ML & Opt Lecture 22

Sequence-to-Sequence Modeling
- architecture using RNNs
- Training & Inference
- Teacher forcing
- Perplexity - performance metric
- Tricks for better performance
- Attention
Sequence-to-Sequence Modeling

“Sutskever et al.
Sequence to sequence learning with neural networks”

Goal: given input-sequence of tokens \((x_1, \ldots, x_T)\)
predict matching output-sequence of tokens \((y_1, \ldots, y_{T'})\)

Canonical example: Machine Translation

Issue: \(T \neq T'\) in general so the many-to-many design paradigm for RNNs may not fit

Soln: encoder-decoder architecture
Seq2Seq Encoder-Decoder architecture

Encoder

Decoder

STEP 1: Input sentence

STEP 2: Encode the input sentence

STEP 3: Decode the sentence

Teacher forcing: only possible during training; helps train faster
Training

- train the encoder & decoder simultaneously (using BPTT) so that given the input sequence \( (x_1, \ldots, x_T) \) and output sequence \( (y_1, \ldots, y_T) \)
  - encoder learns an informative \( h_T \) for decoding
  - decoder learns how to, given \( h_T \) and a special \(<\text{START}>\) token, generate the correct output followed by an \(<\text{END}>\) token.

- Since the output \( \hat{y}_t \) depends on \( \hat{y}_{t-1} \) we can have very slow training if \( \hat{y}_t \) is inaccurate at the start of the sequence, because the errors compound. To mitigate, during training we can use teacher forcing: use \( y_{t-1} \) (the ground truth) as the input to predict \( \hat{y}_t \)
- Given input sequence \((X_1, \ldots, X_T)\) use encoder to obtain \(h_T\).
- Initialize the decoder w/ hidden state \(h_T\) and feed it the \(<\text{START}>\) token.
- At each time, feed \(j_{t-1}\) into the decoder to predict the next token and take \(j_t\) to be the most likely token.
Practical tips

- Use teacher forcing (sometimes)
- Use deep LSTMs in the encoder and decoder
- Reverse the order of the source sequence to introduce shorter length dependencies w/ the target sequence

Muéy bien y tu -> Very well and you

tu y bien Muéy -> Very well and you

- Use "beam decoding" in practice (see Sutskever et al.)
Seq2Seq models w/ attention

Issue: regardless of length of input sequence, all context/historical dependence of the output sequence is contained in a d-dimensional vector $h_t$.

This leads to poor performance for long sequences.

A remedy introduced in Bahdanau et al. 2015, “Neural Machine Translation By Jointly Learning to Align and Translate”:

- allow each output token to attend to each individual hidden state in the input sequence. Captures the idea of focusing on the relevant portion of input sequences. Avoids the need to capture everything relevant to all outputs $\hat{y}_1, \ldots, \hat{y}_T$ in one vector $h_T$. 
Seq2Seq w/ attention

replace our previous hidden state update for the decoder

\[ S_{t-1} \xrightarrow{\text{attention}} S_t \quad \text{i.e.} \quad S_t = g(S_{t-1}, \tilde{y}_t) \]

w/ first finding which historical hidden states from
the input are most relevant

\[ S_{t-1}, [h_1, \ldots, h_T] \xrightarrow{\text{attention}} \alpha_t \in \mathbb{R}^T \text{ a probability distribution over } [h_1, \ldots, h_T] \]
then form a context vector

\[ c_t = \sum_{i=1}^{T} (\alpha_t)_i h_i \quad \text{and update with} \quad S_t = g(S_{t-1}, c_t) \]
Diagram of our Decoder using Attention

\[ h_T = S_0 \]

\[ c_1 = \sum_{i=1}^{T} (\alpha_1)_i h_i \]

\[ c_2 = \sum_{i=1}^{T} (\alpha_2)_i h_i \]
Options for computing attention vectors $\alpha_t$

- **Option 1** (used in Bahdanau et al. 2015)
  
  Take
  
  $$\alpha_i = \mathbf{v}^T \tanh(\mathbf{W} \begin{bmatrix} \mathbf{h}_i \\ s_{t-1} \end{bmatrix})$$
  
  for $i = 1, \ldots, T$

  Parameters to be learned

  $$\alpha_t = \text{Softmax}(\mathbf{\alpha})$$

  "MLP attention"?

- **Option 2** (Dot product attention)

  Idea: generate keys and queries as linear transforms of $\mathbf{h}_i$ and $s_{t-1}$, respectively, and take logits $\mathbf{\alpha}_i$ to be their inner products

  $$\mathbf{w}_h \mathbf{h}_i = k_i \quad \mathbf{\alpha}_i = \langle k_i, q_{t-1} \rangle$$

  $$\mathbf{w}_q s_{t-1} = q_{t-1} \quad \alpha_t = \text{Softmax}(\mathbf{\alpha})$$
Complexity of using attention

- Clear we have introduced parameters

Quadratic dependence on the sequence lengths?

Effort/time to use a RNN seq2seq model:

1) Simple RNN models: $O(T+T')$

2) RNN attention models: $O(TT')$

because for each output token, must compute $a_t \in \mathbb{R}^T$ by comparing to all inputs, then form a weighted average of all inputs.