Neural Network & Seq2Seq Modeling

Zhiyuan Liu

liuzy@tsinghua.edu.cn

THUNLP

Outline

- Neural Network
- Introduction to Seq2Seq
- Machine Translation

Review

- 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 $F_\theta(\cdot)$.

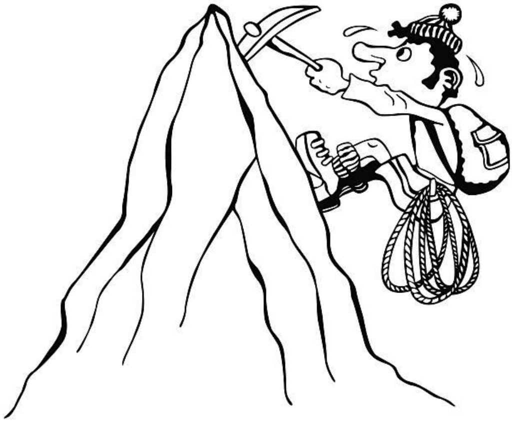
Stochastic Gradient Descent

- Update rule:

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

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:

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

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:

$$\frac{\partial \mathrm{F}}{\partial x} = \begin{bmatrix} \frac{\partial \mathrm{F}_1}{\partial x_1} & \cdots & \frac{\partial \mathrm{F}_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathrm{F}_\mathrm{m}}{\partial x_1} & \cdots & \frac{\partial \mathrm{F}_\mathrm{m}}{\partial x_n} \end{bmatrix} \qquad \left(\frac{\partial \mathrm{F}}{\partial x}\right)_{ij} = \frac{\partial \mathrm{F}_\mathrm{i}}{\partial x_j}$$

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

$$h = f(z)$$

$$z = Wx +$$

$$b$$

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

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:

$$\frac{\partial s}{\partial b} = \underbrace{\boxed{\frac{\partial s}{\partial h}}}_{u^T} \underbrace{\boxed{\frac{\partial h}{\partial z}}}_{\mathrm{diag}(f'(z))} \underbrace{\boxed{\frac{\partial z}{\partial b}}}_{I}$$

Backpropagation

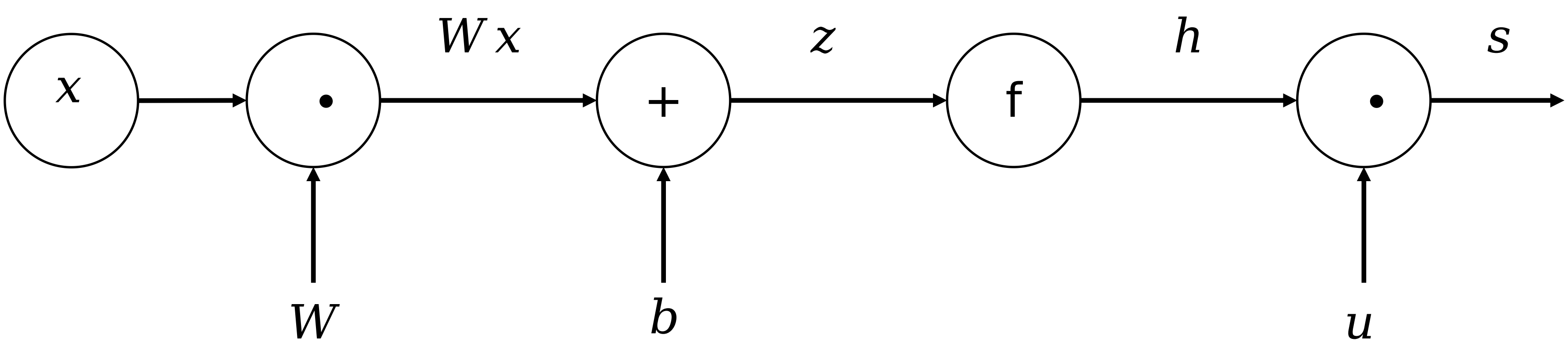
- Compute gradients algorithmically

- 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

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

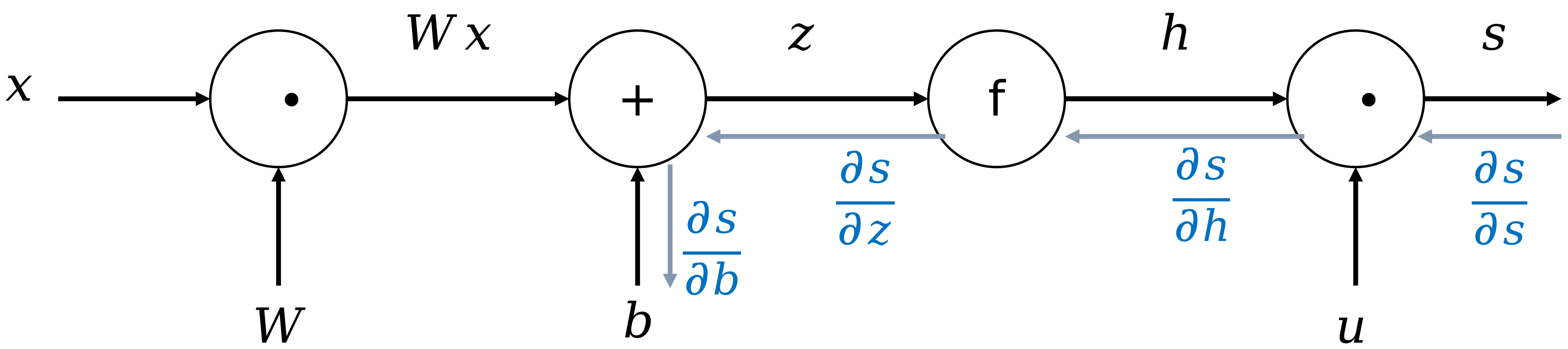


“Forward Propagation”

Backpropagation

- Go backwards along edges
  - Pass along gradients

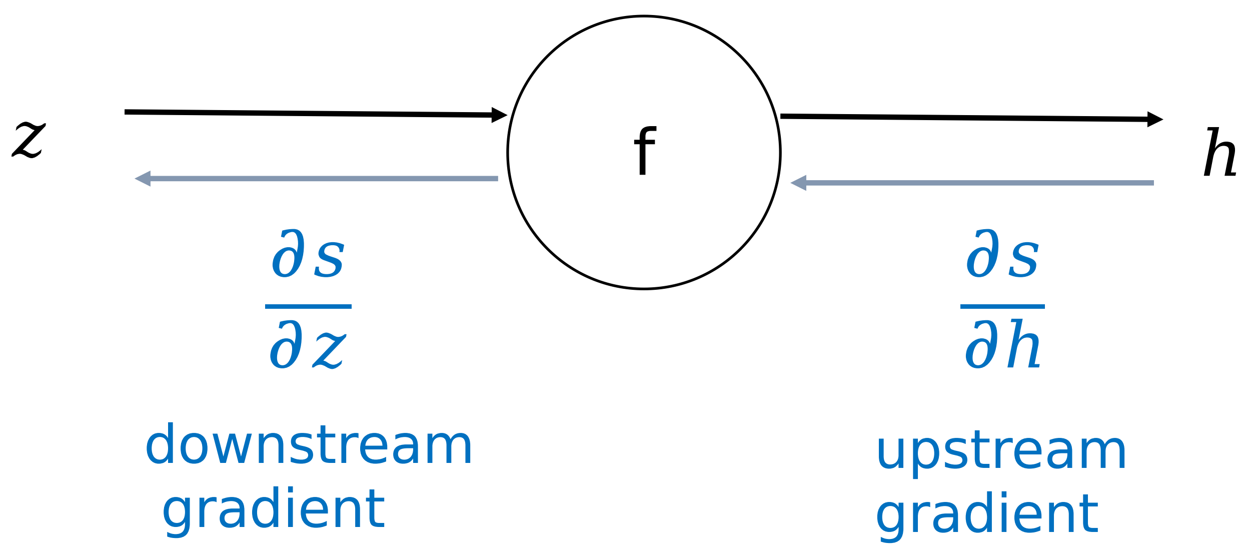
$$s = u^T h$$
$$h = f(z)$$
$$z = Wx + b$$
$$x\text{: input}$$



Backpropagation: Single Node

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

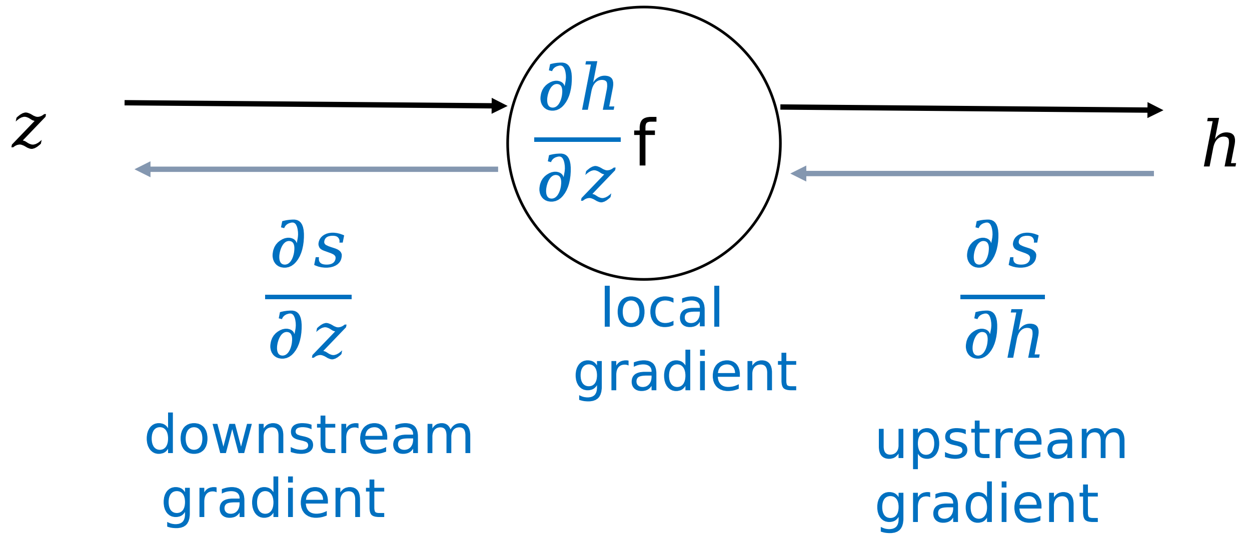
$$h = f(z)$$



Backpropagation: Single Node

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

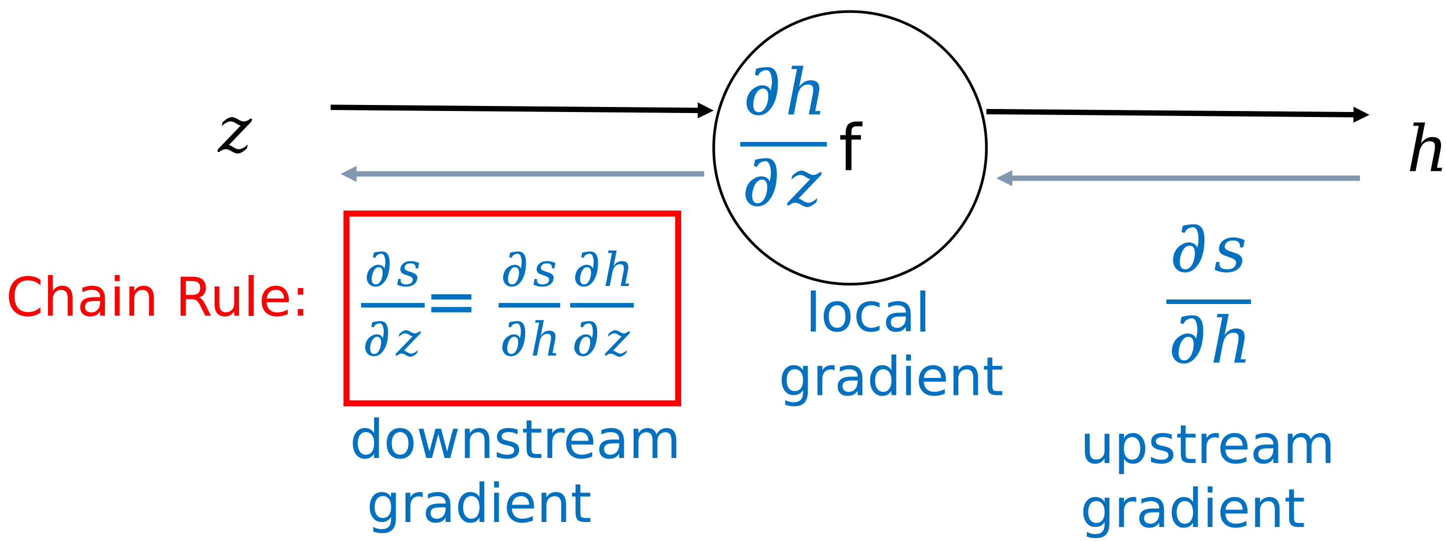
$$h = f(z)$$



Backpropagation: Single Node

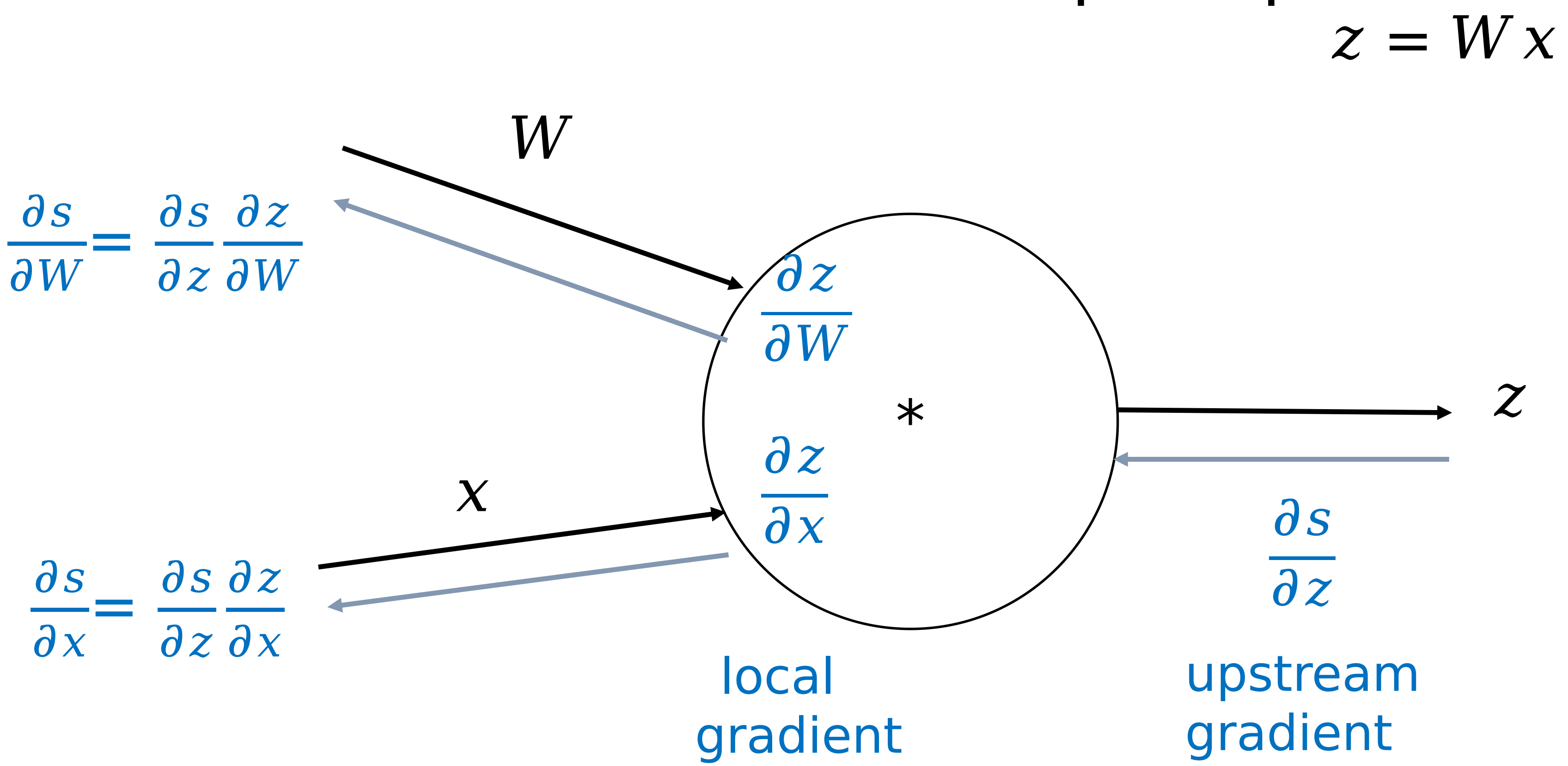
- Each node has a local gradient
  - The gradient of its output with respect to its input
- [downstream gradient] = [upstream gradient] x [local gradient]

$$h = f(z)$$



Backpropagation: Single Node

- What about nodes with multiple inputs?



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

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

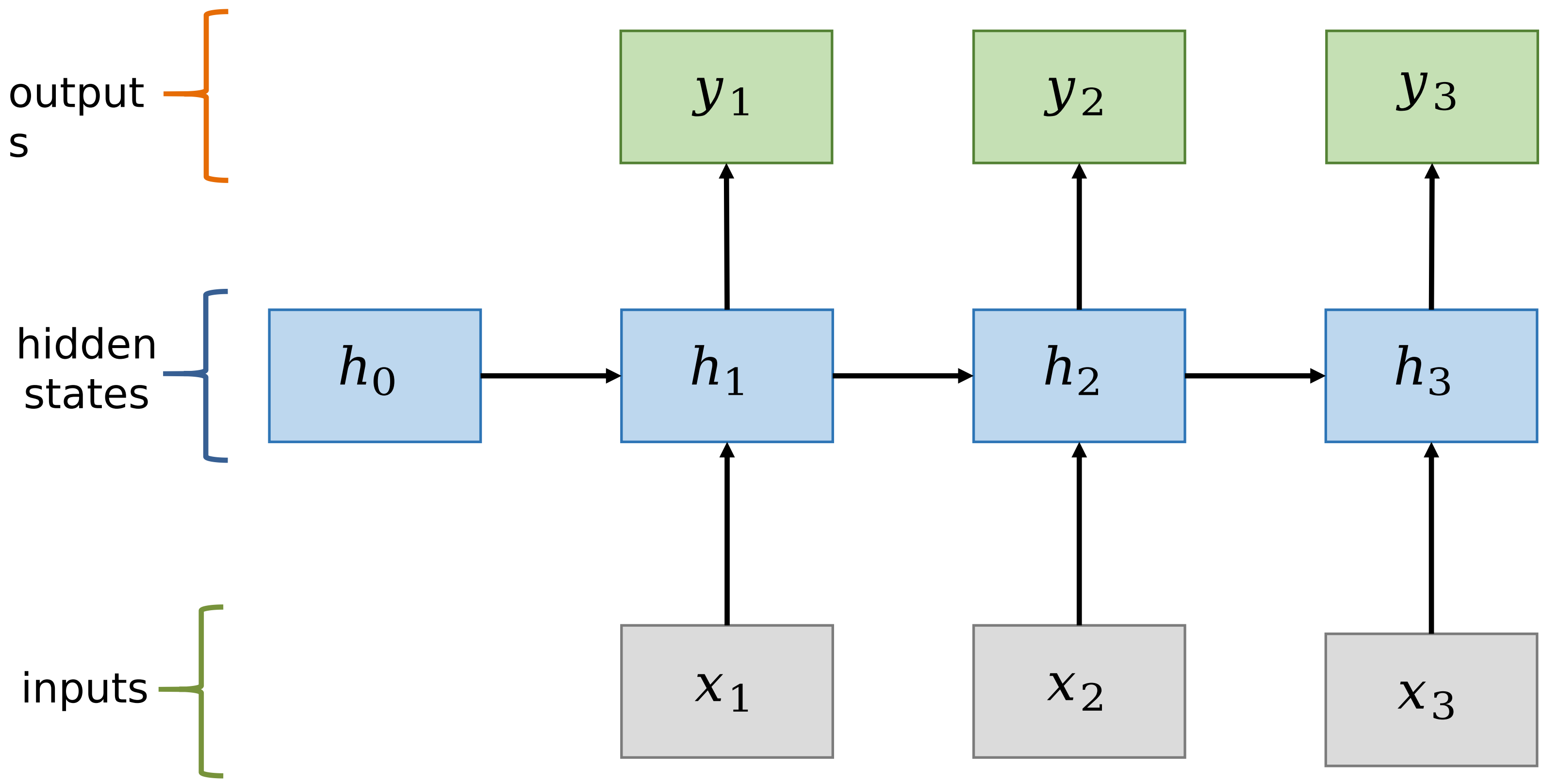
  - Much harder

Sequential Memory

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

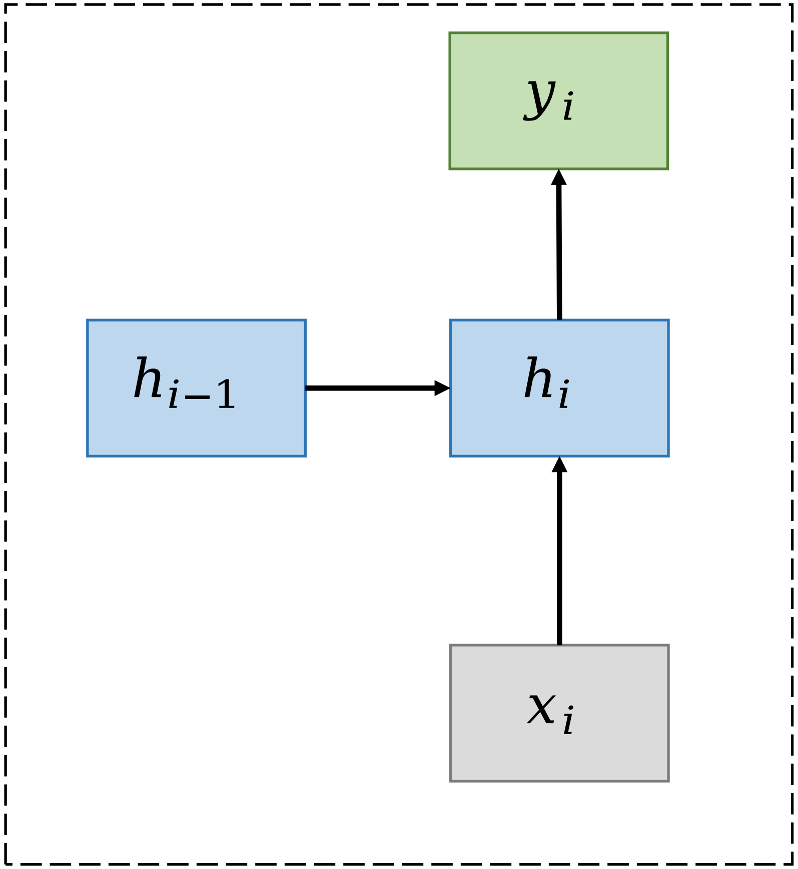
- RNNs update the sequential memory recursively for modeling sequence data

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

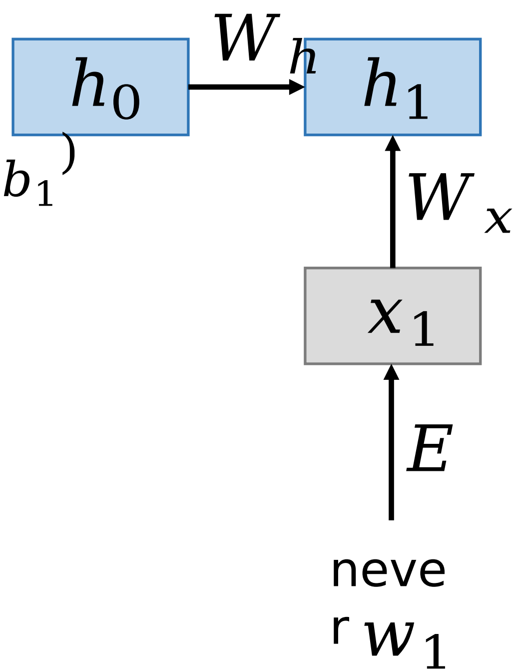
$$h_i = \tanh(W_x x_i + W_h h_{i-1} + b_1)$$

word embeddings

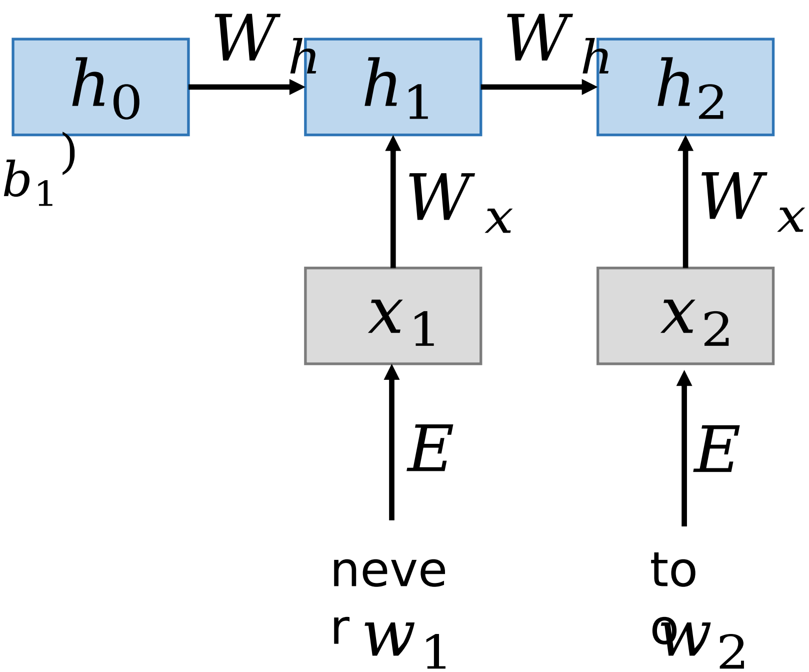
$$x_i = E w_i$$

one-hot vectors

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



RNN Language Model



hidden states

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

word embeddings

$$x_i = E w_i$$

one-hot vectors

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

RNN Language Model

hidden states

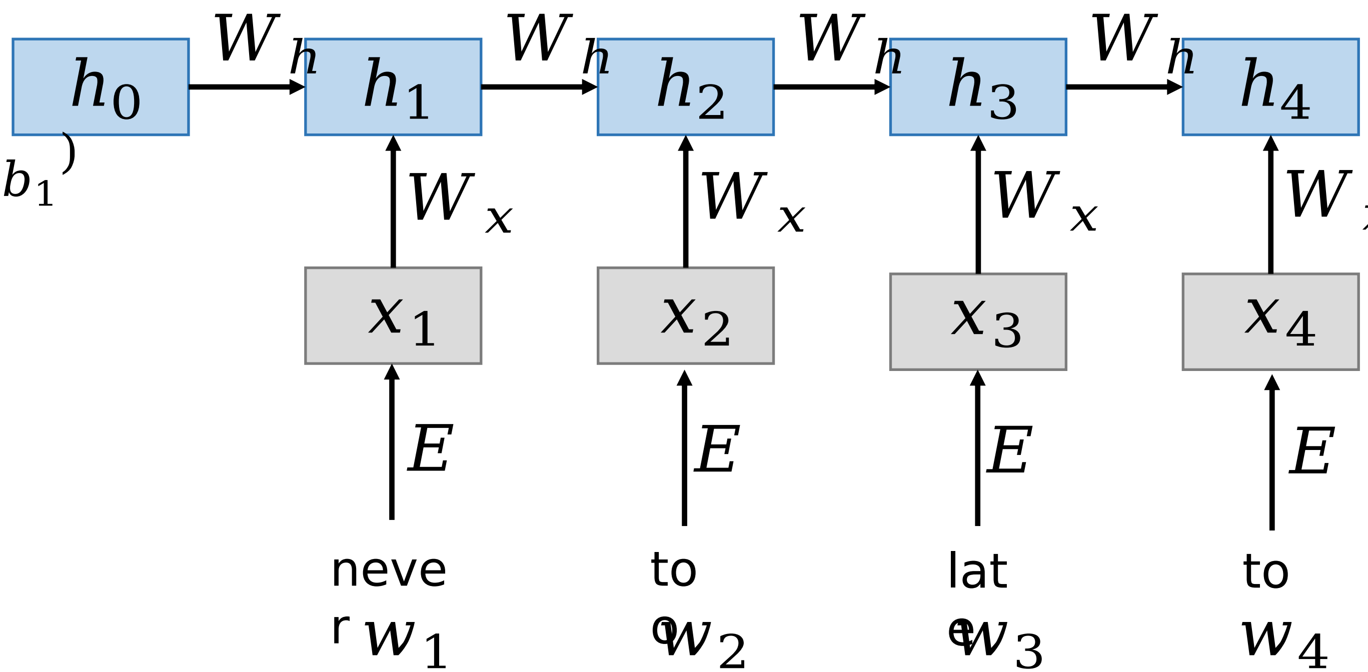
$$h_i = \tanh(W_x x_i + W_h h_{i-1} + b_1)$$

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

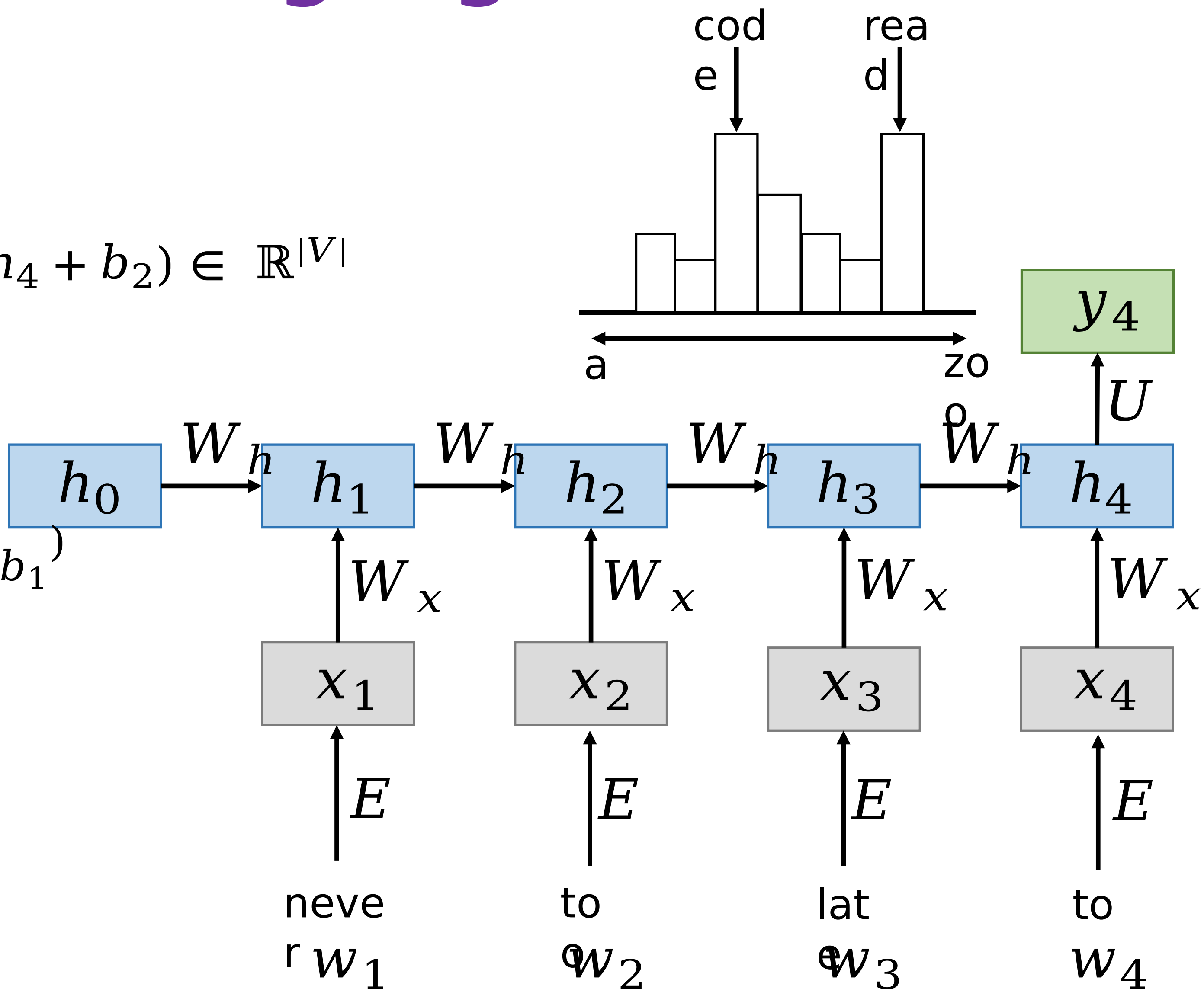
$$h_i = \tanh(W_x x_i + W_h h_{i-1} + b_1)$$

word embeddings

$$x_i = E w_i$$

one-hot vectors

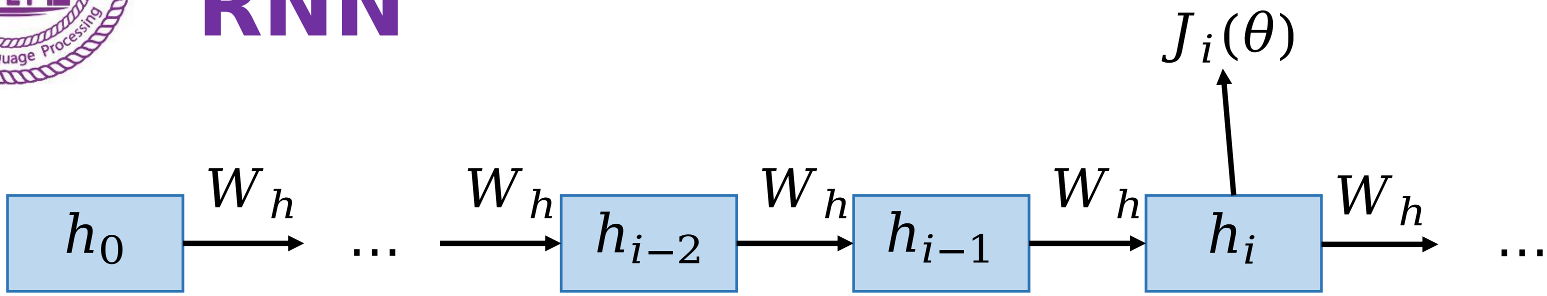
$$w_i \in \mathbb{R}^{|V|}$$



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 $J_i(\theta)$

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

- Recurrent states:

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

- Another parametrization:

$$h_i = W_x x_i + W_\mathrm{h} \tanh(\mathrm{h}_{\mathrm{i}-1}) + \mathrm{b}_1$$

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

Gradient Problem with Vanilla RNN

- More chain rule:

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

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

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

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

  - 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:

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

- Introduce gating mechanism into RNN
- Update gate

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

- Reset gate

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

- 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 $\tilde{h}_i$

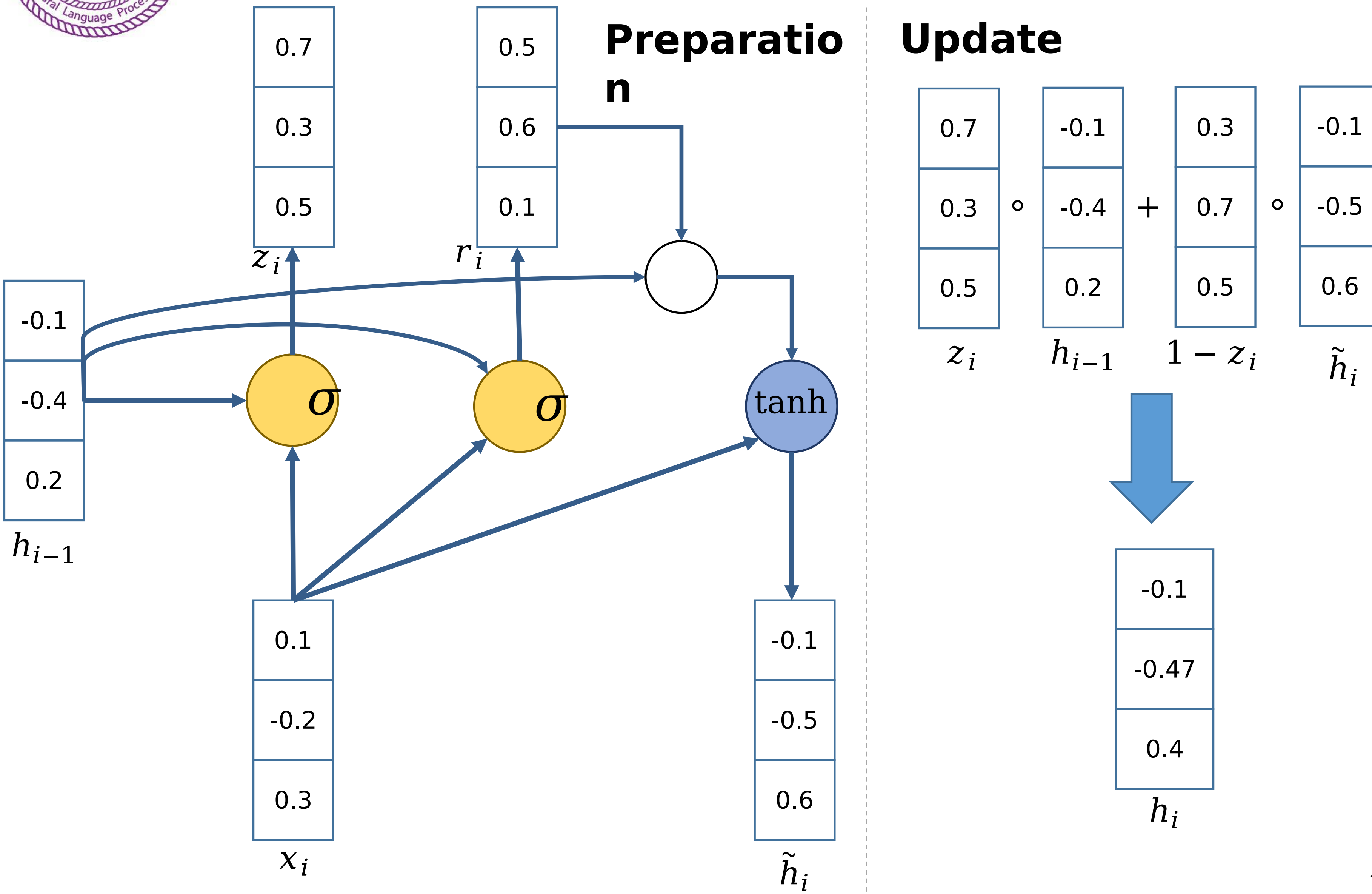
$$\tilde{h}_i = \tanh\left(W_x x_i + r_i * W_h h_{i-1} + b\right)$$

- 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

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

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

- 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.

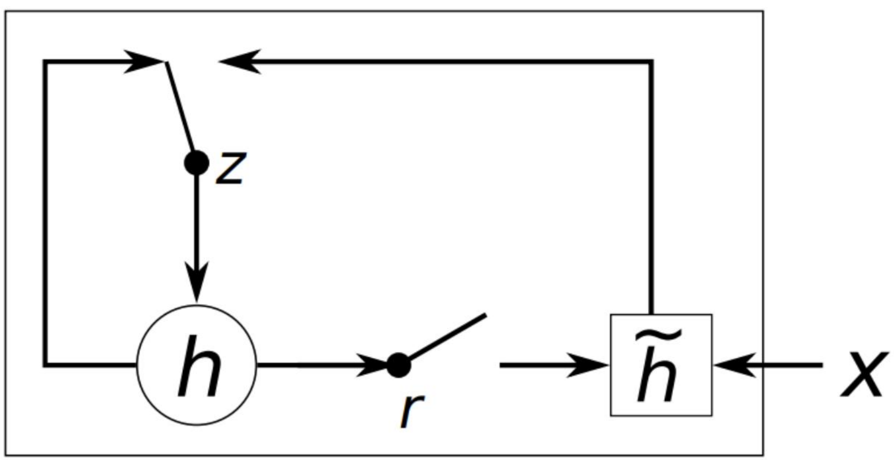
- If z_i close to 1, then we can copy information in that unit through many time steps! (Recall "Constant Error Flow")

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

- 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:

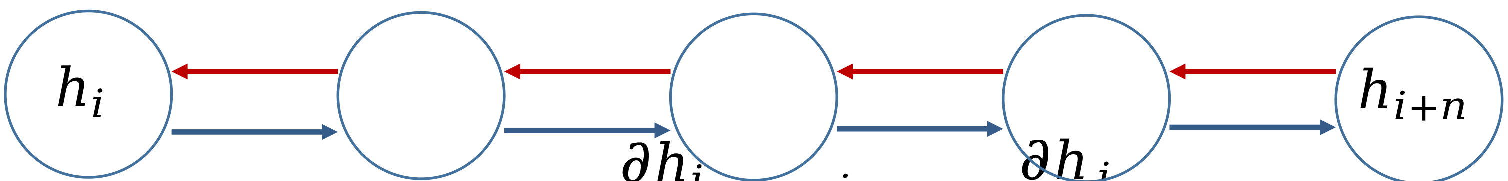


GRU for Vanishing Gradient Problem

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

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

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



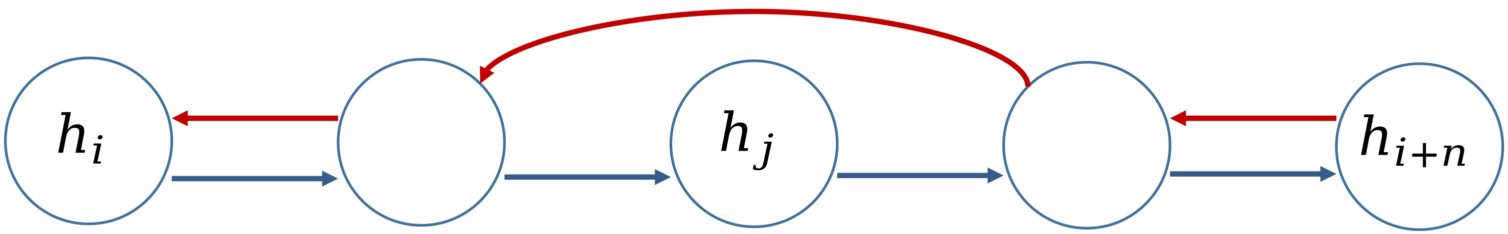
$$\frac{\partial h_i}{\partial h_k} = \Pi_{j=k+1}^{i} \frac{\partial h_j}{\partial h_{j-1}}$$

Create shortcut connection for better backpropagation!

GRU for Vanishing Gradient Problem

- Use update gate z_t to prune unnecessary connections adaptively.

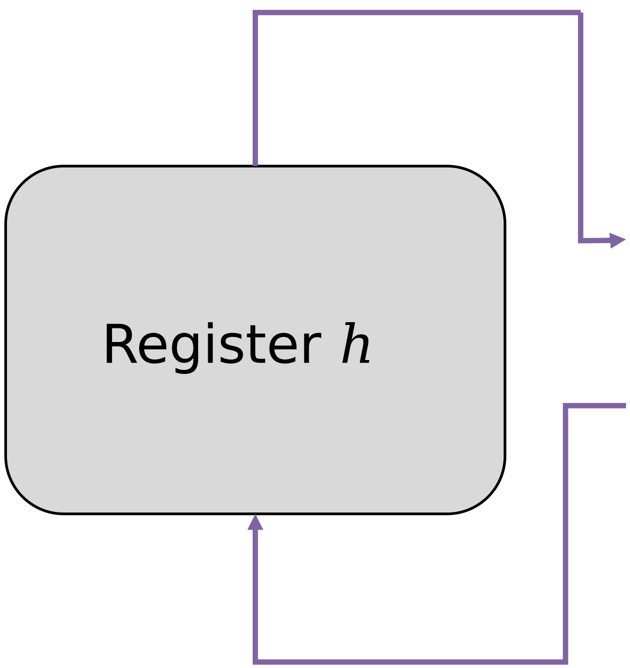
$$h_j = 1 * h_{j-1} + (1-1) * \tilde{h}_j = h_{j-1}$$



- 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



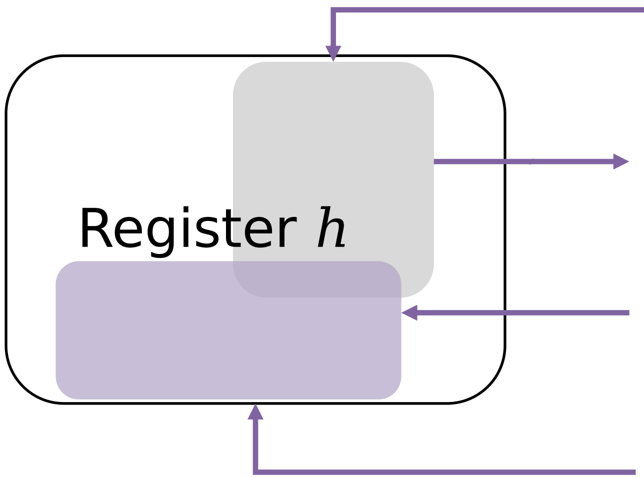
Operation:

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

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

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

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

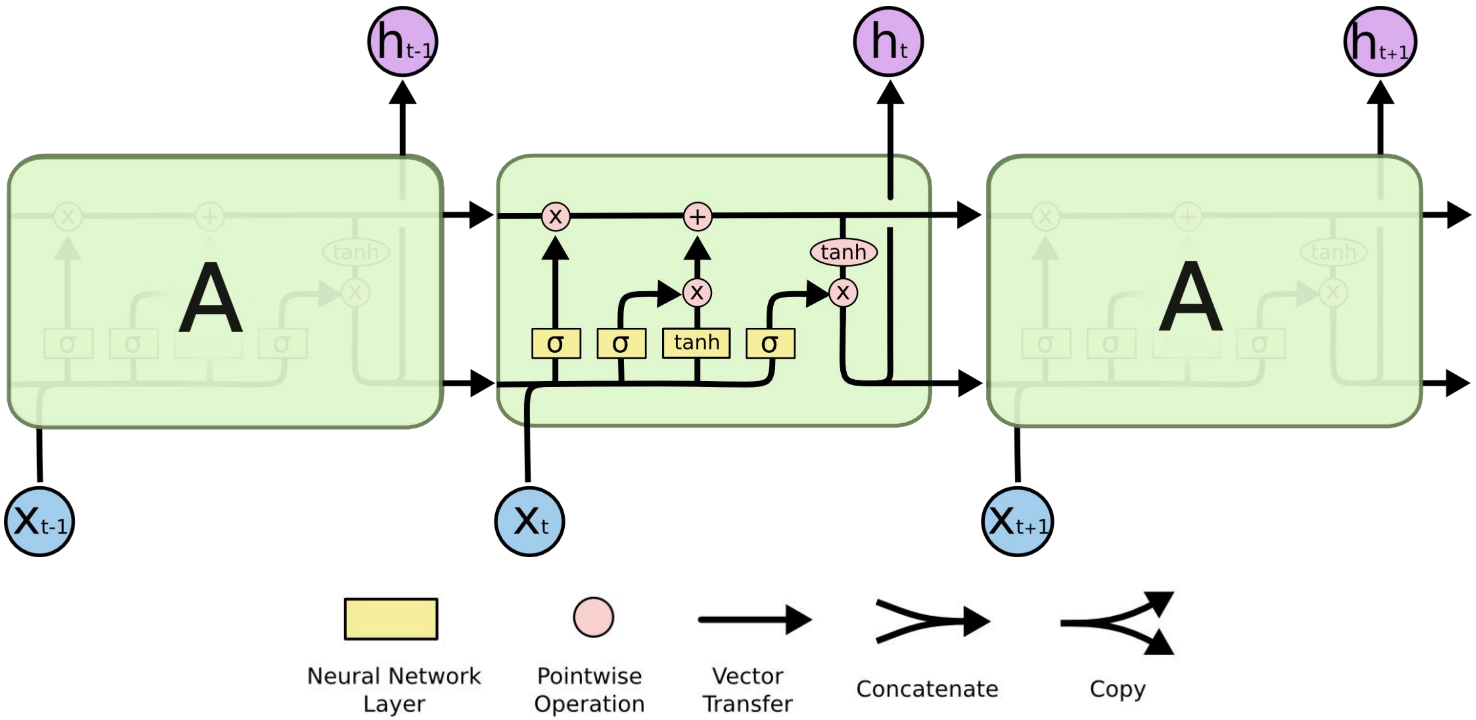
GRU are much more adaptive in updating the hidden state!

Long Short-Term Memory Network (LSTM)

THUNLP

Long Short-Term Memory Network

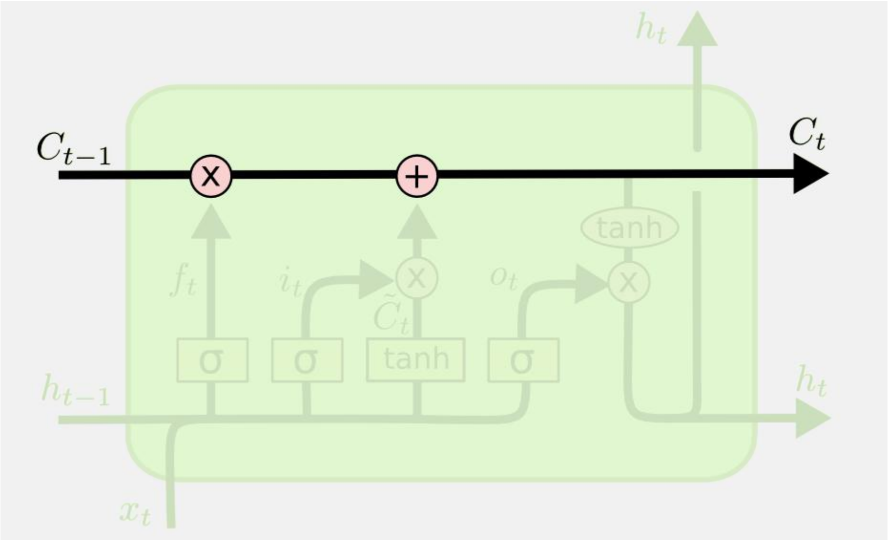
- Long Short-Term Memory network (LSTM)
- LSTM are a special kind of RNN, capable of learning long-term dependencies like GRU



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

Long Short-Term Memory Network

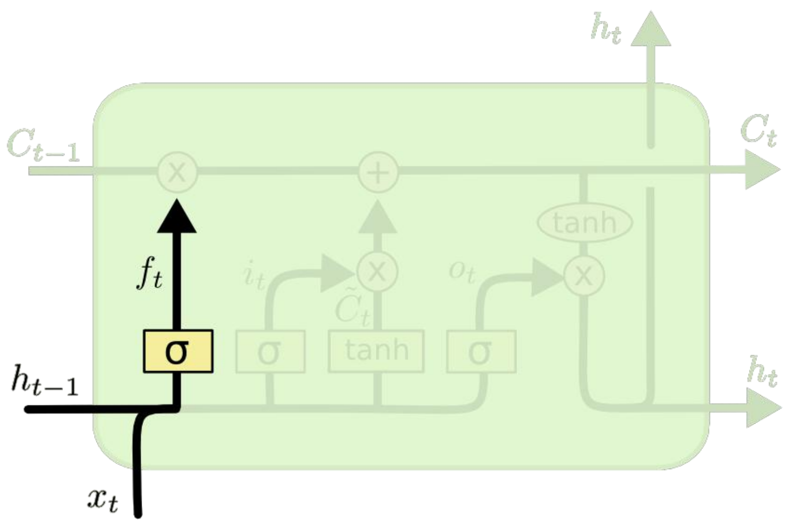
- The key to LSTMs is the cell state C_t



  - 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

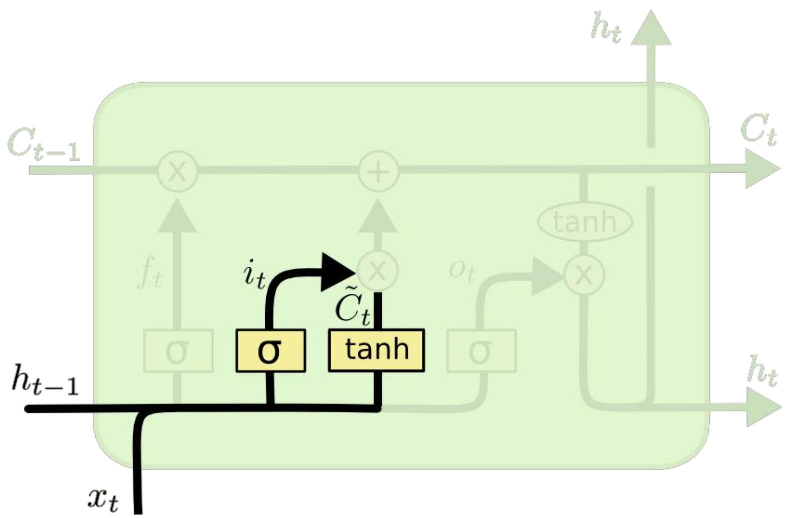


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

- 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 $\widetilde{C}_t$



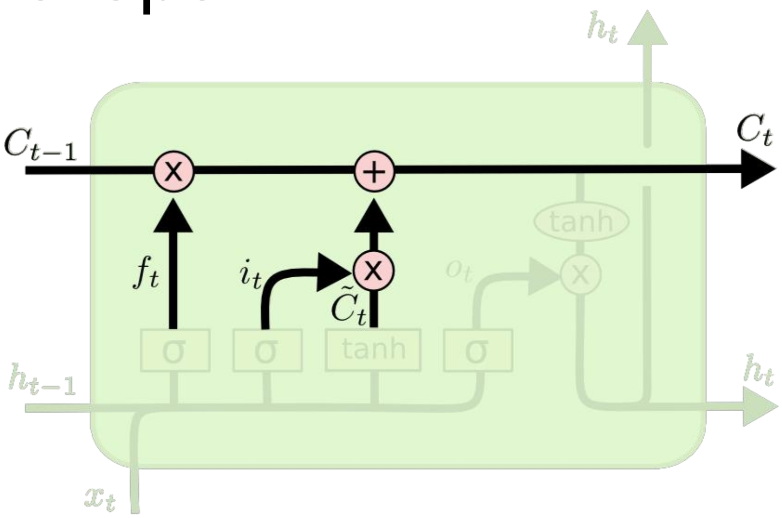
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

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

Long Short-Term Memory Network

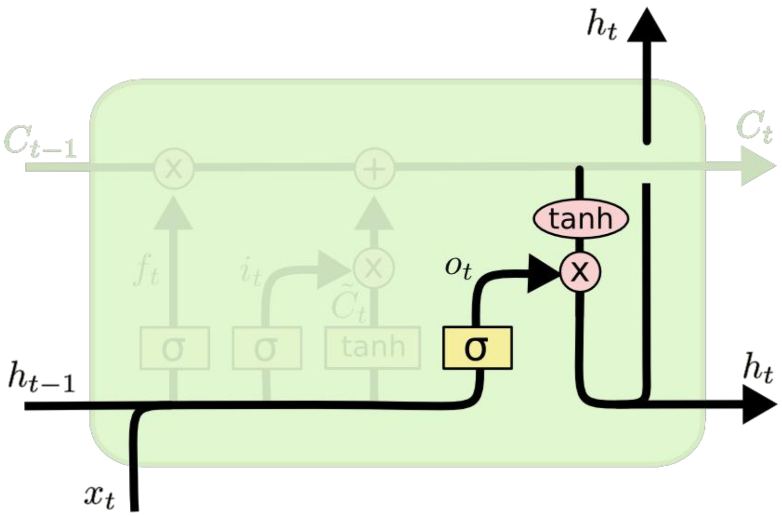
- Update the old cell state C_{t-1}
- Combine the results from the previous two steps



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

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

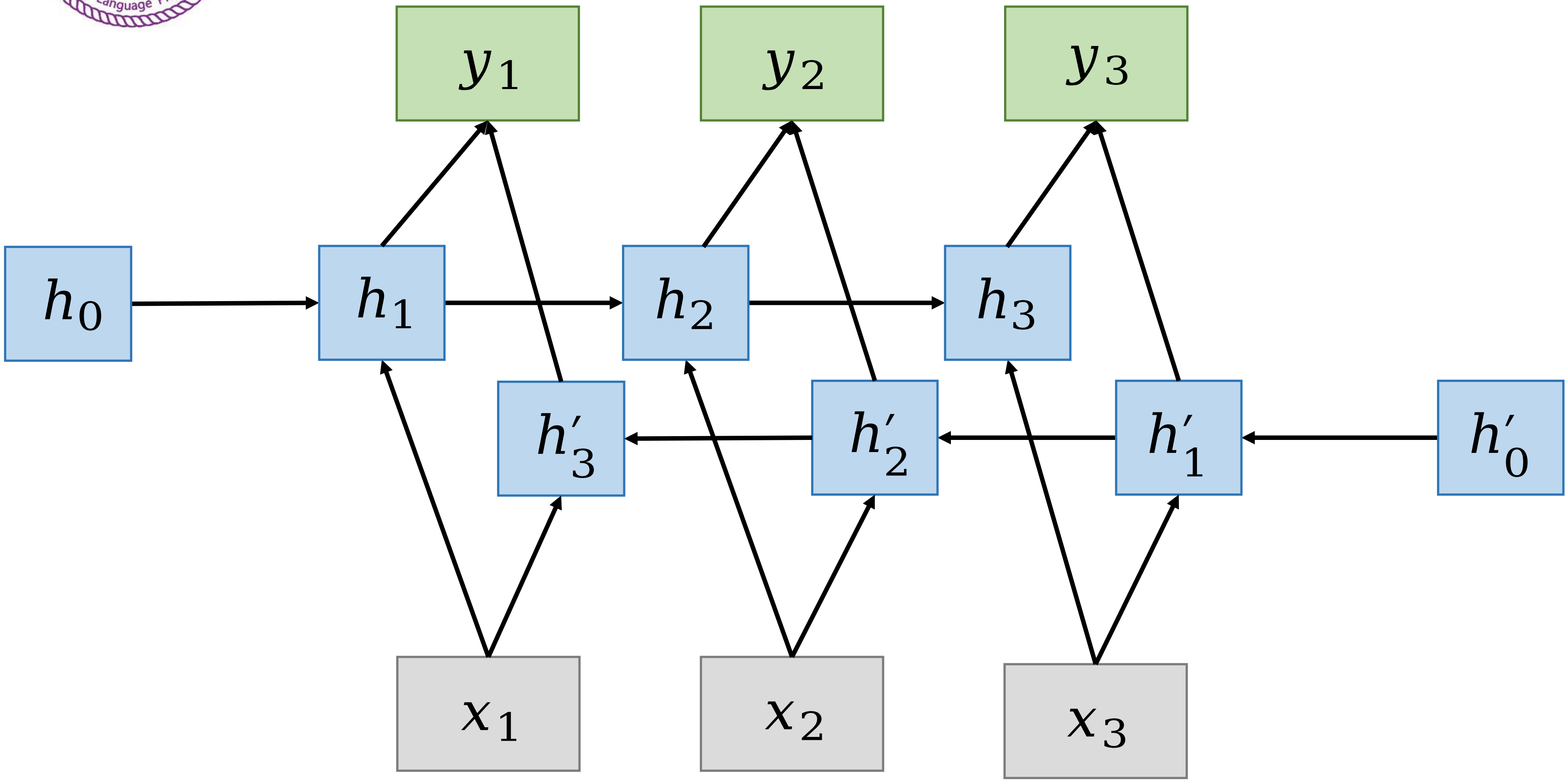
Bidirectional RNNs

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

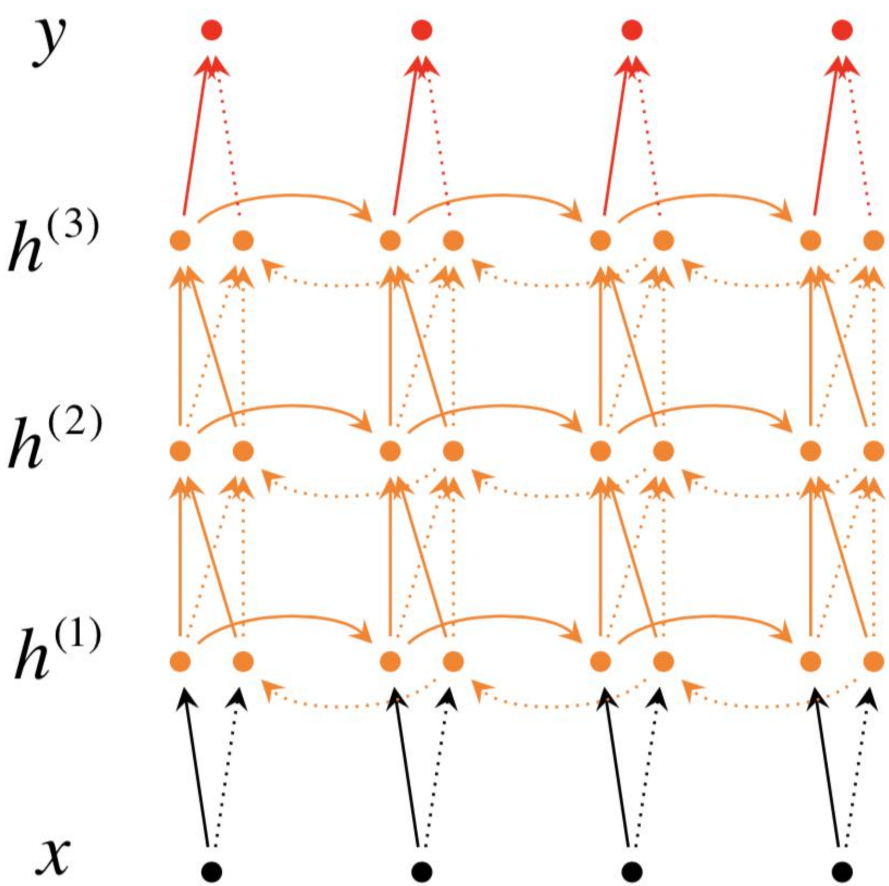
$$h_t = f(x_{t-1}, \ldots, x_2, x_1)$$

- 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



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

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

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

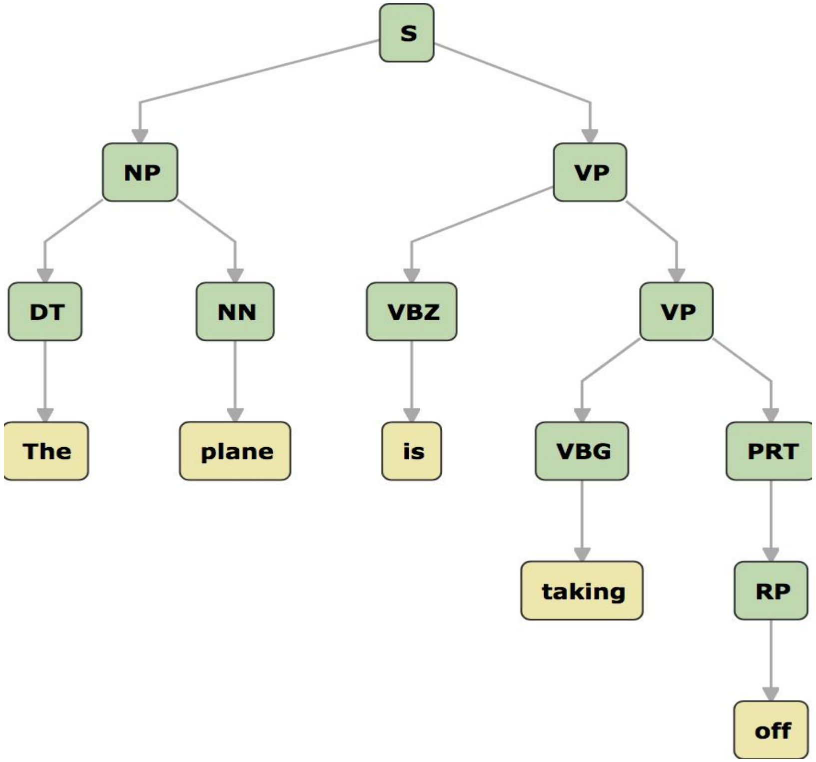
Has multiple layers per time step

Tree LSTM

THUNLP

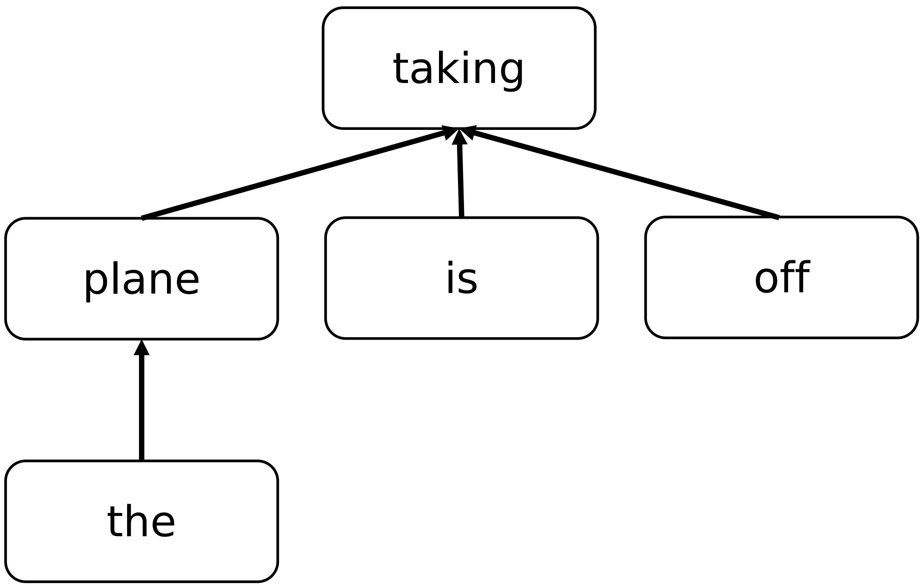
Sentence Structure

- 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: $\boxed{\tilde{h}_j} = \sum_{k \in C(j)} h_k$
  - Input gate: $i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \boxed{\tilde{h}_j} + b^{(i)}\right)$
  - Forget gate: $f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} \boxed{h_k} + b^{(f)}\right)$
  - Output gate: $o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \boxed{\tilde{h}_j} + b^{(o)}\right)$
  - New memory cel $u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \boxed{\tilde{h}_j} + b^{(u)}\right)$
  - Final memory cel $c_j = i_j \odot u_j + \boxed{\sum_{k \in C(j)} f_{jk} \odot c_k}$
  - Final hidden state $h_j = o_j \odot \tanh(c_j)$

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: $$i_j = \sigma\left(W^{(i)}x_j + \boxed{\sum_{\ell=1}^{N} U_{\ell}^{(i)}} h_{j\ell} + b^{(i)}\right)$$
  - Forget gate: $$f_{jk} = \sigma\left(W^{(f)}x_j + \boxed{\sum_{\ell=1}^{N} U_{k\ell}^{(f)}} h_{j\ell} + b^{(f)}\right)$$
  - Output gate: $$o_j = \sigma\left(W^{(o)}x_j + \boxed{\sum_{\ell=1}^{N} U_{\ell}^{(o)}} h_{j\ell} + b^{(o)}\right)$$
  - New memory cell: $$u_j = \tanh\left(W^{(u)}x_j + \boxed{\sum_{\ell=1}^{N} U_{\ell}^{(u)}} h_{j\ell} + b^{(u)}\right)$$
  - Final memory cel $$c_j = i_j \odot u_j + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell}$$
  - Final hidden state $$h_j = o_j \odot \tanh(c_j)$$

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)

possible n-gram
phrases

The plane is taking off

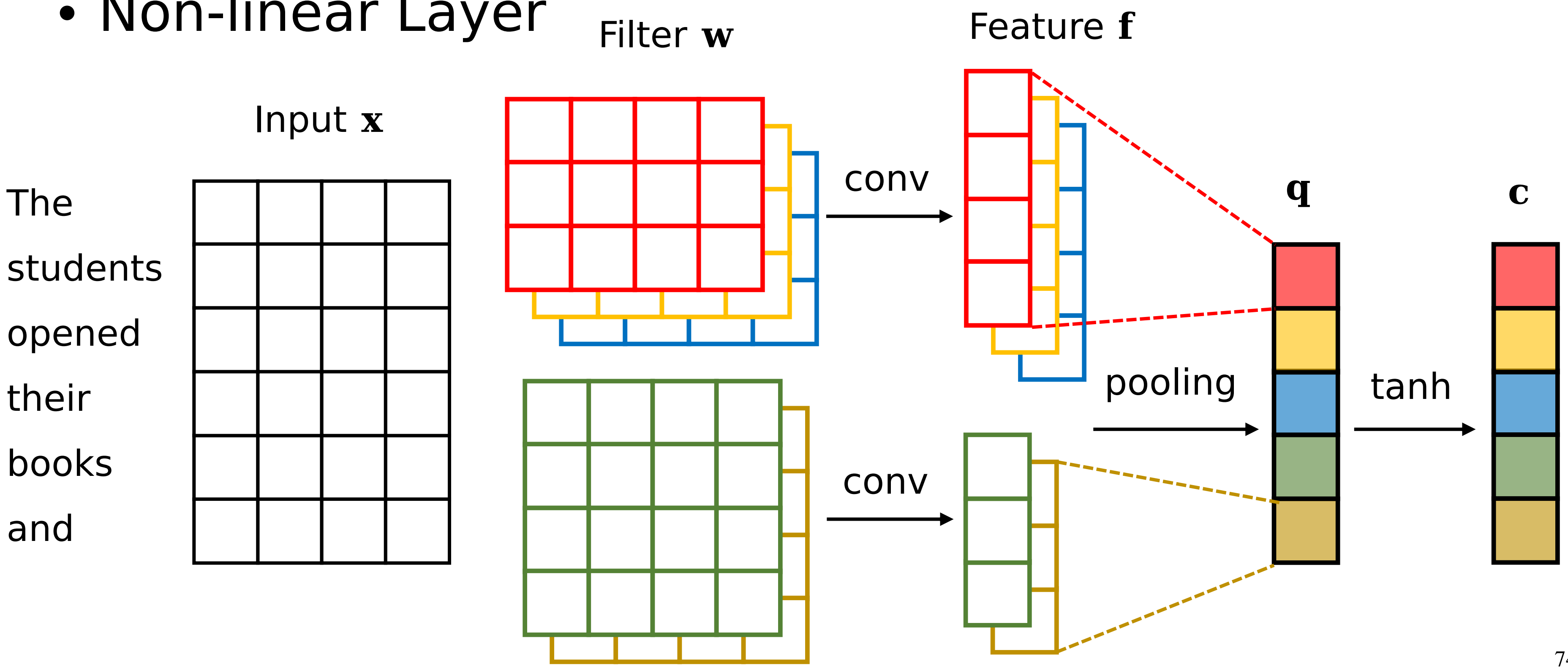
Bigram: *The plane, plane is, is taking, taking off*

Trigram: *The plane is, plane is taking, is taking o*

n-gram: …

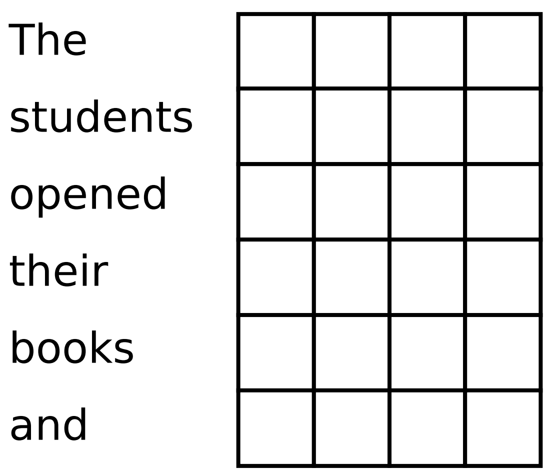
Architecture

- 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)

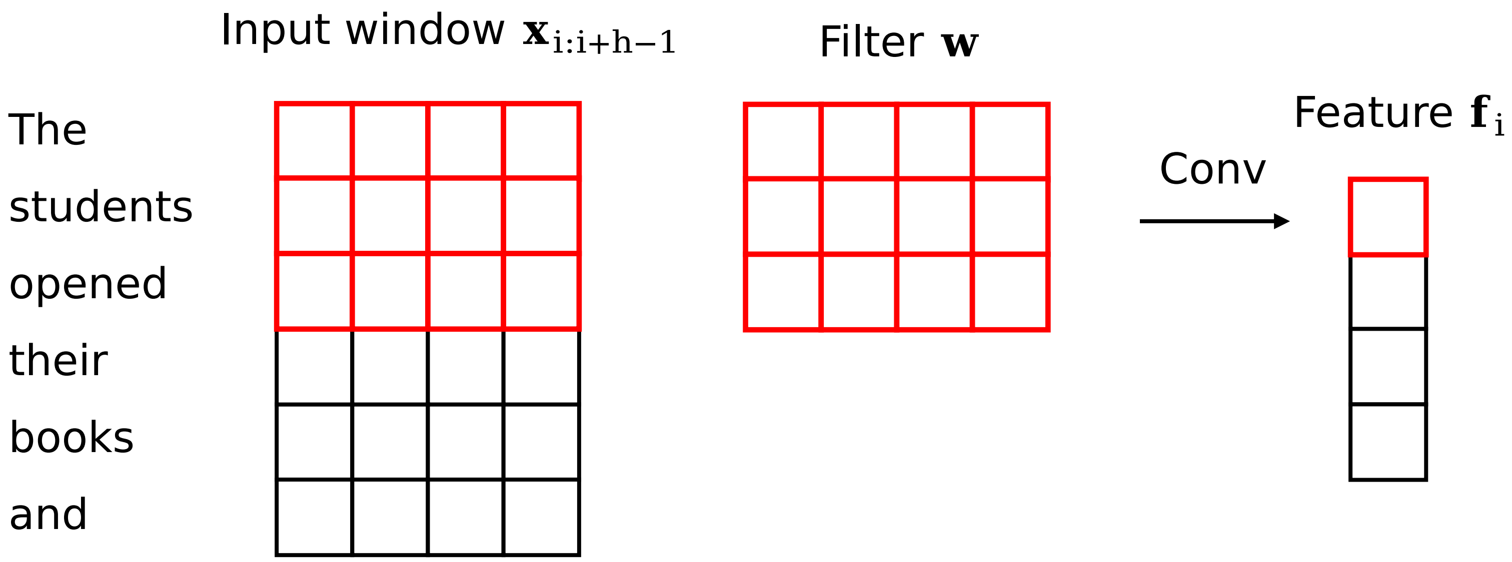
$$\mathbf{f}_{\mathrm{i}} = \mathbf{w} \cdot \mathbf{x}_{\mathrm{i:i+h-1}} + \mathrm{b}, \quad \mathrm{i} = 1, 2, \ldots, \mathrm{n} - \mathrm{h} + 1$$

  - $\mathbf{f} \in \mathrm{R}^{n-h+1}$: convolved feature representation
  - · is dot product

Convolution Layer

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

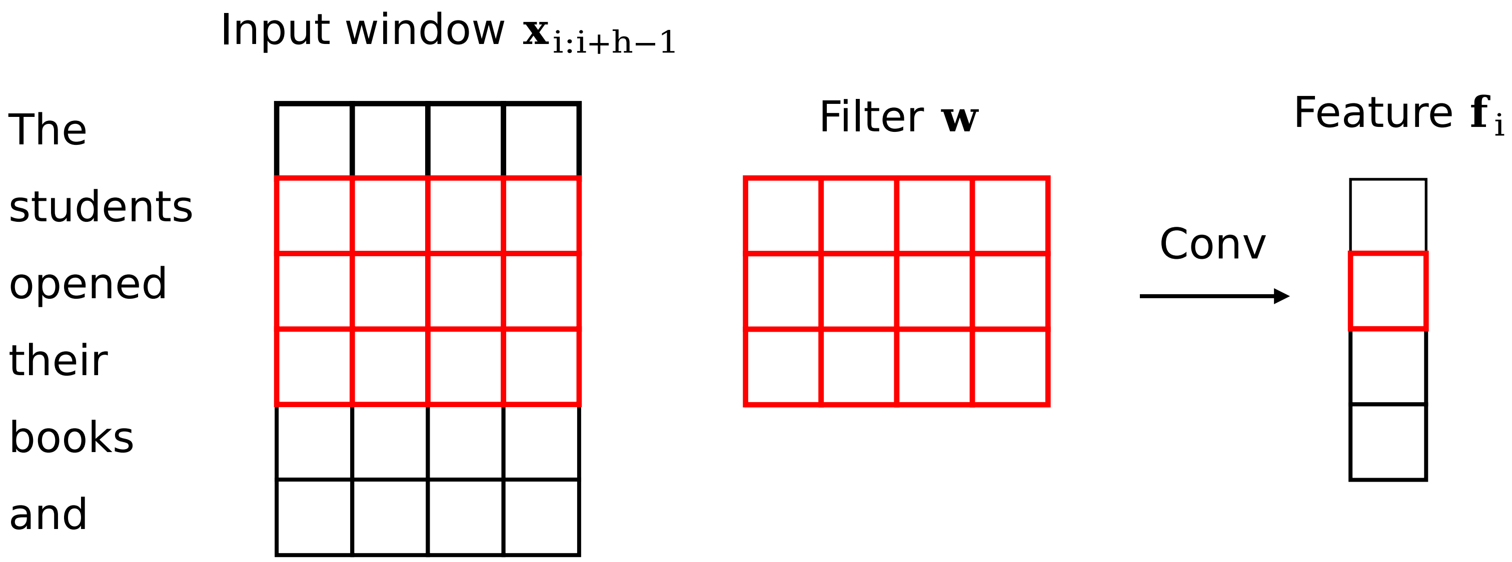
$$\mathbf{f}_i = \mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b, \quad i = 1,2,\ldots,\ n-h+1$$



Convolution Layer

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

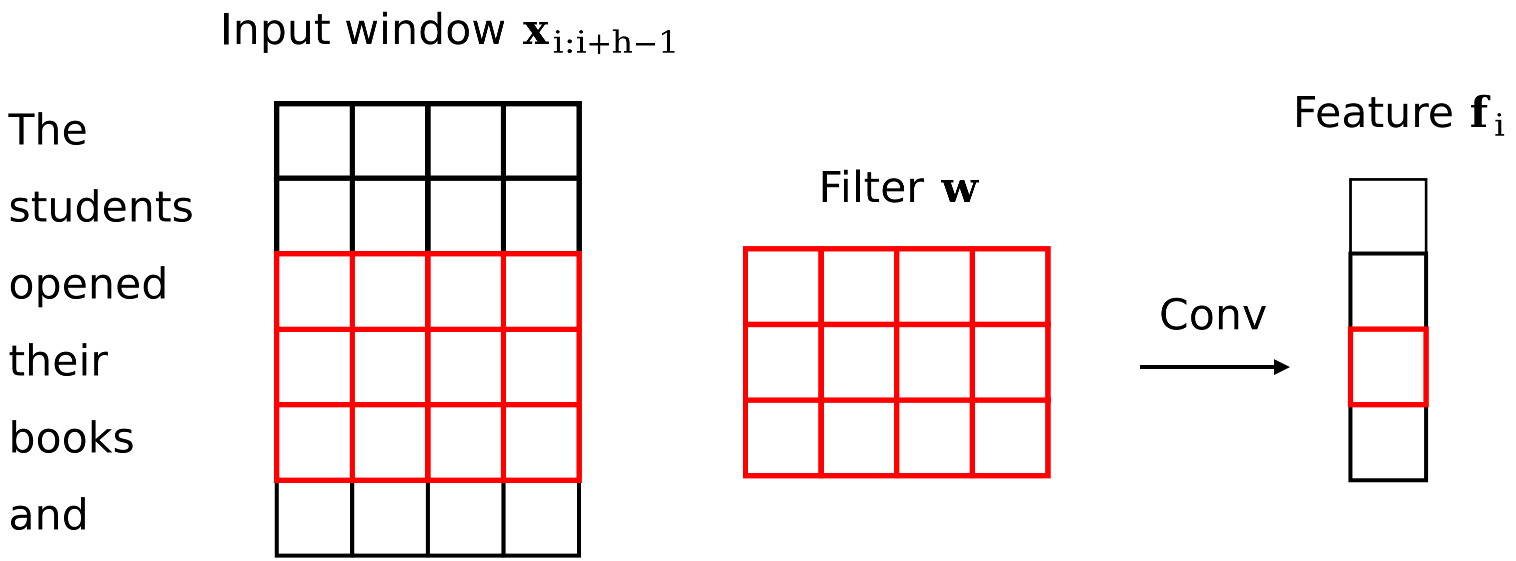
$$\mathbf{f}_i = \mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b, \quad i = 1,2, \ldots, \; n - h + 1$$



Convolution Layer

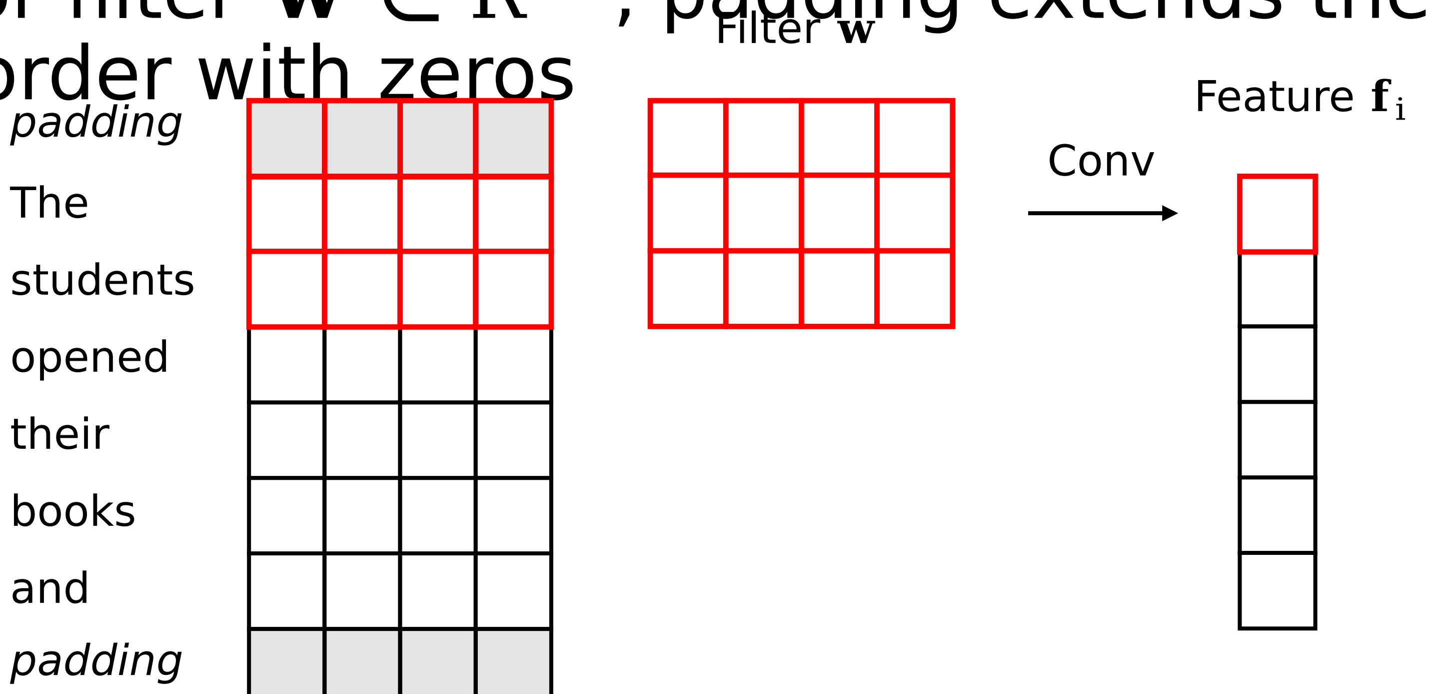
- Extract feature representation from input representation via a sliding convolving filter

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



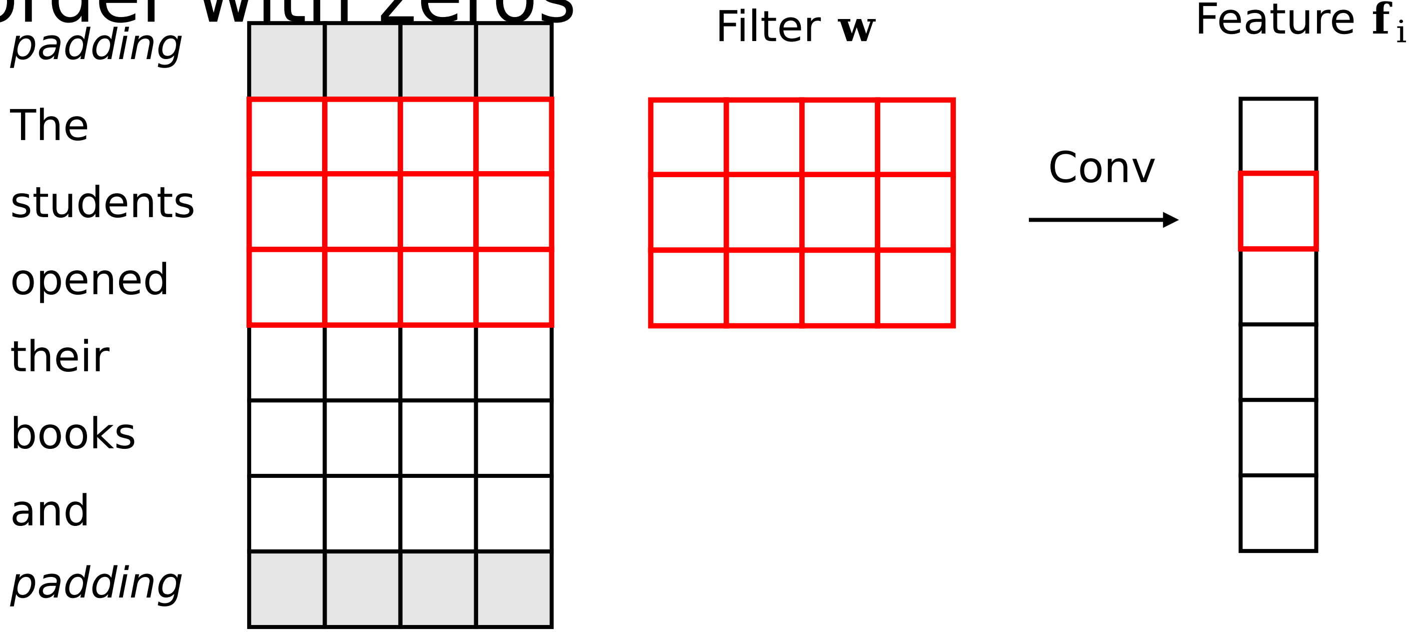
Convolution Layer (with padding)

- 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



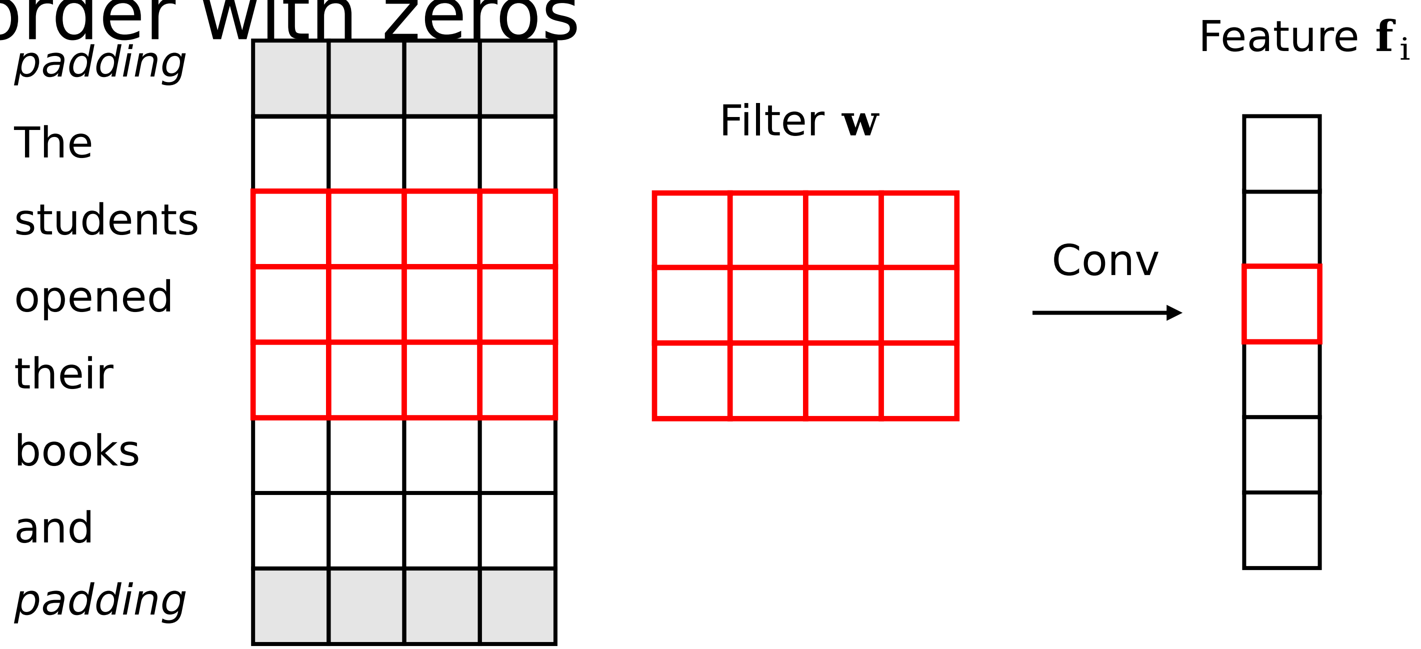
Convolution Layer (with padding)

- 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)

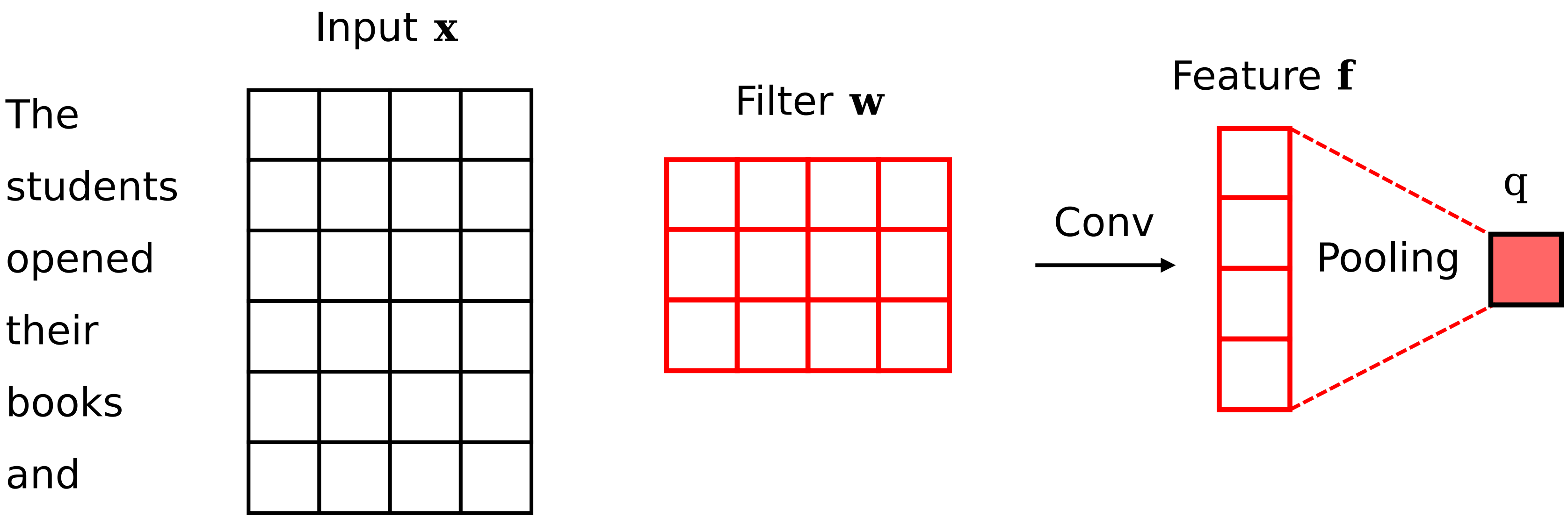
- Padding is an operation that extends the border of the sentence before convolution, to keep the shape of convoluted feature same as input
- For filter $\mathbf{w} \in \mathrm{R}^{hd}$, padding extends the border with zeros



Max-Pooling Layer

- Max-pooling:
  - Extract important features

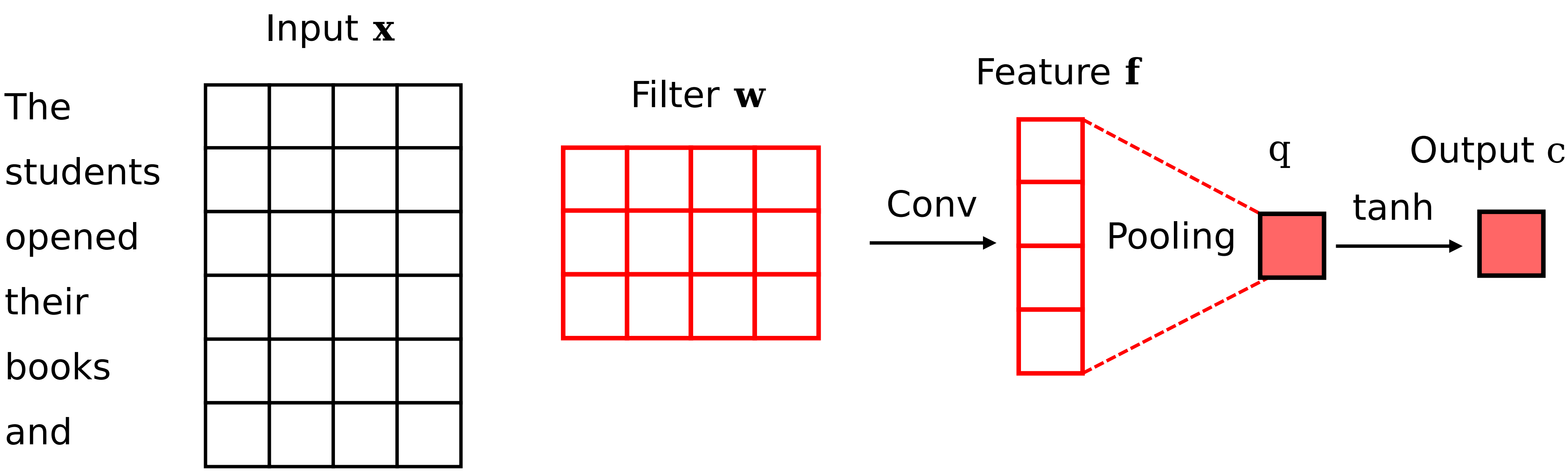
$$q = \max(\mathbf{f})$$



Non-Linear Layer

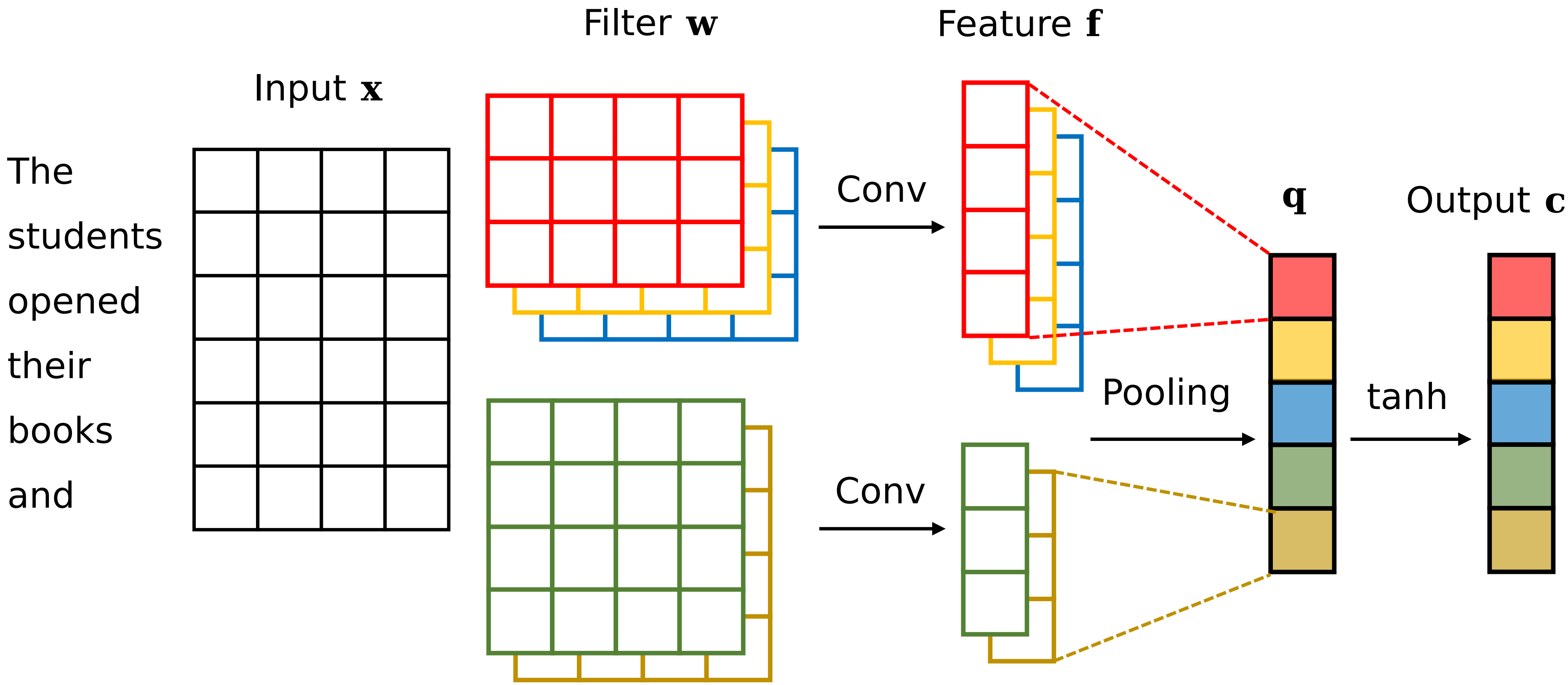
- Non-Linear Activation Function:

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



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

Section 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.

Outline

- Introduction to Seq2Seq
- Machine Translation
    - Introduction
    - Statistical Machine Translation
    - Neural Machine Translation

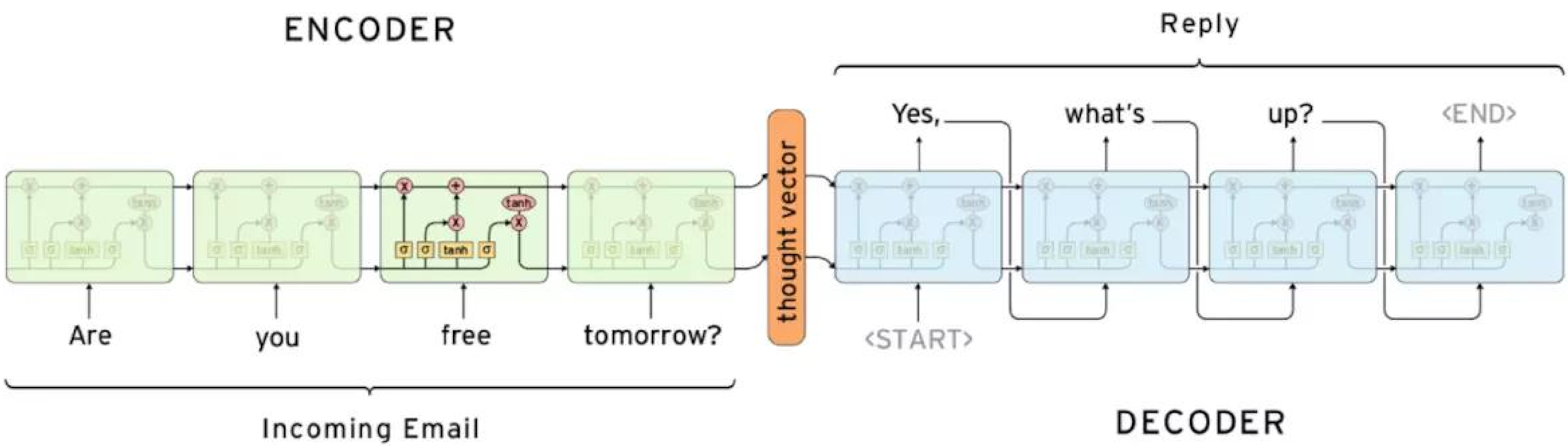
Introduction to Seq2Seq

- Sequence-to-sequence (Seq2Seq): a family of machine learning methods used for language processing

- Architecture:
  - An encoder that produces representations of the source sentence
  - A decoder which is a language model that generates target sentence conditioned on encoding
  - The encoder/decoder can be realized by RNN/GRU/LSTM/Transformer

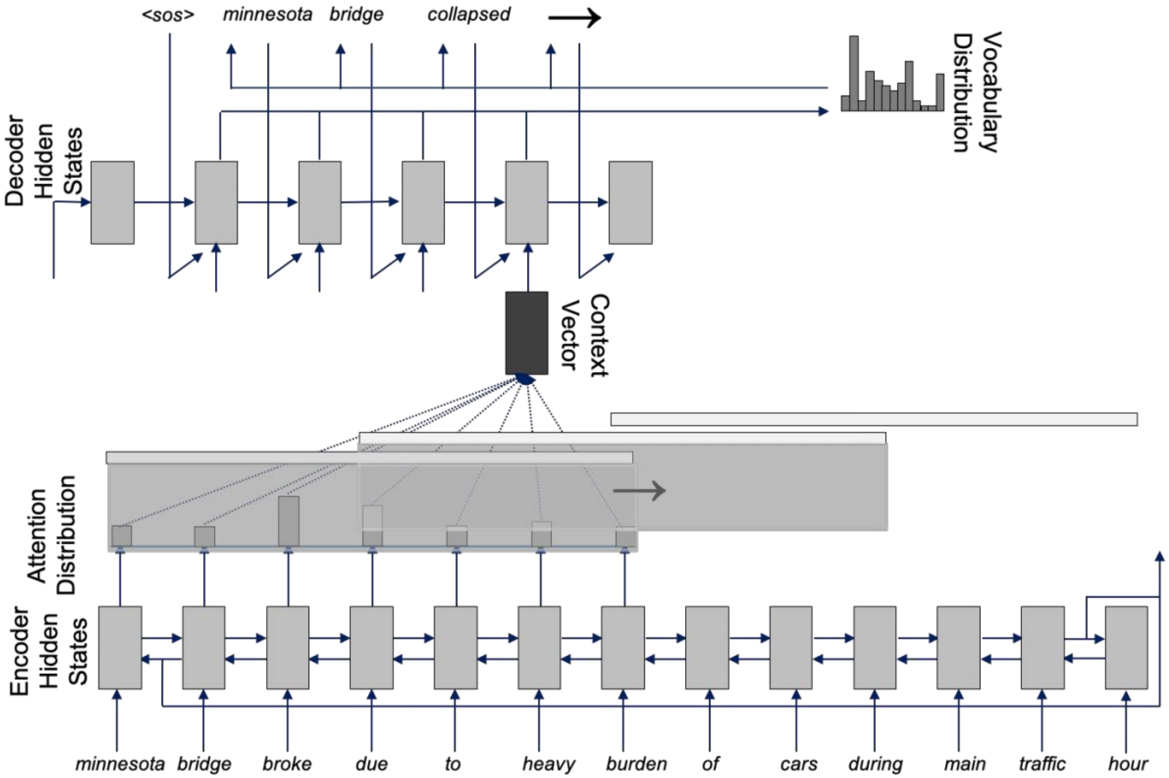
Introduction to Seq2Seq

- Typical Applications:
  - Conversational Models
    - Incoming Email -> Reply



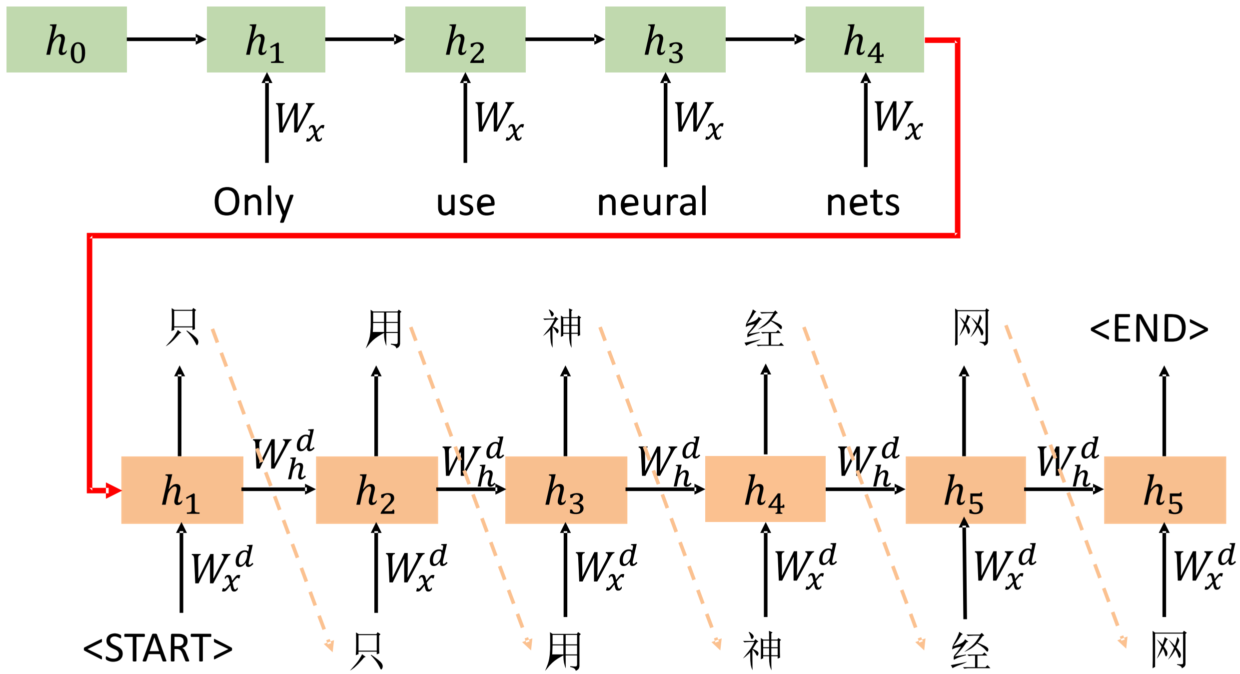
Introduction to Seq2Seq

- Typical Applications:
  - Text Summarization
    - Long text -> Short summary



Introduction to Seq2Seq

- Typical Applications:
  - Machine Translation
    - Language A -> Language B



Introduction to Seq2Seq

- Typical Applications:
  - Conversational Models
    - Incoming Email -> Reply
  - Text Summarization
    - Long text -> Short summary
  - Machine Translation
    - Language A -> Language B
    - We use machine translation as the example in this lecture.

Outline

- Introduction to Seq2Seq
- Machine Translation
  - Introduction
  - Statistical Machine Translation
  - Neural Machine Translation

Outline

- Introduction to Seq2Seq
- Machine Translation
  - Introduction
  - Statistical Machine Translation
  - Neural Machine Translation

Machine Translation

- Machine Translation(MT): the task of translating text from source language to target language

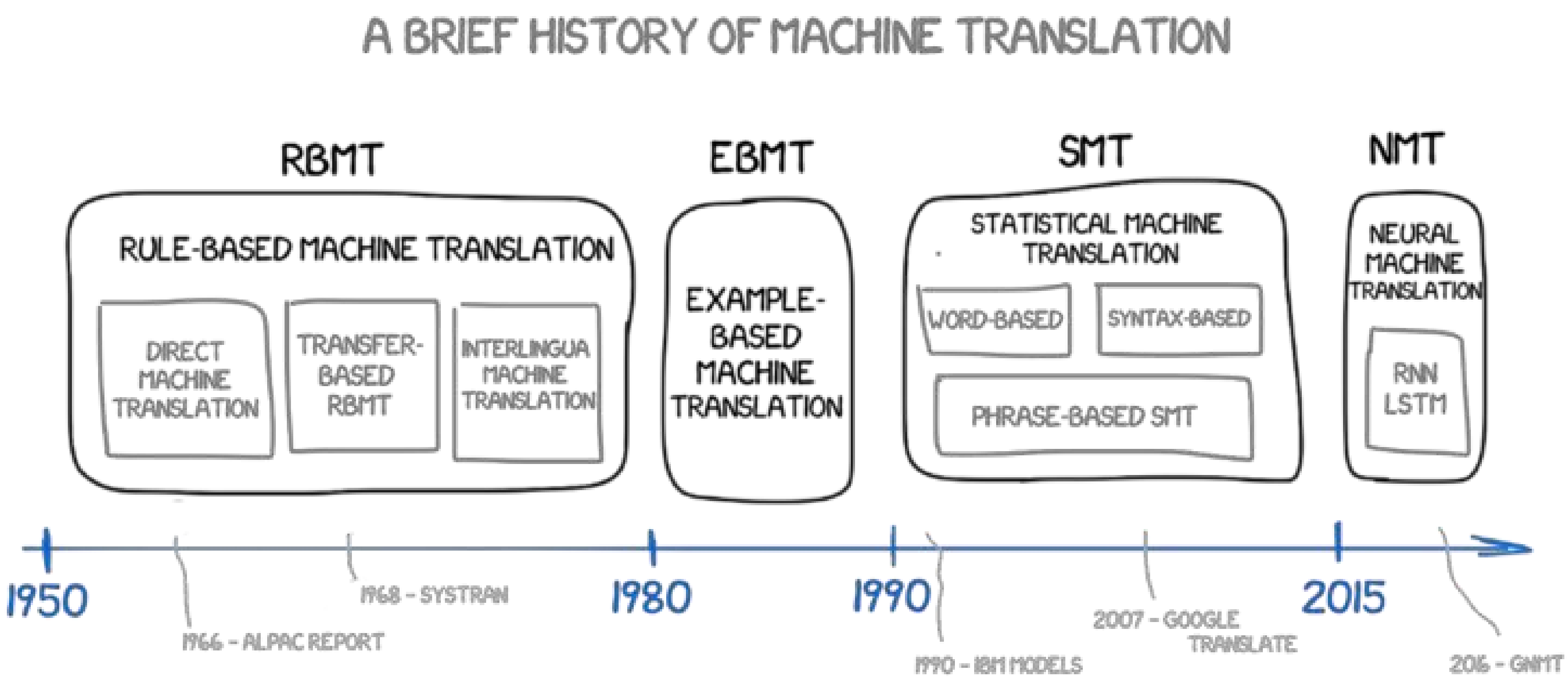
Chinese: 布什与沙龙举行了会谈

↓

English: Bush held a talk with Sharon

Machine Translation

- History (link)



RBMT

- Rule-based machine translation (RBMT)
- Machine Translation research began in the early 1950s
- Mostly Russian → English

  (motivated by the Cold War!)
- Systems were mostly rule-based, using a bilingual dictionary to map Russian words to their English counterparts
- Extremely complicated

EBMT

- Example-based machine translation (1984)
- Translation of fragmental phrases by analogy
- Extract matching templates from bilingual corpus:

English	Chinese
How much is that **red umbrella**?	那个**红雨伞**多少钱？
How much is that **small camera**?	那个______多少钱？

小相机

- Feed the machine with existing translations and don't need to spend years forming rules and exceptions

Outline

- Introduction to Seq2Seq
- Machine Translation
  - Introduction
  - Statistical Machine Translation
  - Neural Machine Translation

Statistical Machine Translation

- Core idea: Learn a probabilistic model from data

x 布什 与 沙龙 举行 了 会谈

⇩

$$\text{argmax}_\text{y} P(y|x;\theta)$$

⇩

y Bush held a talk with Sharon

Statistical Machine Translation

- Suppose we are translating Chinese → English.
- We want to find best English sentence y, given Chinese sentence x

$$\mathrm{argmax}_y P(y|x)$$

- Use Bayes rule to break this down into two components to be learnt separately:

$$= \mathrm{argmax}_y \frac{P(x|y)P(y)}{P(x)}$$

$$= \mathrm{argmax}_y P(x|y)P(y)$$

  - where $P(x)$ is a constant

Statistical Machine Translation

- Translation function:

$$\mathrm{argmax}_{\mathrm{y}} P(x|y)P(y)$$

- Meaning of two components:
  - $P(x|y)$
    - Translation model
    - How words and phrases should be translated (Learned from parallel data)
  - $P(y)$
    - Language model
    - How to write good English (Learned from monolingual data)

Statistical Machine Translation

- Translation function

$$\text{argmax}_y P(x|y)P(y)$$

- How to compute the argmax?
- Enumerate every possible y and calculate the probability
  - Too expensive!
- Answer: Use a heuristic search algorithm to gradually build up the the translation y

Heuristic Search Algorithm

- Example of translation from Chinese to English

布什　与　沙龙　举行　了　会谈

- Steps:

Heuristic Search Algorithm

- Example of translation from Chinese to English

布什 与 沙龙 举行 了 会谈

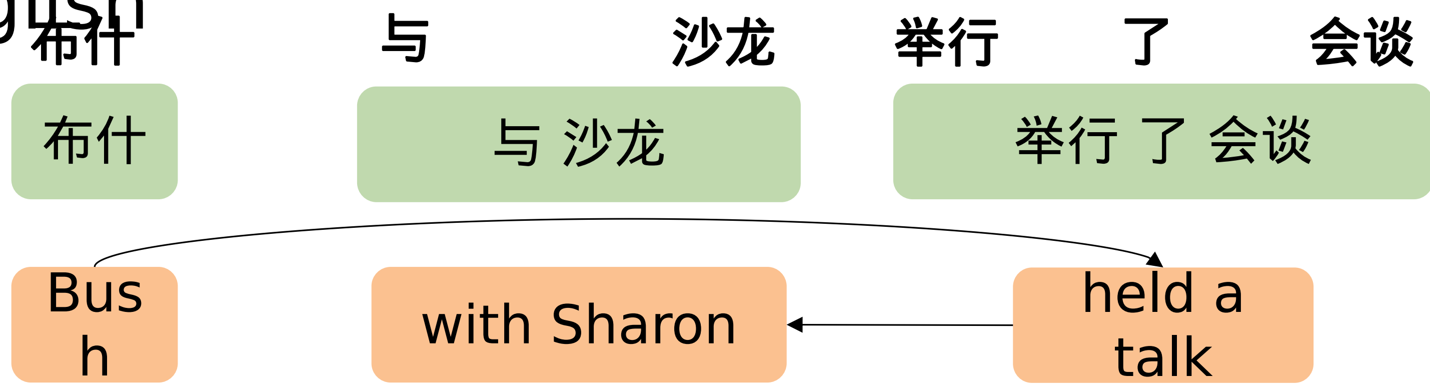
布什 | 与 沙龙 | 举行 了 会谈

Bus h | with Sharon | held a talk

- Steps:
  - Pick phrases in input and translate. Phrases may have multiple words.

Heuristic Search Algorithm

- Example of translation from Chinese to English

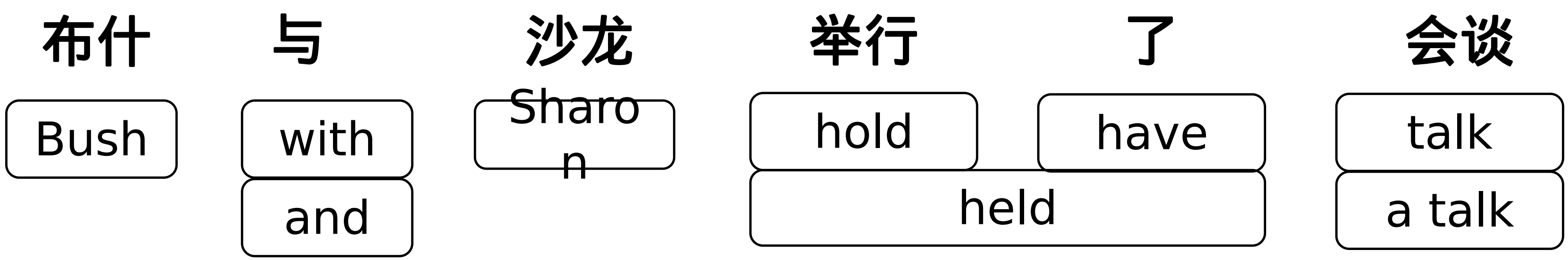


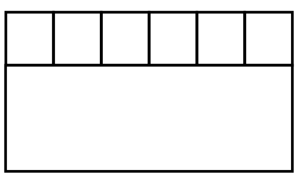
- Steps:
  - Pick phrases in input and translate. Phrases may have multiple words.
  - Allowed to pick phrases regardless of the original order.
  - Sentences with low probabilities are discarded.

Heuristic Search Algorithm

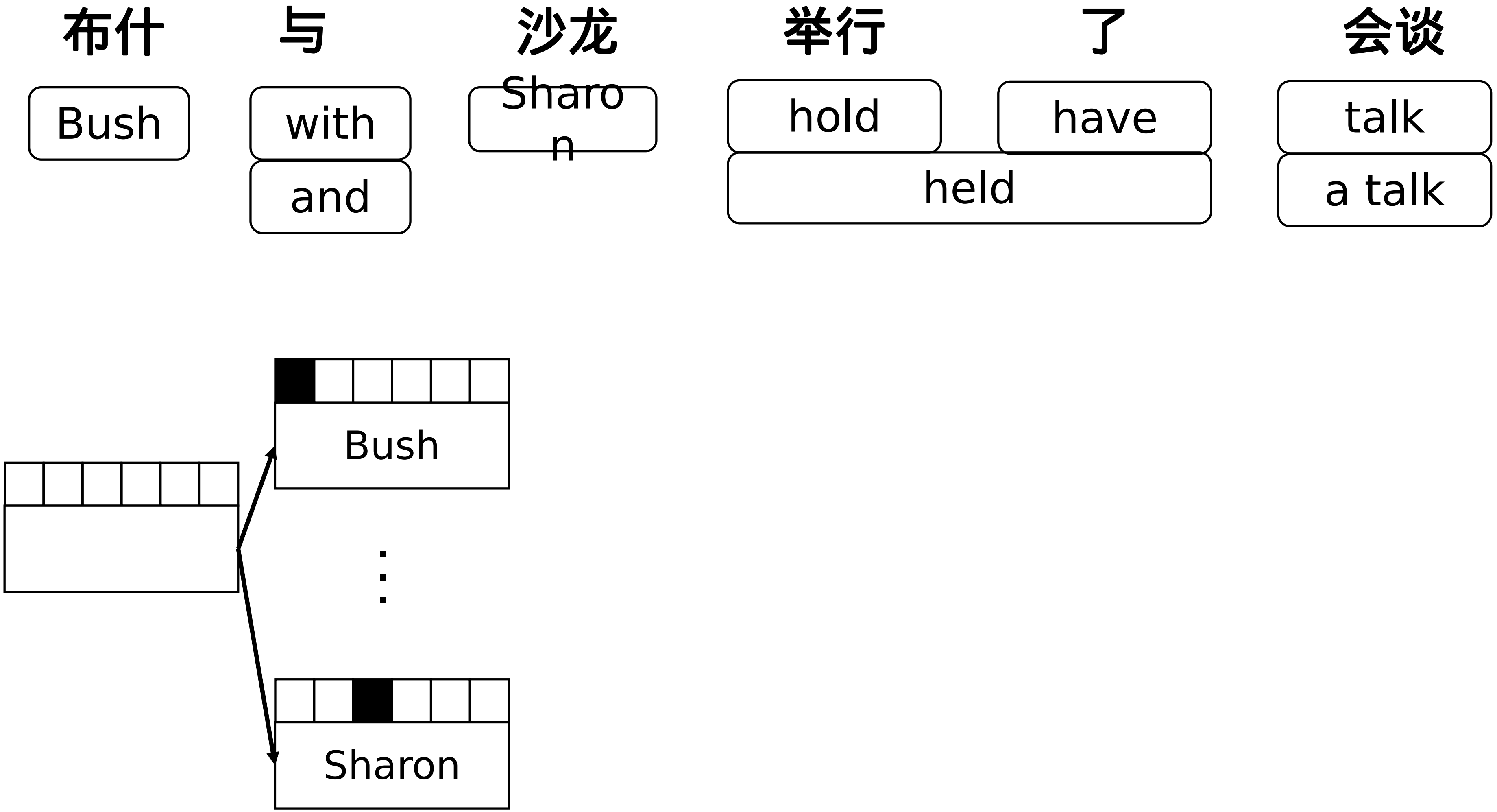
布什	与	沙龙	举行	了	会谈
Bush	with	Sharon	hold	have	talk
	and		held		a talk

Heuristic Search Algorithm

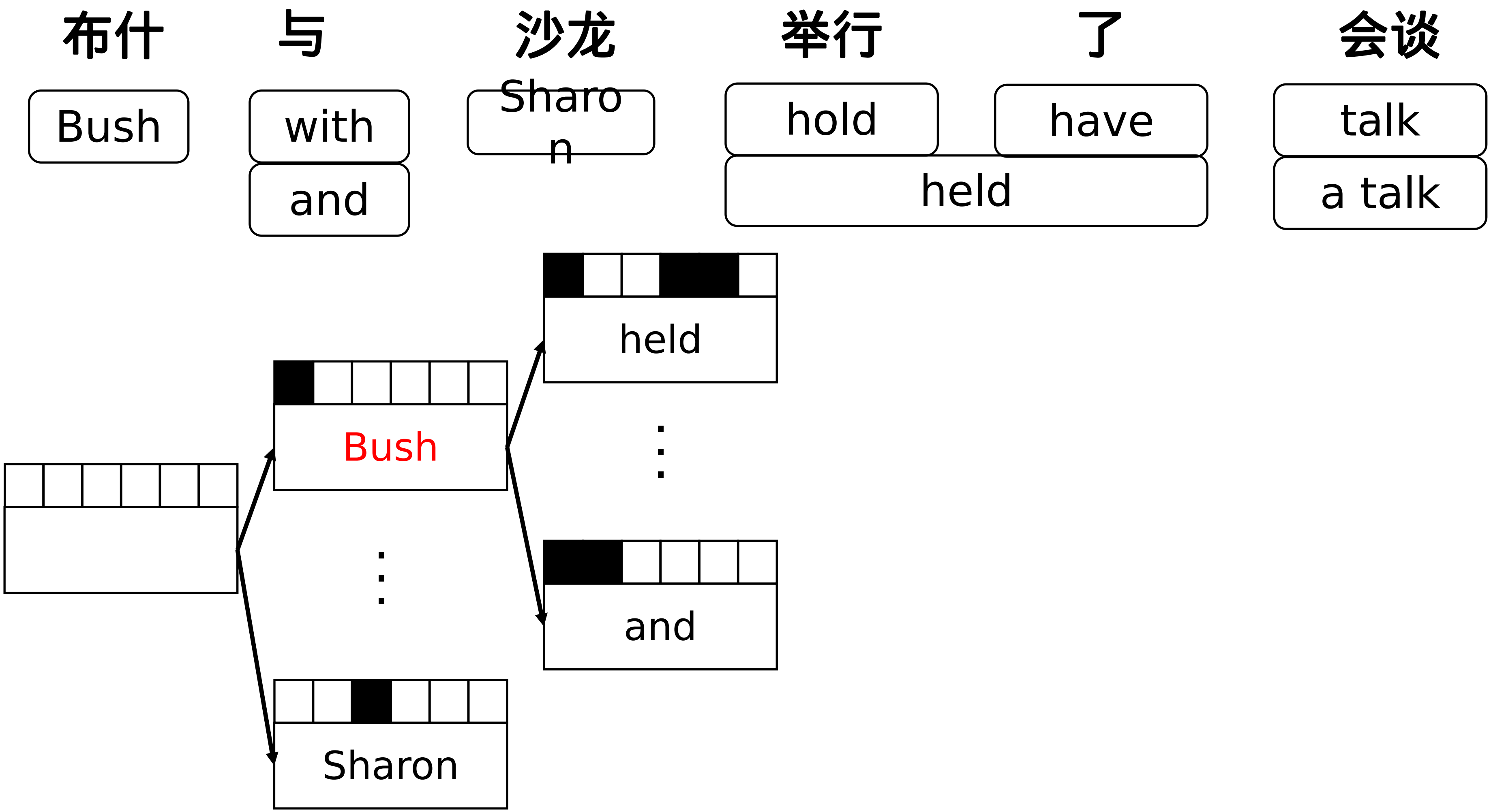




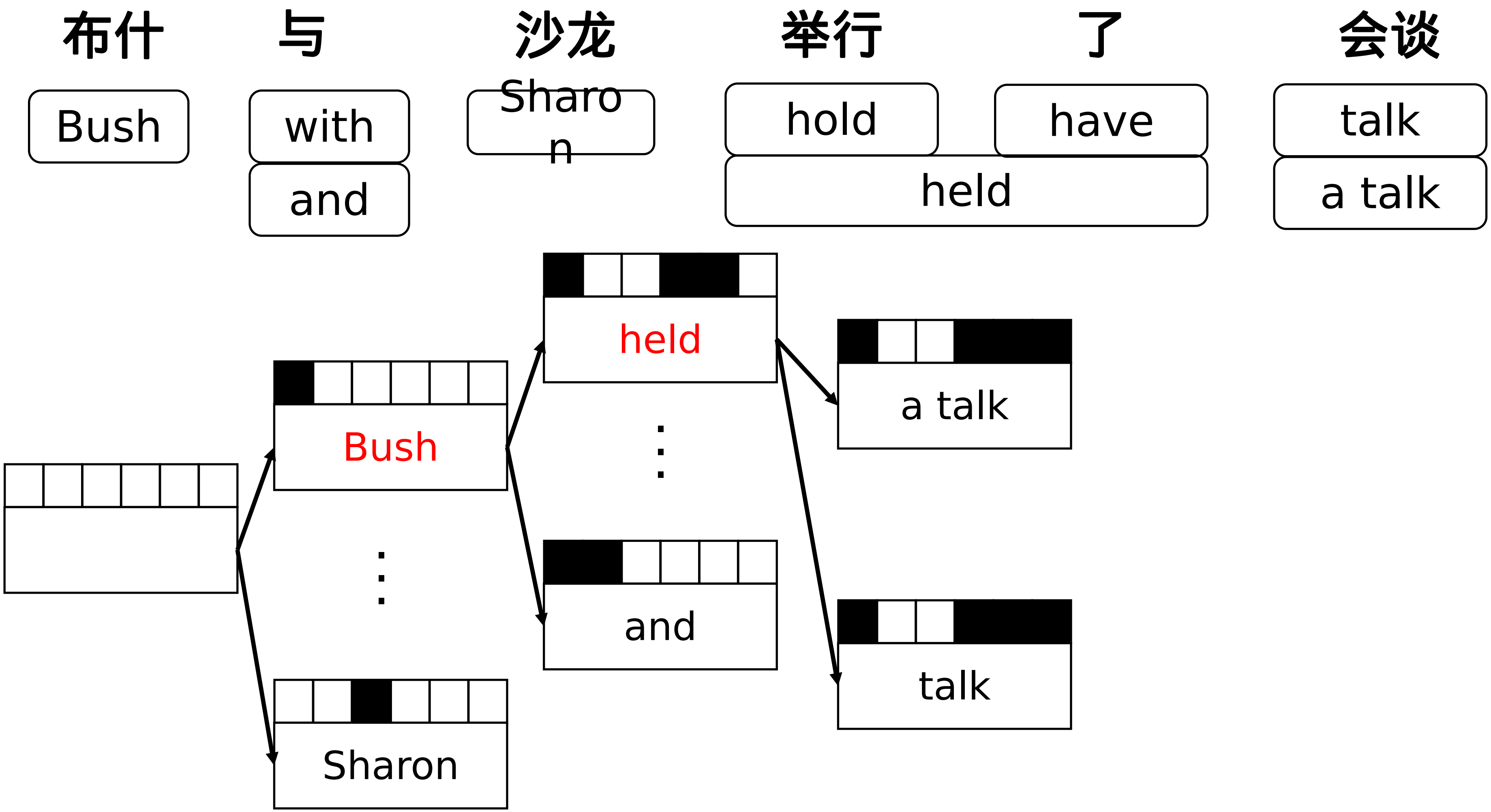
Heuristic Search Algorithm



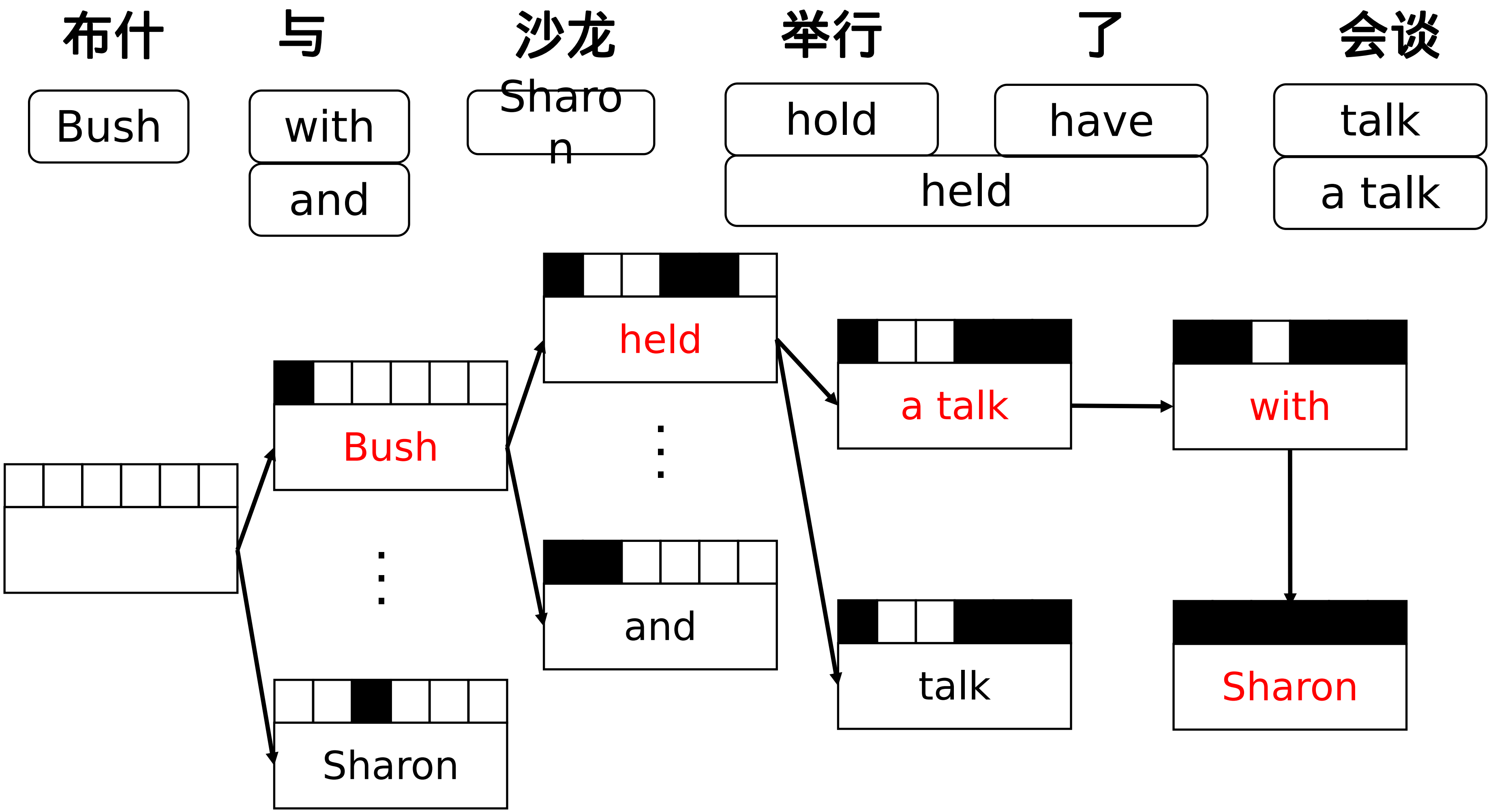
Heuristic Search Algorithm



Heuristic Search Algorithm



Heuristic Search Algorithm



Statistical Machine Translation

- The best systems are extremely complex
  - Hundreds of important details we have not mentioned here
  - Systems have many separately-designed sub-components
  - Lots of feature engineering
    - Need to design features to capture particular language phenomena
  - Lots of human efforts to maintain

Outline

- Introduction to Seq2Seq
- Machine Translation
  - Introduction
  - Statistical Machine Translation
  - Neural Machine Translation

Neural Machine Translation

- Neural Machine Translation (NMT): a way to conduct Machine Translation with a single neural network
- No separated language model and translation model (recall SMT)
- Neural network architecture: Seq2Seq architecture which involves two RNNs
- Recall RNN language model first!

RNNs for language modeling

output distribution

$y_4 = \mathrm{softmax}(U h_4 + b_2) \in \mathbb{R}^{|V|}$

hidden states

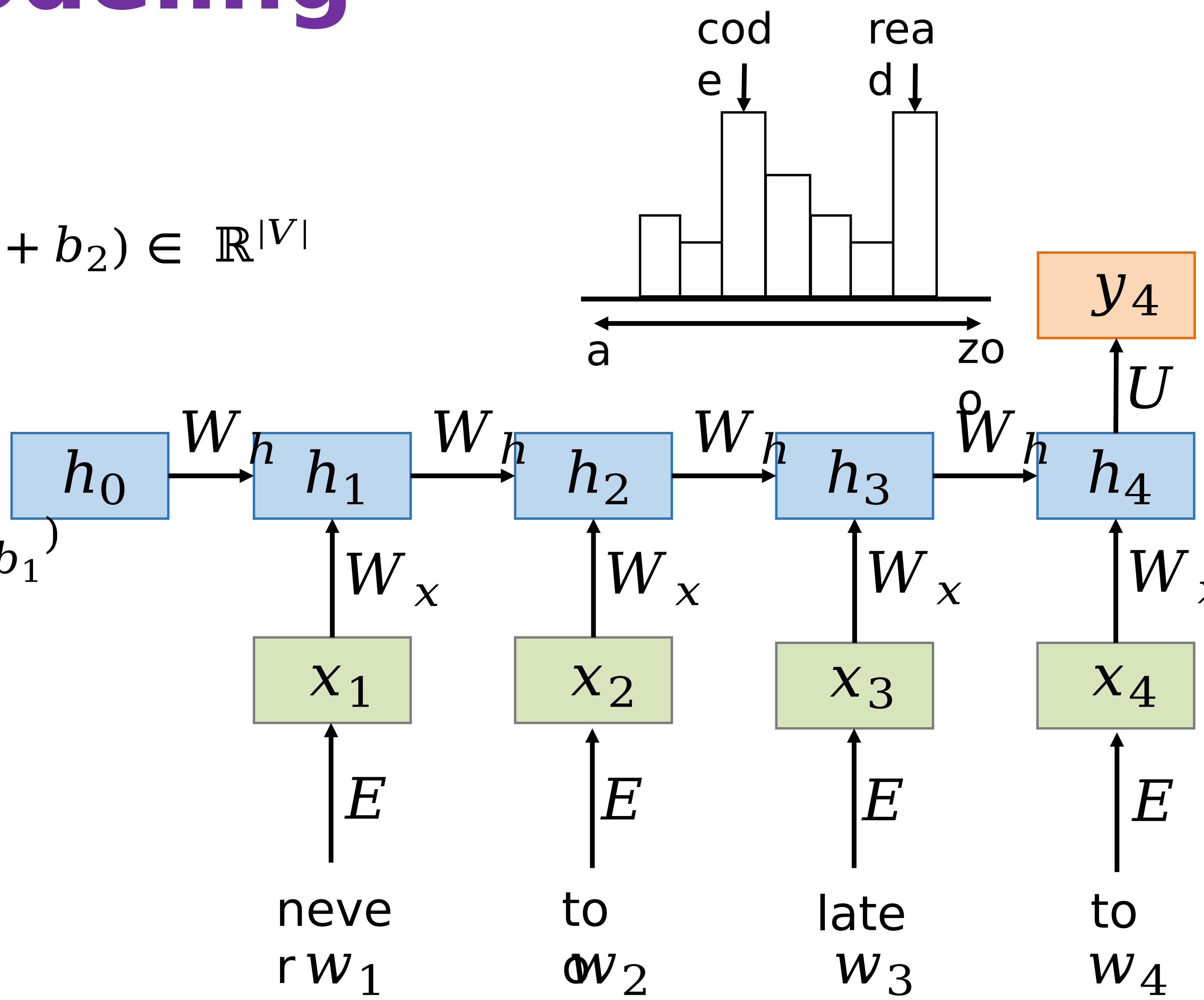
$h_i = \tanh(W_x x_i + W_h h_{i-1} + b_1)$

word embeddings

$x_i = E w_i$

one-hot vectors

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



RNNs for language modeling

- How does the RNN cell work?
- RNN cell takes the current RNN state and a word vector and produces a subsequent RNN state that encodes the sentence so far

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

- Learned weights represent how to combine past information h_{i-1} and current information x_i

RNNs for language modeling

- How does the output function work?

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

- y_4 is a probability distribution over the vocab constructed from the RNN memory and the transformation (U, b_2)
- Softmax function turns scores into a probability distribution

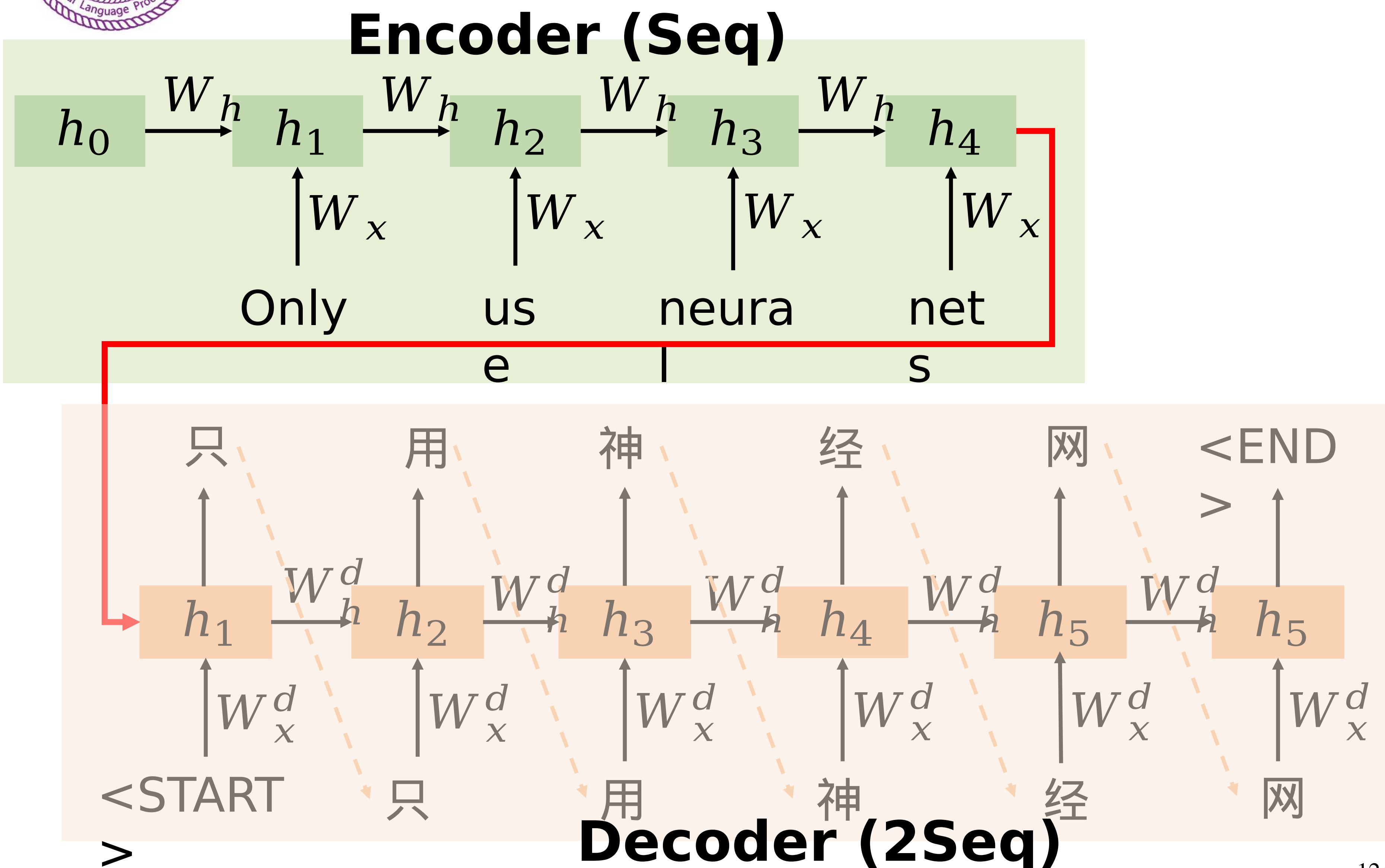
RNNs for Encoding

- Predict things other than next word:
  - POS Tagging
  - Named Entity Recognition
  - Sentiment Classification
  - Relation Classification
- RNNs are good at modeling sequential information
- General idea: Use RNN as an encoder for building the semantic representation of the sentence

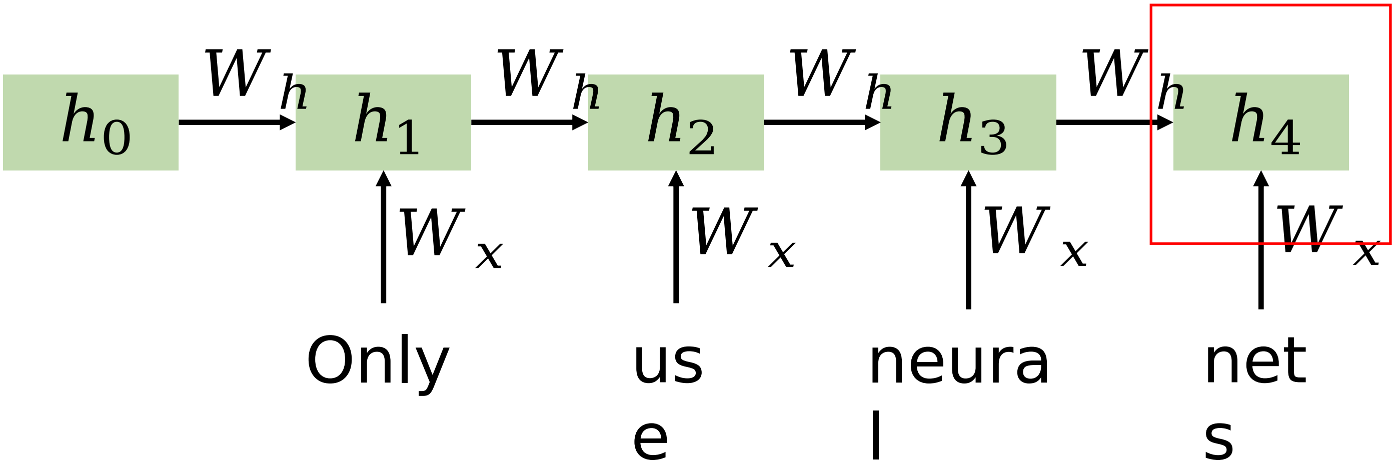
Seq2Seq in MT

- Recall the sequence-to-sequence model
- Two RNNs
  - Encoder RNN
  - Decoder RNN
- Encoder RNN: produces a representation of the source sentence
- Decoder RNN: a language model that generates target sentence conditioned on encoding

Seq2Seq in MT

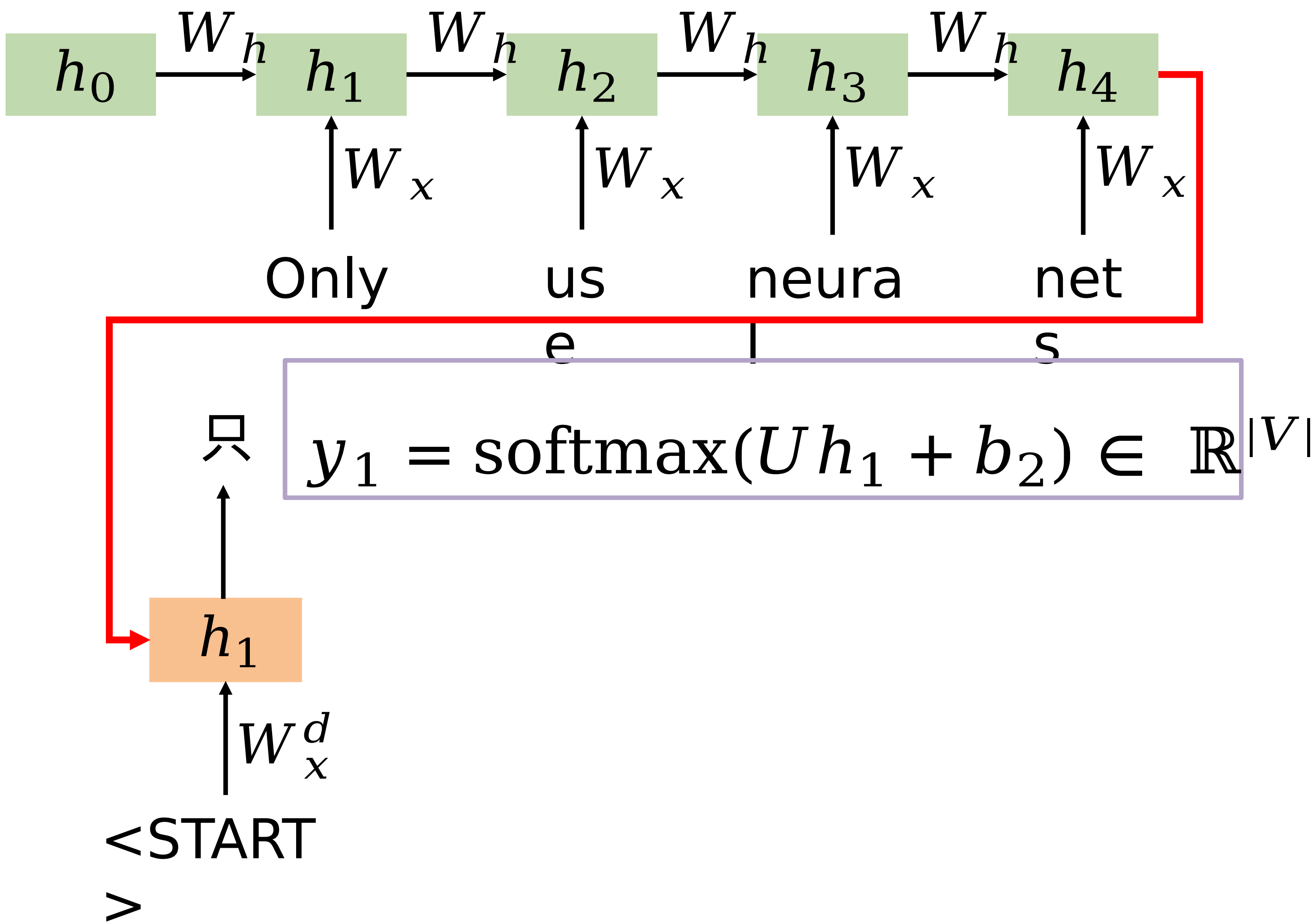


Seq2Seq in MT

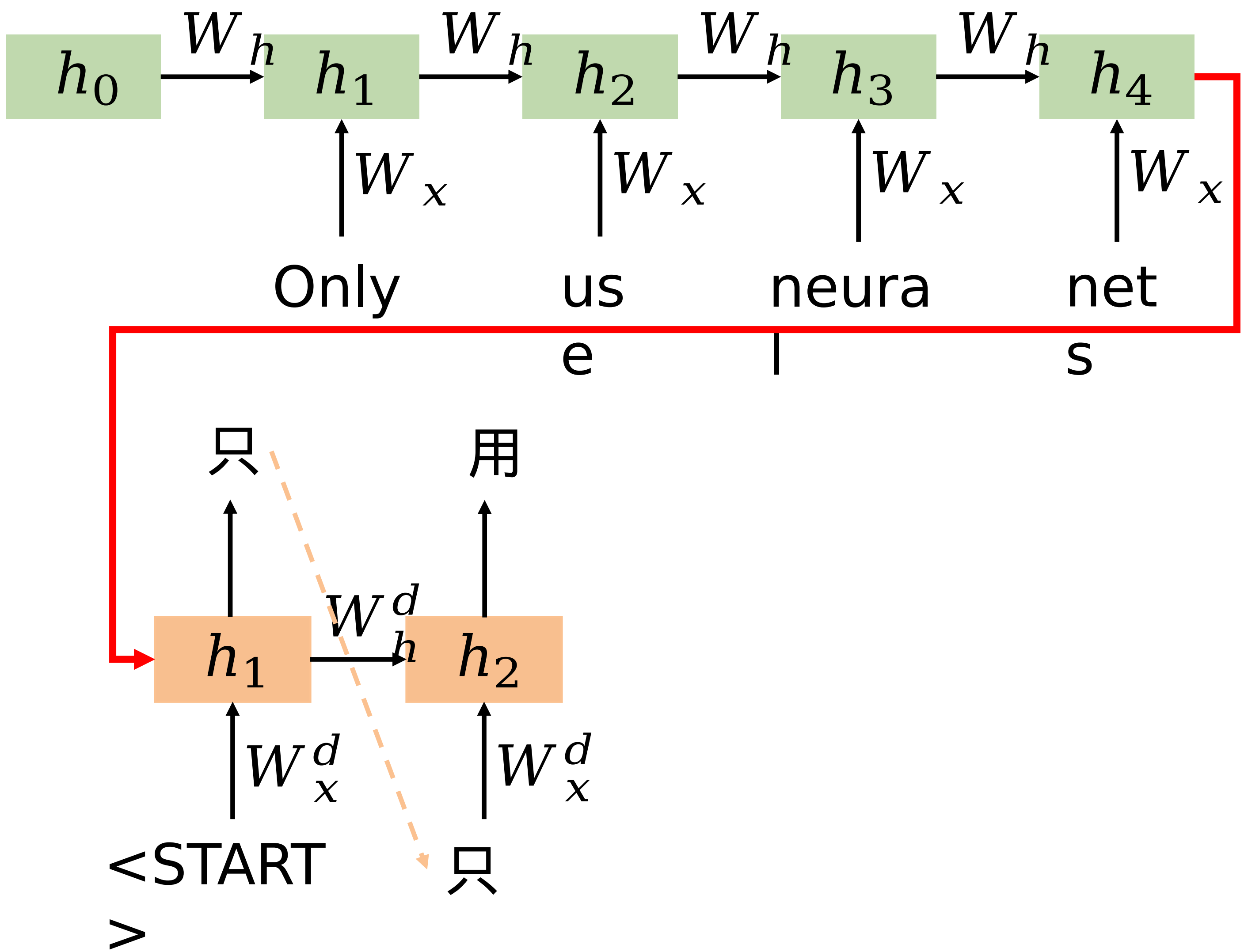


- h_4 is the representation of the source sentence and is provided as the initial hidden state for Decoder RNN

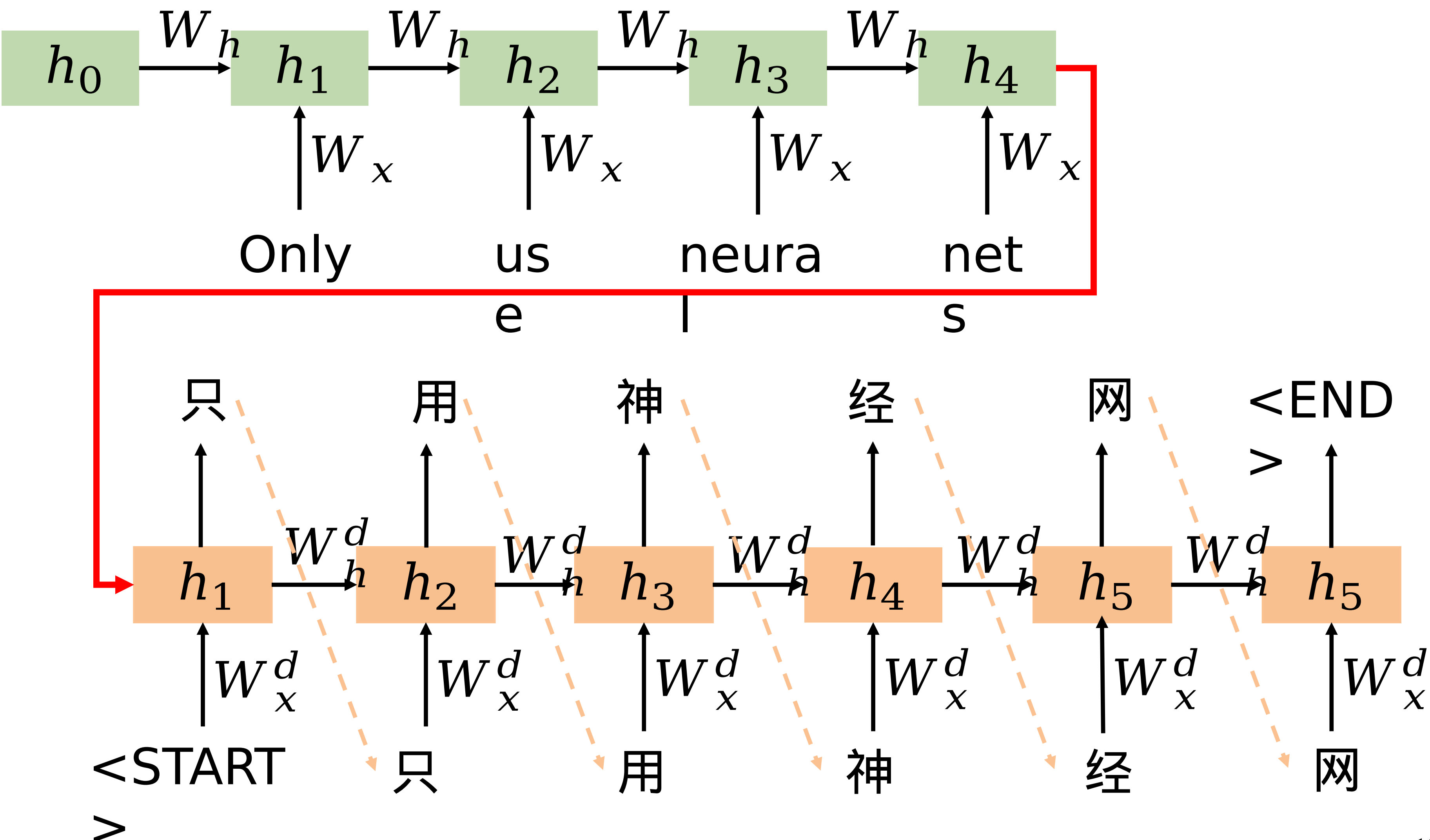
Seq2Seq in MT



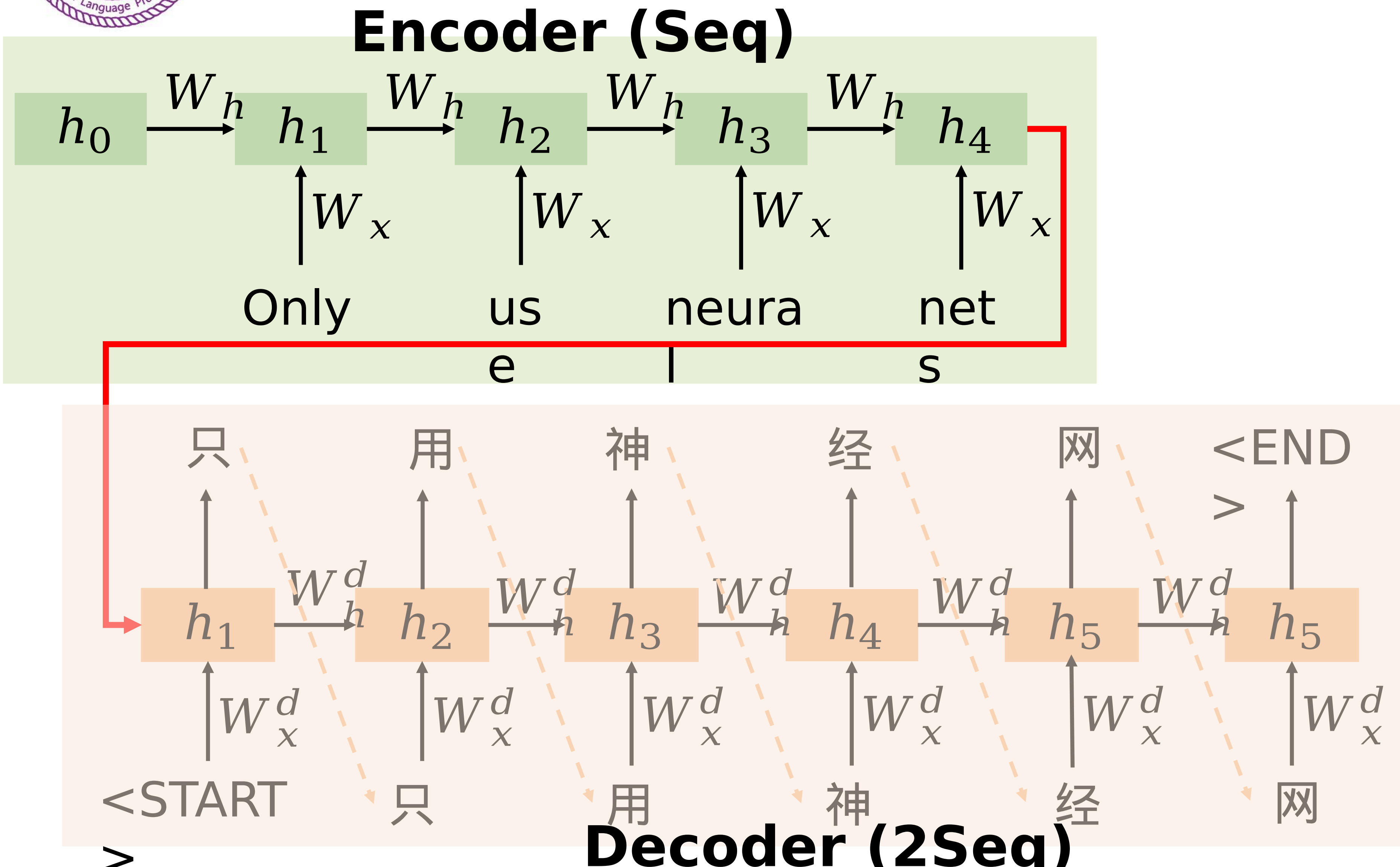
Seq2Seq in MT



Seq2Seq in MT



Seq2Seq in MT



Seq2Seq in MT

- Seq2seq model is an example of a conditional language model
  - Language model: the decoder is predicting the next word of the target sentence y
  - Conditional: its predictions are also conditioned on the source sentence x
- $P(y|x)$ in NMT

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)...P(y_T|y_1, ..., y_{T-1}, x)$$

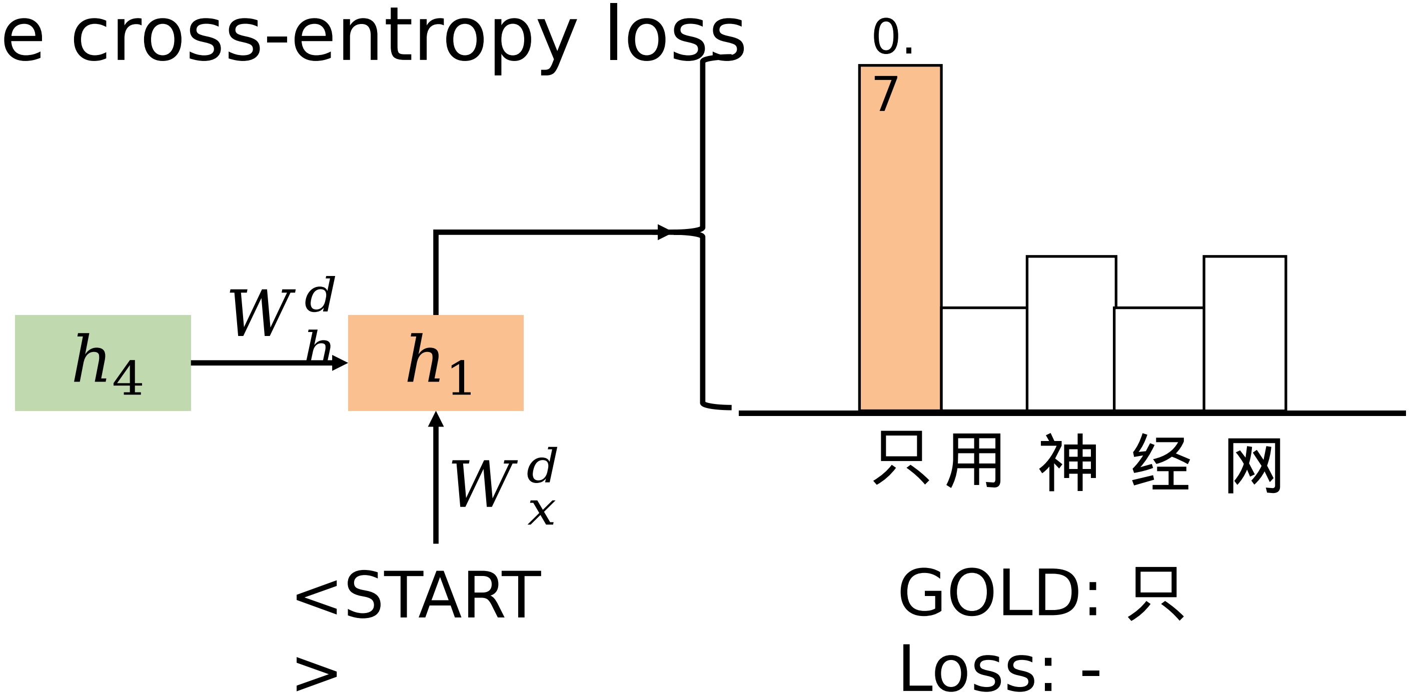
- Different from SMT: $P(x|y)P(y)$
- More direct!

Outline

- Introduction to Seq2Seq
- Machine Translation
  - Introduction
  - Statistical Machine Translation
  - Neural Machine Translation
    - Training Seq2Seq
    - Decoding Strategy
    - Evaluation of MT
    - Summary

Training Seq2Seq

- Force the decoder to generate gold sequence
- Sum of losses for each token as the objective function
- Use cross-entropy loss



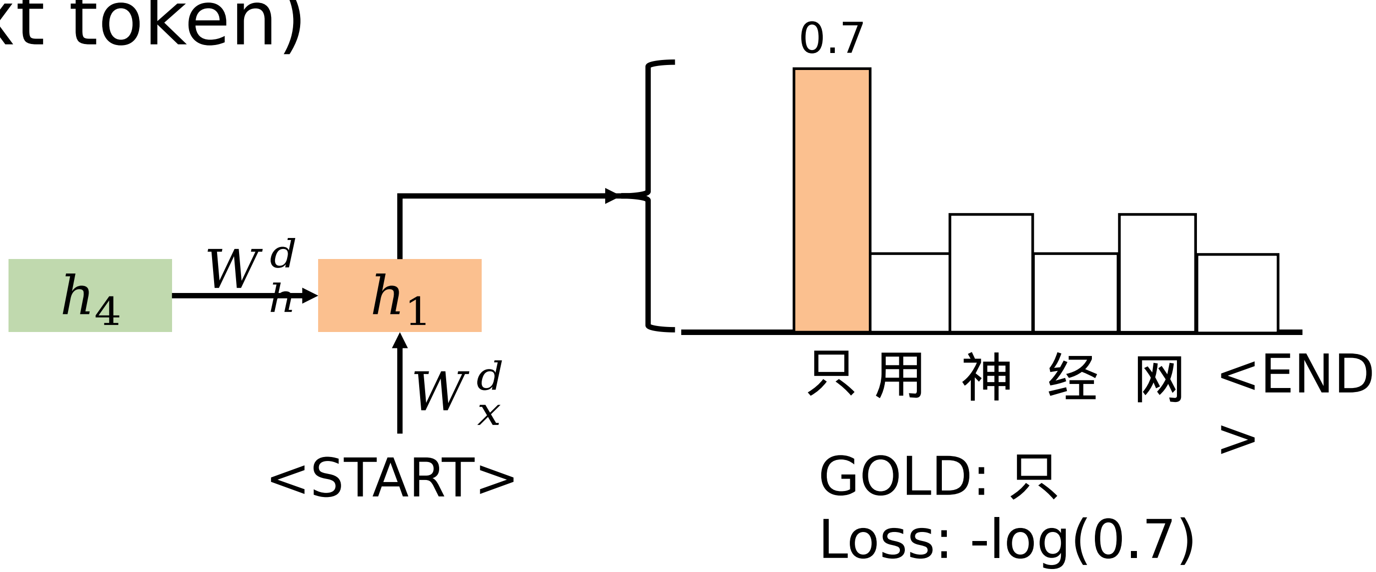
GOLD: 只
Loss: -log(0.7)

Training Seq2Seq

- Cross-entropy loss

$$-\sum_{x} p(x)\log q(x)$$

- $q(x)$: the distribution produced by the network
- $p(x)$: the true distribution (1 on the actual next token)

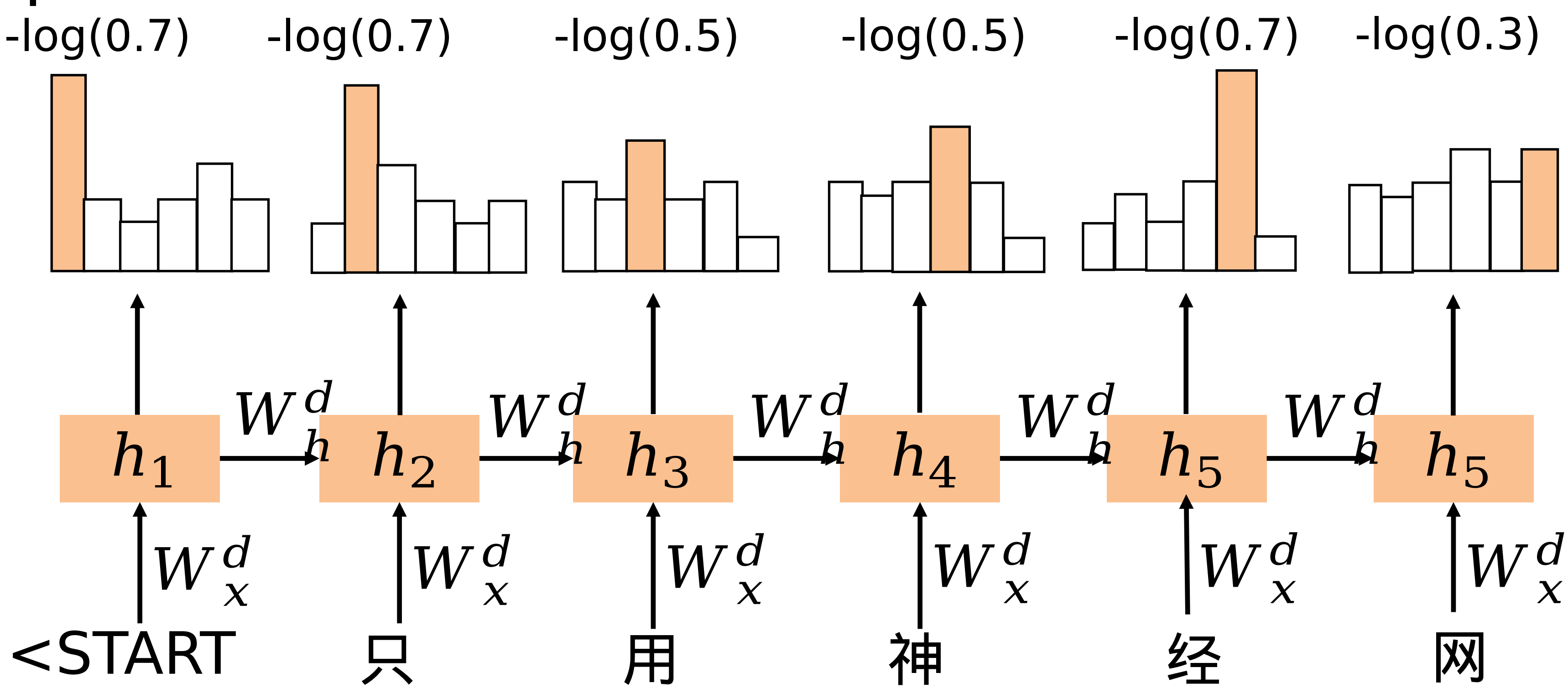


Training Seq2Seq

- Sum losses of each token for sentence loss J

$$J = \mathrm{sum}(-3 * \log(0.7) - 2 * \log(0.5) - \log(0.3))$$

- Minimize the loss J for the given sentence pair



Training Seq2Seq

- Notice!
- Seq2seq is optimized as a unified system
- Backpropagation operates end-to-end
- Update two RNNs simultaneously
- Model parameters include word embeddings!

Outline

- Introduction to Seq2Seq
- Machine Translation
  - Introduction
  - Statistical Machine Translation
  - Neural Machine Translation
    - Training Seq2Seq
    - Decoding Strategy
    - Evaluation of MT
    - Summary

Vanilla Decoder Strategy

- Recall the decoding process

$$\mathrm{argmax}_{\mathrm{y_i}} P(y_i | y_1, \ldots, y_{i-1}, x)$$

- Generate the target sentence by taking argmax on each step of the decoder
- Greedy decoding
- Recall the original translation model

$$\mathrm{argmax}_{\mathrm{y}} P(y | x)$$

- Can the greedy decoding always generate the best y?

Vanilla Decoder Strategy

- In this problem, a greedy strategy does not usually produce a globally optimal solution
- Problem: Greedy decoding has no way to undo decisions!

  布什与沙龙举行了会谈 (Bush held a talk with Sharon)

  →Bush

  →Bush and

  →Bush and Sharon

Beam Search Decoding

- Ideally, we want to find y that maximizes

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)...P(y_T|y_1, ..., y_{T-1}, x)$$

- We could try enumerating all y
  - Complexity $O(V^T)$ where V is vocab size and T is target sequence length → too expensive!

- Beam Search: On each step, keep track of the k most probable partial translations

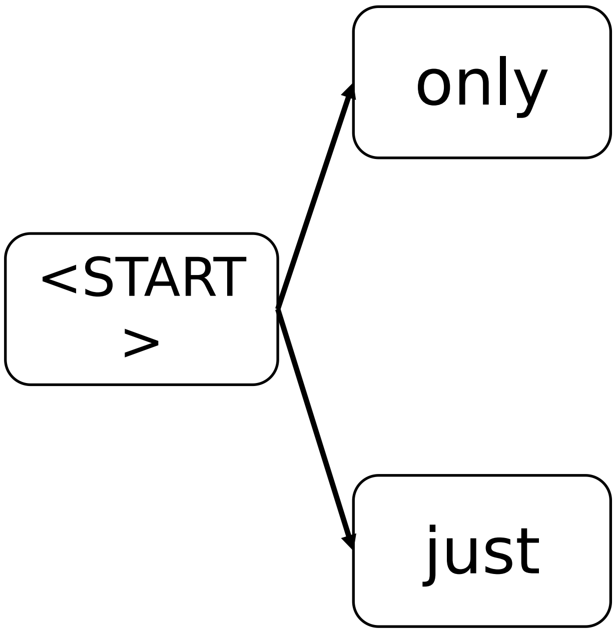
Beam Search Decoding

- Beam Search: On each step, keep track of the k most probable partial translations
- k is the beam size (in practice around 5 to 10)
- Also not guarantee to produce a globally optimal solution
- But results are much more applicable!
- Example:

  只用神经网→ Only use neural nets

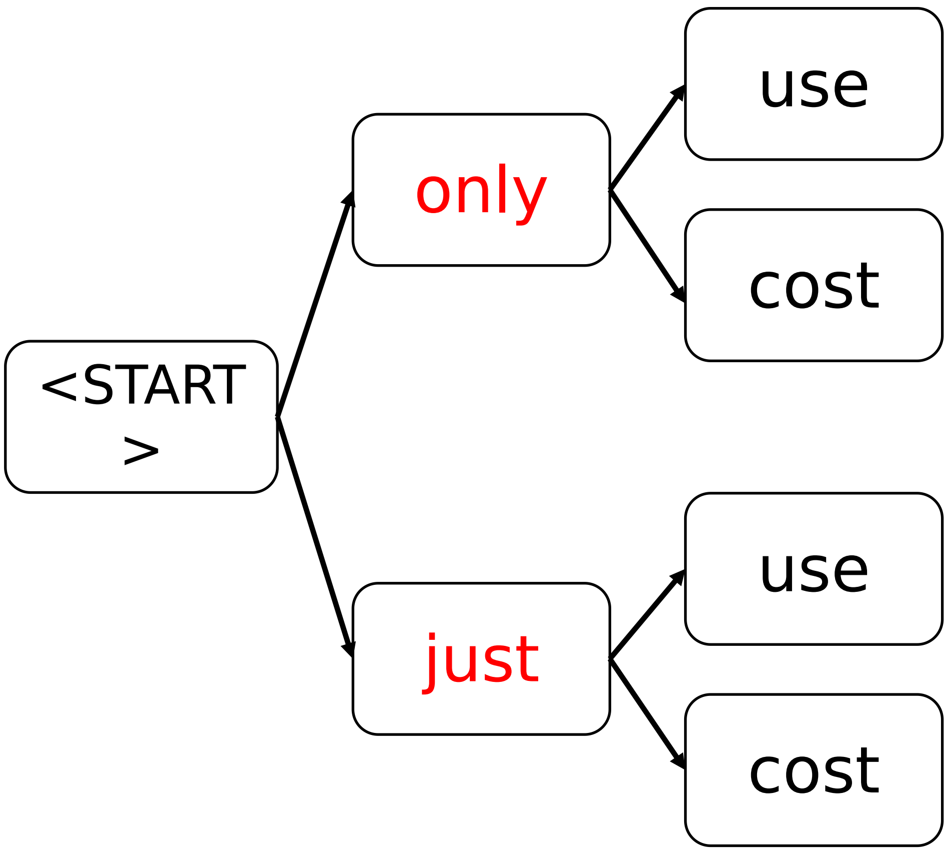
Beam Search Decoding

- Example (beam size = 2)



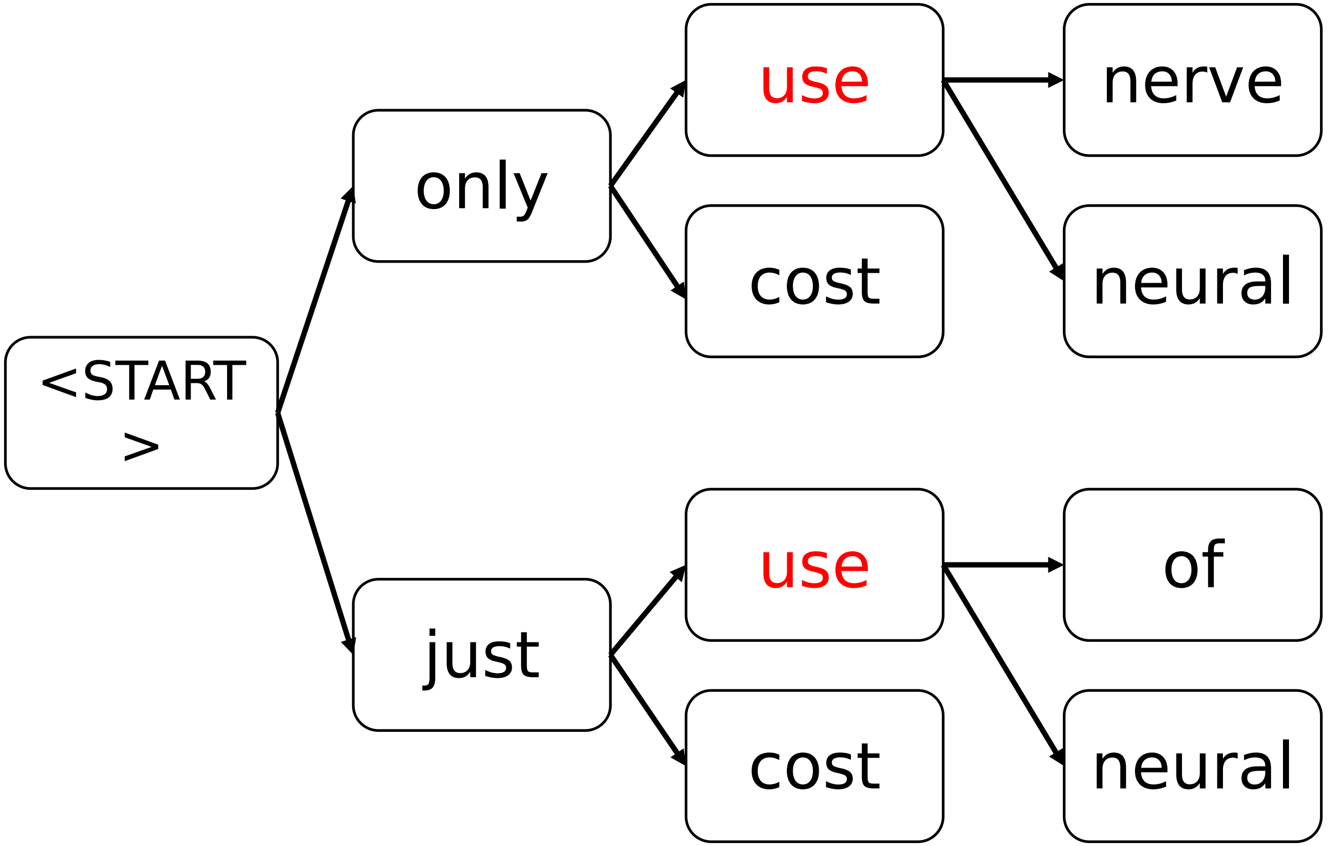
Beam Search Decoding

- Example (beam size = 2)



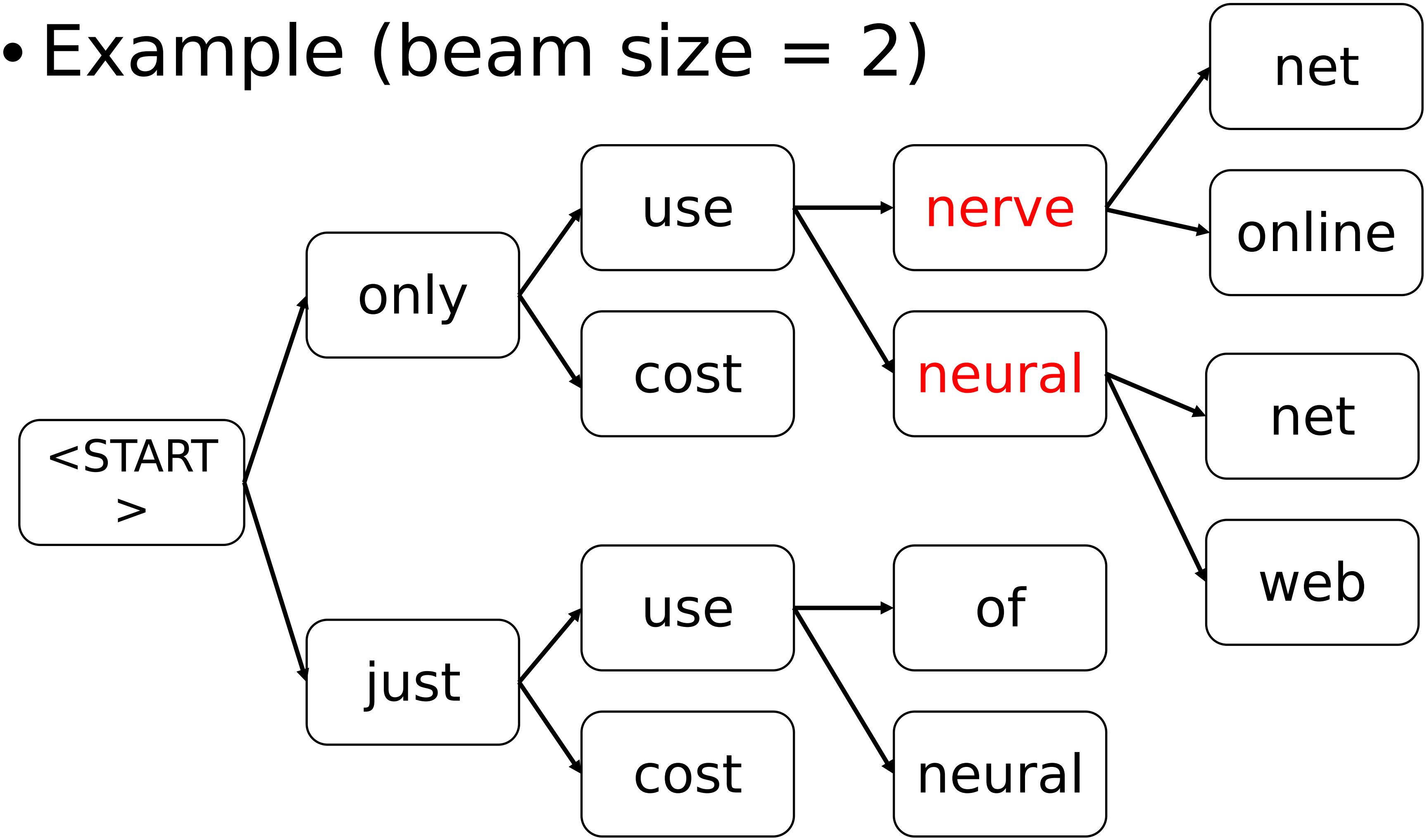
Beam Search Decoding

- Example (beam size = 2)



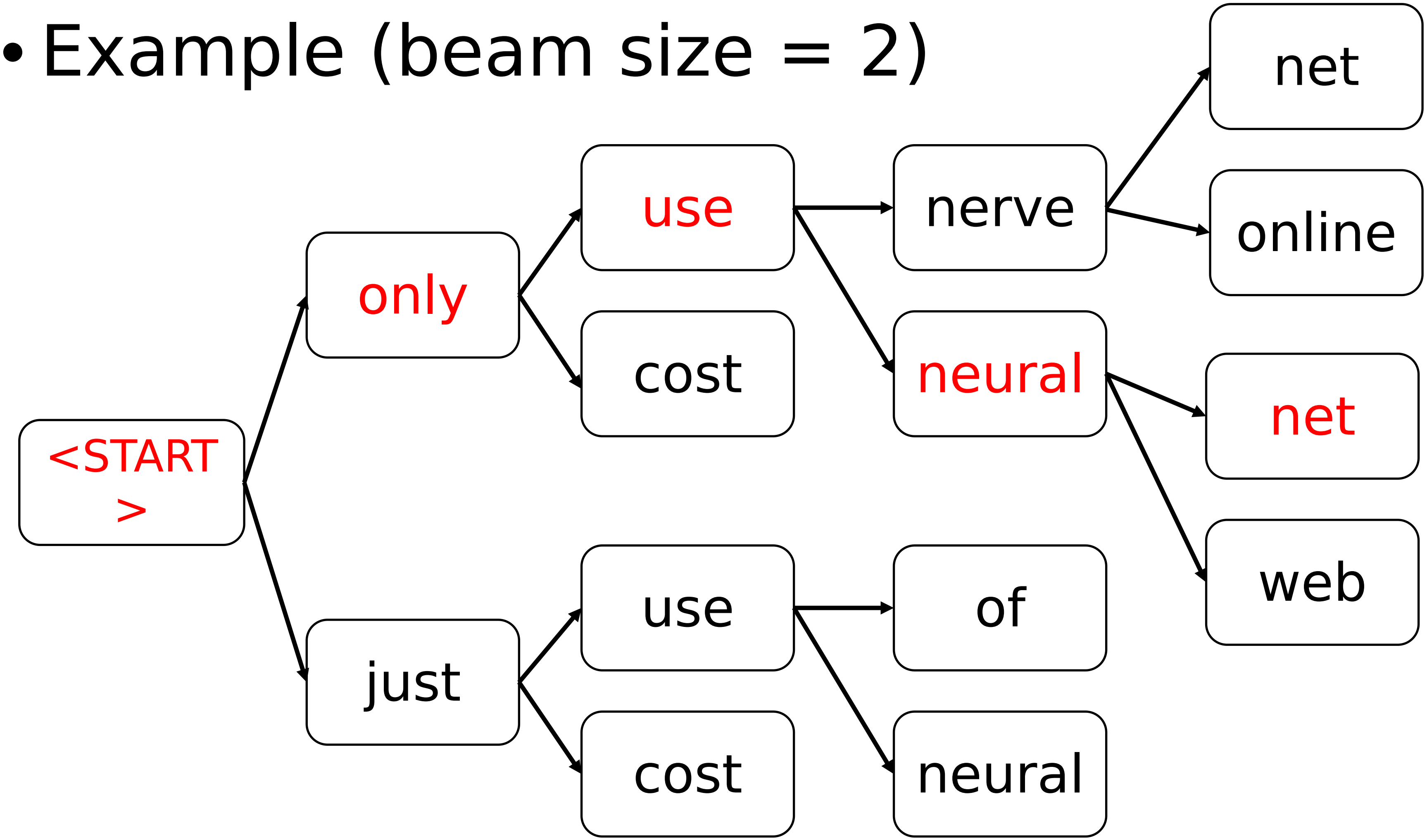
Beam Search Decoding

- Example (beam size = 2)



Beam Search Decoding

- Example (beam size = 2)



Outline

- Introduction to Seq2Seq
- Machine Translation
  - Introduction
  - Statistical Machine Translation
  - Neural Machine Translation
    - Training Seq2Seq
    - Decoding Strategy
    - Evaluation of MT
    - Summary

Evaluation

- **BLEU** (**Bil**ingual **E**valuation **U**nderstudy)
- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - N-gram precision
  - Penalty for too-short system translations

$$BLEU = \mathrm{BP} * \exp\left(\sum_{i=1}^{N} w_i log P_i\right)$$

Evaluation

- **BLEU** (**Bil**ingual **E**valuation **U**nderstudy)

$$BLEU = \mathrm{BP} * \exp\left(\sum_{i=1}^{N} w_i log P_i\right)$$

  - P_i: The i-gram precision
  - BP: Brevity penalty

$$BP = \begin{cases} 1, & l_c > l_r \\ e^{1-\frac{l_r}{l_c}}, & l_c \leq l_r \end{cases}$$

    - l_c: length of the candidate, l_r: length of the reference.

  - Usually we set $N = 4,\ w_i = \frac{1}{4}$

Evaluation

- BLEU example
- Candidate: Airport security Israeli officials are responsible.
- Reference: Israeli officials are responsible for airport security.
  - 1-gram precision: $P_1 = \frac{6}{6}$
  - 2-gram precision: $P_2 = \frac{4}{5}$
  - 3-gram precision: $P_3 = \frac{2}{4}$
  - 4-gram precision: $P_4 = \frac{1}{3}$

Evaluation

- BLEU example

$$P_1 = \frac{6}{6}, P_2 = \frac{4}{5}, P_3 = \frac{2}{4}, P_4 = \frac{1}{3}$$

- The calculation

$$\exp\left(\frac{1}{4}\ \left(\log(1) + \log\left(\frac{4}{5}\right) + \log\left(\frac{2}{4}\right) + \log\left(\frac{1}{3}\right)\right)\right) = 0.386$$

- Considering the brevity penalty

$$BLEU = e^{1-7/6} * 0.386 = 0.327$$

Evaluation

- **BLEU** (**Bi**lingual **E**valuation **U**nderstudy)
- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - Sometime a good translation can get a poor BLEU score because it has low n-gram overlap with human translation

Reference: I ate the apple.

Candidates:
1. I consumed the apple.
2. I ate an apple.
3. I ate the potato.

Outline

- Introduction to Seq2Seq
- Machine Translation
  - Introduction
  - Statistical Machine Translation
  - Neural Machine Translation
    - Training Seq2Seq
    - Decoding Strategy
    - Evaluation of MT
    - Summary

Advantages of NMT

- Compared to SMT, NMT has many advantages:
  - Better performance
    - More fluent
    - Better use of context
    - Better use of phrase similarities
  - A single neural network to be optimized end-to-end
    - No subcomponents to be individually optimized
  - Requires much less human engineering effort
    - No feature engineering
    - Same method for all language pairs

Disadvantages of NMT

- Compared to SMT:
- NMT is less interpretable
  - Hard to debug
- NMT is difficult to control
  - For example, cannot easily specify rules or guidelines for translation
  - Safety concerns!

Summary

- Seq2Seq can use RNN/GRU/LSTM/Transformer to perform encoder-decoder based tasks
  - Training method, decoding strategy, evaluation
  - Various applications
- Machine translation
  - RBMT, EBMT, SMT, NMT
- Attention is critical for improving seq2seq models
- Transformer is powerful in sequence modeling

Reading Material

a. Machine Translation

- **Must-read Papers**

· The Mathematics of Statistical Machine Translation: Parameter Estimation. Peter EBrown, Stephen ADella Pietra, Vincent JDella Pietra, and Robert LMercer. Computational Linguistics 1993 [link]

· (Seq2seq) Sequence to Sequence Learning with Neural Networks. Ilya Sutskever, Oriol Vinyals, and Quoc VLe. NIPS 2014 [link]

· (BLEU) BLEU: a Method for Automatic Evaluation of Machine Translation. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. ACL 2002 [link]

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

Reading Material

a. Machine Translation

- **Further Reading**

· Statistical Phrase-Based Translation. Philipp Koehn, Franz JOch, and Daniel Marcu. NAACL 2003 [link]

· Hierarchical Phrase-Based Translation. David Chiang. Computational Linguistics 2007 [link]

· (Beam Search) Beam Search Strategies for Neural Machine Translation. Markus Freitag and Yaser Al-Onaizan. 2017 [link]

· MT paper list. [link]

· THUMT toolkit. [link]

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

Q&A

THUNLP