Raw Data

Neural Language Models and Transformers

Cornell CS 5740: Natural Language Processing
Yoav Artzi, Spring 2023

Neural Language Models

- LMs so far: count-based estimates of probabilities
  - Counts are brittle and generalize poorly, so we added smoothing
- The quantity that we are focused on estimating (e.g., for tri-gram model):

$$p(\bar{x}) = \prod_{i=1}^{n} p(x_i \mid x_{i-1}, x_{i-2}), \text{ where } x_0, x_{-1} = *, x_i \in \mathcal{V} \cup \{\text{STOP}\}$$

- Can we use neural networks for this task? What would it give us? What are the costs?

Neural Language Models

A Very Simple Approach

- Instead of having count-based distributions, parameterize them

$$p(x_i \mid x_{i-1}, x_{i-2}; \theta)$$

- How would we model this with a neural network?
  - Hint: so far, only learned about MLPs

Neural Language Models

A Very Simple Approach

- A simple MLP-ish model

$$p(x_i = w \mid x_{i-1}, x_{i-2}; \theta) = \text{softmax}(\mathbf{y})_w$$
$$\mathbf{y} = \mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{U}\tanh(\mathbf{d} + \mathbf{H}\mathbf{x})$$
$$\mathbf{x} = [\phi(x_{i-1}); \phi(x_{i-2})]$$

where ϕ is an embedding function, and $\theta = (\mathbf{b}, \mathbf{d}, \mathbf{W}, \mathbf{U}, \mathbf{H}, \mathbf{C}, \phi)$

- The parameters θ are estimated by maximizing the log probability of the data

- During inference, you compute the neural network every time you need a value from the probability distribution

[Bengio et al. 2003]

Neural Language Models

A Very Simple Approach

- A simple MLP-ish model

$$
\begin{aligned}
p(x_i = w \mid x_{i-1}, x_{i-2}; \theta) = \text{softmax}(\mathbf{y})_w \\
\mathbf{y} = \mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{U}\tanh(\mathbf{d} + \mathbf{H}\mathbf{x}) \\
\mathbf{x} = [\phi(x_{i-1}); \phi(x_{i-2})]
\end{aligned}
$$

  where ϕ is an embedding function, and $\theta = (\mathbf{b}, \mathbf{d}, \mathbf{W}, \mathbf{U}, \mathbf{H}, \mathbf{C}, \phi)$

- What does it give us? Think smoothing …

Neural Language Models

A Very Simple Approach

- A simple MLP-ish model

$$
\begin{aligned}
p(x_i = w \mid x_{i-1}, x_{i-2}; \theta) &= \mathrm{softmax}(\mathbf{y})_w \\
\mathbf{y} &= \mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{U}\tanh(\mathbf{d} + \mathbf{H}\mathbf{x}) \\
\mathbf{x} &= [\phi(x_{i-1}); \phi(x_{i-2})]
\end{aligned}
$$

  where ϕ is an embedding function, and $\theta = (\mathbf{b}, \mathbf{d}, \mathbf{W}, \mathbf{U}, \mathbf{H}, \mathbf{C}, \phi)$

- What does it give us? Think smoothing …

$$
\mathrm{softmax}(\mathbf{y})_w = \frac{\exp(y_w)}{\sum_{y \in \mathbf{y}} \exp(y)}
$$

  - What does the $\mathrm{softmax}$ do the smoothing problem?

- What are the costs?

Neural Language Models

- The MLP approach can help with smoothing at some costs
- But essentially makes the same modeling choices
    - Assuming a finite horizon — the Markov assumption
    - We adopted this assumption because of sparsity (i.e., smoothing) challenges
- Can neural networks allow us to revisit these assumptions?

Neural Language Models

Revisiting the Markov Assumption

- The Markov assumption was critical for generalization
- But: it’s terrible for natural language!
    - “I ate a strawberry with some cream”
    - “I ate a strawberry that was picked in the field by the best farmer in the world with some cream”
- Dependencies can bridge arbitrarily long linear distances
    - We saw that already with word2vec
- It get even worse beyond the single sentence

Neural Language Models

An MLP with No Markov Assumption

- Without the Markov assumption, the model is

$$p(\bar{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

- We need to model the parameterized distribution

$$p(x_{i+1} \mid x_1, \ldots, x_i; \theta)$$

  - Note: shifted the index here, because it will make things nicer later on — just a notation change

- How can we do this with the tools we already know?

Neural Language Models

An MLP with No Markov Assumption

- We need to model the parameterized distribution

$$p(x_{i+1} \mid x_1, \ldots, x_i; \theta)$$

- We can just treat the context as a bag of words
  - Then it doesn't matter how long it is
  - A very simple example (two layer MLP)

$$\mathbf{h} = \tanh(\mathbf{W}' \tfrac{1}{i} \textstyle\sum_{j=1}^{i} \phi(x_j) + \mathbf{b}')$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(\mathbf{W}''\mathbf{h} + \mathbf{b}'')$$

Neural Language Models

An MLP with No Markov Assumption

- We can just treat the context as a bag-of-words, for example:

$$\mathbf{h} = \tanh(\mathbf{W}' \tfrac{1}{i} \textstyle\sum_{j=1}^{i} \phi(x_j) + \mathbf{b}')$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(\mathbf{W}''\mathbf{h} + \mathbf{b}'')$$

- Why is this a terrible idea?

Neural Language Models

An MLP with No Markov Assumption

- We can just treat the context as a bag-of-words, for example:

$$\mathbf{h} = \tanh(\mathbf{W}'\frac{1}{i}\sum_{j=1}^{i}\phi(x_j) + \mathbf{b}')$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(\mathbf{W}''\mathbf{h} + \mathbf{b}'')$$

- Why is this a terrible idea?
  - Order matters a lot in language 🤦‍♂️
  - But it worked so well for text categorization … 😮‍💨
  - What may work for tasks that just require focusing on salient words (e.g., topic categorization), is not sufficient for language models (i.e., **next**-word prediction)

Neural Language Models

Bag of Words

- BOW can handle arbitrary length 😁
- But losses any notion of order 😩
- Furthermore, dependencies are complex 😵‍💫
  - Not following linear order
  - Importance follow complex patterns
    - “I ate a strawberry that was picked in the field by the best farmer in the world with some cream”
    - “I ate a strawberry that was picked in the field by the best farmer in the world with clippers”
  - The model needs to focus on different parts in the context to predict different words

Bag of Words

A Uniform Distribution Over Past Words

- We can view BOW as a **attending** to all previous tokens equally
- So can rewrite our simple example MLP using a uniform distribution

$$p(j) = \frac{1}{i} \quad , \quad j = 1, \ldots, i$$

$$\mathbf{h} = \tanh(\mathbf{W}' \textstyle\sum_{j=1}^{i} p(j)\phi(x_j) + \mathbf{b}')$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \mathrm{softmax}(\mathbf{W}''\mathbf{h} + \mathbf{b}'')$$

- What if we want to attend to past tokens in an adaptive way?

Bag of Words

A Uniform Distribution Over Past Words

- We can view BOW as a **attending** to all previous tokens equally
- So can rewrite our simple example MLP using a uniform distribution

$$p(j) = \frac{1}{i} \quad , \quad j = 1, \ldots, i$$

$$\mathbf{h} = \tanh(\mathbf{W}' \textstyle\sum_{j=1}^{i} p(j)\phi(x_j) + \mathbf{b}')$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(\mathbf{W}''\mathbf{h} + \mathbf{b}'')$$

- What if we want to attend to past tokens in an adaptive way?
  - We need a way to do weighted processing of context to represent that different words depend on context differently
  - This weighted processing must reflect ordering

Attention

- An architecture that functions similar to a soft query-key-value dictionary lookup

- Given a query $\mathbf{q} \in \mathbb{R}^{d_k}$ and a key-value dictionary $\{(\mathbf{k}^{(i)}, \mathbf{v}^{(i)})\}_{i=1}^{N}$ where $\mathbf{k}^{(i)} \in \mathbb{R}^{d_k}$, $\mathbf{v}^{(i)} \in \mathbb{R}^{d_v}$

1. Compute a probability distribution over dictionary entries

$$a_i = \mathbf{q} \cdot \mathbf{k}^{(i)} \quad , \quad p(i) = \mathrm{softmax}(\mathbf{a})$$

2. Output $\mathbf{z} \in \mathbb{R}^{d_v}$ is weighted average of values: $\mathbf{z} = \sum_{i=1}^{N} p(i)\mathbf{v}^{(i)}$

Self-attention

- Attention where the query, keys, and values come from the same input
- Given a set of vectors $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ and a query position $j \in 1, \dots, N$ we want to create a weighted sum of all vectors

1. Compute query, keys, and values vectors via linear transformation

$$\mathbf{q} = \mathbf{W}_q \mathbf{x}^{(j)} \quad \mathbf{k}^{(i)} = \mathbf{W}_k \mathbf{x}^{(i)} \quad \mathbf{v}^{(i)} = \mathbf{W}_v \mathbf{x}^{(i)}$$

2. Compute a probability distribution over dictionary entries

$$a_i = \mathbf{q} \cdot \mathbf{k}^{(i)} \quad , \quad p(i) = \mathrm{softmax}(\mathbf{a})$$

3. Output $\mathbf{z} \in \mathbb{R}^{d_v}$ is weighted average of values: $\mathbf{z} = \sum_{i=1}^{N} p(i)\mathbf{v}^{(i)}$

Self-attention

More Important Details

- Computing attention using loops is crazy slow → it is critical to do everything with a few matrix multiplications by packing all keys and values in matrices $\mathbf{K}$ and $\mathbf{V}$

- We usually compute for multiple queries $\mathbf{Q}$, resulting in multiple outputs $\mathbf{Z}$

- Finally, it is common to divide by $\sqrt{d_k}$ because the dot-product is likely to get large in relation the key dimensionality

$$\mathrm{SelfAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{Z} = \mathrm{softmax}(\mathbf{QK}/\sqrt{d_k})\mathbf{V}$$

LM with Self-attention

From BOW to Self-attention

- Reminder, this is the simple BOW LM we showed earlier

$$p(j) = \frac{1}{i} \quad , \quad j = 1, \ldots, i$$

$$\mathbf{h} = \tanh(\mathbf{W}' \textstyle\sum_{j=1}^{i} p(j)\phi(x_j) + \mathbf{b}')$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \mathrm{softmax}(\mathbf{W}''\mathbf{h} + \mathbf{b}'')$$

- We can easily plug in self-attention to create a weighted processing of the context
- The query is computed from the most recent token
- Keys and values are computed from entire context (i.e., all previous tokens)
- Did we solve the issues with BOW?
    - ✅ Words can't depend on context **differently**
    - ❌ Attention is **order** invariant

$$\mathbf{q} = \mathbf{W}_q \phi(x_i)$$

$$\mathbf{K} = \mathbf{W}_k[\phi(x_1) \cdots \phi(x_i)]$$

$$\mathbf{V} = \mathbf{W}_v[\phi(x_1) \cdots \phi(x_i)]$$

$$\mathbf{z} = \mathrm{SelfAttn}(\mathbf{q}, \mathbf{K}, \mathbf{V})$$

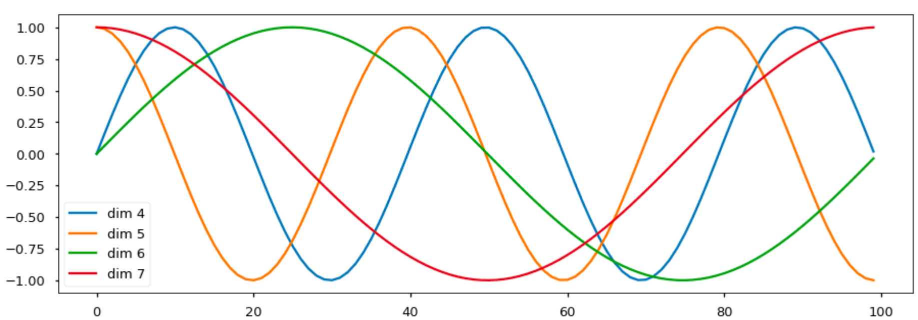
$$\mathbf{h} = \mathbf{W}''\tanh(\mathbf{W}'\mathbf{z} + \mathbf{b}') + \mathbf{b}''$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \mathrm{softmax}(\mathbf{h})$$

Marking Positions

Self-attention with Positional Embeddings

- Idea: let's mark positions
- Learning will figure out what how to use them
- Simple version: **learnable** embeddings $\phi_p(i)$
- More advanced: **fixed** embeddings, where values determined by sine waves, with different frequency and offset of each dimensions



- Either way, add them to token embeddings

$$\mathbf{x}_j = \phi(x_j) + \phi_p(j), j = 1, \ldots, i$$
$$\mathbf{q} = \mathbf{W}_q \mathbf{x}_i$$
$$\mathbf{K} = \mathbf{W}_k [\mathbf{x}_1 \cdots \mathbf{x}_i]$$
$$\mathbf{V} = \mathbf{W}_v [\mathbf{x}_1 \cdots \mathbf{x}_i]$$
$$\mathbf{z} = \mathrm{SelfAttn}(\mathbf{q}, \mathbf{K}, \mathbf{V})$$
$$\mathbf{h} = \mathbf{W}'' \tanh(\mathbf{W}'\mathbf{z} + \mathbf{b}') + \mathbf{b}''$$
$$p(x_{i+1} \mid x_1, \ldots, x_i) = \mathrm{softmax}(\mathbf{h})$$

Self-attention LM

- Did we solve the issues with BOW?
    - ✅ Words can't depend on context **differently**
    - ✅ Attention is **order** invariant
- Let's make it more expressive!

🚀

$$\mathbf{x}_j = \phi(x_j) + \phi_p(j), j = 1, \ldots, i$$
$$\mathbf{q} = \mathbf{W}_q \mathbf{x}_i$$
$$\mathbf{K} = \mathbf{W}_k[\mathbf{x}_1 \cdots \mathbf{x}_i]$$
$$\mathbf{V} = \mathbf{W}_v[\mathbf{x}_1 \cdots \mathbf{x}_i]$$
$$\mathbf{z} = \text{SelfAttn}(\mathbf{q}, \mathbf{K}, \mathbf{V})$$
$$\mathbf{h} = \mathbf{W}'' \tanh(\mathbf{W}'\mathbf{z} + \mathbf{b}') + \mathbf{b}''$$
$$p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(\mathbf{h})$$

Self-attention LM

Multiple Attention Heads

- Words need to attend to different elements in context
- But attention just does weighted average
- So: add more attention heads
- Let L be the number of attention heads

$$\mathbf{x}_j = \phi(x_j) + \phi_p(j), j = 1, \ldots, i$$
$$\mathbf{q}^{(l)} = \mathbf{W}_q^{(l)} \mathbf{x}_i$$
$$\mathbf{K}^{(l)} = \mathbf{W}_k^{(l)} [\mathbf{x}_1 \cdots \mathbf{x}_i]$$
$$\mathbf{V}^{(l)} = \mathbf{W}_v^{(l)} [\mathbf{x}_1 \cdots \mathbf{x}_i]$$
$$\mathbf{z} = [\mathrm{SelfAttn}(\mathbf{q}^{(1)}, \mathbf{K}^{(1)}, \mathbf{V}^{(1)}); \cdots; \mathrm{SelfAttn}(\mathbf{q}^{(L)}, \mathbf{K}^{(L)}, \mathbf{V}^{(L)})]$$
$$\mathbf{h} = \mathbf{W}'' \tanh(\mathbf{W}' \mathbf{z} + \mathbf{b}') + \mathbf{b}''$$
$$p(x_{i+1} \mid x_1, \ldots, x_i) = \mathrm{softmax}(\mathbf{h})$$

Self-attention LM

Add Neural Network Tricks

- Switch activation to GELU (Gaussian Error Linear Unit)

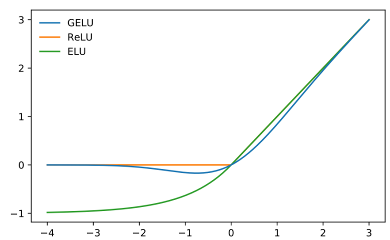


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

- Residual connection: shown to help with training very deep networks

- LayerNorm (LN): shown to improve performance

  - Post-norm (original and here)

$$\mathbf{b} = \mathrm{Module}(\mathrm{LN}(\mathbf{a})) + \mathbf{a}$$

  - Pre-norm (modern)

$$\mathbf{b} = \mathrm{LN}(\mathrm{Module}(\mathbf{a}) + \mathbf{a})$$

$$\mathbf{x}_j = \phi(x_j) + \phi_p(j), j = 1, \ldots, i$$

$$\mathbf{q}^{(l)} = \mathbf{W}_q^{(l)} \mathbf{x}_i$$

$$\mathbf{K}^{(l)} = \mathbf{W}_k^{(l)} [\mathbf{x}_1 \cdots \mathbf{x}_i]$$

$$\mathbf{V}^{(l)} = \mathbf{W}_v^{(l)} [\mathbf{x}_1 \cdots \mathbf{x}_i]$$

$$\mathbf{z} = \mathrm{LN}([\mathrm{SelfAttn}(\mathbf{q}^{(1)}, \mathbf{K}^{(1)}, \mathbf{V}^{(1)}); \cdots; \mathrm{SelfAttn}(\mathbf{q}^{(L)}, \mathbf{K}^{(L)}, \mathbf{V}^{(L)})] + \mathbf{x}_i)$$

$$\mathbf{h} = \mathrm{LN}(\mathbf{W}'' \mathrm{GELU}(\mathbf{W}' \mathbf{z} + \mathbf{b}') + \mathbf{b}'' + \mathbf{z})$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \mathrm{softmax}(\mathbf{h})$$

Self-attention LM

Abstract and Stack It

- Abstract the whole computation as a **Transformer** block
- And stack it

$\text{TransformerBlock}^{\text{k}}(\mathbf{u}_1, \ldots, \mathbf{u}_{\text{i}})$

$$\mathbf{q}^{(l)} = \mathbf{W}_q^{(l)}\mathbf{u}_i$$

$$\mathbf{K}^{(l)} = \mathbf{W}_k^{(l)}[\mathbf{u}_1 \cdots \mathbf{u}_i]$$

$$\mathbf{V}^{(l)} = \mathbf{W}_v^{(l)}[\mathbf{u}_1 \cdots \mathbf{u}_i]$$

$$\mathbf{z} = \text{LN}([\text{SelfAttn}(\mathbf{q}^{(1)}, \mathbf{K}^{(1)}, \mathbf{V}^{(1)}); \cdots; \text{SelfAttn}(\mathbf{q}^{(L)}, \mathbf{K}^{(L)}, \mathbf{V}^{(L)})] + \mathbf{u}_i)$$

$$\mathbf{h}_i^k = \text{LN}(\mathbf{W}''\text{GELU}(\mathbf{W}'\mathbf{z} + \mathbf{b}') + \mathbf{b}'' + \mathbf{z})$$

$$\mathbf{x}_i = \phi(x_i) + \phi_p(i)$$

$$\mathbf{h}_i^1 = \text{TransformerBlock}^1(\mathbf{x}_1, \ldots, \mathbf{x}_{\text{i}})$$

$$\mathbf{h}_i^2 = \text{TransformerBlock}^2(\mathbf{h}_1^1, \ldots, \mathbf{h}_{\text{i}}^1)$$

$$\ldots$$

$$\mathbf{h}_i^k = \text{TransformerBlock}^{\text{k}}(\mathbf{h}_1^{\text{k}-1}, \ldots, \mathbf{h}_{\text{i}}^{\text{k}-1})$$

$$\ldots$$

$$\mathbf{h}_i^K = \text{TransformerBlock}^{\text{K}}(\mathbf{h}_1^{\text{K}-1}, \ldots, \mathbf{h}_{\text{i}}^{\text{K}-1})$$

$$p(x_{i+1} \,|\, x_1, \ldots, x_i) = \text{softmax}(\mathbf{W}^{\mathcal{V}}\mathbf{h}_i^K)$$

Transformers

- A variable length architecture
  - Was not the first architecture to do that
  - But we are not following the chronological order of events
- Key concept: **self-attention**
- Quickly became maybe the most dominant architecture
  - Try to think why

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

[Vaswani et al. 2017]

The Transformer

Decoder-only Variant

$$\text{TransformerBlock}^{\mathrm{k}}(\mathbf{u}_1, \ldots, \mathbf{u}_{\mathrm{i}})$$

$$\mathbf{q}^{(l)} = \mathbf{W}_q^{(l)} \mathbf{u}_i$$

$$\mathbf{K}^{(l)} = \mathbf{W}_k^{(l)} [\mathbf{u}_1 \cdots \mathbf{u}_i]$$

$$\mathbf{V}^{(l)} = \mathbf{W}_v^{(l)} [\mathbf{u}_1 \cdots \mathbf{u}_i]$$

$$\mathbf{z} = \text{LN}([\text{SelfAttn}(\mathbf{q}^{(1)}, \mathbf{K}^{(1)}, \mathbf{V}^{(1)}); \cdots; \text{SelfAttn}(\mathbf{q}^{(L)}, \mathbf{K}^{(L)}, \mathbf{V}^{(L)})] + \mathbf{u}_i)$$

$$\mathbf{h}_i^k = \text{LN}(\mathbf{W}'' \text{GELU}(\mathbf{W}' \mathbf{z} + \mathbf{b}') + \mathbf{b}'' + \mathbf{z})$$

$$\mathbf{x}_i = \phi(x_i) + \phi_p(i)$$

$$\mathbf{h}_i^1 = \text{TransformerBlock}^1(\mathbf{x}_1, \ldots, \mathbf{x}_{\mathrm{i}})$$

$$\mathbf{h}_i^2 = \text{TransformerBlock}^2(\mathbf{h}_1^1, \ldots, \mathbf{h}_{\mathrm{i}}^1)$$

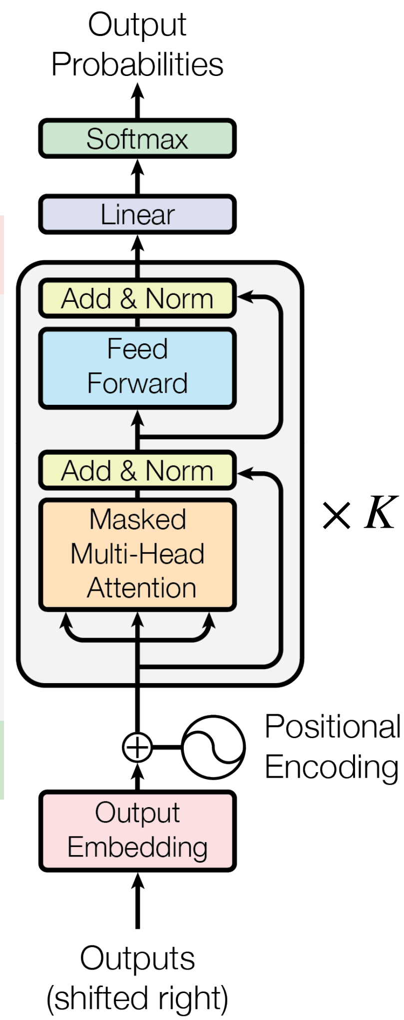
$$\ldots$$

$$\mathbf{h}_i^k = \text{TransformerBlock}^{\mathrm{k}}(\mathbf{h}_1^{\mathrm{k}-1}, \ldots, \mathbf{h}_{\mathrm{i}}^{\mathrm{k}-1})$$

$$\ldots$$

$$\mathbf{h}_i^K = \text{TransformerBlock}^{\mathrm{K}}(\mathbf{h}_1^{\mathrm{K}-1}, \ldots, \mathbf{h}_{\mathrm{i}}^{\mathrm{K}-1})$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(\mathbf{W}^{\mathcal{V}} \mathbf{h}_i^K)$$



Self-attention reminder

$$\text{SelfAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{QK}/\sqrt{d_k})\mathbf{V}$$

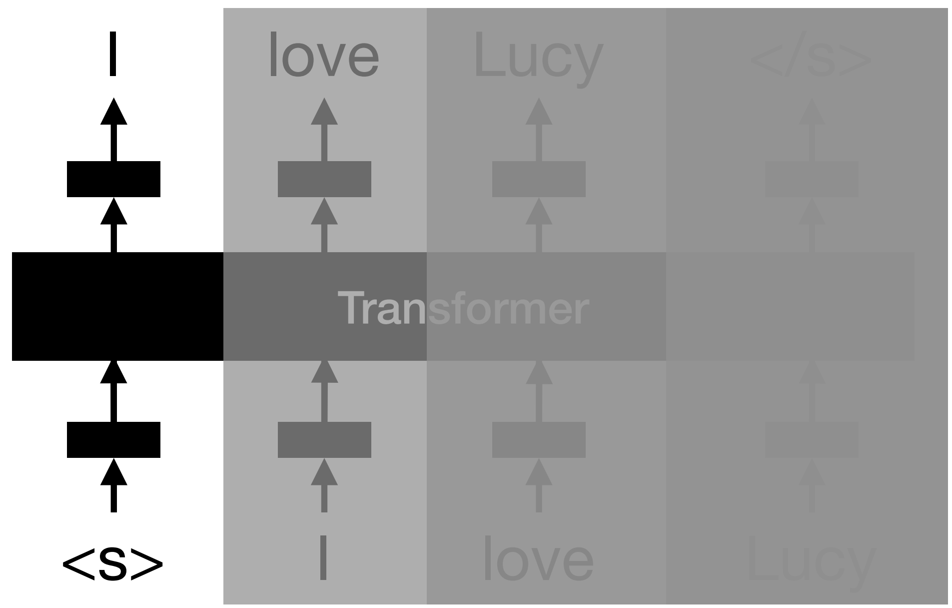
During learning, compute the whole sequence at ones by **masking** items you shouldn't attend to in softmax — easy by setting softmax to $-\infty$

[Vaswani et al. 2017]

Transformer

Shifted Outputs as Inputs

- For each time step:
  - Input: previous word (and everything computed before)
  - Output: probability distribution over the vocabulary



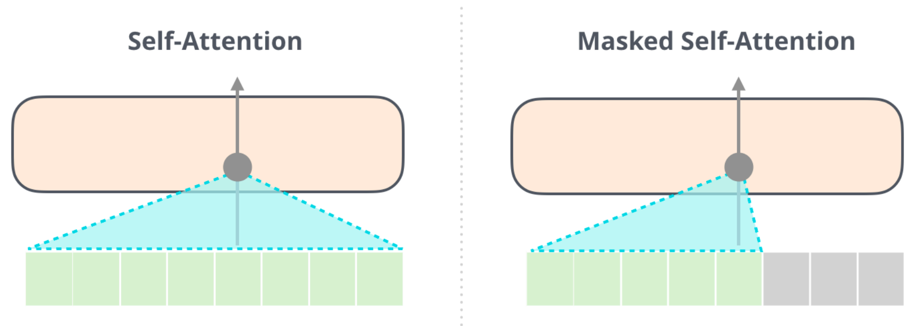
Transformer

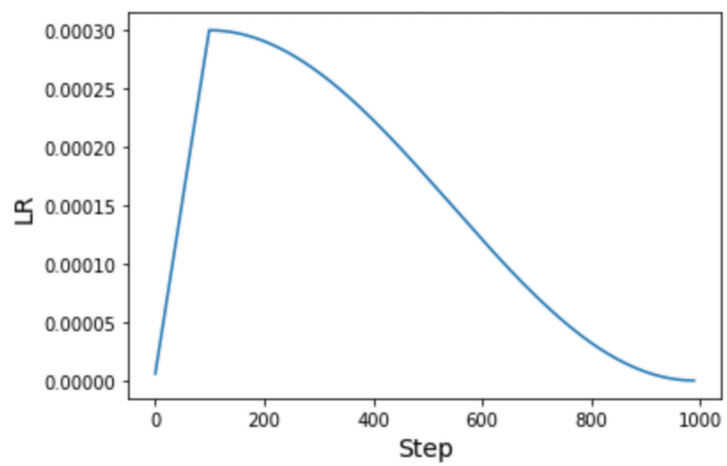
Language Model Training

- Training loss is the per-token negative log likelihood:

$$\mathscr{L} = -\log p(x_i \mid x_1, \ldots, x_{i-1})$$

- During training: we know all tokens
  - So **masked** self-attention
  - To account for ordering
- Transformers are very sensitive to learning rate schedule → linear warm up + cosine decay





Transformer
Issues

- Time and memory complexity
  - Time: attention is quadratic $O(n^2)$ in sequence length n
  - Memory: Need to keep almost all past activation for self-attention
- Positional embeddings mean you can only handle positions up to the length you observed in training
- A lot of existing and ongoing work on both issues

Transformer

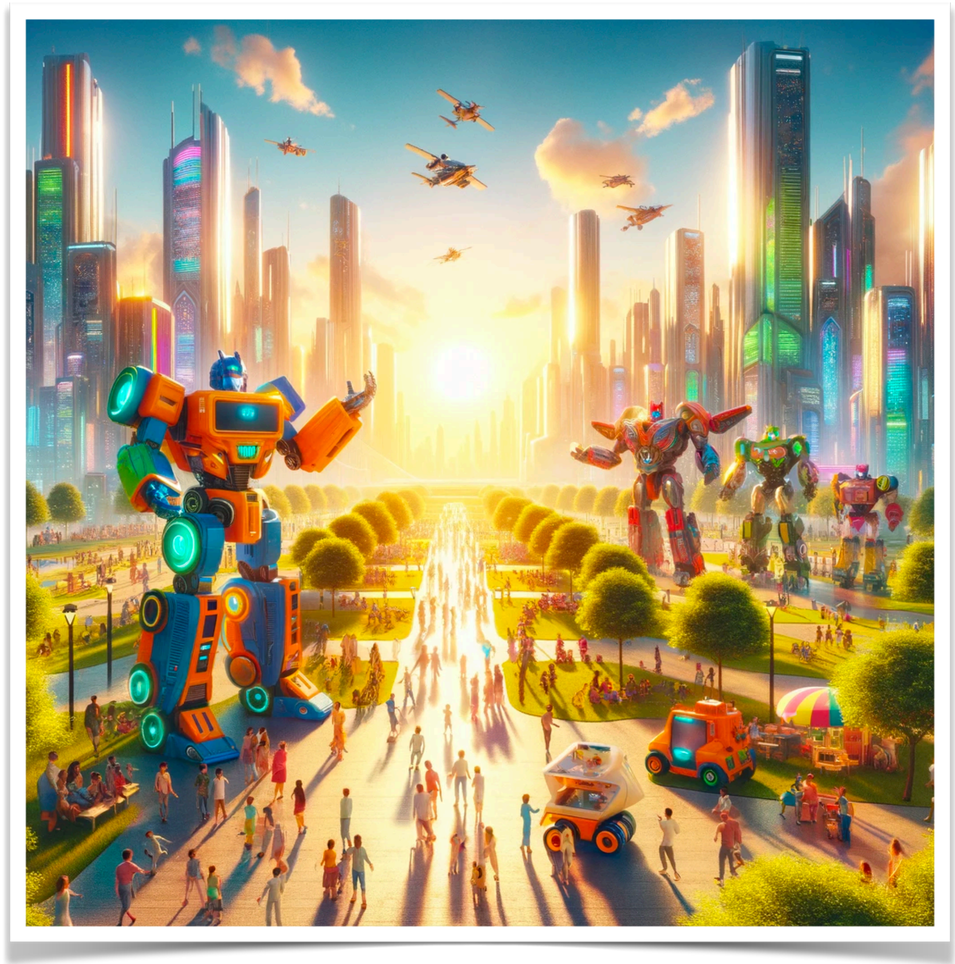
Technical Complexities

- Some complexities you will encounter:
    - Masking self-attention
    - Batching
    - Learning rate sensitivity

Transformers

A Success Story

- Transformers were designed with hardware in mind
  - Especially TPUs, but also GPUs
- Exceptionally designed for scale as far as hardware
- Turns out, also scale well for learning
- Unparalleled success in NLP, vision, speech, RL, science, and other areas



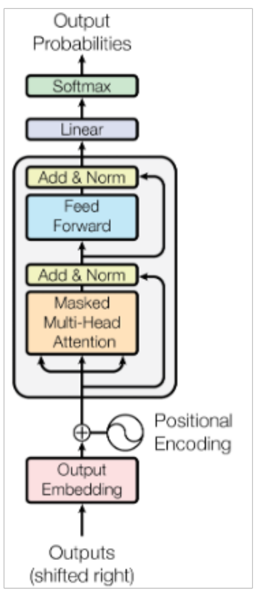
Transformers

Natural Language

Decoder-only

GPT

[sat_]

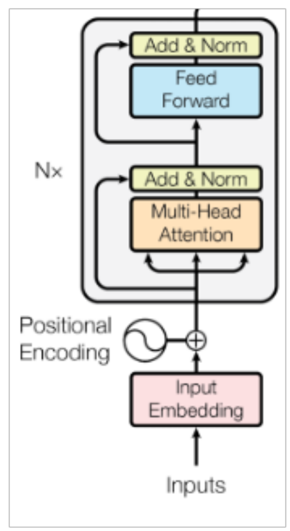


[START] [The_] [cat_]

Encoder-only

BERT

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



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

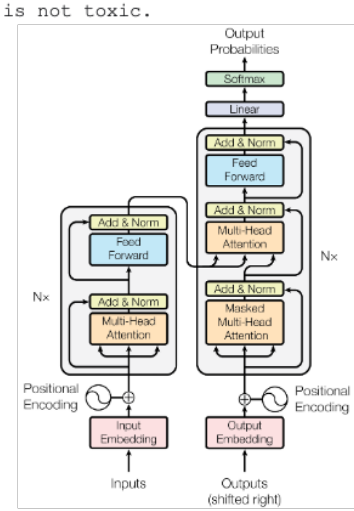
Encoder-decoder

T5

Das ist gut.

A storm in Attala caused 6 victims.

This is not toxic.



Translate EN-DE: This is good.

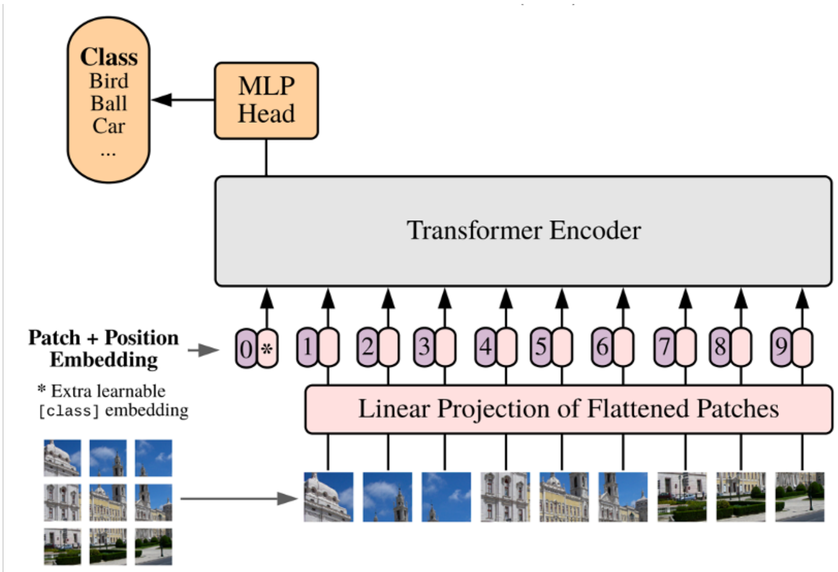
Summarize: state authorities dispatched…

Is this toxic: You look beautiful today!

Transformers

Computer Vision

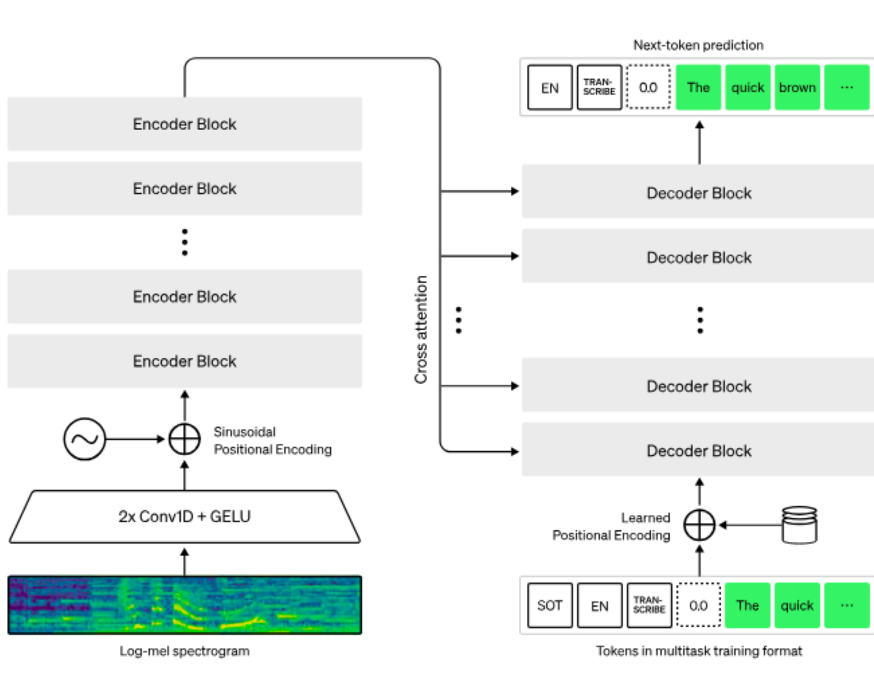
- ViT: cut image to patches
- Project each patch to a vector
- Treat them as token embeddings



[Dosovitskiy et al. 2020]

Transformers
Speech

- Same as computer vision
- But: spectrograms instead of images
- The Whisper model

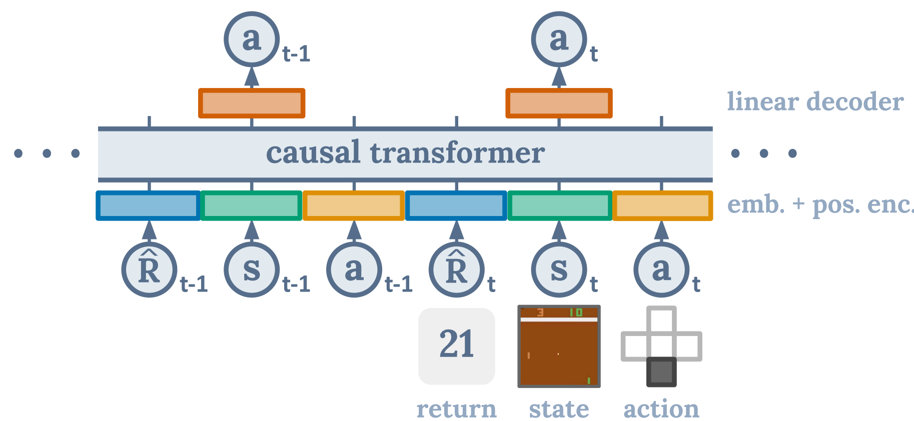


[Radford et al. 2022]

Transformers

Reinforcement Learning (RL)

- **Decision Transformers**
- Inputs are action states and target values
- Value is (in a nutshell) how much reward you want to get
- Outputs are actions

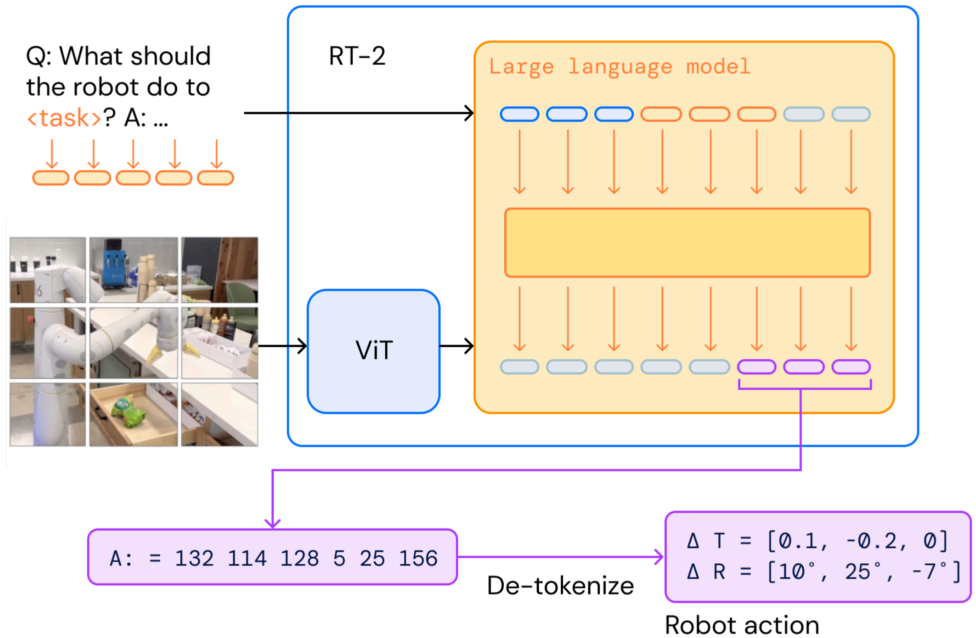


[Chen et al. 2021]

Transformers

Robotics

- Take observations and commands, all tokenized
- Output continuous joint control actions



[Brohan et al. 2023]

Transformers

Everything Everywhere All at Once

- Whatever you can tokenize, the Transformer will take
- What more: you can feed them all to the same model

[image from: https://deepmind.google/discover/blog/rt-2-new-model-translates-vision-and-language-into-action/]

Acknowledgements

- Some content was adapted from slides by Lucas Beyer
- We thank Greg Durrett, Ana Marasović, and Christian von der Weth for very helpful discussions.