Review: RNN

Recurrent Neural Networks are networks in the form of a directed cyclic graph.

They pass previous state information from previous computations to the next.

They can be used to process sequence data with relatively low model complexity when compared to feed forward models.
RNN

*Goal* of RNNs: remember information from the past
“The dog that my family had when I was a child had a fluffy ____.”
RNN

“The dog that my family had when I was a child had a fluffy ____.”

Want: “tail”
RNN Weaknesses

But… RNNs are not very good at remembering things *far* in the past.
RNN Weaknesses

“The dog that my family had when I was a child had a fluffy _____.

• To predict “tail” RNN needs to remember the subject of the sentence - “dog”
RNN Weaknesses

“The dog that my family had when I was a child had a fluffy _____.”

- To predict “tail” RNN needs to remember the subject of the sentence - “dog”
- “dog” and predicted word are separated by 12 words
  - On the outer limit of what a vanilla RNN would be able to remember.
Review: RNN update rule

\[ s_t = \rho((e_t, s_{t-1})W_r + b_r) \]

\[ o_t = \sigma(s_t W_o + b_o) \]
An Illustrative Example:
An Illustrative Example:

What happens to the information about “dog” as we continue through the network?
An Illustrative Example:
An Illustrative Example:

Can imagine that the information about “dog” is stored in some part of the RNN’s hidden state vector
An Illustrative Example:

Through all subsequent RNN steps, we want “dog” to stay the same

RNN steps
An Illustrative Example:

If we think of “dog” as just a few entries in the vector…
An Illustrative Example:

...to preserve “dog”, we need to compute the identity function over the part of the vector that stores it.

But will that happen? No

Why?
How does this affect the hidden state?

RNN update

\[ h_t = \rho((e_t, h_{t-1})W_r + b_r) \]

The hidden state goes through a fully connected layer!
How does this affect the hidden state?

• What will happen to our dog after we multiply our weights by our hidden state?
How does this affect the hidden state?

• What will happen to our dog after we multiply our weights by our hidden state?

\[ \text{Weights} \times \text{Hidden State} = \text{New Hidden State} \]

\[ \cdots + w_{i-1,j} \cdot h_{i-1} + w_{i,j} \cdot \text{dog} + w_{i+1,j} \cdot h_{i+1} + \cdots \]
How does this affect the hidden state?

Dog gets lost in all the other information!

Weights × Hidden State = New Hidden State

\[ \ldots + w_{i-1,j} \cdot h_{i-1} + w_{i,j} \cdot \text{dog} + w_{i+1,j} \cdot h_{i+1} + \ldots \]
How does this affect the hidden state?

• “dog” in hidden state gets combined and mixed with rest of hidden state

   RNN steps

   RNN forgets about the dog after a certain time 😞
RNNs cannot learn “long term” dependency

We need new way to update hidden state!

How?
An analogy to human (or computer) memory:

- RNN hidden state $\rightarrow$ “short term memory/RAM”
  - Like how you lose contents of RAM if you shut down a computer...
  - ...or how human short-term memory fades after time

\[ h_0 \rightarrow \text{RNN Cell} \rightarrow h_1 \rightarrow \text{RNN Cell} \rightarrow h_2 \rightarrow \ldots \rightarrow h_{t-1} \rightarrow \text{RNN Cell} \rightarrow h_t \]

The dog that long...?

fluffy
An analogy to human (or computer) memory:

• RNN hidden state → “short term memory/RAM”
  • Like how you lose contents of RAM if you shut down a computer...
  • ...or how human short-term memory fades after time

• What we want → “long term memory/disk”
  • Some state representing knowledge that persists
  • Like how contents of disk persist across shut-downs...
  • ...or how sleep consolidates human memory into long-term memory

• **Long** Short Term Memory (LSTM)
  • “Short-term memory that persists over time”
  • i.e. “hidden states that remember information for longer”
What is different?

Vanilla RNN

RNN Cell

$\mathbf{h}_{t-1}$ → $\mathbf{h}_t$ → $\mathbf{x}_t$

LSTM

LSTM Cell

$\mathbf{c}_{t-1}$ → $\mathbf{h}_t$ → $\mathbf{c}_t$ → $\mathbf{h}_t$
LSTM

Cell State (long short-term memory)

Hidden State (short term memory)

word embedding
How an LSTM works

• An LSTM consists of 3 major modules:
  • Forget module
  • Remember module
  • Output module
The Complete LSTM

Forget module

Remember module

Output module

\[ h_t \]

\[ c_{t-1} \]

\[ h_{t-1} \]

\[ x_t \]

\[ w b \sigma \]

\[ w b \sigma \]

\[ w b h \]

\[ w b \sigma \]
Forget Module

Say we just predicted “tail” in “My dog has a fluffy _____.”

Next set of words: “I love my dog”
Forget Module

- Model no longer needs to know about “dog”
- Ready to delete information about subject
Forget Module

- $c_{t-1}$
- $h_{t-1}$
- $x_t$
- $w b \sigma$
- $h_t$
Forget Module

• Filters out what gets allowed into the LSTM cell from the last state
  • Example: If it’s remembering gender pronouns, and a new subject is seen, it will forget the old gender pronouns

• Either lets parts of $C_{t-1}$ pass through or not

“Maria loves to dance. She is a graceful dancer. Simon loves to sing”
Forgetting information

- Use pointwise multiplication by a **mask vector** to forget information
  - What do we want to forget from last cell state?

My dog has a fluffy tail. I love my dog
Forgetting information

• Use pointwise multiplication by a **mask vector** to forget information
  • What do we want to forget from last cell state?
  • Output of fully connected + sigmoid is what we want to forget

My dog has a fluffy tail. I love my dog
Forgetting information

- Use pointwise multiplication by a **mask vector** to forget information
  - What do we want to forget from last cell state?
  - Output of fully connected + sigmoid is what we want to forget
  - “Zeros out” a part of the cell state
  - Pointwise multiplication by a learned mask vector is known as **gating**

\[
\begin{align*}
\text{C}_{t-1} & \quad \sigma(W[x_t \ h_{t-1}] + b) & \quad \text{Unforgotten } \text{C}_{t-1} \\
0.3 & \quad 0.0 & \quad 0.0 \\
0.2 & \quad 0.0 & \quad 0.0 \\
\end{align*}
\]
Forget Module

$w b \sigma$ = fully connected layer with sigmoid

$\otimes$ = pointwise multiplication
What’s next?

$w, b, \sigma$ = fully connected layer with sigmoid

$\otimes$ = pointwise multiplication
Remember Module

• We can save information that we want to remember by adding it into “empty” slots in the cell state
Remember Module

• First: use gating to decide what to remember
Gating for ‘selective memory’

• A fully-connected + tanh on [input, memory] computes some new memory

\[
tanh(W_1[x_t \ h_{t-1}] + b_1)
\]
Gating for ‘selective memory’

- A fully-connected + tanh on [input, memory] computes some new memory
- We gate this memory to decide what bits of it we want to remember long-term in the cell state

\[ \text{tanh}(W_1[x_t \ h_{t-1}] + b_1) \quad \text{and} \quad \sigma(W_2[x_t \ h_{t-1}] + b_2) \]
Gating for ‘selective memory’

• A fully-connected + tanh on [input, memory] computes some new memory

• We gate this memory to decide what bits of it we want to remember long-term in the cell state

\[
\begin{align*}
tanh(W_1[x_t \ h_{t-1}] + b_1) & \quad \sigma(W_2[x_t \ h_{t-1}] + b_2) \\
\text{Selected Memory}
\end{align*}
\]
Remember Module

\[ h_t = \text{fully connected layer with sigmoid} \]
\[ w b i = \text{fully connected layer with tanh} \]
\[ \times = \text{pointwise multiplication} \]
\[ \oplus = \text{pointwise addition} \]
Remember Module

• Then: we add this selective memory into the cell state
Remembering information

- Add what we didn’t forget to what we did remember
Why does this solve our problem?

- Cell state never goes through a fully connected layer!
  - Never has to mix up its own information
Output Module

- $w b \sigma$ = fully connected layer with sigmoid
- $w b h$ = fully connected layer with tanh
- $\times$ = pointwise multiplication
- $+$ = pointwise addition
Output Module

• Same structure as the remember module
  • Provides path for short-term memory $h_t$ to temporarily acquire info from the longer-term cell state.
The Complete LSTM

\[ i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \]
\[ f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \]
\[ o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \]
\[ \tilde{c}_t = \tanh(W h_{t-1} + U x_t + b) \]
\[ c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \]
\[ h_t = o_t \odot \tanh(c_t) \]
\[ y_t = h_t \]
The Complete LSTM

• If we never output cell state then why do we have it?
• Is it possible to just have one state, the hidden state?
GRU

- **Gated Recurrent Unit**
- In practice, similar performance and may train faster
  - Removes cell state, computationally more efficient and less complex
- In theory, weaker than LSTMs since it **cannot unboundedly count**
  - Counting: track increment or decrement of variable
  - e.g. Validate brackets in code
    
    ```
    [ ... ( ... { ... } ... ) ... ]
    ```
    Requires counting brackets & nesting levels
Can you guess which part is forget module, which one is remember module, and which one is output module in the GRU?
GRU vs LSTM

Forget module

\[ h_t \]

\[ h_{t-1} \]

\[ x_t \]

\[ c_{t-1} \]

\[ h_t \]

\[ c_t \]

\[ h_{t-1} \]

\[ x_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]

\[ h_t \]
GRU vs LSTM

Remember module
GRU vs LSTM

No direct analogue in LSTM
“Select the part of the memory that is relevant for the current prediction step”
Recap

Limitations of RNNs

- RNN cell state \(\rightarrow\) Hidden state
- Hidden state gets modified over time
- Cannot capture long-term dependency

LSTMs

- Cell state + Hidden state
- 3 modules
- GRU vs LSTM