Practical challenges of RNNs

1. Non-parallel operations that scale with sequence length
2. Long sequences require large memories
3. Hard time when trying to import long-range dependencies

"The cat, which ate a cookie, was hungry."

"The cat was a cookie."

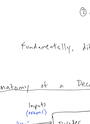
Attention Example: Machine Translation

One solution: Design specialized recurrent units that are better at remembering older hidden states.

1. LSTM (Long Short-Term Memory (unit))
   Hochreiter & Schmidhuber, 1997
2. GRU (Gated recurrent units)
   Kyunghyun Cho et al. 2014

Fundamentally, difficulties remain: Can we find a non-recurrent arch. for sequential data?

Anatomy of a Decoder Transformer



Difficulties lowest of RNNs:

1. Fixed number computational steps
2. Fixed network depth
3. No long-range dependencies

Decoder Transformer Objective: Predict next token given past tokens

- More explicitly, let network model the following probability function:

$$F(\vec{x}) = \vec{y} = \{p_a, p_b, \ldots, p_n\} \approx P(x_t \mid x_1, x_2, \ldots, x_n)$$

network function (decoder) — input token seq. — output prob. distribution

We rely on a cross-entropy loss function:

$$\mathcal{L}(X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}) = -\sum_{i=1}^{N} \log(F_i(\vec{x}_i, \vec{P}))$$

$$\approx -\sum_{i=1}^{N} \log(P(x_i^t \mid x_1^1, x_2^1, \ldots, x_n^1))$$

$$= -\sum_{i=1}^{N} \log(P^{(i)}_{\text{model}})$$

Reduction: Cross-entropy from Bayes:

$$\mathcal{H}_{\vec{P}} = -\sum_{i=1}^{N} \log(P_i^{\text{true}})$$

(for 1 datum and dim) → y_i, p_i, Class

$\lim_{N\to\infty}$ of $\mathcal{H}_{\vec{P}}$ is $\mathcal{H}_{\vec{P}}$

$$= -\log(P^{(i)}_{\text{model}})$$

(for one sample in time series)

$\lim_{\text{batch size}\to\infty}$ of $-\sum_{i=1}^{N} \log(P^{(i)}_{\text{model}})$

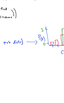
(for one sample in time series)

Outline

1. Positional encoding
2. Attention ("vanilla", "multi-headed", "sparse")
3. How can we make Transformer run with a long ctx?

Positional Encoding

Concept: Add positional/position vector to each input embedding to impart position information



- Pos. enc. can be hidden into embedding space or concatenated (waste of mem.)

Approaches to calculate:

1. Learn pos. emb. matrix

$$\begin{bmatrix} \text{"The"} \\ \text{"cat"} \\ \vdots \end{bmatrix} + \begin{bmatrix} P_{11} & \cdots \\ P_{21} & \cdots \\ \vdots & \end{bmatrix} = \begin{bmatrix} \text{"The"} \\ \text{"cat"} \\ \vdots \end{bmatrix}$$

"positional" embeddings

2. Provide a function

"Attention is all you need" (Vaswani et al. 2017) did this

Attention Mechanisms

Concept: Let network learn "contextualized" embedding for each input as a weighted sum of contextualized embeddings

- Often, I, Q, and V are given same input into residual function

Q: Queries → "What am I looking for?"
K: Keys → "What is in other?"
V: Values → "What information is a match"

- Multi-Head Attention (parallel attention blocks in parallel)

- Extra attention blocks can specialize (analogous to CNN filters)

- Sequential Attention

- Single attention block contextualizes inputs with past information
- Sequential attention blocks contextualize inputs with past/future information
- Sequential " " " " " yet used consistently

- Masked Attention

- Masking attention weights for future tokens is a game-changer network specific approach

Quick fact: "Attention is all you need" (Vaswani et al. 2017)

"Evaluation for position i can only depend on positions less than i."

Training and Generation Procedures

Training Procedure

- Compute loss output relative to target sequence (shifted input sequence)

"<sos>" "The" "cat" "sat"
Decoder Transformer
"The" "cat" "sat" "<eos>"
Loss
Loss
Loss