CS5733 Program Synthesis

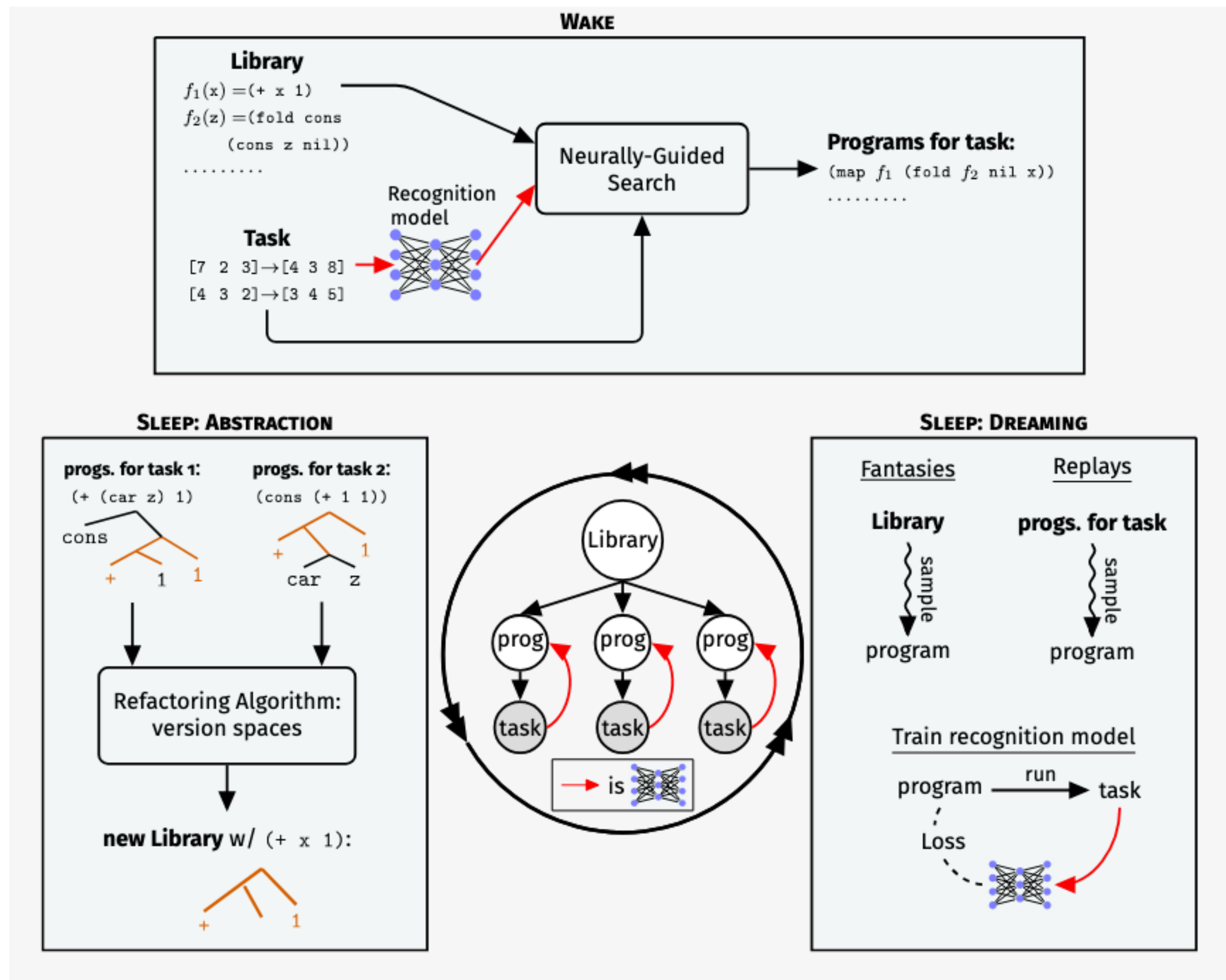
#22. Introduction to Deep Learning for NLP/Sequence Modeling

Ashish Mishra, November 8, 2024

Slides from MIT DeepLearning 6S191

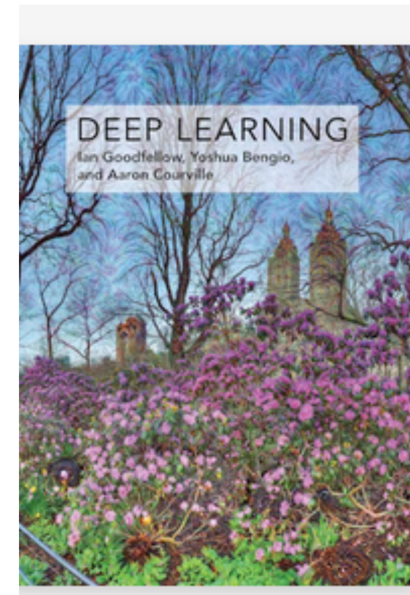
© Alexander Amini and Ava Amini
MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

DreamCoder.



Sources for the Class

```
0.@book{Goodfellow-et-al-2016,
    title={Deep Learning},
    author={Ian Goodfellow and Yoshua Bengio and Aaron Courville},
    publisher={MIT Press},
    note={\url{http://www.deeplearningbook.org}},
    year={2016}
}
```



1. Sepp Hochreiter, Jürgen Schmidhuber, *Long Short-term Memory*, 1997(bibtex)

2. Ilya Sutskever, Oriol Vinyals, Quoc V. Le, *Sequence to Sequence Learning with Neural Networks*, 2014(bibtex)

3. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, \L ukasz Kaiser, Illia Polosukhin, *Attention is All you Need*, 2017(bibtex)

4. **Online Course: MIT 6.S191: Introduction to Deep Learning,**

Borrowing Slides from Sources 0 and 4 Majorly

Outline

1. NLP : What?
    1. N-Gram Models
2. Intro Deep Learning for NLP
    1. Where DL sits in the Machine Learning landscape.
    2. Perceptrons/Neurons, Feed Forward units
    3. Neural Networks, Deep Networks and Training
    4. NNs for Sequence Modeling , RNN
    5. Attention/Self-Attention
    6. LSTMs (tangentially)
    7. Encoder-Decoders

NLP

- Natural language processing (NLP) is the use of human languages, such as English or French, by a computer.

- Many NLP applications are based on language models that define a probability distribution over sequences of words, characters or bytes in a natural language.

- In theory, very generic neural network techniques can be successfully applied to natural language processing,
  - However this will not scale we must usually use techniques that are specialized for processing sequential data.
  - Because the total number of possible words is so large, word-based language models must operate on an extremely high-dimensional and sparse discrete space.
  - Several strategies have been developed to make models of such a space efficient, both in a computational and in a statistical sense.

N-grams

- A **language model** defines a probability distribution over sequences of tokens in a natural language
  - a token may be a word, a character, or even a byte.
  - earliest successful language models were based on models of fixed-length sequences of tokens called n-grams

- An n-gram is a sequence of n tokens.

- For small values of n, models have particular names: unigram for n=1, bigram for n=2, and trigram for n=3

- Training n-gram models is straightforward because the maximum likelihood estimate can be computed simply by counting how many times each possible n gram occurs in the training set

N-grams

The big brown bear scares the children with its roar

$P(scares \mid bear,\ brown)$

Probability of a word depends on the previous n words

Represented with a table: $P\left(w_i \mid w_{i-1},\ w_{i-2},\ \ldots,\ w_{i-n}\right)$

Bigger n makes more accurate, but also more difficult to learn, requires much bigger table

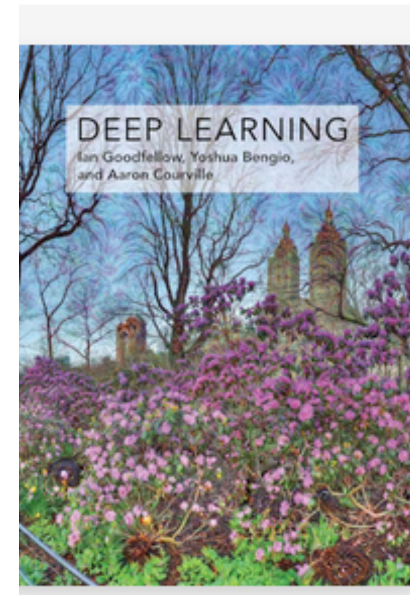
Downsides

- some words require more context than others
- some words carry very little information . E.g roar vs. bear

Neural Language Models

Sources for the Class

```
0.@book{Goodfellow-et-al-2016,
    title={Deep Learning},
    author={Ian Goodfellow and Yoshua Bengio and Aaron Courville},
    publisher={MIT Press},
    note={\url{http://www.deeplearningbook.org}},
    year={2016}
}
```



1. Sepp Hochreiter, Jürgen Schmidhuber, *Long Short-term Memory*, 1997(bibtex)

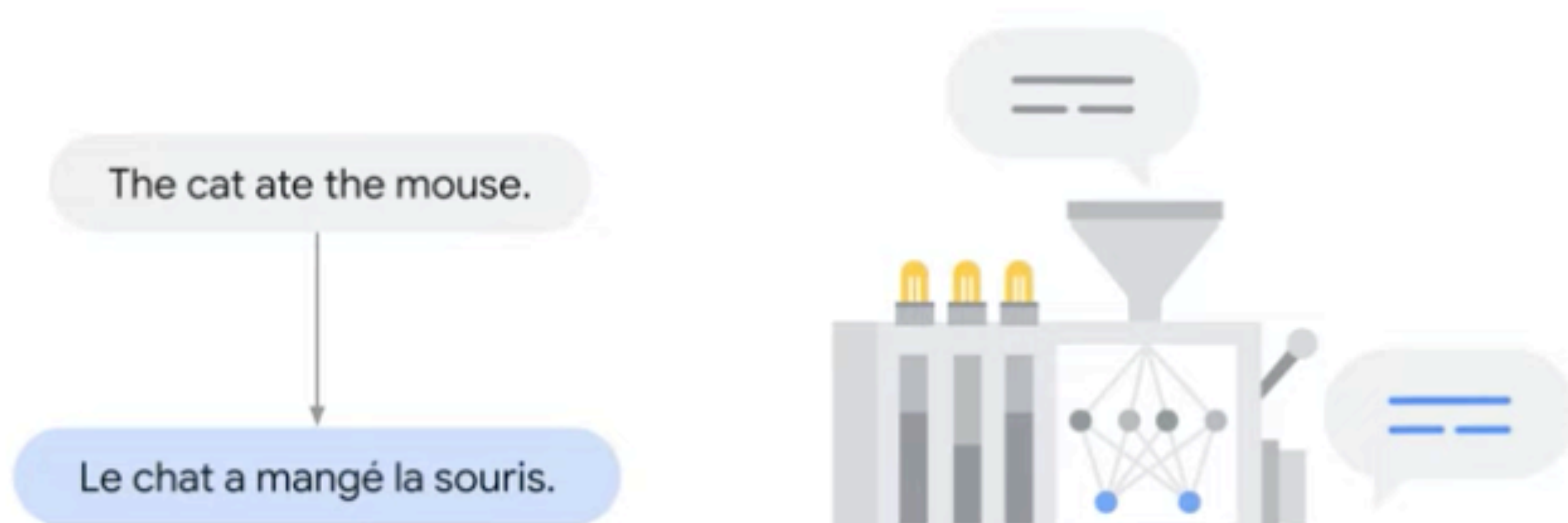
2. Ilya Sutskever, Oriol Vinyals, Quoc V. Le, *Sequence to Sequence Learning with Neural Networks*, 2014(bibtex)

3. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, \L ukasz Kaiser, Illia Polosukhin, *Attention is All you Need*, 2017(bibtex)

4. **Online Course: MIT 6.S191: Introduction to Deep Learning,**

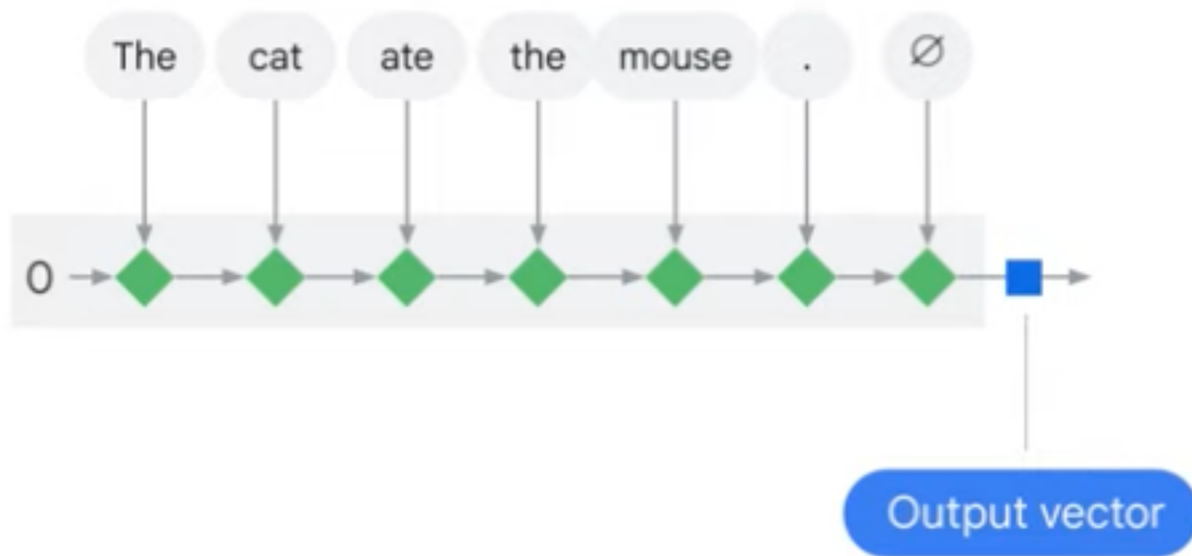
Borrowing Slides from Sources 0 and 4 Majorly

Encoder-Decoder Architecture Overview



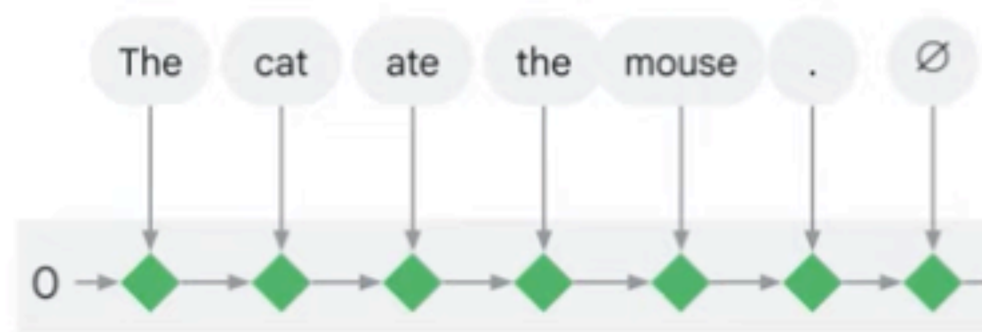
The two stages

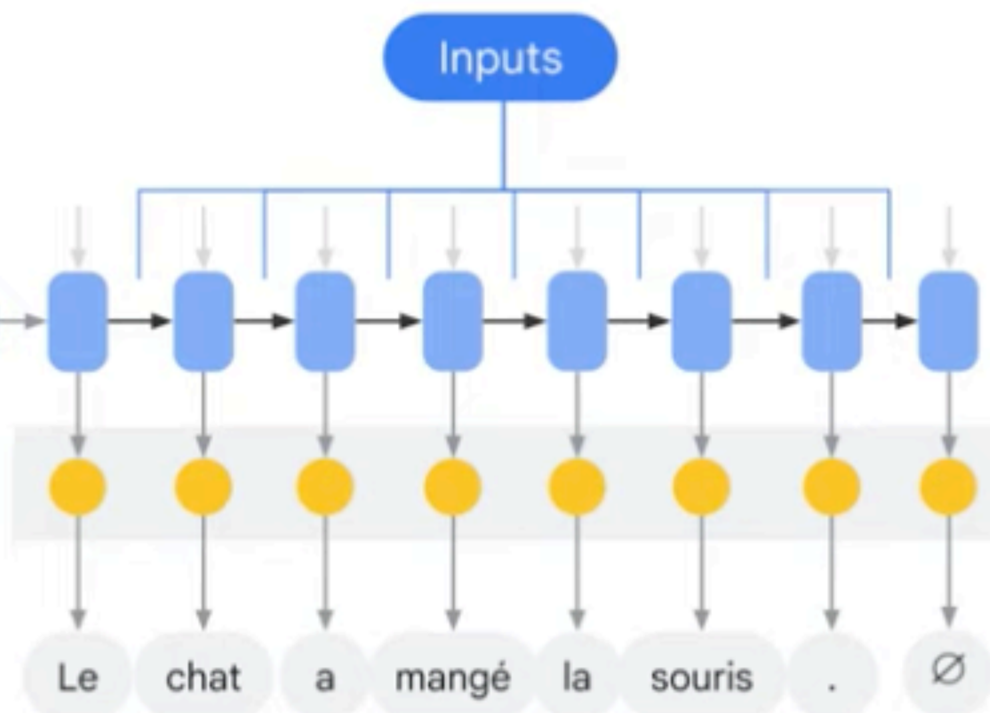
Encoder



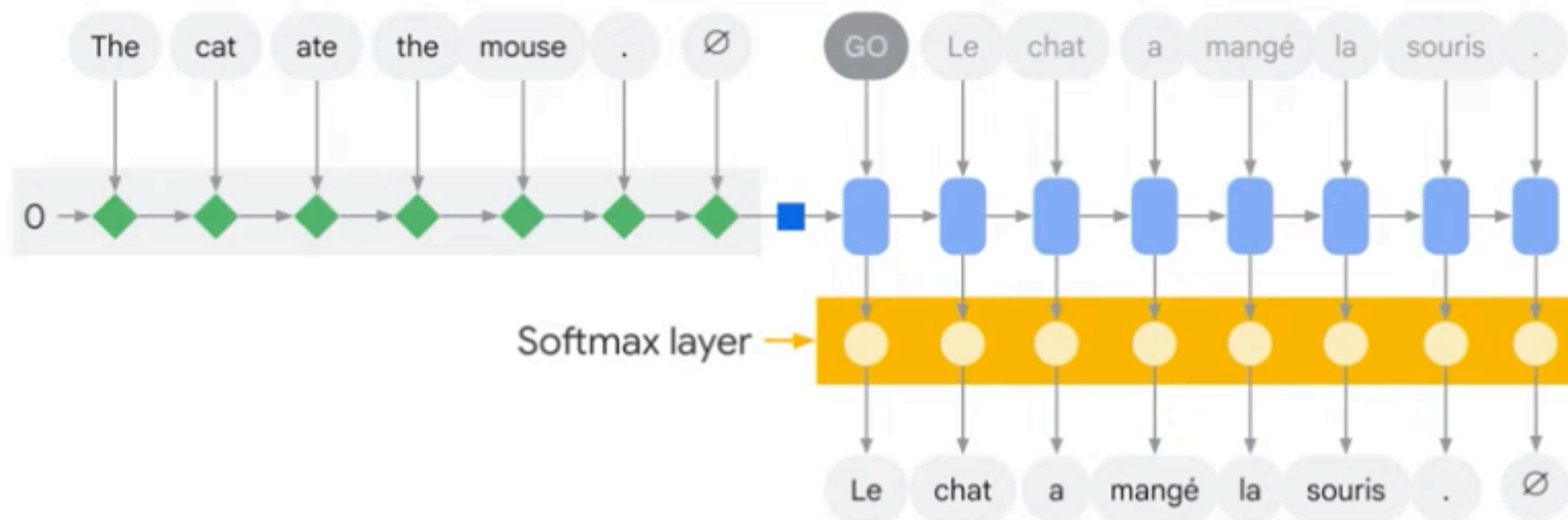
Encoder

Decoder





Encoder Decoder

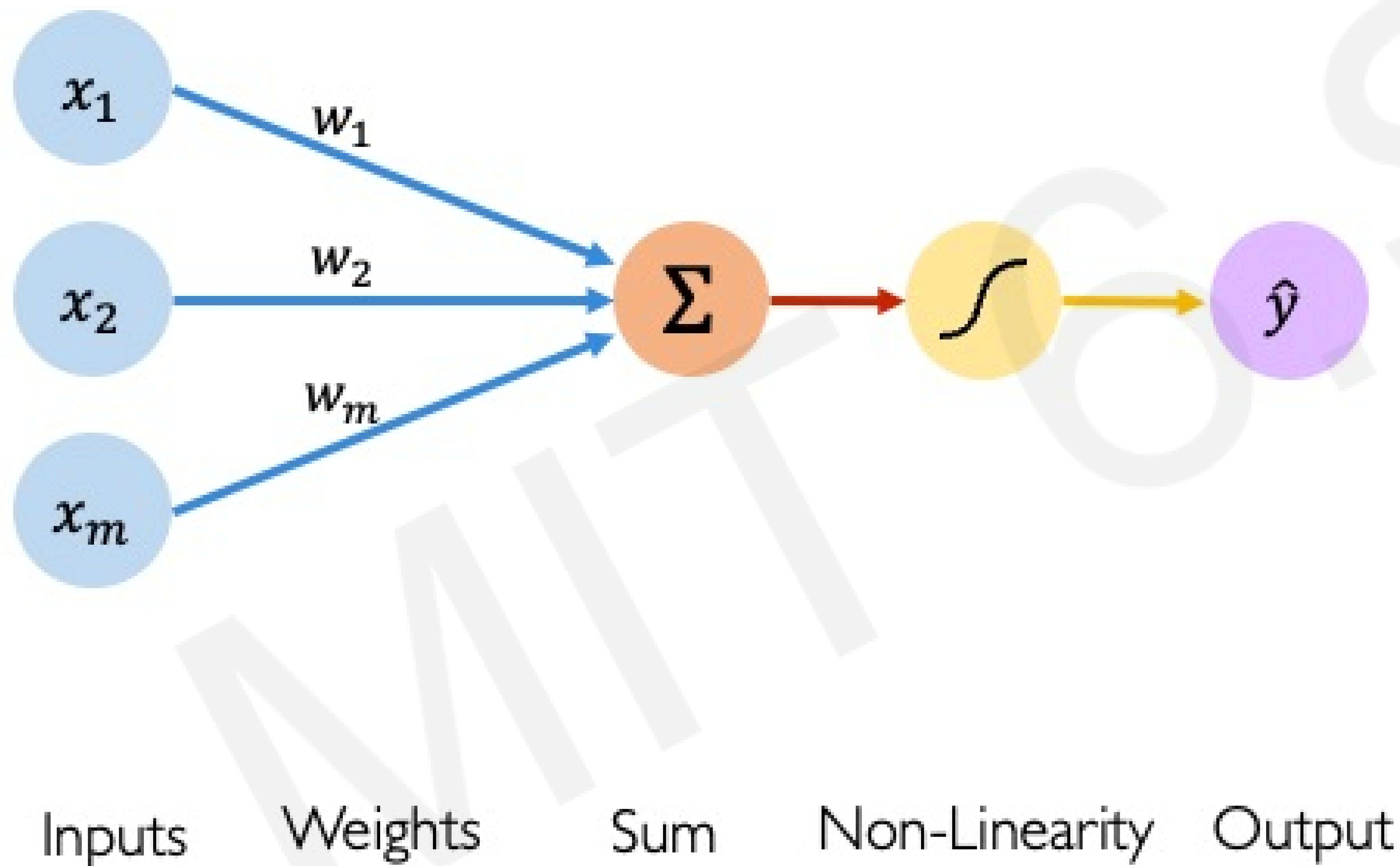


The Perceptron

The structural building block of deep learning

MIT 6.S191

The Perceptron: Forward Propagation



Output

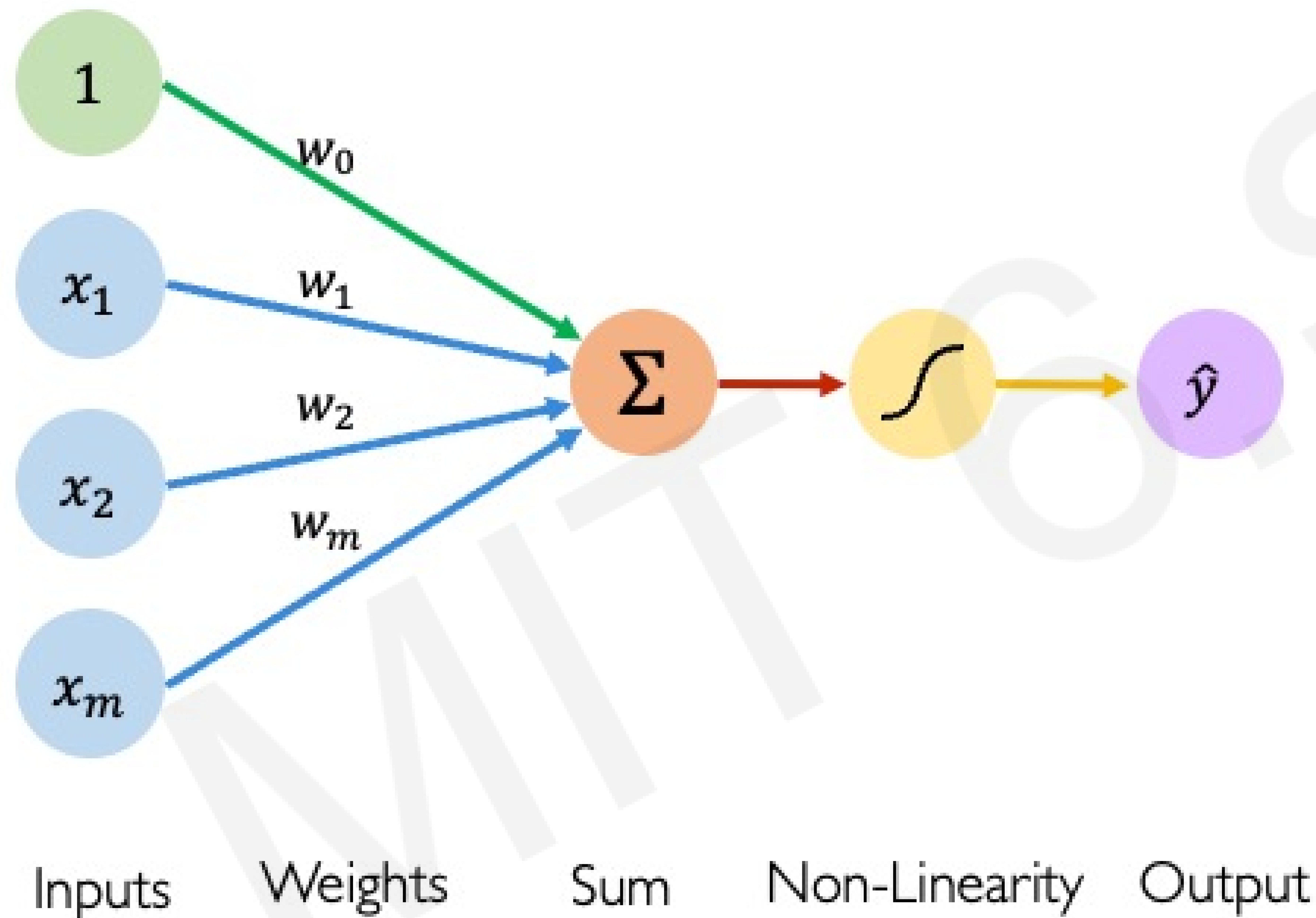
Linear combination of inputs

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i\, w_i\right)$$

Non-linear activation function

MIT 6.S191

The Perceptron: Forward Propagation



Output

Linear combination of inputs

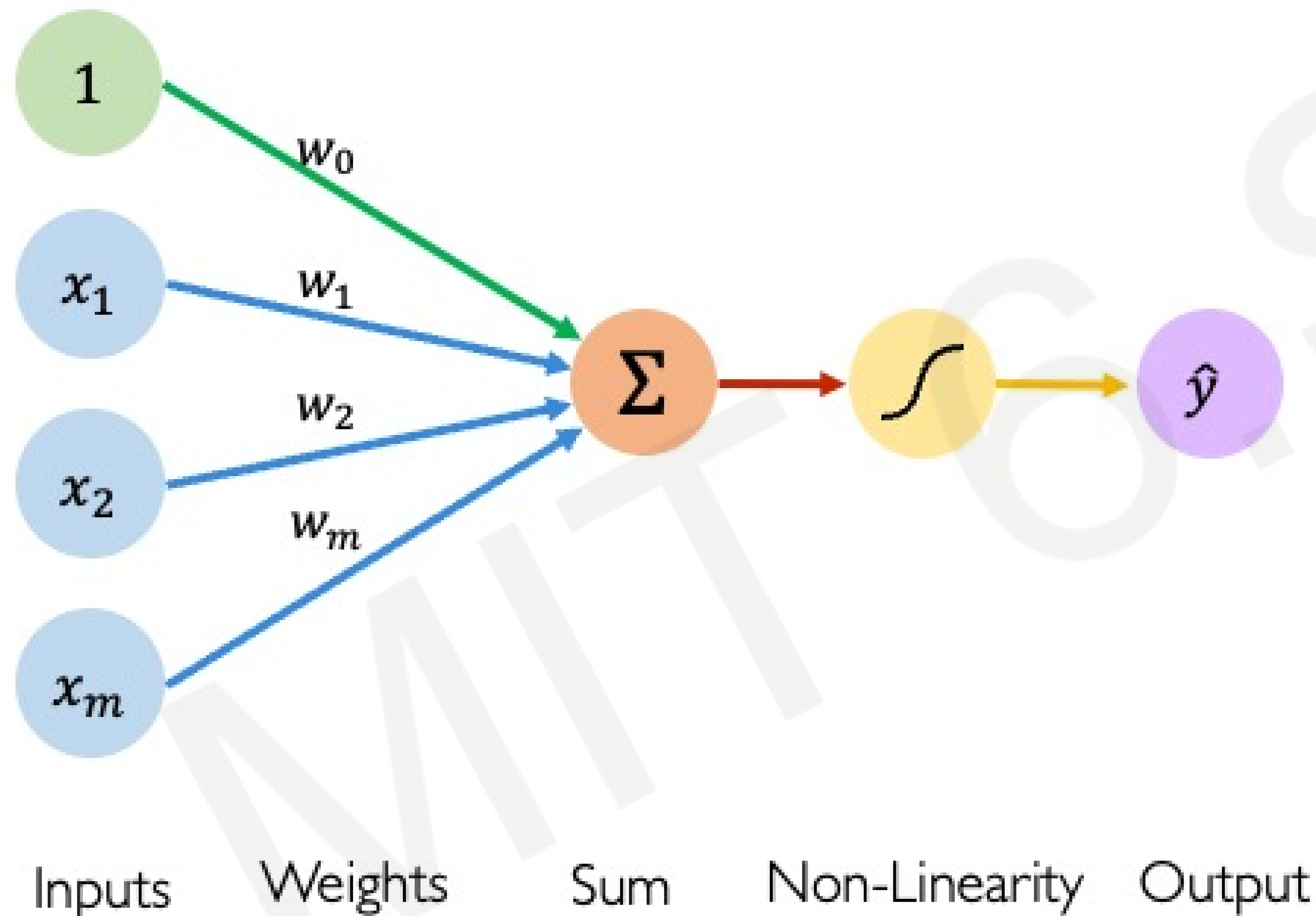
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

Non-linear activation function

Bias

MIT 6.S191

The Perceptron: Forward Propagation



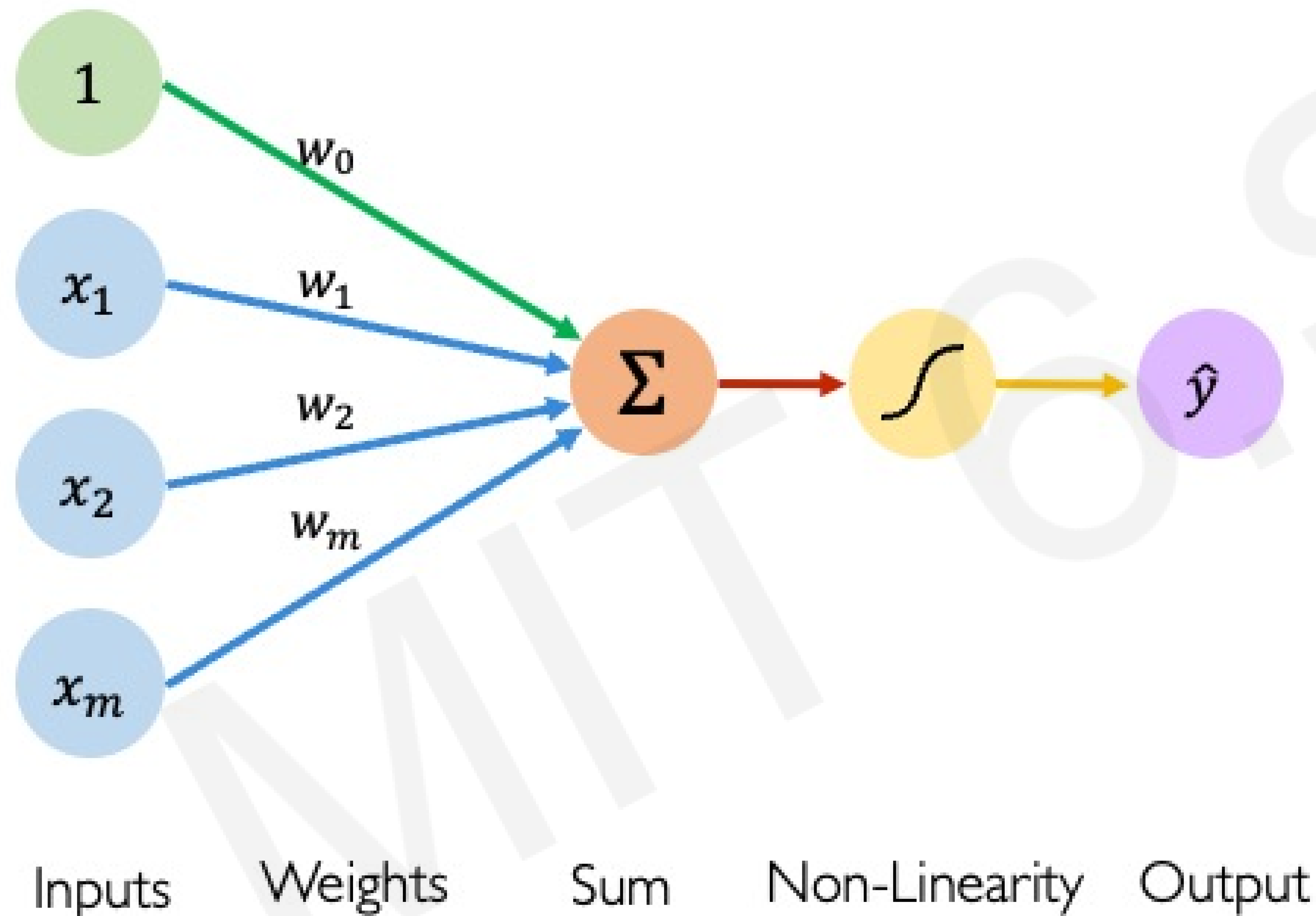
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i\right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

MIT 6.S191

The Perceptron: Forward Propagation

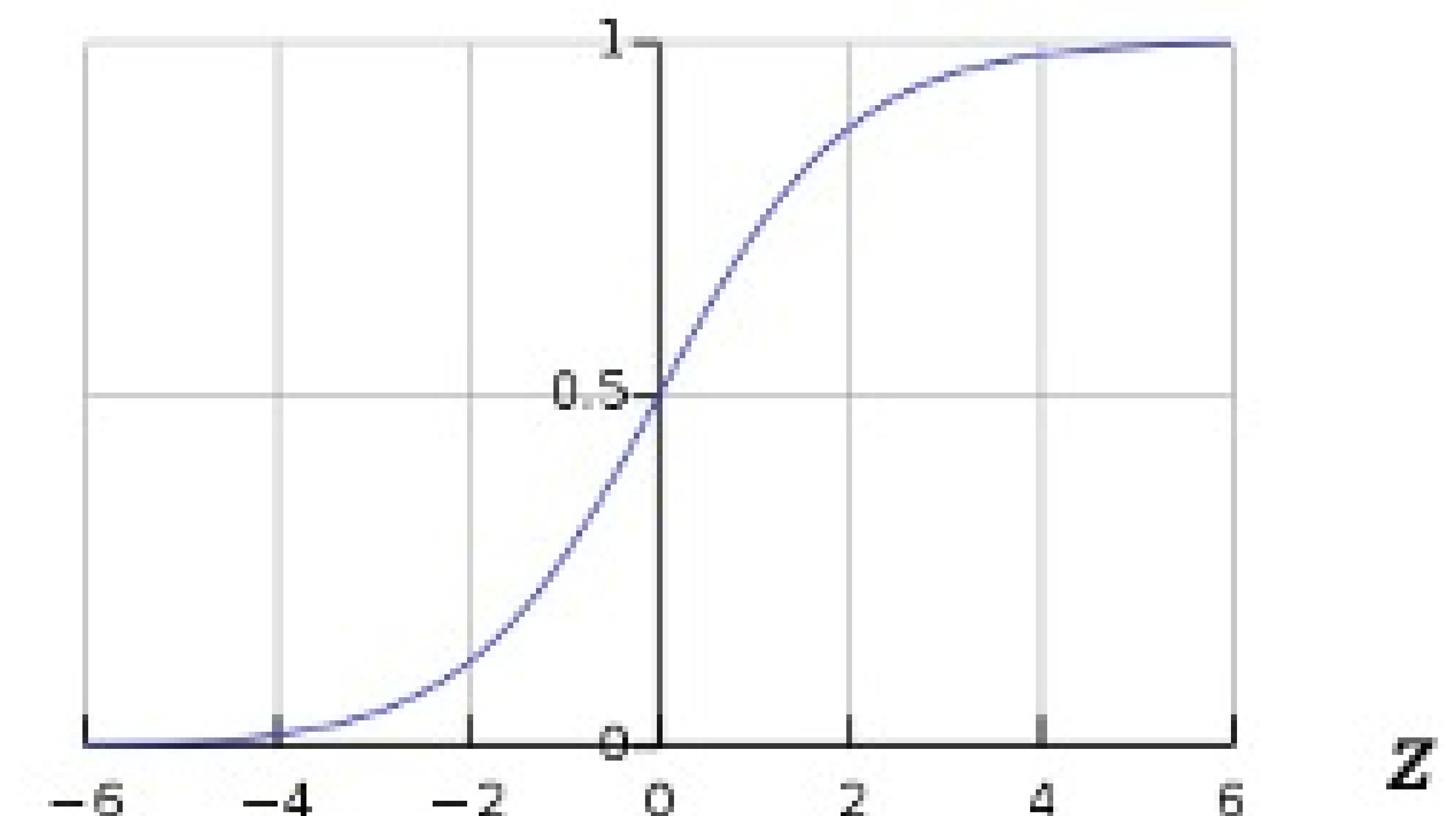


Activation Functions

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

- Example: sigmoid function

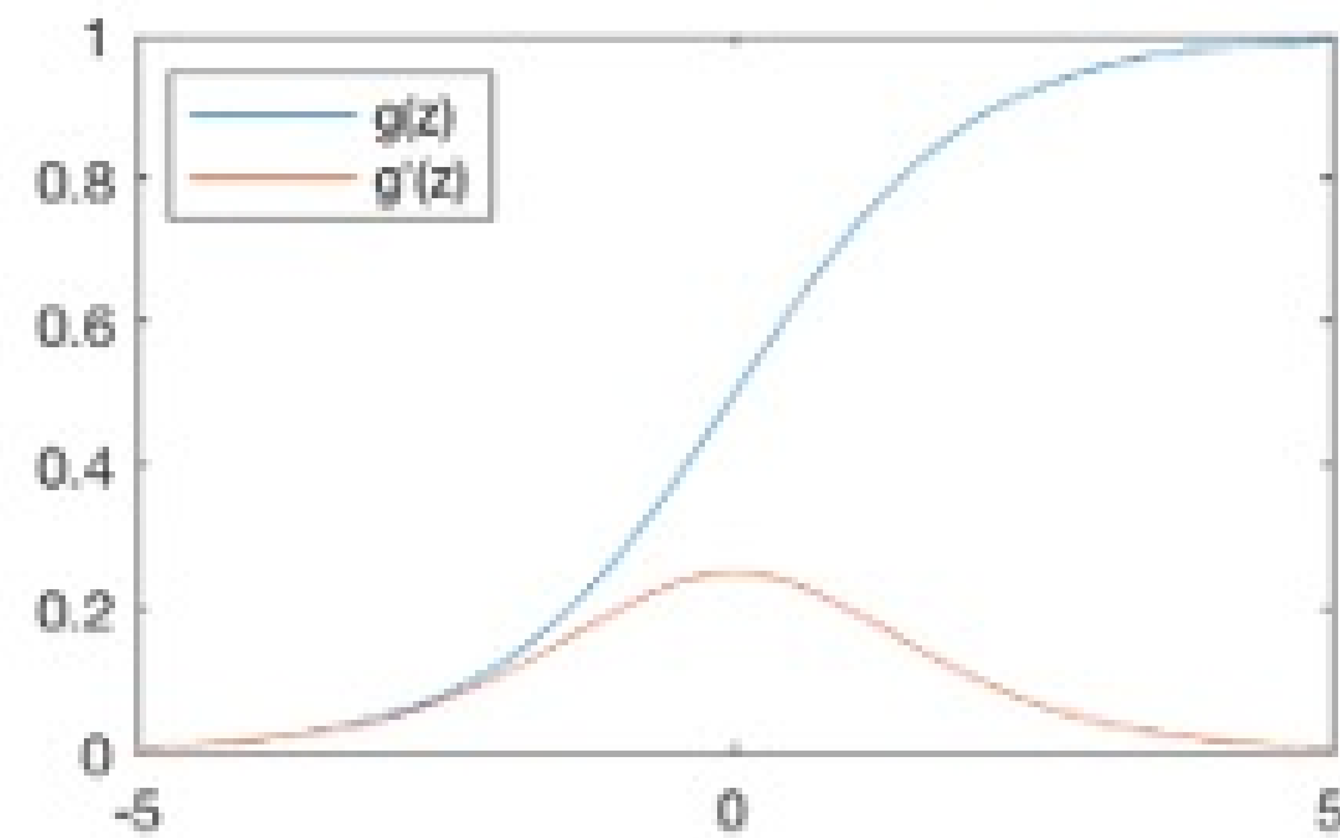
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



MIT 6.S191

Common Activation Functions

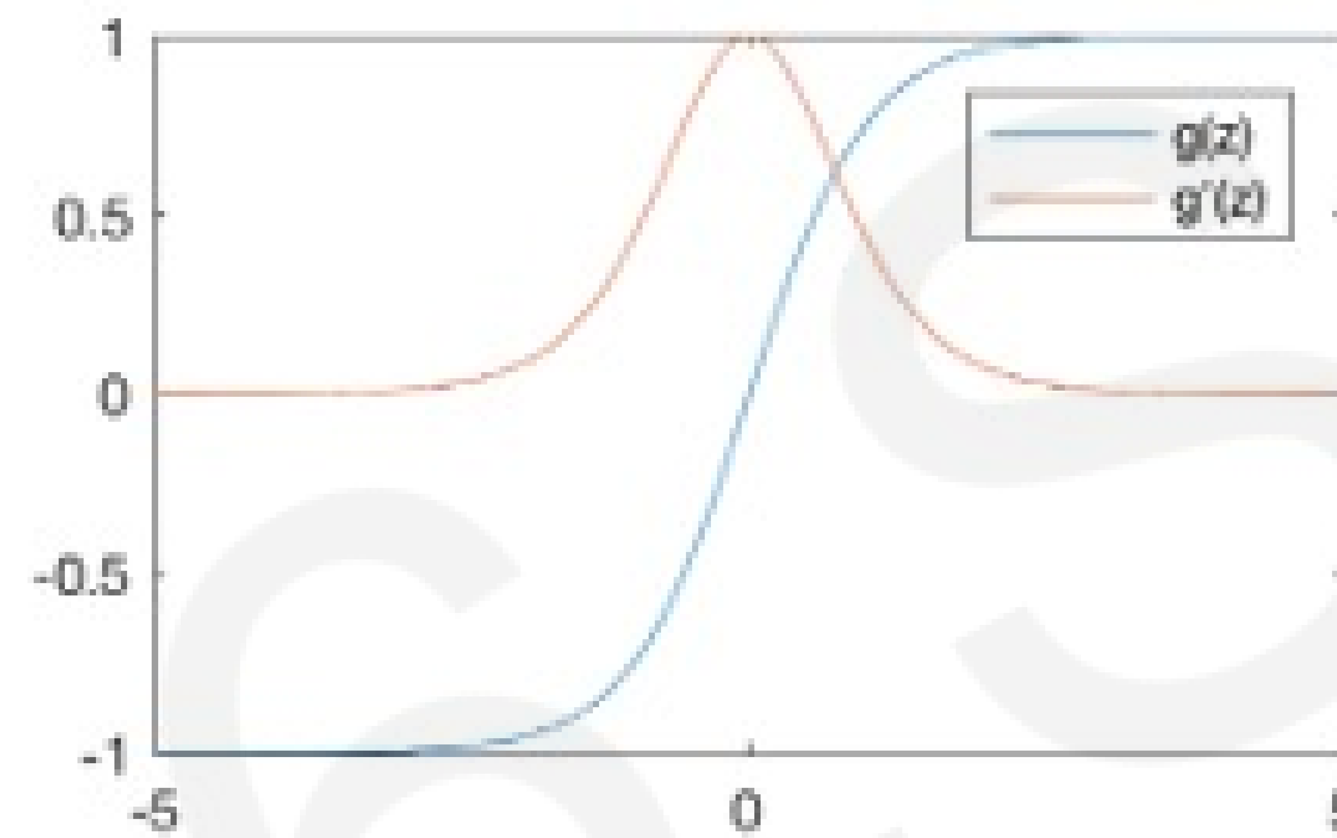
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

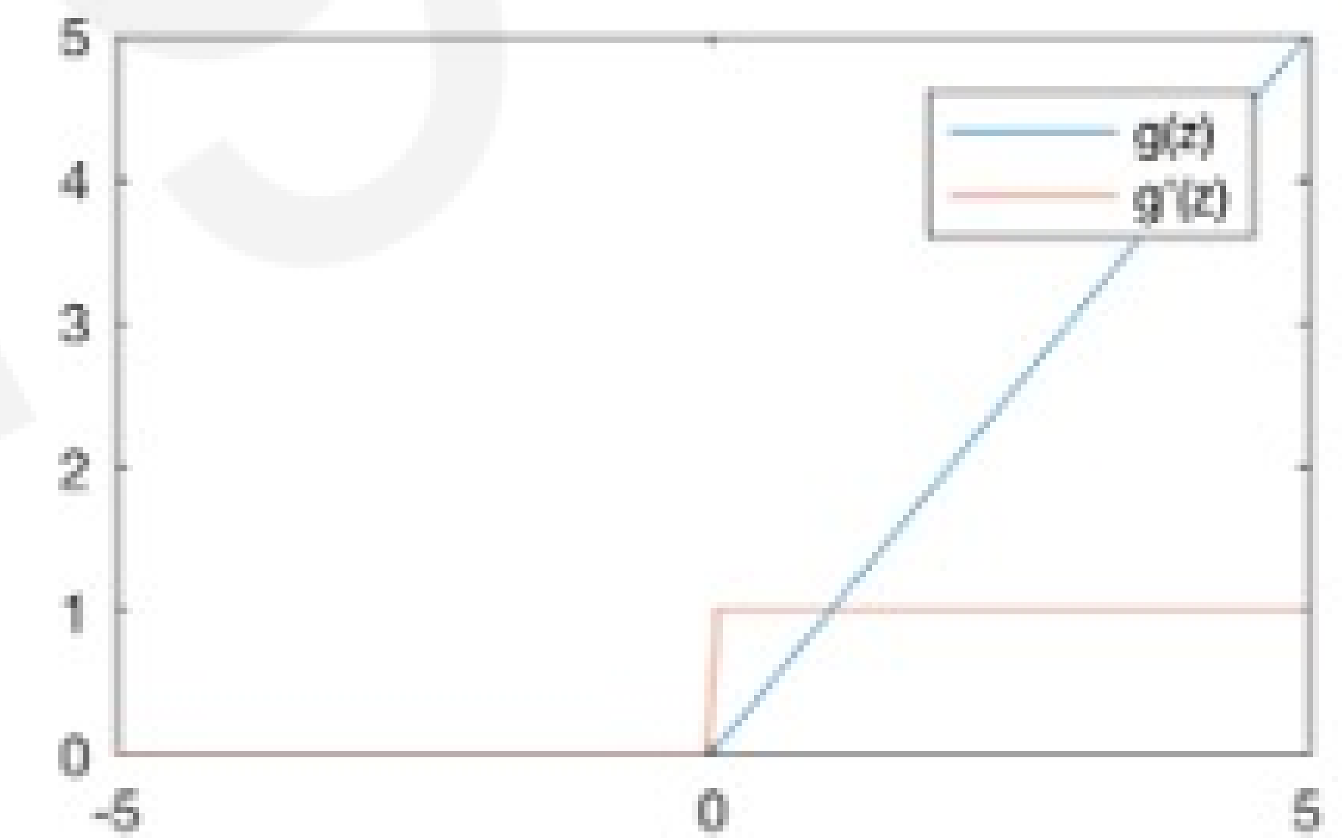
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



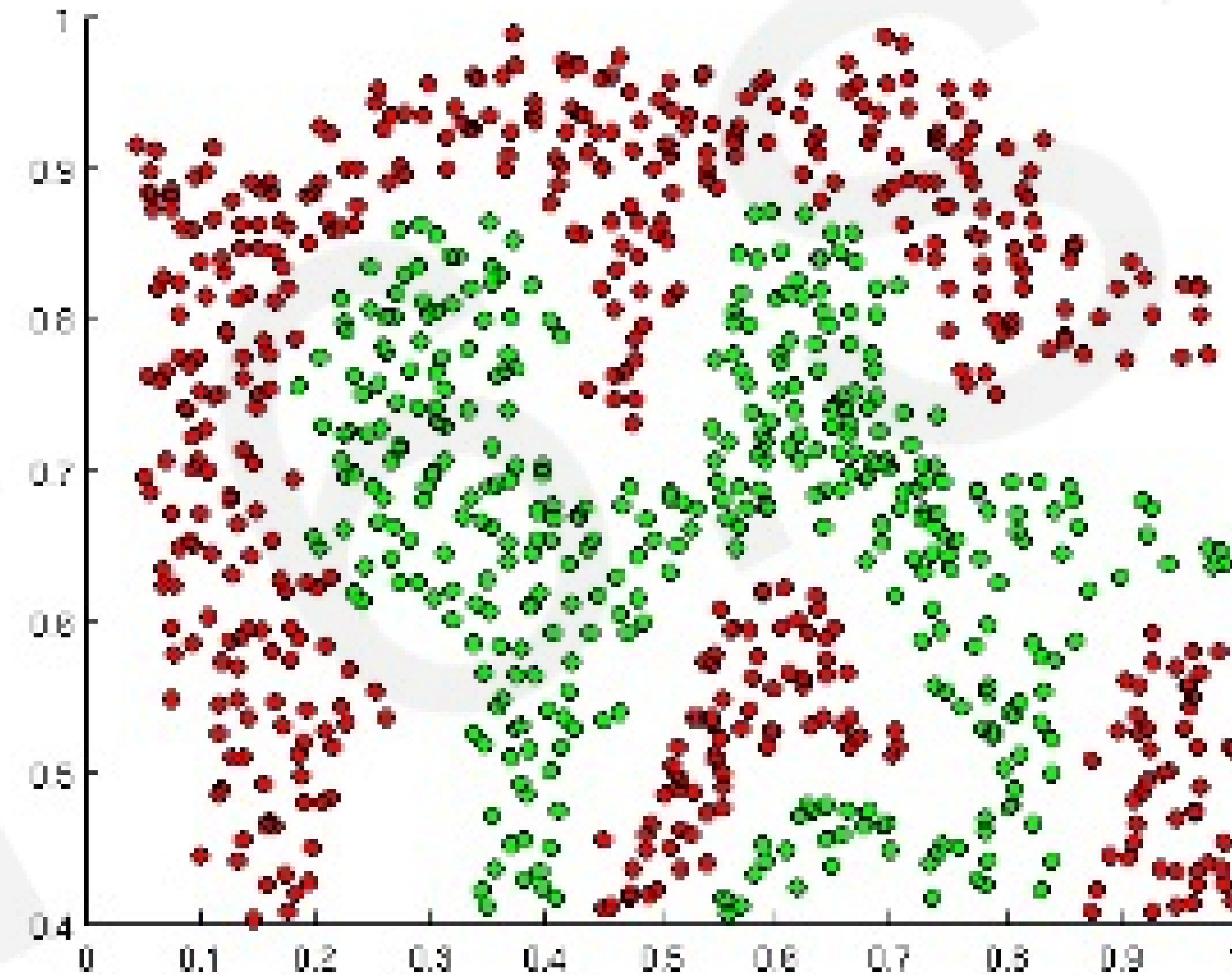
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

NOTE: All activation functions are non-linear

Importance of Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*

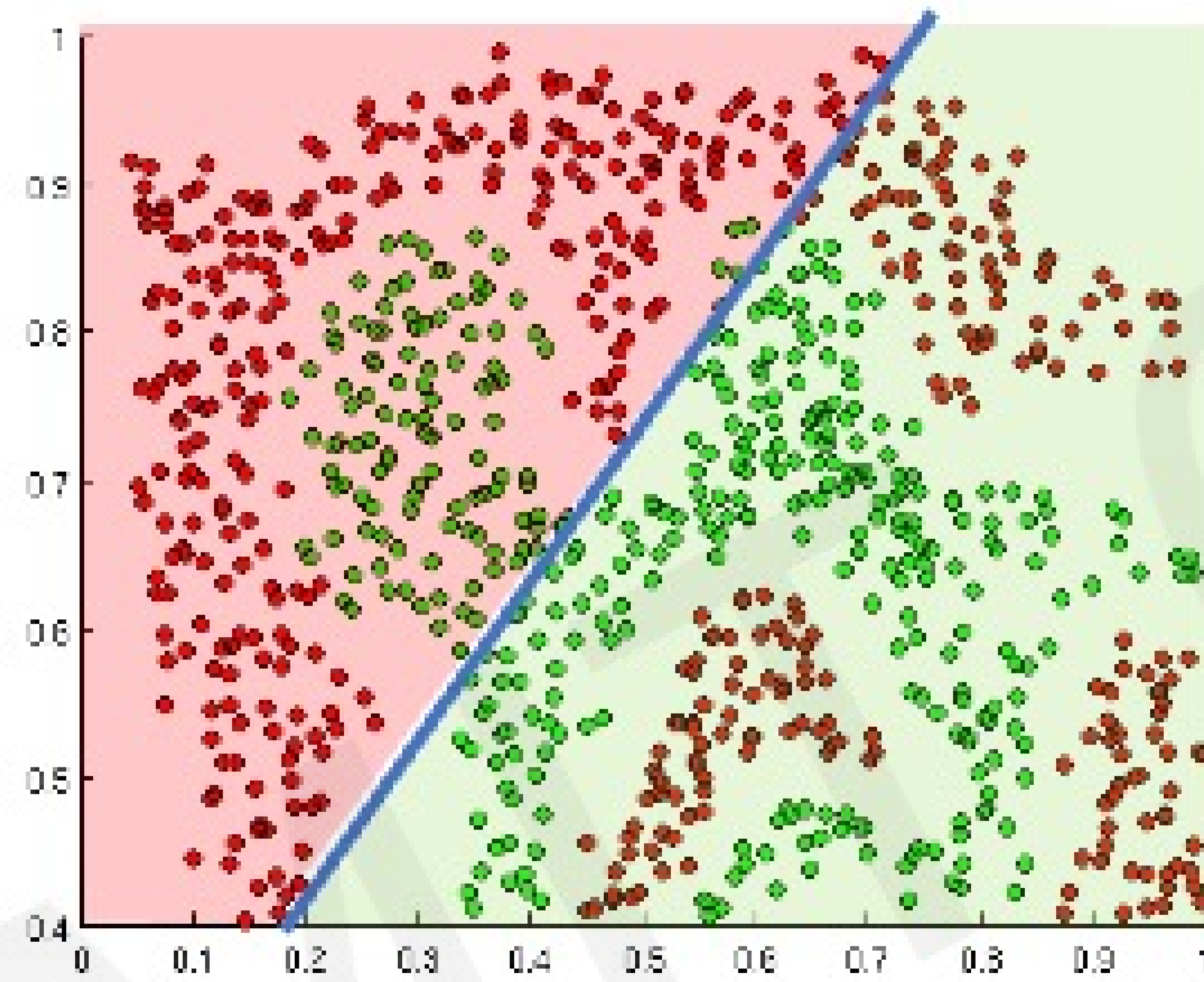


What if we wanted to build a neural network to distinguish green vs red points?

MIT 6.S191

Importance of Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*

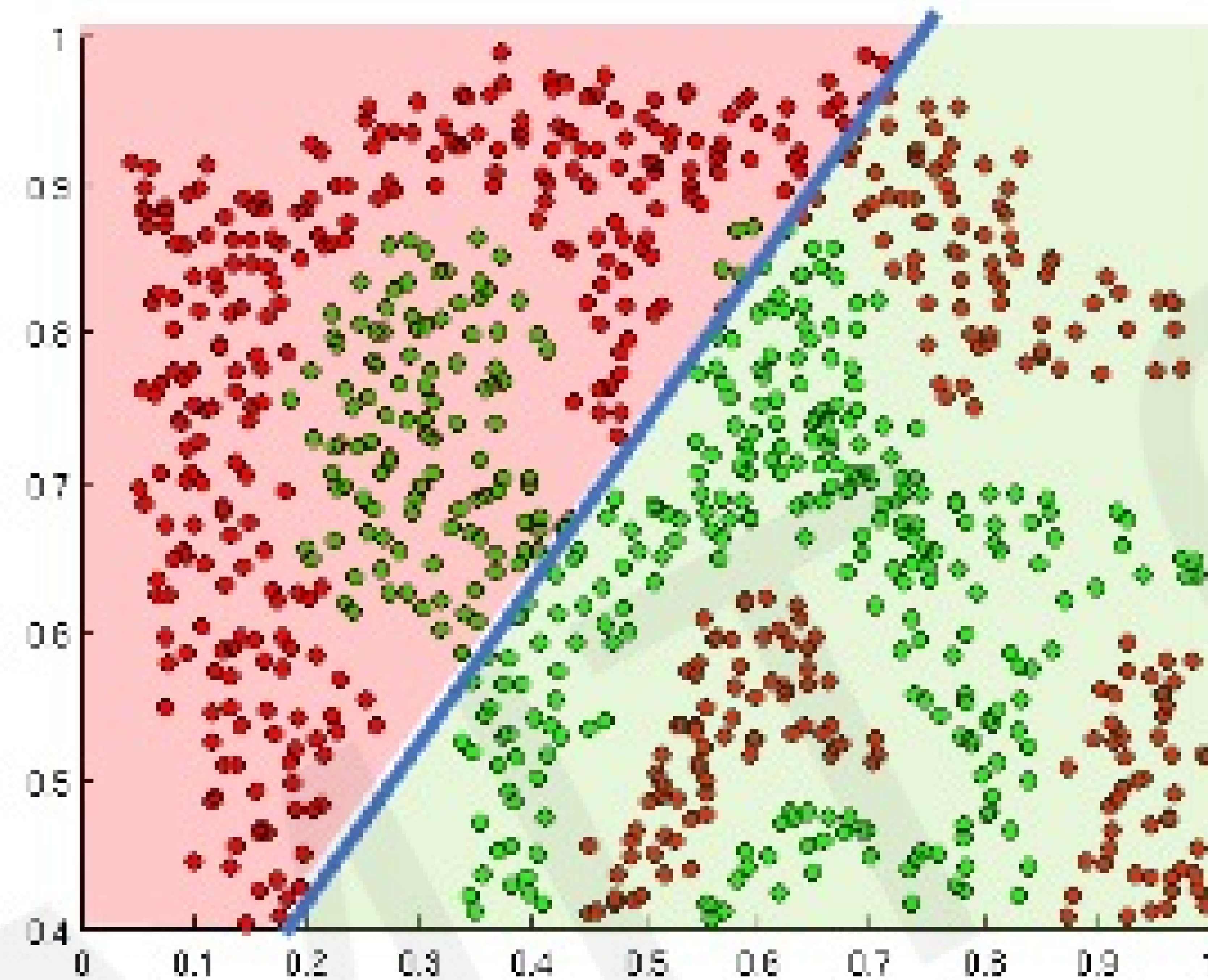


Linear activation functions produce linear decisions no matter the network size

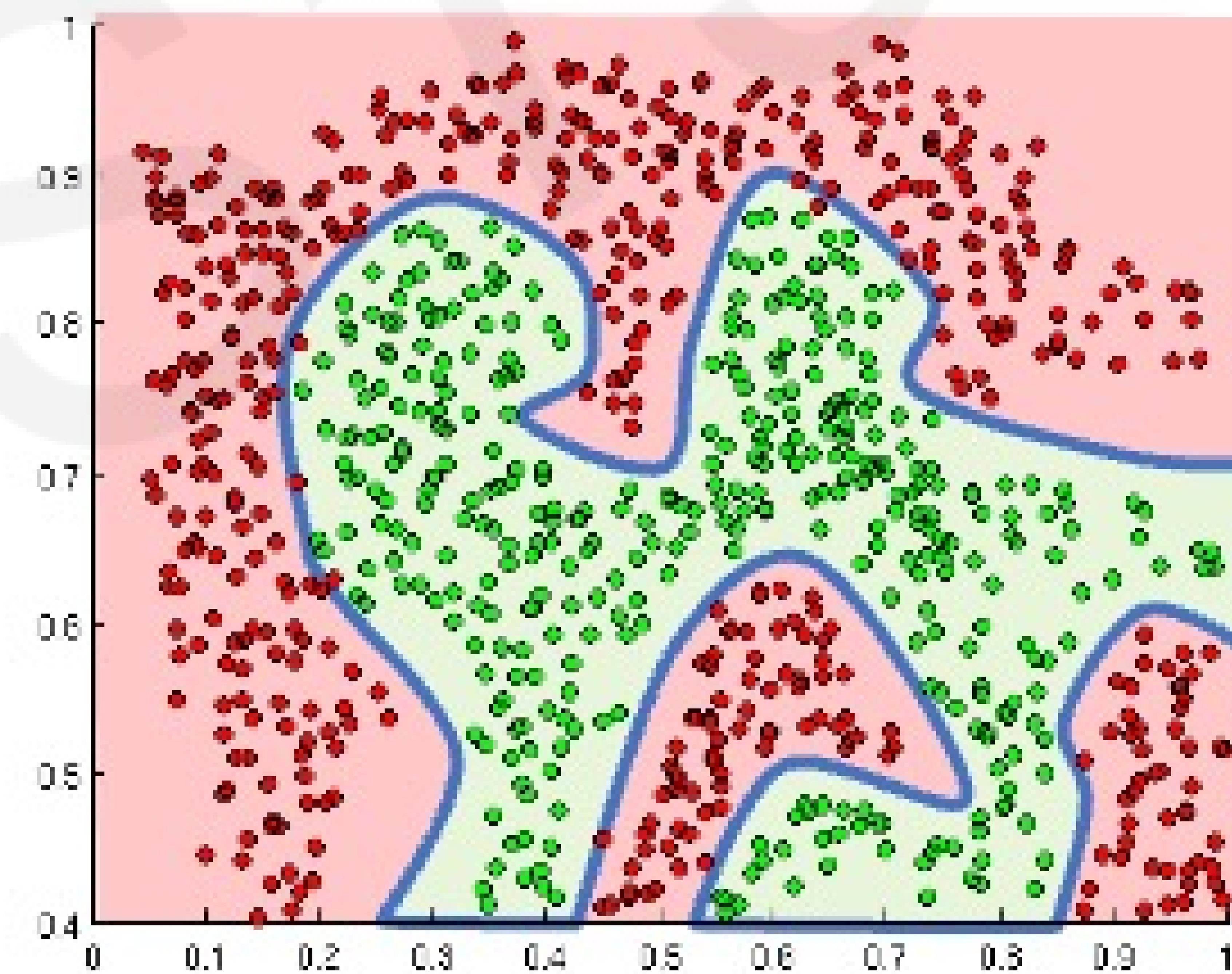
MIT 6.S191

Importance of Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*



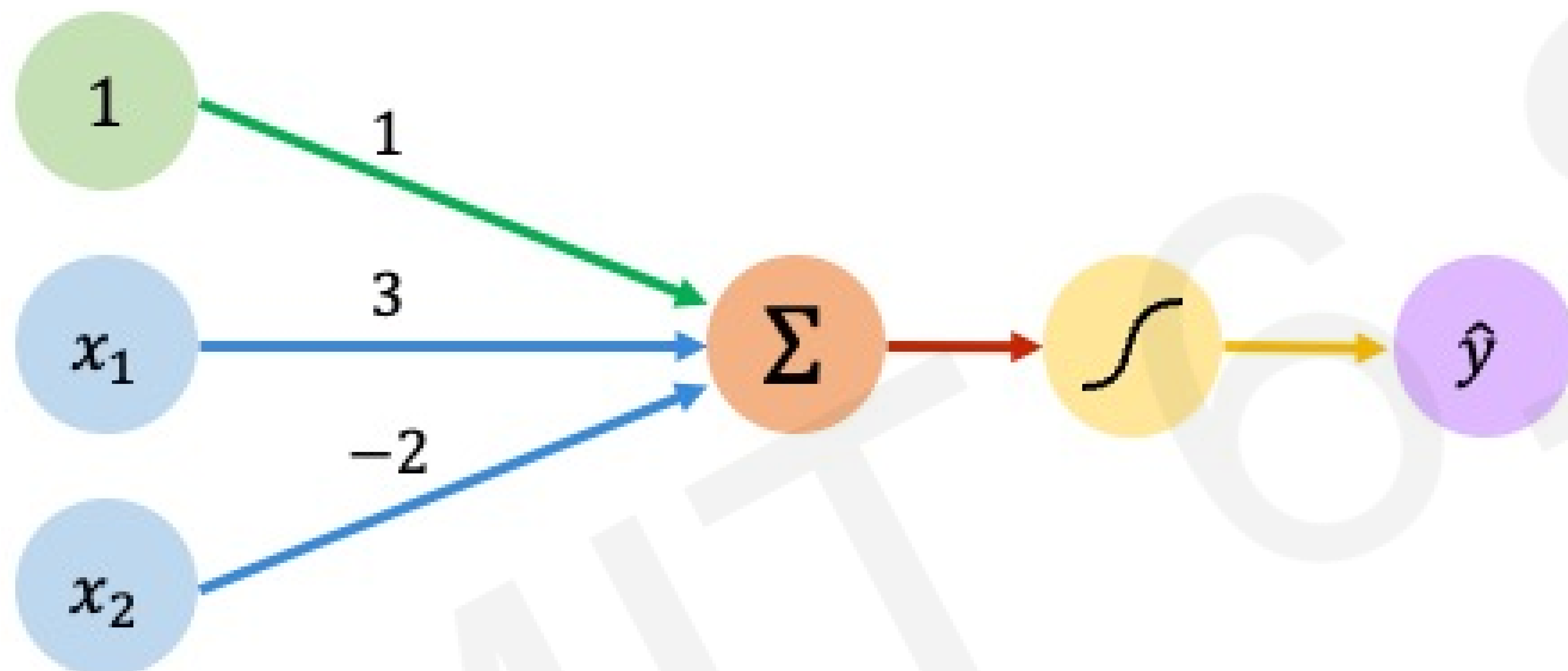
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

MIT 6.S191

The Perceptron: Example



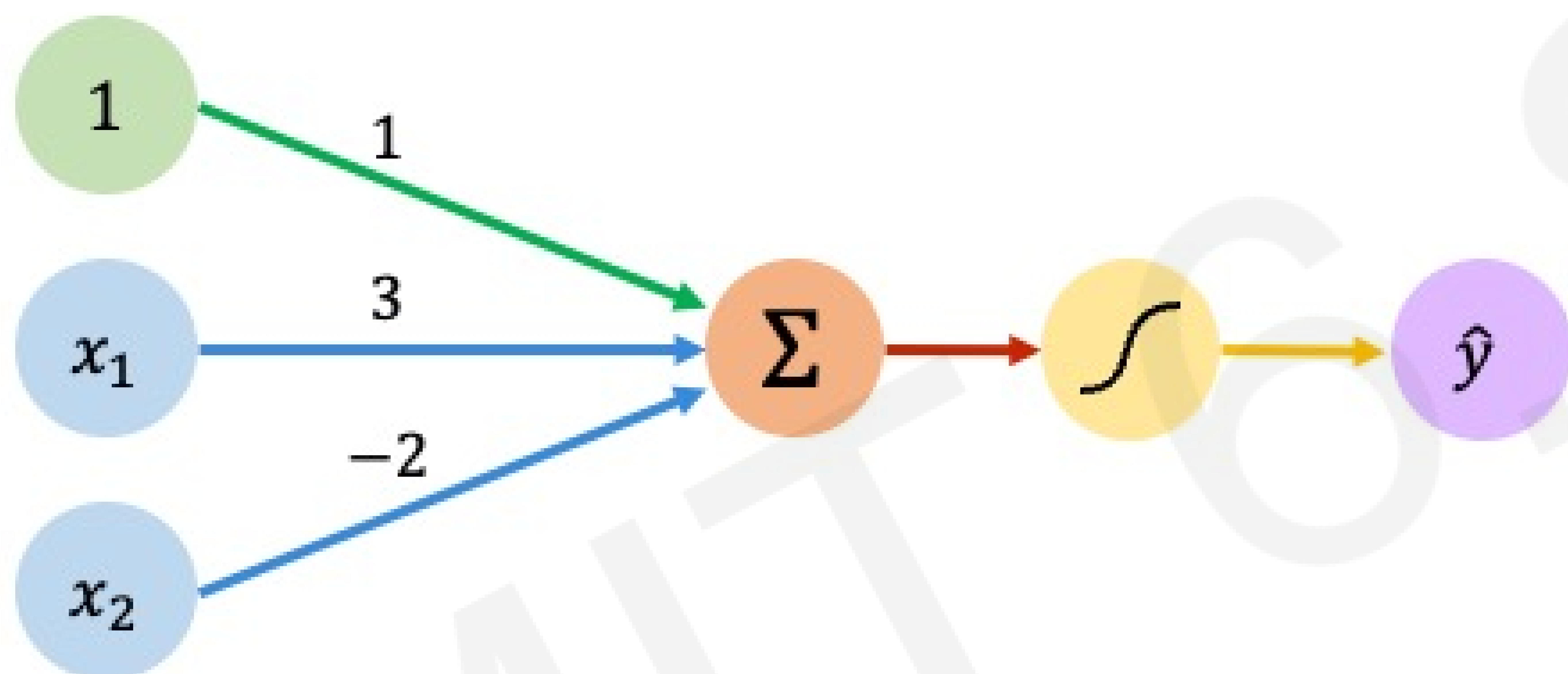
We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

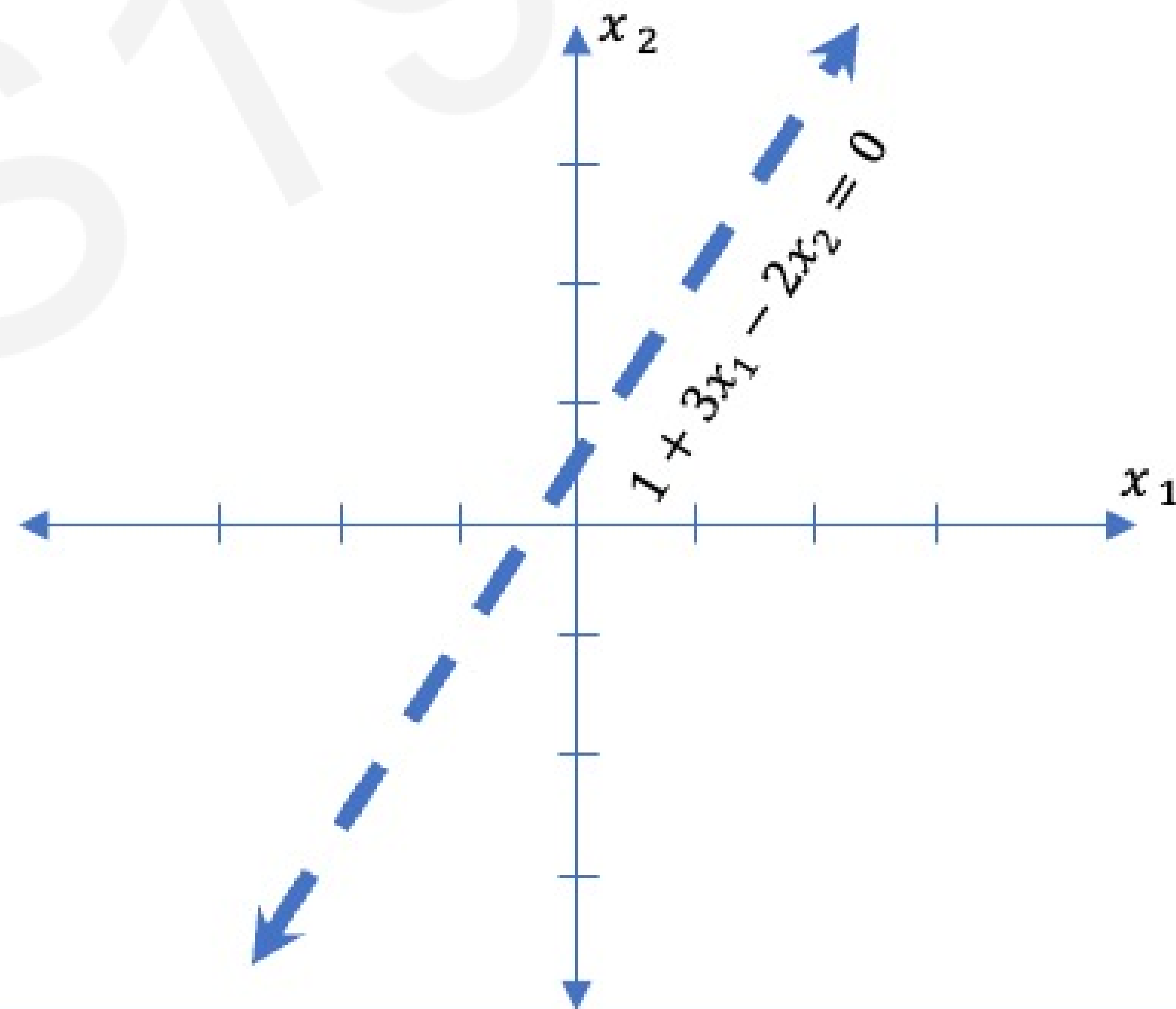
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$
$$\hat{y} = g\left(\underbrace{1 + 3x_1 - 2x_2}_{}\right)$$

This is just a line in 2D!

MIT 6.S191

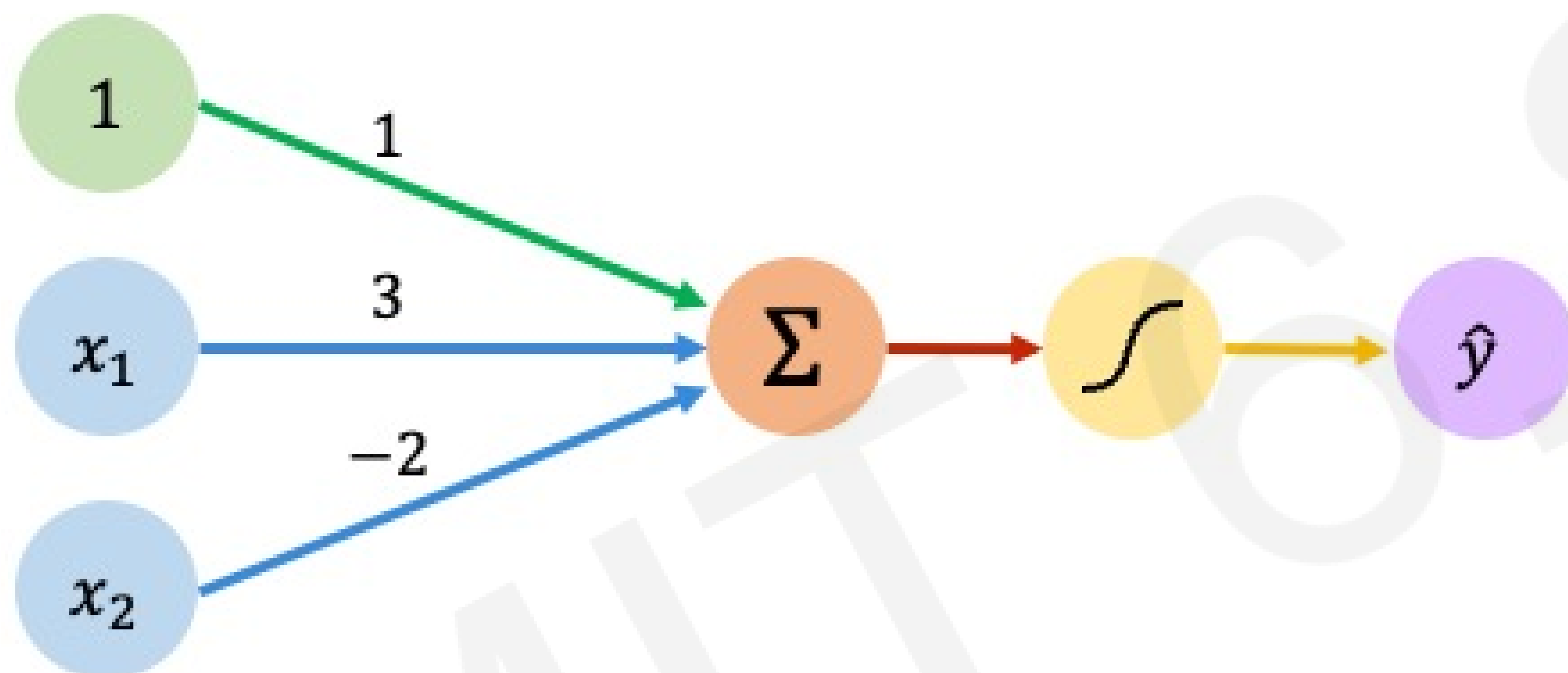
The Perceptron: Example

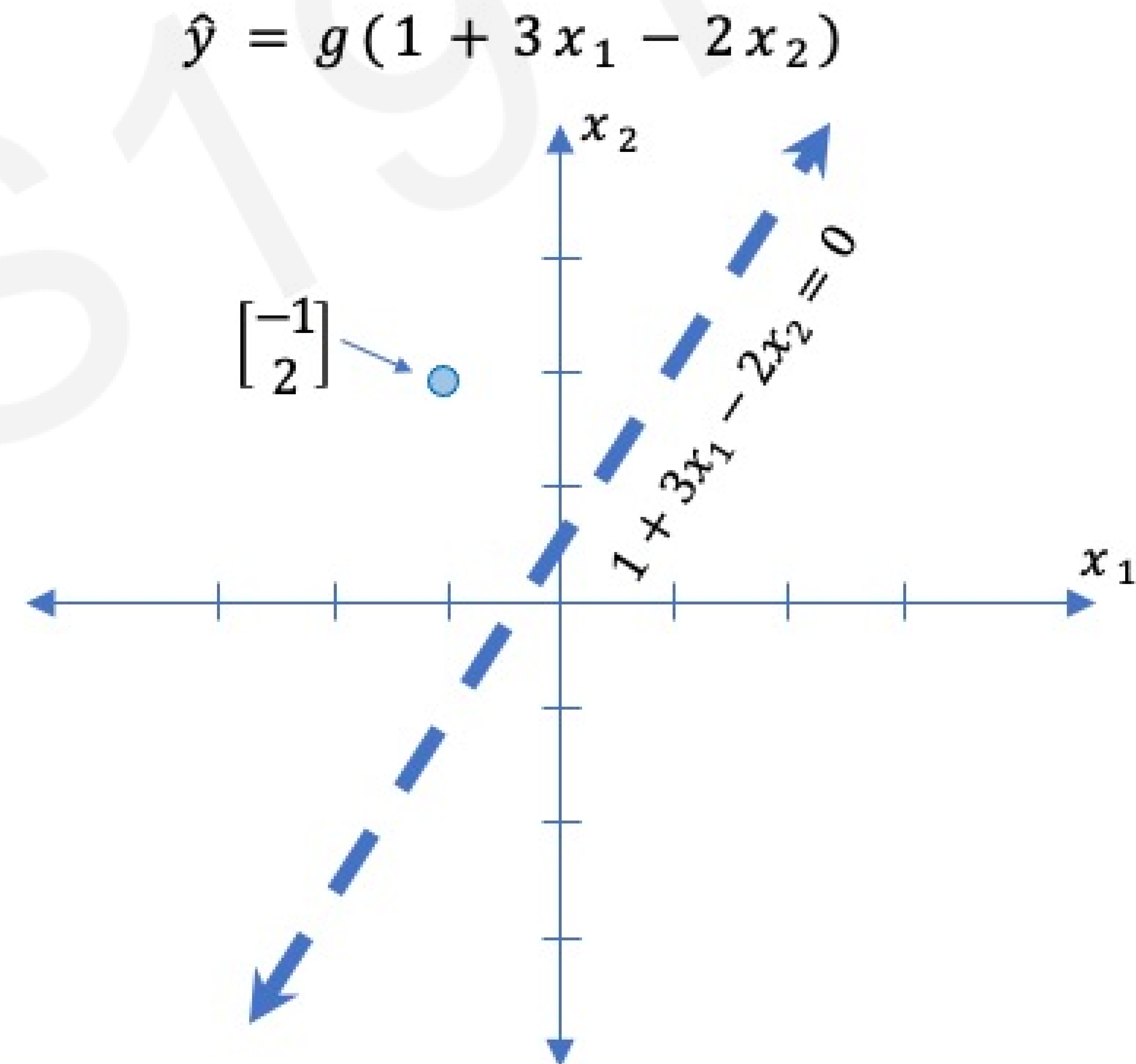




MIT 6.S191

The Perceptron: Example

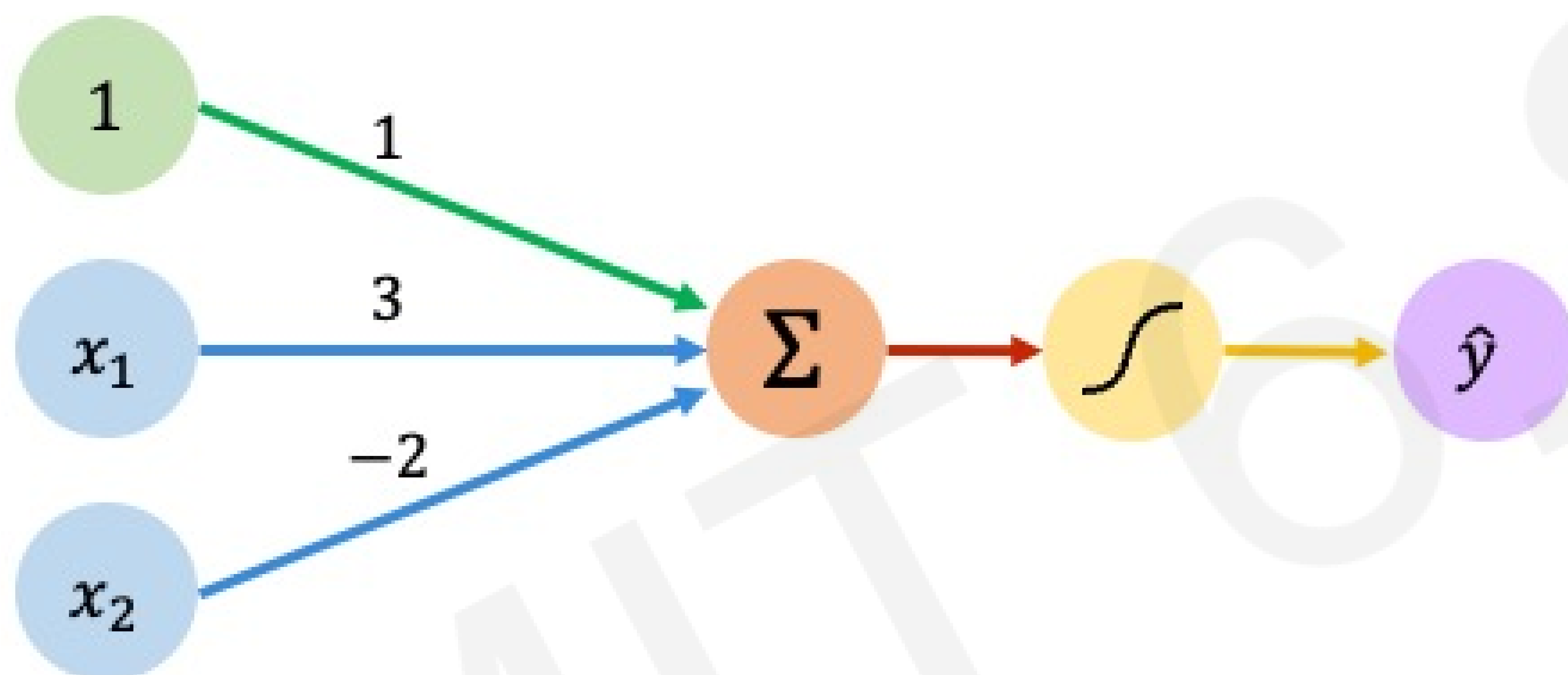




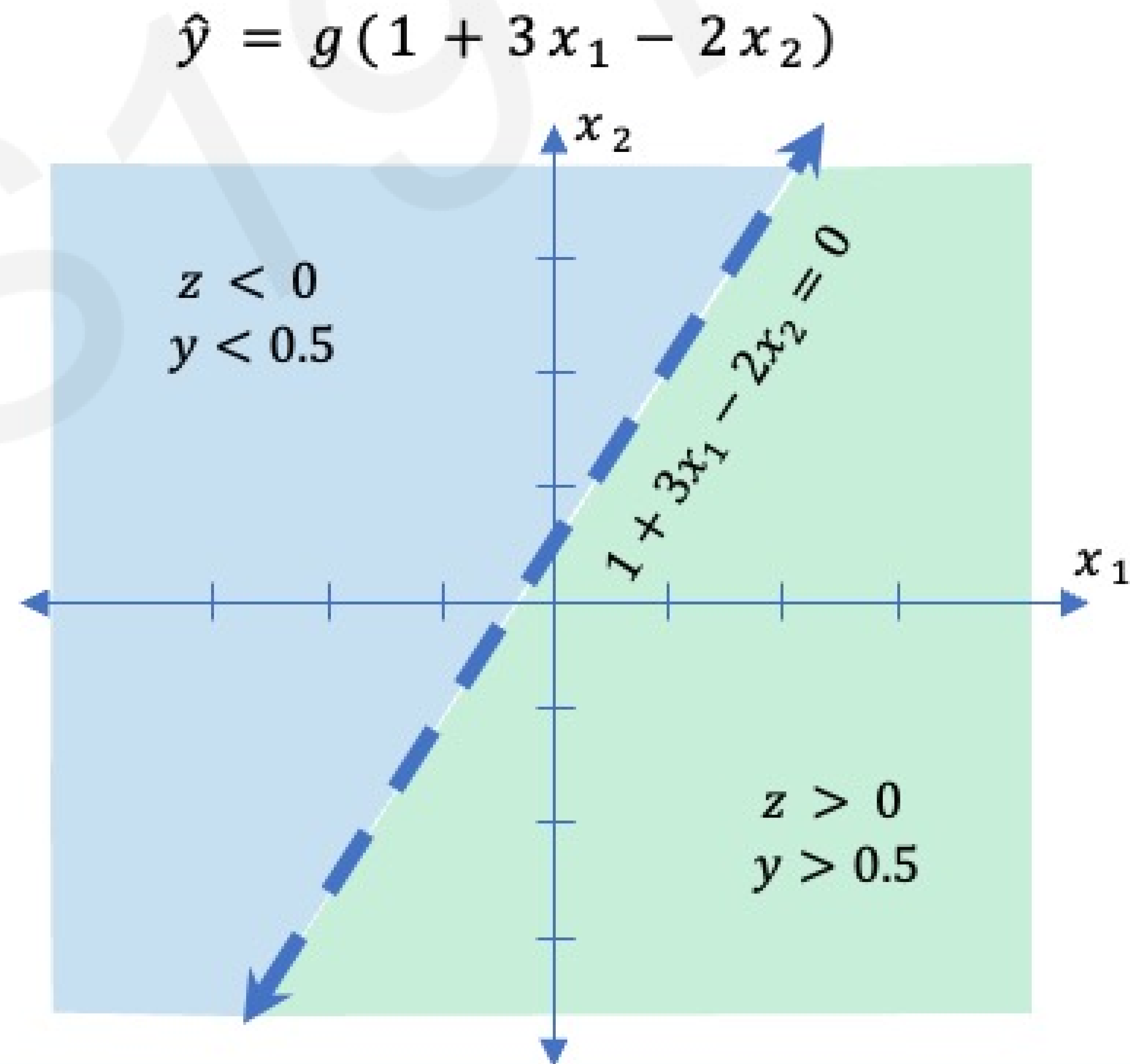
Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$\hat{y} = g\big(1 + (3 * -1) - (2 * 2)\big)$$
$$= g(-6) \approx 0.002$$

The Perceptron: Example



$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



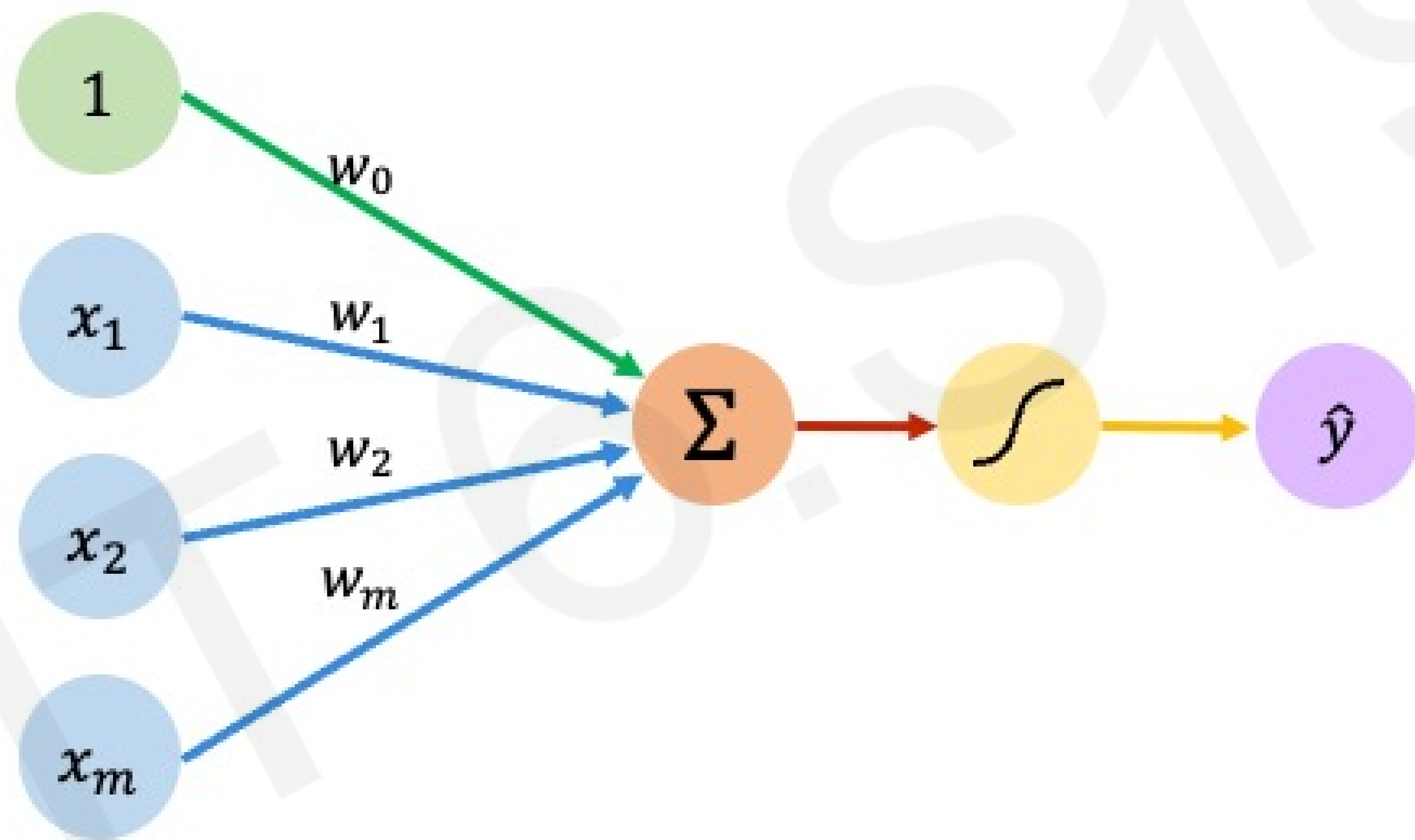
MIT 6.S191

Building Neural Networks with Perceptrons

MIT 6.S191

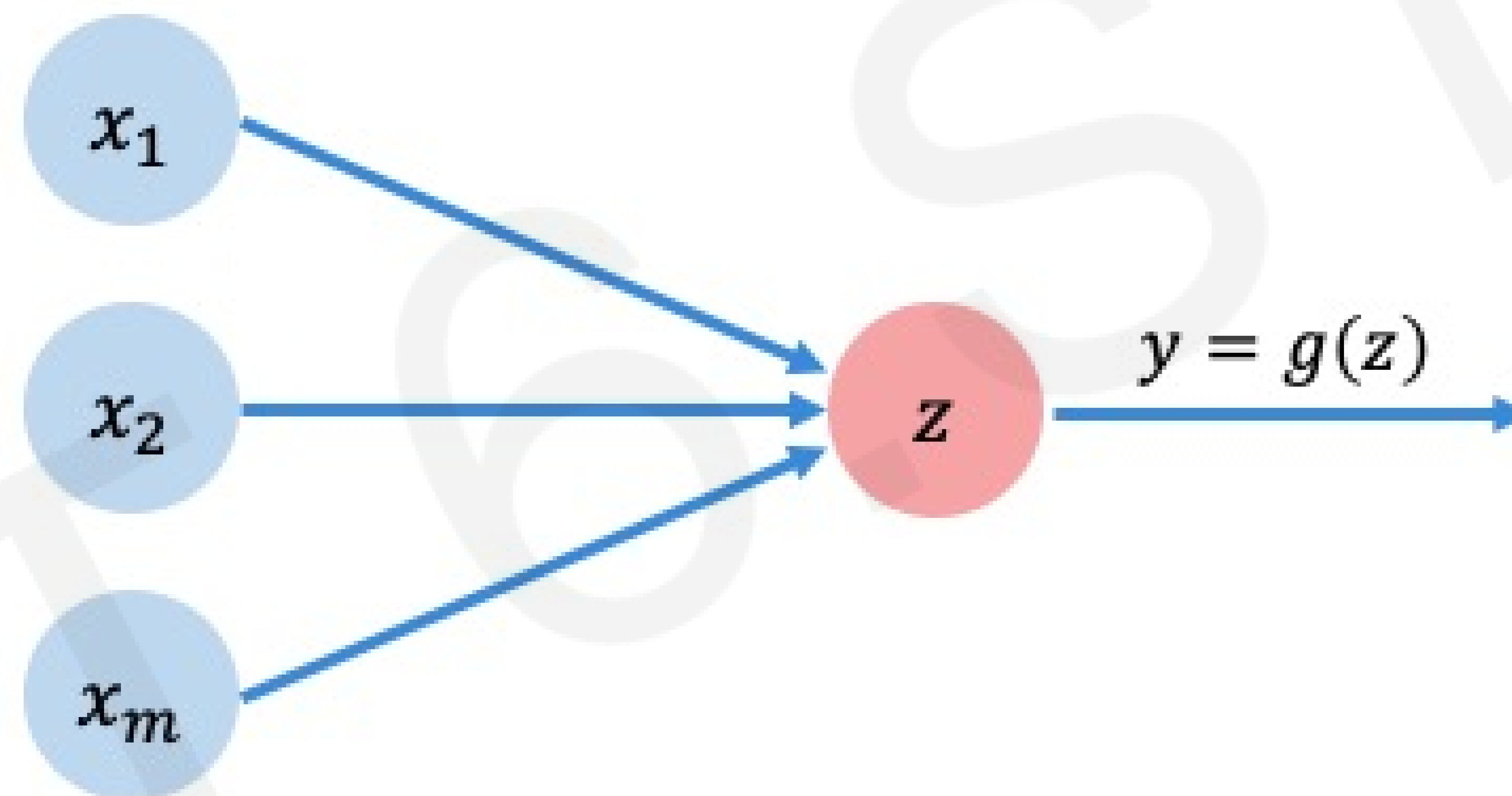
The Perceptron: Simplified

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$



MIT 6.S191

The Perceptron: Simplified

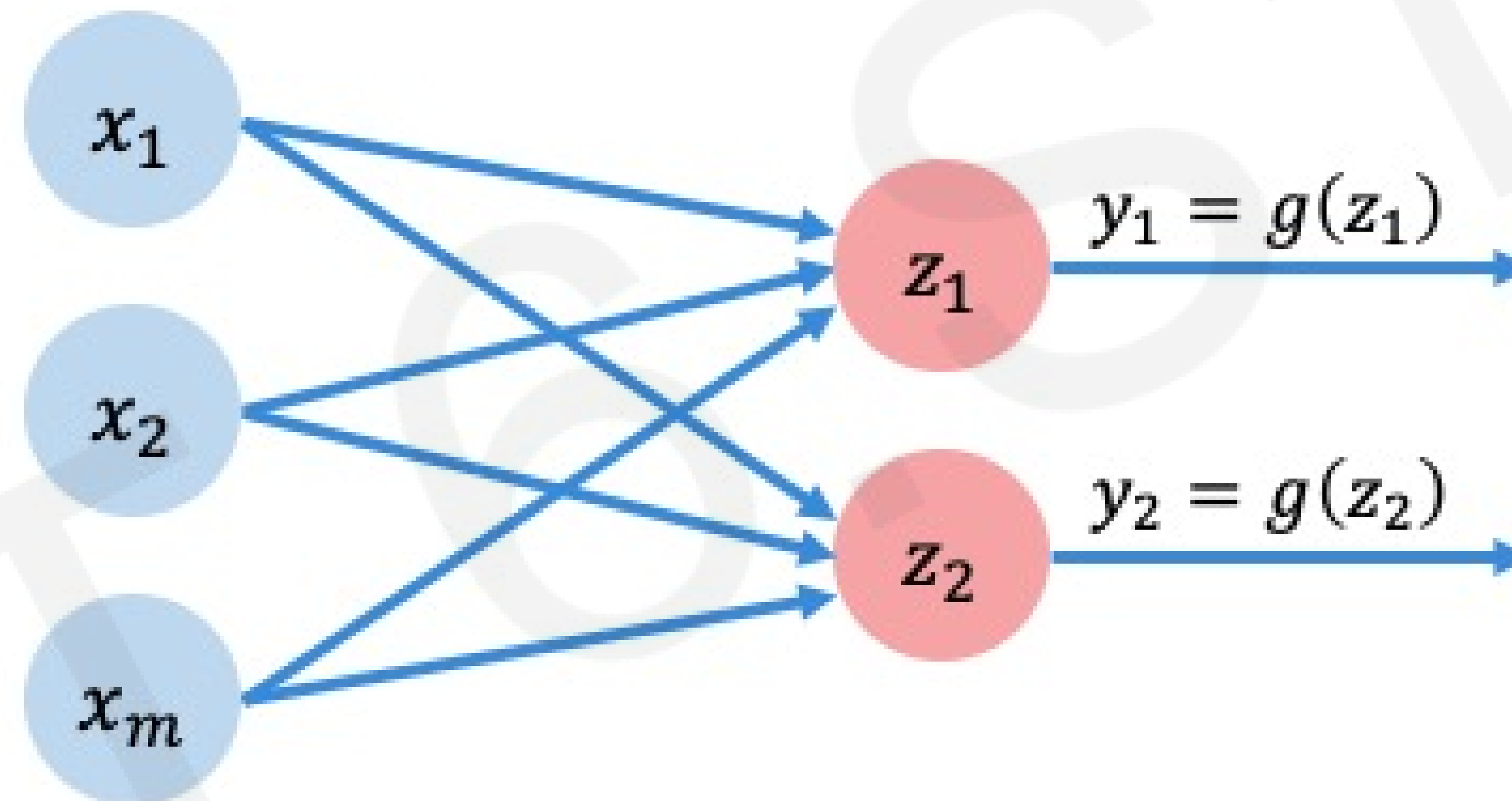


$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

MIT 6.S191

Multi Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

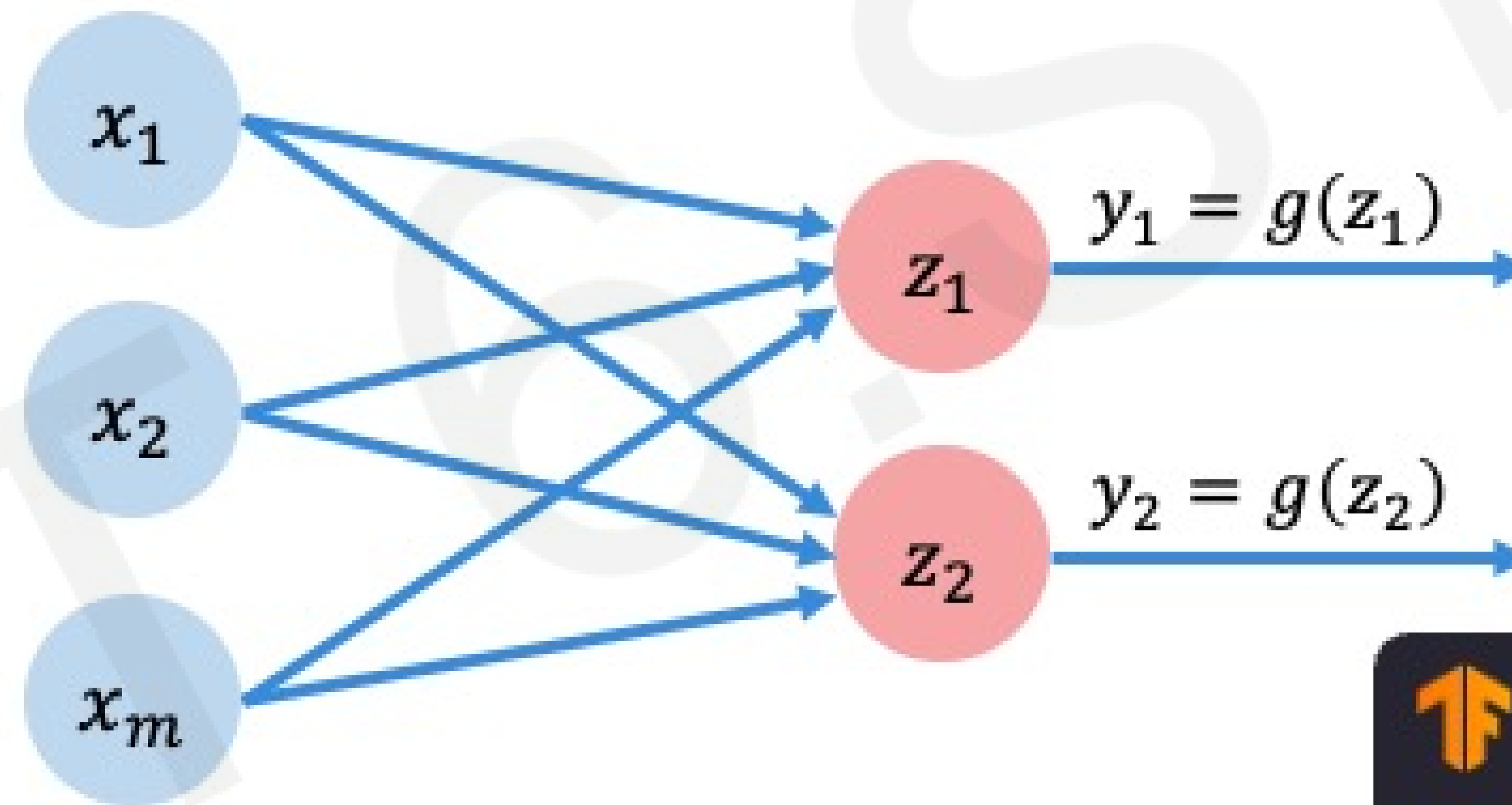
MIT 6.S191

Dense layer from scratch

```
class MyDenseLayer(tf.keras.layers.Layer):
  def __init__(self, input_dim, output_dim):
    super(MyDenseLayer, self).__init__()

    # Initialize weights and bias
    self.W = self.add_weight([input_dim, output_dim])
    self.b = self.add_weight([1, output_dim])

  def call(self, inputs):
    # Forward propagate the inputs
    z = tf.matmul(inputs, self.W) + self.b

    # Feed through a non-linear activation
    output = tf.math.sigmoid(z)

    return output
```

Multi Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers

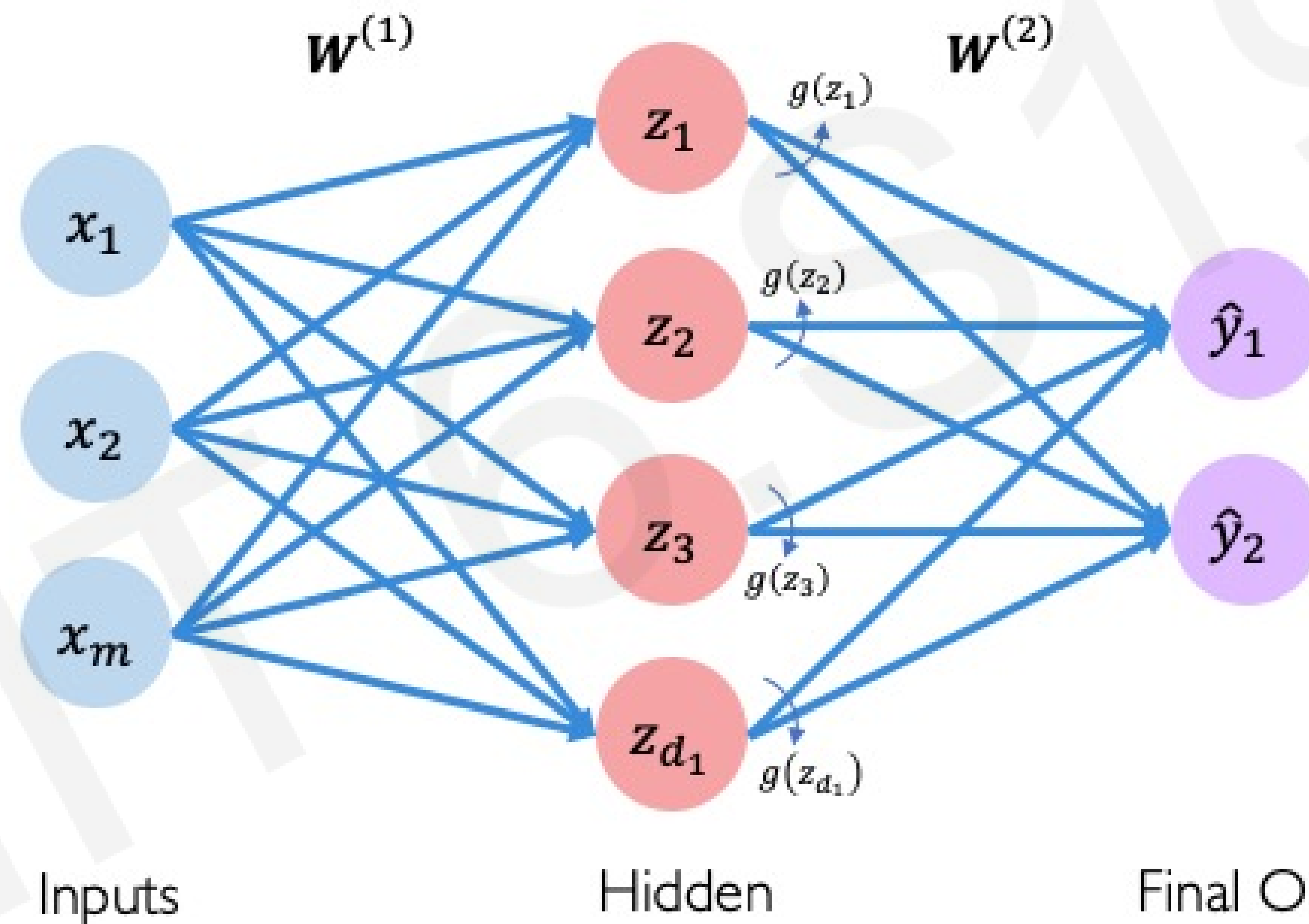



```python
import tensorflow as tf

layer = tf.keras.layers.Dense(
    units=2)
```

$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

MIT 6.S191

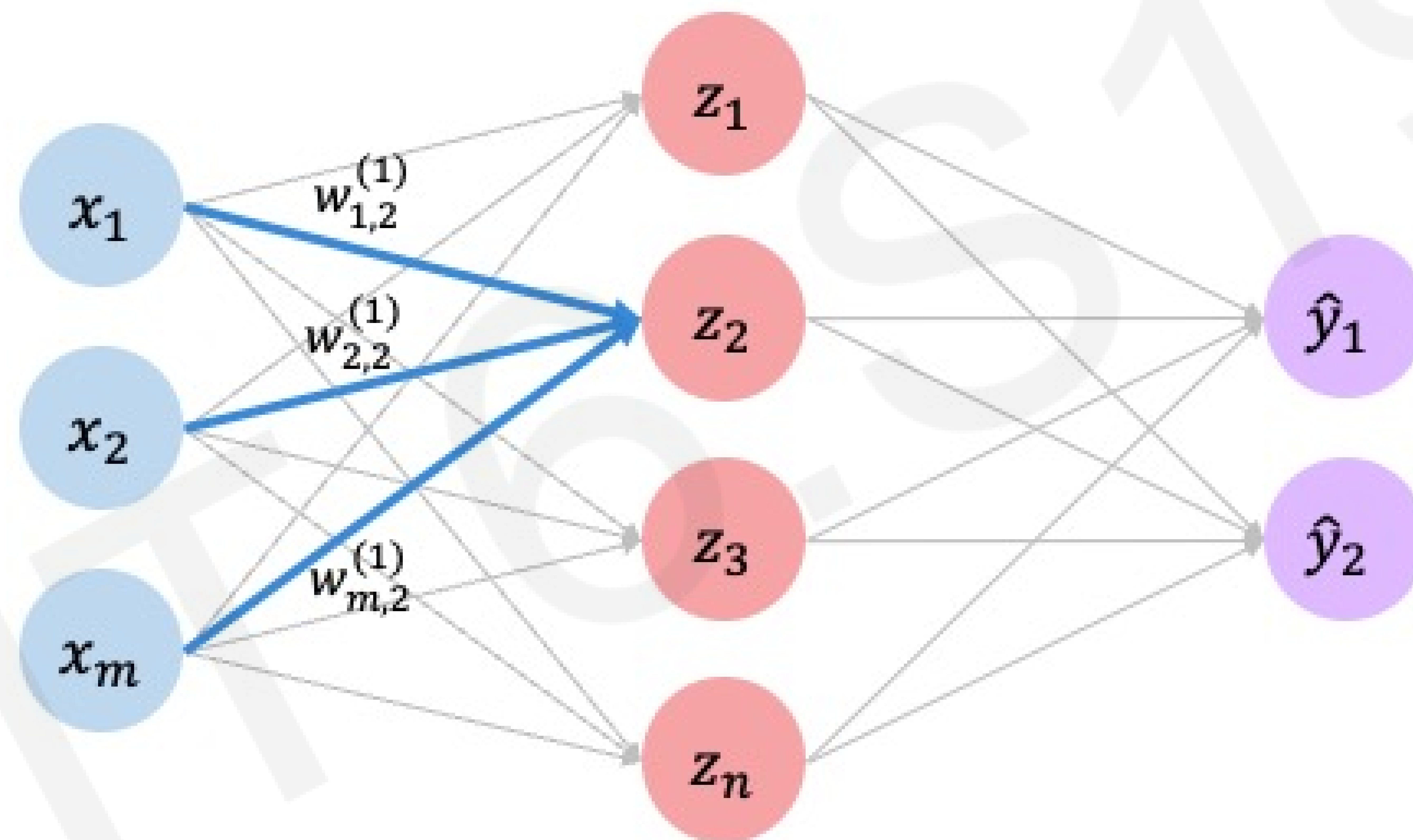
Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) \, w_{j,i}^{(2)} \right)$$

MIT 6.S191

Single Layer Neural Network

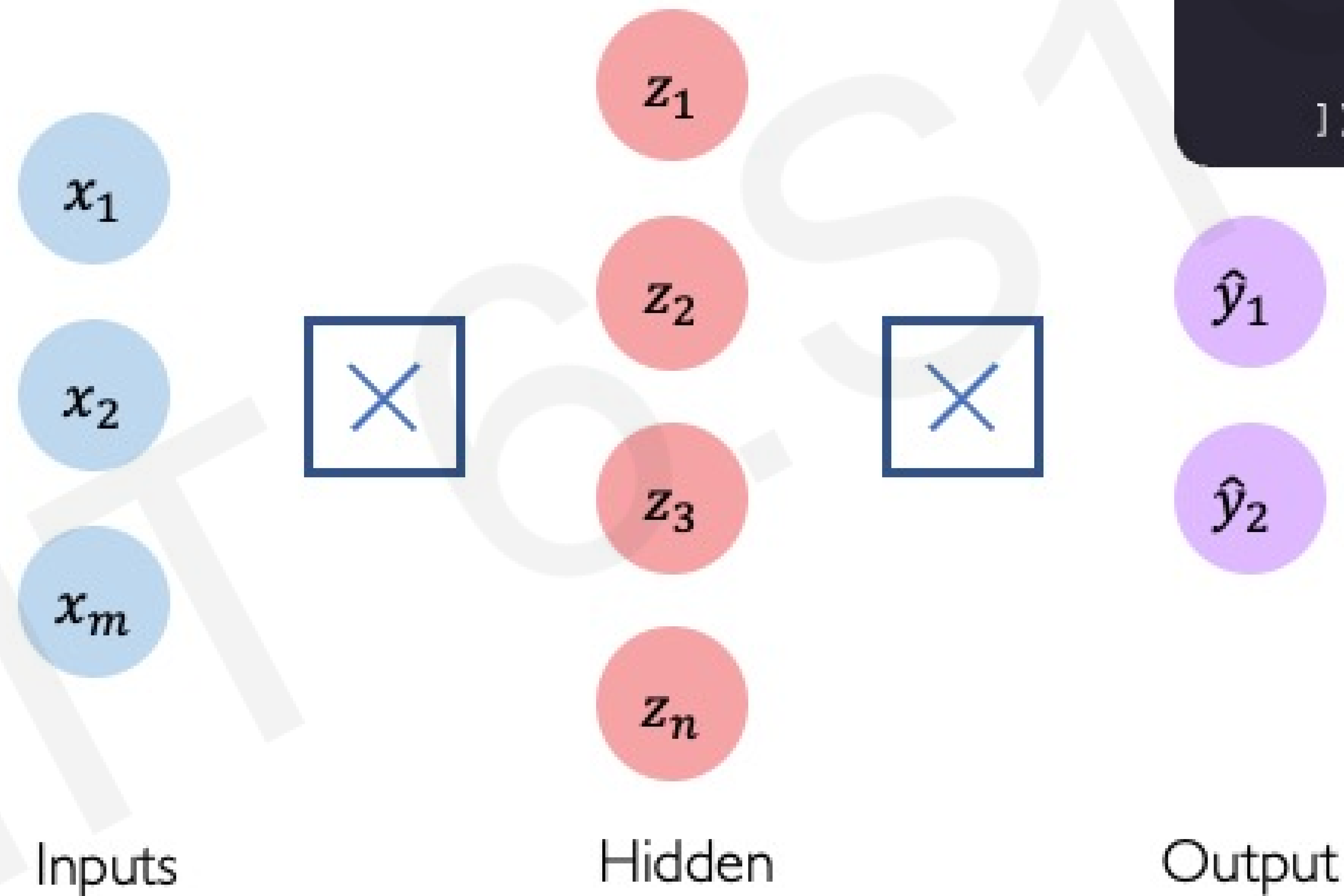


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j\, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1\, w_{1,2}^{(1)} + x_2\, w_{2,2}^{(1)} + x_m\, w_{m,2}^{(1)}$$

MIT 6.S191

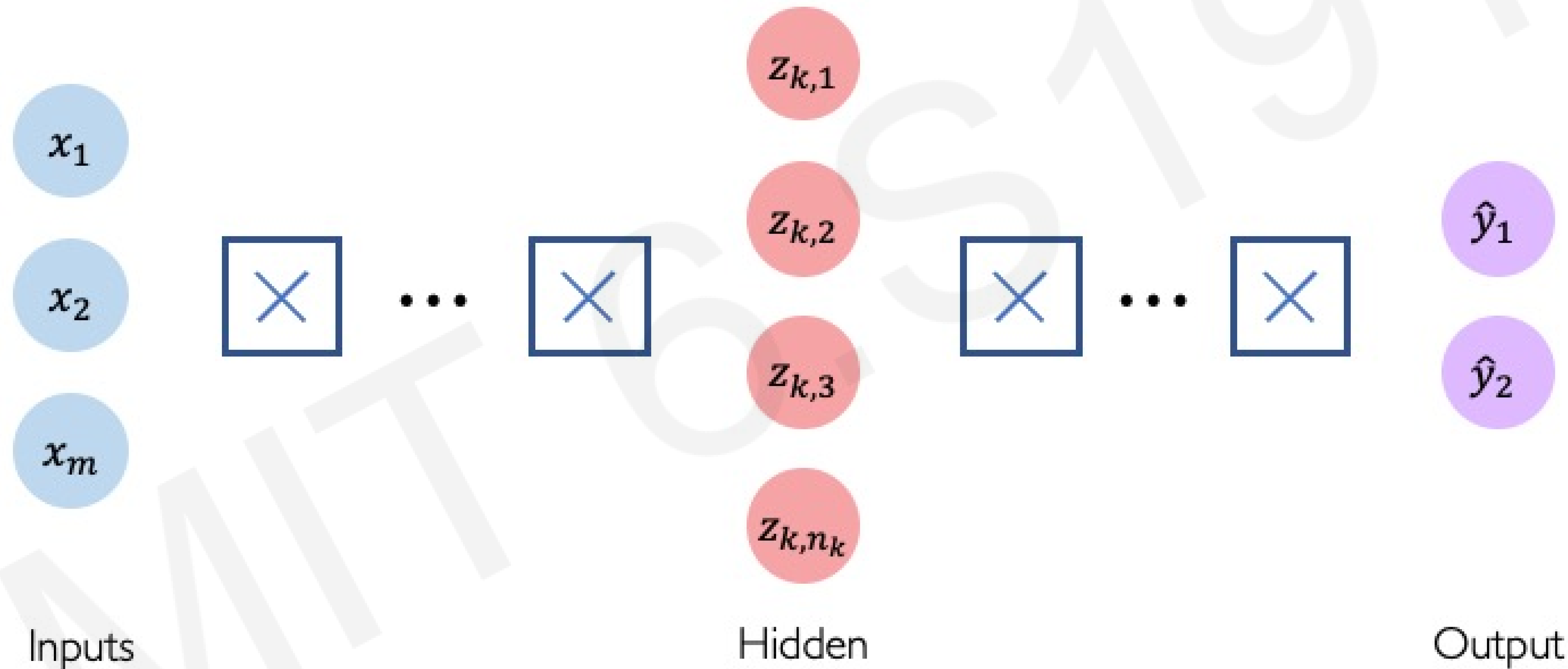
Multi Output Perceptron




```python
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n),
    tf.keras.layers.Dense(2)
])
```

MIT 6.S191

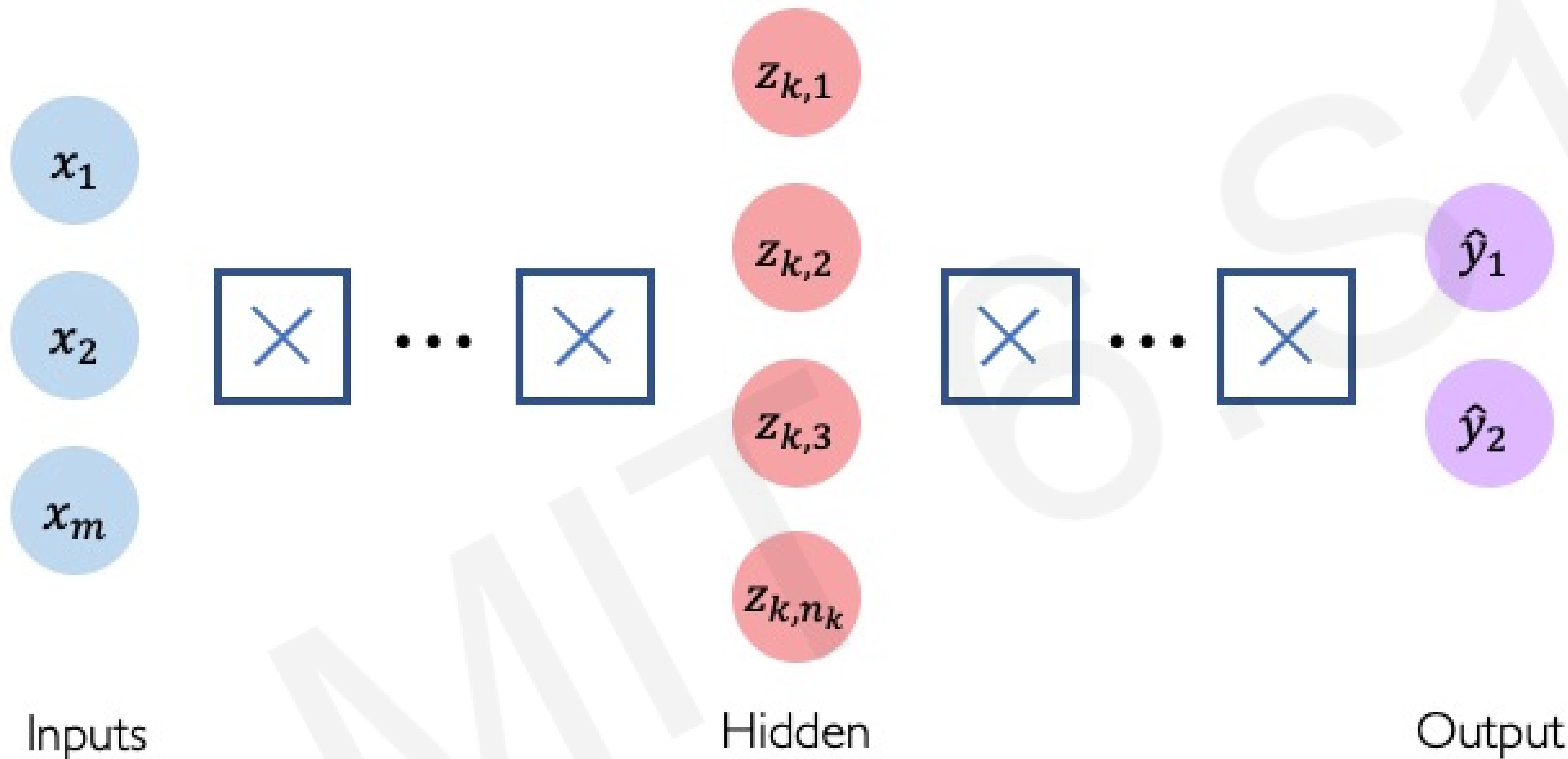
Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

MIT 6.S191

Deep Neural Network




```python
import tensorflow as tf

model = tf.keras.Sequential([
  tf.keras.layers.Dense(n₁),
  tf.keras.layers.Dense(n₂),
  ⋮
  tf.keras.layers.Dense(2)
])
```

$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

MIT 6.S191

Applying Neural Networks

MIT 6.S191

Example Problem

Will I pass this class?

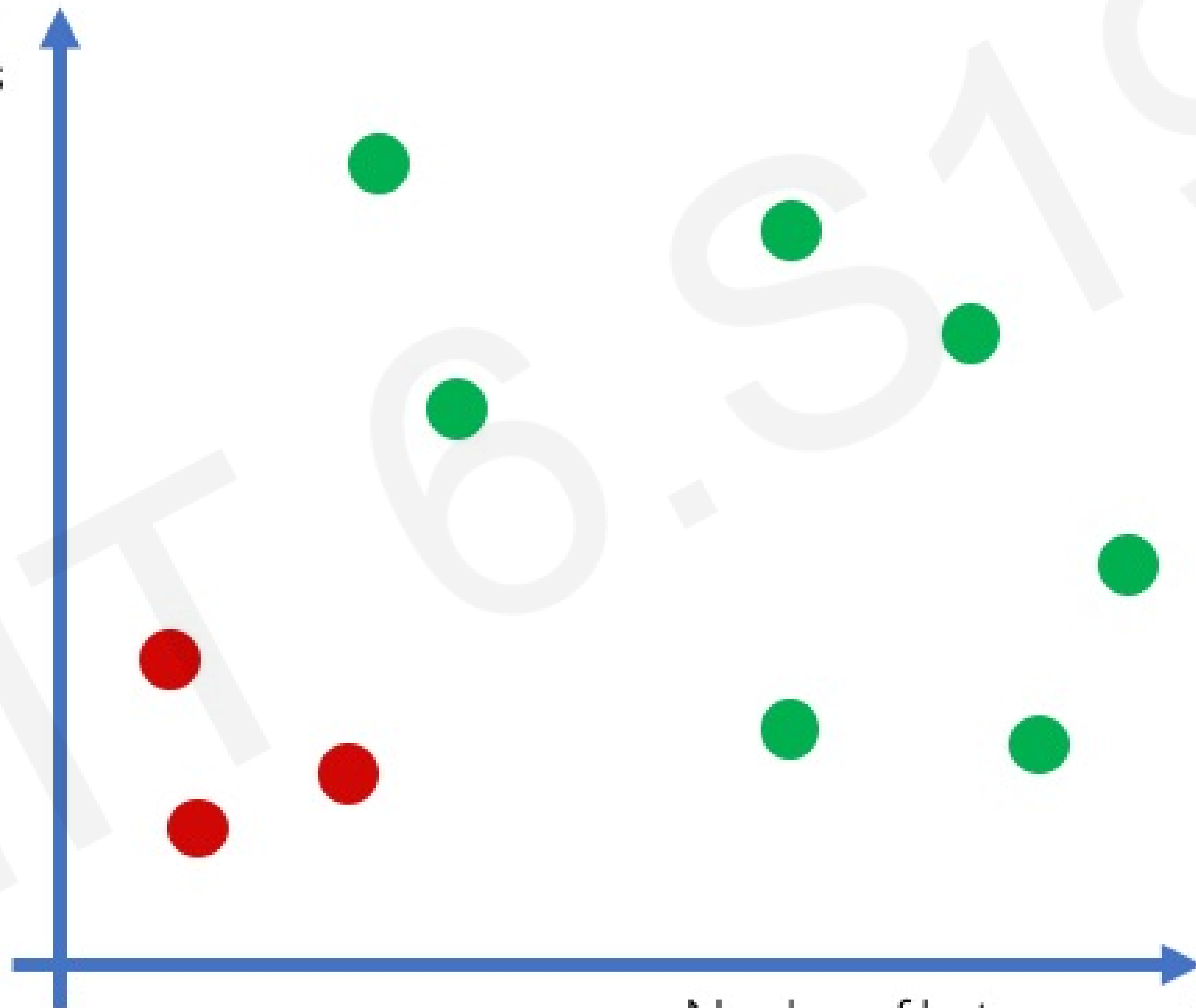
Let's start with a simple two feature model

x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

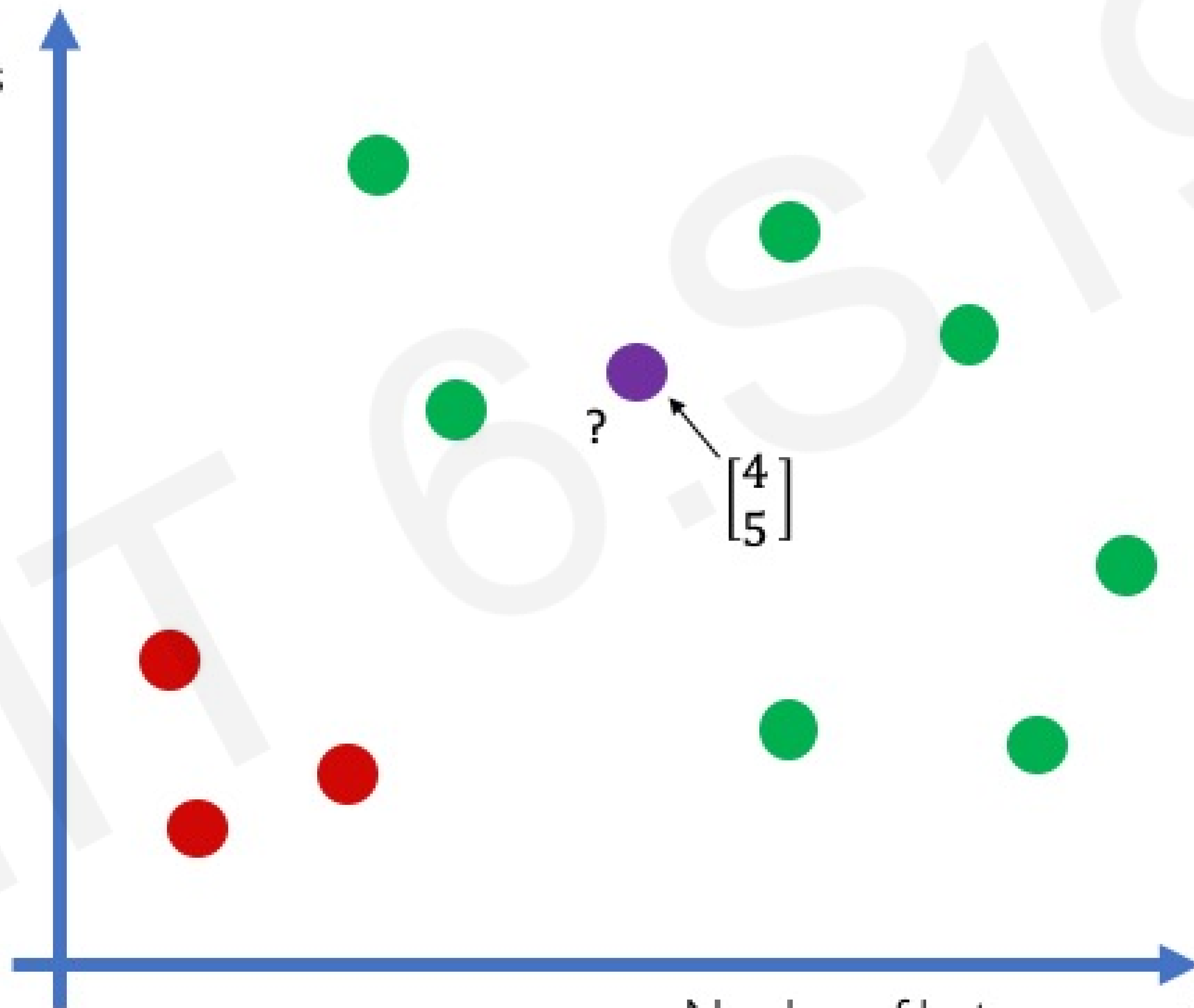
MIT 6.S191

Example Problem: Will I pass this class?



MIT 6.S191

Example Problem: Will I pass this class?

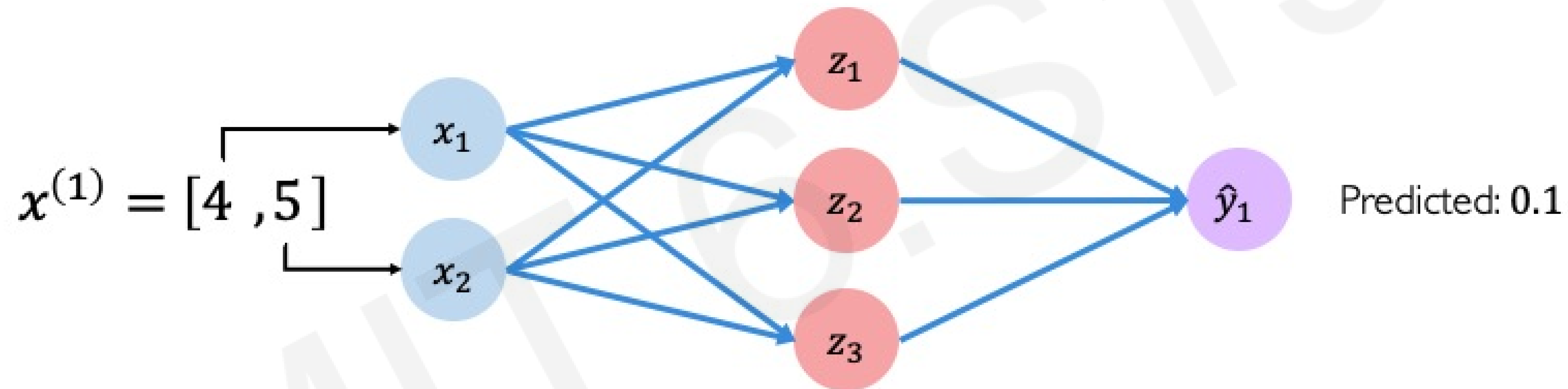


Legend

- Pass
- Fail

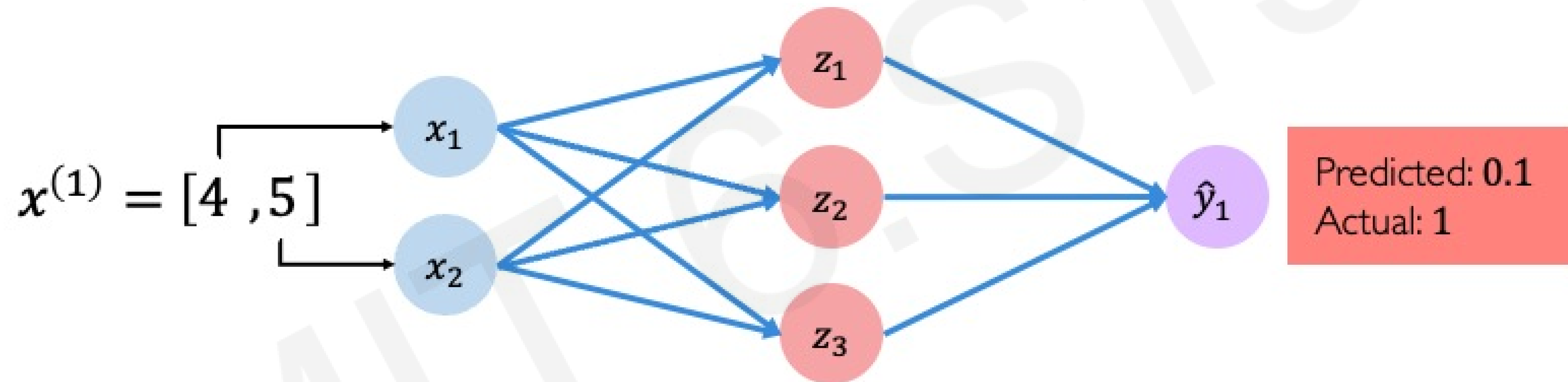
MIT 6.S191

Example Problem: Will I pass this class?



MIT 6.S191

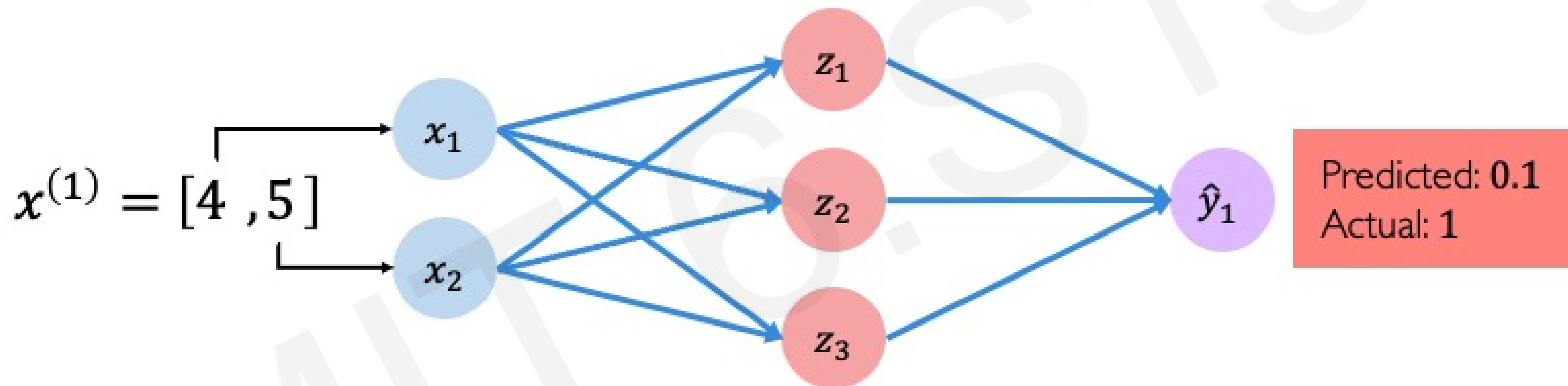
Example Problem: Will I pass this class?



MIT 6.S191

Quantifying Loss

*The **loss** of our network measures the cost incurred from incorrect predictions*



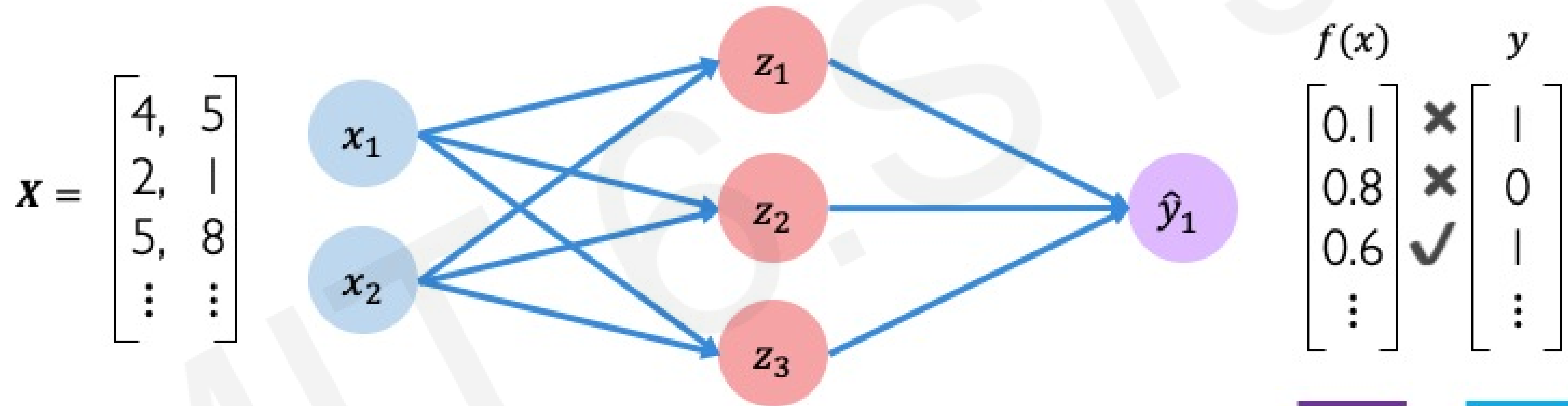
$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

MIT 6.S191

Empirical Loss

*The **empirical loss** measures the total loss over our entire dataset*



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

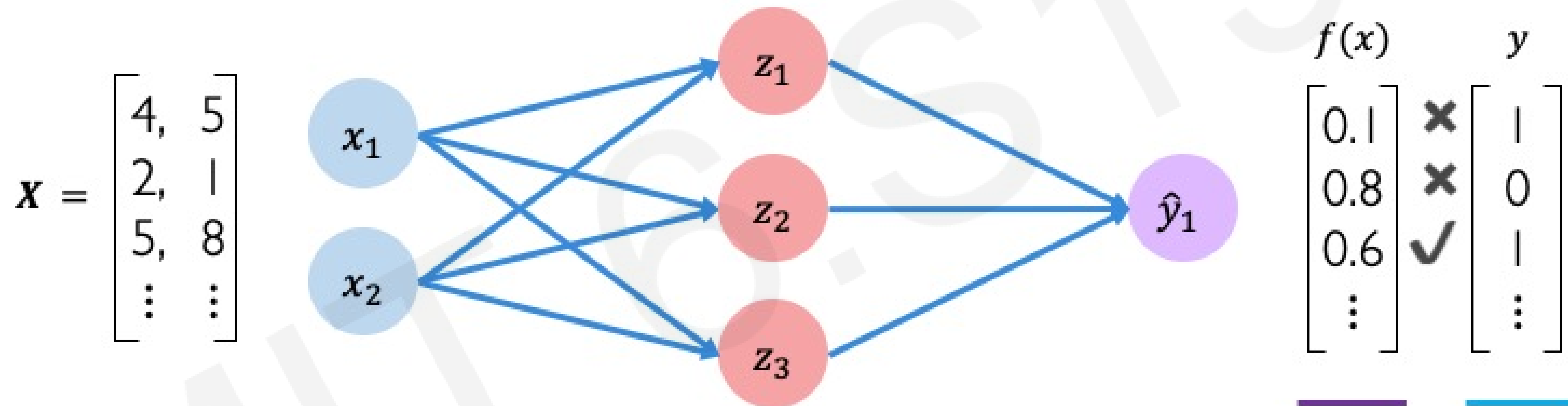
Also known as:

- Objective function
- Cost function
- Empirical Risk

MIT 6.S191

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*



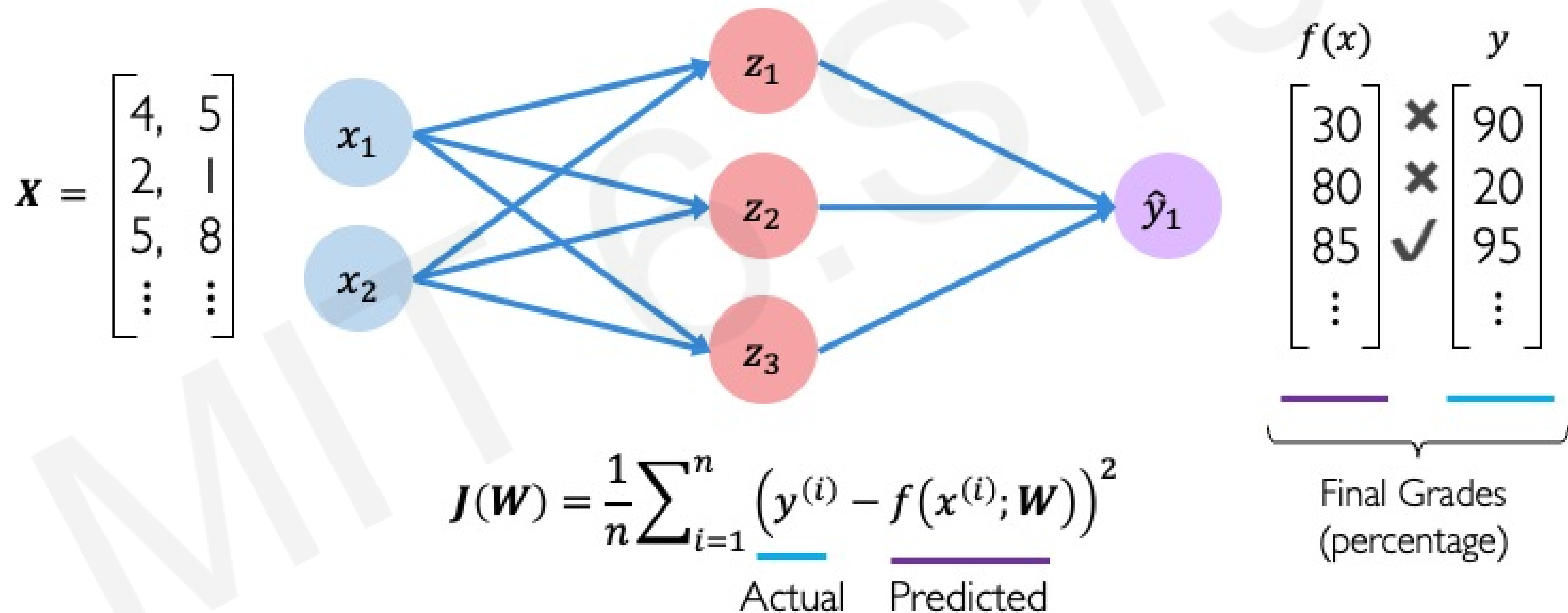
$$J(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1 - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

MIT 6.S191

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*




```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
loss = tf.keras.losses.MSE( y, predicted )
```

MIT 6.S191

Training Neural Networks

MIT 6.S191

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

MIT 6.S191

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

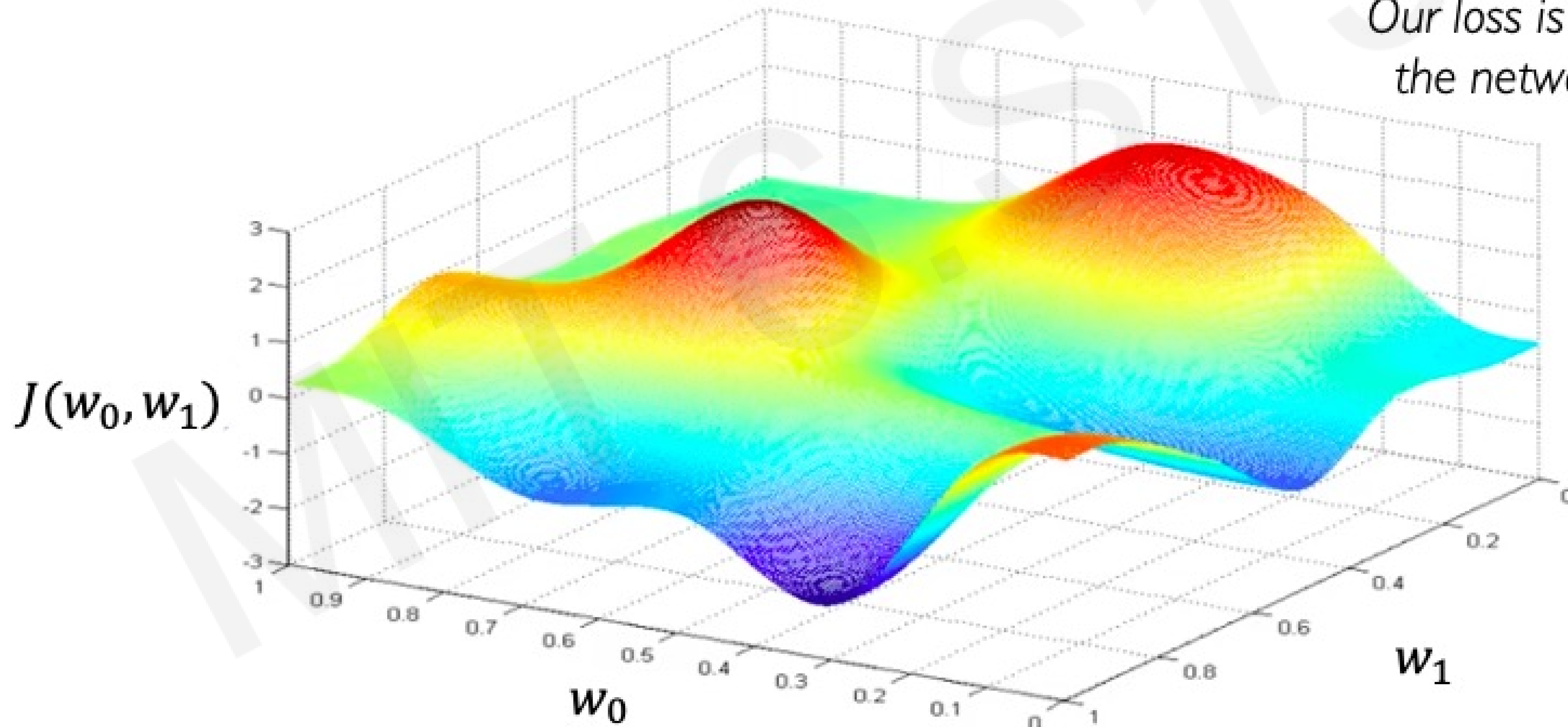
Remember:
$\boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\}$

MIT 6.S191

Loss Optimization

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

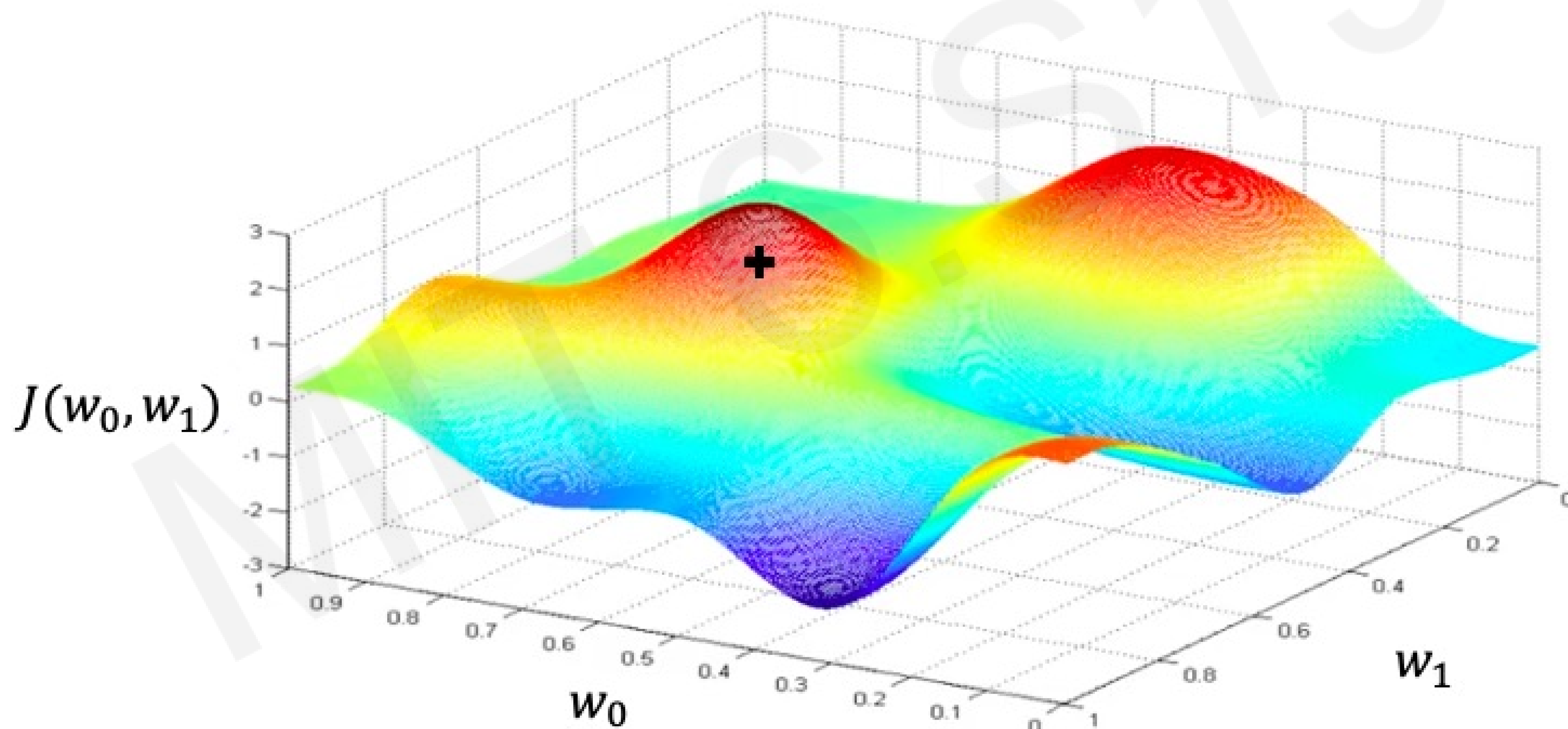
Remember:
Our loss is a function of the network weights!



MIT 6.S191

Loss Optimization

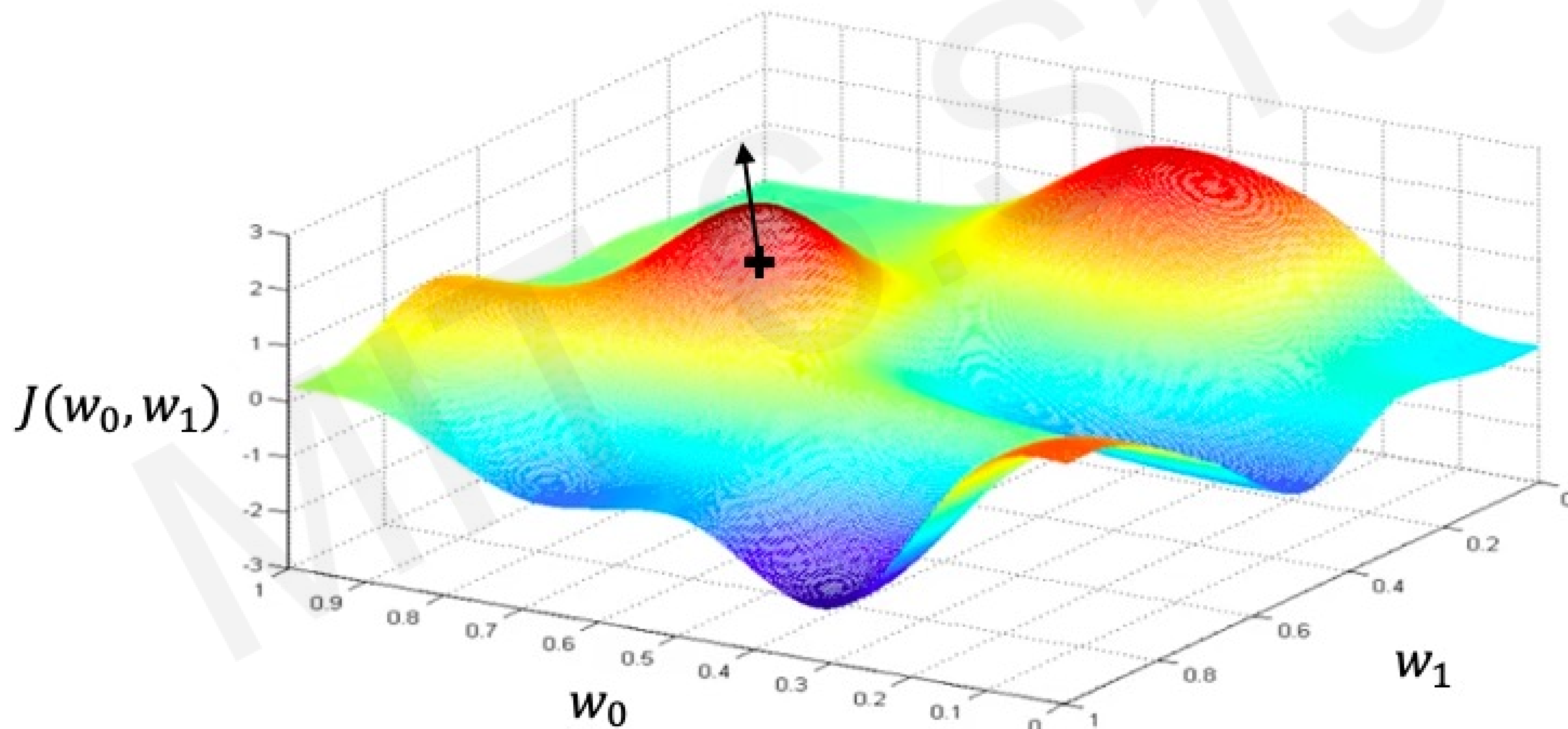
Randomly pick an initial (w_0, w_1)



MIT 6.S191

Loss Optimization

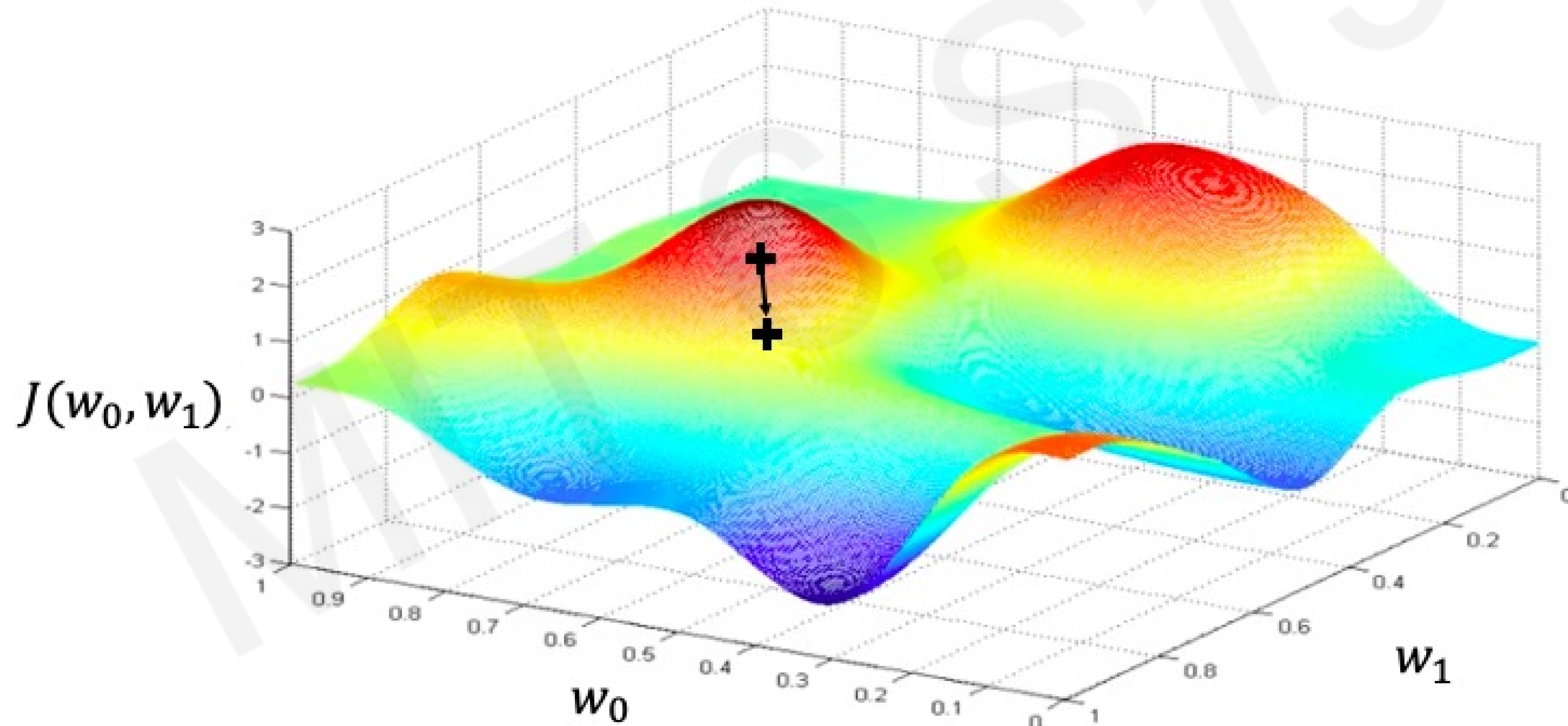
Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$



MIT 6.S191

Loss Optimization

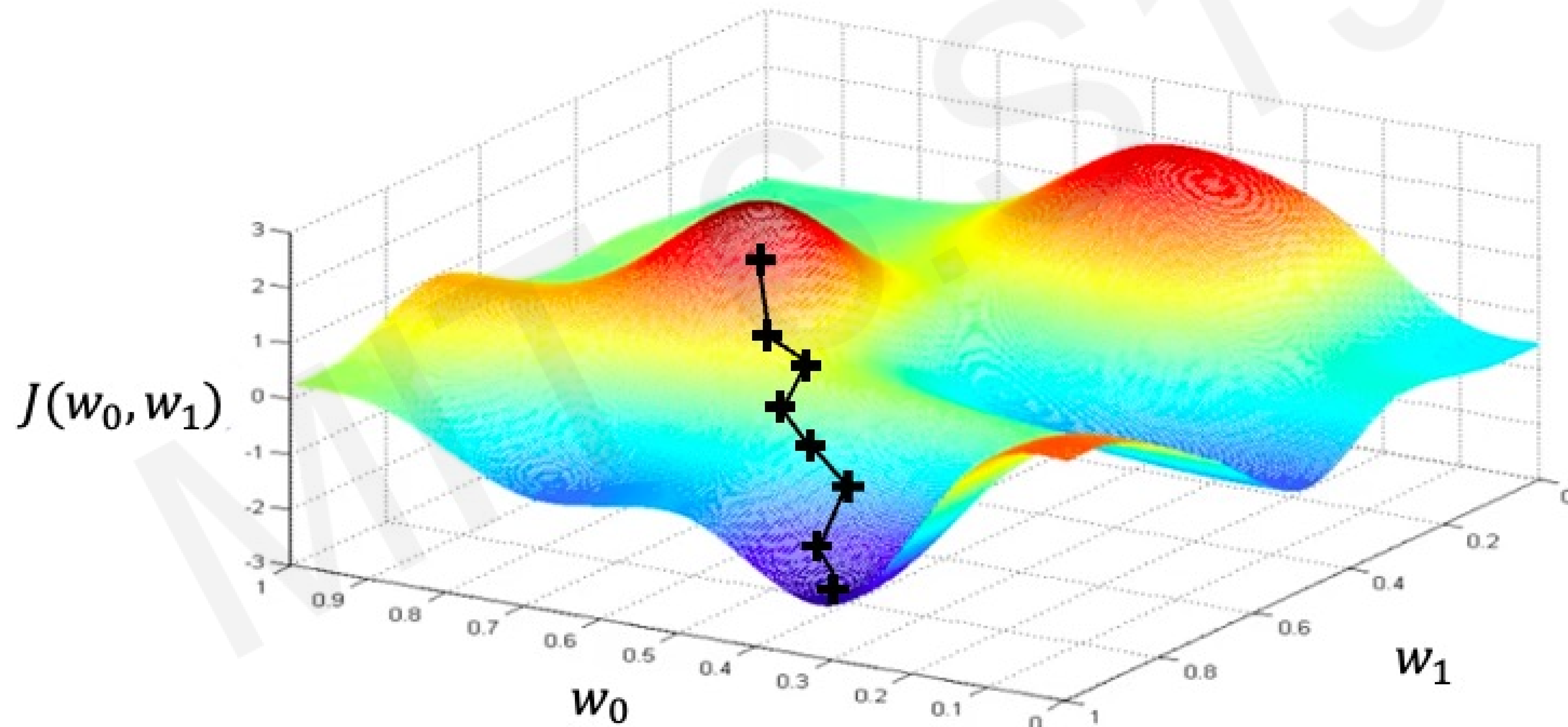
Take small step in opposite direction of gradient



MIT 6.S191

Gradient Descent

Repeat until convergence



MIT 6.S191

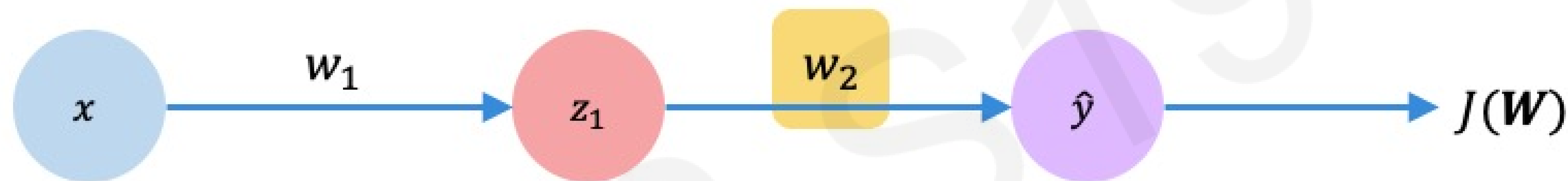
Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

MIT 6.S191

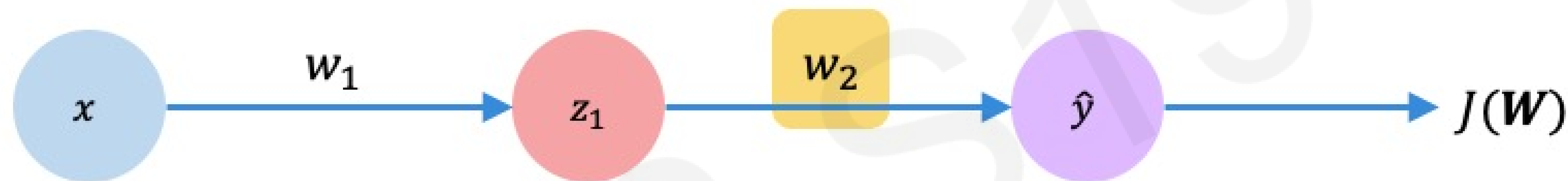
Computing Gradients: Backpropagation



How does a small change in one weight (ex. $\boldsymbol{w_2}$) affect the final loss $\boldsymbol{J(W)}$?

MIT 6.S191

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} =$$

Let's use the chain rule!

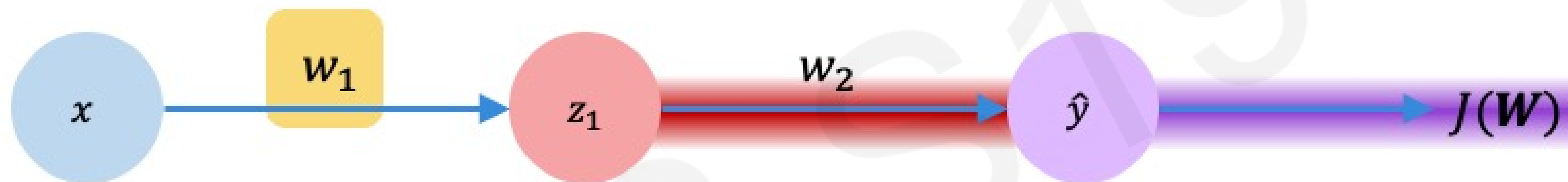
MIT 6.S191

Computing Gradients: Backpropagation

$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

MIT 6.S191

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

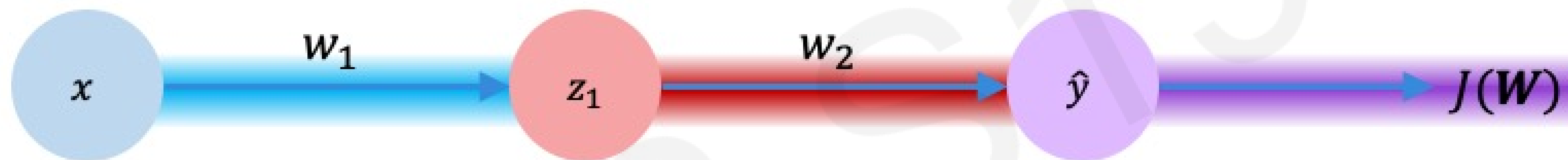
MIT 6.S191

Computing Gradients: Backpropagation

$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

MIT 6.S191

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

MIT 6.S191

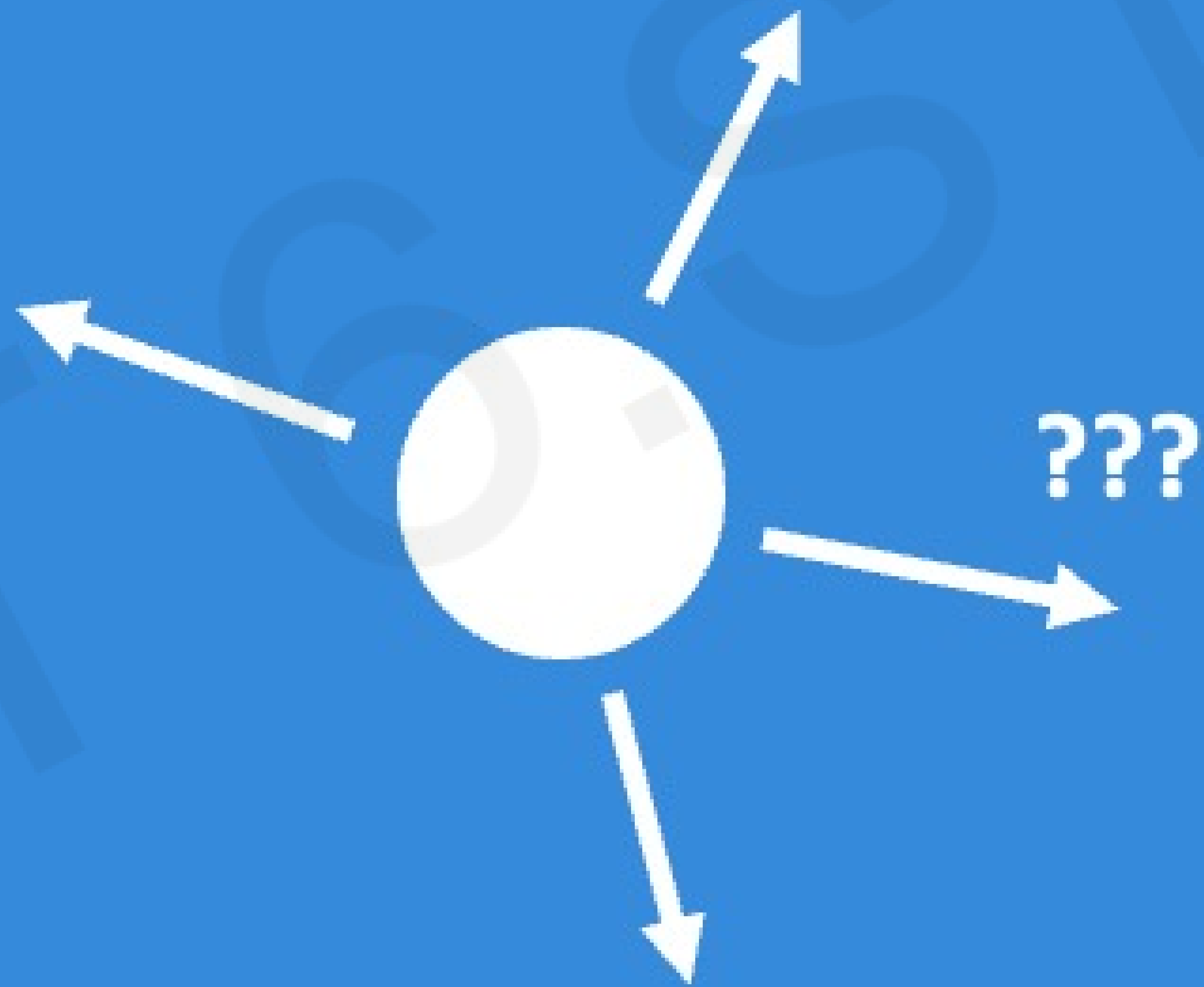
Outline

1. NLP : What?
    1. N-Gram Models
2. Intro Deep Learning for NLP
    1. Where DL sits in the Machine Learning landscape.
    2. Perceptrons/Neurons, Feed Forward units
    3. Neural Networks, Deep Networks and Training
3. **NNs for Sequence Modeling , RNN**
    1. **Attention/Self-Attention**
    2. **LSTMs (tangentially)**
    3. **Encoder-Decoders**

Given an image of a ball,
can you predict where it will go next?

MIT 6.S191

Given an image of a ball,
can you predict where it will go next?



MIT 6.S191

Given an image of a ball,
can you predict where it will go next?

MIT 6.S191

Given an image of a ball, can you predict where it will go next?

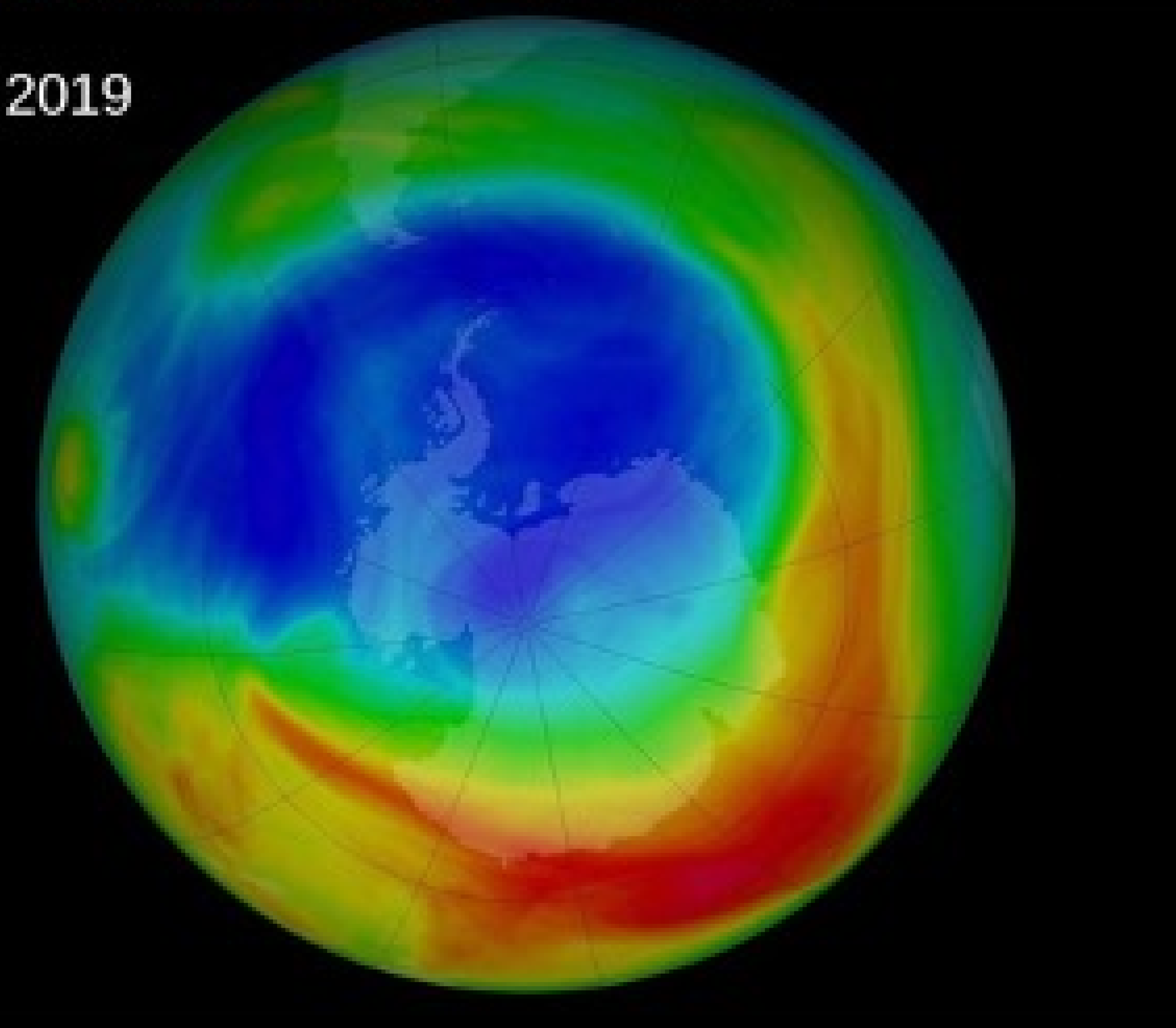
MIT 6.S191

Sequences in the Wild

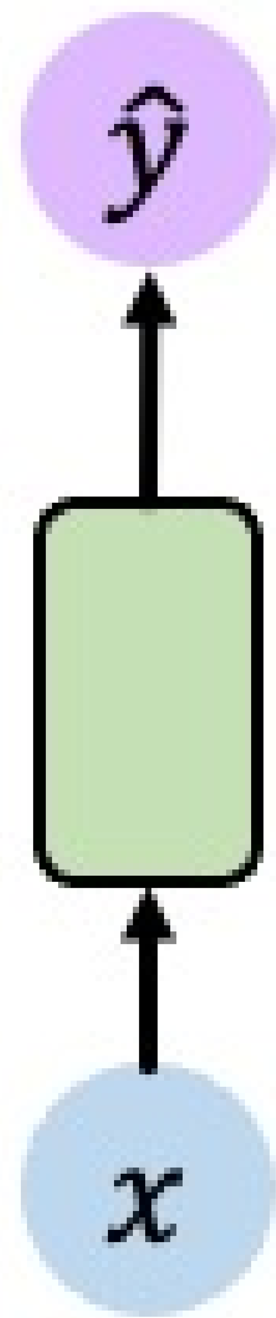
Audio

MIT 6.S191

Sequences in the Wild

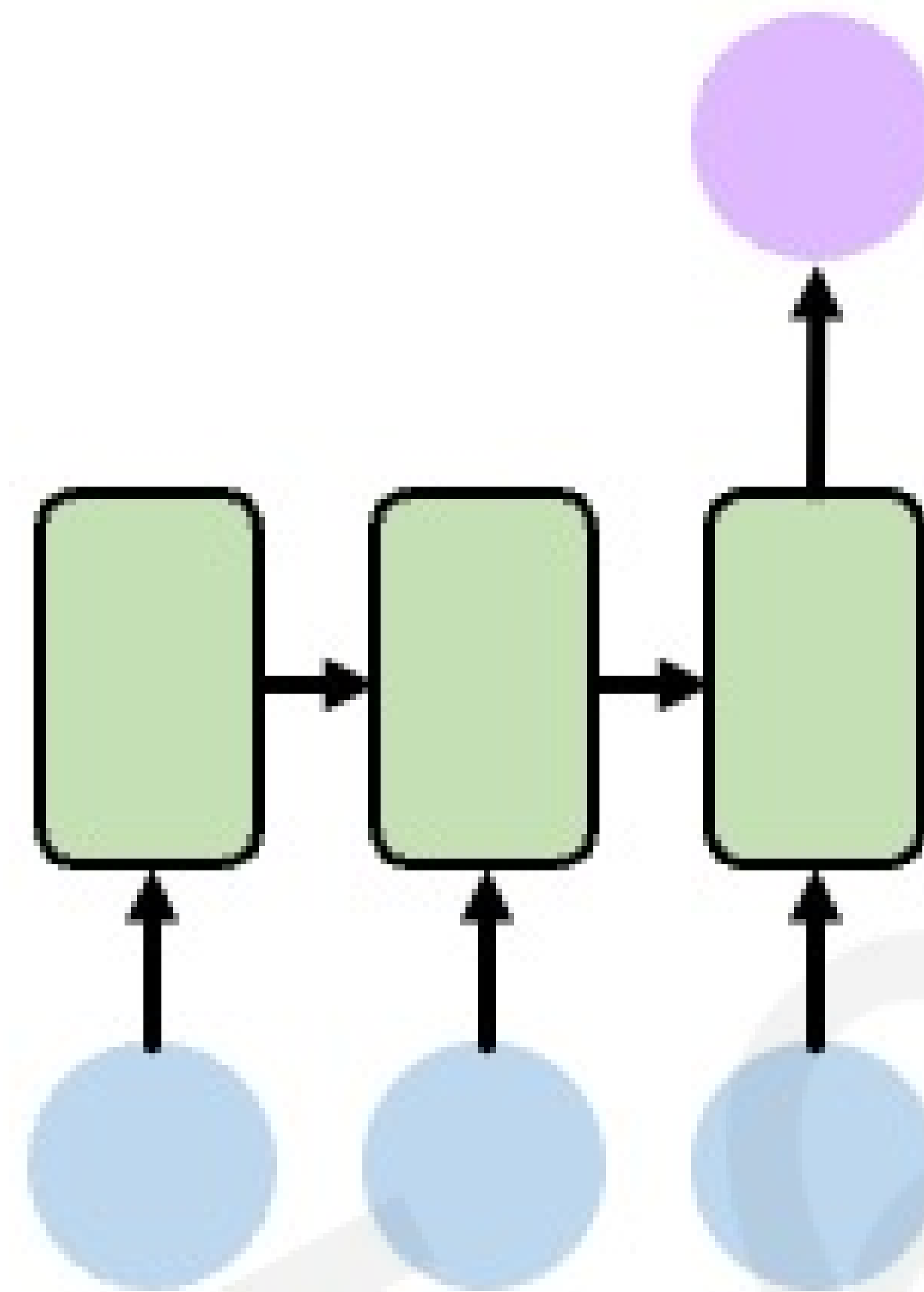


Sequence Modeling Applications

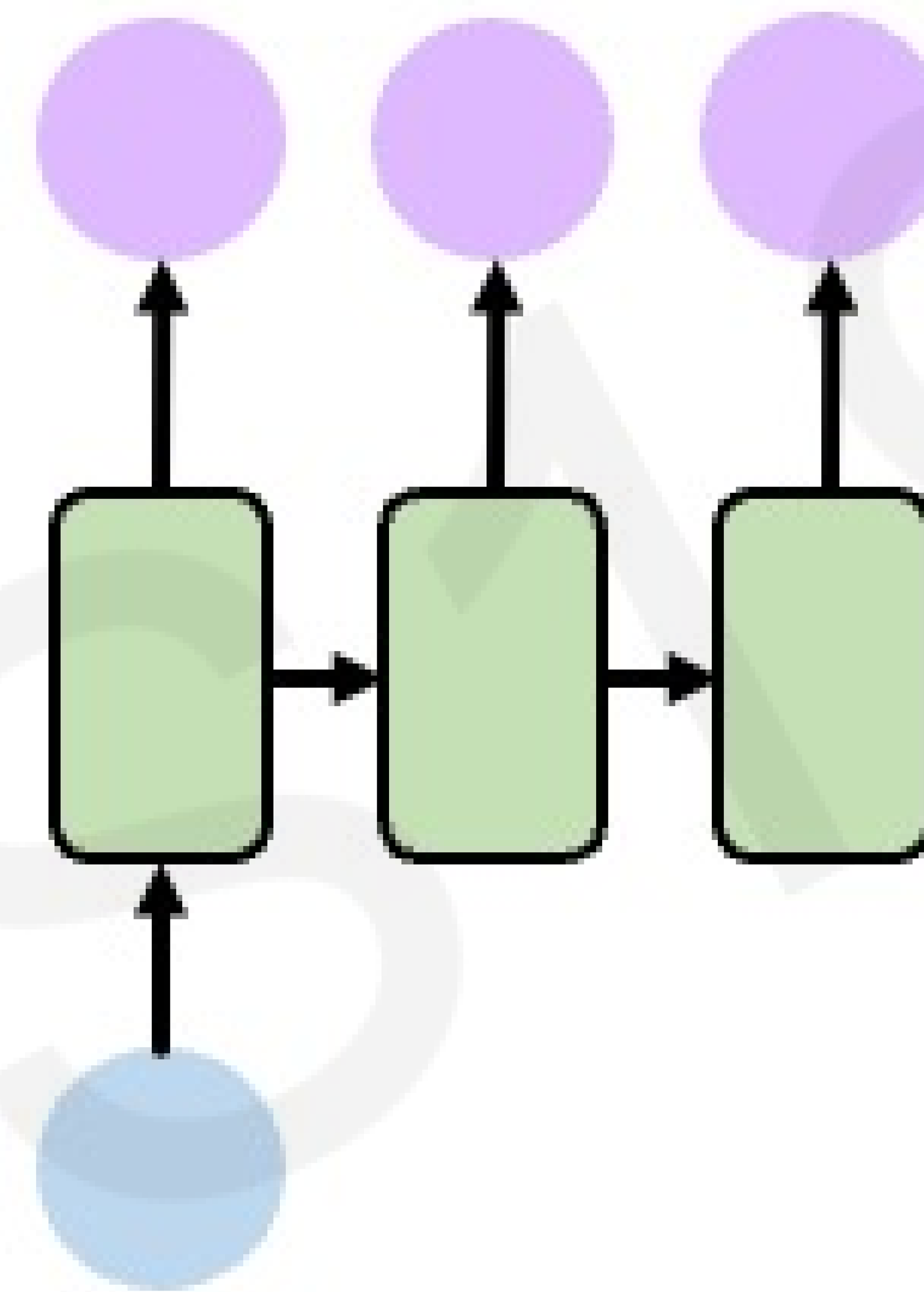


One to One
Binary Classification

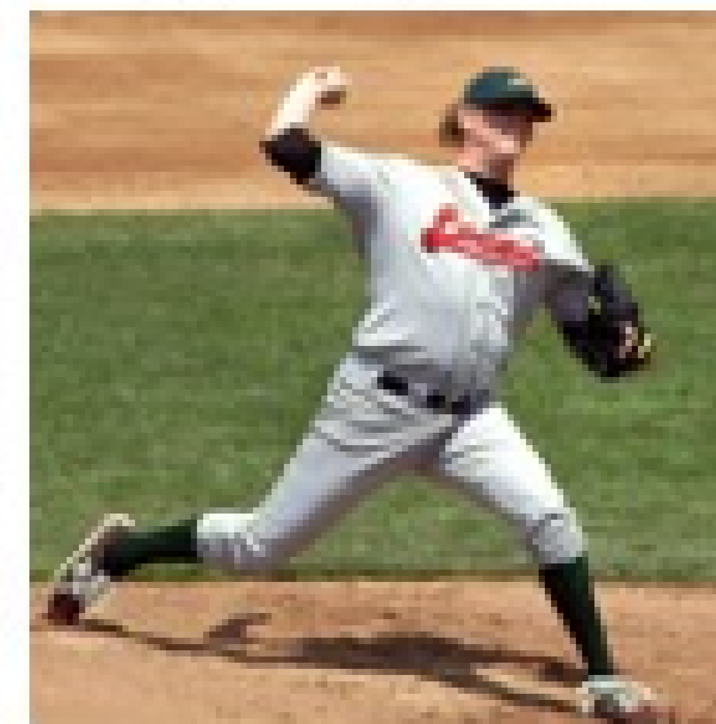
"Will I pass this class?"
Student → Pass?



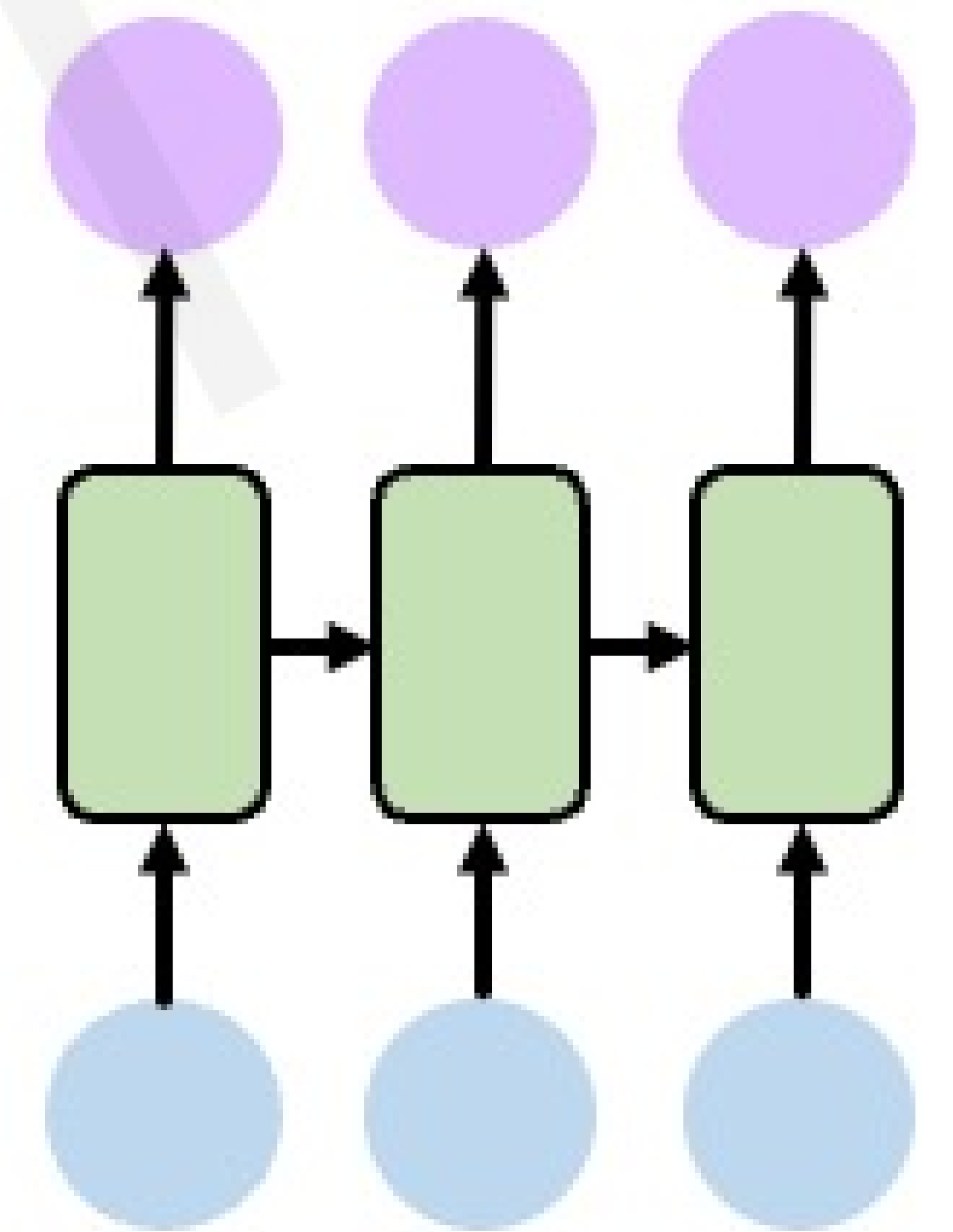
Many to One
Sentiment Classification



One to Many
Image Captioning



"A baseball player throws a ball."



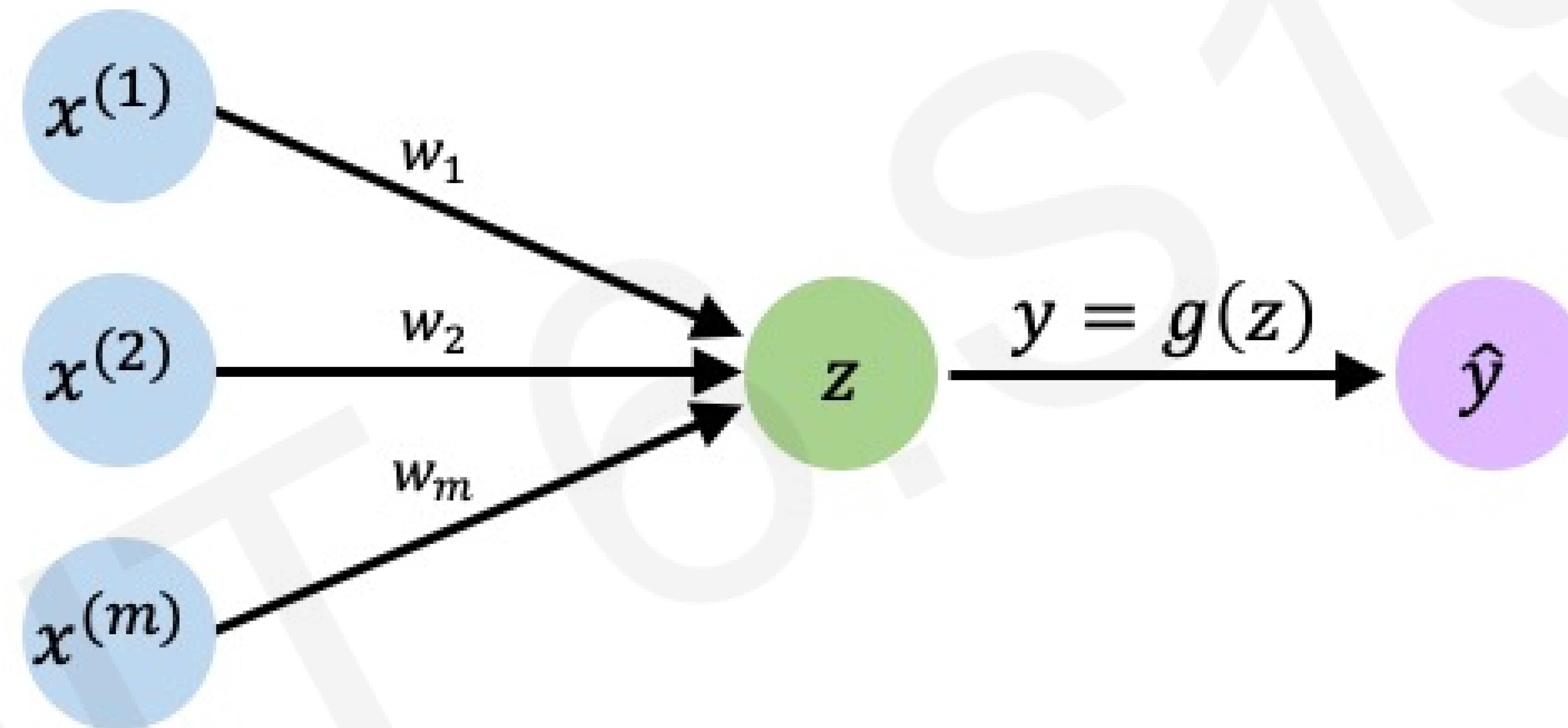
Many to Many
Machine Translation

MIT 6.S191

Neurons with Recurrence

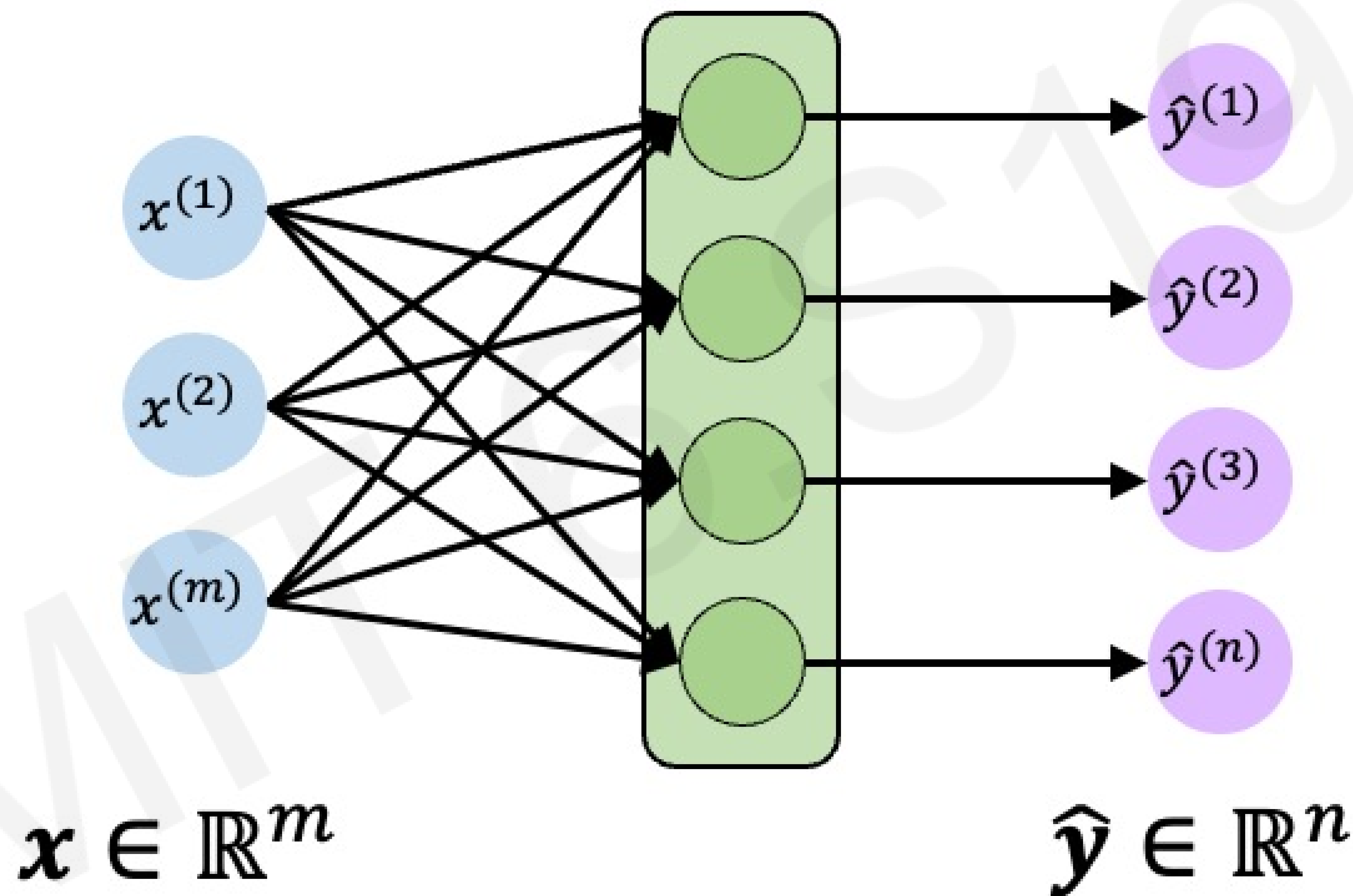
MIT 6.S191

The Perceptron Revisited



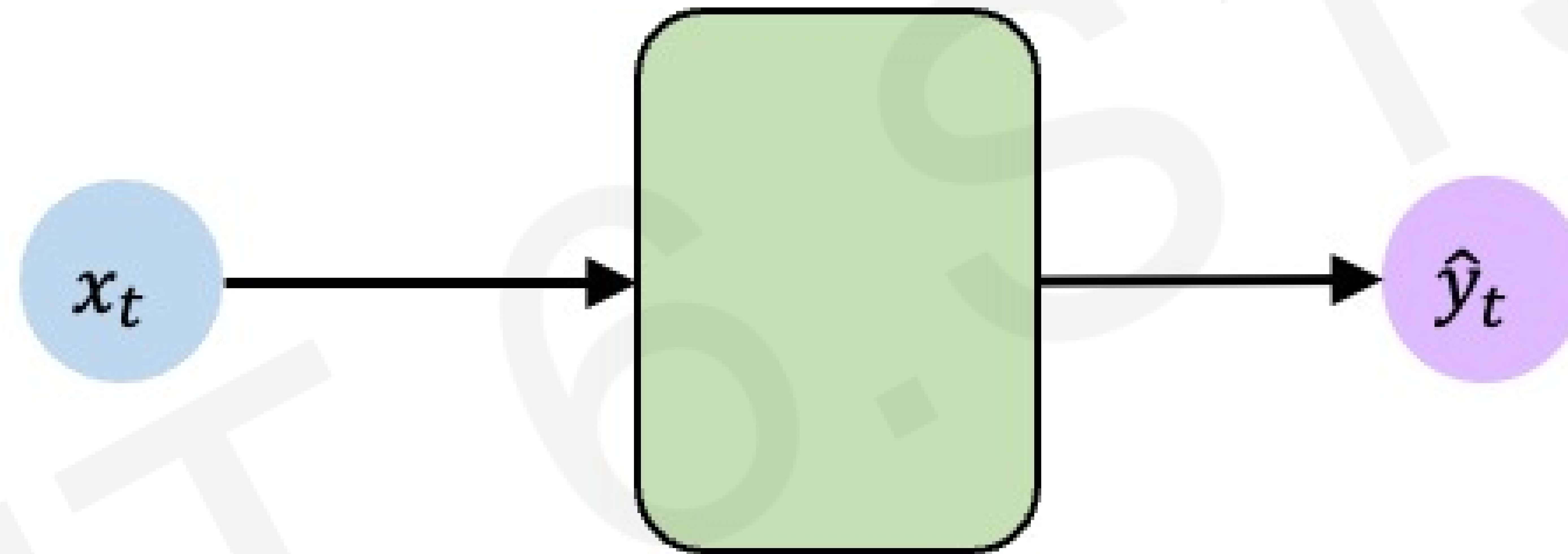
MIT 6.S191

Feed-Forward Networks Revisited



MIT 6.S191

Feed-Forward Networks Revisited



$\boldsymbol{x_t} \in \mathbb{R}^m$

$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$

MIT 6.S191

Handling Individual Time Steps

MIT 6.S191

Neurons with Recurrence

MIT 6.S191

Neurons with Recurrence

$$\hat{y}_t = f(x_t, h_{t-1})$$

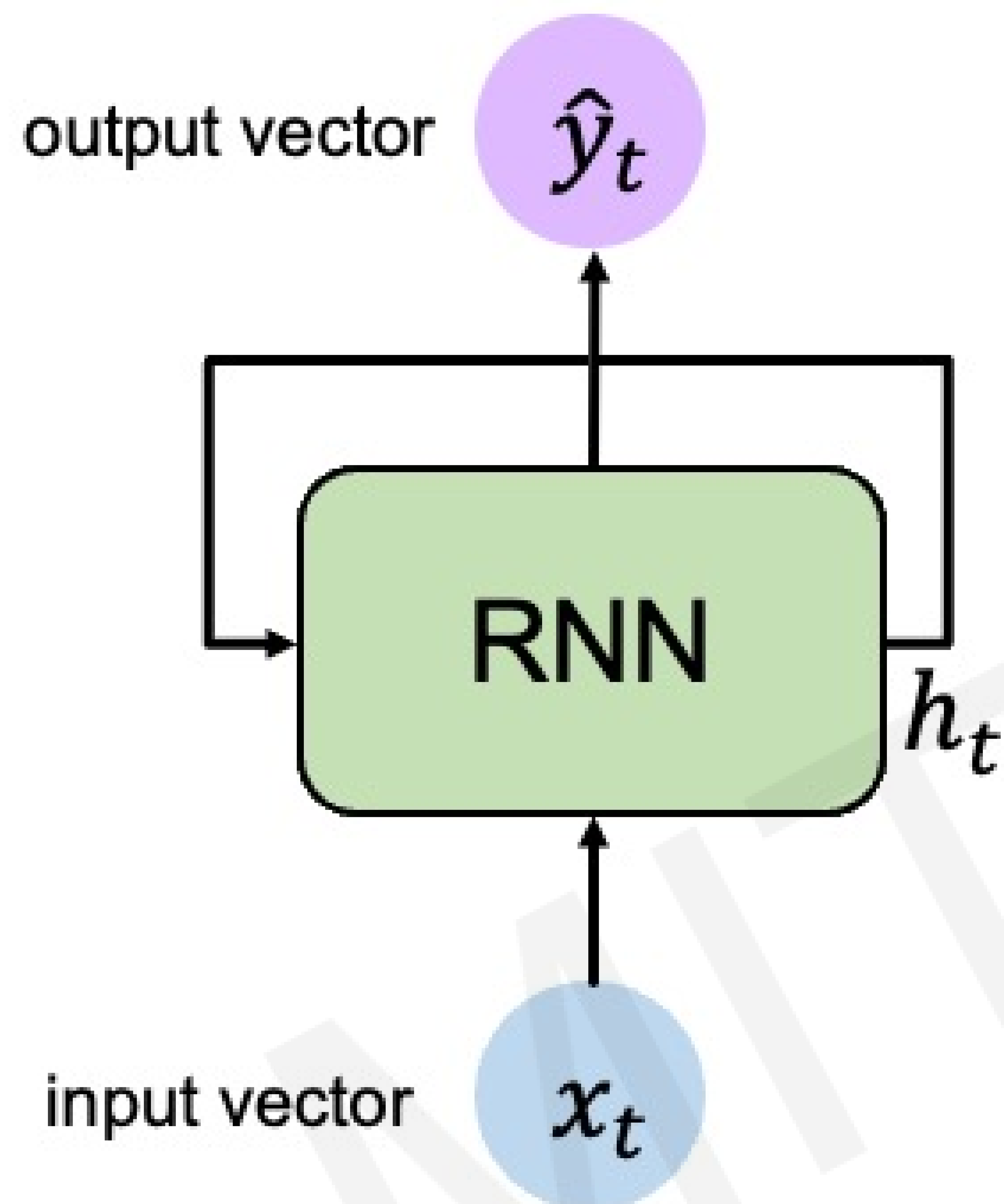
output — input — past memory

MIT 6.S191

Recurrent Neural Networks (RNNs)

MIT 6.S191

Recurrent Neural Networks (RNNs)



Apply a **recurrence relation** at every time step to process a sequence:

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

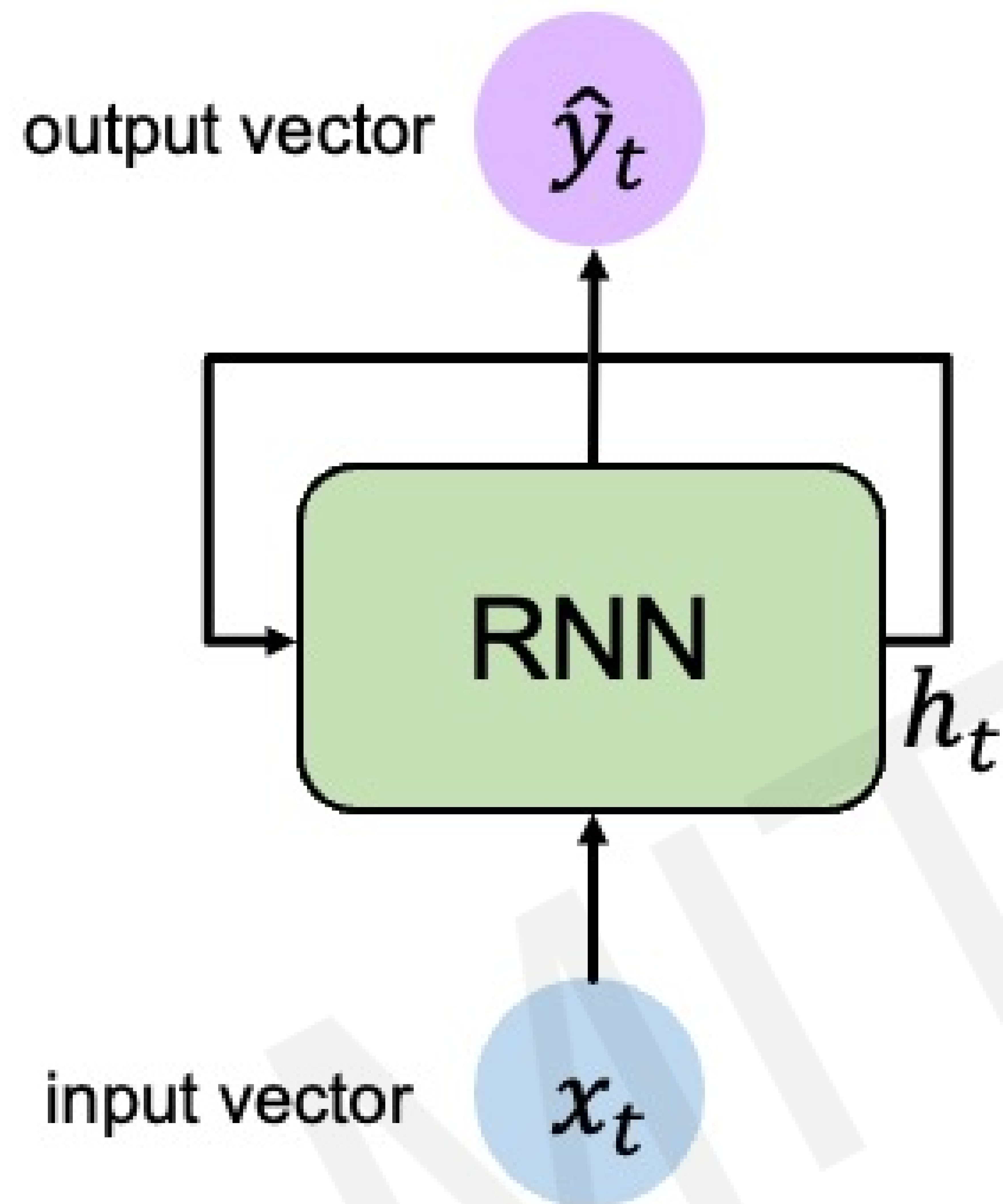
cell state — function with weights W — input — old state

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

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

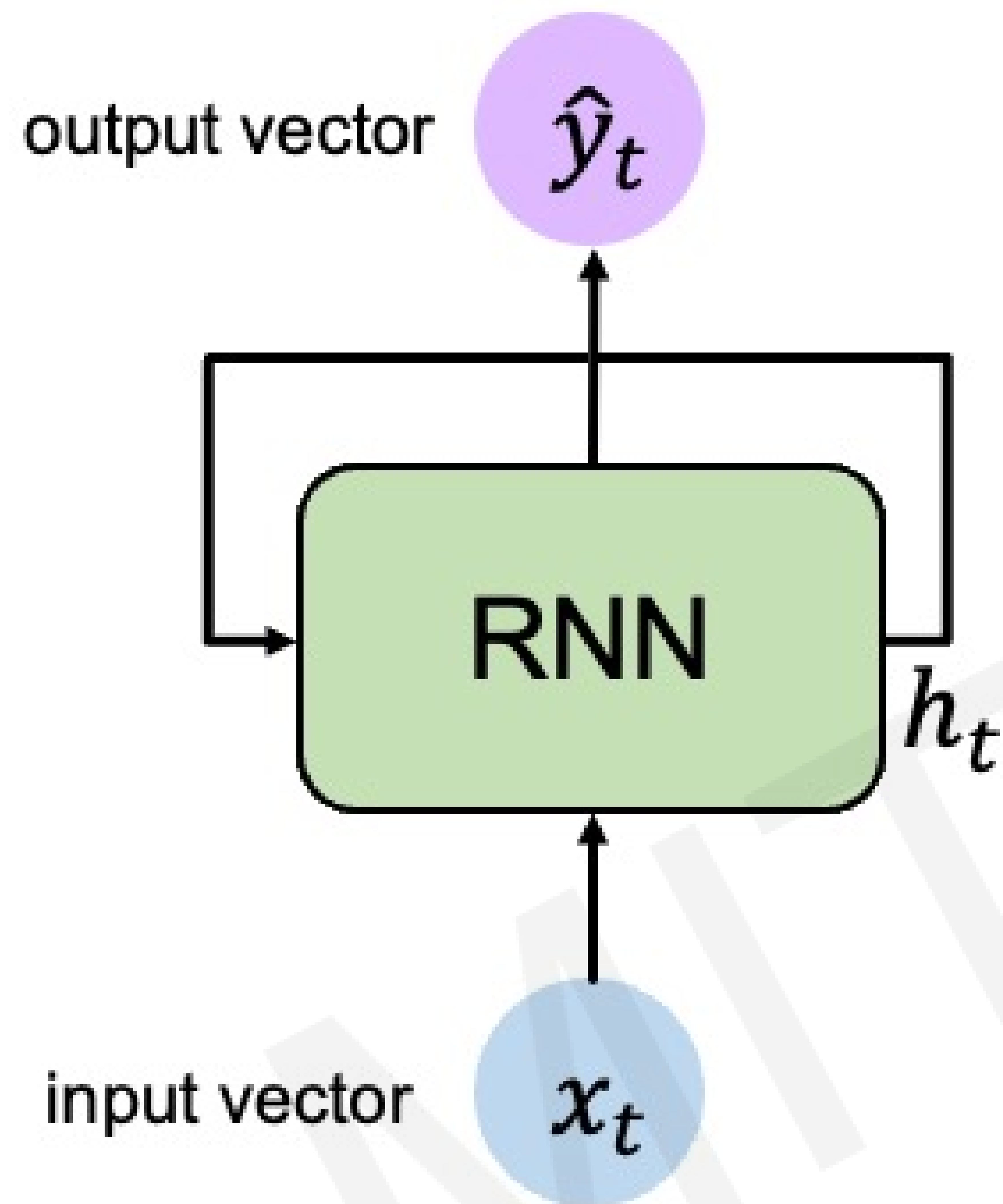
MIT 6.S191

RNN State Update and Output



MIT 6.S191

RNN State Update and Output

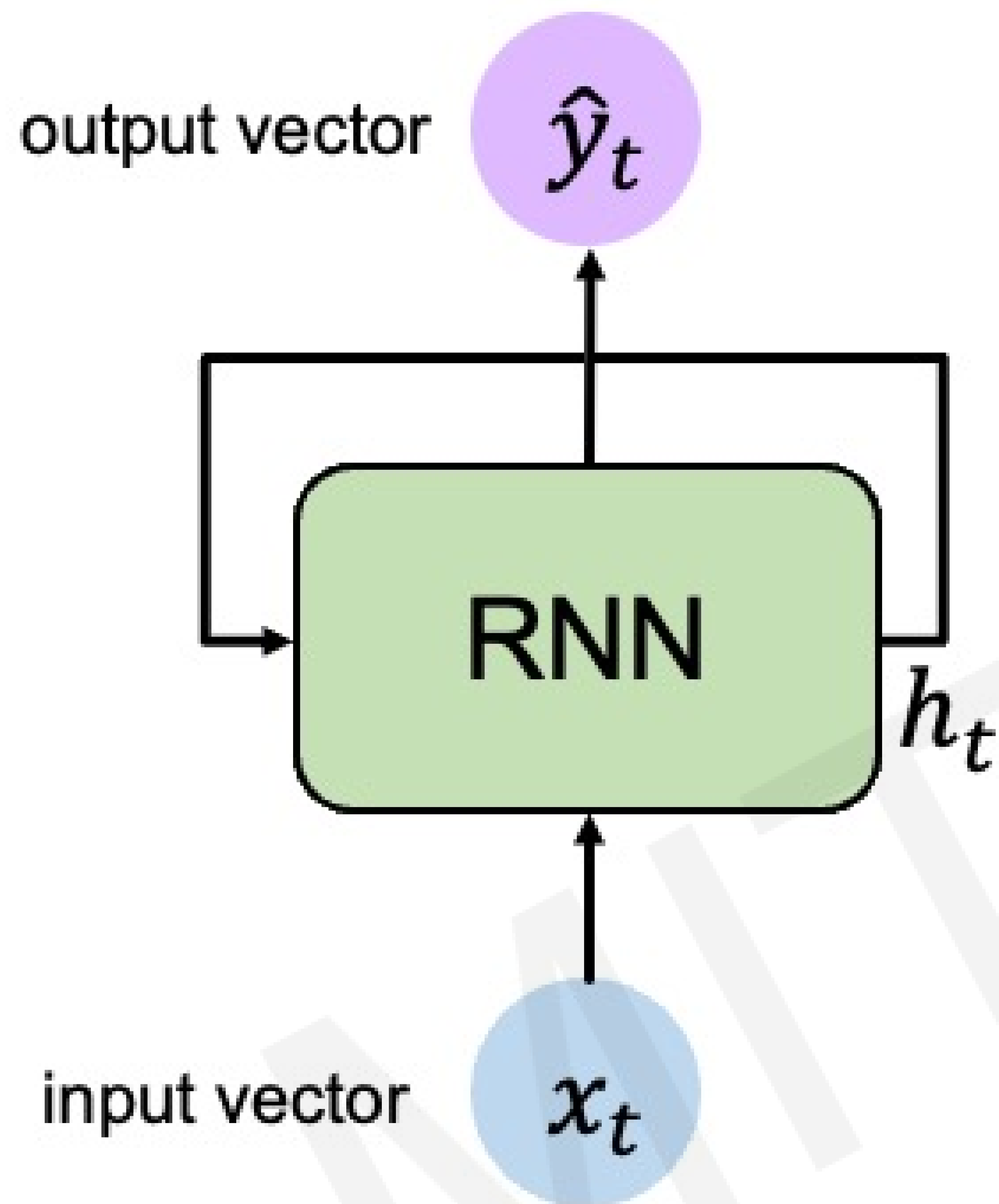


Input Vector

x_t

MIT 6.S191

RNN State Update and Output



Update Hidden State

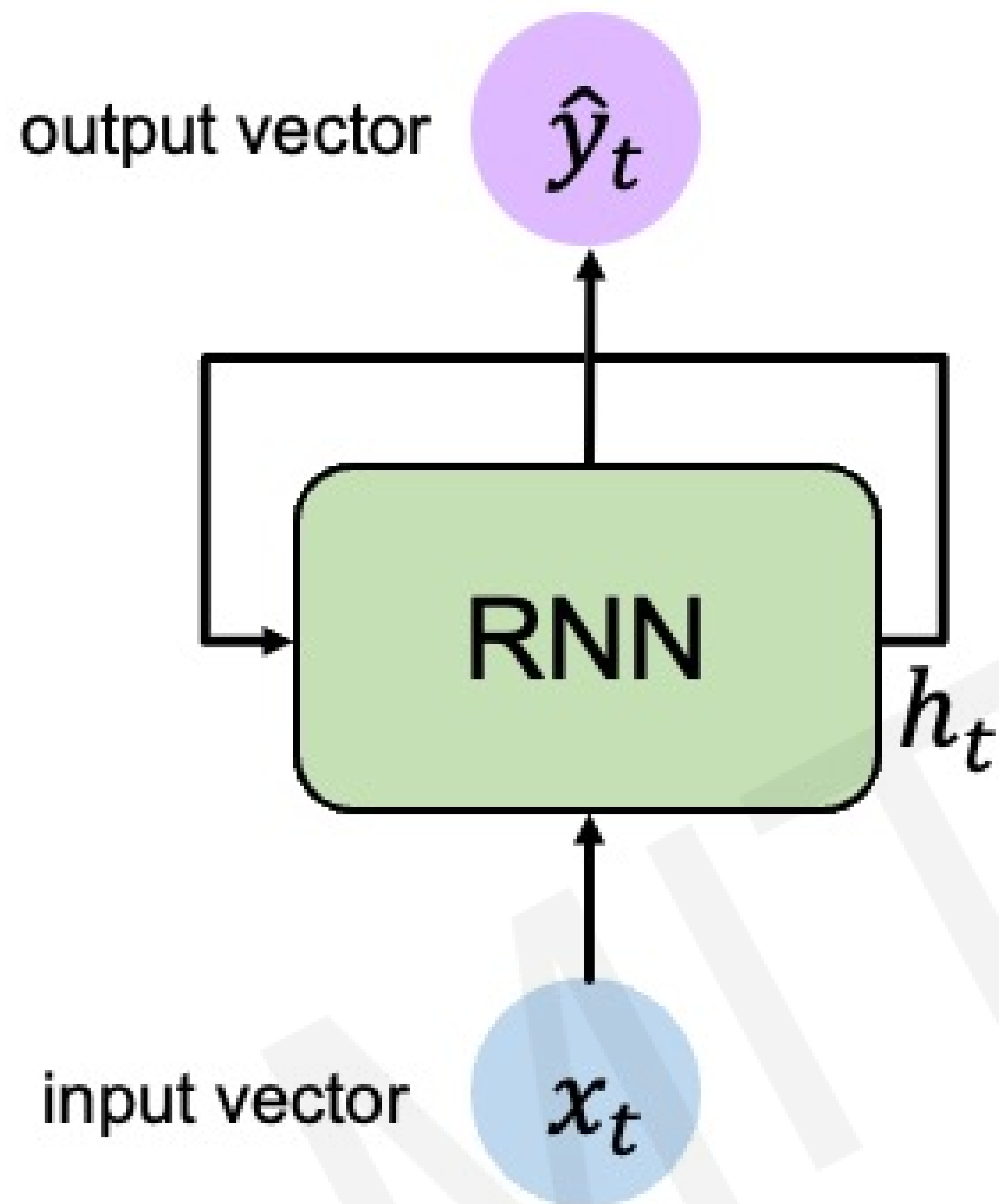
$$h_t = \tanh(\boldsymbol{W}_{hh}^T h_{t-1} + \boldsymbol{W}_{xh}^T x_t)$$

Input Vector

$$x_t$$

MIT 6.S191

RNN State Update and Output



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{hy}^T h_t$$

Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh}^T h_{t-1} + \boldsymbol{W}_{xh}^T x_t)$$

Input Vector

$$x_t$$

MIT 6.S191

RNNs: Computational Graph Across Time

MIT 6.S191

RNNs: Computational Graph Across Time

MIT 6.S191

RNNs for Sequence Modeling

One to One
"Vanilla" NN
Binary classification

Many to One
Sentiment Classification

One to Many
Text Generation
Image Captioning

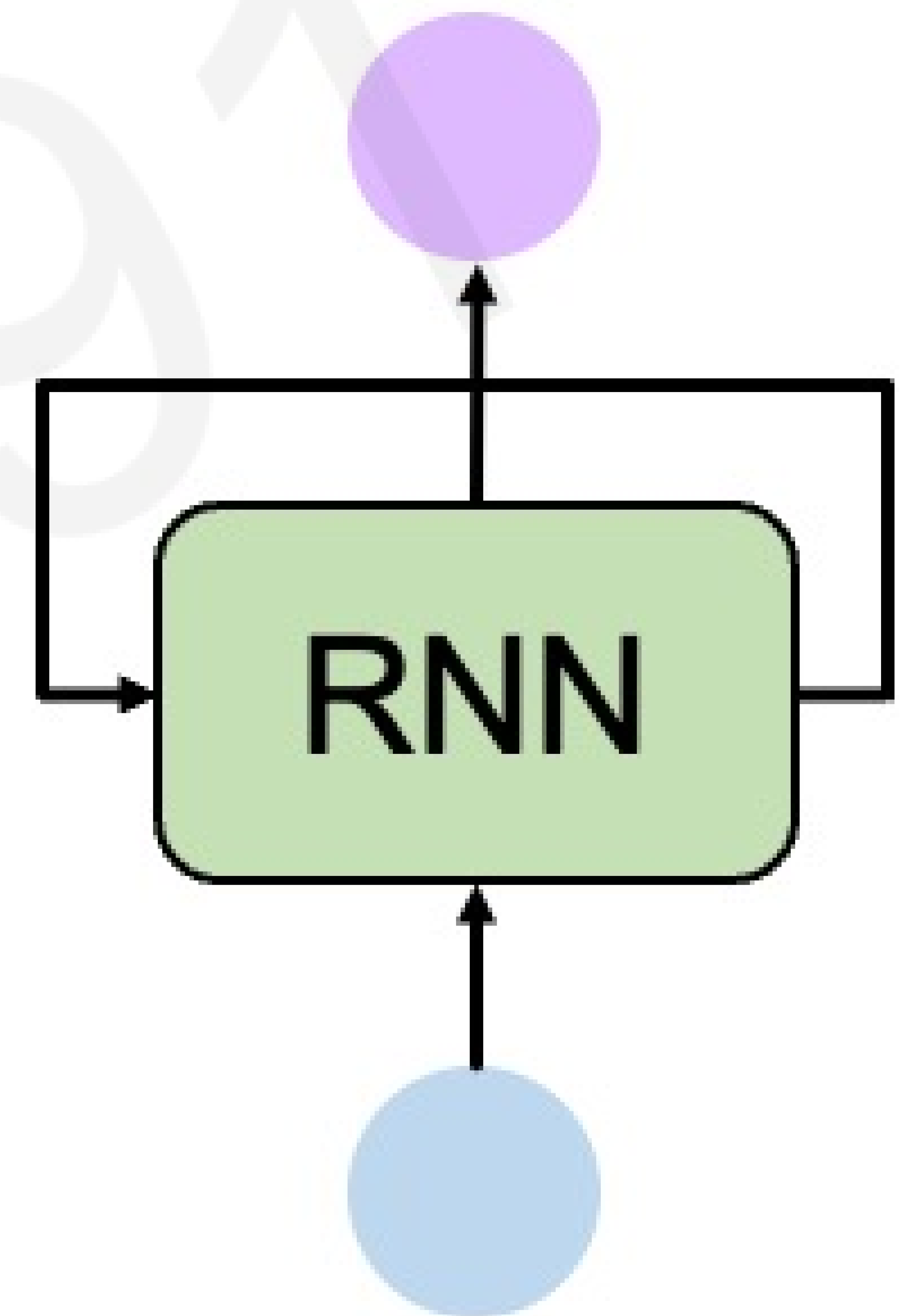
Many to Many
Translation & Forecasting
Music Generation

... and many other architectures and applications ⭐ **6.S191 Lab!**

Sequence Modeling: Design Criteria

To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

"This morning I took my cat for a walk."

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

"This morning I took my cat for a walk."

given these words

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

"This morning I took my cat for a walk."

given these words

predict the next word

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

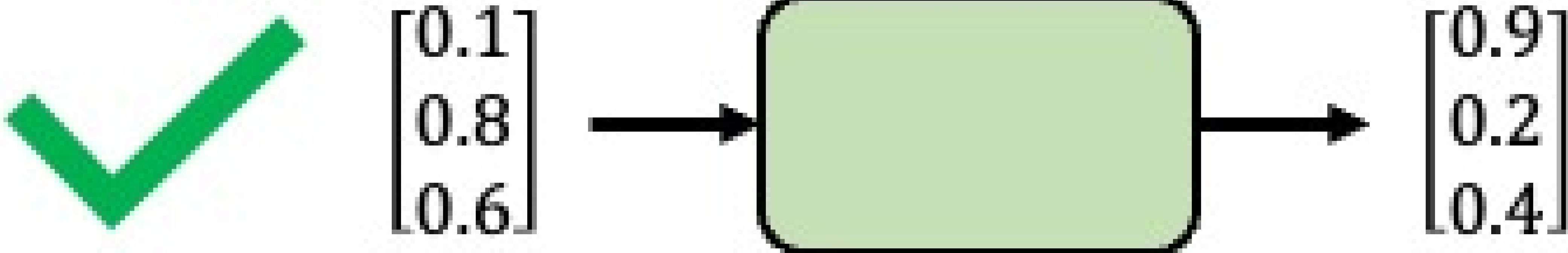
"This morning I took my cat for a walk."

given these words

predict the next word

Representing Language to a Neural Network

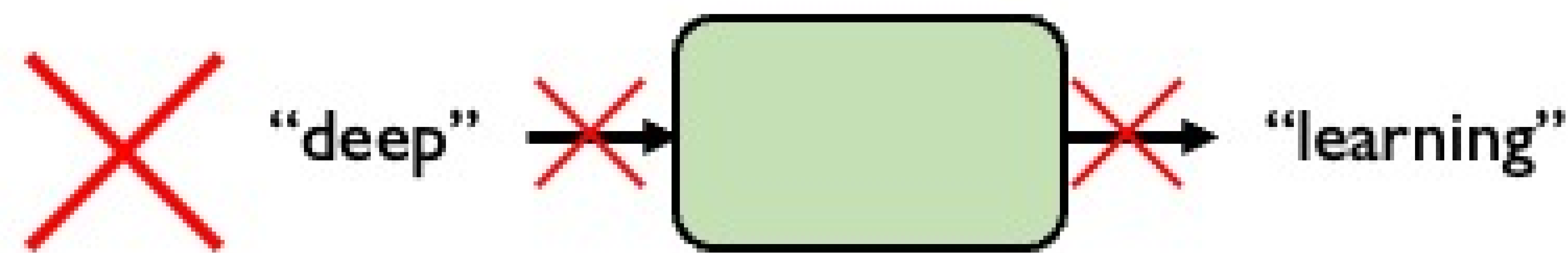
Neural networks cannot interpret words



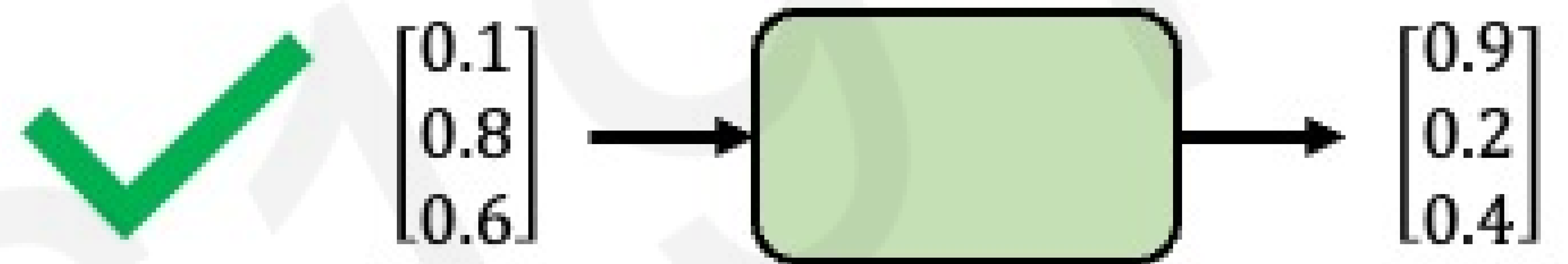
Neural networks require numerical inputs

MIT 6.S191

Encoding Language for a Neural Network

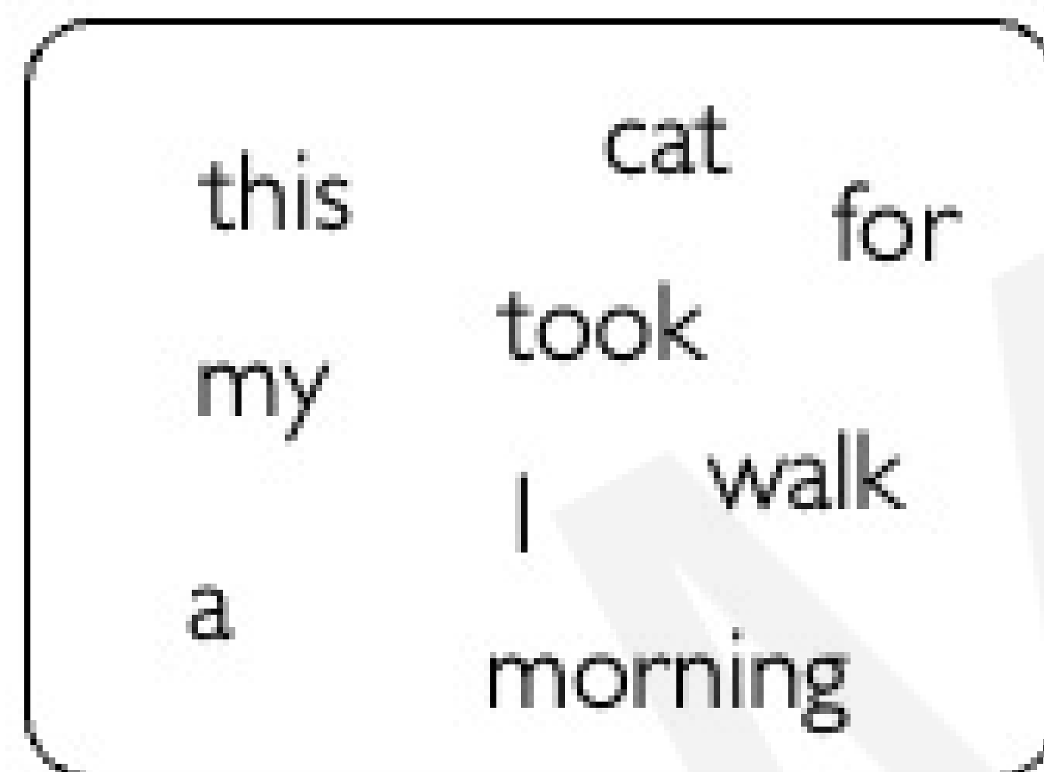


Neural networks cannot interpret words

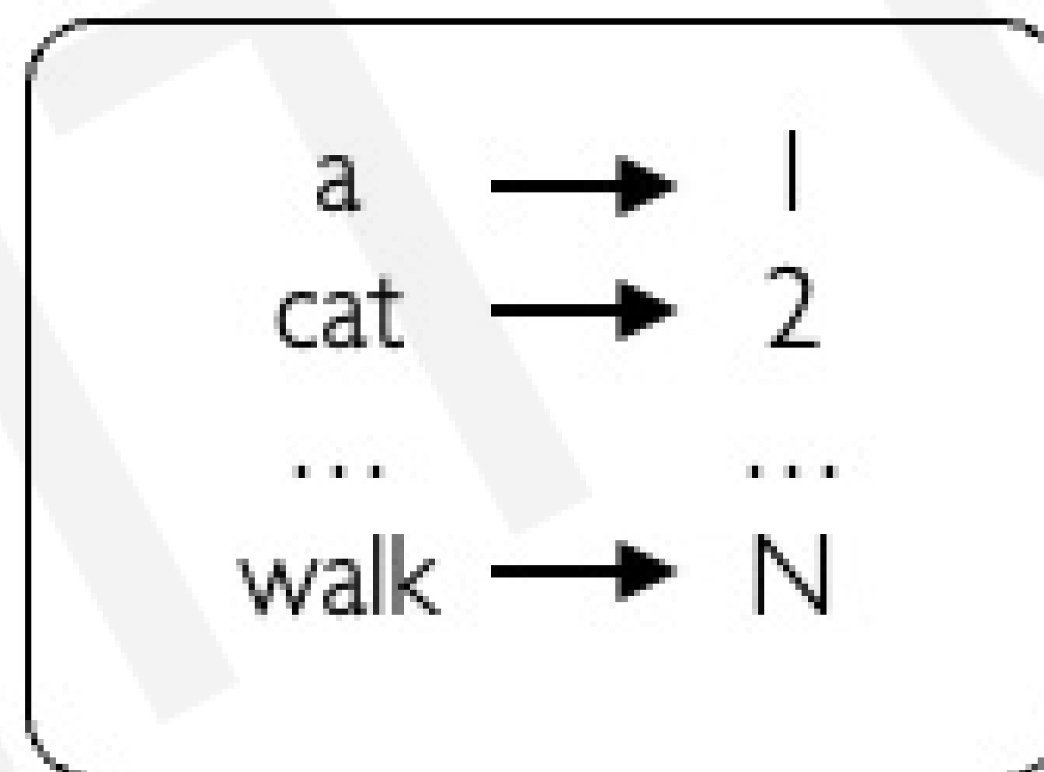


Neural networks require numerical inputs

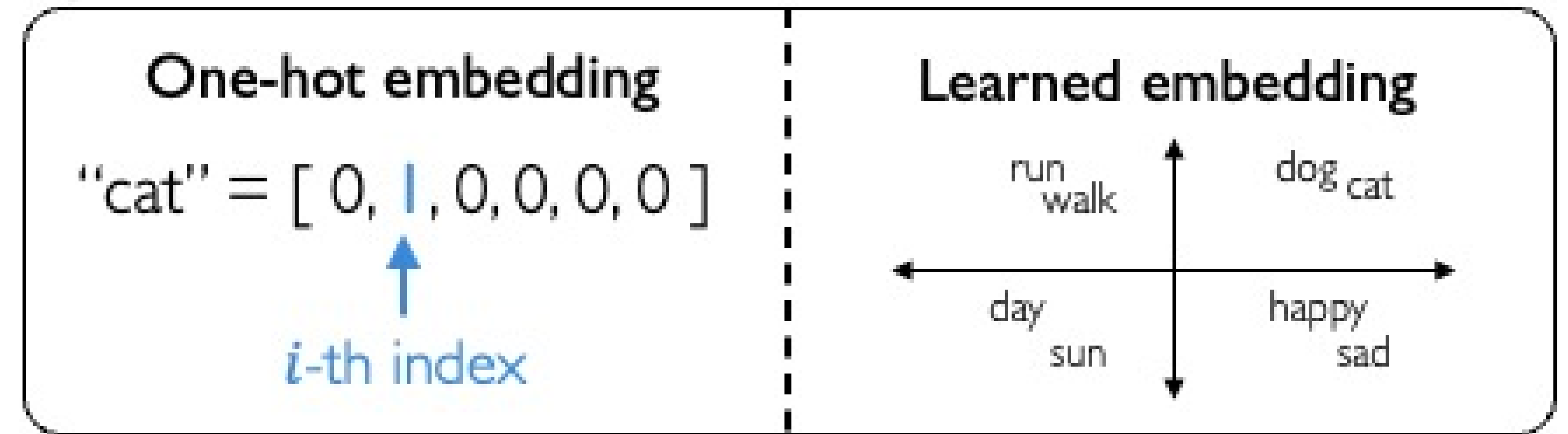
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words



2. Indexing:
Word to index



3. Embedding:
Index to fixed-sized vector

MIT 6.S191

Handle Variable Sequence Lengths

The food was great

vs.

We visited a restaurant for lunch

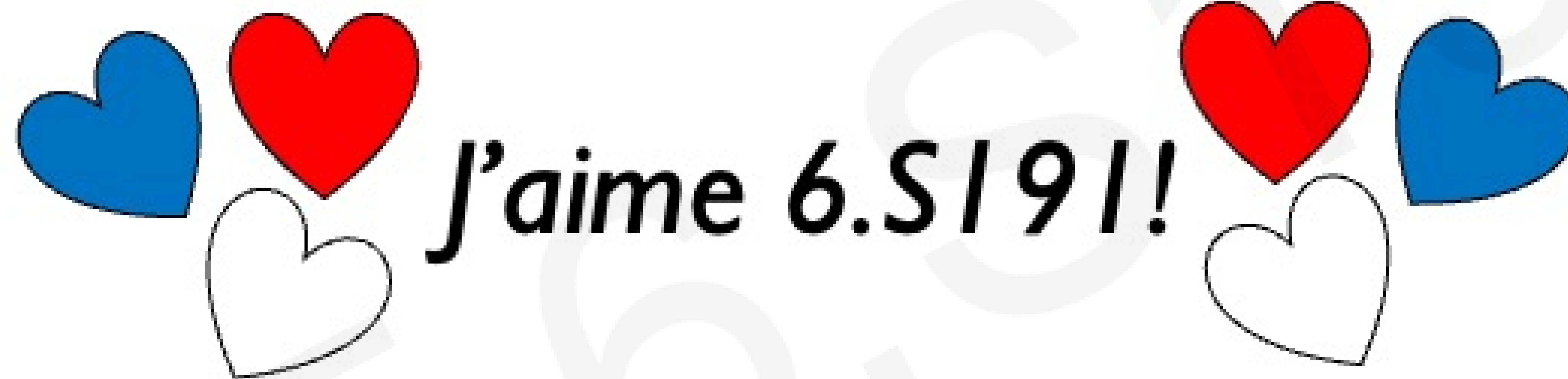
vs.

We were hungry but cleaned the house before eating

MIT 6.S191

Model Long-Term Dependencies

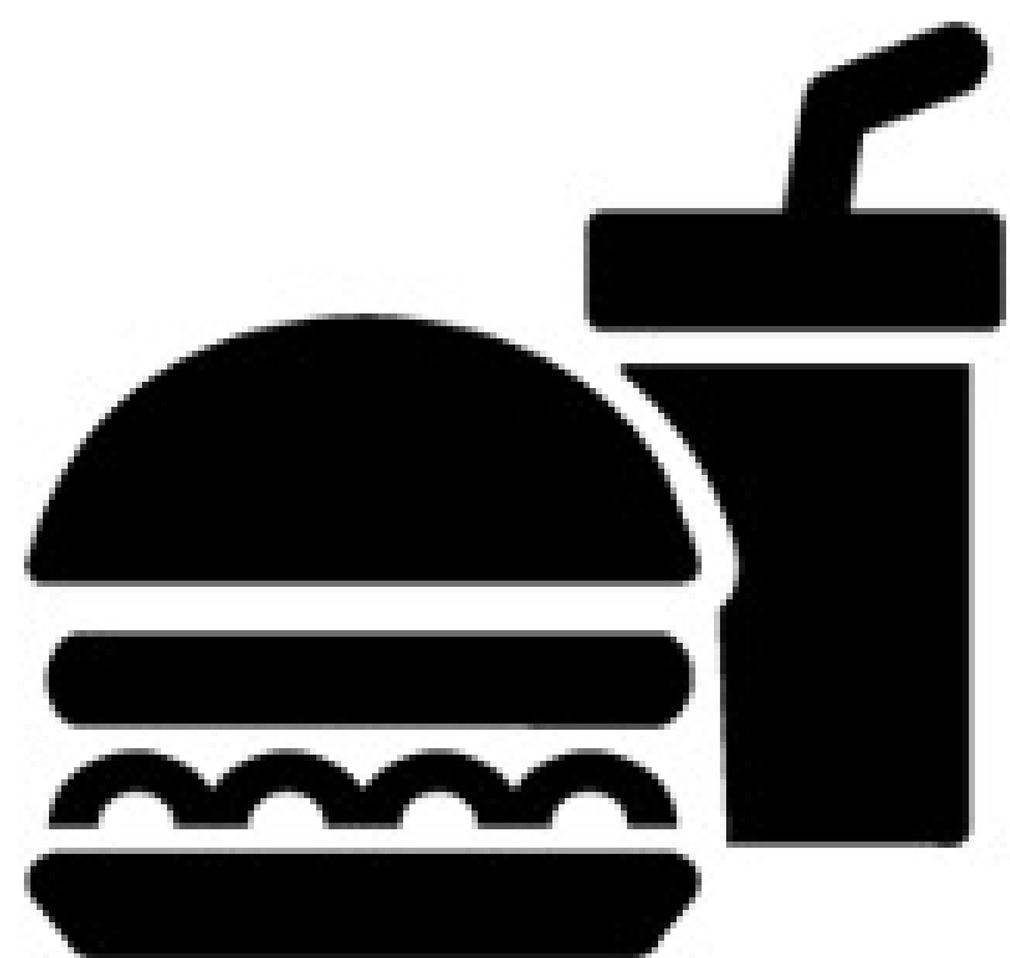
"**France** is where I grew up, but I now live in Boston. I speak fluent ___."



We need information from **the distant past** to accurately predict the correct word.

MIT 6.S191

Capture Differences in Sequence Order



The food was good, not bad at all.

vs.

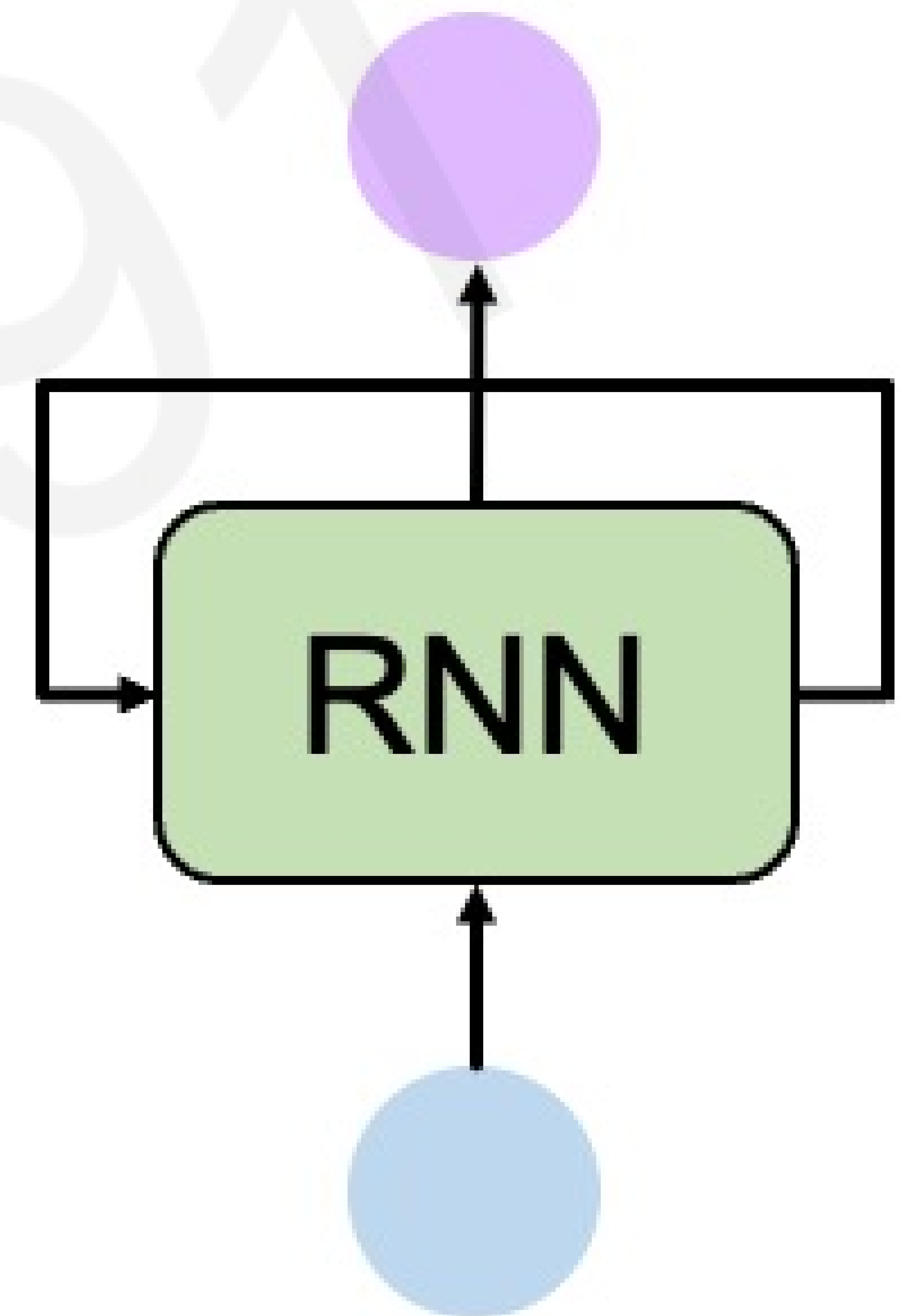
The food was bad, not good at all.

MIT 6.S191

Sequence Modeling: Design Criteria

To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



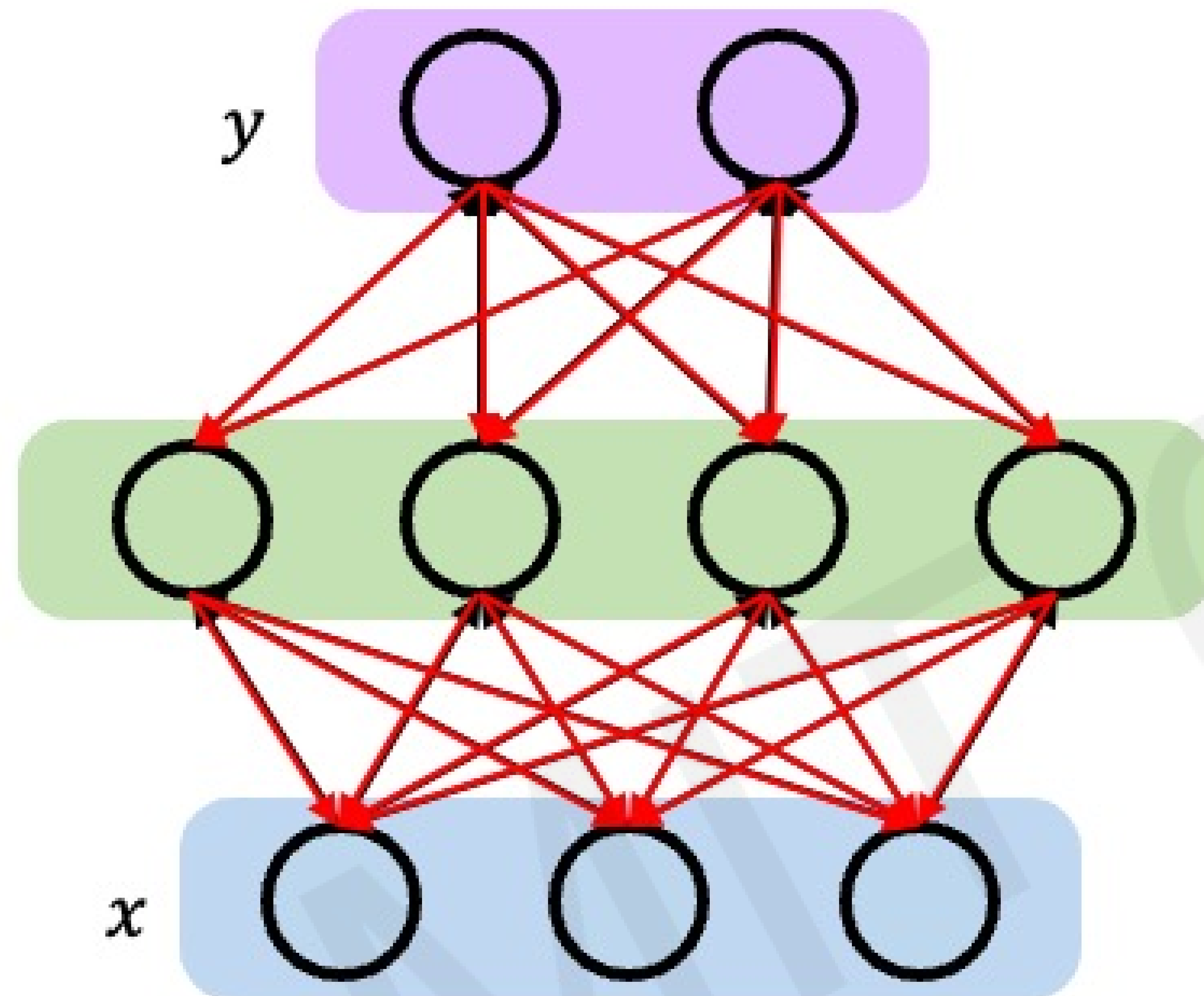
Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria

MIT 6.S191

Backpropagation Through Time (BPTT)

MIT 6.S191

Recall: Backpropagation in Feed Forward Models



Backpropagation algorithm:

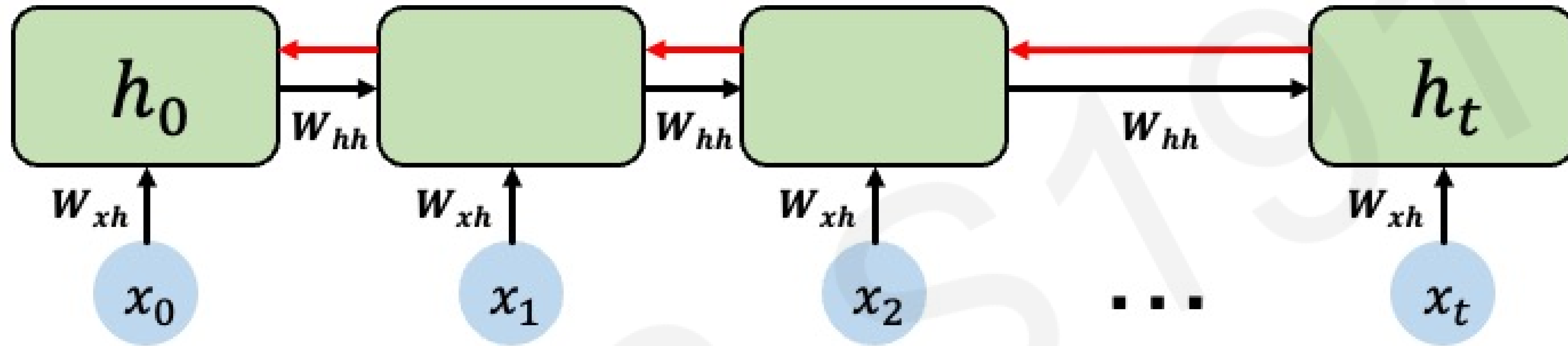
1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

MIT 6.S191

RNNs: Backpropagation Through Time

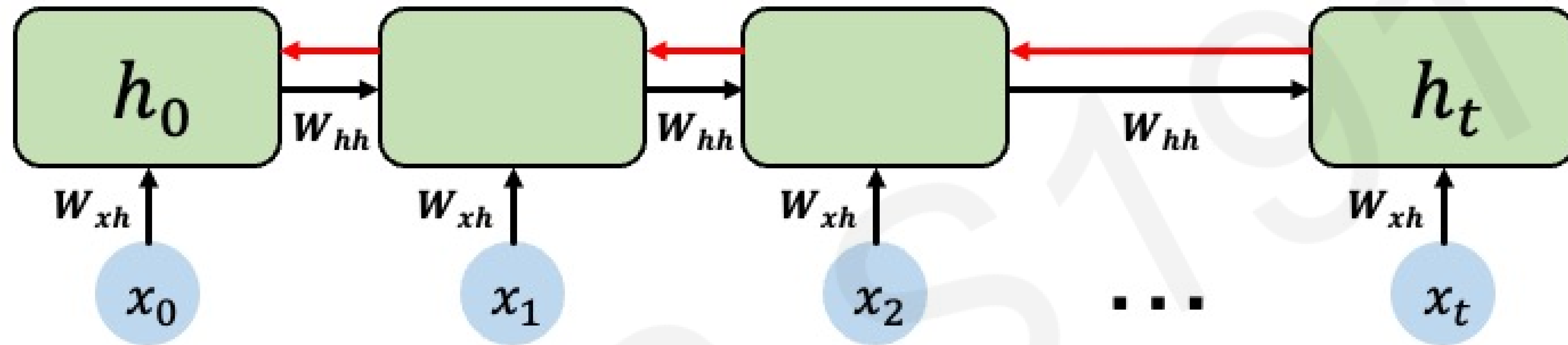
RNNs: Backpropagation Through Time

Standard RNN Gradient Flow



MIT 6.S191

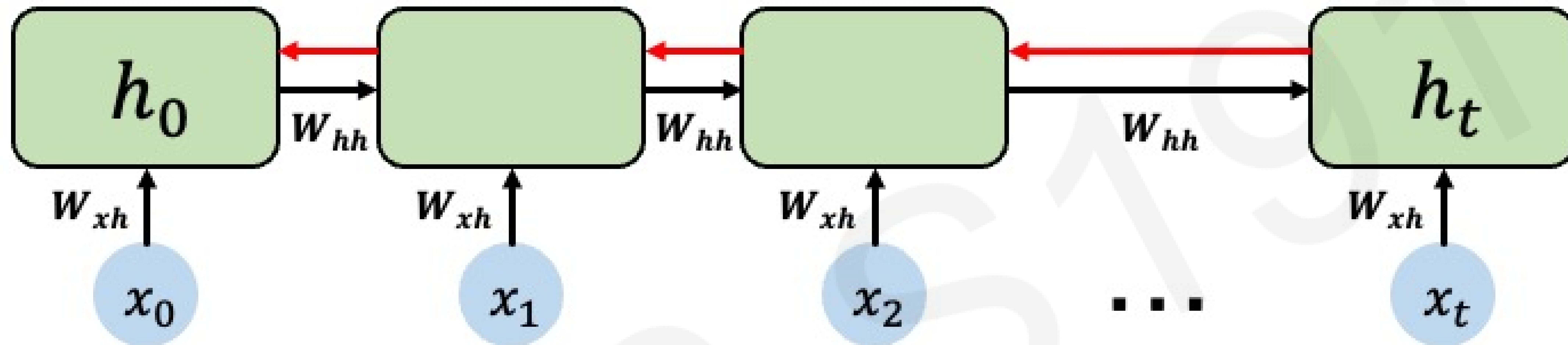
Standard RNN Gradient Flow



Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

MIT 6.S191

Standard RNN Gradient Flow: Exploding Gradients



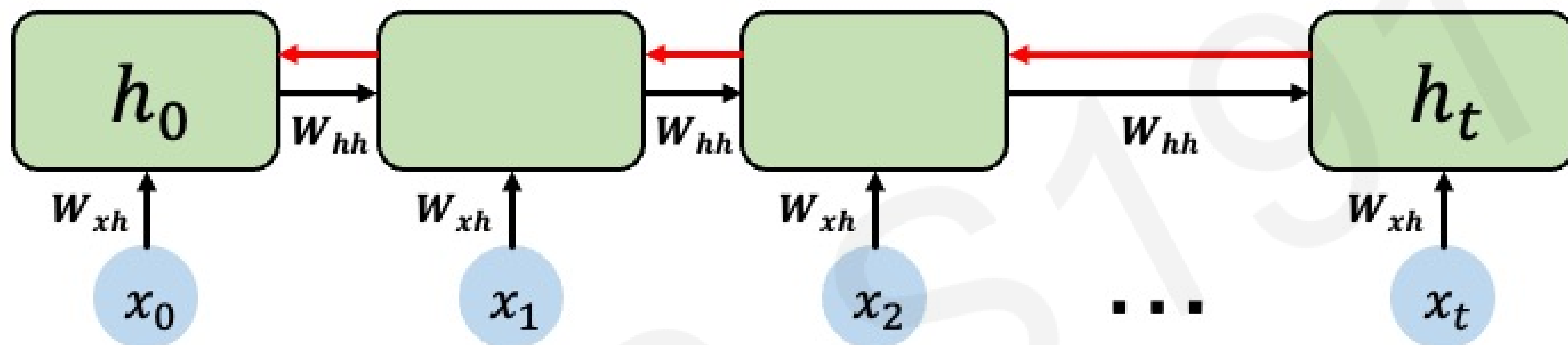
Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

MIT 6.S191

Standard RNN Gradient Flow: Vanishing Gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

Many values < 1:
vanishing gradients

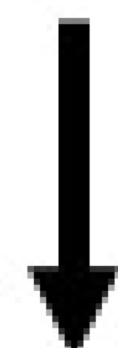
1. Activation function
2. Weight initialization
3. Network architecture

MIT 6.S191

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together



Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies

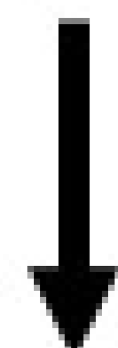
MIT 6.S191

The Problem of Long-Term Dependencies

"The clouds are in the ___"

Why are vanishing gradients a problem?

Multiply many **small numbers** together



Errors due to further back time steps have smaller and smaller gradients

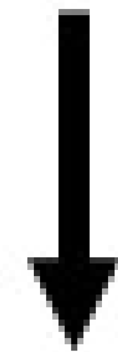
Bias parameters to capture short-term dependencies

MIT 6.S191

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

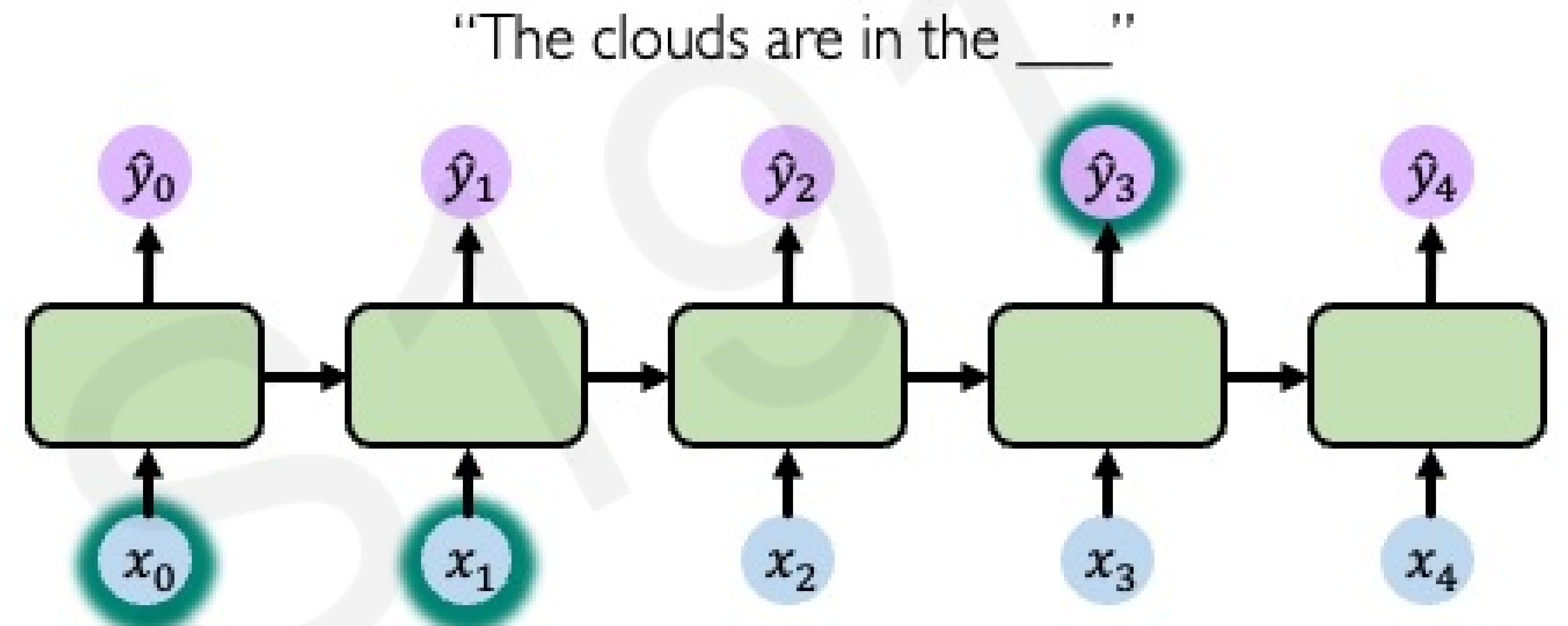
Multiply many **small numbers** together



Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies

"The clouds are in the ___"

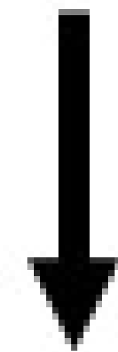


MIT 6.S191

The Problem of Long-Term Dependencies

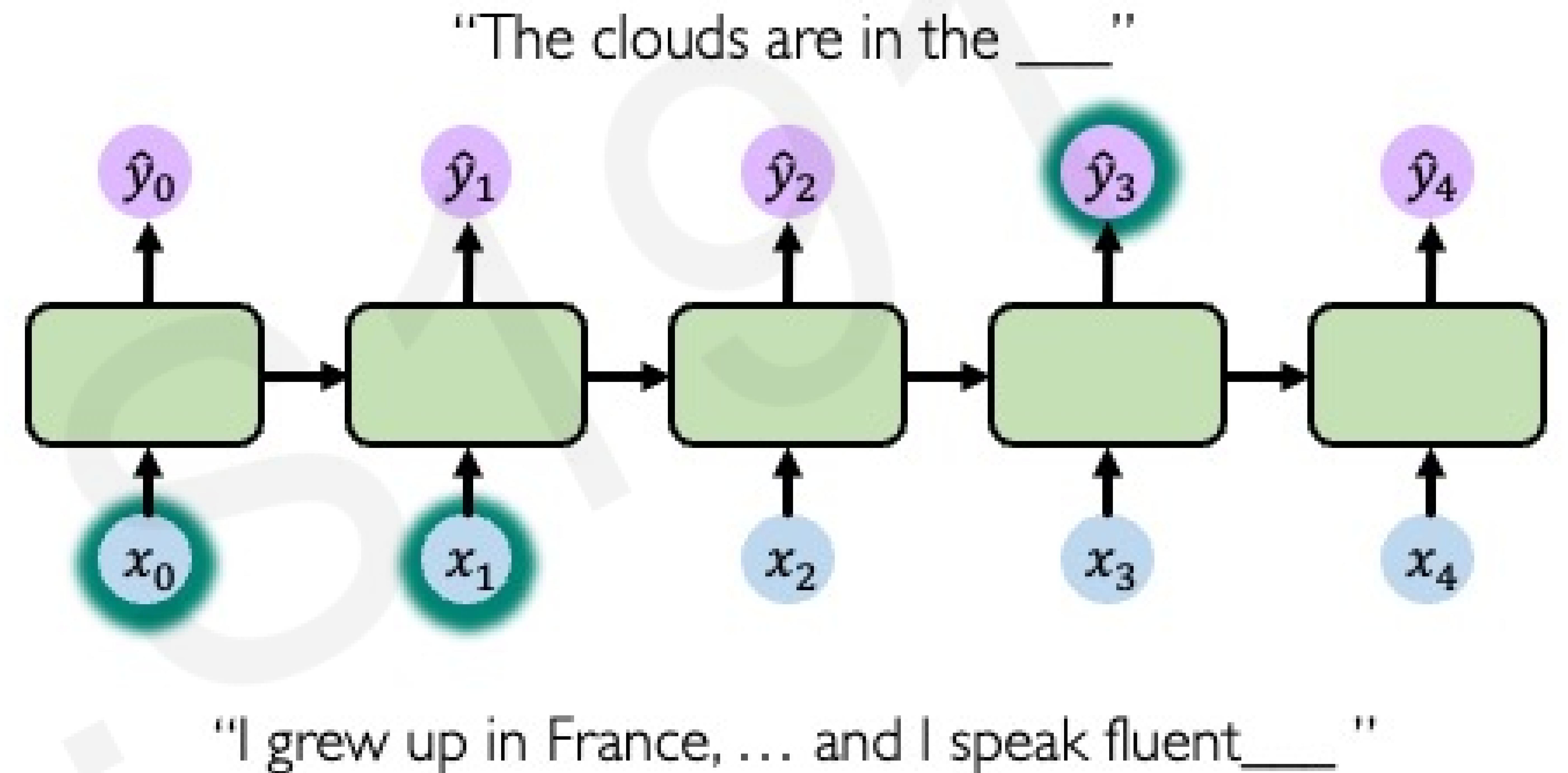
Why are vanishing gradients a problem?

Multiply many **small numbers** together



Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies

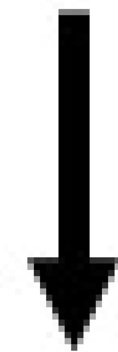


MIT 6.S191

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

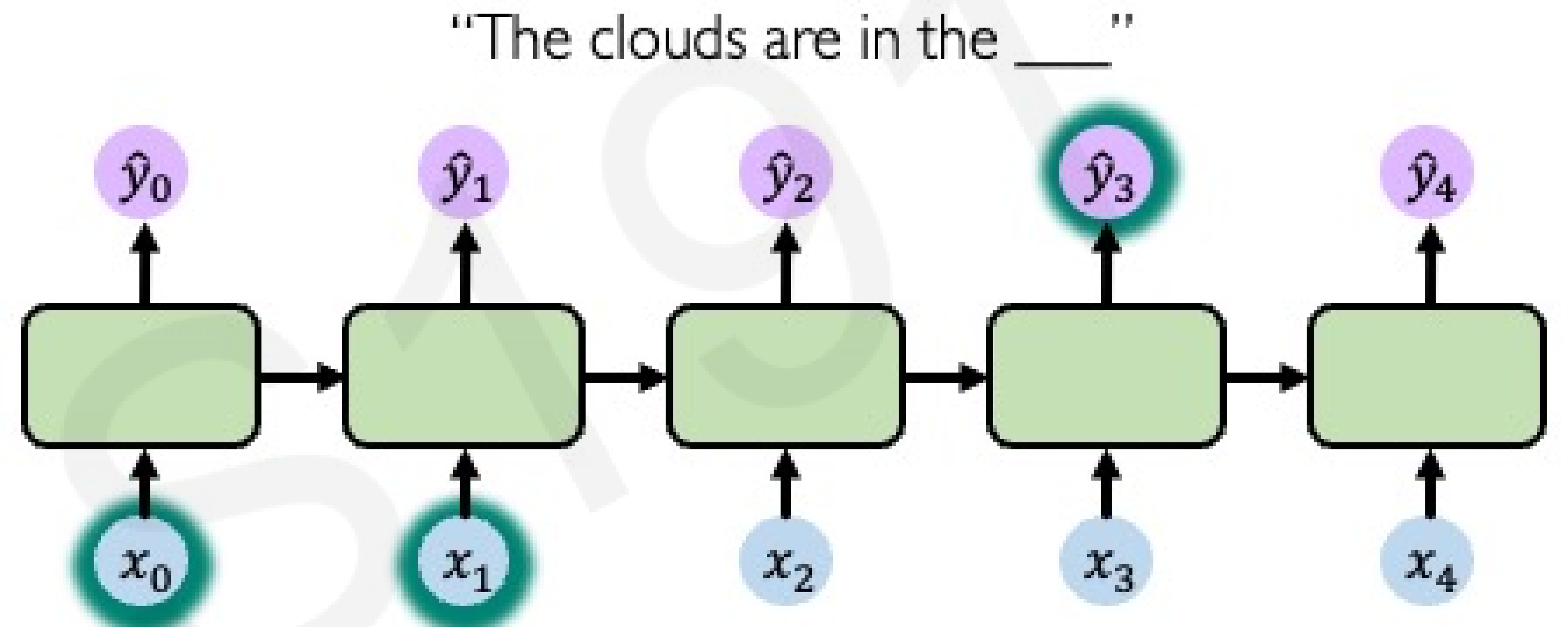
Multiply many **small numbers** together



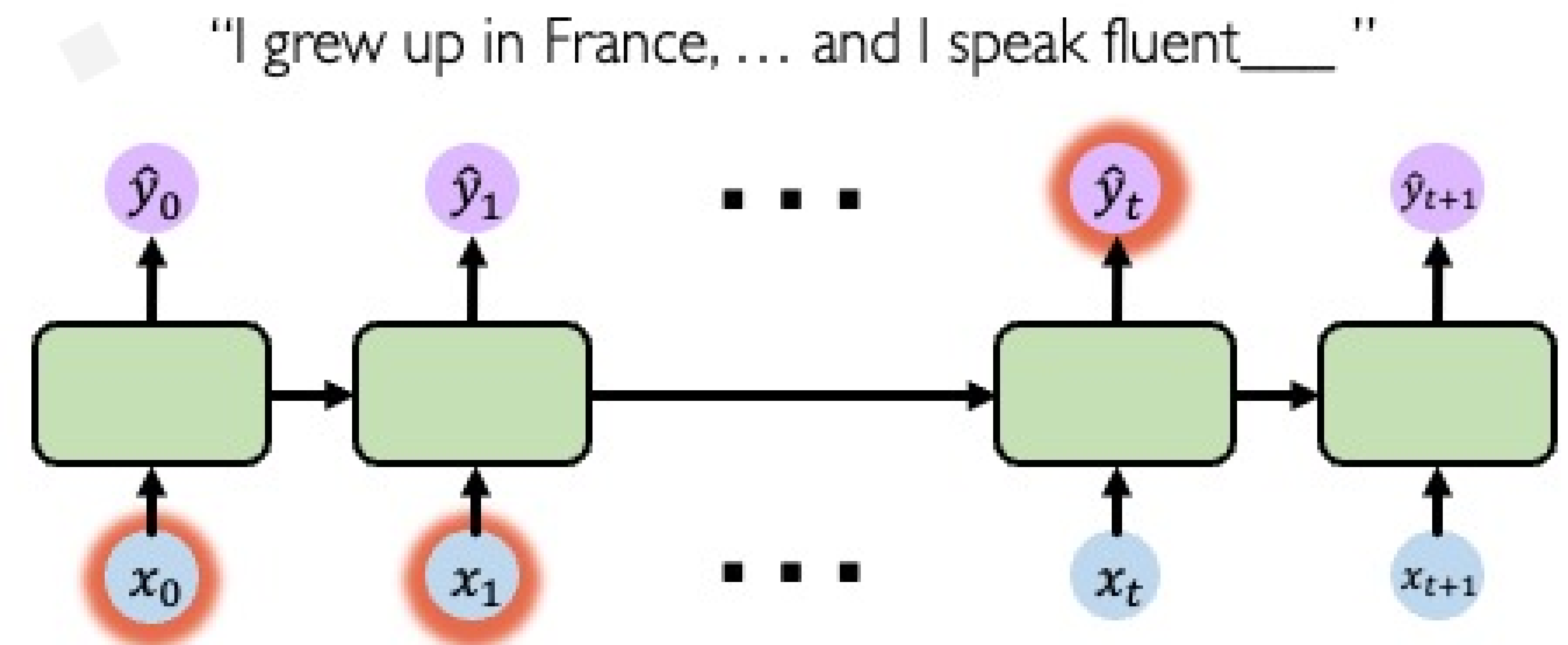
Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies

"The clouds are in the ___"

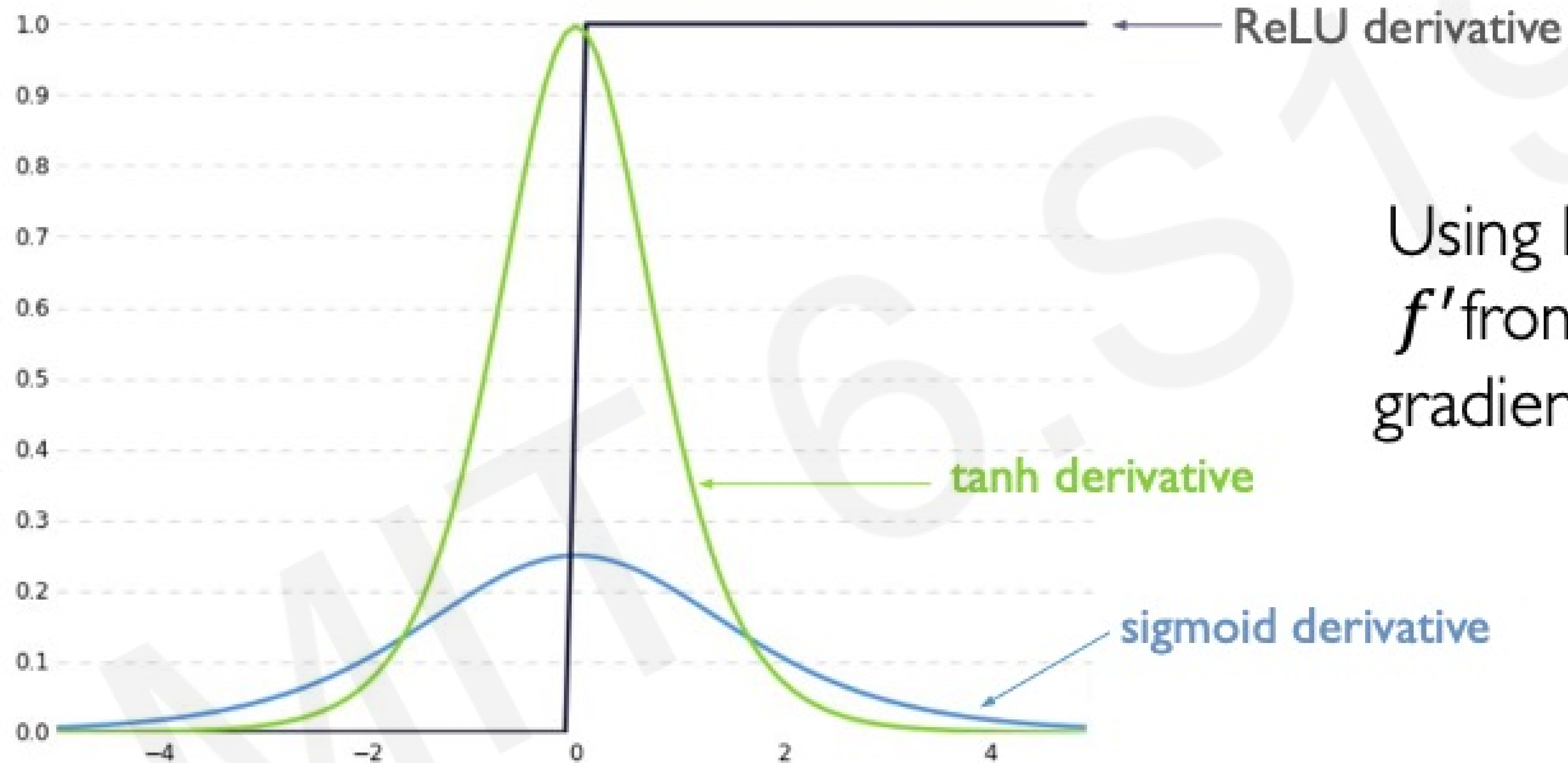


"I grew up in France, … and I speak fluent___ "



MIT 6.S191

Trick #1: Activation Functions



Using ReLU prevents f' from shrinking the gradients when $x > 0$

MIT 6.S191

Trick #2: Parameter Initialization

Initialize **weights** to identity matrix

Initialize **biases** to zero

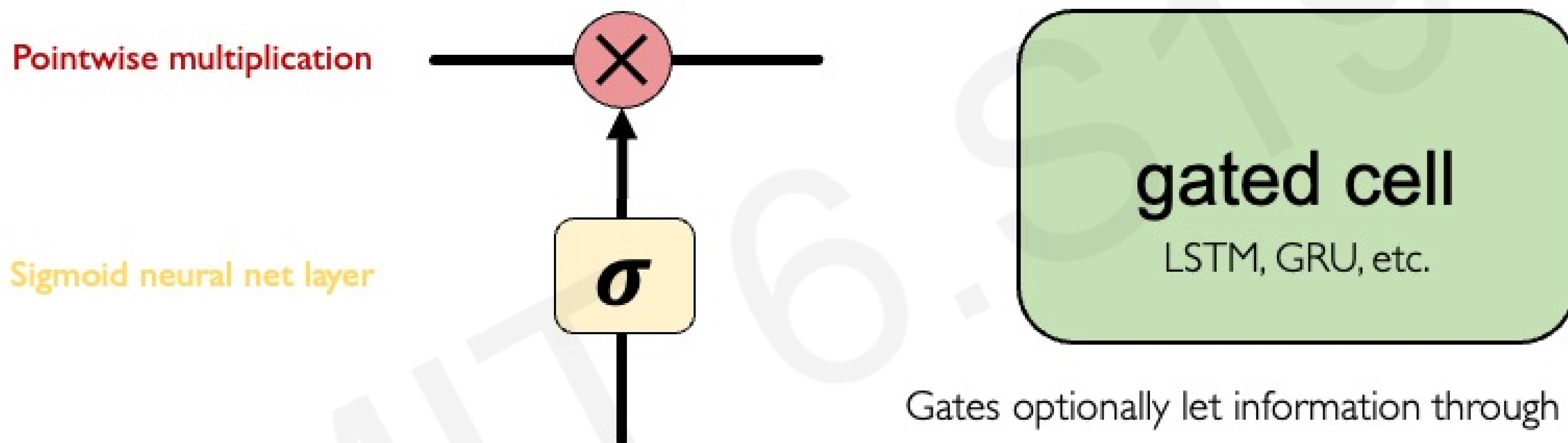
$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

MIT 6.S191

Trick #3: Gated Cells

Idea: use **gates** to selectively **add** or **remove** information within **each recurrent unit with**



Gates optionally let information through the cell

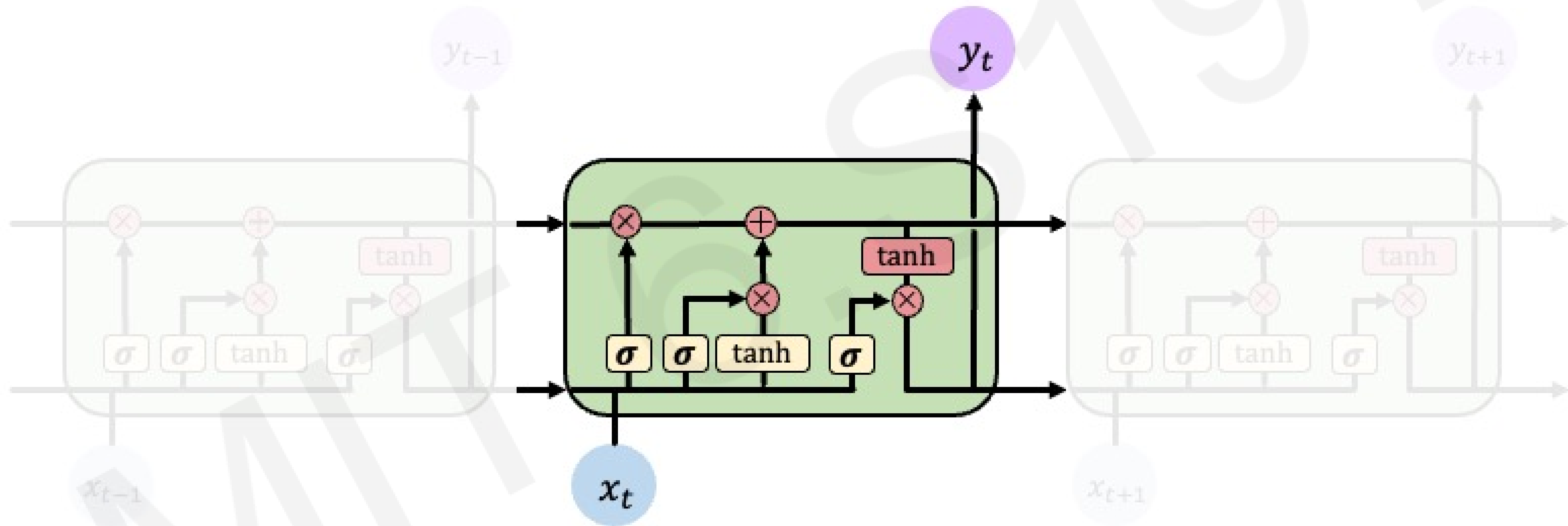
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

MIT 6.S191

Long Short Term Memory (LSTMs)

Gated LSTM cells **control information flow:**

1) Forget 2) Store 3) Update 4) Output



LSTM cells are able to track information throughout many timesteps

```
tf.keras.layers.LSTM(num_units)
```

MIT 6.S191

LSTMs: Key Concepts

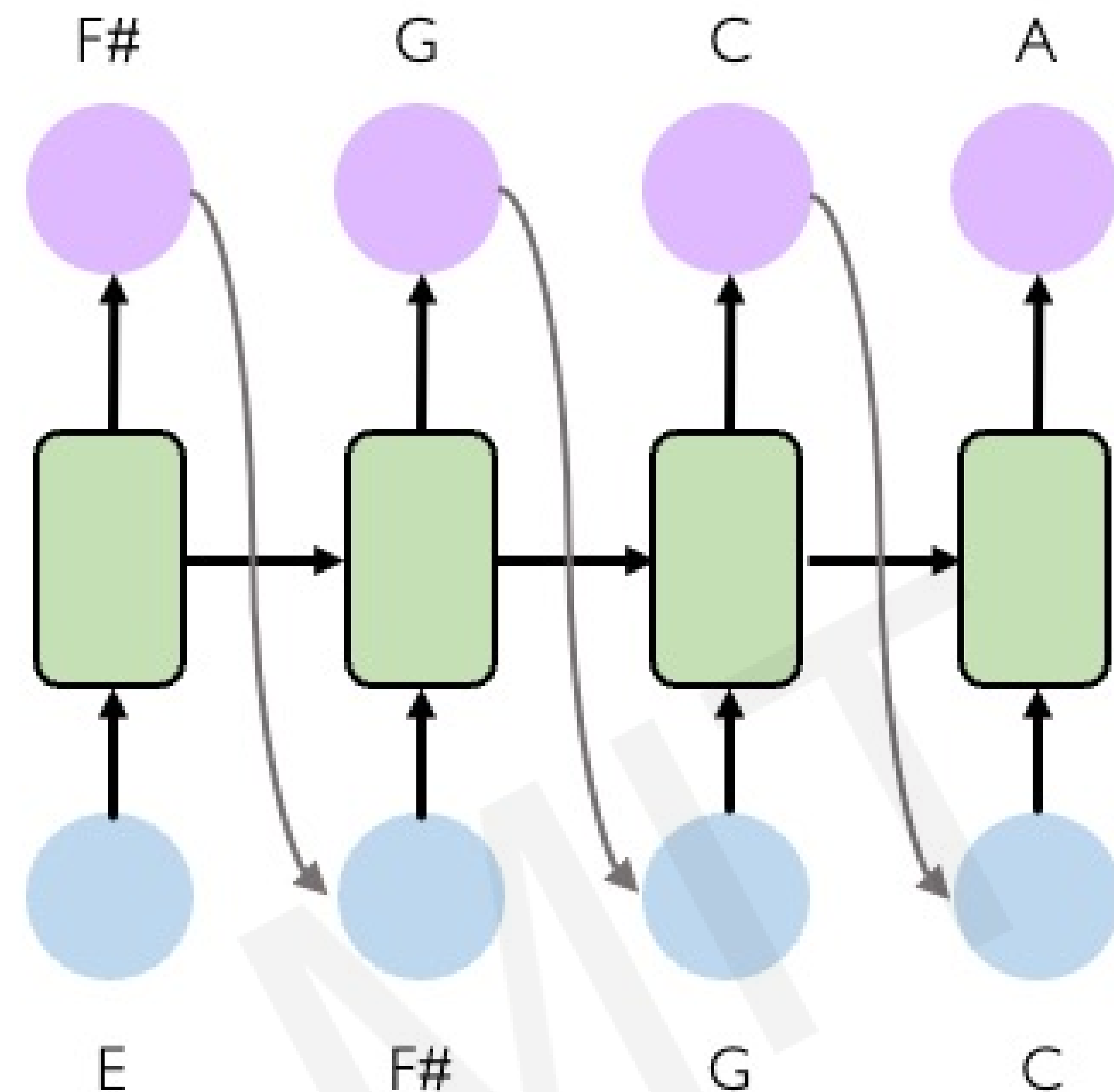
1. Maintain a **cell state**
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with partially **uninterrupted gradient flow**

MIT 6.S191

RNN Applications & Limitations

MIT 6.S191

Example Task: Music Generation



Input: sheet music

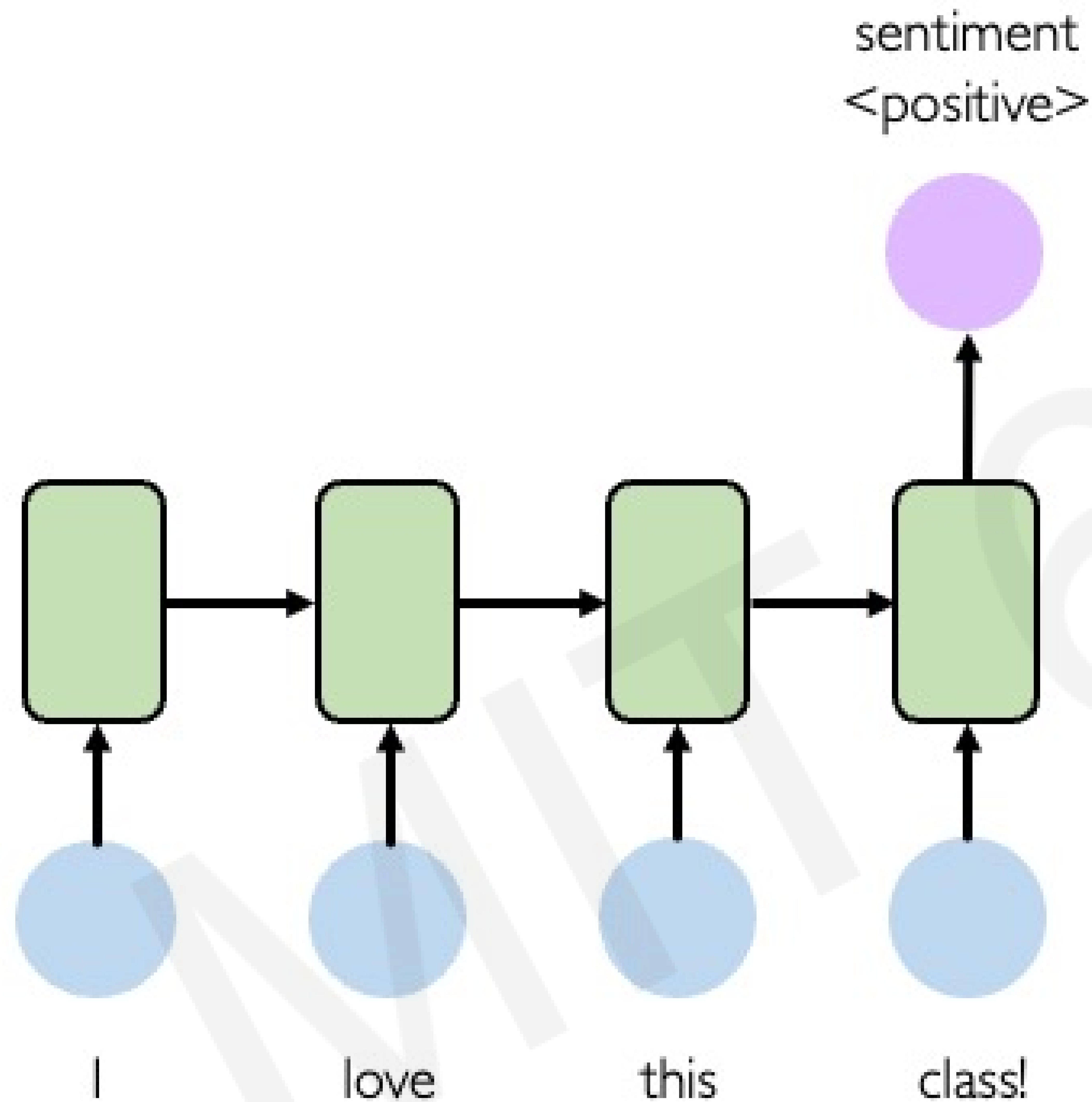
Output: next character in sheet music

Listening to 3rd movement

6.S191 Lab!

MIT 6.S191

Example Task: Sentiment Classification



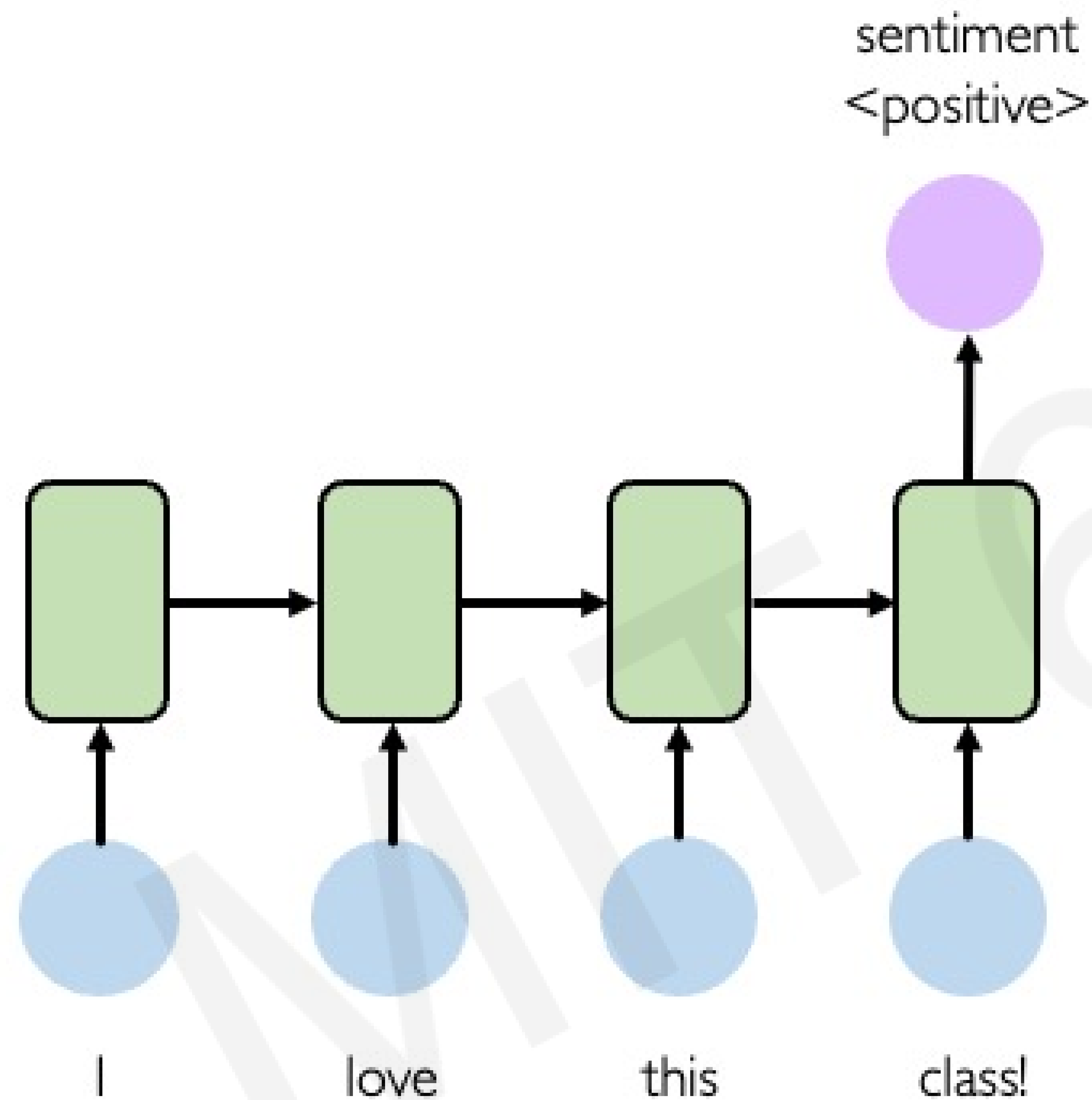
Input: sequence of words

Output: probability of having positive sentiment

```
loss = tf.nn.softmax_cross_entropy_with_logits(y, predicted)
```

MIT 6.S191

Example Task: Sentiment Classification



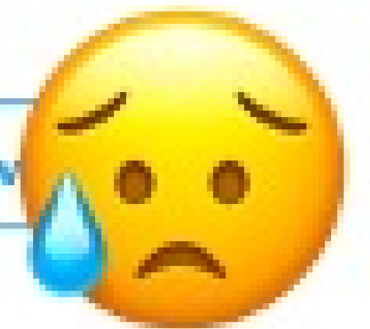
Tweet sentiment classification

Follow

The @MIT Introduction to #DeepLearning is definitely one of the best courses of its kind currently available online introtodeeplearning.com

12:45 PM - 12 Feb 2018

Follow



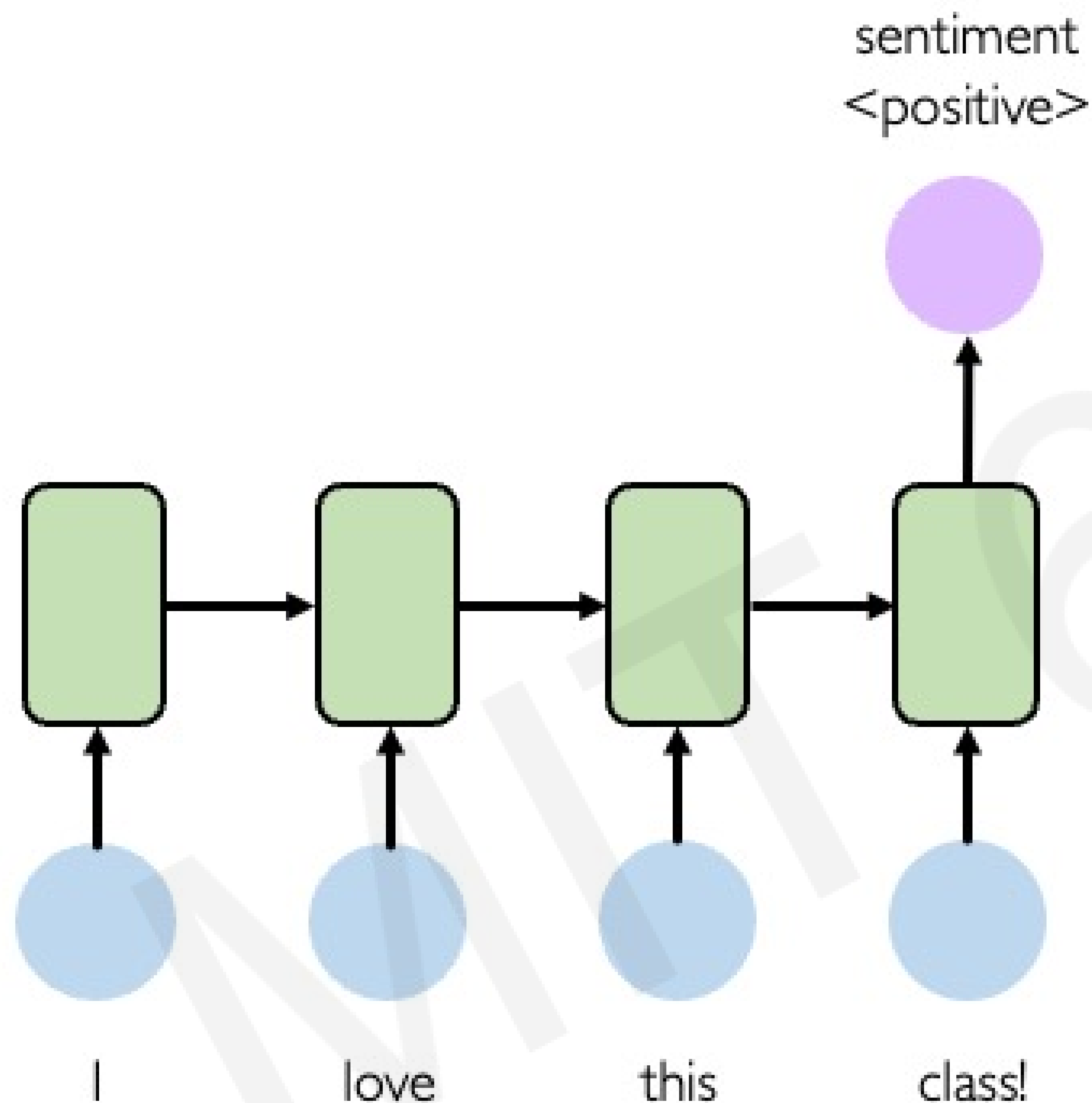
Replying to @Kazuki2048

I wouldn't mind a bit of snow right now. We haven't had any in my bit of the Midlands this winter! :(

2:19 AM - 25 Jan 2019

MIT 6.S191

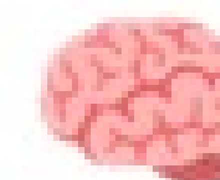
Limitations of Recurrent Models



Limitations of RNNs

Encoding bottleneck

Slow, no parallelization

 Not long memory

MIT 6.S191

Goal of Sequence Modeling

MIT 6.S191

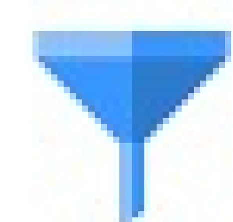
Goal of Sequence Modeling

RNNs: recurrence to model sequence dependencies

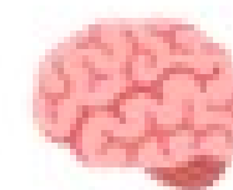
$\hat{y}_0$	$\hat{y}_1$	$\hat{y}_2$		$\hat{y}_{t-2}$	$\hat{y}_{t-1}$	$\hat{y}_t$	output
↑	↑	↑		↑	↑	↑	
▯ →	▯ →	▯	. . .	▯ →	▯ →	▯	feature vector
↑	↑	↑		↑	↑	↑	
x_0	x_1	x_2		x_{t-2}	x_{t-1}	x_t	input

t →

Limitations of RNNs


Encoding bottleneck

Slow, no parallelization


Not long memory

MIT 6.S191

Goal of Sequence Modeling

Can we eliminate the need for recurrence entirely?

Desired Capabilities

Continuous stream

Parallelization

Long memory

output: $\hat{y}_0$, $\hat{y}_1$, $\hat{y}_2$, ..., $\hat{y}_{t-2}$, $\hat{y}_{t-1}$, $\hat{y}_t$

feature vector

input: x_0, x_1, x_2, ..., x_{t-2}, x_{t-1}, x_t

t

MIT 6.S191

Goal of Sequence Modeling

Can we eliminate the need for recurrence entirely?

Desired Capabilities

$\hat{y}_0$	$\hat{y}_1$	$\hat{y}_2$		$\hat{y}_{t-2}$	$\hat{y}_{t-1}$	$\hat{y}_t$

output

feature vector

x_0	x_1	x_2		x_{t-2}	x_{t-1}	x_t

input

t

MIT 6.S191

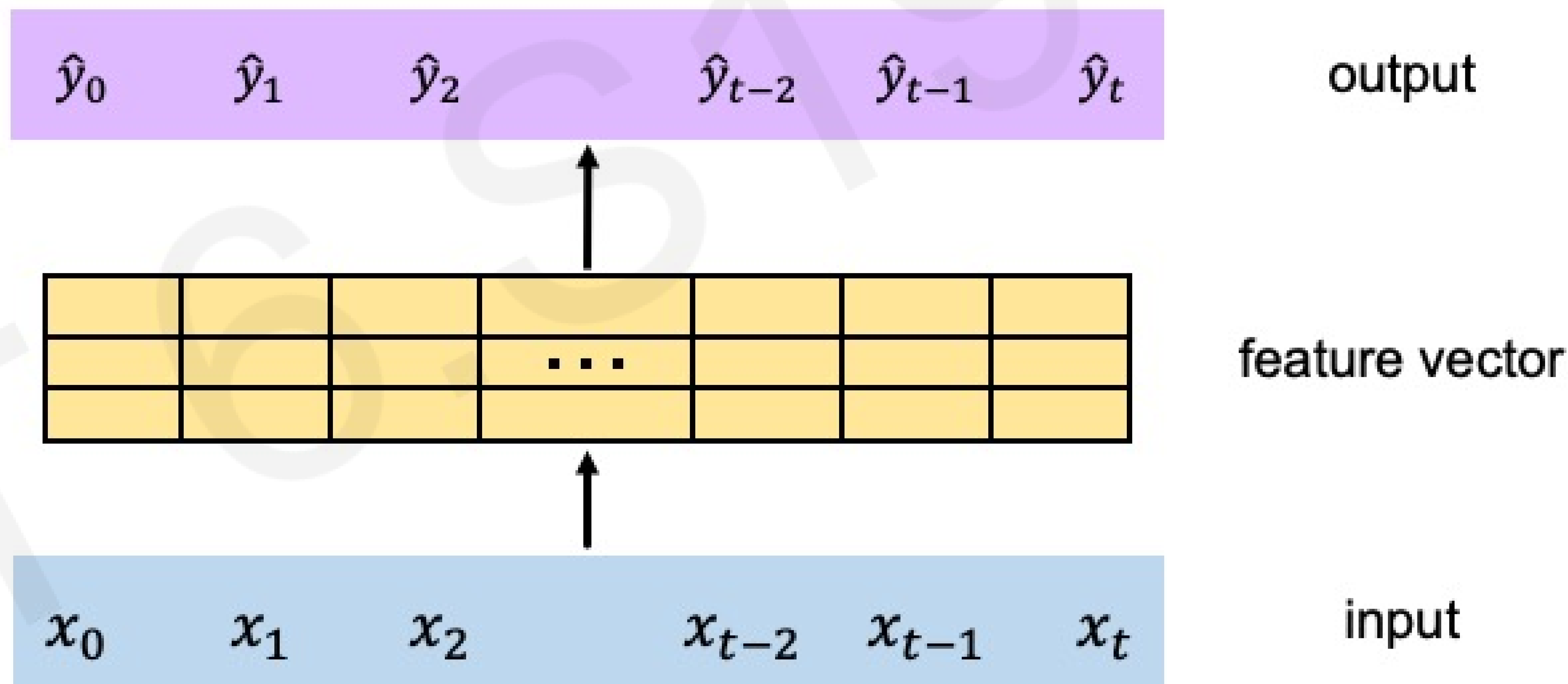
Goal of Sequence Modeling

Idea 1: Feed everything into dense network

- ✓ No recurrence
- ✗ Not scalable
- ✗ No order
- ✗ No long memory

Idea: Identify and attend to what's important

Can we eliminate the need for recurrence entirely?



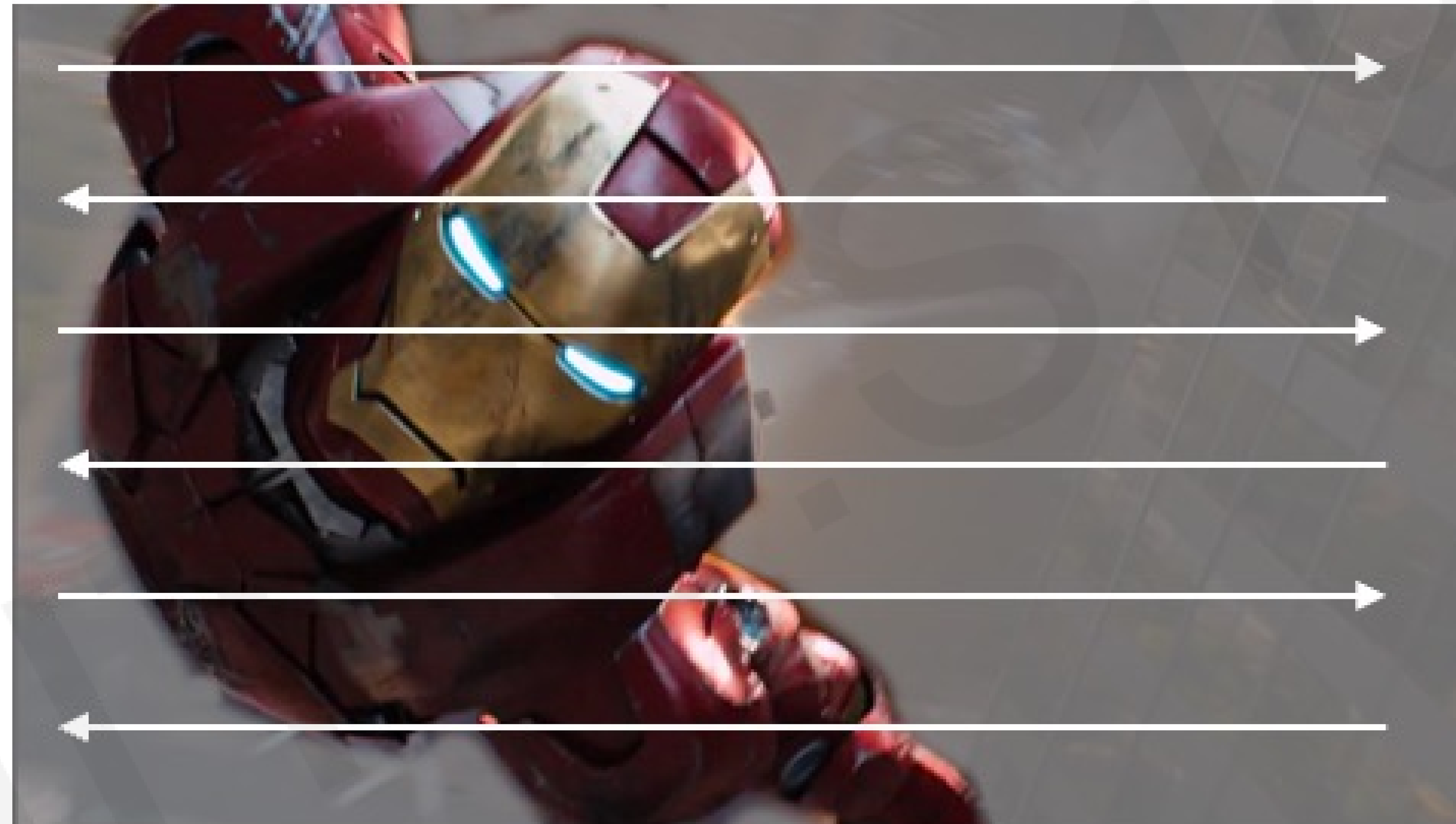
MIT 6.S191

Attention Is All You Need

MIT 6.S191

Intuition Behind Self-Attention

Attending to the most important parts of an input.



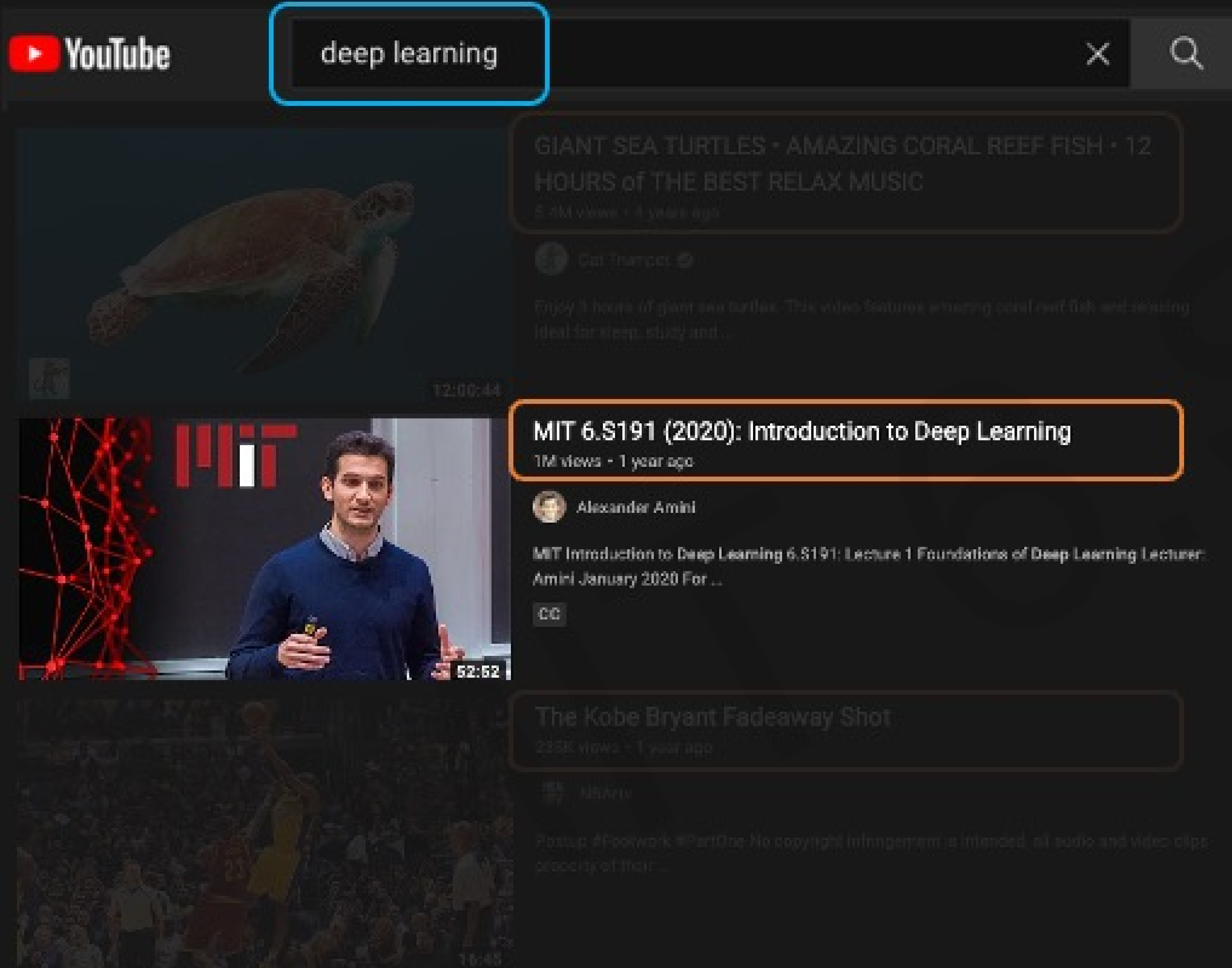
1. Identify which parts to attend to
2. Extract the features with high attention

Similar to a search problem!

A Simple Example: Search

MIT

Understanding Attention with Search



Query (Q)

Key (K_1)

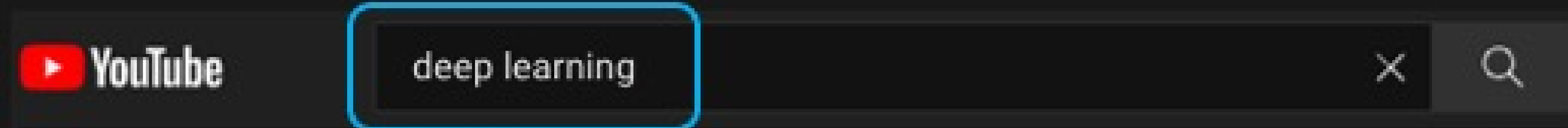
Key (K_2)

Key (K_3)

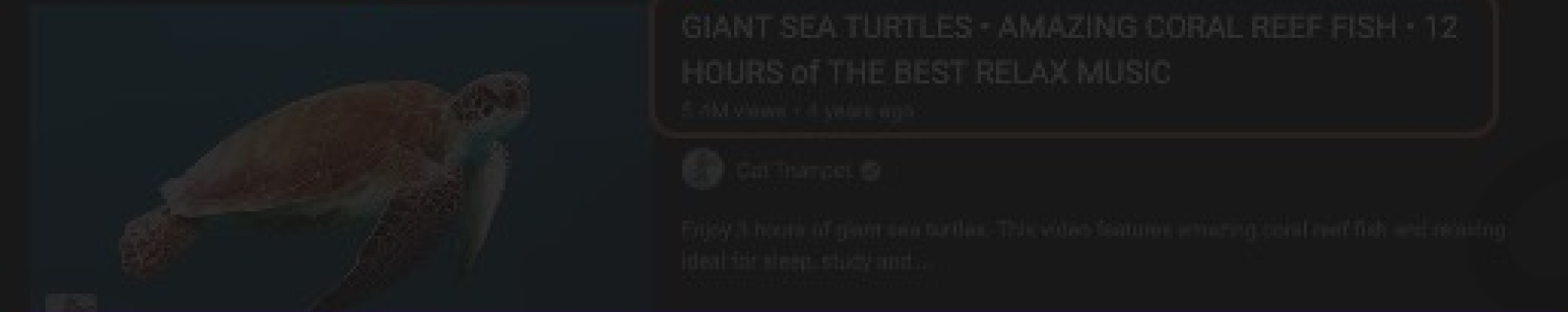
How similar is the key to the query?

1. **Compute attention mask:** how similar is each key to the desired query?

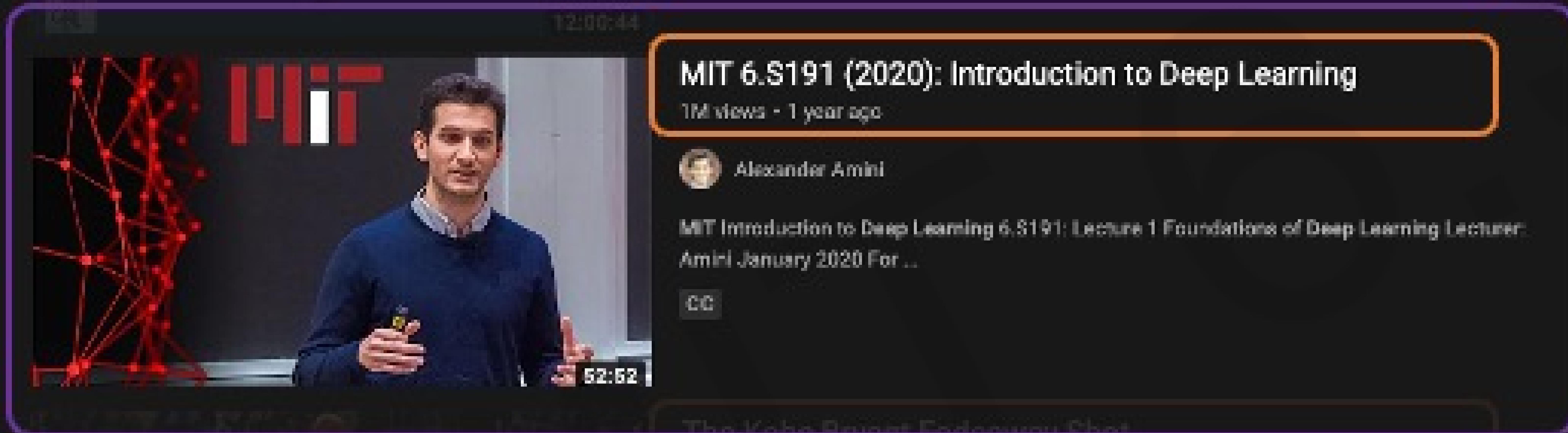
Understanding Attention with Search



Query (Q)

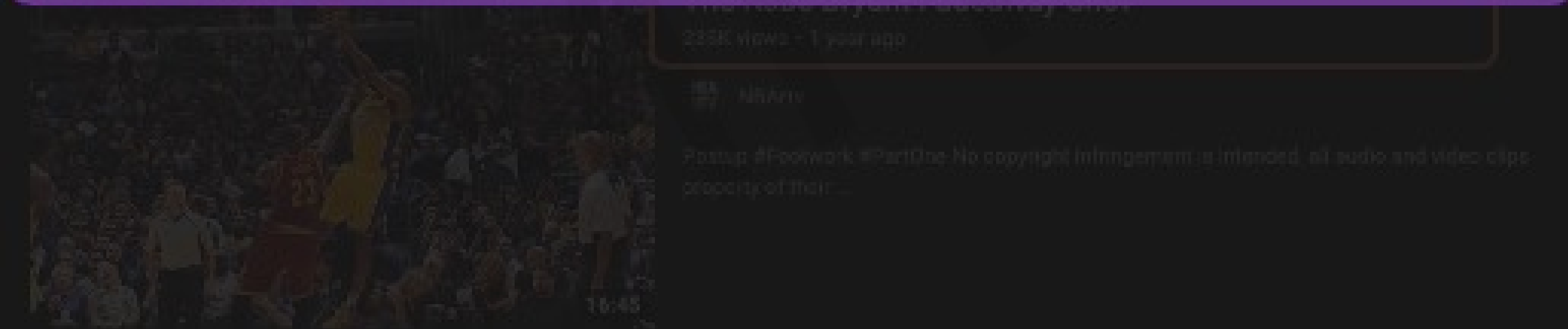


Key (K_1)



Key (K_2)

Value (V)



Key (K_3)

2. **Extract values based on attention:**
Return the values highest attention

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

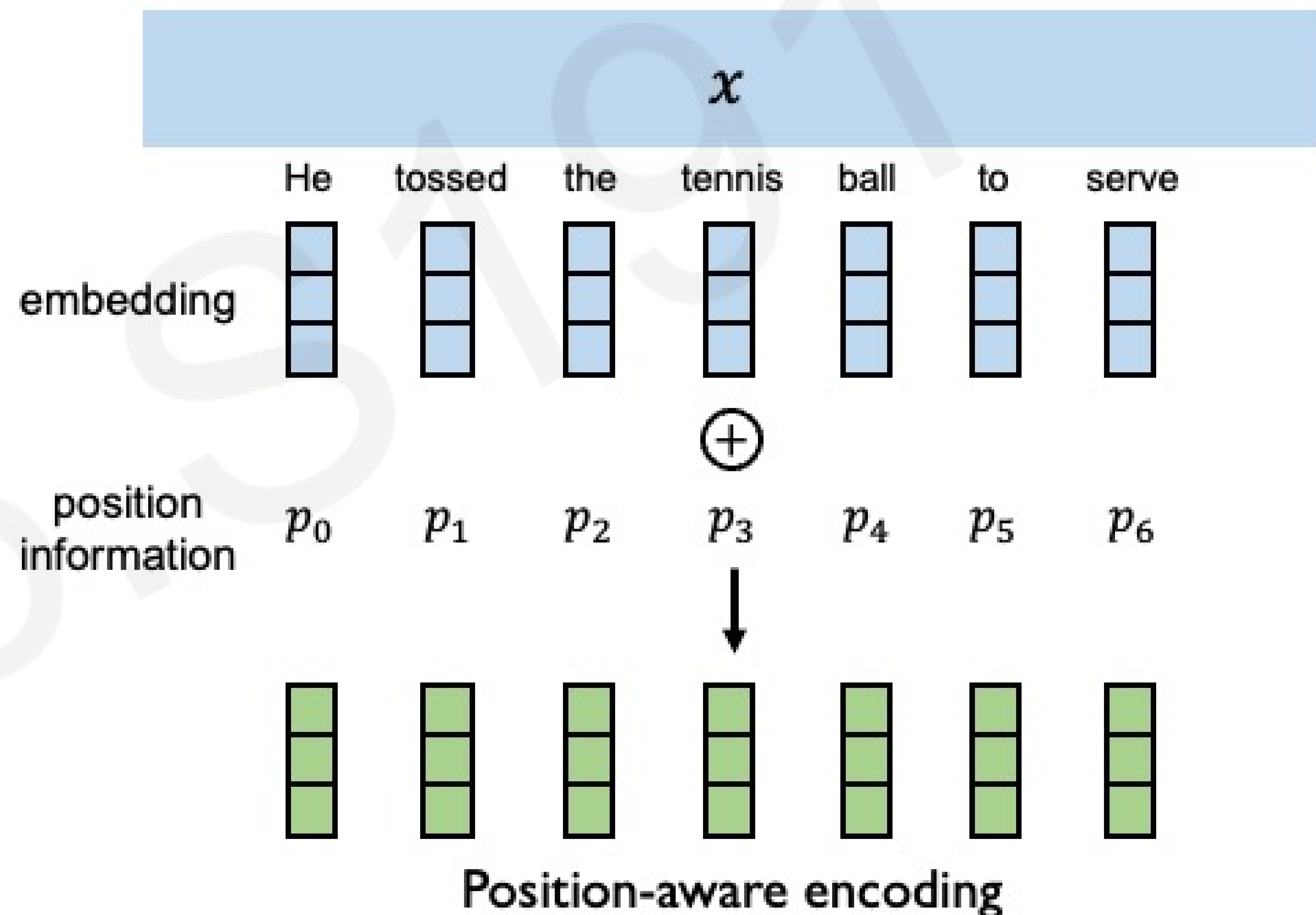
Data is fed in all at once! Need to encode position information to understand order.

MIT 6.S191

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**



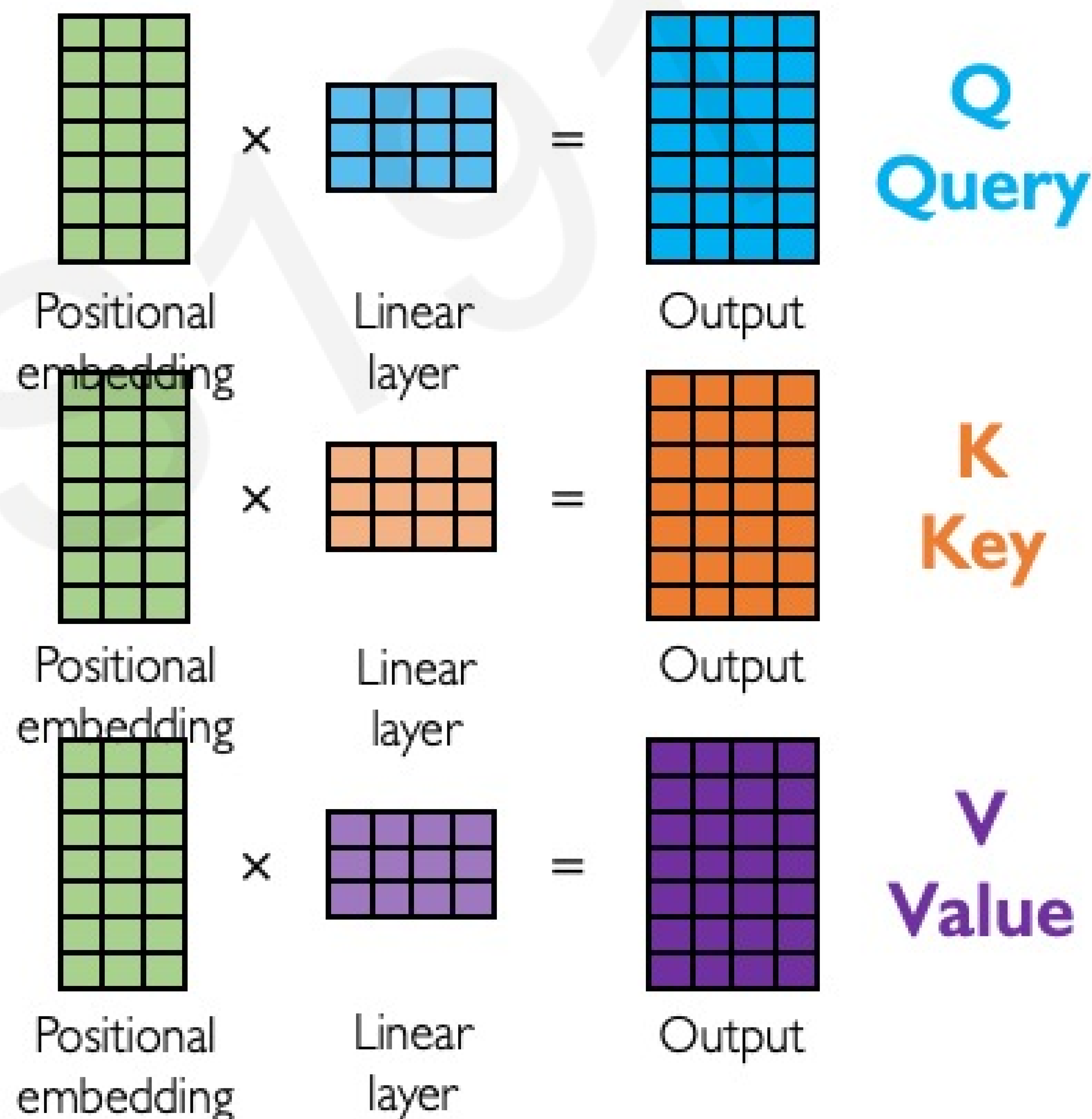
Data is fed in all at once! Need to encode position information to understand order.

MIT 6.S191

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**



MIT 6.S191

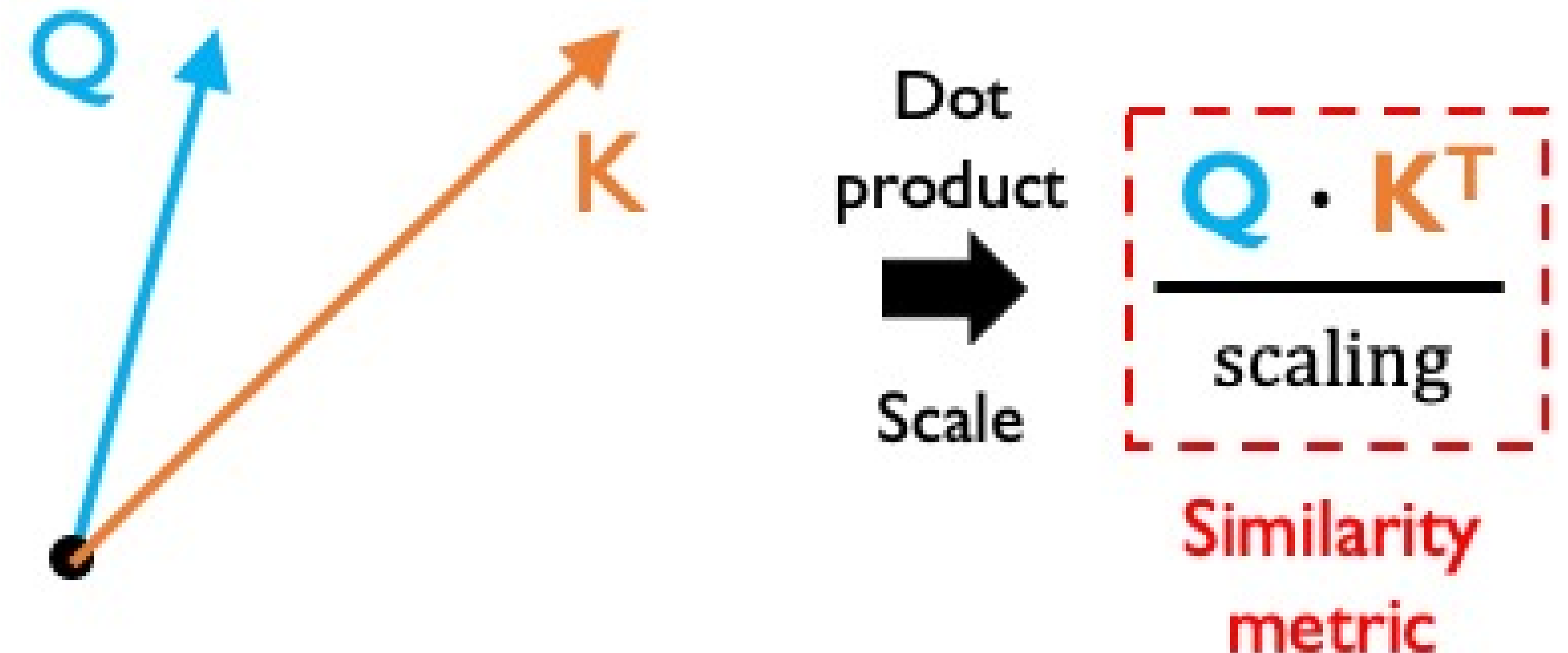
Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Attention score: compute pairwise similarity between each query and key

How to compute similarity between two sets of features?



Also known as the "cosine similarity"

MIT 6.S191

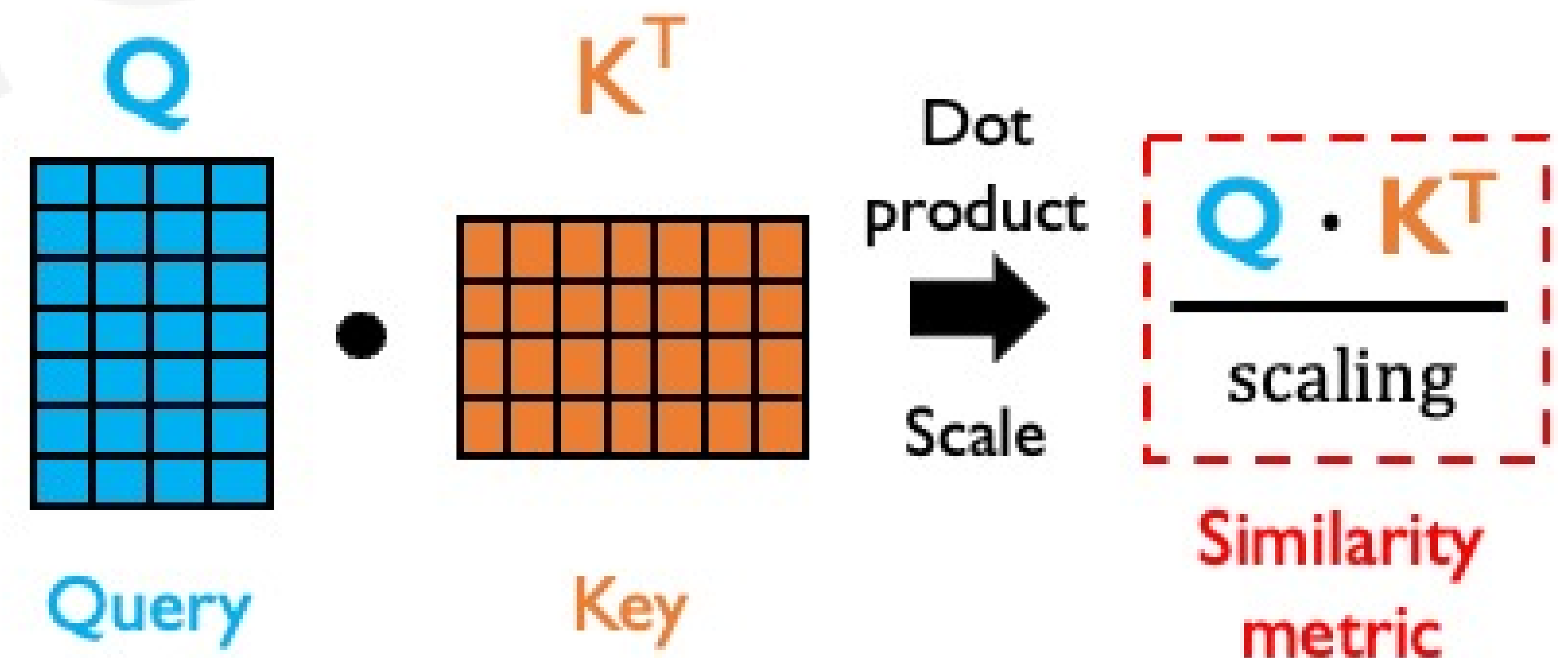
Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Attention score: compute pairwise similarity between each query and key

How to compute similarity between two sets of features?



Also known as the "cosine similarity"

MIT 6.S191

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Attention weighting: where to attend to! How similar is the key to the query?

$$softmax\left(\frac{Q \cdot K^T}{scaling}\right)$$

Attention weighting

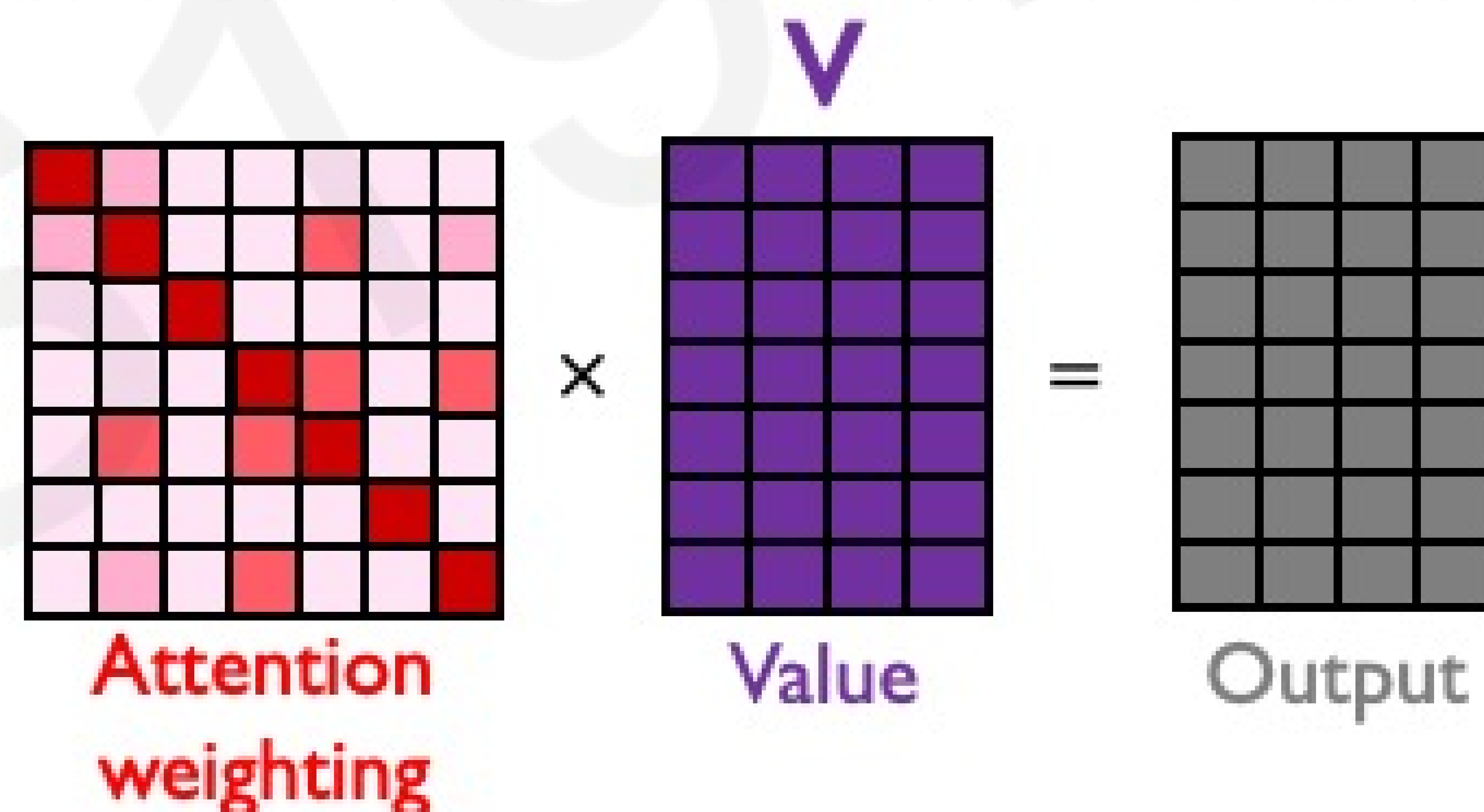
MIT 6.S191

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Last step: self-attend to extract features



$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V = A(Q,K,V)$$

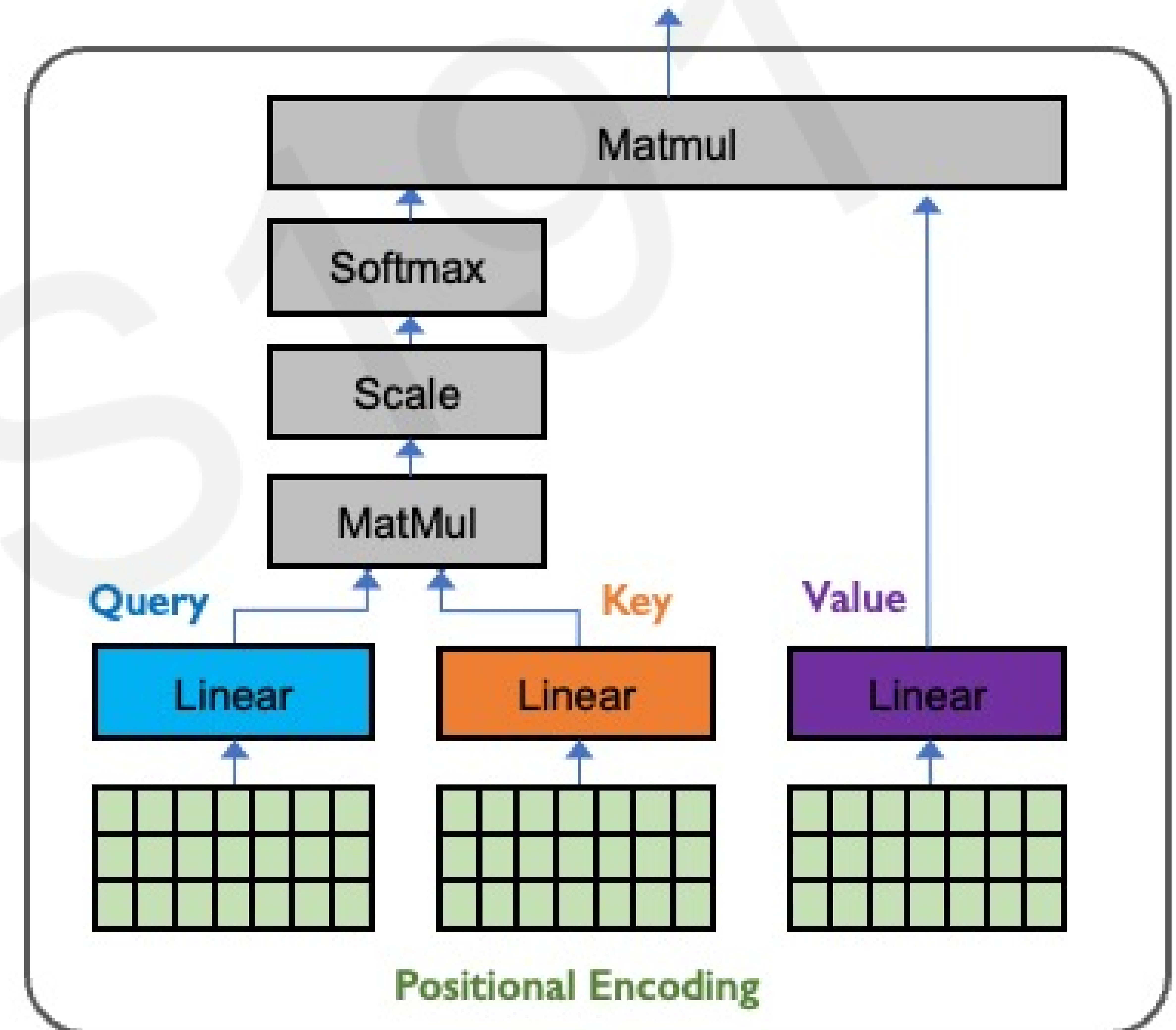
MIT 6.S191

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

These operations form a self-attention head that can plug into a larger network. Each head attends to a different part of input.

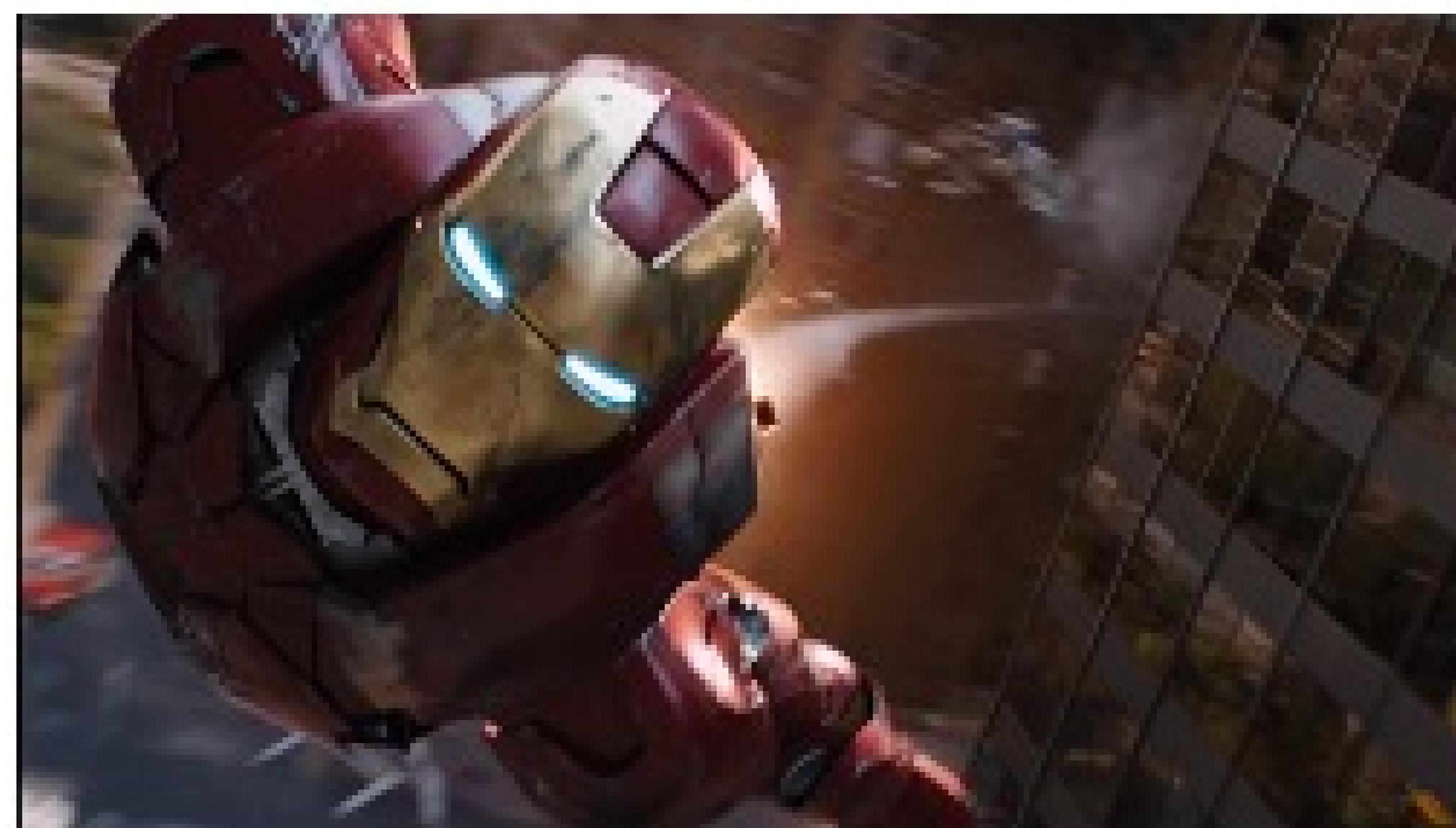


$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V$$

Applying Multiple Self-Attention Heads

Attention weighting

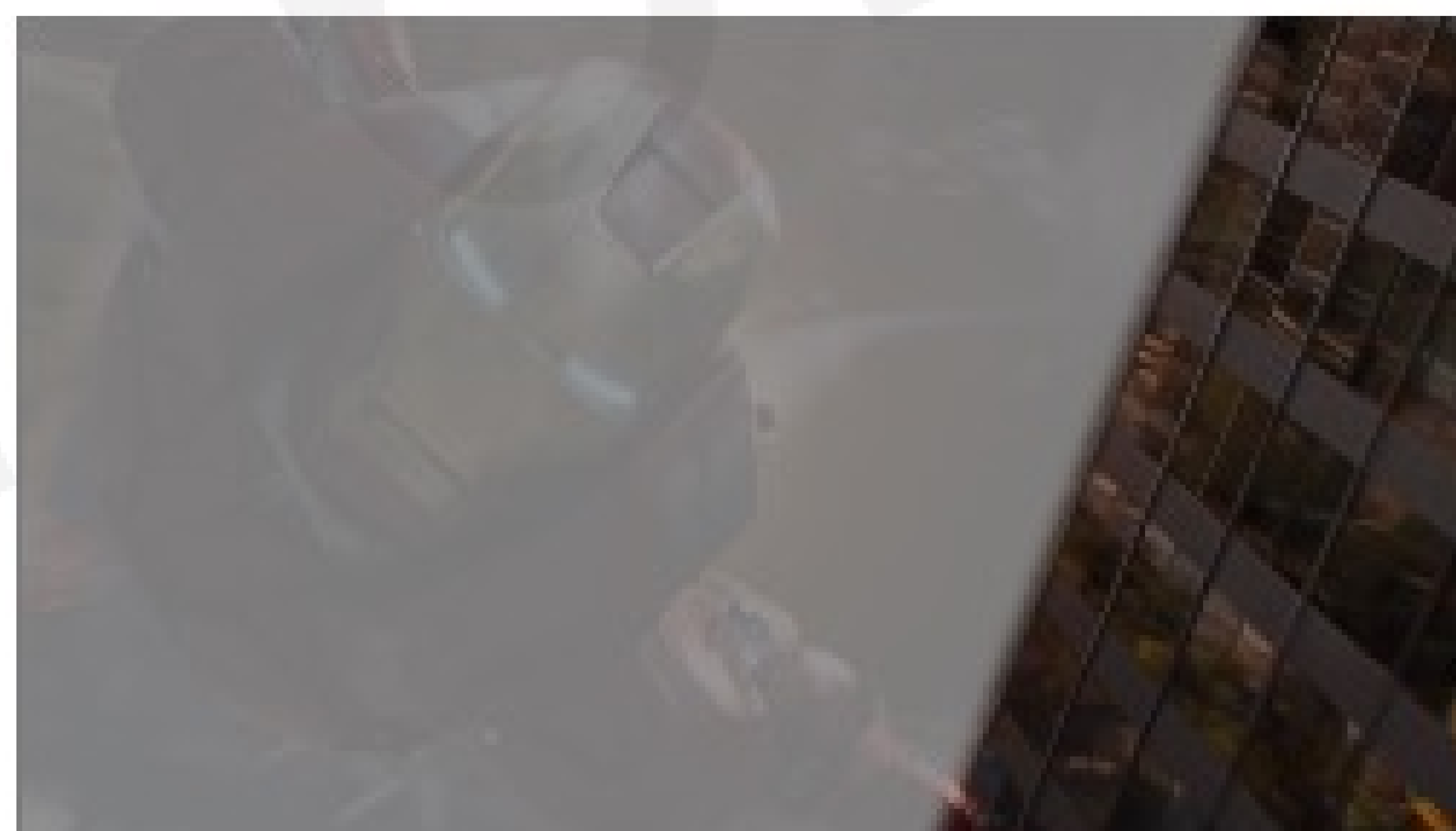
×

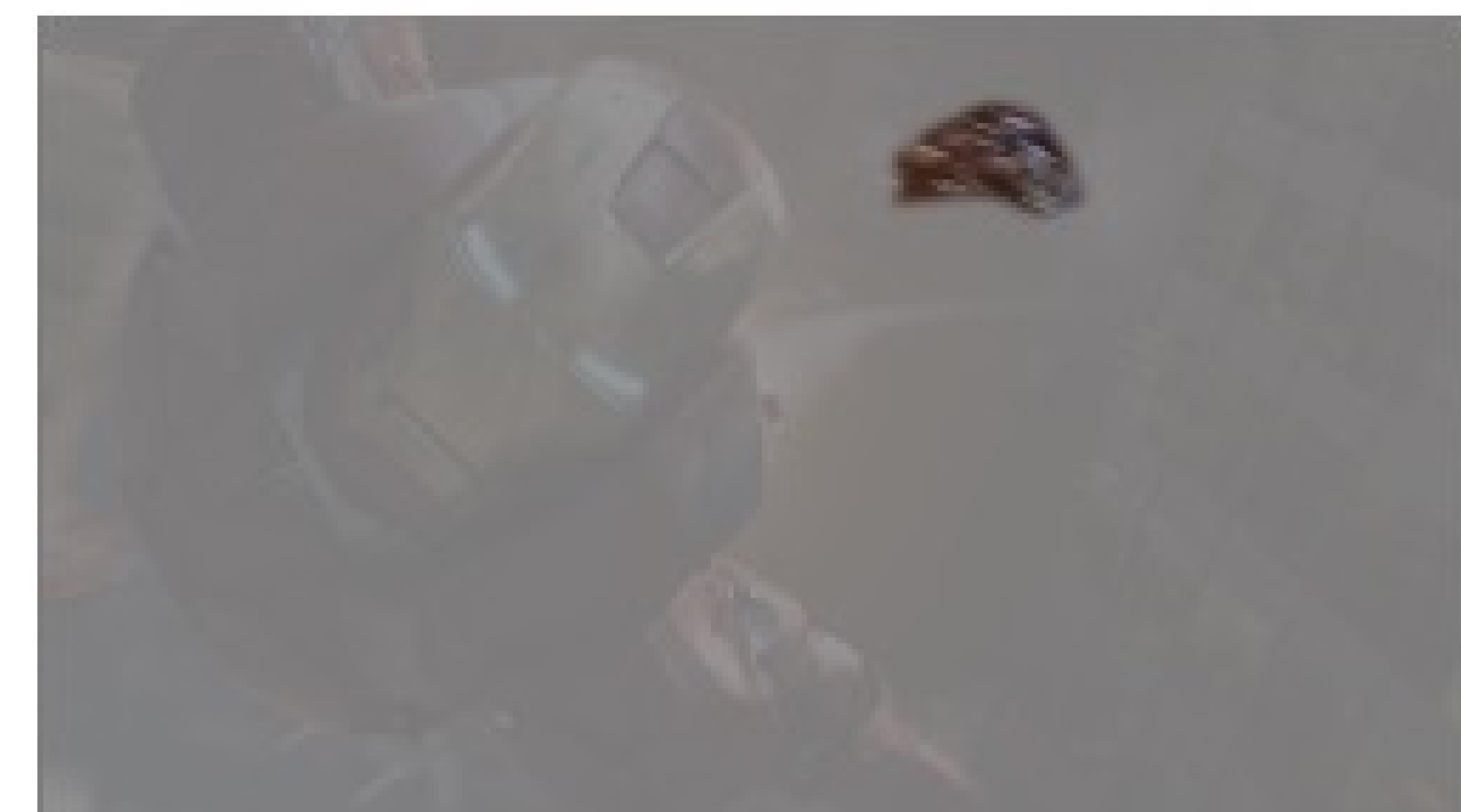

Value

=

Output

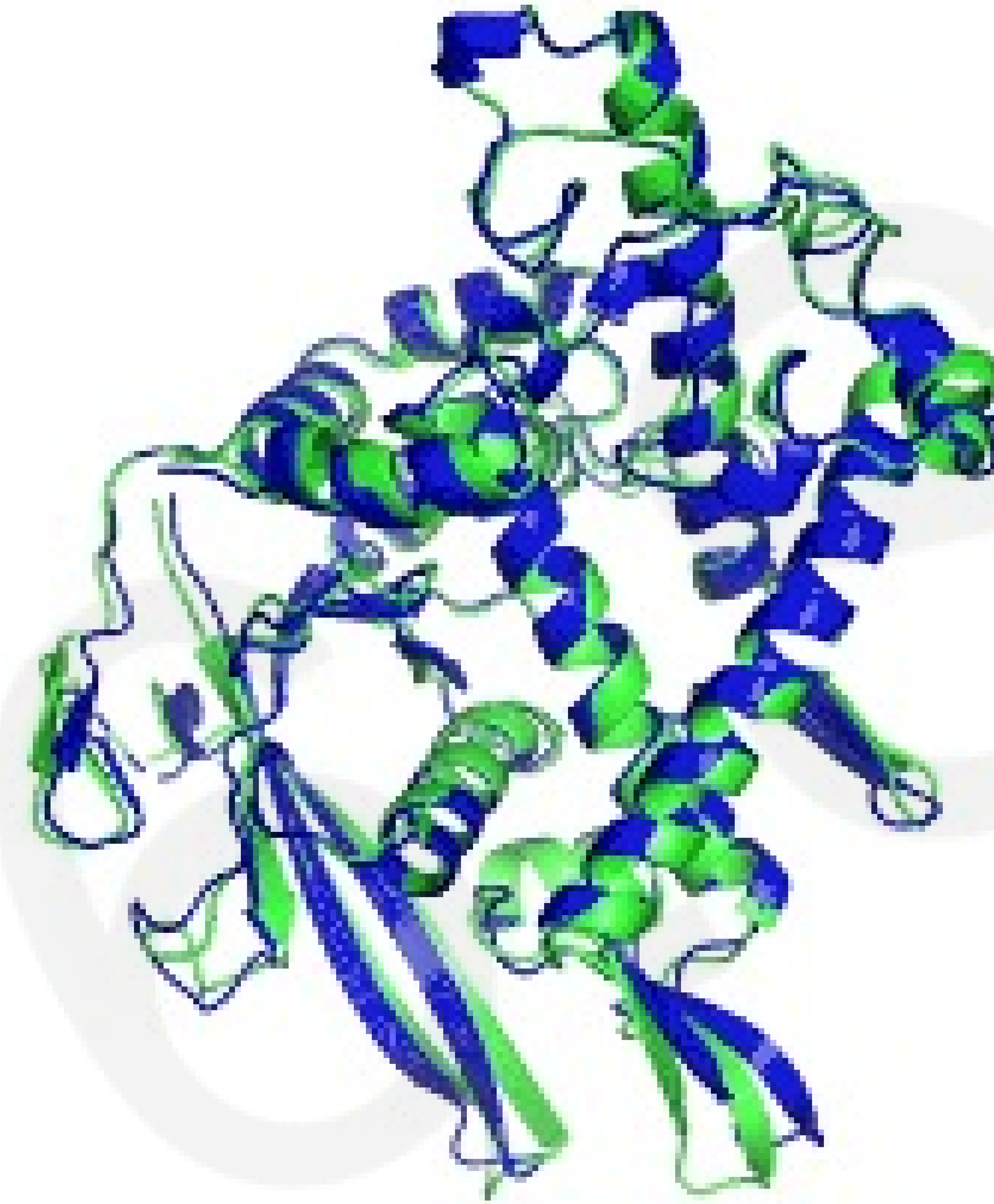
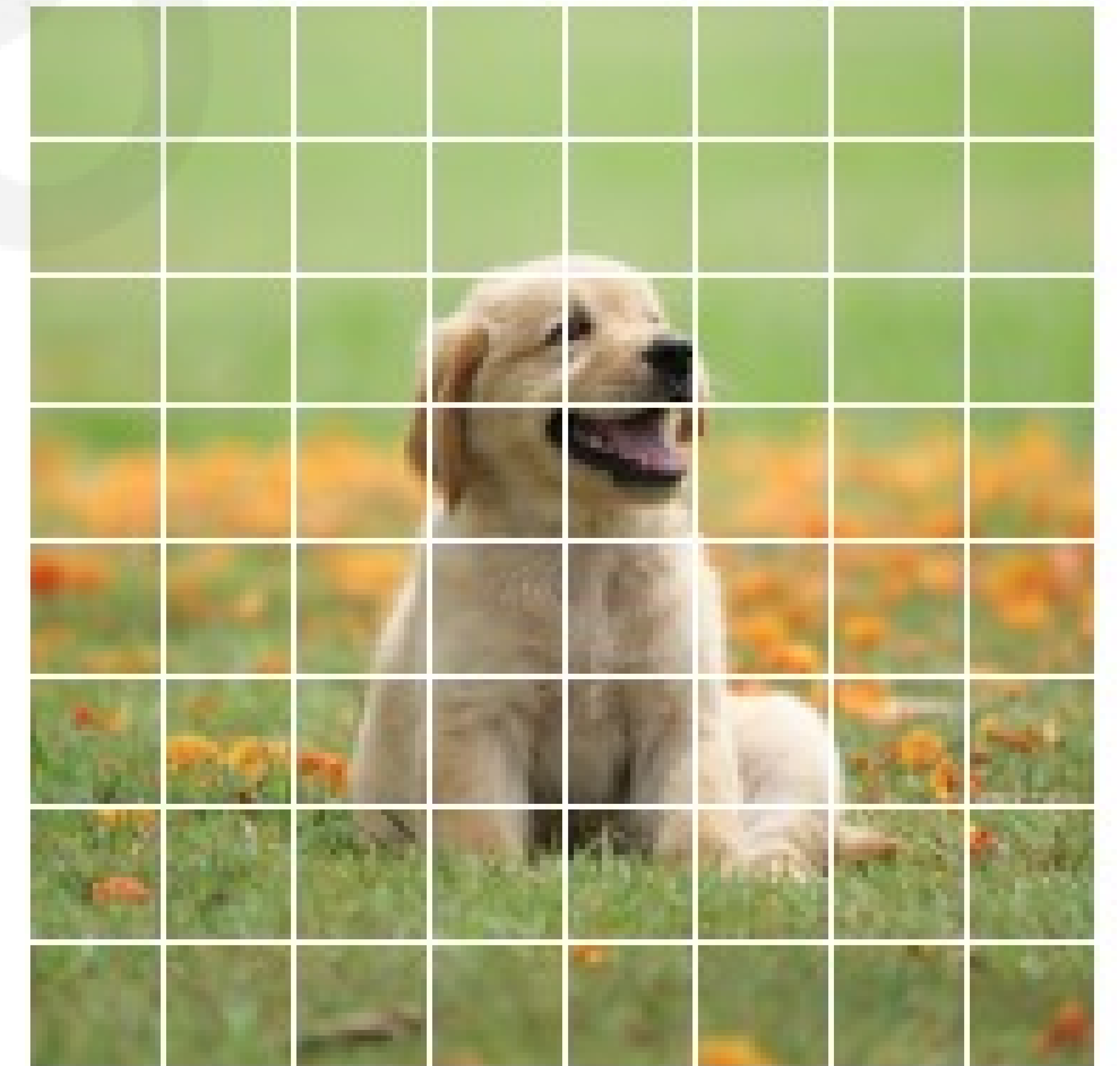
Output of attention head 1


Output of attention head 2


Output of attention head 3

MIT 6.S191

Self-Attention Applied

Language Processing	Biological Sequences	Computer Vision
An armchair in the shape of an avocado		
Transformers: BERT, GPT	AlphaFold2	Vision Transformers
Devlin et al., *NAACL* 2019 Brown et al., *NeurIPS* 2020	Jumper et al., *Nature* 2021	Dosovitskiy et al., *ICLR* 2020

MIT 6.S191

Deep Learning for Sequence Modeling: Summary

1. RNNs are well suited for **sequence modeling** tasks
2. Model sequences via a **recurrence relation**
3. Training RNNs with **backpropagation through time**
4. Models for **music generation**, classification, machine translation, and more
5. Self-attention to model **sequences without recurrence**
6. Self-attention is the basis for many **large language models** – stay tuned!

MIT 6.S191