Natural Language Processing with Deep Learning

CS6010

Lecture 6:

Vanishing Gradients,

Fancy RNNs (LSTMs and GRUs) and applications

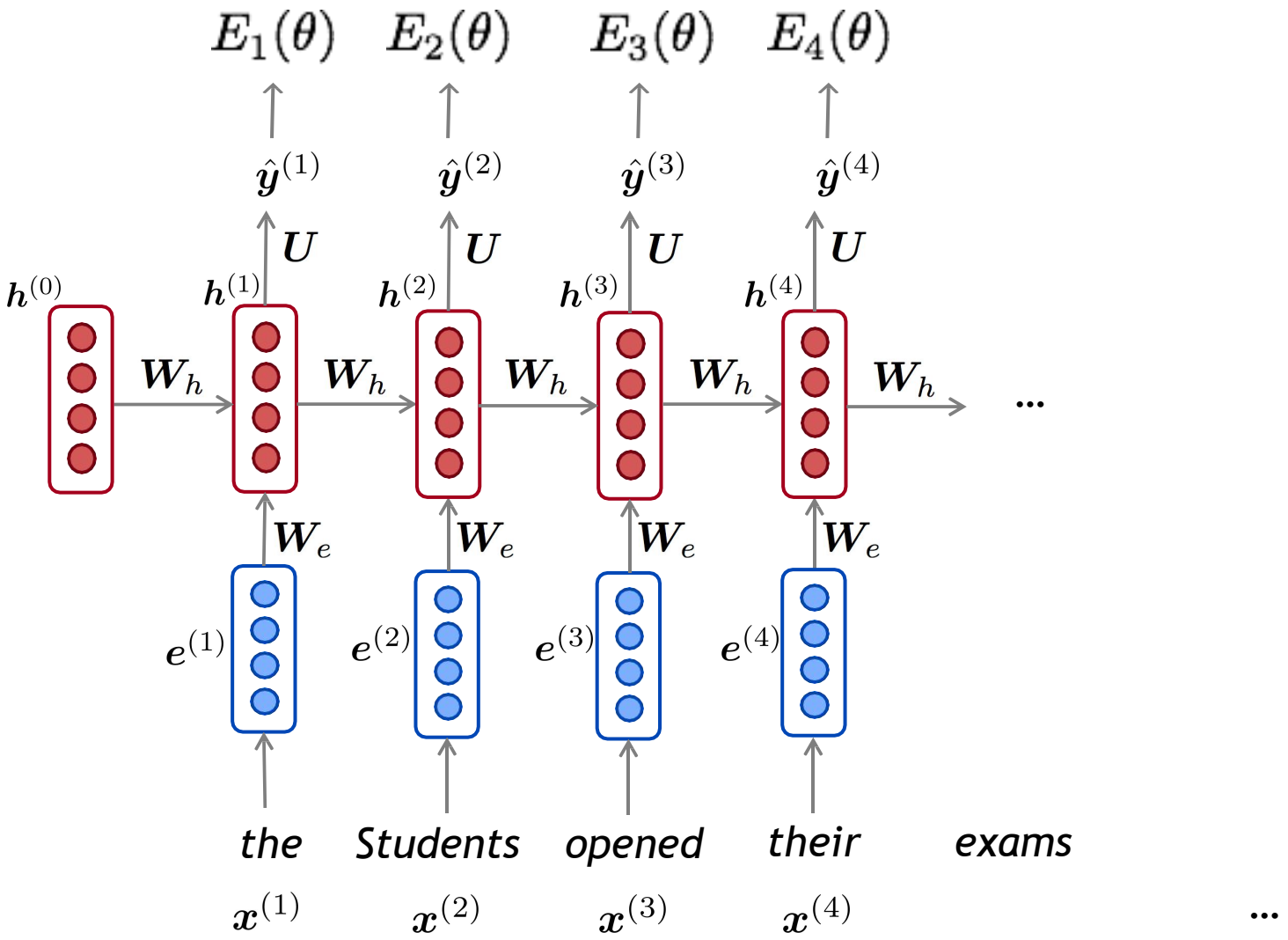
Presenters:
Mirco Milletari
Tram Anh Nguyen
Chenglei Si
Tasbolat Taunyazov
Chen Yang

This lecture

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

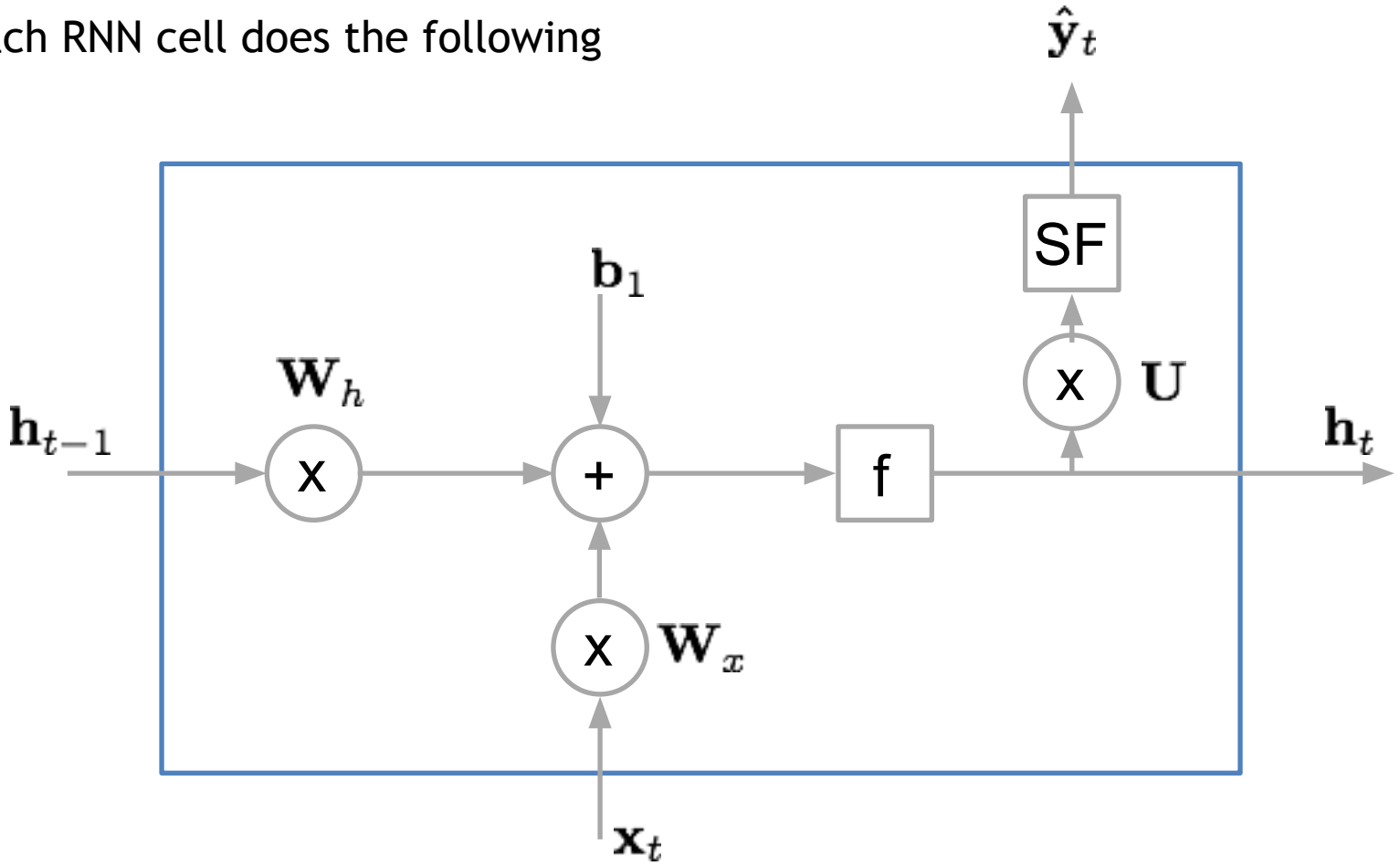
RNN Refresher I

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



RNN Refresher II

- Each RNN cell does the following



In a nutshell:

$$\mathbf{h}_t = f(\mathbf{W}_h\,\mathbf{h}_{t-1} + \mathbf{W}_x\,\mathbf{x}_t + \mathbf{b}_1)$$

$$\hat{\mathbf{y}}_t = softmax(\mathbf{U}\,\mathbf{h}_t + \mathbf{b}_2)$$

The vanishing gradient problem - Details

- Similar but simpler RNN formulation:

$$\mathbf{h}_t = \mathbf{W}_h f(\mathbf{h}_{t-1}) + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}_1$$
$$\hat{\mathbf{y}}_t = \mathbf{U} f(\mathbf{h}_t) + b_2$$

- Total error is the sum of each Error at time step t

$$\frac{dE}{d\mathbf{W}_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{dE_t}{d\mathbf{W}_h}$$

- Take t=2 for example, then ($\mathbf{e}_2 \propto (\hat{\mathbf{y}}_2 - \mathbf{y}_2)$)

$$\frac{dE_2}{d\mathbf{W}_h} = \mathbf{e}_2 \frac{d\hat{\mathbf{y}}_2}{d\mathbf{W}_h} = \mathbf{e}_2 \frac{\partial \hat{\mathbf{y}}_2}{\partial \mathbf{h}_2} \left\{ f(\mathbf{h}_1) + \mathbf{W}_h f'(\mathbf{h}_1) f(\mathbf{h}_0) + \mathbf{W}_h f'(\mathbf{h}_1) \mathbf{W}_h f'(\mathbf{h}_0) \frac{\partial \mathbf{h}_0}{\partial \mathbf{W}_h} \right\}$$
$$= \frac{\partial E_2}{\partial \hat{\mathbf{y}}_2} \frac{\partial \hat{\mathbf{y}}_2}{\partial \mathbf{h}_2} \sum_{k=0}^{2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_h}$$

R. Pascanu et al. JMLR: W&CP volume 28 (2013)

The vanishing gradient problem - Details

- In particular, we have defined: $\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_0} = \prod_{j=1}^{2} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}$

- Generalising to -t- and counting from k=1 we get

$$\frac{dE_t}{d\mathbf{W}_h} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_h}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} = \prod_{j=k+1}^{t} \hat{\mathbf{W}}_h^T \, diag[f'(\mathbf{h}_{j-1})]$$

- Each partial is a Jacobian:

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

Analyze the norms of the Jacobians

1. Assume $|f'(\mathbf{h}_i)|$ is bounded
2. $||diag[f'(\mathbf{h}_{j-1})|| \leq \gamma \in \mathcal{R}$

Using the Cauchy-Schwarz inequality

$$\forall j \left\| \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\| = ||\mathbf{W}_h^T \, diag[f'(\mathbf{h}_{j-1})|| \leq ||\mathbf{W}_h^T|| \, ||diag[f'(\mathbf{h}_{j-1})||$$

If the largest eigenvalue of W is $\lambda_1 < 1/\gamma$, then

$$\forall j \left\| \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\| \leq \frac{1}{\gamma}\gamma < 1 \quad \xrightarrow{\text{General}} \quad \forall j \left\| \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\| \leq \eta < 1$$

$$\eta = \beta_W \, \beta_h$$

Consider the full expression now

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\|$$

- Do the simple case first!

$$\left\| \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1} \right\| = \left\| \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \right\| \leq \left\| \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \right\| \left\| \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \right\| \leq \eta^2$$

- From direct inspection or by looking at scaling dimensions, one finds the general result

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\| \leq \eta^{t-k}$$

- This can be very small in the long term! Mathematically

$$t \gg k \quad \Rightarrow \quad \eta^{t-k} \to 0 \qquad (\eta < 1)$$

- For the exploding gradient condition, consider $\lambda_1 > 1/\gamma$, then going through the same steps one finds $\eta > 1$, from which the exploding gradient follows.

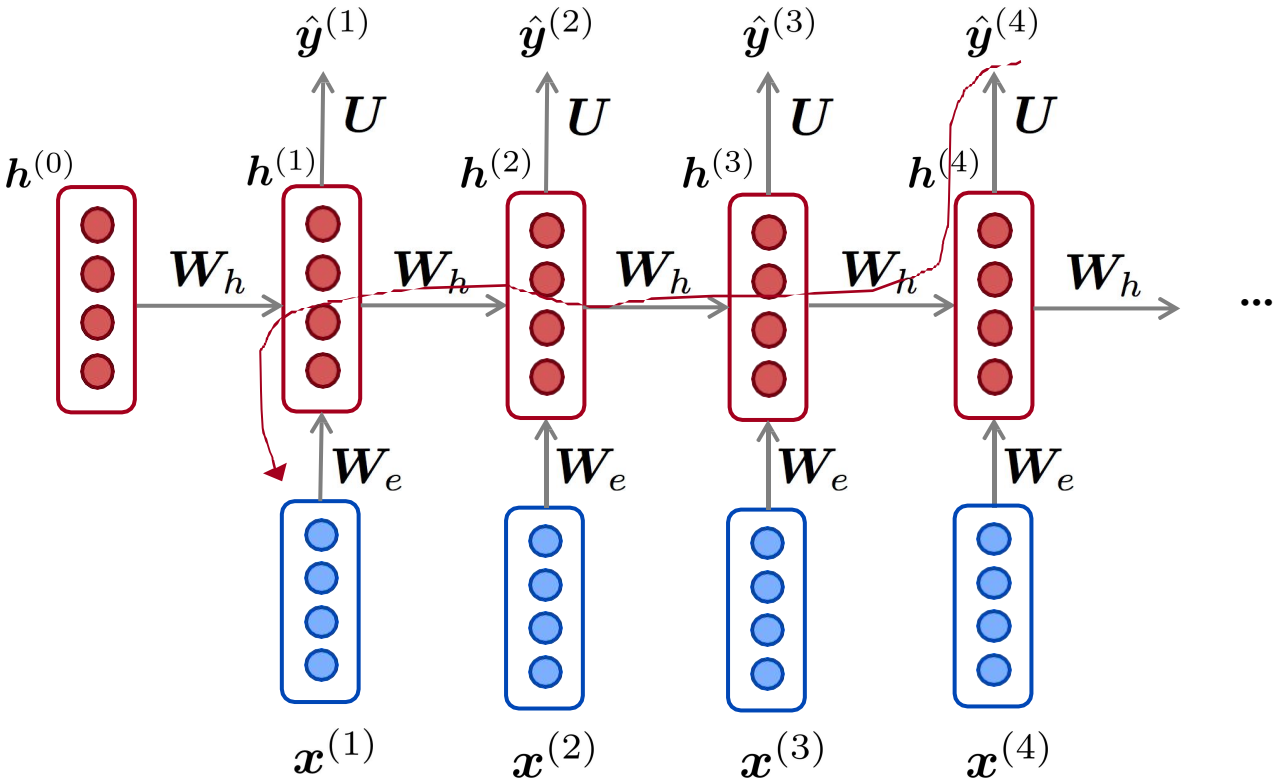
Why is the vanishing gradient a problem?

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 time t+k ?

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?

When we only observe

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\| \leq \eta^{t-k}$$

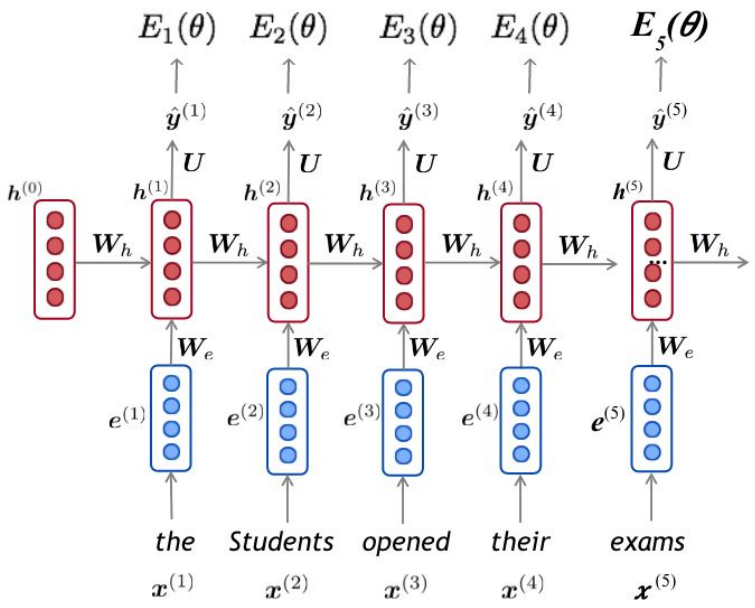
going to 0

We cannot tell whether

1. No dependency between *t* and *k* in data, or
2. Wrong configuration of parameters, e.g. wrong initialization

Why is the vanishing gradient a problem?

1. No dependency between *t* and *t-n* in data



$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right)$$

$$\begin{aligned} \frac{\partial E_5}{\partial \theta} &= \frac{\partial E_4}{\partial \theta} + \frac{\partial E_3}{\partial \theta} + \frac{\partial E_2}{\partial \theta} + \frac{\partial E_1}{\partial \theta} \\ &= \frac{\partial E_4}{\partial \mathbf{h}_5} \frac{\partial \mathbf{h}_5}{\partial \mathbf{h}_4} \frac{\partial \mathbf{h}_4}{\partial \theta} \quad \text{(Chain rule)} \\ &+ \frac{\partial E_3}{\partial \mathbf{h}_5} \frac{\partial \mathbf{h}_5}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \theta} \\ &+ \dots \\ &+ \frac{\partial E_1}{\partial \mathbf{h}_5} \frac{\partial \mathbf{h}_5}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \theta} \end{aligned}$$

When $\frac{\partial \mathbf{h}_5}{\partial \mathbf{h}_1} \to 0$, $\frac{\partial E_1}{\partial \theta} \to 0$

Therefore, no dependency between 5th word and 1st word. More specifically, model fails to get feedback from previous words, e.g. “the”, “students” for context

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 ___ .

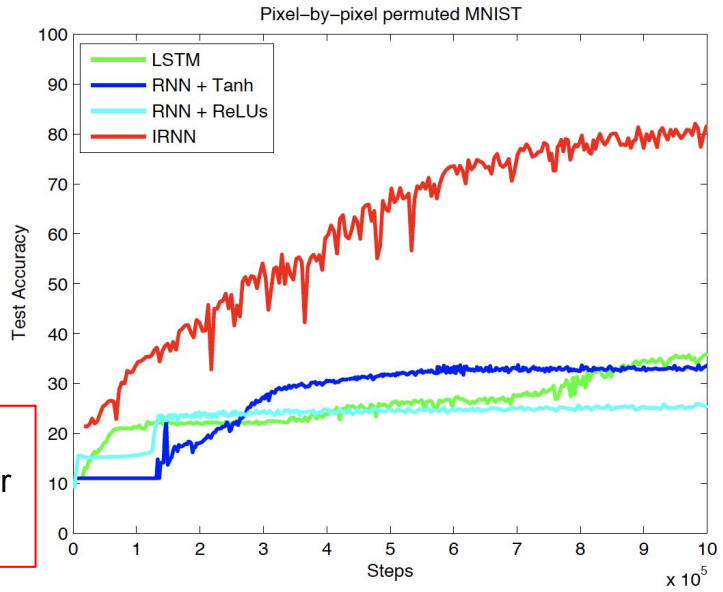
One solution: Initialization + ReLUs

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

- Initialize W(*)'s to *identity matrix I* and f(z) = rect(z) = max(z, 0)

- → Huge Difference!

Activation fn:
rect = Rectified Linear Unit (ReLU)

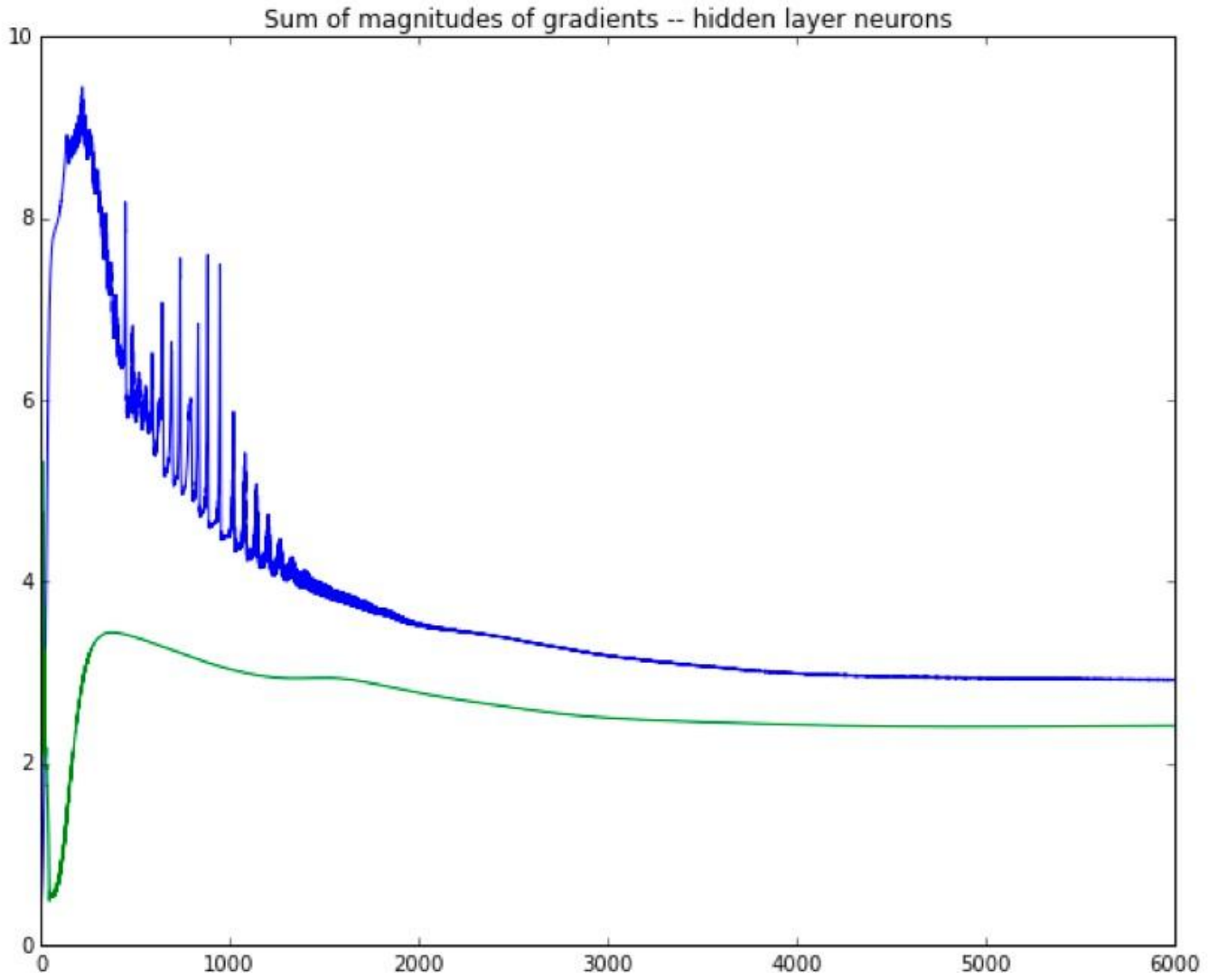


- Initialization idea first introduced in *Parsing with Compositional Vector Grammars*, Socher et al. 2013

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

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.

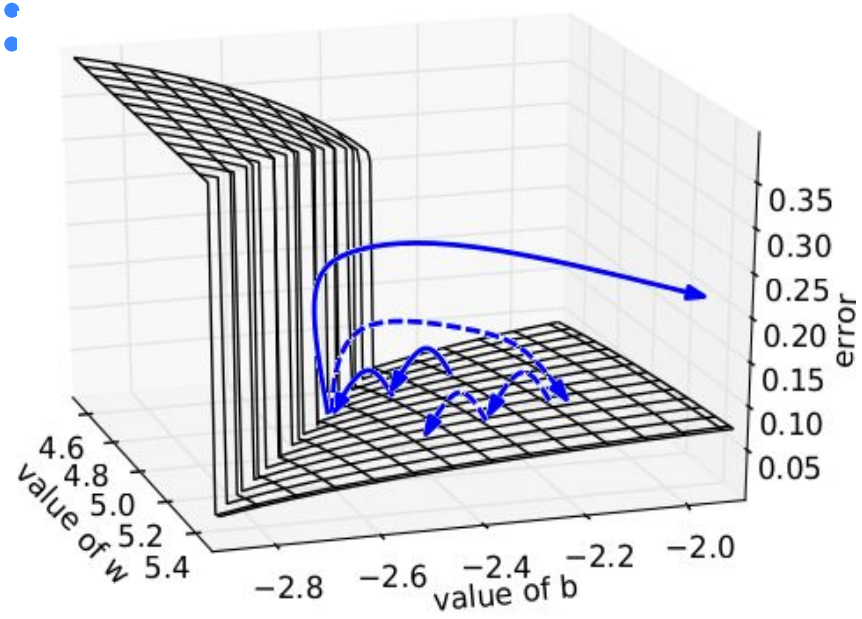
Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
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
- There's another method of clipping *absolute* value for gradient.

Exploding gradient: Gradient clipping intuition

$$x_t = w\sigma(x_{t-1}) + b$$



- One hidden unit RNN model
- Given initial state x0 = 0.5 → Train for

a specific target value after 50 steps

- Gradient explodes → High curvature walls
- Solid lines: standard gradient descent trajectories
- Dashed lines: gradients rescaled to fixed size

Recap
Parameter update in SGD:

$$w := w - \eta \nabla Q(w)$$

"On the difficulty of training Recurrent Neural Networks", Pascanu et al. 2013

Summary for problems with training RNNs

	Vanishing gradient	**Exploding gradient**
What gives rise?	$\left\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\right\| \to 0$ Gradients of many time steps ago approaching 0	$\left\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\right\| > 1$ Gradients of many time steps ago getting very large
How to fix?	1. Good initialization (Identity matrix) 2. ReLU 3. Hessian-Free Optimization [Martens, 2010] (not covered in this lecture)	1. Gradient Clipping (Absolute Clipping or Norm Clipping) 2. ReLU 3. Weight Regularization 4. Structural damping [Sutskever, 2011] (not covered in this lecture)
Can we do more? Yes, we can! By using different architectures	1. LSTM [Hochreiter, Schmidhuber, 1999] 2. GRU [Cho, 2014] 3. Residual Net [He, 2015] (not covered in this lecture)	1. LSTM

Reference for papers on the next slide

References

1. Good initialization + ReLU
   (*A Simple Way to Initialize Recurrent Networks of Rectified Linear Units*, Le et al. 2015)
2. Hessian-Free Optimization
   (*Deep learning via Hessian-free optimization*, J. Martens, 2010)
3. Absolute Gradient Clipping
   (*Tomas Mikolov PhD Thesis*, Mikolov, 2012)
4. Norm Clipping
   (*On the difficulty of training Recurrent Neural Networks*, Pascanu et al, 2013)
5. Hessian-Free + Structural Damping
   (*Generating text with recurrent neural networks*, Sutskever et al, 2011)
6. LSTM
   (*Long short-term memory*, Hochreiter et al, 1997)
7. GRU
   (*On the properties of neural machine translation: Encoder-decoder approaches*, Cho, 2014)
8. Residual Network (ResNet)
   (*Deep Residual Learning for Image Recognition*, He, 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

Gated Recurrent Units (GRU)

GRUs

Goal:

- “Memorize” long distance dependencies

- Allow error messages to flow back to faraway timesteps (solve vanishing gradient problem)

GRUs

- Standard RNN computes hidden layer at next time step: $$h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$$
- GRU first computes an update **gate** (another layer with dimension (d_h,n)) 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

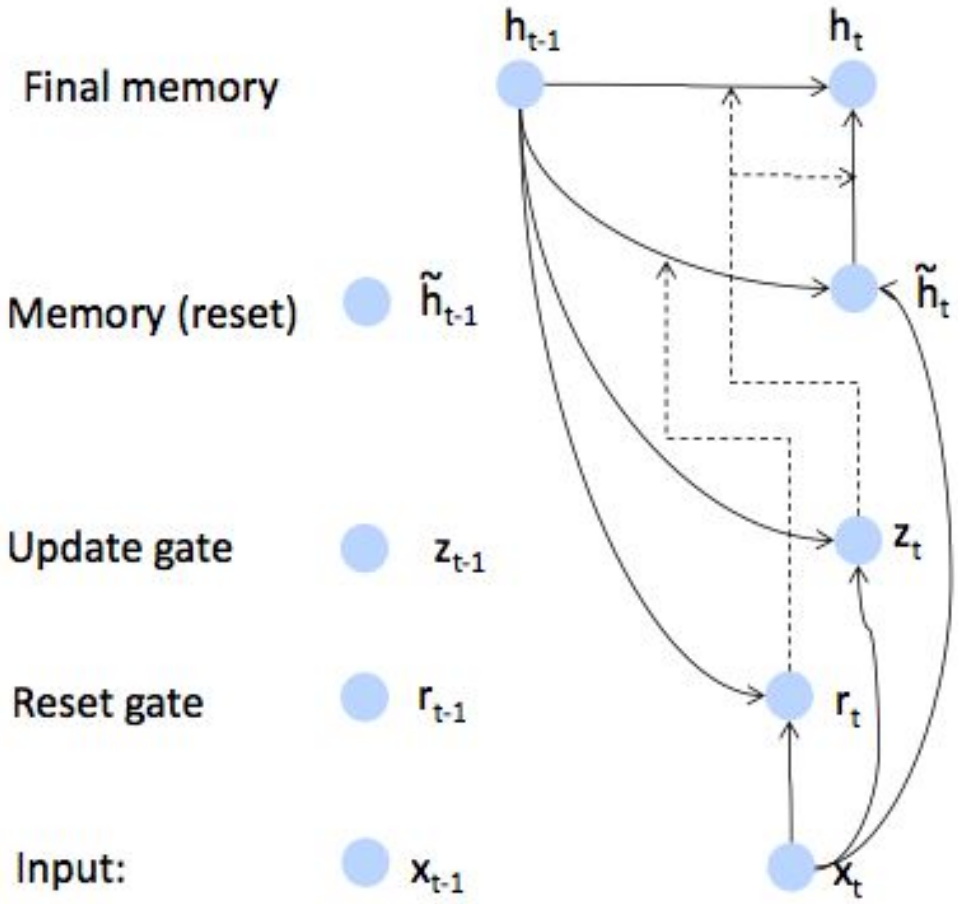
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

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



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

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

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

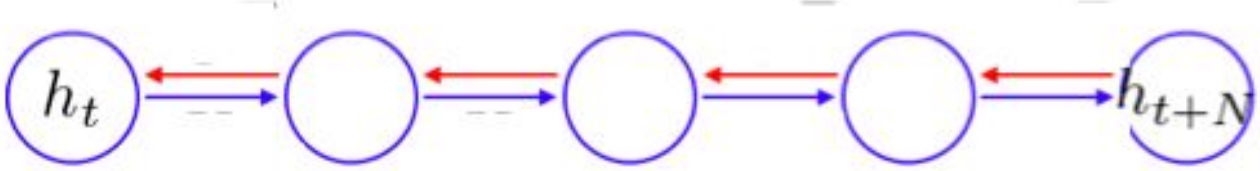
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

How do Gated Recurrent Units fix vanishing gradient problems?

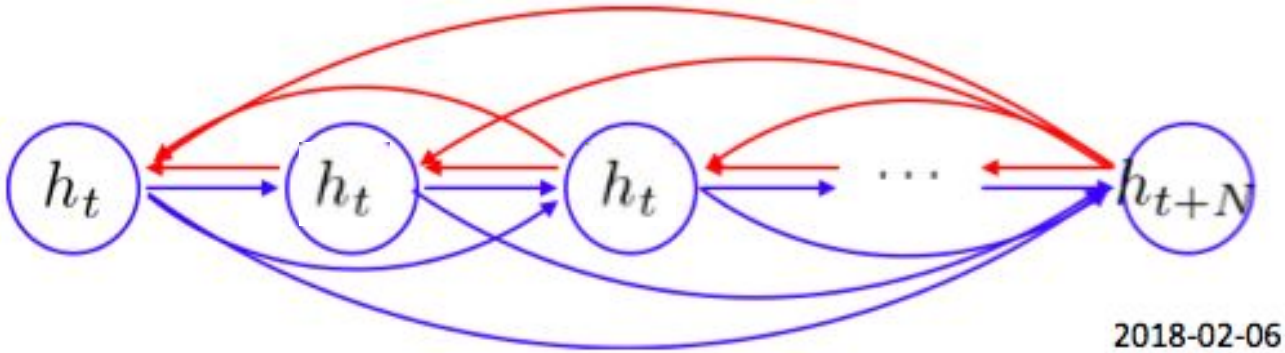
- Is the problem with standard RNNs the naïve transition function?

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

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



- Perhaps we can create shortcut connections.



GRU intuition

- Update gate z controls how much of previous memory should matter now.
  - If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient**!

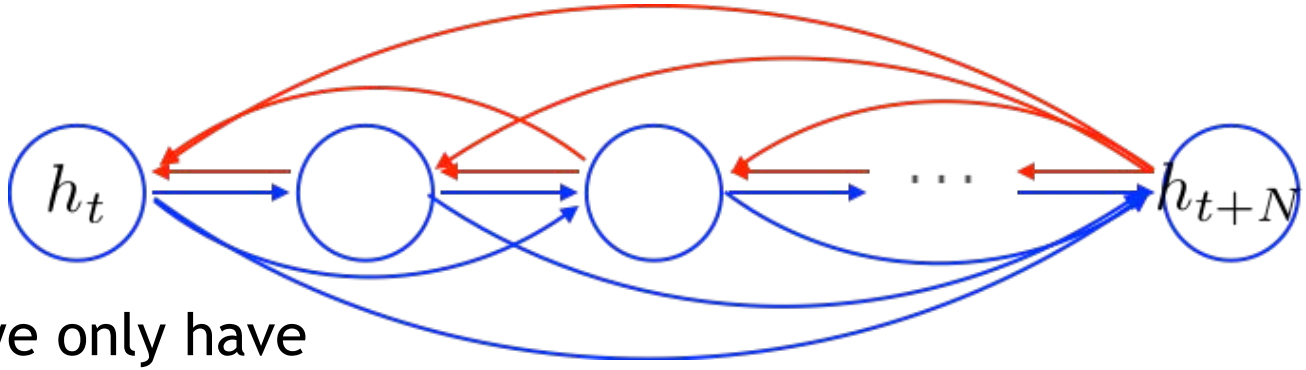
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$

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

Then why do we still need the reset gate?

How do Gated Recurrent Units fix vanishing gradient problems?

- We can learn *adaptive* shortcut connections. (update gate)
- Let the net prune unnecessary connections *adaptively*. (reset gate)



- Suppose we only have update gate, we can't clear our memory of the past
- That's what the gates do. And that's why we want two gates.

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

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

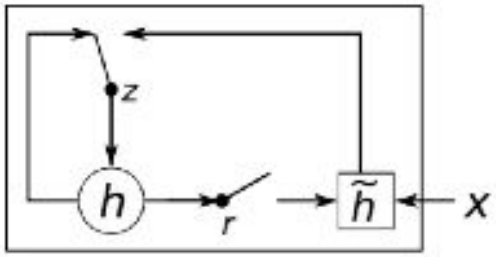
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$

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

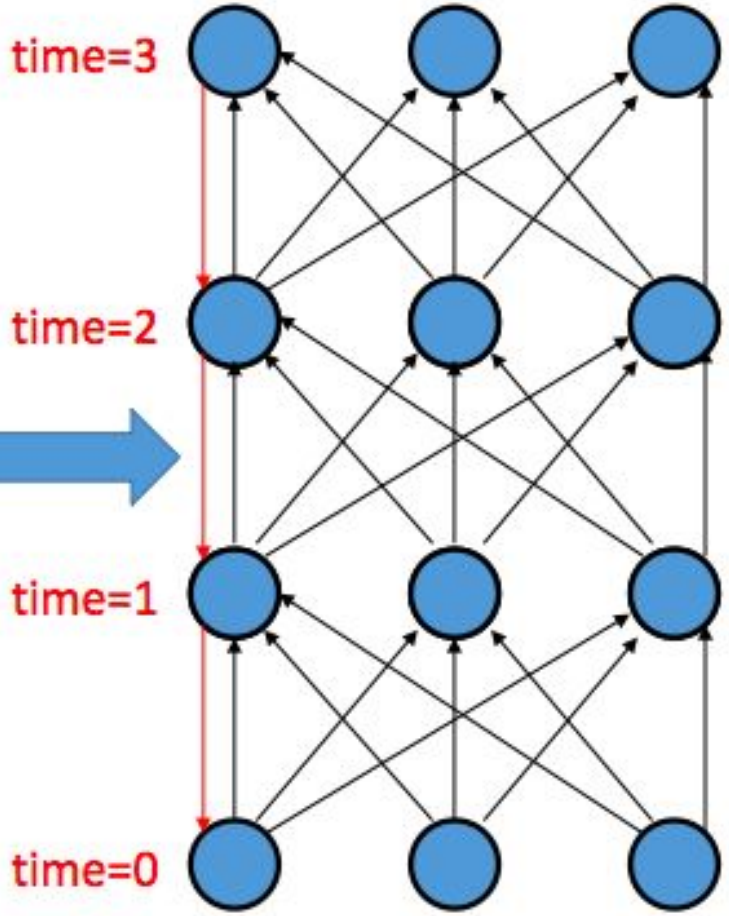
Why do GRUs help with the vanishing gradient problem?

- *We had:*
  - $\frac{\partial J^{(t)}}{\partial W} = \sum_{k=1}^{t} \frac{\partial J^{(t)}}{\partial y_t} \frac{\partial y_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial J^{(t)}}{\partial \hat{y}_k} \frac{\hat{y}_k}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$
  - $\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \leq \alpha^{t-j-1}$
- *Now:*
  - $\frac{\partial h_j}{\partial h_{j-1}} = z_j + (1 - z_j) \frac{\partial \tilde{h}_j}{\partial h_{j-1}}$
  - $\frac{\partial h_j}{\partial h_{j-1}}$ is 1 for $z_j = 1$

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



$z_j = 1$ ➡ ignore



"Shutting" the update gate lets us essentially "skip" layers when calculating the gradient.

This ameliorates the vanishing, exploding gradient problem.

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

Performance Comparison

			tanh	GRU	LSTM
Music Datasets	Nottingham	train	3.22	2.79	3.08
		test	**3.13**	3.23	3.20
	JSB Chorales	train	8.82	6.94	8.15
		test	9.10	**8.54**	8.67
	MuseData	train	5.64	5.06	5.18
		test	6.23	**5.99**	6.23
	Piano-midi	train	5.64	4.93	6.49
		test	9.03	**8.82**	9.03
Ubisoft Datasets	Ubisoft dataset A	train	6.29	2.31	1.44
		test	6.44	3.59	**2.70**
	Ubisoft dataset B	train	7.61	0.38	0.80
		test	7.62	**0.88**	1.26

Table 2: The average negative log-probabilities of the training and test sets.

from: Chung et al., https://arxiv.org/pdf/1412.3555.pdf

Performance Comparison

Sentiment Analysis Task

- **Predict sentiment (positive/negative)of IMDB movie review**
- **training : 20000, validation: 5000**
- **vocab size: 10000**
- **hidden unit dimension: 24**
- **word embedding dimension: 16**
- **all models trained on GTX 1080**
- **dataset from:** http://ai.stanford.edu/~amaas/data/sentiment/

Performance Comparison

RNN Structure	validation accuracy (%)	training time per epoch
vanilla RNN	83.74	12s
GRU	87.26	40s
bidirectional GRU	87.32	82s

Source code (notebook):
https://github.com/NoviScl/DeepLearning/blob/master/tensorflow_tutorials/GRU_IMDB.ipynb

Variants of GRU

- Jozefowicz et al. (2015) conducted a thorough architecture search where they evaluated over ten thousand different RNN architectures.

- They proposed three variants of GRU. These variants sometimes achieve higher accuracy than GRU on some tasks, but none of them can consistently outperform GRU.
http://proceedings.mlr.press/v37/jozefowicz15.pdf

MUT1:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + b_{\text{z}}) \\ r &= \text{sigm}(W_{\text{xr}}x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + \tanh(x_t) + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}h_t + b_{\text{z}}) \\ r &= \text{sigm}(x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh}x_t + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}\tanh(h_t) + b_{\text{z}}) \\ r &= \text{sigm}(W_{\text{xr}}x_t + W_{\text{hr}}h_t + b_{\text{r}}) \\ h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh}x_t + b_{\text{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

Variants of GRU

Minimal Gated Unit (same gate for update and reset)

GRU (Gated Recurrent Unit)

$$\boldsymbol{z}_t = \sigma\left(W_z\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_z\right), \tag{5a}$$

$$\boldsymbol{r}_t = \sigma\left(W_r\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_r\right), \tag{5b}$$

$$\tilde{\boldsymbol{h}}_t = \tanh\left(W_h\left[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_h\right), \tag{5c}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t. \tag{5d}$$

MGU (Minimal Gated Unit, the proposed method)

$$\boldsymbol{f}_t = \sigma\left(W_f\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_f\right), \tag{6a}$$

$$\tilde{\boldsymbol{h}}_t = \tanh\left(W_h\left[\boldsymbol{f}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_h\right), \tag{6b}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{f}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{f}_t \odot \tilde{\boldsymbol{h}}_t. \tag{6c}$$

(Zhou et al., 2016) https://arxiv.org/pdf/1603.09420.pdf

Variants of GRU

Performance of MGU

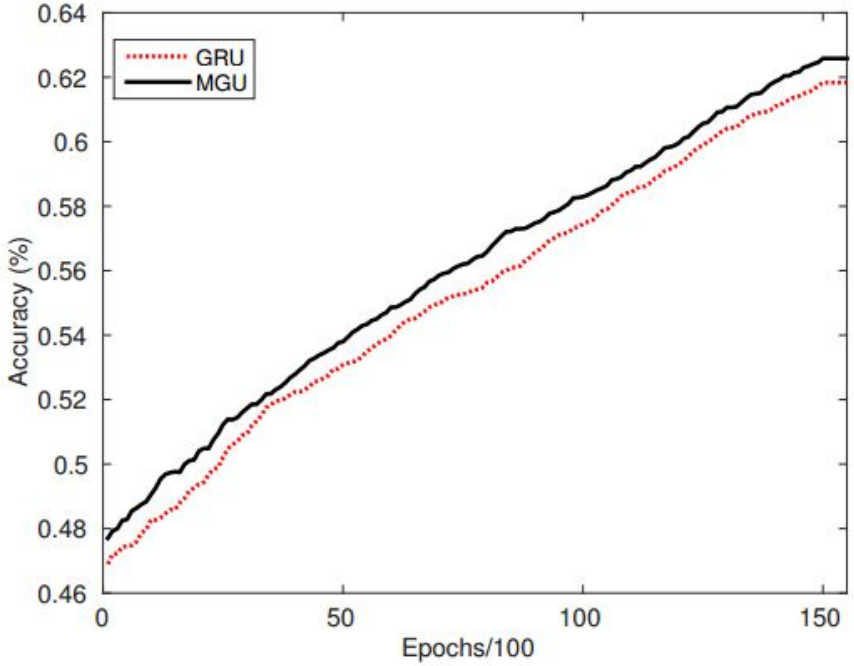


Figure 3. Test set classification accuracy comparison (MGU vs. GRU) on the IMDB dataset. Higher is better.

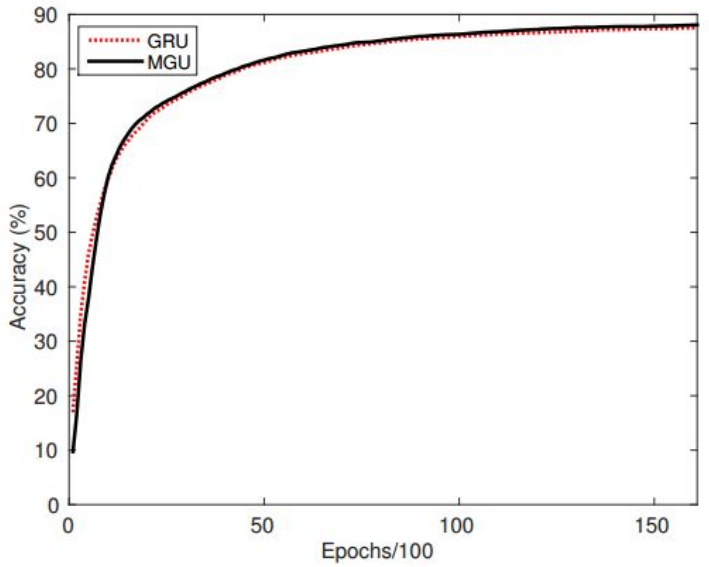


Figure 4. Test set classification accuracy comparison (MGU vs. GRU) on the MNIST dataset. This is the first task, where the sequence length is 28. Higher is better.

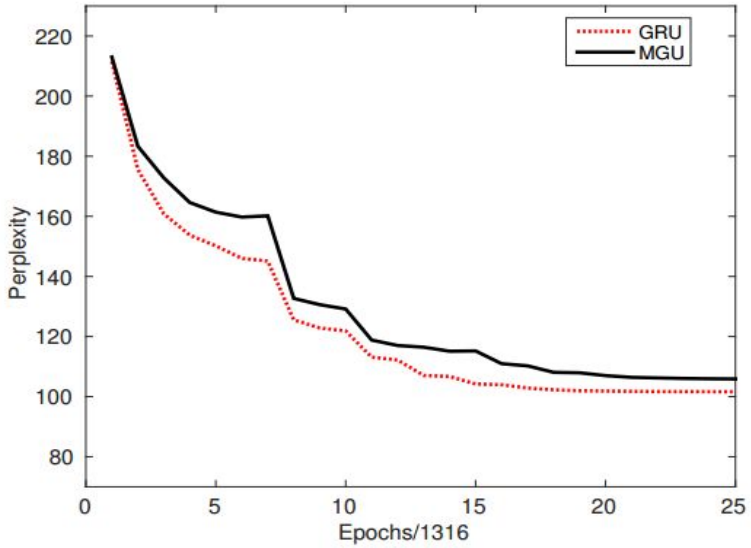


Figure 5. Test set perplexity comparison (MGU vs. GRU with 500 hidden units) on the PTB dataset. Lower is better.

Backprop for GRU

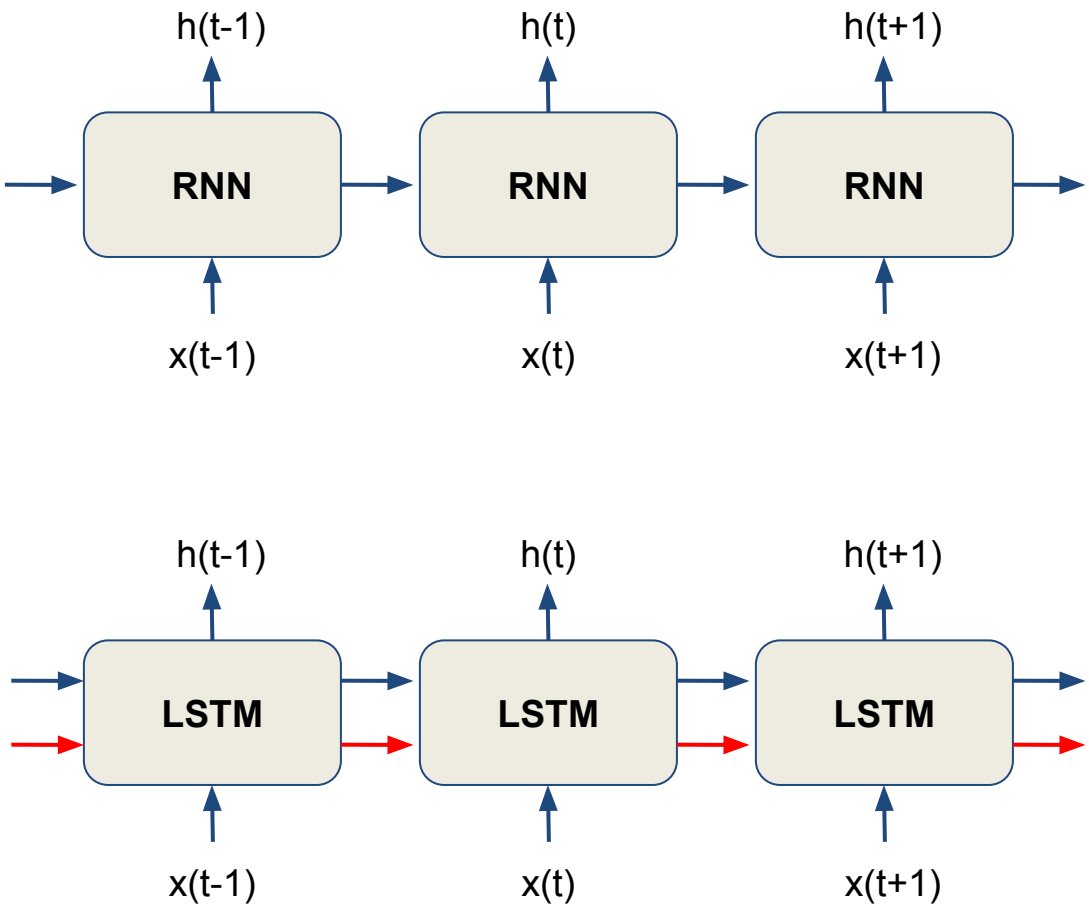
Please refer to:
https://medium.com/swlh/only-numpy-deriving-forward-feed-and-back-propagation-in-gated-recurrent-neural-networks-gru-8b6810f91bad

Long-Short-Term-Memory (LSTMS)

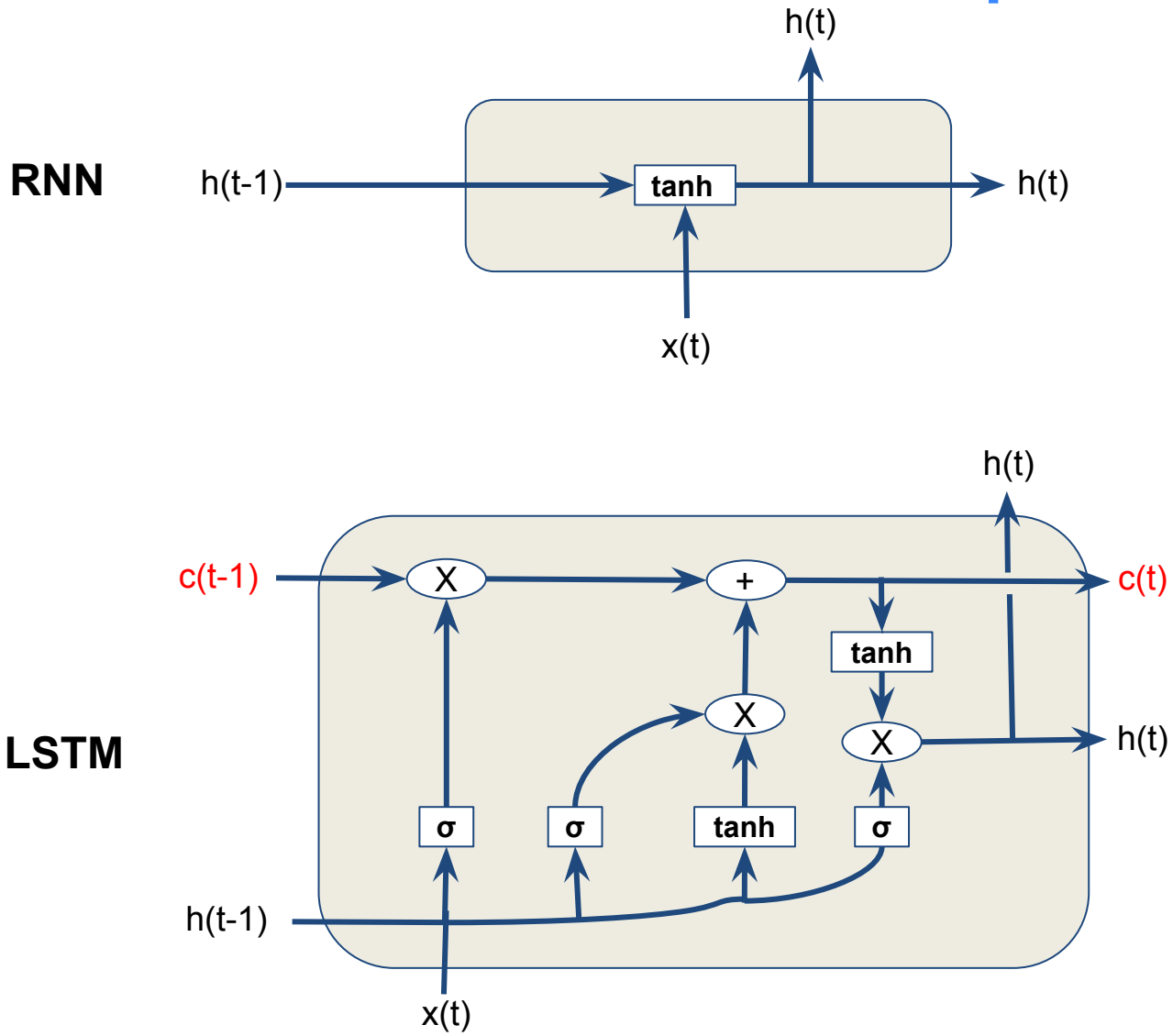
Some history on LSTMs

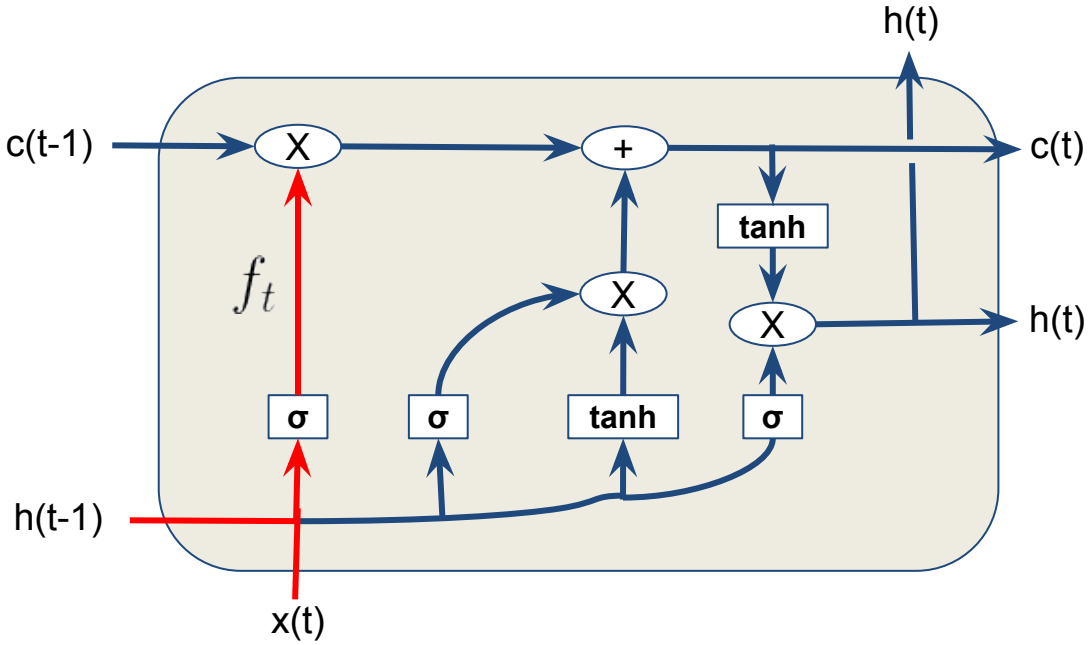
- First introduced by Dr.Hochreiter and Dr. Schmidhurber in the paper named “Long Short Term Memory”
- GRU is still type of LSTM (less complex)
- LSTMs perform really good in NLP, speech and video recognition
- LSTM unit consists of **cell**, **input**, **output** and **forget** gates

LSTM vs RNN: *big* picture



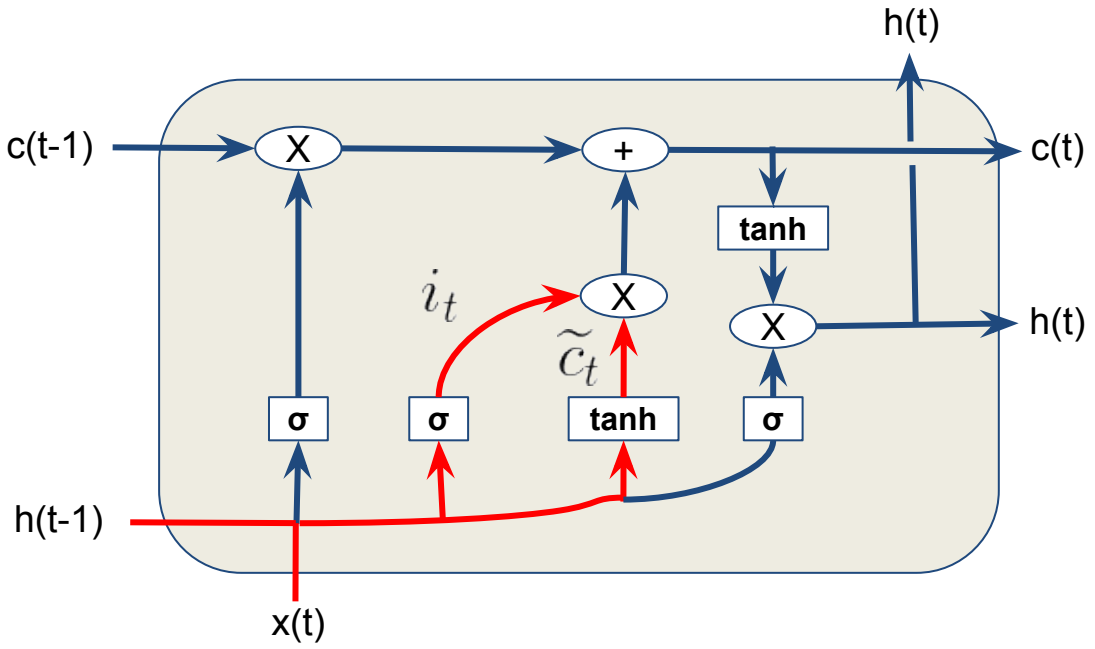
LSTM vs RNN: *inside* picture





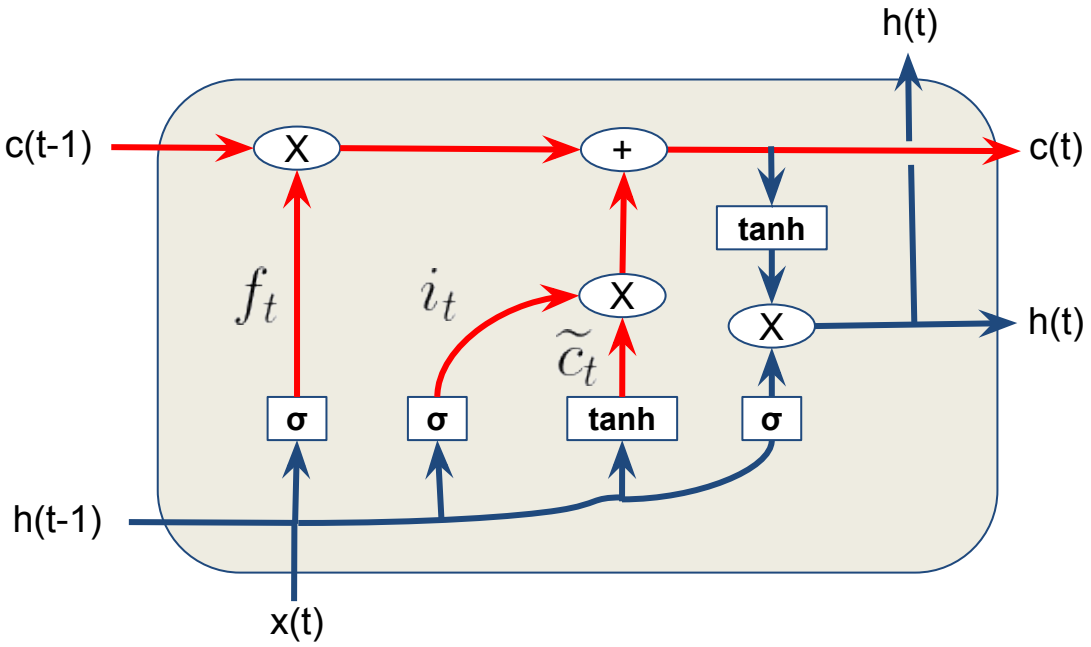
Forget gate: $f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$

Intuition: we want to *forget* some *information* about the subject, if *new information* describes it. Ex: two people talking about their cars with different colors: when one of them speaks, we want to forget about other person's car's color



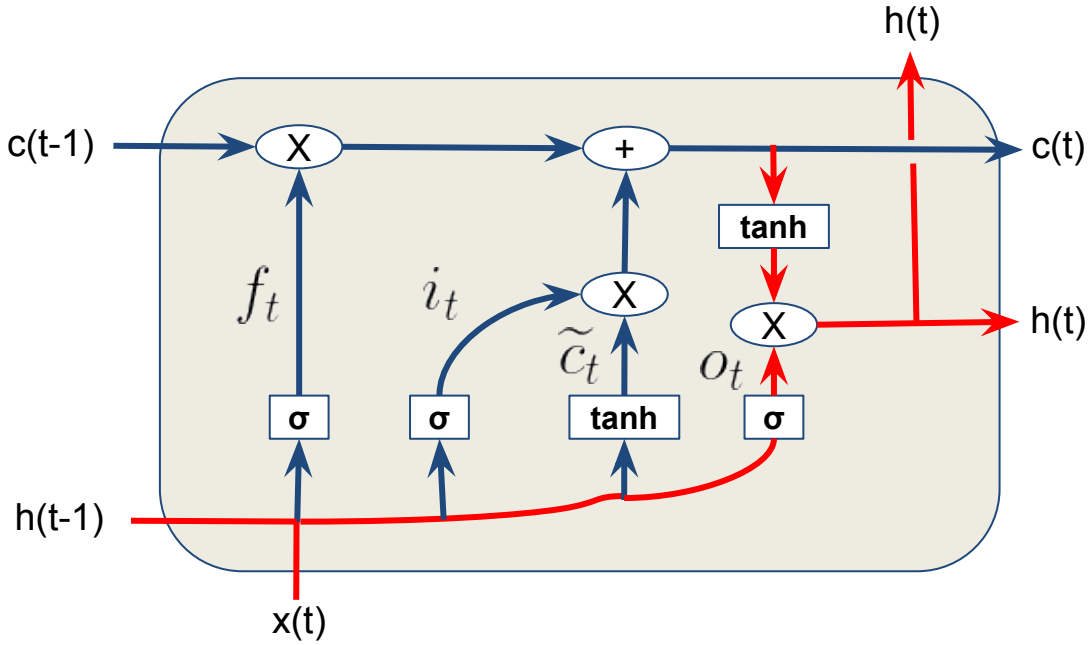
Input gate: $i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$

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



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

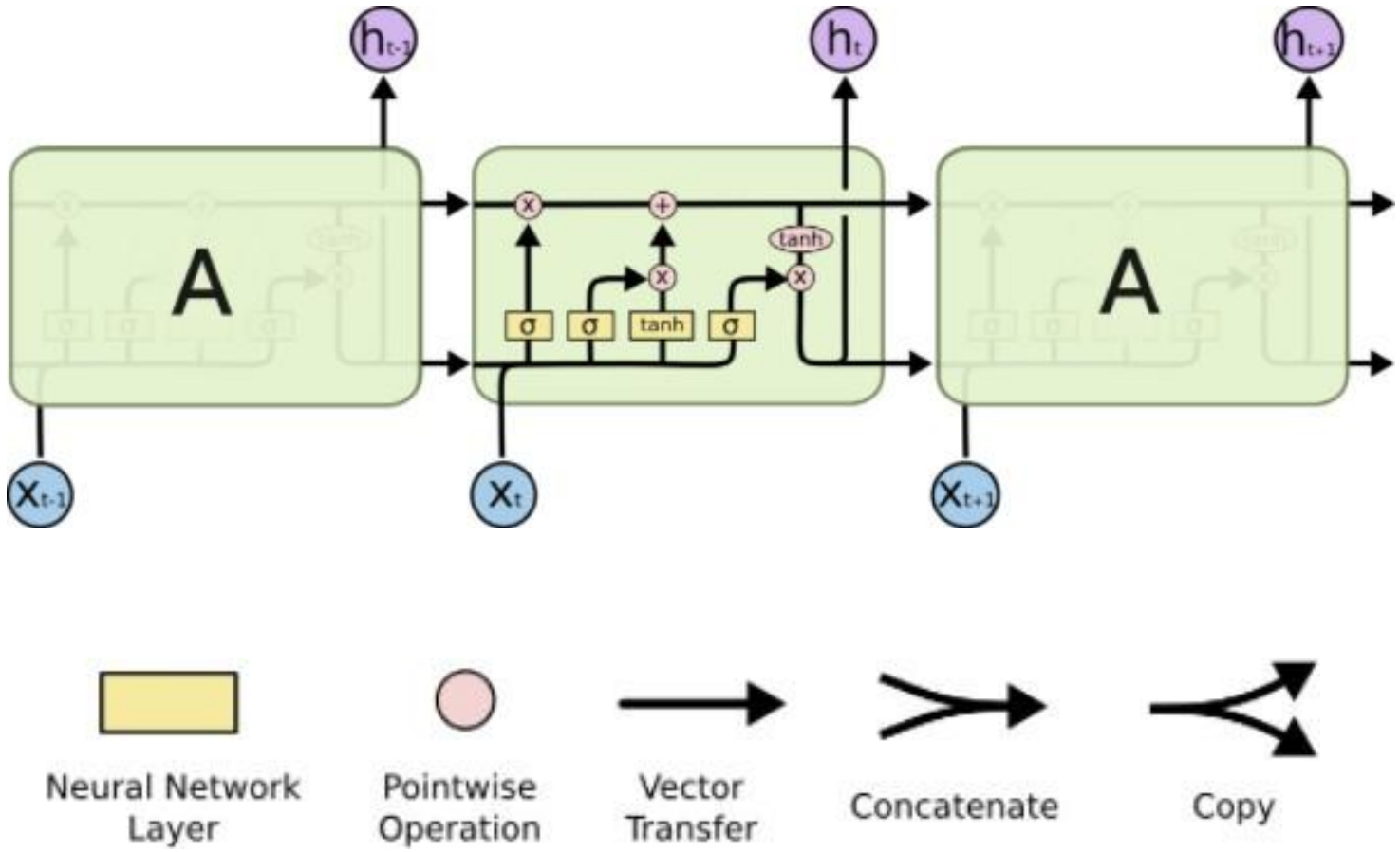
This part is the the secret! (Of other recent things like ResNets too!) Rather than multiplying, we get c_t by adding the non-linear stuff and c_{t-1} ! There is a direct, linear connection between c_t and c_{t-1}.



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

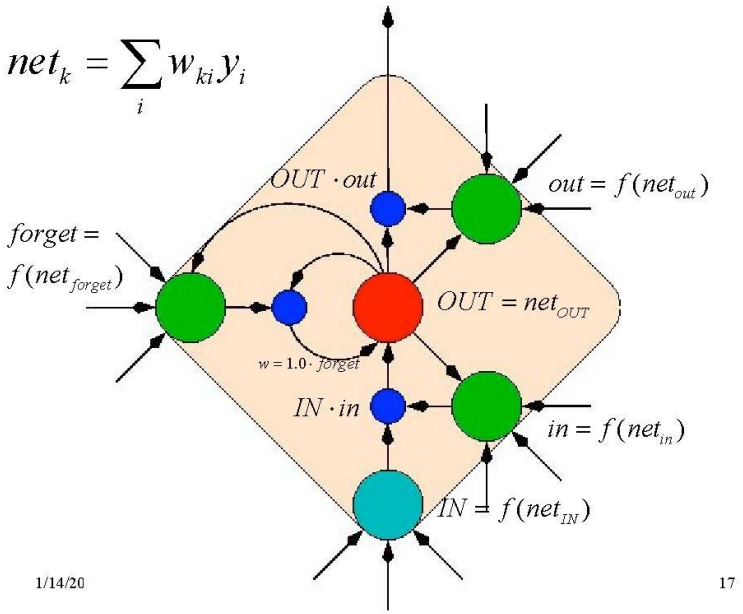
Hidden state: $h_t = o_t \circ \tanh(c_t)$

Some visualizations

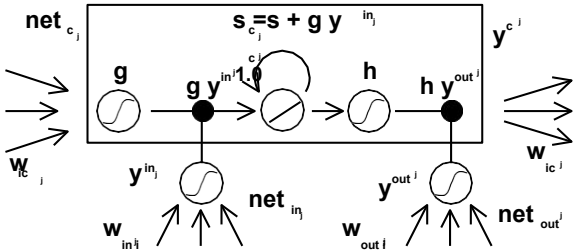


By Chris Olah:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

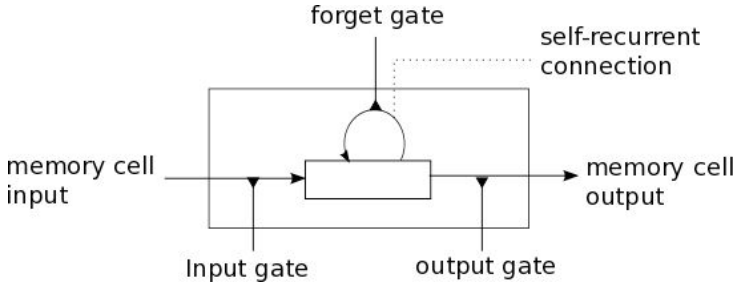
Most illustrations a bit overwhelming ;)



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



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

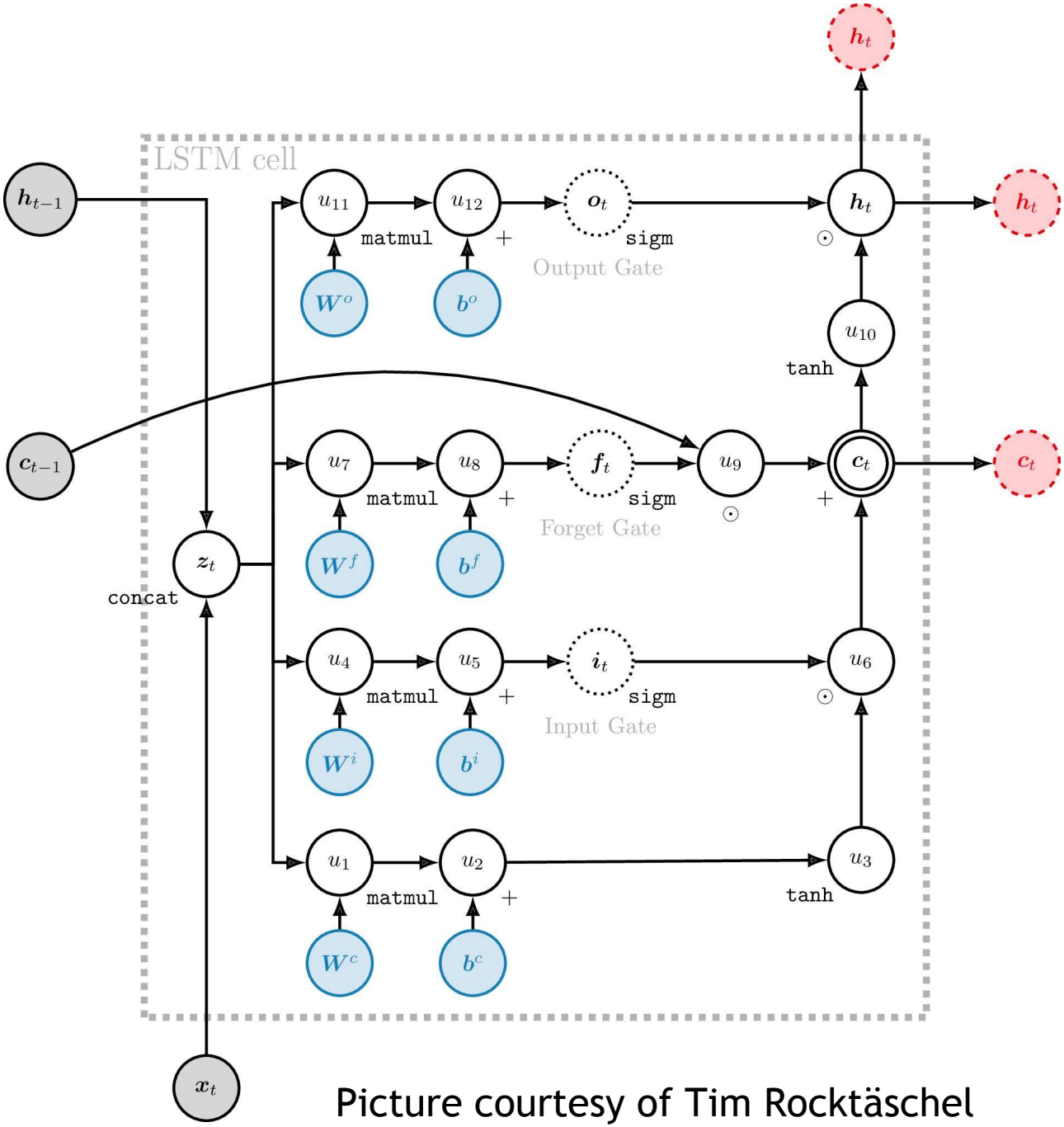


http://deeplearning.net/tutorial/lstm.html

Intuition: memory cells can keep information intact, unless inputs makes them forget it or overwrite it with new input.

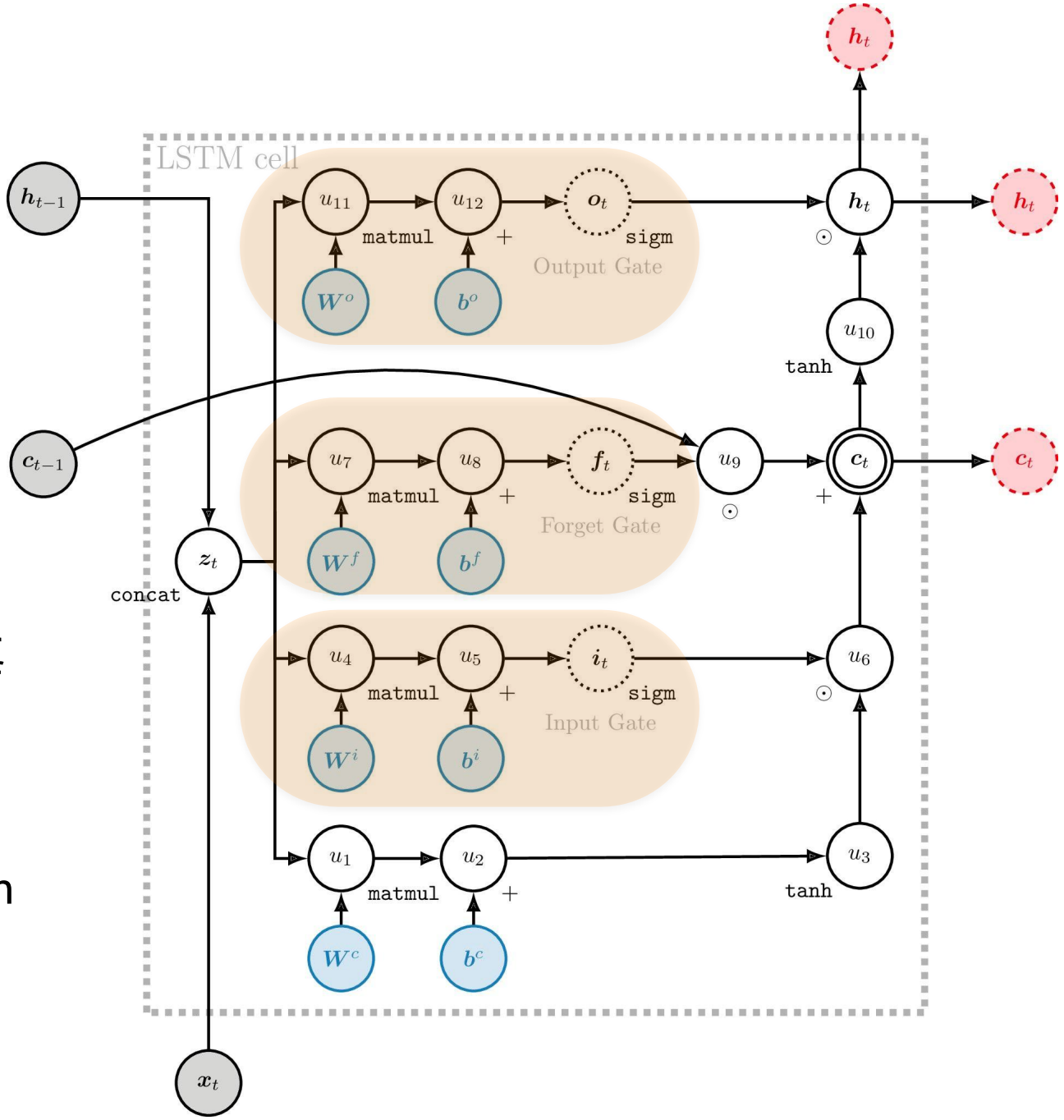
Cell can decide to output this information or just store it

Another LSTM visualization inspired by code



Picture courtesy of Tim Rocktäschel

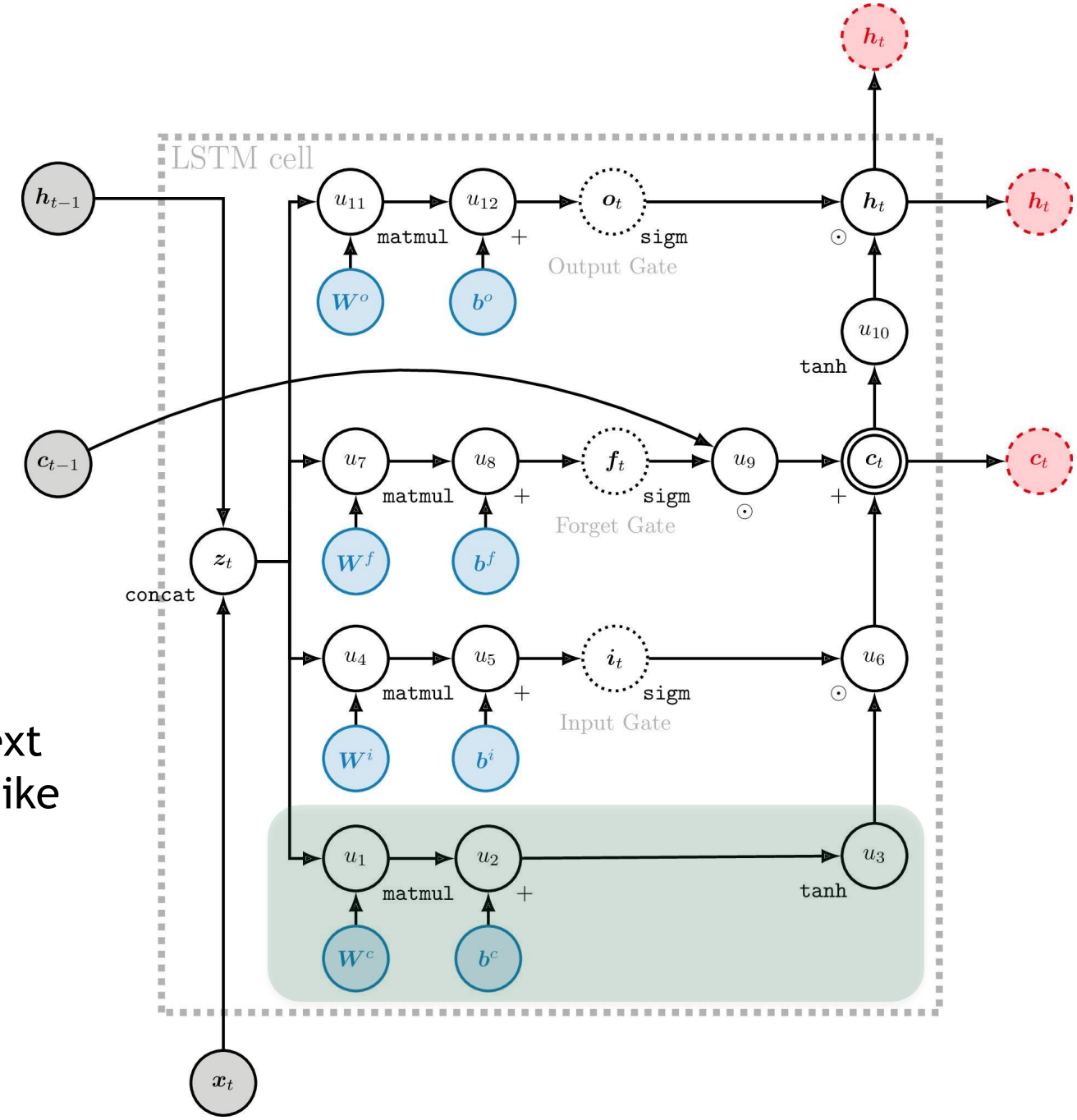
The LSTM



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



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	~45

Sequence to Sequence Learning by Sutskever et al. 2014

Deep LSTMs (with a lot more tweaks)

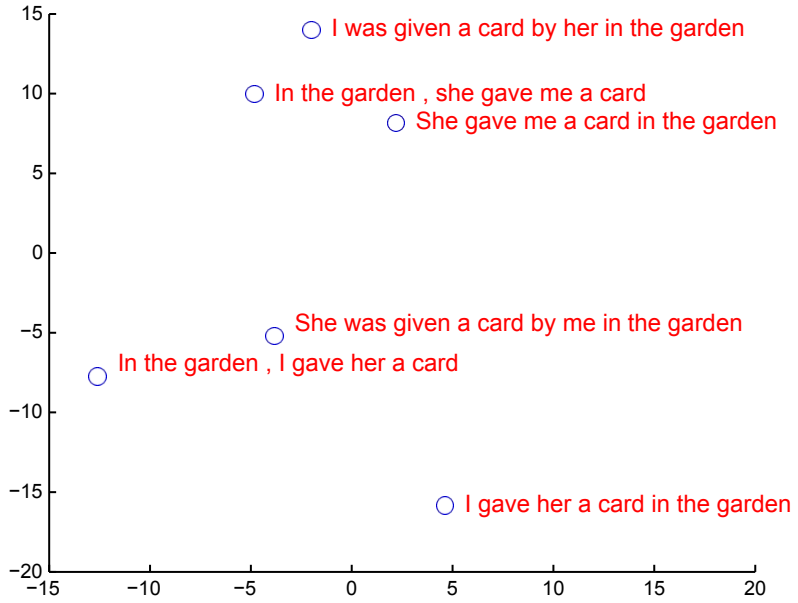
WMT 2016 competition results

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
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

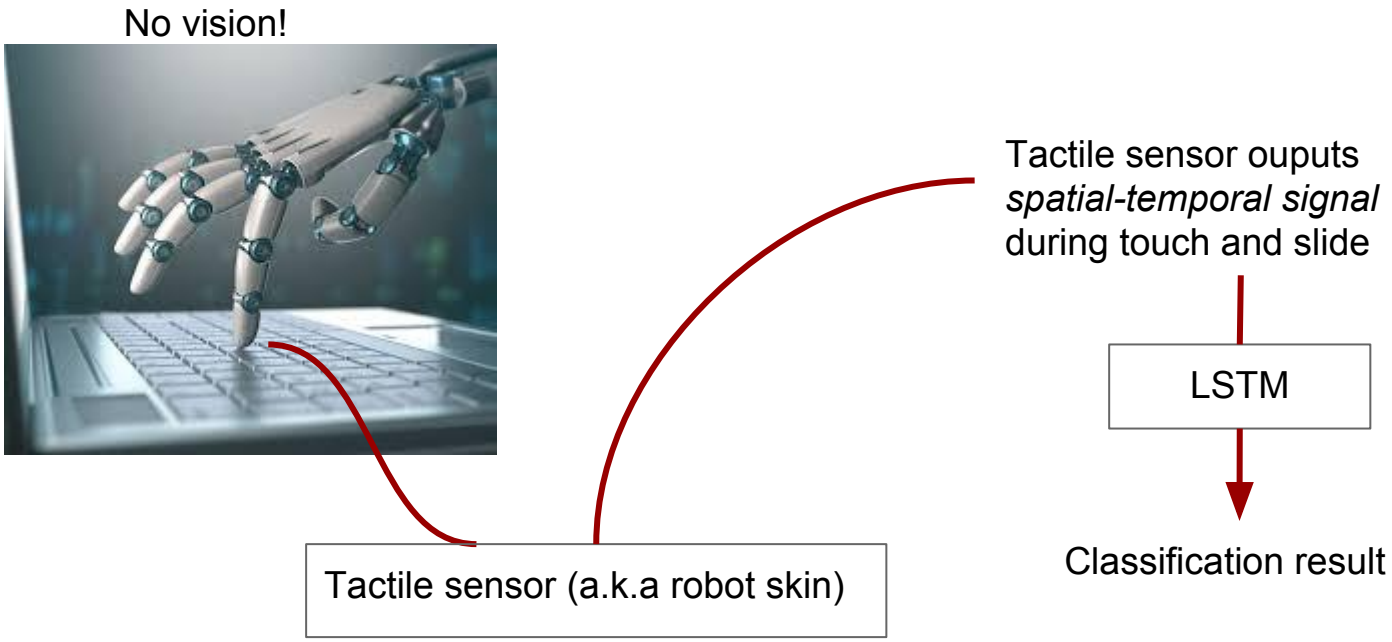
PCA of vectors from last time step hidden layer



Sequence to Sequence Learning by Sutskever et al. 2014

Example LSTM in Robotics application

- **Robots can learn about environment through LSTM**



Bidirectional RNN

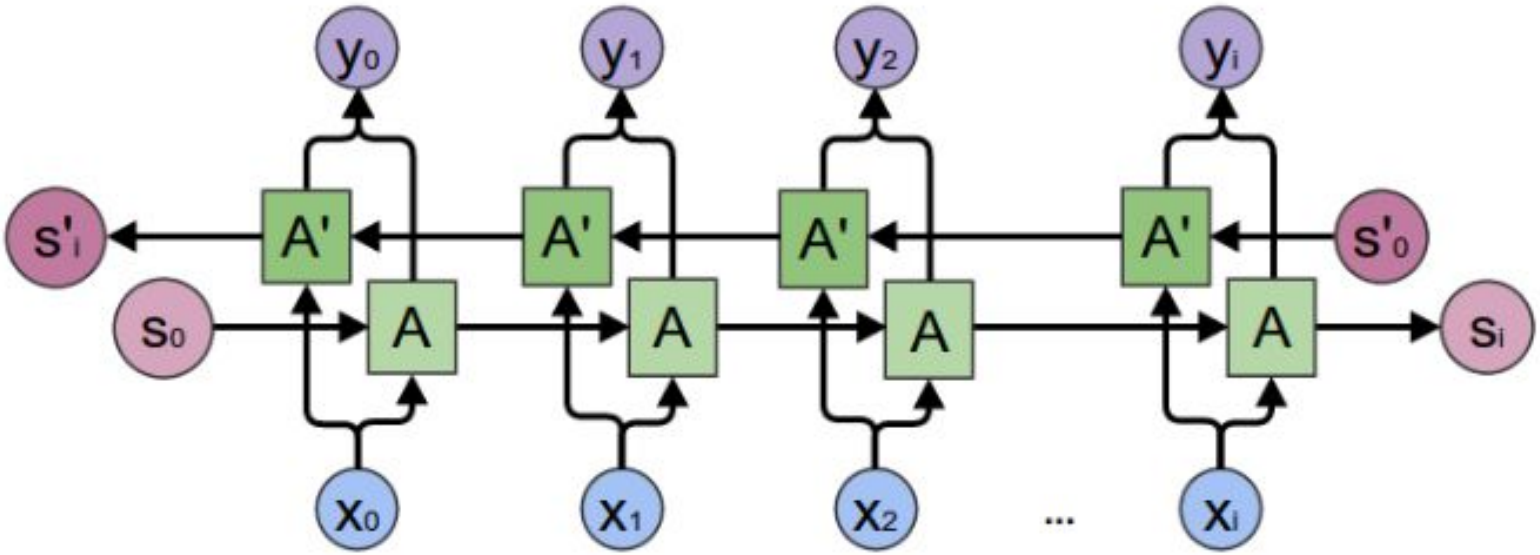
Problem:For classification you want to incorporate information from words both preceding and following

For example:
“He said, Teddy bears are on sale”
“He said, Teddy Roosevelt was a great President”

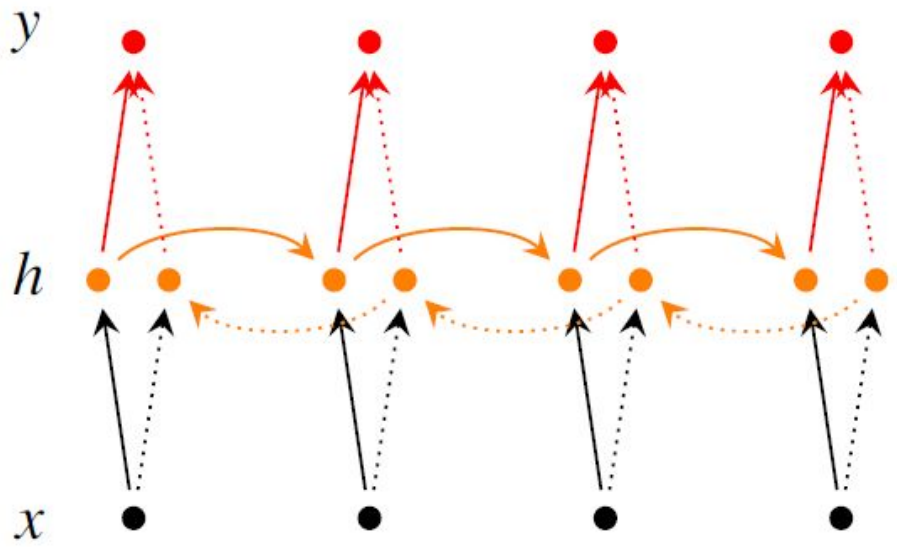
Bidirectional RNN

Two type of connections:

1) One going forward in time, which helps us learn from previous representations
2) Another going backward in time, which helps us learn from future representations



Bidirectional RNN



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

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

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

- $h = [\vec{h}; \overleftarrow{h}]$ represents the past and future around a single token
- Weights will be reused
- The repeating module in a Bidirectional RNN could be a conventional RNN, LSTM or GRU

Bidirectional RNN

Forward Pass

for $t = 1$ to T **do**
  Forward pass for the forward hidden layer, storing activations at each timestep
for $t = T$ to 1 **do**
  Forward pass for the backward hidden layer, storing activations at each timestep
for all t, in any order **do**
  Forward pass for the output layer, using the stored activations from both hidden layers

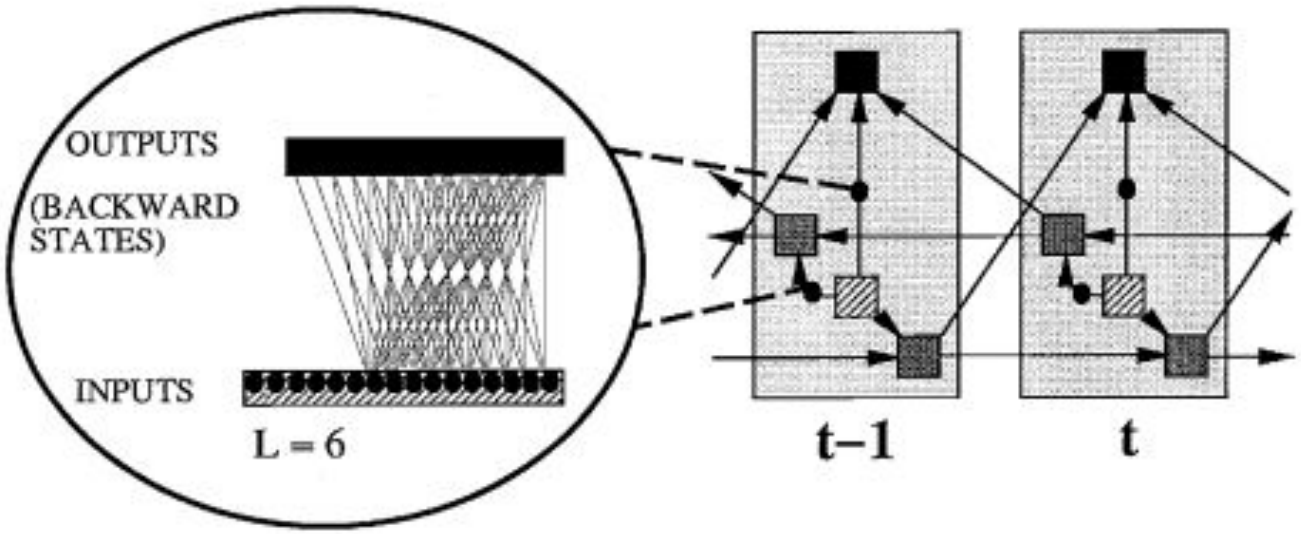
Bidirectional RNN

Backward Pass

for all t, in any order **do**
  Backward pass for the output layer, storing δ terms at each timestep
for $t = T$ to 1 **do**
  BPTT backward pass for the forward hidden layer, using the stored δ terms from the output layer
for $t = 1$ to T **do**
  BPTT backward pass for the backward hidden layer, using the stored δ terms from the output layer

Bidirectional RNN

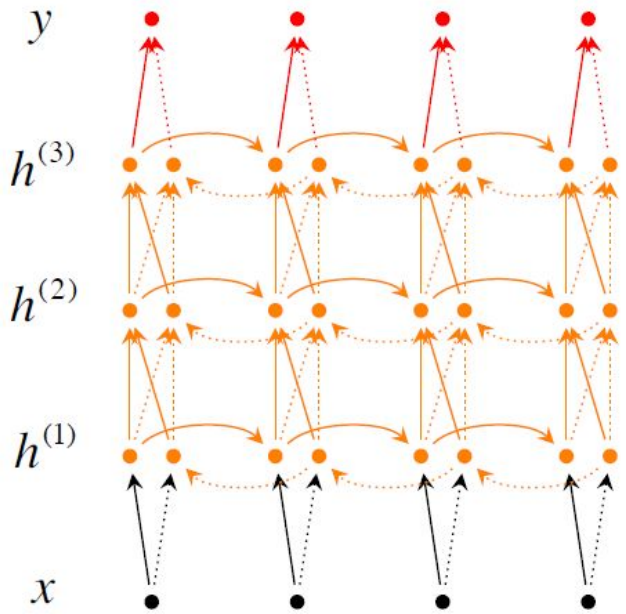
A modified Bidirectional RNN



Bidirectional Recurrent Neural Networks

Mike Schuster and Kuldip K. Paliwal, Member, IEEE, 1997

Deep Bidirectional RNN



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

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

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

Each intermediate neuron receives three sets of parameters from

- Previous time-step (in the same RNN layer)
- left-to-right RNN (previous RNN hidden layer)
- right-to-left RNN (previous RNN hidden layer)

Thank You!