CS182: Introduction to Machine Learning – RNN LMs + Transformer LMs

Yujiao Shi
SIST, ShanghaiTech
Spring, 2025

Introduction

- Our goal: Build intelligent algorithms to make sense of data
  - Example: Recognizing objects in images

red panda (*Ailurus fulgens*)

  - Example: Predicting what would happen next

Vondrick et al. CVPR2016

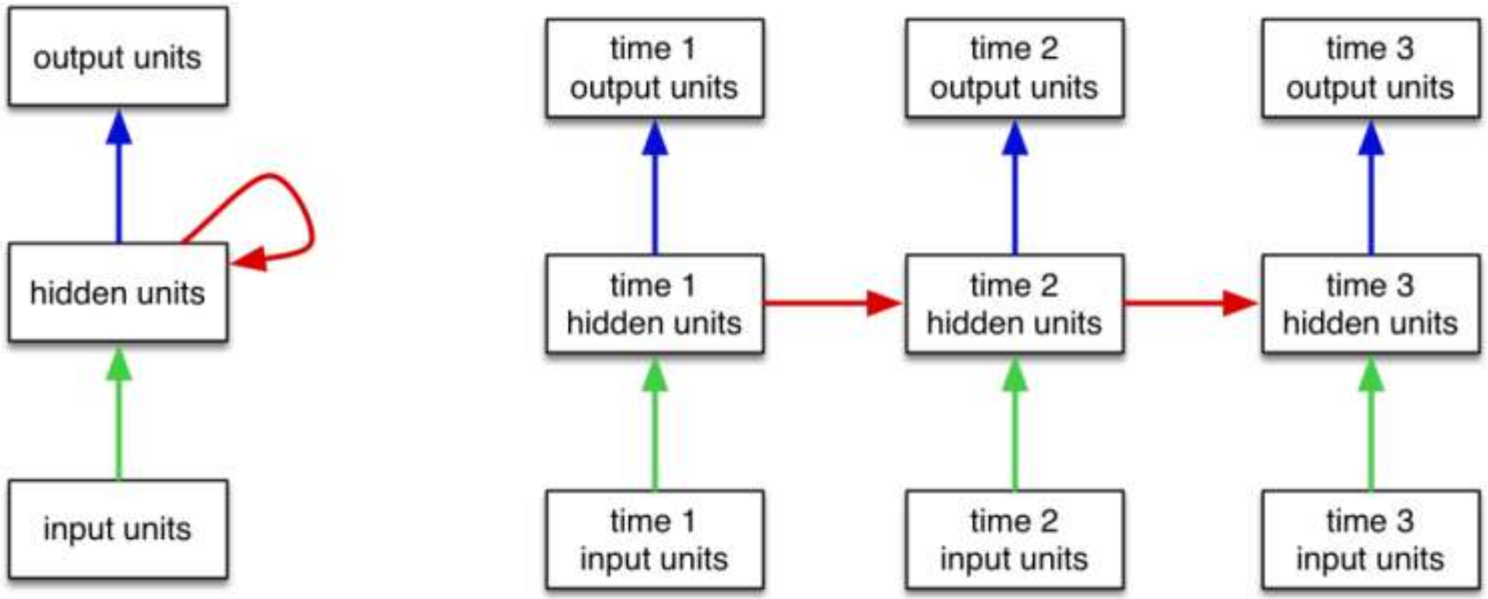
Introduction

- Our goal: Build intelligent algorithms to make sense of data
  - Example: Recognizing objects in images
  - Example: Predicting what would happen next

Given an initial still frame,

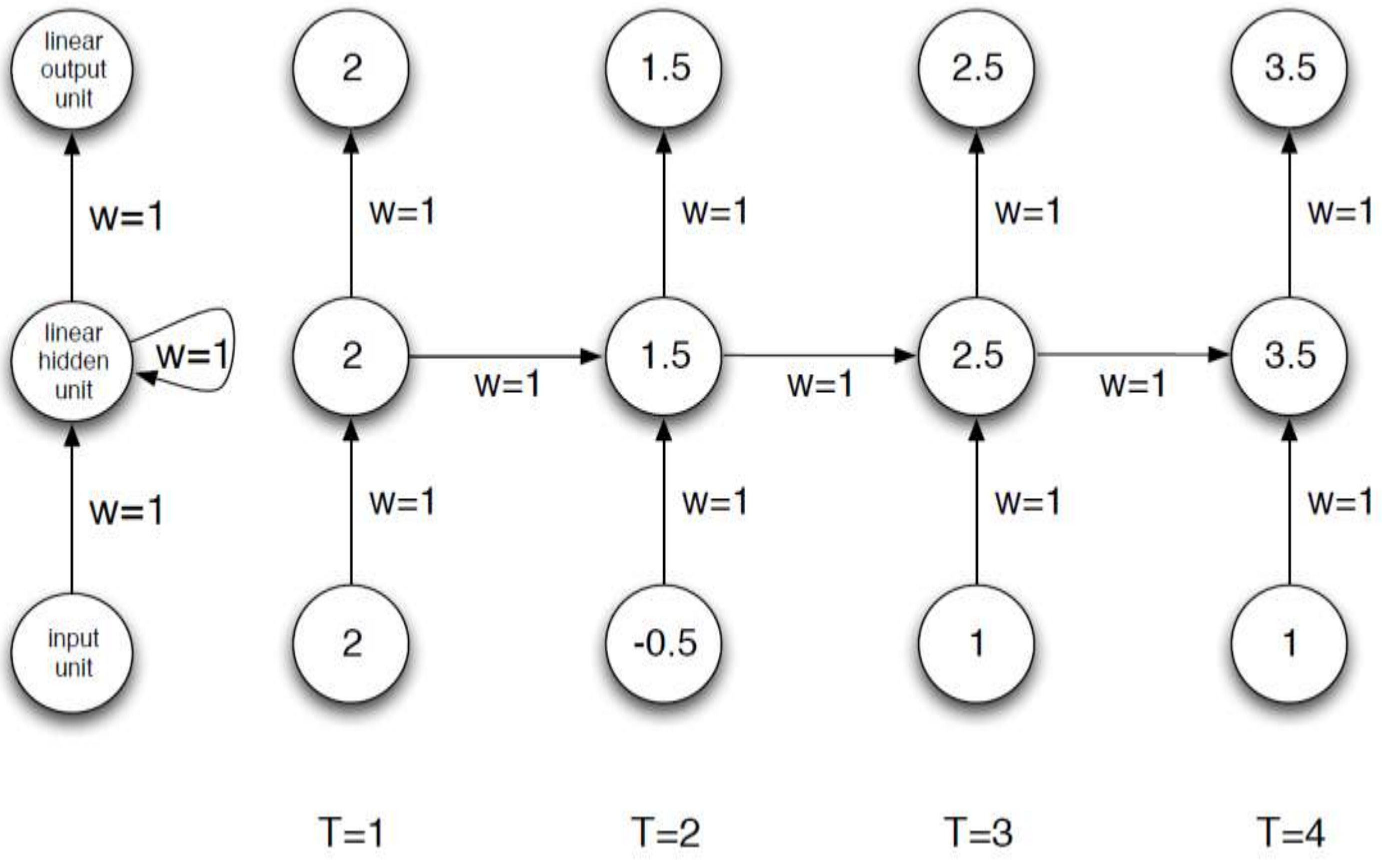
Recurrent Neural Network

- Recurrent Neural Network as a dynamical system with one set of hidden units feeding into themselves
  - The network’s graph has self-loops
- The RNN’s graph can be unrolled by explicitly representing the units at all time steps
  - The weights and biases are shared



RNN examples

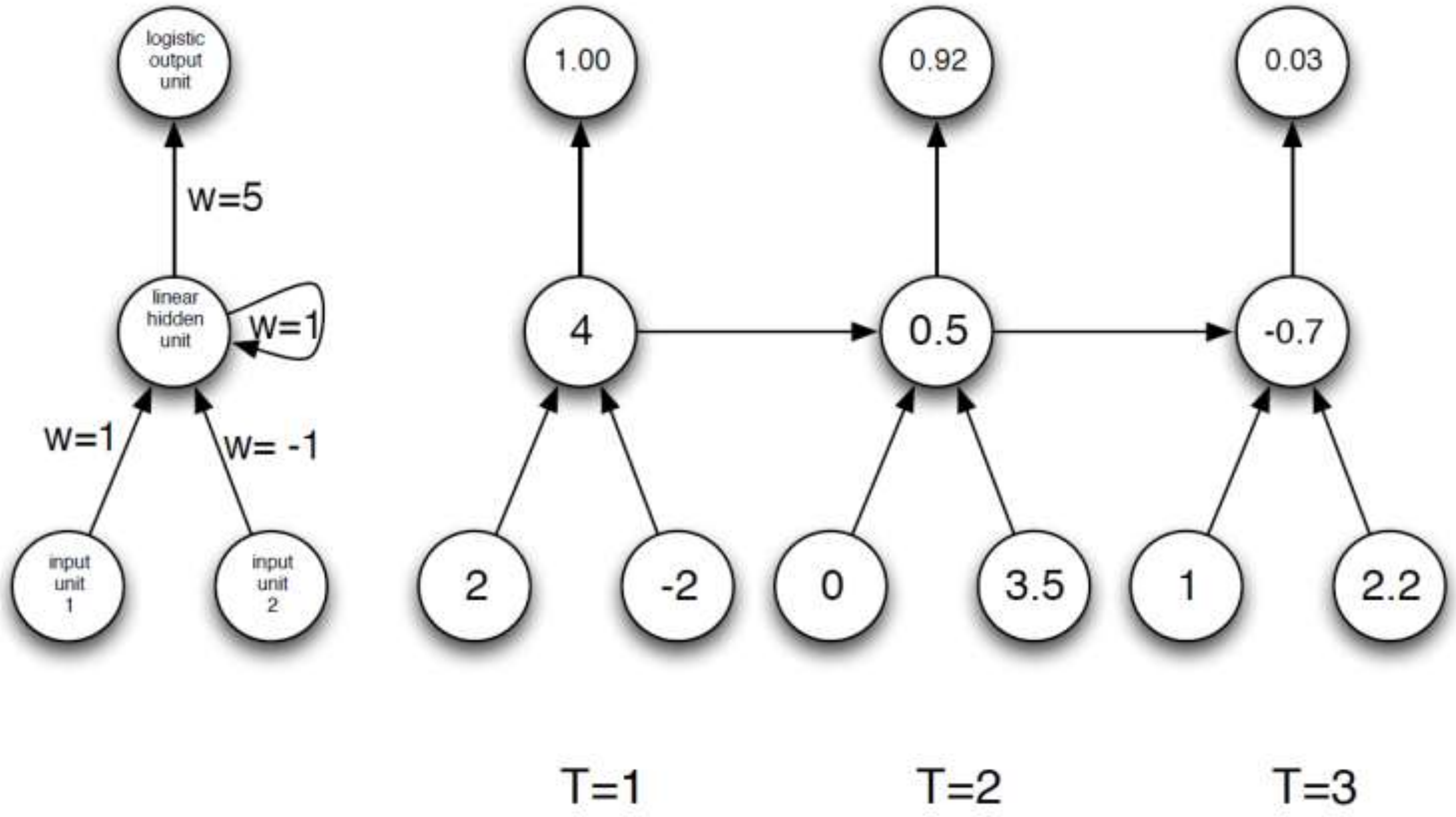
- Summation network



RNN examples

上海科技大学 ShanghaiTech University

- Summation & comparison network



Recurrent Neural Network

- General formulation

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

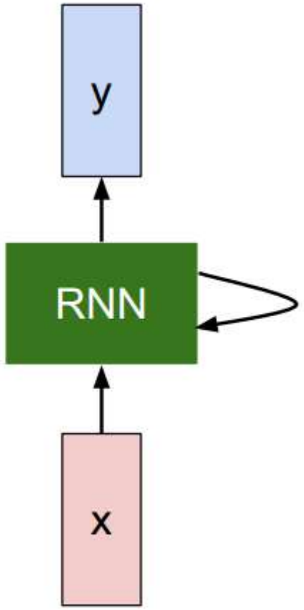
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W



Recurrent Neural Network

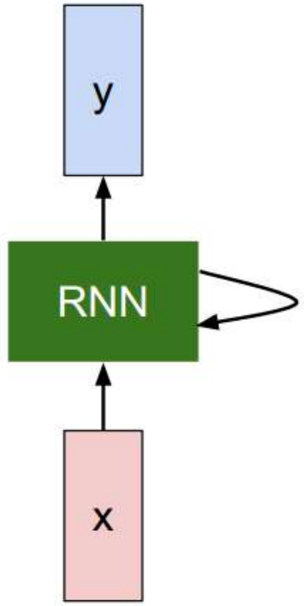
上海科技大学 ShanghaiTech University

- General formulation

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

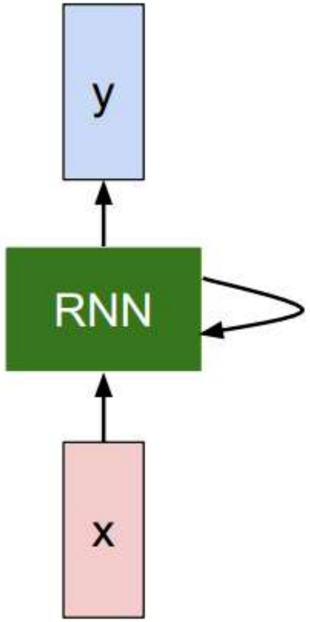
Notice: the same function and the same set of parameters are used at every time step.



(Vanilla)Recurrent Neural Network

- General formulation

The state consists of a single *"hidden"* vector **h**:



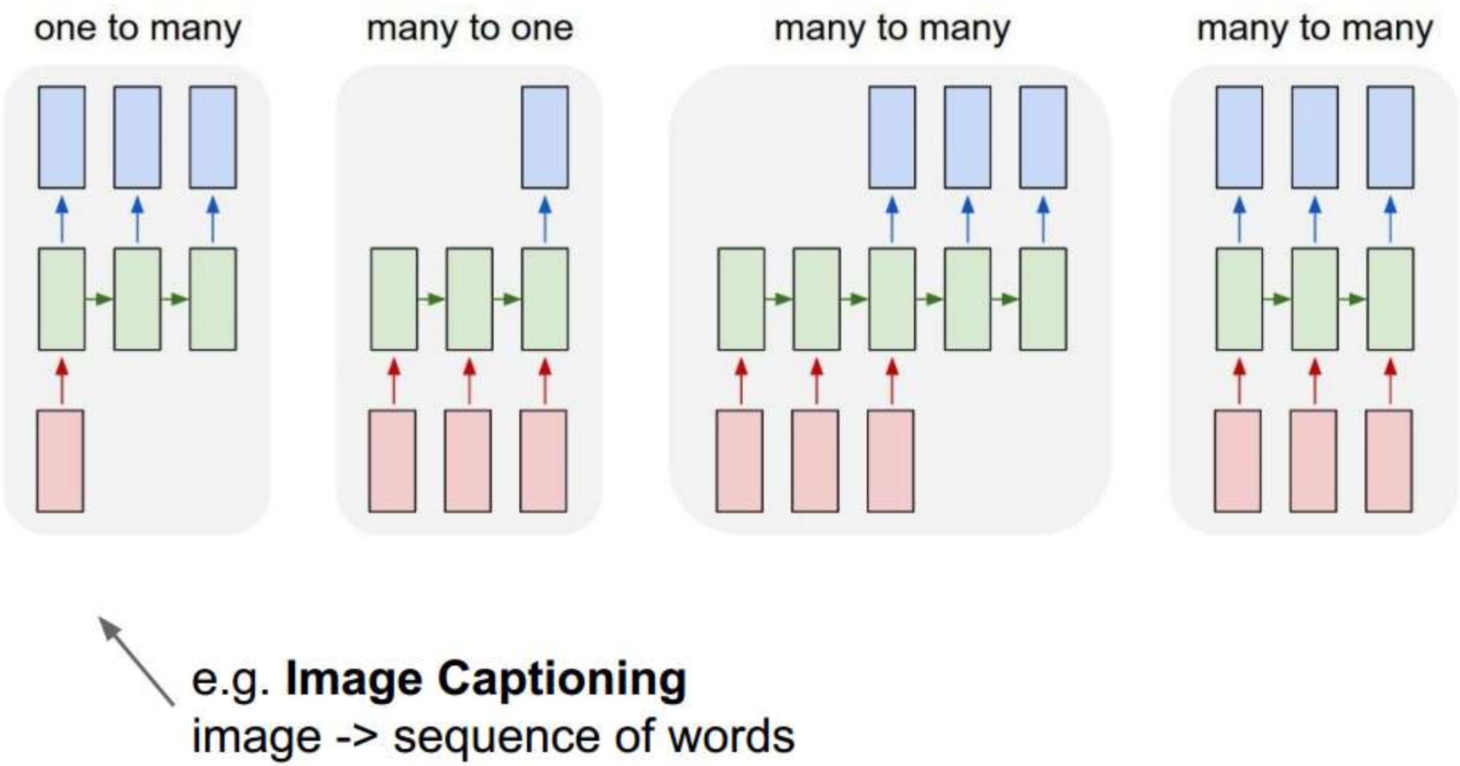
$$h_t = f_W(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$

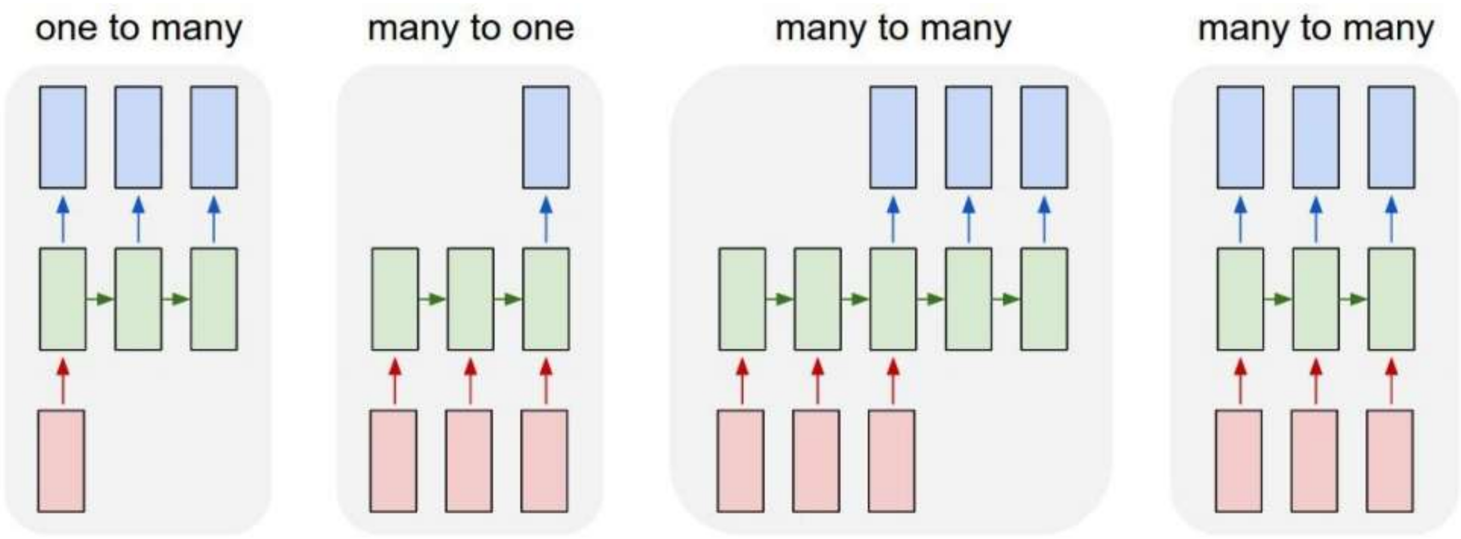
Recurrent Neural Network

- Recurrent Neural Networks: model variants



Recurrent Neural Network

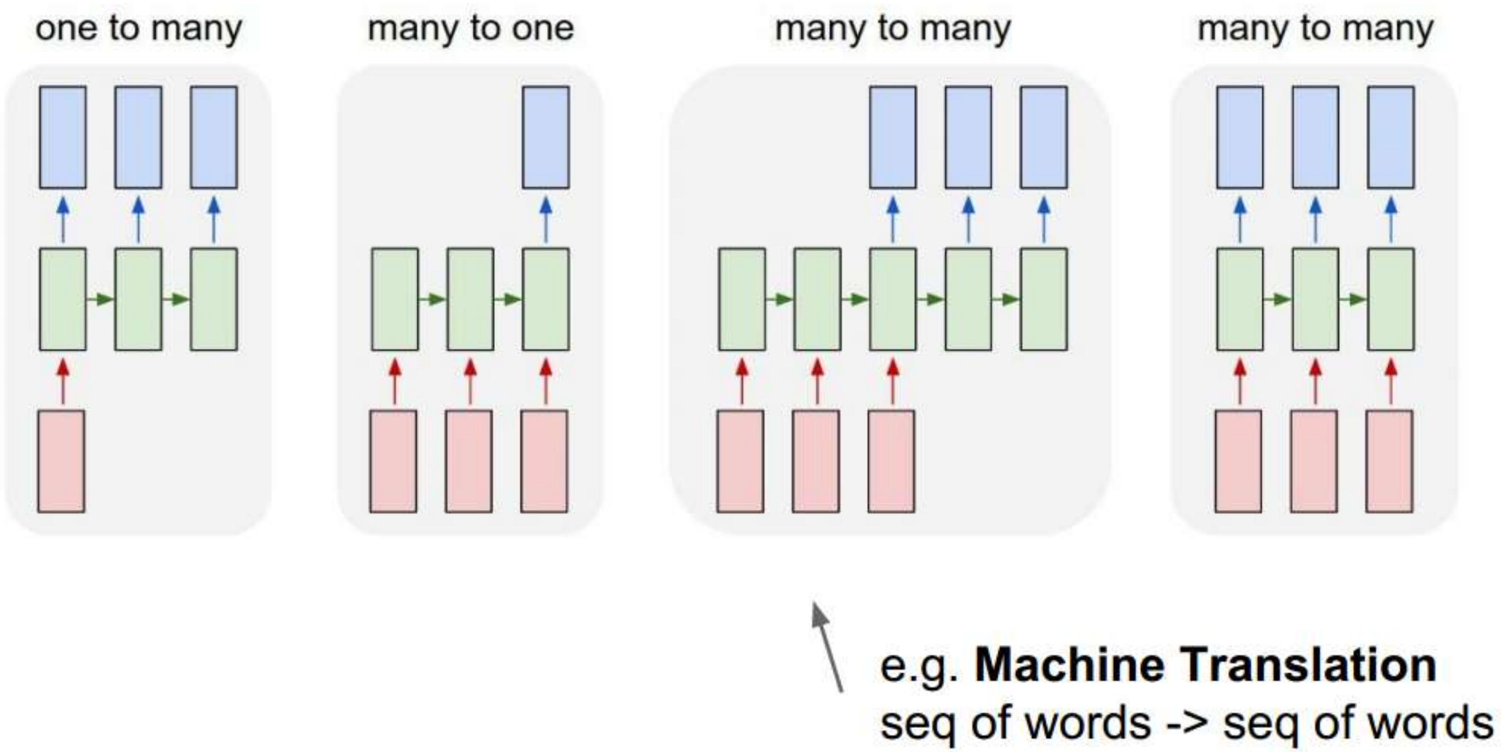
- Recurrent Neural Networks: model variants



e.g. **Sentiment Classification**
sequence of words -> sentiment

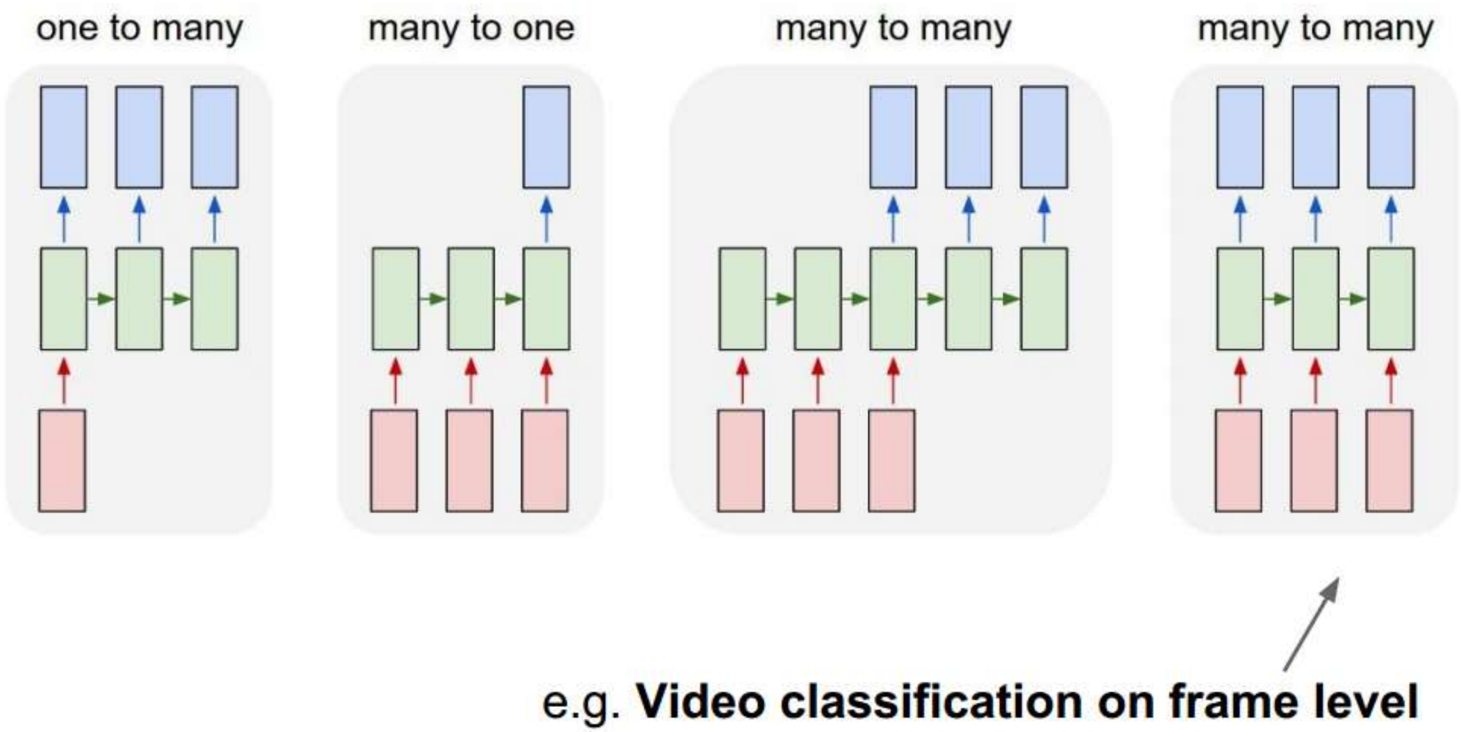
Recurrent Neural Network

- Recurrent Neural Networks: model variants



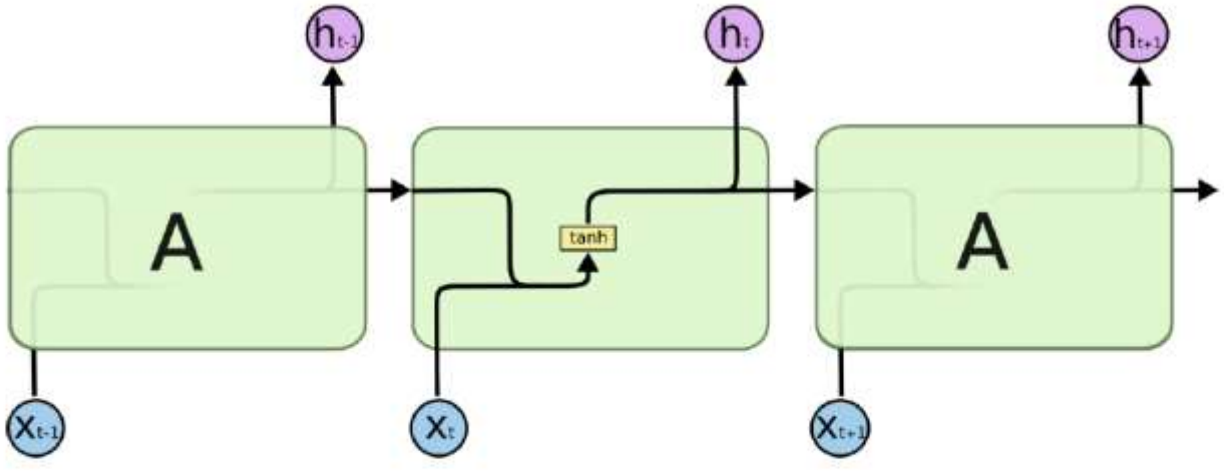
Recurrent Neural Network

- Recurrent Neural Networks: model variants



Standard RNN

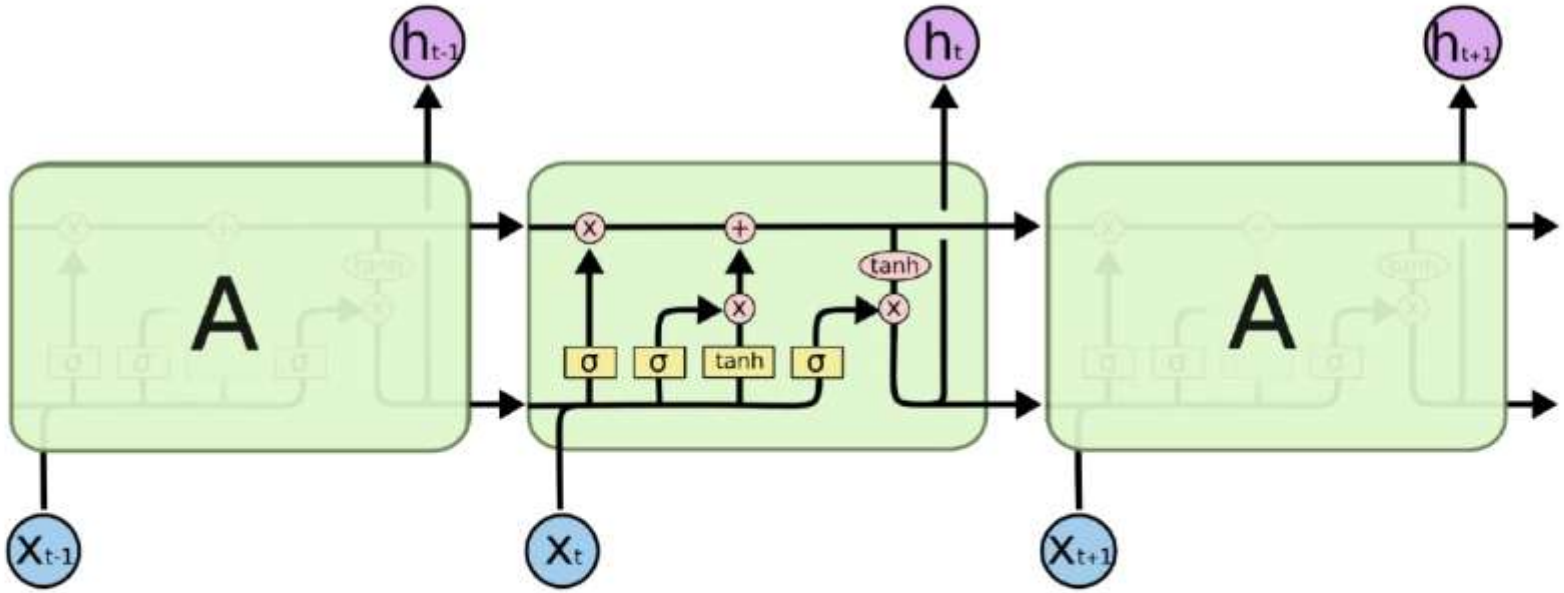
- Recall



- Each recurrent neuron receives past outputs and current input
- Pass through a tanh function

Long Short Term Memory(LSTM)

- LSTM uses multiplicative gates that decide if something is important or not

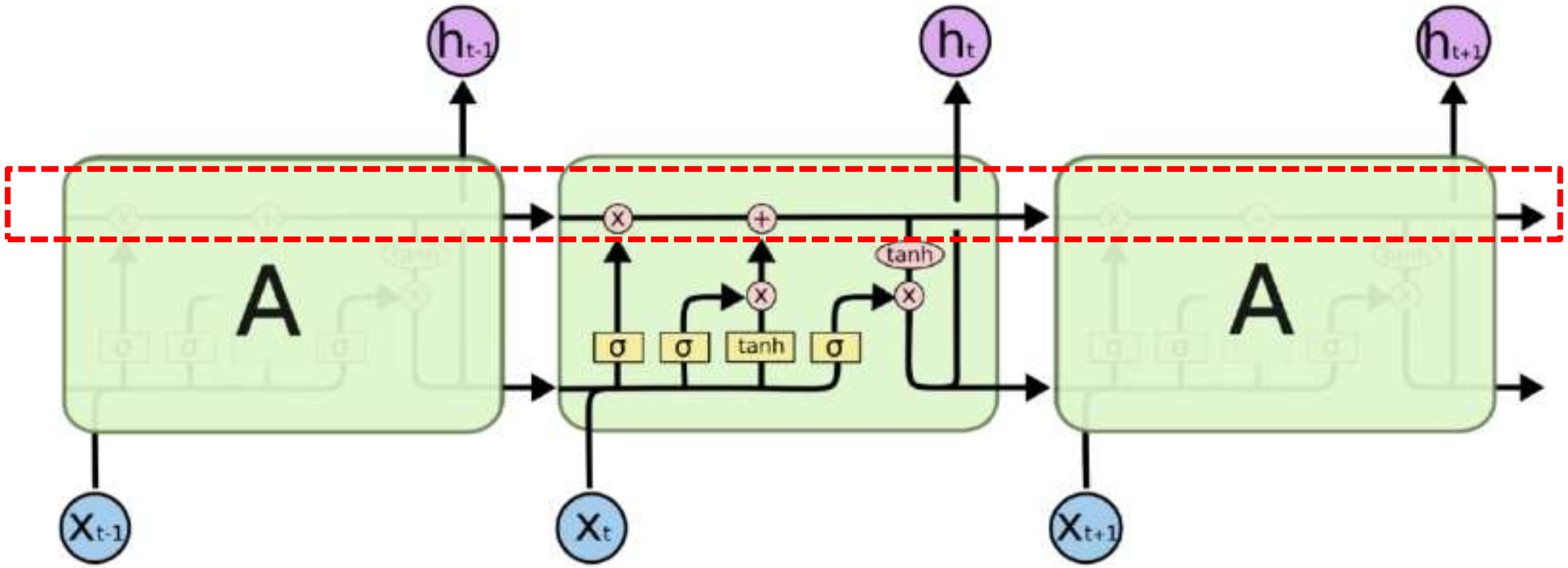


Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation

Long Short Term Memory(LSTM)

上海科技大学 ShanghaiTech University

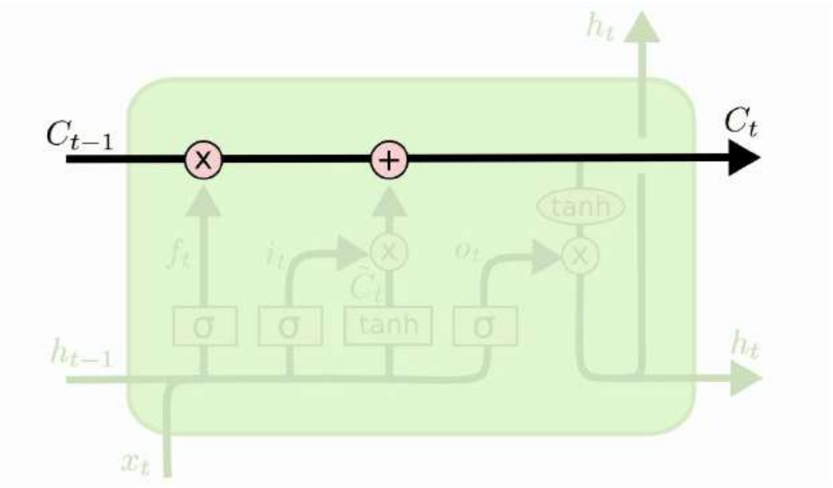
- Key component: a remembered cell state



Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation

LSTM: cell state

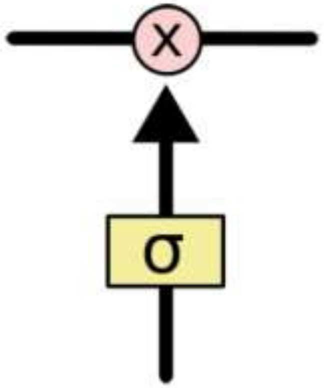
- A linear history
  - Carries information through
  - Only affected by a gate and addition of current information, which is also gated



LSTM: gates

Gates are simple sigmoid units with output range in (0,1)

- Controls how much of the information will be let through

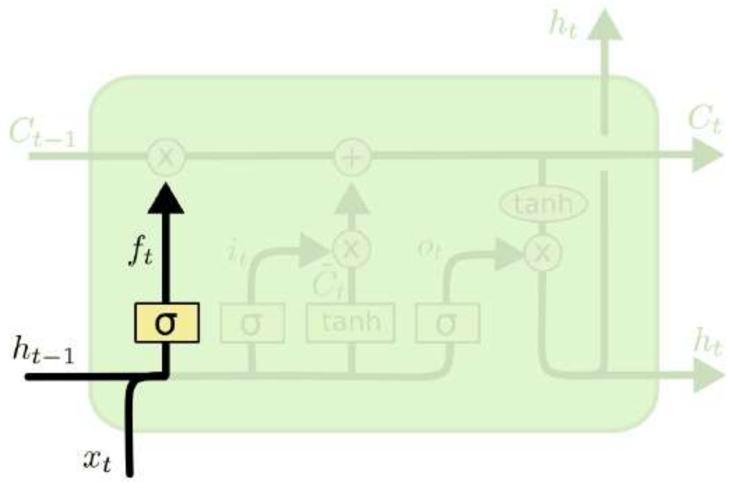


- Three gates
  - Forget gate
  - Input gate
  - Output gate

LSTM: forget gate

- The first gate determines whether to carry over the history or to forget it
  - Soft decision: how much of the history C_{t-1} to carry over
  - Determined by the current input x_t and the previous state h_{t-1}
  - can be viewed as partial key-value pairs

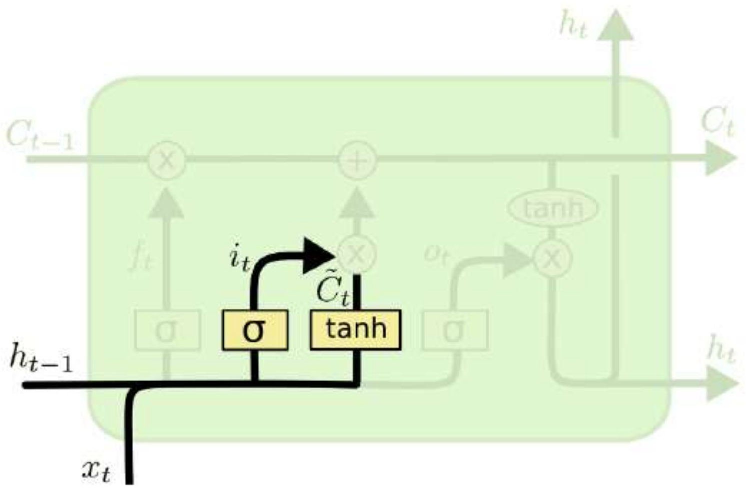
$$\langle h_{t-1}, C_{t-1} \rangle$$



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

LSTM: input gate

- The second gate has two parts
  - A gate that decides if it is worth remembering
  - A nonlinear transformation that extracts new and interesting information from the input
  - Both use the current input and the previous state

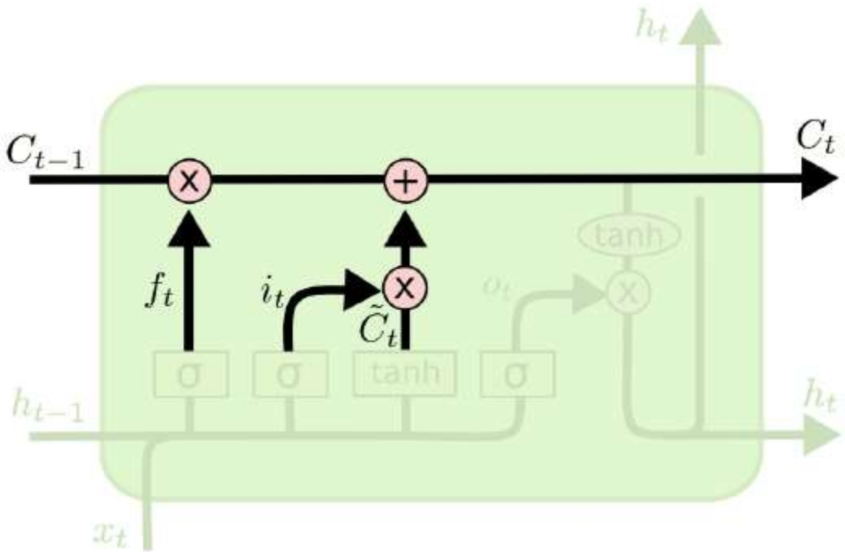


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

LSTM: Memory cell update

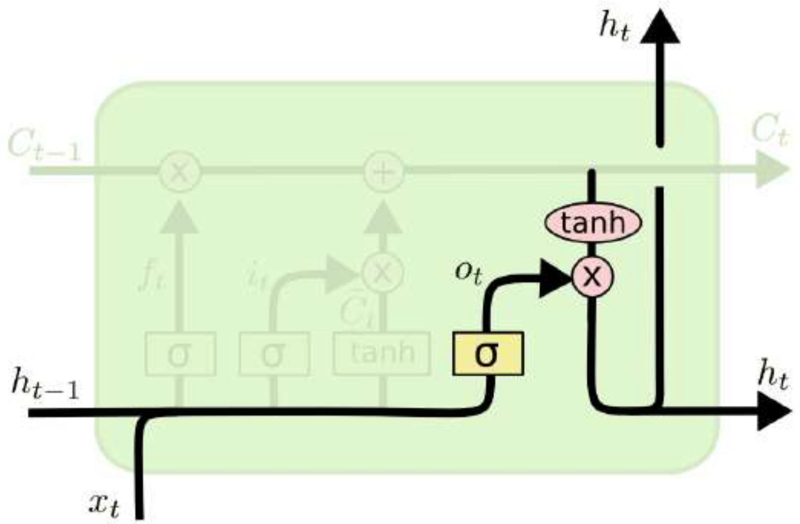
- The output of the second part is added into the current memory cell
  - Can be viewed as value update in a key-value pair
  - The input and state jointly decide how much of history info is kept and how much of embedded input info is added
  - A dynamic mixture of experts at each time step



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

LTSM: Output gate

- The third gate is the output gate
  - To decide if the memory cell contents are worth reporting at this time using the current input and previous state
- The output of the cell or the state
  - A nonlinear transform of the cell values
  - Compress it with tanh to make it in (-1,1)
  - Note the separation of key-value representation



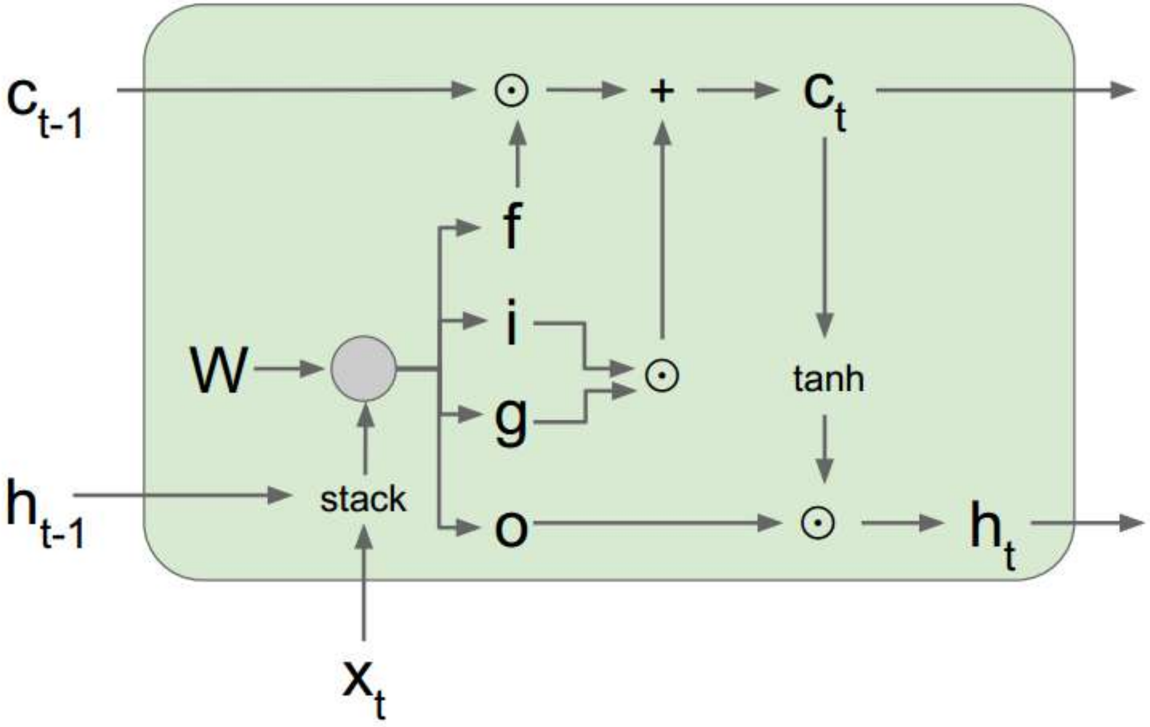
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

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

Long Short Term Memory(LSTM)

上海科技大学 ShanghaiTech University

[Hochreiter et al., 1997]



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

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

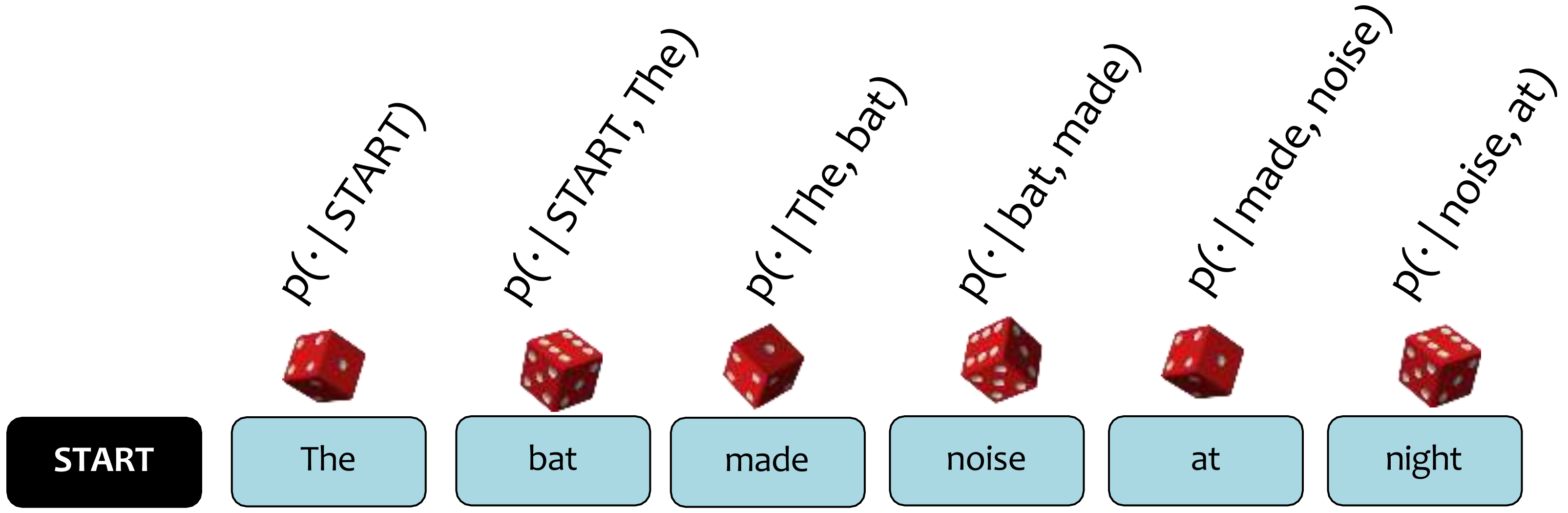
$$h_t = o \odot \tanh(c_t)$$

BACKGROUND:
N-GRAM LANGUAGE MODELS

n-Gram Language Model

上海科技大学 ShanghaiTech University

- *Goal*: Generate realistic looking sentences in a human language
- *Key Idea*: condition on the last n-1 words to sample the nth word



n-Gram Language Model

上海科技大学
ShanghaiTech University

- *Question*: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n-Gram Model (n=2)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1})$$

$p(w_1, w_2, w_3, \ldots, w_6) =$

The	$p(w_1)$
The bat	$p(w_2 \mid w_1)$
bat made	$p(w_3 \mid w_2)$
made noise	$p(w_4 \mid w_3)$
noise at	$p(w_5 \mid w_4)$
at night	$p(w_6 \mid w_5)$

n-Gram Language Model

- *Question*: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n-Gram Model (n=3) $$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

$$p(w_1, w_2, w_3, \ldots, w_6) =$$

The						$p(w_1)$
The	bat					$p(w_2 \mid w_1)$
The	bat	made				$p(w_3 \mid w_2, w_1)$
	bat	made	noise			$p(w_4 \mid w_3, w_2)$
		made	noise	at		$p(w_5 \mid w_4, w_3)$
			noise	at	night	$p(w_6 \mid w_5, w_4)$

n-Gram Language Model

上海科技大学
ShanghaiTech University

- *Question*: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n-Gram Model (n=3)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

$p(w_1, w_2, w_3, \ldots, w_6) =$

The

$p(w_1)$

The

The

Note: This is called a **model** because we made some **assumptions** about how many previous words to condition on (i.e. only n-1 words)

Learning an n-Gram Model

<u>Question</u>: How do we **learn** the probabilities for the n-Gram Model?

$p(w_t \mid w_{t-2} = \text{The}, w_{t-1} = \text{bat})$

w_t	$p(\cdot \mid \cdot, \cdot)$
ate	0.015
…	
flies	0.046
…	
zebra	0.000

$p(w_t \mid w_{t-2} = \text{made}, w_{t-1} = \text{noise})$

w_t	$p(\cdot \mid \cdot, \cdot)$
at	0.020
…	
pollution	0.030
…	
zebra	0.000

$p(w_t \mid w_{t-2} = \text{cows}, w_{t-1} = \text{eat})$

w_t	$p(\cdot \mid \cdot, \cdot)$
corn	0.420
…	
grass	0.510
…	
zebra	0.000

Learning an n-Gram Model

Question: How do we **learn** the probabilities for the n-Gram Model?

Answer: From data! Just **count** n-gram frequencies

… the **cows eat grass**…
… our **cows eat hay** daily…
… factory-farm **cows eat corn**…
… on an organic farm, **cows eat hay** and…
… do your **cows eat grass** or corn?...
… what do **cows eat if** they have…
… **cows eat corn** when there is no…
… which **cows eat which** foods depends…
… if **cows eat grass**…
… when **cows eat corn** their stomachs…
… should we let **cows eat corn**?...

$p(w_t \mid w_{t-2} = \text{cows}, w_{t-1} = \text{eat})$

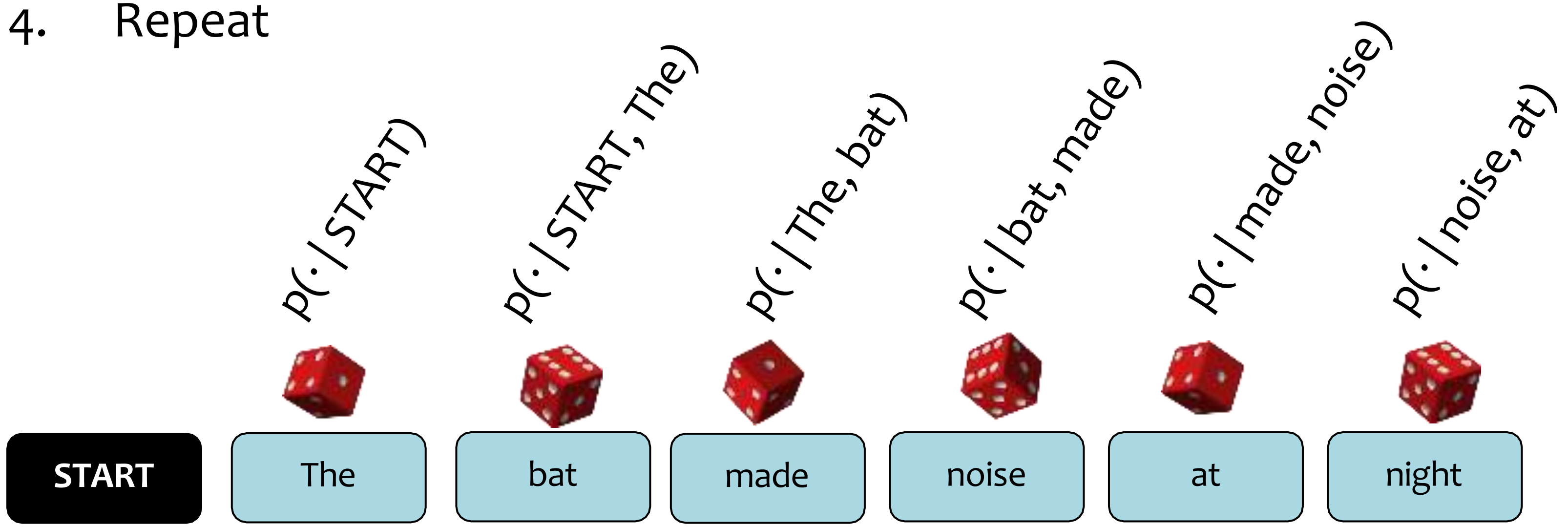
w_t	$p(\cdot \mid \cdot, \cdot)$
corn	4/11
grass	3/11
hay	2/11
if	1/11
which	1/11

Sampling from a Language Model

<u>Question</u>: How do we sample from a Language Model?

<u>Answer</u>:

1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to $p(w_t \mid w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat



Sampling from a Language Model

上海科技大学
ShanghaiTech University

Question: How do we sample from a Language Model?
Answer:
1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to $p(w_t \mid w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat

Training Data (Shakespeaere)	5-Gram Model
I tell you, friends, most charitable care ave the patricians of you. For your wants, Your suffering in this dearth, you may as well Strike at the heaven with your staves as lift them Against the Roman state, whose course will on The way it takes, cracking ten thousand curbs Of more strong link asunder than can ever Appear in your impediment. For the dearth, The gods, not the patricians, make it, and Your knees to them, not arms, must help.	`Approacheth, denay. dungy Thither! Julius think: grant,--O Yead linens, sheep's Ancient, Agreed: Petrarch plaguy Resolved pear! observingly honourest adulteries wherever scabbard guess; affirmation--his monsieur; died. jealousy, chequins me. Daphne building. weakness: sun-rise, cannot stays carry't, unpurposed. prophet-like drink; back-return 'gainst surmise Bridget ships? wane; interim? She's striving wet;`

32

RECURRENT NEURAL NETWORK (RNN) LANGUAGE MODELS

Recurrent Neural Networks (RNNs)

上海科技大学
ShanghaiTech University

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathbb{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathbb{R}^J$

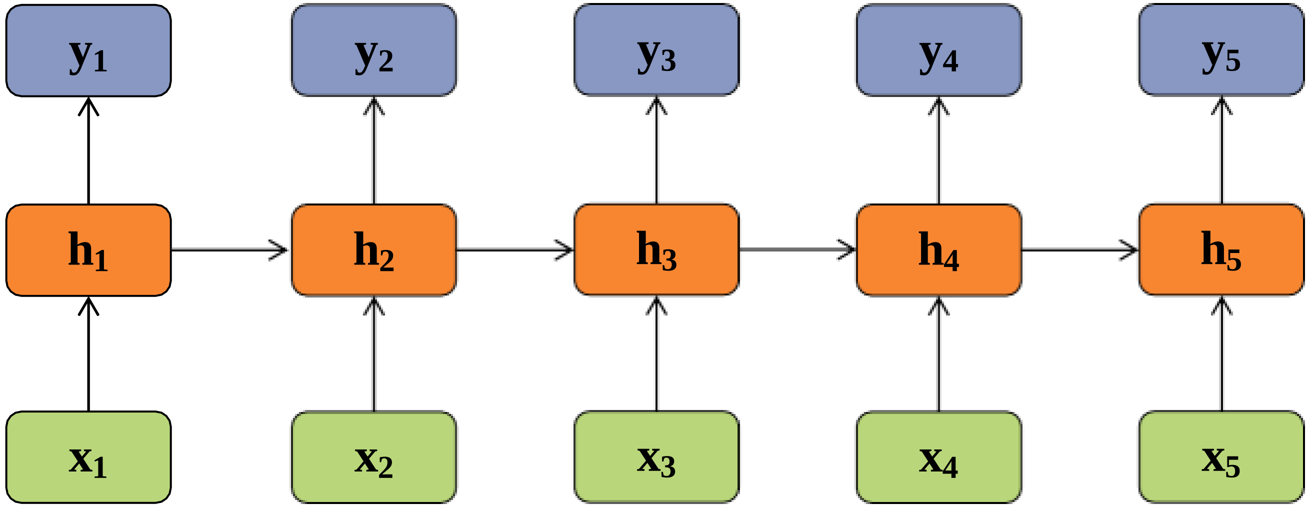
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathbb{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

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

$$y_t = W_{hy}h_t + b_y$$



Recall…

The Chain Rule of Probability

- *Question*: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

Chain rule of probability: $$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, \ldots, w_1)$$

$p(w_1, w_2, w_3, \ldots, w_6) =$

The

The

The

The

The

The

$p(w_1)$

Note: This is called the chain **rule** because it is **always** true for every probability distribution

RNN Language Model

上海科技大学
ShanghaiTech University

RNN Language Model: $p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid f_{\boldsymbol{\theta}}(w_{t-1}, \ldots, w_1))$

$p(w_1, w_2, w_3, \ldots, w_6) =$

The						$p(w_1)$
The	bat					$p(w_2 \mid f_\theta(w_1))$
The	bat	made				$p(w_3 \mid f_\theta(w_2, w_1))$
The	bat	made	noise			$p(w_4 \mid f_\theta(w_3, w_2, w_1))$
The	bat	made	noise	at		$p(w_5 \mid f_\theta(w_4, w_3, w_2, w_1))$
The	bat	made	noise	at	night	$p(w_6 \mid f_\theta(w_5, w_4, w_3, w_2, w_1))$

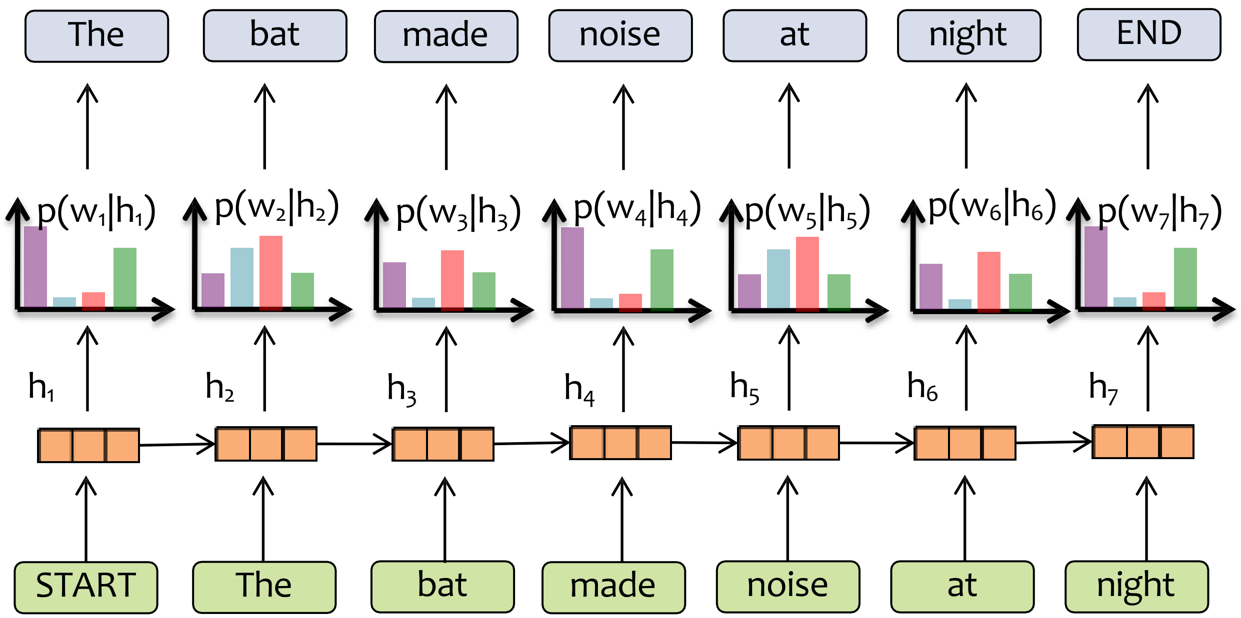
Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector

RNN Language Model

上海科技大学
ShanghaiTech University

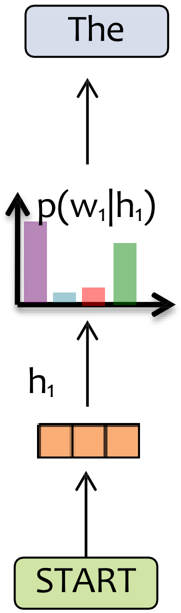


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

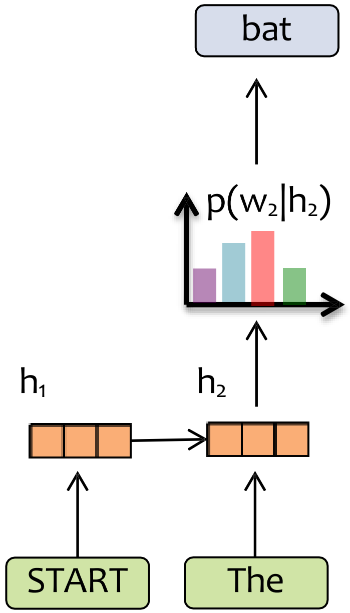


<u>*Key Idea*</u>:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

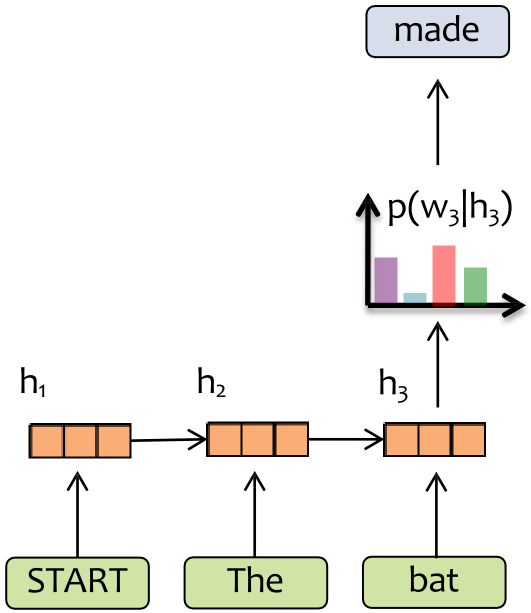


<u>Key Idea</u>:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

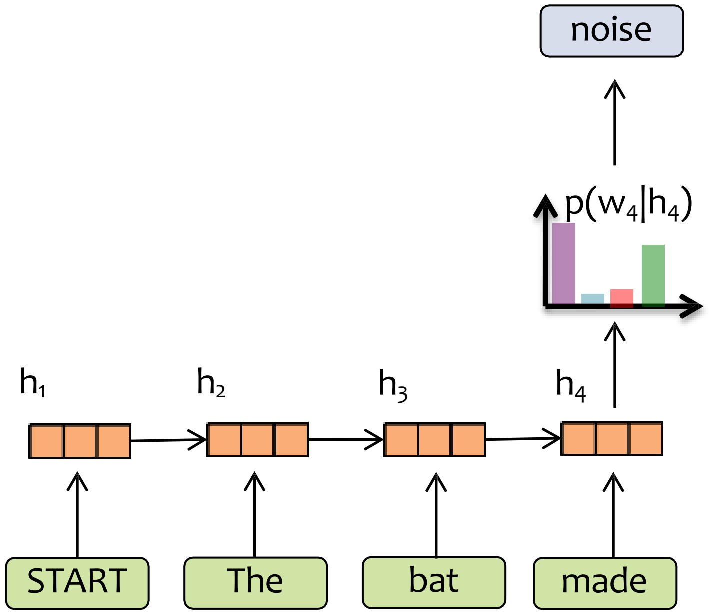


<u>*Key Idea*</u>:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model



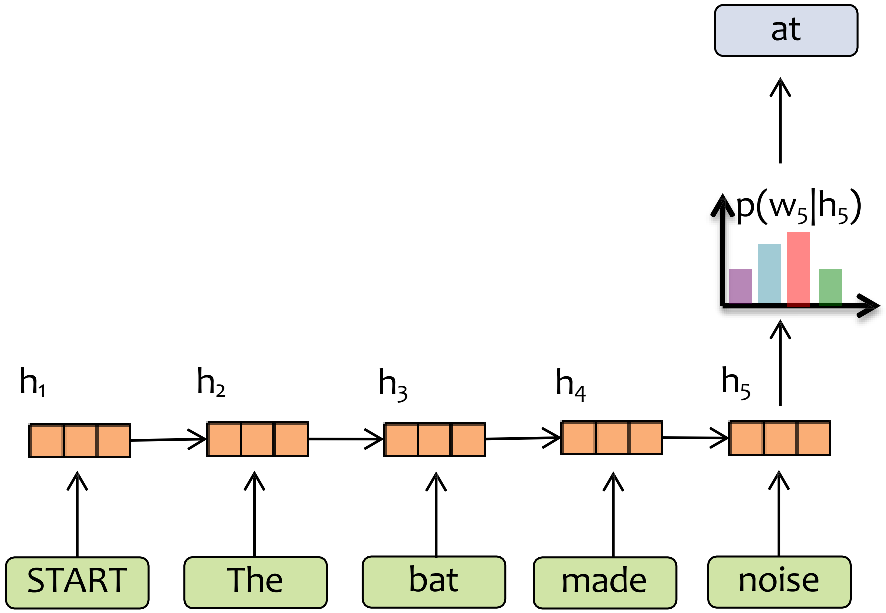
<u>*Key Idea*</u>:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

上海科技大学
ShanghaiTech University



<u>Key Idea</u>:

(1) convert all previous words to a **fixed length vector**

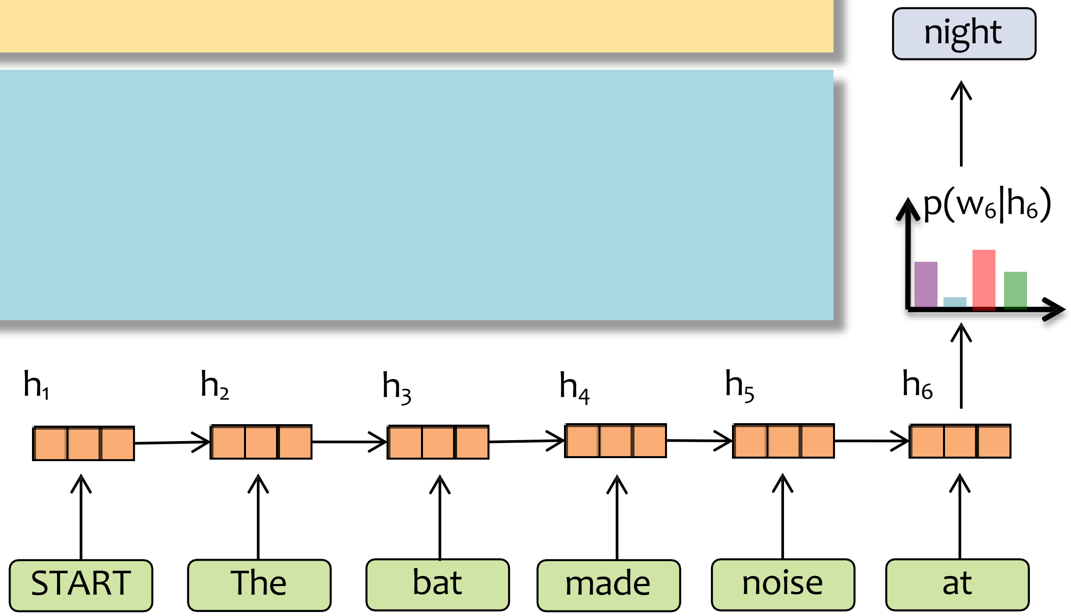
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

上海科技大学 ShanghaiTech University

Question: How can we create a distribution $p(w_t|h_t)$ from h_t?

Answer:

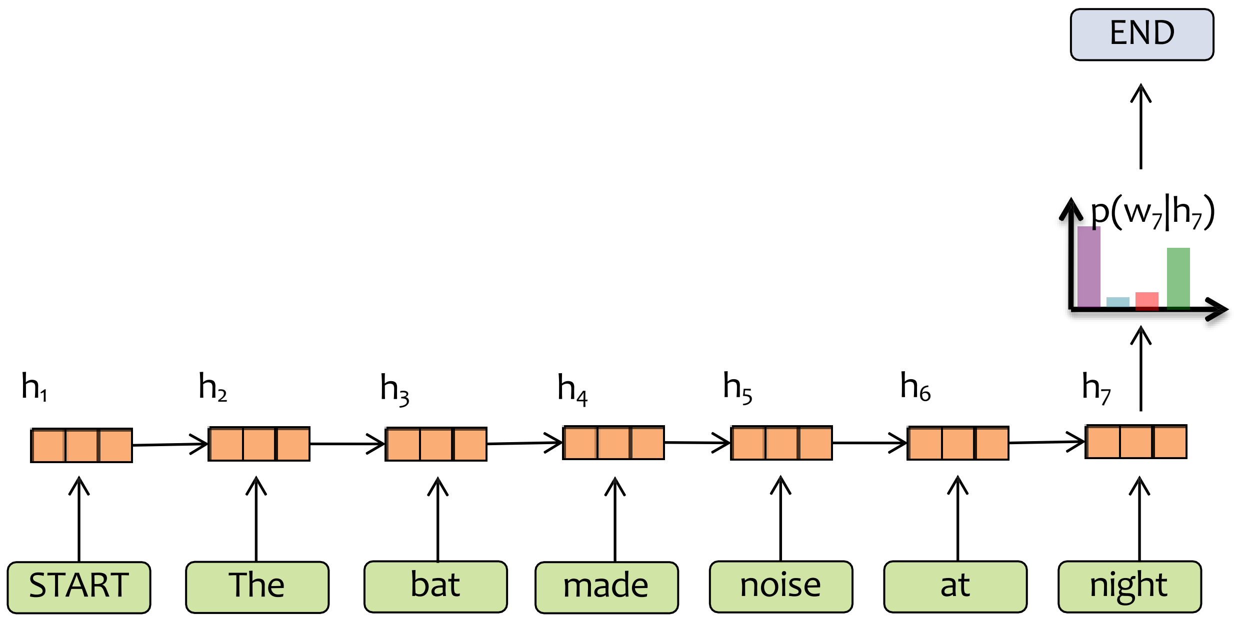


<u>Key Idea</u>:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

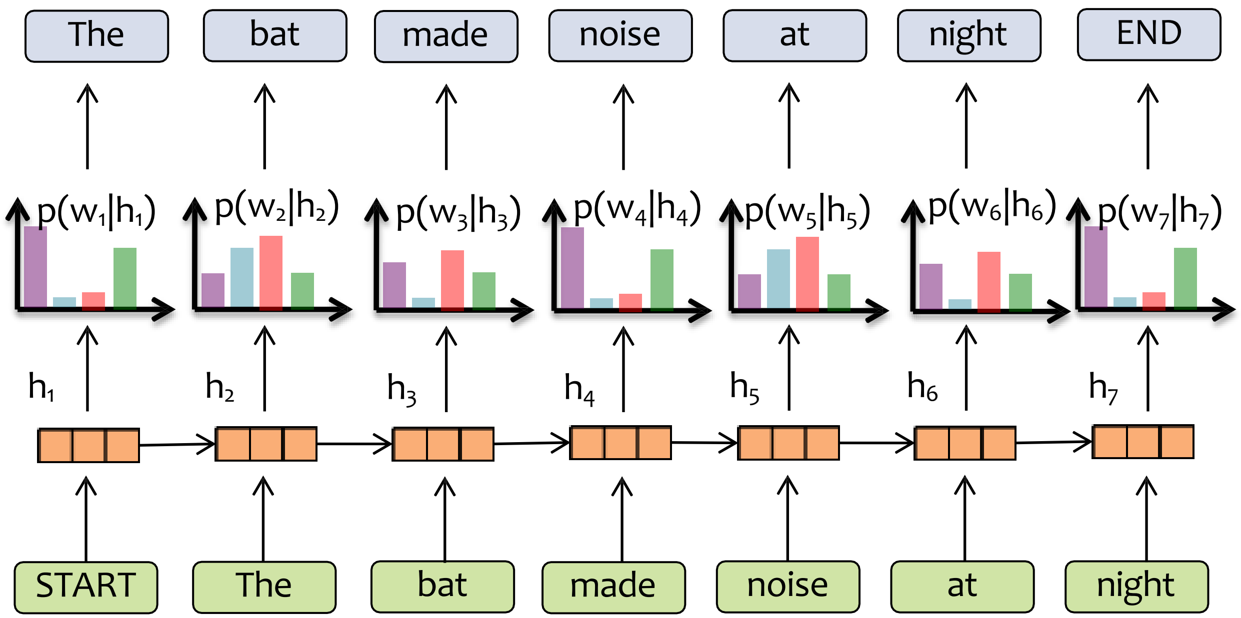


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model



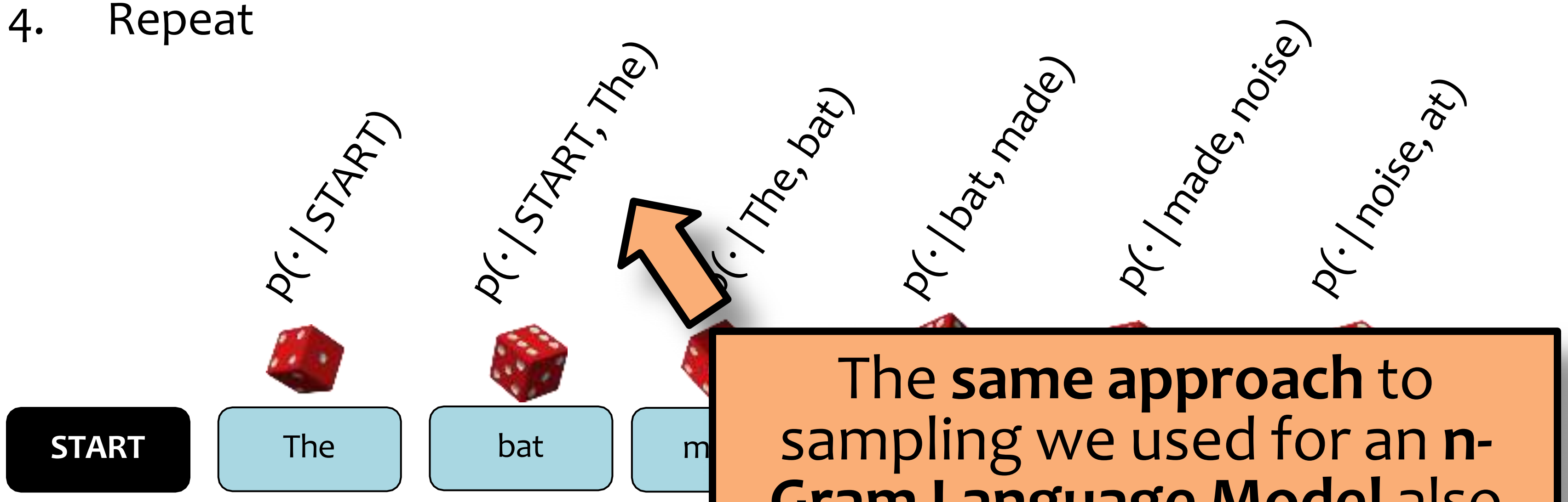
$$p(w_1, w_2, w_3, \ldots, w_T) = p(w_1 \mid h_1)\, p(w_2 \mid h_2) \ldots\, p(w_2 \mid h_T)$$

Sampling from a Language Model

Question: How do we sample from a Language Model?

Answer:

1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to $p(w_t \mid w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat



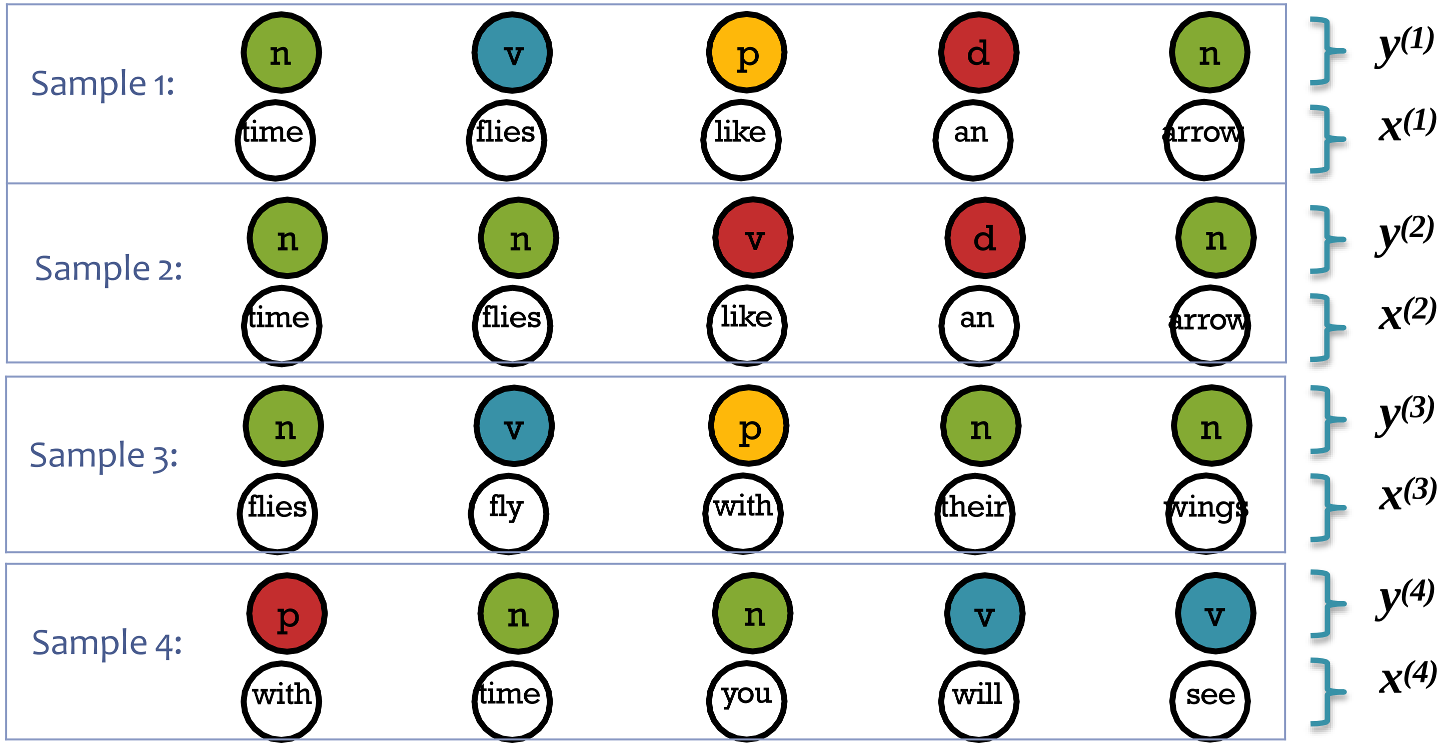
The **same approach** to sampling we used for an **n-Gram Language Model** also works here for an **RNN Language Model**

LEARNING AN RNN

47

Dataset for Supervised Part-of-Speech (POS) Tagging

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



SGD and Mini-batch SGD

上海科技大学
ShanghaiTech University

Recall...

Algorithm 1 SGD

Initialize $\theta^{(0)}$

$s = 0$

for $t = 1, 2, \ldots, T$ **do**

  for $i \in$ shuffle$(1, \ldots, N)$ **do**

    Select the next training point (x_i, y_i)

    Compute the gradient $g^{(s)} = \nabla J_i(\theta^{(s-1)})$

    Update parameters $\theta^{(s)} = \theta^{(s-1)} - \eta g^{(s)}$

    Increment time step $s = s + 1$

  Evaluate average training loss $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$

return $\theta^{(s)}$

SGD and Mini-batch SGD

上海科技大学 ShanghaiTech University

Recall...

Algorithm 1 Mini-Batch SGD

1: Initialize $\theta^{(0)}$
2: Divide examples $\{1,\ldots,N\}$ randomly into batches $\{I_1,\ldots,I_B\}$
3: where $\bigcup_{b=1}^{B} I_b = \{1,\ldots,N\}$ and $\bigcap_{b=1}^{B} I_b = \emptyset$
4: $s = 0$
5: **for** $t = 1, 2, \ldots, T$ **do**
6:     **for** $b = 1, 2, \ldots, B$ **do**
7:         Select the next batch I_b, where $m = |I_b|$
8:         Compute the gradient $g^{(s)} = \frac{1}{m}\sum_{i\in I_b} \nabla J_i(\theta^{(s)})$
9:         Update parameters $\theta^{(s)} = \theta^{(s-1)} - \eta g^{(s)}$
10:         Increment time step $s = s + 1$
11:     Evaluate average training loss $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$
12: **return** $\theta^{(s)}$

RNN

Algorithm 1 Elman RNN

```
procedure FORWARD(x_{1:T}, W_ah, W_ax, b_a, W_yh, b_y)
    Initialize the hidden state h_0 to zeros
    for t in 1 to T do
        Receive input data at time step t: x_t
        Compute the hidden state update:
            a_t = W_ah · h_{t-1} + W_ax · x_t + b_a
            h_t = σ(a_t)
        Compute the output at time step t:
            y_t = W_yh · h_t + b_y
```

procedure FORWARD($x_{1:T}$, W_{ah}, W_{ax}, b_a, W_{yh}, b_y)

Initialize the hidden state h_0 to zeros

for t in 1 to T **do**

Receive input data at time step t: x_t

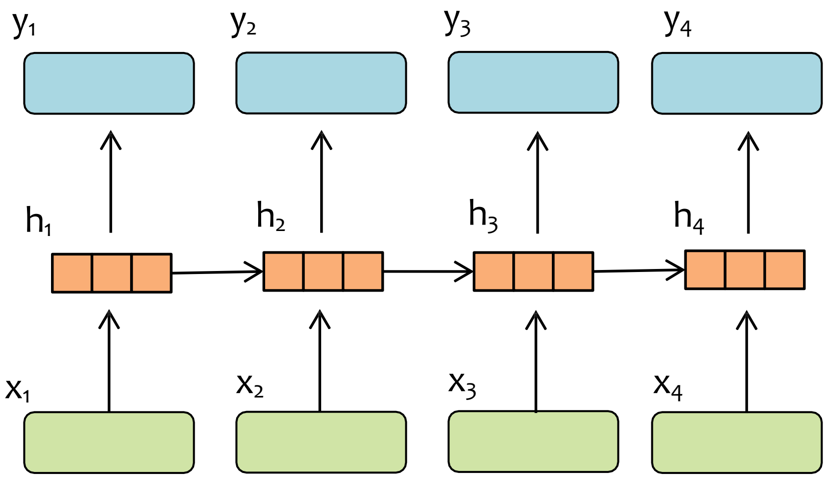
Compute the hidden state update:

$$a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$$

$$h_t = \sigma(a_t)$$

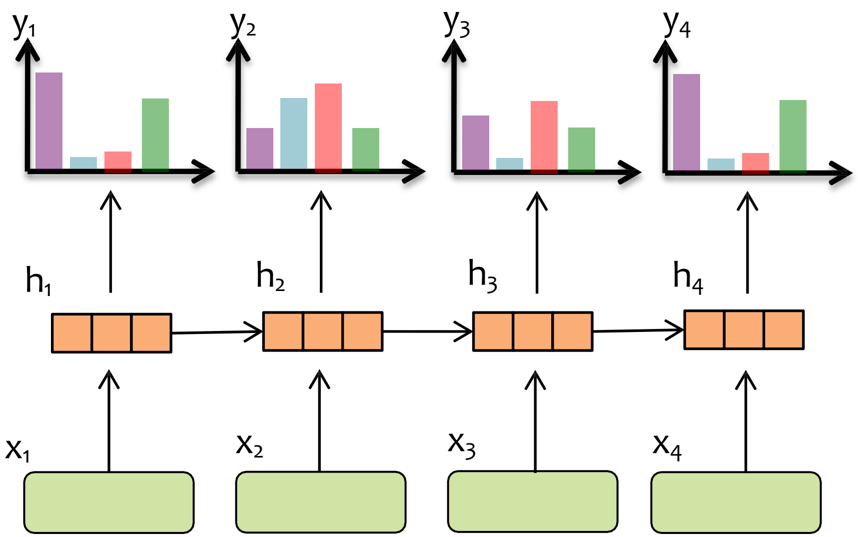
Compute the output at time step t:

$$y_t = W_{yh} \cdot h_t + b_y$$



51

RNN



Algorithm 1 Elman RNN

procedure FORWARD($x_{1:T}$, W_{ah}, W_{ax}, b_a, W_{yh}, b_y)

  Initialize the hidden state h_0 to zeros

  for t in 1 to T **do**

    Receive input data at time step t: x_t

    Compute the hidden state update:

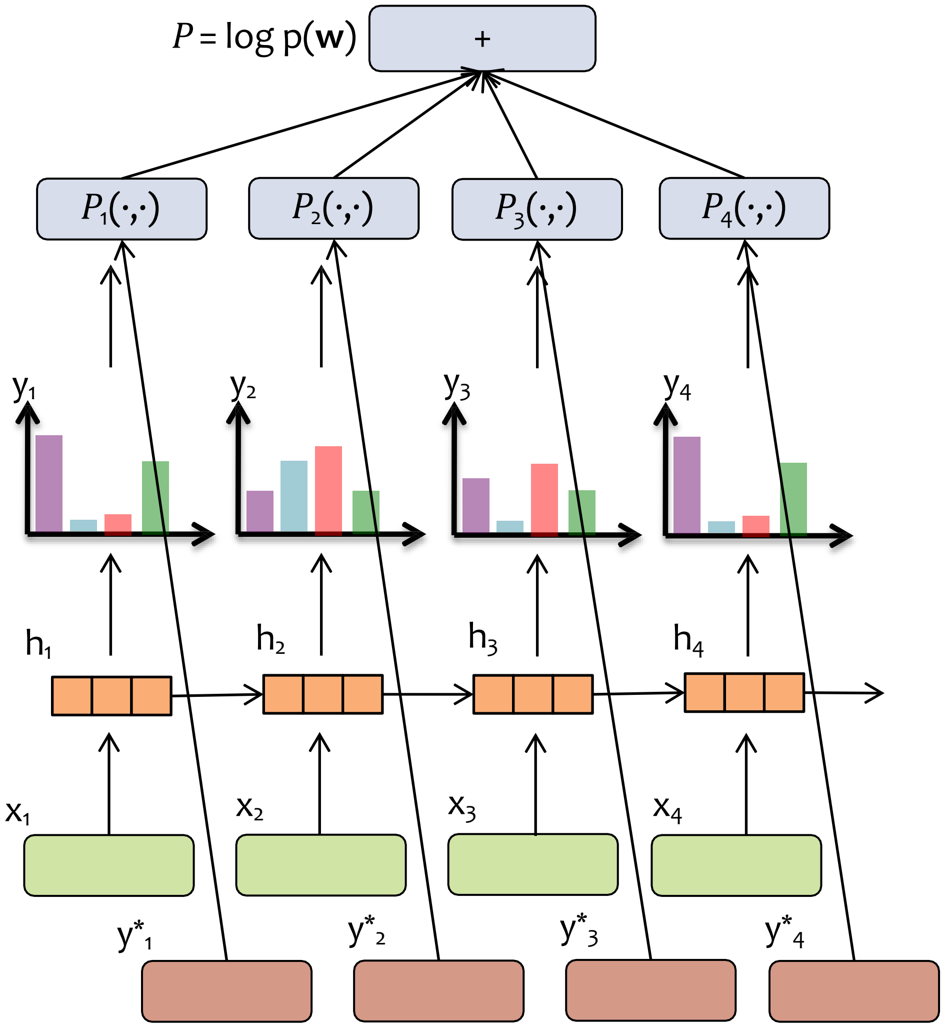
      $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$

      $h_t = \sigma(a_t)$

    Compute the output at time step t:

      $y_t = \text{softmax}(W_{yh} \cdot h_t + b_y)$

RNN + Loss



Algorithm 1 Elman RNN + Loss

procedure FORWARD($x_{1:T}$, $y^*_{1:T} W_{ah}, W_{ax}, b_a, W_{yh}, b_y$)
  Initialize the hidden state h_0 to zeros
  for t in 1 to T **do**
    Receive input data at time step t: x_t
    Compute the hidden state update:
      $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$
      $h_t = \sigma(a_t)$
    Compute the output at time step t:
      $y_t = \text{softmax}(W_{yh} \cdot h_t + b_y)$
    Compute the cross-entropy loss at time step t:
      $\ell_t = -\sum_{k=1}^{K} (y_t^*)_k \log((y_t)_k)$
  Compute the total loss:
  $\ell = \sum_{t=1}^{T} \ell_t$

LEARNING AN RNN-LM

Learning a Language Model

<u>Question</u>: How do we **learn** the probabilities for the n-Gram Model?

<u>Answer</u>: From data! Just **count** n-gram frequencies

… the **cows eat grass**…
… our **cows eat hay** daily…
… factory-farm **cows eat corn**…
… on an organic farm, **cows eat hay** and…
… do your **cows eat grass** or corn?...
… what do **cows eat if** they have…
… **cows eat corn** when there is no…
… which **cows eat which** foods depends…
… if **cows eat grass**…
… when **cows eat corn** their stomachs…
… should we let **cows eat corn**?...

$p(w_t \mid w_{t-2} = \text{cows}, w_{t-1} = \text{eat})$

w_t	$p(\cdot \mid \cdot, \cdot)$
corn	4/11
grass	3/11
hay	2/11
if	1/11
which	1/11

MLE for n-gram LM

- This counting method gives us the **maximum likelihood estimate** of the n-gram LM parameters
- We can derive it in the usual way:
  - **Write the likelihood** of the sentences under the n-gram LM
  - **Set the gradient to zero** and impose the constraint that the probabilities sum-to-one
  - **Solve** for the MLE

Learning a Language Model

MLE for Deep Neural LM

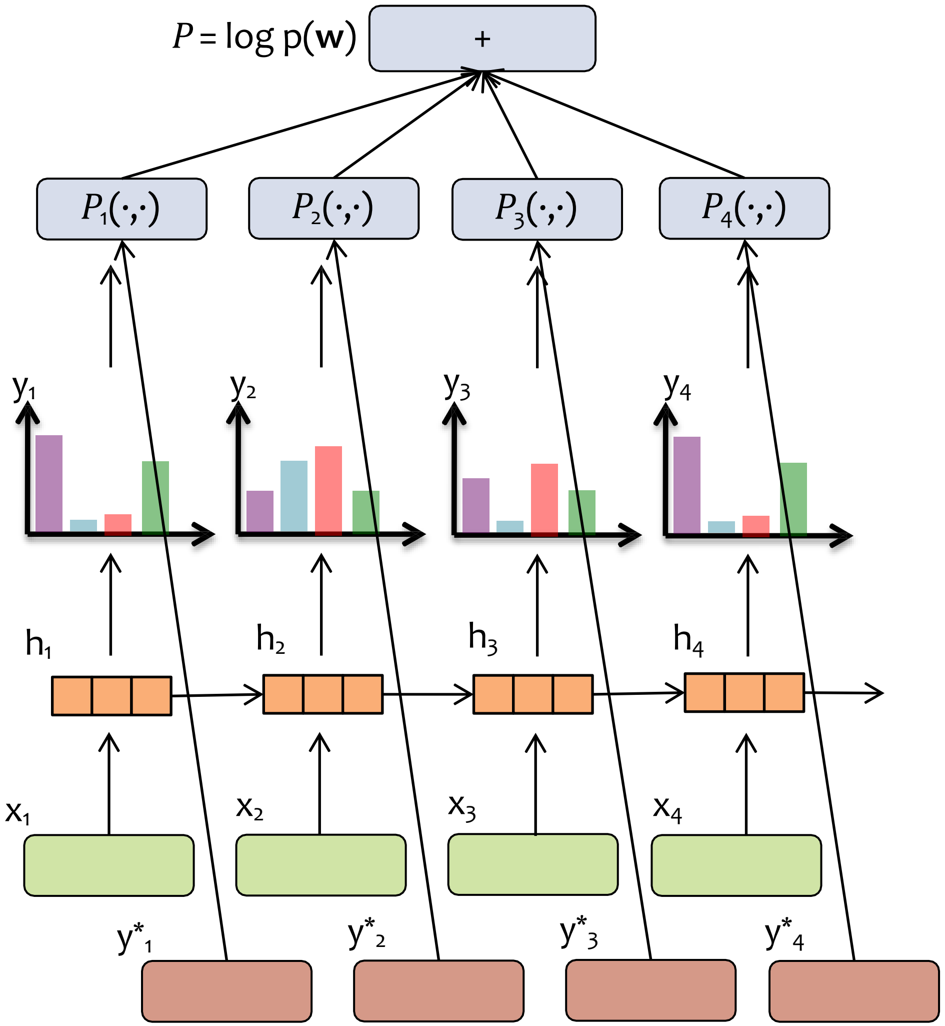
- We can also use maximum likelihood estimation to learn the parameters of an RNN-LM or Transformer-LM too!
- But **not in closed form** – instead we follow a different recipe:
  - Write the **likelihood** of the sentences under the Deep Neural LM model
  - Compute the **gradient** of the (batch) likelihood w.r.t. the parameters **by AutoDiff**
  - Follow the negative gradient using **Mini-batch SGD** (or your favorite optimizer)

MLE for n-gram LM

- This counting method gives us the **maximum likelihood estimate** of the n-gram LM parameters
- We can derive it in the usual way:
  - **Write the likelihood** of the sentences under the n-gram LM
  - **Set the gradient to zero** and impose the constraint that the probabilities sum-to-one
  - **Solve** for the MLE

RNN + Loss

How can we use this to compute the loss for an RNN-LM?



Algorithm 1 Elman RNN + Loss

procedure FORWARD($x_{1:T}$, $y^*_{1:T} W_{ah}, W_{ax}, b_a, W_{yh}, b_y$)

  Initialize the hidden state h_0 to zeros

  for t in 1 to T **do**

    Receive input data at time step t: x_t

    Compute the hidden state update:

      $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$

      $h_t = \sigma(a_t)$

    Compute the output at time step t:

      $y_t = \text{softmax}(W_{yh} \cdot h_t + b_y)$

    Compute the cross-entropy loss at time step t:

      $\ell_t = -\sum_{k=1}^{K} (y^*_t)_k \log((y_t)_k)$

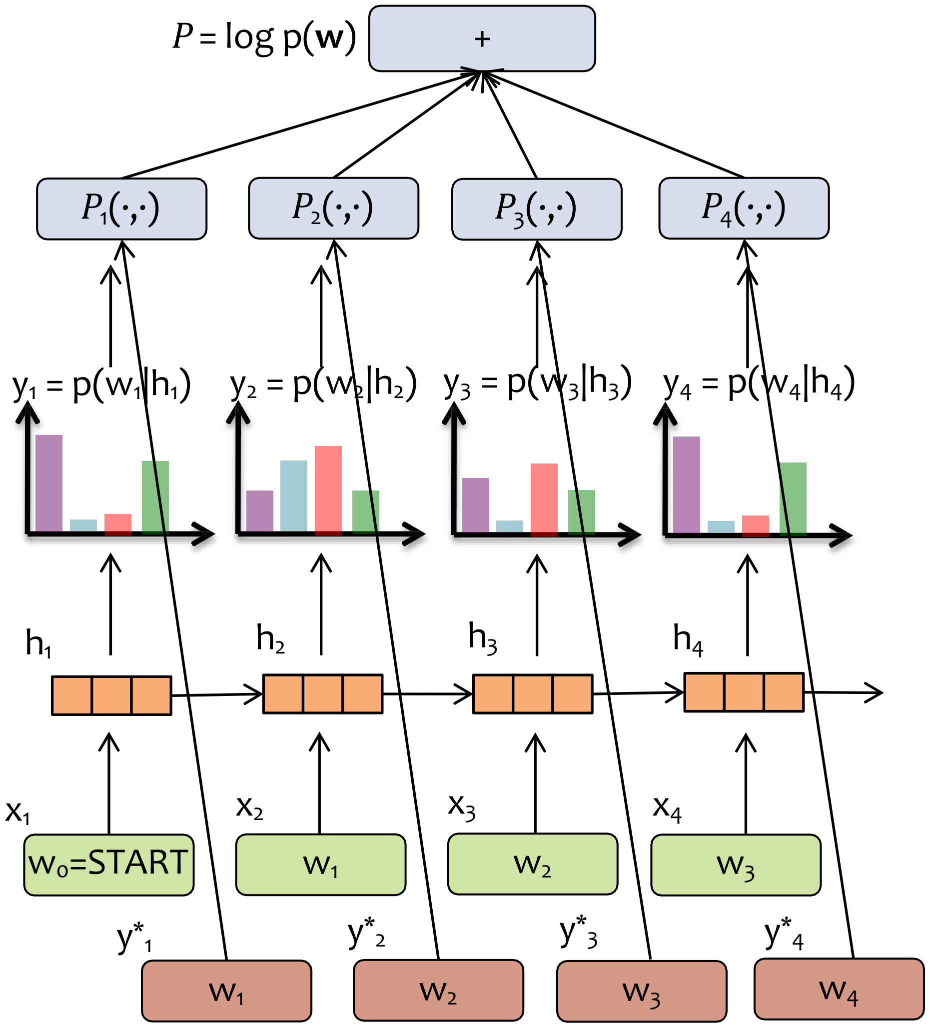
  Compute the total loss:

    $\ell = \sum_{t=1}^{T} \ell_t$

RNN-LM + Loss

How can we use this to compute the loss for an RNN-LM?

$$\log p(\mathbf{w}) = \log p(w_1, w_2, w_3, \ldots, w_T)$$
$$= \log p(w_1 \mid h_1) + \ldots + \log p(w_T \mid h_T)$$



Algorithm 1 Elman RNN + Loss

procedure FORWARD($x_{1:T}$, $y^*_{1:T}$ $W_{ah}, W_{ax}, b_a, W_{yh}, b_y$)

  Initialize the hidden state h_0 to zeros

  for t in 1 to T **do**

    Receive input data at time step t: x_t

    Compute the hidden state update:

      $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$

      $h_t = \sigma(a_t)$

    Compute the output at time step t:

      $y_t = \text{softmax}(W_{yh} \cdot h_t + b_y)$

    Compute the cross-entropy loss at time step t:

      $\ell_t = -\sum_{k=1}^{K} (y_t^*)_k \log((y_t)_k)$

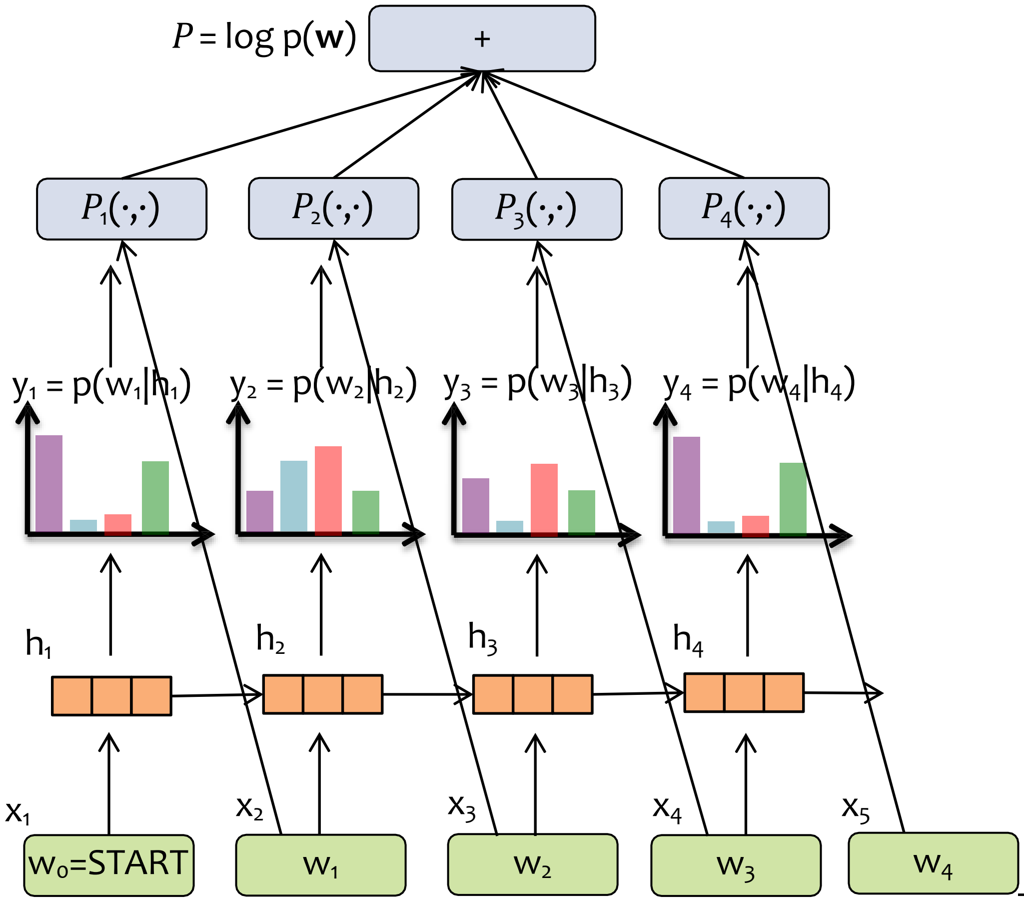
  Compute the total loss:

    $\ell = \sum_{t=1}^{T} \ell_t$

RNN-LM + Loss

How can we use this to compute the loss for an RNN-LM?

$$\log p(\mathbf{w}) = \log p(w_1, w_2, w_3, \ldots, w_T)$$
$$= \log p(w_1 \mid h_1) + \ldots + \log p(w_T \mid h_T)$$



Algorithm 1 Elman RNN + Loss

procedure FORWARD($x_{1:T}$, $y^*_{1:T}$ W_{ah}, W_{ax}, b_a, W_{yh}, b_y)
    Initialize the hidden state h_0 to zeros
    for t in 1 to T **do**
        Receive input data at time step t: x_t
        Compute the hidden state update:
            $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$
            $h_t = \sigma(a_t)$
        Compute the output at time step t:
            $y_t = \text{softmax}(W_{yh} \cdot h_t + b_y)$
        Compute the cross-entropy loss at time step t:
            $\ell_t = -\sum_{k=1}^{K} (y_t^*)_k \log((y_t)_k)$
    Compute the total loss:
    $\ell = \sum_{t=1}^{T} \ell_t$

Learning an RNN-LM

上海科技大学 ShanghaiTech University

- Each training example is a sequence (e.g. sentence), so we have training data D = {$\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$, … , $\mathbf{w}^{(N)}$}
- The objective function for a Deep LM (e.g. RNN-LM or Tranformer-LM) is typically the log-likelihood of the training examples:

  $$J(\boldsymbol{\theta}) = \Sigma_i \log p_{\boldsymbol{\theta}}(\mathbf{w}^{(i)})$$

- We train by mini-batch SGD (or your favorite flavor of mini-batch SGD)

$$\log p(\mathbf{w}) = \log p(w_1, w_2, w_3, \ldots, w_T)$$
$$= \log p(w_1 \mid h_1) + \log p(w_2 \mid h_2) + \ldots + \log p(w_T \mid h_T)$$