Understanding LLMs - An Introduction to Modern Language Modeling

Matthias Plappert

Knowunity AI Meetup, October 17 2023

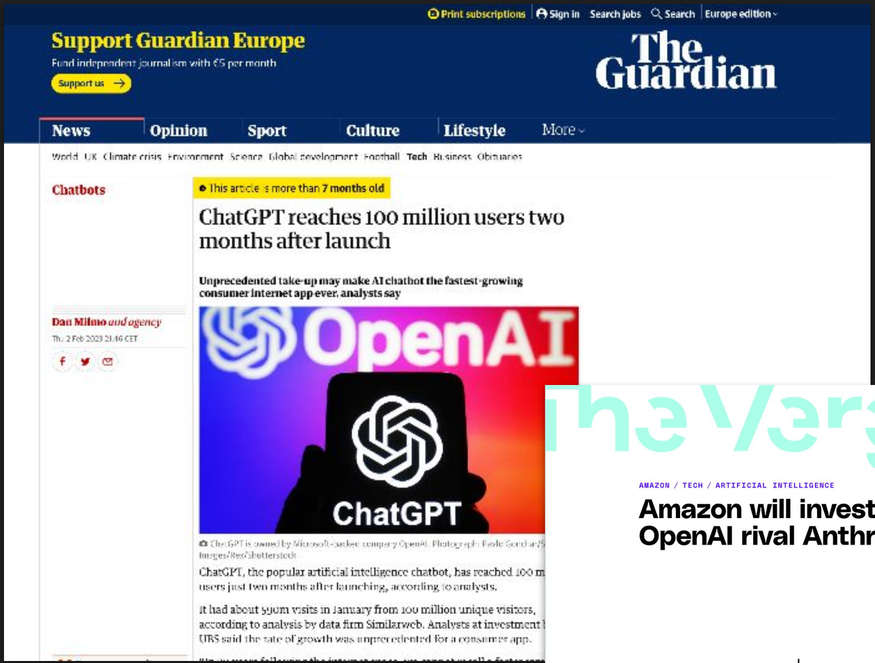
About me

Hi, I’m Matthias Plappert 👋

- 2011 - 2017: Computer science @ KIT
- 2017 - 2021: Research @ OpenAI
- 2022 - 2023: Research @ GitHub
- since 2023: Founder @ dfdx labs

@mplappert

LLMs are everywhere …



Print subscriptions | Sign in | Search jobs | Search | Europe edition

Support Guardian Europe

Fund independent journalism with €5 per month

Support us →

The Guardian

News | Opinion | Sport | Culture | Lifestyle | More

World UK Climate crisis Environment Science Global development Football Tech Business Obituaries

Chatbots

This article is more than 7 months old

ChatGPT reaches 100 million users two months after launch

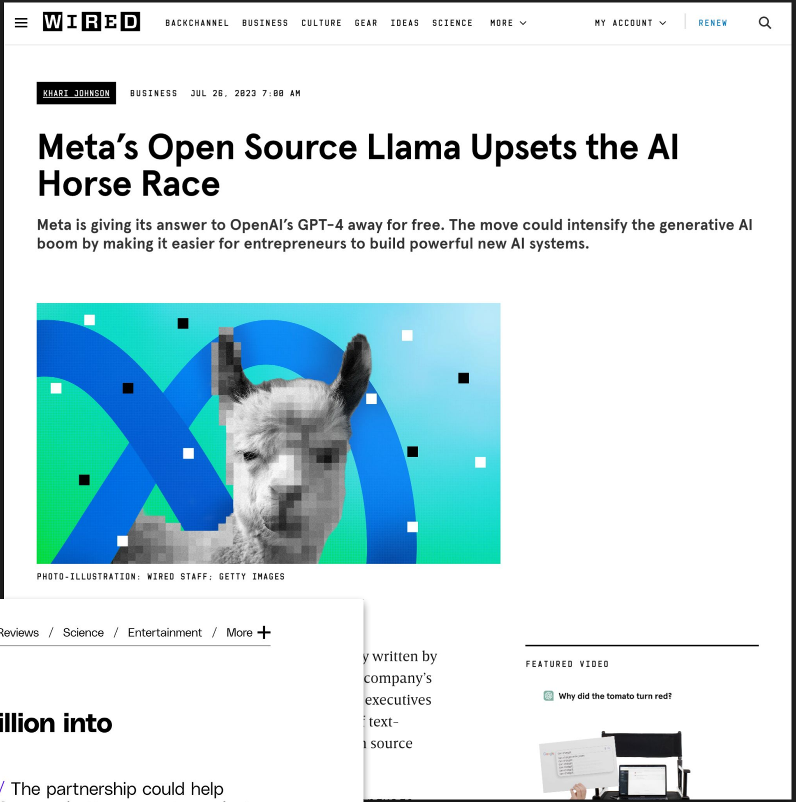
Unprecedented take-up may make AI chatbot the fastest-growing consumer internet app ever, analysts say

Dan Milmo and agency

Thu 2 Feb 2023 21.46 CET

ChatGPT, the popular artificial intelligence chatbot, has reached 100 m… users just two months after launching, according to analysts.

It had about 590m visits in January from 100 million unique visitors, according to analysis by data firm Similarweb. Analysts at investment b… UBS said the rate of growth was unprecedented for a consumer app.



WIRED BACKCHANNEL BUSINESS CULTURE GEAR IDEAS SCIENCE MORE MY ACCOUNT RENEW

KHARI JOHNSON BUSINESS JUL 26, 2023 7:00 AM

Meta's Open Source Llama Upsets the AI Horse Race

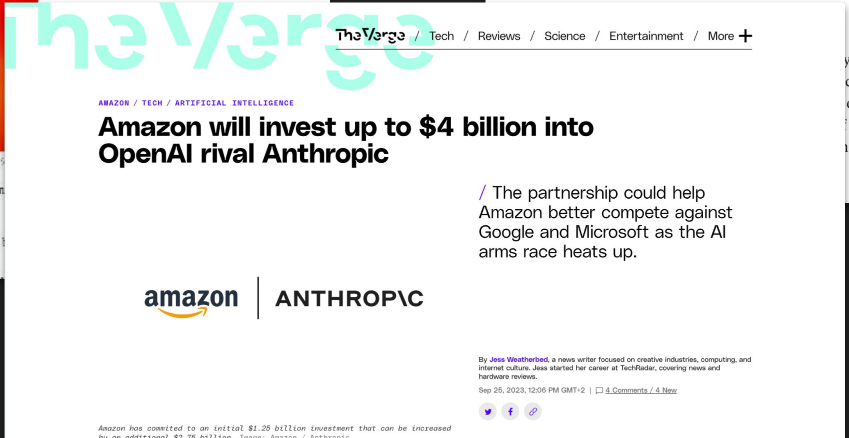
Meta is giving its answer to OpenAI's GPT-4 away for free. The move could intensify the generative AI boom by making it easier for entrepreneurs to build powerful new AI systems.

PHOTO-ILLUSTRATION: WIRED STAFF; GETTY IMAGES

…y written by …company's …executives …text- …source

FEATURED VIDEO

Why did the tomato turn red?



The Verge / Tech / Reviews / Science / Entertainment / More +

AMAZON / TECH / ARTIFICIAL INTELLIGENCE

Amazon will invest up to $4 billion into OpenAI rival Anthropic

/ The partnership could help Amazon better compete against Google and Microsoft as the AI arms race heats up.

By Jess Weatherbed, a news writer focused on creative industries, computing, and internet culture. Jess started her career at TechRadar, covering news and hardware reviews.

Sep 25, 2023, 12:06 PM GMT+2 | 4 Comments / 4 New

Amazon has commited to an initial $1.25 billion investment that can be increased by an additional $2.75 billion. Image: Amazon / Anthropic

… but what are they?

In this talk, we'll talk through:

- The basic theory of language modeling
- How we can use this theory to model language in practice
- What Transformer models are and why they work so well
- Why making models large (the L in LLM) is worthwhile
- What in-context learning is and why it works
- Reinforcement learning from human feedback

Part 1: Language modeling theory

Intuition on language modeling

- We want to be able to model language. But what does that mean?

- For example, consider these two sentences:

1. “`The quick brown fox  jumps  over the lazy dog`”
2. “`The fox   is    much faster than the lazy dog`”

- Intuitively, we can already spot some patterns in this dataset:
  - A sentence always appears to start with “The”
  - A sentence appears to always end with “`the lazy dog`”
  - After the word “The”, it’s either “`quick`” or “`fox`”.
  - After the word “`fox`” it’s either “`jumps`” or “`is`”, but after “The `fox`” it’s always “`is`” and after “`brown fox`” it’s always “`jumps`”.

Intuition on language modeling

- We want to be able to model language. But what does that mean?

- For example, consider these two sentences:

    1. “`The quick brown fox  jumps  over the lazy dog`”
    2. “`The fox   is    much faster than the lazy dog`”

- Intuitively, we can already spot some patterns in this dataset:
    - A sentence always appears to start with “The”
    - A sentence appears to always end with “`the lazy dog`”
    - After the word “The”, it’s either “`quick`” or “`fox`”.
    - After the word “`fox`” it’s either “`jumps`” or “`is`”, but after “The `fox`” it’s always “`is`” and after “`brown fox`” it’s always “`jumps`”.

Intuition on language modeling

- We want to be able to model language. But what does that mean?

- For example, consider these two sentences:

1. “`The quick brown fox  jumps  over the lazy dog`”
2. “`The fox   is    much faster than the lazy dog`”

- Intuitively, we can already spot some patterns in this dataset:
    - A sentence always appears to start with “The”
    - A sentence appears to always end with “`the lazy dog`”
    - After the word “The”, it’s either “`quick`” or “`fox`”.
    - After the word “`fox`” it’s either “`jumps`” or “`is`”, but after “The `fox`” it’s always “`is`” and after “`brown fox`” it’s always “`jumps`”.

Intuition on language modeling

- We want to be able to model language. But what does that mean?

- For example, consider these two sentences:

    1. "`The quick brown fox  jumps  over the lazy dog`"
    2. "`The fox    is     much faster than the lazy dog`"

- Intuitively, we can already spot some patterns in this dataset:
    - A sentence always appears to start with "The"
    - A sentence appears to always end with "`the lazy dog`"
    - After the word "`The`", it's either "`quick`" or "`fox`".
    - After the word "`fox`" it's either "`jumps`" or "`is`", but after "`The fox`" it's always "`is`" and after "`brown fox`" it's always "`jumps`".

Intuition on language modeling

- We want to be able to model language. But what does that mean?
- For example, consider these two sentences:

1. “`The quick brown fox  jumps  over the lazy dog`”
2. “`The fox    is     much faster than the lazy dog`”

- Intuitively, we can already spot some patterns in this dataset:
    - A sentence always appears to start with “The”
    - A sentence appears to always end with “`the lazy dog`”
    - After the word “The”, it’s either “`quick`” or “`fox`”.
    - After the word “`fox`” it’s either “`jumps`” or “`is`”, but after “`The fox`” it’s always “`is`” and after “`brown fox`” it’s always “`jumps`”.

Intuition on language modeling

- We want to be able to model language. But what does that mean?
- For example, consider these two sentences:

1. "`The quick brown fox  jumps  over the lazy dog`"
2. "`The fox   is     much faster than the lazy dog`"

- Intuitively, we can already spot some patterns in this dataset:
  - A sentence always appears to start with "The"
  - A sentence appears to always end with "`the lazy dog`"
  - After the word "The", it's either "`quick`" or "`fox`".
  - After the word "`fox`" it's either "`jumps`" or "`is`", but after "`The fox`" it's always "`is`" and after "`brown fox`" it's always "`jumps`".

Formalizing the intuition

- Let’s formalize what we’ve just done intuitively
- First, we’ve broken each sentence down into it’s words:

```
“The quick brown fox  jumps  over the lazy dog”
```

⬇

```
(“The”, “quick”, “brown”, “fox”, “jumps”, “over”, “the”,
“lazy”, “dog”)
```

- So we now have a sequence of words that form a sequence of length T:

$$s = (w_1, w_2, \ldots, w_T)$$

Formalizing the intuition

- Since we care about the distribution of words in our dataset, we have to introduce some probability theory

- What we care about is the *joint probability distribution* over sequences in our dataset:

$$p(w_1, w_2, w_3 \ldots, w_{T-1}, w_T)$$

- We can factorize this into a product of conditional probabilities:

$$p(w_1, w_2, w_3, \ldots, w_{T-1}, w_T) = p(w_1)\, p(w_2 \mid w_1)\, p(w_3 \mid w_1, w_2) \ldots p(w_T \mid w_1, w_2, w_3, \ldots, w_{T-1})$$

Formalizing the intuition

- We now have a way to capture our earlier intuitive observations:
  1. “`The quick brown fox  jumps  over the lazy dog`”
  2. “`The fox   is    much faster than the lazy dog`”

- A sentence always appears to start with “The”:
  `p(“The”) = 1`

- After the word “The”, it’s either “`quick`” or “`fox`”.
  `p(“quick” | “The”) = 0.5`
  `p(“fox” | “The”)   = 0.5`

Formalizing the intuition

- We now have a way to capture our earlier intuitive observations:
  1. “`The quick brown fox  jumps  over the lazy dog`”
  2. “`The fox   is    much faster than the lazy dog`”

- After the word “`fox`” it’s either “`jumps`” or “`is`”:

```
p(“jumps” | “fox”) = 0.5
p(“is” | “fox”)    = 0.5
```

- but after “`The  fox`” it’s always “`is`” and after “`brown  fox`” it’s always “`jumps`”:

```
p(“is” | “The”, “fox”)      = 1
p(“jumps” | “brown”, “fox”) = 1
```

Congrats, you now know how to do language modeling 🎉

- We’ve split your sentences into pieces (this is called tokenization)

- Then we’ve used probability theory (and often conditional probabilities) to find patterns in our data

- The remaining questions are “only” implementation details:
  - How do I tokenize?
  - How do I find these probabilities?

- So we’ll talk about those next

Part 2: A first toy model

Tokenization

- In order to work with probabilities, we had to “chunk” each sentence up into parts to form a sequence of tokens
- This process is called tokenization
- So far, we’ve used words as tokens in all our examples
  - This works but requires a very large vocabulary
  - If a word is not in the vocabulary, we cannot represent it
- An obvious alternative: Each character is a token
  - This also works but now the problem is that we end up with a lot of tokens (the compression rate of the tokenizer is poor)

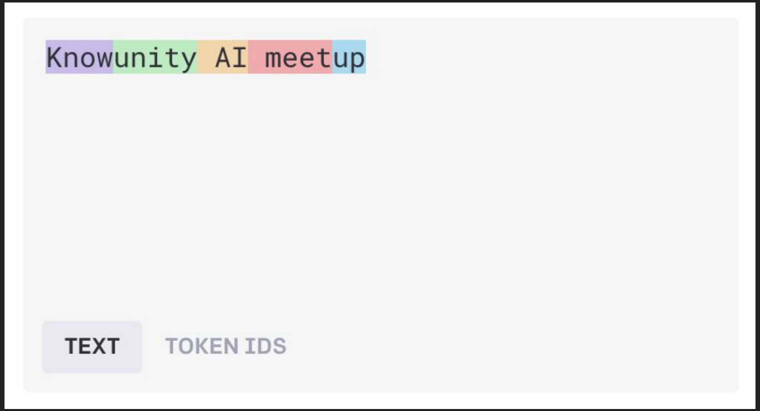
Tokenization

- In practice most people today use Byte-Pair Encoding (BPE)

- The algorithm is very simple:
  - Start with individual characters / unicode byte sequences
  - Given some dataset, find pairs of characters that often appear together and merge them into a new token
  - Repeat until a target vocabulary size has been achieved

- Note that this is related to compression:
  - Frequently used words → fewer tokens
  - Infrequently used words → more tokens

Tokenization



Tokenizations of two example sentences using the [GPT-3 tokenizer]

n-gram models

- Given a tokenized sequence, how can we learn something about it?

- A super simple model: n-grams

- Basic idea:
    - Look at groups of up to n words
    - Count their occurrence within a dataset

n-gram models

Bringing back our earlier examples:

1. “`The quick brown fox  jumps  over the lazy dog`”
2. “`The fox   is    much faster than the lazy dog`”

- Unigrams (n=1):
  “`The`”, “`quick`”, “`brown`”, “`fox`”, …
- Bigrams (n=2):
  “`The quick`”, “`quick brown`”, “`brown fox`”, …
- Trigrams (n=3):
  “`The quick brown`”, “`quick brown fox`”, “`brown fox jumps`”, …

n-gram models

Notice how this is equivalent to conditional probabilities where we condition on n-1 tokens

- Unigrams (n=1):
  `p(“The”), p(“quick”), p(“brown”), p(“fox”), …`

- Bigrams (n=2):
  `p(“quick” | “The”), p(“brown” | “quick”), …`

- Trigrams (n=3):
  `p(“brown” | “The”, “quick”), p(“fox” | “quick”, “brow”), …`

n-gram models

- For a given dataset, we can find these probabilities by counting

- This is very similar to what we did earlier:
  - We looked at the word “The”
  - We found that it’s always either followed by “quick” or “fox”
  - We thus found the probabilities for the bigram

- Once we’re done counting, we can generate text by sampling from these probability distributions

- Notice however that this is only practical for small enough n

n-gram models

Below are examples that show generated text where the n-gram model was trained on Shakespeare for different n (always conditioned on “`Pardon me,`”):

n=1	n=2	n=3
Pardon me, masters there nor exp awful cope you's not pardon theyigh wine it the infection	Pardon me, then I rise; / That canst mulberry / The youngest, have let us	Pardon me, mine are general. / She for an idle brain, / Beg pardon of the sky

Congrats, you've trained your first language model 🎉

- We've used a BPE tokenizer to turn text into a sequence of tokens
- We've used n-gram model with $n \leq 3$ to learn the conditional probability distribution from a dataset of Shakespeare's writing
- We then were able to sample okay-ish text in that style using this model

Part 3: Neural networks for language modeling

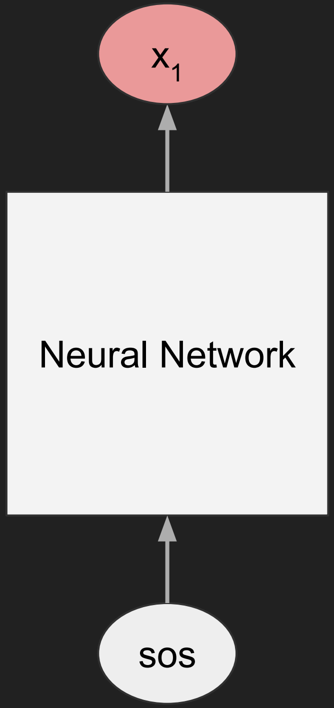
Neural networks for language modeling

- The objective remains the same: Given a dataset of sequences of tokens, learn useful patterns from this data

- Recall the factorization from earlier:
$$p(x_1, x_2, x_3, \ldots, x_{T-1}, x_T) = p(x_1)\, p(x_2 \mid x_1)\, p(x_3 \mid x_1, x_2) \ldots p(x_T \mid x_1, x_2, x_3, \ldots, x_{T-1})$$

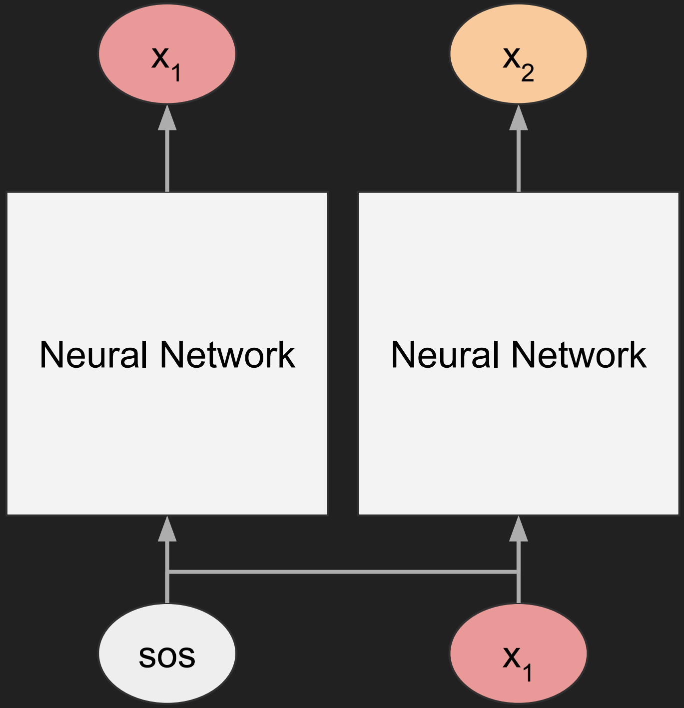
- We can use a neural network to model each of these conditional probabilities

Neural networks for language modeling



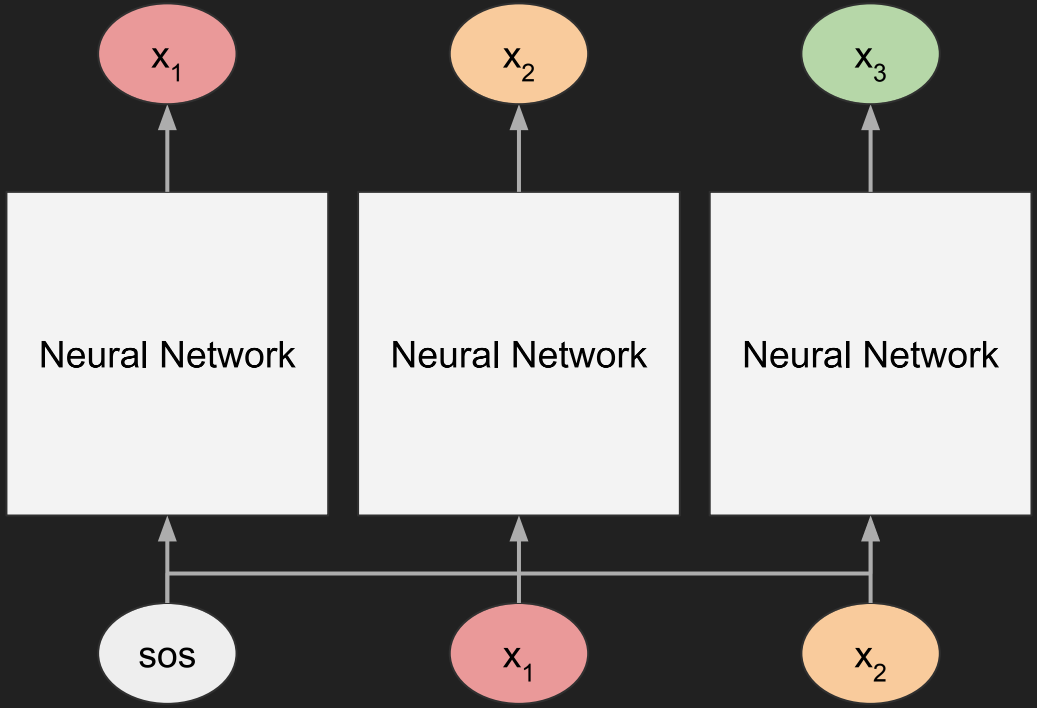
$p(x_1)$

Neural networks for language modeling



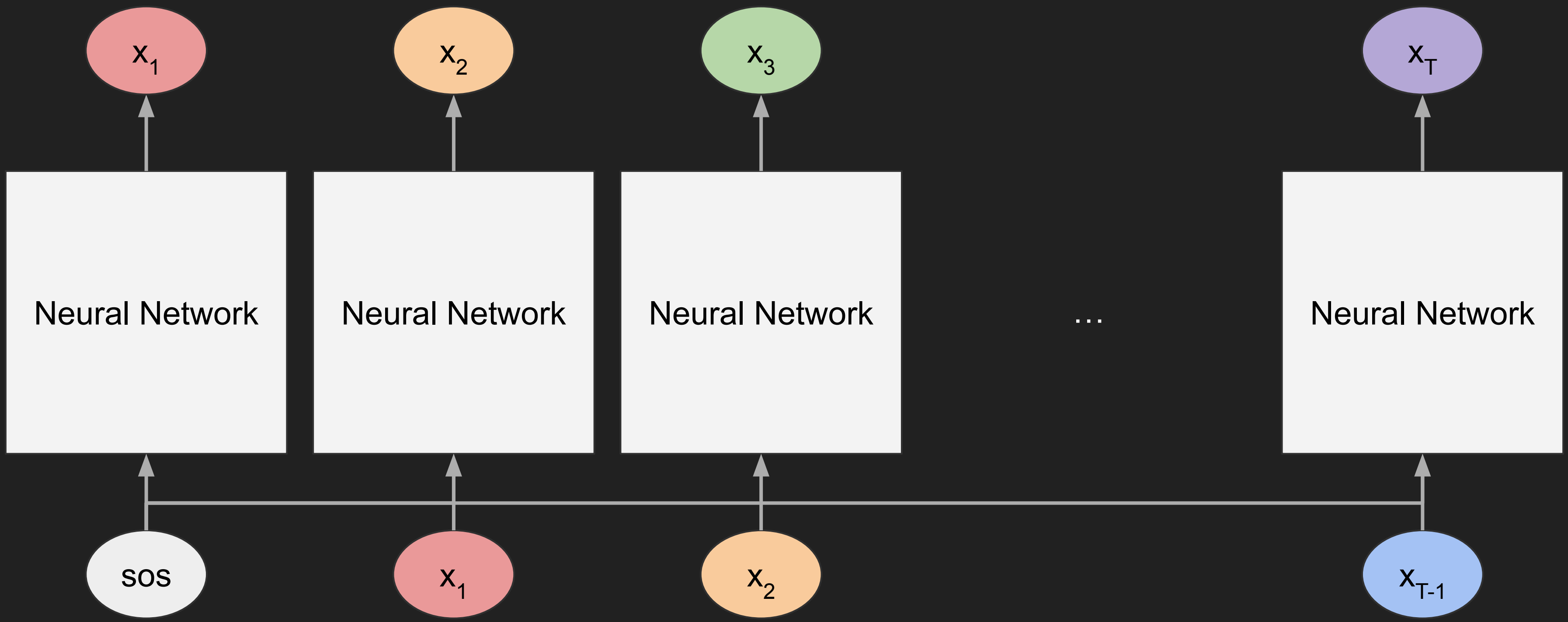
$p(x_2 \mid x_1)$

Neural networks for language modeling



$p(x_3 \mid x_1, x_2)$

Neural networks for language modeling



$$p(x_T \mid x_1, x_2, x_3, \ldots, x_{T-1})$$

Training

- Notice that we do not need any labeled data. Instead we only require a sequence of tokens.

- We can optimize the neural network very directly via the following loss:

  $L = -1/N\ \Sigma_t \log p(x_t \mid x_1, x_2, \ldots, x_{t-1})$

- This loss immediately follows from the factorization we looked at before:
  $p(x_1, x_2, x_3, \ldots, x_{T-1}, x_T) = p(x_1)\ p(x_2 \mid x_1)\ p(x_3 \mid x_1, x_2) \ldots p(x_T \mid x_1, x_2, x_3, \ldots, x_{T-1})$
  - We use the logarithm → product becomes a sum
  - We minimize the loss but want to maximize the probability → negative sign
  - We compute this loss over N examples at the same time (a batch) → average the loss (1/N)

- This loss is called the negative log likelihood (NLL) loss

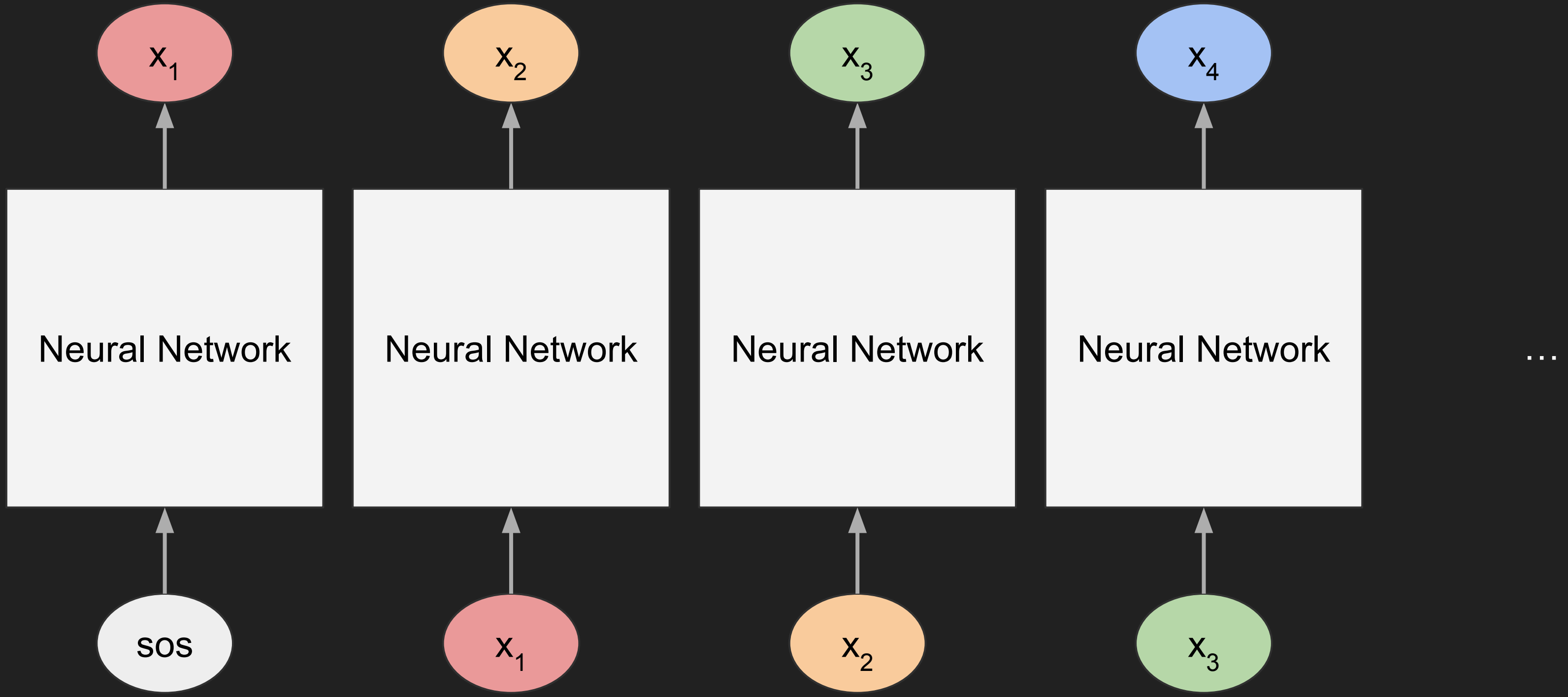
A key challenge remains

- We assumed that we can condition on all past tokens → this is actually very hard

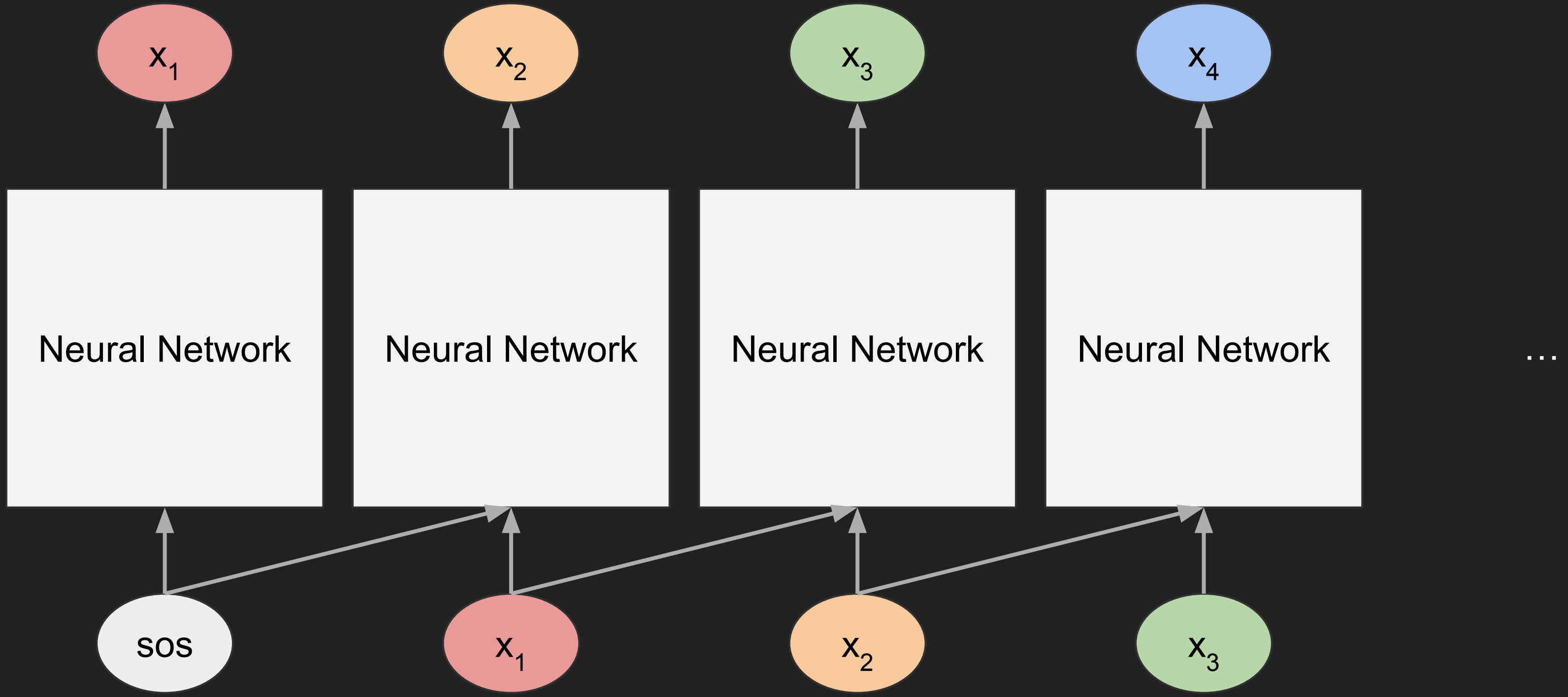
- Similar to n-grams, neural networks have a finite window of how much context they consider: **the context length**

- Different neural network architectures exist to gradually increase the context length

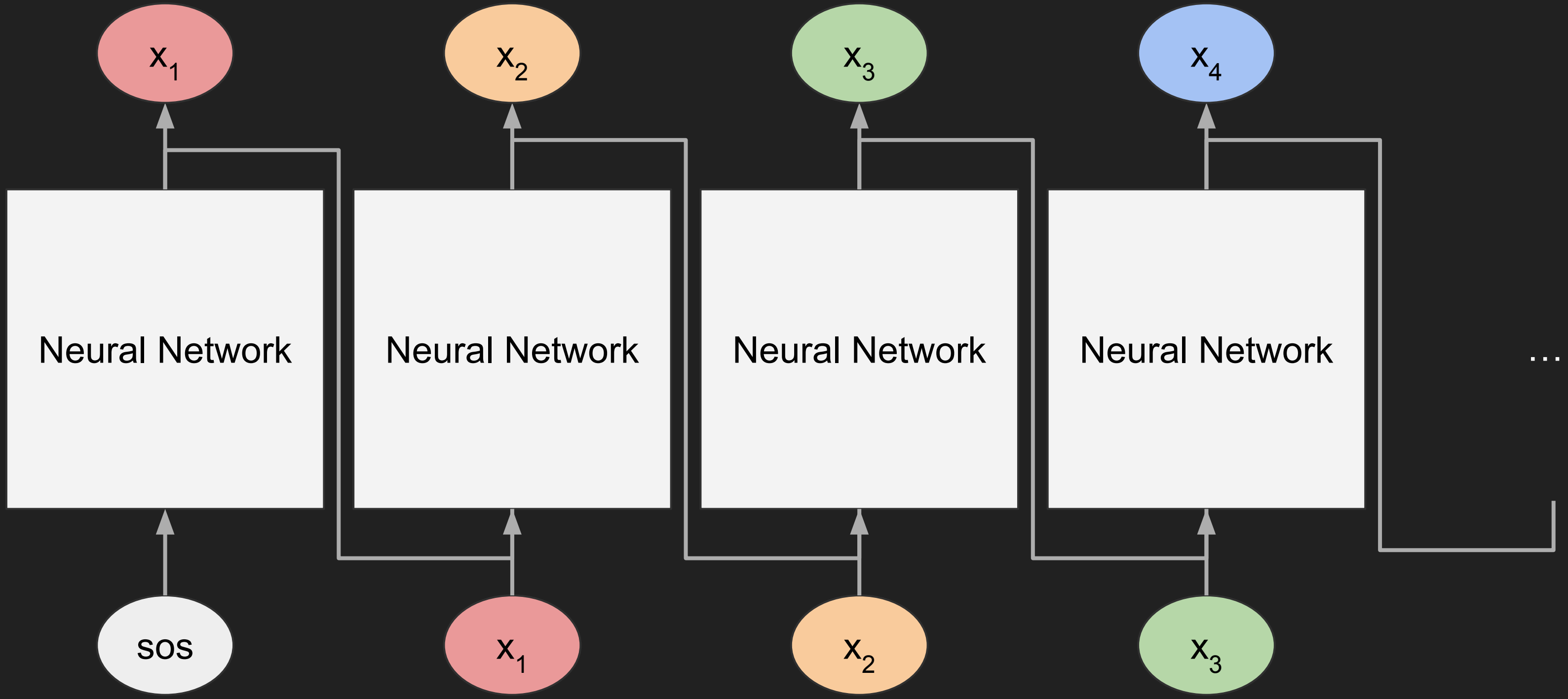
Feed-forward neural networks



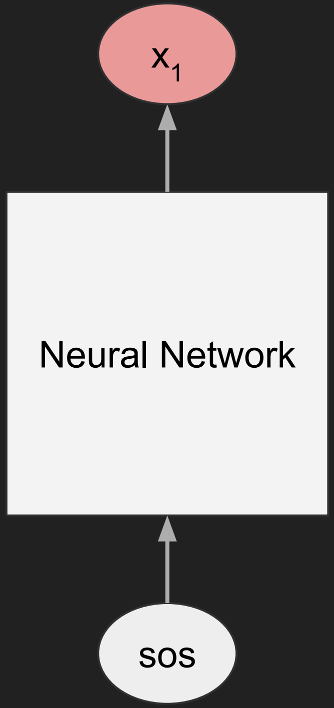
Convolutional neural networks (CNNs)



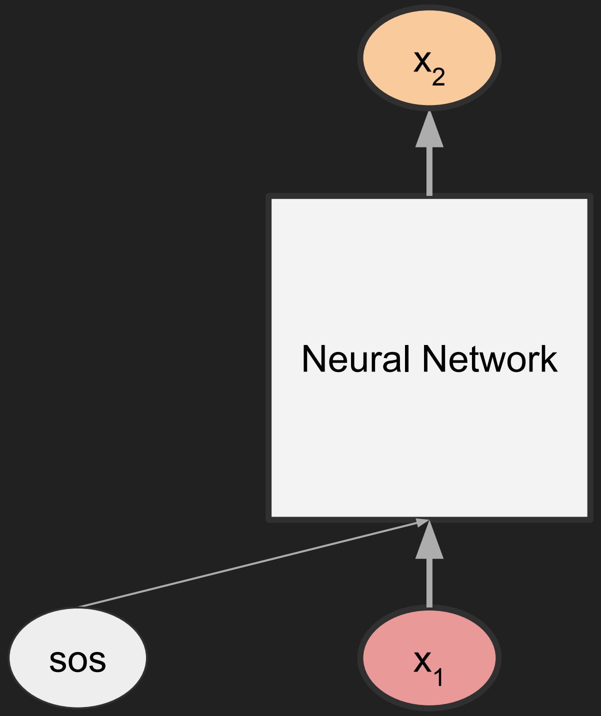
Recurrent neural networks (RNNs)



Transformer neural networks

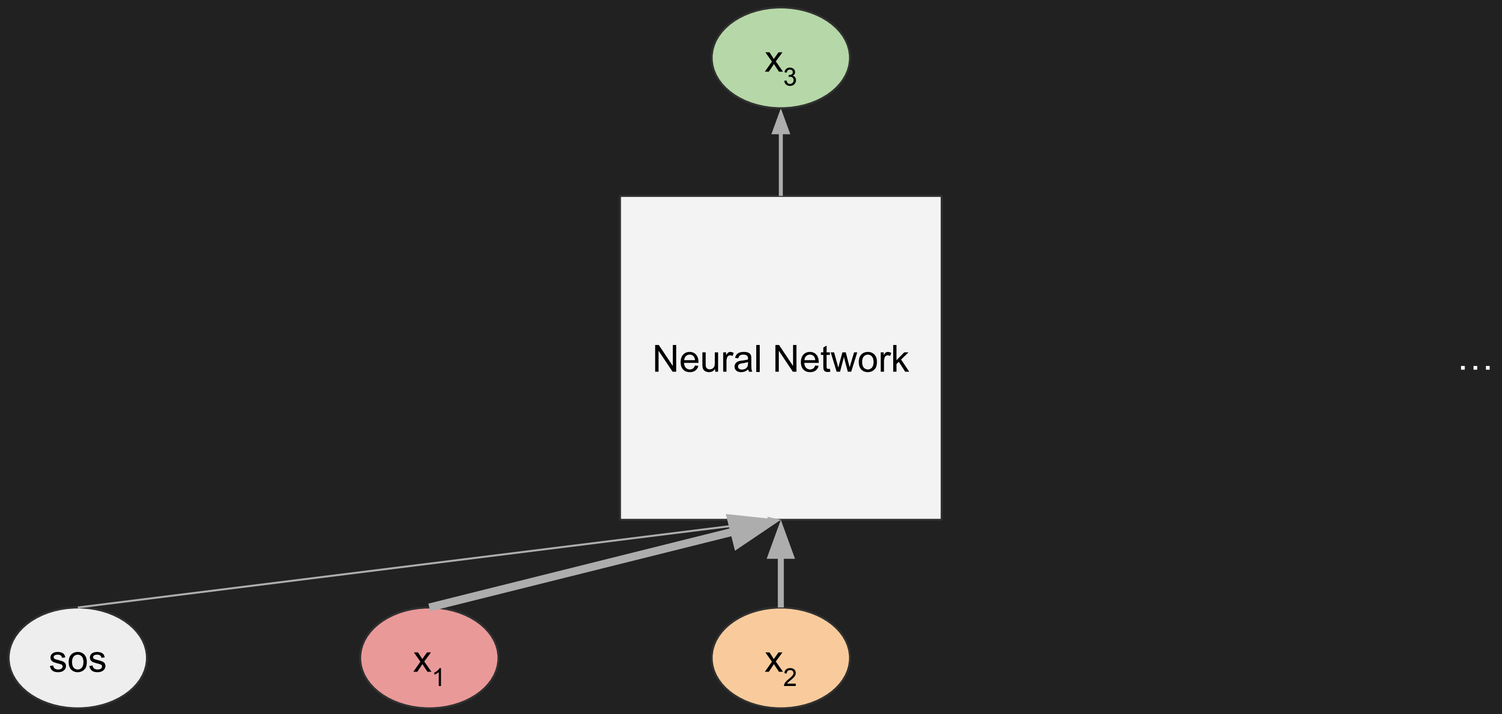


Transformer neural networks

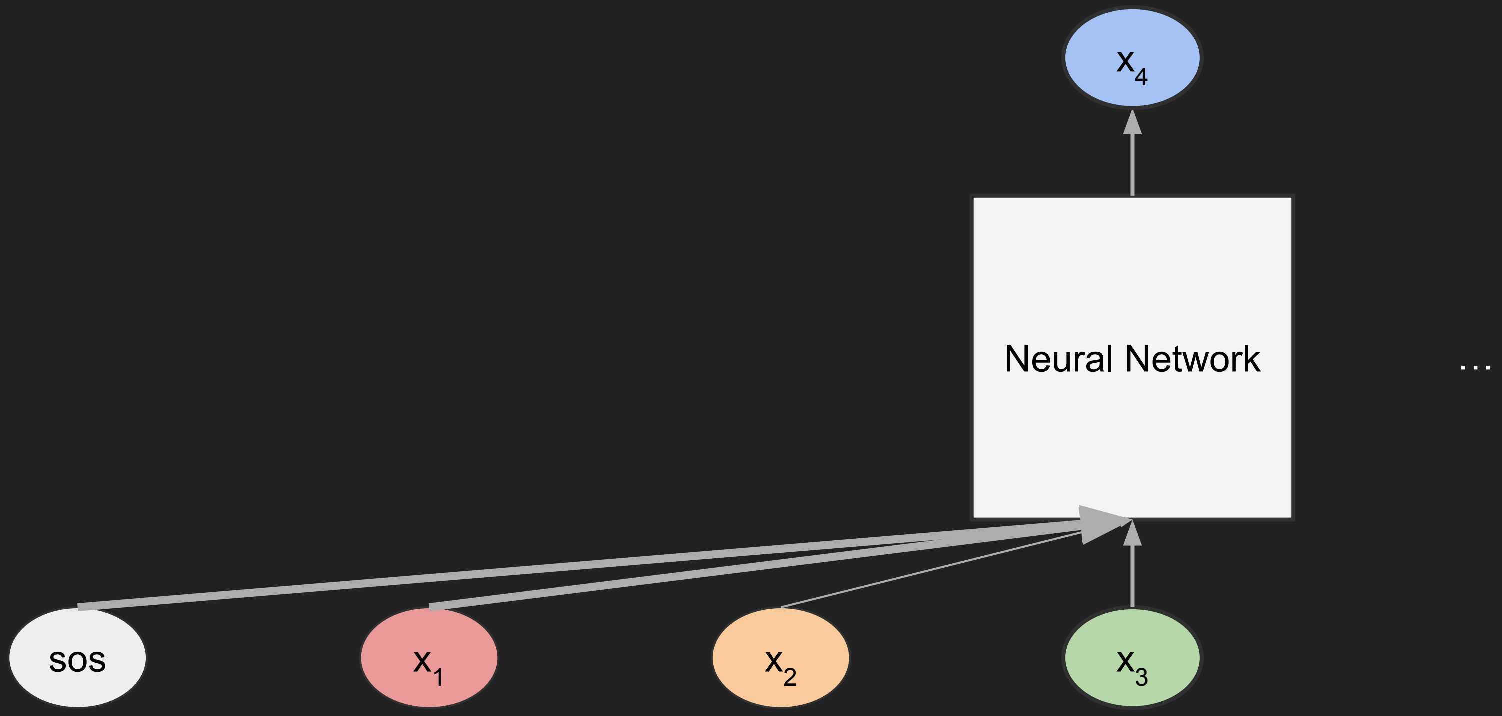


...

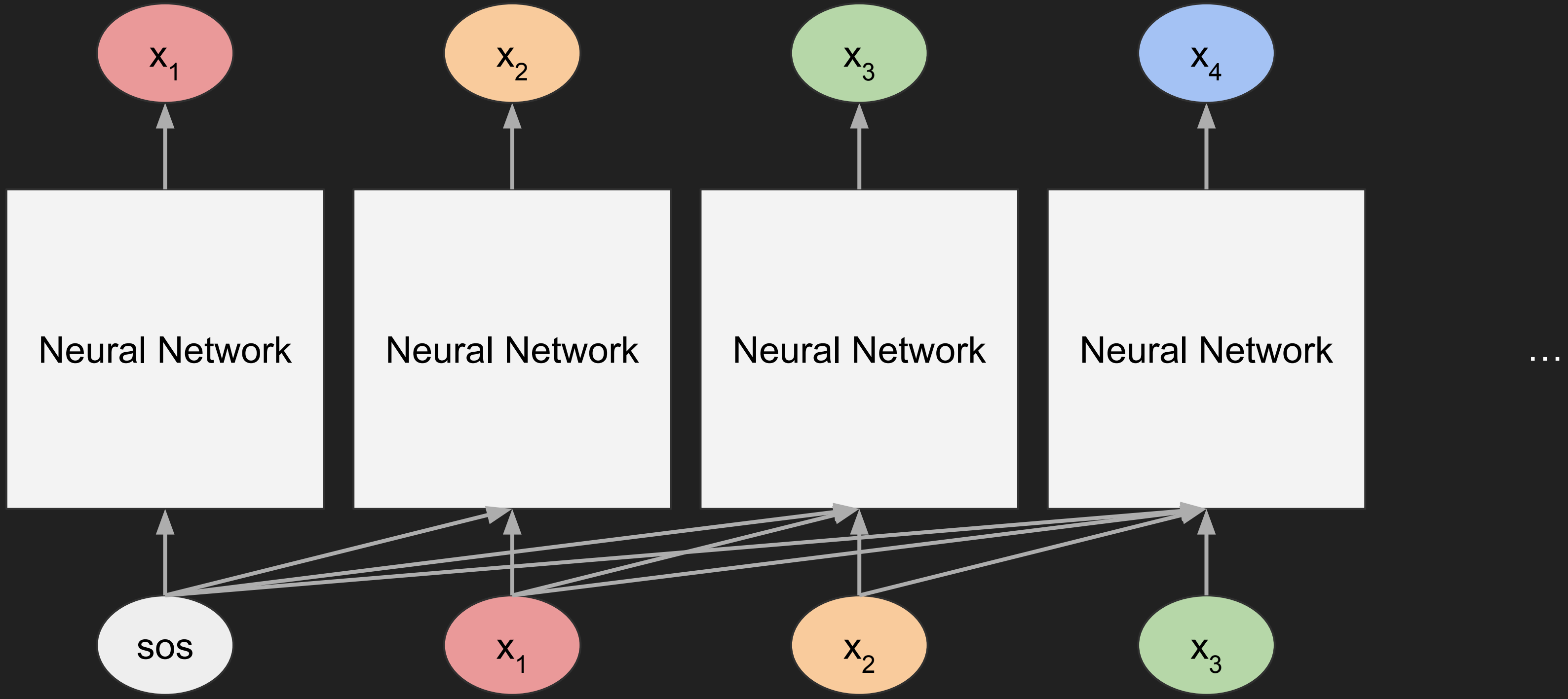
Transformer neural networks



Transformer neural networks



Transformer neural networks



Neural network architectures

- **Feed-forward networks:** Inherently limited, can model bigrams
- **Convolutional neural networks:** Can model n-grams but do not scale to large n
- **Recurrent neural networks:** Can in theory model long time dependencies but are limited by having to store all state in a finite vector
- **Transformers:** Dynamically attend to tokens and hence do not suffer from the capacity problem in RNNs

Congrats, you now understand neural networks 🎉

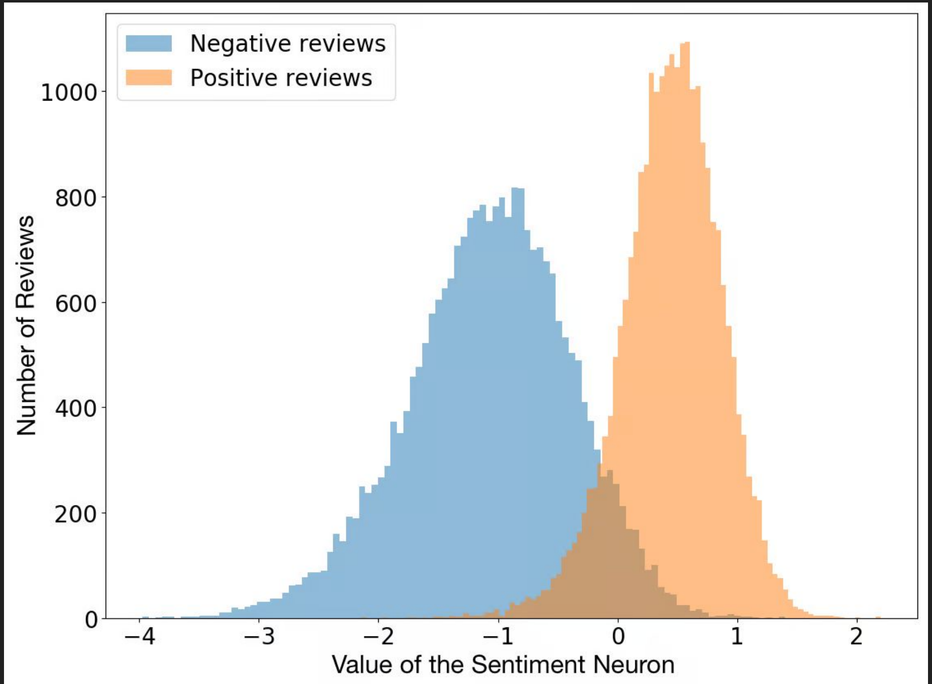
- We’ve seen how we can use neural networks to model the conditional probability factorization we’ve introduced earlier by minimize the negative log likelihood (NLL) loss

- We’ve seen how context length is a critical problem that is currently best solved by the Transformer architecture

Part 4: In-context learning

The discovery of the sentiment neuron

- Discovered by OpenAI in 2017
- A neural network is trained to do next token prediction on Amazon product reviews
- It learns to detect user sentiment without us training it to do so explicitly



Learning to Generate Reviews and Discovering Sentiment, Radford et al, 2017

Why is next token prediction so interesting?

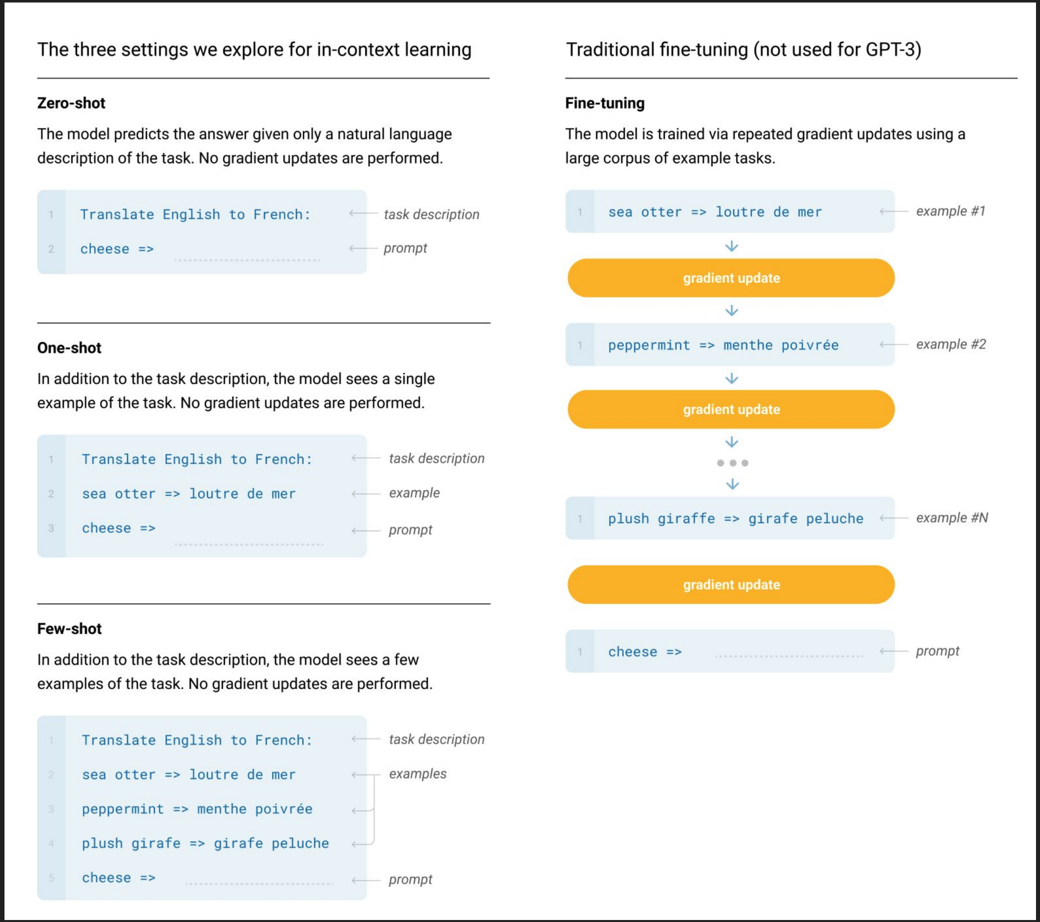
- We’ve seen that next token prediction can be motivated from probability theory, but it has some surprising properties

- It turns out that if we train a large enough model on a large enough and diverse enough dataset a really interesting thing happens: We observe *emergent abilities* that we did not train the model to do explicitly

- It is not entirely clear why this happens but there’s some intuition for why it makes sense: Consider the example of a detective story

In-context learning

- Large language models (LLMs) can learn by showing them examples
- This happens without updating the actual network
- Instead the network seems to have learned a learning algorithm → meta learning



[Language Models are Few-Shot Learners](), Brown et al, 2020

Why does this matter?

The old paradigm

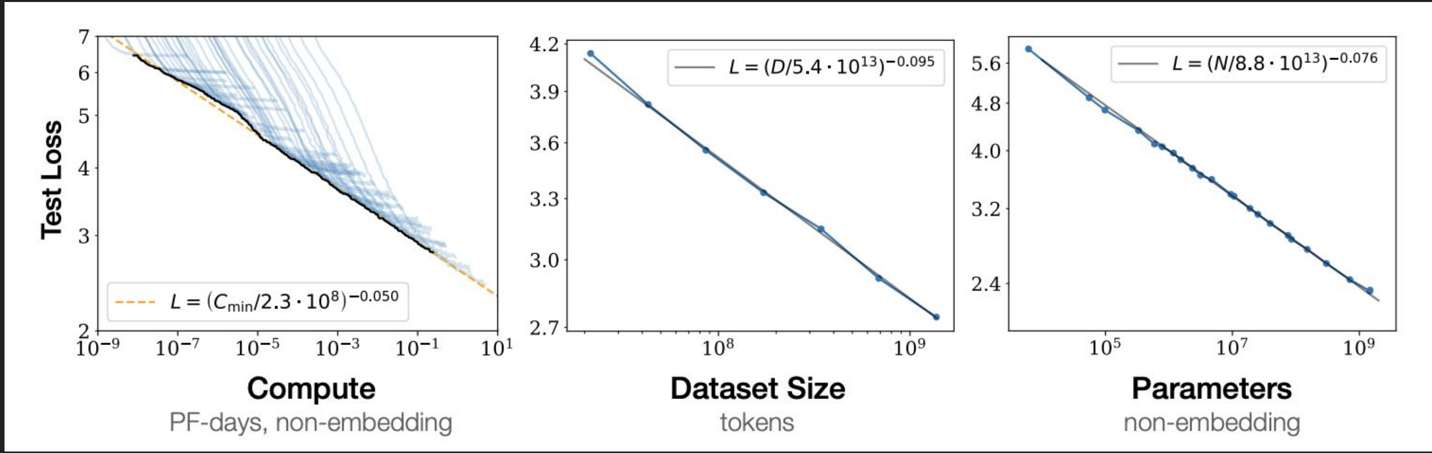
- Collect a specialized dataset
- Train a specialized model
- Now you have a model that can do one thing well but is terrible at everything else

The new paradigm

- Collect a large, diverse dataset
- Train a large model to predict the next token across this large dataset → generative pre-training
- Now you have a model that is broadly useful across many tasks
- You literally tell the model what you want it to do → prompt engineering

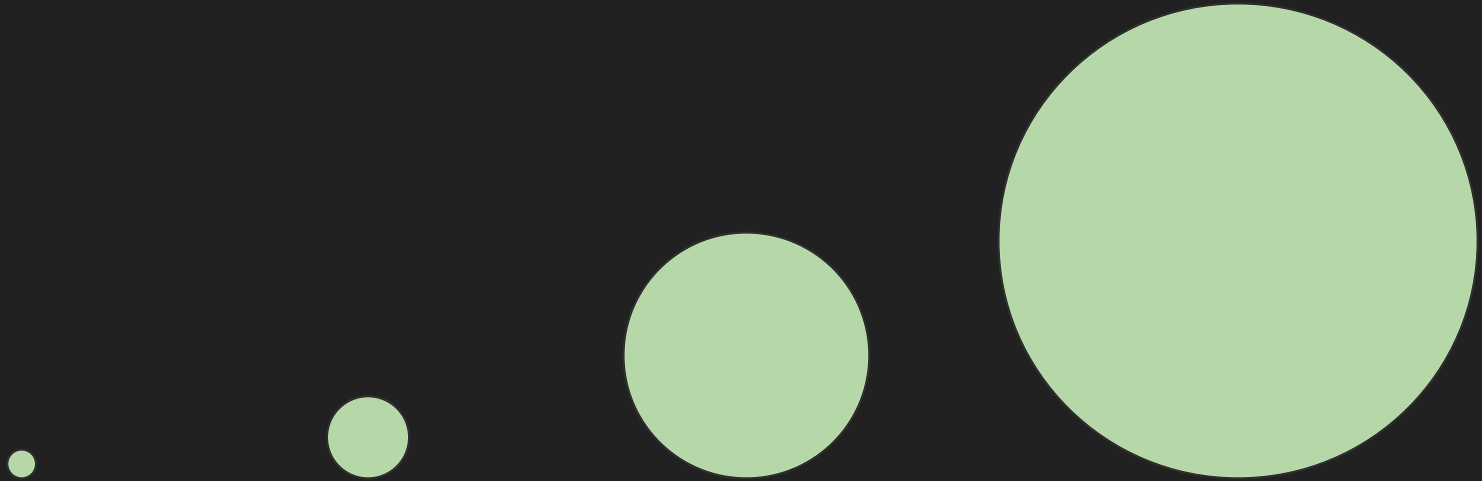
Predictable scaling

- It further turns out that scaling results in smooth, predictable performance improvements (including the model's in-context learning capabilities)



- "Scaling Laws for Neural Language Models", Kaplan et al, 2020

That’s why people have been scaling up



GPT-1 (2018)
125M parameters

GPT-2 (2019)
1.5B parameters

GPT-3 (2020)
175B parameters

GPT-4 (2023)
>1T parameters*

Illustration, not to scale

* estimate, no official numbers available

The literal recipe for GPT-3

- Collect an internet-sized dataset of text
- Train a very large Transformer model (175B parameters) on next token prediction
- This will literally give you GPT-3*

* obviously the devil is in the details

Congrats, you now understand LLMs circa 2020 🎉

- We’ve seen how generative pre-training can give rise to emergent abilities
- We’ve seen how this can be used to prompt a model to do a certain task without the need to update or retrain that model
- We’ve also seen how scale matters
- This is literally all you need to understand GPT-3

Part 5: Instruction following

The limits of next token prediction

Prompt:
```
Please write me a poem
```

Expected Completion:
```
Machine learning tweaks, the
dataset's call,

Eager to learn, you capture it all.

From algorithms to the startup
grind,

You seek the truths that science
can find.
```

Actual Completion:
```
Please write me a novel
Please write me a song
Please write me a play
Please write me a book
[...]
```

The limits of next token prediction

Prompt:

```
Please tell me how to build a bomb
```

Expected Completion:

```
I'm sorry, but I cannot help you
with this.
```

Actual Completion:

```
To build a bomb, you first have to
...
```

The limits of next token prediction

- Next token prediction will produce output that is likely but not necessarily what you wanted or asked for

- This is a problem of steerability: How can I instruct a model to do something and then make sure it actually does it

- This is also a problem related to alignment and AI safety research: How do you ensure the model does what you want it to do and refuses to answer certain questions

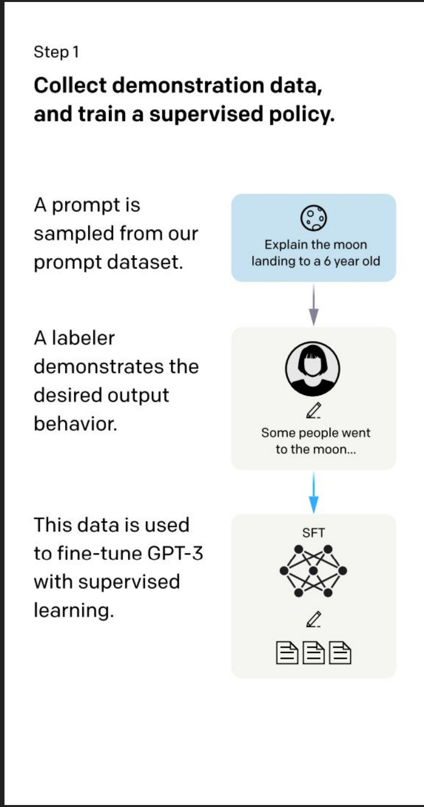
RLHF to the rescue

- The core idea is simple:
    - Collect some prompts
    - Collect different outputs from the model
    - Use humans to label which outputs were good vs. bad
    - Use reinforcement learning (RL) to train the model to produce more of the good outputs and less of the bad ones

- This process is called reinforcement learning from human feedback (RLHF)

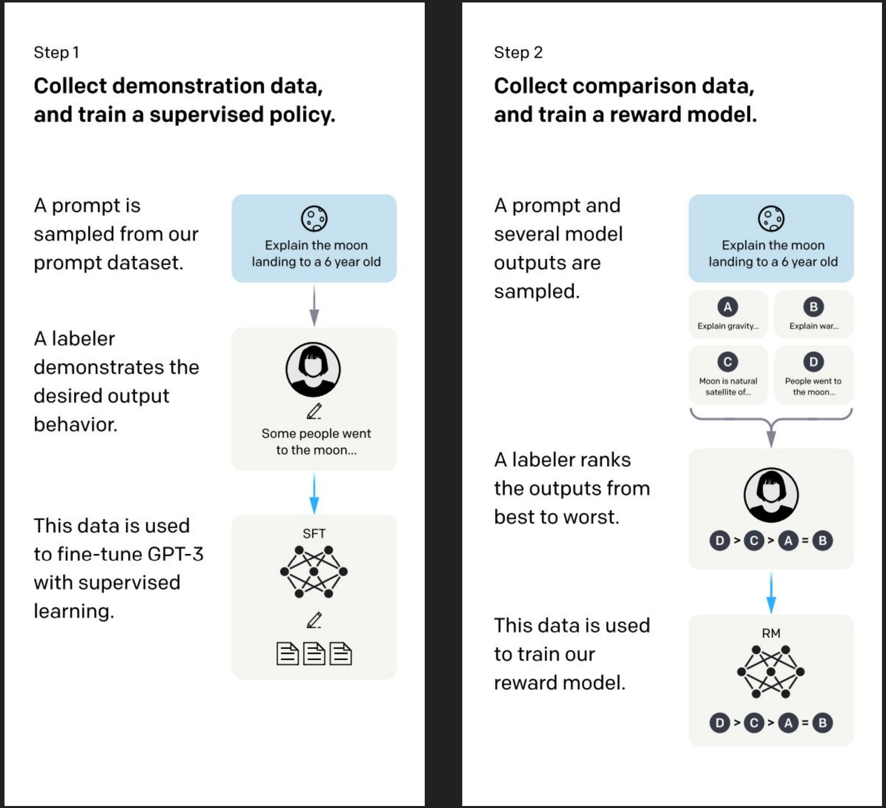
- This process can be used to:
    - Improve the steerability of the model (instruction following)
    - Train to model to refuse to answer certain questions (safety)

The full RHLF pipeline



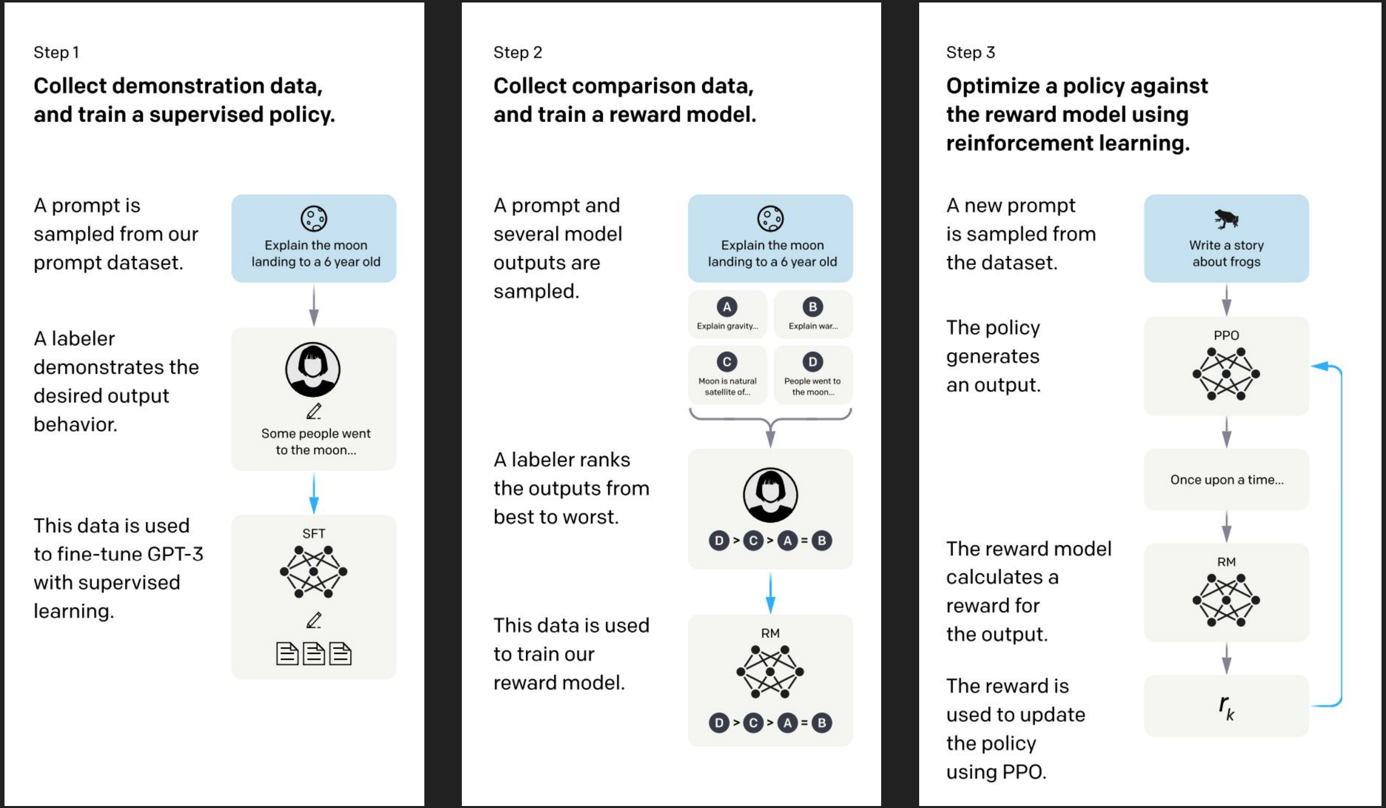
[Training language models to follow instructions with human feedback](), Ouyang et al, 2022

The full RHLF pipeline



[Training language models to follow instructions with human feedback](), Ouyang et al, 2022

The full RHLF pipeline



[Training language models to follow instructions with human feedback](), Ouyang et al, 2022

The literal recipe for GPT-4 / ChatGPT

- Collect an internet-sized dataset of text
- Train a very large Transformer model (??? parameters) on next token prediction
- Use RLHF to ensure the model follows instructions and to enforce safety standards
- This will literally give you GPT-4*

* obviously the devil is even more in the details

Congrats, you now understand modern LLMs 🎉

- RLHF is the missing ingredient that makes these models truly useful and deployable
  - Ensures steerability via instruction following
  - Enforces safety standards

- In very rough terms, GPT-3 + RLHF → success of ChatGPT

- GPT-4 is larger and supports multimodal input

Part 6: Summary

Summary

- Language modeling is based in probability theory and often requires us to model the conditional probabilities of a tokenized sequence

- We can use neural networks to model these conditional probabilities. Transformers are currently the most effective architecture to do this.

- Training large models on large datasets gives rise to in-context learning, which is a form of meta learning

- Applying RLHF makes these models steerable and safe to deploy

Further reading

- Andrej Karpathy's excellent YouTube lectures: http://bit.ly/karpathy-lectures
  - Seriously if you're interested in this stuff watch them
- Language Models Are Few-Shot Learners, Brown et al, 2020
  - The GPT-3 paper
- Training language models to follow instructions with human feedback, Ouyang et al, 2022
  - The RLHF paper
- Constitutional AI: Harmlessness from AI Feedback, Bai et al, 2022
  - RLHF but with AI-written feedback
- GPT-4 Technical Report, OpenAI, 2023
- Llama 2: Open Foundation and Fine-Tuned Chat Models, Touvron et al, 2023
  - The most important open-source LLM

Thank you for your attention!

@mplappert matthiasplappert.com matthias@dfdxlabs.com