LONG SHORT TERM MEMORY

LSTM NETWORKS
Simple RNN – recurrent neural network

Recurrent connection

Output

Input

Loop. State
PLAIN OLD NN

activation(dot(W, input_t) + b)

RNN

activation(dot(W, input_t) + dot(U, state_t) + b)
RUN

\[
\begin{align*}
\text{y}_1 & \rightarrow \text{R} & \rightarrow \text{W} \\
\text{y}_2 & \rightarrow \text{R} & \rightarrow \text{W} \\
\text{y}_3 & \rightarrow \text{R} & \rightarrow \text{W} \\
\text{y}_4 & \rightarrow \text{R} & \rightarrow \text{W} \\
\text{y}_5 & \rightarrow \text{R} & \rightarrow \text{W}
\end{align*}
\]
RNN

MAIN PURPOSE

Remember what is has seen so far (state)

Capture Long Distance Dependencies
MAIN PURPOSE

Remember what is has seen so far (state)

Capture Long Distance Dependencies

Not so good
TURN OUT ...  

Simple RNNs are too simple to deal with long-distance dependencies

A RNN variant, LSTM work much better.
Almost all the applications of RNN use LSTM

- Speech recognition
- Machine translation
- Image captioning
- ...


May look complicated

“Not so complicated—merely complex.” - Chollet
CHOLLET’S EXPLANATION

Time

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \]

States:
- State 2
- State 3
- State 4
- State 5

Outputs:
- \( y_1 \)
- \( y_2 \)
- \( y_3 \)
- \( y_4 \)
- \( y_5 \)
### Plain Vanilla RNN

- **Output t-1**
  
  \[ \text{Output t-1} = \text{activation} \left( W_o \times \text{input}_t - 1 + U_o \times \text{state}_t - 1 + b_o \right) \]

- **Output t**
  
  \[ \text{Output t} = \text{activation} \left( W_o \times \text{input}_t + U_o \times \text{state}_t + b_o \right) \]

- **Output t+1**
  
  \[ \text{Output t+1} = \text{activation} \left( W_o \times \text{input}_t + 1 + U_o \times \text{state}_t + 1 + b_o \right) \]
**LSTM**

Output $t-1 = \text{activation}(Wo \cdot \text{input}_{t-1} + Uo \cdot \text{state}_{t-1} + Vo \cdot c_{t-1} + bo)$

Output $t = \text{activation}(Wo \cdot \text{input}_t + Uo \cdot \text{state}_t + Vo \cdot c_t + bo)$

Output $t+1 = \text{activation}(Wo \cdot \text{input}_{t+1} + Uo \cdot \text{state}_{t+1} + Vo \cdot c_{t+1} + bo)$

Time:
- $x\ t-1$
- $x\ t$
- $x\ t+1$
LSTM

Output t-1 → Output t → Output t+1

Carry track: information flows → add new info → forget selected info

Output t = activation(Wo * input_t + Uo * state_t + Vo * c_t + bo)
Compute new carry based on previous carry

Output $t-1$

Output $t$

Output $t+1$

State $t$

State $t+1$

Time $x t-1$

$x t$

$x t+1$

$\text{compute new carry}$

$\text{compute new carry}$

$\text{Carry track}$

Output $t = \text{activation}(W_o \cdot \text{input}_t + U_o \cdot \text{state}_t + V_o \cdot c_t + b_o)$
As well as current input

Output $t-1$

State $t$

Output $t$

Output $t+1$

Output $t = \text{activation}(W_0 \ast \text{input}_t + U_0 \ast \text{state}_t + V_0 \ast c_t + b_0)$

Carry track

Compute new carry

State $t+1$

Time $x t-1$

$\vdots$

Time $x t$

$\vdots$

Time $x t+1$
And the current state

Output $t-1$  

State $t$  

Output $t$  

State $t+1$  

Output $t+1$  

Output $t = \text{activation}(W_o \ast \text{input}_t + U_o \ast \text{state}_t + V_o \ast c_t + b_o)$  

Carry track  

Compute new carry  

Compute new carry  

And the current state
THAT’S THE Mid-level description
A CLOSER LOOK

Pseudocode
output_t = activation(dot(state_t, Uo) +
                dot(input_t, Wo) +
                dot(c_t, Vo) + bo)
output_t = activation(dot(state_t, Uo) +
dot(input_t, Wo) +
dot(c_t, Vo) + bo)

dot([1, 2, 3, 4, 5], [10, 20, 30, 40, 50])
   = 1 * 10 + 2 * 20 + 3 * 30 + 4 * 40 + 5 * 50
   = 10 + 40 + 90 + 160 + 250
   = 590
Forget gate layer -> what information are we going to throw out

\[ f_t = \text{activation}(\text{dot}(\text{state}_t, U_f) + \text{dot}(\text{input}_t, W_f) + b_f) \]
Input gate layer -> contributes to the information are we going to keep

\[ i_t = \text{activation}(\text{dot}(\text{state}_t, U_i) + \text{dot}(\text{input}_t, W_i) + b_i) \]
K-gate - Keep gate

\[ k_t = \text{activation}(\text{dot}(\text{state}_t, U_k) + \text{dot}(\text{input}_t, W_k) + b_k) \]
Compute new carry

\[ c_{t+1} = i_t \cdot k_t + c_t \cdot f_t \]
In 2009, deep multidimensional LSTM networks demonstrated the power of deep learning with many nonlinear layers, by winning three ICDAR 2009 competitions in connected handwriting recognition, without any prior knowledge about the three different languages to be learned.

A Novel Connectionist System for Improved Unconstrained Handwriting Recognition
The fire brigade has arrived.
Adenauer is in a tough spot. Waiting.
B ring support and comfort to
Commonwealth countries do
Socrates was a Classical Greek philosopher. Credited as one of the founders of Western philosophy, he is an enigmatic figure known only through the classical accounts of his students. Plato's dialogues are the most comprehensive accounts of Socrates to survive from antiquity. Forging an accurate picture of the historical Socrates and his philosophical viewpoints is problematic at best. This issue is known as the Socratic problem. The knowledge of the man, his life, and his philosophy is based on writings by the students and contemporaries. Foremost among them is Plato, however, works by Xenophon, Aristotle, and Aristophanes also provide important insights. The difficulty of finding the real Socrates arises because these works are often philosophical or dramatic texts rather than straightforward histories. Aside from Thucydides who makes no mention of Socrates or philosophers in general, there is in fact no such thing as a straightforward history contemporary with Socrates that dealt with his own time and place.

contemporaries. Foremost among works by Xenophon, Aristotle, and Aristophanes provide important insights. The real Socrates arises because these works are often philosophical or dramatic. Thucydides, who makes no mention of
**CANONICAL SEQ2SEQ LEARNING PHASE**

- Encoder RNN layer(s)
  - processes the input sequence and returns its own internal state.
  - We discard the outputs of the encoder RNN, only recovering the state.

- Decoder RNN
  - trained to predict the next characters of the target sequence, given previous characters of the target sequence.
  - uses as initial state the state vectors from the encoder, which is how the decoder obtains information about what it is supposed to generate.
INFERENCE PHASE

- Run source language sentence through encoder to get state
- Feed state and start symbol (for ex., <S>) to decoder to get first character/word.
- Repeat until you generate </S> (end of sequence symbol)
North Korea is firing up reactor

Norður-Kóreu er að hleypa upp reactor
**INFERENCCE PHASE**

"The weather is nice"  
LSTM encoder  
...  
Internal LSTM states (h, c)  
"Il fait beau[STOP]"

"[START]Il fait beau"  
LSTM decoder
output_t = activation(dot(state_t, Uo) +
    dot(input_t, Wo) +
    dot(c_t, Vo) + bo)
state_t+1 = dot(output_t, activation(c_t))
**APPLIED DEEP LEARNING IS A VERY EMPIRICAL PROCESS**

From Andrew Ng

- Build first model quickly
- Use train/validation/test set
- Evaluate and adjust parameters

Idea

Experiment

Code
Quickly iterate over your ML project following these strategic guidelines

- Build your first model quickly
- Define a single optimizing metric
- Define constraints and satisficing metrics

- Use a 98/1/1% train/dev/test set distribution
- Dev/test sets have the same data distribution
- Dev/test sets contain only examples to optimize for

- Evaluate errors to estimate bias, variance and data mismatch
- Manually analyze and label 100 misclassified examples
- Consider end-to-end, transfer or multitask learning
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)

c_{t+1} = i_t * k_t + c_t * f_t
output_t = activation(dot(state_t, Uo) +
    dot(input_t, Wo) +
    dot(c_t, Vo) + bo)

dot([1, 2, 3, 4, 5], [10, 20, 30, 40, 50])
    = 1 * 10 + 2 * 20 + 3 * 30 + 4 * 40 + 5 * 50
    = 10 + 40 + 90 + 160 + 250
    = 590