Word Representation and Neural Network

Zhiyuan Liu

liuzy@tsinghua.edu.cn

THUNLP

Word Representation

THUNLP

Word Representation

- Word representation: a process that transform the symbols to the machine understandable meanings

- Definition of meaning (Webster Dictionary)
  - The thing one intends to convey especially by language
  - The logical extension of a word

- How to represent the meaning so that the machine can understand?

Goal of Word Representation

- Goal of Word Representation
  - Compute word similarity
    - WR(Star) ≃ WR(Sun)
    - WR(Motel) ≃ WR(Hotel)
  - Infer word relation
    - WR(China) − WR(Beijing) ≃ WR(Japan) - WR(Tokyo)
    - WR(Man) ≃ WR(King) − WR(Queen) + WR(Woman)
    - WR(Swimming) ≃ WR(Walking) − WR(Walk) + WR(Swim)

Synonym and Hypernym

- Use a set of related words, such as synonyms and hypernyms to represent a word
  - e.g. WordNet, a resource containing synonym and hypernym sets.

Synonyms of "Good" in WordNet:
(n)good,goodness
(n)commodity,trade_good,good
(s)full,good
(s)adept,expert,good,practiced,proficient,skillful
(s)estimable,good,honorable,respectable
(s)beneficial,good
...

Hypernyms of "NLP" in WordNet:

```
[Synset('information_science.n.01'),
 Synset('science.n.01'),
 Synset('discipline.n.01'),
 Synset('knowledge_domain.n.01'),
 Synset('content.n.05'),
 Synset('cognition.n.01'),
 Synset('psychological_feature.n.01'),
 Synset('abstraction.n.06'),
 Synset('entity.n.01')]
```

Problems of Synonym/Hypernym Representation

- Missing nuance
  - ("proficient", "good") are synonyms only in some contexts
- Missing new meanings of words
  - Apple (fruit → IT company)
  - Amazon (jungle → IT company)
- Subjective
- Data sparsity
- Requires human labor to create and adapt

One-Hot Representation

- Regard words as discrete symbols
- Word ID or one-hot representation
- E.g.

word	ID	one-hot vector
star	2	[0, 0, 1, 0, 0, 0, …]
sun	3	[0, 0, 0, 1, 0, 0, …]

  - Vector dimension = # words in vocabulary
  - Order is not important

Problems of One-Hot Representation

- $similarity(star,\ sun) = (v_{star},\ v_{sun}) = 0$

- All the vectors are orthogonal. No natural notion of similarity for one-hot vectors.

Represent Word by Context

- The meaning of a word 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.
- Use context words to represent stars
  - Co-occurrence Counts
  - Words Embeddings

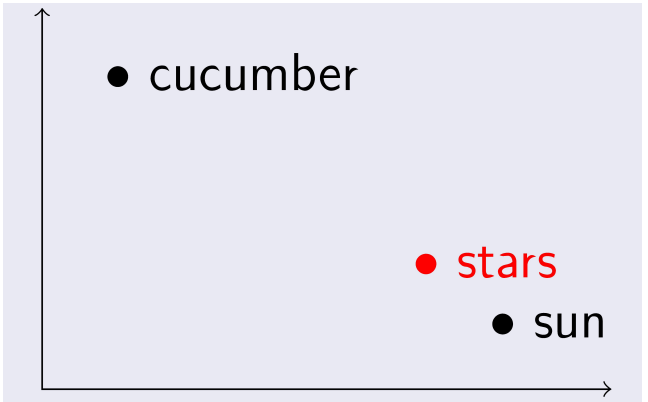
he curtains open and the stars shining in on the barely
ars and the cold , close stars " . And neither of the w
rough the night with the stars shining so brightly , it
made in the light of the stars . It all boils down , wr
surely under the bright stars , thrilled by ice-white
sun , the seasons of the stars ? Home , alone , Jay pla
m is dazzling snow , the stars have risen full and cold

Co-Occurrence Counts

- Count-based distributional representation

he curtains open and the stars shining in on the barely
ars and the cold , close stars " . And neither of the w
rough the night with the stars shining so brightly , it
made in the light of the stars . It all boils down , wr
surely under the bright stars , thrilled by ice-white
sun , the seasons of the stars ? Home , alone , Jay pla
m is dazzling snow , the stars have risen full and cold

- Term-Term matrix
  - How often a word occurs with another
- Term-Document matrix
  - How often a word occurs in a document



	shining	bright	trees	dark	look
stars	38	45	2	27	12

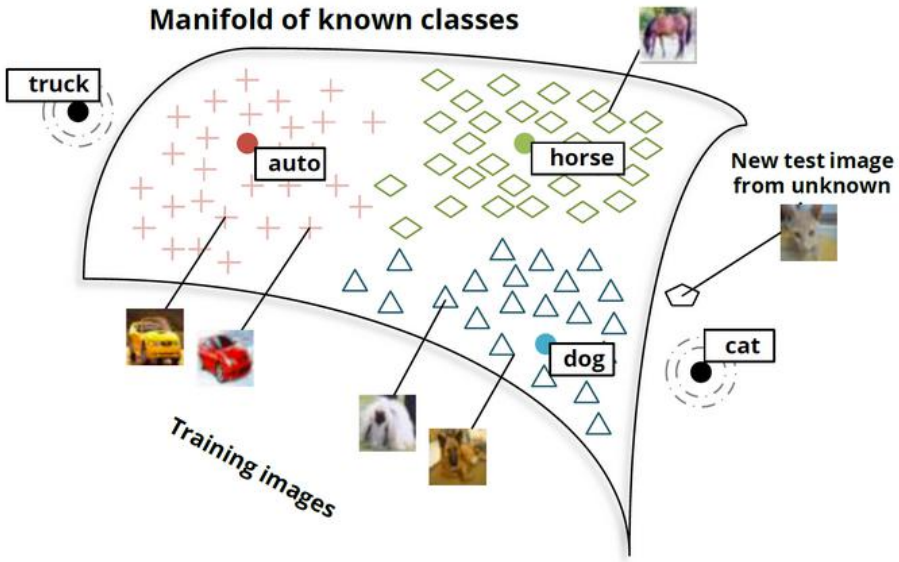
Problems of Count-Based Representation

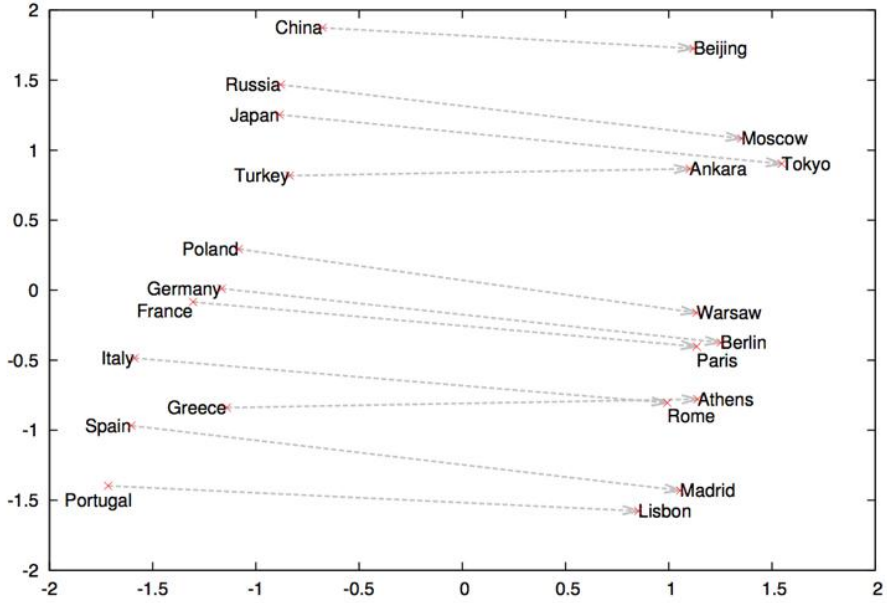
- Increase in size with vocabulary
- Require a lot of storage
- sparsity issues for those less frequent words

  → Subsequent classification models will be less robust

Word Embedding

- Distributed Representation
  - Build a dense vector for each word learned from large-scale text corpora
  - Learning method: word2vec





Distributed Word Representation

THUNLP

Language Model

- A language model is a probability distribution over a sequence of words
  - Compute joint probability of a sequence of words:

$$P(W) = P(w_1, w_2, \cdots, w_n)$$

  - Compute conditional probability of an upcoming word w_n :

$$P(w_n | w_1, w_2, \cdots, w_{n-1})$$

- How to compute the sentence probability?

Language Model

- Assumption: the probability of an upcoming word is only determined by all its previous words

1. $P(Never, too, late, to, learn) =$

$$P(Never) \times P(too|Never) \times P(late|Never, too) \times$$

$$P(to|Never, too, late) \times P(learn|Never, too, late, to)$$

2. $P(learn|Never, too, late, to) = \frac{P(Never,too,late,to,learn)}{P(Never,too,late,to)}$

- Language Model

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

N-gram Model

- Limitation: When the sentence is long, the computation process can be very slow
  - For the last word: $P(learn|It, is, never, too, late, to)$

- N-gram
  - Restrict the number of previous words (2 or 3)
  - Back-off or smoothing method can deal with new word combinations

N-gram Model

Andrei Markov

- Markov assumption

$$P(w_1, w_2, \cdots, w_n) \approx \prod_i P(w_i | w_{i-k}, \cdots, w_{i-1})$$

  - And we have:

$$P(w_i | w_1, w_2, \cdots, w_{i-1}) \approx P(w_i | w_{i-k}, \cdots, w_{i-1})$$

- Simplifying Language Model
  - Bigram (N-Gram, N = 2)

$$P(learn | Never, too, late, to) \approx P(learn | to)$$

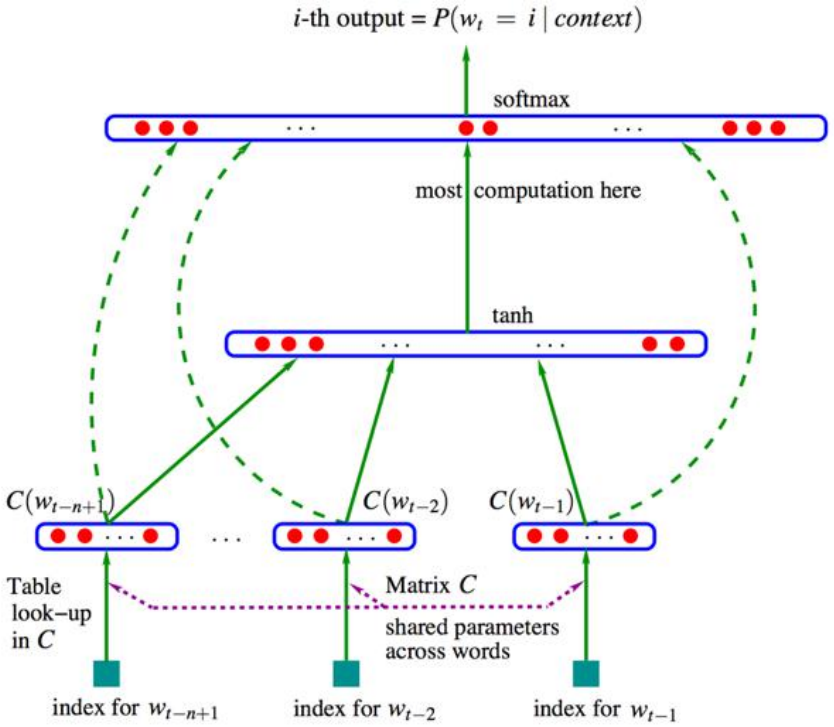
  - Trigram (N-Gram, N = 3)

$$P(learn | Never, too, late, to) \approx P(learn | late, to)$$

Problems of N-Grams

- Not considering contexts farther than 1 or 2 words
- Not capturing the similarity between words
  - The cat is walking in the bedroom
  - A dog was running in a room
    - N-gram cannot find the similar semantics and grammatical role between cat and dog, walking and running

Neural Language Model



Journal of Machine Learning Research 3 (2003) 1137–1155 Submitted 4/02; Published 2/03

A Neural Probabilistic Language Model

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Réjean Ducharme DUCHARME@IRO.UMONTREAL.CA
Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Christian Jauvin JAUVINC@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal, Montréal, Québec, Canada

Editors: Jaz Kandola, Thomas Hofmann, Tomaso Poggio and John Shawe-Taylor

Abstract

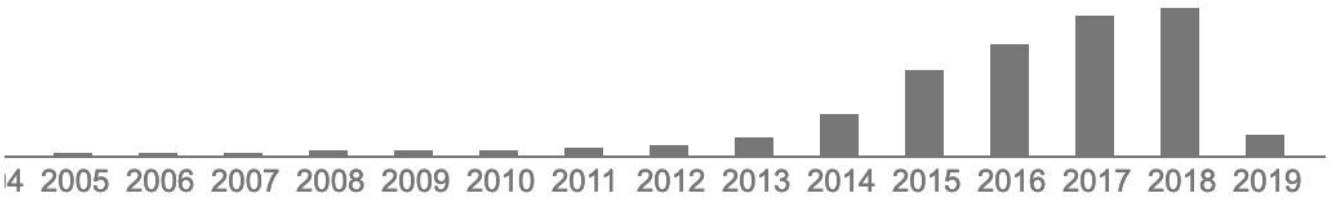
A goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the **curse of dimensionality**: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. Traditional but very successful approaches based on n-grams obtain generalization by concatenating very short overlapping sequences seen in the training set. We propose to fight the curse of dimensionality by **learning a distributed representation for words** which allows each training sentence to inform the model about an exponential number of semantically neighboring sentences. The model learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar (in the sense of having a nearby representation) to words forming an already seen sentence. Training such large models (with millions of parameters) within a reasonable time is itself a significant challenge. We report on experiments using neural networks for the probability function, showing on two text corpora that the proposed approach significantly improves on state-of-the-art n-gram models, and that the proposed approach allows to take advantage of longer contexts.

Keywords: Statistical language modeling, artificial neural networks, distributed representation, curse of dimensionality

1. Introduction

A fundamental problem that makes language modeling and other learning problems difficult is the *curse of dimensionality*. It is particularly obvious in the case when one wants to model the joint distribution between many discrete random variables (such as words in a sentence, or discrete attributes in a data-mining task). For example, if one wants to model the joint distribution of 10 consecutive words in a natural language with a vocabulary V of size 100,000, there are potentially $100000^{10} - 1 = 10^{50} - 1$ free parameters. When modeling continuous variables, we obtain generalization more easily (e.g. with smooth classes of functions like multi-layer neural networks or Gaussian mixture models) because the function to be learned can be expected to have some local smoothness properties. For discrete spaces, the generalization structure is not as obvious: any change of these discrete variables may have a drastic impact on the value of the function to be esti-

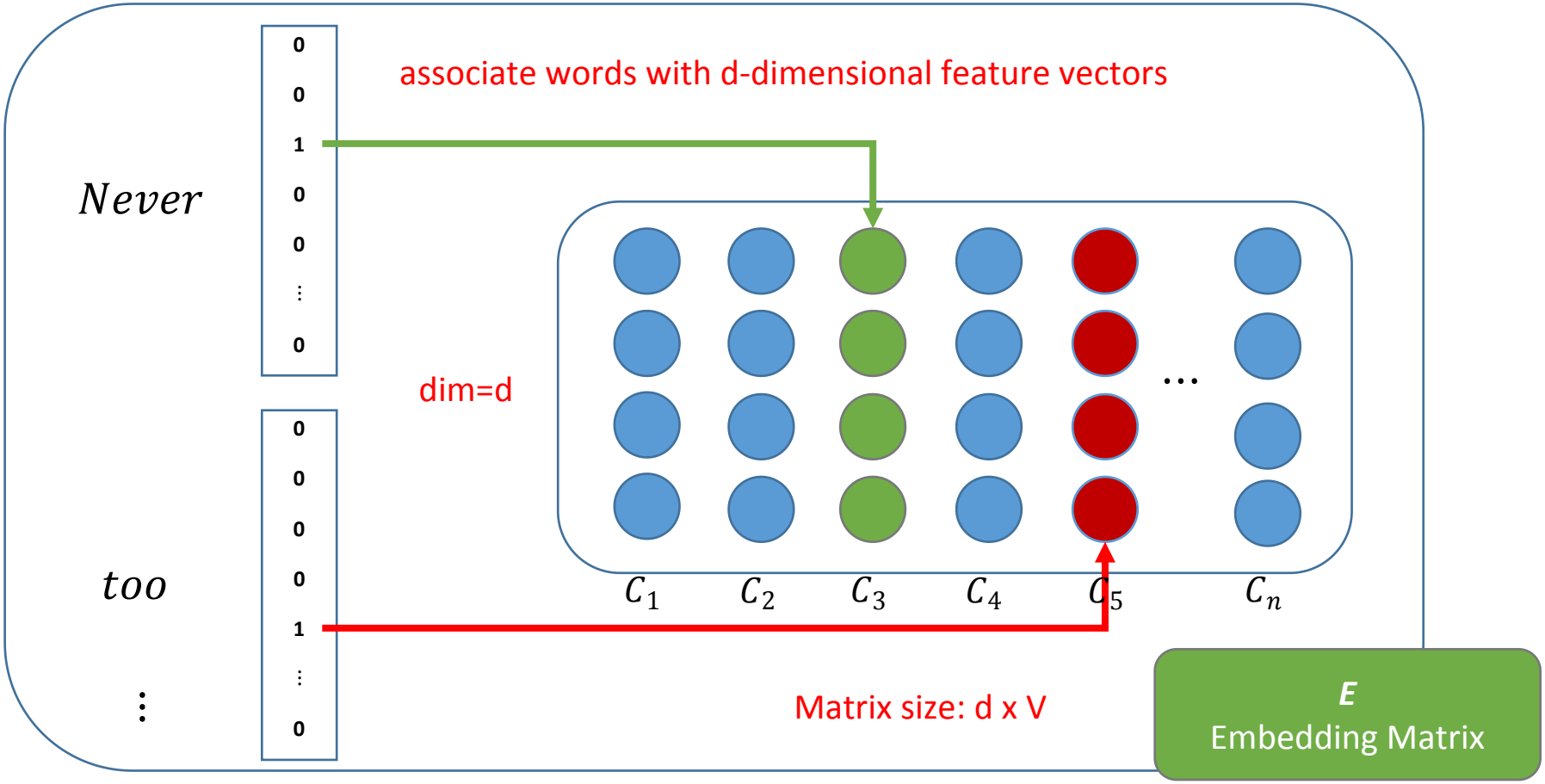
Cited by 4373



Yoshua Bengio, et al. A neural probabilistic language model. JMLR 2003.

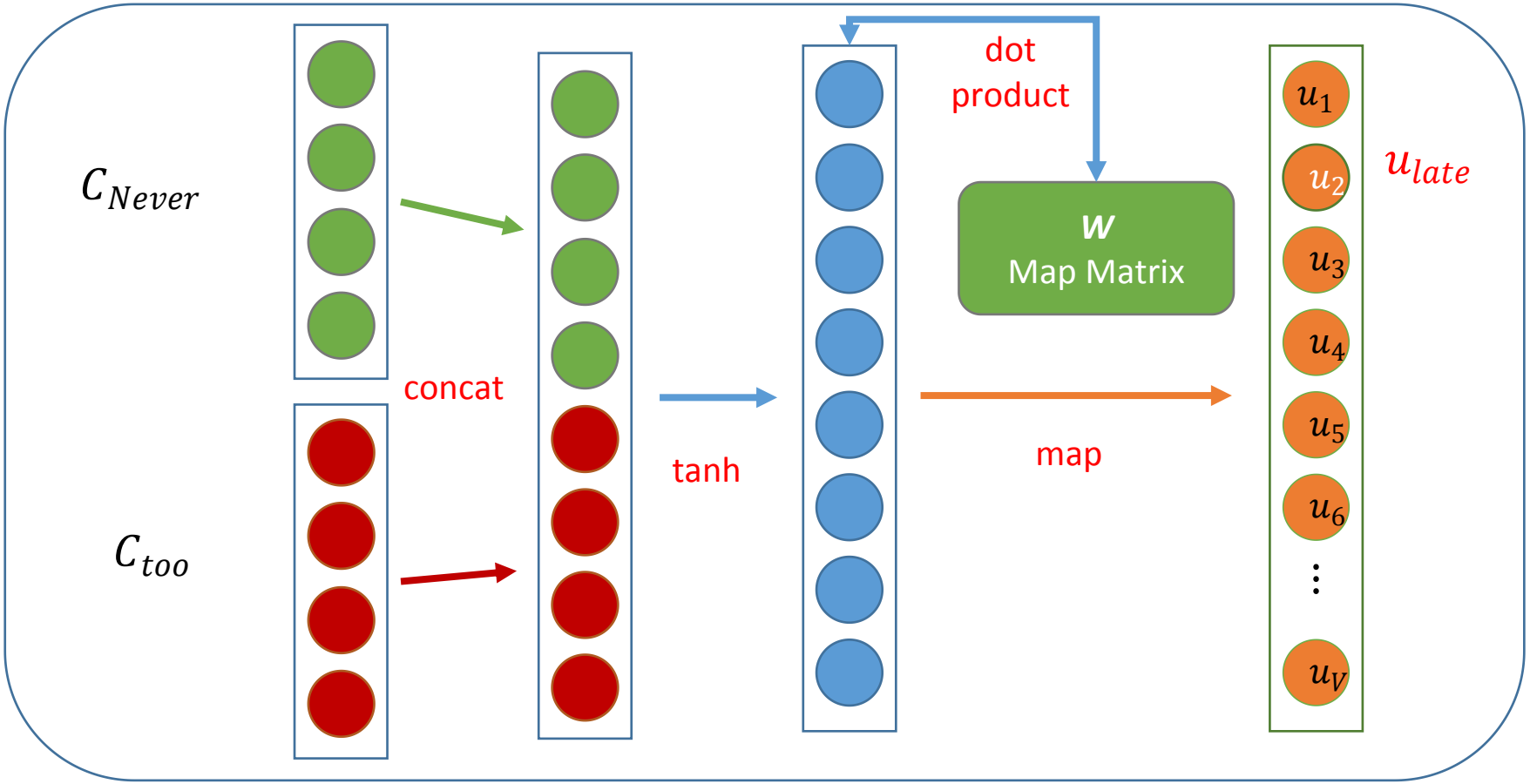
Lookup Table

- *Never too late to learn*

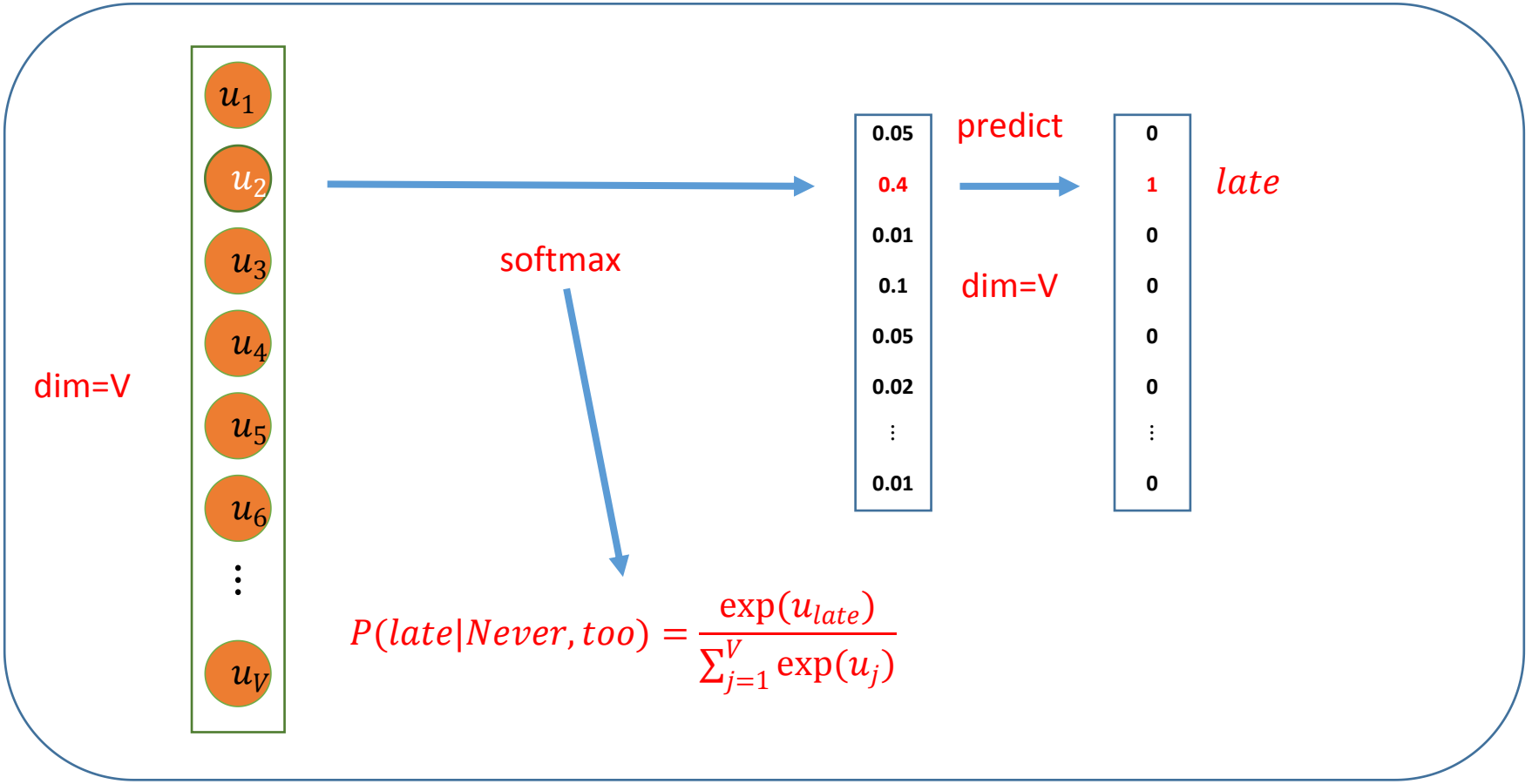


Probability of the Next Word

$$u_i = \sum_j W_{ij} I_j + b_j$$



Softmax and Prediction



Neural Language Model

- A neural language model is a language model based on neural networks to learn distributed representations of words
  - Associate words with distributed vectors
  - Compute the joint probability of word sequences in terms of the feature vectors
  - Optimize the word feature vectors (embedding matrix E) and the parameters of the loss function (map matrix W)

Challenges of Language Models

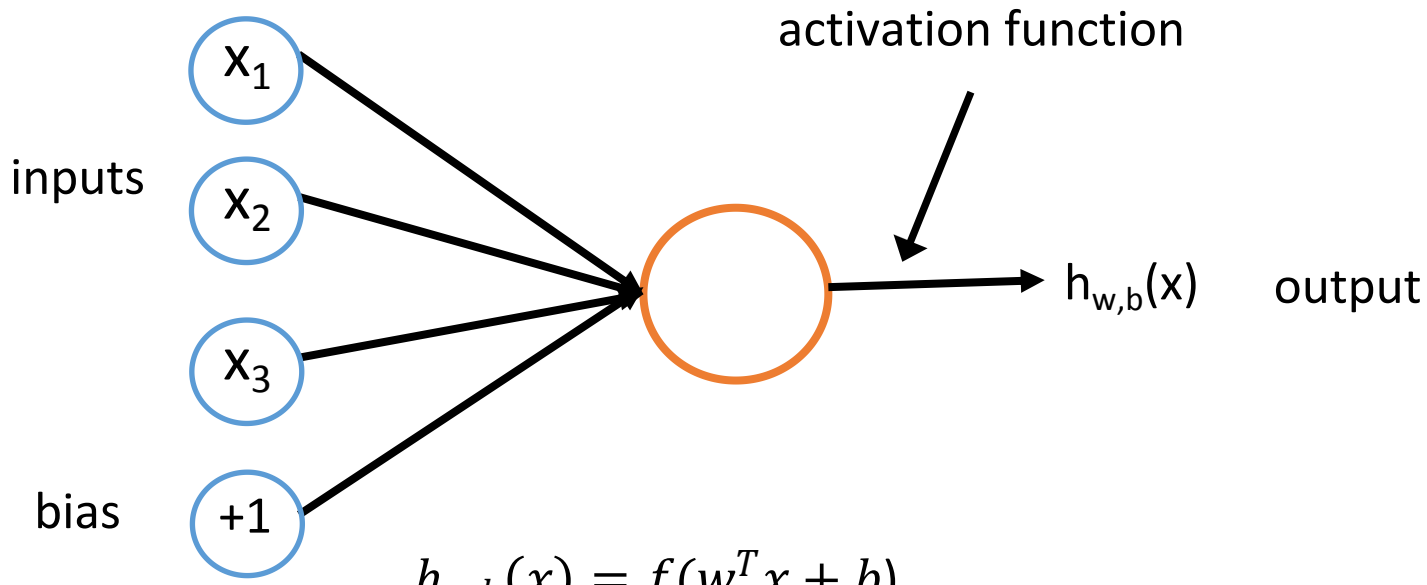
- The vocabulary size can be very large, maybe hundreds of thousands of different words
  - Too many parameters
    - Lookup Table (Embedding Matrix)
    - Map matrix
  - Too many computation
    - Non-linear operation (Tanh)
    - Softmax for each words every step: compute vocabulary size times of conditional probabilities
- Word2vec is one of our choices

Neural Network

THUNLP

A Simple Neuron

- A neuron is a computational unit with n inputs and 1 output and parameters $w,\ b$



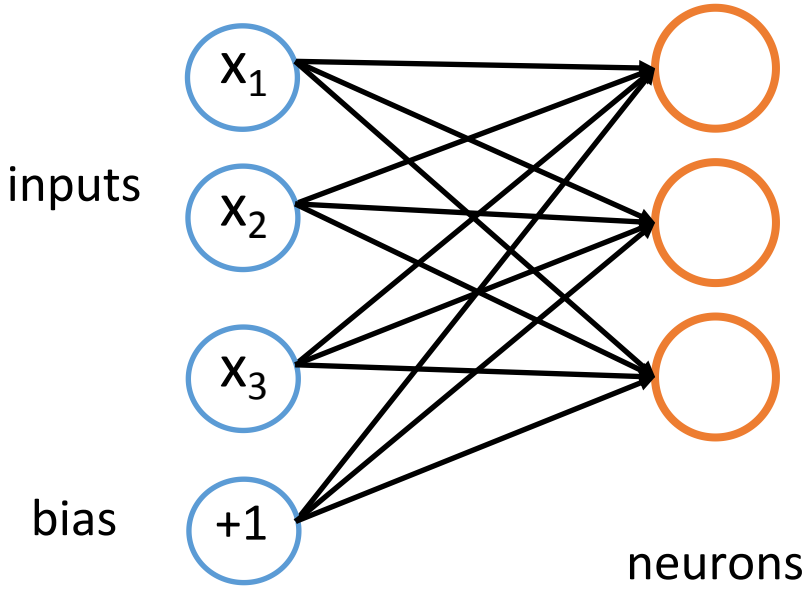
$$h_{w,b}(x) = f(w^T x + b)$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

w, b are the parameters of this neuron

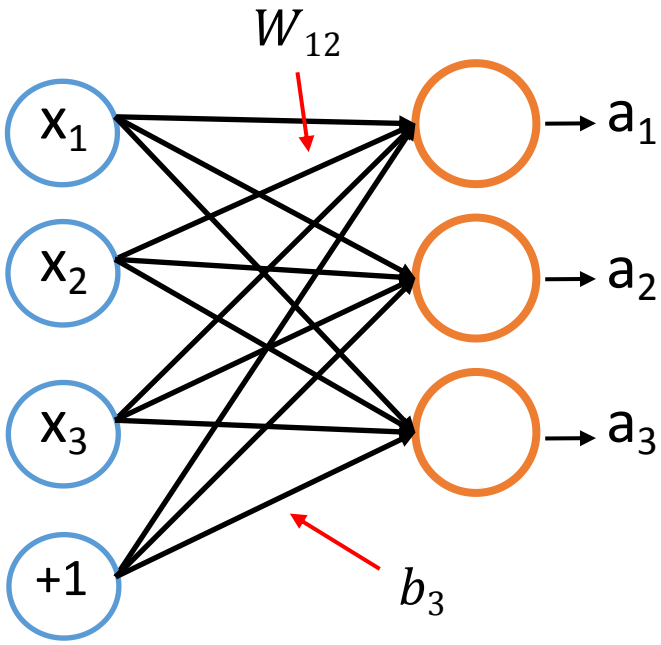
Neural Network

- A single layer neural network: Hooking together many simple neurons



Matrix Notation

- A single layer neural network: Hooking together many simple neurons



$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

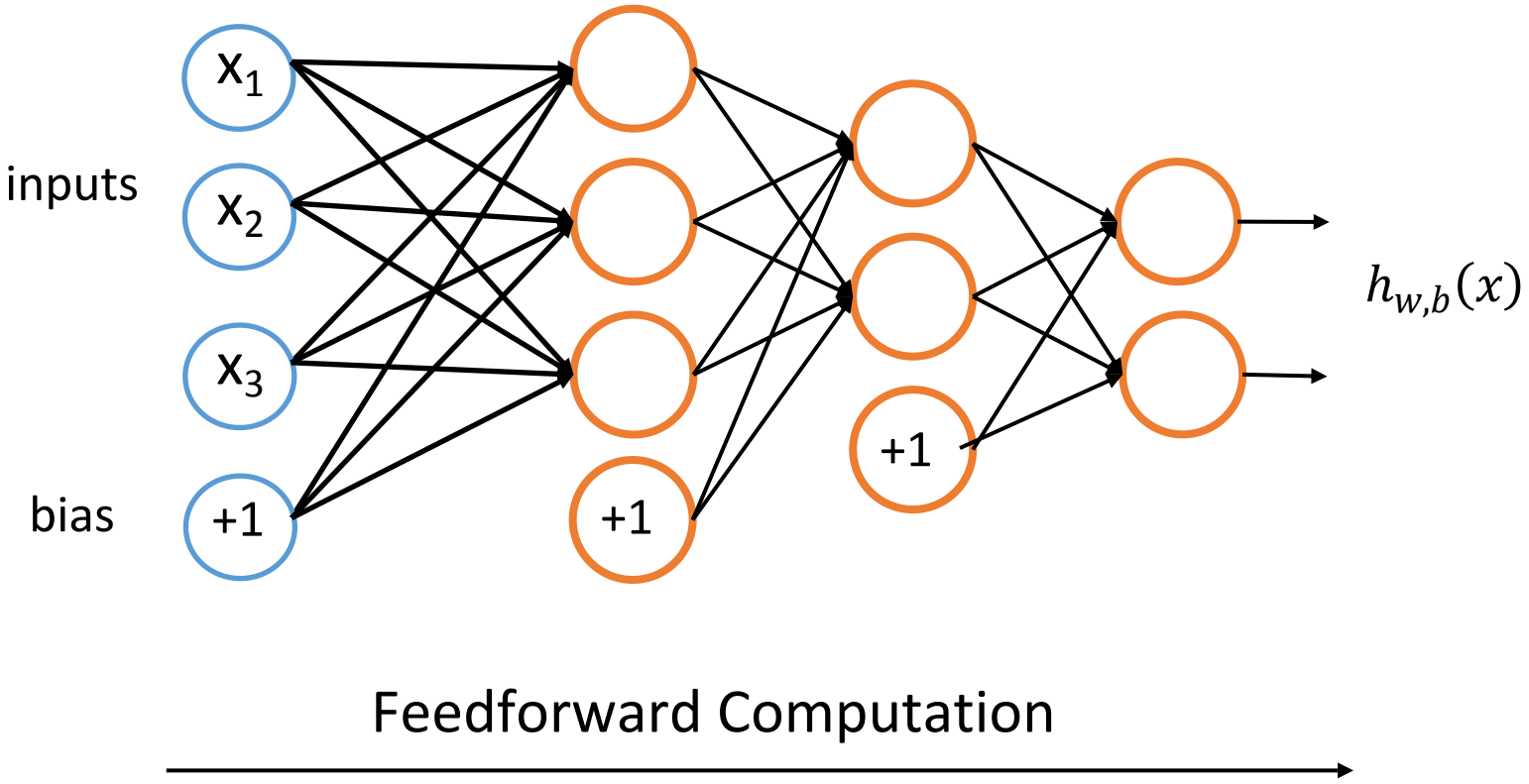
$$a_3 = f(W_{31}x_1 + W_{32}x_2 + W_{33}x_3 + b_3)$$

In matrix form:

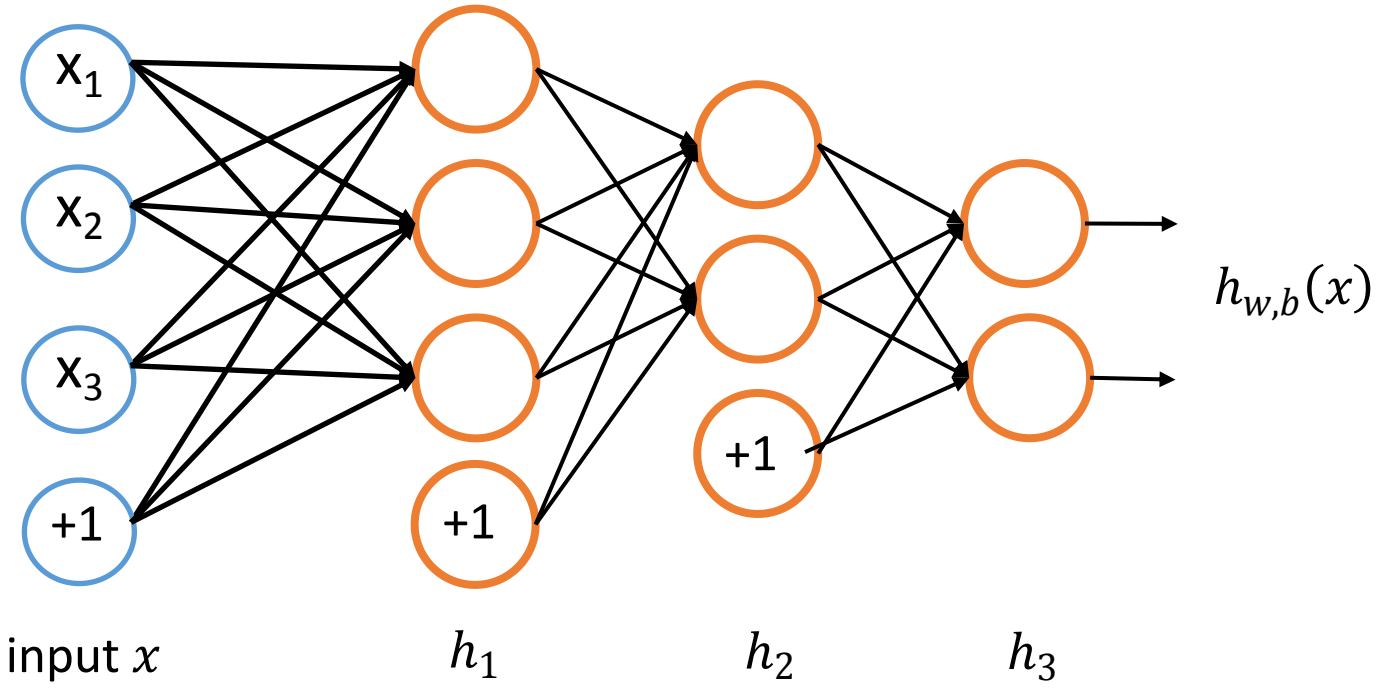
$$a = f(Wx + b)$$

Multilayer Neural Network

- Stacking multiple layers of neural networks



Feedforward Computation



$$\mathrm{h}_1 = f(W_1 x + b_1)$$

$$\mathrm{h}_2 = f(W_2 h_1 + b_2)$$

$$\mathrm{h}_3 = f(W_3 h_2 + b_3)$$

Why use non-linearities (f)?

- Without non-linearities, deep neural networks cannot do anything more than a linear transform

- Extra layers could just be compiled down into a single linear transform:

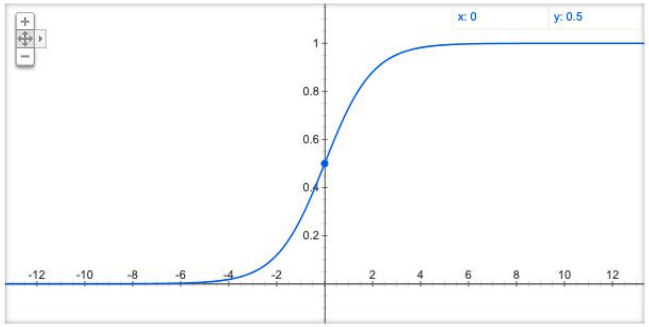
$$W_2(W_1x + b_1) + b_2 = Wx + b$$

- With non-linearities, neural networks can approximate more complex functions with more layers!

Choices of non-linearities

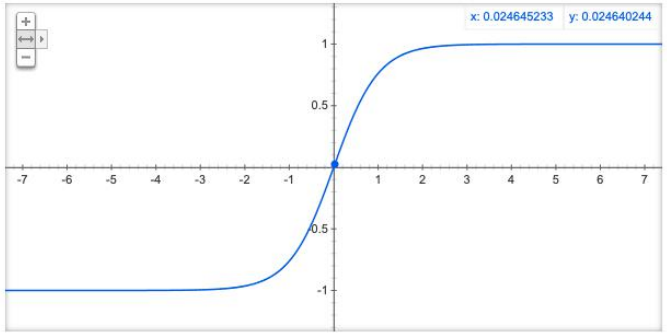
- Sigmoid

$$f(z) = \frac{1}{1 + e^{-z}}$$



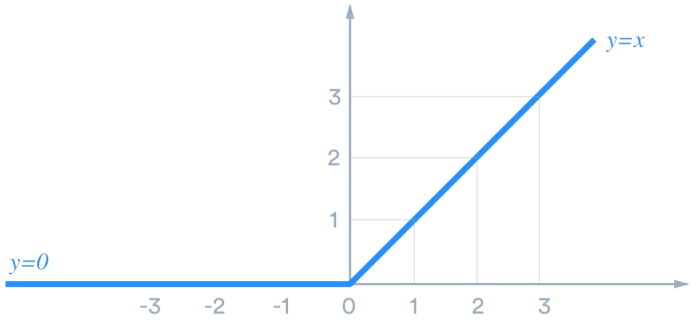
- Tanh

$$f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



- ReLU

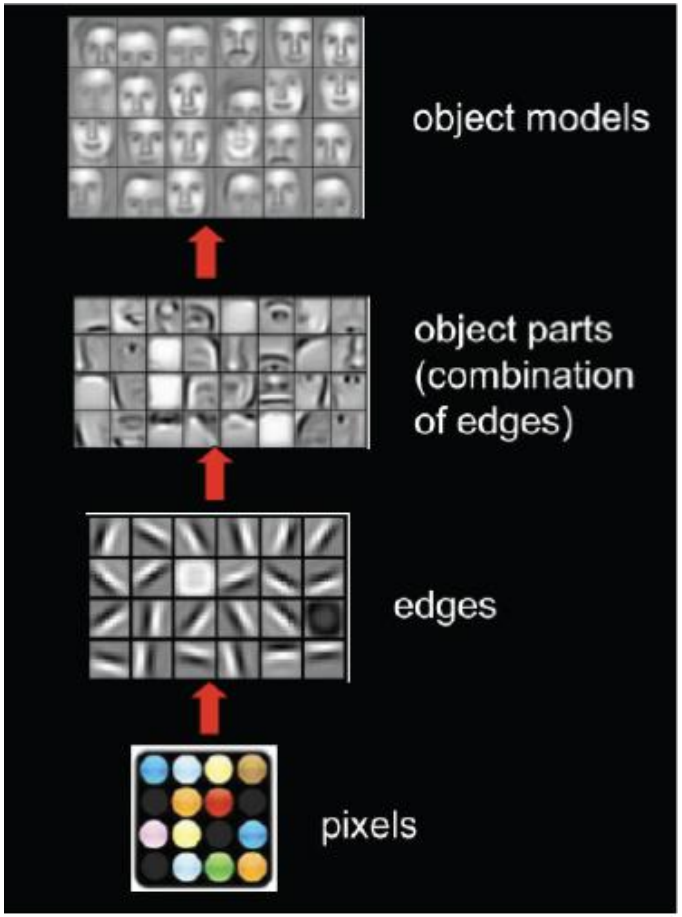
$$f(z) = \max(z, 0)$$



- …

Why have multiple layers?

- Neural network can represent more complicated features by hierarchical representations.



Summary

- Simple neuron

$$h_{w,b}(x) = f(w^T x + b)$$

- Single layer neural network

$$h_{w,b}(x) = f(Wx + b)$$

- Multilayer neural network
  - Stack multiple layers of neural networks
- Non-linearity activation function
  - Enable neural nets to represent more complicated features

How to Train a Neural Network

THUNLP

Training Objective

- First set up a training objective for the model:
  - Given N training examples $\{(x_i, y_i)\}_{i=1}^{N}$ where x_i and y_i are the attributes and price of a computer. We want to train a neural network $F_\theta(\cdot)$ which takes the attributes x as input and predicts its price y. A reasonable training objective is

$$\min_\theta J(\theta) = \min_\theta \frac{1}{N} \sum_{i=1}^{N} (y_i - F_\theta(x_i))^2 ,$$

where θ is the parameters in neural network $F_\theta(\cdot)$.

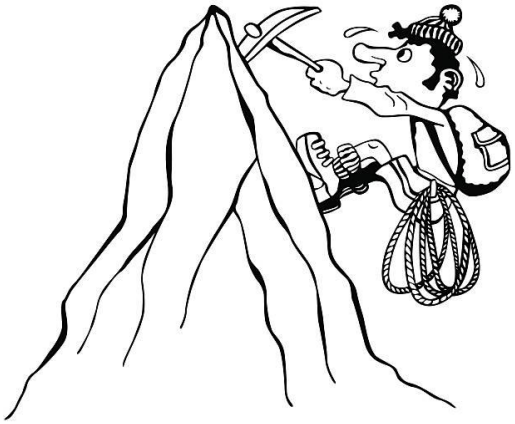
Stochastic Gradient Descent

- Update rule:

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} \mathrm{J}(\theta)$$

α is step size or learning rate

- Just like climbing a mountain
  - find the steepest direction
  - take a step



Gradients

- Given a function with 1 output and n inputs:

$$\mathrm{F}(x) = F(x_1, x_2 \ldots x_n)$$

- Its gradient is a vector of partial derivatives:

$$\frac{\partial \mathrm{F}}{\partial x} = \left[\frac{\partial \mathrm{F}}{\partial x_1}, \frac{\partial \mathrm{F}}{\partial x_2} \cdots \frac{\partial \mathrm{F}}{\partial x_n}\right]$$

Jacobian Matrix: Generalization of the Gradient

- Given a function with m outputs and n inputs:

$$\mathrm{F}(x) = [F_1(x_1, x_2 \ldots x_n), F_2(x_1, x_2 \ldots x_n) \ldots F_m(x_1, x_2 \ldots x_n)]$$

- Its Jacobian matrix is an $m \times n$ matrix of partial derivatives:

$$\frac{\partial \mathrm{F}}{\partial x} = \begin{bmatrix} \frac{\partial \mathrm{F}_1}{\partial x_1} & \cdots & \frac{\partial \mathrm{F}_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathrm{F}_\mathrm{m}}{\partial x_1} & \cdots & \frac{\partial \mathrm{F}_\mathrm{m}}{\partial x_n} \end{bmatrix}$$

$$\left(\frac{\partial \mathrm{F}}{\partial x}\right)_{ij} = \frac{\partial \mathrm{F}_\mathrm{i}}{\partial x_j}$$

Chain Rule for Jacobians

- For one-variable functions: multiply derivatives

$$z = 3y$$

$$y = x^2$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = 3 \times 2x = 6x$$

- For multiple variables: multiply Jacobians

$$h = f(z)$$

$$z = Wx + b$$

$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial z}\frac{\partial z}{\partial x} = \ldots$$

Back to Neural Network

- Given $s = u^T h, h = f(z), z = Wx + b$, what is $\frac{\partial s}{\partial b}$?

Back to Neural Network

- Given $s = u^T h, h = f(z), z = Wx + b,$ what is $\frac{\partial s}{\partial b}$?
  - Apply the chain rule:

$$\frac{\partial s}{\partial b} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial b}$$

u^T $\quad \mathrm{diag}(\mathrm{f}'(z)) \quad$ I

Backpropagation

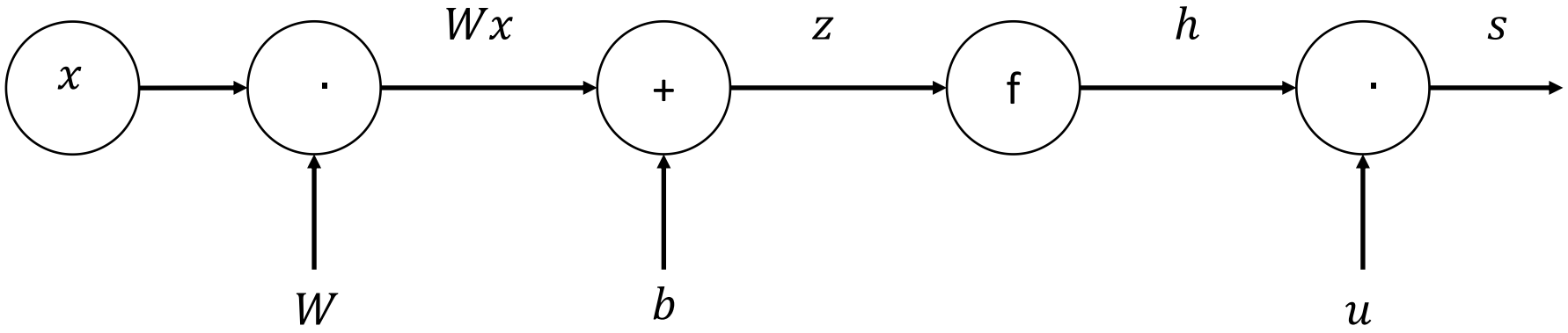
- Compute gradients algorithmically

- Used by deep learning frameworks (TensorFlow, PyTorch, etc.)

Computational Graphs

- Representing our neural net equations as a graph
  - Source node: inputs
  - Interior nodes: operations
  - Edges pass along result of the operation

$$s = u^T h$$
$$h = f(z)$$
$$z = Wx + b$$
x input

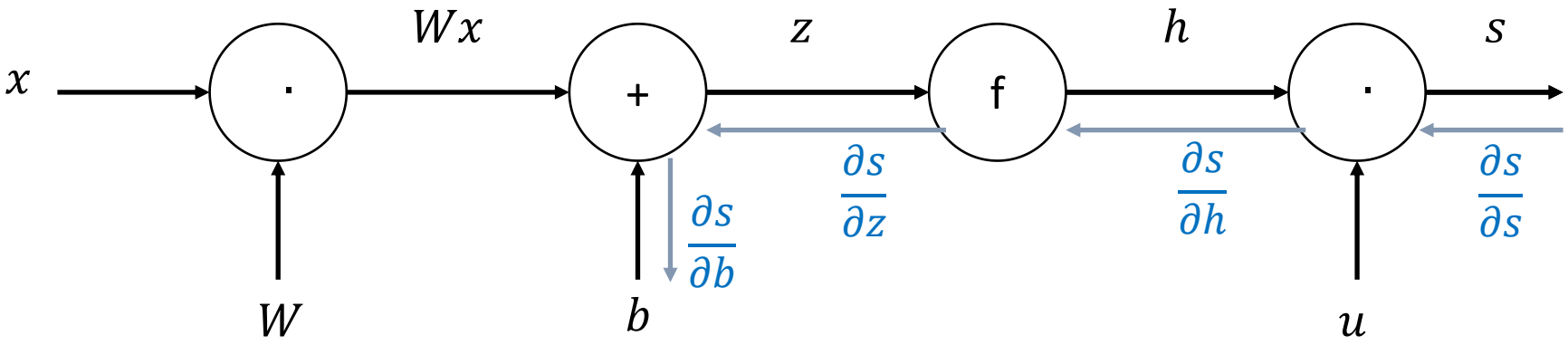


"Forward Propagation"

Backpropagation

- Go backwards along edges
  - Pass along gradients

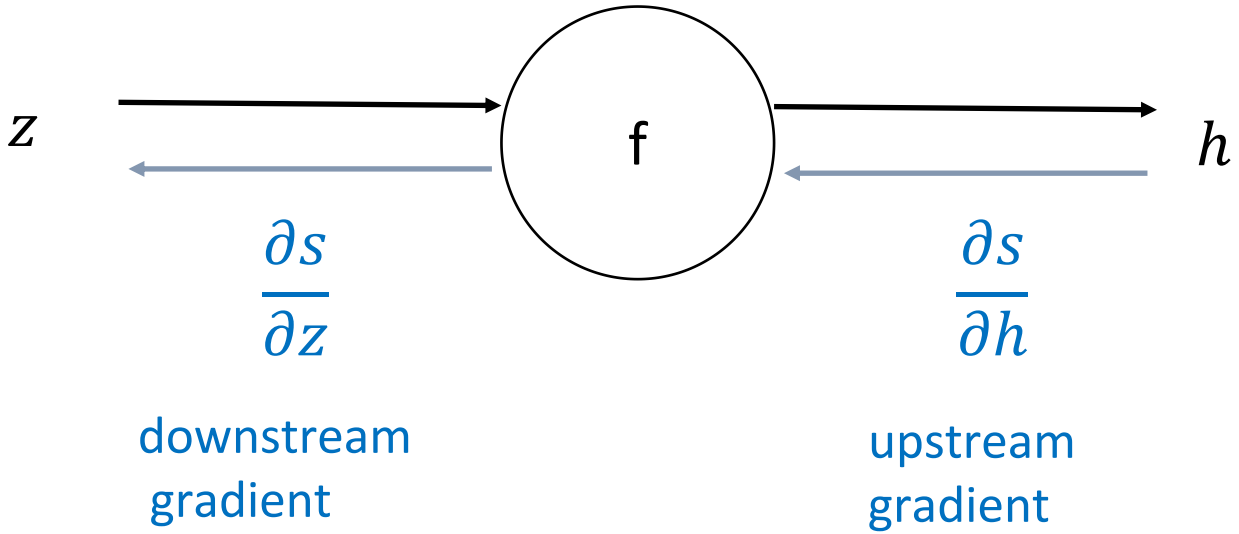
$$s = u^T h$$
$$h = f(z)$$
$$z = Wx + b$$
x: input



Backpropagation: Single Node

- Node receives an "upstream gradient"
- Goal is to pass on the correct "downstream gradient"

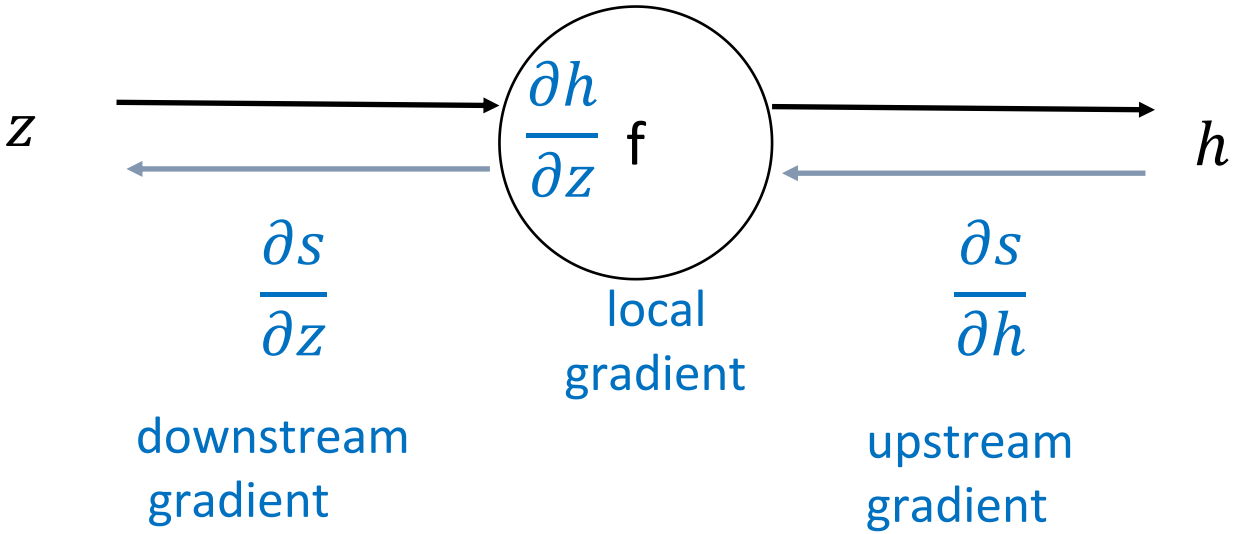
$$h = f(z)$$



Backpropagation: Single Node

- Each node has a local gradient
  - The gradient of its output with respect to its input

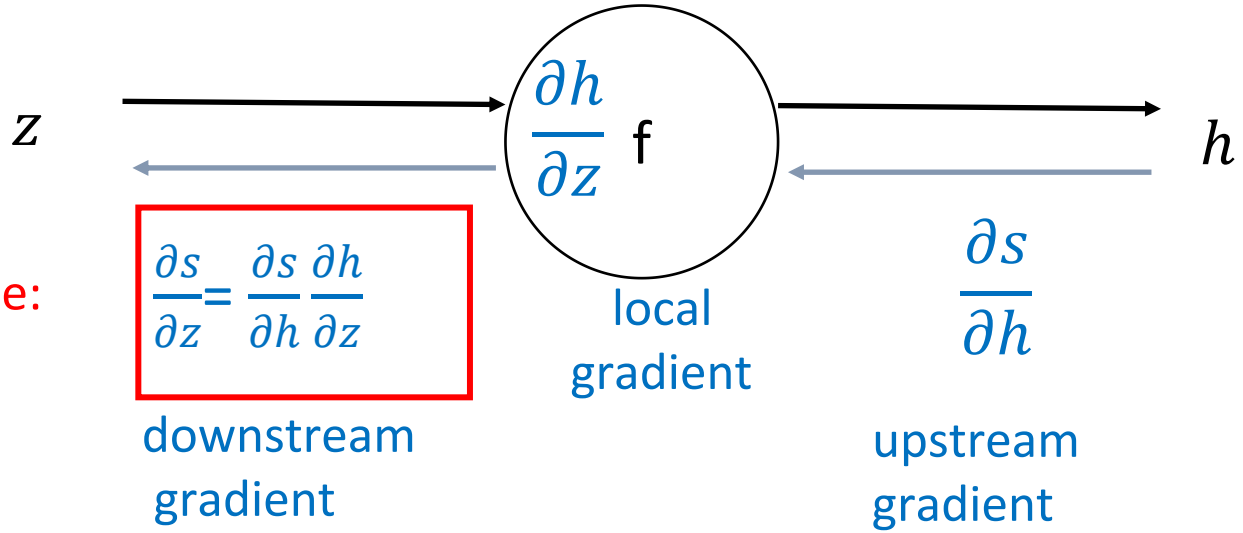
$$h = f(z)$$



Backpropagation: Single Node

- Each node has a local gradient
  - The gradient of its output with respect to its input
- [downstream gradient] = [upstream gradient] x [local gradient]

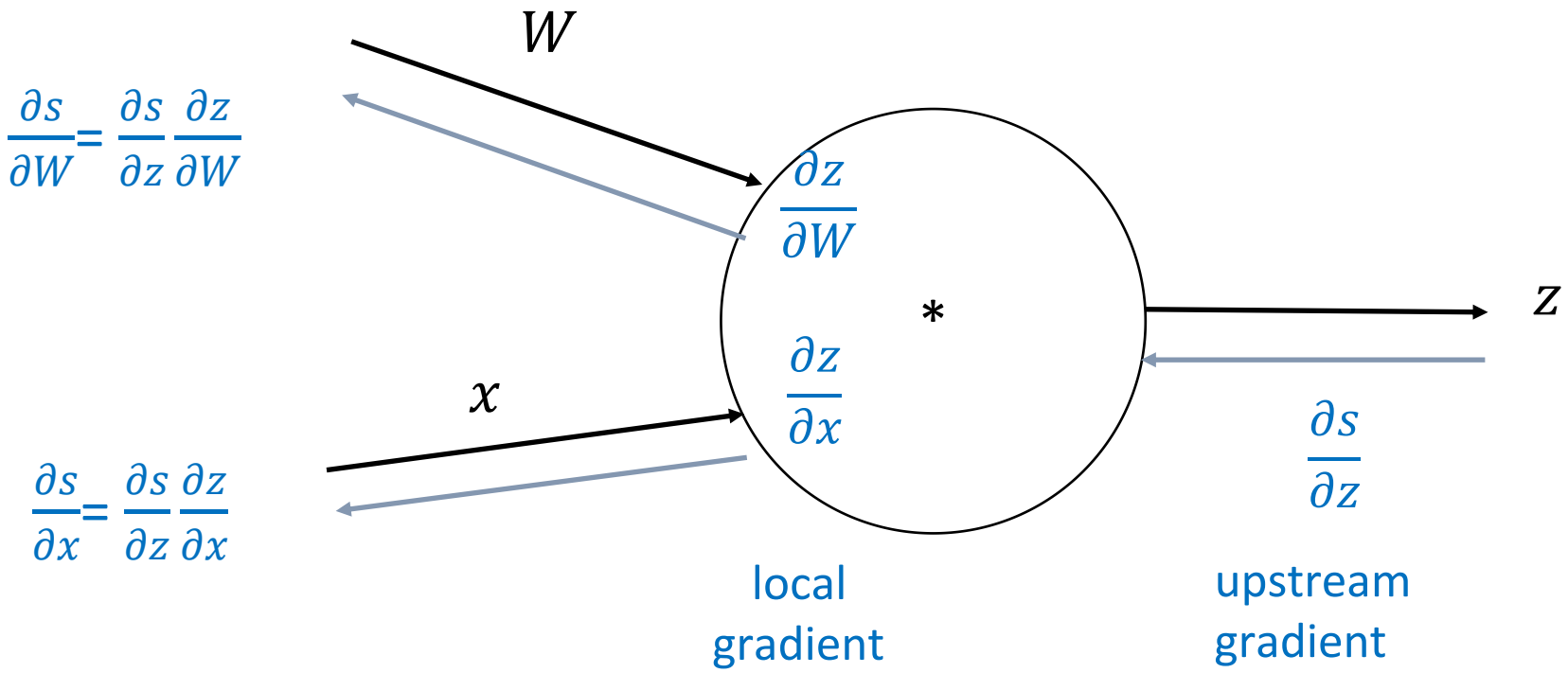
$$h = f(z)$$



Backpropagation: Single Node

- What about nodes with multiple inputs?

$$z = Wx$$



An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps:

$$a = x + y$$

$$\mathrm{b} = \max(y, z)$$

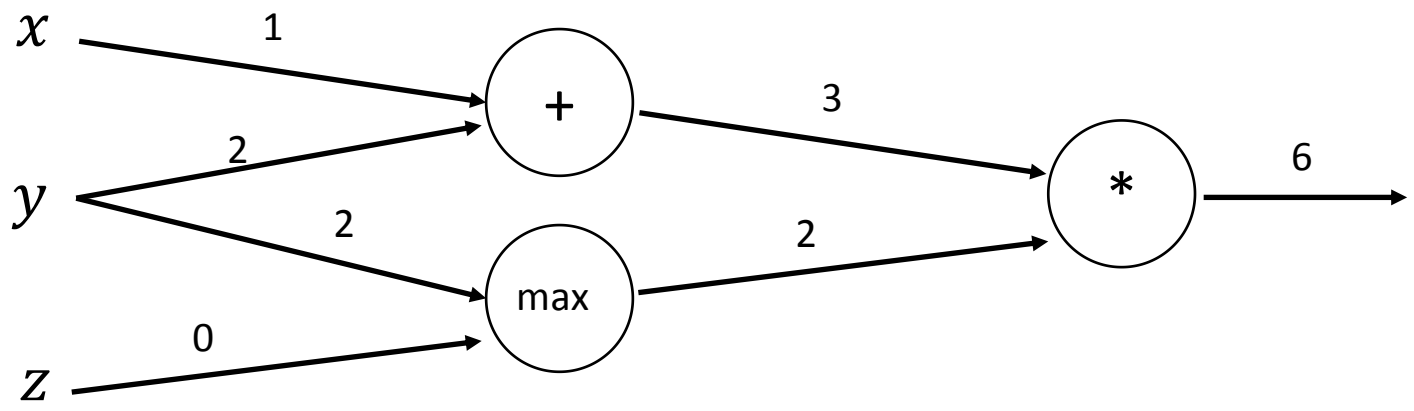
$$f = ab$$

Local gradients:

$$\frac{\partial a}{\partial x} = 1, \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1, \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b, \frac{\partial f}{\partial b} = a$$



An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps:

$$a = x + y$$

$$b = \max(y, z)$$

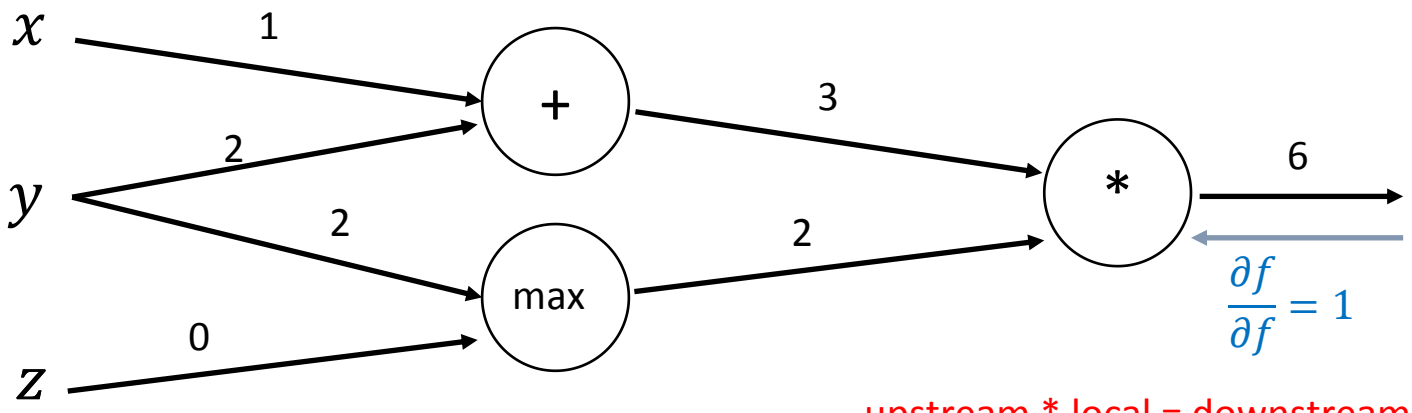
$$f = ab$$

Local gradients:

$$\frac{\partial a}{\partial x} = 1, \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1, \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b, \frac{\partial f}{\partial b} = a$$



upstream * local = downstream

An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps:

$$a = x + y$$

$$b = \max(y, z)$$

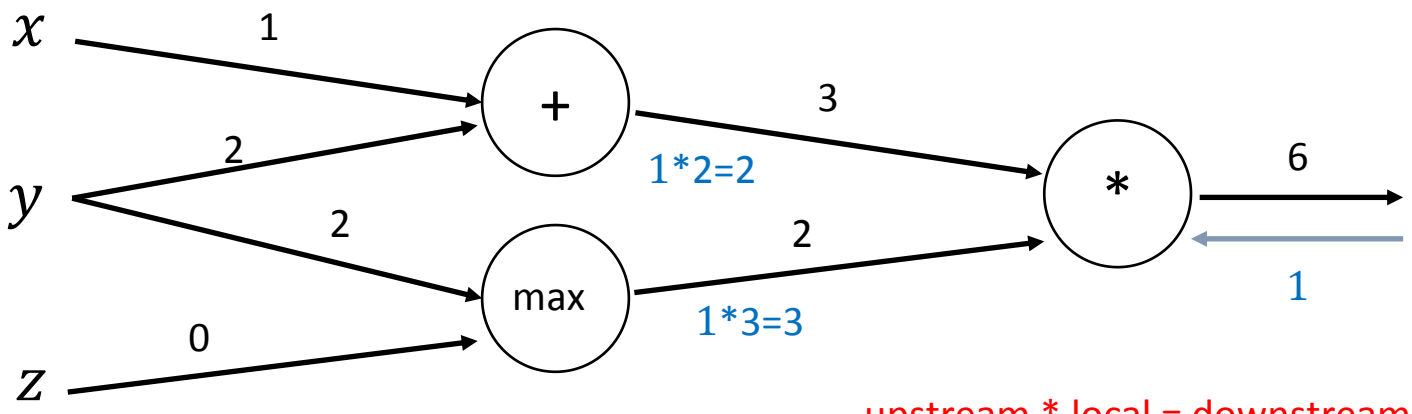
$$f = ab$$

Local gradients:

$$\frac{\partial a}{\partial x} = 1, \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1, \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b, \frac{\partial f}{\partial b} = a$$



upstream * local = downstream

An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps:

$$a = x + y$$

$$\mathrm{b} = \max(y, z)$$

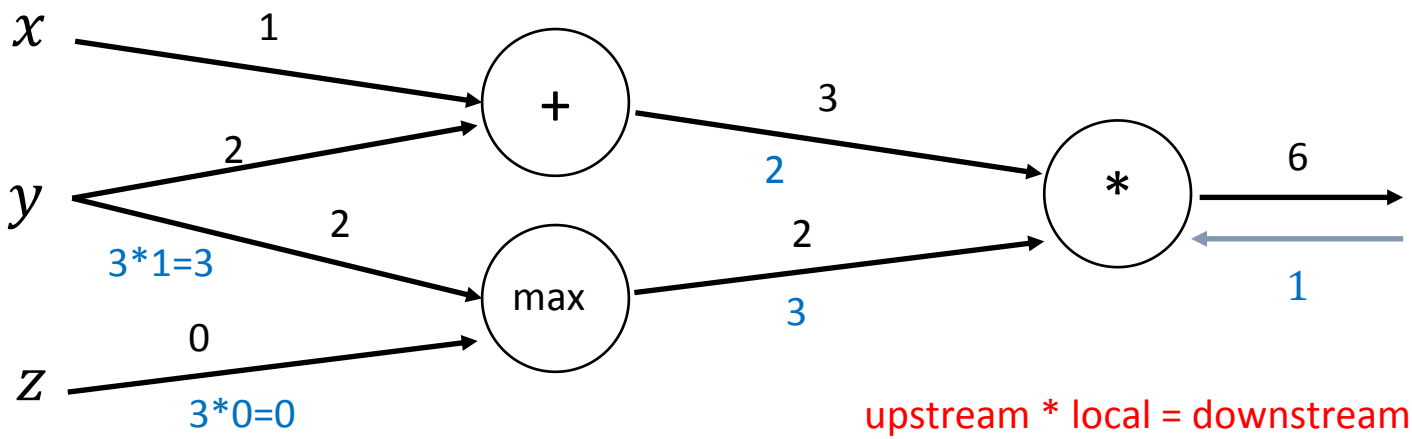
$$f = ab$$

Local gradients:

$$\frac{\partial a}{\partial x} = 1, \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1, \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b, \frac{\partial f}{\partial b} = a$$



upstream * local = downstream

An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps:

$$a = x + y$$

$$b = \max(y, z)$$

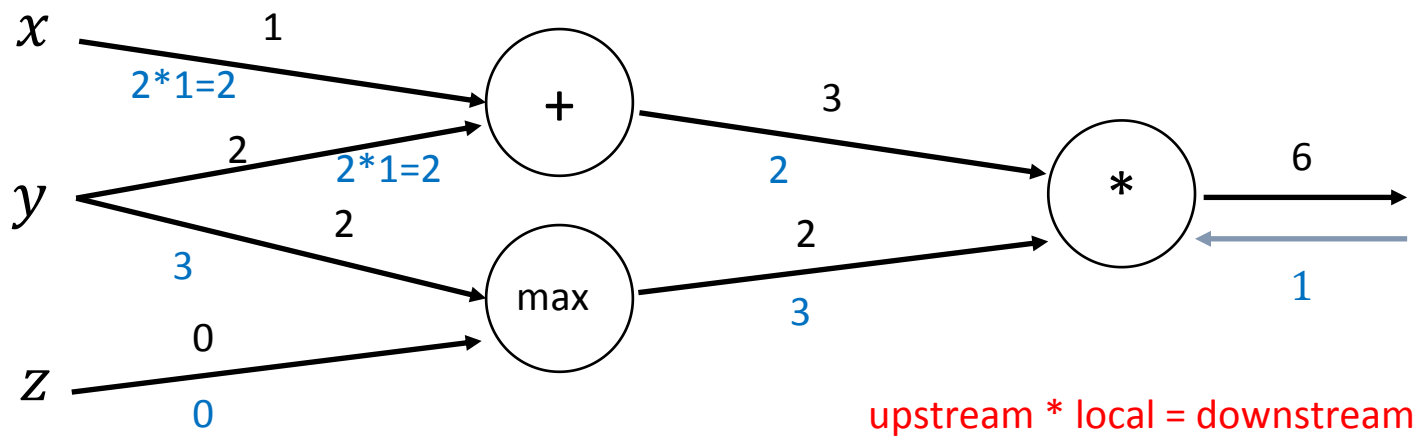
$$f = ab$$

Local gradients:

$$\frac{\partial a}{\partial x} = 1, \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1, \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b, \frac{\partial f}{\partial b} = a$$



An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

Forward prop steps:

$$a = x + y$$

$$b = \max(y, z)$$

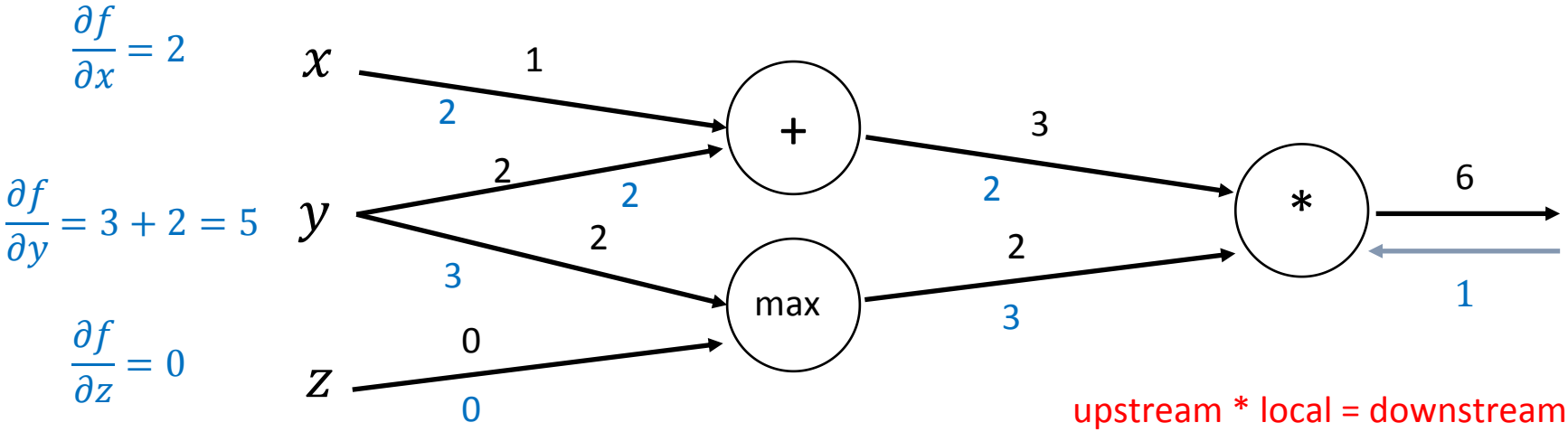
$$f = ab$$

Local gradients:

$$\frac{\partial a}{\partial x} = 1, \frac{\partial a}{\partial y} = 1$$

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1, \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$$

$$\frac{\partial f}{\partial a} = b, \frac{\partial f}{\partial b} = a$$



upstream * local = downstream

Summary

- Backpropagation: recursively apply the chain rule along computational graph
  - [downstream gradient] = [upstream gradient] x [local gradient]
- Forward pass: compute results of operation and save intermediate values
- Backward: apply chain rule to compute gradient

RNN & CNN
Sentence Modeling

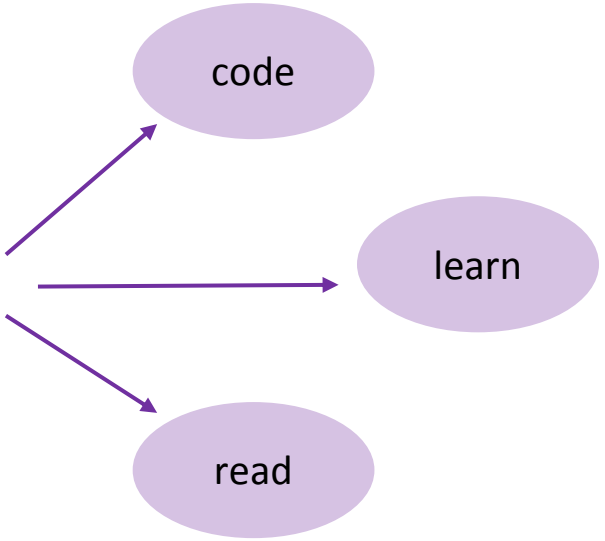
THUNLP

Review of Language Model

- Language Modeling is the task of predicting the upcoming word
  - Compute conditional probability of an upcoming word w_n:

$$P(w_n|w_1, w_2, \cdots, w_{n-1})$$

Never too late to _____



Review of Language Model

- N-gram Model
  - Collect statistics about how frequent different n-grams are, and use these to predict next word.
  - E.g., 4-gram

$$P(w_j|never\ to\ late\ to) = \frac{count(too\ late\ to\ w_j)}{count(too\ late\ to)}$$

  - Problem:
    - Need to store count for all possible *n*-grams. So model size is $O(\exp(n))$.

Review of Language Model

- The sparsity problem in traditional statistical models like N-gram cannot handle new expressions!
  - What if "$too\ late\ to$" never occurred in data?

- Neural architectures can solve the sparsity problem, which significantly benefit language modeling

Simple Neural Network

- Given a sequence of words, use separate parameters for each word of the time index

I	went	to	China	in	2019

$$f_1(I) + f_2(went) + f_3(to) + f_4(China) + f_5(in) + f_6(2019)$$

in	2019	I	went	to	China

$$f_1(in) + f_2(2019) + f_3(I) + f_4(went) + f_5(to) + f_6(China)$$

1. Inefficiency to adapt the change of order
2. Unable to modify a variable length

Better Neural Architectures

- Similar to language modeling, many sentence-level NLP tasks need a model that can process any length input.

- Parameter sharing makes it possible to extend and apply the model to examples with different length.

Better Neural Architectures

- Mainstream networks for sentence modeling
  - Recurrent Neural Networks (RNNs)
  - Convolutional Neural Networks (CNNs)

Recurrent Neural Networks (RNNs)

THUNLP

Sequential Memory

- Key concept for RNNs: Sequential memory during processing sequence data
- Sequential memory of human:
  - Say the alphabet in your head

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

  - Pretty easy

Sequential Memory

- Key concept for RNNs: Sequential memory during processing sequence data
- Sequential memory of human:
  - Say the alphabet backward

Z Y X W V U T S R Q P O N M L K J I H G F E D C B A

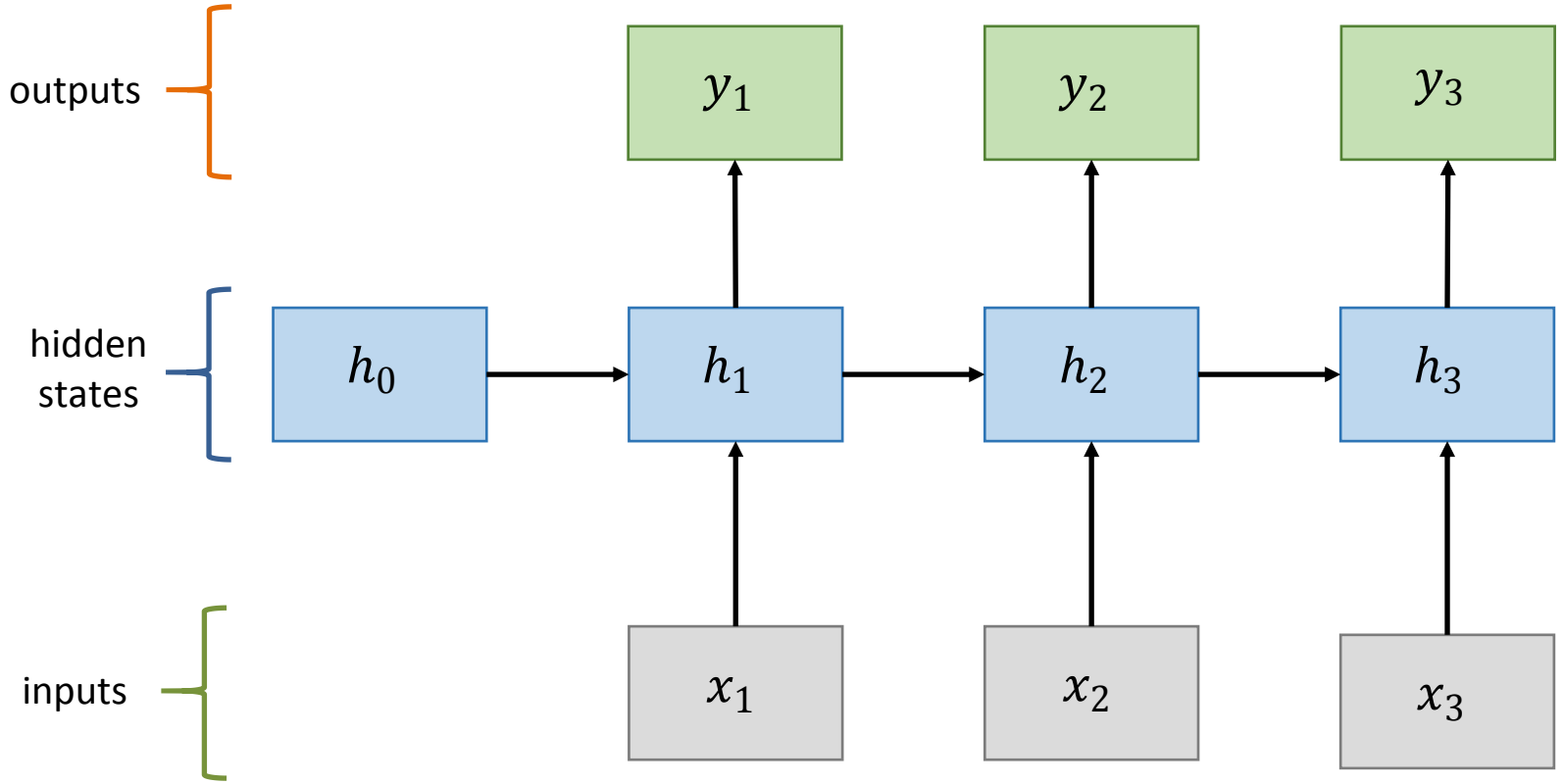
  - Much harder

Sequential Memory

- Definition: a mechanism that makes it easier for your brain to recognize sequence patterns

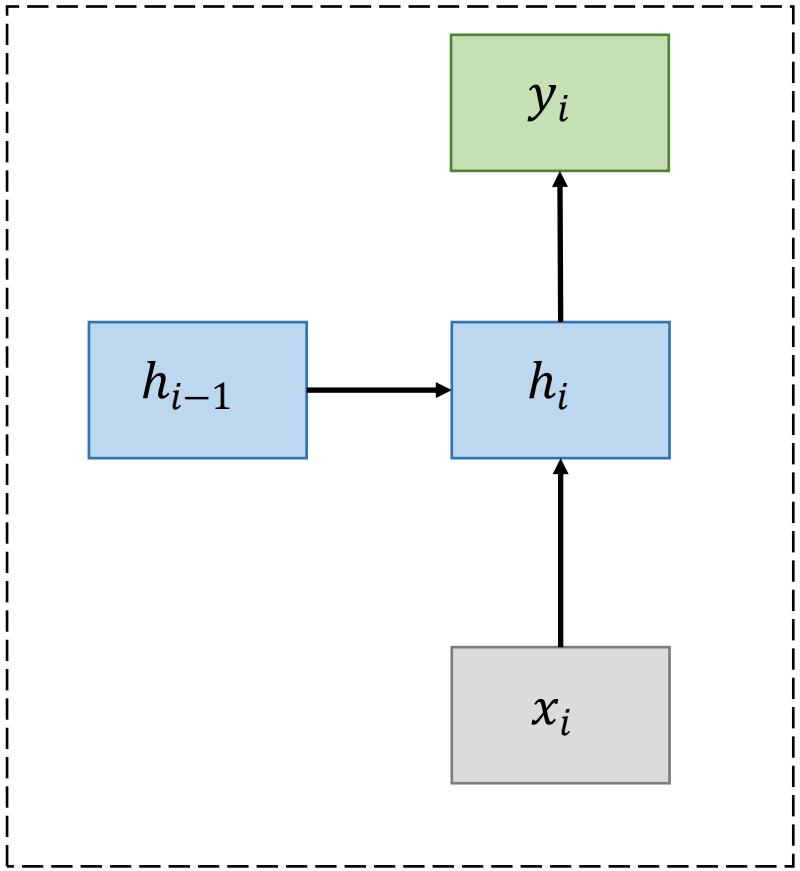
- RNNs update the sequential memory recursively for modeling sequence data

Recurrent Neural Networks



Recurrent Neural Networks

- RNN Cell



$$h_i = \tanh\left(W_x x_i + W_h h_{i-1} + b\right)$$

$$y_i = F(h_i)$$

Recurrent Neural Networks

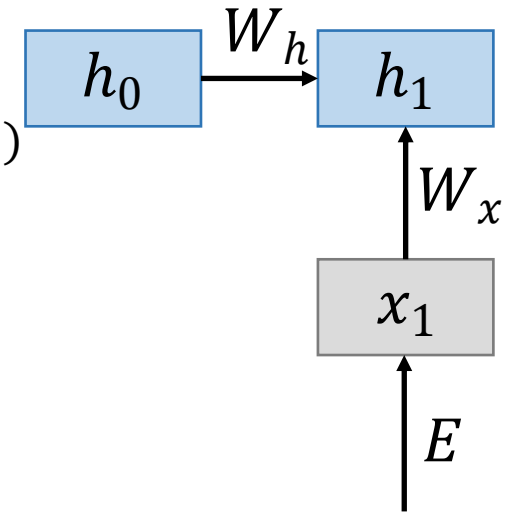
- Advantages:
  - Can process any length input
  - Model size does not increase for longer input
  - Weights are shared across timesteps
  - Computation for step i can (in theory) use information from many steps back

- Disadvantages:
  - Recurrent computation is slow
  - In practice, it's difficult to access information from many steps back

RNN Language Model

hidden states

$$h_i = \tanh\left(W_x x_i + W_h h_{i-1} + b_1\right)$$



word embeddings

$$x_i = E w_i$$

one-hot vectors

$$w_i \in \mathbb{R}^{|V|}$$

RNN Language Model

hidden states

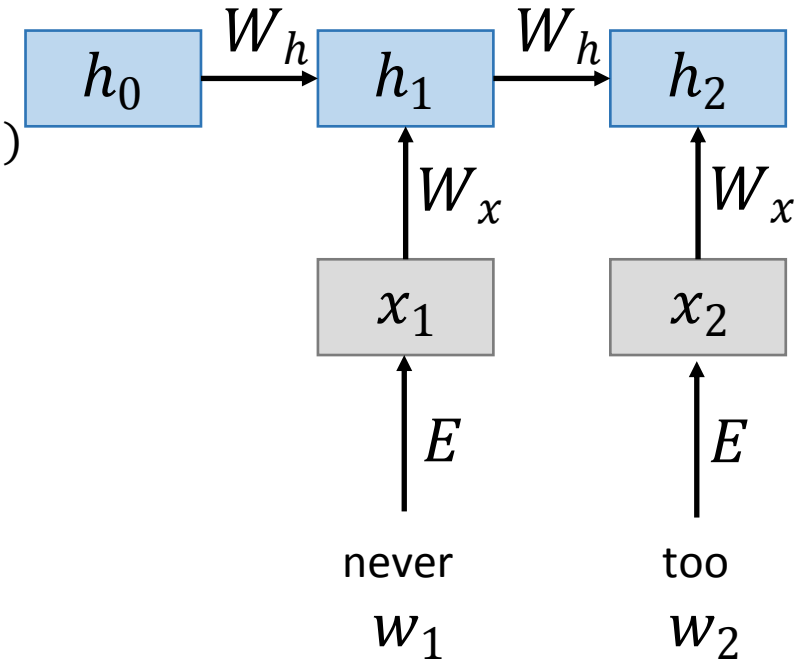
$$h_i = \tanh\left(\begin{matrix} W_x x_i + \\ W_h h_{i-1} + b_1 \end{matrix}\right)$$

word embeddings

$$x_i = E w_i$$

one-hot vectors

$$w_i \in \mathbb{R}^{|V|}$$



RNN Language Model

hidden states

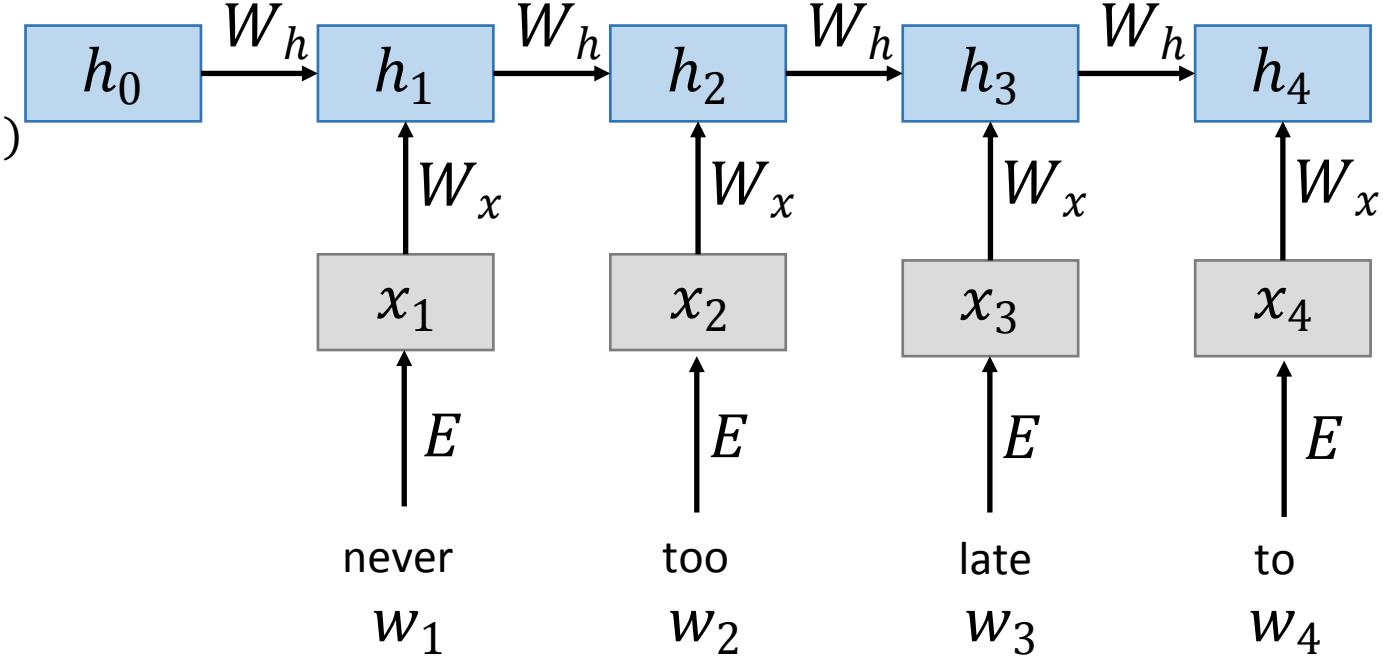
$$h_i = \tanh\left(\begin{matrix} W_x x_i + \\ W_h h_{i-1} + b_1 \end{matrix}\right)$$

word embeddings

$$x_i = E w_i$$

one-hot vectors

$$w_i \in \mathbb{R}^{|V|}$$



RNN Language Model

output distribution

$y_4 = softmax(Uh_4 + b_2) \in \mathbb{R}^{|V|}$

hidden states

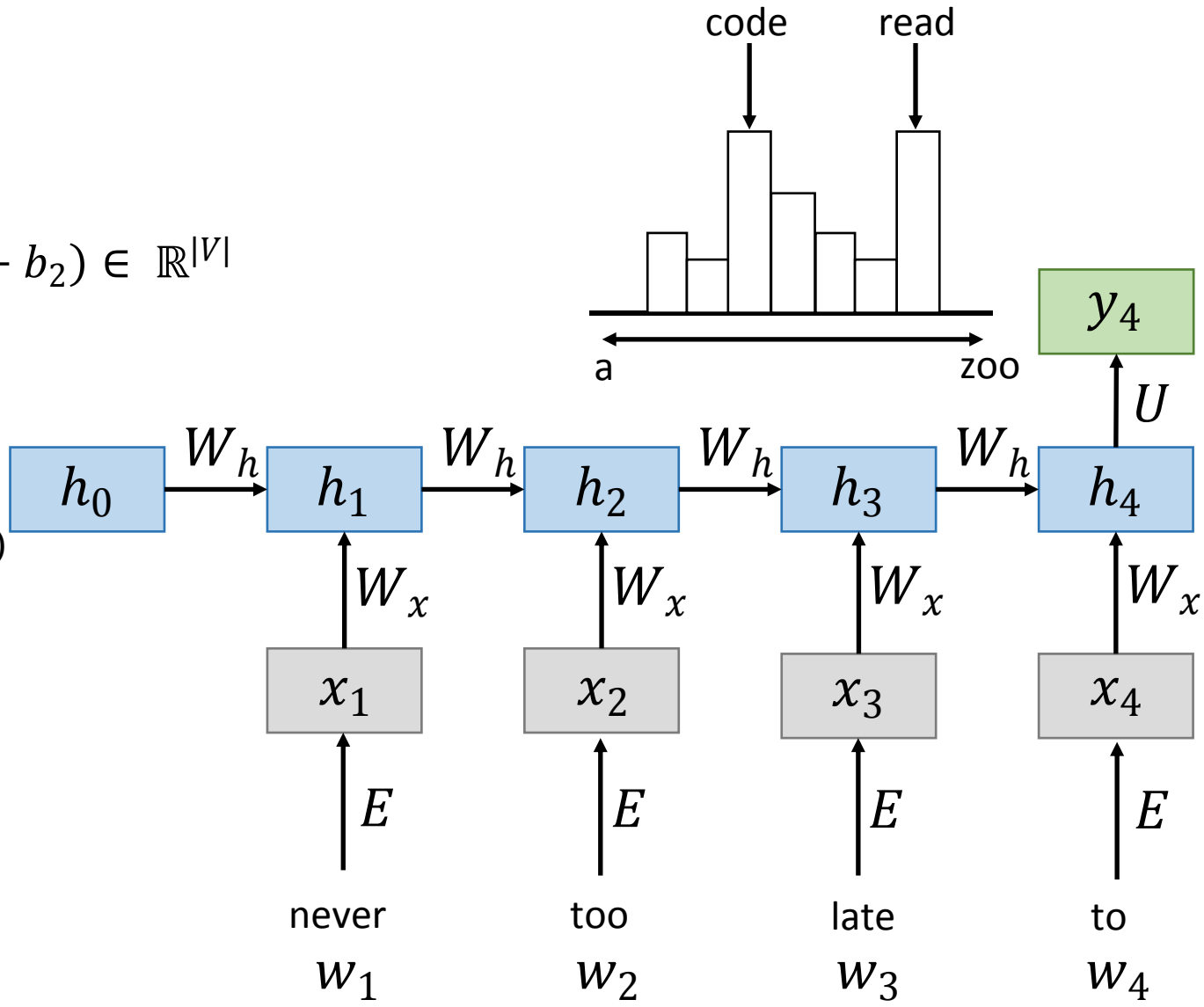
$h_i = \tanh\left(\begin{matrix} W_x x_i + \\ W_h h_{i-1} + b_1 \end{matrix}\right)$

word embeddings

$x_i = Ew_i$

one-hot vectors

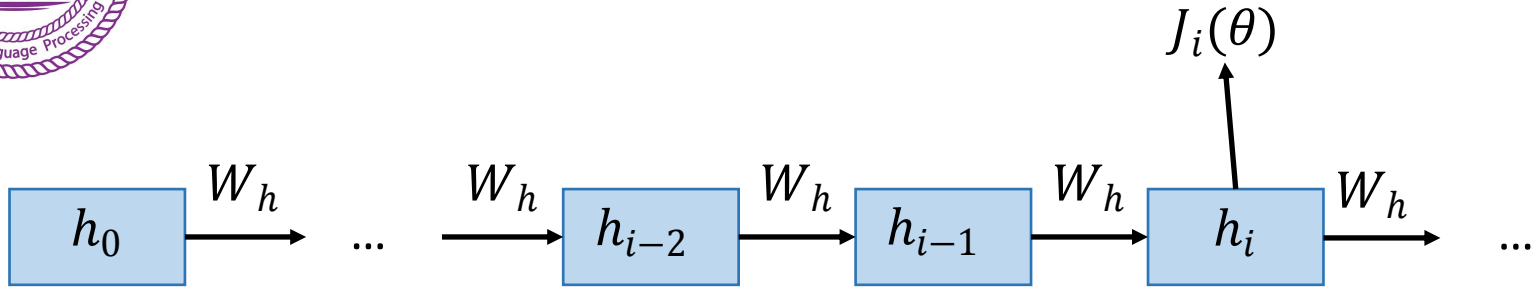
$w_i \in \mathbb{R}^{|V|}$



Training RNN Language Model

- Get a big corpus which is a sequence of words $w_1, w_2, \cdots, w_n$
- Using RNN, compute output distribution y_i for every step i
  - Predict probability distribution of every word, given words so far
- Loss function on step i is usual cross-entropy between our predicted probability distribution and the true next word
  - $J_i(\theta) = CE(w_{i+1},\ y_i) = -\sum_{j=1}^{|V|} w_{i+1,j} \log y_{i,j}$

Gradient Problem with Vanilla RNN



- Question
  - What is the derivative of $J_i(\theta)$ w.r.t. the repeated weight matrix W_h?

The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears.

- Answer

$$\frac{\partial J_i}{\partial W_h} = \sum_{k=1}^{i} \frac{\partial J_i}{\partial y_i} \frac{\partial y_i}{\partial h_i} \frac{\partial h_i}{\partial h_k} \frac{\partial h_k}{\partial W_h}$$

Gradient Problem with Vanilla RNN

- The derivative of $J_i(\theta)$

$$\frac{\partial J_i}{\partial W_h} = \frac{\partial J_i}{\partial y_i}\frac{\partial y_i}{\partial h_i}\sum_{k=1}^{i}\frac{\partial h_i}{\partial h_k}\frac{\partial h_k}{\partial W_h}$$

- Recurrent states:

$$h_i = \tanh\left(W_x x_i + W_h h_{i-1} + b_1\right)$$

- Another parametrization:

$$h_i = W_x x_i + W_{\mathrm{h}}\tanh(\mathrm{h_{i-1}}) + \mathrm{b_1}$$

- For convenience, we use the second equation to analyze the gradient problem

Gradient Problem with Vanilla RNN

- More chain rule:

$$h_i = W_x \mathrm{x_i} + \mathrm{W_h} \tanh(\mathrm{h_{i-1}}) + \mathrm{b_1}$$

$$\frac{\partial h_i}{\partial h_k} = \Pi_{j=k+1}^{i} \frac{\partial h_j}{\partial h_{j-1}}$$

$$\left|\left| \frac{\partial h_j}{\partial h_{j-1}} \right|\right| \leq ||W_h|| * ||diag[\tanh'(h_{i-1})]||$$

$$\left|\left| \frac{\partial h_i}{\partial h_k} \right|\right| = \left|\left| \Pi_{j=k+1}^{i} \frac{\partial h_j}{\partial h_{j-1}} \right|\right| \leq (\beta_W \beta_h)^{i-k}$$

  - where we defined β_W, β_h as the upper bounds of $W_h, diag[\tanh'(h_{i-1})]$ norms

Gradient Problem with Vanilla RNN

- In the same way a product of k real numbers can shrink to zero or explode to infinity, so can a product of matrices
- When $\beta_x \beta_h < 1$, it will lead to the vanishing gradients problem
- When $\beta_x \beta_h > 1$, it will lead to the exploding gradients problem
- Gradients can be seen as a measure of influence of the past on the future

Gradient Problem with Vanilla RNN

- 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.

$$\frac{\partial h_i}{\partial h_k} = \Pi_{j=k+1}^{i} \frac{\partial h_j}{\partial h_{j-1}} \approx 0$$

RNN Variants

THUNLP

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
  - GRU
  - LSTM
- Main ideas:
  - Keep around memories to capture long distance dependencies

Gated Recurrent Unit (GRU)

THUNLP

Gated Recurrent Unit (GRU)

- Vanilla RNN computes hidden layer at next time step directly:

$$h_i = \tanh\left(W_x x_i + W_h h_{i-1} + b\right)$$

- Introduce gating mechanism into RNN
- Update gate

$$z_i = \sigma\left(W_x^{(z)} x_i + W_h^{(z)} h_{i-1} + b^{(z)}\right)$$

- Reset gate

$$r_i = \sigma\left(W_x^{(r)} x_i + W_h^{(r)} h_{i-1} + b^{(r)}\right)$$

- Gates are used to balance the influence of the past and the input

Gated Recurrent Unit (GRU)

- Update gate

$$z_i = \sigma\left(W_x^{(z)} x_i + W_h^{(z)} h_{i-1} + b^{(z)}\right)$$

- Reset gate

$$r_i = \sigma\left(W_x^{(r)} x_i + W_h^{(r)} h_{i-1} + b^{(r)}\right)$$

- New activation $\tilde{h}_i$

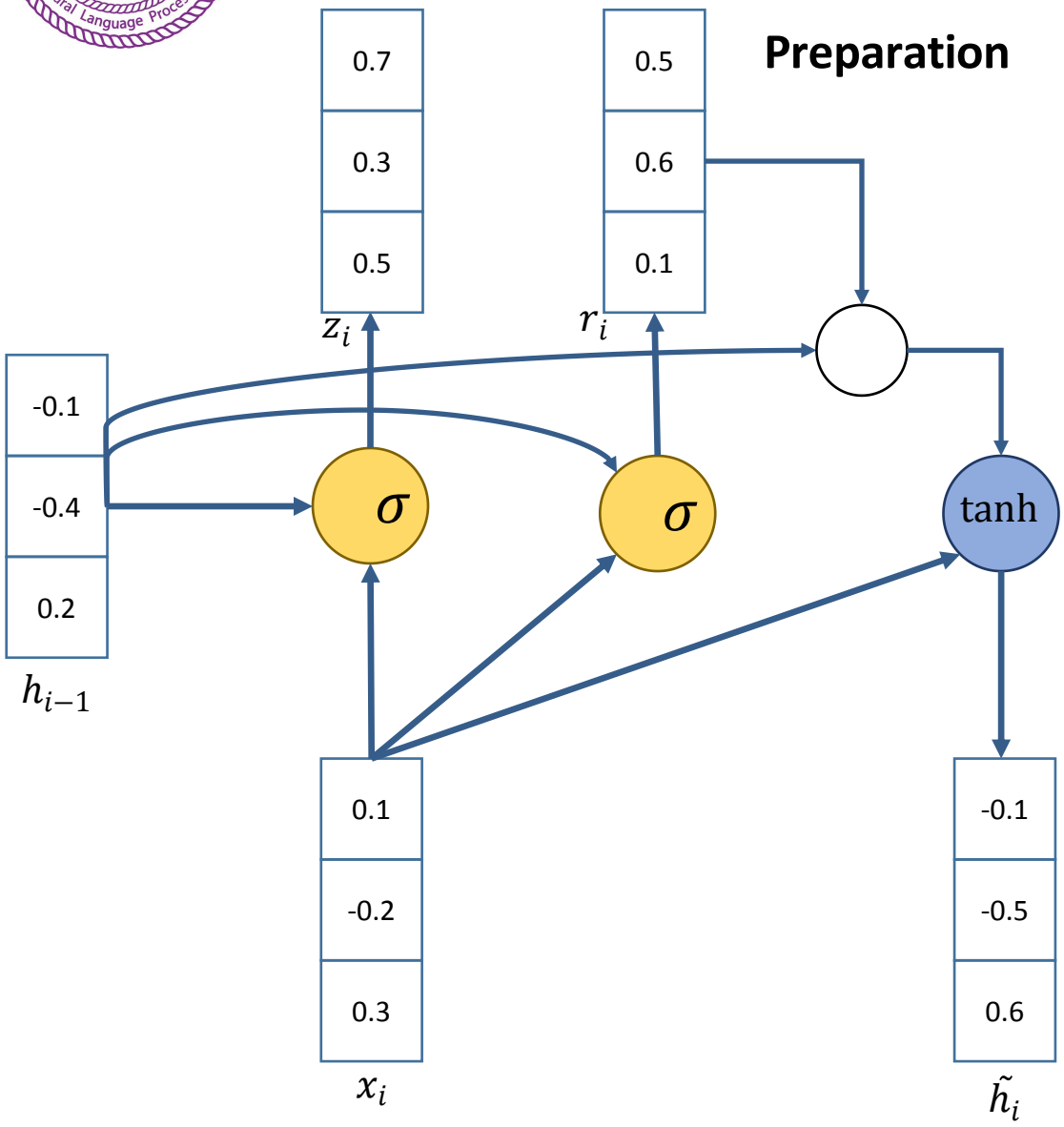
$$\tilde{h}_i = \tanh\left(W_x x_i + r_i * W_h h_{i-1} + b\right)$$

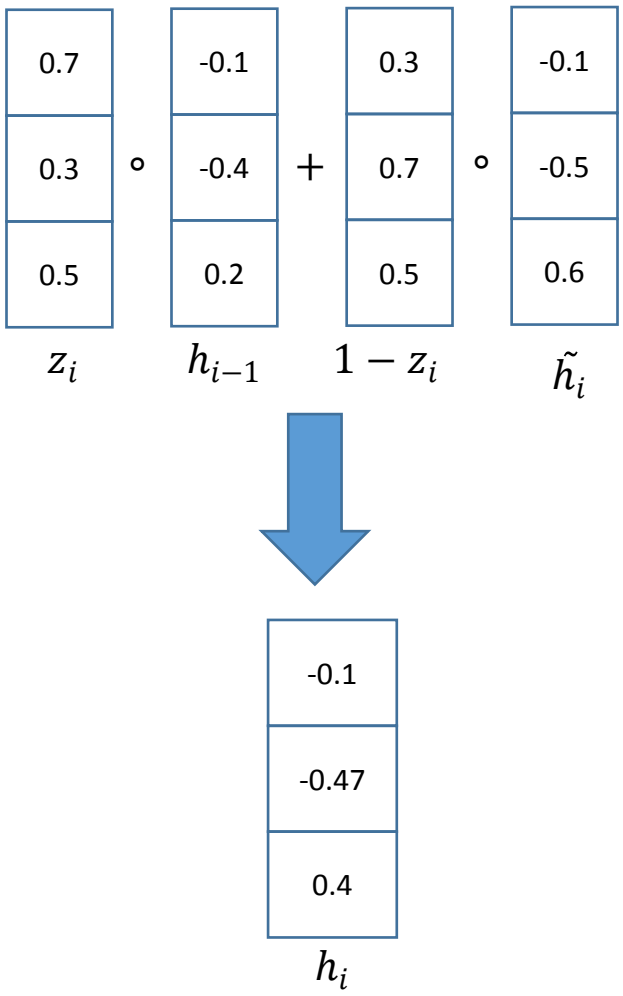
- Final hidden state h_i

$$h_i = z_i * h_{i-1} + (1 - z_i) * \tilde{h}_i$$

- Where $*$ refers to element-wise product

Gated Recurrent Unit (GRU)





Gated Recurrent Unit (GRU)

- If reset r_i is close to 0

$$\tilde{h}_i \approx \tanh\left(W_x x_i + 0 * W_h h_{i-1} + b\right)$$

$$\tilde{h}_i \approx \tanh\left(W_x x_i + b\right)$$

- Ignore previous hidden state, which indicates the current activation is irrelevant to the past.

- E.g., at the beginning of a new article, the past information is useless for the current activation.

Gated Recurrent Unit (GRU)

- Update gate z_i controls how much of past state should matter compared to the current activation.

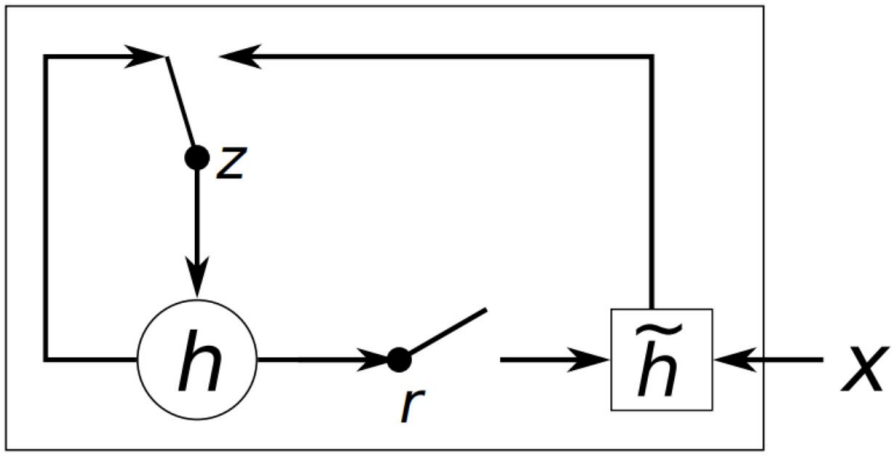
- If z_i close to 1, then we can copy information in that unit through many time steps! (Recall "Constant Error Flow")

$$h_i = 1 * h_{i-1} + (1 - 1) * \tilde{h}_i = h_{i-1}$$

- If z_i close to 0, then we drop information in that unit and fully take the current activation.

Gated Recurrent Unit (GRU)

- Units with short-term dependencies often have reset gates r_i very active
- Units with long term dependencies have active update gates z_i
- The graphical illustration:

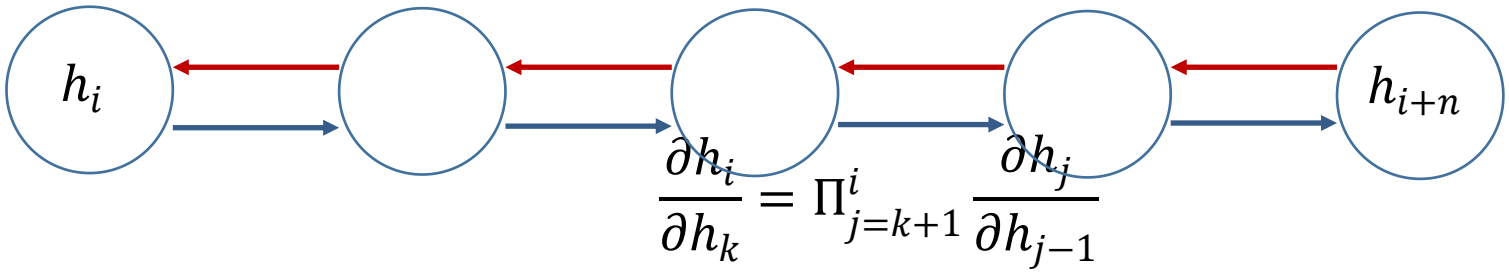


GRU for Vanishing Gradient Problem

- Recall the vanishing gradient problem with the transition function of vanilla RNNs

$$h_i = \tanh\left(W_x x_i + W_h h_{i-1} + b\right)$$

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

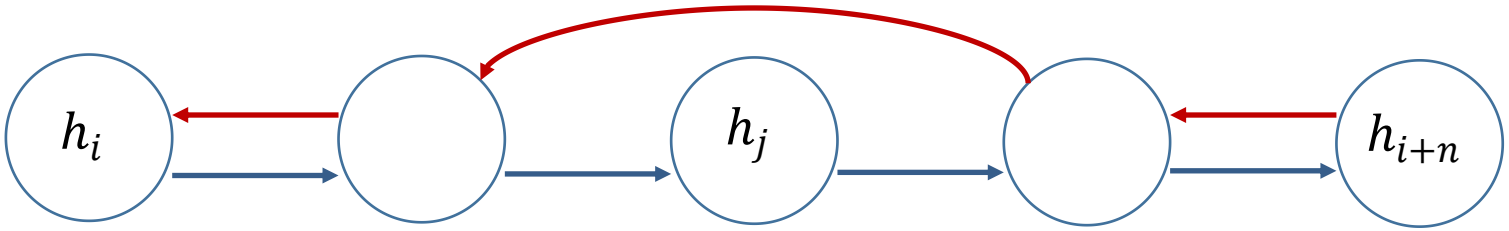


Create shortcut connection for better backpropagation!

GRU for Vanishing Gradient Problem

- Use update gate z_t to prune unnecessary connections adaptively.

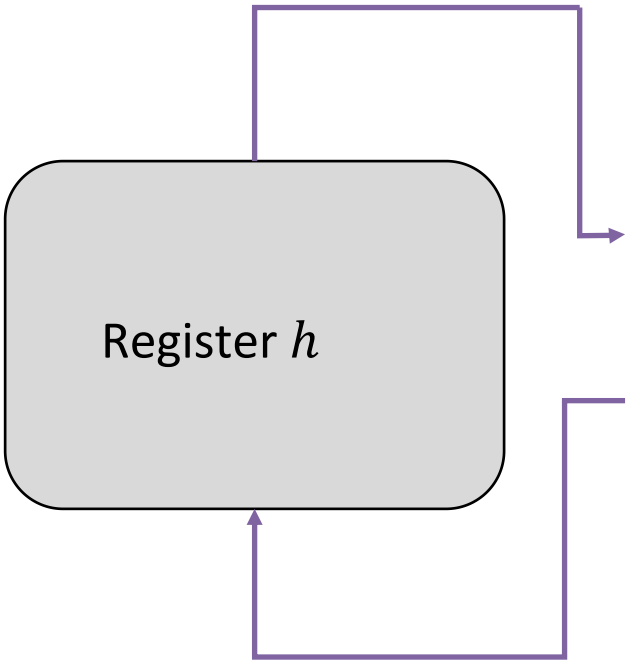
$$h_j = 1 * h_{j-1} + (1-1) * \tilde{h}_j = h_{j-1}$$



- We have adaptive shortcut connections with gates, which prevents the gradient from vanishing during backpropagation.

GRU Comparison to Vanilla RNN

- We treat the hidden state h as the information register for sentence modeling
- Vanilla RNN



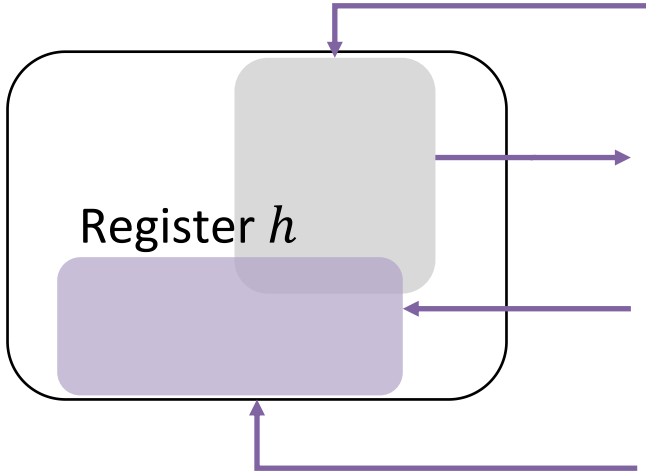
Operation:

1. Read the whole register h
2. Update the whole register

$$h_i = \tanh\left(W_x x_i + W_h h_{i-1} + b\right)$$

GRU Comparison to Vanilla RNN

- GRU



Operation:

1. Select a readable subset
2. Read the subset
3. Select a writable subset
4. Update the subset

$$h_i = z_i \circ h_{i-1} + (1 - z_i) \circ \tilde{h}_i$$

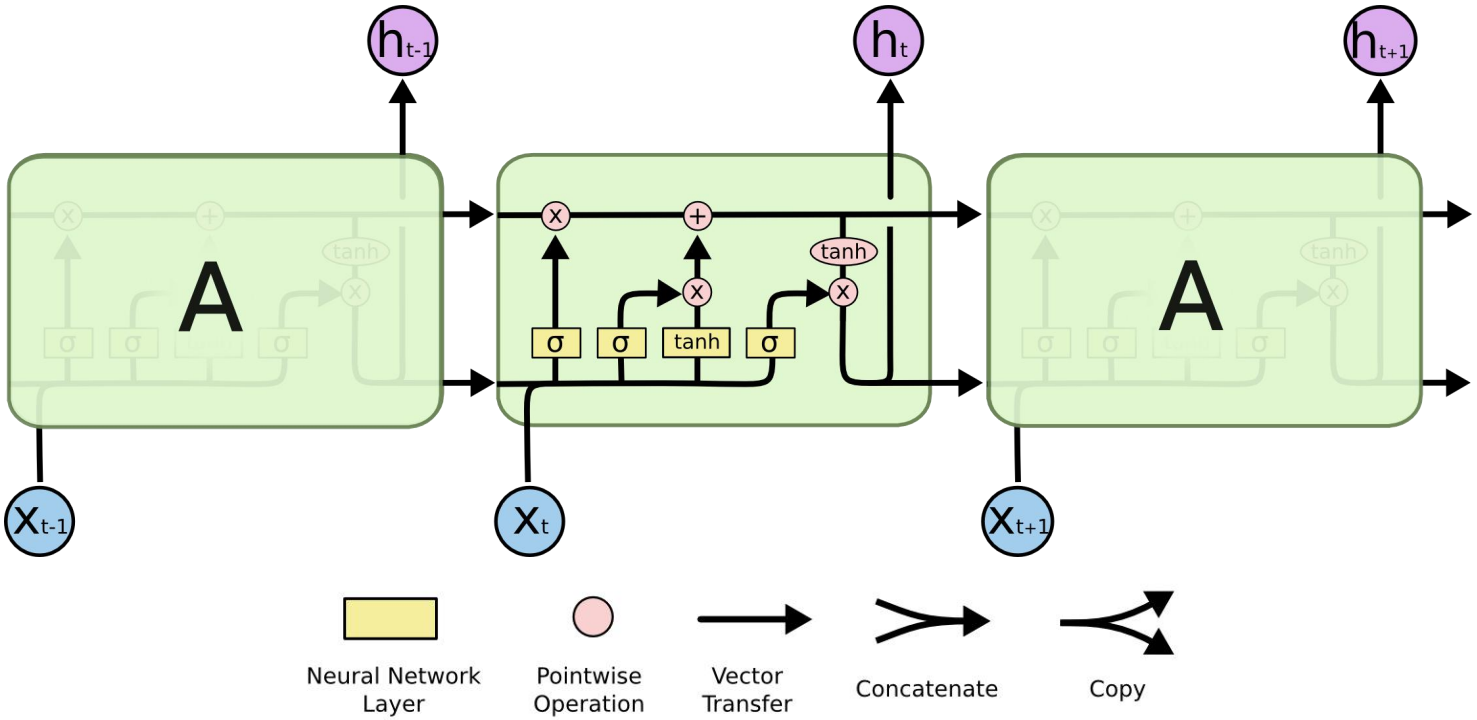
GRU are much more adaptive in updating the hidden state!

Long Short-Term Memory Network (LSTM)

THUNLP

Long Short-Term Memory Network

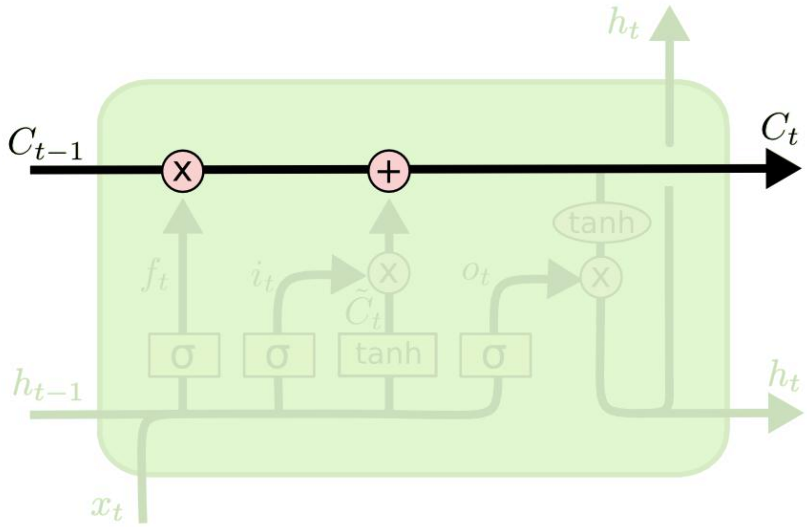
- Long Short-Term Memory network (LSTM)
- LSTM are a special kind of RNN, capable of learning long-term dependencies like GRU



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

Long Short-Term Memory Network

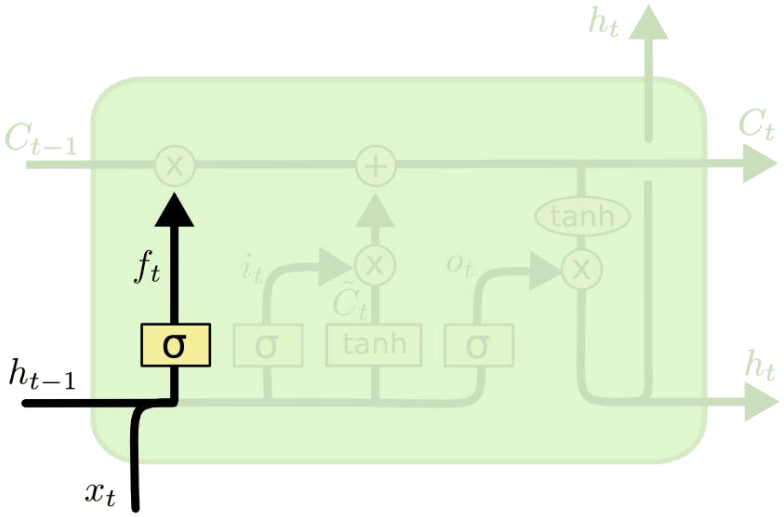
- The key to LSTMs is the cell state C_t



- Extra vector for capturing long-term dependency
- Runs straight through the entire chain, with only some minor linear interactions
- Easy to remove or add information to the cell state

Long Short-Term Memory Network

- The first step is to decide what information to throw away from the cell state
- Forget gate f_t

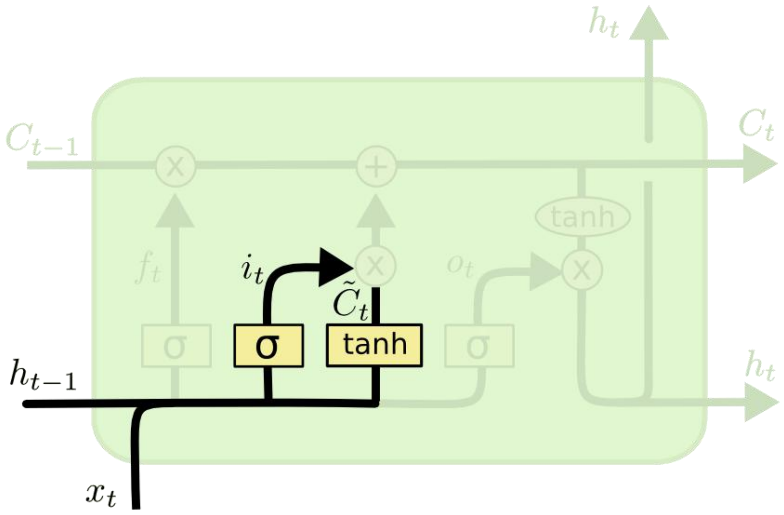


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- Where $[h_{t-1},\ x_t]$ is the concatenation of vectors
- Forget past if $f_t = 0$

Long Short-Term Memory Network

- The next step is to decide what information to store in the cell state
- Input gate i_t and new candidate cell state $\tilde{C}_t$



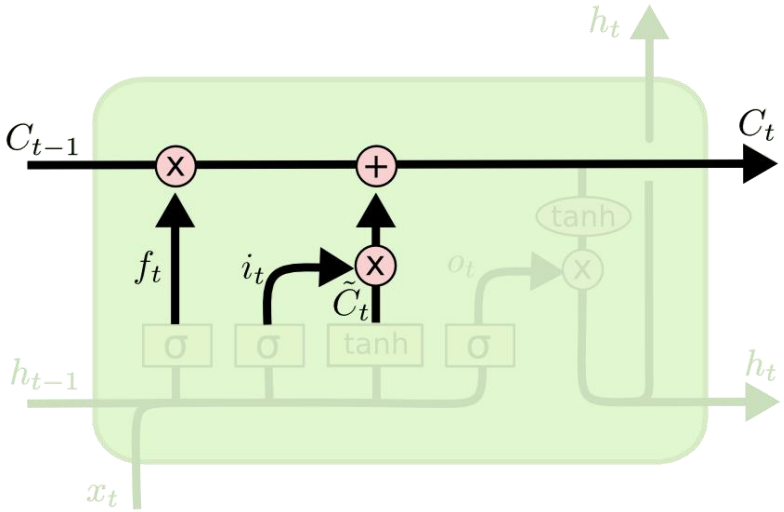
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- Recall z_t and $\tilde{h}_t$ in GRUs

Long Short-Term Memory Network

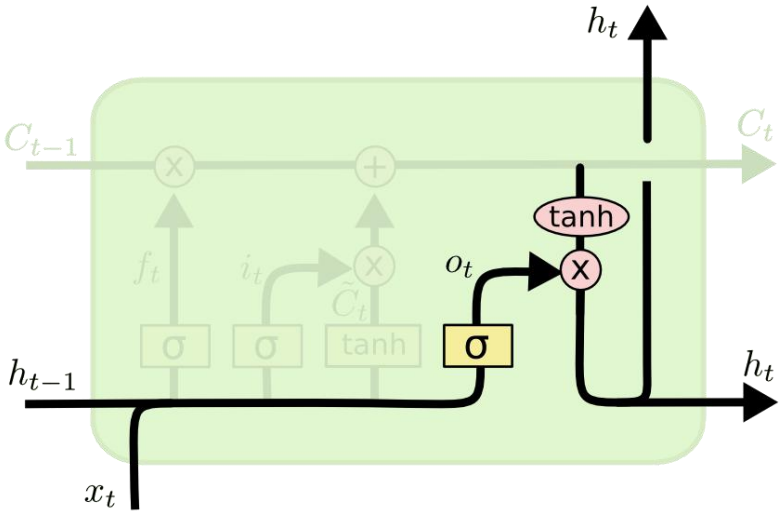
- Update the old cell state C_{t-1}
- Combine the results from the previous two steps



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Long Short-Term Memory Network

- The final step is to decide what information to output
- Adjust the sentence information for a specific word representation



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Long Short-Term Memory Network

- Powerful especially when stacked and made even deeper (each hidden layer is already computed by a deep internal network)

- Very useful if you have plenty of data

Bidirectional RNNs

THUNLP

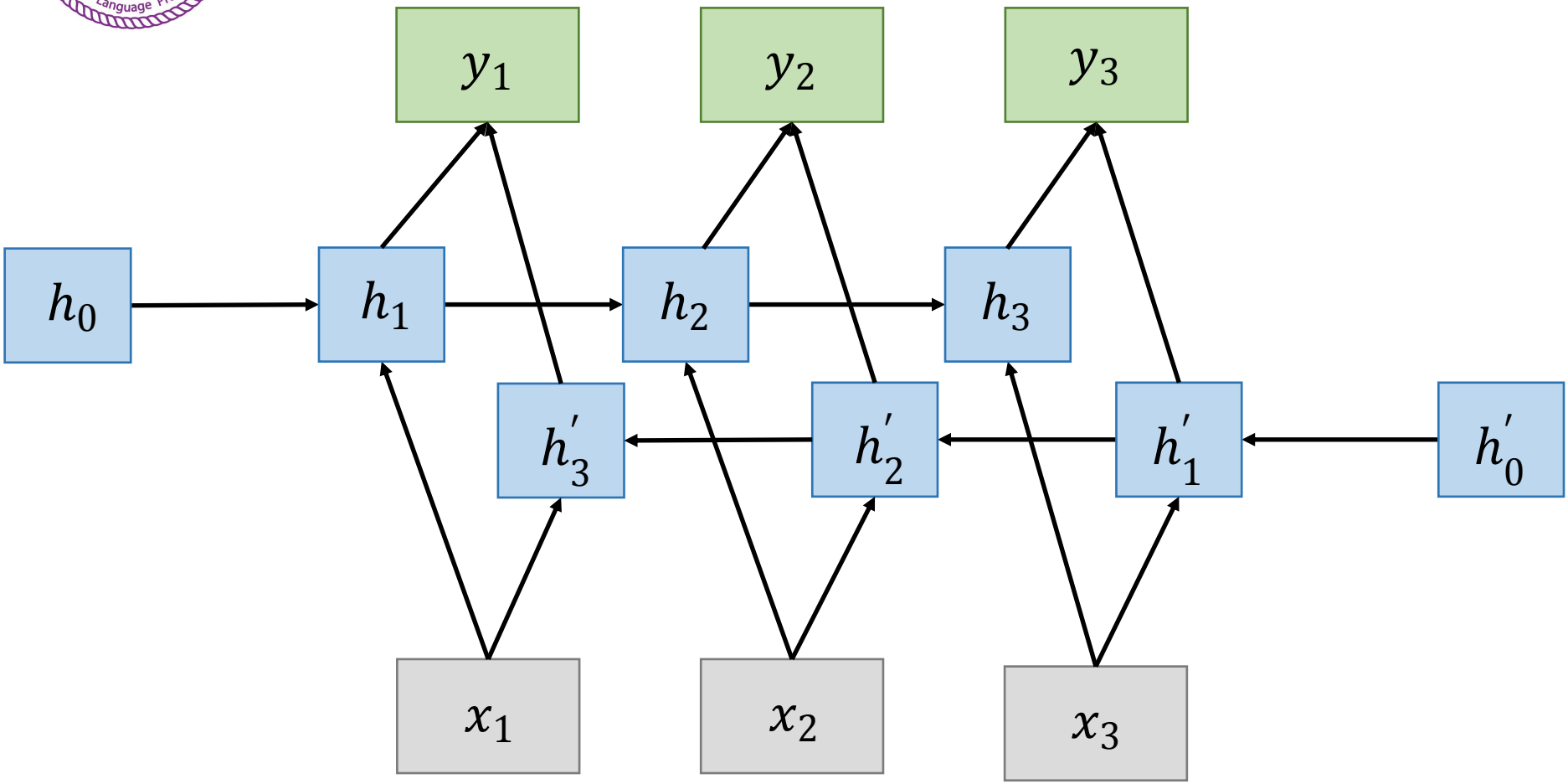
Bidirectional RNNs

- In traditional RNNs, the state at time t only captures information from the past

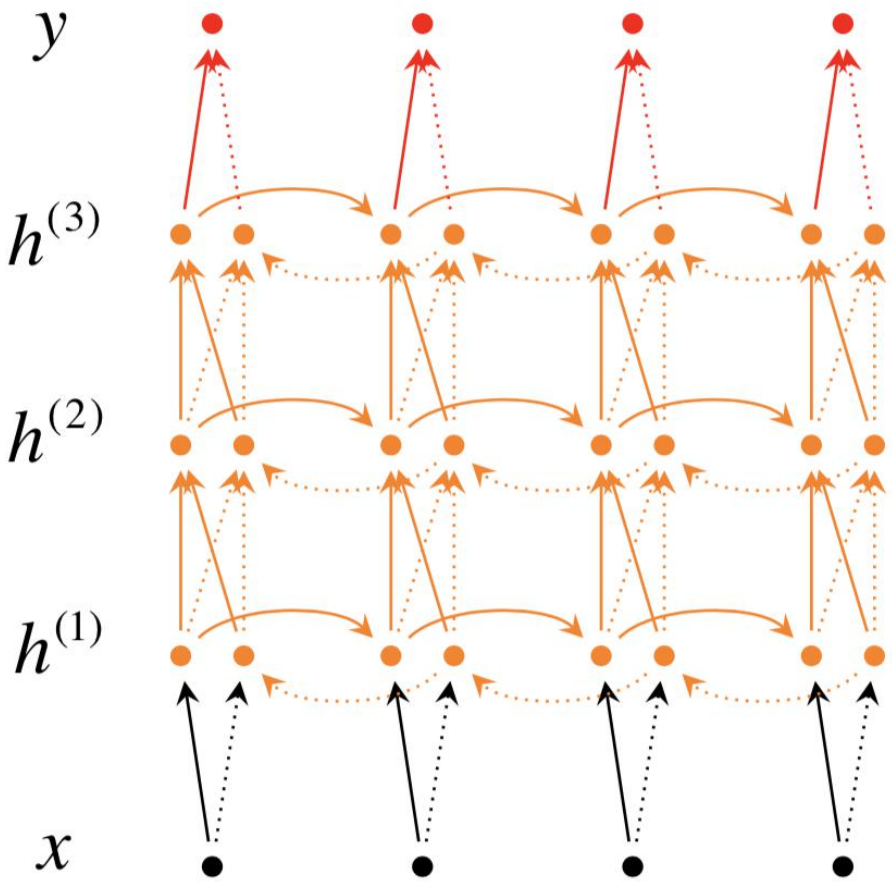
$$h_t = f(x_{t-1}, \ldots, x_2, x_1)$$

- Problem: in many applications, we want to have an output y_t depending on the whole input sequence
- For example
  - Handwriting recognition
  - Speech recognition

Bidirectional RNNs



Deep Bidirectional RNNs



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

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

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

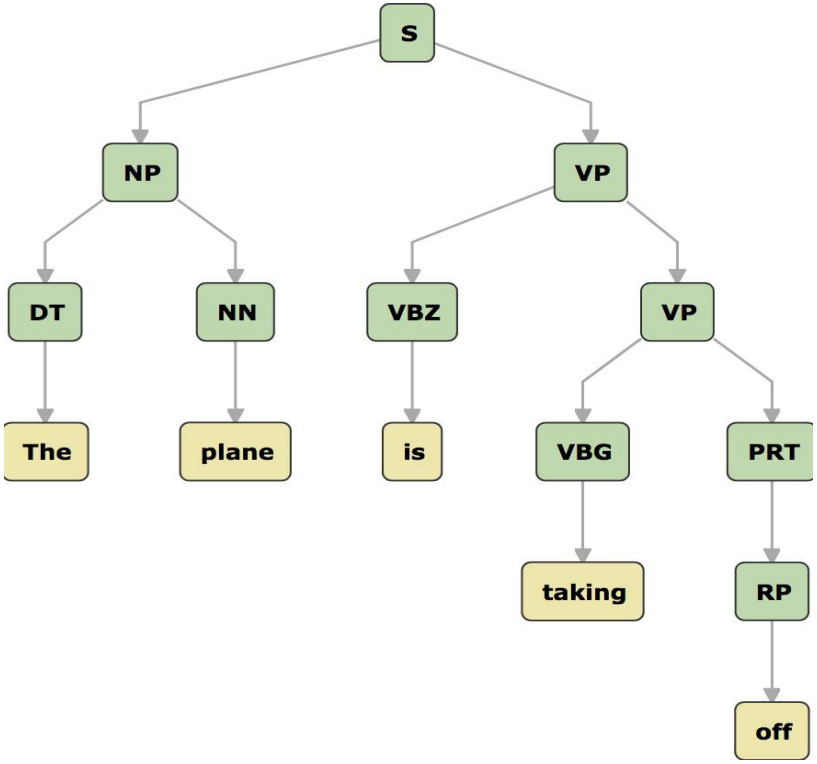
Has multiple layers per time step

Tree LSTM

THUNLP

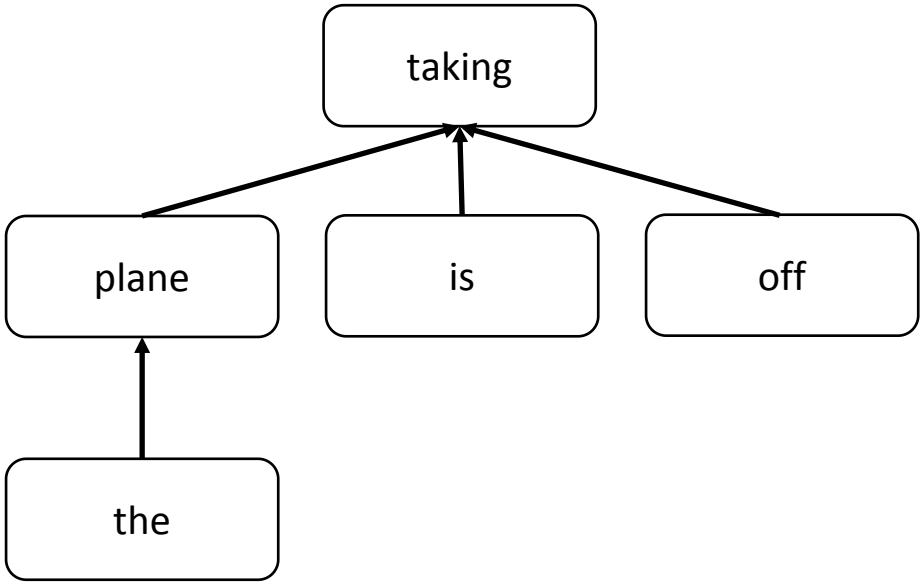
Sentence Structure

- Sentence is not a simple linear sequence
- Constituency-based parse tree
  - *The plane is taking off*



Sentence Structure

- Dependency-based parse tree
  - *The plane is taking off*



Sentence Structure

- Multi-step backpropagation -> Vanishing gradient problem
- The structure of sentence: directly present the long term dependencies -> Solve the vanishing gradient problem !
- Improve encoding of sentences:
  - Better backpropagation
  - Encoding extra structured information
- Two variants
  - Child-sum tree LSTM
  - N-ary tree LSTM

Child-sum Tree LSTM

- Sum over all the children of a node: can be used for any number of children
  - Hidden state: $\tilde{h}_j = \sum_{k \in C(j)} h_k$
  - Input gate: $i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right)$
  - Forget gate: $f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right)$
  - Output gate: $o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right)$
  - New memory cell: $u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right)$
  - Final memory cell: $c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k$
  - Final hidden state: $h_j = o_j \odot \tanh\left(c_j\right)$

Child-sum Tree LSTM

- The order of sequence is lost
- Suitable for trees with high branching factor
  - Branching factor: the outdegree, the number of children at each node
- Work with variable number of children
- Share gates' weight (including forget gate) among children
- Application
  - Dependency Tree-LSTM

N-ary Tree LSTM

- Use different parameters for each node
  - Input gate:
  - Forget gate:
  - Output gate:
  - New memory cell:
  - Final memory cell:
  - Final hidden state:

$$i_j = \sigma\left(W^{(i)}x_j + \sum_{\ell=1}^{N} U_{\ell}^{(i)} h_{j\ell} + b^{(i)}\right)$$

$$f_{jk} = \sigma\left(W^{(f)}x_j + \sum_{\ell=1}^{N} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right)$$

$$o_j = \sigma\left(W^{(o)}x_j + \sum_{\ell=1}^{N} U_{\ell}^{(o)} h_{j\ell} + b^{(o)}\right)$$

$$u_j = \tanh\left(W^{(u)}x_j + \sum_{\ell=1}^{N} U_{\ell}^{(u)} h_{j\ell} + b^{(u)}\right)$$

$$c_j = i_j \odot u_j + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell}$$

$$h_j = o_j \odot \tanh\left(c_j\right)$$

N-ary Tree LSTM

- Each node must have at most N children
- Forget gate can be parameterized so that the siblings affect each other
- Application
  - Constituency Tree-LSTM

Summary

- Recurrent Neural Network
  - Sequential Memory
  - Vanishing gradient problem
- RNN Variants
  - Gated Recurrent Unit (GRU)
  - Long Short-Term Memory Network (LSTM)
  - Bidirectional Recurrent Neural Network
  - Tree LSTM

Convolutional Neural Networks (CNNs)

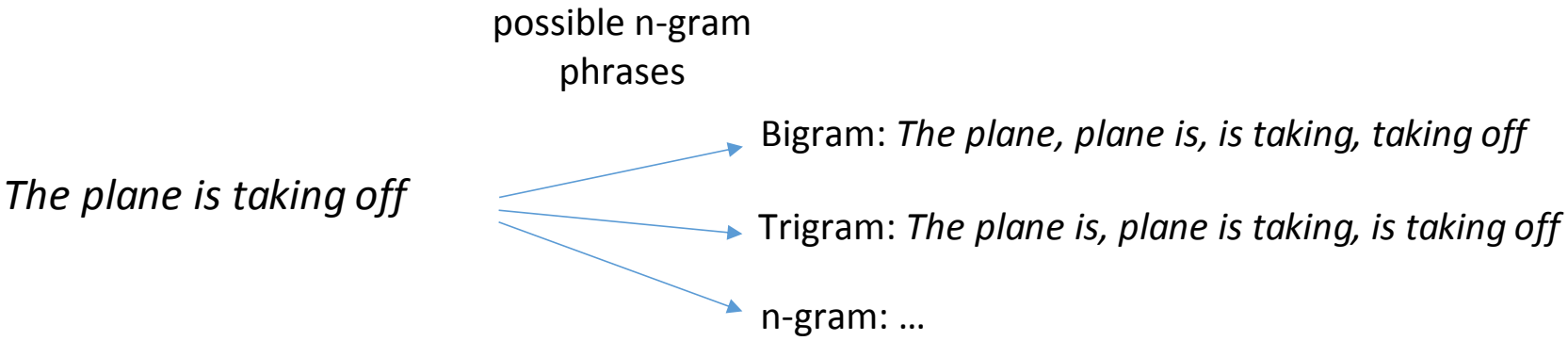
THUNLP

CNN for Sentence Representation

- Convolutional Neural Networks (CNNs)
  - Generally used in Computer Vision (CV)
  - Achieve promising results in a variety of NLP tasks:
    - Sentiment classification
    - Relation classification
    - …
- CNNs are good at extracting local and position-invariant patterns
  - In CV, colors, edges, textures, etc.
  - In NLP, phrases and other local grammar structures

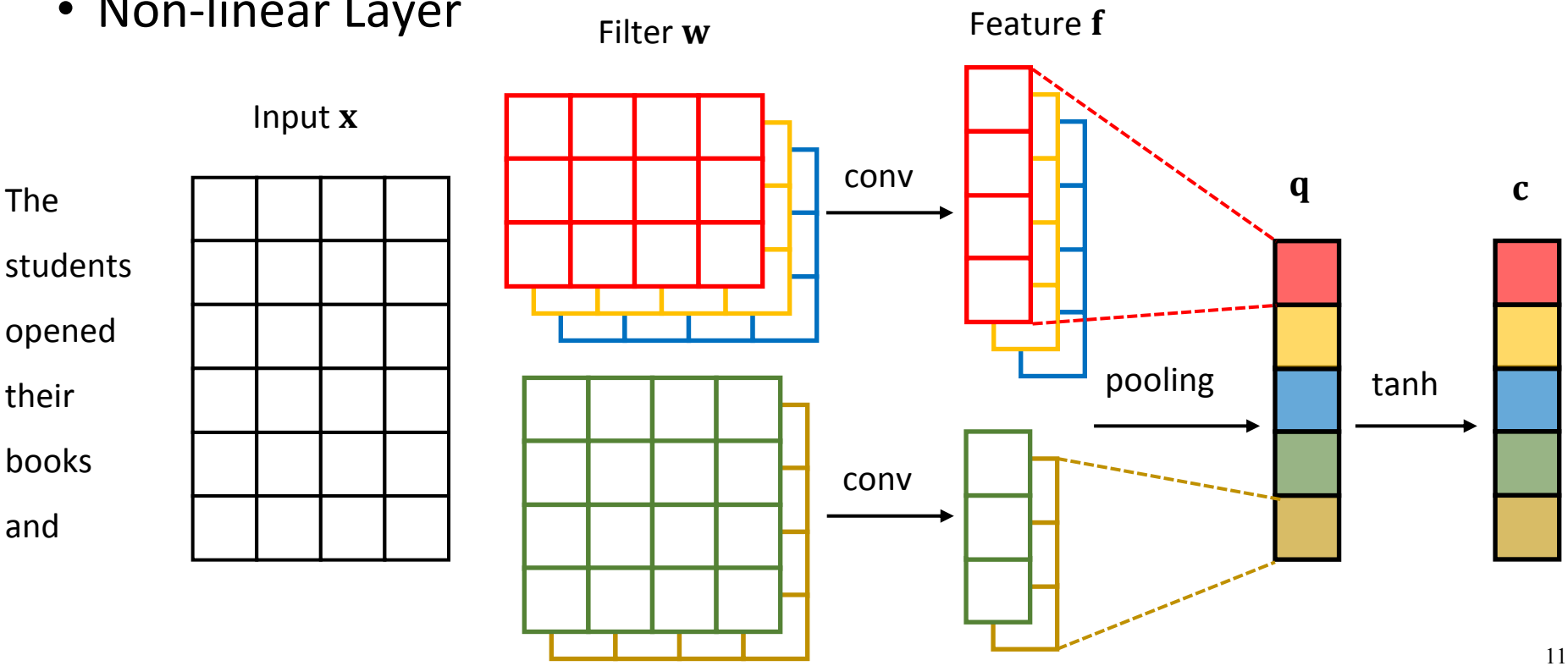
CNN for Sentence Representations

- CNNs extract patterns by:
  - Computing representations for all possible n-gram phrases in a sentence.
  - Without relying on external linguistic tools (e.g., dependency parser)



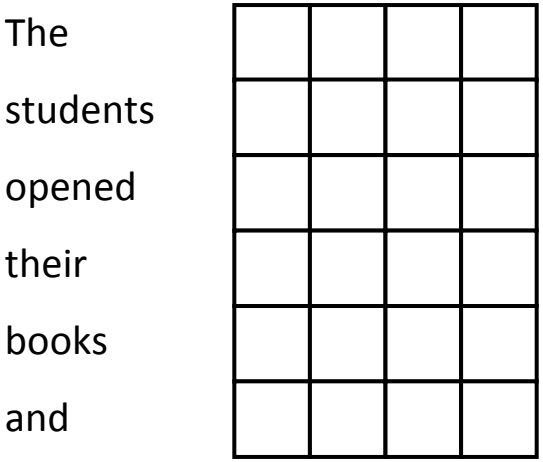
Architecture

- Input Layer
- Convolutional Layer
- Max-pooling Layer
- Non-linear Layer



Input Layer

- Transform words into input representations $\mathbf{x}$ via word embeddings
- $\mathbf{x} \in \mathrm{R}^{m \times d}$: input representation
  - m is the length of sentence
  - d is the dimension of word embeddings



Convolution Layer

- Extract feature representation from input representation via a sliding convolving filter
  - $\mathbf{x} \in \mathrm{R}^{m \times d}$: input representation
  - $\mathbf{x}_{i:i+j} \in \mathrm{R}^{(j+1)d}$: (j+1)-gram representation, concatenation of $\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+j}$
  - $\mathbf{w} \in \mathrm{R}^{h \times d}$: convolving filter, b is a bias term (h is window size)
  - $\mathbf{f} \in \mathrm{R}^{n-h+1}$: convolved feature representation

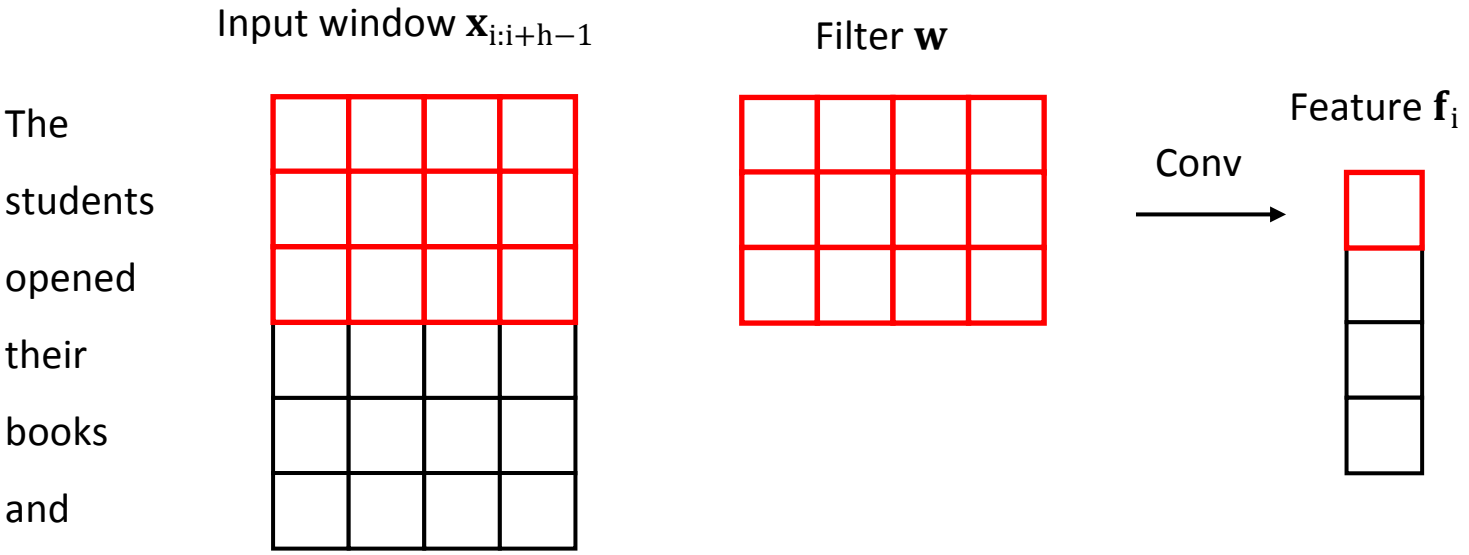
    $$\mathbf{f}_\mathrm{i} = \mathbf{w} \cdot \mathbf{x}_{\mathrm{i:i+h-1}} + \mathrm{b}, \quad \mathrm{i} = 1,2, \ldots, \mathrm{n} - \mathrm{h} + 1$$

  - $\cdot$ is dot product

Convolution Layer

- Extract feature representation from input representation via a sliding convolving filter

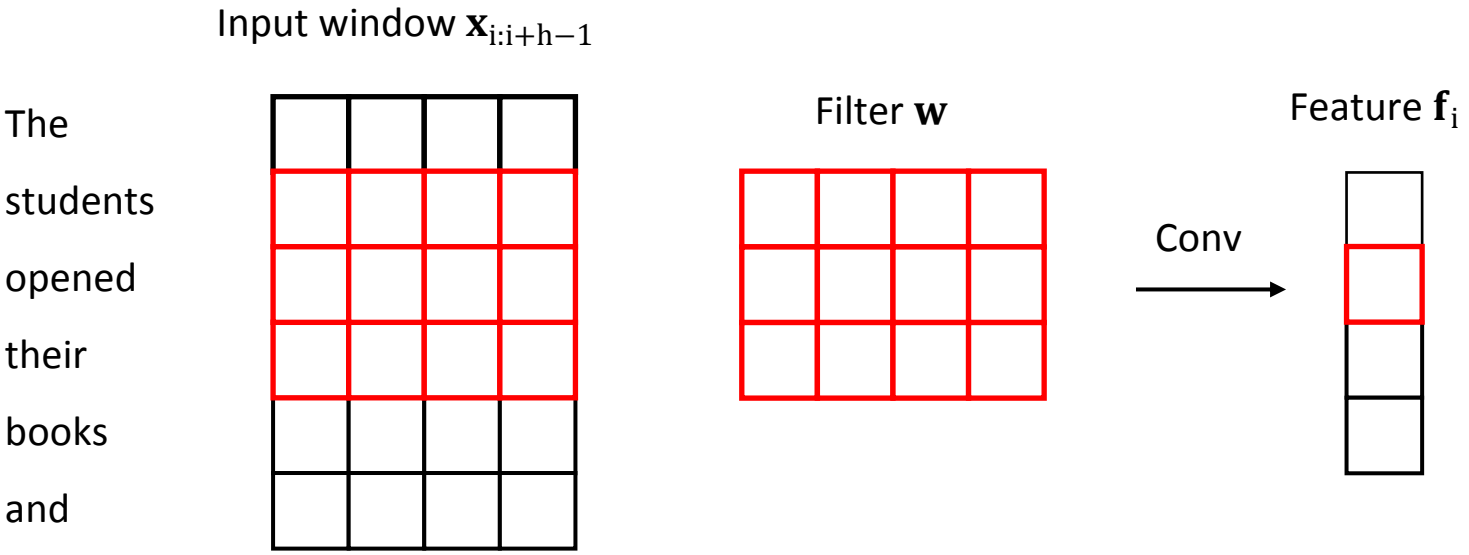
$$\mathbf{f}_i = \mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b, \quad i = 1,2, \ldots, n - h + 1$$



Convolution Layer

- Extract feature representation from input representation via a sliding convolving filter

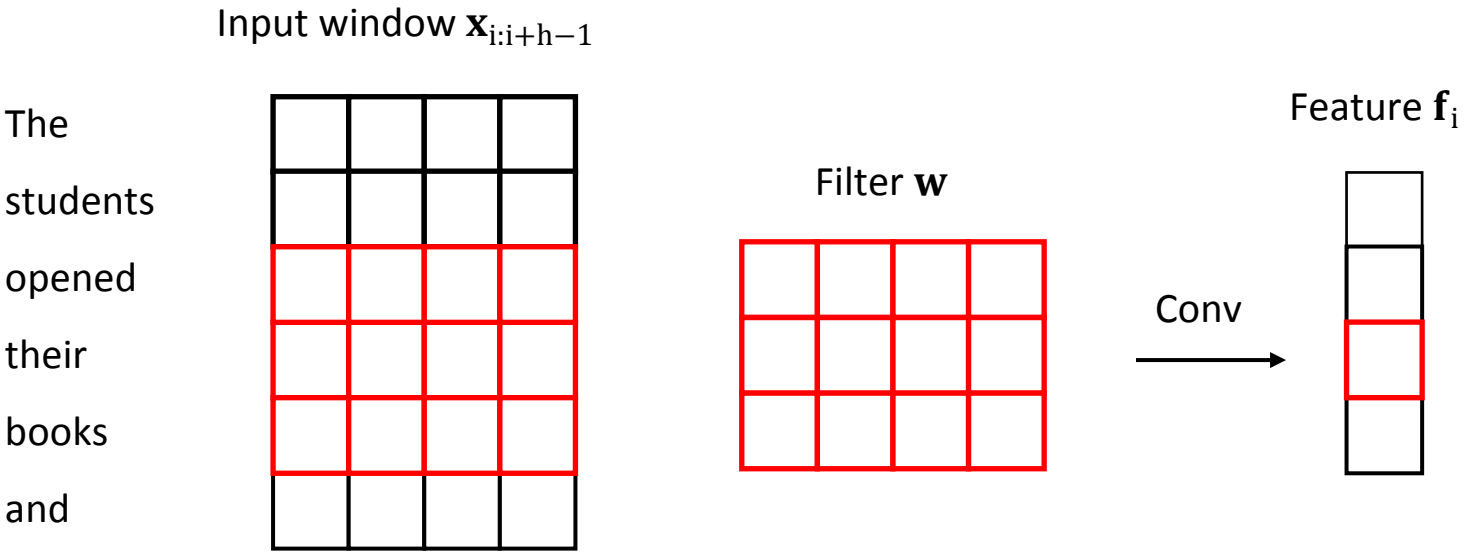
$$\mathbf{f}_i = \mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b, \quad i = 1,2, \ldots, n - h + 1$$



Convolution Layer

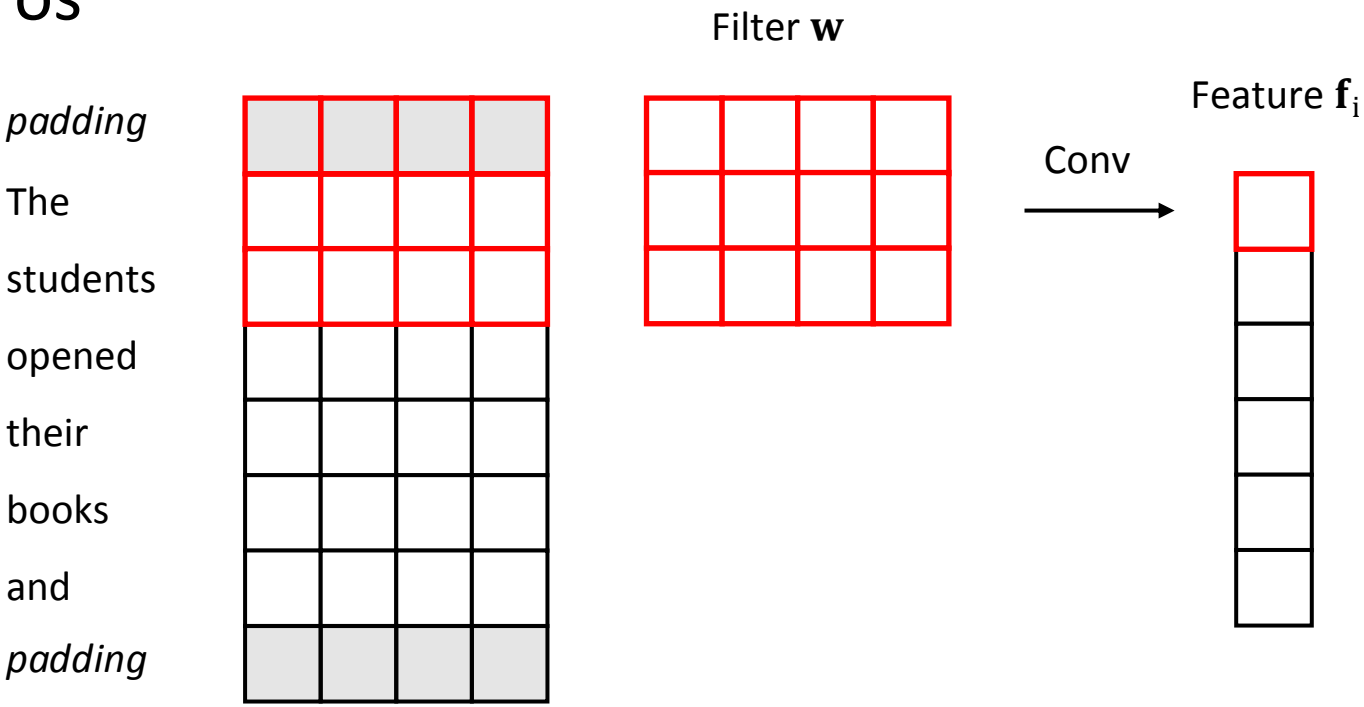
- Extract feature representation from input representation via a sliding convolving filter

$$\mathbf{f}_i = \mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b, \quad i = 1,2, \ldots, n - h + 1$$



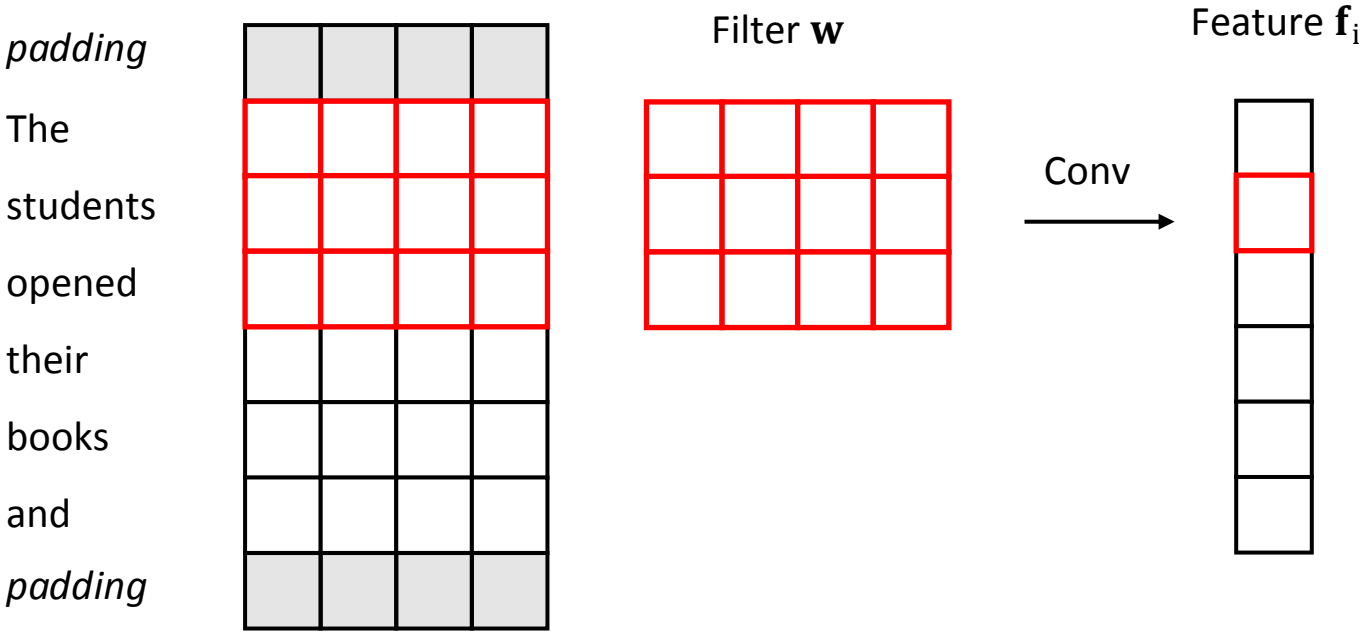
Convolution Layer (with padding)

- Padding is an operation that extends the border of the sentence before convolution, to keep the shape of convoluted feature same as input
- For filter $\mathbf{w} \in \mathrm{R}^{hd}$, padding extends the border with zeros



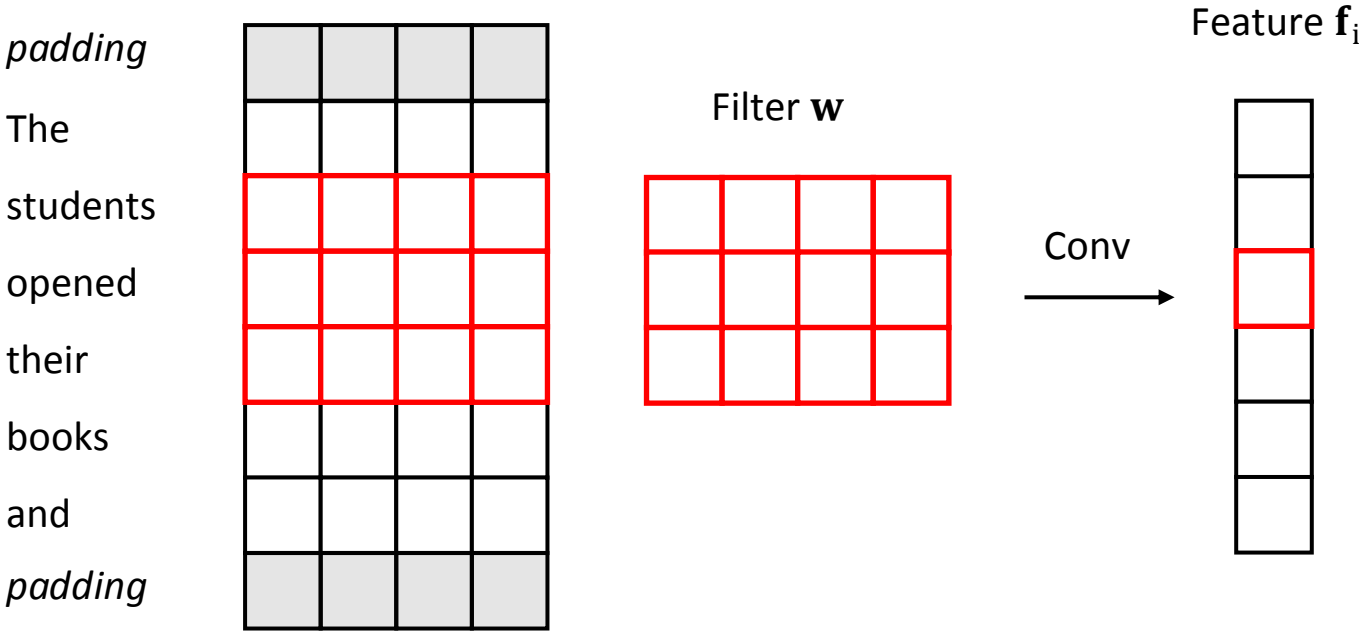
Convolution Layer (with padding)

- Padding is an operation that extends the border of the sentence before convolution, to keep the shape of convoluted feature same as input
- For filter $\mathbf{w} \in \mathrm{R}^{hd}$, padding extends the border with zeros



Convolution Layer (with padding)

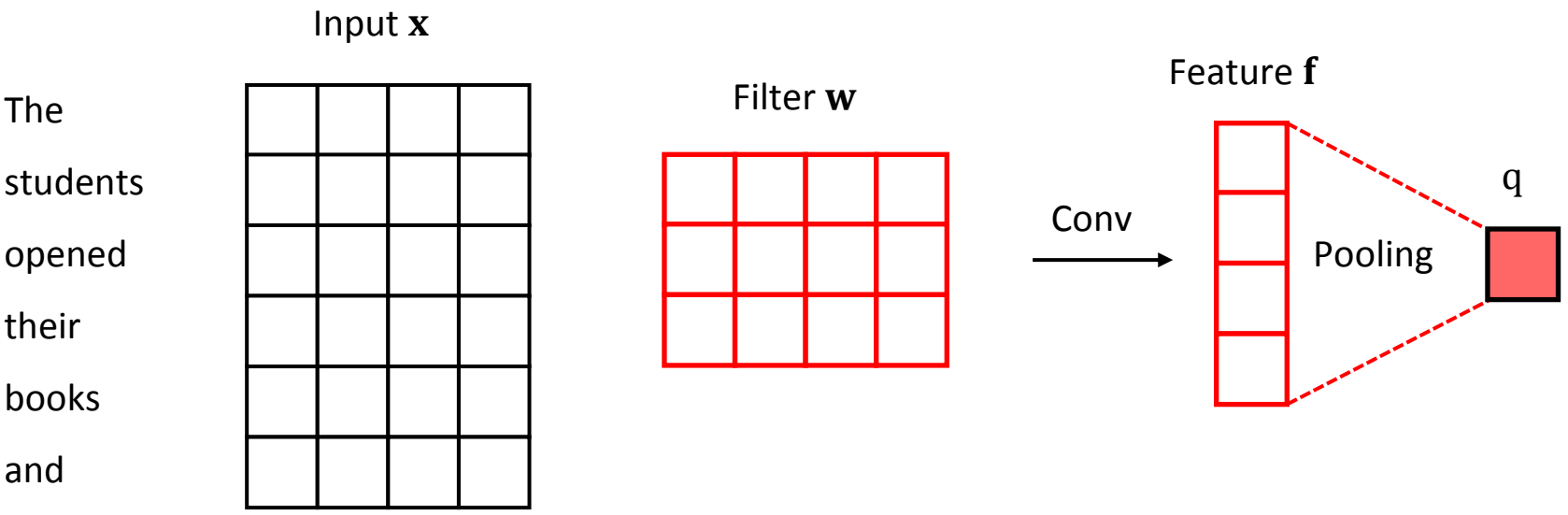
- Padding is an operation that extends the border of the sentence before convolution, to keep the shape of convoluted feature same as input
- For filter $\mathbf{w} \in \mathrm{R}^{hd}$, padding extends the border with zeros



Max-Pooling Layer

- Max-pooling:
  - Extract important features

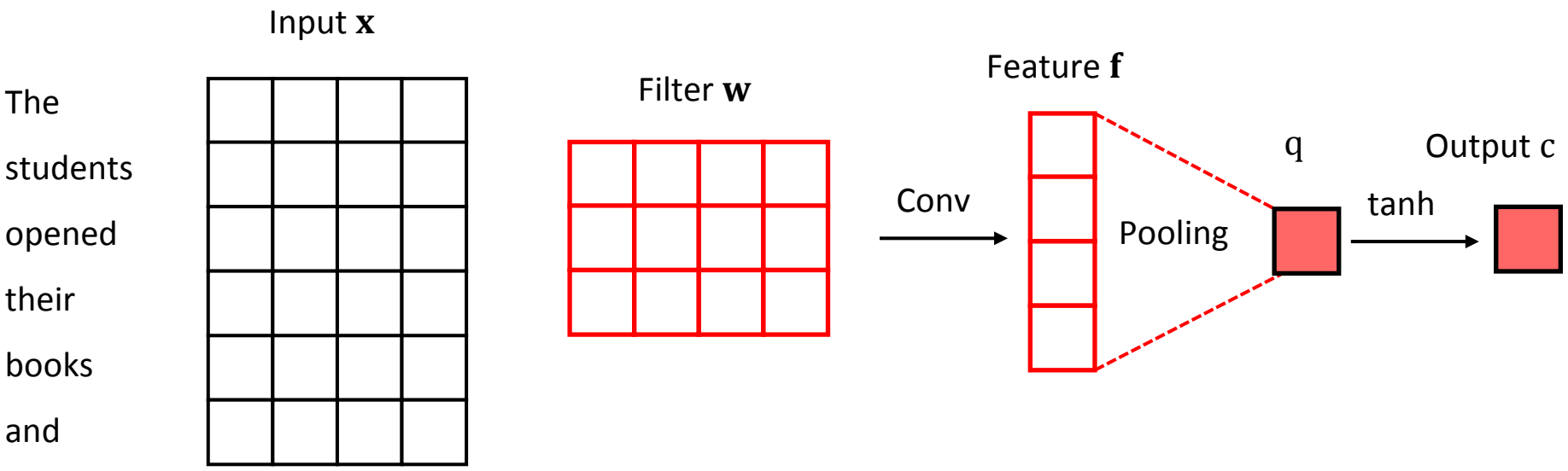
$$q = \max(\mathbf{f})$$



Non-Linear Layer

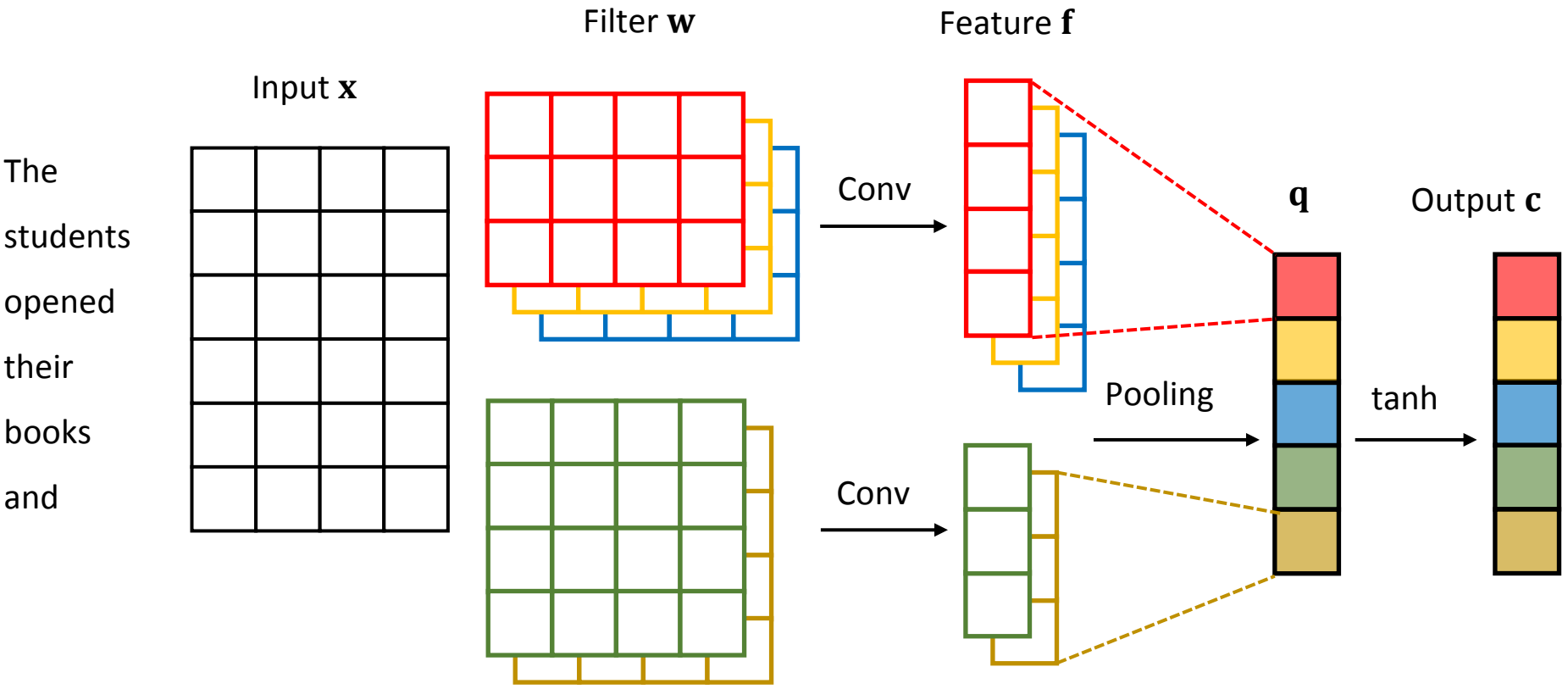
- Non-Linear Activation Function:

$$c = \tanh(q) = \frac{e^{q} - e^{-q}}{e^{q} + e^{-q}}$$



Convolution with multiple filters

- Extract feature representation via multiple filters to capture different n-gram patterns



Compare CNN with RNN

- CNN vs. RNN

	CNNs	RNNs
Advantages	Extracting local and position-invariant features	Modeling long-range context dependency
Parameters	Less parameters	More parameters
Parallelization	Better parallelization within sentences	Cannot be parallelized within sentences

Summary

- Convolutional Neural Network
  - Architecture
    - Input layer
    - Convolution layer
    - Max-pooling layer
    - Non-linear layer
  - Extract local features
  - Capture different n-gram patterns

Class Summary

- Word Representation
  - Synonym, One-hot, Count-based, Distributed
- Neural Network
  - Backpropagation
- RNN & CNN
  - Sentence modeling

Reading Material

a. Word Representation

· Linguistic Regularities in Continuous Space Word Representations. Tomas Mikolov, Wen-tau Yih and Geoffrey Zweig. NAACL 2013. [link]

· Glove: Global Vectors for Word Representation. Jeffrey Pennington, Richard Socher and Christopher D. Manning. EMNLP 2014. [link]

· Deep Contextualized Word Representations. Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer. NAACL 2018. [link]

b. RNN & CNN

· ImageNet Classification with Deep Convolutional Neural Networks. NIPS 2012 [link]

· Convolutional Neural Networks for Sentence Classification. EMNLP 2014 [link]

· Long short-term memory. MIT Press 1997 [link]

For reading material recommendation of this course, please refer to our github.


TSINGHUA UNIVERSITY
-1911-


Tsinghua University
NLP
Natural Language Processing

Q&A

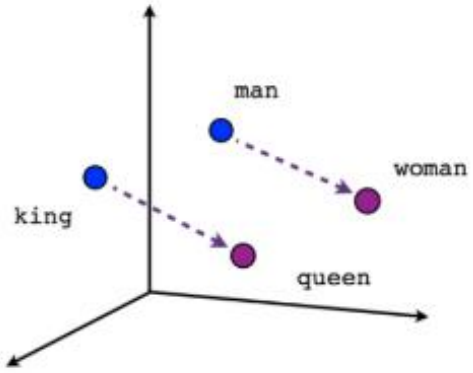
THUNLP

Appendix
Word Embedding

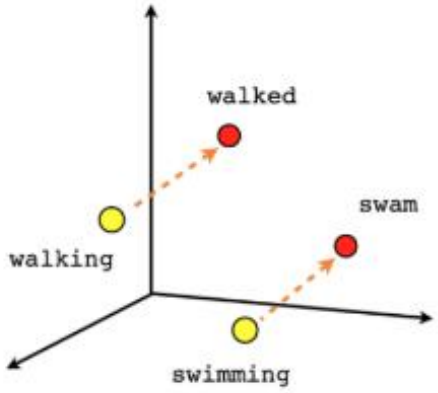
THUNLP

Word2Vec

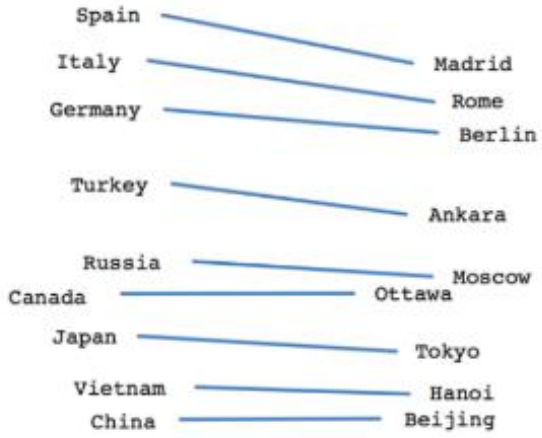
- Word2vec uses shallow neural networks that associate words to distributed representations
- It can capture many linguistic regularities, such as:



Male-Female



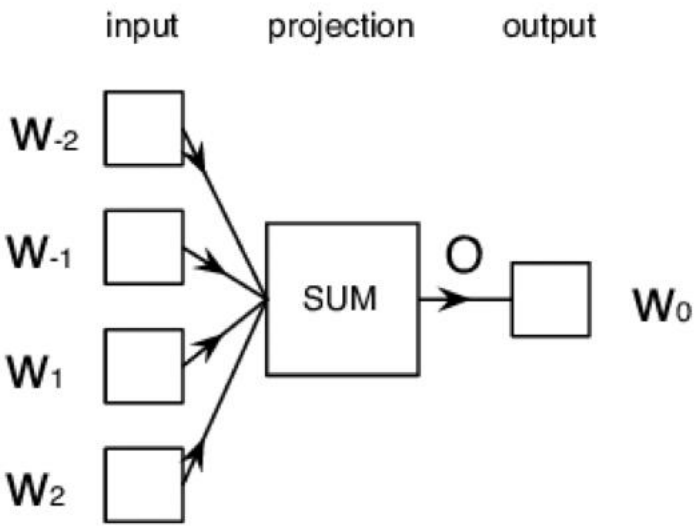
Verb tense



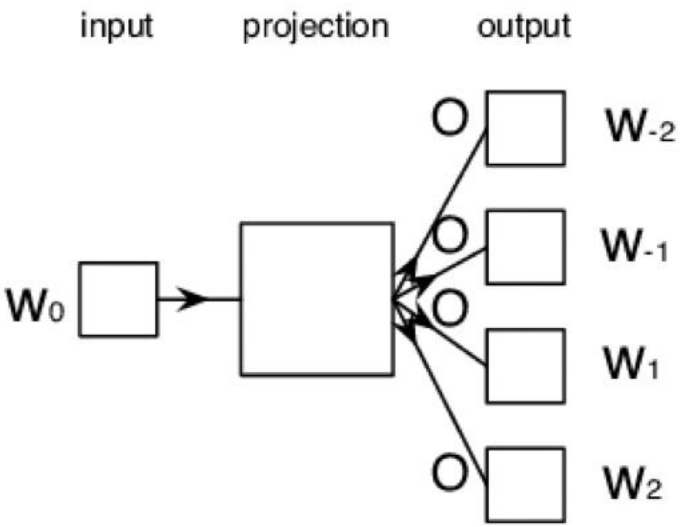
Country-Capital

Typical Models

- Word2vec can utilize two architectures to produce distributed representations of words:
  - Continuous bag-of-words (CBOW)
  - Continuous skip-gram



CBOW

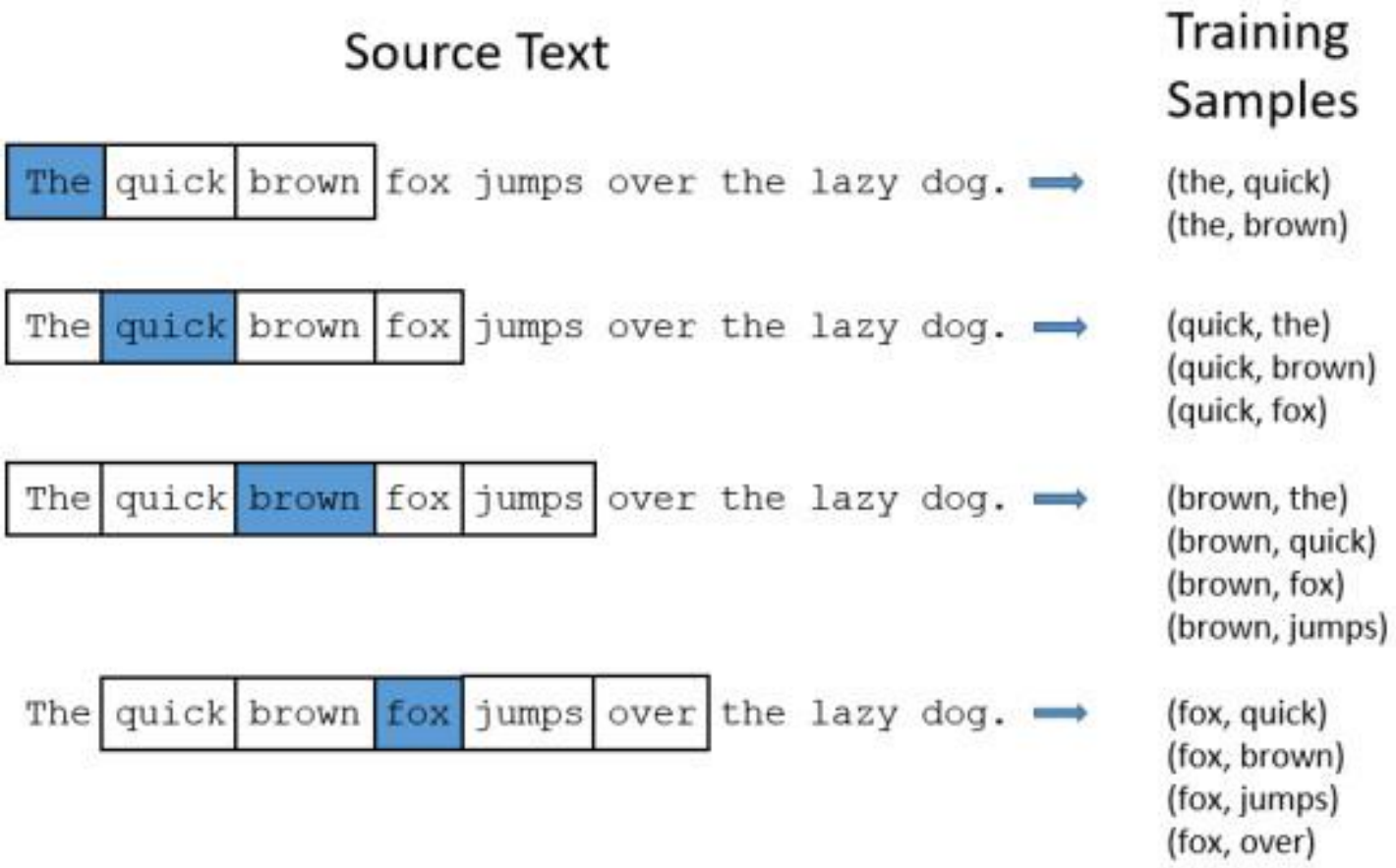


Skip-Gram

Sliding Window

- Word2vec uses a sliding window of a fixed size moving along a sentence
- In each window, the middle word is the target word, other words are the context words
  - Given the context words, CBOW predicts the probabilities of the target word
  - While given a target word, skip-gram predicts the probabilities of the context words

An Example of the Sliding Window



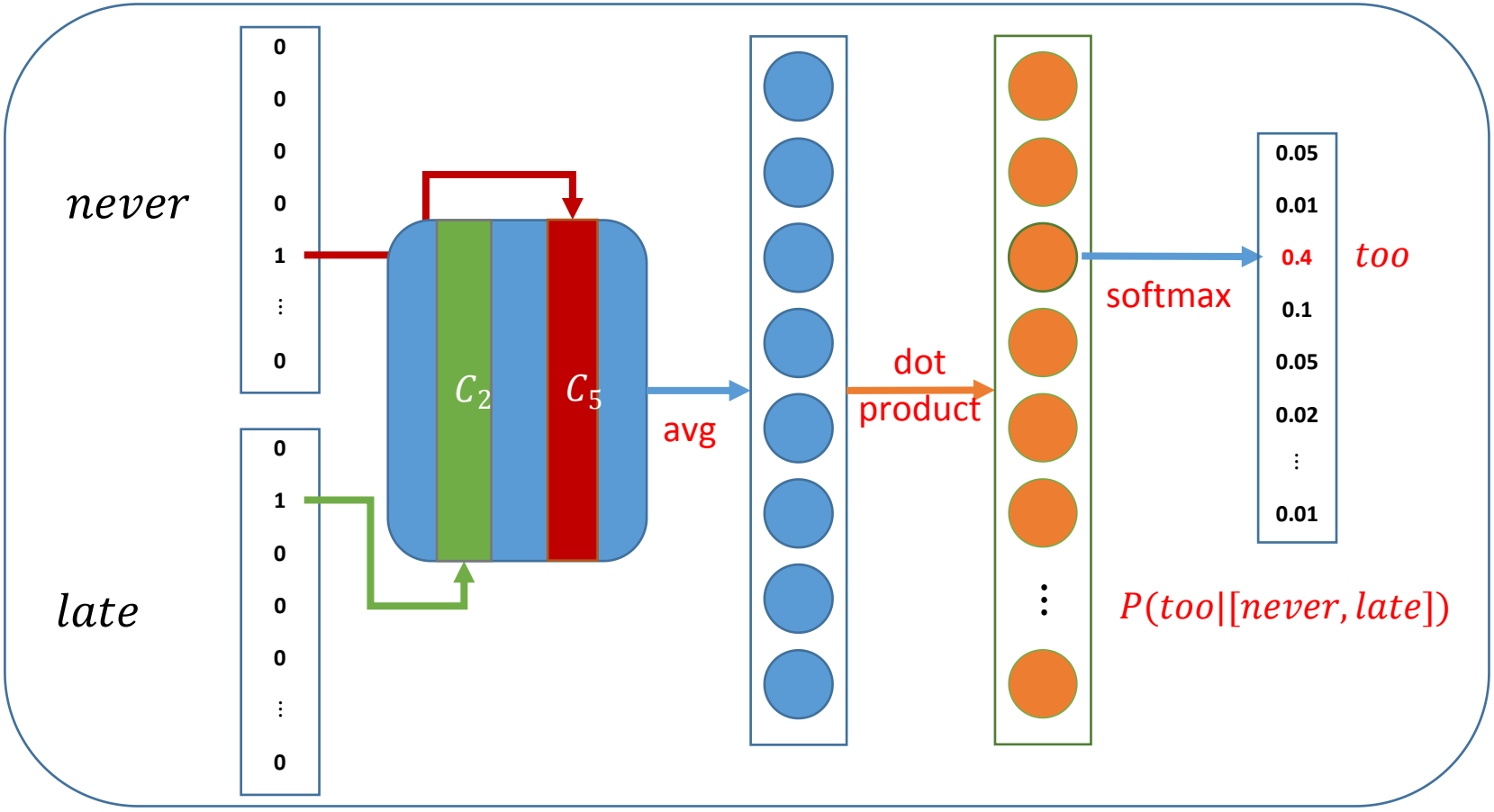
Sliding window size = 5

Continuous Bag-of-Words

- In CBOW architecture, the model predicts the target word given a window of surrounding context words
- According to the bag-of-word assumption: The order of context words does not influence the prediction
  - Suppose the window size is 5
    - $Never\ too\ late\ to\ learn$
    - $P(late|[never,\ too,\ to, learn])$ …

Continuous Bag-of-Words

- *Never too late to learn*



Problems of CBOW

- Consider these two sequences:
  - ***Never too late to learn***
  - ***Never too late to teach***

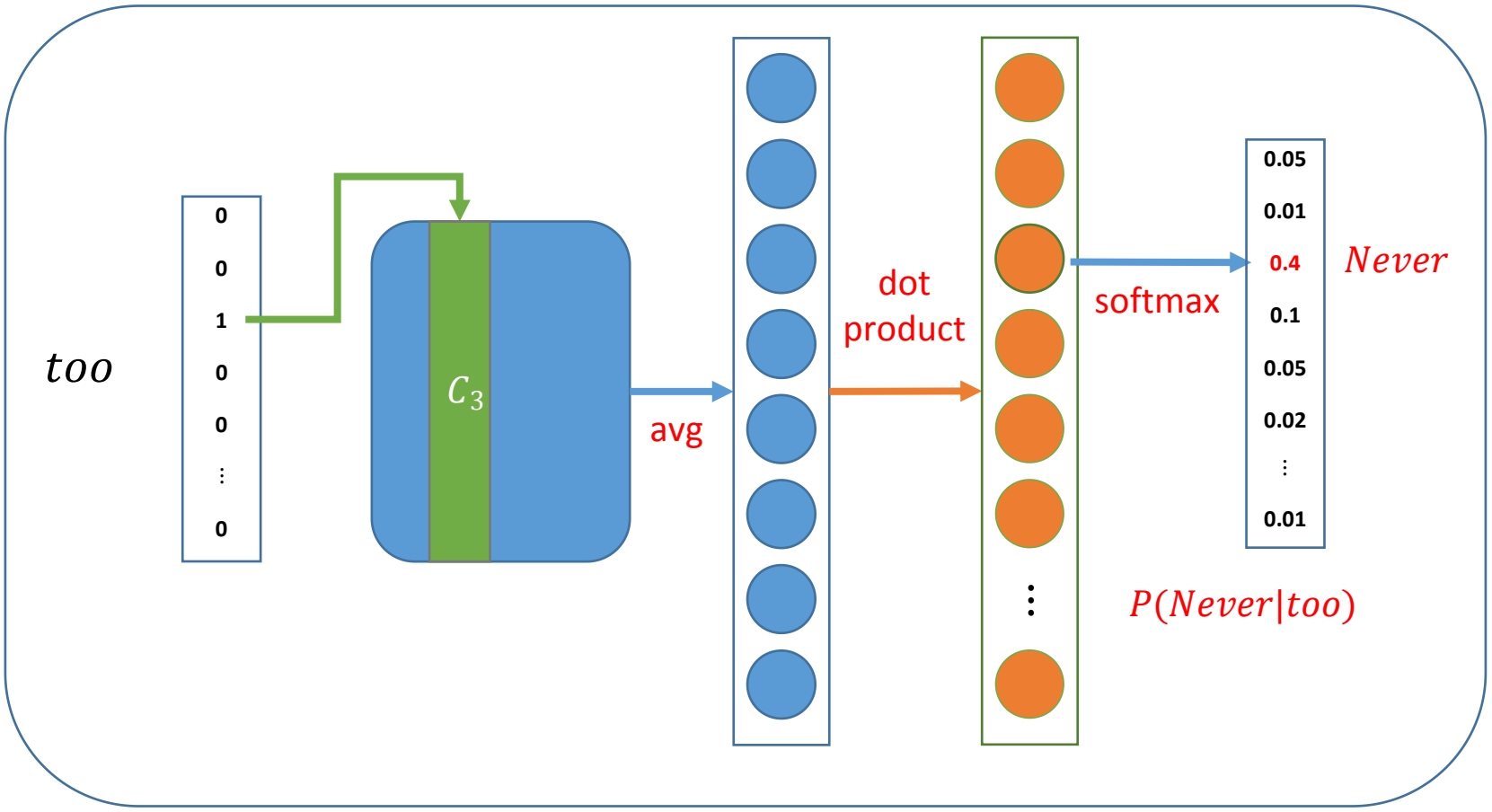
  (The context words of ***"learn"*** and ***"teach"*** are exactly the same. So they tend to have similar representations.)

Continuous Skip-Gram

- In skip-gram architecture, the model predicts the context words from the target word
  - Suppose the window size is 5
    - $Never\ too\ late\ to\ learn$
    - $P([too, late]|Never)$, $P([Never, late, to]|too)$, …
  - Skip-gram predict one context word each step, and the training samples are:
    - $P(too|Never)$, $P(late|Never)$, $P(Never|too)$, $P(late|too)$, $P(to|too)$, …

Continuous Skip-Gram

- *Never too late to learn*



CBOW and Skip-Gram

- We use the softmax function to compute the prediction
- We also use cross-entropy loss function to measure the distribution difference between the prediction and the ground truth
- Optimize the both embedding and map matrices by optimization algorithms

Problems of Full Softmax

- When the vocabulary size is very large
  - Softmax for all the words every step depends on a huge number of model parameters, which is computationally impractical
  - We need to improve the computation efficiency

Improving Computational Efficiency

- In fact, we do not need a full probabilistic model in word2vec
- There are two main improvement methods for word2vec:
  - Negative sampling
  - Hierarchical softmax

Negative Sampling

- As we discussed before, the vocabulary is very large, which means our model has a tremendous number of weights need to be updated every step
- The idea of negative sampling is, to only update a small percentage of the weights every step
- Take skip-gram for example, negative sampling aims to differentiate noisy words from the context words given the target word, focuses on learning high-quality word embedding

Negative Sampling

- Since we have the vocabulary and know the context words, we can select a couple of words not in the context word list by probability:

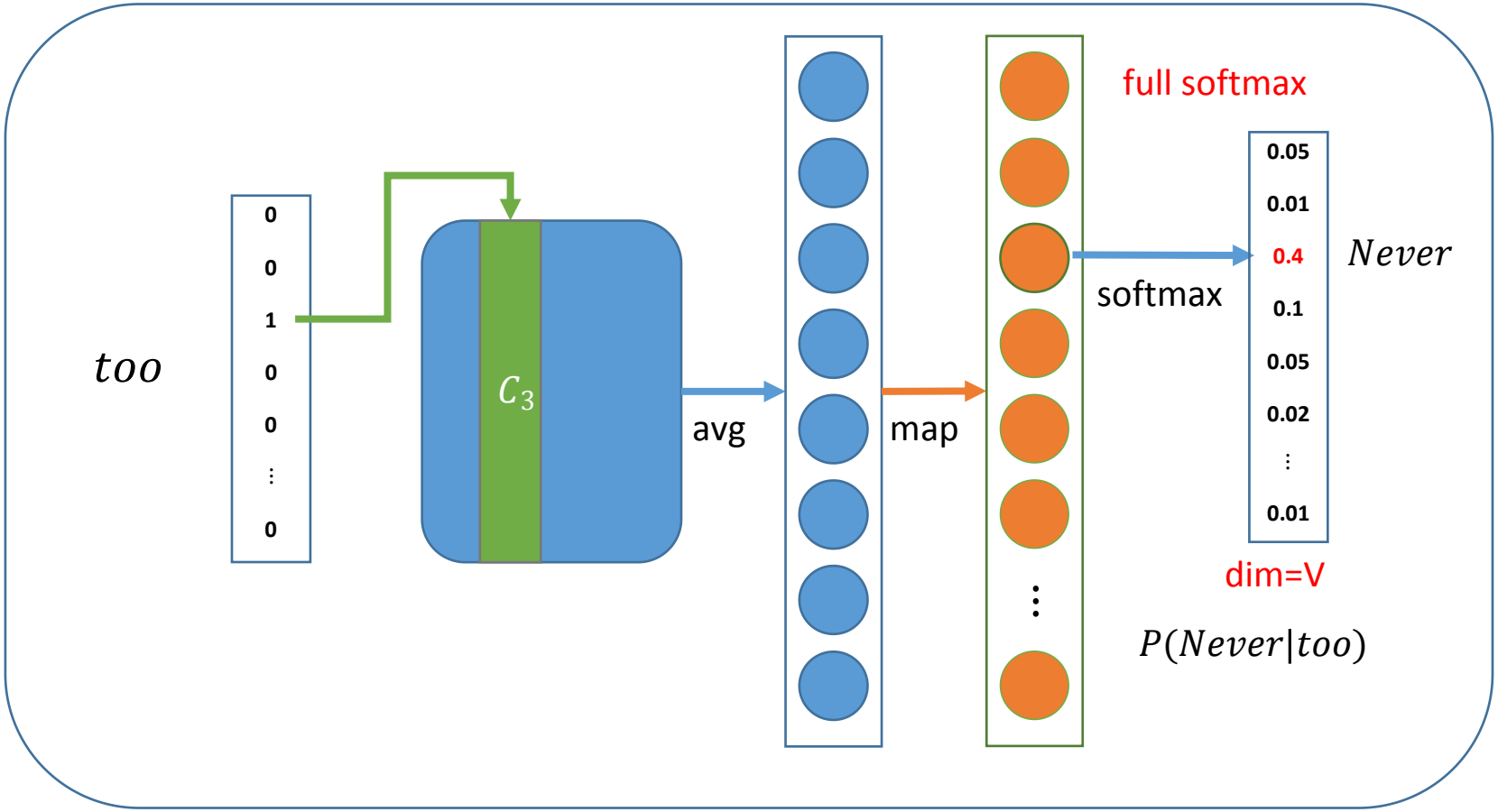
$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{V} f(w_j)^{3/4}}$$

$f(w_i)$ is the frequency of w_i, compared to $\frac{f(w_i)}{\sum_{j=1}^{V} f(w_j)}$, this can increase the probability of low-frequency words.

- If word w_i is a context word of w_j, then the representation of w_j should be more similar to that of w_i than others

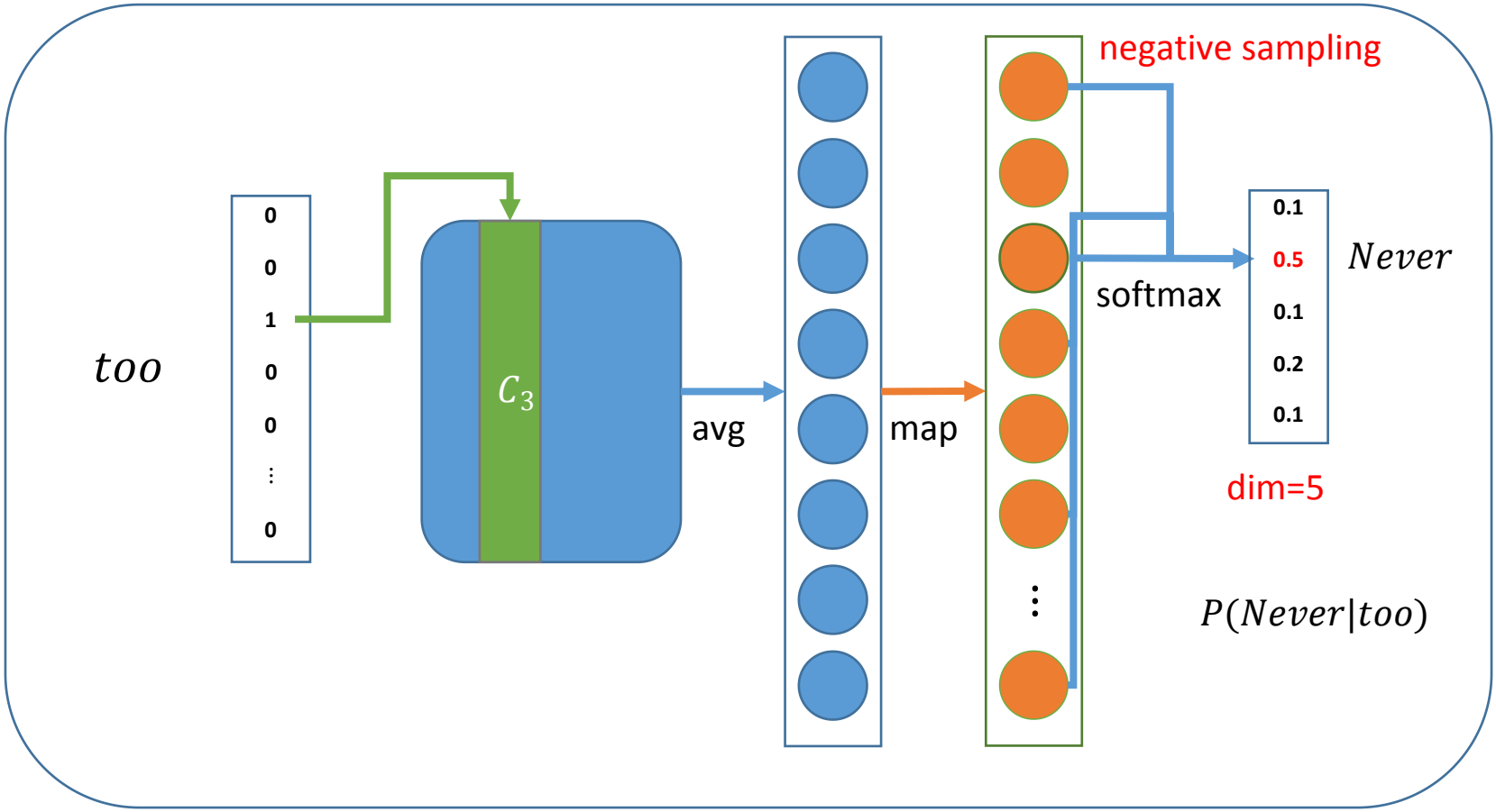
Negative Sampling

- Here is one training step of skip-gram model which computes all probabilities at the output layer



Negative Sampling

- Suppose we only sample 4 negative words:

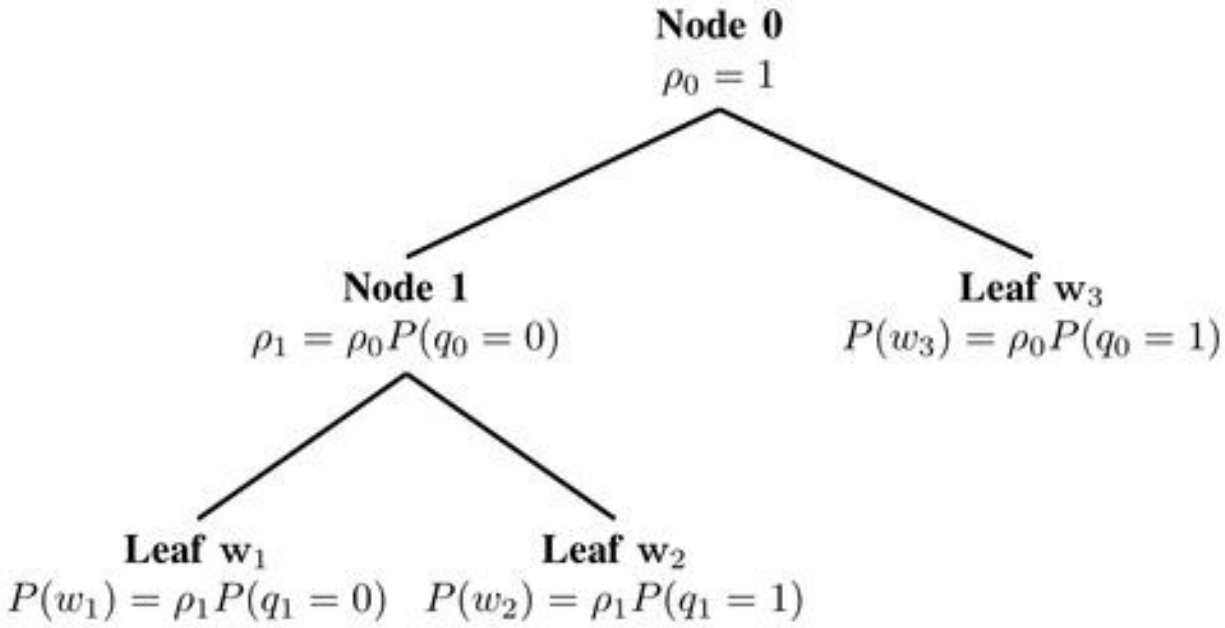


Negative Sampling

- Then we can compute the loss, and optimize the weights (not all of the weights) every step
- Suppose we have a weight matrix of size $300 \times 10{,}000$, the output size is 5
- We only need to update 300×5 weights, that is only 0.05% of all the weights

Hierarchical Softmax

- The hierarchical softmax groups words by frequency with a Huffman tree, reduces the computation complexity of each step from V to $\log V$
- The probability of one word can be compute as follows:



Other Tips for Learning Word Embeddings

- **Sub-Sampling.** Rare words can be more likely to carry distinct information, according to which, sub-sampling discards words w with probability:

$$1 - \sqrt{t/f(w)}$$

$f(w)$ is the word frequency and t is an adjustable threshold.

Other Tips for Learning Word Embeddings

- **Soft sliding window.** Sliding window should assign less weight to more distant words
- Define the max size of the sliding window as S_{max}, the actual window size is randomly sampled between 1 and S_{max} for every training sample
- Thus, those words near the target word are more likely to be in the window

Another Word Vectors Model: GloVe

- The Global Vectors (GloVe) model is an unsupervised learning algorithm for obtaining vector representations for words
- GloVe aims to combine the count-based matrix factorization and the context-based skip-gram model together
- The main intuition of GloVe is that the ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning

Jeffrey Pennington. GloVe: Global Vectors for Word Representation. EMNLP 2014.

Evaluation: Word Analogy

- Google analogy test dataset
  - A proportional analogy holds between two word pairs: *a*:*a** :: *b*:*b** (*a* is to *a** as *b* is to *b**) For example, *Tokyo* is to *Japan* as *Paris* is to *France*
  - 14 relations (9 morphological and 5 semantic)
    - capital; currency; opposite …

Word1 (a)	Word2 (a*)	Word3 (b)	Word4 (b*)
Athens	Greece	Beijing	
write	writes	search	
acceptable	unacceptable	aware	

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In Proceedings of International Conference on Learning Representations (ICLR).

Evaluation: Word Similarity

- WordSimilarity-353 (WordSim353)
  - English word pairs along with human-assigned similarity judgements (total 353 pairs)

Word1	Word2	Human score (mean)
love	sex	6.77
tiger	cat	7.35
stock	egg	1.85

  - Evaluation: Spearman's correlation coefficient
    - The correlation of word pair similarity between model and human annotations

Finkelstein et al. Placing Search in Context: The Concept Revisited, ACM Transactions on Information Systems, 2002

Evaluation: Word Similarity

- WordSimilarity-353 (WordSim353)
  - Spearman's correlation coefficient
    - A nonparametric measure of rank correlation (statistical dependence between the rankings of two variables)
    - Equation: $\frac{\sum_{i=1}^{N}(R_i-\bar{R})(S_i-\bar{S})}{[\sum_{i=1}^{N}(R_i-\bar{R})^2\sum_{i=1}^{N}(S_i-\bar{S})^2]^{\frac{1}{2}}}$
      - Model1:-0.5
      - Model2:1.0

Word1	Word2	Human score (mean)	Model1	Model2
love	sex	6.77	0.5	0.5
tiger	cat	7.35	0.1	0.6
stock	egg	1.85	0.2	0.1

$\bar{R}$=mean(R_1, R_2, R_3) $\bar{S}$=mean(S_1, S_2, S_3)

Finkelstein et al. Placing Search in Context: The Concept Revisited, ACM Transactions on Information Systems, 2002

Co-Occurrence Counts

- Term-Term matrix (co-occurrence matrix)
  - how often a word occurs with another

$$C(i,\ j) = \sum_{\Delta x=-m}^{m} \sum_{x=-\Delta x}^{n-\Delta x-1} \begin{cases} 1,\ if \Delta x \neq 0,\ I(x) = i\ and\ I(x+\Delta x) = j \\ 0,\ otherwise \end{cases}$$

  - $\mathbf{n}$ is the length of sentence $\boldsymbol{I}$, $\mathbf{m}$ is the length of the window
  - Captures both syntactic and semantic information
    - The shorter the windows, the more syntactic the representation ($m \in [1,3]$)
    - The longer the windows, the more semantic the representation ($m \in [4,10]$)

Term-Term Matrix

- Example corpus
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.
- Tokens
  - I
  - Like
  - Deep
  - Learning
  - NLP
  - Enjoy
  - Flying
  - .

Term-Term Matrix

- Example corpus
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.
- Term-Term matrix with window length $m = 2$

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	1	0	1	1	0
like	2	0	0	1	1	1	0	1
enjoy	1	0	0	0	0	0	1	1
deep	1	1	0	0	1	0	0	1
learning	0	1	0	1	0	0	0	1
NLP	1	1	0	0	0	0	0	1
flying	1	0	1	0	0	0	0	1
.	0	1	1	1	1	1	1	0

Term-Document Matrix

- Term-Document matrix
  - how often a word occurs in a document
  - Latent Semantic Analysis can be further used for finding latent topics
- Example corpus
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	D1	D2	D3
I	1	1	1
like	1	1	0
enjoy	0	0	1
deep	1	0	0
learning	1	0	0
NLP	0	1	0
flying	0	0	1
.	1	1	1

Co-Occurrence Counts

- Term-Document matrix
  - Each document is a count vector in $\mathbb{N}^V$ (a column)
  - Each word is a count vector in $\mathbb{N}^D$ (a row)

  (D is the number of documents within the corpus, and V is the vocabulary size.)

counts	D1	D2	D3
I	1	1	1
like	1	1	0
enjoy	0	0	1
deep	1	0	0
learning	1	0	0
NLP	0	1	0
flying	0	0	1
.	1	1	1

- In large corpora, we expect to see:
  - Two words are similar if their vectors are similar
  - Two documents are similar if their vectors are similar

Similarity Measurement

- Given 2 word vectors v and w, we can use the dot product to measure the similarity

$$(v,\ w) = v \cdot w = \sum_{i=1}^{D} v_i \cdot w_i = v_1 w_1 + v_2 w_2 + \ldots + v_D w_D$$

  - Dot product is larger if the vector is longer
  - The length of a vector represents the word's frequency

$$length(v) = |v| = \sqrt{\sum_{i=1}^{D} v_i^2}$$

  - Longer length means the word occurs more frequent, which will have larger dot products

Similarity Measurement

- We can normalize the similarity using vector's length
- In practice, we often use cosine similarity

$$cosine\_similarity(v, w) = \frac{v \cdot w}{|v||w|}$$

- This is the cosine of the angle θ between the v and w, since

$$v \cdot w = |v||w| \cos\theta$$

$$\cos\theta = \frac{v \cdot w}{|v||w|} = cosine\_similarity(v, w)$$

Dense Vectors

- Short vectors are easier to use as features in machine learning
- Dense vectors have a better generalization ability

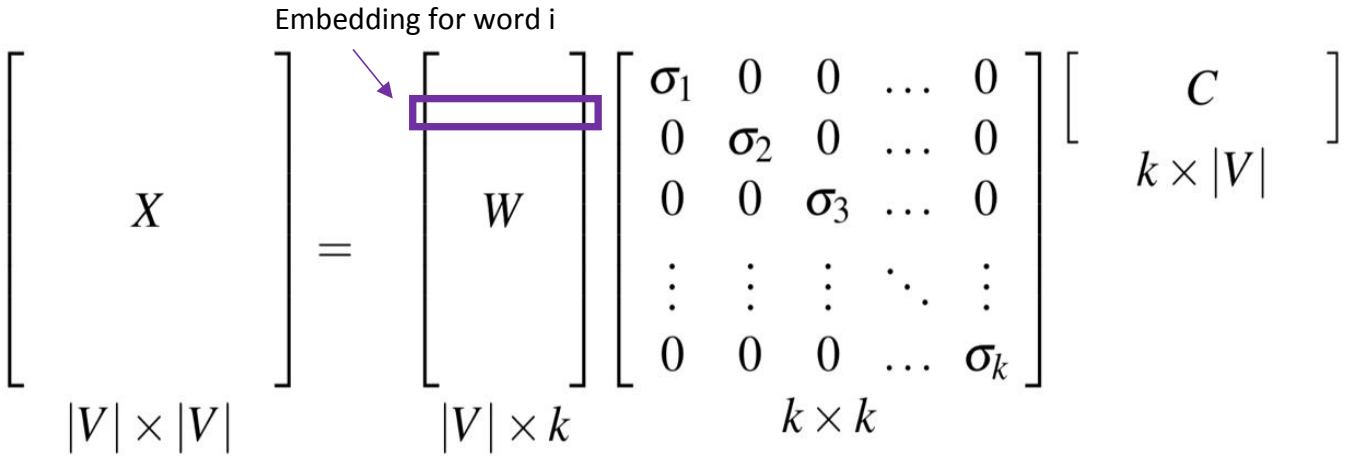
Singular Value Decomposition

- Singular Value Decomposition (SVD)
  - M is a $m \times n$ matrix
  - M can be represented as $M = U\Sigma V^*$
    - U is a $m \times k$ matrix, rows corresponding to original rows, but k columns represents a dimension in a new latent space, such that k column vectors are orthogonal to each other
    - Σ is a $k \times k$ diagonal matrix of singular values expressing the importance of each dimension
    - V^* is a $k \times n$ matrix, columns corresponding to original columns, but k rows corresponding to the singular values
  - If k is not small enough, we can keep the top-k singular values (like 300) to obtain a least-squares approximation to M
  - Reduce word representation dimension

More refer to https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/book-chapter-4.pdf

Dense Vectors

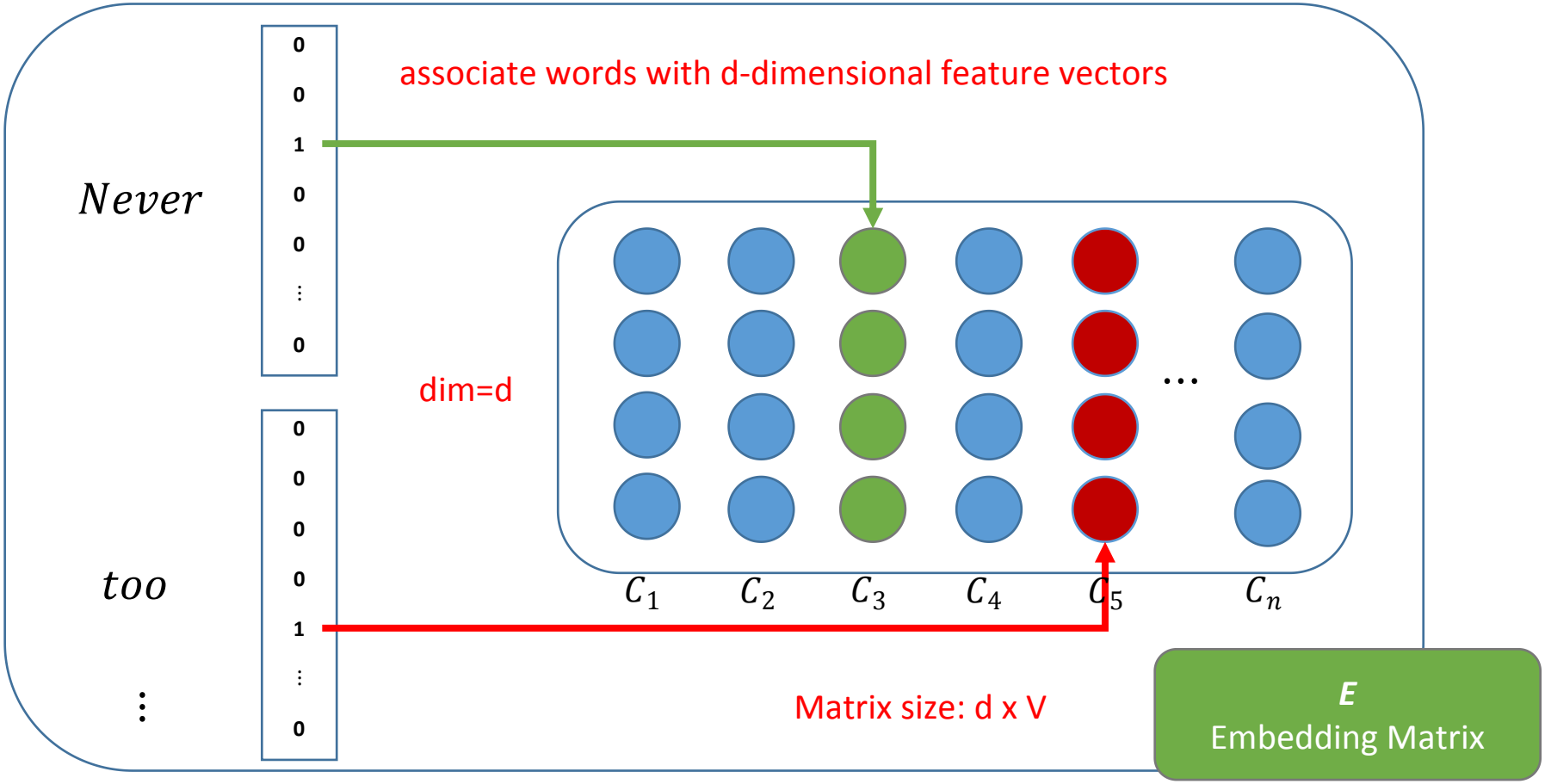
- SVD can be applied to term-term matrix (co-occurrence matrix) to create dense vectors



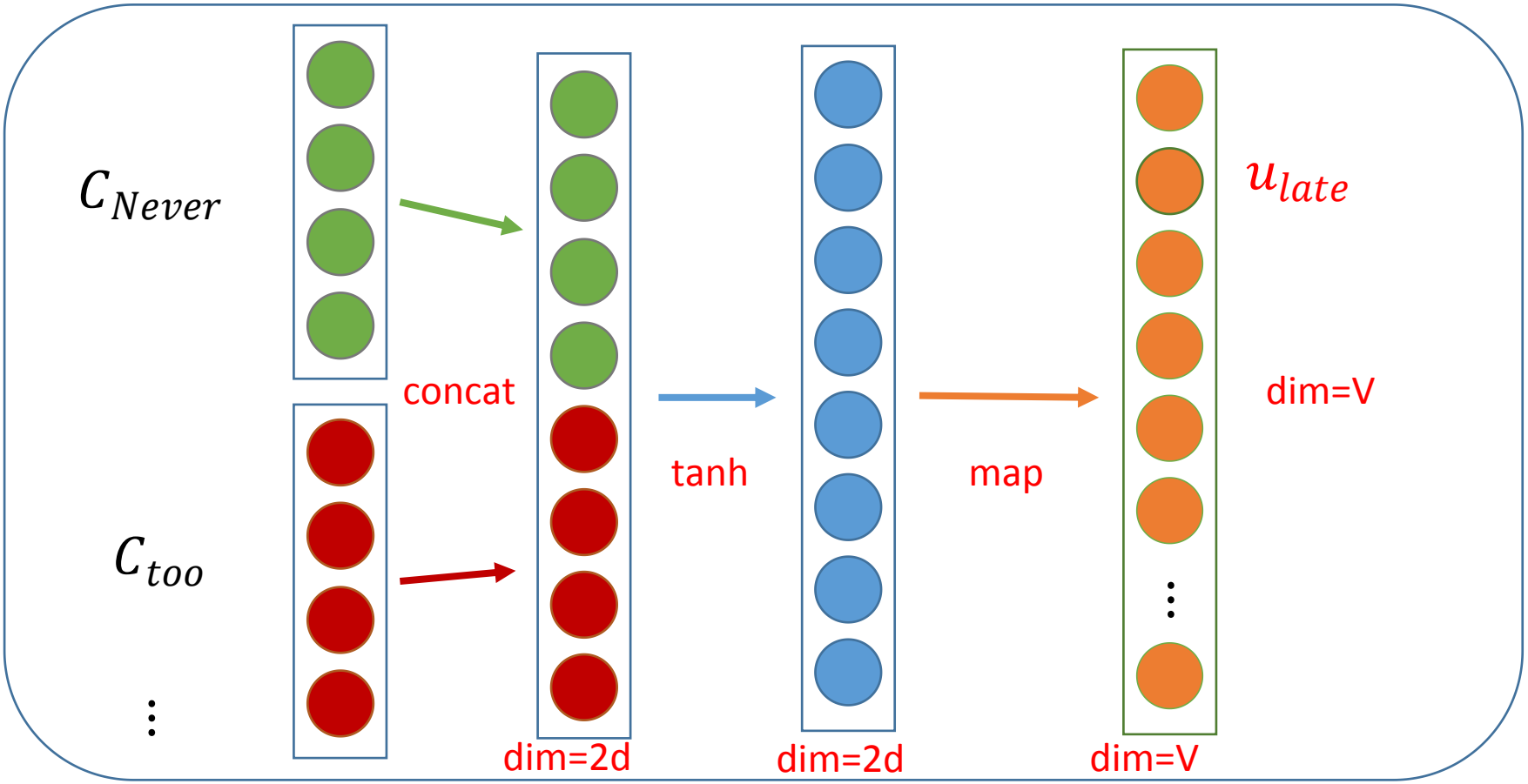
- X is th … sional representation of each word w

Lookup Table

- *Never too late to learn*



Probability of Next word

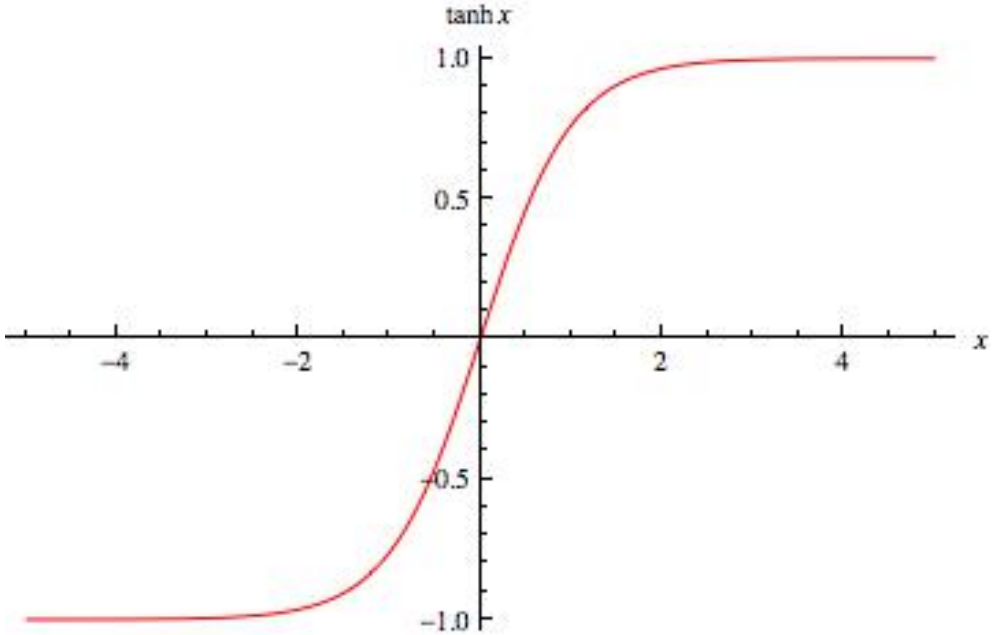


Hyperbolic Tangent Function

- The Hyperbolic Tangent Function (tanh)

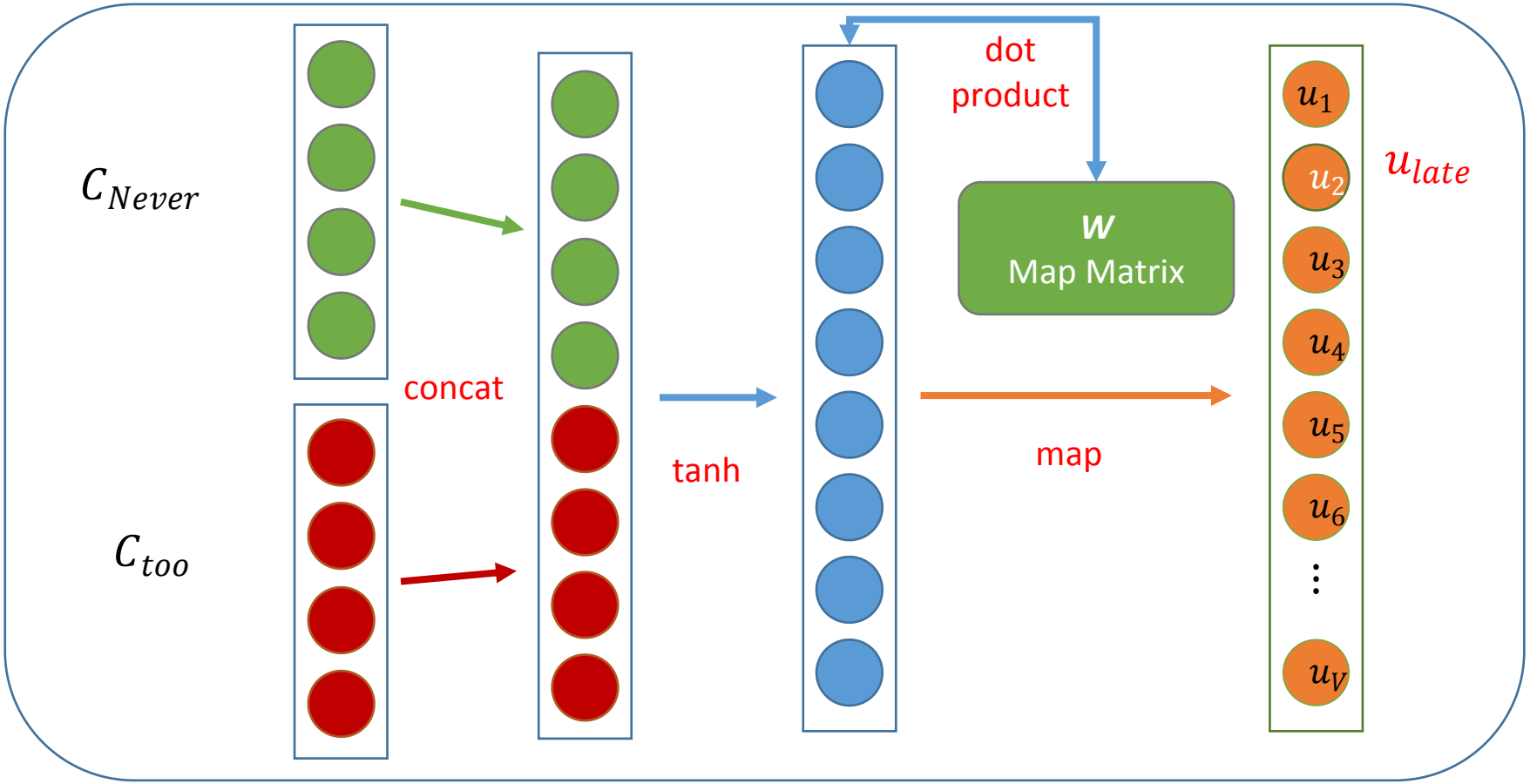
$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

  - It is a non-linear function, and the value is between $[-1,1]$. An intuitive view:

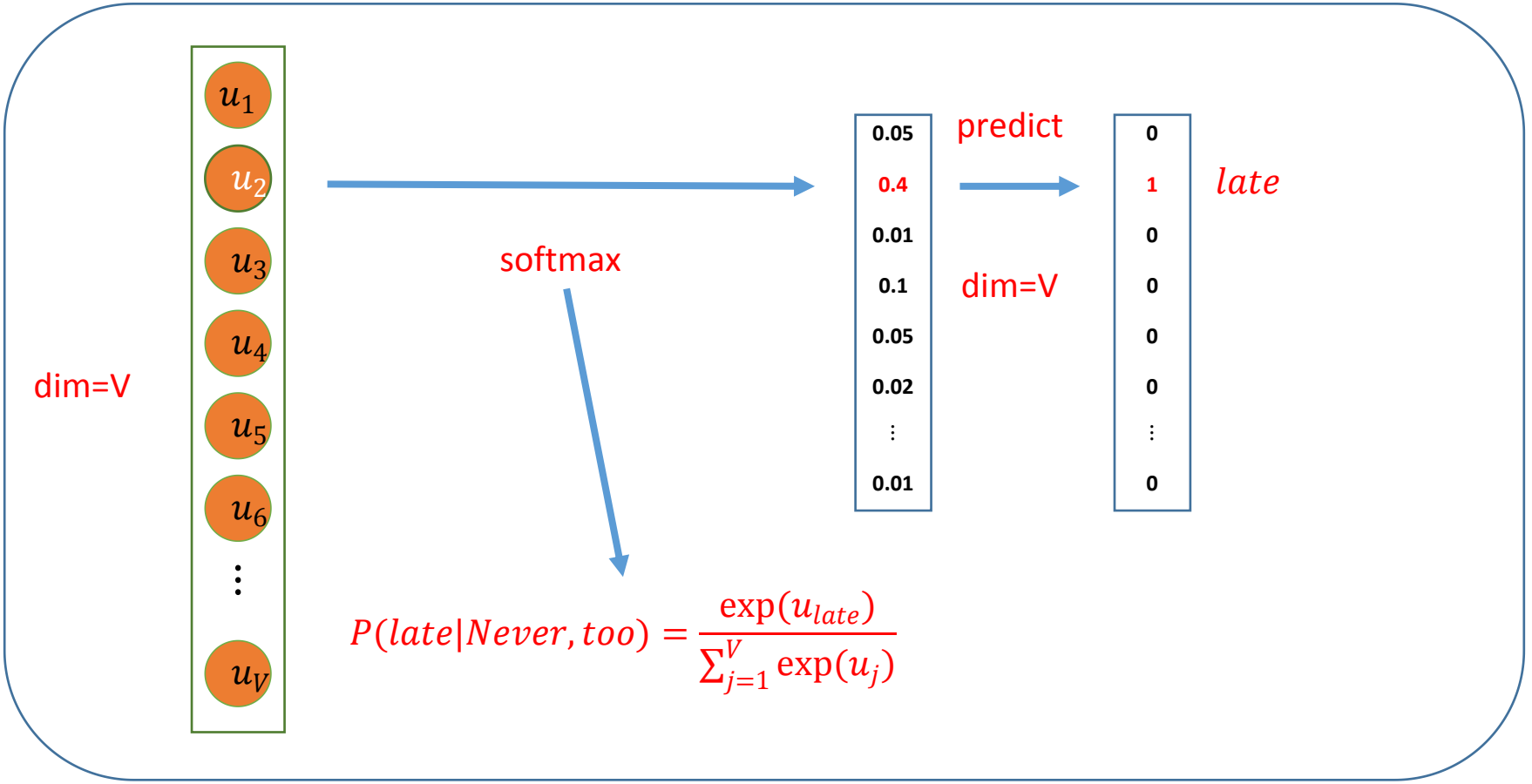


Probability of the Next Word

$$u_i = \sum_j W_{ij} I_j + b_j$$



Softmax and Prediction



Similarity Measurement

- First, we should measure the difference between the prediction and the ground truth:

$$Loss(late, \hat{late}) = -(1 \times \log 0.4)$$

$$Loss(w_i, \hat{w}_i) = -y_i \log \hat{y}_i$$

  - y_i is the i-th ground truth, $\hat{y}_i$ is the i-th prediction

The Cross-Entropy Loss Function

- Then we sum these up:

$$Loss(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$$

  - This is the cross-entropy loss function
  - When $y_i = 0$, $Loss(y, \hat{y}) = 0$, $y_i = 1$, $Loss(y, \hat{y}) = -\log \hat{y}_i$
  - We need $\hat{y}_i \to 1$ when $y_i = 1$ to reduce the loss

More refer to https://en.wikipedia.org/wiki/Cross_entropy

Back-Propagation Algorithm

- Compute the gradient of the cross-entropy loss function using chain rule

$$\frac{\partial(Loss)}{\partial(W_{ij})} = \frac{\partial(Loss)}{\partial(\hat{y}_i)} \frac{\partial(\hat{y}_i)}{\partial(u_i)} \frac{\partial(u_i)}{\partial(W_{ij})}$$

  - W_{ij} is an item of weight matrix at the i-th row and the j-th column. Finally we have:

$$\frac{\partial(Loss)}{\partial(W_{ij})} = I_j(\hat{y}_i - y_i)$$

    - We can use the optimization algorithms, such as stochastic gradient descent (SGD) to adjust the W_{ij} to reduce the loss
    - With the same manner, we can optimize parameters in every layer of the model

Optimization

- Stochastic gradient descent (SGD) is a stochastic approximation of gradient descent optimization
- Stochastic means samples are selected randomly
- Suppose we learn weights W to minimize the loss $Loss(W)$, the iterative method:

$$W = W - \tau \frac{\partial(Loss)}{\partial(W)}$$

  - τ is a step size (also called learning rate)
  - Optimize the weights iteratively

Example Jacobian: Activation Function

- Given $h = F(z),$ where $h_i = f(z_i), h, z \in \mathbb{R}^n$, what is $\frac{\partial h}{\partial z}$?

$$\frac{\partial h}{\partial z} = \begin{bmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{bmatrix} = \text{diag}(\text{f'}(z))$$

- More examples:

$$\frac{\partial}{\partial x}(Wx + b) = \text{W}$$

$$\frac{\partial}{\partial b}(Wx + b) = \text{I}$$

$$\frac{\partial}{\partial u}(u^T h) = h^T$$

Long Short-Term Memory Network

- Operation of each time step
  - Input gate: $i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$
  - Forget gate: $f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$
  - Output gate: $o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$
  - New memory cell: $\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$
- Final output
  - Final memory cell: $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
  - Final hidden state: $h_t = o_t \circ \tanh(c_t)$

Document Representation Learning

THUNLP

Document Representation Learning

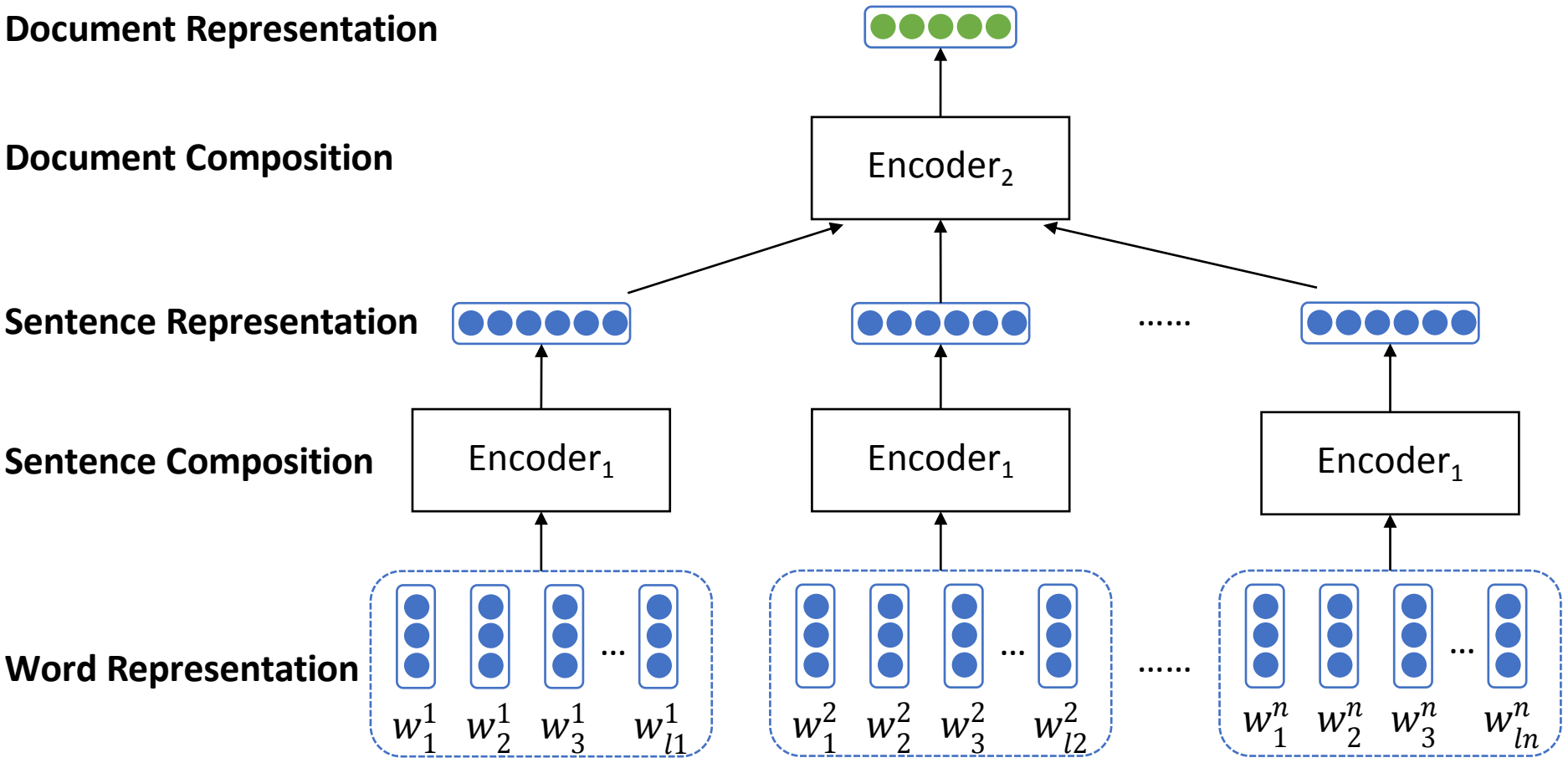
- Document representation learning is quite important in many downstream tasks, e.g., web retrieval, question answering, sentiment analysis.

- In practice, CNN/RNN are not powerful enough to capture enough information from documents which consists of hundreds of words.

Hierarchical Structured Network

- Challenge: encoding the intrinsic relations between sentences into the semantic meaning of documents.
- To address this issue, a hierarchical structured network is proposed.
  - Learn sentence representation
  - Encode semantics of sentences into the document representation

Hierarchical Structured Network



Hierarchical Attention Network

- Hierarchical Attention Network (HAN)
  - HAN incorporates two levels of attention mechanisms into previous hierarchical network.
  - The attention mechanisms enable HAN to attend differentially to more and less important content when constructing the document representation

Attention Mechanism

- Attention mechanism enables models to pay greater attention to certain factors when processing a sequence.

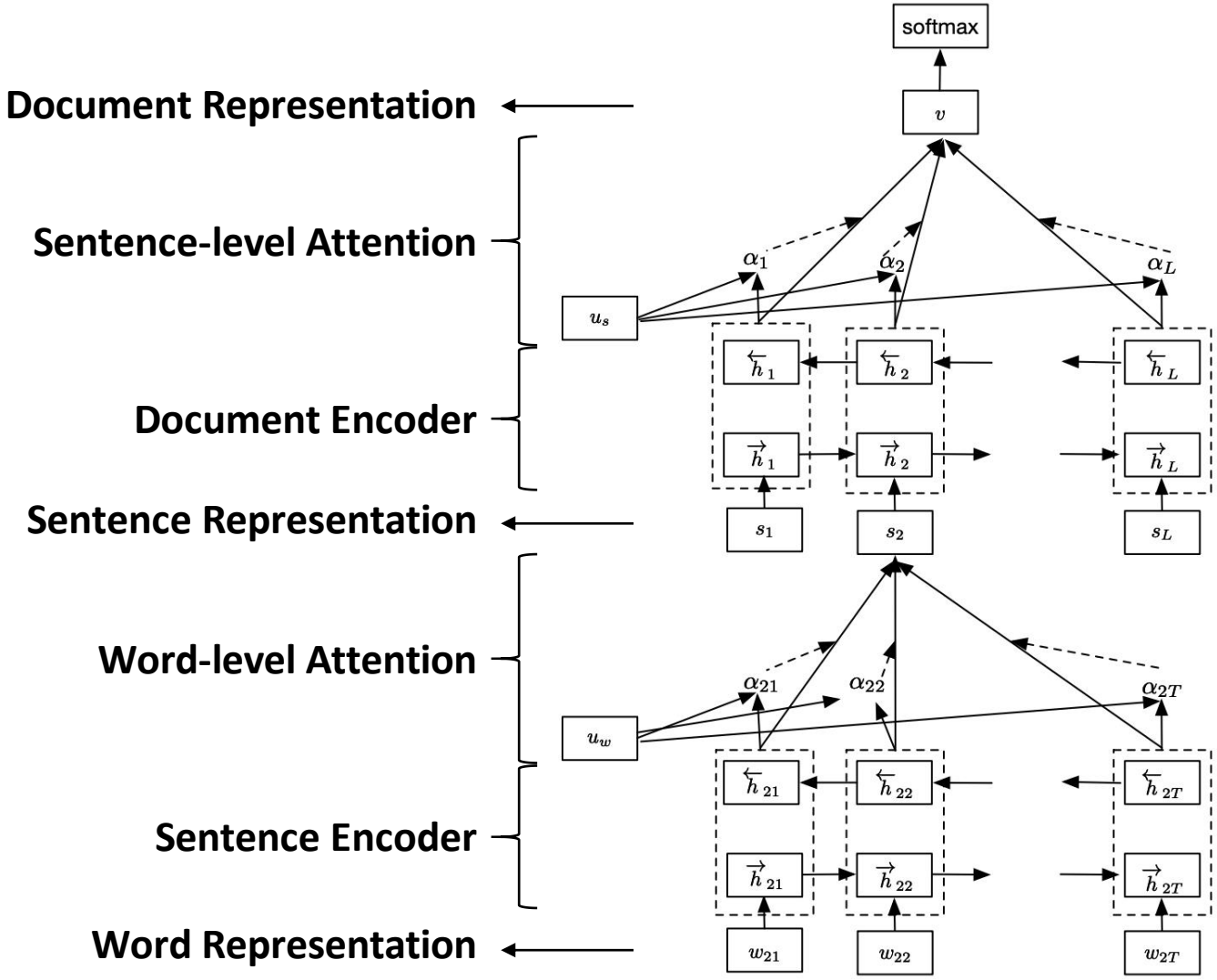
- Given a vector sequence $s = \{v_1, v_2, \ldots, v_n\}$, $v_i \in \mathbb{R}^d$, and a query $u \in \mathbb{R}^d$, we calculate the representation r as follows:

$$a_i = \frac{\exp(v_i \cdot u)}{\sum_j \exp(v_j \cdot u)}$$

$$r = \sum_i a_i v_i$$

Amazing! Pork belly is so delicious! I will go back here.

Hierarchical Attention Network



Thanks

Zhiyuan Liu

liuzy@tsinghua.edu.cn

THUNLP