# Deep Learning for Text Mining

## Part 2. Recurrent Neural Networks (RNN)



1

## Outline

- Optimize Neural Network
  - Stochastic Gradient Descent (Recap)
  - Backpropagation
- Recurrent Neural Network (RNN)
  - Vanilla RNN
  - Gated RNN



2

# Stochastic Gradient Descent (Recap)

- Training data $\mathrm{D} = \{(x_i, y_i)\}_{i=1}^{n}$
- Loss function $l_i(w) \stackrel{\text{def}}{=} l(f_w(x_i), y_i)$, model $f_w$ parameterized by $w$
- The goal of training: $\min\limits_{w} \frac{1}{n} \sum_{i=1}^{n} l_i(w)$
- Random mini-batch: sample $\mathrm{B} \overset{unif}{\sim} \{1,2,\ldots,n\}$
- Update $w$ using the gradient computed from the mini-batch:

$$w^{(k)} := w^{(k-1)} - \eta_k \nabla \left( \frac{1}{|B|} \sum_{i \in B} l_i(w^{(k-1)}) \right)$$



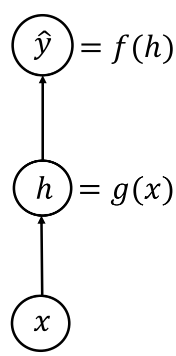


[L.P. Morency: CMU 11-777]

# Gradient Computation

- Chain rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial h} \frac{\partial h}{\partial x}$$

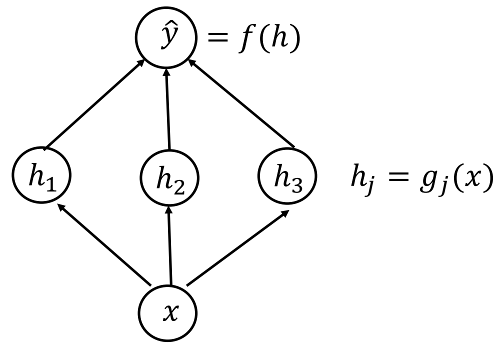


[L.P. Morency: CMU 11-777]

## Gradient Computation (cont'd)

- Chain rule

$$\frac{\partial f}{\partial x} = \sum_j \frac{\partial f}{\partial h_j} \frac{\partial h_j}{\partial x}$$








[L.P. Morency: CMU 11-777]

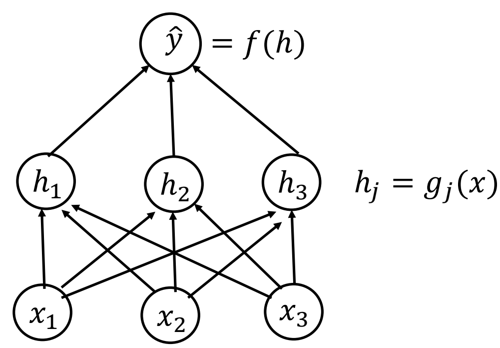
## Gradient Computation (cont'd)

- Chain rule

$$\frac{\partial f}{\partial x_1} = \sum_j \frac{\partial f}{\partial h_j} \frac{\partial h_j}{\partial x_1}$$

$$\frac{\partial f}{\partial x_2} = \sum_j \frac{\partial f}{\partial h_j} \frac{\partial h_j}{\partial x_2}$$

$$\frac{\partial f}{\partial x_3} = \sum_j \frac{\partial f}{\partial h_j} \frac{\partial h_j}{\partial x_3}$$



# Gradient Computation (cont'd)

- the gradient (scalar-by-vector)

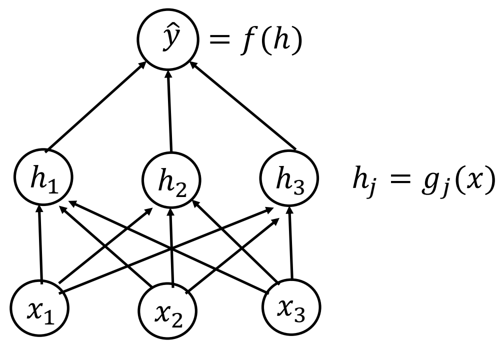
$$\nabla_x f = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \frac{\partial f}{\partial x_3}\right]$$

$$= \boxed{\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right)^T} \boxed{\nabla_{\boldsymbol{h}} \boldsymbol{f}}$$

Jacobian matrix of size $|h| \times |x|$

Gradient vector of size |h|

$$\begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \frac{\partial h_1}{\partial x_3} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \frac{\partial h_2}{\partial x_3} \\ \frac{\partial h_3}{\partial x_1} & \frac{\partial h_3}{\partial x_2} & \frac{\partial h_3}{\partial x_3} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial f}{\partial h_1} \\ \frac{\partial f}{\partial h_2} \\ \frac{\partial f}{\partial h_3} \end{bmatrix}$$



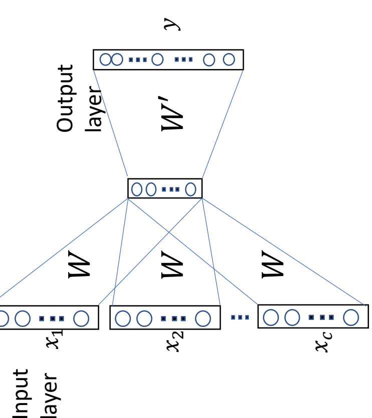


@Yiming Yang, lecture on Deep Learning for Text Mining

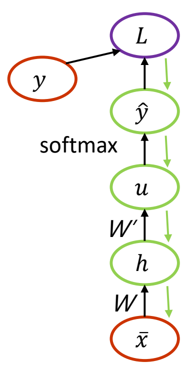

7

# Example: Word Embedding in CBOW

- Architecture




- Chained Dependencies




$$\text{loss}(y, \hat{y}) = -\log p(w_O | w_{I1}, \ldots, w_{IC})$$
$$= -u_{j^*} + \log\left(\textstyle\sum_{j'=1}^{|V|} \exp(u_{j'})\right)$$

$$\hat{y} \in \mathbb{R}^d, \qquad \hat{y}_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

$$u \in \mathbb{R}^d, \qquad u = (W')^T \mathrm{h}$$

$$h \in \mathbb{R}^k, \qquad h = W^T \mathrm{x}$$

$$\bar{x} = \frac{1}{C}(x_1 + x_2 + \cdots + x_C)$$



8

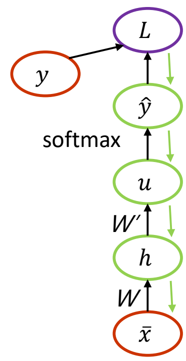
# Optimization Problem

- **Objective**

$$\min_{W,W'} \frac{1}{n}\sum_{i=1}^{n} L_i(W, W')$$

(we omit i on the right for simplicity).

- **Model Parameters**
  - Matrices W and W'
- **Initialization**
  - Randomly set W and W'.
- **Training Process**
  - Forward Propagation (black arrows) and backpropagation (green arrows) iteratively until convergence





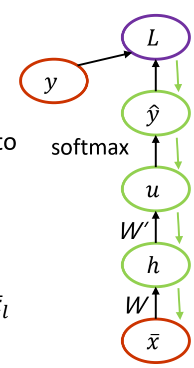

# Training Process

- **Forward propagation (black arrows)**
  - For each $(x, y)$ pair, use the current $W$ and W′ to compute the values of each variable (h, $u$, $\hat{y}$ and $L$).
- **Backpropagation (green arrows)**
  - Use the values in the forward propagation to compute the gradient, e.g.

$$\frac{\partial L}{\partial W'_{ij}} = \frac{\partial L}{\partial u_j} \cdot \frac{\partial u_j}{\partial W'_{ij}} = (\hat{y}_j - t_j) h_i$$

$$\frac{\partial L}{\partial W_{li}} = \frac{\partial L}{\partial h_i} \cdot \frac{\partial h_i}{\partial W_{ki}} = \sum_{j=1}^{|V|} (\hat{y}_j - t_j) \cdot W'_{ij} \bar{x}_l$$

  - Update W and W' with the gradients (slides #24 and 25 in the previous lecture).




$$\text{loss}(y, \hat{y}) = -\log p(w_O | w_{I1}, \ldots, w_{IC})$$
$$= -y_{j^*} = \text{-}log \frac{\exp(u_{j*})}{\sum_{j'=1}^{V} \exp(u_{j'})}$$
$$= -u_{j^*} + \log(\sum_{j'=1}^{|V|} \exp(u_{j'}))$$

$$u = (W')^T \text{h}, \quad u_j = \sum_{i=1}^{k} W'_{ij} h_i$$

$$h = W^T \text{x}$$
$$h_i = W_{1i}\bar{x}_1 + W_{2i}\bar{x}_2 + \cdots + W_{|V|i}\bar{x}_{|V|} \in R$$

$$\bar{x} = \frac{1}{C}(x_1 + x_2 + \cdots + x_C)$$

# Outline

- Optimize Neural Network
  - Stochastic Gradient Descent (Recap)
  - Backpropagation
- Recurrent Neural Network (RNN)
  - Vanilla RNN
  - Gated RNN





# Vanilla Neural Network

Example: a multi-class classifier for text categorization

- Input $x \in \mathbb{R}^d$s a vector representation of a document
- Output $\hat{y} = g(W_{hy}h)$ is a probability distribution over $K$ categories, whose elements are

$$g_j(z) = \frac{\exp(z_j)}{\sum_{j'} \exp(z_{j'})} \in (0,1) \text{ for } j = 1, \cdots, K$$

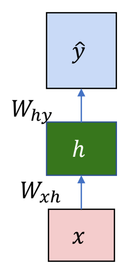
- Hidden layer $h = f(W_{xh}x)$ with

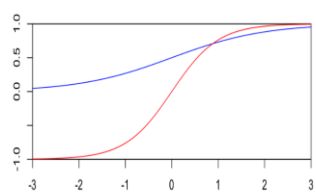
$$f(\mathrm{z}) = \tanh(\mathrm{z}) \in (\pm 1)$$

is the rescaled logistic sigmoid function

$$\sigma(z) \stackrel{\text{def}}{=} \frac{e^z}{1+e^z}$$

$$\tanh(z) \stackrel{\text{def}}{=} \frac{e^z - e^{-z}}{e^z + e^{-z}} = 2\sigma(2z) - 1$$



# Limitation of Vanilla Neural Network

- Not taking the sequential order of input variables into account
- Not modeling the sequential dependencies among output variables
- Cannot support language modeling, for example, the task of predicting future word(s) based on previously observed ones





[Adapted from F.F. Li et al. Stanford CS231n]

# Recurrent Neural Network

- Modeling a sequence of $(x_t, h_t, y_t)$ with the recurrence formula for step $t$ as

$$\boxed{h_t} = \boxed{f_w}(\boxed{h_{t-1}}, \boxed{x_t})$$

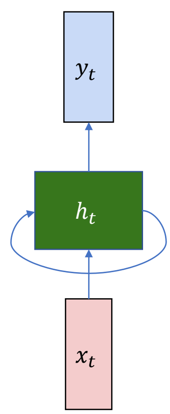
New state

Old state

Some function parameterized by w

Input vector at some time step

- Notice the same function ($f$) and the same set of model parameters ($w$) are used at every time step.



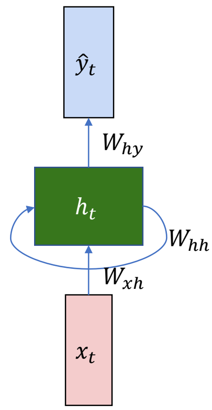
[Adapted from F.F. Li et al. Stanford CS231n]

## Specifically, we may define

$$\hat{y}_t = g(W_{hy}h_t) = W_{hy}h_t$$

$$h_t = f(h_{t-1}, x_t) \stackrel{\text{def}}{=} \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

for $t = 1,2,3, \ldots$, usually we have $h_0 = \vec{0}$



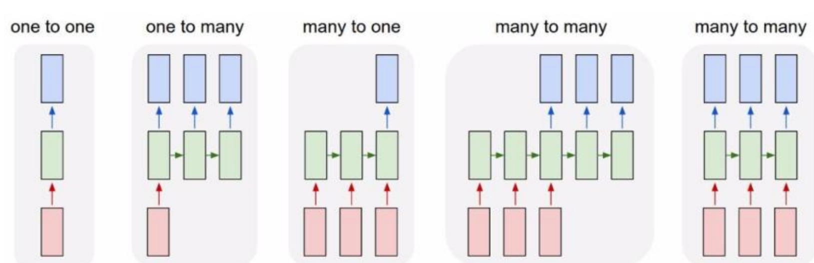



15

[Adapted from F.F. Li et al. Stanford CS231n]

## Different RNNs

- Vanilla Neural Network (e.g., for binary image classification)



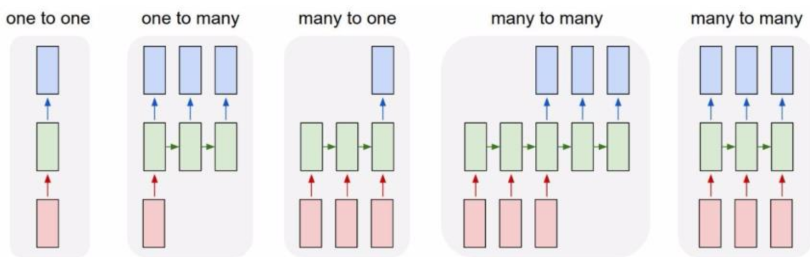



16

[Adapted from F.F. Li et al. Stanford CS231n]

# Different RNNs

- E.g. Image Captioning
  Image → word sequence



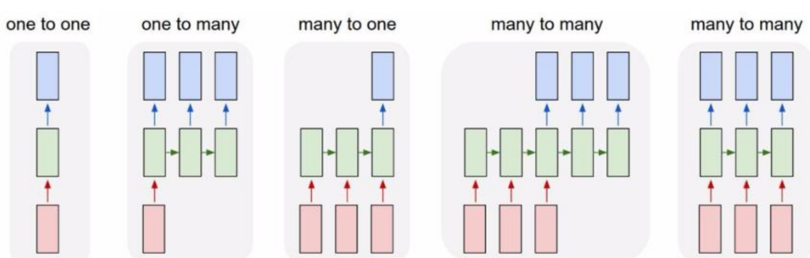



17

[Adapted from F.F. Li et al. Stanford CS231n]

# Different RNNs

E.g. Text classification
Word sequence → Topic label



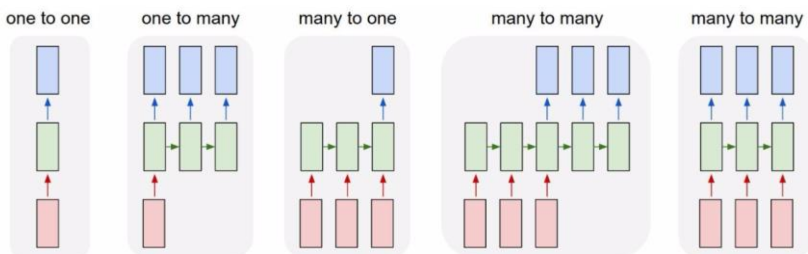



18

[Adapted from F.F. Li et al. Stanford CS231n]

# Different RNNs

E.g. Machine Translation
Word seq. → Word seq.



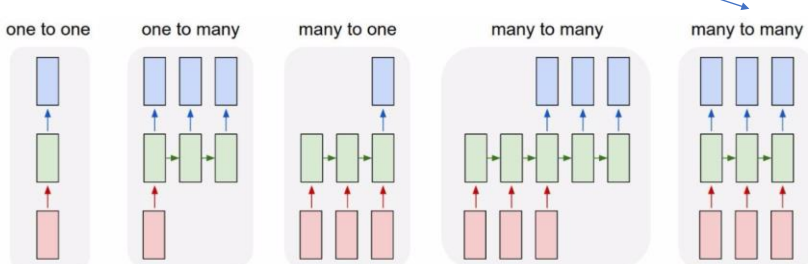



19

[Adapted from F.F. Li et al. Stanford CS231n]

# Different RNNs

E.g. POS tagging
Word seq. → Label seq.






20

# Example

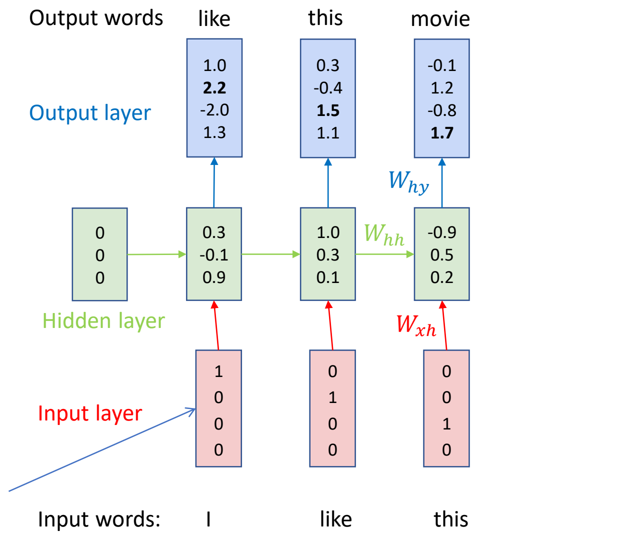
[Adapted from F.F. Li et al. Stanford CS231n]

**Word-level language model**

Vocabulary: [“I”, “like”, “this”, “movie”]

Example training sequence: “I like this movie”

It can also be a dense vector trained by word2vec or GloVe






21

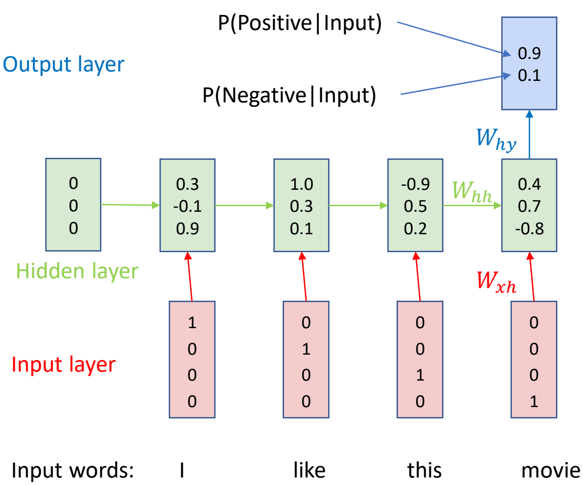
# Example

[Adapted from F.F. Li et al. Stanford CS231n]

**Word-level sentiment classification example**

Vocabulary: [“I”, “like”, “this”, “movie”]

Example training sequence: (“I like this movie”, Positive)



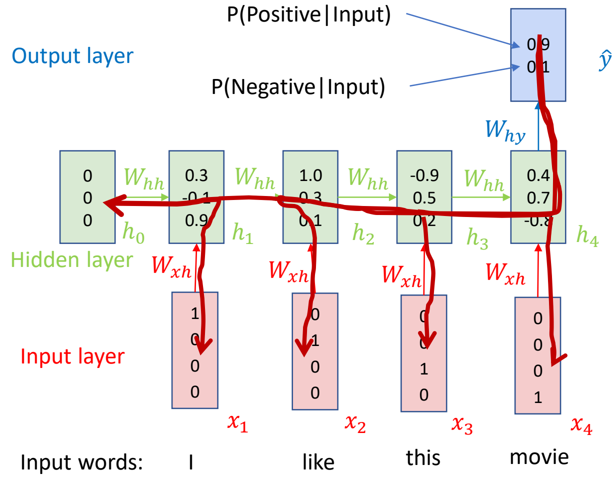



22

[Adapted from F.F. Li et al. Stanford CS231n]

# The vanishing gradient problem

- Multiplying the same matrix at each time step during backpropagation
- Causing the gradient to become very small or very large quickly, which is called the **vanishing or exploding gradient**






23

# Why do gradients vanish (or explode) in neural nets?

- "Neural Networks and Deep Learning", Chapter 5, by Michael Nielsen (Dec 2017), [http://neuralnetworksanddeeplearning.com/chap5.html](http://neuralnetworksanddeeplearning.com/chap5.html)
- Example: a multi-layer nnet with a sigmoid function at each layer
- Showing how the gradient of the output variable w.r.t. an input-layer variable would vanish or explode when the number of layers increases
- More generally, "neural networks suffer from an unstable gradient problem."



24

[based on the RNN tutorial by WILDML, 2015]

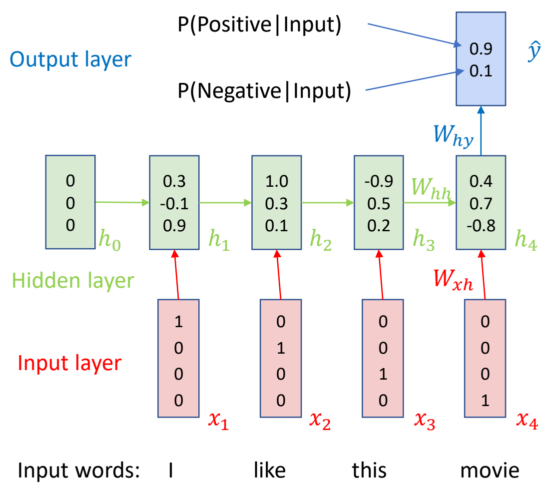
# Why do gradients vanish in RNN?

- Suppose the loss function is $L$. We would (incorrectly) compute $\frac{\partial L}{\partial W_{hh}}$ as

$$\frac{\partial L}{\partial W_{hh}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial h_4}{\partial W_{hh}}$$

- But $h_4 = f_w(h_3, x_4)$ depends on $h_3$, which also depends on $W_{hh}$. Thus the correct formula is

$$\frac{\partial L}{\partial W_{hh}} = \sum_{k=1}^{4} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial h_4}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$








[based on the RNN tutorial by WILDML, 2015]

# Why do gradients vanish in RNN (cont'd)

Recall: $h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$

Rewrite the gradient as:

$$\frac{\partial L}{\partial W_{hh}} = \sum_{k=1}^{4} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial h_4}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$

$$= \sum_{k=1}^{4} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_4} \boxed{\left( \prod_{j=k+1}^{4} \frac{\partial h_j}{\partial h_{j-1}} \right)} \frac{\partial h_k}{\partial W_{hh}}$$

Multiply the Jacobian matrix multiple times

[based on the RNN tutorial by WILDML, 2015]

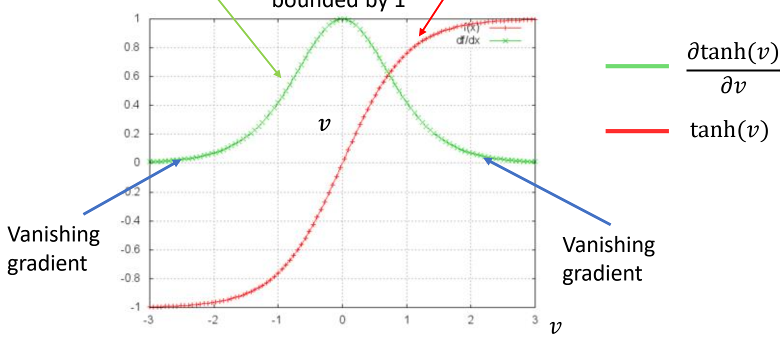
## Why do gradients vanish in RNN (cont'd)

$$\prod_{j=k+1}^{4} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{4} \boxed{\frac{\partial \tanh(v_j)}{\partial v_j}} \frac{\partial v_j}{\partial h_{j-1}}$$

$v_t = W_{hh} h_{t-1} + W_{xh} x_t$

$\boxed{h_t = \tanh(v_t)}$

Gradient bounded by 1




Notice that $\frac{\partial h_j}{\partial h_{j-1}}$ is between 0 and 1, could be very near zero -- multiplying it several times would result in a vanishing gradient.





## Outline

- Optimize Neural Network
  - Stochastic Gradient Descent (Recap)
  - Backpropagation
- Recurrent Neural Network (RNN)
  - Vanilla RNN
  - Gated RNN

# Gated Recurrent Neural Networks

- Addressing the gradient vanishing/exploding issue in naive RNN
- Controlling information flow via gates, which allow some time steps in the sequential process to be skipped
- Representative Approach: Long Short Term Memory (LSTM)

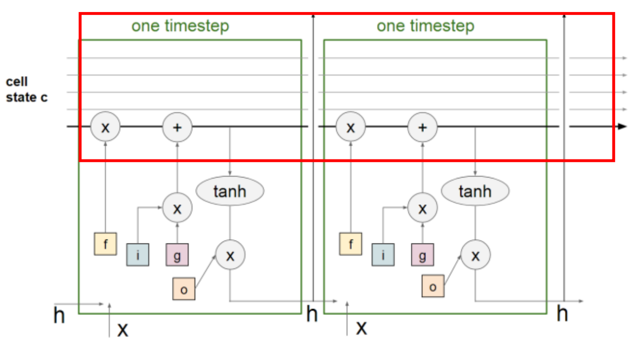




[Adapted from F.F. Li et al. Stanford CS231n]

# Long Short-Term Memory (LSTM)

- Using "cell state" c (a vector) to control the information flow forward over time
  - It only involves in linear operations and hence is less prone to vanishing gradient (unlike function $tahn$ in the gradient of latent h )
- Using "gates" (f, i, g, o) to allow context/gradient to pass through without changing



# Plain RNN vs. LSTM

- In Vanilla RNN we have

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

- We can rewrite it as

$$h_t = \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \quad \text{where } W = [W_{hh} \quad W_{xh}]$$

- LSTM applies **the gate functions** to vanilla RNN as

f is the "forget gate";
i is the "input gate;
o is the "output gate;
g is a mapping function.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \boxed{\begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix}} \boxed{W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}}$$

Same as in plain RNN

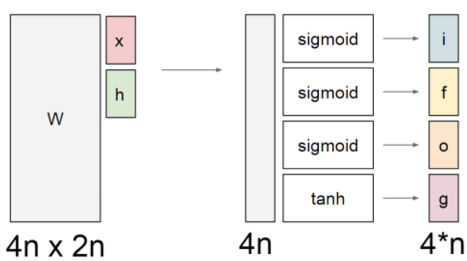




[Adapted from F.F. Li et al. Stanford CS231n]

# LSTM details

- Assume $x$ and $h$ have the same dimension $n$



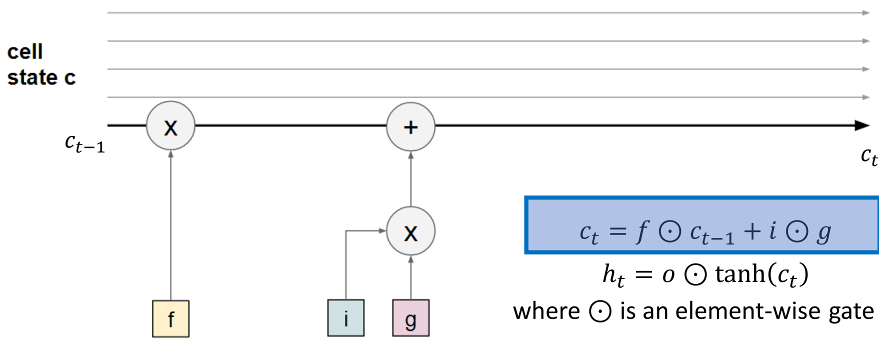

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

where $\odot$ is an element-wise gate operator

[Adapted from F.F. Li et al. Stanford CS231n]

# LSTM details




$$\boxed{c_t = f \odot c_{t-1} + i \odot g}$$

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

where $\odot$ is an element-wise gate operator

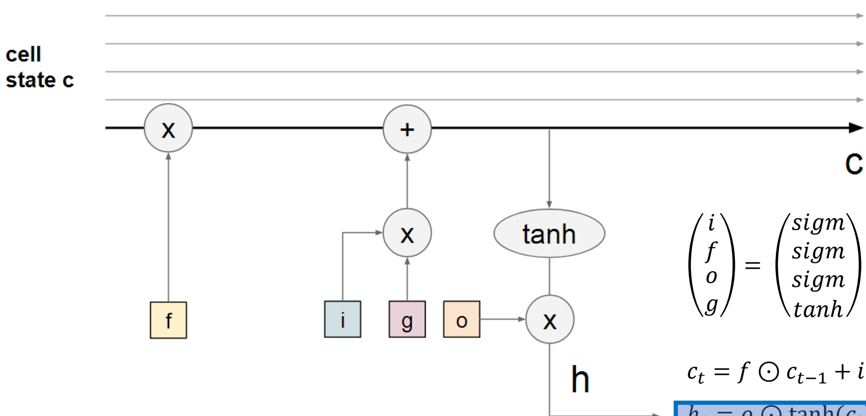
- If $f$ is all ones and $i$ is all zeros, the current cell ignores the current signals (h and x combined in g) and carries all the memory to the next step.
- What if $f$ is all zeros and $i$ is all ones?





[Adapted from F.F. Li et al. Stanford CS231n]

# LSTM details



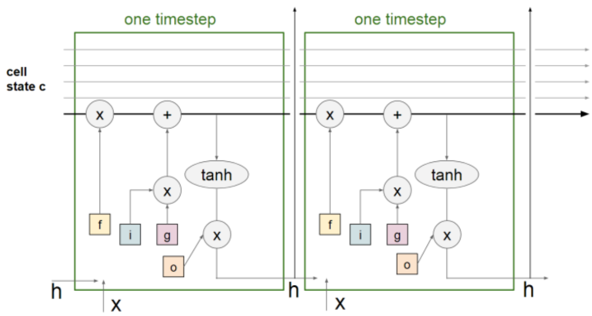

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$\boxed{h_t = o \odot \tanh(c_t)}$$

Hidden state $h_t$ is the output of tanh activation where outgate $o$ controls $h_t$ going forward not.

# LSTM details




- Information flow: Both $c_t$ and $h_t$ carries information to the next time step. But $c_t$ is only updated by linear operations while $h_t$ is updated by a non-linear one.
- GRU (Gated Recurrent Unit) combines c and h into a single latent variable instead two in LSTM (and is more popular).





# Performance on language modeling [Yoon Kim et al. AAAI 2016]

| | $PPL$ | Size | |
|---|---|---|---|
| LSTM-Word-Small | 97.6 | 5 m | |
| LSTM-Char-Small | 92.3 | 5 m | |
| LSTM-Word-Large | 85.4 | 20 m | |
| LSTM-Char-Large | 78.9 | 19 m | |
| KN-5 (Mikolov et al. 2012) | 141.2 | 2 m | 5-gram LM |
| RNN$^\dagger$ (Mikolov et al. 2012) | 124.7 | 6 m | Plain RNN LM |
| RNN-LDA$^\dagger$ (Mikolov et al. 2012) | 113.7 | 7 m | |
| genCNN$^\dagger$ (Wang et al. 2015) | 116.4 | 8 m | |
| FOFE-FNNLM$^\dagger$ (Zhang et al. 2015) | 108.0 | 6 m | |
| Deep RNN (Pascanu et al. 2013) | 107.5 | 6 m | |
| Sum-Prod Net$^\dagger$ (Cheng et al. 2014) | 100.0 | 5 m | |
| LSTM-1$^\dagger$ (Zaremba et al. 2014) | 82.7 | 20 m | |
| LSTM-2$^\dagger$ (Zaremba et al. 2014) | 78.4 | 52 m | |

**Table 3:** Performance of our model versus other neural language models on the English Penn Treebank test set. $PPL$ refers to perplexity (lower is better) and size refers to the approximate number of parameters in the model. KN-5 is a Kneser-Ney 5-gram language model which serves as a non-neural baseline. $^\dagger$For these models the authors did not explicitly state the number of parameters, and hence sizes shown here are estimates based on our understanding of their papers or private correspondence with the respective authors.

# Reference


- Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
- Louis-Philippe Morency, Tadas Baltrusaitis, **CMU 11-777: Advanced Multimodal Machine Learning**
- Fei-Fei Li, Andrej Karpathy, Justin Johnson Stanford CS231n: Convolutional Neural Networks for Visual Recognition
- RNN tutorial by WILDML http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
- Christopher Olah's blog: Understanding LSTM Networks
- Denny Britz: Recurrent Neural Networks tutorial