



10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

RNN LMs + Transformer LMs

Matt Gormley, Henry Chai

Lecture 18

Oct. 30, 2024

Reminders

- **Homework 6: Learning Theory & Generative Models**
 - Out: Sun, Oct 27
 - Due: Sat, Nov 2, 11:59pm
- **Exam 2: Thu, Nov 7, 6:45 pm - 8:45 pm**

EXAM 2 LOGISTICS

Exam 2

- **Time / Location**
 - **Time: Thu, Nov. 7, 6:45pm – 8:45pm**
 - **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
 - Linear Regression

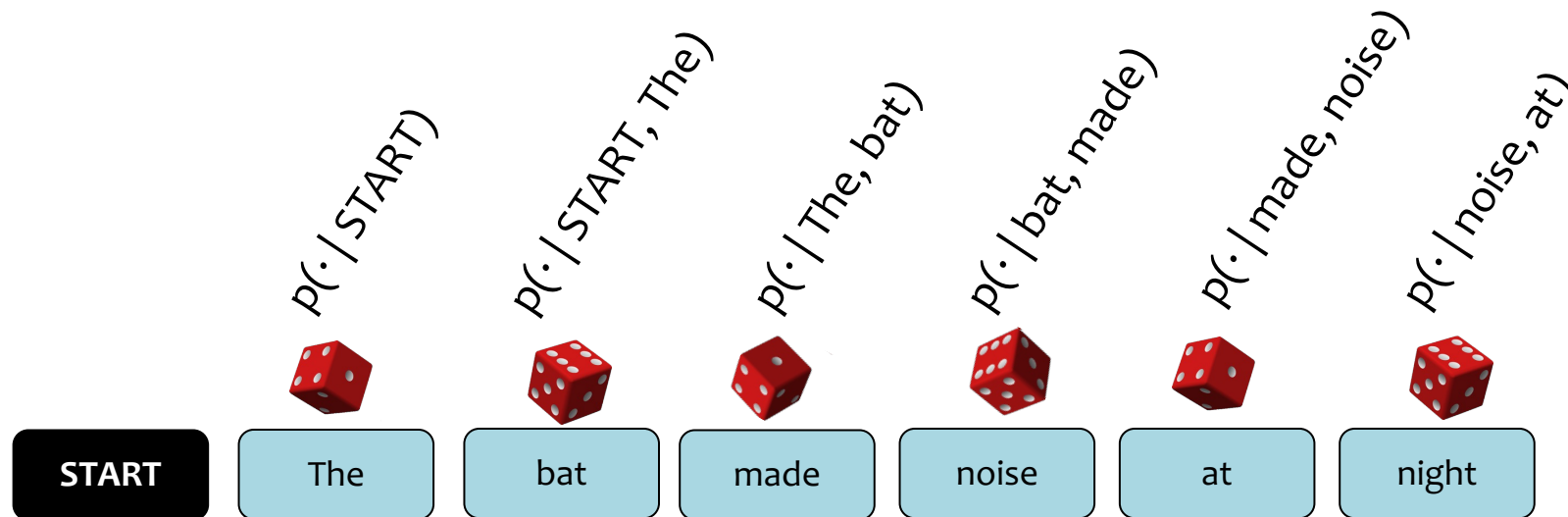
Topics for Exam 2

- 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

BACKGROUND: N-GRAM LANGUAGE MODELS

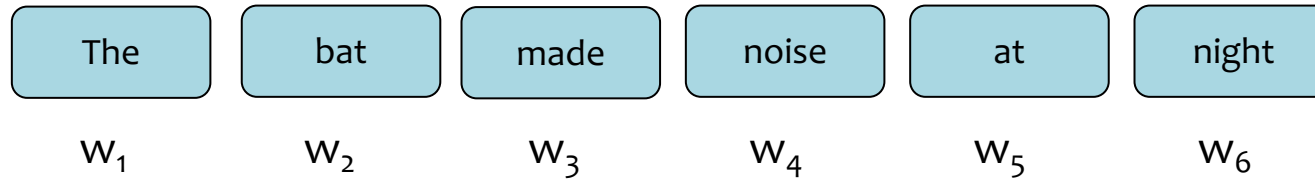
n-Gram Language Model

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



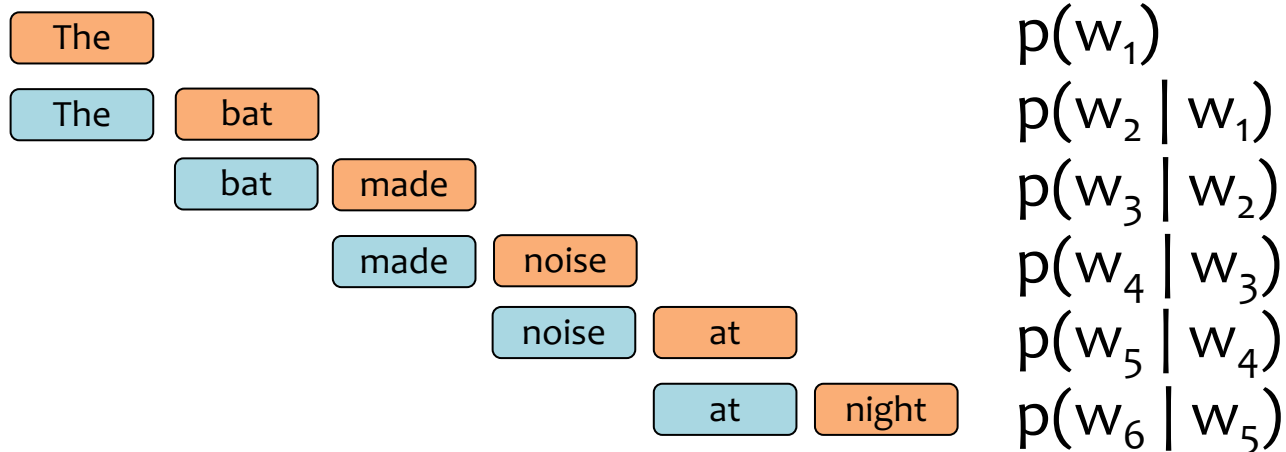
n-Gram Language Model

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



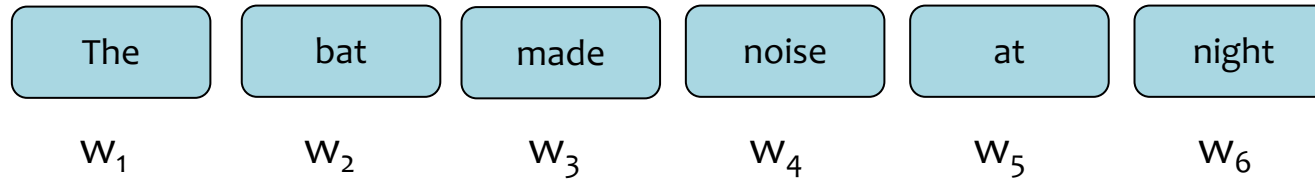
n-Gram Model (n=2)
$$p(w_1, w_2, \dots, w_T) = \prod_{t=1}^T p(w_t | w_{t-1})$$

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



n-Gram Language Model

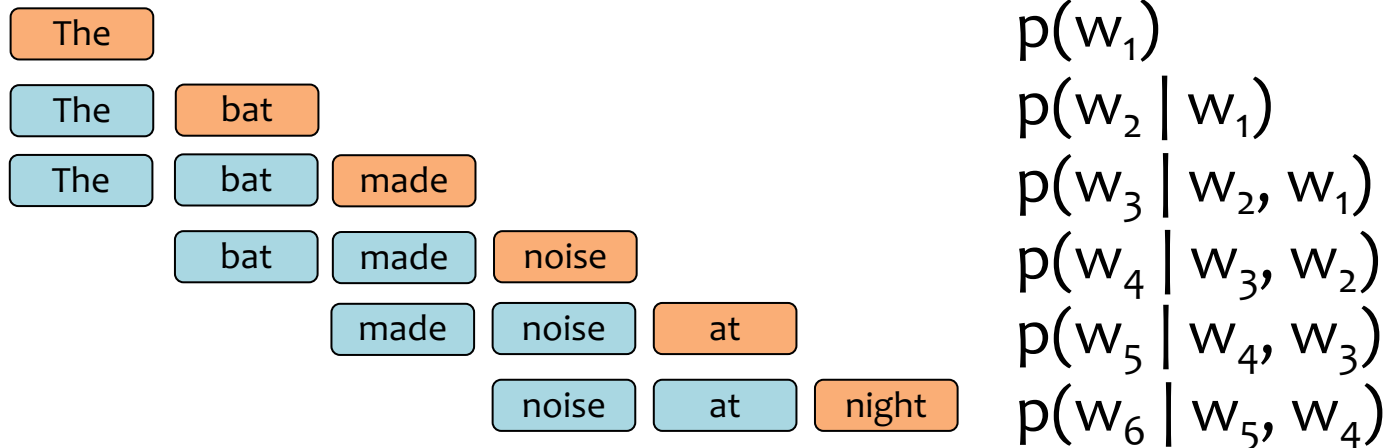
Question: How can we **define** a probability distribution over a sequence of length T?



n-Gram Model (n=3)

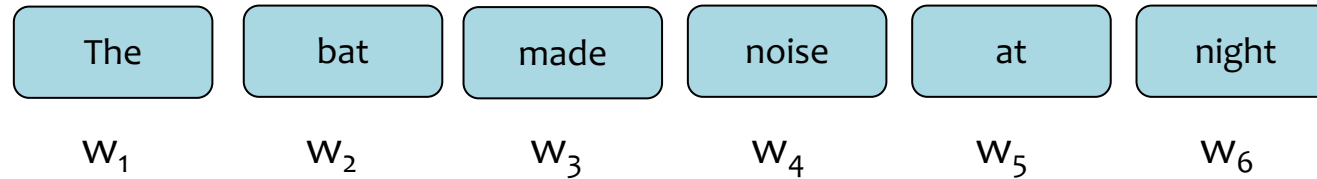
$$p(w_1, w_2, \dots, w_T) = \prod_{t=1}^T p(w_t | w_{t-1}, w_{t-2})$$

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



n-Gram Language Model

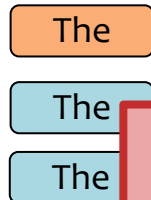
Question: How can we **define** a probability distribution over a sequence of length T?



n-Gram Model (n=3)

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

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




$$p(w_1) \\ p(w_2 | w_1)$$


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


Question: 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**?...

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


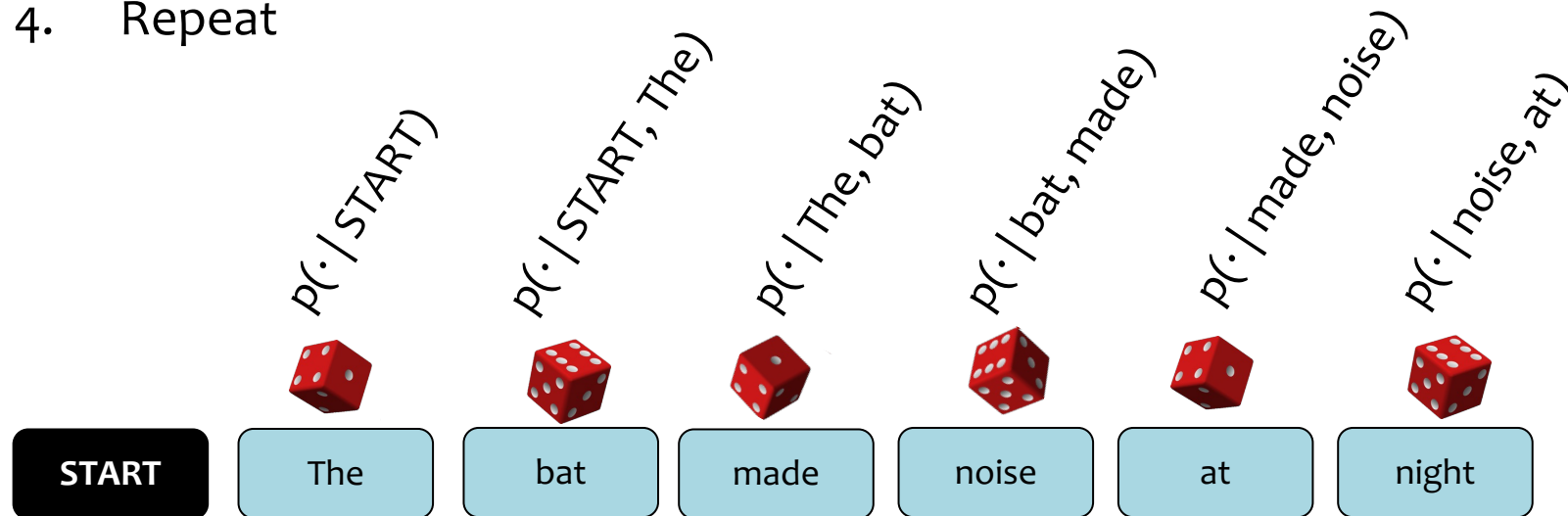
w_t	$p(\cdot \mid \cdot, \cdot)$
corn	4/11
grass	3/11
hay	2/11
if	1/11
which	1/11

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 $p(w_t | w_{t-2}, w_{t-1})$
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 $p(w_t | w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat

Training Data (Shakespeare)

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.

5-Gram Model

Approacheth, denay. dungy
Thither! Julius think: grant,--0
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)

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

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

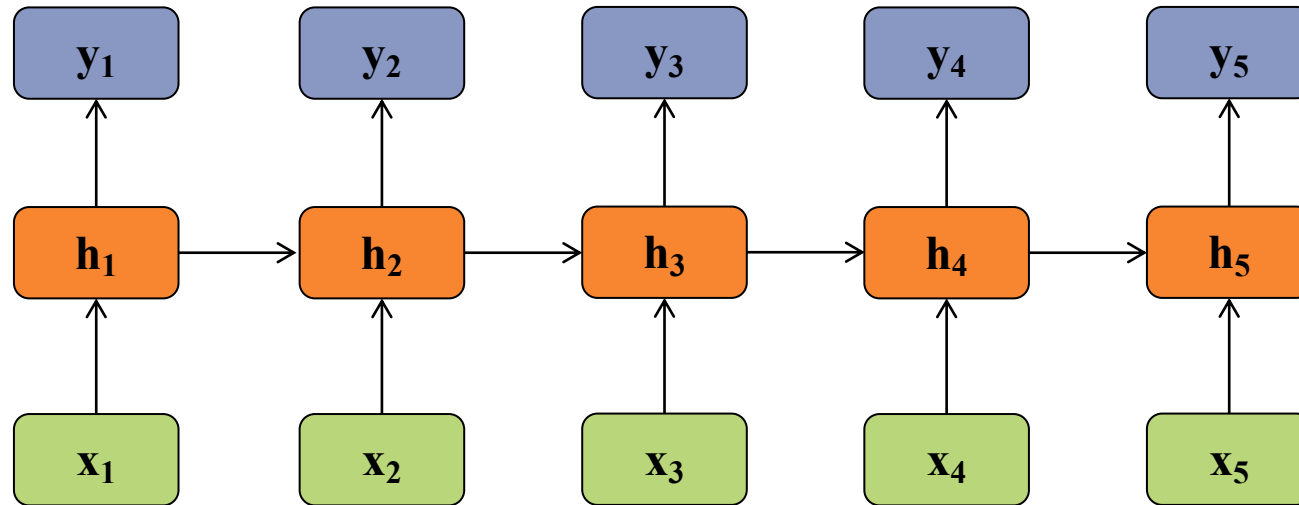
outputs: $\mathbf{y} = (y_1, y_2, \dots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: \mathcal{H}

Definition of the RNN:

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

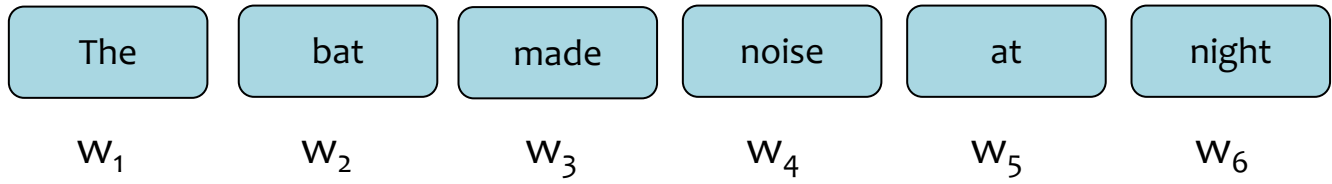
$$y_t = W_{hy}h_t + b_y$$



Recall...

The Chain Rule of Probability

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

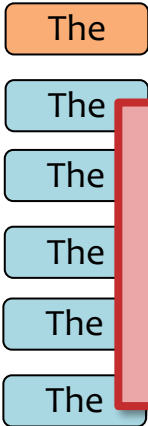


Chain rule of probability:
$$p(w_1, w_2, \dots, w_T) = \prod_{t=1}^T p(w_t | w_{t-1}, \dots, w_1)$$

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

$$p(w_1)$$

$$p(w_2 | w_1)$$



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

$$p(w_6 | w_5, w_4, w_3, w_2, w_1)$$

RNN Language Model

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

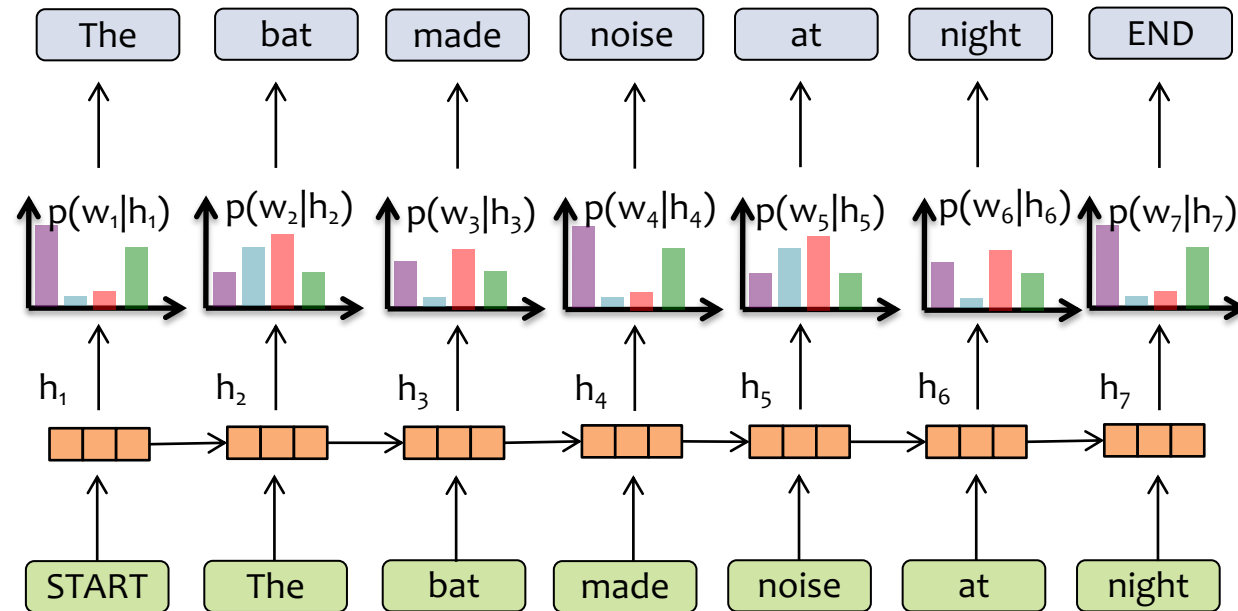
$$p(w_1, w_2, w_3, \dots, w_6) =$$

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

Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector

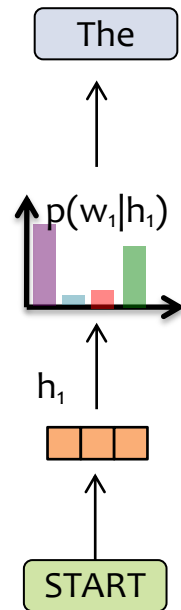
RNN Language Model



Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

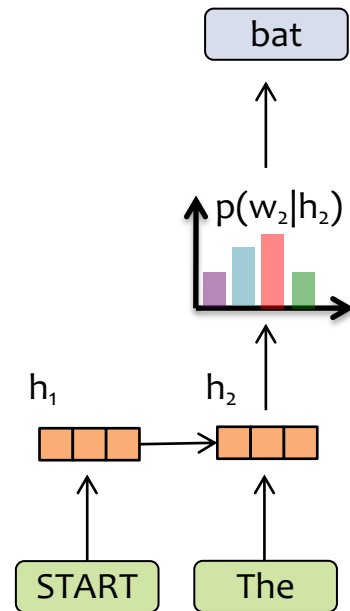
RNN Language Model



Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

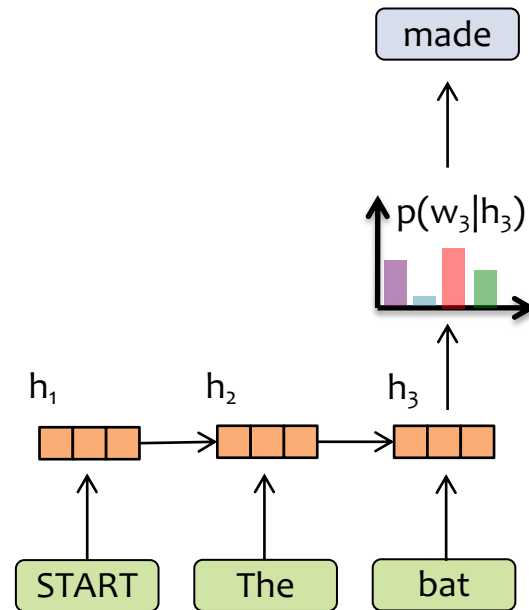
RNN Language Model



Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

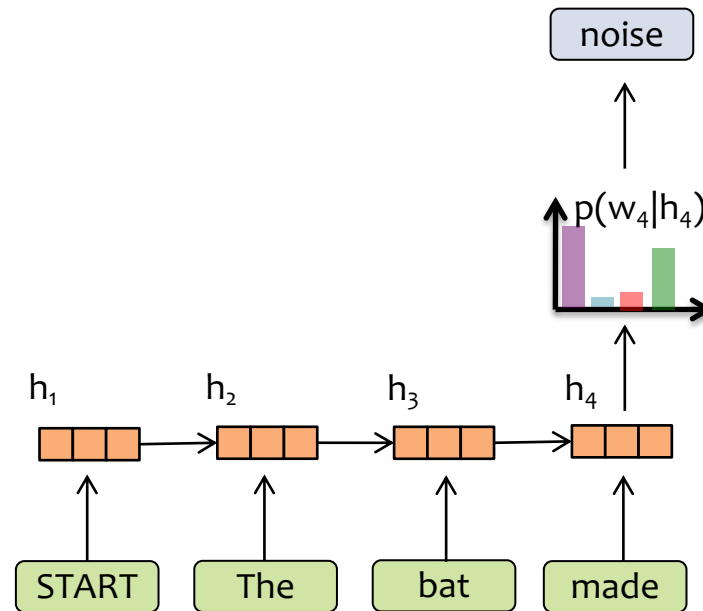
RNN Language Model



Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

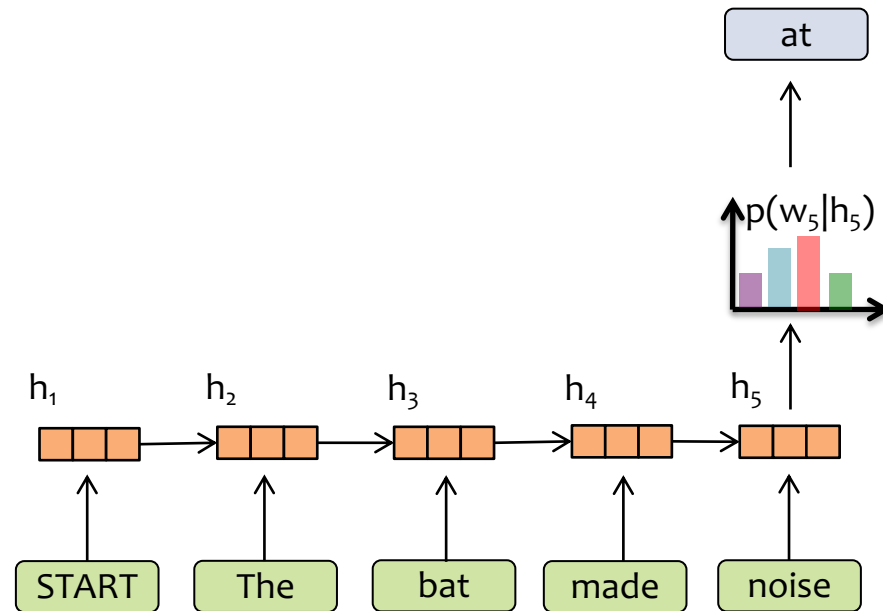
RNN Language Model



Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

RNN Language Model



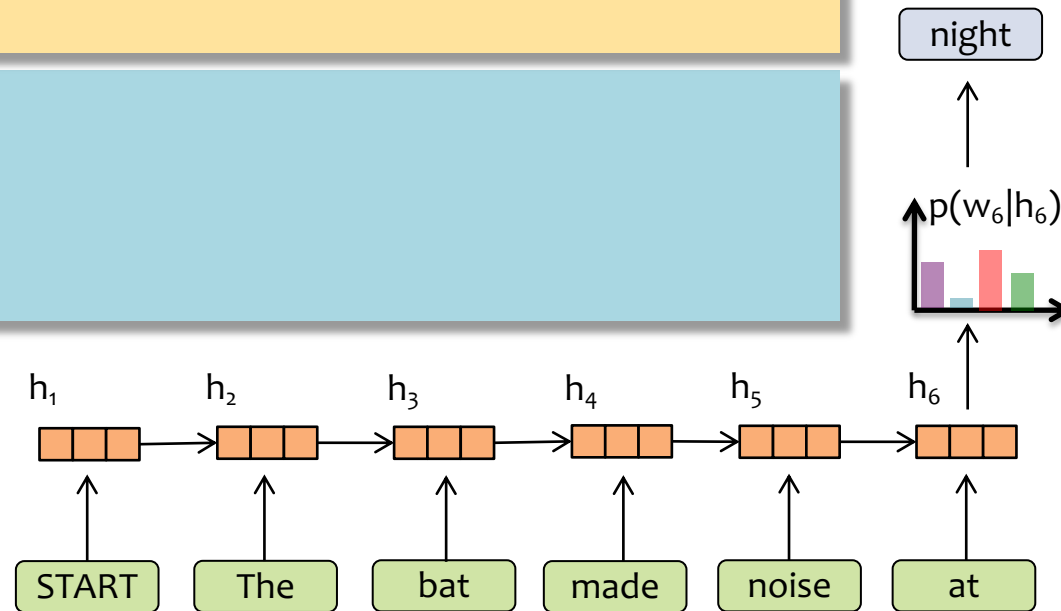
Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

RNN Language Model

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

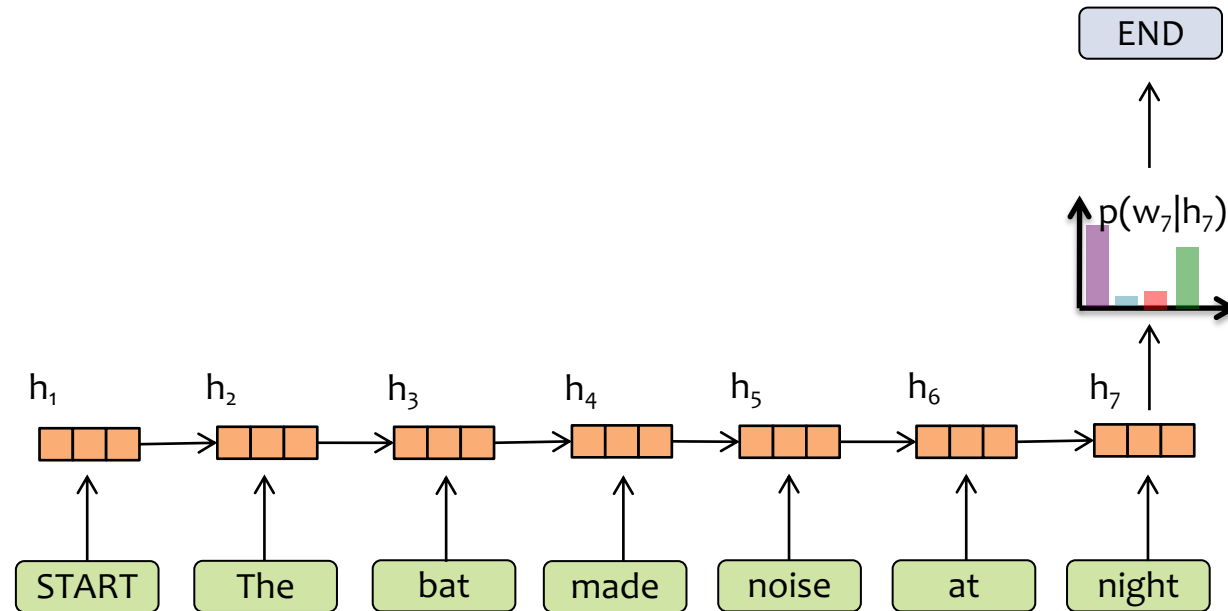
Answer:



Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

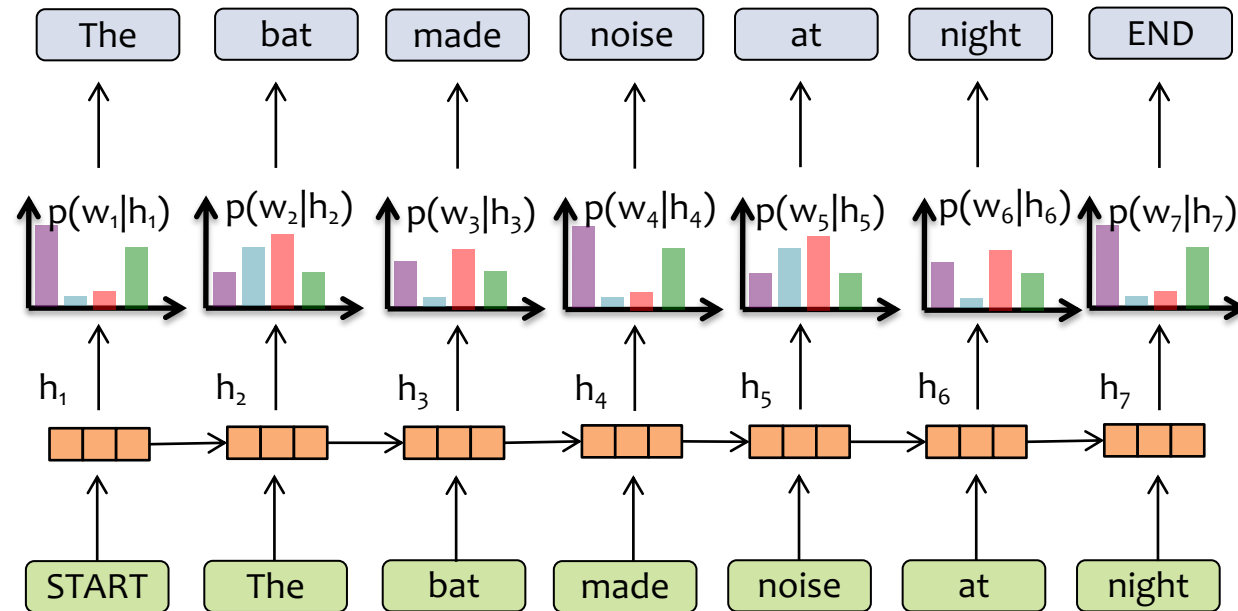
RNN Language Model



Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

RNN Language Model



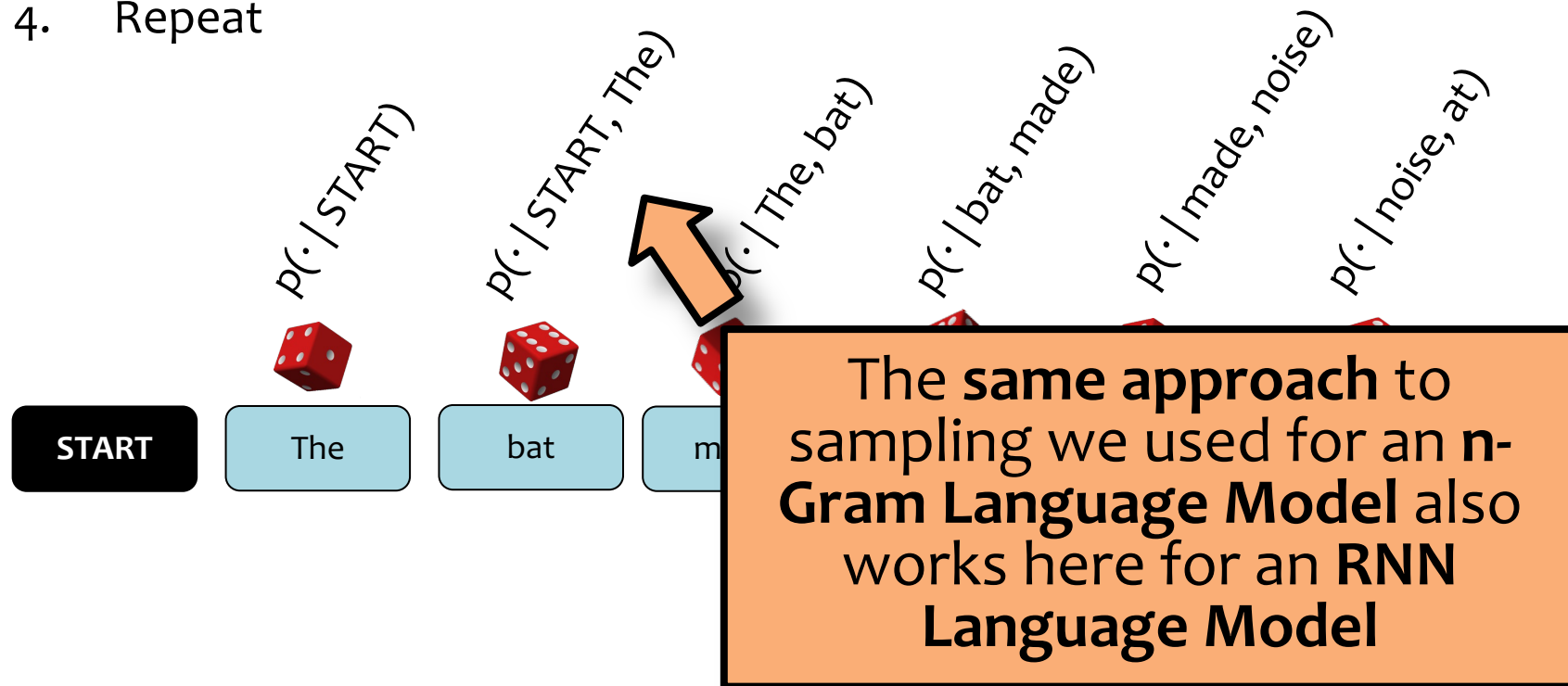
$$p(w_1, w_2, w_3, \dots, w_T) = p(w_1 | h_1) p(w_2 | h_2) \dots p(w_T | h_T)$$

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 $p(w_t | w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat



Sampling from an RNN-LM

??

VIOLA: Why, Salisbury must find his flesh and thought
That which I am not apt, 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
tender; and, for your love, I would be
as I must, for my own honour, if he
fore, out of my love to you, I came hither
to acquaint you withal, that either you might stay him
from his intended, or brook such disgrace well as he
shall run into, in that 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?!



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 apt, 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.

Sampling from an RNN-LM

RNN-LM Sample

VIOLA: Why, Salisbury must find his flesh and thought
That which I am not apt, 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.

Sampling from an RNN-LM

??

VIOLA: Why, Salisbury must find his flesh and thought
That which I am not apt, 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
tender; and, for your love, I would be
as I must, for my own honour, if he
fore, out of my love to you, I came hither
to acquaint you withal, that either you might stay him
from his intended, or brook such disgrace well as he
shall run into, in that 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?!

LEARNING AN RNN

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

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

Sample 1:						} $\mathbf{y}^{(1)}$
						} $\mathbf{x}^{(1)}$
Sample 2:						} $\mathbf{y}^{(2)}$
						} $\mathbf{x}^{(2)}$
Sample 3:						} $\mathbf{y}^{(3)}$
						} $\mathbf{x}^{(3)}$
Sample 4:						} $\mathbf{y}^{(4)}$
						} $\mathbf{x}^{(4)}$

SGD and Mini-batch SGD

Algorithm 1 SGD

```
1: Initialize  $\theta^{(0)}$ 
2:
3:
4:  $s = 0$ 
5: for  $t = 1, 2, \dots, T$  do
6:   for  $i \in \text{shuffle}(1, \dots, N)$  do
7:     Select the next training point  $(x_i, y_i)$ 
8:     Compute the gradient  $g^{(s)} = \nabla J_i(\theta^{(s-1)})$ 
9:     Update parameters  $\theta^{(s)} = \theta^{(s-1)} - \eta g^{(s)}$ 
10:    Increment time step  $s = s + 1$ 
11:    Evaluate average training loss  $J(\theta) = \frac{1}{n} \sum_{i=1}^n J_i(\theta)$ 
12: return  $\theta^{(s)}$ 
```

SGD and Mini-batch SGD

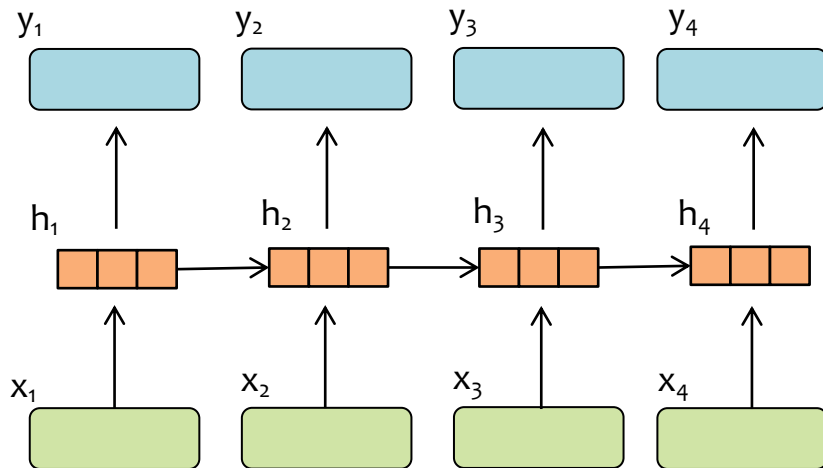
Algorithm 1 Mini-Batch SGD

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

RNN

Algorithm 1 Elman RNN

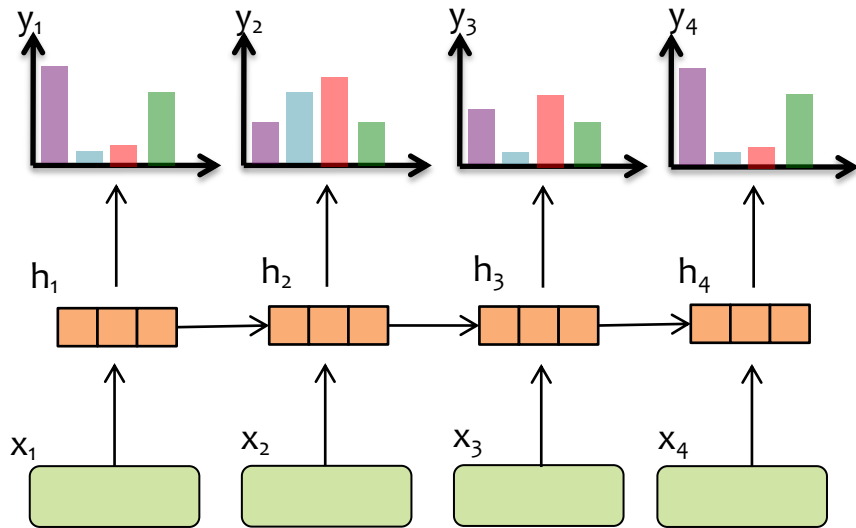
- 1: **procedure** FORWARD($x_{1:T}, W_{ah}, W_{ax}, b_a, W_{yh}, b_y$)
 - 2: Initialize the hidden state h_0 to zeros
 - 3: **for** t in 1 to T **do**
 - 4: Receive input data at time step t : x_t
 - 5: Compute the hidden state update:
 - 6: $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$
 - 7: $h_t = \sigma(a_t)$
 - 8: Compute the output at time step t :
 - 9: $y_t = W_{yh} \cdot h_t + b_y$
-



RNN

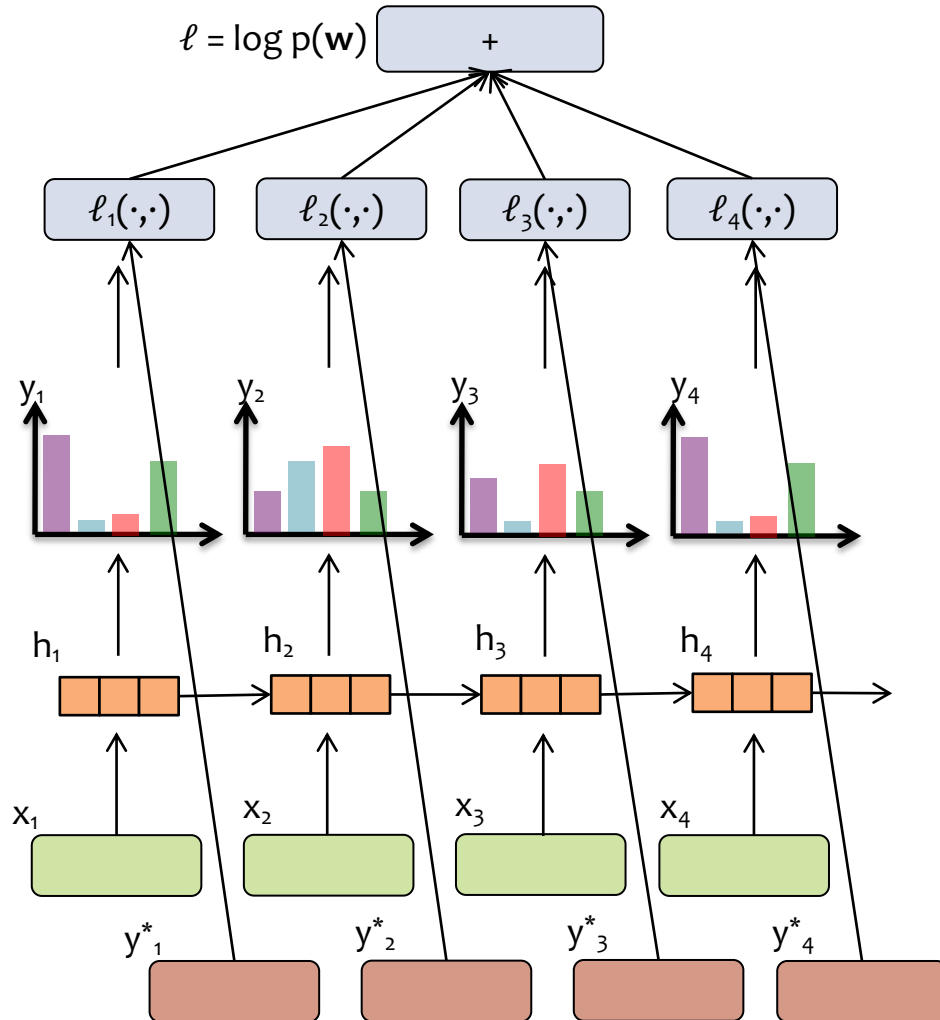
Algorithm 1 Elman RNN

- 1: **procedure** FORWARD($x_{1:T}, W_{ah}, W_{ax}, b_a, W_{yh}, b_y$)
 - 2: Initialize the hidden state h_0 to zeros
 - 3: **for** t in 1 to T **do**
 - 4: Receive input data at time step t : x_t
 - 5: Compute the hidden state update:
6: $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$
7: $h_t = \sigma(a_t)$
 - 8: Compute the output at time step t :
9: $y_t = \text{softmax}(W_{yh} \cdot h_t + b_y)$
-



RNN + LOSS

Algorithm 1 Elman RNN + Loss




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

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

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


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

... 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**?...

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

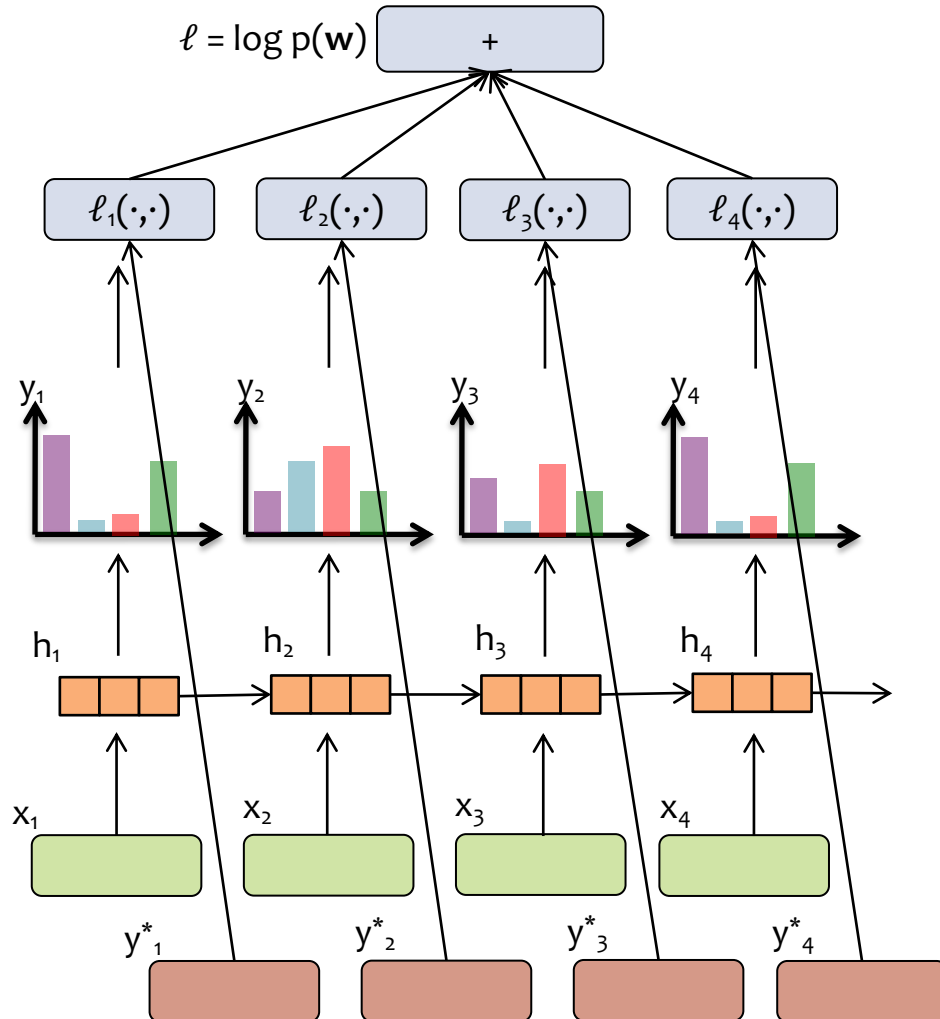
- This counting method gives us the **maximum likelihood estimate** of the n-gram LM parameters
- We can derive it in the usual way:
 - **Write the likelihood** of the sentences under the n-gram LM
 - **Set the gradient to zero** and impose the constraint that the probabilities sum-to-one
 - **Solve**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

RNN + LOSS

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

Algorithm 1 Elman RNN + Loss

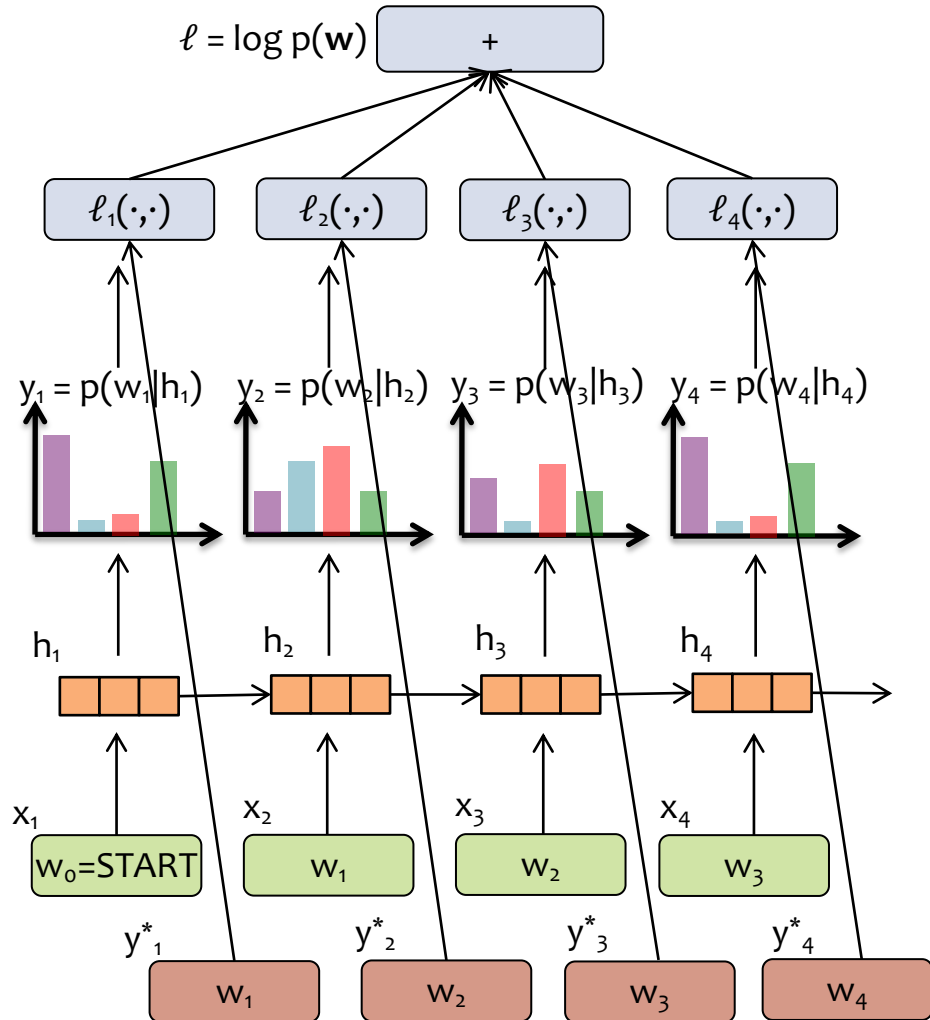
- 1: **procedure** FORWARD($x_{1:T}, y_{1:T}^*, W_{ah}, W_{ax}, b_a, W_{yh}, b_y$)
- 2: Initialize the hidden state h_0 to zeros
- 3: **for** t in 1 to T **do**
- 4: Receive input data at time step t : x_t
- 5: Compute the hidden state update:
6: $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$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



RNN-LM + LOSS

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

$$\begin{aligned} \log p(\mathbf{w}) &= \log p(w_1, w_2, w_3, \dots, w_T) \\ &= \log p(w_1 | h_1) + \dots + \log p(w_T | h_T) \end{aligned}$$



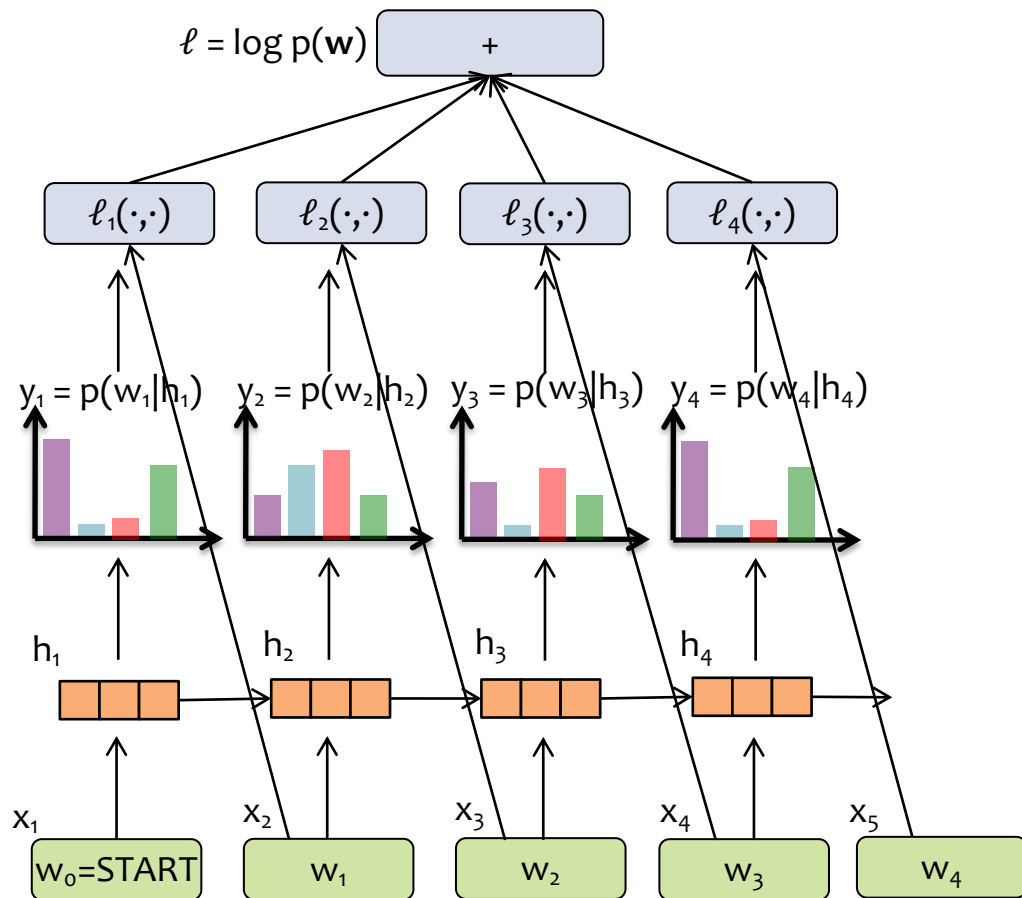
Algorithm 1 Elman RNN + Loss

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

RNN-LM + LOSS

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

$$\begin{aligned} \log p(\mathbf{w}) &= \log p(w_1, w_2, w_3, \dots, w_T) \\ &= \log p(w_1 | h_1) + \dots + \log p(w_T | h_T) \end{aligned}$$



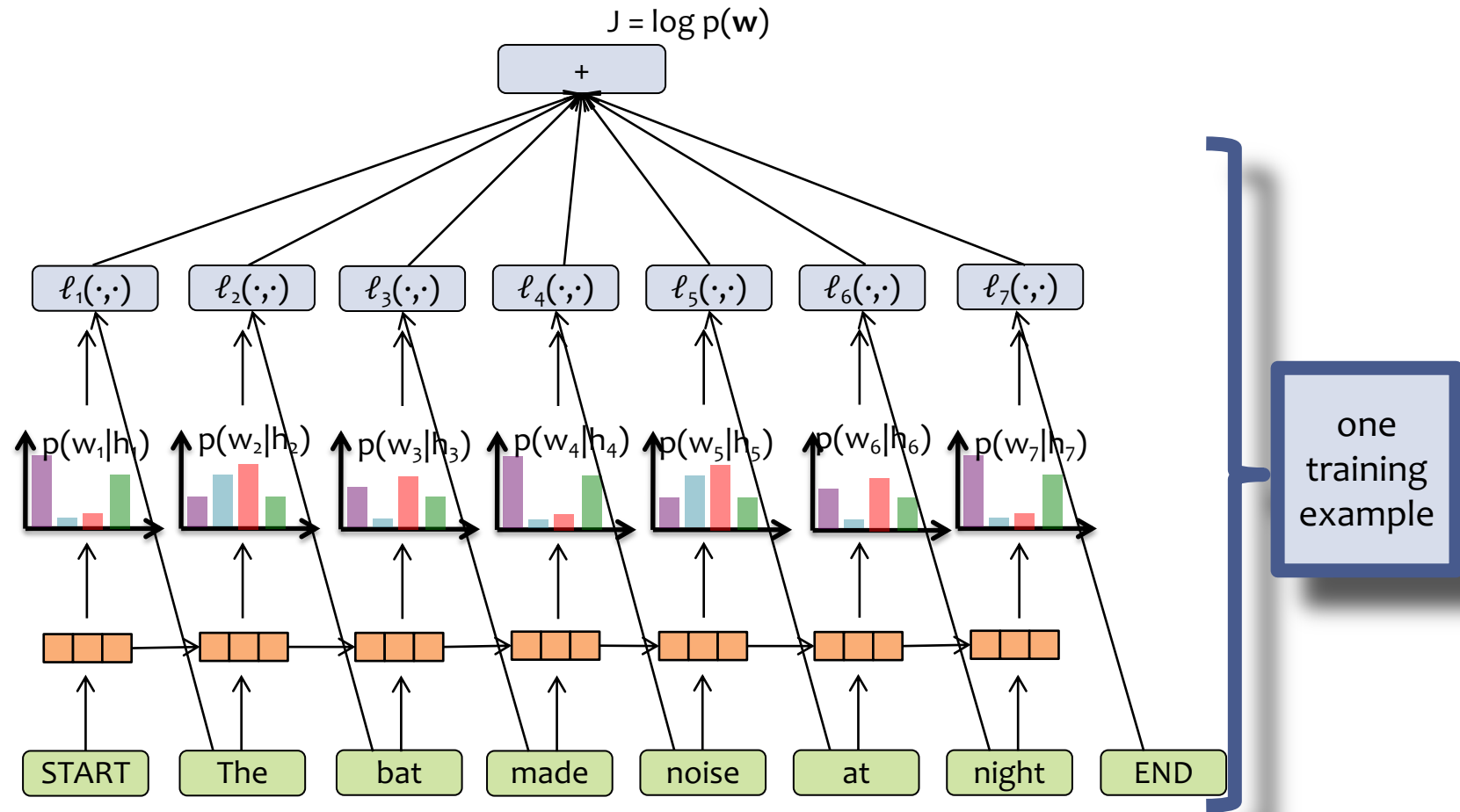
Algorithm 1 Elman RNN + Loss

- 1: **procedure** FORWARD($x_{1:T}, y_{1:T}^*, W_{ah}, W_{ax}, b_a, W_{yh}, b_y$)
- 2: Initialize the hidden state h_0 to zeros
- 3: **for** t in 1 to T **do**
- 4: Receive input data at time step t : x_t
- 5: Compute the hidden state update:
- 6: $a_t = W_{ah} \cdot h_{t-1} + W_{ax} \cdot x_t + b_a$
- 7: $h_t = \sigma(a_t)$
- 8: Compute the output at time step t :
- 9: $y_t = \text{softmax}(W_{yh} \cdot h_t + b_y)$
- 10: Compute the cross-entropy loss at time step t :
- 11: $l_t = - \sum_{k=1}^K (y_t^*)_k \log((y_t)_k)$
- 12: Compute the total loss:
- 13: $l = \sum_{t=1}^T l_t$

Learning an RNN-LM

- Each training example is a sequence (e.g. sentence), so we have training data $D = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \dots, \mathbf{w}^{(N)}\}$
- The objective function for a Deep LM (e.g. RNN-LM or Transformer-LM) is typically the log-likelihood of the training examples:
$$J(\boldsymbol{\theta}) = \sum_i \log p_{\boldsymbol{\theta}}(\mathbf{w}^{(i)})$$
- We train by mini-batch SGD (or your favorite flavor of mini-batch SGD)

$$\begin{aligned} \log p(\mathbf{w}) &= \log p(w_1, w_2, w_3, \dots, w_T) \\ &= \log p(w_1 | h_1) + \log p(w_2 | h_2) + \dots + \log p(w_T | h_T) \end{aligned}$$



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

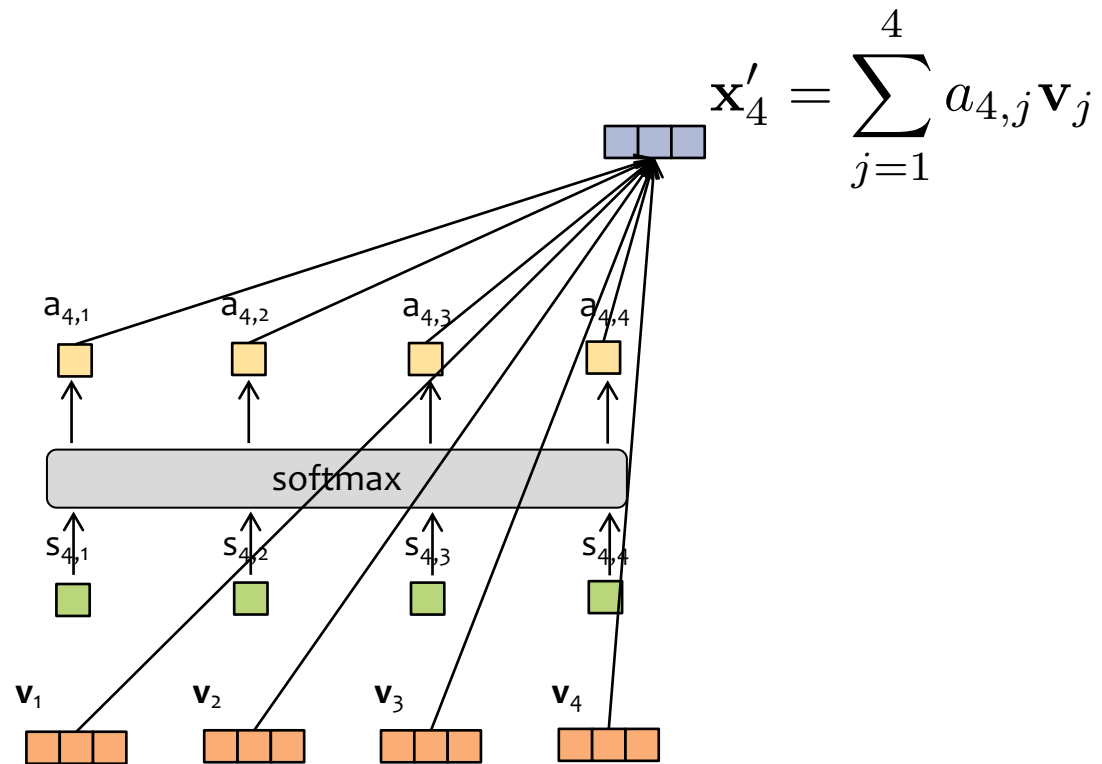
What is ChatGPT?

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

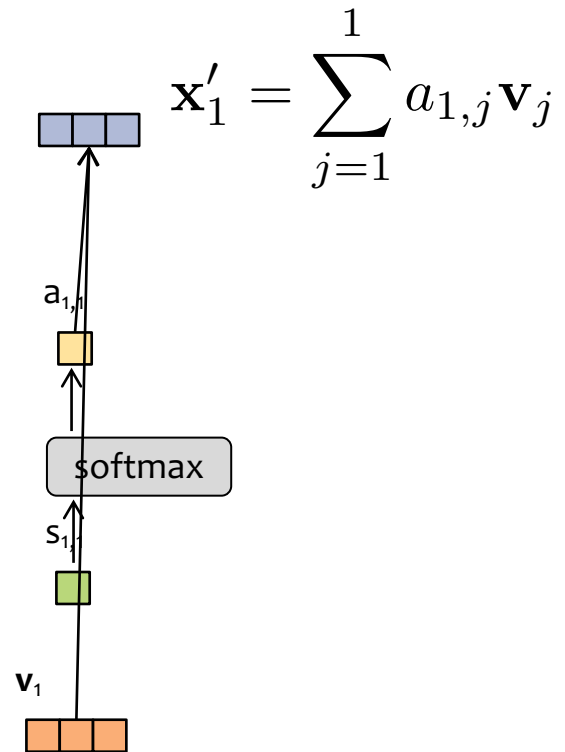
Transformer Language Models

MODEL: GPT

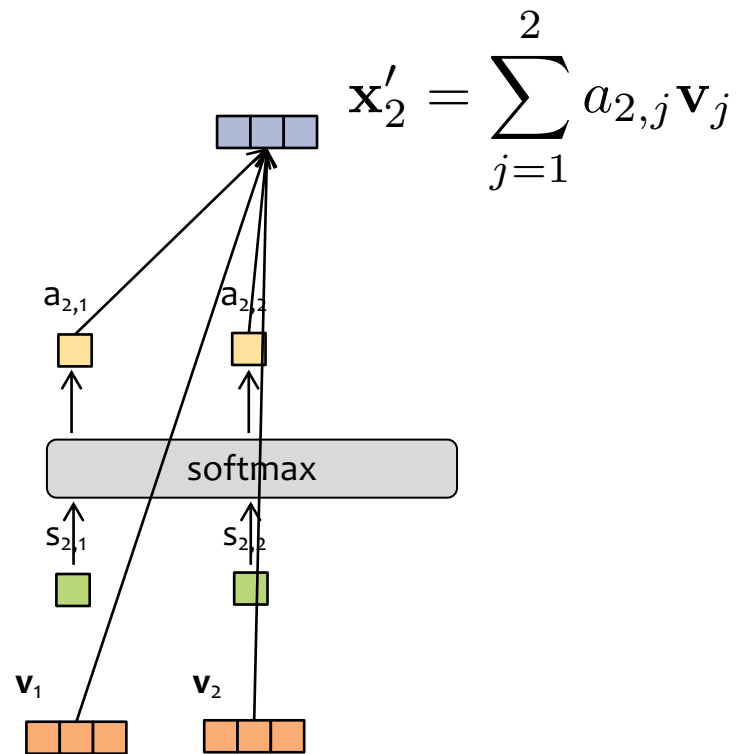
Attention



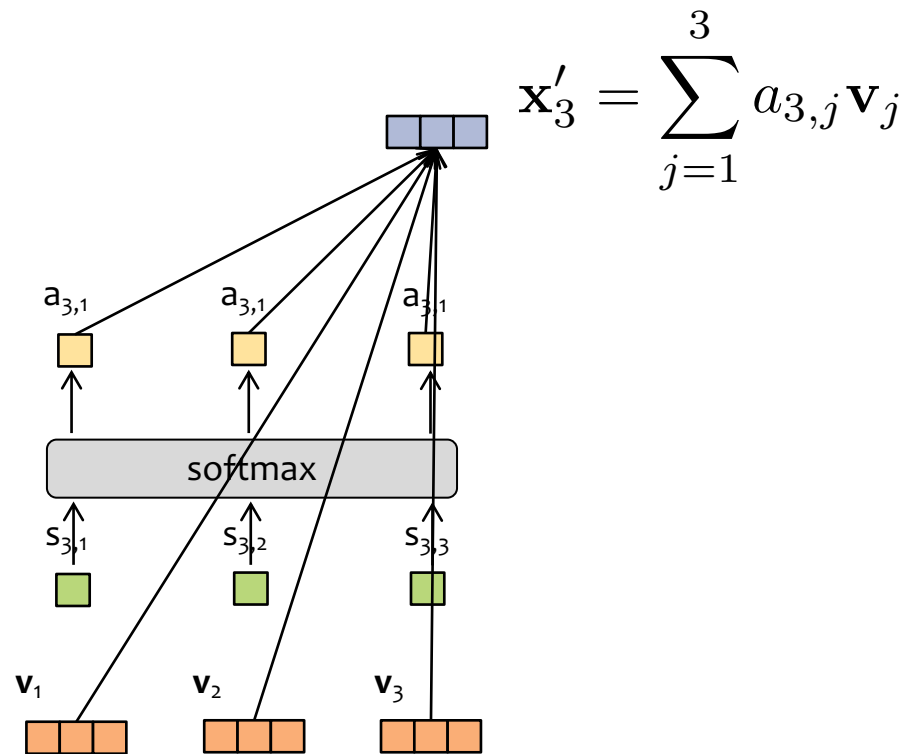
Attention



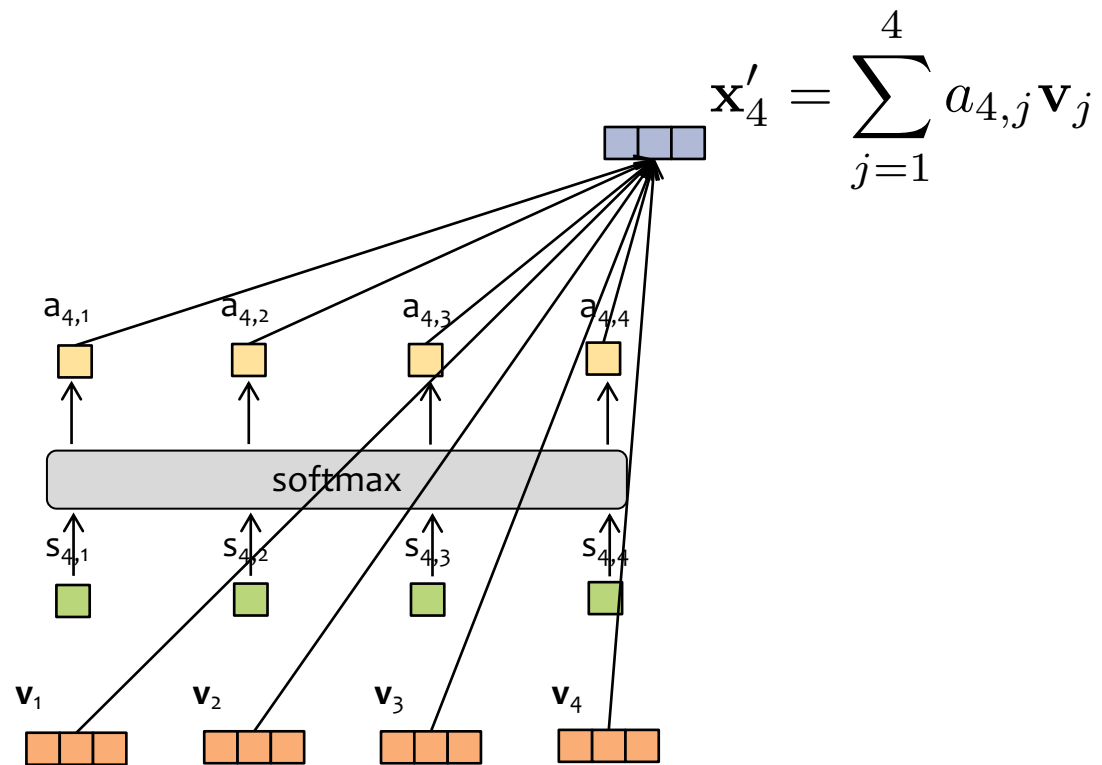
Attention



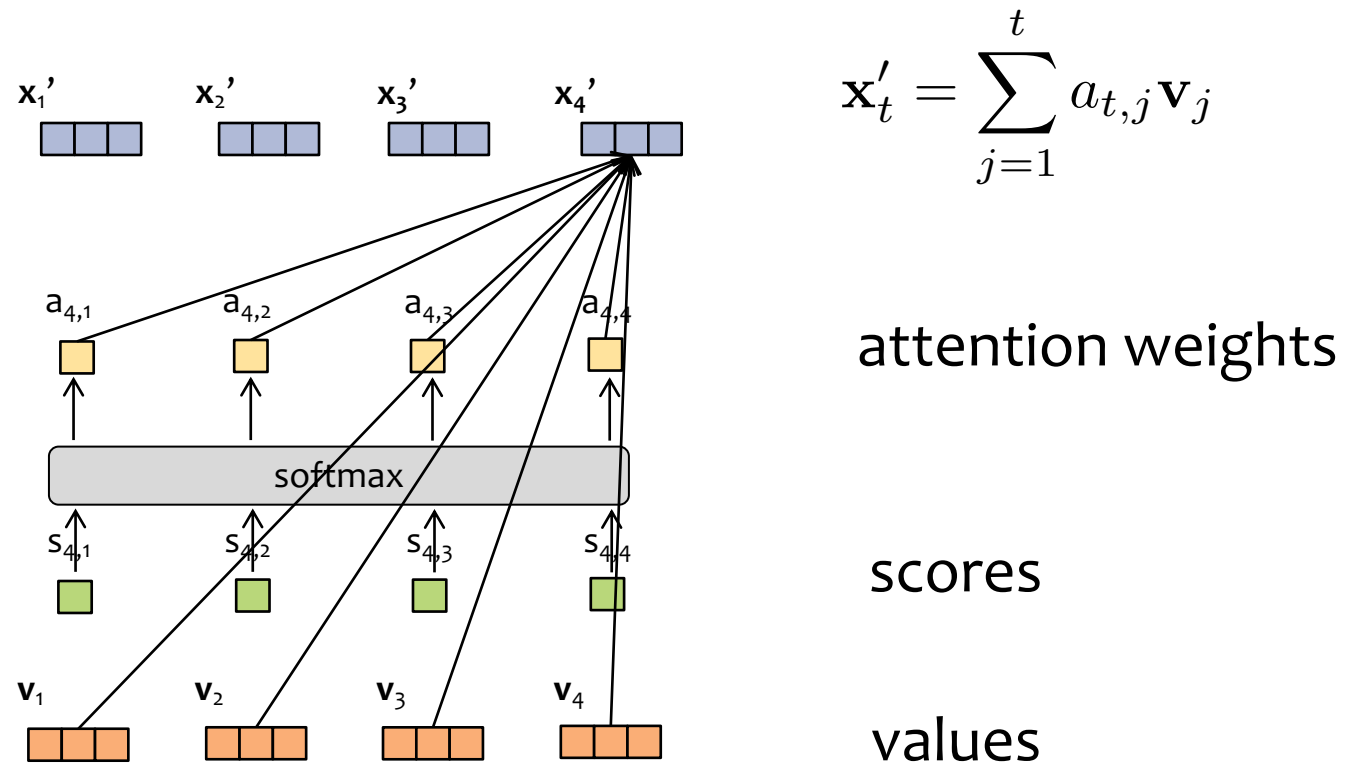
Attention



Attention

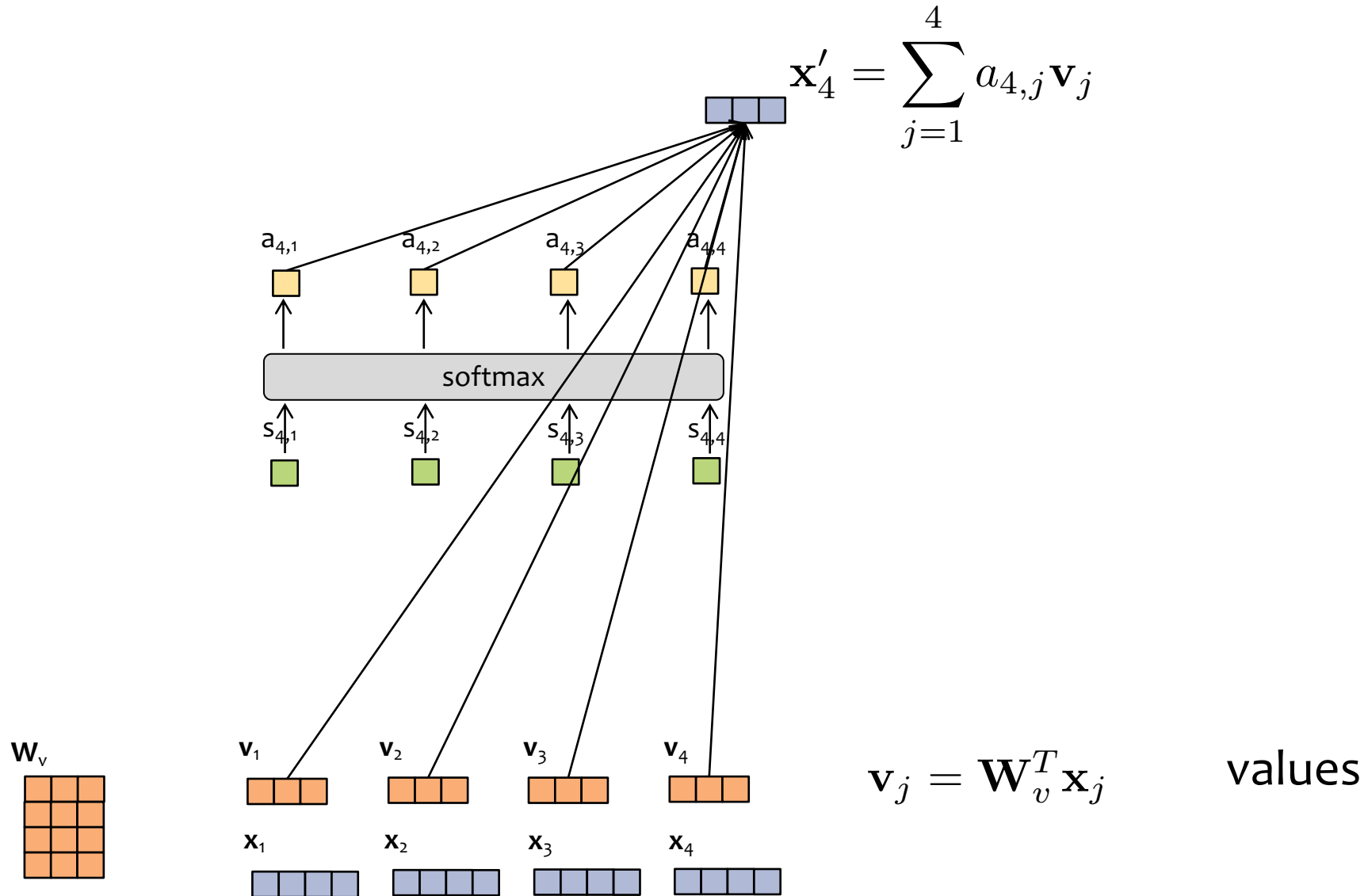


Attention

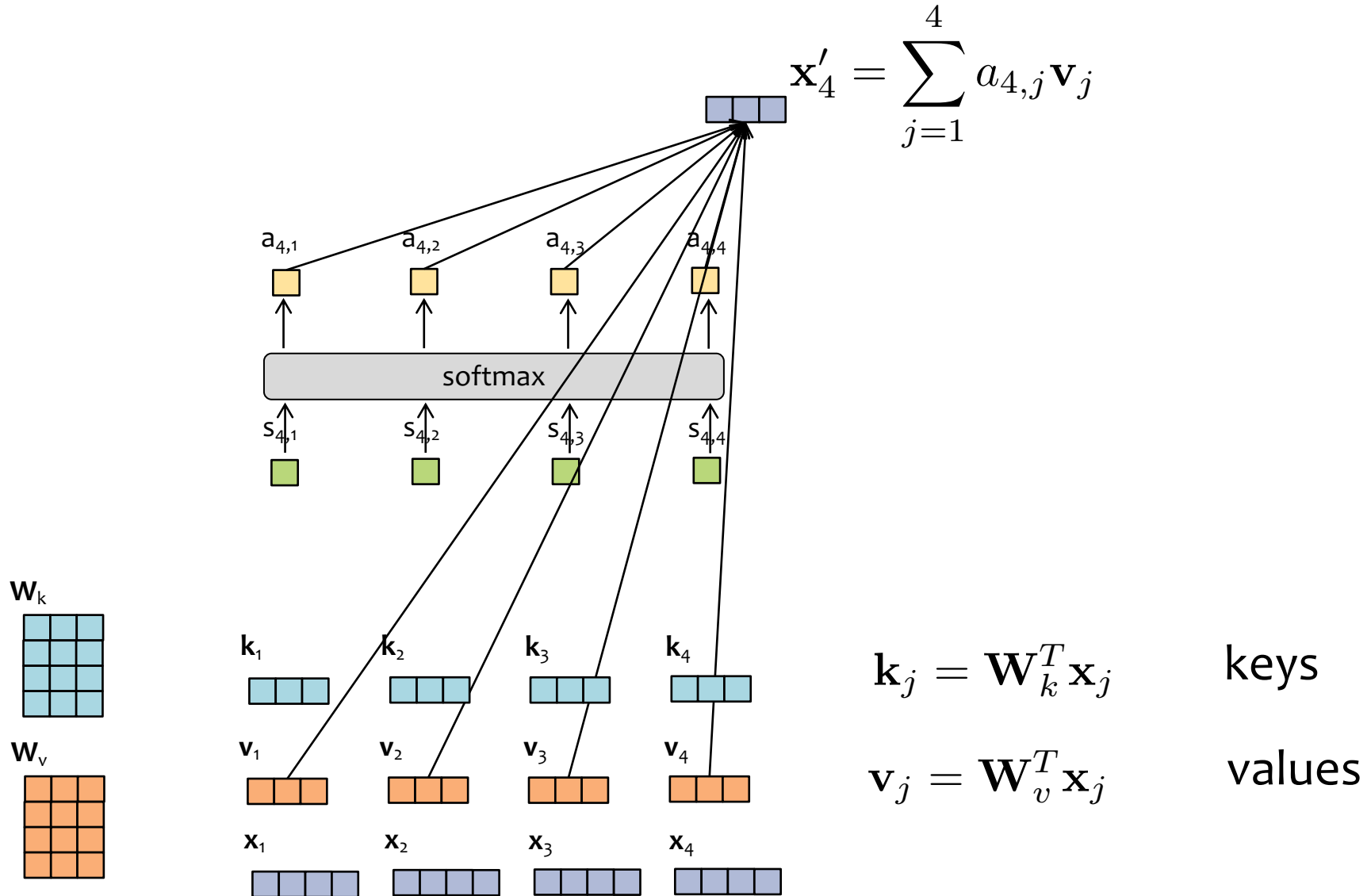


$$\mathbf{x}'_t = \sum_{j=1}^t a_{t,j} \mathbf{v}_j$$

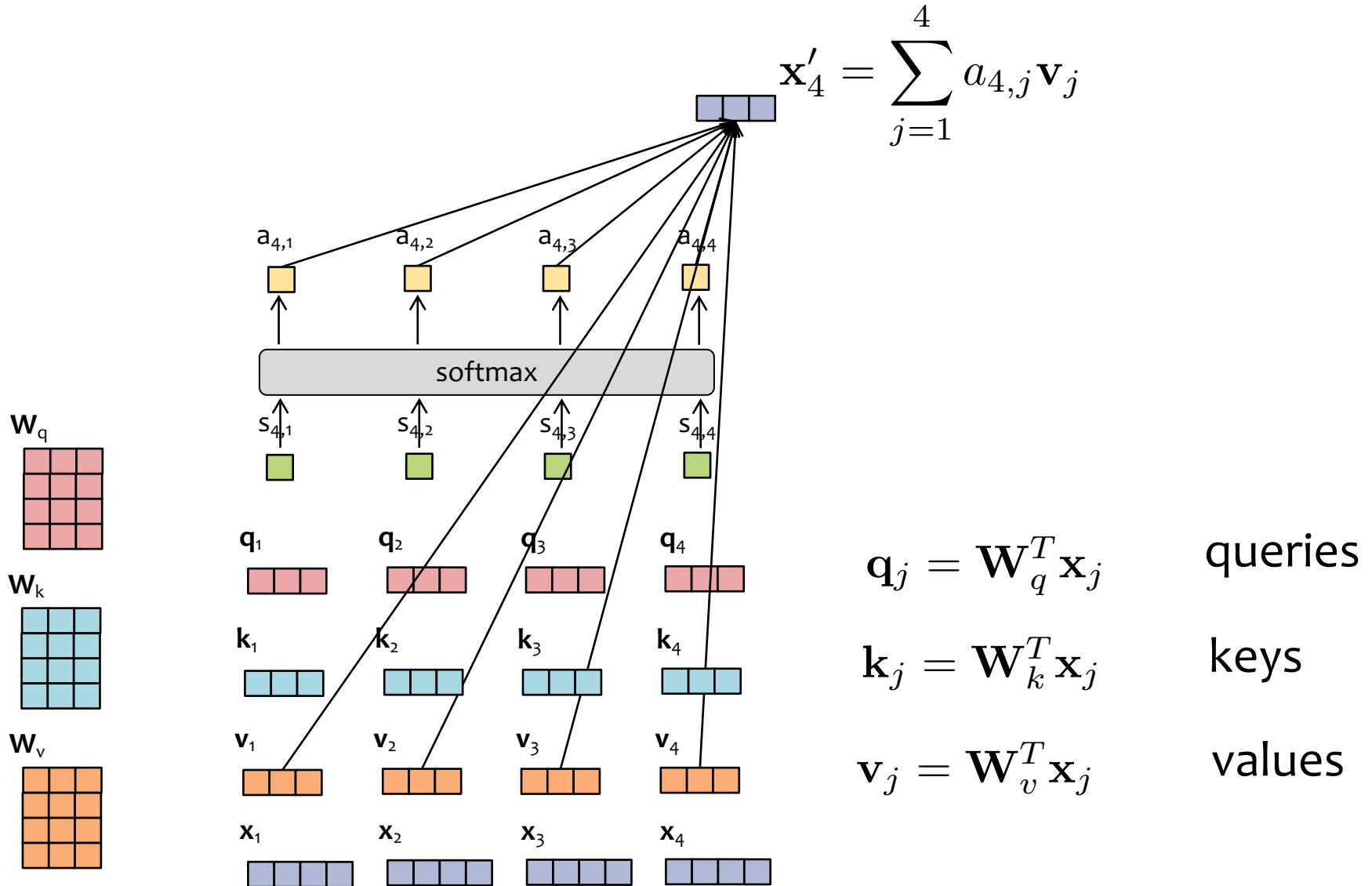
Scaled Dot-Product Attention



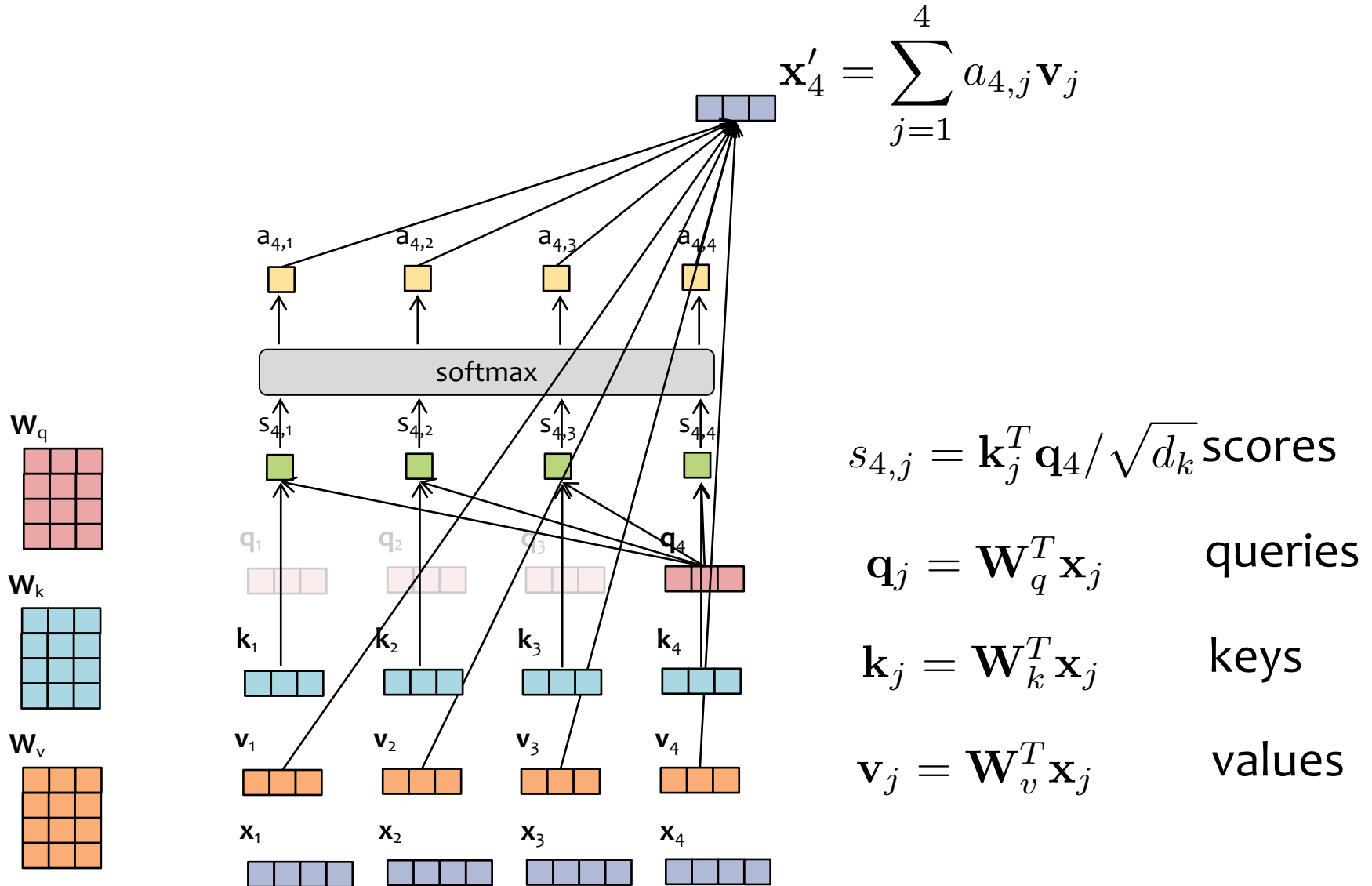
Scaled Dot-Product Attention



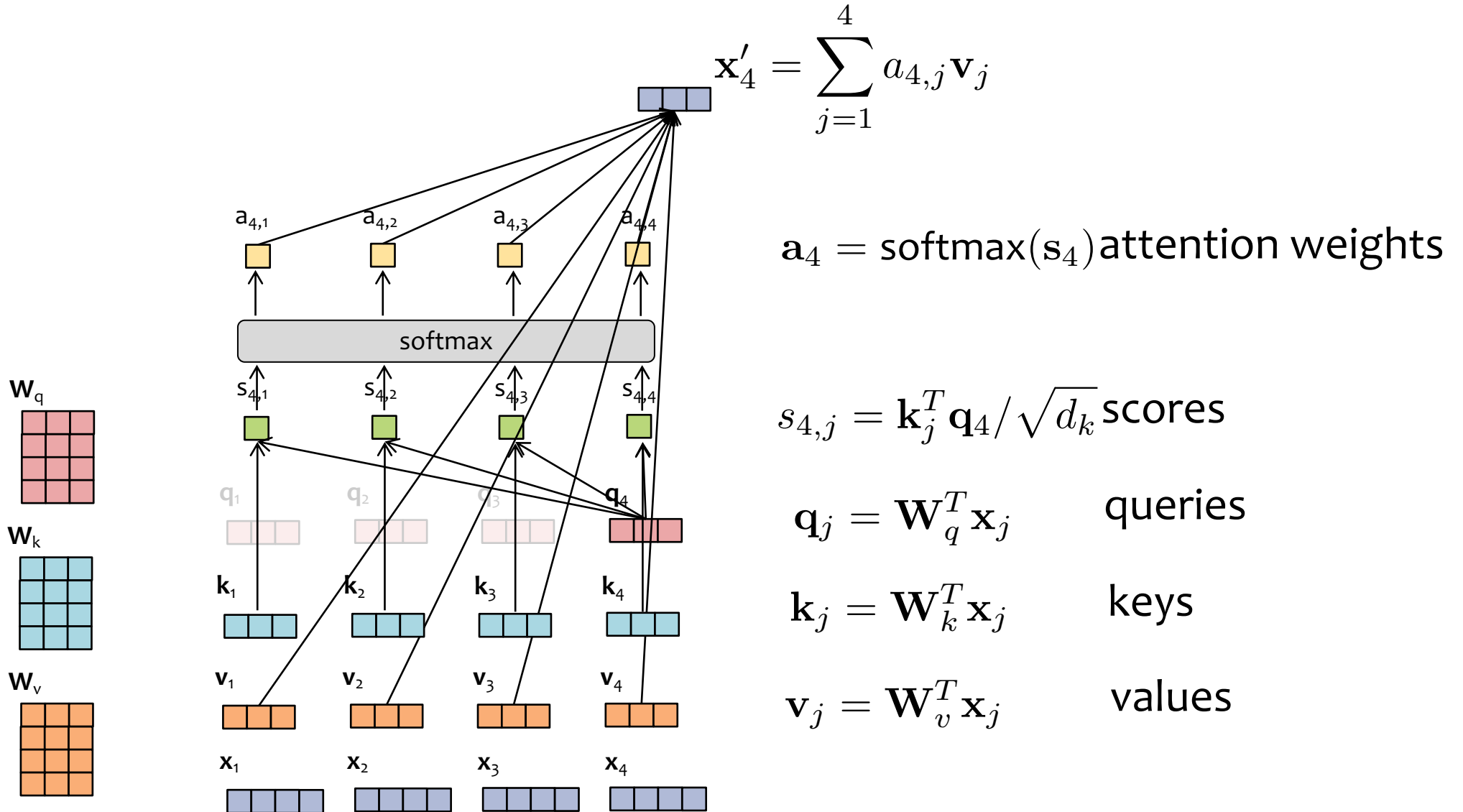
Scaled Dot-Product Attention



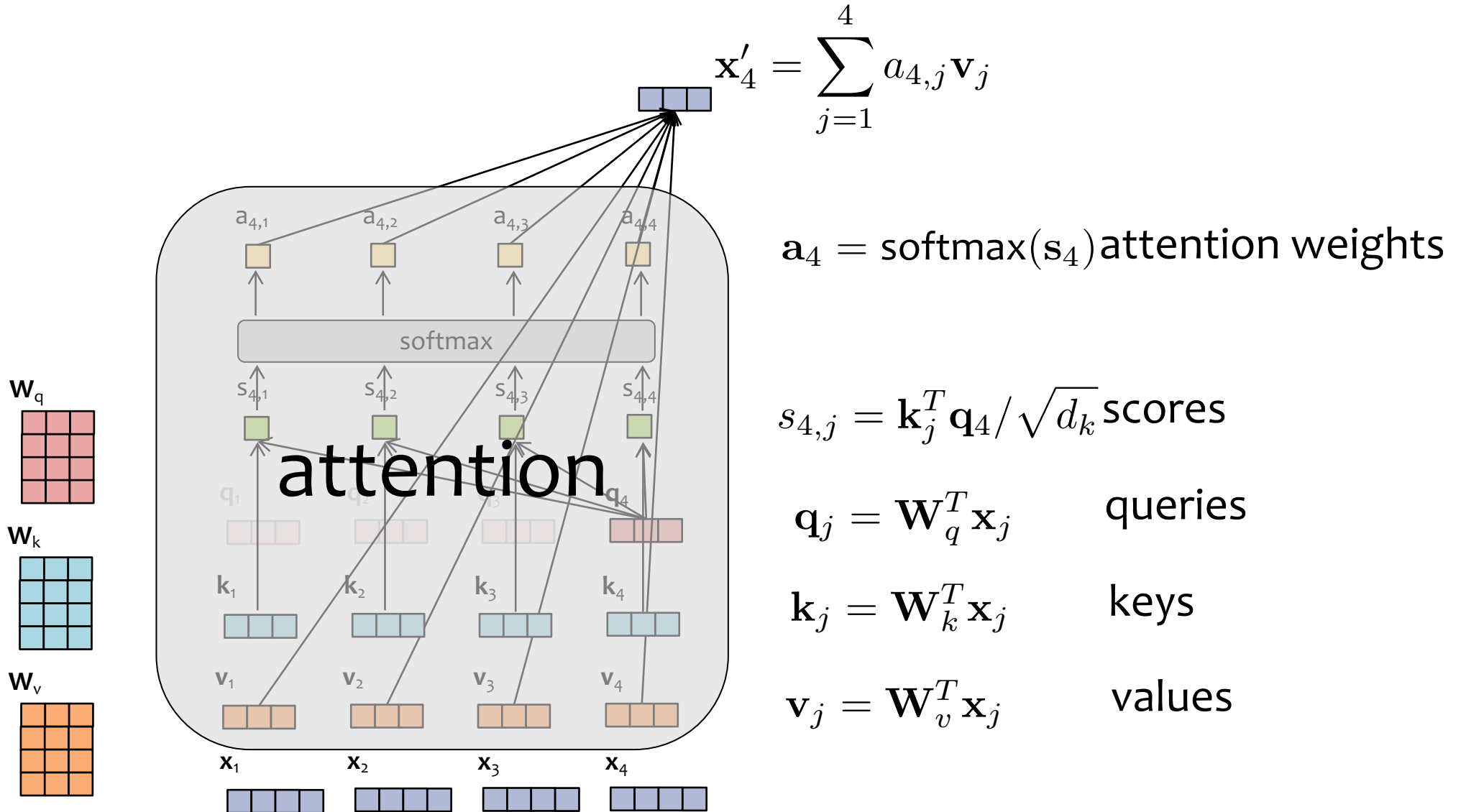
Scaled Dot-Product Attention



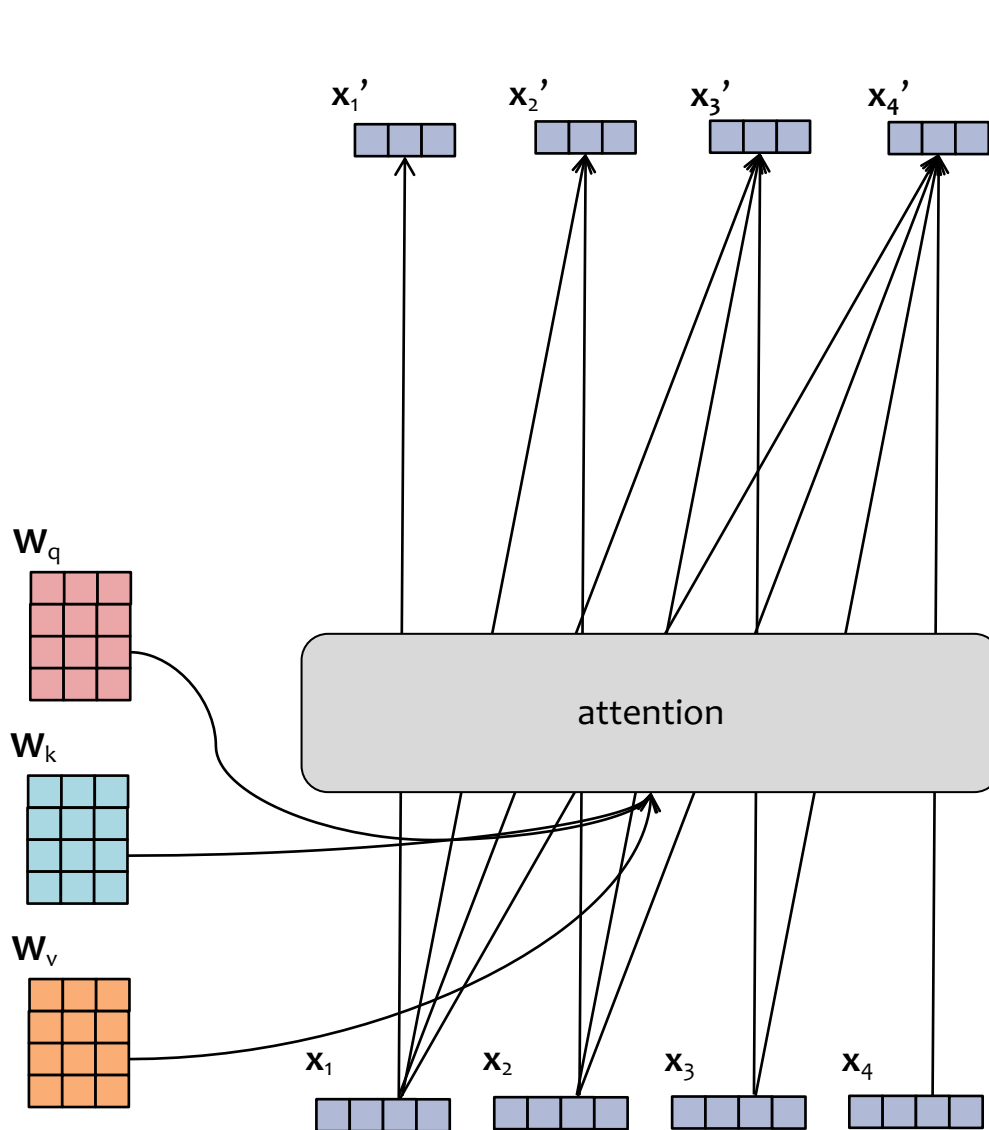
Scaled Dot-Product Attention



Scaled Dot-Product Attention



Scaled Dot-Product Attention



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

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

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

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

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

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

Animation of 3D Convolution

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

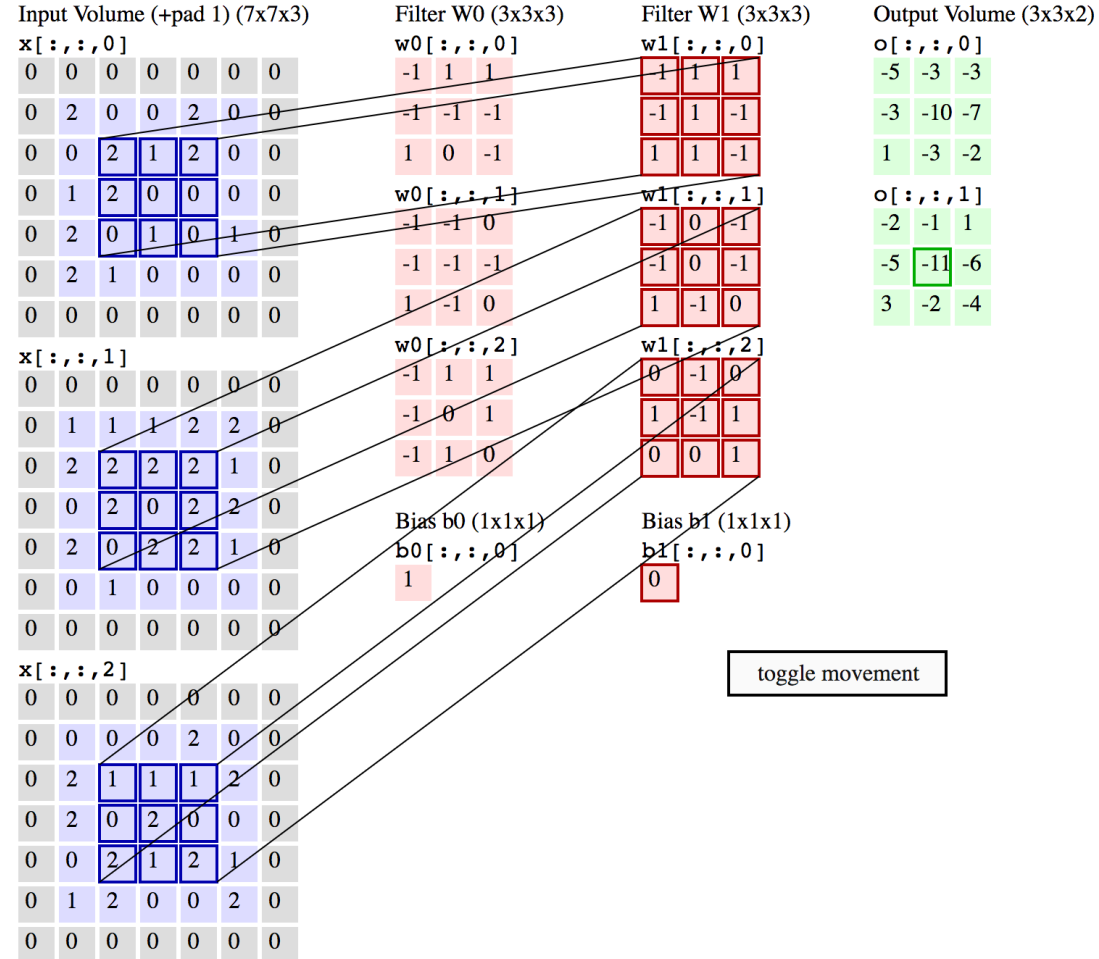
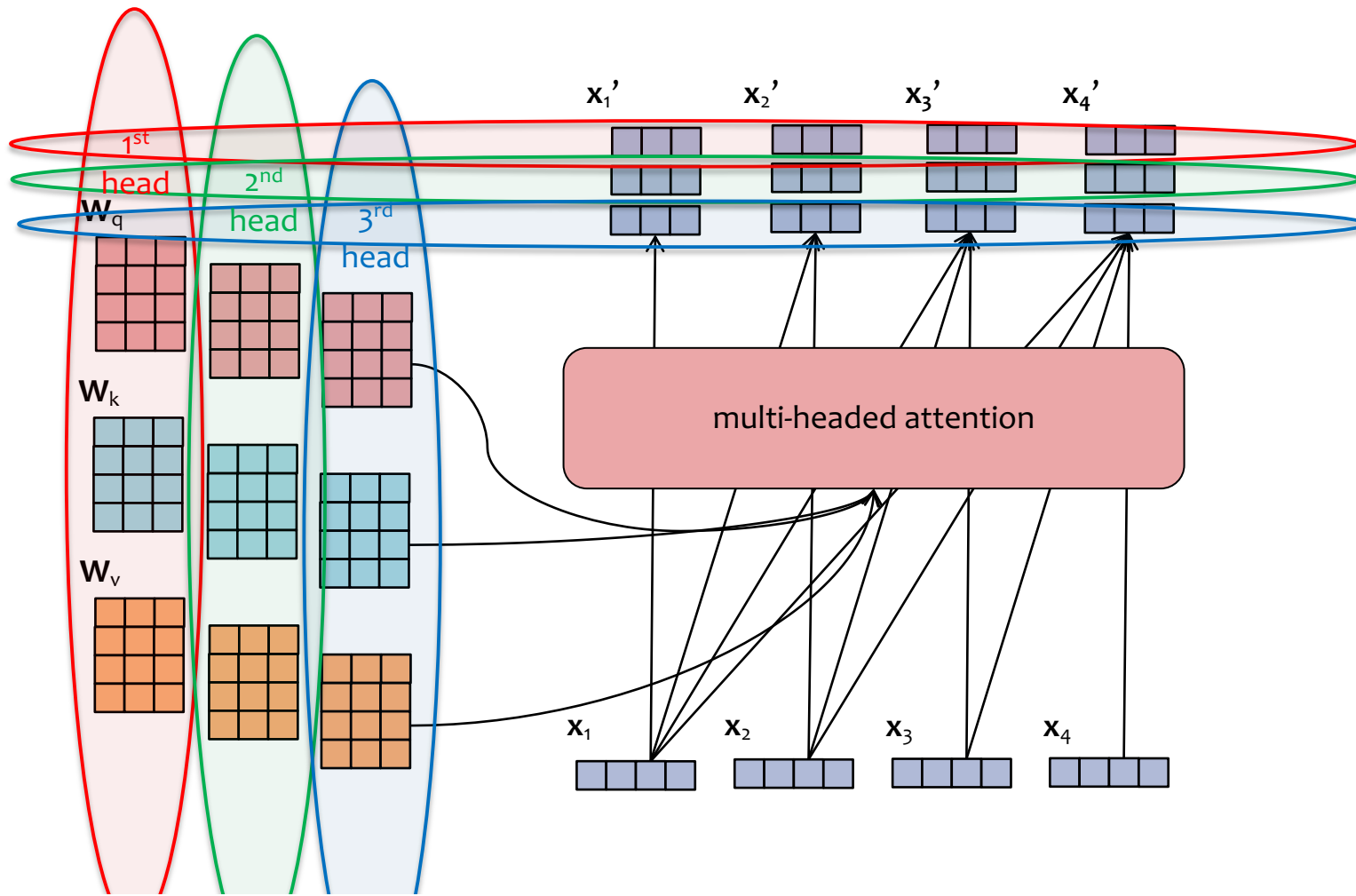


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

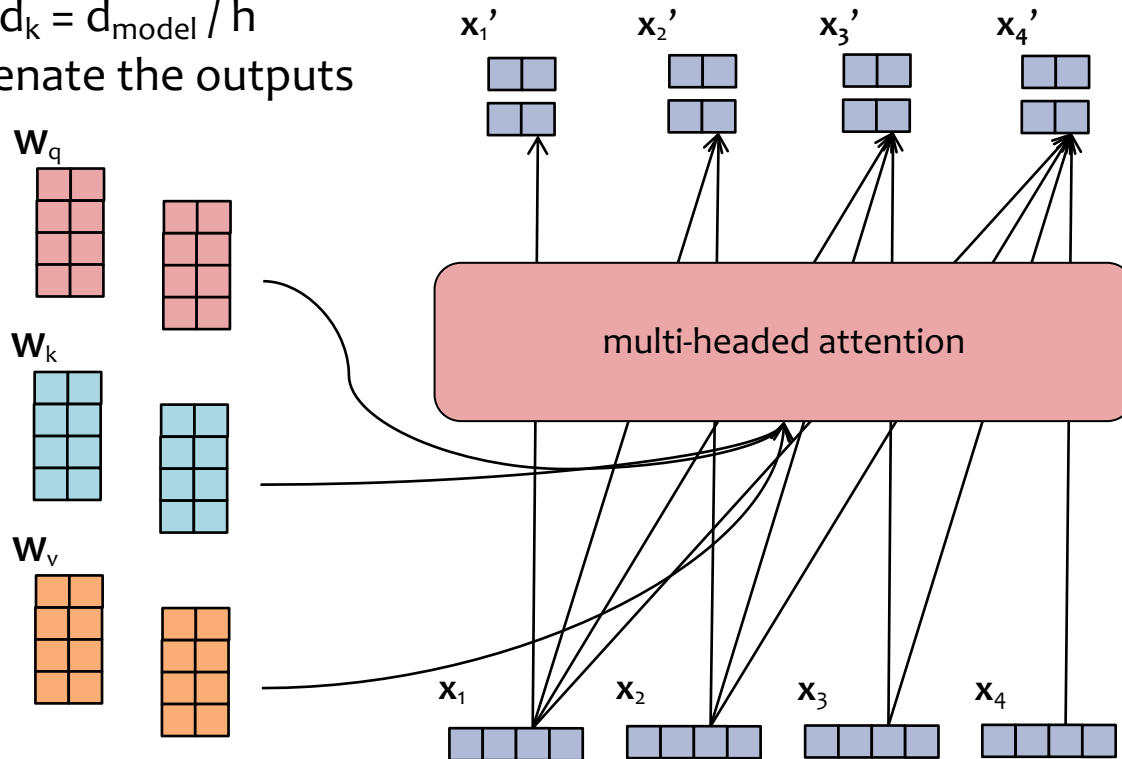
Multi-headed Attention



- 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

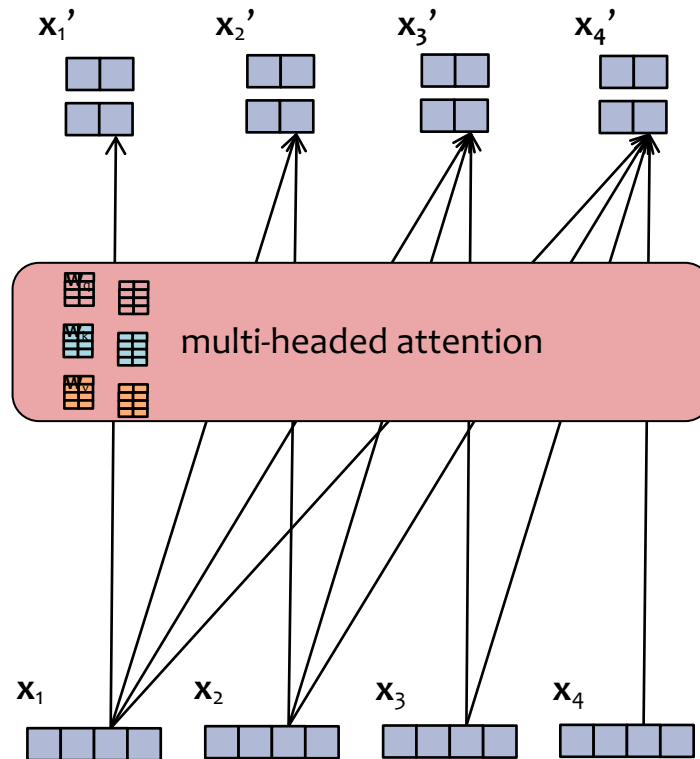
- To ensure the dimension of the **input** embedding x_t is the same as the **output** embedding x_t' , Transformers usually choose the embedding sizes and number of heads appropriately:
 - $d_{\text{model}} = \text{dim. of inputs}$
 - $d_k = \text{dim. of each output}$
 - $h = \# \text{ of heads}$
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

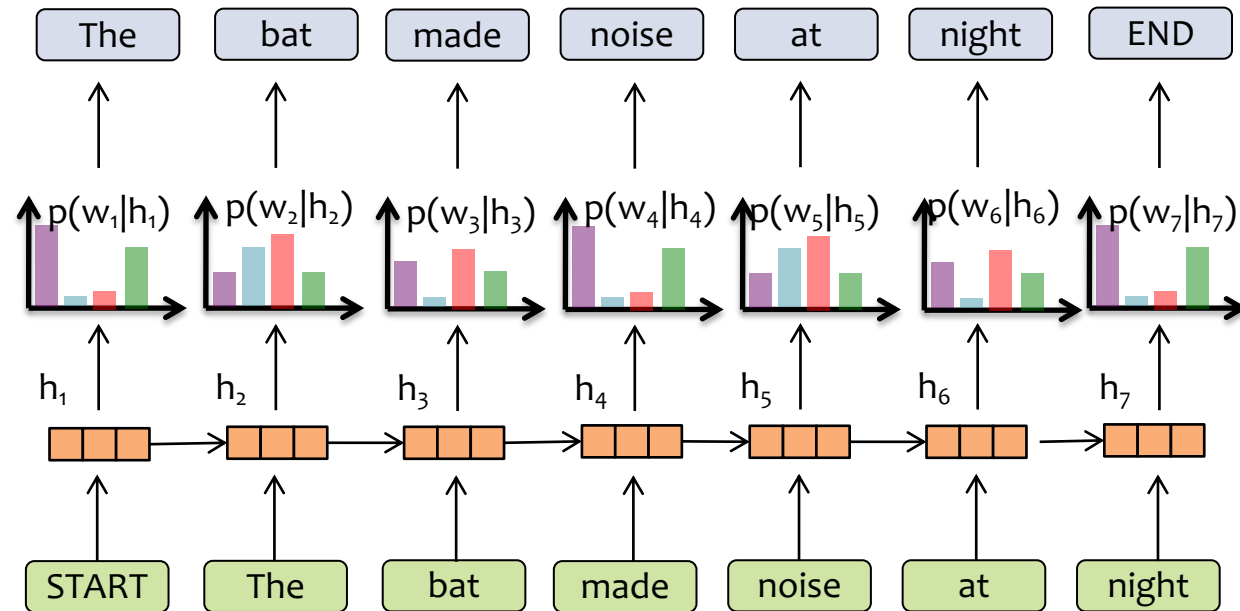
Multi-headed Attention

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

RNN Language Model



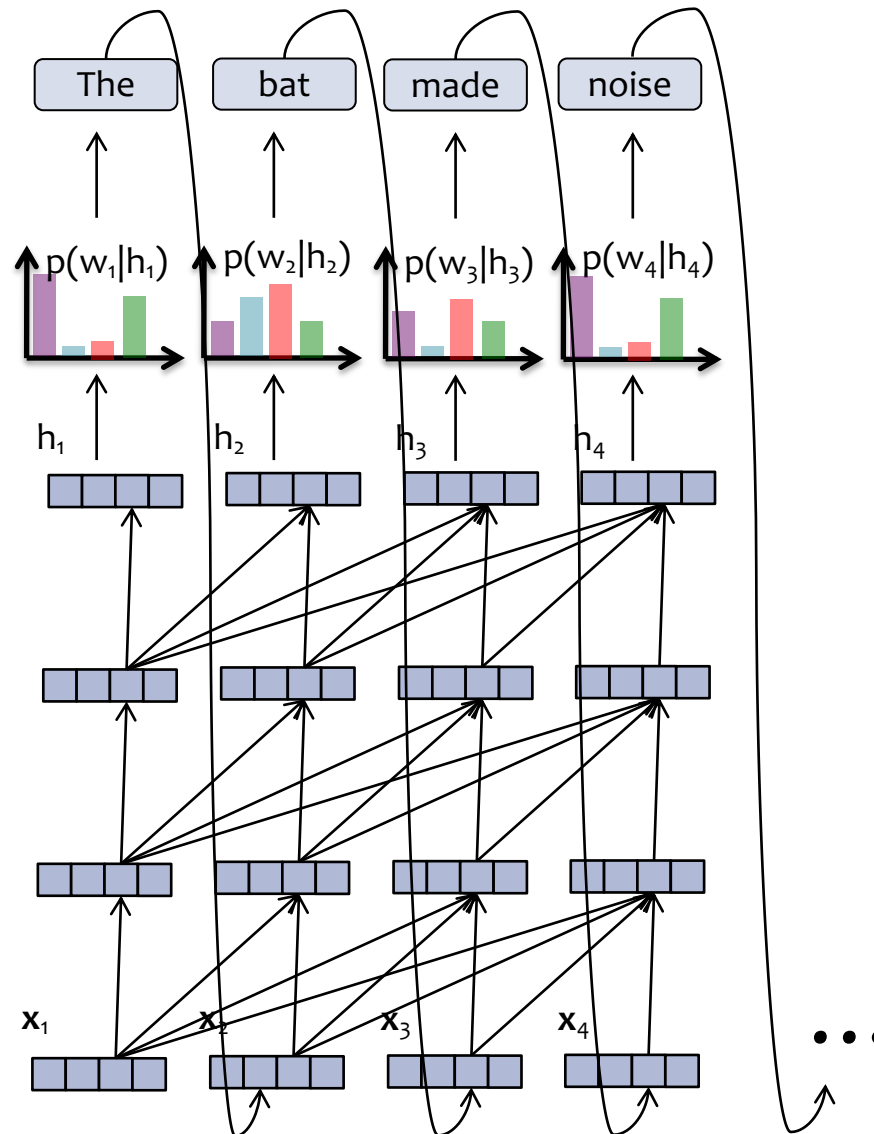
Key Idea:

- (1) convert all previous words to a **fixed length vector**
- (2) define distribution $p(w_t | f_{\theta}(w_{t-1}, \dots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_{\theta}(w_{t-1}, \dots, w_1)$

Transformer Language Model

Important!

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



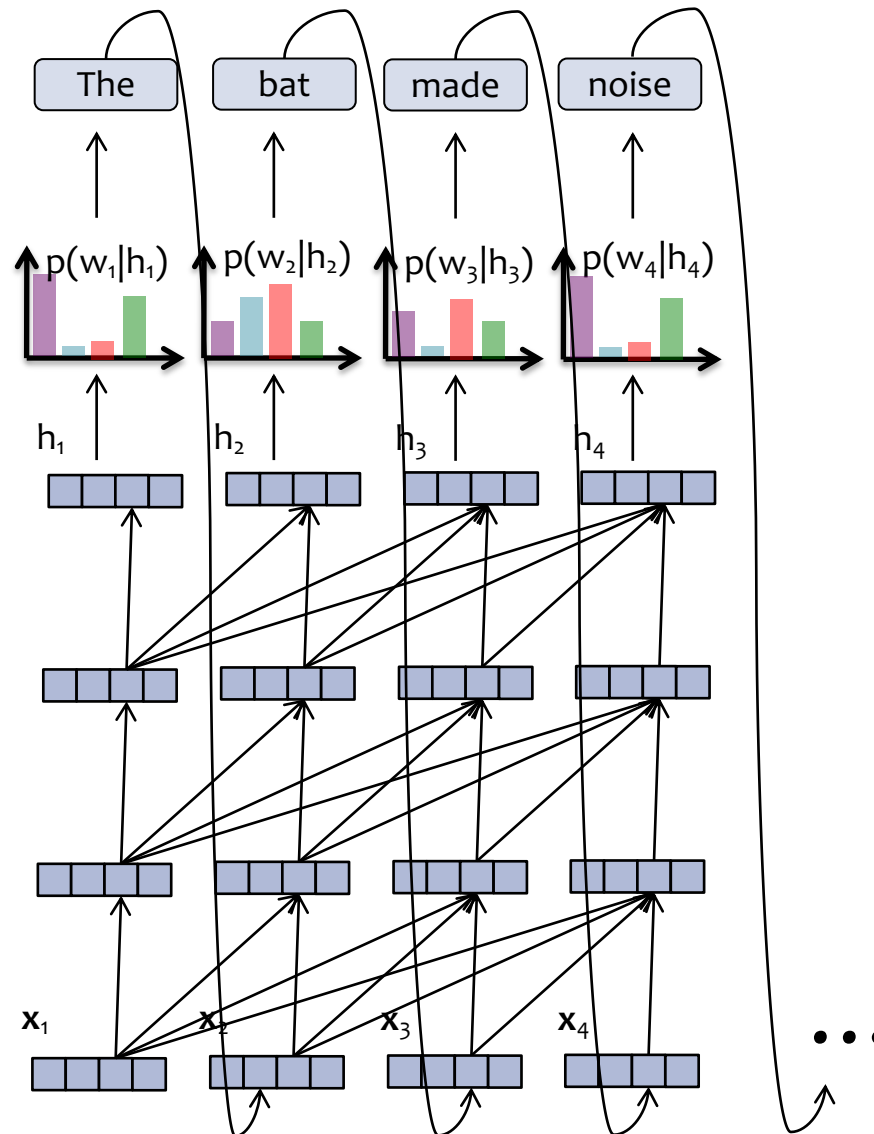
Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

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

Transformer Language Model

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

- *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 $\mathbf{a} \in \mathbb{R}^K$, LayerNorm computes output $\mathbf{b} \in \mathbb{R}^K$:

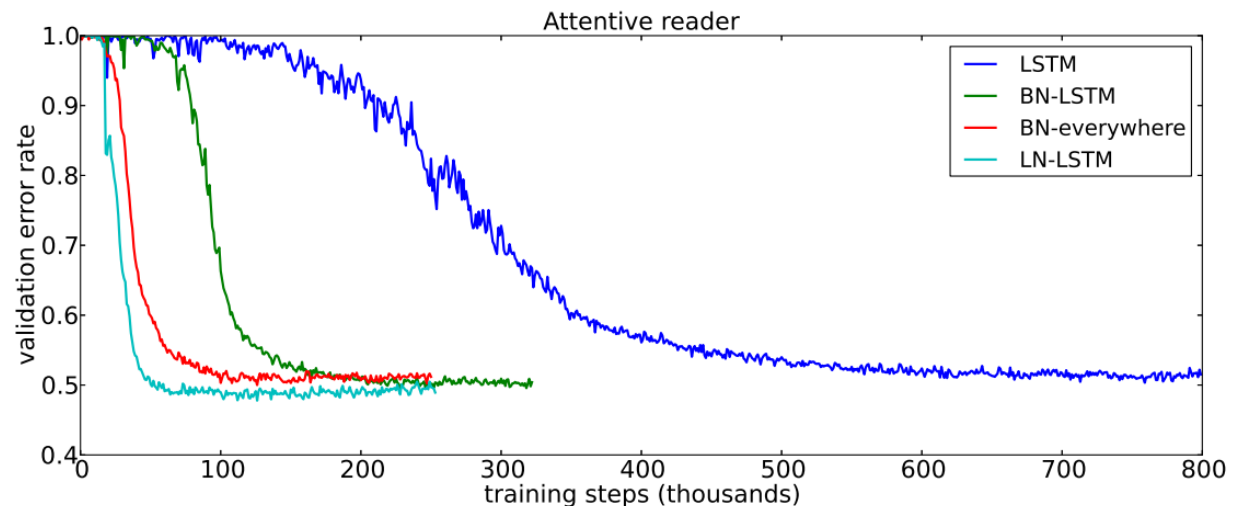
$$\mathbf{b} = \gamma \odot \frac{\mathbf{a} - \mu}{\sigma} \oplus \beta$$

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

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

and parameters $\gamma \in \mathbb{R}^K, \beta \in \mathbb{R}^K$.

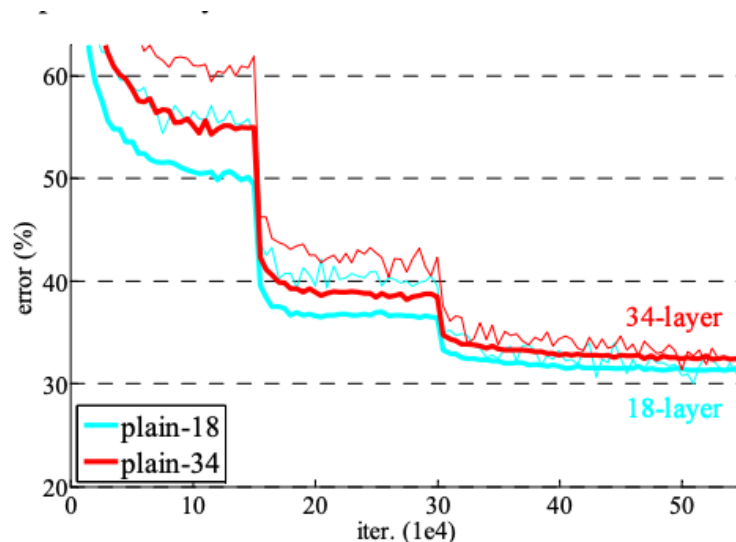
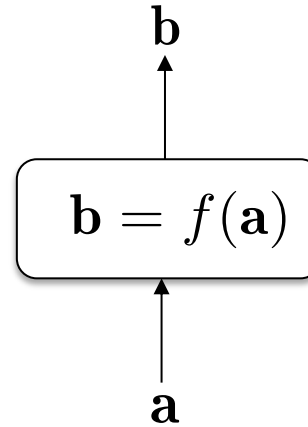
\odot and \oplus denote elementwise multiplication and addition.



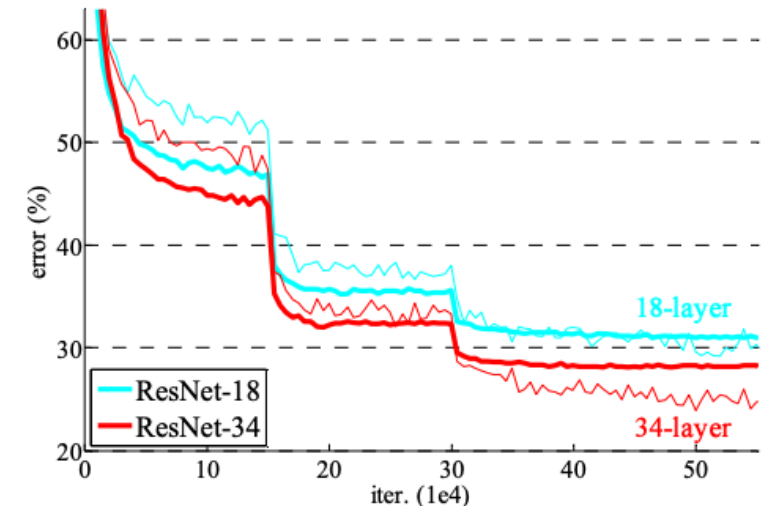
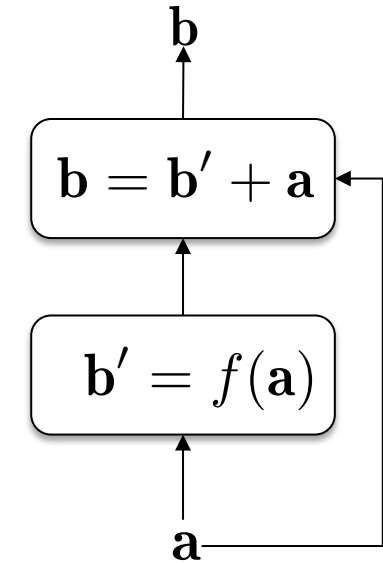
Residual Connections

- *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



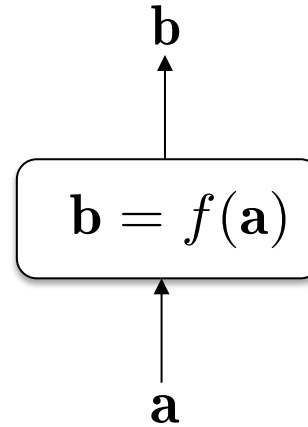
Residual Connection



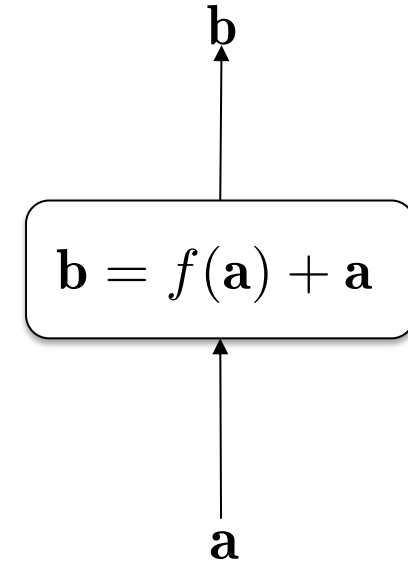
Residual Connections

- *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



Why are residual connections helpful?

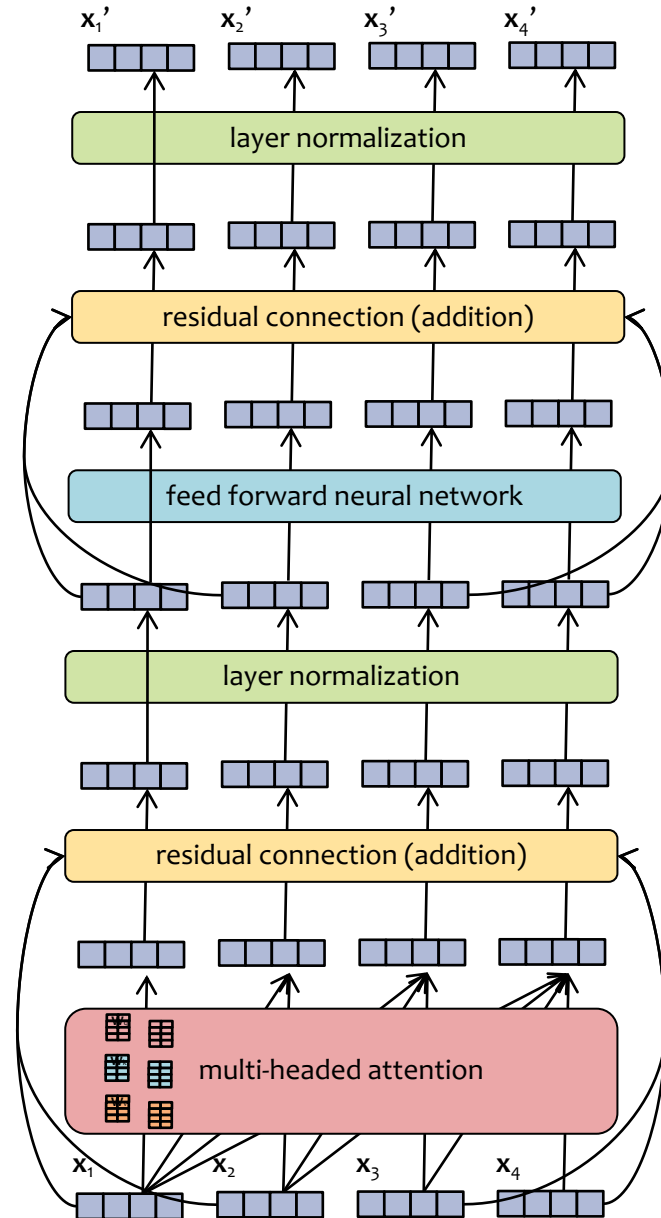
Instead of $f(a)$ having to learn a full transformation of a , $f(a)$ only needs to learn an additive modification of a (i.e. the residual).

Transformer Layer

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.



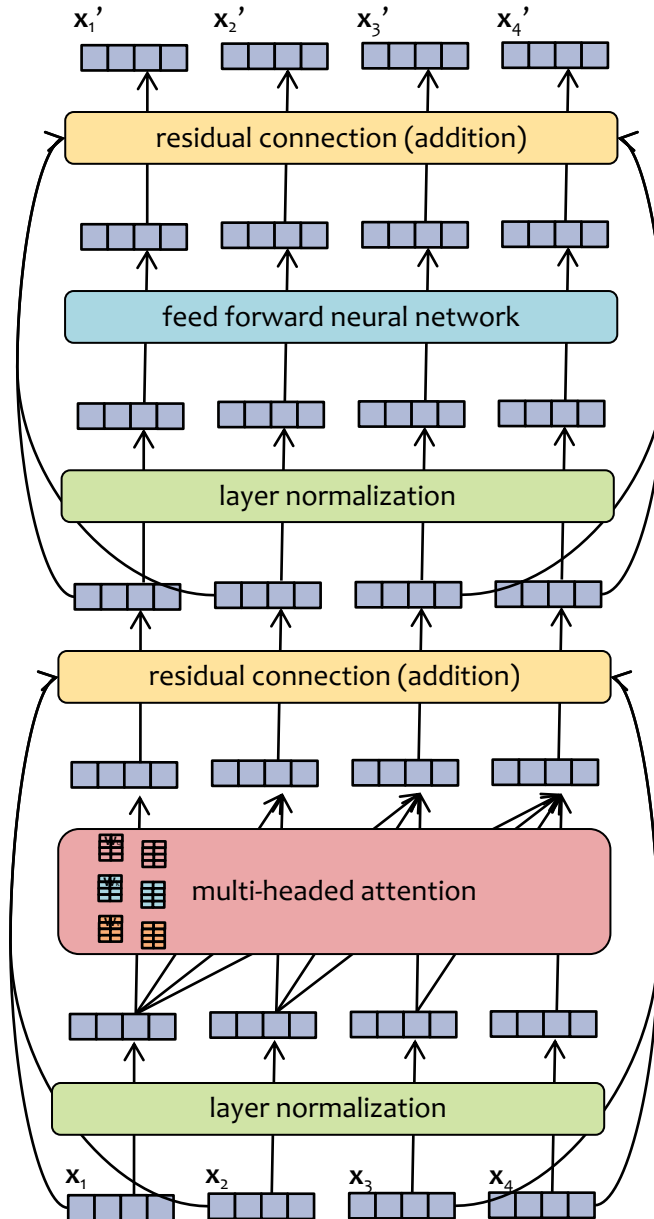
Each layer of a Transformer LM consists of several **sublayers**:

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

Transformer Layer

Pre-LN Version:

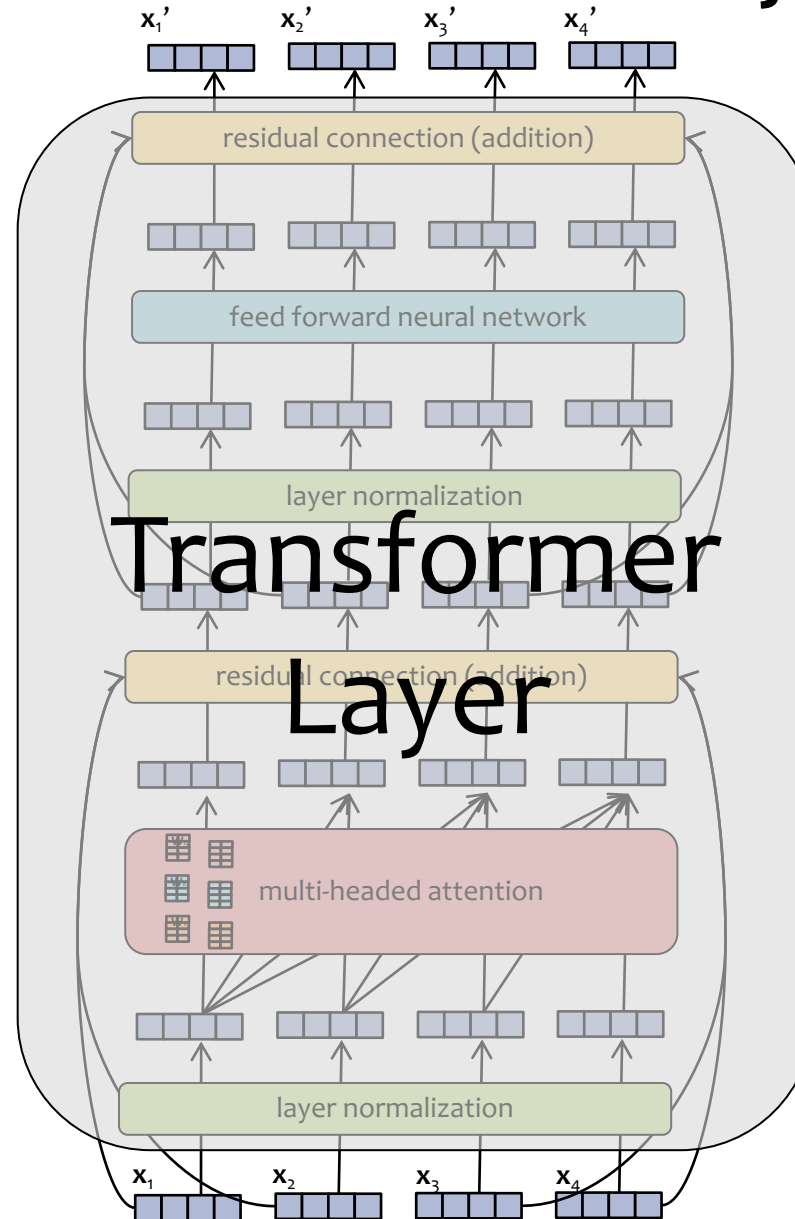
However, subsequent work found that reordering such that the LayerNorm's came at the beginning of each set of 3 layers, the multi-headed attention and feed-forward NN layers tend to be better behaved (i.e. tricks like warm-up are less important).



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

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

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



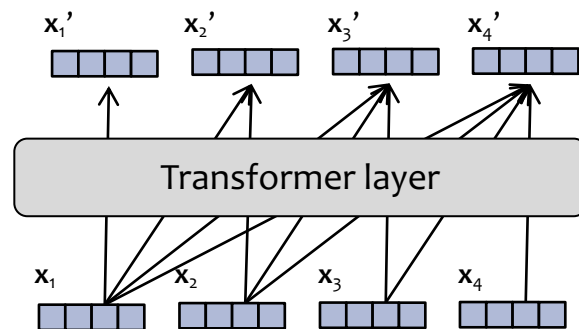
Each layer of a Transformer LM consists of several **sublayers**:

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

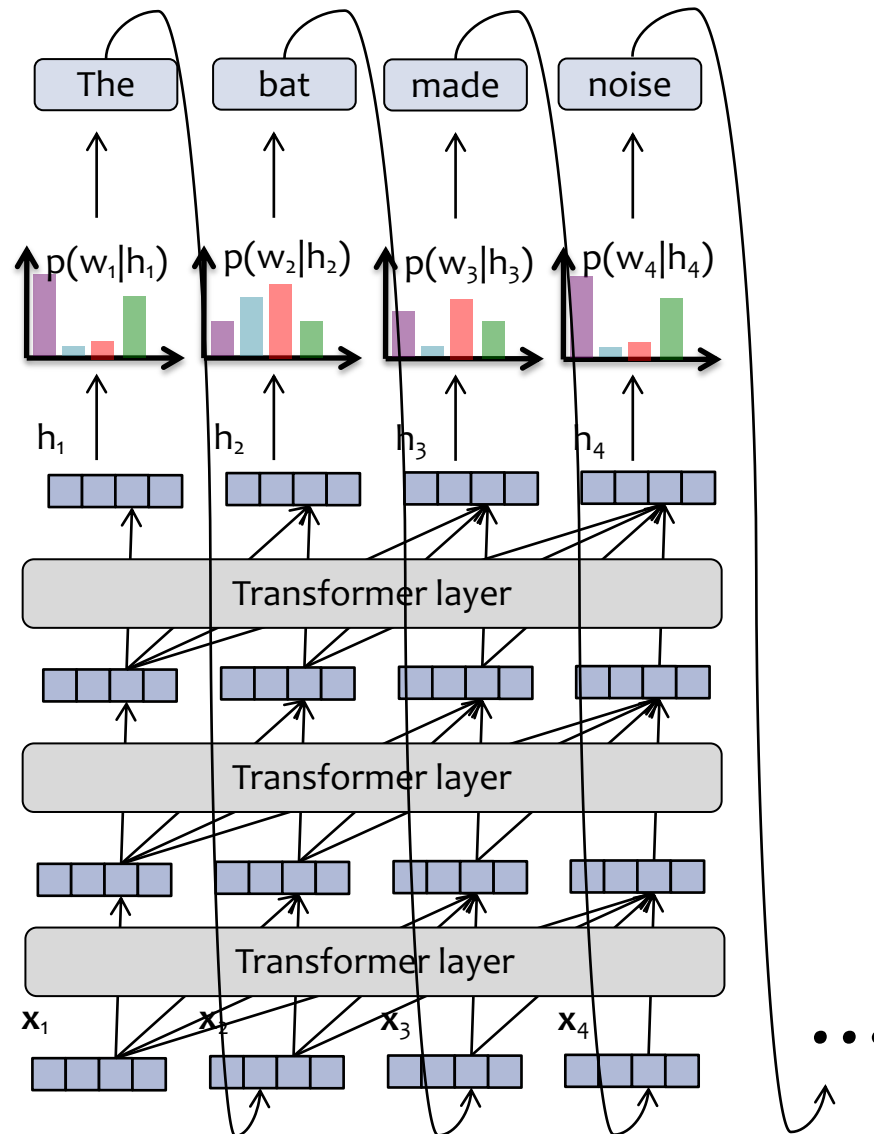
Transformer Layer

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



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:

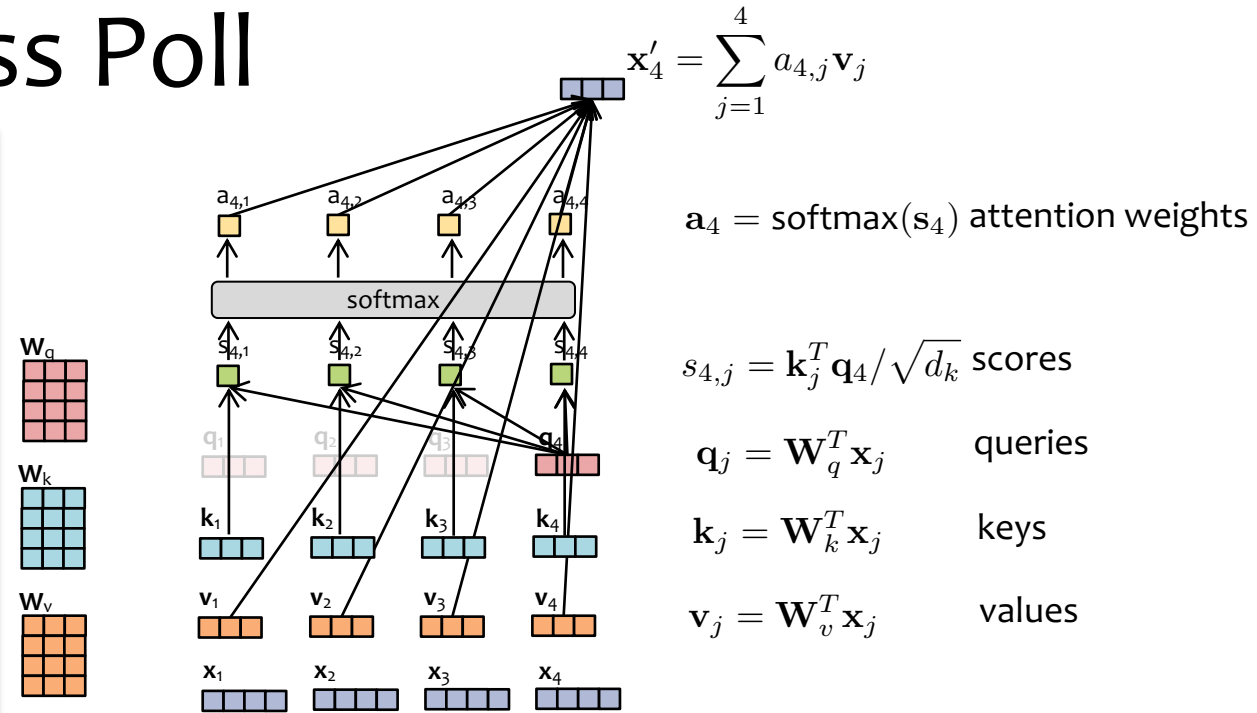
- $x_1 = [1, 0, 0, 0]$ $a_{4,1} = 0.1$
- $x_2 = [0, 1, 0, 0]$ $a_{4,2} = 0.2$
- $x_3 = [0, 0, 2, 0]$ $a_{4,3} = 0.6$
- $x_4 = [0, 0, 0, 1]$ $a_{4,4} = 0.1$

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

Now suppose we swap the embeddings x_2 and x_3 such that

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

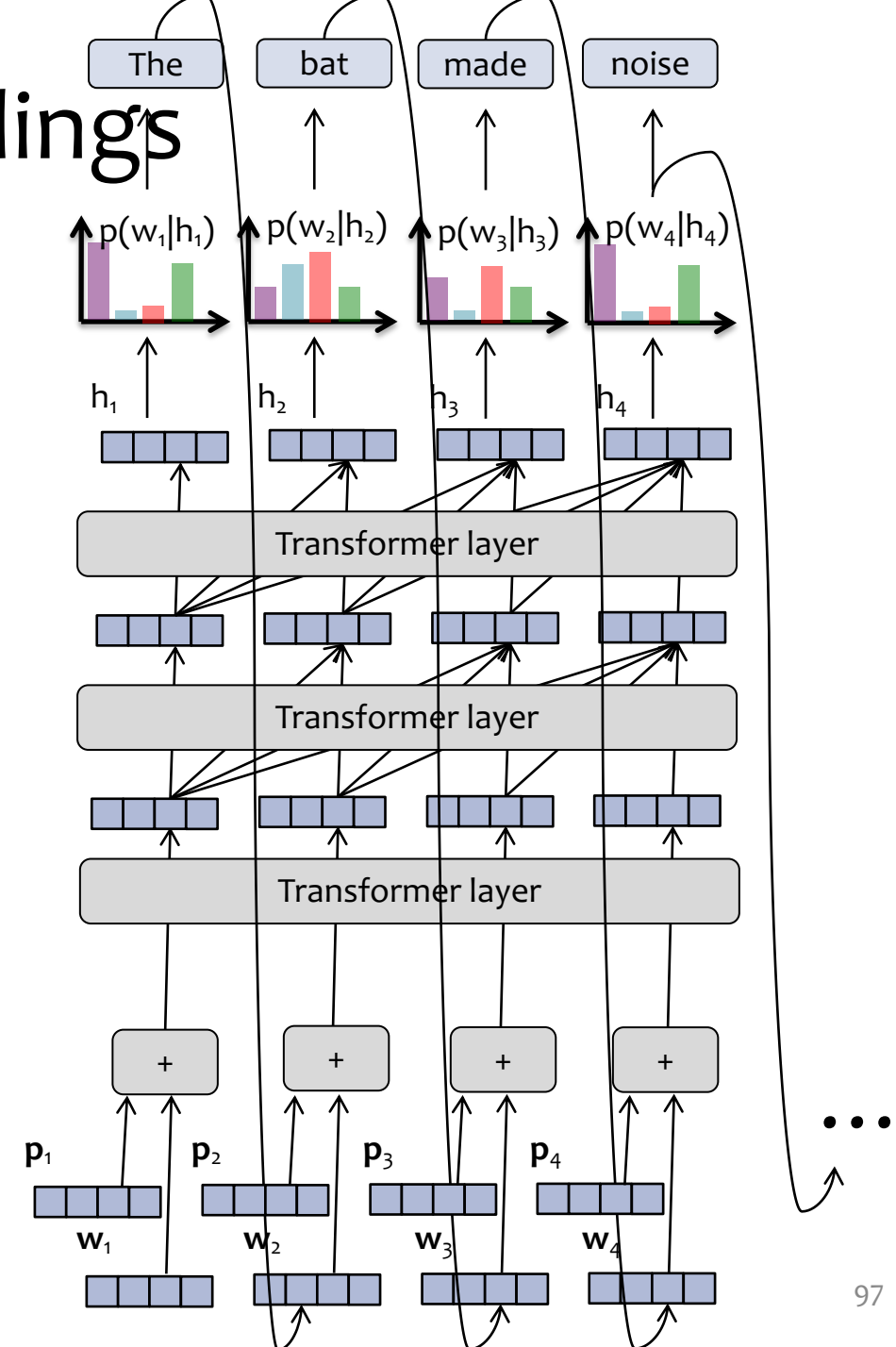
What is the new value of x_4' ?



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)



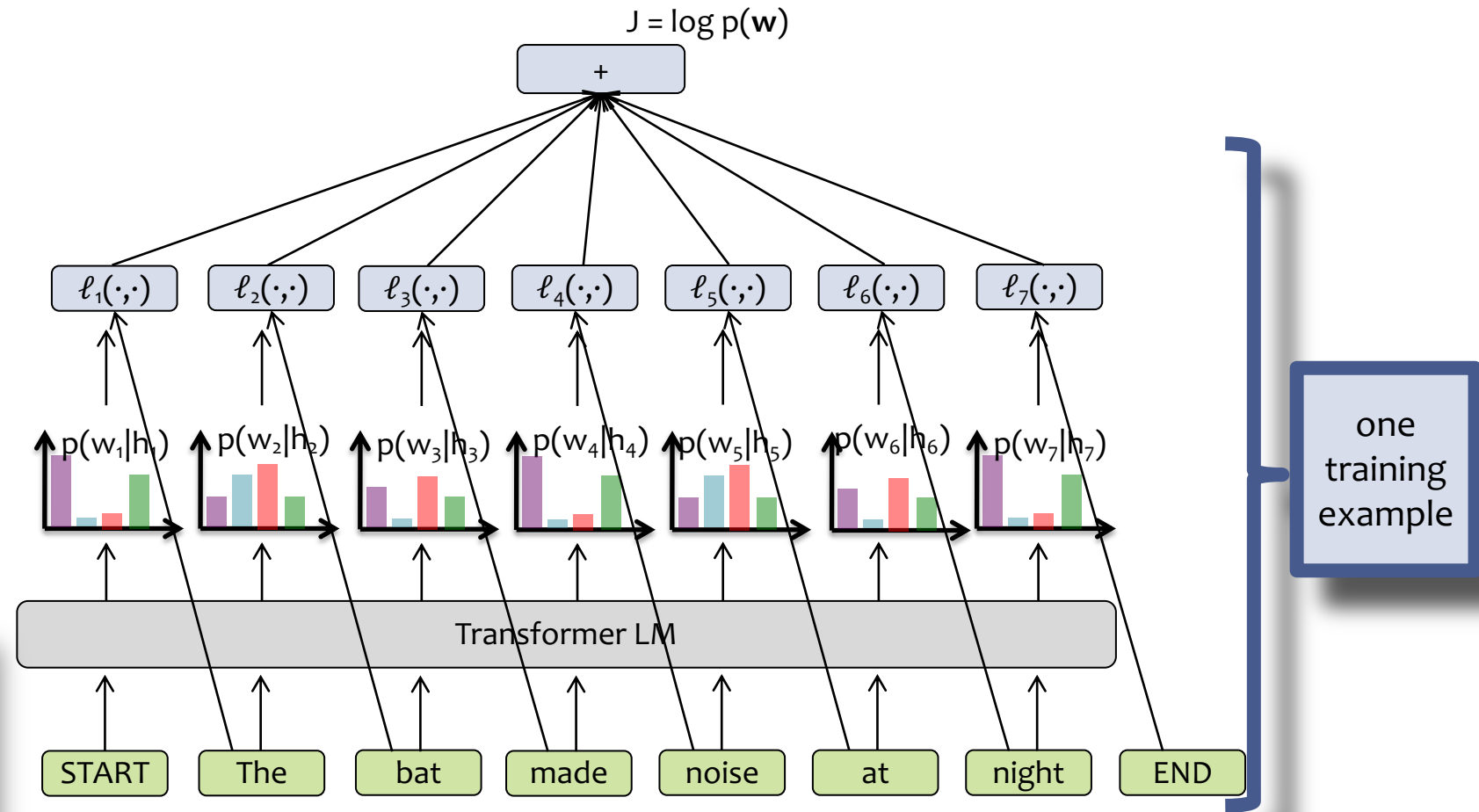
LEARNING A TRANSFORMER LM

Learning a Transformer LM

- Each training example is a sequence (e.g. sentence), so we have training data $D = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \dots, \mathbf{w}^{(N)}\}$
- The objective function for a Deep LM (e.g. RNN-LM or Transformer-LM) is typically the log-likelihood of the training examples:
$$J(\theta) = \sum_i \log p_{\theta}(\mathbf{w}^{(i)})$$
- We train by mini-batch SGD (or your favorite flavor of mini-batch SGD)

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

$$\begin{aligned} \log p(\mathbf{w}) &= \log p(w_1, w_2, w_3, \dots, w_T) \\ &= \log p(w_1 | h_1) + \log p(w_2 | h_2) + \dots + \log p(w_T | h_T) \end{aligned}$$



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

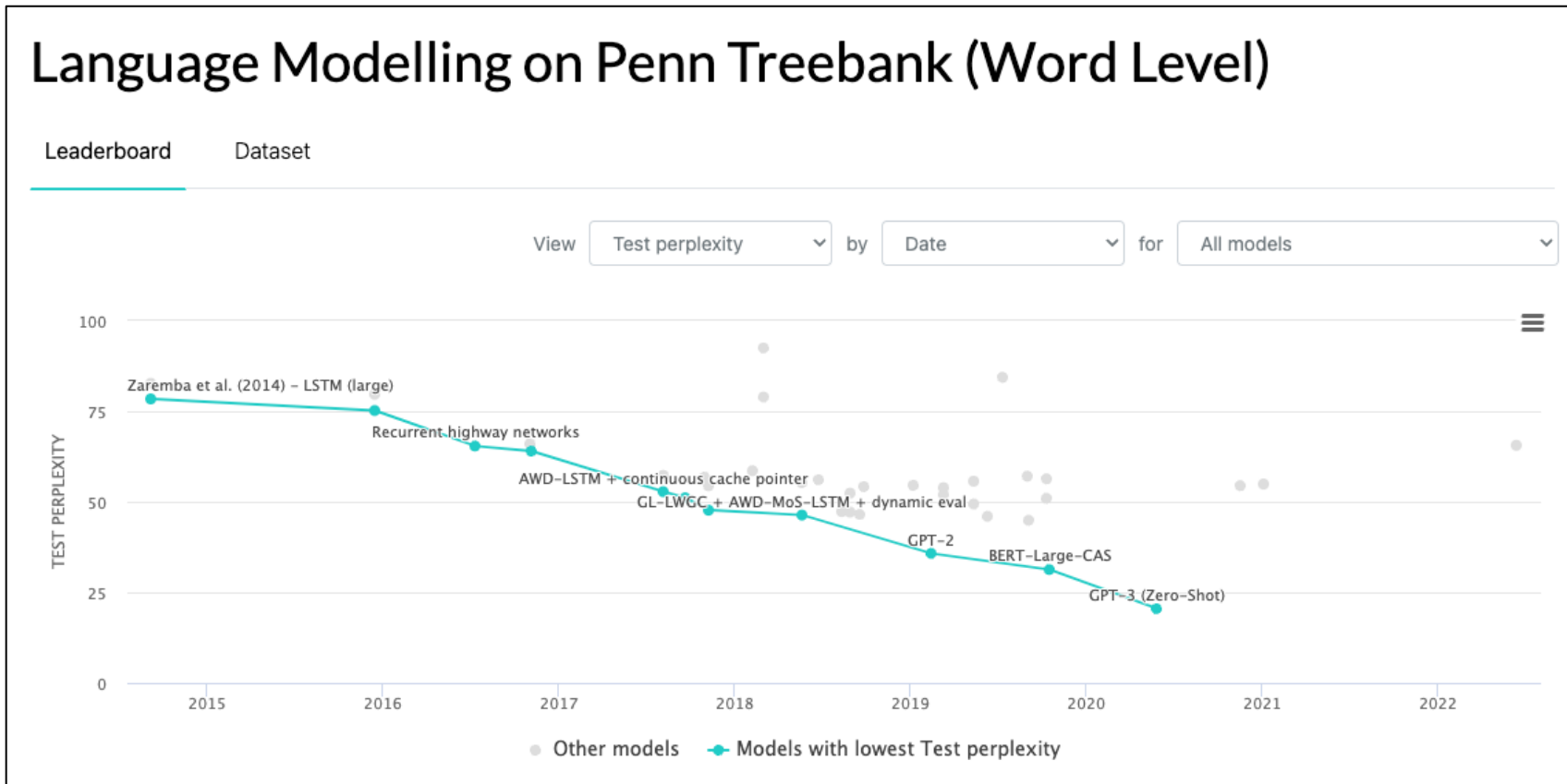


Figure from <https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word>

GPT-3

- 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?

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	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-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.

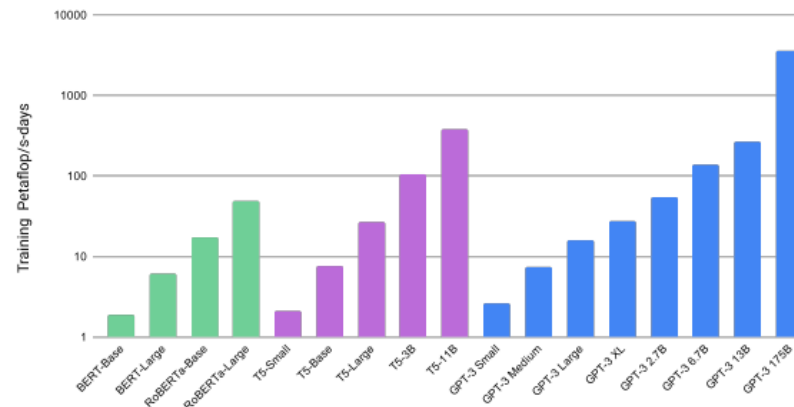


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH⁺20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly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

Recap

Two parts: **Deep Learning** and **Language Modeling**

Deep Learning

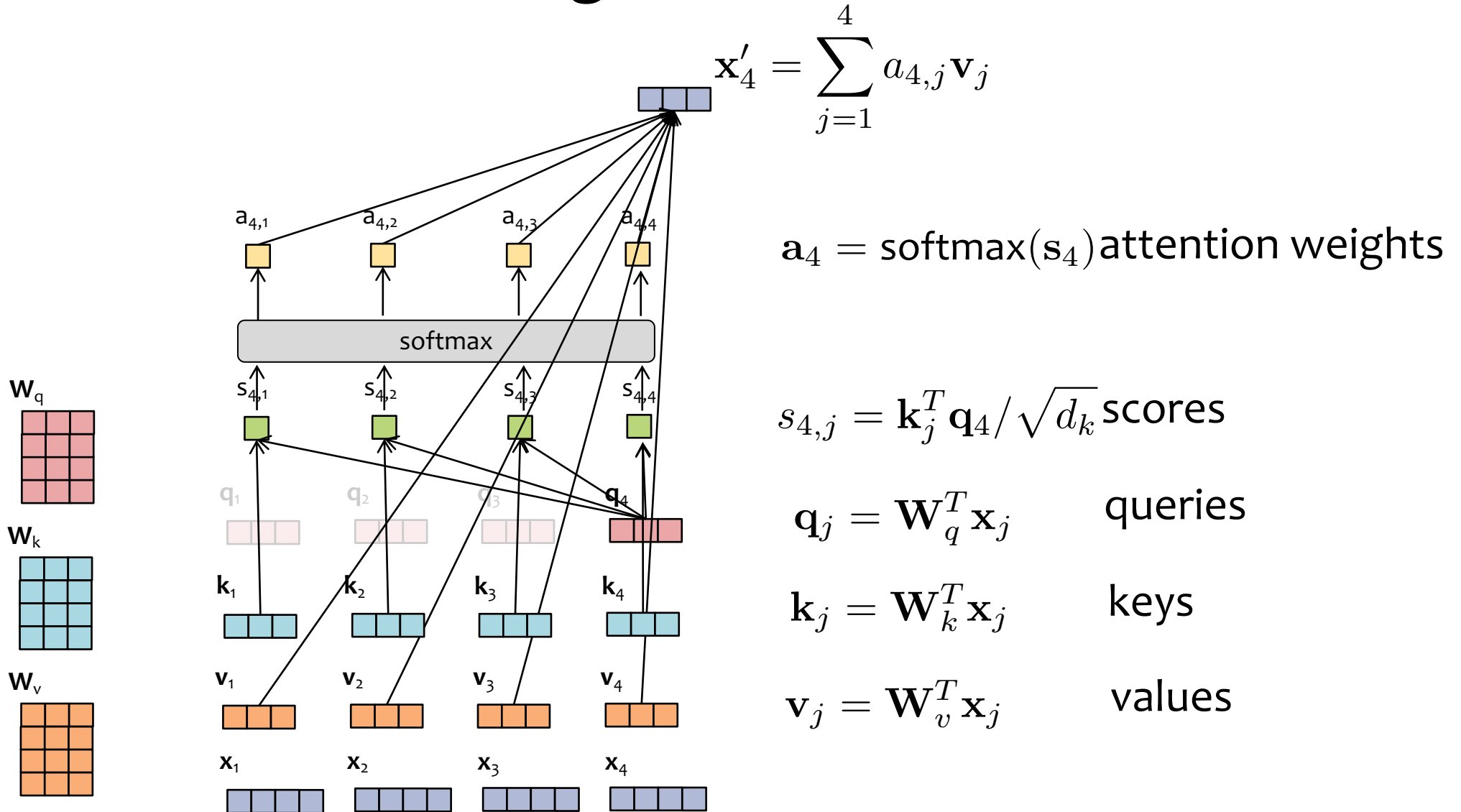
- AutoDiff
 - is a tool for **computing gradients** of a differentiable function, $b = f(a)$
 - the key building block is a **module** with a `forward()` and `backward()`
 - sometimes define f as **code** in `forward()` by chaining existing modules together
- Computation Graphs
 - are another way to define f (more conducive to slides)
 - we are considering various (deep) computation graphs: (1) CNN (2) RNN (3) RNN-LM (4) Transformer-LM
- Learning a Deep Network
 - deep networks (e.g. CNN/RNN) are trained by optimizing an objective function with SGD
 - compute gradients with AutoDiff

Language Modeling

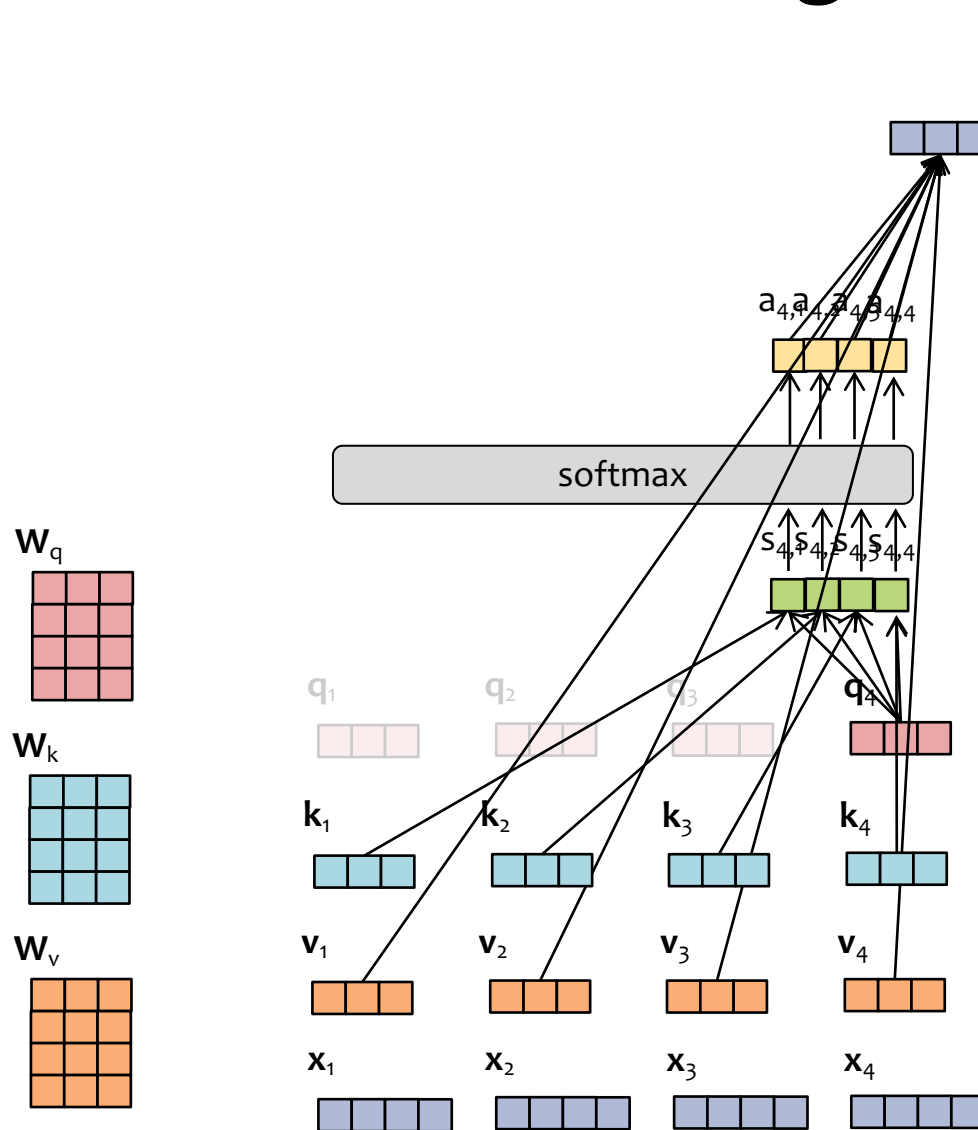
- key idea: condition on previous words to **sample the next word**
- to define the **probability** of the next word...
 - ...n-gram LM uses collection of massive 50k-sided **dice**
 - ...RNN-LM or Transformer-LM use a **neural network**
- Learning an LM
 - n-gram LMs are easy to learn: just **count** co-occurrences!
 - a RNN-LM / Transformer-LM is trained just like other deep neural networks

IMPLEMENTING A TRANSFORMER LM

Matrix Version of Single-Headed Attention



Matrix Version of Single-Headed Attention



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

$\mathbf{a}_4 = \text{softmax}(s_4)$ attention weights

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

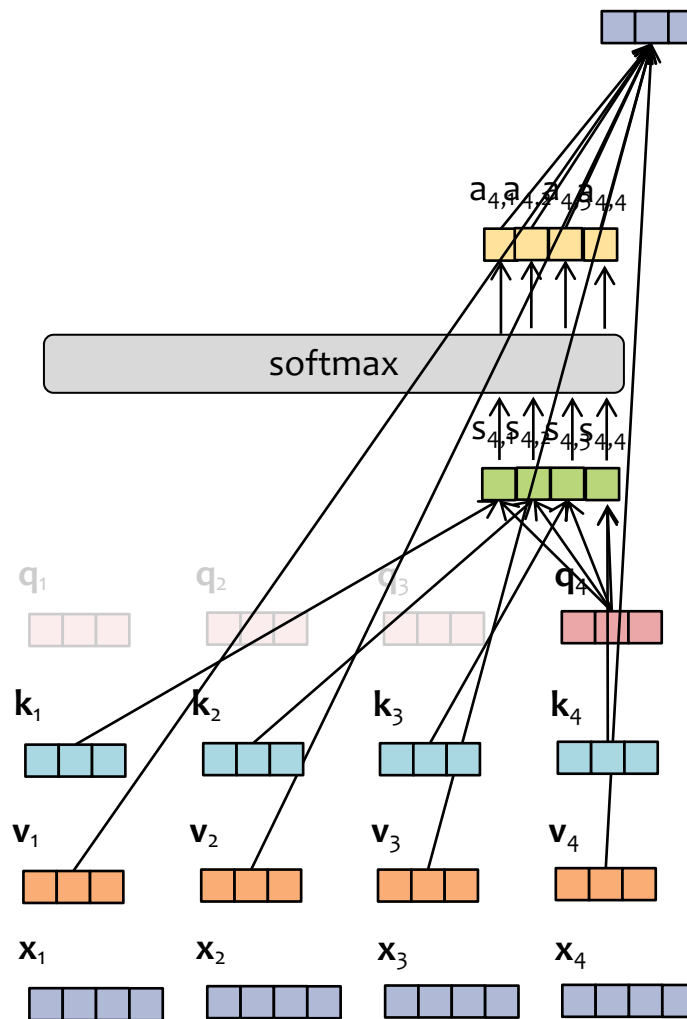
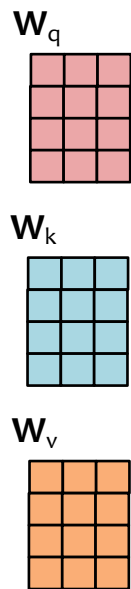
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

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



$$\mathbf{X}' = \mathbf{A}\mathbf{V} = \text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d_k})\mathbf{V}$$

$$\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_4]^T = \text{softmax}(\mathbf{S})$$

$$\mathbf{S} = [\mathbf{s}_1, \dots, \mathbf{s}_4]^T = \mathbf{Q}\mathbf{K}^T / \sqrt{d_k}$$

$$\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_4]^T = \mathbf{X}\mathbf{W}_q$$

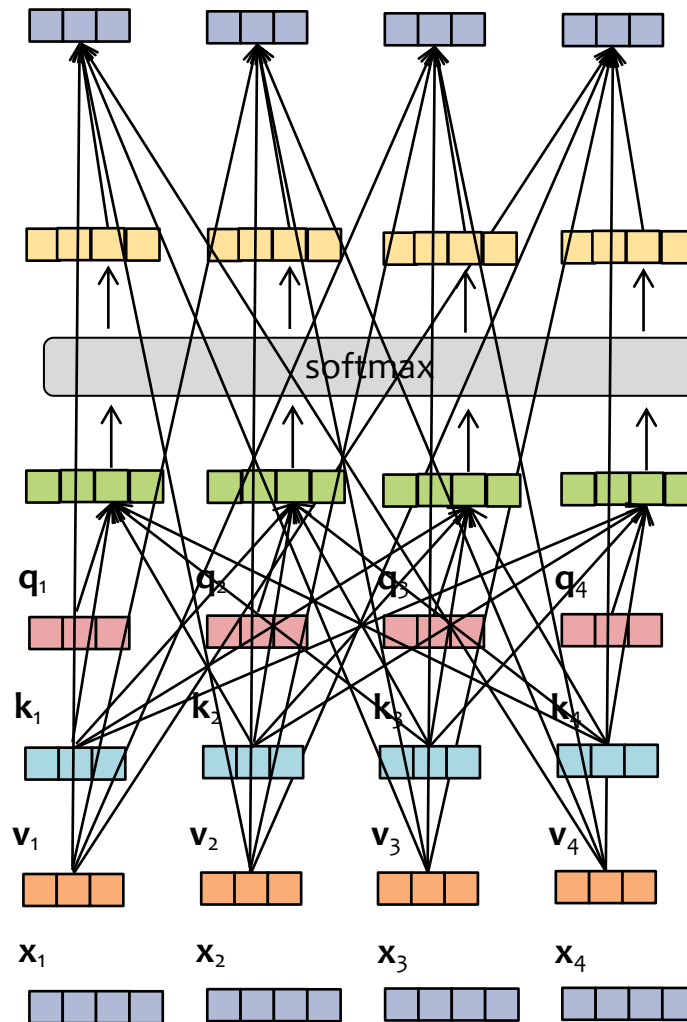
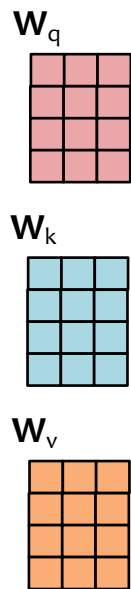
$$\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_4]^T = \mathbf{X}\mathbf{W}_k$$

$$\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_4]^T = \mathbf{X}\mathbf{W}_v$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$

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



$$\mathbf{X}' = \mathbf{A}\mathbf{V} = \text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d_k})\mathbf{V}$$

$$\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_4]^T = \text{softmax}(\mathbf{S})$$

$$\mathbf{S} = [\mathbf{s}_1, \dots, \mathbf{s}_4]^T = \mathbf{Q}\mathbf{K}^T / \sqrt{d_k}$$

$$\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_4]^T = \mathbf{X}\mathbf{W}_q$$

$$\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_4]^T = \mathbf{X}\mathbf{W}_k$$

$$\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_4]^T = \mathbf{X}\mathbf{W}_v$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$

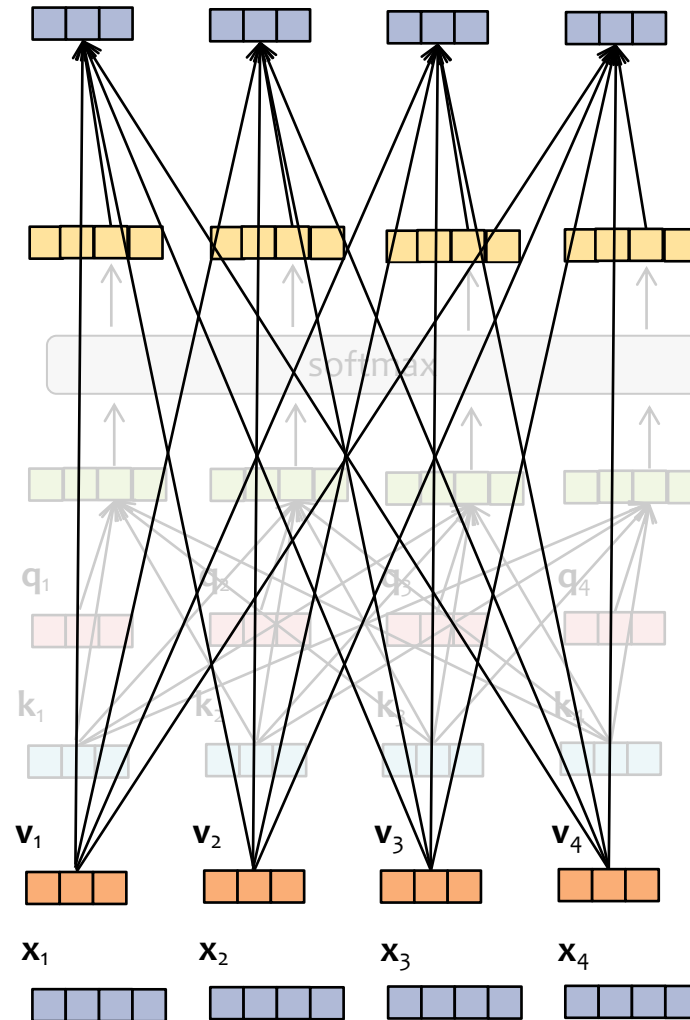
Matrix Version of Single-Headed Attention

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!



$$\mathbf{X}' = \mathbf{A}\mathbf{V} = \text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d_k})\mathbf{V}$$

$$\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_4]^T = \text{softmax}(\mathbf{S})$$

$$\mathbf{S} = [\mathbf{s}_1, \dots, \mathbf{s}_4]^T = \mathbf{Q}\mathbf{K}^T / \sqrt{d_k}$$

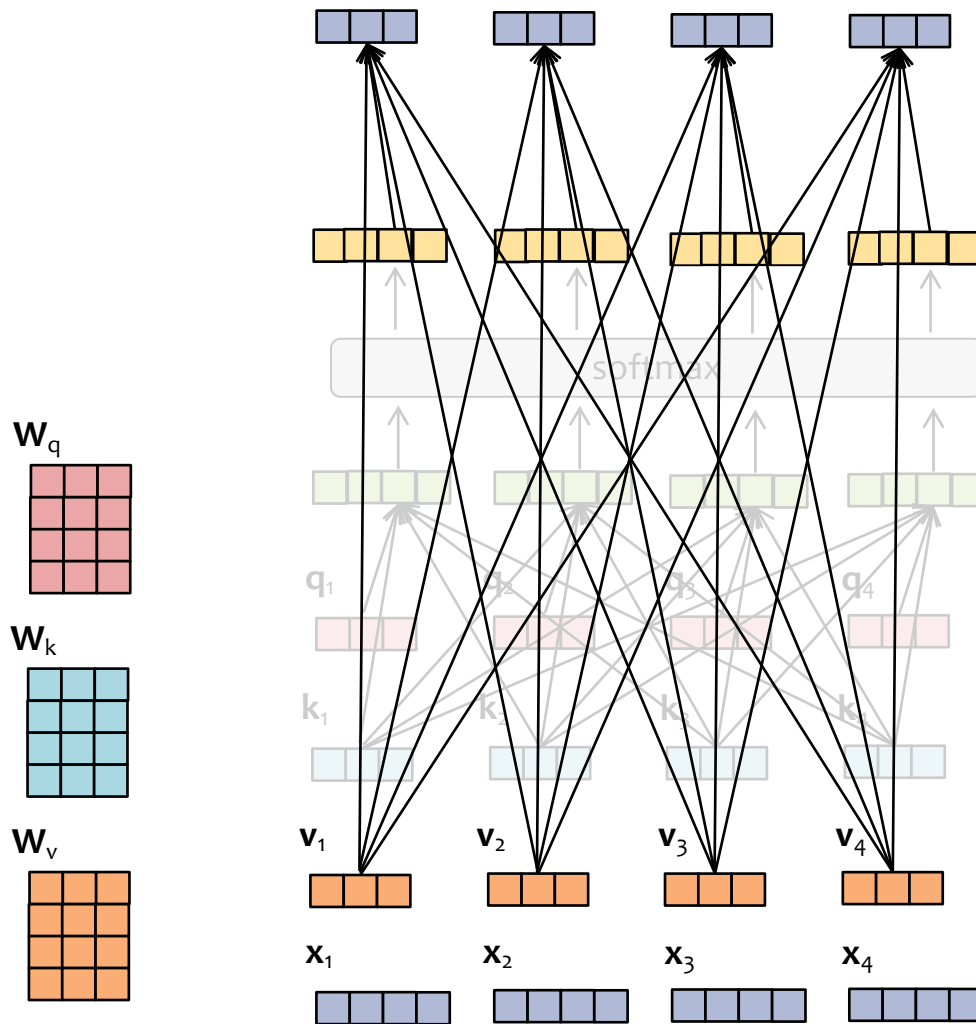
$$\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_4]^T = \mathbf{X}\mathbf{W}_q$$

$$\mathbf{K} = [\mathbf{k}_1, \dots, \mathbf{k}_4]^T = \mathbf{X}\mathbf{W}_k$$

$$\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_4]^T = \mathbf{X}\mathbf{W}_v$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$

Matrix Version of Single-Headed Attention



$$\mathbf{X}' = \mathbf{A}\mathbf{V} = \text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d_k})\mathbf{V}$$

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

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T / \sqrt{d_k}$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}_k$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_v$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$

Question: How is the softmax applied?

- A. column-wise
- B. row-wise

Answer:

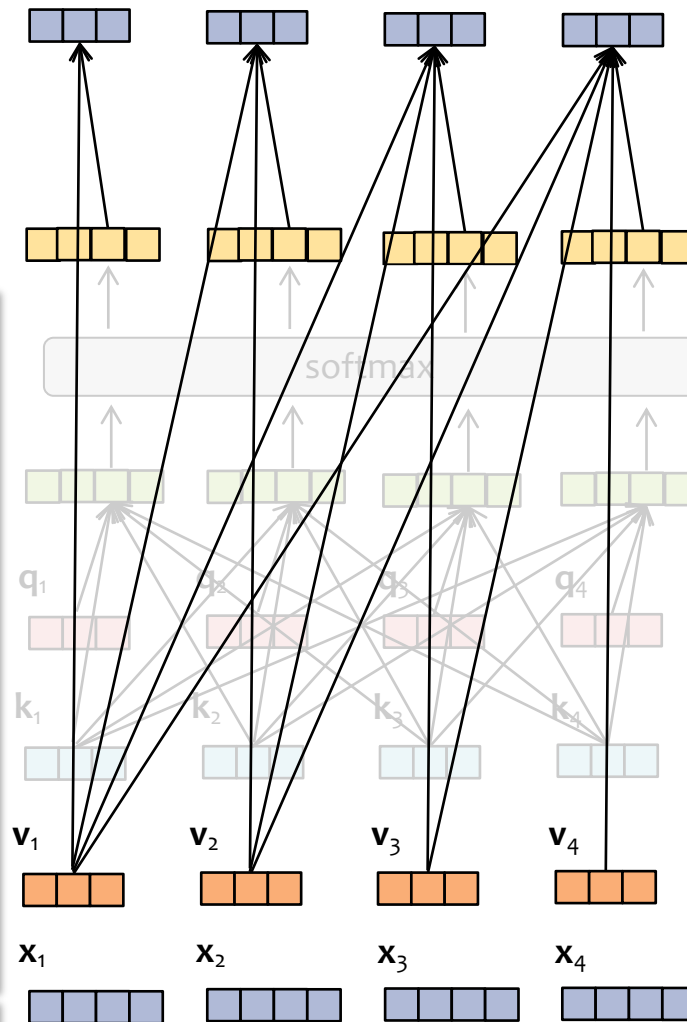
Matrix Version of Single-Headed (Causal) Attention

Insight: if some element in the input to the softmax is $-\infty$, then the corresponding output is 0!

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

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

Answer:



$$\mathbf{X}' = \mathbf{A}\mathbf{V} = \text{softmax}(\mathbf{Q}\mathbf{K}^T / \sqrt{d_k} + \mathbf{M})\mathbf{V}$$

$$\mathbf{A}_{\text{causal}} = \text{softmax}(\mathbf{S} + \mathbf{M})$$

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T / \sqrt{d_k}$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q$$

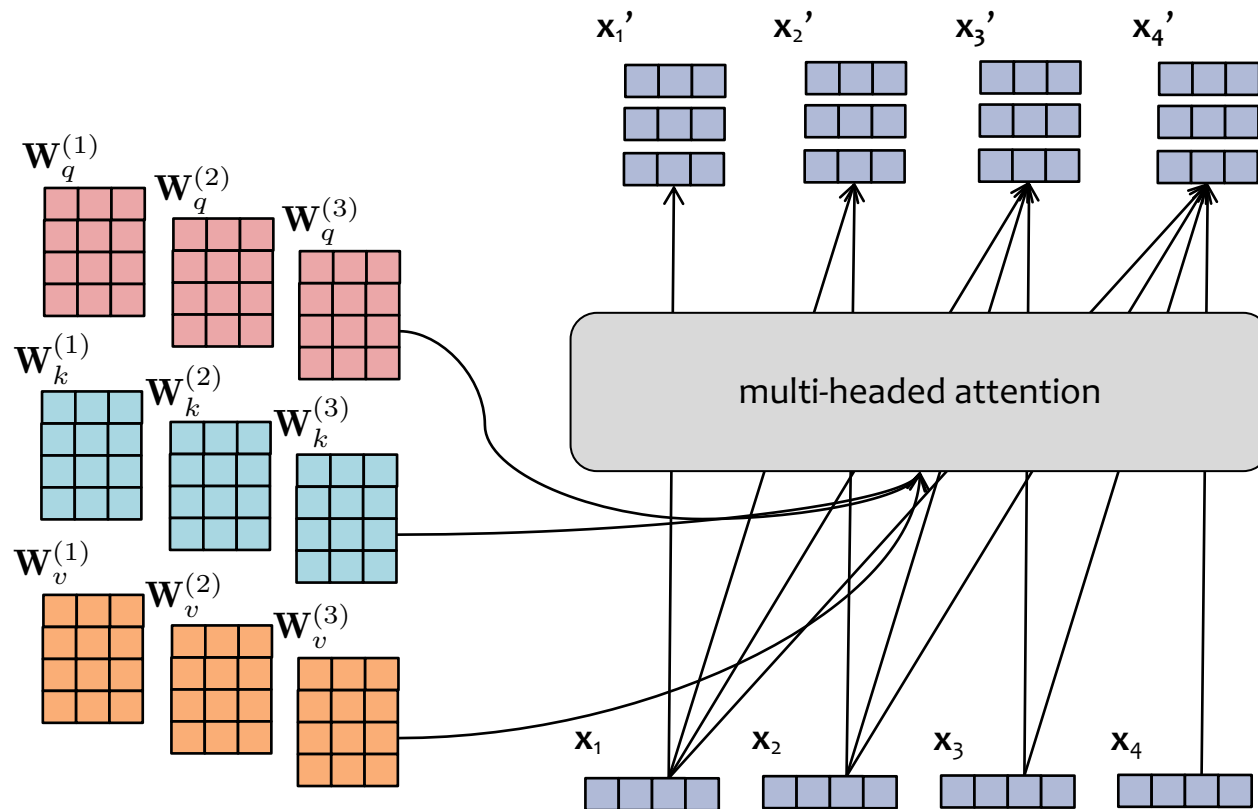
$$\mathbf{K} = \mathbf{X}\mathbf{W}_k$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_v$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$

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

Matrix Version of Multi-Headed (Causal) Attention



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

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

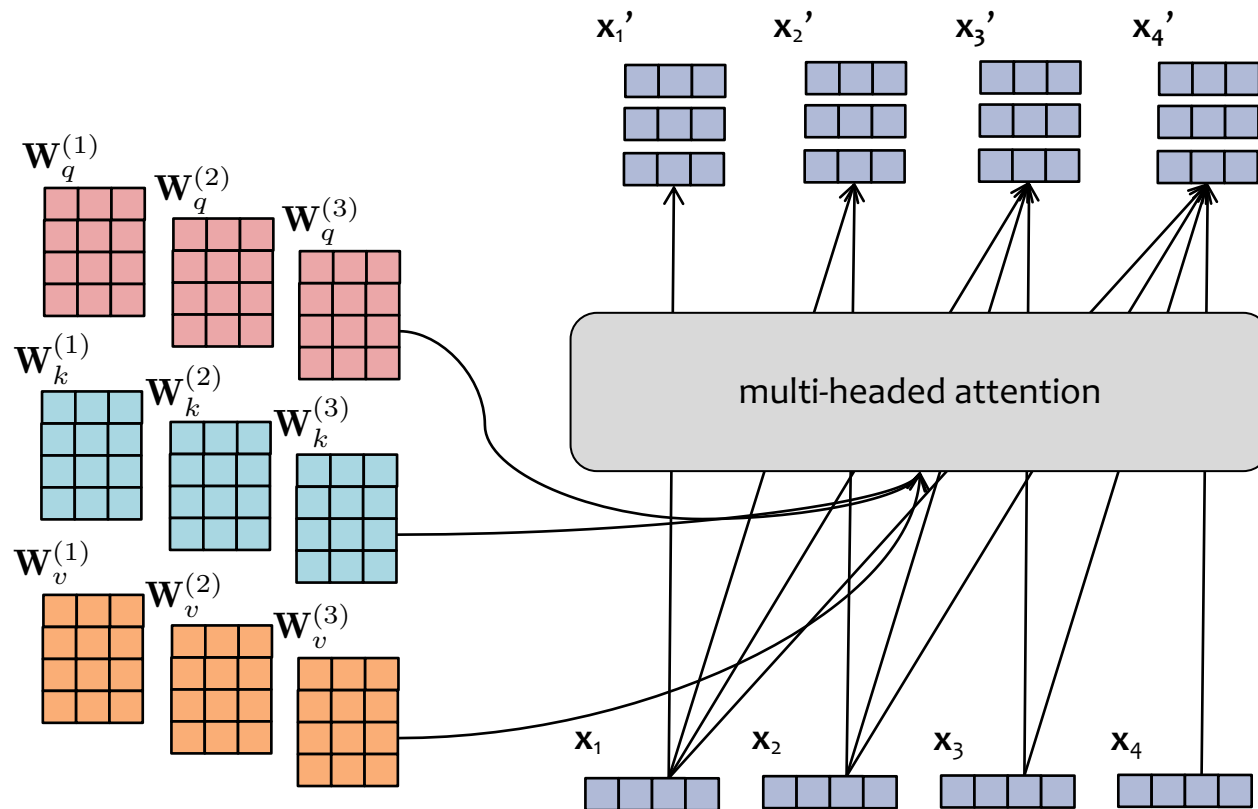
$$\mathbf{Q}^{(i)} = \mathbf{X} \mathbf{W}_q^{(i)}$$

$$\mathbf{K}^{(i)} = \mathbf{X} \mathbf{W}_k^{(i)}$$

$$\mathbf{V}^{(i)} = \mathbf{X} \mathbf{W}_v^{(i)}$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$

Matrix Version of Multi-Headed (Causal) Attention



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

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

$$\mathbf{Q}^{(i)} = \mathbf{X} \mathbf{W}_q^{(i)}$$

$$\mathbf{K}^{(i)} = \mathbf{X} \mathbf{W}_k^{(i)}$$

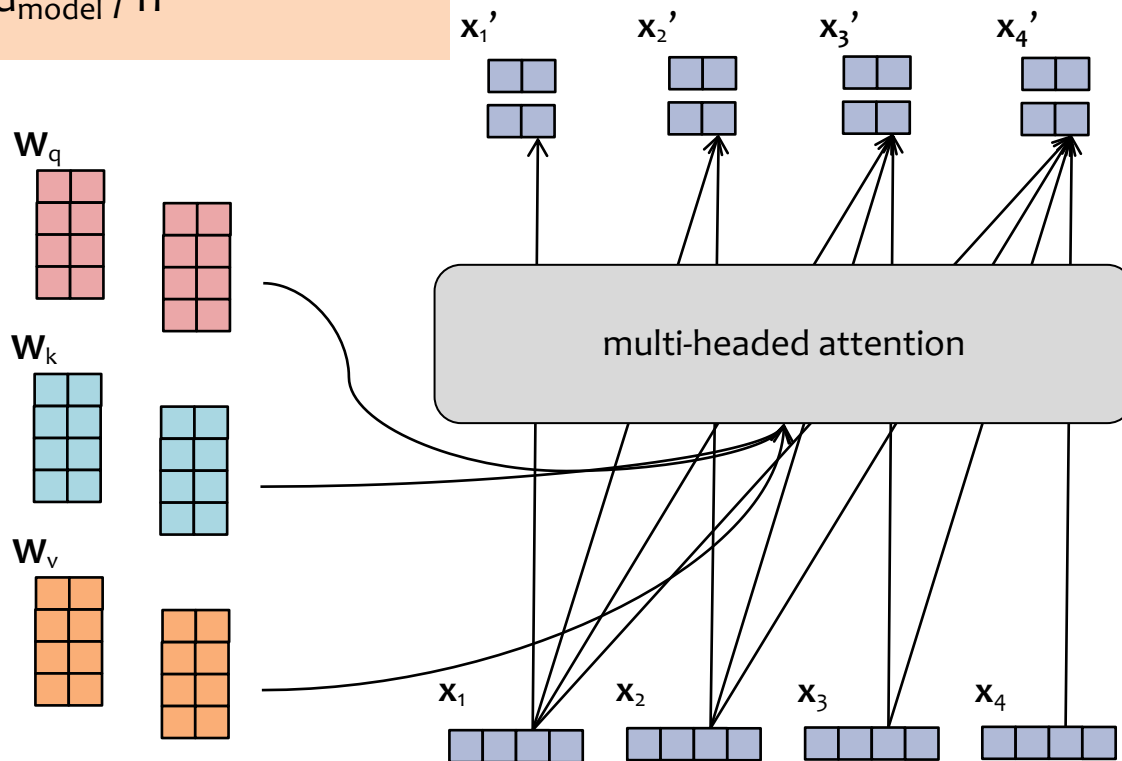
$$\mathbf{V}^{(i)} = \mathbf{X} \mathbf{W}_v^{(i)}$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$

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:

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



Construction of Multi-Headed (Causal) Attention

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

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

$$\mathbf{Q}^{(i)} = \mathbf{X} \mathbf{W}_q^{(i)}$$

$$\mathbf{K}^{(i)} = \mathbf{X} \mathbf{W}_k^{(i)}$$

$$\mathbf{V}^{(i)} = \mathbf{X} \mathbf{W}_v^{(i)}$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$