Fancy Recurrent Neural Networks

Richard Socher

Material from cs224d.stanford.edu

Recap of most important concepts

Word2Vec $$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

Glove $$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

Nnet & Max-margin $$s = U^T f(Wx + b)$$

$$J = \max(0, 1 - s + s_c)$$

Recap of most important concepts

Multilayer Nnet

&

Backprop

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$

$$\boxed{\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)}),}$$

$$\boxed{\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)}}$$

Recap of most important concepts

Recurrent Neural Networks

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right) \\ \hat{y}_t &= \mathrm{softmax}\left(W^{(S)}h_t\right) \end{aligned}$$

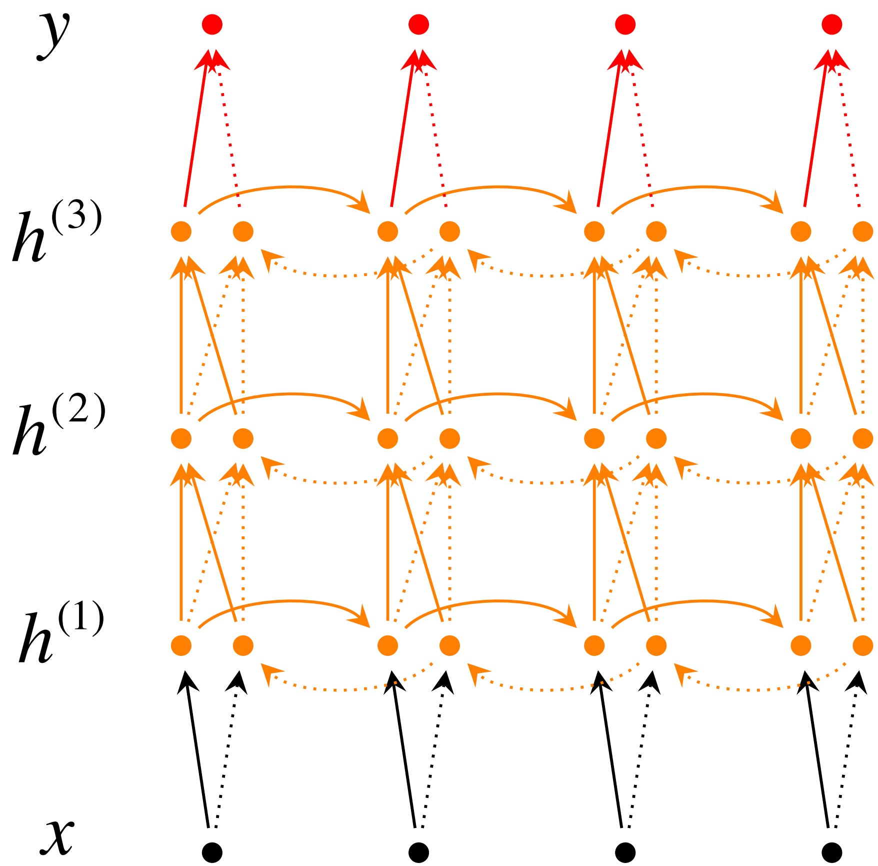
Cross Entropy Error

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

Mini-batched SGD

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_{t:t+B}(\theta)$$

Deep Bidirectional RNNs by Irsoy and Cardie



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Each memory layer passes an intermediate sequential representation to the next.

Better Recurrent Units

- More complex hidden unit computation in recurrence!
- Gated Recurrent Units (GRU)
introduced by Cho et al. 2014 (see reading list)
- Main ideas:
  - keep around memories to capture long distance dependencies
  - allow error messages to flow at different strengths depending on the inputs

GRUs

- Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$
- GRU first computes an update **gate** (another layer) based on current input word vector and hidden state

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

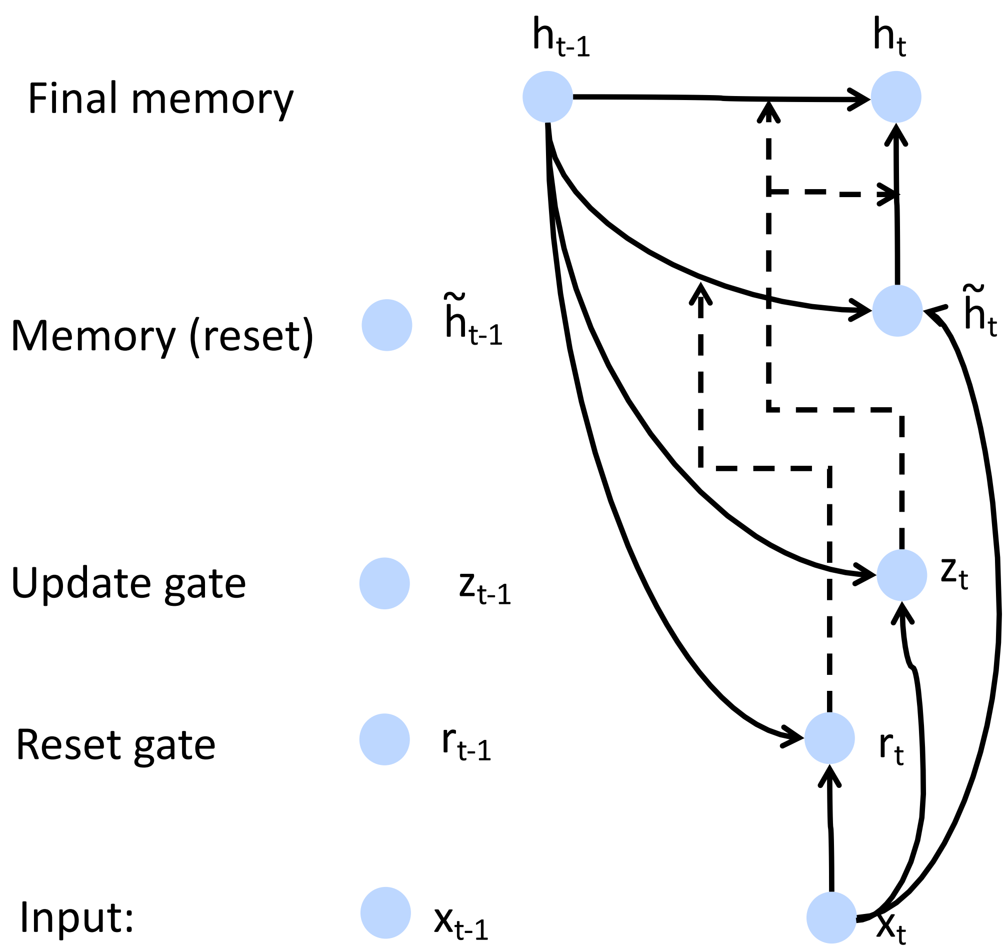
- Compute reset gate similarly but with different weights

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

GRUs

- Update gate $$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
- Reset gate $$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
- New memory content: $\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$
  If reset gate unit is ~0, then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

Attempt at a clean illustration



$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$

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

GRU intuition

- If reset is close to 0, ignore previous hidden state → Allows model to drop information that is irrelevant in the future

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

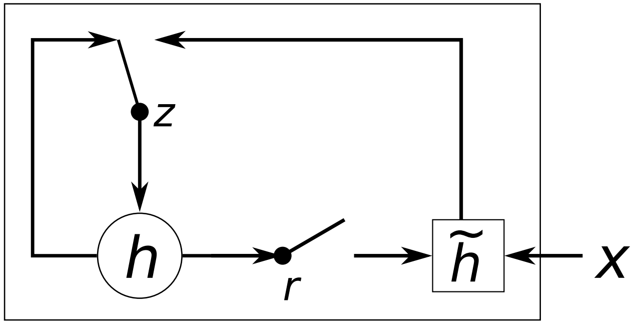
- Update gate z controls how much of past state should matter now.
  - If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient**!
- Units with short-term dependencies often have reset gates very active

GRU intuition

- Units with long term dependencies have active update gates z

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

- Illustration:

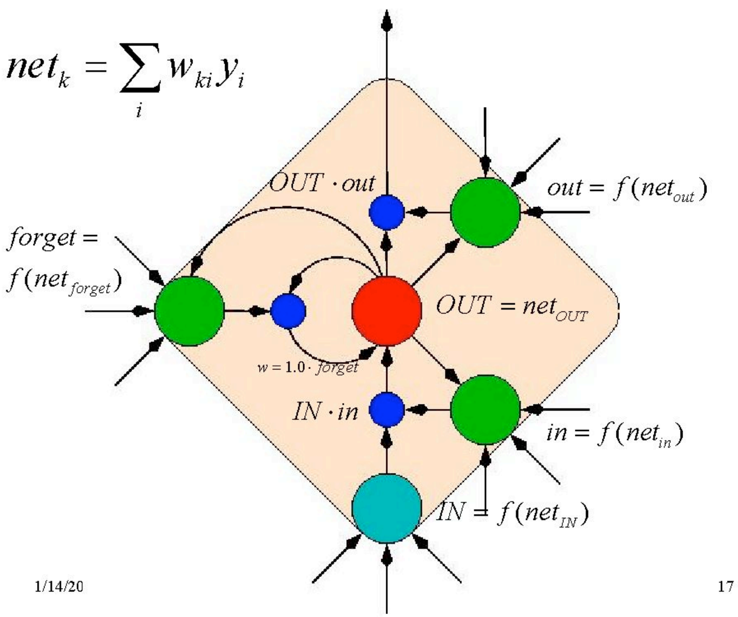


- Derivative of $\frac{\partial}{\partial x_1} x_1 x_2$? → rest is same chain rule, but implement with **modularization** or automatic differentiation

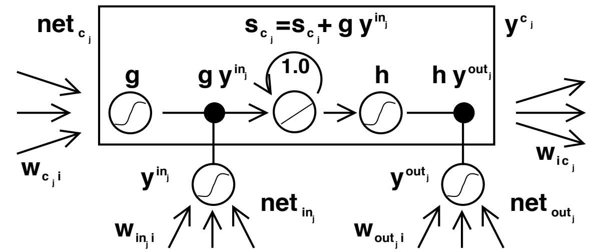
Long-short-term-memories (LSTMs)

- We can make the units even more complex
- Allow each time step to modify
  - Input gate (current cell matters) $i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$
  - Forget (gate 0, forget past) $f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$
  - Output (how much cell is exposed) $o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$
  - New memory cell $\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$
- Final memory cell: $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
- Final hidden state: $h_t = o_t \circ \tanh(c_t)$

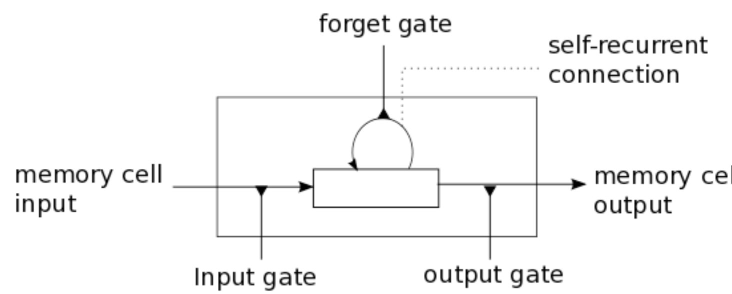
Illustrations all a bit overwhelming ;)



http://people.idsia.ch/~juergen/lstm/sld017.htm



Long Short-Term Memory by Hochreiter and Schmidhuber (1997)



http://deeplearning.net/tutorial/lstm.html

Intuition: memory cells can keep information intact, unless inputs makes them forget it or overwrite it with new input.
Cell can decide to output this information or just store it

LSTMs are currently very hip!

- En vogue default model for most sequence labeling tasks

- Very powerful, especially when stacked and made even deeper (each hidden layer is already computed by a deep internal network)

- Most useful if you have lots and lots of data

Summary

- Recurrent Neural Networks are powerful
- A lot of ongoing work right now
- Gated Recurrent Units even better
- LSTMs maybe even better (jury still out)
- This was an advanced lecture → gain intuition, encourage exploration
- Next up: Recursive Neural Networks simpler and also powerful :)