CS2109S: Introduction to AI and Machine Learning

Lecture 11:
Neural Networks on Sequential Data

8 April 2025

PollEv.com/conghuihu365

Outline

- Recurrent Neural Networks
  - Motivation
  - Recurrent Neural Networks
  - Applications
- Self-Attention
  - Self-Attention Layer
  - Positional Encoding
- Issues with Deep Learning
  - Overfitting
  - Vanishing/Exploding Gradient

Outline

- **Recurrent Neural Networks**
  - Motivation
  - Recurrent Neural Networks
  - Applications
- Self-Attention
  - Self-Attention Layer
  - Positional Encoding
- Issues with Deep Learning
  - Overfitting
  - Vanishing/Exploding Gradient

Sequential Data

Sequential data refers to data where the order of elements matters, and each element depends on its position in the sequence.

Text data:

“we saw this saw”

Audio data:

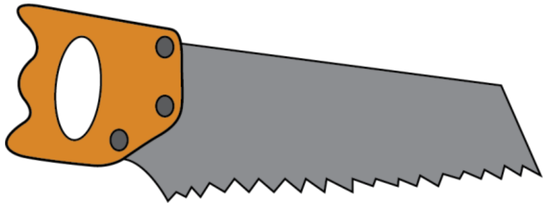
Video data:

Image credit:
https://towardsdatascience.com/all-you-need-to-know-to-start-speech-processing-with-deep-learning-102c916edf62
https://openaccess.thecvf.com/content/ICCV2021/papers/Arnab_ViViT_A_Video_Vision_Transformer_ICCV_2021_paper.pdf

Sequential Data

Text data: "we saw this saw"

	we	saw	this	saw
	Verb?	Verb?	Verb?	Verb?
True Label:	No	Yes	No	No



One-hot Encoding:

- For each word, create a vector with length equal to the vocabulary size.
- Each word is assigned a unique index, and its corresponding vector is all zeros except for a 1 in the position of that index.

Word	Index	One-hot encoded vector
apple	0	[1, 0, 0, 0]
we	1	[0, 1, 0, 0]
this	2	[0, 0, 1, 0]
saw	3	[0, 0, 0, 1]

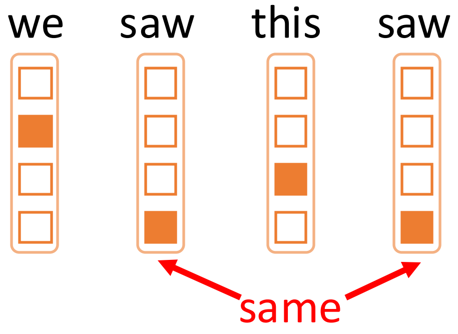
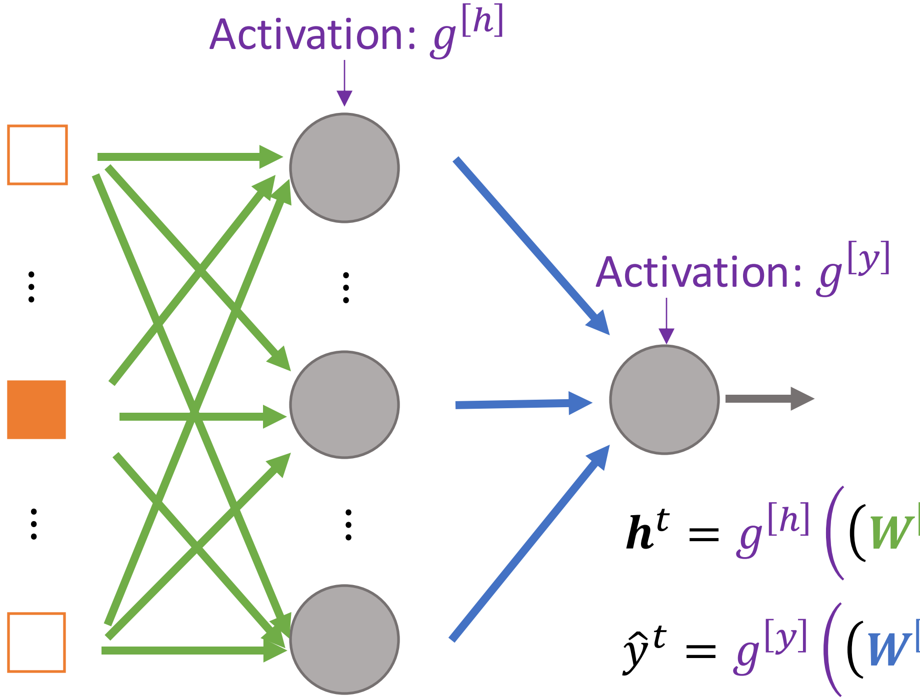


Image credit: https://www.cleanpng.com/png-chainsaw-clip-art-hammer-cartoon-4747426/4.html

Sequential Data: A Naïve Attempt

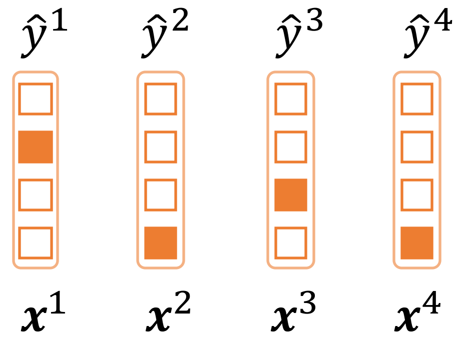
Text data: “we saw this saw”

	we	saw	this	saw
	Verb?	Verb?	Verb?	Verb?
True Label:	No	Yes	No	No



$$\boldsymbol{h}^t = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^t\right)$$

$$\hat{y}^t = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^\top \boldsymbol{h}^t\right)$$



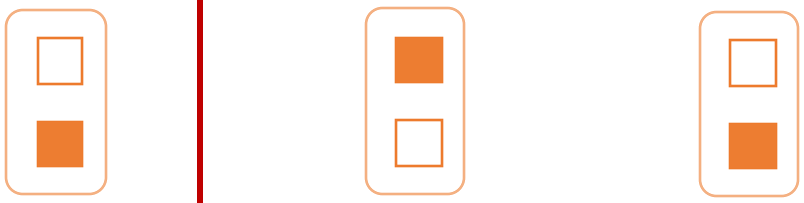
$\boldsymbol{x}^2$ and $\boldsymbol{x}^4$ are both one-hot vectors representing “saw”.

- $\boldsymbol{x}^2 = \boldsymbol{x}^4$
- $\hat{y}^2 = \hat{y}^4$
- For the two “saw”, one of the predicted labels must be incorrect.
- We should not predict each label independently.

Obtain contextual information by using RNN!

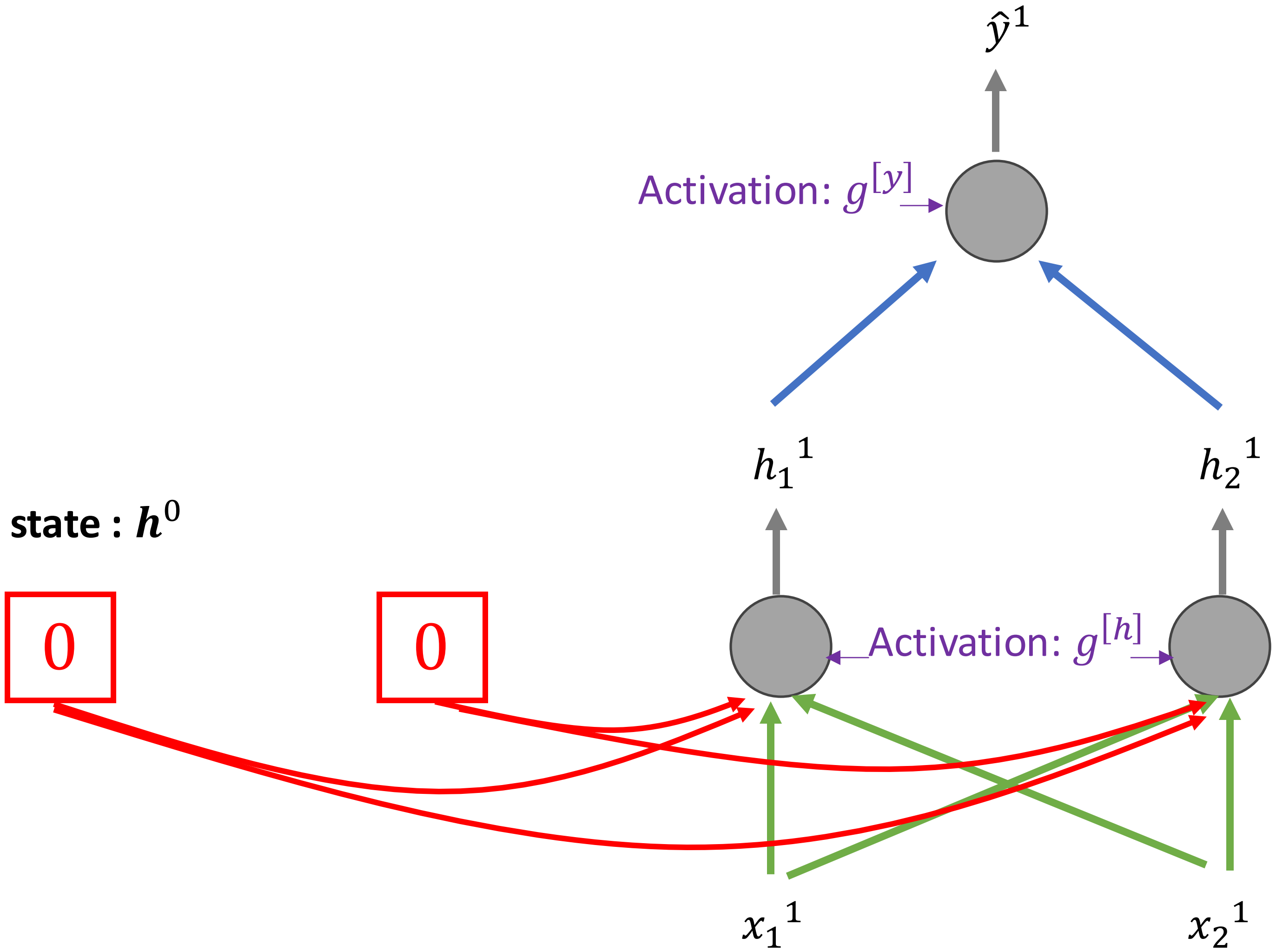
Recurrent Neural Network

A sequence with 3 elements



$\boldsymbol{x}^1 = \begin{bmatrix} {x_1}^1 \\ {x_2}^1 \end{bmatrix}$ $\boldsymbol{x}^2 = \begin{bmatrix} {x_1}^2 \\ {x_2}^2 \end{bmatrix}$ $\boldsymbol{x}^3 = \begin{bmatrix} {x_1}^3 \\ {x_2}^3 \end{bmatrix}$

Hidden state : $\boldsymbol{h}^0$



A time step is a single iteration in an RNN where it processes one element of a sequence.

Time step 1:

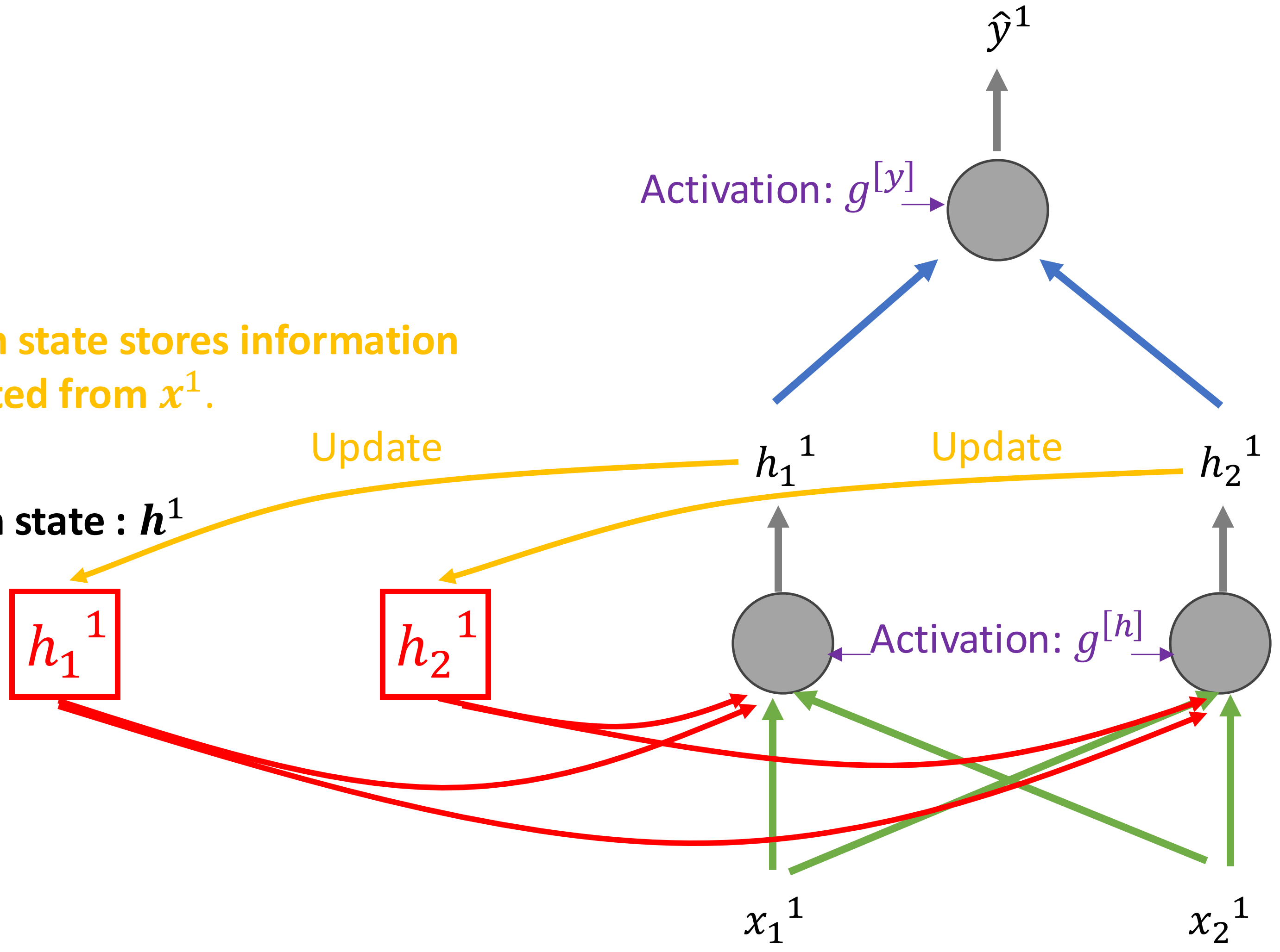
Hidden state: $\boldsymbol{h}^0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

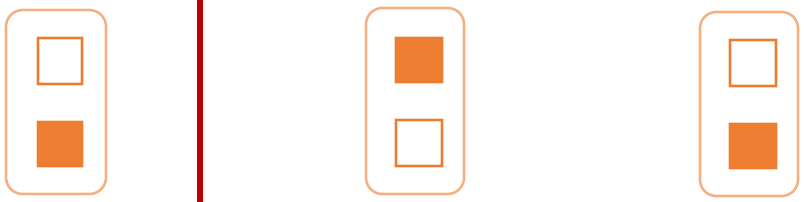
$$\boldsymbol{h}^1 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^{\top}\boldsymbol{x}^1 + \left(\boldsymbol{W}^{[hh]}\right)^{\top}\boldsymbol{h}^0\right)$$

Recurrent Neural Network

Hidden state stores information extracted from x^1.

Hidden state : $\boldsymbol{h}^1$





$$\boldsymbol{x}^1 = \begin{bmatrix} x_1{}^1 \\ x_2{}^1 \end{bmatrix} \quad \boldsymbol{x}^2 = \begin{bmatrix} x_1{}^2 \\ x_2{}^2 \end{bmatrix} \quad \boldsymbol{x}^3 = \begin{bmatrix} x_1{}^3 \\ x_2{}^3 \end{bmatrix}$$

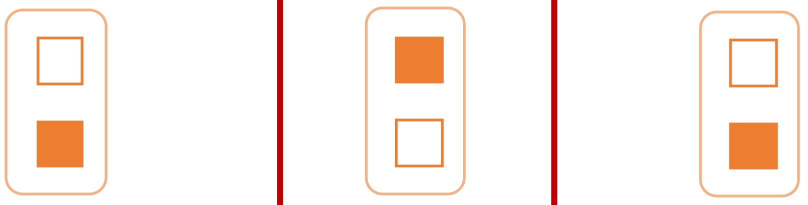
Time step 1:

Hidden state: $\boldsymbol{h}^1 = \begin{bmatrix} h_1{}^1 \\ h_2{}^1 \end{bmatrix}$

$$\boldsymbol{h}^1 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^1 + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^0\right)$$

$$\hat{y}^1 = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^\top \boldsymbol{h}^1\right)$$

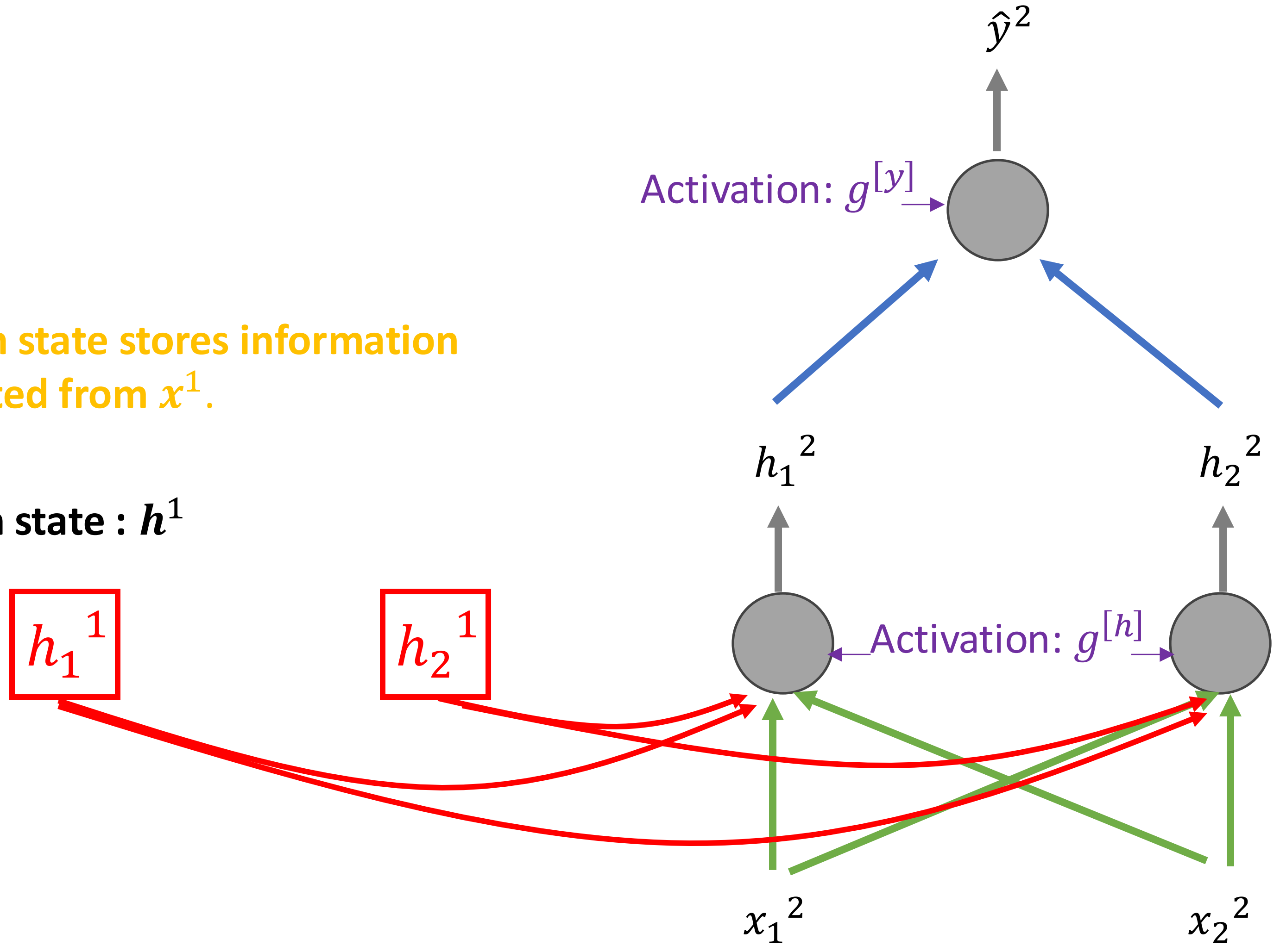
Recurrent Neural Network



$$\boldsymbol{x}^1 = \begin{bmatrix} {x_1}^1 \\ {x_2}^1 \end{bmatrix} \qquad \boldsymbol{x}^2 = \begin{bmatrix} {x_1}^2 \\ {x_2}^2 \end{bmatrix} \qquad \boldsymbol{x}^3 = \begin{bmatrix} {x_1}^3 \\ {x_2}^3 \end{bmatrix}$$

Hidden state stores information extracted from x^1.

Hidden state : $\boldsymbol{h}^1$

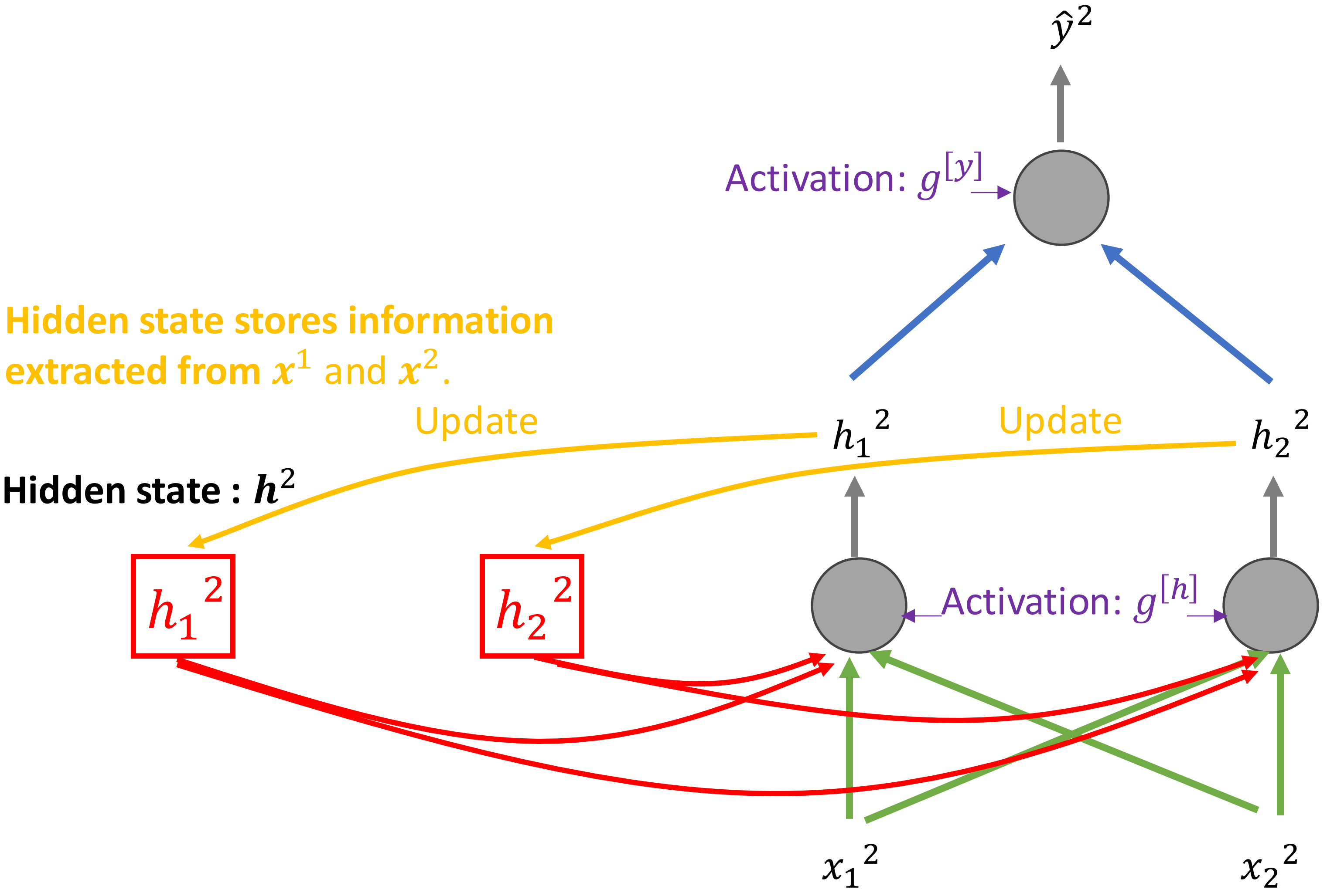


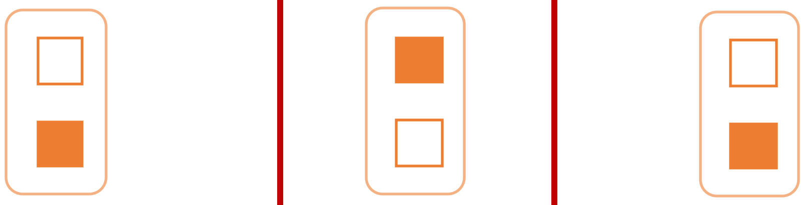
Time step 2:

Hidden state: $\boldsymbol{h}^1 = \begin{bmatrix} {h_1}^1 \\ {h_2}^1 \end{bmatrix}$

$$\boldsymbol{h}^2 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^2 + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^1\right)$$

Recurrent Neural Network





$\boldsymbol{x}^1 = \begin{bmatrix} x_1{}^1 \\ x_2{}^1 \end{bmatrix}$ $\quad \boxed{\boldsymbol{x}^2 = \begin{bmatrix} x_1{}^2 \\ x_2{}^2 \end{bmatrix}}$ $\quad \boldsymbol{x}^3 = \begin{bmatrix} x_1{}^3 \\ x_2{}^3 \end{bmatrix}$

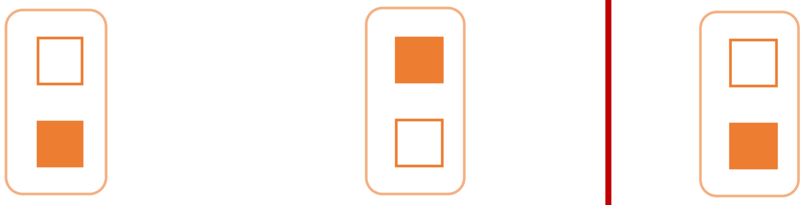
Time step 2:

Hidden state: $\boldsymbol{h}^2 = \begin{bmatrix} h_1{}^2 \\ h_2{}^2 \end{bmatrix}$

$$\boldsymbol{h}^2 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^2 + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^1\right)$$

$$\hat{y}^2 = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^\top \boldsymbol{h}^2\right)$$

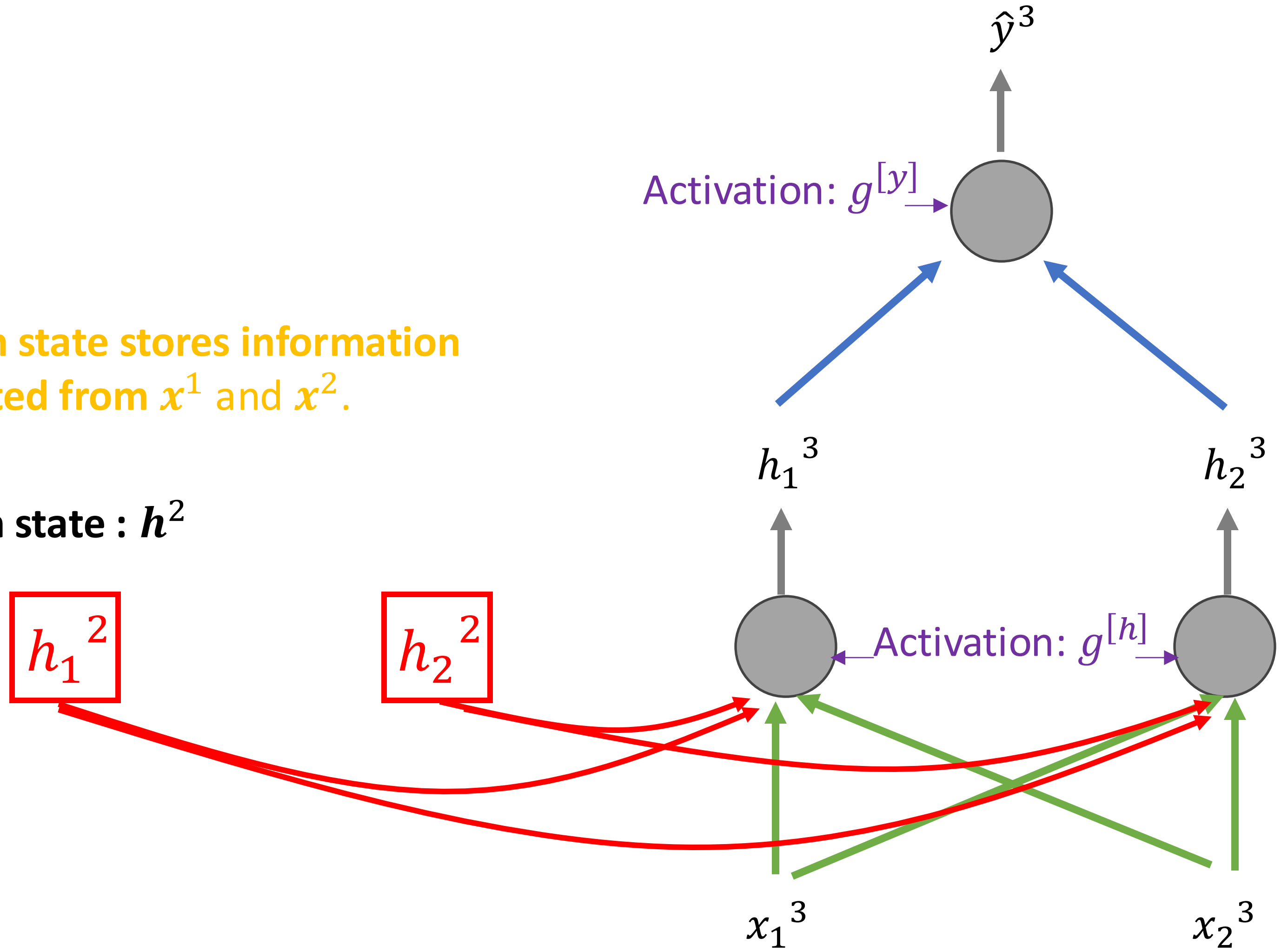
Recurrent Neural Network



$\boldsymbol{x}^1 = \begin{bmatrix} {x_1}^1 \\ {x_2}^1 \end{bmatrix}$ $\quad \boldsymbol{x}^2 = \begin{bmatrix} {x_1}^2 \\ {x_2}^2 \end{bmatrix}$ $\quad \boxed{\boldsymbol{x}^3 = \begin{bmatrix} {x_1}^3 \\ {x_2}^3 \end{bmatrix}}$

Hidden state stores information extracted from x^1 and x^2.

Hidden state : $\boldsymbol{h}^2$

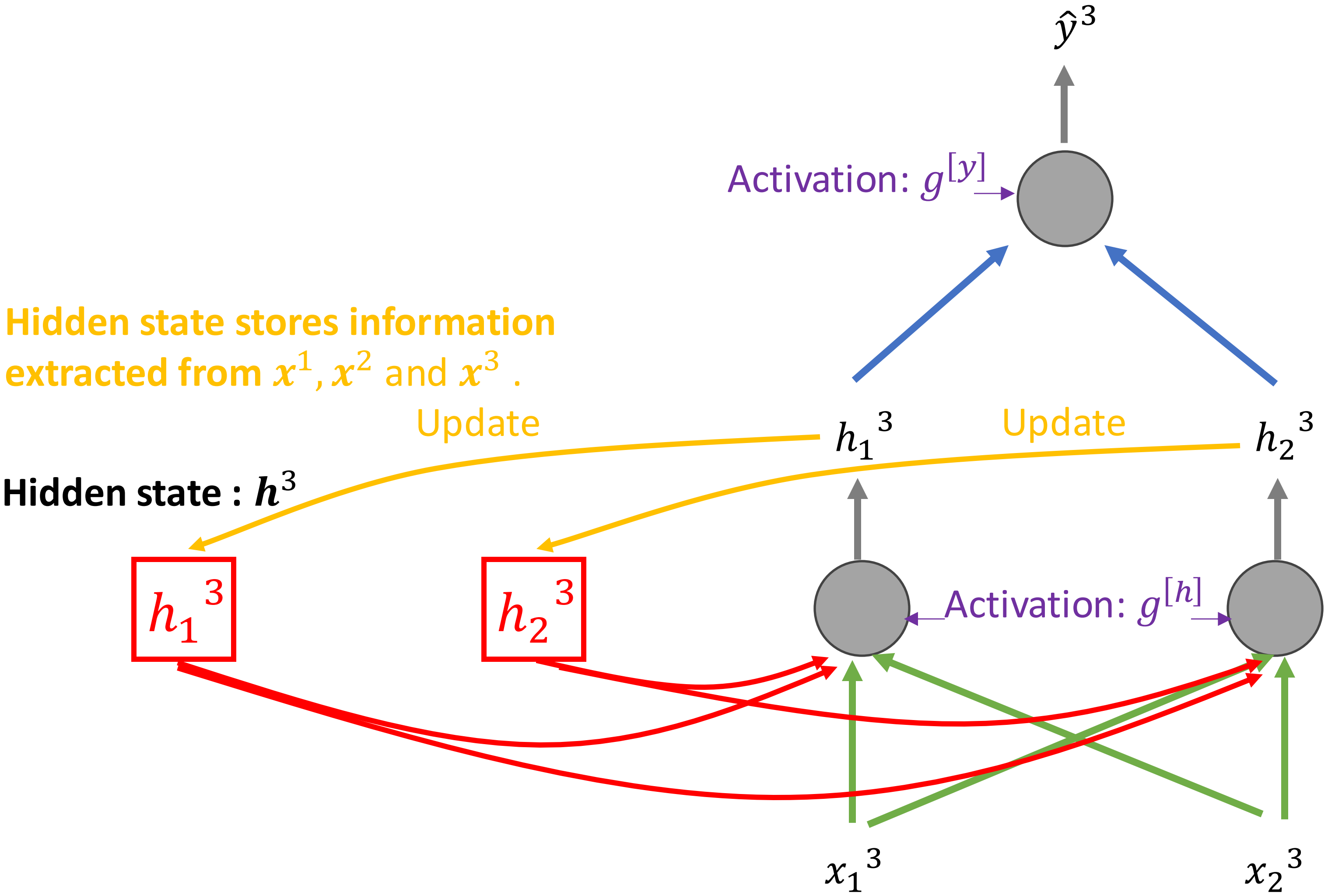


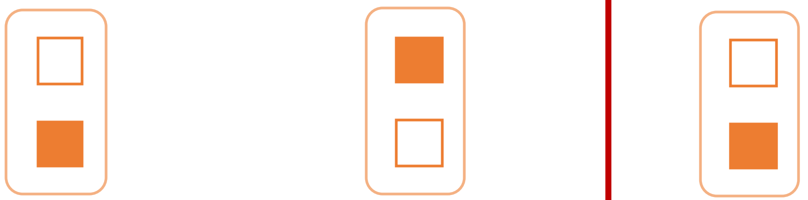
Time step 3:

Hidden state: $\boldsymbol{h}^2 = \begin{bmatrix} {h_1}^2 \\ {h_2}^2 \end{bmatrix}$

$$\boldsymbol{h}^3 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^3 + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^2\right)$$

Recurrent Neural Network





$\boldsymbol{x}^1 = \begin{bmatrix} x_1{}^1 \\ x_2{}^1 \end{bmatrix}$ $\boldsymbol{x}^2 = \begin{bmatrix} x_1{}^2 \\ x_2{}^2 \end{bmatrix}$ $\boldsymbol{x}^3 = \begin{bmatrix} x_1{}^3 \\ x_2{}^3 \end{bmatrix}$

Time step 3:

Hidden state: $\boldsymbol{h}^3 = \begin{bmatrix} h_1{}^3 \\ h_2{}^3 \end{bmatrix}$

$$\boldsymbol{h}^3 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^3 + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^2\right)$$

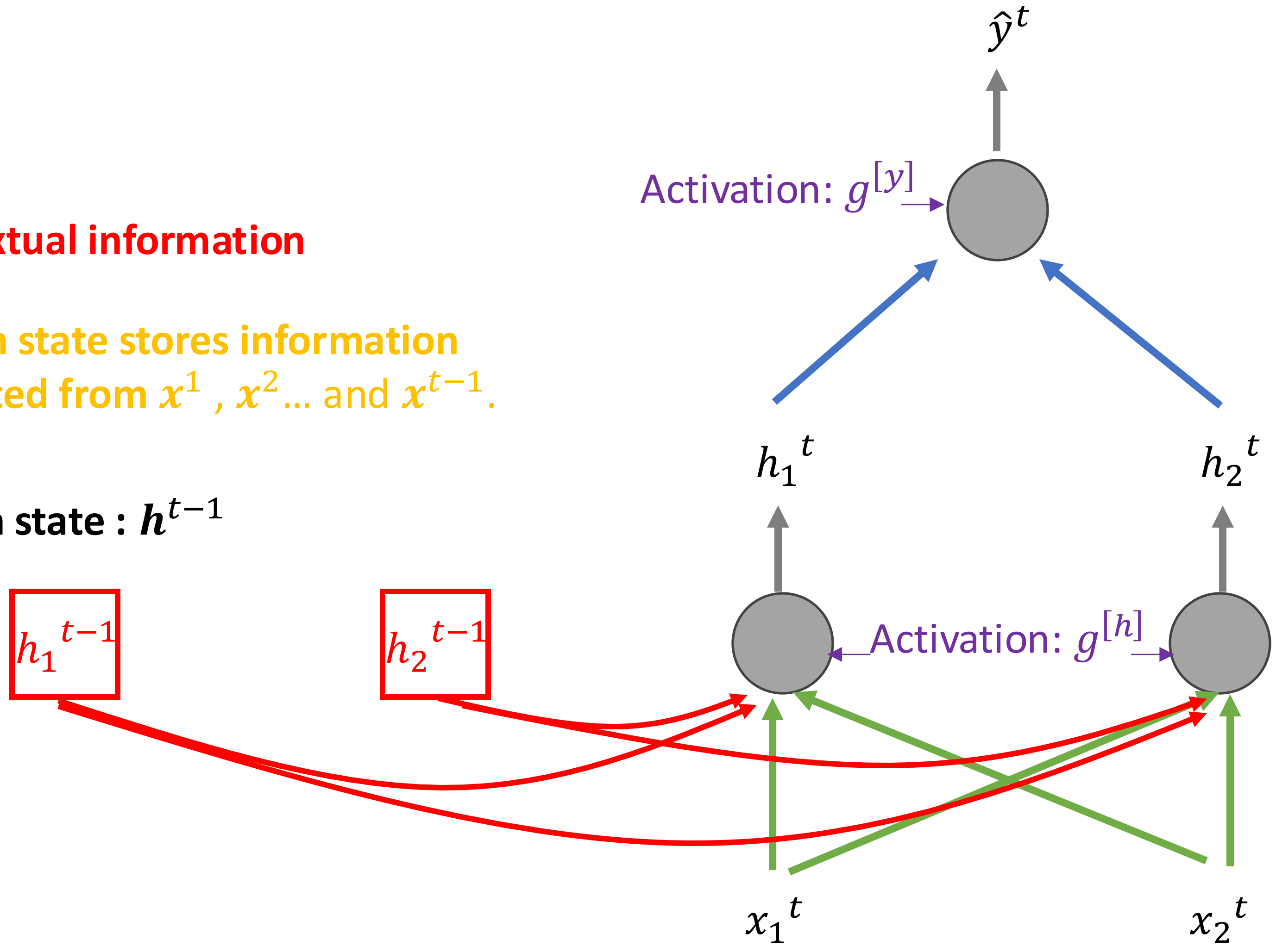
$$\hat{y}^3 = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^\top \boldsymbol{h}^3\right)$$

Recurrent Neural Network

Contextual information

Hidden state stores information extracted from $\boldsymbol{x}^1$, $\boldsymbol{x}^2$... and $\boldsymbol{x}^{t-1}$.

Hidden state : $\boldsymbol{h}^{t-1}$



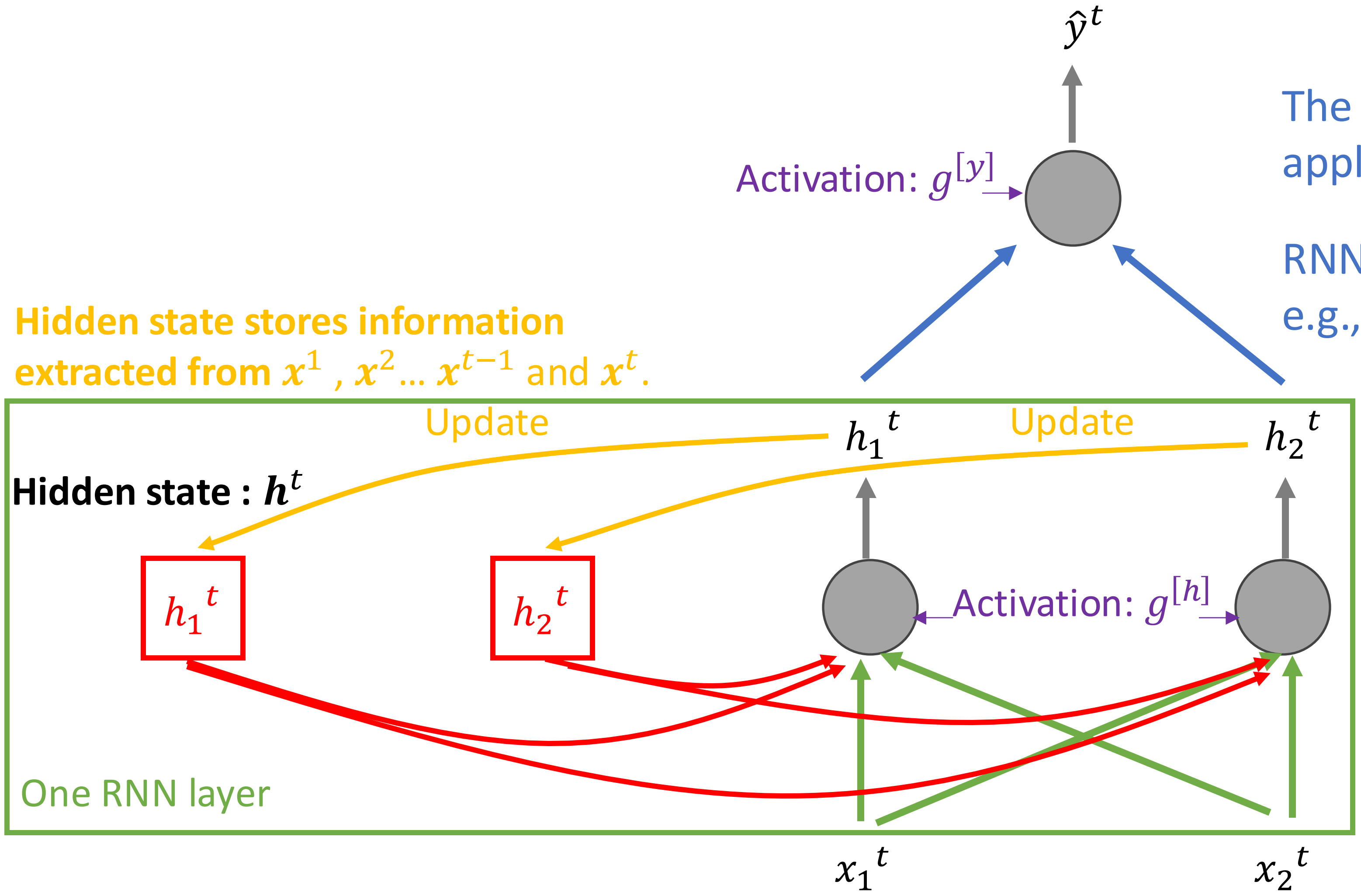
Time step $\boldsymbol{t}$:

Hidden state: $\boldsymbol{h}^{t-1} = \begin{bmatrix} {h_1}^{t-1} \\ {h_2}^{t-1} \end{bmatrix}$

$$\boldsymbol{h}^t = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^t + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^{t-1}\right)$$

Recurrent Neural Network

Hidden state stores information extracted from $\boldsymbol{x}^1$, $\boldsymbol{x}^2$... $\boldsymbol{x}^{t-1}$ and $\boldsymbol{x}^t$.



The same weights ($\boldsymbol{W}^{[xh]}$, $\boldsymbol{W}^{[hh]}$, $\boldsymbol{W}^{[hy]}$) are applied at each time step.

RNN can handle sentences of varying lengths, e.g., we saw this saw v.s. we saw this old saw.

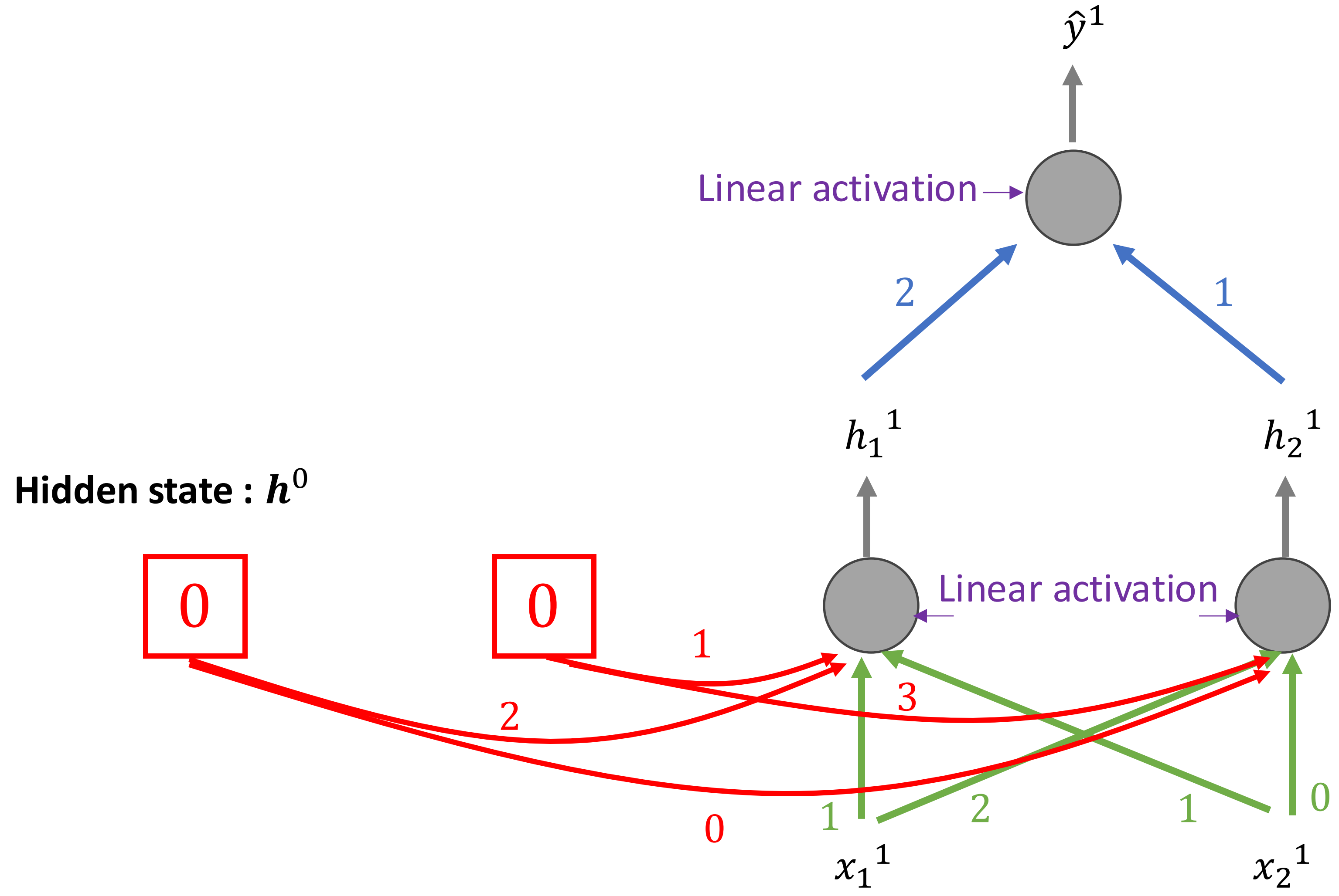
Time step $\boldsymbol{t}$:

Hidden state: $\boldsymbol{h}^t = \begin{bmatrix} h_1{}^t \\ h_2{}^t \end{bmatrix}$

$$\boldsymbol{h}^t = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^t + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^{t-1}\right)$$

$$\hat{y}^t = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^\top \boldsymbol{h}^t\right)$$

Recurrent Neural Network: Example



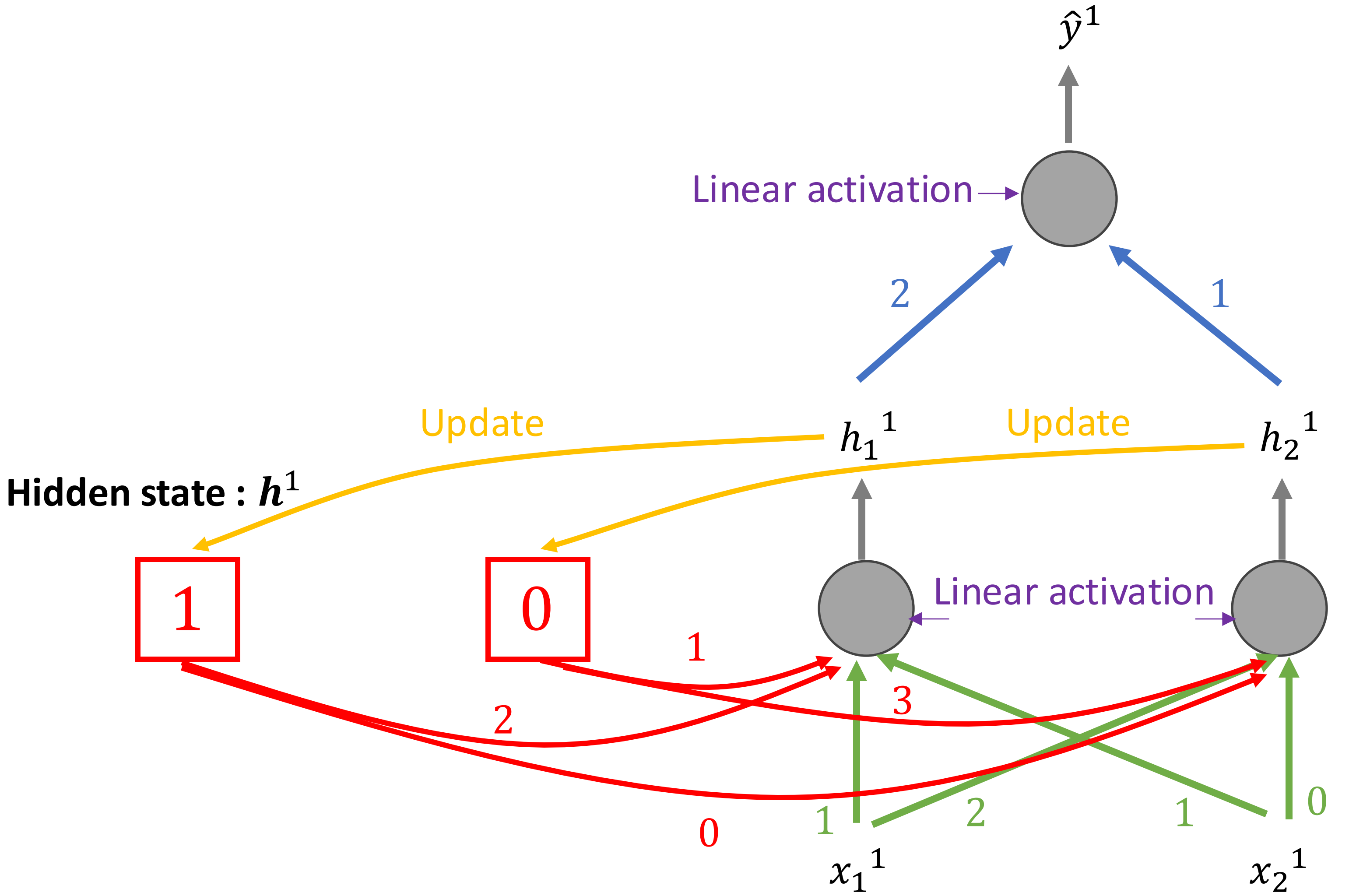
$\boldsymbol{x}^1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ $\quad \boldsymbol{x}^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\quad \boldsymbol{x}^3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

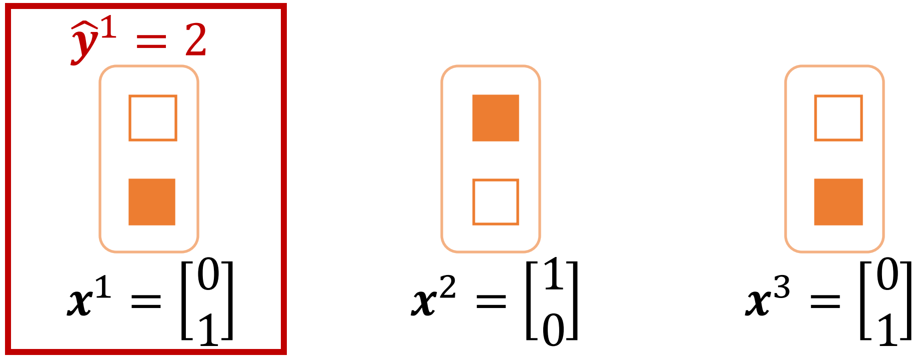
Time step 1:

Hidden state: $\boldsymbol{h}^0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$$\begin{aligned}\boldsymbol{h}^1 &= g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^{\top}\boldsymbol{x}^1 + \left(\boldsymbol{W}^{[hh]}\right)^{\top}\boldsymbol{h}^0\right) \\ &= \left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}\right)^{\top}\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \left(\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}\right)^{\top}\begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\end{aligned}$$

Recurrent Neural Network: Example





Time step 1:

Hidden state: $\boldsymbol{h}^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$$\boldsymbol{h}^1 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^1 + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^0\right)$$

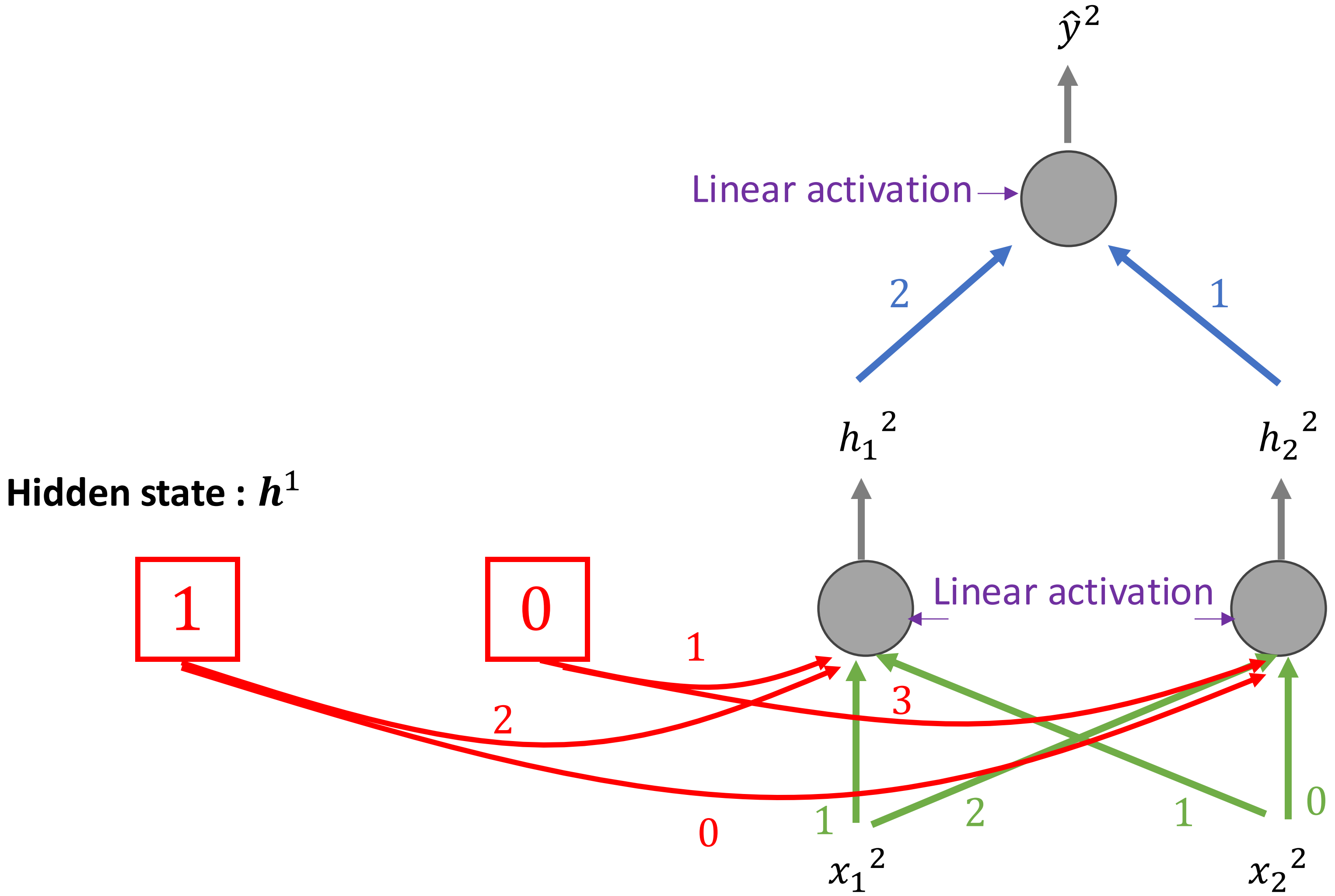
$$= \left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}\right)^\top \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \left(\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}\right)^\top \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

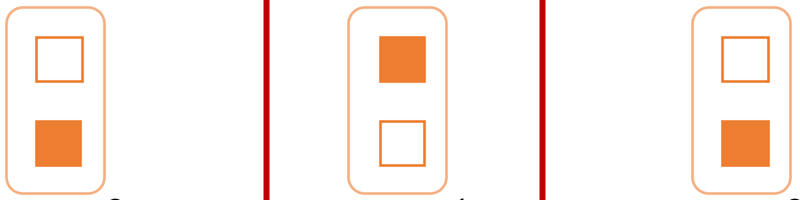
$$\hat{y}^1 = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^\top \boldsymbol{h}^1\right)$$

$$= \left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right)^\top \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2$$

Recurrent Neural Network: Example



$\hat{\boldsymbol{y}}^1 = 2$



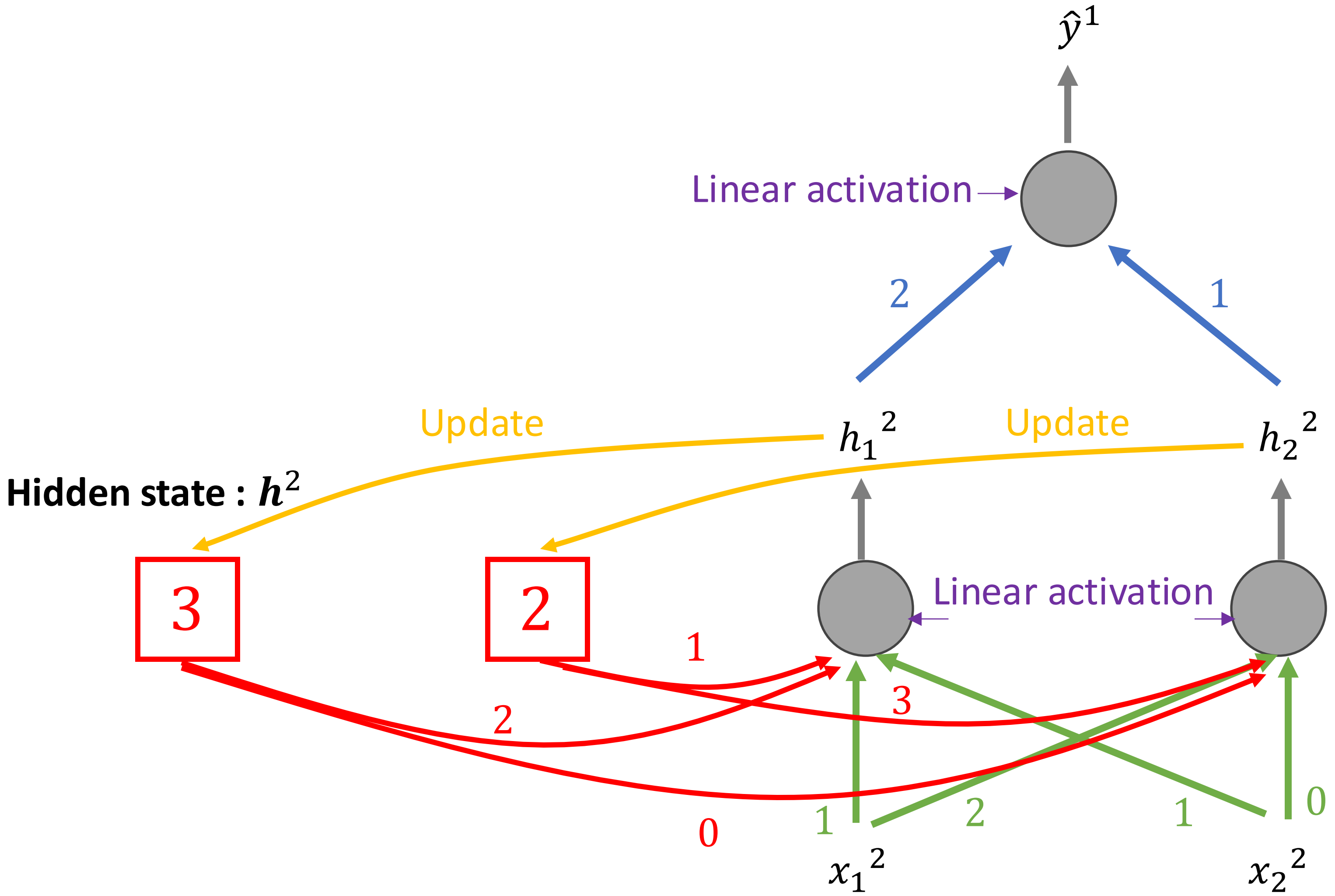
$\boldsymbol{x}^1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ $\boxed{\boldsymbol{x}^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}}$ $\boldsymbol{x}^3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Time step 2:

Hidden state: $\boldsymbol{h}^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$$\boldsymbol{h}^2 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^{\top}\boldsymbol{x}^2 + \left(\boldsymbol{W}^{[hh]}\right)^{\top}\boldsymbol{h}^1\right)$$
$$= \left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}\right)^{\top}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \left(\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}\right)^{\top}\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

Recurrent Neural Network: Example



$\hat{\boldsymbol{y}}^1 = 2$ $\qquad$ $\hat{\boldsymbol{y}}^2 = 8$

$\boldsymbol{x}^1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ $\qquad$ $\boldsymbol{x}^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\qquad$ $\boldsymbol{x}^3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Time step 2:

Hidden state: $\boldsymbol{h}^2 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$

$$\boldsymbol{h}^2 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^2 + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^1\right)$$

$$= \left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}\right)^\top \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \left(\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}\right)^\top \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

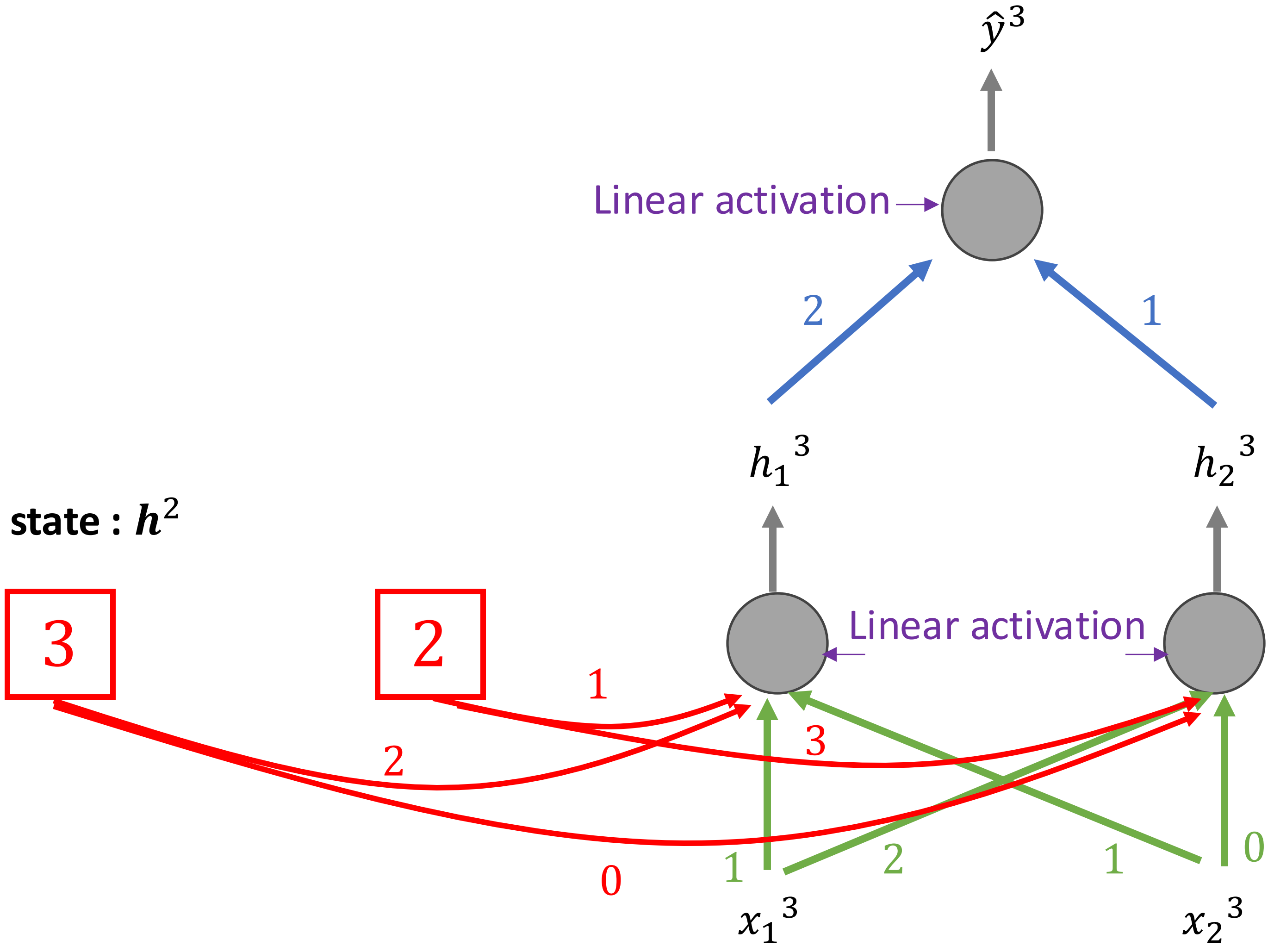
$$= \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\hat{y}^2 = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^\top \boldsymbol{h}^2\right)$$

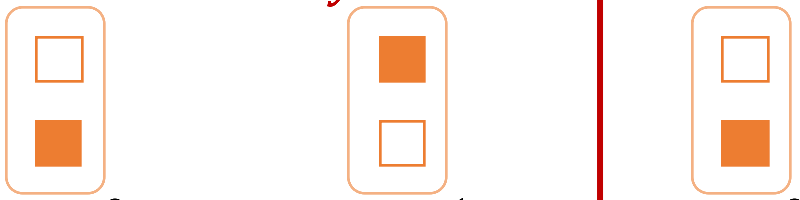
$$= \left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right)^\top \begin{bmatrix} 3 \\ 2 \end{bmatrix} = 8$$

Recurrent Neural Network: Example

Hidden state : $\boldsymbol{h}^2$



$\hat{\boldsymbol{y}}^1 = 2$ $\quad$ $\hat{\boldsymbol{y}}^2 = 8$



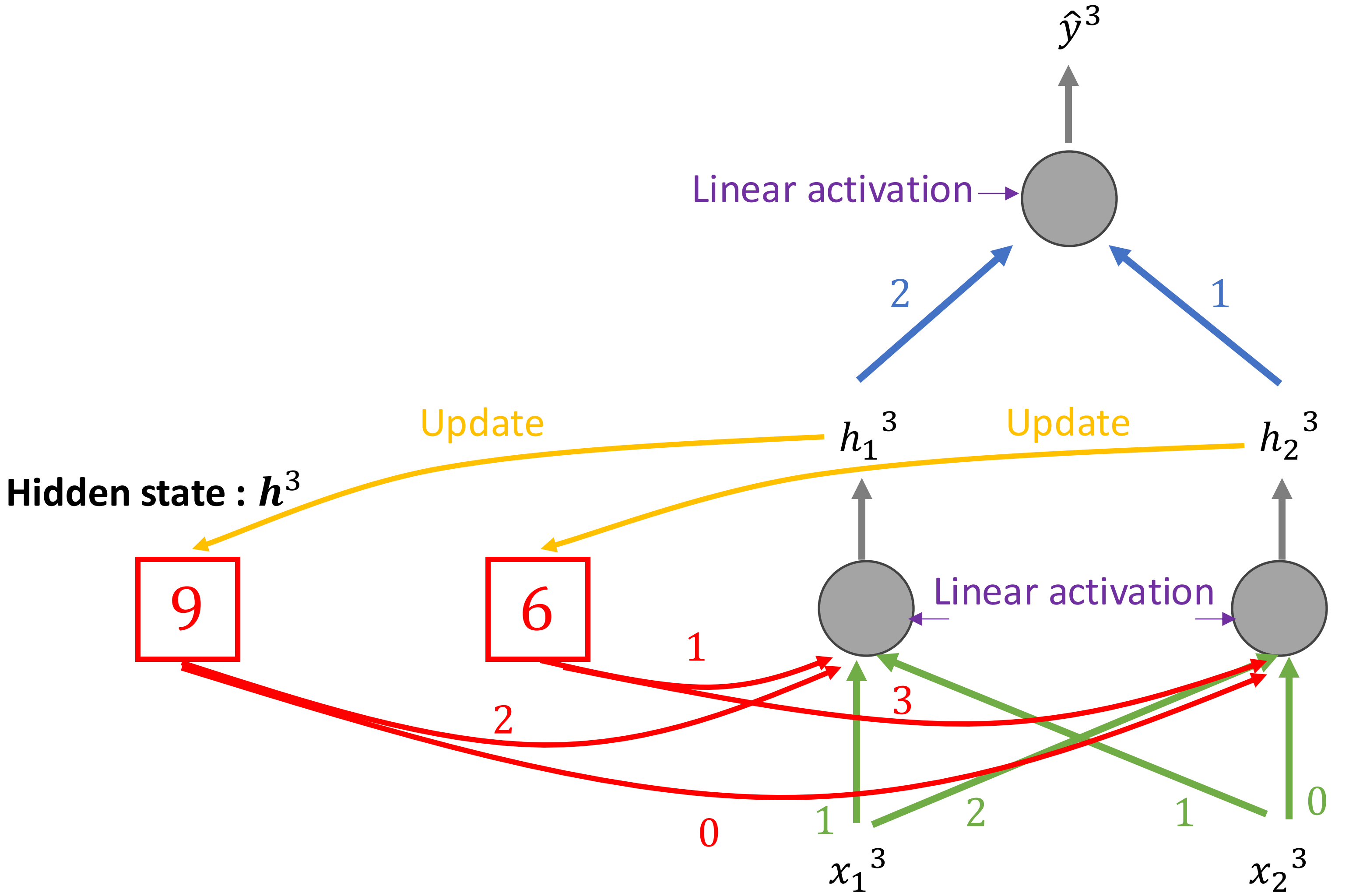
$\boldsymbol{x}^1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ $\quad$ $\boldsymbol{x}^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\quad$ $\boldsymbol{x}^3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

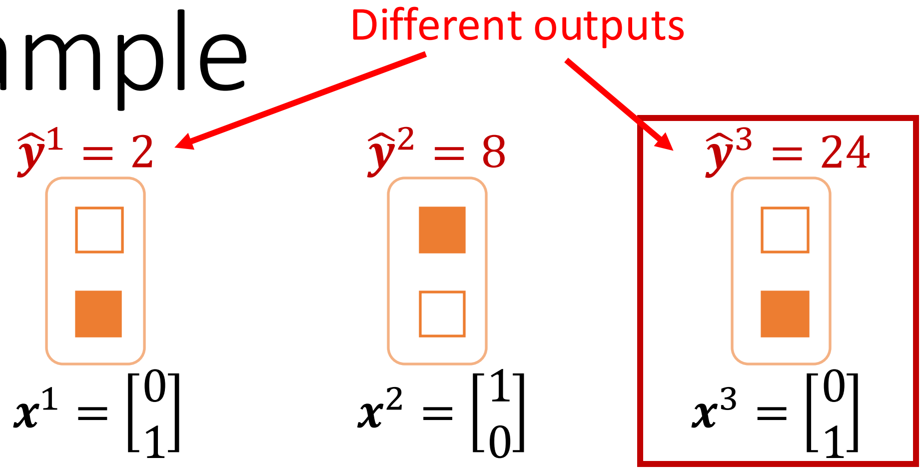
Time step 3:

Hidden state: $\boldsymbol{h}^2 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$

$$\boldsymbol{h}^3 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^{\top}\boldsymbol{x}^3 + \left(\boldsymbol{W}^{[hh]}\right)^{\top}\boldsymbol{h}^2\right)$$
$$= \left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}\right)^{\top}\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \left(\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}\right)^{\top}\begin{bmatrix} 3 \\ 2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 8 \\ 6 \end{bmatrix} = \begin{bmatrix} 9 \\ 6 \end{bmatrix}$$

Recurrent Neural Network: Example





Time step 3:

Hidden state: $\boldsymbol{h}^3 = \begin{bmatrix} 9 \\ 6 \end{bmatrix}$

$$\boldsymbol{h}^3 = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^\top \boldsymbol{x}^3 + \left(\boldsymbol{W}^{[hh]}\right)^\top \boldsymbol{h}^2\right)$$

$$= \left(\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}\right)^\top \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \left(\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}\right)^\top \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

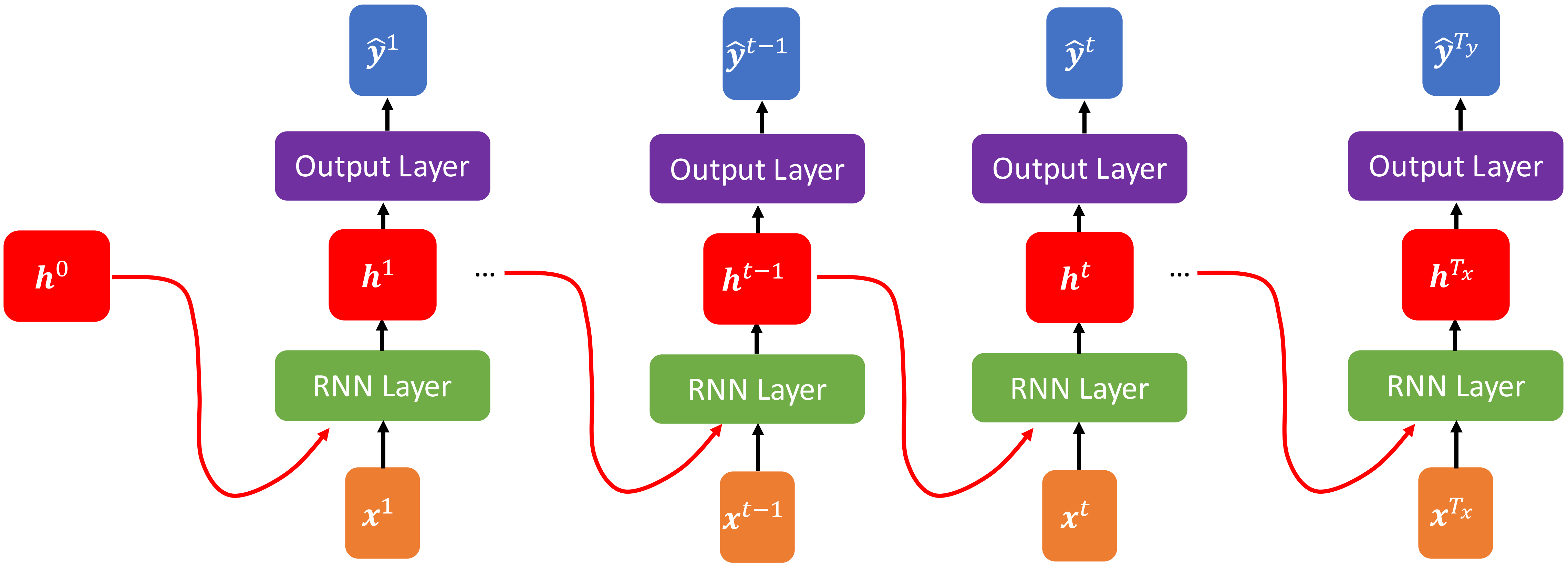
$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 8 \\ 6 \end{bmatrix} = \begin{bmatrix} 9 \\ 6 \end{bmatrix}$$

$$\hat{y}^3 = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^\top \boldsymbol{h}^3\right)$$

$$= \left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right)^\top \begin{bmatrix} 9 \\ 6 \end{bmatrix} = 24$$

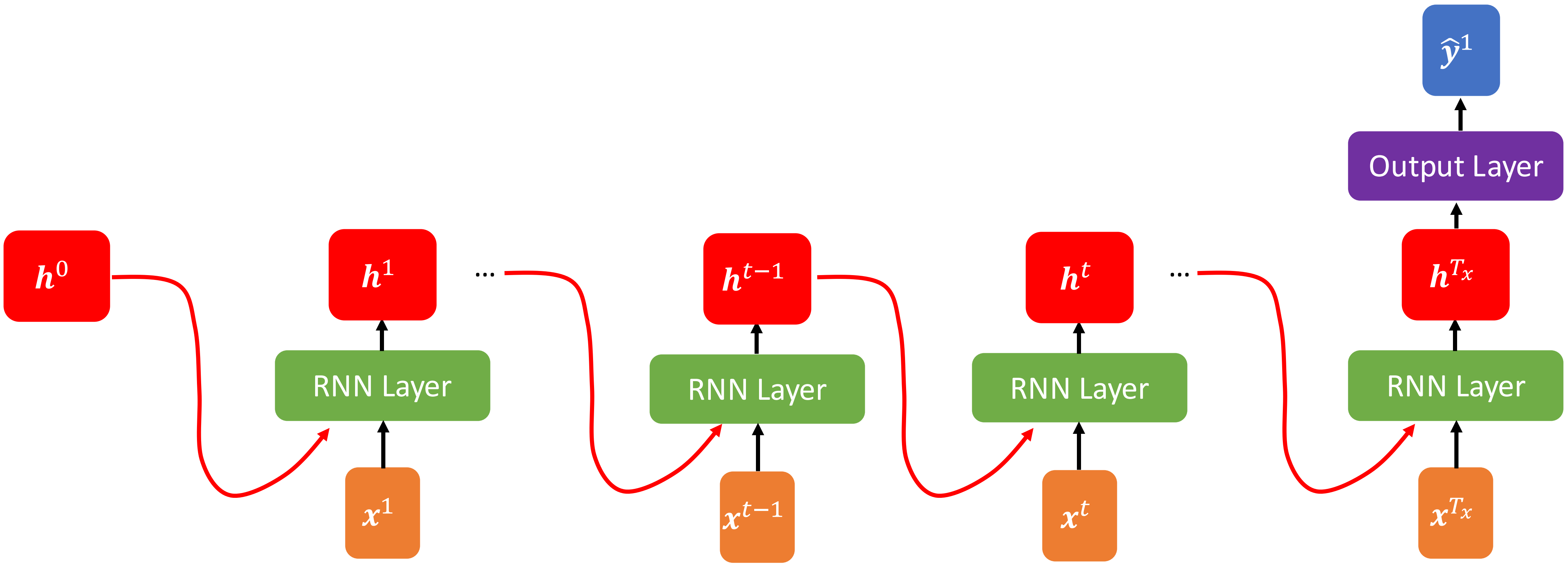
Recurrent Neural Network: Many-to-Many

$T_x = T_y > 1$



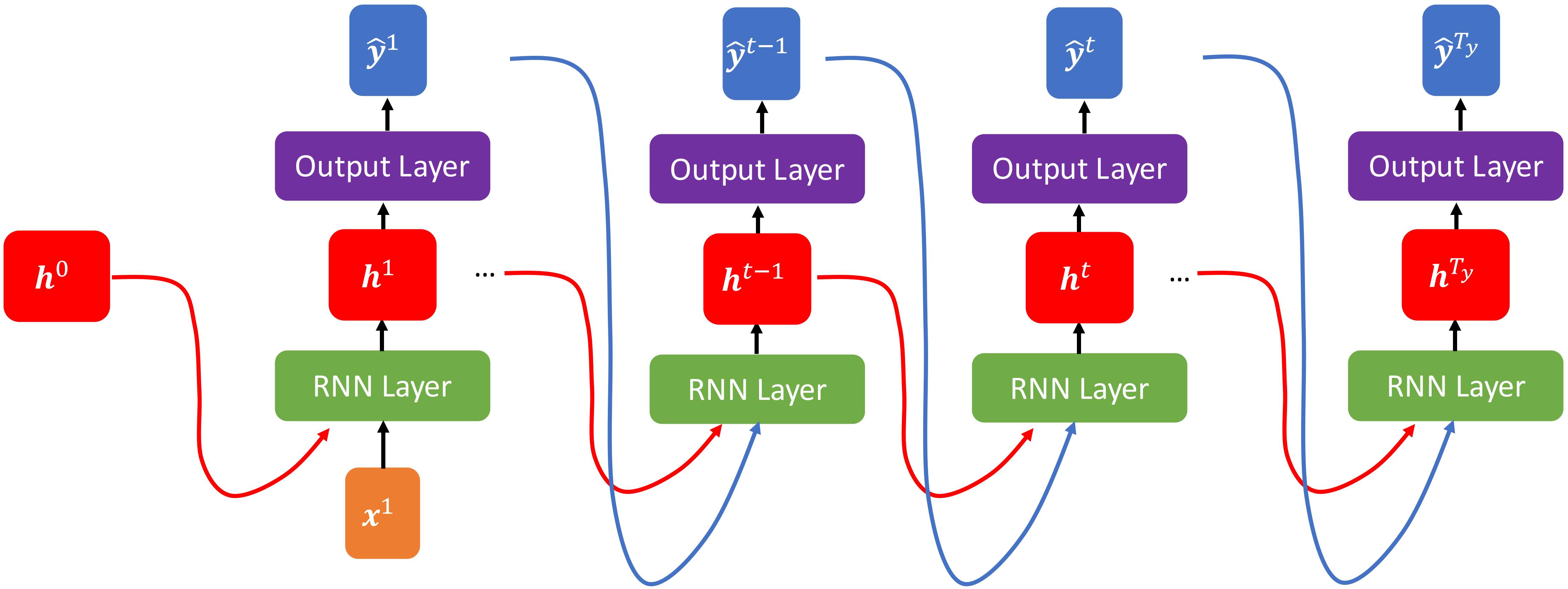
Recurrent Neural Network: Many-to-One

$T_x > 1, T_y = 1$



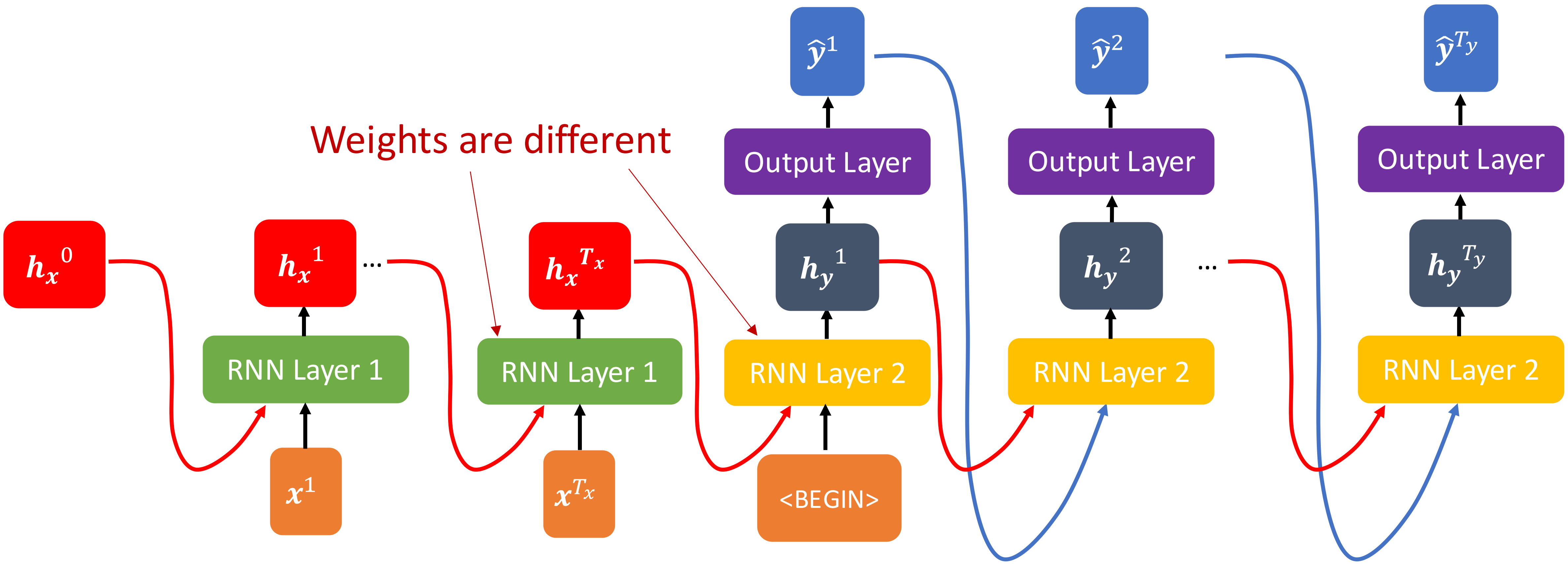
Recurrent Neural Network: One-to-Many

$T_x = 1, T_y > 1$



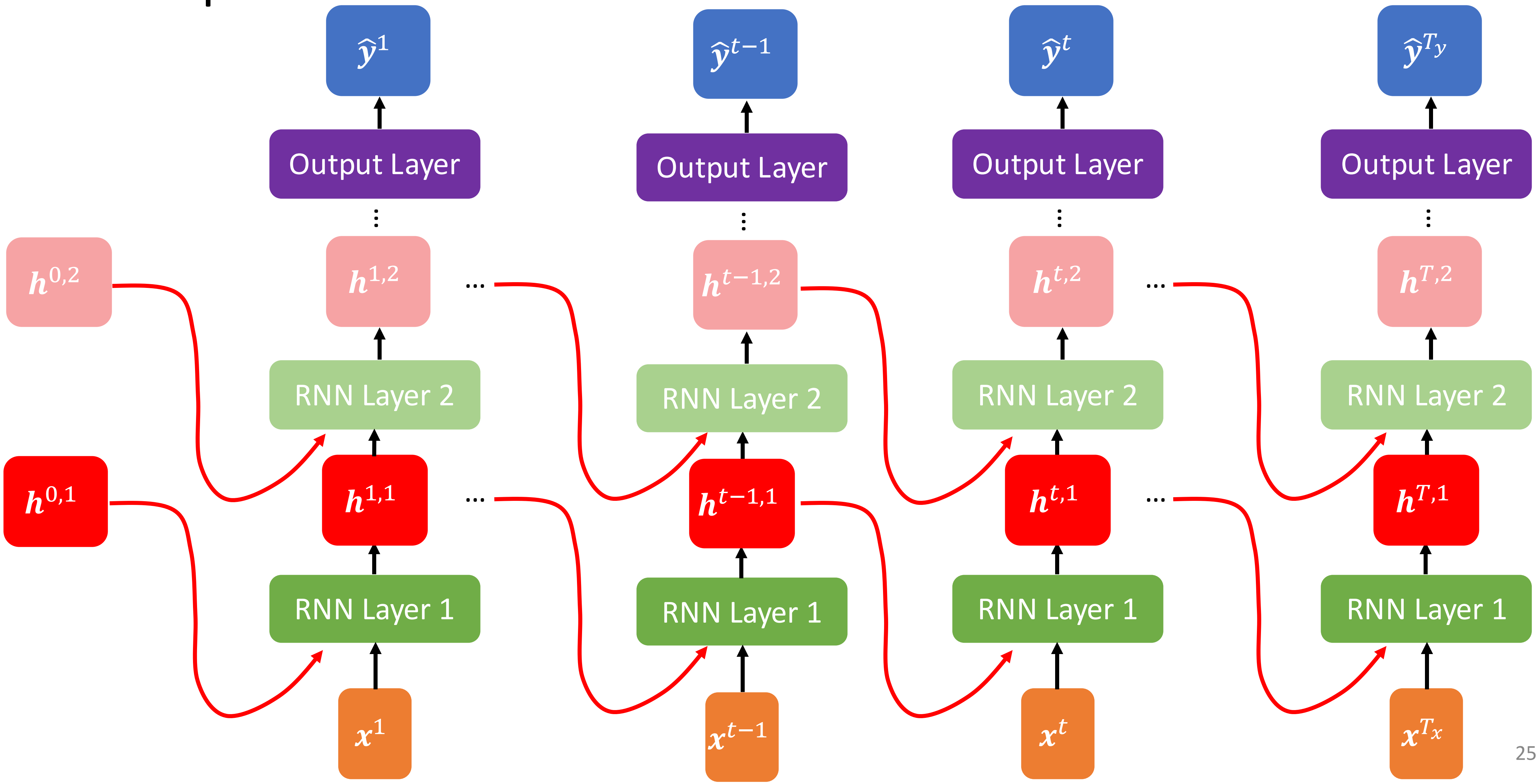
Recurrent Neural Network: Many-to-Many

$T_x \neq T_y, T_x > 1, T_y > 1$



<BEGIN> should be added into the vocabulary and encoded like other words.

Deep Recurrent Neural Networks



Applications of RNN

Sentiment Analysis :

"Decent effort. The plot could have been better."

Speech Recognition:

"Fuzzy Wuzzy was a bear. Fuzzy Wuzzy had no hair."

Video Captioning:

"A man is running."

Source: https://datahacker.rs/003-rnn-architectural-types-of-different-recurrent-neural-networks/

Properties of RNN

- **Capture contextual information**

- **The prediction at time step t must wait until all previous steps have been completed.**

Not parallelism-friendly

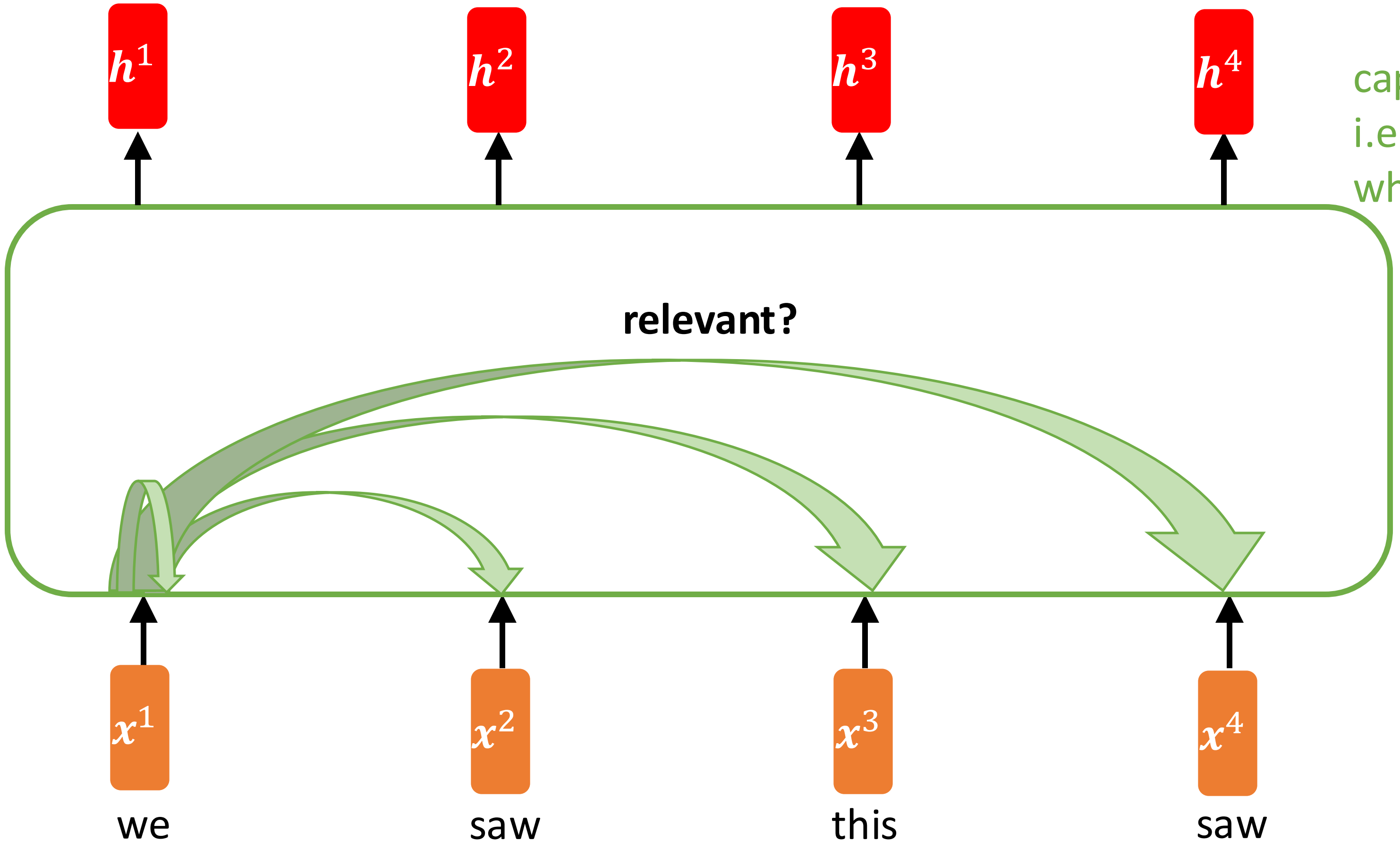
Is it possible to capture contextual information without waiting for previous time steps?

Self-attention layer!

Outline

- Recurrent Neural Networks
  - Motivation
  - Recurrent Neural Networks
  - Applications
- **Self-Attention**
  - Self-Attention Layer
  - Positional Encoding
- Issues with Deep Learning
  - Overfitting
  - Vanishing/Exploding Gradient

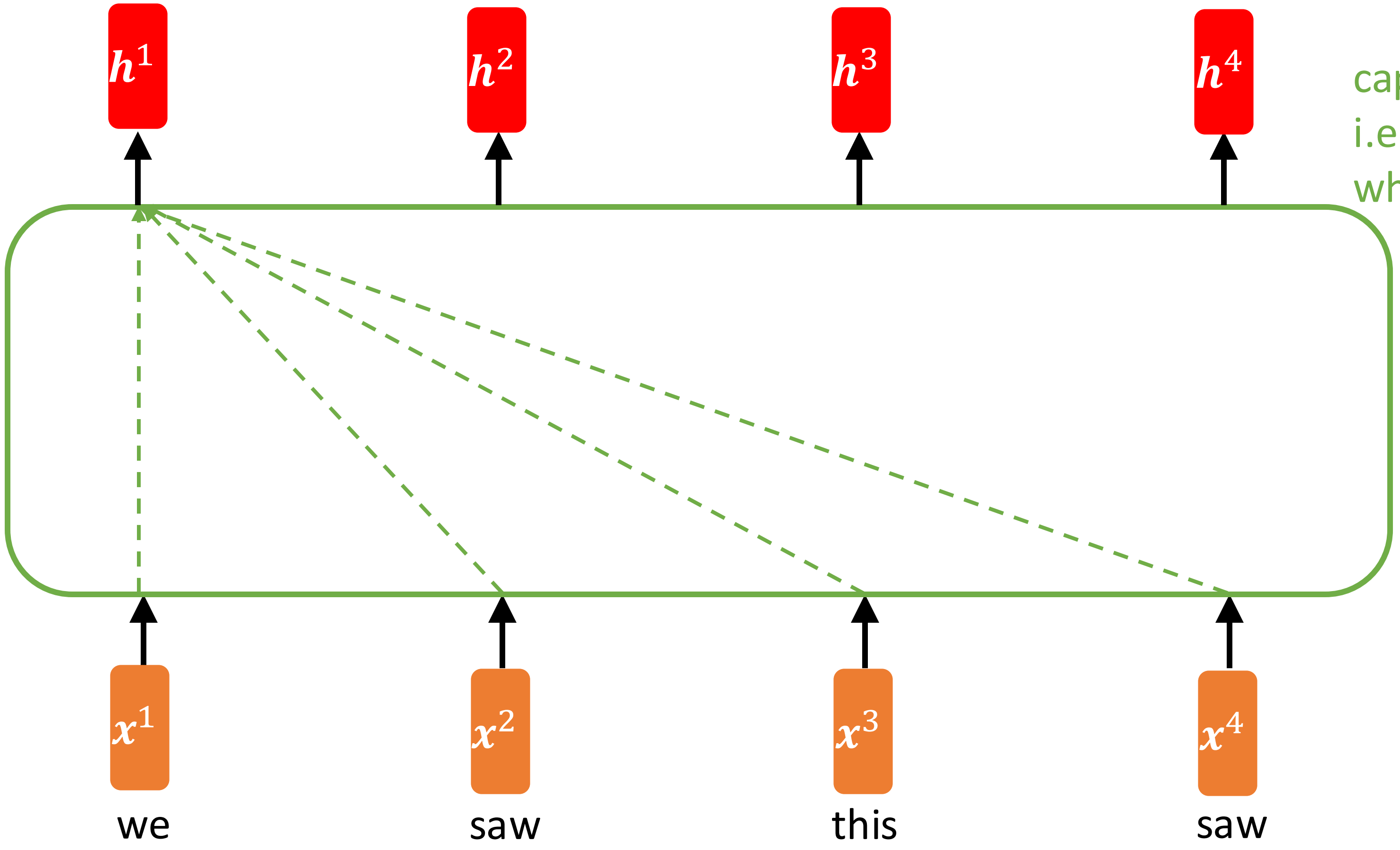
Self-Attention Layer



capture contextual information, i.e., useful information from the whole input sequence.

Identify whether the information from elements in the input sequence is relevant for generating the current output.

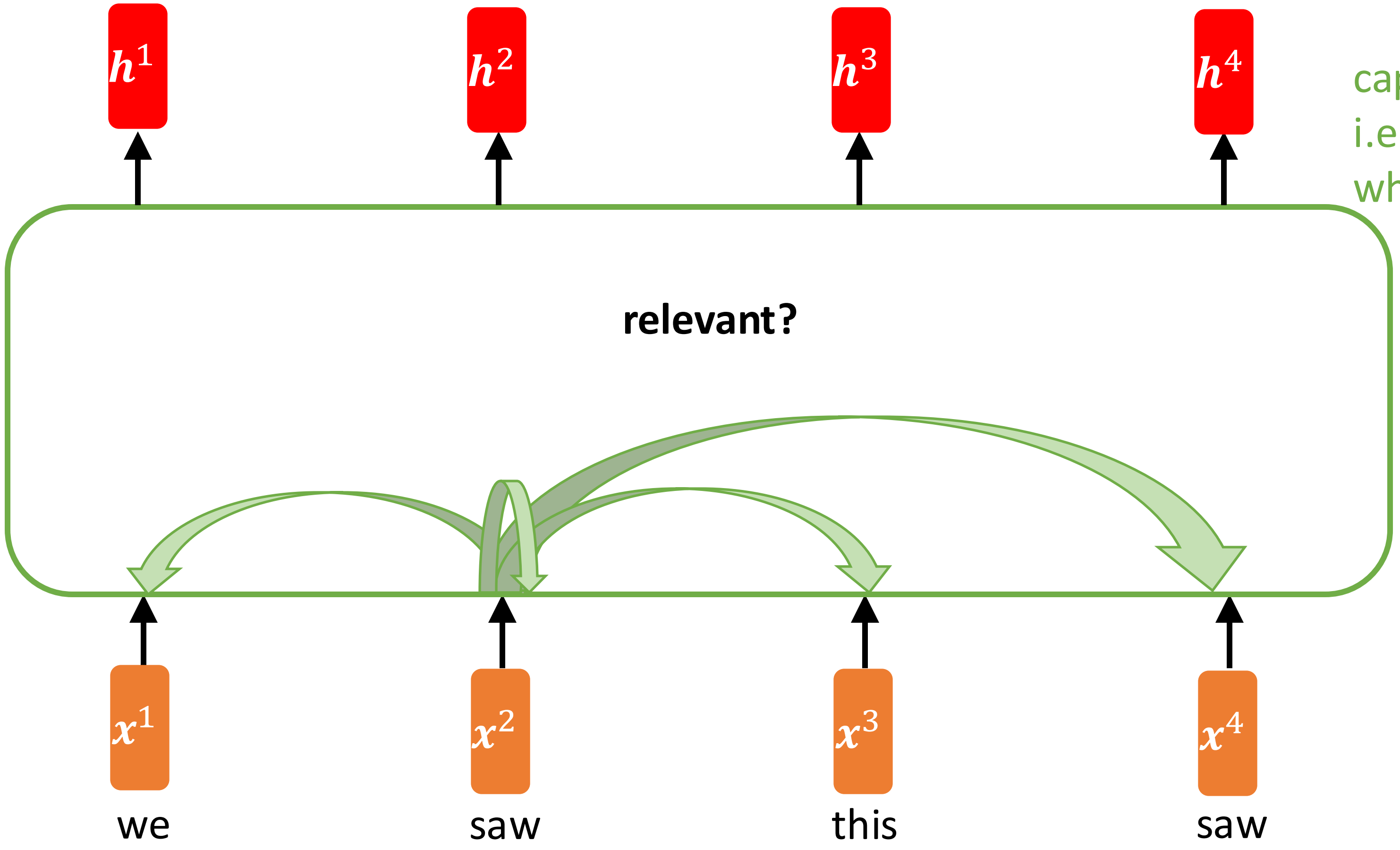
Self-Attention Layer



capture contextual information, i.e., useful information from the whole input sequence.

Aggregate useful information from all elements in the sequence to predict the current output.

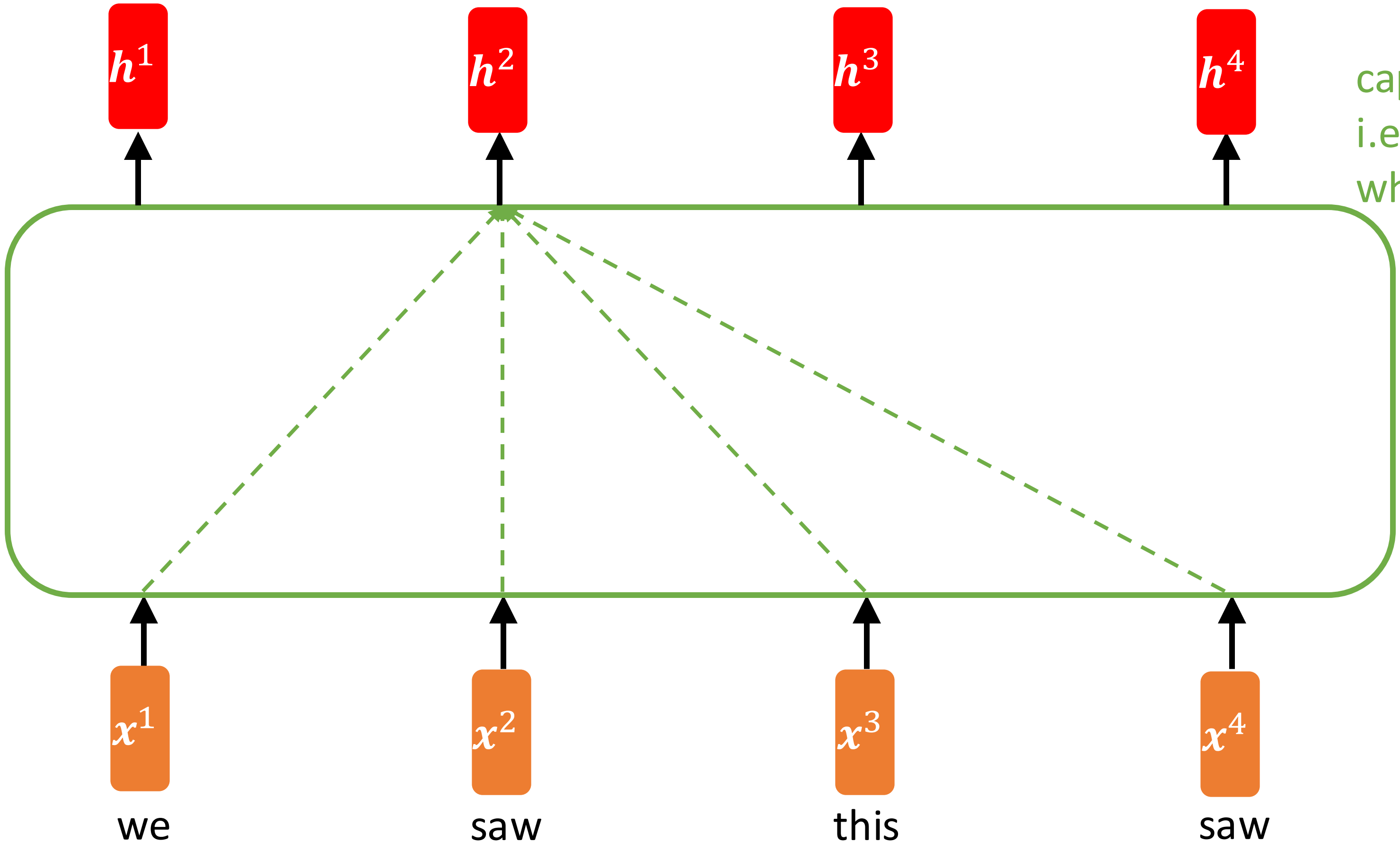
Self-Attention Layer



capture contextual information, i.e., useful information from the whole input sequence.

Identify whether the information from elements in the input sequence is relevant for generating the current output.

Self-Attention Layer



capture contextual information, i.e., useful information from the whole input sequence.

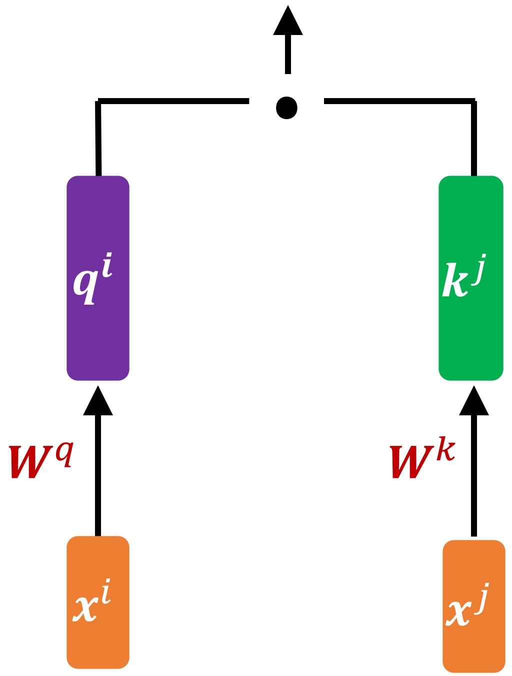
Aggregate useful information from all elements in the sequence to predict the current output.

Self-Attention Layer

- **Attention score:** Determine how much focus (or "attention") each part of the input sequence ($\boldsymbol{x}^j$) should receive when processing a specific element ($\boldsymbol{x}^i$).

$$\boldsymbol{\alpha}_{ij} = \boldsymbol{k}^j \cdot \boldsymbol{q}^i = (\boldsymbol{k}^j)^\top \boldsymbol{q}^i$$

scalar



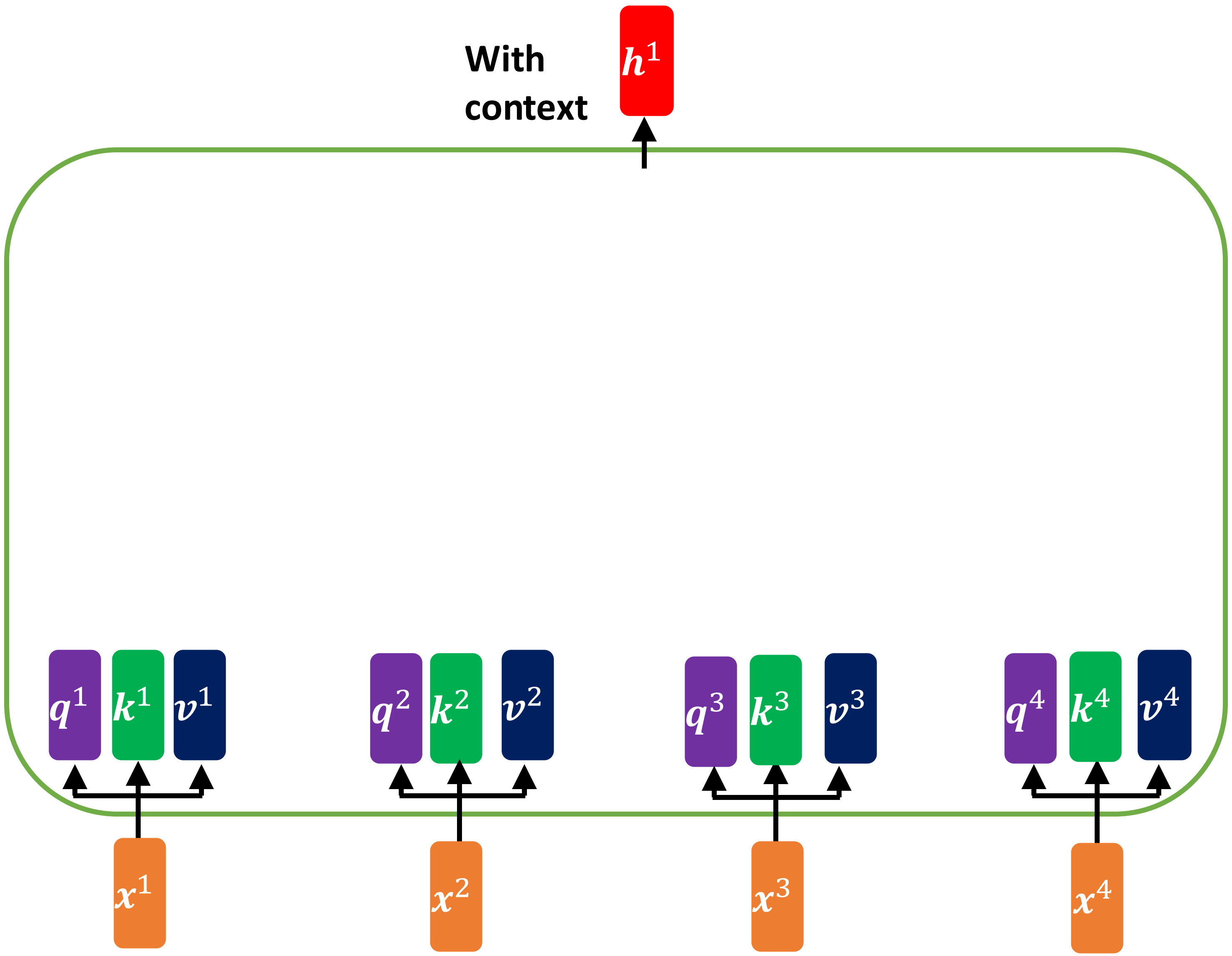
Query: $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key: $\boldsymbol{k}^j = \boldsymbol{W}^k \boldsymbol{x}^j$

The matrices $\boldsymbol{W}^q$ and $\boldsymbol{W}^k$ enable the model to extract useful features for attention score calculation from the inputs

$\boldsymbol{W}^q$ and $\boldsymbol{W}^k$ are trainable weight matrices.

Self-Attention Layer



Step 1: Linear Projection

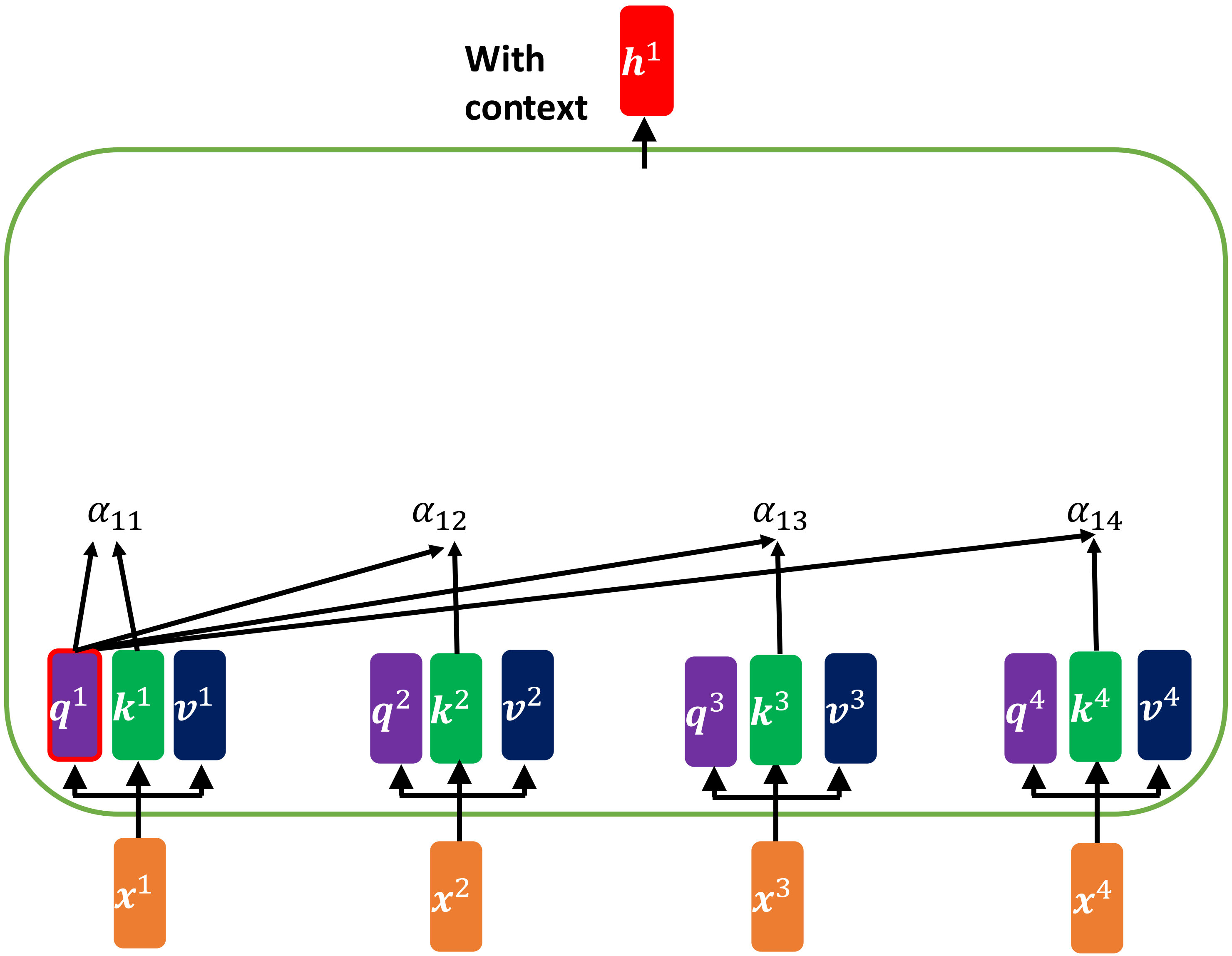
Transform input vectors into Query, Key, and Value using weights matrices (shared across inputs)

Query: $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key: $\boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{x}^i$

Value: $\boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{x}^i$

Self-Attention Layer



Step 2: Compute the attention scores:

$\alpha_{1j} = \boldsymbol{k}^{\boldsymbol{j}} \cdot \boldsymbol{q}^1 = (\boldsymbol{k}^j)^\top \boldsymbol{q}^1$

Step 1: Linear Projection

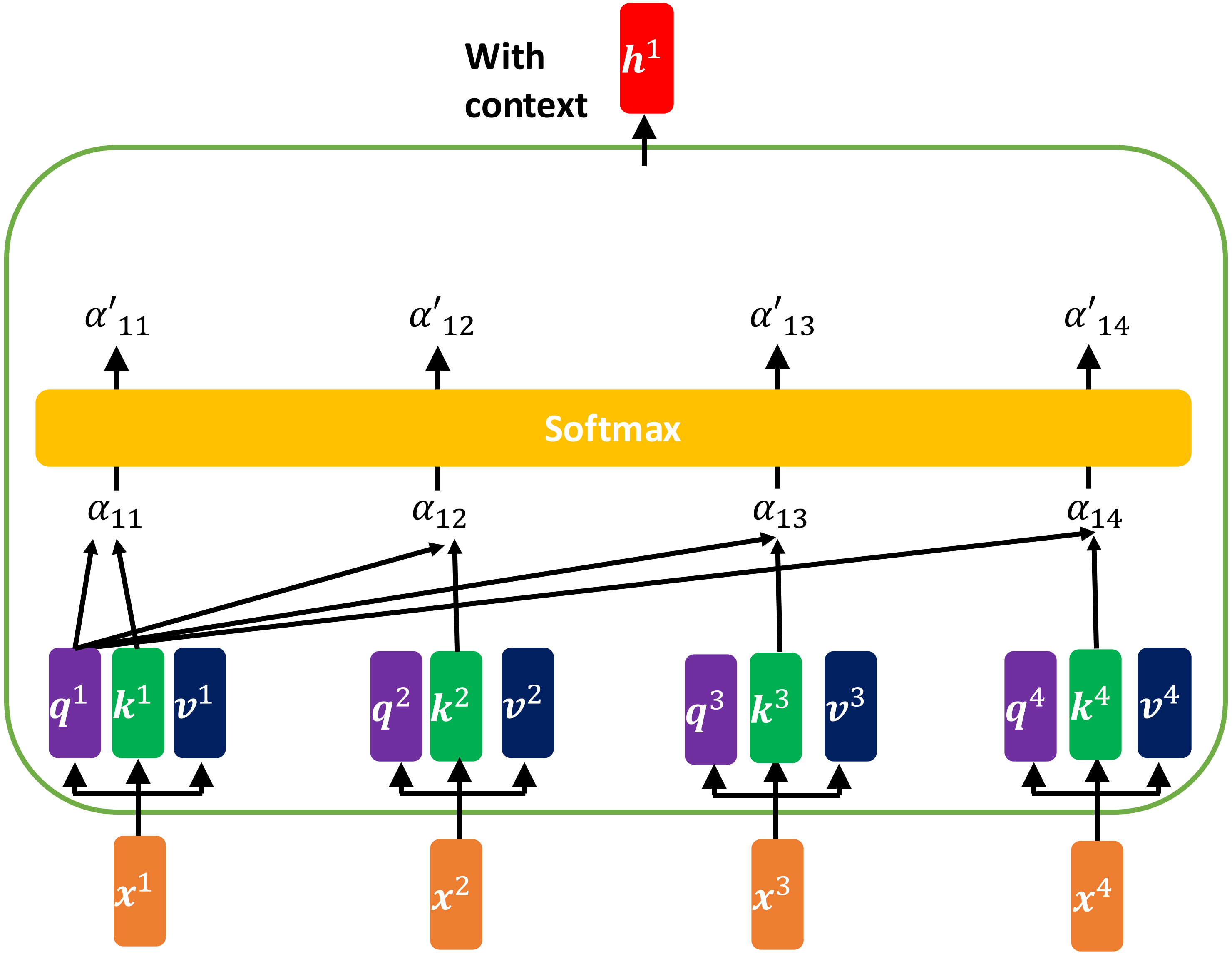
Transform input vectors into Query, Key, and Value using weights matrices (shared across inputs)

Query: $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key: $\boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{x}^i$

Value: $\boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{x}^i$

Self-Attention Layer



Step 3: Apply Softmax: $\alpha'_{1j} = \frac{e^{\alpha_{1j}}}{\sum_j e^{\alpha_{1j}}}$

Step 2: Compute the attention scores:
$\alpha_{1j} = \boldsymbol{k^j} \cdot \boldsymbol{q}^1 = (\boldsymbol{k}^j)^\top \boldsymbol{q}^1$

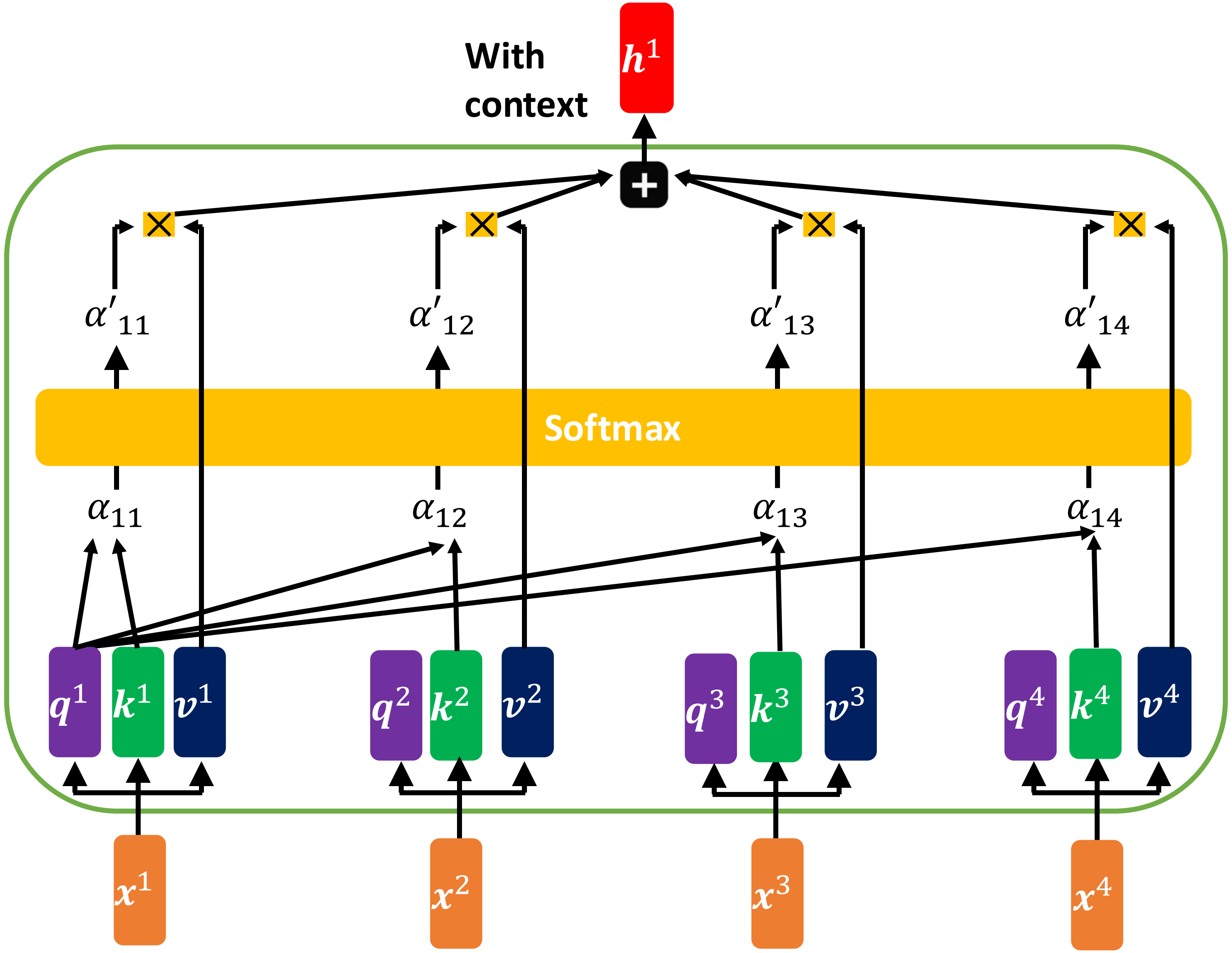
Step 1: Linear Projection
Transform input vectors into Query, Key, and Value using weights matrices (shared across inputs)

Query: $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key: $\boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{x}^i$

Value: $\boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{x}^i$

Self-Attention Layer



The actual content to aggregate

Step 4: Aggregate information: Multiply Values by attention score (after Softmax)

$$h^1 = \sum_j \alpha'_{1j} v^j$$

Step 3: Apply Softmax: $\alpha'_{1j} = \frac{e^{\alpha_{1j}}}{\sum_j e^{\alpha_{1j}}}$

Step 2: Compute the attention scores:

$\alpha_{1j} = k^j \cdot q^1 = (k^j)^\top q^1$

Step 1: Linear Projection

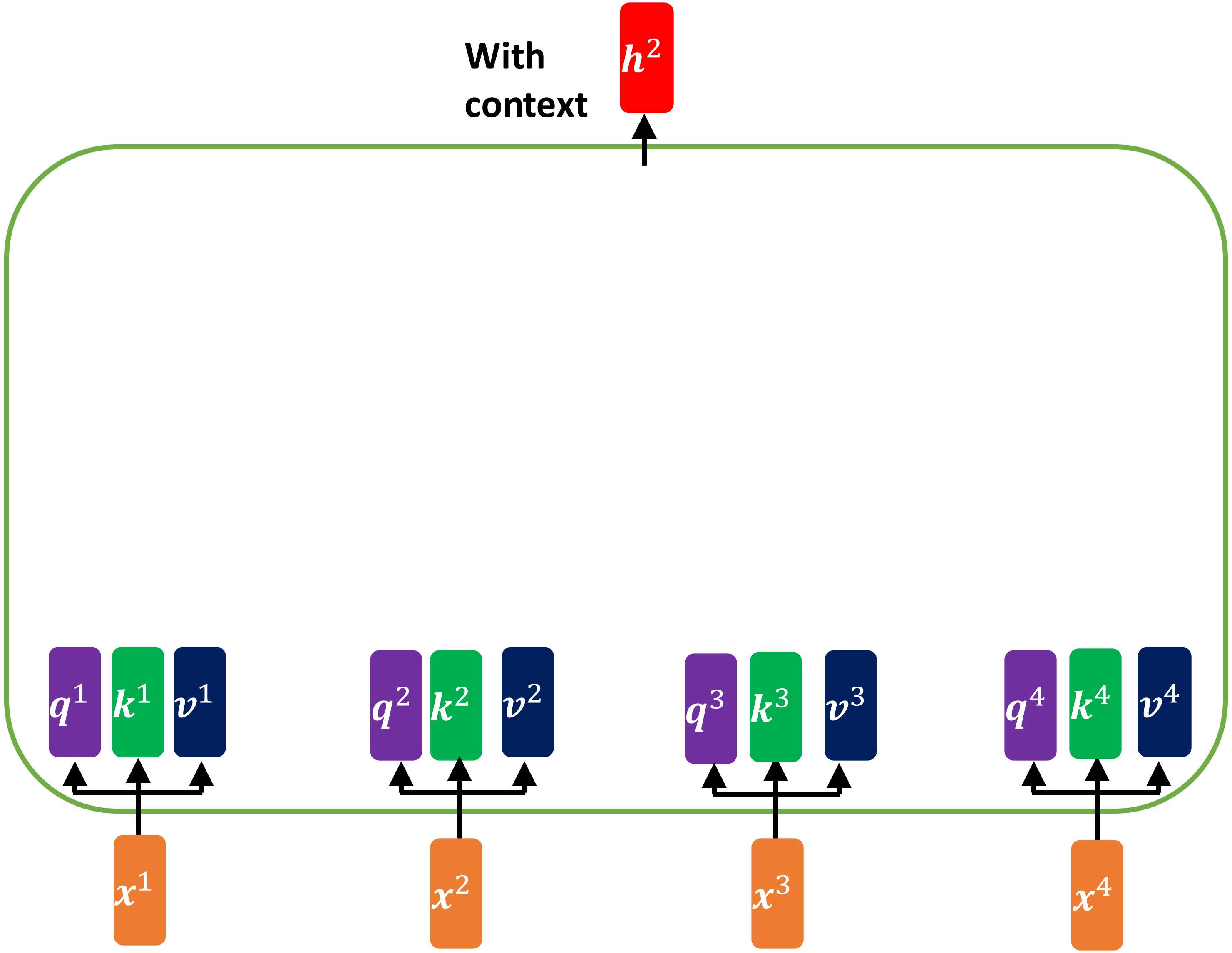
Transform input vectors into Query, Key, and Value using weights matrices (shared across inputs)

Query: $q^i = W^q x^i$

Key: $k^i = W^k x^i$

Value: $v^i = W^v x^i$

Self-Attention Layer



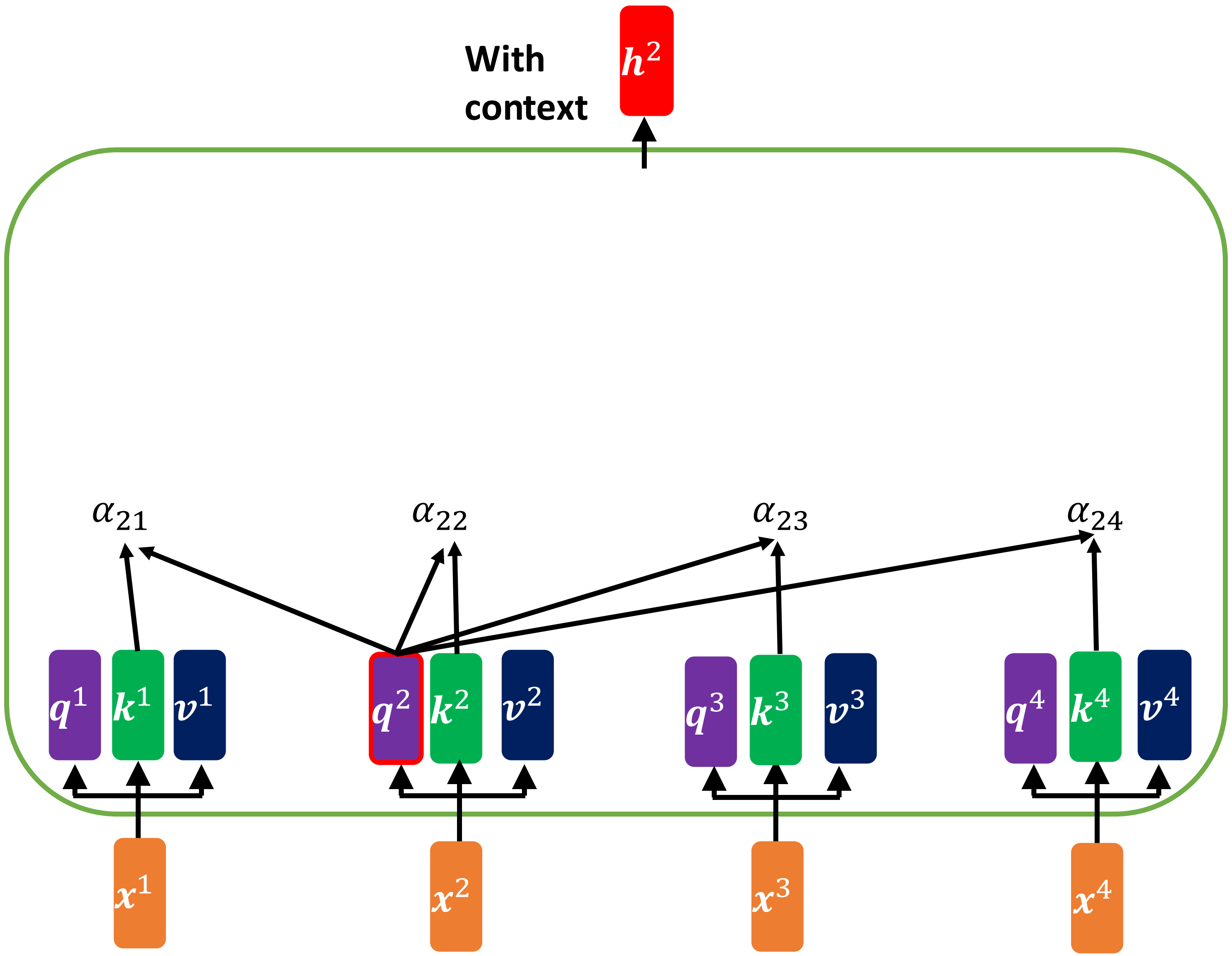
Step 1: Linear Projection
Transform input vectors into Query, Key, and Value using weights matrices (shared across inputs)

Query: $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key: $\boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{x}^i$

Value: $\boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{x}^i$

Self-Attention Layer



Step 2: Compute the attention scores:

$$\alpha_{2j} = \boldsymbol{k}^j \cdot \boldsymbol{q}^2 = (\boldsymbol{k}^j)^\top \boldsymbol{q}^2$$

Step 1: Linear Projection

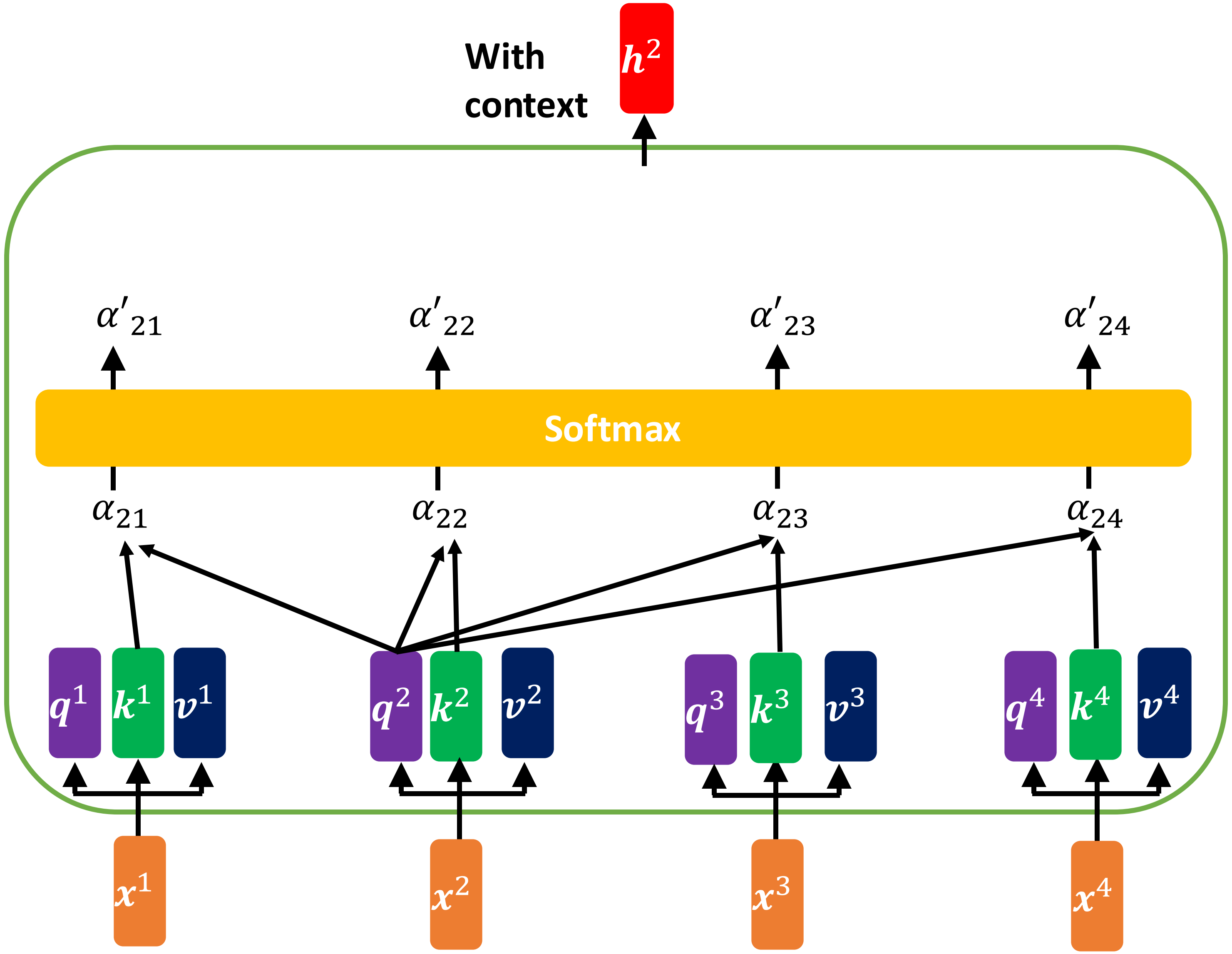
Transform input vectors into Query, Key, and Value using weights matrices (shared across inputs)

Query: $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key: $\boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{x}^i$

Value: $\boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{x}^i$

Self-Attention Layer



Step 3: Apply Softmax: $\alpha'_{2j} = \frac{e^{\alpha_{2j}}}{\sum_j e^{\alpha_{2j}}}$

Step 2: Compute the attention scores:
$\alpha_{2j} = \boldsymbol{k}^j \cdot \boldsymbol{q}^2 = (\boldsymbol{k}^j)^\top \boldsymbol{q}^2$

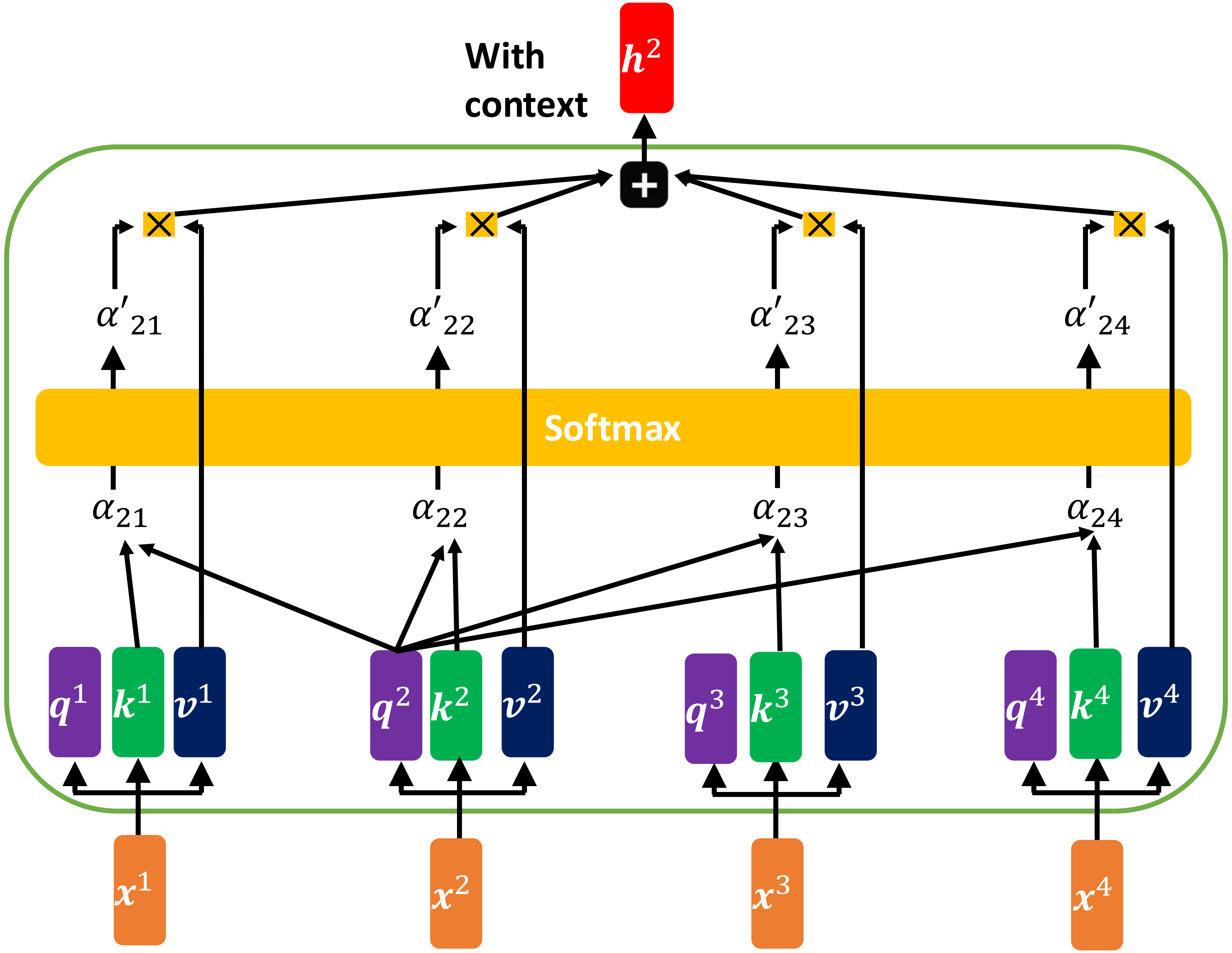
Step 1: Linear Projection
Transform input vectors into Query, Key, and Value using weights matrices (shared across inputs)

Query: $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key: $\boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{x}^i$

Value: $\boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{x}^i$

Self-Attention Layer



Step 4: Aggregate information:
Multiply Values by attention score (after Softmax)

$$\boldsymbol{h}^2 = \sum_j \alpha'_{2j} \boldsymbol{v}^j$$

Step 3: Apply Softmax: $\alpha'_{2j} = \frac{e^{\alpha_{2j}}}{\sum_j e^{\alpha_{2j}}}$

Step 2: Compute the attention scores:

$\alpha_{2j} = \boldsymbol{k}^j \cdot \boldsymbol{q}^2 = (\boldsymbol{k}^j)^\top \boldsymbol{q}^2$

Step 1: Linear Projection
Transform input vectors into Query, Key, and Value using weights matrices (shared across inputs)

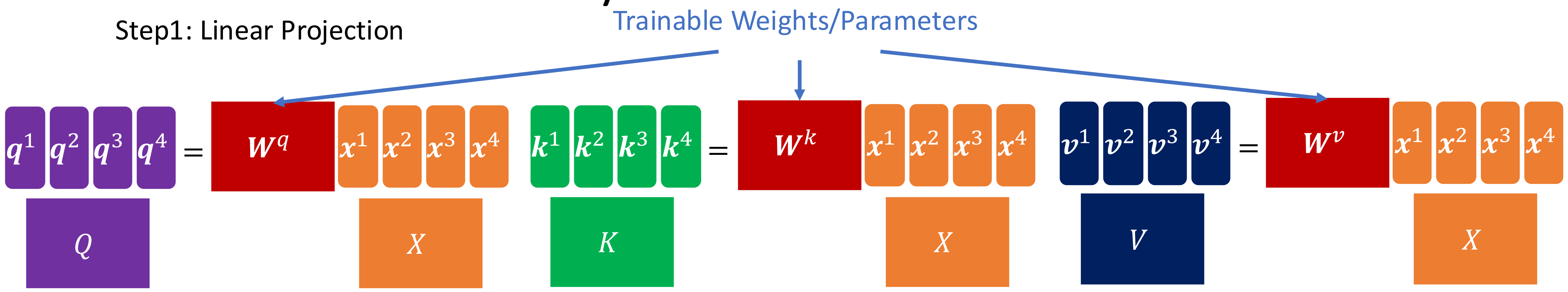
Query: $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key: $\boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{x}^i$

Value: $\boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{x}^i$

Self-Attention Layer

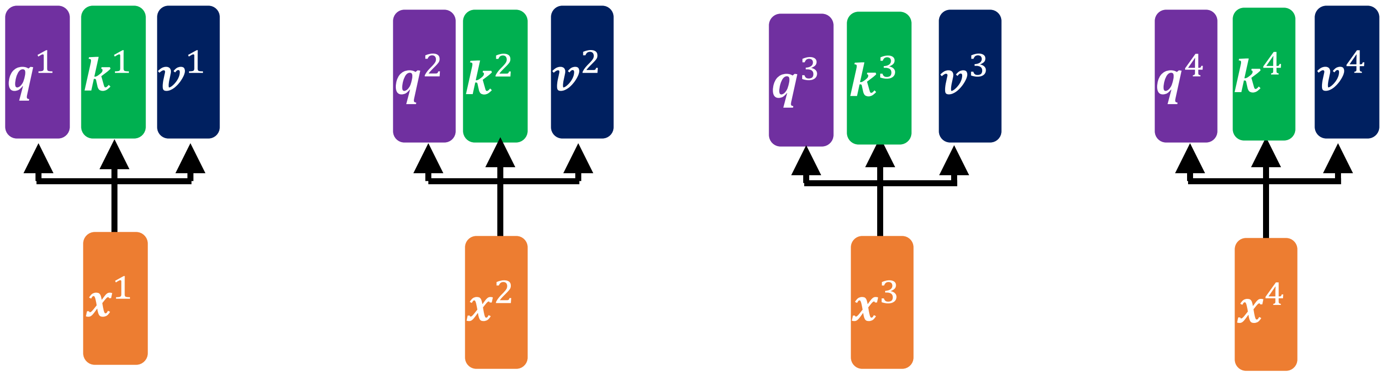
Step1: Linear Projection



Query $\boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{x}^i$

Key $\boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{x}^i$

Value $\boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{x}^i$



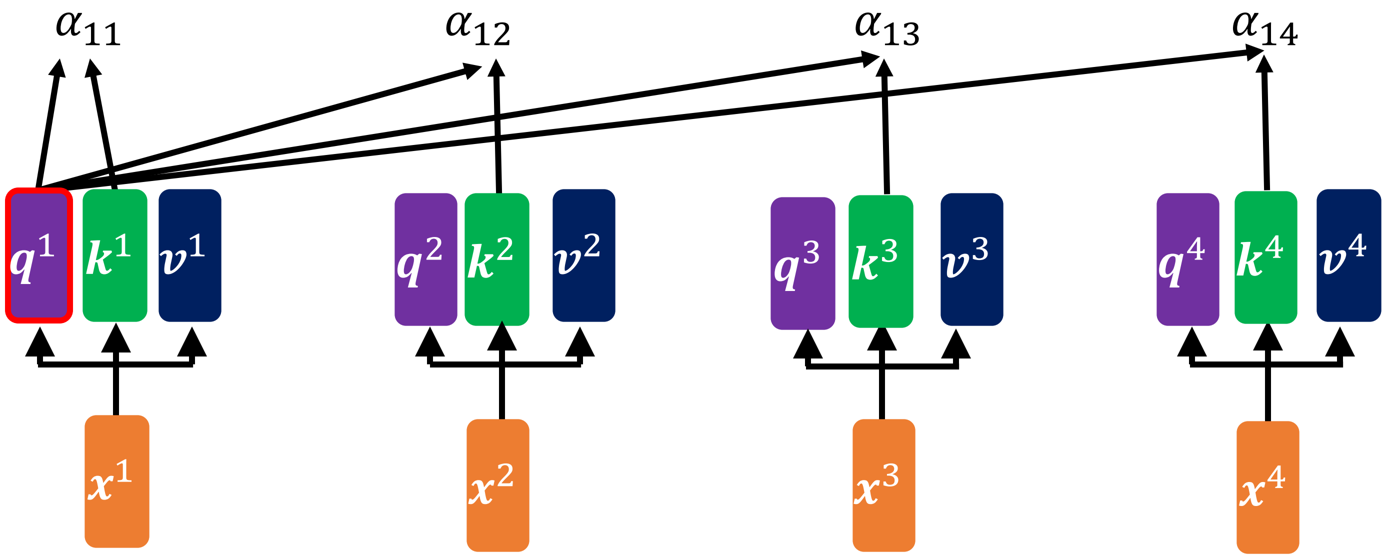
Self-Attention Layer

Step2: Compute the attention scores

$\alpha_{11} = (\boldsymbol{k}^1)^\top \boldsymbol{q}^1$ $\alpha_{12} = (\boldsymbol{k}^2)^\top \boldsymbol{q}^1$

$\alpha_{13} = (\boldsymbol{k}^3)^\top \boldsymbol{q}^1$ $\alpha_{14} = (\boldsymbol{k}^4)^\top \boldsymbol{q}^1$

$$\begin{bmatrix} \alpha_{11} \\ \alpha_{12} \\ \alpha_{13} \\ \alpha_{14} \end{bmatrix} = \begin{bmatrix} (\boldsymbol{k}^1)^\top \\ (\boldsymbol{k}^2)^\top \\ (\boldsymbol{k}^3)^\top \\ (\boldsymbol{k}^4)^\top \end{bmatrix} \boldsymbol{q}^1$$



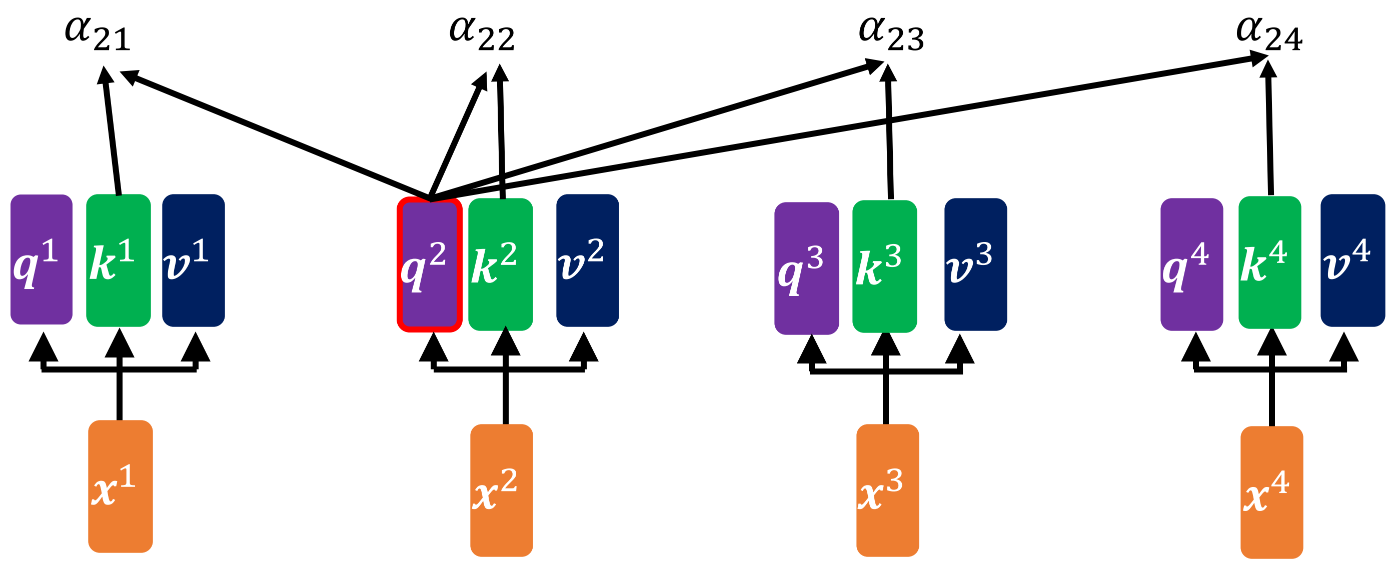
Self-Attention Layer

Step2: Compute the attention scores

$$\alpha_{21} = (k^1)^\top q^2 \qquad \alpha_{22} = (k^2)^\top q^2$$

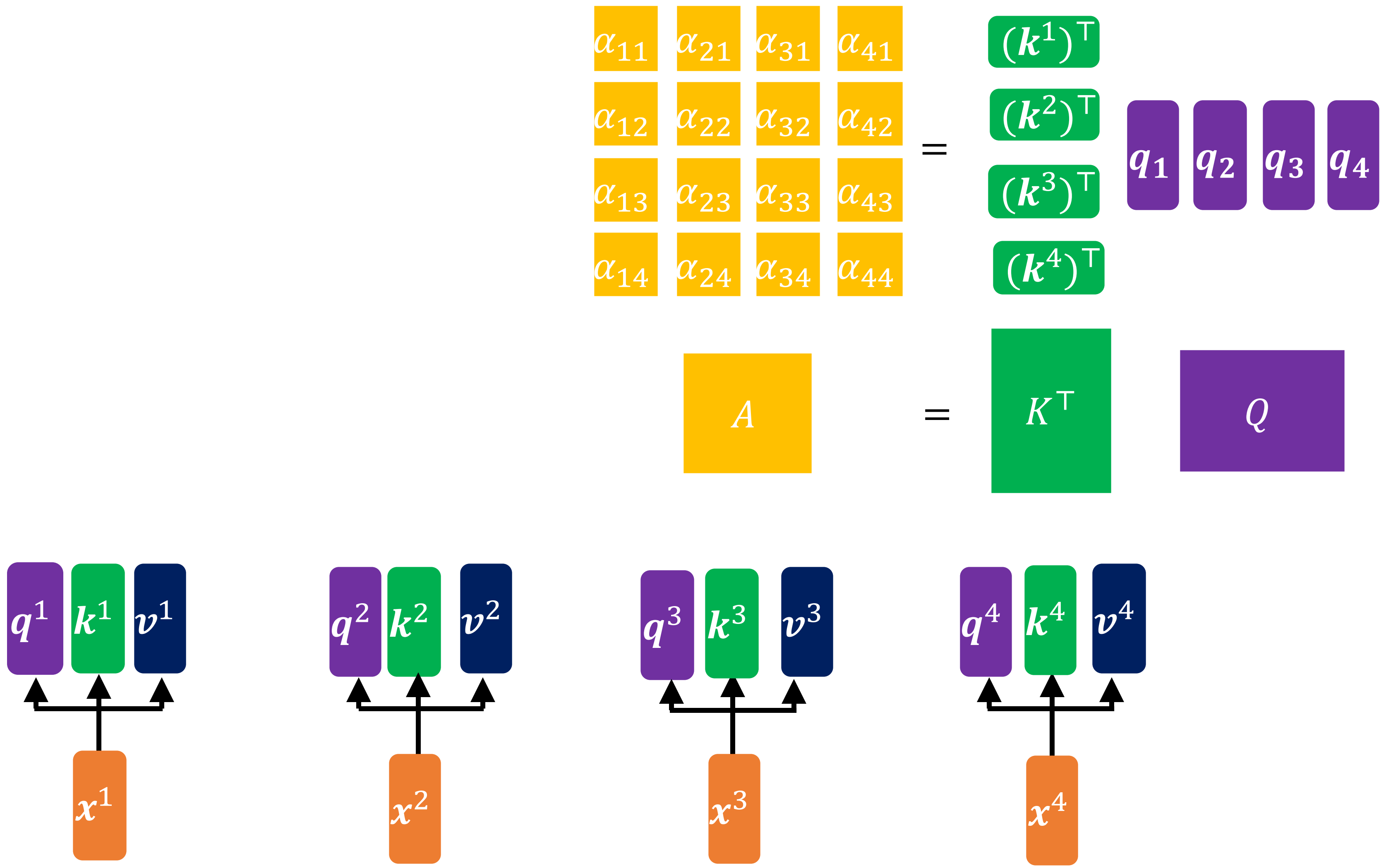
$$\alpha_{23} = (k^3)^\top q^2 \qquad \alpha_{24} = (k^4)^\top q^2$$

$$\begin{bmatrix} \alpha_{21} \\ \alpha_{22} \\ \alpha_{23} \\ \alpha_{24} \end{bmatrix} = \begin{bmatrix} (k^1)^\top \\ (k^2)^\top \\ (k^3)^\top \\ (k^4)^\top \end{bmatrix} q^2$$



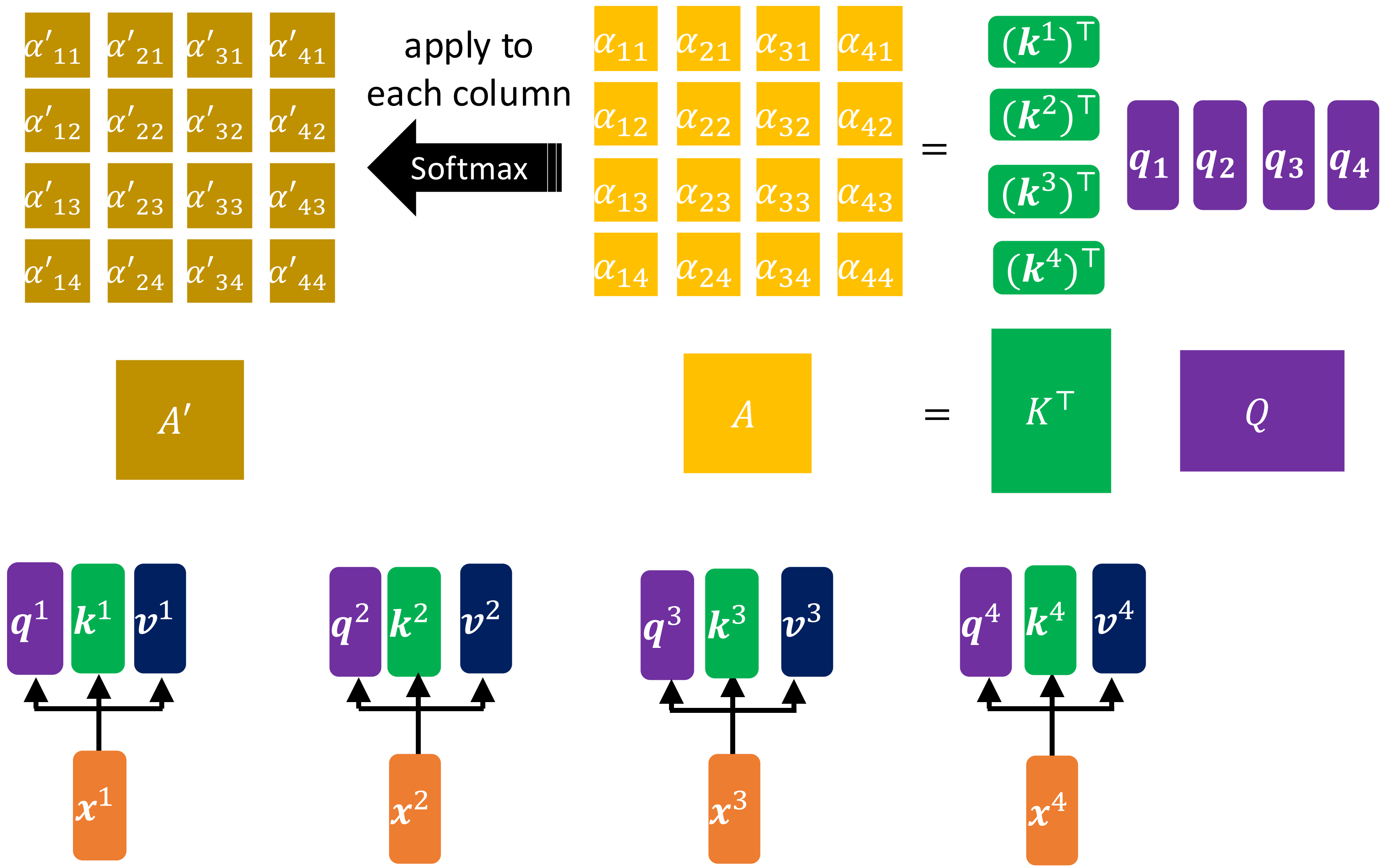
Self-Attention Layer

Step2: Compute the attention scores



Self-Attention Layer

Step3: Apply Softmax



Background: Weighted Sum of Vectors

- Given vectors $\boldsymbol{v}^1, \boldsymbol{v}^2, \ldots, \boldsymbol{v}^N$ and the corresponding weights for each vector $a^1, a^2, \ldots, a^N$, we can compute the weighted sum of vectors:

$$\boldsymbol{s} = \sum_{j=1}^{N} a^j \boldsymbol{v}^j = \begin{bmatrix} | & | & & | \\ \boldsymbol{v}^1 & \boldsymbol{v}^2 & \ldots & \boldsymbol{v}^N \\ | & | & & | \end{bmatrix} \begin{bmatrix} a^1 \\ a^2 \\ \vdots \\ a^N \end{bmatrix}$$

Stack the vectors as columns in a matrix

Arrange the weights as a column vector

- For example:

$$\boldsymbol{v}^1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \boldsymbol{v}^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \boldsymbol{v}^3 = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \quad \text{and } a^1 = 0.5,\ a^2 = 0.2,\ a^3 = 0.3$$

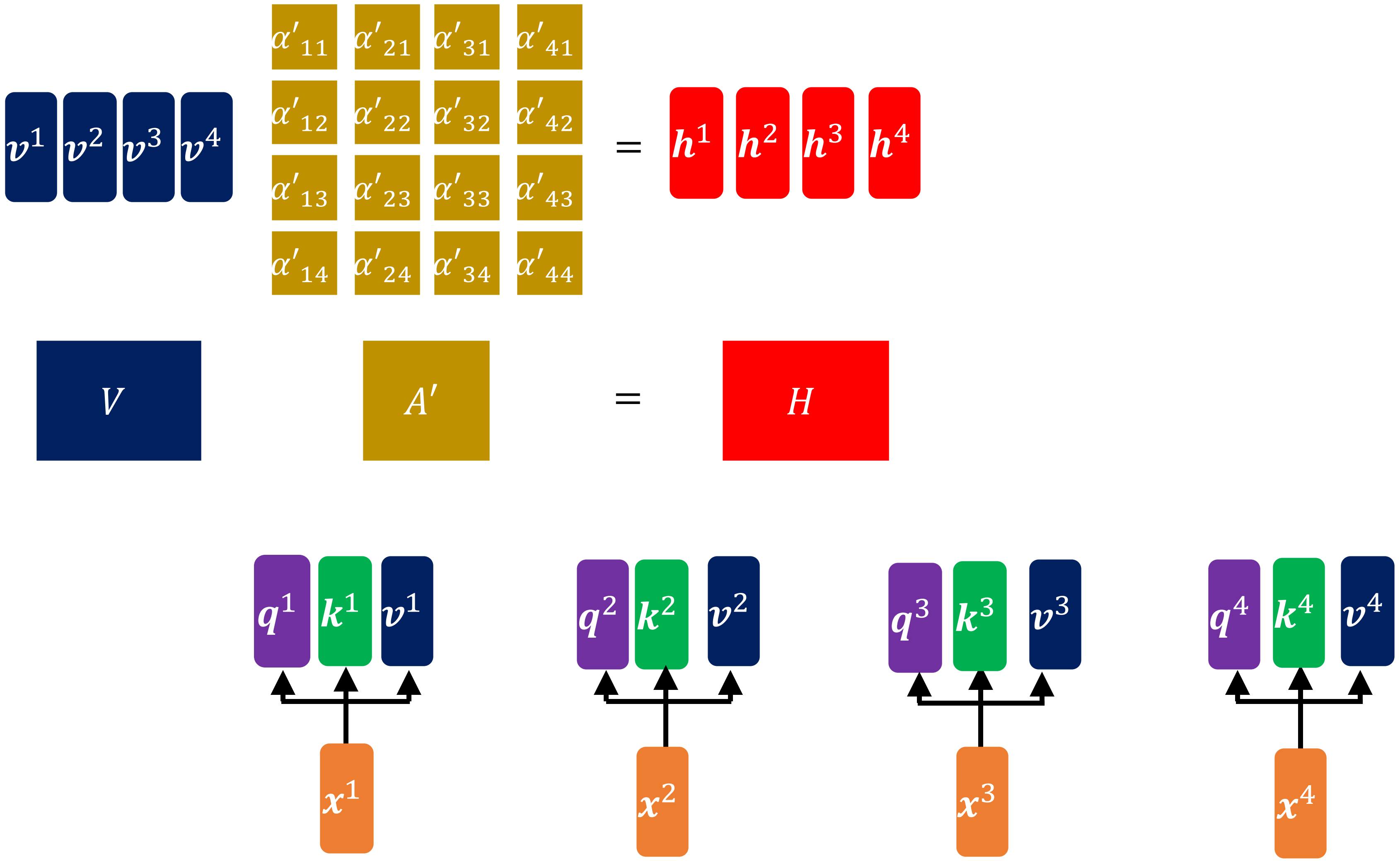
$$\boldsymbol{s} = 0.5 \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 0.2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 0.3 \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \times 0.5 \\ 2 \times 0.5 \end{bmatrix} + \begin{bmatrix} 1 \times 0.2 \\ 0 \times 0.2 \end{bmatrix} + \begin{bmatrix} 0 \times 0.3 \\ 2 \times 0.3 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 1.6 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 0 \\ 2 & 0 & 2 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.2 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 1 \times 0.5 + 1 \times 0.2 + 0 \times 0.3 \\ 2 \times 0.5 + 0 \times 0.2 + 2 \times 0.3 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 1.6 \end{bmatrix}$$

Self-Attention Layer

Step4: Aggregate Information: Multiply Values by attention score (after Softmax)

$$\boldsymbol{h}^i = \sum_j \alpha'_{ij} \boldsymbol{v}^j$$



Self-Attention Layer

$\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3, \boldsymbol{h}_4$
Can be generated together, not need to wait for previous time steps to complete first

Step1: Linear Projection

$$\boldsymbol{q}^1\ \boldsymbol{q}^2\ \boldsymbol{q}^3\ \boldsymbol{q}^4 = \boldsymbol{W}^q\ \boldsymbol{x}^1\ \boldsymbol{x}^2\ \boldsymbol{x}^3\ \boldsymbol{x}^4 \qquad \boldsymbol{k}^1\ \boldsymbol{k}^2\ \boldsymbol{k}^3\ \boldsymbol{k}^4 = \boldsymbol{W}^k\ \boldsymbol{x}^1\ \boldsymbol{x}^2\ \boldsymbol{x}^3\ \boldsymbol{x}^4 \qquad \boldsymbol{v}^1\ \boldsymbol{v}^2\ \boldsymbol{v}^3\ \boldsymbol{v}^4 = \boldsymbol{W}^v\ \boldsymbol{x}^1\ \boldsymbol{x}^2\ \boldsymbol{x}^3\ \boldsymbol{x}^4$$

$$Q \qquad X \qquad K \qquad X \qquad V \qquad X$$

Step2: Compute the attention scores

$$\begin{matrix} (\boldsymbol{k}^1)^\top \\ (\boldsymbol{k}^2)^\top \\ (\boldsymbol{k}^3)^\top \\ (\boldsymbol{k}^4)^\top \end{matrix}\ \boldsymbol{q}^1\ \boldsymbol{q}^2\ \boldsymbol{q}^3\ \boldsymbol{q}^4 = \begin{matrix} \alpha_{11} & \alpha_{21} & \alpha_{31} & \alpha_{41} \\ \alpha_{12} & \alpha_{22} & \alpha_{32} & \alpha_{42} \\ \alpha_{13} & \alpha_{23} & \alpha_{33} & \alpha_{43} \\ \alpha_{14} & \alpha_{24} & \alpha_{34} & \alpha_{44} \end{matrix}$$

$$K^\top \quad Q = A$$

Step3: Apply Softmax

$$\xrightarrow{\text{Softmax}} \begin{matrix} \alpha'_{11} & \alpha'_{21} & \alpha'_{31} & \alpha'_{41} \\ \alpha'_{12} & \alpha'_{22} & \alpha'_{32} & \alpha'_{42} \\ \alpha'_{13} & \alpha'_{23} & \alpha'_{33} & \alpha'_{43} \\ \alpha'_{14} & \alpha'_{24} & \alpha'_{34} & \alpha'_{44} \end{matrix}$$

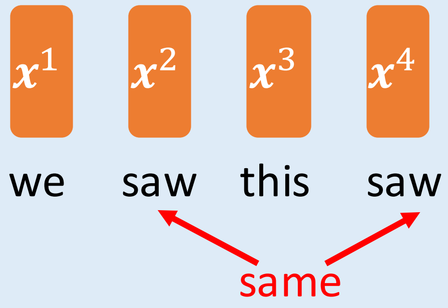
$$A'$$

Step4: : Aggregate Information

$$\boldsymbol{v}^1\ \boldsymbol{v}^2\ \boldsymbol{v}^3\ \boldsymbol{v}^4\ \begin{matrix} \alpha'_{11} & \alpha'_{21} & \alpha'_{31} & \alpha'_{41} \\ \alpha'_{12} & \alpha'_{22} & \alpha'_{32} & \alpha'_{42} \\ \alpha'_{13} & \alpha'_{23} & \alpha'_{33} & \alpha'_{43} \\ \alpha'_{14} & \alpha'_{24} & \alpha'_{34} & \alpha'_{44} \end{matrix} = \boldsymbol{h}^1\ \boldsymbol{h}^2\ \boldsymbol{h}^3\ \boldsymbol{h}^4$$

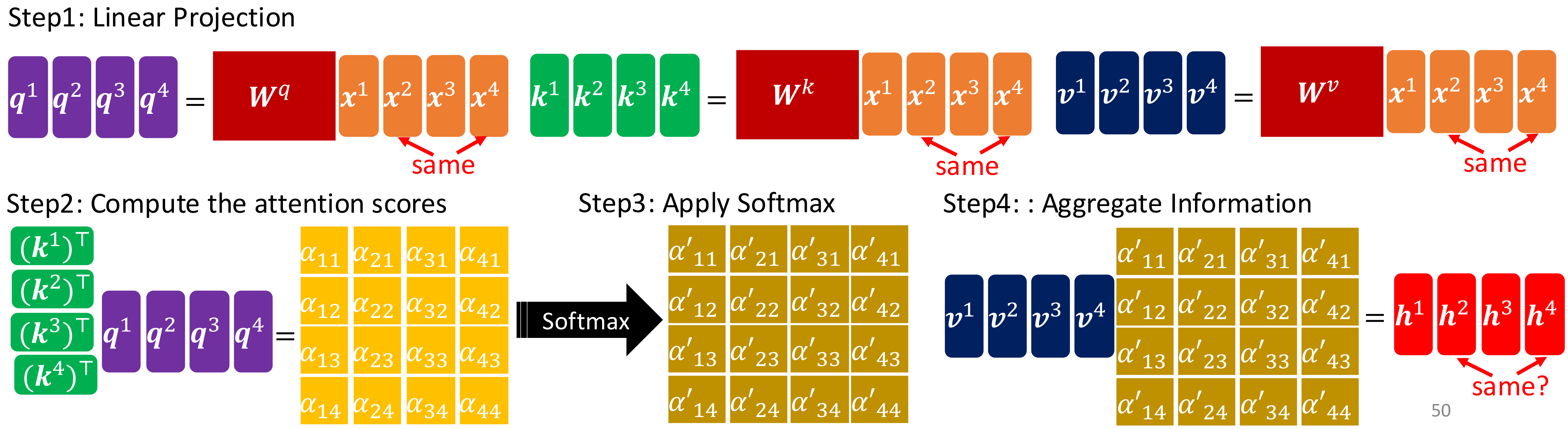
$$V \quad A' = H$$

Poll Everywhere



PollEv.com/conghuihu365

- Consider the sentence “we saw this saw”. If we use one-hot encoding to represent the words and input these one-hot vectors into a self-attention layer, can the self-attention layer generate **different** outputs for the two instances of “saw”?



Self-Attention Layer: Example

x^1 x^2 x^3 x^4

we saw this saw

same

Step1: Linear Projection

$$\begin{bmatrix} q^1 & q^2 & q^3 & q^4 \end{bmatrix} = W^q \begin{bmatrix} x^1 & x^2 & x^3 & x^4 \end{bmatrix}$$

same same

$$\begin{bmatrix} k^1 & k^2 & k^3 & k^4 \end{bmatrix} = W^k \begin{bmatrix} x^1 & x^2 & x^3 & x^4 \end{bmatrix}$$

same same

$$\begin{bmatrix} v^1 & v^2 & v^3 & v^4 \end{bmatrix} = W^v \begin{bmatrix} x^1 & x^2 & x^3 & x^4 \end{bmatrix}$$

same same

Step2: Compute the attention scores

$$\begin{bmatrix} (k^1)^\top \\ (k^2)^\top \\ (k^3)^\top \\ (k^4)^\top \end{bmatrix} \begin{bmatrix} q^1 & q^2 & q^3 & q^4 \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \boxed{\alpha_{21}} & \alpha_{31} & \boxed{\alpha_{41}} \\ \alpha_{12} & \boxed{\alpha_{22}} & \alpha_{32} & \boxed{\alpha_{42}} \\ \alpha_{13} & \boxed{\alpha_{23}} & \alpha_{33} & \boxed{\alpha_{43}} \\ \alpha_{14} & \boxed{\alpha_{24}} & \alpha_{34} & \boxed{\alpha_{44}} \end{bmatrix}$$

same same

Step3: Apply Softmax

$$\xrightarrow{\text{Softmax}} \begin{bmatrix} \alpha'_{11} & \boxed{\alpha'_{21}} & \alpha'_{31} & \boxed{\alpha'_{41}} \\ \alpha'_{12} & \boxed{\alpha'_{22}} & \alpha'_{32} & \boxed{\alpha'_{42}} \\ \alpha'_{13} & \boxed{\alpha'_{23}} & \alpha'_{33} & \boxed{\alpha'_{43}} \\ \alpha'_{14} & \boxed{\alpha'_{24}} & \alpha'_{34} & \boxed{\alpha'_{44}} \end{bmatrix}$$

same

Step4: : Aggregation Information

$$\begin{bmatrix} v^1 & v^2 & v^3 & v^4 \end{bmatrix} \begin{bmatrix} \alpha'_{11} & \boxed{\alpha'_{21}} & \alpha'_{31} & \boxed{\alpha'_{41}} \\ \alpha'_{12} & \boxed{\alpha'_{22}} & \alpha'_{32} & \boxed{\alpha'_{42}} \\ \alpha'_{13} & \boxed{\alpha'_{23}} & \alpha'_{33} & \boxed{\alpha'_{43}} \\ \alpha'_{14} & \boxed{\alpha'_{24}} & \alpha'_{34} & \boxed{\alpha'_{44}} \end{bmatrix} = \begin{bmatrix} h^1 & h^2 & h^3 & h^4 \end{bmatrix}$$

same same

Cannot distinguish between the first “saw” (verb) and the second “saw” (noun)!

Need to add positional information to differentiate the two “saw”!

Positional Encoding

- Positional encoding explicitly injects positional information into the original input features (e.g., one-hot vectors):

$$\boldsymbol{x}'^{i} = \boldsymbol{x}^{i} + PE^{i}$$

Unique for each position!

$\boldsymbol{x}^{i}$: Original input feature vector for position i, where $\boldsymbol{x}^{i} \in \mathbb{R}^{d}$.
PE^{i}: Positional encoding vector for position i, where $PE^{i} \in \mathbb{R}^{d}$
$\boldsymbol{x}'^{i}$: Final input feature vector after adding positional encoding.

Positional Encoding: $\mathbf{sin}$ and $\mathbf{cos}$ Funtions

- $\sin$ and $\cos$ functions can be used to generate the positional encoding as follows:

$$PE(i, 2k) = \sin\left(\frac{i}{10000^{\frac{2k}{d}}}\right), \qquad PE(i, 2k+1) = \cos\left(\frac{i}{10000^{\frac{2k}{d}}}\right)$$

i: The position index.

$2k$ and $2k+1$: The dimension indices of the positional encoding vector.

k starts from 0 and goes up to $\frac{d}{2} - 1$, where d is the input feature dimension.

Source: https://arxiv.org/pdf/1706.03762

Positional Encoding: **sin** and **cos** Funtions

- Suppose the input feature dimension is $d = 4$,
- PE^1 can be generate as follows:

$$PE(i, 2k) = \sin\left(\frac{i}{10000^{\frac{2k}{d}}}\right)$$

$$PE(i, 2k+1) = \cos\left(\frac{i}{10000^{\frac{2k}{d}}}\right)$$

$$0 \leq k \leq \frac{d}{2} - 1$$

$$0 \leq k \leq \frac{4}{2} - 1 = 1$$

- $i = 1,\ k = 0$
- $PE(1, 0) = \sin\left(\frac{1}{10000^{\frac{2\times 0}{4}}}\right) = \sin\left(\frac{1}{10000^{\frac{0}{4}}}\right) = \sin(1).$
- $PE(1, 1) = \cos\left(\frac{1}{10000^{\frac{2\times 0}{4}}}\right) = \cos(\frac{1}{10000^{\frac{0}{4}}}) = \cos(1)$

- $i = 1,\ k = 1$
- $PE(1, 2) = \sin\left(\frac{1}{10000^{\frac{2\times 1}{4}}}\right) = \sin(\frac{1}{10000^{\frac{1}{2}}}) = \sin(\frac{1}{100}).$
- $PE(1, 3) = \cos\left(\frac{1}{10000^{\frac{2\times 1}{4}}}\right) = \cos(\frac{1}{10000^{\frac{1}{2}}}) = \cos(\frac{1}{100})$

$$PE^1 = \begin{bmatrix} \sin(1) \\ \cos(1) \\ \sin(\frac{1}{100}) \\ \cos(\frac{1}{100}) \end{bmatrix}$$

Positional Encoding: $\sin$ and $\cos$ Funtions

$$PE(i, 2k) = \sin\left(\frac{i}{10000^{\frac{2k}{d}}}\right)$$

$$PE(i, 2k+1) = \cos\left(\frac{i}{10000^{\frac{2k}{d}}}\right)$$

$$0 \le k \le \frac{d}{2} - 1$$

$$0 \le k \le \frac{4}{2} - 1 = 1$$

- Suppose the input feature dimension is $d = 4$,
- PE^2 can be generate as follows:

- $i = 2,\ k = 0$
- $PE(2,0) = \sin\left(\frac{2}{10000^{\frac{2\times 0}{4}}}\right) = \sin\left(\frac{2}{10000^{\frac{0}{4}}}\right) = \sin(2).$
- $PE(2,1) = \cos\left(\frac{2}{10000^{\frac{2\times 0}{4}}}\right) = \cos(\frac{2}{10000^{\frac{0}{4}}}) = \cos(2)$

- $i = 2,\ k = 1$
- $PE(2,2) = \sin\left(\frac{2}{10000^{\frac{2\times 1}{4}}}\right) = \sin(\frac{2}{10000^{\frac{1}{2}}}) = \sin\left(\frac{2}{100}\right) = \sin\left(\frac{1}{50}\right).$
- $PE(2,3) = \cos\left(\frac{2}{10000^{\frac{2\times 1}{4}}}\right) = \cos(\frac{2}{10000^{\frac{1}{2}}}) = \cos(\frac{2}{100})) = \cos(\frac{1}{50})$

$$PE^2 = \begin{bmatrix} \sin(2) \\ \cos(2) \\ \sin(\frac{1}{50}) \\ \cos(\frac{1}{50}) \end{bmatrix}$$

Outline

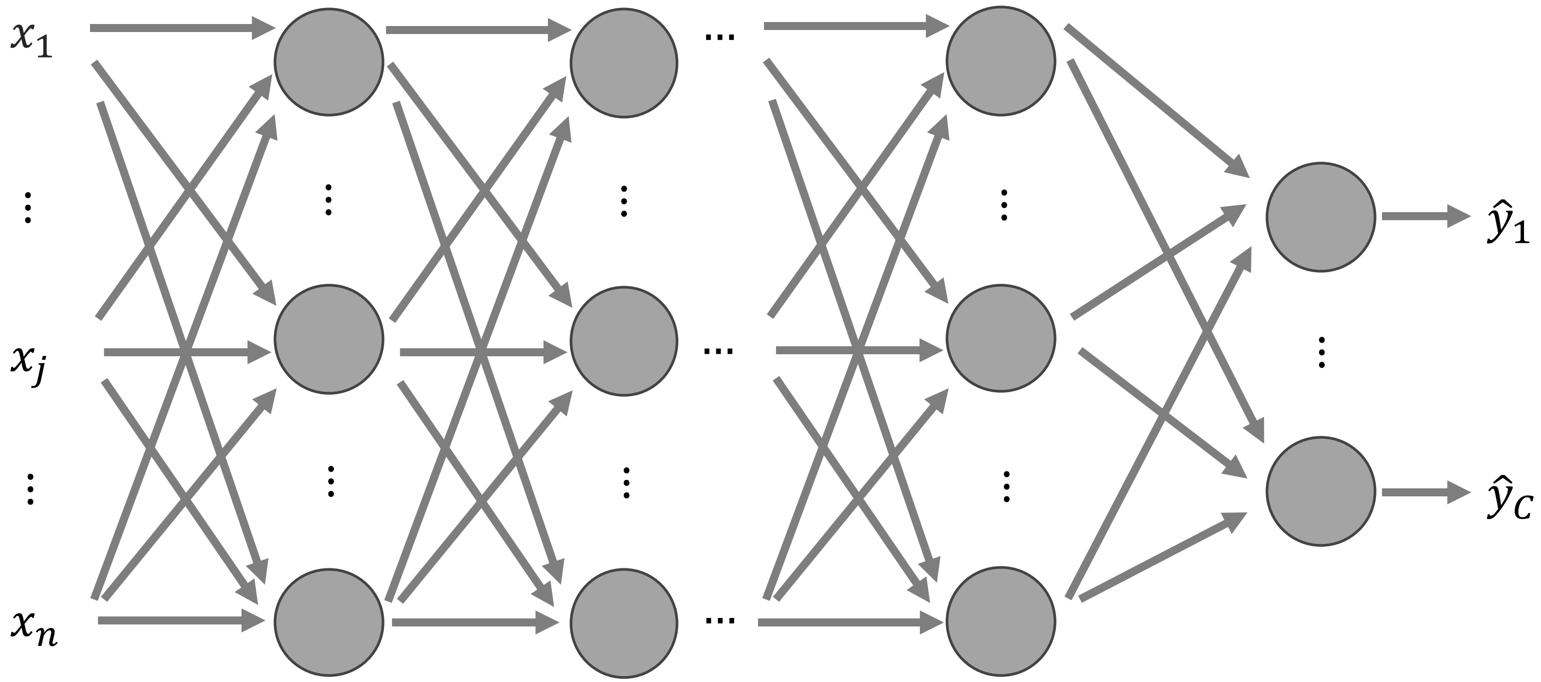
- Recurrent Neural Networks
  - Motivation
  - Recurrent Neural Networks
  - Applications
- Self-Attention
  - Self-Attention Layer
  - Positional Encoding
- **Issues with Deep Learning**
  - Overfitting
  - Vanishing/Exploding Gradient

Issues with Deep Learning

- Overfitting
- Gradient Vanishing/Exploding

Dropout

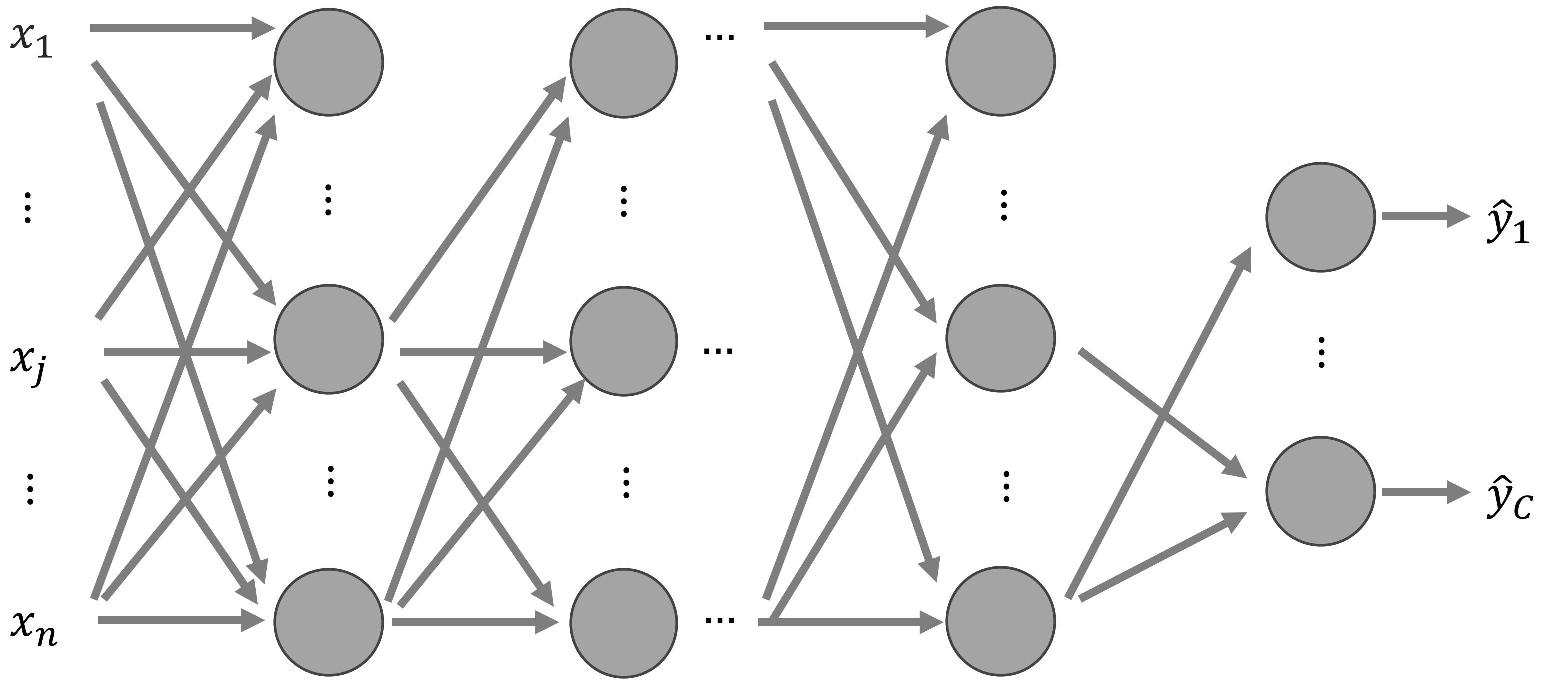
Prevent Overfitting



During training, **randomly** set some neurons' output to 0

Dropout

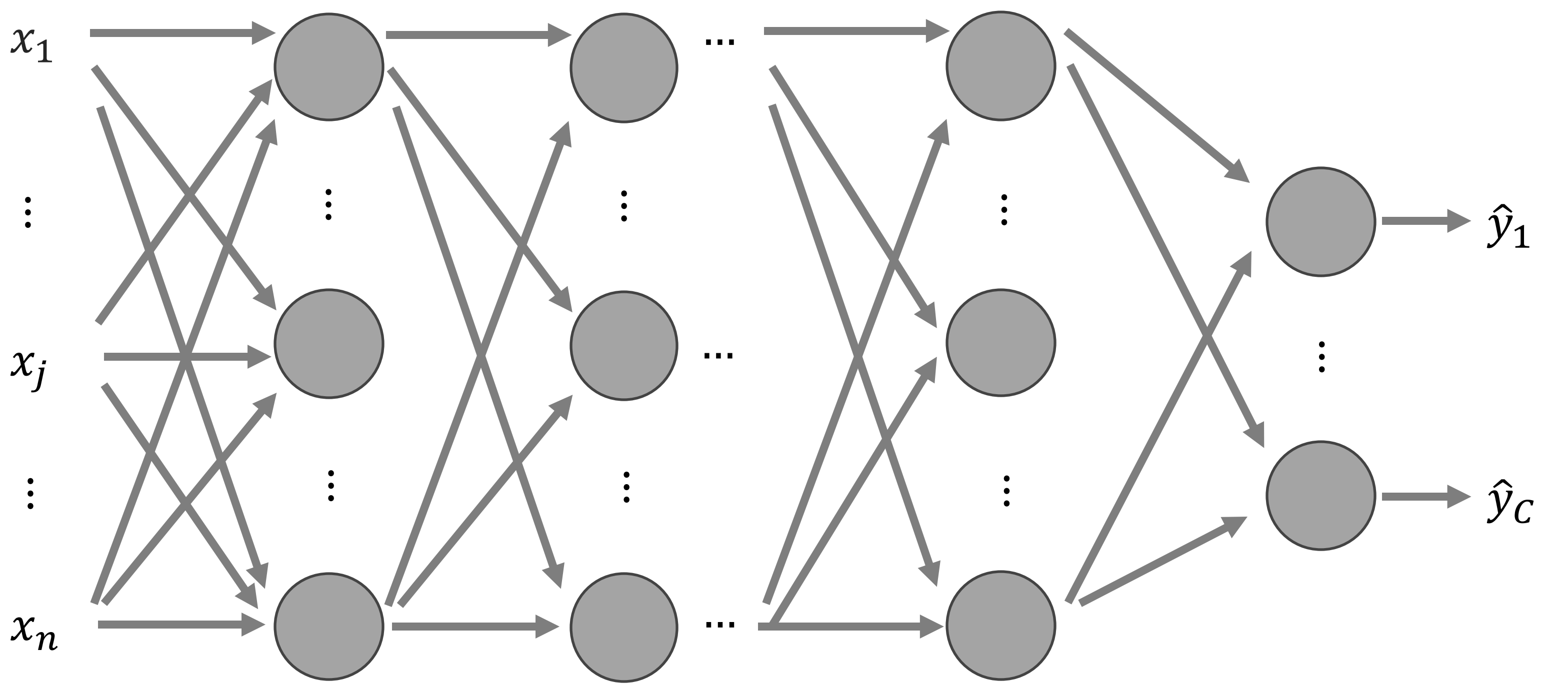
Prevent Overfitting



During training, **randomly** set some neurons' output to 0

Dropout

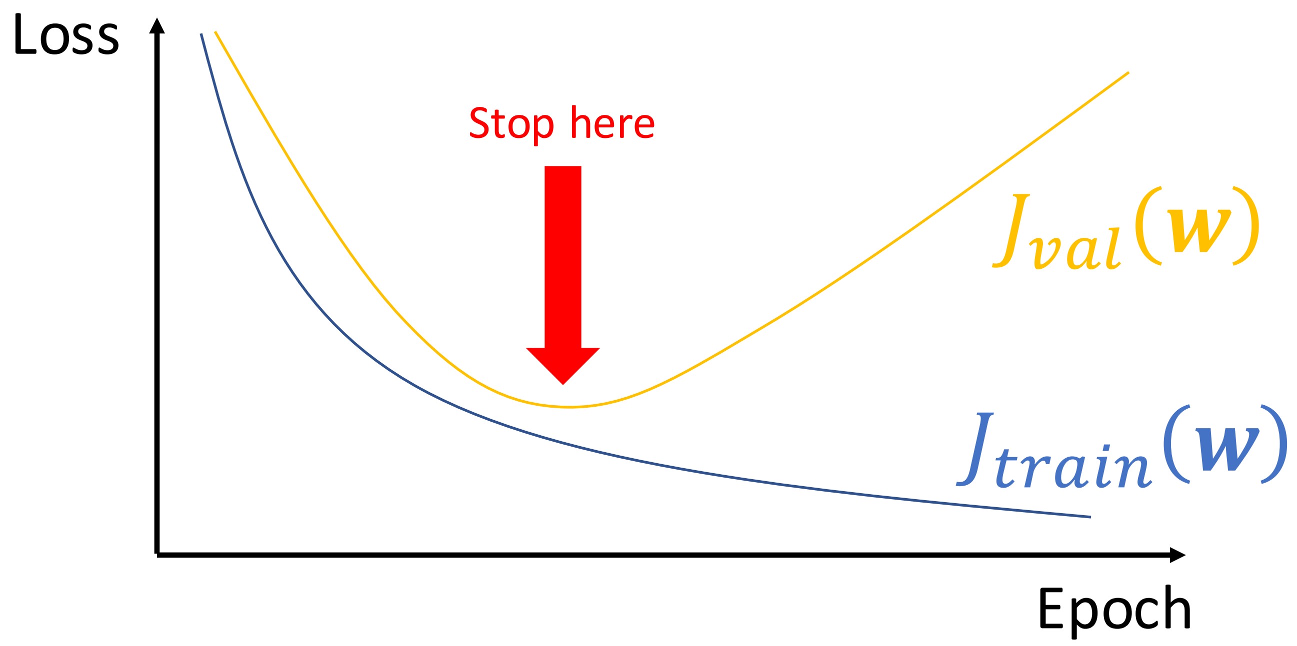
Prevent Overfitting



During training, **randomly** set some neurons' output to 0

Early Stopping

Prevent Overfitting



Vanishing/Exploding Gradient

- **Vanishing gradient:**
  - Small gradients got multiplied again and again until it reaches almost zero.
  - Mitigation: Change Activation Functions (Will be discussed during tutorial)

- **Exploding gradient:**
  - Large gradients got multiplied again and again until it overflows.
  - Mitigation: Gradient Clipping ~ clip gradient within range [-clip_value, clip_value]

Image credit:
https://neptune.ai/blog/vanishing-and-exploding-gradients-debugging-monitoring-fixing
https://medium.com/geekculture/how-to-deal-with-vanishing-and-exploding-gradients-in-neural-networks-24eb00c80e84

Summary

- **Recurrent Neural Networks**
  - Recurrent Neural Networks can capture contextual information.
  - RNN types: Many-to-Many, Many-to-One, and One-to-Many
  - Applications: sentiment analysis, speech recognition, etc
- **Self-Attention**
  - Self-Attention layer can handle sequential data in parallel.
  - Positional Encoding can be used to encode the positional information.
- **Issues with Deep Learning**
  - Overfitting: Dropout, Early Stopping
  - Vanishing Gradient: Change activation function
  - Exploding Gradient: Gradient Clipping

Coming Up Next Week

- **Transformer**
- **AI Ethics**
- **…**

To Do

- **Lecture Training 11**
  - +250 Free EXP
  - +100 Early bird bonus