ECE/CS 559 Lecture 25 11/26/2024

Last time: RNNs

① **Language modeling**

Language as a sequence of symbols (tokens) $x_1, x_2, \dots, x_t, \dots$

Objective: Maximize $P(x_1, x_2, \dots, x_t, \dots)$ of data

However, just like images, this is a high-dimensional space. Any specific sequence has exponentially small probability!

Instead: Write $P(x_1, x_2, \dots, x_t, \dots)$ as $P(x_1) \cdot P(x_2|x_1) \cdots P(x_t|x_{t-1}, \dots) \cdots$

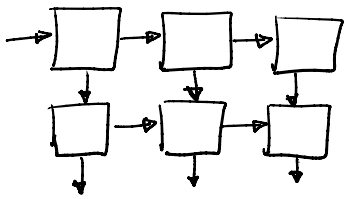
Maximize $\sum_t \log P(x_{t+1} | x_t, x_{t-1}, \dots)$

↑ this probability is not as small!

- RNNs can be used to represent these prediction probabilities. How? Often, we let y_t (the output) be logits. We go from logits to probabilities via a softmax:

$$P(x_{t+1} = i \mid x_t, x_{t-1}, \dots) = \frac{\exp(y_t[i])}{\sum_{i'} \exp(y_t[i'])}$$

- Issue with RNN: they're best with multiple layers



← This layer has to finish before we can do backpropagation on this layer

Not Parallelizable

② **(Self) Attention**

- Instead of recurrence, why not directly assess what past information is relevant? (like in databases)
  - Query: What the current symbol represents
  - Key: The relevance of other sequence symbols to a query.
  - Value: What each sequence query contributes

weights, R^k, embedding $\in \mathbb{R}^d$, attention weights

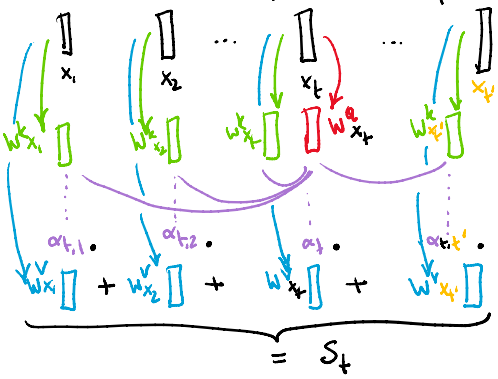
$$Q_{t,h} = W_h^Q x_t$$
$$K_{t',h} = W_h^K x_{t'}$$
$$V_{t',h} = W_h^V x_{t'}$$

$$\alpha_{t,t',h} = \text{softmax}\left(\frac{1}{\sqrt{k}} Q_{t,h}^T K_{t',h}\right)$$

scaled; inner product

(ignore all h at first)

- Then, to generate the new state variable, we combine values based on attention $S_t = W^O \underset{h}{\text{concat}} \sum_{t'} \alpha_{t,t',h} V_{t',h}$
- We could also have multiple heads, copies w/ different weights



(issue: T^2 complexity)

Notes:
- Each head is given by the triple (W^Q, W^K, W^V)
- The weights don't change by position.
- Position info can be placed inside x_t (positional embedding)
- Usually dimensions are maintained: #heads × d = k, W^O square.

③ **Transformers**

- Attention only tells us where the relevant information is
- We still need to act on it. We do it using:

(a) Add and normalize (residual connection)

$$S' = \text{LayerNorm}(S + X)$$

(b) 2-Layer Feedforward Neural Network

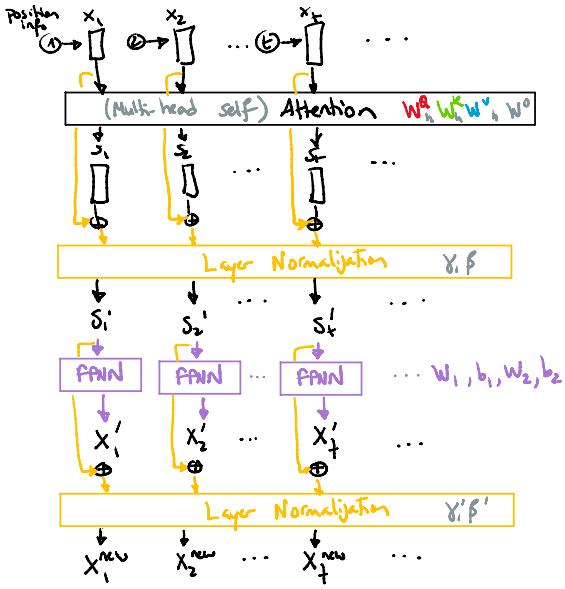
$$X'_t = W^2 \text{Relu}(W^1 S'_t + b^1) + b^2$$

(c) Add and normalize (again)

$$X^{new} = \text{LayerNorm}(S' + X')$$

$$\mu = \frac{1}{T}\sum_t S_t + X_t$$
$$\sigma^2 = \frac{1}{T}\sum_t (S_t + X_t - \mu)^2$$
$$S'_t = \gamma \frac{S_t + X_t - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

learnable; learnable



Parallelizable

per head per inner product

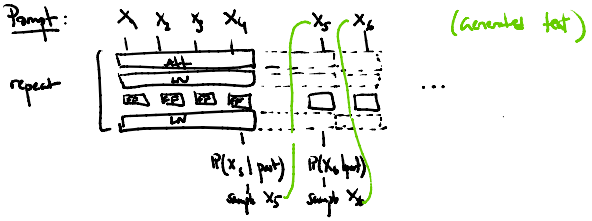
per feature

per position

(4) Using a transformer:

(a) Decoder-only architecture (ChatGPT)

- After the prompt, we put outputs as inputs.
- Attention restricted to past outputs (since we generate forward)



- Variations: Beam Search of most likely generation:
  - Maintain k sequences, sample m (expand to km), prune back to top k

(b) Encoder-Decoder Architectures

- The outputs of an encoder transformer become the inputs of a decoder transformer, usually in sandwiched layer: