Precept 7: seq2seq, attention, and transformers

Samyak Gupta
04/7/2023
PSA

Start assignment 4 early
It’s more involved than the previous ones
Agenda

- seq2seq
- Attention
- Transformers
Machine Translation

“Source” language  ➔  “Target” language

Difficult due to nuances of language
seq2seq models

**Goal:** Transform from a source sequence to a target sequence
seq2seq (with RNNs) for machine translation

Key idea: use two RNNs
seq2seq (with RNNs) for machine translation

Key idea: use two RNNs

(In assignment 4, your encoder and decoder will be based on transformers instead of RNNs)
seq2seq encoder

**Encoder**: Transform some source sequence into a hidden representation
Encoder: Transform some source sequence into a hidden representation

Step 1: Transform word to a vector (using embeddings matrix)
seq2seq encoder

**Encoder:** Transform some source sequence into a hidden representation

**Step 1:** Transform word to a vector
(using embeddings matrix)

Sentence: hello world.
**seq2seq encoder**

**Encoder:** Transform some source sequence into a hidden representation

**Step 1:** Transform word to a vector (using embeddings matrix $E^{(s)}$)

Sentence: *hello world*. 

<table>
<thead>
<tr>
<th>Vocab Size</th>
<th>E^{(s)}</th>
<th>Embedding Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>world</td>
<td></td>
</tr>
</tbody>
</table>
seq2seq encoder

**Encoder**: Transform some source sequence into a hidden representation

**Step 1**: Transform word to a vector (using embeddings matrix $E^{(s)}$)

Sentence: *hello world*.
seq2seq encoder

**Encoder:** Transform some source sequence into a hidden representation

**Step 1:** Transform word to a vector (using embeddings matrix)
**Step 2:** Compute hidden state using word embedding and last hidden state

Sentence: *hello world*.
seq2seq encoder

**Encoder**: Transform some source sequence into a hidden representation

**Step 1**: Transform word to a vector (using embeddings matrix)

**Step 2**: Compute hidden state using word embedding and last hidden state

Repeat!
seq2seq encoder

**Encoder:** Transform some source sequence into a hidden representation

**Step 1:** Transform word to a vector (using embeddings matrix)

**Step 2:** Compute hidden state using word embedding and last hidden state

Sentence: *hello world*.
seq2seq encoder

**Encoder**: Transform some source sequence into a hidden representation

**Step 1**: Transform word to a vector (using embeddings matrix)

**Step 2**: Compute hidden state using word embedding and last hidden state
seq2seq encoder

**Encoder:** Transform some source sequence into a hidden representation

**Step 1:** Transform word to a vector (using embeddings matrix)

**Step 2:** Compute hidden state using word embedding and last hidden state

Sentence: *hello world*
**seq2seq encoder**

**Encoder:** Transform some source sequence into a hidden representation

**Step 1:** Transform word to a vector (using embeddings matrix)

**Step 2:** Compute hidden state using word embedding and last hidden state

**Key Idea:** We’ve converted a variable length sequence to a fixed length representation

*Sentence: hello world*. (encoded representation)
seq2seq decoder

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding
**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding (using matrix $E^{(t)}$)
seq2seq decoder

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding (using matrix $E^{(t)}$)

- Vocab Size
- Embedding Size

- $<\text{bos}>$
- bonjour
- salut
- $<\text{eos}>$

$E^{(t)}$, $h^{enc}$, word embedding, $<\text{bos}>$
seq2seq decoder

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding

**Step 2:** Compute hidden state using word embedding and last hidden state
seq2seq decoder

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding

**Step 2:** Compute hidden state using word embedding and last hidden state

**Step 3:** Predict word using hidden state
seq2seq decoder

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding

**Step 2:** Compute hidden state using word embedding and last hidden state

**Step 3:** Predict word using hidden state
seq2seq decoder

**Decoder:** Using an encoded representation, predict a target sequence

1. **Step 1:** Transform previous predicted token to word embedding
2. **Step 2:** Compute hidden state using word embedding and last hidden state
3. **Step 3:** Predict word using hidden state

Recall: Output embeddings give us a probability distribution over outputs
**seq2seq decoder**

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding

**Step 2:** Compute hidden state using word embedding and last hidden state

**Step 3:** Predict word using hidden state

Recall: Output embeddings give us a probability distribution over outputs. Picking only the highest probability is called “greedy” decoding

\[
y_t = \text{softmax}(h_t W_{o}^T)
\]
**seq2seq decoder**

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding

**Step 2:** Compute hidden state using word embedding and last hidden state

**Step 3:** Predict word using hidden state
seq2seq decoder

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding
**Step 2:** Compute hidden state using word embedding and last hidden state
**Step 3:** Predict word using hidden state
seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding

**Step 2:** Compute hidden state using word embedding and last hidden state

**Step 3:** Predict word using hidden state

Repeat!
**seq2seq decoder**

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding

**Step 2:** Compute hidden state using word embedding and last hidden state

**Step 3:** Predict word using hidden state
seq2seq decoder

**Decoder**: Using an encoded representation, predict a target sequence

**Step 1**: Transform previous predicted token to word embedding

**Step 2**: Compute hidden state using word embedding and last hidden state

**Step 3**: Predict word using hidden state
seq2seq decoder

**Decoder:** Using an encoded representation, predict a target sequence

**Step 1:** Transform previous predicted token to word embedding

**Step 2:** Compute hidden state using word embedding and last hidden state

**Step 3:** Predict word using hidden state

Repeat process until model predicts `<eos>`
Key idea: Improve quality and variety of generations by tracking $k$ best hypotheses at each step.
Beam Search Decoding Example \((k=2)\)
Beam Search Decoding Example ($k=2$)
Beam Search Decoding Example ($k=2$)
Beam Search Decoding Example ($k=2$)
Beam Search Decoding Example ($k=2$)
Beam Search Decoding Example \((k=2)\)
Beam Search Decoding Example ($k=2$)
Beam Search Decoding Example ($k=2$)

\[
\log P\text{ (arrived the|x)} = -2.3 \\
\log P\text{ (arrived witch|x)} = -3.9 \\
\log P\text{ (the|x)} = -1.6 \\
\log P\text{ (the green|x)} = -1.6 \\
\log P\text{ (the witch|x)} = -2.1 \\
\log P\text{ (arrived|the, green, witch, x)} \\
\log P\text{ (END|the, green, witch, arrived, x)} \\
\log P\text{ (the green witch arrived|x)} = \log P\text{ (the|x)} + \log P\text{ (green|the,x)} + \log P\text{ (witch|the, green,x)} + \log P\text{ (arrived|the, green, witch, x)} + \log P\text{ (END|the, green, witch, arrived, x)}
\]
Beam Search Decoding Example \((k=2)\)

**Caveat:** The log probabilities should be normalized!

\[
\begin{align*}
\text{log } P(\text{arrived}|x) &= -.69 \\
\text{the } &= -.69 \\
\text{log } P(\text{arrived witch}|x) &= -.2.3 \\
\text{witch } &= -.3.9 \\
\text{log } P(\text{the}|x) &= -.92 \\
\text{green } &= -.92 \\
\text{log } P(\text{the green}|x) &= -.1.6 \\
\text{log } P(\text{the green witch}|x) &= -.51 \\
\text{log } P(\text{arrived the green witch}|x) &= -.2.1 \\
\text{log } P(\text{arrived the green witch came}|x) &= -.1.6 \\
\text{log } P(\text{arrived the green witch by}|x) &= -.61 \\
\text{log } P(\text{arrived the green witch arrived}|x) &= -.36 \\
\text{log } P(\text{arrived the green witch arrived}|x) &= -.51 \\
\text{log } P(\text{arrived the green witch arrived}|x) &= -.2.2 \\
\text{log } P(\text{arrived the green witch arrived}|x) &= -.4.4 \\
\text{log } P(\text{arrived the green witch arrived}|x) &= -.3.8 \\
\text{END } &= -.2.5 \\
\text{END } &= -.7.2 \\
\text{END } &= -.4.8 \\
\text{END } &= -.2.3 \\
\text{END } &= -.2.7
\end{align*}
\]
Vanilla seq2seq models are limited!

- Encoded representation is a “bottleneck” (must contain all relevant information from context!)
- Suffers from same issues as RNNs:
  - Vanishing gradients
  - Inefficient utilization of hardware
Vanilla seq2seq models are limited!

- Encoded representation is a “bottleneck” (must contain all relevant information from context!)
- Suffers from same issues as RNNs:
  - Vanishing gradients
  - Inefficient utilization of hardware

Adding attention to seq2seq can help solve representation bottleneck
seq2seq with Attention

**Key Idea:** Let the decoder pick the parts of the encoder hidden states that it needs (i.e. “pay attention to” specific encoder hidden states)
seq2seq encoder with Attention

Encoder (with attention): Exactly the same as before! (except we also use hidden states $h_1$, $h_2$, $h_3$)

Step 1: Transform word to a vector (using embeddings matrix)
Step 2: Compute hidden state using word embedding and last hidden state

Sentence: hello world.

(encoded representation)
seq2seq decoder with Attention

Decoder (with Attention): Using all hidden states from the encoder, predict a target sequence

Step 1: Transform previous predicted token to word embedding
Step 2: Compute decoder hidden state using word embedding
Step 3 (new): Compute context for decoder using all encoder hidden states
Step 4: Predict word using hidden state combined with context vector
seq2seq decoder with Attention

Decoder (with Attention): Using all hidden states from the encoder, predict a target sequence
Attention: Mathematical formulation

1. Prepare inputs
   - Encoder hidden states: $h_1^{enc}, \ldots, h_n^{enc}$
     (n: # of words in source sentence)
   - Decoder hidden state at time $t$: $h_t^{dec}$

2. Score each hidden state
   - Attention scores:
     $$e^t = [g(h_1^{enc}, h_t^{dec}), \ldots, g(h_n^{enc}, h_t^{dec})] \in \mathbb{R}^n$$

3. Softmax the scores
   - Attention distribution:
     $$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^n$$

4. Multiply each vector by its softmaxed score
   - Weighted sum of encoder hidden states:
     $$a_t = \sum_{i=1}^{n} \alpha_i h_i^{enc} \in \mathbb{R}^h$$

5. Sum up the weighted vectors
   - Context vector for decoder time step #4

Combine $a_t$ and $h_t^{dec}$ to predict next word
Transformer Architecture

Encoder

Decoder

Source Sequence Input

Target Sequence (i.e. current progress generating words)
Transformer Encoder

Source sequence $(x_1, \ldots, x_n)$
Transformer Encoder: Positional + Word Embedding

Source sequence \((x_1, \ldots, x_n)\)

Embedded source sequence \(\mathbb{R}^{n \times d_1}\)

Input and Positional Embedding

- **Embedding with Time Signal**
- **Positional Encoding**
- **Embeddings**

Input: Je, suis, étudiant
Transformer Encoder: Multi-Head Self Attention

**Self-Attention:**

\[ W^Q_i \in \mathbb{R}^{d_1 \times d_q}, \ W^K_i \in \mathbb{R}^{d_1 \times d_k}, \ W^V_i \in \mathbb{R}^{d_1 \times d_v} \]

**Step 1:**

\[ X \times W^Q = Q \]

\[ X \times W^K = K \]

\[ X \times W^V = V \]

**Step 2:**

\[
\operatorname{softmax}\left( \frac{Q K^T}{\sqrt{d_k}} \right)
\]

**MultiHead Attention:**

\[ W^O \in \mathbb{R}^{d \times d_2} \]

\[ \text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \]

\[ \text{head}_i = \text{Attention}(XW^Q_i, XW^K_i, XW^V_i) \]
Transformer Encoder: Multi-Head Self Attention

Self-Attention:

\[ W^Q_i \in \mathbb{R}^{d_1 \times d_q}, \ W^K_i \in \mathbb{R}^{d_1 \times d_k}, \ W^V_i \in \mathbb{R}^{d_1 \times d_v} \]

Step 1:

\[ \text{Embedded source sequence} \]

Step 2:

\[ \text{softmax} \left( \frac{Q \cdot K^T}{\sqrt{d_k}} \right) \]

In practice, \( d_1 = d_2 \)

After Multi-Head Attention

\[ \mathbb{R}^{n \times d_1} \]

Embedded source sequence

\[ \mathbb{R}^{n \times d_1} \]

Source sequence

\((x_1, \ldots, x_n)\)

"Self" attention means Q, K, V are all computed from a single sequence

MultiHead Attention:

\[ W^O \in \mathbb{R}^{d \times d_2} \]

\[ \text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \]

\[ \text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V) \]
Transformer Encoder: Add & Norm

Add & Norm:

LayerNorm(x + Sublayer(x))

LayerNorm

\[ y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} \cdot \gamma + \beta \]
Transformer Encoder: Feed Forward

Feed Forward

\[ \text{FFN}(x_i) = \text{ReLU}(x_i W_1 + b_1) W_2 + b_2 \]

- \( W_1 \in \mathbb{R}^{d \times d_{ff}} \), \( b_1 \in \mathbb{R}^{d_{ff}} \)
- \( W_2 \in \mathbb{R}^{d_{ff} \times d} \), \( b_2 \in \mathbb{R}^{d} \)

Compute transformation over each value in the sequence independently
Transformer Encoder: Final Add & Norm

After Final Add & Norm
\[ \mathbb{R}^{n \times d_1} \]

After Feed Forward
\[ \mathbb{R}^{n \times d_1} \]

After Add & Norm
\[ \mathbb{R}^{n \times d_1} \]

After Multi-Head Attention
\[ \mathbb{R}^{n \times d_1} \]

Embedded source sequence
\[ \mathbb{R}^{n \times d_1} \]

Source sequence \((x_1, \ldots, x_n)\)

Add & Norm:
\[ \text{LayerNorm}(x + \text{Sublayer}(x)) \]

LayerNorm
\[ y = \frac{x - \text{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \ast \gamma + \beta \]
Transformer Decoder:

Target sequence \( (<\text{bos}>, x_1, \ldots, x_m) \)

Output and Positional Embedding

- **EMBEDDING WITH TIME SIGNAL**
  - \( x_1 \)
  - \( x_2 \)
  - \( x_3 \)

- **POSITIONAL ENCODING**
  - \( t_1 \)
  - \( t_2 \)
  - \( t_3 \)

- **EMBEDDINGS**
  - \( x_1 \)
  - \( x_2 \)
  - \( x_3 \)

- Embedded target sequence \( \mathbb{R}^{m \times d_1} \)

- Output Embedding

- Outputs (shifted right)
Transformer Decoder: Masked Multi-Head Attention

Target sequence \(<\text{bos}>, x_1, \ldots, x_m\)

Masked Multi-Head Attention

Embedded target sequence

Output Embedding

Positional Encoding

Outputs (shifted right)

MultiHead Attention:

\[ W^O \in \mathbb{R}^{d_x \times d_z}, W^K \in \mathbb{R}^{d_x \times d_k}, W^V \in \mathbb{R}^{d_x \times d_v} \]

**Step 1:**

\[ \text{Elementwise Multiply by Mask} \] (equivalent to setting masked indices to \(-\infty\))

**Step 2:**

\[ \sqrt{d_k} \]

**Step 3:**

\[ \text{softmax}() \]
Transformer Decoder:

Add & Norm:

LayerNorm($x + \text{Sublayer}(x)$)

LayerNorm

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} \cdot \gamma + \beta$$
Transformer Decoder: Multi-Head (Cross) Attention

Cross-Attention: \( W^Q_i \in \mathbb{R}^{d_1 \times d_q}, W^K_i \in \mathbb{R}^{d_1 \times d_k}, W^V_i \in \mathbb{R}^{d_1 \times d_v} \)

Step 1:
- \( Q \times W^Q \rightarrow Q \)
- \( K \times W^K \rightarrow K \)
- \( V \times W^V \rightarrow V \)

Step 2:
- \( \text{softmax}(\frac{K^T Q}{\sqrt{d_k}}) \)

“Cross” attention means Q, K, V are computed from separate sequences

MultiHead Attention: \( W^O \in \mathbb{R}^{d \times d_2} \)

\[
\text{MultiHead}(Q, K, V) = \text{Concat} (\text{head}_1, ..., \text{head}_h) W^O
\]

\[
\text{head}_i = \text{Attention}(XW^Q_i, XW^K_i, XW^V_i)
\]
Transformer Decoder: Multi-Head (Cross) Attention

Masked Multi-Head Attention $\mathbb{R}^{m \times d_1}$
After Add & Norm $\mathbb{R}^{m \times d_1}$
Masked Multi-Head Attention $\mathbb{R}^{m \times d_1}$
Embedded target sequence $\mathbb{R}^{m \times d_1}$
Target sequence $(<\text{bos}>, x_1, \ldots, x_m)$

Cross-Attention: $W^Q_i \in \mathbb{R}^{d_1 \times d_q}$, $W^K_i \in \mathbb{R}^{d_1 \times d_k}$, $W^V_i \in \mathbb{R}^{d_1 \times d_v}$

Step 1:
- $Q = X \cdot W^Q$
- $K = X \cdot W^K$
- $V = X \cdot W^V$

Step 2:
- "Cross" attention means $Q$, $K$, $V$ are computed from separate sequences

MultiHead Attention: $W^O \in \mathbb{R}^{d \times d_2}$

MultiHead($Q, K, V$) = Concat(head$_1$, ..., head$_h$)$W^O$

head$_i$ = Attention($XW^Q_i$, $XW^K_i$, $XW^V_i$)

“Cross” attention means $Q$, $K$, $V$ are computed from separate sequences
Transformer Decoder: Add & Norm

Add & Norm:
\[
\text{LayerNorm}(x + \text{Sublayer}(x))
\]

LayerNorm
\[
y = \frac{x - \text{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \times \gamma + \beta
\]
Transformer Decoder: Feed Forward

Feed Forward
\[ \mathbb{R}^{m \times d_1} \]
Add & Norm
\[ \mathbb{R}^{m \times d_1} \]
Masked Multi-Head Attention
\[ \mathbb{R}^{m \times d_1} \]
After Add & Norm
\[ \mathbb{R}^{m \times d_1} \]
Masked Multi-Head Attention
\[ \mathbb{R}^{m \times d_1} \]
Embedded target sequence
\[ \mathbb{R}^{m \times d_1} \]

Output Probabilities

Softmax

Linear

Add & Norm

Feed Forward

Add & Norm

Multi-Head Attention

Add & Norm

Masked Multi-Head Attention

Output Embedding

Positional Encoding

Target sequence
\(<\text{bos}>, x_1, \ldots, x_m)\)

Feed Forward

\[ \text{FFN}(x_i) = \text{ReLU}(x_i W_1 + b_1) W_2 + b_2 \]

\[ W_1 \in \mathbb{R}^{d \times d_{ff}}, \quad b_1 \in \mathbb{R}^{d_{ff}} \]

\[ W_2 \in \mathbb{R}^{d_{ff} \times d}, \quad b_2 \in \mathbb{R}^{d} \]
Transformer Decoder: Add & Norm

Add & Norm: $\mathbb{R}^{m \times d_1}$
Feed Forward: $\mathbb{R}^{m \times d_1}$
Add & Norm: $\mathbb{R}^{m \times d_1}$
Masked Multi-Head Attention: $\mathbb{R}^{m \times d_1}$
After Add & Norm: $\mathbb{R}^{m \times d_1}$
Masked Multi-Head Attention: $\mathbb{R}^{m \times d_1}$
Embedded target sequence: $\mathbb{R}^{m \times d_1}$

Target sequence: $(\text{<bos>}, x_1, \ldots, x_m)$

Add & Norm:

LayerNorm$(x + \text{Sublayer}(x))$

LayerNorm

$$y = \frac{x - \text{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \ast \gamma + \beta$$
Transformer: Final output

\[ \text{softmax}(W_\theta h_i) \] Compute transformation over concatenated states