Recurrent Neural Networks

ML Instruction Team, Fall 2022

CE Department
Sharif University of Technology

Backpropagation through time (BPTT)

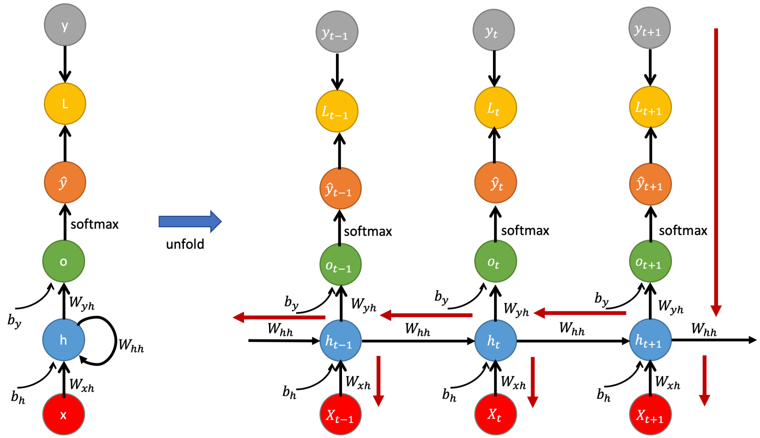

Figure: Simple RNN Computational Graph, source

Backpropagation through time (BPTT)

- Suppose that in this example, we have

$$h_t = tanh(X_t.W_{xh} + h_{t-1}.W_{hh} + b_h)$$

$$o_t = h_t.W_{yh} + b_y$$

$$y_t = Softmax(o_t)$$

- And our loss function is Log Loss. So as you remember, we have

$$L(y, \hat{y}) = \sum_{t=1}^{T} L_t(y_t, \hat{y}_t) = -\sum_{t=1}^{T} y_t \, log \, \hat{y}_t = -\sum_{t=1}^{T} y_t \, log \, [Softmax(o_t))]$$

Backpropagation through time (BPTT)

- Now, we are going to calculate derivation of L w.r.t $W_{yh}, W_{hh}, W_{xh}, b_y, b_h$
  - Part I: The Straight Ones

$$\frac{\partial L}{\partial W_{yh}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W_{yh}}$$

$$= \sum_{t=1}^{T} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial W_{yh}}$$

$$= \sum_{t=1}^{T} (\hat{y}_t - y_t) \otimes h_t$$

$$\frac{\partial L}{\partial b_y} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial b_y} = \sum_{t=1}^{T} (\hat{y}_t - y_t)$$

Backpropagation through time (BPTT)

- Cont.
  - Part II: The Tricky Ones

$$\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial W_{hh}}$$

we know that h_t is a function of h_{t-1} and W_{hh}, h_{t-1} itself is a function of W_{hh} and h_{t-2}, and so on. Thus, we have

$$\frac{\partial h_t}{\partial W_{hh}} = \left(\frac{\partial h_t}{\partial W_{hh}}\right)_{h_{t-1}} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_{hh}}$$

$$\frac{\partial h_{t-1}}{\partial W_{hh}} = \left(\frac{\partial h_{t-1}}{\partial W_{hh}}\right)_{h_{t-2}} + \frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial W_{hh}}$$

In conclusion, the following equation holds (by substitution)

$$\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\left(\sum_{k=1}^{t}\left(\prod_{j=k}^{t-1}\frac{\partial h_{j+1}}{\partial h_j}\right)\left(\frac{\partial h_k}{\partial W_{hh}}\right)_{h_{k-1}}\right)$$

Backpropagation through time (BPTT)

- Cont. It's also true that

$$\prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j} = \frac{\partial h_t}{\partial h_k}$$

That leads us to

$$\frac{\partial L_t}{\partial W_{hh}} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \left(\frac{\partial h_k}{\partial W_{hh}} \right)_{h_{k-1}}$$

and

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \left(\frac{\partial h_k}{\partial W_{hh}} \right)_{h_{k-1}}$$

Backpropagation through time (BPTT)

- Cont. Similar to the mentioned way, we could compute derivative of L w.r.t W_{xh}, b_h

$$\frac{\partial L}{\partial W_{xh}} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \left(\frac{\partial h_k}{\partial W_{xh}} \right)_{h_{k-1}}$$

$$\frac{\partial L}{\partial b_h} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \left(\frac{\partial h_k}{\partial b_h} \right)_{h_{k-1}}$$

Backpropagation through time (BPTT)

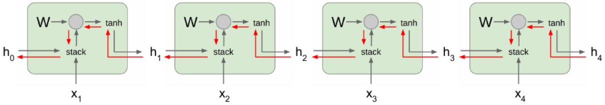

Figure: Vanilla RNN Gradient Flow, source

- RNN Training Issues
  - Computing gradient of h_0 involves many factors of W (and repeated $tanh$)
    What can we do to solve the problem? TBPTT
  - Vanishing & Exploding gradients
    What can we do to solve exploding? gradient clipping
    What can we do to solve vanishing? changing the structure

Truncated Backpropagation through time (TBPTT)

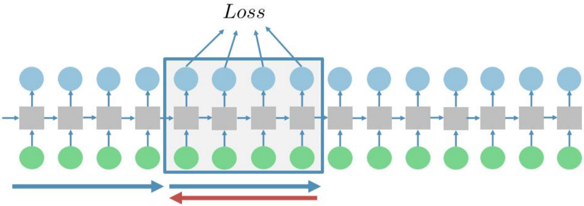

Figure: TBPTT for $k_1 = k_2 = 4$, source

- TBPTT Pseudo Code
  1. for t from 1 to T do
  2. Run the RNN one step
  3. if t divides k_1 then
  4. Run BPTT from t down to $t - k_2$
  5. end if
  6. end for

RNN Unit

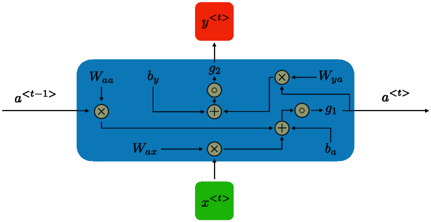

Figure: RNN Unit, source

- For a simple RNN unit we had:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\mathrm{y}^{<t>} = g_2(W_{ga}a^{<t>} + b_y)$$

Gated Recurrent Unit (GRU)

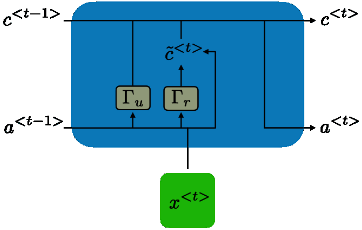

Figure: GRU Unit, source

$$\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_r = \sigma(W_r[a^{<t-1>}, x^{<t>}] + b_r)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$c^{<t>} = a^{<t>}$$

Long Short-Term Memory (LSTM)

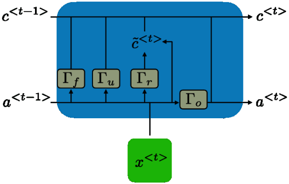

Figure: LSTM Unit, source

$$\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_r = \sigma(W_r[a^{<t-1>}, x^{<t>}] + b_r)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (\Gamma_f) * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * c^{<t>}$$

Type of Gates

- Update Gate Γ_u
  - How much past should matter
- Relevance Gate Γ_r
  - Drop previous information
- Forget gate Γ_f
  - Erase a cell or not
- Output gate Γ_o
  - How much to reveal of a cell

LSTM Walk Through

- The first step in our LSTM is to decide what information we're going to throw away from the cell state.

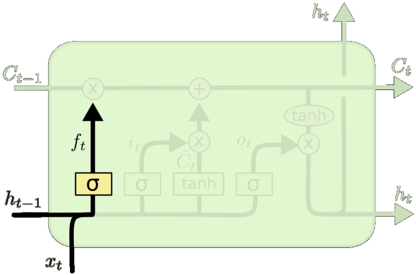

Figure: Forget Gate Layer, source

LSTM Walk Through

- The next step is to decide what new information we're going to store in the cell state.

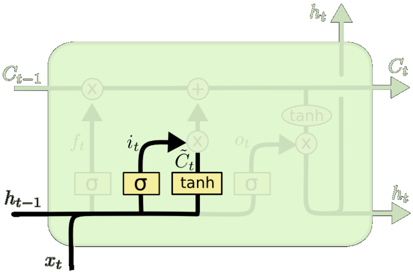

Figure: Input Gate Layer, source

LSTM Walk Through

- It's now time to update the old cell state, Ct−1, into the new cell state Ct.

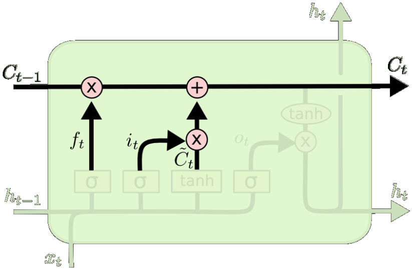

Figure: Update Cell State, source

LSTM Walk Through

- Finally, we need to decide what we're going to output. This output will be based on our cell state

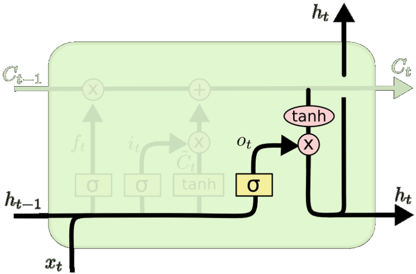

Figure: Output Gate Layer, source

Why LSTMs?

- The LSTM does have the ability to remove and add information to the cell state.
- The gates in the previous slide let the information to pass through the units.
- The gates value are between zero and one and specify how much information should be let through.
- They also somehow solve the vanishing gradient.
- We can use more blocks of them so there will be more information to remember.

How LSTMs solve vanishing gradients

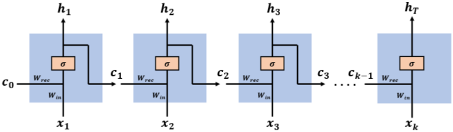

Figure: Simple Recurrent Neural Network, source

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_k}{\partial W} = \frac{\partial L_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \dots \frac{\partial c_2}{\partial c_1} \frac{\partial c_1}{\partial W} = \frac{\partial L_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \left(\prod_{t=2}^{k} \frac{\partial c_t}{\partial c_{t-1}}\right) \frac{\partial c_1}{\partial W}$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'(W_{rec}.c_{t-1} + W_{in}.x_t) W_{rec}$$

How LSTMs solve vanishing gradients

$$\frac{\partial L_k}{\partial W} = \frac{\partial L_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} (\prod_{t=2}^{k} \sigma'(W_{rec}.c_{t-1} + W_{in}.x_t) W_{rec}) \frac{\partial c_1}{\partial W}$$

- For lagre K the gradient tends to vanish or if W_{rec} is large enough it cause exploding gradient which is solved by *Gradient Clipping*.

How LSTMs solve vanishing gradients

- In LSTM we also have

$$\frac{\partial L_k}{\partial W} = \frac{\partial L_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}(\prod_{t=2}^{k}\frac{\partial c_t}{\partial c_{t-1}})\frac{\partial c_1}{\partial W}$$

- But

$$c^t = \Gamma_u * \tilde{c}^t + (\Gamma_f) * c^{t-1}$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial \Gamma_f}{\partial c_{t-1}}.c_{t-1} + \Gamma_f + \frac{\partial \Gamma_u}{\partial c_{t-1}}.\tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}}.\Gamma_u$$

- Consider that the *forget gate* is added to other terms and allows better control of gradient values. But it doesn't guarantee that there is no vanishing or exploding in gradient.

Bidirectional RNN

- How to get information from future?

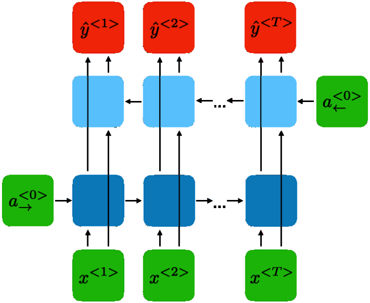

Figure: BRNN, source

$$y^{<t>} = g(W_y[\vec{a}^{<t>},\ \overleftarrow{a}^{<T-t>}] + b_y)$$

Bidirectional RNN

- They are usually used in natural language processing.
- They are powerful for modeling dependencies between words and phrases in both directions of the sequence because every component of an input sequence has information from both the past and present.

Back-propagation in BRNN

- It is exactly the same as simple RNN

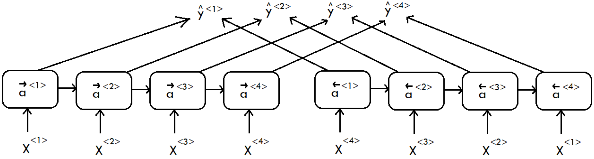

Figure: BRNN, source

Teacher Forcing

- Teacher forcing is a method for training recurrent neural networks more efficiently.
- Teacher forcing works by using the actual output at the current time step $y^{(t-1)}$ as input in the next time step, rather than the $o^{(t-1)}$generated by the network.

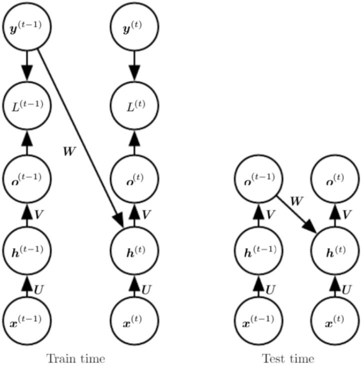

Figure: Teacher Forcing, source

Teacher Forcing

- The problem with RNNs is that we need previous time step output as input for next time step.
- This technique allows us to prevent backpropagation through time which was complex and time-consuming.
- With teacher forcing the model will be trained faster.
- During inference, since there is usually no ground truth available, the RNN model will need to feed its own previous prediction back to itself for the next prediction.

An application: Automatic Speech Recognition

- The problem with RNNs is that we need previous time step output as input for next time step.
- This technique allows us to prevent backpropagation through time which was complex and time-consuming.
- With teacher forcing the model will be trained faster.
- During inference, since there is usually no ground truth available, the RNN model will need to feed its own previous prediction back to itself for the next prediction.

Thank You!

Any Question?