of words $w_1, ..., w_n$?

$$
\begin{aligned}
P(w_1, .., w_n) &= P(w_1)P(w_2, ..., w_n|w_1) \\
&= P(w_1)P(w_2|w_1)P(w_3, ..., w_n|(w_1, w_2)) \\
&= P(w_1)P(w_2|w_1)P(w_3|(w_1, w_2))P(w_4, ..., w_n|(w_1, w_2, w_3)) \\
&... \\
&= \prod_{k=1}^{n} P(w_k|(w_1, ..., w_{k-1})
\end{aligned}
$$

iteratively applying the **Chain Rule of Probability**.

Curse of Dimensionality

We can compute probability of a sentence

$$P(w_1, .., w_n) \quad = \quad \prod_{k=1}^{n} P(w_k | (w_1, ..., w_{k-1})$$

by multiplying a sequence of conditional probabilities.

- We can not estimate each individual conditional probability because its highly unlikely that a stable estimate does exist.
- Similarly, it would be computationally infeasible.
- Parameter space (number of conditional probabilities that need to be estimated) grows exponentially in n.

N-gram models

A bigram model defined as

$$P(w_1, ..., w_n) = \prod_{k=1}^{n} P(w_k | w_{k-1})$$

This assumption that the probability of a word depends only on the previous word is called the **Markov** assumption. Here we approximate $P(w_k|(w_1, ..., w_{k-1})) \approx P(w_k|w_{k-1})$
Markov models are a class of probabilistic models that assume that the future can be predicted without looking *too far* into the past.
We can generalize to N-gram models

$$P(w_1, ..., w_n) = \prod_{k=1}^{n} P(w_k | w_{k-1}, ..., w_{k-N+1})$$

where $P(w_k|(w_1, ..., w_{k-1})) \approx P(w_k|w_{k-1}, w_{k-2}, ...w_{k-N+1})$

What do we learn?

- N-gram models capture syntactic features and general knowledge (here, knowledge about the underlying speeker)
- It turns out that linguistic features such as word sequences "I want" are reasonably frequent and they capture linguistic features: verbs tend to follow subject as indicated by "I".
- N-gram models can be trained by counting and normalization

Trade-off: Higher order N-gram versus lower order N-grams

Unigram Trump Babbler

```
## [1] "prosperity not talking about the game and more you mean but ill tell you know shes going t"
## [2] "o terminate obamacare with a lot its deadly totally destabilized the worst human being tre"
## [3] "ated me so im not smart and tell you get on trade agreement you look we have to end "
```

Bigram Trump Babbler

```
## [1] "from pakistan and he was talking to some place it costs 3 billion and i said ok then i hav"
## [2] "e to happen with these two nations and must regard them with their families we mourn as on"
## [3] "e united people with force purpose and determination but the muslims living in this "
```

Trigram Trump Babbler

```
## [1] "permanently admits more than 100000 immigrants from the middle east our government has bee"
## [2] "n admitting ever growing numbers year after year without any effective plan for our own se"
## [3] "curity in fact clintons state department was in charge of admissions and the admissions pr"
## [4] "ocess for people applying to enter from overseas "
```

Quadrigram Trump Babbler

```
## [1] "tough as nails hes going to be your champion im going to be good for womens health issues "
## [2] "its very important to me it was instructional when i did the art of the deal pac after the"
## [3] " book they have all these pacs and the money comes in and its "
```

as N increases, the number of parameters to be estimated explodes. In addition, as we have seen, the matrices are very sparse - many zeroes!

Curse of Dimensionality of N-gram model

Suppose you have a vocabulary of size $|V|$. Assuming no constraints imposed by language structure. How many different conditional probabilites are there to estimate?

- There are $|V|$ sentences, containing exactly 1 words.
- There are $|V| \times |V|$ sentences containing 2 words
- There are $|V| \times |V| \times |V|$ sentences containing 3 words

In total there are $|V|^N$ parameters in an n-gram for vocabulary size $|V|$.

Babbling Donald Trump

```
library(ngram)
TRUMP <- readLines(con = "../../Data/Trump-Speeches.txt")
# concatenate into one massive string, remove empty lines
TRUMP <- TRUMP[-grep("^SPEECH|^$", TRUMP)]
TRUMP <- gsub(" +", " ", TRUMP)
TRUMP <- paste(TRUMP, collapse = " ")
TRUMP <- ngram(TRUMP, n = 3)
str_break(babble(TRUMP, genlen = 35, seed = 130))
```

```
## [1] "I have as big a heart as anybody. We want to win Iowa, folks. Because look, I love the poo"
## [2] "rly educated. We're the smartest people, we're the most loyal people by far. Everybody say"
## [3] "s it. "
```

Plan

How humans process information

Language Modeling Basics

Embeddings as workhorse for large language models

LLM Building Blocks

Logistic Regression as a Neural Network

Developing scalable classifiers

Working with LLMs

How is text represented in large language models?

`Embeddings` are the workhorse of large language models – they provide **context aware** representation of text.

Distributional Hypothesis: "You shall know a word by the company it keeps." Word2Vec operationalizes this:

- Words that appear in similar contexts should have similar vectors.
- Training task: Given a word, predict its context (Skip-gram), or given context, predict the word (CBOW).

Result:

- A dense vector space where semantic similarity is geometric proximity.

Deriving the Skip-Gram Objective

General idea: We model a sequence of words $w_1, w_2, \ldots, w_T$.

- Suppose for a given center word w_t, our model generates *all* context words:

$$P(w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c} \mid w_t)$$

- We make a **conditional independence assumption**:

$$P(\text{context} \mid w_t) = \prod_{\substack{-c \leq j \leq c \\ j \neq 0}} P(w_{t+j} \mid w_t)$$

- Then, the full corpus likelihood becomes:

$$P(w_1, \ldots, w_T) \approx \prod_{t=1}^{T} \prod_{\substack{-c \leq j \leq c \\ j \neq 0}} P(w_{t+j} \mid w_t)$$

- Each word contributes $2 \times c$ training pairs.

A simple example with $c = 1$

Illustrating for ease of less notification clutter with $c = 1$:

- Suppose for *any given center word* w_t, our model generates *all* context words:

$$P(w_{t-1}, w_{t+1} \mid w_t)$$

  To train this we would need to know $P(w_{t-1}, w_t, w_{t+1})$ for all arrangements of words.

- So we only worry about a model of the *joint* distribution *within context* window $c = 1$.
- We make a **conditional independence assumption**:

$$P(w_{t-1}, w_{t+1} \mid w_t) = P(w_{t-1} \mid w_t)P(w_{t+1} \mid w_t)$$

- Then, the full corpus likelihood becomes:

$$P(w_1, \ldots, w_T) \approx \prod_{t=1}^{T} P(w_{t-1} \mid w_t)P(w_{t+1} \mid w_t)$$

From General Likelihood to Softmax Modeling

Take logs to convert products into sums:
The general Skip-Gram objective:

$$\log \mathcal{L} = \sum_{t=1}^{T} \sum_{\substack{-c \le j \le c \\ j \neq 0}} \log P(w_{t+j} \mid w_t)$$

Key question: How do we model $P(w_O \mid w_I)$?

Idea:

- Represent each word with a vector $v_w \in \mathbb{R}^d$ (input) and $v'_w \in \mathbb{R}^d$ (output).
- Make $P(w_O \mid w_I)$ large when v_{w_I} and v'_{w_O} are similar.
- Use the dot product to measure similarity.

From General Likelihood to Softmax Modeling

Model: Softmax over all vocabulary words:

$$P(w_O \mid w_I) = \frac{\exp({v'_{w_O}}^\top v_{w_I})}{\sum_{w=1}^{V} \exp({v'_w}^\top v_{w_I})}$$

where vector $v_w \in \mathbb{R}^d$ (input) and $v'_w \in \mathbb{R}^d$ (output). We get a $P(w_O \mid w_I)$ for every word in the vocabulary V.

We **interpret** the dot-product as a measure of *similarity*:

$${v'_w}^\top v_{w_I}$$

Why?

Link to Cosine Similarity

The dot product between two vectors a and b is:

$$a^\top b = \|a\|\|b\| \cos(\theta)$$

where θ is the angle between a and b.
Thus, the dot product is a **linear transformation** of the cosine similarity, scaled by the vector norms, i.e..

$$v'_w{}^\top v_{w_I} \propto \cos(\theta)$$

And if we normalize embeddings such that $\|v'_{w_O}\| = \|v_{w_I}\| = 1$, then:

$$v'_{w_O}{}^\top v_{w_I} = \cos(\theta)$$

In this case, softmax scoring is directly proportional to the cosine similarity.

Modeling the Conditional Probability

Let:

- w_I = input (center) word
- w_O = output (context) word
- $v_{w_I} \in \mathbb{R}^{d \times 1}$ = "input" vector of w_I
- ${v'_{w_O}}^\top \in \mathbb{R}^{1 \times d}$ = "output" vector of w_O
- d = dimensionality of the embedding space (e.g., 100, 300)

The probability of observing w_O given w_I is modeled using a softmax over the vocabulary:

$$P(w_O \mid w_I) = \frac{\exp({v'_{w_O}}^\top v_{w_I})}{\sum_{w=1}^{V} \exp({v'_w}^\top v_{w_I})}$$

This assigns high probability when v'_{w_O} and v_{w_I} are similar (i.e., their dot product is high).

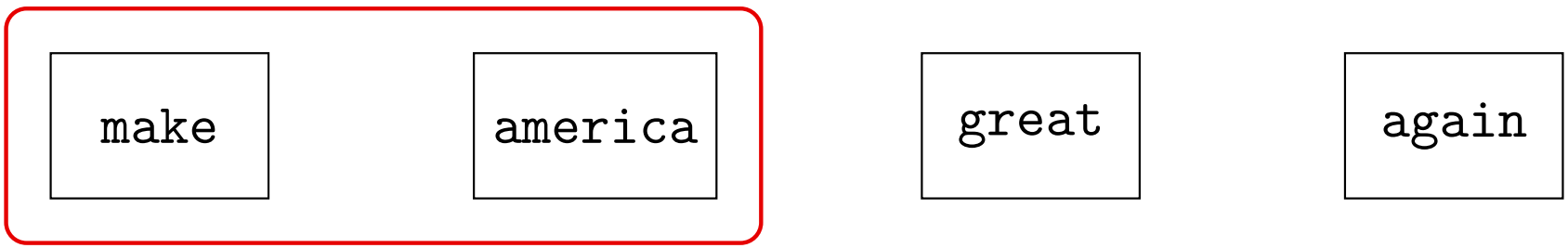
How Skip-Gram Generates Training Pairs

Sentence: `make america great again` **Window size = 1** (context of 1 word on either side)

Generated (center, context) pairs:

How Skip-Gram Generates Training Pairs

Sentence: `make america great again` **Window size = 1** (context of 1 word on either side)

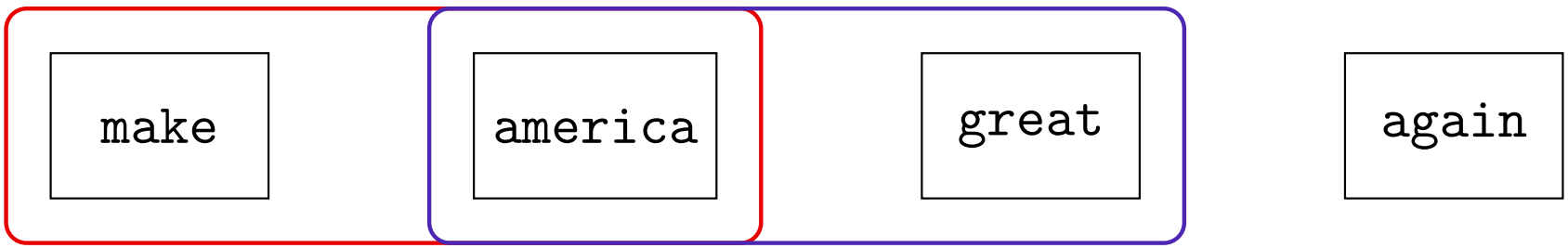


Generated (center, context) pairs:

- (make, america)

How Skip-Gram Generates Training Pairs

Sentence: `make america great again` **Window size = 1** (context of 1 word on either side)

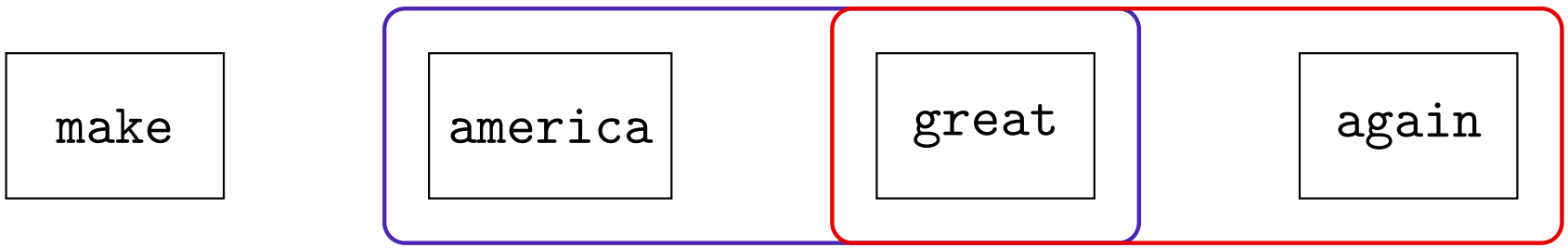


Generated (center, context) pairs:

- (make, america)
- (america, make), (america, great)

How Skip-Gram Generates Training Pairs

Sentence: `make america great again` **Window size = 1** (context of 1 word on either side)

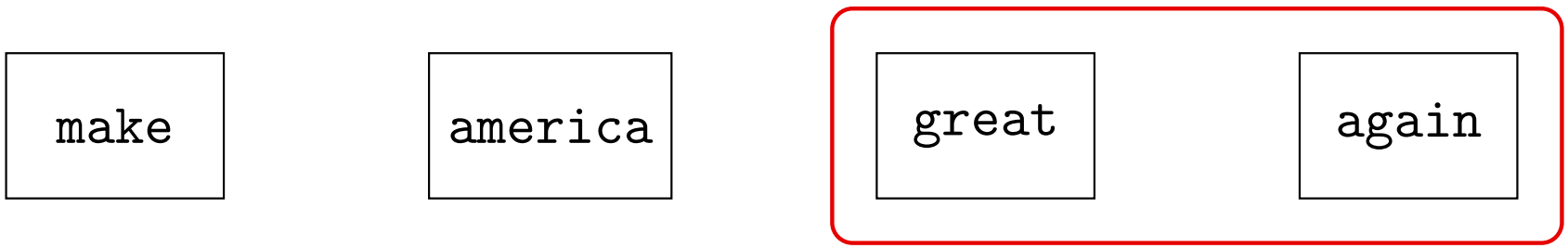


Generated (center, context) pairs:

- (make, america)
- (america, make), (america, great)
- (great, america), (great, again)

How Skip-Gram Generates Training Pairs

Sentence: `make america great again` **Window size = 1** (context of 1 word on either side)



Generated (center, context) pairs:

- (make, america)
- (america, make), (america, great)
- (great, america), (great, again)
- (again, great)

Gradient Descent in Skip-Gram

At each training step:

- Take each center word w_I and each context word w_O.
- Compute scores: $\text{score}_w = {v'_w}^\top v_{w_I}$ for all words w.
- Apply softmax to get probabilities.
- Compute loss: $-\log P(w_O|w_I)$.
- Compute gradient with respect to parameters.
- Update parameters using **stochastic gradient descent**.

Stochastic Gradient Descent to Train Embeddings

We now use the (center, context) pairs to train the embeddings.

- Each pair (`center = great, context = america`) contributes one term to the overall loss:

$$\log P(\texttt{america} \mid \texttt{great}) = \log \frac{\exp({v'_{\texttt{america}}}^{\top} v_{\texttt{great}})}{\sum_{w \in V} \exp({v'_{w}}^{\top} v_{\texttt{great}})}$$

- where V is indicating the total vocabulary and d is the embedding dimensionality.
- The loss for this pair is:

$$\mathcal{L}_{(w_I, w_O)} = -\log P(w_O \mid w_I)$$

- We next derive the first derivative:

$$\frac{\partial \mathcal{L}_{(w_I, w_O)}}{\partial v_{w_I}} = \sum_{w \in V} P(w \mid w_I) v'_w - v'_{w_O} \quad \in \mathbb{R}^d$$

Stochastic Gradient Descent to Train Embeddings

Derivation of the gradient with respect to v_{w_I}:

$$\begin{aligned}\mathcal{L}_{(w_I,w_O)} &= -\log \frac{\exp({v'_{w_O}}^\top v_{w_I})}{\sum_{w\in V}\exp({v'_w}^\top v_{w_I})} \\ &= -{v'_{w_O}}^\top v_{w_I} + \log\left(\sum_{w\in V}\exp({v'_w}^\top v_{w_I})\right)\end{aligned}$$

Differentiating:

$$\frac{\partial \mathcal{L}}{\partial v_{w_I}} = -v'_{w_O} + \sum_{w\in V} P(w \mid w_I) v'_w \in \mathbb{R}^d$$

This gradient nudges v_{w_I} toward v'_{w_O} and away from all other v'_w.
SGD update:

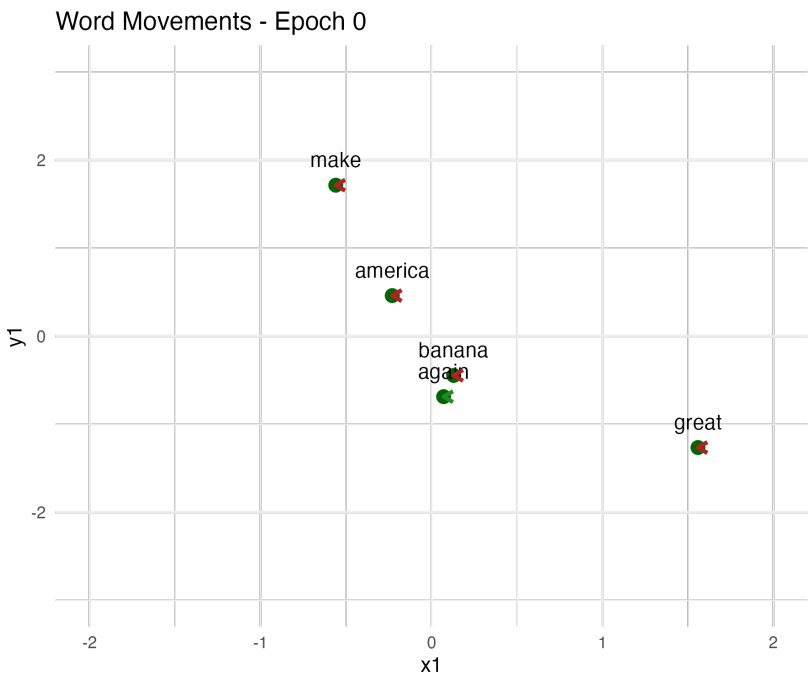
$$v_{w_I} \leftarrow v_{w_I} - \eta \frac{\partial \mathcal{L}_{(w_I,w_O)}}{\partial v_{w_I}} \quad (\text{in } \mathbb{R}^d)$$

Why Do We Need Gradient Descent?

- Vocabulary size $|V|$ often large
- The softmax function is nonlinear **non-convex optimization problem** with no closed solution
- Gradient descent provides an iterative method to find good embeddings
- Initialise with assigning random numbers to each vector $v_w \in \mathbb{R}$
- Predicting context words requires assigning a probability to every word in the vocabulary.
- Computing exact probabilities for each pair would be computationally intensive
- We need an efficient method to **adjust** the embeddings step-by-step:
  - Reduce the loss $-\log P(w_O|w_I)$.
  - Improve predictions over time.

Gradient descent provides a scalable way to *iteratively* improve embeddings.

Gradient Descent in Action

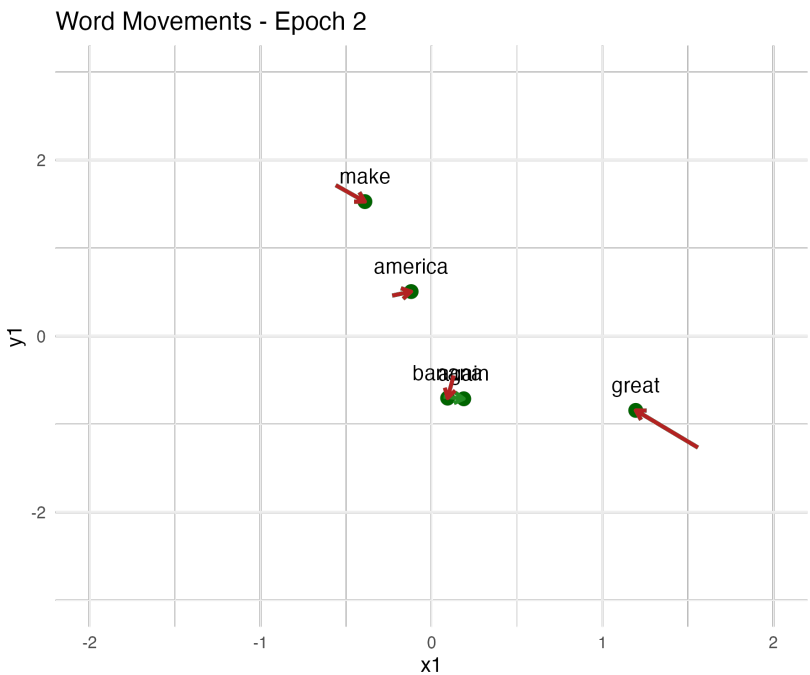


- Initially, embeddings are random and unstructured.
- Positive training pairs (e.g., `great`, `america`) are pulled closer.
- Negative pairs (e.g., `banana`, `great`) are pushed apart.
- Each update step mimics a gradient:

$$v_w \leftarrow v_w \pm \eta(v_{w'} - v_w)$$

- Arrows visualize how vectors shift in response to training pairs.

Gradient Descent in Action

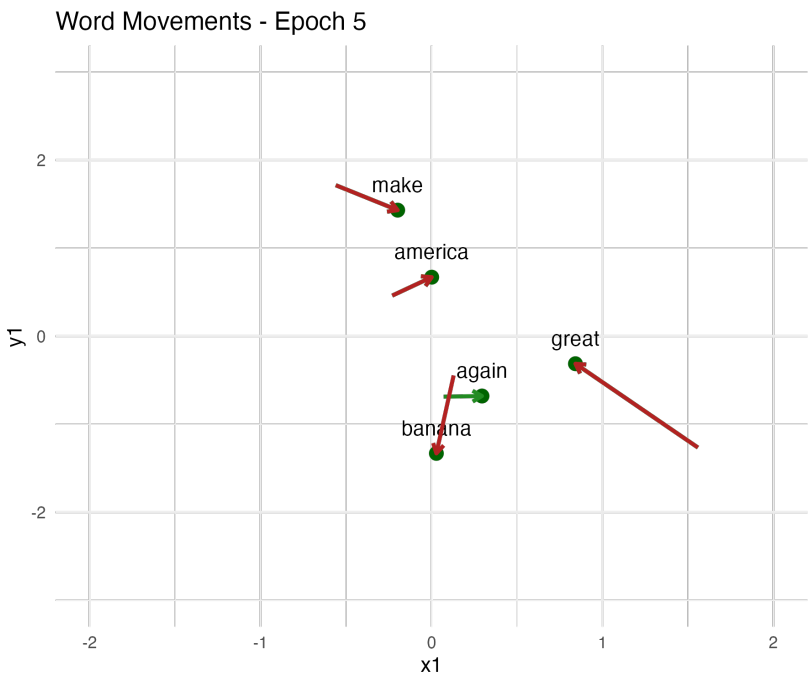


- Initially, embeddings are random and unstructured.
- Positive training pairs (e.g., `great`, `america`) are pulled closer.
- Negative pairs (e.g., `banana`, `great`) are pushed apart.
- Each update step mimics a gradient:

$$v_w \leftarrow v_w \pm \eta(v_{w'} - v_w)$$

- Arrows visualize how vectors shift in response to training pairs.

Gradient Descent in Action

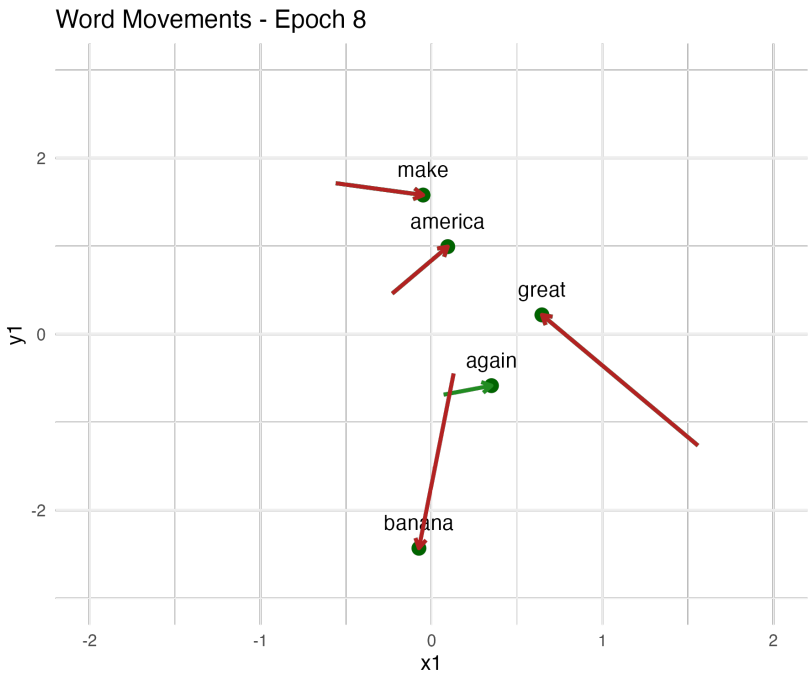


- Initially, embeddings are random and unstructured.
- Positive training pairs (e.g., `great`, `america`) are pulled closer.
- Negative pairs (e.g., `banana`, `great`) are pushed apart.
- Each update step mimics a gradient:

$$v_w \leftarrow v_w \pm \eta(v_{w'} - v_w)$$

- Arrows visualize how vectors shift in response to training pairs.

Gradient Descent in Action

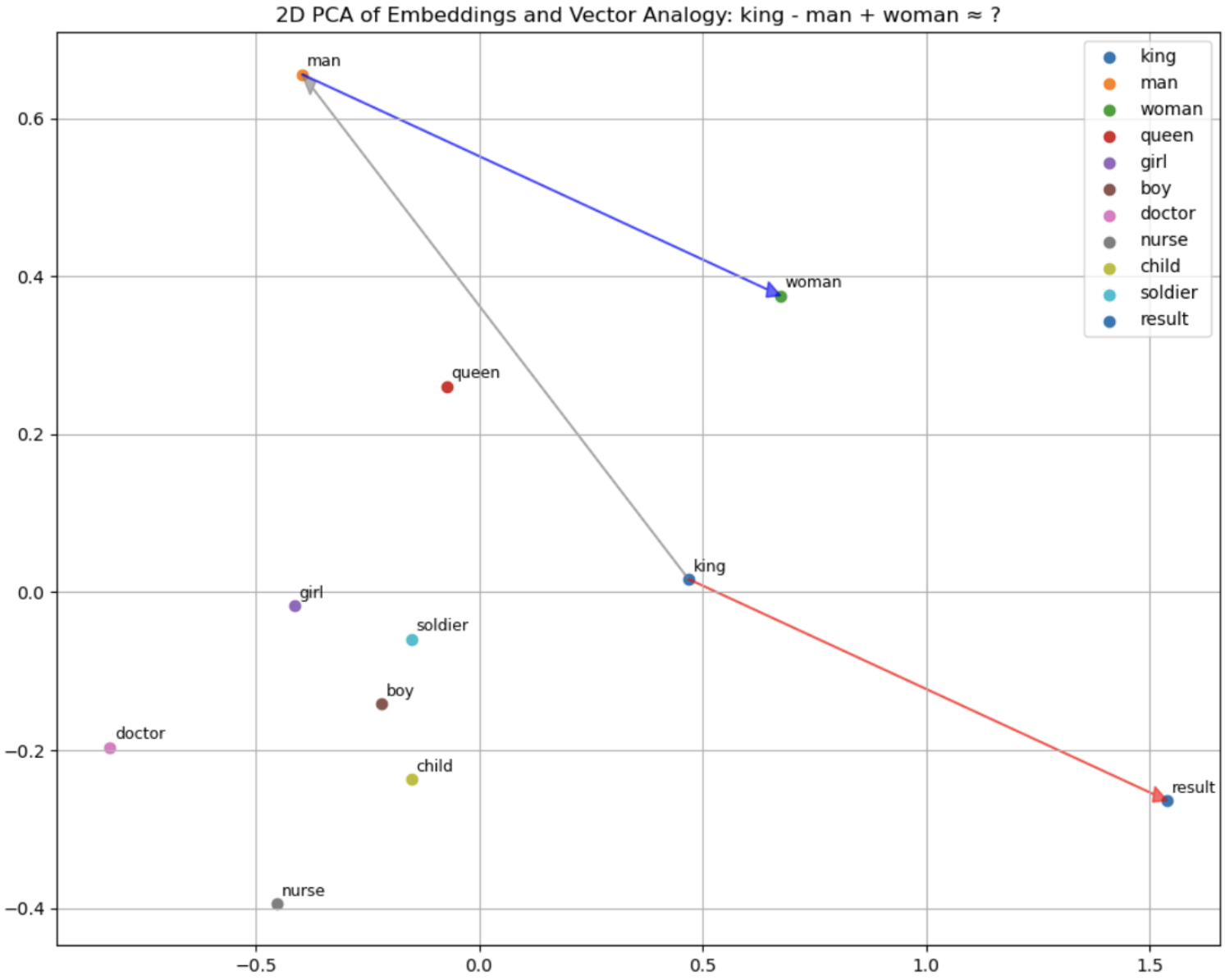


- Initially, embeddings are random and unstructured.
- Positive training pairs (e.g., `great`, `america`) are pulled closer.
- Negative pairs (e.g., `banana`, `great`) are pushed apart.
- Each update step mimics a gradient:

$$v_w \leftarrow v_w \pm \eta(v_{w'} - v_w)$$

- Arrows visualize how vectors shift in response to training pairs.

Visualisation of Word Embeddings via PCA

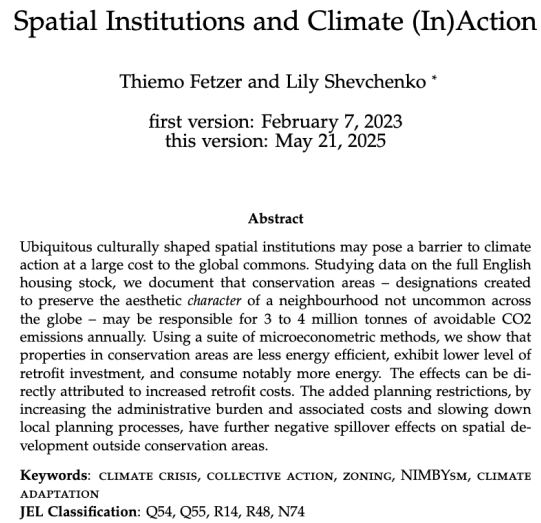


Embeddings and Semantic Search

- Embeddings are the workhorse to most RAG applications

  Example: `planning applications` for local councils across the UK
- Expanding work with Lily Shevchenko on *Spatial Institutions and Climate (In)Action*, we are building a large database of planning applications

  Study how individual spatial institutions constrain e.g. housing development



Spatial Institutions and Climate (In)Action

Thiemo Fetzer and Lily Shevchenko *

first version: February 7, 2023
this version: May 21, 2025

Abstract

Ubiquitous culturally shaped spatial institutions may pose a barrier to climate action at a large cost to the global commons. Studying data on the full English housing stock, we document that conservation areas – designations created to preserve the aesthetic *character* of a neighbourhood not uncommon across the globe – may be responsible for 3 to 4 million tonnes of avoidable CO2 emissions annually. Using a suite of microeconometric methods, we show that properties in conservation areas are less energy efficient, exhibit lower level of retrofit investment, and consume notably more energy. The effects can be directly attributed to increased retrofit costs. The added planning restrictions, by increasing the administrative burden and associated costs and slowing down local planning processes, have further negative spillover effects on spatial development outside conservation areas.

Keywords: CLIMATE CRISIS, COLLECTIVE ACTION, ZONING, NIMBYSM, CLIMATE ADAPTATION
JEL Classification: Q54, Q55, R14, R48, N74

Example of planning applications

Planning – Application Summary

Help with this page

PA/19/02266/NC | Proposed installation of 6 no antennas, 1 no 600mm dish, 1 no 300mm dish, 2 no flatpack frames, 1 no power cabinet and 1 no meter cabinet at ground level together with ancillary development. | Lariat Court, 34 Nellie Cressall Way, London E3 4RQ

Save search | Refine search | Track | Print

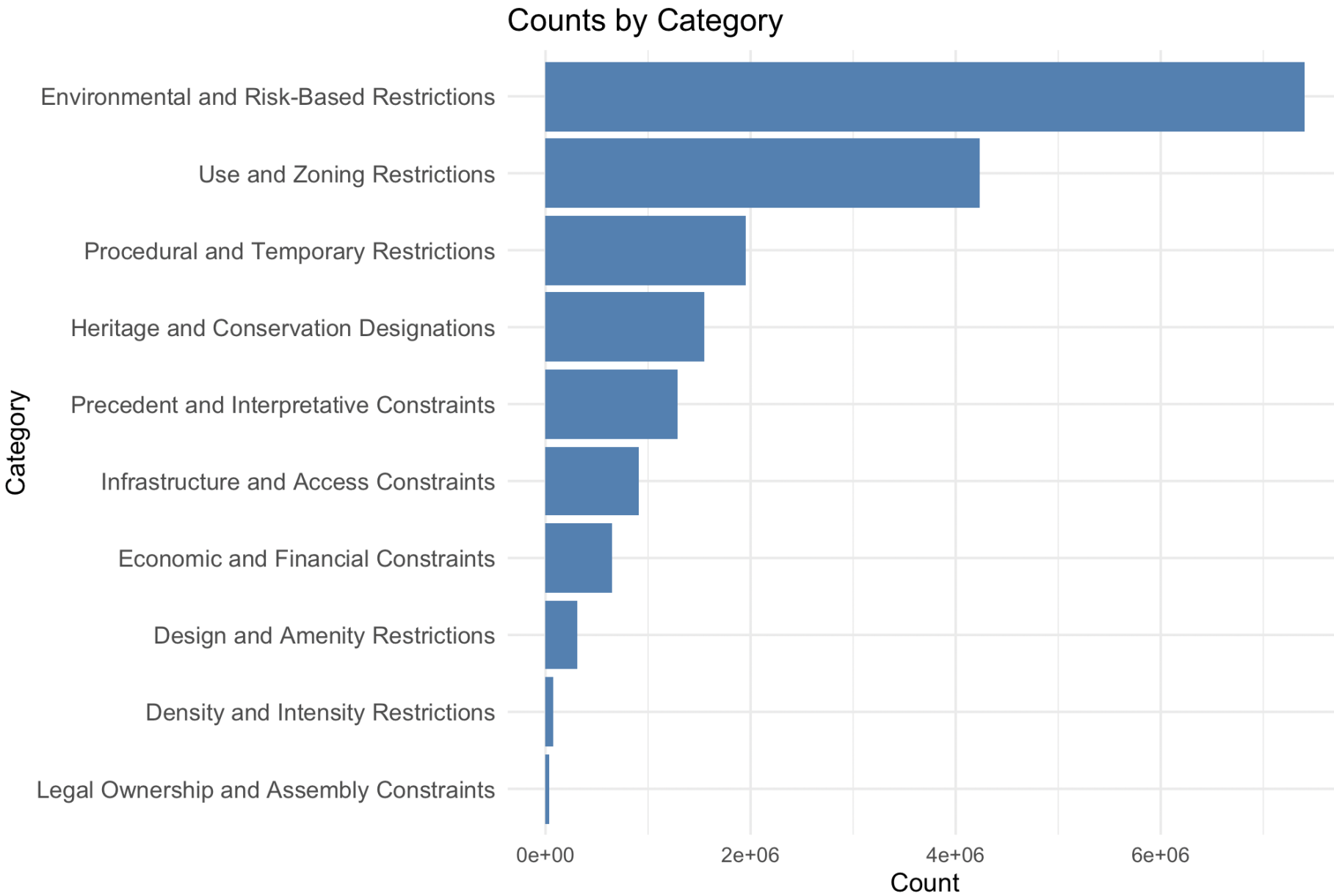
Details | Comments (1) | Documents (10) | Map | Related Cases (0)

Summary | Further Information | Important Dates

Reference	PA/19/02266/NC
Application Validated	Fri 25 Oct 2019
Address	Lariat Court, 34 Nellie Cressall Way, London E3 4RQ
Proposal	Proposed installation of 6 no antennas, 1 no 600mm dish, 1 no 300mm dish, 2 no flatpack frames, 1 no power cabinet and 1 no meter cabinet at ground level together with ancillary development.
Status	Decided
Decision	Permit
Decision Issued Date	Wed 18 Dec 2019
Appeal Decision	Not Available

put together database of 10 million + planning applications, together with their council discussions and minutes to understand NIMBYism.

Want to understand the impact of different constraints



Embeddings and semantic search

- A sample of around 9.5 million planning applications each with short description
- Use semantic search and matching to identify *identical* planning applications subject to different *constraints*
- This requires creating creating pairwise comparisons of the application in embedding space
- `9.5 million documents x 1024 embedding dimensionality x 4 bytes for 32-bit float precision = 35 GB`
- Quite often, building embeddings on the fly is not feasible for applications but will get back to this later

Constructing matching estimator in semantic space

For each planning application...

- Find a comparison planning application in the same `category`
- In the same `planing authority`
- In the same `year`
- That is most `similar` in terms of the planning text

$\rightarrow$ distinguish impact of different place-based restrictions on planning process

What is the impact of different *stack* of spatial constraints?

```
> etable(list(r1,r2,r3))
                                                  model 1             model 2             model 3
Dependent Var.:                             obstructive_dum     obstructive_dum     obstructive_dum

Constant                                 0.1516*** (0.0079)
heritage_and_conservation_designations 0.0474*** (0.0094) 0.0198*** (0.0031) 0.0198*** (0.0031)
Fixed-Effects:                           ----------------- ----------------- -----------------
pair_id                                                 No                 Yes                 Yes
predicted_main_type                                     No                  No                 Yes
________________________________________ _________________ _________________ _________________
S.E.: Clustered                                 by: plan_auth       by: plan_auth       by: plan_auth
Observations                                        1,252,065           1,252,065           1,252,065
R2                                                    0.00264             0.65649             0.65651
Within R2                                                  --             0.00045             0.00045
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

$\rightarrow$ observe that heritage based restrictions *increase* obstructions in the planning process

But how do different constraints *interact*

The same planning application may be **treated** by a bundle of different constraints.

If there are 10 different constraint types, you can have

- $\binom{10}{1}$ ways of selecting a single constraint

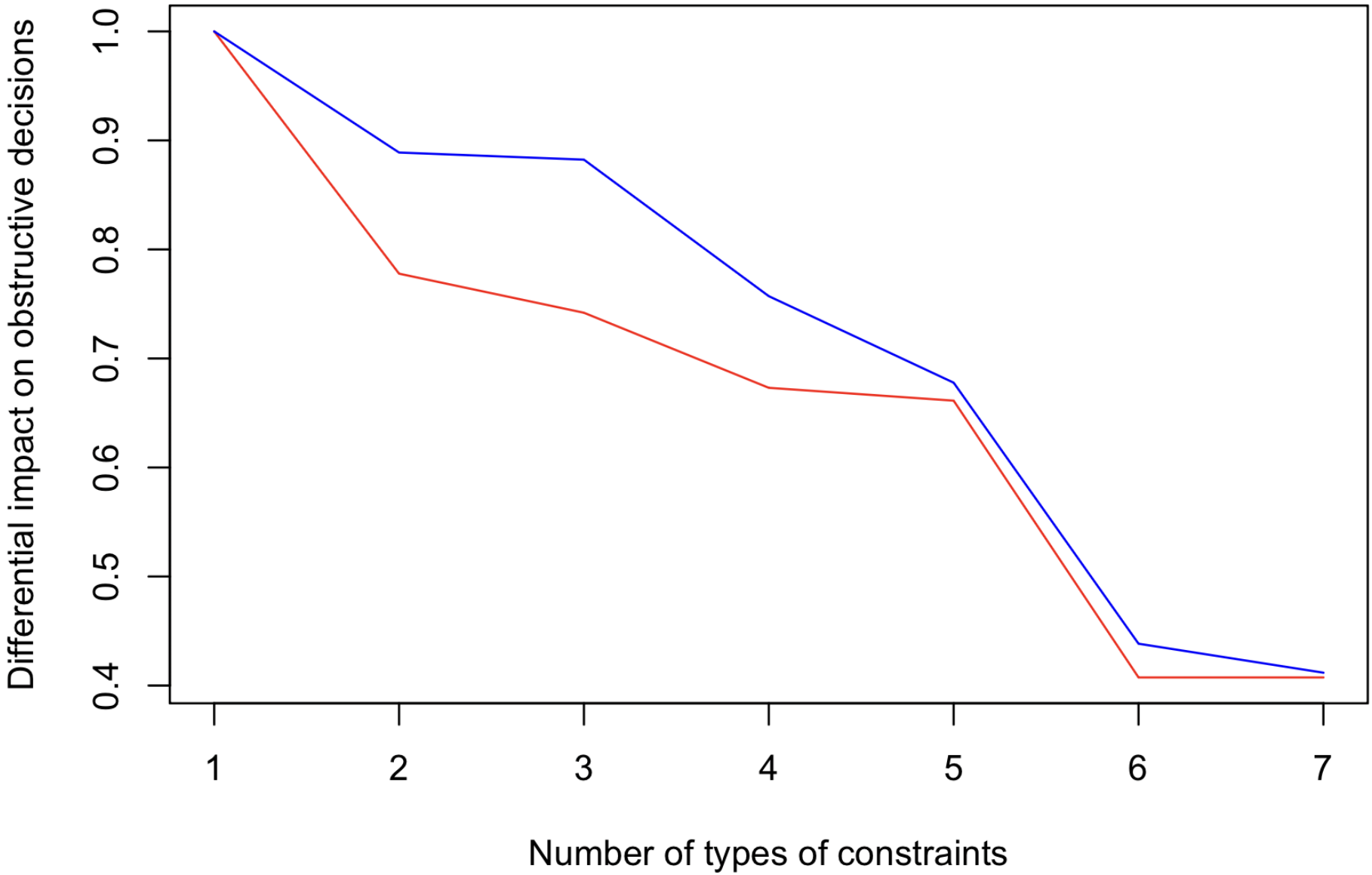
  `Heritage and Conservation Designations,Infrastructure and Access, ...`

- $\binom{10}{2}$ ways of having two constraints

  `Heritage and Conservation Designations + Environmental and Risk-Based Restrictions`

- ...
- $\binom{10}{10} = 1$ way of having **all constraints**

But how do different constraints *interact*



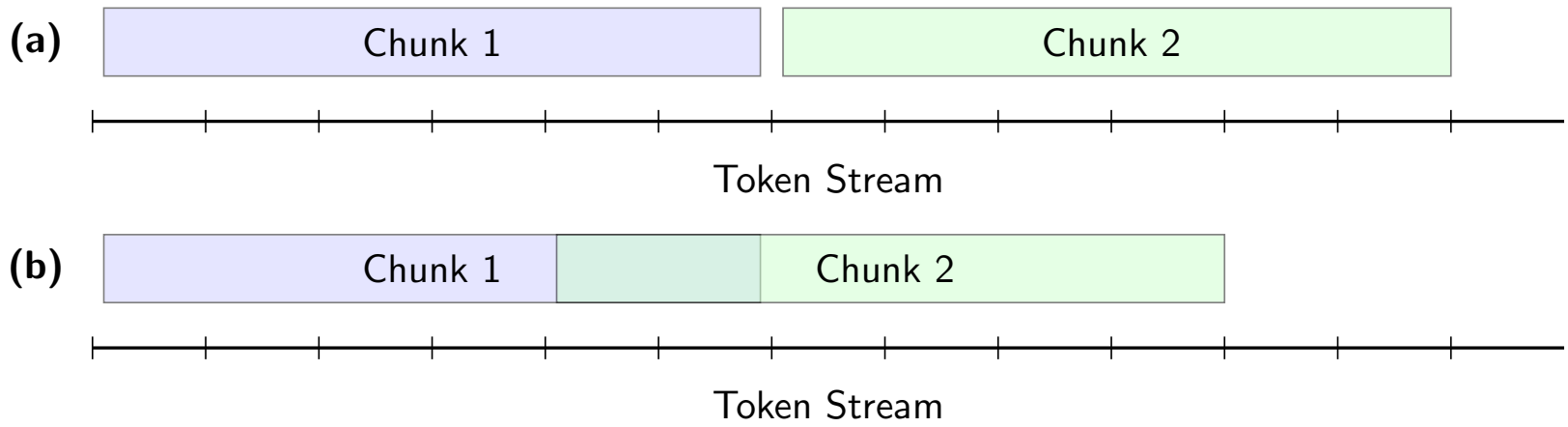
How do you work with `embeddings` day in day out?

Showcase **Anything LLM**

Working with Text Embeddings: Chunking Strategy

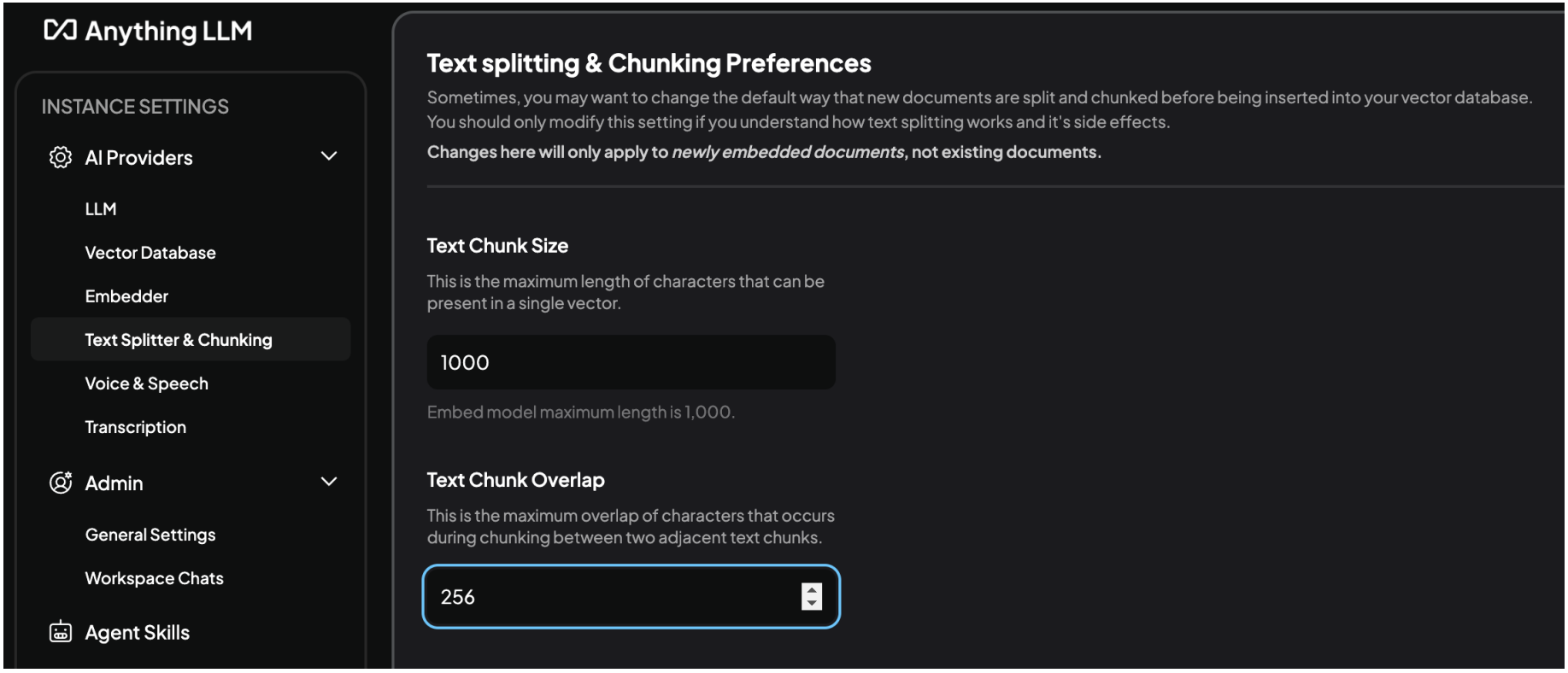
Given limitations to context windows and often large datasets it still is necessary to work with embeddings. Use text chunking to split into manageable sizes.

- `chunk_size` (e.g., 300–1000 tokens)
- `chunk_overlap` (e.g., 50–200 tokens)

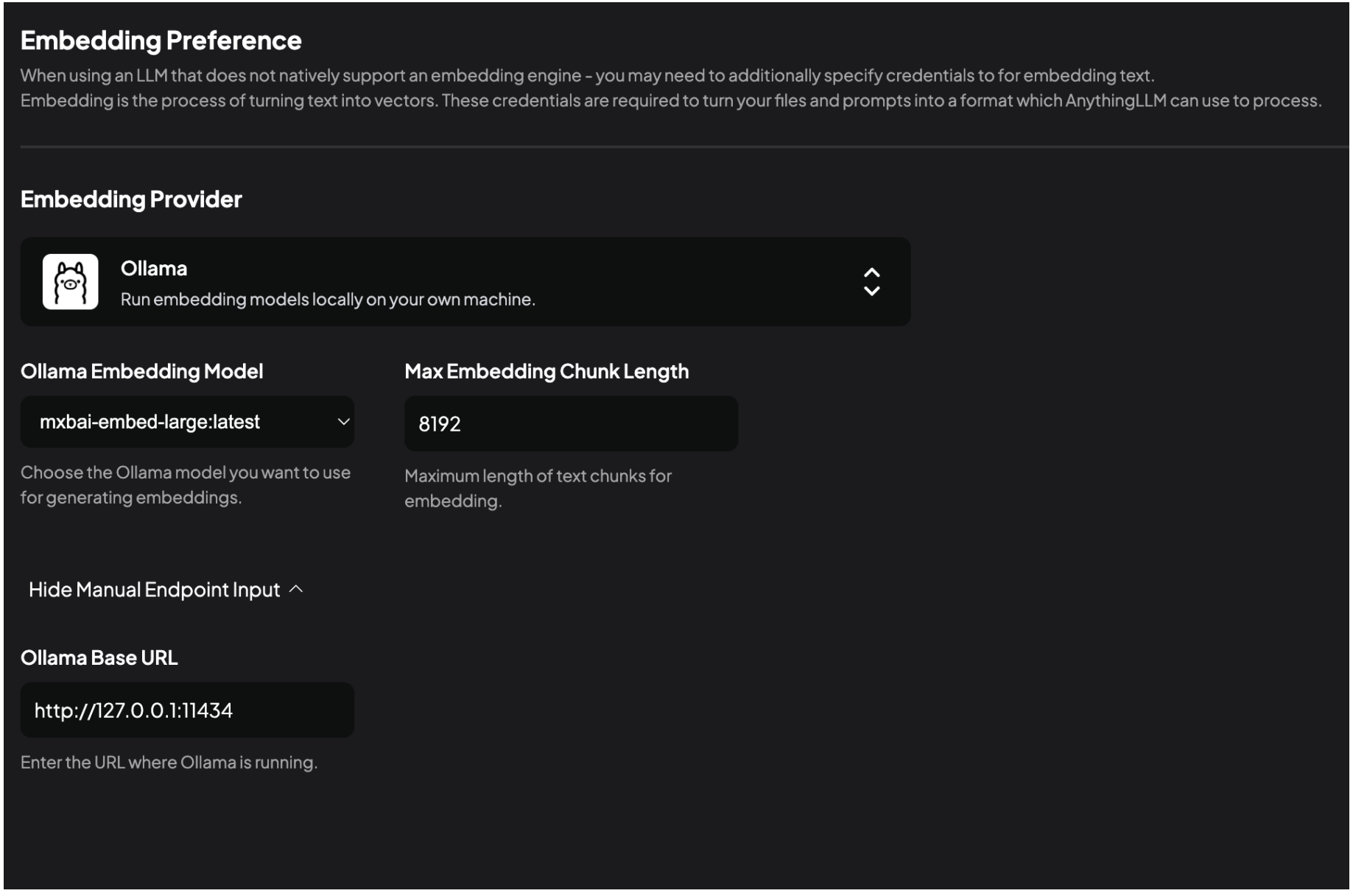


Typical guidance: **Short documents**: No need to chunk. One vector per doc, **Medium documents (1–2k tokens)** , operate with chunk size: 500–800 and Overlap: 100–200. For **Long documents (¿2k tokens)**, use sliding window chunking with overlap and aggregate embeddings (mean pooling or attention re-weighted).

Settings for chunking & embedding in AnythingLLM



Settings for chunking & embedding in AnythingLLM



Chunking and Embedding using OpenAI API

```
def chunk_text(text, size=512, overlap=100):
    chunks = []
    for i in range(0, len(text), size - overlap):
        chunk = text[i:i+size]
        chunks.append(chunk)
    return chunks

embeddings = [
  openai.Embedding.create(input=chunk, model="text-
     embedding-3-small")
  for chunk in chunk_text(document)
]
```

Plan

Plan

How humans process information

Language Modeling Basics

Embeddings as workhorse for large language models

LLM Building Blocks

Logistic Regression as a Neural Network

Developing scalable classifiers

Working with LLMs

Recap Logistic Regression: A Generative Likelihood Model

We model the conditional probability of binary labels:

$$P(y_i = 1 \mid \mathbf{x}_i) = \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})$$

Assuming a Bernoulli likelihood:

$$\mathcal{L}(\boldsymbol{\beta}) = \prod_{i=1}^{n} \left[\sigma(\mathbf{x}_i^\top \boldsymbol{\beta})\right]^{y_i} \left[1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})\right]^{1-y_i}$$

We maximize the log-likelihood:

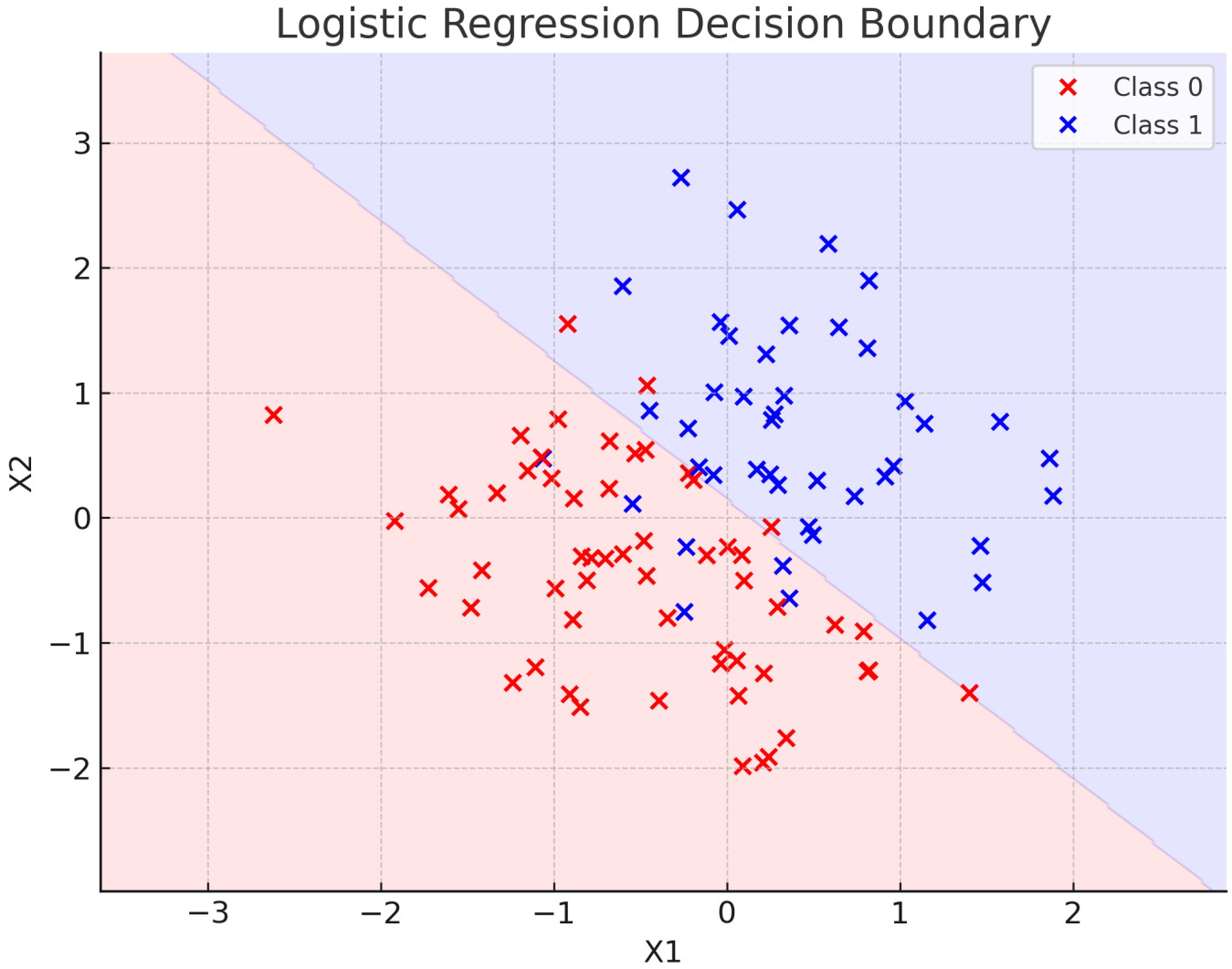
$$\log \mathcal{L}(\boldsymbol{\beta}) = \sum_{i=1}^{n} y_i \log \sigma(\mathbf{x}_i^\top \boldsymbol{\beta}) + (1 - y_i) \log(1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\beta}))$$

This is equivalent to minimizing the **binary cross-entropy loss**:

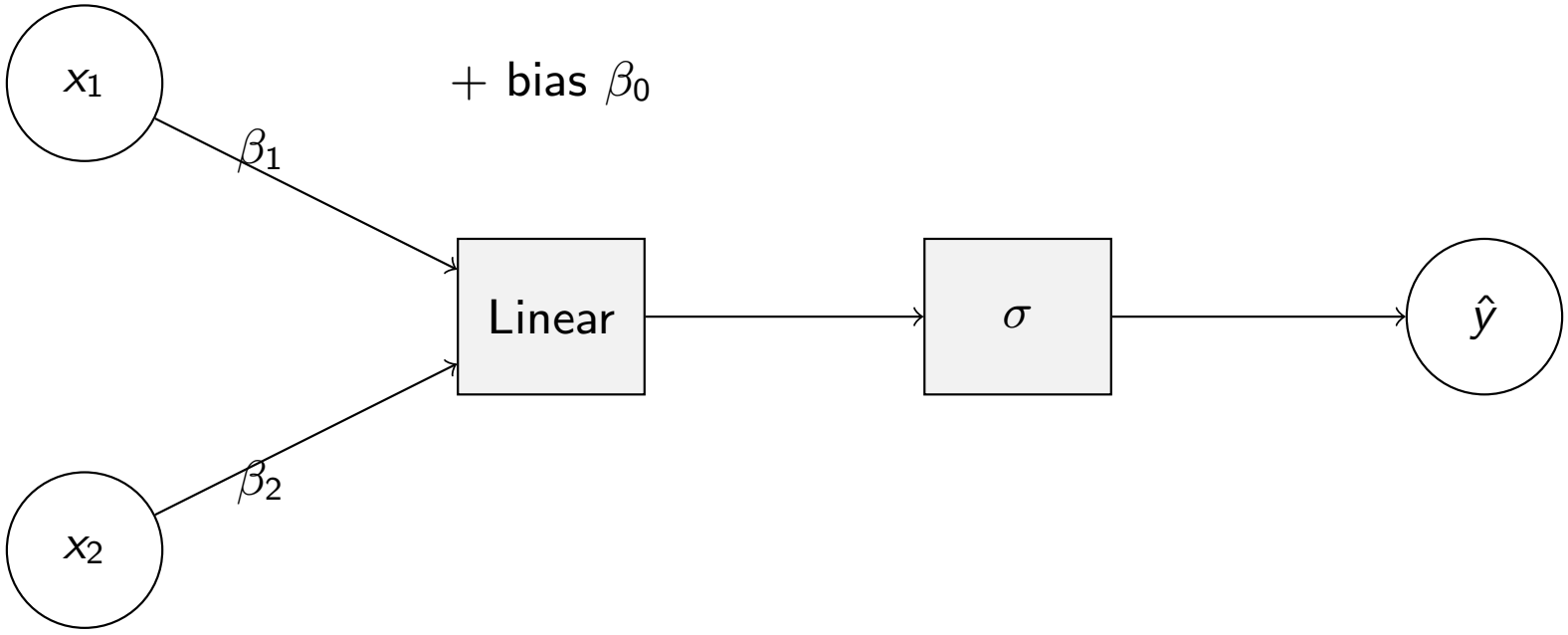
$$\mathcal{L}_{\text{CE}} = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log \sigma(\mathbf{x}_i^\top \boldsymbol{\beta}) + (1 - y_i) \log \left(1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})\right)\right]$$

Logistic Regression Fitting

A typical logistic regression decision boundary that is learned e.g. via MLE



Visualizing Logistic Regression as a Neural Network



- General form:

$$\hat{y} = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2)$$

- Example (fitted):

$$\hat{y} = \sigma(-0.82 + 1.10 x_1 + 0.85 x_2)$$

- Logistic regression is equivalent to a single-layer neural network.

What about the σ function?

A linear probability model has:

$$\sigma = \mathbf{x}_i^\top \boldsymbol{\beta}$$

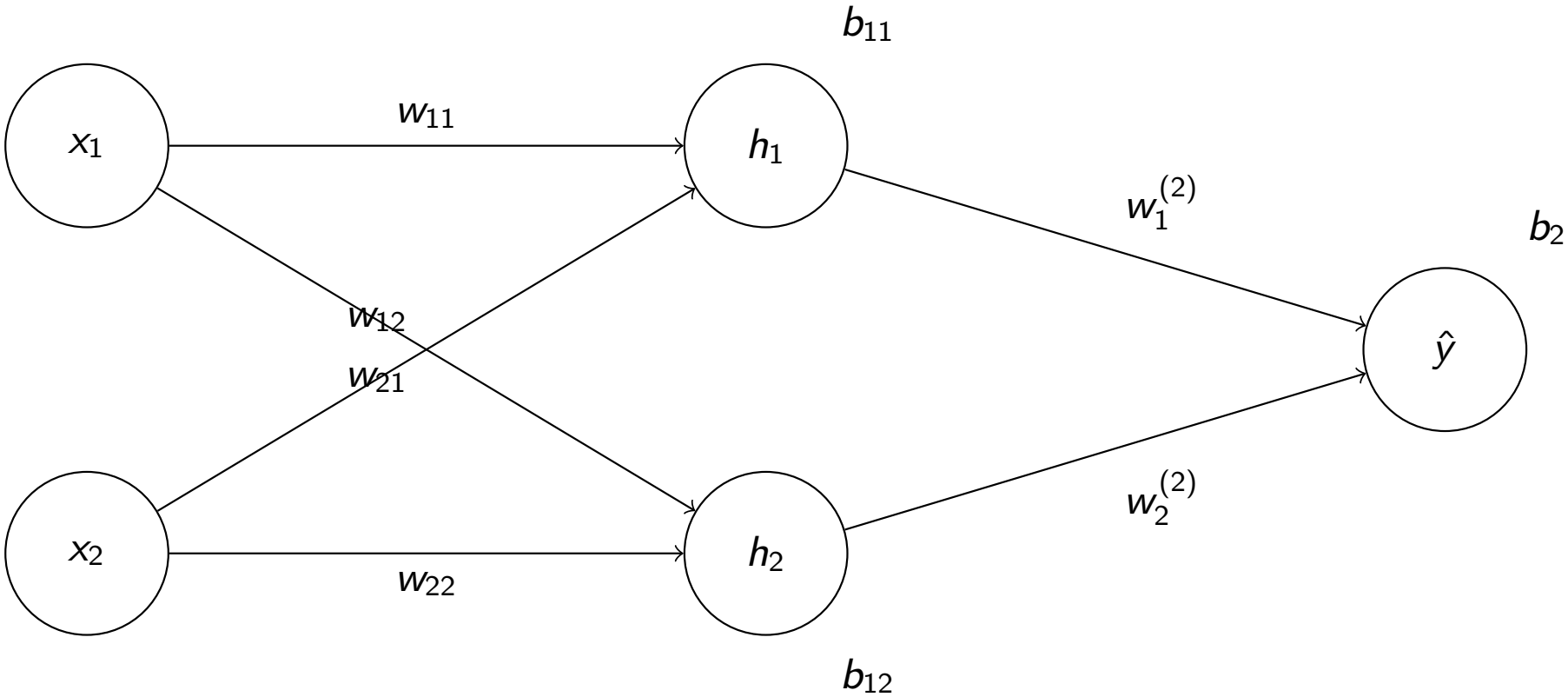
But this can produce fitted probabilities $P(y_i = 1 \mid \mathbf{x}_i)$ that are negative or above 1.
A common replacement is a sigmoid or *softmax* representation. The sigmoid function maps real numbers into the interval $(0, 1)$:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Applied to our model:

$$P(y_i = 1 \mid \mathbf{x}_i) = \frac{1}{1 + \exp(-\mathbf{x}_i^\top \boldsymbol{\beta})}$$

Example Neural Network with One Hidden Layer With Two Neurons



Each edge is labeled with its corresponding weight symbol. Biases are annotated next to each layer.

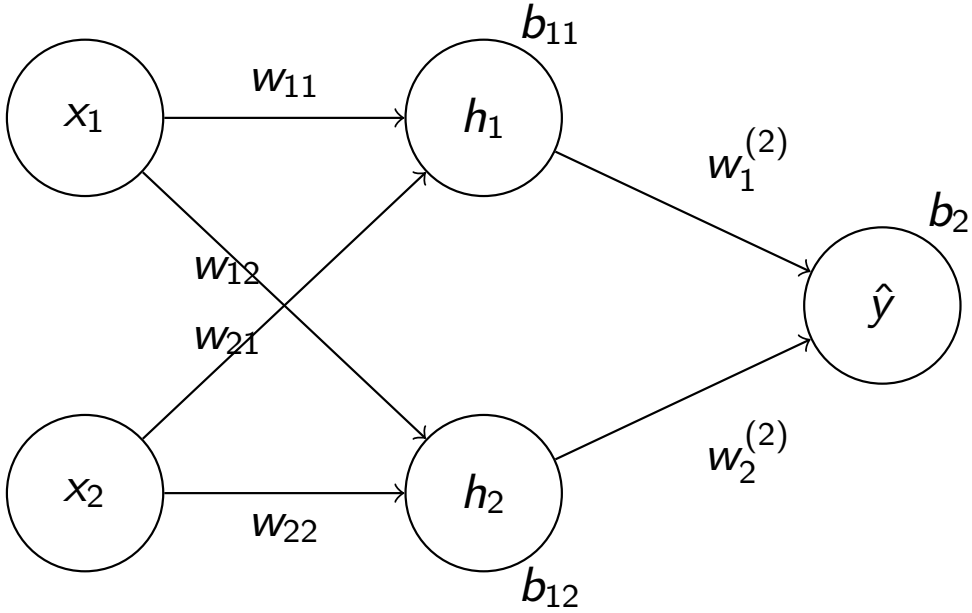
An Example Neural Network

Model Definition:

$$\hat{y} = \sigma\left(\mathbf{W}_2^\top \cdot \mathrm{f}(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + b_2\right)$$

Parameters:

- $\mathbf{W}_1 \in \mathbb{R}^{2\times 2}$ input hidden weights
- $\mathbf{b}_1 \in \mathbb{R}^2$: biases hidden neurons
- $\mathbf{W}_2 \in \mathbb{R}^{2\times 1}$ hidden output weights
- $b_2 \in \mathbb{R}$: output bias



Each edge is labeled with its corresponding parameter. Biases are shown next to the hidden/output layers.

Forward Pass:

$$\mathbf{z}_1 = \mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1$$
$$\mathbf{h}_1 = \mathrm{f}(\mathbf{z}_1)$$
$$z_2 = \mathbf{W}_2^\top \cdot \mathbf{h}_1 + b_2$$
$$\hat{y} = \sigma(z_2)$$

Forward Pass Equations Spelled Out

Hidden Layer (Pre-activations):

$$z_1 = w_{11}x_1 + w_{21}x_2 + b_{11}$$
$$z_2 = w_{12}x_1 + w_{22}x_2 + b_{12}$$

Hidden Layer (Activations):

$$h_1 = f(z_1)$$
$$h_2 = f(z_2)$$

Output Layer (Pre-activation):

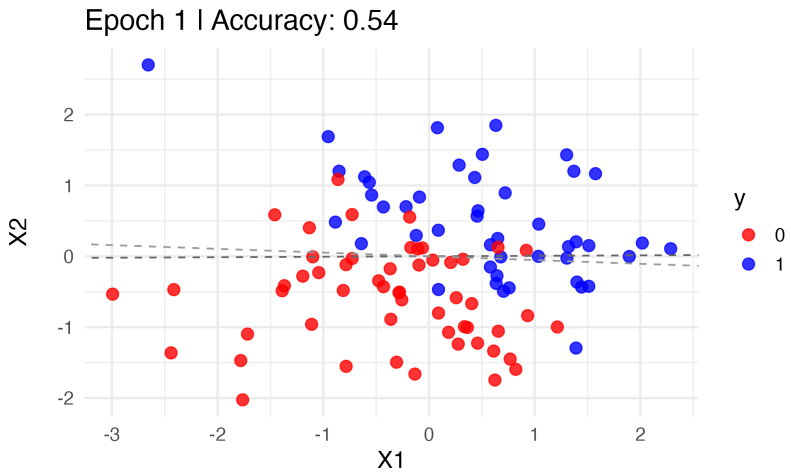
$$z_{\text{out}} = w_1^{(2)} h_1 + w_2^{(2)} h_2 + b_2$$

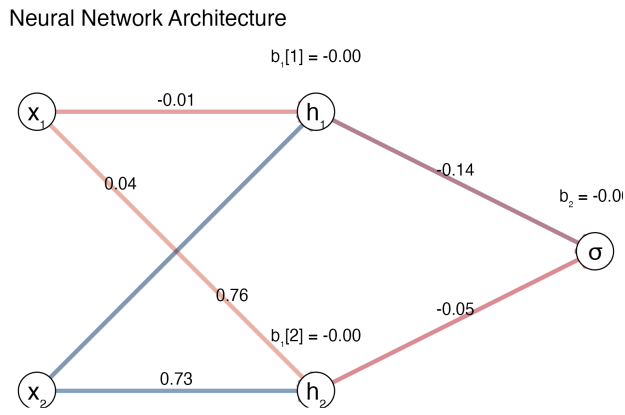
Predicted Output:

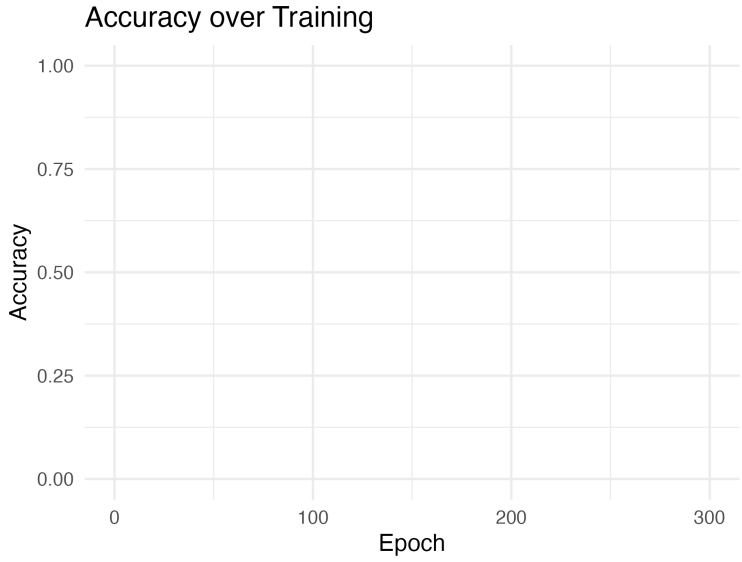
$$\hat{y} = \psi(z_{\text{out}})$$

Where $f(\cdot)$ is the activation function (e.g. ReLU or sigmoid), and $\psi(\cdot)$ is the output activation (e.g. identity or sigmoid).

Neural Network Training: Gradient Descent in Action







$$y = \sigma(W_2 \cdot \mathrm{ReLU}(W_1 \cdot X + b_1) + b_2)$$

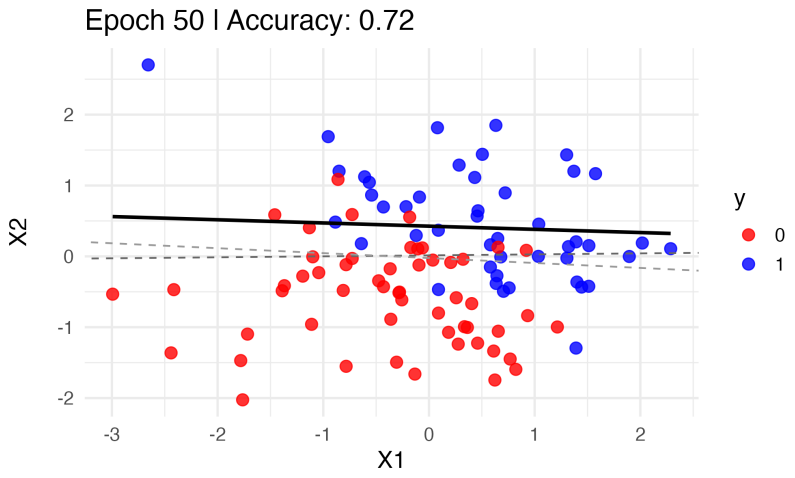
$$W_1 = [[-0.01, 0.04], [0.76, 0.73]]$$

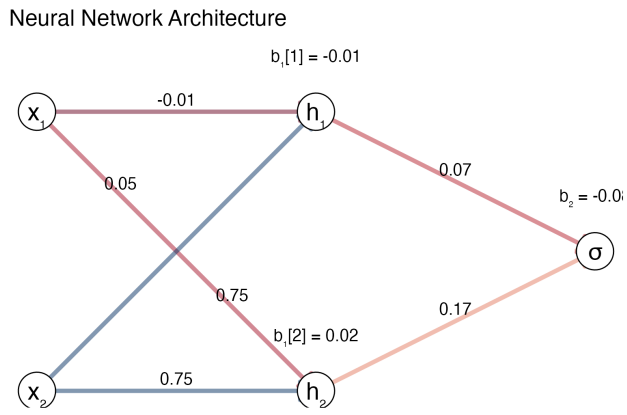
$$b_1 = [-0.00, -0.00]$$

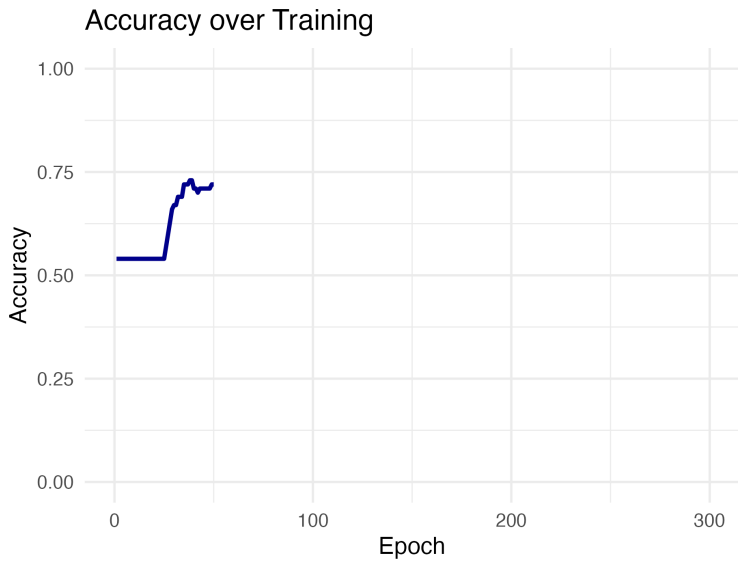
$$W_2 = [-0.14, -0.05]$$

$$b_2 = -0.00$$

Neural Network Training: Gradient Descent in Action







$$y = \sigma(W_2 \cdot \mathrm{ReLU}(W_1 \cdot X + b_1) + b_2)$$

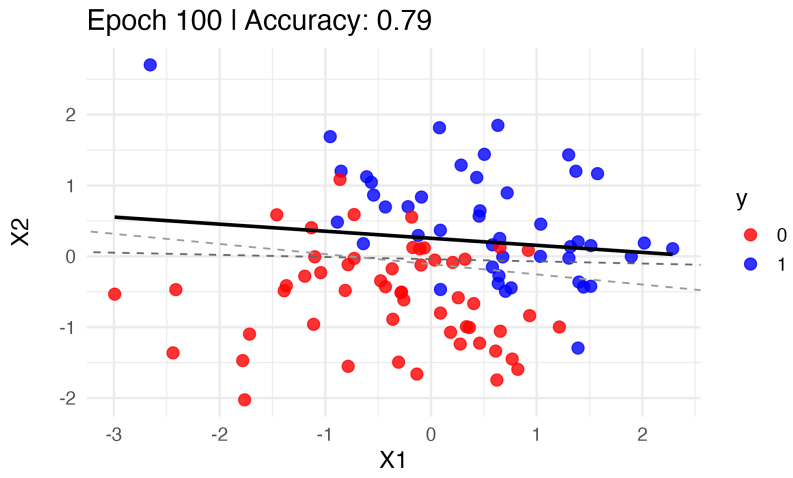
$$W_1 = [[-0.01, 0.05], [0.75, 0.75]]$$

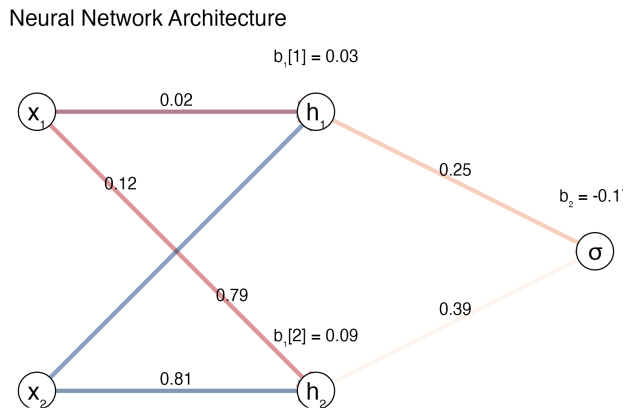
$$b_1 = [-0.01, 0.02]$$

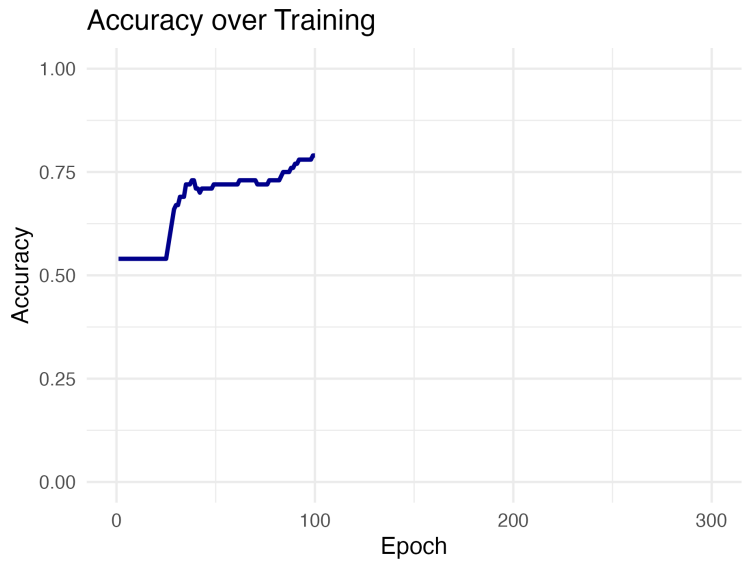
$$W_2 = [0.07, 0.17]$$

$$b_2 = -0.08$$

Neural Network Training: Gradient Descent in Action







$$y = \sigma(W_2 \cdot \mathrm{ReLU}(W_1 \cdot X + b_1) + b_2)$$

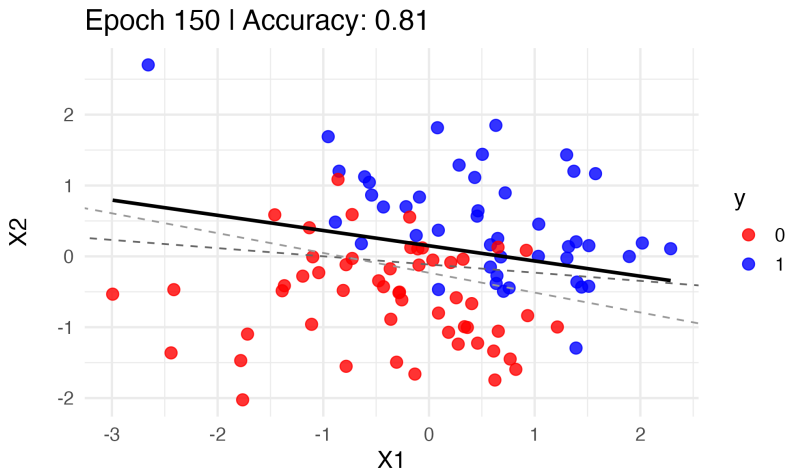
$$W_1 = [[0.02, 0.12], [0.79, 0.81]]$$

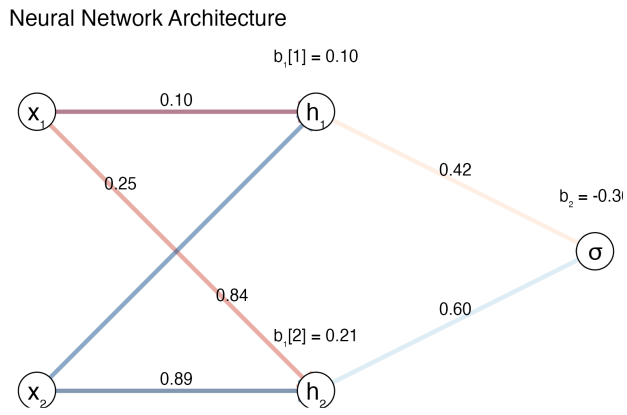
$$b_1 = [0.03, 0.09]$$

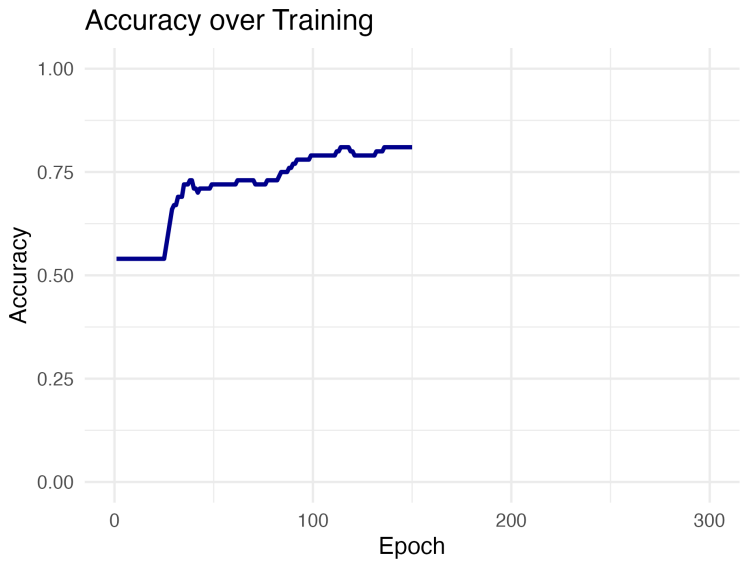
$$W_2 = [0.25, 0.39]$$

$$b_2 = -0.17$$

Neural Network Training: Gradient Descent in Action







$$y = \sigma(W_2 \cdot \mathrm{ReLU}(W_1 \cdot X + b_1) + b_2)$$

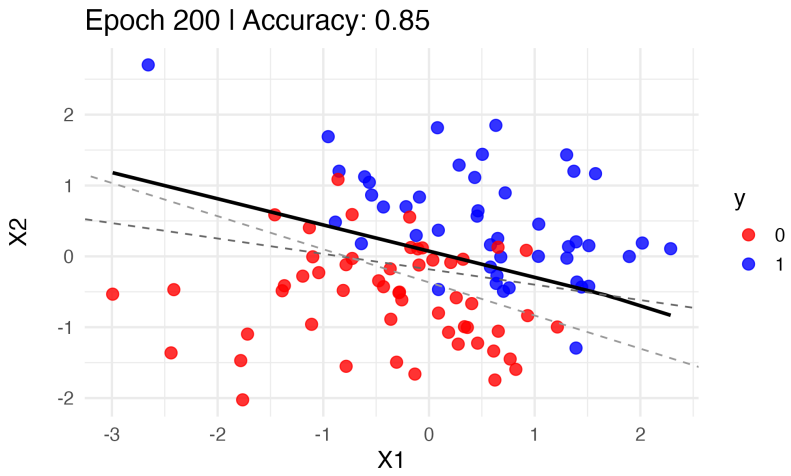
$$W_1 = [[0.10, 0.25], [0.84, 0.89]]$$

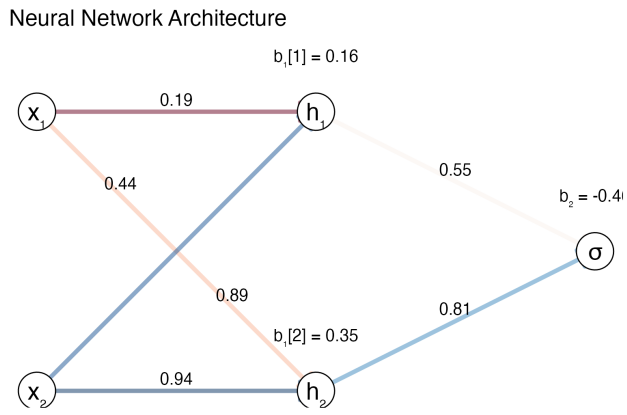
$$b_1 = [0.10, 0.21]$$

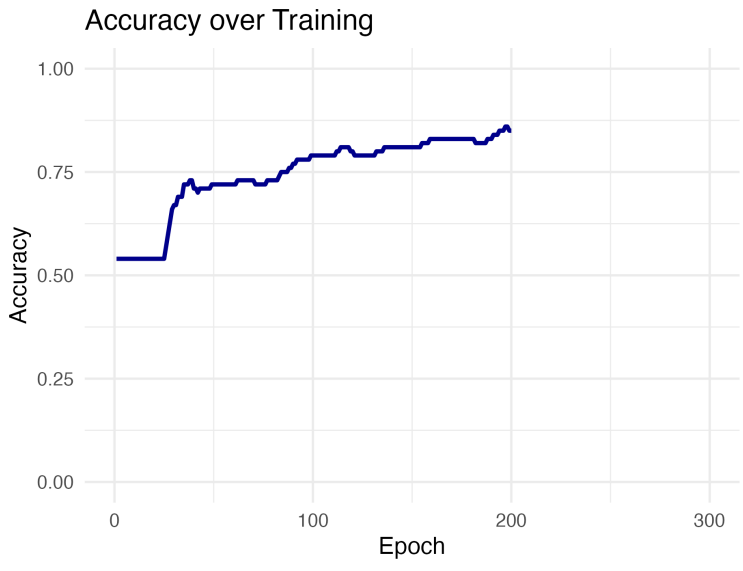
$$W_2 = [0.42, 0.60]$$

$$b_2 = -0.30$$

Neural Network Training: Gradient Descent in Action







$$y = \sigma(W_2 \cdot \mathrm{ReLU}(W_1 \cdot X + b_1) + b_2)$$

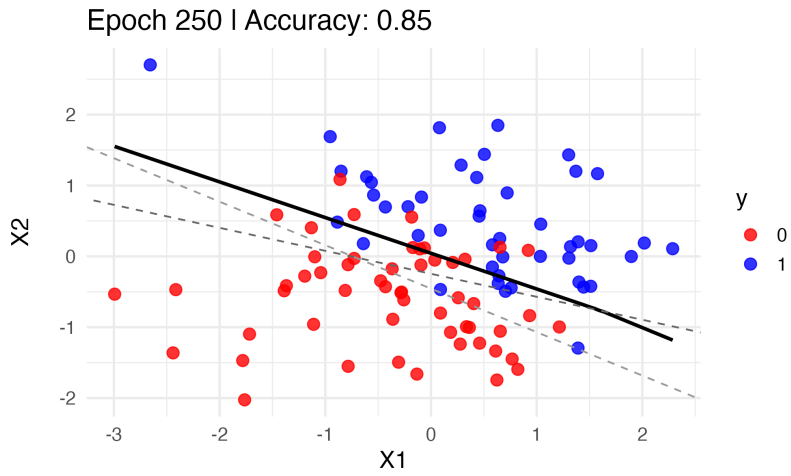
$$W_1 = [[0.19, 0.44], [0.89, 0.94]]$$

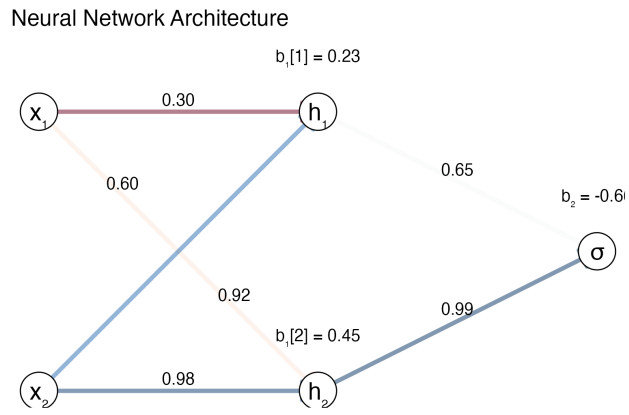
$$b_1 = [0.16, 0.35]$$

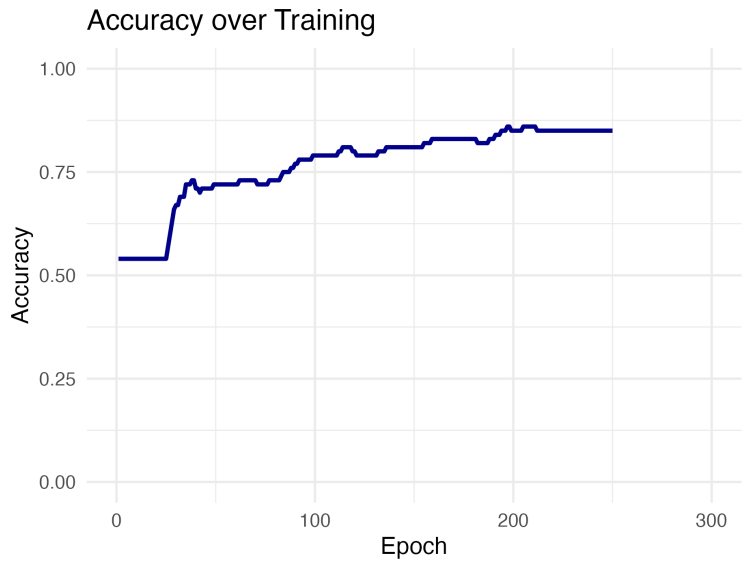
$$W_2 = [0.55, 0.81]$$

$$b_2 = -0.46$$

Neural Network Training: Gradient Descent in Action







$$y = \sigma(W_2 \cdot \mathrm{ReLU}(W_1 \cdot X + b_1) + b_2)$$

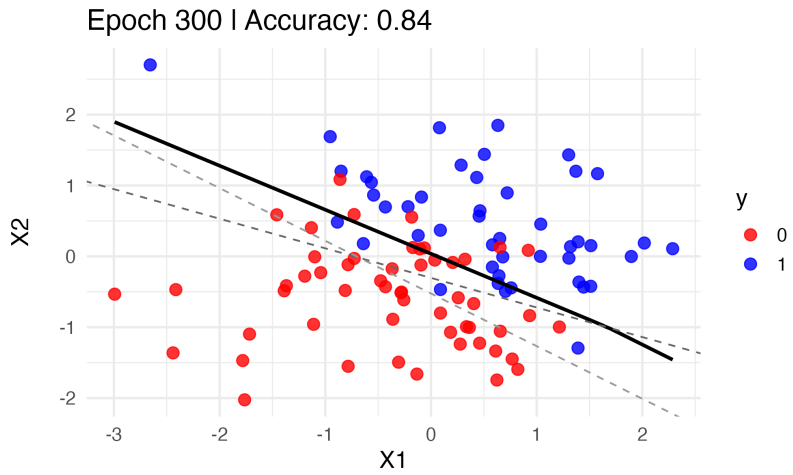
$$W_1 = [[0.30, 0.60], [0.92, 0.98]]$$

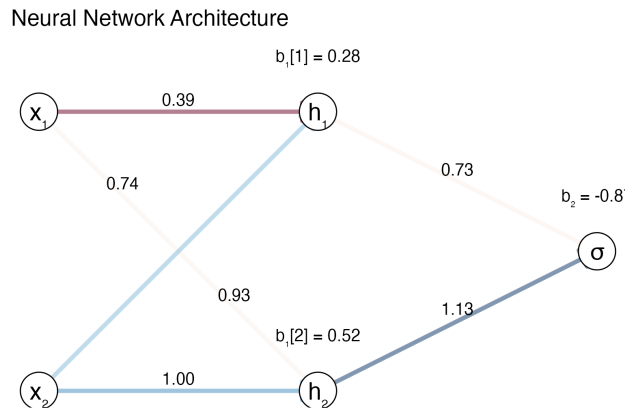
$$b_1 = [0.23, 0.45]$$

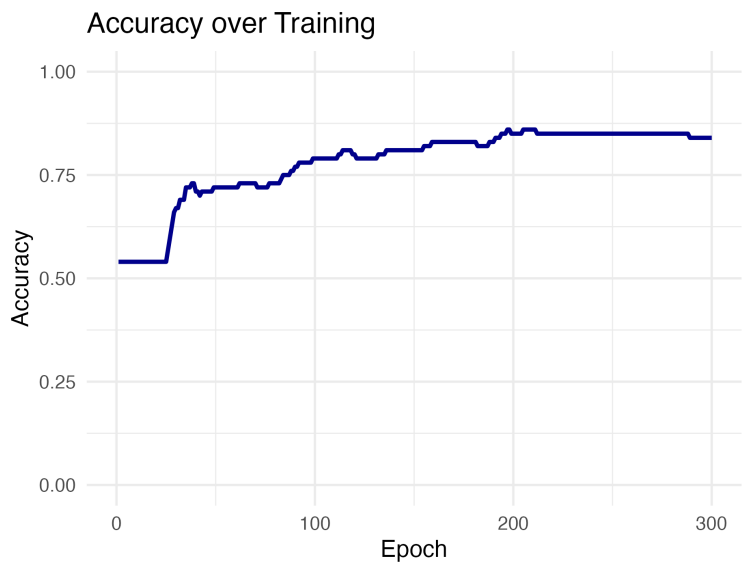
$$W_2 = [0.65, 0.99]$$

$$b_2 = -0.66$$

Neural Network Training: Gradient Descent in Action







$$y = \sigma(W_2 \cdot \mathrm{ReLU}(W_1 \cdot X + b_1) + b_2)$$

$$W_1 = [[0.39, 0.74], [0.93, 1.00]]$$

$$b_1 = [0.28, 0.52]$$

$$W_2 = [0.73, 1.13]$$

$$b_2 = -0.87$$

Forward Pass - Prediction Step

In each epoch, the forward pass takes the current weights and biases and produces predictions for all input samples.

For each input example $\mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$, the neural network computes:

$$\mathbf{z}_1^{(i)} = \mathbf{W}_1 \cdot \mathbf{x}^{(i)} + \mathbf{b}_1 \qquad \text{(shape: } 2 \times 1\text{) - linear combination}$$

$$\mathbf{h}_1^{(i)} = \mathrm{f}(\mathbf{z}_1^{(i)}) \qquad \text{(hidden activation)}$$

$$z_2^{(i)} = \mathbf{W}_2^{\top} \cdot \mathbf{h}_1^{(i)} + b_2 \qquad \text{(shape: scalar)}$$

$$\hat{y}^{(i)} = \sigma(z_2^{(i)}) \qquad \text{(predicted probability)}$$

Where $\mathbf{x}^{(i)} \in \mathbb{R}^{2\times 1}$, $\mathbf{W}_1 \in \mathbb{R}^{2\times 2}$, $\mathbf{b}_1 \in \mathbb{R}^{2\times 1}$, $\mathbf{W}_2 \in \mathbb{R}^{2\times 1}$, $b_2 \in \mathbb{R}$

Intuition: the forward pass is applying a recipe: the ingredients (inputs and parameters) are used to produce a prediction (the output) for each input. This prediction will later be compared to the true label during the backward pass.

Backward Pass - How the Network Learns

The backward pass computes how each weight and bias contributed to the prediction error, and how to adjust them to reduce future errors.

1. **Loss Function (per observation):**

$$\mathcal{L}^{(i)} = -\left[y^{(i)} \log(\sigma(z_2^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma(z_2^{(i)}))\right]$$

2. **Total Loss (Objective Function):**

$$\mathcal{L}_{\text{total}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}^{(i)}$$

Loss is computed for each observation, but the gradient is taken with respect to the mean (or sum) across all data.

3. **Core Idea:** Use **chain rule** to compute gradients of the loss vis-a-vis all parameters: weights and biases in both hidden and output layers.

Backpropagation: This is the systematic application of the chain rule, layer by layer in reverse, to compute gradients efficiently.

Gradient Computation via Chain Rule for one $(\mathbf{x}^{(i)}, y^{(i)})$

Output layer:

$$\frac{\partial \mathcal{L}^{(i)}}{\partial z_2^{(i)}} = \hat{y}^{(i)} - y^{(i)} \quad \text{(simplified from sigmoid + cross-entropy)}$$

$$\frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{W}_2} = (\hat{y}^{(i)} - y^{(i)}) \cdot \mathbf{h}_1^{(i)} \qquad \frac{\partial \mathcal{L}^{(i)}}{\partial b_2} = \hat{y}^{(i)} - y^{(i)}$$

Backpropagation into hidden layer:

$$\frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{h}_1^{(i)}} = (\hat{y}^{(i)} - y^{(i)}) \cdot \mathbf{W}_2$$

$$\frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{z}_1^{(i)}} = \frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{h}_1^{(i)}} \circ \mathrm{f}'(\mathbf{z}_1^{(i)})$$

Weight Update Rule (Gradient Descent)

At each step:

$$\mathbf{W}_1 \leftarrow \mathbf{W}_1 - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}$$

$$\mathbf{b}_1 \leftarrow \mathbf{b}_1 - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1}$$

$$\mathbf{W}_2 \leftarrow \mathbf{W}_2 - \eta \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}_2}$$

$$b_2 \leftarrow b_2 - \eta \cdot \frac{\partial \mathcal{L}}{\partial b_2}$$

$\rightarrow$ here is where the learning rate η comes in (check out the R code)

Our Simple Single Layer Neural Network with 2-Input, 2-Hidden Neurons Network

Parameter	Label	Gradient Expression	Plain English Explanation
Output pre-activation	z_{out}	$\frac{\partial \mathcal{L}}{\partial z_{\text{out}}} = \hat{y} - y$	Error term from output (prediction minus target)
Output weights	$w_1^{(2)}, w_2^{(2)}$	$\frac{\partial \mathcal{L}}{\partial w_j^{(2)}} = (\hat{y} - y) \cdot h_j$	Gradient w.r.t. weight from hidden unit h_j to output
Output bias	b_2	$\frac{\partial \mathcal{L}}{\partial b_2} = \hat{y} - y$	Same as the output error term
Hidden pre-activations	z_1, z_2	$\frac{\partial \mathcal{L}}{\partial z_j} = (\hat{y} - y) \cdot w_j^{(2)} \cdot f'(z_j)$	Backpropagate output error to each hidden neuron via chain rule
Hidden weights	w_{ij}	$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial z_j} \cdot x_i$	Gradient of loss w.r.t. weights from input x_i to hidden unit j
Hidden biases	b_{11}, b_{12}	$\frac{\partial \mathcal{L}}{\partial b_1[j]} = \frac{\partial \mathcal{L}}{\partial z_j}$	Same as hidden neuron error term

Notes:

- $\hat{y} = \sigma(z_{\text{out}})$ is the predicted probability.
- $f'(z_j)$ is the derivative of the hidden layer activation.
- $w_j^{(2)}$ is the weight from hidden neuron j to output.
- Chain rule is applied to backpropagate error from output to inputs.

What is f and Why Do We Need It?

In neural networks, **nonlinear activation functions** are essential for modeling complex patterns. One of the most widely used is the Rectified

Linear Unit activation:

$$\mathrm{f}(z) = \max(0, z)$$

That is:

- If $z > 0$, output is z
- If $z \leq 0$, output is 0

This is explicitly non-linear function.

Why Do We Need Activation Functions f?

Without a non-linear functions like f, a neural network with multiple layers reduces to a **linear transformation**:

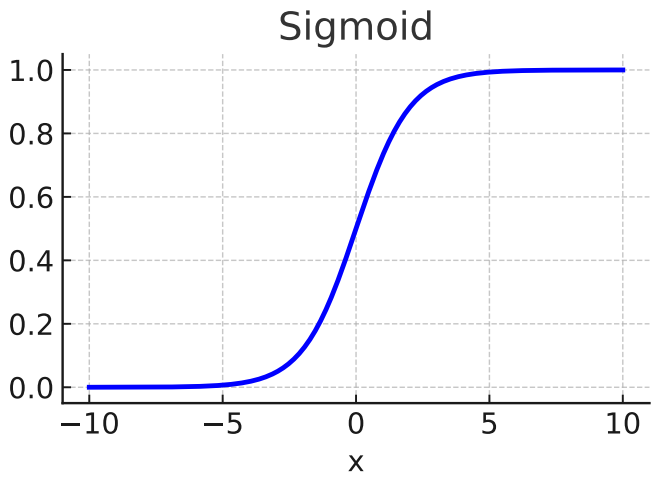
$$\sigma(\mathbf{W}_2 \cdot (\mathbf{W}_1 \cdot \mathbf{x})) = \sigma(\mathbf{W}_{\text{combined}} \cdot \mathbf{x})$$

Since this is just a matrix multiplication of 2 x 2 matrix W_1 with a 2 x 1 matrix W_2 yielding a 2 x 1 vector.

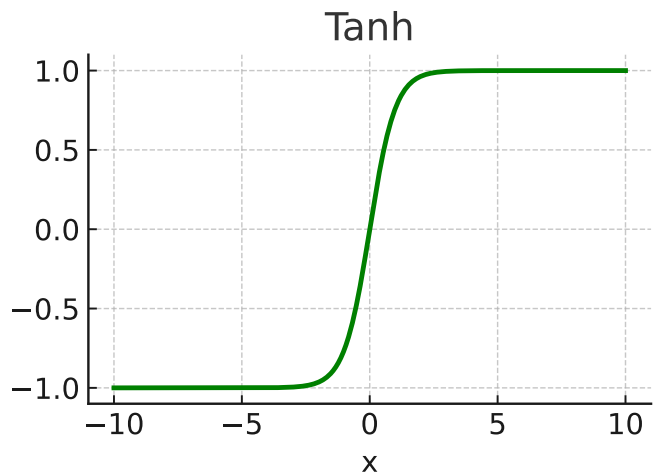
This is functionally the same as logistic regression!

$\rightarrow$ f introduces **non-linearity**, enabling the network to **approximate complex decision boundaries**.

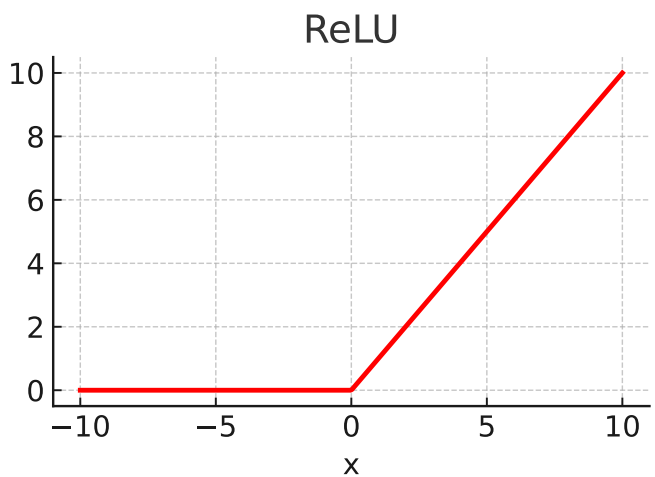
Alternative Activation Functions



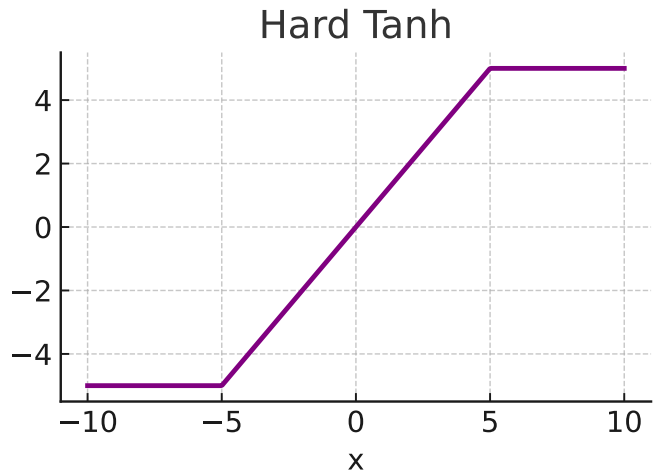
Sigmoid
Smooth S-curve; saturates at 0 and 1.



Tanh
Zero-centered; maps to $(-1, 1)$.



ReLU
Zero for $x < 0$; linear for $x > 0$.



Hard Tanh
Linearly bounded between -5 and $+5$.

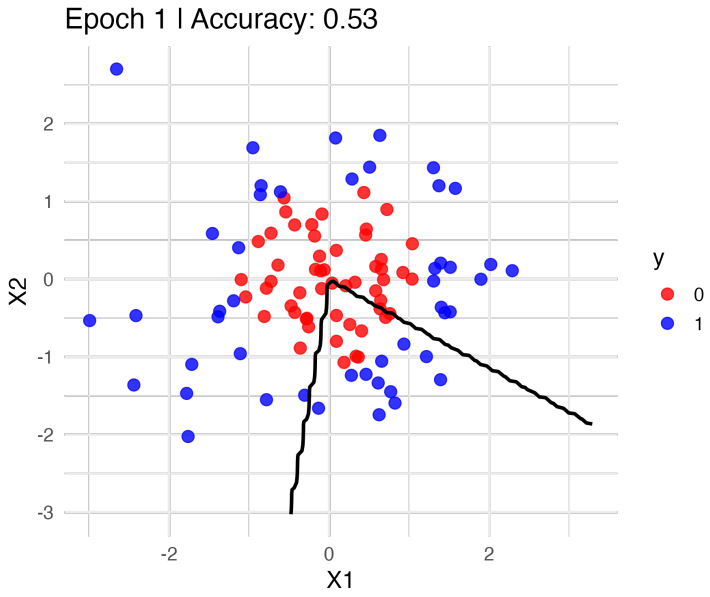
Why do we like RELU or hard tanh?

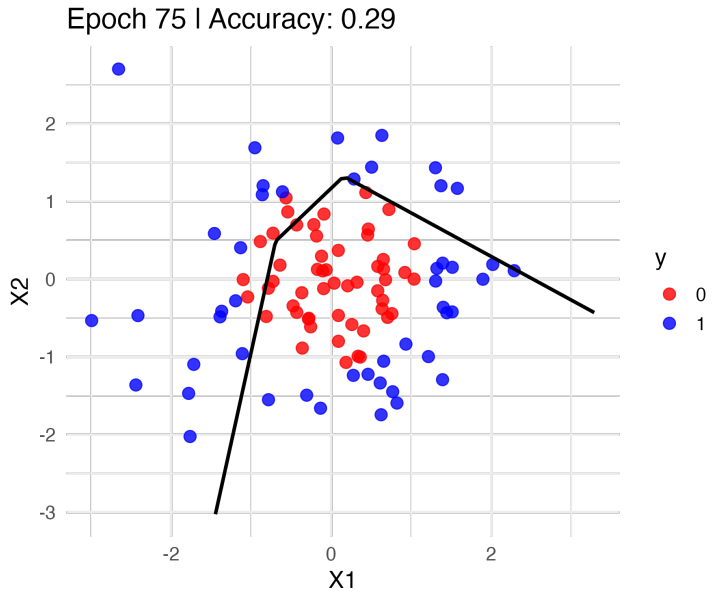
$\rightarrow$ strong preference for RELU and Hard TanH, why?

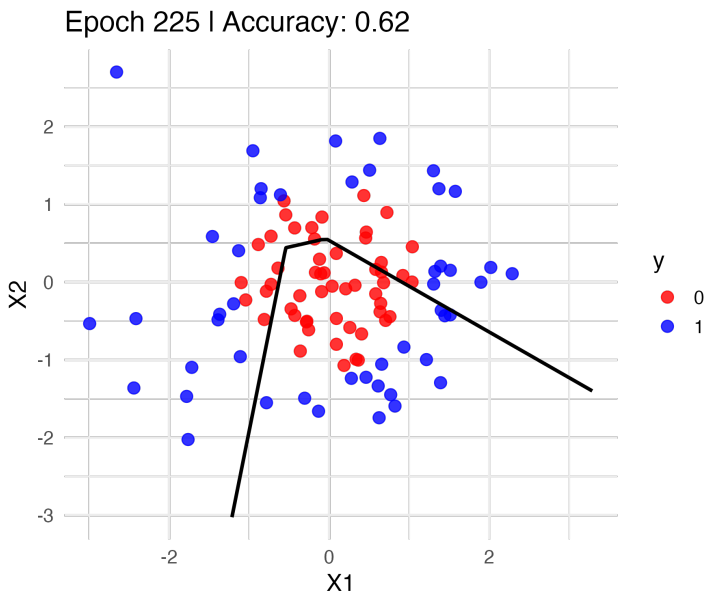
- **Simple**: Fast to compute, easy to differentiate
- **Sparse activation**: Most neurons output zero - makes networks efficient
- **Avoids vanishing gradients** (compared to sigmoid or tanh)

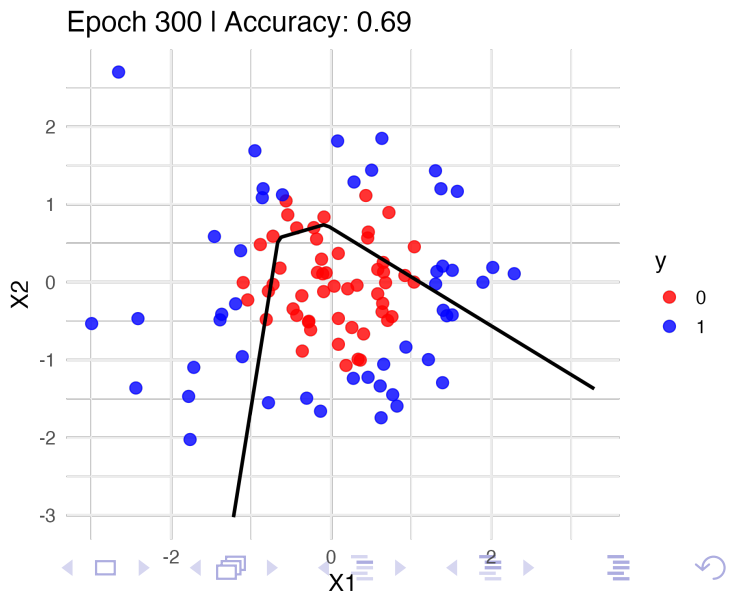
Each f "unit" acts like a **hinge** or **kink** that bends the decision surface. That is why RELU is often also referred to as "hinge Loss".
Using multiple f neurons in a layer allows a network to **combine piecewise linear functions** - forming nonlinear, expressive mappings from inputs to outputs.

Example with non-linearly separable data









Plan

How humans process information

Language Modeling Basics

Embeddings as workhorse for large language models

LLM Building Blocks

Logistic Regression as a Neural Network

Developing scalable classifiers

Working with LLMs

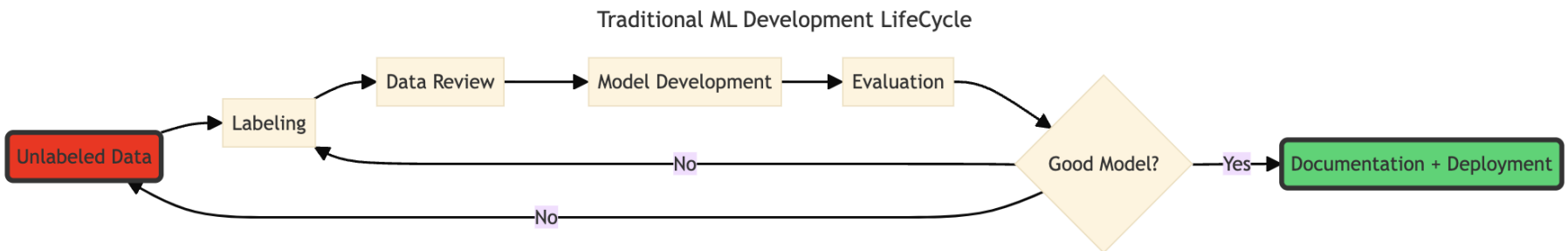
Classification is a most common application

- Working with unstructured data or even just for data filtering may require **deployment of classifiers**
- Human brain is constantly in process of filtering information or `classifying` – small kids, object sorting etc
- Building classification systems and nomenclatures enables functional diplomacy
- Traditionally classification was done with a suite of techniques typically distinguishing **generative** versus **discriminative** classifiers
  - Naive Bayes classification
  - Logistic regression or multinomial logistic regression
  - Random forests, trees etc
  - Support Vector machines

$\rightarrow$ LLMs can be used to replace traditional machine-learning approaches or can be used to supercharge

Traditional machine learning cycle

Traditional Machine Learning cycle



with many iterative steps going from model learning to evaluation in a circular fashion is overhead heavy and slow.

Image from `https://github.com/kenhktsui/anyclassifier`

With LLMs we can jump this process

- Use well curated **prompt** you can instruct an LLM to perform a classification task
- Prompt may include **examples** in the context window
- Typically main trade-off is over what and how an LLM is being used

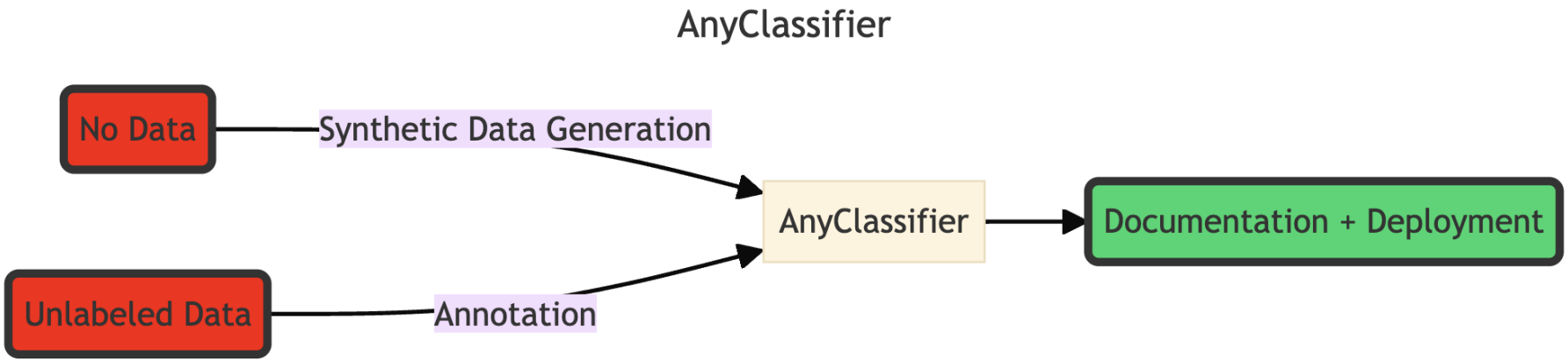
  Research use: often times limited in focus to inference

  Production use: focus on prediction and real time performance

$\rightarrow$ lets look at some examples

Anything LLM classifier

Approach involves creation of synthetic data via LLM and subsequent scaling



of synthetic data on to the population of unlabelled data

Image from `https://github.com/kenhktsui/anyclassifier`

Model Options in `anyclassifier`

SetFit (default):

- Uses Sentence Transformers to embed text
- Trains a logistic regression or linear classifier on top
- Fast, efficient, scalable for large datasets
- Avoids full fine-tuning $\Rightarrow$ much faster than traditional transformers

FastText:

- Shallow classifier using n-gram features and embeddings
- Developed by Facebook; scales to millions of examples
- Closest to classical ML (e.g., SVM) in spirit and performance

Transformers (optional):

- Fully fine-tuned models (e.g., BERT)
- More accurate, but slower and computationally heavy

What's Automated?

- **LLM-driven labeling:** High-quality pseudo-labels from LLMs
- **Few-shot learning:** Improve label quality with example inputs
- **Synthetic data generation:** LLMs can generate labeled training samples

Plan

How humans process information

Language Modeling Basics

Embeddings as workhorse for large language models

LLM Building Blocks

Logistic Regression as a Neural Network

Developing scalable classifiers

Working with LLMs

Model Parameters Parameters and Model Behavior

The following parameters shape how the transformer decodes output during inference:

- **Response format**: If specified (e.g., JSON schema), the model conditions output structure – affects token probability masking.
- **Temperature**: Controls softmax sharpness
- **Top P** : Nucleus sampling truncates token distribution to smallest set whose cumulative probability $\geq p$. Helps limit long tail of unlikely completions.
- **Frequency penalty** : Penalizes tokens already generated — implemented via logit bias before softmax.
- **Presence penalty** : Penalizes any prior appearance of a token — encourages introducing new topics.

Where Temperature and Sampling Fit in Generation

Inference Pipeline Steps:

1. Transformer outputs logits $z_t(w)$ for all tokens.
2. Logits are adjusted:
   - Apply temperature: $z_t(w)/T$
   - Apply presence/frequency penalties: specific $z_t(w)$ adjusted
3. Apply softmax:

$$P(w_t \mid w_{<t}) = \frac{\exp(z_t(w)/T)}{\sum_{w'} \exp(z_t(w')/T)}$$

4. Apply Top-P Sampling:
   - Sort tokens by $P(w)$
   - Select smallest set where cumulative $\sum P(w) \geq p$
   - Sample from this truncated set

Note: This sequence governs how language is sampled at each step of generation.

Temperature and the Softmax Distribution

In a text generation task, we model:

$$P(w_t \mid w_1, w_2, \ldots, w_{t-1}) = \frac{\exp\left(\frac{z_t(w_t)}{T}\right)}{\sum_{w'} \exp\left(\frac{z_t(w')}{T}\right)}$$

- w_t: next token to be predicted
- $z_t(w)$: unnormalized logit for token w at position t

Temperature inflates or deflates the transformer derived scores

- $T = 1$: standard softmax
- $T < 1$: sharper distribution - high-probability tokens become even more likely (low entropy)
- $T > 1$: flatter distribution - encourages exploration (higher entropy)

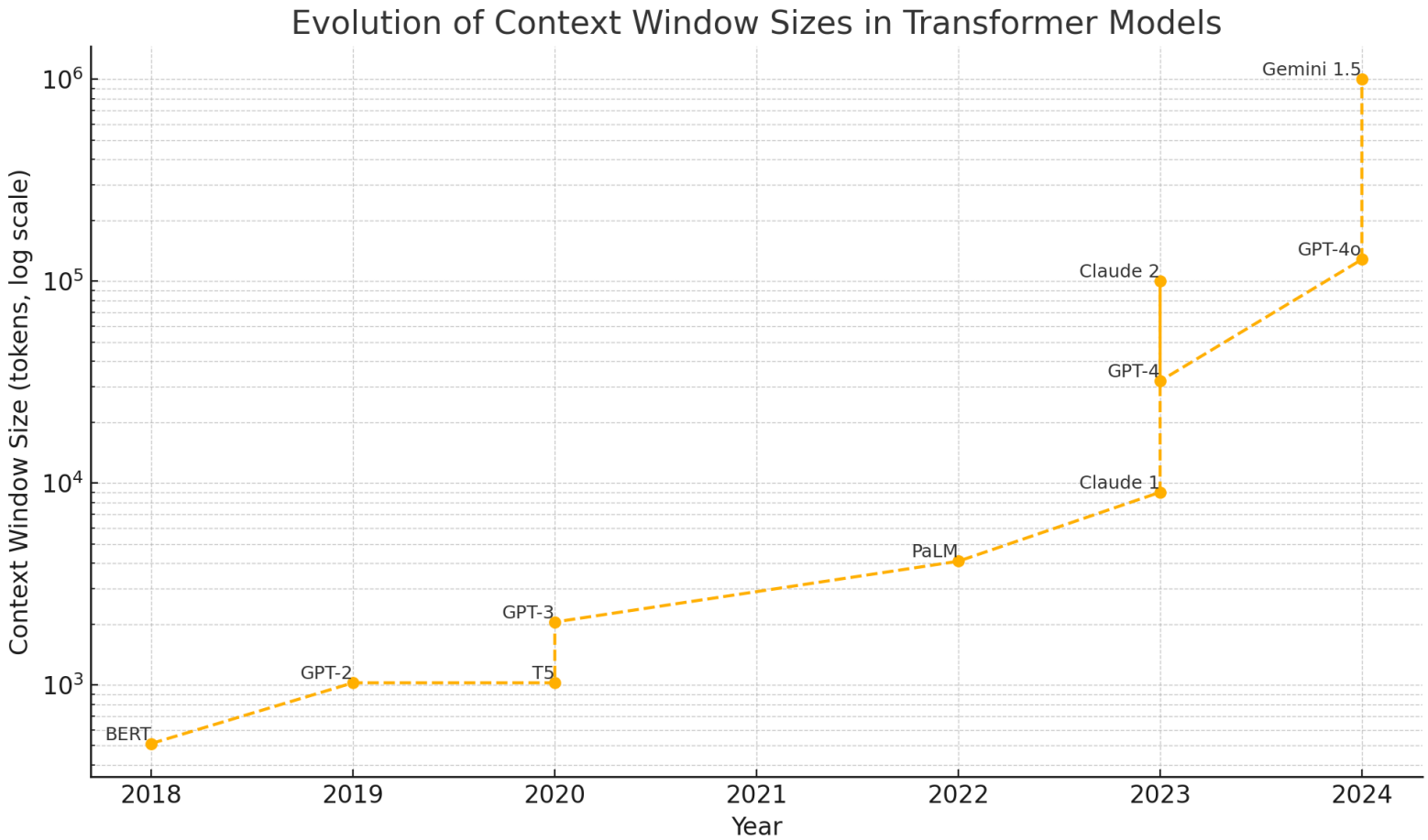
Temperature scaling is applied *after* logits are produced by the transformer but *before* sampling of words.

Where are we?

- We have learned the basic building blocs and training methods for large language models
- But how should we think of the interplay of the blocks in day to day use
- How do the tools work in what context and for what purpose?
- Cover today some common pipelines or architectures for LLM deployment

$\rightarrow$ evolution of LLMs has followed scaling patterns with ever expanding context windows, increase in number of parameters and development of reasoning capabilities

Context Window Evolution



$\rightarrow$ exponential evolution of context window with frontier models achieving up to 1 million tokens

A human life in context?

Let us go back to the amount of information that a human life generates or can transmit through speech

- We said its around `0.5 MB` per day
- Average human life has several vocabulary expansions, but assume we live 80 years in *speaking mode*

  `365 x 80 x 0.5 MB = 14.6 GB`

  In textual encoding approximately **3.76 billion** tokens are contained in 14 GB of plain text, assuming an average of 4 characters per token.
- Present LLMs are far from being able to replicate a full human as represented in textual output produced through context
- But we have seen massive and non-linear growth in capability

Problems with LLM approach

- Some data sources extremely high volume high frequency data

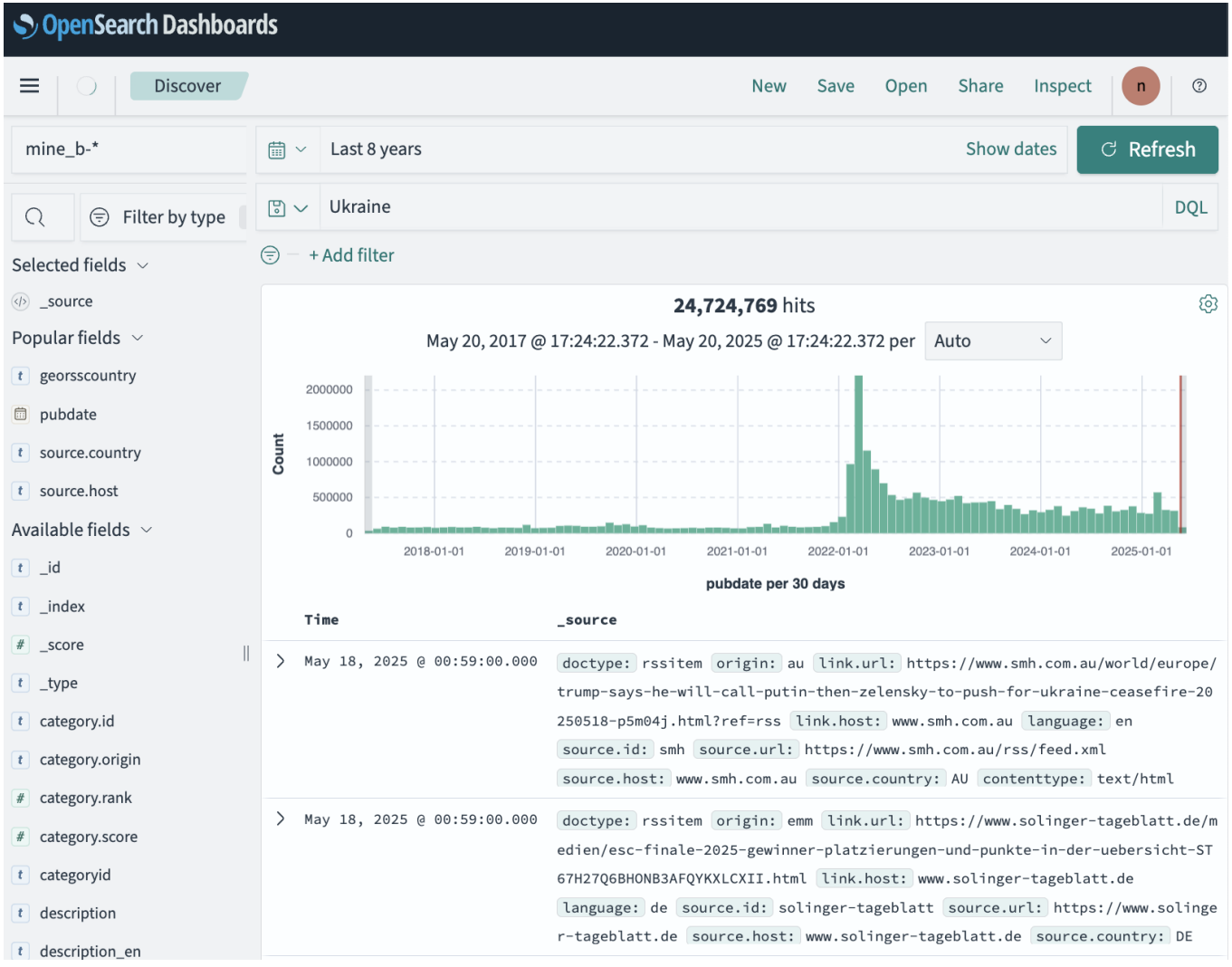
  Think: social media, online streaming platforms etc. or even some text
- If every human were to produce `0.5 MB` of spoken text a day this would still be

  8 billion humans producing text all day amounts to `8 billion x 0.5 MB = 3,204,346 GB`
- Most (signal) intelligence agencies engage in *social listening*

  Need MUCH faster processing of information and quick setup of social listening pipelines
- This is why there is a battle over control of two dimensions

  `digital identity` + `digital infrastructure`

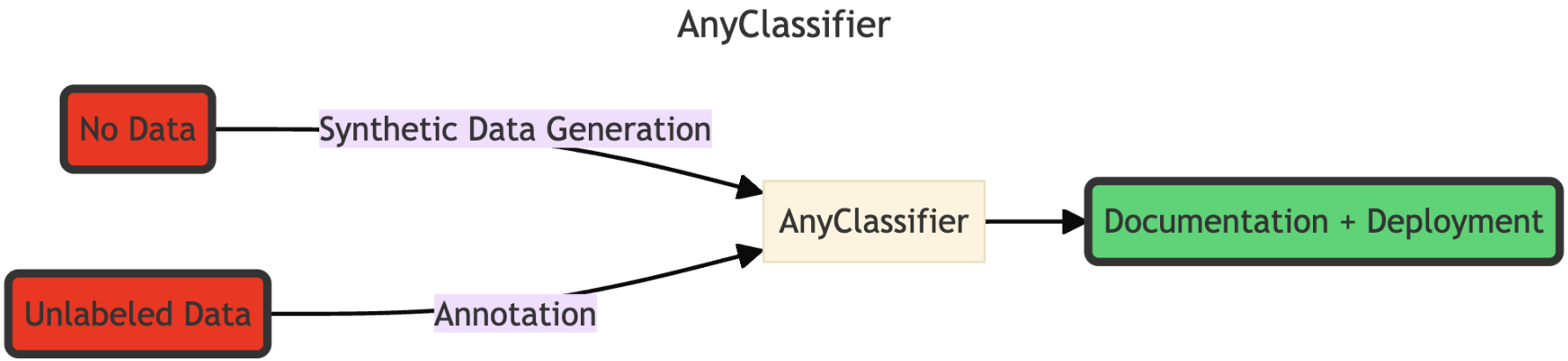
Challenges around social listening



→ and quickly changing topic space

How can we combine quality of LLMs with efficiency of bespoke models?

Enter solutions like `anyclassifier`



with many iterative steps going from model learning to evaluation in a circular fashion is overhead heavy and slow.

Common use cases

- **Data categorization or classification** – applying of *nomenclatures*

  Example: classification of gender or origins of names
- **Information retrieval applications** – retrieving a *needle from a haystack*

  Example: semantic search – memory expansion
- **Dimensionality reduction** – summarization of *large datasets*

  Example: document summaries
- **Dataset construction** – leaning on stock of embodied knowledge in LLM

  Example: production networks
- **Retrieval augmented dataset construction** – prompt and context into context

  Example: causal claims in economics
- **Reasoning tasks** – complex bipartite matching tasks with incomplete information

  Example: harmonisation of opinion polling datasets

Retrieval augmented generation