S&DS 365 / 665

Intermediate Machine Learning

Transformers and LLM Scaling

November 20

Yale

For Today

Sequence models

- Transformer architecture
- Pseudocode
- LLM scaling laws
- Finetuning

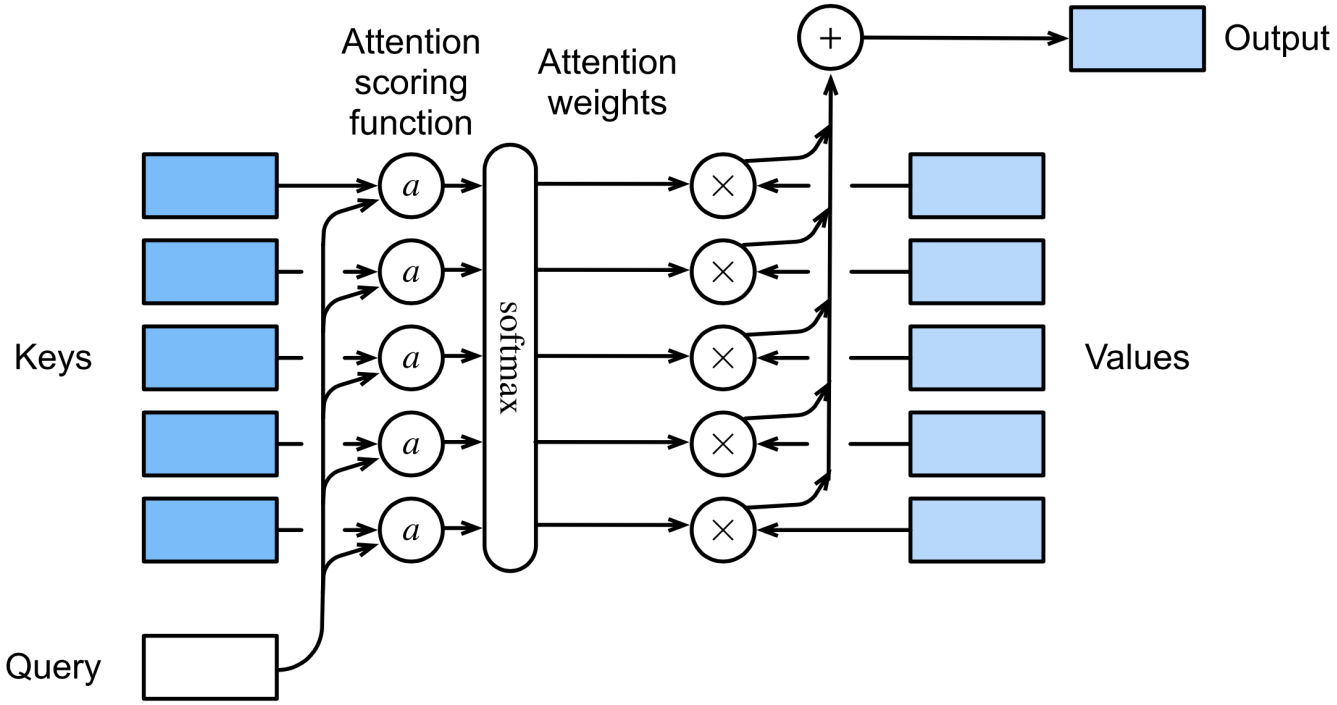
Attention, please!

- Quiz 5 (last!) is posted (RL, HMMs, RNNs, GRUs)
- Assn 5 out; due Dec. 4 (GRUs, Transformers)
- Final exam: Sunday Dec 15, 2pm
- Practice exams are posted
- Review sessions TBA

https://registrar.yale.edu/general-information/final-exams

Attention in terms of query/key/values

Basic idea behind attention mechanisms:



Attention in terms of query/key/values

Basic idea behind attention mechanisms:

Attention in terms of query/key/values

Basic idea behind attention mechanisms:

$$\begin{aligned}\mathrm{Attn}(q, \{(k_1, v_1), \ldots (k_m, v_m)\}) &= \mathrm{Attn}(q, (k_{1:m}, v_{1:m})) \\ &= \sum_{i=1}^{m} w_i(q, k_{1:m}) v_i\end{aligned}$$

where weights w_i are softmax of attention scores $a(q, k)$:

$$w_i(q, k_{1:m}) = \frac{\exp(a(q, k_i))}{\sum_{j=1}^{m} \exp(a(q, k_j))}$$

Kernel regression as attention

Recall the kernel regression estimator:

$$\widehat{m}(x) = \sum_{i=1}^{n} w(x, x_{1:n}) y_i$$

Here the attention scores (for Gaussian kernel) are

$$a(x, x_i) = -\frac{1}{2h^2} \|x - x_i\|^2$$

query:	test point x
keys:	data $x_1, \ldots, x_n$
values:	responses $y_1, \ldots, y_n$

Dot product attention

Note that if x_i and x have a fixed norm then the attention scores are just scaled dot products:

$$a(x, x_i) = \frac{1}{h^2} x^T x_i$$

If query and keys have same dimension d, dot product attention is

$$a(q, k) = q^T k / \sqrt{d} \in \mathbb{R}$$

and

$$\text{Attn}(Q, K, V) = \text{Softmax}\left(QK^T / \sqrt{d}\right) V$$

Scaling by $\sqrt{d}$ keeps variance constant

Key steps in Transformers

- *Tokenize* the text into a fixed vocabulary
- *Embed* the tokens into high-dimensional vectors
- *Mix* the embeddings in the context using “Attention”
- *Map* the resulting vectors using a neural network

The vocabulary problem

- Choice of vocabulary is important
- We each know $\approx$40K words
- What if a word is not in the vocabulary?
- Need to be able to process–and generate–arbitrary text

Tokenization and the BPE method were first developed in the 2019 OpenAI GPT-2 paper, Radford et al. “Language models are unsupervised multitask learners”

Tokenization

```
Fugetsu-Do, a mochi store in the Little Tokyo neighborhood of Los
Angeles, has existed for 121 years, with a lasting impact on its
community.

Billy was a very baaaaaaaaad goat.

This is another $%^*ish example.
```

This is what we see

platform.openai.com/tokenizer

Tokenization

Fugetsu-Do, a mochi store in the Little Tokyo neighborhood of Los Angeles, has existed for 121 years, with a lasting impact on its community.

Billy was a very baaaaaaaad goat.

This is another $%^*ish example.

Text | Token IDs

This is how the tokenizer groups bytes into tokens

platform.openai.com/tokenizer

Tokenization

```
[77845, 19209, 84, 12, 5519, 11, 264, 4647, 14946, 3637, 304, 279, 15013,
27286, 12818, 315, 9853, 12167, 11, 706, 25281, 369, 220, 7994, 1667, 11,
449, 264, 29869, 5536, 389, 1202, 4029, 382, 97003, 574, 264, 1633, 293,
29558, 33746, 329, 54392, 382, 2028, 374, 2500, 400, 46999, 9, 819, 3187,
382]
```

Text | Token IDs

This is what the LLM sees—the “atoms” of an LLM

platform.openai.com/tokenizer

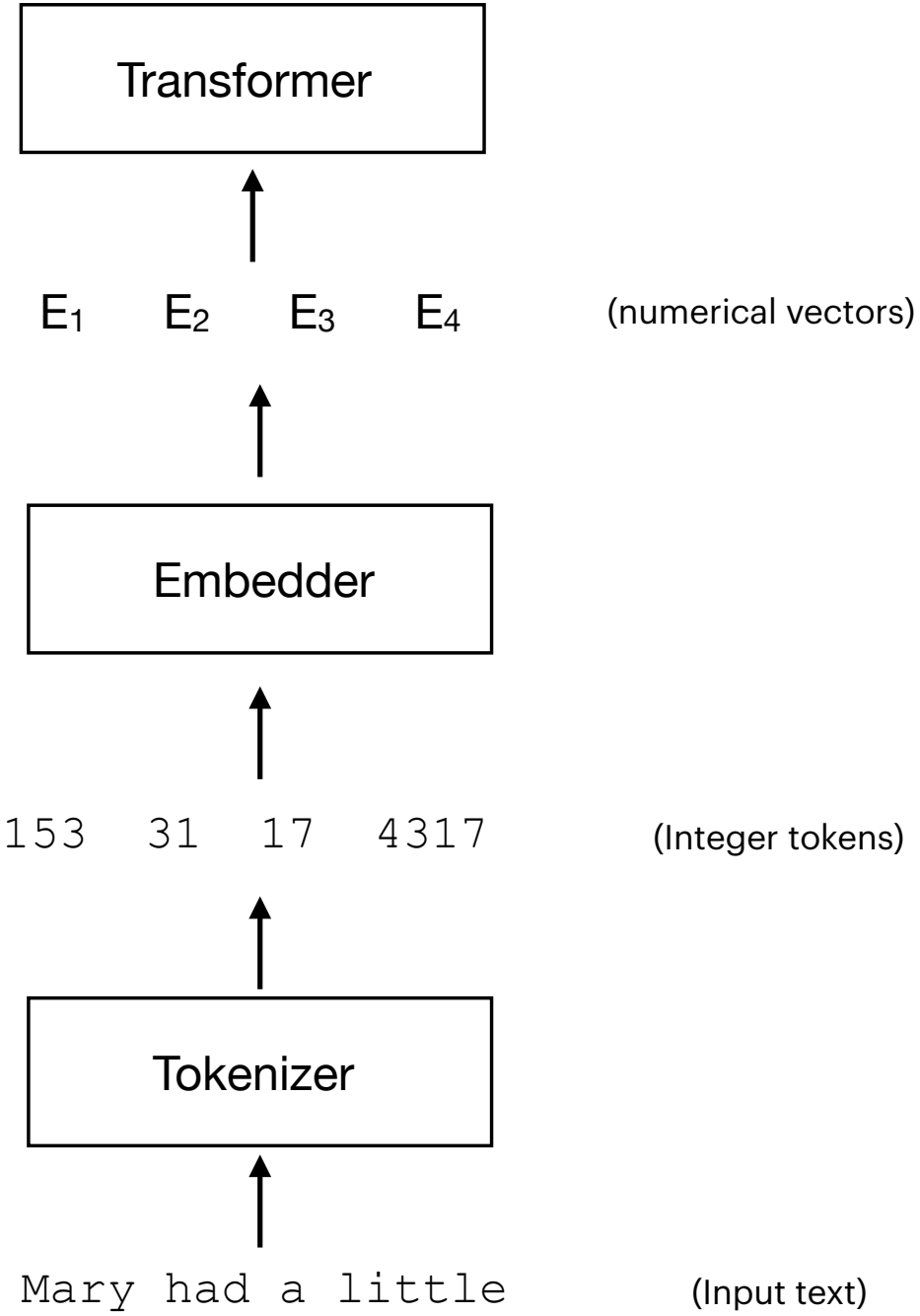
Tokenizer properties

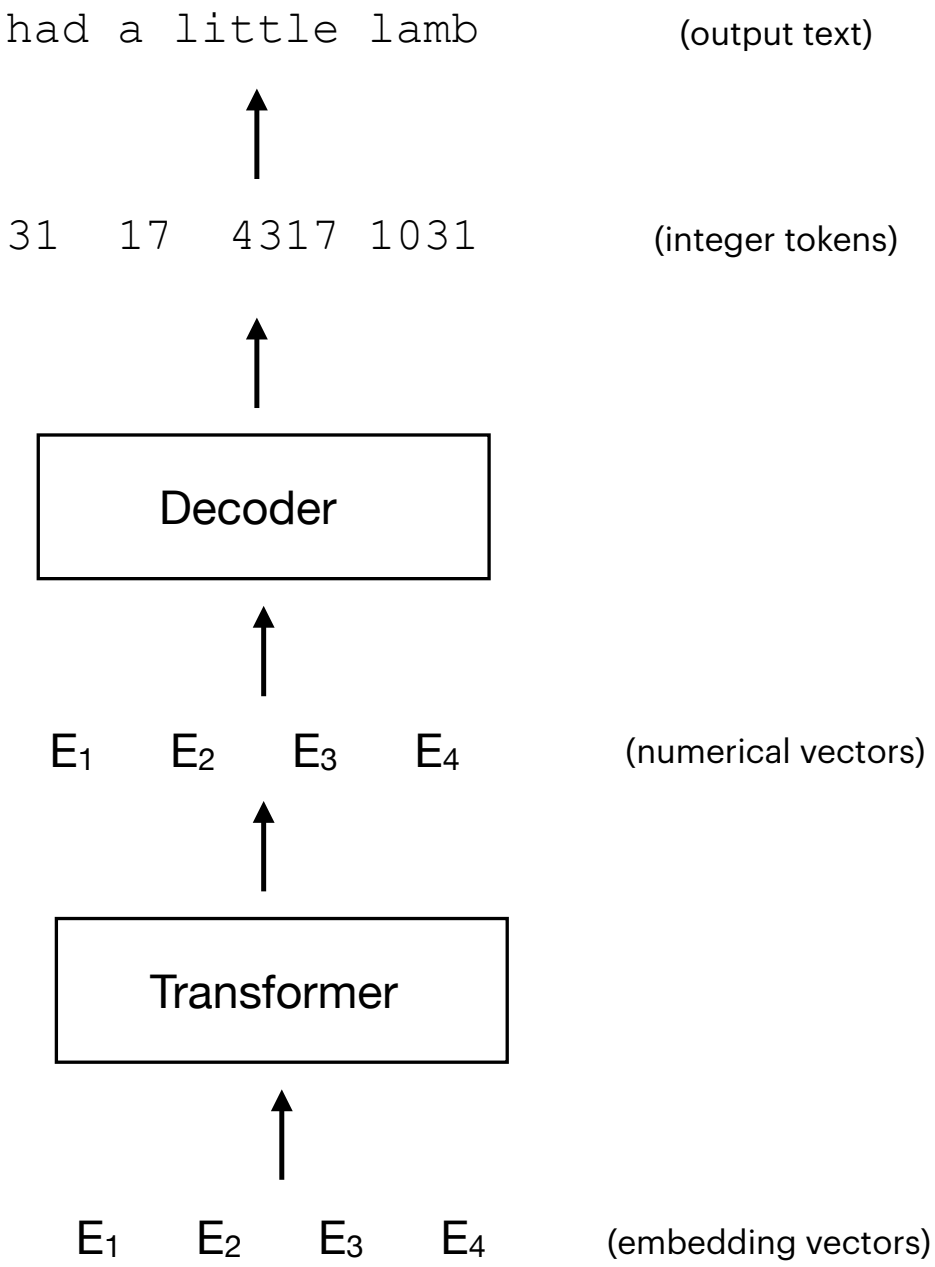
- *Reversible:* Can convert from tokens back to original text
- *General:* Works on arbitrary text, even very different from training
- *Compressive:* Each token is about 4 bytes
- *Statistical:* Will break strings into frequently occurring pieces

BPE Algorithm – Byte Pair Encoding

1. Input: Large sequence of bytes (UTF-8 encoding)
2. Initialize: Each byte is a token; V=256
3. Iterate for K steps:
   i. Find pair of tokens (s, t) with largest count of co-occuring
   ii. Replace each occurrence of (s, t) with V
   iii. $V \leftarrow V + 1$
4. Final vocabulary size: $256 + K$

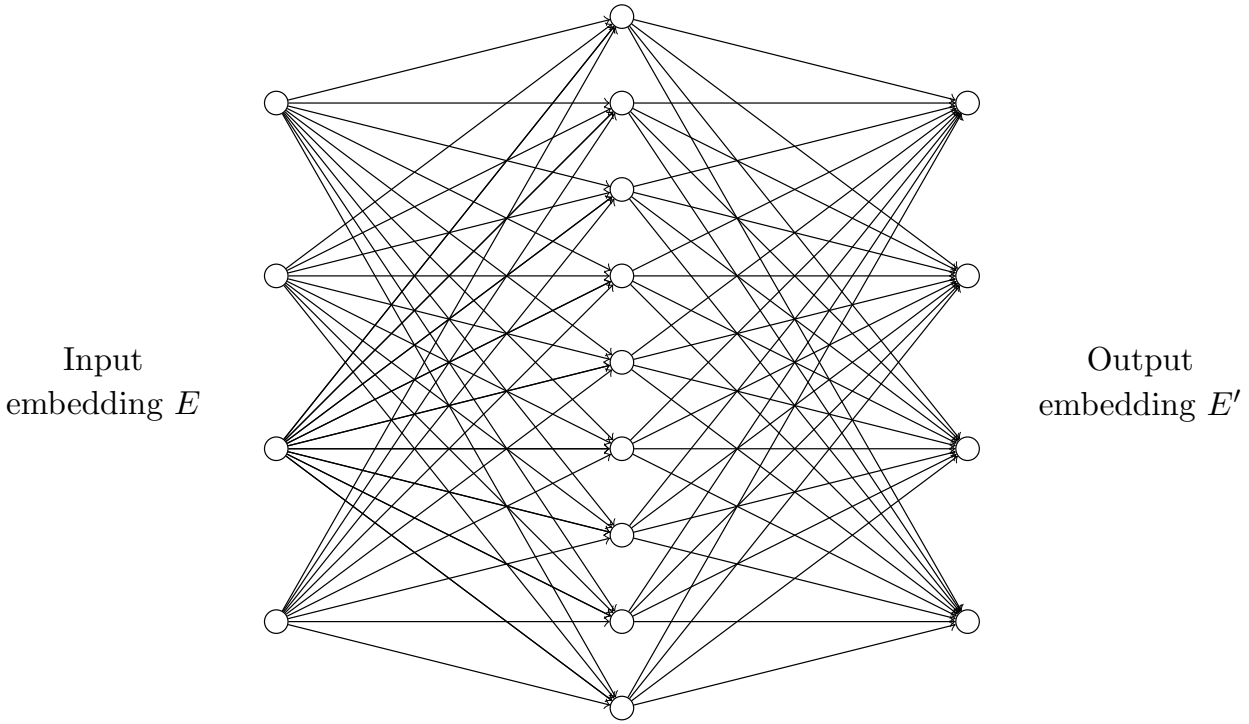
https://github.com/karpathy/minbpe; https://www.youtube.com/watch?v=zduSFxRajkE


Transformer
E1 E2 E3 E4
(numerical vectors)
Embedder
153 31 17 4317
(Integer tokens)
Tokenizer
Mary had a little
(Input text)


had a little lamb
(output text)
31 17 4317 1031
(integer tokens)
Decoder
E1 E2 E3 E4
(numerical vectors)
Transformer
E1 E2 E3 E4
(embedding vectors)

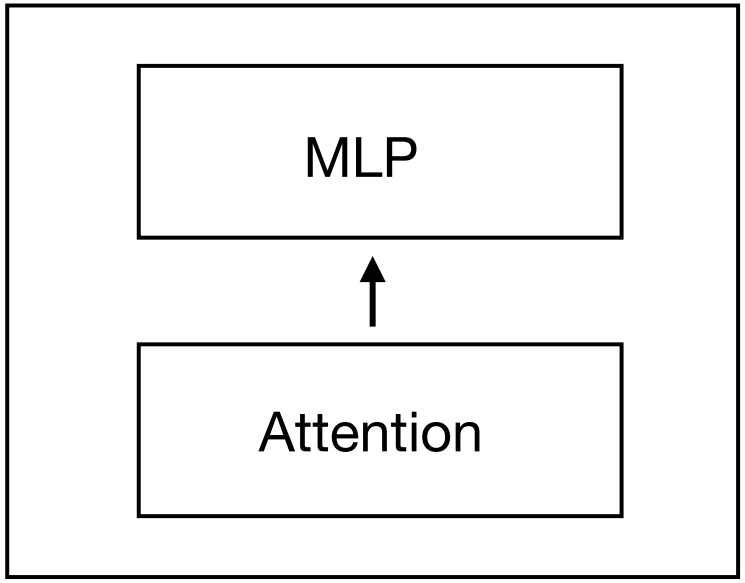
Adding nonlinearities

The last step is to take the resulting vectors (after they have been mixed up), and map them to new vectors using an MLP:

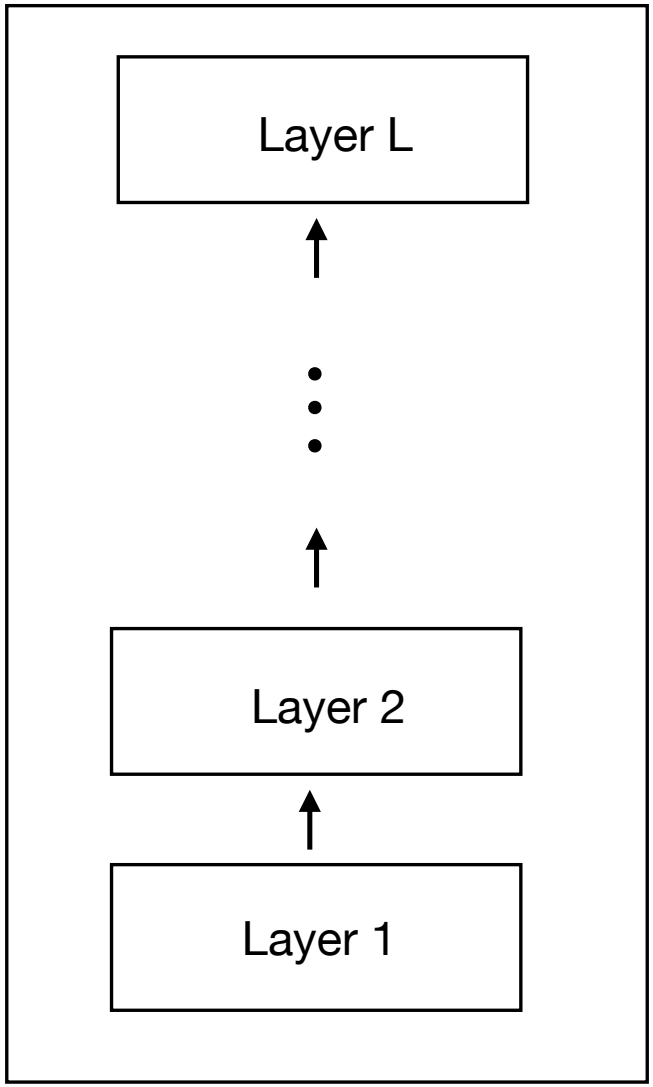


Creates new “features” to be used as embeddings in the next layer

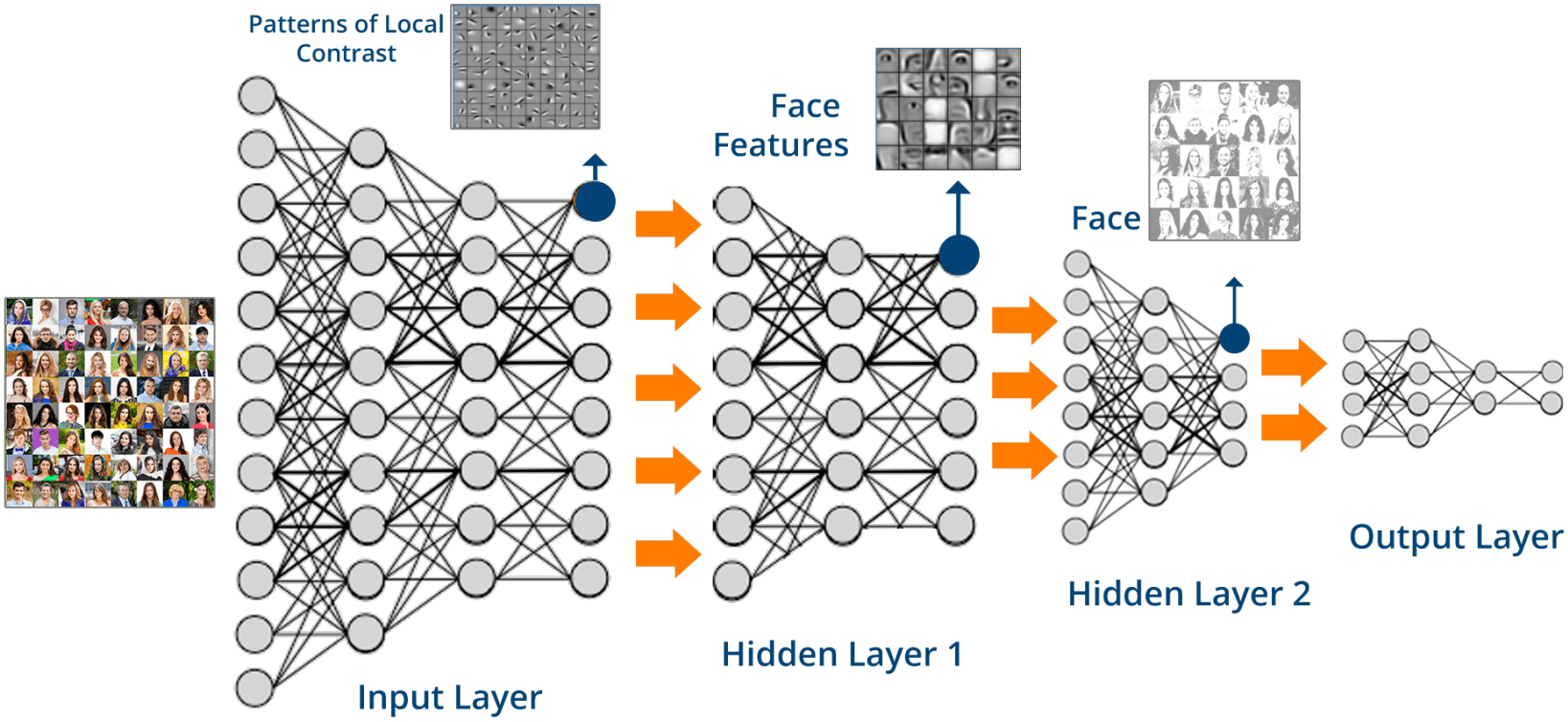
Transformer Layer



Transformer

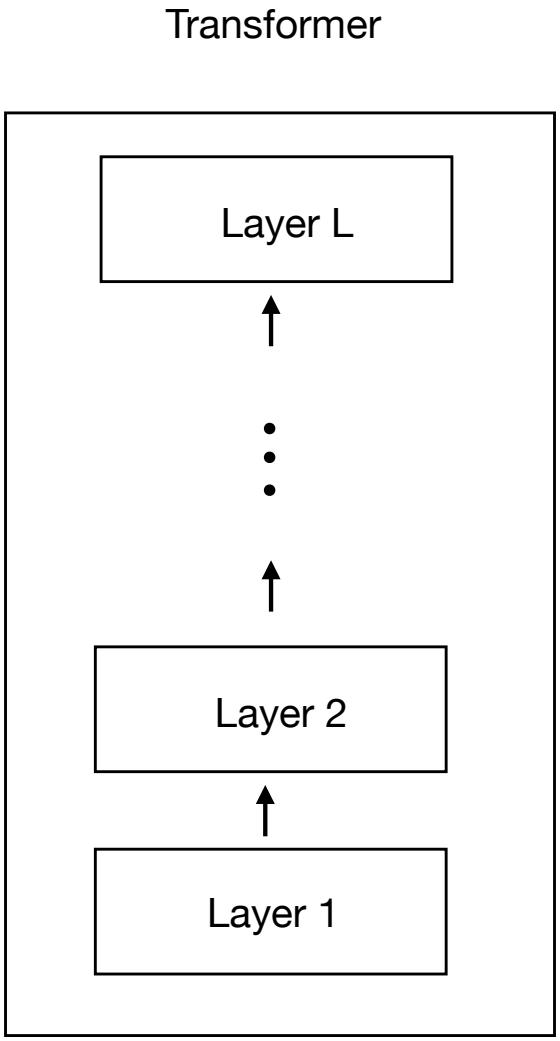


Increasing complexity of attention patterns



- Early layers capture local patterns of image constrast
- Later layers capture larger patterns that match parts of faces

Increasing complexity of attention patterns



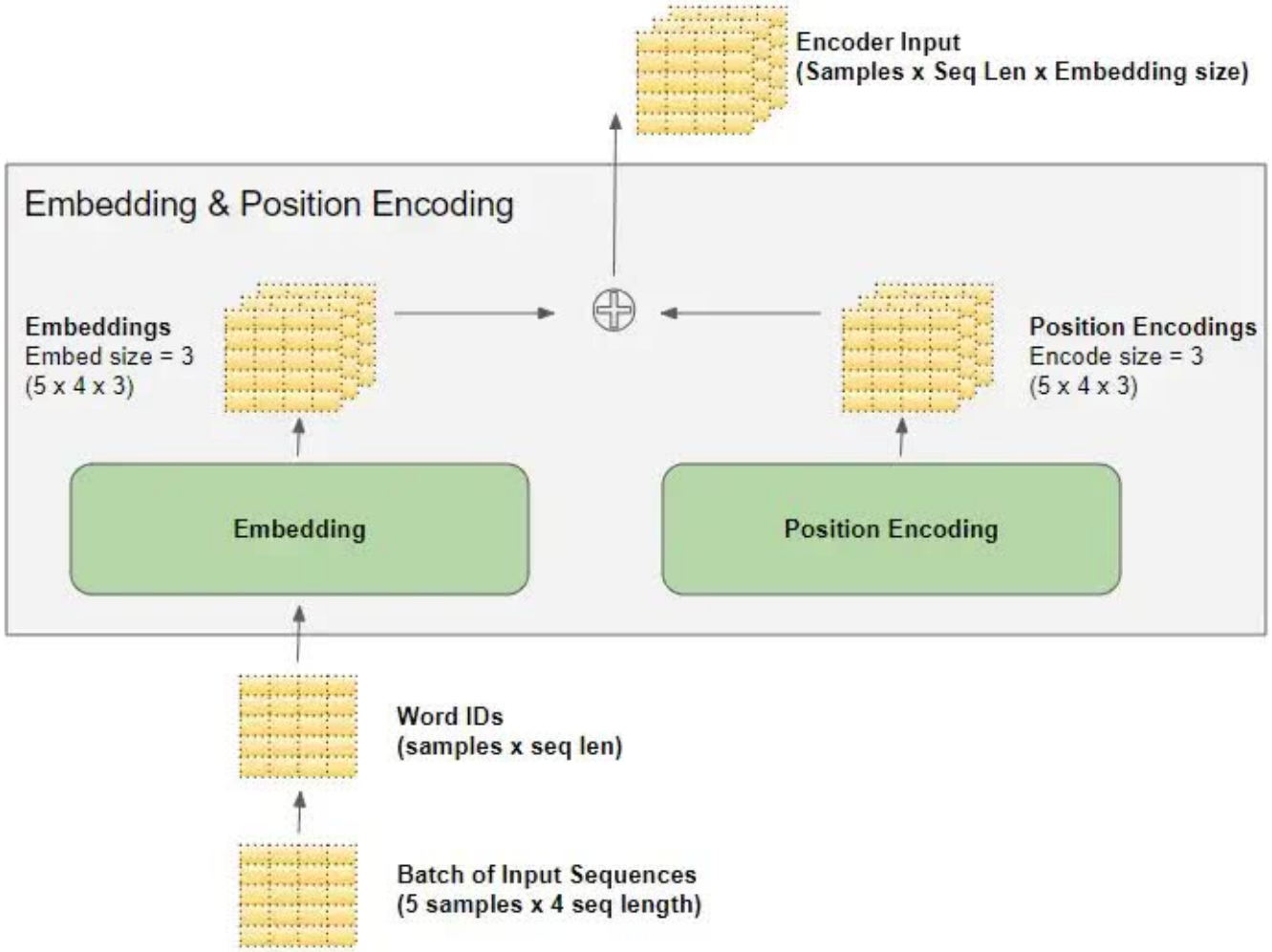
features of tokens
spanning long range

⋮

local token statistics
(1-2 tokens back)

More detail on attention

Text processed in batches; embedding stage looks like this:



https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Positional encoding

Positional encoding is an interesting way of getting position information into the model

Positional encoding

- Positions are integers, but neural nets can't handle integers
- Could use binary encoding; fixed precision
- Transformers use a sine/cosine basis

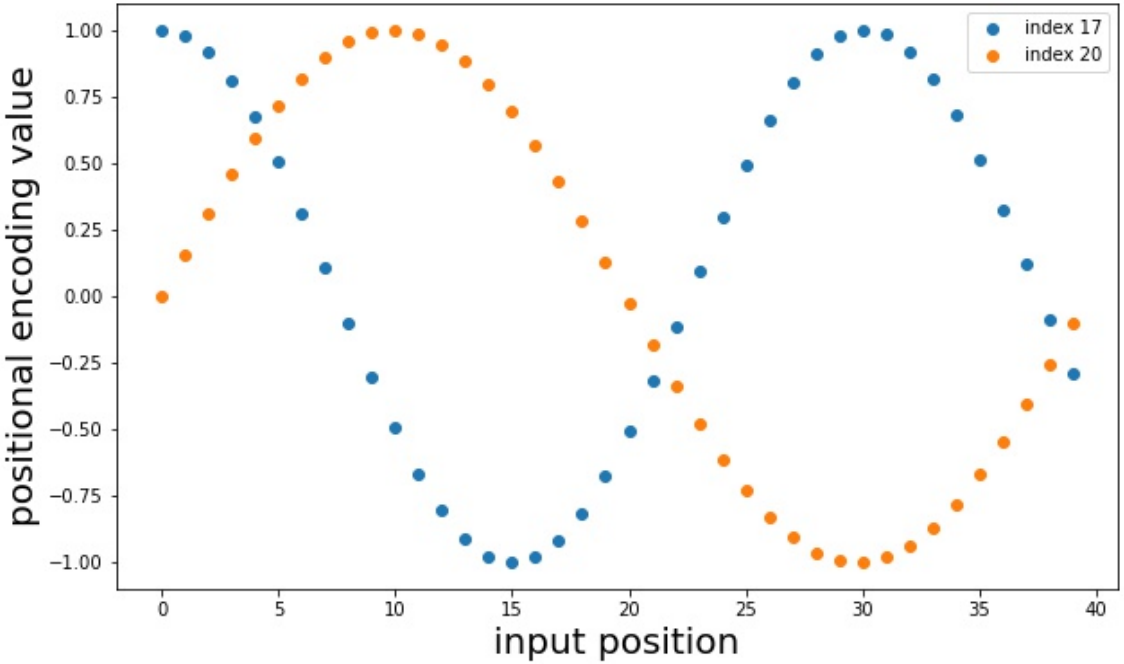
Positional encoding

For the tth word in the sequence, components are

$$\mathrm{PE}_{(t,2i)} = \sin\left(\frac{t}{C^{2i/d}}\right)$$
$$\mathrm{PE}_{(t,2i+1)} = \cos\left(\frac{t}{C^{2i/d}}\right)$$

where d is the embedding dimension and C is a maximum sequence length

Positional encoding



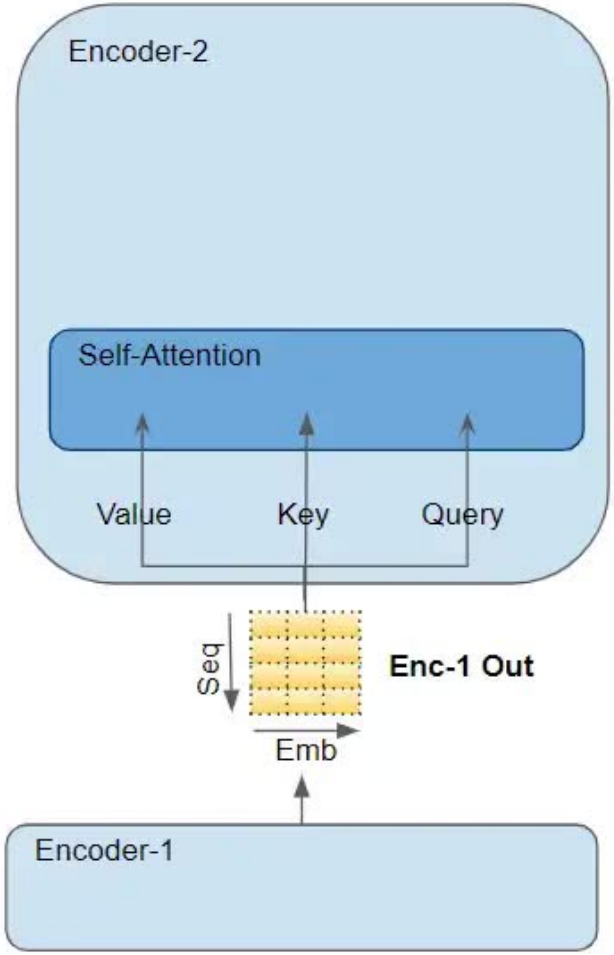
Two components for a length 40 input

Positional encoding

- Encoding of one position is linearly predictable from another, given their offset distance (phase shift)
- If the positional encoding is removed, the resulting model is permutation equivariant

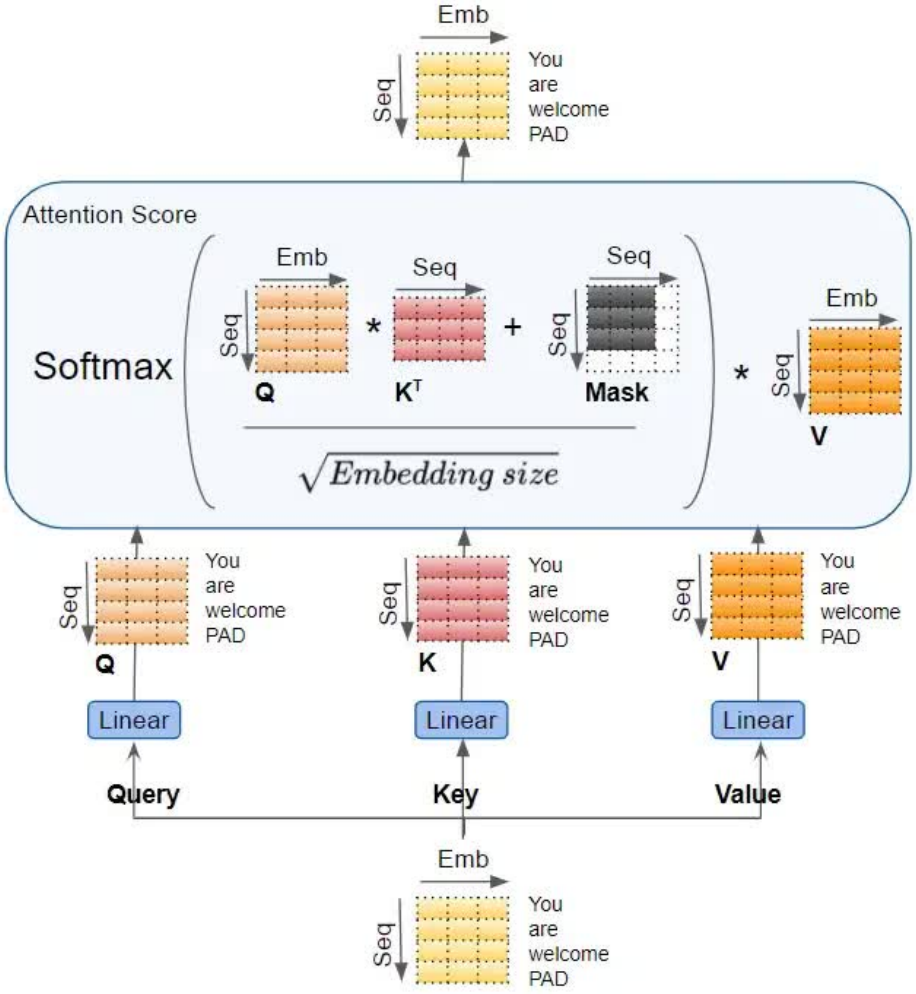
Multihead Attention

Detail of self-attention component

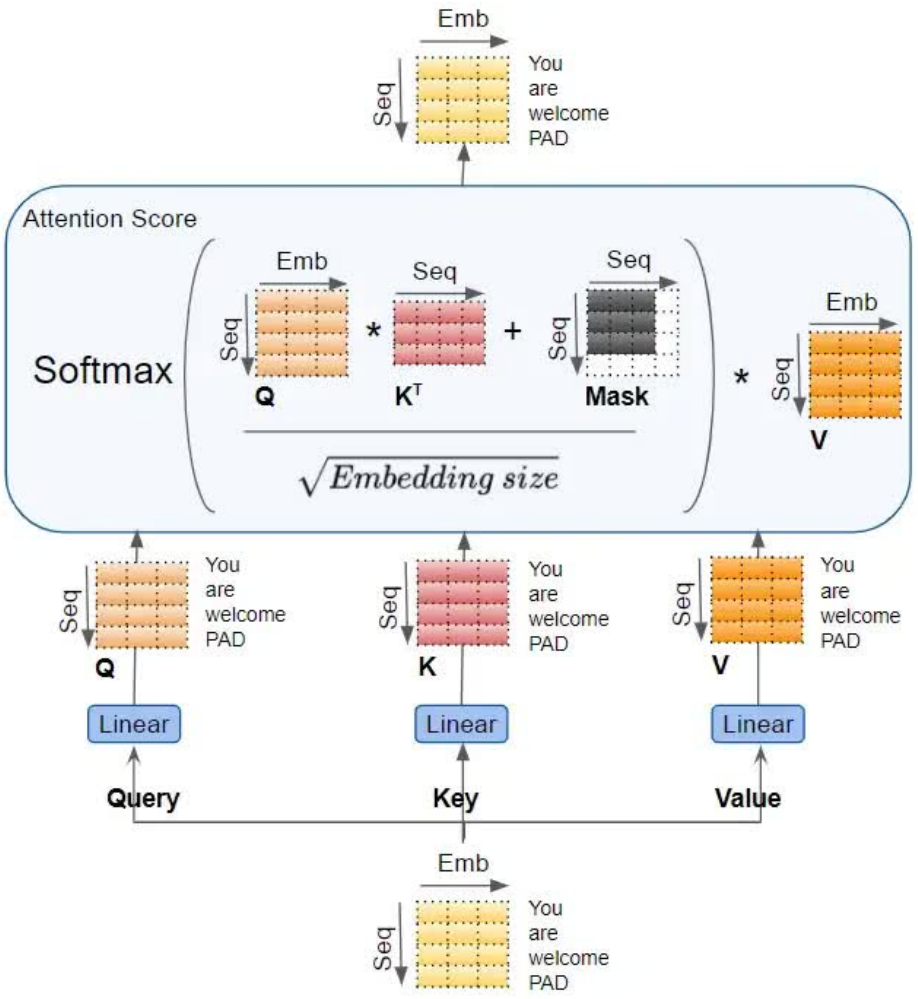


Multihead Attention

Detail of self-attention component (Blue are trainable parameters):

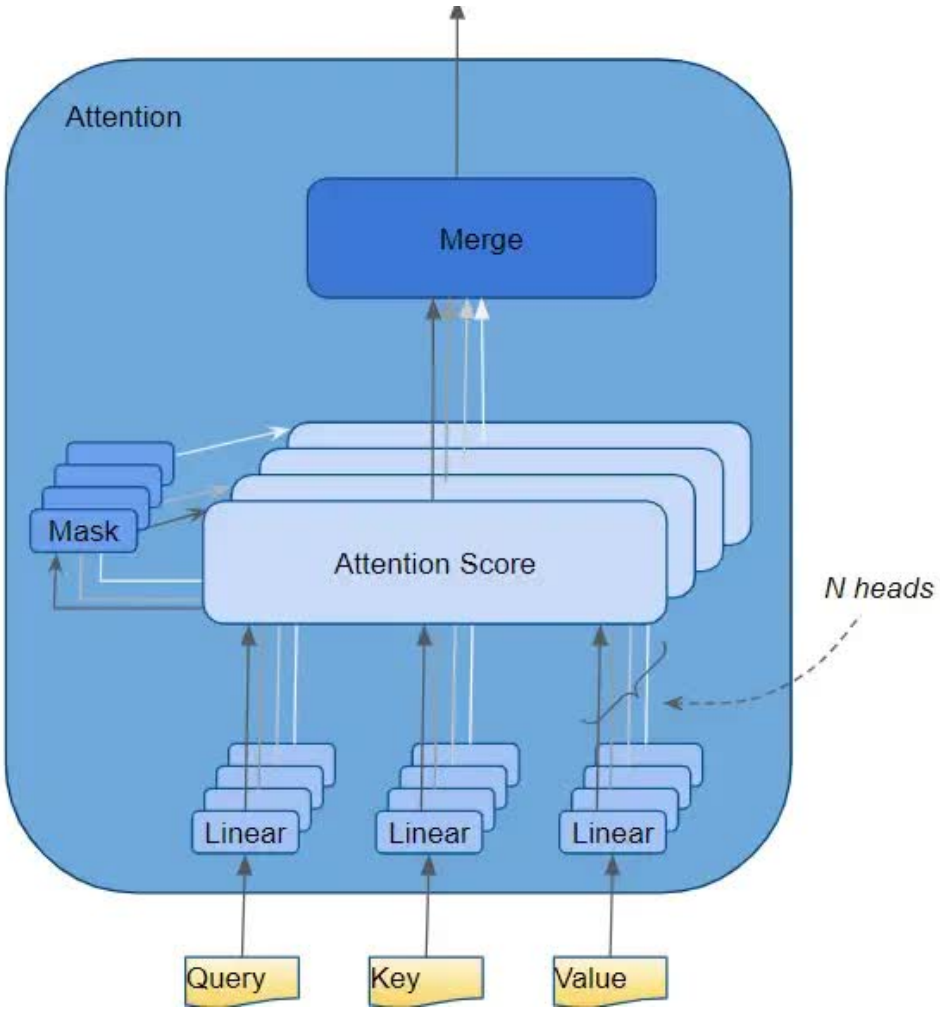


Multihead Attention



Mask is used in decoder to prevent use of future words

Multihead Attention



Multiple attention vectors are computed, then merged (concatenated)

Computation

- Everything is implemented as big matrix multiplies
- Carried out very efficiently on GPUs
- GPUs are optimized for linear equations (vector graphics)
- Allows scaling models to very large size

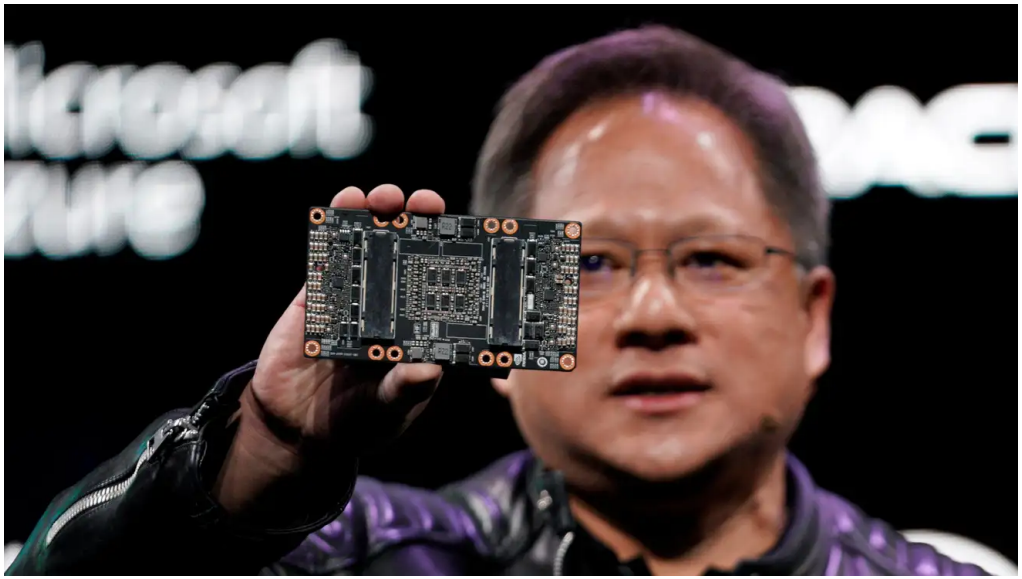


Photo from Reuters; https://stackoverflow.com/questions/51344018/why-can-gpu-do-matrix-multiplication-faster-than-cpu

Final step

Generating new tokens

Rolling a D50,257

- Transformer assigns a score to each token
- Converted to a probability
- Generating next token equivalent to rolling weighted 50,257-sided die

Remixing Noon

Text generated from Channel Skin by Jeff Noon

"The whole forest had been anesthetised, her temples wired, her senses stimulated, her eyes were not on Eva, not on Eva, not on Eva, not on anybody in that same realm, the land of dreams and nightmares."

Viability: 0.000000326%

The whole forest had been anesthetised, her temples wired, her senses stimulated, her eyes were not on Eva, not on Eva, not on Eva, not on anybody in that same realm, the land of dreams and nightmares.
room blurry in her place looked as though
feel of stickiness, light to stare, and hand, welcomed tuned to
feeling... it came feeling lust
strangest
instructed in the [many]
injected.
reading the local news, letters as on
mistake.
a while.
her armour.
lifted up in a series once more
put together from a location guise
given her new voice, companion this chance
digitally mutated, excised of
recorded once, filmed, years
issued for that particular
set up in a series one, Melissa's
within a
decorated especially for this
remove by the [many]
welcomed sighed opened kiss touch frown gabbed
out and
into the [many]
palms, head, eyes, mind, own
team.
glowed with
transmitted life.
sudden desire. burbles
the [many]
life.
a colour in strange colours
bioluminescence the
bound.
lost and in the clouded.
unfocused, half hooded; she
bleary, half closed but
dazzled, half-blinded by what
red-lined; she may have be.
and off.
and off.
and off.
and off.
and off.
Eva
anybody in that same room.
anybody in that same room.
anybody in that same room.
anybody in that same room.
flickering
briefly.
waited for trade. some
lush-golden of hair.
scanned her heartbroken
he felt proud
sat there
could be [many]
away from [many]
and
to the [many]
Fixed on a some faraway nonexistent
bright and fierce
burning
glow, to
where they stung, were
and closed.
ears plugged, mouth.
staring
and
her in the clouded.
herself
moment.
moment.
moment.

Generation

Generation algorithm has two “knobs” to control:

1. *Top-k*: Restricts to just the top k tokens
   - Low k: Low variability
   - High k: Higher variability

Generation

Generation algorithm has two “knobs” to control:

2. *Temperature T*: Lower temperature favors more likely tokens
   - Low temperature: Low variability
   - High temperature: High variability

Generation

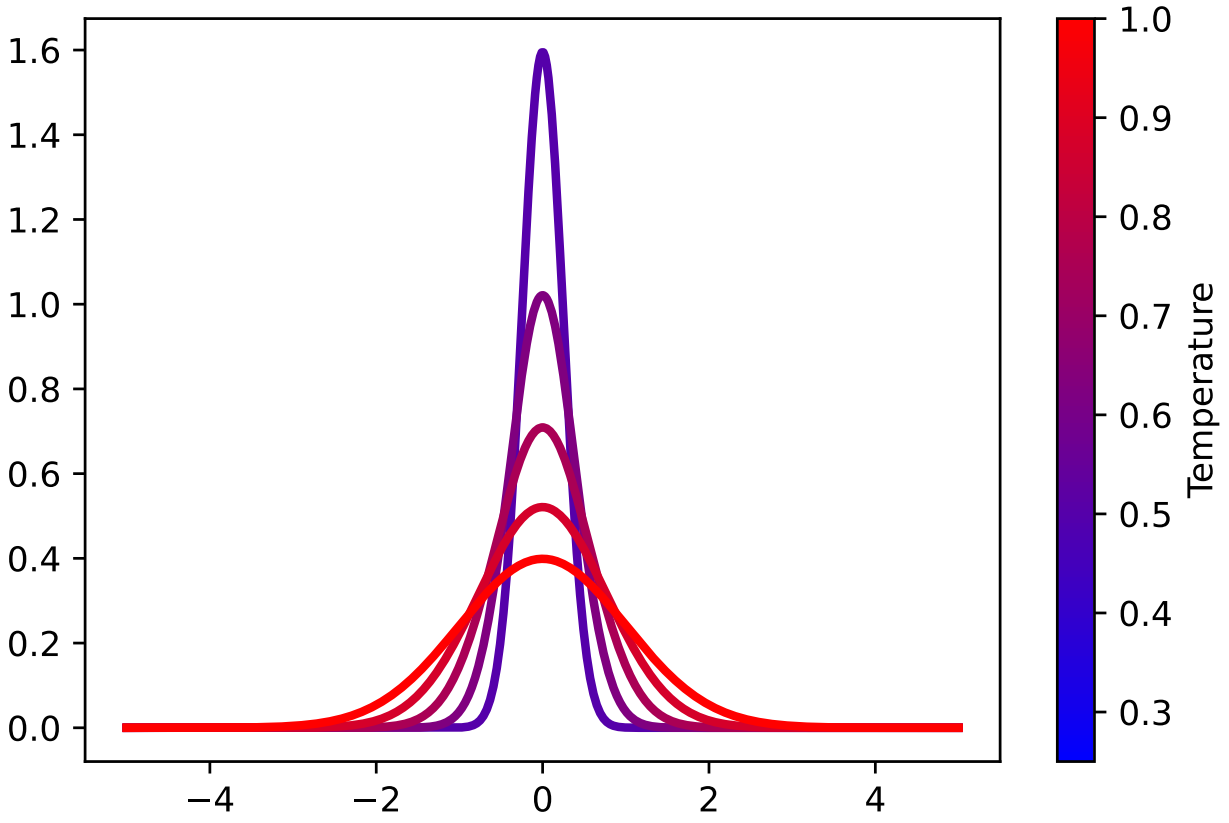
At temperature T, if the scores of the top k tokens are

$$\text{score}_1, \text{score}_2, \ldots, \text{score}_k$$

the weights on the die are proportional to

$$e^{\text{score}_1/T}, e^{\text{score}_2/T}, \ldots, e^{\text{score}_k/T}$$

Temperature



Temperature $T = 1$

One day, Timmy met a monster. The monster was big and green, but it was harmless. Timmy was scared, but he wanted to be brave. He walked closer and closer to the monster. Suddenly, the monster snapped its fingers! Timmy was so scared that he started to cry. The monster said, “Don’t be scared! I’m harmless!” But Timmy was still sc

One day, Timmy met a monster. The monster had big teeth and green skin. Timmy was scared and ran away. But the monster called out, “Wait! I don’t want to hurt you. I just want to be your friend.” Timmy stopped and looked back. “Really?” he asked. “Yes, I permit you to be my friend,” the monster replied. Timmy was surprised. He didn’t expect

https://github.com/karpathy/llama2.c

Temperature $T = 0$

One day, Timmy met a monster. The monster was big and scary. Timmy was scared and ran away. The monster chased Timmy. Timmy ran and ran until he found a big tree. He climbed up the tree and the monster couldn't reach him. Timmy was safe. Timmy saw a bird and said, "Hello bird, can you help me? The monster is chasing me." The bird said, "

One day, Timmy met a monster. The monster was big and scary. Timmy was scared and ran away. The monster chased Timmy. Timmy ran and ran until he found a big tree. He climbed up the tree and the monster couldn't reach him. Timmy was safe. Timmy saw a bird and said, "Hello bird, can you help me? The monster is chasing me." The bird said, "

Temperature $T = 1.5$

One day, Timmy met a monster. The monster was big and scary. Timmy didn't know what to do. He felt bitter because he wanted to play with the monster. The monster said, "Hello, Timmy. Can you show me around the forest?" Timmy wasn't sure if he believed him or not. He said, "I'm sorry, do you have room for crawlers?" The monster guesses sad

One day, Timmy met a monster. The monster was big and scary, but Timmy wasn't scared. "Hi," said Timmy. The monster was surprised. He couldn't understand what Benny had said. "What are you up to?" he asked. Timmy picked him up one by one and he saw doziles of Ell Party coming in on a streak street. "I amdri catientased!"

Transformer architecture: Pseudocode

```
def Transformer(tokens):

    x = embed_tokens(tokens)
    x = x + positional_encoding(len(tokens))

    for l in [L]:
        x = x + Attention(l, x)
        x = x + MLP(l, x)

    logits = x @ W + b
    return logits
```

Attention: Pseudocode

```
def Attention(l, x)
    head_outputs = []
    for h in [n_heads]:

        Q = x @ W_q[l, h]
        K = x @ W_k[l, h]
        V = x @ W_v[l, h]

        attn_scores = Q @ K.T / sqrt(d)
        attn_scores = Softmax(attn_scores)

        head_out = attn_scores @ V
        head_outputs = head_outputs + head_out

    return head_outputs @ W_o[l]
```

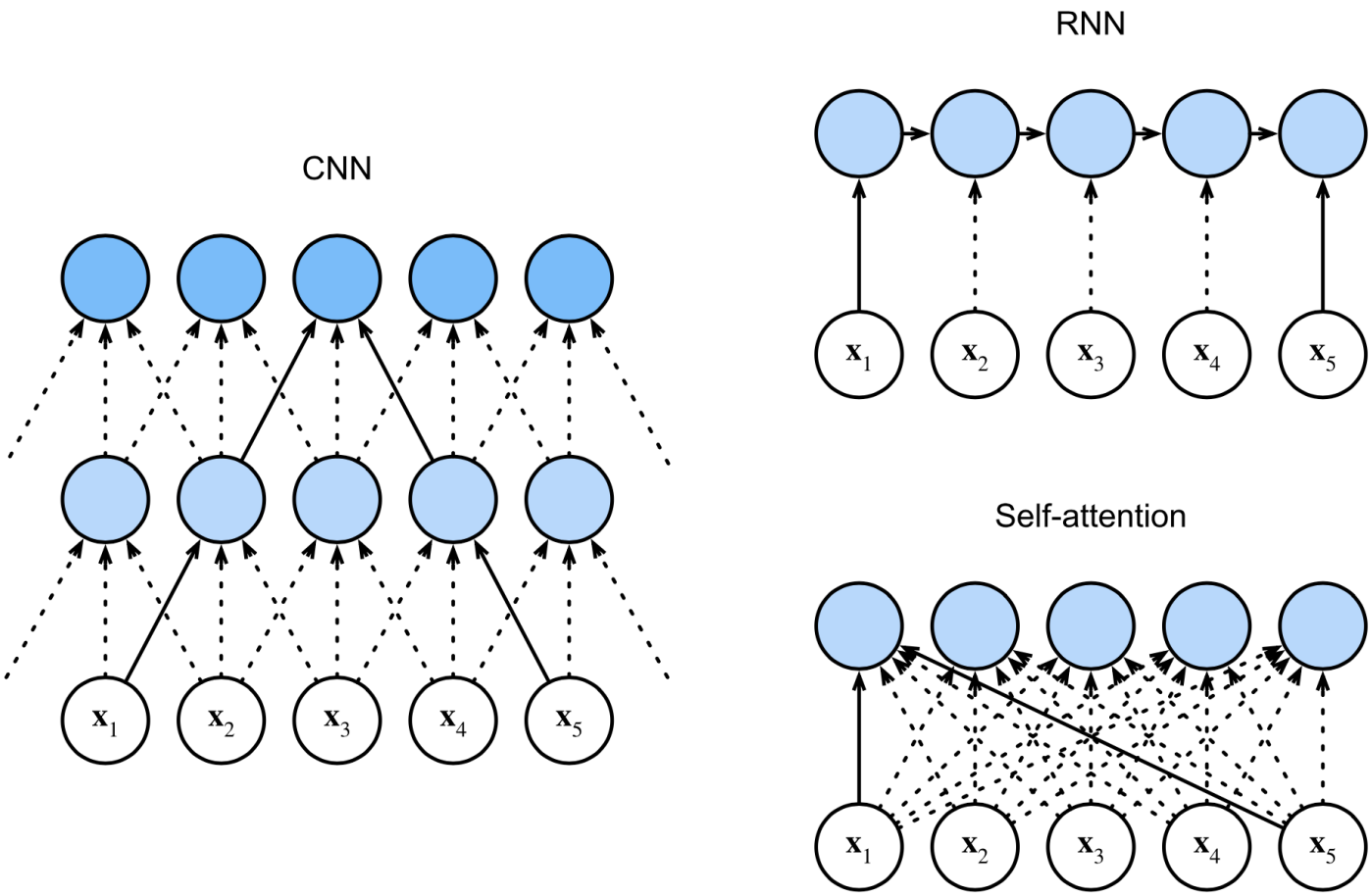
MLP: Pseudocode

```
def MLP(l, x)

    h = x @ W1[l] + b1[l]
    h = relu(h)
    o = h @ W2[l] + b2[l]

    return o
```

Generation: Pseudocode

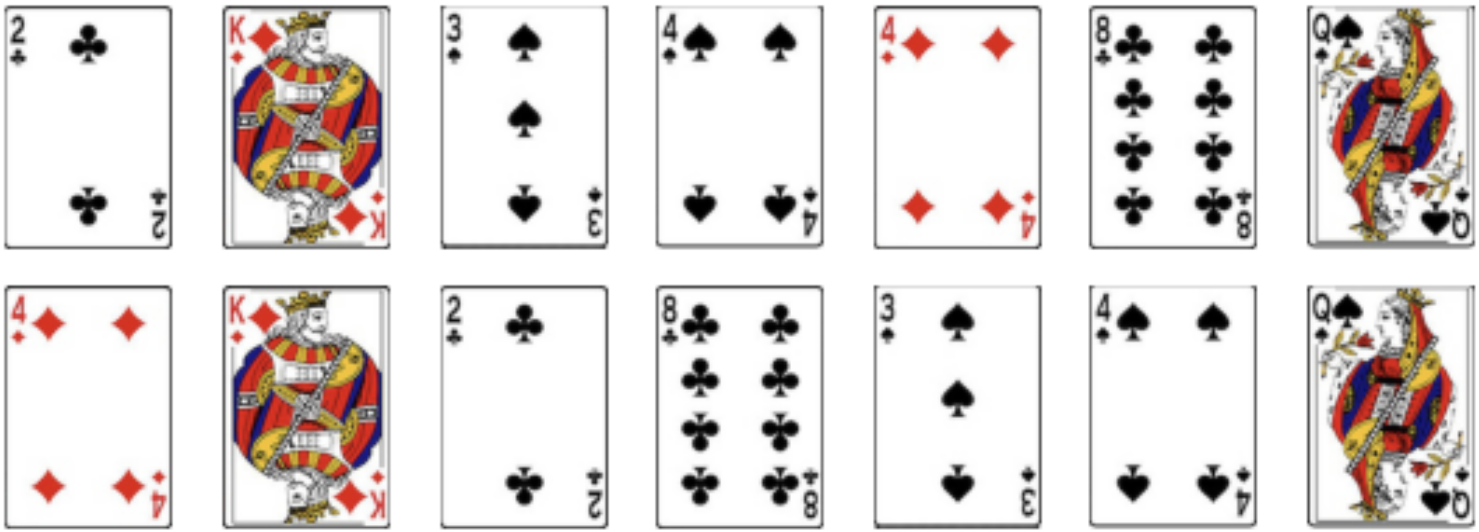
```
def generate(prompt_text, temperature):

  generated_tokens = tok.encode(prompt_text)

  for i in [tokens_to_generate]:
     logits = Transformer(generated_tokens)
     if temperature == 0:
        next_token = argmax(logits)
     else:
        logits = logits / temperature
        probs = softmax(logits)
        next_token = sample(vocab_size, probs)
     generated_tokens += next_token

generated_text = tok.decode(generated_tokens)
return generated_text
```

Abstract comparison



Demo

```
for i in range(5):
    j = np.random.choice(range(len(output)))
    deck.show_hand(source_test[j,1:(hand_size+1)])
    deck.show_hand(output[j,1:(hand_size+1)])
    print('\n')
```



The Transformer architecture

How large are the models?

How does size of LLM scale?

Design choices: number of tokens V, dimension d of embeddings, and number of Transformer layers L

- Embeddings have $d \cdot V$ parameters
  - d numbers for each token, V tokens

How does size of LLM scale?

Design choices: number of tokens V, dimension d of embeddings, and number of Transformer layers L

- Self-Attention has $\approx 4d^2$ parameters
  - $d \times d$ matrix for queries (across heads)
  - $d \times d$ matrix for keys (across heads)
  - $d \times d$ matrix for values (across heads)
  - $d \times d$ matrix for combining heads

How does size of LLM scale?

Design choices: number of tokens V, dimension d of embeddings, and number of Transformer layers L

- MLP applied after has $\approx 8d^2$ parameters
  - Two layers
  - Hidden layer has $4 \cdot d$ neurons
  - Output layer has d neurons
  - $4d^2$ weights in each layer; $8d^2$ weights overall

How does size of LLM scale?

Design choices: number of tokens V, dimension d of embeddings, and number of Transformer layers L

- Embeddings have $d \cdot V$ parameters
- In each Transformer layer:
  - Attention has about $4d^2$ parameters
  - MLP has about $8d^2$ parameters
- Total trainable parameters: $12d^2 \cdot L + d \cdot V$

Size of models

GPT-2 (2019)

$$d = 1{,}600$$
$$V = 50{,}257$$
$$L = 48$$
$$12 \cdot d^2 \cdot L + d \cdot V \approx 1.5 \text{ billion parameters}$$

- Each GPT-2 Transformer layer has about 30 million parameters

Size of models

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%

en.wikipedia.org/wiki/GPT-3. Note: training often refers to number of tokens *processed*; requires multiple passes through data

Size of models

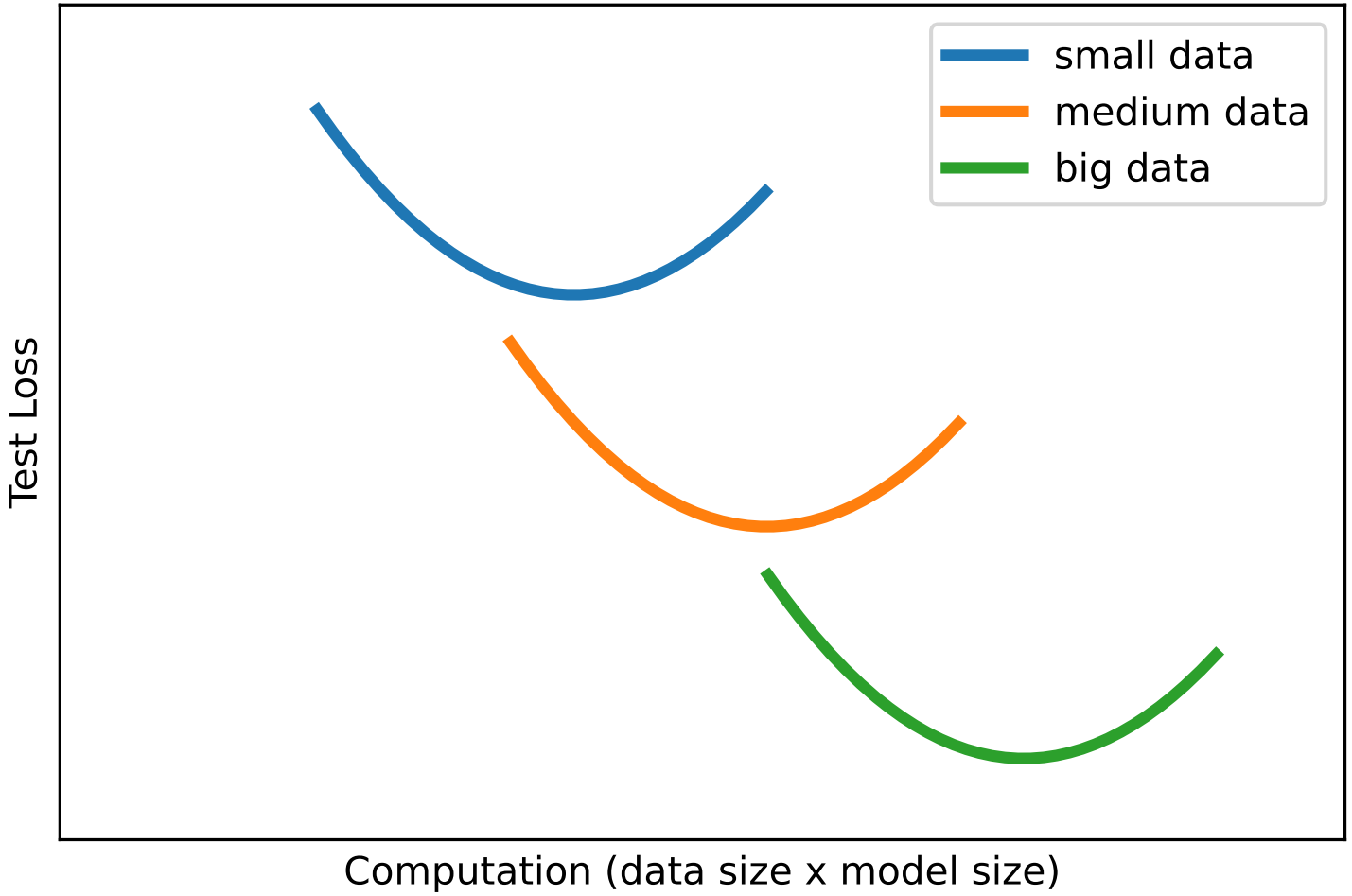
GPT-3 (2020)

$$d = 12{,}288$$
$$V = 50{,}257$$
$$L = 96$$
$$12 \cdot d^2 \cdot L + d \cdot V \approx 173 \text{ billion parameters}$$

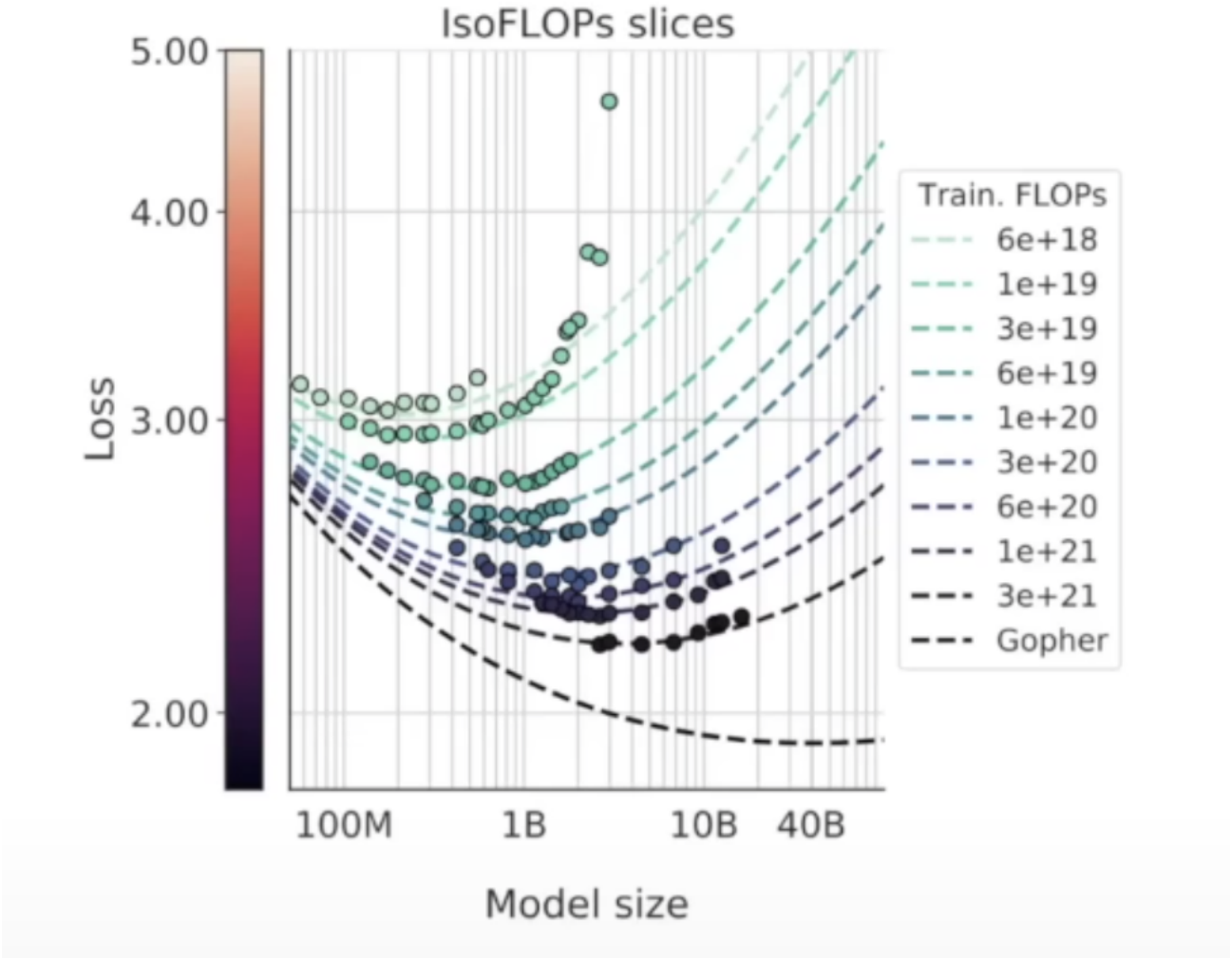
- Each GPT-3 Transformer layer has about 1.8 billion parameters

GPT-4 (2023) data and size unknown; thought to be 1.76 trillion. The human brain has about 100 trillion synapses.

Scaling behavior of LLM models



Scaling behavior of LLM models



Original painting by Johannes Vermeer; transformed (pixelated) by acagastya., CC0, via Wikimedia Commons

ANNALS OF ARTIFICIAL INTELLIGENCE

CHATGPT IS A BLURRY JPEG OF THE WEB

OpenAI's chatbot offers paraphrases, whereas Google offers quotes. Which do we prefer?

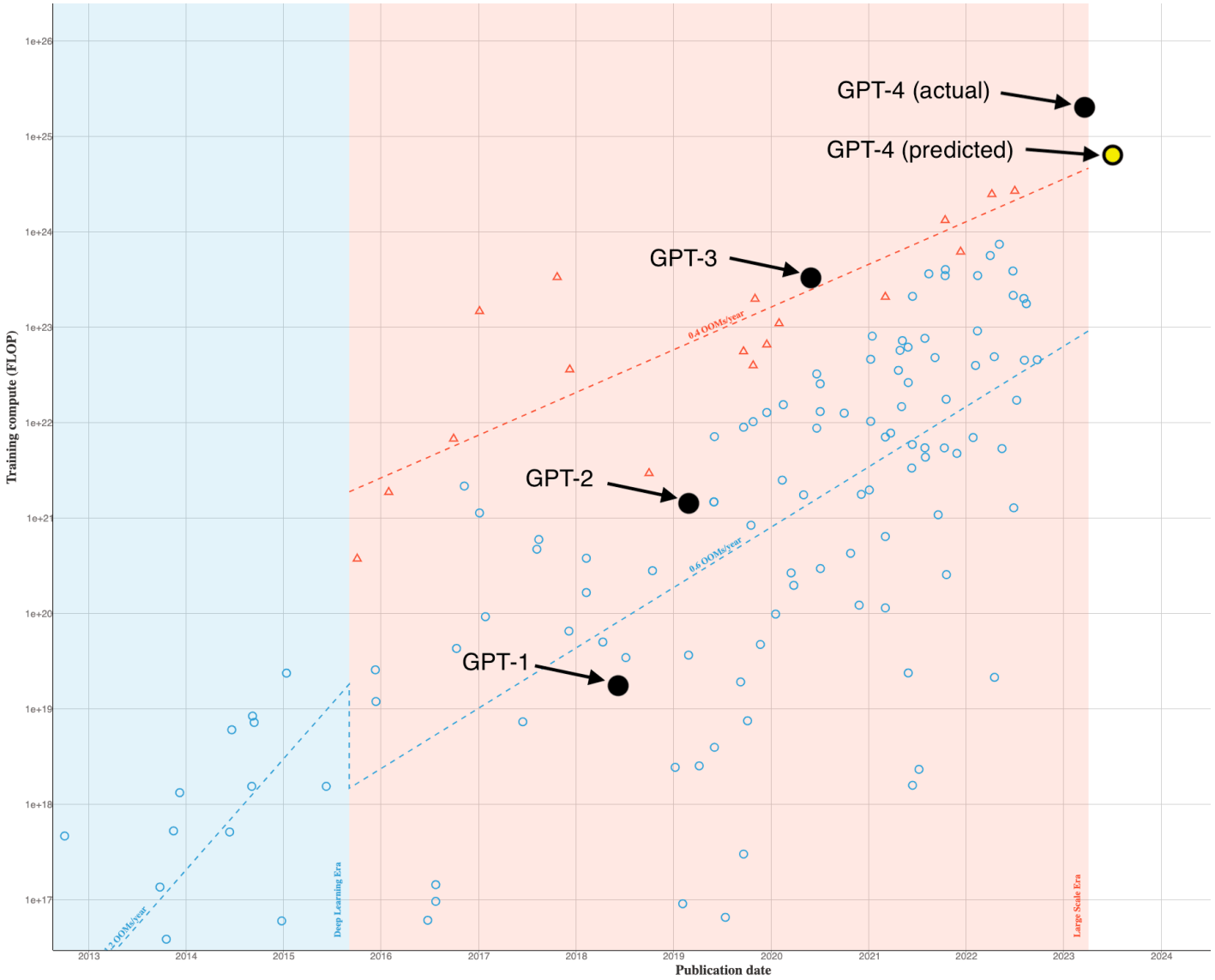
By Ted Chiang

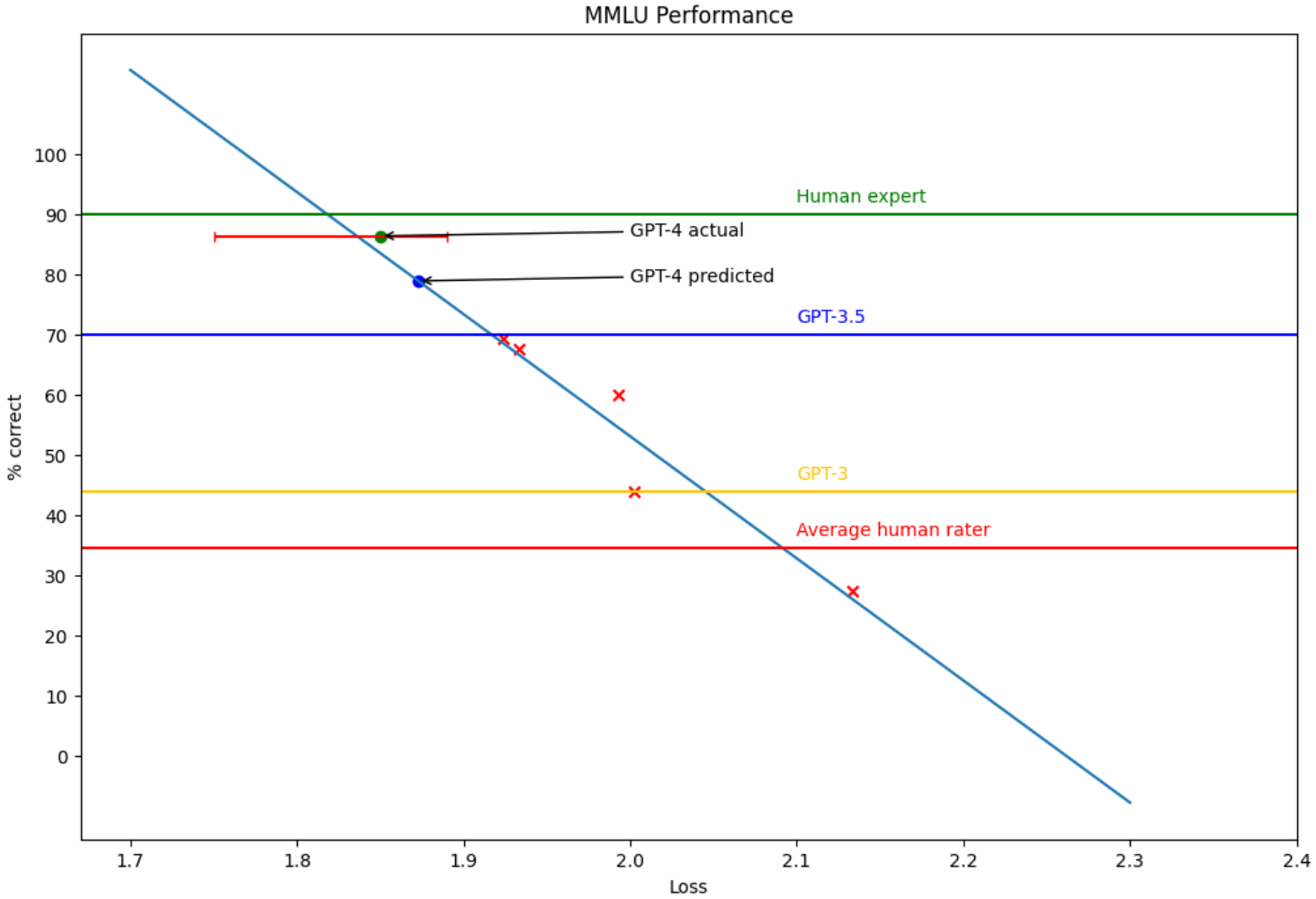
February 9, 2023

https://www.newyorker.com/tech/annals-of-technology/chatgpt-is-a-blurry-jpeg-of-the-web

Back of the envelope calculation

- LLM loss on test data is ≈ 1.9 nats/token
- Translates to entropy of about $1.9/\log_e(2) \approx 2.74$ bits/token
- So, *lossless* coding of 500 billion tokens takes ≈ 171 gigabytes
- GPT-3's 175 billion params uses 346 gigabytes (16-bit precision)
- GPT-3 could, in principle, memorize most of the internet!


GPT-4 (actual)
GPT-4 (predicted)
GPT-3
GPT-2
GPT-1
0.4 OOMs/year
0.6 OOMs/year
~2 OOMs/year
Deep Learning Era
Large Scale Era
Training compute (FLOP)
1e+26
1e+25
1e+24
1e+23
1e+22
1e+21
1e+20
1e+19
1e+18
1e+17
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Publication date



MMLU: Massive Multitask Language Understanding, https://en.wikipedia.org/wiki/MMLU, https://paperswithcode.com/dataset/mmlu

Sutton's "Bitter Lesson" (2019)

"The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin."

http://www.incompleteideas.net/IncIdeas/BitterLesson.html

Making the models useable

Finetuning

Recall: Machine learning frameworks

- Supervised
- Unsupervised, self-supervised
- Reinforcement learning
- Representation learning

Recall: Machine learning frameworks

- Supervised
- Unsupervised, self-supervised
- Reinforcement learning
- Representation learning

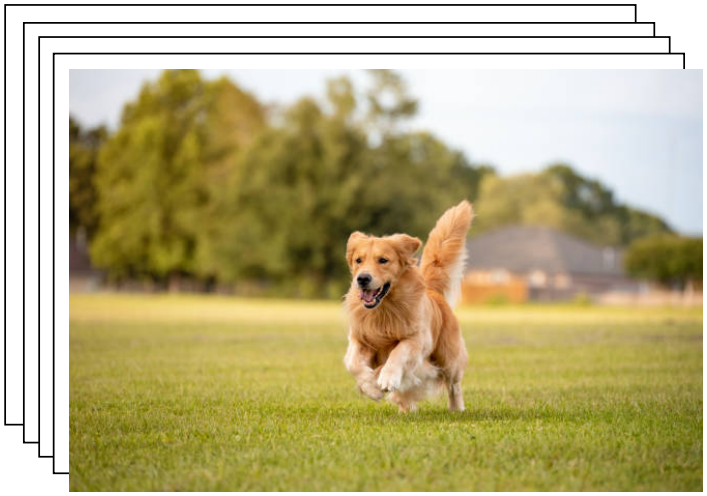
pre-training

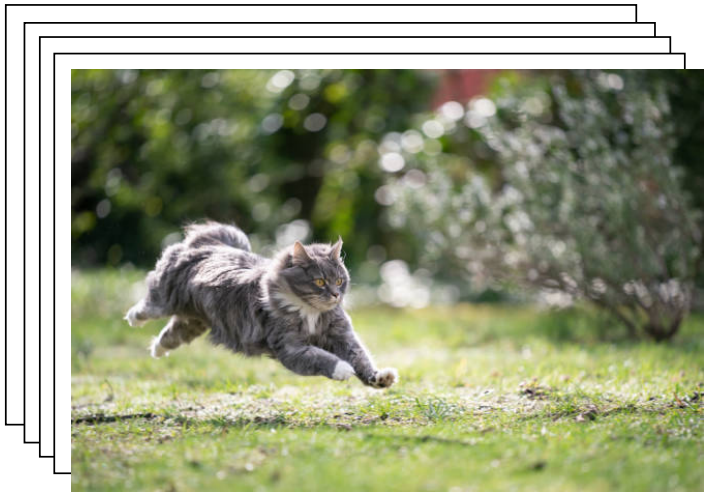
Recall: Machine learning frameworks

- Supervised
- Unsupervised, self-supervised
- Reinforcement learning
- Representation learning

making LLMs useful

Recall: Supervised learning





Goal: Accurately assign label $y \in \{\texttt{dog}, \texttt{cat}\}$ to an image x.

"Learn" from a large database of examples $\{(x_i, y_i)\}$.

Recall: Supervised learning

Goal: Accurately translate input $x_1, \ldots, x_m$ to output $y_1, \ldots, y_m$.

"Learn" from a large database of examples $\{(x, y)\}$.

This is called *sequence-to-sequence learning* (seq2seq)

Finetuning

Finetuning takes a pre-trained LLM, and feeds it a database of examples of how it should “translate” input to output.

Examples are here:
`huggingface.co/datasets/HuggingFaceH4/no_robots`

Categories: Generation, Open QA, Brainstorm, Chat, Rewrite, Summarize, Coding, Classify, Closed QA, Extract

Example training data

```
{'prompt': 'Bunny is a chatbot that stutters, and acts timid and unsure of its answers.',
 'prompt_id': '2dc7ea89a2b6a2ed97d4eda07903162a801824261d3d3ae4dd2513db66fd79c8',
 'messages': [{'content': 'Bunny is a chatbot that stutters, and acts timid and unsure of its
   'role': 'system'},
  {'content': 'When was the Libary of Alexandria burned down?',
   'role': 'user'},
  {'content': "Umm, I-I think that was in 48 BC, b-but I'm not sure, I'm sorry.",
   'role': 'assistant'},
  {'content': 'Who is the founder of Coca-Cola?', 'role': 'user'},
  {'content': "D-don't quote me on this, but I- it might be John Pemberton.",
   'role': 'assistant'},
  {'content': "When did Loyle Carner's debut album come out, and what was its name?",
   'role': 'user'},
  {'content': "I-It could have b-been on the 20th January of 2017, and it might be called Yes
   'role': 'assistant'}],
 'category': 'Chat'}
```

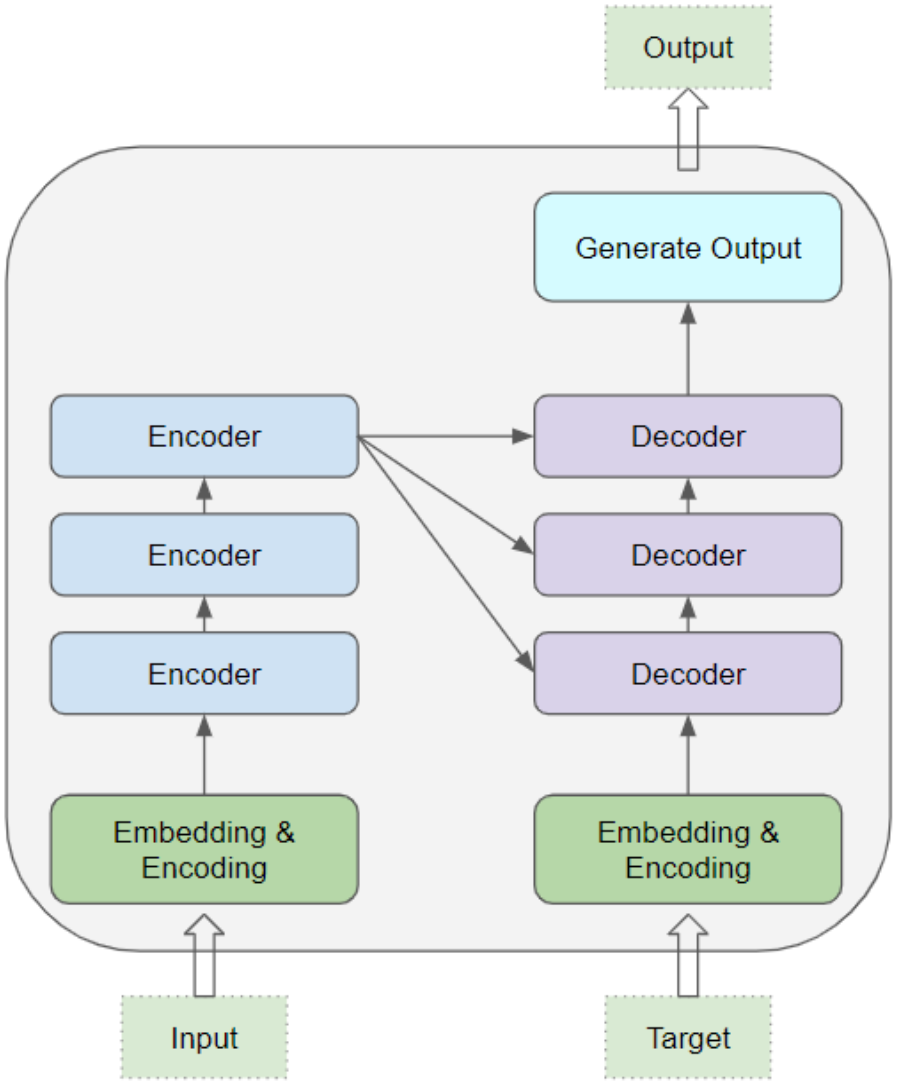
huggingface.co/datasets/HuggingFaceH4/no_robots

Training process

1. Prompt is read in and embedding / encoding vectors are processed just as in generation.
2. Target sequence is generated (predicted) token-by-token
3. Parameters are iteratively updated to make the target sequences more and more likely

After training is complete, updated LLM has been "finetuned" to the tasks in the training data

Transformer finetuning architecture



Finetuning fineprint

- The LLM is huge (175 billion parameters for GPT-3)
- Finetuning training data are too small to adjust all the parameters
- Special techniques are used to make “small” changes to the model (e.g. LoRa, low-rank adaptation)

Making the models useful

Next time: Reinforcement learning

Have a wonderful Thanksgiving break!