8/28/2024: Transformer LMs

1. Autoregressive LM
2. Using Transformers as LMs

LMs

- ☆ Autoregressive (GPT family, Llama family, ...)
- Masked LMs (BERT)

<u>Autoregressive LM</u>: Takes input = $w_1, \ldots, w_T$ (T tokens)

represent first T tokens of a document

Output: Distribution over the next token w_{T+1}

$$= P_{LM}(w_{T+1} \mid \underbrace{w_1, \ldots, w_T}_{\text{prefix}} ; \theta)$$

↑ model parameters

Big Assumption: All tokens come from a fixed set V ("vocabulary")

↳ Under this assumption, this distribution is over $|V|$ possibilities

⇒ represent it as vector of dimension $|V|$

Note: Implicitly, this defines probability distribution over longer sequences.

For any string $S = w_1, \ldots, w_T$:

$$P_{LM}(S ; \theta) = \prod_{t=1}^{T} p(w_t \mid w_1, \ldots, w_{t-1} ; \theta)$$

$\boxed{\text{How to train?}}$ To pretrain an autoregressive LM:

Need dataset D of many strings $s = s_1, \ldots, s_T$

<u>Choose θ to minimize</u>

$$L(\theta) = \underbrace{\frac{1}{|D|} \sum_{s \in D}}_{\text{every string}} \underbrace{\frac{1}{T} \sum_{t=1}^{T}}_{\text{every token in string}} - \underbrace{\log P_{LM}(s_t \mid s_1, \ldots, s_{t-1})}_{\text{log prob of the real next token}}$$

By stochastic gradient descent

Transformers

$$\begin{array}{lcccc}
 & & & & \text{predicted } p(w_5 \mid w_1, \dots, w_4) \\
 & & & & \uparrow \\
\text{Layer 2} & h_1^{(2)} & h_2^{(2)} & h_3^{(2)} & \boxed{h_4^{(2)}} \\
 & \uparrow & \uparrow & \uparrow & \uparrow \\
\text{Layer 1} & h_1^{(1)} & h_2^{(1)} & h_3^{(1)} & h_4^{(1)} \\
 & \uparrow & \uparrow & \uparrow & \uparrow \\
 & x_1 & x_2 & x_3 & x_4 \\
 & \uparrow & \uparrow & \uparrow & \uparrow \\
 & w_1 & w_2 & w_3 & w_4
\end{array}$$

✓ ③ Generate final prediction = vector in $\mathbb{R}^{|V|}$

✓ ② Generate hidden states $\in \mathbb{R}^d$ for layers $1, 2, \dots, L$

✓ ① Generate token embedding $\in \mathbb{R}^d$

Input tokens $\in V$

Step 1: Token Embeddings

What is a token?

Problem with word-level: A LOT of possible words → Massive Vocabulary
- New names → "unknown" word problem
- Typos

Either a word or part of a word!
Uncommon words will be split into multiple tokens! } subword tokenization

Example: "$\underbrace{\text{Aragorn}}_{3 \text{ tokens}}$ $\underbrace{\text{instructed}}_{1}$ $\underbrace{\text{Frodo}}_{2 \text{ tokens}}$ $\underbrace{\text{to}}_{1}$ $\underbrace{\text{mind}}_{1}$ ... "

Common Strategy for subword tokenization → Byte Pair Encoding (BPE)

Sentence Piece tokenization

How to map tokens $w_1, \dots, w_T$ to vectors $x_1, \dots, x_T$

One common strategy: Absolute positional embeddings

Goal: each x_t must encode 2 things:

- Identity of t-th token w_t
- Order of all the tokens

Solution: add together 2 vectors, 1 for each piece of info

① Encode identity of the token:

Learn a parameter matrix $W^{embed} \in \mathbb{R}^{|V| \times d}$

corresponding vector for w_t is $W^{embed}[w_t]$

= "w_t"-th row of matrix

if we number all tokens from $1, 2, \ldots, |V|$

② Encode position of the token:

Learn a vector p_t for each $t = 1, \ldots, T$

Final embedding $x_t = W^{embed}[w_t] + p_t$

Note of caution: Some modern models don't use absolute position embeds. Instead have tricks to encode "relative position"

Step 3: Generating Predictions

Input: Final hidden state $h_T^{(L)} \in \mathbb{R}^d$

L: total # of layers

last timestep

Output: Probability distribution over V

$$\underbrace{\text{Softmax}\Big(\underbrace{W^{unembed}}_{\in \mathbb{R}^{|V| \times d}} \times \boxed{LN}\big(\underbrace{h_T^{(L)}}_{\mathbb{R}^d}\big)\Big)}_{\mathbb{R}^{|V|}}$$

Layer Normalization

$$\underset{\mathbb{R}^n}{\text{Softmax}(v)} = \left[\frac{e^{v_1}}{\sum_{i=1}^{n} e^{v_i}}, \ldots, \frac{e^{v_n}}{\sum_{i=1}^{n} e^{v_i}}\right]$$

Key properties: • Every entry is positive
• Sums to 1
$\Rightarrow$ Valid probability distribution!

What is Layer Norm?

Trick to ensure that vector is at a "reasonable" scale

Input: Vector $x \in \mathbb{R}^n$

$\boxed{\text{Step 1}}$: Normalize x's entries to have 0 mean & variance 1

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \quad , \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

$$\text{Normalized } x = \tilde{x} = \frac{x - \mu}{\sigma}$$

$\boxed{\text{Step 2}}$: Rescale & shift by learned parameters $a \in \mathbb{R}^n$, $b \in \mathbb{R}^n$

$$LN(x) = \underbrace{a \odot}_{\text{elementwise re-scaling}} \underset{\text{normalized } x}{\tilde{x}} + \underbrace{b}_{\text{elementwise shift}}$$

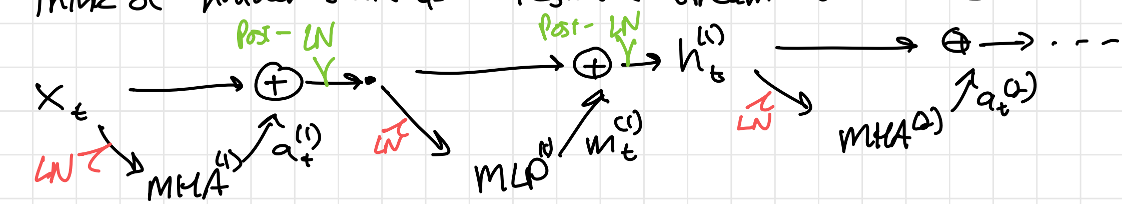
Why? Want some flexibility in scale of vector entries

$\boxed{\text{Step 2: Computing hidden states}}$

$$\underset{\text{token index}}{h_t^{(\ell)}} \leftarrow \text{Layer} \quad = h_t^{(\ell-1)} + \underbrace{a_t^{\ell}}_{\text{output of } \ell\text{-th multi-headed attention block (MHA)}} + \underbrace{m_t^{\ell}}_{\text{output of } \ell\text{-th feedforward network/MLP}}$$

Base case: $h_t^{(0)} = x_t$

Think of hidden states as "residual stream" of vectors



"stream": Constantly changing sequence of vectors,
Read & write from this stream

"residual": Adding result of block to previous state is called "residual connection"

Post-LN: old way of applying LN – after adding to stream

Pre-LN: new way – after read from stream

MHA: Retrieve relevant info from other tokens

MLP: Does some processing for current token

Multi-headed attention

First: what is single-headed attention?

- New hyperparam d_{attn} (size of vector that each head uses)
- Input: sequence of vectors $[u_1, \dots, u_T]$
- Output: vector $\in \mathbb{R}^{d_{attn}}$

How? 3 parameter matrices $W^Q, W^K, W^V \in \mathbb{R}^{d_{attn} \times d}$
query key value

① Compute query vector $q_T = W^Q \cdot u_T$ "what info are we looking for"

② Compute key vectors $k_t = W^K \cdot u_t$ for all $t = 1, \dots, T$
"what info is available at each token"

③ Compute value vectors $v_t = W^V \cdot u_t$ for all $t = 1, \dots, T$
"Actual info desired"

④ Try to match query with the keys:

Compute $s_t = \dfrac{q_T^{\,T} k_t}{\sqrt{d_{attn}}}$ for each $t = 1, \dots, T$

⑤ Convert to probabilities by Softmax:

$$[p_1, \dots, p_T] = \text{Softmax}([s_1, \dots, s_T])$$

⑥ output = $\sum_{t=1}^{T} p_t \cdot v_t$

From single head to multi-head:

MHA layer has n_{attn} attention heads
(from this, $d_{attn} = d / n_{attn}$)
- Each head has separate W^Q, W^K, W^V parameters
yields outputs $o_1, \dots, o_{n_{attn}}$

Final output of MHA is

$$MHA(u_1, \dots, u_T) = \underbrace{W^O}_{\text{parameter matrix of size } d \times d} \times \underbrace{\text{Concat}([o_1, \dots, o_{n_{attn}}])}_{\in \mathbb{R}^d}$$

In Transformer:

There's MHA^l at each layer $l = 1, \dots, L$

$$a_t^l = MHA^l(LN(h_1^{(l-1)}), \dots, LN(h_t^{(l-1)}))$$

Feedforward Network / Multi-Layer Perceptron
(FFN) (MLP)

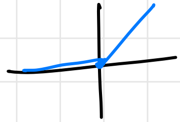
Most basic: 2-layer MLP:

Input vector $x \in \mathbb{R}^d$, output $y \in \mathbb{R}^d$

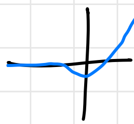
$$y = \underbrace{W^{(2)}}_{d \times d_{hidden}} \cdot \underbrace{f}_{\text{elementwise non-linearity}} (\underbrace{W^{(1)}}_{d_{hidden} \times d} x + \underbrace{b^{(1)}}_{d_{hidden}}) + \underbrace{b^{(2)}}_{d}$$

What is f?

- ReLU



- Swish



How used in TF?

$$m_t^{(\ell)} = \underset{\substack{\uparrow \\ \ell\text{-th layer's FFN}}}{FFN^{(\ell)}}\left(LN\left(h_t^{(\ell-1)} + a_t^{(\ell)}\right)\right)$$