Cyclic Computation

- Neural nets we've encountered so far have been directed acyclic graphs
Cyclic Computation

- Neural nets we've encountered so far have been directed acyclic graphs
- What happens when we add a cycle?
Cyclic Computation

Layer activations represented as vectors (x,h,y)

\[ h = \tanh(W_{ih}x) \]

\[ y = \text{softmax}(W_{ho}h) \]
Cyclic Computation

- Layer activations represented as vectors \((x,h,y)\).
- Matrix multiplications compute weighted sums of previous layer activations.

\[
\begin{align*}
\mathcal{X} & \\
h = \tanh(W_{ih}x) & \\
y = \text{softmax}(W_{ho}h)
\end{align*}
\]
Cyclic Computation

- Layer activations represented as vectors (x,h,y)
- Matrix multiplications compute weighted sums of previous layer activations
- Non-linear function (ReLU, softmax) of the weighted sum input yields activation

\[ h = \tanh(W_{ih}x) \]

\[ y = \text{softmax}(W_{ho}h) \]
Cyclic Computation

- To deal with recurrency, we need a notion of time
- Hidden layer works the same as before, but it now incorporates a weighted sum of its own previous activations
- Usually $h_0 = 0$

$\mathbf{x}_t$

$h_t = \tanh(W_{ih}x_t + W_{hh}h_{t-1})$

$y_t = \text{softmax}(W_{ho}h_t)$
Unrolling RNNs

\[
\begin{align*}
    &h_0 \xrightarrow{W_{hh}} h_1 \\
    &\quad \downarrow W_{ih} \quad \downarrow W_{ih} \quad \downarrow W_{ih} \\
    &y_1 \xrightarrow{W_{ho}} y_2 \xrightarrow{W_{ho}} y_3 \\
    &x_1 \xrightarrow{W_{ih}} h_1 \xrightarrow{W_{hh}} h_2 \xrightarrow{W_{hh}} h_3 \\
    &x_2 \xrightarrow{W_{ih}} h_2 \xrightarrow{W_{hh}} h_3 \\
    &x_3 \xrightarrow{W_{ih}} h_3
\end{align*}
\]
Just Remember

Every output depends on every previous input.
Source: The Unreasonable Effectiveness of Recurrent Neural Networks, Andrej Karpathy

http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Image Captioning

- $h_0$ initialized with output from a convolutional neural network
- Previous output word passed as next input

Source: Deep Visual-Semantic Alignments for Generating Image Descriptions
Sentiment Analysis

Diagram showing a sequence of hidden states $H_{t-1}, H_t, H_{t+1}, H_{t+2}$, each with associated words $X_{t-1}$, $X_t$, $X_{t+1}$, $X_{t+2}$, $X_{t+3}$, and output $Y$.
Source: Sequence to Sequence Learning with Neural Networks
Training RNNs

- Use BackPropagation Through Time (BPTT)
- Quite memory intensive; may have to use truncated BPTT
- Unrolled diagrams come in very handy when thinking about this
Drawbacks of RNNs

- Harder to parallelize because of their sequential nature, therefore slower than feed-forward networks
- Difficult to train, vulnerable to both vanishing gradients and exploding gradients
  - Solution: clamp the gradients
- RNNs are forgetful; often have trouble learning long-term dependencies
LSTMs and GRUs

**RNN**

**LSTM**

**GRU**

\[
i_t = \sigma(x_t U^i + h_{t-1} W^i)
\]

\[
f_t = \sigma(x_t U^f + h_{t-1} W^f)
\]

\[
o_t = \sigma(x_t U^o + h_{t-1} W^o)
\]

\[
\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)
\]

\[
C_t = \sigma(f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t)
\]

\[
h_t = \tanh(C_t) \cdot o_t
\]

\[
z_t = \sigma(x_t U^z + h_{t-1} W^z)
\]

\[
r_t = \sigma(x_t U^r + h_{t-1} W^r)
\]

\[
\tilde{h}_t = \tanh(x_t U^h + (r_t \cdot h_{t-1}) W^h)
\]

\[
h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t
\]
Attentional RNNs

- Solves the problem of a fixed size hidden state having to encode arbitrary length input sequences
- Remember all previous hidden states, predict attention weights over them when producing output sequence

Source: Neural Machine Translation by Jointly Learning to Align and Translate

Take away points

- RNNs are neural networks with internal state
- Every output is affected by every previous input
- RNNs are very versatile and well suited for sequence modeling
- To compute gradients, must unroll and backpropagate through time
- In practice, LSTM and GRU layers are normally used since they outperform vanilla RNNs