LLM Bootcamp 2023

LLM Foundations

Sergey Karayev

April 21, 2023

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

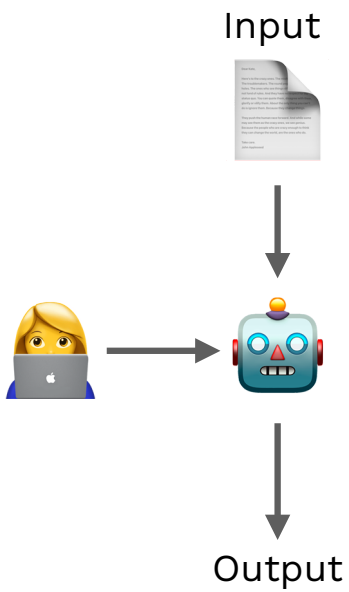
Training & Inference

Running a Transformer

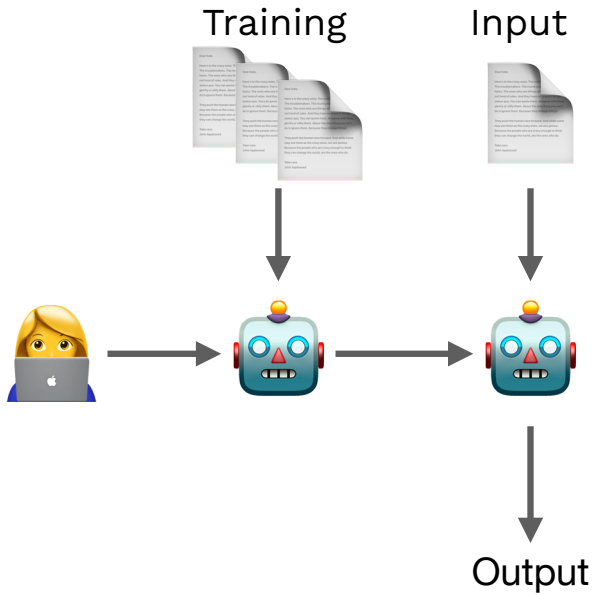
00

Foundations of Machine Learning

Traditional Programming vs Machine Learning



Software 1.0



Software 2.0

Types of Machine Learning

Unsupervised Learning

Learn structure of data to generate more data

"This product does what it is supposed __"

Supervised Learning

Learn how data maps to labels to recognize or predict

→ *cat*

→ *"Hey Siri"*

Reinforcement Learning

Learn how to act in an environment to obtain reward

Converging on just...

Supervised or Self-supervised Learning

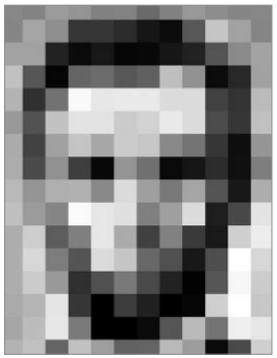
"This product does what it is supposed __" → *"to."*

→ *cat*

→ *"Hey Siri"*

→ *next move*

Inputs and outputs are always just 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"

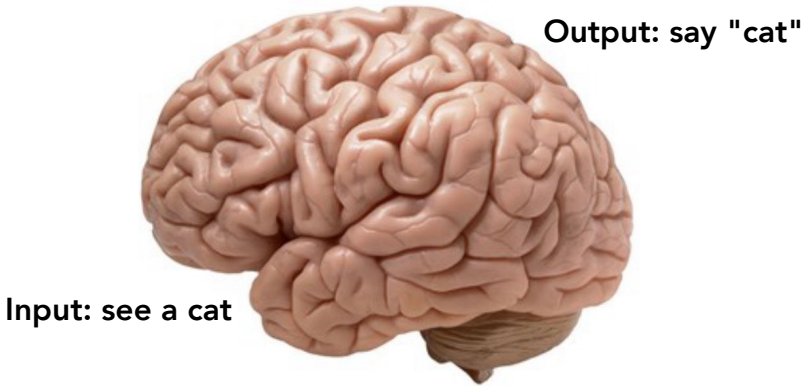
"As good as The Godfather"

"🔥 no cap"

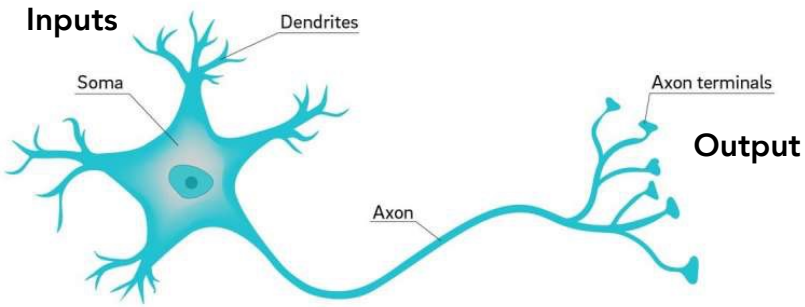
How is it done?

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

Inspiration



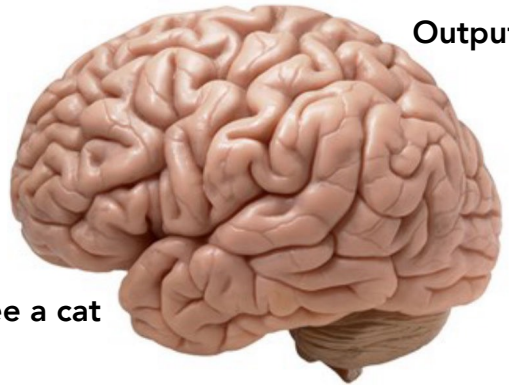
https://www.the-scientist.com/the-nutshell/what-made-human-brains-so-big-36663



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

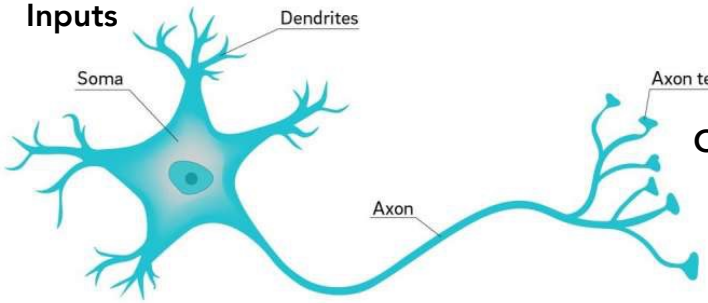
Formalization



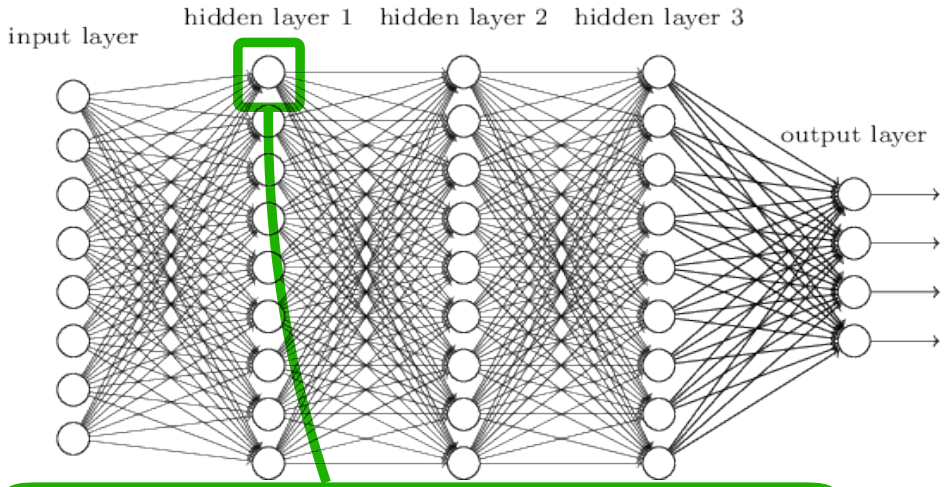
Output: say "cat"

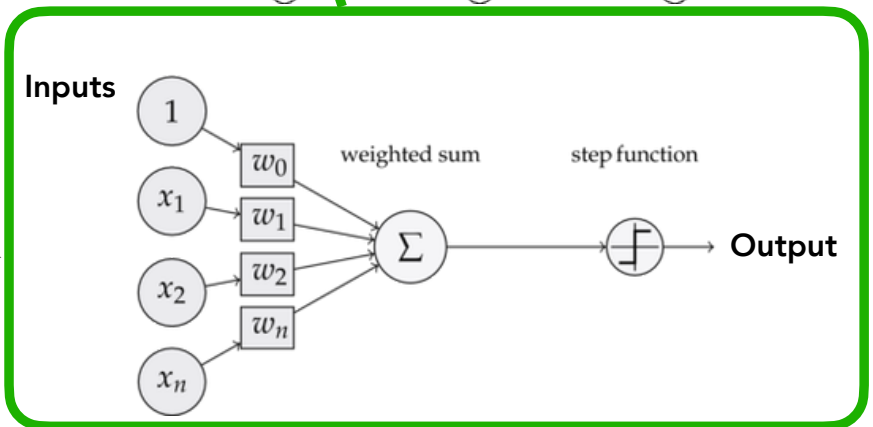
Input: see a cat

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



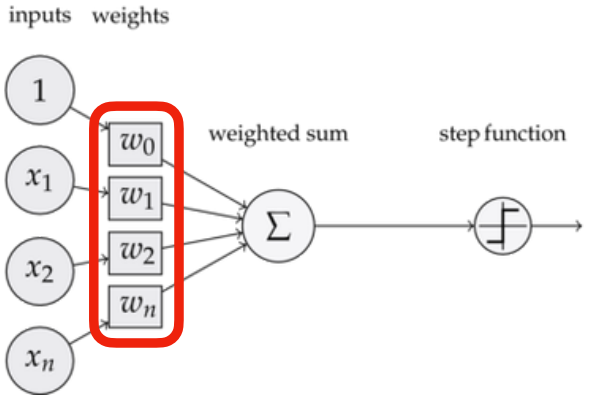
https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html





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

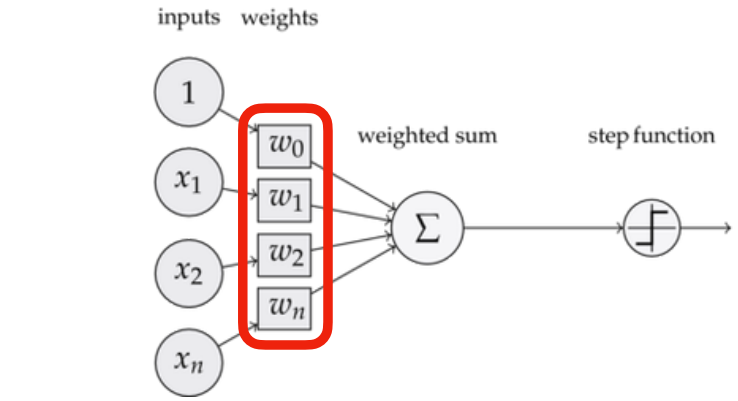
A "perceptron" is a vector of numbers



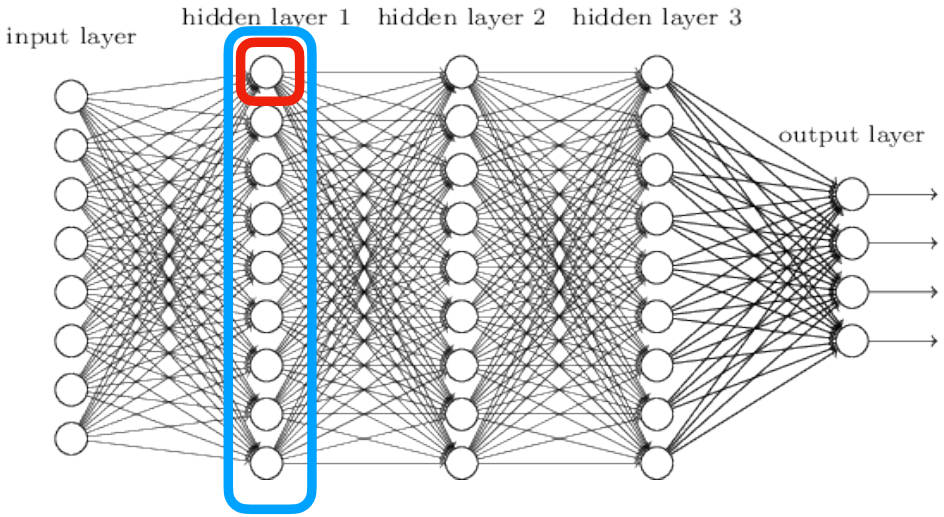
https://www.jessicayung.com/explaining-tensorflow-code-for-a-multilayer-perceptron/

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

A "layer" is a matrix of numbers



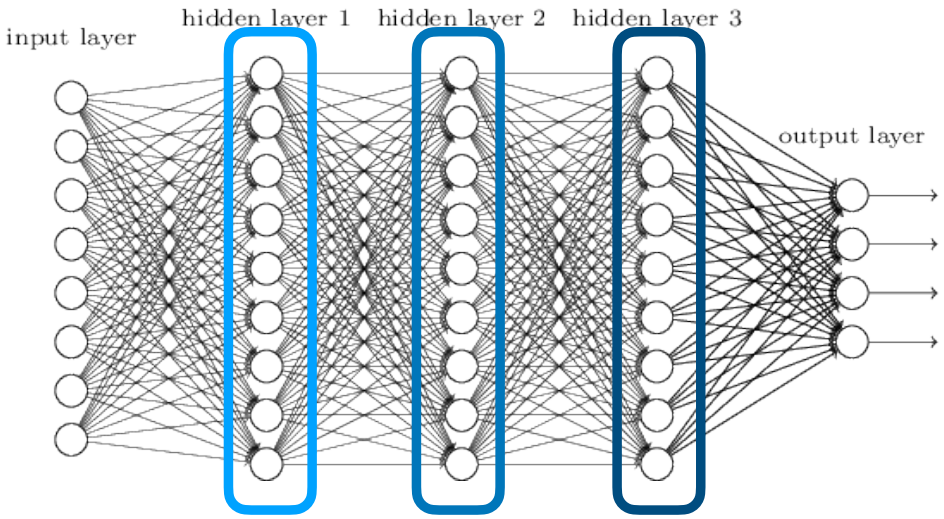
https://www.jessicayung.com/explaining-tensorflow-code-for-a-multilayer-perceptron/

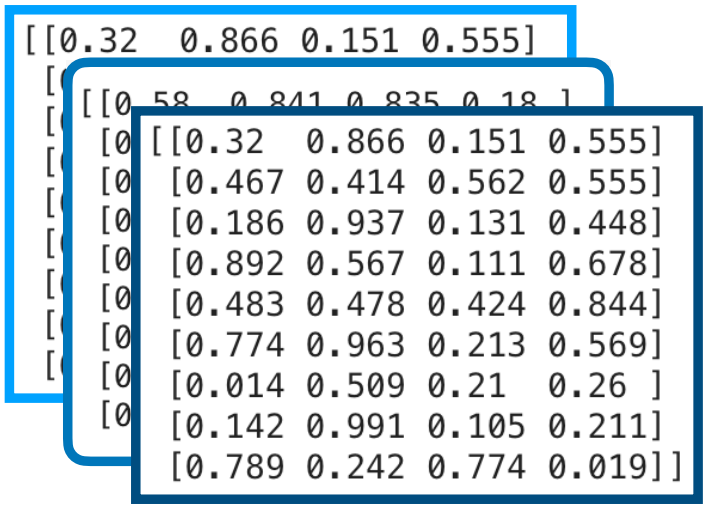



```
[[0.58  0.841 0.835 0.18 ]
 [0.405 0.813 0.309 0.562]
 [0.422 0.229 0.46  0.152]
 [0.673 0.429 0.441 0.243]
 [0.9   0.744 0.234 0.856]
 [0.971 0.486 0.175 0.248]
 [0.258 0.588 0.478 0.266]
 [0.236 0.496 0.077 0.557]
 [0.413 0.322 0.372 0.741]]
```

The neural network is a set of matrices

Called "parameters" or "weights"

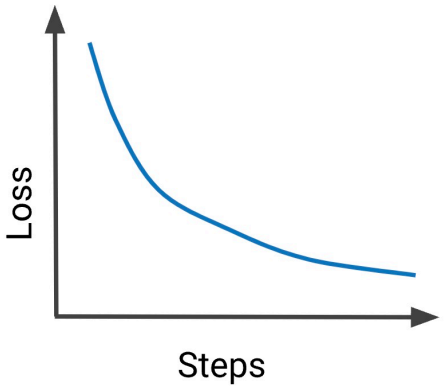




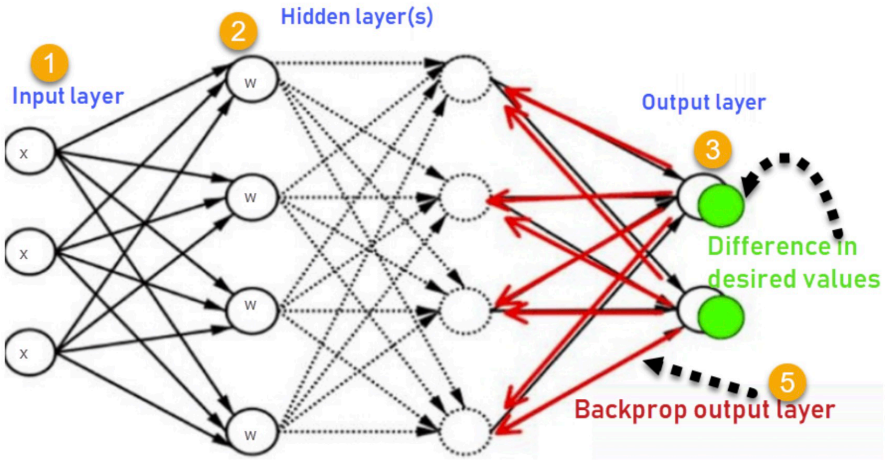
NN operations are just matrix multiplications.
GPUs are really fast at matrix multiplications.

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

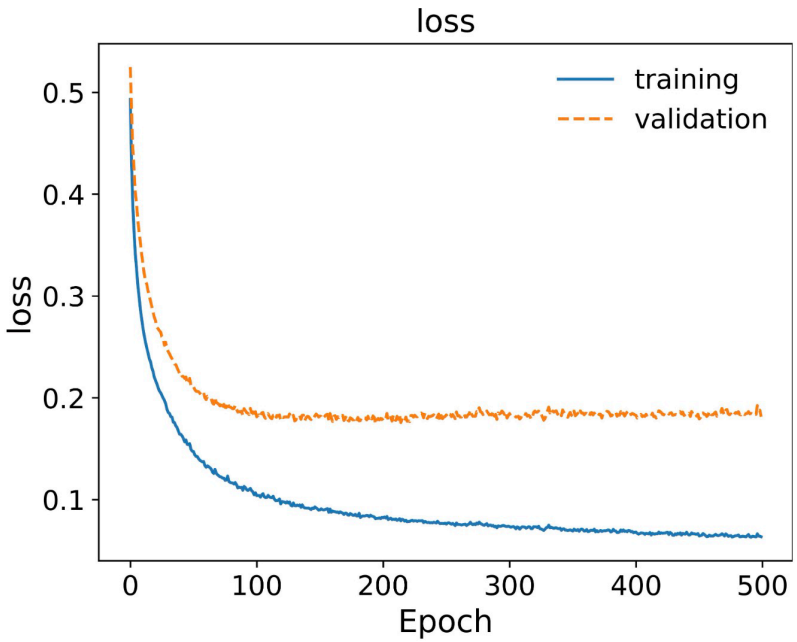


https://www.guru99.com/backpropogation-neural-network.htm

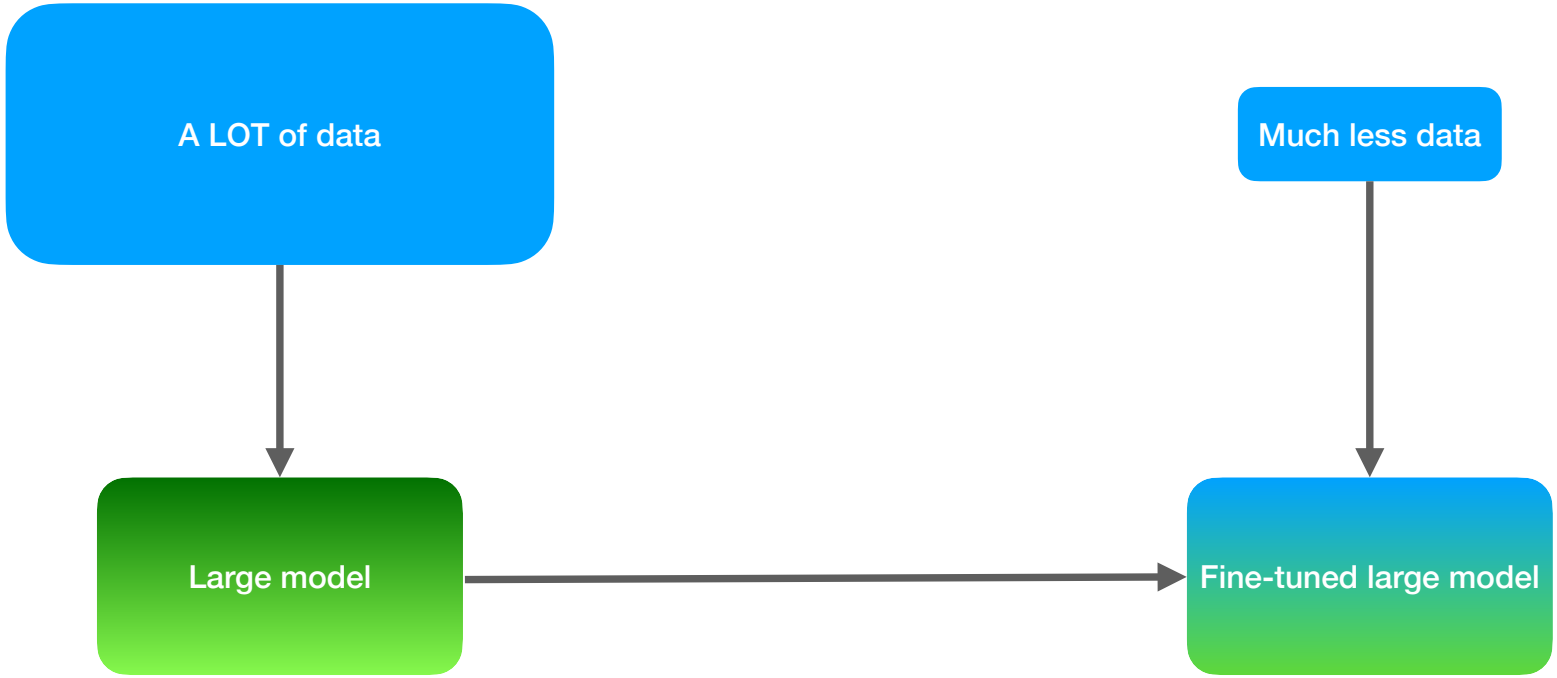
https://developers.google.com/machine-learning/testing-debugging/metrics/interpretic

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!

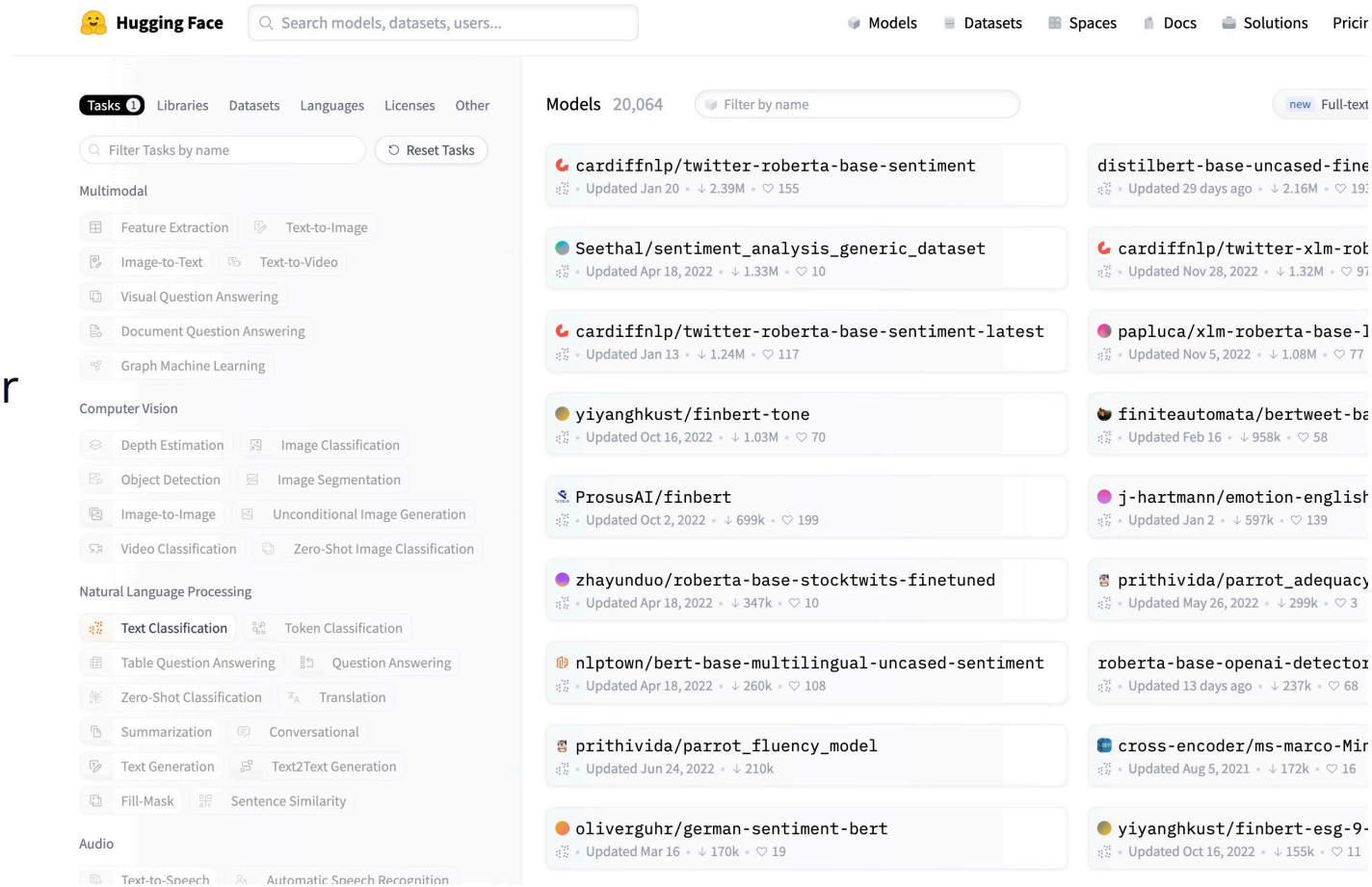


Pre-training:
slow training on a lot of data

Fine-tuning:
fast training on a little data

Model Hubs

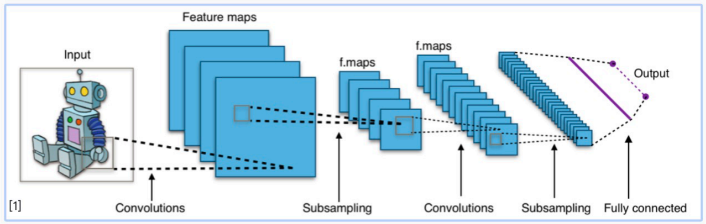
- People share pre-trained models!
- 🤗: the most popular Model Hub
  - 180K models
  - 30K datasets



Before ~2020: each task had its own NN architecture

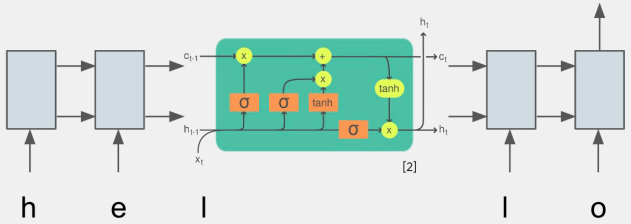
Computer Vision

Convolutional NNs (+ResNets)



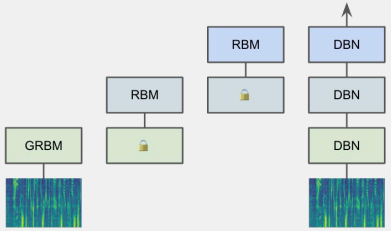
Natural Lang. Proc.

Recurrent NNs (+LSTMs)



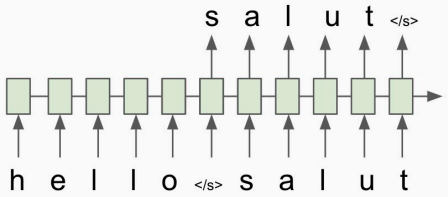
Speech

Deep Belief Nets (+non-DL)



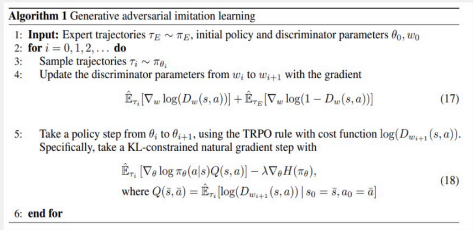
Translation

Seq2Seq



RL

BC/GAIL



Algorithm 1 Generative adversarial imitation learning

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

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

5: 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

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

6: **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

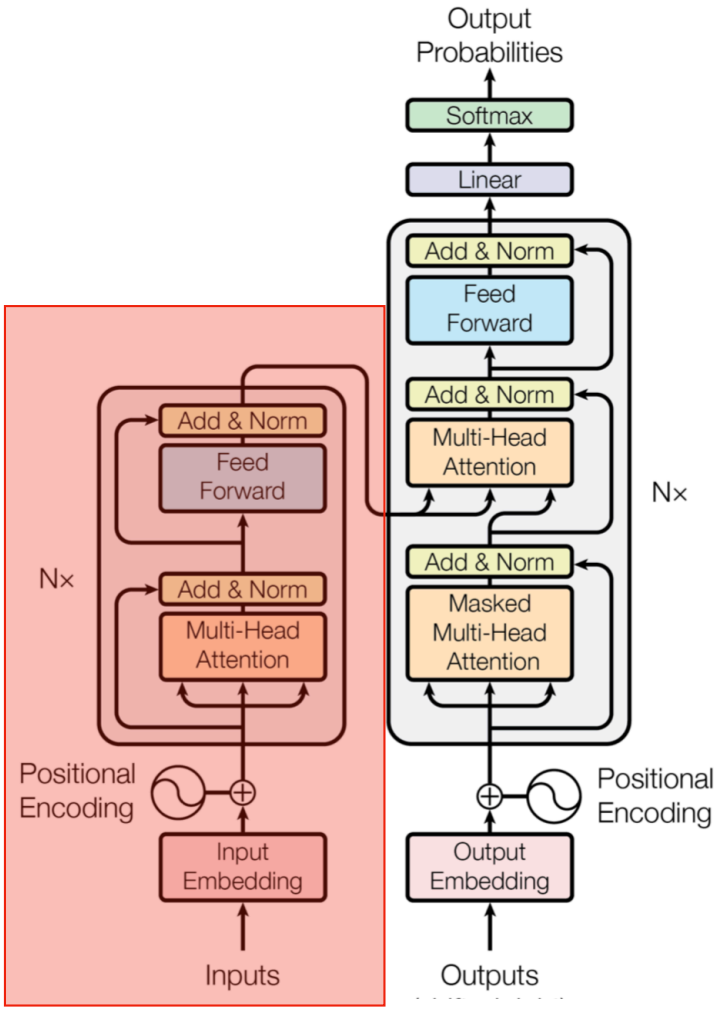
Transformer cartoon (DALL-E)

01

The Transformer Architecture

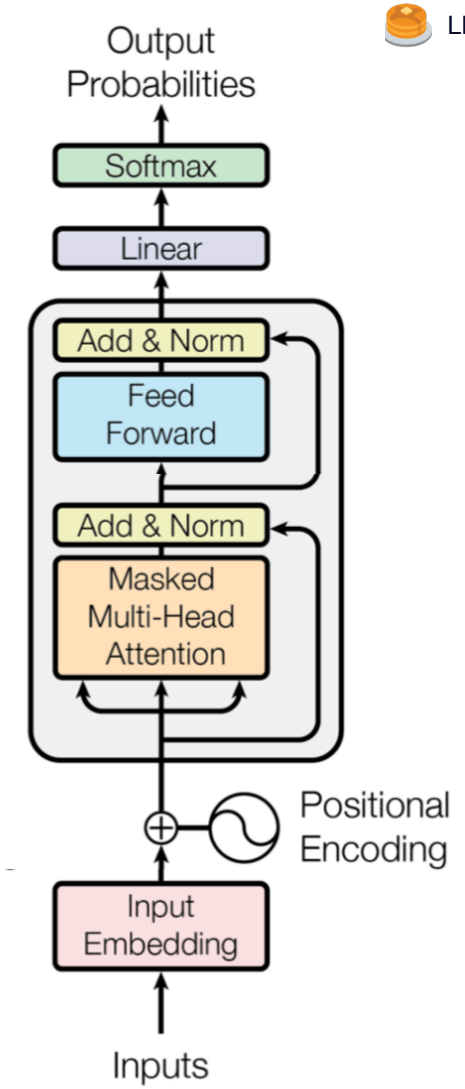
Attention is all you need (2017)

- 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



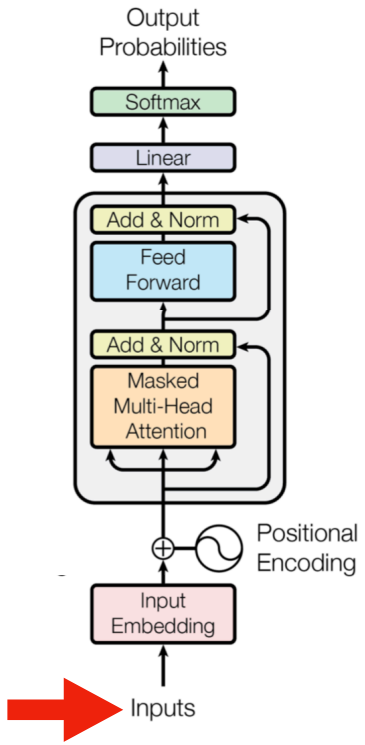
Transformer Decoder Overview

- 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.



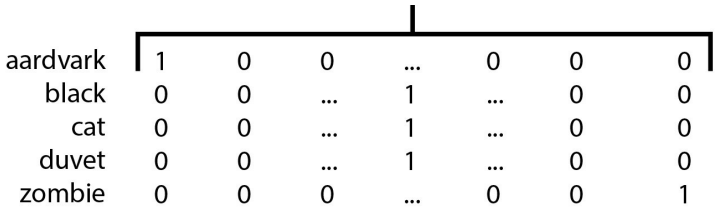
Inputs

- 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. 3 -> [0, 0, 0, 1, 0, 0, 0, 0, ...]

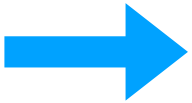


Input Embedding

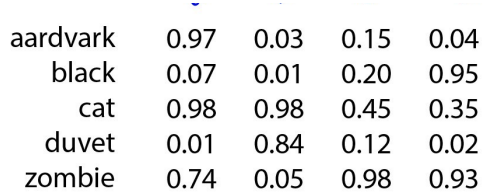
- One-hot vectors are poor representations of words or tokens
  - e.g. distance between "cat" and "kitten" is the same as between "cat" and "tractor"
- Solution: learn an embedding matrix!
  - (The simplest NN layer type)



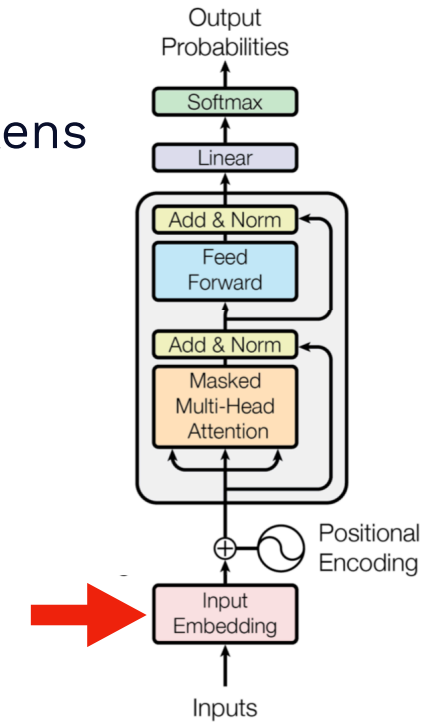
VxV



VxE
embedding matrix

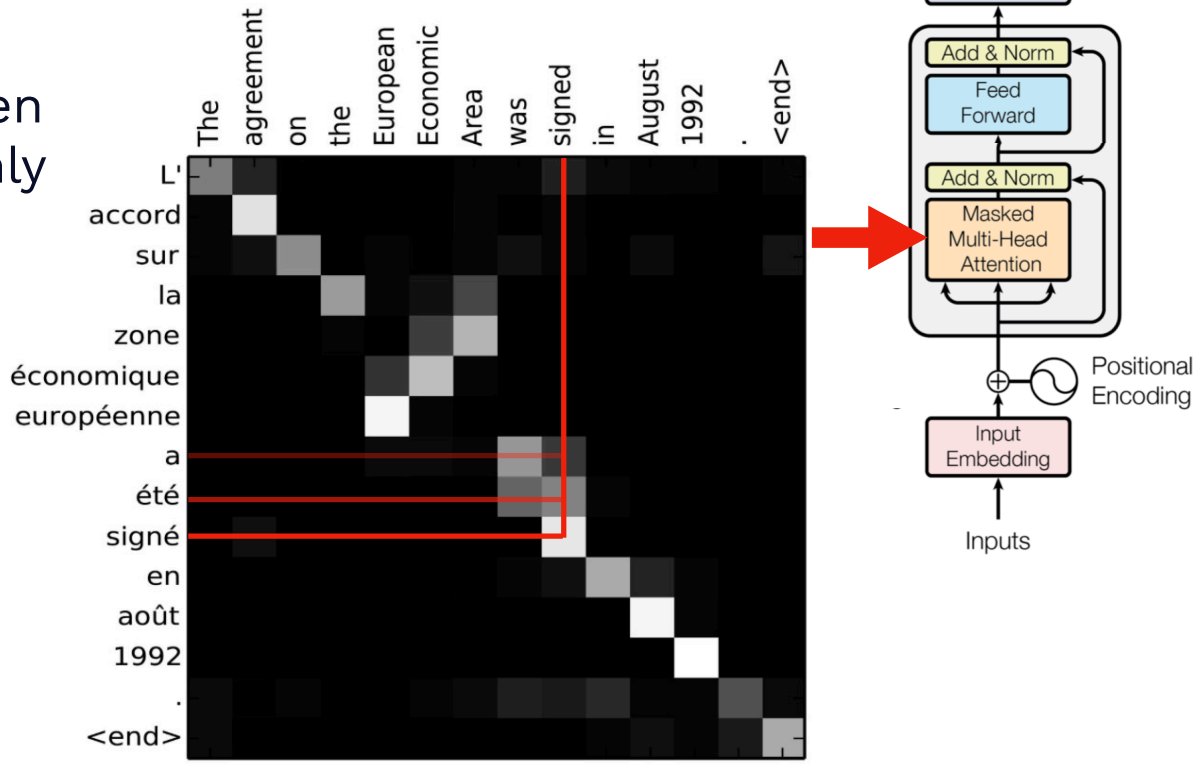


VxE



Attention

- (Ignore "Masked Multi-Head" for now)
- Key insight: for a given token in the output sequence, only one or a few tokens in the input sequence are most important
- Introduced in 2015 for translation tasks



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

Basic self-attention

- Input: sequence of vectors

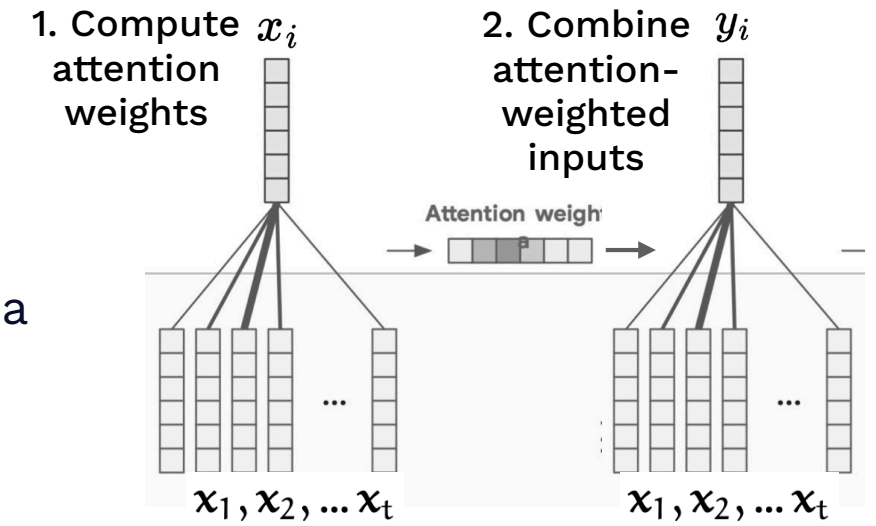
$$\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_t$$

- 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 $w'_{ij} = \mathbf{x}_i{}^{\mathsf{T}} \mathbf{x}_j$
  - (made to sum to 1) $w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$

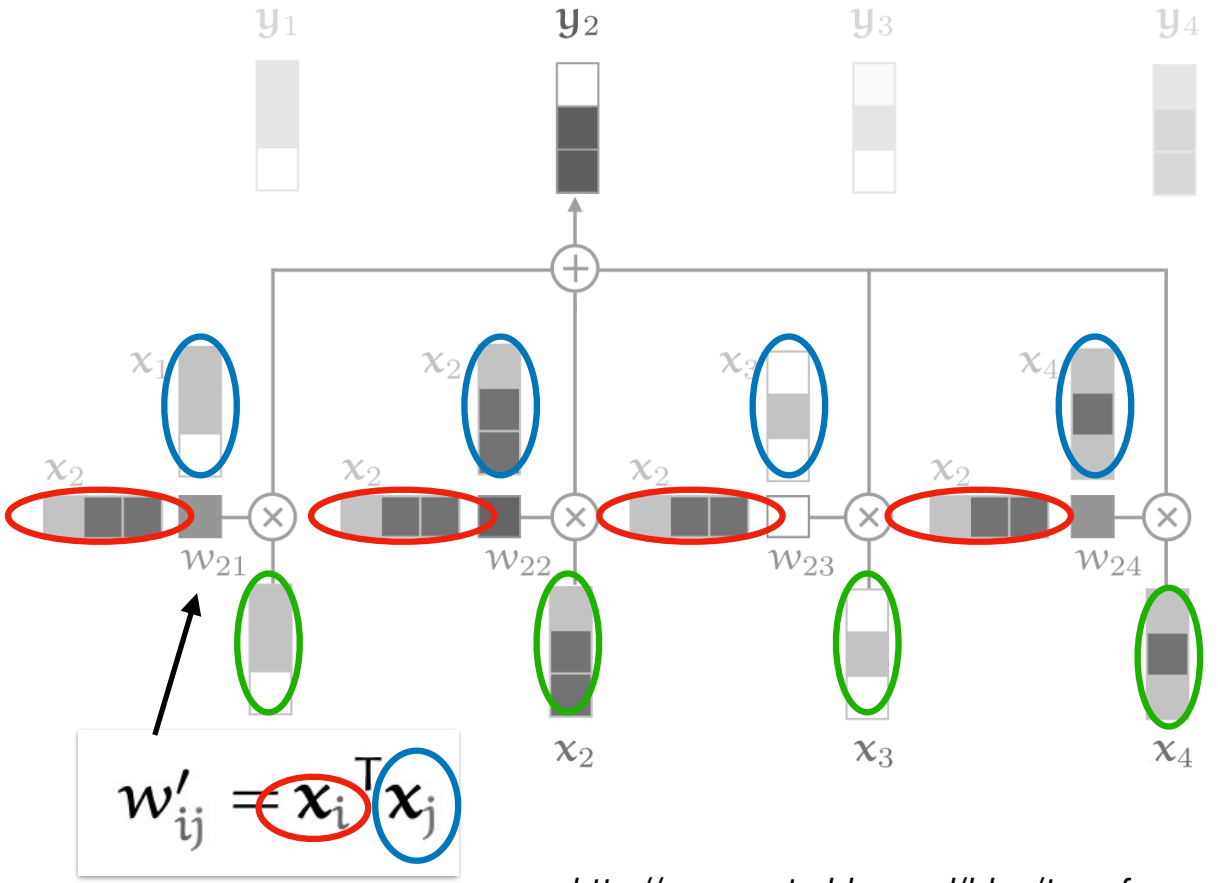


http://www.peterbloem.nl/blog/transformers

Basic self-attention

$$y_i = \sum_j w_{ij} x_j$$

- Note that every input vector x_i is used in 3 ways:
  - Query x_i
  - Key $x_1, x_2, \ldots x_t$
  - Value $x_1, x_2, \ldots x_t$



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

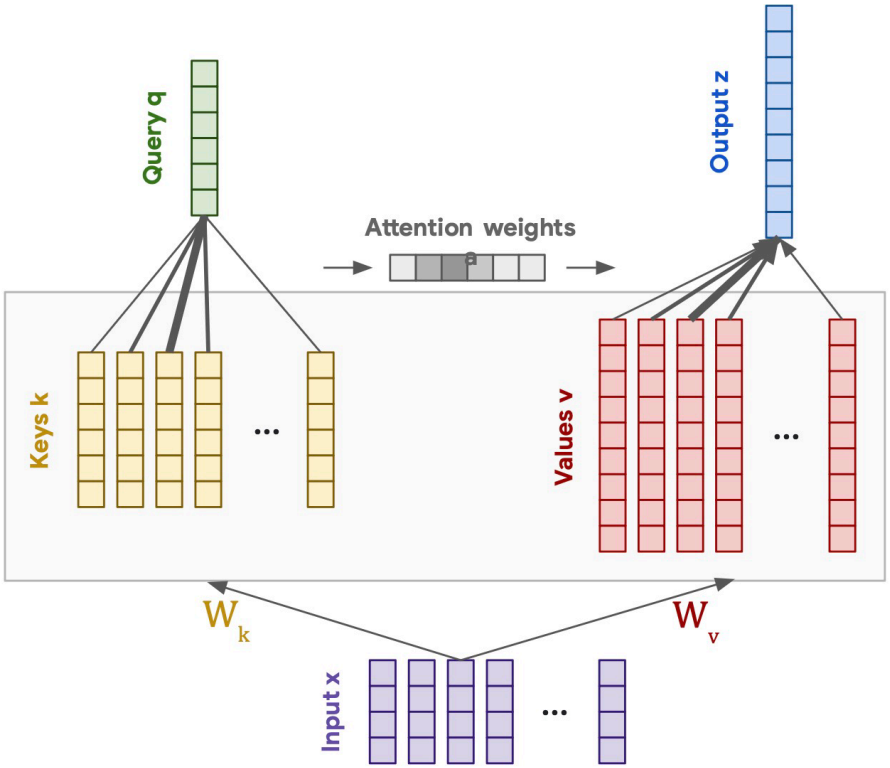
$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i$$

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

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

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

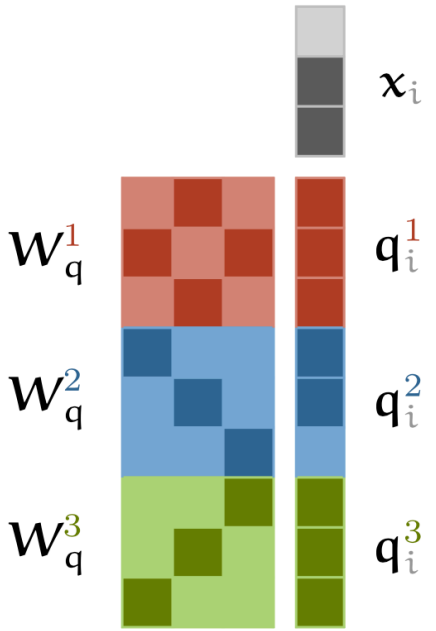
http://www.peterbloem.nl/blog/transformers



http://lucasb.eyer.be/transformer

Multi-head attention

- 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.
  - (Actually implemented as a single matrix, anyway.)



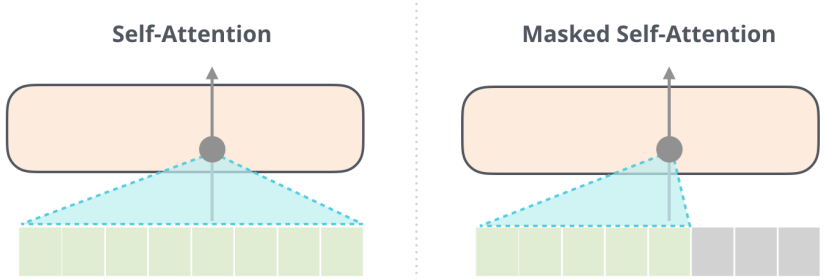
3-headed attention

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

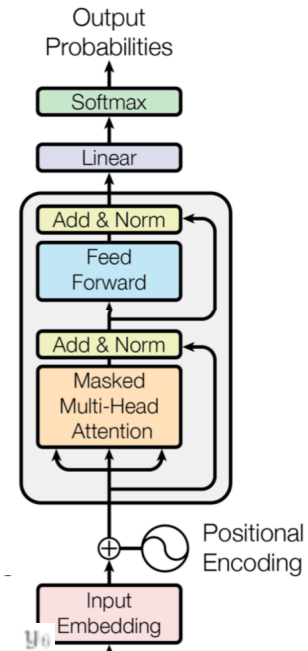
Masking attention

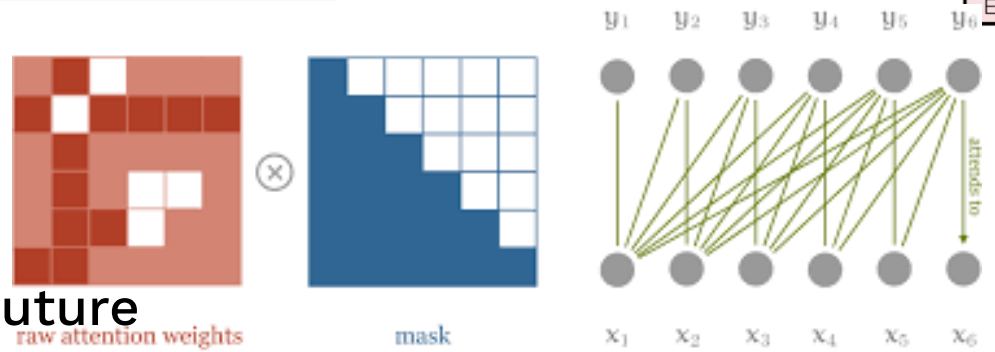
In training:

- 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!**



https://jalammar.github.io/illustrated-



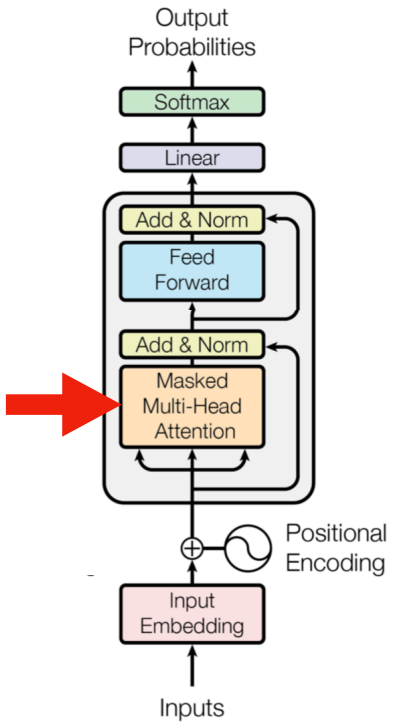


http://www.peterbloem.nl/blog/transformers

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

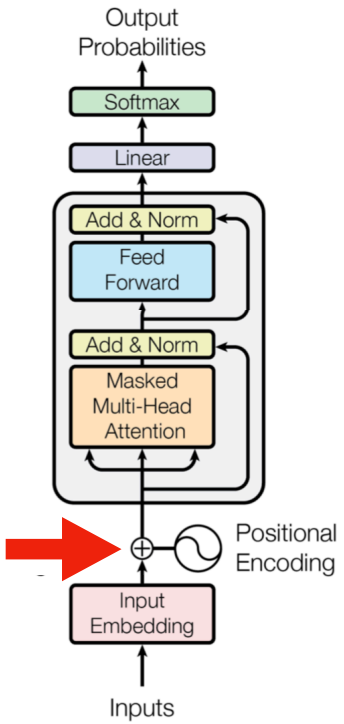
$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i$$

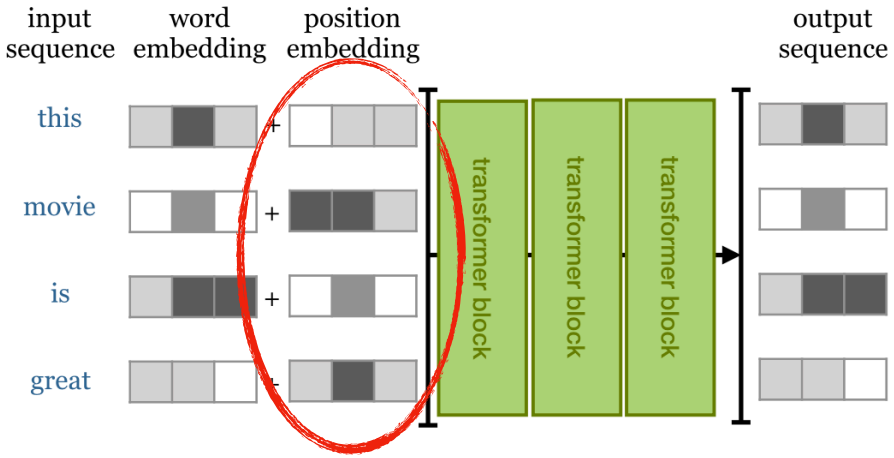
$$w'_{ij} = \mathbf{q}_i{}^\top \mathbf{k}_j$$

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

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

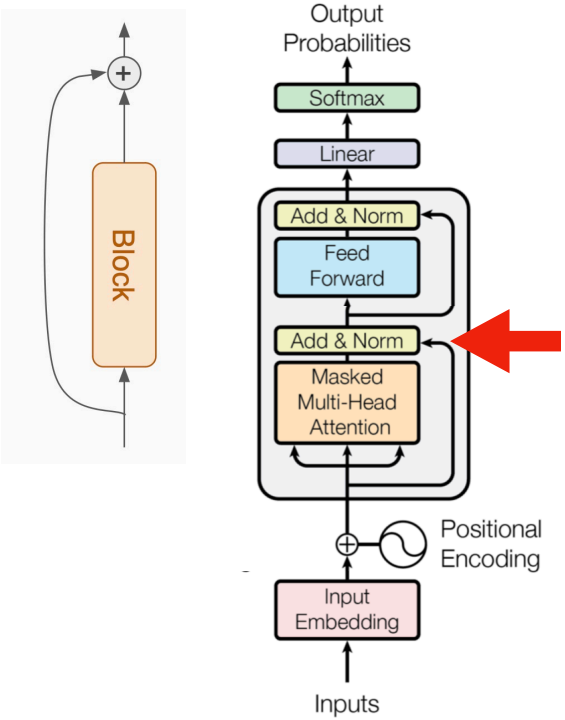
- Attention is totally position-invariant!
  - e.g. [this, movie, is, great] is the same as [movie, this, great, is]
- So, let's add position-encoding vectors to embedding vectors
  - It really is that simple





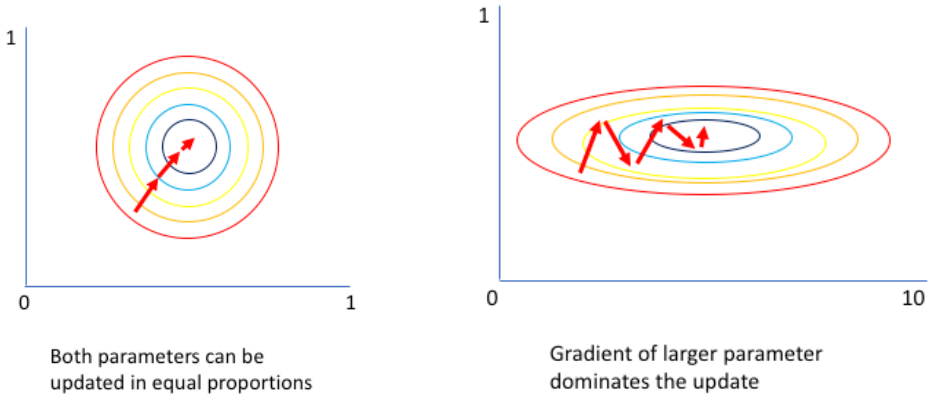
Add

- "Skip connections" aka "residual blocks"
  - output = module(input) + input
  - Allows gradient to flow from the loss function all the way to the first layer
  - (Possible because each module's output is the same shape as its input)

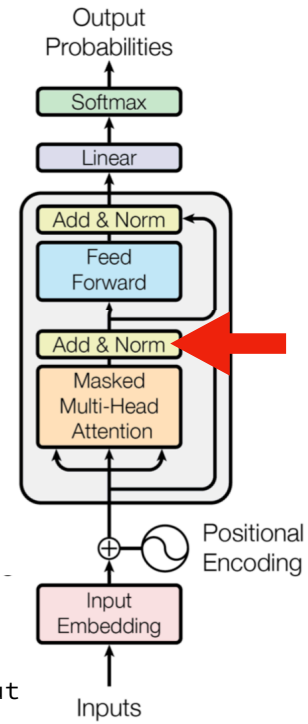


Layer Normalization

- Neural net modules perform best when input vectors have uniform mean and std in each dimension.
- As inputs flow through the network, means and std's get blown out.
- Layer Normalization is a hack to reset things to where we want them in between layers.



https://arxiv.org/pdf/1803.08494.pdf

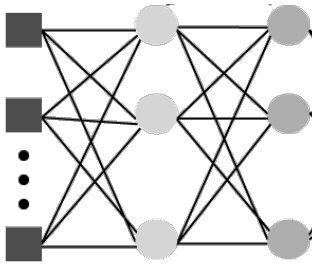


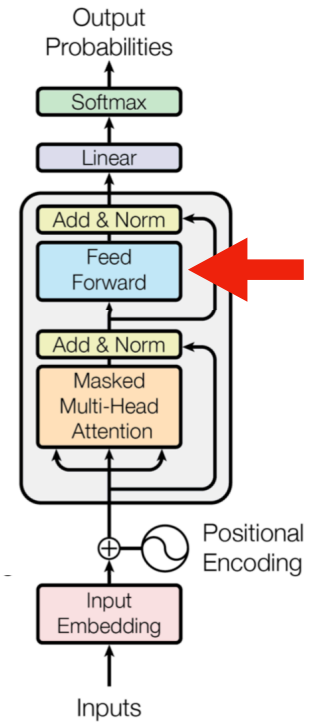

```
output = module(layer_norm(input)) + input
```

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

Feed Forward Layer

- Standard Multi-Layer Perceptron with one hidden layer
- Defined $y = W_2(GeLU(W_1x + b_1)) + b_2$
- Conceptual view:
  - token (augmented with other relevant tokens that it has seen) comes in...
  - ...and "upgrades" its representation

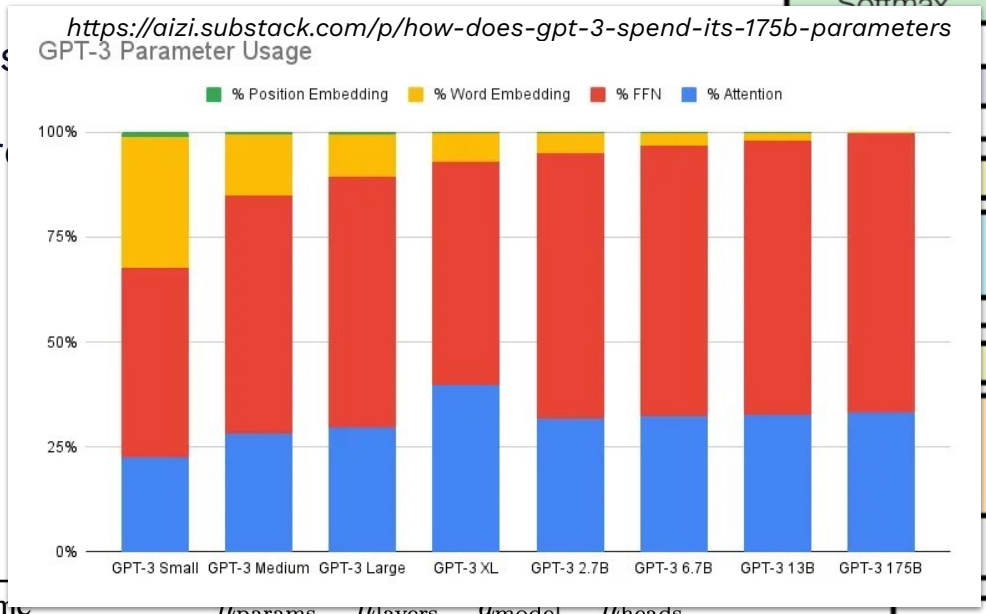




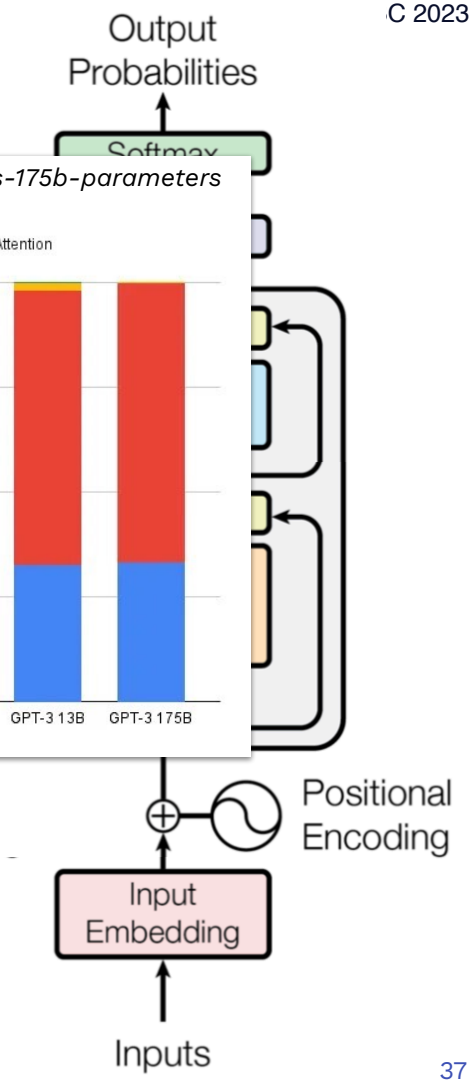
Transformer Architecture

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

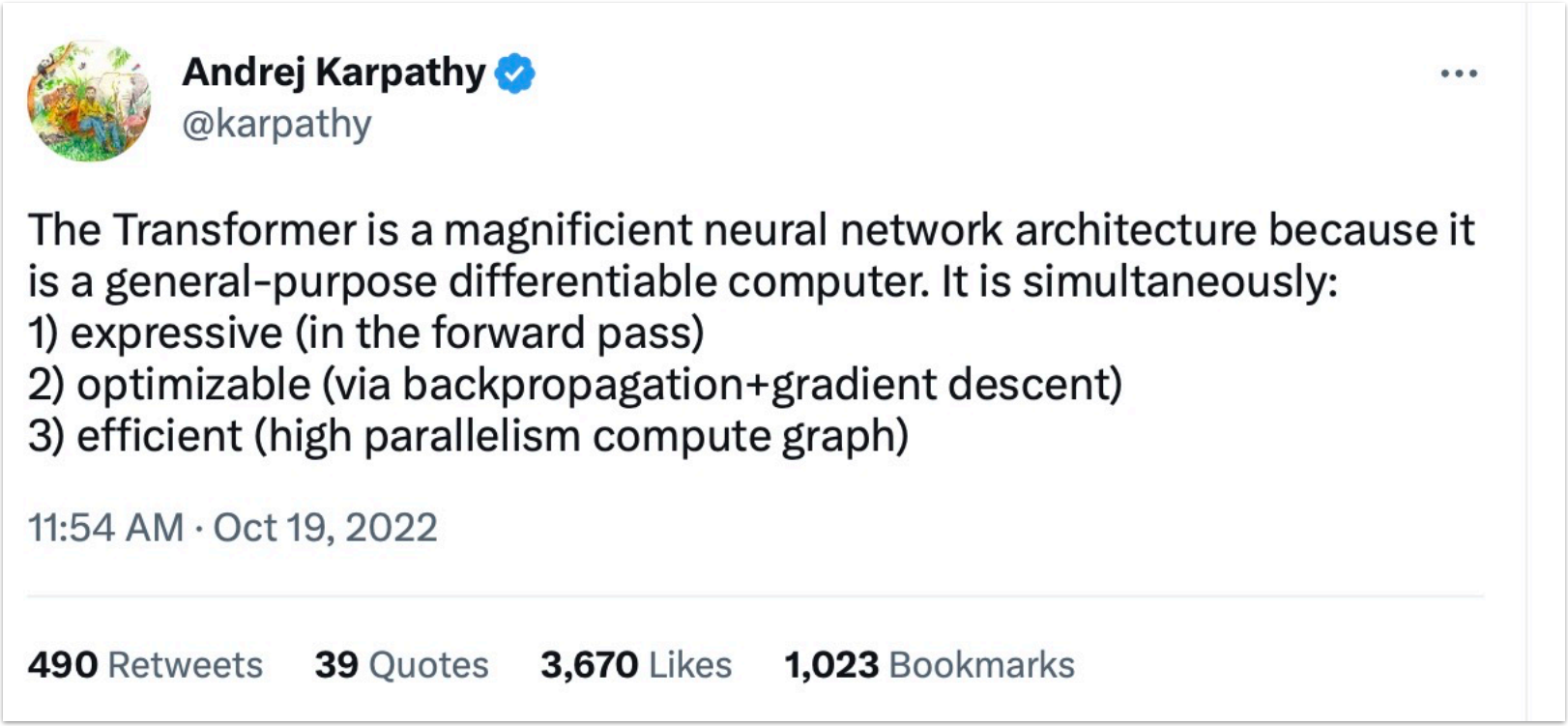
https://aizi.substack.com/p/how-does-gpt-3-spend-its-175b-parameters



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?



Andrej Karpathy
@karpathy

The Transformer is a magnificient neural network architecture because it is a general-purpose differentiable computer. It is simultaneously:
1) expressive (in the forward pass)
2) optimizable (via backpropagation+gradient descent)
3) efficient (high parallelism compute graph)

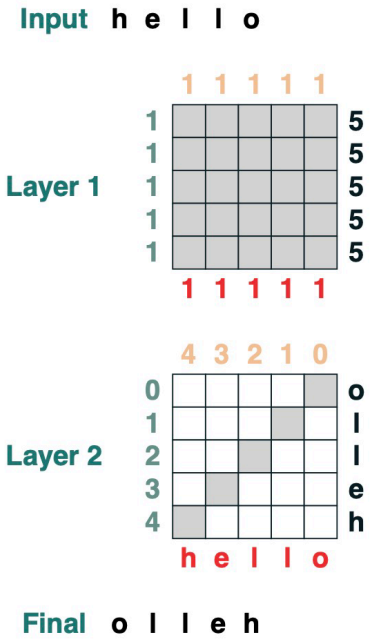
11:54 AM · Oct 19, 2022

490 Retweets **39** Quotes **3,670** Likes **1,023** Bookmarks

Thinking like Transformers

- Restricted Access Sequence Processing (RASP, 2021): programming language of Transformer-implementable operations

```
def flip():
    length = (key(1) == query(1)).value(1)
    flip = (key(length - indices - 1) == query(indices)).value(tokens)
    return flip
flip()
```

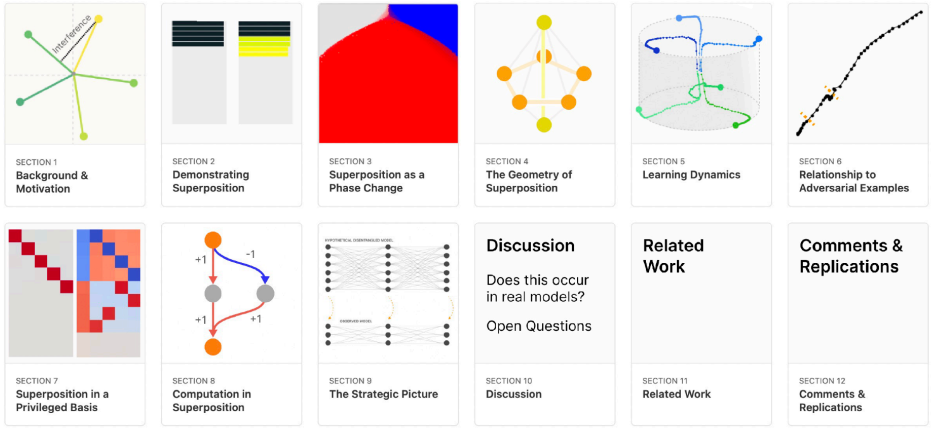


https://srush.github.io/raspy/

We mostly don't understand it, though

- Much great work from Anthropic if this has captured your curiosity!

Toy Models of Superposition



AUTHORS
Nelson Elhage*, Tristan Hume*, Catherine Olsson*, Nicholas Schiefer*, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg*, Christopher Olah*

AFFILIATIONS
Anthropic, Harvard

PUBLISHED
Sept 14, 2022

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

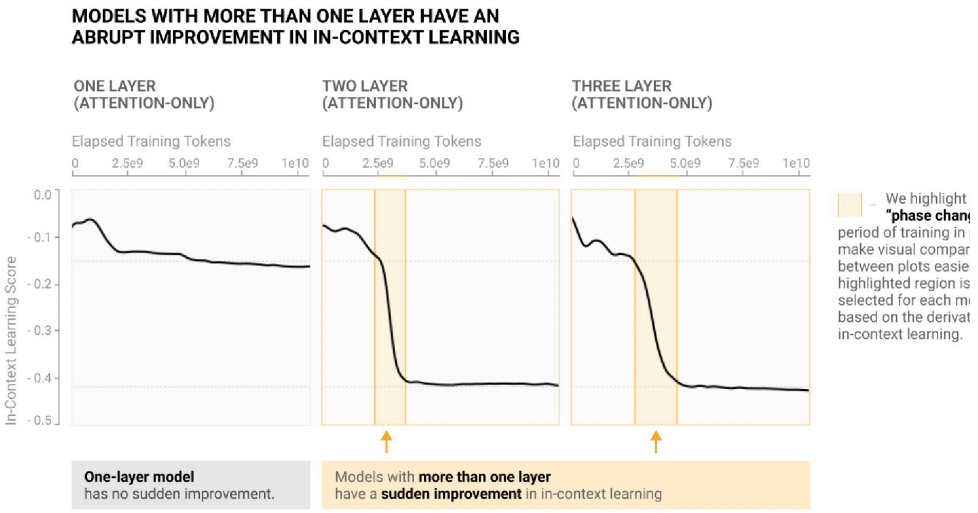
In-context Learning and Induction Heads

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

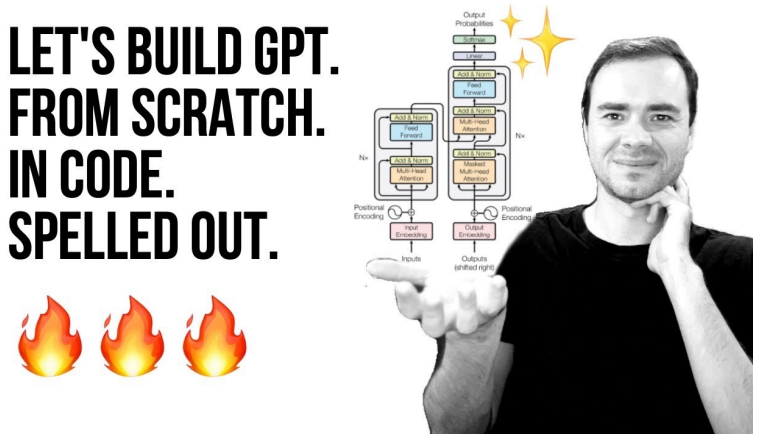
* Core Research Contributor; † Core Infrastructure Contributor; ‡ Correspondence to colah@anthropic.com; Author contributions statement below.



We highlight "phase chan 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 and MLP blocks



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?