Recurrent Neural Network (CNN)
Gated RNNs

M. R. Hasan
Deep Learning
Readings

• Geron: 15
• Zhang: 9
• Chollet: 6
What We Will Cover

• Introduction to Gated RNNs for preventing the vanishing gradient problem
• Gated Recurrent Unit (GRU)
• Long Short-Term Memory (LSTM)
RNN: The Vanishing Gradient Problem

- The absence of long-term dependency in RNNs is a direct consequence of the vanishing gradient problem.
- We will discuss two key techniques to resolve the vanishing gradient problem.
  - Weight initialization & activation
  - Gated RNNs
Gated RNNs to Prevent Vanishing Gradients
Gated RNNs

• We will discuss an effective strategy to resolve the vanishing gradient problem in RNNs.
• Previously we discussed why long-term dependencies are impossible to learn by simple RNNs.
• This is due to the vanishing gradient problem.
• An effective approach to resolve this problem is to modify the architecture of the RNN.

First, let’s discuss the main problem.
Gated RNNs

• In this example, the RNN needs to retain the state of "Republic" to infer "Plato"
• The states of the other input are not very useful for the inference.
• Ideally we want information ($H_{t-1} \& \frac{\partial L}{\partial W}$) to flow to and from the input state “Republic” without any loss.
Gated RNNs

• Thus, a RNN needs to retain the memory of the long-range states.
• Due to the vanishing gradient problem, the long-range states are less updated.
• As a consequence, RNNs are unable to retain long-term memory making inferences poor for very long sequences.
Gated RNNs

- The goal is to ensure the information flow ($H_{t-1}$ & $\frac{\partial \mathcal{L}}{\partial W}$) to and from some **selective hidden states**.
Gated RNNs

• This can be done by installing **controller gates** inside the RNN cell.
Gated RNNs

• The gate controls **how much of the current state** (short-term memory $X_t$) to pass to the next state as compared to the value of **previous state** (long-term memory $H_{t-1}$).

![Diagram of Gated RNNs](image)
Gated RNNs

- We discuss **two effective gated RNN cells** for retaining long-range dependency.
  - Gated Recurrent Unit (GRU) **2014**
  - Long Short-term Memory (LSTM) **1997**
- Although the GRU cell was proposed much later we discuss it first due to its design simplicity.
Gated Recurrent Unit (GRU)
Gated Recurrent Unit

• Ideally we expect a RNN to store information of a state ($H_{t-1}$) that will be needed in future computation (by a future state).
• This state information should be **retained for as long** it is needed.

Once the information (of a past state) is used by a state (future state), i.e., the past state’s utility is fulfilled, then it should be **reset**.
Gated Recurrent Unit

- We also expect that there are states that **don’t carry pertinent information** (e.g., states between Republic & Plato).
- We don’t want to use information of a current state \( (X_t) \) of this type to update the previous state (that may carry information from a distant past state).

\[
H_t = \tanh(X_t W_{xh} + H_{t-1} W_{hh} + \vec{b}_h)
\]
Gated Recurrent Unit

• Thus, we want the RNN to **ignore or reduce** the importance of some current states $X_t$ (e.g., states between *Republic* & *Plato*).
• The backpropagation algorithm should be able to grow the importance of some states (e.g., “*Republic*”) while lower importance of other states (e.g., states between *Republic* & *Plato*).

$$H_t = \tanh(X_t W_{xh} + H_{t-1} W_{hh} + \tilde{b}_h)$$
Gated Recurrent Unit

• These **two expectations** are summarized as follows.
  - Remembering (selective past states) $H_{t-1}$
  - Forgetting (selective current states) $X_t$

We need to modify the architecture of the **vanilla RNN** cell such that these two expectations are met.
Gated Recurrent Unit

- GRU modifies the vanilla RNN in two ways:
  - Create a channel to **flow the past memory** or hidden state $H_{t-1}$ for retaining long-term memory (Update)
  - Create a channel to **reduce (reset)** the influence of past and put more emphasis on the current observation (Reset)

**Vanilla RNN**

$H_t = tanh(X_tW_x + H_{t-1}W_h + b_h)$

**GRU**

$H_t = tanh(X_tW_x + H_{t-1}W_h + \tilde{b}_h)$

$\tilde{H}_t$ = Reset $H_{t-1}$ or Short-term memory

$H_t$ = Update $H_{t-1}$ or Long-term memory
Gated Recurrent Unit

• The first channel “remembers” the past.
• The second channel “forgets” the past.
• First, we will discuss how to “forget” the past.
GRU: Control (reset) Past & Encourage Short-Term Memory

- Let’s see how a past state $H_{t-1}$ influences the current state $H_t$ in a vanilla RNN.
- We use a mini-batch input $X_t$ with $m$ samples and $d$ features.

$$H_t = \tanh(X_tW_{xh} + H_{t-1}W_{hh} + \vec{b}_h)$$
GRU: Control (reset) Past & Encourage Short-Term Memory

• For nonlinear activation the $tanh$ is used in this illustration.

\[ H_t = tanh(X_tW_{xh} + H_{t-1}W_{hh} + b_h) \]
GRU: Control (reset) Past & Encourage Short-Term Memory

- Now say that we want to control the past state $H_{t-1}$.
- We could connect a logic gate with it such that the gate is either open (outputs value 1) or close (outputs value 0).
- Let’s call it a reset gate and denote it by $R_t$.

Vanilla RNN

$$H_t = tanh(X_tW_{xh} + H_{t-1}W_{hh} + \tilde{b}_h)$$
GRU: Control (reset) Past & Encourage Short-Term Memory

• Using this gate, we can **assert our control on the past** and create a **new current state** \( \tilde{H}_t \).

• We will decide later whether we will use the updated current state as the final one.

So, let’s call it the candidate current state and denote it by \( \tilde{H}_t \)

\[
\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \otimes H_{t-1}) W_{hh} + \tilde{b}_h)
\]
GRU: Control (reset) Past & Encourage Short-Term Memory

- But how do we **design the reset gate**?
- Since its output ranges between 0 and 1, we can design it using a fully-connected (FC) neural network with **sigmoid activation**.
- Sigmoid squashes the output signal between the range 0 and 1.

\[ R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \]

The reset gate will use the **current input** and the **past hidden state** to determine its output (whether to produce a value closer to 0 or 1).
GRU: Control (reset) Past & Encourage Short-Term Memory

- Observe that the output of the reset gate is **multiplied** (element-wise since it’s a matrix) with the past state $H_{t-1}$.
- If the reset gate output is 0, then the past state is **completely ignored** for computing the current candidate state $\tilde{H}_t$.

Thus, now we know how to “forget” the past.

Let’s see how to “**remember**” the past.
GRU: Remember (Update) Past & Encourage Long-Term Memory

- To remember the past, we need another logic gate.
- It will determine whether to retain the past state $H_{t-1}$ or the current candidate state $\tilde{H}_t$. 

\[
R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)
\]

\[
Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)
\]
GRU: Remember (Update) Past & Encourage Long-Term Memory

• In other words, this gate will **update the current state** $H_t$ by either retaining the past state $H_{t-1}$ or using only the current observation $\tilde{H}_t$.

• We call this gate as the **update gate** and denote it with $Z_t$.

$$H_t = Z_t \otimes H_{t-1} + (1 - Z_t) \otimes \tilde{H}_t$$
GRU: Remember (Update) Past & Encourage Long-Term Memory

- The update gate outputs values between 0 (close) and 1 (open).
- For example, if $Z_t = 0$, the GRU uses the candidate current state to update its state (i.e., forgets the past).

On the other hand, if $Z_t = 1$, then GRU only retains the past.
GRU: Remember (Update) Past & Encourage Long-Term Memory

- The update gate is designed using a FC network with \textit{sigmoid activation}:

\[
Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + \vec{b}_z)
\]
GRU: Remember (Update) Past & Encourage Long-Term Memory

- We see that GRU enables two things.
  - Capture short-term dependencies via the reset gate
  - Capture long-term dependencies via the update gate
GRU Equations

\[ R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \]

\[ \tilde{H}_t = \text{tanh}(X_t W_{xh} + (R_t \otimes H_{t-1}) W_{hh} + b_h) \]

\[ Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z) \]

\[ H_t = Z_t \otimes H_{t-1} + (1 - Z_t) \otimes \tilde{H}_t \]
Long Short-Term Memory (LSTM)
• We will build the LSTM architecture from our understanding of the GRU architecture.
Long Short-Term Memory (LSTM)

• Let’s compare the two sets of equations from these two architectures.

![Diagram showing LSTM and GRU equations]

**GRU Equations**

- $h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes \tilde{h}_t$
- $\tilde{c}_t = \text{tanh}(x_t w_x + h_{t-1} w_h + b)$
- $c_t = f_t \otimes c_{t-1} + i_t \otimes \tilde{c}_t$
- $o_t = \sigma(x_t w_o + h_{t-1} w_o + b)$
- $h_t = o_t \otimes \text{tanh}(c_t)$

**LSTM Equations**

- $r_t = \sigma(x_t W_r + h_{t-1} W_r + b_r)$
- $\tilde{h}_t = \text{tanh}(x_t W_h + (r_t \otimes h_{t-1}) W_h + b_h)$
- $i_t = \sigma(x_t W_i + h_{t-1} W_i + b_i)$
- $f_t = \sigma(x_t W_f + h_{t-1} W_f + b_f)$
- $c_t = f_t \otimes c_{t-1} + i_t \otimes \tilde{c}_t$
- $h_t = c_t \otimes \text{tanh}(c_t)$
- LSTM state is split into two vectors: hidden-state $H_t$ and memory state or cell state $C_t$.

- Here the $C_t$ could be considered as the long-term state, while the $H_t$ is the short-term state.
• **A key difference** between LSTM and GRU is that in LSTM the memory state (similar to the hidden state in GRU) is **updated using two gates** (in GRU we used only one gate): forget gate & input gate.

---

**GRU Equations**

\[
\begin{align*}
R_t &= \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \\
\tilde{h}_t &= \tanh(X_t W_{xh} + (R_t \otimes H_{t-1}) W_{hh} + b_h) \\
Z_t &= \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z) \\
H_t &= Z_t \otimes H_{t-1} + (1 - Z_t) \otimes \tilde{h}_t \\
I_t &= \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \\
\tilde{c}_t &= \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c) \\
F_t &= \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \\
c_t &= F_t \otimes c_{t-1} + I_t \otimes \tilde{c}_t \\
o_t &= \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \\
O_t &= \tanh(c_t) \\
H_t &= O_t \otimes \tanh(c_t)
\end{align*}
\]

**LSTM Equations**

\[
R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \\
\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \otimes H_{t-1}) W_{hh} + b_h) \\
Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z) \\
\tilde{h}_t = Z_t \otimes H_{t-1} + (1 - Z_t) \otimes \tilde{H}_t \\
I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \\
\tilde{c}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c) \\
F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \\
c_t = F_t \otimes c_{t-1} + I_t \otimes \tilde{c}_t \\
o_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \\nH_t = O_t \otimes \tanh(c_t)
\]
• A **key difference** between LSTM and GRU is that in LSTM the memory state (similar to the hidden state in GRU) is **updated using two gates** (in GRU we used only one gate): forget gate & input gate.
- LSTM processes **two types of state information**.
  - Past memory state $C_{t-1}$ (contains *long-term* memory)
  - Current (candidate) memory state $\tilde{C}_t$ (contains *short-term* memory)
- LSTM processes **two types of state information**.
  - Past memory state $C_{t-1}$ (contains *long-term* memory)
  - Current (candidate) memory state $\tilde{C}_t$ (contains *short-term* memory)
• Then, it creates the **updated memory state** by determining whether to **forget** the past memory state (long-term memory) and **retain** the candidate memory state (short-term memory).

\[ C_t = F_t \otimes C_{t-1} + I_t \otimes \tilde{C}_t \]
Thus, to begin with, it needs at least **two gates**:
- **Forget gate**: to control the long-term memory
- **Input gate**: to control the current state information (short-term memory)

\[ C_t = F_t \otimes C_{t-1} + I_t \otimes \tilde{C}_t \]
• It, then, uses an **output gate** to determine how much of the updated memory state should be passed to the next hidden state.

• Let’s discuss these **three gates**.

\[
H_t = O_t \otimes \tanh(C_{t-1})
\]
LSTM: Forget Gate – Controls Long-Term Memory

- The **forget gate** controls the flow of the long-term memory from the memory state.
- It is learned by a **FC network with sigmoid** using the past hidden state and the current input (outputs values between 0 and 1).

\[
F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)
\]

- Controls how much of long-term memory to be retained

\[
F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)
\]
LSTM: Input Gate – Controls Short-Term Memory

- The **input gate** is used to determine how much of the current state information to be retained.
- Thus, first we compute a **candidate memory state** using the past hidden state and the current input.

\[
\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + \tilde{b}_c)
\]
LSTM: Input Gate – Controls Short-Term Memory

- The $\tilde{C}_t$ can be considered as a state that contains only short-term information as it is based on the past hidden-state (which is a short-term state) and the current input state.

$$\tilde{C}_t = \tanh(X_tW_{xc} + H_{t-1}W_{hc} + \tilde{b}_c)$$
LSTM: Input Gate – Controls Short-Term Memory

- This short-term state $\tilde{c}_t$ is controlled by the input gate.
- Input gate $I_t$ is learned by a FC network with sigmoid using the past hidden state and the current input (outputs values $0 \sim 1$).

$$I_t = \sigma(X_tW_{xi} + H_{t-1}W_{hi} + b_i)$$
LSTM: Input Gate – Controls Short-Term Memory

- Both the forget gate and input gate are used to determine whether to retain long-term or short memory in the updated memory state.

\[ C_t = F_t \otimes C_{t-1} + I_t \otimes \tilde{C}_t \]

![Diagram of LSTM](image)

- Controls how much of long-term memory to be retained
- Controls how much of short-term memory to be retained
- Memory State
- Forget Gate
- Input Gate
- Hidden State
- Short-term state
- Long-term or short-term memory?
LSTM: Input Gate – Controls Short-Term Memory

- If $F_t = 1$ and $I_t = 0$: Long-term memory would be preserved dropping short-term memory.
- If $F_t = 0$ and $I_t = 1$: Long-term memory would be discarded, only short-term memory is retained.
LSTM: Input Gate – Controls Short-Term Memory

- Observe that if an **input is important**, LSTM preserves it by the input gate.
- For an important input, the input gate learns a **non-zero value**.
- Then, the important input is **added to the long-term memory**.

Thus, the input gate helps to **preserve an important input**.
LSTM: Input Gate – Controls Short-Term Memory

• The long-term memory is passed from one timestep to the next through a long sequence without losing its memory as long as the forget gate is not set to 0.

• Whenever the LSTM decides not to retain a long-term memory, it can “reset” it by using a 0 value of the forget gate.
LSTM: Long-term vs Short-term Memory

- In this example, we want to **retain** the state of “Republic”, and **ignore** the subsequent states up to “Plato”.
- Thus, we need to retain the memory state for “Republic” $C_2$ and discard input states $X_t$ from $t=3$ to $t=6$. 

![Diagram](image-url)
The long-term memory of “Republic” is retained by setting forget gate to 1, and input gate for the subsequent states to 0.

Once info about “Republic” is used at \( t=7 \), then it is “reset” by the forget gate using the value 0.
LSTM: Update Hidden State by the Output Gate

- Finally, the **output gate** controls which parts of the long-term state should be read and output at this timestep, both to $H_t$ and to $Y_t$.

$$H_t = O_t \odot \tanh(C_t)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$
LSTM: Update Hidden State by the Output Gate

- Just like the input and forget gates, the output gate is learned by a **FC network with sigmoid** using the past hidden state and the current input.

\[
O_t = \sigma(x_tW_{xo} + h_{t-1}W_{ho} + b_o)
\]
LSTM: Update Hidden State by the Output Gate

- Output gate = 1: the LSTM effectively passes all memory information through to the predictor.
- Output gate = 0: it retains all the information only within the memory cell and perform no further processing.

\[
O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)
\]

\[
H_t = O_t \odot \tanh(C_t)
\]
• In short, an LSTM cell can learn to:
  - Recognize an important **input** (that’s the role of the input gate)
  - Store it in the **long-term state**
  - Preserve it for as long as it is needed (that’s the role of the forget gate)
  - Extract it whenever it is needed.

This explains why these cells have been amazingly successful at capturing long-term patterns in time series, long texts, audio recordings, and more.
Next: Gated RNNs for Preventing the Vanishing Gradient Problem