Vanishing Gradients and Fancy RNNs

Shubham Gupta

February 5, 2020

1 Introduction

- Learn about problems with RNN
- More RNN variants

2 Vanishing gradient problem

- Occurs in RNN
- Small gradient in each step reduces the overall gradient signal as it back-propogates further.
- Why is it a problem?
  - Gradient signal from faraway is lost because it's much smaller than gradient signal from closeby.
  - Model weights are only updated with respect to *near effects*, not long term effects.
  - *TLDR*: Model will not learn the parameters well and hence will have weak predictablity.
  - Gradient is the effect of the past on the future.
  - If it doesnt learn the parameters, then either
    * No dependency at t and t+1
    * Or it learns wrong parameters to capture true dep between t and $t + 1$
  - Syntactic recency: Pays attention to syntax of sentence i.e longer language dependency
  - Sequential recency: Pays attention to things that only happen recently
  - Due to vanishing gradient problem, RNN learns sequency recency more.

3 Exploding gradients

- Gradient too big $\implies$ SGD update too big $\theta^{new} = \theta^{old} - \alpha\delta_\theta J(\theta$
- Solution: **Gradient clipping**
- If norm of gradient is above threshold, normalize gradient before applying SGD update
- Normalize gradient by setting max and min thresholds. This will prevent gradient from chaging drastically, thereby avoiding exploding gradiesnts problem.

3.1 Fix vanishing gradients problem

- Seperate memory for longer dependencies
- Solution: **LSTM**
- At step t, there is hidden state h^t and cell state c^t
- Can erase, read and write cell state
- Gates control whether they will write, read, etc.
- Gates are also vectors
- Gates are dynamic. Diff on each step t.
- Gates are as follows:
  - *Forget gate*: $\sigma(W_f h^{t-1} + U_f x^t + b_f)$
  - *Input gate*: $\sigma(W_i h^{t-1} + U_f x^t + b_i)$
  - *Output gate*: $\sigma(W_o h^{t-1} + U_f x^t + b_o)$
- New cell content: $c^t = tanh(W_c h^{t-1} + U_f x^t + b_c)$
- Forget some info using the forget gate
- Hidden state read output from some cell
- Solves vanishing gradient problem
  - Preserves info over many timesteps. If forget gate is set to remember everything on every t, the info in the cell is preserved indefinitely.
  - Harder for RNN to do the same

4 Gated Recurrent Units

- Keep strengths of LSTM but remove complexity
- Input x^t and hidden state h^t (no cell state)
- Update gate and reset gate

$$u^t = \sigma(W_u h^{t-1} + U_u x^t + b_u) r^t = \sigma(W_r h^{t-1} + U_r x^t + b_r) \tag{1}$$

- How does it solve vanishing gradient?
  - Easier to retain long term info
  - If u^t is 0, then h^t is kept the same at every step.

5 LSTM vs GRU

- LSTM and GRU most widely used
- GRU is *quicker to compute*
- No other pros/cons
- LSTM is **good default choice**
- Start with LSTM, switch to GRU for faster training

6 Gradient problems

- Occurs in almost all deep networks
- Solution: Add some direct connections in the network
- Example: Residual connections or skip-connections
- Makes deep networks easier to train
- **DenseNet**: Directly connect everything to everything
- **HighwayNet**: Identity connection vs transformation layer is controlled by a dynamic gate

7 Bidirectional RNN

- Take information from L-R AND R-L
- Forward and backward RNN. Concatenate both outputs.
- Train both RNN together
- Can be used **only if we have the entire input sequence**
- cant be used to do language modelling

8 Multi layer rnn

- RNN are already deep on one dim(unroll timesteps)
- Apply multiple RNN
- Compute more complex representations
- Also called **Stacked RNN**
- Perform well
- Not as deep as normal cnn. Expensive to compute

9 Conclusion

- LSTM are powerful but GRU is faster
- Clip your gradients
- Use bidirectionality when possible
- Multi layer RNN are powerful. Use it with skip connections and lots of compute lol