Recurrent Neural Networks
Deep learning is a popular area in AI.
State-Space Model

- $h_t$: hidden state
- $X_t$: input
- $Y_t$: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- $h_{-1}$: initial state
Recurrent Neural Network

- $h_t$: hidden state
- $X_t$: input
- $Y_t$: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- $h_{-1}$: initial state

Fully-connect NN vs. RNN
- $h_t$: a vector summarizes all past inputs (a.k.a. “memory”)
- $h_{-1}$ affects the entire dynamics (typically set to zero)
- $X_t$ affects all the outputs and states after $t$
- $Y_t$ depends on $X_0, \ldots, X_t$
Recurrent Neural Network

- $h_t$: hidden state
- $X_t$: input
- $Y_t$: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- $h_{-1}$: initial state

Fully-connect NN vs. RNN

- RNN can be viewed as repeated applying fully-connected NNs
- $h_t = \sigma_1(W^{(1)}X_t + W^{(11)}h_{t-1} + b^{(1)})$
- $Y_t = \sigma_2(W^{(2)}h_t + b^{(2)})$
- $\sigma_1, \sigma_2$ are activation functions (sigmoid, ReLU, tanh, etc)
Recurrent Neural Network

Stack \( K \) layers of fully-connected NN

- \( h_t^{(k)} \): hidden state
- \( X_t \): input
- \( Y_t \): output
- \( h_t^{(1)} = f_1^{(1)}(h_t^{(1)}_{t-1}, X_t; \theta) \)
- \( h_t^{(k)} = f_1^{(k)}(h_t^{(k)}_{t-1}, h_t^{(k-1)}; \theta) \)
- \( Y_t = f_2(h_t^{(K)}; \theta) \)
- \( h_{-1}^{(k)} \): initial states
Training Recurrent Neural Network

- $h_t$: hidden state
- $X_t$: input
- $Y_t$: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- $h_{-1}$: initial state

Data: $\{(X_t, D_t)\}_{t=1}^{T}$ (RNN can handle more general data format)

Loss $L(\theta) = \sum_{t=1}^{T} \ell(Y_t, D_t)$

Goal: learn $\theta$ by gradient-based method
  - Back propagation
Back Propagation Through Time

- \( h_t = \sigma_1(W^{(1)}X_t + W^{(11)}h_{t-1} + b^{(1)}) \)
- \( Y_t = \sigma_2(W^{(2)}h_t + b^{(2)}) \)
- \( Z_t^{(1)}: \) pre-activation of hidden state
  \( (h_t = \sigma_1(Z_t^{(1)})) \)
- \( Z_t^{(2)}: \) pre-activation of output
  \( (Y_t = \sigma_2(Z_t^{(2)})) \)
Back Propagation Through Time
Back Propagation Through Time
Extensions

What if $Y_t$ depends on the entire inputs?

- Biredirectional RNN:
  - AN RNN for forward dependencies: $t=0,\ldots,T$
  - An RNN for backward dependencies: $t=T,\ldots,0$
  - $Y_t = f_2(h^f_t, h^b_t; \theta)$
Extensions

RNN for sequence classification (sentiment analysis)

- $Y = \max_{t} Y_t$
- Cross-entropy loss
Practical issues of RNN

Linear RNN derivation
Practical issues of RNN: training

Gradient explosion and gradient vanishing
Techniques for avoiding gradient explosion

- Gradient clipping
- Identity initialization
- Truncated backprop through time
  - Only backprop for a few steps
Preserve Long-Term Memory

- Difficult for RNN to preserve long-term memory
  - The hidden state $h_t$ is constantly being written (short-term memory)
  - Use a separate cell to maintain long-term memory
Long Short-Term Memory Network

LSTM (Hochreiter & Schmidhuber, ’97)
- RNN architecture for learning long-term dependencies
- $\sigma$: layer with sigmoid activation
Long Short-Term Memory Network

LSTM (Hochreiter & Schmidhuber, ’97)
• Core idea: maintain separate state $h_t$ and cell $c_t$ (memory)
• $h_t$: full update every step
• $c_t$: only *partially* update through gates
  • $\sigma$ layer outputs importance ($[0,1]$) for each entry and only modify those entries of $c_t$
Long Short-Term Memory Network

Forget gate $f_t$

- $f_t$ outputs whether we want to “forget” things in $c_t$
  - Compute $c_{t-1} \odot f_t$ (element-wise)
  - $f_t(i) \rightarrow 0$: want to forget $c_t(i)$
  - $f_t(i) \rightarrow 1$: we want to keep the information in $c_t(i)$

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$
Long Short-Term Memory Network

Input gate $i_t$
- $i_t$ extracts useful information from $X_t$ to update memory
  - $\tilde{c}_t$: information from $X_t$ to update memory
  - $i_t$: which dimension in the memory should be updated by $X_t$
    - $i_t(j) \rightarrow 1$: we want to use the information in $\tilde{c}_t(j)$ to update memory
    - $i_t(t) \rightarrow 0$: $\tilde{c}_t(j)$ should not contribute to memory

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$
Long Short-Term Memory Network

Memory update

- $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
- $f_t$ forget gate; $i_t$ input data
- $f_t \odot c_{t-1}$: drop useless information in old memory
- $i_t \odot \tilde{c}_t$: add selected new information from current input

\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \]
Long Short-Term Memory Network

Output gate $o_t$

- Next hidden state $h_t = o_t \odot \tanh(c_t)$
  - $\tanh(c_t)$: non-linear transformation over all past information
  - $o_t$: choose important dimensions for the next state
    - $o_t(j) \rightarrow 1$: $\tanh(c_t(j))$ is important for the next state
    - $o_t(j) \rightarrow 0$: $\tanh(c_t(j))$ is not important

\[
\begin{align*}
o_t &= \sigma(W_o \ [h_{t-1}, x_t] + b_o) \\
h_t &= o_t \ast \tanh(C_t)
\end{align*}
\]
Long Short-Term Memory Network

\[ h_t = o_t \odot \tanh(c_t) \]
\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \]
\[ Y_t = g(h_t) \]

Remarks:
1. No more matrix multiplications for \( c_t \)
2. LSTM does not have guarantees for gradient explosion/vanishing
3. LSTM is the dominant architecture for sequence modeling from ’13 - ’16.
4. Why tanh
LSTM Variant

Peephold Connections (Gers & Schmidhuber ’00)
• Allow gates to take in $c_t$ information

$$f_t = \sigma (W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma (W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma (W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$
LSTM Variant

Simplified LSTM
- Assume $i_t = 1 - f_t$
- Only two gates are needed: fewer parameters

\[
C_t = f_t \ast C_{t-1} + (1 - f_t) \ast \tilde{C}_t
\]
LSTM Variant

Gated Recurrent Unit (GRU, Cho et al. ’14)

• Merge $h_t$ and $c_t$: much fewer parameters

\[
\begin{align*}
    z_t &= \sigma (W_z \cdot [h_{t-1}, x_t]) \\
    r_t &= \sigma (W_r \cdot [h_{t-1}, x_t]) \\
    \tilde{h}_t &= \tanh (W \cdot [r_t \ast h_{t-1}, x_t]) \\
    h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]
LSTM application: language model

- Autoregressive language model: $P(X; \theta) = \prod_{t=1}^{L} P(X_t \mid X_{i<t}; \theta)$
  - $X$: a sentence
  - Sequential generation
- LSTM language model
  - $X_t$: word at position $t$.
  - $Y_t$: softmax over all words
- Data: a collection of texts:
  - Wiki
LSTM application: text classification

Bi-directional LSTM and them run softmax on the final hidden state.