Natural Language Processing with Deep Learning

CS224N/Ling284

Lecture 9:
Vanishing Gradients
and Fancy RNNs (LSTMs and GRUs)

Richard Socher

Announcements

- Assignment 2: due Thursday

- Project proposal: due Thursday

- Midterm logistics: Fill out form on Piazza if you can’t do main midterm, have special requirements, or other special case
  - Alternate midterm is *this Friday*!
  - Practice midterms are on the website
  - Midterm review session: in-class this Thursday

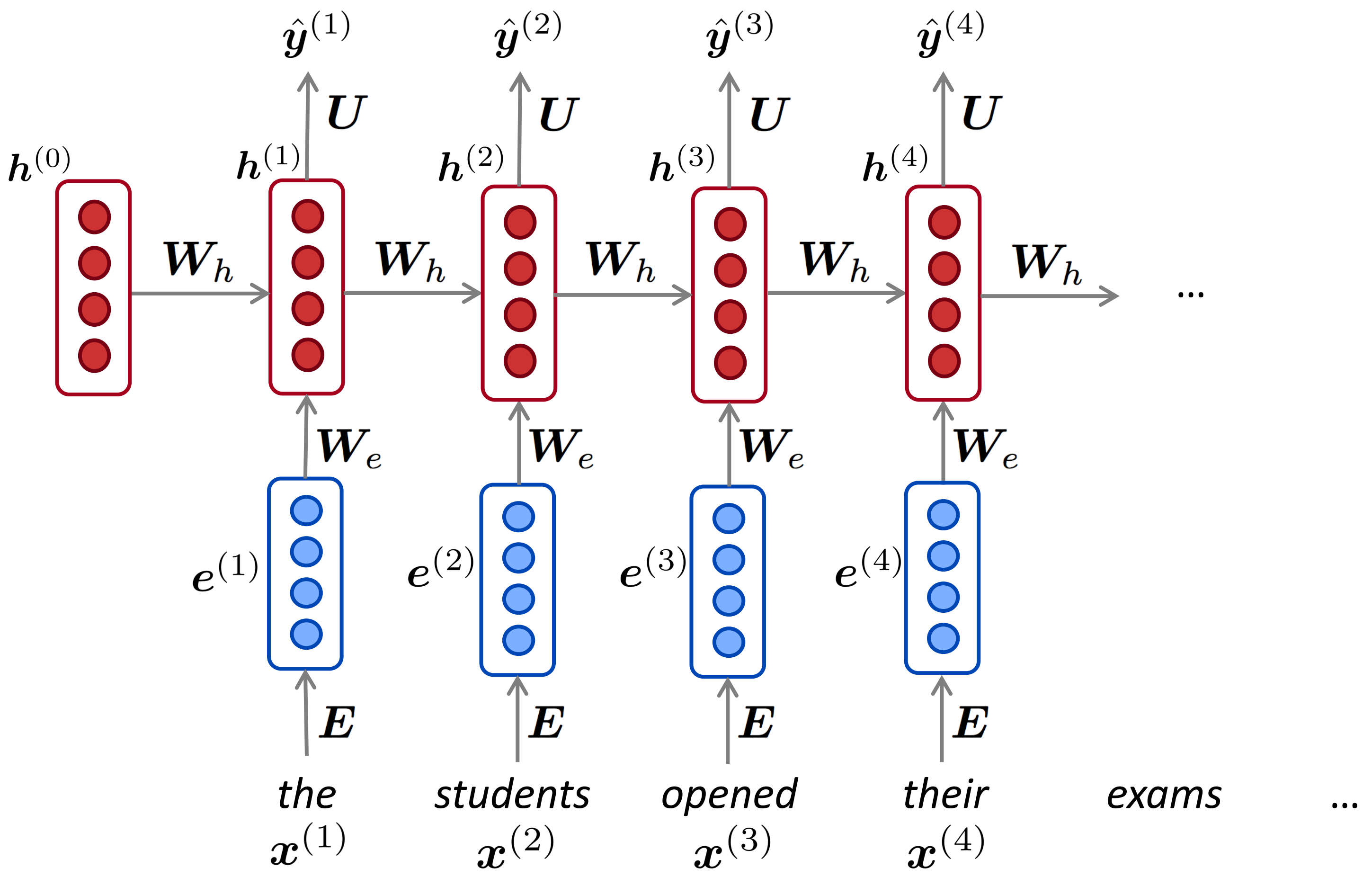
- Poster session time and location:
  - **5:30-8:30pm** at McCaw Hall at the Alumni Center
  - Note *time has changed*

This lecture

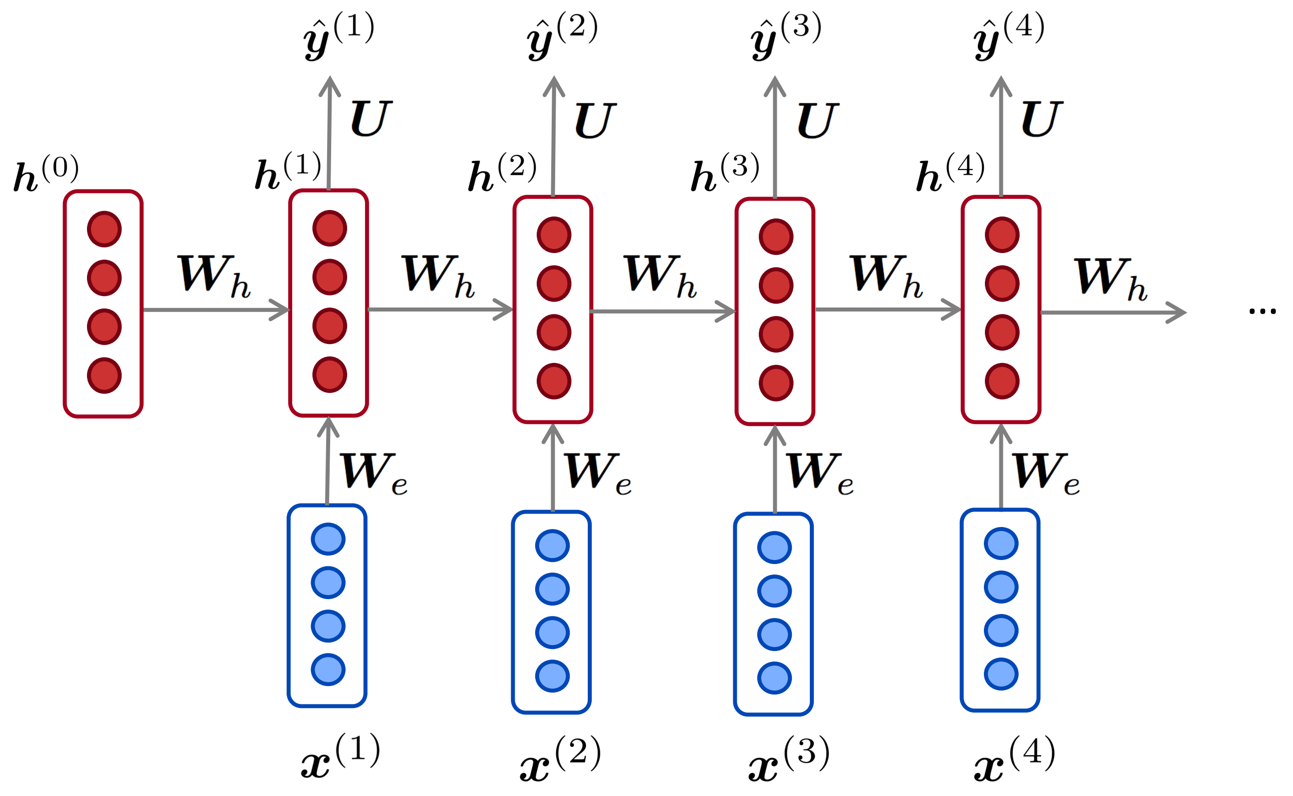
- Vanishing Gradient problem
- Fancy RNNs:
  - GRU
  - LSTM (!)
  - Bidirectional
  - Multi-layer

RNN Refresher

- Multiply the same matrix at each time step during forward prop



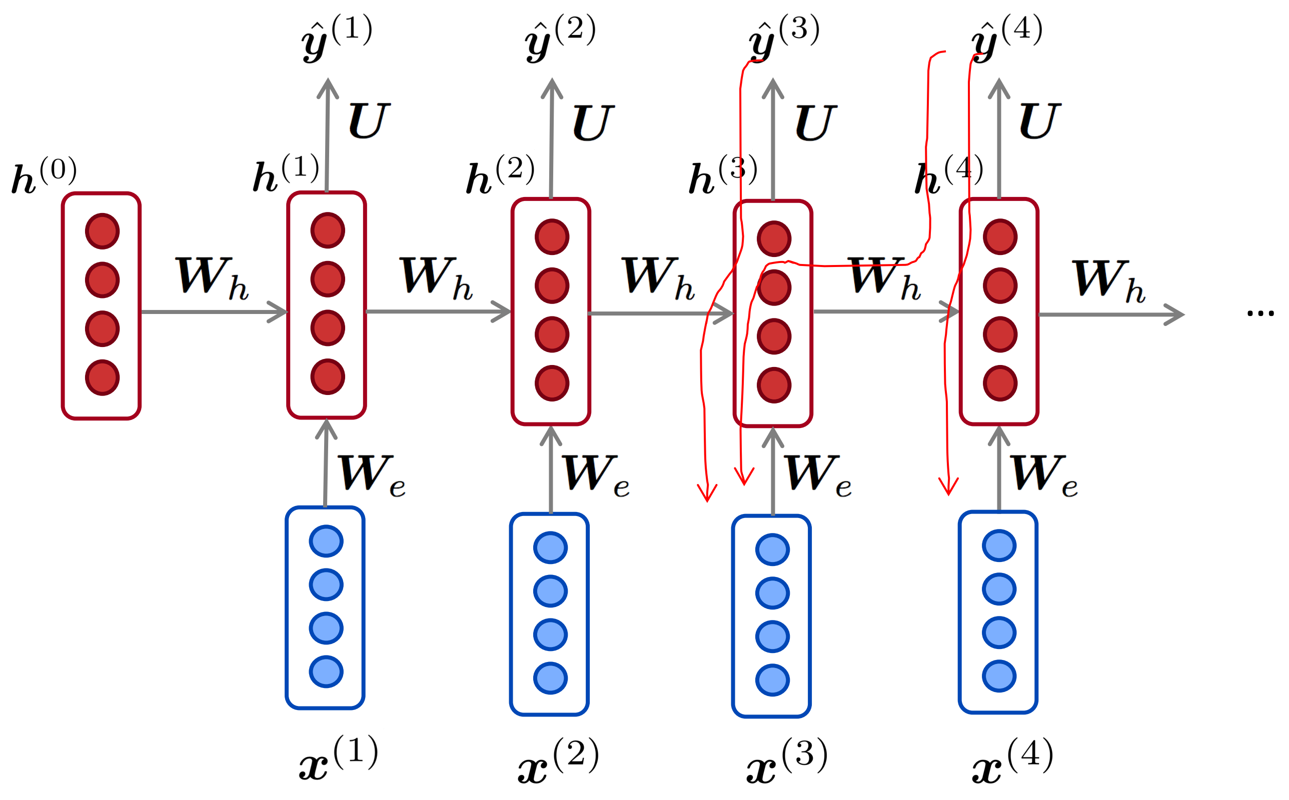
Simplify inputs to just x (usually word vectors)



- Ideally inputs from many time steps ago can modify output *y*
- Take $\frac{\partial E_2}{\partial W}$ for an example RNN with 2 time steps! Insightful!

The vanishing/exploding gradient problem

- Multiply the same matrix at each time step during backprop



The vanishing gradient problem - Details

- Similar but simpler RNN formulation:

$$\begin{aligned} h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\ \hat{y}_t &= W^{(S)} f(h_t) \end{aligned}$$

- Total error is the sum of each error (aka cost function, aka J in previous lectures when it was cross entropy error, could be other cost functions too), but at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

- Hardcore chain rule application:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

The vanishing gradient problem - Details

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

We'll show this can quickly become very small or very large

- Remember: $$h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$$
- More chain rule, remember:

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

- Each partial is a Jacobian:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

The vanishing gradient problem - Details

- Analyzing the norms of the Jacobians yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\mathrm{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

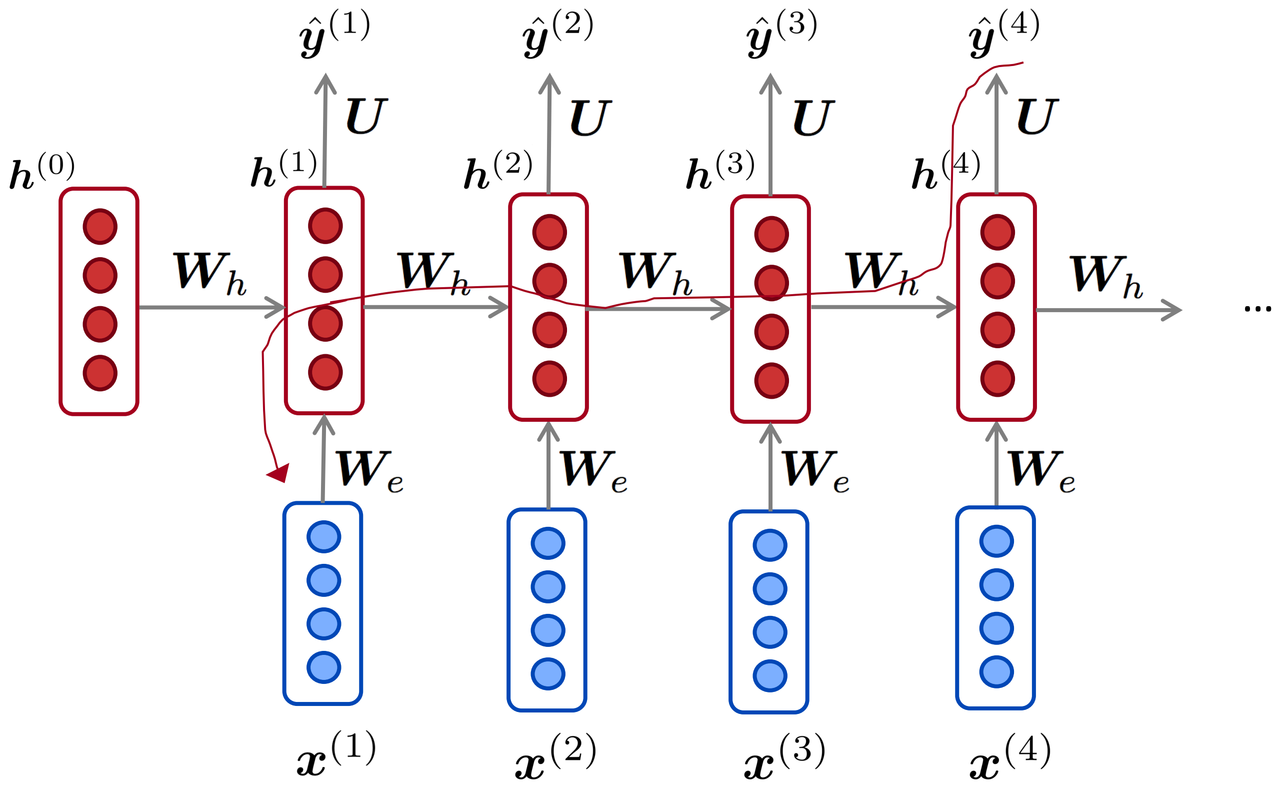
- Where we defined β's as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

- This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. → **Vanishing or exploding gradient**

Why is the vanishing gradient a problem?

- Ideally, the error E^t on step t can flow backwards, via backprop, and allow the weights on a previous timestep (maybe many timesteps ago) to change.



Why is the vanishing gradient a problem?

1. Gradients can be seen as a measure of influence of the past on the future

2. How does the perturbation at time t affect predictions at t+n?

Why is the vanishing gradient a problem?

When we only observe

$$\left\|\frac{\partial h_t}{\partial h_k}\right\| = \left\|\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}\right\| \leq (\beta_W \beta_h)^{t-k}$$ going to 0

We cannot tell whether

1. No dependency between *t* and *t*+*n* in data, or
2. Wrong configuration of parameters

The vanishing gradient problem for language models

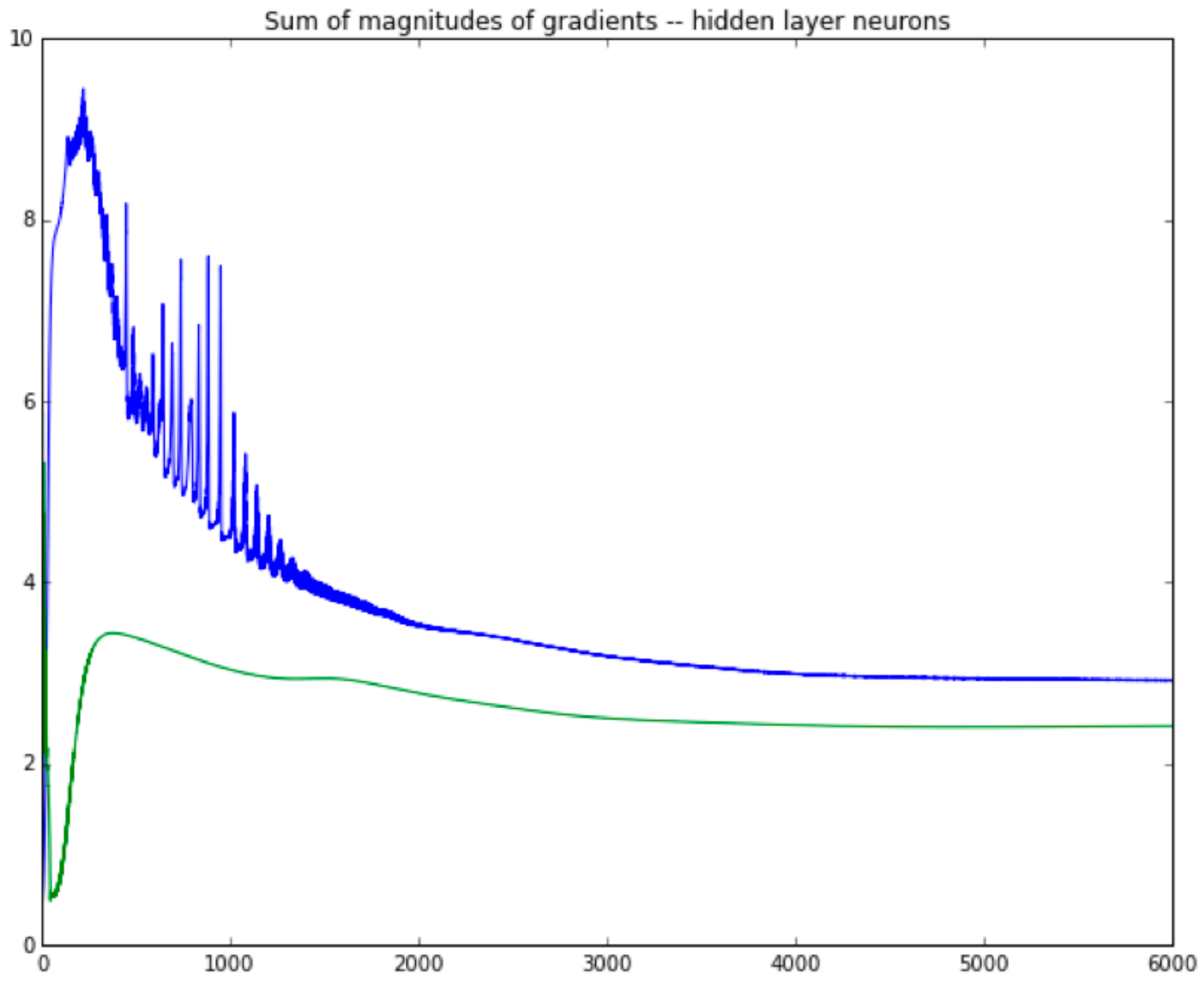
- The vanishing gradient problem can cause problems for RNN Language Models:
- When predicting the next word, information from many time steps in the past is not taken into consideration.

- Example:

  Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____

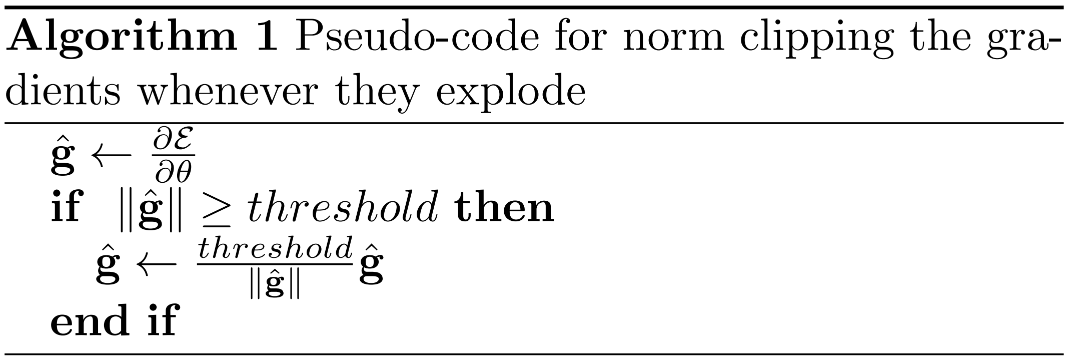
```
In [21]: plt.plot(np.array(relu_array[:6000]),color='blue')
         plt.plot(np.array(sigm_array[:6000]),color='green')
         plt.title('Sum of magnitudes of gradients -- hidden layer neurons')

Out[21]: <matplotlib.text.Text at 0x10a331310>
```



Trick for exploding gradient: clipping trick

- The solution first introduced by Mikolov is to clip gradients so that their norm has some maximum value.



Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

```
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if
```

- Makes a big difference in RNNs and many other unstable models

Gradient clipping intuition

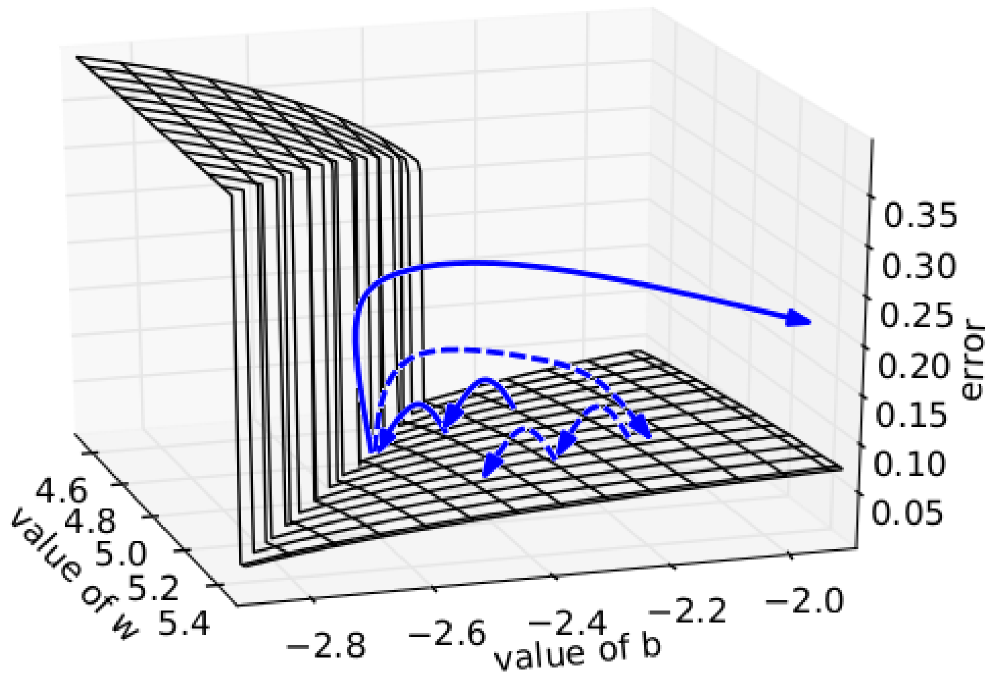


Figure from paper:
On the difficulty of training Recurrent Neural Networks, Pascanu et al. 2013

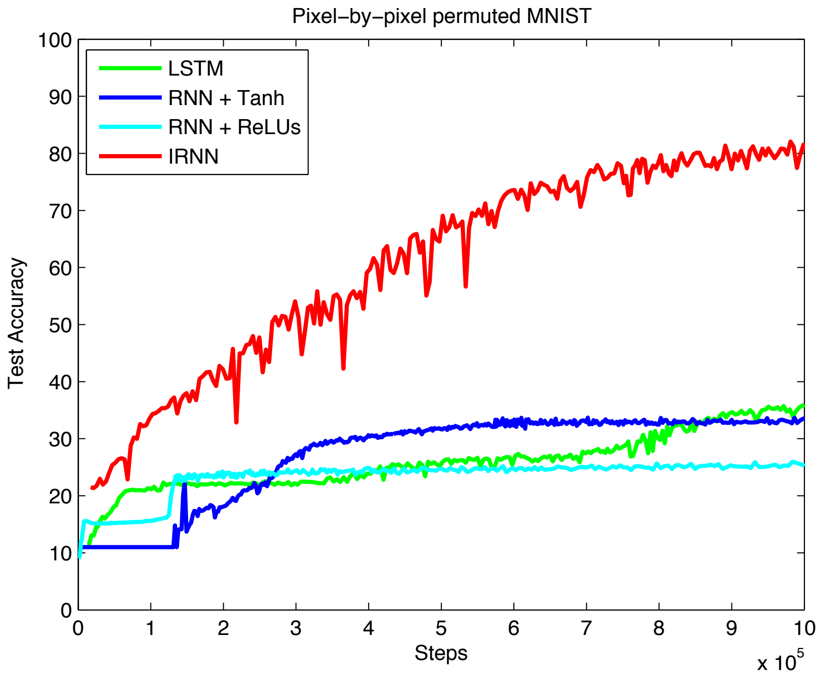
- Error surface of a single hidden unit RNN,
- High curvature walls
- Solid lines: standard gradient descent trajectories
- Dashed lines gradients rescaled to fixed size

One solution: Initialization + ReLus!

- You can improve the Vanishing Gradient Problem with good initialization and ReLUs.

- Initialize $W^{(*)}$'s to identity matrix I and f(z) $= \mathrm{rect}(z) = \max(z,0)$

- → Huge difference!



- Initialization idea first introduced in *Parsing with Compositional Vector Grammars*, Socher et al. 2013

- New experiments with recurrent neural nets in *A Simple Way to Initialize Recurrent Networks of Rectified Linear* Units, Le et al. 2015

Main solution for better RNNs: Better Units

- The main solution to the Vanishing Gradient Problem is to use a more complex hidden unit computation in recurrence!
- Gated Recurrent Units (GRU) introduced by [Cho et al. 2014] and LSTMs [Hochreiter & Schmidhuber, 1999]
- 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: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$
- #Simplified from last lecture's: $\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$
- GRU first computes an update **gate** (another layer) based on current input word vector and hidden state

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

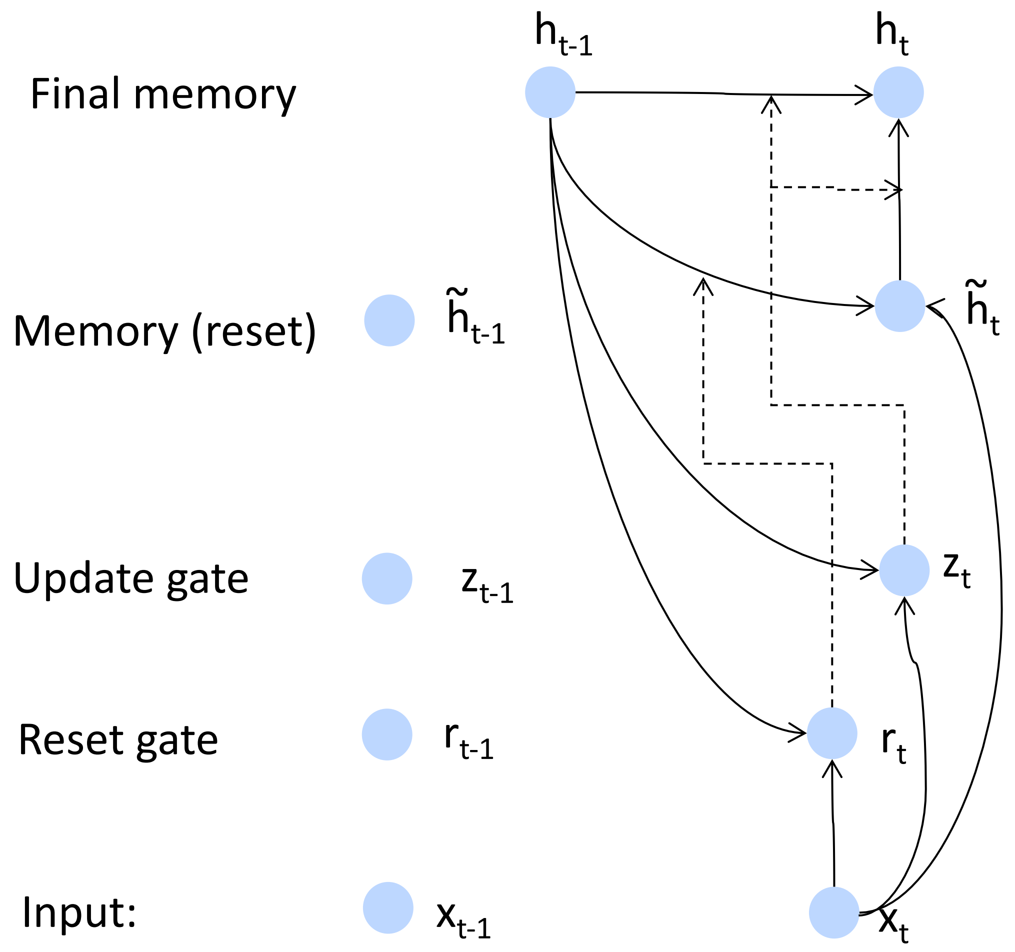
- Compute reset gate similarly but with different weights

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

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

GRU 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

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

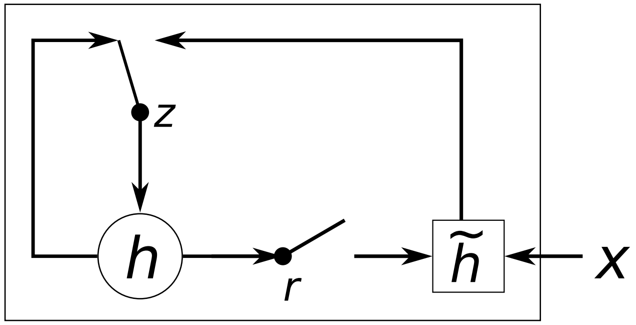
- Update gate z controls how much of past state should matter now.
  - If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient**!
- Units with short-term dependencies often have reset gates very active

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:



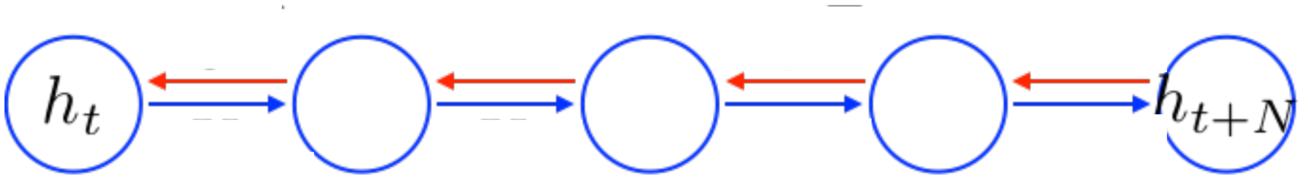
- Derivative of $\frac{\partial}{\partial x_1} x_1 x_2$? → rest is same chain rule, but implement with **modularization** or automatic differentiation

How do Gated Recurrent Units fix vanishing gradient problems?

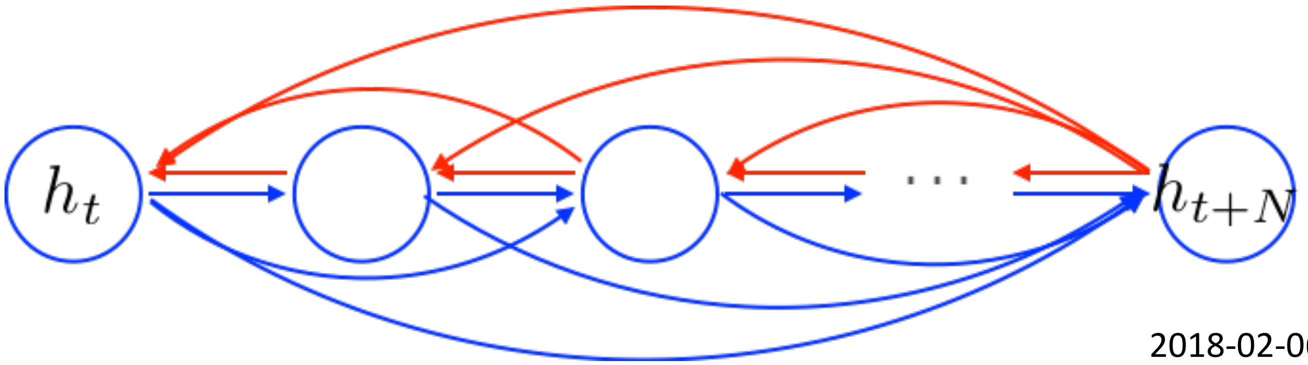
- Is the problem with standard RNNs the naïve transition function?

$$h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$$

- It implies that the error must backpropagate through all the intermediate nodes:

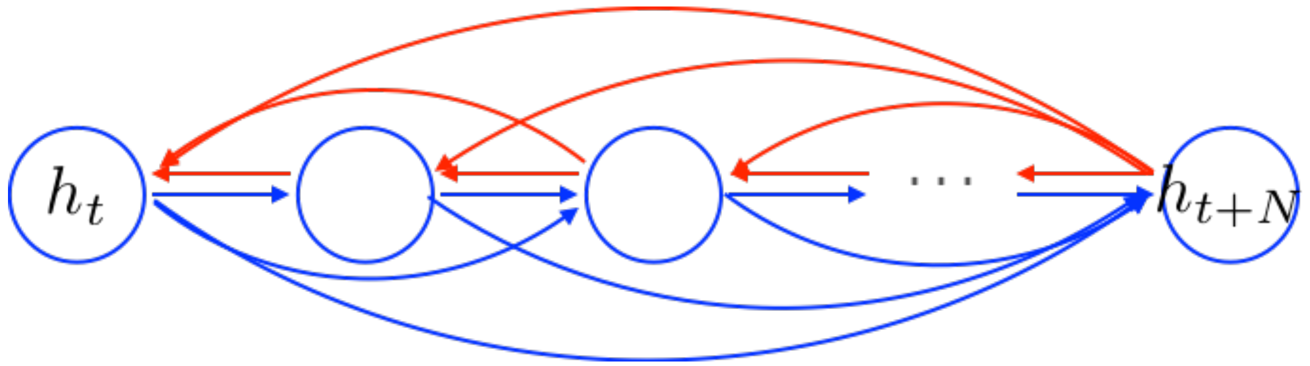


- Perhaps we can create shortcut connections.



How do Gated Recurrent Units fix vanishing gradient problems?

- Perhaps we can create *adaptive* shortcut connections.
- Let the net prune unnecessary connections *adaptively*.



- That's what the gates do.

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

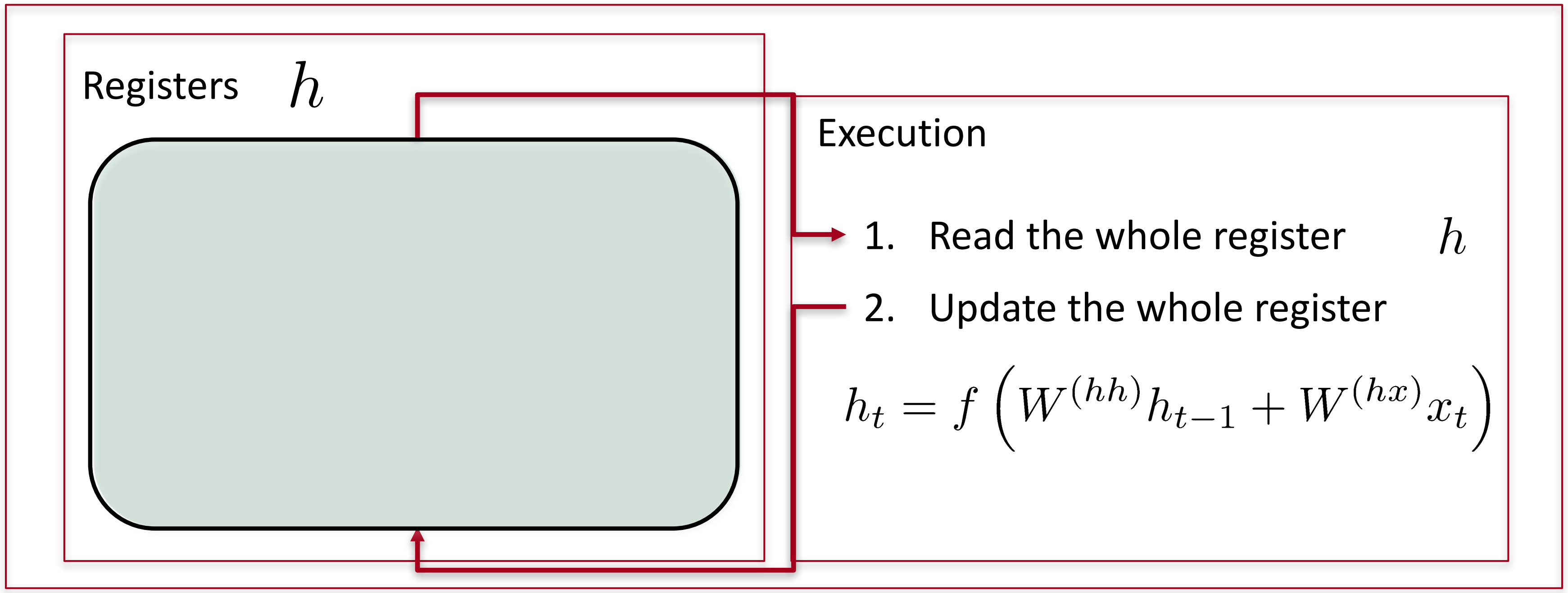
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

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

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

GRU Comparison to Standard tanh-RNN

Vanilla RNN …



Registers h

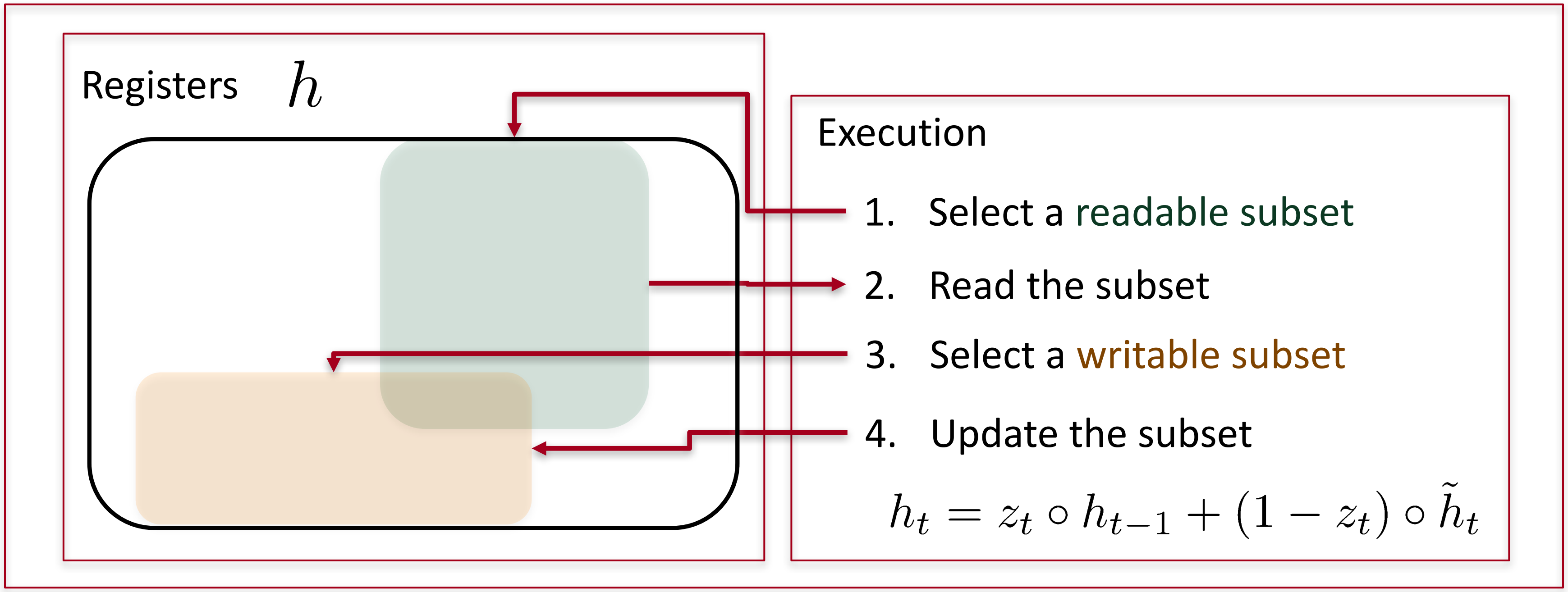
Execution

1. Read the whole register h
2. Update the whole register

$$h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$$

GRU Comparison to Standard tanh-RNN

GRU …



1. Select a readable subset
2. Read the subset
3. Select a writable subset
4. Update the subset

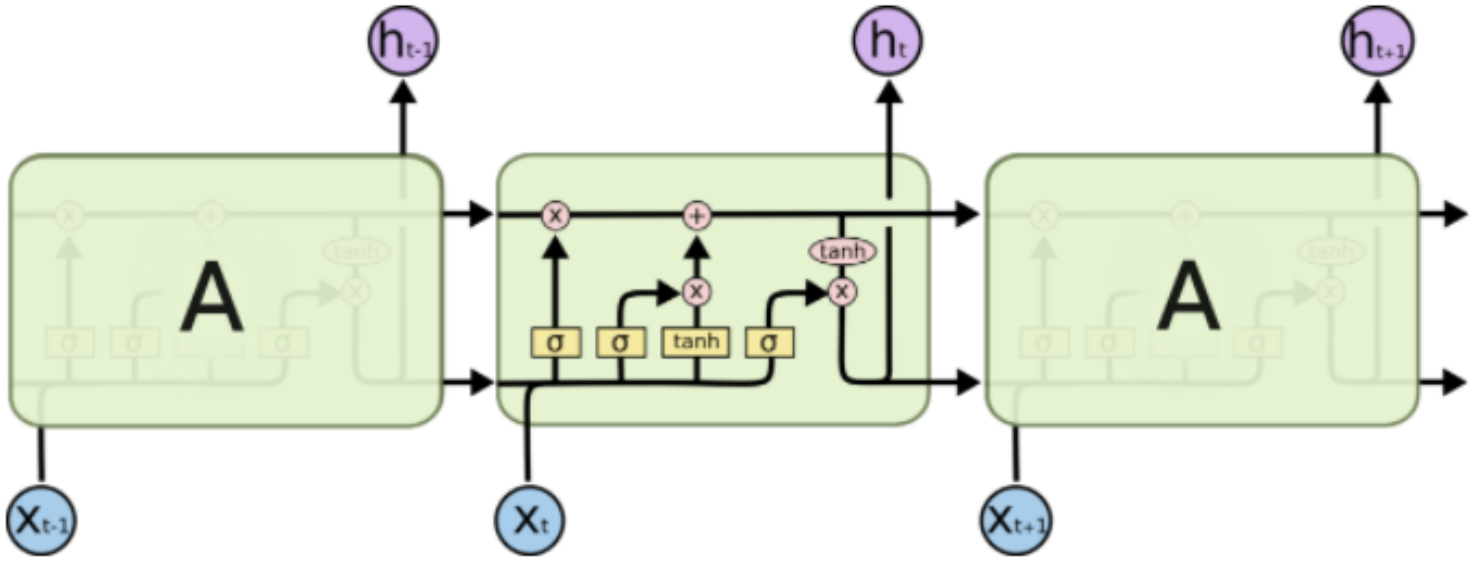
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

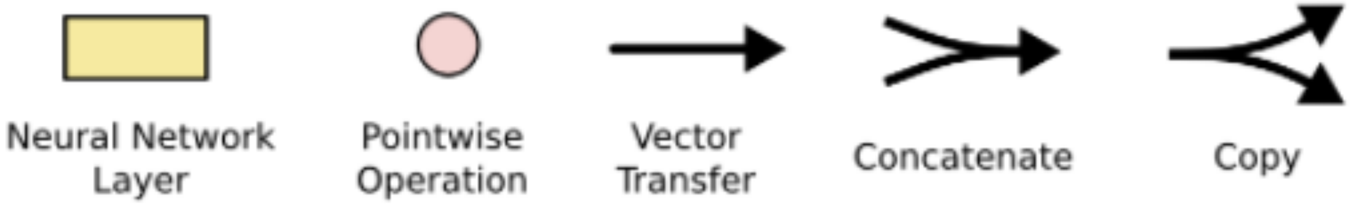
Gated recurrent units are much more versatile and adaptive in which elements of the hidden vector h they update!

Long-short-term-memories (LSTMs)

- LSTM is even more complex than GRU
- 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: $h_t = o_t \circ \tanh(c_t)$

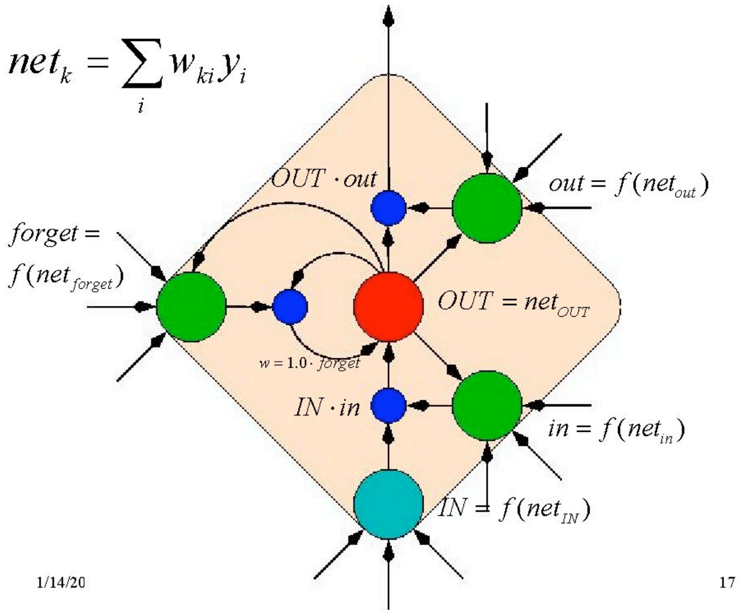
Some visualizations



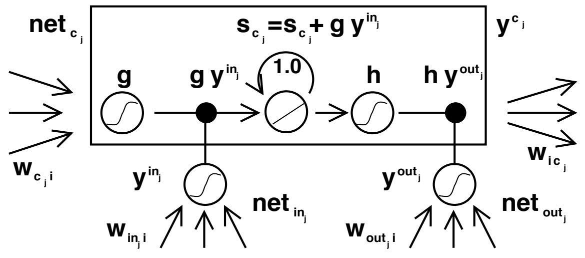


By Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

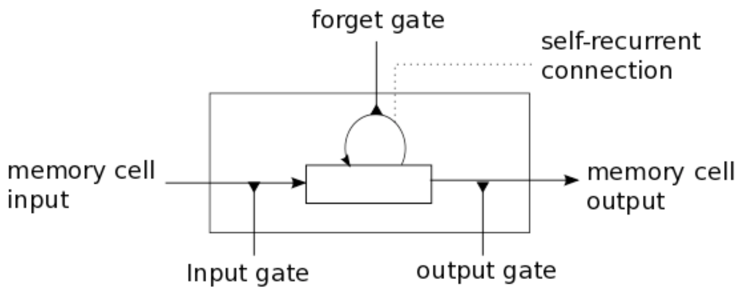
Most illustrations a bit overwhelming ;)



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



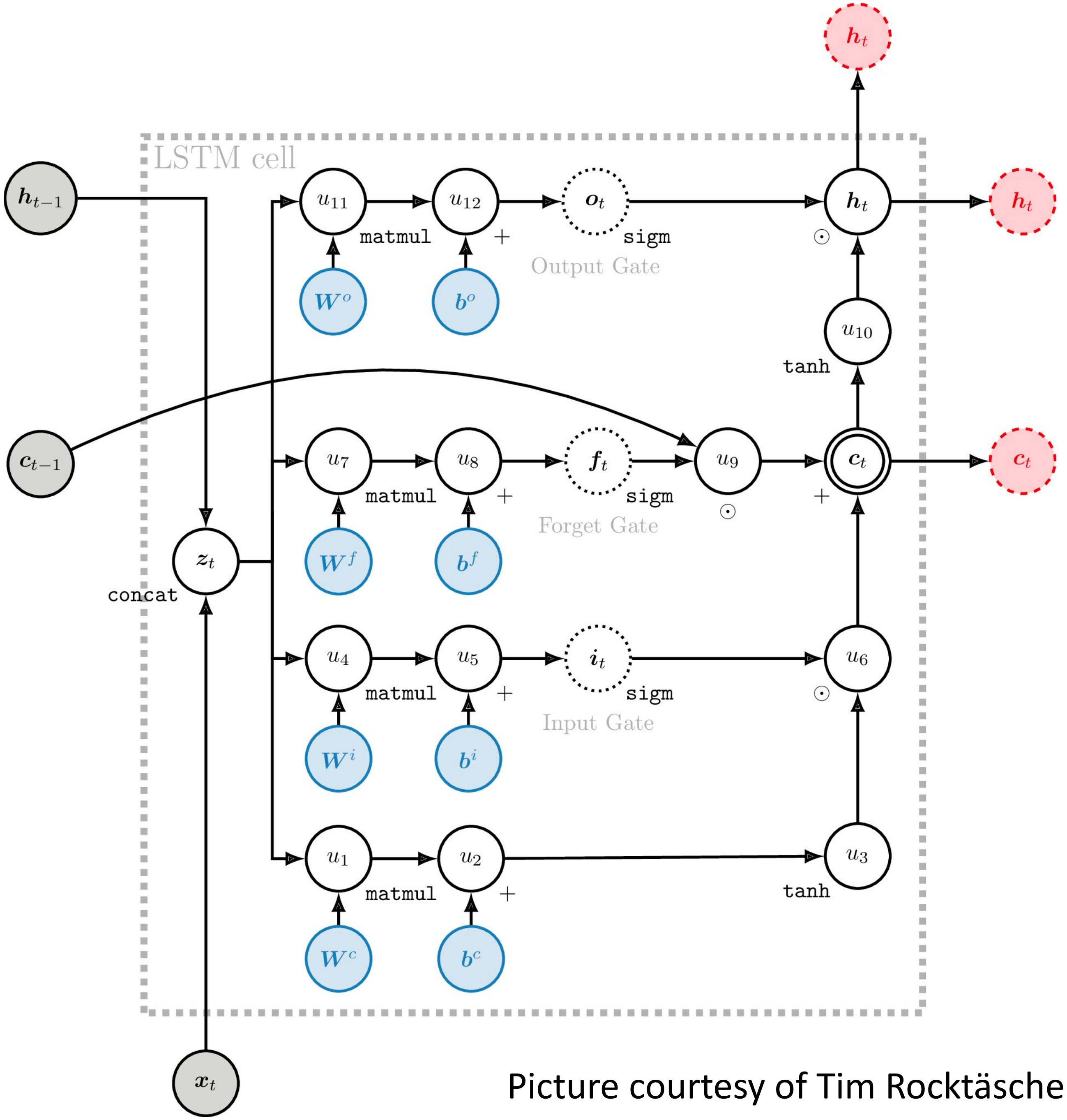
Long Short-Term Memory by Hochreiter and Schmidhuber (1997)



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

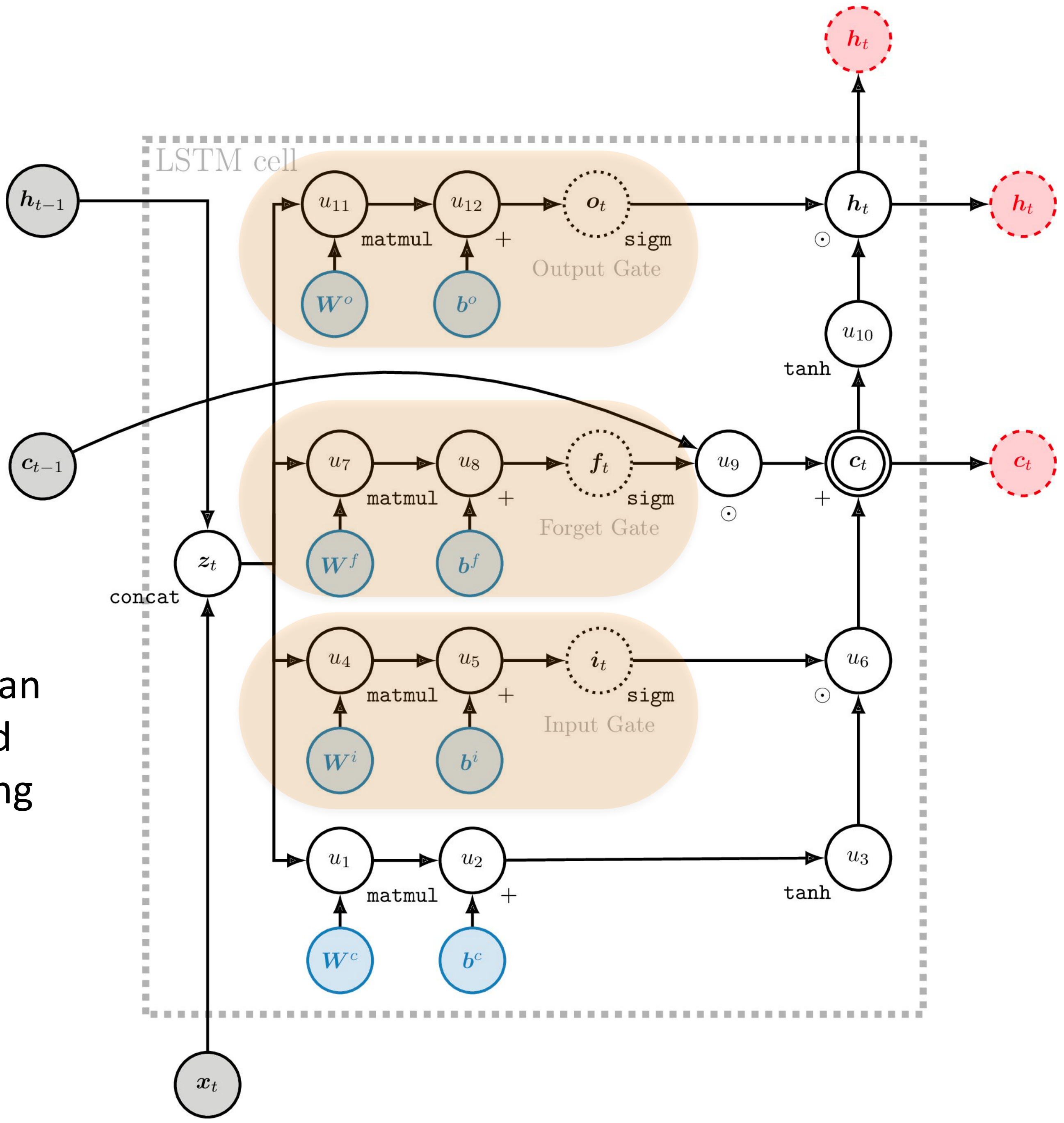
Intuition: memory cells can keep information intact, unless inputs makes them forget it or overwrite it with new input.
Cell can decide to output this information or just store it

Another LSTM visualization inspired by code



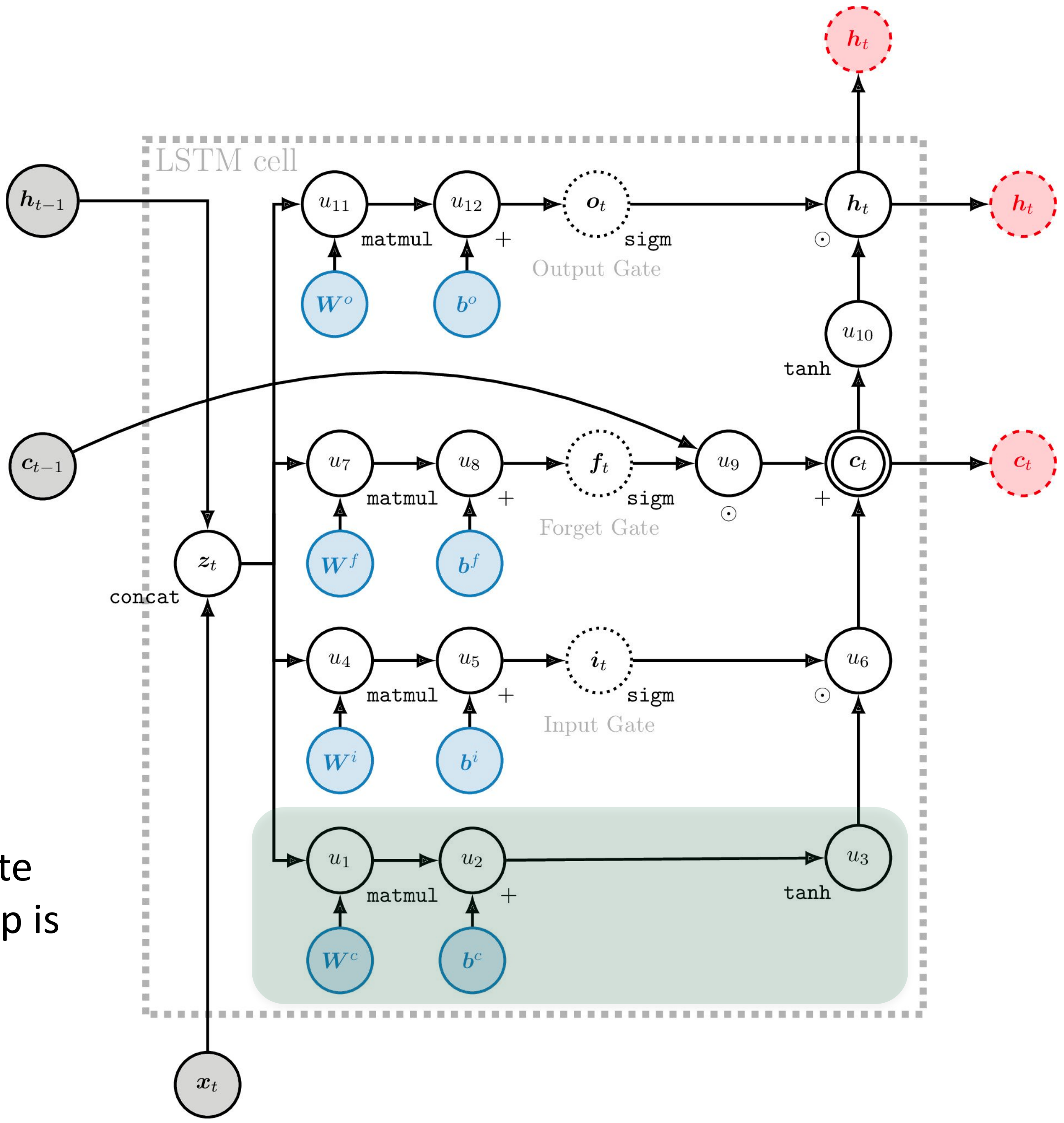
Picture courtesy of Tim Rocktäschel

The LSTM



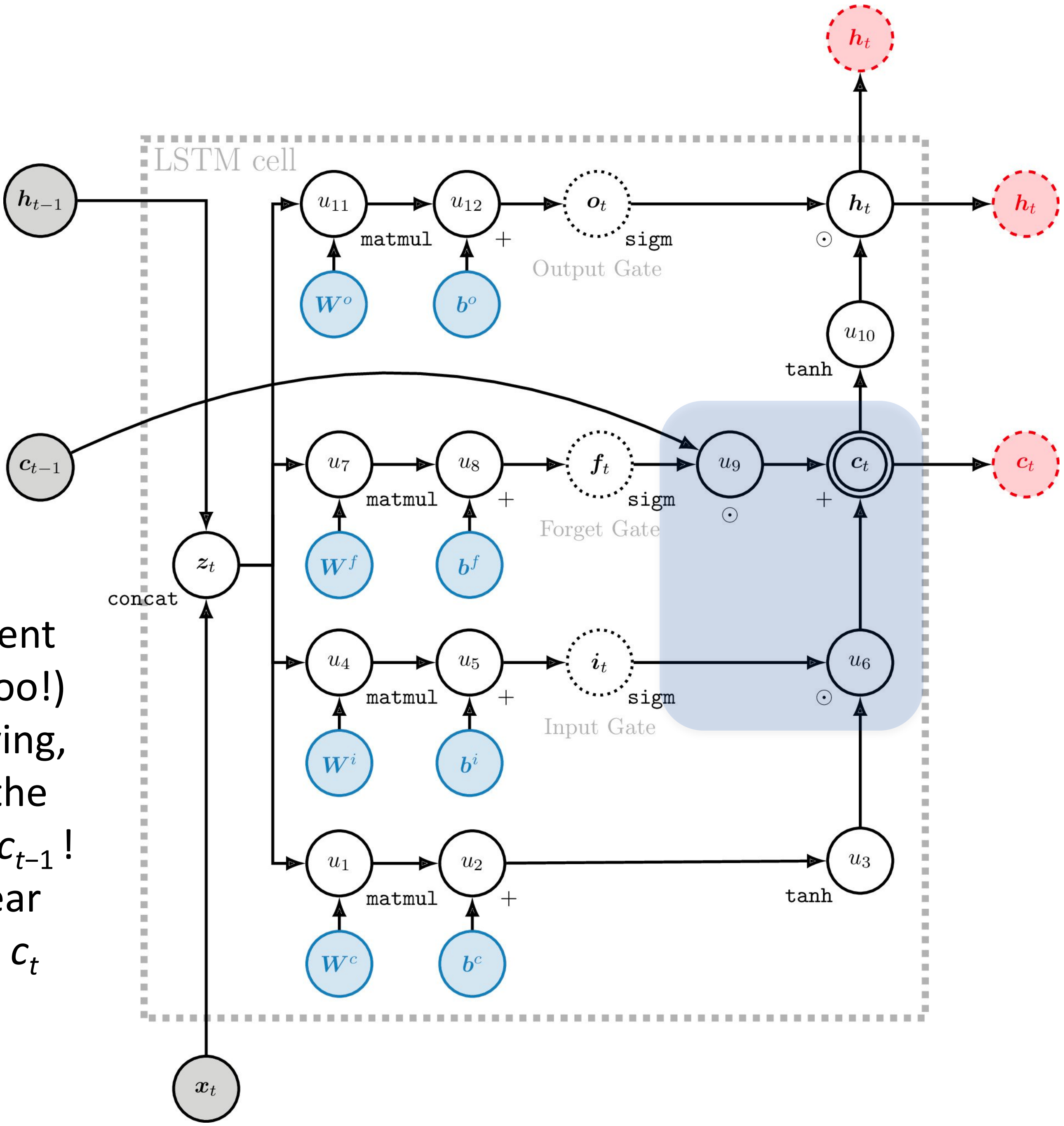
The LSTM gates all operations so stuff can be forgotten/ignored rather than it all being crammed on top of everything else

The LSTM



The non-linear update for the next time step is just like an RNN

The LSTM



This part is the the secret! (Of other recent things like ResNets too!) Rather than multiplying, we get c_t by adding the non-linear stuff and c_{t-1} ! There is a direct, linear connection between c_t and c_{t-1}.

LSTM visualization after training for character language modeling (predict the next character)

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

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

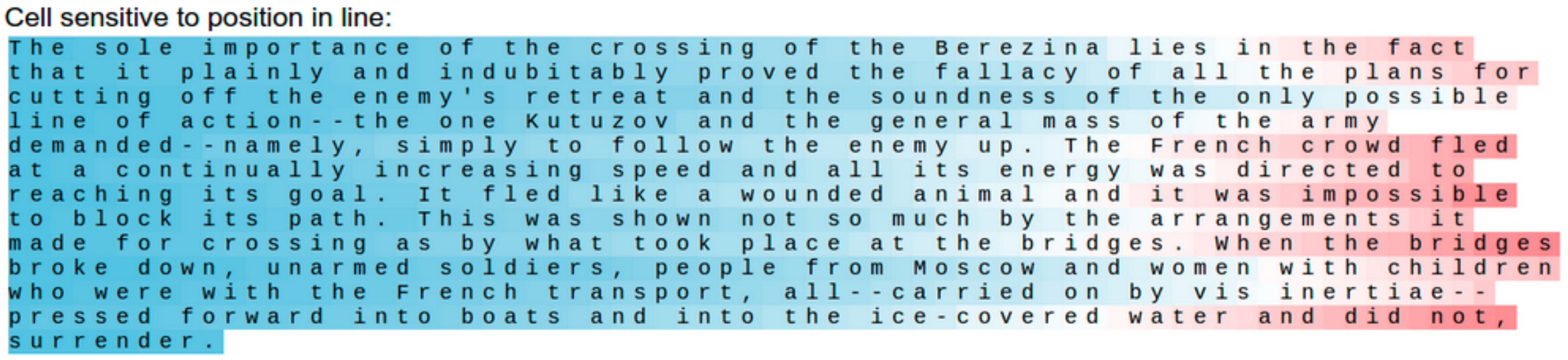
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$

$$\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \tanh(c_t)$$

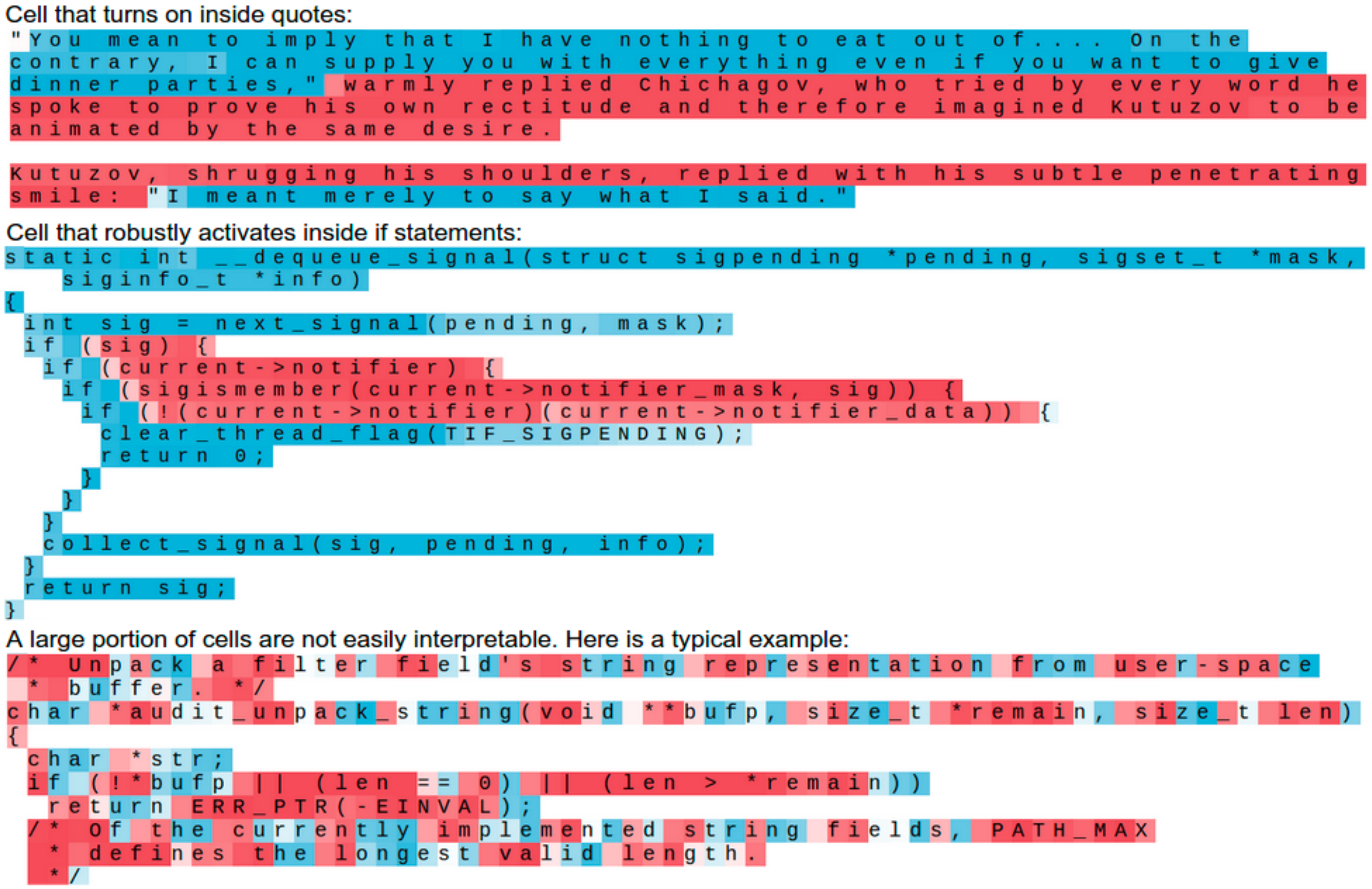
Visualizing activation of $\tanh(c_t)$:


Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

From: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LSTM visualization after training for character language modeling (predict the next character)



Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
   siginfo_t *info)
{
 int sig = next_signal(pending, mask);
 if (sig) {
  if (current->notifier) {
   if (sigismember(current->notifier_mask, sig)) {
    if (!(current->notifier)(current->notifier_data)) {
     clear_thread_flag(TIF_SIGPENDING);
     return 0;
    }
   }
  }
  collect_signal(sig, pending, info);
 }
 return sig;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
 char *str;
 if (!*bufp || (len == 0) || (len > *remain))
  return ERR_PTR(-EINVAL);
 /* Of the currently implemented string fields, PATH_MAX
  * defines the longest valid length.
  */
```

From: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LSTMs are a great default for all sequence problems

- 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

Deep LSTMs compared to traditional systems 2015

Method	**test BLEU score (ntst14)**
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

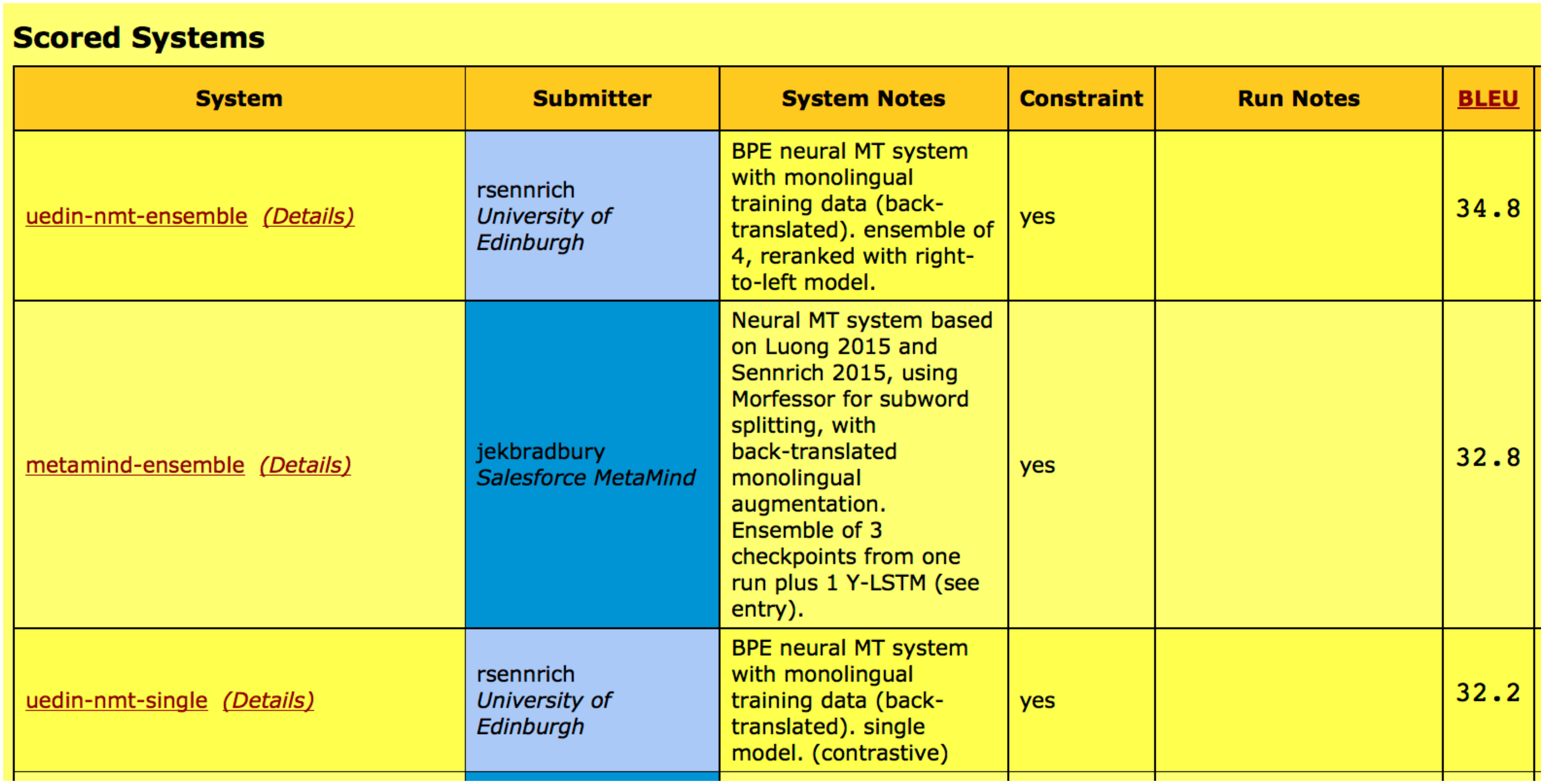
Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.

Method	**test BLEU score (ntst14)**
Baseline System [29]	33.30
Cho et al. [5]	34.54
Best WMT'14 result [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	$\sim$45

Sequence to Sequence Learning by Sutskever et al. 2014

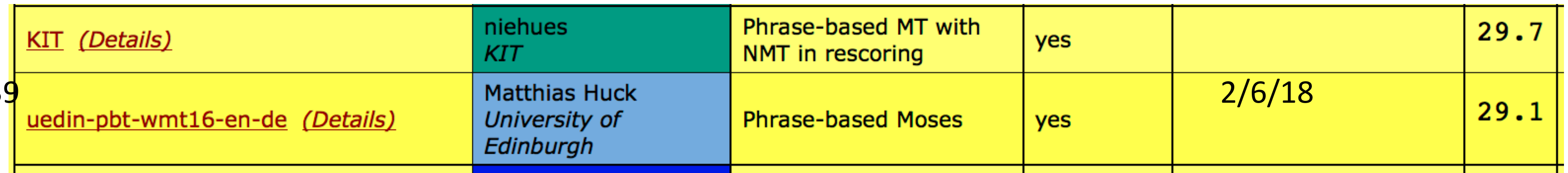
Deep LSTMs (with a lot more tweaks)

WMT 2016 competition results



Scored Systems

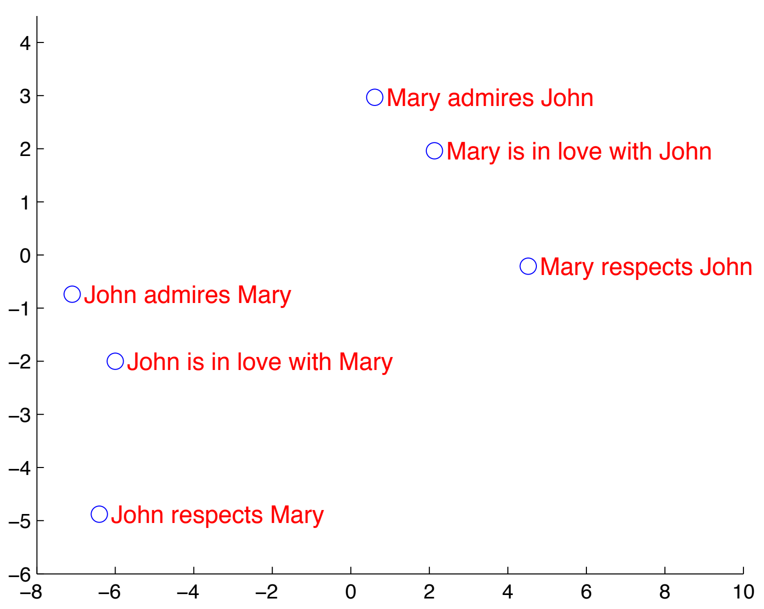
System	Submitter	System Notes	Constraint	Run Notes	BLEU
uedin-nmt-ensemble (Details)	rsennrich *University of Edinburgh*	BPE neural MT system with monolingual training data (back-translated). ensemble of 4, reranked with right-to-left model.	yes		34.8
metamind-ensemble (Details)	jekbradbury *Salesforce MetaMind*	Neural MT system based on Luong 2015 and Sennrich 2015, using Morfessor for subword splitting, with back-translated monolingual augmentation. Ensemble of 3 checkpoints from one run plus 1 Y-LSTM (see entry).	yes		32.8
uedin-nmt-single (Details)	rsennrich *University of Edinburgh*	BPE neural MT system with monolingual training data (back-translated). single model. (contrastive)	yes		32.2

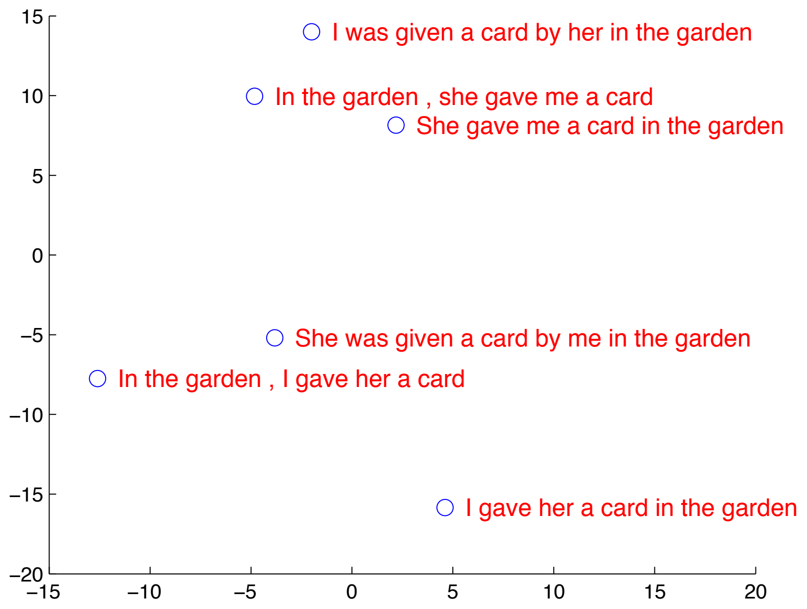


System	Submitter	System Notes	Constraint	Run Notes	BLEU
KIT (Details)	niehues *KIT*	Phrase-based MT with NMT in rescoring	yes		29.7
uedin-pbt-wmt16-en-de (Details)	Matthias Huck *University of Edinburgh*	Phrase-based Moses	yes		29.1

Deep LSTM for Machine Translation

PCA of vectors from last time step hidden layer

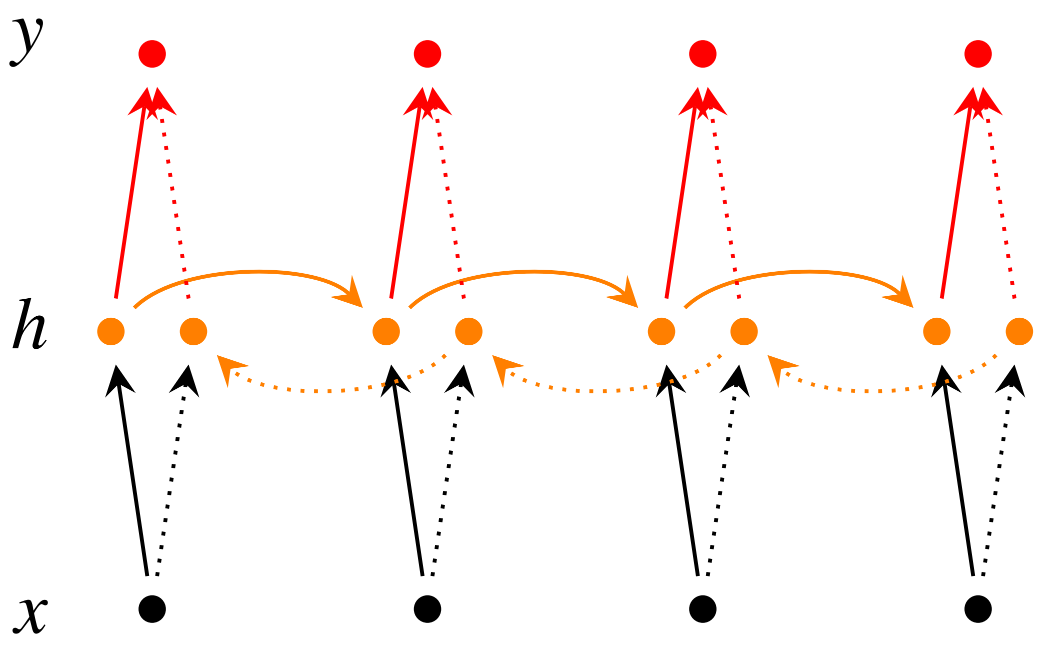




Sequence to Sequence Learning by Sutskever et al. 2014

Bidirectional RNNs

Problem: For classification you want to incorporate information from words both preceding and following



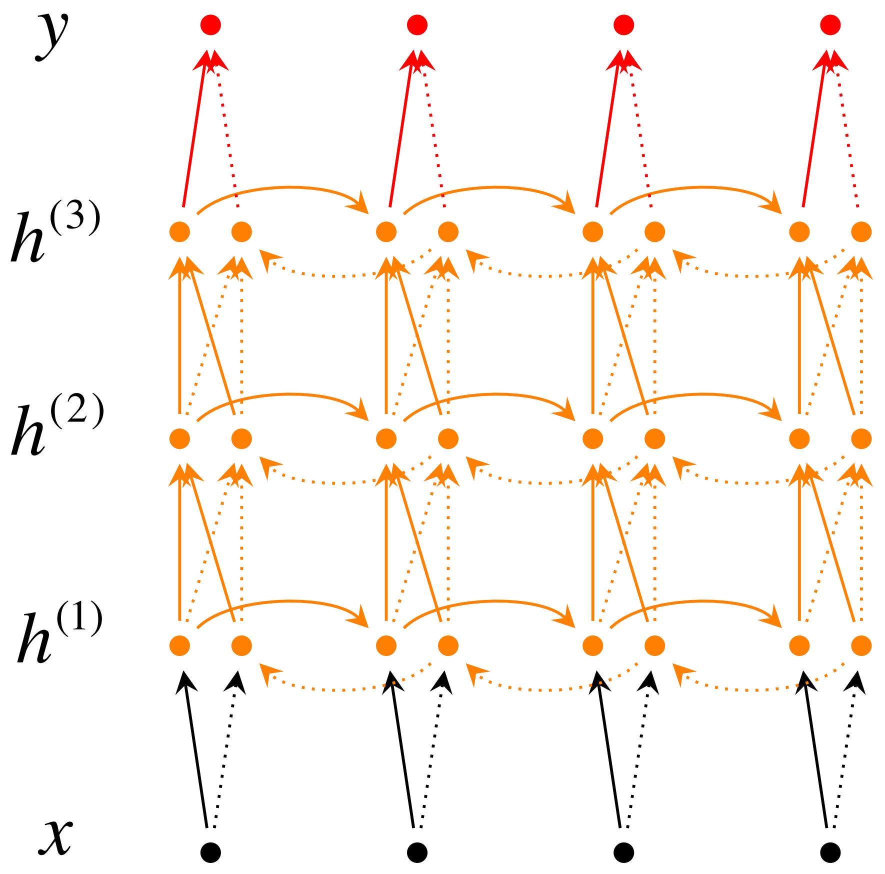
$$\overrightarrow{h}_t = f(\overrightarrow{W} x_t + \overrightarrow{V} \overrightarrow{h}_{t-1} + \overrightarrow{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W} x_t + \overleftarrow{V} \overleftarrow{h}_{t+1} + \overleftarrow{b})$$

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

$h = [\overrightarrow{h}; \overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.

Deep Bidirectional RNNs



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

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

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

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

Next up!

Midterm review!

Gated Recurrent Units Comparison (different notation)

Two most widely used gated recurrent units

Gated Recurrent Unit

[Cho et al., EMNLP2014;
Chung, Gulcehre, Cho, Bengio, DLUFL2014]

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

$$\tilde{h}_t = \tanh(W\,[x_t] + U(r_t \odot h_{t-1}) + b)$$

$$u_t = \sigma(W_u\,[x_t] + U_u h_{t-1} + b_u)$$

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

Long Short-Term Memory

[Hochreiter & Schmidhuber, NC1999;
Gers, Thesis2001]

$$h_t = o_t \odot \tanh(c_t)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$\tilde{c}_t = \tanh(W_c\,[x_t] + U_c h_{t-1} + b_c)$$

$$o_t = \sigma(W_o\,[x_t] + U_o h_{t-1} + b_o)$$

$$i_t = \sigma(W_i\,[x_t] + U_i h_{t-1} + b_i)$$

$$f_t = \sigma(W_f\,[x_t] + U_f h_{t-1} + b_f)$$

Training a (gated) RNN

1. Use an LSTM or GRU: *it makes your life so much simpler!*
2. Initialize recurrent matrices to be orthogonal
3. Initialize other matrices with a sensible (**small!**) scale
4. Initialize forget gate bias to 1: *default to remembering*
5. Use adaptive learning rate algorithms: *Adam, AdaDelta, …*
6. Clip the norm of the gradient: *1–5 seems to be a reasonable threshold when used together with Adam or AdaDelta.*
7. Either only dropout vertically or learn how to do it right
8. *Be patient!*

[Saxe et al., ICLR2014;
Ba, Kingma, ICLR2015;
Zeiler, arXiv2012;
Pascanu et al., ICML2013]