# 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

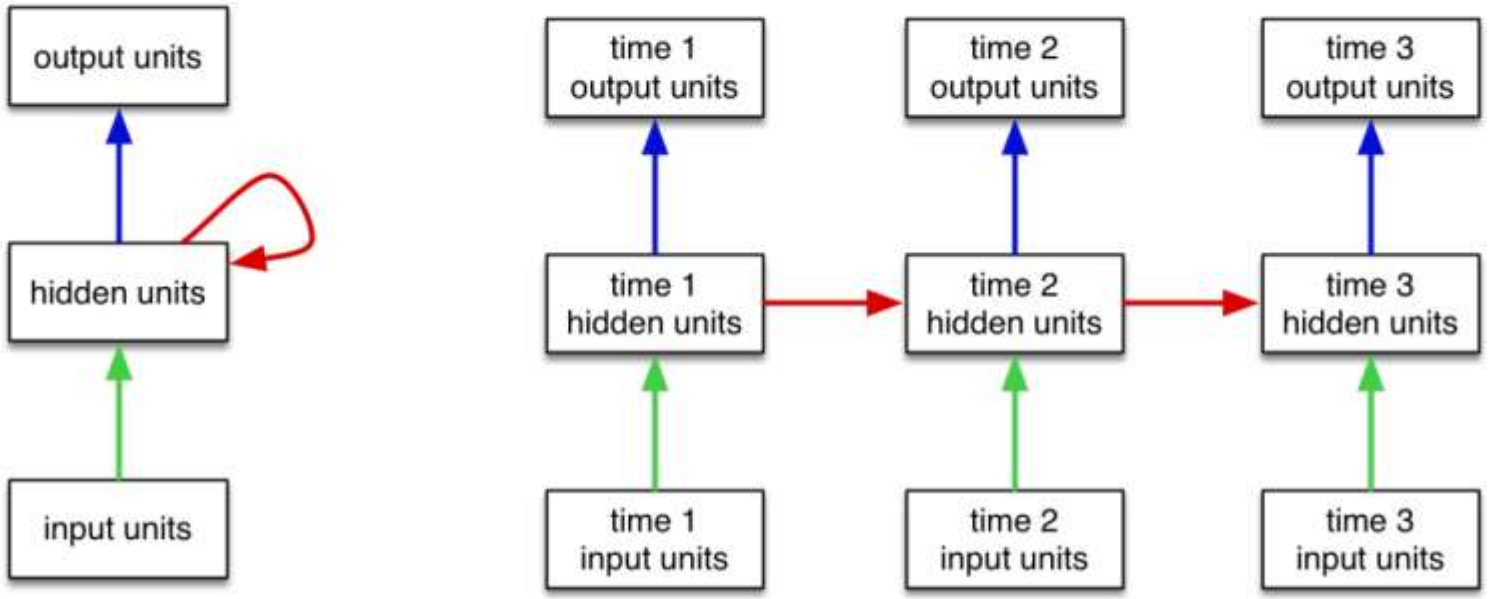
# Introduction

- 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,

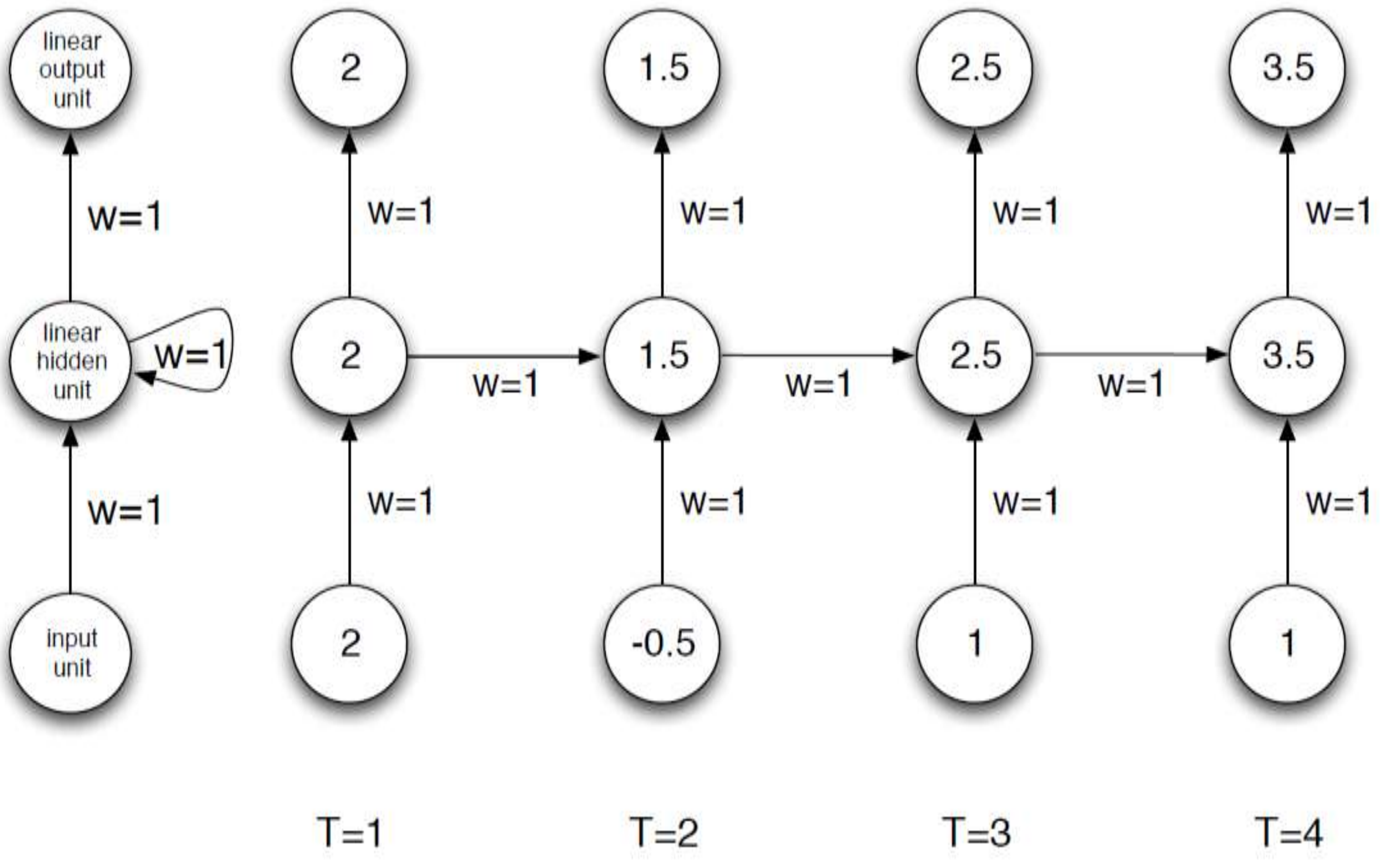
# Recurrent Neural Network

- 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



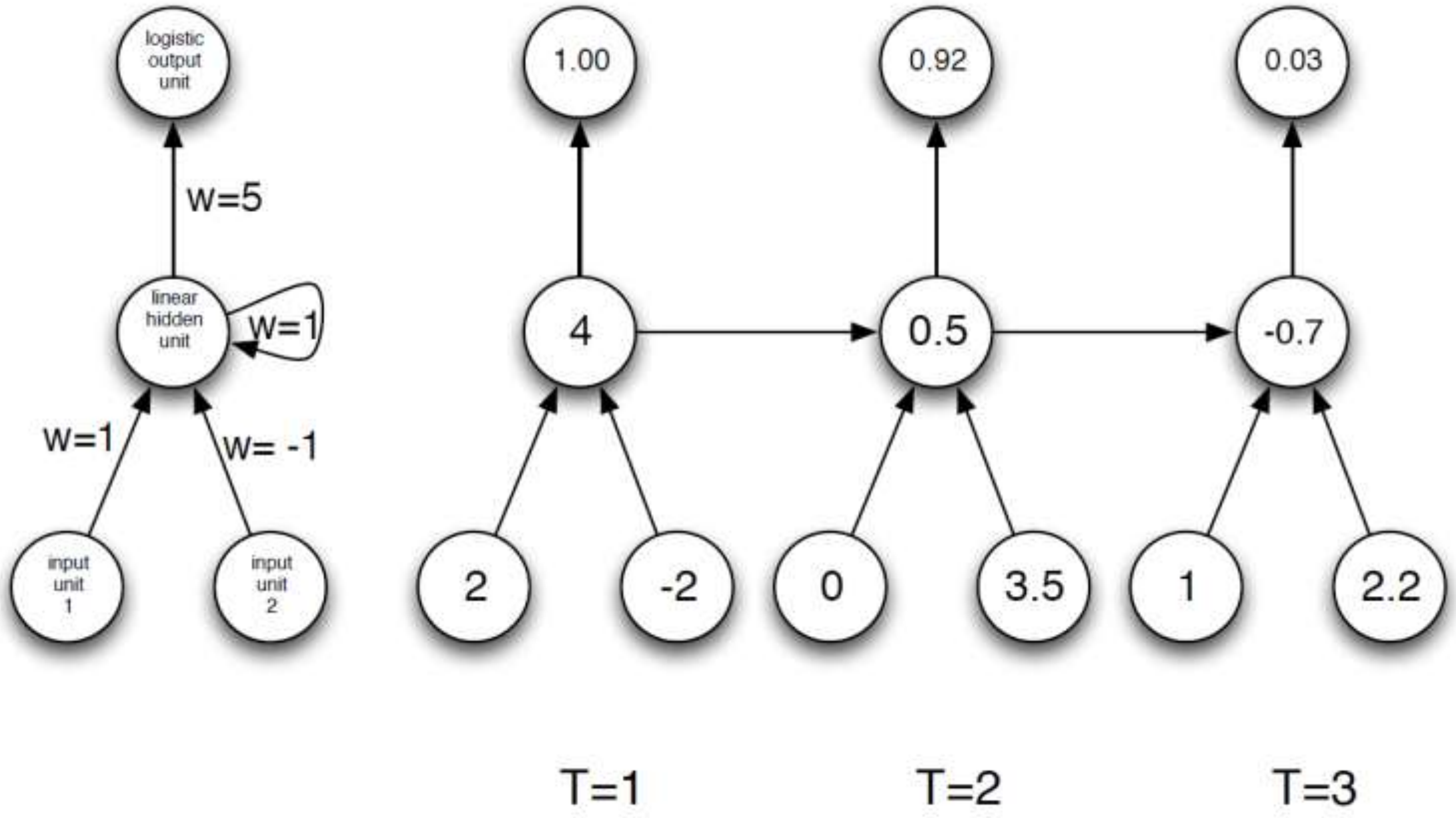
# RNN examples

- Summation network



# RNN examples

- Summation & comparison network



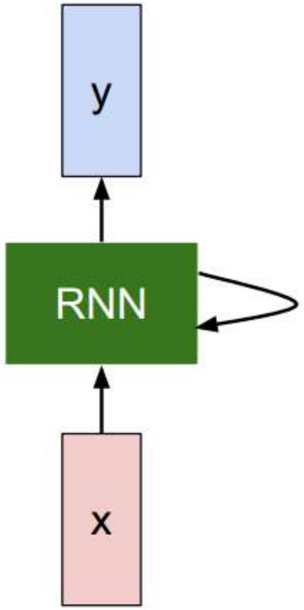
# Recurrent Neural Network



- General formulation

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

$$\underbrace{\boxed{h_t}}_{\text{new state}} = \underbrace{\boxed{f_W}}_{\text{some function with parameters W}}(\underbrace{\boxed{h_{t-1}}}_{\text{old state}}, \underbrace{\boxed{x_t}}_{\text{input vector at some time step}})$$



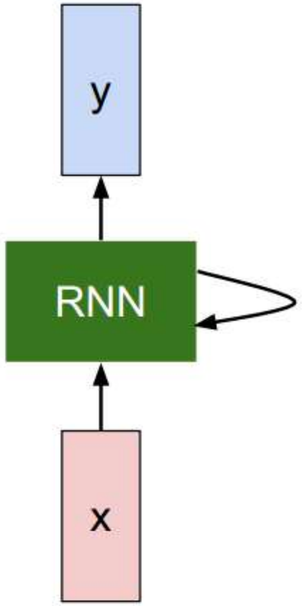
# Recurrent Neural Network

- General formulation

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

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

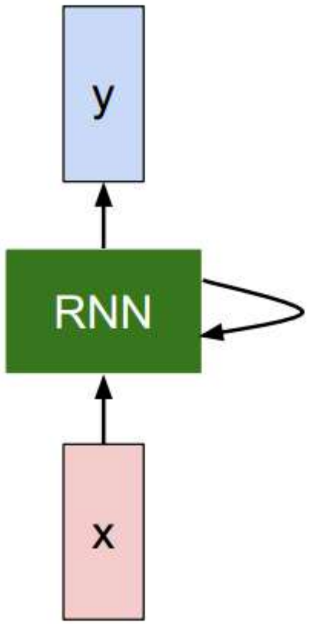
Notice: the same function and the same set of parameters are used at every time step.



# (Vanilla)Recurrent Neural Network

- General formulation

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



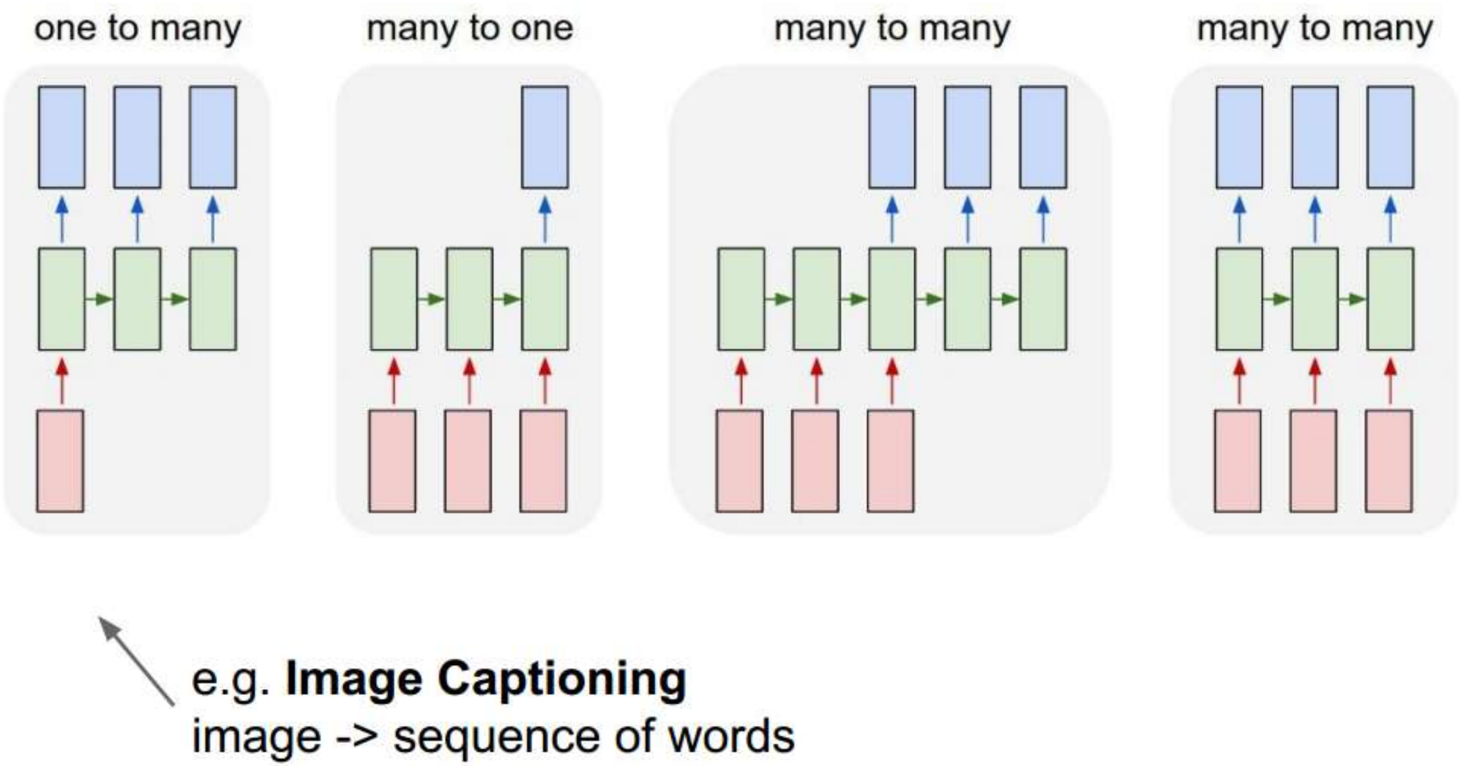

$$h_t = f_W(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$

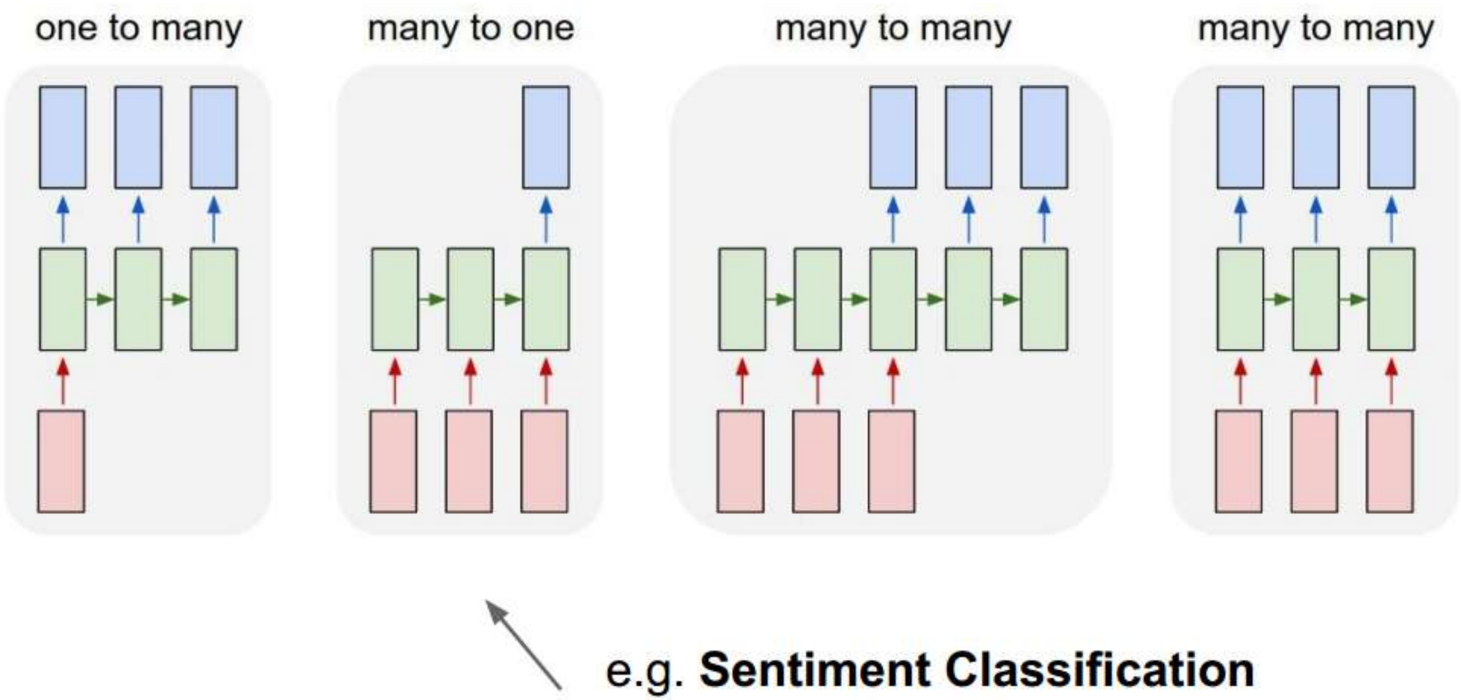
# Recurrent Neural Network

- Recurrent Neural Networks: model variants



# Recurrent Neural Network

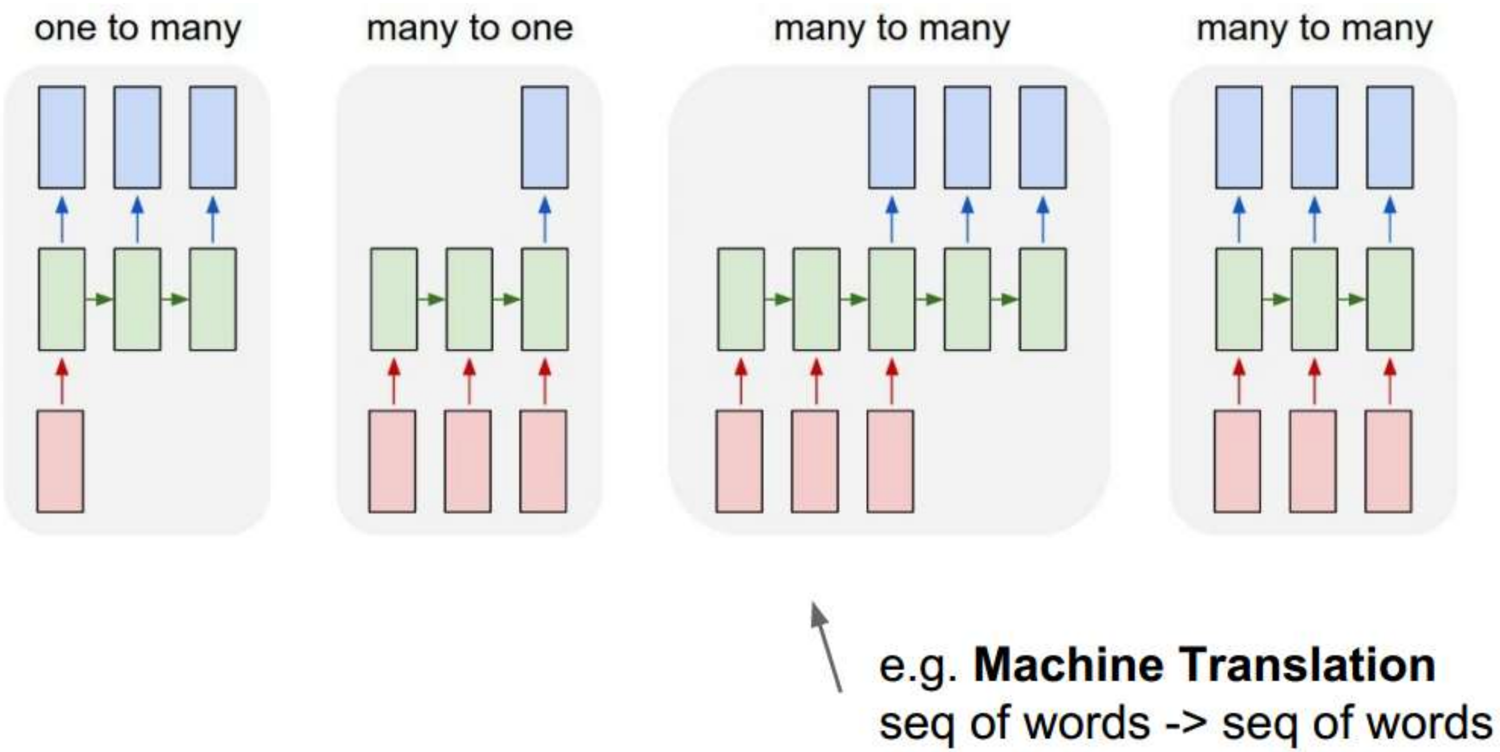
- Recurrent Neural Networks: model variants




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

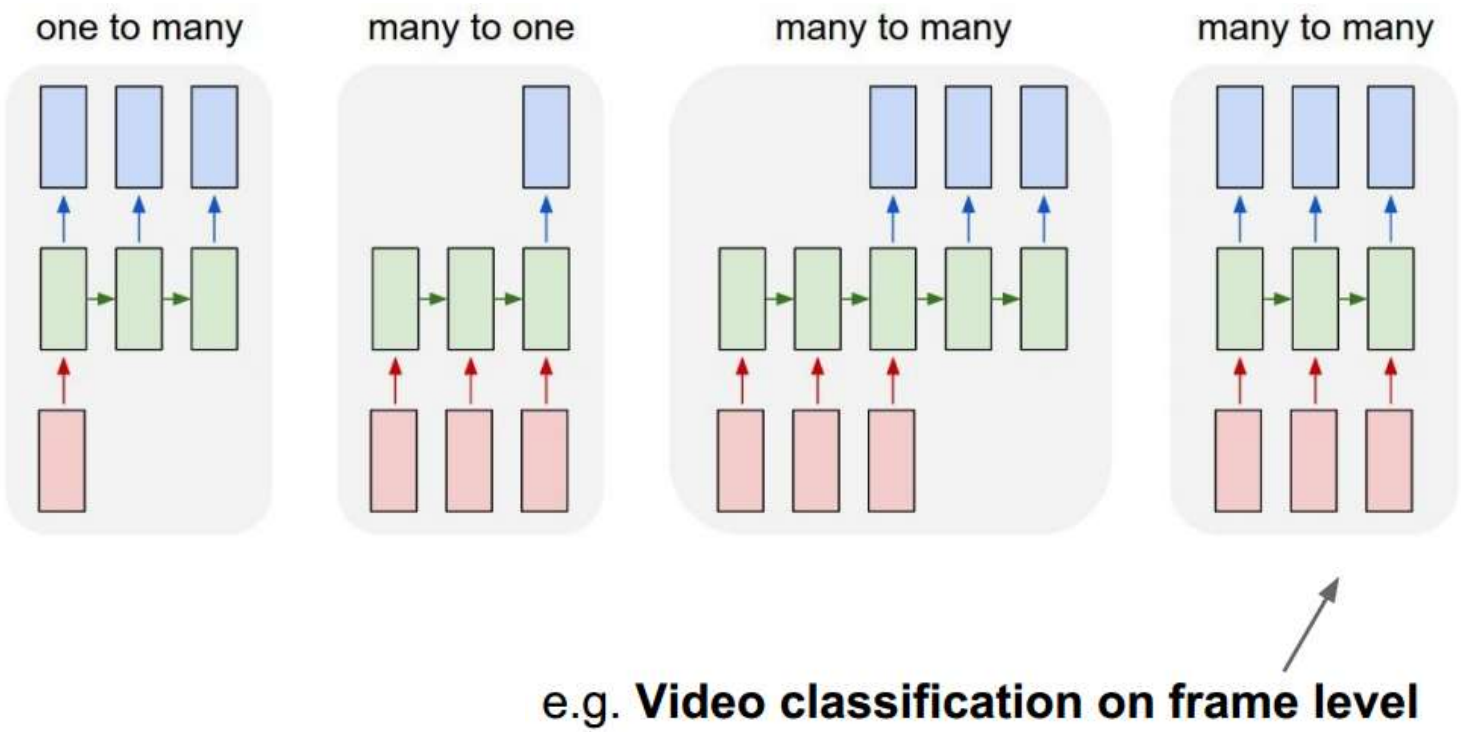
# Recurrent Neural Network

- Recurrent Neural Networks: model variants



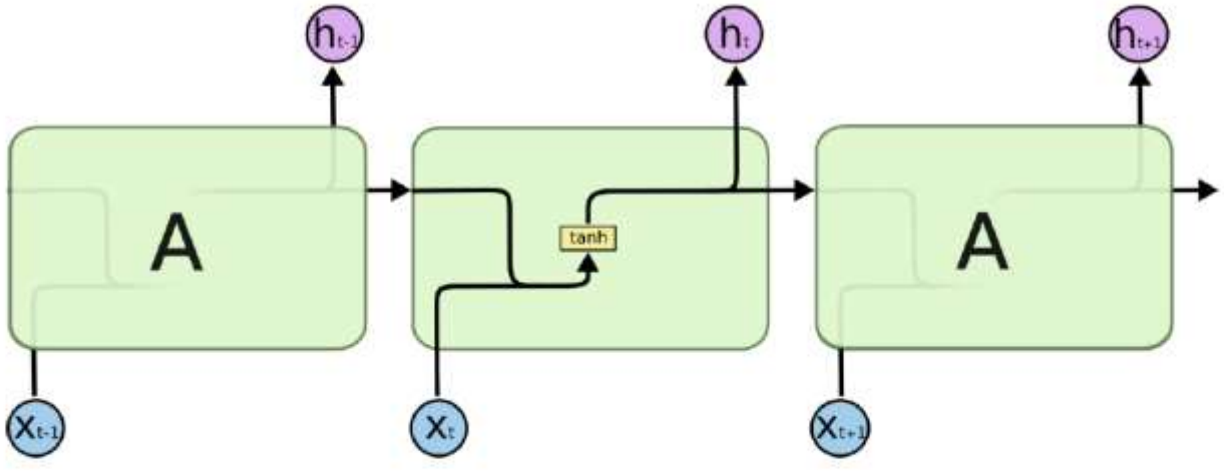
# Recurrent Neural Network

- Recurrent Neural Networks: model variants



# Standard RNN

- Recall

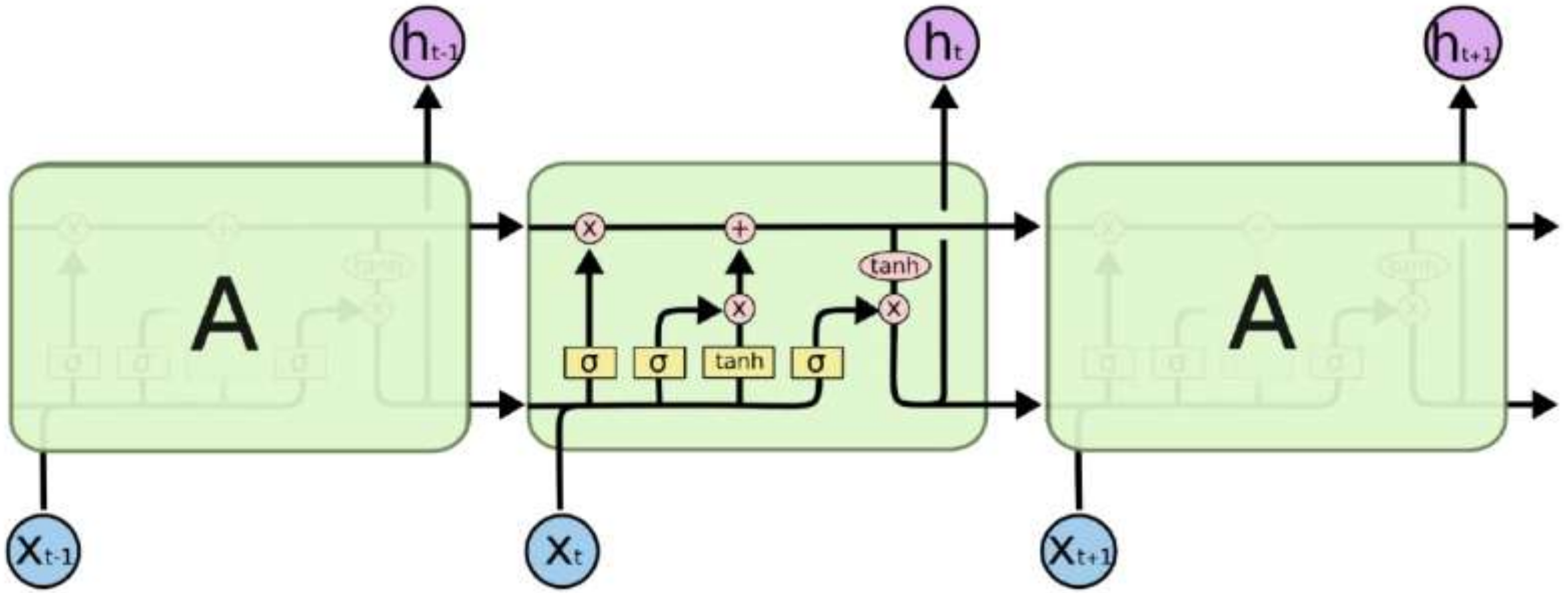



- Each recurrent neuron receives past outputs and current input
- Pass through a tanh function

# Long Short Term Memory(LSTM)

上海科技大学
ShanghaiTech University

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

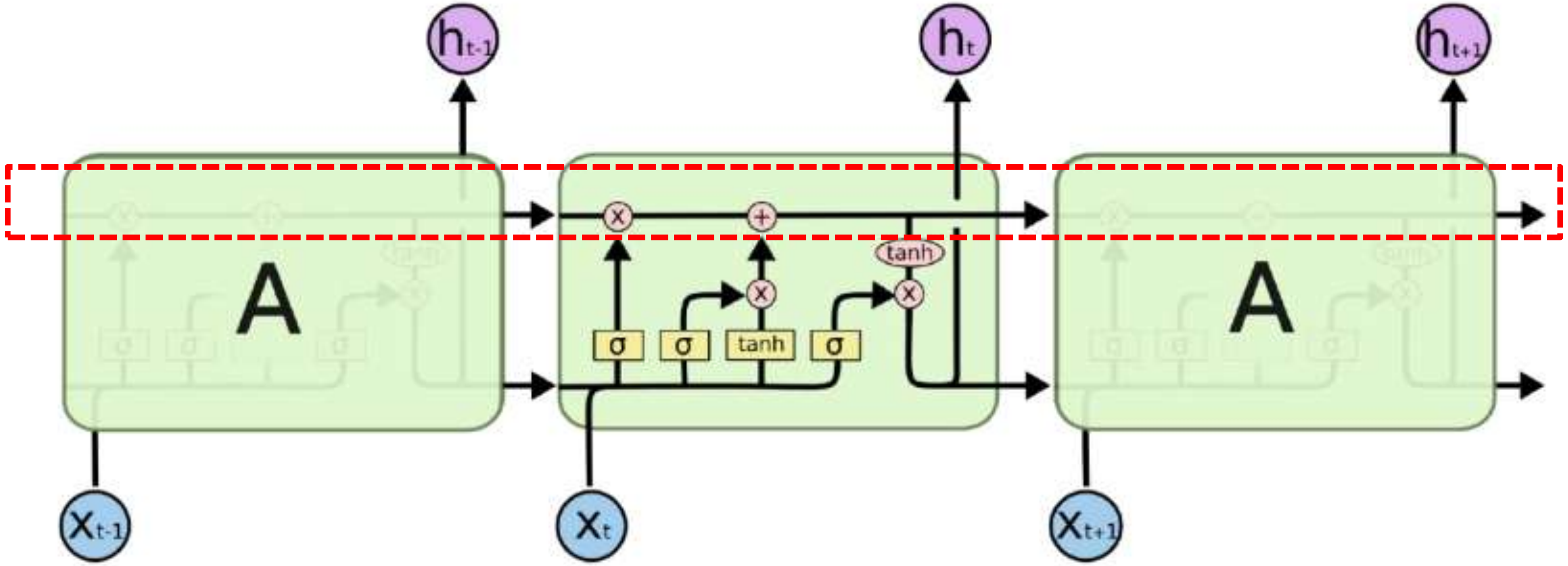



Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation

# Long Short Term Memory(LSTM)

上海科技大学 ShanghaiTech University

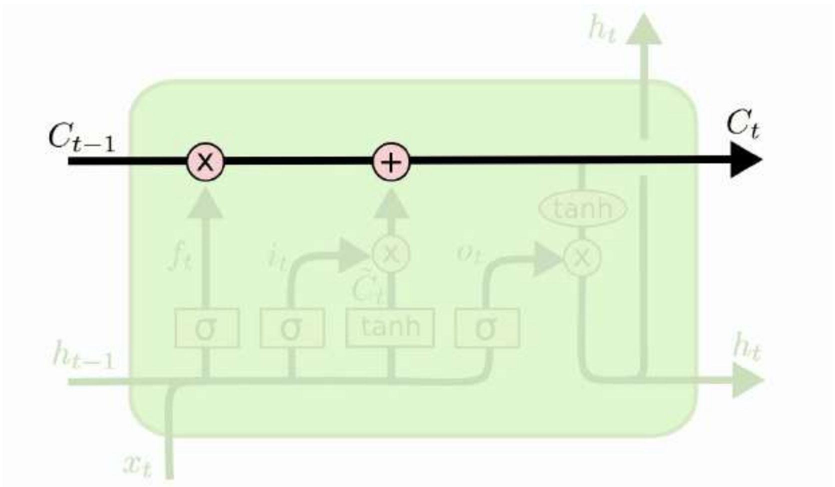
- Key component: a remembered cell state




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

# LSTM: cell state

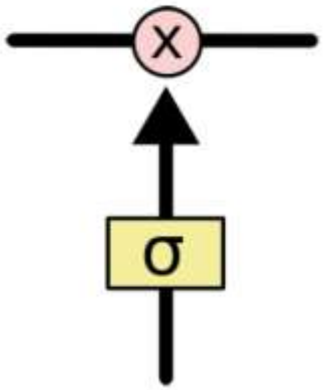
- A linear history
  - Carries information through
  - Only affected by a gate and addition of current information, which is also gated



# LSTM: gates

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

- Controls how much of the information will be let through

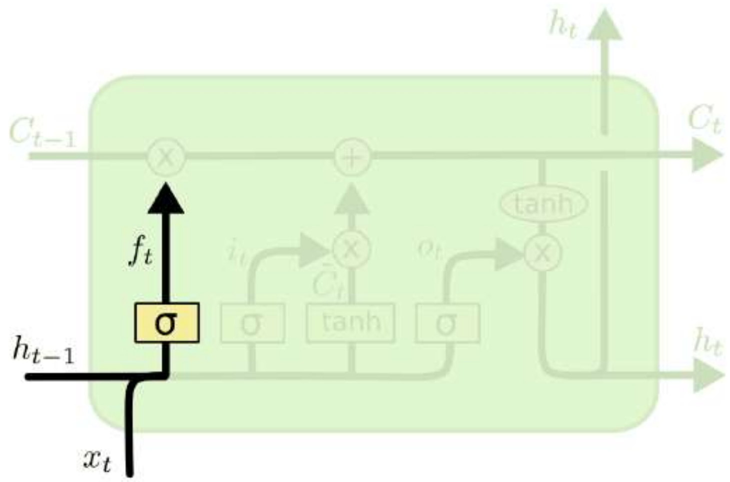



- Three gates
  - Forget gate
  - Input gate
  - Output gate

# 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

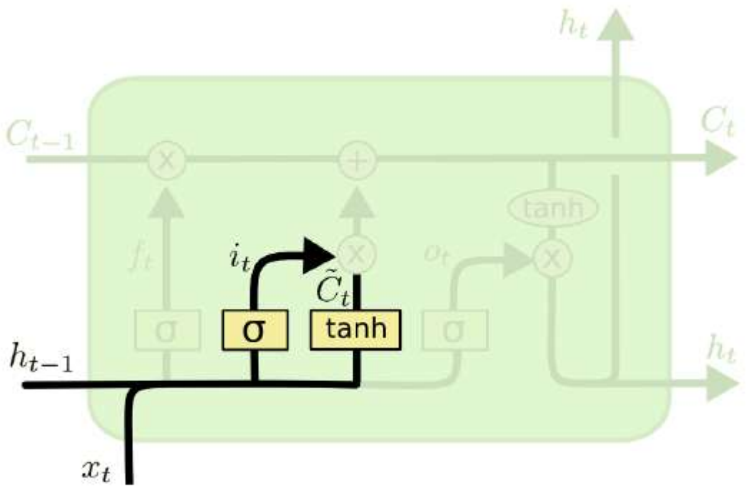
$$\langle h_{t-1}, C_{t-1} \rangle$$




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

# 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

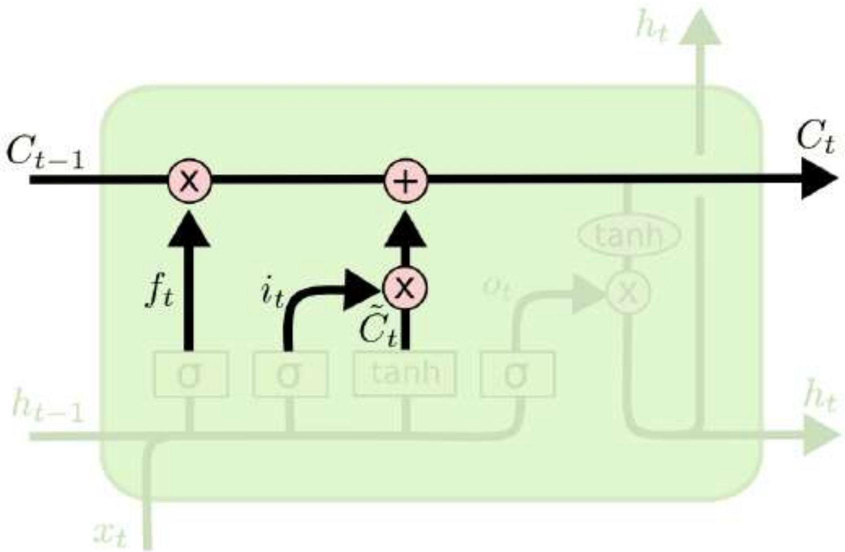



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

# LSTM: Memory cell update

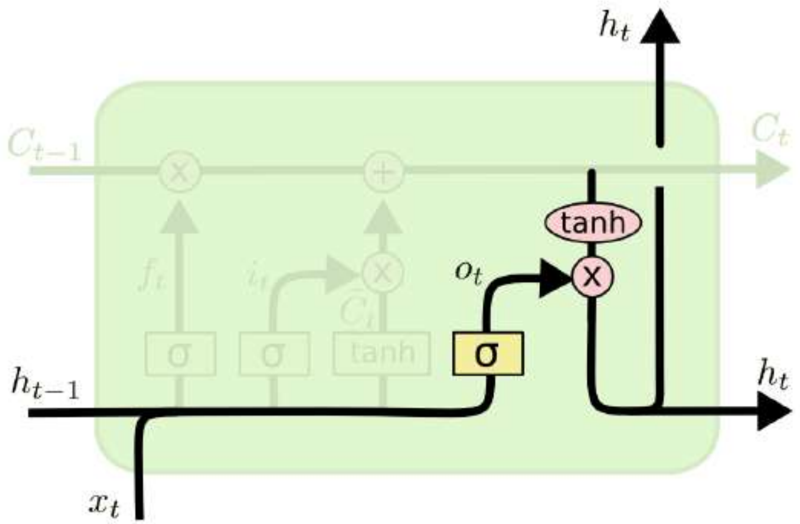
- The output of the second part is added into the current memory cell
  - Can be viewed as value update in a key-value pair
  - The input and state jointly decide how much of history info is kept and how much of embedded input info is added
  - A dynamic mixture of experts at each time step




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

# LSTM: 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



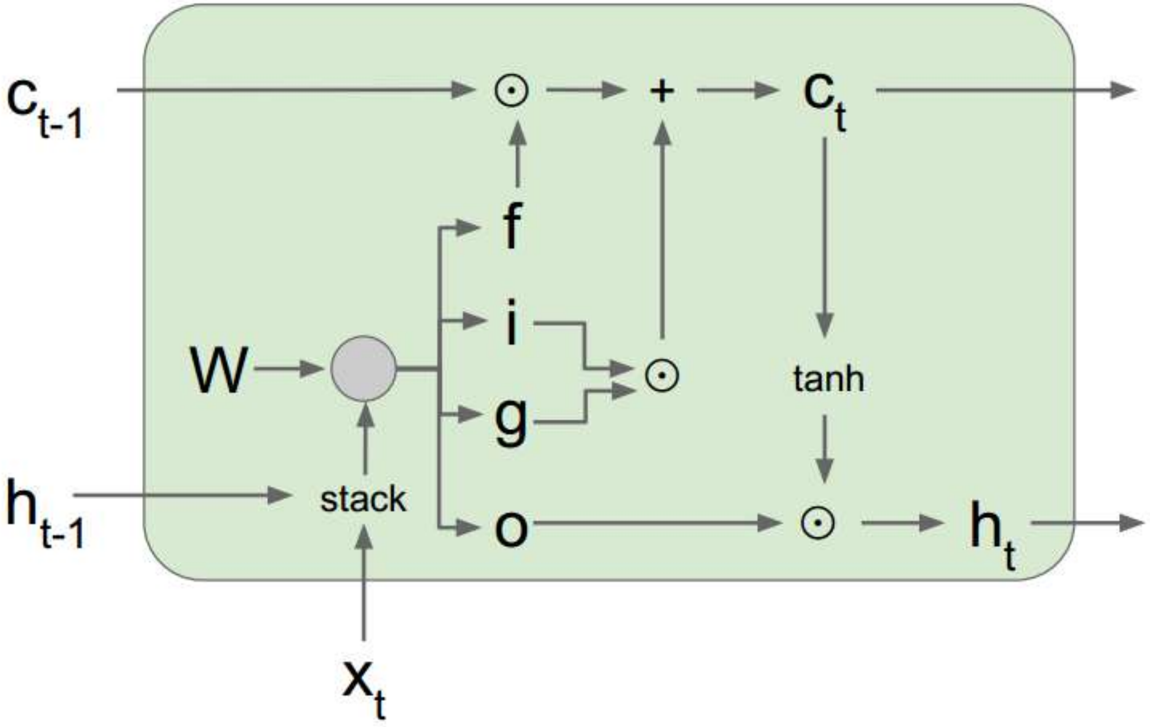

$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

# Long Short Term Memory(LSTM)

上海科技大学 ShanghaiTech University

*[Hochreiter et al., 1997]*




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

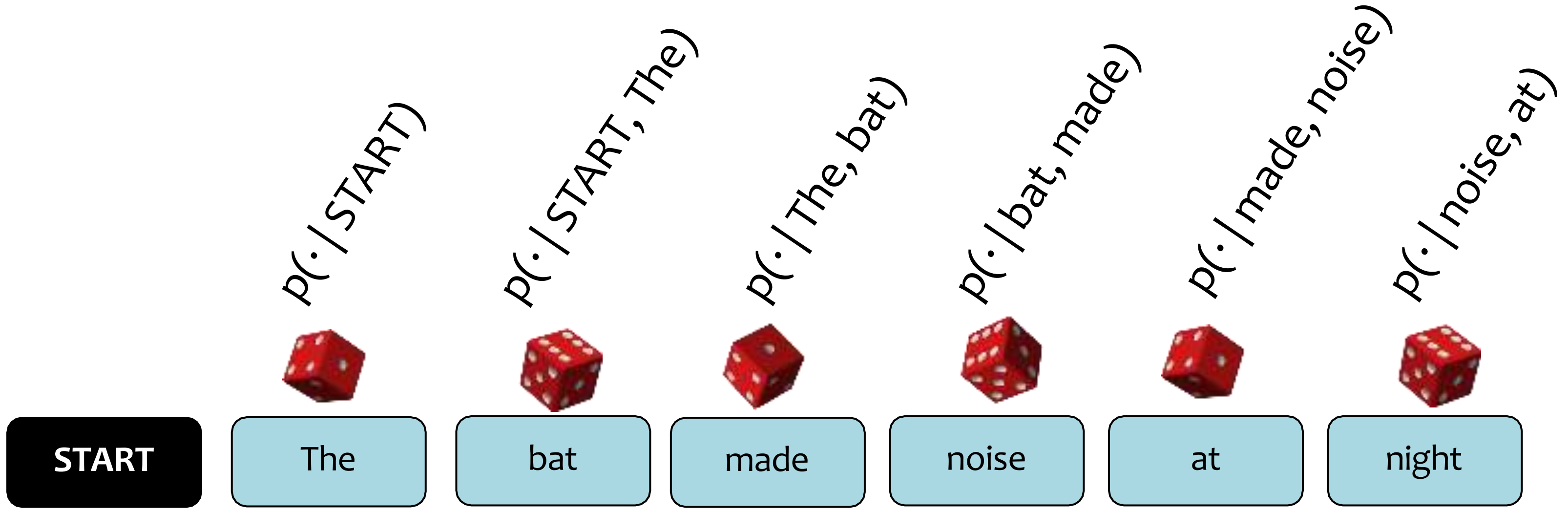
$$c_t = f \odot c_{t-1} + i \odot g$$

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

# BACKGROUND:
# N-GRAM LANGUAGE MODELS

# n-Gram Language Model

- *<u>Goal</u>*: Generate realistic looking sentences in a human language
- *<u>Key Idea</u>*: condition on the last n-1 words to sample the $n^{th}$ 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

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 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



- *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

*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

上海科技大学 ShanghaiTech University

*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**?...

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

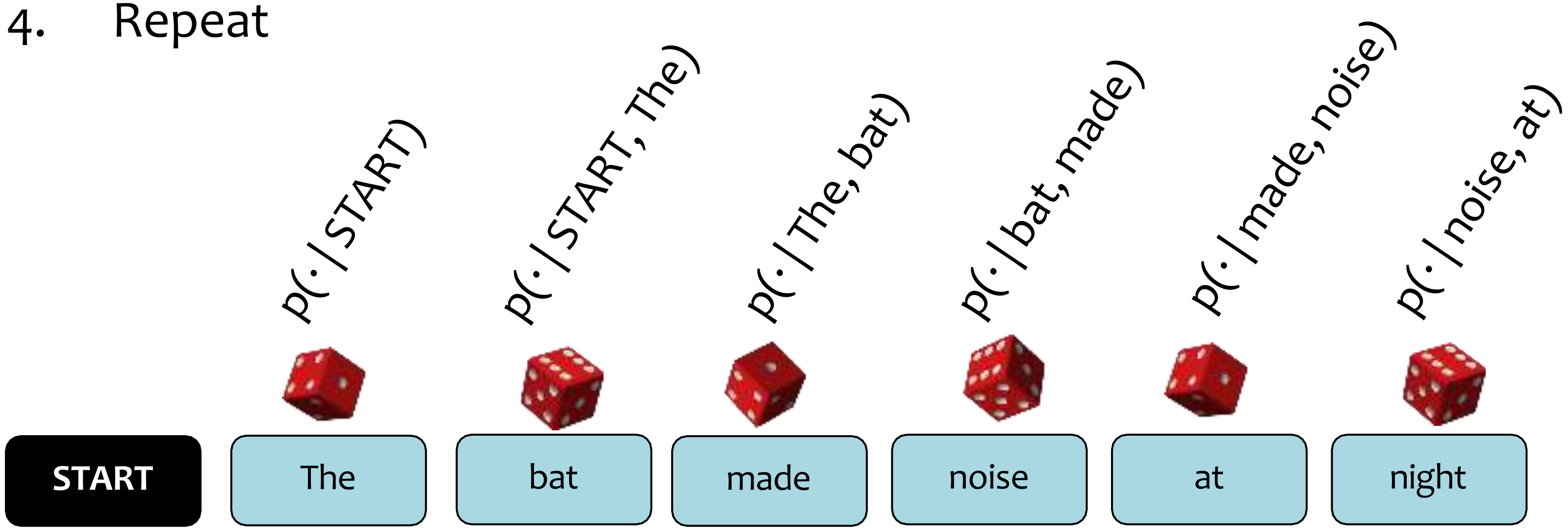
| $w_t$ | $p(\cdot \mid \cdot, \cdot)$ |
|---|---|
| corn | 4/11 |
| grass | 3/11 |
| hay | 2/11 |
| if | 1/11 |
| which | 1/11 |

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

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

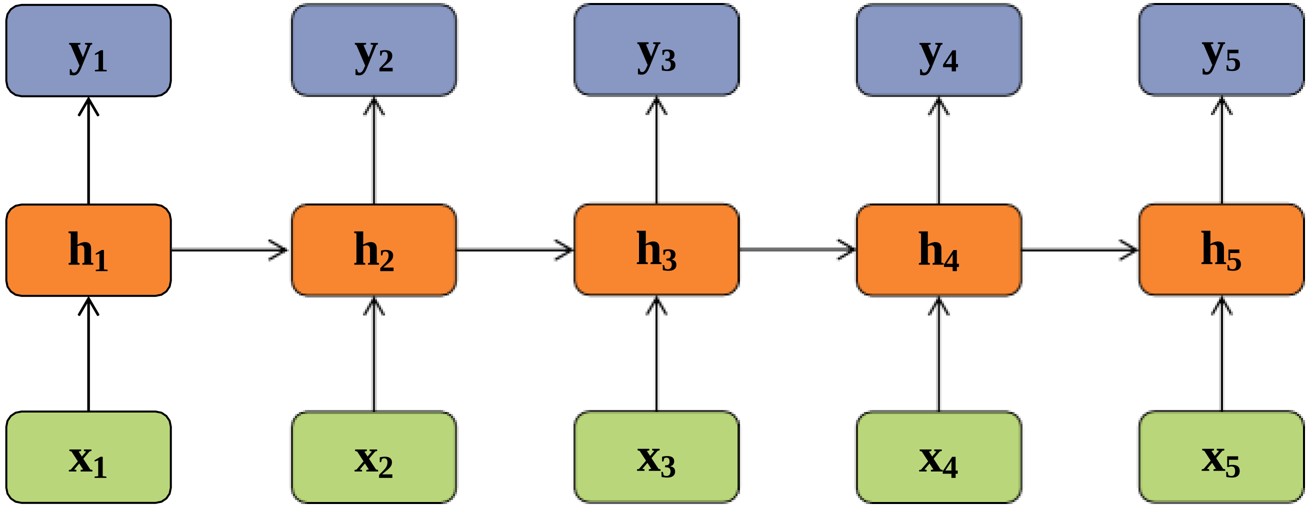
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathrm{R}^K$

nonlinearity: $\mathrm{H}$

Definition of the RNN:

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

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



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 $\quad p(w_1)$

The $\quad p(w_2 \mid w_1)$

The

The

The $\quad \ldots w_1)$

The $\quad p(w_6 \mid w_5, w_4, w_3, w_2, w_1)$

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

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

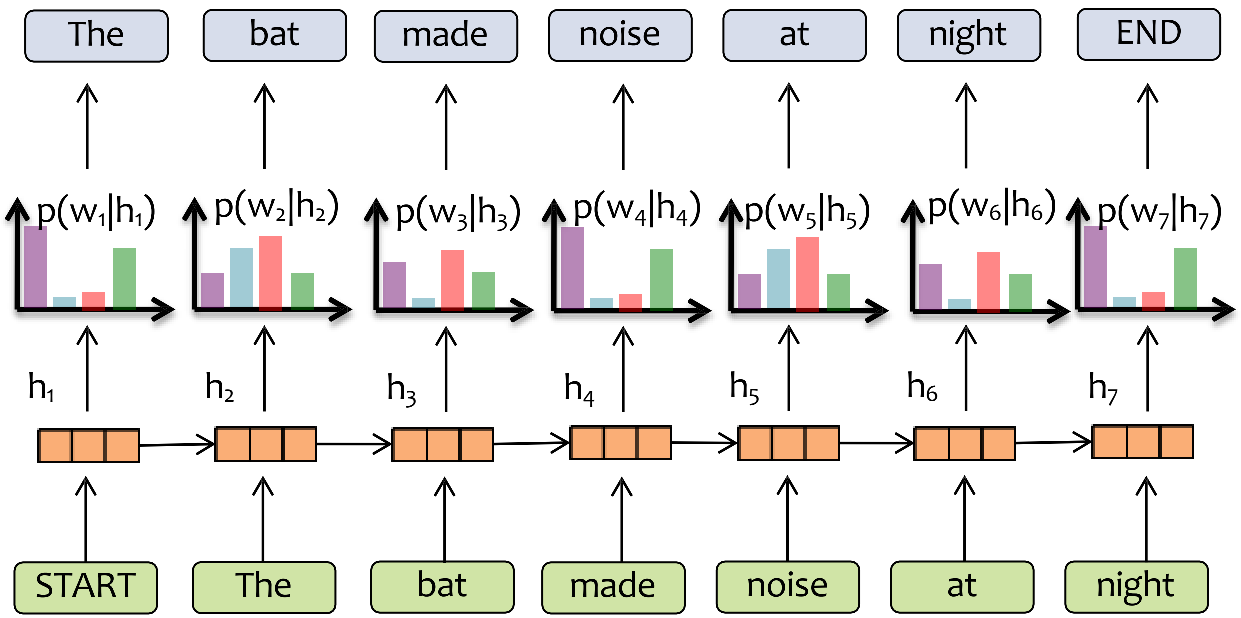
| | | | | | | |
|---|---|---|---|---|---|---|
| The | | | | | | $p(w_1)$ |
| The | bat | | | | | $p(w_2 \mid f_\theta(w_1))$ |
| The | bat | made | | | | $p(w_3 \mid f_\theta(w_2, w_1))$ |
| The | bat | made | noise | | | $p(w_4 \mid f_\theta(w_3, w_2, w_1))$ |
| The | bat | made | noise | at | | $p(w_5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

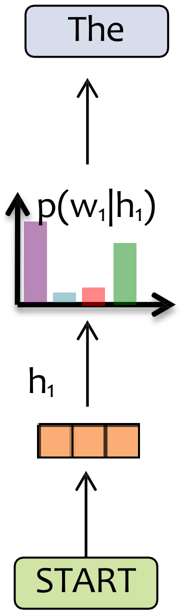



*Key Idea*:

(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



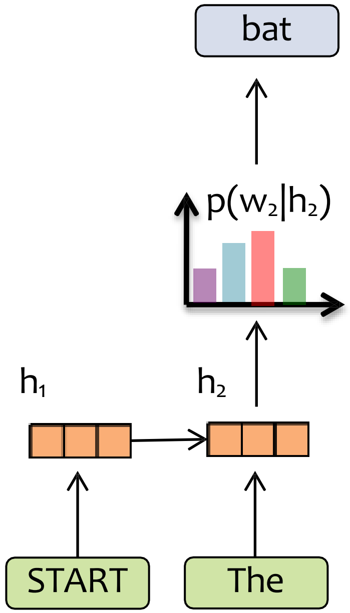

*Key Idea*:

(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





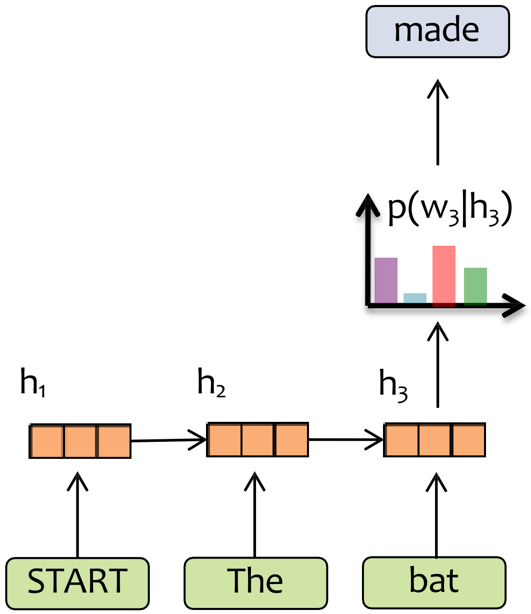

*Key Idea*:

(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

上海科技大学 ShanghaiTech University

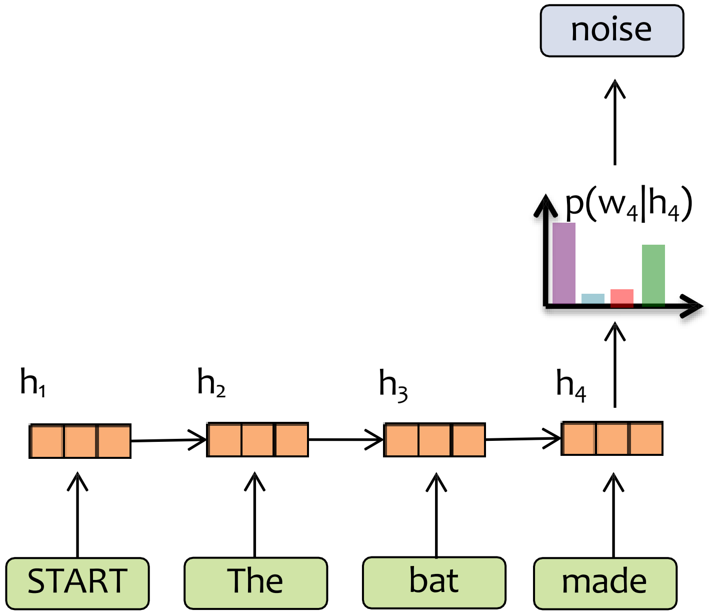



*<u>Key Idea</u>*:

(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

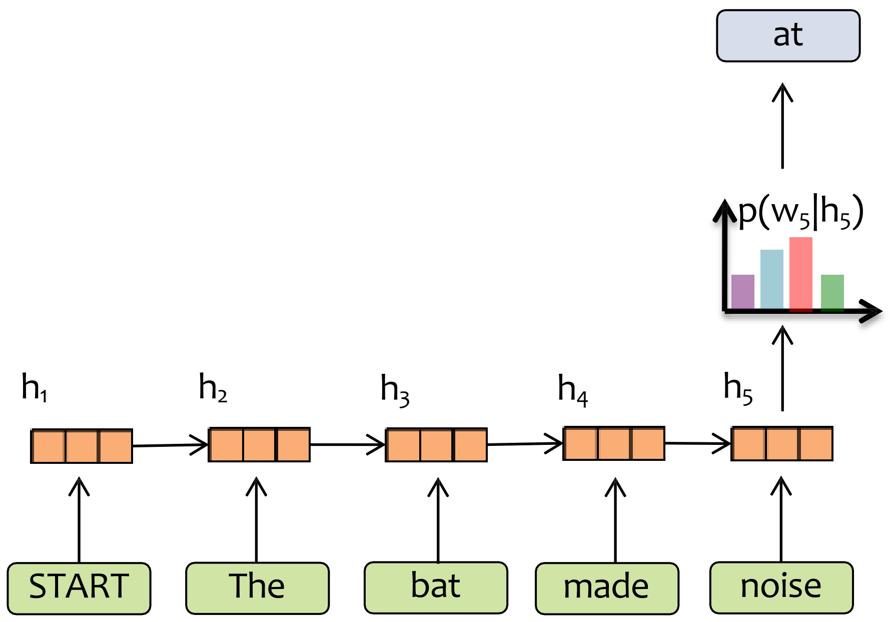



*Key Idea*:

(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>*:

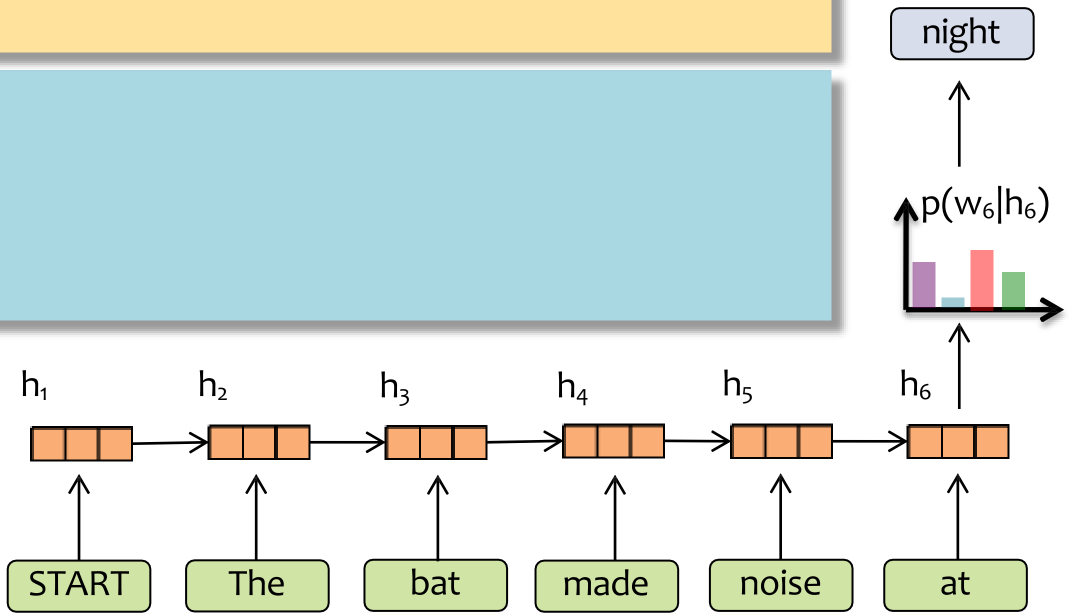
(1) convert all previous words to a **fixed length vector**

(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

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

**Answer:**

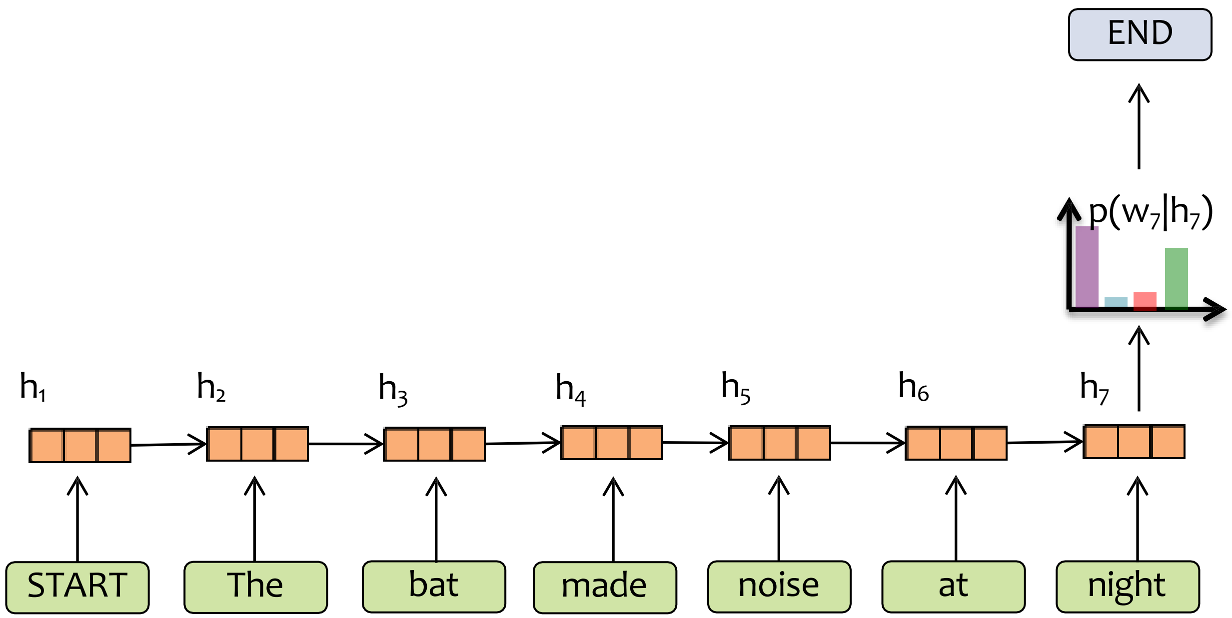



*<u>Key Idea</u>*:

(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

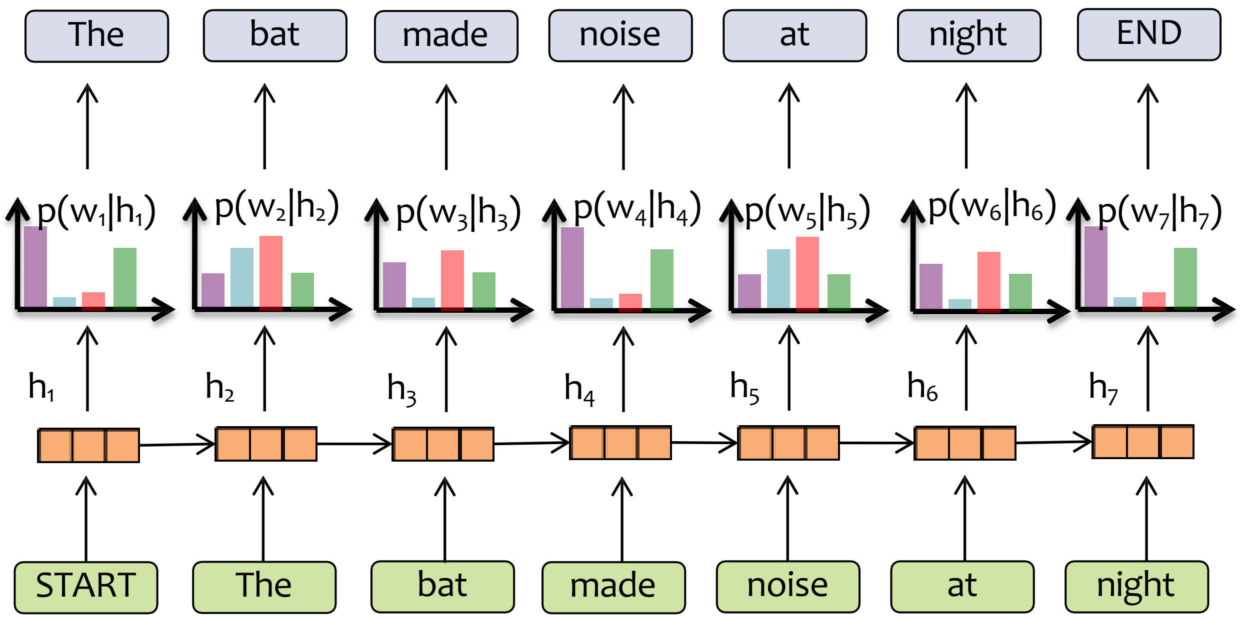



*Key Idea*:

(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



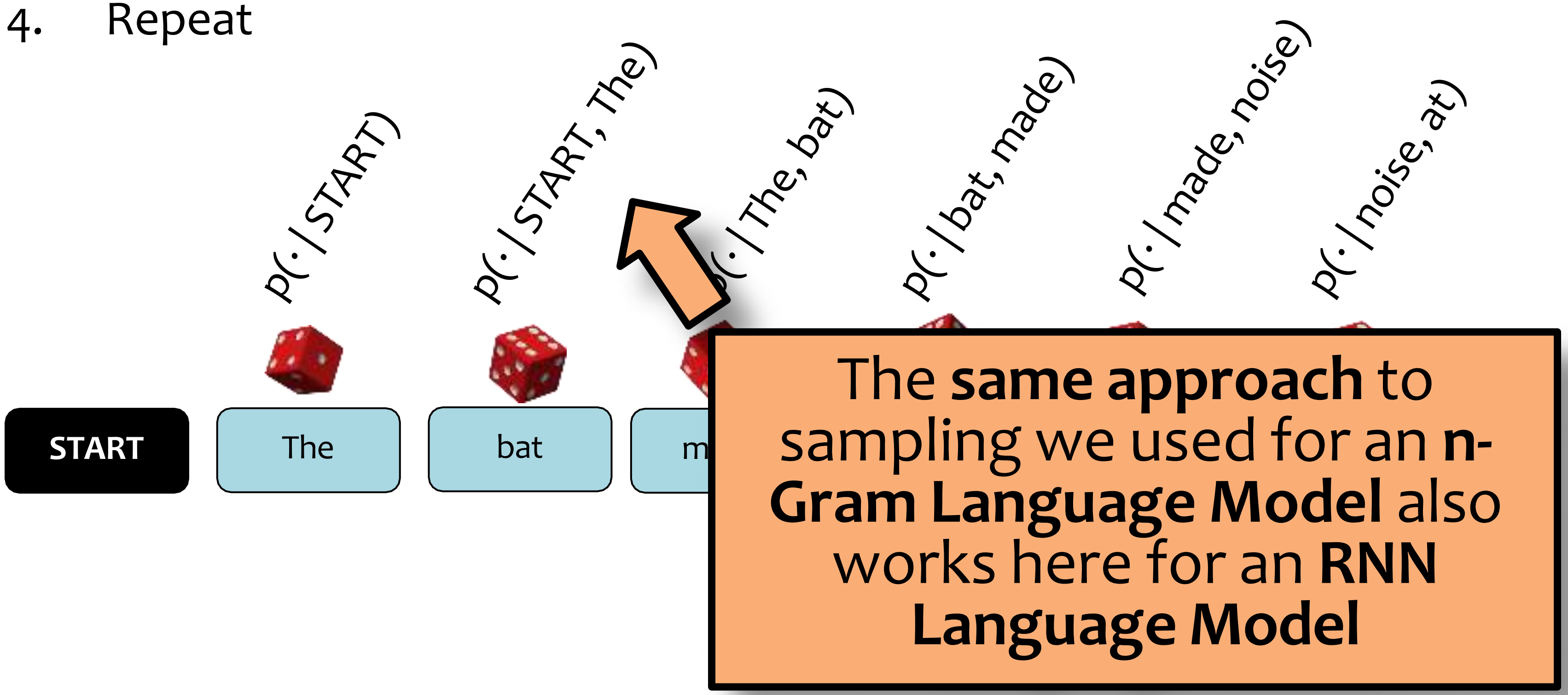

$$p(w_1, w_2, w_3, \ldots, w_T) = p(w_1 \mid h_1)\, p(w_2 \mid h_2) \ldots\, p(w_2 \mid h_T)$$

# 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**

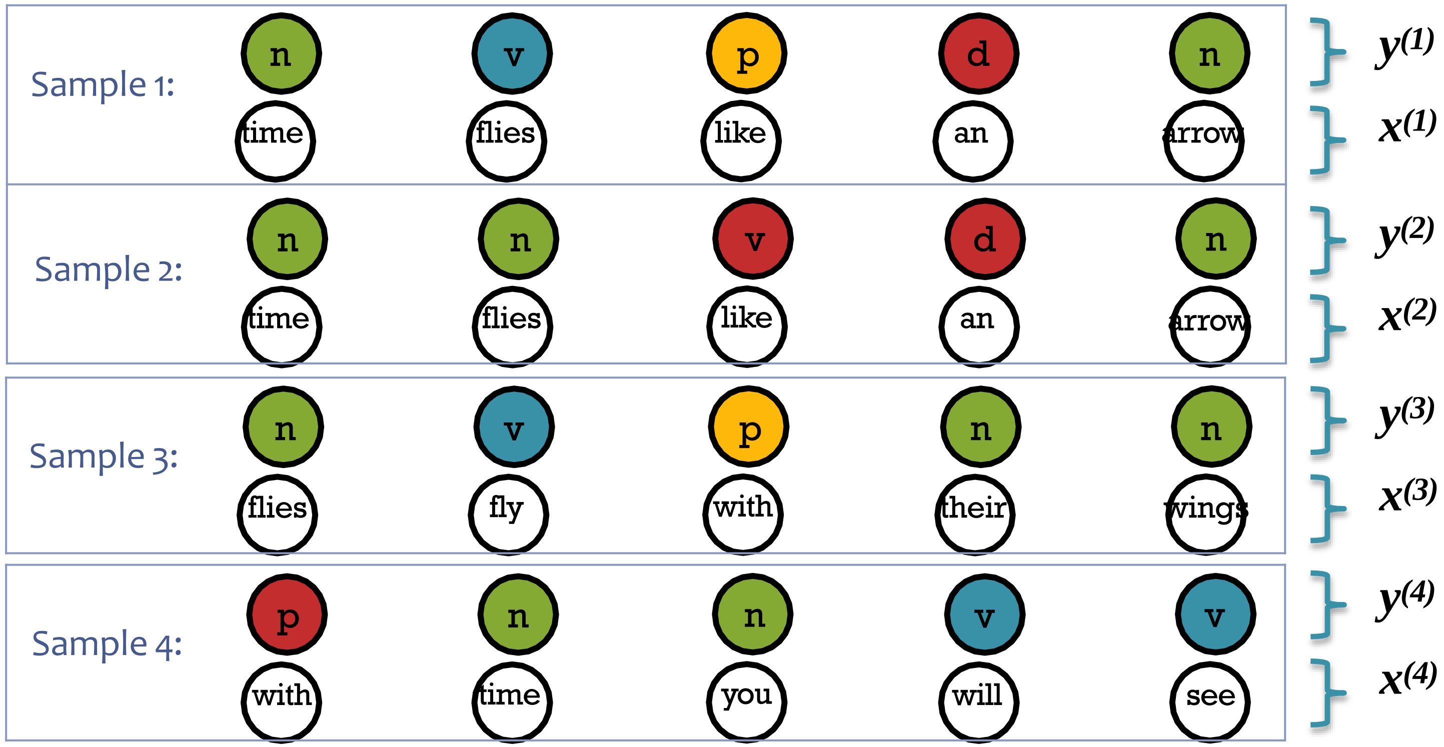
# LEARNING AN RNN


47

Recall...

# 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

**Algorithm 1** SGD

Initialize $\theta^{(0)}$

$s = 0$

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

  **for** $i \in \text{shuffle}(1, \ldots, N)$ **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

Recall...

**Algorithm 1** Mini-Batch SGD

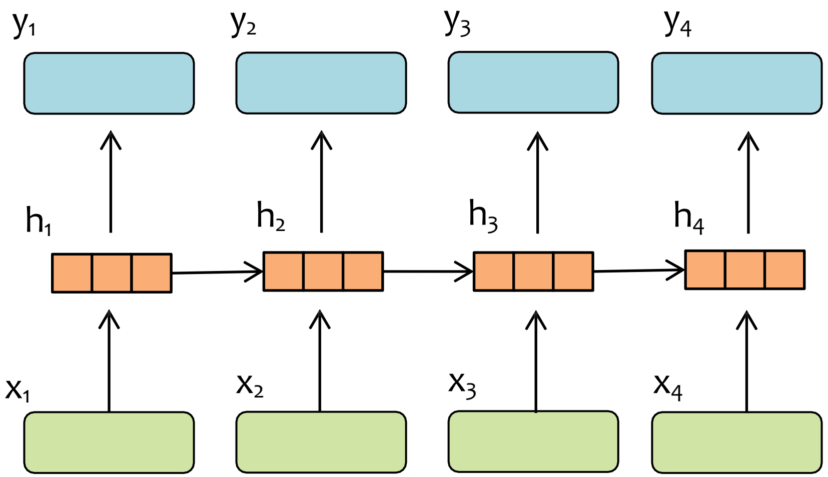
Initialize $\theta^{(0)}$
Divide examples $\{1, \ldots, N\}$ randomly into batches $\{I_1, \ldots, I_B\}$
where $\bigcup_{b=1}^{B} I_b = \{1, \ldots, N\}$ and $\bigcap_{b=1}^{B} I_b = \emptyset$
$s = 0$
**for** $t = 1, 2, \ldots, T$ **do**
    **for** $b = 1, 2, \ldots, B$ **do**
        Select the next batch $I_b$, where $m = |I_b|$
        Compute the gradient $g^{(s)} = \frac{1}{m} \sum_{i \in I_b} \nabla J_i(\theta^{(s)})$
        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** $\theta^{(s)}$

# RNN

上海科技大学 ShanghaiTech University

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

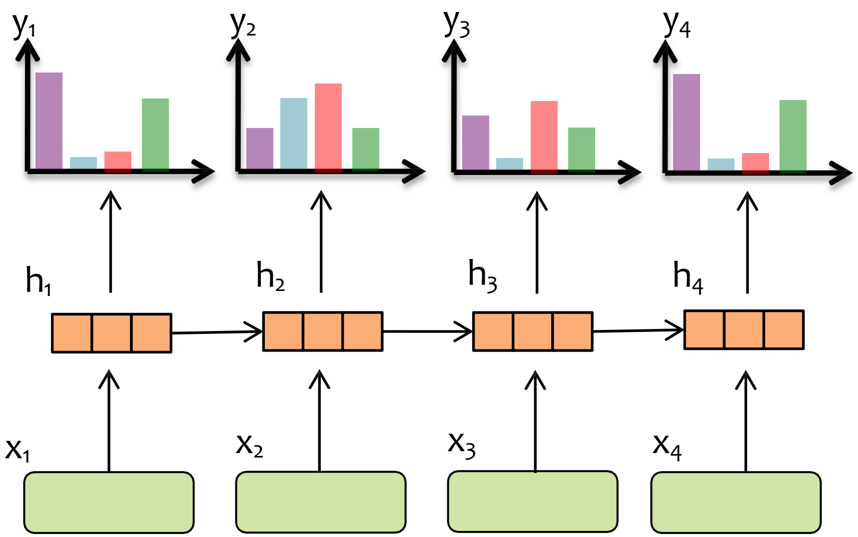




51

# RNN

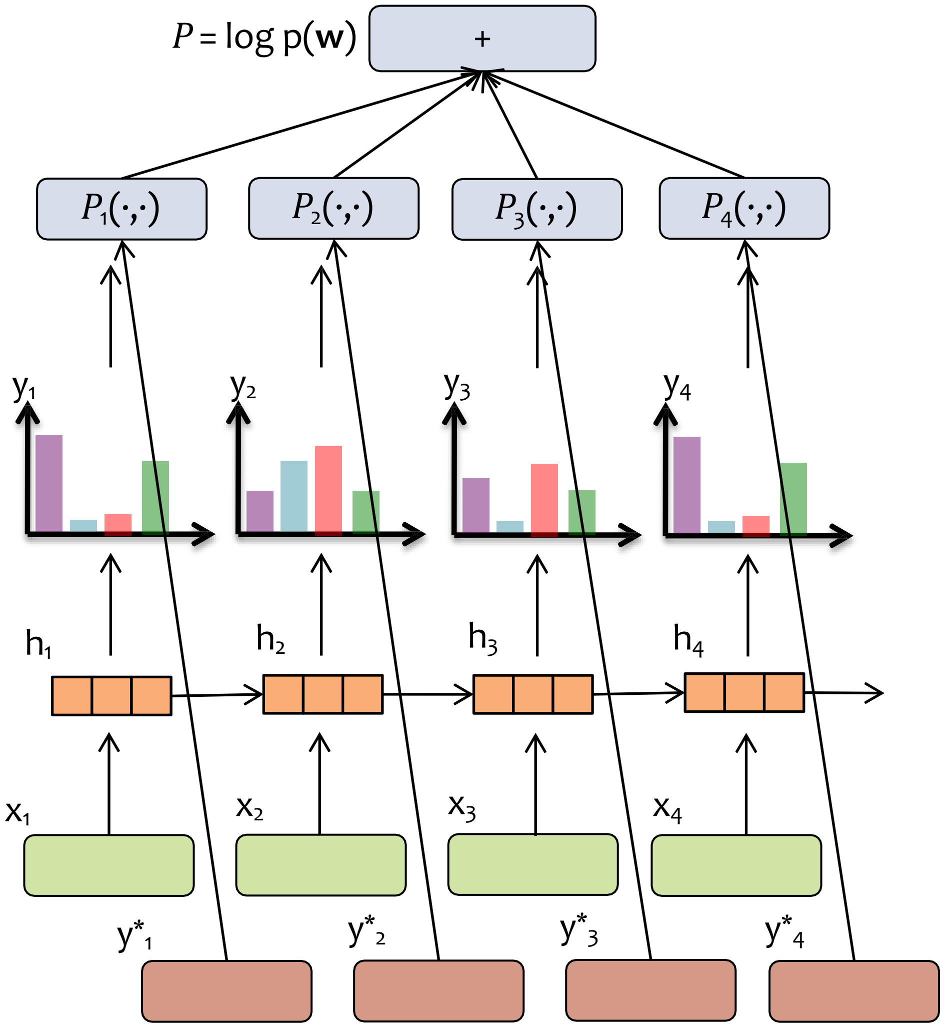





**Algorithm 1** Elman RNN

**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:

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


53

# LEARNING AN RNN-LM

# Learning a Language Model

上海科技大学
ShanghaiTech University

*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 |

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


55

# Learning a Language Model

上海科技大学 ShanghaiTech University

## 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)

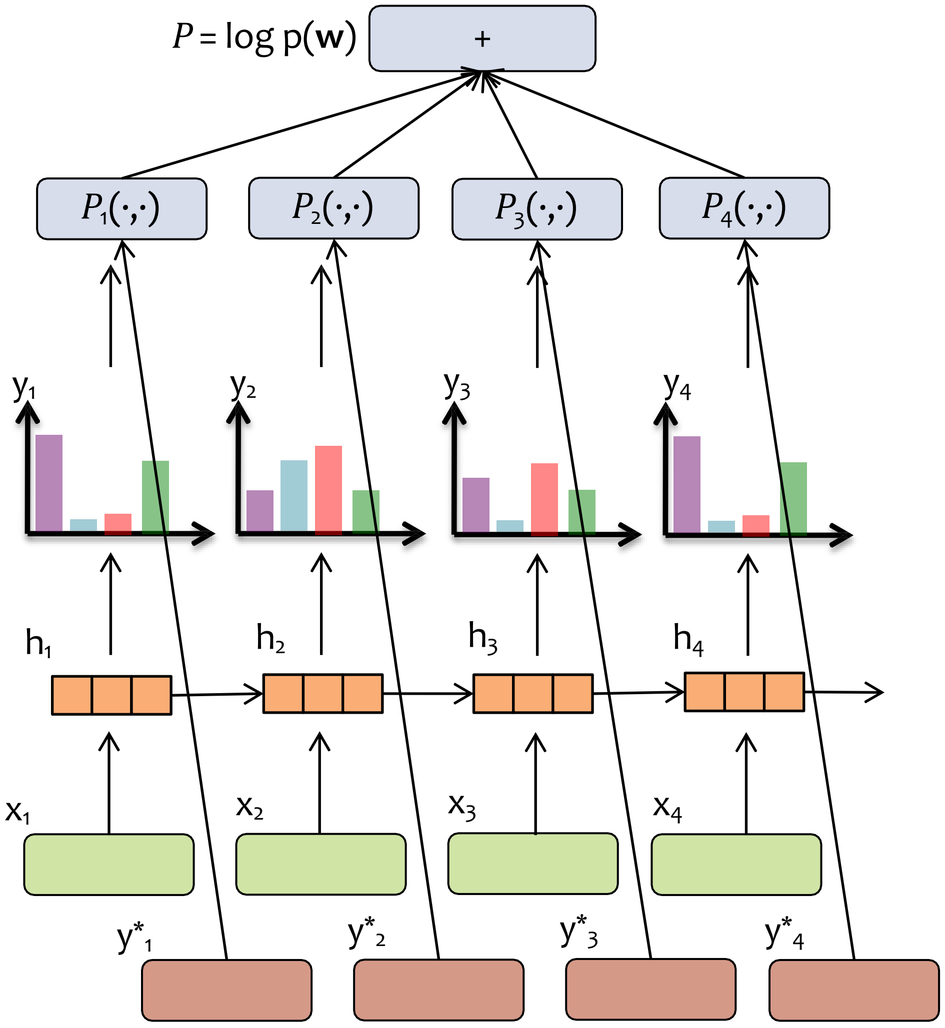
## MLE for n-gram LM

- 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?





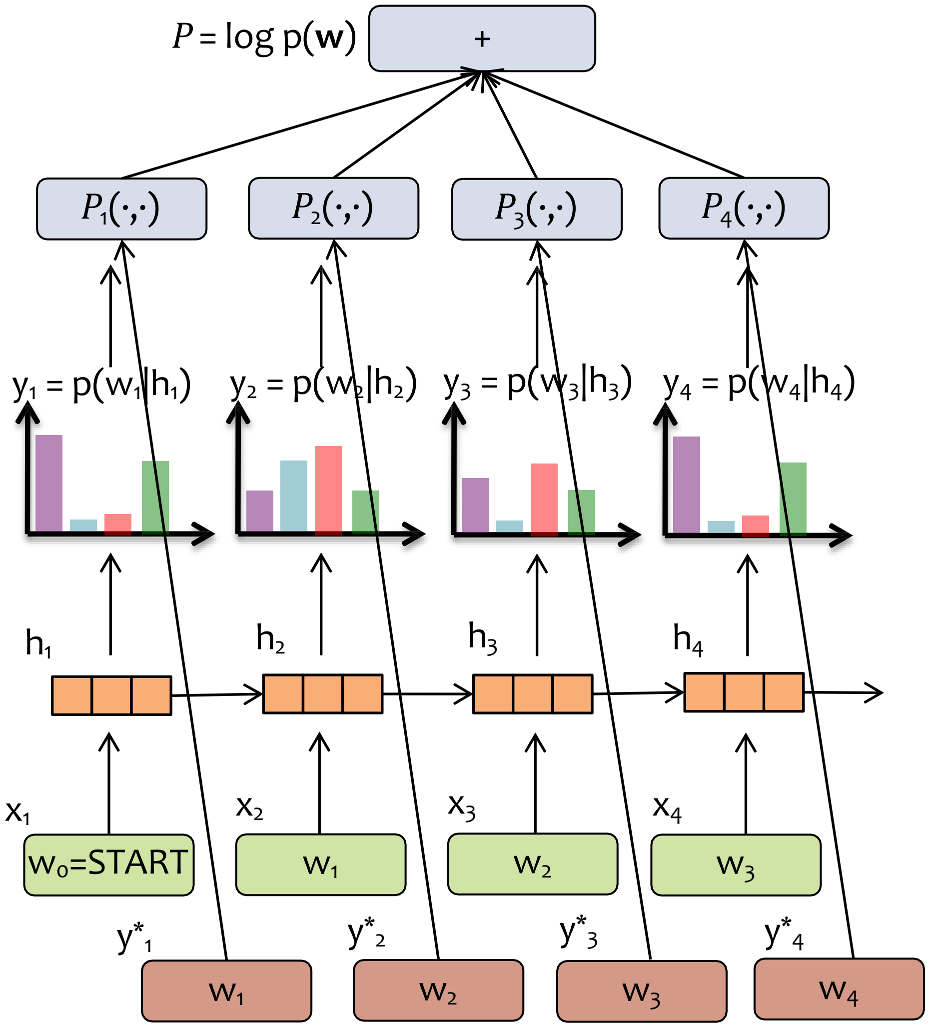

**Algorithm 1** Elman RNN + Loss

**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?

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




**Algorithm 1** Elman RNN + Loss

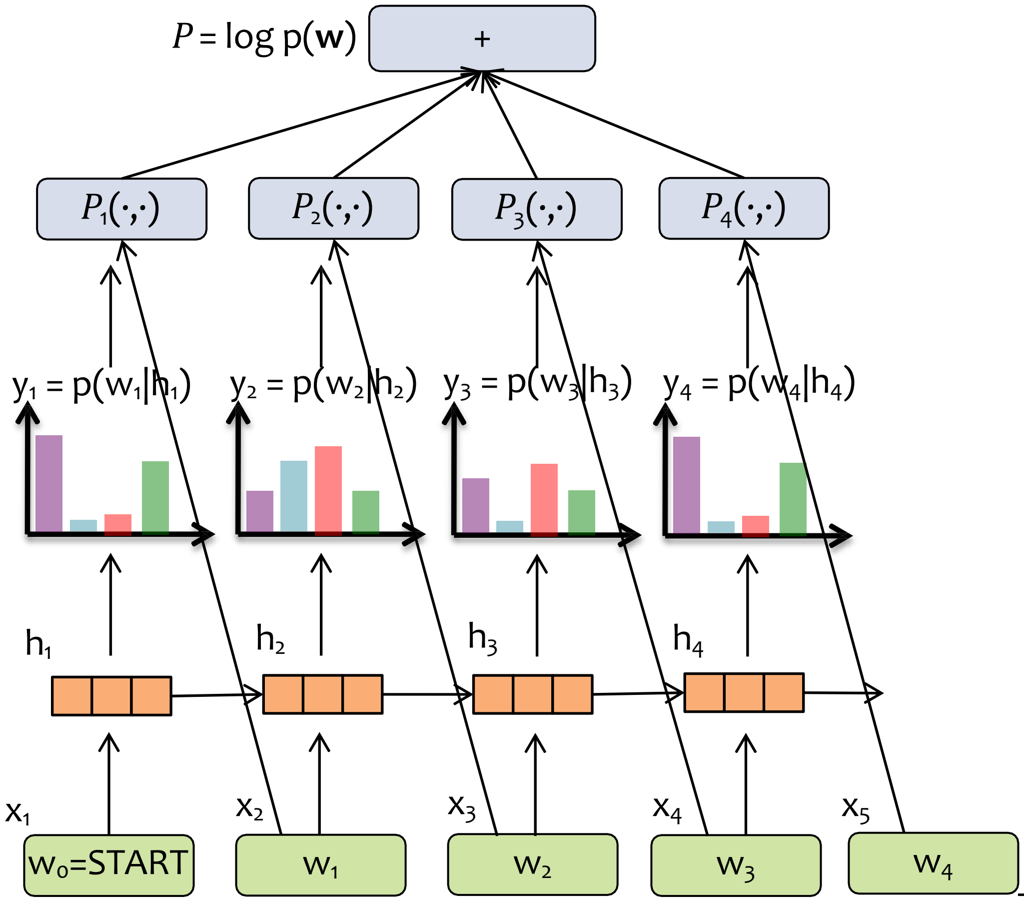
**procedure** FORWARD($x_{1:T}$, $y^*_{1:T}$ $W_{ah}$, $W_{ax}$, $b_a$, $W_{yh}$, $b_y$)
    Initialize the hidden state $h_0$ to zeros
    **for** $t$ in 1 to $T$ **do**
        Receive input data at time step $t$: $x_t$
        Compute the hidden state update:
            $a_t = W_{ah} \cdot h_{t-1}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


58

# RNN-LM + Loss

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

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



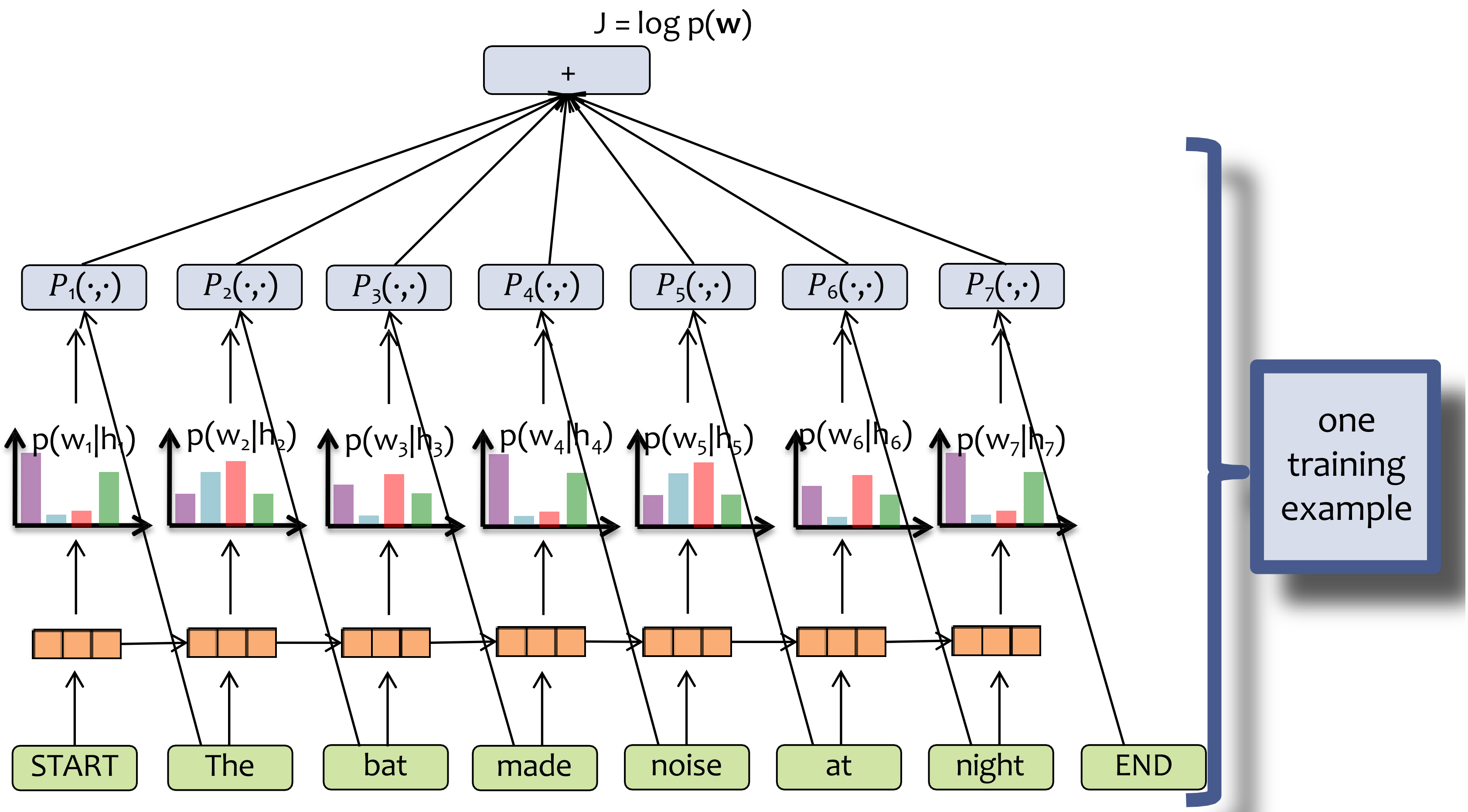

**Algorithm 1** Elman RNN + Loss

**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

- 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)

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





60

# LARGE LANGUAGE MODELS


61

# How large are LLMs?

上海科技大学 ShanghaiTech University

Comparison of some recent **large language models** (LLMs)

| Model | Creators | Year of release | Training Data (# tokens) | Model Size (# parameters) |
| --- | --- | --- | --- | --- |
| GPT-2 | OpenAI | 2019 | ~10 billion (40Gb) | 1.5 billion |
| GPT-3 | OpenAI | 2020 | 300 billion | 175 billion |
| PaLM | Google | 2022 | 780 billion | 540 billion |
| Chinchilla | DeepMind | 2022 | 1.4 trillion | 70 billion |
| LaMDA (cf. Bard) | Google | 2022 | 1.56 trillion | 137 billion |
| LLaMA | Meta | 2023 | 1.4 trillion | 65 billion |
| LLaMA-2 | Meta | 2023 | 2 trillion | 70 billion |
| GPT-4 | OpenAI | 2023 | ? | ? (1.76 trillion) |
| Gemini (Ultra) | Google | 2023 | ? | ? (1.5 trillion) |
| LLaMA-3 | Meta | 2024 | 15 trillion | 405 billion |


62

# What is ChatGPT?



- ChatGPT is a large (in the sense of having many parameters) language model, fine-tuned to be a dialogue agent
- The base language model is GPT-3.5 which was trained on a large quantity of text

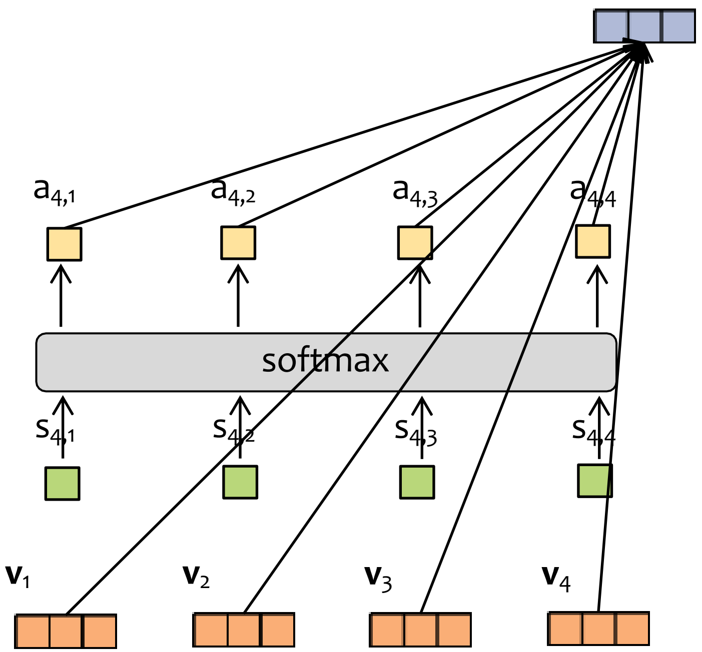

63

Transformer Language Models

# MODEL: GPT


64

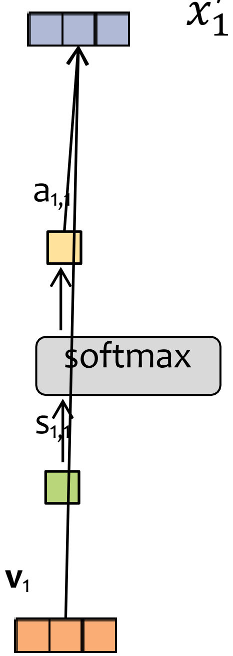
# Attention




$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

# Attention

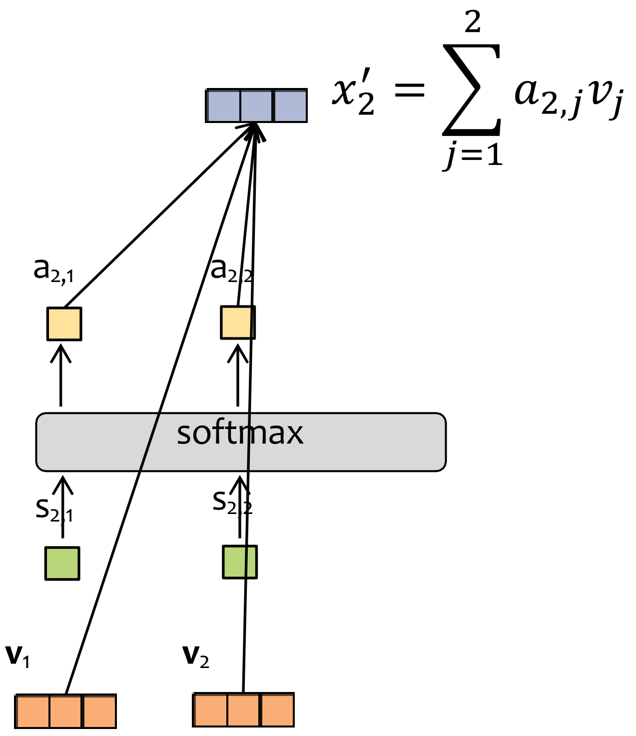





$$x_1' = \sum_{j=1}^{1} a_{1,j} v_j$$

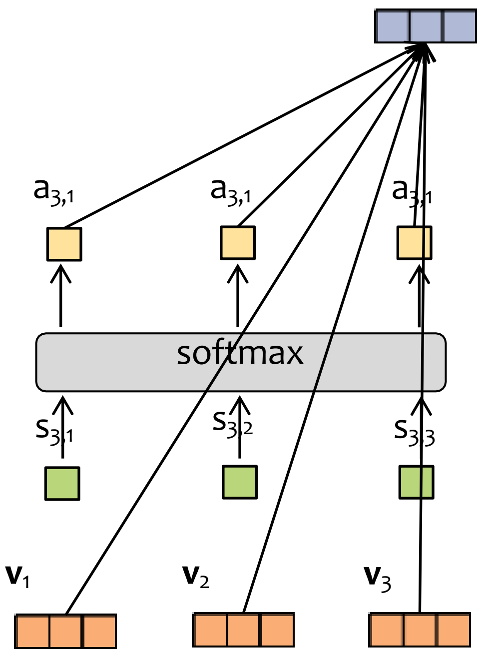
# Attention

上海科技大学
ShanghaiTech University



# Attention

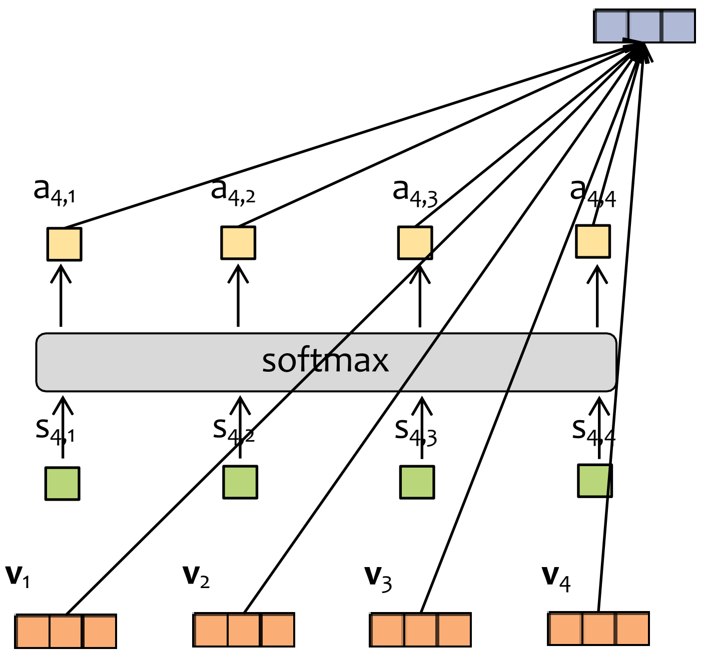





$$x'_3 = \sum_{j=1}^{3} a_{3,j} v_j$$

# Attention

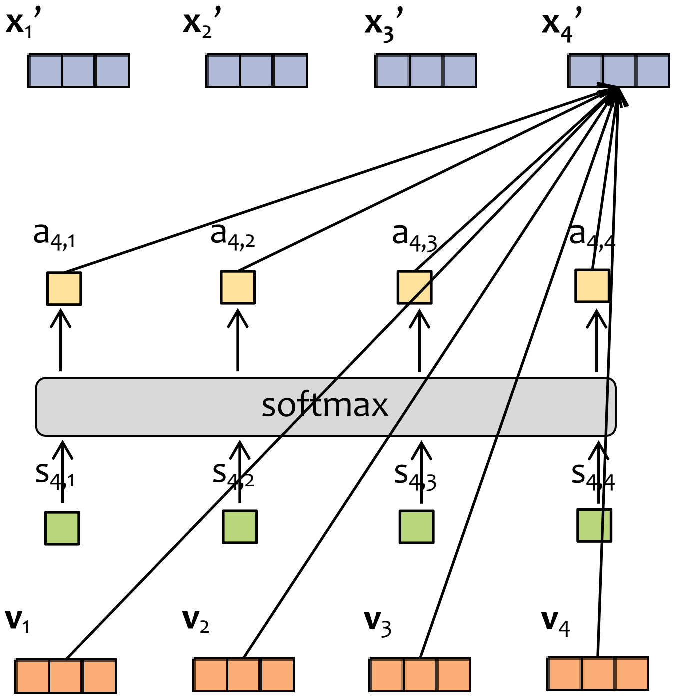





$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

# Attention






$$x'_t = \sum_{j=1}^{t} a_{t,j} v_j$$

attention weights

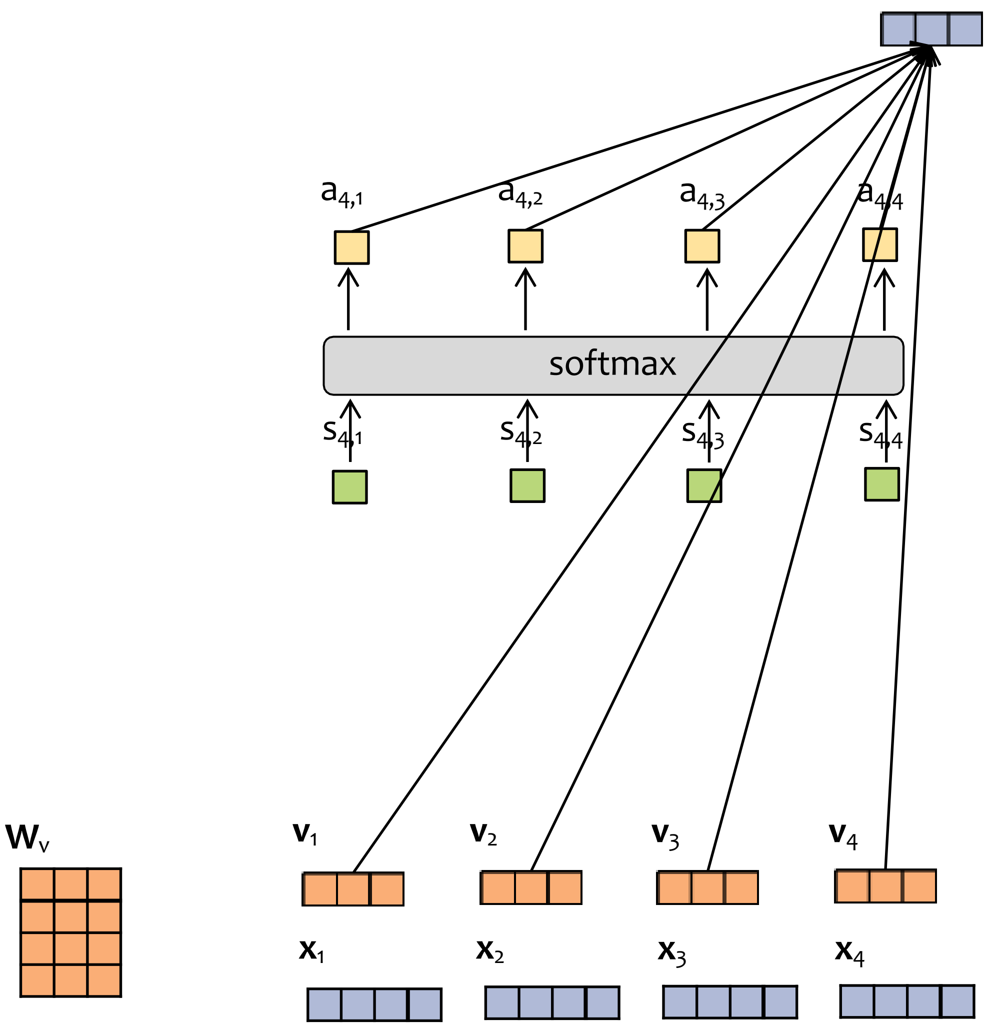
scores

values

# Scaled Dot-Product Attention



$$x_4' = \sum_{j=1}^{4} a_{4,j} v_j$$



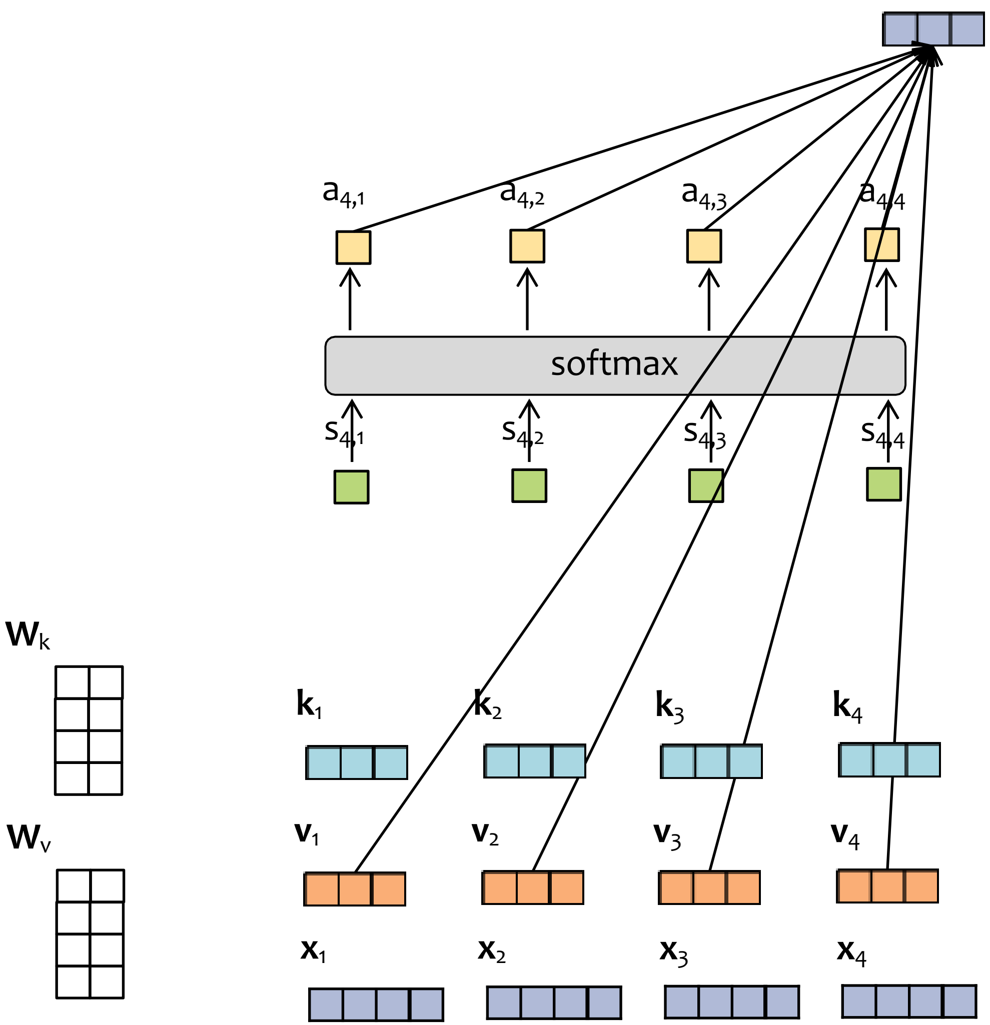

$$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$$

values

# Scaled Dot-Product Attention

上海科技大学
ShanghaiTech University

$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

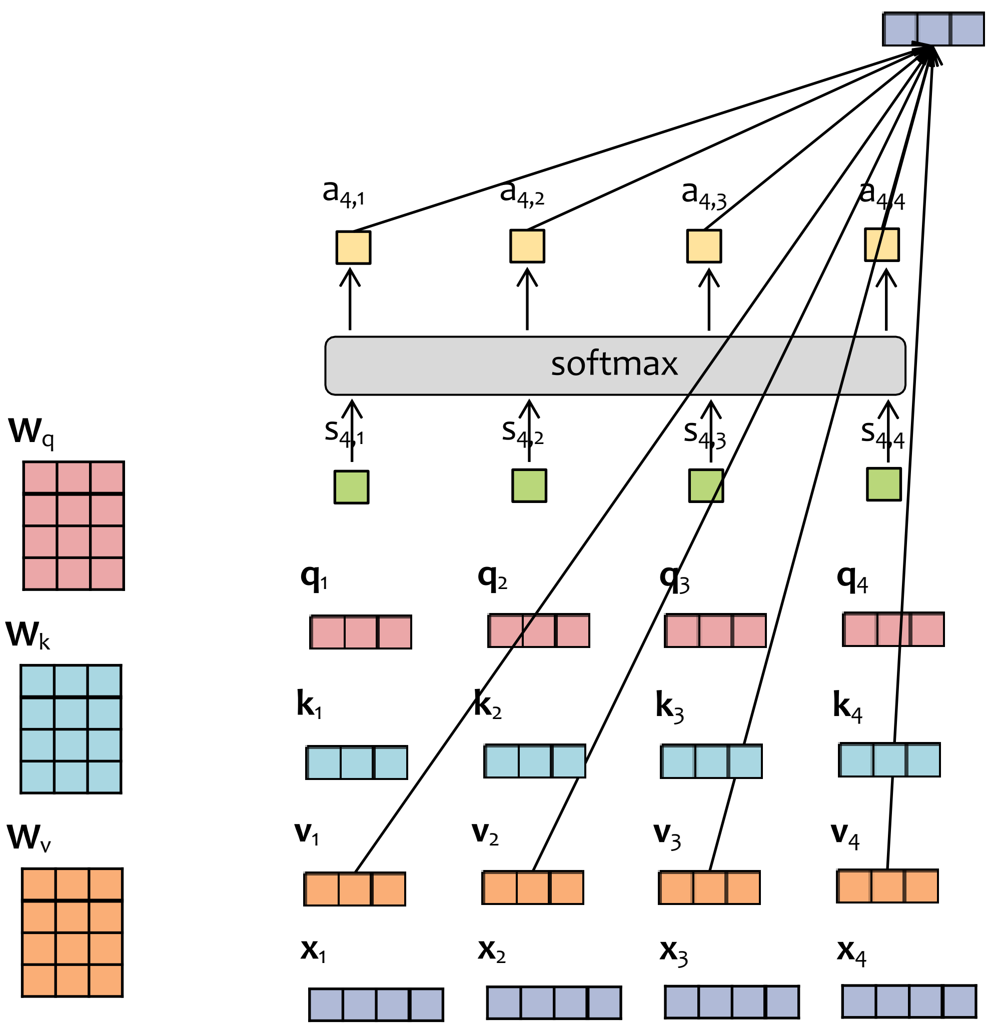



$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Scaled Dot-Product Attention






$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

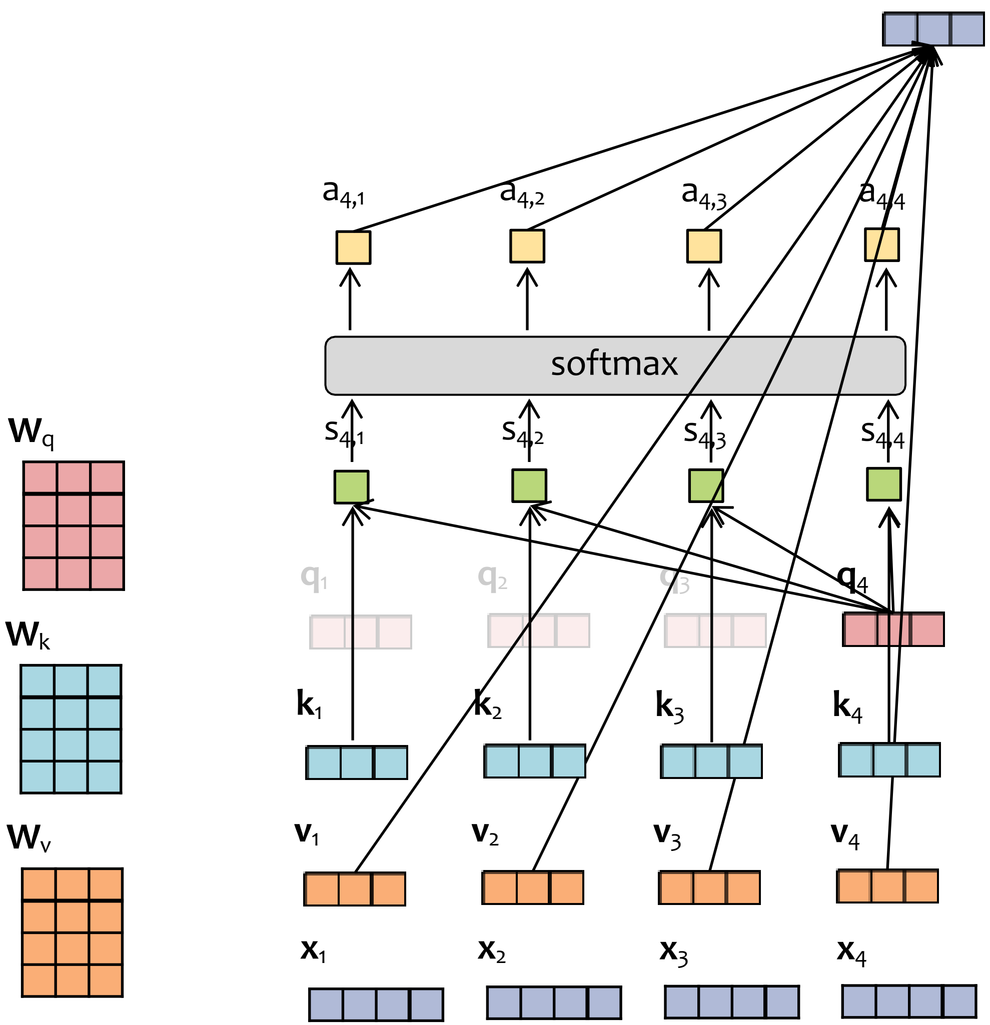
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Scaled Dot-Product Attention

上海科技大学
ShanghaiTech University




$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

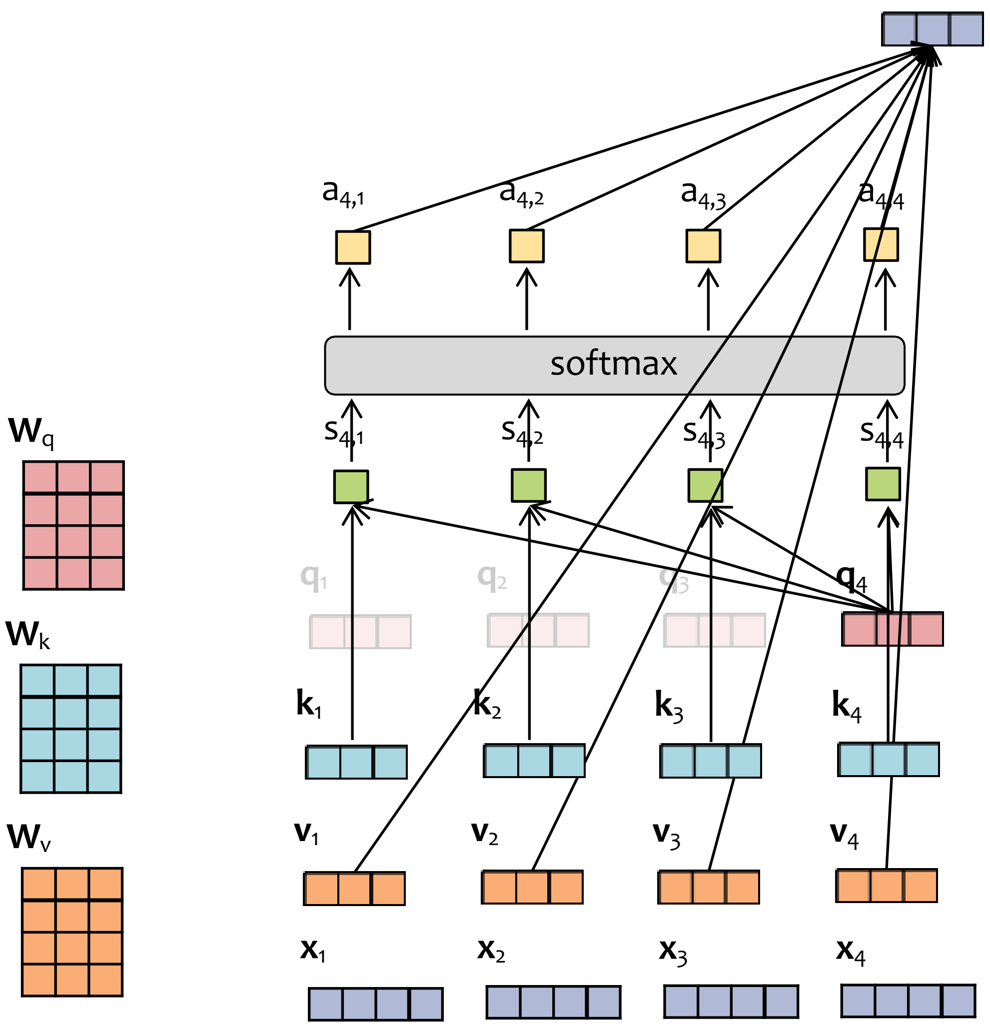
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Scaled Dot-Product Attention






$$x_4' = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{a}_4 = \mathrm{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

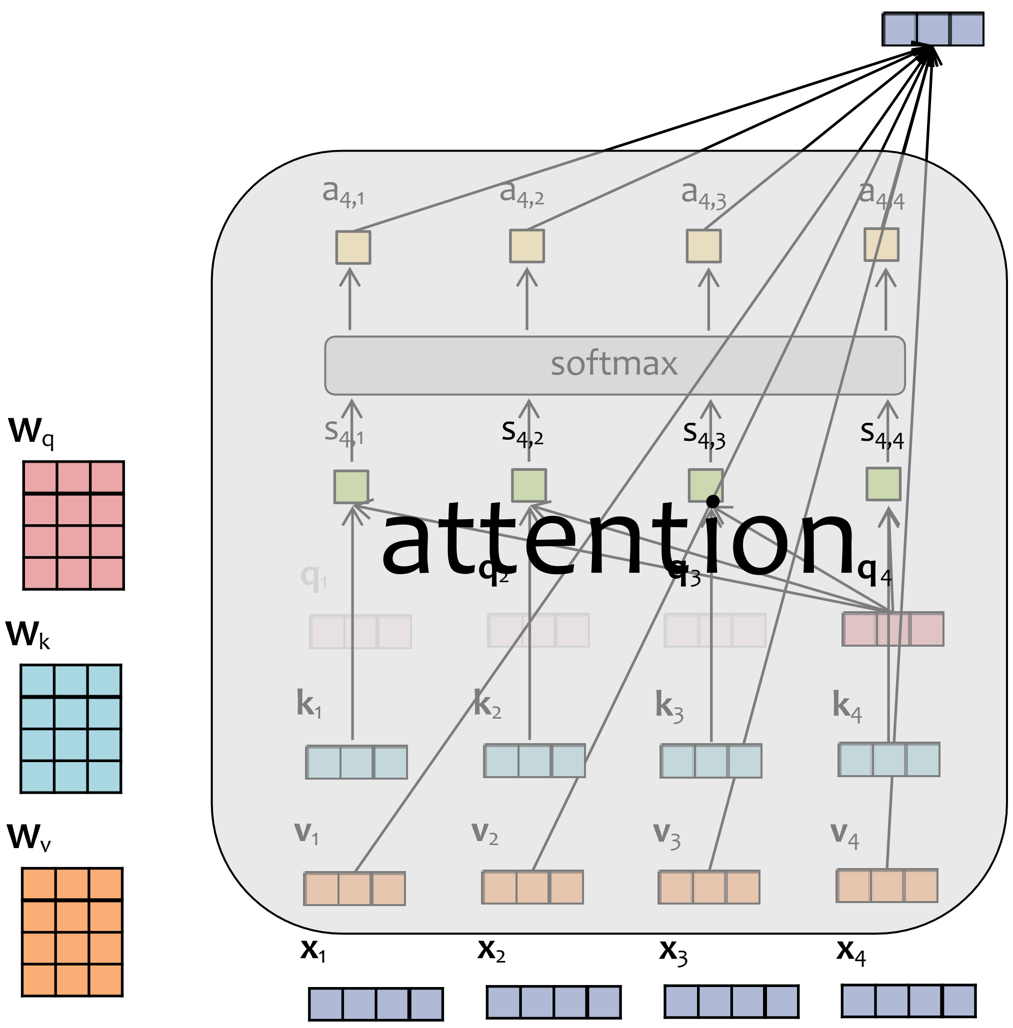
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Scaled Dot-Product Attention






$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

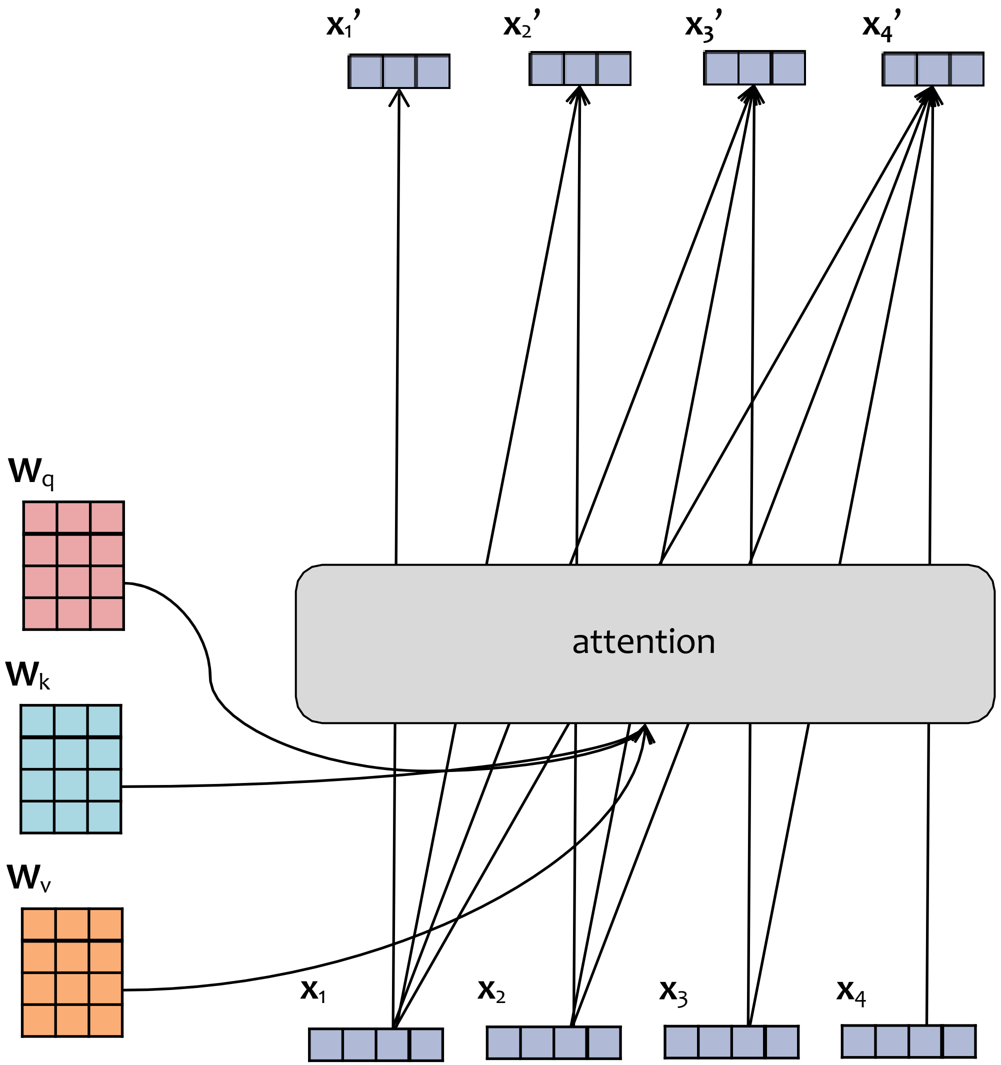
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Scaled Dot-Product Attention

上海科技大学 ShanghaiTech University




$$x'_t = \sum_{j=1}^{t} a_{t,j} v_j$$

$\mathbf{a}_t$ = softmax($\mathbf{s}_t$) attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Animation of 3D Convolution

Recall...

http://cs231n.github.io/convolutional-networks/

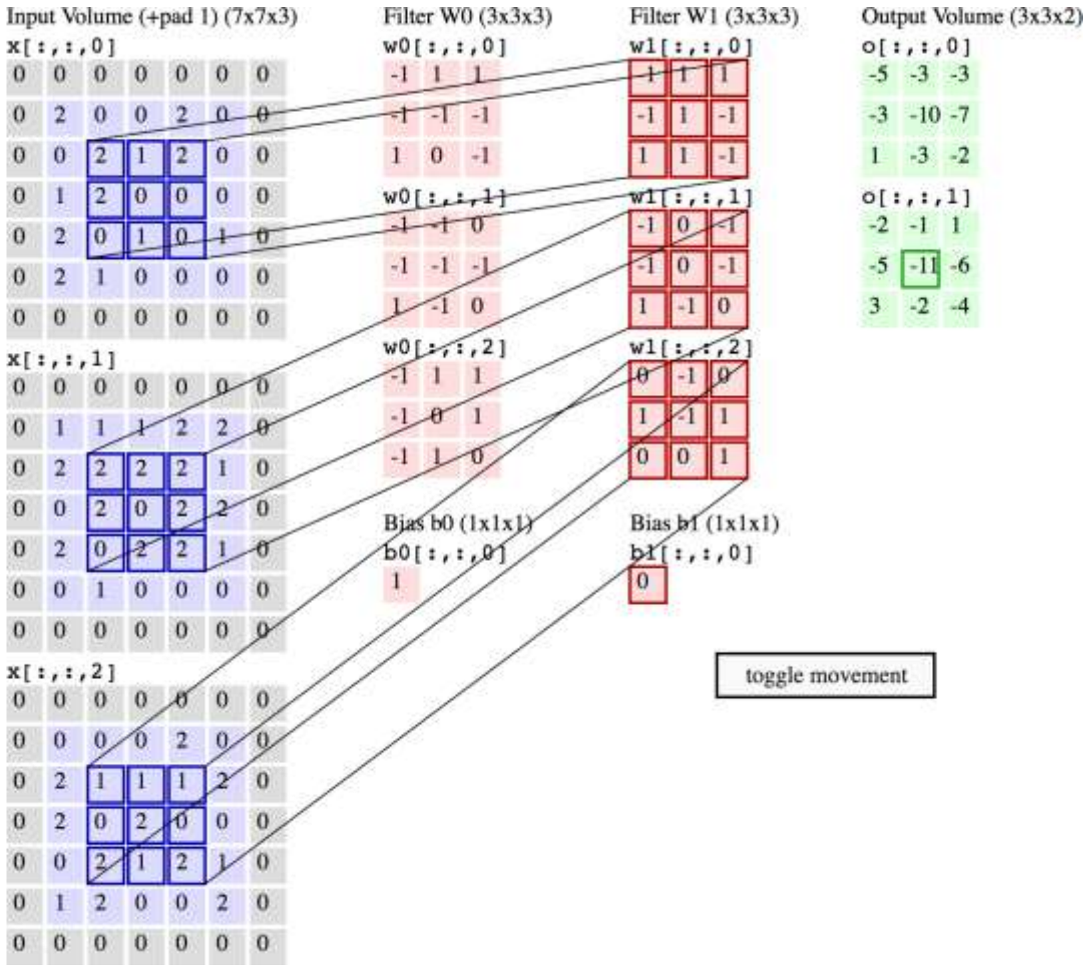
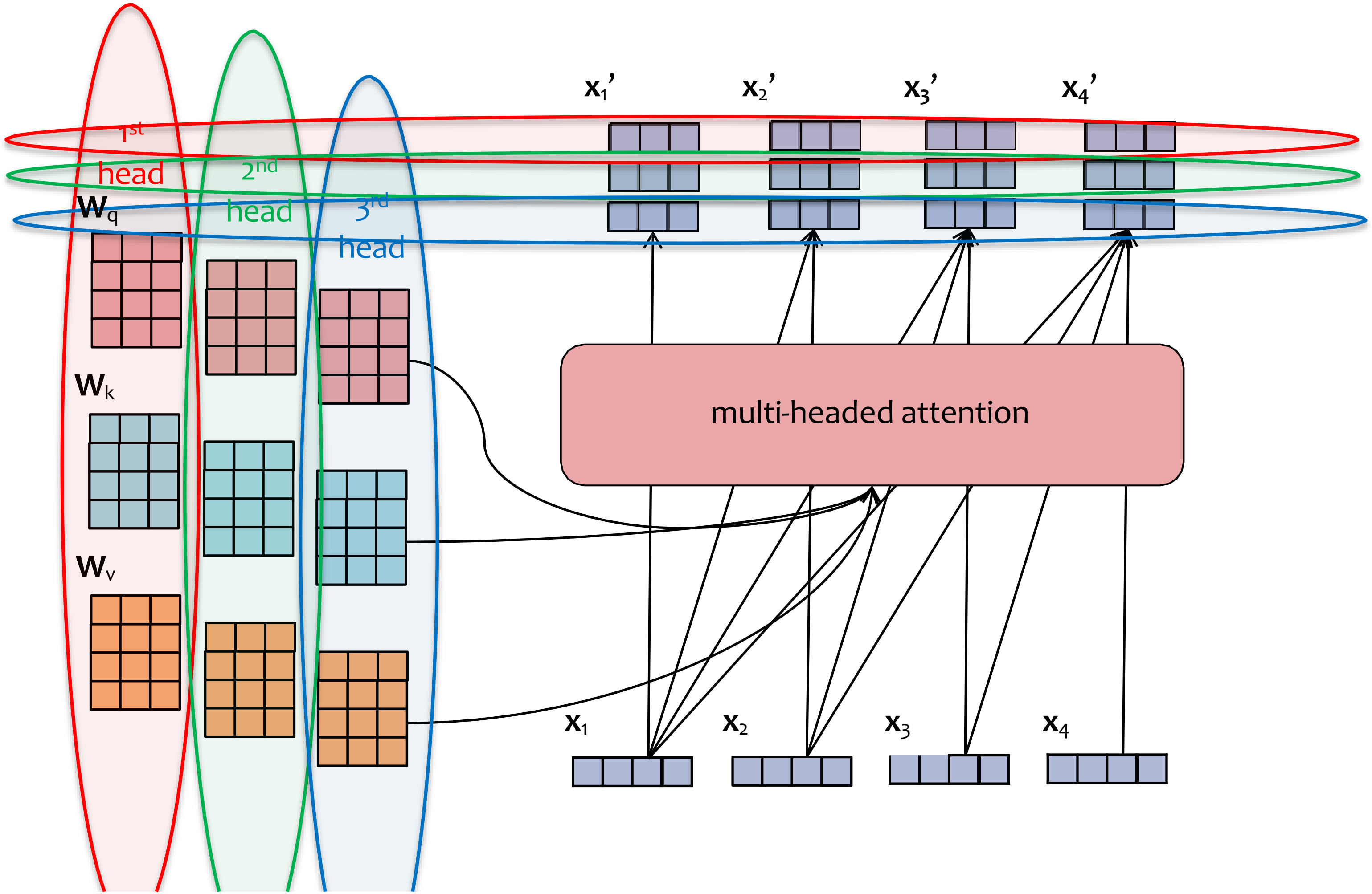



Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

# Multi-headed Attention

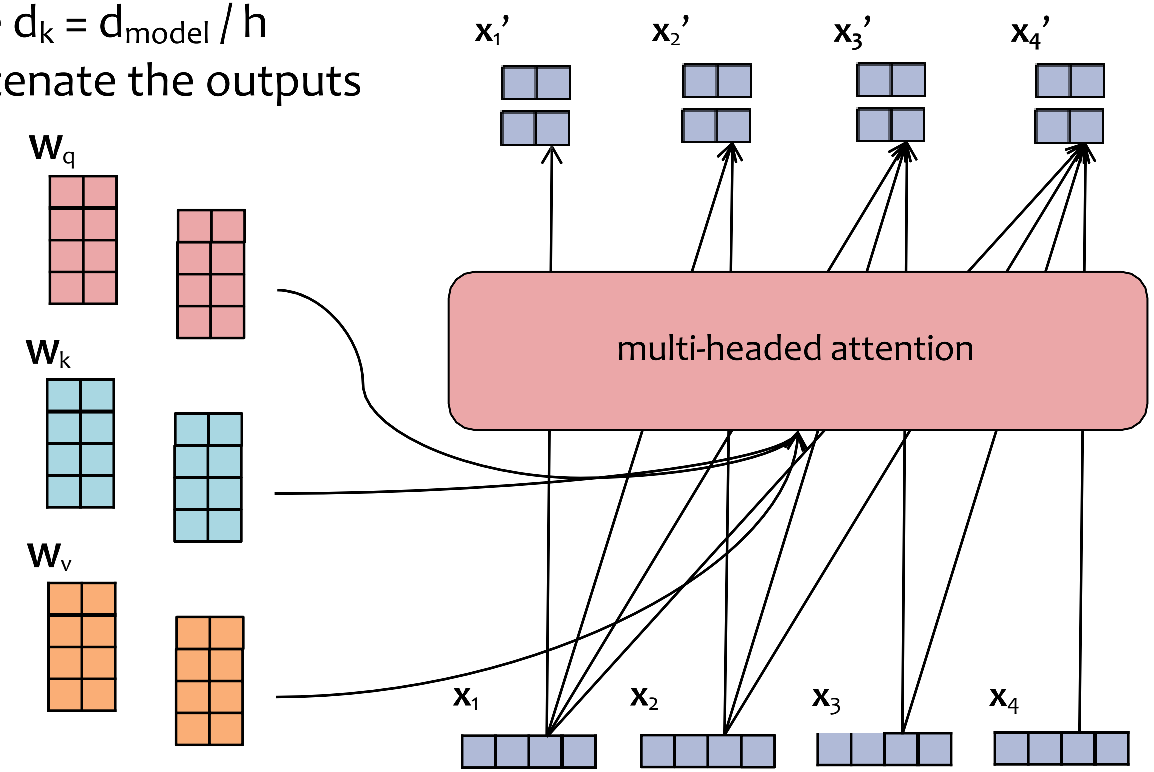





- Just as we can have **multiple channels** in a **convolution** layer, we can use **multiple heads** in an **attention** layer
- Each head gets **its own parameters**
- We can **concatenate** all the outputs to get a single vector for each time step

# Multi-headed Attention

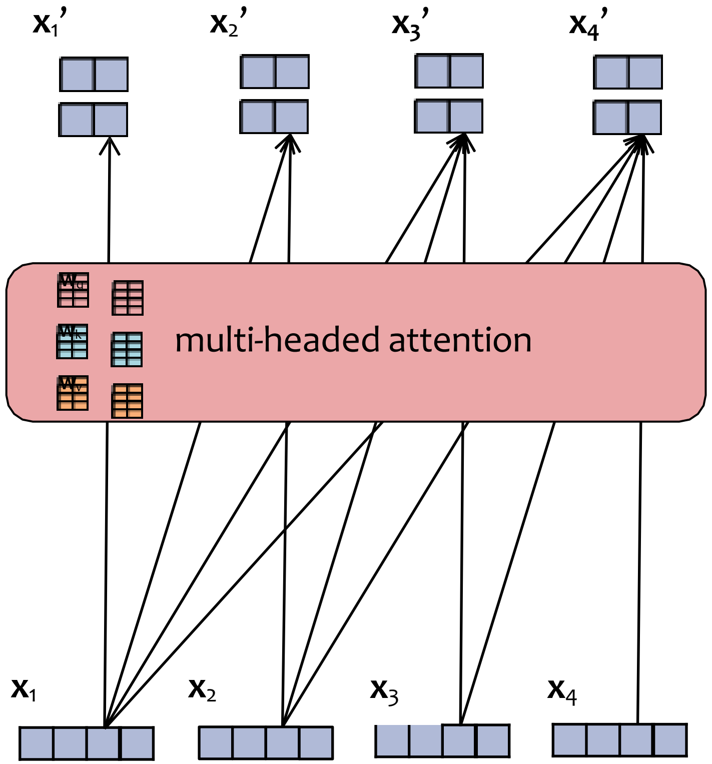
- To ensure the dimension of the **input** embedding $\mathbf{x}_t$ is the same as the **output** embedding $\mathbf{x}_t'$, Transformers usually choose the embedding sizes and number of heads appropriately:
  - $d_{model}$ = dim. of inputs
  - $d_k$ = dim. of each output
  - h = # of heads
  - Choose $d_k = d_{model} / h$
- Then concatenate the outputs




- Just as we can have **multiple channels** in a **convolution** layer, we can use **multiple heads** in an **attention** layer
- Each head gets **its own parameters**
- We can **concatenate** all the outputs to get a single vector for each time step

# Multi-headed Attention

- To ensure the dimension of the **input** embedding $\mathbf{x}_t$ is the same as the **output** embedding $\mathbf{x}_t'$, Transformers usually choose the embedding sizes and number of heads appropriately:
  - $d_{model}$ = dim. of inputs
  - $d_k$ = dim. of each output
  - h = # of heads
  - Choose $d_k = d_{model} / h$
- Then concatenate the outputs



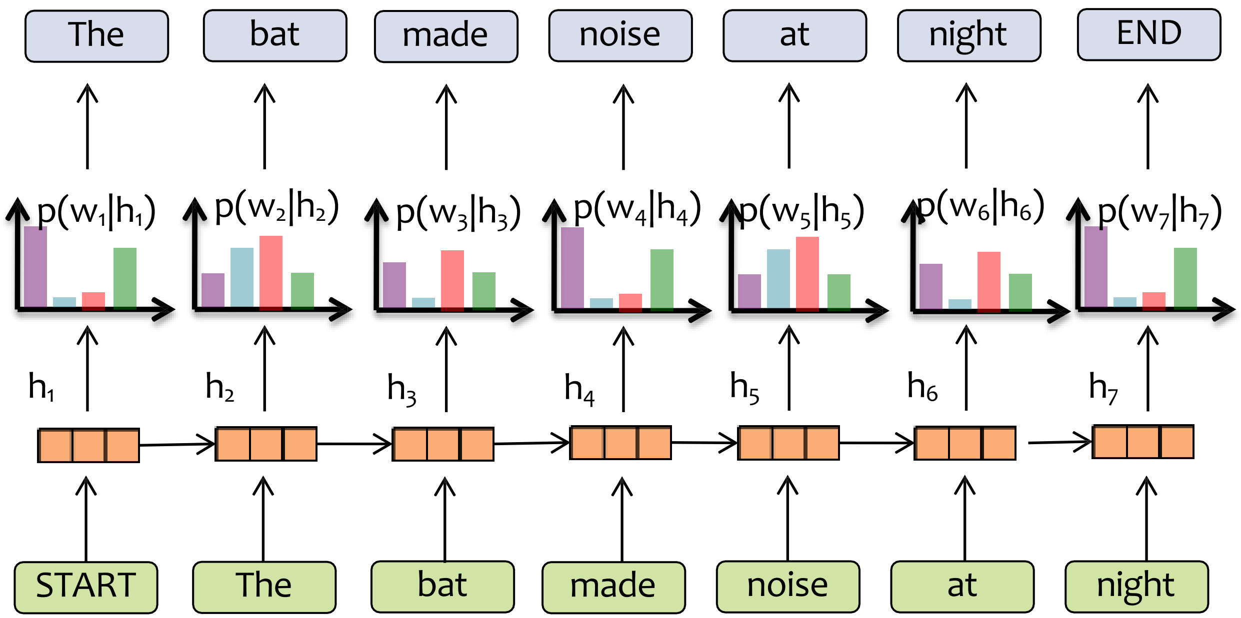

- Just as we can have **multiple channels** in a **convolution** layer, we can use **multiple heads** in an **attention** layer
- Each head gets **its own parameters**
- We can **concatenate** all the outputs to get a single vector for each time step


81

# RNN Language Model

上海科技大学 ShanghaiTech University

Recall...




*Key Idea*:

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

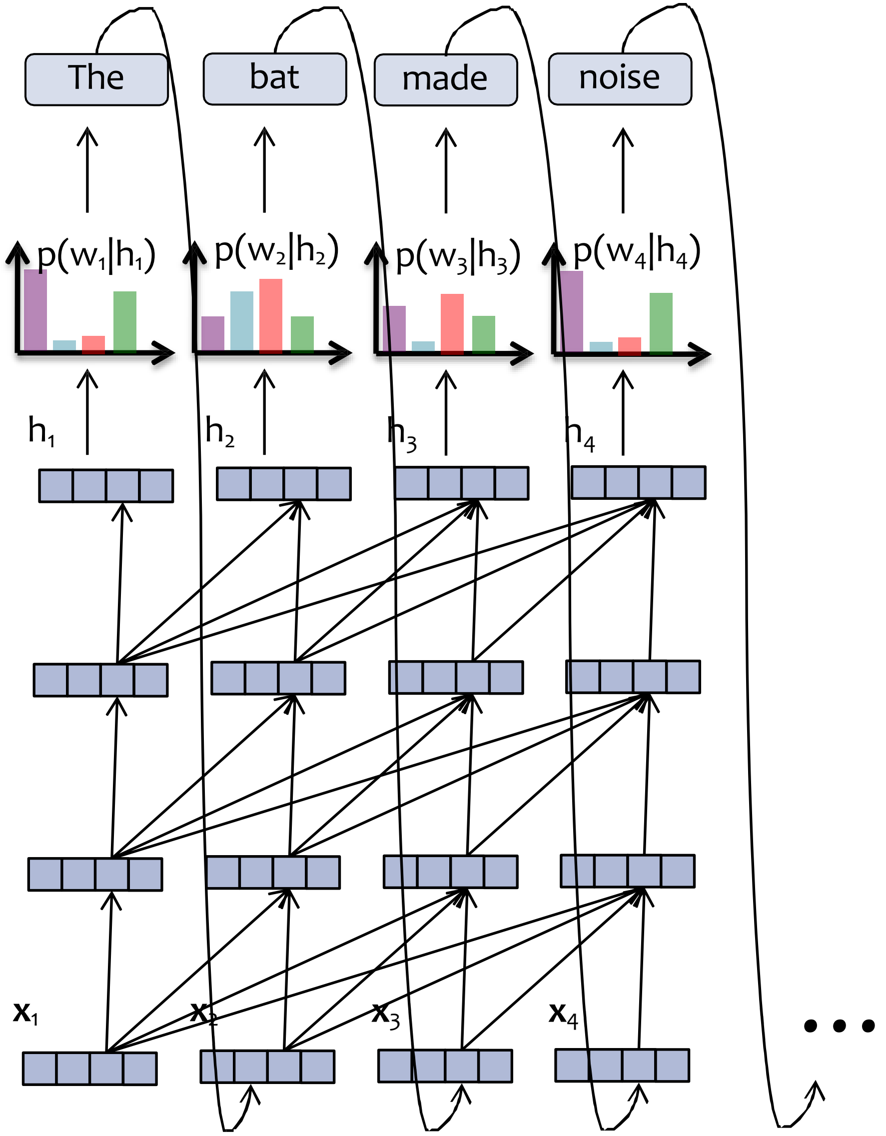
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

# Transformer Language Model

上海科技大学
ShanghaiTech University

Important!

- RNN computation graph grows **linearly** with the number of input tokens
- Transformer-LM computation graph grows **quadratically** with the number of input tokens




Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

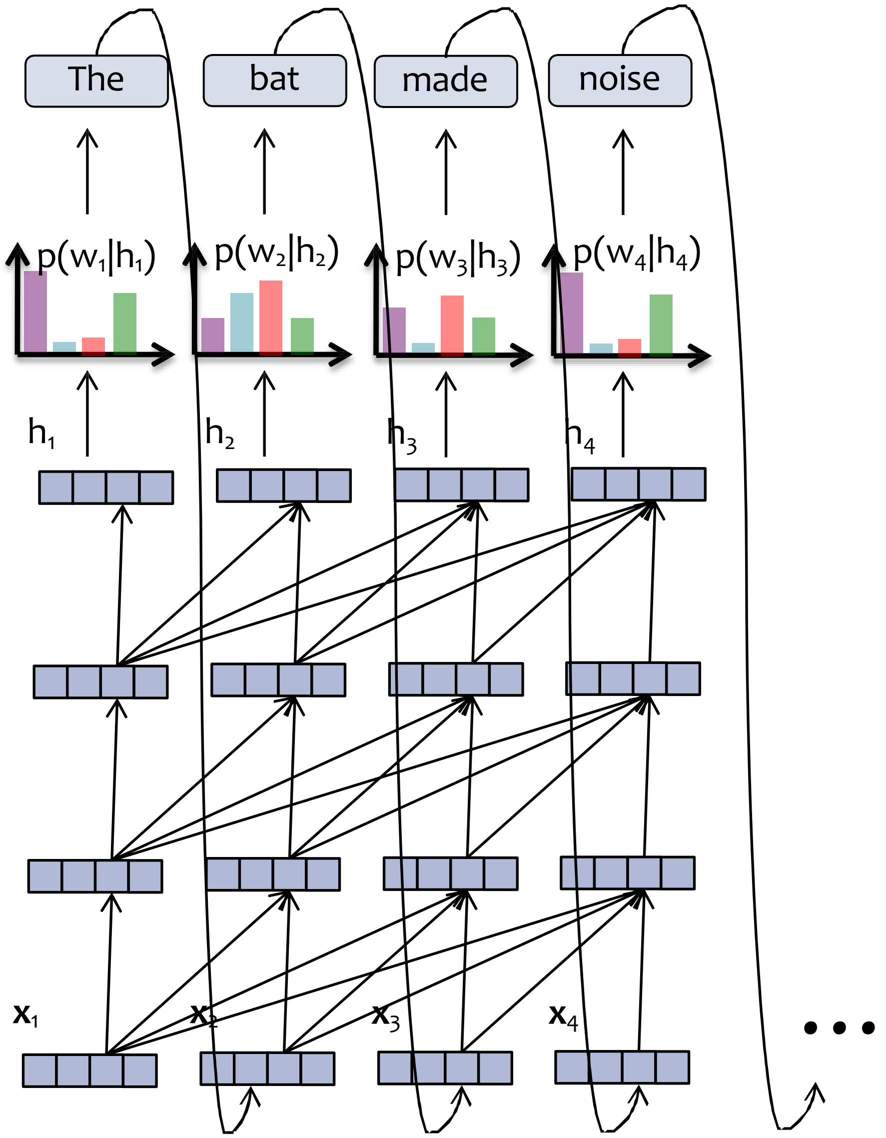
The language model part is just like an RNN-LM!

# Transformer Language Model

上海科技大学
ShanghaiTech University

Important!

- RNN computation graph grows **linearly** with the number of input tokens
- Transformer-LM computation graph grows **quadratically** with the number of input tokens




**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

The language model part is just like an RNN-LM!

# Layer Normalization

上海科技大学 ShanghaiTech University

- *The Problem*: **internal covariate shift** occurs during training of a deep network when a small change in the low layers amplifies into a large change in the high layers
- *One Solution:* **Layer normalization** normalizes each layer and learns elementwise gain/bias
- Such normalization allows for higher learning rates (for **faster convergence**) without issues of diverging gradients

Given input a $\in \mathbb{R}^K$, LayerNorm computes output b $\in \mathbb{R}^K$:

$$b = \gamma \odot \frac{a-u}{\sigma} \oplus \beta$$

where we have mean $\mu = \frac{1}{K}\Sigma_{k=1}^{K} a_k$ ,

standard deviation $\sigma = \sqrt{\frac{1}{K}\Sigma_{k=1}^{K}(a_k-\mu)^2}$ ,

and parameters γ $\in \mathbb{R}^K$, β $\in \mathbb{R}^K$.

$\odot$ and $\oplus$ denote elementwise multiplication and addition.

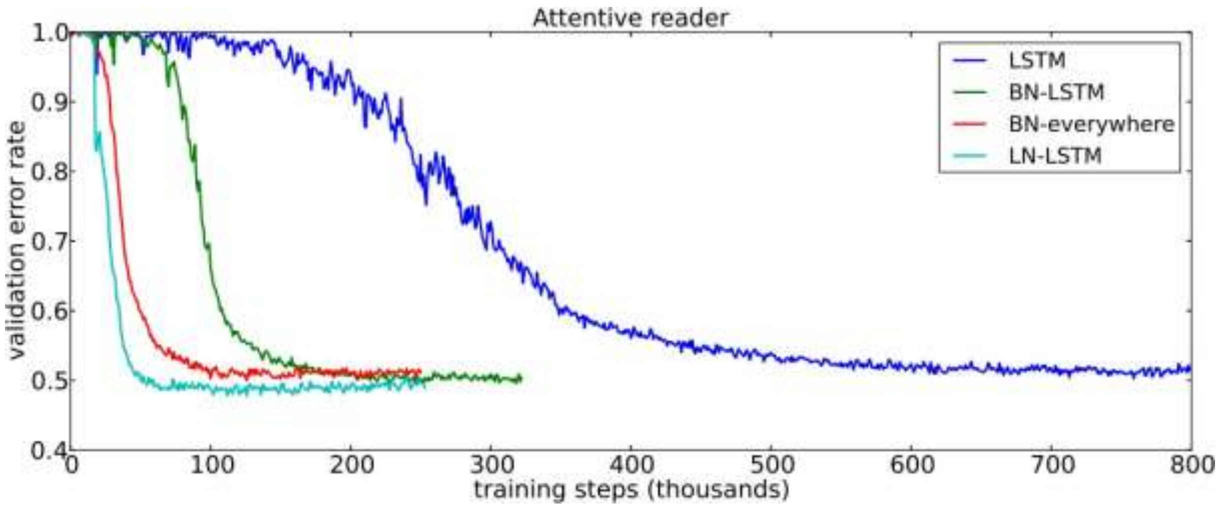
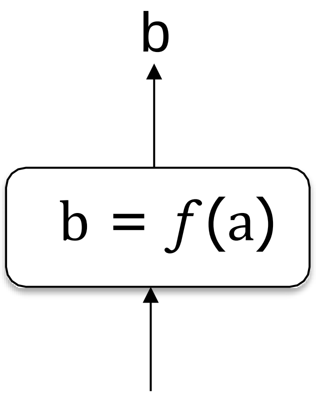



Figure from https://arxiv.org/pdf/1607.06450.pdf

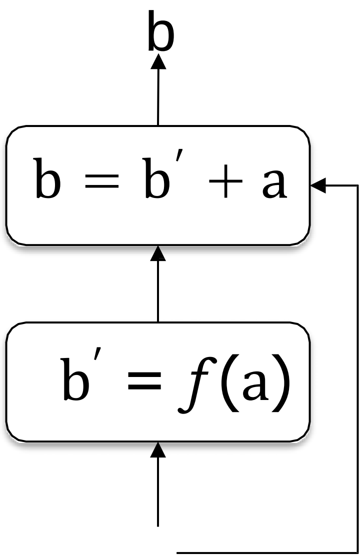
# Residual Connections

- *The Problem:* as network depth grows very large, a **performance degradation** occurs that is not explained by overfitting (i.e. train / test error both worsen)
- *One Solution:* **Residual connections** pass a copy of the input alongside another function so that information can flow more directly
- These residual connections allow for **effective training of very deep networks** that perform better than their shallower (though still deep) counterparts

Plain Connection




Residual Connection



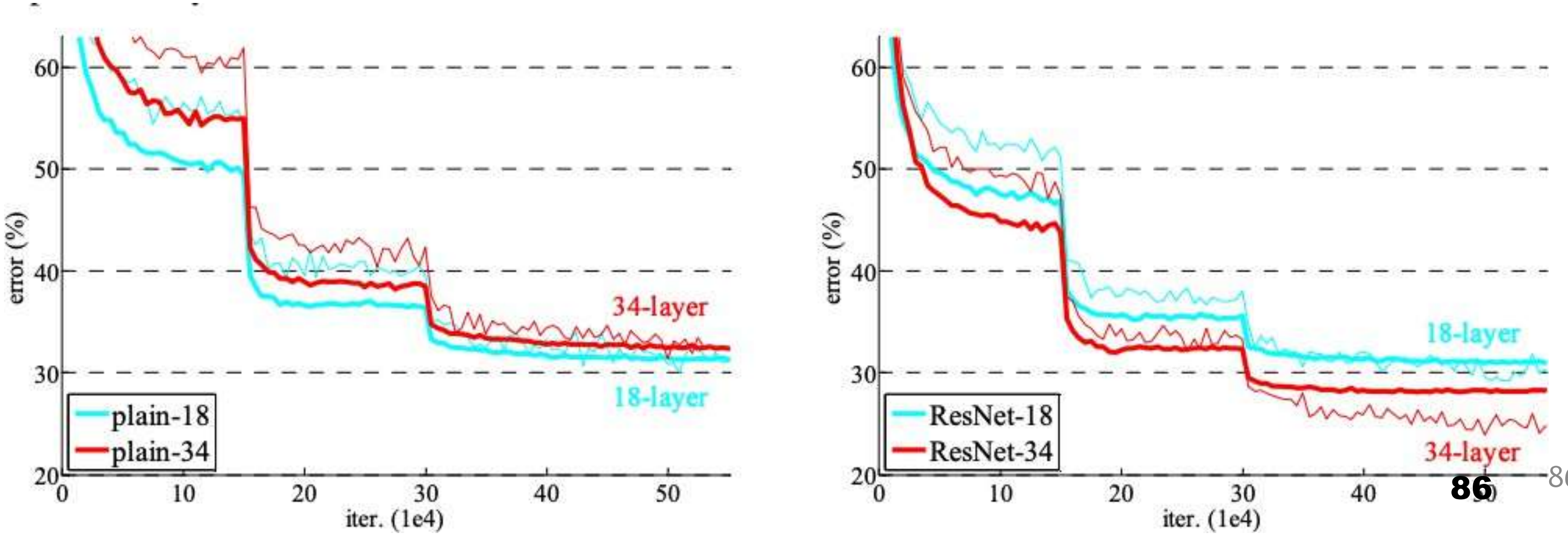
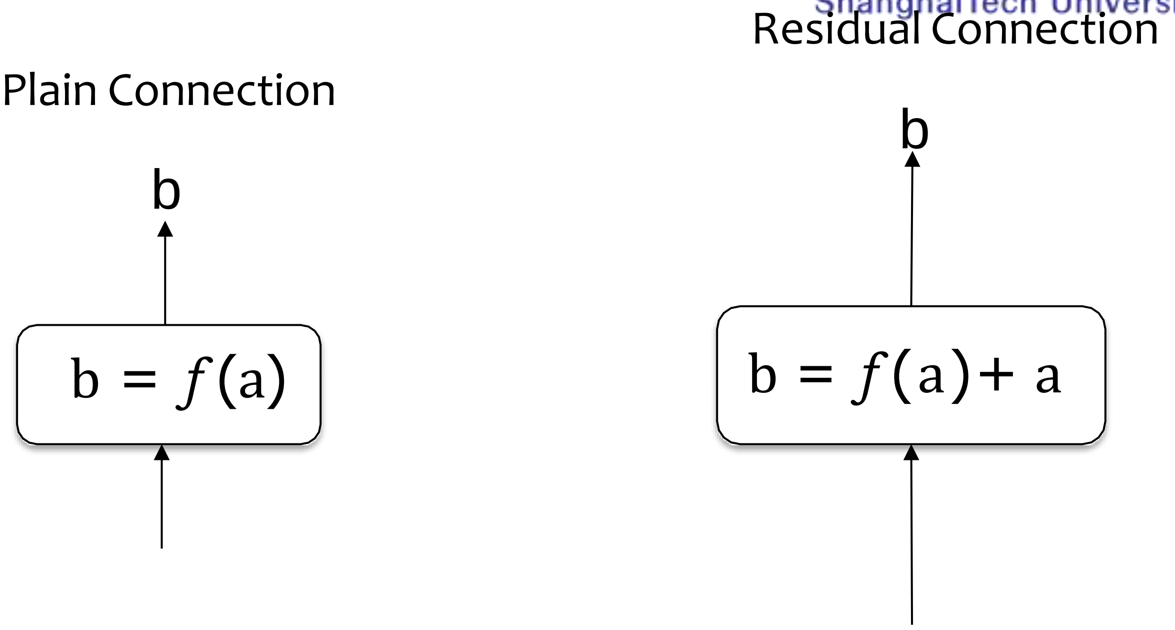




Figure from https://arxiv.org/pdf/1512.03385.pdf

# Residual Connections

上海科技大学
ShanghaiTech University

- *The Problem:* as network depth grows very large, a **performance degradation** occurs that is not explained by overfitting (i.e. train / test error both worsen)
- *One Solution:* **Residual connections** pass a copy of the input alongside another function so that information can flow more directly
- These residual connections allow for **effective training of very deep networks** that perform better than their shallower (though still deep) counterparts




**Why are residual connections helpful?**

Instead of f(a) having to learn a full transformation of a, f(a) only needs to learn an additive modification of a (i.e. the residual).

Figure from https://arxiv.org/pdf/1512.03385.pdf

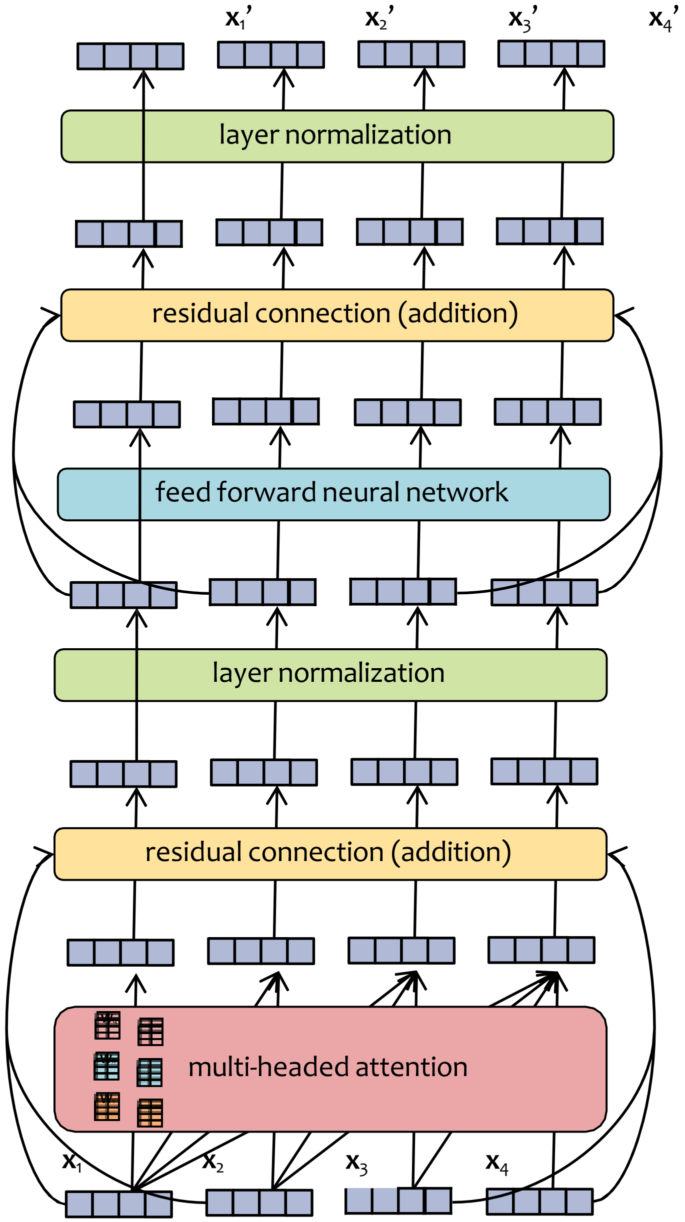
# Transformer Layer

上海科技大学 ShanghaiTech University

**Post-LN Version:**

This is the version of the Transformer Layer that was introduced in the original paper in 2017.

The LayerNorm modules occur at the end of each set of 3 layers.




**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

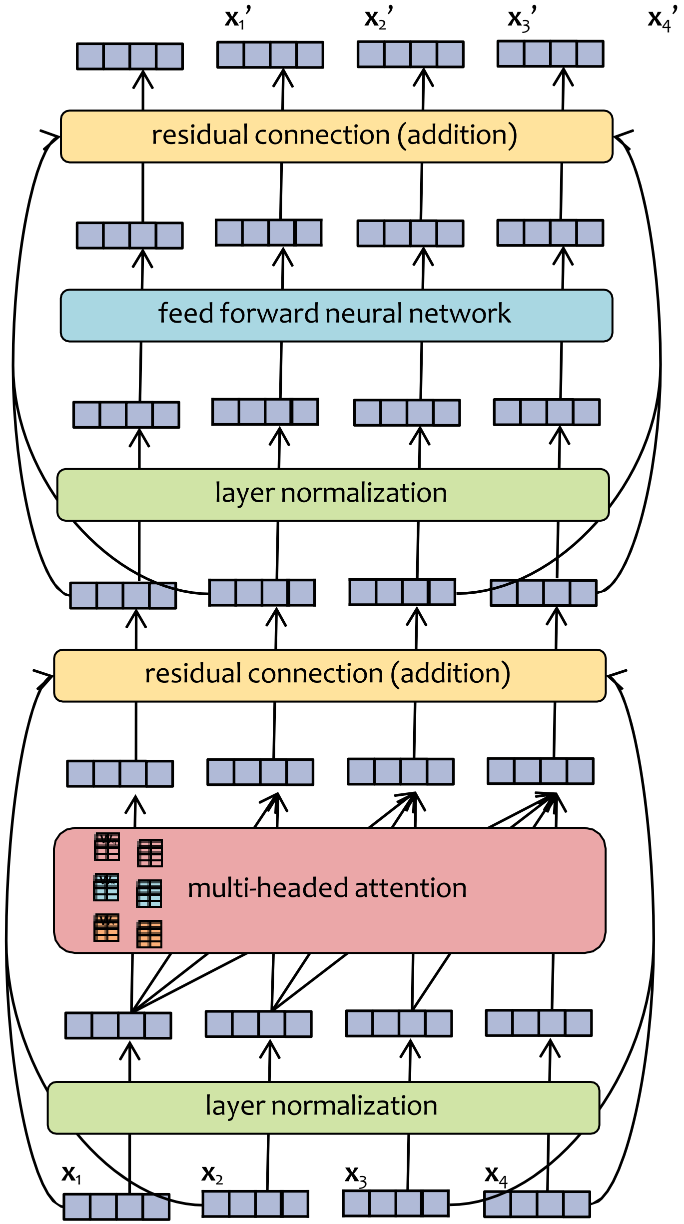

88

# Transformer Layer

上海科技大学
ShanghaiTech University

**Pre-LN Version:**

However, subsequent work found that reordering such that the LayerNorm's came at the beginning of each set of 3 layers, the multi-headed attention and feed-forward NN layers tend to be better behaved (i.e. tricks like warm-up are less important).

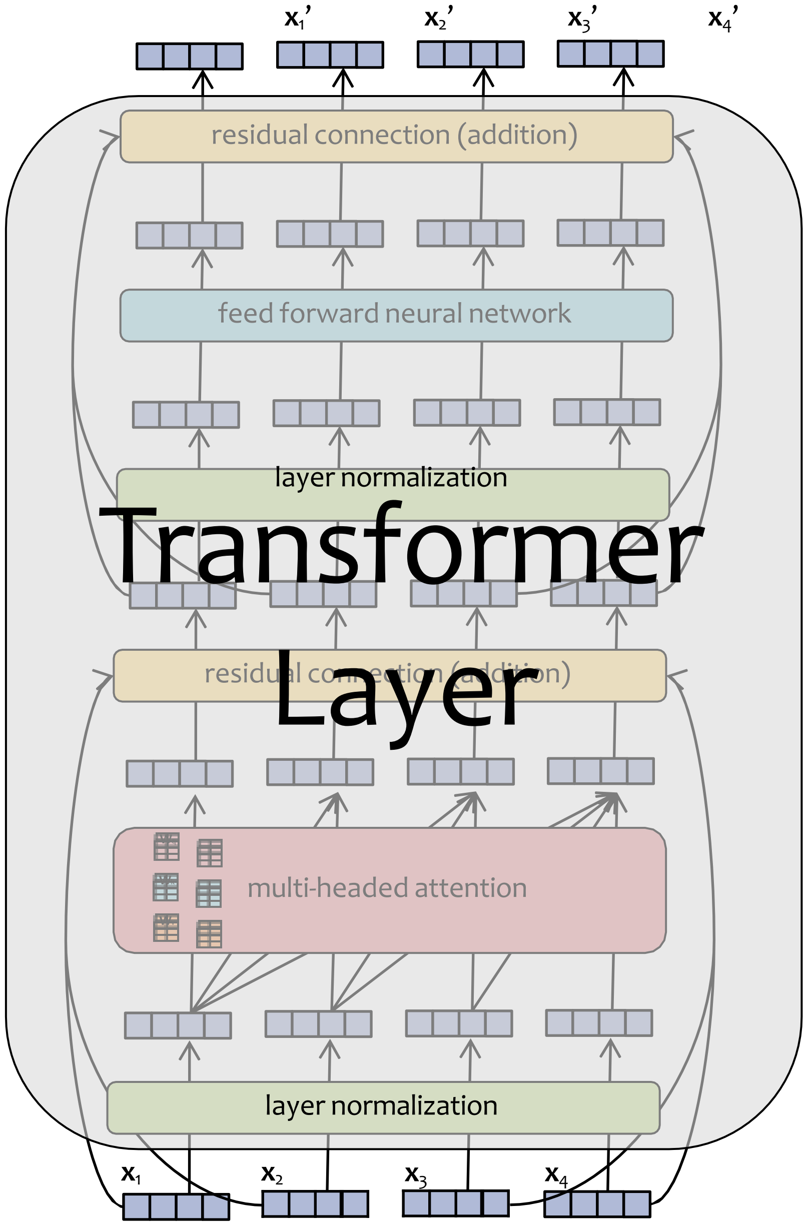



**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections


89

# Transformer Layer




Transformer Layer

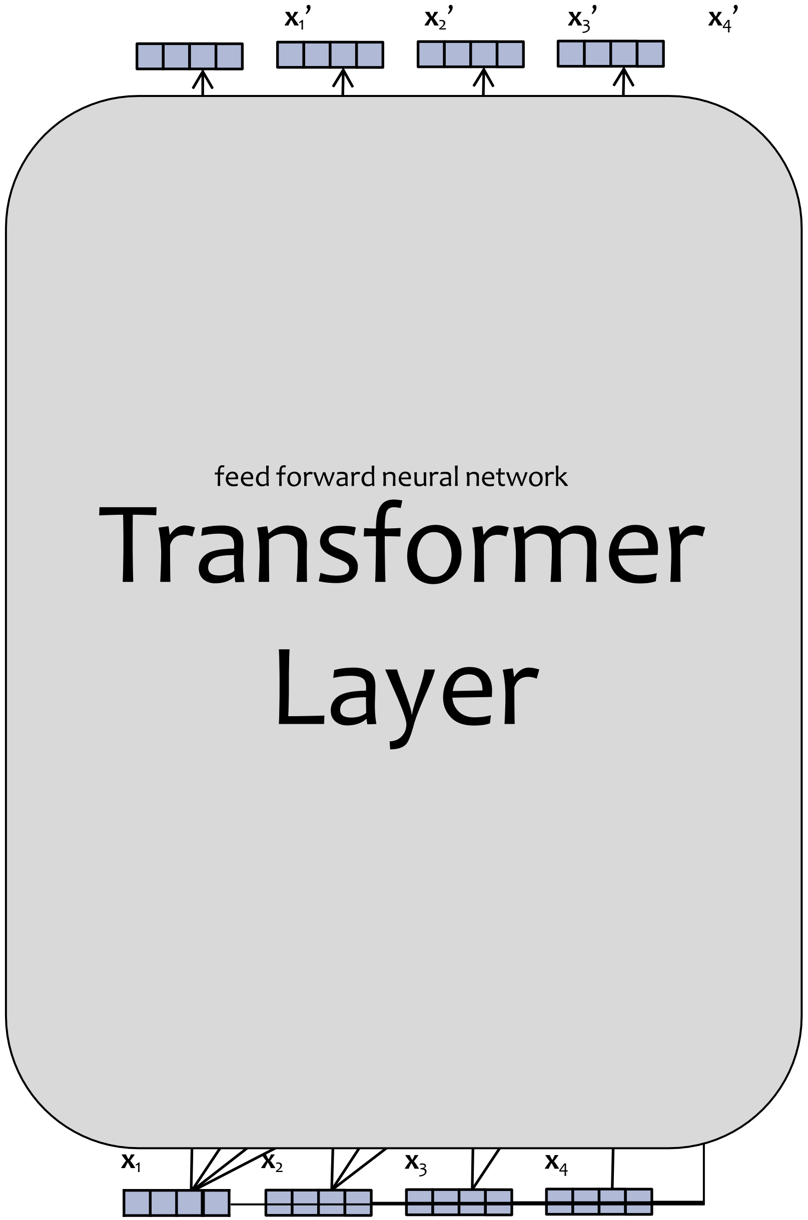
**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections


90

# Transformer Layer






**Each layer** of a Transformer LM consists of several **sublayers:**

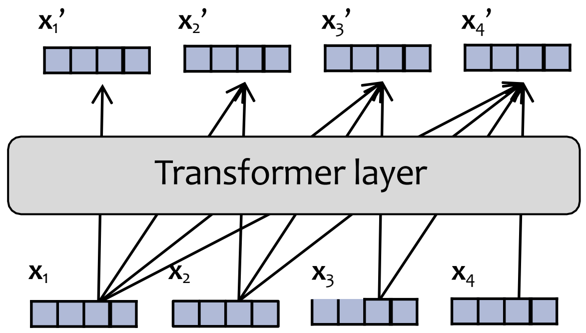
1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

# Transformer Layer



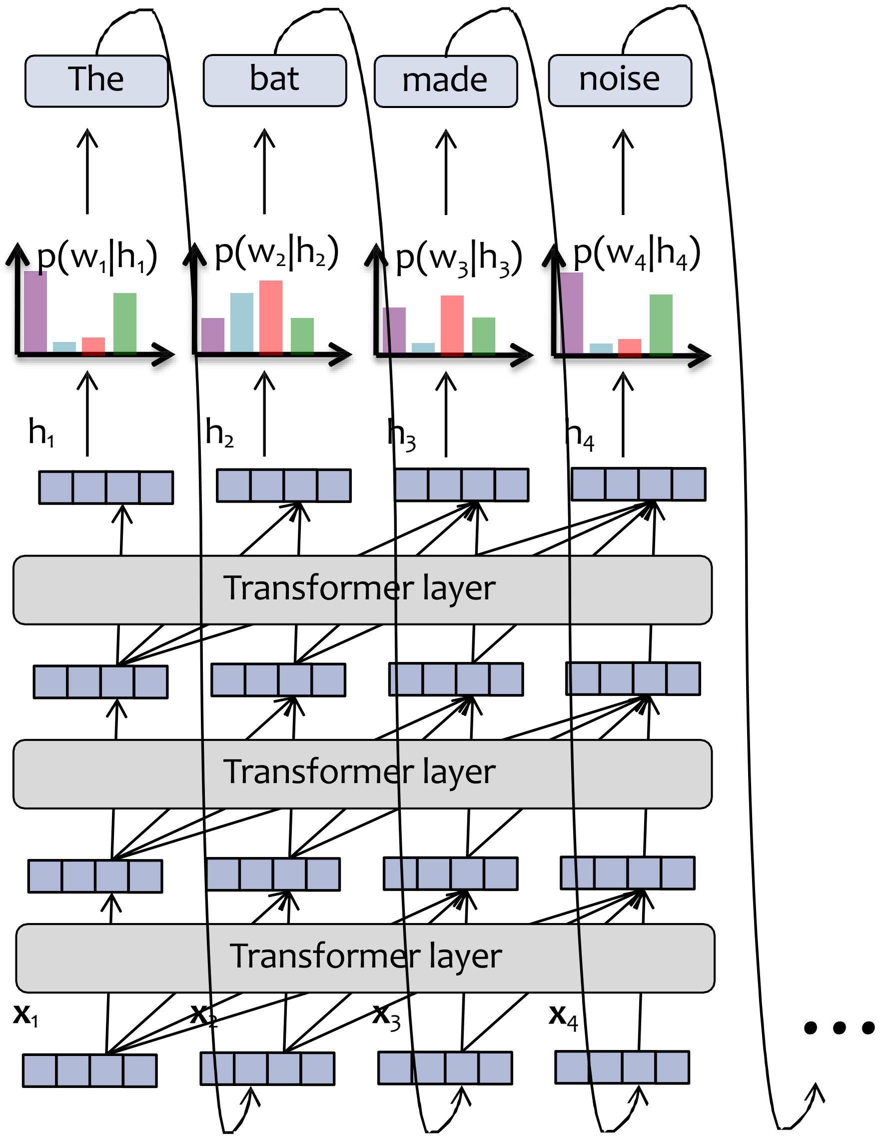
**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections



# Transformer Language Model

上海科技大学
ShanghaiTech University




**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

The language model part is just like an RNN-LM.

# In-Class Poll

**Question:**

Suppose we have the following input embeddings and attention weights:

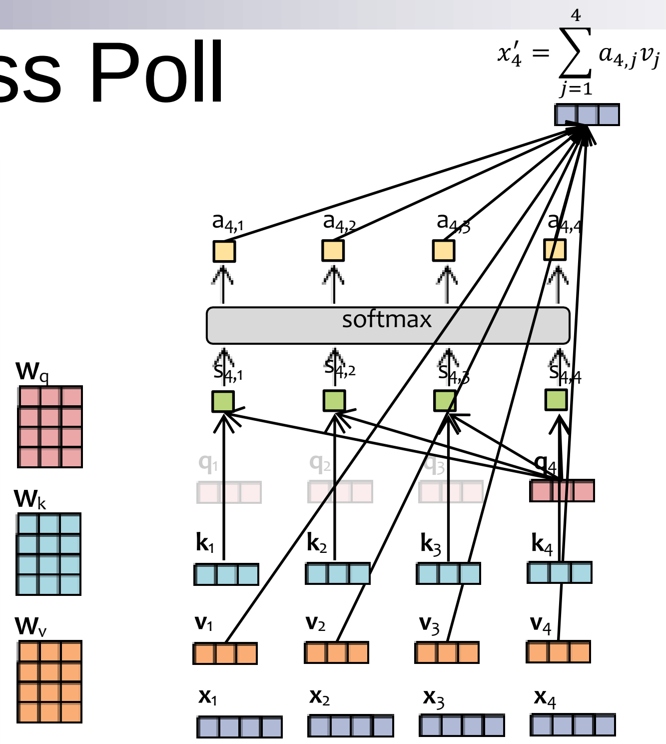
- $x_1 = [1,0,0,0]$ $a_{4,1} = 0.1$
- $x_2 = [0,1,0,0]$ $a_{4,2} = 0.2$
- $x_3 = [0,0,2,0]$ $a_{4,3} = 0.6$
- $x_4 = [0,0,0,1]$ $a_{4,4} = 0.1$

And $W_v = I$. Then we can compute $x_4'$.

Now suppose we swap the embeddings $x_2$ and $x_3$ such that

- $x_2 = [0,0,2,0]$
- $x_3 = [0,1,0,0]$

What is the new value of $x_4'$?




上海科技大学 ShanghaiTech University

$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

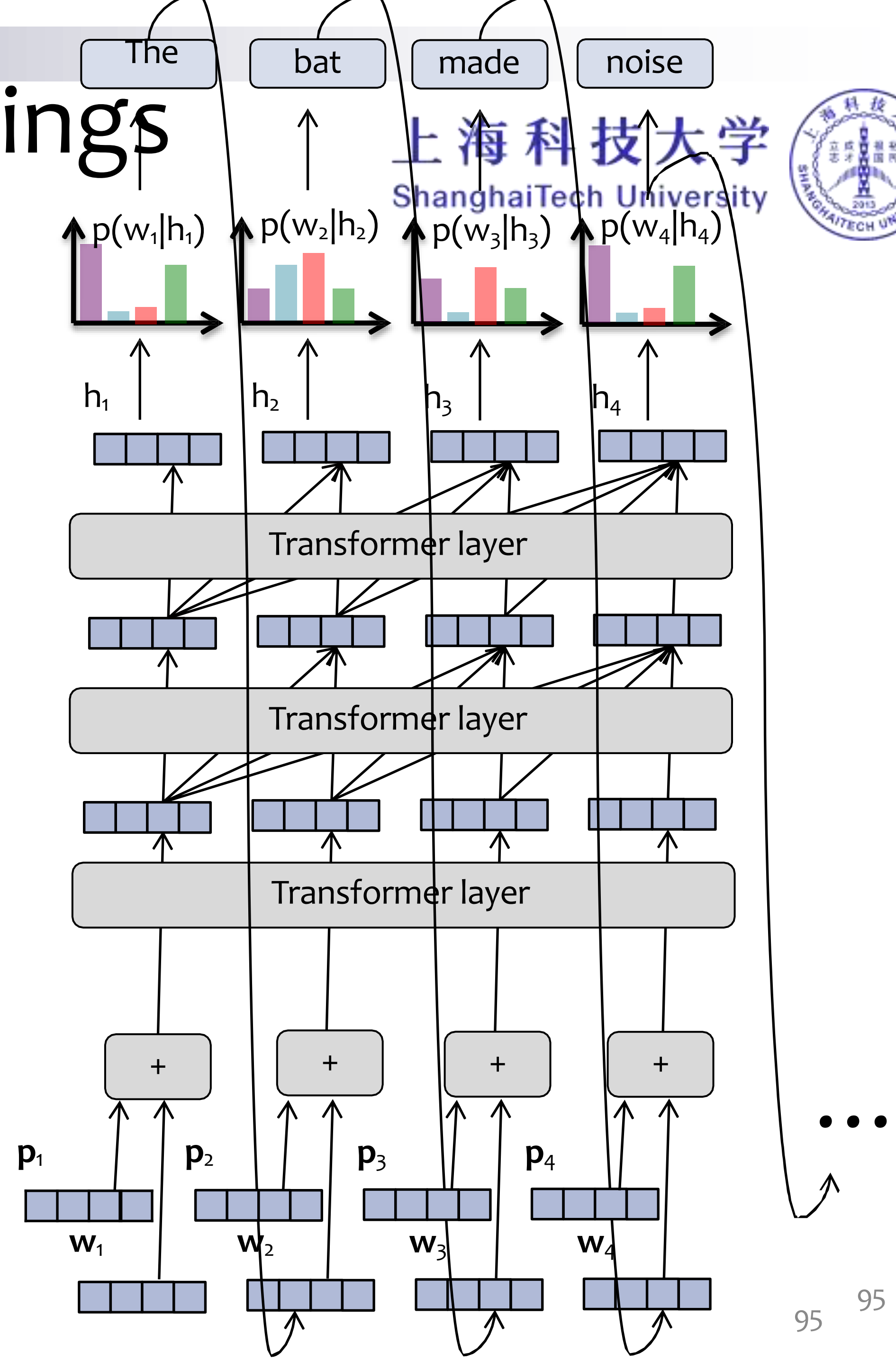
$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

**Answer:**

# Position Embeddings

- The Problem: Because attention is position invariant, we **need** a way to learn about positions
- The Solution: Use (or learn) a collection of position specific embeddings: $\mathbf{p}_t$ represents what it means to be in position t. And add this to the word embedding $\mathbf{w}_t$.
  The **key idea** is that every word that appears in position t uses the same position embedding $\mathbf{p}_t$
- There are a number of varieties of position embeddings:
  - Some are fixed (based on sine and cosine), whereas others are learned (like word embeddings)
  - Some are absolute (as described above) but we can also use relative position embeddings (i.e. relative to the position of the query vector)





95

上海科技大学
ShanghaiTech University

# LEARNING A TRANSFORMER LM

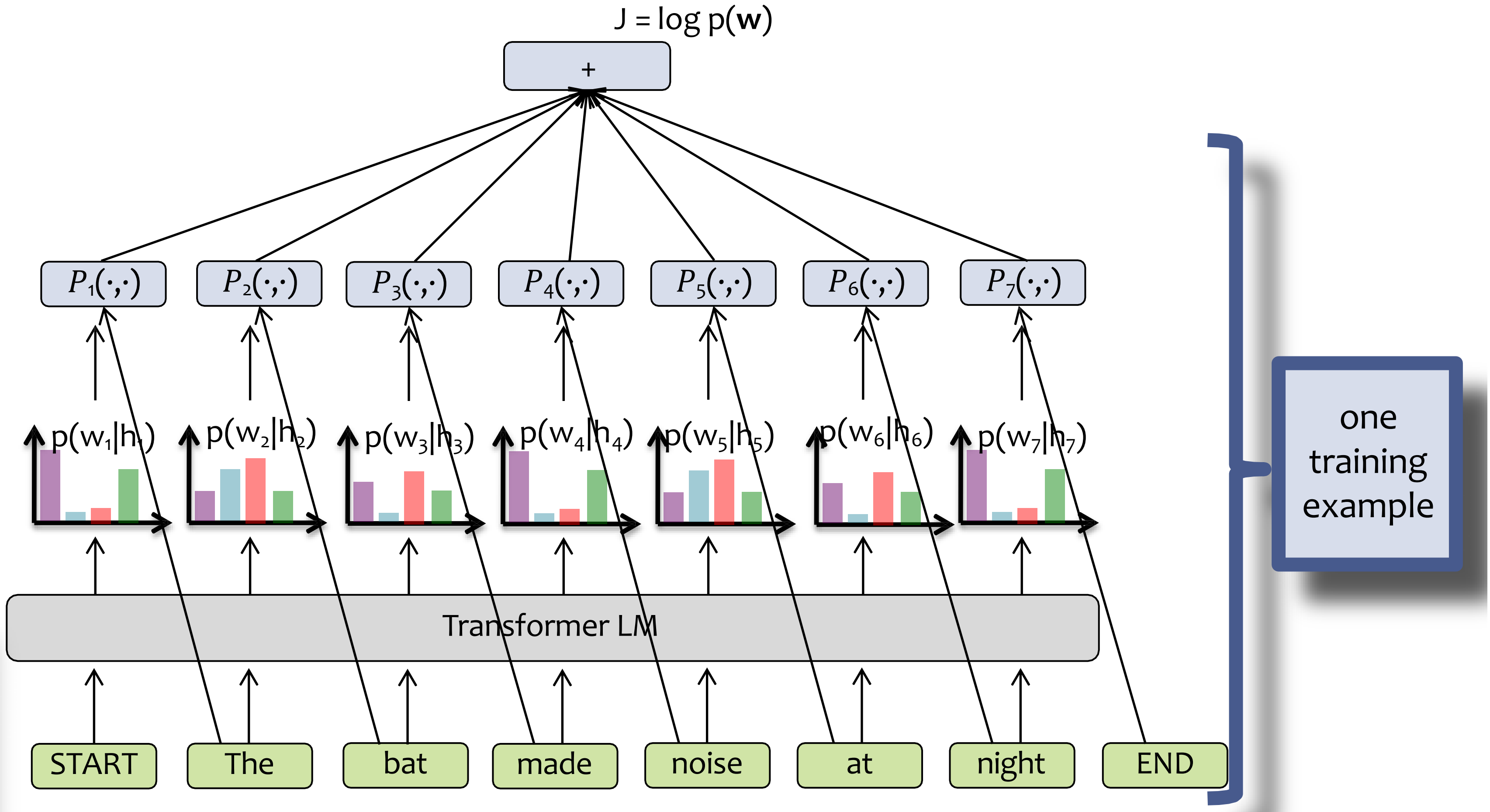

96

# Learning a Transformer LM



- 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)

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




Training a Transformer-LM is the same, except we swap in a different deep language model.


97

# Language Modeling

**An aside:**

- State-of-the-art language models currently tend to rely on **transformer networks** (e.g. GPT-3)
- RNN-LMs comprised most of the early neural LMs that **led to** current SOTA architectures

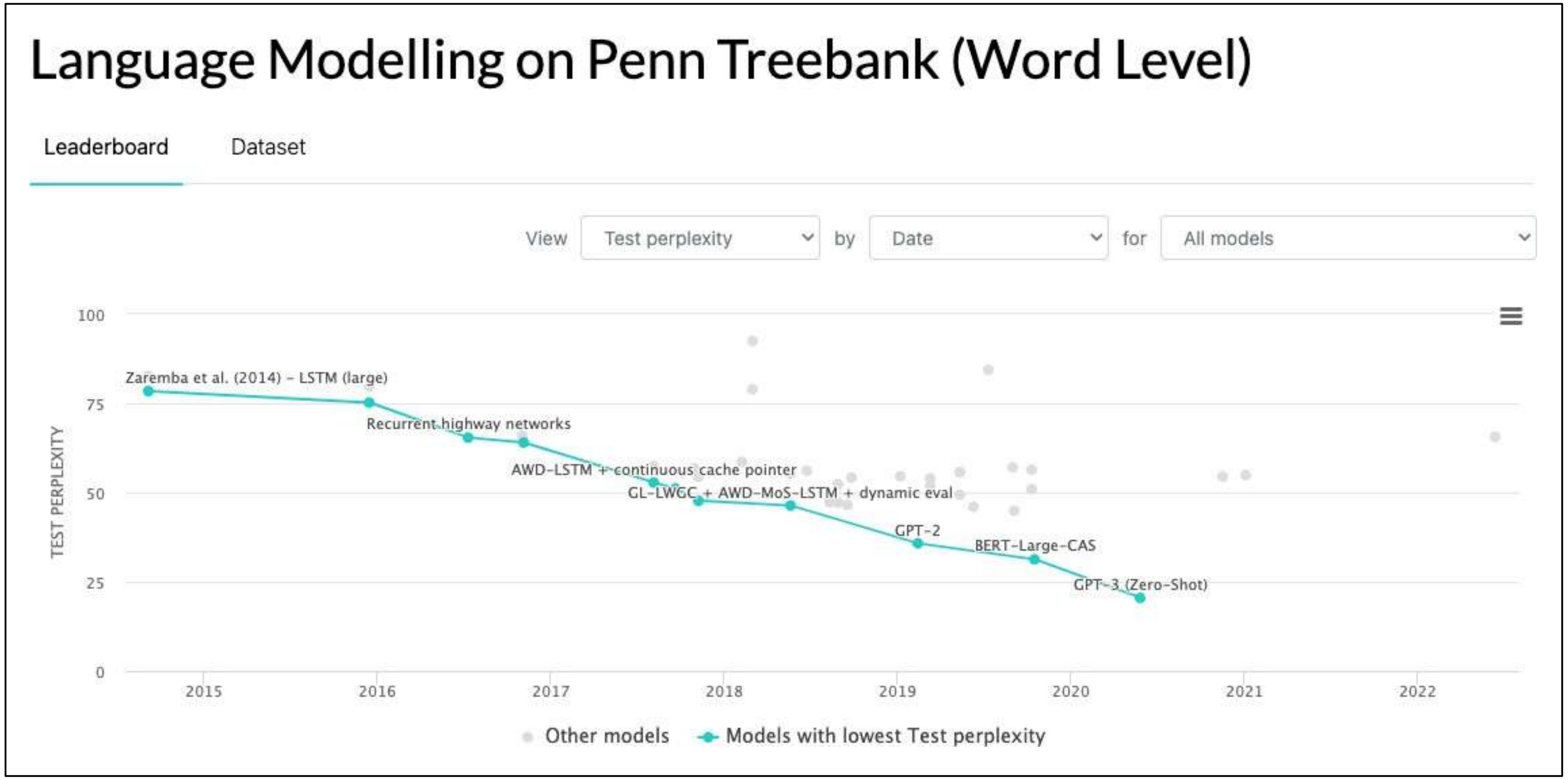



Figure from https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

# GPT-3

上海科技大学
ShanghaiTech University

- GPT stands for Generative Pre-trained Transformer
- GPT is just a Transformer LM, but with a huge number of parameters

| Model | # layers | dimension of states | dimension of inner states | # attention heads | # params |
|---|---|---|---|---|---|
| GPT (2018) | 12 | 768 | 3072 | 12 | 117M |
| GPT-2 (2019) | 48 | 1600 | -- | -- | 1542M |
| GPT-3 (2020) | 96 | 12288 | 4*12288 | 96 | 175000M |

# Why does efficiency matter?

上海科技大学
ShanghaiTech University

## Case Study: GPT-3

- # of training tokens = 500 billion
- # of parameters = 175 billion
- # of cycles = 50 petaflop/s-days (each of which are 8.64e+19 flops)

| Dataset | Quantity (tokens) | Weight in training mix | Epochs elapsed when training for 300B tokens |
|---|---|---|---|
| Common Crawl (filtered) | 410 billion | 60% | 0.44 |
| WebText2 | 19 billion | 22% | 2.9 |
| Books1 | 12 billion | 8% | 1.9 |
| Books2 | 55 billion | 8% | 0.43 |
| Wikipedia | 3 billion | 3% | 3.4 |

**Table 2.2: Datasets used to train GPT-3.** "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

| Model Name | $n_{params}$ | $n_{layers}$ | $d_{model}$ | $n_{heads}$ | $d_{head}$ | Batch Size | Learning Rate |
|---|---|---|---|---|---|---|---|
| GPT-3 Small | 125M | 12 | 768 | 12 | 64 | 0.5M | $6.0 \times 10^{-4}$ |
| GPT-3 Medium | 350M | 24 | 1024 | 16 | 64 | 0.5M | $3.0 \times 10^{-4}$ |
| GPT-3 Large | 760M | 24 | 1536 | 16 | 96 | 0.5M | $2.5 \times 10^{-4}$ |
| GPT-3 XL | 1.3B | 24 | 2048 | 24 | 128 | 1M | $2.0 \times 10^{-4}$ |
| GPT-3 2.7B | 2.7B | 32 | 2560 | 32 | 80 | 1M | $1.6 \times 10^{-4}$ |
| GPT-3 6.7B | 6.7B | 32 | 4096 | 32 | 128 | 2M | $1.2 \times 10^{-4}$ |
| GPT-3 13B | 13.0B | 40 | 5140 | 40 | 128 | 2M | $1.0 \times 10^{-4}$ |
| GPT-3 175B or "GPT-3" | 175.0B | 96 | 12288 | 96 | 128 | 3.2M | $0.6 \times 10^{-4}$ |

**Table 2.1:** Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

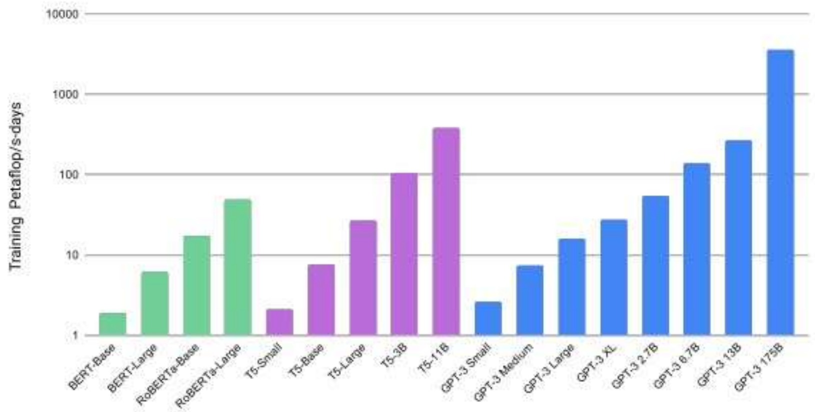


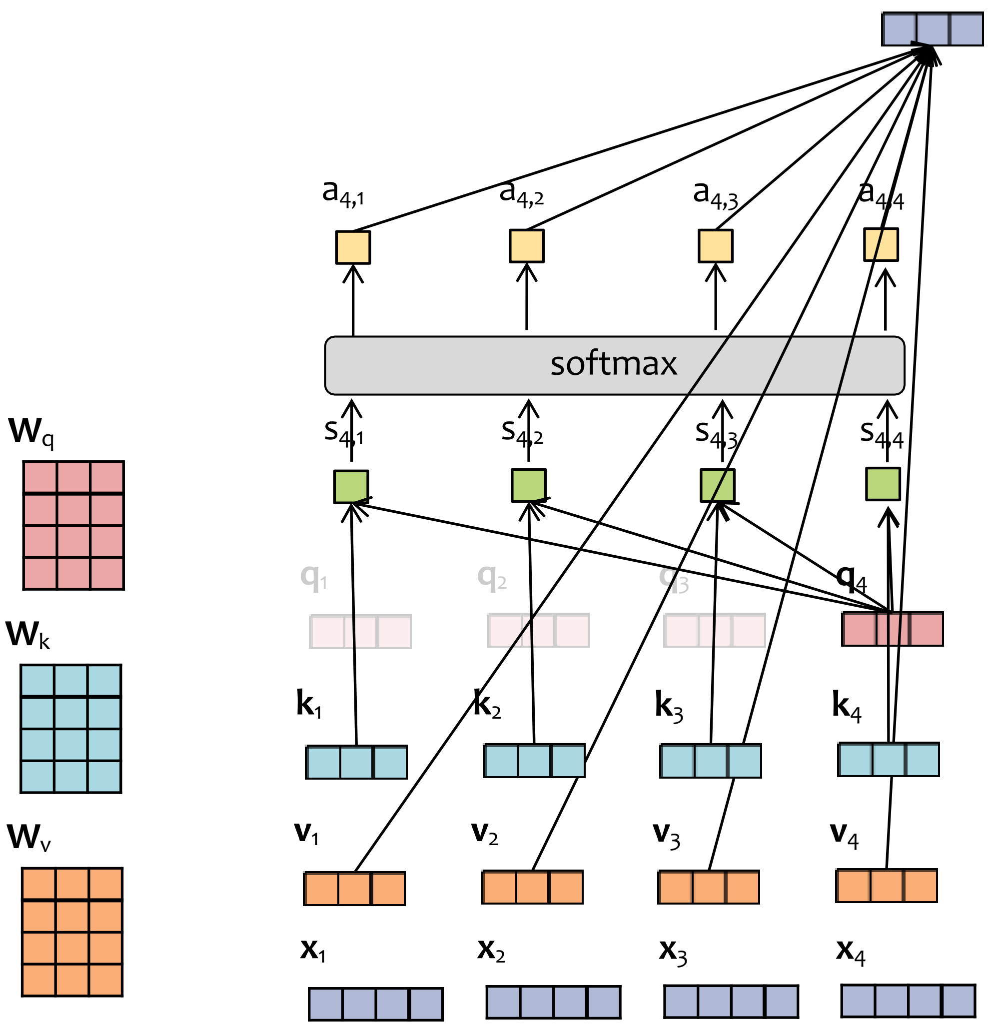

**Figure 2.2: Total compute used during training.** Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

Figure from https://arxiv.org/pdf/2005.14165.p

# IMPLEMENTING A TRANSFORMER LM

# Matrix Version of Single-Headed Attention






$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$attention weights

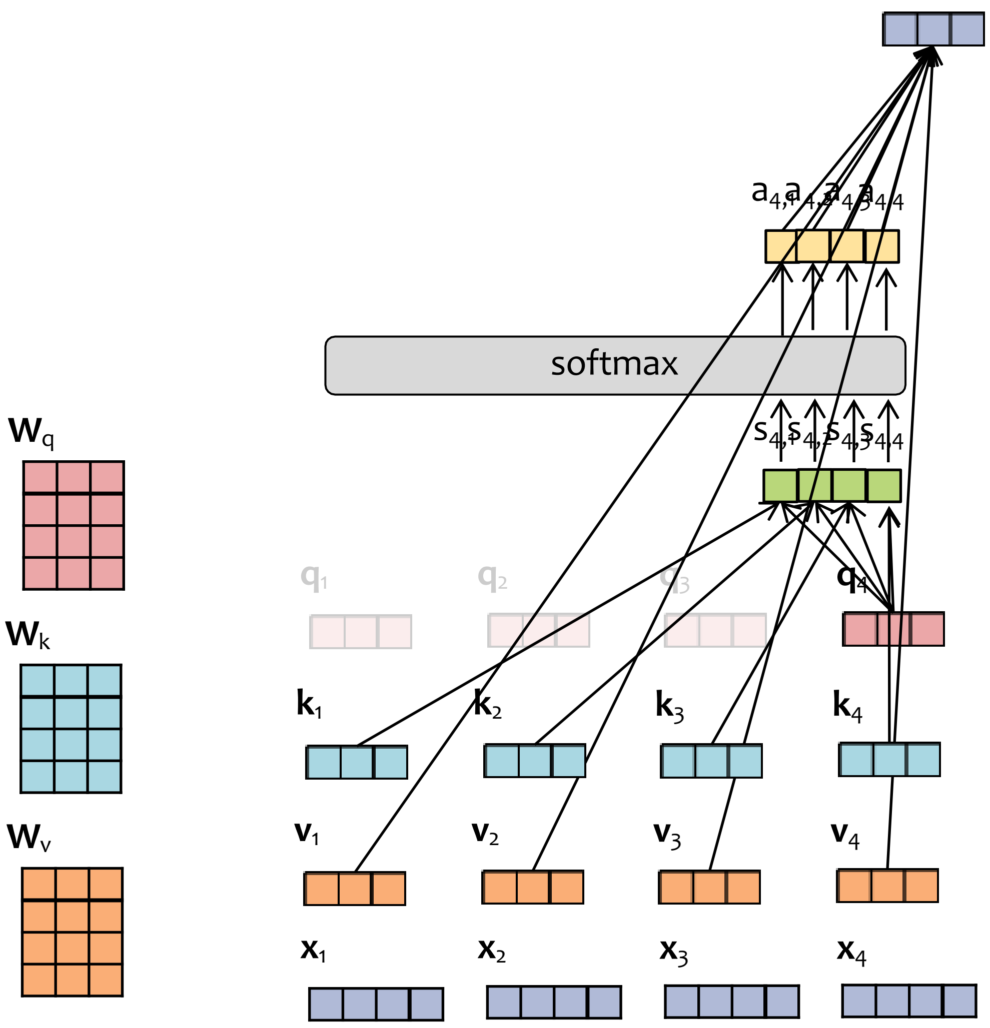
$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Matrix Version of Single-Headed Attention




$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

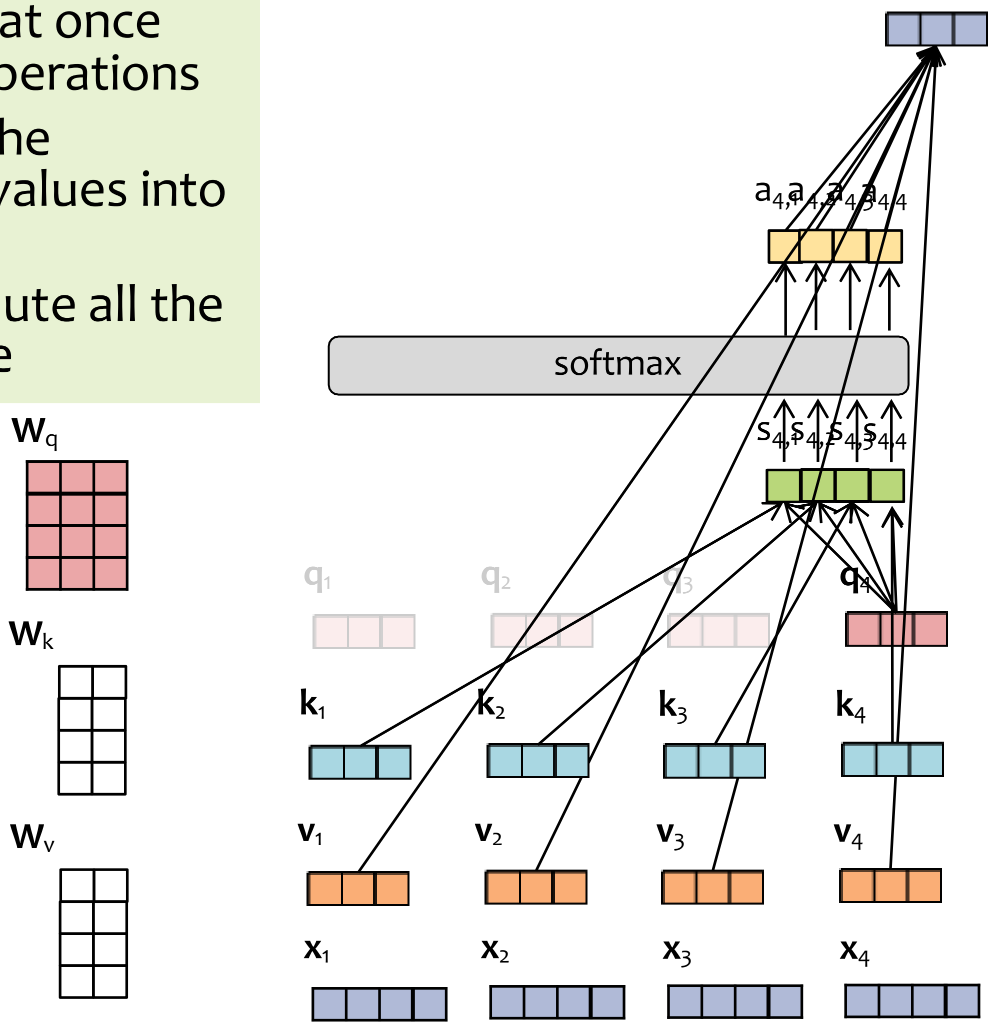
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Matrix Version of Single-Headed Attention

- For speed, we compute all the queries at once using matrix operations
- First we pack the queries, keys, values into matrices
- Then we compute all the queries at once




$$X' = AV = \text{softmax}(QK^T / \sqrt{d_k})V$$

$$A = [a_1, \ldots, a_4]^T = \text{softmax}(S)$$

$$S = [s_1, \ldots, s_4]^T = QK^T / \sqrt{d_k}$$

$$Q = [q_1, \ldots, q_4]^T = XW_q$$

$$K = [k_1, \ldots, k_4]^T = XW_k$$

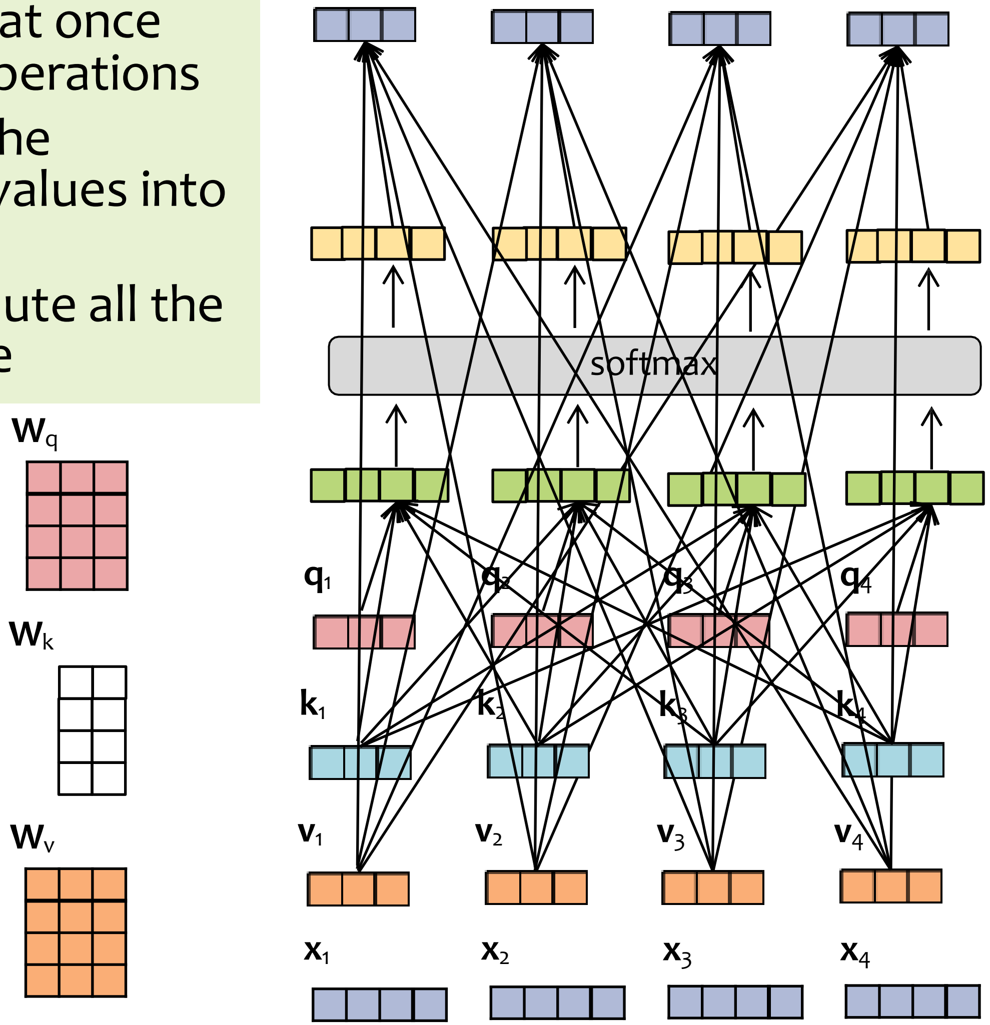
$$V = [v_1, \ldots, v_4]^T = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

# Matrix Version of Single-Headed Attention

上海科技大学 ShanghaiTech University

- For speed, we compute all the queries at once using matrix operations
- First we pack the queries, keys, values into matrices
- Then we compute all the queries at once




$$X' = AV = \text{softmax}(QK^T/\sqrt{d_k})V$$

$$A = [a_1, \ldots, a_4]^T = \text{softmax}(S)$$

$$S = [s_1, \ldots, s_4]^T = QK^T/\sqrt{d_k}$$

$$Q = [q_1, \ldots, q_4]^T = XW_q$$

$$K = [k_1, \ldots, k_4]^T = XW_k$$

$$V = [v_1, \ldots, v_4]^T = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

# Matrix Version of Single-Headed Attention

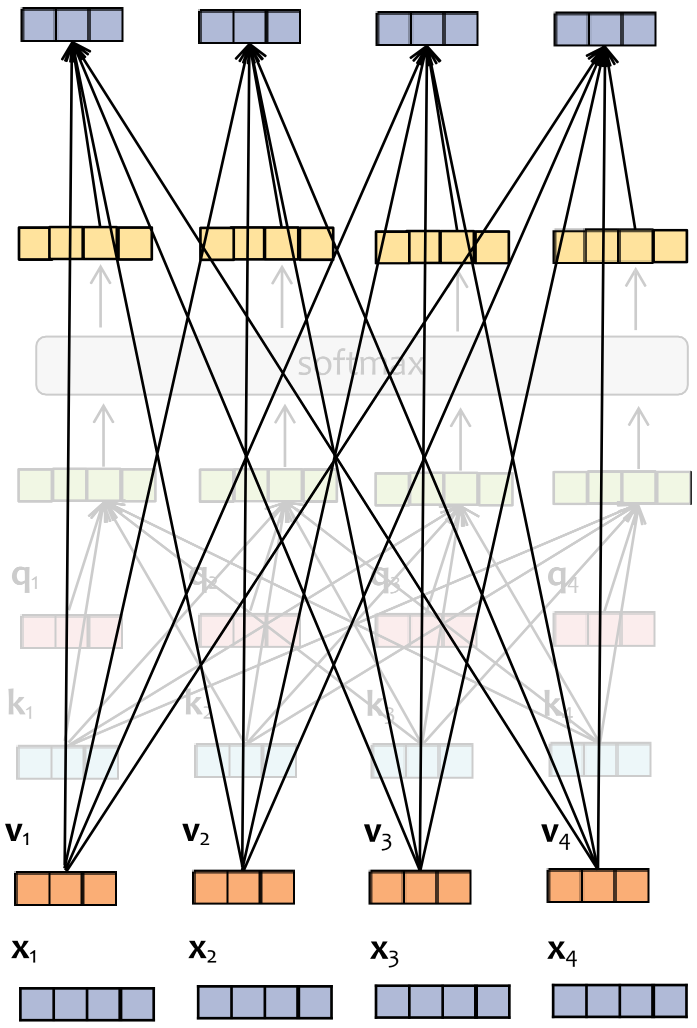
上海科技大学 ShanghaiTech University

Holy cow, that's a lot of new arrows… do we always want/need all of those?

- Suppose we're training our transformer to predict the next token(s) given the input…
- … then attending to tokens that come after the current token is cheating!

So what is this model?

- This version is the *standard* Transformer block. (more on this later!)
- But we want the Transformer LM block
- And that requires masking!




$$X' = AV = \text{softmax}(QK^T / \sqrt{d_k})V$$

$$A = [a_1, \dots, a_4]^T = \text{softmax}(S)$$

$$S = [s_1, \dots, s_4]^T = QK^T / \sqrt{d_k}$$

$$Q = [q_1, \dots, q_4]^T = XW_q$$

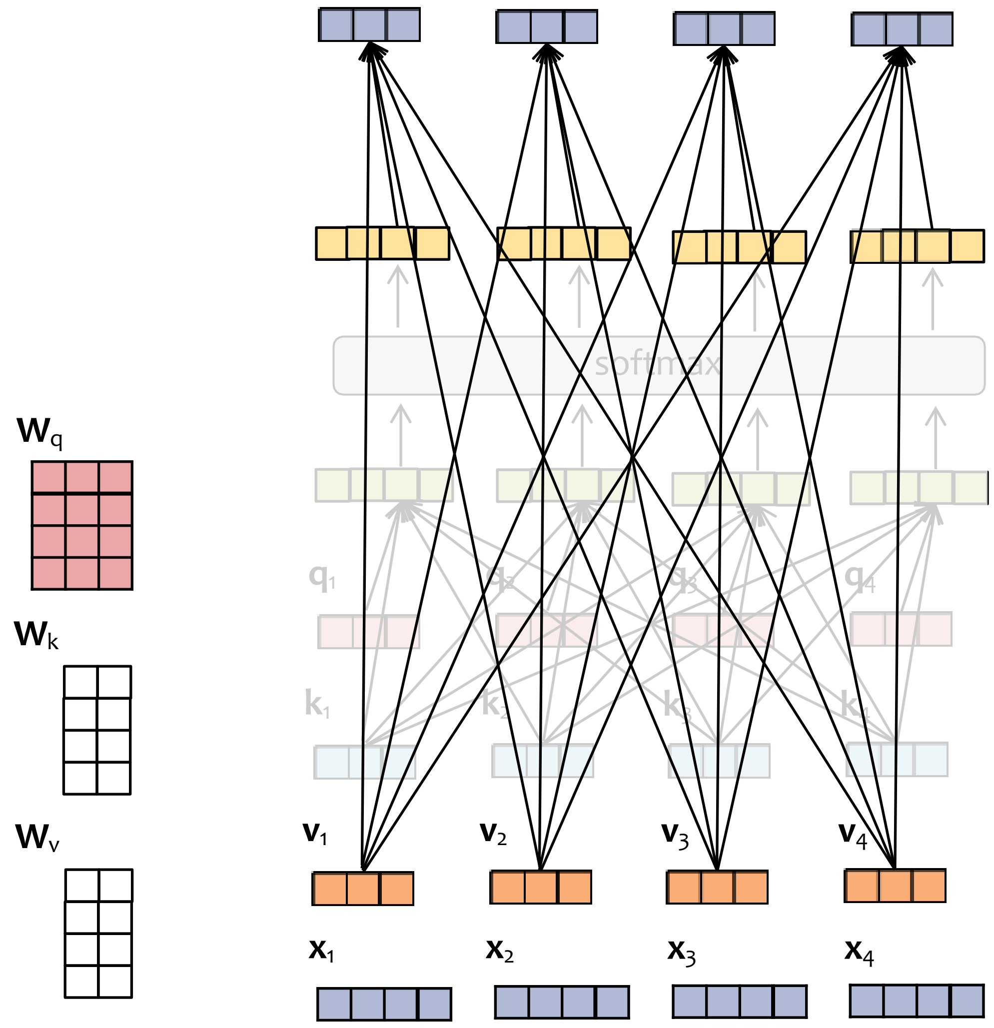
$$K = [k_1, \dots, k_4]^T = XW_k$$

$$V = [v_1, \dots, v_4]^T = XW_v$$

$$X = [x_1, \dots, x_4]^T$$

# Matrix Version of Single-Headed Attention






$$X' = AV = \text{softmax}(QK^T / \sqrt{d_k})V$$

$$A = \text{softmax}(S)$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_q$$

$$K = XW_k$$

$$V = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

**Question:** How is the softmax applied?

A. column-wise

B. row-wise

**Answer:**

# Matrix Version of Single-Headed (Causal) Attention

上海科技大学 ShanghaiTech University

**Insight:** if some element in the input to the softmax is -∞, then the corresponding output is 0!

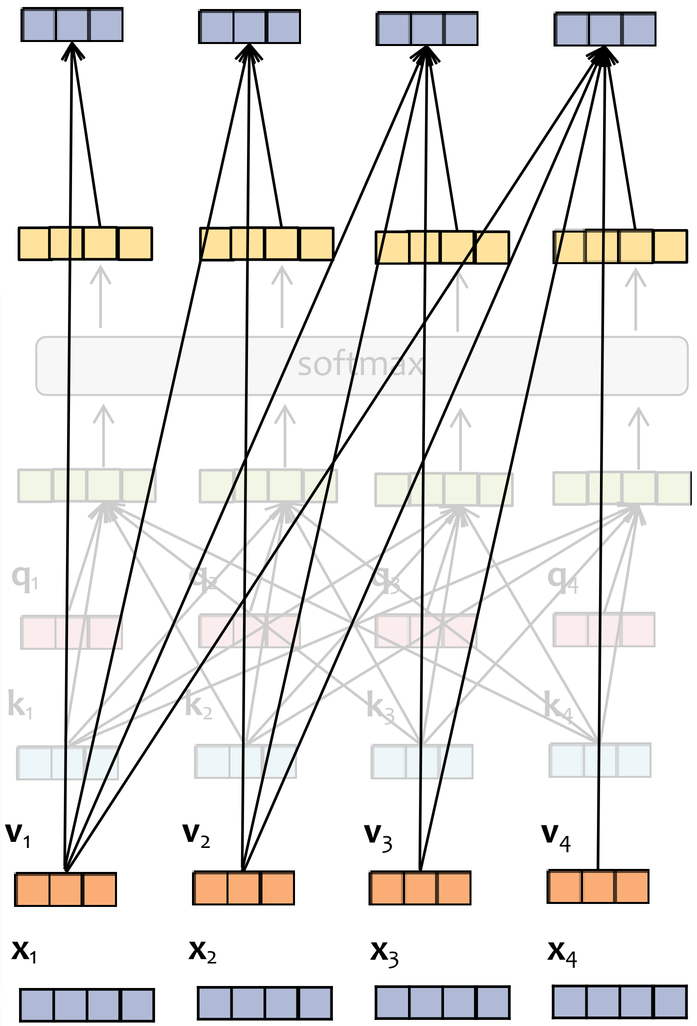
**Question:** For a causal LM which is the correct matrix?

A: $M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -\infty & 0 & 0 & 0 \\ -\infty & -\infty & 0 & 0 \\ -\infty & -\infty & -\infty & 0 \end{bmatrix}$

B: $M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$

C: $M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty \\ -\infty & -\infty & -\infty & 0 \end{bmatrix}$

**Answer:**




$$X' = AV = \text{softmax}(QK / \sqrt{d_k})V$$

$$A_{\text{causal}} = \text{softmax}(S + M)$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_q$$

$$K = XW_k$$

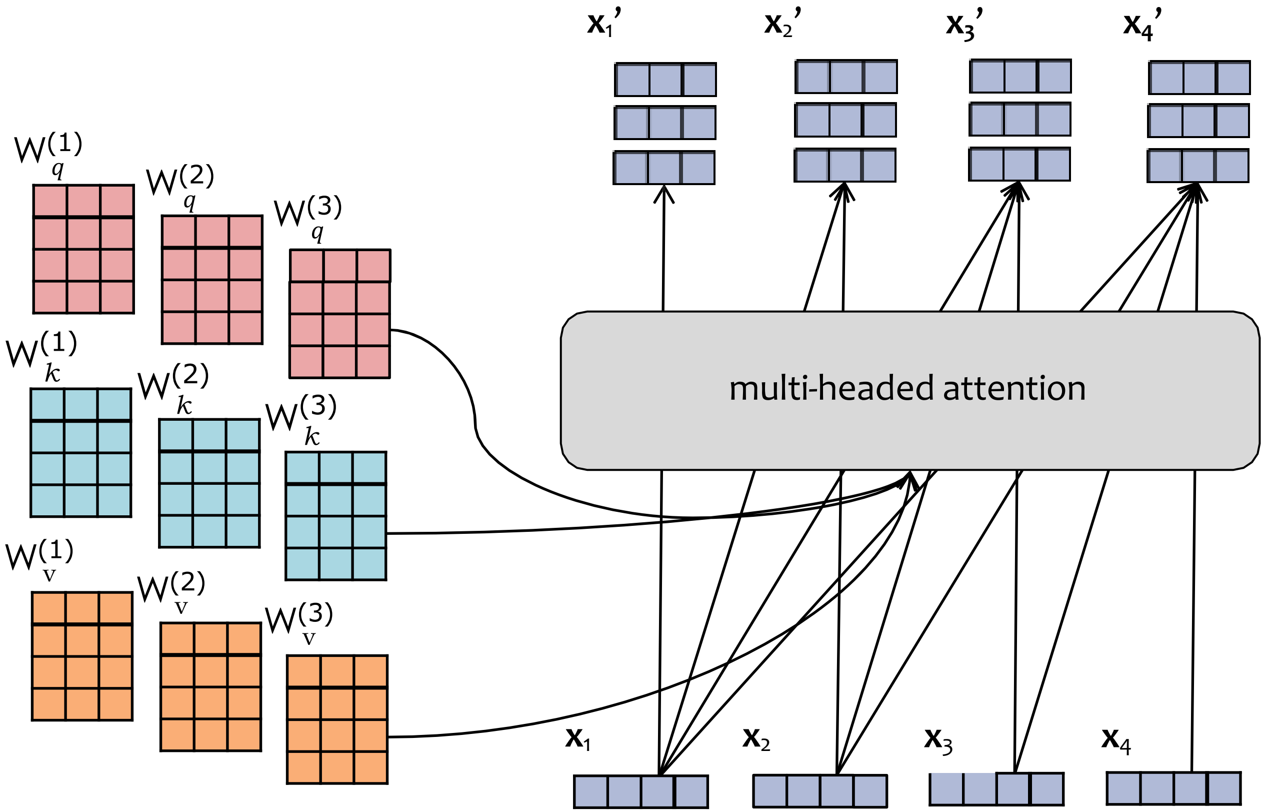
$$V = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

In practice, the attention weights are computed for all time steps T, then we mask out (by setting to –inf) all the inputs to the softmax that are for the timesteps to the right of the query.

# Matrix Version of Multi-Headed (Causal) Attention

上海科技大学 ShanghaiTech University




$$X = \text{concat}(X'^{(1)}, X'^{(2)}, X'^{(3)})$$

$$X'^{(i)} = \text{softmax}\left(\frac{Q^{(i)}(K^{(i)})^T}{\sqrt{d_k}} + M\right) V^{(i)}$$

$$Q^{(i)} = XW_q^{(i)}$$

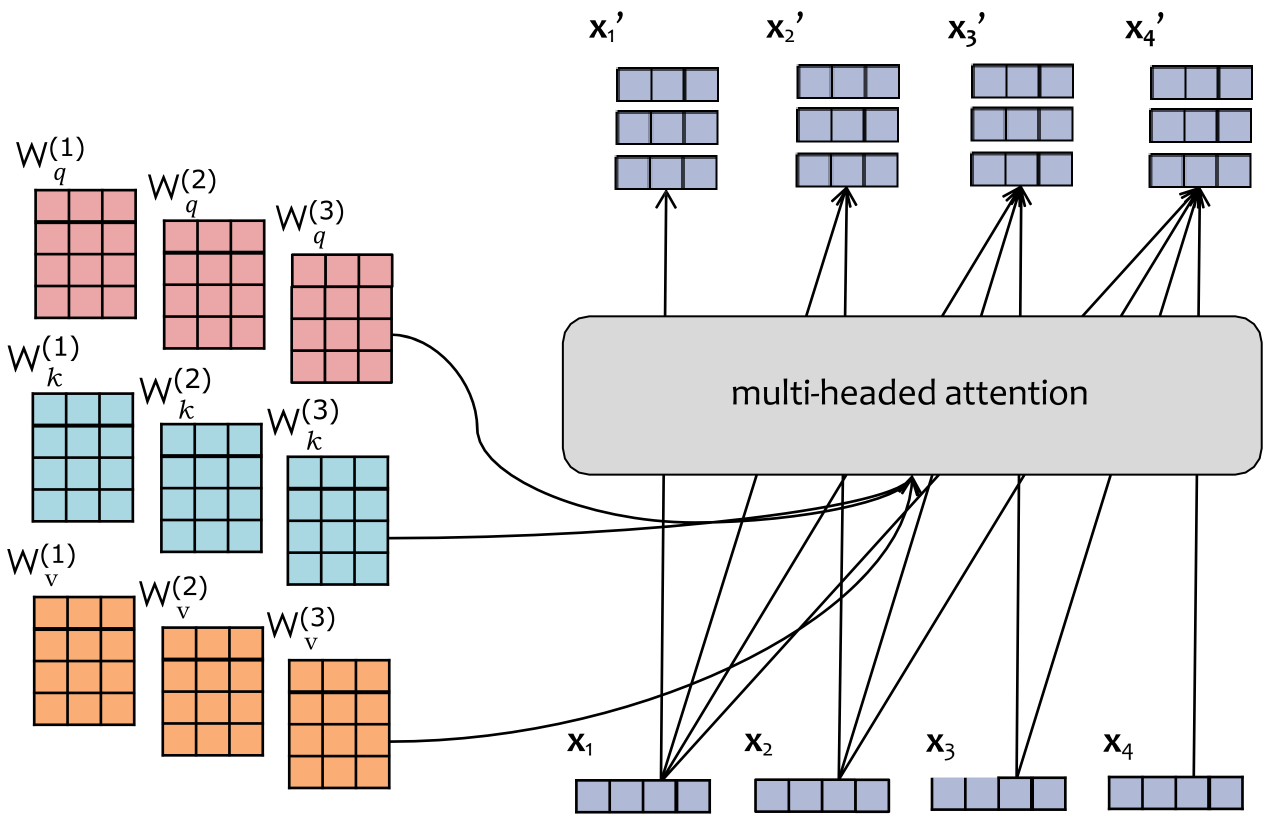
$$K^{(i)} = XW_k^{(i)}$$

$$V^{(i)} = XW_v^{(i)}$$

$$X = [x_1, \ldots, x_4]^T$$

# Matrix Version of Multi-Headed (Causal) Attention

上海科技大学 ShanghaiTech University




$$X = \text{concat}(X'^{(1)}, \ldots, X'^{(h)})$$

$$X'^{(i)} = \text{softmax}\left(\frac{Q^{(i)}(K^{(i)})^T}{\sqrt{d_k}} + M\right) V^{(i)}$$

$$Q^{(i)} = XW_q^{(i)}$$

$$K^{(i)} = XW_k^{(i)}$$

$$V^{(i)} = XW_v^{(i)}$$

$$X = [x_1, \ldots, x_4]^T$$

**Recall:**

To ensure the dimension of the **input** embedding $\mathbf{x}_t$ is the same as the **output** embedding $\mathbf{x}_t$', Transformers usually choose the embedding sizes and number of heads appropriately:

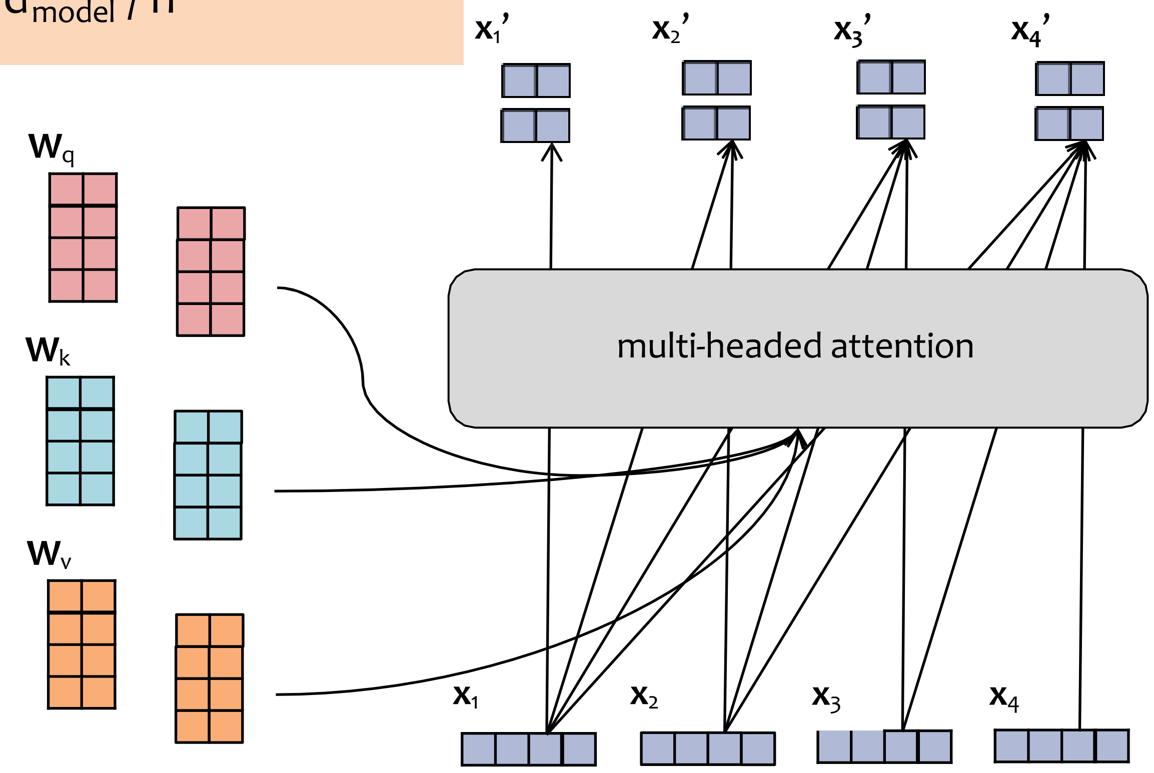
- $d_{model}$ = dim. of inputs
- $d_k$ = dim. of each output
- h = # of heads
- Choose $d_k = d_{model} / h$

# Multi-Headed (Causal) Attention

上海科技大学 ShanghaiTech University

$$X = \text{concat}(X'^{(1)}, \ldots, X'^{(h)})$$

$$X'^{(i)} = \text{softmax}\left(\frac{Q^{(i)}(K^{(i)})^T}{\sqrt{d_k}} + M\right) V^{(i)}$$




$$Q^{(i)} = XW_q^{(i)}$$

$$K^{(i)} = XW_k^{(i)}$$

$$V^{(i)} = XW_v^{(i)}$$

$$X = [x_1, \ldots, x_4]^T$$