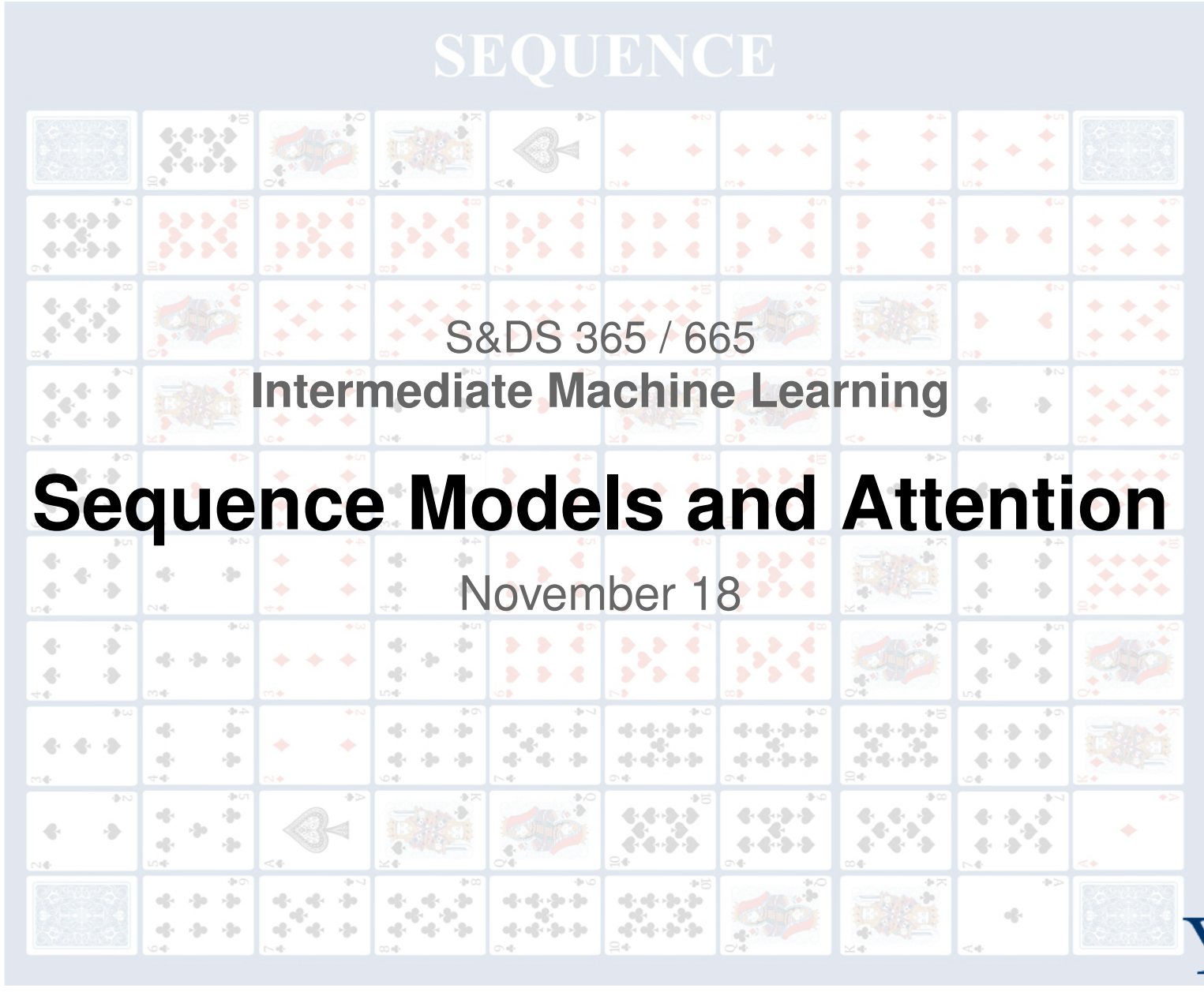


S&DS 365 / 665

Intermediate Machine Learning

Sequence Models and Attention

November 18

Yale

For Today

Sequence models

- Memory circuits: GRUs (recap)
- Sequence-to-sequence models
- Attention mechanisms
- Transformers

But first some reminders

- Quiz 5 (last!) on Wednesday, Nov 20 (RL, HMMs, RNNs, GRUs)
- Assn 5 out; due Dec. 4
- Final exam: Sunday Dec 15, 2pm
- Practice exams are posted
- Review sessions TBA

---

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

Recurrent neural networks

In an RNN, current output generated from a distributed state—a vector of neurons

- State is a deterministic, a nonlinear function of previous state and current input symbol.
- Dependence on earlier x_t's is encoded in the state
- Output generated stochastically by sampling from

$$\text{Softmax}(\beta^T h_t)$$

where $h_t = f(x_1, \ldots, x_{t-1})$ is the state vector.

“Vanilla” RNNs

In principle the state h_t can carry information from far in the past.

In practice, the gradients vanish (or explode) so this doesn’t really happen

We need other mechanisms to “remember” information from far away and use it to predict future words

“Vanilla” RNNs

State is updated according to

$$h_t = \tanh\left(W_{hh} h_{t-1} + W_{hx} x_t + b_h\right)$$

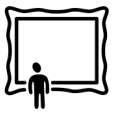
We modify this with two types of “neural circuits” for storing information to be used downstream

LSTMs and GRUs

Both LSTMs and GRUs have longer-range dependencies than vanilla RNNs.

We went through this in detail for GRUs, which are simpler, more efficient, and more commonly used

Gated recurrent units (GRUs)



High level idea:

- Learn when to update hidden state to “remember” important pieces of information
- Keep them in memory until they are used
- Reset or “forget” this information when no longer useful

GRUs

GRUs make use of “gates” denoted by Γ (Greek G for “Gate”)

$\Gamma = 1$: “the gate is open” and information flows through

$\Gamma = 0$: “the gate is closed” and information is blocked

GRUs

Two types of gates are used:

Γ^u: When open, information from long-term memory is propagated. When closed, information from local state is used.

Γ^r: When closed, the local state is reset. When open, the state is updated as in a “vanilla” RNN.

Note: These are usually called the “update” and “reset” gates.

GRUs

The state evolves according to

$$c_t = \tanh\left(W_{hx}x_t + \Gamma_t^r \odot W_{hh}h_{t-1} + b_h\right)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

GRUs

The state evolves according to

$$c_t = \tanh\left(W_{hx}x_t + \Gamma_t^r \odot W_{hh}h_{t-1} + b_h\right)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

- c_t is the "candidate state" computed using the usual "vanilla RNN" state, after possibly resetting some components.
- When the long-term memory gate is open ($\Gamma^u = 1$), the information gets sent through directly. *This deals with vanishing gradients*.
- The gates are multi-dimensional, applied componentwise
- Prediction of the next word is made using h_t.

GRUs

Everything needs to be differentiable, so the gate is actually “soft” and between zero and one.

The gates are computed as

$$\Gamma_t^u = \sigma\left(W_{ux} x_t + W_{uh} h_{t-1} + b_u\right)$$

$$\Gamma_t^r = \sigma\left(W_{rx} x_t + W_{rh} h_{t-1} + b_r\right)$$

where σ is the sigmoid function.

Putting it together

GRU state update equations

$$\Gamma_t^u = \sigma\left(W_{ux} x_t + W_{uh} h_{t-1} + b_u\right)$$

$$\Gamma_t^r = \sigma\left(W_{rx} x_t + W_{rh} h_{t-1} + b_r\right)$$

$$c_t = \tanh\left(W_{hx} x_t + \Gamma_t^r \odot W_{hh} h_{t-1} + b_h\right)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

There are minor variants on this architecture that are sometimes used.

Putting it together

The reset gate is sometimes moved inside the linear map

GRU state update equations

$$\Gamma_t^u = \sigma\left(W_{ux} x_t + W_{uh} h_{t-1} + b_u\right)$$

$$\Gamma_t^r = \sigma\left(W_{rx} x_t + W_{rh} h_{t-1} + b_r\right)$$

$$c_t = \tanh\left(W_{hx} x_t + W_{hh}\left(\Gamma_t^r \odot h_{t-1}\right) + b_h\right)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

There are minor variants on this architecture that are sometimes used.

GRUs: Example

Example:

```
The leaves, as the weather turned cold in New
Haven, fall silently from the trees in November.
```

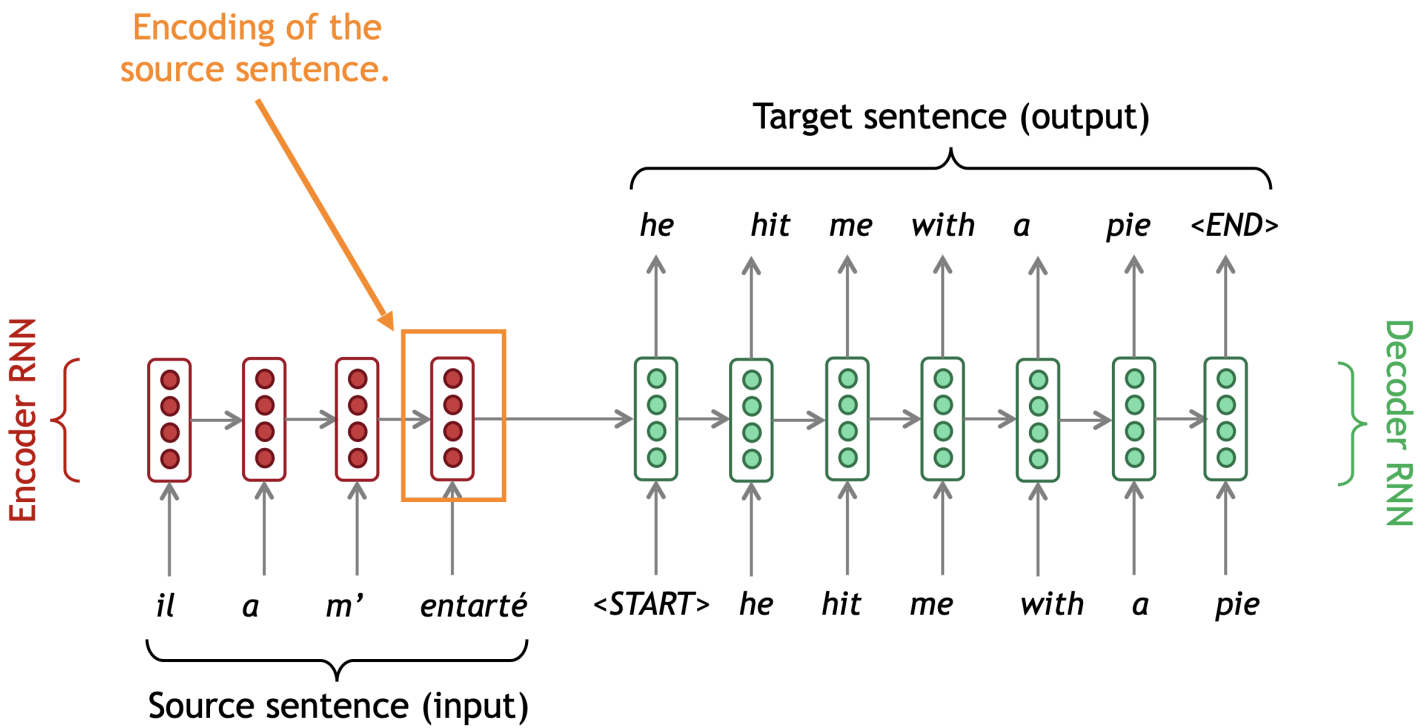
We want to keep `leaves` in memory. It's the subject of the sentence, and plural — syntax

It also has a "foliage" meaning that is relevant when we predict the words `trees` and `November` — semantics

We stepped through how the GRU handles this (conceptually)

Sequence-to-sequence models

- Important in translation
- Uses two RNNs (GRUs or LSTMs): Encoder and Decoder



"Sequence to sequence learning with neural networks," Sutskever et al. 2014, https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf. They also reversed the order of the input. Figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Sequence-to-sequence models

- The goal of Seq2seq is to estimate the conditional probability

$$p(y_1, \ldots, y_T \mid x_1, \ldots, x_S)$$

- Encoder RNN computes the fixed dimensional representation $h(x_{1:S})$; decoder RNN then computes

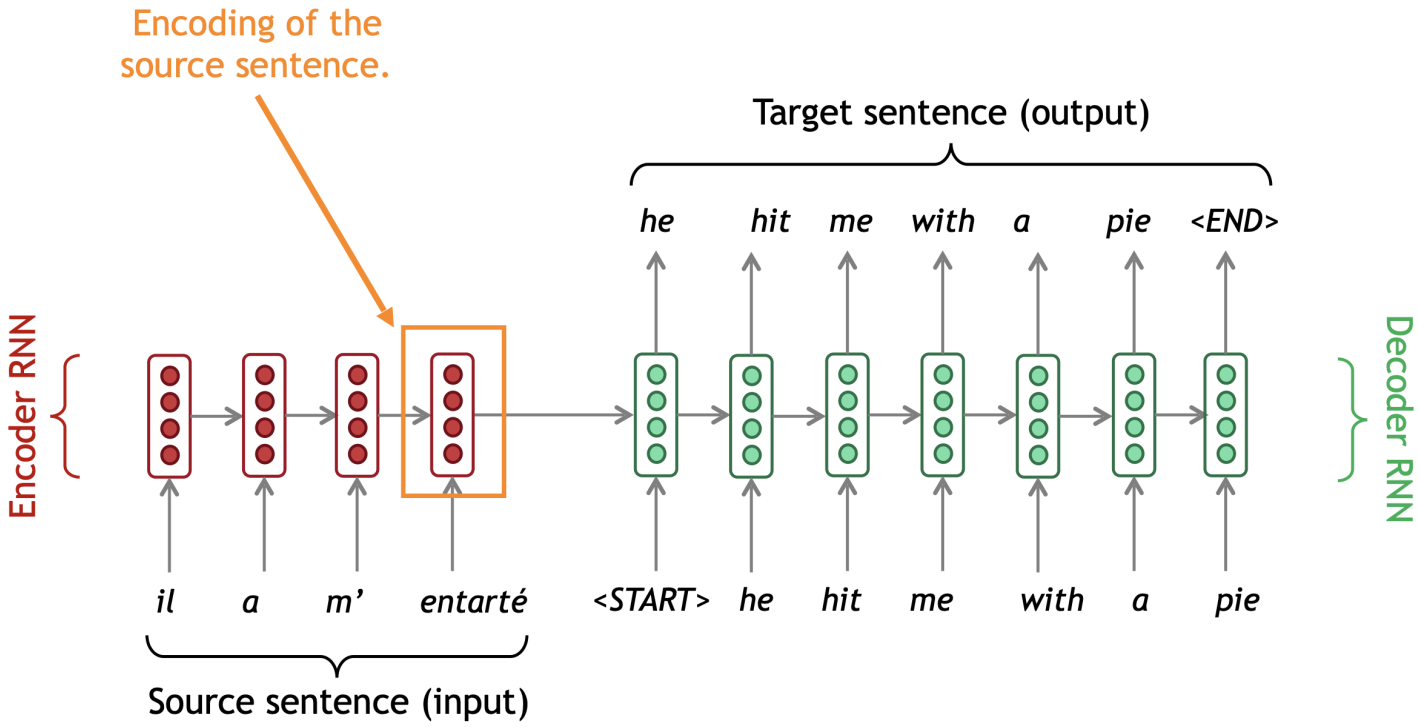
$$\prod_{t=1}^{T} p(y_t \mid h, y_1, \ldots, y_{t-1})$$

- Original paper uses 4-layer LSTMs with 1000 neurons in each layer; trained for 10 days for a machine translation task.

In the example of the previous slide, the state h is the red vector over entarté. Recall: This is very similar to what we did with LaTeX equation models conditioned on topics.

Sequence-to-sequence models

This results in a “bottleneck” problem—all the information needs to be funneled through that final state.



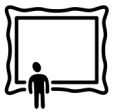
Important modification: Attention

Pay attention!

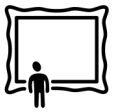
entarté!

https://en.wikipedia.org/wiki/Pieing

Attention

 On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

Attention



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

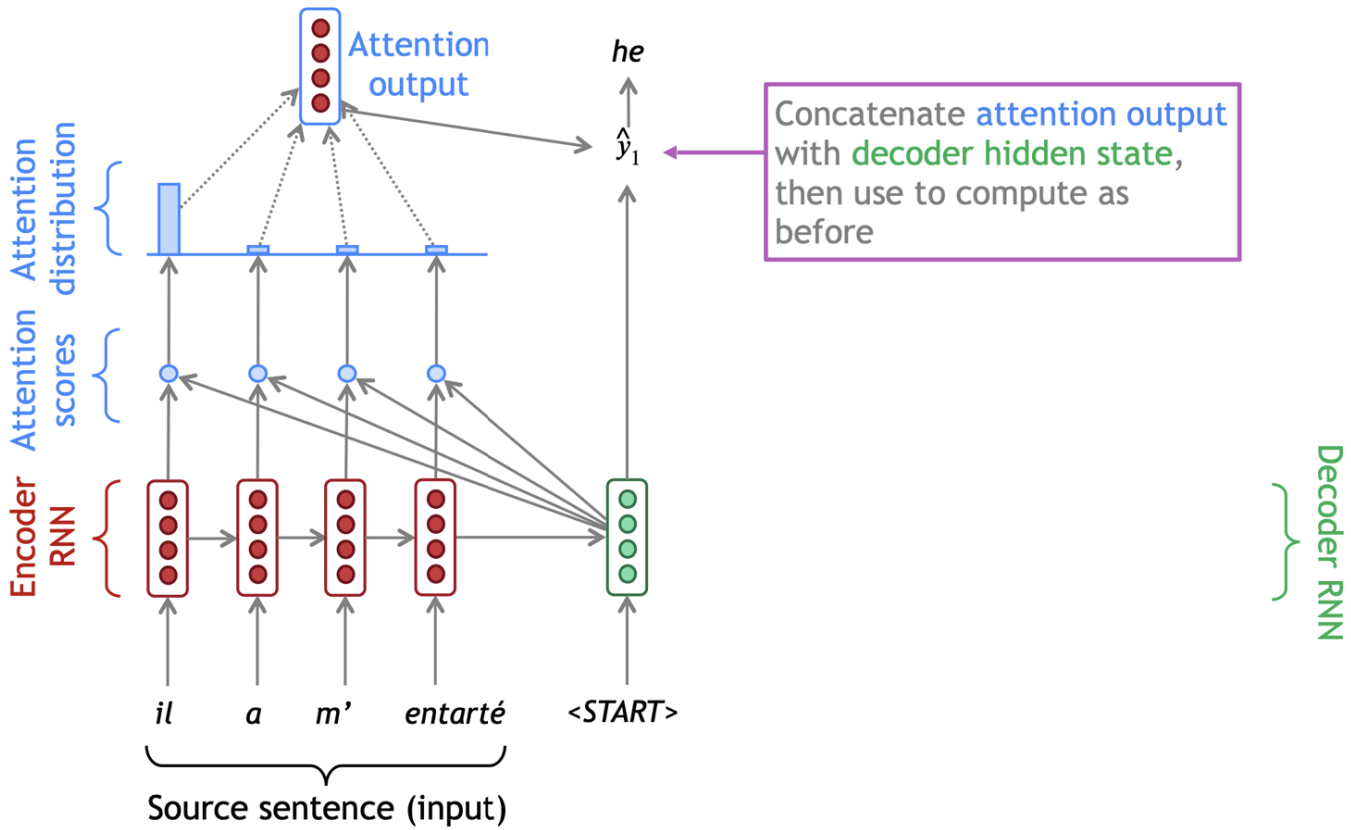
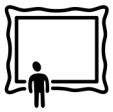


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

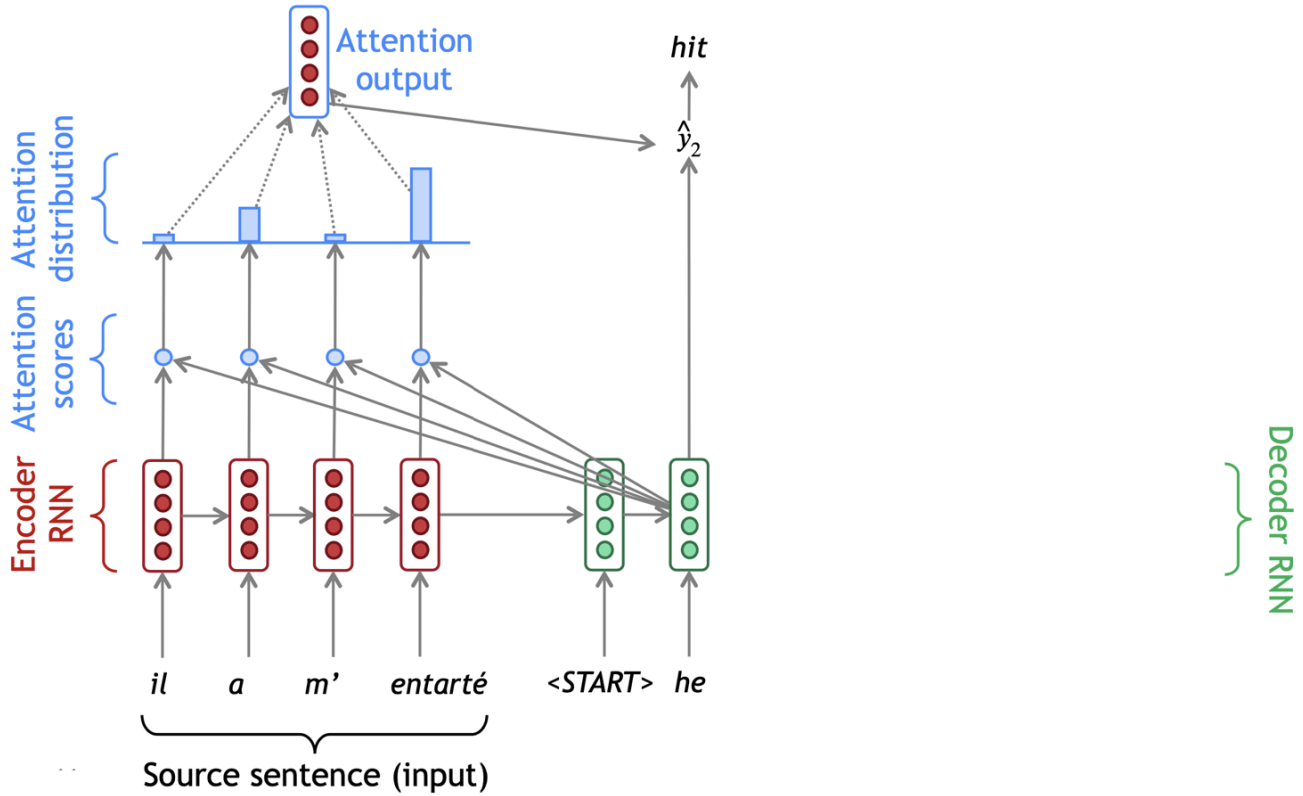
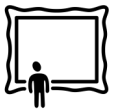


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

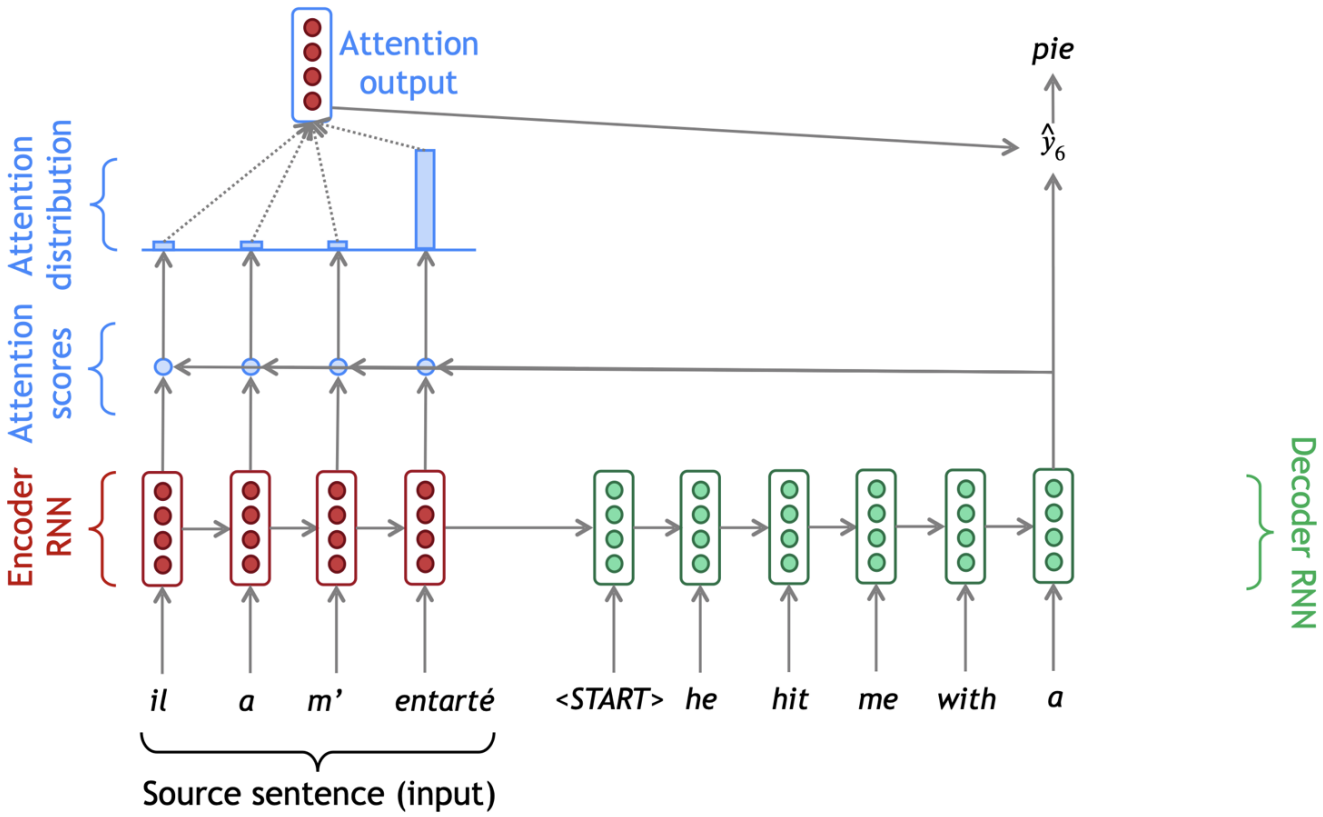
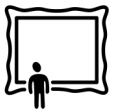


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention

 On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

The two elephants played with an orange ball

The two elephants played with an orange ball

 → orange

 → ball

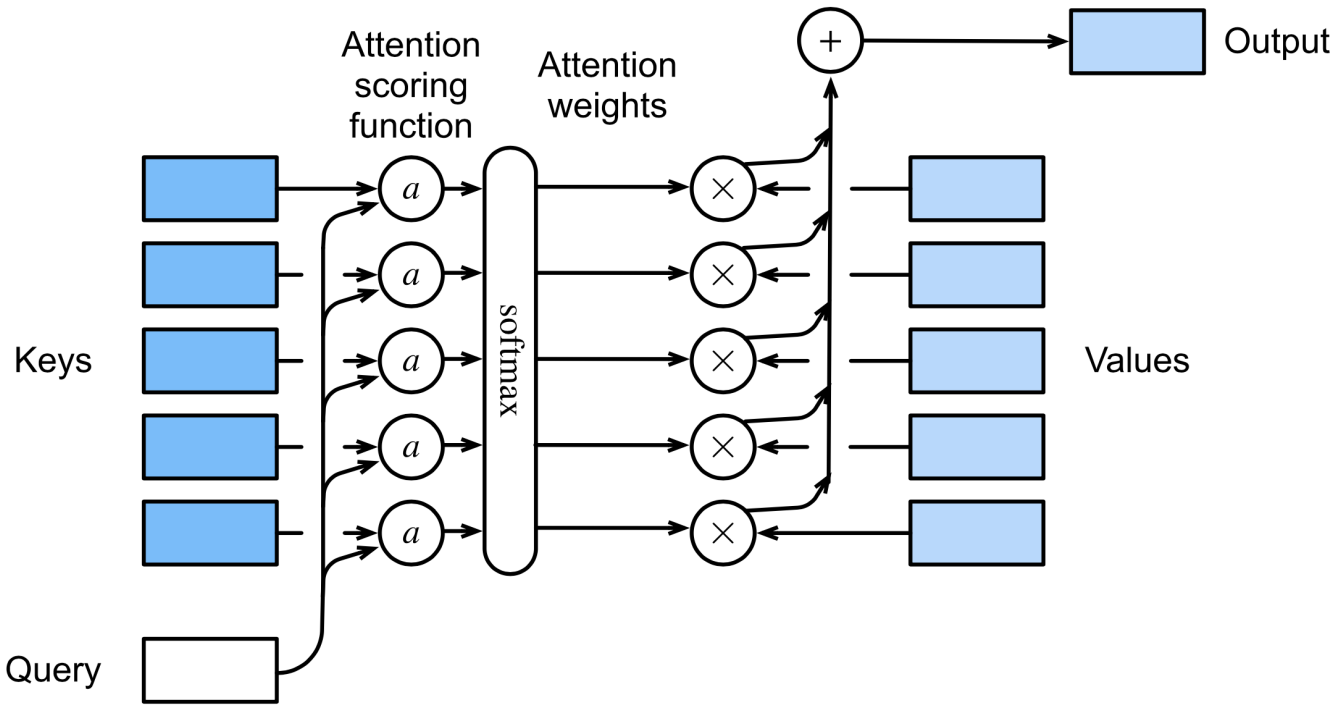
Attention in terms of query/key/values

Basic idea behind attention mechanisms:

- Neural networks so far: Hidden activations are linear combinations of input activations, followed by nonlinearity: $h = \varphi(Wu)$
- A more flexible model:
  - We have a set of m feature vectors or values $V \in \mathbb{R}^{m \times v}$
  - The model dynamically chooses which to use
  - based on how similar a query vector $q \in \mathbb{R}^q$ is to a set of m keys $K \in \mathbb{R}^{m \times k}$.
  - If q is most similar to key i, then we use value (feature) v_i.

Attention in terms of query/key/values

Basic idea behind attention mechanisms:



Attention in terms of query/key/values

Basic idea behind attention mechanisms:

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

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

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

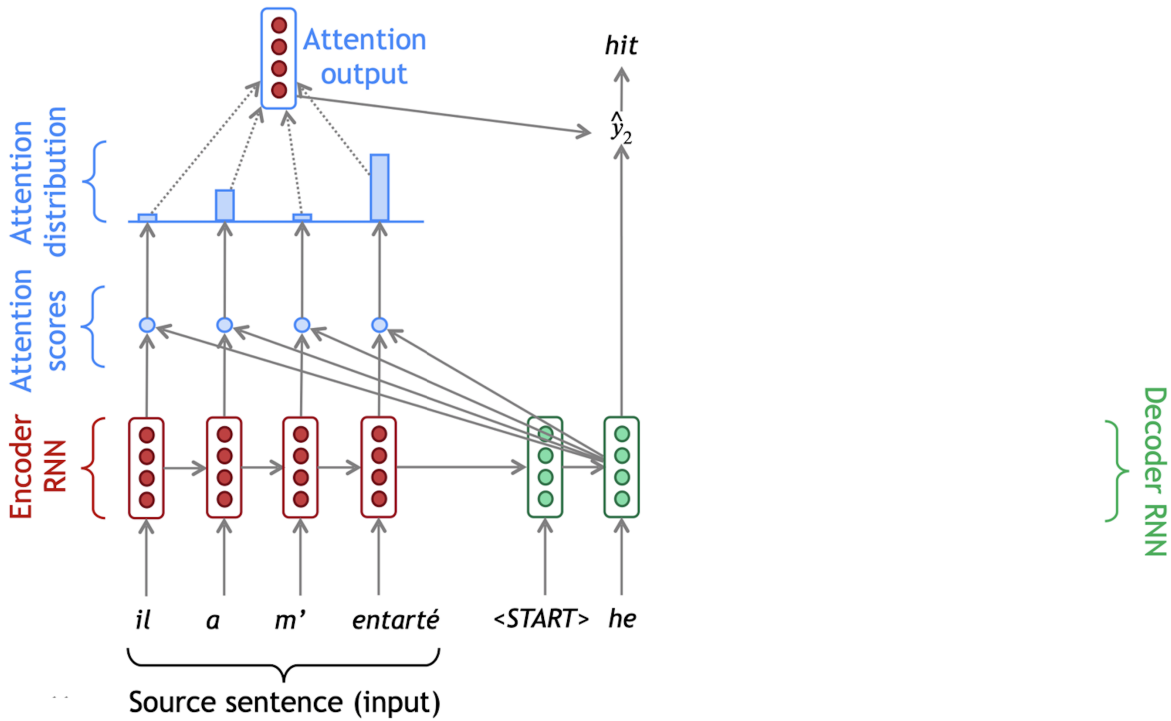
and

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

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

Retour d'entarté

In the *entarté* example, the keys and values are the red state vectors, the queries are the green state vectors



Transformers

The current state-of-the-art is based on *transfomers*

- Attention is the key ingredient
- Rather than processing sequences word-by-word, transformers handle larger chunks of text at once
- The separation between words matters less

Key steps

- *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 ≈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”

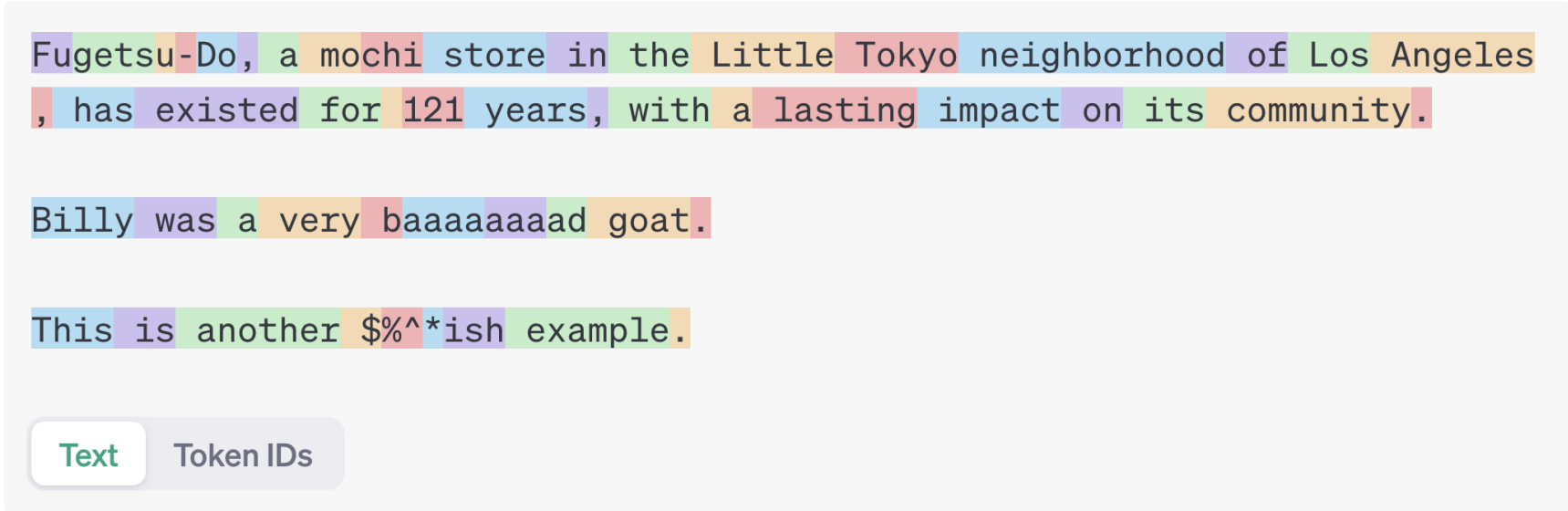
Tokenization

```
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 baaaaaaaad goat.

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

This is what we see

platform.openai.com/tokenizer

Tokenization



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

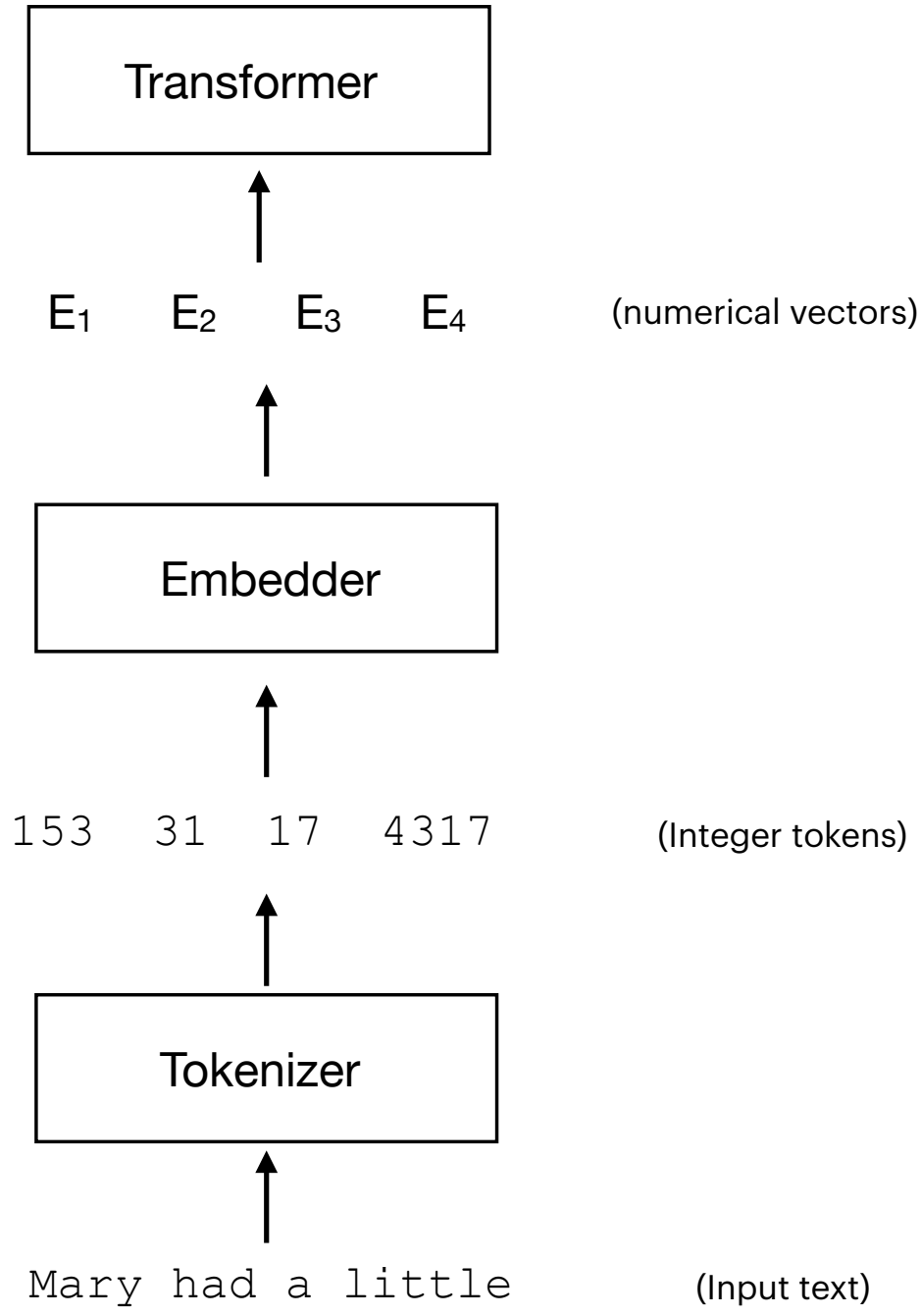
- *Reversible:* Can convert from tokens back to original text
- *General:* Works on arbitrary text, even very different from training
- *Compressive:* Each token is about 4 bytes
- *Statistical:* Will break strings into frequently occurring pieces

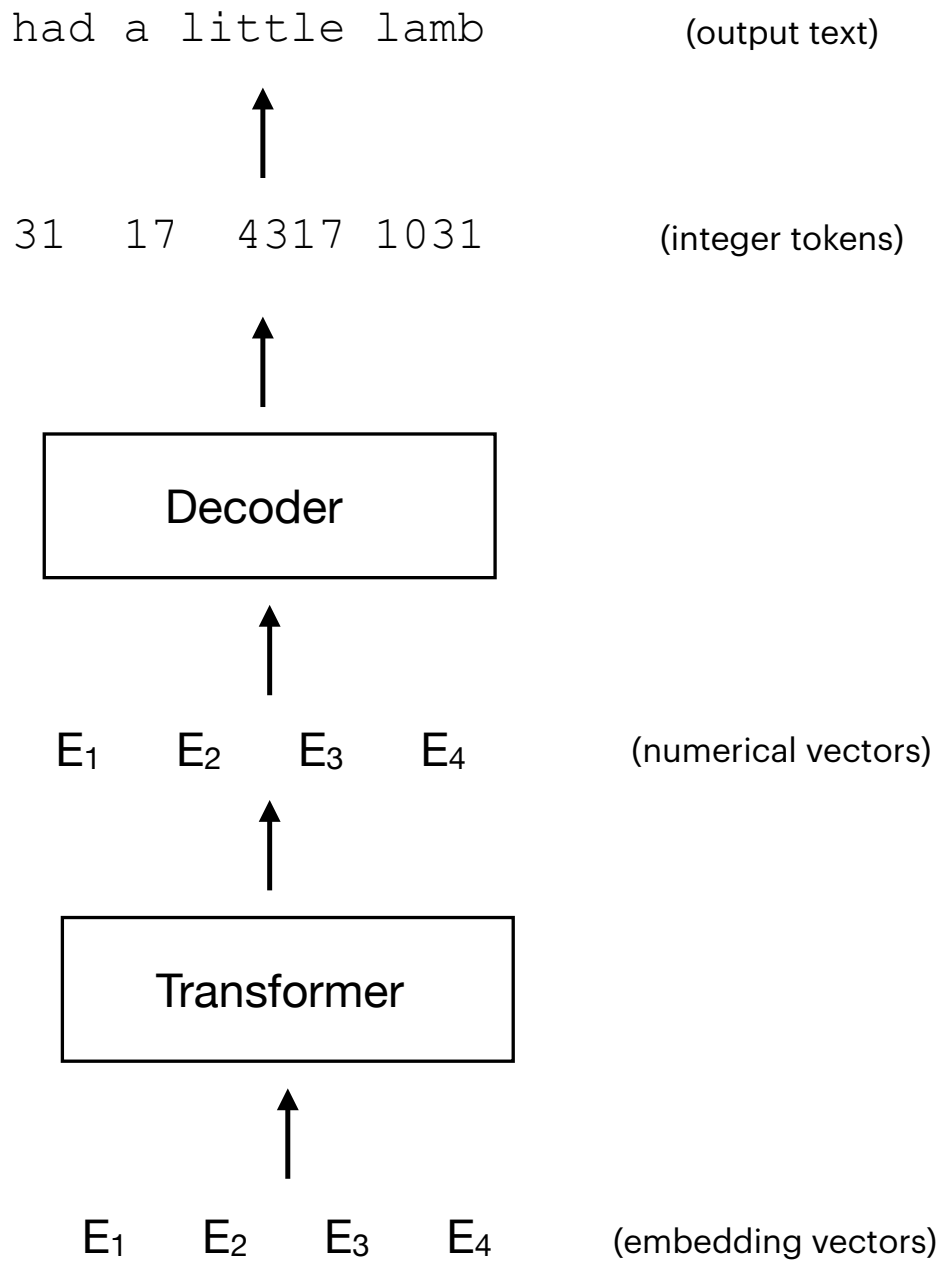
BPE Algorithm – Byte Pair Encoding

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)

The key problem is integrating information across long distances

The leaves, as the weather turned cold in New Haven, [ ] silently from the [ ] in [ ].

The key problem is integrating information across long distances

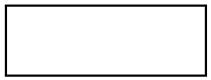
The leaves, as the weather turned cold in New Haven, fall silently from the trees in October.

The key problem is integrating information across long distances

`The leaves, as the weather turned cold in New Haven, [fall] silently from the [trees] in [October].`

To predict `fall` and `trees`, the model needs to refer back to `leaves` in memory (context):

- It's the subject of the sentence, and plural — syntax
- It has a "foliage" meaning — semantics

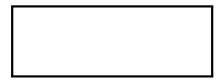
The leaves, as the weather turned cold in New Haven, 

`The leaves, as the weather turned cold in New Haven,` [ ]

Intuition:

Embedding vector $\phi(\texttt{leaves})$ is added to the vector used to fill in the first blank.

Since $\phi(\texttt{leaves})$ is similar to $\phi(\texttt{fall})$, this boosts up the chances of generating `fall`

The leaves, as the weather turned cold in New Haven, fall silently from the 

`The leaves, as the weather turned cold in New Haven, fall silently from the` ☐

Intuition:

Embedding vectors $\phi($`leaves`$)$ and $\phi($`fall`$)$ are then added to the vector used to fill in the next blank.

This boosts up the chances of generating `trees`

The use of “teacher forcing” fills in the actual word when predicting later words.

Attention

- Which embeddings are added to current position is determined by *Attention*
- Attention is essentially a (very large) system of linear equations
- The weights in the linear system are optimized to extract the most predictive properties

Attention

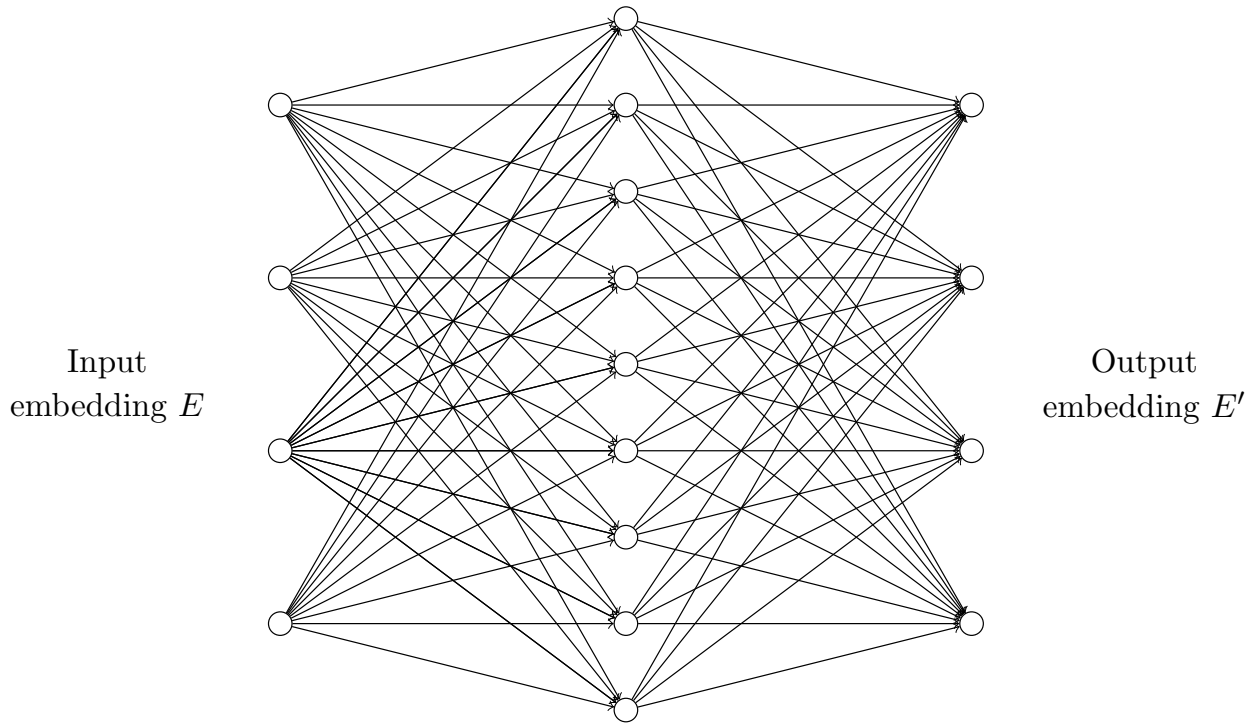
- Which embeddings are added to current position is determined by *Attention*
- Attention is essentially a (very large) system of linear equations
- The weights in the linear system are optimized to extract the most predictive properties
- Attention is organized into multiple “heads”
  - Some may capture “syntax” (`subject` $\rightarrow$ `jverb`)
  - Some may capture “semantics” (`leaves` $\rightarrow$ `trees`)
  - But they are generally very difficult to interpret!

More intuition

- Different attention vectors are concatenated
- Capture different meanings of the sequence
- Words “hungry” and “sweet” are predicted using all of them

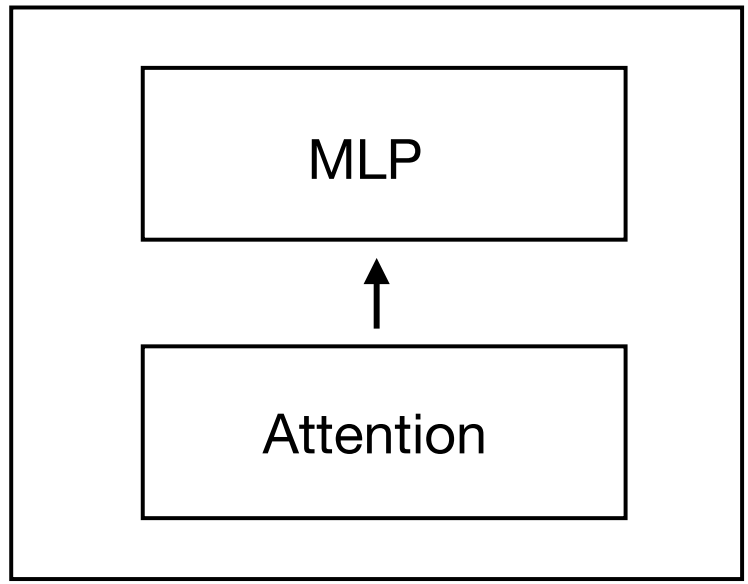
Adding nonlinearities

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

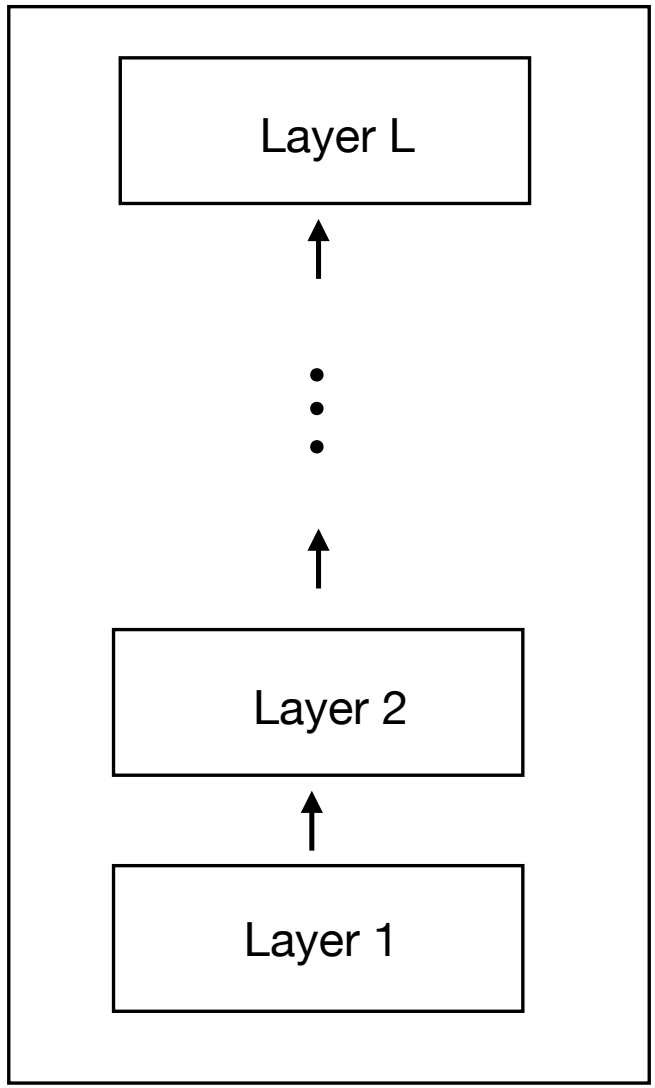


Creates new “features” to be used as embeddings in the next layer

Transformer Layer

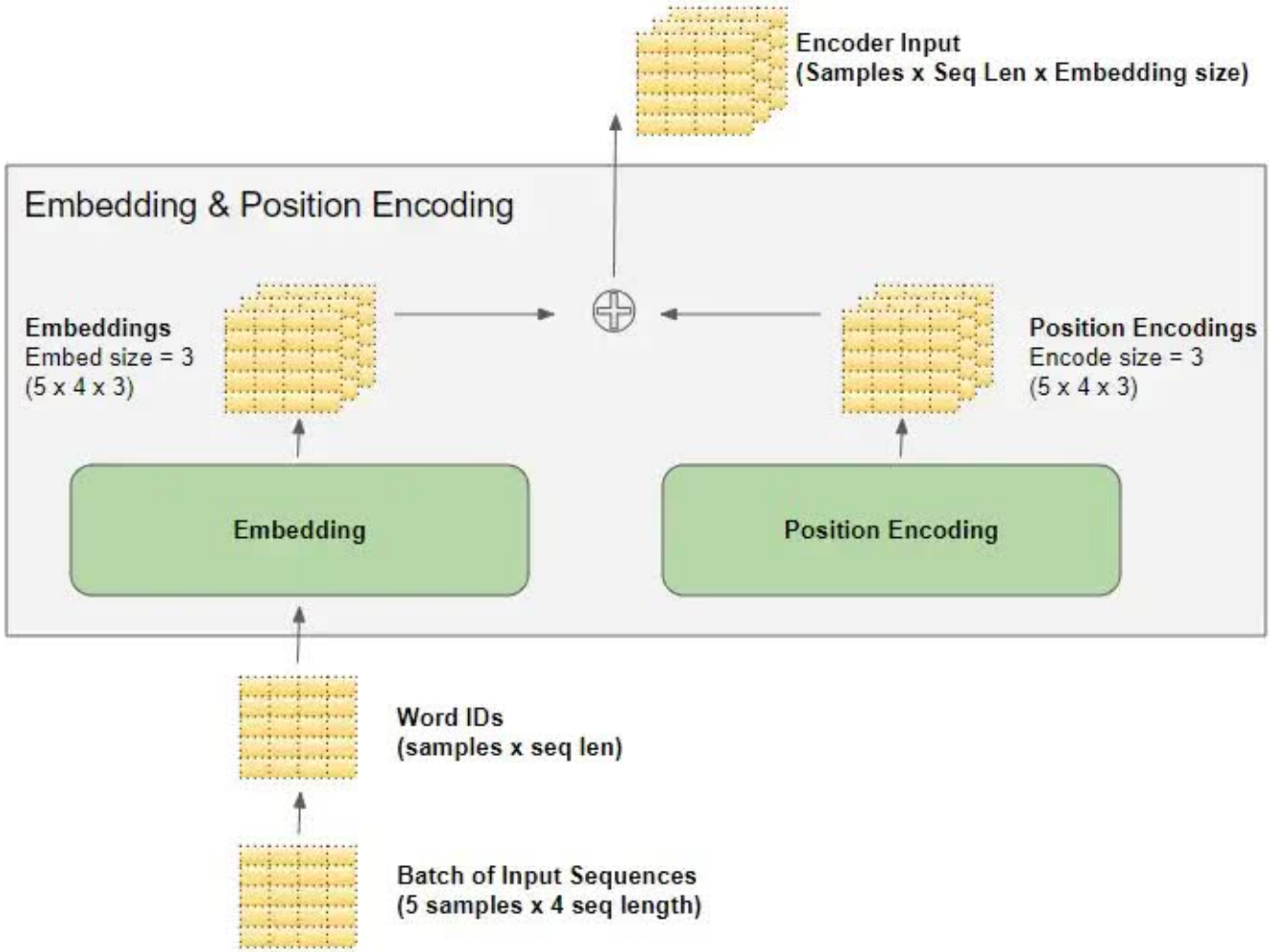


Transformer



More detail on attention

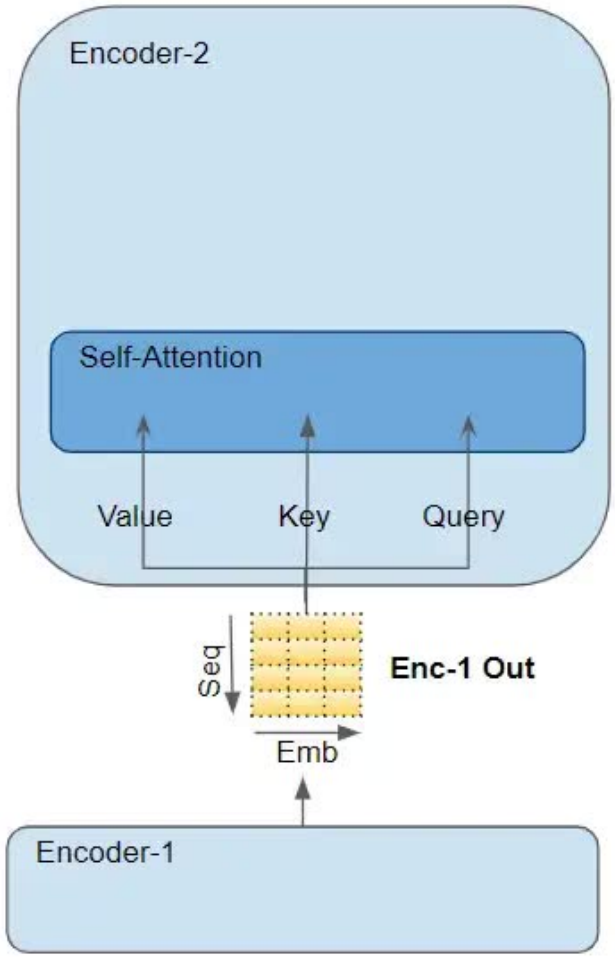
Text processed in batches; embedding stage looks like this:



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

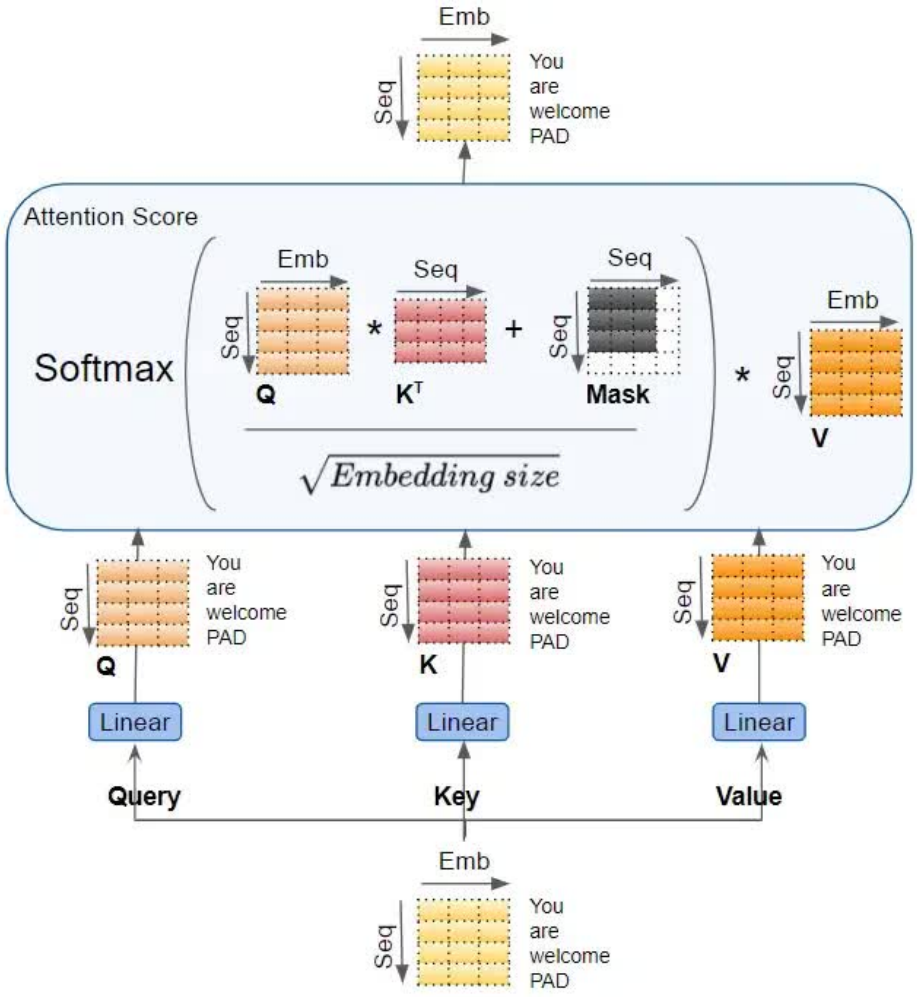
Multihead Attention

Detail of self-attention component

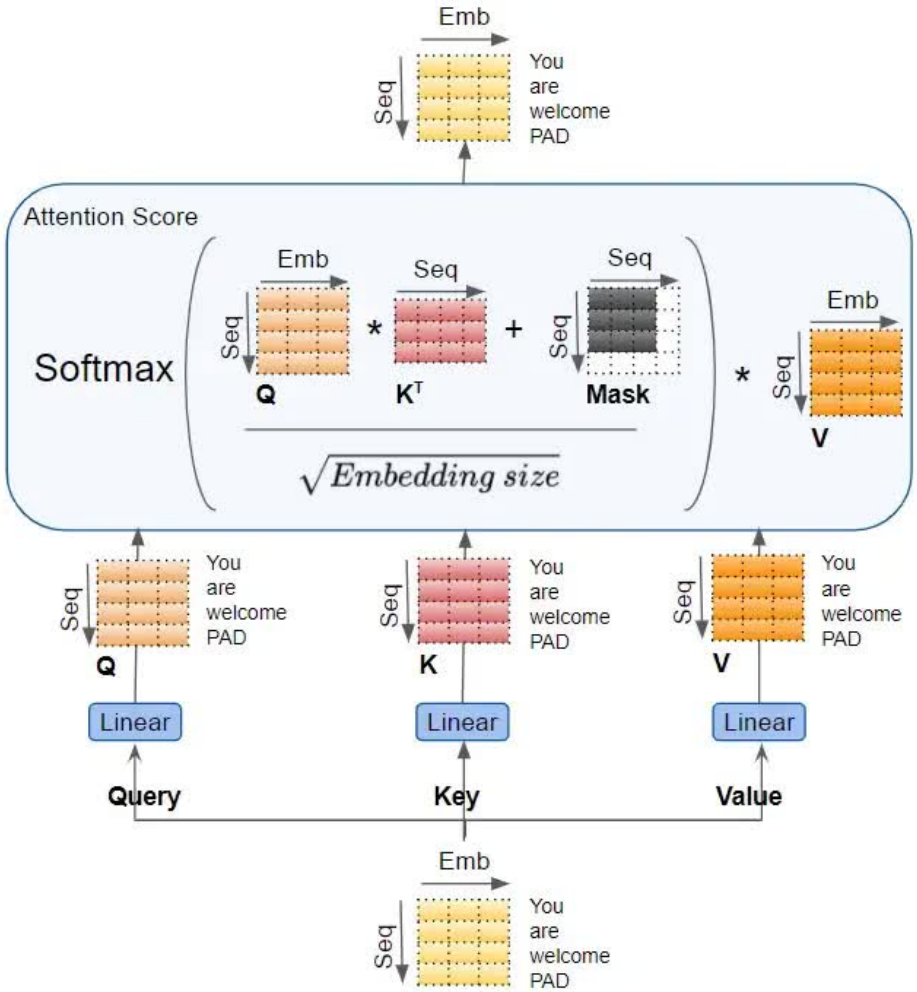


Multihead Attention

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

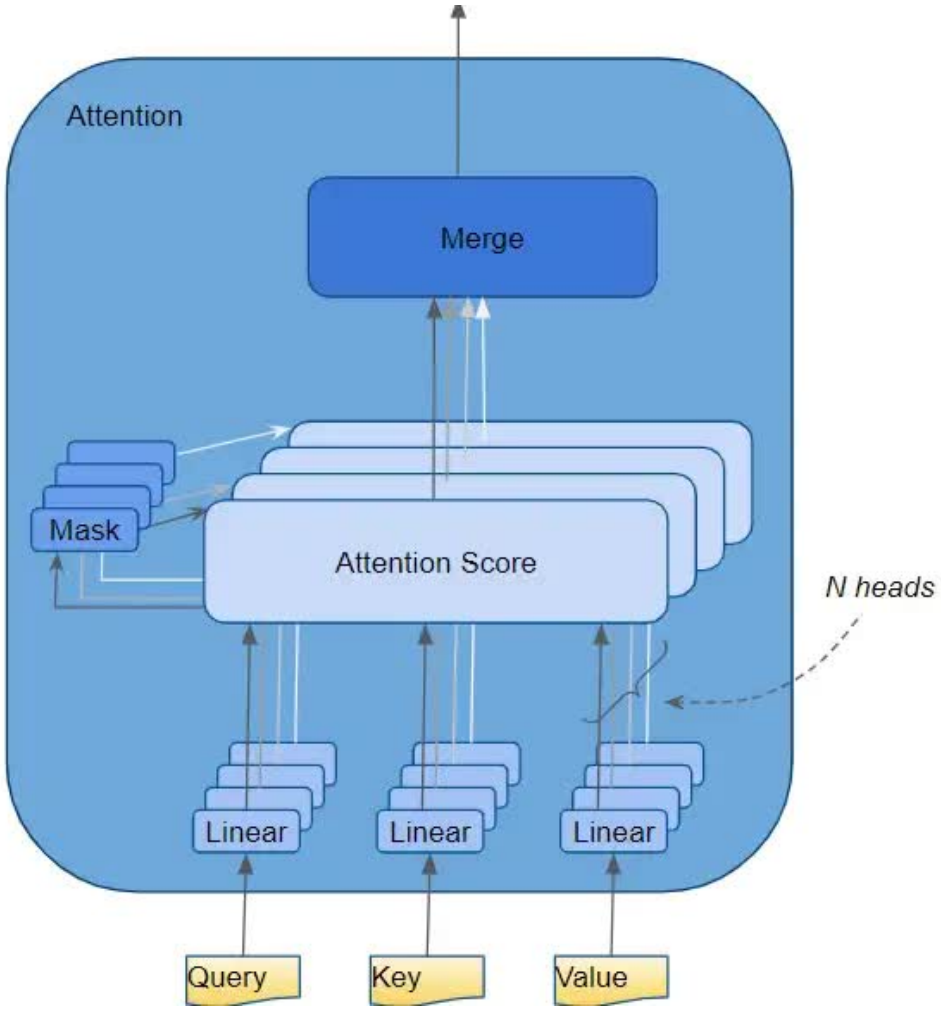


Multihead Attention



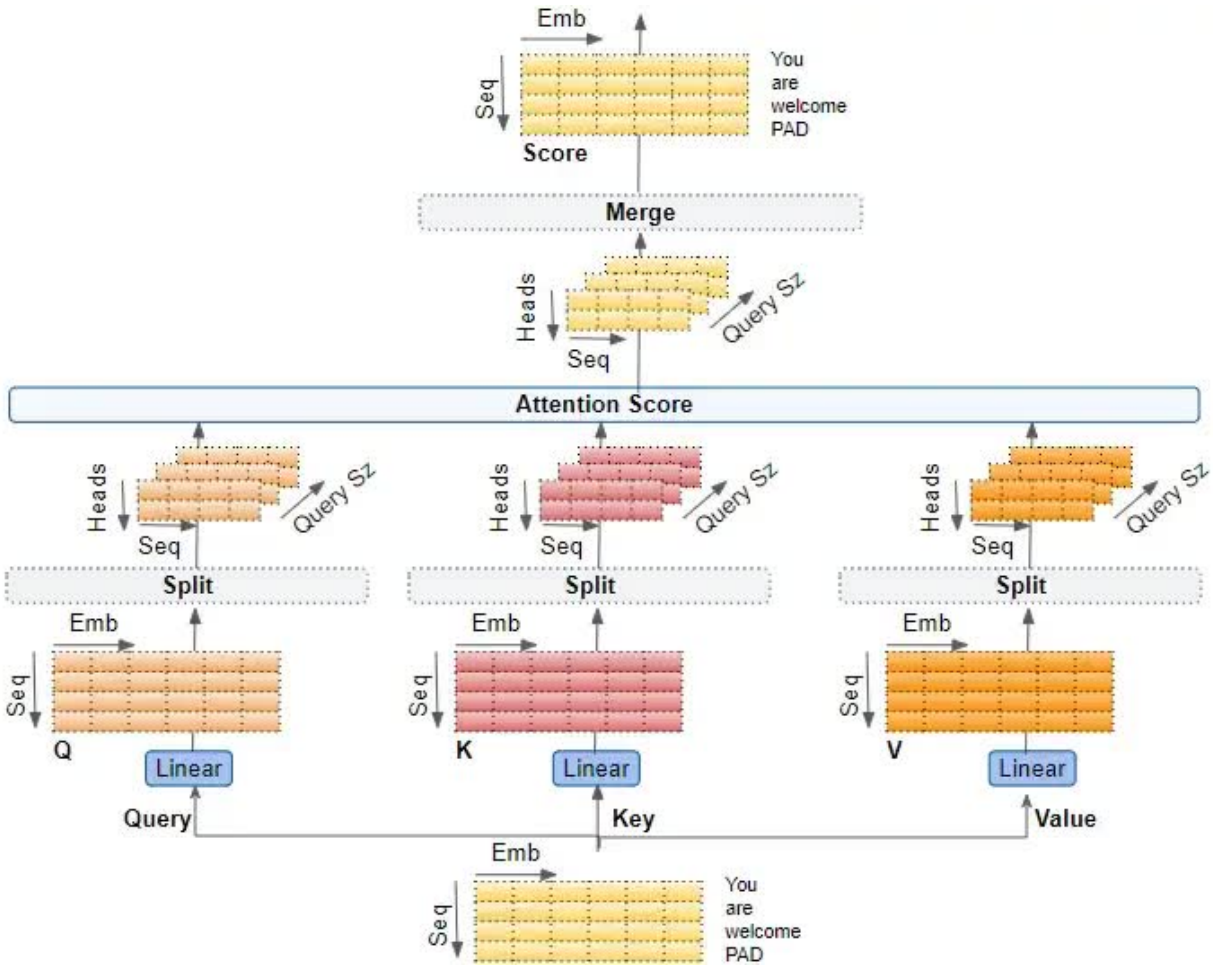
Mask is used in decoder to prevent use of future words

Multihead Attention



Multiple attention vectors are computed, then merged (concatenated)

Multihead Attention



Multiple attention embeddings are computed, then merged

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

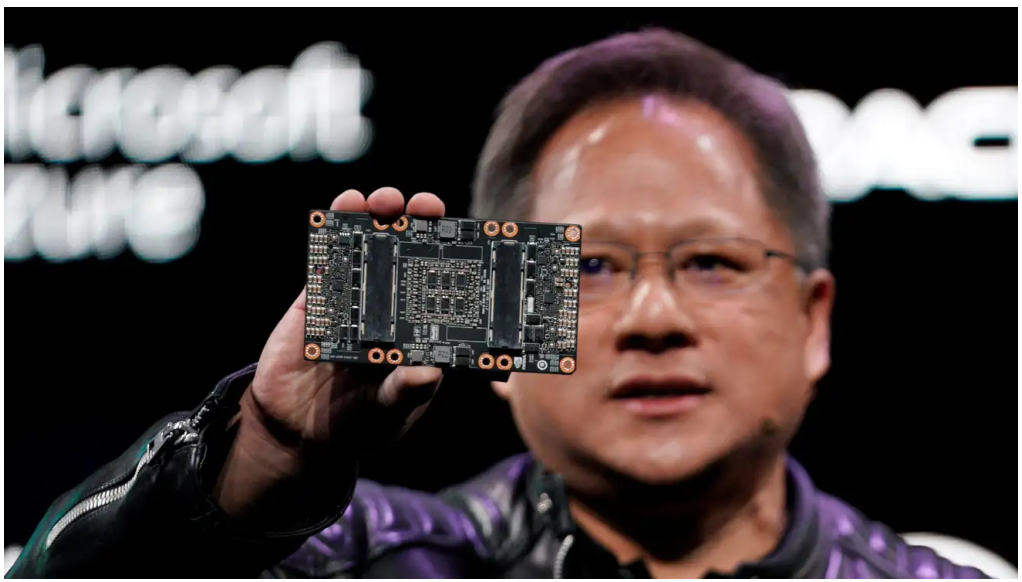


Photo from Reuters; https://stackoverflow.com/questions/51344018/why-can-gpu-do-matrix-multiplication-faster-than-cpu

Back to intuition

- Different attention vectors are concatenated
- Capture different meanings of the sequence
- Words “hungry” and “sweet” are predicted using all of them

The residual stream

For *each* occurrence of a token in context, the GPT-3 Transformer maps it to 97 *different* embeddings (aka encodings)

- The first embedding represents the general token, across all contexts — 12,288 trainable numbers
- Each of the 96 layers then transforms this embedding using Attention and an MLP
- Result is 96 $\times$ 12,288 numbers for each token occurrence — more than one million total numbers (2 million bytes)
- Not *directly* interpretable by humans; but can be probed

---

www.anthropic.com/research/mapping-mind-language-model

The residual stream

The context length for GPT-3 is 2,048 tokens

- Total numbers for each context: 2,048 $\times$ 1.1 million $\approx$ 2 billion
- The numbers for previous layer are added to the numbers for the next layer
- This is called the *residual stream*
- Number of parameters in model: 175 billion
  - We'll describe how to count parameters next

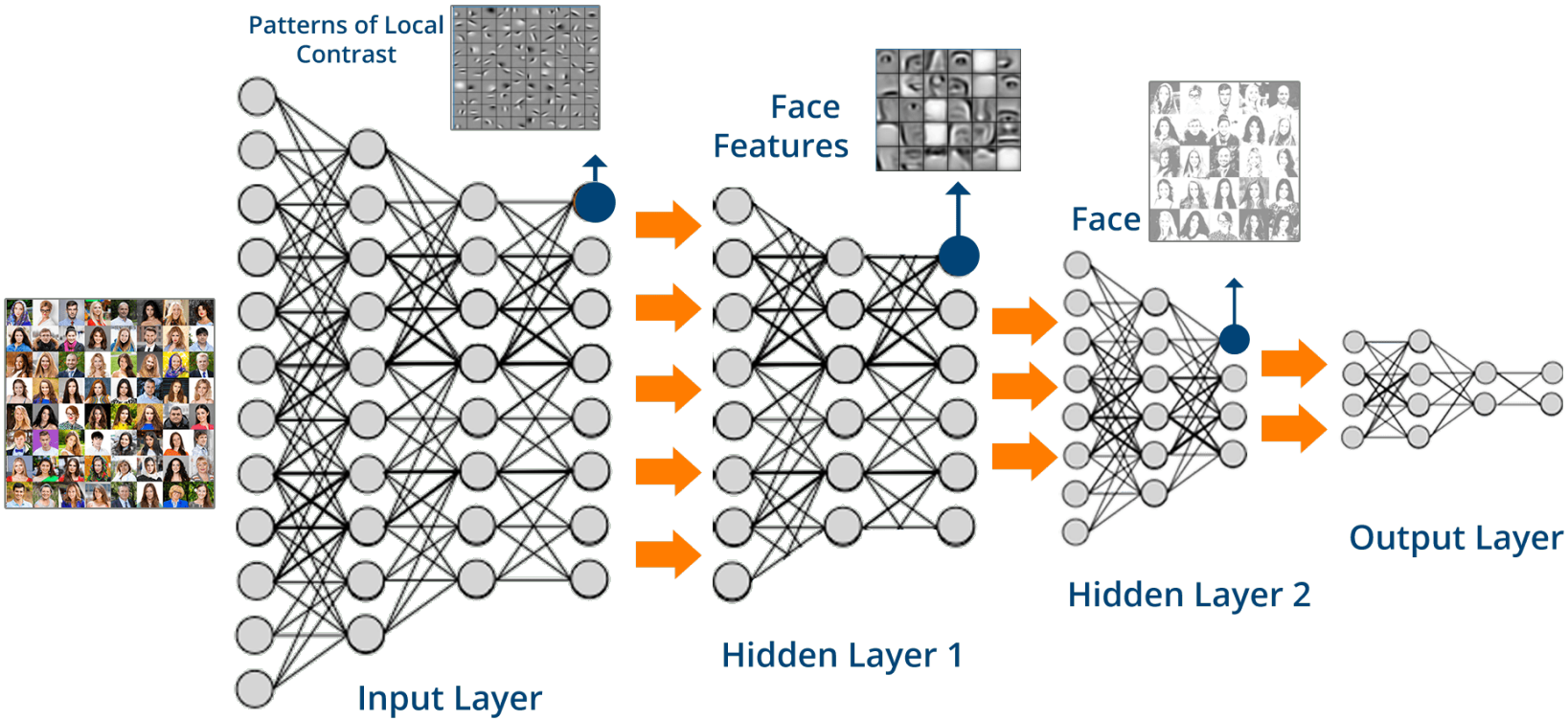
---

www.anthropic.com/research/mapping-mind-language-model

Transformer architecture

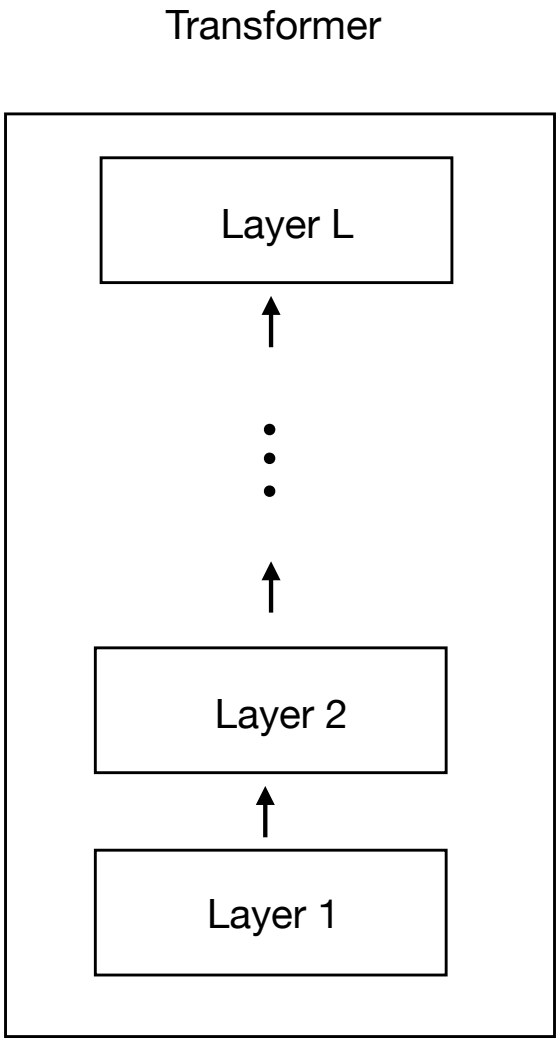
1. Input: sequence of tokens $t_1, \ldots, t_n$
2. $x = \texttt{embed\_tokens}(t)$
3. $x \leftarrow x + \texttt{positional\_encoding}([\text{n}])$
4. Iterate for $\ell \in [L]$:
    i. $x \leftarrow x + \texttt{Attention}_\ell(x)$
    ii. $x \leftarrow x + \texttt{MLP}_\ell(x)$
5. Output: $y = \texttt{prediction\_head}(x)$

Increasing specificity of patterns



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

Increasing specificity of patterns



features of tokens
spanning long range

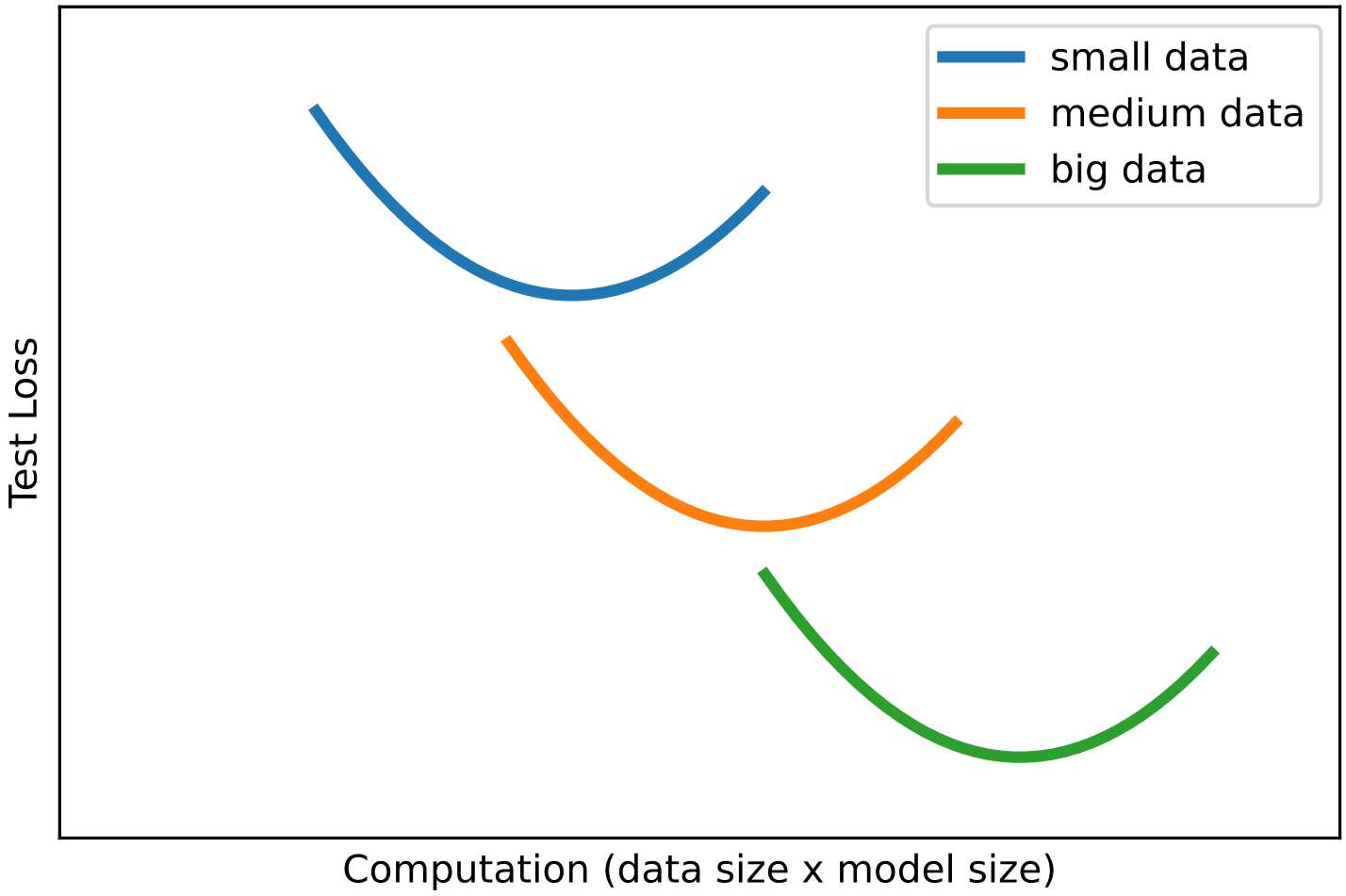
⋮

local token statistics
(1-2 tokens back)

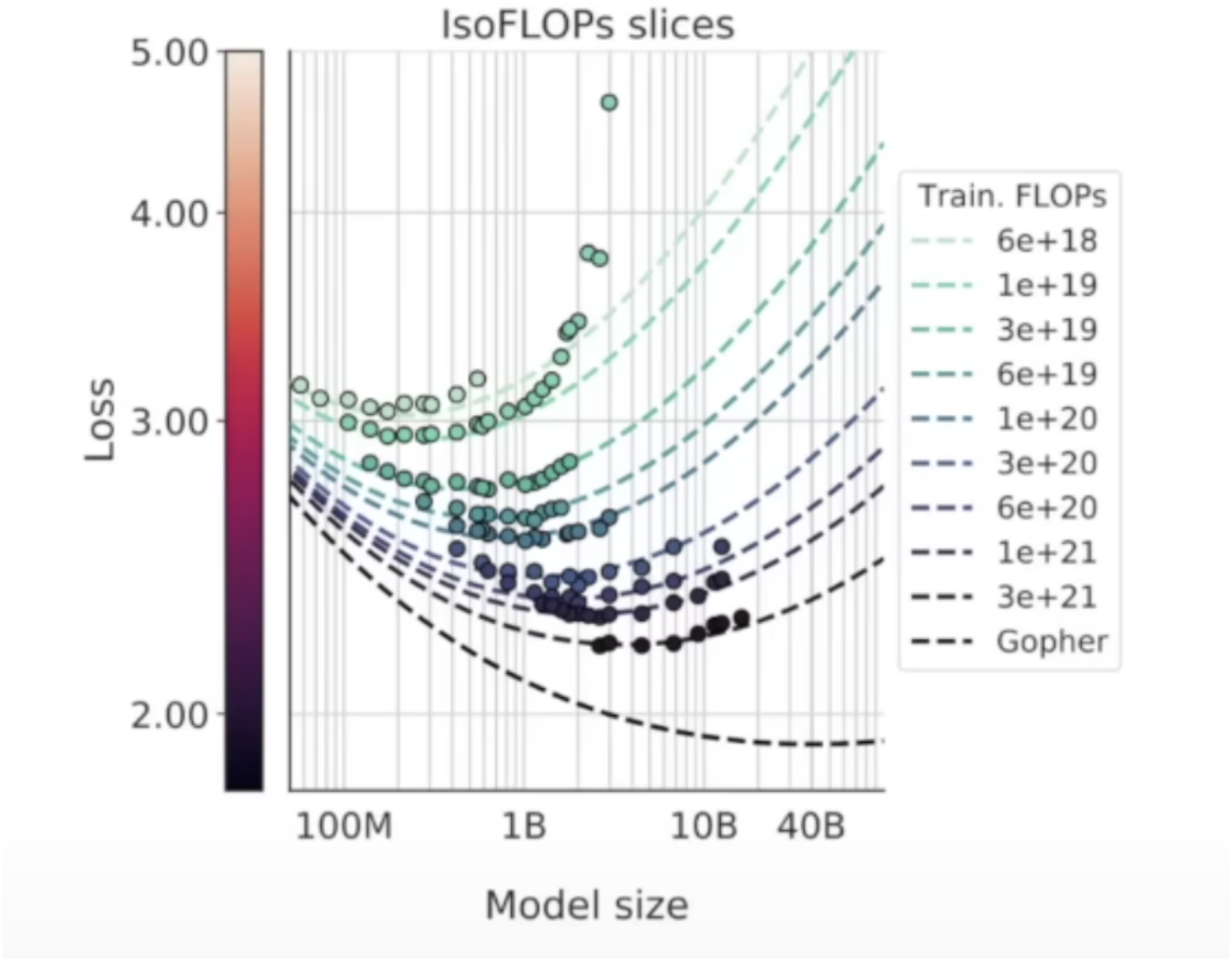
The Transformer architecture

How large are the models?

Recall: Scaling behavior of ML models



Recall: Scaling behavior of ML 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

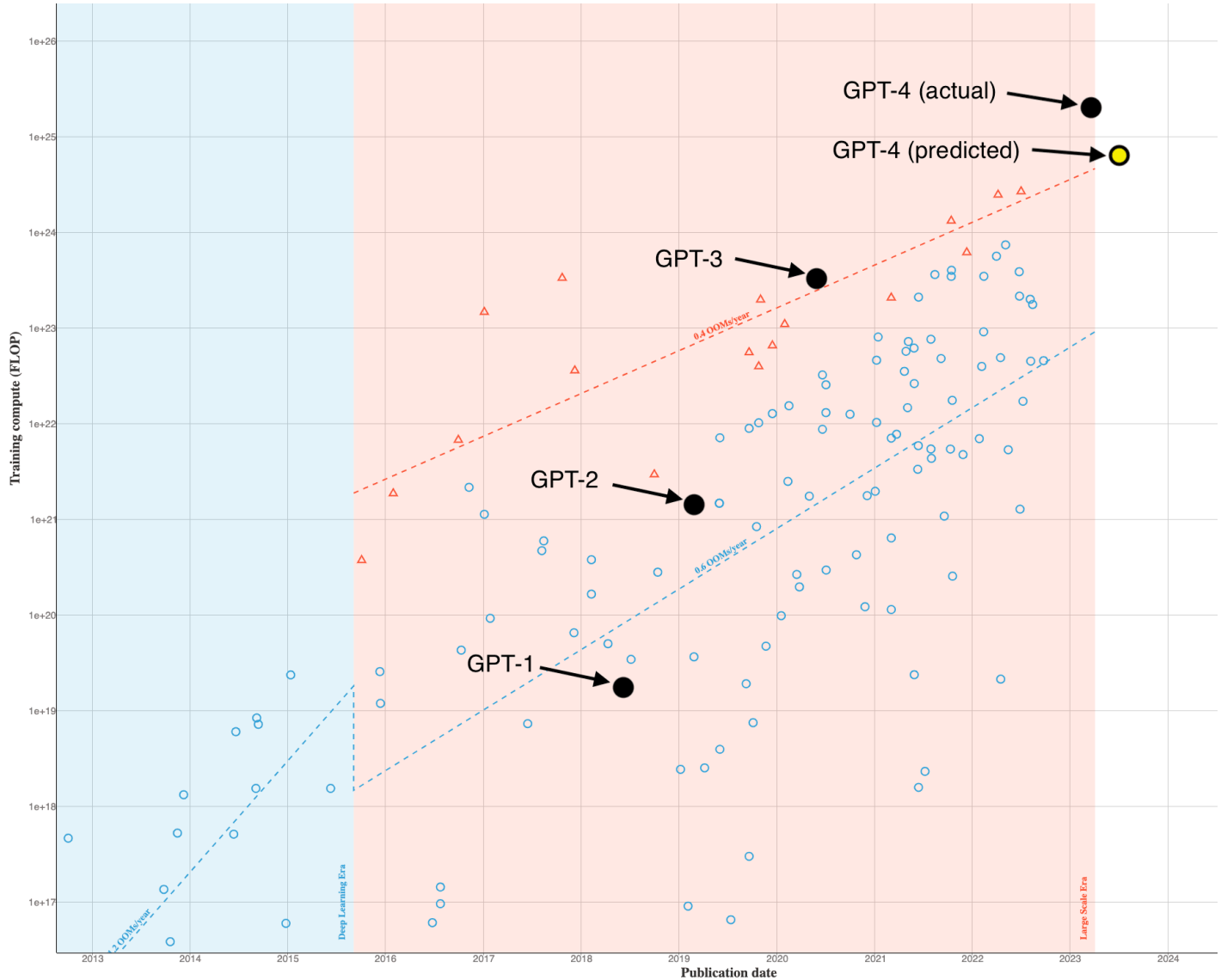
Size of models

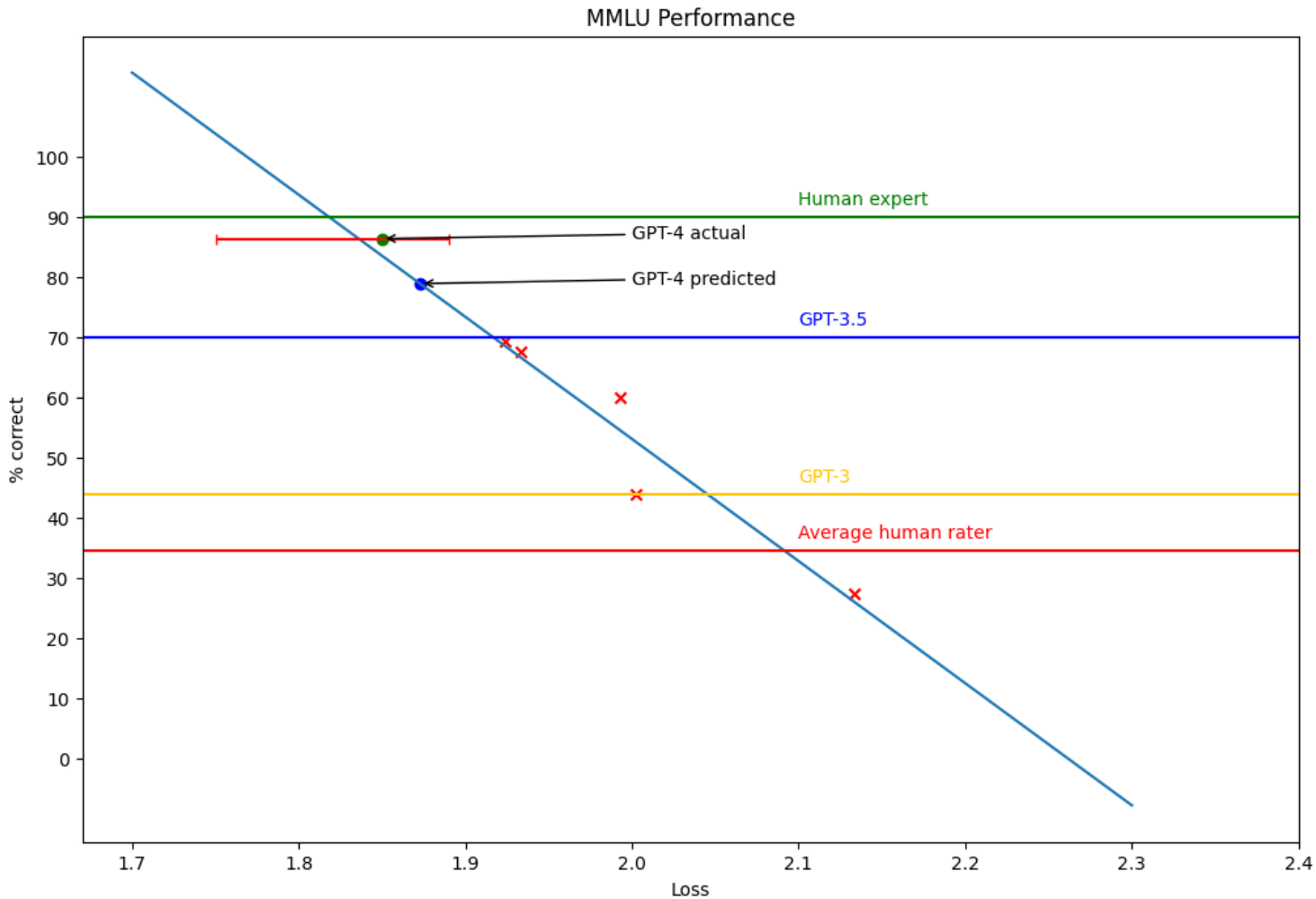
GPT-3 (2020)

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

- 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.


GPT-4 (actual)
GPT-4 (predicted)
GPT-3
GPT-2
GPT-1
0.4 OOMs/year
0.6 OOMs/year
2.0 OOMs/year
Deep Learning Era
Large Scale Era
Training compute (FLOP)
Publication date
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



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