CPSC 532P / LING 530A: Deep Learning for Natural Language Processing (DL-NLP)

Muhammad Abdul-Mageed

muhammad.mageed@ubc.ca

Natural Language Processing Lab

The University of British Columbia

Table of Contents

Vanishing and Exploding Gradients

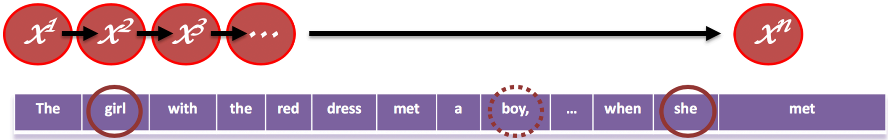

Figure: A very long sequence, modelled with an RNN. Gradients can vanish and we will not know which gender a pronoun should be (male or female, e.g., in an MT task), or to which entity the pronoun refers (boy or girl)

Gradient Problems

- Gradients can **explode**, in which case we can clip them.
- Gadients can also **vanish**, which is a more serious problem.

Solving Long-Term Dependencies

Solutions For Gradient Problems

- Long-Short Term Memory (**LSTM**) networks introduced to solve the problem of long-term dependencies
- Gated Recurrent Units (**GRU**) (Cho et al., 2014; Chung et al., 2014): Simplification of LSTMs.
- We will **introduce GRUs first**, as they are simpler
- Some notation for LSTM modified from Andrew Ng, for pedagogical simplicity

Introducing a Memory Cell

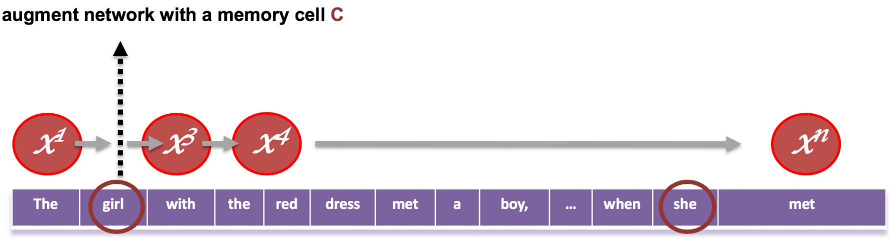

Figure: We will augment the network with a memory cell

Gradient Problems

- The **memory cell** will help us retain information over long sequences
- For example, we can still know **we need pronoun she**, maintaining the **female gender** (and retaining the correct **reference** to "the girl")

Simple GRU

1: Simple GRU Cell

$$z_t = \sigma(W_z.[h_{t-1}, x_t])$$

$$\tilde{h}_t = tanh(Wx.[h_{t-1}, x_t])$$

$$h_t = z_t * \tilde{h}_t + (1 - z_t) * h_{t-1}$$

GRU Cells Notation Translation Table

- z: Update gate
- $\tilde{h}$: New candidate memory
- h_t: GRU output

Relevance Gate in GRU

2: GRU Cell

$$z_t = \sigma(W_z.[h_{t-1}, x_t])$$

$$r_t = \sigma(W_r.[h_{t-1}, x_t])$$

$$\tilde{h}_t = tanh(Wx.[r_t * h_{t-1}, x_t])$$
$$h_t = z_t * \tilde{h}_t + (1 - z_t) * h_{t-1}$$

GRU Cells Notation Translation Table

- **r**: Relevance state *(reset gate)*

Bias Added to GRU

3: GRU With Bias

$$z_t = \sigma(W_z.[h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r.[h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = tanh(Wx.[r_t * h_{t-1}, x_t] + b_h)$$

$$h_t = z_t * \tilde{h}_t + (1 - z_t) * h_{t-1}$$

GRU Cells Notation Translation Table

- **z**: Update gate
- **r**: Relevance state *(reset gate)*
- $\tilde{h}$: New candidate memory
- h_t: GRU activation (output)

GRU Update Rule

4: Simple GRU Update

$$h_t = z_t * \tilde{h}_t + (1 - z_t) * h_{t-1}$$

GRU Gates

- **z**: Update gate is result of a a sigmoid (between 0 and 1)
- **z close to zero**: We multiply by $\sim$ zero, so we update candidate $\tilde{h}$ very little (almost keep value of old memory cell h_t)
- **z close to zero**: We multiply by $\sim$ 1 and subtract by $\sim$ 1, so old cell becomes almost equal to candidate)

How is LSTM Different?

5: Recall: GRU

$$z_t = \sigma(W_z.[h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r.[h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = tanh(Wx.[r_t * h_{t-1}, x_t] + b_h)$$

$$h_t = z_t * \tilde{h}_t + (1 - z_t) * h_{t-1}$$

$$a_t = h_t$$

Note

- Cell activation a_t is the same as h_t.
- At each step, we start with $h_t = a_t$. **They are different in LSTM.**

Changes

- Parts in red **will change!**
- a_t $!=$ h_t and so we will use a_t
- We will also **add new parts**: **forget gate** and **output gate** ...

6: Recall: GRU

$$z_t = \sigma(W_z.[h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r.[h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = tanh(Wx.[r_t * h_{t-1}, x_t] + b_h)$$

$$h_t = z_t * \tilde{h}_t + (1 - z_t) * h_{t-1}$$

$$a_t = h_t$$

Alternative Notation (Simpler)

GRU Cells Notation Translation Table

- **z**: Update state $\rightarrow \Gamma_u$
- **r**: Relevance state *(reset gate)* $\rightarrow \Gamma_r$
- $\tilde{h}$: New candidate memory cell state $\rightarrow \tilde{C}_t$
- h_t: GRU output $\rightarrow C_t$

Updating $\tilde{h}_t$

- To acquire $\tilde{h}_t$, instead of h_{t-1}, we use the new a_{t-1} (since a_{t-1} is acquired differently than h_{t-1}, as we explain later)

7: New Candidate $\tilde{C}_t$

$$\tilde{h}_t = tanh(W_x.[a_{t-1}, x_t])$$

LSTM: New Candidate Cell $\tilde{C}_t$ (New Notation)

Updating $\tilde{C}_t$

- To acquire $\tilde{C}_t$, instead of C_{t-1}, we use the new a_{t-1} (since a_{t-1} is acquired differently than C_{t-1}, as we explain later)

8: New Candidate $\tilde{C}_t$

$$\tilde{C}_t = tanh(W_c.[a_{t-1}, x_t])$$

LSTM: Update and Forget Gates

Changes

- We will not use an **relevance gate** (Γ_r)
- Instead of using one update gate Γ_u, we will use **two gates** to control cell content: Γ_u **(update gate)**, sometimes called ***input gate*** f_i, and Γ_f **(forget gate)**
- **Forget gate** will give the new memory cell C_t the option to keep or forget the **old cell** (C_{t-1}), but just add to it via **update gate** (Γ_u)

9: LSTM

$$\Gamma_u = \sigma(W_u.[a_{t-1}, x_t])$$

$$\Gamma_f = \sigma(W_f.[a_{t-1}, x_t])$$

$$C_t = \Gamma_u * \tilde{C}_t + \Gamma_f * C_{t-1}$$

Output Gate

- As mentioned, we will use an **output gate (Γ_o)**
- **Output gate** will enable us to update our a_t via element-wise multiplication by Γ_o

10: Output Gate

$$\Gamma_o = \sigma(W_o.[a_{t-1}, x_t])$$

$$a_t = \Gamma_o * tanh(C_t)$$

LSTM: Putting it All Together

11: LSTM

$$\tilde{C}_t = tanh(W_c.[a_{t-1}, x_t])$$

$$\Gamma_u = \sigma(W_u.[a_{t-1}, x_t])$$

$$\Gamma_f = \sigma(W_f.[a_{t-1}, x_t])$$

$$\Gamma_o = \sigma(W_o.[a_{t-1}, x_t])$$

$$C_t = \Gamma_u * \tilde{C}_t + \Gamma_f * C_{t-1}$$

$$a_t = \Gamma_o * tanh(C_t)$$

LSTM: Putting it All Together, With Bias

12: LSTM

$$\tilde{C}_t = tanh(W_c.[a_{t-1}, x_t] + b_c)$$

$$\Gamma_u = \sigma(W_u.[a_{t-1}, x_t] + b_u)$$

$$\Gamma_f = \sigma(W_f.[a_{t-1}, x_t] + b_f)$$

$$\Gamma_o = \sigma(W_o.[a_{t-1}, x_t] + b_o)$$

$$C_t = \Gamma_u * \tilde{C}_t + \Gamma_f * C_{t-1}$$

$$a_t = \Gamma_o * tanh(C_t)$$

$$\tilde{C}_t = tanh(W_c.[a_{t-1}, x_t])$$
$$\Gamma_u = \sigma(W_u.[a_{t-1}, x_t])$$
$$\Gamma_f = \sigma(W_f.[a_{t-1}, x_t])$$
$$\Gamma_o = \sigma(W_o.[a_{t-1}, x_t])$$
$$C_t = \Gamma_u * \tilde{C}_t + \Gamma_f * C_{t-1}$$
$$a_t = \Gamma_o * tanh(C_t)$$

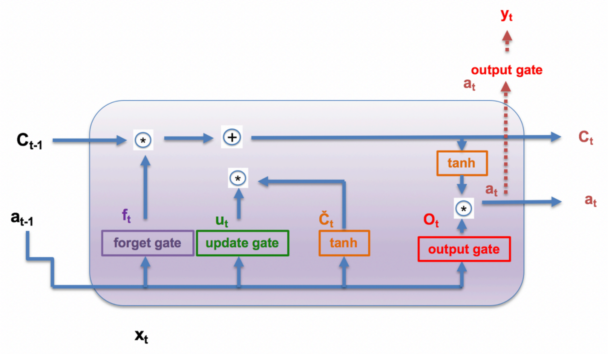

Figure: LSTM cell. [Inspired by Chris Olah]

Stacking LSTM Cells

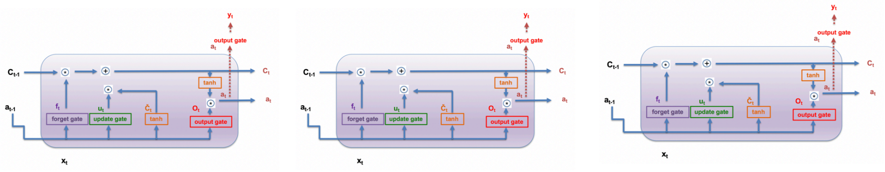

Figure: LSTM cell.s stacked. Note: Each cell will need new-indexing (not shown in the Figure)