Transformer; Syntax in SMT

Ondřej Bojar

April 25, 2019
Reminder: Seq2seq with Attention

\[
\begin{align*}
&\langle s \rangle \\
&x_1 \downarrow \\
&h_0 \quad h_1 \quad h_2 \quad h_3 \quad h_4 \\
&\alpha_0 \times \alpha_1 \times \alpha_2 \times \alpha_3 \\
&s_{i-1} \quad s_i \quad s_{i+1} \\
&\sim y_i \quad \sim y_{i+1}
\end{align*}
\]
Inputs:

decoder state $s_i$

encoder states $h_j = [\overrightarrow{h_j}; \overrightarrow{h_j}] \ \forall i = 1 \ldots T_x$

where $\overrightarrow{h_j} = \text{RNN}_\text{enc}(h_{j-1}, x_j) = \tanh(U_e \overrightarrow{h_{j-1}} + W_e E_e x_j + b_e)$

Attention energies: $e_{ij} = v_a^\top \tanh (W_a s_{i-1} + U_a h_j + b_a)$

Attention distribution: $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$

Context vector: $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$
Decoder state:

\[ s_i = \tanh(U_d s_{i-1} + W_d E_d \hat{y}_{i-1} + C c_i + b_d) \]

Output projection:

\[ t_i = \tanh(U_o s_i + W_o E_d \hat{y}_{i-1} + C_o c_i + b_o) \]

...context vector is mixed with the hidden state

Output distribution:

\[ p(y_i = k \mid s_i, y_{i-1}, c_i) \propto \exp(W_o t_i)_k + b_k \]
Transformer
Attention is All You Need (Vaswani et al., 2017)
Transformer Detailed Walkthroughs

Transformer Illustrated:
- http://jalammar.github.io/illustrated-transformer/

Paper annotated with PyTorch code:
- http://nlp.seas.harvard.edu/2018/04/03/attention.html

Summary at Medium:
- https://medium.com/@adityathiruvengadam/
  transformer-architecture-attention-is-all-you-need-aecc
Self-Attention

See slides 46–53 (pages 53–60) by Jindřich Libovický.

Three uses of multi-head attention in Transformer

- **Encoder-Decoder Attention:**
  - $Q$: previous decoder layers; $K = V$: outputs of encoder
  - $⇒$ Decoder positions attend to all positions of the input.

- **Encoder Self-Attention:**
  - $Q = K = V$: outputs of the previous layer of the encoder
  - $⇒$ Encoder positions attend to all positions of previous layer.

- **Decoder Self-Attention:**
  - $Q = K = V$: outputs of the previous decoder layer.
  - Masking used to prevent depending on future outputs.
  - $⇒$ Decoder attends to all its previous outputs.
Syntax in NMT
Ways of Adding Syntax to NMT

- Source Syntax Attached to Tokens.
  - Factored NMT (+standard ‘flat’ word embeddings).
  - Graph-Convolutional Networks (word embs. reflect structure).

- Source Syntax Reflected in Encoder.
  - Syntax reflected in network structure.
  - We are skipping TreeLSTM.

- Target Syntax through:
  - Interleave.
  - Multi-Decoder (sharing ends after encoder or attention or decoder).

- Somewhat suspicious results...
Why Syntax in NMT?

Motivation same as in SMT: long-distance dependencies.

- **What** city is the Taj Mahal in?
- **Where** is the Taj Mahal ∅?

The handling of the *in* depends on the *What/Where*.

- CCG tags for *is* differ \(\Rightarrow\) dependency highlighted.

- **What** \((S[wq]/(S[q]/NP))/N\) city is \((S[q]/P\ P)/NP\) the Taj Mahal in?
- **Where** \(S[wq]/(S[q]/NP)\) is \((S[q]/NP)/NP\) the Taj Mahal?
Linguistic Features in NMT in General

- **Source word factors easy to incorporate:**
  - Concatenate embeddings of the various factors.
  - POS tags, morph. features, source dependency labels help en↔de and en→ro (Sennrich and Haddow, 2016).

- **Target word factors:**
  - Interleave for morphology: (Tamchyna et al., 2017)
    
    Source (BPE) there are a million different kinds of pizza.
    Baseline (BPE) existují miliony druhů piz@@ zy.
    Interleave VB3P existovat NNIP1 milion NNIP2 druh NNFS2 pizza Z:

  - Interleave for syntax: (Nadejde et al., 2017)
    
    Source BPE Obama receives Net+ an+ yahu in the capital of USA
    Target NP Obama ((S[dcl]\NP)/PP)/NP receives NP Net+ an+ yahu PP/NP in NP.
Source side: Enrich tokens.
- Concatenate the embeddings of the individual vocabularies.
- Derive word embeddings from syntax.

Target side:
- Predict interleaved
  - Tamchyna et al. (2017) did this for morphology.
- Have two decoders
  - Worked worse since the outputs differed in length.
Figure 2: A 2-layer syntactic GCN on top of a convolutional encoder. Loop connections are depicted with dashed edges, syntactic ones with solid (dependents to heads) and dotted (heads to dependents) edges. Gates and some labels are omitted for clarity.
- CNN-derived embeddings of nodes’ syntactic neighbourhood included: (parent, siblings).
- Two mechanisms:
  - Concatenated to standard embeddings.
  - Separate attention over these word-level annotations
- +1.6 BLEU on Chinese-to-English.
Tree-GRU Encoder:
- Constituency syntax of the tree provides additional states.
Bidirectional tree encoder.

- Can be seen as many RNNs running from each word up to the root and back to the word.
Tree coverage model:

- Attention coverage depends on source syntax.
- Without it (left), output is repeated.
(Tree) Coverage in Attention

\[
C_{j,i} = \text{GRU}(C_{j-1,i}, \alpha_{j,i}, d_{j-1}, h_i)
\]

\[
e_{j,i} = v_a^T \tanh(W_a d_{j-1} + U_a h_i + V_a C_{j-1,i})
\]

\(C_{j,i}\) coverage of src word \(i\) when producing tgt word \(j\),

\(h_i\) encoder state at word \(i\), \(d_j\) decoder state at word \(j\),

\(\alpha_{j,i}\) attention weight of src \(i\) when producing word \(j\),

\(e_{j,i}\) “energy” for tgt word \(j\) from src word \(i\) (i.e. \(\alpha\) before softmax)

\[
C_{j,i} = \text{GRU}(C_{j-1,i}, \alpha_{j,i}, d_{j-1}, h_i, C_{j-1,L(i)}, \alpha_{j,L(i)})
\]

\[
C_{j-1,R(i)}, \alpha_{j,R(i)}
\]

\(L(i)\) and \(R(i)\): left and right childs of (binarized) src tree for non-leaf nodes.
Suspicious Results on Multi-Tasking
Promoting Source Syntax in Transformer

See the poster of CICLING 2019.
Dominik Macháček:
My students Dan Kondratyuk and Ronald Cardenas retried Nadejde et al. (2017) with:

- sequence-to-sequence model,
- Transformer model.

Predicting target syntax using:

- a secondary decoder
  (The sequence of CCG tags may not match the translated sentence.)
- interleaving.

As tags, they used:

- correct CCG tags, • random tags, • a single dummy tag.
Predicting Target Syntax
Predicting Target Syntax

![Graph showing the progression of Seq2seq models over training steps (millions)]
Predicting Target Syntax

![Graph showing training steps (millions) for different models: Baseline, CCG, Random, Same, and Interleaved Transformer. The x-axis represents training steps in millions, ranging from 0 to 28. The y-axis represents Interleaved, ranging from 0 to 30. The graph includes lines for Baseline (blue), CCG (red), Random (brown), Same (black crosses), and Interleaved Transformer (blue).]
Predicting Target Syntax

Transformer

Training steps (millions)

Multi-Decoder

Baseline CCG Random Same

Transformer
Transformer is a great replacement for RNN:

- Explicit syntax can be useful.
  - Very many options possible.
  - All so far tested on a handful of languages and in incomparable settings.
  - Some gains hard to reproduce.
  - The best setup very unclear at the moment.

... Stay tuned or bring your own option.


