Natural Language Processing with Deep Learning
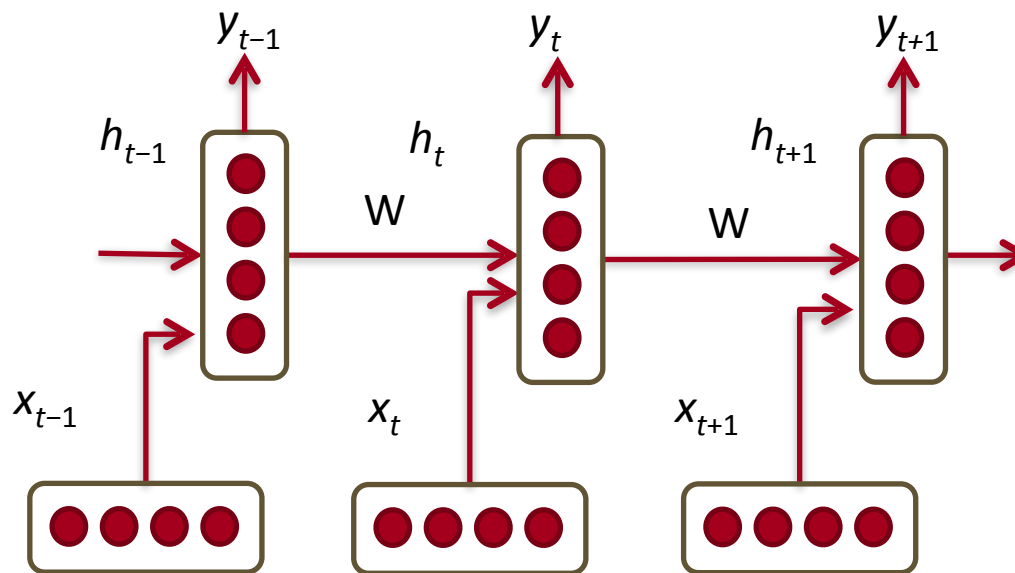# CS224N/Ling284

Lecture 8:
Recurrent Neural Networks

Christopher Manning and Richard Socher

Recurrent Neural Networks!

- RNNs tie the weights at each time step

- Condition the neural network on all previous words
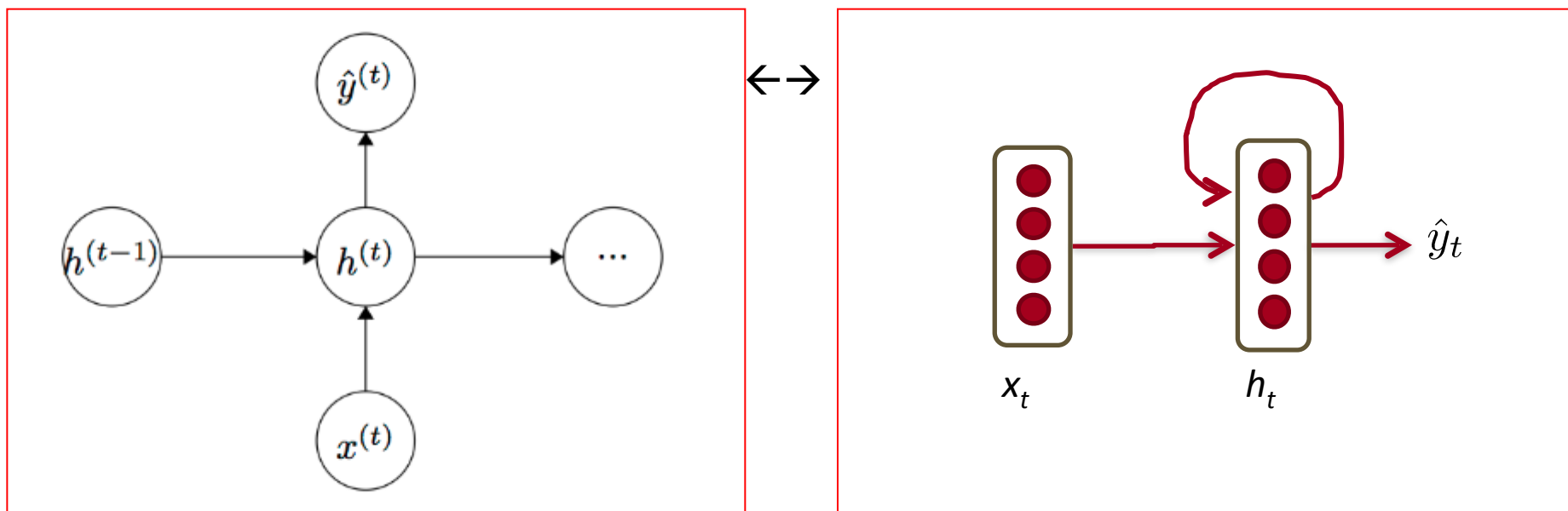
- RAM requirement only scales with number of words

2/2/17

Recurrent Neural Network language model

Given list of word **vectors**: $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$

At a single time step:

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

$$\hat{y}_t = \mathrm{softmax}\left(W^{(S)} h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$

$\leftrightarrow$

Recurrent Neural Network language model

Main idea: <mark>we use the same set of W weights at all time steps!</mark>

Everything else is the same:

$$h_t = \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right)$$

$$\hat{y}_t = \text{softmax}\left(W^{(S)}h_t\right)$$

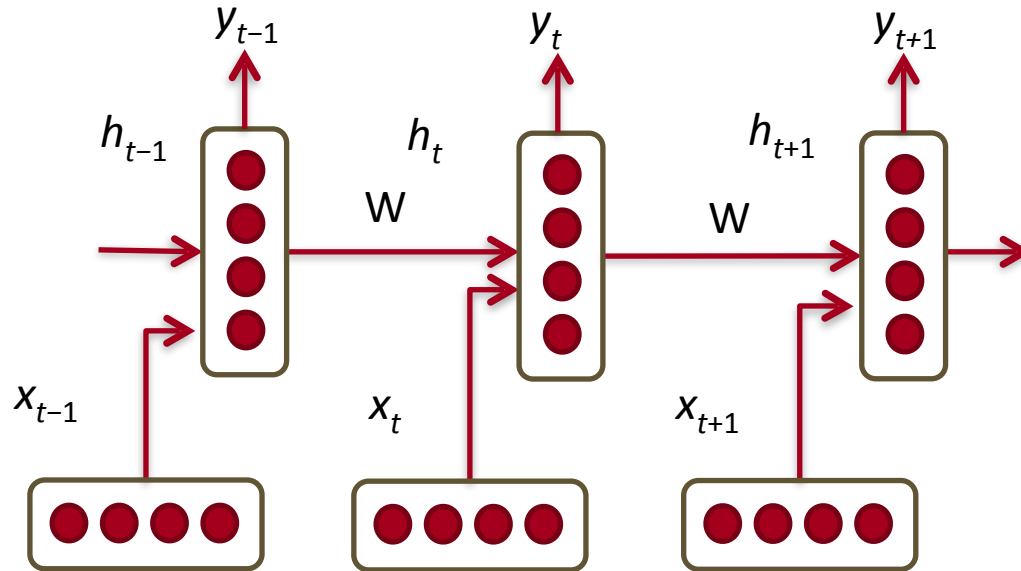$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$

$h_0 \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer at time step 0

$x_{[t]}$ is the column vector of L at index [t] at time step t

$W^{(hh)} \in \mathbb{R}^{D_h \times D_h}$ $\qquad W^{(hx)} \in \mathbb{R}^{D_h \times d}$ $\qquad W^{(S)} \in \mathbb{R}^{|V| \times D_h}$
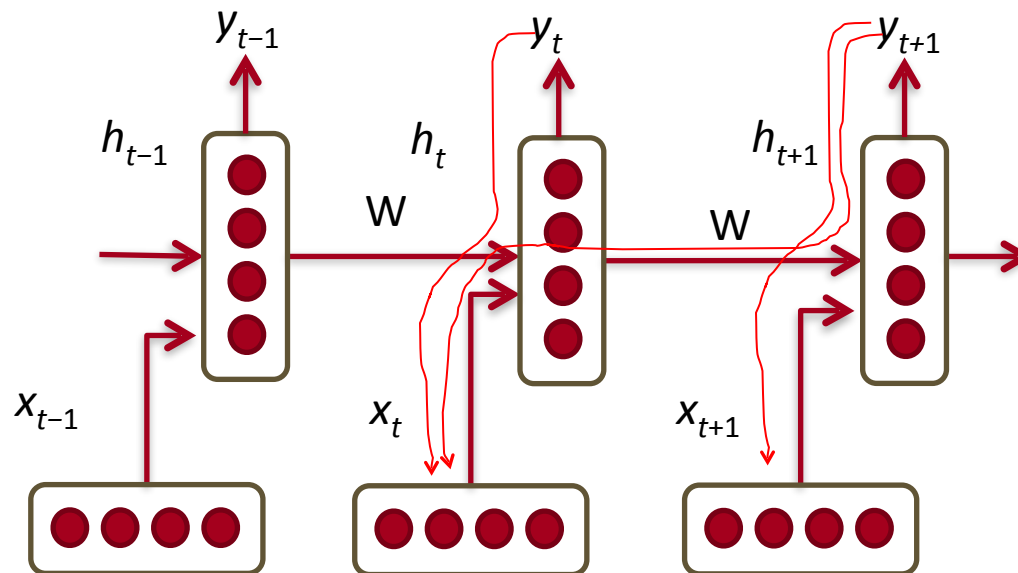
Training RNNs is hard

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

- Ideally inputs from many time steps ago can modify output *y*
- Take $\dfrac{\partial E_2}{\partial W}$ 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:

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

- Total error is the sum of each error at time steps t

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

- Hardcore chain rule application:

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

2/2/17

The vanishing gradient problem - Details

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

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

- Remember: $\quad h_t \;=\; W f(h_{t-1}) + W^{(hx)} x_{[t]}$
- 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:

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

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**

2/2/17

Why is the vanishing gradient a problem?

- The error at a time step ideally can tell a previous time step from many steps away to change during backprop
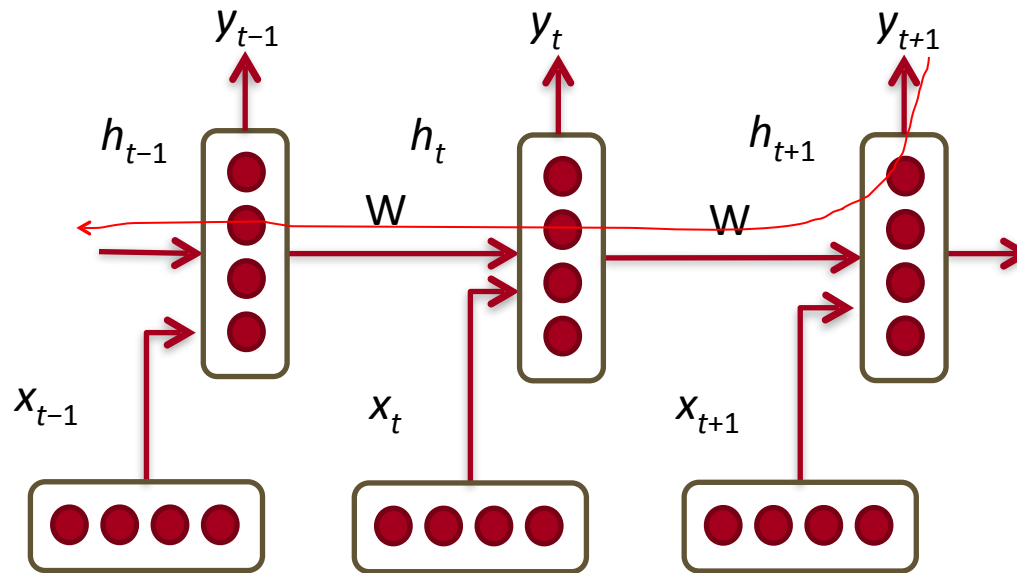
The vanishing gradient problem for language models

- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word

- Example:

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

Trick for exploding gradient: clipping trick

- The solution first introduced by Mikolov is to clip gradients to a 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**

$\quad \hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$

end if

- Makes a big difference in RNNs.

2/2/17

For vanishing gradients: Initialization + ReLus!

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

- →   Huge difference!



Pixel-by-pixel permuted MNIST
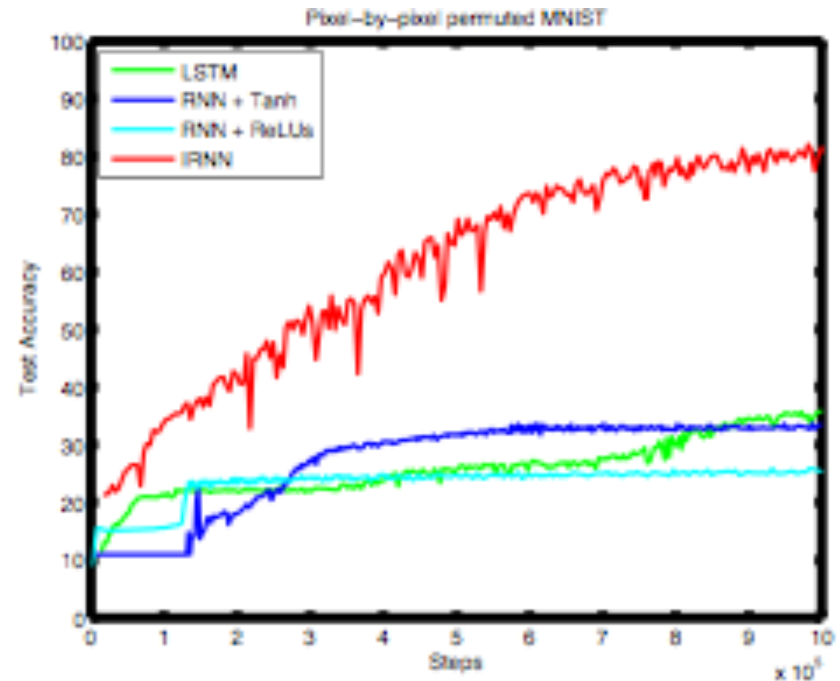
- Initialization idea first introduced in *Parsing with Compositional Vector Grammars*, Socher et al. 2013

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

2/2/17

One last implementation trick

- You only need to pass backwards through your sequence once and accumulate all the deltas from each E_t

Bidirectional RNNs

Problem: For classification you want to incorporate information from words both preceding and following
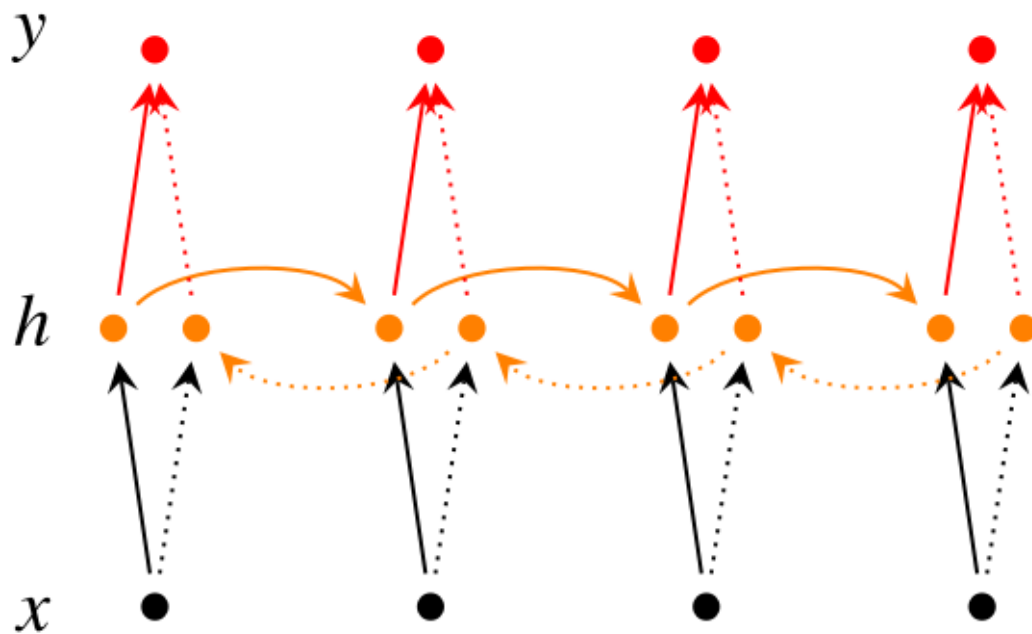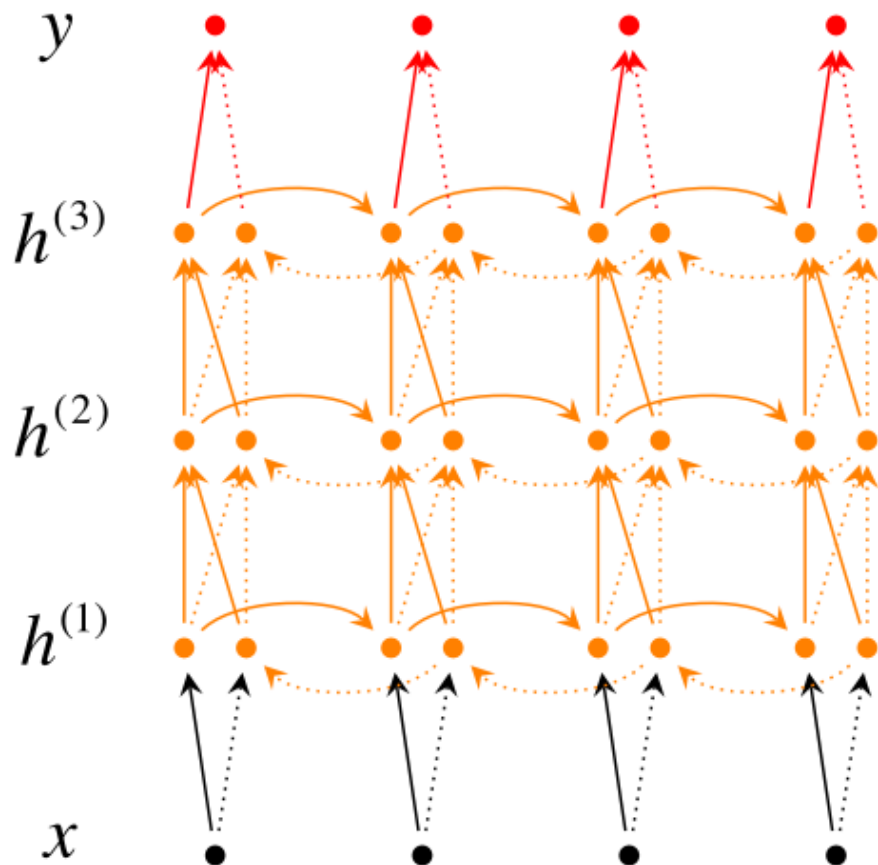
$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

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

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

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

2/2/17

Deep Bidirectional RNNs

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

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

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

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

2/2/17

Recap

- Recurrent Neural Network is one of the best deepNLP model families

- Training them is hard because of vanishing and exploding gradient problems

- They can be extended in many ways and their training improved with many tricks (more to come)

- Next week: Most important and powerful RNN extensions with LSTMs and GRUs

2/2/17