10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

# RNN LMs
# +
# Transformer LMs

Matt Gormley & Geoff Gordon
Lecture 18
Oct. 29, 2025

# Reminders

- **Homework 6: Learning Theory & Generative Models**
  - Out: Mon, Oct 27
  - Due: Sat, Nov 01 at 11:59pm
  - (only two grace/late days permitted)
- **Programming Quiz 2: Fri, Oct 31, in-class**
  - Focus: HW4 and HW5 programming
- **Exam 2: Thu, Nov 6, 7:00 pm – 9:00 pm**
  - Scope: Lectures 8 - 16

# EXAM 2 LOGISTICS

# Exam 2

- **Time / Location**
  - **Time:** **Thu, Nov. 6, 7:00pm – 9:00pm**
  - **Location & Seats:** You have all been split across multiple rooms. Everyone has an assigned seat in one of these room. Please watch Piazza carefully for announcements.
- **Logistics**
  - Covered material: Lecture 8 – Lecture 16
  - Format of questions:
    - Multiple choice
    - True / False (with justification)
    - Derivations
    - Short answers
    - Interpreting figures
    - Implementing algorithms on paper
  - No electronic devices
  - You are allowed to **bring** one 8½ x 11 sheet of notes (front and back, handwritten with pen/pencil or tablet)

# Topics for Exam 1

- Foundations
  - Probability, Linear Algebra, Geometry, Calculus
  - Optimization
- Important Concepts
  - Overfitting
  - Experimental Design
- Classification
  - Decision Tree
  - KNN
  - Perceptron
- Regression
  - KNN Regression
  - Decision Tree Regression
  - Linear Regression

# Topics for Exam 2

- Classification
  - Binary Logistic Regression
- Important Concepts
  - Stochastic Gradient Descent
  - Regularization
  - Feature Engineering
- Feature Learning
  - Neural Networks
  - Basic NN Architectures
  - Backpropagation
- Learning Theory
  - PAC Learning
  - MLE / MAP
- Societal Impacts of ML
- Regression
  - Linear Regression

# RECURRENT NEURAL NETWORKS

# Recurrent Neural Networks (RNNs)

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

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

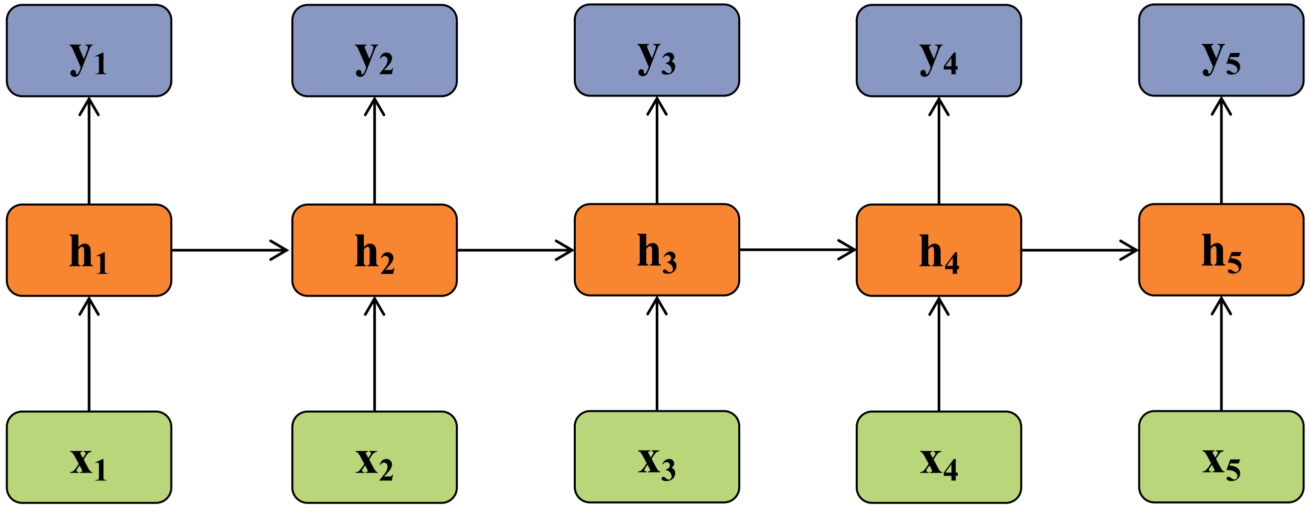
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

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

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



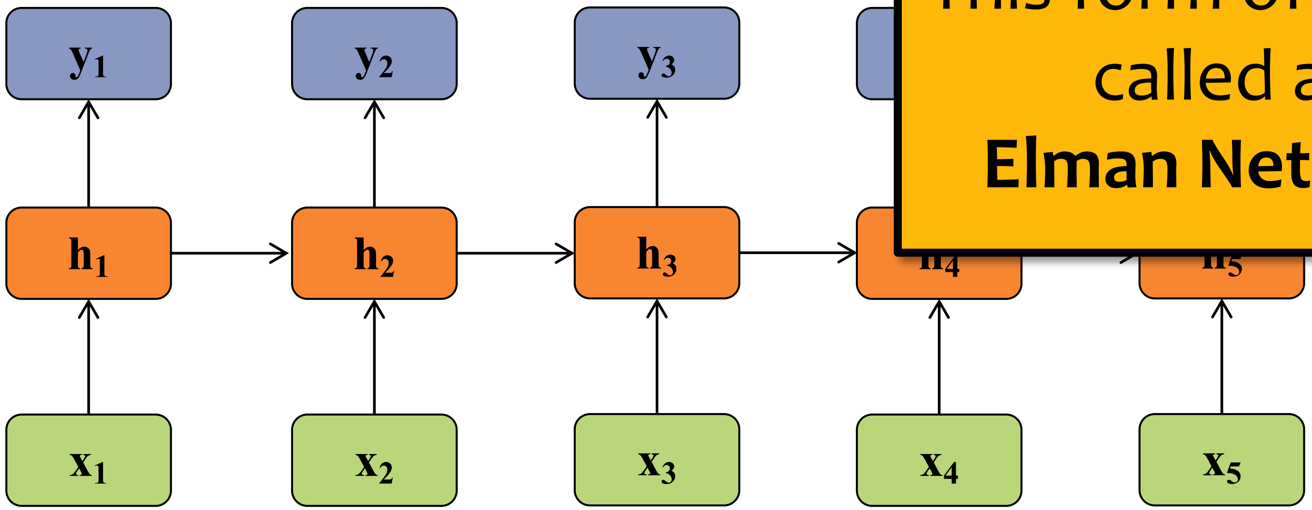
# Recurrent Neural Networks (RNNs)

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{hidden units: } \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

Definition of the RNN:

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

This form of RNN is called an **Elman Network**



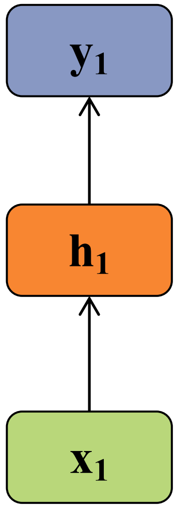
# Recurrent Neural Networks (RNNs)

$$\begin{aligned} \text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\ \text{hidden units: } & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\ \text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\ \text{nonlinearity: } & \mathcal{H} \end{aligned}$$

Definition of the RNN:

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

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




- If $T=1$, then we have a standard feed-forward neural net with one hidden layer, which requires **fixed size inputs/outputs**
- By contrast, an RNN can handle arbitrary length inputs/outputs because $T$ can vary from example to example
- The key idea is that we reuse the same parameters at every timestep, always building off of the previous hidden state

# A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

# A Recipe for Machine Learning

1\. 

- Recurrent Neural Networks (RNNs) provide another form of **decision function**
- An RNN is just another differential function

2\. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

4\. Train with SGD:

(take small steps opposite the gradient)

- We'll compute the gradient efficiently with backpropagation

$$- \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

# Bidirectional RNN

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

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

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

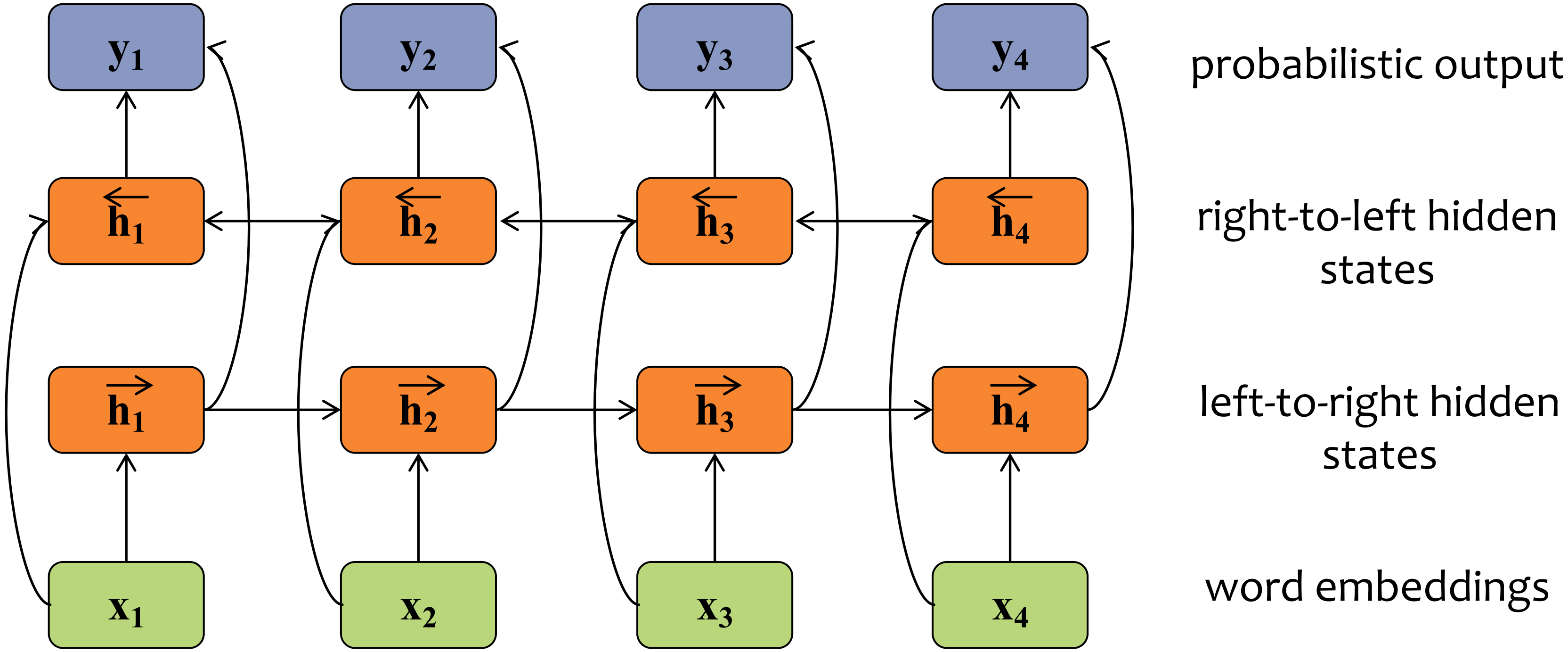
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



# Deep RNNs

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

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

nonlinearity: $\mathcal{H}$

Recursive Definition:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right)$$

$$y_t = W_{h^N y} h_t^N + b_y$$

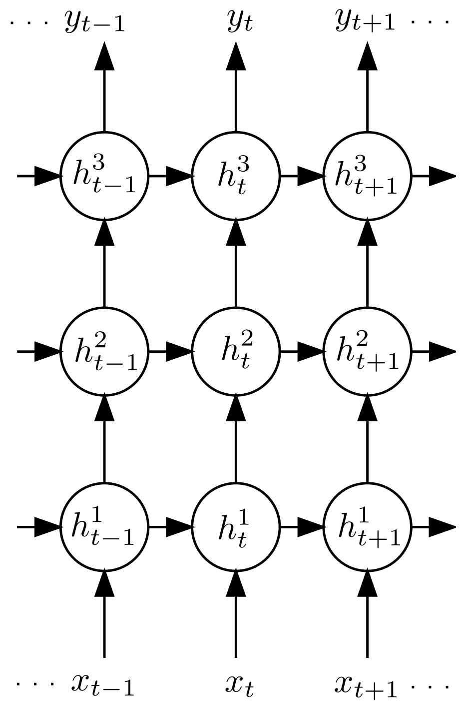
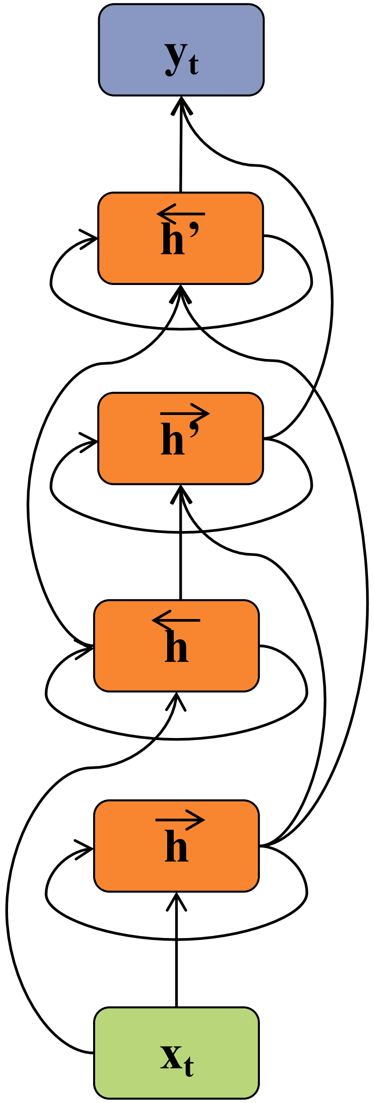



Figure from (Graves et al., 2013)

# Deep Bidirectional RNNs

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

- Notice that the upper level hidden units have input from **two previous layers** (i.e. wider input)
- Likewise for the output layer



# CNN & RNN Learning Objectives

*You should be able to…*

- Implement the common layers found in Convolutional Neural Networks (CNNs) such as linear layers, convolution layers, max-pooling layers, and rectified linear units (ReLU)
- Explain how the shared parameters of a convolutional layer could learn to detect spatial patterns in an image
- Describe the backpropagation algorithm for a CNN
- Identify the parameter sharing used in a basic recurrent neural network, e.g. an Elman network
- Apply a recurrent neural network to model sequence data
- Differentiate between an RNN and an RNN-LM

# BACKGROUND:
# N-GRAM LANGUAGE MODELS

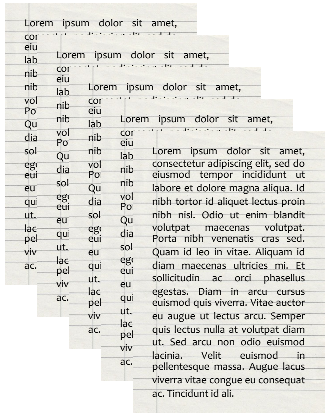
# Human Language Technologies

## Speech Recognition

## Machine Translation

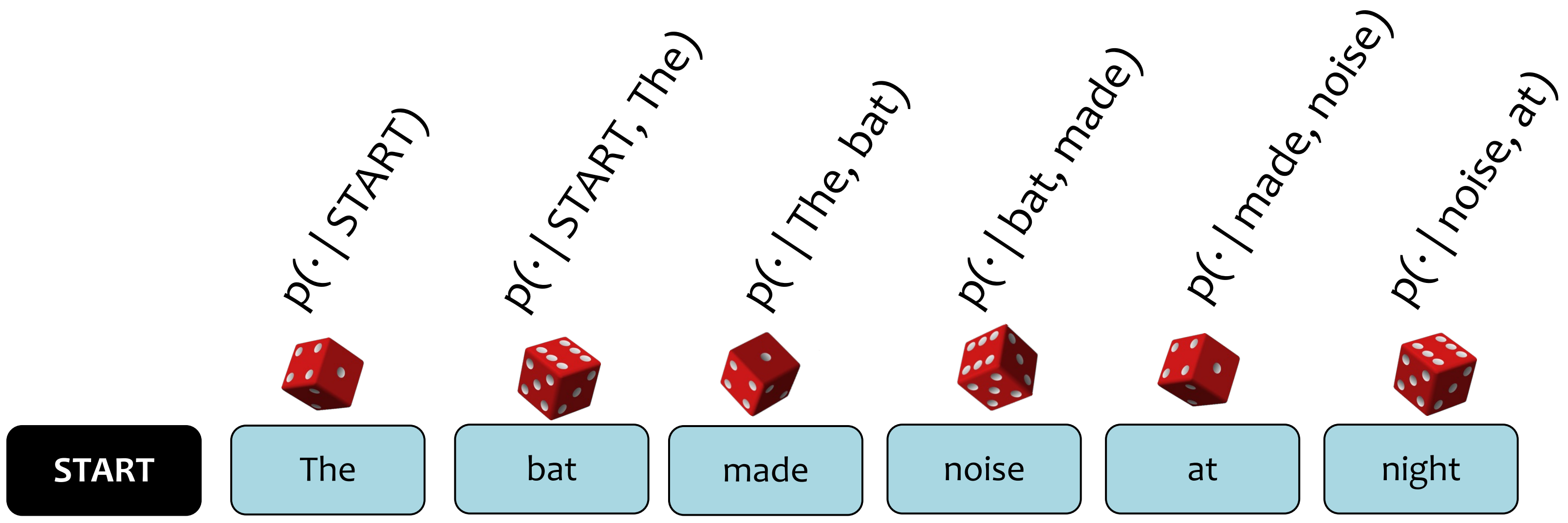
기계 번역은 특히 영어와 한국어와 같은 언어 쌍의 경우 매우 어렵습니다.

## Summarization



# 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

*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

*<u>Question</u>*: 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 $\quad p(w_1)$

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

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

*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**?...

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

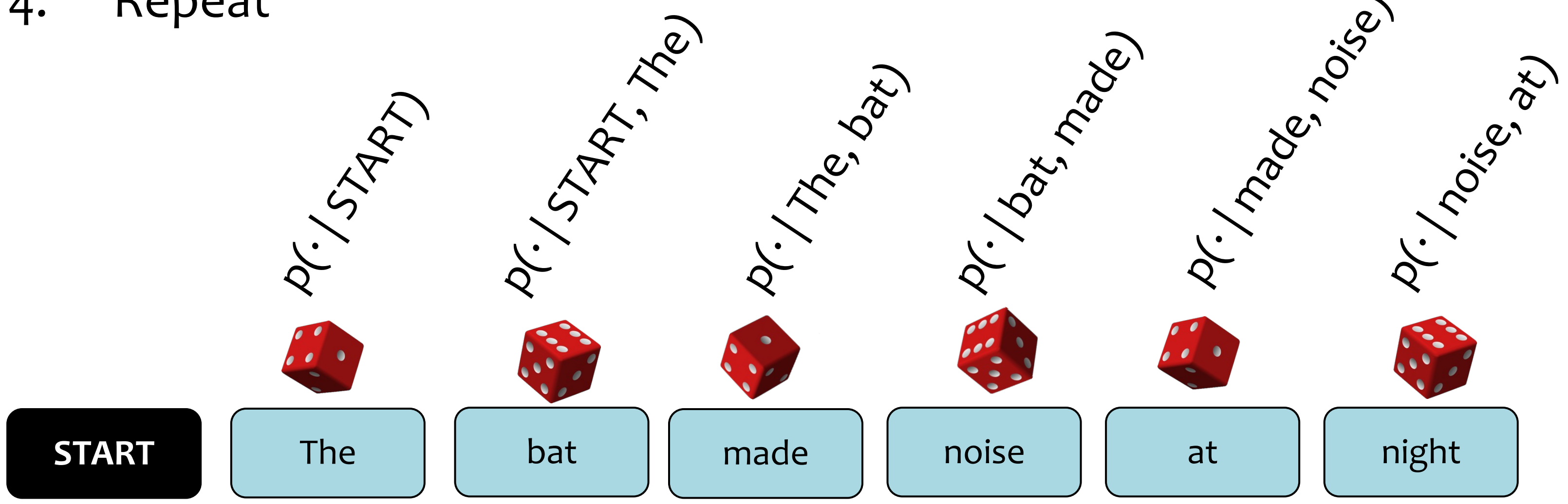
| $w_t$ | $p(\cdot \mid \cdot, \cdot)$ |
|---|---|
| corn | 4/11 |
| grass | 3/11 |
| hay | 2/11 |
| if | 1/11 |
| which | 1/11 |

# Sampling from a Language Model

*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



# 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

| 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;` |

# RECURRENT NEURAL NETWORK (RNN) LANGUAGE MODELS

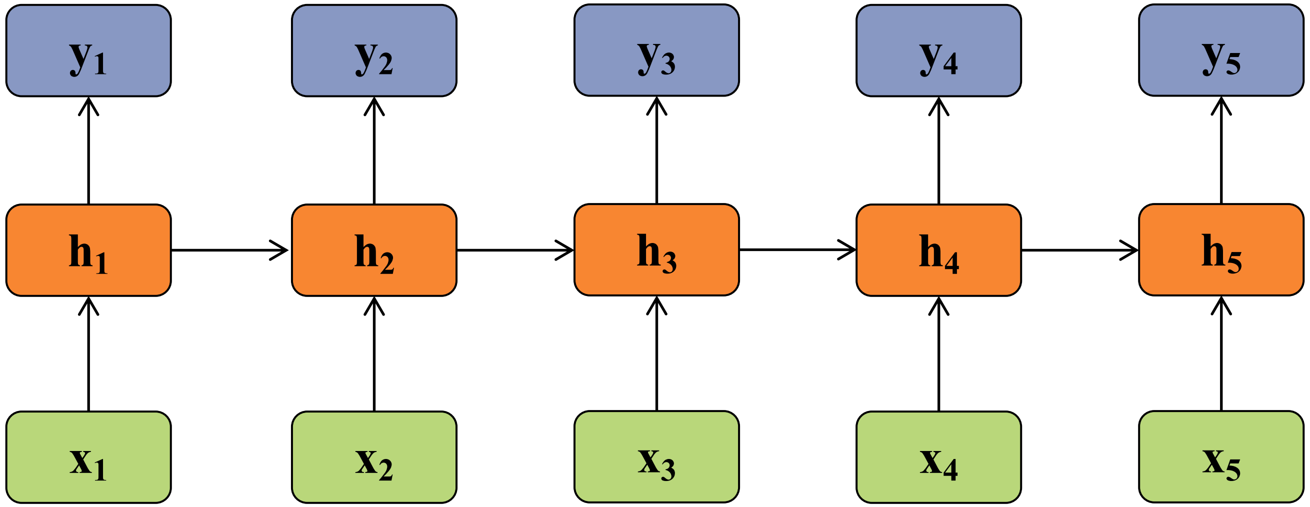
# Recurrent Neural Networks (RNNs)

$$
\begin{aligned}
\text{inputs:}\ & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units:}\ & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs:}\ & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity:}\ & \mathcal{H}
\end{aligned}
$$

Definition of the RNN:

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

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



# The Chain Rule of Probability

Recall...

*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

$p(w_1)$

The

The

The

The

The

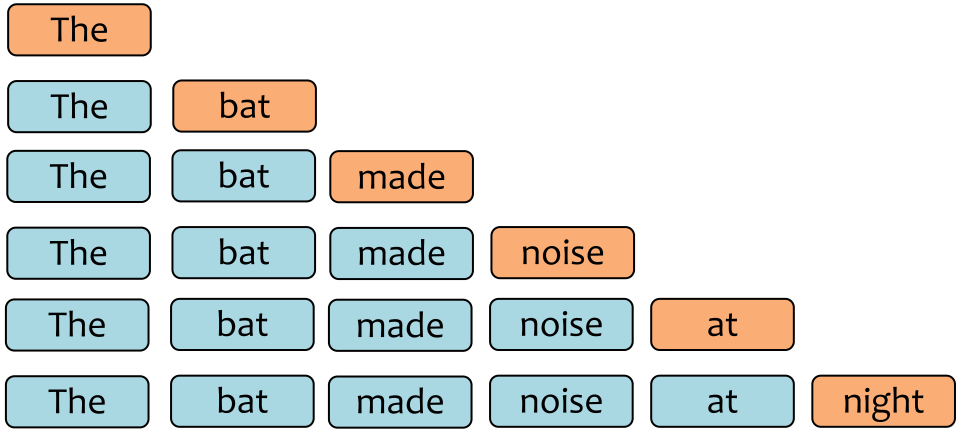
$p(w_6 \mid w_5, w_4, w_3, w_2, w_1)$

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

# RNN Language Model

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

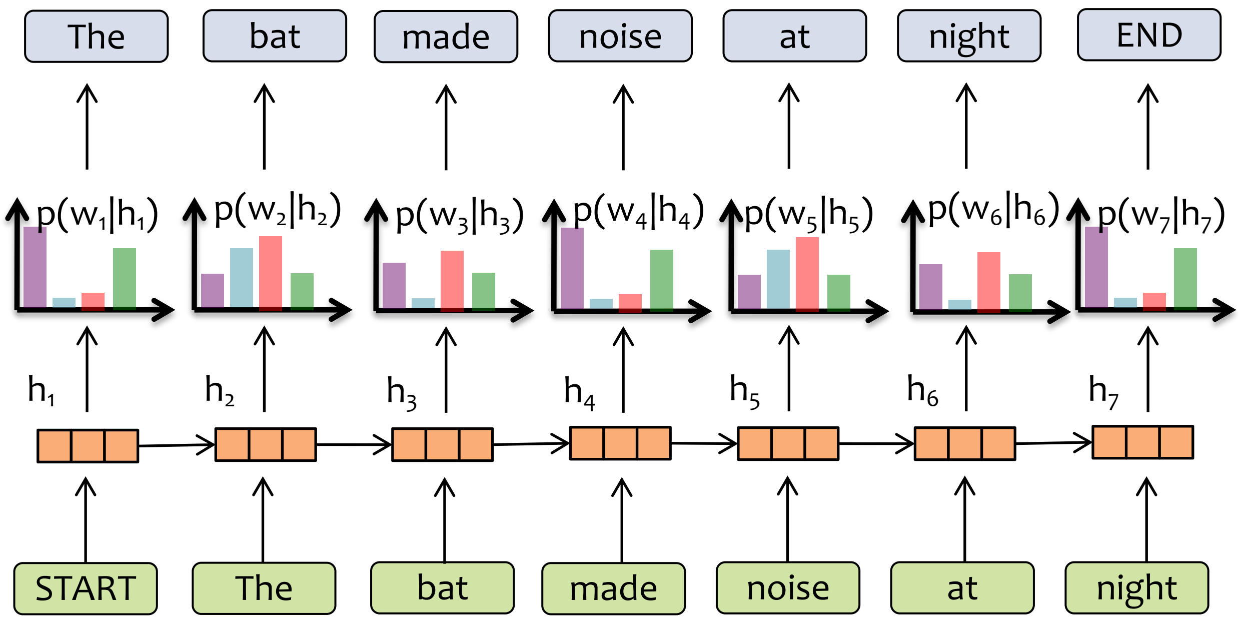
$p(w_1, w_2, w_3, \ldots, w_6) =$




$p(w_1)$
$p(w_2 \mid f_\theta(w_1))$
$p(w_3 \mid f_\theta(w_2, w_1))$
$p(w_4 \mid f_\theta(w_3, w_2, w_1))$
$p(w_5 \mid f_\theta(w_4, w_3, w_2, w_1))$
$p(w_6 \mid f_\theta(w_5, w_4, w_3, w_2, w_1))$

*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

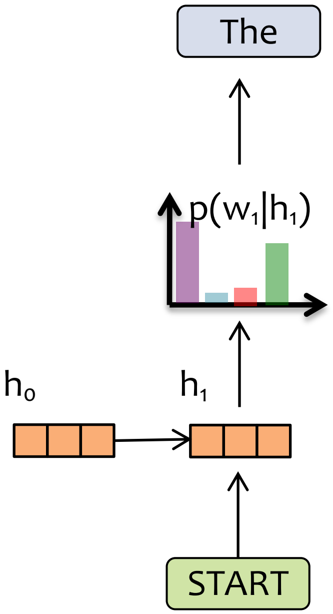



<u>*Key Idea*</u>:

(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

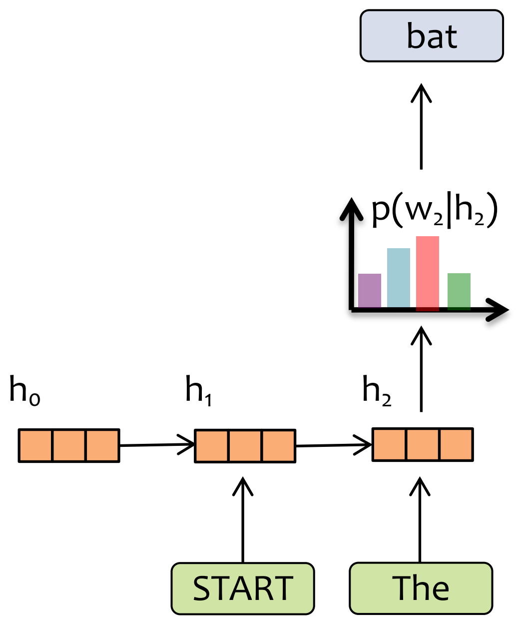



*<u>Key Idea</u>*:

(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

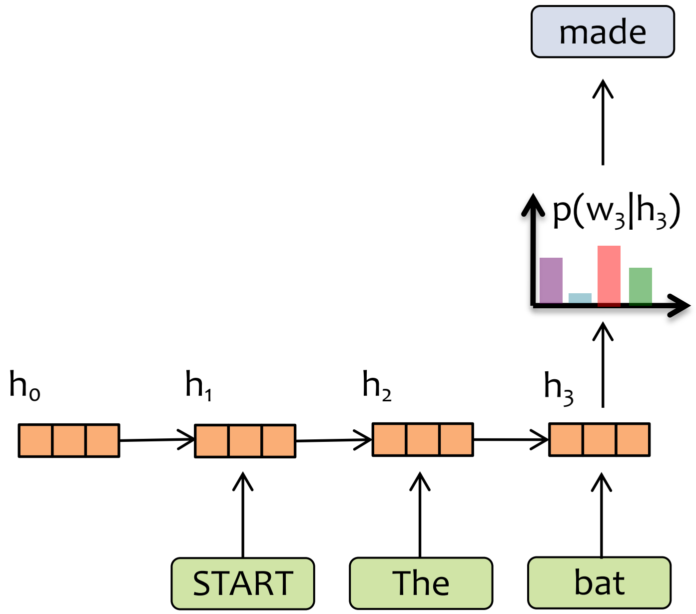



*<u>Key Idea</u>*:

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

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

# RNN Language Model

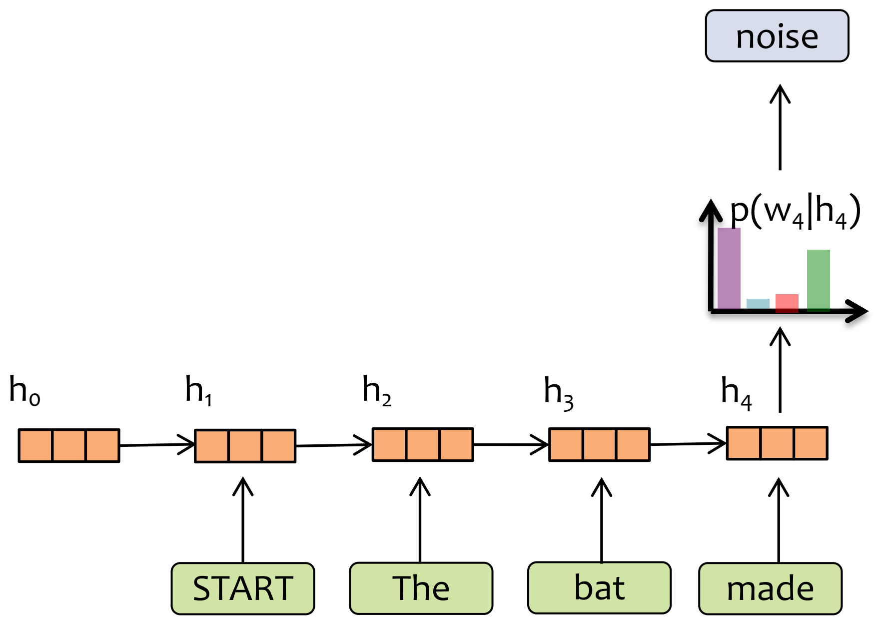



*Key Idea*:

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

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

# RNN Language Model

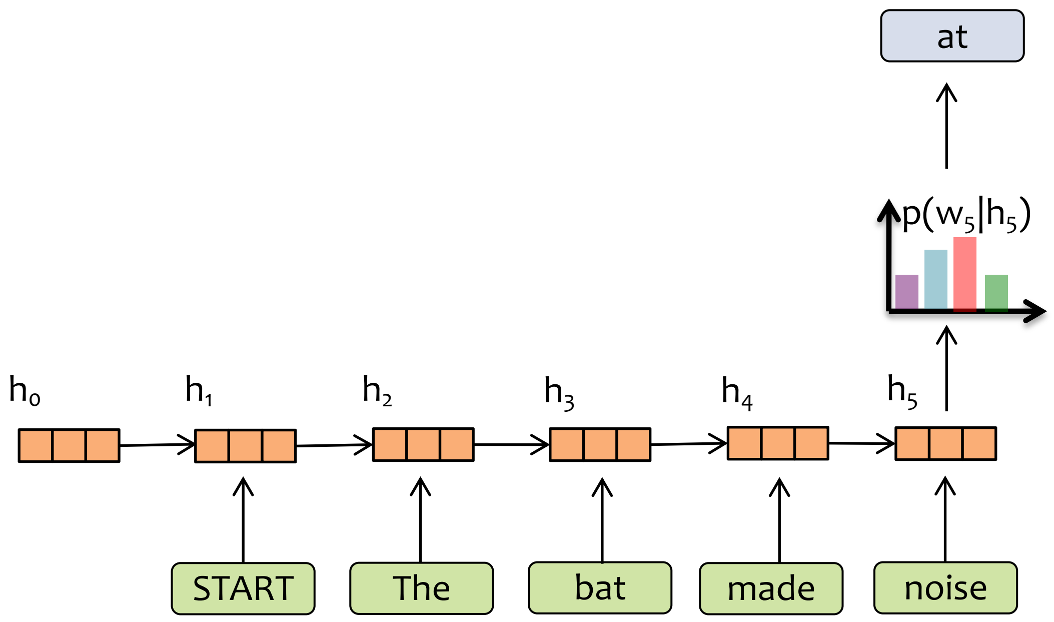



<u>*Key Idea*</u>:

(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




*Key Idea*:

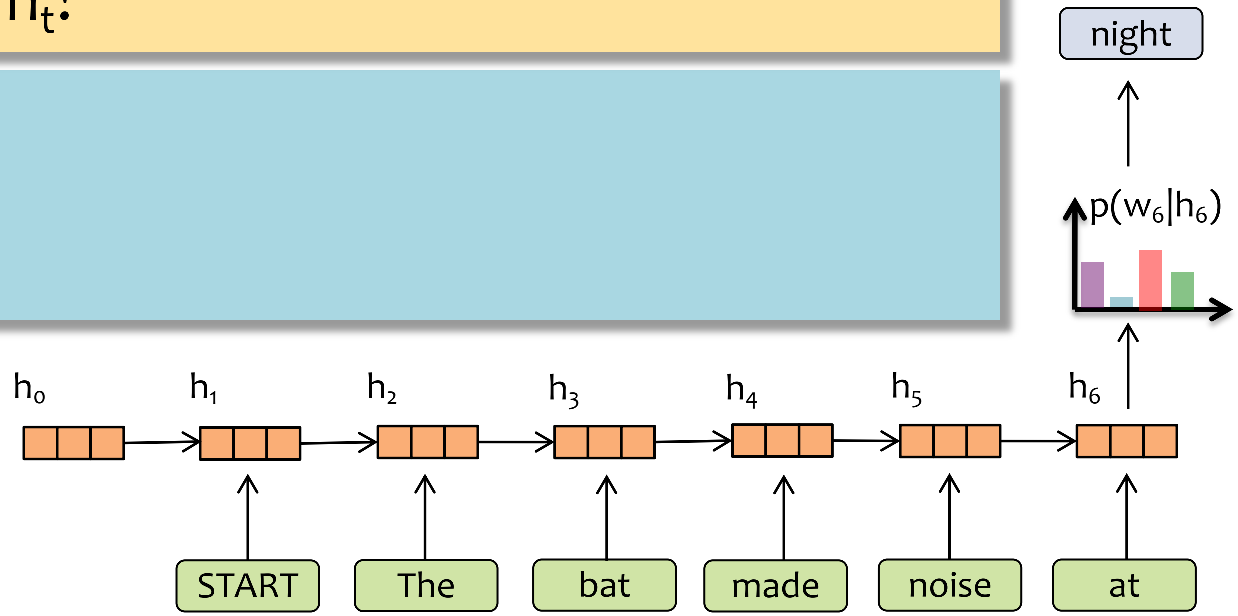
(1) convert all previous words to a **fixed length vector**

(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

**Poll Question 1:** How can we create a distribution $p(w_t|h_t)$ from $h_t$?

**Answer:**

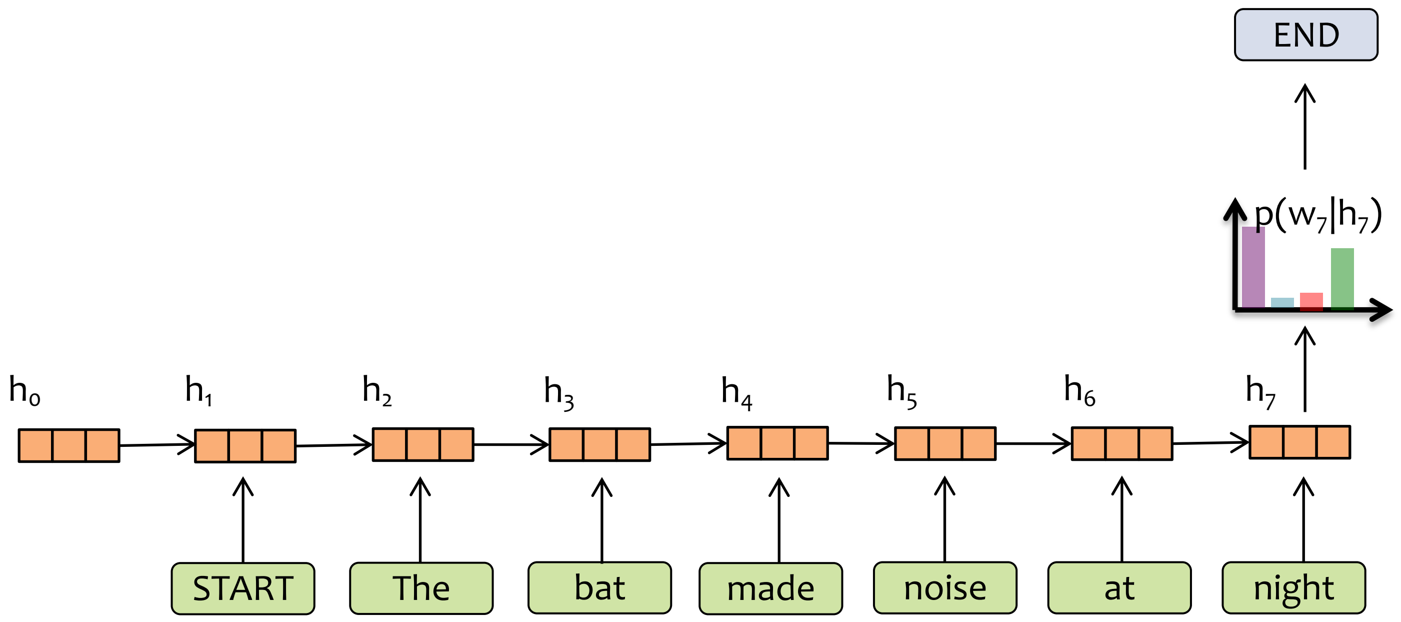



*Key Idea*:

(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

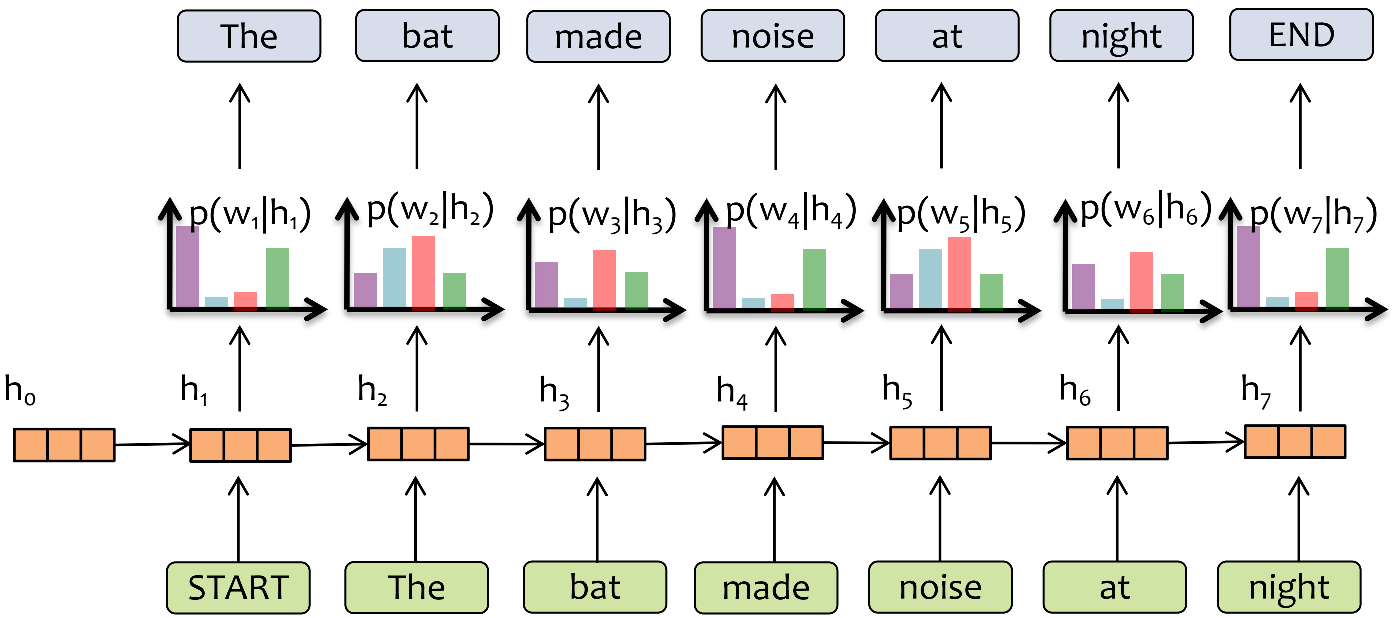



*Key Idea*:

(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



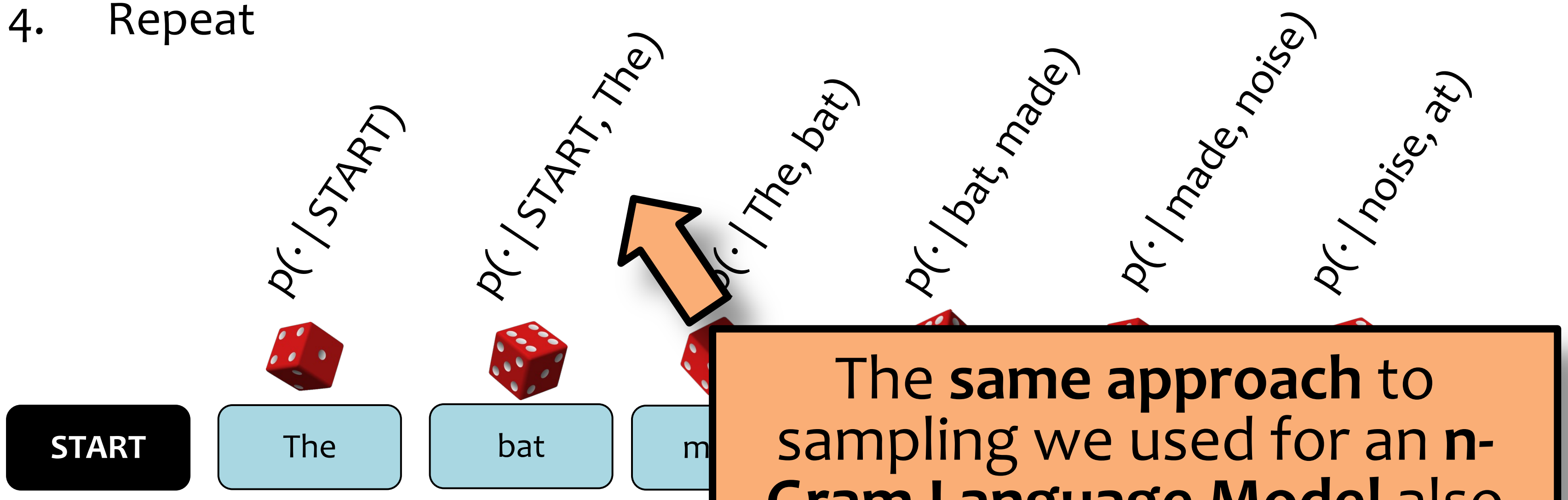

$$p(w_1, w_2, w_3, \dots, w_T) = p(w_1 \mid h_1)\, p(w_2 \mid h_2) \dots p(w_T \mid h_T)$$

# 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**

# Sampling from an RNN-LM

**??**

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy m there My power to give thee but so much service in the noble bondman here, Would her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

**??**

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is ender; and, for your love, I would be , as I must, for my own honour, if he ore, out of my love to you, I came hither to acquaint you withal, that either you might stay him from his intend or brook such disgrace well as he shall run into, in tha is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Which is the real Shakespeare?!

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

# Sampling from an RNN-LM

## Shakespeare's As You Like It

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

## RNN-LM Sample

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is but young and tender; and, for your love, I would be loath to foil him, as I must, for my own honour, if he come in: therefore, out of my love to you, I came hither to acquaint you withal, that either you might stay him from his intendment or brook such disgrace well as he shall run into, in that it is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

# Sampling from an RNN-LM

## RNN-LM Sample

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

## Shakespeare's As You Like It

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is but young and tender; and, for your love, I would be loath to foil him, as I must, for my own honour, if he come in: therefore, out of my love to you, I came hither to acquaint you withal, that either you might stay him from his intendment or brook such disgrace well as he shall run into, in that it is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

# Sampling from an RNN-LM

**??**

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy m… there My power to give thee but so much … service in the noble bondman here, Would… her wine.

KING LEAR: O, if you were a feeble s…, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

**??**

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is …ender; and, for your love, I would be …, as I must, for my own honour, if he …ore, out of my love to you, I came hither … withal, that either you might stay him from his intend… …r brook such disgrace well as he shall run into, in tha… is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

**Which is the real Shakespeare?!**

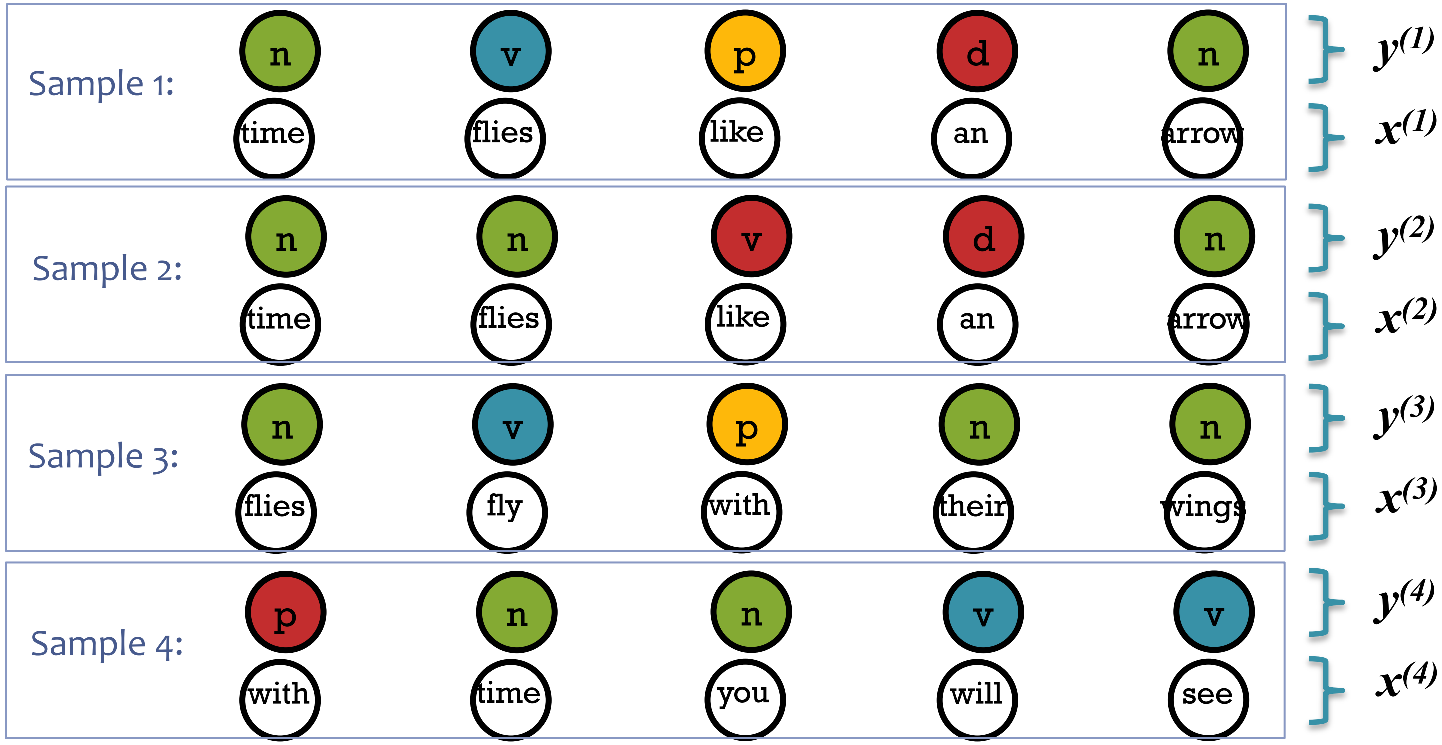
Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

# LEARNING AN RNN

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

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** $\theta^{(s)}$

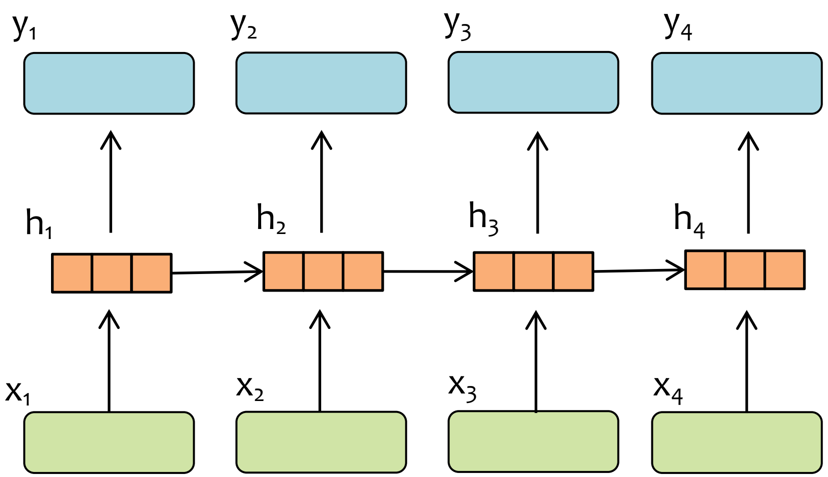
# SGD and Mini-batch SGD

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




**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:

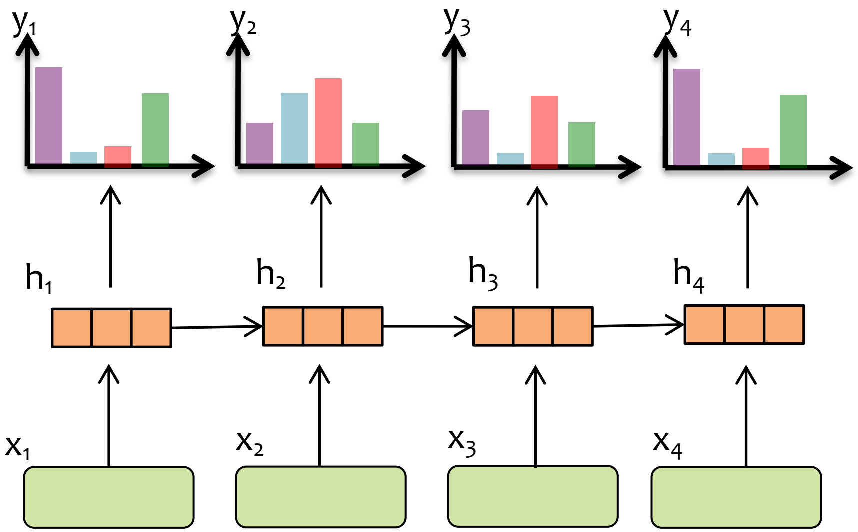
$$a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$$

$$h_t = \sigma(a_t)$$

Compute the output at time step $t$:

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

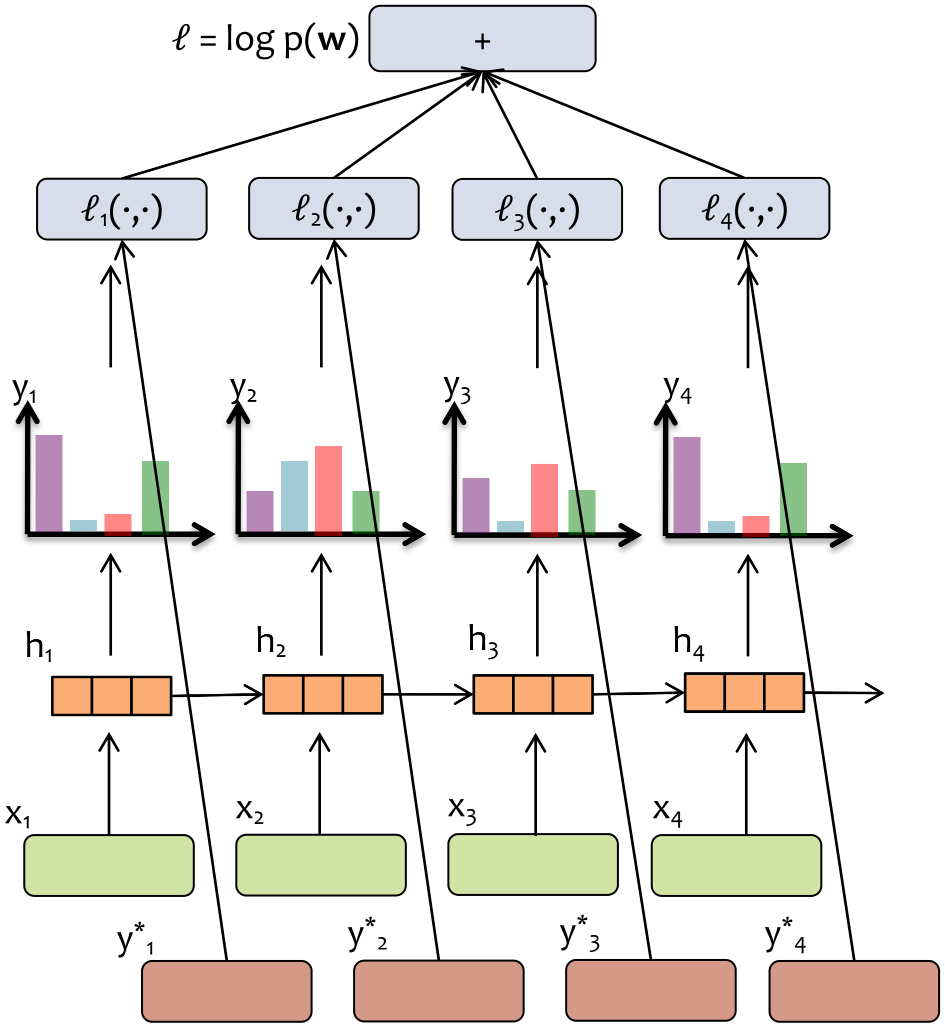
# RNN




**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 = softmax(W_yh · 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

*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

# Learning a Language Model

**MLE for Deep Neural LM**

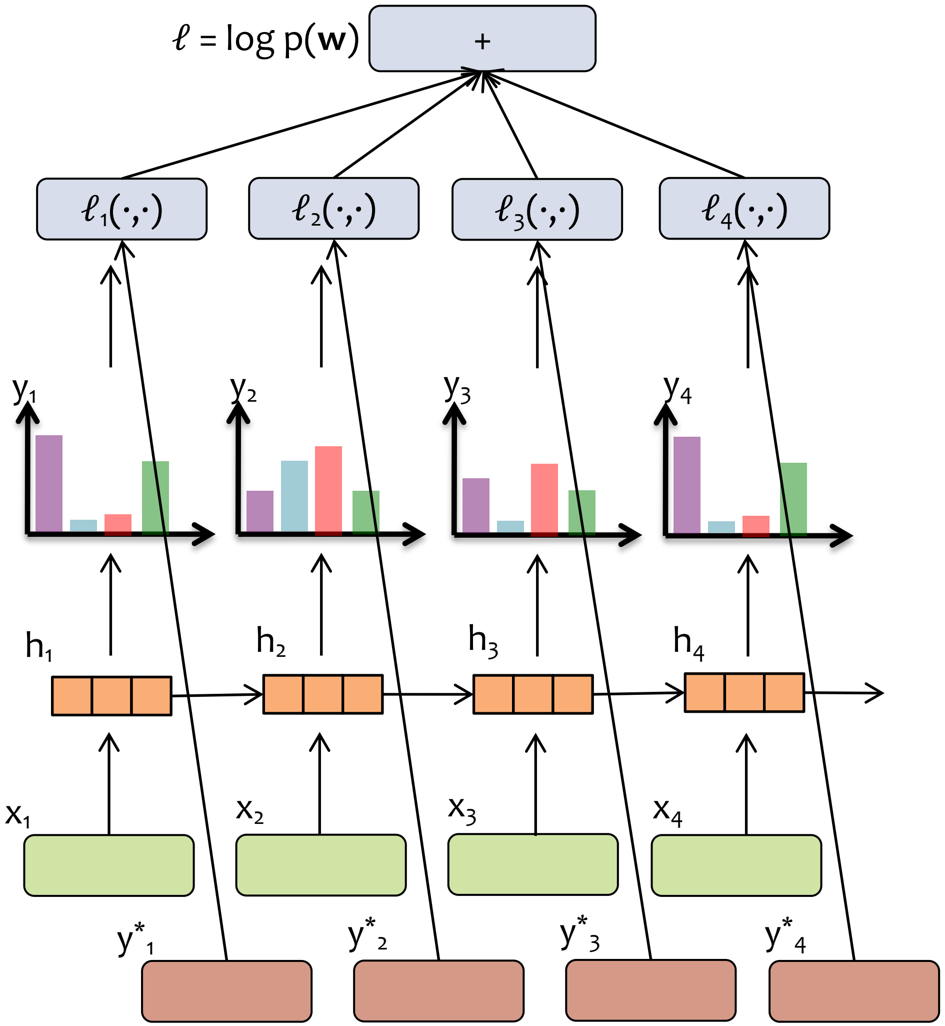
- We can also use maximum likelihood estimation to learn the parameters of an RNN-LM or Transformer-LM too!
- But **not in closed form** – instead we follow a different recipe:
  - Write the **likelihood** of the sentences under the Deep Neural LM model
  - Compute the **gradient** of the (batch) likelihood 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 $t$:

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

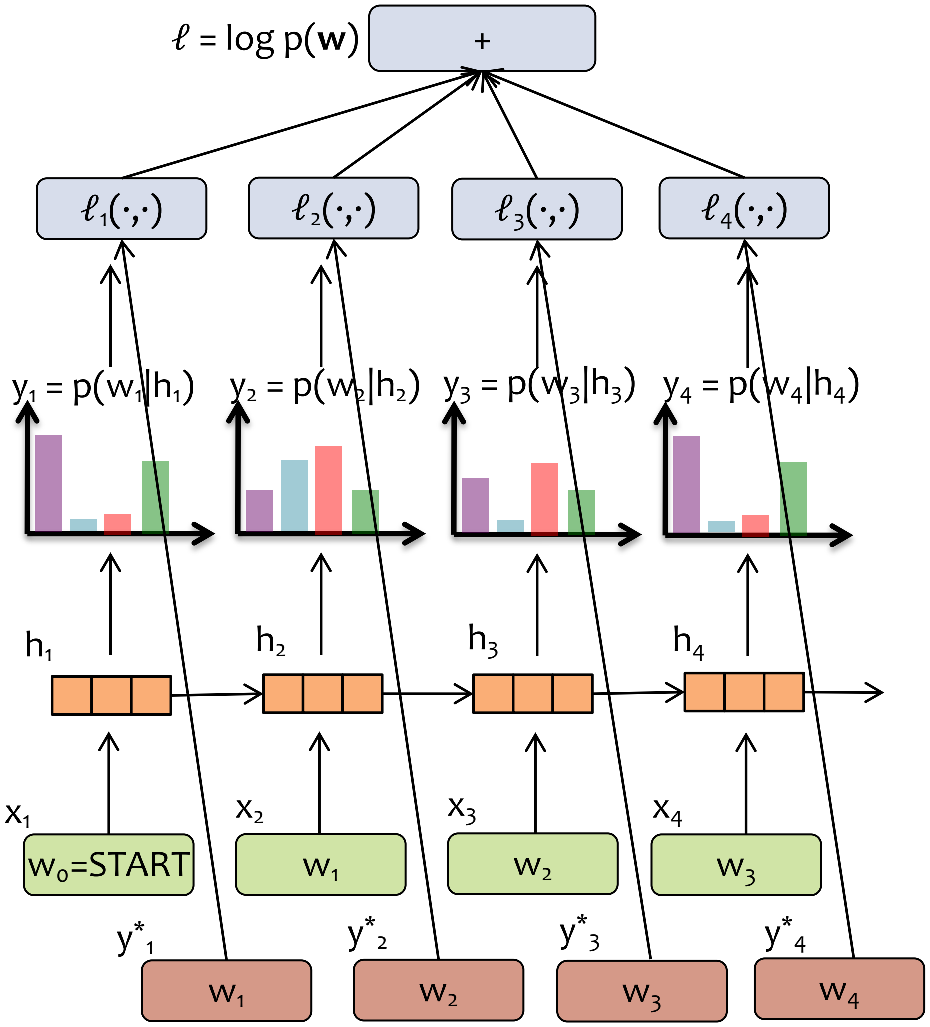
  Compute the total loss:

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

# RNN-LM + Loss

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

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



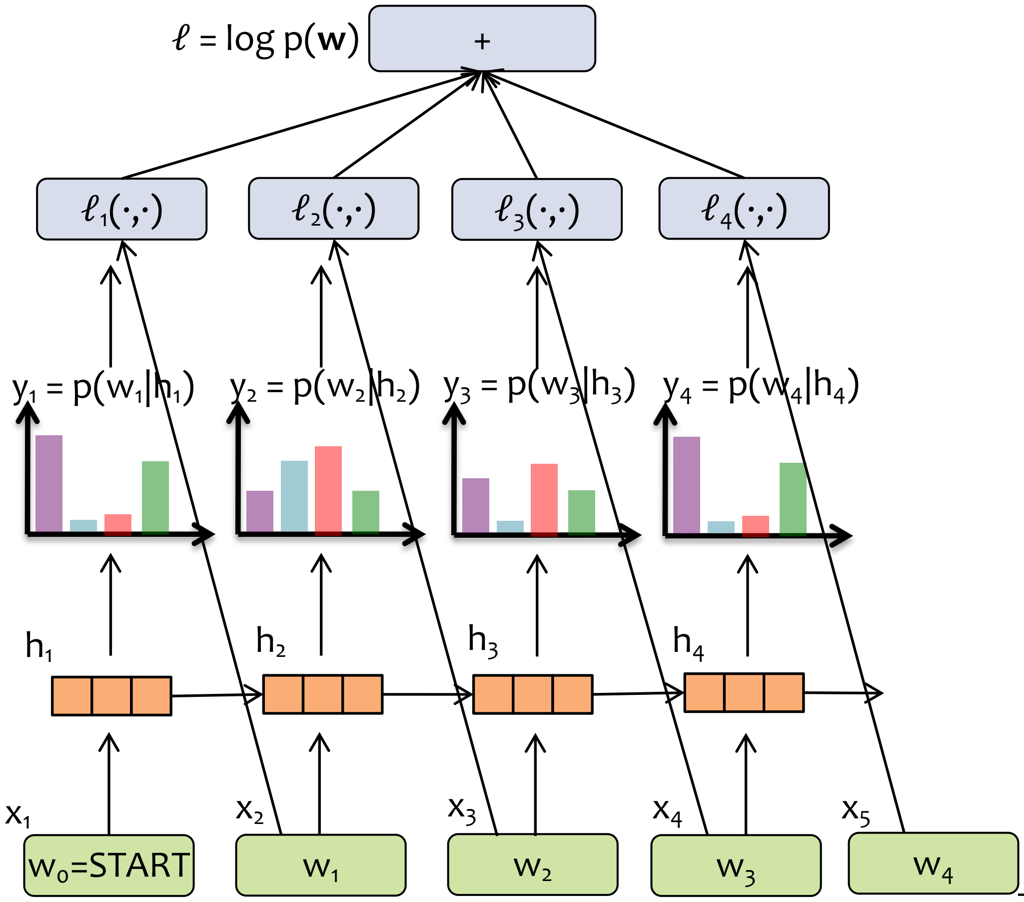

**Algorithm 1** Elman RNN + Loss

**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

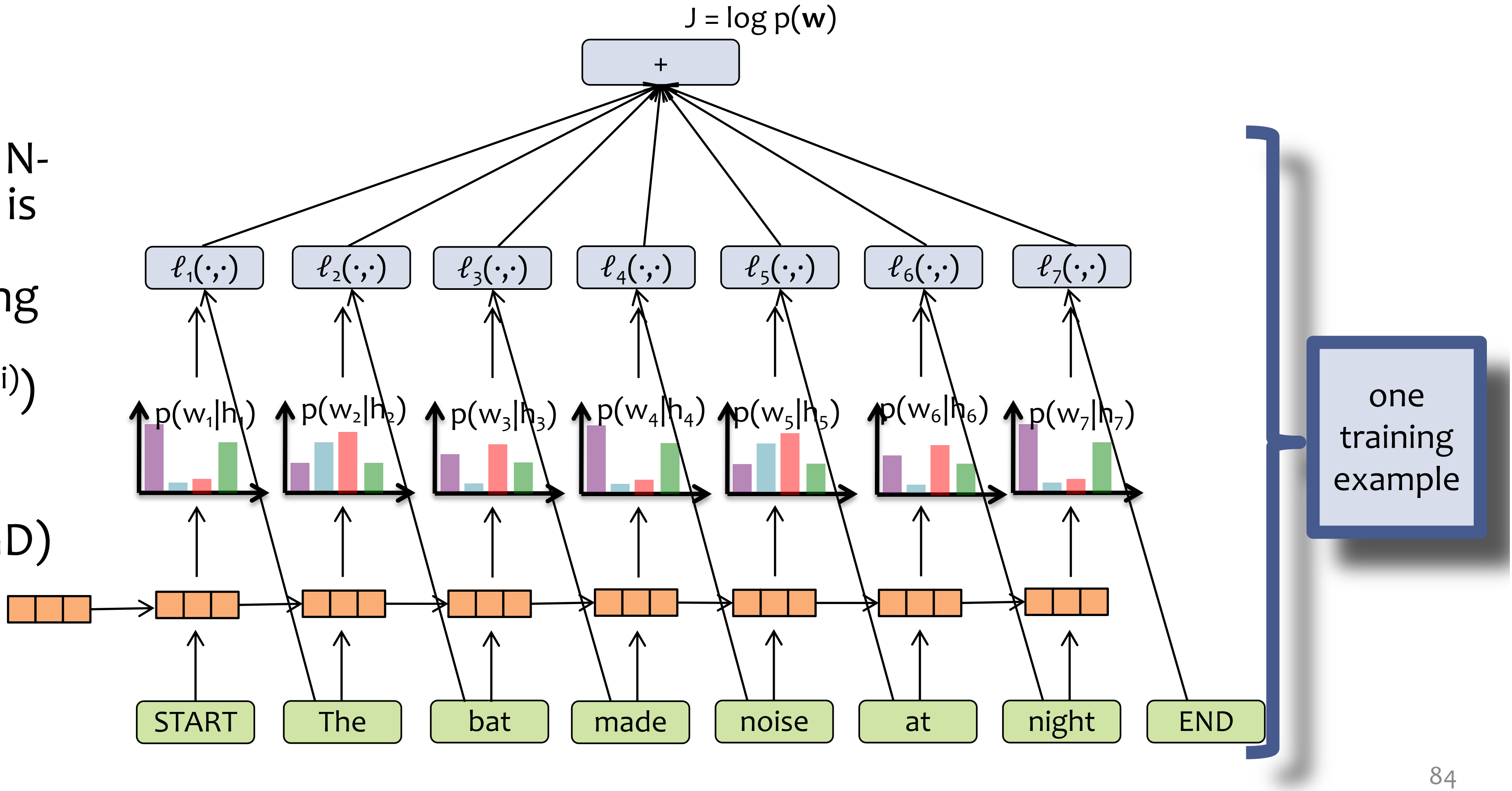
```
procedure FORWARD(x_{1:T}, y*_{1:T} W_ah, W_ax, b_a, W_yh, b_y)
    Initialize the hidden state h_0 to zeros
    for t in 1 to T do
        Receive input data at time step t: x_t
        Compute the hidden state update:
            a_t = W_ah · h_{t-1} + W_ax · x_t + b_a
            h_t = σ(a_t)
        Compute the output at time step t:
            y_t = softmax(W_yh · h_t + b_y)
        Compute the cross-entropy loss at time step t:
            ℓ_t = - Σ_{k=1}^{K} (y*_t)_k log((y_t)_k)
    Compute the total loss:
    ℓ = Σ_{t=1}^{T} ℓ_t
```

$$a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$$
$$h_t = \sigma(a_t)$$
$$y_t = \text{softmax}(W_{yh} \cdot h_t + b_y)$$
$$\ell_t = -\sum_{k=1}^{K} (y_t^*)_k \log((y_t)_k)$$
$$\ell = \sum_{t=1}^{T} \ell_t$$

# 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)

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



# LARGE LANGUAGE MODELS

# How large are LLMs?

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 |

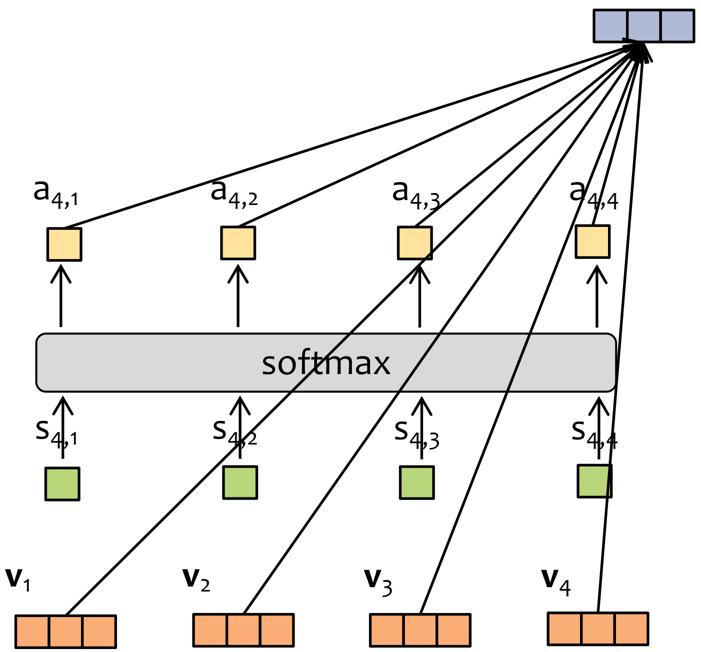
# What is ChatGPT?

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

The key building block for Transformer language models

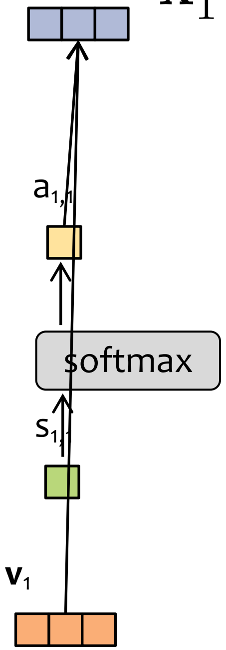
# ATTENTION

# Attention



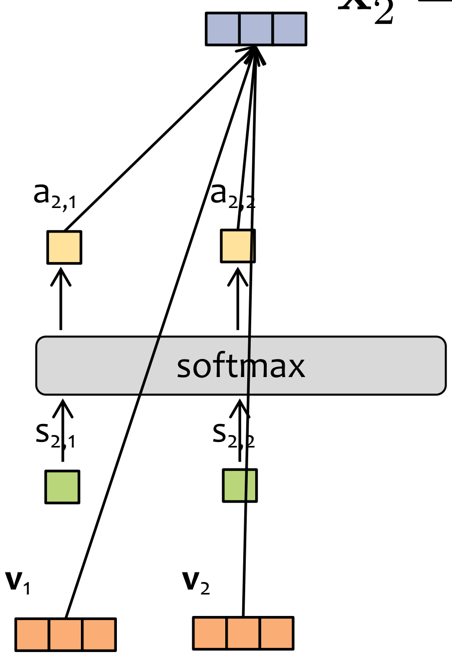

$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j} \mathbf{v}_j$$

# Attention



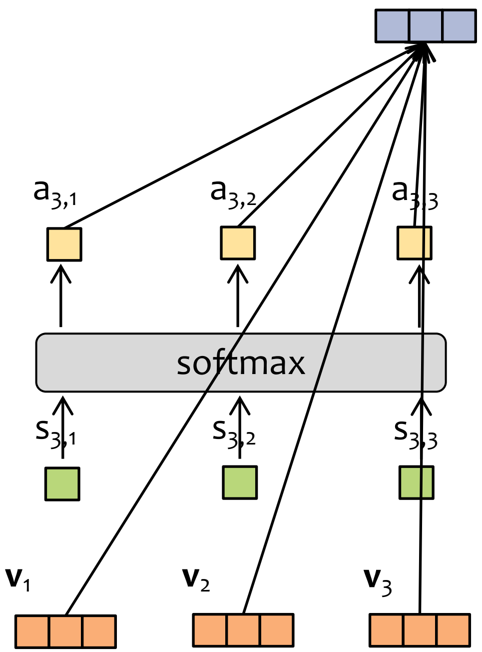

$$\mathbf{x}'_1 = \sum_{j=1}^{1} a_{1,j}\mathbf{v}_j$$

# Attention




$$\mathbf{x}'_2 = \sum_{j=1}^{2} a_{2,j}\mathbf{v}_j$$

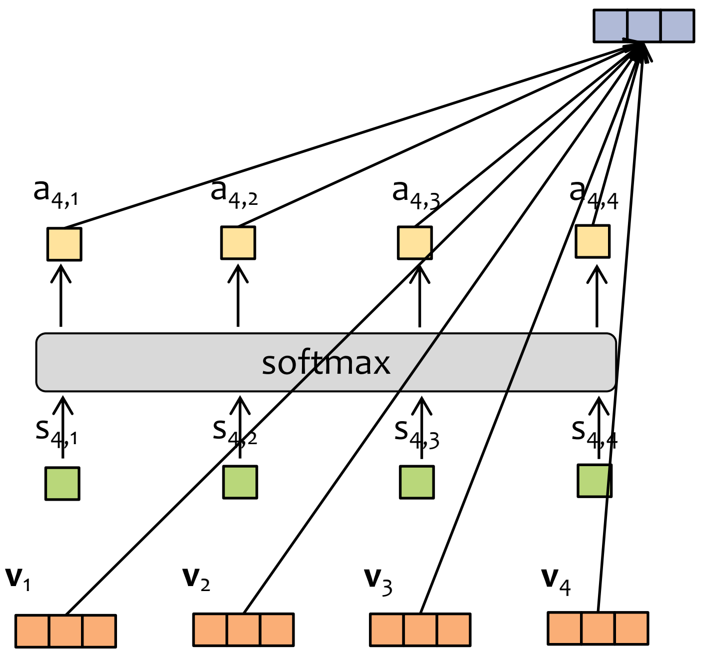
# Attention



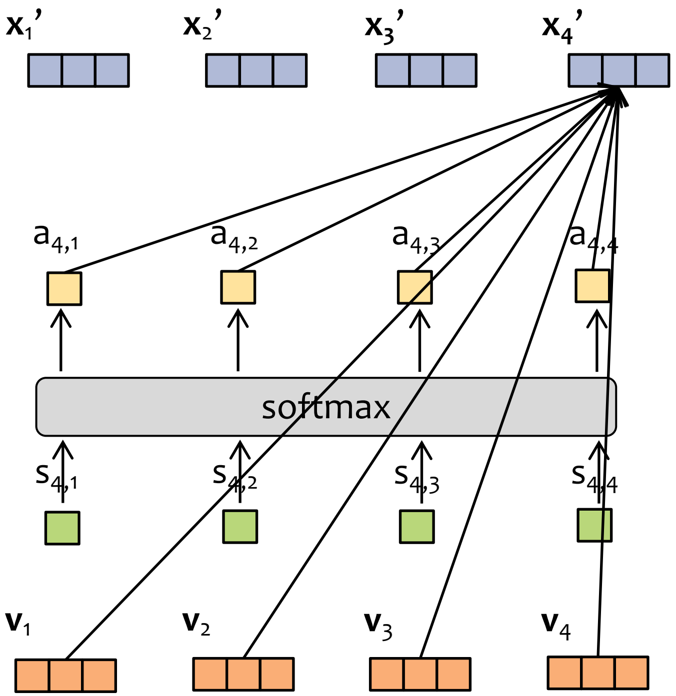

$$\mathbf{x}'_3 = \sum_{j=1}^{3} a_{3,j}\mathbf{v}_j$$

# Attention

$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$



# Attention




$$\mathbf{x}'_t = \sum_{j=1}^{t} a_{t,j} \mathbf{v}_j$$

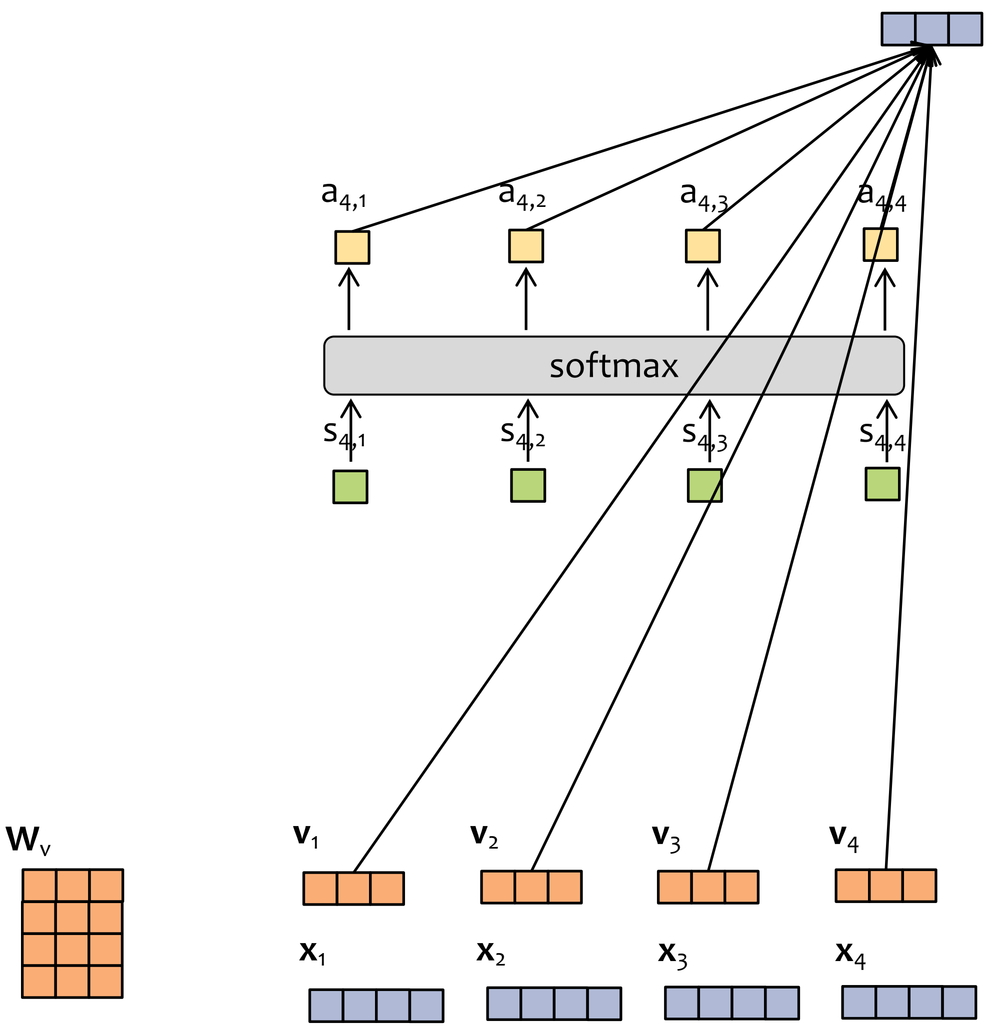
attention weights

scores

values

# Scaled Dot-Product Attention

$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$

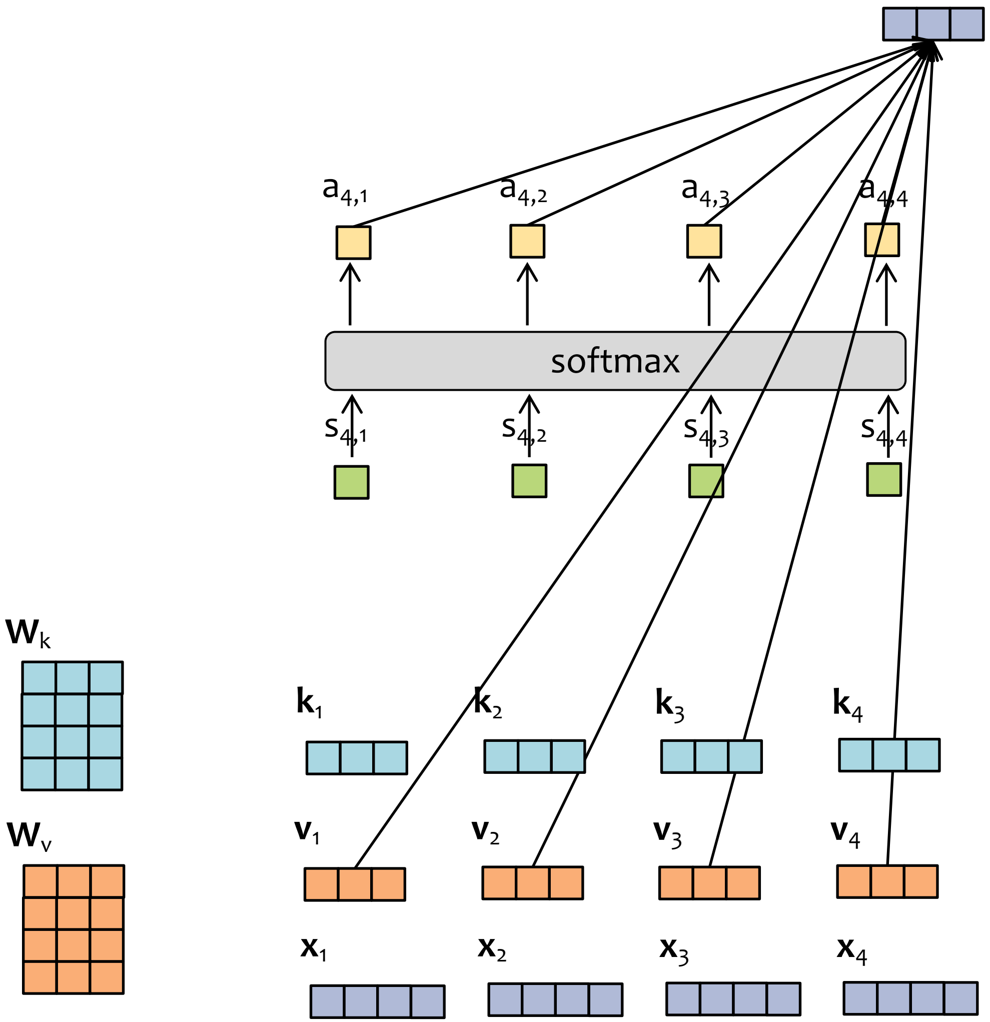



$$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$$

values

# Scaled Dot-Product Attention

$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$

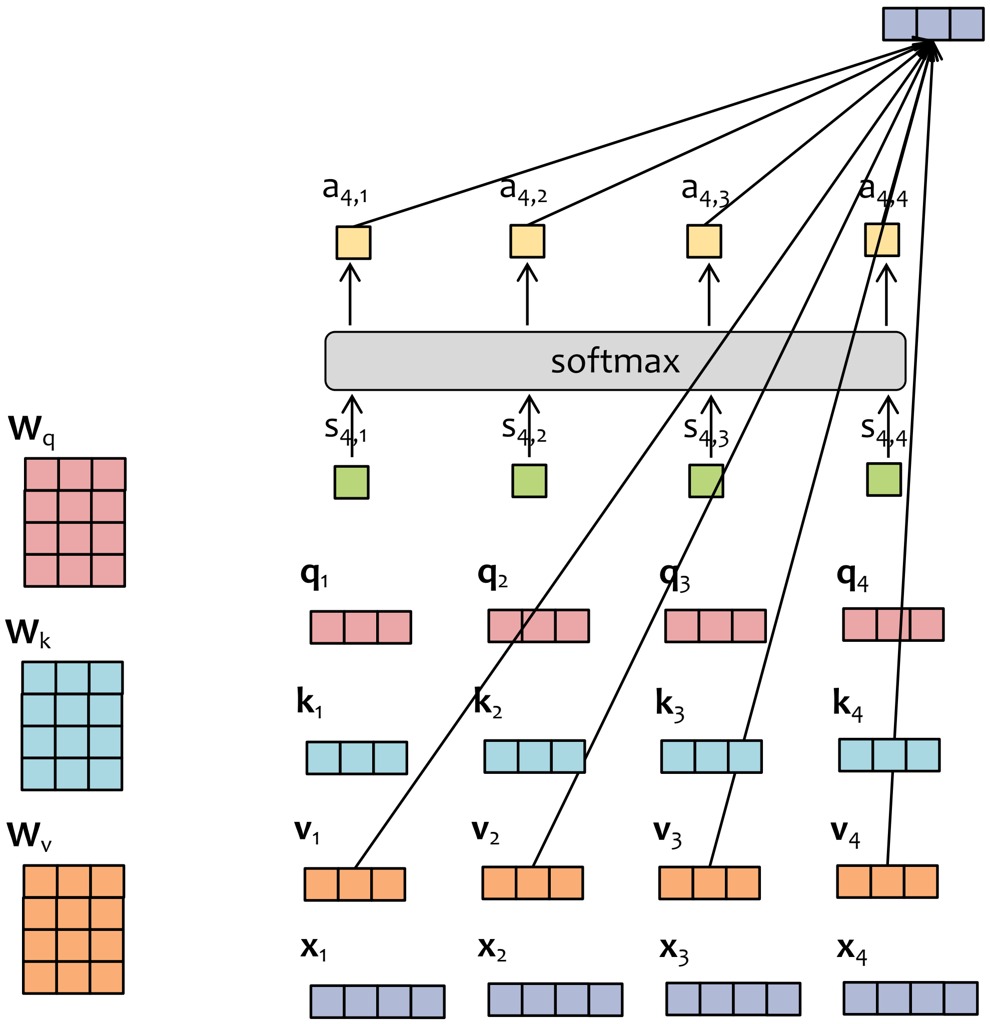



$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$$ keys

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

# Scaled Dot-Product Attention

$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$

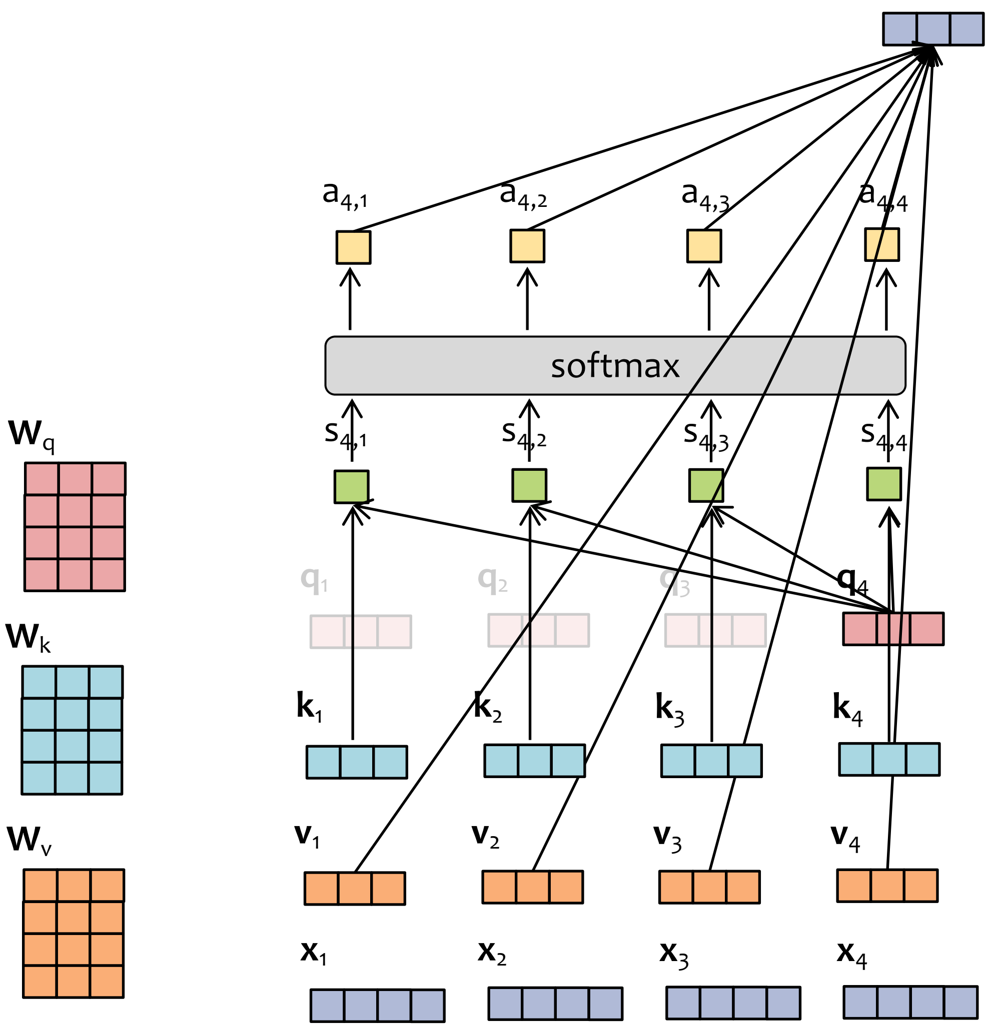



$$\mathbf{q}_j = \mathbf{W}_q^T\mathbf{x}_j$$ queries

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

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

# Scaled Dot-Product Attention




$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$

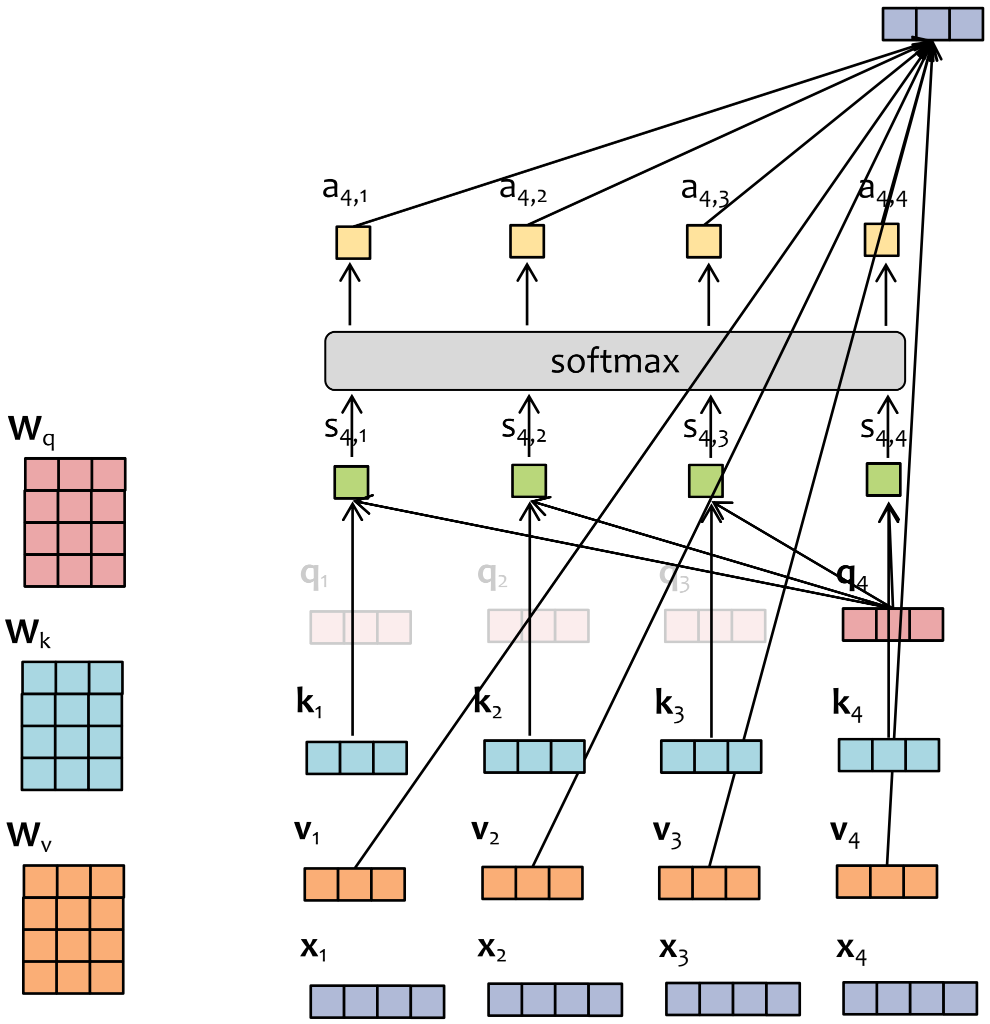
$$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$$ scores

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




$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$

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

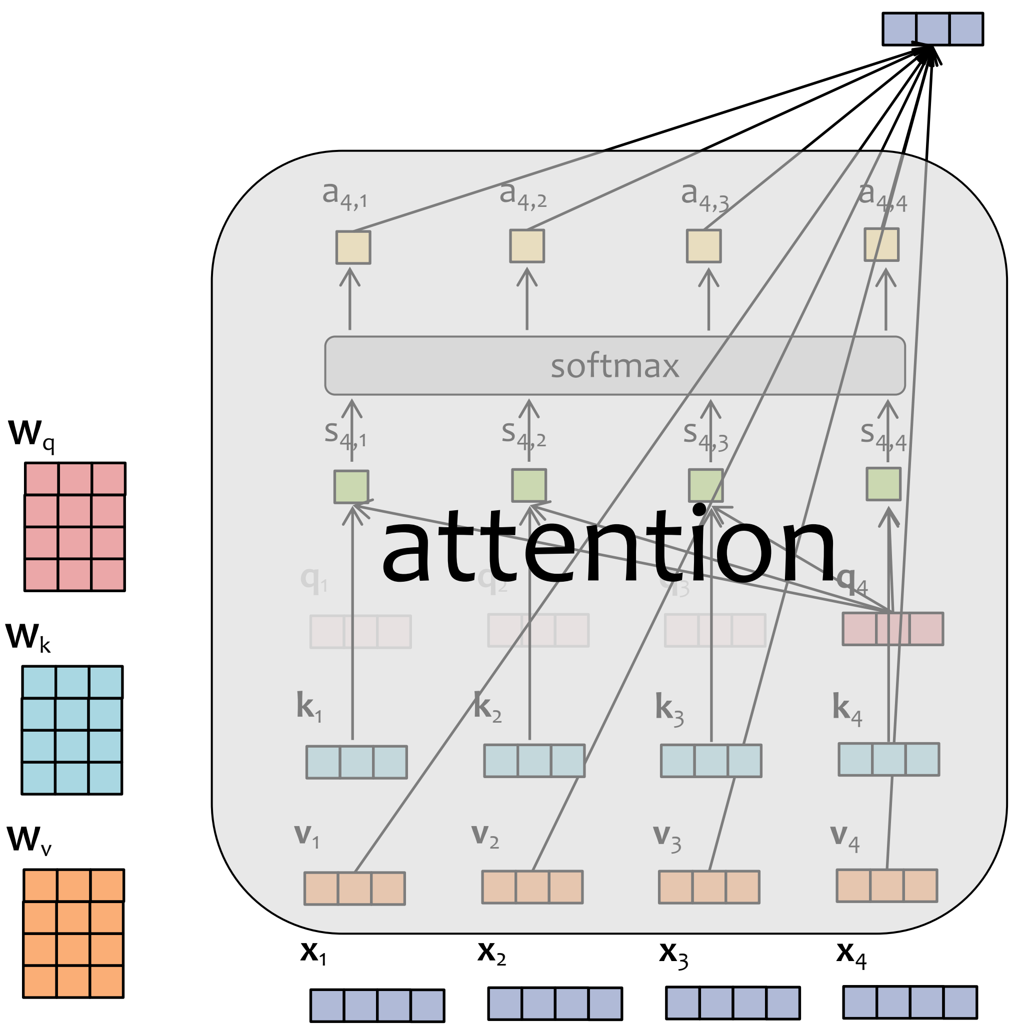
$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

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




$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j} \mathbf{v}_j$$

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

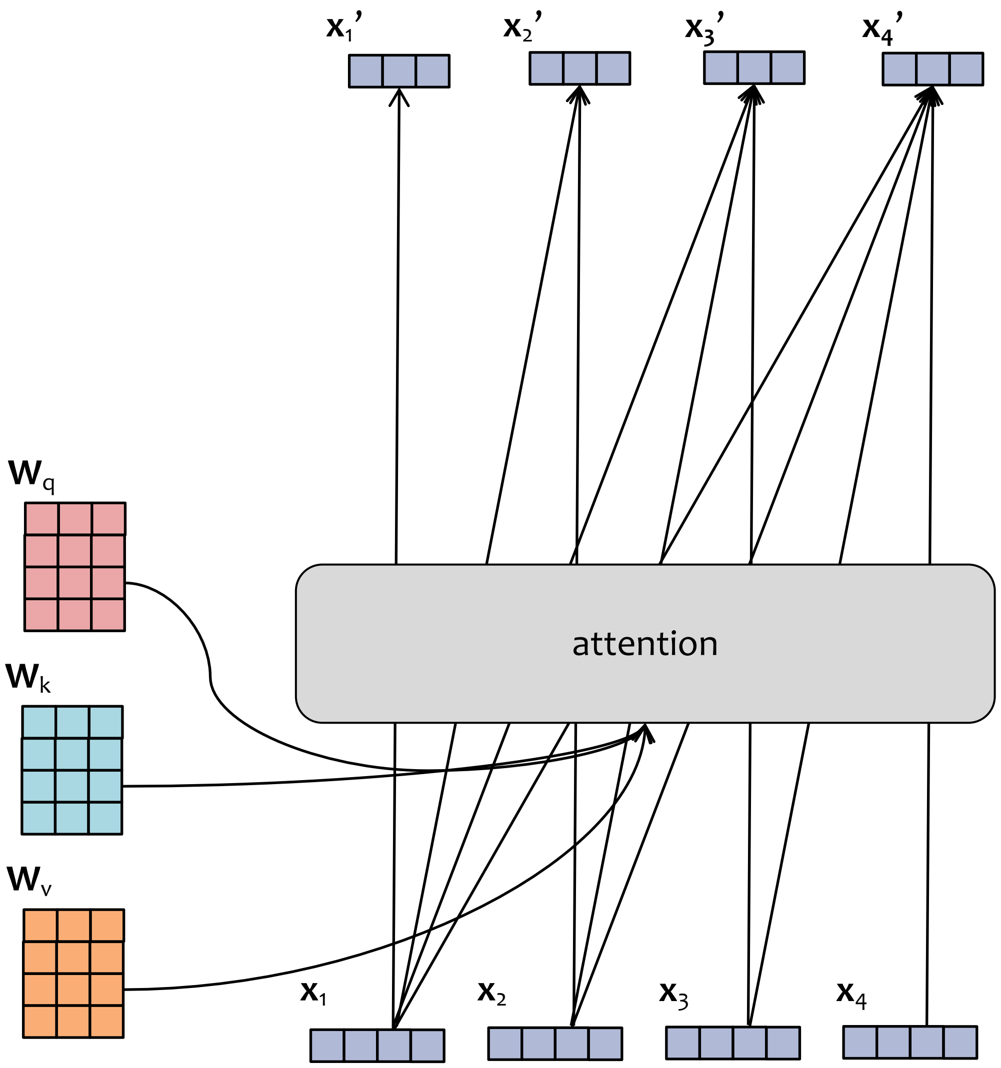
$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

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




$$\mathbf{x}'_t = \sum_{j=1}^{t} a_{t,j}\mathbf{v}_j$$

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

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

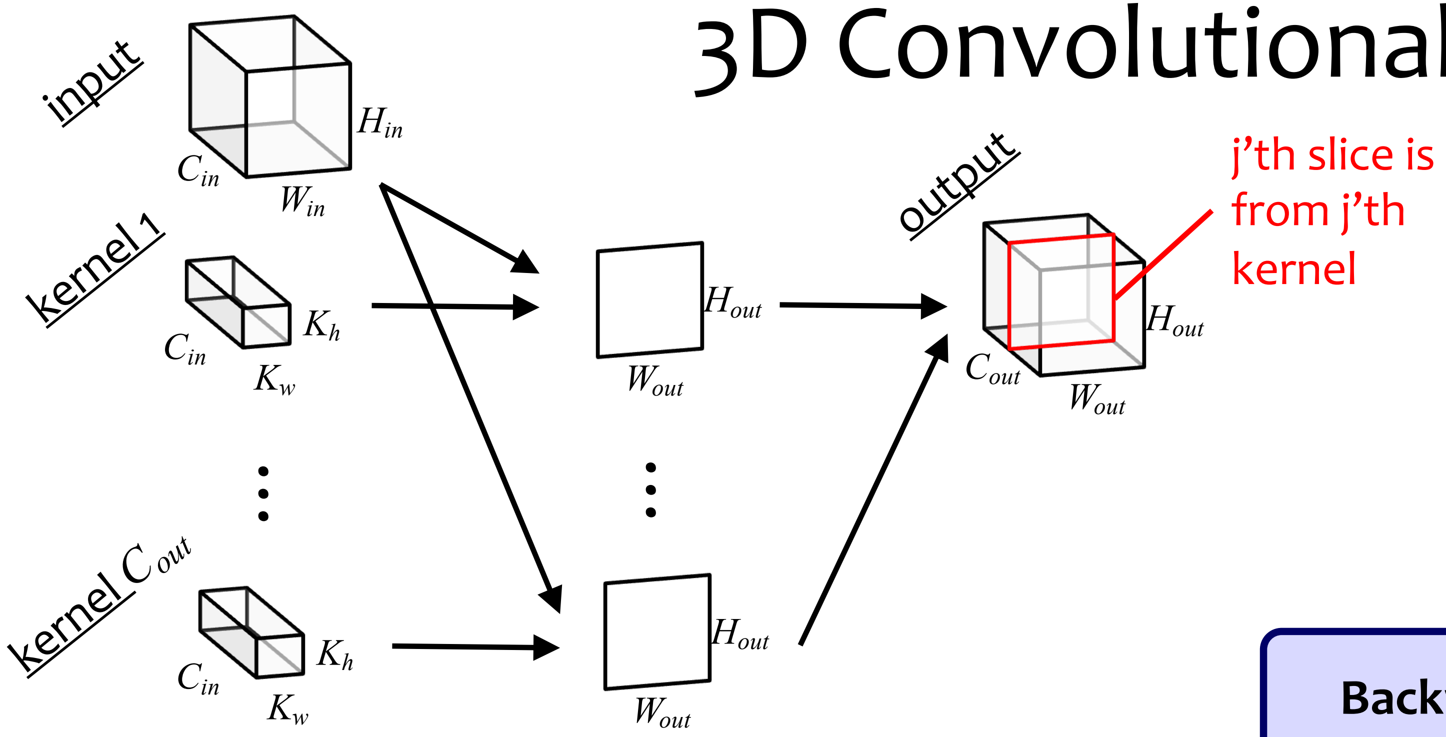
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

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

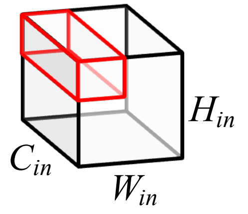
$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# 3D Convolutional Layer

Recall...




Convolution in 3D




**Forward:**

$$y^{(c')}_{h',w'} = \beta^{(c')} + \sum_{c=1}^{C_{\text{in}}} \sum_{m=1}^{K_{\text{h}}} \sum_{n=1}^{K_{\text{w}}} x^{(c)}_{h'+ms,w'+ns} \cdot \alpha^{(c',c)}_{m,n}$$

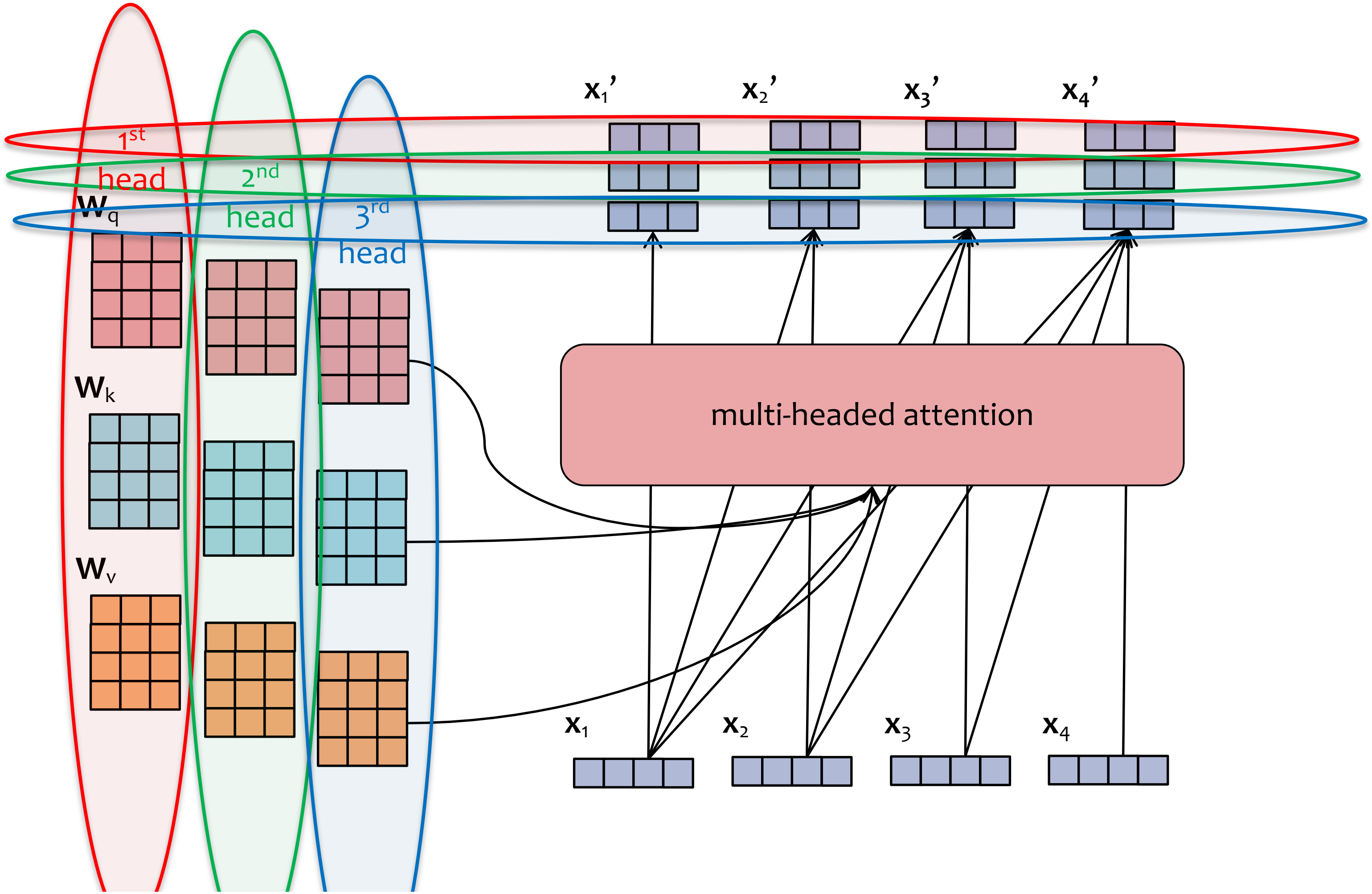
**Backward:**

$$\frac{\partial J}{\partial \alpha^{(c',c)}_{m,n}} = \sum_{h'=1}^{H_{\text{out}}} \sum_{w'=1}^{W_{\text{out}}} \frac{\partial J}{\partial y^{(c')}_{h',w'}} \cdot x^{(c)}_{h'+ms,w'+ns}$$

$$\frac{\partial J}{\partial \beta^{(c')}} = \sum_{h'=1}^{H_{\text{out}}} \sum_{w'=1}^{W_{\text{out}}} \frac{\partial J}{\partial y^{(c')}_{h',w'}}$$

$s \in \mathbb{Z}$ (stride)

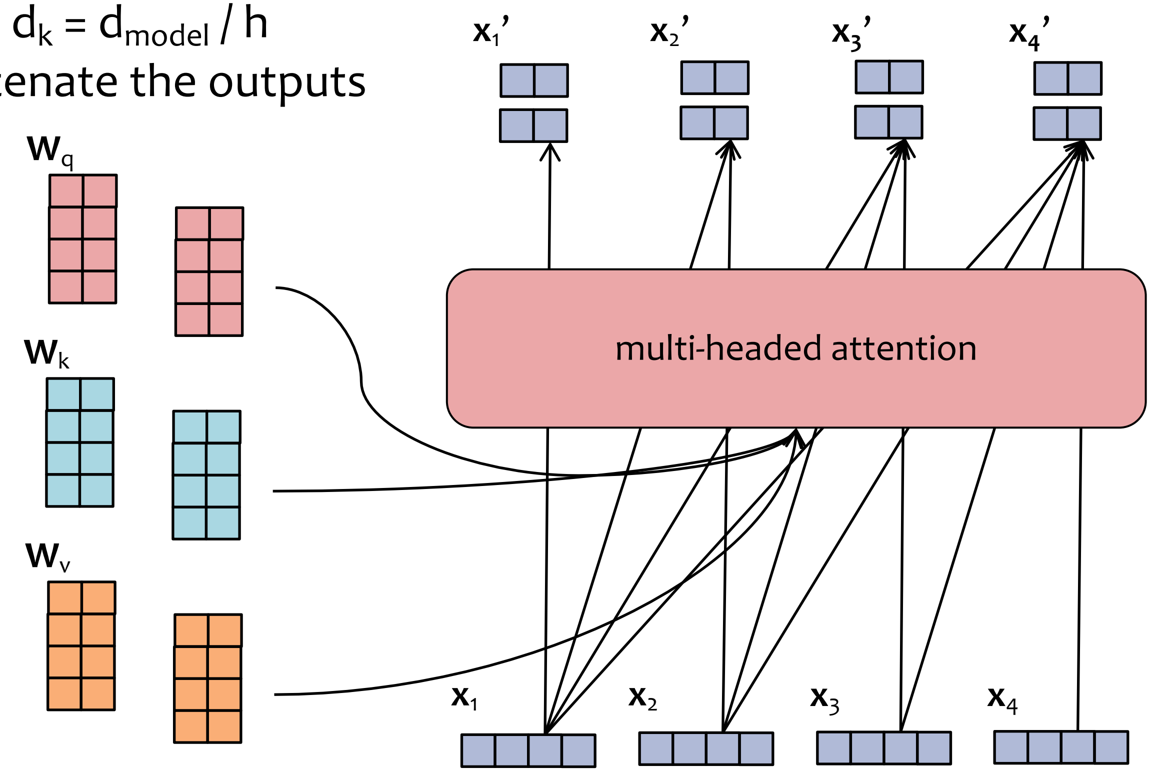
# Multi-headed Attention




- 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

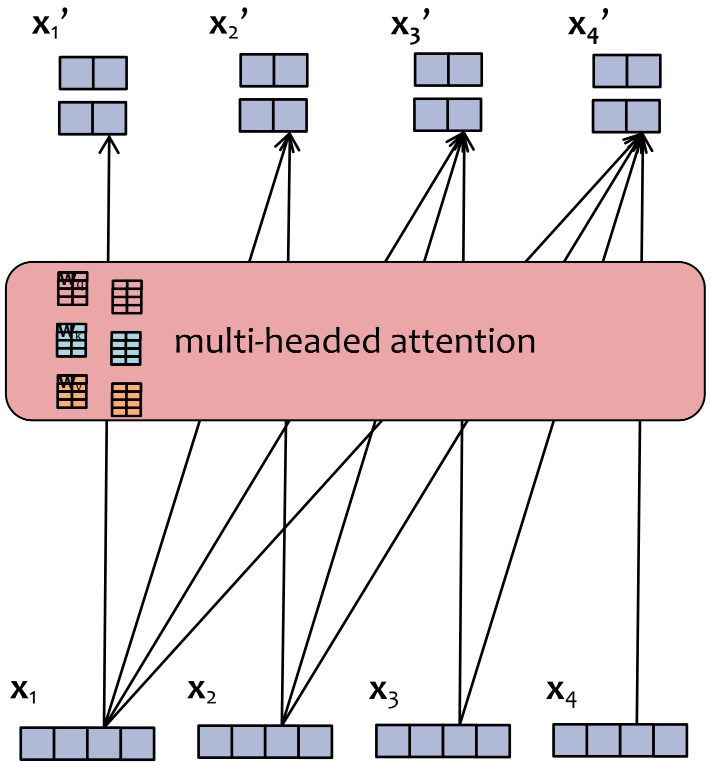
- To ensure the dimension of the **input** embedding $\mathbf{x}_t$ is the same as the **output** embedding $\mathbf{x}_t'$, Transformers usually choose the embedding sizes and number of heads appropriately:
  - $d_{model}$ = dim. of inputs
  - $d_k$ = dim. of each output
  - $h$ = # of heads
  - Choose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



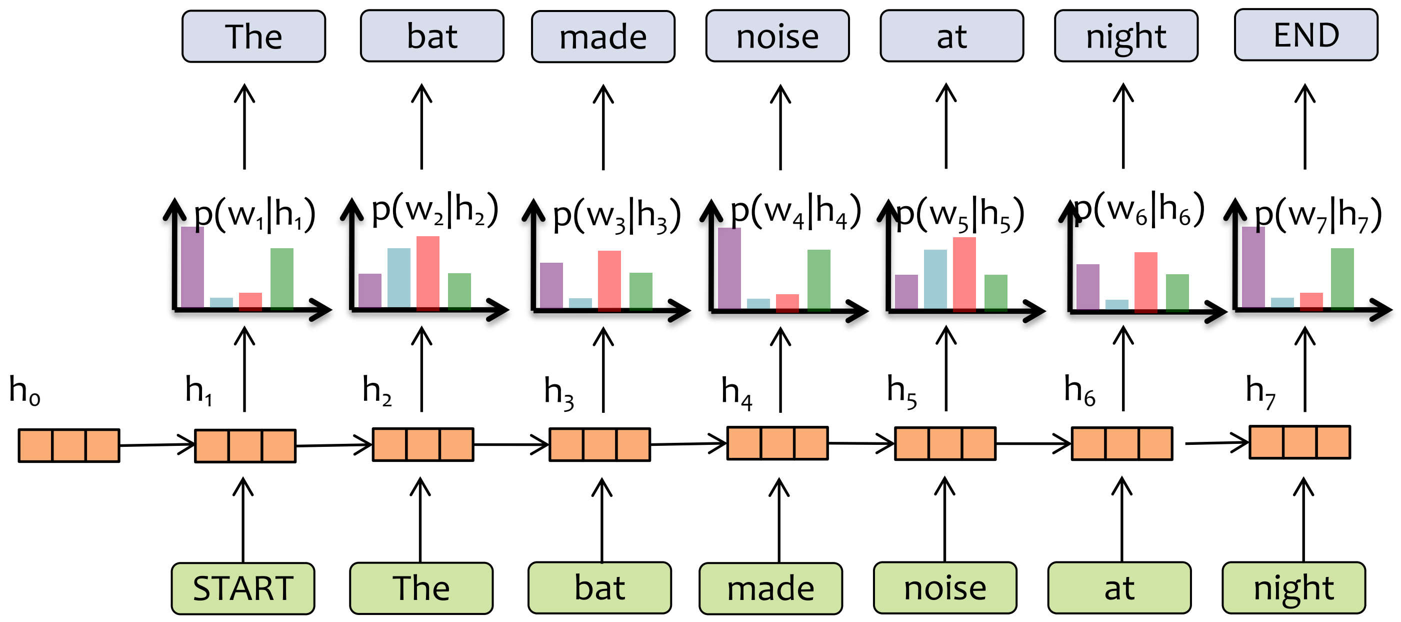

- Just as we can have **multiple channels** in a **convolution** layer, we can use **multiple heads** in an **attention** layer
- Each head gets **its own parameters**
- We can **concatenate** all the outputs to get a single vector for each time step

Generative Pretrained Transformers (GPT)

# TRANSFORMER LANGUAGE MODELS

# RNN Language Model

Recall...

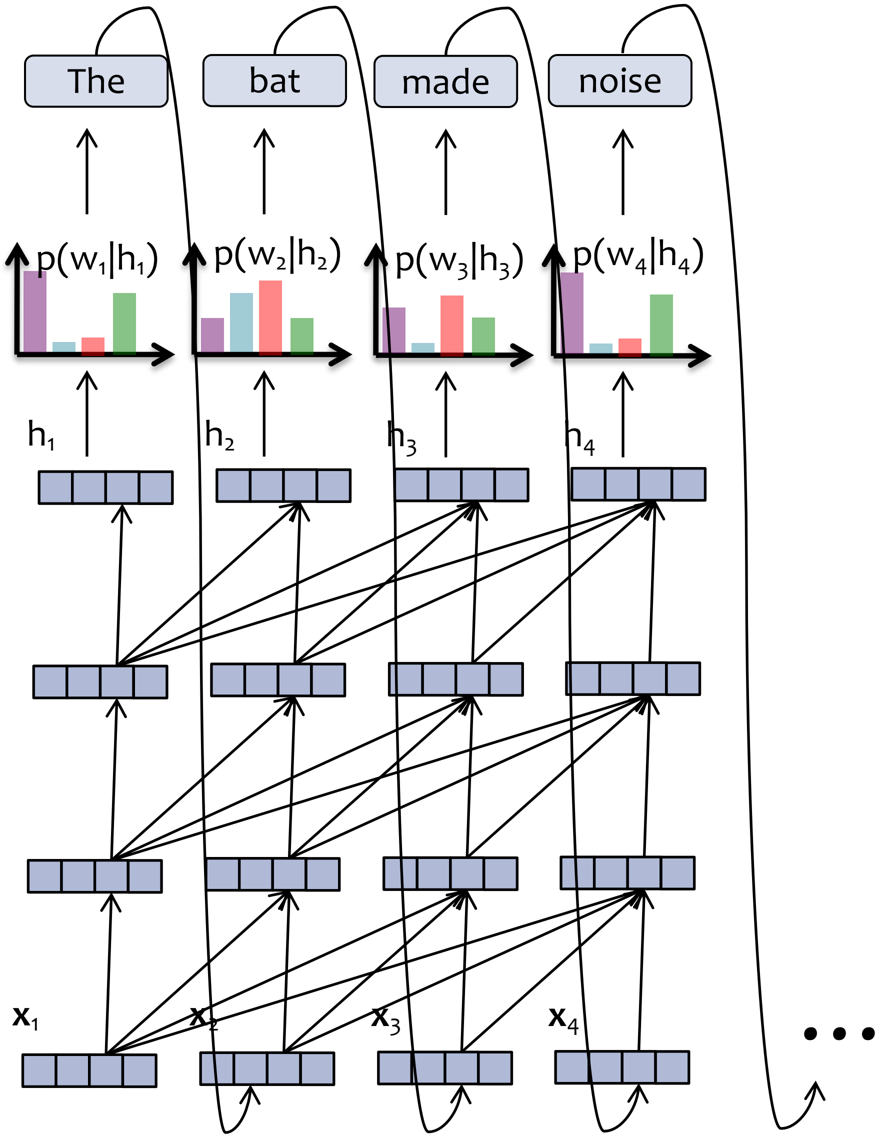



*Key Idea*:

(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

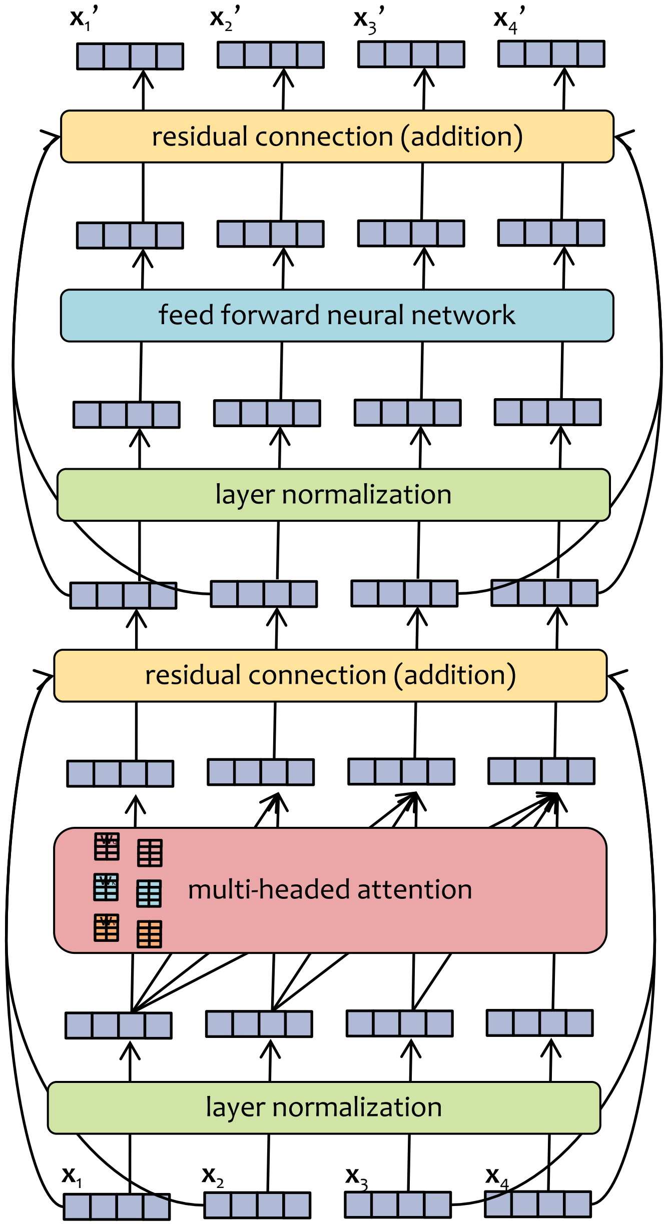



The language model part is just like an RNN-LM!

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

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

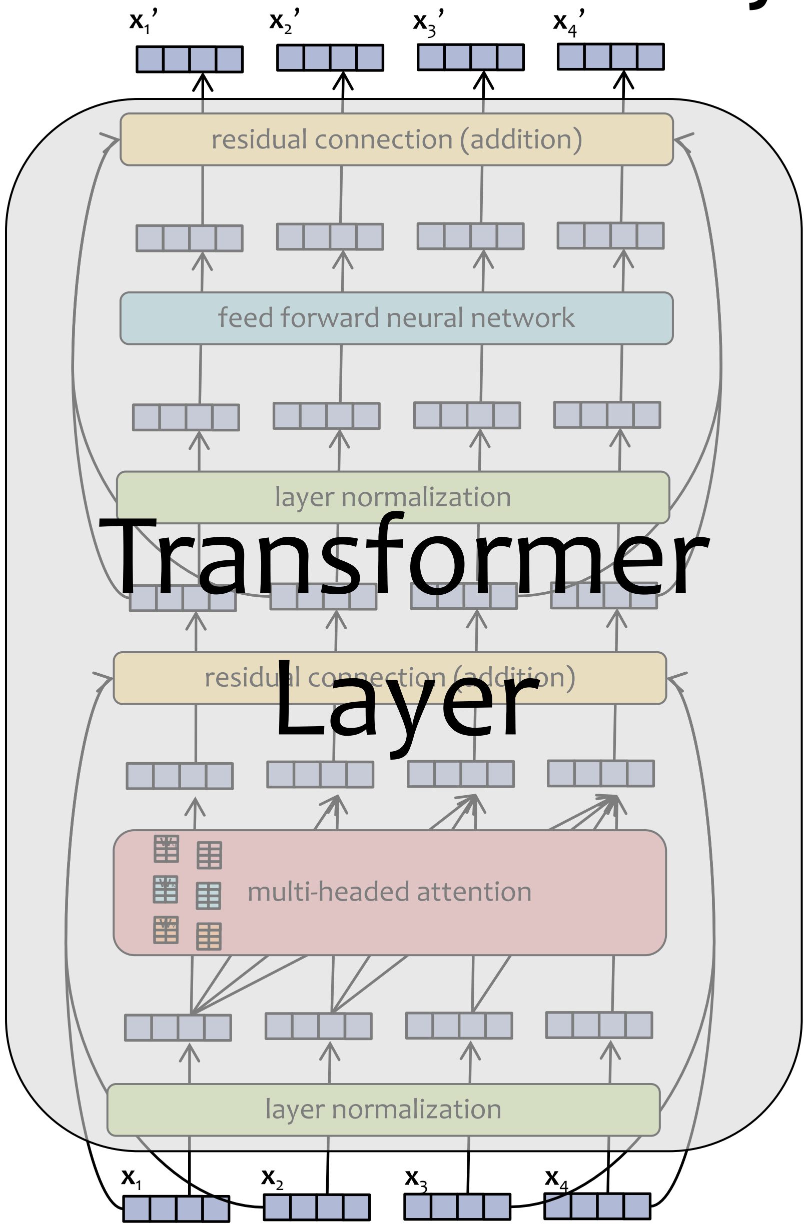
# Transformer Layer




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

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

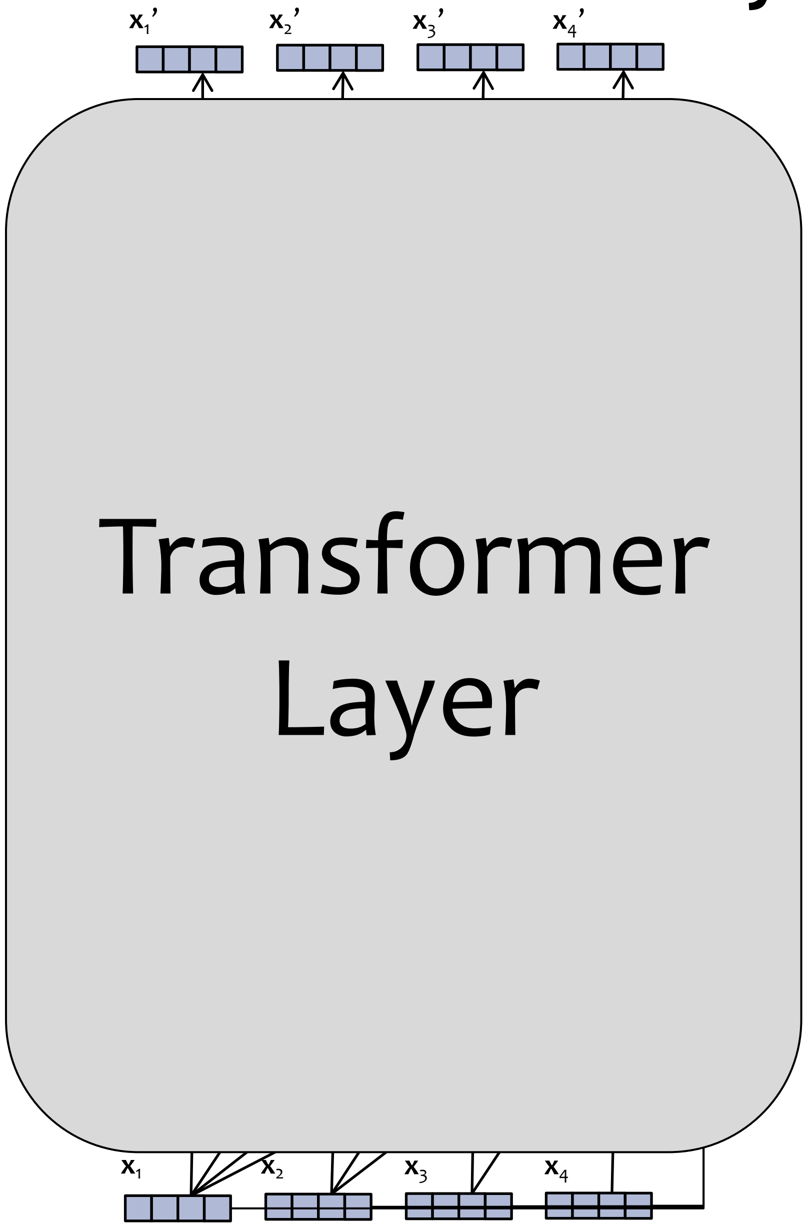
# Transformer Layer




**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



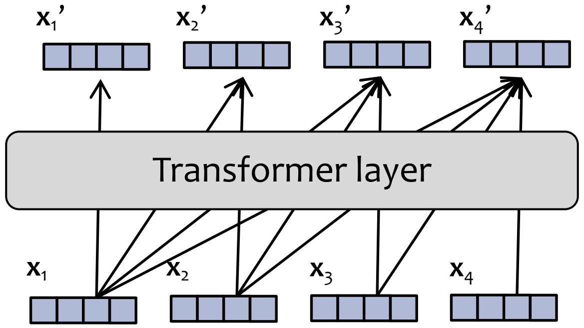

**Each layer** of a Transformer LM consists of several **sublayers:**

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

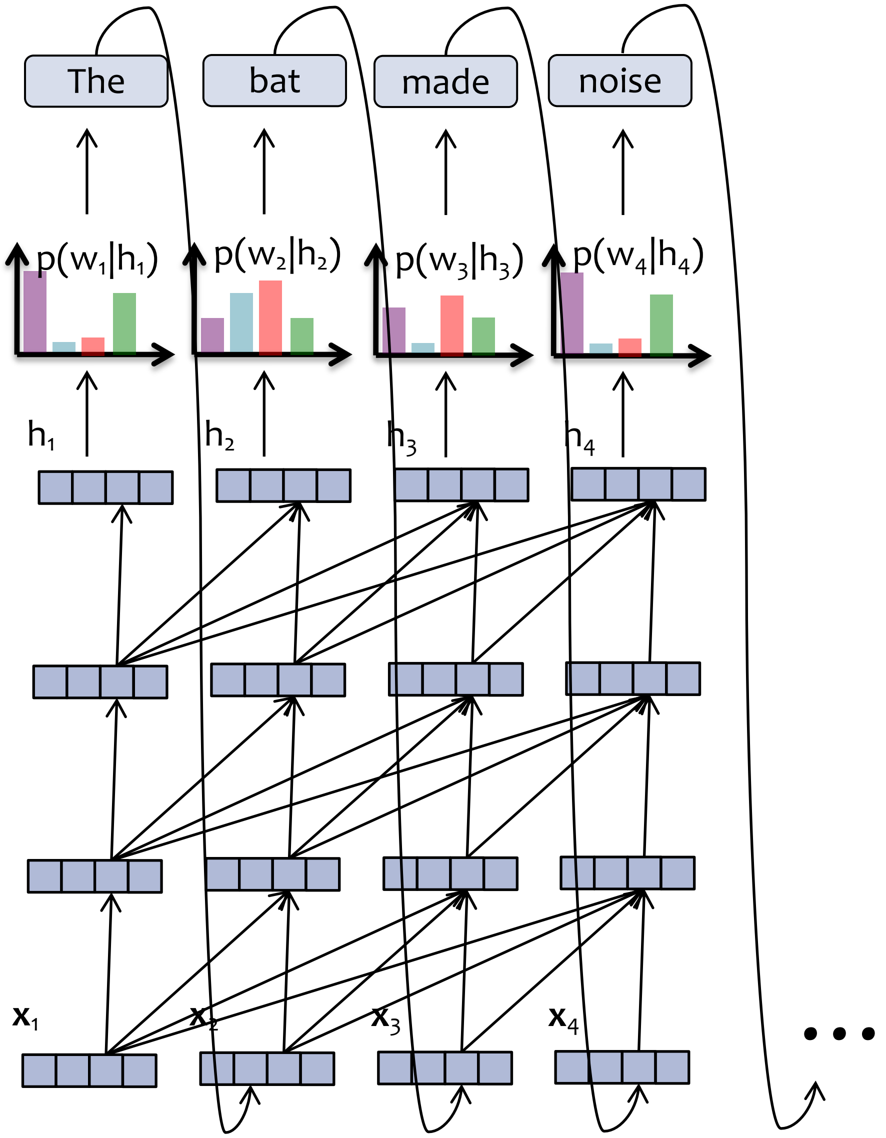
# Transformer Layer

**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




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

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

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

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