ML and Optimization  Lecture 21

- Tools for using RNNs for NLP problems:
  - Embedding layer and tokenization
- RNN example: AG-NEWs classification
- Long Short Term Memory (LSTM) networks
- Bidirectional and deep RNNs
Example Application (NLP)

Classification of text sequences

Input: “Iraq Halts Oil Imports.”
Output: class 3 (business/economic news)

Idea: split input sentence into tokens (tokenization), convert them to vectors via a lookup table (word embeddings), then use the many-to-one RNN design followed by a softmax layer.
We use a three step process:

1. Tokenize the input
2. Look up the index of each token into our vocabulary
3. Use a look-up table (embedding layer) to find the vectors corresponding to each word

Inside the model:

- Use the index of each token to find the corresponding embedding vector.
- The resulting vectors are then passed through the model.

We find it useful to augment the input sequence to indicate the start and end of inputs, e.g., `<BOS>` and `EOS`.
Ex:

"Iraq Halts Oil Imports"

["<BOS>", "Iraq", "Halts", "Oil", "Imports", "<EOS>"]

[1, 23, 532, 11, 72, 2]

in each minibatch, we use the updated embeddings for the tokens

d = embedding dimension

[ ]

[ ]

[ ]

[ ]

[ ]

[ ]
Let $V = \begin{bmatrix} v_1^T \\ \vdots \\ v_m^T \end{bmatrix} \in \mathbb{R}^{m \times d}$ be the look-up table for our vocabulary.

The final architecture for this NLP task looks like

```
h_0 → h_1 → h_2 → \ldots → h_T
```

with

```
1 \times 1 → 23 \times 23 → 532 \times 532 → \text{logits layer}
```

where $h_0$, $h_1$, $h_2$, \ldots, $h_T$ are the hidden states, and $1$, $23$, $532$ are the respective input dimensions.
To train, we backprop (BPTT) over the logit layer, the parameters for the RNN, and the embedding layer.

**Practical considerations:**

- to embed out-of-vocabulary words, use a special **UNKNOWN** token in $V$

- because we often want to learn over minibatches and this is very inefficient when the sequences have different lengths, we:
  1) chop sequences to have a finite maximum length
  2) pad shorter sequences with a special $<$PAD$>$ token to have the maximum length, e.g.

  $<$BOS$>$ Iraq Halib Oil Exports $<$EOS$>$
  $<$BOS$>$ Belarus Shuts Borders $<$EOS$>$ $<$PAD$>$
Problem (Vanishing Gradients)

Simple ANNs work well for short sequences, but the hidden state acts as a primitive memory, so we can run into issues of the gradient vanishing or exploding for longer sequences. This is because they attempt to retain all information on the sequence prefix seen before.
Resolution (in practice, by consensus, the most popular RNN architectures)

LSTM (Long-Short Term Memories; 1997)
GRUs (Gated Recurrent Units; 2014)

Focus on LSTMs: basic intuition is to selectively choose what the network will remember or forget.

Introduce:
- cell state to represent our global memory that we can update to remember or forget, $C_t$
- forget gate - determines what we forget in the cell state at each time $t$, $f_t$
- input gate - determines what we remember in our cell state at time $t$, $i_t$
Flow for LSTM

\[ C_{t-1} \]

\[ f_t \]

\[ h_{t-1} \]

\[ i_t \]

\[ C_t \]

\[ h_t \]

\[ o_t \]

logistic sigmoid

logistic sigmoid

tanh

log sigmoid
Corresponding formulas:

\[ i_t = \text{sigmoid} \left( \mathbf{W}_{h,i} h_{t-1} + \mathbf{W}_{x,i} x_t \right) \]

\[ f_t = \text{sigmoid} \left( \mathbf{W}_{h,f} h_{t-1} + \mathbf{W}_{x,f} x_t \right) \]

\[ c_t = f_t \circ c_{t-1} + i_t \circ \tanh \left( \mathbf{W}_{h,c} h_{t-1} + \mathbf{W}_{x,c} x_t \right) \]

\[ o_t = \text{sigmoid} \left( \mathbf{W}_{h,o} h_{t-1} + \mathbf{W}_{x,o} x_t \right) \]

\[ h_t = o_t \circ \tanh \left( \mathbf{W}_{c,h} c_t \right) \]

Learn via BPTT all the parameters for these gates.
LSTMs are very popular (among RNN models). GRUs were introduced to simplify LSTMs:

- have a hidden state, no cell state
- only have two gates: reset and update gate

Faster to train, but LSTMs are more popular
Bidirectional RNN

"Very —, how are you?"

complete with well, good, etc.

The "future" can provide useful information in many tasks.

Bidirectional RNNs predict the current output \( \hat{y}_t \) using the "past" and the "future"

\[
\begin{align*}
\overrightarrow{h}_t &= f(x_t, \overrightarrow{h}_{t-1}, \overrightarrow{w}_h) \\
\overleftarrow{h}_t &= f(x_t, \overleftarrow{h}_{t+1}, \overleftarrow{w}_h) \\
h_t &= [\overrightarrow{h}_t | \overleftarrow{h}_t] \\
\hat{y}_t &= g(h_t, \omega) 
\end{align*}
\]
Deep RNNs (typically used when "LSTMs" or "RNNs" are mentioned)

- Train layer-wise, by computing all outputs from layer $l$ so we have the inputs for layer $l+1$
- Modify BPTT to account for dependencies between layers