

# **COMP5212: Machine Learning**

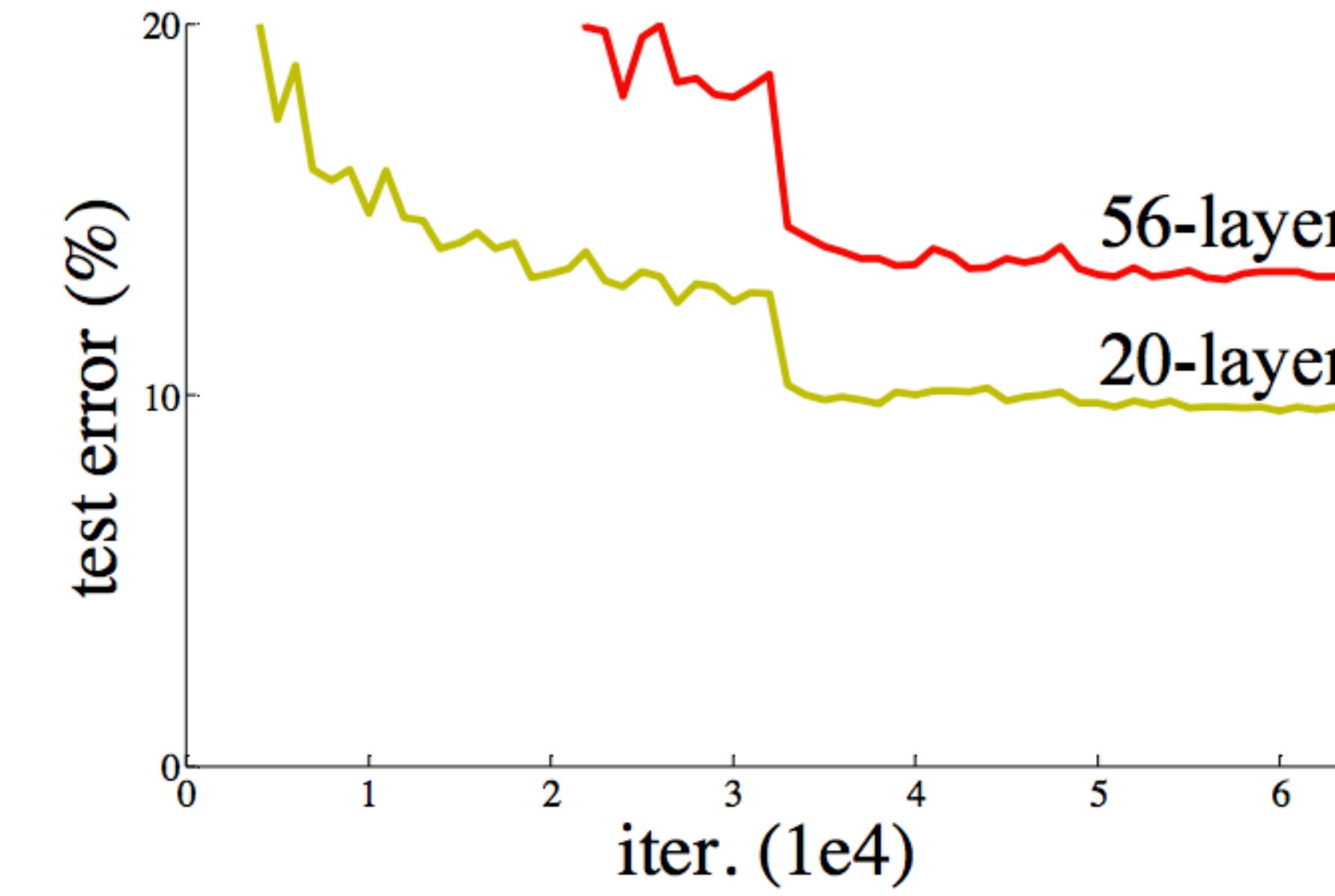
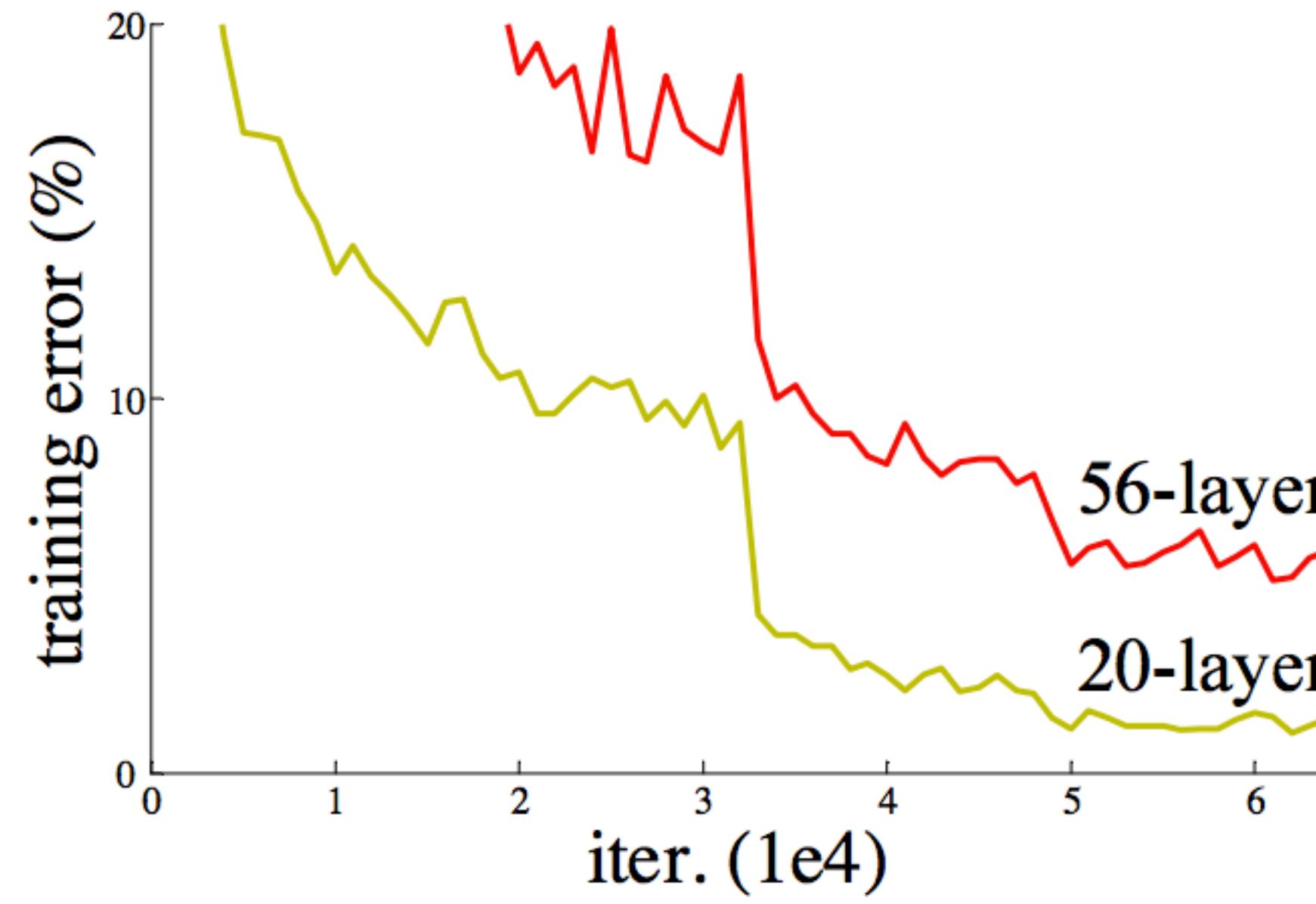
**Lecture 14**

**Minhao Cheng**

# Convolutional Neural Network

## Residual Networks

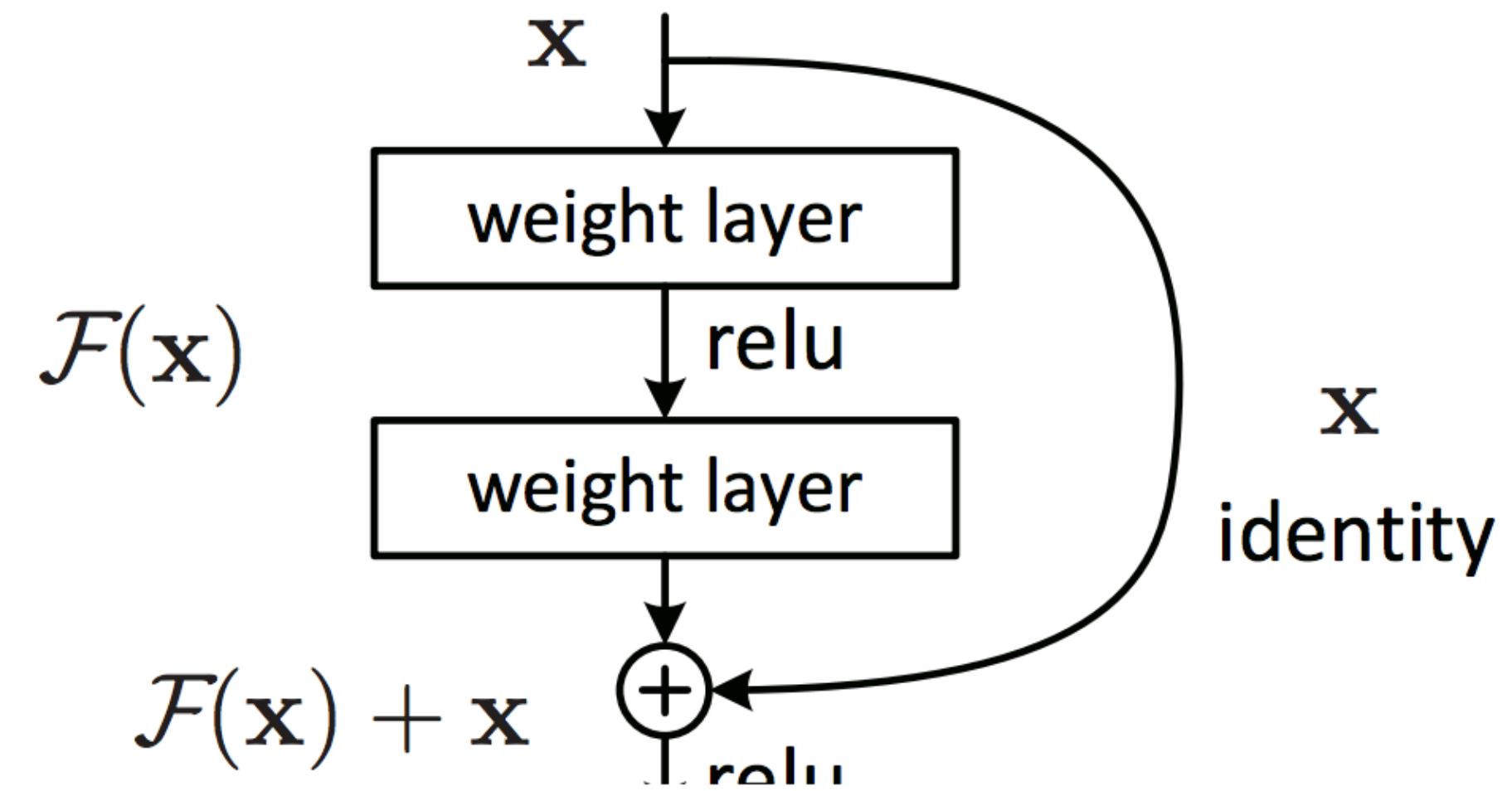
- Very deep convnets do not train well – **vanishing gradient problem**



# Convolutional Neural Network

## Residual Networks

- Key idea: introduce “pass through” into each layer

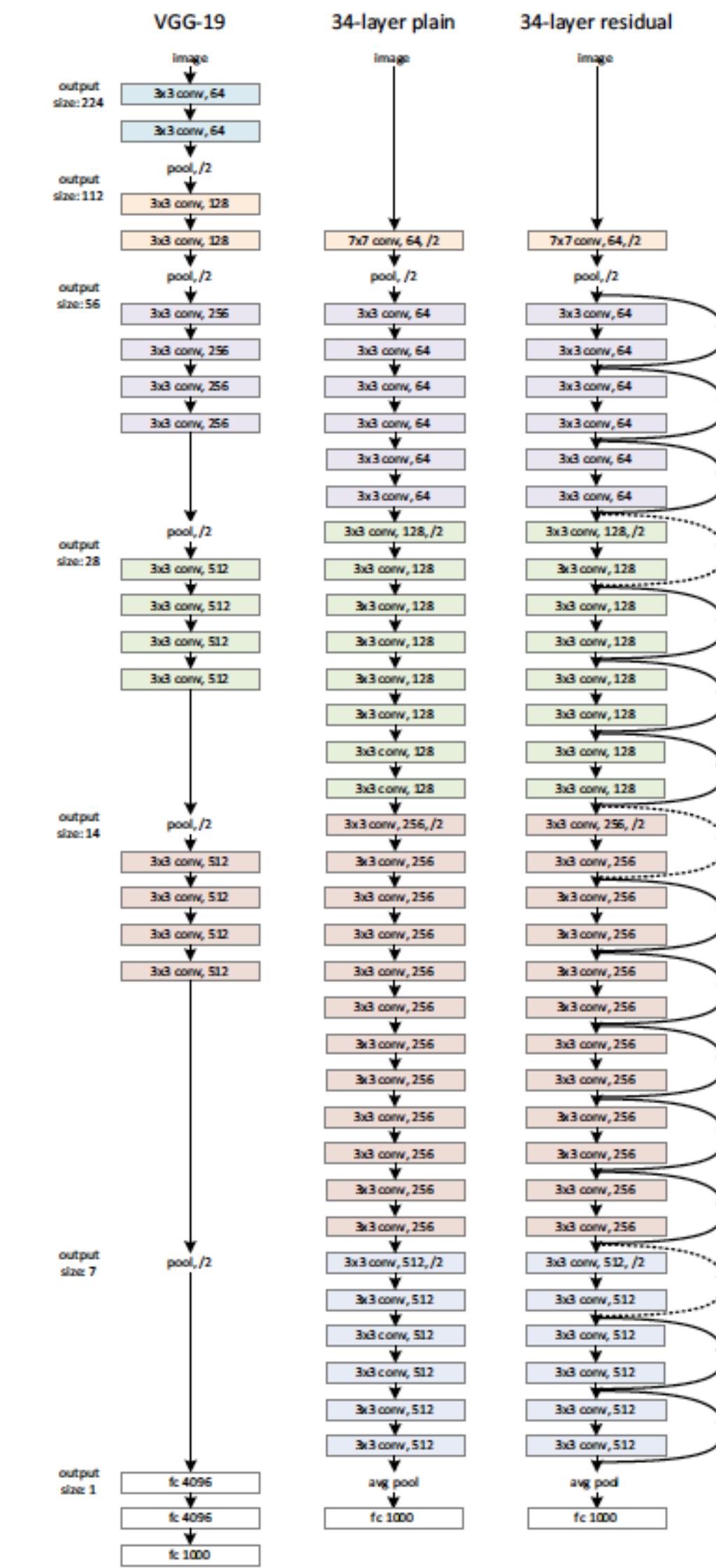


- Thus, only residual needs to be learned

# Convolutional Neural Network Residual Networks

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43 <sup>†</sup>
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	<b>19.38</b>	<b>4.49</b>

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except <sup>†</sup> reported on the test set).



# **Neural Networks for NLP**

# Powerful language model

## Modeling agents, beliefs and actions

*Pat watches a demonstration of a bowling ball and a leaf being dropped at the same time in a vacuum chamber. Pat, who is a physicist, predicts that the bowling ball and the leaf will fall at the same rate.*

Changing the last sentence of the prompt, we get:

*...Pat, who has never seen this demonstration before, predicts that the bowling ball will fall to the ground first. This is incorrect. In a vacuum chamber, there is no air*

# Powerful language model

## Generating code

```
5 // Determine whether the sentiment of text is positive
6 // Use a web service
7 async function isPositive(text: string): Promise<boolean> {
8   const response = await fetch(`http://text-processing.com/api/sentiment/`, {
9     method: "POST",
10    body: `text=${text}`,
11    headers: {
12      "Content-Type": "application/x-www-form-urlencoded",
13    },
14  });
15  const json = await response.json();
16  return json.label === "pos";
17 }
```

# Powerful language model

## Multitask assistant

**I need to throw a dinner party for 6 people who are vegetarian. Can you suggest a 3-course menu with a chocolate dessert?**

[[Microsoft Bing](#)]

(Also see OpenAI's ChatGPT,  
Google's Bard, Anthropic's Claude)

The screenshot shows a Microsoft Bing AI interface. At the top right are icons for thumbs up, thumbs down, and three dots for more options. The AI's response is as follows:

Sure, I can suggest a 3-course menu with a chocolate dessert for your vegetarian dinner party. Here are some possible options:

- Starter: **Wild Mushroom Tartlets with Onion Sauce** <sup>1</sup> or **Vegan Popcorn Tofu Nuggets** <sup>2</sup>
- Main: **Vegan Butternut Squash Mac** <sup>2</sup> or **Vegetarian Three-Bean Chili** <sup>3</sup>
- Dessert: **Chocolate Lava Cake** <sup>4</sup> or **Chocolate Pasta with Chocolate Hazelnut Cream Sauce, White Chocolate Shavings, and Fresh Berries** <sup>4</sup>

I hope this helps you plan your party. Enjoy! 😊

Learn more: [1. booths.co.uk](https://1.booths.co.uk) +10 more

**How to let computer understand  
natural language?**

# Word representation

**Previously commonest NLP solution:** Use, e.g., **WordNet**, a thesaurus containing lists of **synonym sets** and **hypercnyms** (“is a” relationships)

*e.g., synonym sets containing “good”:*

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
                          ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
...
adverb: well, good
adverb: thoroughly, soundly, good
```

*e.g., hypernyms of “panda”:*

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(pandaclosure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

# Word representation

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a **localist** representation

Means one 1, the rest 0s

Such symbols for words can be represented by **one-hot vectors**:

motel = [0 0 0 0 0 0 0 0 0 1 0 0 0]

hotel = [0 0 0 0 0 0 1 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

# Word representation

## Problems

**Example:** in web search, if a user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”

But:

$$\begin{aligned}\text{motel} &= [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0] \\ \text{hotel} &= [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0]\end{aligned}$$

These two vectors are **orthogonal**

There is no natural notion of **similarity** for one-hot vectors!

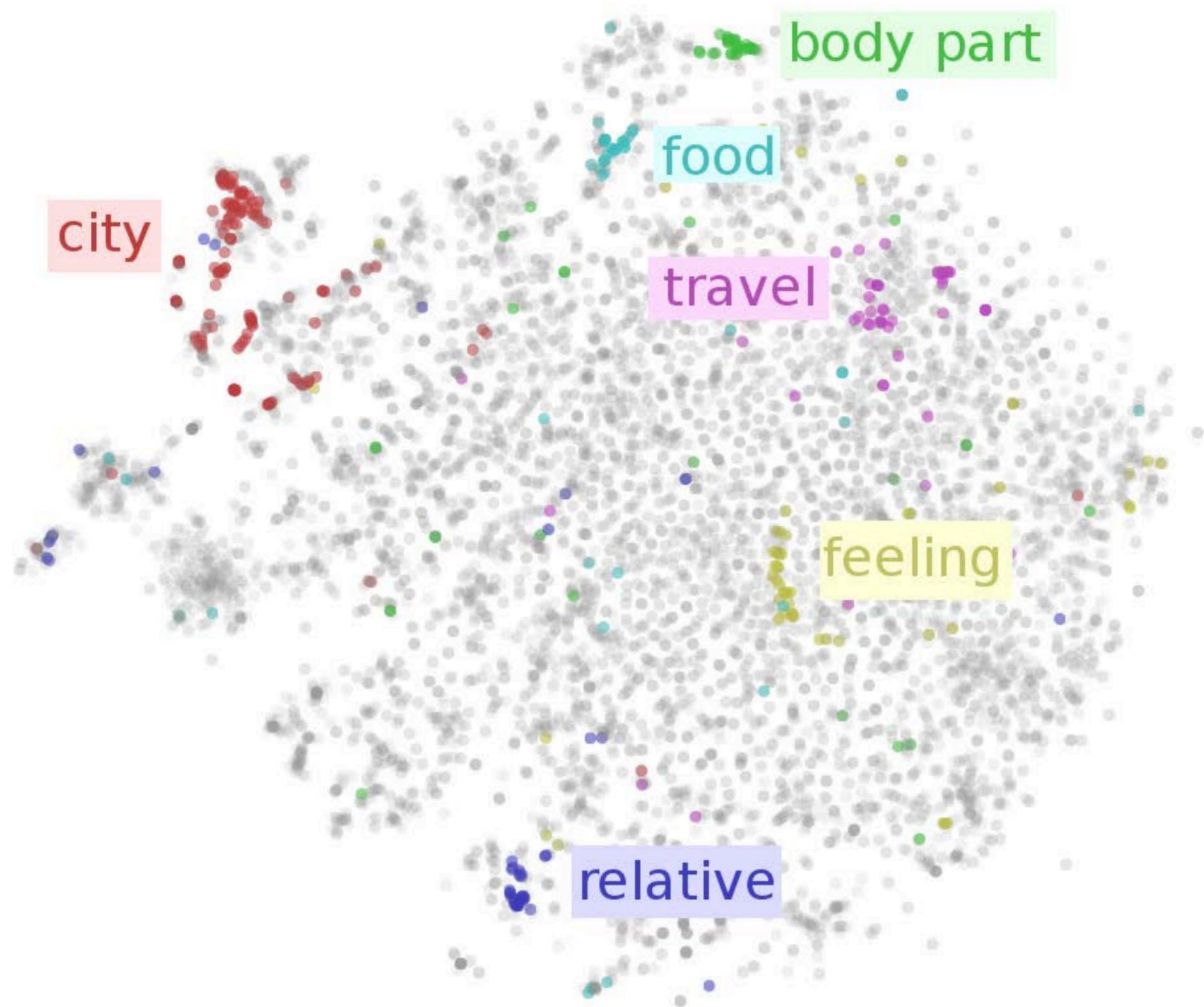
**Solution:**

- Could try to rely on WordNet’s list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

# Learning word embeddings

## word vectors

- Use large (unlabeled) corpus to learn a useful word representation
  - Learn a vector for each word based on the corpus
  - Hopefully the vector represents some semantic meaning
  - Two different perspectives but led to similar results:
    - Glove (Pennington et al., 2014)
    - Word2vec (Mikolov et al., 2013)



# Representing words by their context



- **Distributional semantics:** A word's meaning is given by the words that frequently appear close-by
  - “*You shall know a word by the company it keeps*” (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word  $w$  appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of  $w$  to build up a representation of  $w$

...government debt problems turning into **banking** crises as happened in 2009...

...saying that Europe needs unified **banking** regulation to replace the hodgepodge...

...India has just given its **banking** system a shot in the arm...



These **context words** will represent **banking**

# Learning word embeddings

## Context information

- For each word  $w_i$ , define the “contexts” of the word as the words surrounding it in an  $L$ -sized window:

$w_{i-L-2}, w_{i-L-1}, \underbrace{w_{i-L}, \dots, w_{i-1}}_{\text{contexts of } w_i}, w_i, \underbrace{w_{i+1}, \dots, w_{i+L}}_{\text{contexts of } w_i}, w_{i+L+1}, \dots$

- 

- Get a collection of (word, context) pairs, denoted by  $D$ .

# Learning word embeddings

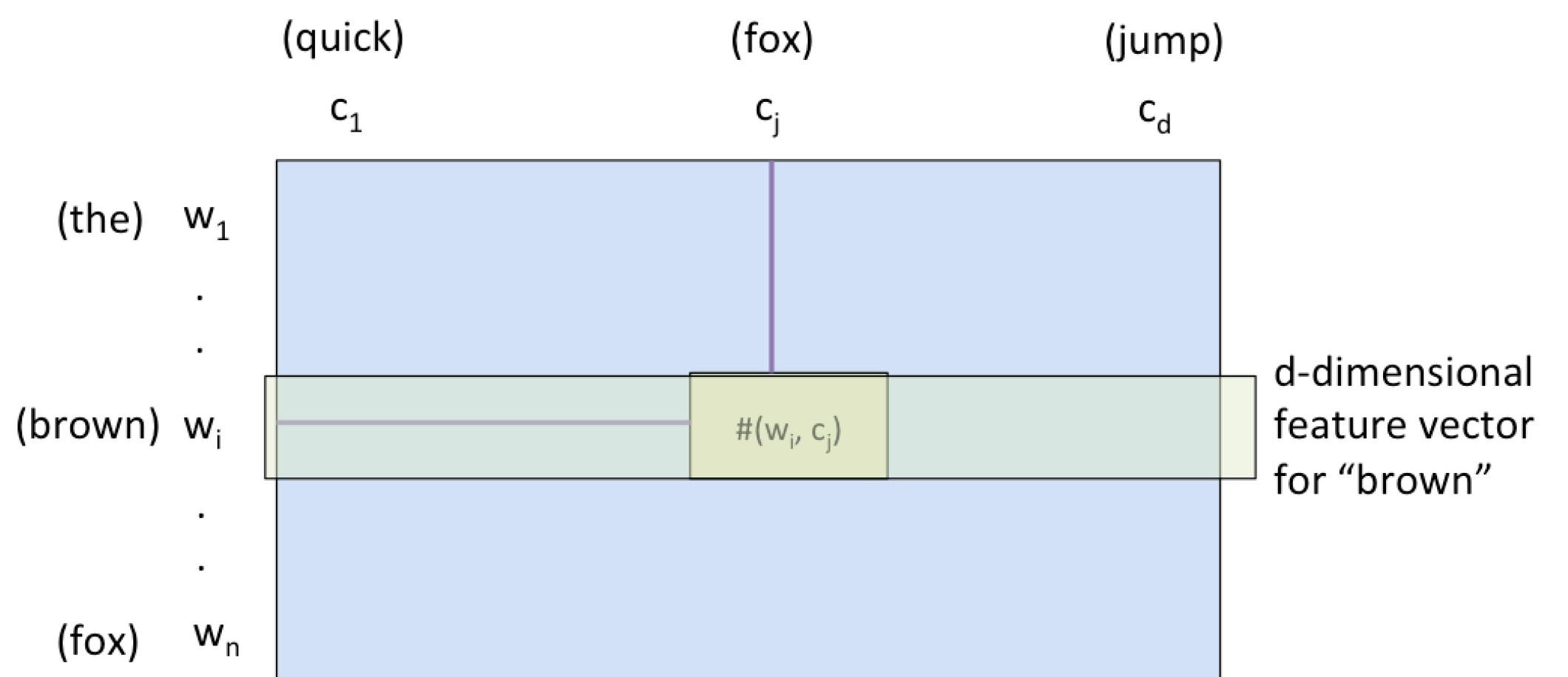
## Examples

Source Text	Training Samples
The <b>quick</b> brown fox jumps over the lazy dog. ➔	(the, quick) (the, brown)
The quick <b>brown</b> fox jumps over the lazy dog. ➔	(quick, the) (quick, brown) (quick, fox)
The quick <b>brown</b> fox <b>jumps</b> over the lazy dog. ➔	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox <b>jumps</b> over the lazy dog. ➔	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

# Learning word embeddings

## Use bag-of-word model

- Idea 1: Use the bag-of-word model to ``describe'' each word
- Assume we have context words  $c_1, \dots, c_d$  in the corpus, compute
  - $\#(w, c_i) :=$  number of times the pair  $(w, c_i)$  appears in  $D$
- For each word  $w$ , form a  $d$ -dimensional (sparse) vector to describe  $w$ 
  - $\#(w, c_1), \dots, \#(w, c_d)$ ,



# Learning word embeddings

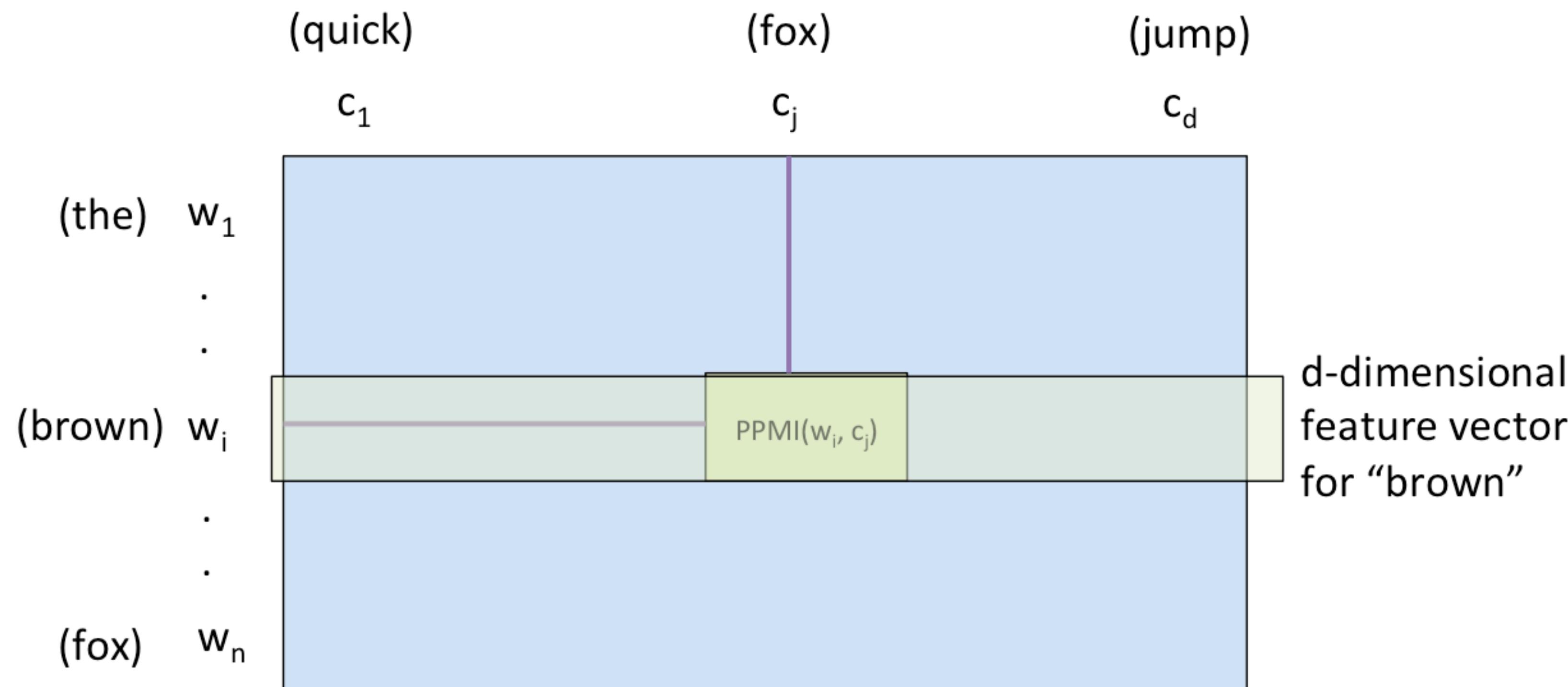
## PMI/PPMI Representation

- Instead of using co-occurrence count  $\#(w, c)$ , we can define pointwise mutual information:

- $\text{PMI}(w, c) = \log\left(\frac{\hat{P}(w, c)}{\hat{P}(w)\hat{P}(c)}\right) = \log\frac{\#(w, c)}{\#(w)\#(c)}D,$
- $\#(w) = \sum_c \#(w, c)$ : number of times word  $w$  occurred in  $D$
- $\#(c) = \sum_w \#(w, c)$ : number of times context  $c$  occurred
- $D$  : number of pairs in  $D$
- Positive PMI (PPMI) usually achieves better performance:
  - $\text{PPMI}(w, c) = \max(\text{PMI}(w, c), 0)$
  - $M^{\text{PPMI}}$ : a  $n$  by  $d$  word feature matrix, each row is a word and each column is a context

# Learning word embeddings

## PPMI Matrix



# Learning word embeddings

## Generalized Low-rank Embedding

- SVD basis will minimize
  - $\min_{W,V} \|M^{\text{PPMI}} - WV^T\|_F^2$
- Glove (Pennington et al., 2014)
  - Negative sampling (less weights to 0s in  $M^{\text{PPMI}}$ )
  - Adding bias term:
    - $M^{\text{PPMI}} \approx WV^T + b_w e^T + e b_c^T$
  - Use  $W$  or  $V$  as the word embedding matrix

# Learning word embeddings

## Word2vec (Mikolov et al., 2013)

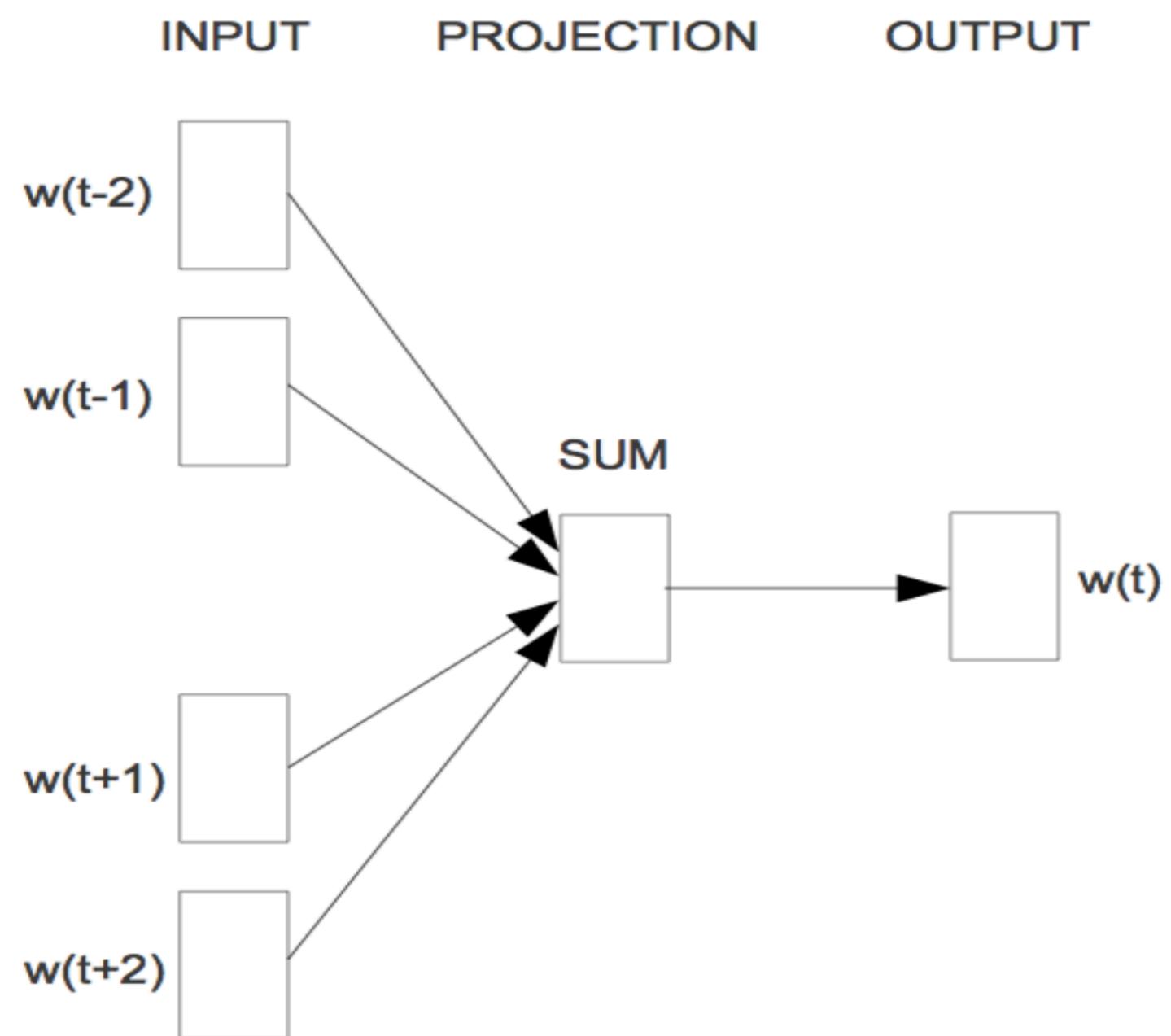
- A neural network model for learning word embeddings
- Main idea:
  - Predict the target words based on the neighbors (CBOW)
  - Predict neighbors given the target words (Skip-gram)



# Learning word embeddings

## CBOW (Continuous Bag-of-Words model)

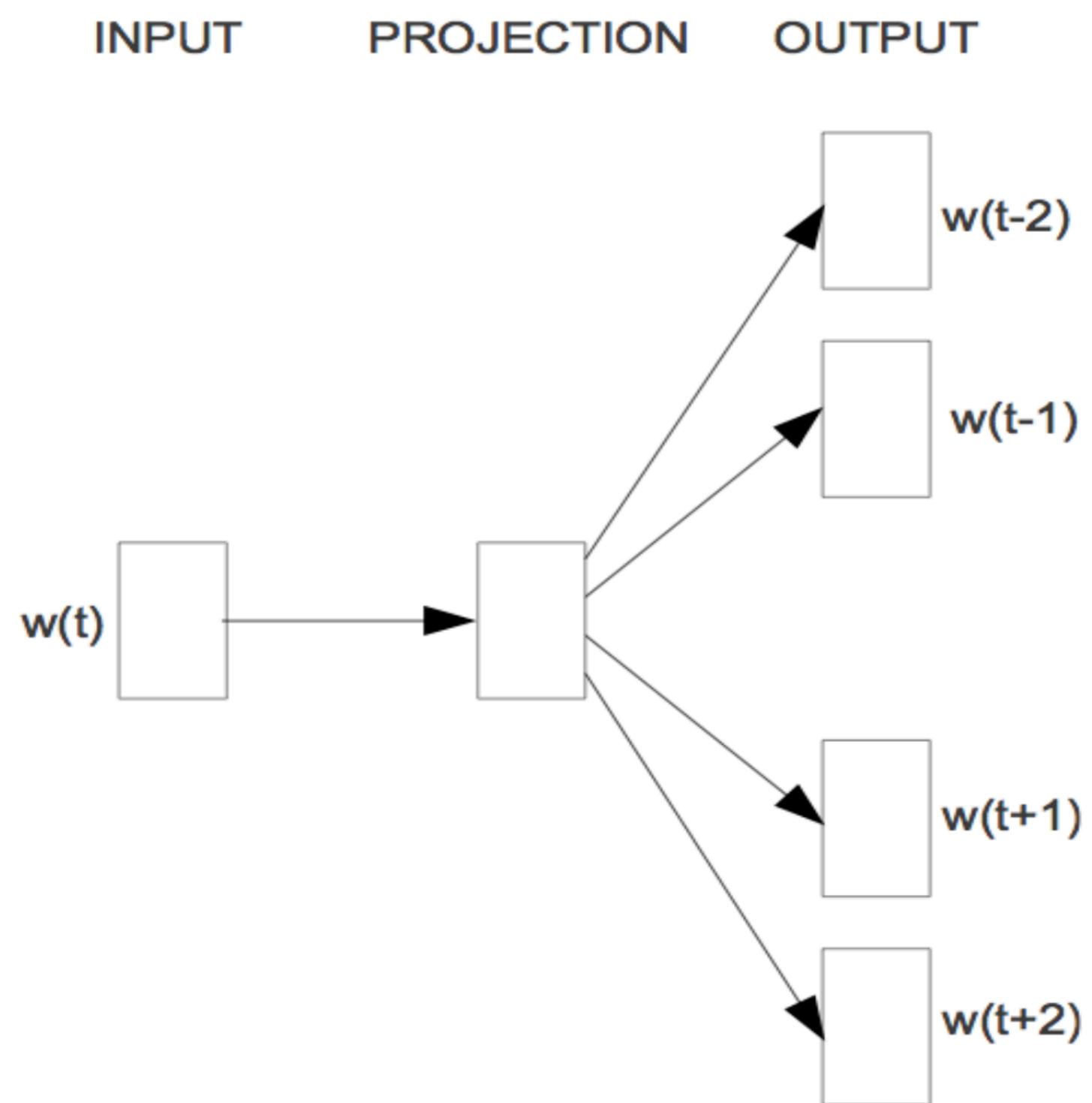
- Predict the target words based on the neighbors



# Learning word embeddings

## Skip-gram

- Predict neighbors using target word



# Learning word embeddings

## More on skip-gram

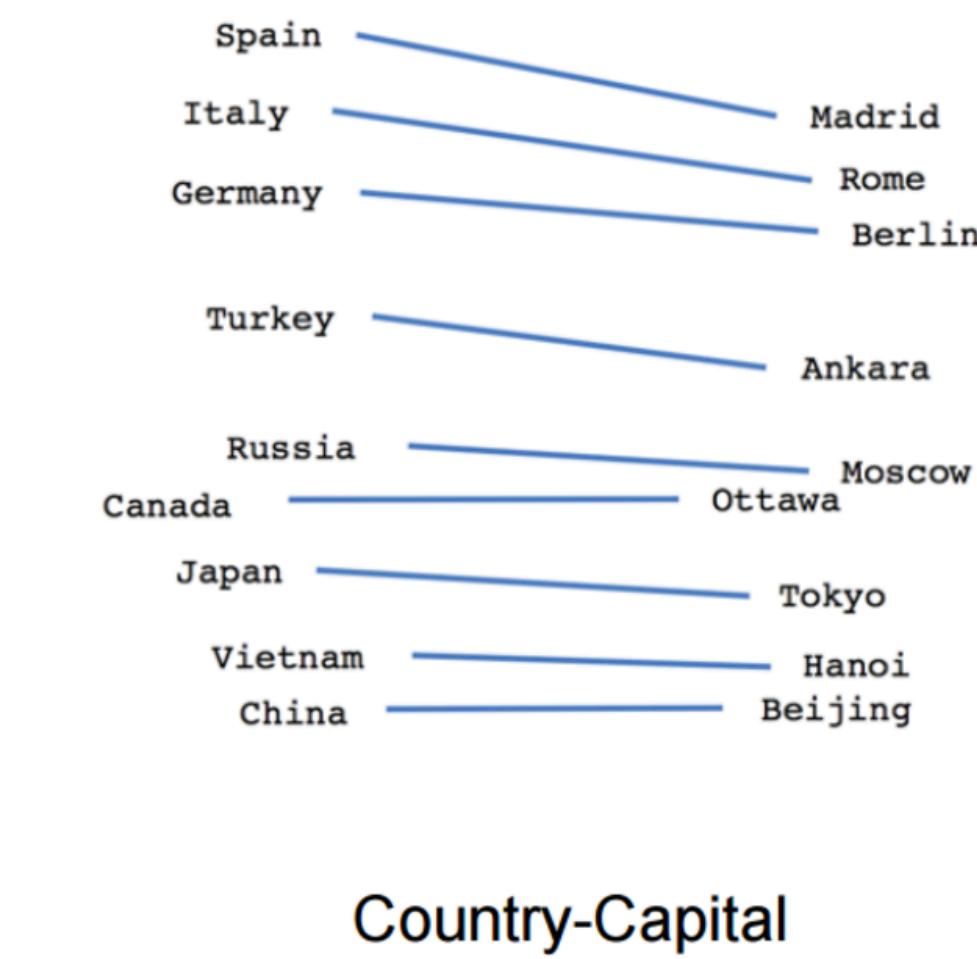
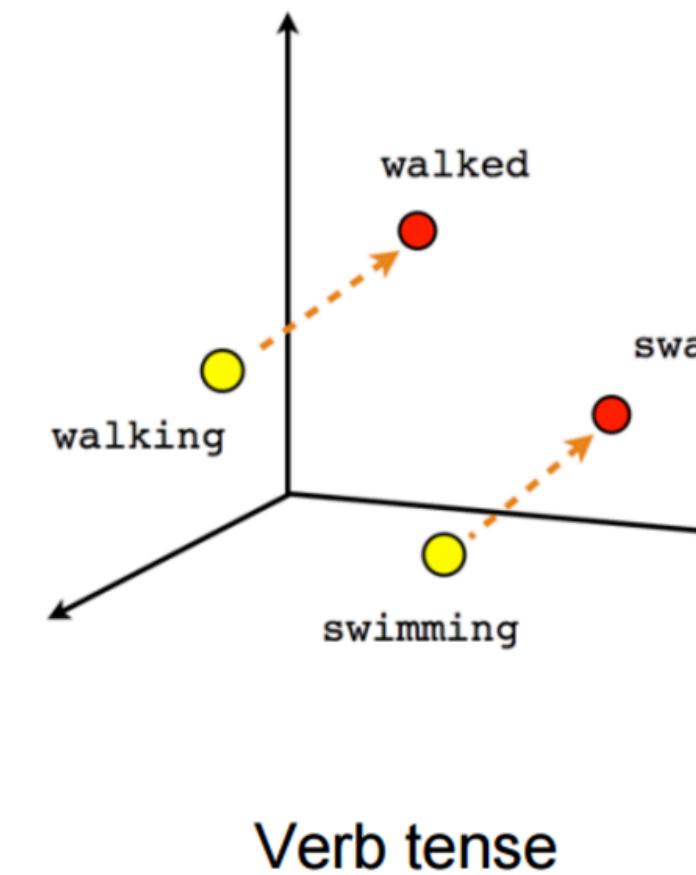
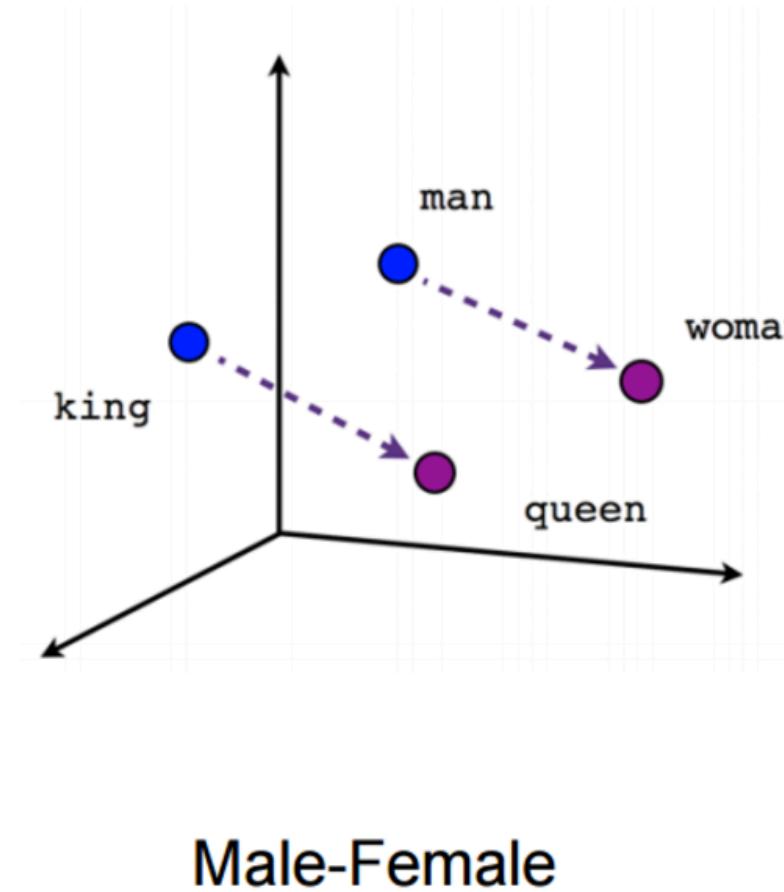
- Learn the probability  $P(w_{t+j} | w_t)$ : the probability to see  $w_{t+j}$  in target word  $w_t$ 's neighborhood
- Every word has two embeddings:
  - $v_i$  serves as the role of target
  - $u_i$  serves as the role of context
- Model probability as softmax:

$$\bullet \quad P(o | c) = \frac{e^{u_o^T v_c}}{\sum_{w=1}^W e^{u_w^T v_c}}$$

# Learning word embeddings

## Results

- The low-dimensional embeddings are (often) meaningful:



# Representation for sentence/document

## Word vectors + linear model

- Example: text classification (e.g., sentiment prediction, review score prediction)
- Linear model:  $y \approx \text{sign}(w^T x)$  (e.g., by linear SVM/logistic regression)
- $w_i$ : the “contribution” of each word

# Representation for sentence/document

## Word vectors + Fully connected network

- $f(x) = W_L \sigma(W_{L-1} \cdots \sigma(W_0 x))$
- The first layer  $W_0$  is a  $d_1$  by  $d$  matrix:
  - Each column  $w_i$  is a  $d_1$  dimensional representation of  $i$ -th word (word embedding)
  - $W_0 x = x_1 w_1 + x_2 w_2 + \cdots + x_d w_d$  is a linear combination of these vectors
  - $W_0$  is also called the word embedding matrix
  - Final prediction can be viewed as an  $L - 1$  layer network on  $W_0 x$  (average of word embeddings)
- Not capturing the sequential information

# Recurrent Neural Network

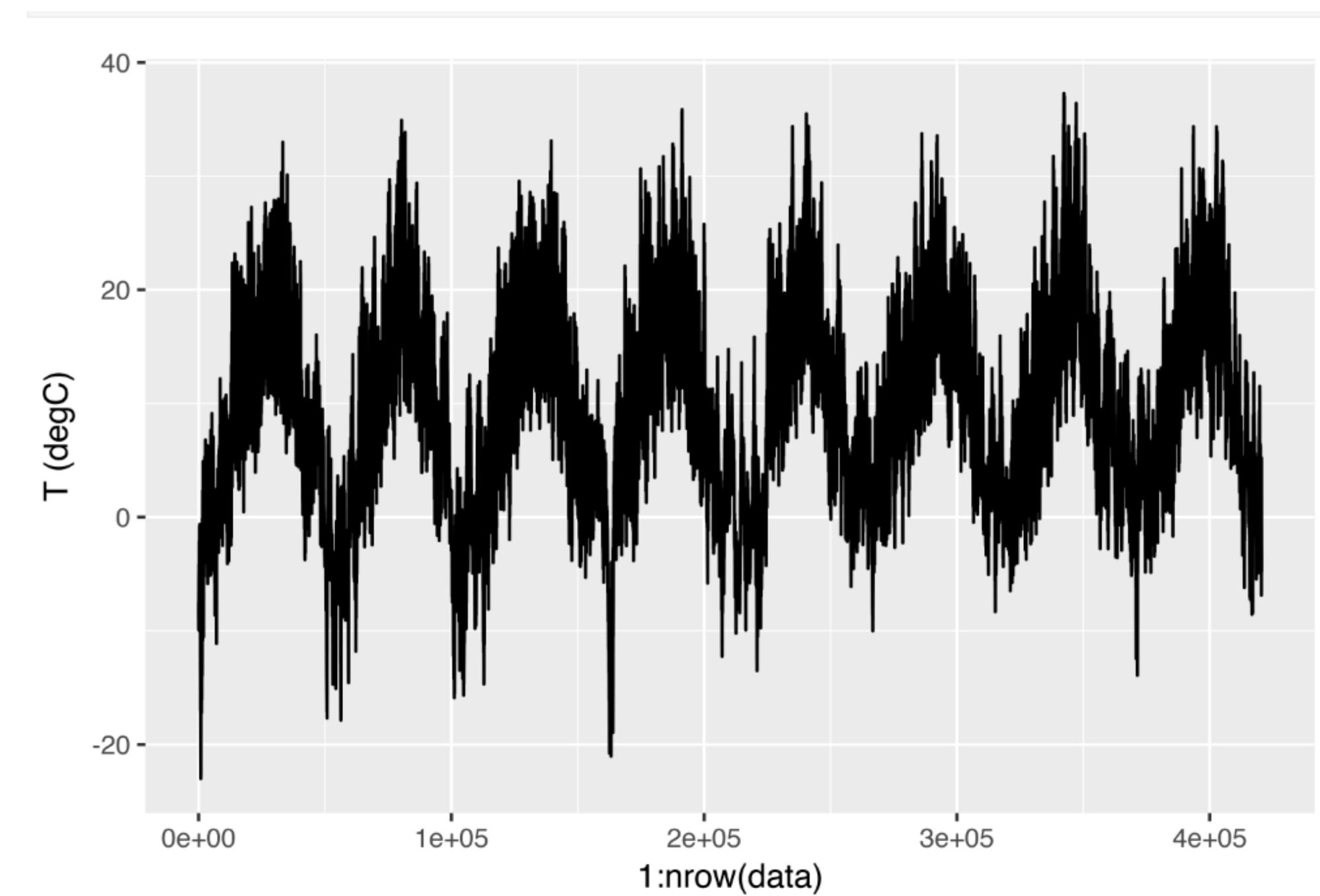
## Time series/Sequence data

- Input:  $\{x_1, x_2, \dots, x_T\}$ 
  - Each  $x_t$  is the feature at time step  $t$
  - Each  $x_t$  can be a  $d$ -dimensional vector
- Output:  $\{y_1, y_2, \dots, y_T\}$ 
  - Each  $y_t$  is the output at step  $t$
  - Multi-class output or Regression output:
    - $y_t \in \{1, 2, \dots, L\}$  or  $y_t \in \mathbb{R}$

# Recurrent Neural Network

## Example: Time Series Prediction

- Climate Data:
  - $x_t$ : temperature at time  $t$
  - $y_t$ : temperature (or temperature change) at time  $t + 1$
- Stock Price: Predicting stock price



# Recurrent Neural Network

## Example: Language Modeling

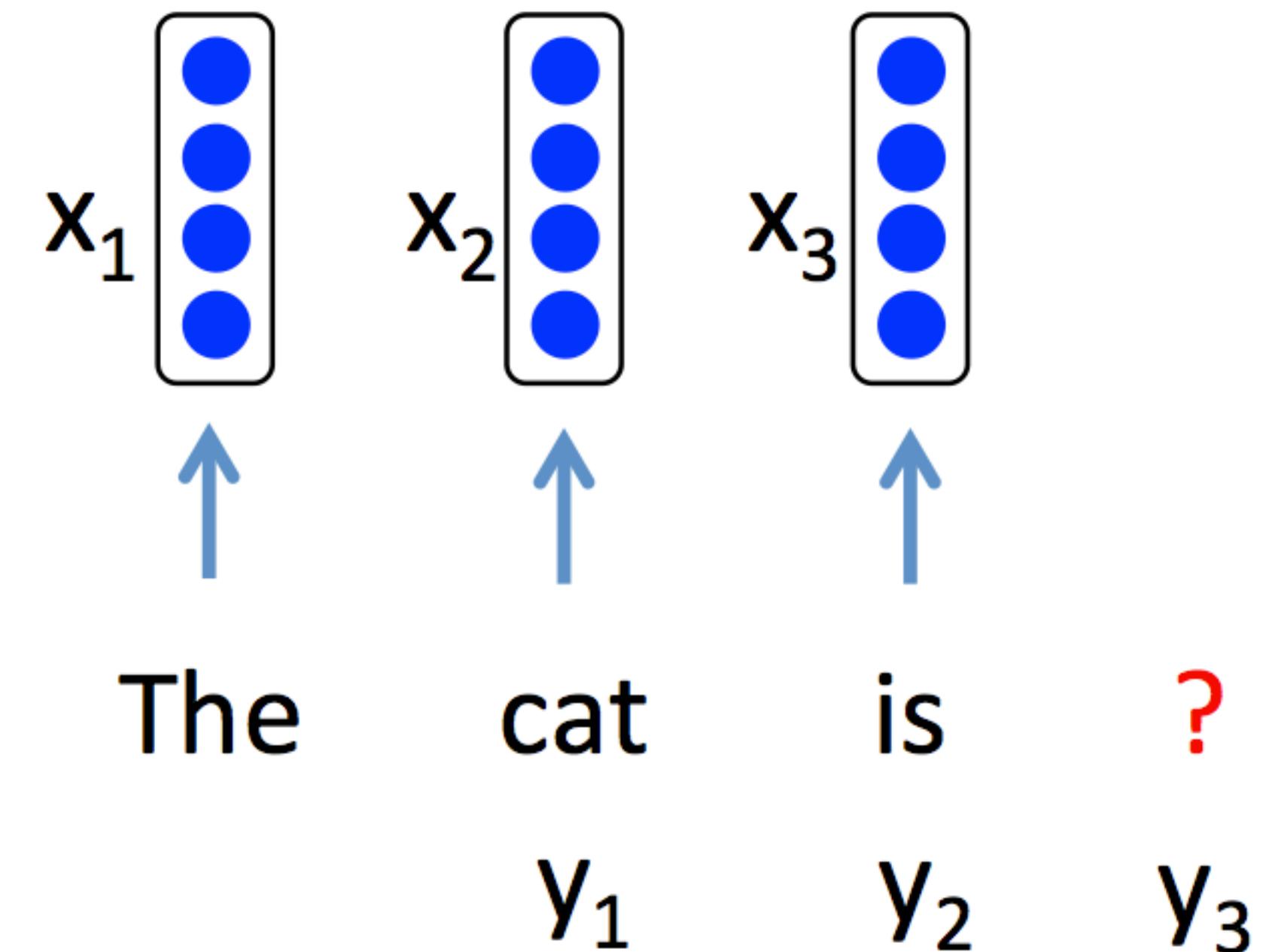
The cat is ?

# Recurrent Neural Network

## Example: Language Modeling

The cat is ?

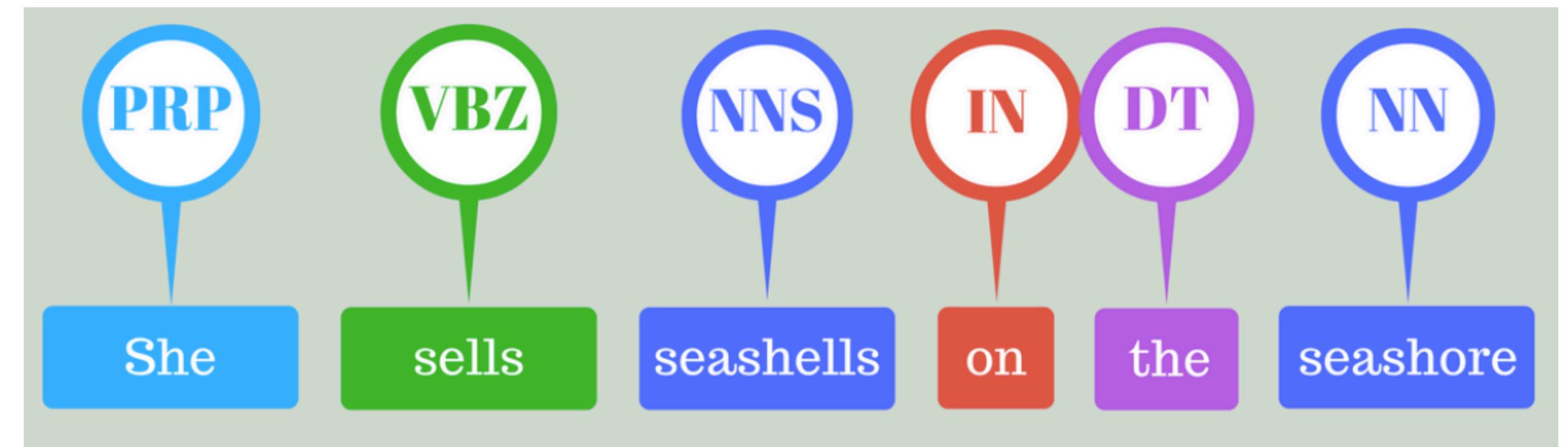
- $x_t$ : one-hot encoding to represent the word at step  $t$  ( $[0, \dots, 0, 1, 0, \dots, 0]$ )
- $y_t$ : the next word
  - $y_t \in \{1, \dots, V\}$   $V$ : Vocabulary size



# Recurrent Neural Network

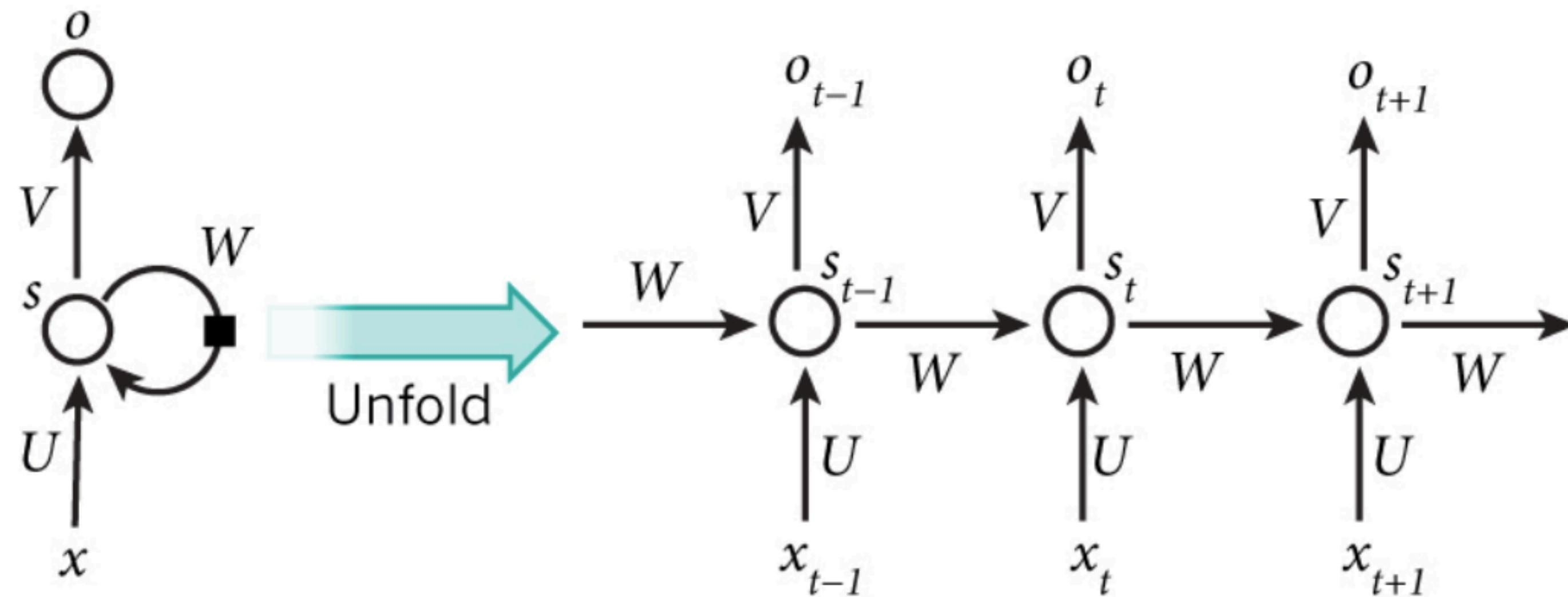
## Example: POS Tagging

- Part of Speech Tagging:
  - Labeling words with their Part-Of-Speech (Noun, Verb, Adjective, ...)
  - $x_t$ : a **vector** to represent the word at step  $t$
  - $y_t$ : label of word  $t$



# Recurrent Neural Network

## Example: POS Tagging



- $x_t$ :  $t$ -th input
- $s_t$ : hidden state at time  $t$  ("memory" of the network)
  - $s_t = f(Ux_t + Ws_{t-1})$
  - $W$ : transition matrix,  $U$ : word embedding matrix,  $s_0$  usually set to be 0
- Predicted output at time  $t$ :
  - $o_t = \arg \max_i (Vs_t)_i$

# Recurrent Neural Network

## Recurrent Neural Network (RNN)

- Training: Find  $U, W, V$  to minimize empirical loss:
- Loss of a sequence:

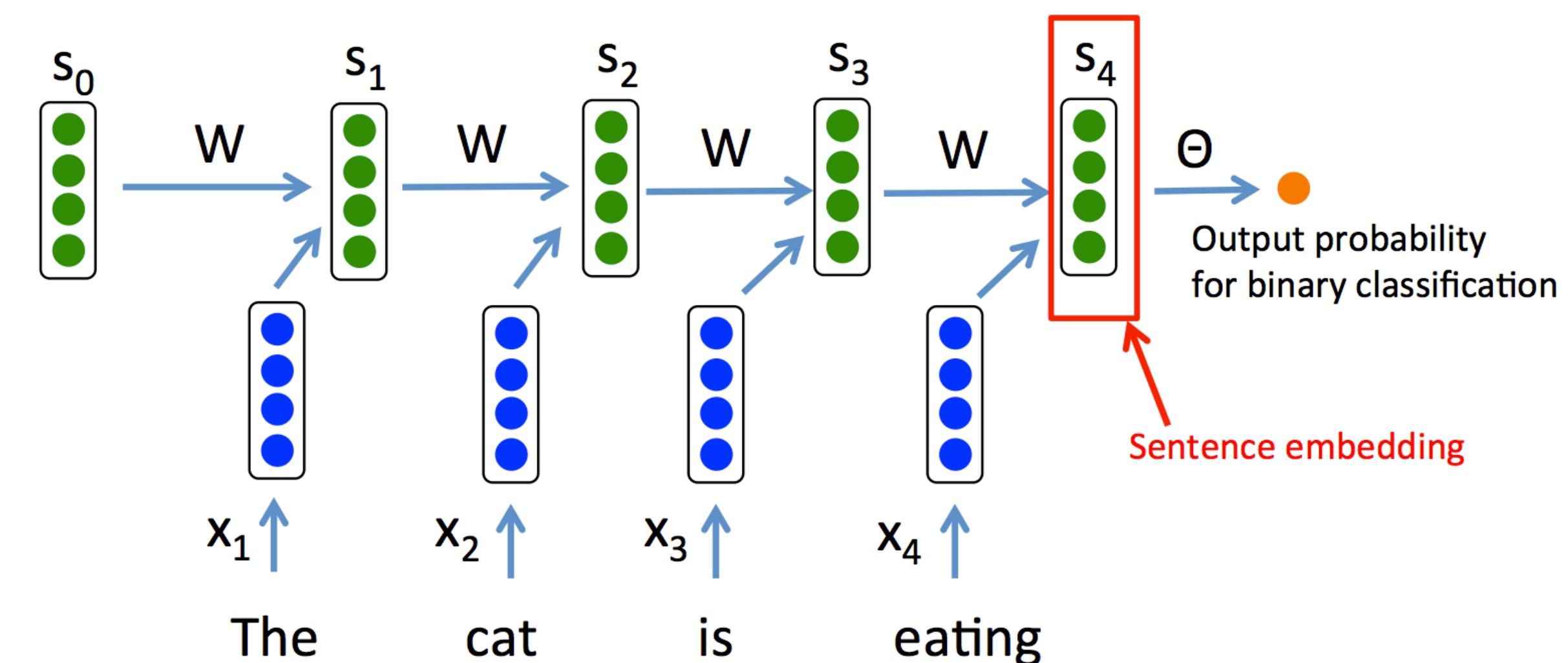
- $$\sum_{t=1}^T \text{loss}(Vs_t, y_t)$$

- ( $s_t$  is a function of  $U, W, V$ )
- Loss on the whole dataset:
  - Average loss over all sequences
  - Solved by SGD/Adam

# Recurrent Neural Network

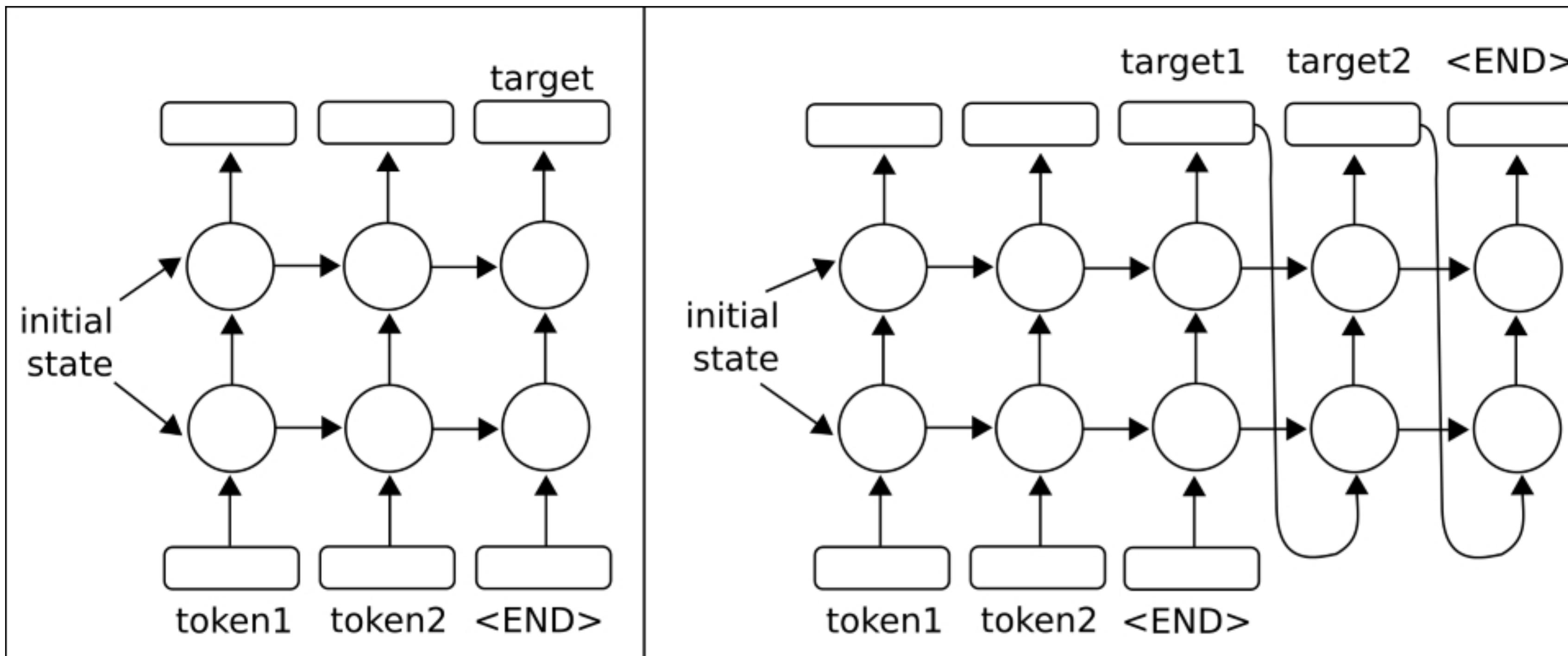
## RNN: Text Classification

- Not necessary to output at each step
- Text Classification:
  - sentence → category
  - Output only at the final step
- Model: add a fully connected network to the final embedding



# Recurrent Neural Network

## Multi-layer RNN



# Recurrent Neural Network

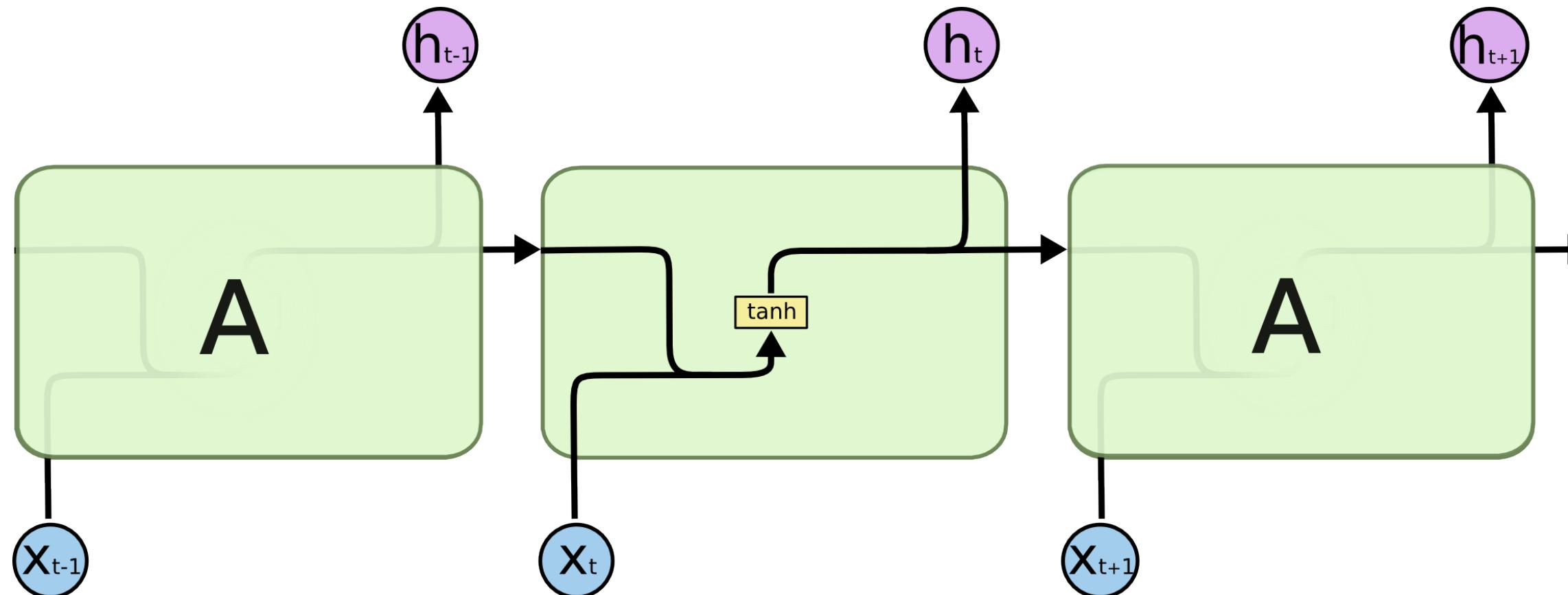
## Problems of Classical RNN

- Hard to capture long-term dependencies
- Hard to solve (vanishing gradient problem)
- Solution:
  - LSTM (Long Short Term Memory networks)
  - GRU (Gated Recurrent Unit)
  - ...

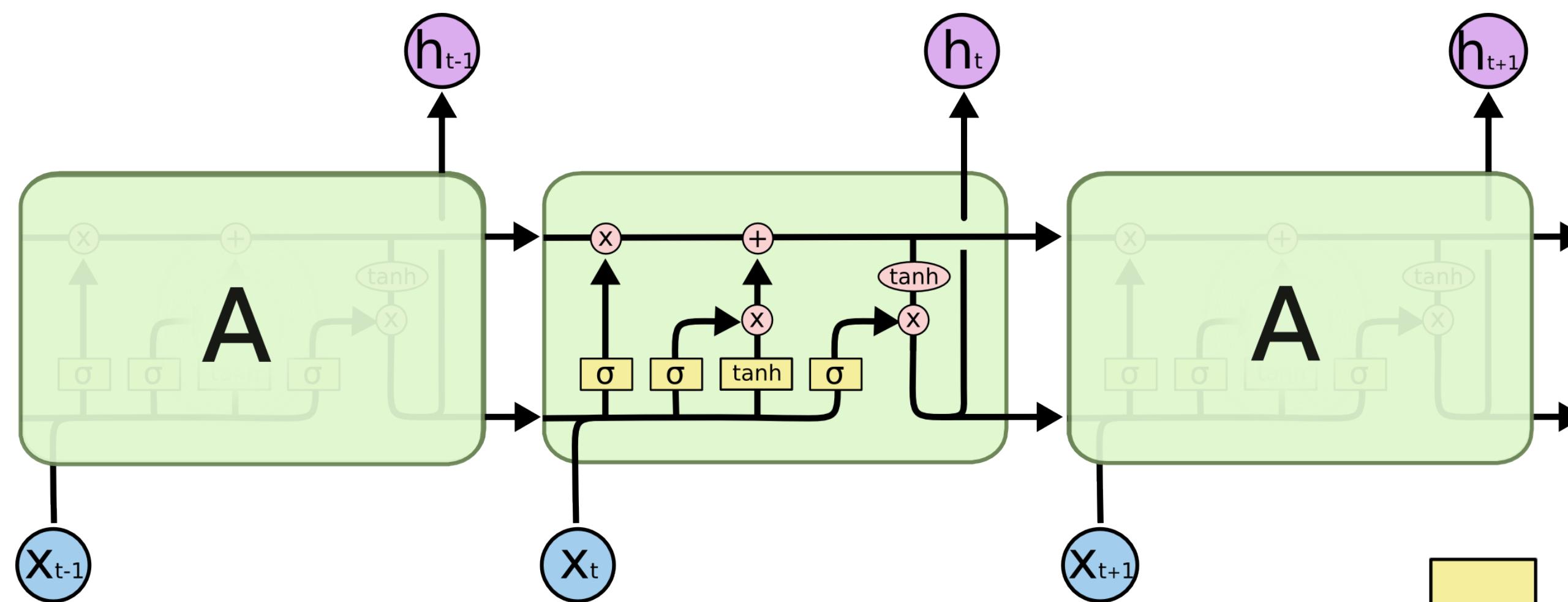
# Recurrent Neural Network

## LSTM

- RNN:



- LSTM:



Neural Network Layer

Pointwise Operation

Vector Transfer

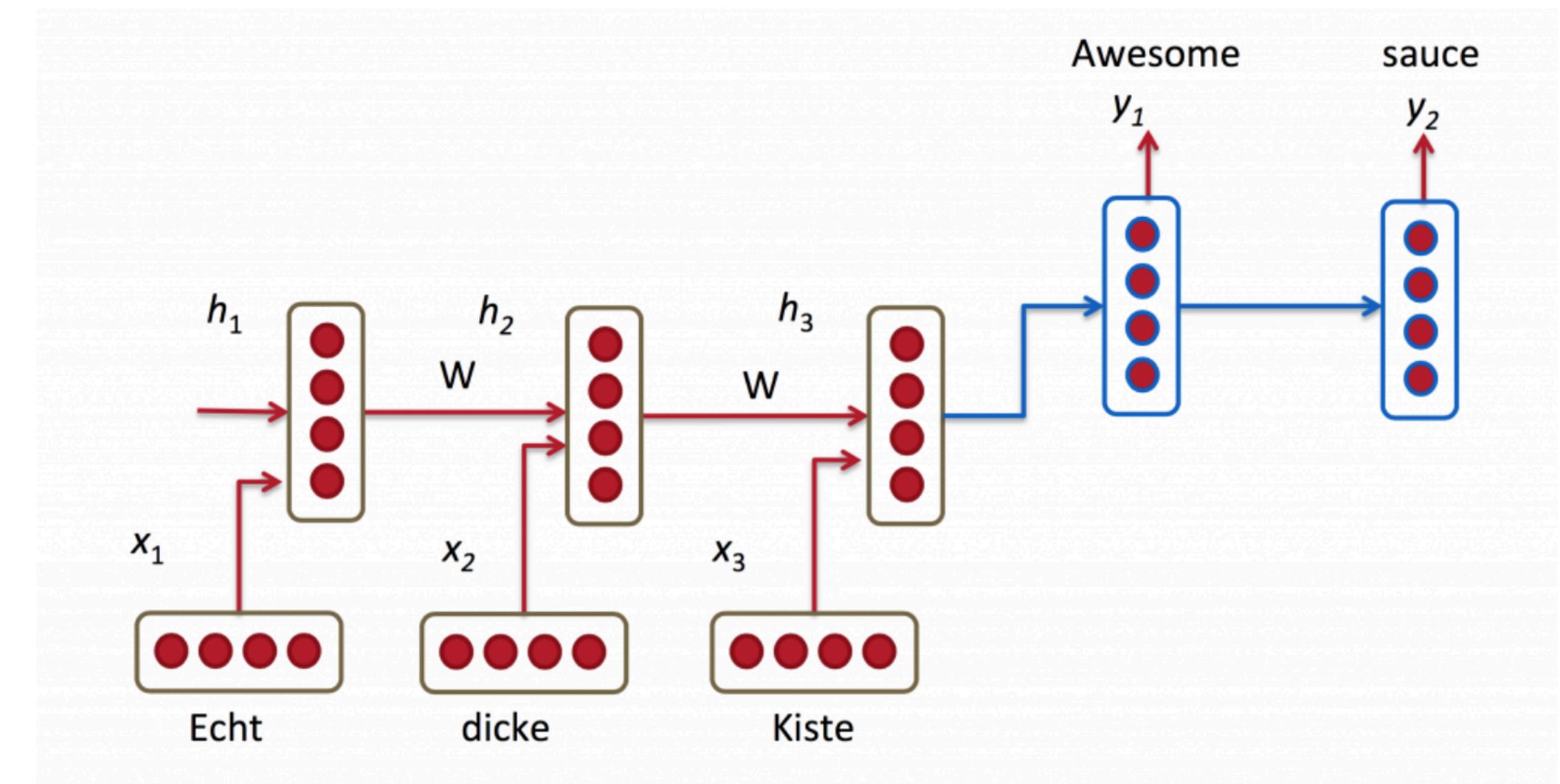
Concatenate

Copy

# Recurrent Neural Network

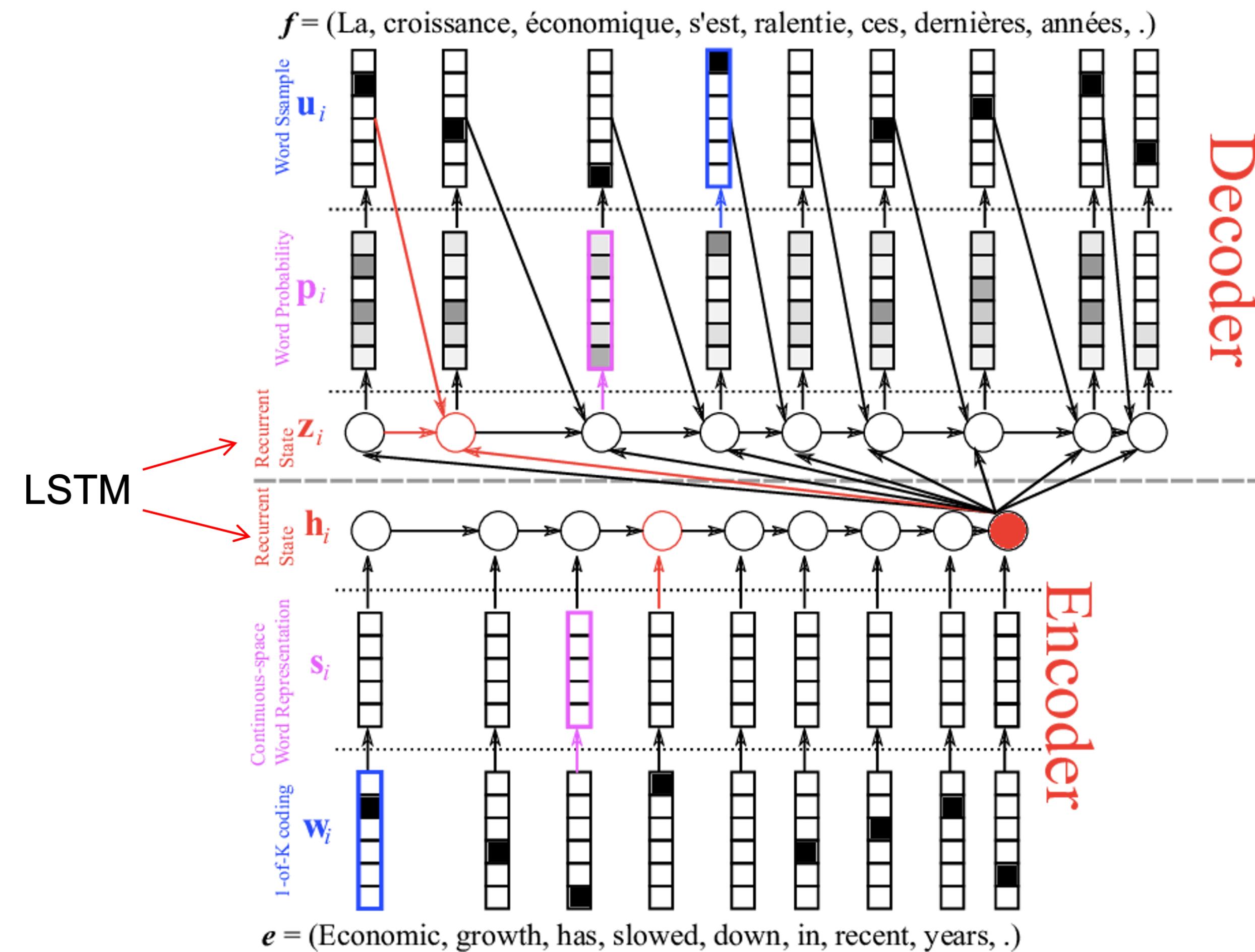
## Neural Machine Translation (NMT)

- Output the translated sentence from an input sentence
- Training data: a set of input-output pairs (supervised setting)
- Encoder-decoder approach:
  - Encoder: Use (RNN/LSTM) to encode the input sentence into a latent vector
  - Decoder: Use (RNN/LSTM) to generate a sentence based on the latent vector



# Recurrent Neural Network

## Neural Machine Translation



# Recurrent Neural Network

## Attention in NMT

- Usually, each output word is only related to a subset of input words (e.g., for machine translation)
- Let  $u$  be the **current decoder latent state**,  $v_1, \dots, v_n$  be the **latent state for each input word**
- Compute the weight of each state by
  - $p = \text{Softmax}(u^T v_1, \dots, u^T v_n)$
  - Compute the context vector by  $Vp = p_1 v_1 + \dots + p_n v_n$

# Recurrent Neural Network

## Attention in NMT

