Alternative Architectures for Neural Machine Translations

Weiting (Steven) Tan
• RNN + Attention Recap
• Convolutional Neural Net
  • How it works in encoder/decoder
  • CNN + Attention
• Transformer
Feed-Forward Layer

- Classic neural network component
- Given an input vector $x$, matrix multiplication $M$ with adding a bias vector $b$
  \[ Mx + b \]
- Adding a non-linear activation function
  \[ y = \text{activation}(Mx + b) \]
- Notation
  \[ y = FF_{\text{activation}}(x) = a(Mx + b) \]
Recurrent Neural Network

- propagate state $s$
- over time steps $t$
- receiving an input $x_t$ at each turn

\[ s_t = f(s_{t-1}, x_t) \]

(state may be computed as a feed-forward layer)
Recurrent Neural Network

- propagate state $s$
- over time steps $t$
- receiving an input $x_t$ at each turn

$$s_t = f(s_{t-1}, x_t)$$

(state may computed may as a feed-forward layer)
\[ S_t = f(S_{t-1}, x_t) = \sigma(C(w_3 S_{t-1}, w_5 z_t + b)) \]

\[ y_t = \text{softmax} \left( \sigma(w_3 y_{St} + b) \right) \]

These equations are related to a neural network model, where \( S_t \) represents the hidden state at time \( t \), \( x_t \) is the input at time \( t \), and \( y_t \) is the output at time \( t \). The functions are composed of linear transformations followed by a non-linear activation function \( \sigma \).

The diagram illustrates the flow of data through the network, with \( S_0, x_1, x_2, x_3, x_4 \) as inputs and \( y_1, y_2, y_3, y_4 \) as outputs. The parameters \( w_3, w_5 \) are weight matrices, and \( b \) is a bias vector.
RNN’s Issue: Vanishing Gradient

- Mitigation:
  - GRU
  - LSTM
RNN with Attention

- Luong et al 2015
- Encoder-Decoder Attention
RNN with Attention

• Luong et al 2015
• Encoder-Decoder Attention

Other ways to compute attention

- Dot product: $a(s_{i-1}, h_j) = s_{i-1}^T h_j$
- Scaled dot product: $a(s_{i-1}, h_j) = \frac{1}{||h_j||} s_{i-1}^T h_j$
- General: $a(s_{i-1}, h_j) = s_{i-1}^T W_a h_j$
- Local: $a(s_{i-1}) = W_a s_{i-1}$
E-D w. Attention

Encoder Out: \[ S_1, S_2, S_3, S_4 \]

Decoder

\[ \alpha = \text{softmax}(h_i^T S_i, h_i^T S_2, h_i^T S_3, h_i^T S_4) \]

\[ C_i = \alpha \odot S_i \] (Dot product)
E-D w. Attention \[
Q = \text{softmax}(h^T S_i, h^T S_2, h^T S_3, h^T S_4)
\]

Encoder Out: \([S_1, S_2, S_3, S_4]\]

\[C_i = \text{softmax}(Q, \text{scalar})\]

Decoder
RNN’s issue: SLOW

Alternative Sequence Modeling
  • Convolutional Neural Networks
  • Transformer (Self-Attention)

Encoder: parallelized computation
Decoder: parallelized during training, autoregressive during inference
ConvNet for 2D Images
ConvNet for Seq2Seq (Encoder)

- First end-to-end neural machine translation model of the modern era [Kalchbrenner and Blunsom, 2013]
- Encoder
\[ \sigma(w^T x + b) \]

\[ x \in \mathbb{R}^d \]

\[ x_c = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^{3d} \]

\[ w \in \mathbb{R}^{d \times 3d} \]
ConvNet with Attention

- Gehring et al. 2017
- **Encoder:**
  - Padding from left and right (keep seq-len)
  - Parallel computation
ConvNet with Attention

- Decoder:
  - Autoregressive
  - Stacking of feed-forward net as decoding is processed
\[ a_i = h_i^T S_i \]
\[ C_i = \sum a_i S_i \]
\[ y_i = \arg\max_j \sigma(W_j C_i) \]

**Encoder Out**

\[ S_1, S_2, S_3, S_4, S_5 \]

**Key & Value**

**Query**

**First layer 2 representation**

**FF**
ConvNet with Attention

- Encoder-Decoder
- Attention is the simple dot product
Transformer

1. Self-Attention and Encoder-Decoder Attention
2. Multi-Head Attention
3. Positional Encoding/Embedding

Jay Alammar [https://jalammar.github.io/illustrated-transformer/](https://jalammar.github.io/illustrated-transformer/)
Transformer

Encoder 1
- Residual Addition with Normalization
- Self-Attention
- Feed Forward
- Residual Addition with Normalization
- Self-Attention

Encoder 2
- Residual Addition with Normalization
- Self-Attention
- Feed Forward
- Residual Addition with Normalization

Decoder 1
- Residual Addition with Normalization
- Encoder-Decoder Attention
- Self-Attention
- Residual Addition with Normalization

Decoder 2
- FULLY CONNECTED LINEAR
- SOFTMAX
- Decorder 2
Transformer Encoder Attention

Scaled Dot-Product Attention

Multi-Head Attention
Input

Embedding

Queries

Keys

Values

Score

Divide by 8 \((\sqrt{d_k})\)

Softmax

Softmax

X

Value

Sum
Self-Attention

\[ a_1 = w_a \cdot x_1 \]
\[ k_1 = w_k \cdot x_1 \quad \in \mathbb{R}^h \]
\[ v_1 = w_v \cdot x_1 \]

\[ \beta \in \mathbb{R}^d \]
\[ w_a \in \mathbb{R}^{m \times d} \]
\[ w_k \in \mathbb{R}^{m \times d} \]
\[ w_v \in \mathbb{R}^{m \times d} \]

\[ a_1 = q_1 v_1 \]
\[ a_1 v_1 \]
\[ a_1 v_2 \]
\[ a_1 v_3 \]
\[ a_1 v_4 \]

\[ z = \text{softmax}_i \left[ q_1 v_1, q_2 v_2, q_3 v_3, q_4 v_4 \right] \]

\[ z_i = e^{z_i} \cdot v_i \quad \in \mathbb{R}^h \]
Encoder Output: \([z_1, z_2, z_3, z_4]\)

Multi-head:
\[
\begin{align*}
W_0 \cdot w & \rightarrow [z_1', z_2', z_3', z_4'] \\
W_k \cdot w & \rightarrow [z_1^k, z_2^k, z_3^k, z_4^k] \\
W_i \cdot w & \rightarrow [z_1^i, z_2^i, z_3^i, z_4^i] \\
\end{align*}
\]

For \(x_i\) now we have \(e^{\mathbf{z}_h} \in \mathbb{R}^h\) and \(W \in \mathbb{R}^{h \times h}\) for FF layers.
Decoder’s Self-attention

\[ a, b, c, d \quad \rightarrow \quad v, w, x, y \]

Attention:

\[ Z = \text{softmax} \left( [ q^T u, q^T v, q^T w, q^T x] \right) \]

\[ z_i = \sum_{i=1}^n a_i v_i \in \mathbb{R}^k \]

for \( x_1 \)

\[ Z = \text{softmax} \left( [ q^T u, -\text{int}, -\text{int}, -\text{int}] \right) \]

for \( x_2 \)

\[ Z = \text{softmax} \left( [ q^T u, q^T v, -\text{int}, -\text{int}] \right) \]

--

for \( x_4 \)

\[ Z = \text{softmax} \left( [ q^T u, q^T v, q^T w, q^T x] \right) \]

\[ \text{mask: } \left[ \begin{array}{c} \_ \_ \_ \_ \\ -\text{int} \end{array} \right] \]
Transformer Decoder

• Self-attention with mask
• Encoder-Decoder Attention
  • Query is from decoder
  • Key and Value are from Encoder
Encoder-Decoder Attention in transformer

Encoder Output

\[ [S_1, S_2, S_3, S_4] \]

both key and value

Encoder Output \[ \rightarrow [h_1^T s_1, h_2^T s_2, \ldots, h_4^T s_4] \]

\[ q = \text{softmax}(h^T s) \]

\[ z_i = \leq q_i, s_i \]

\[ h_4 \]

query

self-attention

\[ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \]
Positional Embedding

INPUT

EMBEDDING WITH TIME SIGNAL

EMBEDDING WITH TIME SIGNAL

POSITIVE ENCODING

EMBEDDINGS

x₁ = t₁ + x₁

x₂ = t₂ + x₂

x₃ = t₃ + x₃

INPUT

Je

suis

étudiant
Positional Embedding

\[
\overrightarrow{p_t} = f(t) := \begin{cases} 
\sin(\omega_k \cdot t), & \text{if } i = 2k \\
\cos(\omega_k \cdot t), & \text{if } i = 2k + 1 
\end{cases}
\]

\[
\omega_k = \frac{1}{10000^{2k/d}}
\]
Positional Embedding

https://github.com/jalammar/jalammar.github.io/blob/master/notebooks/transformer/transformer_positional_encoding_graph.ipynb