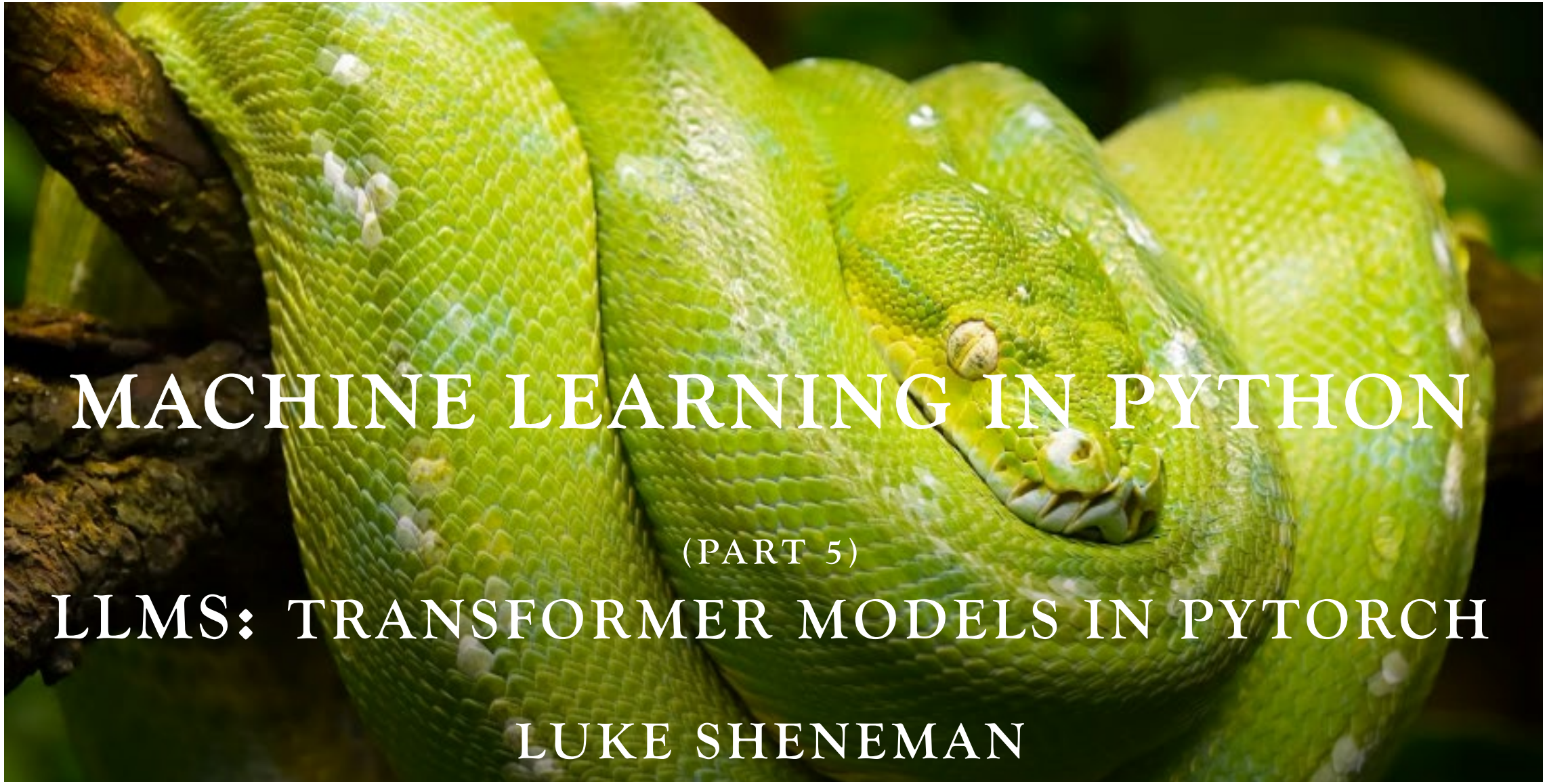


MACHINE LEARNING IN PYTHON

(PART 5)

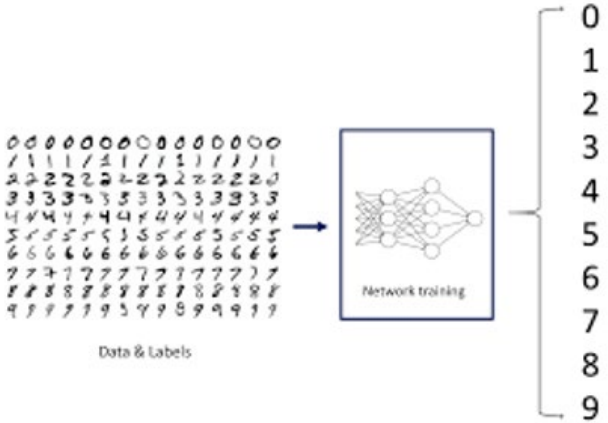
LLMS: TRANSFORMER MODELS IN PYTORCH

LUKE SHENEMAN

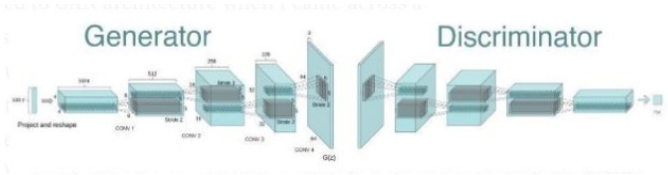
MACHINE LEARNING IN PYTHON SERIES

Part 1: The Basics

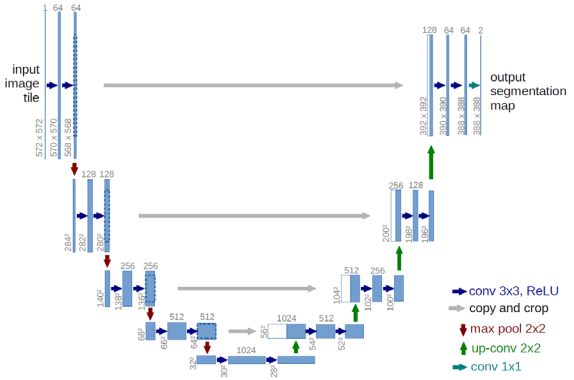
MNIST

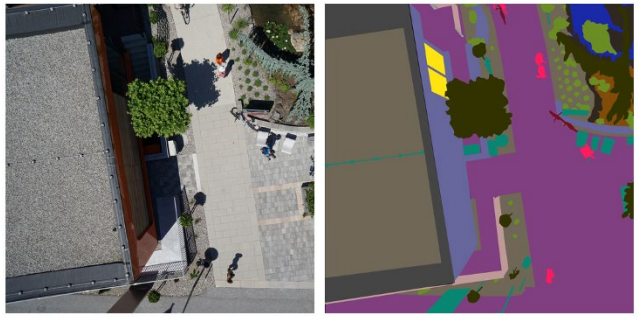


Part 2: Generative Adversarial Networks (GANs) - PyTorch

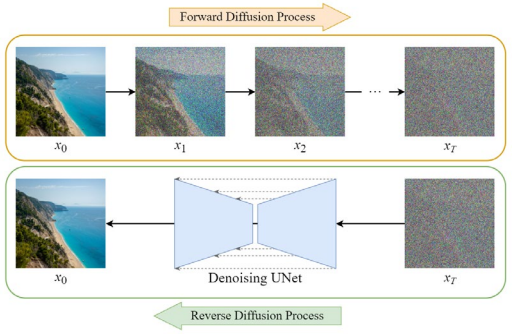


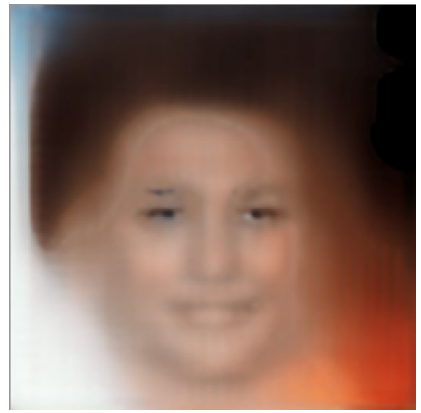
Part 3: Semantic Segmentation in TensorFlow





Part 4: Diffusion Models In PyTorch





GENERATIVE ARTIFICIAL INTELLIGENCE

Text to Image (Stable Diffusion)

Text to Video

EDUCATION

ChatGPT passes medical, law, and business exams

The AI text generator has educators debating its role in the classroom.

Real estate agents say they can't imagine working without ChatGPT now

By Samantha Murphy Kelly, CNN Business
Updated 4:06 PM EST, Sat January 28, 2023

NEWS | 18 January 2023

ChatGPT listed as author on research papers: many scientists disapprove

At least four articles credit the AI tool as a co-author, as publishers scramble to regulate its use.

Chris Stokel-Walk

Science journals ban listing of ChatGPT as co-author on papers

Some publishers also banning use of bot in preparation of submissions but others see its adoption as inevitable

19 Ways to Use ChatGPT in Your Classroom

ChatGPT Is Making Universities Rethink Plagiarism

Students and professors can't decide whether the AI chatbot is a research tool—or a cheating engine.

MUSICO

An AI that writes music

Musico is an AI-driven software engine that generates music. It can react to gesture, movement, code or other sound.

LISTEN TO DEMOS

Google has developed a music-making AI bot

The AI takeover hits the music industry.

By Chase DiBenedetto on January 29, 2023

MY VOICE IS NO LONGER MY PASSWORD —

Microsoft's new AI can simulate anyone's voice with 3 seconds of audio

Text-to-speech model can preserve speaker's emotional tone and acoustic environment.

BENJ EDWARDS - 1/9/2023, 2:15 PM

Denoising EEG Signals for Real-World BCI Applications Using GANs

Eoin Brophy[1,2*], Peter Redmond[1,3], Andrew Fleury[4], Maarten De Vos[5,6], Geraldine Boylan[2] and Tomás Ward[1,3]

Review

Generative adversarial networks unlock new methods for cognitive science

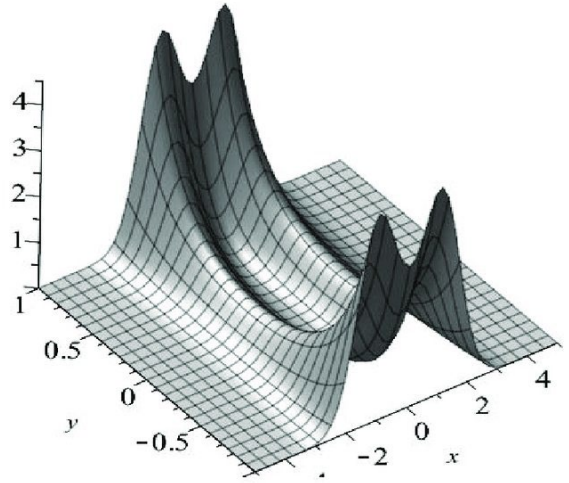
Lore Goetschalckx [1, 2], Alex Andonian [3],

GANs for Driving Biological Research

GitHub Copilot

Generative AI: Learn a *latent* representation of the distribution of our complex training data and then sample from it

Training Data

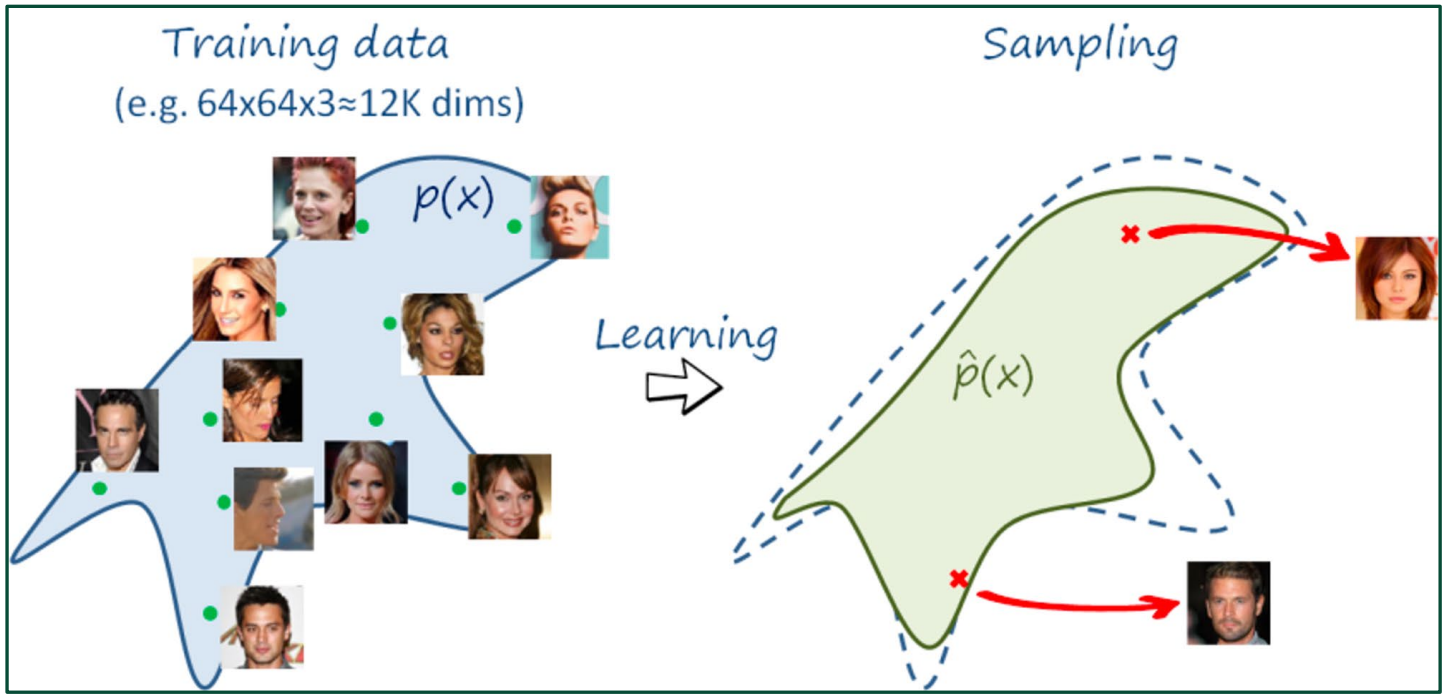


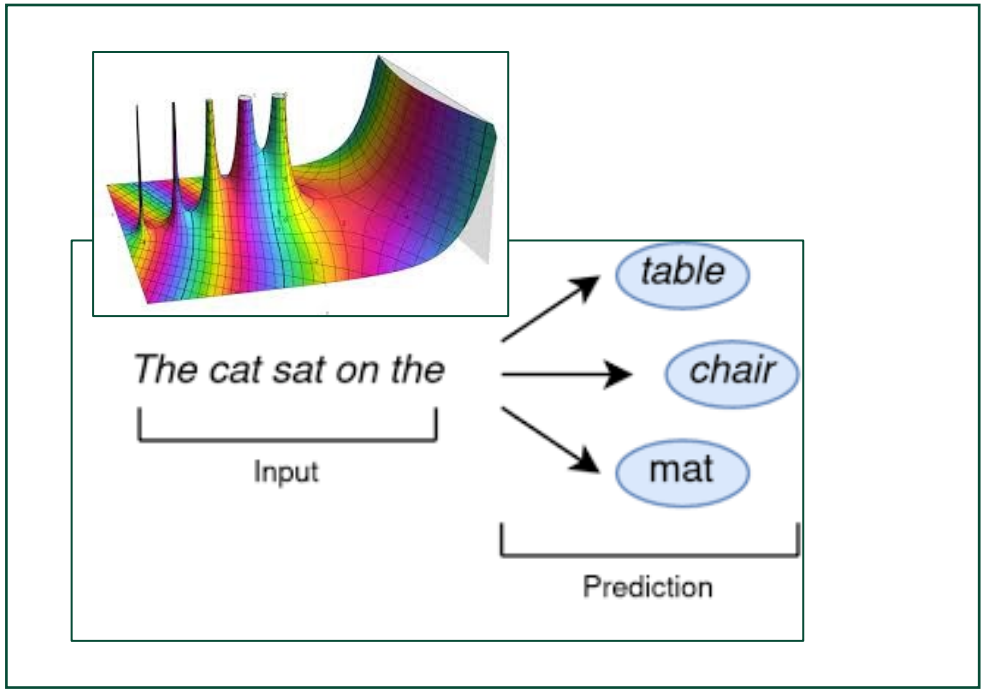
Deep Learning

Diffusion, etc.

ChatGPT

Transformers, etc.

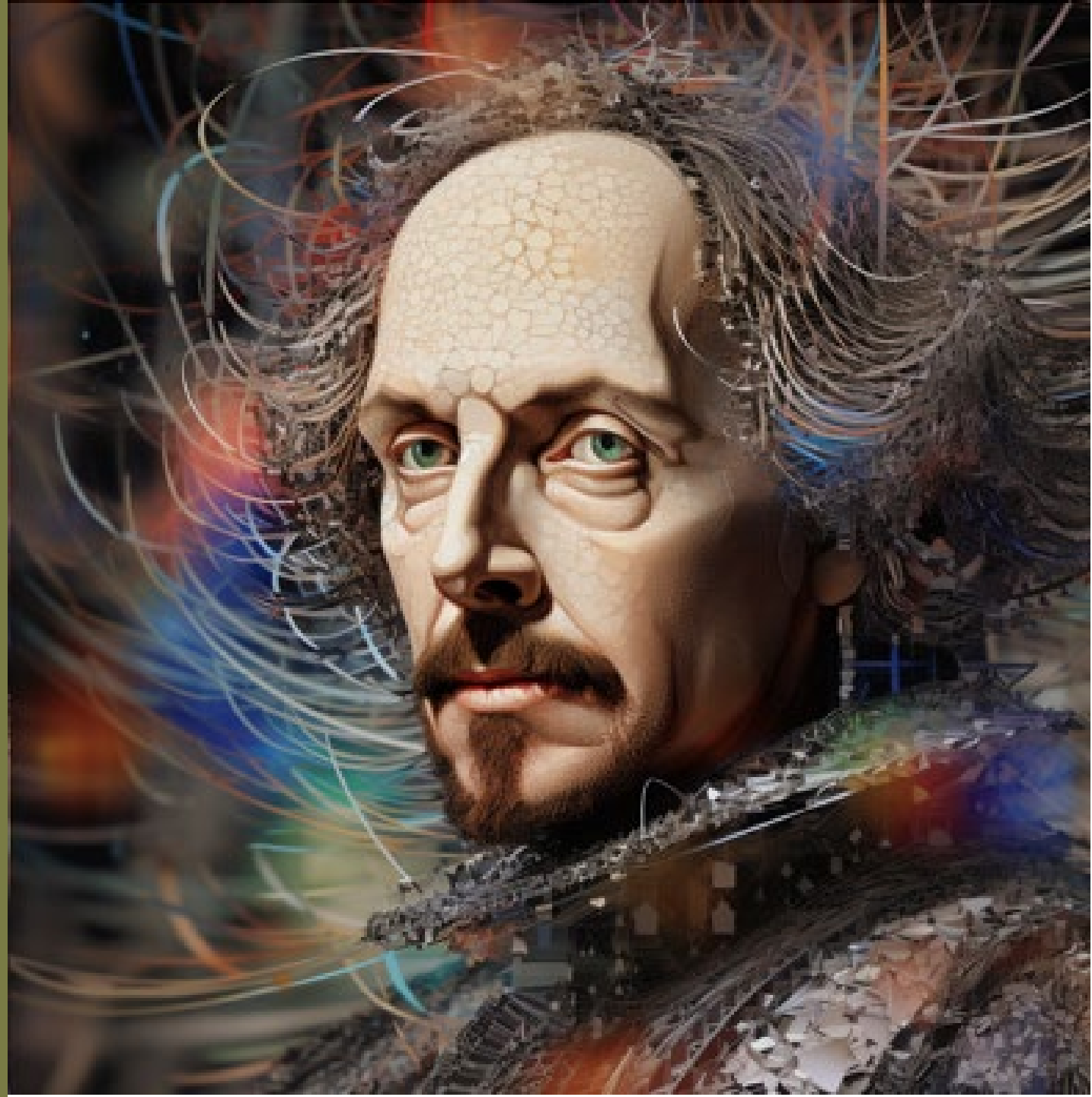




We are going to build a Shakespeare GPT using a transformer model.

It will learn from scratch from the complete works of Shakespeare.

It will emit an eternal stream of Shakespeare



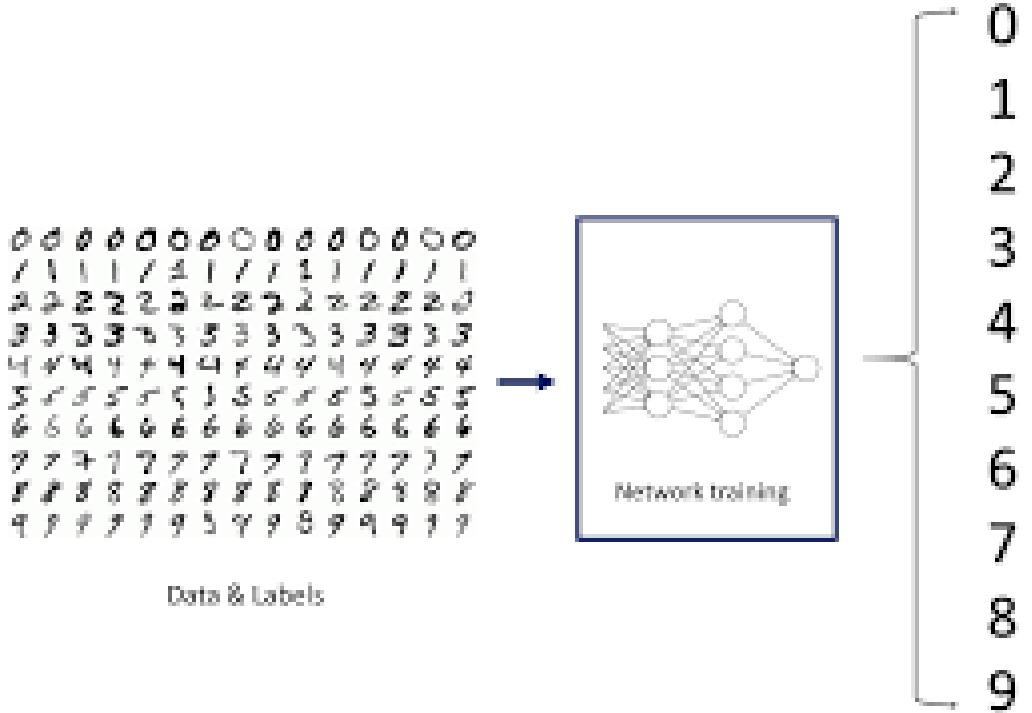
REVIEW

- Recap from Parts 1-4
  - Machine Learning Basics
  - Neural Networks
- Tensors
- GPUs and CUDA
- PyTorch

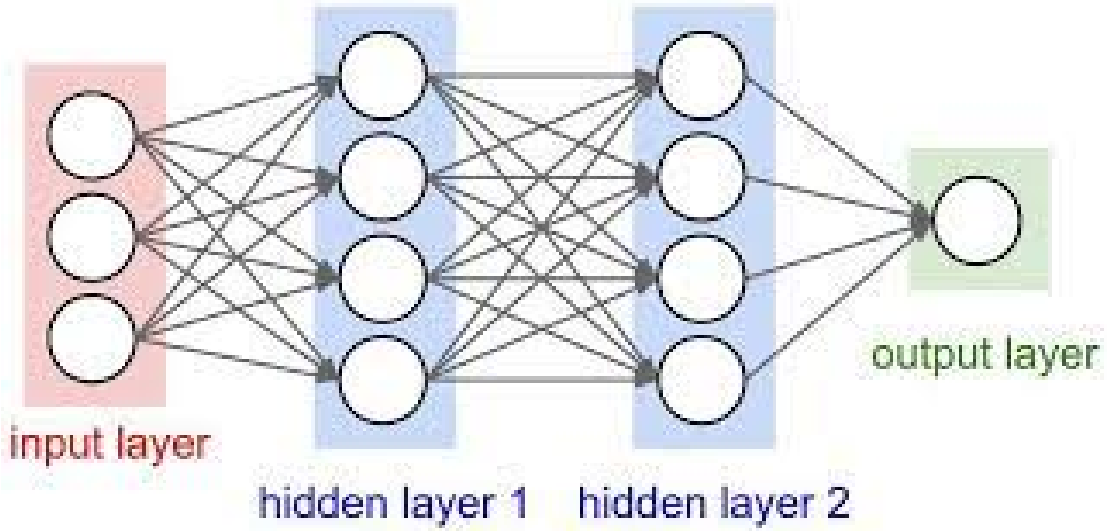
REVIEW OF BASICS

- Machine learning is a data-driven method for creating models for prediction, optimization, classification, generation, and more
- Python and scikit-learn, TensorFlow, PyTorch
- MNIST
- Artificial Neural Networks (ANNs)

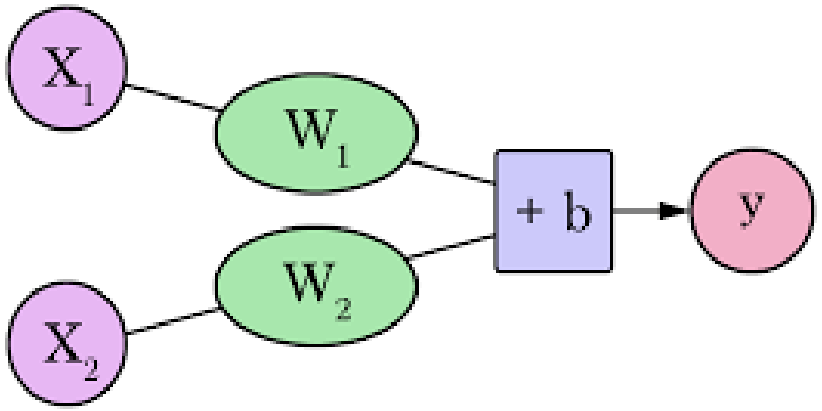
MNIST



NEURAL NETWORK BASICS




```python
from sklearn.neural_network import MLPClassifier

# Instantiate a Neural Network and train our model
nn = MLPClassifier(hidden_layer_sizes=(50,25),
                   max_iter=50,
                   n_iter_no_change=50,
                   activation = 'relu',
                   solver='adam',
                   random_state=42,
                   verbose=True)
nn.fit(X_train, y_train)
```

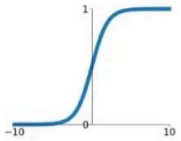


Weights and Biases

Activation Functions

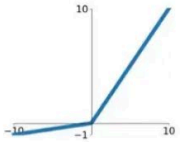
Sigmoid

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



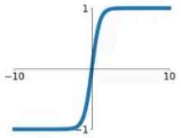
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

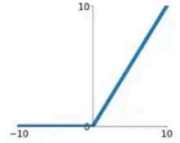


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

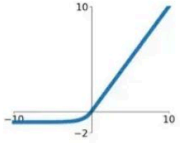
ReLU

$\max(0, x)$

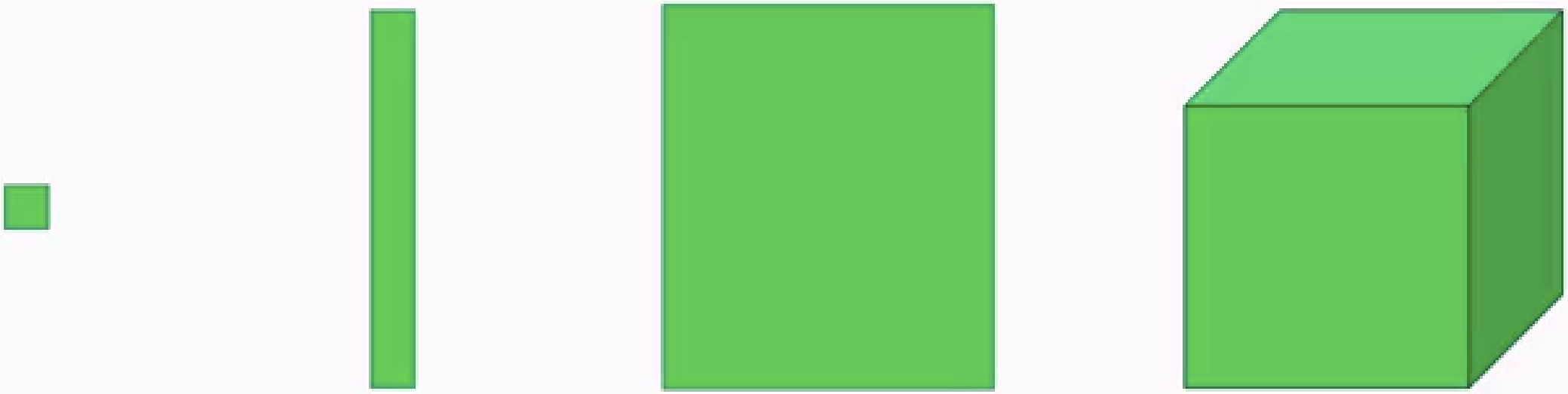


ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



A tensor is an N-dimensional array of data



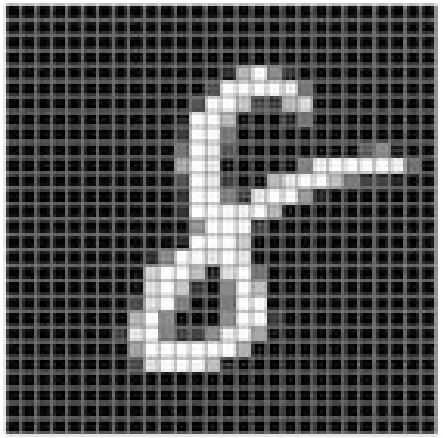
Rank 0
Tensor
scalar

Rank 1
Tensor
vector

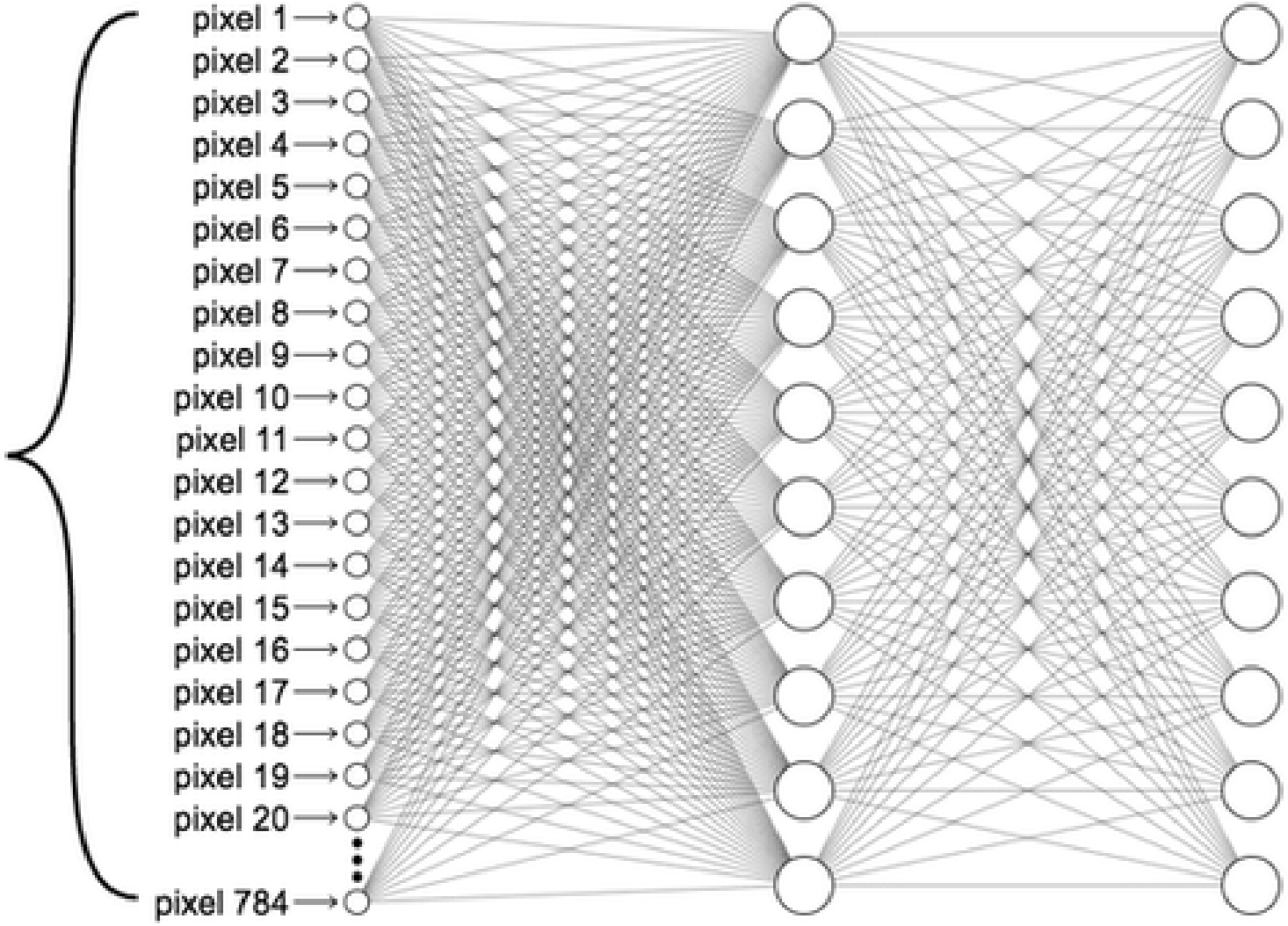
Rank 2
Tensor
matrix

Rank 3
Tensor

FULLY-CONNECTED NEURAL NETWORKS

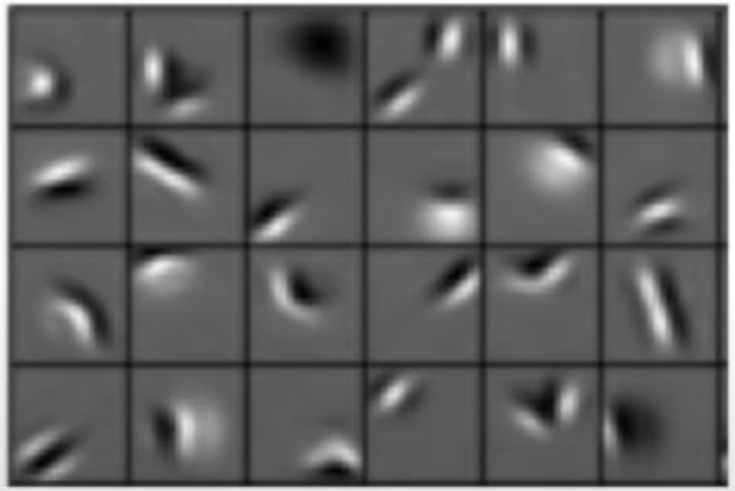


Images are tensors!



FEATURE HIERARCHIES

Low level features



Edges, dark spots

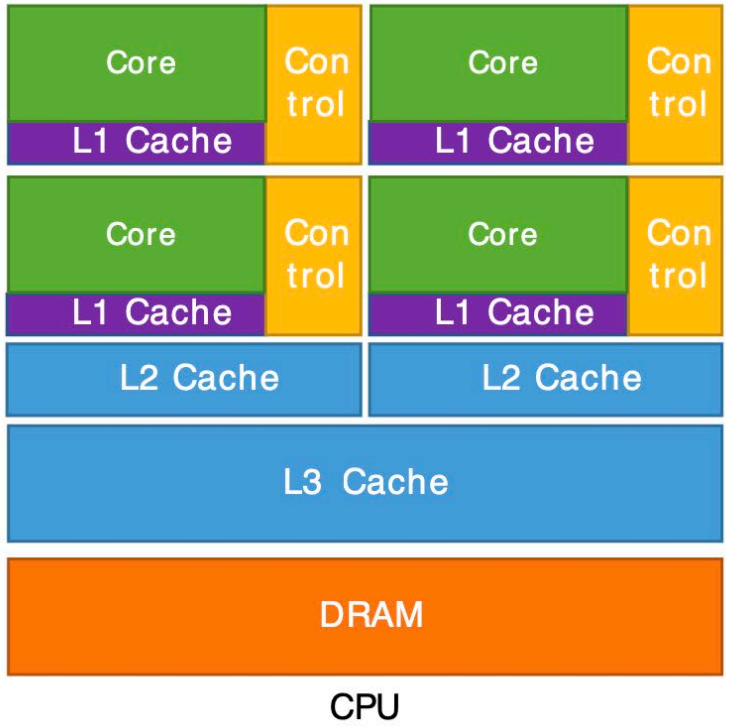
Mid level features

Eyes, ears, nose

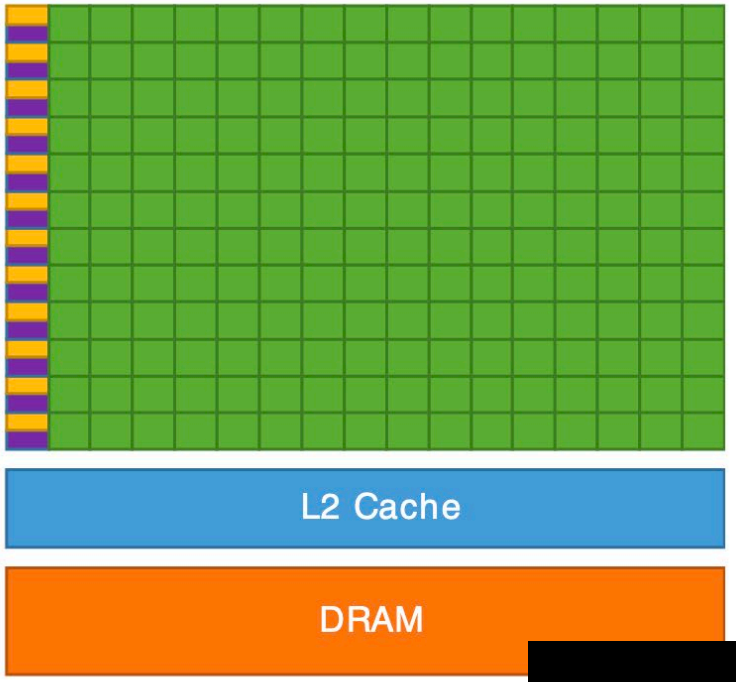
High level features

Facial structure

We need image *filters* to help us extract features



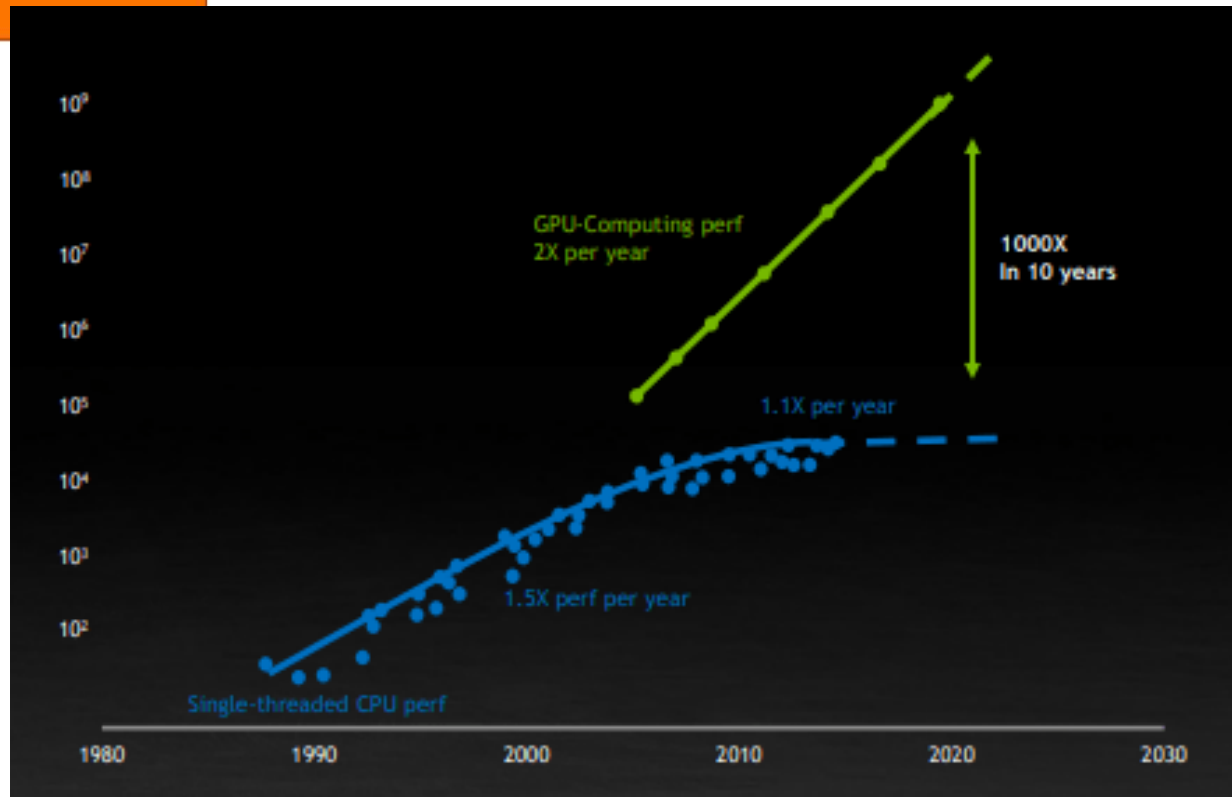
CPU



GPU

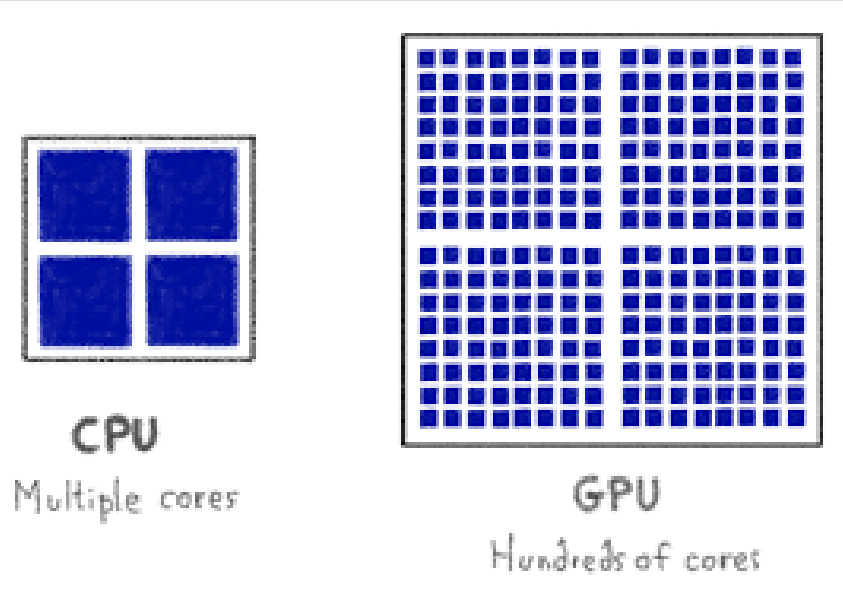
GPU vs. CPU

"Moore's Law for CPUs is Dead"



WHY GPUS EXACTLY?

- CNNs are all about matrix and vector operations (multiplication, addition)
- GPUs can perform parallel multiplication and addition steps per each clock cycle.



Frameworks make GPUs Easy

TensorFlow

NVIDIA CUDA

PyTorch

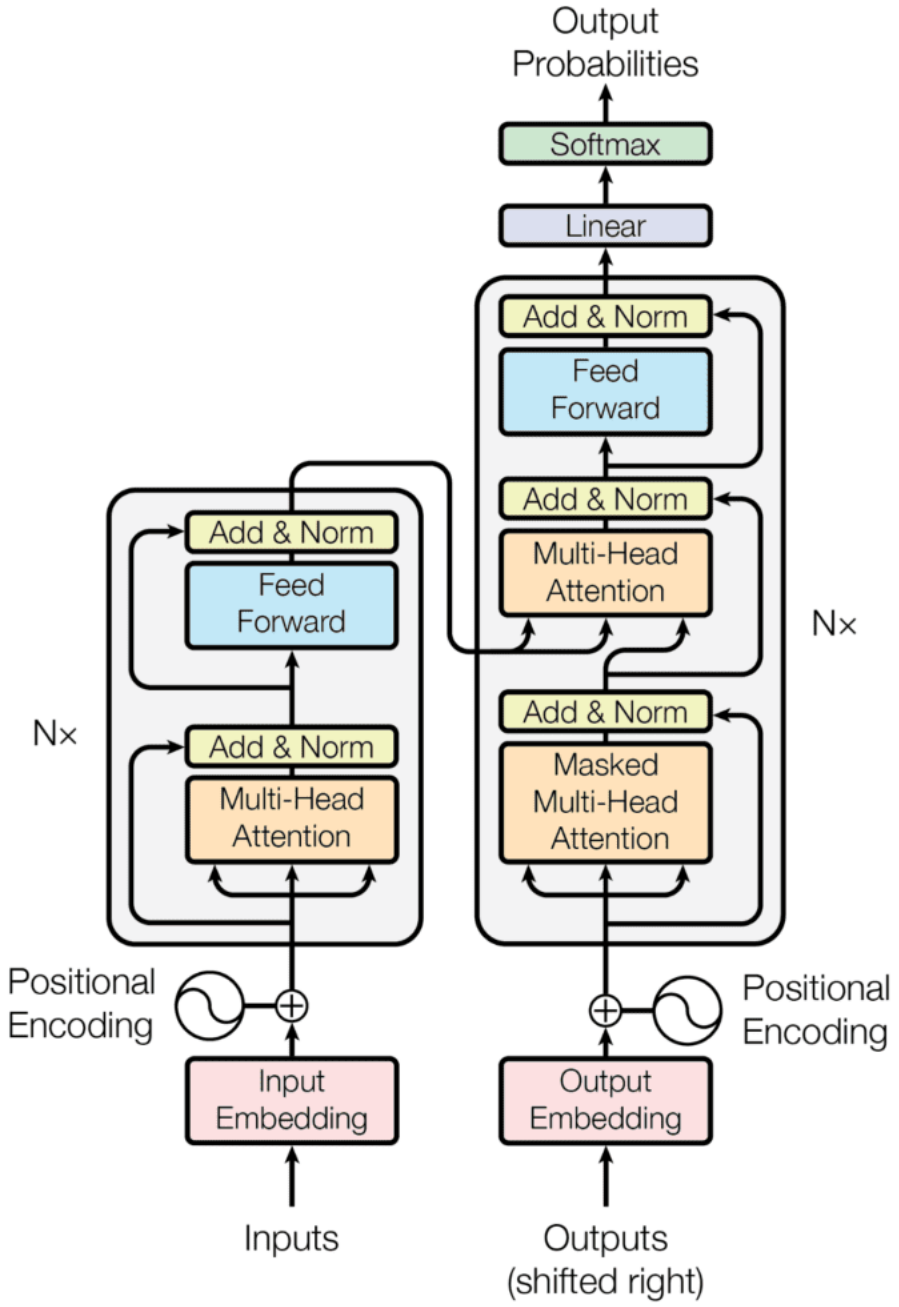
M1

M2

PyTorch

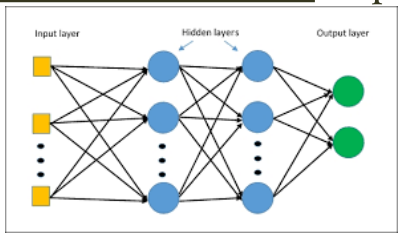
PyTorch | AMD INSTINCT

TRANSFORMER MODELS

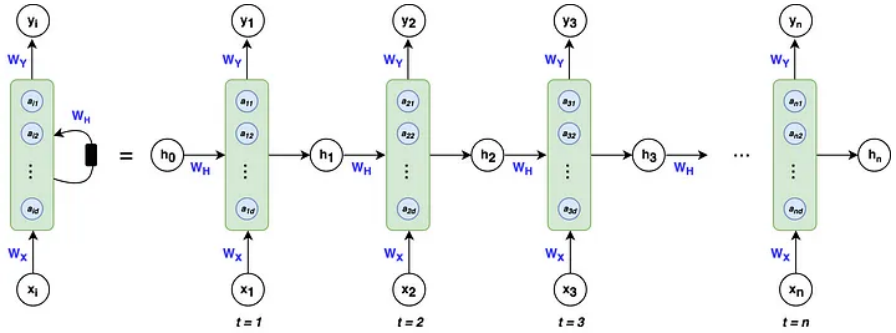


BACK IN THE OLDEN DAYS (*5 YEARS AGO*)…

- Sophisticated language models are not possible with simpler, traditional neural networks.
- **Multi-level Perceptrons (MLPs):** sequential information is lost (forgotten).

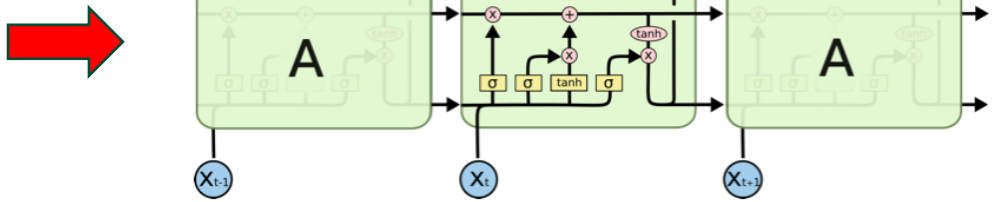


- **Recurrent Neural Networks (RNNs):** *They* can work with sequential data! Yay!



But… * Training is slow and sequential.
* Vanishing gradients limit context length

- **Long Short-Term Memory (LSTMs):** A form of RNN with improved memory…but still slow to train without parallelism.



"*Old man tinkering with RNNs*" - *Midjourney*

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

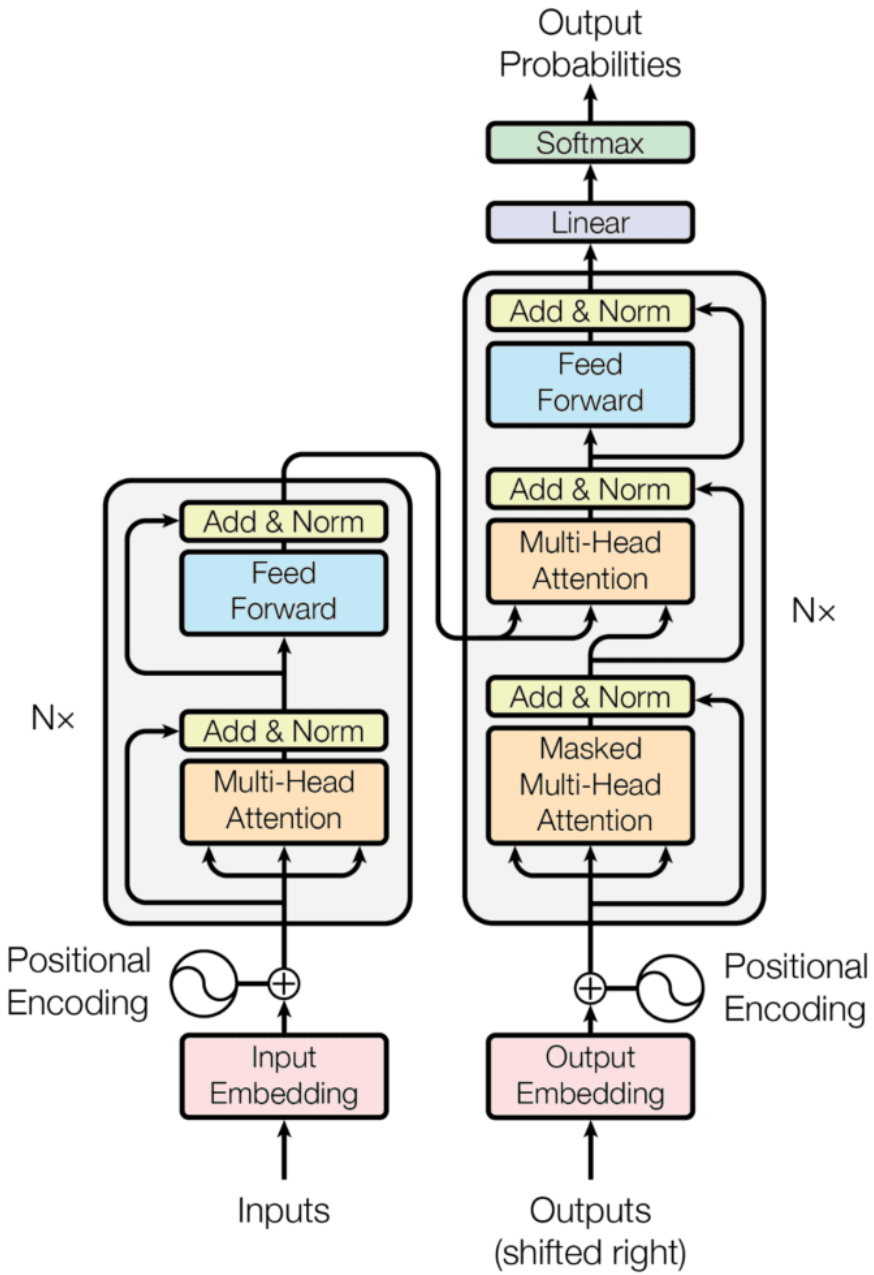
1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

In 2017, everything changed.

TRANSFORMER MODELS

- Works only on sequential data
- Uses a new method called “Attention” or ”Self-Attention”
- Replacing CNNs and RNNs across many data modalities
  - (text, images, video, audio)
- Fast to train (*very parallelizable*)



Transformer Architecture

USES OF TRANSFORMERS

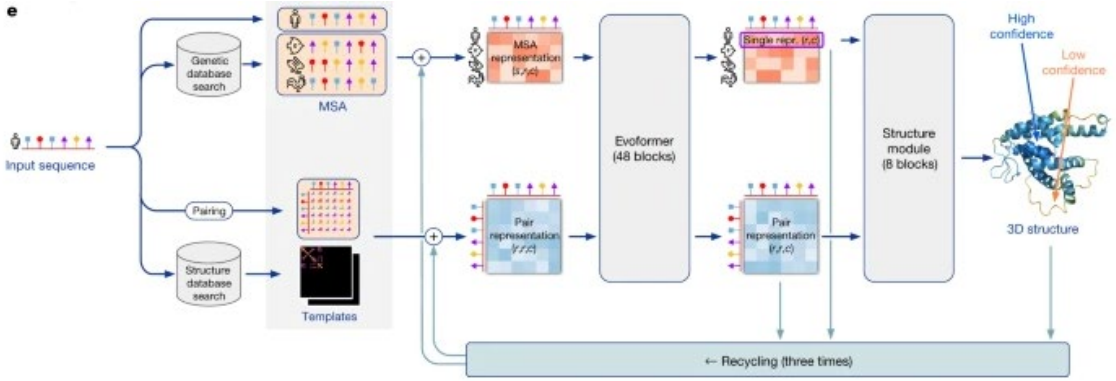
- Large Language Models
- Science and Engineering
- Audio and Speech
- Computer Vision

CHATGPT

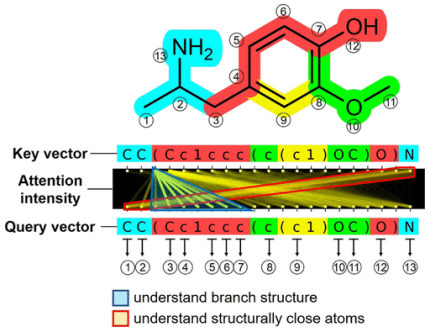
Bard

ANTHROPIC CLAUDE 2

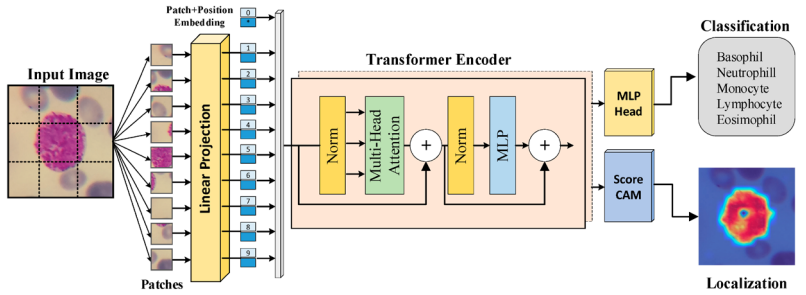
Meta LLaMA



AlphaFold uses transformers for protein folding



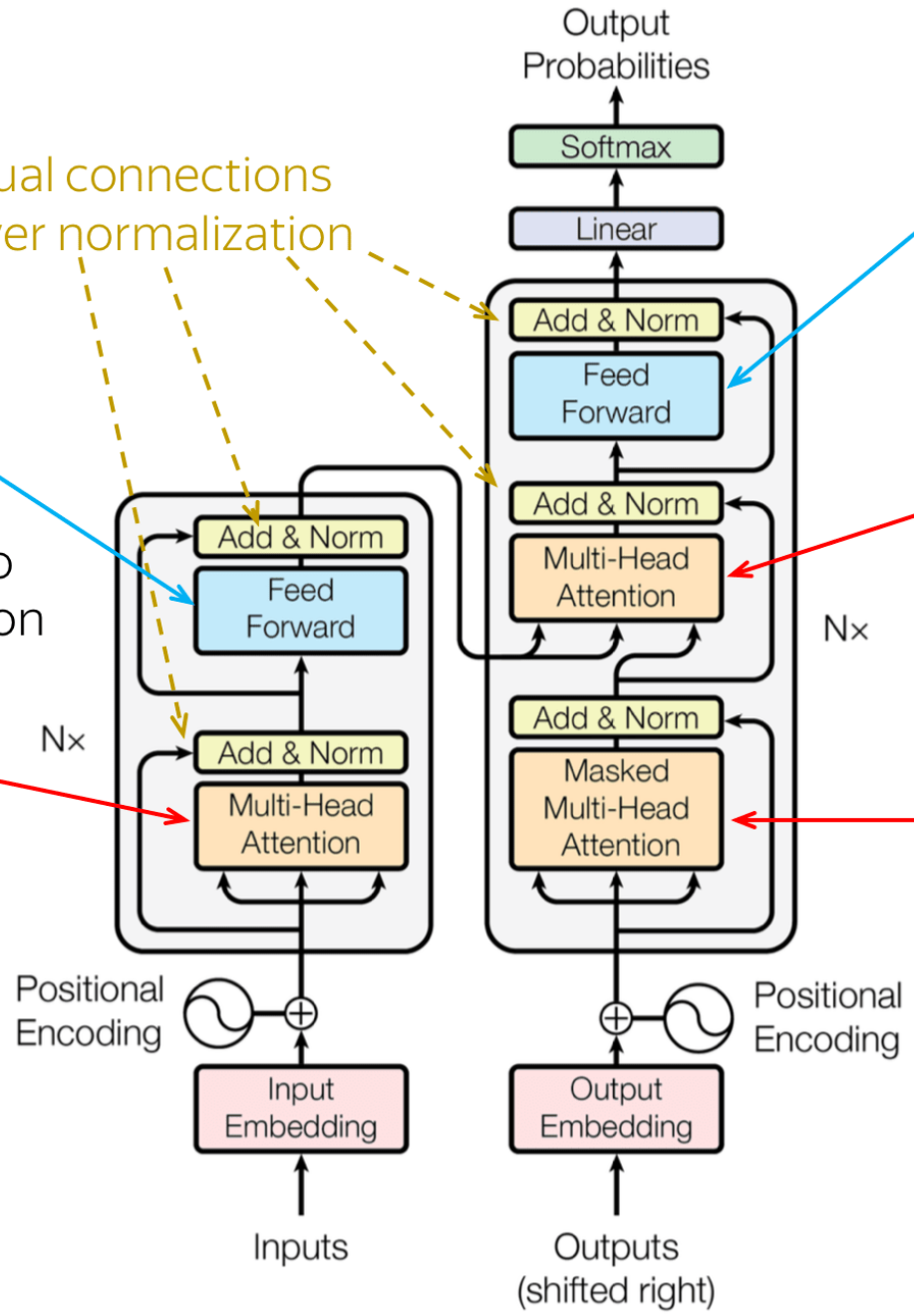
Chemistry



Feed-forward network:
after taking information from other tokens, take a moment to think and process this information

Encoder self-attention:
tokens look at each other

queries, keys, values are computed from encoder states



Feed-forward network:
after taking information from other tokens, take a moment to think and process this information

Decoder-encoder attention:
target token looks at the source

queries – from decoder states; keys and values from encoder states

Decoder self-attention (masked):
tokens look at the previous tokens

queries, keys, values are computed from decoder states

DECODER-ONLY TRANSFORMERS

Look at current text sequence

Predict the next word

Add word to our current text sequence

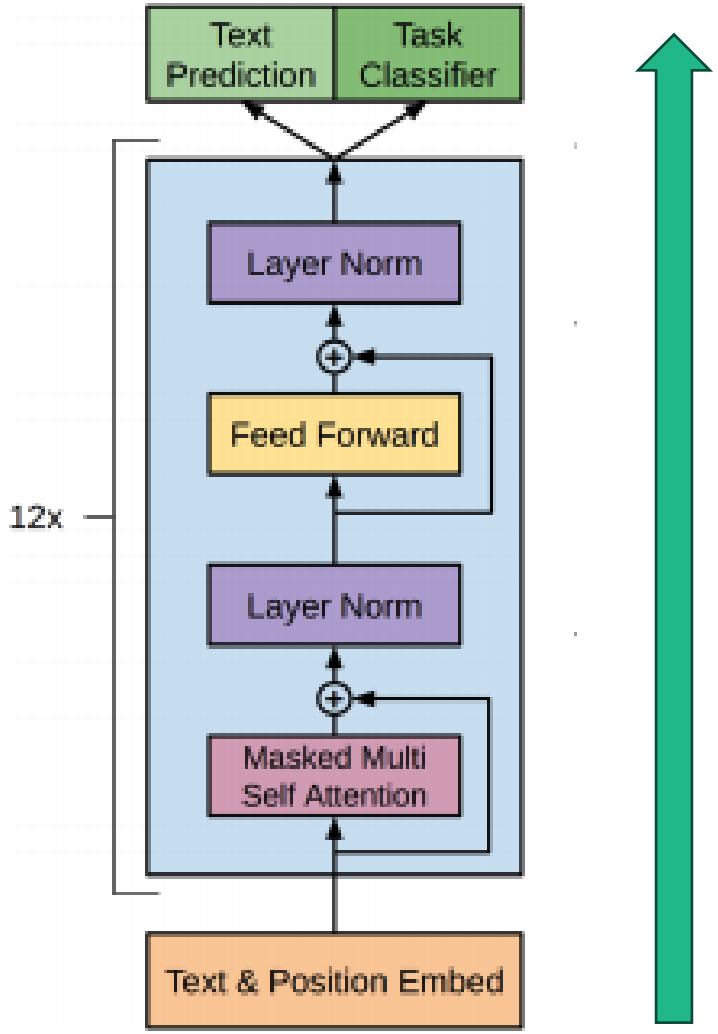
Repeat

No, really, it predicts next tokens.

And that is all. (*But that is a lot!*)

PARTS OF A DECODER-ONLY TRANSFORMER

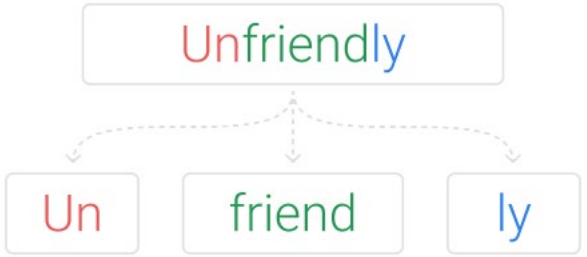
- Tokenizer
- Embedding Layer
- Positional Encoding
- Decoder Blocks
  - Self-Attention
  - Multiple-Head Attention
  - Feed-Forward Neural Network
  - Normalization
- Output Layer:
- Linear layer + Softmax



**ChatGPT is Decoder-Only*

TOKENIZER

- Neural networks work with numbers, so let's convert text to numbers first.
- Tokenize at character, sub-word, word, or partial sentence levels.
- Most language models use subword tokens



n = 50,257 subwords for GPT-2

- We're going to go with character-level tokens

  Vocabulary Size = about 70 tokens

  Trivial tokenizer! (Just a lookup table.)

```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Example of tokenizing text
text = "Hello, world!"
encoded_input = tokenizer(text)
print(encoded_input)
```

Popular Pre-Trained Tokenizers

BERT from Google
BART and RoBERTa from Meta
GPT Tokenizers from OpenAI

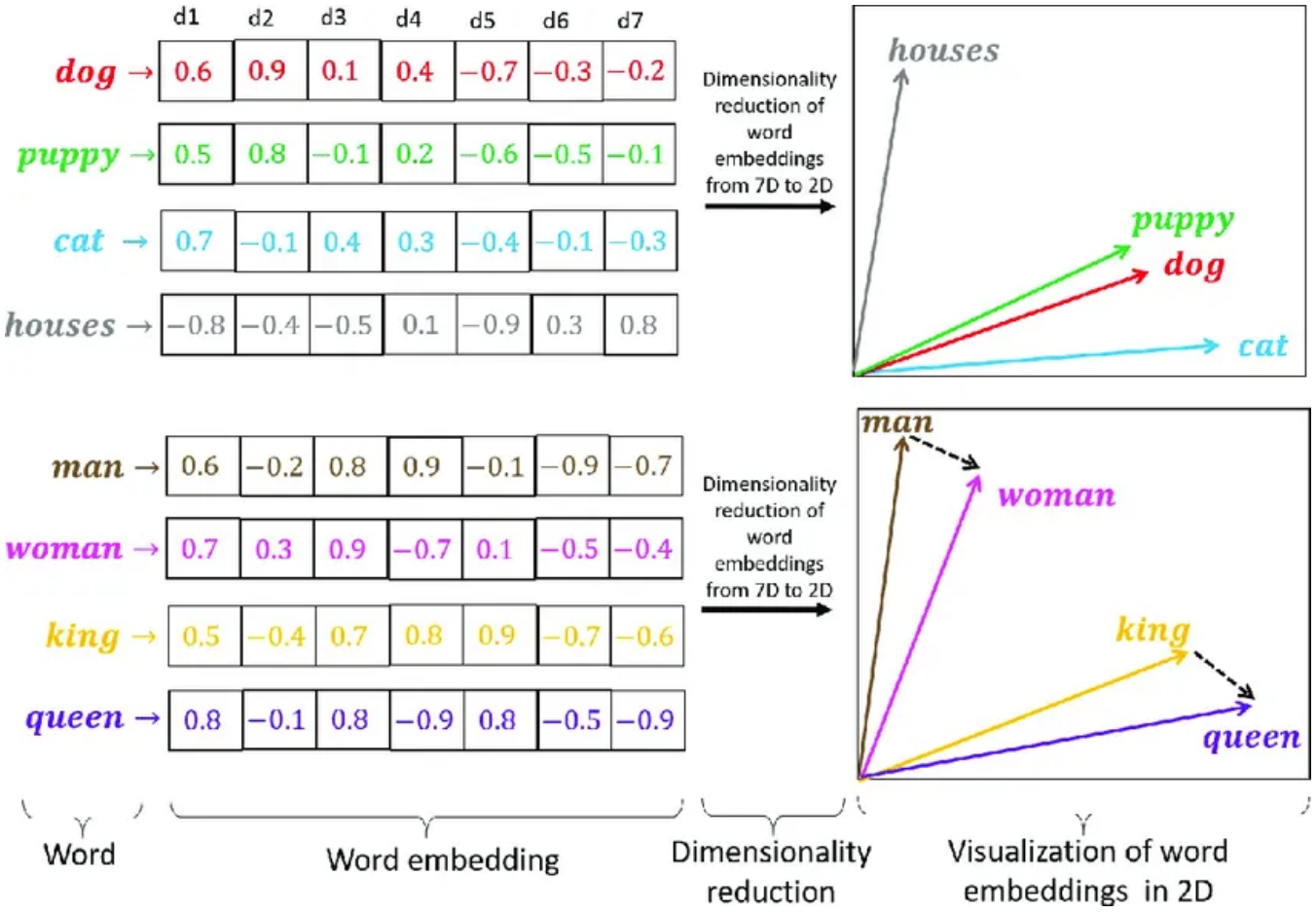
TOKEN EMBEDDINGS

Vector = amplitude and direction

Token String	Token ID	Embedded Token Vector
'<s>'	0	[0.1150, -0.1438, 0.0555, ...]
'<pad>'	1	[0.1149, -0.1438, 0.0547, ...]
'</s>'	2	[0.0010, -0.0922, 0.1025, ...]
'<unk>'	3	[0.1149, -0.1439, 0.0548, ...]
'.'	4	[-0.0651, -0.0622, -0.0002, ...]
' the'	5	[-0.0340, 0.0068, -0.0844, ...]
','	6	[0.0483, -0.0214, -0.0927, ...]
' to'	7	[-0.0439, 0.0201, 0.0189, ...]
' and'	8	[0.0523, -0.0208, -0.0254, ...]
' of'	9	[-0.0732, 0.0070, -0.0286, ...]
' a'	10	[-0.0194, 0.0302, -0.0838, ...]
		...

EMBEDDING LAYER

- It is a 2D tensor of shape: *vocabulary_size* **X** *embedding dimension*
- OpenAI GPT-3 (Davinci) embedding layer is: 50,257 **X** 12,888



Vector embeddings capture the deeper semantic context of a word or text chunk…or image…or anything.

The semantics of an object are defined by its multi-dimensional and multi-scale co-occurrence and relationships with other objects in the training data

Semantic vector embeddings are learned from vast amounts of data.

EMBEDDING LAYER

A compressed, lossy version of the entire training corpus.

Gestalt representation of human language, concepts, facts, and more as scraped off the internet.

Semantics expressed only as hyperdimensional relationships between tokens.

GPT3:

Internet compressed to 2.6GB spreadsheet

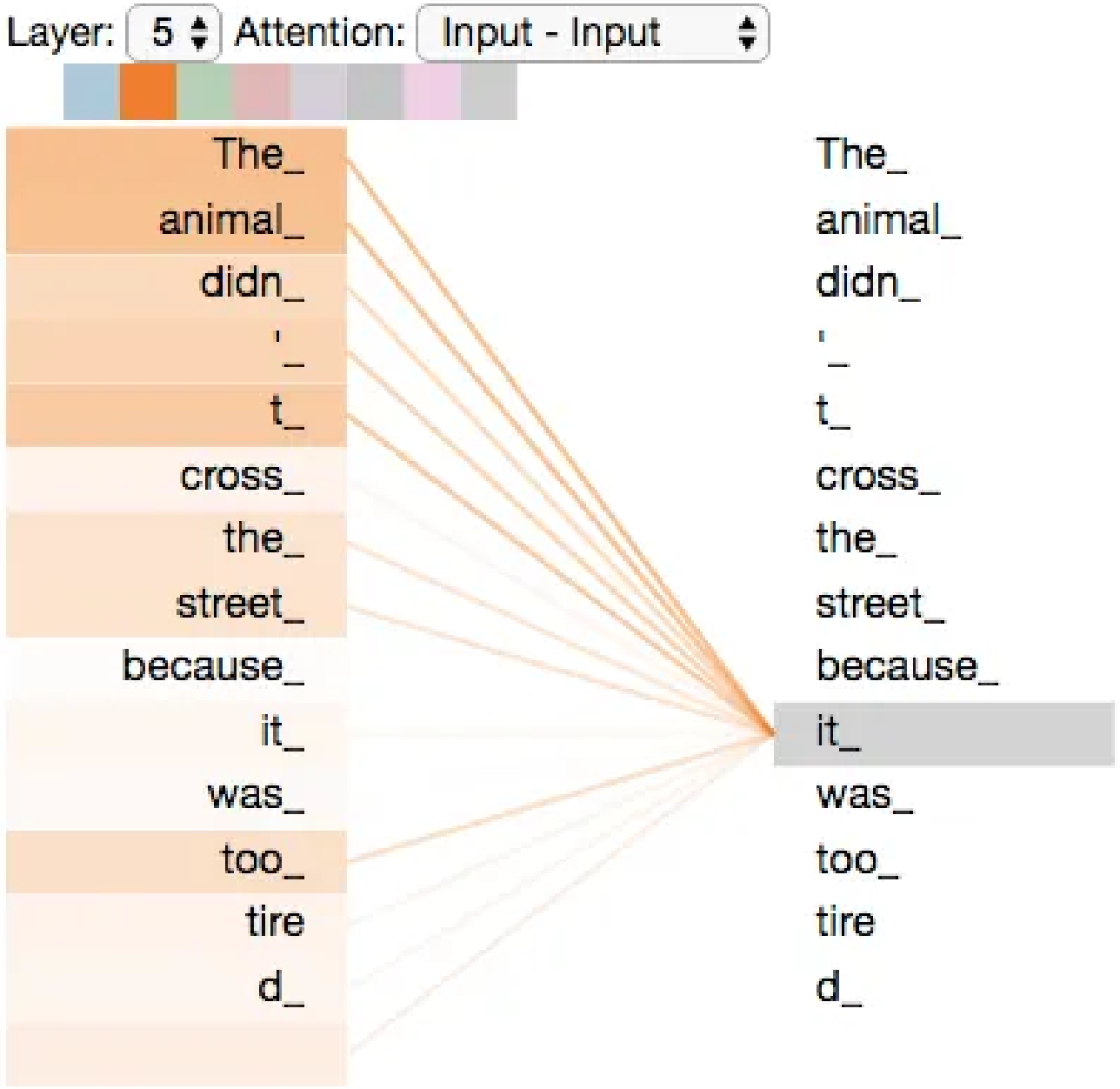
The semantic embedding layer is what the transformer operates on.

Token embeddings are *learned* as the LLM trains on next token prediction.

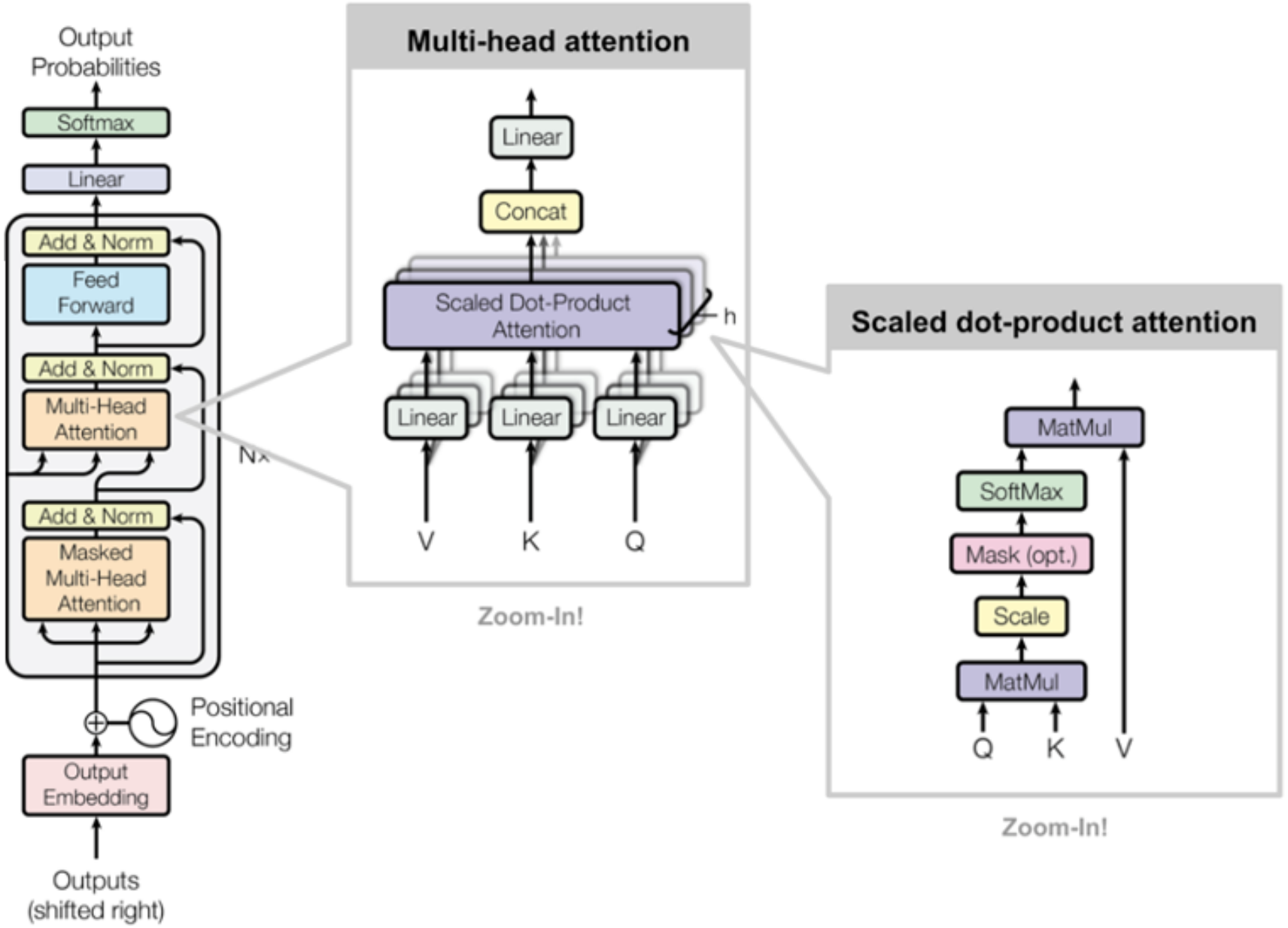
OUR AI SHAKESPEARE EMBEDDING LAYER

- Vocab Size = 107 unique characters
- Embedding Size = 64
- Number of Layers = 6
- Number of Attention Heads per Layer: 8

SELF-ATTENTION



SELF-ATTENTION HEADS



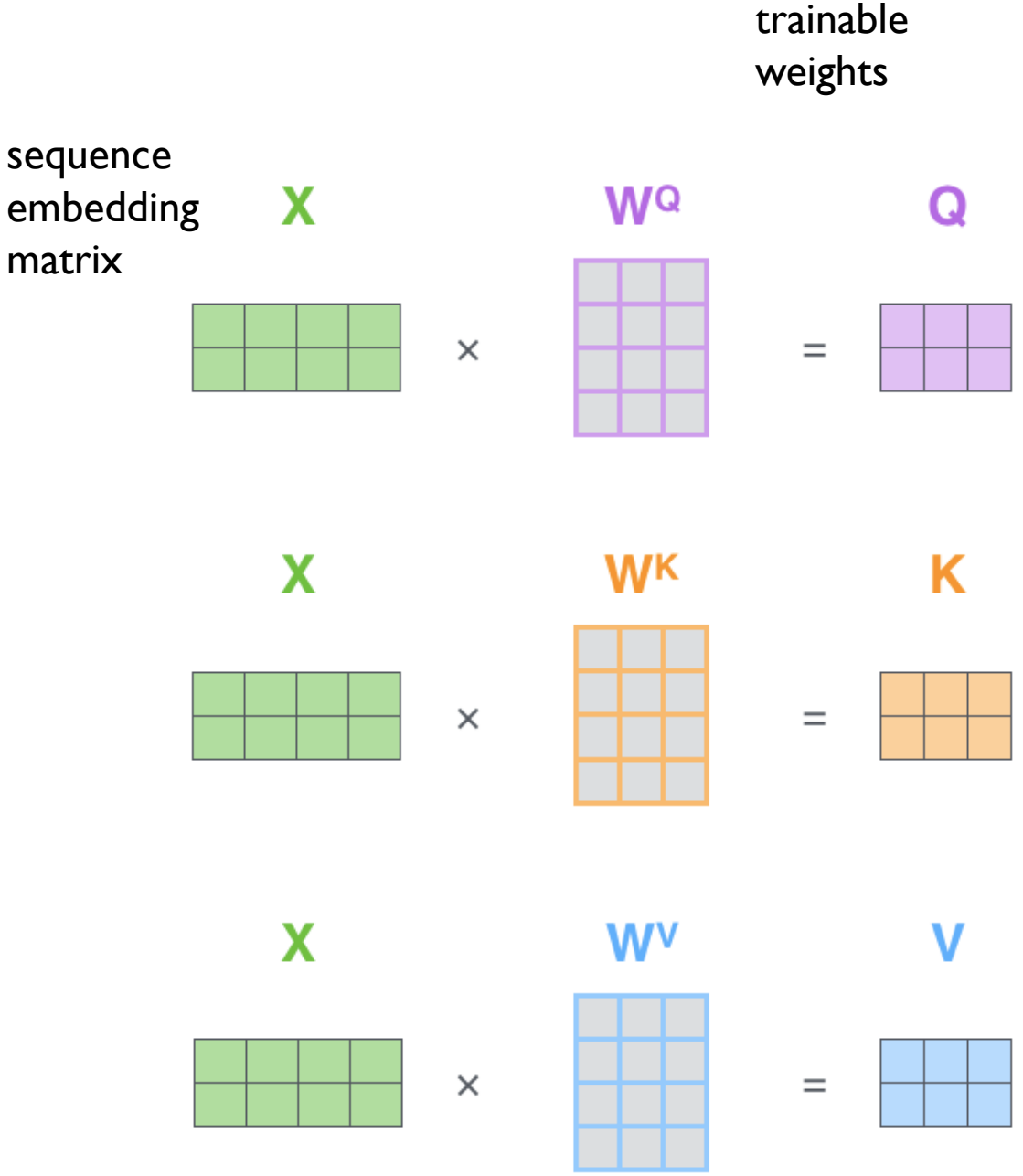
We used multiple heads

Each head focuses on a different semantic or contextual aspect of the input sequence (i.e. context window)

The heads learn what to focus on to optimize the networks

SELF-ATTENTION HEAD

- For every token in the input sequence (context window) compute three vectors:
  - **Query (Q)** – A vectorized encoding that captures how much each token in the sequence should be attended to relative to the current token.
  - **Key (K)** – A vectorized encoding used to represent a scoring of each token's query.
  - **Value (V)** – A vectorized representation of the actual content of the token.



SELF-ATTENTION SCORE CALCULATION

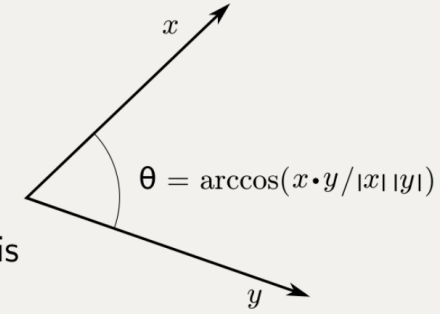
- For every token, compute the dot product between this token's Q vector and all other K vectors.
- Attention Score = Q * K^T
- The result is an n X n matrix where n = sequence length (context window size)
- Scale by sqrt(len(K))
- Softmax() -> attention ***weights***
- Output = Attention Weights * V vector

Dot product

Algebraic definition :

Taking the vectors $\mathbf{a}=[a_1, a_2, \ldots, a_n]$ and $\mathbf{b}=[b_1, b_2, \ldots, b_n]$ with vector space n, the dot product is

$$\mathbf{a}\cdot\mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

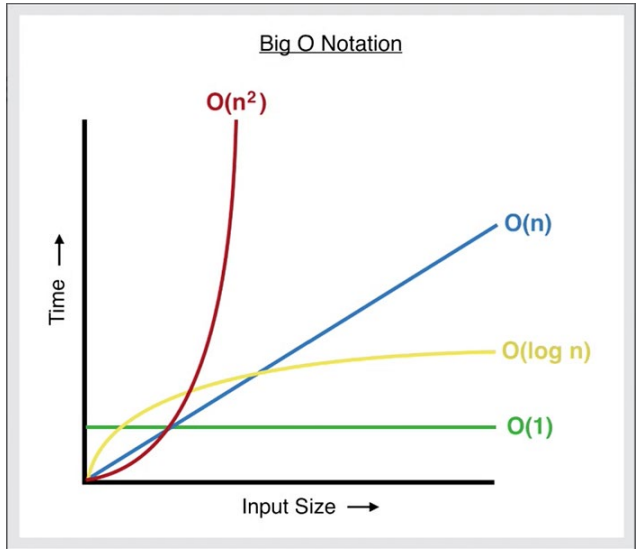


Geometric definition :

The dot product of two Euclidean vectors **a** and **b** is

$$\mathbf{a}\cdot\mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\|\cos\theta,$$

where θ is the angle between vectors and $\|\mathbf{a}\|$ and $\|\mathbf{b}\|$ their respective magnitudes

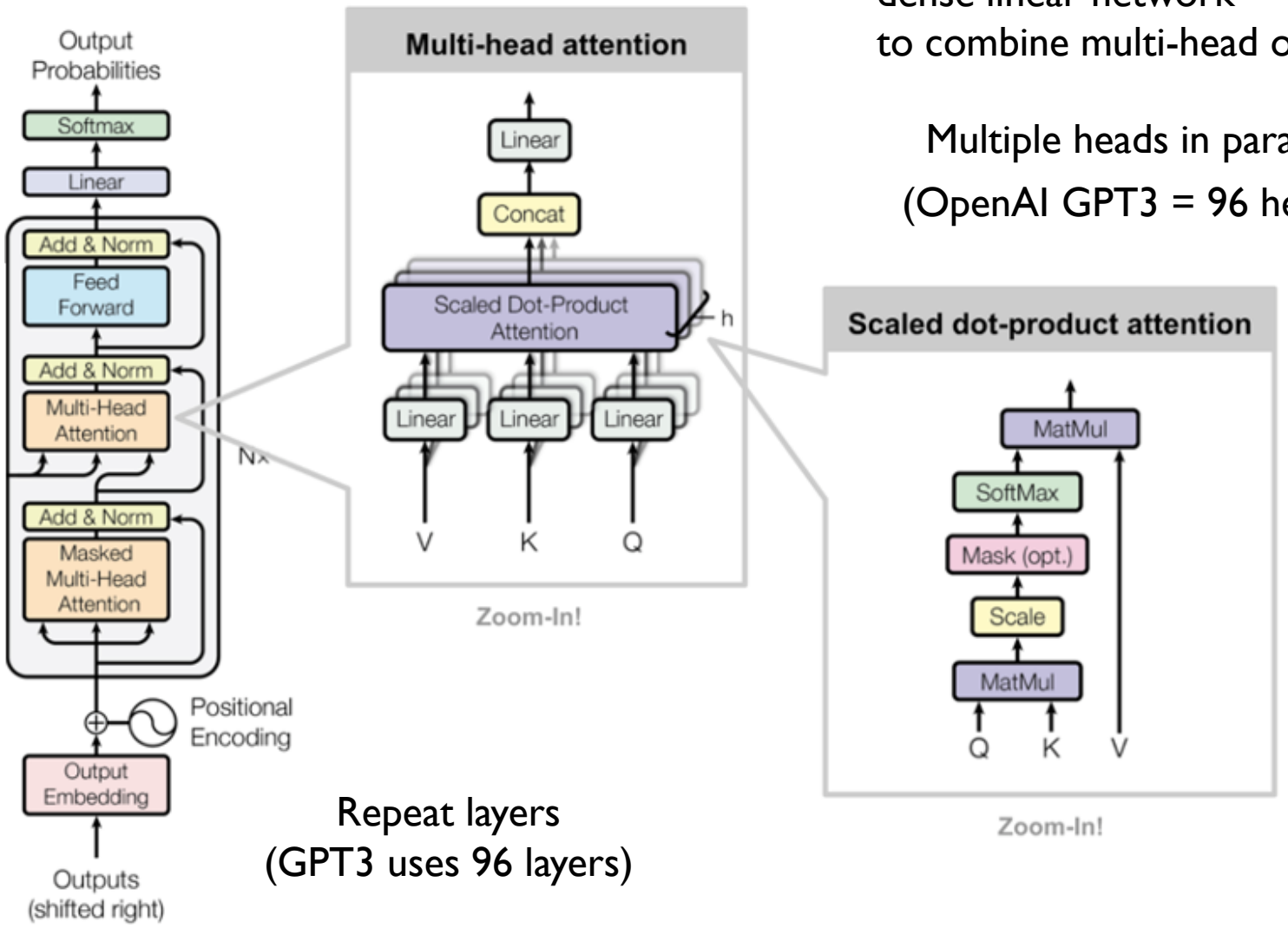


Pairwise dot products are $O(n^2)$

Transformers are $O(n^2)$

n = size of context window

SELF-ATTENTION SCORES – PART 2



dense linear network
to combine multi-head output

Multiple heads in parallel

(OpenAI GPT3 = 96 heads)

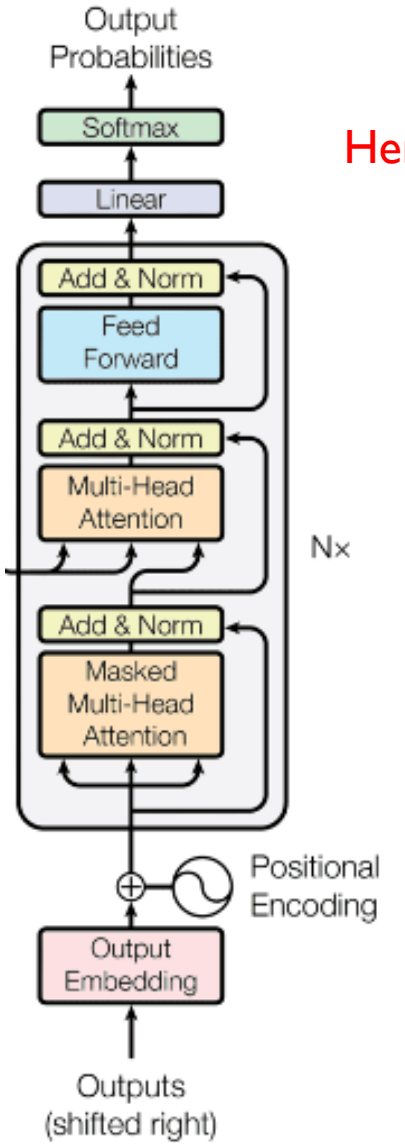
Repeat layers
(GPT3 uses 96 layers)

All dot products and self-attention scores for a given transformer block can be computed in parallel. (*yay GPUs!*)

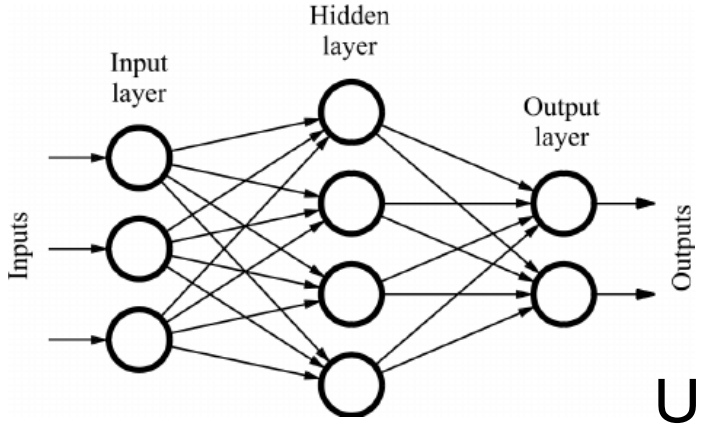
Successive transformer block layers must be sequential

FEED FORWARD NETWORK (FFN) LAYER

Output tensor
(***batch, sequence_len, vocab_size***)



Here!



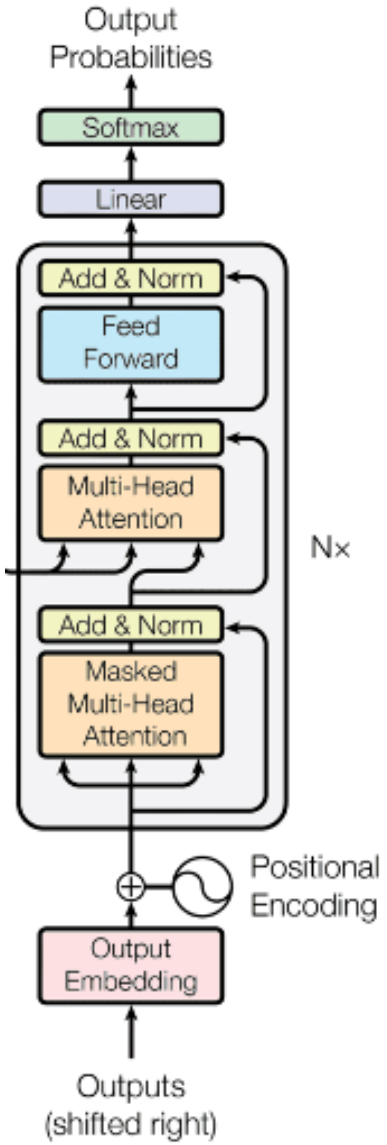
expansion projection

Unlike self-attention scoring, operates on each position separately

Adds representation capability for each token in the sequence (context window) independently

Allows network to make complex transformations on the sequence data after self-attention scores are computed

POSITIONAL ENCODING!



Almost Forgot!!

Transformers work on sequences BUT they don't track sequential order (attention scores are calculated in parallel!)

Sequential order of tokens is pretty ***important*** in language ☺

We need to tell the transformer about the order of words

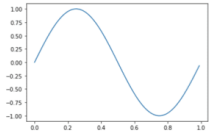
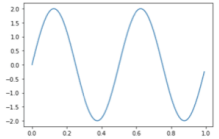
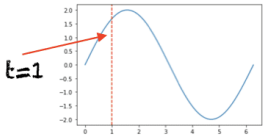
POSITIONAL ENCODING

We obviously need another tensor!

Sequence	Index of token	Positional Encoding Matrix			
I	0	P_{00}	P_{01}	…	P_{0d}
am	1	P_{10}	P_{11}	…	P_{1d}
a	2	P_{20}	P_{21}	…	P_{2d}
Robot	3	P_{30}	P_{31}	…	P_{3d}

Positional Encoding Matrix for the sequence 'I am a robot'

Sinusoidal Position Encoding

Equation	Graph	Frequency	Wavelength
$\sin(2\pi t)$		1	1
$\sin(2 * 2\pi t)$		2	1/2
$\sin(t)$		$1/2\pi$	2π
$\sin(ct)$	Depends on c	$c/2\pi$	$2\pi/c$

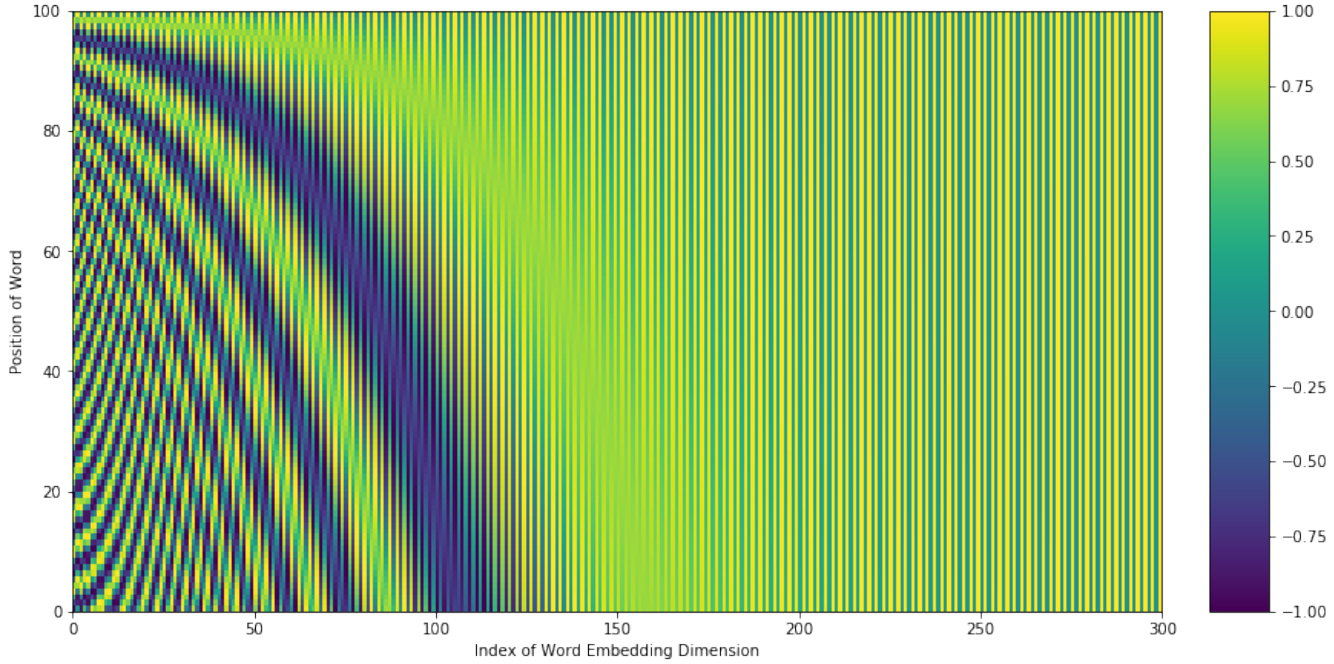
SINUSOIDAL POSITIONAL ENCODING

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Even positions are sin()

Odd positions are cos()



+

Token String	Token ID	Embedded Token Vector
'<s>'	0	[0.1150, -0.1438, 0.0555, ...]
'<pad>'	1	[0.1149, -0.1438, 0.0547, ...]
'</s>'	2	[0.0010, -0.0922, 0.1025, ...]
'<unk>'	3	[0.1149, -0.1439, 0.0548, ...]
'.'	4	[-0.0651, -0.0622, -0.0002, ...]
' the'	5	[-0.0340, 0.0068, -0.0844, ...]
','	6	[0.0483, -0.0214, -0.0927, ...]
' to'	7	[-0.0439, 0.0201, 0.0189, ...]
' and'	8	[0.0523, -0.0208, -0.0254, ...]
' of'	9	[-0.0732, 0.0070, -0.0286, ...]
' a'	10	[-0.0194, 0.0302, -0.0838, ...]

UMMMM…

- *If I add these sinusoidal waveforms to the existing values in my token embeddings for my current sequence, doesn't that change the semantic meaning of my tokens????*
- Absolutely. And that's okay.
- A token's semantic meaning DOES change depending on its relative position within a sequence.
- Relax. *The transformer will figure this all out during training.*

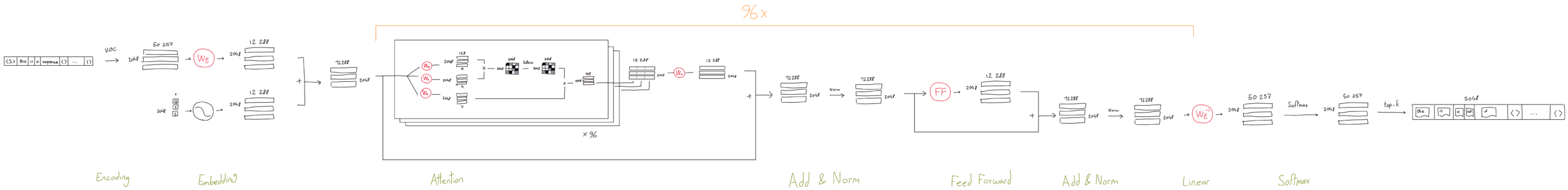
Luke destroyed the Death Star

≠

The Death Star destroyed Luke

The Entire GPT3-175B (Davinci) Transformer Architecture on a Napkin

175 Billion Trainable Parameters



GPT-3 training data[1]:9

Dataset	# tokens	Proportion within training
Common Crawl	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

COMMON CRAWL

OpenAI

Training GPT3 Davinci:

$4.6 Million in Azure
335 GPU Years (V100)

Today: ~$500K

A SHORT WORD ABOUT CLIP

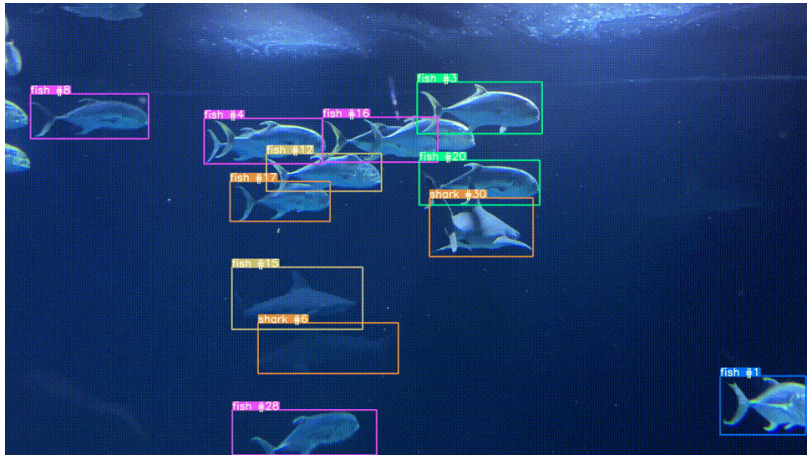
USING CLIP IS TRIVIAL

```python
import torch
import clip
from PIL import Image

device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device)

image = preprocess(Image.open("CLIP.png")).unsqueeze(0).to(device)
text = clip.tokenize(["a diagram", "a dog", "a cat"]).to(device)

with torch.no_grad():
    image_features = model.encode_image(image)
    text_features = model.encode_text(text)

    logits_per_image, logits_per_text = model(image, text)
    probs = logits_per_image.softmax(dim=-1).cpu().numpy()

print("Label probs:", probs)  # prints: [[0.9927937  0.00421068 0.00299572]]
```

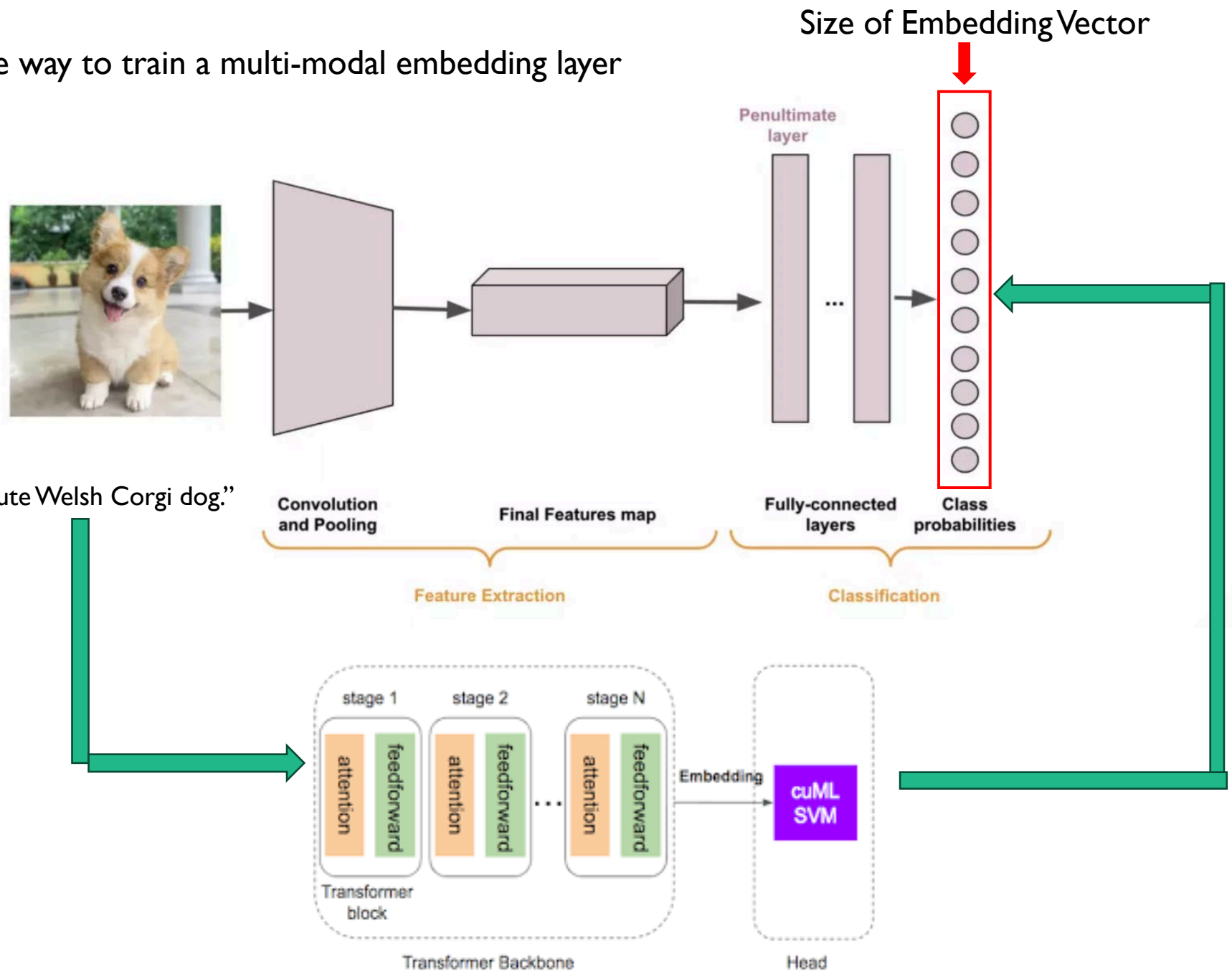
https://github.com/openai/CLIP

Zero-shot classifications!
Conditioning Generative AI (DALL-E)
Generating captions for images or video
Image similarity search
Content Moderation
Object Tracking

A couple of people standing next to an elephant.

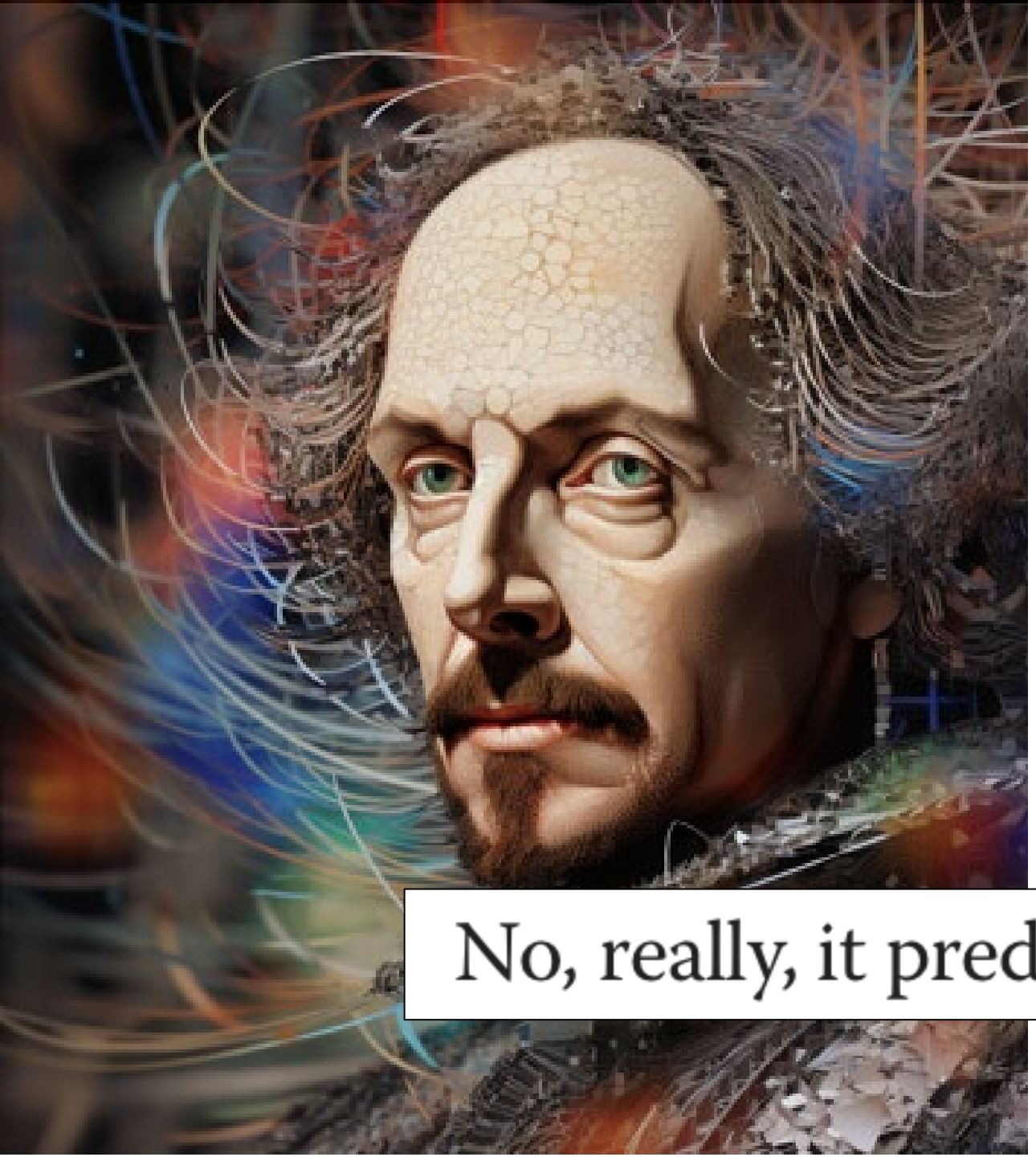


One way to train a multi-modal embedding layer



Learning your semantic embeddings first (GPT)

Train your CNN to predict matching embedding vectors



NOW FOR CODE

github.com/sheneman/shakeGPT

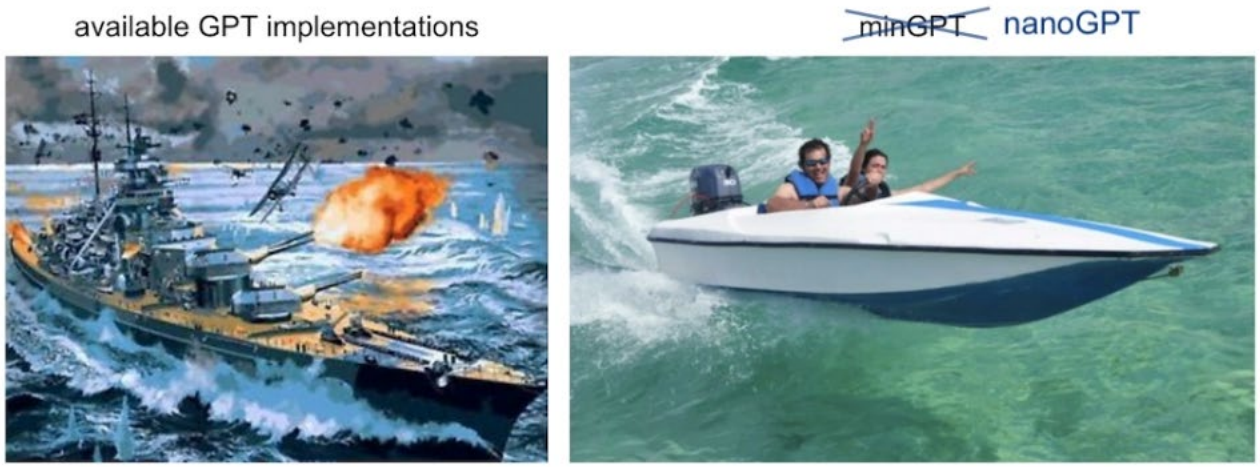
No, really, it predicts next tokens.

SOME RESOURCES

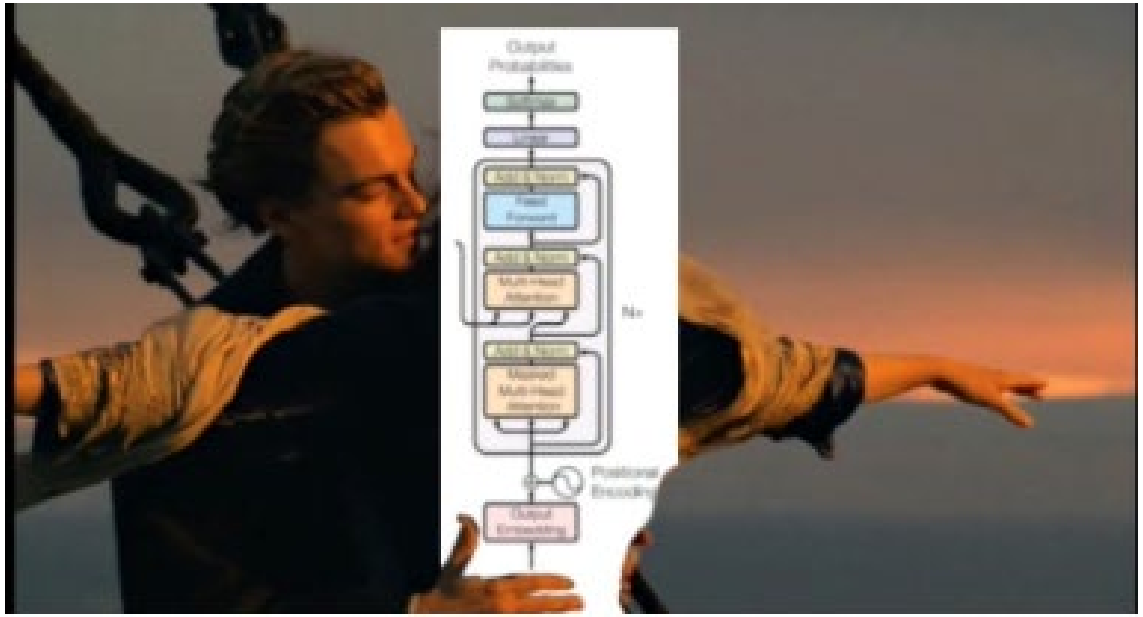
Andrej Karpathy

’15-17 Co-Founder OpenAI
’17-22 Tesla AI Director
‘23 Back at OpenAI!

nanoGPT



- https://github.com/karpathy/nanoGPT
  - A 300-line training loop + 300-line GPT model
  - Can load the GPT2 weights
  - Also has a Shakespeare demo. Better than mine. ☹



https://www.youtube.com/@AndrejKarpathy