# Artificial Neural Networks
## 人工神经网络

权小军教授
中山大学计算机学院

quanxj3@mail.sysu.edu.cn

2023 年 5 月 16 日

# Lecture 9 - Recurrent Neural Network I
# 循环神经网络 I

# Lecture 9.1 Word Embedding 词嵌入

# Motivation: natural language processing

- Sentence or document classification (topic, sentiment)
- Topic Modeling 主题模型
- Machine Translation 机器翻译
- Chatbots, dialogue system 聊天机器人/对话系统
- Summarization 文本摘要
- Text generation 文本生成

# Word representation

- One-hot vectors are
  - long (length $|V| = 20{,}000$ to $50{,}000$)
  - sparse (most elements are zero)
- Alternative: learned vectors which are
  - short (length 50-1000)
  - dense (most elements are non-zero)

# Sparse vs. dense vectors

- Why dense vectors?
  - Short vectors may be easier to use as features in machine learning (fewer weights to tune)
  - Dense vectors may generalize better than explicit counts
  - Dense vectors may do better at capturing synonymy:
    - *car* and *automobile* are synonyms; but are distinct dimensions
    - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
  - In practice, they work better

# Common methods for getting short dense vectorsn

- "Neural Language Model" -inspired models
  - Word2vec (skipgram, CBOW), GloVe
- Singular Value Decomposition (奇异值分解，SVD)
  - A special case of this is called LSA -Latent Semantic Analysis
- Alternative to these "static embeddings":
  - Contextual Embeddings (语境嵌入,ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

# Simple static embeddings you can download!

- Word2vec (Mikolov et al)
  - Site: https://code.google.com/archive/p/word2vec/
- GloVe (Pennington, Socher, Manning)
  - Site: http://nlp.stanford.edu/projects/glove/

# Word2vec

- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: predict rather than count
- Word2vec provides various options. We'll do:
  - skip-gram with negative sampling (SGNS)

# Word2vec

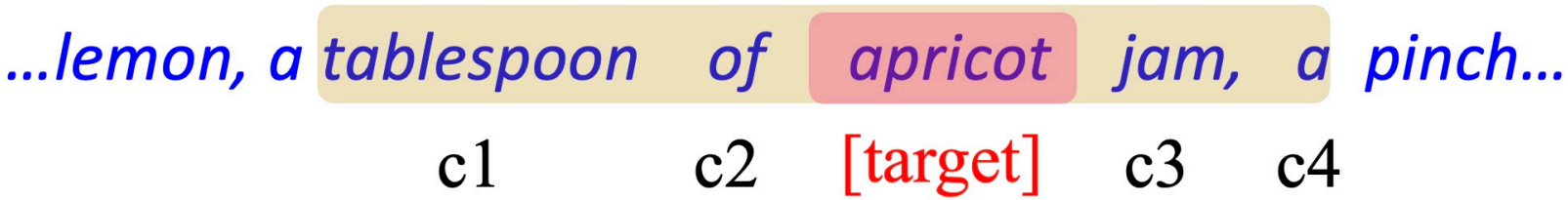
- Train a classifier on a binary prediction task:
  - Is *w* likely to show up near *bank*"?
- We don't actually care about this task
  - But we'll take the learned classifier weights as the word embeddings
- Big idea: self-supervision (自监督):
  - A word *c* that occurs near *bank* in the corpus as the gold "correct answer" for supervised learning
  - No need for human labels

# Approach: predict if candidate word c is a "neighbor"

1. Treat the target word $t$ and a neighboring context word $c$ as positive examples.
2. Randomly sample other words in the dictionary to get negative examples
3. Train a logistic regression classifier to distinguish those two cases
4. Use the learned weights as the embeddings

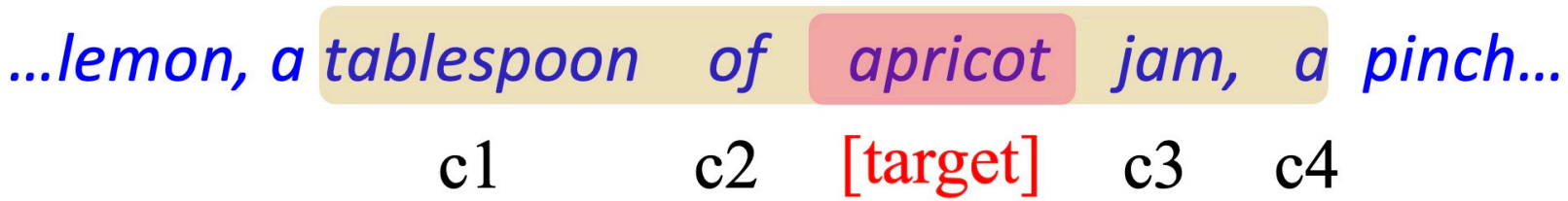
# Skip-Gram Training Data

- Assume a +/- 2 word window, given training sentence:



# Skip-Gram Classifier

- Assume a +/- 2 word window, given training sentence:




- Goal: train a classifier given a candidate (word, context) pair
  (*apricot*, *jam*)
  (*apricot*, *like*)
  ...
- And assigns each pair a probability:

$$P(+|w, c)$$

$$P(-|w, c) = 1 - P(+|w, c)$$

# Similarity is computed from dot product

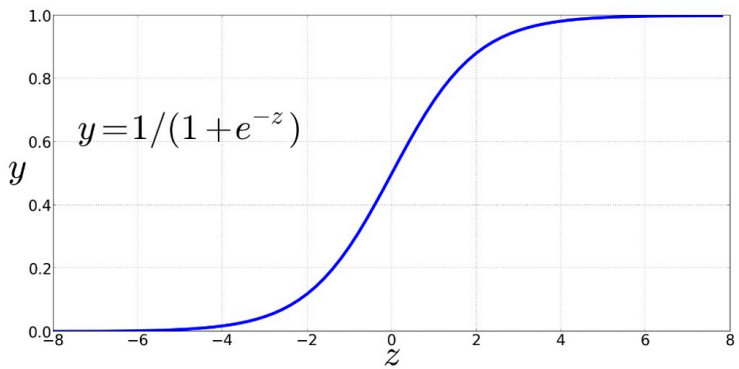
- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product
- So:
  - Similarity$(w, c) \propto w \cdot c$
- We'll need to normalize to get a probability
  - (cosine isn't a probability either)

# Turning dot products into probabilities

- Sim$(w, c) \approx w \cdot c$
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w,c) &= 1 - P(+|w,c) \\ &= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)} \end{aligned}$$



# How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

- This is for one context word, but have many context words.

# How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

- This is for one context word, but have many context words.
- We'll assume independence and just multiply them:

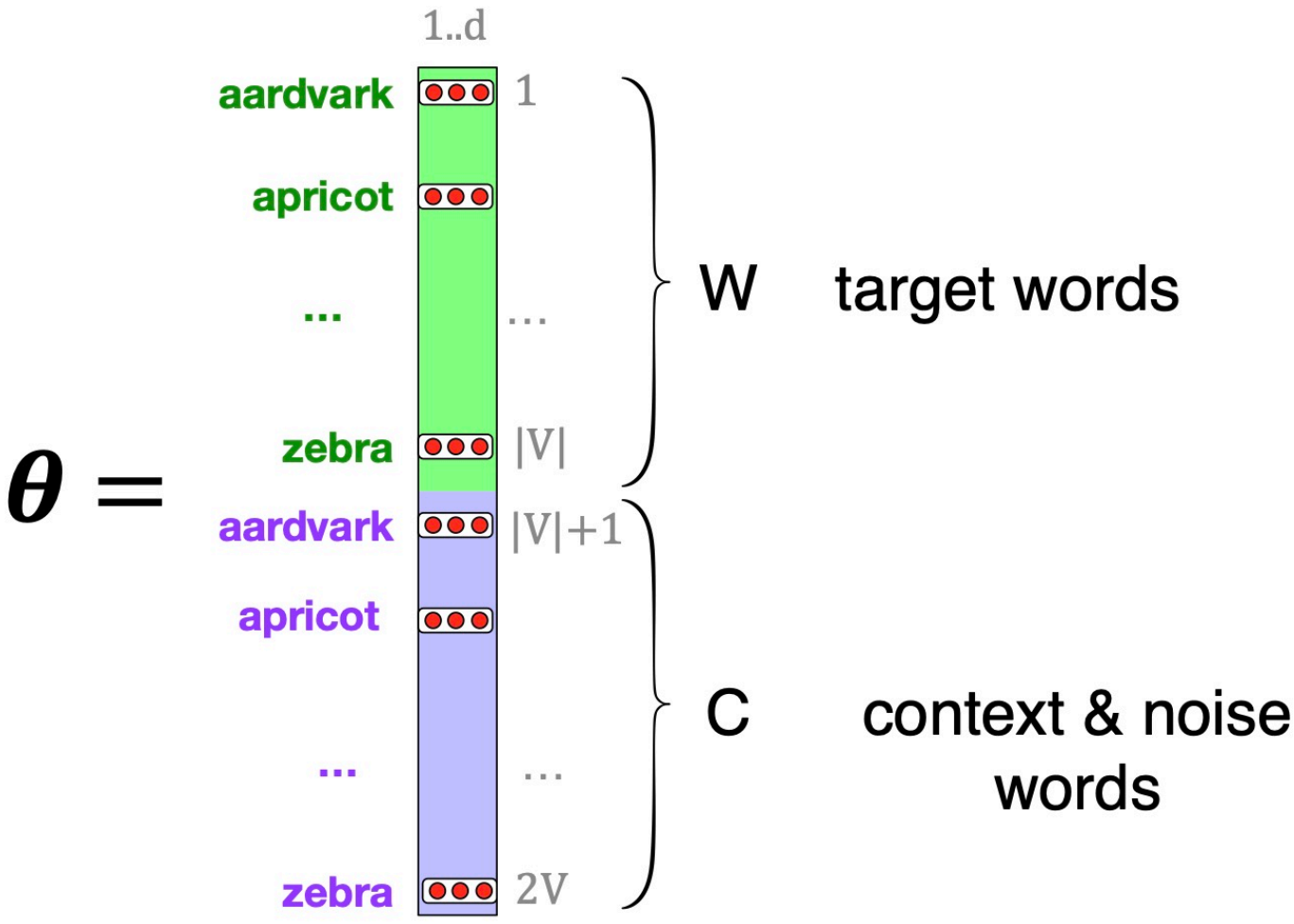
$$P(+|w,c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P(+|w,c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

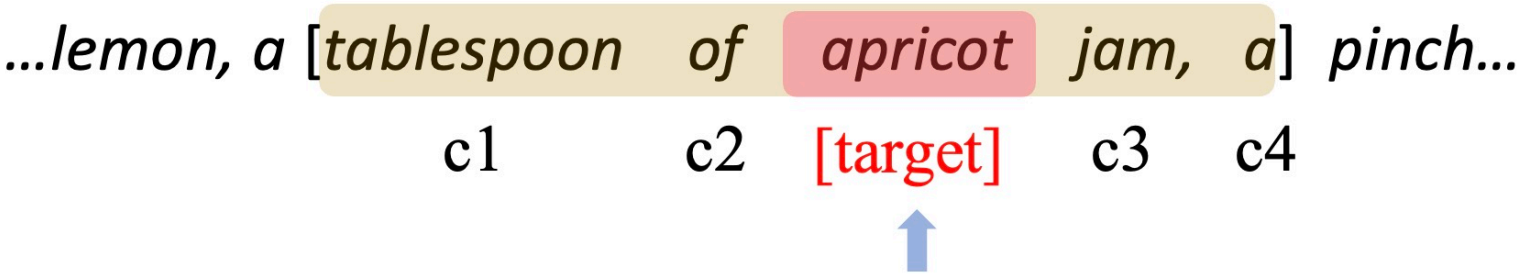
# Skip-gram classifier: summary

- A probabilistic classifier, given
  - a target word $w$
  - its context window of $L$ words $c_{1:L}$
- Estimates probability that $w$ occurs in this window based on similarity of $w$ (embeddings) to $c_{1:L}$ (embeddings).
- To compute this, we just need embeddings for all the words.

# These embeddings we'll need: a set for **W**, a set for **C**



# Skip-Gram Training data




**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

For each positive example we'll select *k* negative examples, sampling by frequency.

# Skip-Gram Training data

*...lemon, a [tablespoon of apricot jam, a] pinch...*

c1 c2 [target] c3 c4

**positive examples +**

| t | c |
|---|---|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

**negative examples -**

| t | c | t | c |
|---|---|---|---|
| apricot | aardvark | apricot | seven |
| apricot | my | apricot | forever |
| apricot | where | apricot | dear |
| apricot | coaxial | apricot | if |

# Word2vec: how to learn vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors.
- The goal of learning is to adjust those word vectors so that:
  - Maximize the similarity of the target word, context word pairs ($w$, $cpos$) drawn from the positive data
  - Minimize the similarity of the ($w$, $cneg$) pairs drawn from the negative data.

# Loss function for one $w$ with $c_{pos}$ , $c_{neg1}$ ...$c_{negk}$

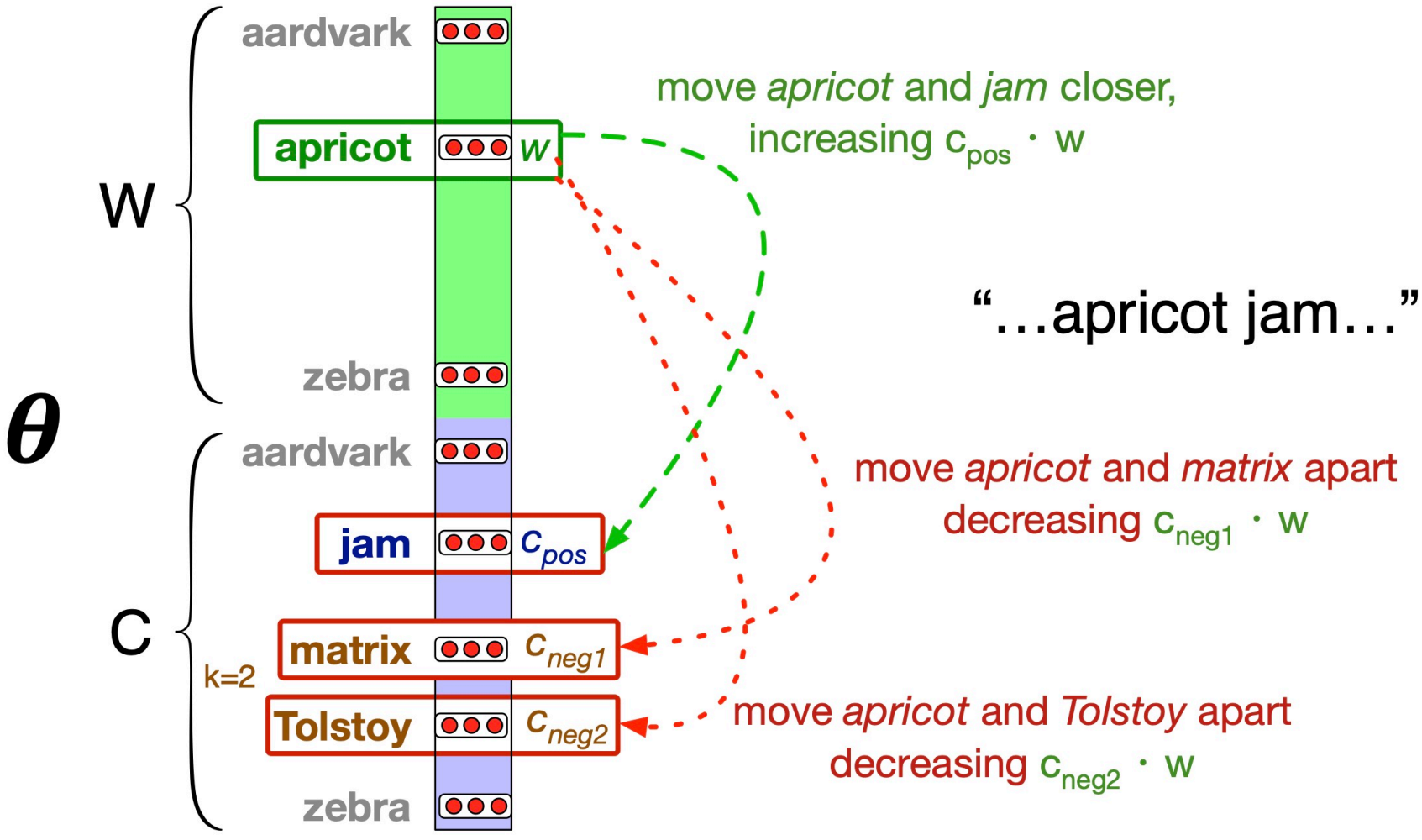
- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

# Learning the classifier

- How to learn?
  - Stochastic gradient descent! 随机梯度下降
- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.

# Intuition of one step of gradient descent



# The derivatives of the loss function

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w)]c_{neg_i}$$

# Update equation in SGD

Start with randomly initialized C and W matrices, then do updates

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta \left[ [\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w^{t})]c_{neg_i} \right]
\end{aligned}
$$

# Two sets of embeddings

- SGNS learns two sets of embeddings
  - Target embeddings matrix **W**
  - Context embedding matrix **C**
- It's common to just add them together, representing word $i$ as the vector $w_i + c_i$

[2]Throughout this note, we assume that the words and the contexts come from distinct vocabularies, so that, for example, the vector associated with the word *dog* will be different from the vector associated with the context *dog*. This assumption follows the literature, where it is not motivated. One motivation for making this assumption is the following: consider the case where both the word *dog* and the context *dog* share the same vector $v$. Words hardly appear in the contexts of themselves, and so the model should assign a low probability to $p(dog|dog)$, which entails assigning a low value to $v \cdot v$ which is impossible.

*Yoav Goldberg and Omer Levy. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method*

# Summary: How to learn word2vec (skip-gram) embeddings

- Start with $V$ random $d$-dimensional vectors as initial embeddings
  - Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

# Question

- **Question**: Why do we use negative sampling (负采样)?

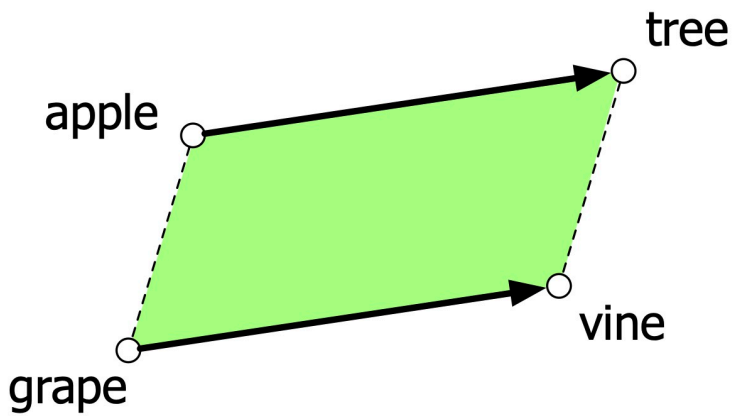
# Lecture 9.2 Properties of Embeddings
# 词嵌入的属性

# Analogical relations 类比关系

- The classic parallelogram (平行四边形) model of analogical reasoning (Rumelhart and Abrahamson 1973)

To solve: "*apple is to tree as grape is to* _____"

Add *tree* − *apple* to *grape* to get ***vine***



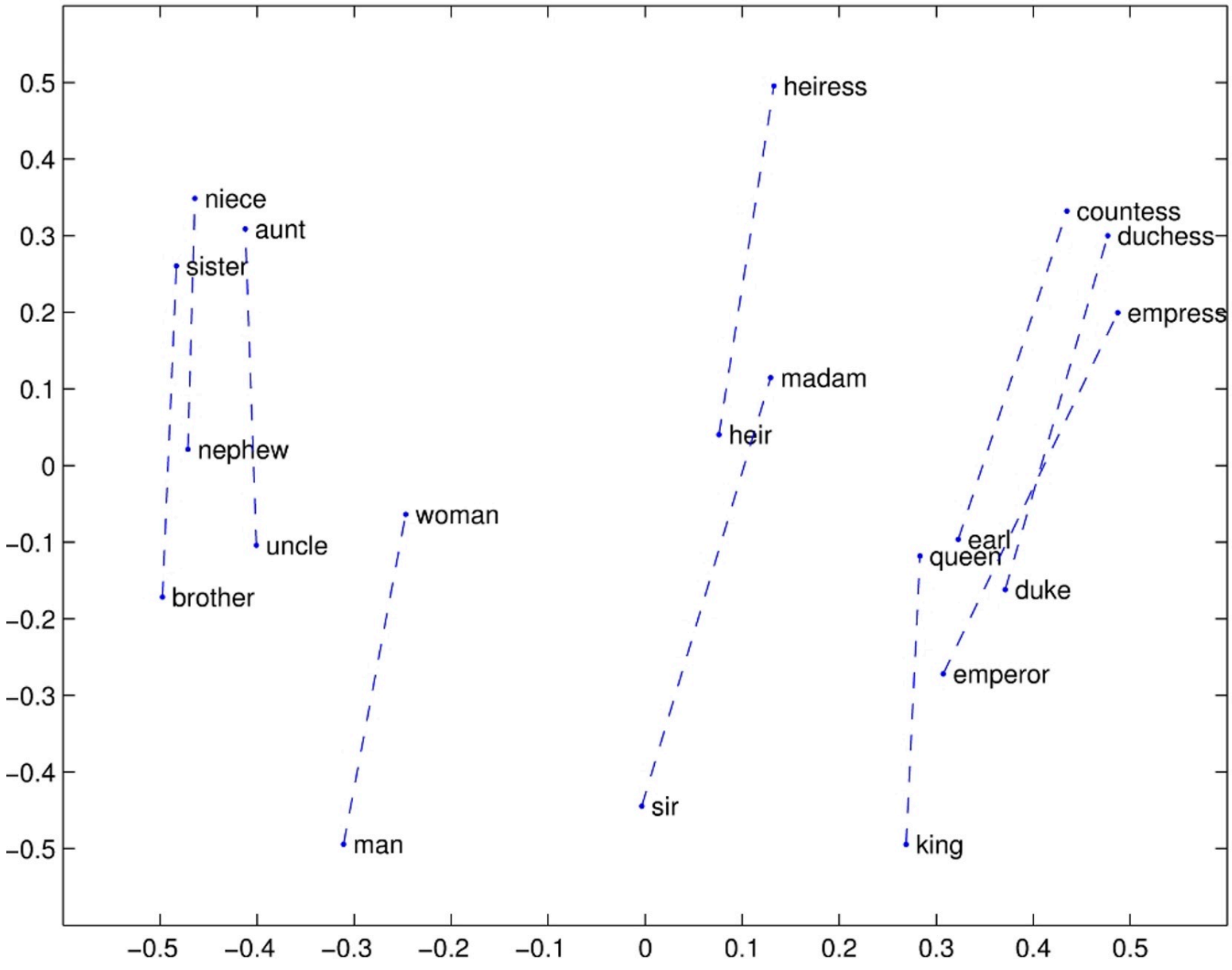
# Analogical relations via parallelogram

- The parallelogram method can solve analogies with both sparse and dense embeddings

$$\overrightarrow{\texttt{king}} - \overrightarrow{\texttt{man}} + \overrightarrow{\texttt{woman}} \text{ is close to } \overrightarrow{\texttt{queen}}$$

$$\overrightarrow{\texttt{Paris}} - \overrightarrow{\texttt{France}} + \overrightarrow{\texttt{Italy}} \text{ is close to } \overrightarrow{\texttt{Rome}}$$

# Structure in GloVE Embedding space



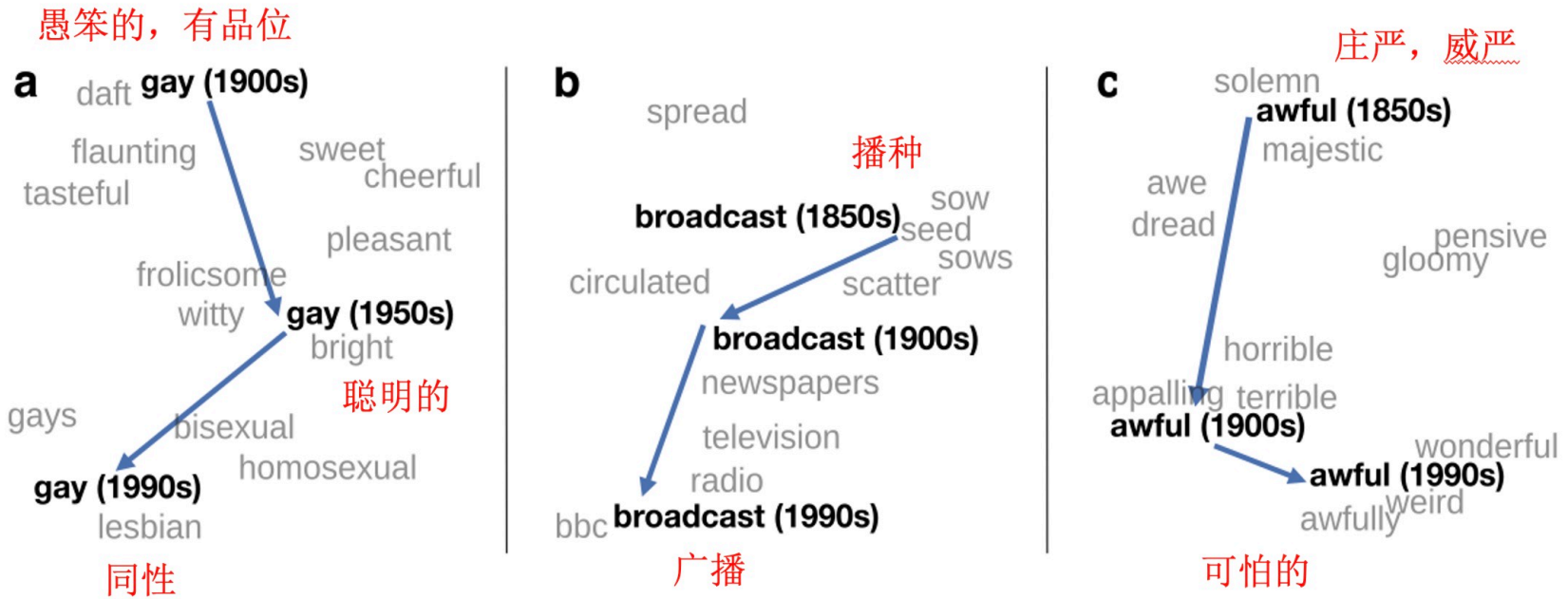
# Caveats with the parallelogram method

- It only seems to work for **frequent words**, **small distances** and **certain relations** (e.g., relating countries to capitals), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)
- Understanding analogy is an open area of research (Peterson et al. 2020)

# Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data



# Embeddings reflect cultural bias!

- Ask "*Paris* : *France* :: *Tokyo* : *x*"
  - *x* = *Japan*
- Ask "*father* : *doctor* :: *mother* : *x*"
  - *x* = *nurse*
- Ask "*man* : *computer programmer* :: *woman* : *x*"
  - *x* = *homemaker*

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

# Question

- **Question**: Can the vector values be negative?

# Word vector examples

明艳：( 0.53, 5.64, 3.45, -1.24, -3.24, 2.55, 5.66, 2.46, -4.27, -2.27, 1.56 )

浓艳：( 0.47, 3.46, 6.13, -1.33, -3.44, 2.87, 5.98, 2.41, -1.21, -2.21, 1.52 )

**Result of word clustering**

| Category | Adjective (Chinese)/形容词 |
|---|---|
| 1 | 明艳，浓艳，优美，柔媚，淡雅，素净，艳丽，多姿，清丽，…… |
| 2 | 进取，勤勉，肯于，坚韧，果敢，坦荡，坚韧，谦卑，不畏，…… |
| 3 | 宽容，包容，隐忍，忍耐，坦诚，坦荡，宽悯，坚韧，懂得，…… |
| 4 | 情绪化，急躁，暴躁，记仇，神经质，善变，孤僻，喜怒无常，…… |

# Lecture 9.3 Language Models 语言模型

# It only begins...

Word2vec is just for word representation.

How to capture meaning of sentence/paragraph?

# It only begins...

Word2vec is just for word representation.

How to capture meaning of sentence/paragraph?

We need to consider order of words in text!

# Language models

Language models: assign a probability to a sequence of words, such that reasonable sequences have higher probabilities, e.g.,

- $p(\text{“I like apples”}) > p(\text{“I sit apples”})$
- $p(\text{“I like apples”}) > p(\text{“like I apples”})$

# Language models

Language models: assign a probability to a sequence of words, such that reasonable sequences have higher probabilities, e.g.,

- $p(\text{“I like apples”}) > p(\text{“I sit apples”})$
- $p(\text{“I like apples”}) > p(\text{“like I apples”})$

Auto-regressive modeling of sequence $(\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_n)$ using Chain Rule (链式法则) of Probability:

$$p(\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_n) = p(\mathbf{w}_0) \cdot p(\mathbf{w}_1|\mathbf{w}_0) \ldots p(\mathbf{w}_n|\mathbf{w}_{n-1}, \mathbf{w}_{n-2}, \ldots, \mathbf{w}_0)$$

# Language models

Language models: assign a probability to a sequence of words, such that reasonable sequences have higher probabilities, e.g.,

- $p(\text{“I like apples”}) > p(\text{“I sit apples”})$
- $p(\text{“I like apples”}) > p(\text{“like I apples”})$

Auto-regressive modeling of sequence $(\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_n)$ using Chain Rule (链式法则) of Probability:

$$p(\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_n) = p(\mathbf{w}_0) \cdot p(\mathbf{w}_1|\mathbf{w}_0) \ldots p(\mathbf{w}_n|\mathbf{w}_{n-1}, \mathbf{w}_{n-2}, \ldots, \mathbf{w}_0)$$

- $p(\cdot)$ can be a neural network!
- $p(\cdot)$ could capture meaning of sequential information

# Chain Rule for joint probability of words

$$P(w_1 w_2 \cdots w_n) = \prod_i P(w_i \mid w_1 w_2 \cdots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

# Applications of language models

**Language model: assign a probability to a sentence**

- Machine Translation (机器翻译) :
  - $P$(high winds tonight) $>$ $P$(large winds tonight)
- Spell Correction (拼写纠正)
  - *The office is about fifteen* ***minuets*** *from my house*

    $P$(about fifteen minutes from) $>$$P$(about fifteen minuets from)
- Speech Recognition (语音识别)
  - $P$(I saw a van) $>>$ $P$(eyes awe of an)
- Summarization (自动摘要), question-answering, etc.

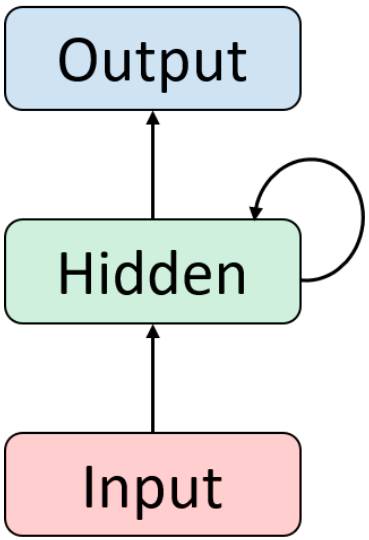
# Lecture 9.4 Recurrent neural networks (RNN)
# 循环神经网络

# Then...

What neural networks can represent $p(\cdot|\cdot)$?

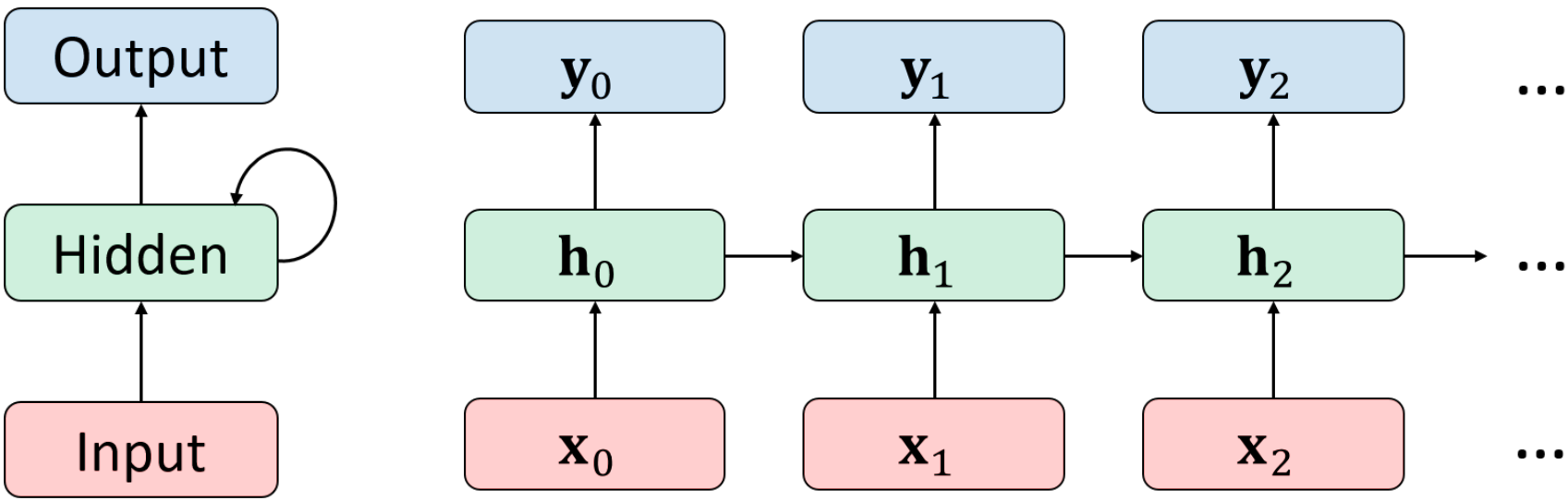
# Recurrent neural networks (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.



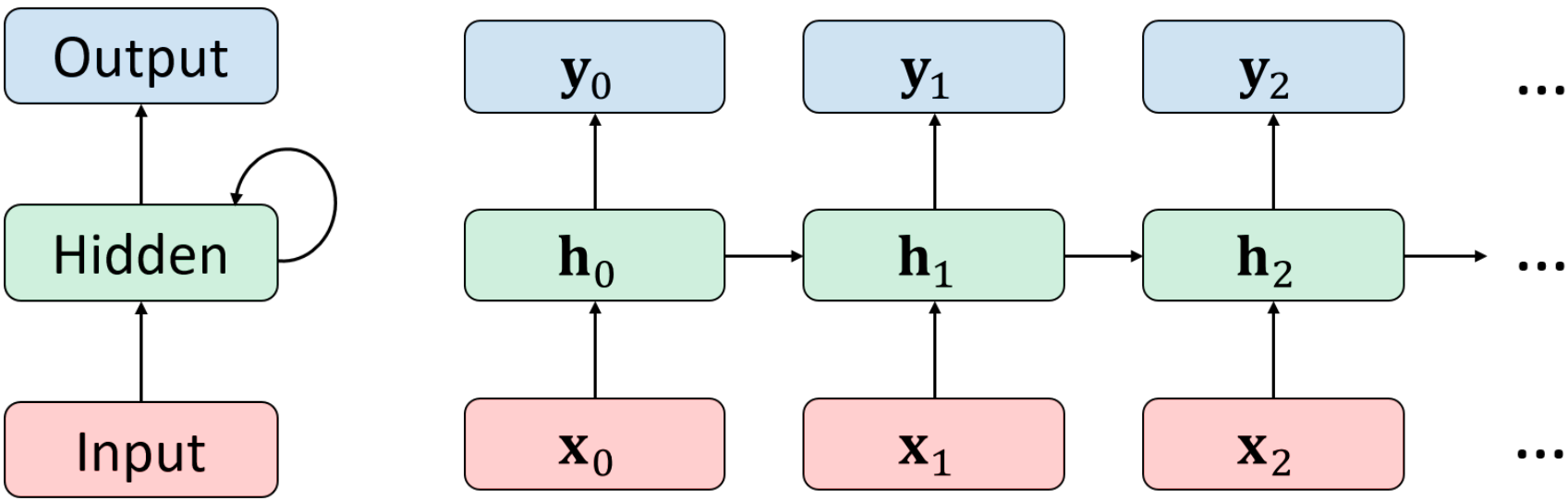
# Recurrent neural networks (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.
- Unroll to process an input sequence $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$

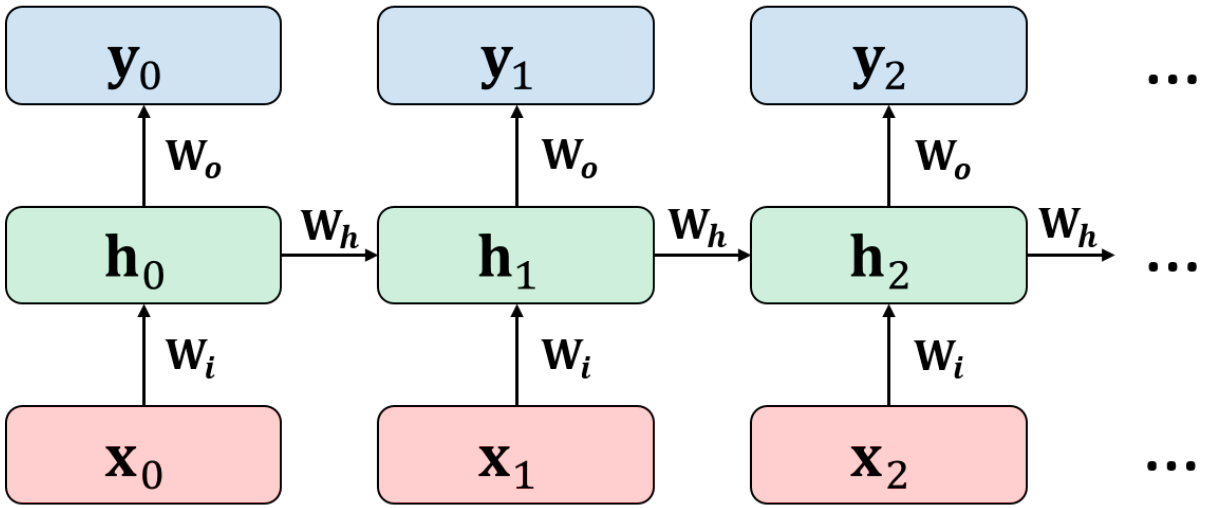


# Recurrent neural networks (RNN): basics

- Recurrent neural network: output of hidden layer at each time step is part of input to hidden layer at next time step.
- Unroll to process an input sequence $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots)$
- RNN is a DEEP neural network model



# RNN basics



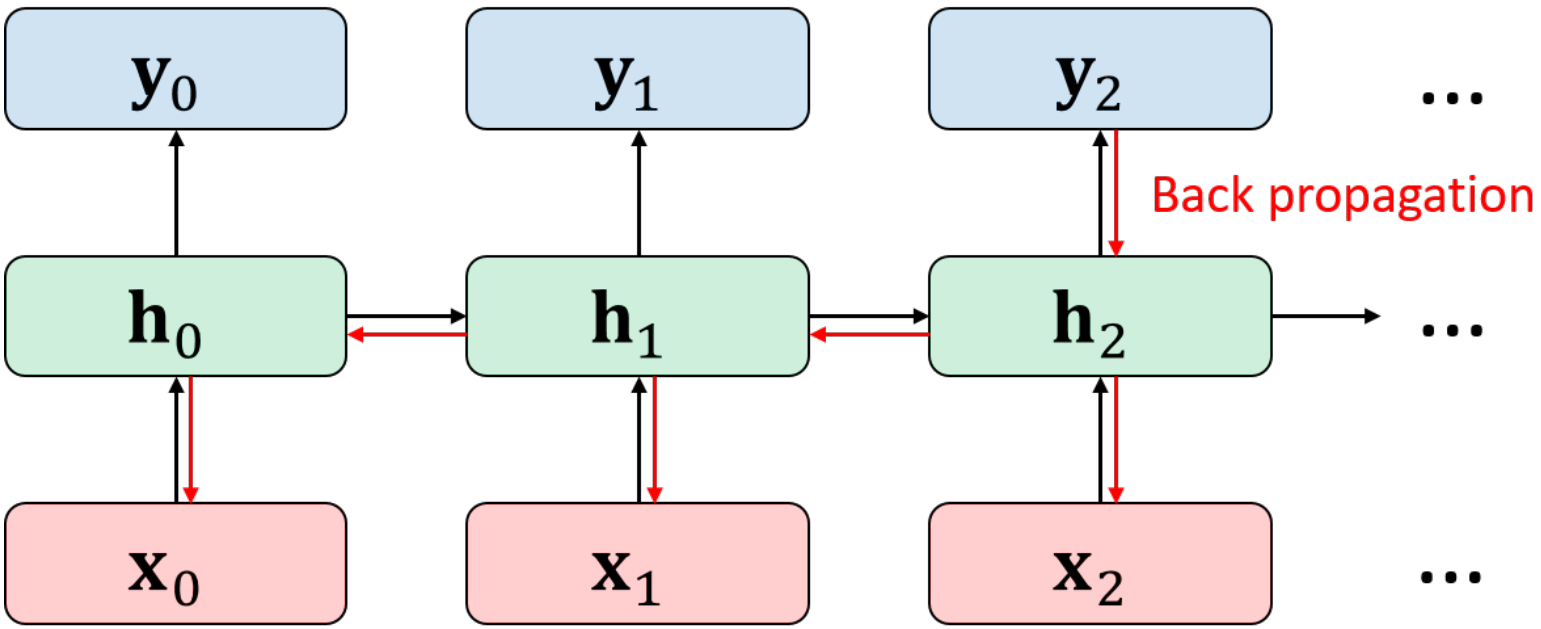

$$\begin{aligned} \mathbf{h}_t &= g(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t + \mathbf{b}_h) \\ \mathbf{y}_t &= \sigma(\mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o) \end{aligned}$$

- $g(\cdot)$: activation function, often *tanh*; $\sigma(\cdot)$: softmax function
- Same functions (model parameters) used at every time step!

**Note: every time step can have an output! The output of one time step can be input to the next time step**

# RNN training

- Multiply same matrix at each time step during forward prop
- Inputs from many time steps back can affect output $y_t$
- Multiply the same matrix at each time step during backprop



# RNN training

- Training RNN is hard
- Similar but simpler RNN formulation:

$$\begin{aligned} \mathbf{h}_t &= \mathbf{W} g(\mathbf{h}_{t-1}) + \mathbf{W}_i \mathbf{x}_t \\ \mathbf{y}_t &= \sigma(\mathbf{W}_o \mathbf{h}_t) \end{aligned}$$

# RNN training

- Training RNN is hard
- Similar but simpler RNN formulation:

$$\begin{aligned} \mathbf{h}_t &= \mathbf{W} g(\mathbf{h}_{t-1}) + \mathbf{W}_i \mathbf{x}_t \\ \mathbf{y}_t &= \sigma(\mathbf{W}_o \mathbf{h}_t) \end{aligned}$$

- Total loss is the sum of loss over all time steps, then

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \mathbf{W}}$$

# RNN training

- Training RNN is hard
- Similar but simpler RNN formulation:

$$\begin{aligned} \mathbf{h}_t &= \mathbf{W} g(\mathbf{h}_{t-1}) + \mathbf{W}_i \mathbf{x}_t \\ \mathbf{y}_t &= \sigma(\mathbf{W}_o \mathbf{h}_t) \end{aligned}$$

- Total loss is the sum of loss over all time steps, then

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \mathbf{W}}$$

- With chain rule:

$$\frac{\partial L_t}{\partial \mathbf{W}} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

# RNN training

- So far:

$$\begin{aligned} \mathbf{h}_t &= \mathbf{W} g(\mathbf{h}_{t-1}) + \mathbf{W}_i \mathbf{x}_t \\ \frac{\partial L_t}{\partial \mathbf{W}} &= \sum_{k=1}^{t} \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}} \end{aligned}$$

- With chain rule again

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

This process may cause **gradient exploding** (梯度爆炸) or **vanishing gradient (梯度消失)** .

# RNN training: gradient exploding

- Solution to gradient exploding (梯度爆炸) : **gradient clipping** (梯度裁剪)

$$\begin{aligned} \text{if } \|\mathbf{g}\| &> v, \\ \text{then } \mathbf{g} &\leftarrow \frac{\mathbf{g}v}{\|\mathbf{g}\|} \text{, where } v \text{ is a threshold} \end{aligned}$$

- Gradient clipping well solved gradient exploding!

# RNN training: vanishing gradient

**Vanishing gradient (梯度消失)** will cause:

- Prediction error at time step $t$ would not tell a far-away previous step $k$ to change during backprop.
- Vanishing gradient makes RNN unable to capture long-term relationship between items far away from each other!

# Long short-term memory (长短时记忆网，LSTM)

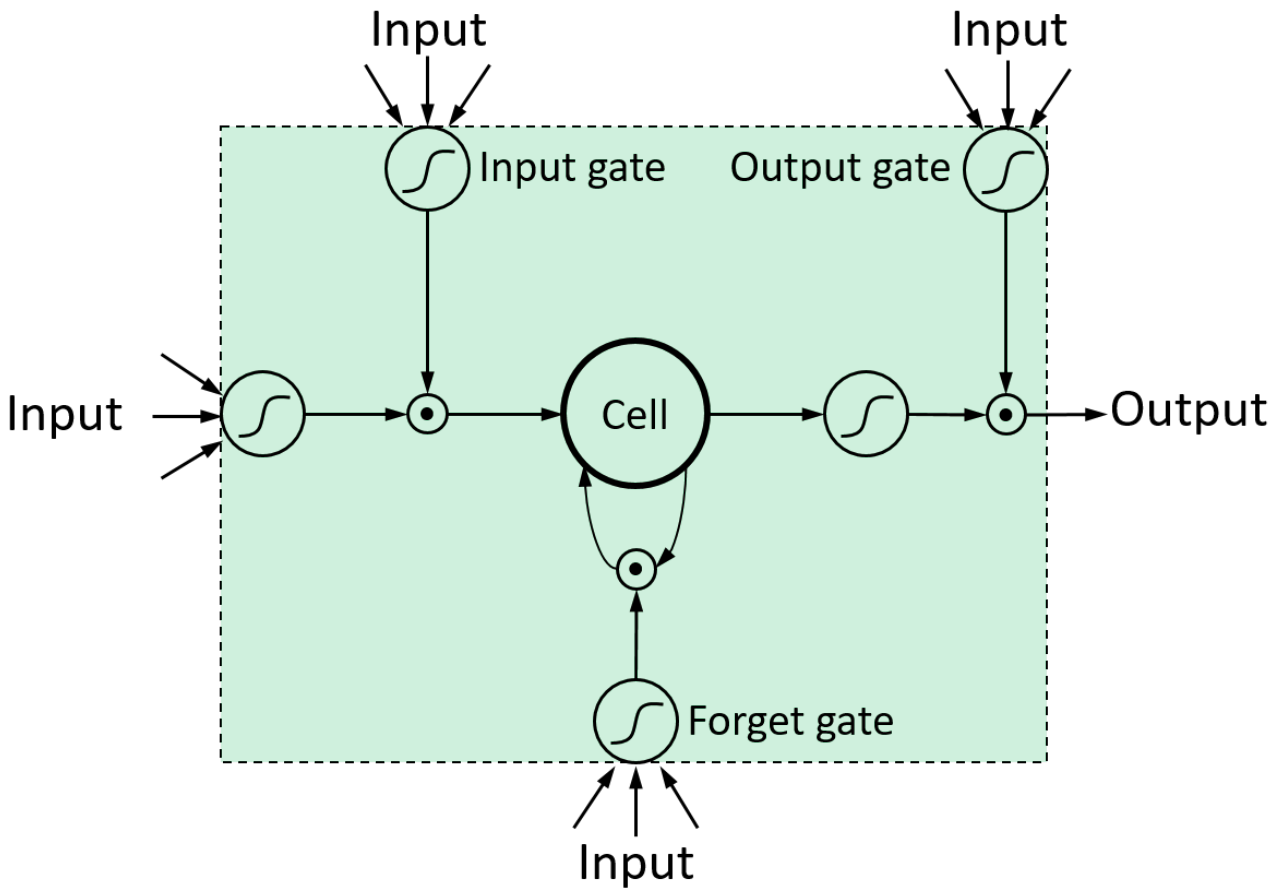
- LSTM as basic unit of RNN reduces gradient vanishing
- “short-term memory”: a small amount of information
- “long”: information can last for a long period of time

# Long short-term memory (长短时记忆网，LSTM)

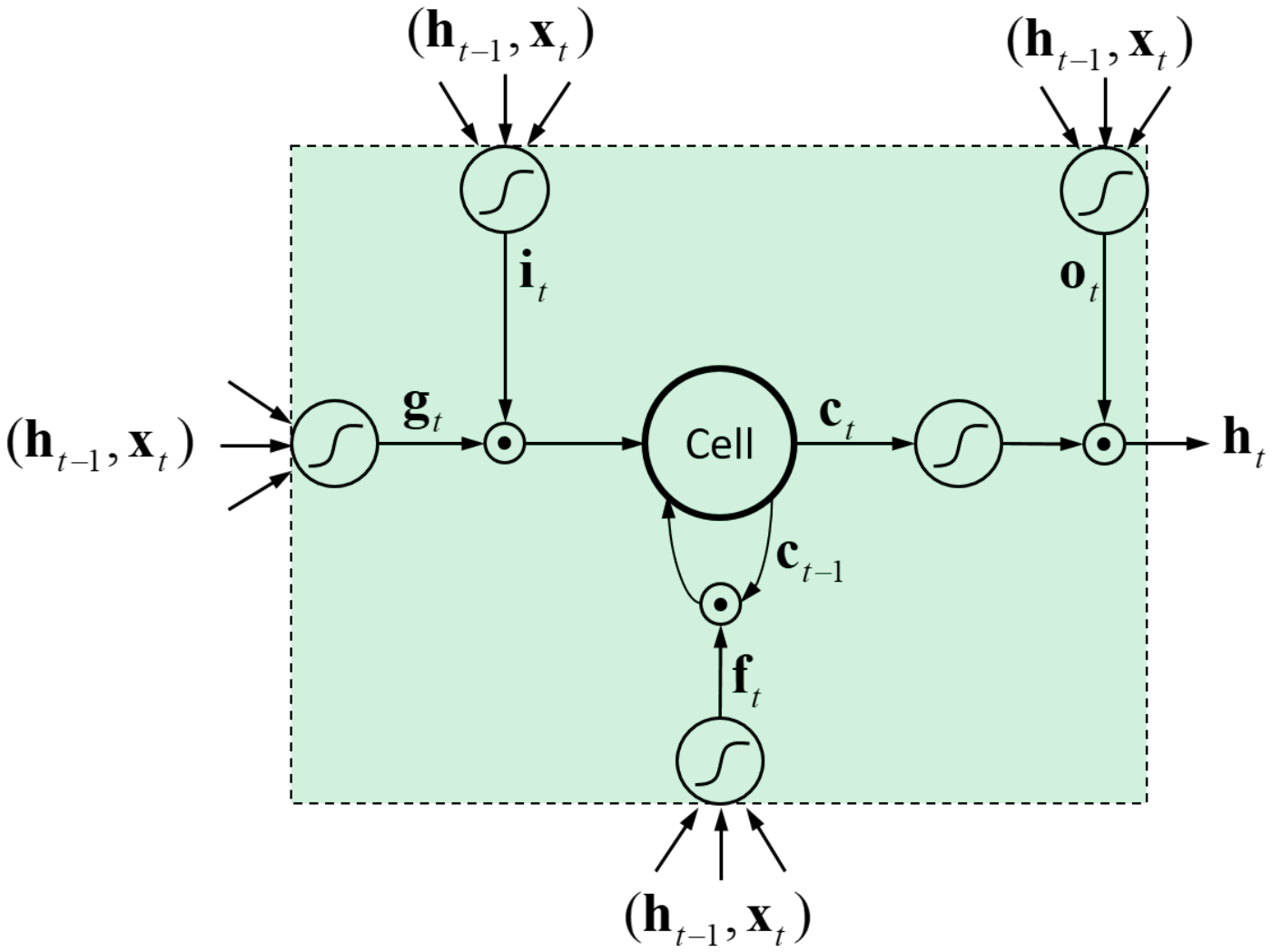
- LSTM as basic unit of RNN reduces gradient vanishing
- “short-term memory”: a small amount of information
- “long”: information can last for a long period of time
- LSTM: cell, input gate, output gate, (un)forget gate
- Cell for “remembering” information over arbitrary time steps, hence the word “memory” in LSTM
- Gates as regulators of the flow of signals through LSTM

# LSTM

- Input gate (输入门): whether/how much to write to cell
- Output gate (输出门): how much to reveal cell
- Forget gate (遗忘门): whether/how much to erase cell



# LSTM


$(\mathbf{h}_{t-1}, \mathbf{x}_t)$
$(\mathbf{h}_{t-1}, \mathbf{x}_t)$
$\mathbf{i}_t$
$\mathbf{o}_t$
$(\mathbf{h}_{t-1}, \mathbf{x}_t)$
$\mathbf{g}_t$
Cell
$\mathbf{c}_t$
$\mathbf{h}_t$
$\mathbf{c}_{t-1}$
$\mathbf{f}_t$
$(\mathbf{h}_{t-1}, \mathbf{x}_t)$

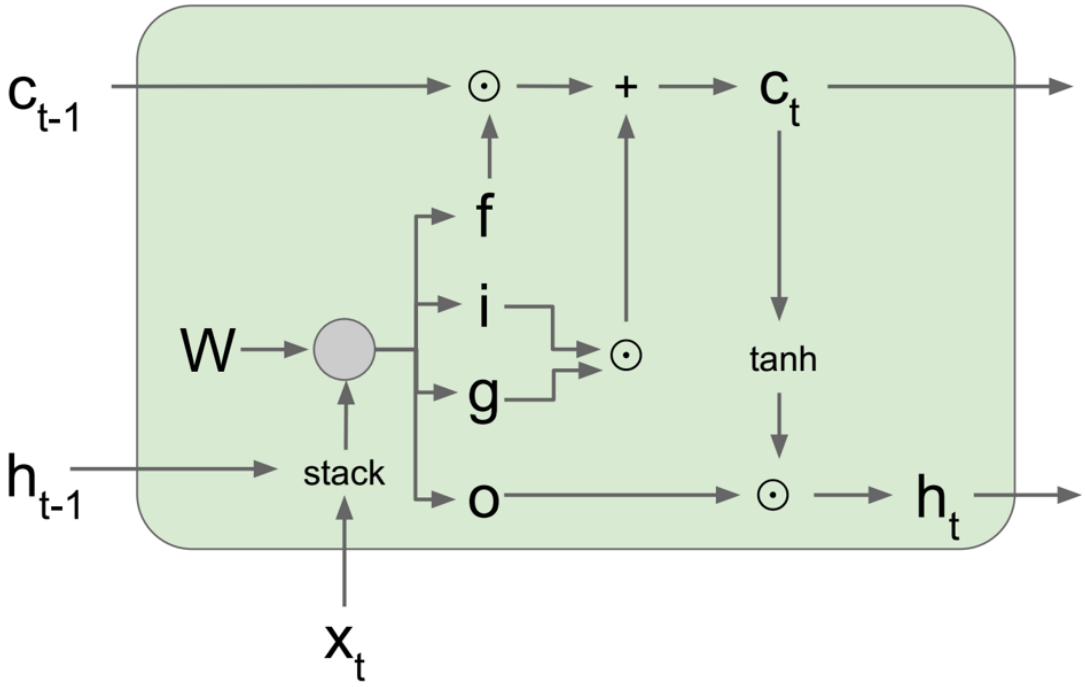
# LSTM

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{g}_t &= tanh(\mathbf{W}_g \mathbf{x}_t + \mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{b}_g) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t)
\end{aligned}
$$

- $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$: input, forget, and output gate; $\sigma$: sigmoid function
- $\mathbf{g}_t$: new signal to update cell
- $\mathbf{c}_t$: updated cell; $\mathbf{h}_t$: new hidden state
- Well chosen activation function (tanh) is critical
- Three times more parameters than RNN

# LSTM

- An alternative diagram representation of LSTM




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

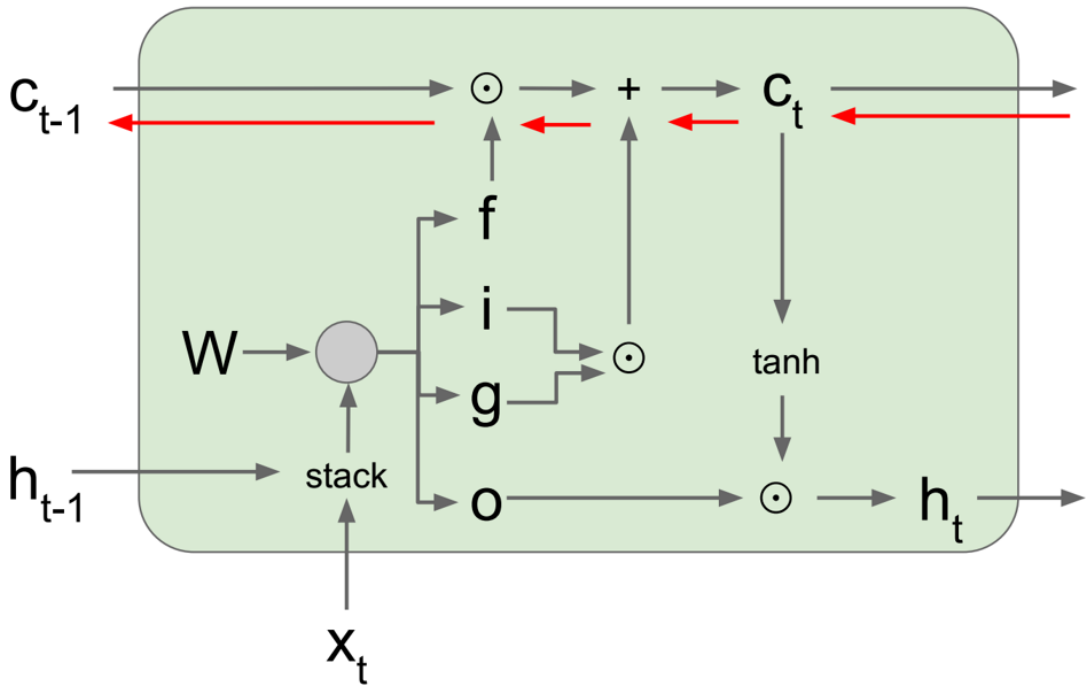
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Figures and content here and in the next 9 slide mainly from Stanford CS231n Lecture 10, 2017

# Why LSTM can reduce gradient vanishing

- Additive path between $\mathbf{c}_t$ and $\mathbf{c}_{t-1}$




Backpropagation from $c_t$ to $c_{t-1}$ only elementwise multiplication by f, no matrix multiply by W

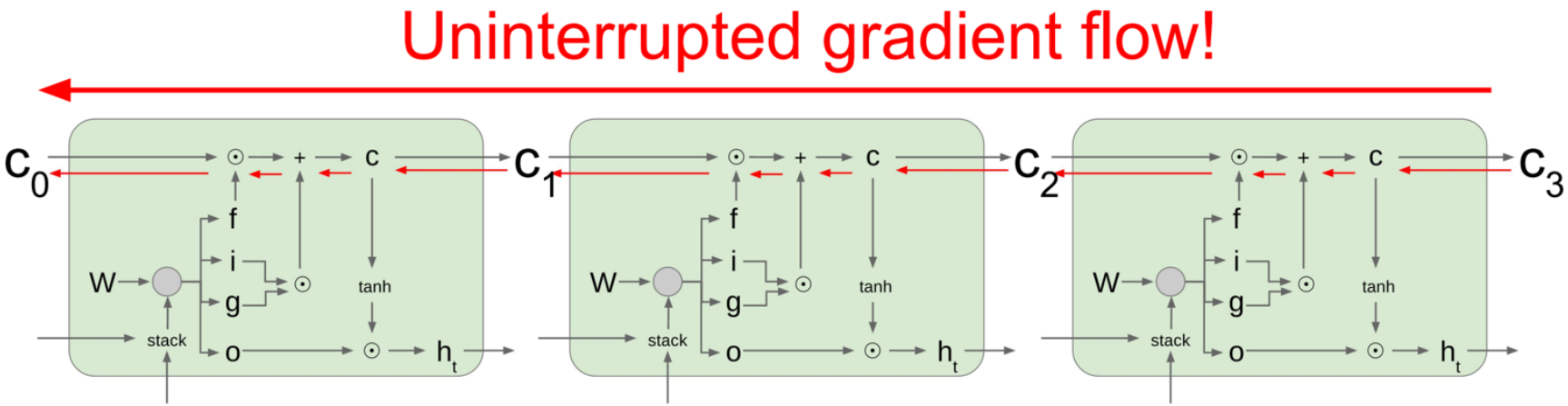
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

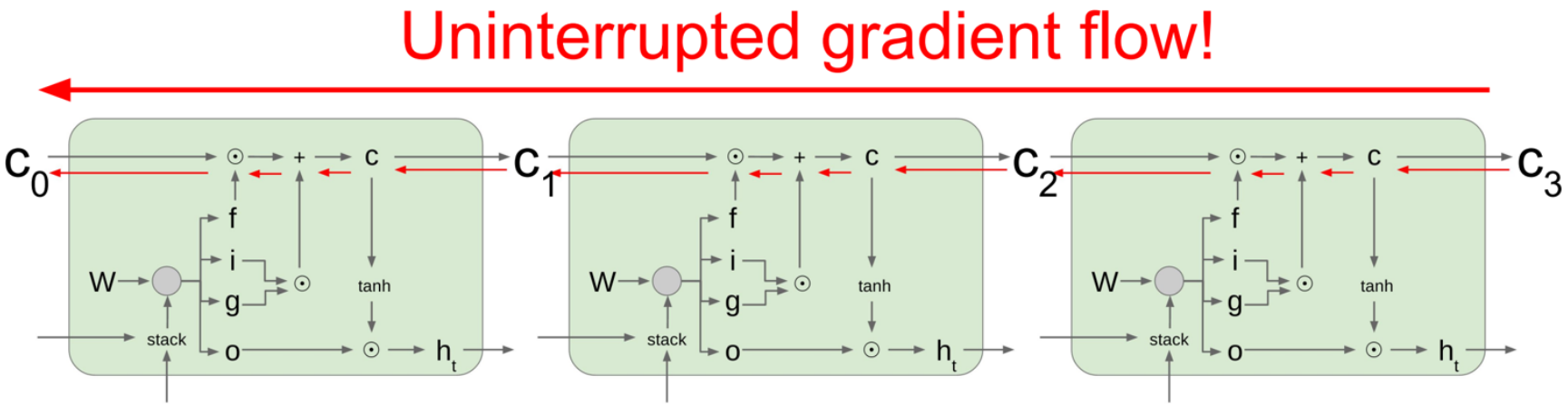
# Why LSTM can reduce gradient vanishing

- Gradient signal can easily back propagate through multiple time steps (if forget gate is open)



# Why LSTM can reduce gradient vanishing

- Gradient signal can easily back propagate through multiple time steps (if forget gate is open)
- Reminder: skip connections in ResNet



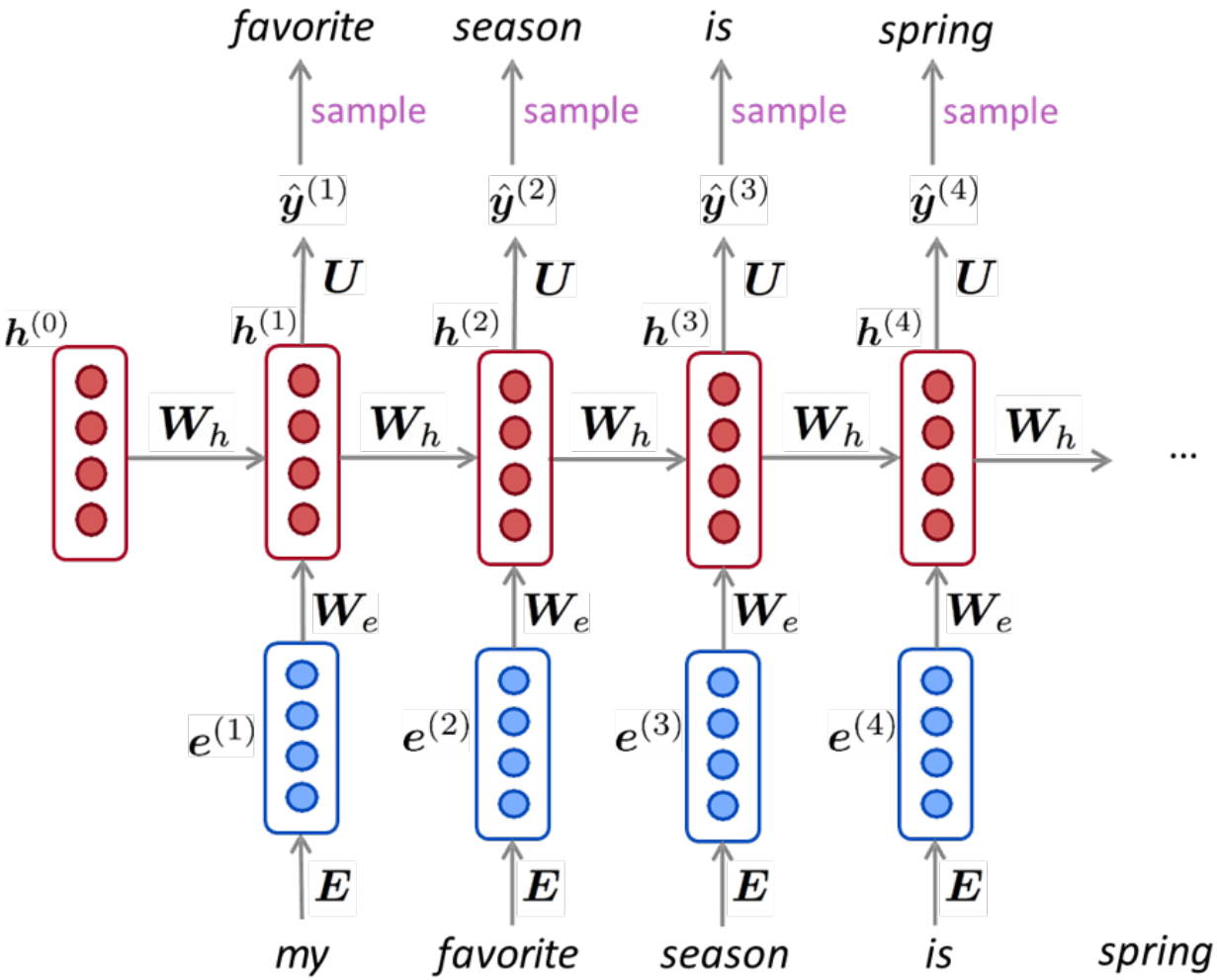
# Gated recurrent unit (GRU)

$$
\begin{aligned}
\mathbf{r}_t &= \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r) \\
\mathbf{z}_t &= \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \\
\hat{\mathbf{h}}_t &= tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \\
\mathbf{h}_t &= \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \hat{\mathbf{h}}_t
\end{aligned}
$$

- One gate less than LSTM, so fewer parameters
- No "cell", only hidden vector $\mathbf{h}_t$ passed to next unit
- No systematic difference between GRU and LSTM
- People tend to use LSTM more

# RNN for Text Generation

- RNN can be used to generate text through recurrent sampling



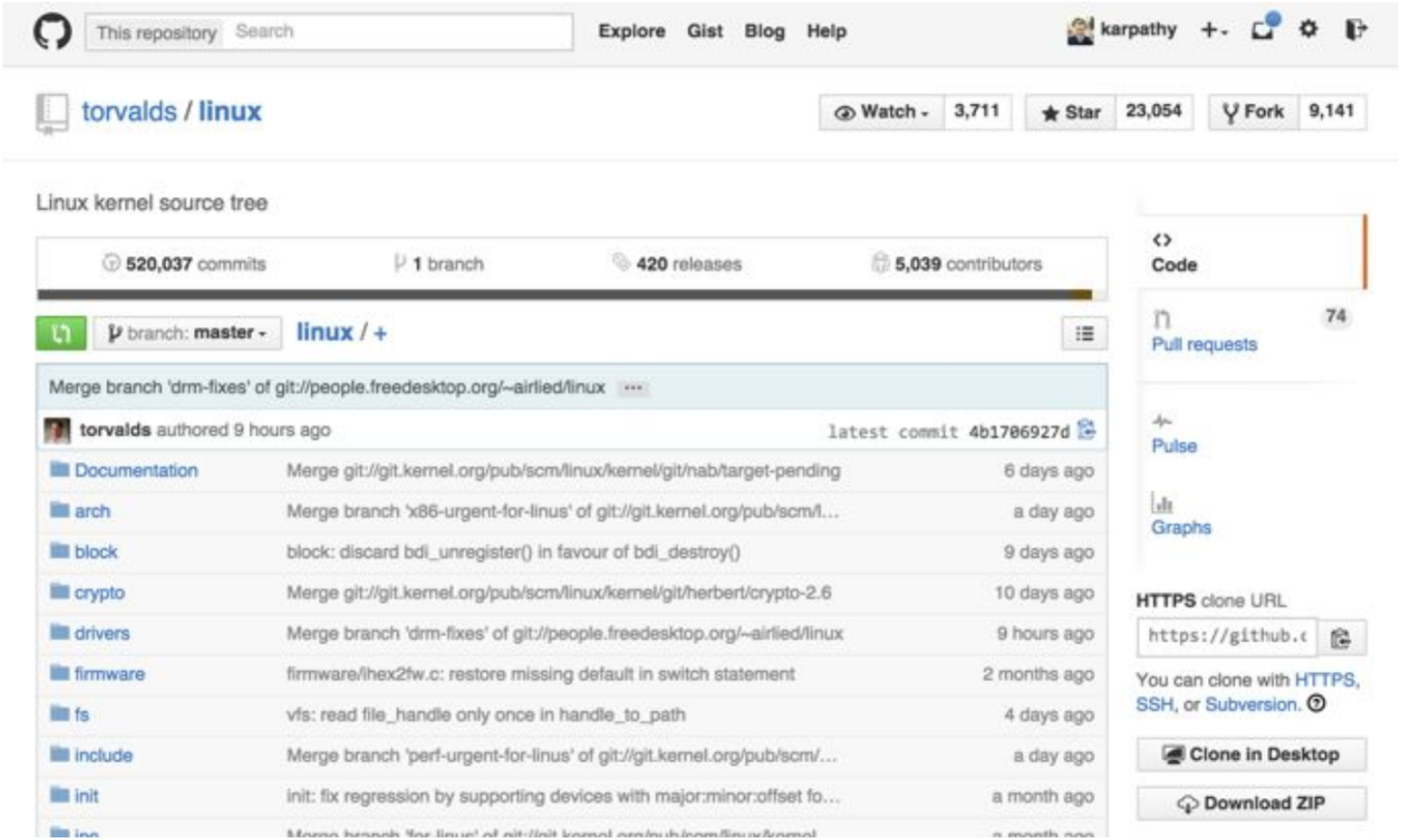
# RNN for Text Generation

- For example, in the text of Obama's speech:

*The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. The promise of the men and women who were still going*

# RNN for Text Generation

- With C source code: predict next character based on previous characters



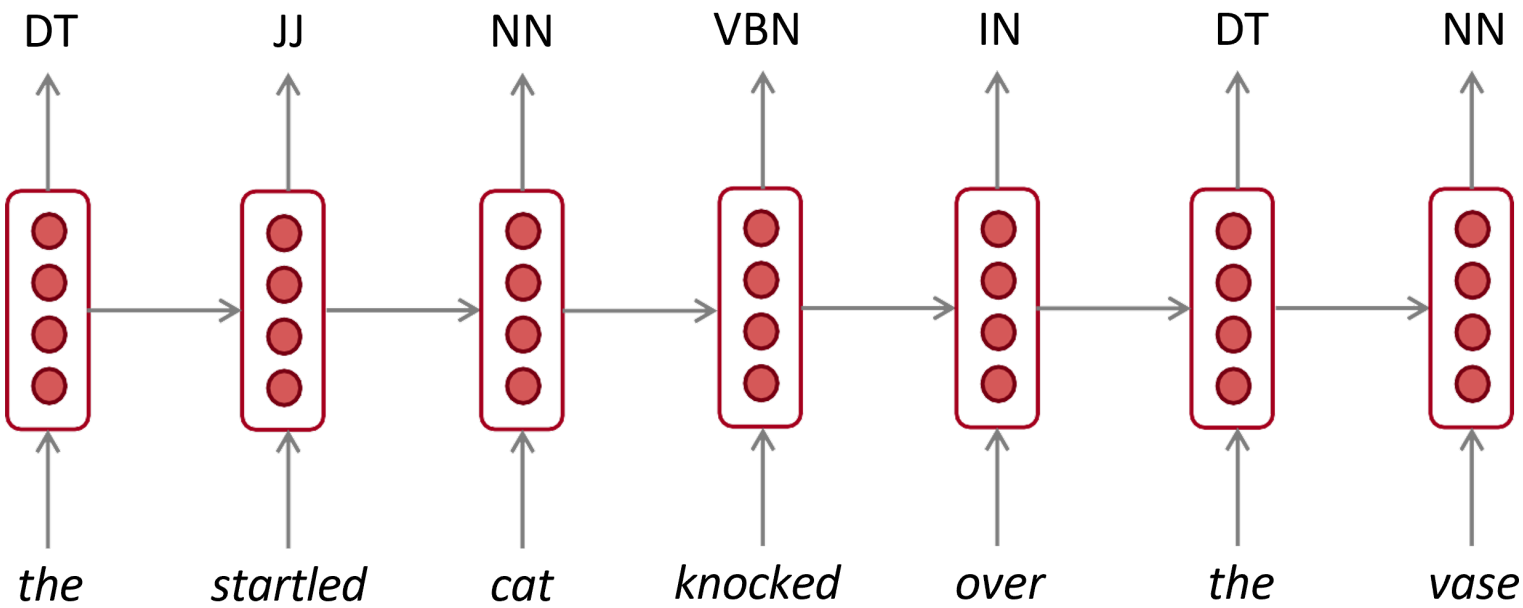
# RNN for Text Generation

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Generated C code

# Applications of RNN

- RNN can be used for sequence tagging e.g., part of speech tagging (词性标注)

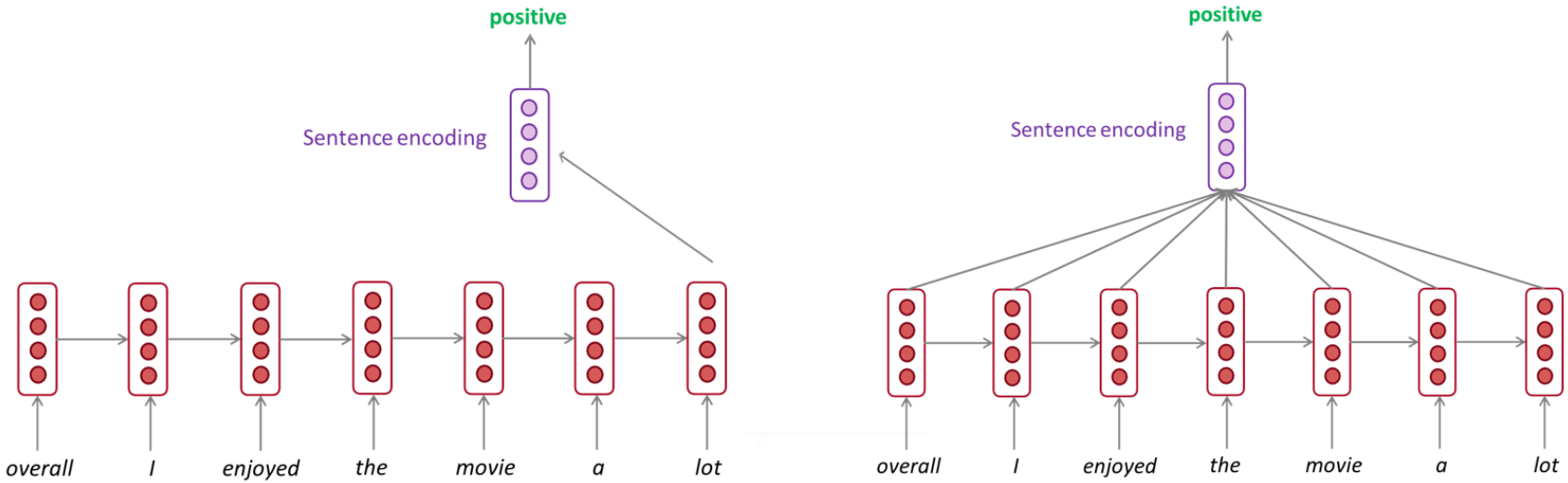


# Applications of RNN

- RNN can also be used for text classification (文本分类)
- How to predict the category?

# Applications of RNN

- RNN can also be used for text classification (文本分类)
- How to predict the category?

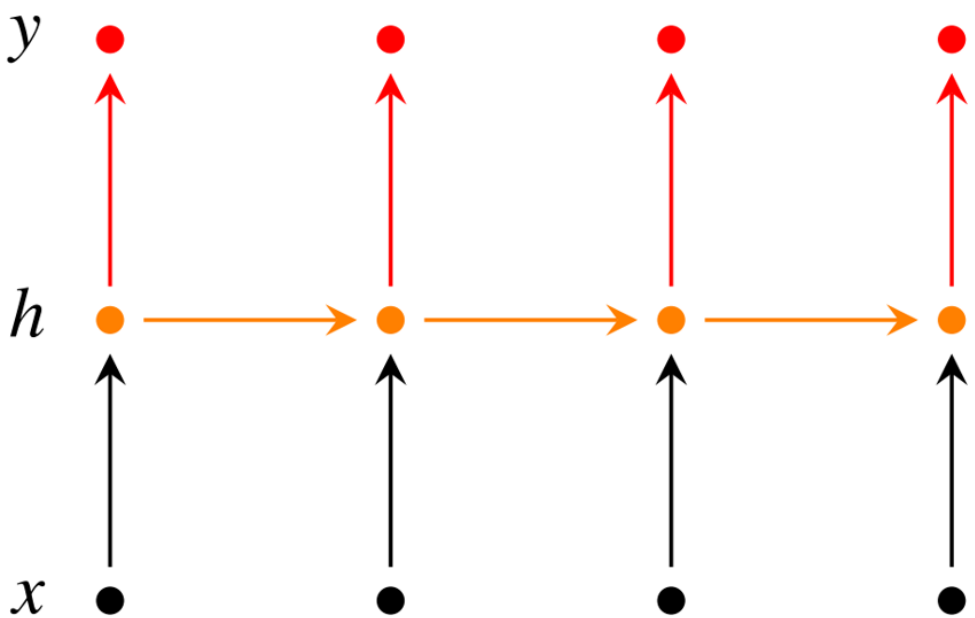


# RNN variants in structure...

RNN can have more complex structures!

# Vanilla RNN

- People may use different notations for RNN.
- $h_t$ summarizes the sentence up to time step $t$.
- Problem: for some tasks, it would be better to incorporate information from both preceding and following words.

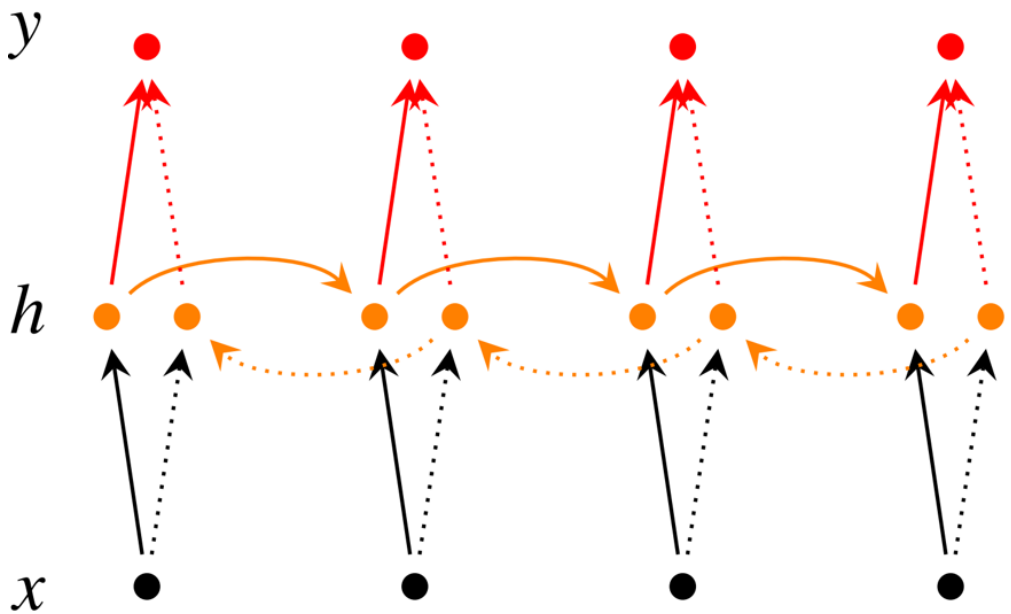



$$h_t = f(Wx_t + Vh_{t-1} + b)$$
$$y_t = g(Uh_t + c)$$

Figures here and in the next 2 slide from Stanford CS224d, Lecture 8, 2016

# Bidirectional RNN (BiRNN)

- Bidirectional RNN captures sequential information from both directions.
- RNN unit could be LSTM or others



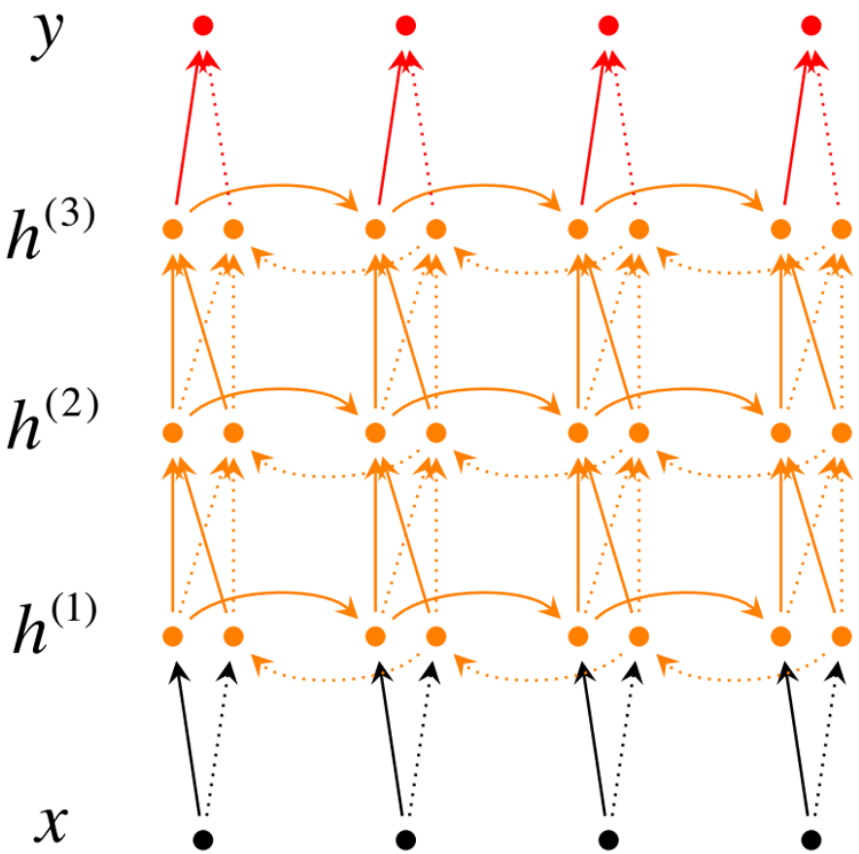

$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

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

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

# Deep bidirectional RNN

- Deep BiRNN: each layer passes an intermediate sequential representation to the next layer.




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

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

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

# Summary

- Word2vec as input to RNN models
- Gradient clipping to reduce exploding issue
- Vanilla RNN may not well capture long-term relationships
- LSTM can capture long-term relationships
- LSTM can reduce gradient vanishing issue
- Different RNN structures/outputs for different apps

Further reading:

- Mikolov, Sutskever, Chen, Corrado, Dean, "Distributed representations of words and phrases and their compositionality", NIPS, 2013
- Hochreiter, Sepp, Schmidhuber, "Long short-term memory", Neural computation, 1997

Thank you!