Generative AI: Foundations

CPEN 322

Agenda

00

Foundations of ML

Speedrun
key ideas in ML

01

Transformer Architecture

Core ideas and
notable examples

02

Notable LLMs

T5, GPT, Chinchilla, et al

03

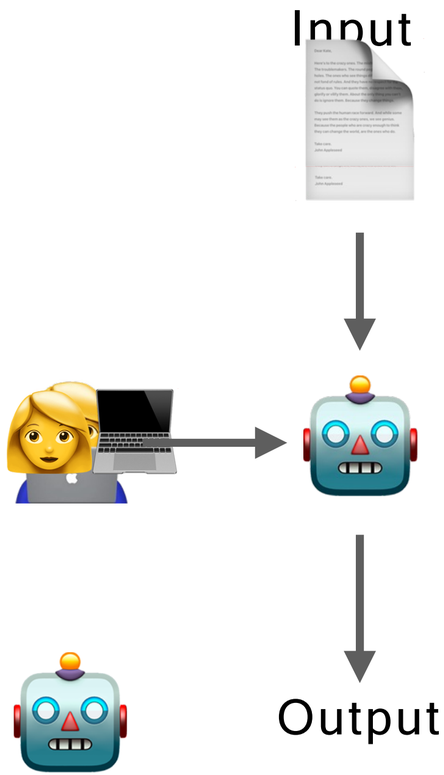
Training & Inference

Running a Transformer

00

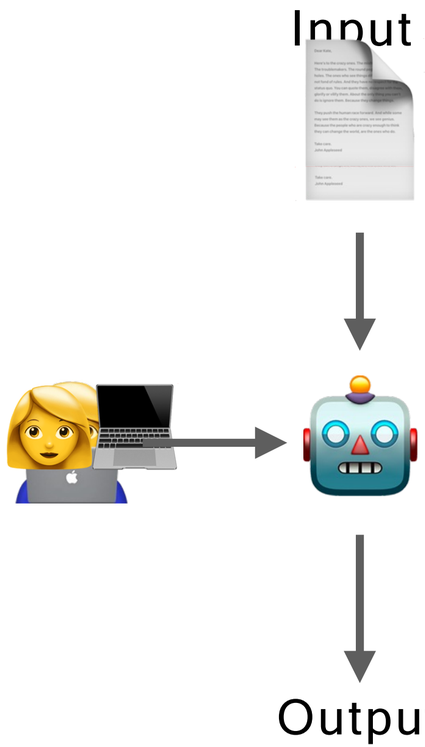
Foundations of Machine Learning

Traditional Programming

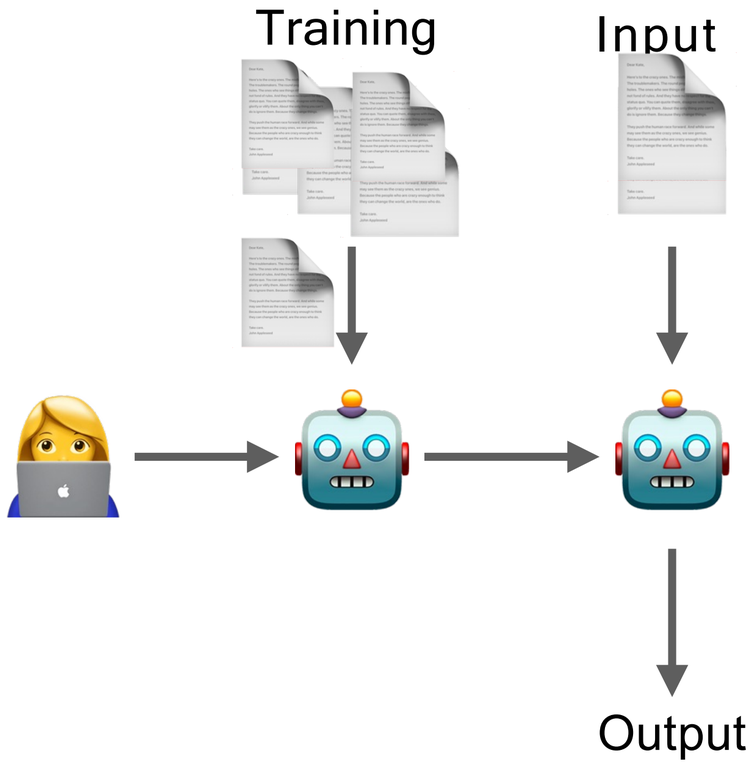


Software 1.0

Traditional Programming vs Machine Learning



Software 1.0



Software 2.0

Types of Machine Learning

Unsupervised Learning

Learn structure of data to predict (e.g., clustering)

Supervised Learning

Learn how data maps to labels to recognize or predict

Reinforcement Learning

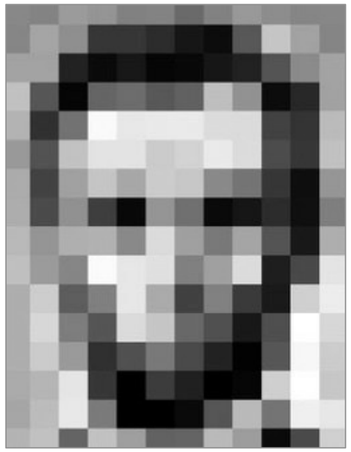
Learn how to act in an environment to obtain reward

"This product does what it is supposed __"

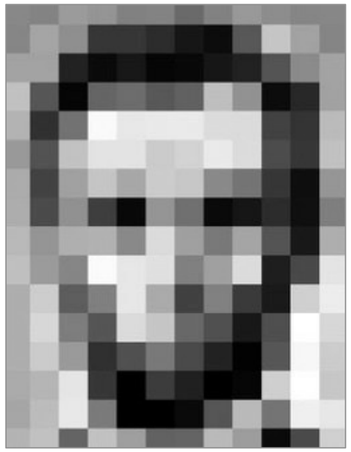
→ *cat*

→ *"Hey Siri"*

Embedding: Inputs and outputs are always numbers

	Input	Output
What we see		"Lincoln"

Embedding: Inputs and outputs are always numbers

	Input	Output
What we see		"Lincoln"
What the machine "sees"	157 153 174 168 150 152 129 151 172 161 155 155 182 163 74 75 62 33 17 110 210 180 180 180 50 14 34 6 10 33 48 106 159 206 109 5 124 131 111 120 204 166 15 56 194 68 137 251 237 239 239 228 227 87 71 172 105 207 233 233 214 220 239 228 98 74 188 88 179 209 185 215 211 158 139 75 20 189 97 165 84 10 168 134 11 31 62 22 199 168 191 193 158 227 178 143 182 106 36 205 174 155 252 236 231 149 178 228 43 95 190 216 116 149 236 187 86 150 79 38 218 190 224 147 108 227 210 127 102 36 101 255	[76, 105, 110, 99, 111, 108, 110]

Why is this hard?

- Infinite variety of inputs can all mean the same thing
- Meaningful differences can be tiny
- Structure of the world is complex

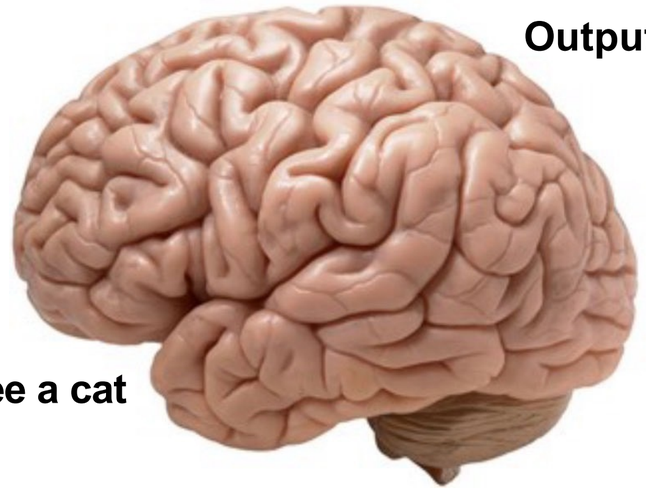
"I loved this movie"

"As good as The Godfather"

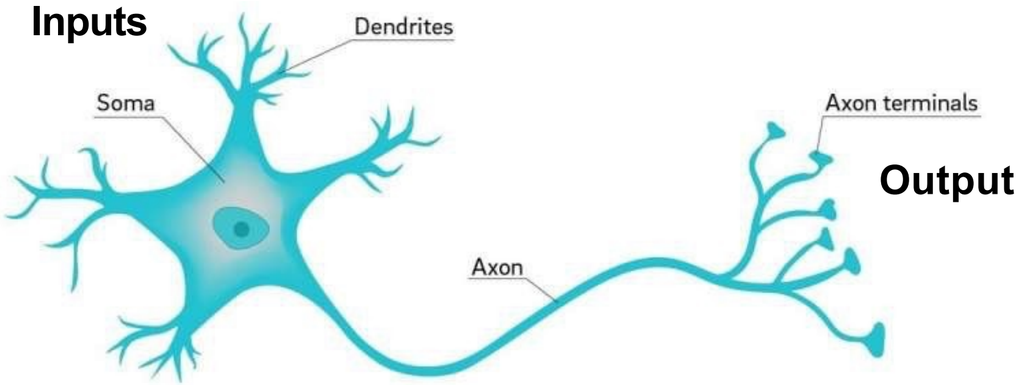
How is it done?

- Many methods for Machine Learning
  - Logistic Regression
  - Support Vector Machines
  - Decision Trees
- But one is dominant
  - Neural Networks (also called Deep Learning)

Inspiration



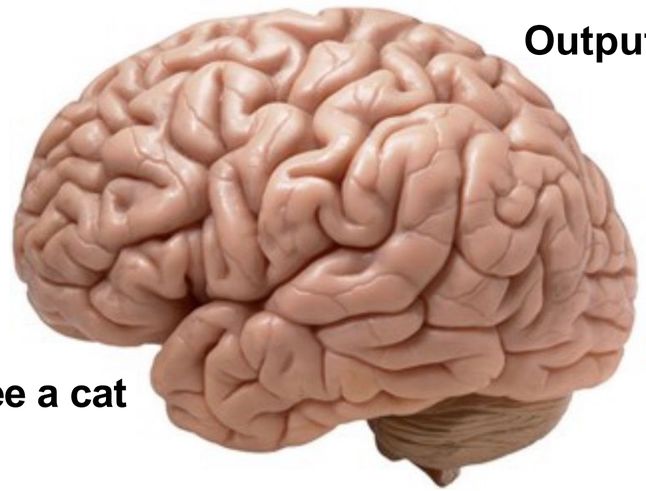
https://www.the-scientist.com/the-nutshell/what-made-human-brains-so-big-36663



https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html

- Inspired by what we know to be intelligent: the **brain**
- The brain is composed of billions of **neurons**
- Each neuron receives electrical **inputs** and sends an electrical **output**
- The brain itself has high-level inputs and outputs

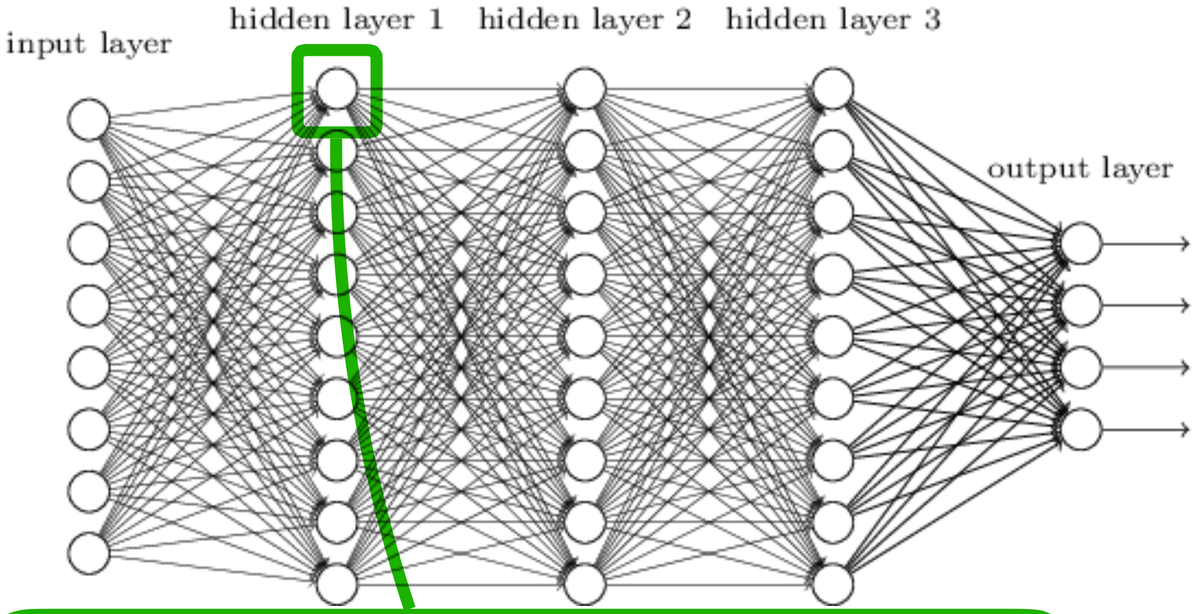
Formalization

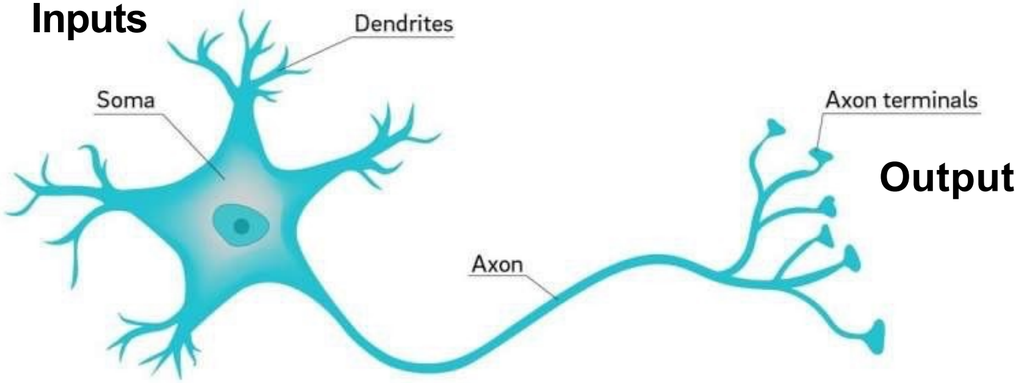


Output: say "cat"

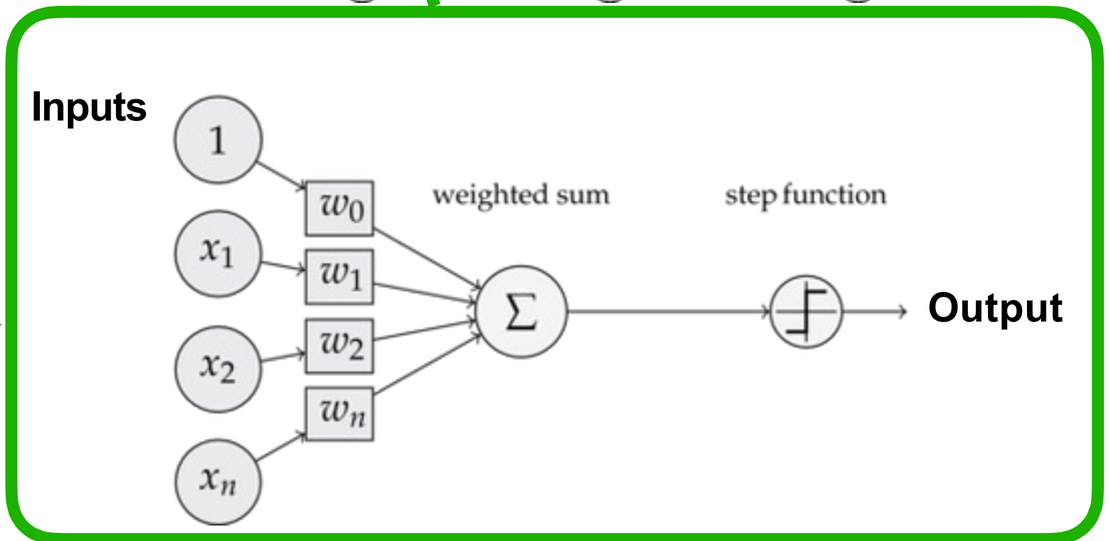
Input: see a cat

https://www.the-scientist.com/the-nutshell/what-made-human-brains-so-big-36663



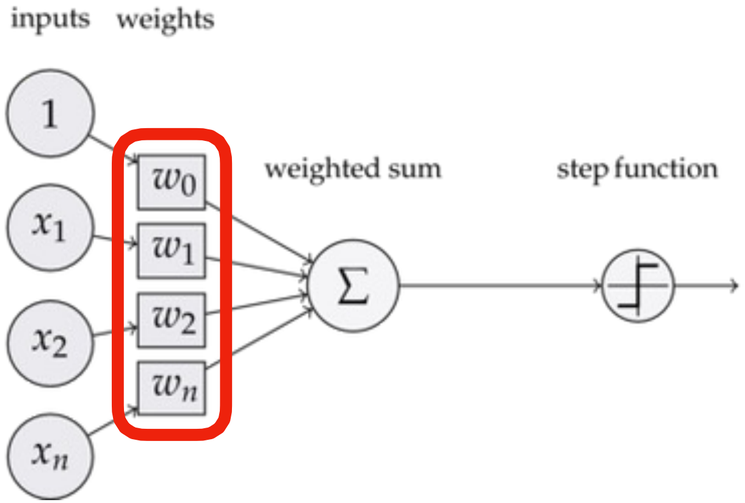


https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html



https://www.jessicayung.com/explaining-tensorflow-code-for-a-multilayer-perceptron/

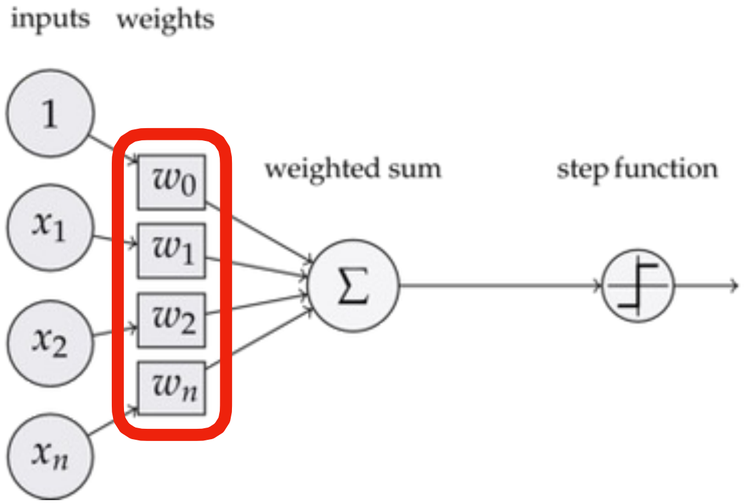
A "perceptron" is a vector of numbers




```
[0.58  0.841 0.835 0.18 ]
```

https://www.jessicayung.com/explaining-tensorflow-code-for-a-multilayer-perceptron/

Step Function

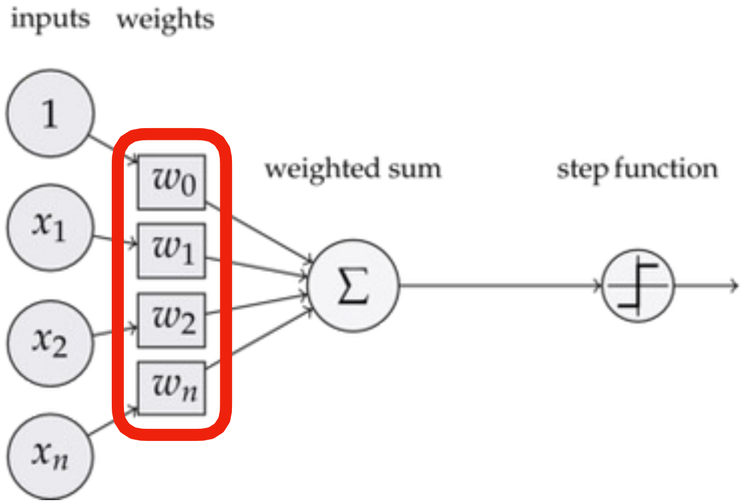



```
[0.58  0.841 0.835 0.18 ]
```

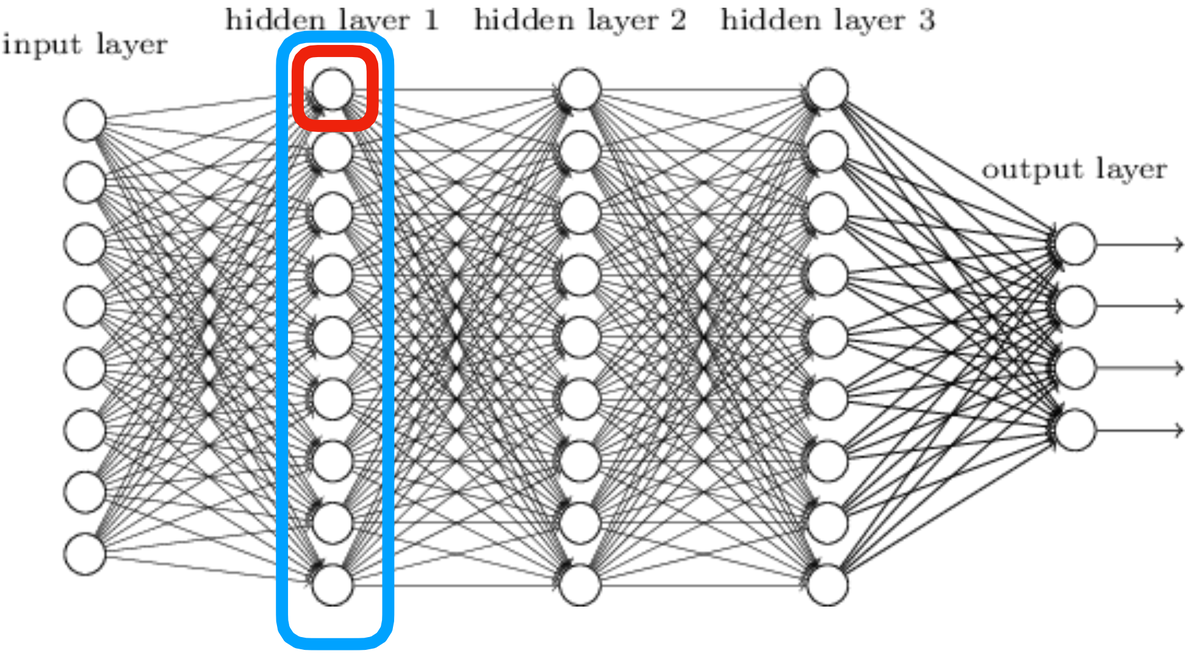
https://www.jessicayung.com/explaining-tensorflow-code-for-a-multilayer-perceptron/

- Step function: also called threshold function
  - Decides based on the weighted sum if the output is 0 or 1
- In modern deep learning, the step function is less often used as it is not differentiable at the threshold point
- Instead, smoother functions that approximate the step function but retain differentiability are used
- For example: **sigmoid** outputs a continuous value in the **range** (0, 1)

A "layer" is a matrix of numbers



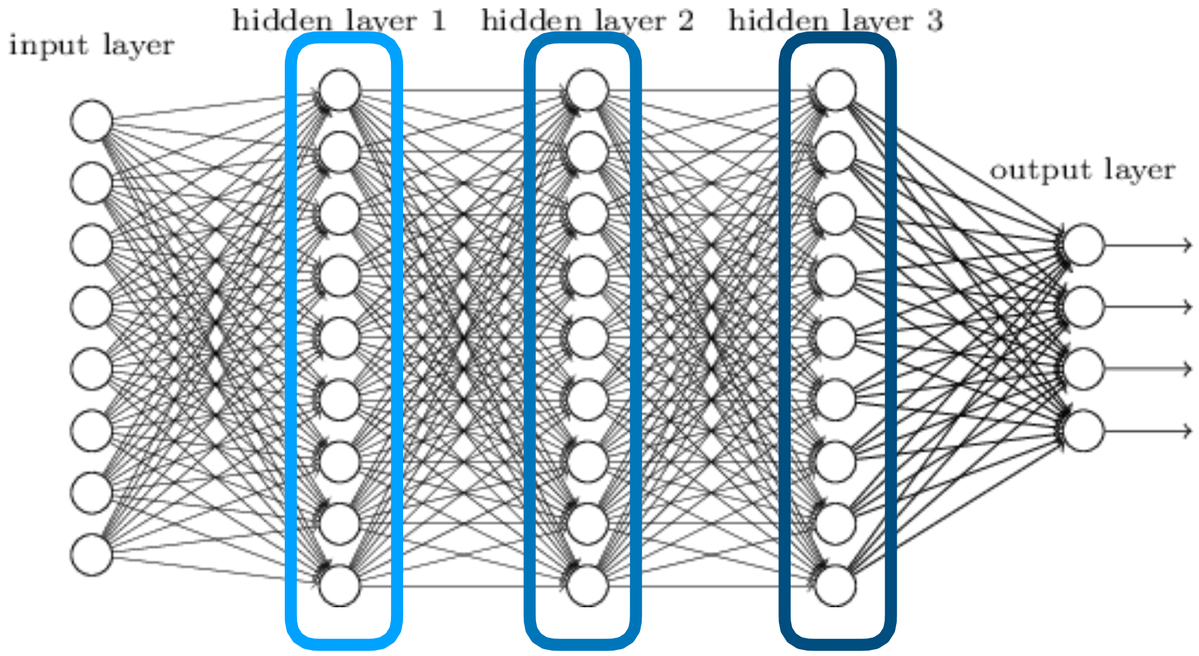
https://www.jessicayung.com/explaining-tensorflow-code-for-a-multilayer-perceptron/




```
[[0.58  0.841 0.835 0.18 ]
 [0.405 0.813 0.309 0.562]
 [0.422 0.229 0.46  0.152]
 [0.673 0.429 0.441 0.243]
 [0.9   0.744 0.234 0.856]
 [0.971 0.486 0.175 0.248]
 [0.258 0.588 0.478 0.266]
 [0.236 0.496 0.077 0.557]
 [0.413 0.322 0.372 0.741]]
```

The neural network is a set of matrices

Called "parameters" or "weights"

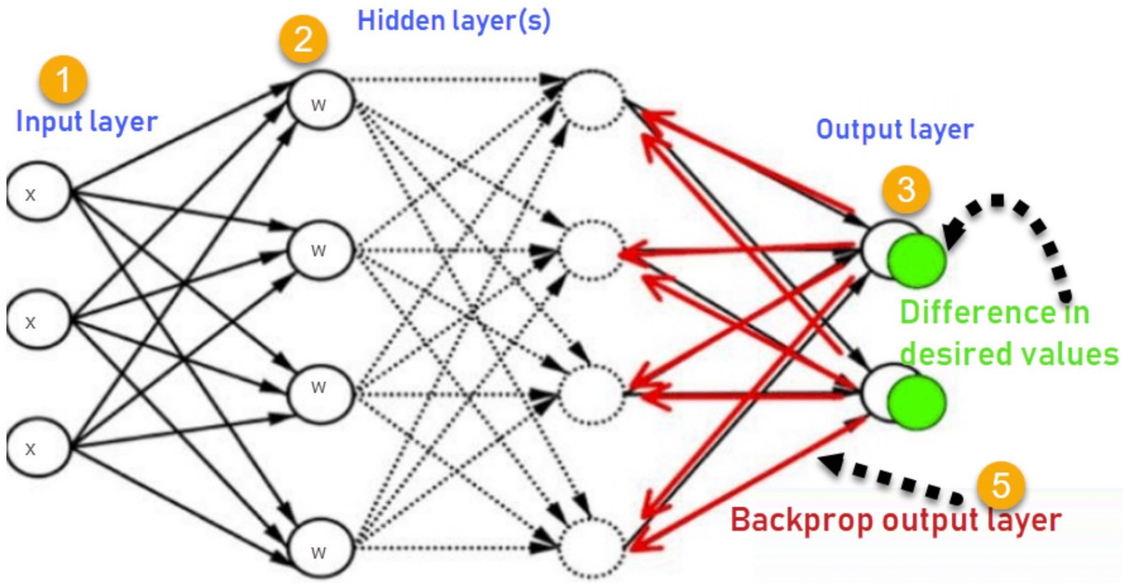



```
[[0.32  0.866 0.151 0.555]
 [[0.58  0.841 0.835 0.18 ]
  [[0.32  0.866 0.151 0.555]
   [0.467 0.414 0.562 0.555]
   [0.186 0.937 0.131 0.448]
   [0.892 0.567 0.111 0.678]
   [0.483 0.478 0.424 0.844]
   [0.774 0.963 0.213 0.569]
   [0.014 0.509 0.21  0.26 ]
   [0.142 0.991 0.105 0.211]
   [0.789 0.242 0.774 0.019]]
```

NN operations are just matrix multiplications.
GPUs are really fast at matrix multiplications.

Training

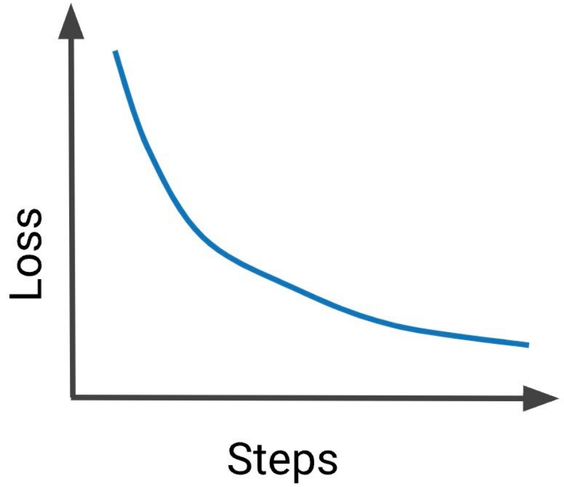
- **Dataset** contains n items
- **Batch**: a subset of n to use for training
  - If n is large, it won't fit in memory
- **Epoch**: going over the entire n items in the dataset



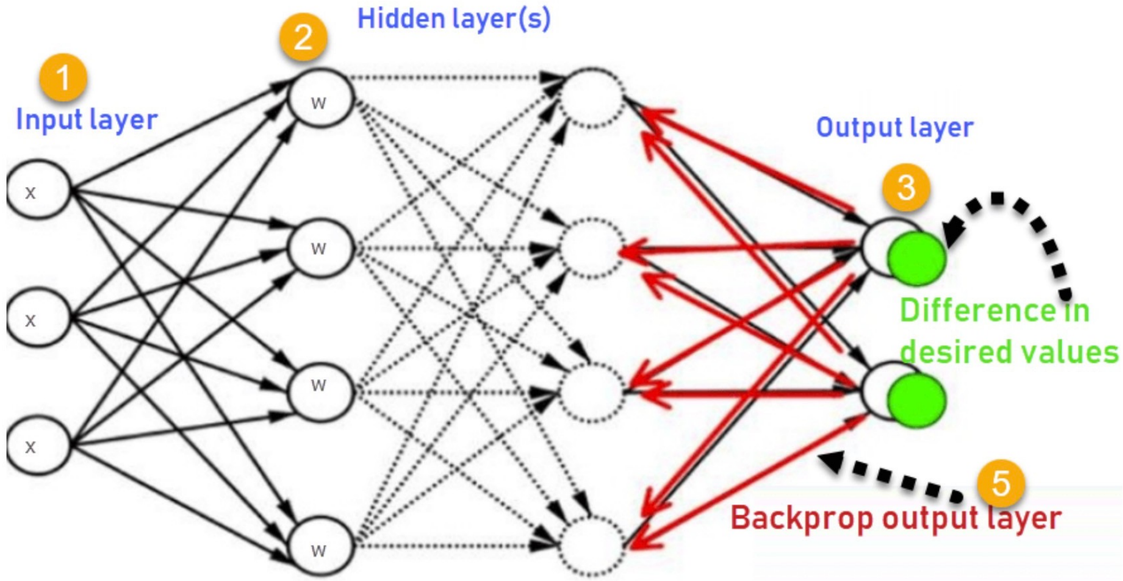
https://www.guru99.com/backpropogation-neural-network.html

Training

$$loss_{CE} = -\sum(y_i * \log(y_i'))$$



- Data X (e.g. images), labels y (e.g. labels)
- Take a little batch of data x:
  - Use the current model to make a prediction x -> y'
  - Compute loss(y, y')
  - Back-propagate the loss through all the layers of the model
- Repeat until loss stops decreasing

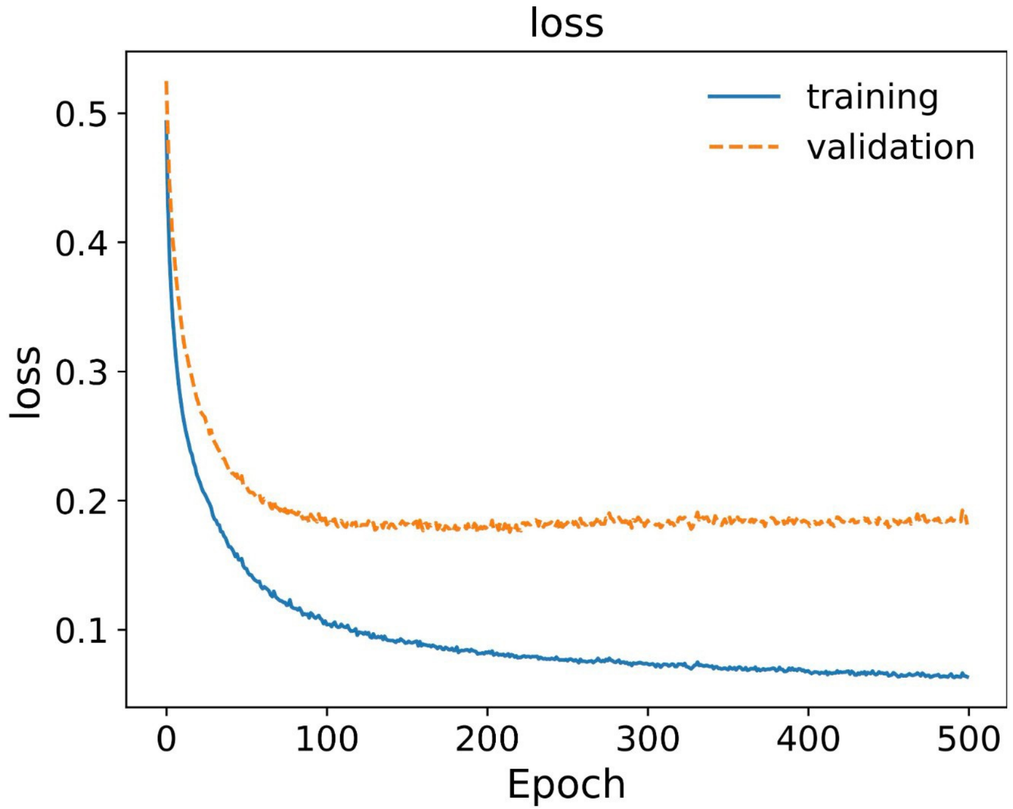


https://www.guru99.com/backpropogation-neural-network.html

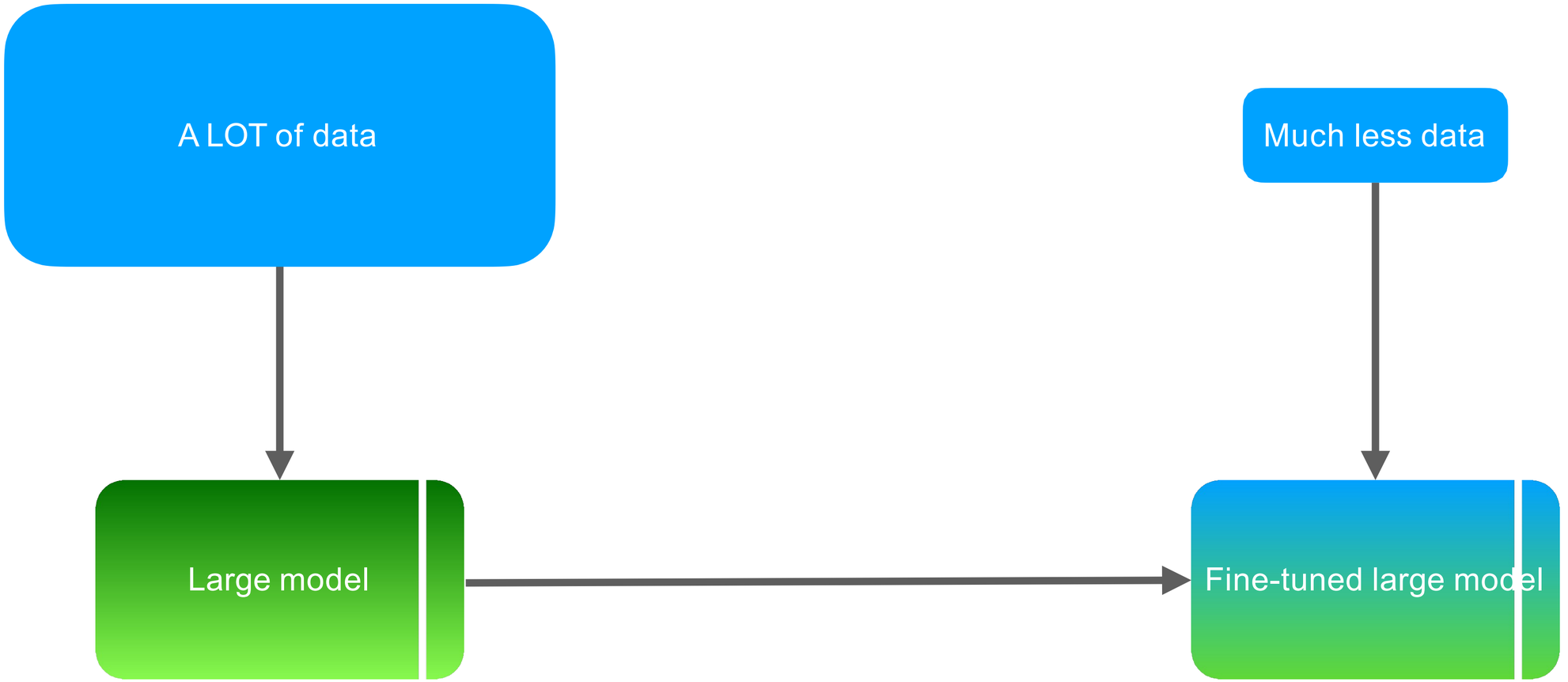
https://developers.google.com/machine-learning/testing-debugging/metrics/interpretic

Dataset Splitting

- Split (X, y) into training (~80%), validation (~10%), and test (~10%) sets
- Validation set is for
  - ensuring that training is not "overfitting"
  - setting hyper-parameters of the model (e.g. number of parameters)
- Test set is for measuring validity of predictions on new data



THIS APPLIES TO YOUR EXPERIMENTATION WITH PROMPTS!

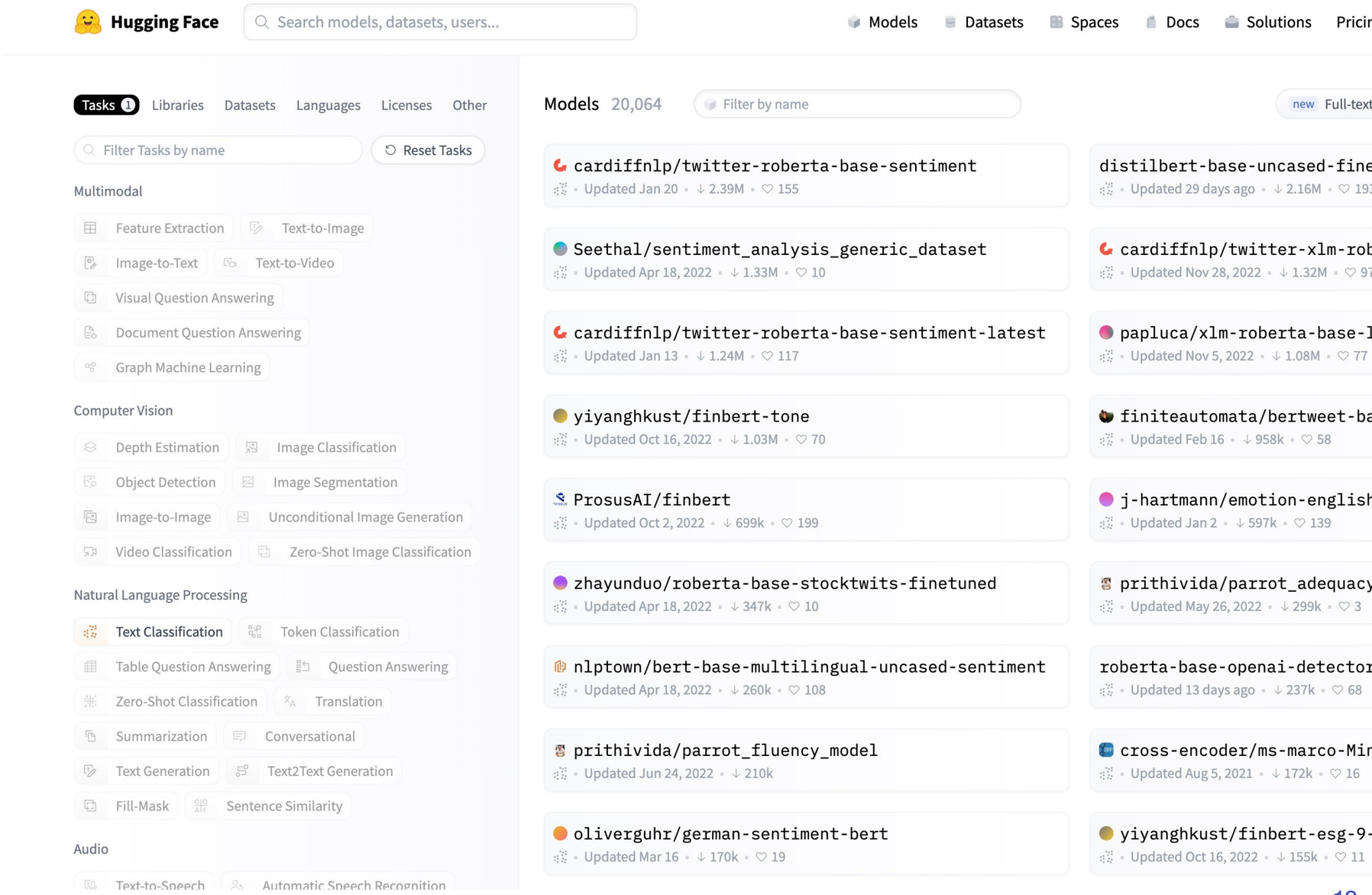


Pre-training:
slow training on a lot of data

Fine-tuning:
fast training on a little data

Model Hubs

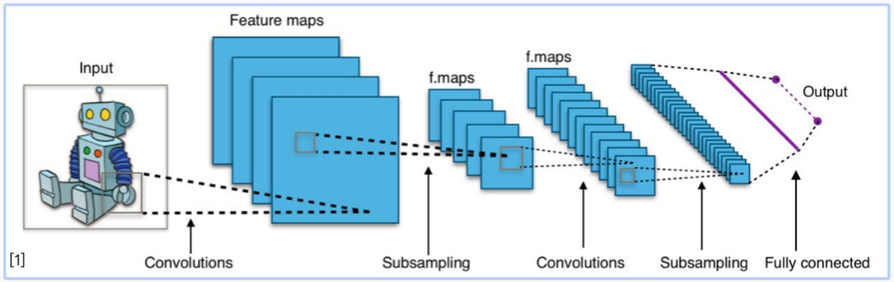
- People share pre-trained models!
  - 180K models
  - 30K datasets



Before ~2020: each task had its own NN architecture

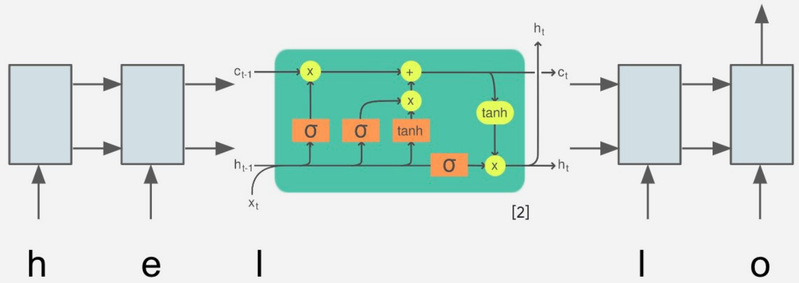
Computer Vision

Convolutional NNs (+ResNets)



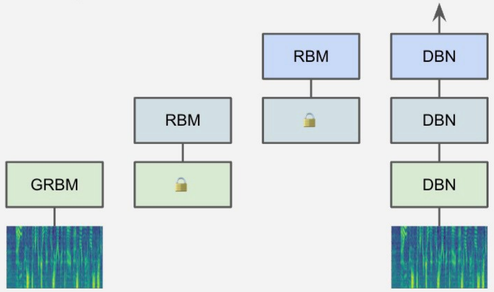
Natural Lang. Proc.

Recurrent NNs (+LSTMs)



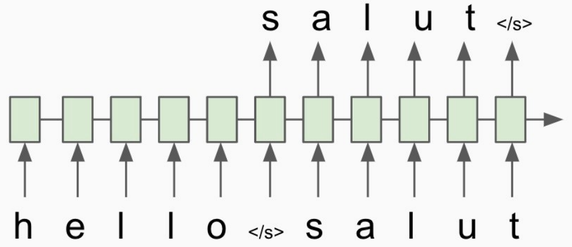
Speech

Deep Belief Nets (+non-DL)



Translation

Seq2Seq



RL

BC/GAIL

Algorithm 1 Generative adversarial imitation learning

Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
for $i = 0, 1, 2, \ldots$ **do**
  Sample trajectories $\tau_i \sim \pi_{\theta_i}$
  Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \quad (17)$$

  Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta), \quad \text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \,|\, s_0 = \bar{s}, a_0 = \bar{a}] \quad (18)$$

end for

[1] CNN image CC-BY-SA by Aphex34 for Wikipedia https://commons.wikimedia.org/wiki/File:Typical_cnn.png
[2] RNN image CC-BY-SA by GChe for Wikipedia https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg

http://lucasb.eyer.be/transformer

Now: all is Transformers

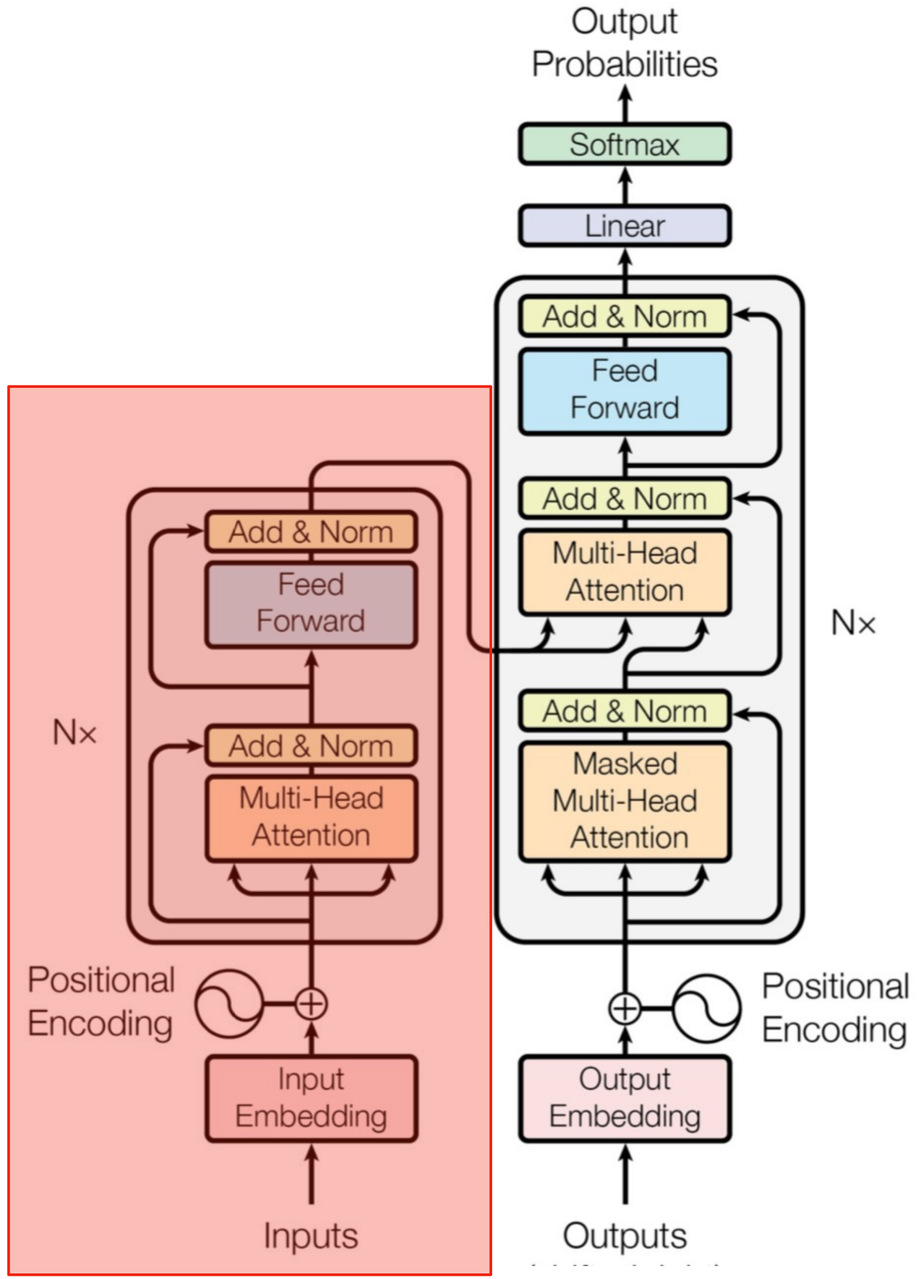
Transformer cartoon (DALL-E)

The Transformer Architecture

Attention is all you need (2017)

https://arxiv.org/abs/1706.03762

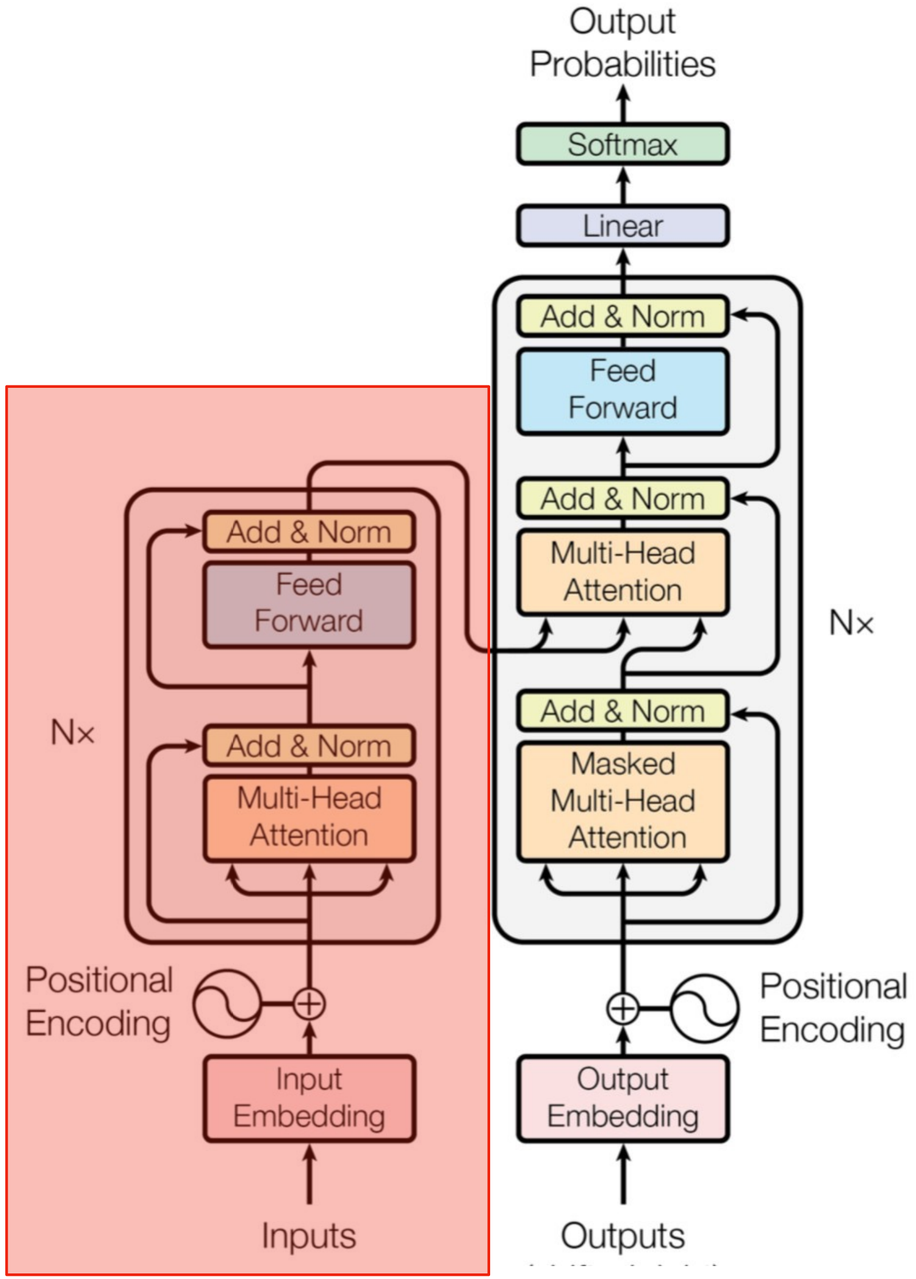
- Ground-breaking architecture that set SOTA on first translation and later all other NLP tasks
- For simplicity, can just look at one half of it



Transformer

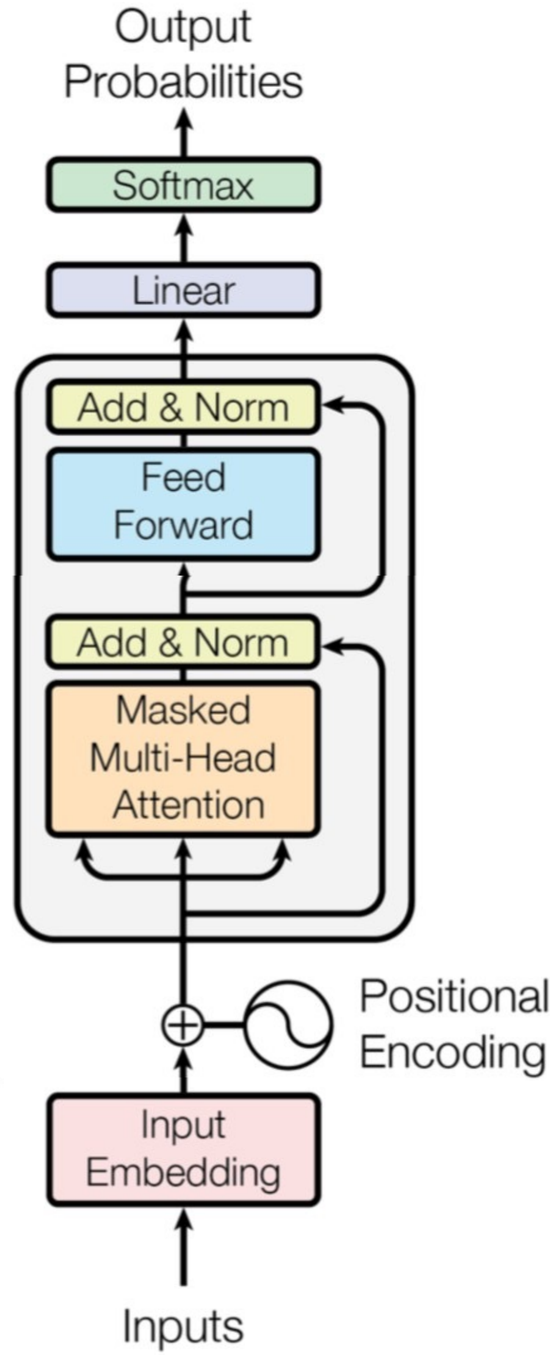
A breakthrough in AI that changes how we process data (i.e., text).

The core idea: Instead of reading word by word, transformers focus on the whole sentence and **important** parts of a sentence all at once.



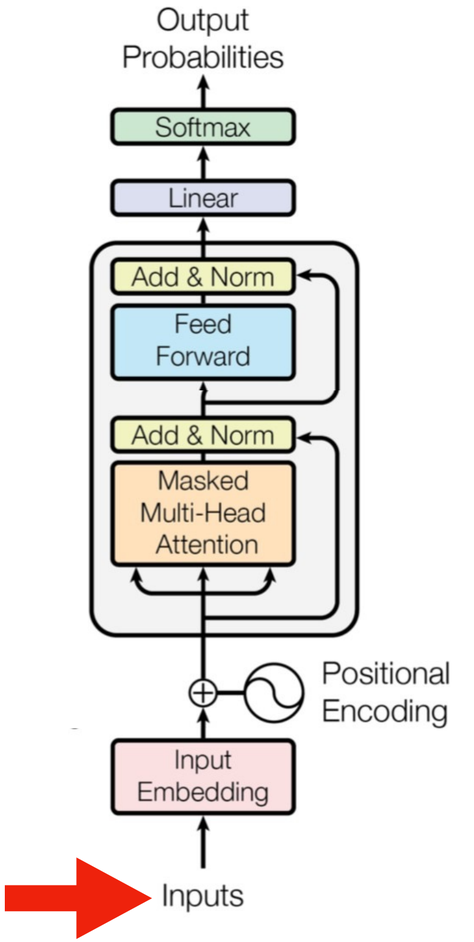
Transformer Decoder Overview

- Task is to complete text
  - "It's a blue" -> "sundress"
- Inputs: a sequence of N tokens
  - [It's, a, blue]
- Output:
  - Probability distribution over the next token
- Inference:
  - Sample the next token from the distribution, append it to inputs, run through the model again, sample, append, etc.



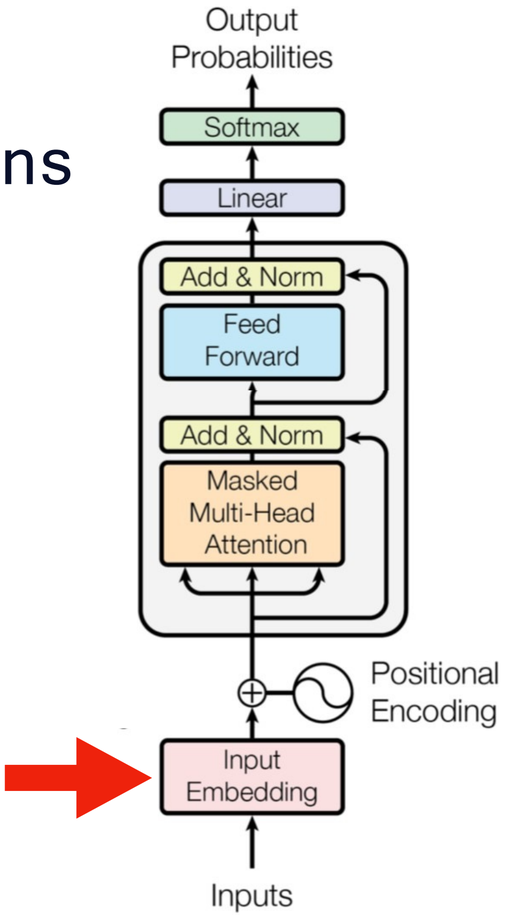
Vectorizing Inputs

- Inputs need to be vectors of numbers
    - Start with original text:
        - "It's a blue sundress."
- Turn into a sequence of tokens:
    - [<SOS>, It, 's, a, blue, sund, ress, ., <EOS>]
- Turn into vocabulary IDs:
    - [0, 1026, 338, 257, 4171, 37437, 601, 13, 1]
- Each ID can be represented by a one-hot vector
    - e.g. 13 -> [0, 0, 0, 1, 0, 0, 0, 0, ...] (all zeros, but one)



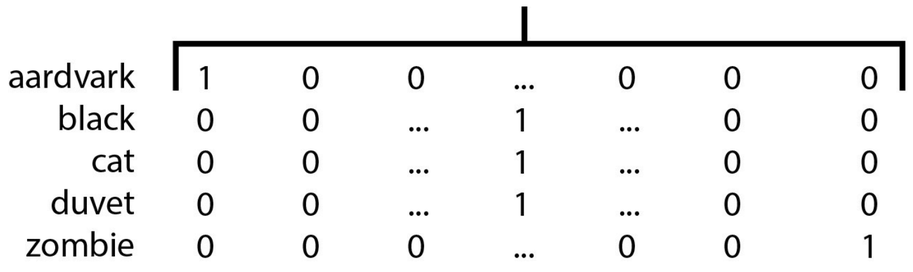
One-hot vectors

- One-hot vectors are poor representations of words or tokens
  - e.g. distance between "cat" and "kitten" is the same as between "cat" and "tractor"
- Cat: [1, 0, 0]
- Kitten: [0, 1, 0]
- Tractor: [0, 0, 1]



Input Embedding

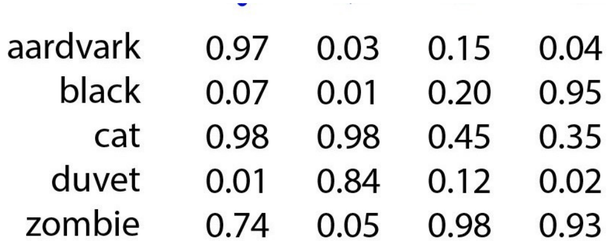
- Solution: **learn** an embedding matrix!
  - An embedding maps each word to a high-dimensional vector of continuous numbers
  - learned through training a model on a specific task
    - e.g., Word2Vec: learns embeddings on large corpora of text



aardvark	1	0	0	…	0	0	0
black	0	0	…	1	…	0	0
cat	0	0	…	1	…	0	0
duvet	0	0	…	1	…	0	0
zombie	0	0	0	…	0	0	1

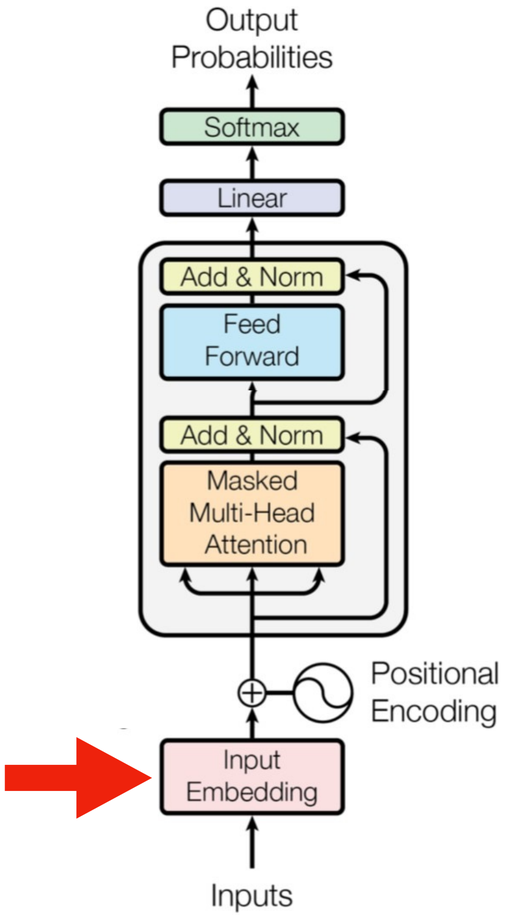
VxV

VxE
embedding matrix



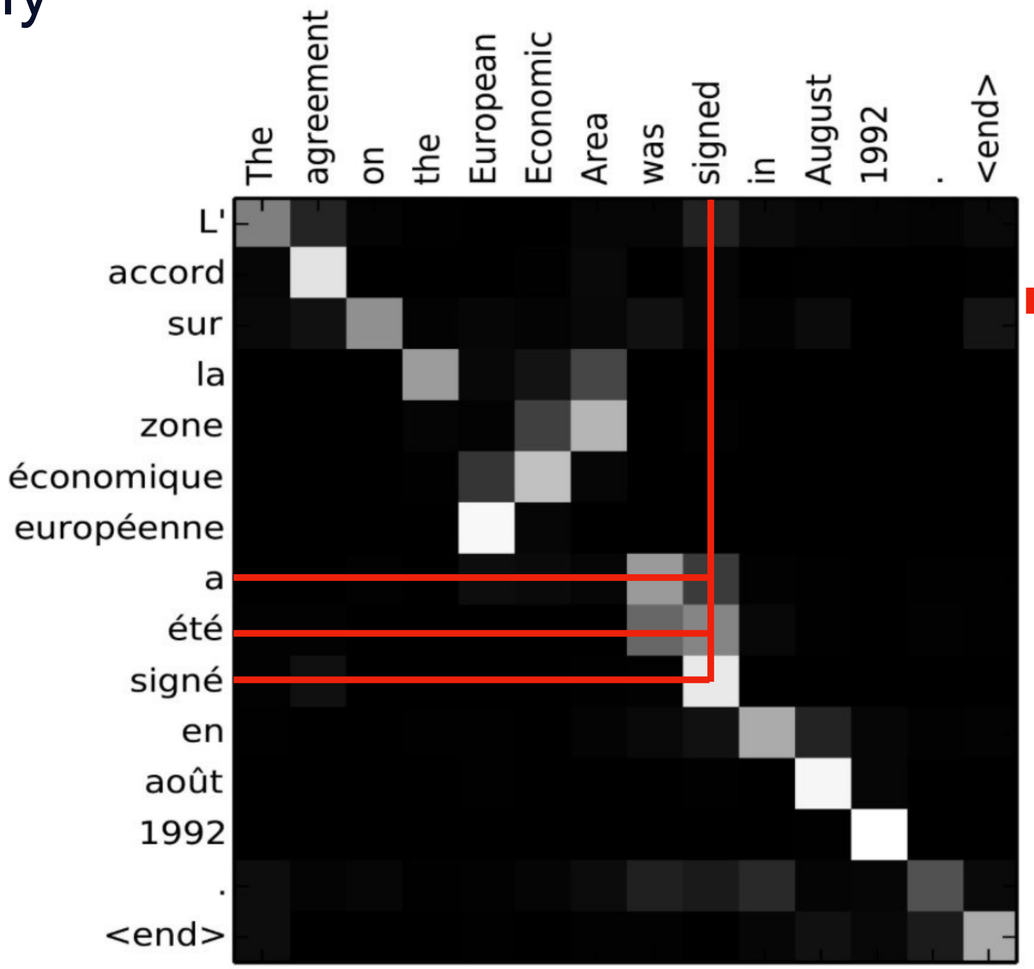
aardvark	0.97	0.03	0.15	0.04
black	0.07	0.01	0.20	0.95
cat	0.98	0.98	0.45	0.35
duvet	0.01	0.84	0.12	0.02
zombie	0.74	0.05	0.98	0.93

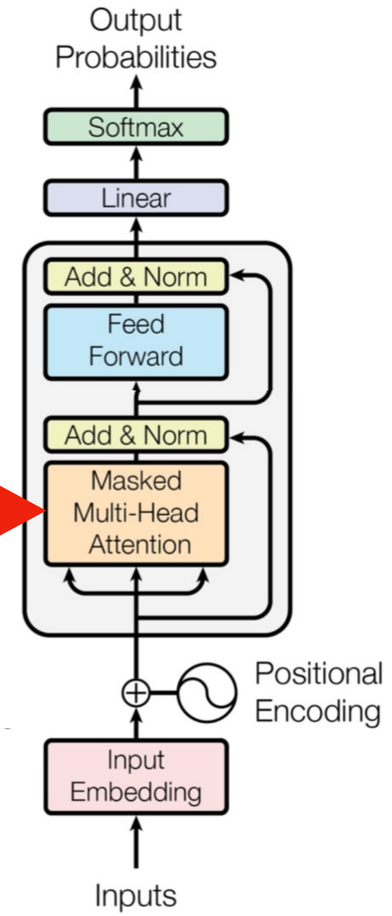
VxE



Attention

- Key insight: for a given token in the output sequence, only one or a few tokens in the input sequence are **most important**
- **Find those important relations**
- Introduced in 2015 for translation tasks





https://lilianweng.github.io/posts/2018-06-24-attention/

Basic self-attention

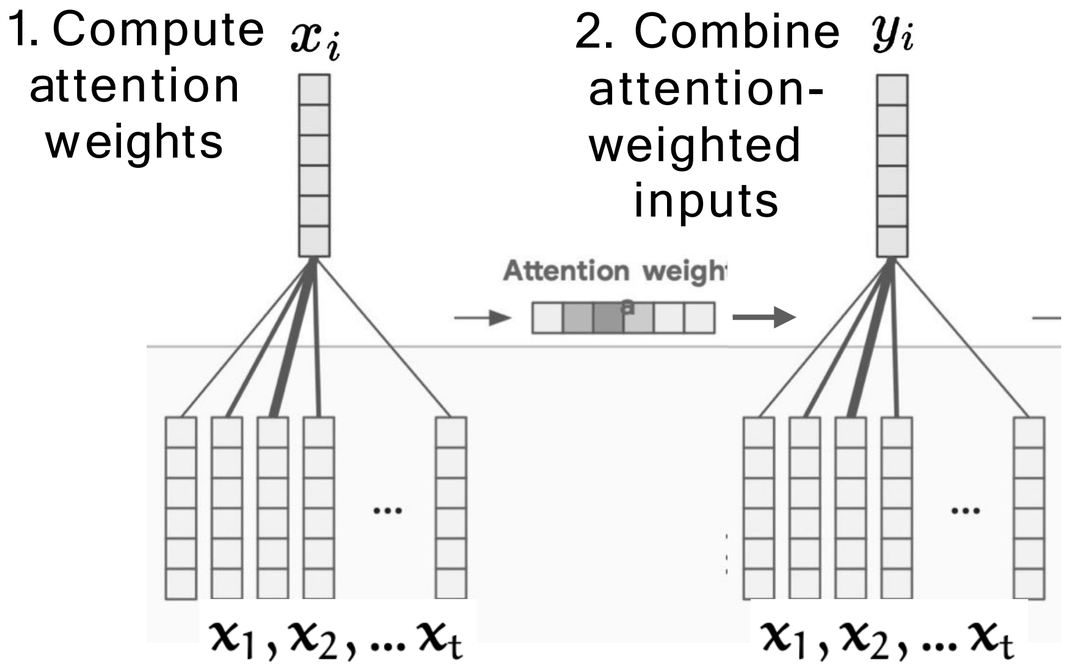
- Input: sequence of vectors

$$\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_t$$

- Output: sequence of vectors, each one a weighted sum of the input sequence

$$\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_t$$

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{x}_j$$



- weight is just dot product between input vectors

$$w'_{ij} = \mathbf{x}_i{}^T \mathbf{x}_j$$

- (made to sum to 1)

$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$$

http://www.peterbloem.nl/blog/transformers

Basic self-attention

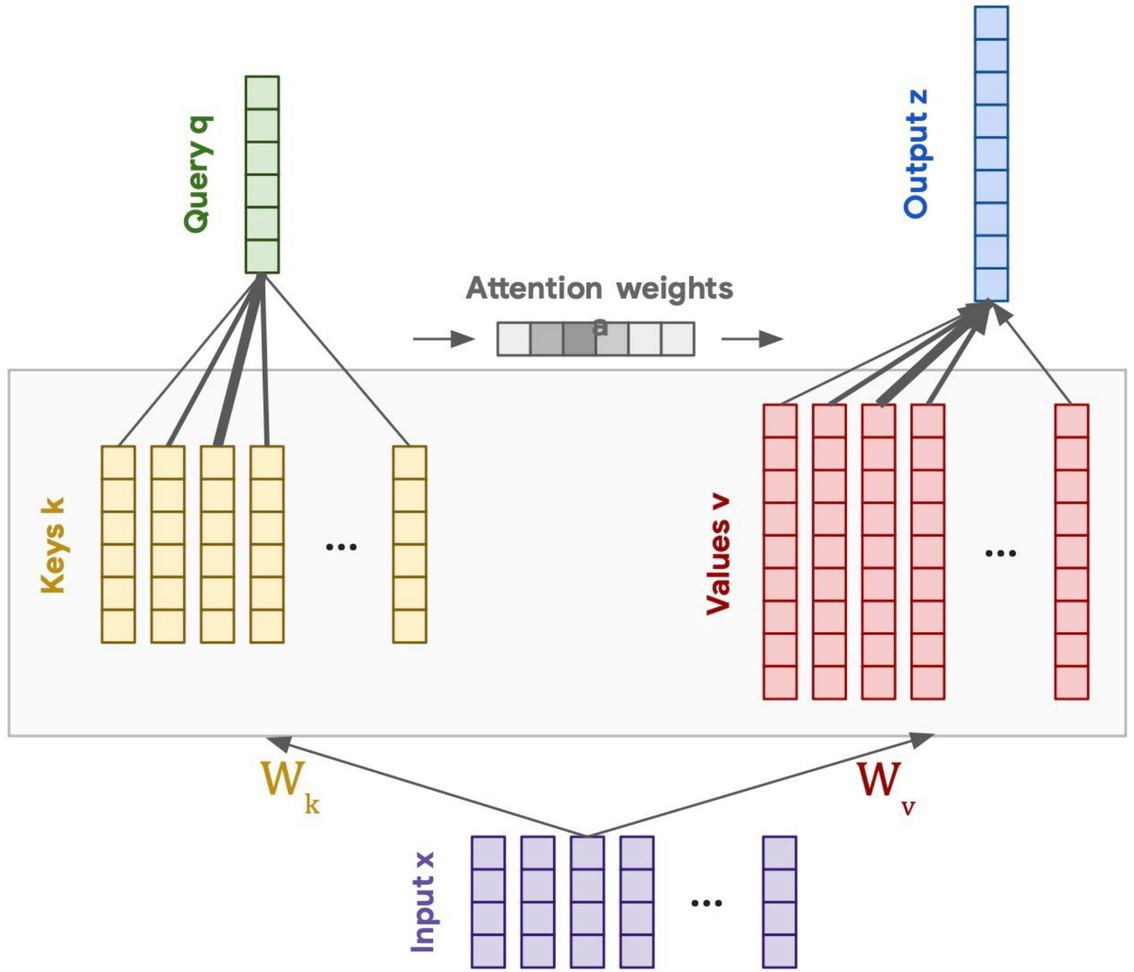
- Problem: there's no learning involved!
- Solution: project inputs into query, key, value roles
- Learning these matrices = learning attention

$$\mathbf{q}_i = \boldsymbol{W}_q \boldsymbol{x}_i \qquad \mathbf{k}_i = \boldsymbol{W}_k \boldsymbol{x}_i \qquad \boldsymbol{v}_i = \boldsymbol{W}_v \boldsymbol{x}_i$$

$$w'_{ij} = \mathbf{q}_i{}^{\mathsf{T}} \mathbf{k}_j$$

$$w_{ij} = \mathrm{softmax}(w'_{ij})$$

$$\mathbf{y}_i = \sum_j w_{ij} \boldsymbol{v}_j \,.$$

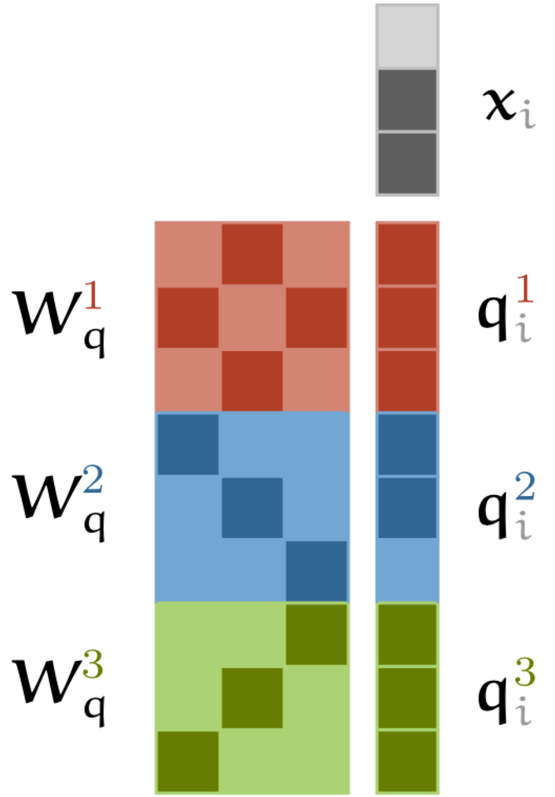


http://lucasb.eyer.be/transformer

http://www.peterbloem.nl/blog/transformers

Multi-head attention

- Different parts of the sentence could have different relations
- We can allow different ways of transforming into queries, keys, and values to be learned
- Simply means learning different sets of W_q, W_k, and W_v matrices simultaneously.



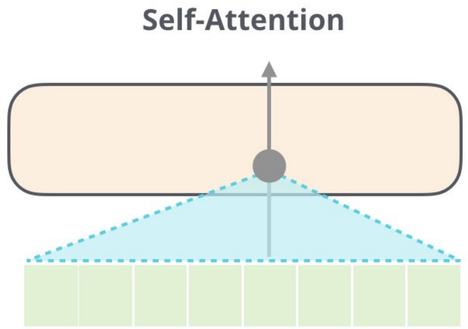
3-headed attention

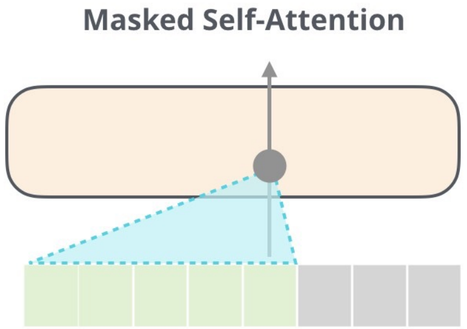
http://www.peterbloem.nl/blog/transformers

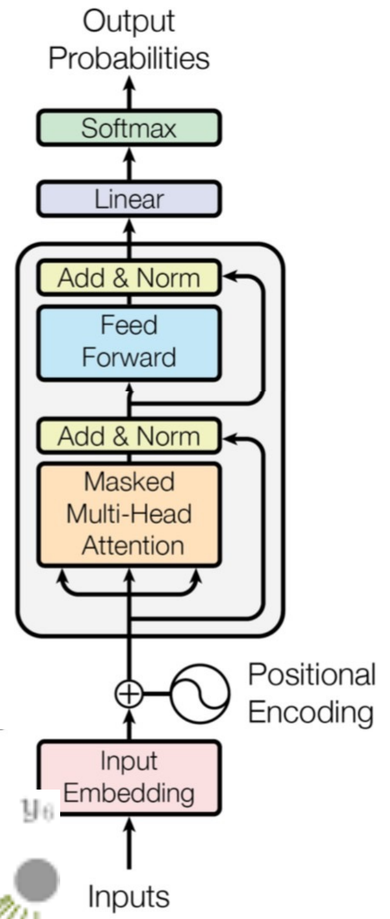
Masking attention

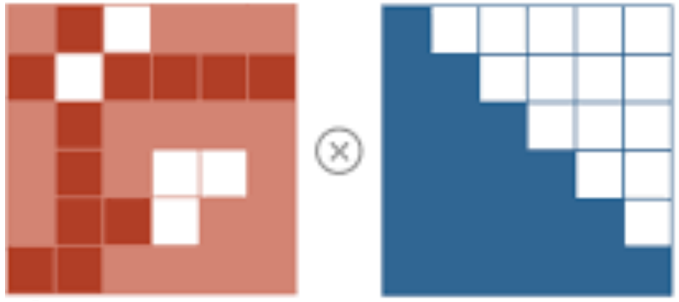
In training:

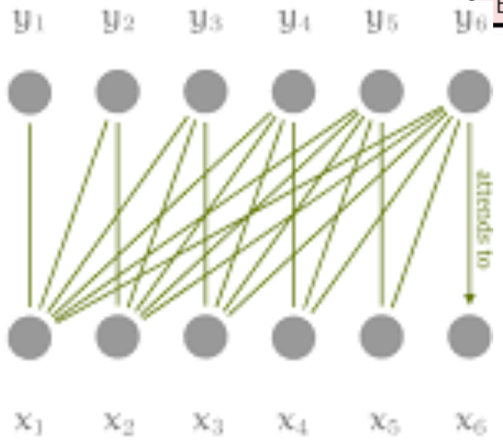
- Inputs: a sequence of N tokens
  - [It's, a, blue, <BLANK>, <BLANK>, .
- Ground-truth Outputs: a sequence of N tokens
  - [a, blue, sundress, <BLANK>, <BLANK>, ...]
- Actual Outputs: Instead of words, vectors of probabilities over the vocabulary
  - **Crucially: all output probabilities are computed at the same time!**







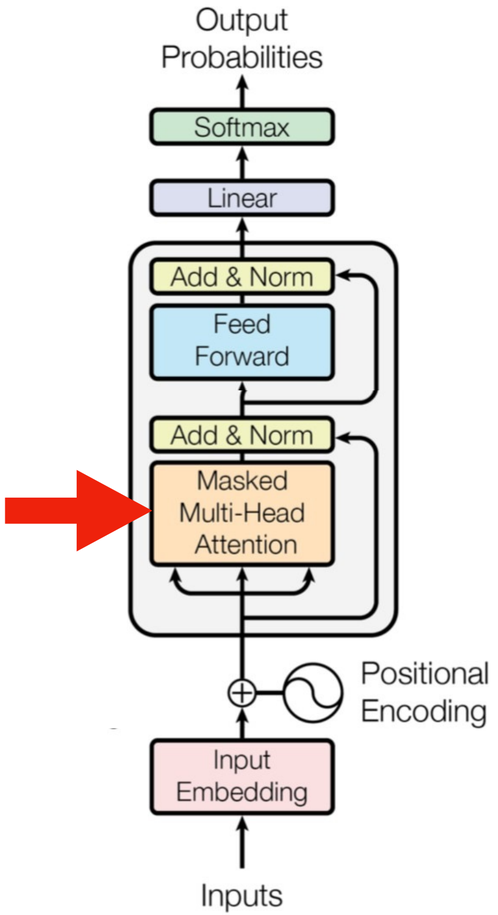




Note how you shouldn't see future tokens when predicting

Masked Multi-Head Attention

- Conceptual view:
  - token comes in
  - gets "augmented" with previously-seen tokens that seem relevant (masked self-attention)
  - this happens in several different ways simultaneously (multiple heads)
- NOTE: there's no notion of "position" so far!



Positional Encoding

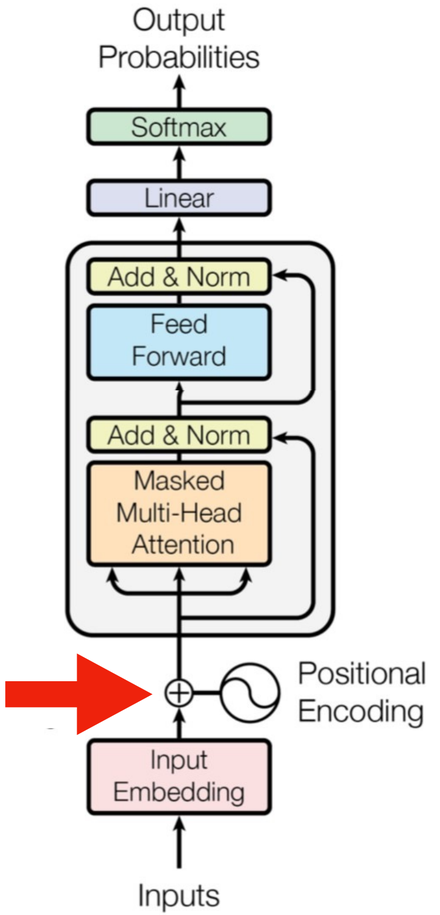
$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i$$

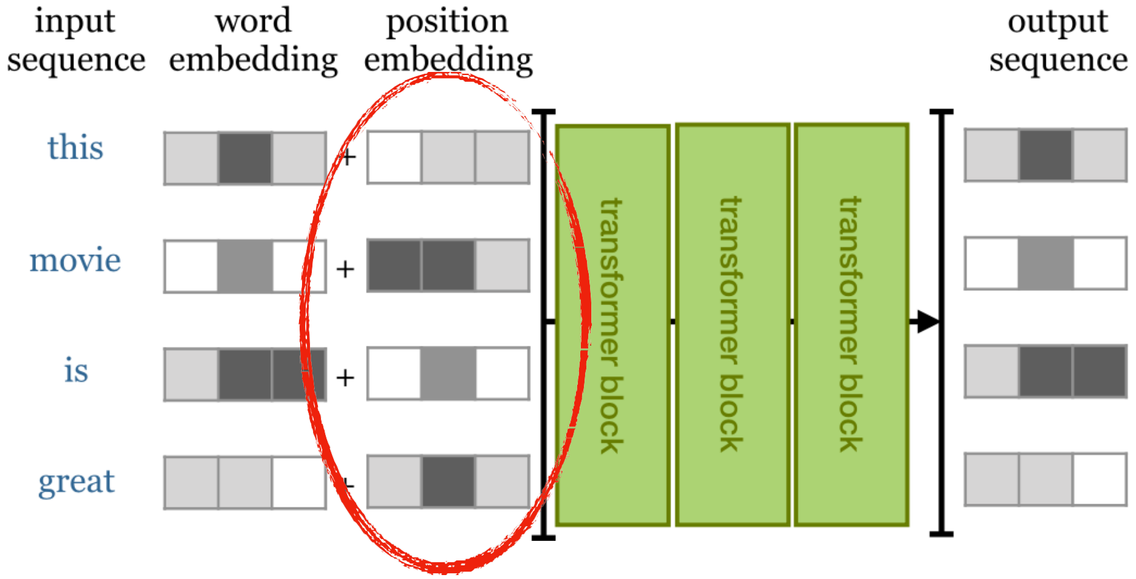
$$w'_{ij} = \mathbf{q}_i{}^{\mathsf{T}} \mathbf{k}_j$$

$$w_{ij} = \mathrm{softmax}(w'_{ij})$$

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{v}_j \,.$$

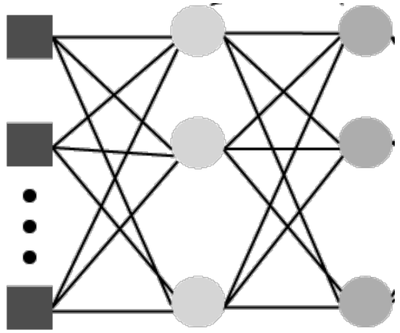
- Attention is totally position-invariant!
  - e.g. [this, movie, is, great] is the same as [movie, this, great, is]
- So, let's add position-encoding vectors to embedding vectors
  - It really is that simple



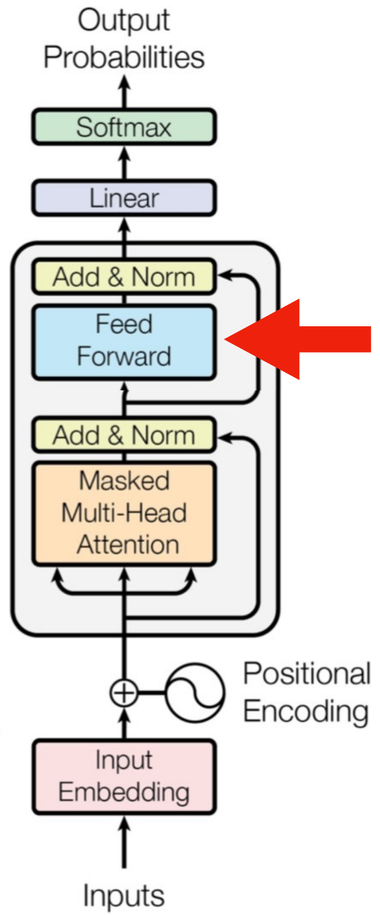


Feed Forward Layer

- Standard Multi-Layer Perceptron with one hidden layer

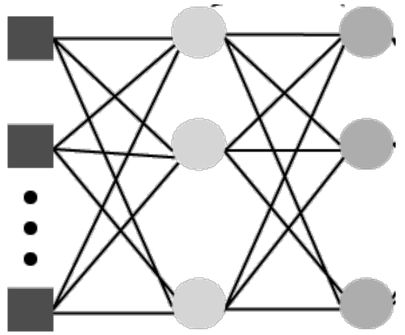


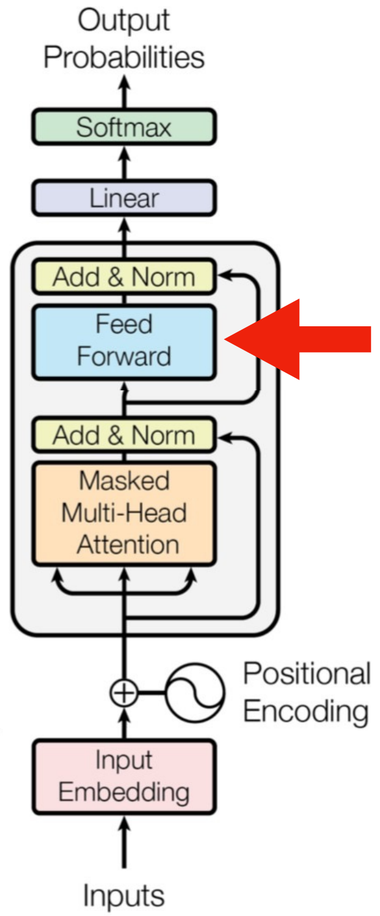
- Conceptual view:
  - token (augmented with other relevant tokens that it has seen) comes in...
  - ...and "upgrades" its representation



Feed Forward Layer

- Standard Multi-Layer Perceptron with one hidden layer



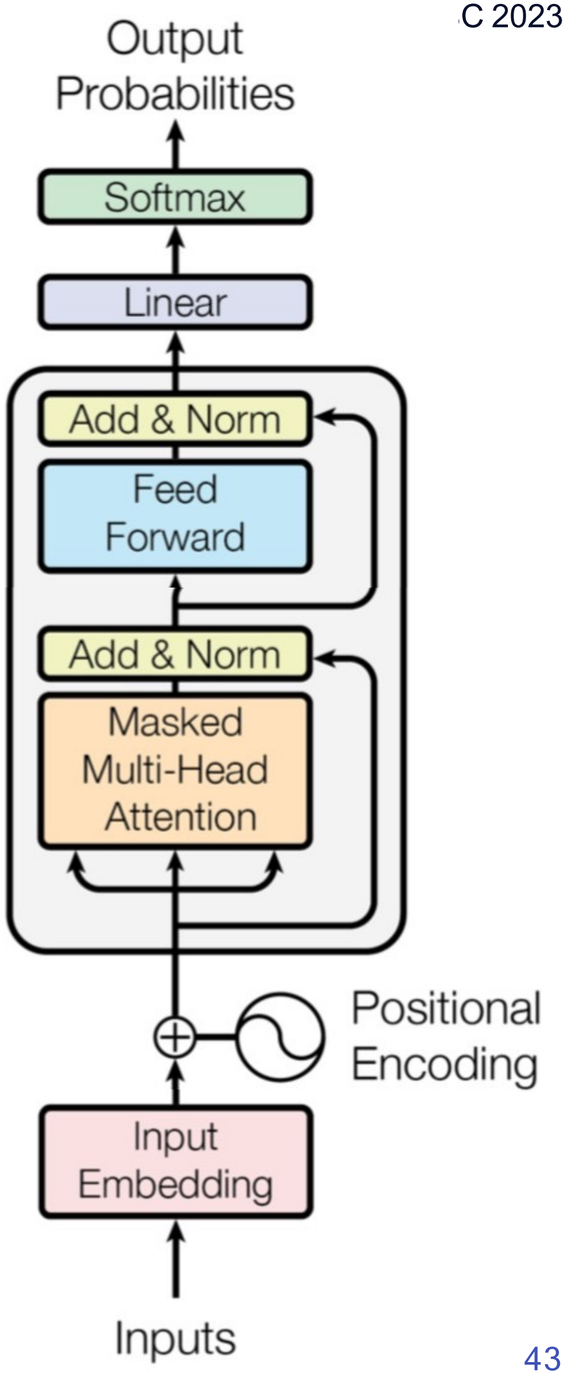


After the attention mechanism aggregates and weighs information across the input sequence, the feed-forward network applies a series of linear transformations to this aggregated information. This process allows the model to learn complex relationships and patterns beyond what the attention mechanism can do on its own.

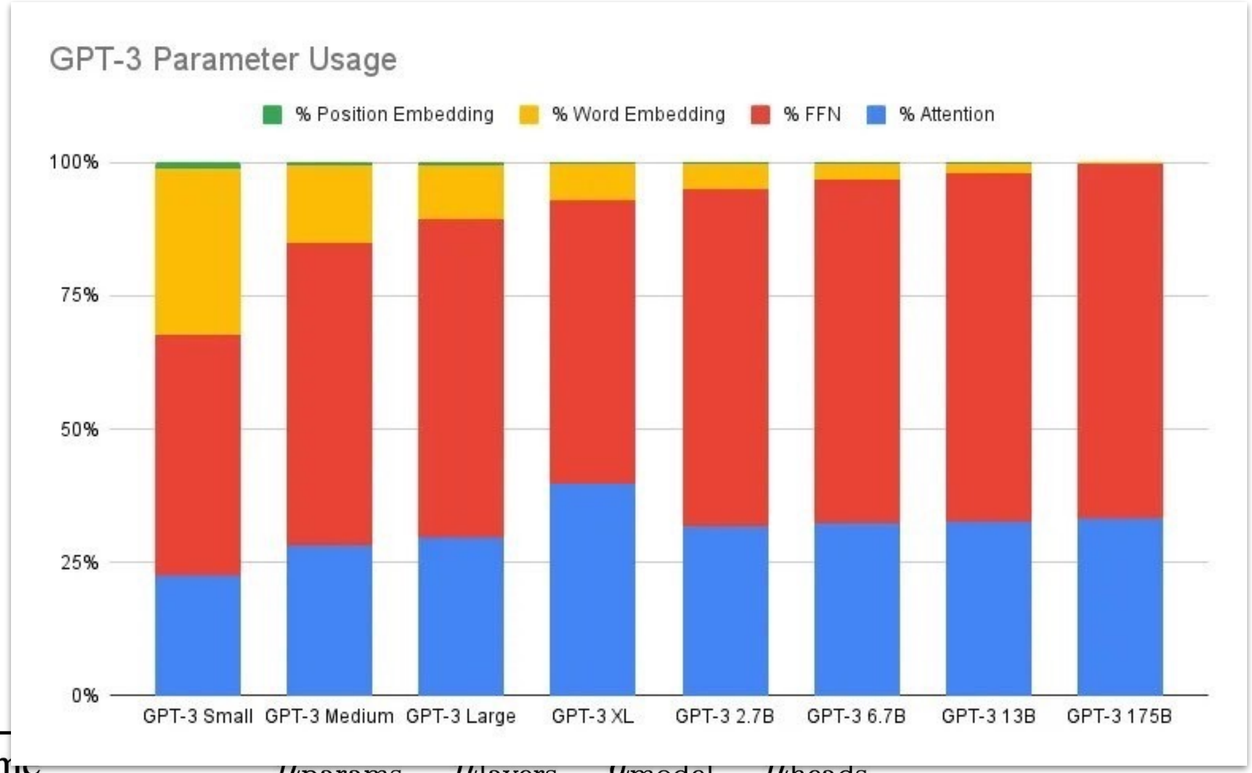
Transformer Architecture

- The main Transformer Layer is stacked many times
- The overall hyperparameters are:
  - Number of layers
  - Embedding dimension
  - Number of attention heads
- The largest models are ~70% feed-forward weights

https://aizi.substack.com/p/how-does-gpt-3-spend-its-175b-parameters

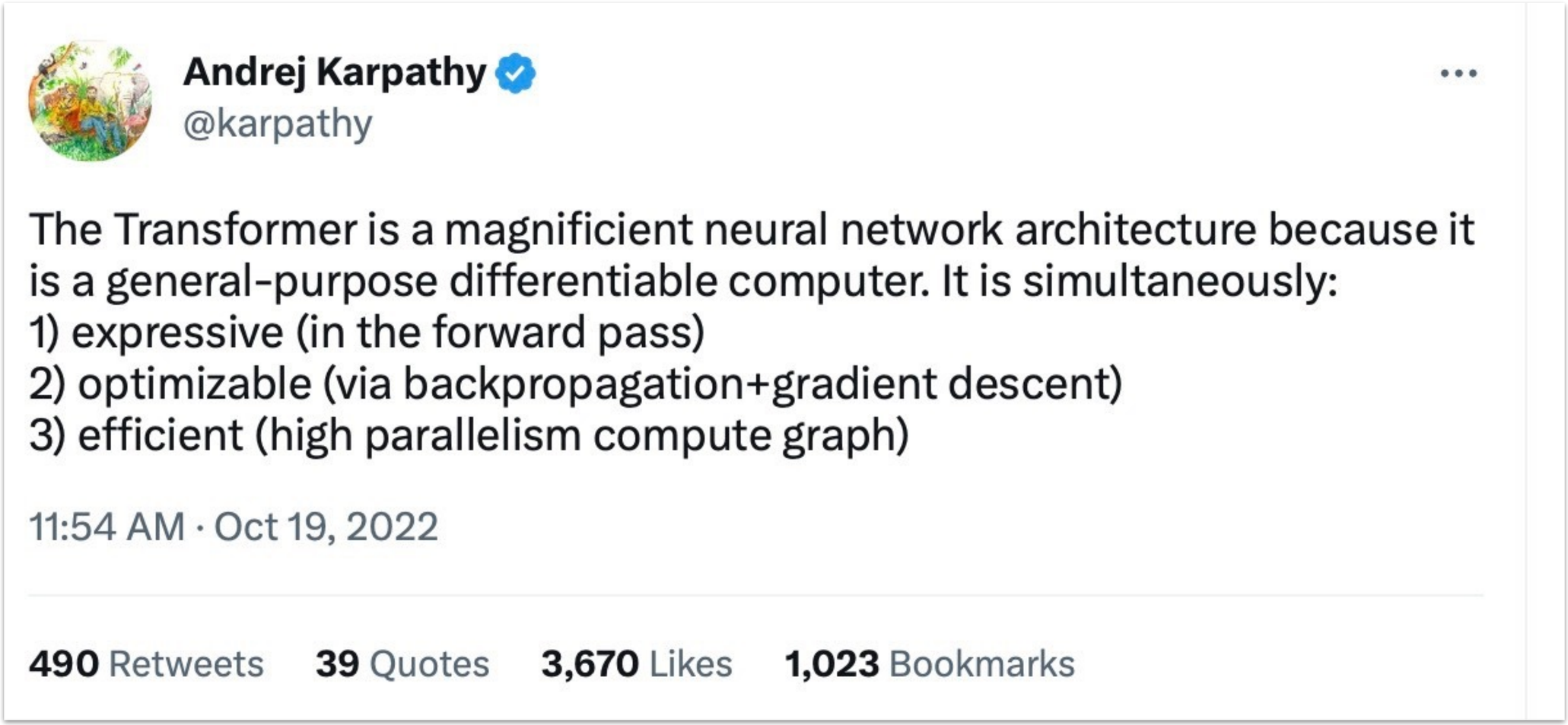


Transformer Architecture



Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}
GPT-3 Small	125M	12	768	12
GPT-3 Medium	350M	24	1024	16
GPT-3 Large	760M	24	1536	16
GPT-3 XL	1.3B	24	2048	24
GPT-3 2.7B	2.7B	32	2560	32
GPT-3 6.7B	6.7B	32	4096	32
GPT-3 13B	13.0B	40	5140	40
GPT-3 175B or “GPT-3”	175.0B	96	12288	96

Why does this work so well?



We mostly don't understand it, though

In-context Learning and Induction Heads

AUTHORS

Catherine Olsson*, Nelson Elhage*, Neel Nanda*, Nicholas Joseph†, Nova DasSarma†, Tom Henighan†, Ben Mann†, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, Chris Olah‡

AFFILIATION

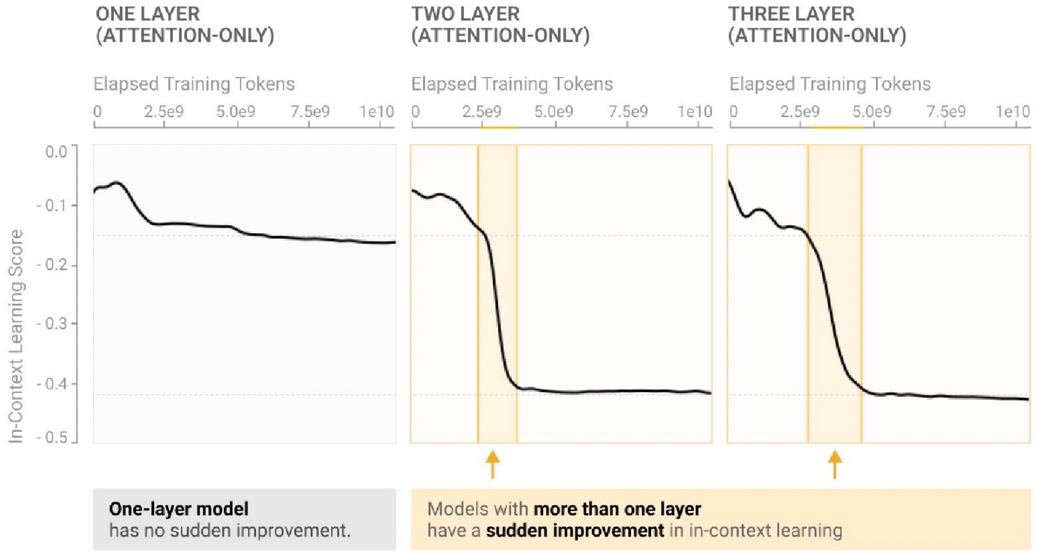
Anthropic

PUBLISHED

Mar 8, 2022

* Core Research Contributor; † Core Infrastructure Contributor; ‡ Correspondence to colah@anthropic.com; Author contributions statement below.

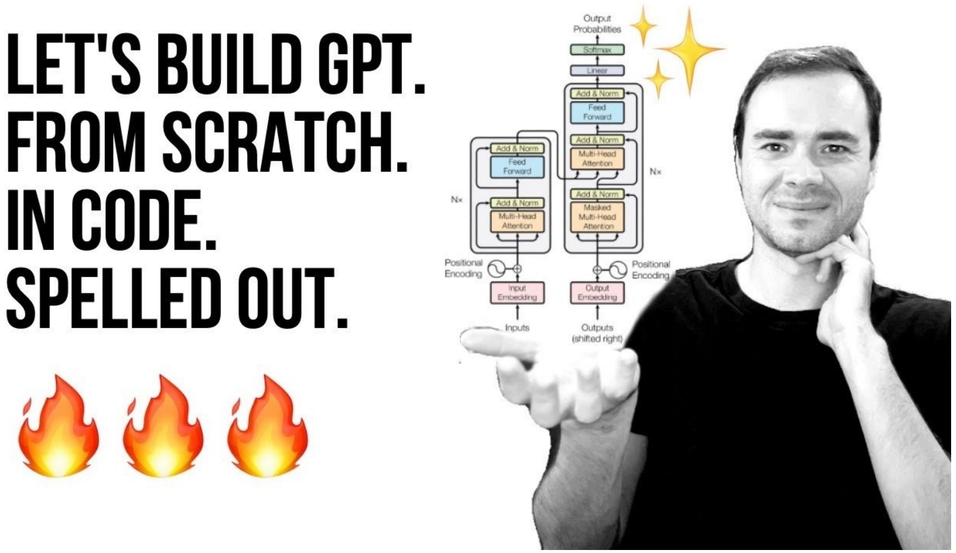
MODELS WITH MORE THAN ONE LAYER HAVE AN ABRUPT IMPROVEMENT IN IN-CONTEXT LEARNING



We highlight
"phase chang
period of training in
make visual compar
between plots easie
highlighted region is
selected for each m
based on the derivat
in-context learning.

Should you be able to code a Transformer?

- Definitely not necessary!
- BUT: it's not difficult, it is fun, and is probably worth doing
- Andrej Karpathy's GPT-2 implementation is <400 lines of code, including Attention



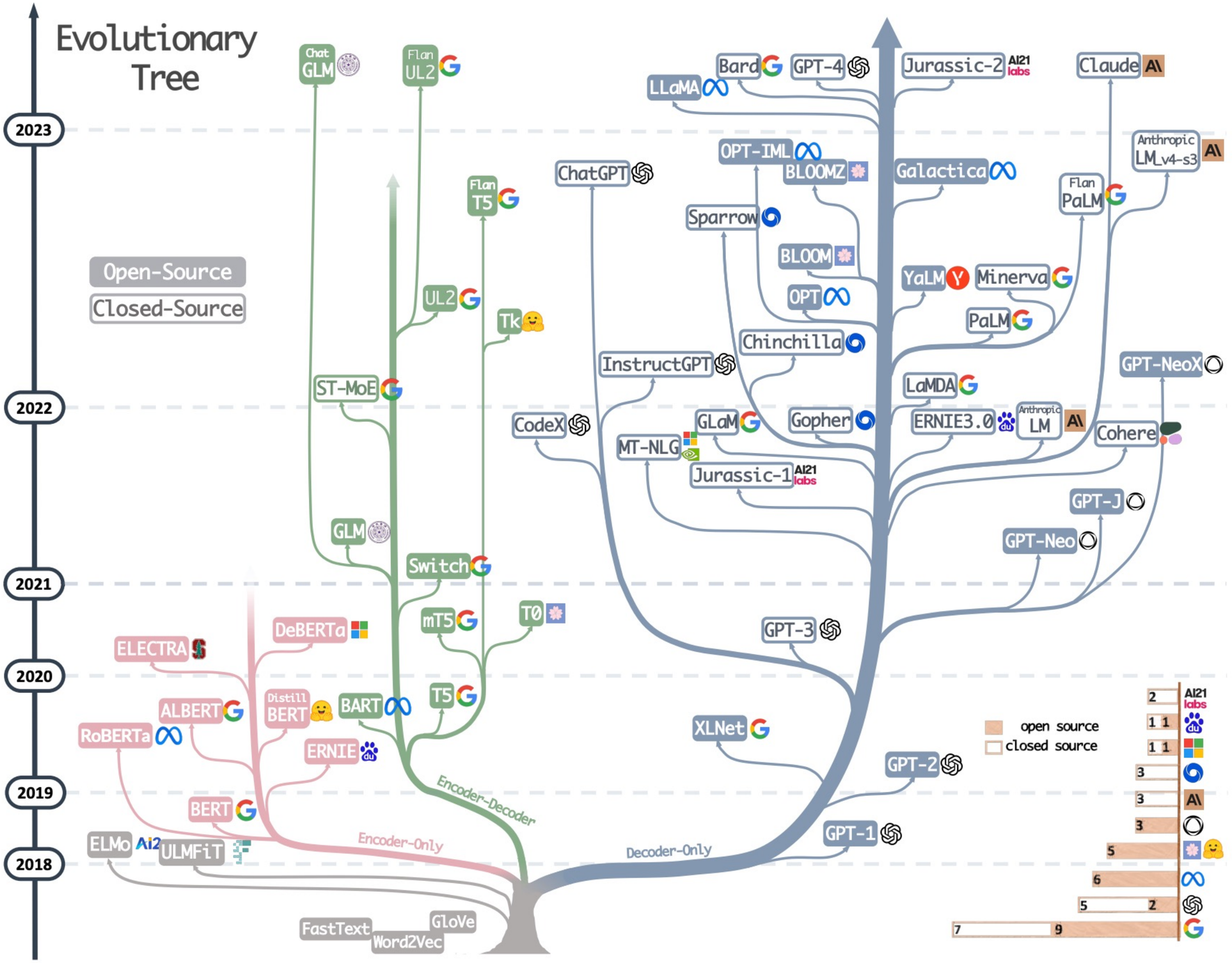
https://www.youtube.com/watch?v=kCc8FmEb1nY

Resources

- Lucas Beyer's Lecture on Transformers
- Peter Bloem's "Transformers from Scratch"
- Nelson Elhage's "Transformers for Software Engineers" for a different view
- Andrej Karpathy's entire Neural Networks: Zero to Hero video series
- Lillian Weng's "The Transformer Family v2" megapost

Questions?

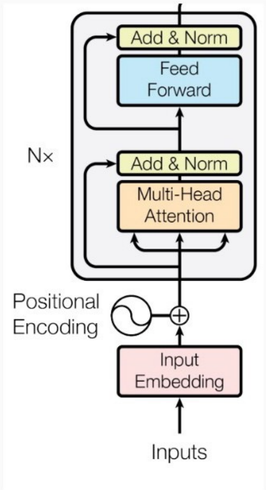
Notable LLMs


Evolutionary Tree
2023
2022
2021
2020
2019
2018
Open-Source
Closed-Source
Chat GLM
Flan UL2
Flan T5
UL2
Tk
ST-MoE
GLM
Switch
mT5
T0
DeBERTa
ELECTRA
ALBERT
Distill BERT
BART
T5
RoBERTa
ERNIE
BERT
ELMo
Ai2
ULMFiT
FastText
GloVe
Word2Vec
Encoder-Only
Encoder-Decoder
Decoder-Only
LLaMA
Bard
GPT-4
Jurassic-2
Claude
OPT-IML
BLOOMZ
ChatGPT
Galactica
Anthropic LM_v4-s3
Sparrow
Flan PaLM
BLOOM
YaLM
Minerva
OPT
PaLM
Chinchilla
InstructGPT
GPT-NeoX
LaMDA
CodeX
GLaM
Gopher
ERNIE3.0
Anthropic LM
Cohere
MT-NLG
Jurassic-1
GPT-J
GPT-Neo
GPT-3
XLNet
GPT-2
GPT-1
open source
closed source
2
1 1
1 1
3
3
3
5
6
5 2
7 9

Three Easy Pieces

BERT

[*] [*] **[sat_]** [*] **[the_]** [*]



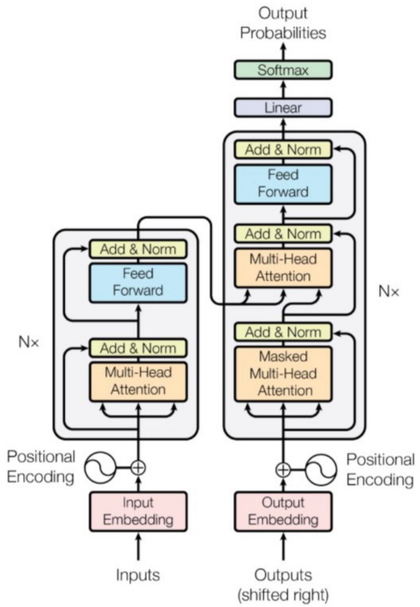
[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

T5

Das ist gut.

A storm in Attala caused 6 victims.

This is not toxic.



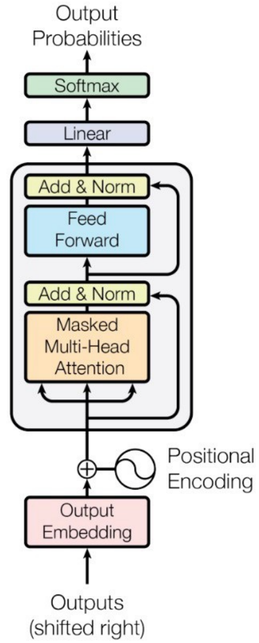
Translate EN-DE: This is good.

Summarize: state authorities dispatched…

Is this toxic: You look beautiful today!

GPT

[sat_]



[START] [The_] [cat_]

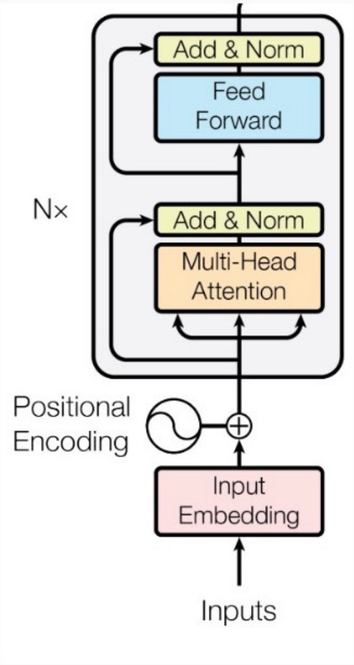
Transformer image source: "Attention Is All You Need" paper

From Lukas Beyer's lecture Intro to Transformers.

BERT (2019)

- *Bidirectional* Encoder Representations from Transformers
- Encoder-only (no attention masking)
- 110M params
- 15% of all words masked out
- Was great, now dated

[*] [*] **[sat_]** [*] **[the_]** [*]



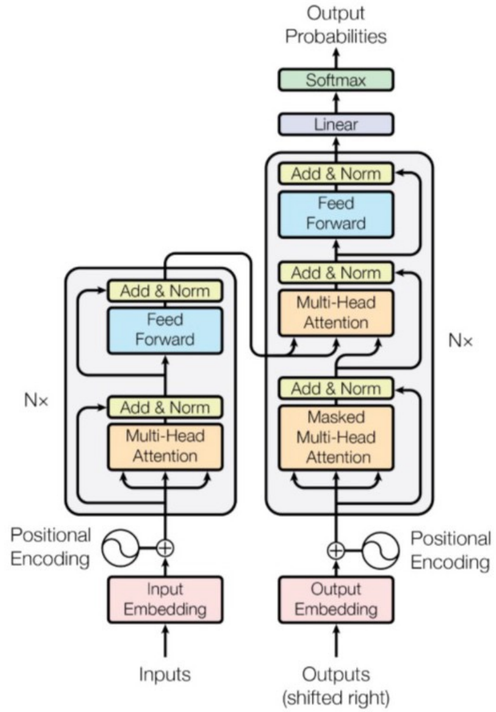
[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

From Lukas Beyer's lecture Intro to Transformers.

T5: Text-to-Text Transfer Transformer (2020)

- Input and output are both text strings
- Encoder-Decoder architecture
- 11Bparameters
- Still could be a good choice for fine-tuning!

```
Das ist gut.
A storm in Attala caused 6 victims.
This is not toxic.
```

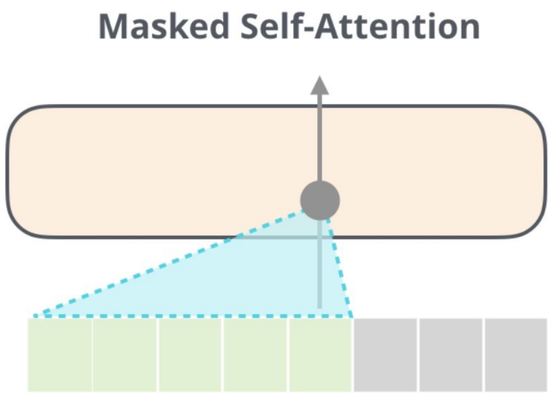


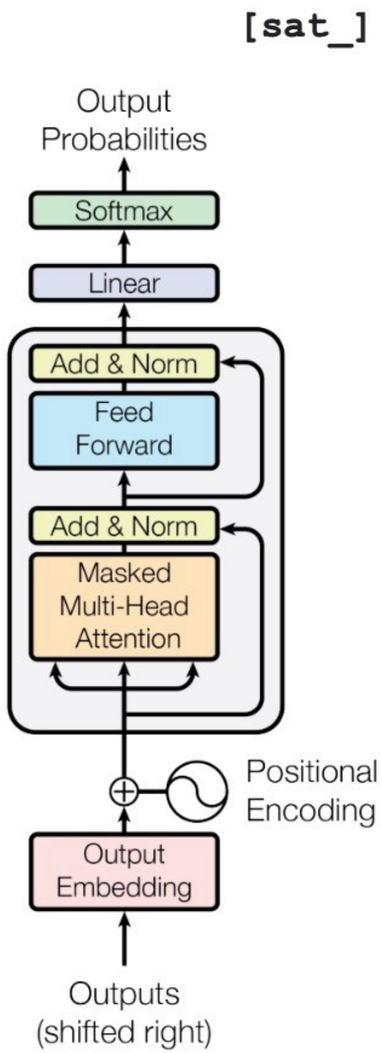

```
Translate EN-DE: This is good.
Summarize: state authorities dispatched…
Is this toxic: You look beautiful today!
```

From Lukas Beyer's lecture Intro to Transformers.

GPT / GPT-2 (2019)

- Generative Pre-trained Transformer
- Decoder-only (uses masked self-attention)
- 1.5B

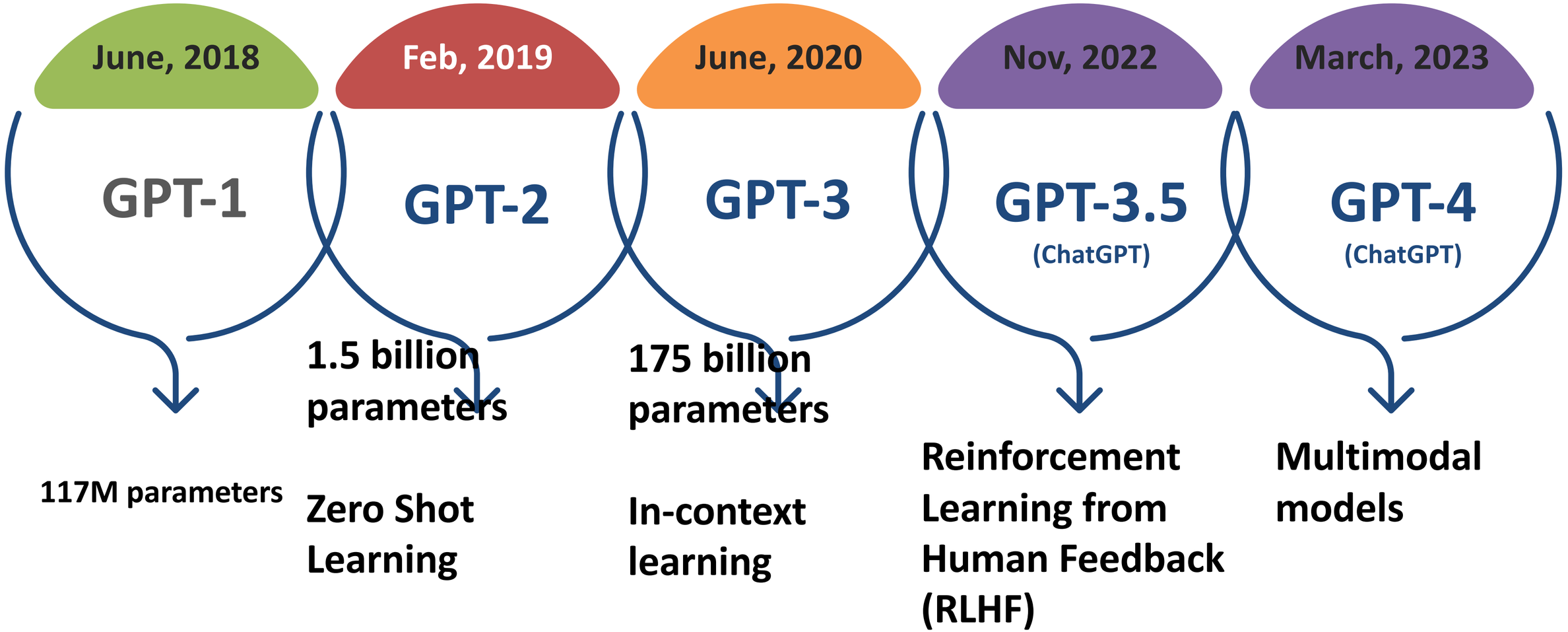




[START] [The_] [cat_]

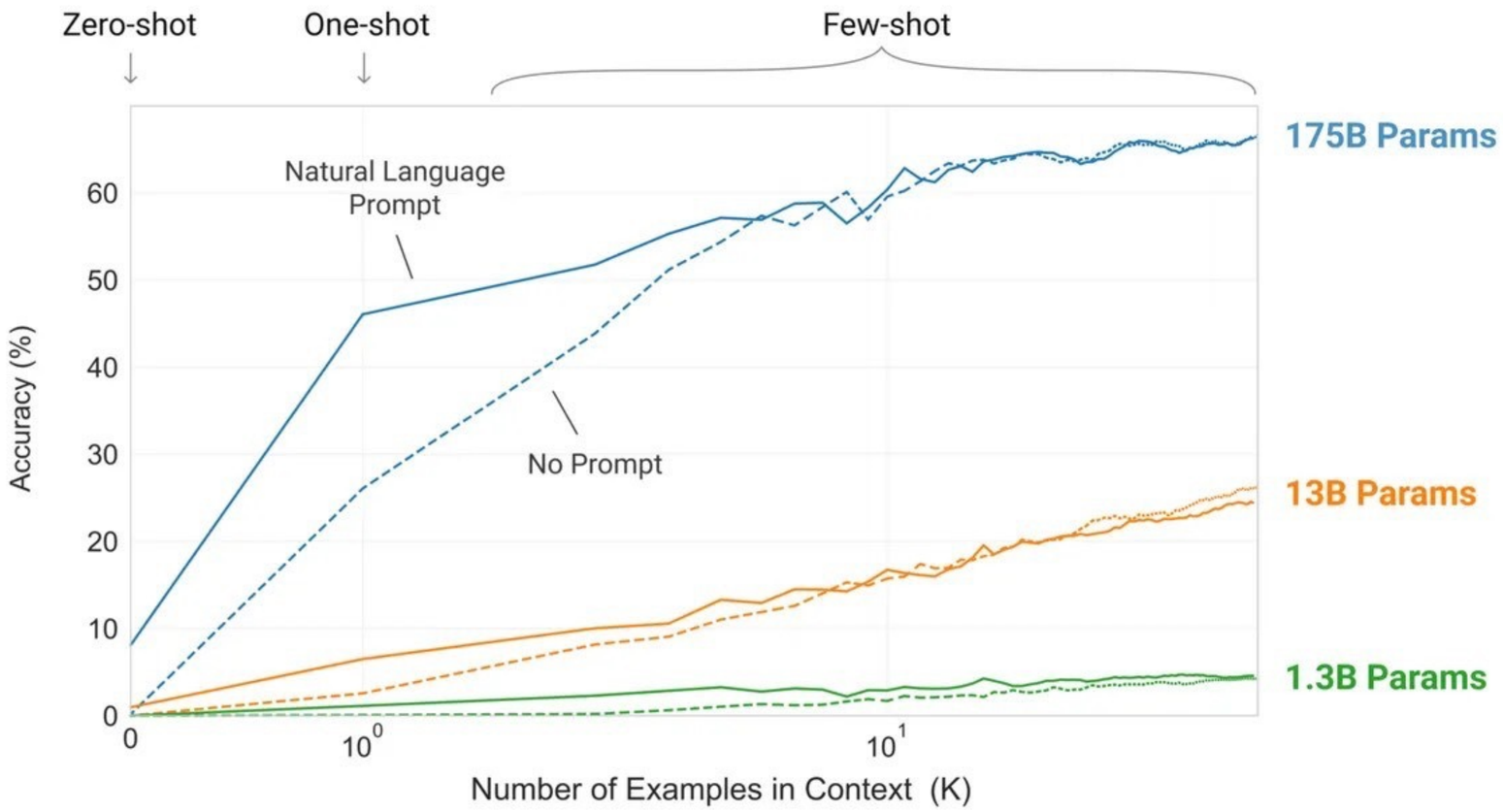
From Lukas Beyer's lecture Intro to Transformers.

GPT Timeline



GPT-3 (2020)

- Just like GPT-2, but 100x larger (175B params)
- Exhibited unprecedented few-shot and zero-shot learning



Zero-shot

The model predicts the answer given only a natural language discription of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
cheese =>                           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese =>                           ← prompt
```

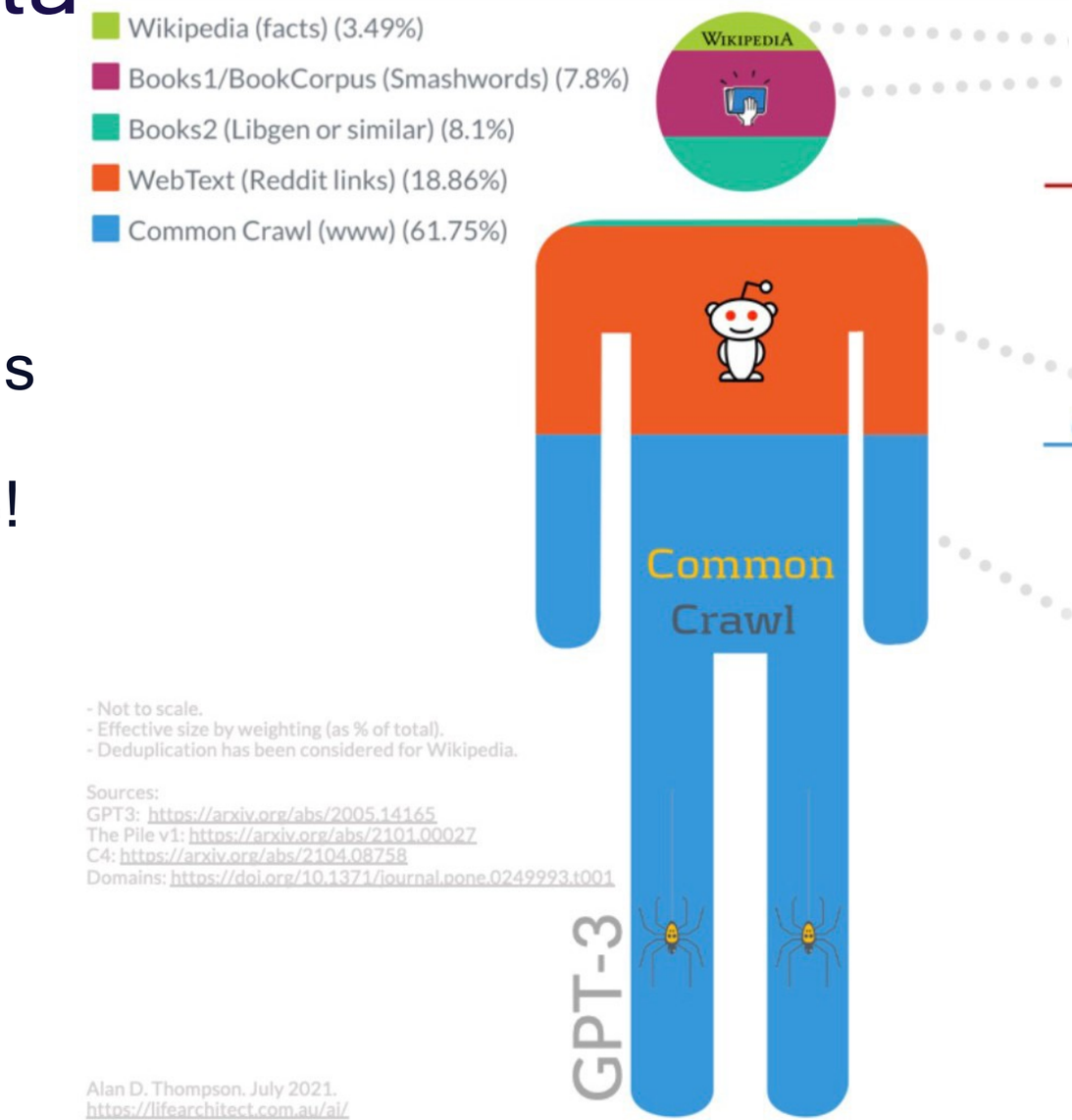
Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée        ←
plush girafe => girafe peluche      ←
cheese =>                           ← prompt
```

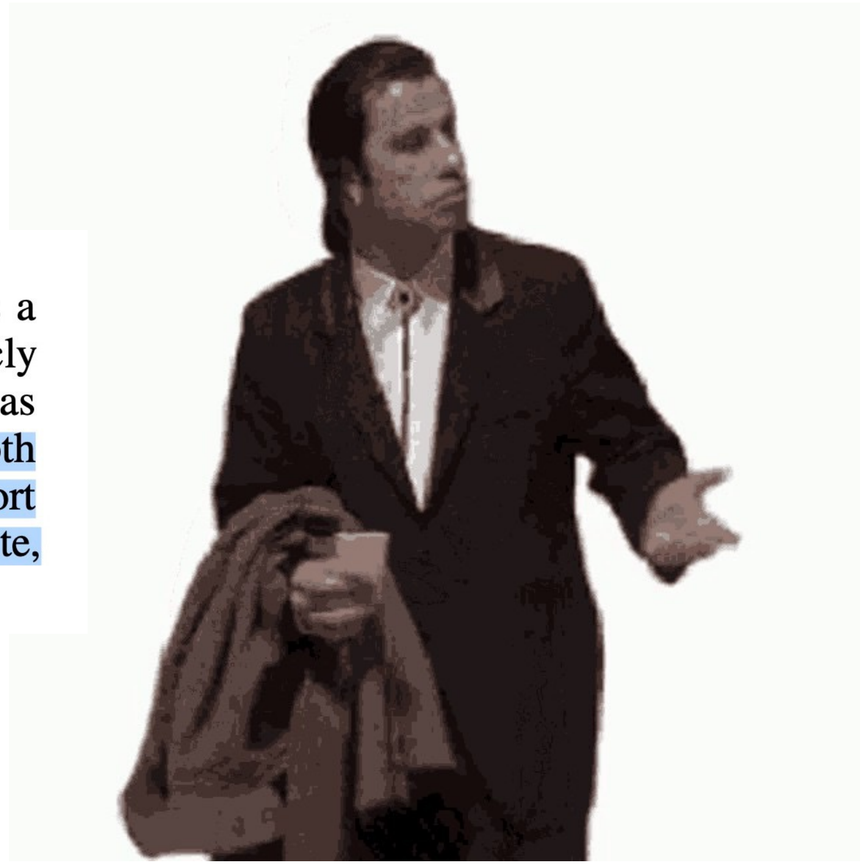
GPT-3 Training Data

- For a total of 500B tokens
- But trained on only 300B!



GPT-4 (2023)

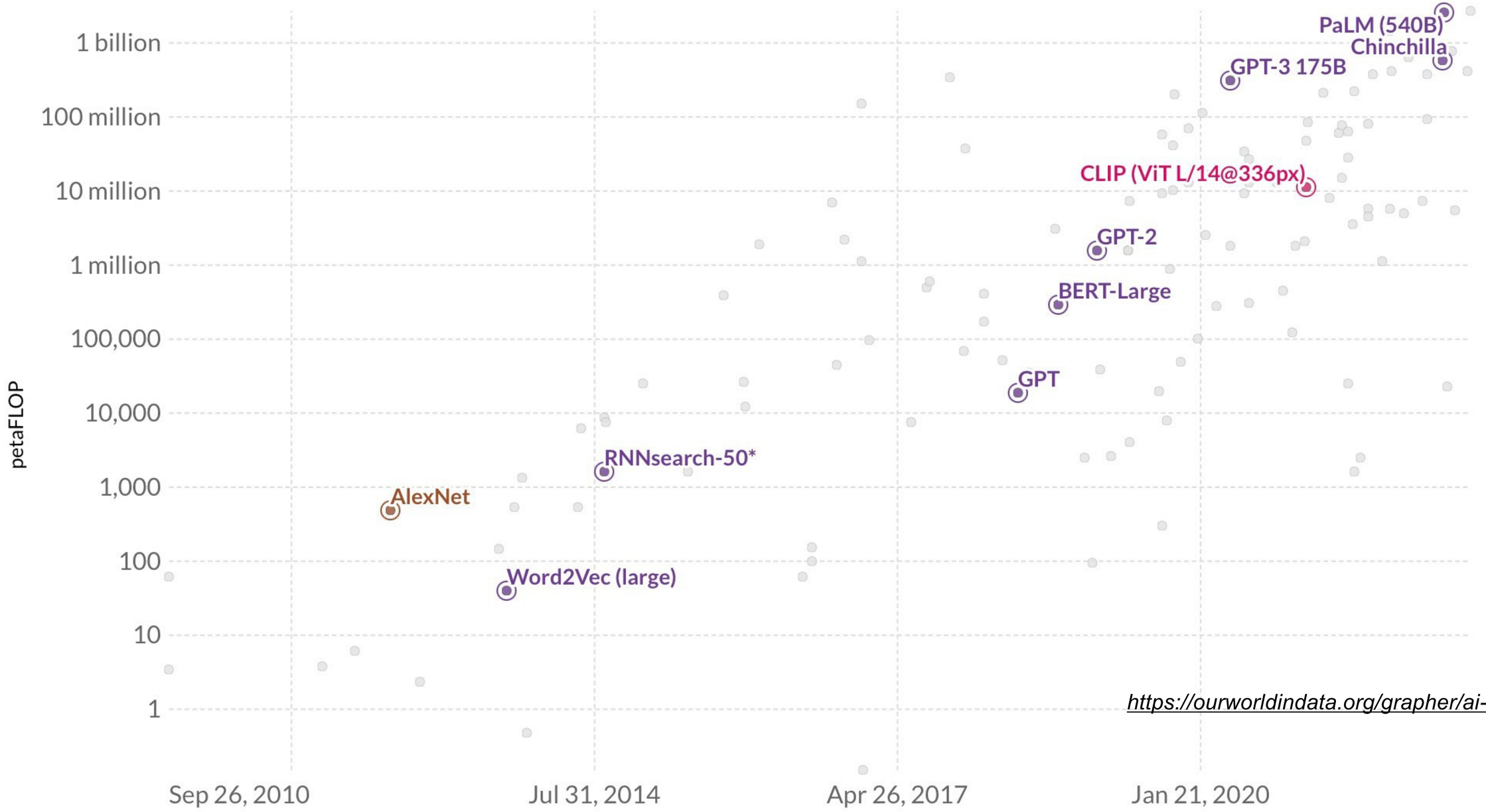
This report focuses on the capabilities, limitations, and safety properties of GPT-4. GPT-4 is a Transformer-style model [39] pre-trained to predict the next token in a document, using both publicly available data (such as internet data) and data licensed from third-party providers. The model was then fine-tuned using Reinforcement Learning from Human Feedback (RLHF) [40]. Given both the competitive landscape and the safety implications of large-scale models like GPT-4, this report contains no further details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar.



Computation used to train notable AI systems

Computation is measured in petaFLOP, which is 10^{15} floating-point operations.

LINEAR | LOG | Select systems | Zoom to selection



https://ourworldindata.org/grapher/ai-training-computation

But what exactly is the relationship between model size and dataset size?

But what exactly is the relationship between model size and dataset size?

Larger models tend to perform better, but the gains from increasing model size diminish unless the dataset size also increases.

Introducion to LLM APIs

Few LLM APIs

- OpenAI APIs
  - Models available: GPT-3.5 Turbo, GPT-4
  - Tasks: Completion, fine-tuning, function calling
- Anthropic
  - Models available: Claude Instant, Claude 2
  - Tasks: Completion.
- AWS Bedrock
  - Models available: Titan, Llama 2, and many more

OpenAI ChatGPT API calls

- Install OpenAI library
- Get an OpenAI API key

API calls are as easy as invoking a REST API

```
secret_key = 'copy and paste your key here'


import openai

# Set up your OpenAI API credentials
openai.api_key = secret_key

# output = openai.Model.list()

output = openai.Completion.create(
    model="text-curie-001",
    prompt="tel me a joke",
)



print(output)
```

Introduction to Running LLMs Locally

Running LLMs Locally

- There are open-source ecosystems offering collections of chatbot LLMs similar to ChatGPT, which can be run locally on your computer.

- Why?
  - Offline Mode
  - Privacy and Security
  - Cost

GPT4All

- GPT4All is an LLM that you can install on your computer
- Try different models locally with UI setting

What models are supported by the GPT4All ecosystem?

Currently, there are six different model architectures that are supported:

1. GPT-J - Based off of the GPT-J architecture with examples found here
2. LLaMA - Based off of the LLaMA architecture with examples found here
3. MPT - Based off of Mosaic ML's MPT architecture with examples found here
4. Replit - Based off of Replit Inc.'s Replit architecture with examples found here
5. Falcon - Based off of TII's Falcon architecture with examples found here
6. StarCoder - Based off of BigCode's StarCoder architecture with examples found here

Start With GPT-J

Libraries to Develop LLM Applications

Popular libraries for LLM Ecosystem

- LangChain
  - Popular Python library for chaining multiple generative models together
- Streamlit
  - For building ChatGPT like web interface
- Hugging Face
  - Python tools for pre-processing, training, fine-tuning, and deployment
- Pinecone, Chroma DB, Milvus
  - Vector databases

AI Feature Proposal

Due 25th February

You will need to integrate an AI-based feature into your web application. Before you begin your implementation, you will need to submit a proposal for the feature you will be implementing, thinking about the scope and the overall design of the feature, and how you will integrate AI. This proposal does not need to be a precise design document but rather should be a rough sketch to make your idea a little more concrete.

Upon submitting your proposal, we will provide feedback on the above, and either return "approve" or "needs revision".