Today: LSTMs, GRUs (vaguely)
Bidirectional RNNs
Deep RNNs
code examples: sentiment analysis for IMDB dataset

Problem (Vanishing gradients)
Simple RNN works for short sequences, but the hidden state acts as a primitive memory, so run into issues of gradient vanishing/exploding for longer sequences. This is because it attempts to retain all the information on the sequence prefix seen before.
Resolution (in practice, by consensus, the most popular architectures)

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

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

Introduce:
- cell state to augment our memory in addition to the hidden state, $C_t$ (in cell state)
- forget gate — determine what we forget at each time, $f_t$
- input gate — determine what we remember in our cell state, $i_t$
Flow diagram for LSTM
\[ i_t = \text{sigmoid}(w_{hi}h_{t-1} + w_{xi}x_t) \]
\[ f_t = \text{sigmoid}(w_{hf}h_{t-1} + w_{xf}x_t) \]
\[ c_t = f_t \odot c_{t-1} + i_t \odot \tanh(w_{hc}h_{t-1} + w_{xc}x_t) \]
\[ o_t = \text{sigmoid}(w_{ho}h_{t-1} + w_{xo}x_t) \]
\[ h_t = o_t \odot \tanh(w_{ch}c_t) \]

Learn via backprop all the parameters for these gates
LSTMs are very popular. GRUs were introduced to simplify the LSTM architecture:
- have only a hidden state, no cell state
- only two gates: reset and update gates
Faster to train, but LSTMs are more popular.
Bidirectional RNN

Very → how are you?
well → good, etc.

The future can provide useful information as well.

Standard RNN

Bidirectional RNN
Bidirectional RNNs predict the current output $y_t$ using the past and the future:

\[
\begin{align*}
\overrightarrow{h}_t &= f(x_t, h_{t-1}, \overrightarrow{\omega}_h) \\
\overleftarrow{h}_t &= f(x_t, h_{t+1}, \overleftarrow{\omega}_h) \\
\overrightarrow{h}_t &= \left[\begin{array}{c}
\overrightarrow{h}_t \\
\overleftarrow{h}_t
\end{array}\right] \\
y_t &= g(h_t, \omega_0)
\end{align*}
\]

Useful for non-causal tasks where context is useful, e.g., NLP tagging tasks (NER, POS, etc.), change point detection in time series, translation.

BTT needs to be modified to account for future dependencies.
Deep RNNs (typically used when "LSTMs" or "RNNs" are mentioned)

- train layer-wise, by computing all outputs from layer $d$ so have inputs for layer $d+1$
- modify BIT to account for dependencies between layers
Sequence-to-Sequence Modeling (2014 used for very good machine translation, popularized deep LSTM seq-to-seq model)

Sutskever et al. “Sequence to Sequence Learning with NNS”

Goal: given input sequence of tokens \((x_1, \ldots, x_T)\)

predict output sequence of tokens \((y_1, \ldots, y_{T'})\)

Canonical example: Machine Translation

Issue: \(T\) may be different than \(T'\), so the many-to-many design paradigm for RNNs may not fit

Soln: encoder-decoder architecture

\[
\text{many-to-one RNN} \quad \uparrow \quad \text{one-to-many RNN}
\]
Sequence-to-Sequence architecture

Encoder

Decoder

(teacher forcing only during training)

START

<END>
Training:

- train the encoder & decoder simultaneously (using BPTT) so that given the input sequence \((x_1, ..., x_T)\) and output sequence \((y_1, ..., y_T-1)\):
  
  - 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 propagate. Instead, use teacher forcing: use \(y_{T-1}\) (the ground truth) as the input to predict \(y_T\) at training time
As our main inputs: \((x_1, \ldots, x_T)\) and \((\langle \text{START} \rangle, y_1, \ldots, y_T, \langle \text{END} \rangle)\) and as training outputs: \((y_1, \ldots, y_T', \langle \text{END} \rangle)\).
Inference

- Given input sequence \((x_1, \ldots, x_T)\) use encoder to obtain \(h_T\).
- Initialize the decoder with hidden state \(h_T\) and feed in the \texttt{<START> token}.
- At each time, feed \(y_{t-1}\) into the decoder to predict the next token and take \(y_t\) to be the most likely token.
Details

- use deep LSTMs (GRUs) in the encoder and decoder

- reverse the order of the source sequence to introduce shorter length dependencies w/ the target sequence

May bien y to Very well and you

to y bien May Very well and you

- use "beam decoding" in practice (see Sutskever et al.)