Introduction to Deep Learning

Alexander Amini

MIT 6.S191

January 27, 2020

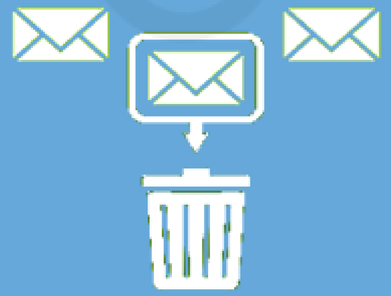
What is Deep Learning?

Artificial Intelligence

Any technique that enables computers to mimic human behavior

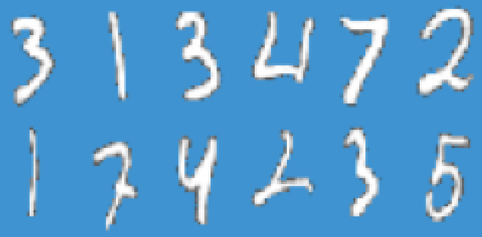
Machine Learning

Ability to learn without explicitly being programmed



Deep Learning

Extract patterns from data using neural networks



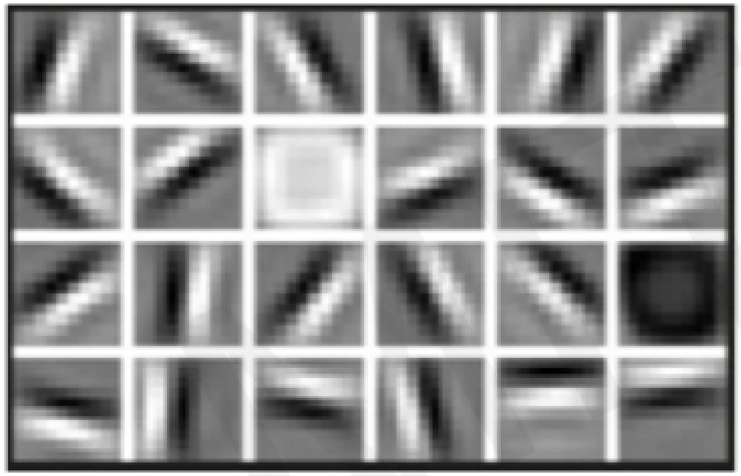
MIT 6.S191

Why Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

High Level Features

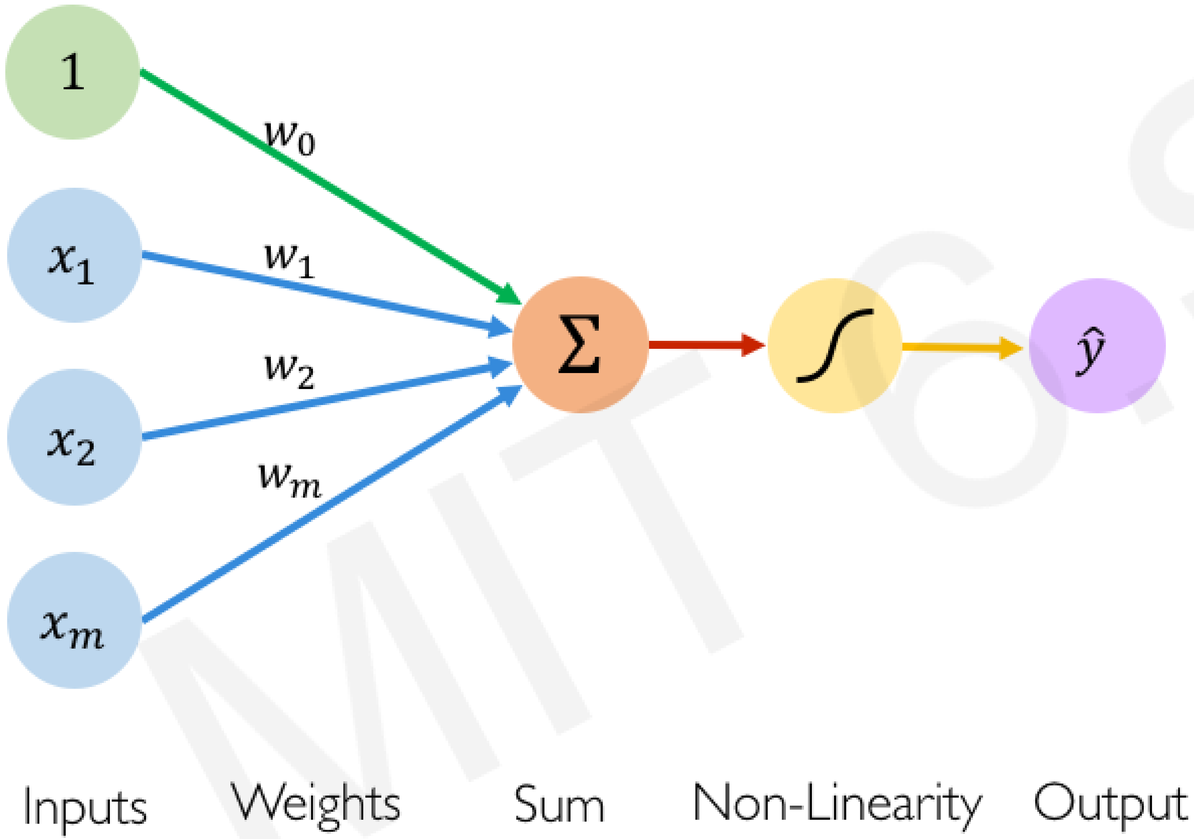
Facial Structure

The Perceptron

The structural building block of deep learning

MIT 6.S191

The Perceptron: Forward Propagation



Output

Linear combination of inputs

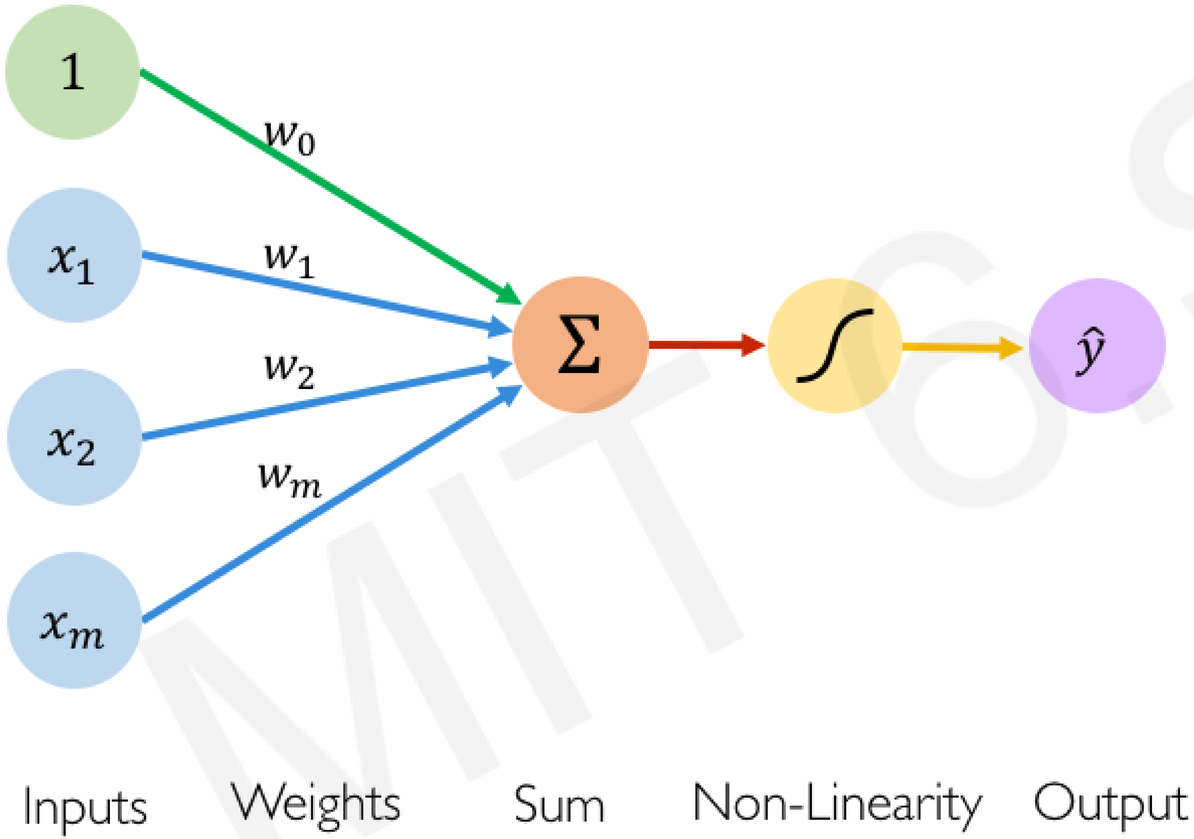
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Non-linear activation function

Bias

MIT 6.S191

The Perceptron: Forward Propagation



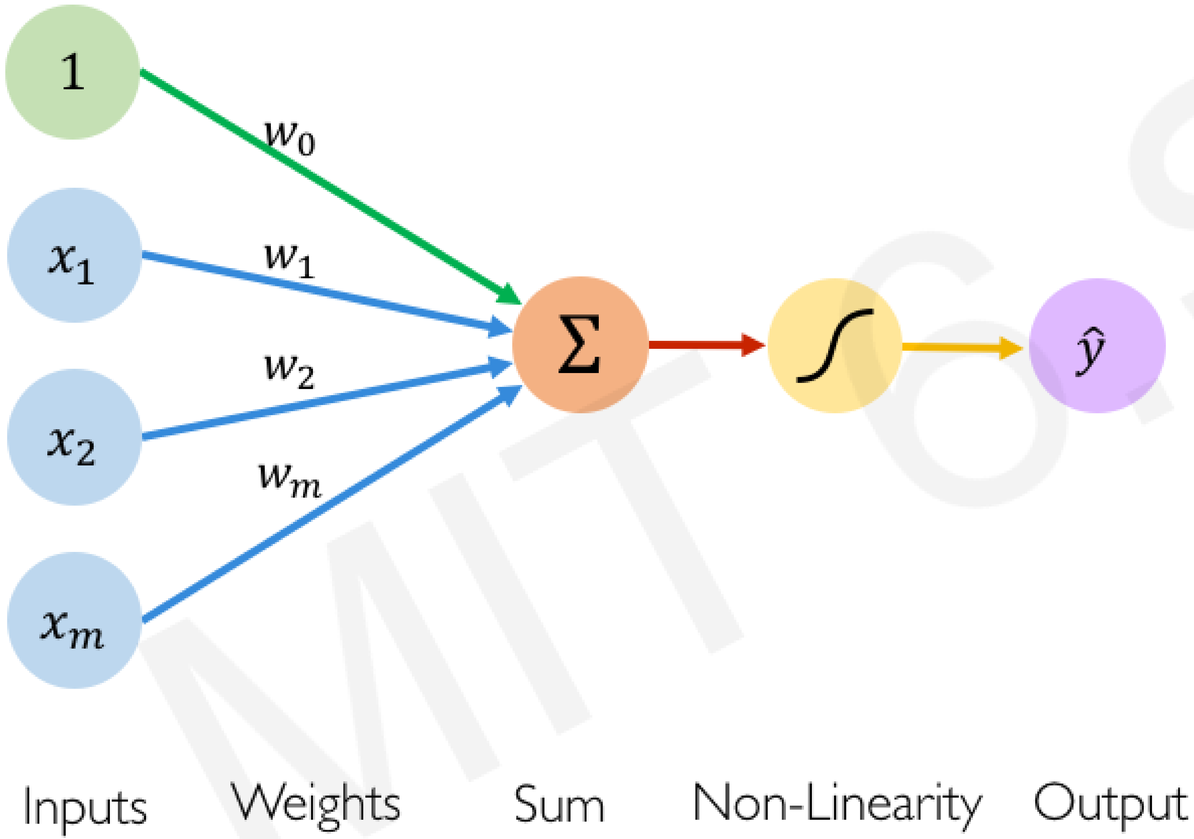
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i\right)$$

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

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

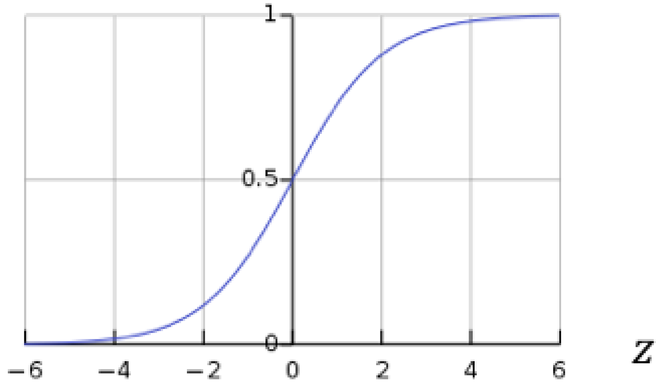


Activation Functions

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

- Example: sigmoid function

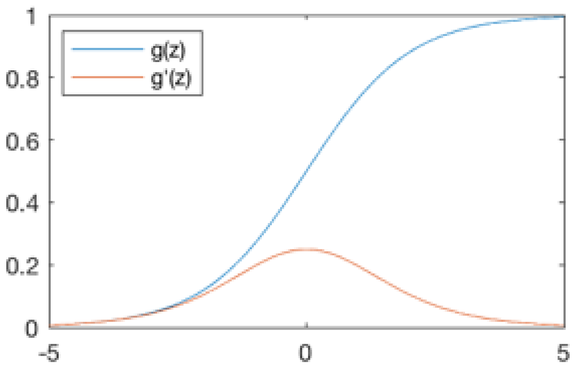
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



MIT 6.S191

Common Activation Functions

Sigmoid Function

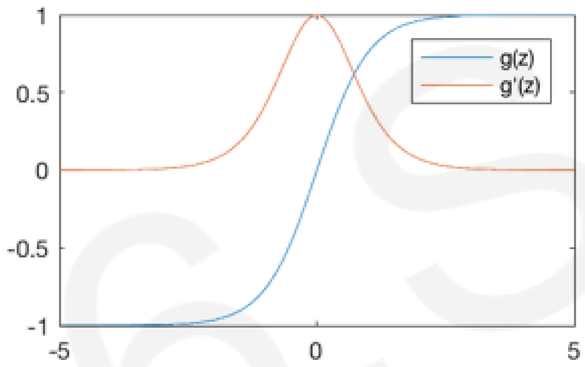


$$g(z) = \frac{1}{1 + e^{-z}}$$

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

```
tf.math.sigmoid(z)
```

Hyperbolic Tangent

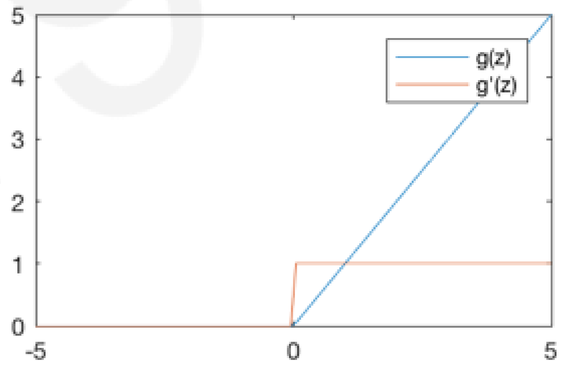


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

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

```
tf.math.tanh(z)
```

Rectified Linear Unit (ReLU)



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

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

```
tf.nn.relu(z)
```

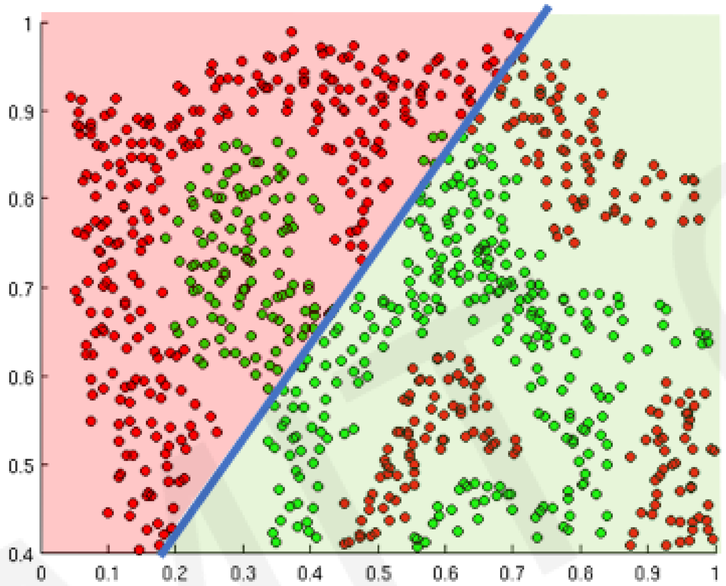
TensorFlow code blocks

NOTE: All activation functions are non-linear

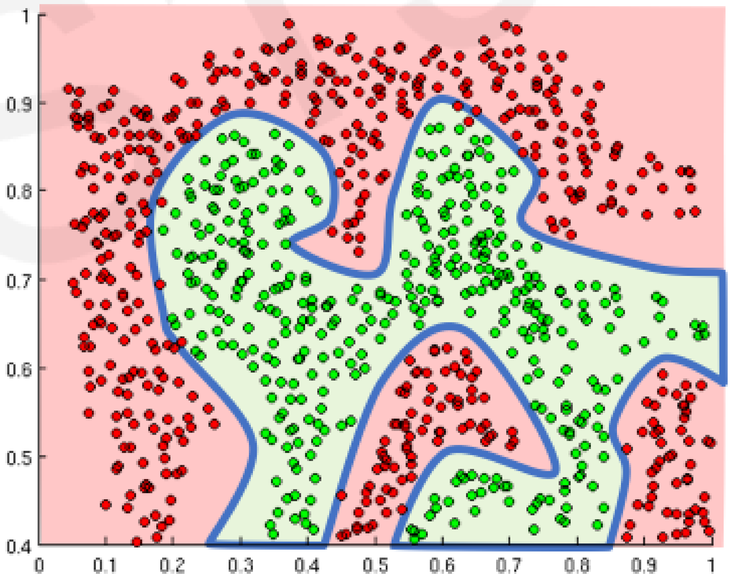
MIT 6.S191

Importance of Activation Functions

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



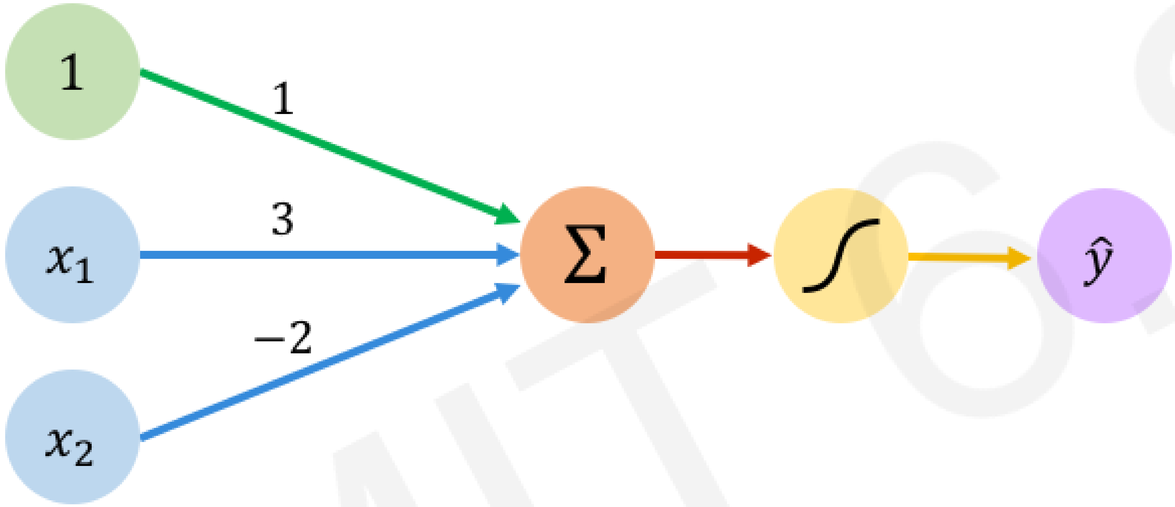
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

MIT 6.S191

The Perceptron: Example



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

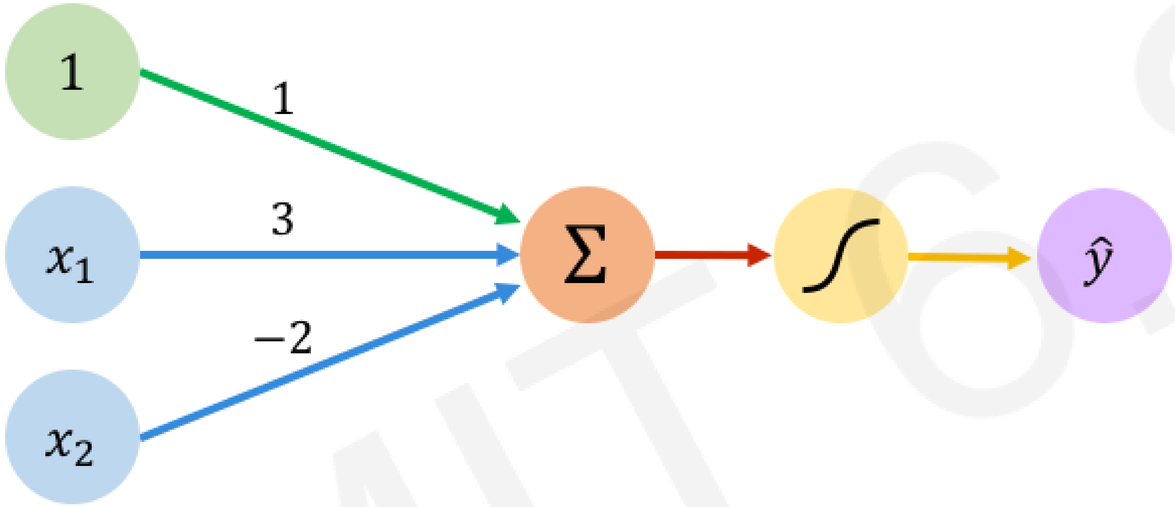
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

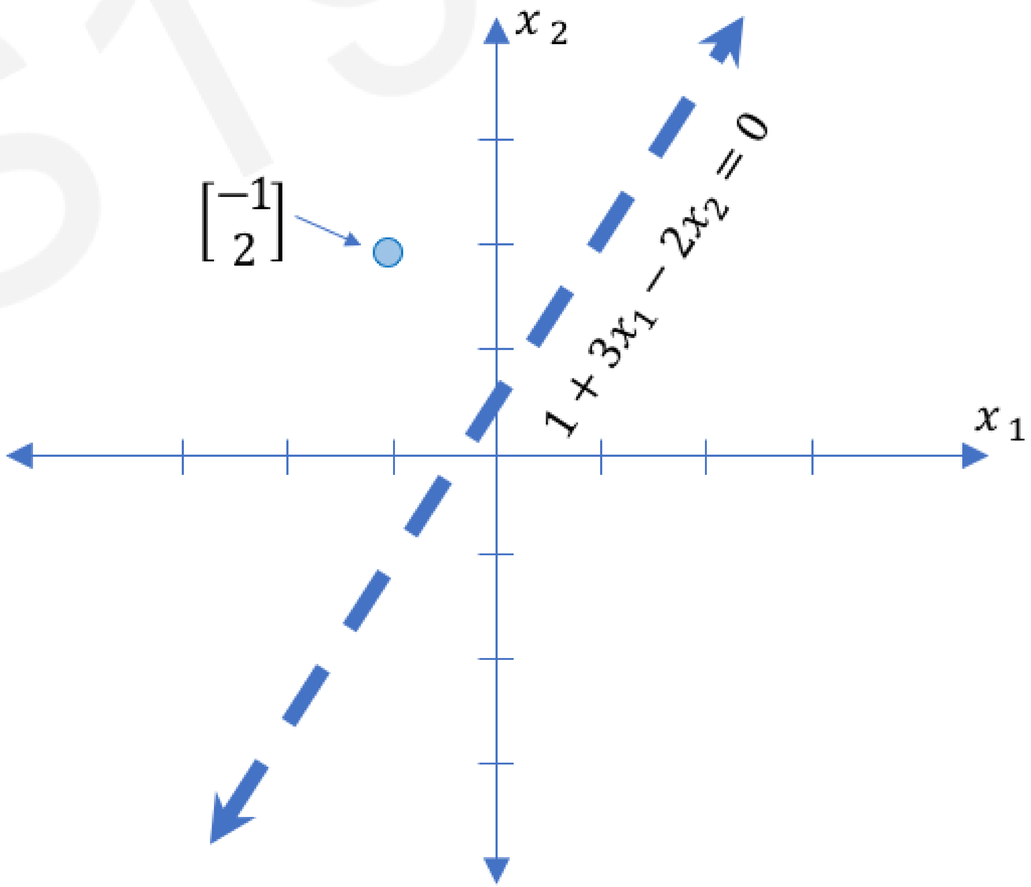
$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

MIT 6.S191

The Perceptron: Example



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

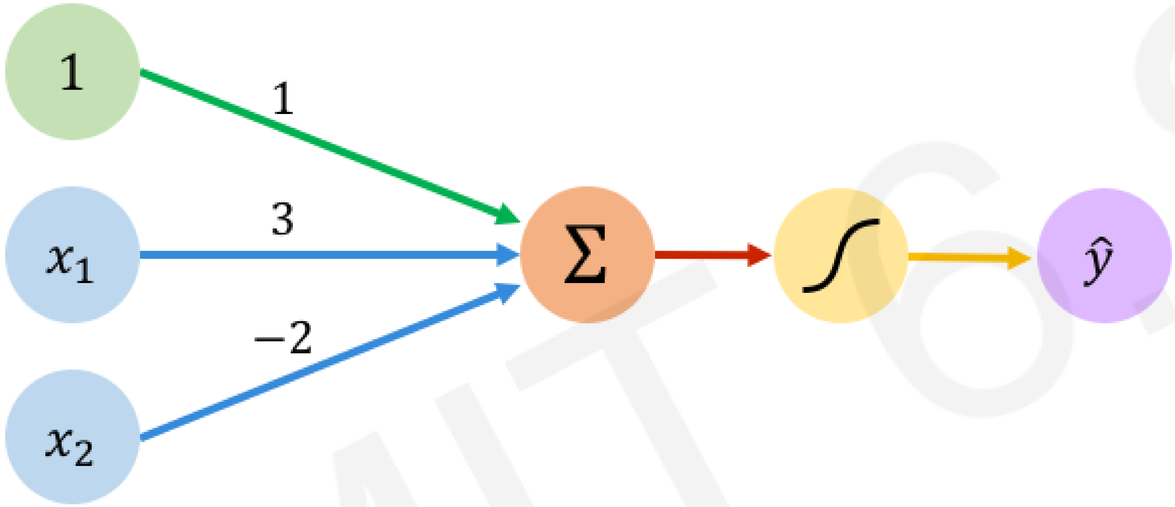


Assume we have input: $\boldsymbol{X} = \begin{bmatrix}-1\\2\end{bmatrix}$

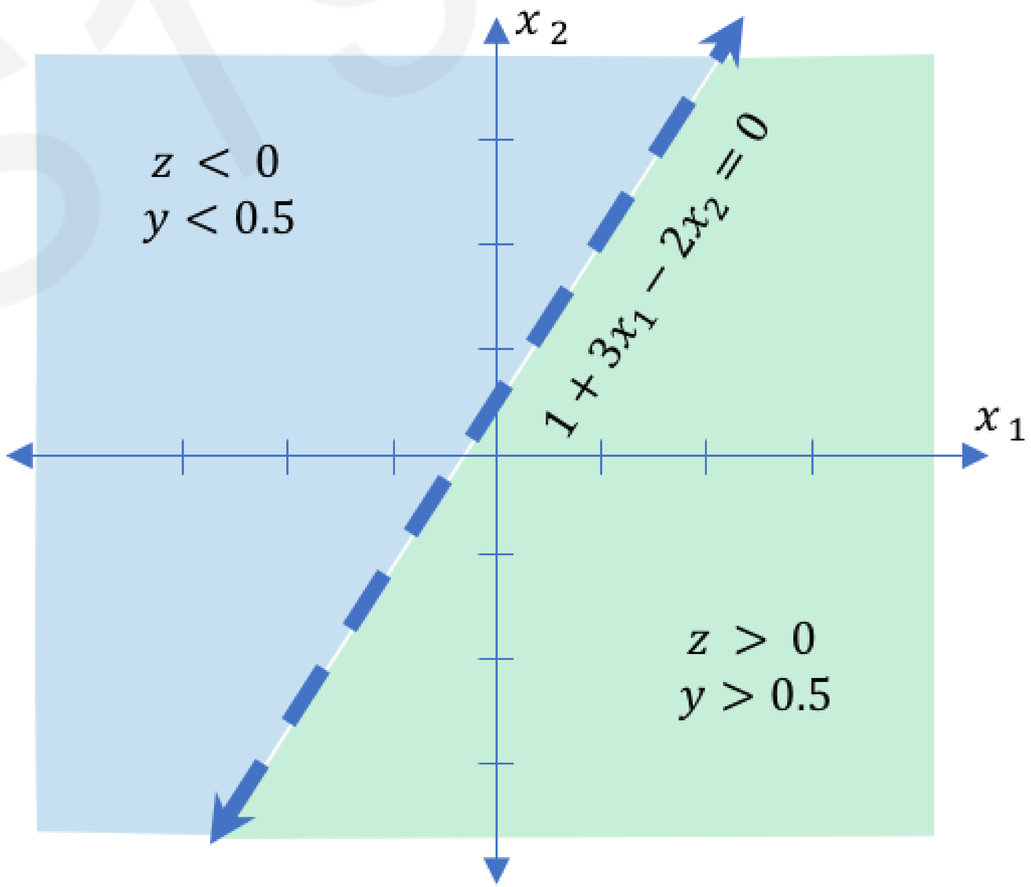
$$\hat{y} = g\big(1 + (3 * -1) - (2 * 2)\big)$$
$$= g(-6) \approx 0.002$$

MIT 6.S191

The Perceptron: Example



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



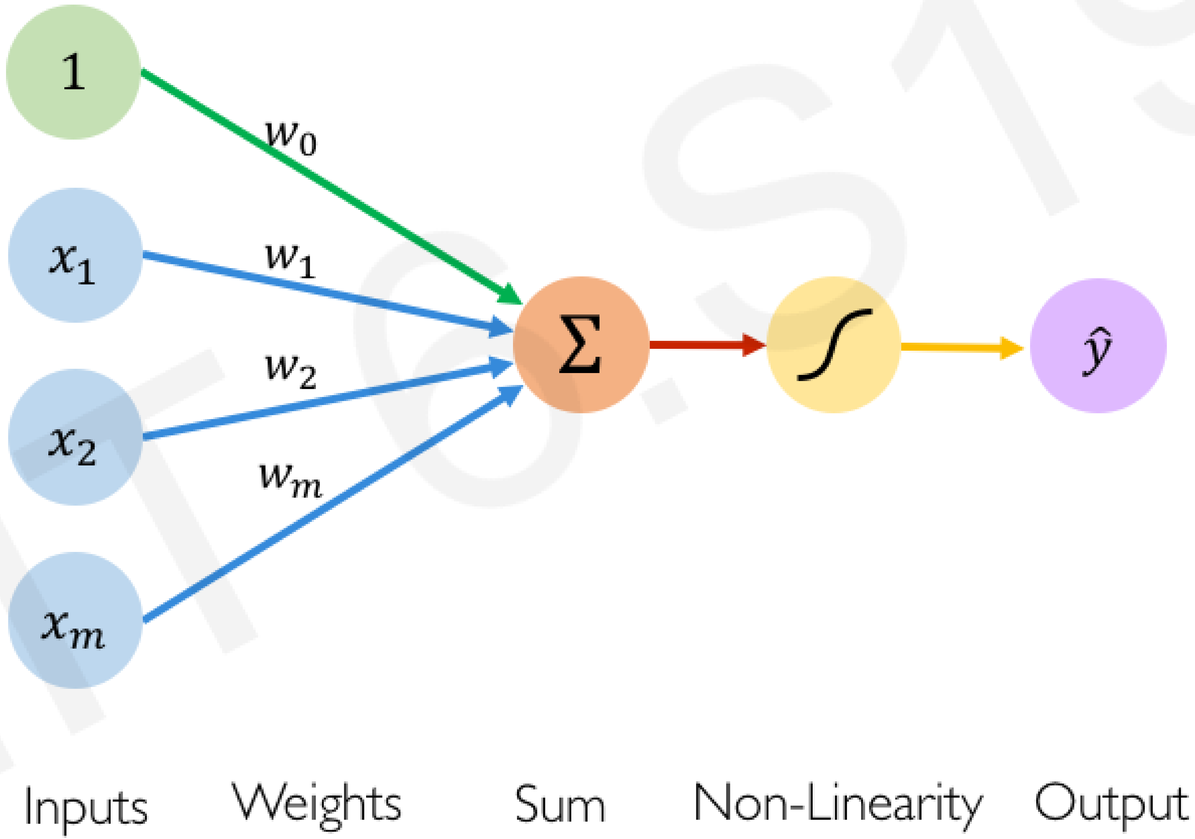
MIT 6.S191

Building Neural Networks with Perceptrons

MIT 6.S191

The Perceptron: Simplified

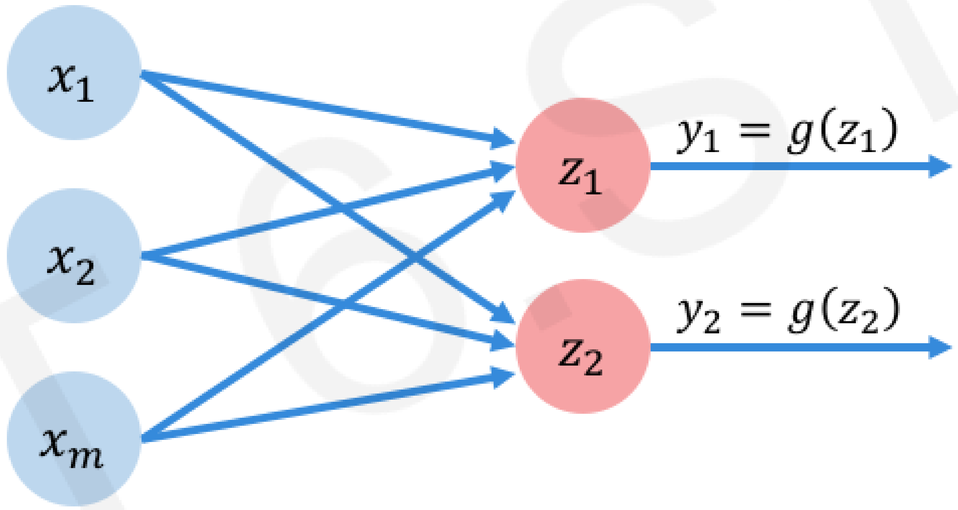
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$



MIT 6.S191

Multi Output Perceptron

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



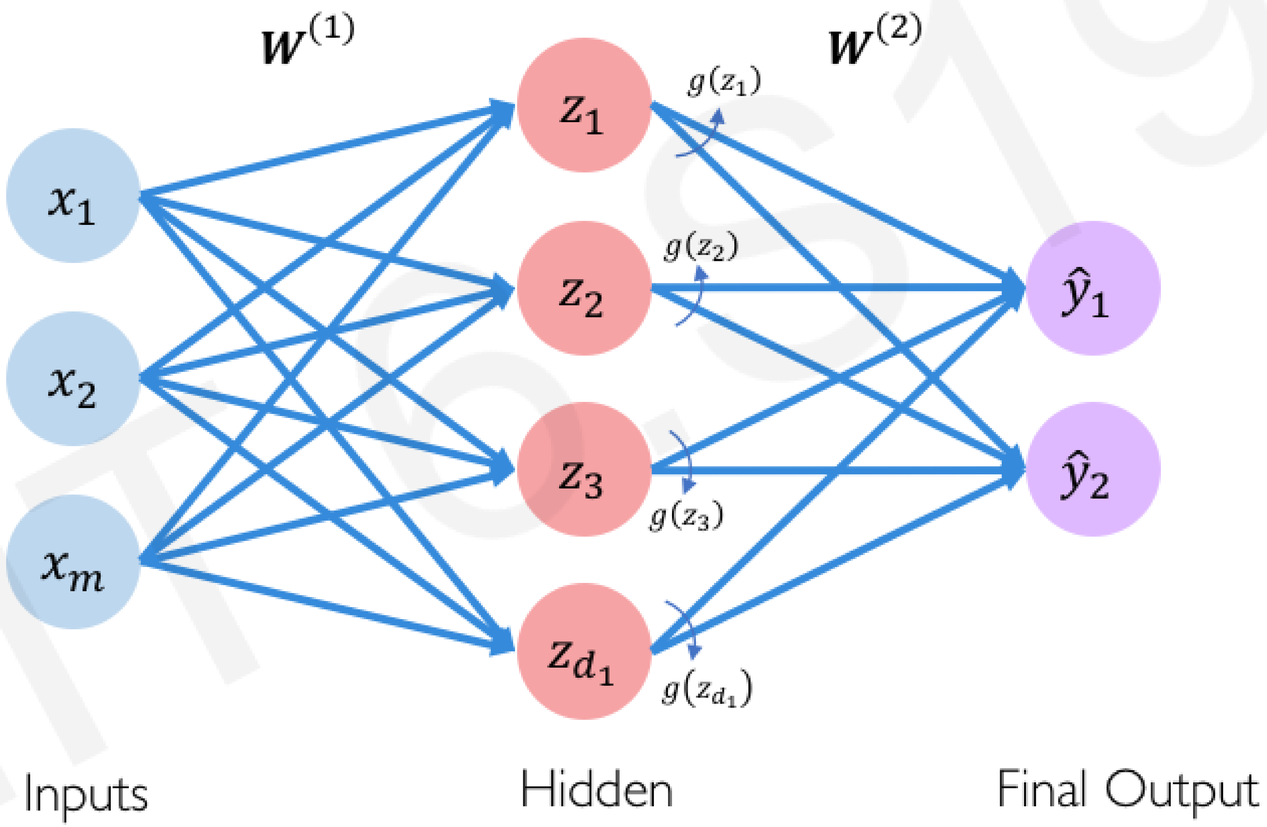
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

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
```

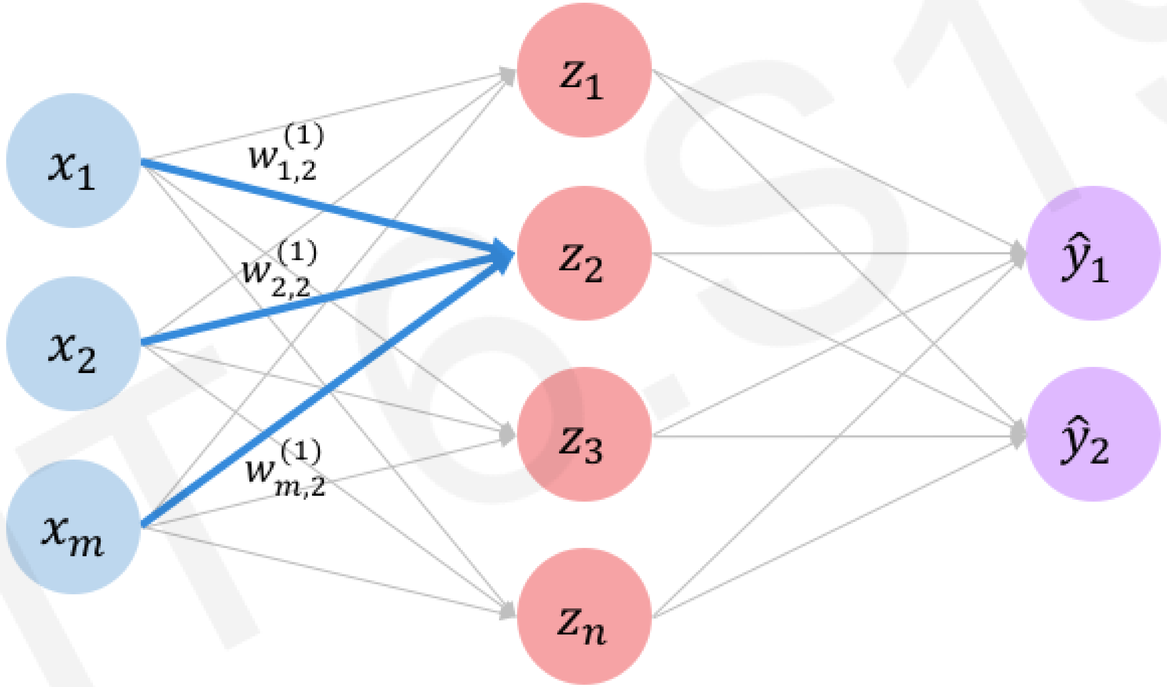
Single Layer Neural Network



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

MIT 6.S191

Single Layer Neural Network

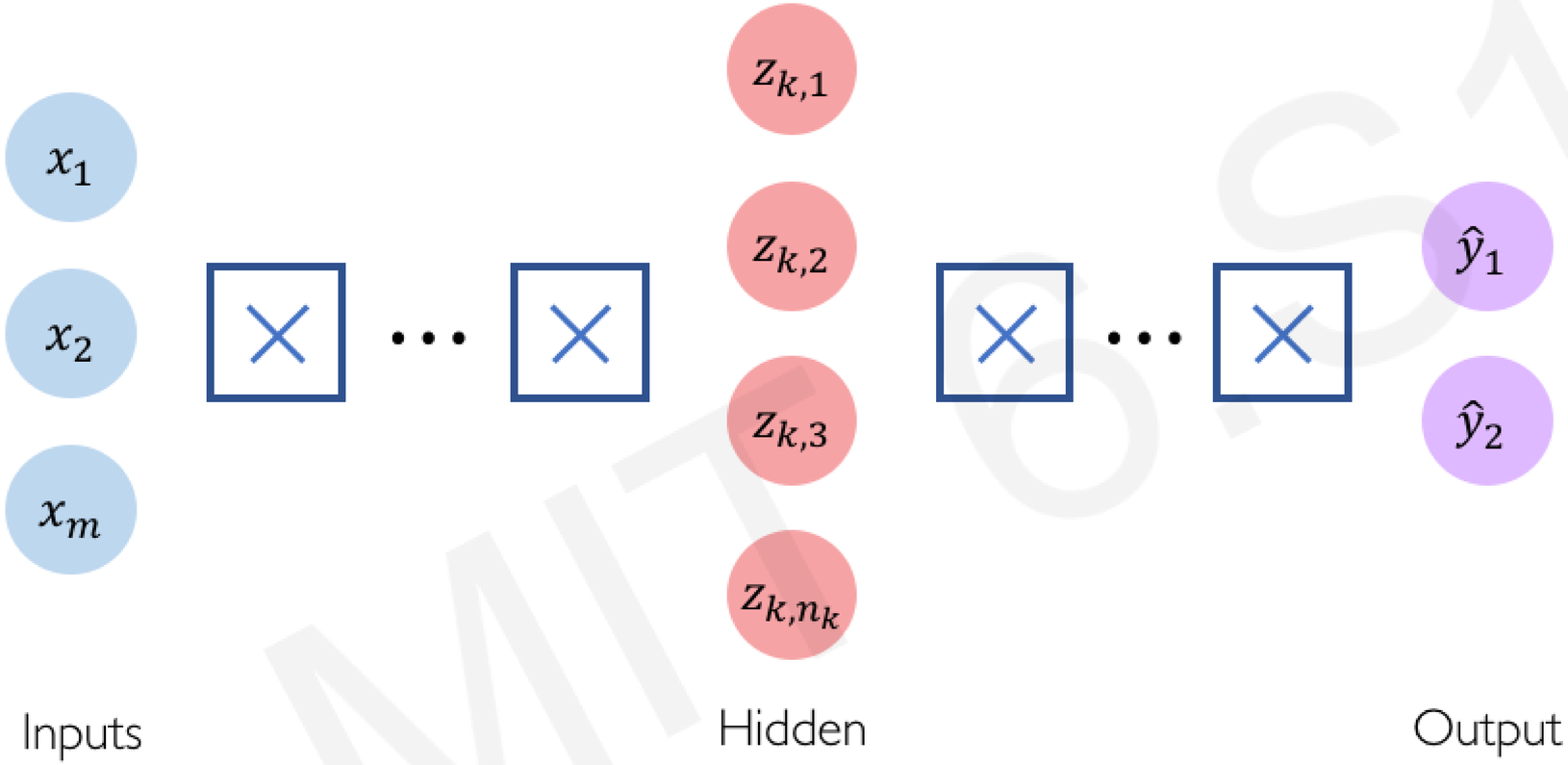


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

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

MIT 6.S191

Deep Neural Network




```
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?

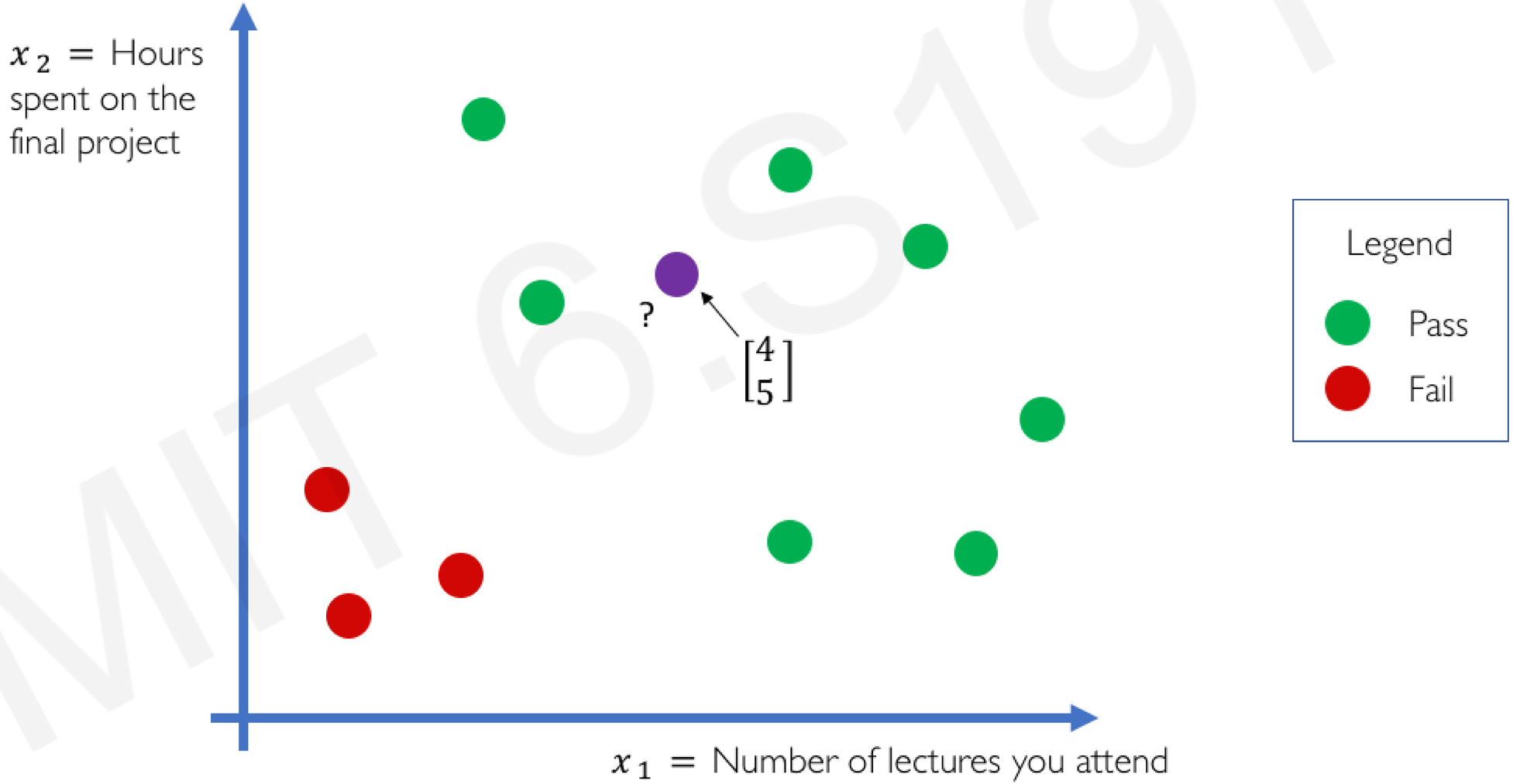
Let's start with a simple two feature model

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?

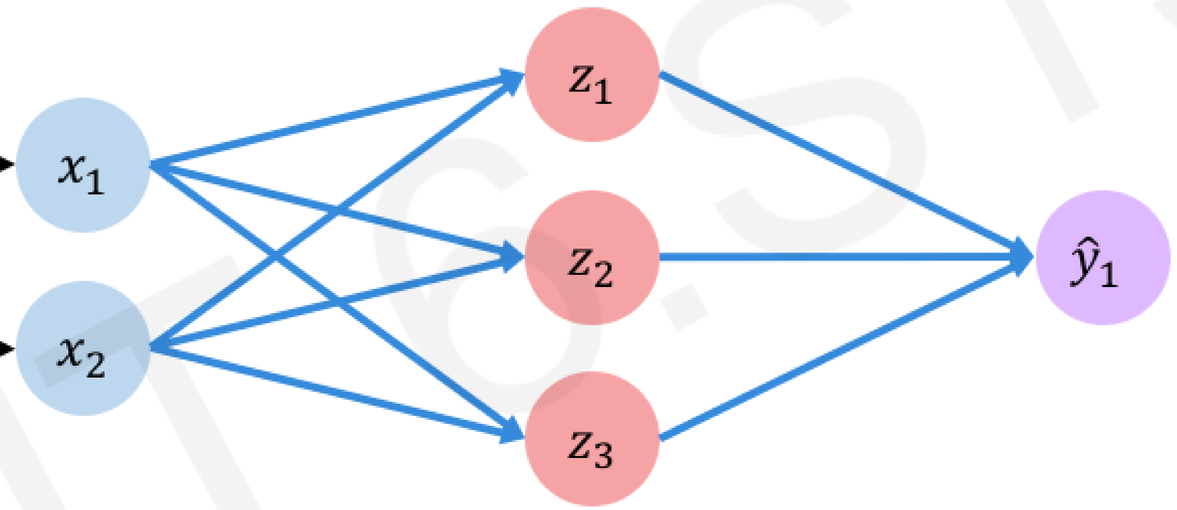


MIT 6.S191

Quantifying Loss

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

$$x^{(1)} = [4\ ,5]$$



Predicted: 0.1
Actual: 1

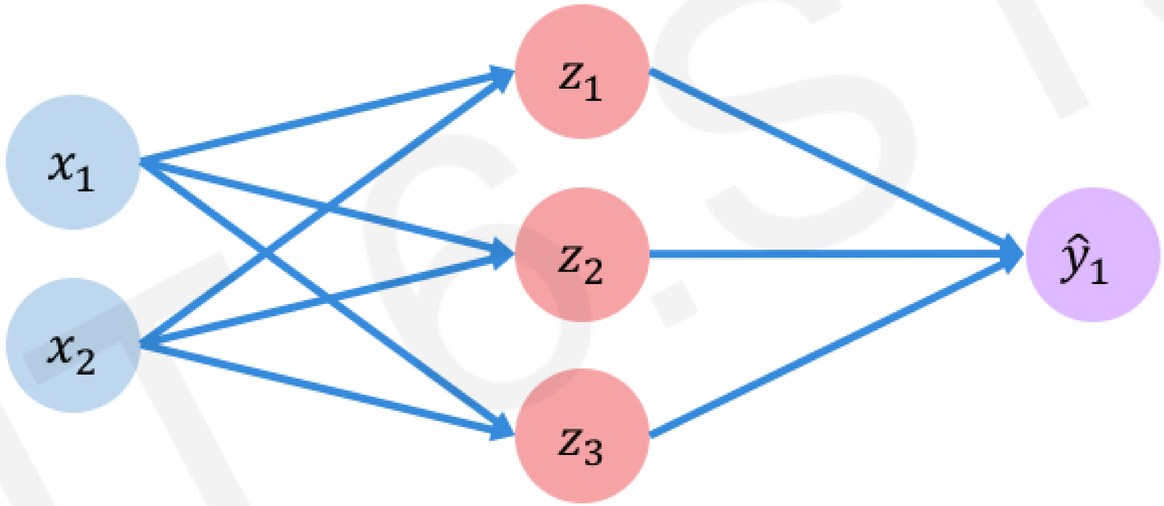
$$\mathcal{L}\left(\underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\right)$$

MIT 6.S191

Empirical Loss

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

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{ccc} f(x) & & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

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

Also known as:

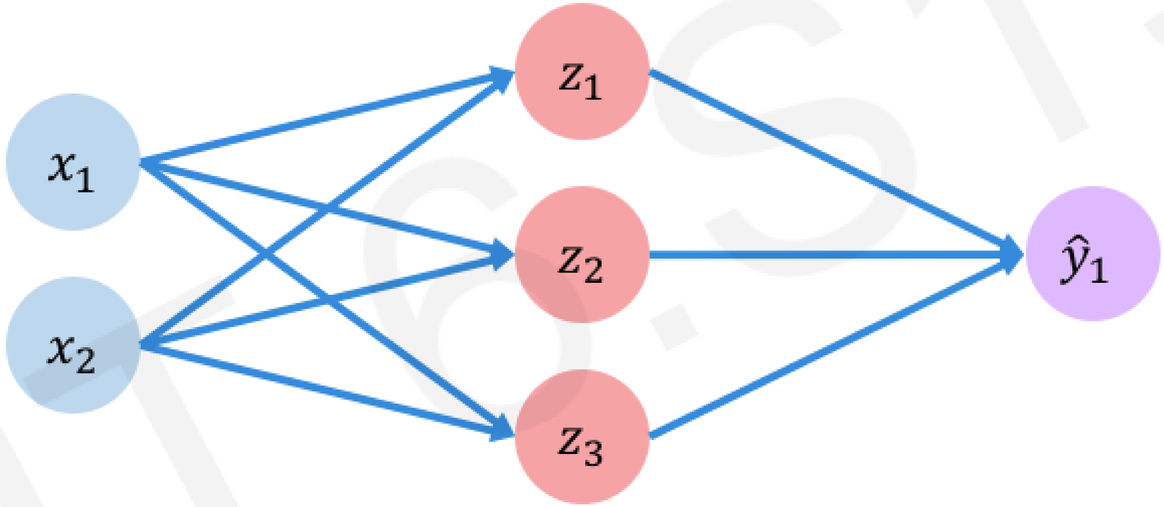
- Objective function
- Cost function
- Empirical Risk

MIT 6.S191

Binary Cross Entropy Loss

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

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{ccc} f(x) & & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

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

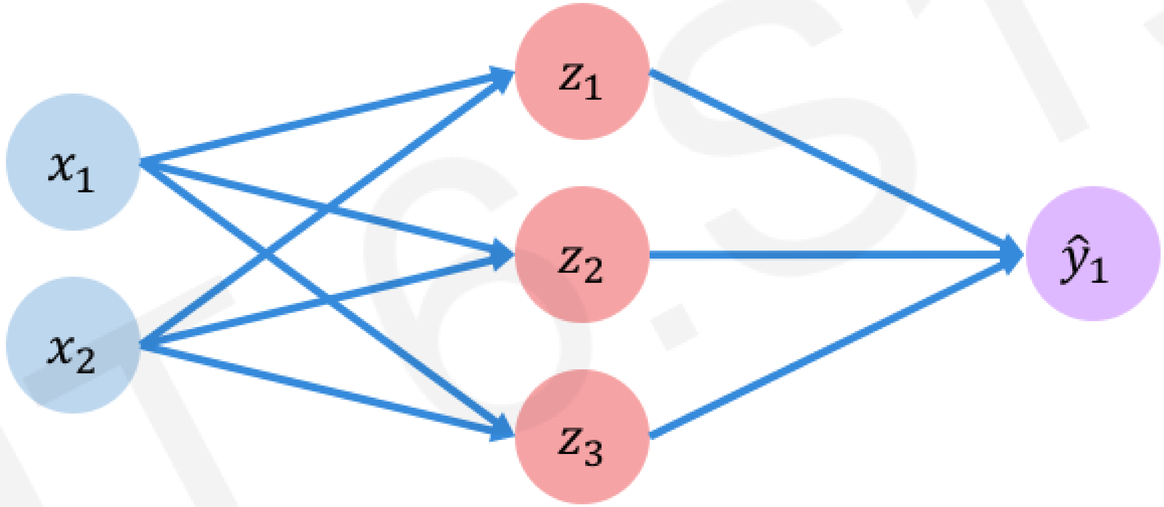
```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

MIT 6.S191

Mean Squared Error Loss

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

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{ccc} f(x) & & y \\ \begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} & \begin{matrix} \times \\ \times \\ \checkmark \\ \\ \end{matrix} & \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix} \end{array}$$

Final Grades (percentage)

$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\left(x^{(i)};\boldsymbol{W}\right)}_{\text{Predicted}}\right)^2$$

```
loss = tf.reduce_mean( tf.square(tf.subtract(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***

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

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

Remember:

$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \dots\right\}$$

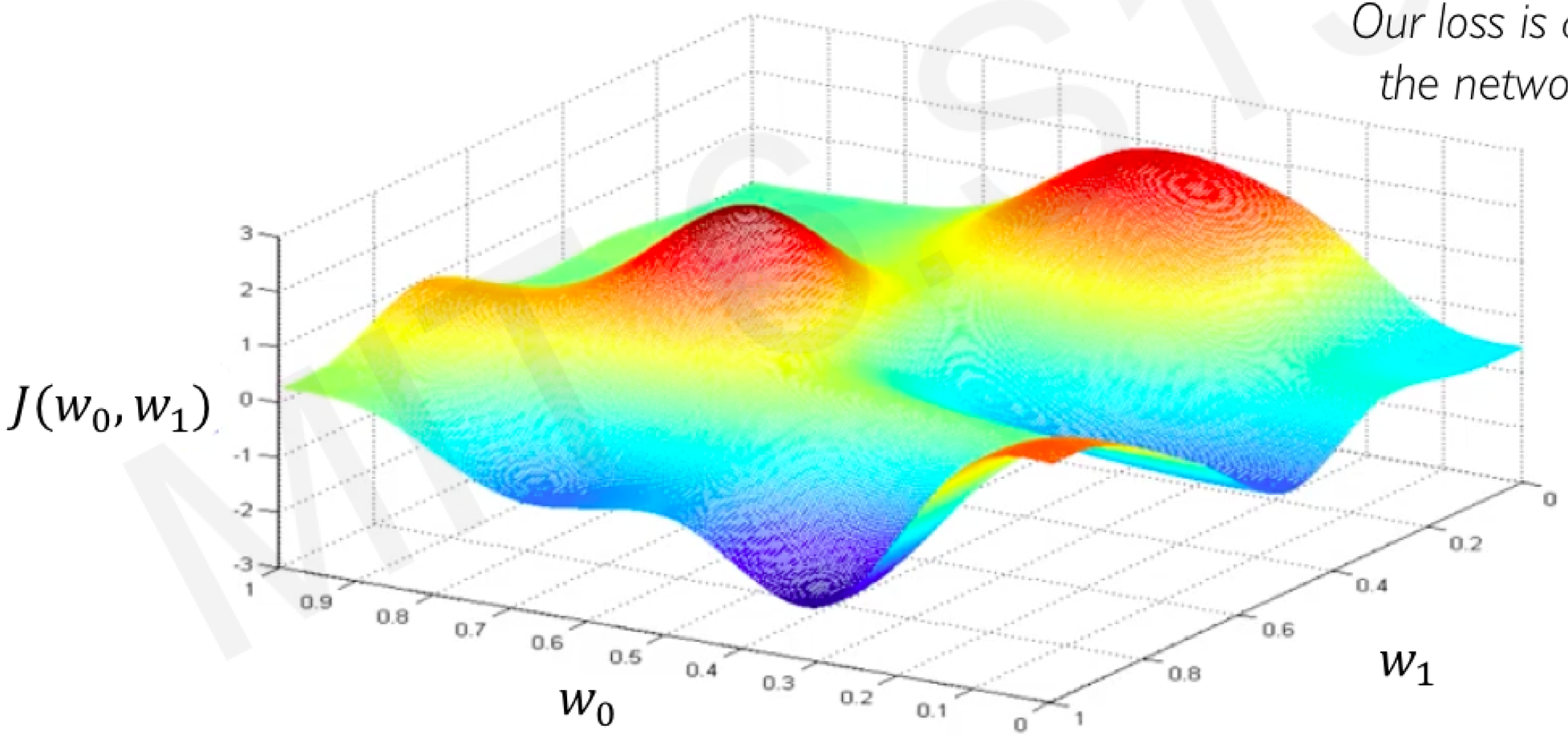
MIT 6.S191

Loss Optimization

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

Remember:

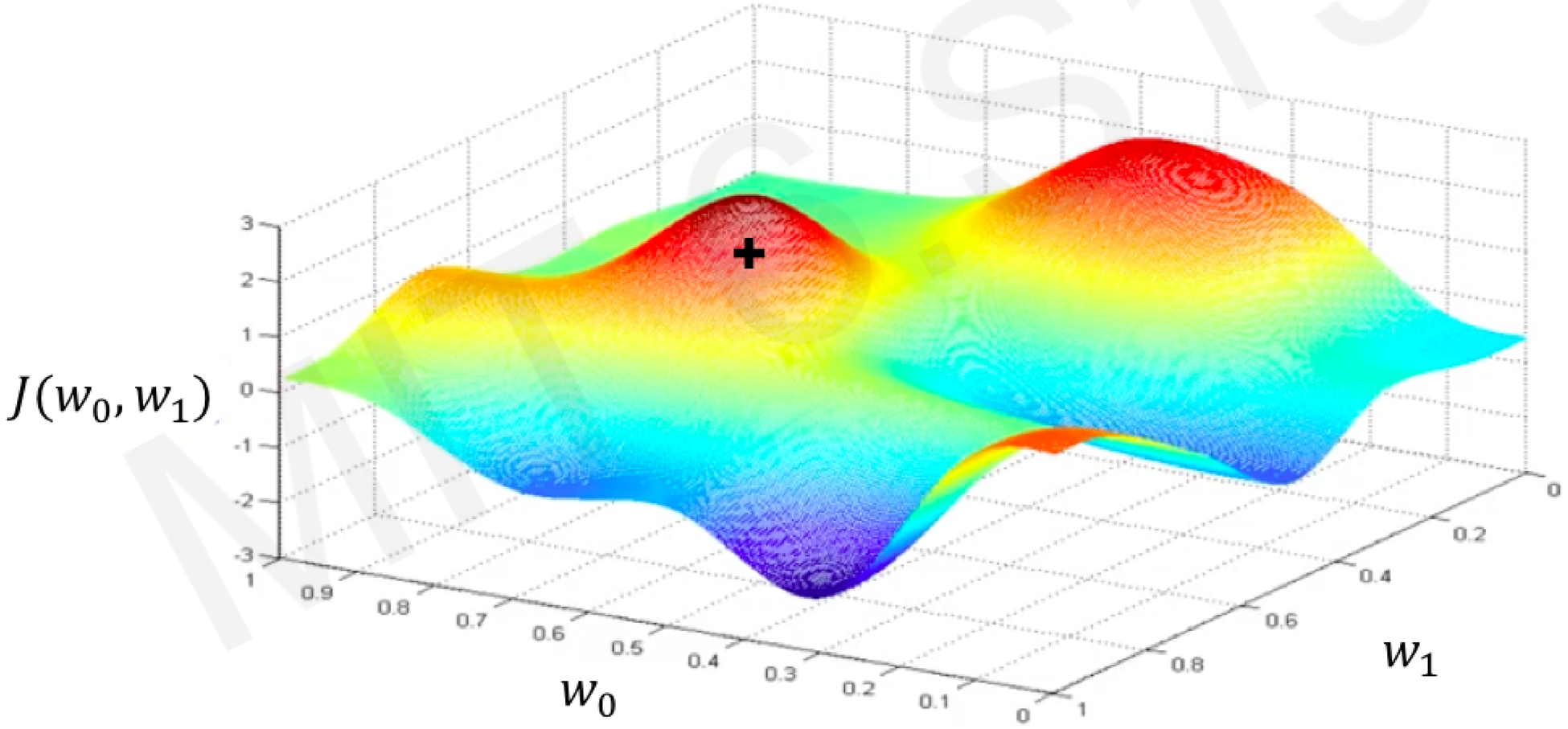
Our loss is a function of the network weights!



MIT 6.S191

Loss Optimization

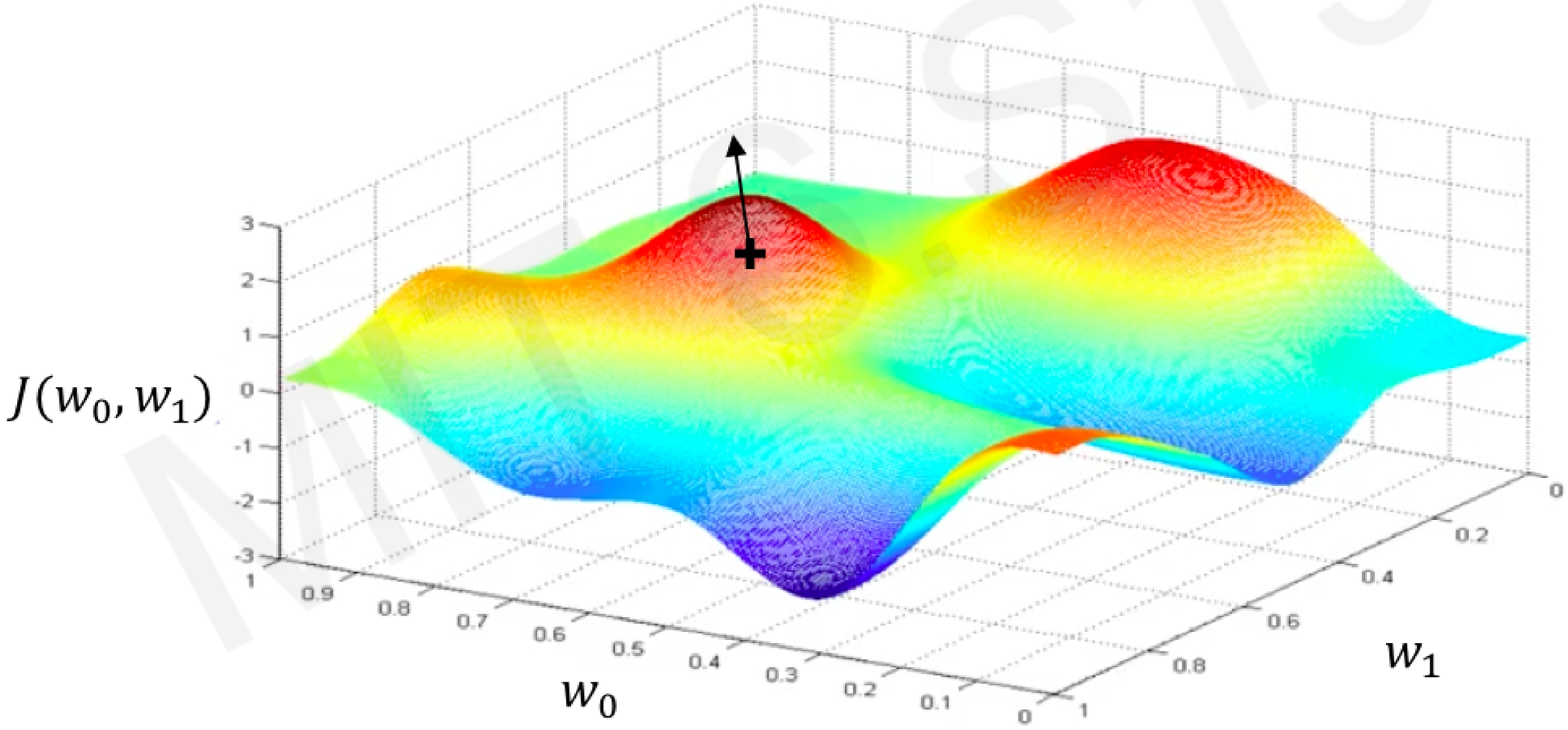
Randomly pick an initial (w_0, w_1)



MIT 6.S191

Loss Optimization

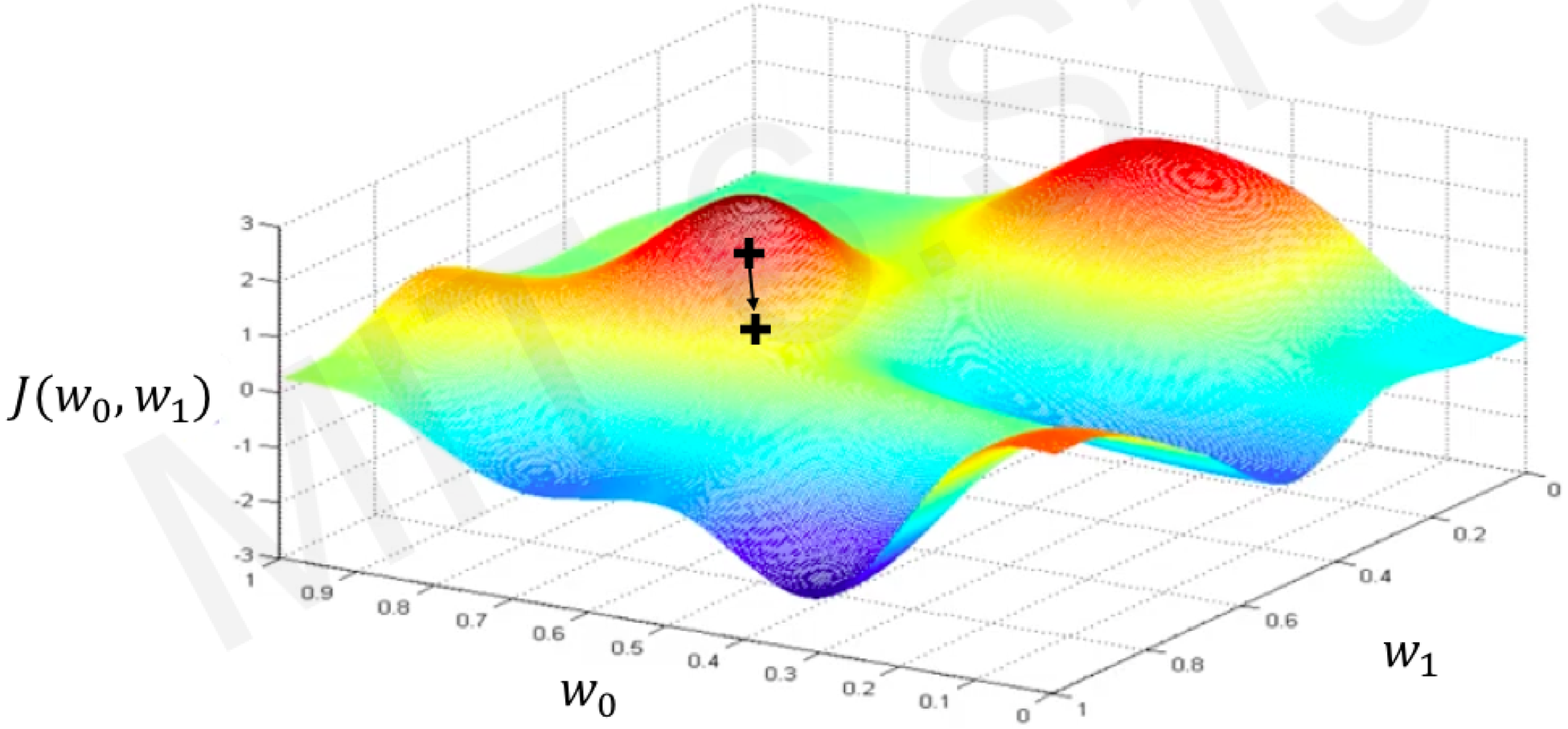
Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$



MIT 6.S191

Loss Optimization

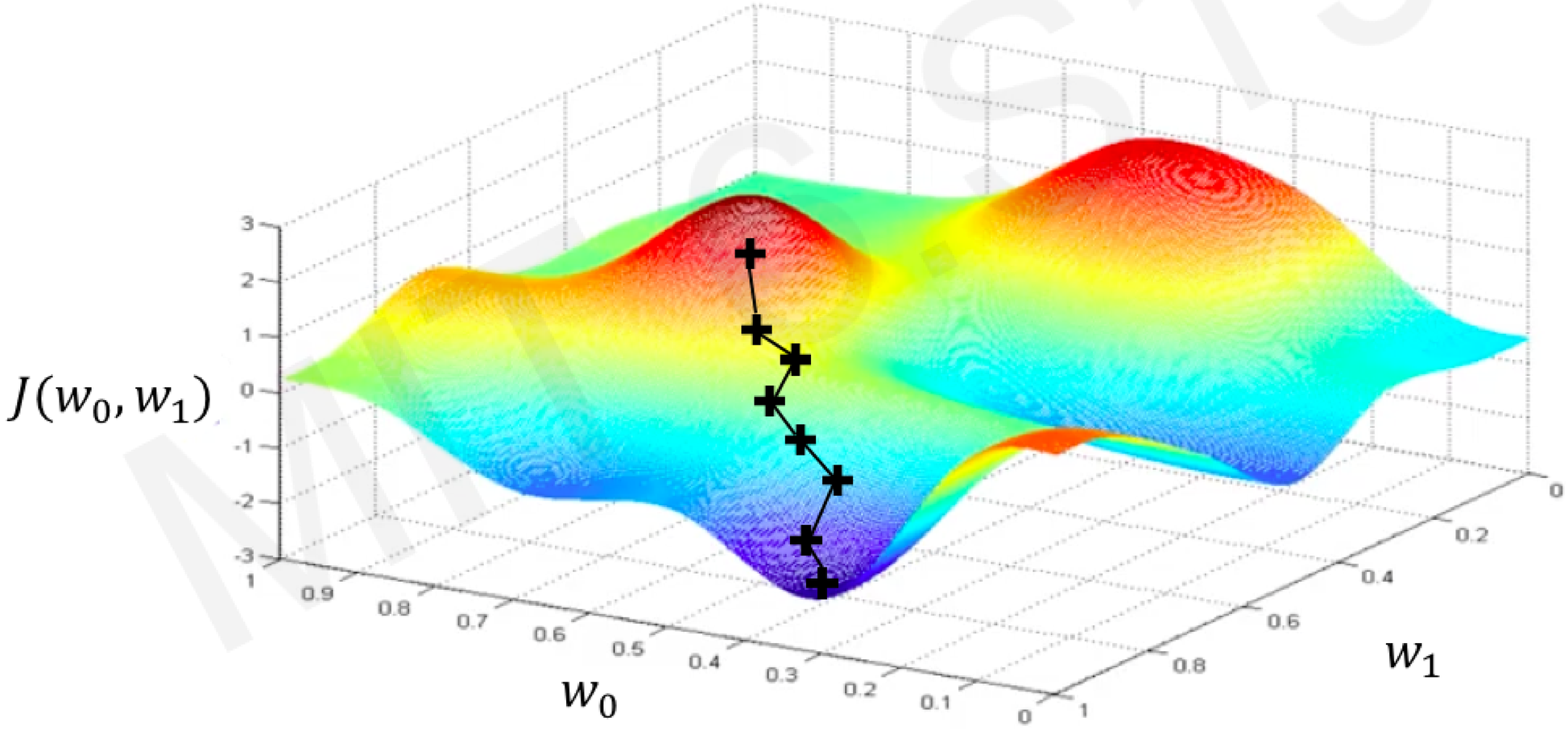
Take small step in opposite direction of gradient



MIT 6.S191

Gradient Descent

Repeat until convergence



MIT 6.S191

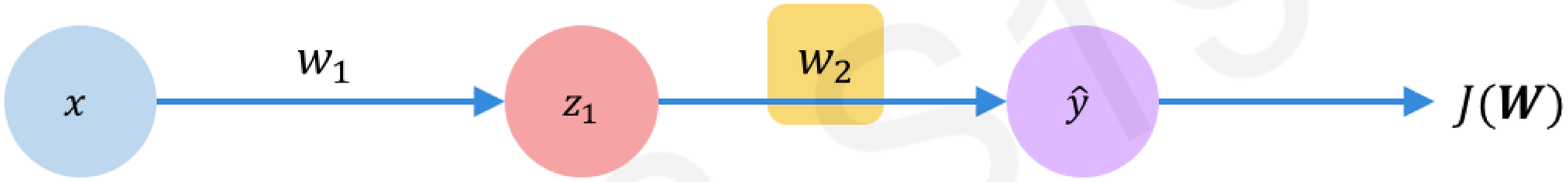
Gradient Descent

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

```
import tensorflow as tf

weights = tf.Variable([tf.random.normal()])

while True:   # loop forever
    with tf.GradientTape() as g:
        loss = compute_loss(weights)
        gradient = g.gradient(loss, weights)

    weights = weights - lr * gradient
```

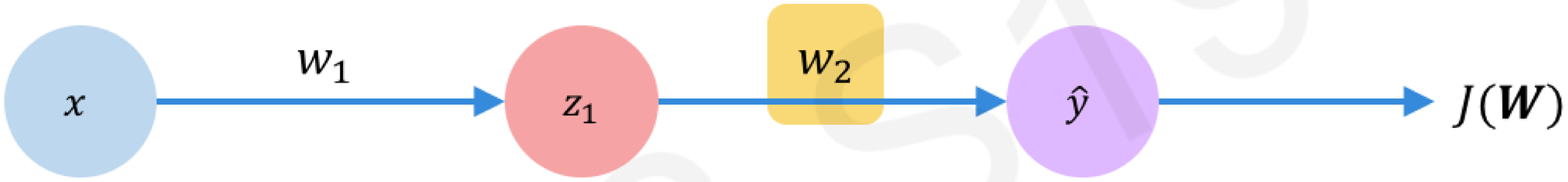
Computing Gradients: Backpropagation



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

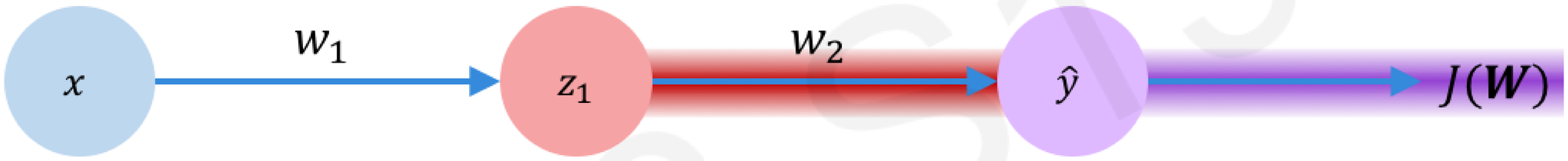


$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} =$$

Let's use the chain rule!

MIT 6.S191

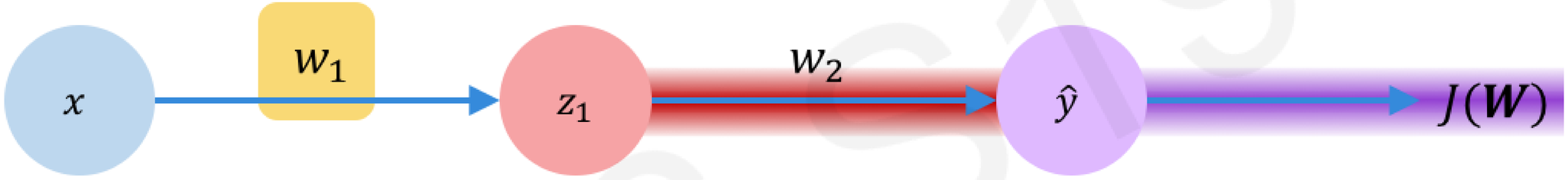
Computing Gradients: Backpropagation



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

MIT 6.S191

Computing Gradients: Backpropagation



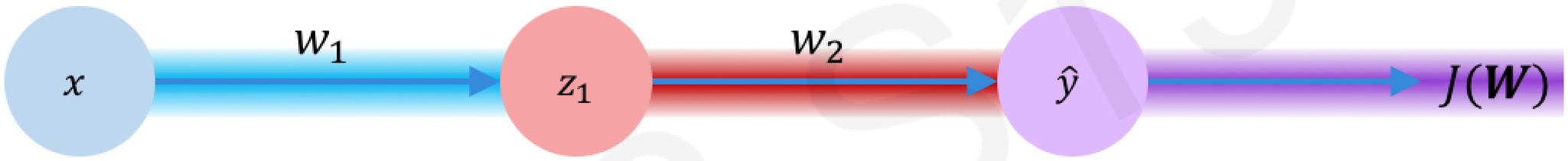
$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

MIT 6.S191

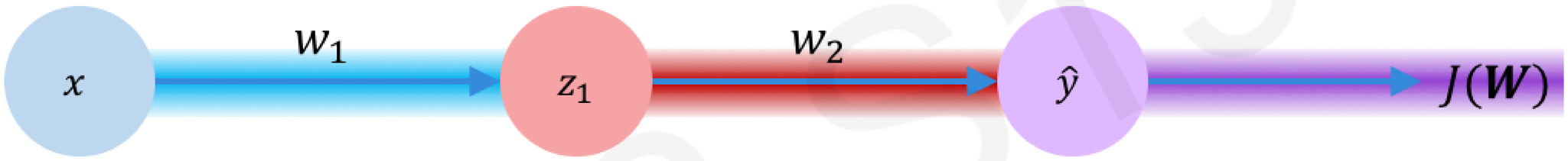
Computing Gradients: Backpropagation



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

MIT 6.S191

Computing Gradients: Backpropagation



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

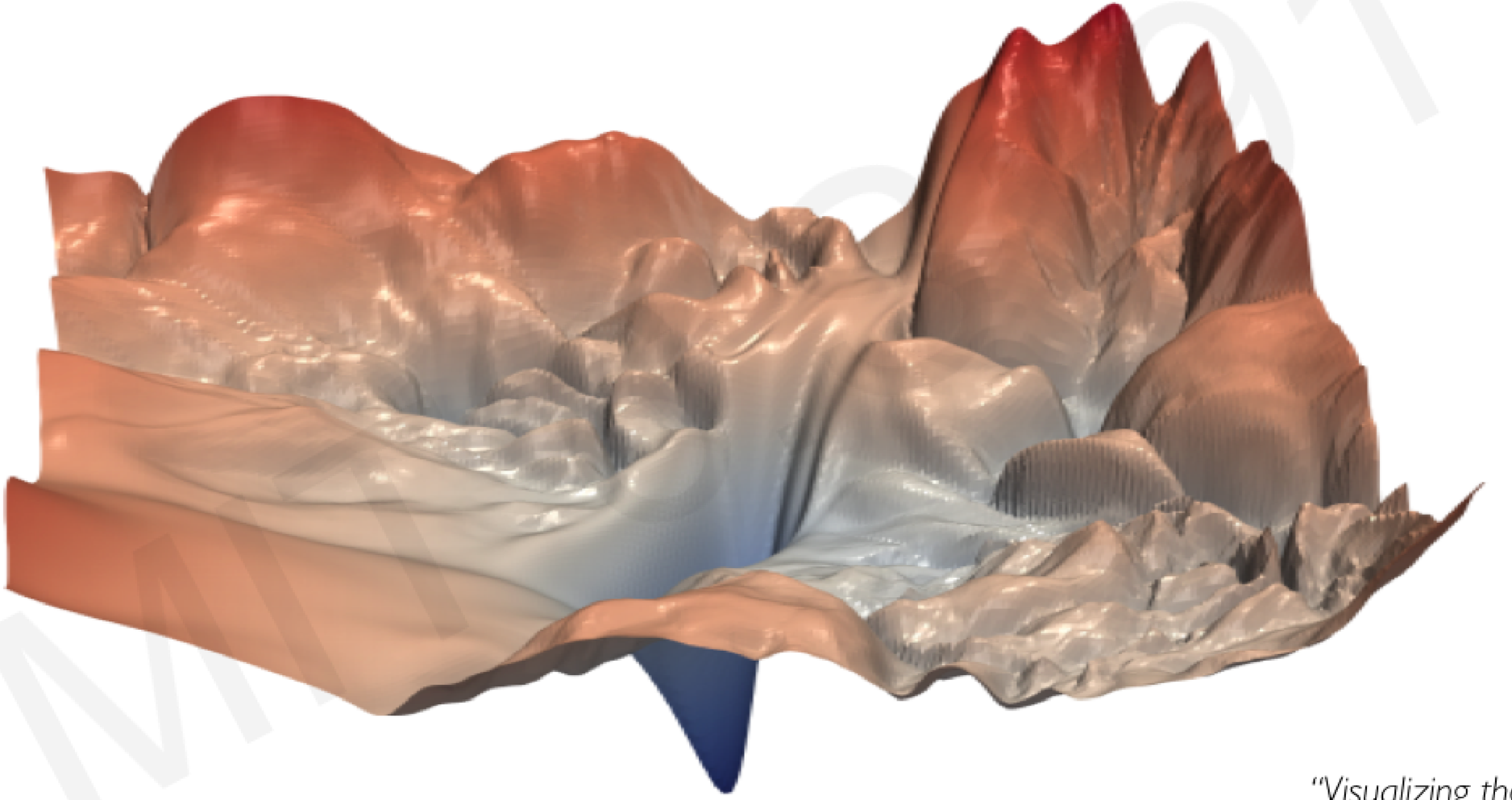
Repeat this for ***every weight in the network*** *using gradients from later layers*

MIT 6.S191

Neural Networks in Practice: Optimization

MIT 6.S191

Training Neural Networks is Difficult



"Visualizing the loss landscape of neural nets". Dec 2017.

Loss Functions Can Be Difficult to Optimize

Remember:

Optimization through gradient descent

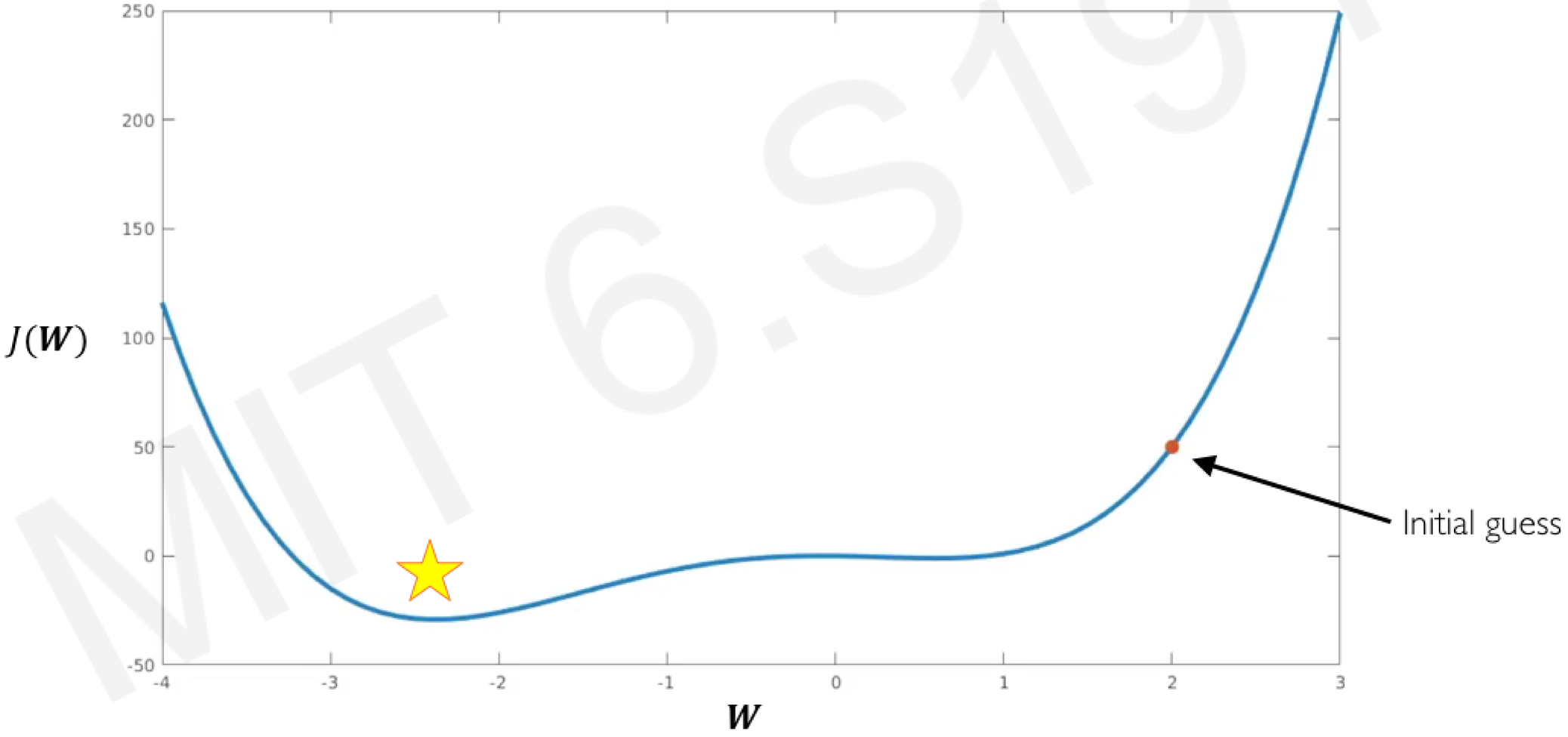
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \boxed{\eta} \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

MIT 6.S191

Setting the Learning Rate

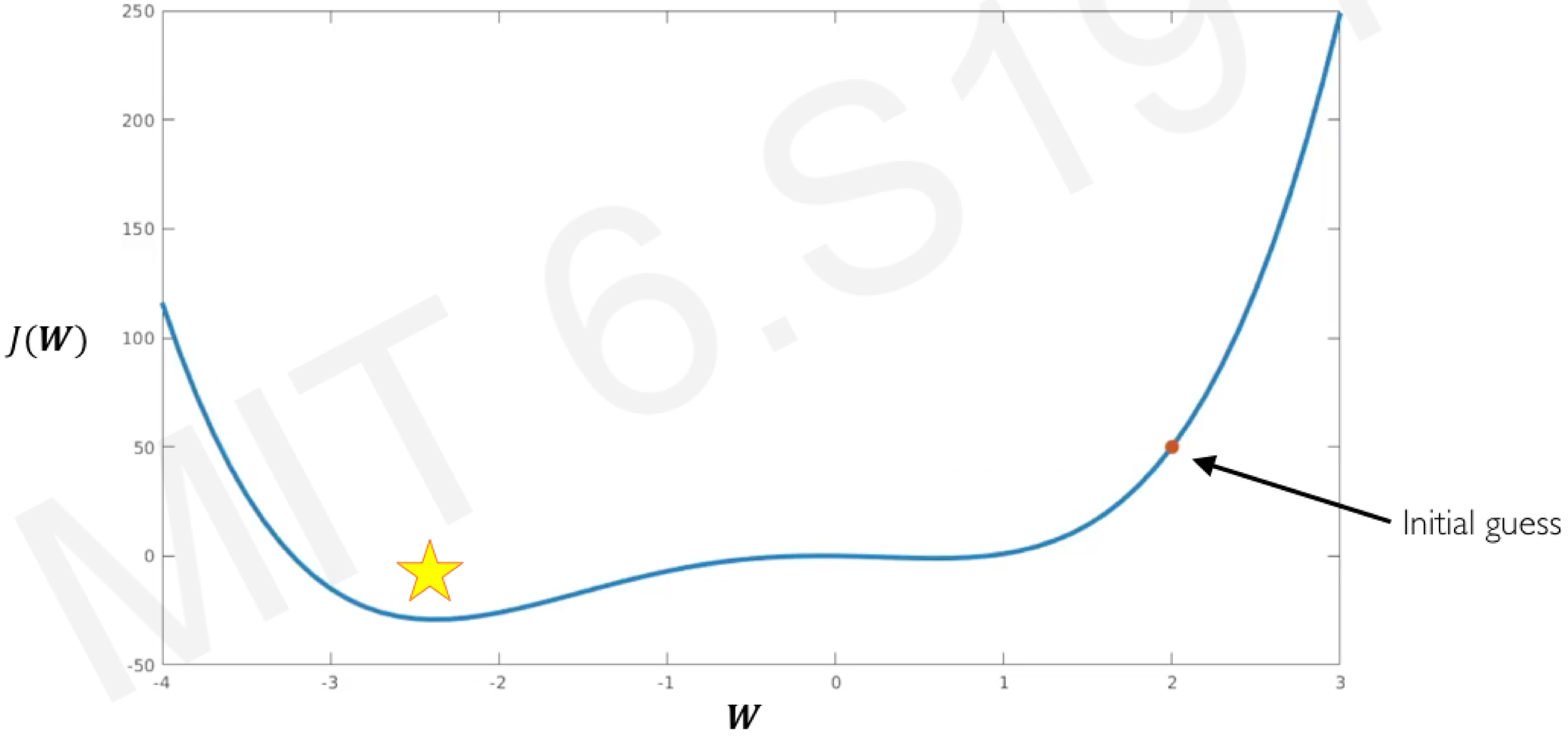
Small learning rate *converges slowly and gets stuck in false local minima*



MIT 6.S191

Setting the Learning Rate

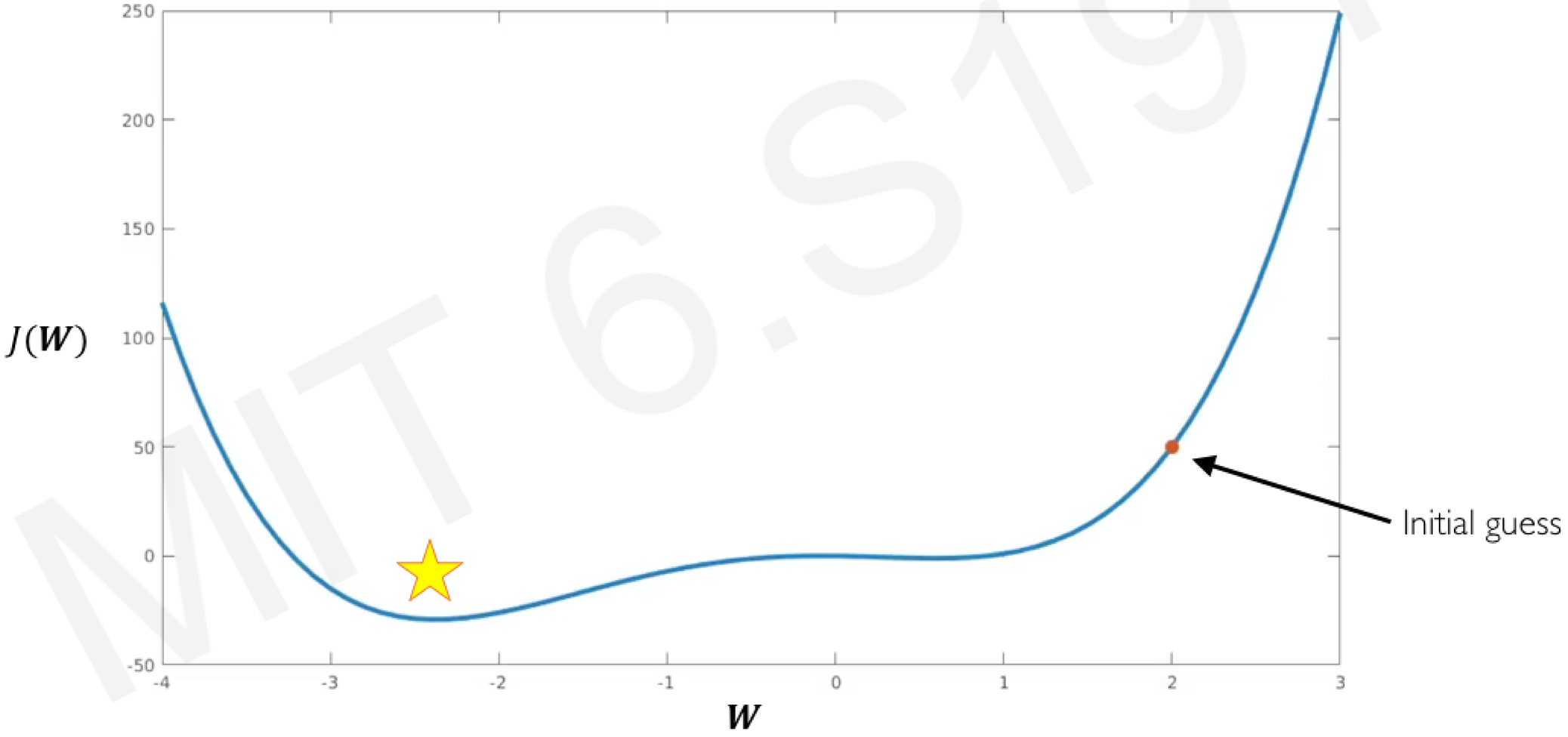
Large learning rates overshoot, become unstable and diverge



MIT 6.S191

Setting the Learning Rate

Stable learning rates converge smoothly and avoid local minima



MIT 6.S191

How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

Idea 2:

Do something smarter!

Design an adaptive learning rate that "adapts" to the landscape

MIT 6.S191

Adaptive Learning Rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc...

MIT 6.S191

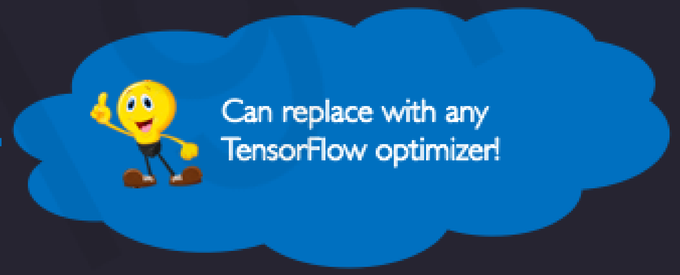
Gradient Descent Algorithms

Algorithm	TF Implementation	Reference
• SGD	`tf.keras.optimizers.SGD`	Kiefer & Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function." 1952.
• Adam	`tf.keras.optimizers.Adam`	Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
• Adadelta	`tf.keras.optimizers.Adadelta`	Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
• Adagrad	`tf.keras.optimizers.Adagrad`	Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
• RMSProp	`tf.keras.optimizers.RMSProp`	

Additional details: http://ruder.io/optimizing-gradient-descent/

Putting it all together

```
import tensorflow as tf

model = tf.keras.Sequential([...])

# pick your favorite optimizer
optimizer = tf.keras.optimizer.SGD()

while True: # loop forever

    # forward pass through the network
    prediction = model(x)

    with tf.GradientTape() as tape:
        # compute the loss
        loss = compute_loss(y, prediction)

    # update the weights using the gradient
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables)))
```



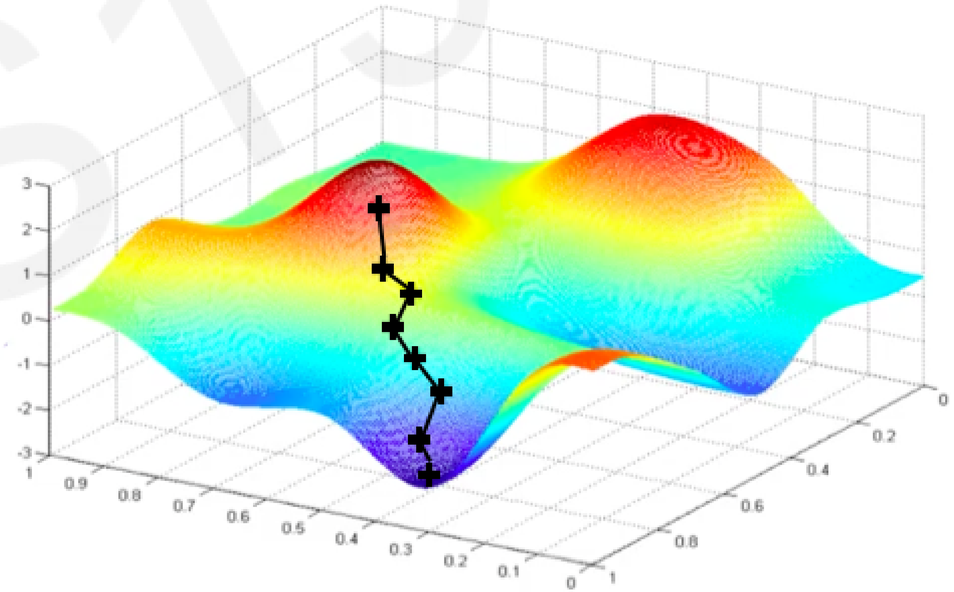
Neural Networks in Practice: Mini-batches

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



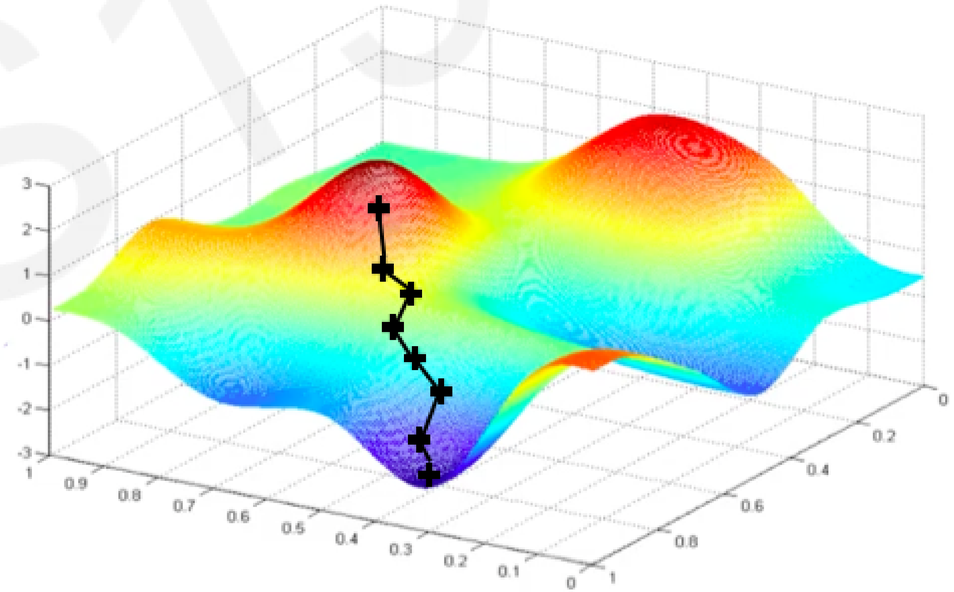
Can be very **computationally intensive** to compute!

MIT 6.S191

Stochastic Gradient Descent

Algorithm

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



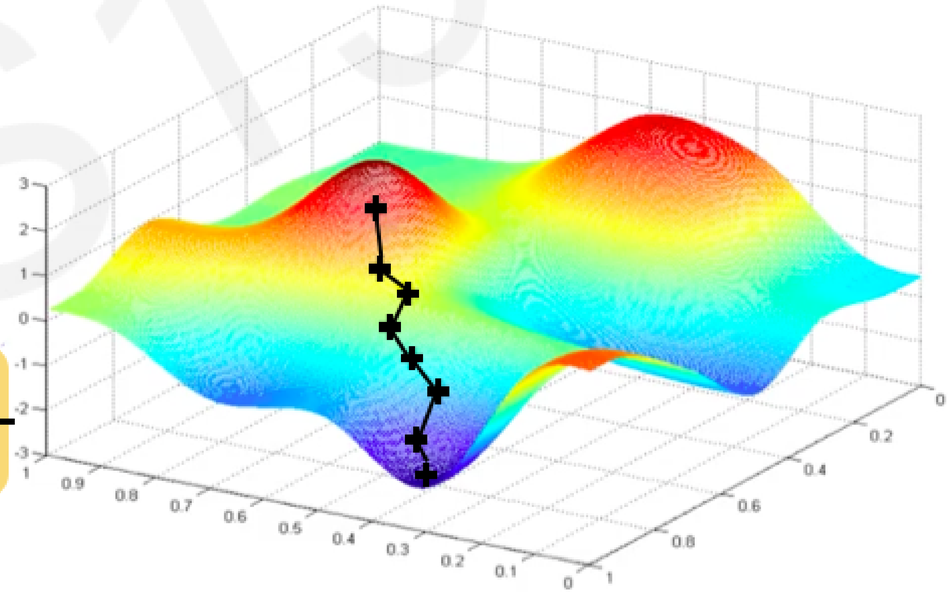
Easy to compute but
very noisy (stochastic)!

MIT 6.S191

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

MIT 6.S191

Mini-batches while training

More accurate estimation of gradient

Smoother convergence

Allows for larger learning rates

Mini-batches lead to fast training!

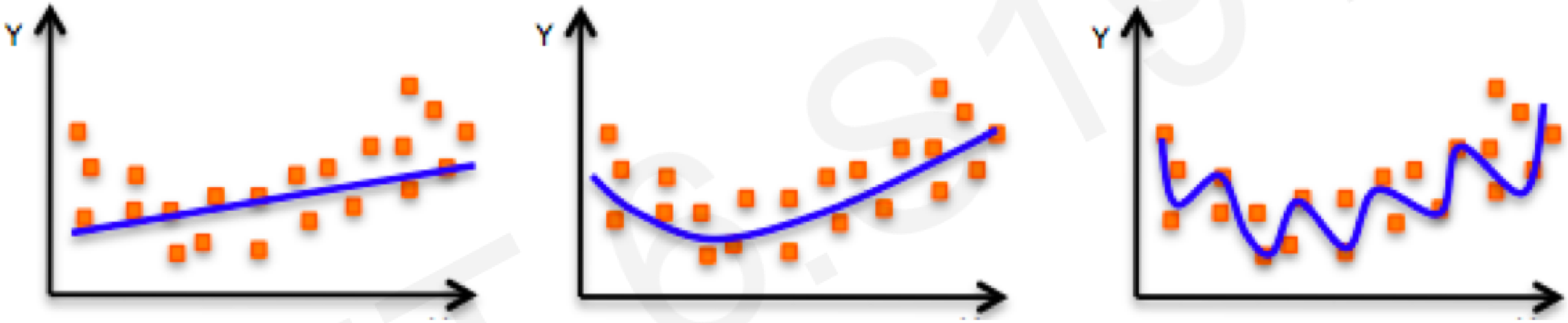
Can parallelize computation + achieve significant speed increases on GPU's

MIT 6.S191

Neural Networks in Practice: Overfitting

MIT 6.S191

The Problem of Overfitting



Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

MIT 6.S191

Regularization

What is it?

Technique that constrains our optimization problem to discourage complex models

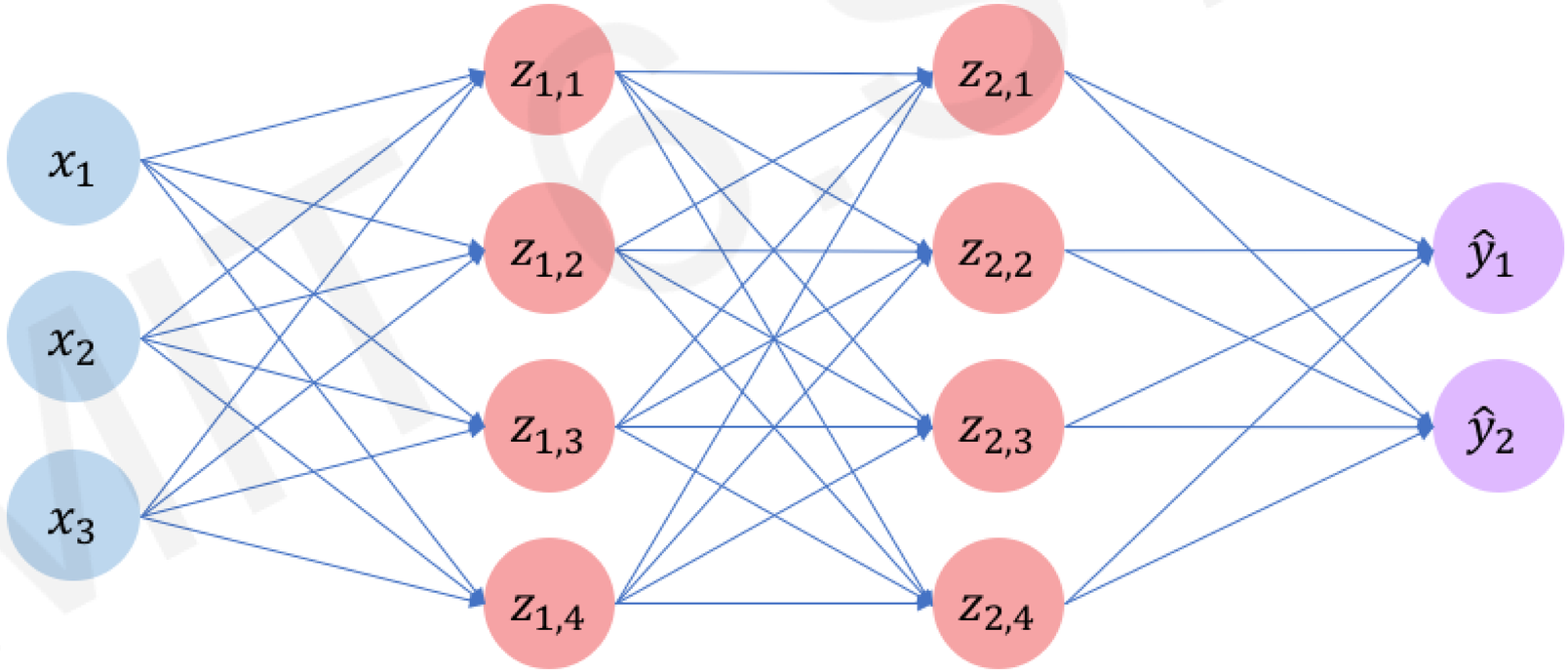
Why do we need it?

Improve generalization of our model on unseen data

MIT 6.S191

Regularization 1: Dropout

- During training, randomly set some activations to 0

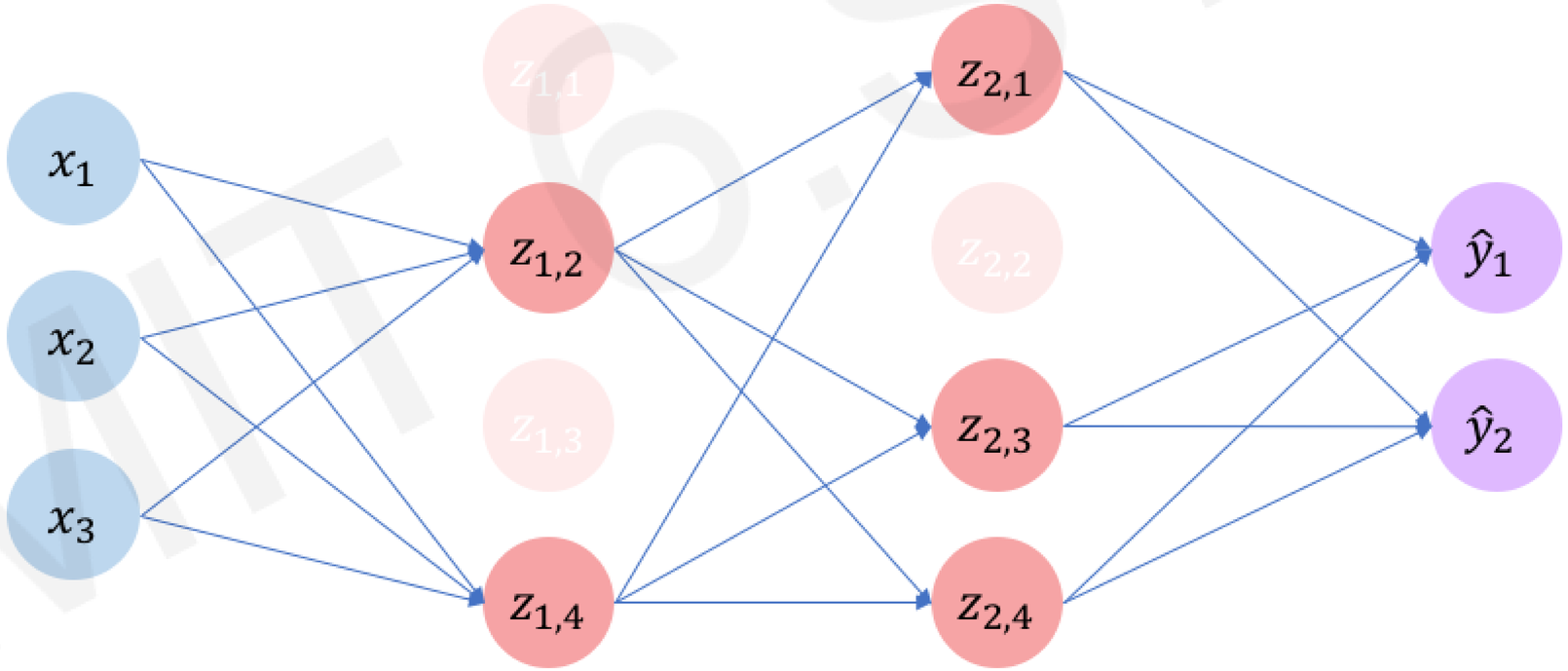


MIT 6.S191

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

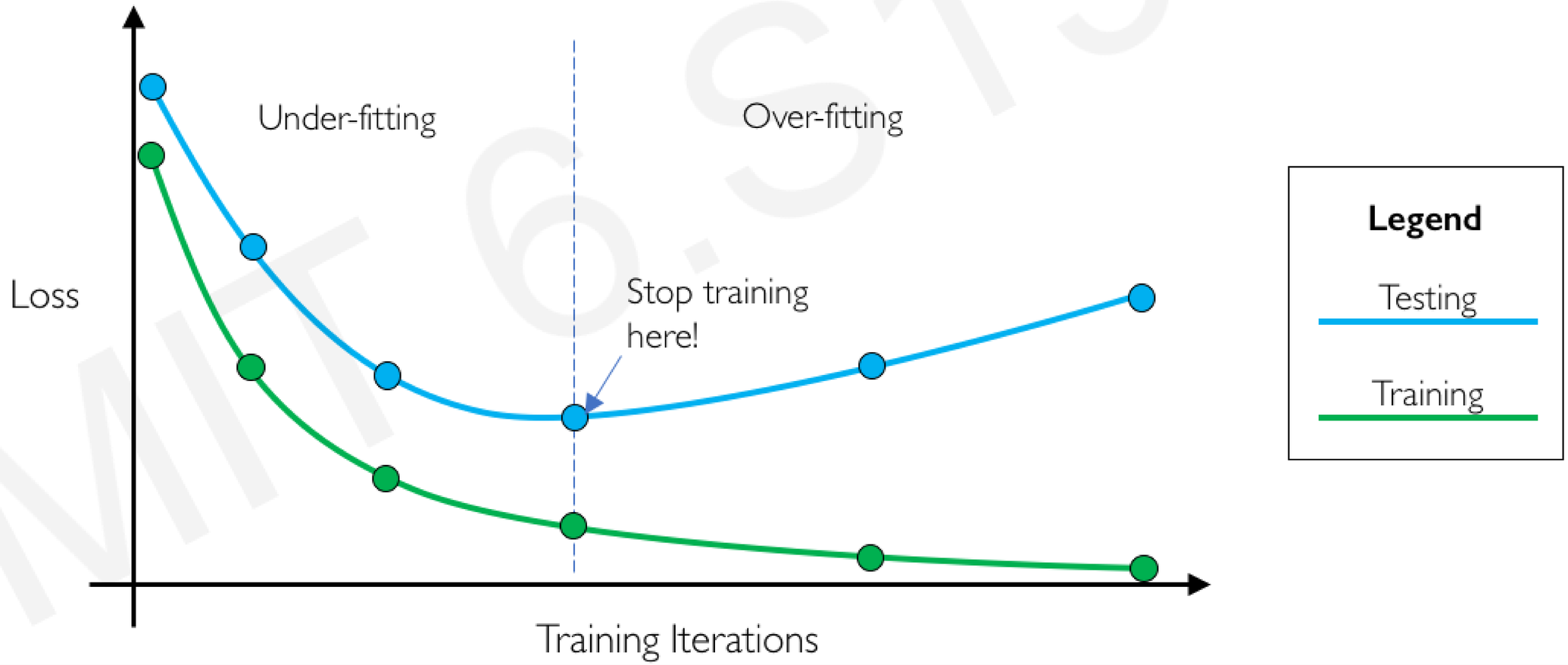
```
tf.keras.layers.Dropout(p=0.5)
```



MIT 6.S191

Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

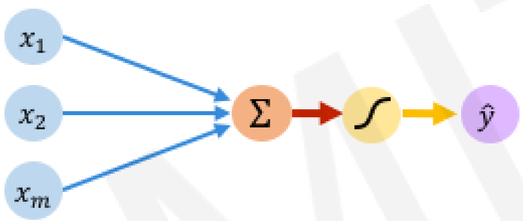


MIT 6.S191

Core Foundation Review

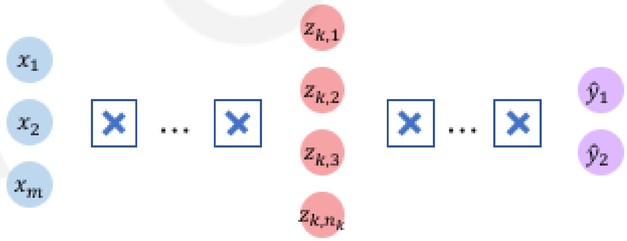
The Perceptron

- Structural building blocks
- Nonlinear activation functions



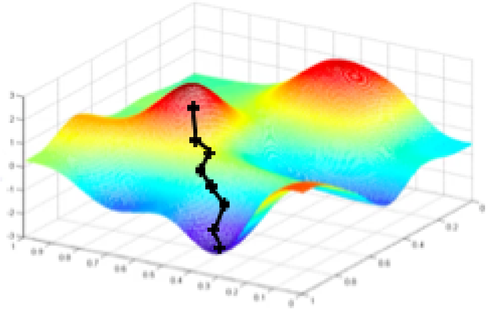
Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



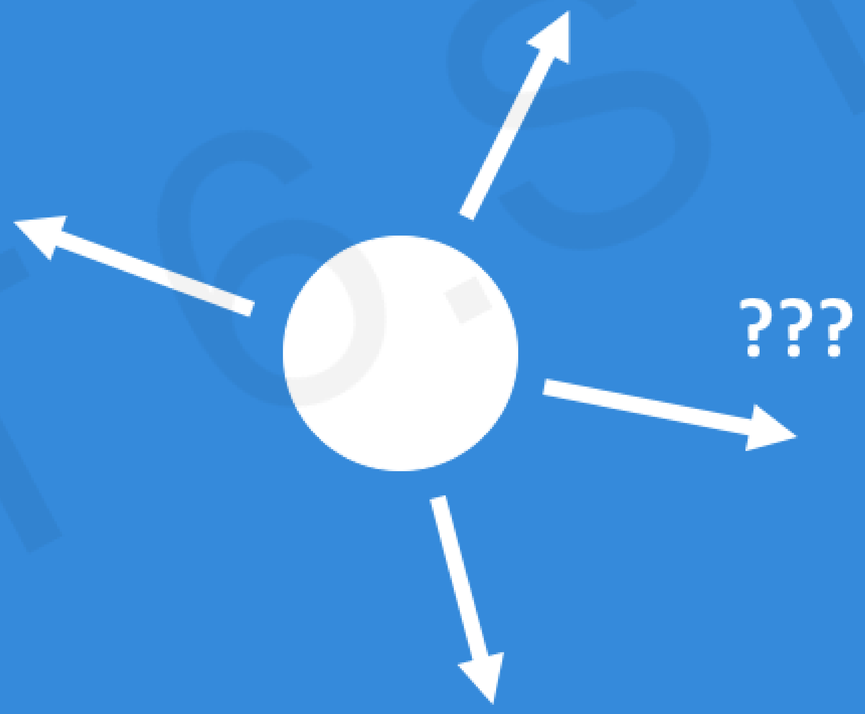
Training in Practice

- Adaptive learning
- Batching
- Regularization



MIT 6.S191

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



MIT 6.819

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

character:

6.S191 Introduction to Deep Learning

word:

Text

MIT 6.S191

Idea #1: Use a Fixed Window

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

given these two words

predict the next word

One-hot feature encoding: tells us what each word is

[1 0 0 0 0 0 1 0 0 0]

for

a

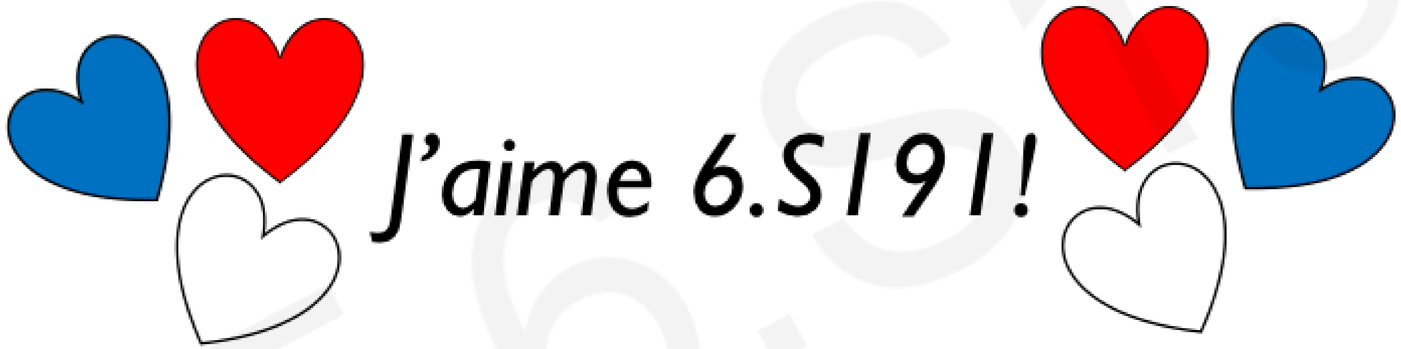
↓

prediction

MIT 6.S191

Problem #1: Can't 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

Idea #2: Use Entire Sequence as Set of Counts

"This morning I took my cat for a"

↓

"bag of words"

[0 1 0 0 1 0 0 ... 0 0 1 1 0 0 0 1]

↓

prediction

MIT 6.S191

Problem #2: Counts Don't Preserve Order

The food was good, not bad at all.

vs.

The food was bad, not good at all.

MIT 6.S191

Idea #3: Use a Really Big Fixed Window

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

given these words

predict the next word

[1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]

morning I took this cat

prediction

MIT 6.S191

Problem #3: No Parameter Sharing

[1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]

this morning took the cat

Each of these inputs has a **separate parameter**:

[0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 ...]

this morning

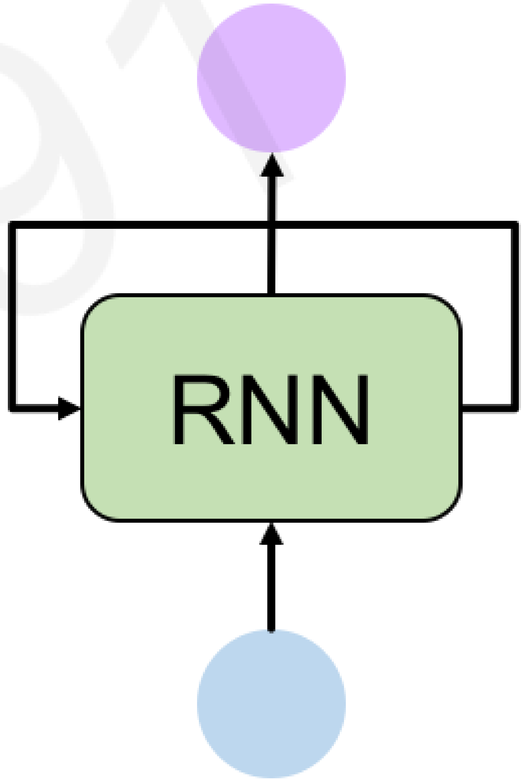
Things we learn about the sequence **won't transfer** if they appear **elsewhere** in the sequence.

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



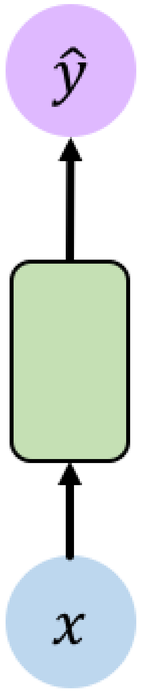
Today: **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems

MIT 6.S191

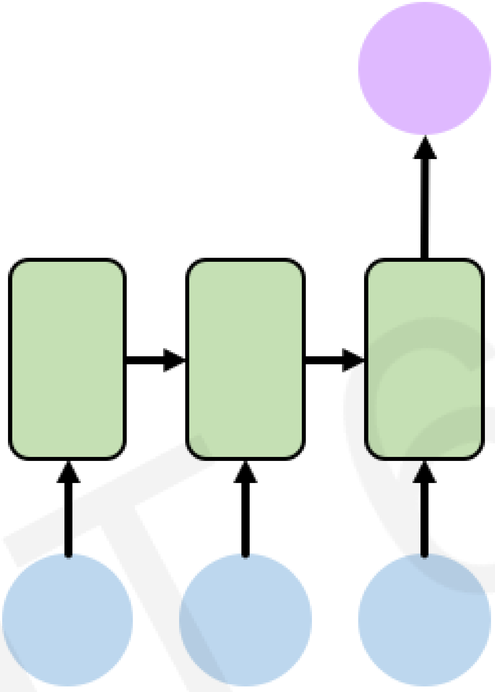
Recurrent Neural Networks (RNNs)

MIT 6.S191

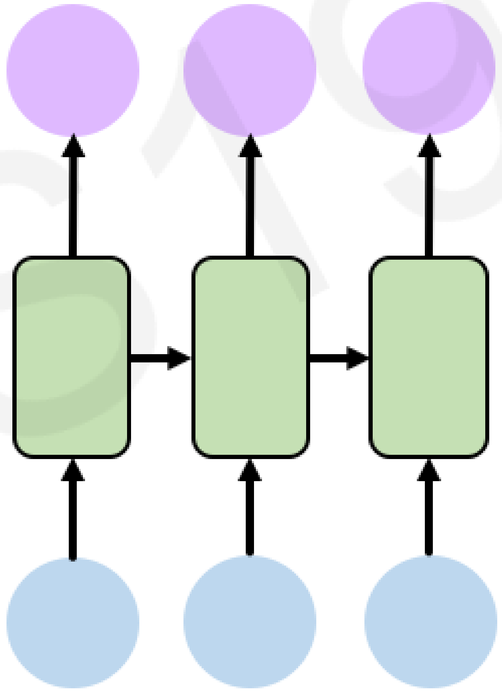
Recurrent Neural Networks for Sequence Modeling



One to One
"Vanilla" neural network



Many to One
Sentiment Classification



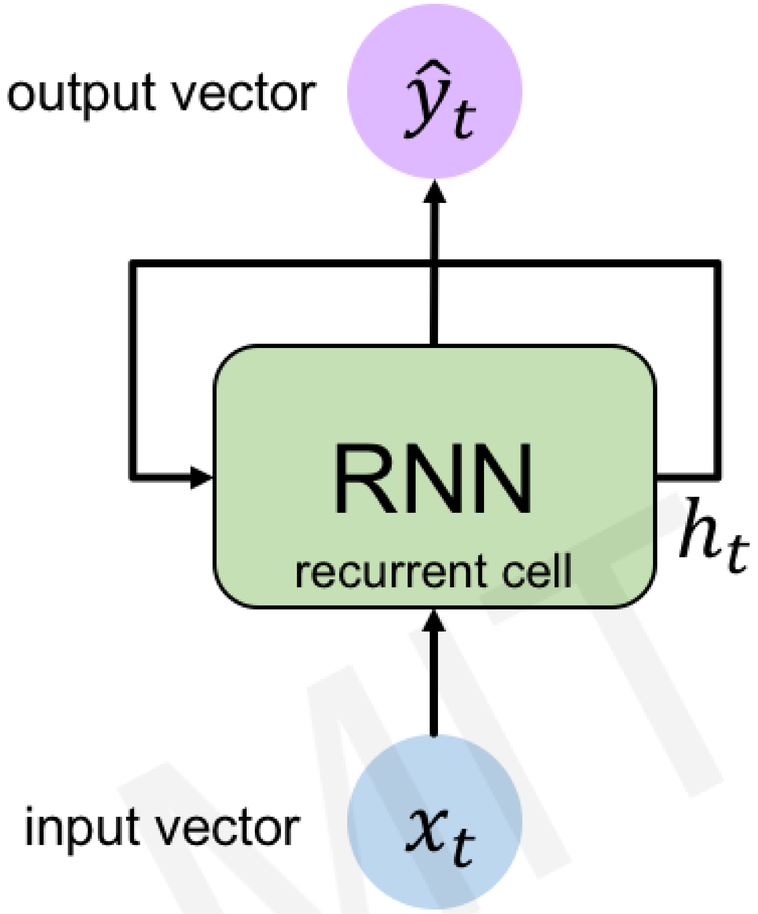
Many to Many
Music Generation

6.S191 Lab!

… and many other architectures and applications

MIT 6.S191

Recurrent Neural Network (RNN)



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

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

cell state

function parameterized by W

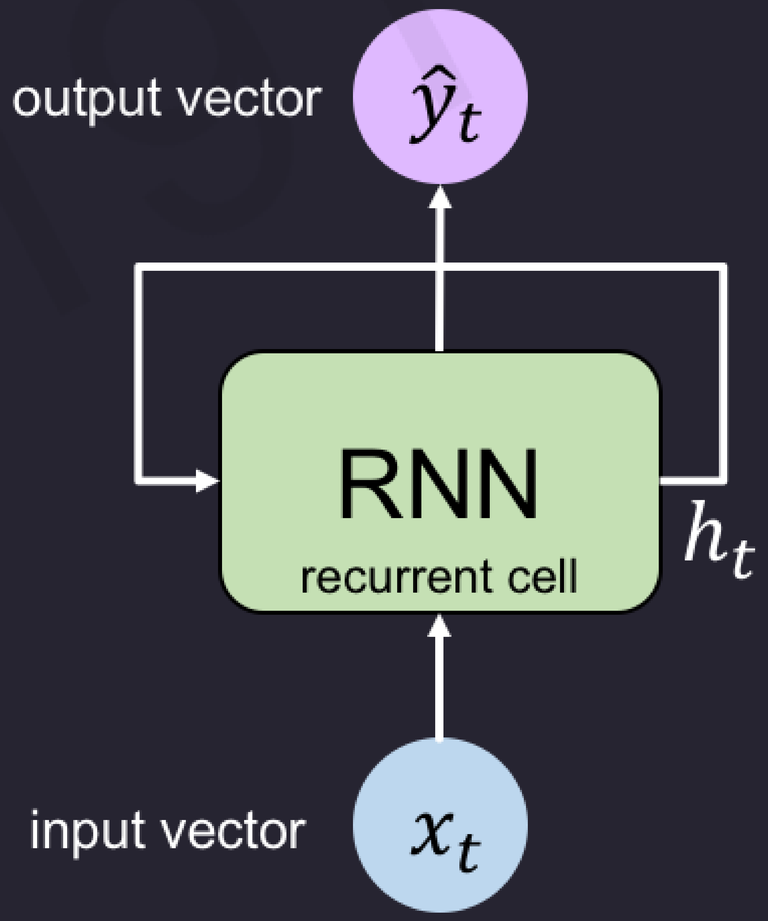
old state

input vector at time step *t*

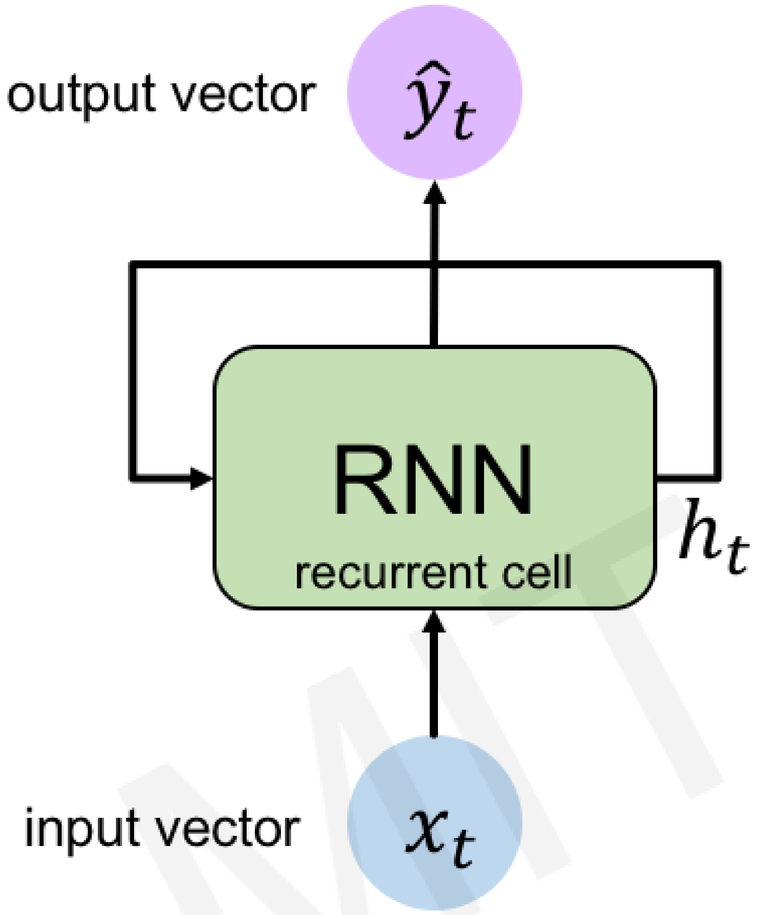
Note: the same function and set of parameters are used at every time step

RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```



RNN State Update and Output



Output Vector

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

Update Hidden State

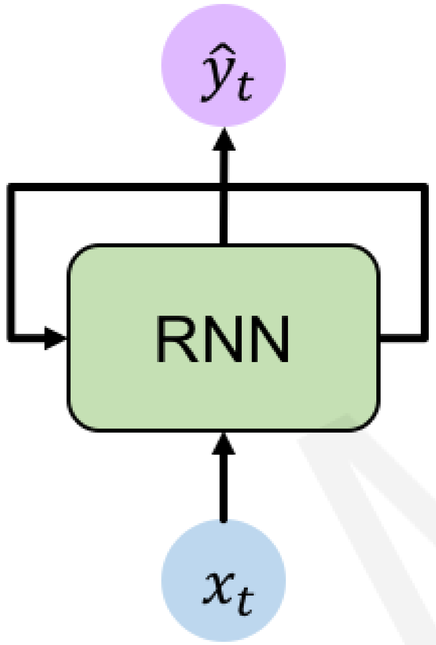
$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

Input Vector

$$x_t$$

MIT 6.S191

RNNs: Computational Graph Across Time



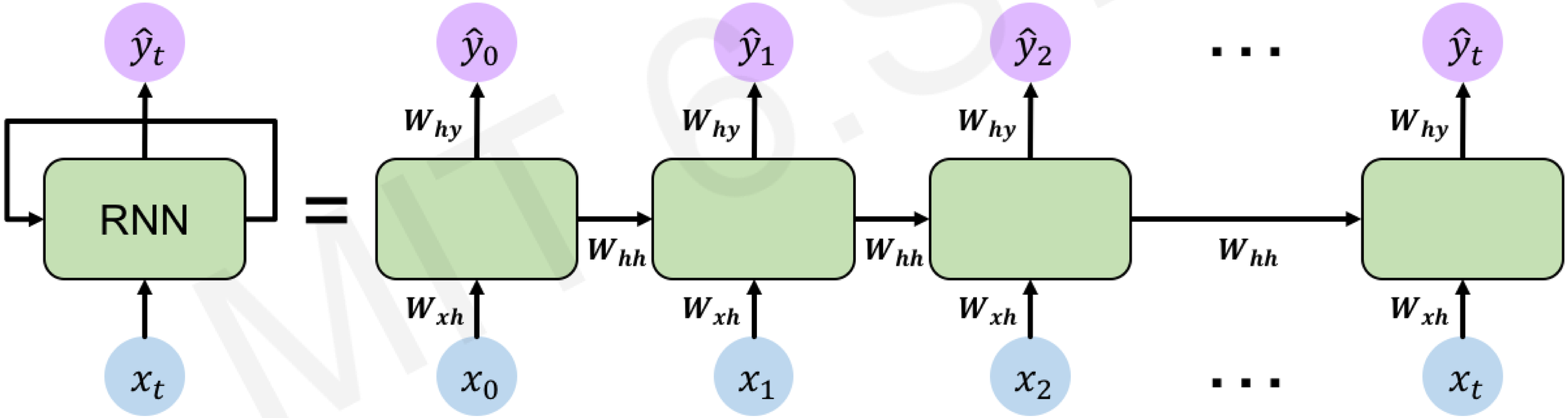
=

Represent as computational graph unrolled across time

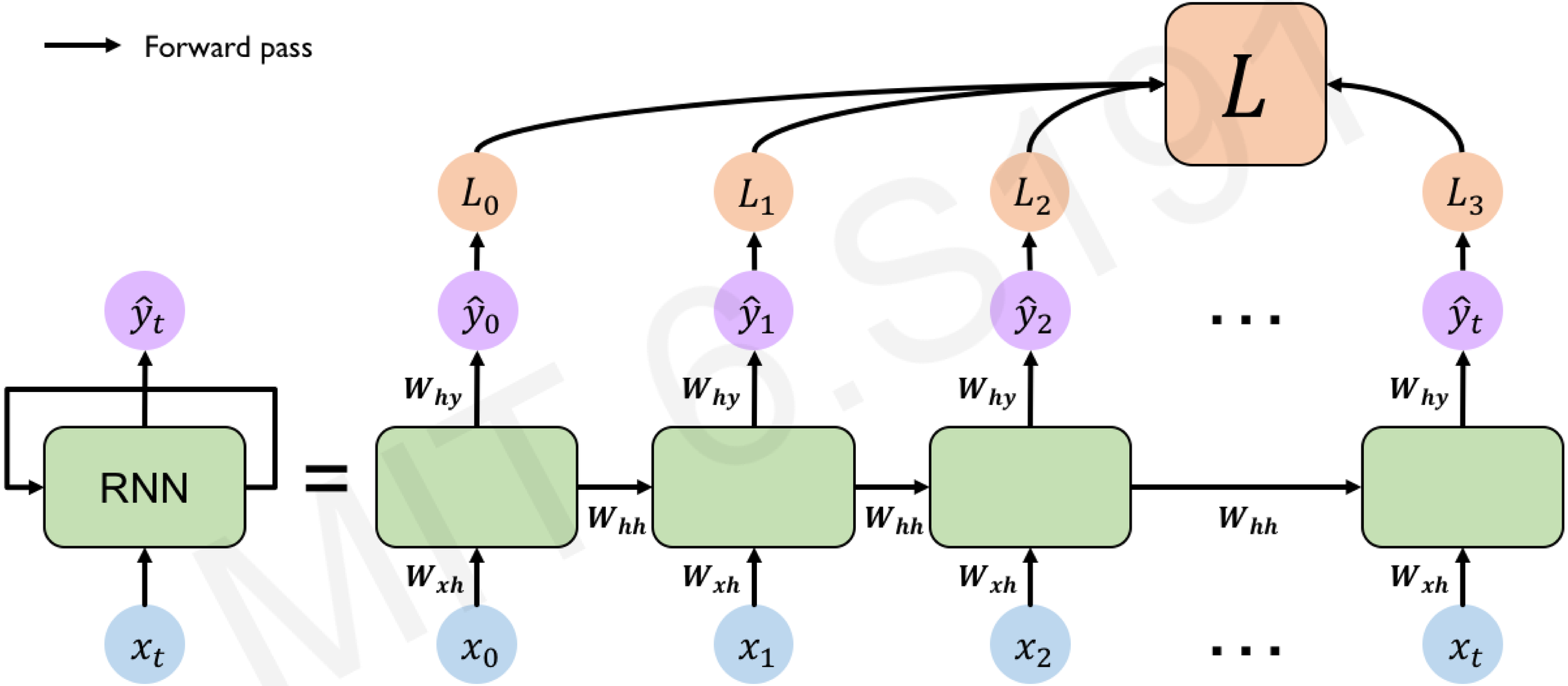
MIT 6.S191

RNNs: Computational Graph Across Time

Re-use the **same weight matrices** at every time step



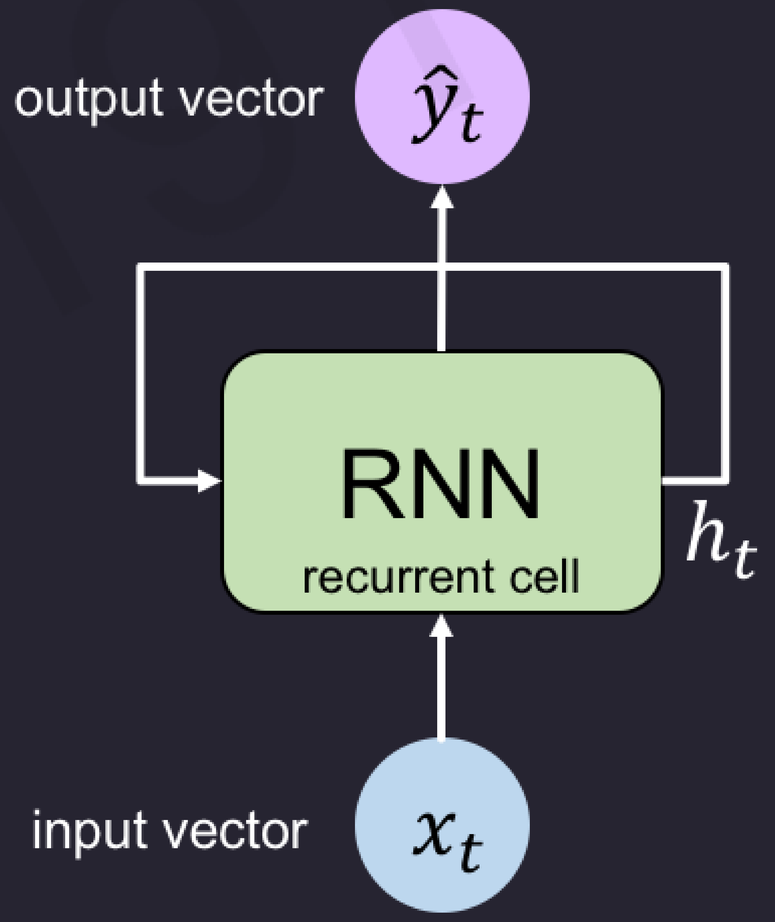
RNNs: Computational Graph Across Time



MIT 6.S191

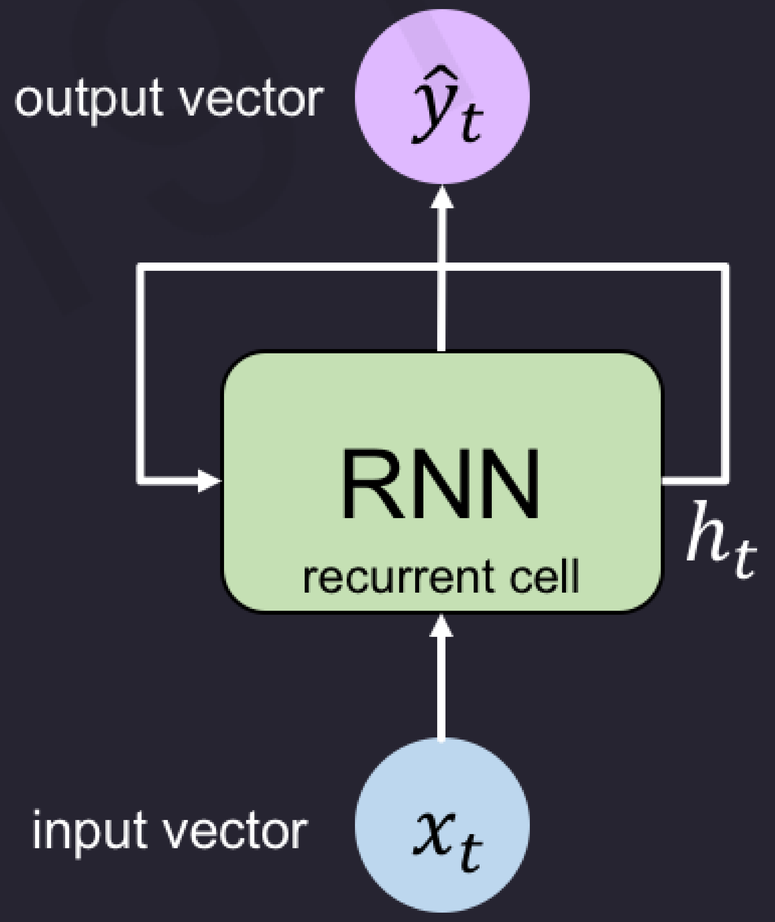
RNNs from Scratch

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

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



RNN Implementation in TensorFlow

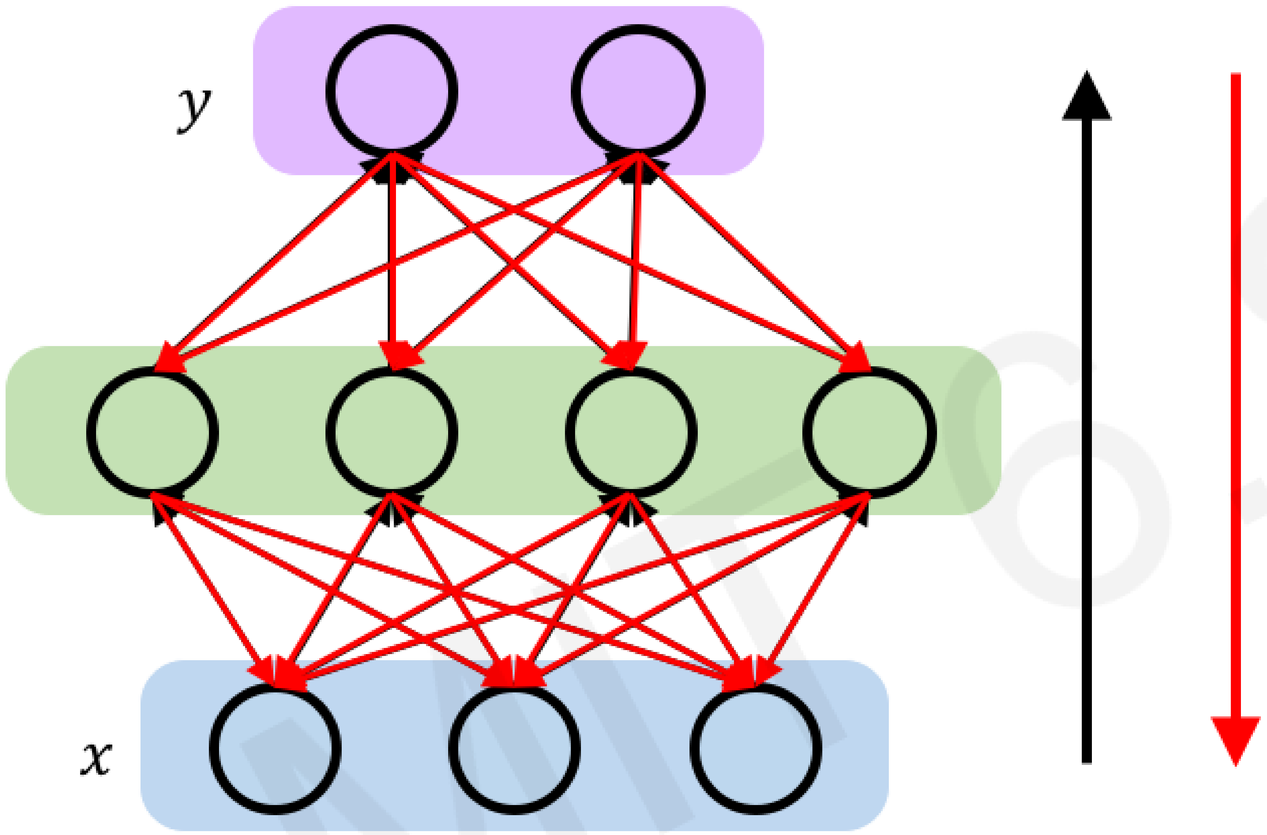
```
tf.keras.layers.SimpleRNN(rnn_units)
```



Backpropagation Through Time (BPTT)

MIT 6.S191

Recall: Backpropagation in Feed Forward Models

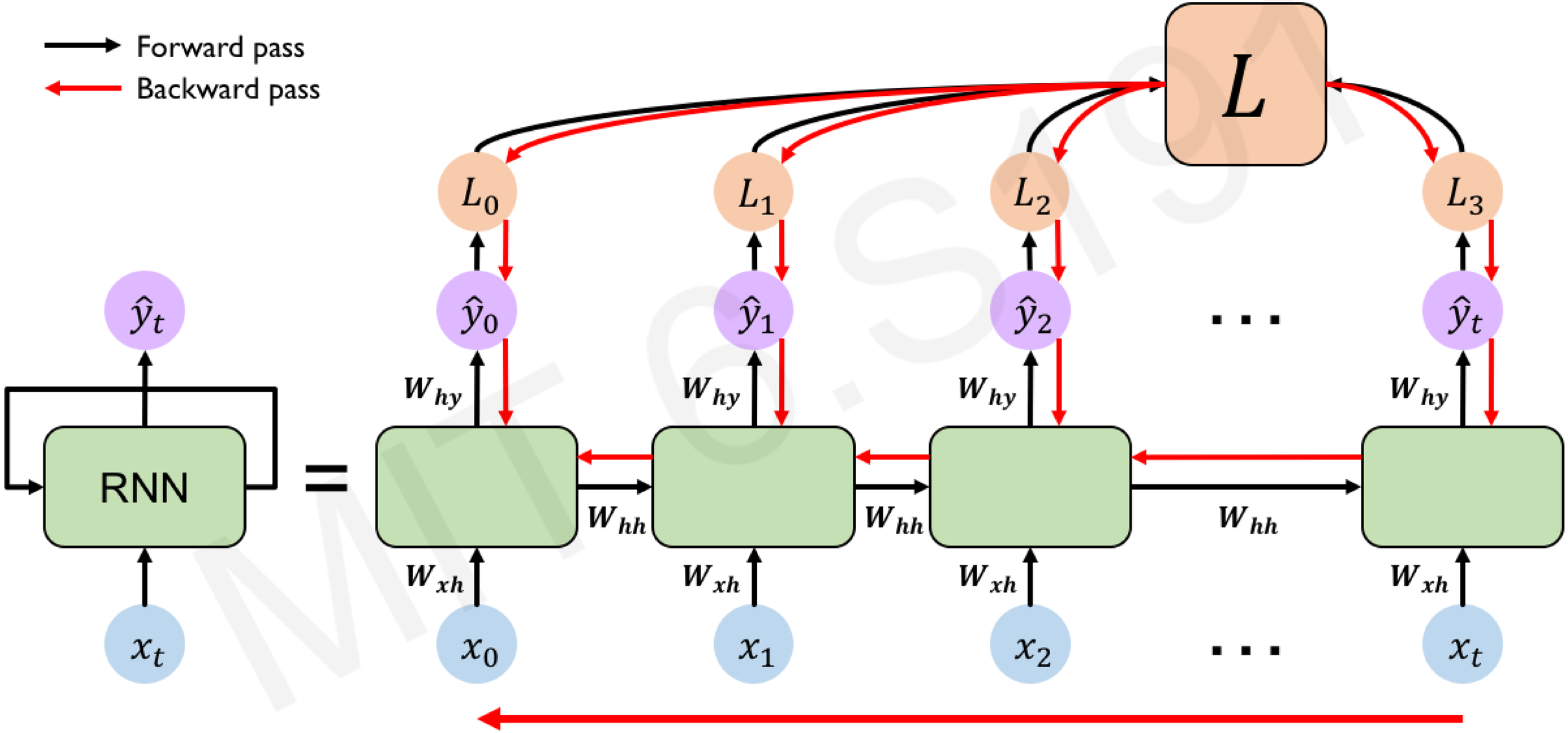


Backpropagation algorithm:

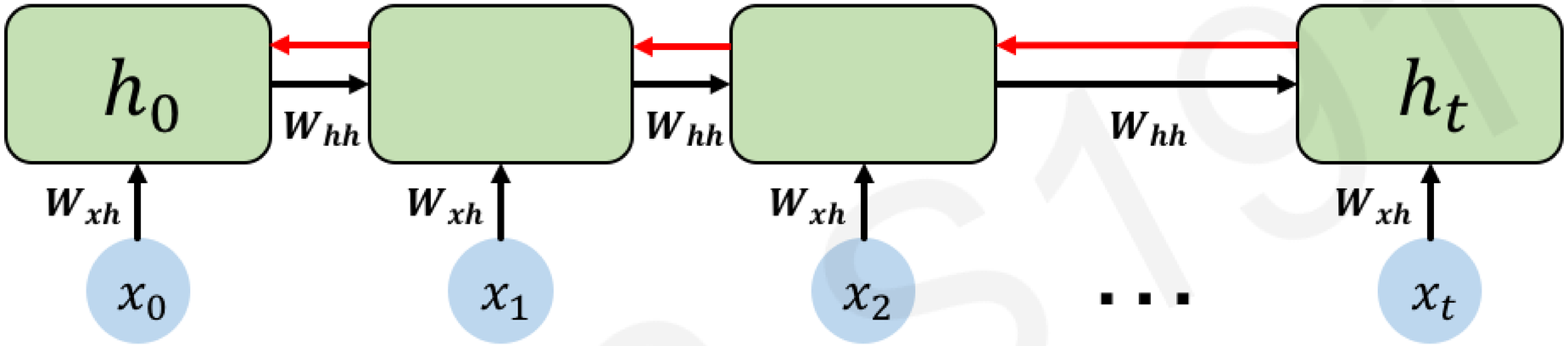
1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

MIT 6.S191

RNNs: Backpropagation Through Time



Standard RNN Gradient Flow: Exploding Gradients



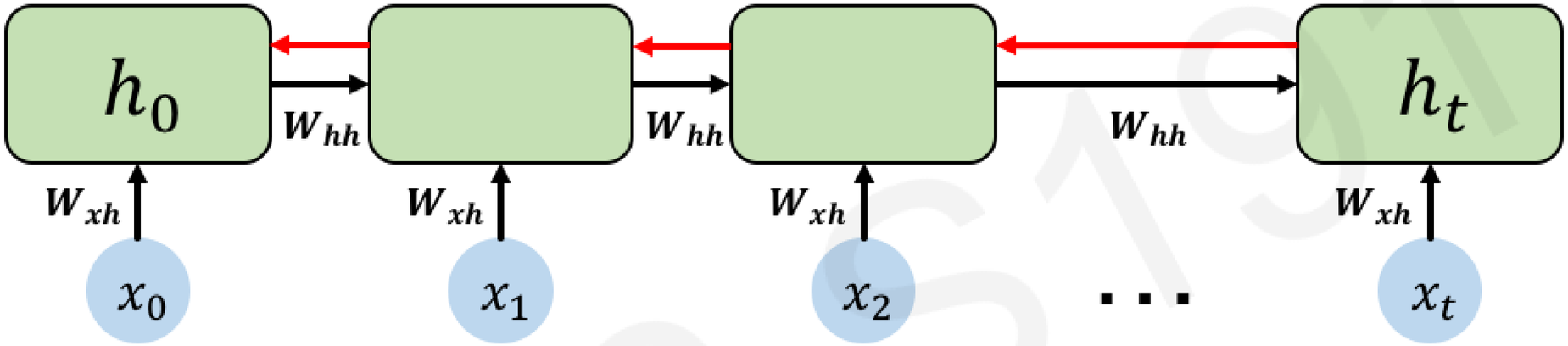
Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

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

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

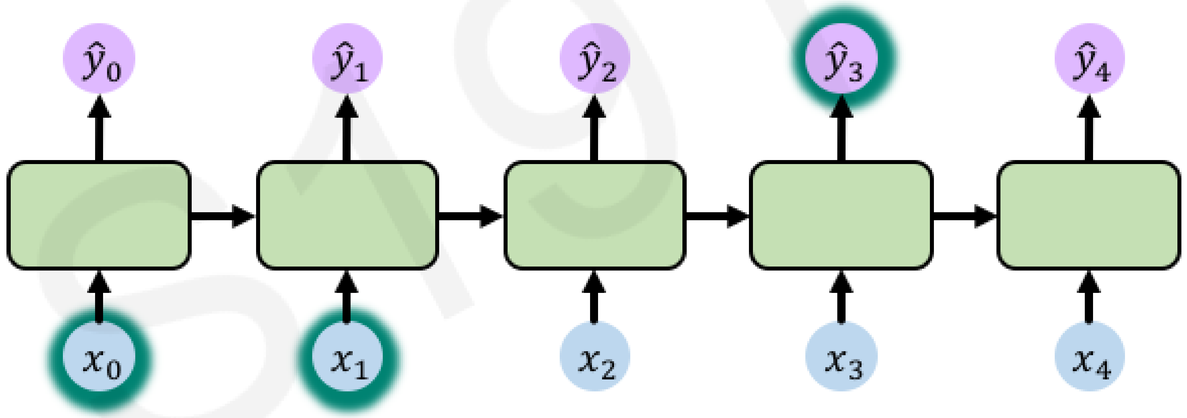
↓

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

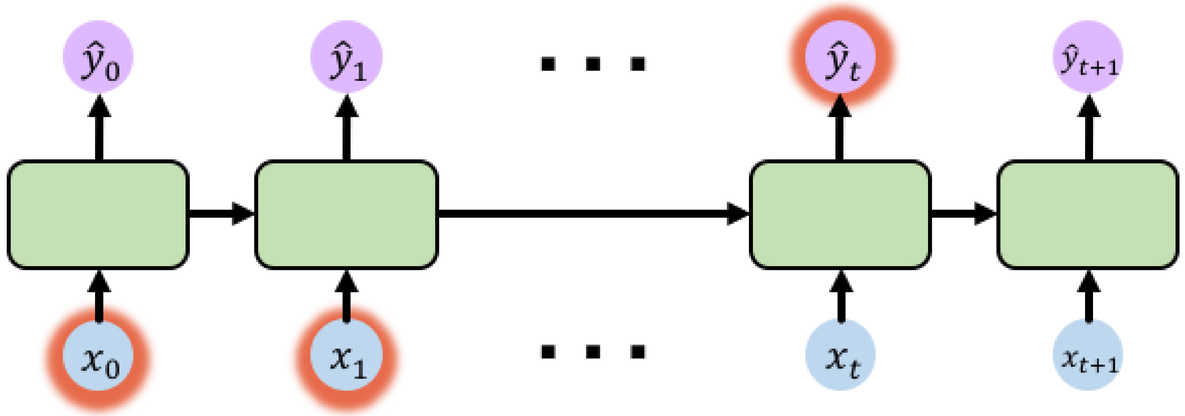
↓

Bias parameters to capture short-term dependencies

"The clouds are in the ___"

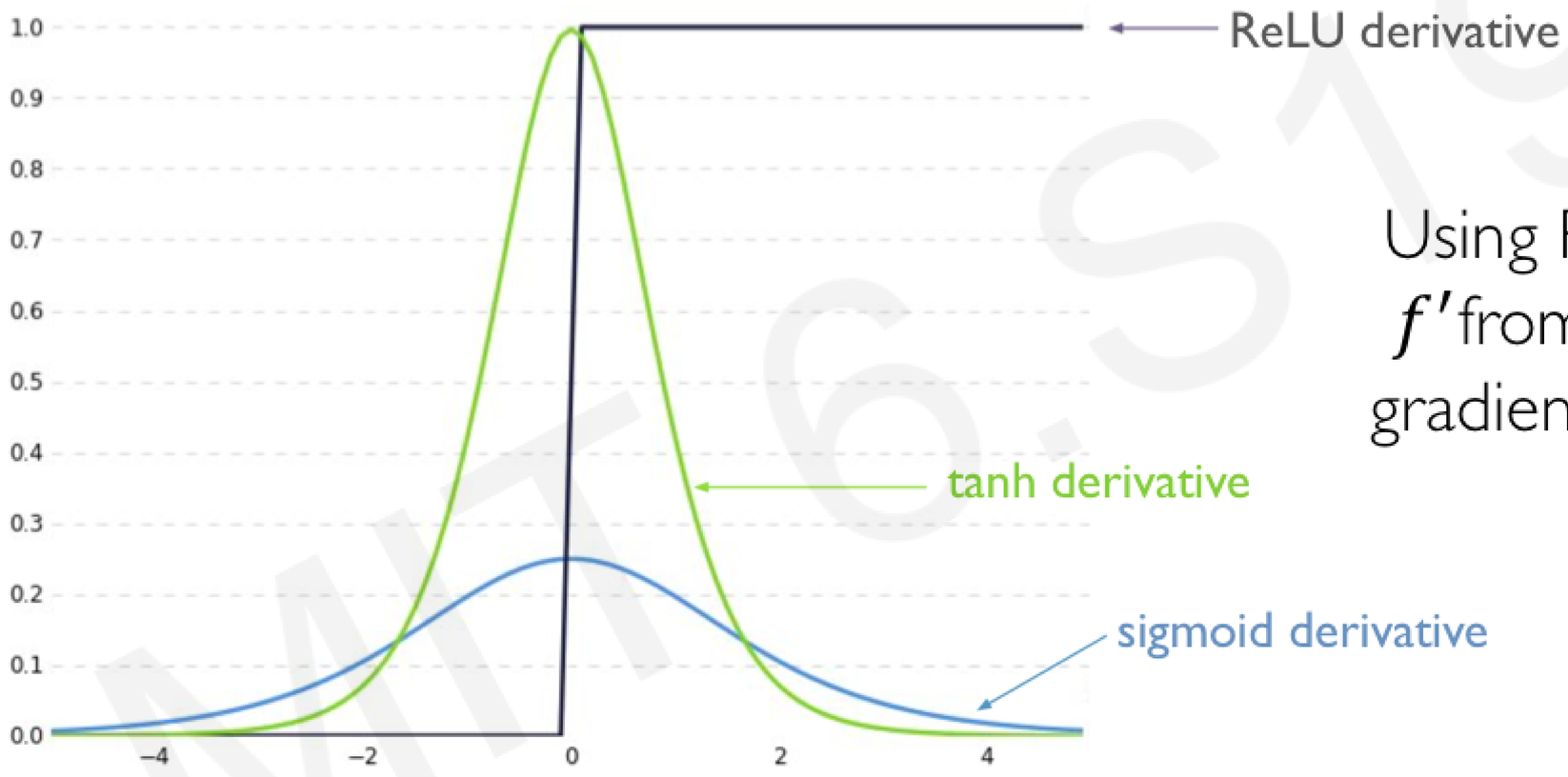


"I grew up in France, … and I speak fluent___ "



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

Solution #3: Gated Cells

Idea: use a more **complex recurrent unit with gates** to control what information is passed through

gated cell

LSTM, GRU, etc.

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

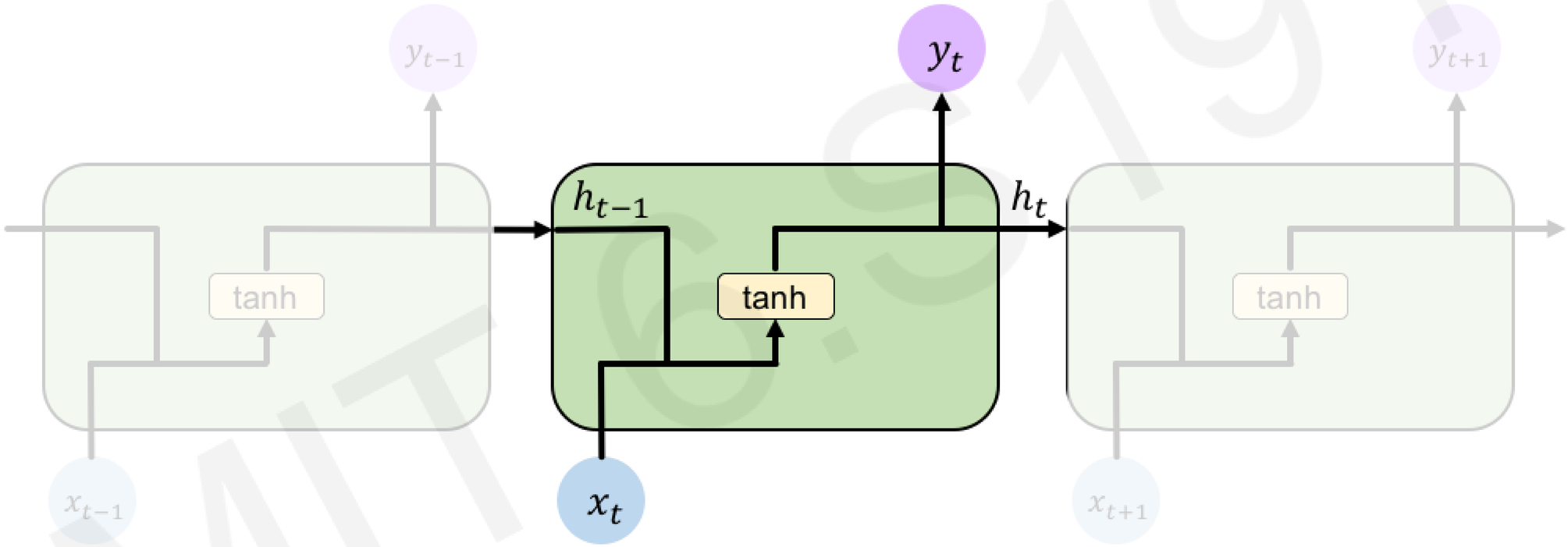
MIT 6.S191

Long Short Term Memory (LSTM) Networks

MIT 6.S191

Standard RNN

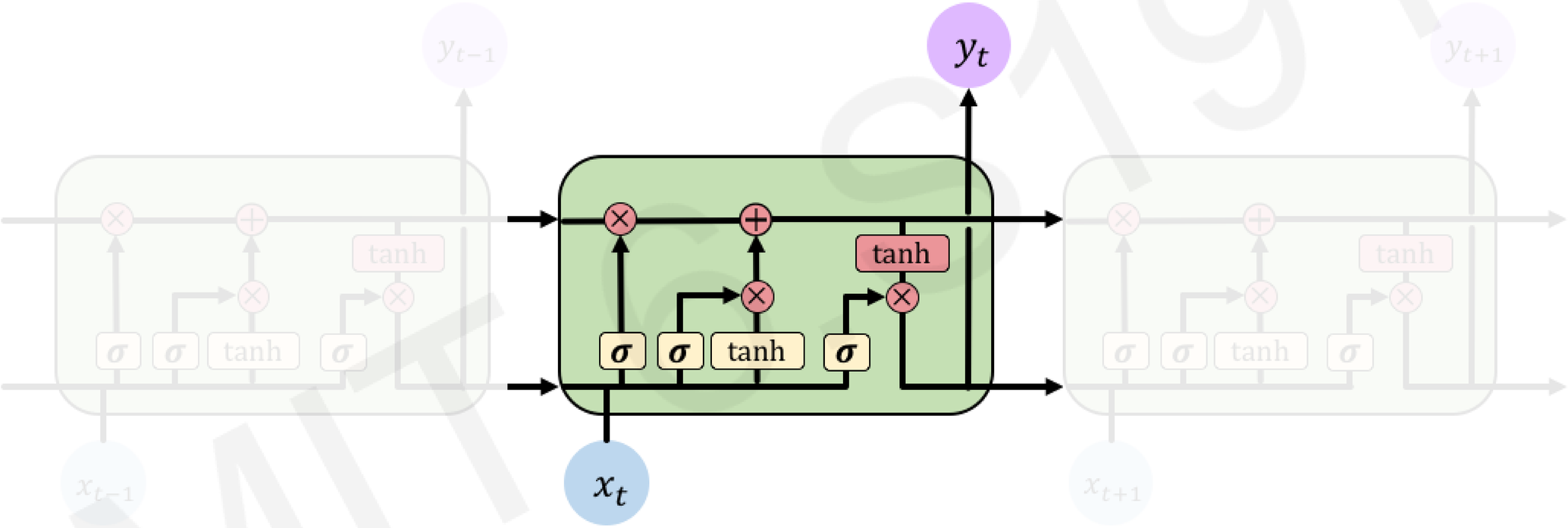
In a standard RNN, repeating modules contain a **simple computation node**



MIT 6.S191

Long Short Term Memory (LSTMs)

LSTM modules contain **computational blocks** that **control information flow**



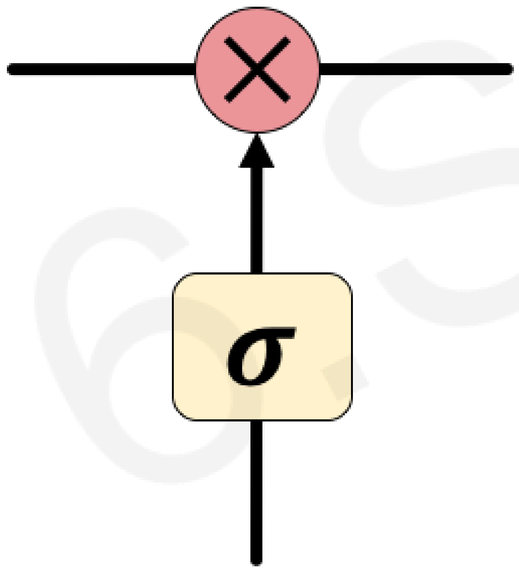
LSTM cells are able to track information throughout many timesteps

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


Hochreiter & Schmidhuber, *Neural Computation* 1997.

Long Short Term Memory (LSTMs)

Information is **added** or **removed** through structures called **gates**



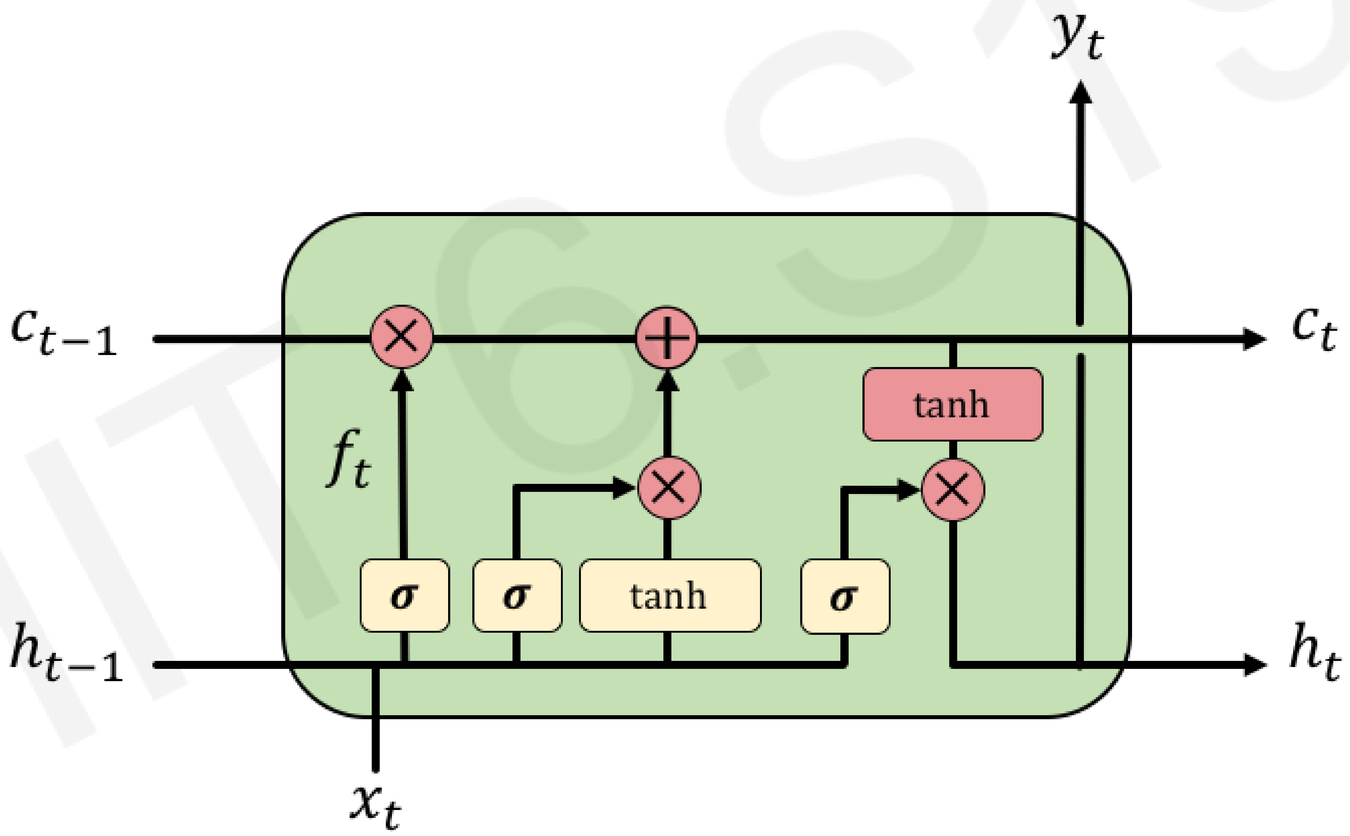
Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

MIT 6.S191

Long Short Term Memory (LSTMs)

How do LSTMs work?

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

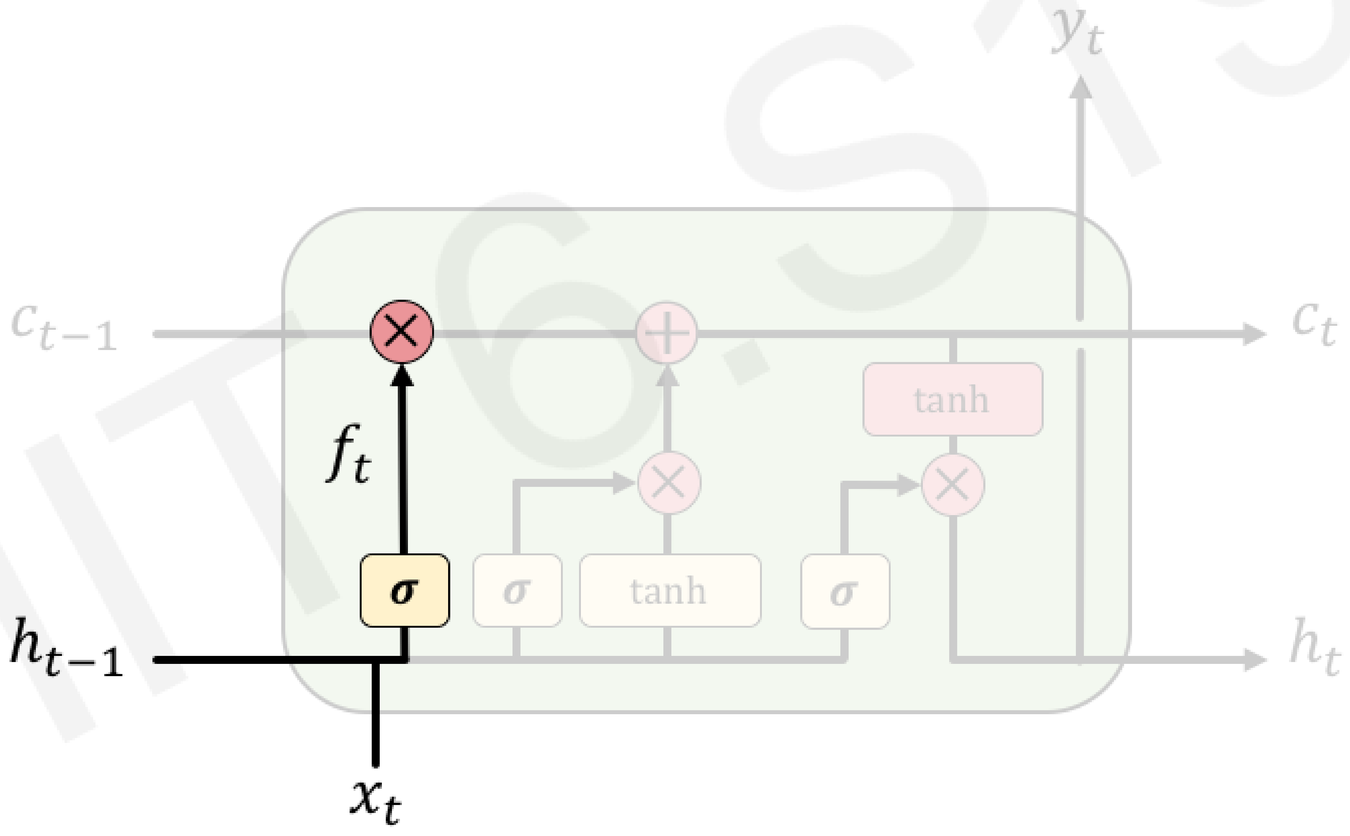


MIT 6.S191

Long Short Term Memory (LSTMs)

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

LSTMs **forget irrelevant** parts of the previous state

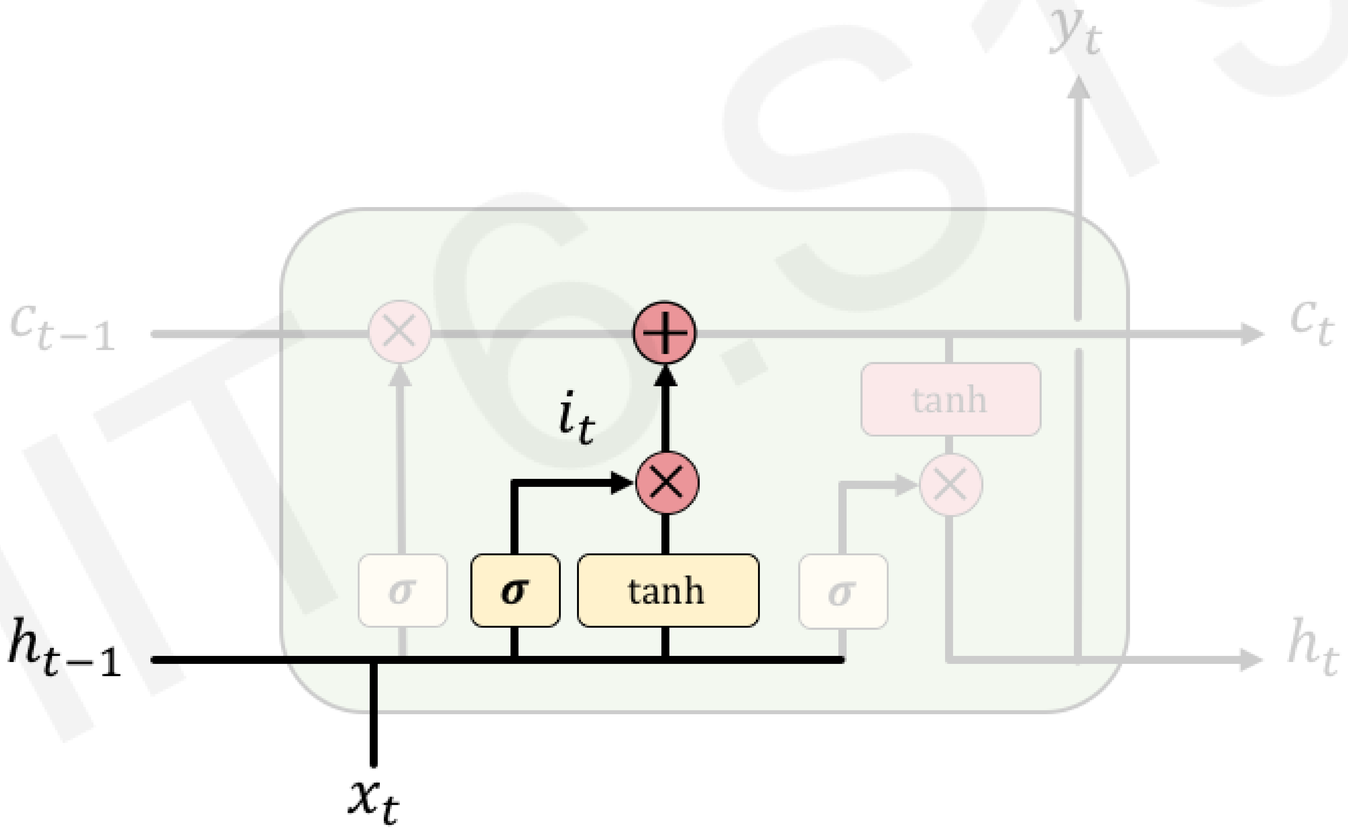


MIT 6.S191

Long Short Term Memory (LSTMs)

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

LSTMs **store relevant** new information into the cell state

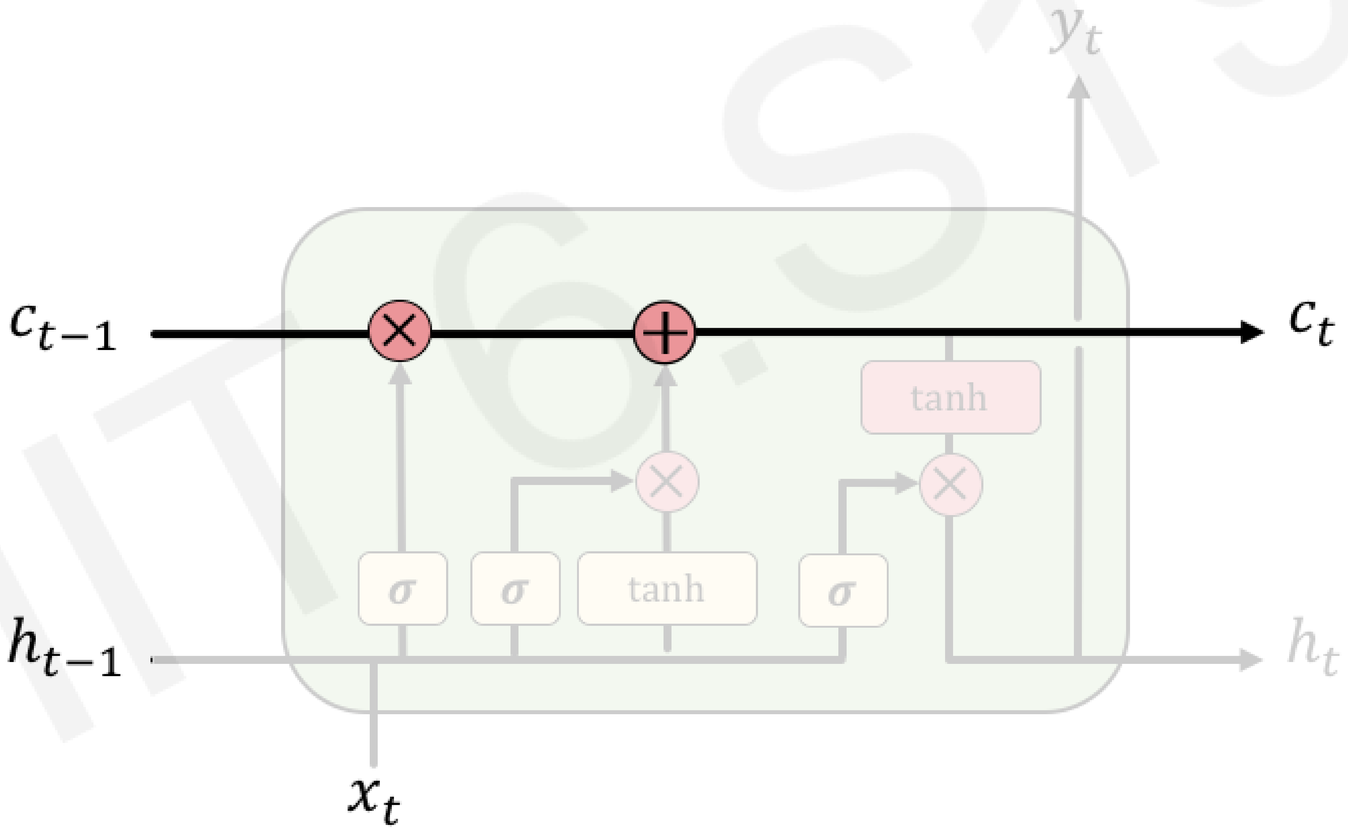


MIT 6.S191

Long Short Term Memory (LSTMs)

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

LSTMs **selectively update** cell state values

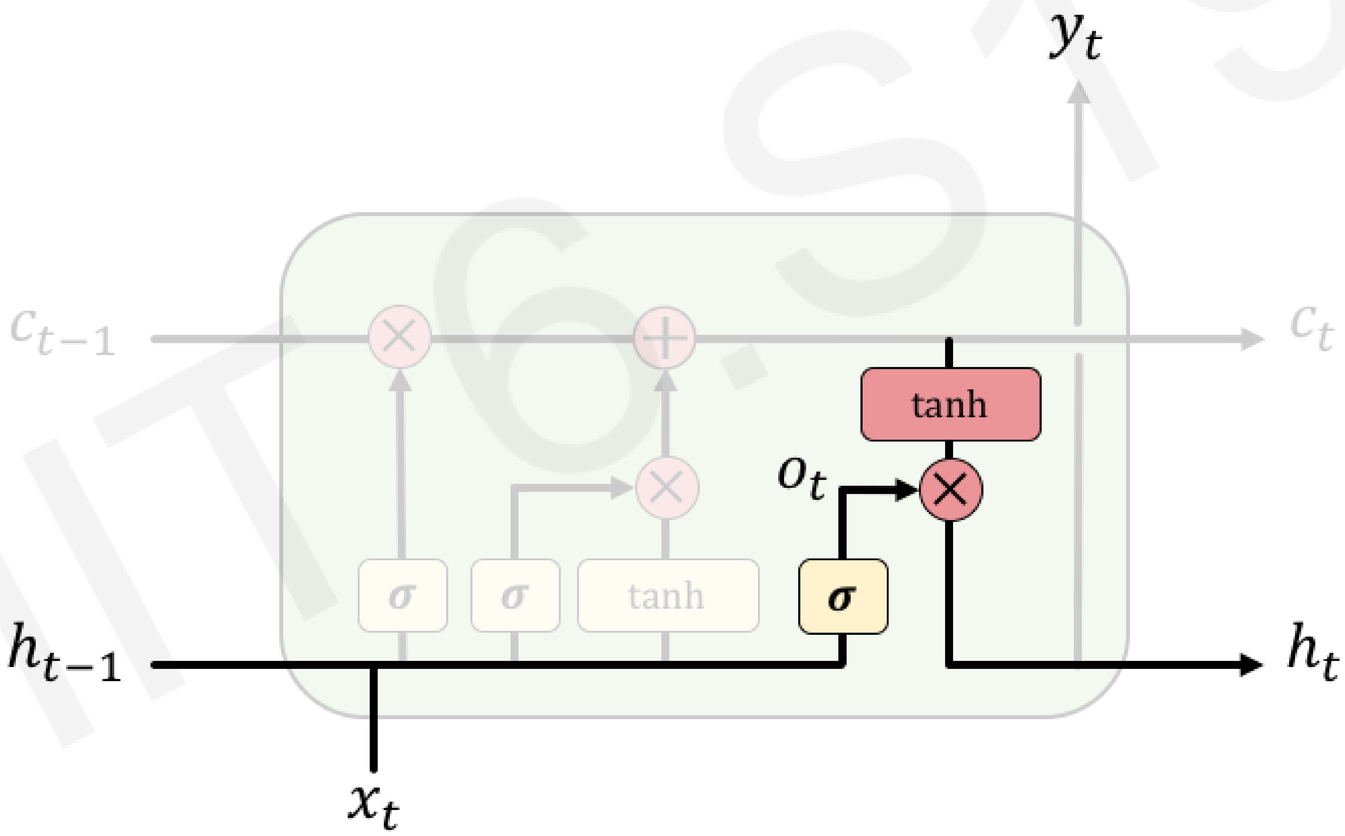


MIT 6.S191

Long Short Term Memory (LSTMs)

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

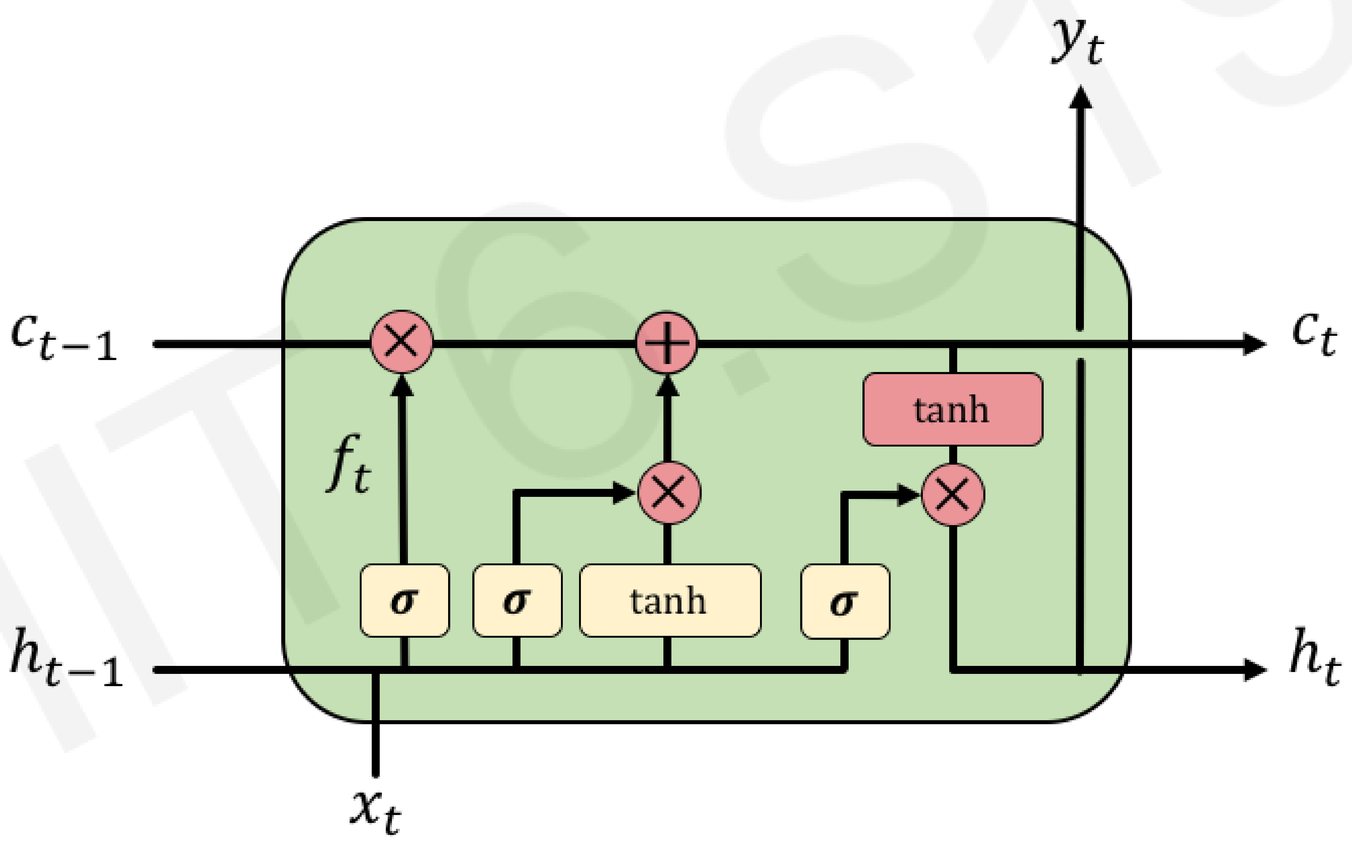
The **output gate** controls what information is sent to the next time step



MIT 6.S191

Long Short Term Memory (LSTMs)

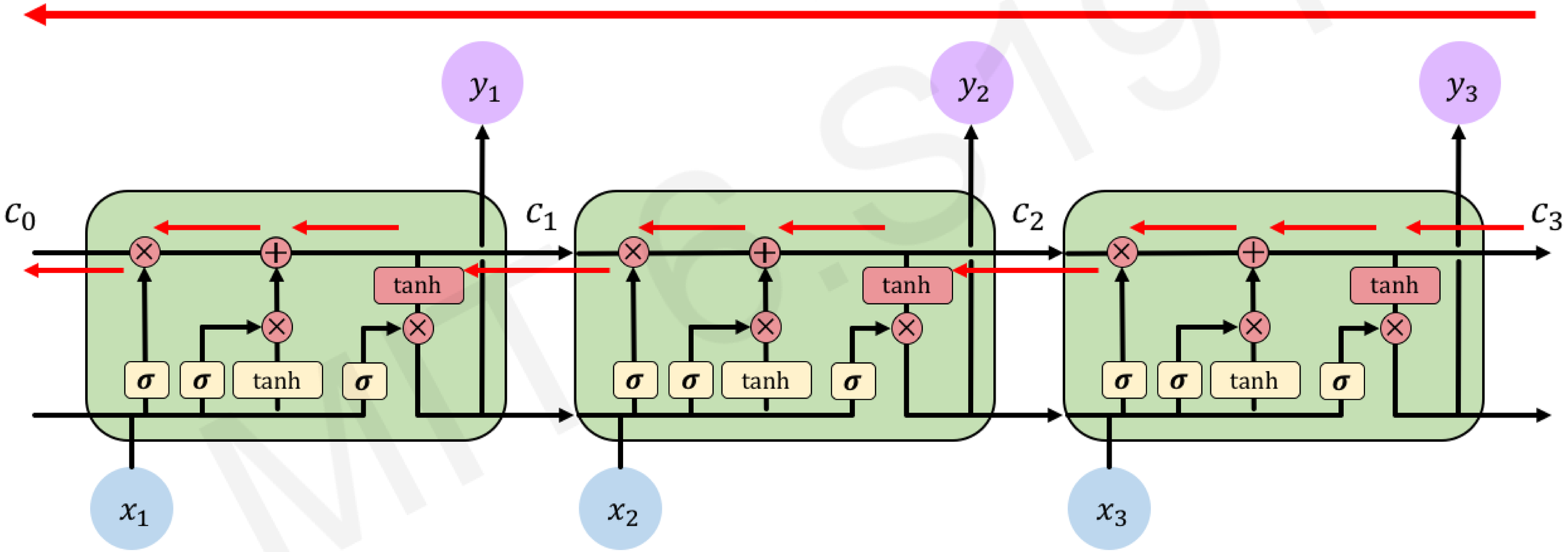
1) Forget 2) Store 3) Update 4) Output



MIT 6.S191

LSTM Gradient Flow

Uninterrupted gradient flow!



MIT 6.S191

LSTMs: Key Concepts

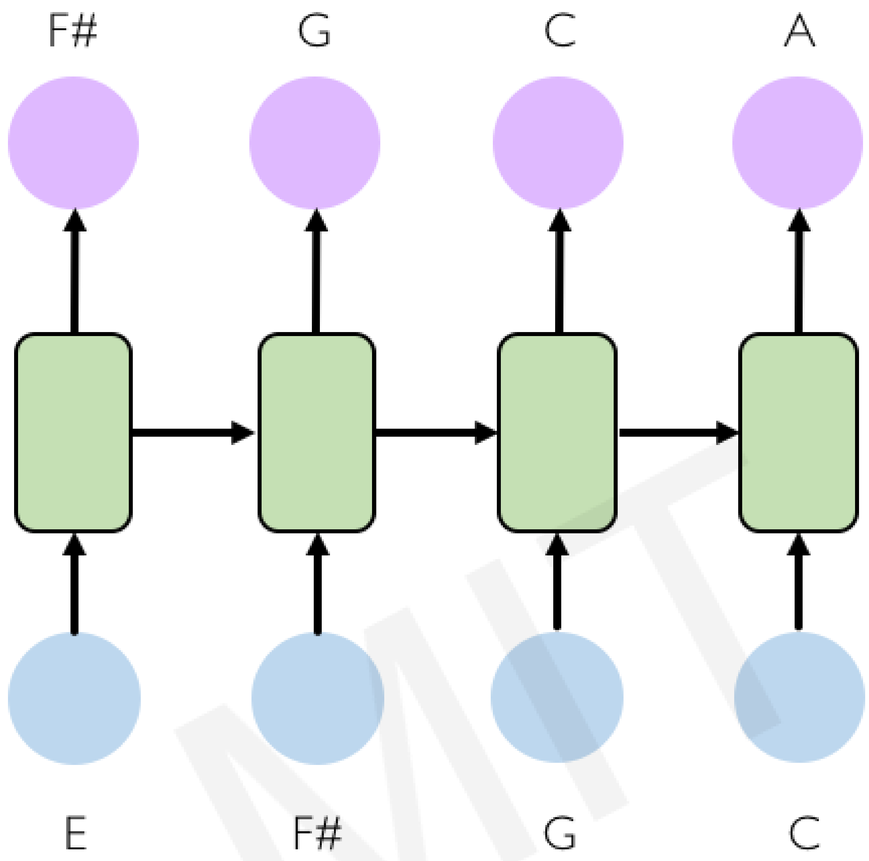
1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered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

MIT 6.S191

RNN Applications

MIT 6.S191

Example Task: Music Generation



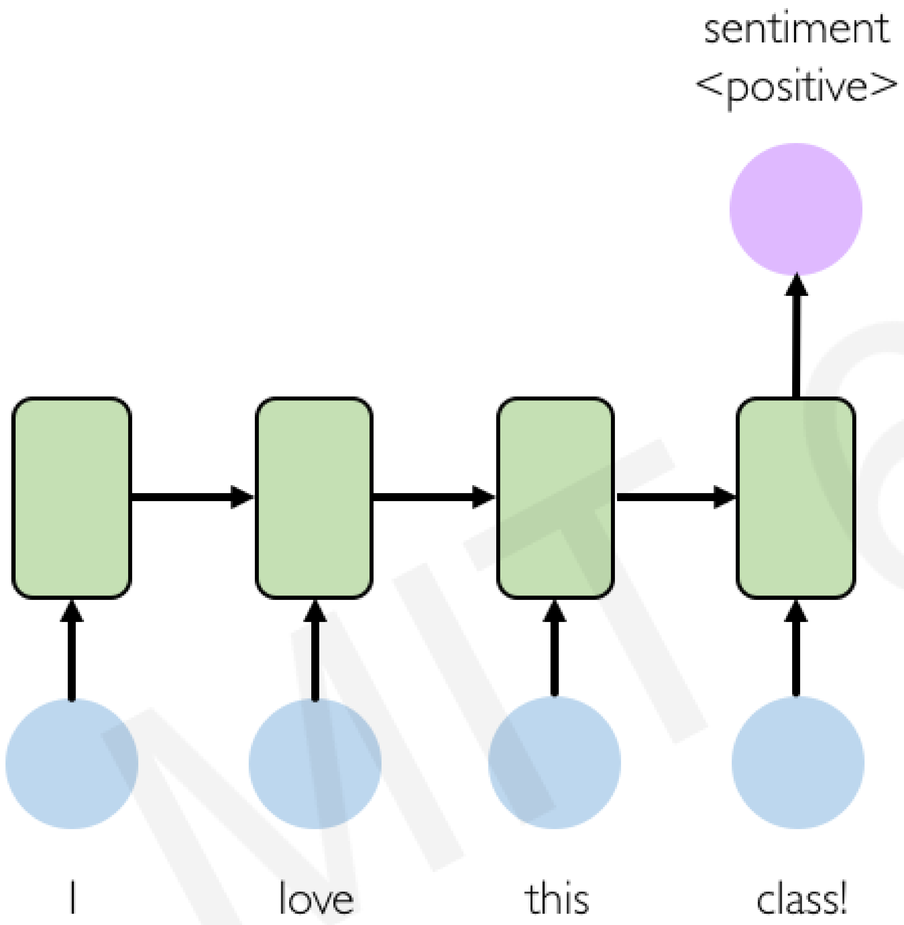
Input: sheet music

Output: next character in sheet music

6.S191 Lab!

MIT 6.S191

Example Task: Sentiment Classification



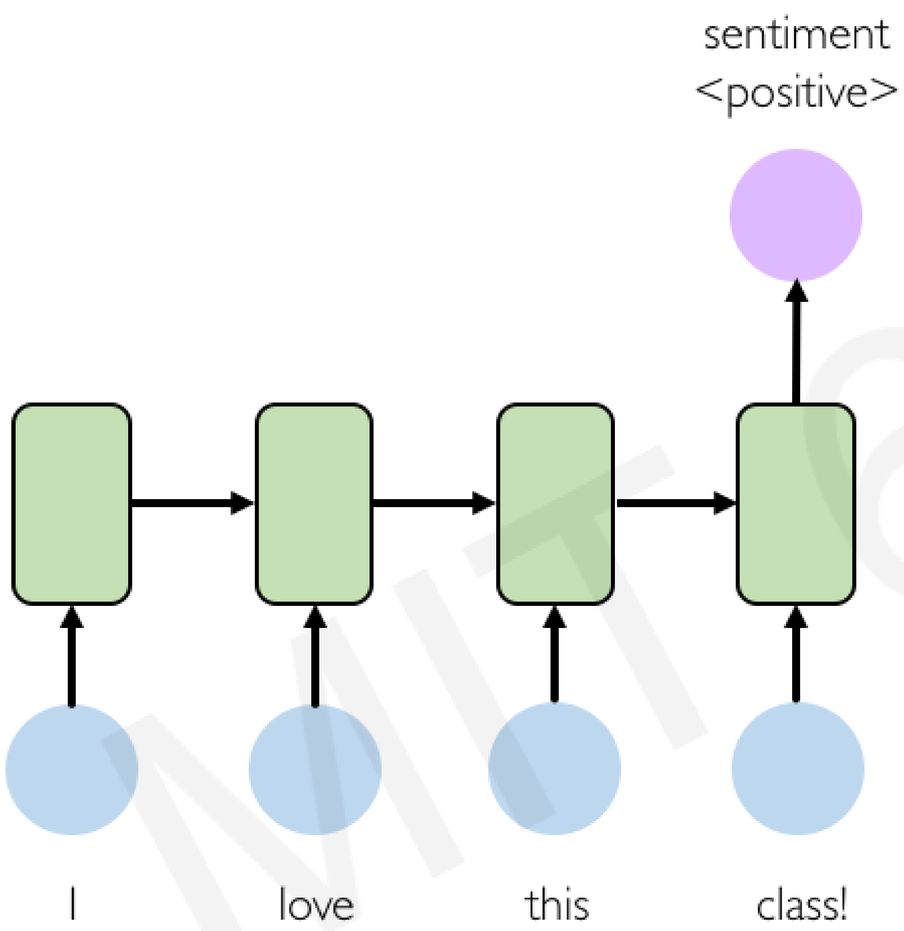
Input: sequence of words

Output: probability of having positive sentiment

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

MIT 6.S191

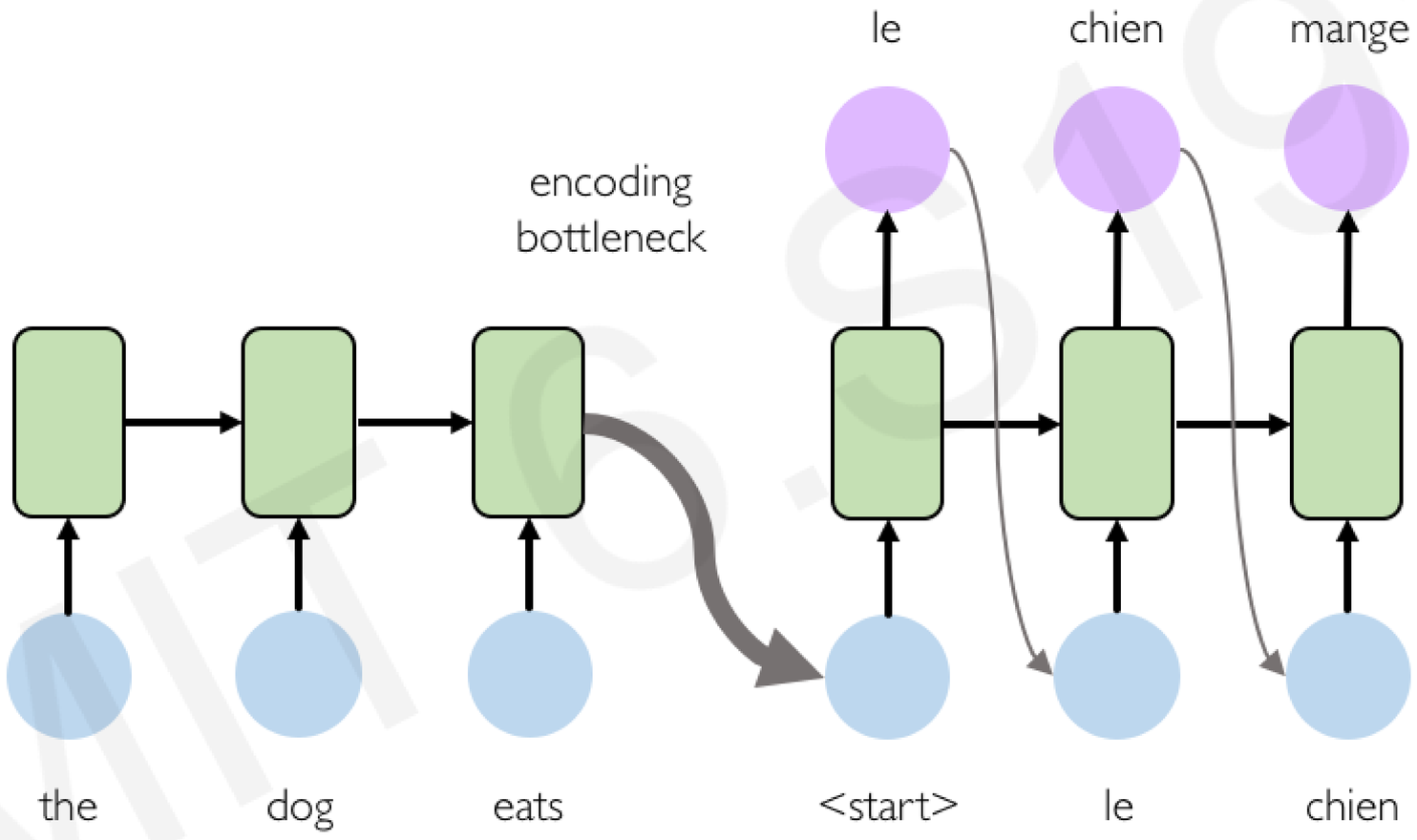
Example Task: Sentiment Classification



Tweet sentiment classification

MIT 6.S191

Example Task: Machine Translation



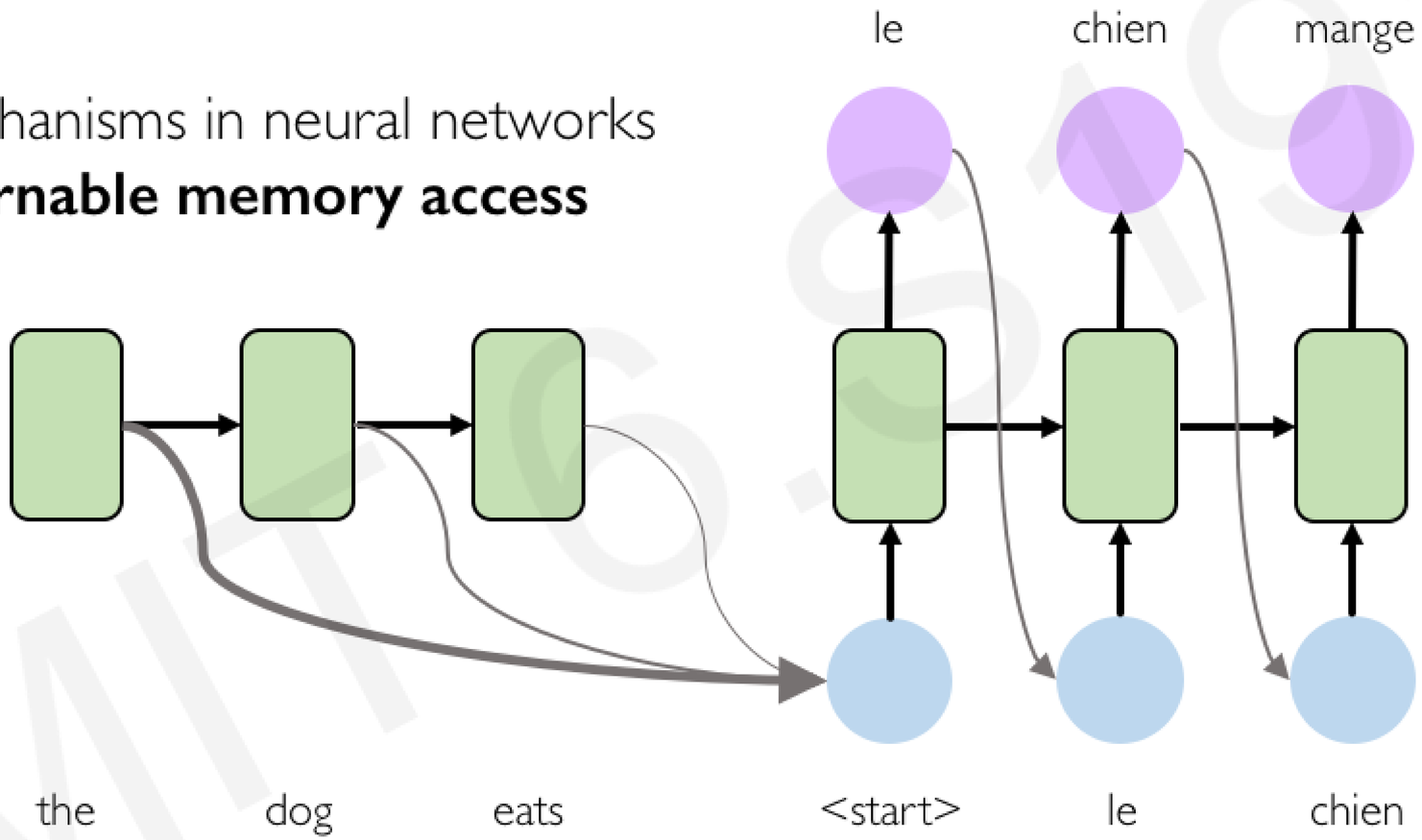
Encoder (English)

Decoder (French)

MIT 6.S191

Attention Mechanisms

Attention mechanisms in neural networks provide **learnable memory access**



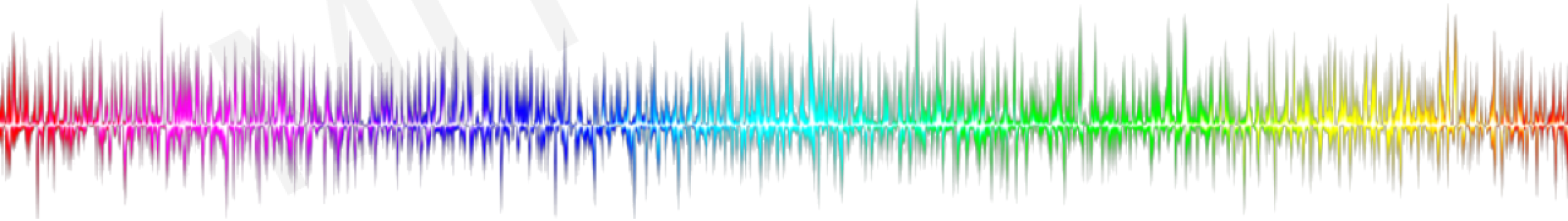
Encoder (English)

Decoder (French)

MIT 6.S191

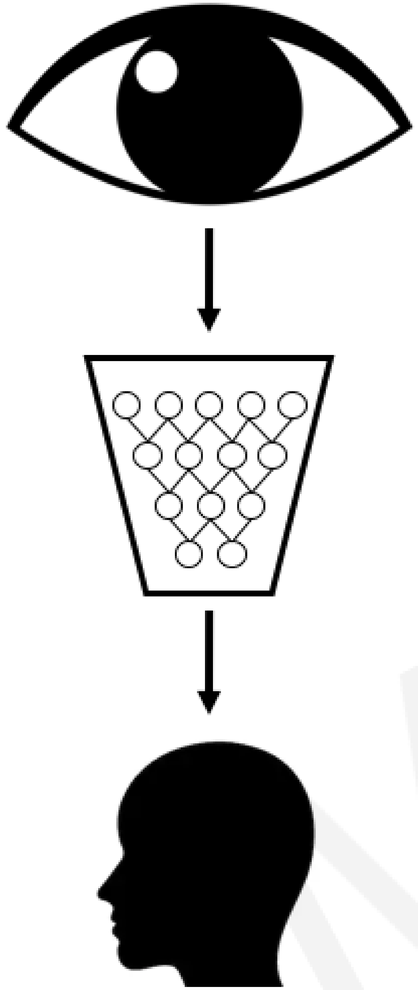
Deep Learning for Sequence Modeling: Summary

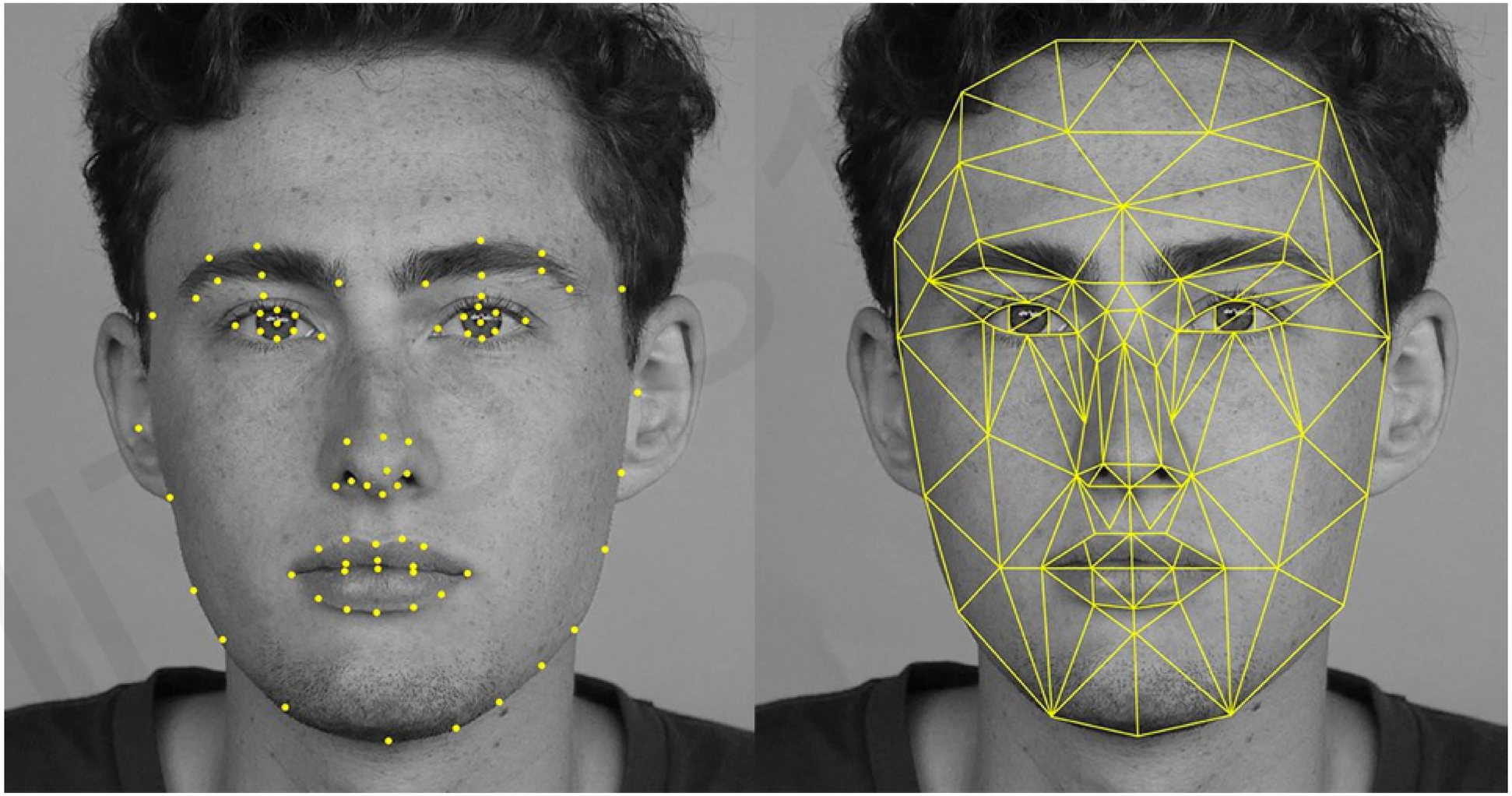
1. RNNs are well suited for **sequence modeling** tasks
2. Model sequences via a **recurrence relation**
3. Training RNNs with **backpropagation through time**
4. Gated cells like **LSTMs** let us model **long-term dependencies**
5. Models for **music generation**, classification, machine translation, and more



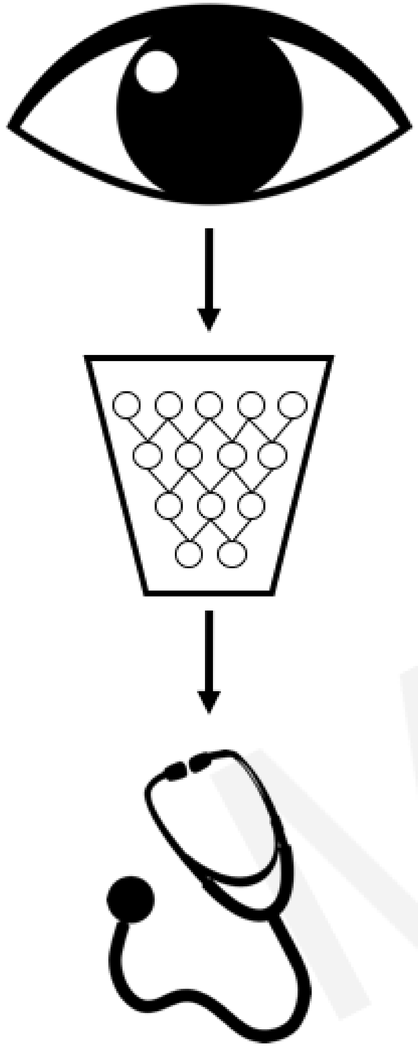
MIT 6.S191

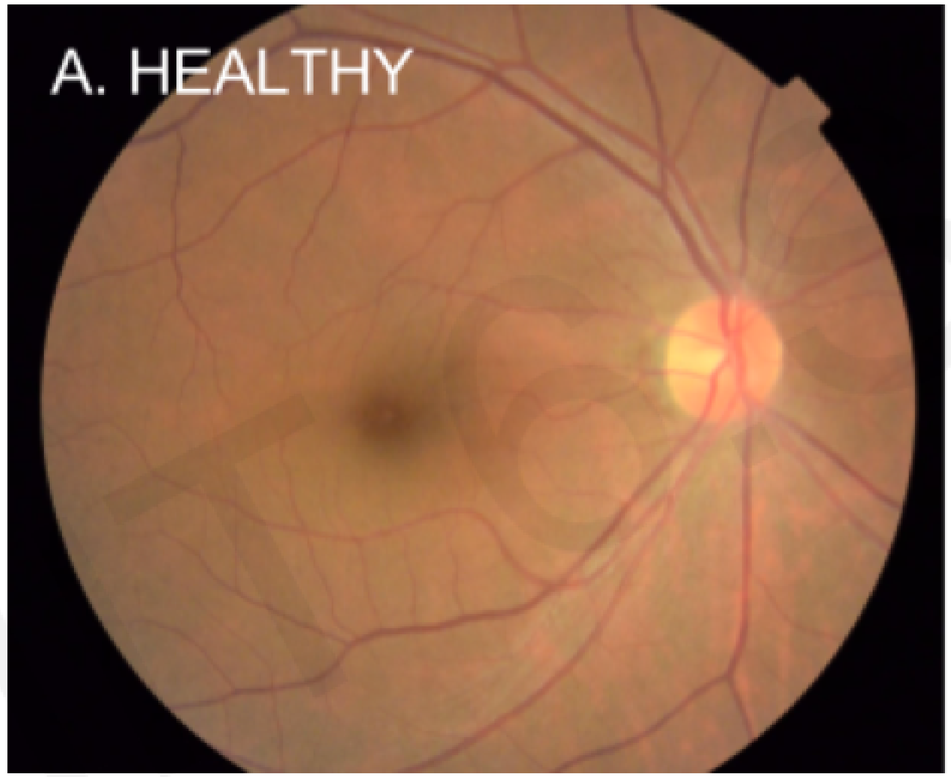
Impact: Facial Detection & Recognition

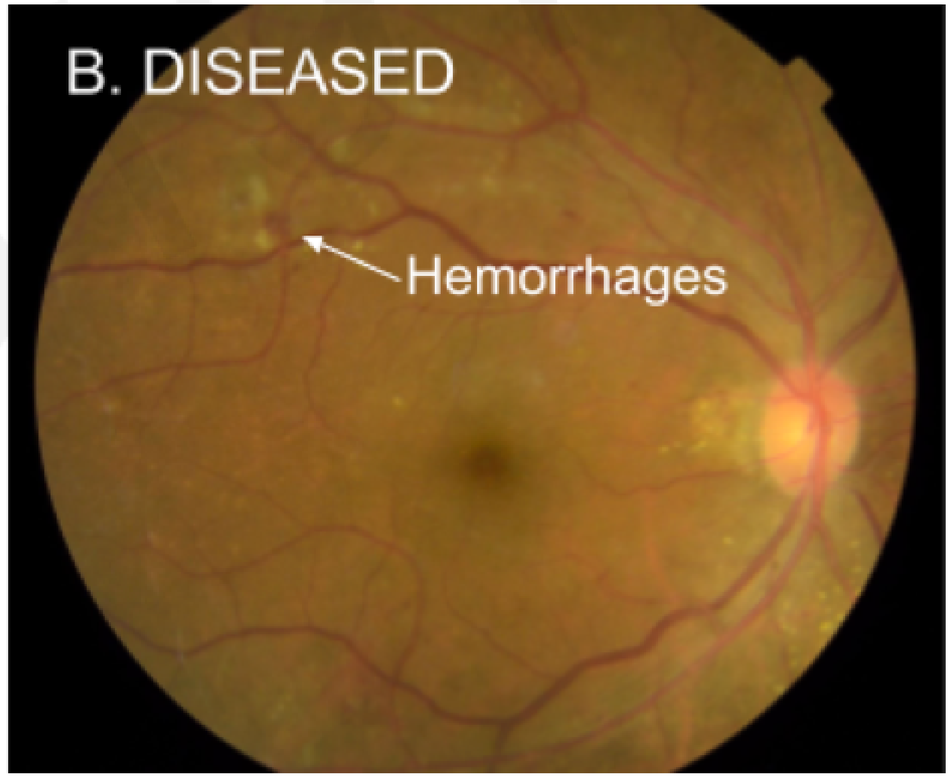




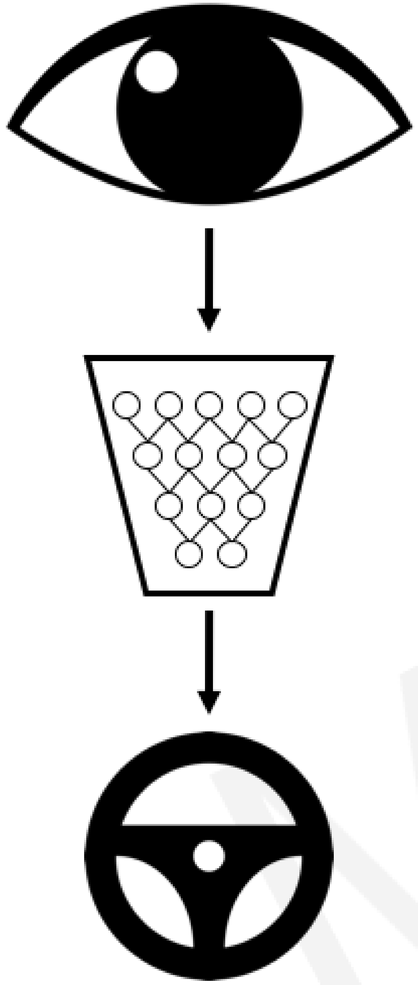
Impact: Medicine, Biology, Healthcare







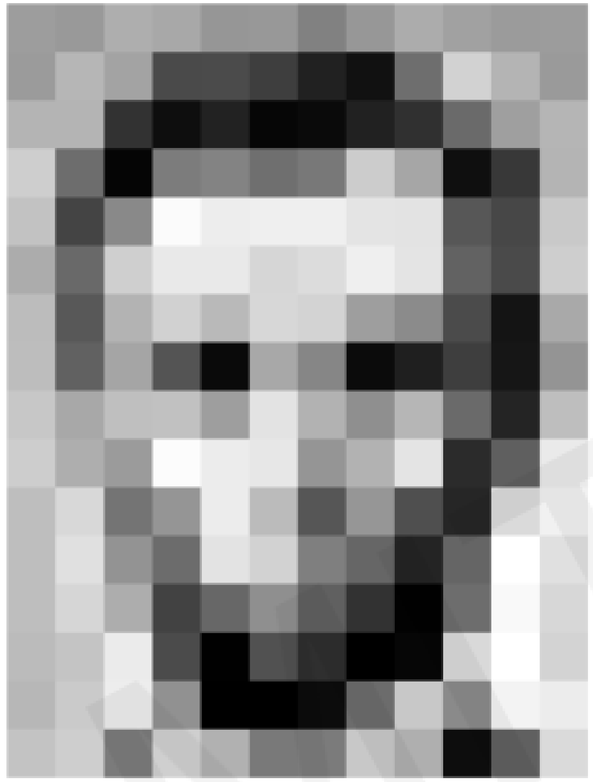
Impact: Self-Driving Cars



What Computers “See”

MIT 6.S191

Images are Numbers



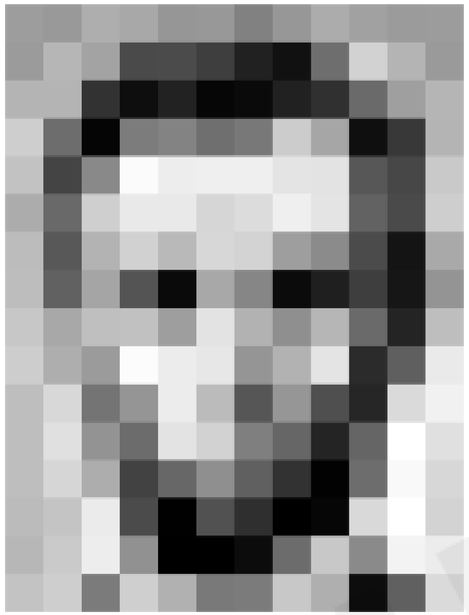
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Tasks in Computer Vision



Input Image

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel Representation

classification

Lincoln	0.8
Washington	0.1
Jefferson	0.05
Obama	0.05

- **Regression**: output variable takes continuous value
- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class

MIT 6.S191

High Level Feature Detection

Let's identify key features in each image category

Nose,
Eyes,
Mouth

Wheels,
License Plate,
Headlights

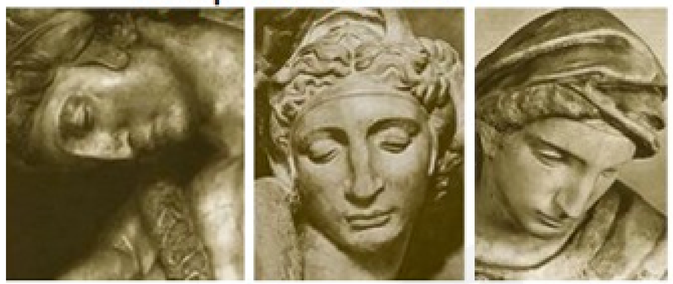
Door,
Windows,
Steps

MIT 6.S191

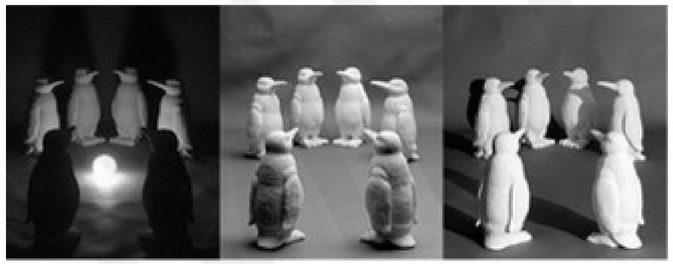
Manual Feature Extraction

Domain knowledge → Define features → Detect features to classify

Viewpoint variation

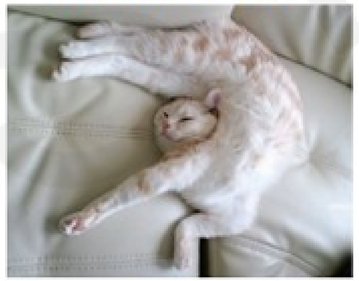


Illumination conditions



Scale variation

Deformation



Occlusion

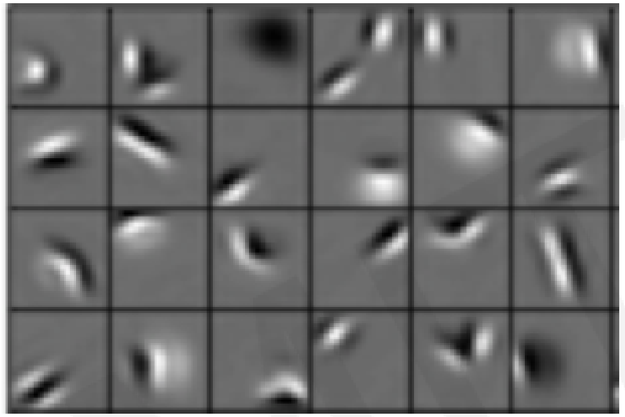
Background clutter

Intra-class variation

Learning Feature Representations

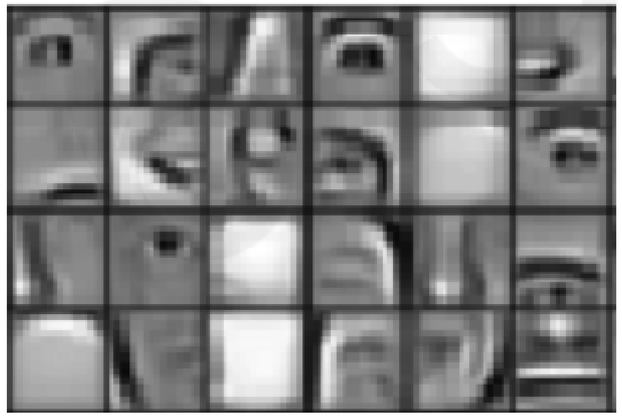
Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

High level features

Facial structure

MIT 6.S191

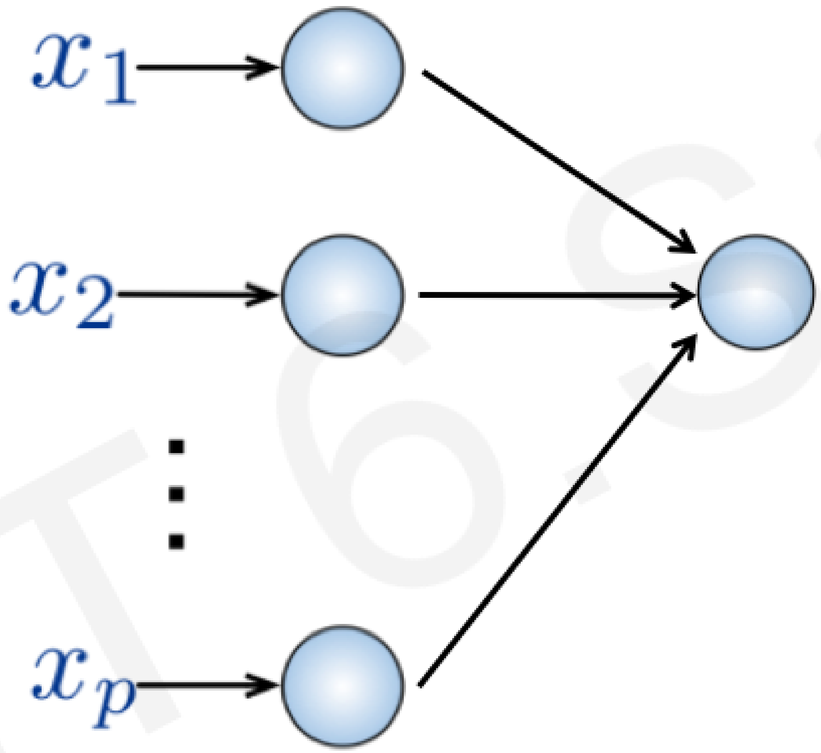
Learning Visual Features

MIT 6.S191

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values



Fully Connected:

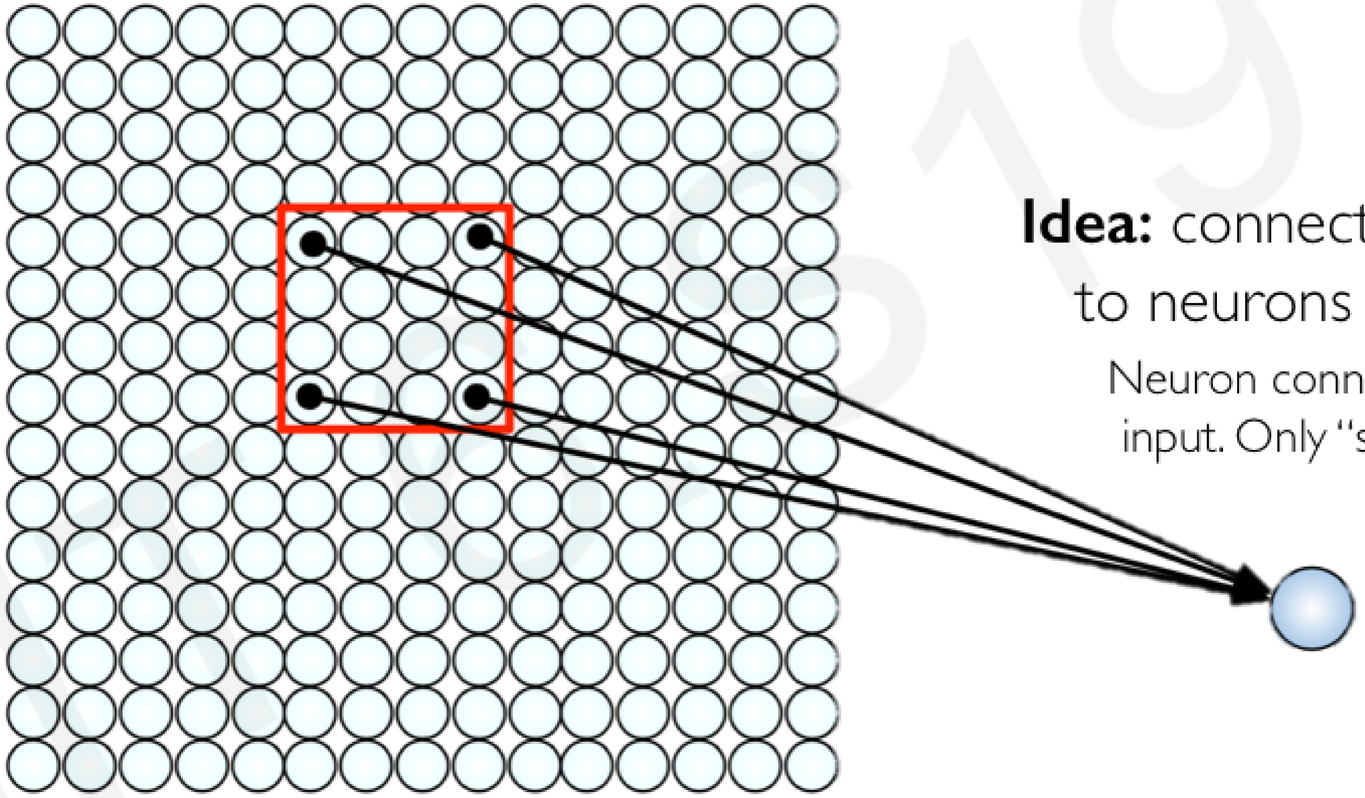
- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

MIT 6.S191

Using Spatial Structure

Input: 2D image.
Array of pixel values

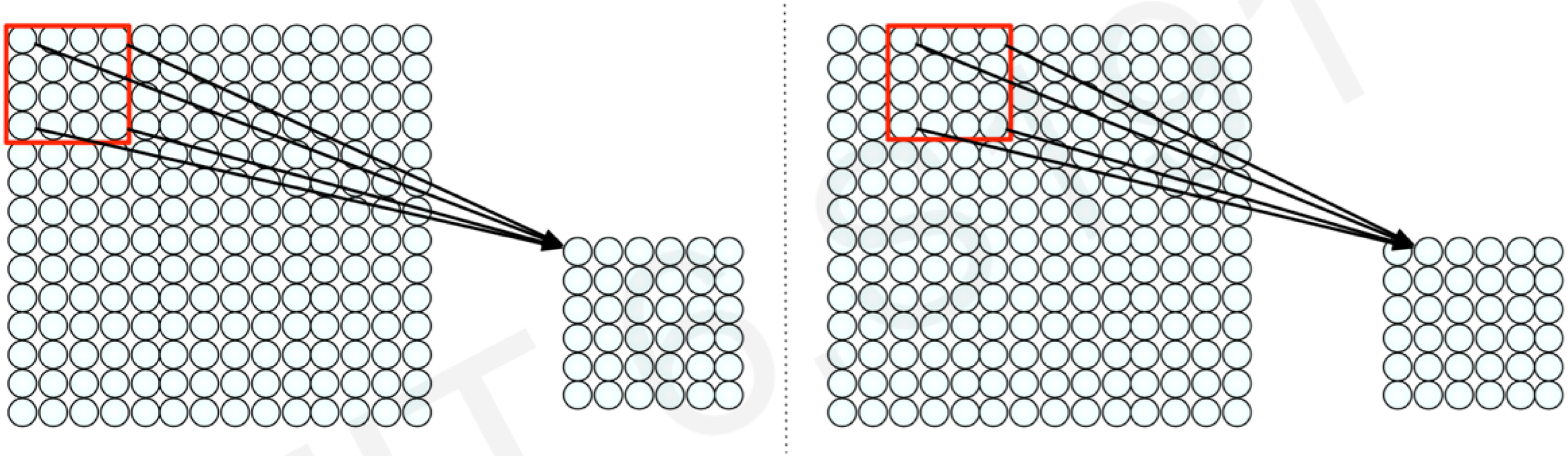


Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.

MIT 6.S191

Using Spatial Structure



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

Applying Filters to Extract Features

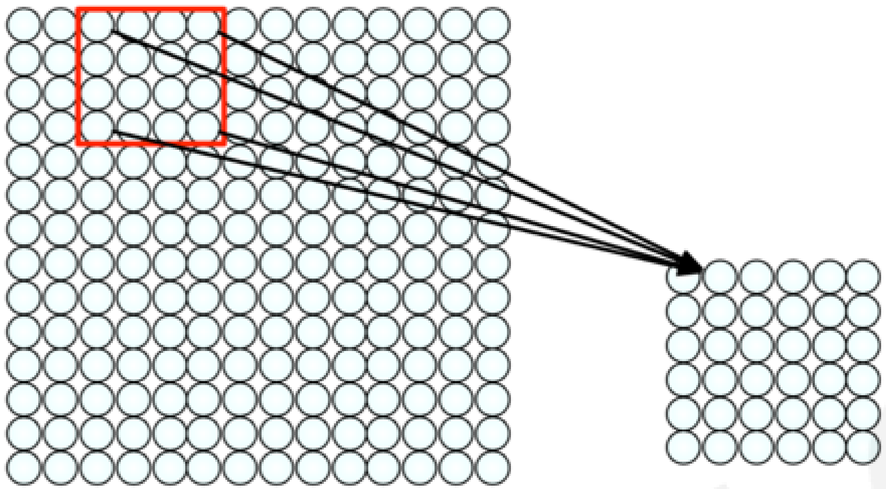
1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) Spatially **share** parameters of each filter
(features that matter in one part of the input should matter elsewhere)

MIT 6.S191

Feature Extraction with Convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

MIT 6.S191

Feature Extraction and Convolution

A Case Study

MIT 6.S191

X or X?

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

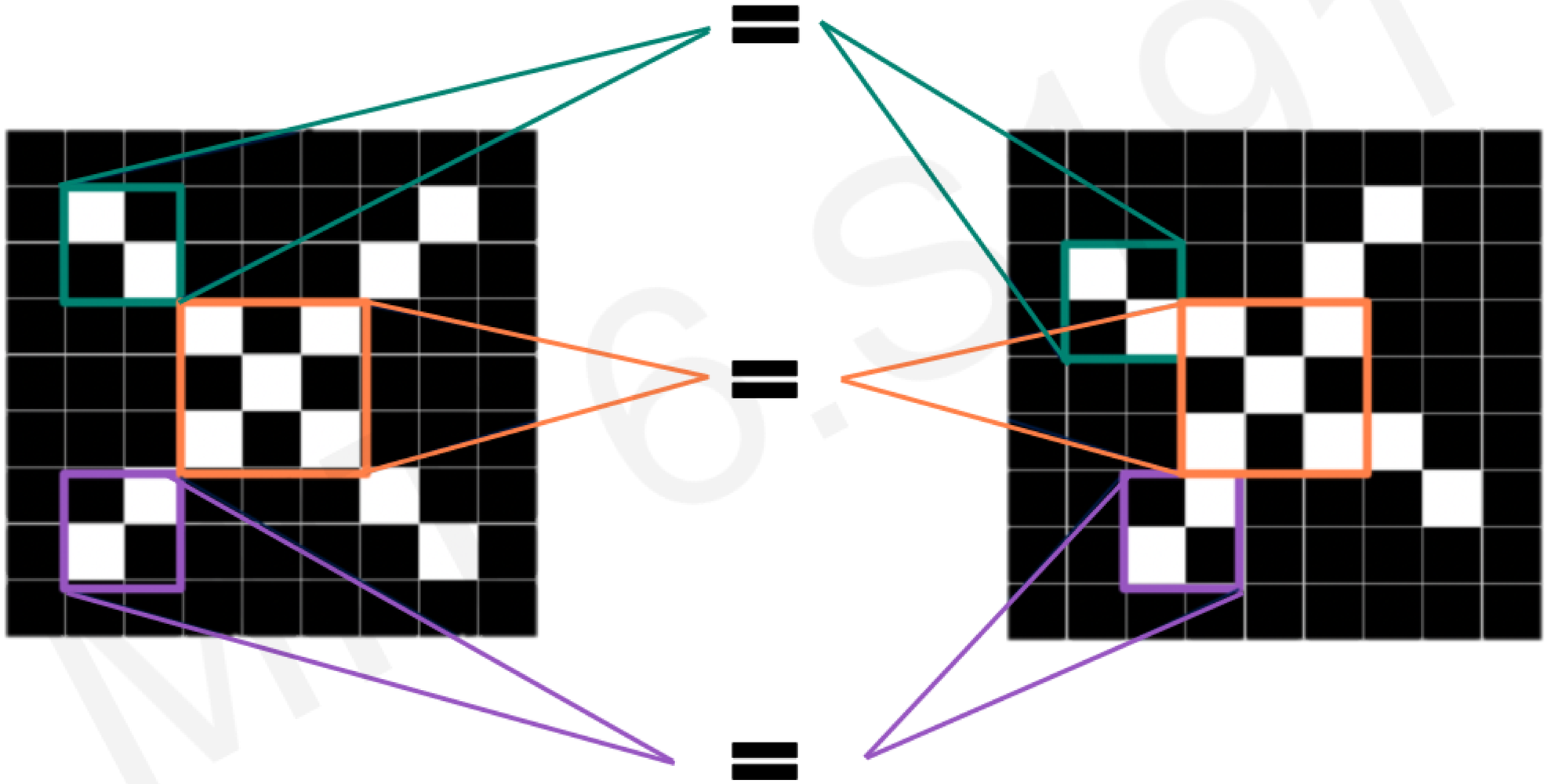
?

≠

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

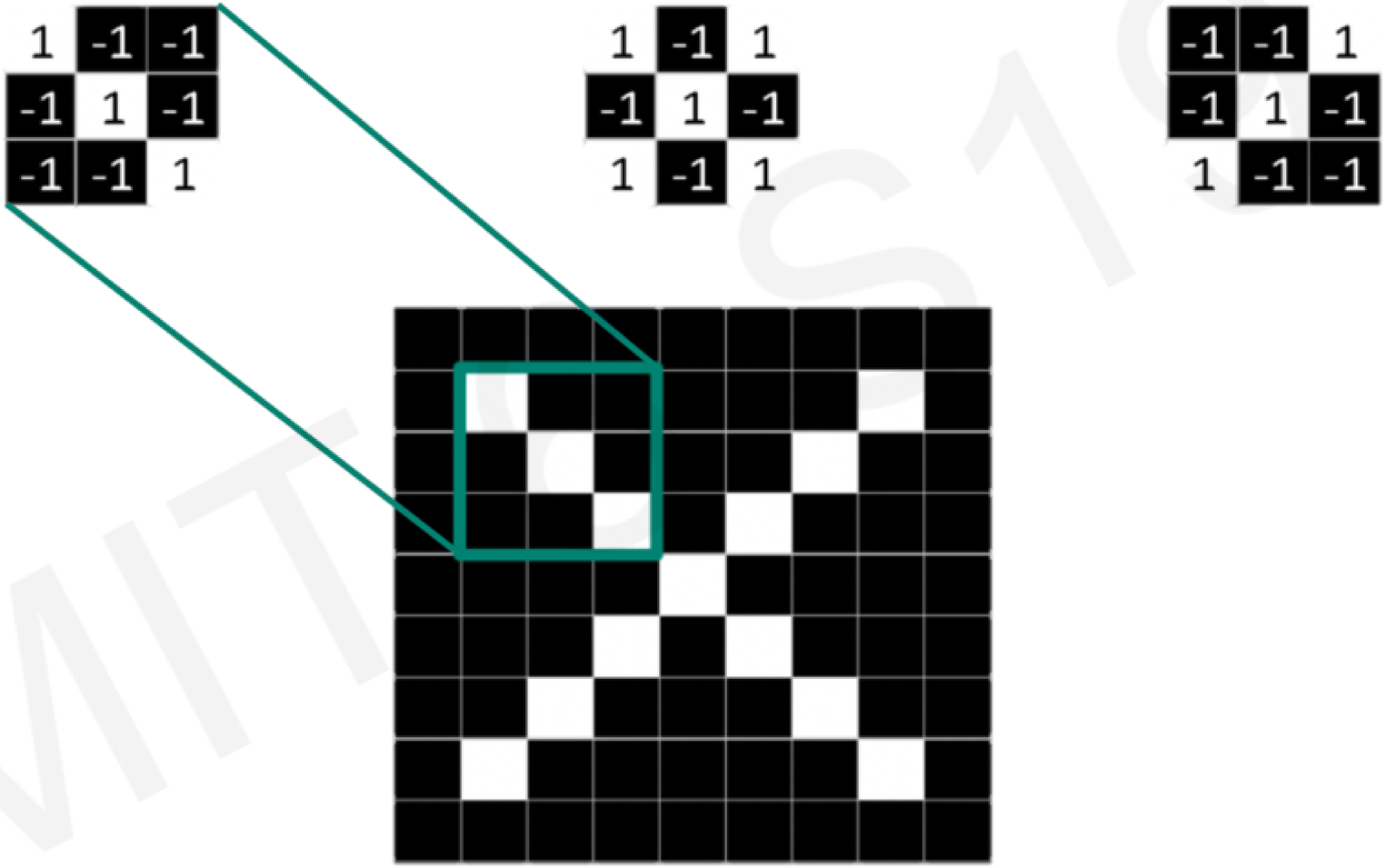
Features of X



MIT 6.S191

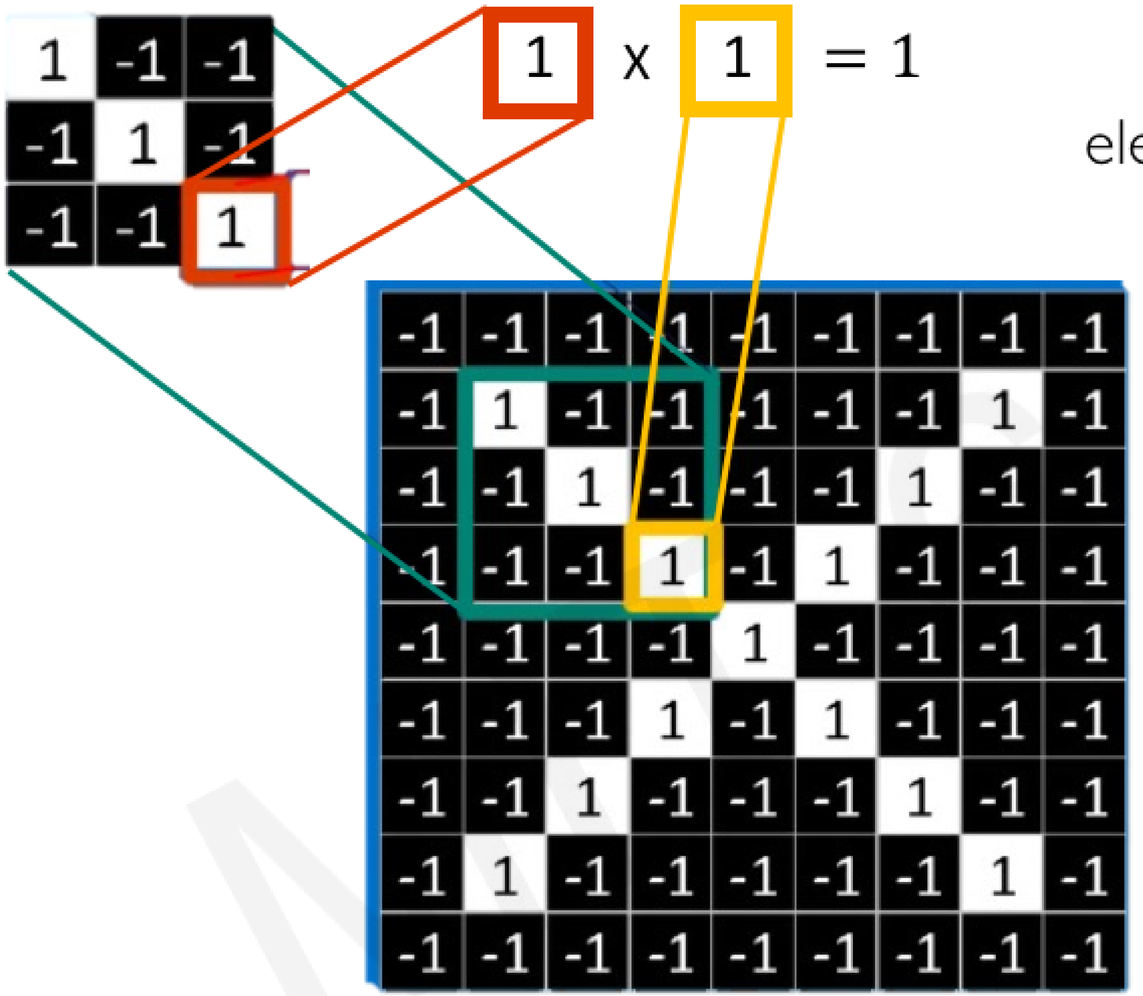
Filters to Detect X Features

filters



MIT 6.S191

The Convolution Operation



element wise multiply

add outputs

$$\odot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & \boxed{1} \end{bmatrix} = 9$$

MIT 6.S191

The Convolution Operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

image

$\otimes$

1	0	1
0	1	0
1	0	1

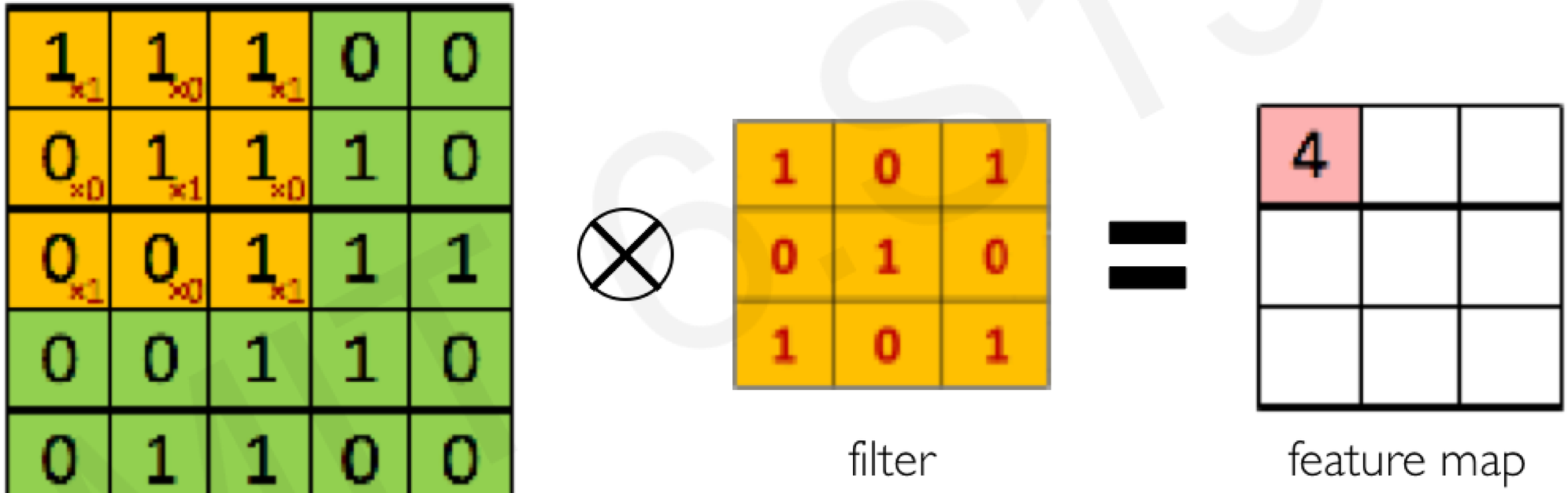
filter

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

MIT 6.S191

The Convolution Operation

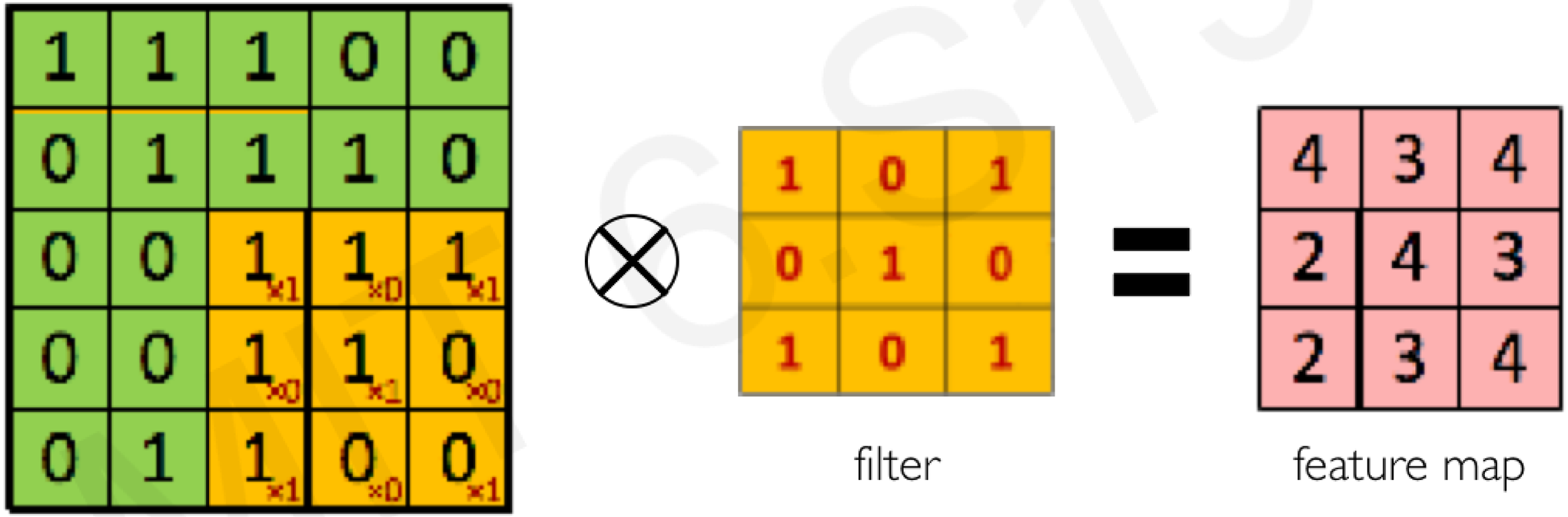
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



MIT 6.S191

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



Producing Feature Maps

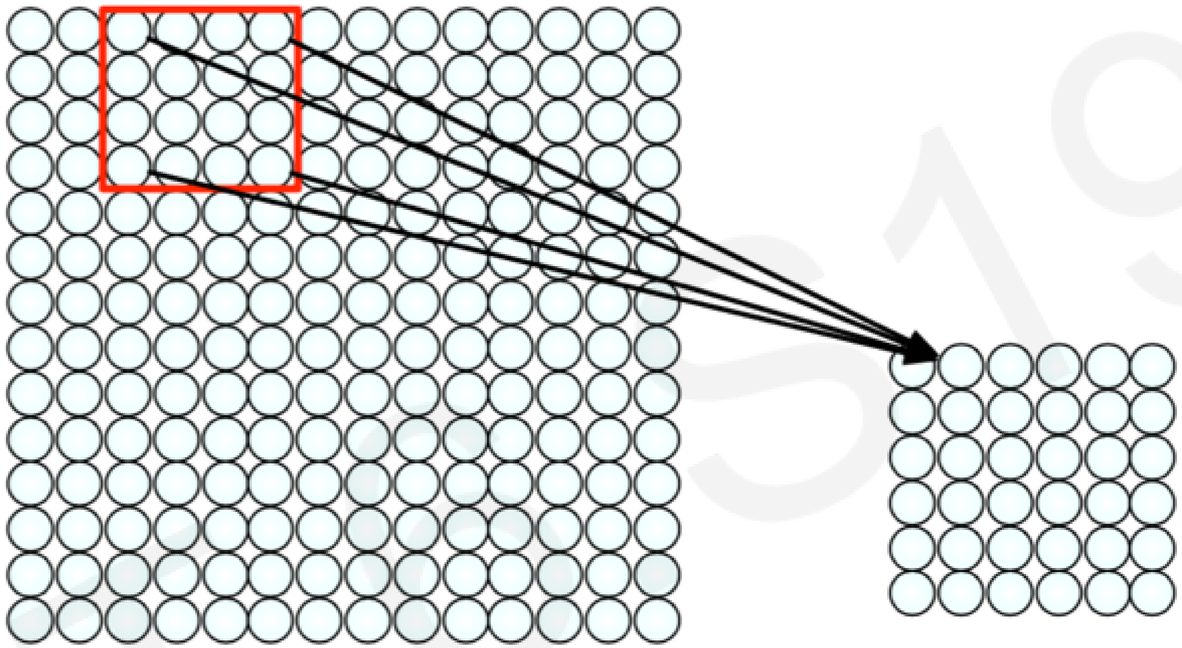
Original

Sharpen

Edge Detect

"Strong" Edge Detect

Feature Extraction with Convolution



1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

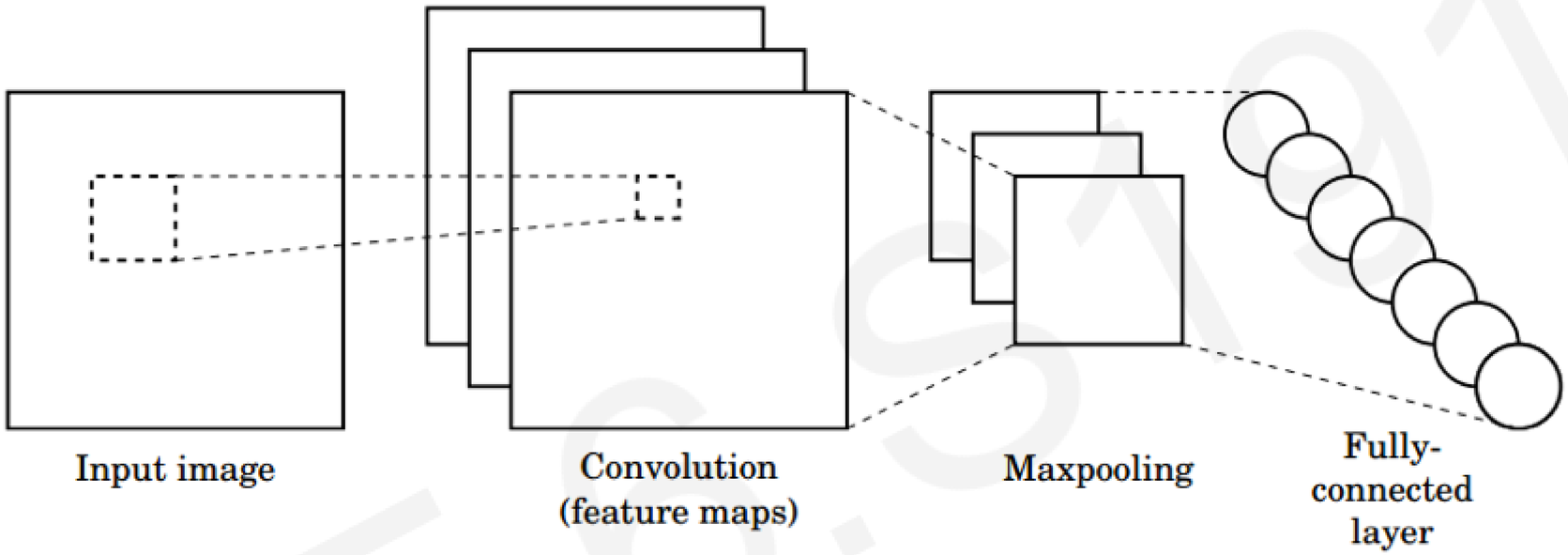
3) **Spatially share** parameters of each filter

MIT 6.S191

Convolutional Neural Networks (CNNs)

MIT 6.S191

CNNs for Classification



1. **Convolution**: Apply filters to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

`tf.keras.layers.Conv2D`

`tf.keras.activations.*`

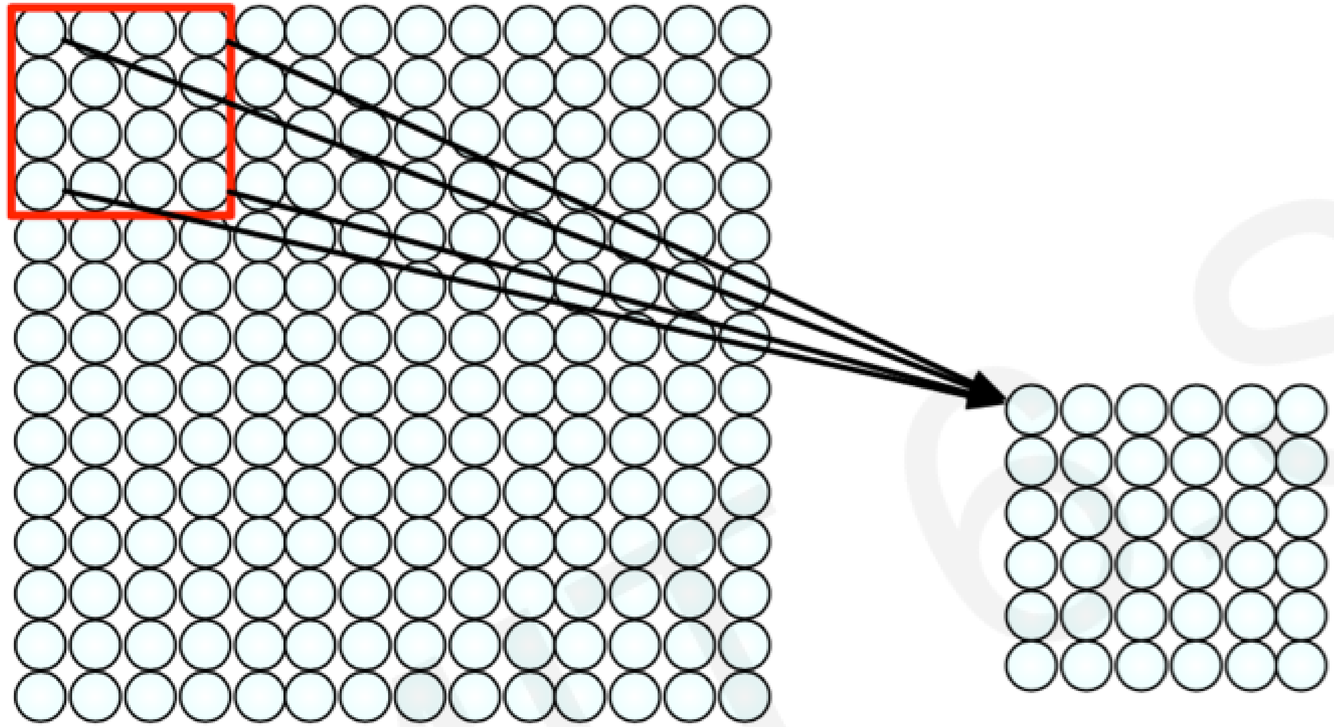
`tf.keras.layers.MaxPool2D`

Train model with image data.
Learn weights of filters in convolutional layers.

MIT 6.S191

Convolutional Layers: Local Connectivity

`tf.keras.layers.Conv2D`



For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

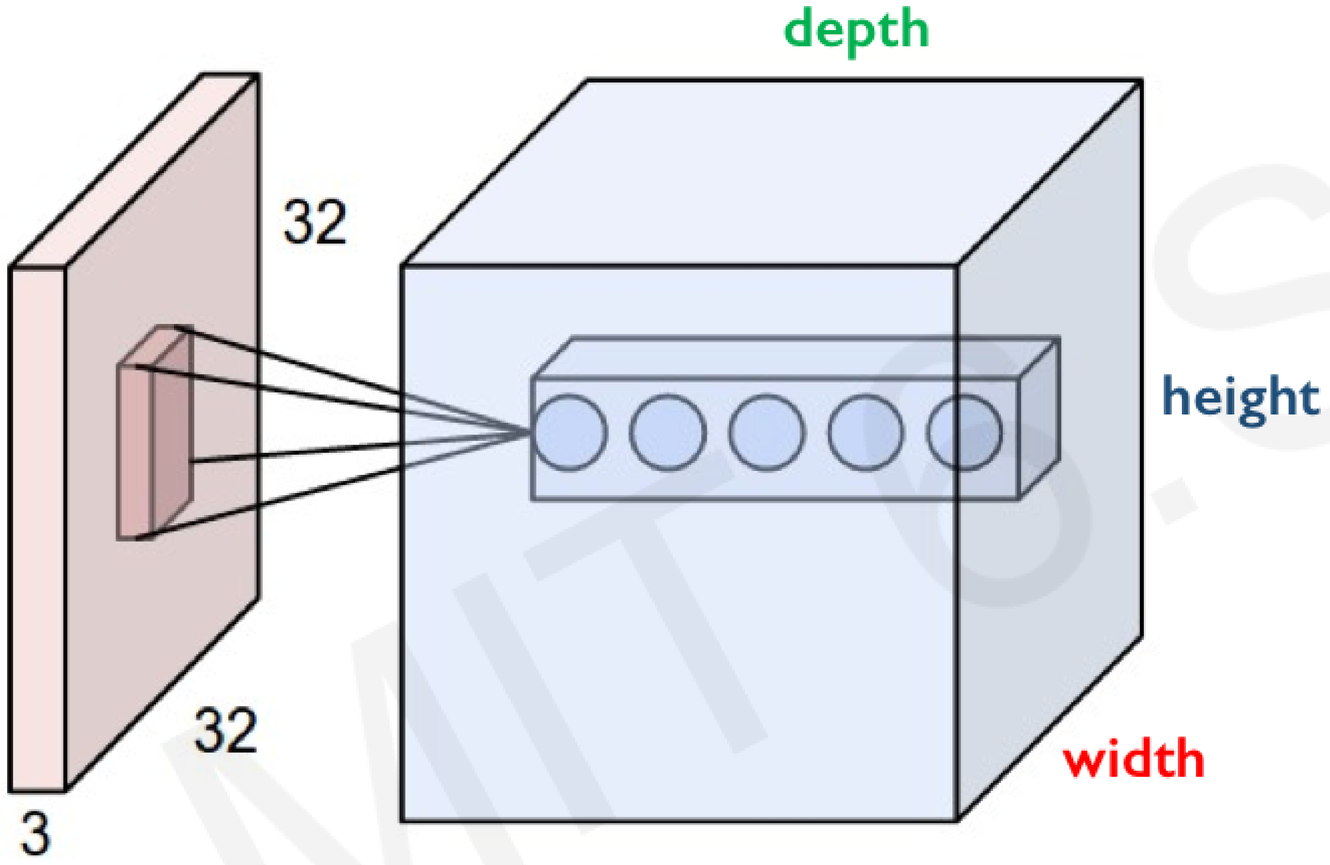
4x4 filter: matrix of weights w_{ij}

$$\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}\, x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

1) applying a window of weights
2) computing linear combinations
3) activating with non-linear function

CNNs: Spatial Arrangement of Output Volume



Layer Dimensions:

$h \; x \; w \; x \; d$

where h and w are spatial dimensions
d (depth) = number of filters

Stride:

Filter step size

Receptive Field:

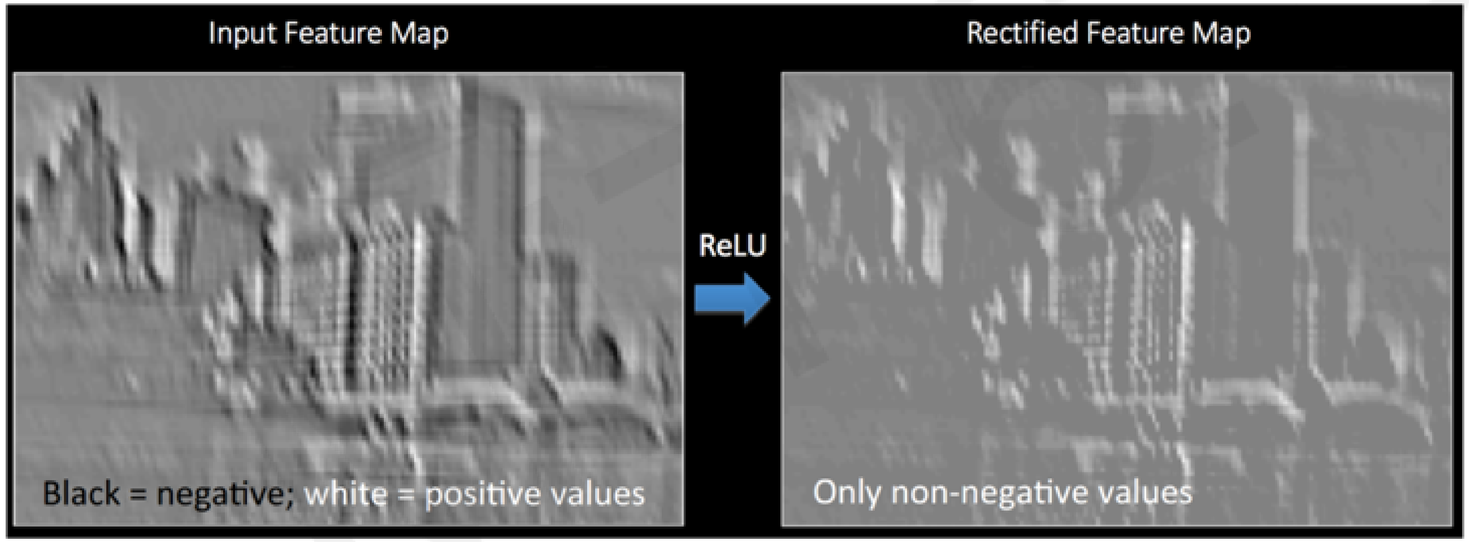
Locations in input image that
a node is path connected to

```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

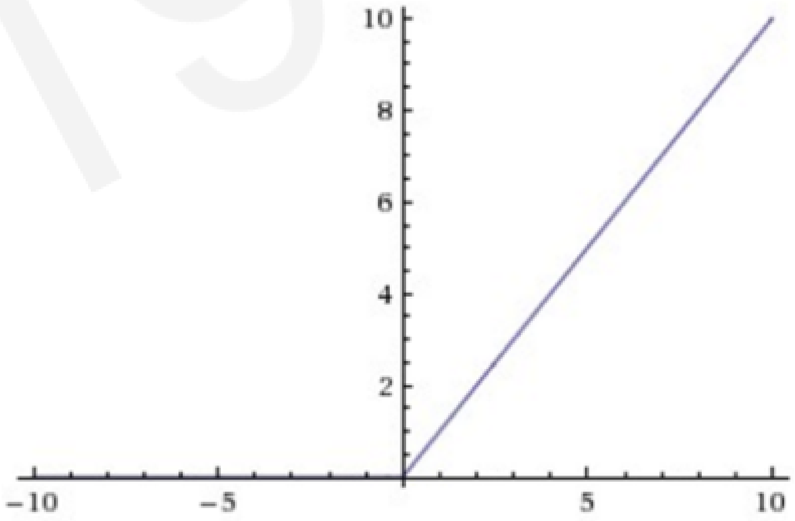
MIT 6.S191

Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



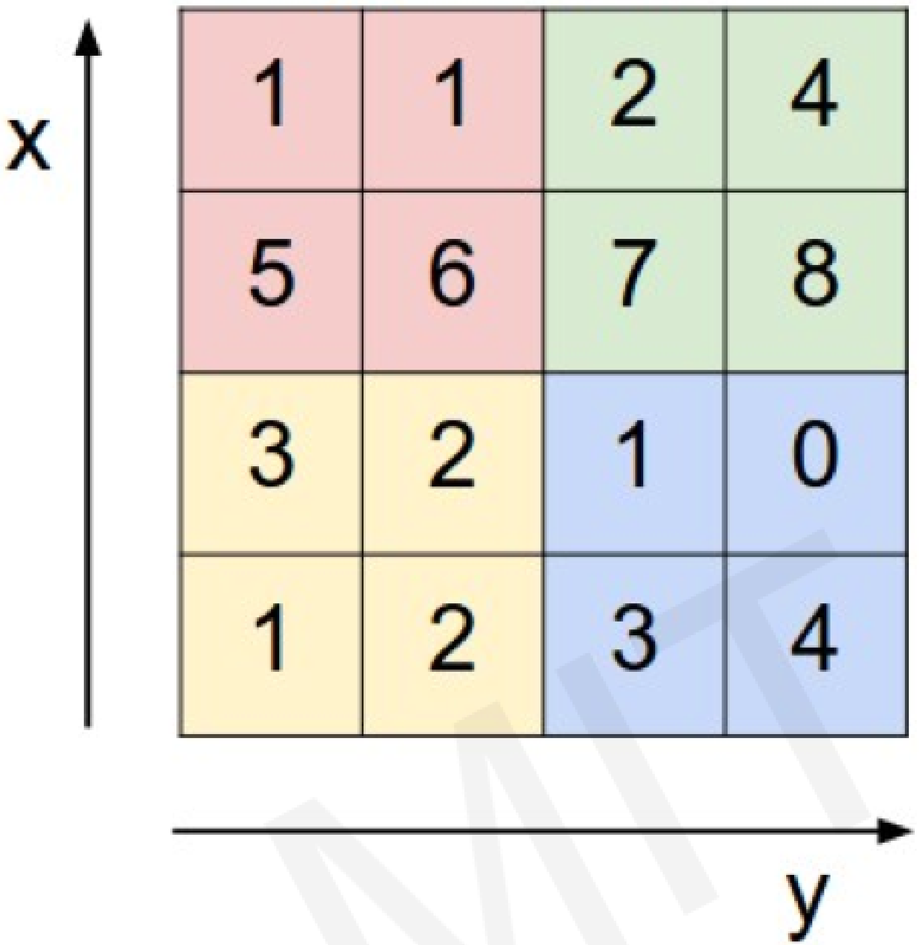
Rectified Linear Unit (ReLU)



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

`tf.keras.layers.ReLU`

Pooling



max pool with 2x2 filters and stride 2

```
tf.keras.layers.MaxPool2D(
    pool_size=(2,2),
    strides=2
)
```

6	8
3	4

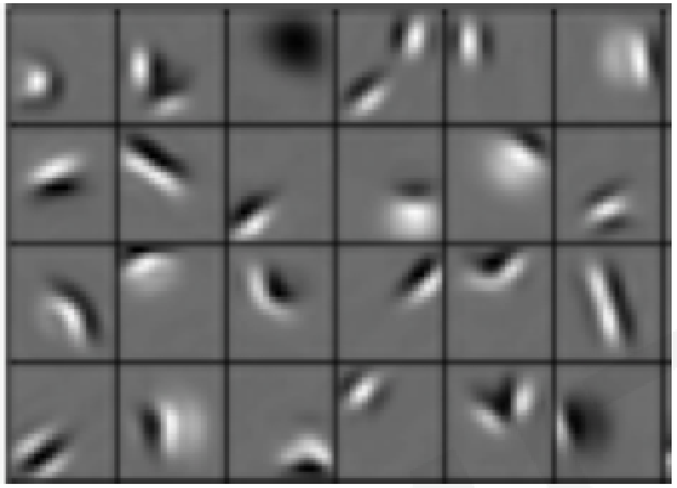
1) Reduced dimensionality
2) Spatial invariance

How else can we downsample and preserve spatial invariance?

MIT 6.S191

Representation Learning in Deep CNNs

Low level features



Edges, dark spots

Conv Layer 1

Mid level features

Eyes, ears, nose

Conv Layer 2

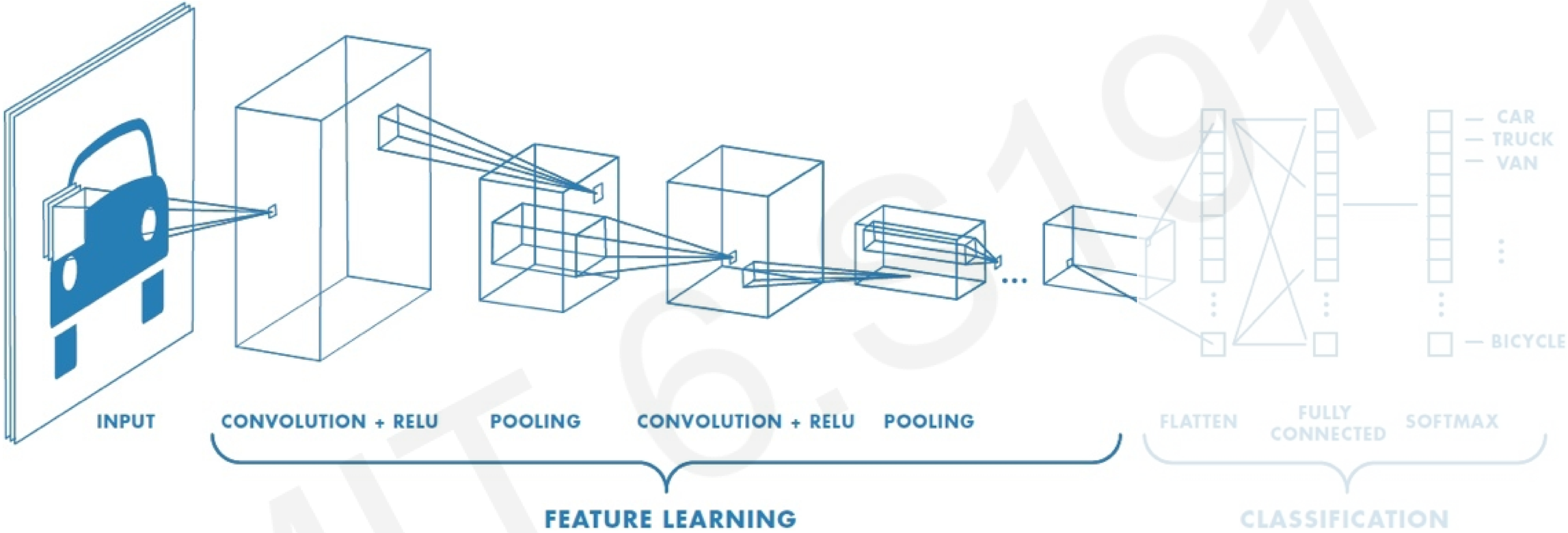
High level features

Facial structure

Conv Layer 3

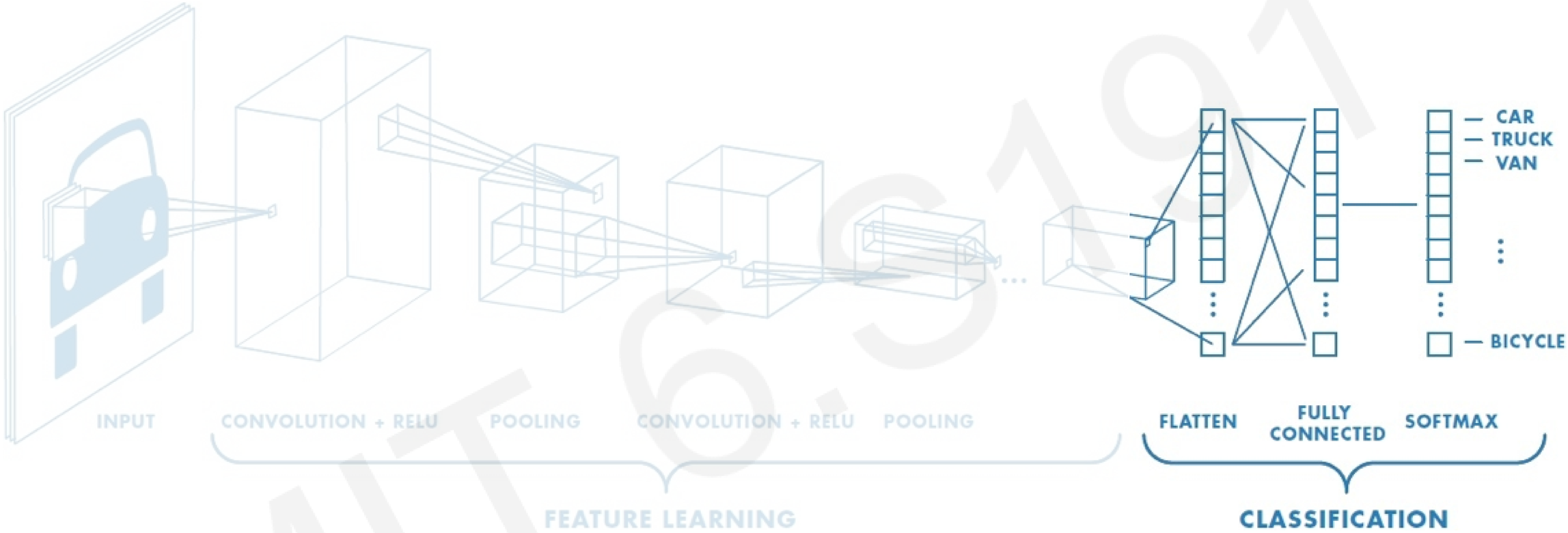
MIT 6.S191

CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNNs for Classification: Class Probabilities

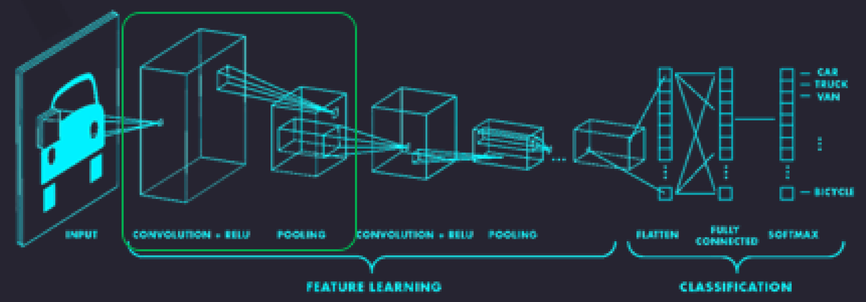


- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Putting it all together

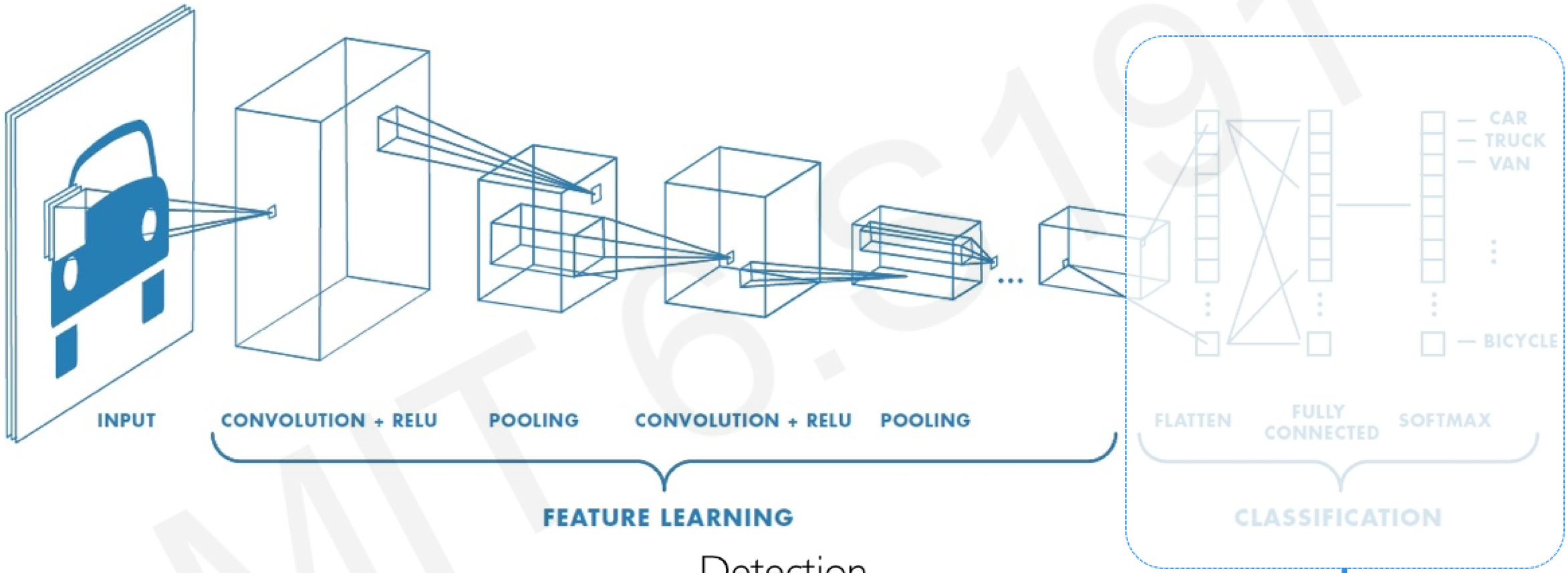
```
import tensorflow as tf

def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # second convolutional layer
        tf.keras.layers.Conv2D(64, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # fully connected classifier
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs
    ])
    return model
```



An Architecture for Many Applications

MIT 6.S191

An Architecture for Many Applications



Detection

Semantic segmentation

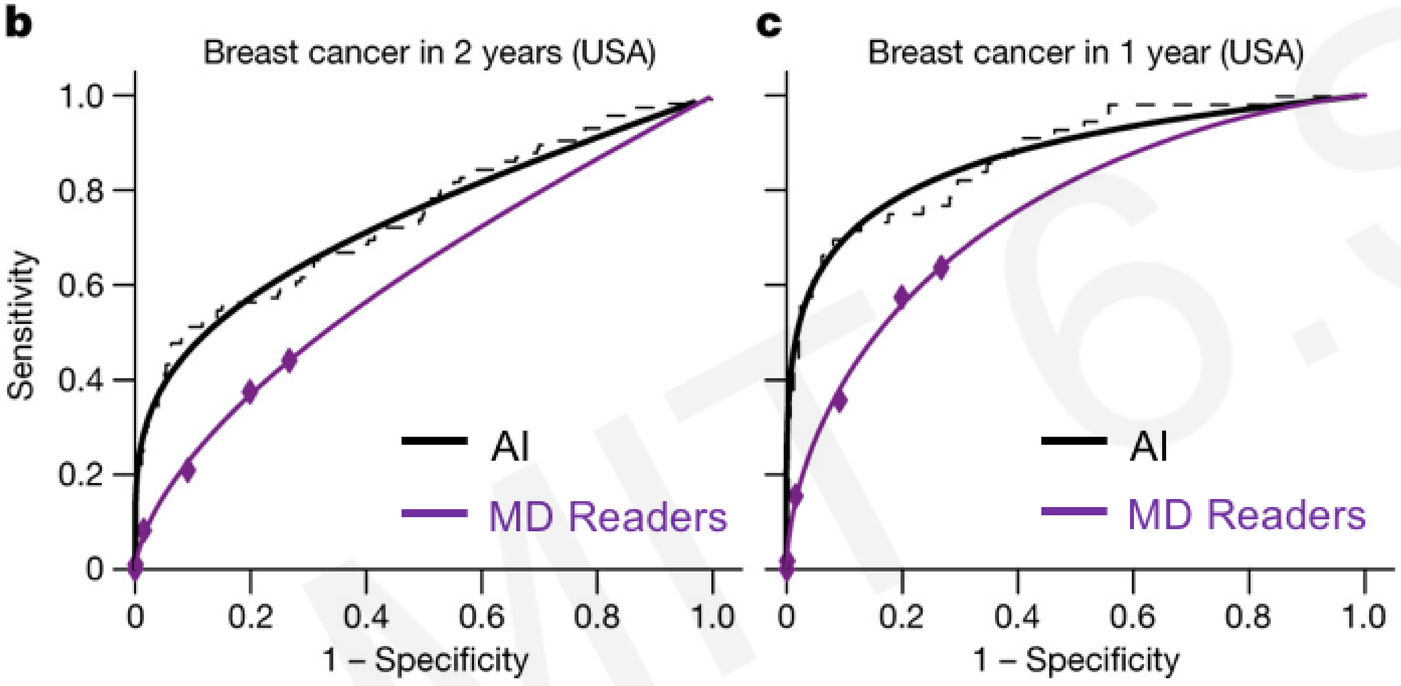
End-to-end robotic control

MIT 6.S191

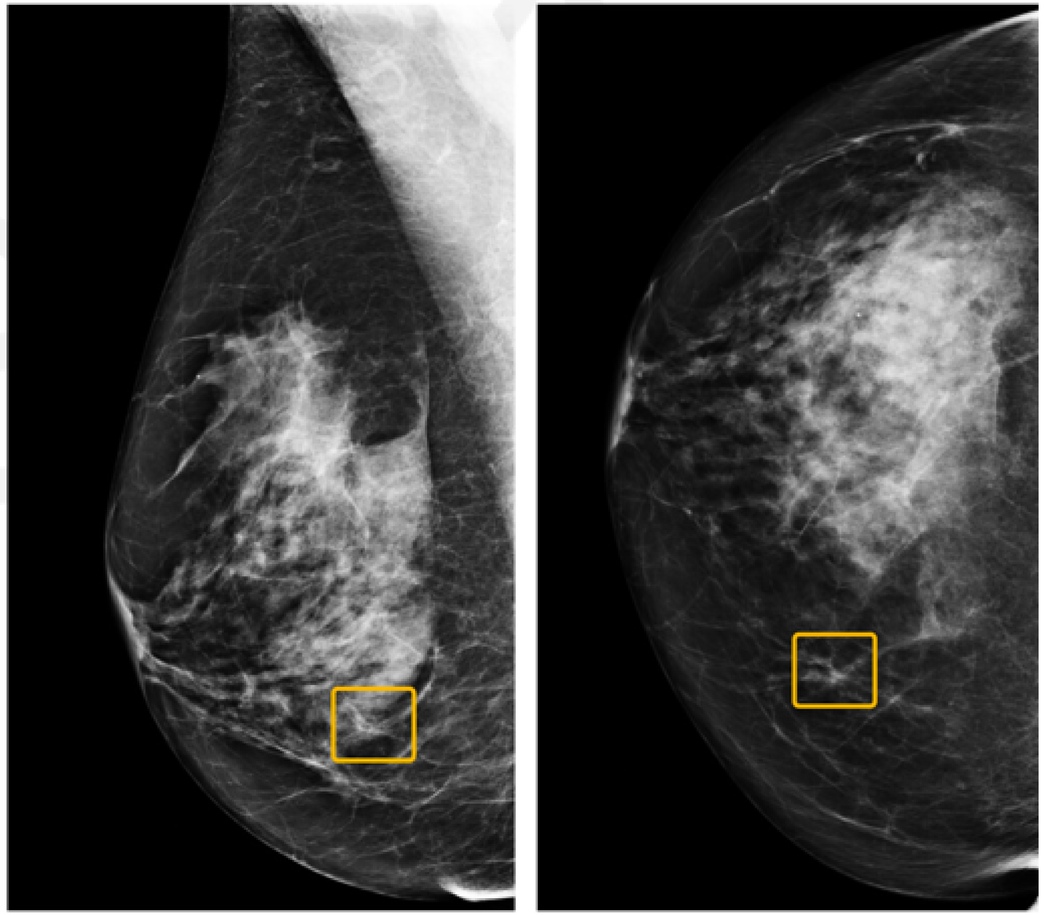
Detection: Breast Cancer Screening

International evaluation of an AI system for breast cancer screening

nature



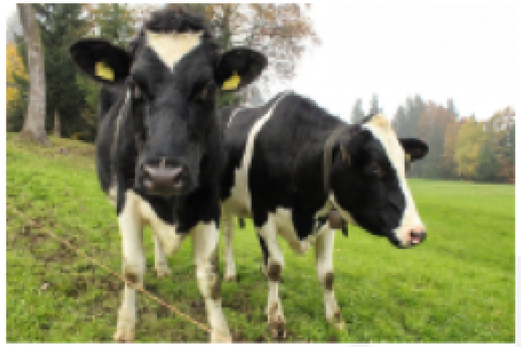
CNN-based system outperformed expert radiologists at detecting breast cancer from mammograms



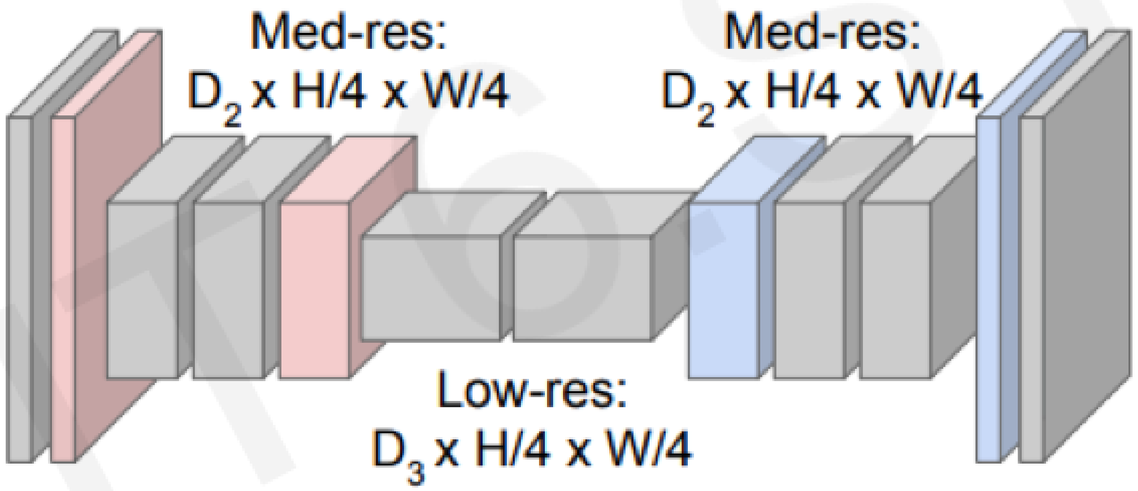
Breast cancer case missed by radiologist but detected by AI

Semantic Segmentation: Fully Convolutional Networks

FCN: Fully Convolutional Network.
Network designed with all convolutional layers,
with **downsampling** and **upsampling** operations



Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

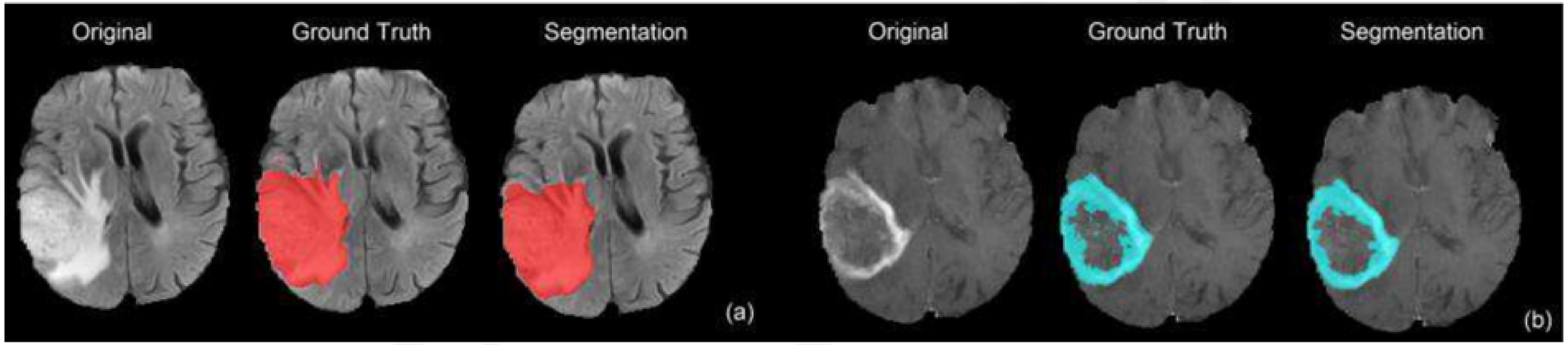
High-res:
D_1 x H/2 x W/2

Predictions:
H x W

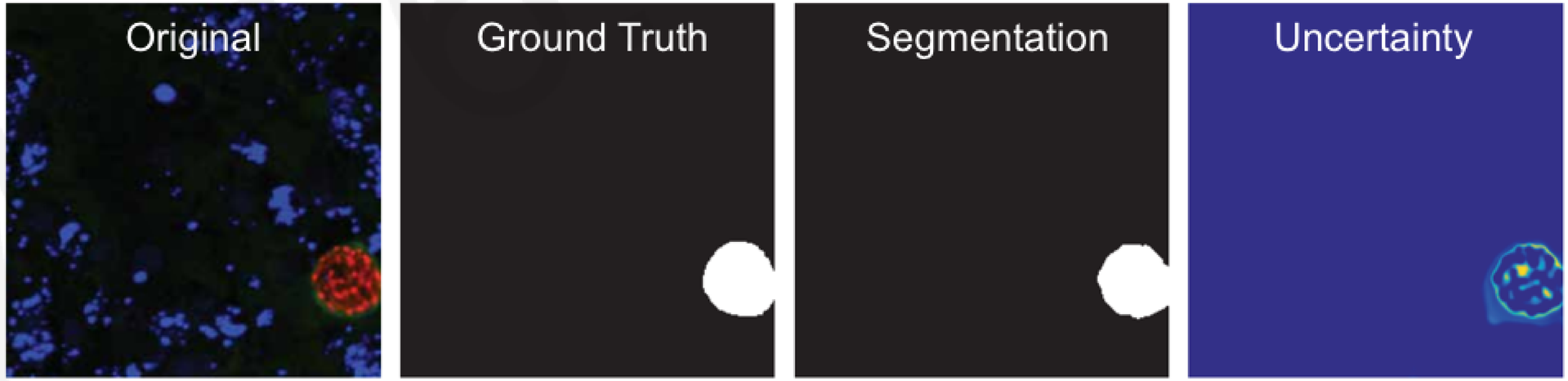
`tf.keras.layers.Conv2DTranspose`

Semantic Segmentation: Biomedical Image Analysis

Brain Tumors
Dong+ *MIUA* 2017.



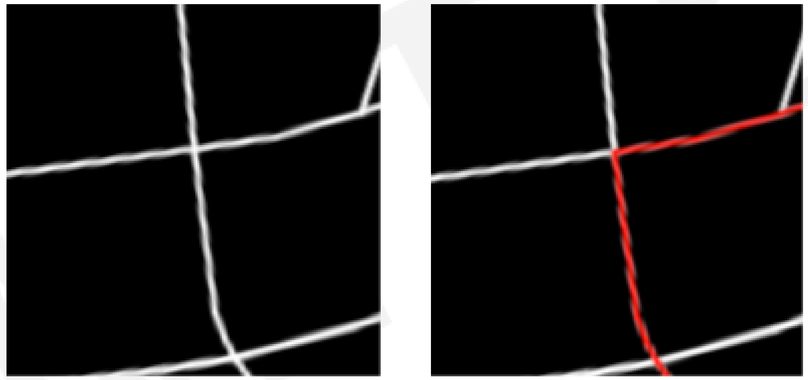
Malaria Infection
Soleimany+ *arXiv* 2019.



Self-Driving Cars: Navigation from Visual Perception

Raw Perception
I
(ex. camera)

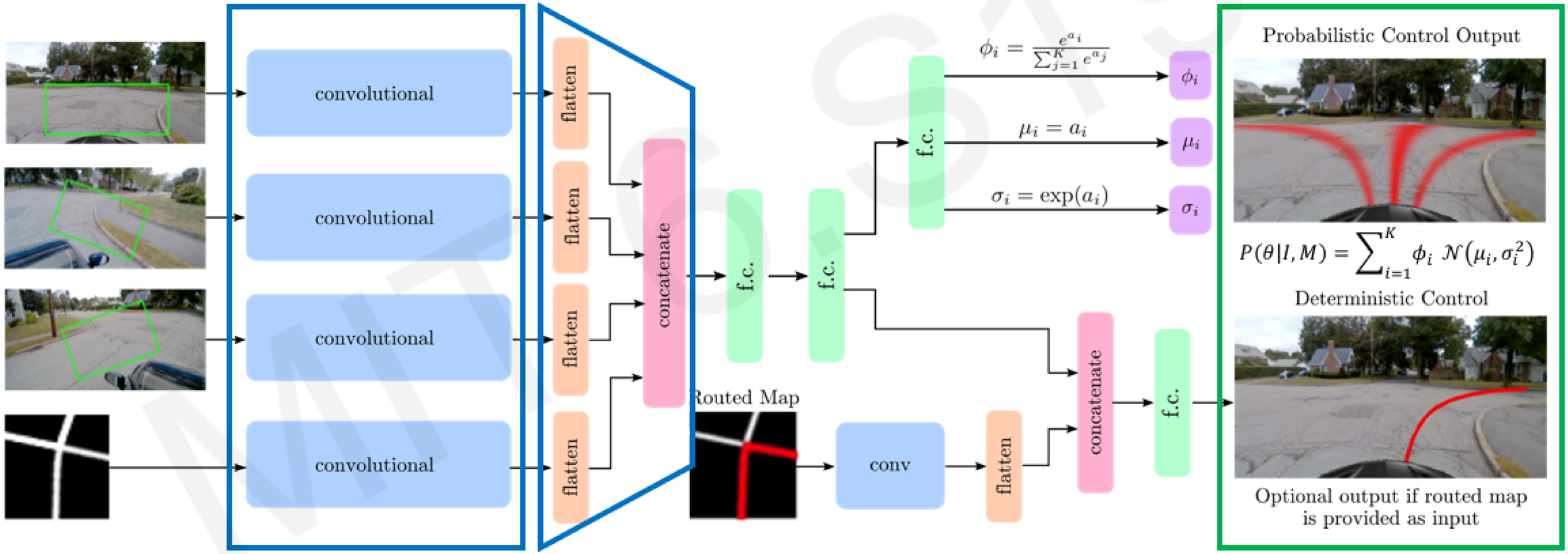
Coarse Maps
M
(ex. GPS)



Possible Control Commands

End-to-End Framework for Autonomous Navigation

Entire model is trained end-to-end **without any human labelling or annotations**



MIT 6.S191

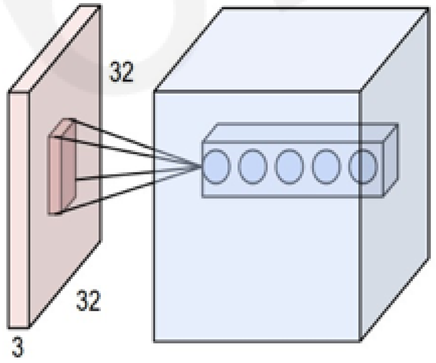
Deep Learning for Computer Vision: Summary

Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction

CNNs

- CNN architecture
- Application to classification
- ImageNet



Applications

- Segmentation, image captioning, control
- Security, medicine, robotics

MIT 6.S191

Supervised vs unsupervised learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$$x \rightarrow y$$

Examples: Classification, regression, object detection, semantic segmentation, etc.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn the *hidden* or *underlying structure* of the data

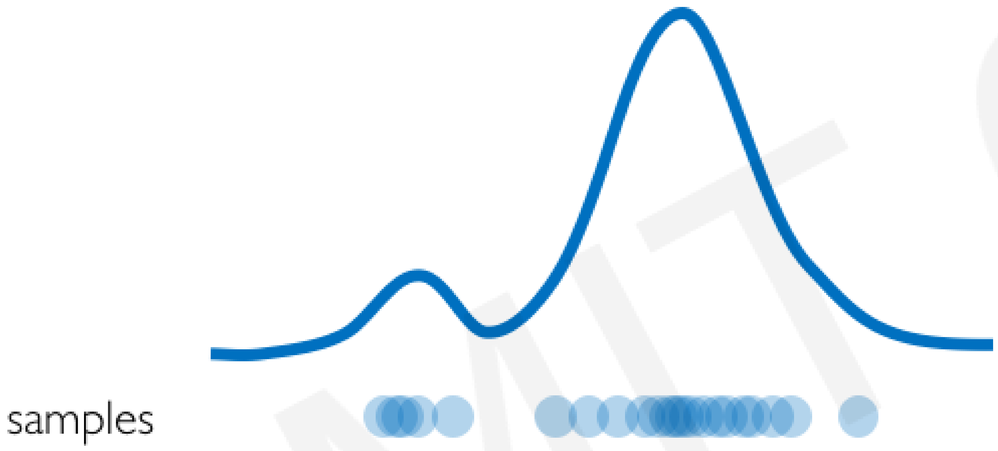
Examples: Clustering, feature or dimensionality reduction, etc.

MIT 6.S191

Generative modeling

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

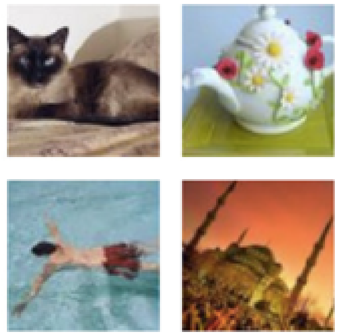
Density Estimation



Sample Generation

Input samples

Training data $\sim P_{data}(x)$



Generated samples

Generated $\sim P_{model}(x)$

How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

MIT 6.S191

Why generative models? Debiasing

Capable of uncovering **underlying features** in a dataset

Homogeneous skin color, pose

VS

Diverse skin color, pose, illumination

How can we use this information to create fair and representative datasets?

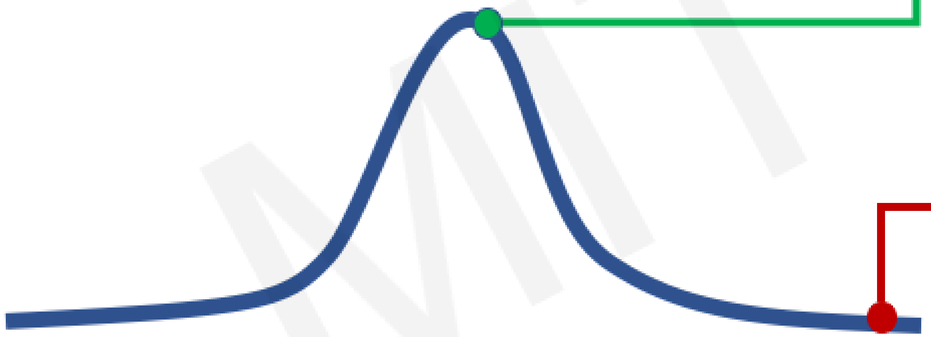
6.S191 Lab

Why generative models? Outlier detection

- **Problem:** How can we detect when we encounter something new or rare?
- **Strategy:** Leverage generative models, detect outliers in the distribution
- Use outliers during training to improve even more!

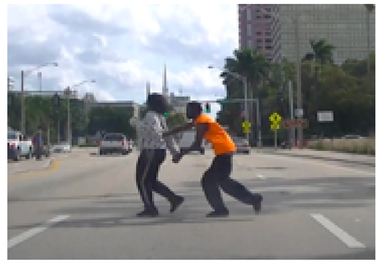
95% of Driving Data:
(1) sunny, (2) highway, (3) straight road

Detect outliers to avoid unpredictable behavior when training



Edge Cases

Harsh Weather

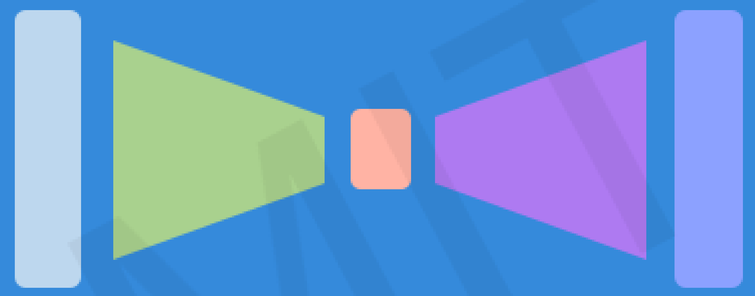


Pedestrians

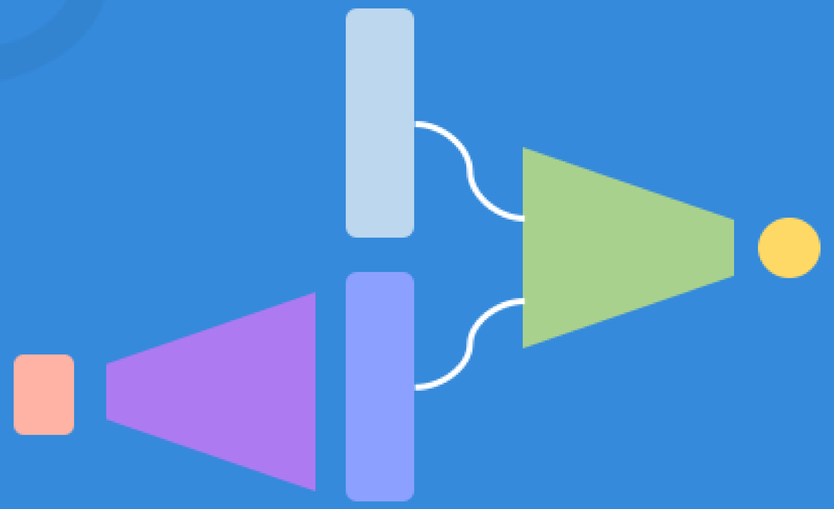
MIT 6.S191

Latent variable models

Autoencoders and Variational Autoencoders (VAEs)



Generative Adversarial Networks (GANs)



MIT 6.S191

What is a latent variable?

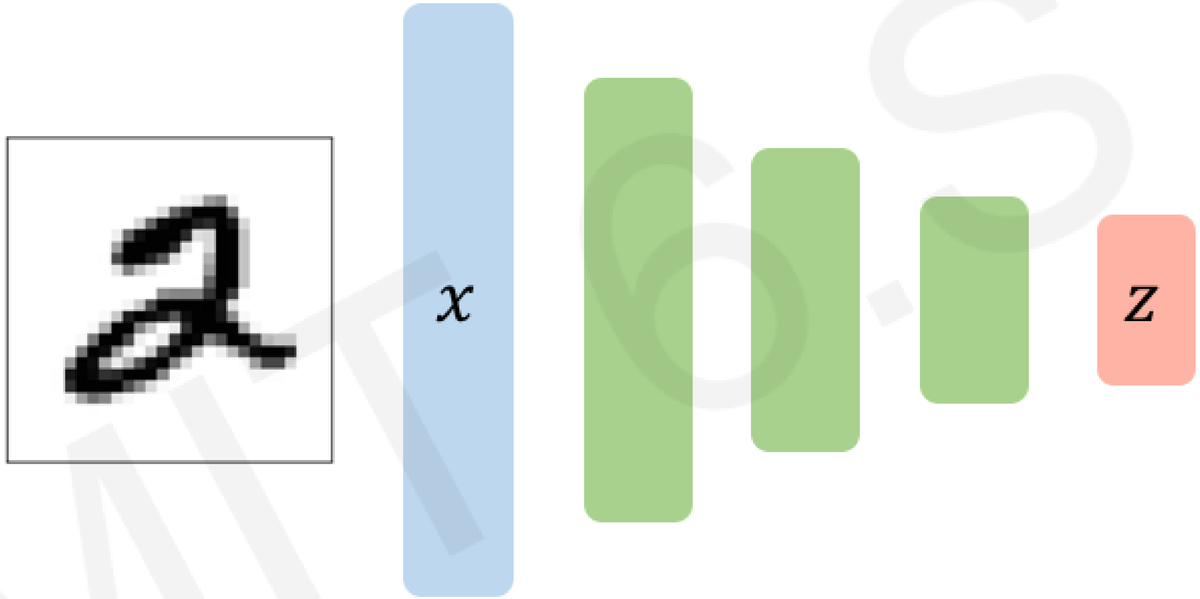
Can we learn the **true explanatory factors**, e.g. latent variables, from only observed data?

Autoencoders

MIT 6.S191

Autoencoders: background

Unsupervised approach for learning a **lower-dimensional** feature representation from unlabeled training data



Why do we care about a low-dimensional z?

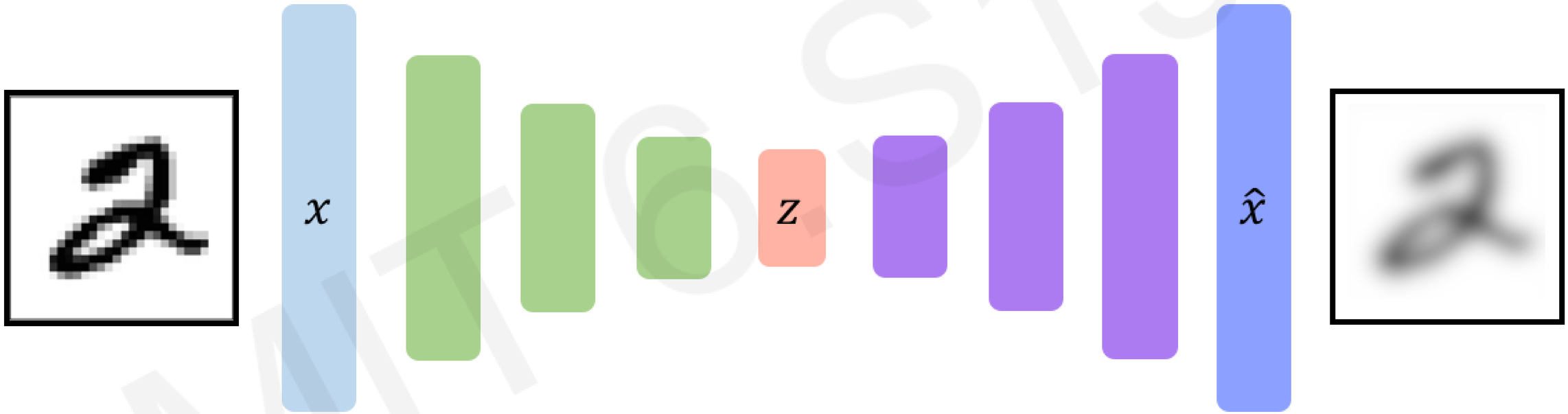
"Encoder" learns mapping from the data, x, to a low-dimensional latent space, z

MIT 6.S191

Autoencoders: background

How can we learn this latent space?

Train the model to use these features to **reconstruct the original data**



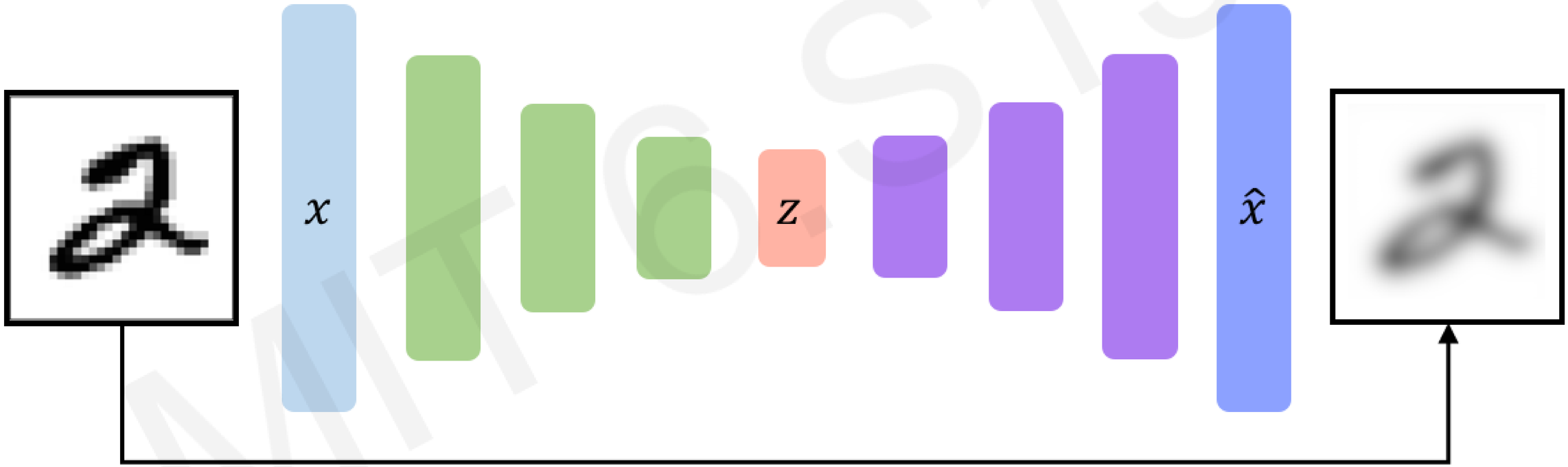
"Decoder" learns mapping back from latent, z, to a reconstructed observation, $\hat{x}$

MIT 6.S191

Autoencoders: background

How can we learn this latent space?

Train the model to use these features to **reconstruct the original data**



$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

Loss function doesn't use any labels!!

MIT 6.S191

Dimensionality of latent space → reconstruction quality

Autoencoding is a form of compression!
Smaller latent space will force a larger training bottleneck

2D latent space

5D latent space

Ground Truth

Autoencoders for representation learning

Bottleneck hidden layer forces network to learn a compressed latent representation

Reconstruction loss forces the latent representation to capture (or encode) as much "information" about the data as possible

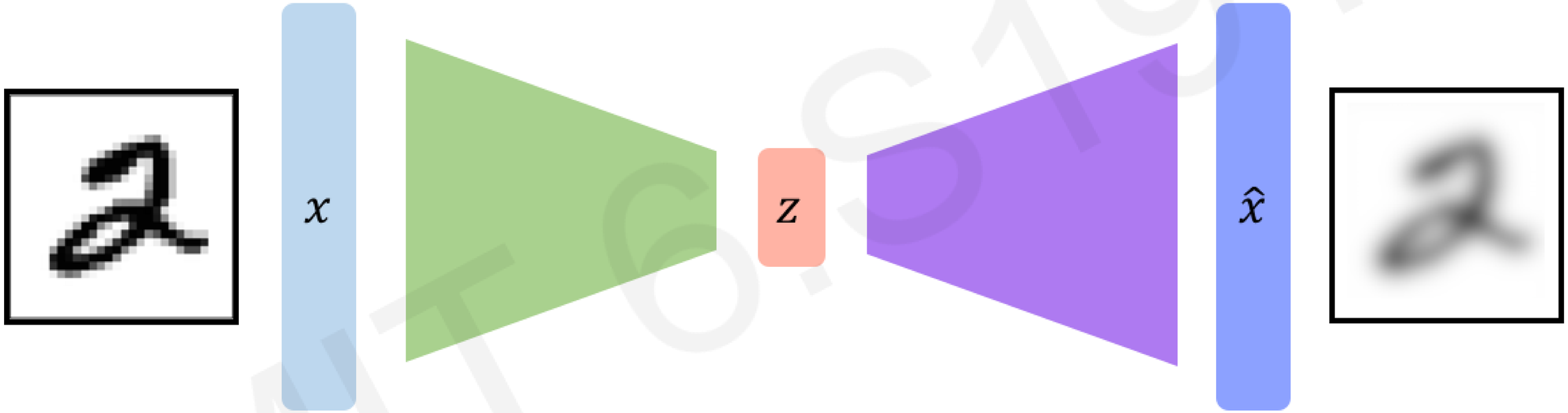
Autoencoding = **Auto**matically **encoding** data

MIT 6.S191

Variational Autoencoders (VAEs)

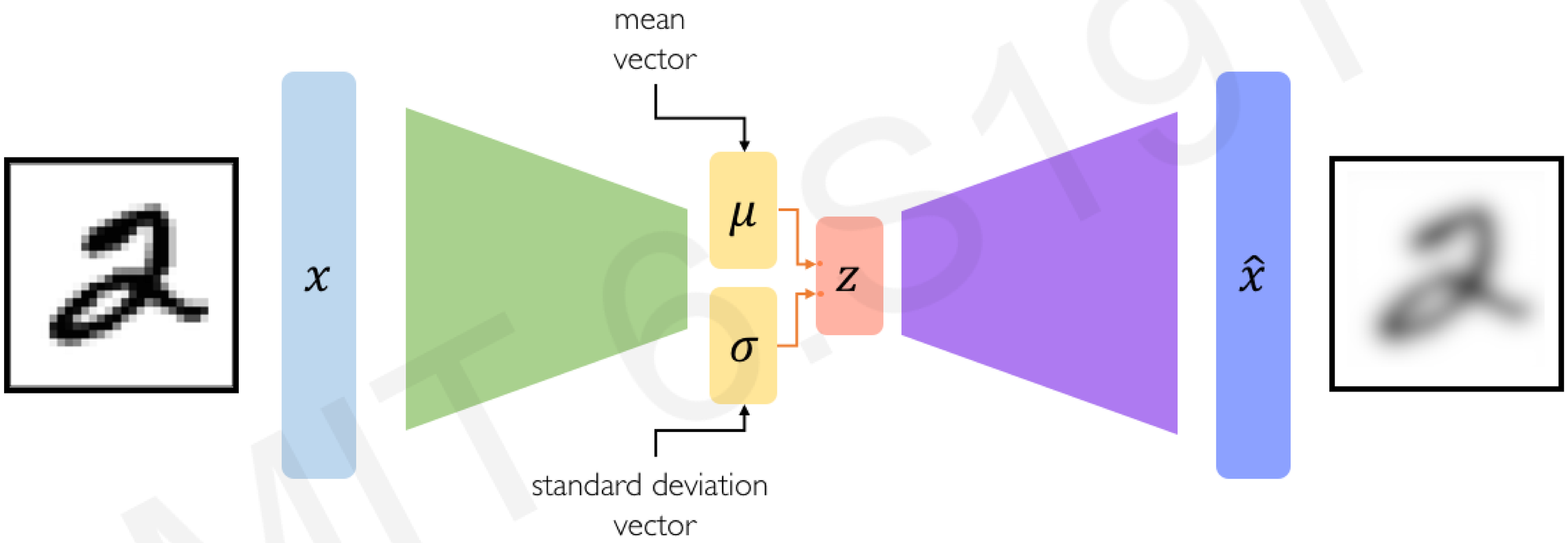
MIT 6.S191

Traditional autoencoders



MIT 6.S191

VAEs: key difference with traditional autoencoder

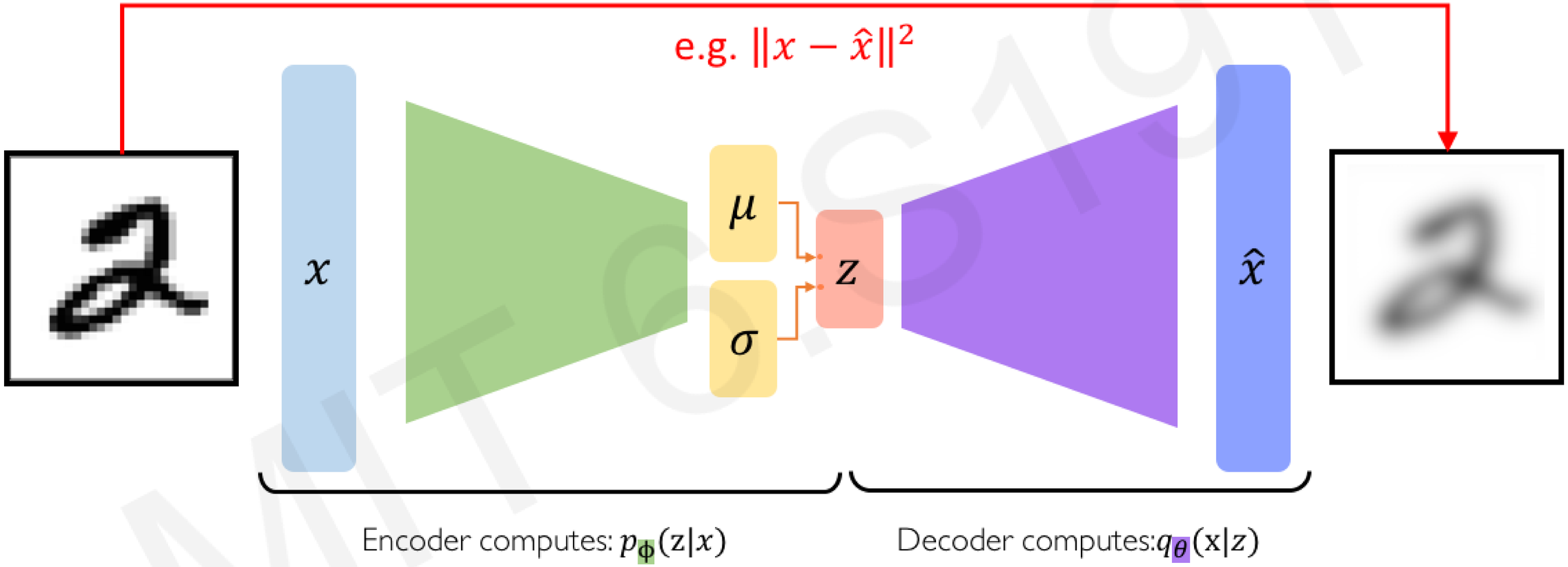


Variational autoencoders are a probabilistic twist on autoencoders!

Sample from the mean and standard dev. to compute latent sample

MIT 6.S191

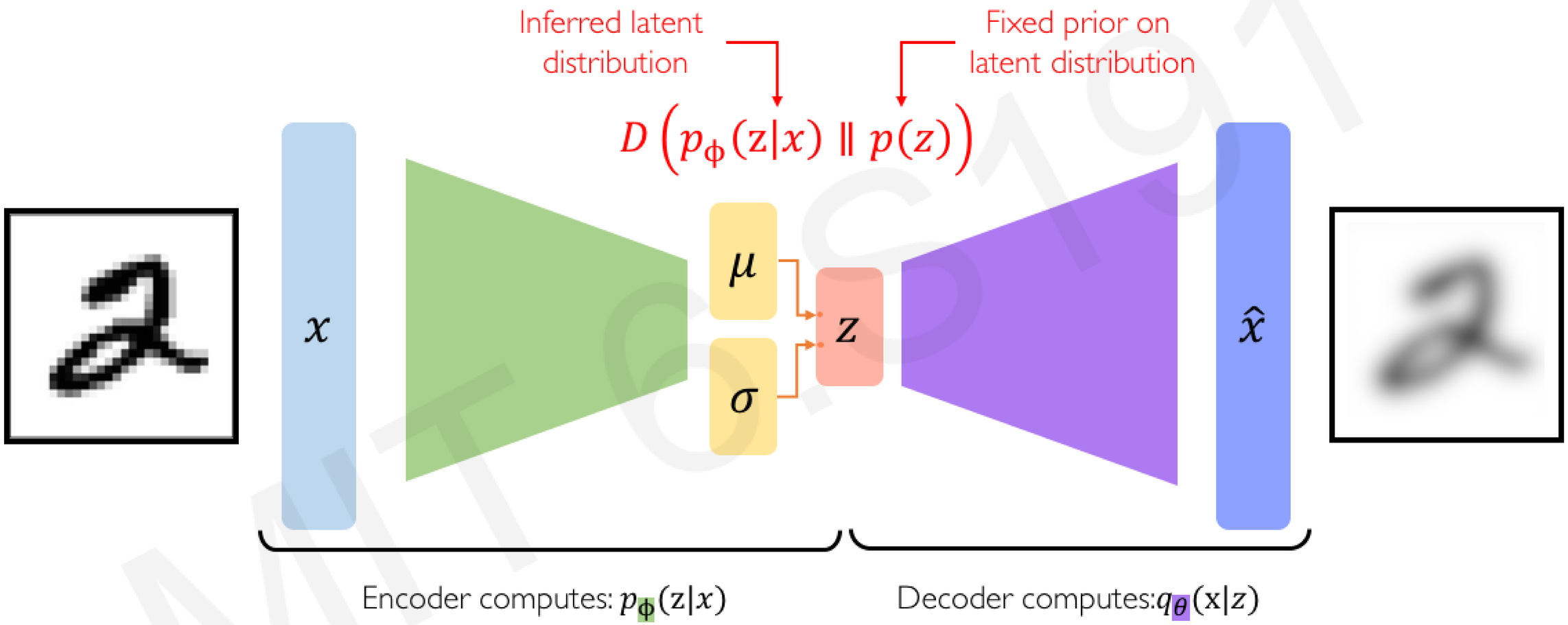
VAE optimization



$$\mathcal{L}(\phi, \theta, x) = \boxed{\text{(reconstruction loss)}} + \text{(regularization term)}$$

MIT 6.S191

VAE optimization



$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \boxed{\text{(regularization term)}}$$

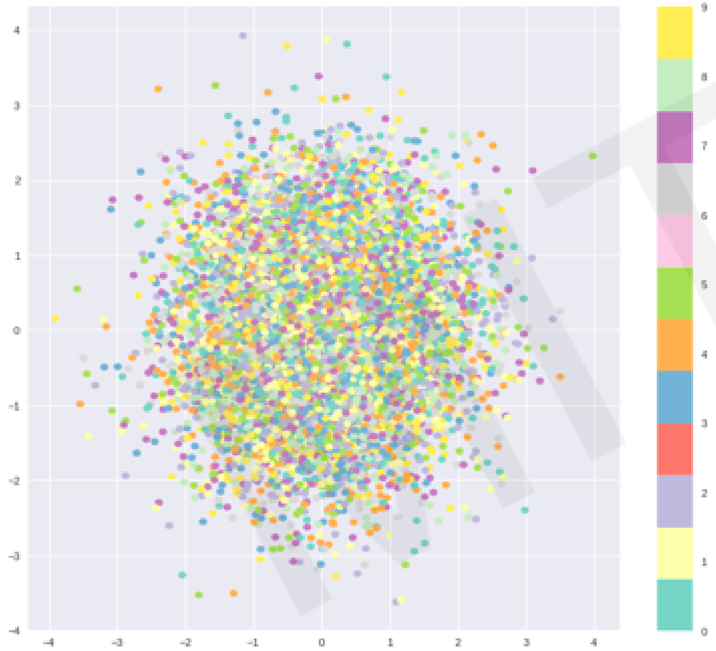
MIT 6.S191

Priors on the latent distribution

$$D\left(p_\phi(z|x) \parallel p(z)\right)$$

Inferred latent distribution

Fixed prior on latent distribution



Common choice of prior:

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

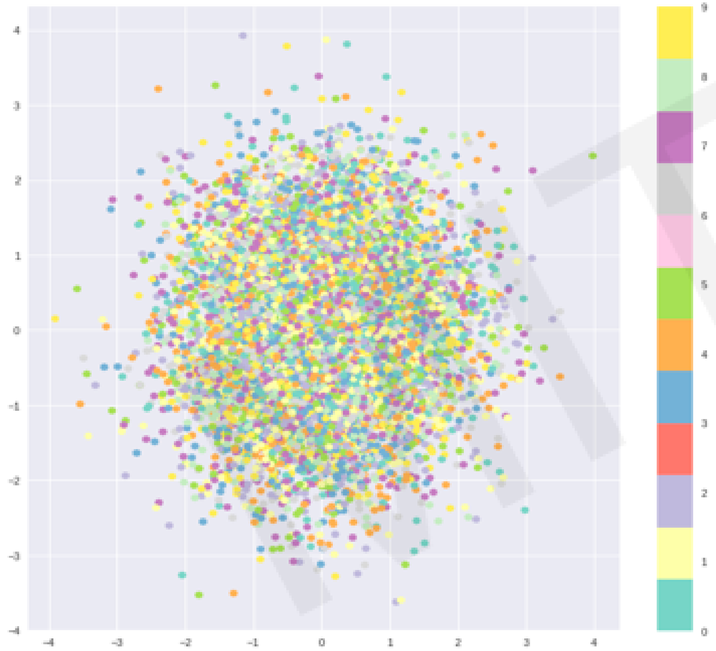
- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (ie. memorizing the data)

Priors on the latent distribution

$$D\left(p_\phi(z|x) \parallel p(z)\right)$$

$$= -\frac{1}{2}\sum_{j=0}^{k-1}\left(\sigma_j + \mu_j^2 - 1 - \log\sigma_j\right)$$

KL-divergence between the two distributions



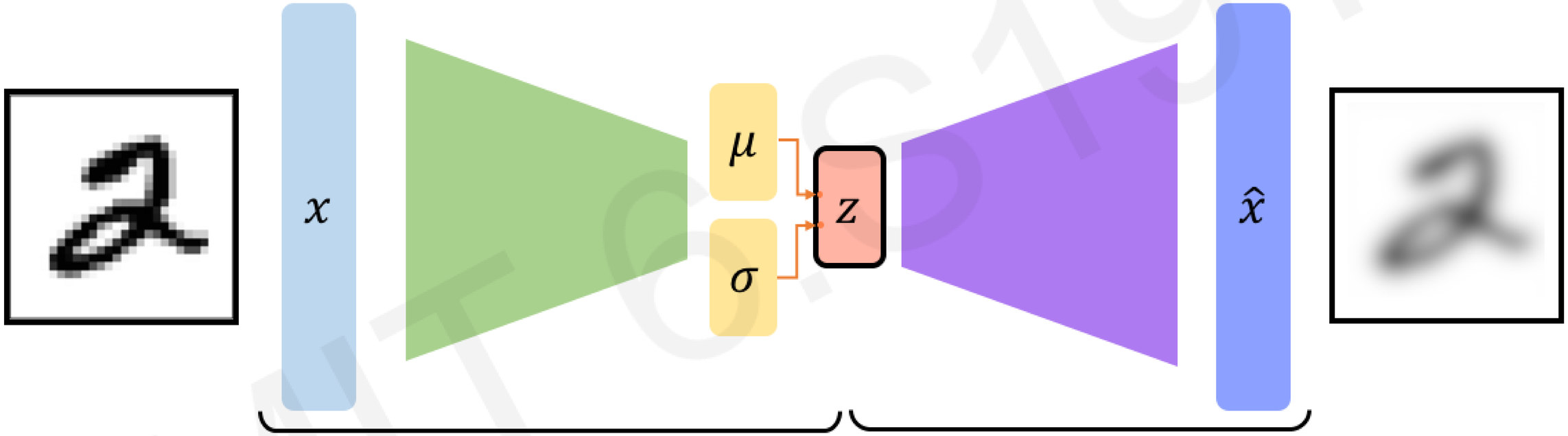
Common choice of prior:

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (ie. memorizing the data)

VAEs computation graph

Problem: We cannot backpropagate gradients through sampling layers!



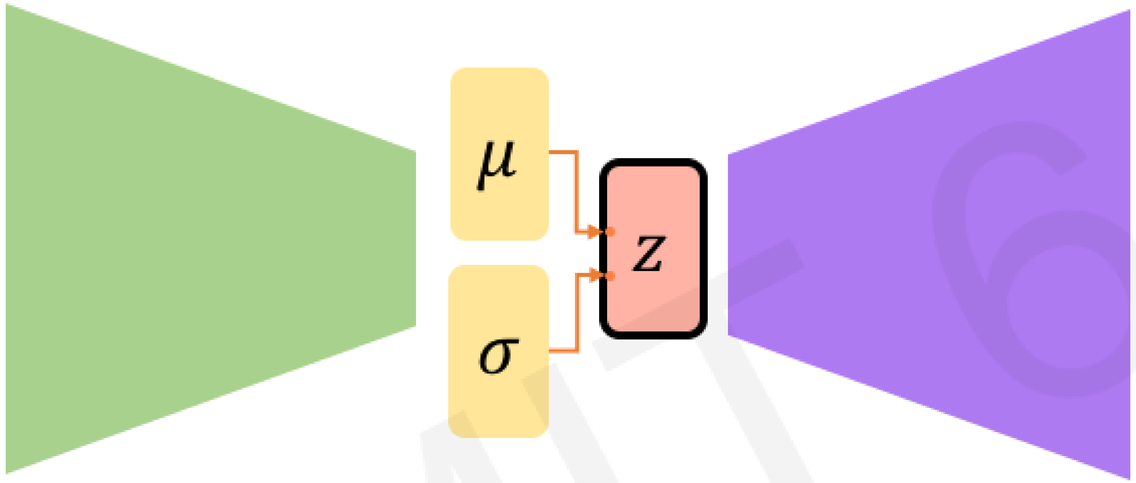
Encoder computes: $p_\phi(z|x)$

Decoder computes: $q_\theta(x|z)$

$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \text{(regularization term)}$$

MIT 6.S191

Reparametrizing the sampling layer



Key Idea:

$$\cancel{z \sim \mathcal{N}(\mu, \sigma^2)}$$

Consider the sampled latent vector z as a sum of

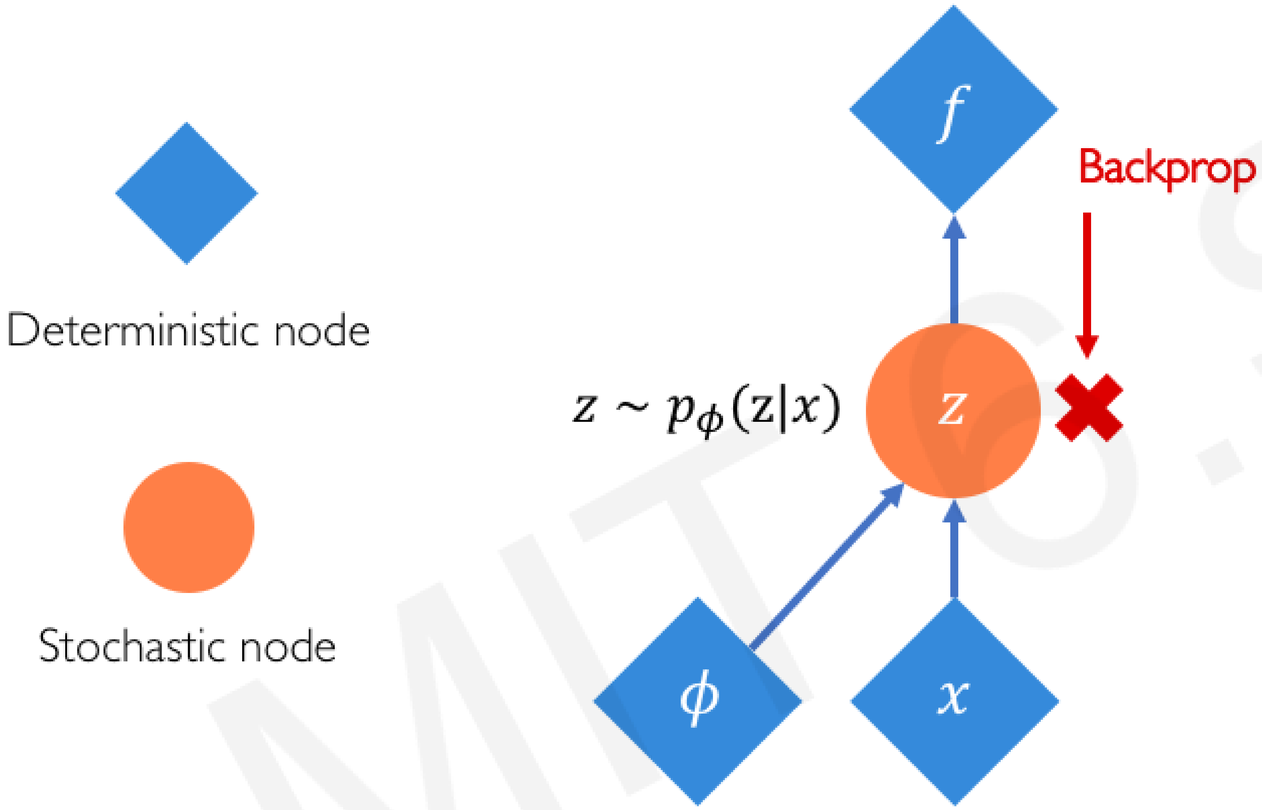
- a fixed μ vector,
- and fixed σ vector, scaled by random constants drawn from the prior distribution

$$\Rightarrow z = \mu + \sigma \odot \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0,1)$

MIT 6.S191

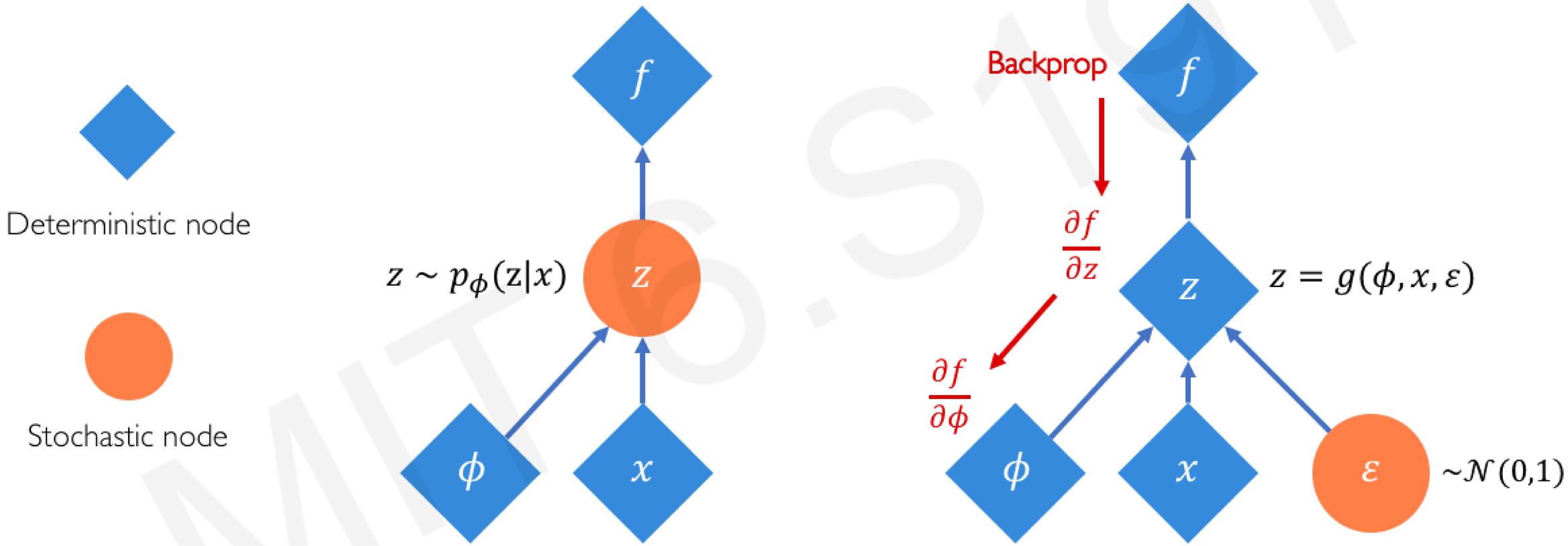
Reparametrizing the sampling layer



Original form

MIT 6.S191

Reparametrizing the sampling layer



Original form

Reparametrized form

MIT 6.S191

VAEs: Latent perturbation

Slowly increase or decrease a **single latent variable**
Keep all other variables fixed

Head pose

Different dimensions of z encodes **different interpretable latent features**

MIT 6.S191

VAEs: Latent perturbation

Smile

Head pose

Ideally, we want latent variables that are uncorrelated with each other

Enforce diagonal prior on the latent variables to encourage independence

Disentanglement

Why generative models? Debiasing

Capable of uncovering **underlying latent variables** in a dataset

Homogeneous skin color, pose

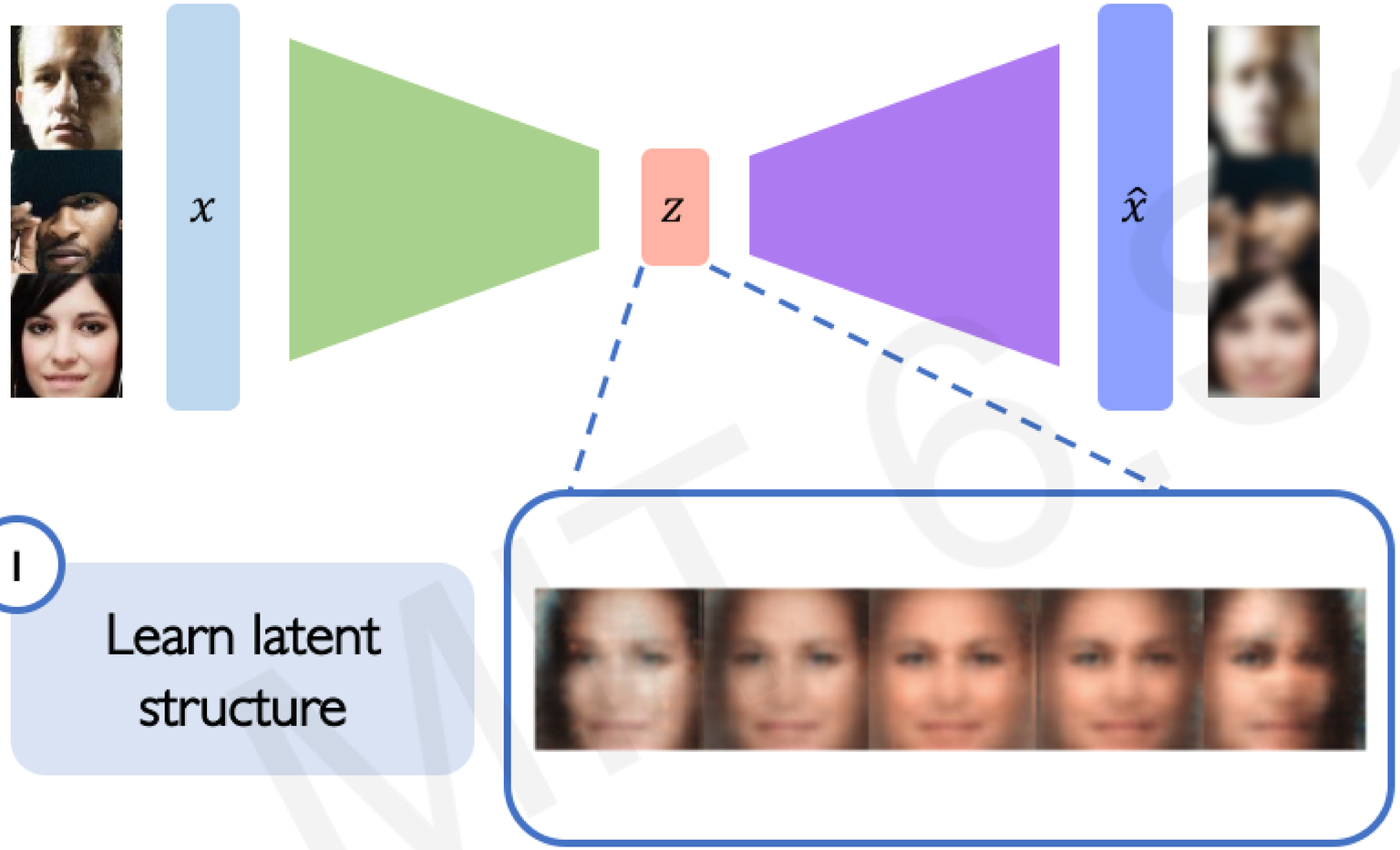
VS

Diverse skin color, pose, illumination

How can we use latent distributions to create fair and representative datasets?

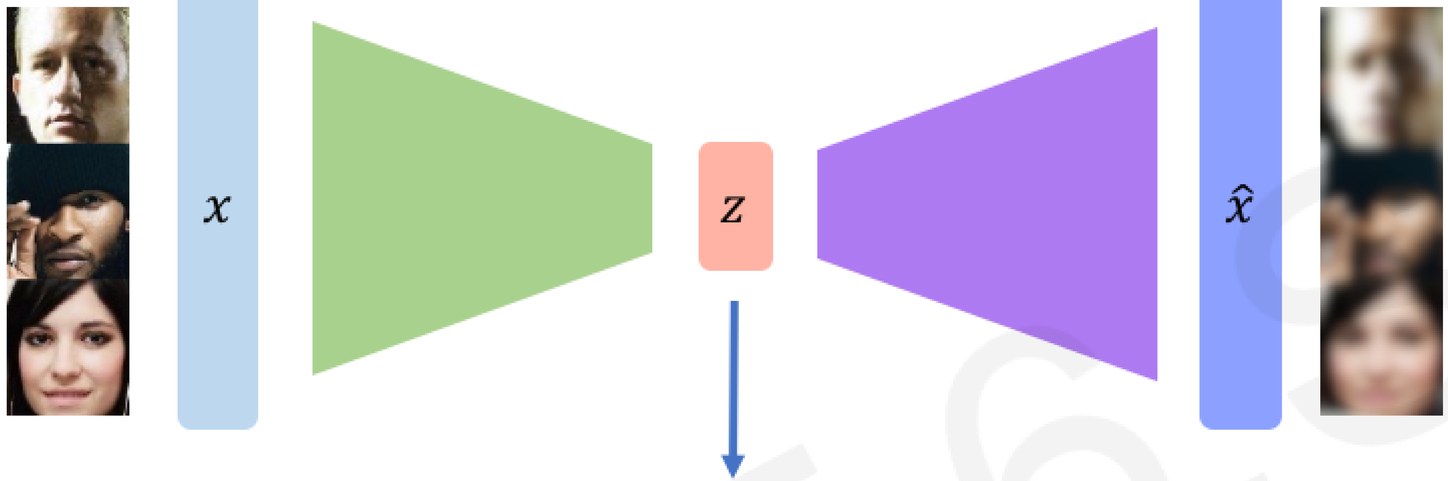
6.S191 Lab

Mitigating bias through learned latent structure



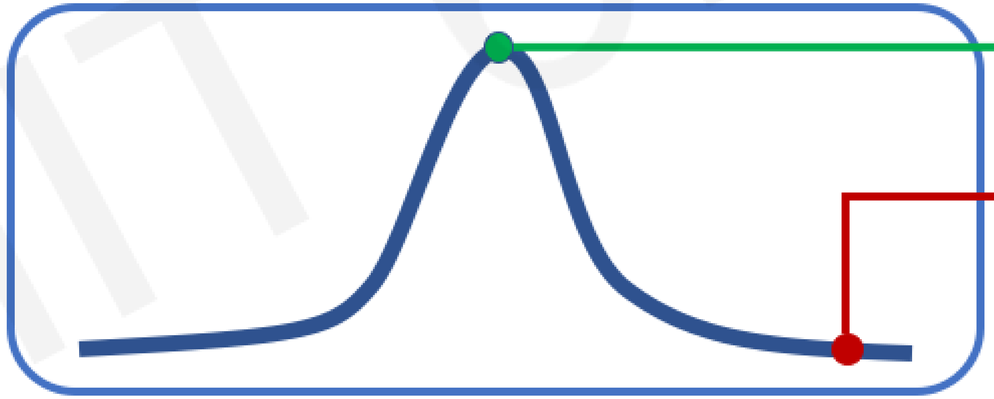
MIT 6.S191

Mitigating bias through learned latent structure



2

Estimate distribution

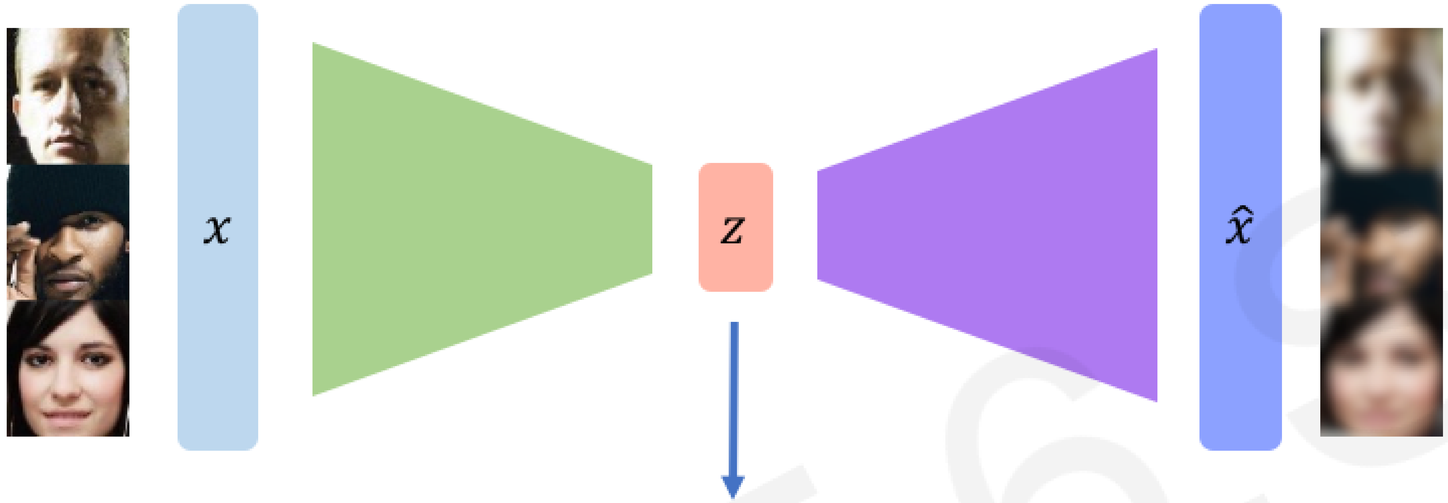


Homogeneous skin color, pose

Diverse skin color, pose, illumination

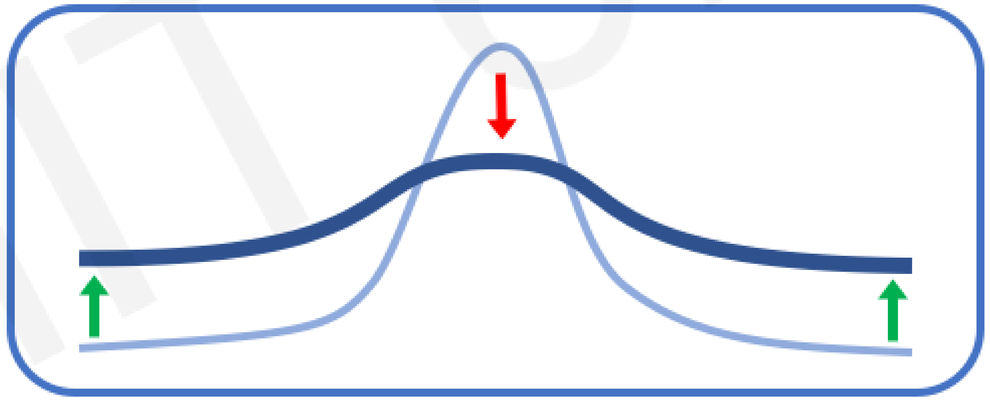
MIT 6.S191

Mitigating bias through learned latent structure



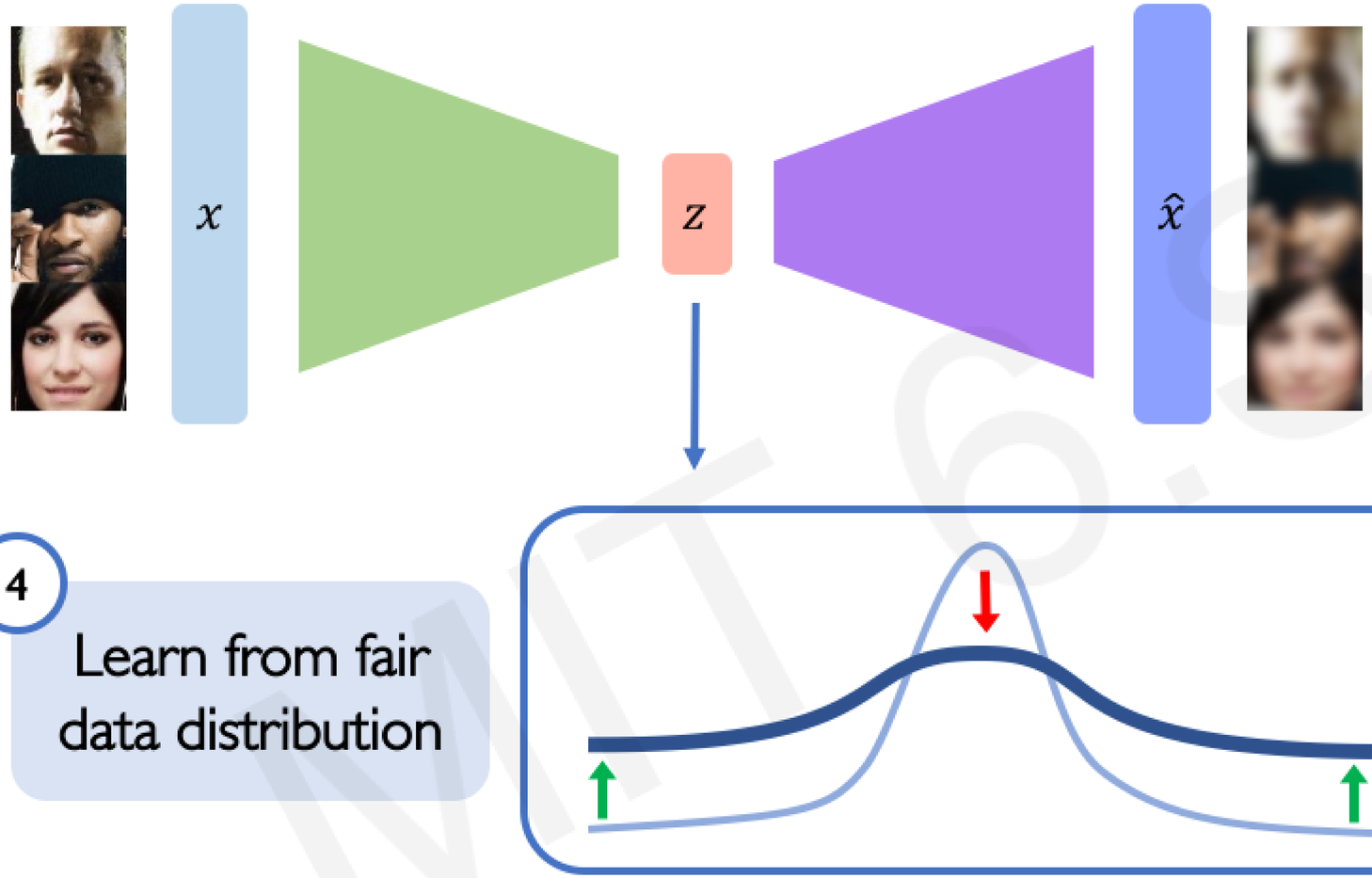
3

Adaptively resample data



MIT 6.S191

Mitigating bias through learned latent structure



4

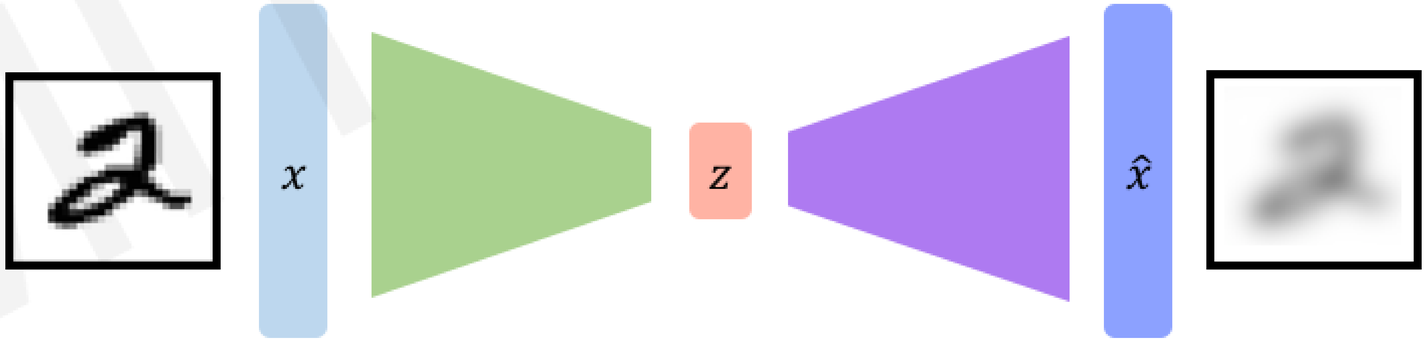
Learn from fair data distribution

Latent distributions used to create fair and representative dataset

6.S191 Lab

VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples



MIT 6.S191

Generative Adversarial Networks (GANs)

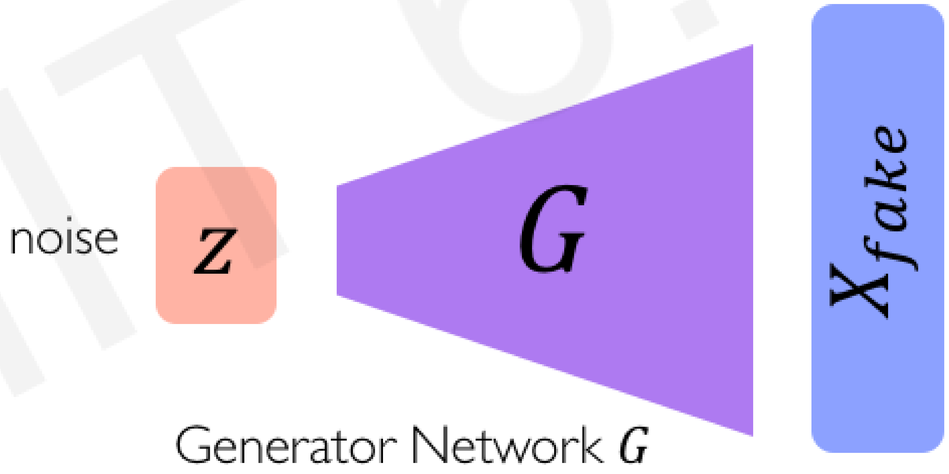
MIT 6.S191

What if we just want to sample?

Idea: don't explicitly model density, and instead just sample to generate new instances.

Problem: want to sample from complex distribution – can't do this directly!

Solution: sample from something simple (noise), learn a transformation to the training distribution.



"fake" sample from the training distribution

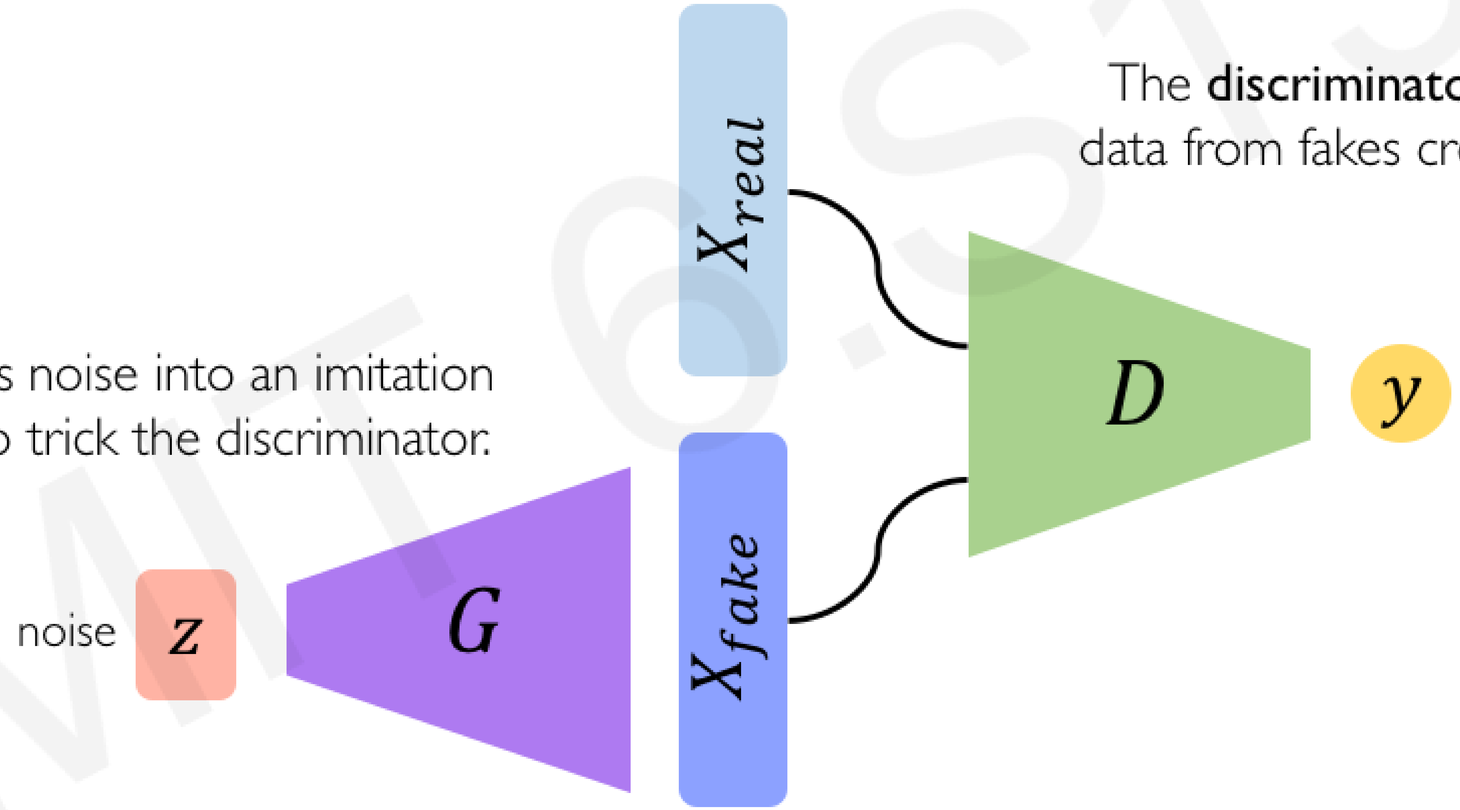
MIT 6.S191

Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.

The **discriminator** tries to identify real data from fakes created by the generator.

The **generator** turns noise into an imitation of the data to try to trick the discriminator.



MIT 6.S191

Intuition behind GANs

Discriminator looks at both real data and fake data created by the generator.

Discriminator

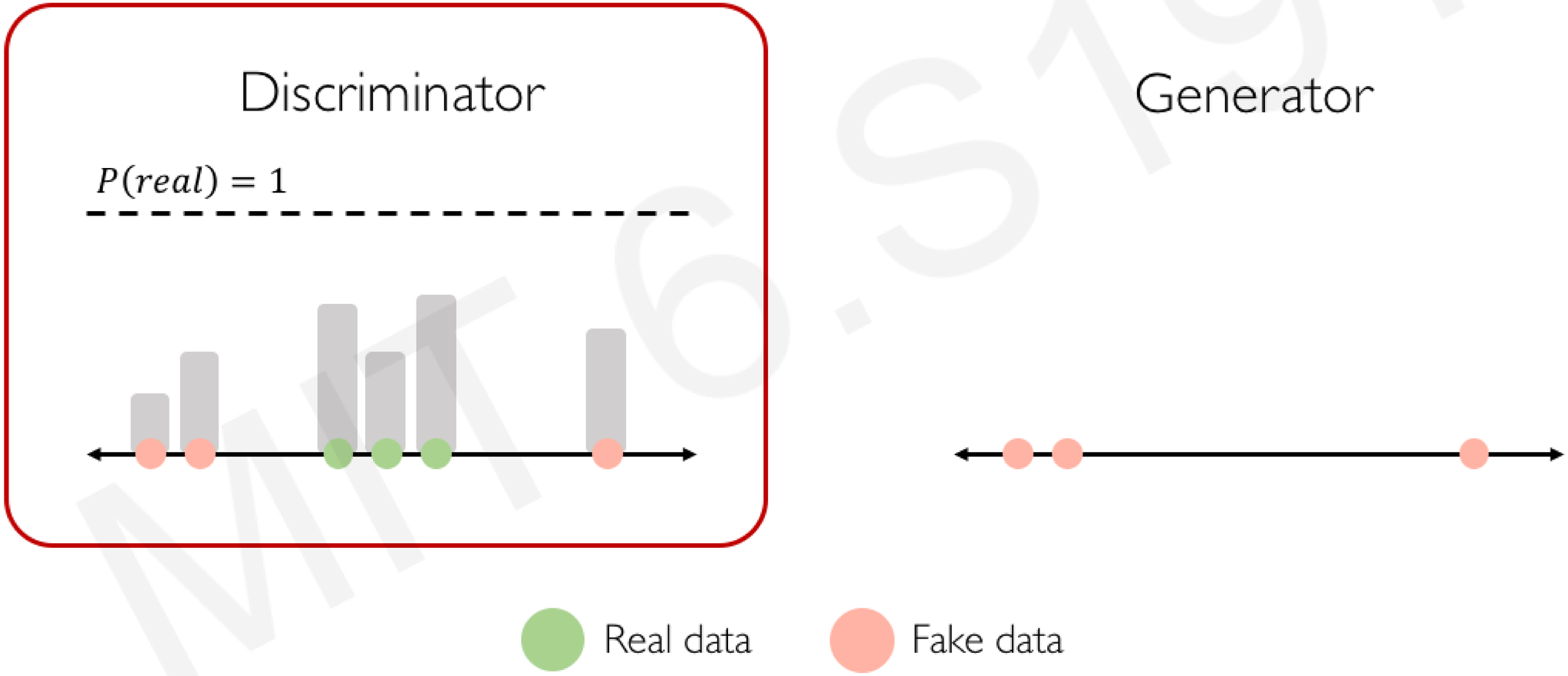
Generator

Real data Fake data

MIT 6.S191

Intuition behind GANs

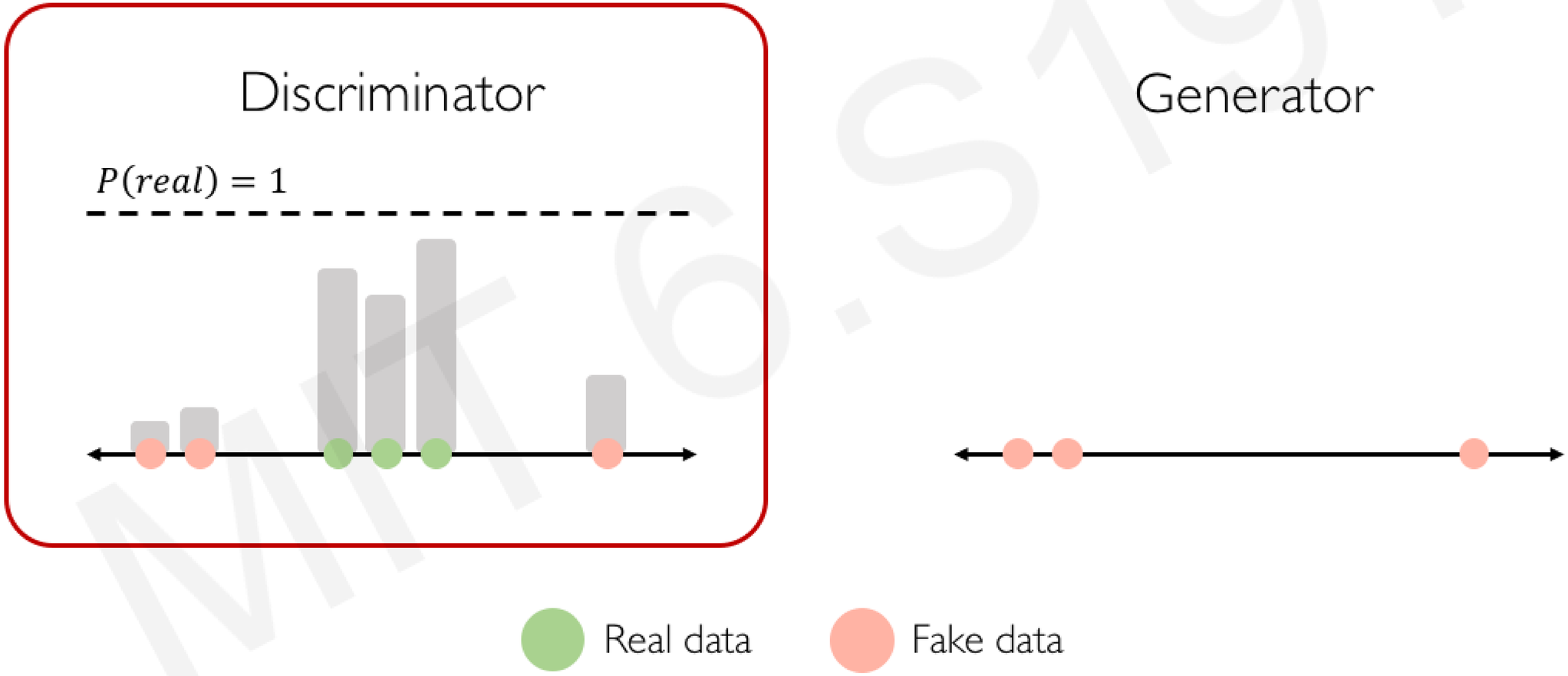
Discriminator tries to predict what's real and what's fake.



MIT 6.S191

Intuition behind GANs

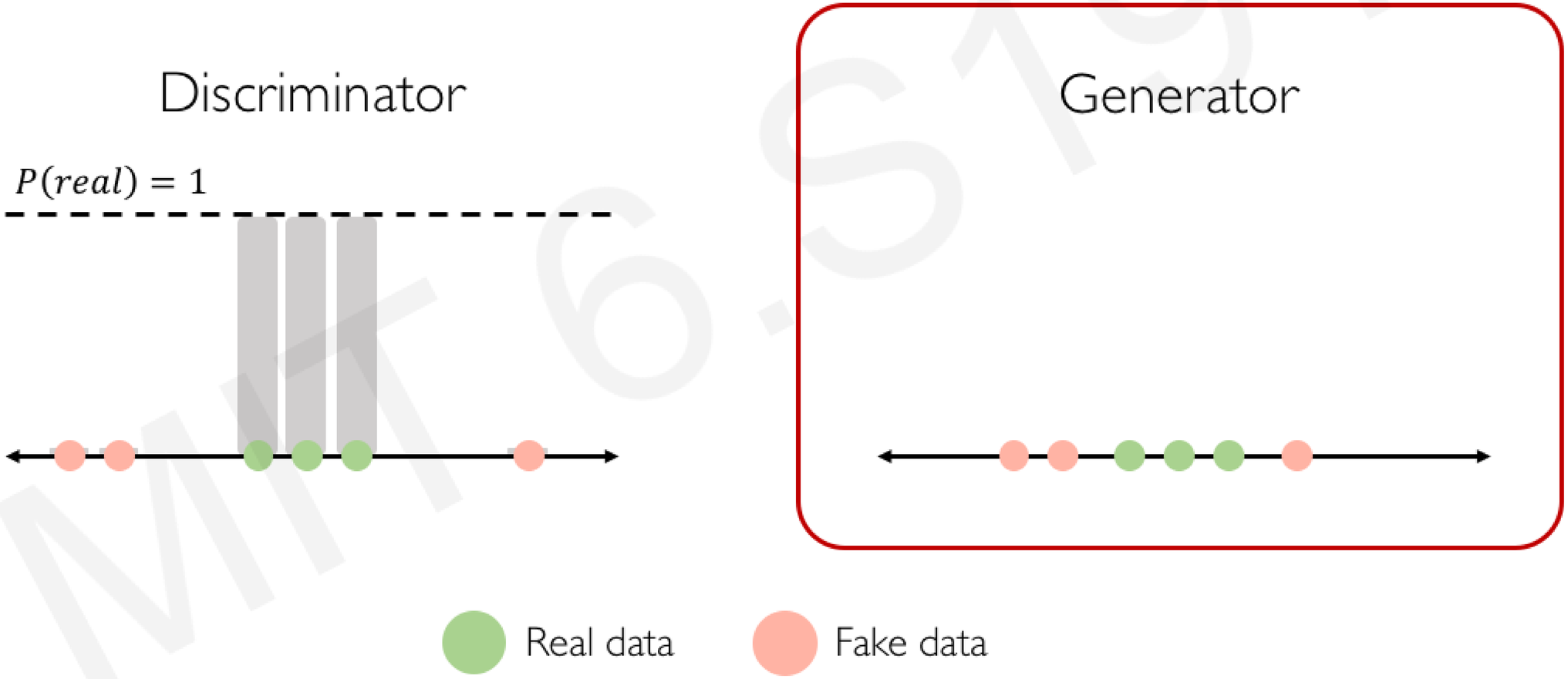
Discriminator tries to predict what's real and what's fake.



MIT 6.S191

Intuition behind GANs

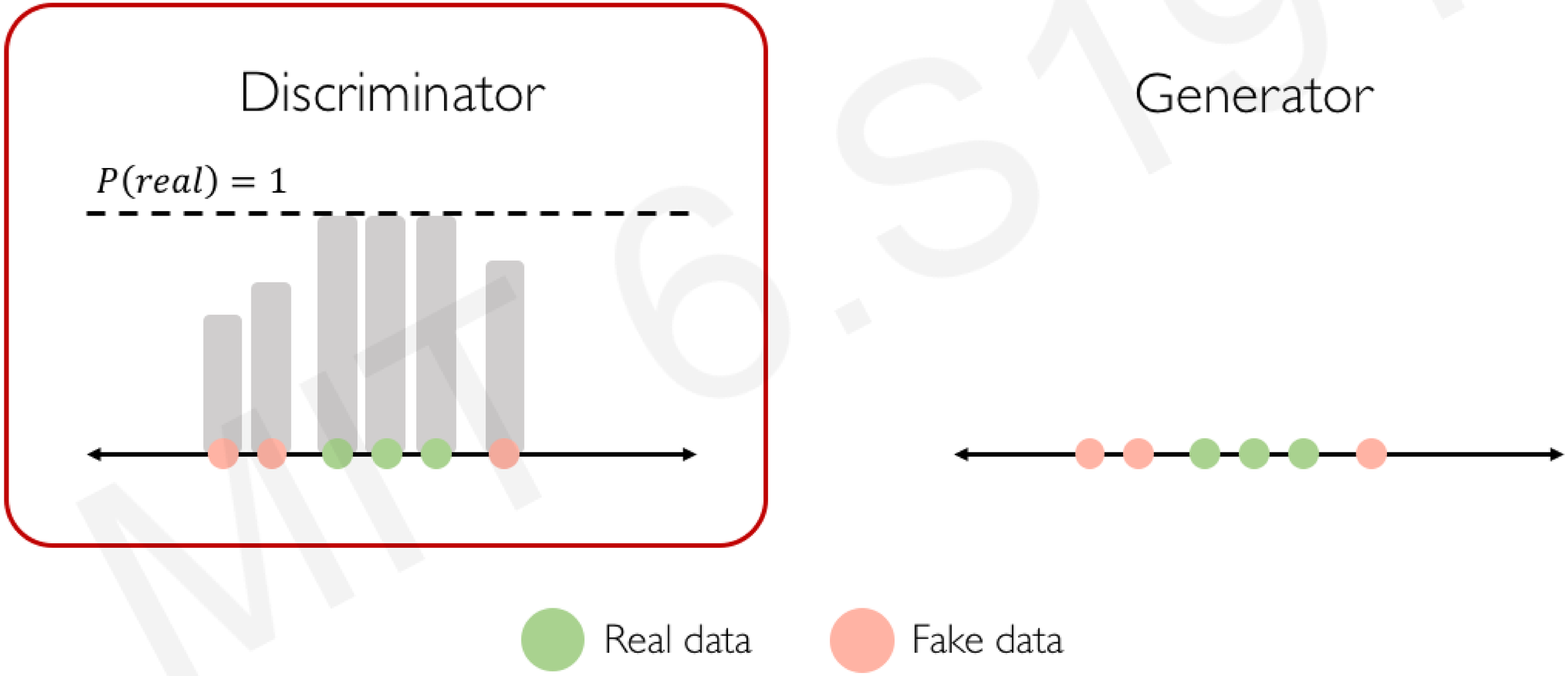
Generator tries to improve its imitation of the data.



MIT 6.S191

Intuition behind GANs

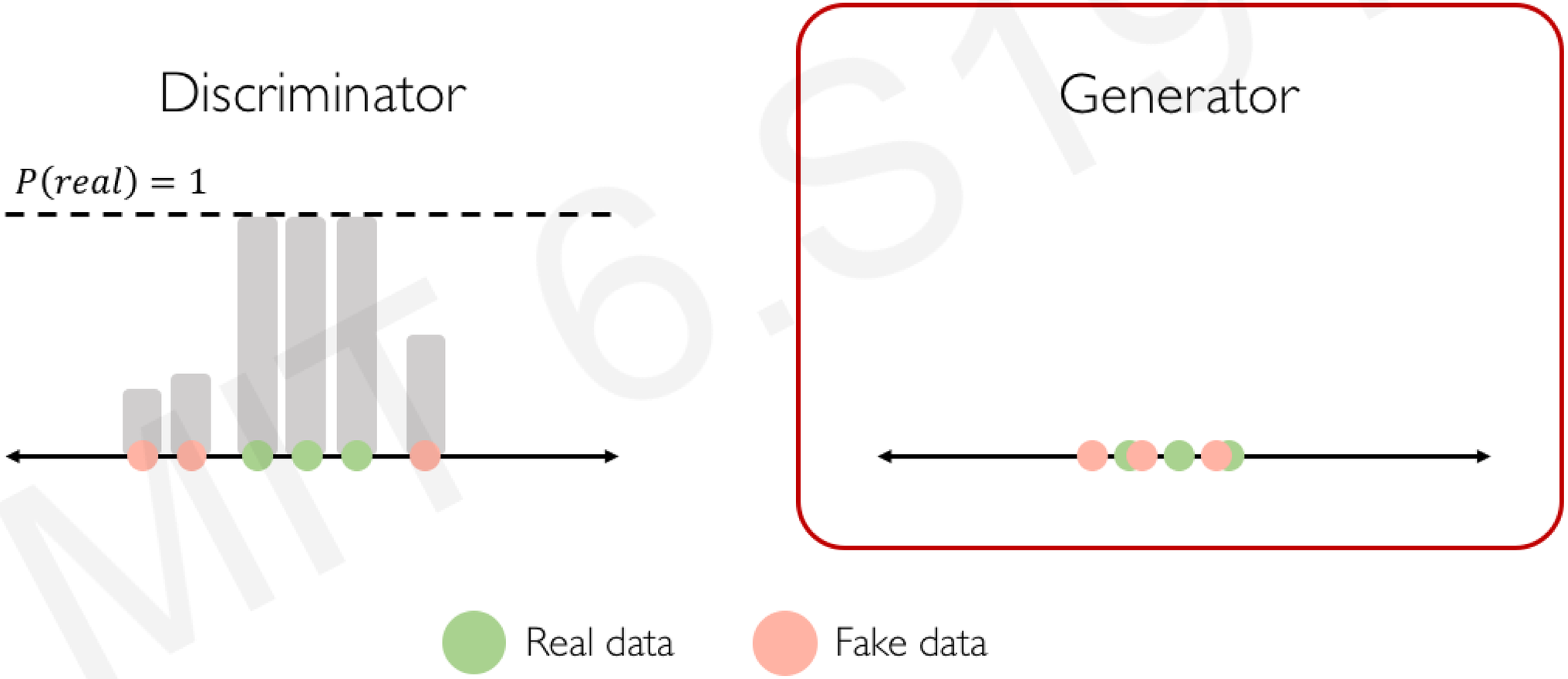
Discriminator tries to predict what's real and what's fake.



MIT 6.S191

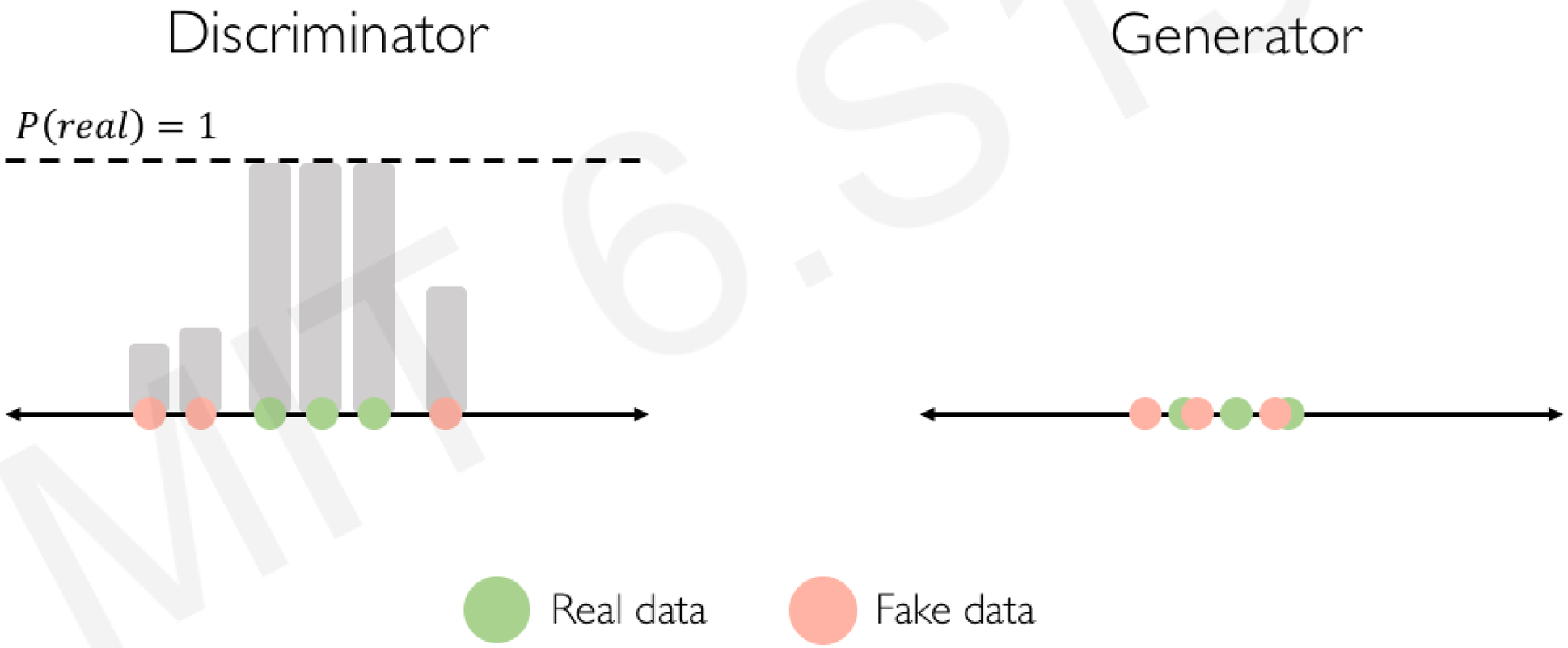
Intuition behind GANs

Generator tries to improve its imitation of the data.



Intuition behind GANs

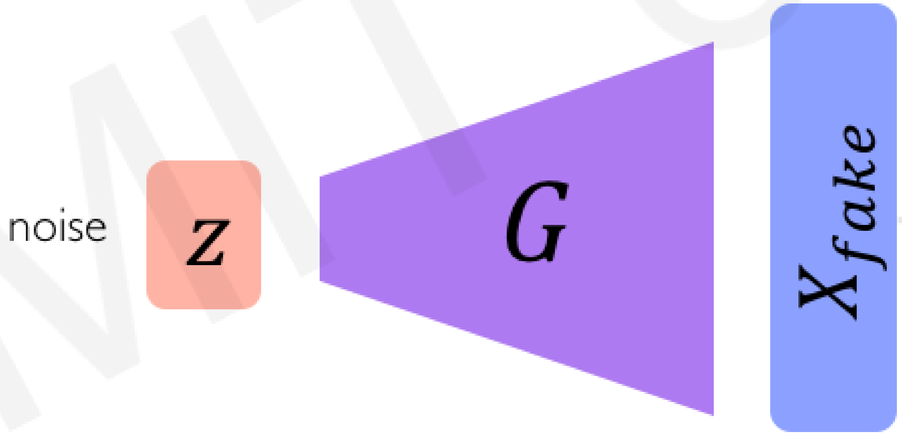
Discriminator tries to identify real data from fakes created by the generator.
Generator tries to create imitations of data to trick the discriminator.



MIT 6.S191

Generating new data with GANs

After training, use generator network to create **new data** that's never been seen before.



Training GANs

Discriminator tries to identify real data from fakes created by the generator.
Generator tries to create imitations of data to trick the discriminator.

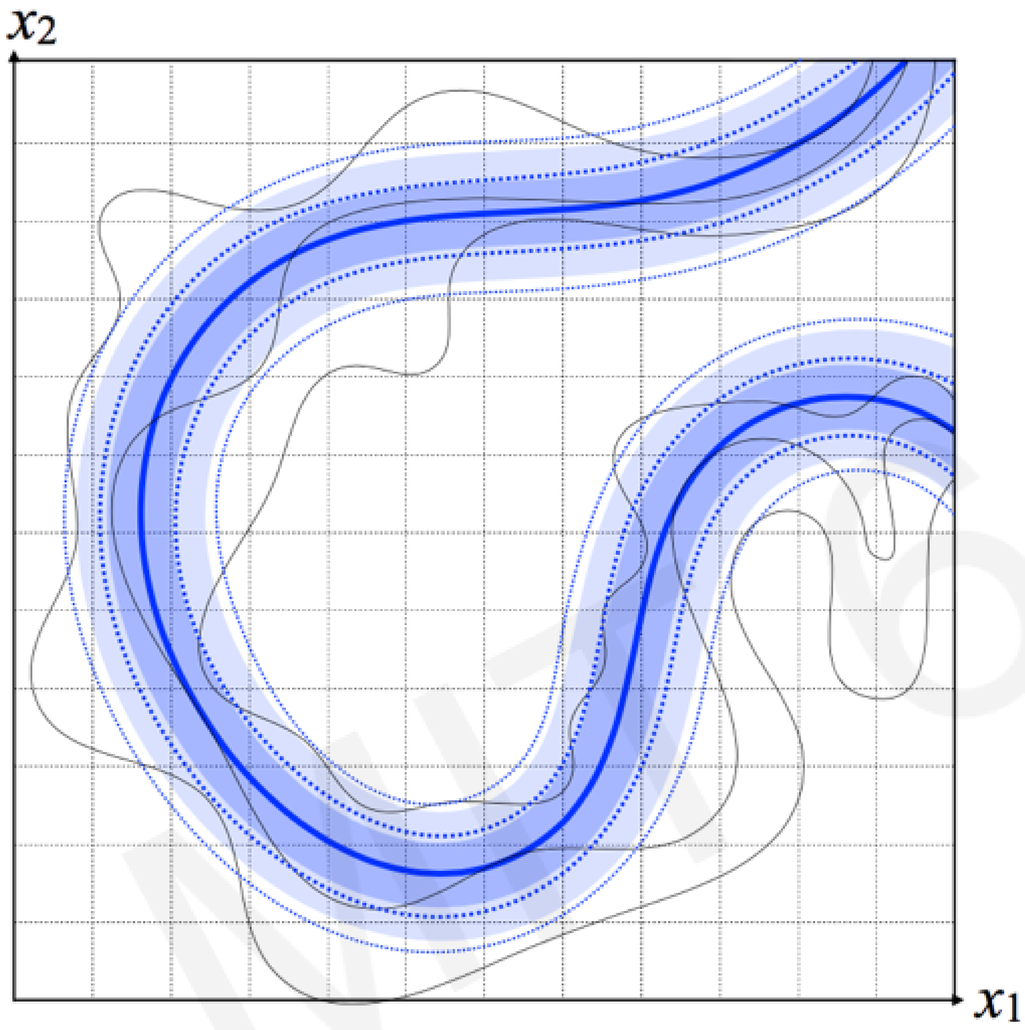
Train GAN jointly via **minimax** game:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

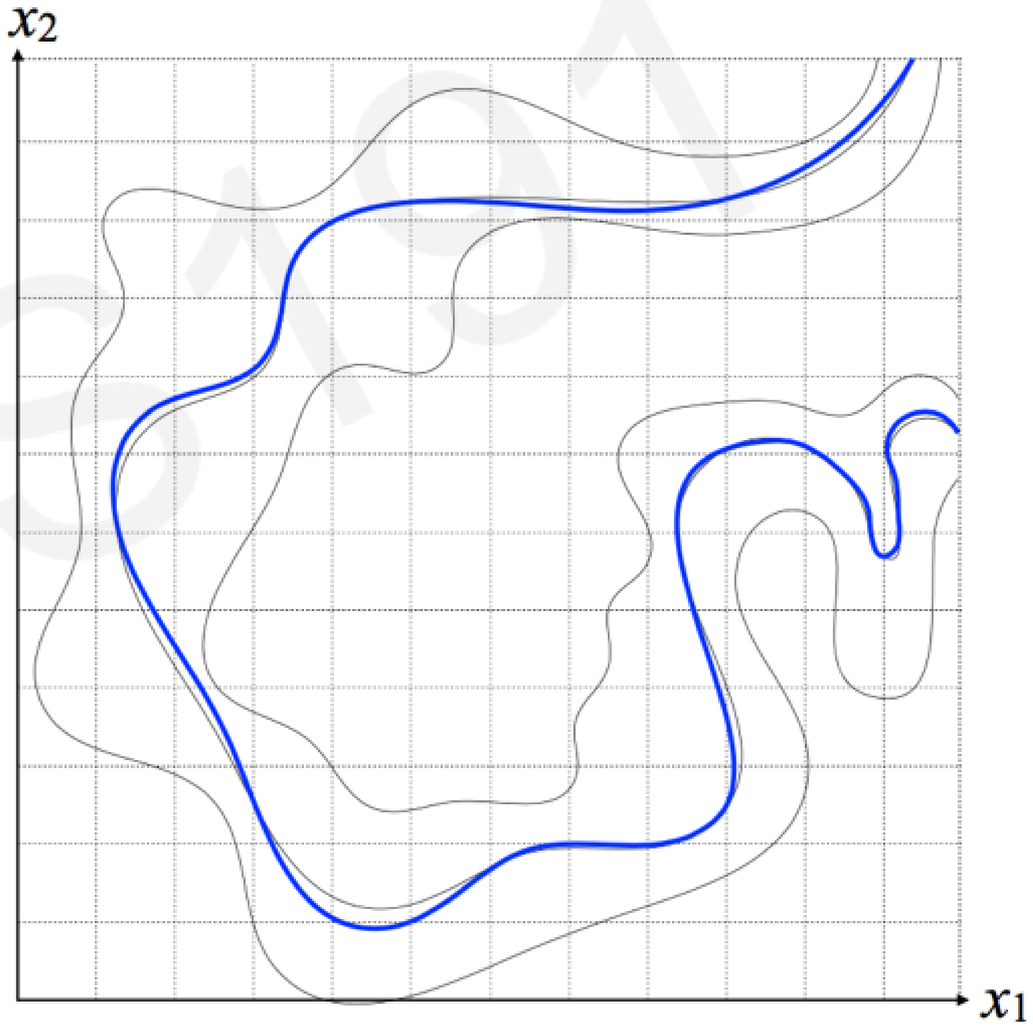
Discriminator wants to maximize objective s.t. $D(x)$ close to 1, $D(G(z))$ close to 0.
Generator wants to minimize objective s.t. $D(G(z))$ close to 1.

MIT 6.S191

Why GANs?

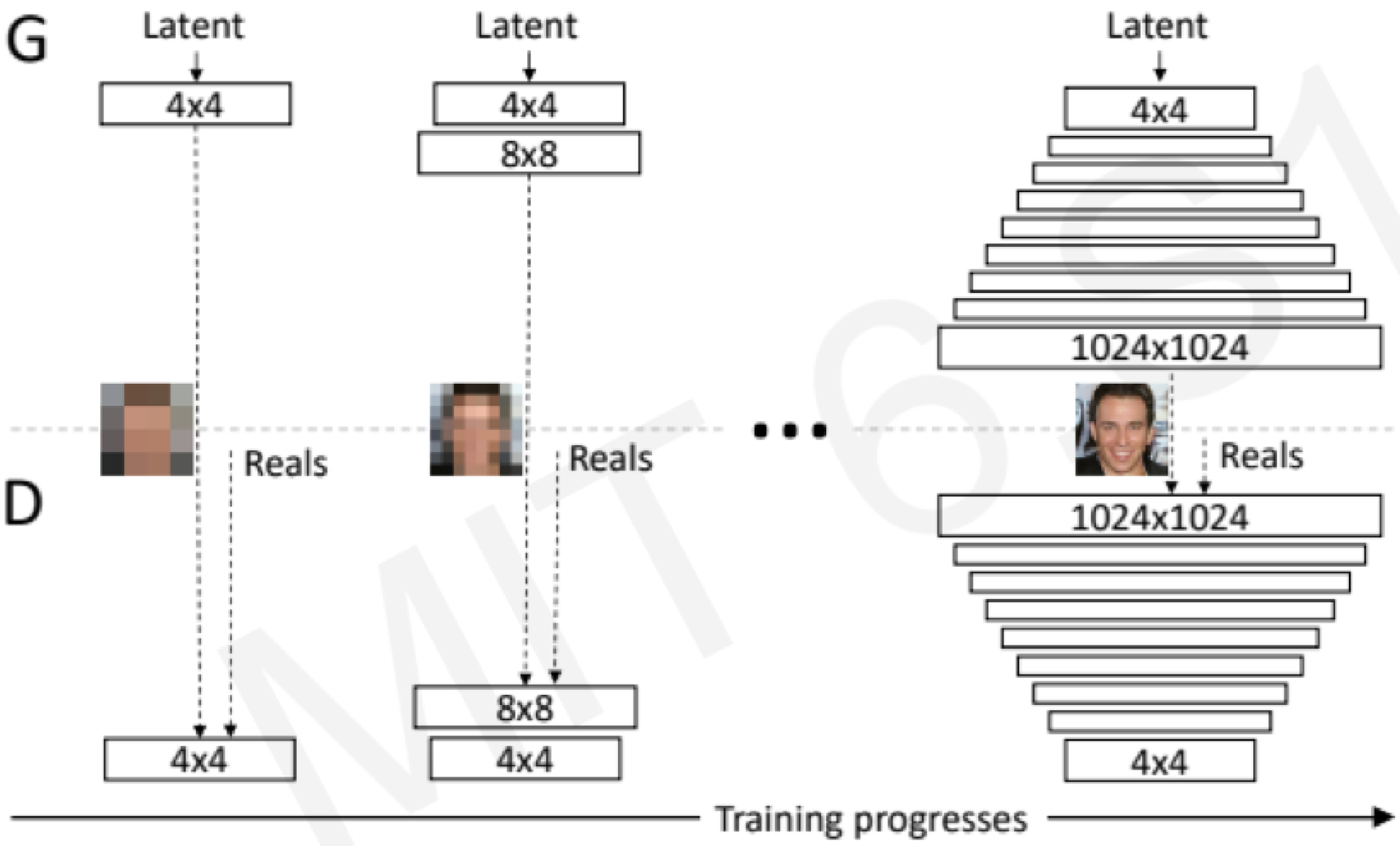


more traditional max-likelihood approach



GAN

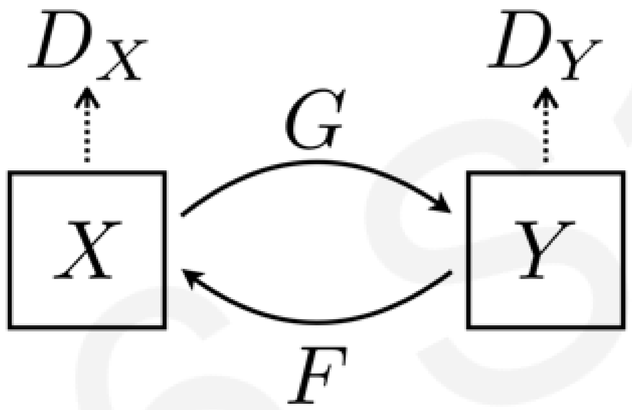
Progressive growing of GANs (NVIDIA)



MIT 6.S191

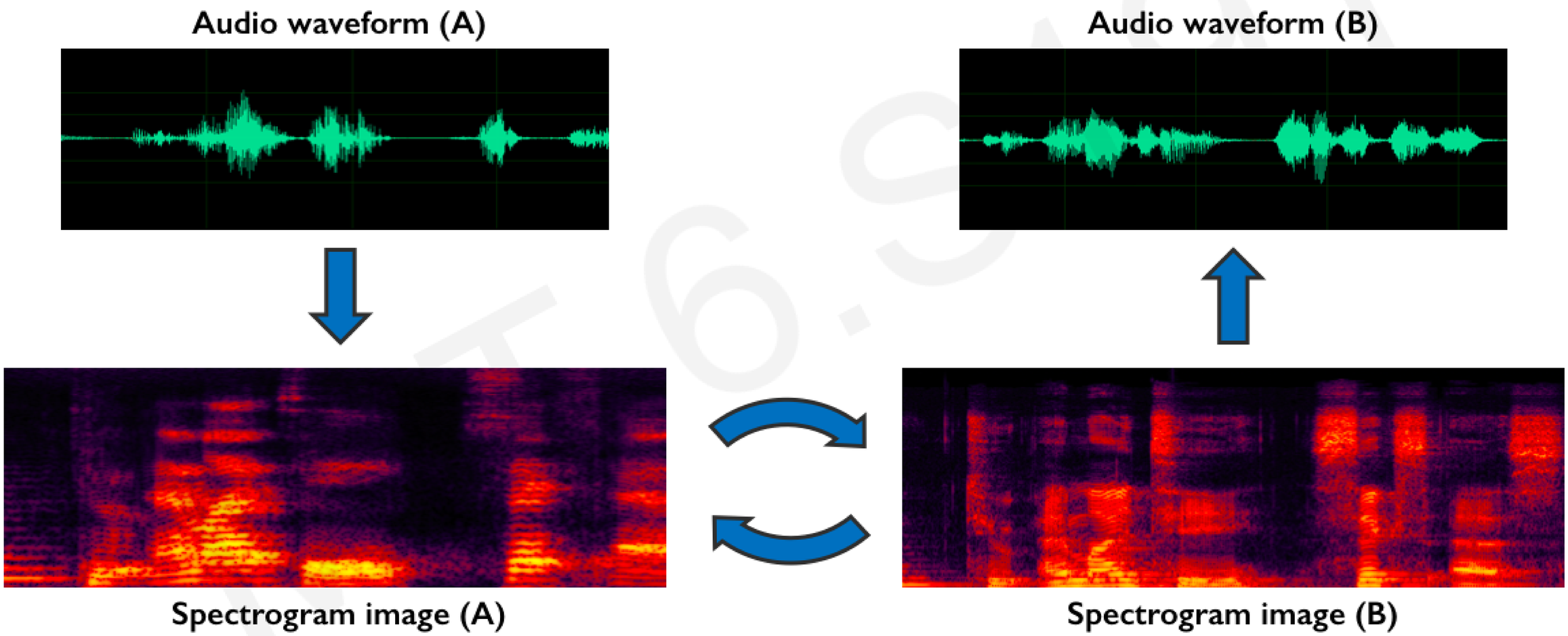
CycleGAN: domain transformation

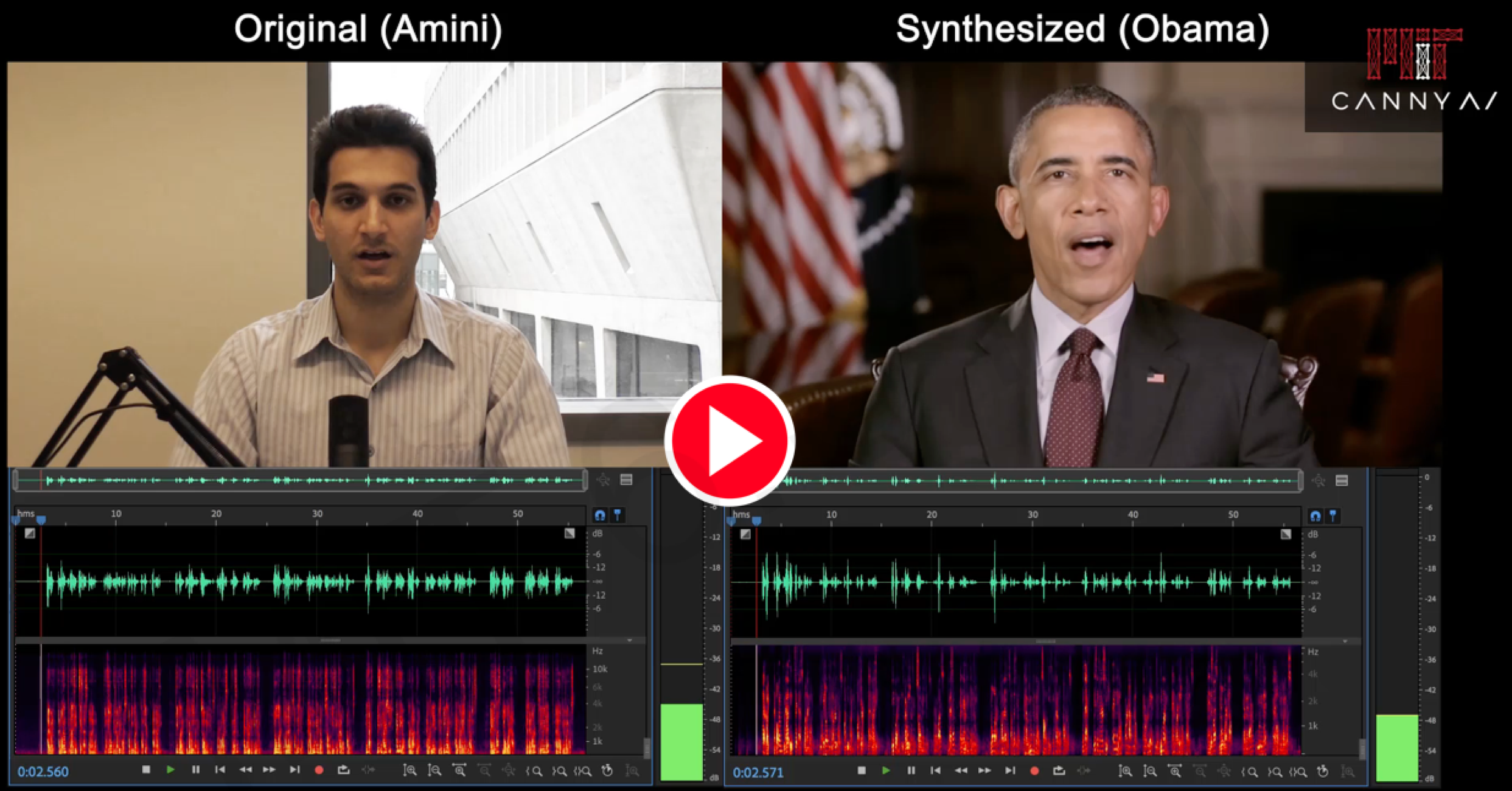
CycleGAN learns transformations across domains with unpaired data.



Zhu+ *ICCV* 2017. 1/28/20

CycleGAN: Transforming speech

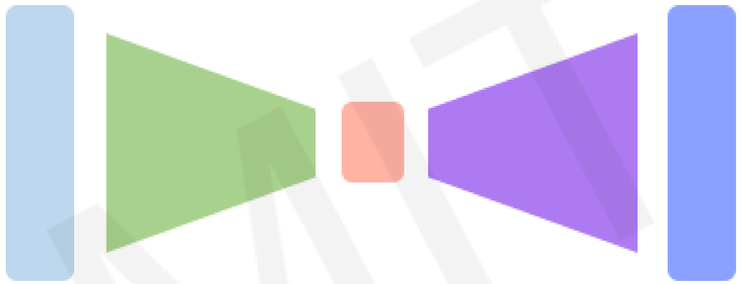



Original (Amini)
Synthesized (Obama)
CANNYAI
0:02.560
0:02.571

Deep Generative Modeling: Summary

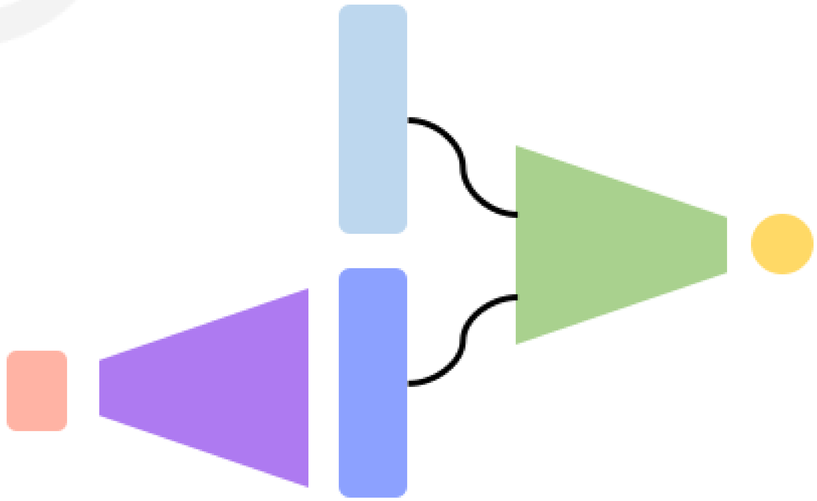
Autoencoders and Variational Autoencoders (VAEs)

Learn **lower-dimensional** latent space and **sample** to generate input reconstructions



Generative Adversarial Networks (GANs)

Competing **generator** and **discriminator** networks



MIT 6.S191

Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$$x \rightarrow y$$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

Apple example:

Eat this thing because it will keep you alive.

MIT 6.S191

Reinforcement Learning (RL): Key Concepts

AGENT

Agent: takes actions.

MIT 6.S191

Reinforcement Learning (RL): Key Concepts

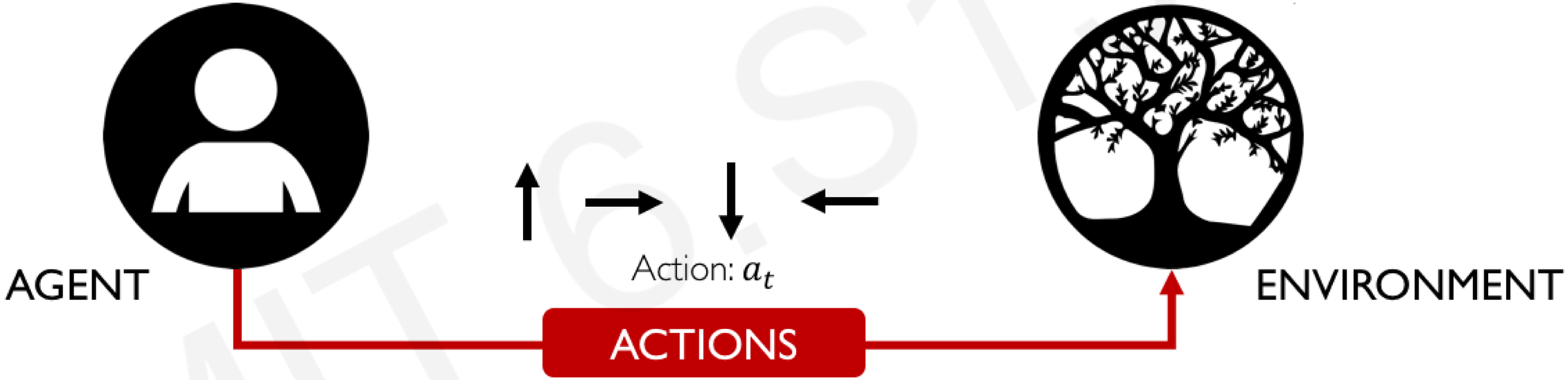
AGENT

ENVIRONMENT

Environment: the world in which the agent exists and operates.

MIT 6.S191

Reinforcement Learning (RL): Key Concepts



Action: a move the agent can make in the environment.

Action space A: the set of possible actions an agent can make in the environment

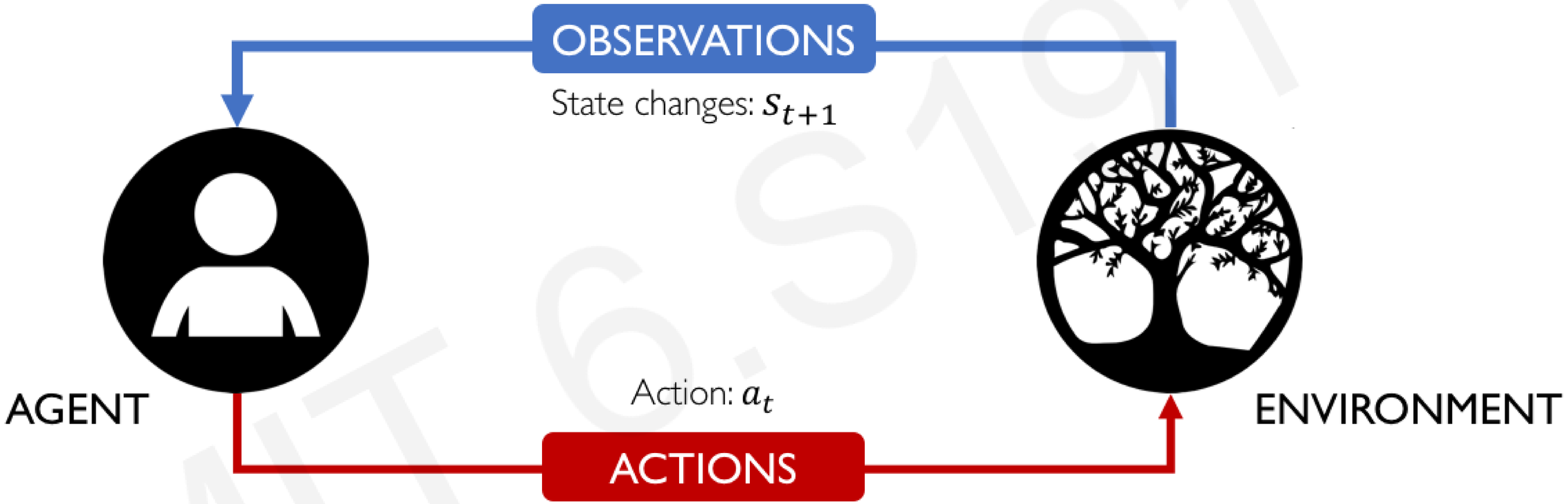
MIT 6.S191

Reinforcement Learning (RL): Key Concepts

Observations: of the environment after taking actions.

MIT 6.S191

Reinforcement Learning (RL): Key Concepts



State: a situation which the agent perceives.

MIT 6.S191

Reinforcement Learning (RL): Key Concepts

Reward: feedback that measures the success or failure of the agent's action.

MIT 6.S191

Reinforcement Learning (RL): Key Concepts

Total Reward
(Return)

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} \ldots + r_{t+n} + \cdots$$

MIT 6.S191

Reinforcement Learning (RL): Key Concepts

Discounted Total Reward **(Return)**

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} \dots + \gamma^{t+n} r_{t+n} + \cdots$$

γ: discount factor; $0 < \gamma < 1$

MIT 6.S191

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

MIT 6.S191

Defining the Q-function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total reward, R_t, is the discounted sum of all rewards obtained from time t

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

The Q-function captures the **expected total future reward** an agent in state, s, can receive by executing a certain action, a

MIT 6.S191

How to take actions given a Q-function?

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

(state, action)

Ultimately, the agent needs a **policy $\pi(s)$,** to infer the **best action to take** at its state, s

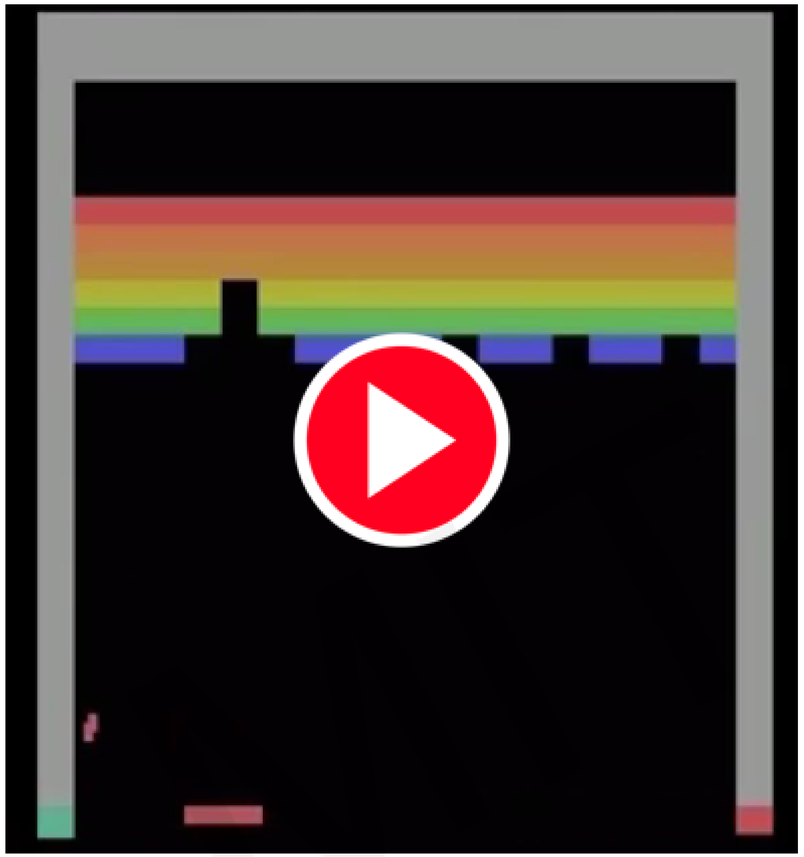
Strategy: the policy should choose an action that maximizes future reward

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

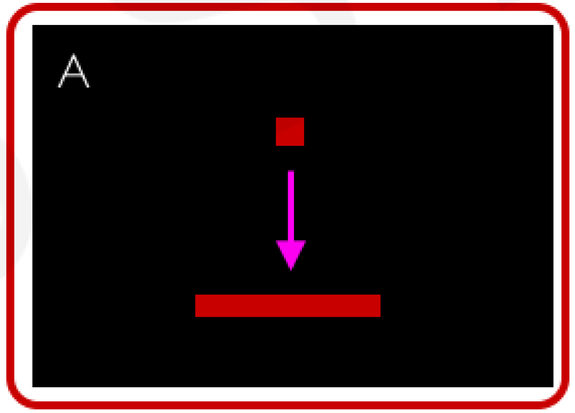
MIT 6.S191

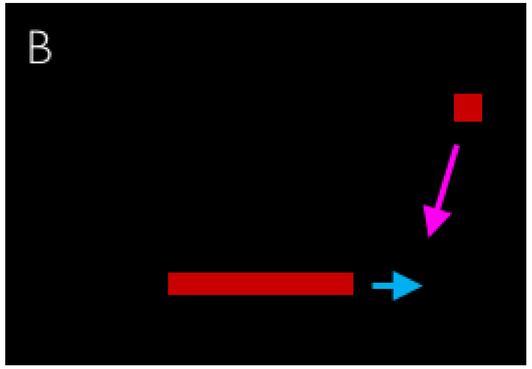
Digging deeper into the Q-function

Example: Atari Breakout - Middle



It can be very difficult for humans to accurately estimate Q-values

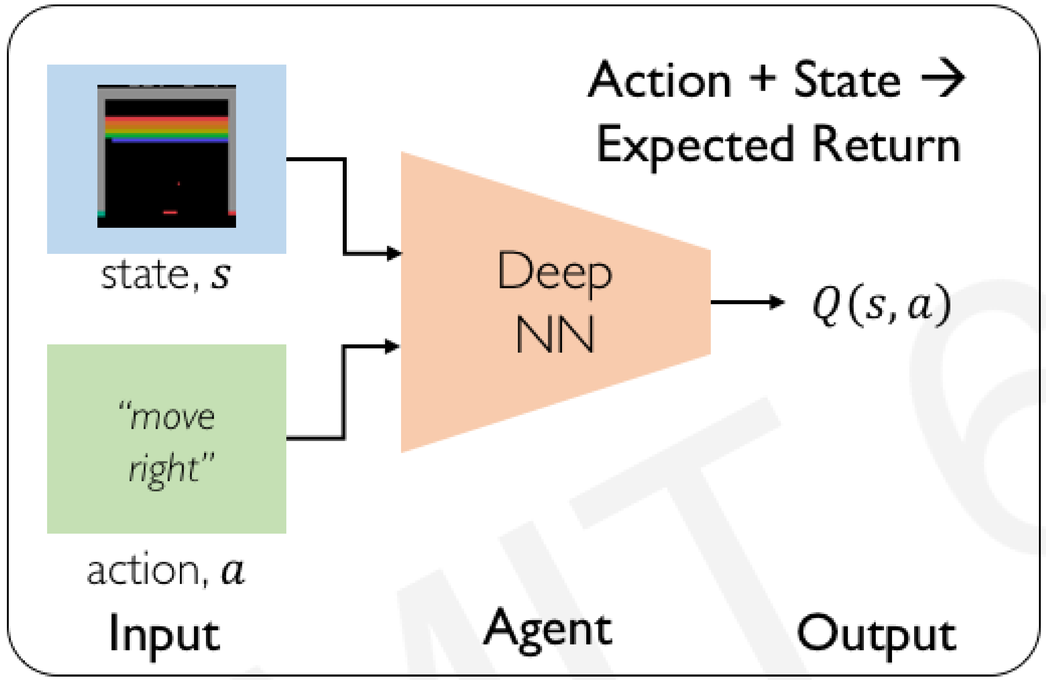


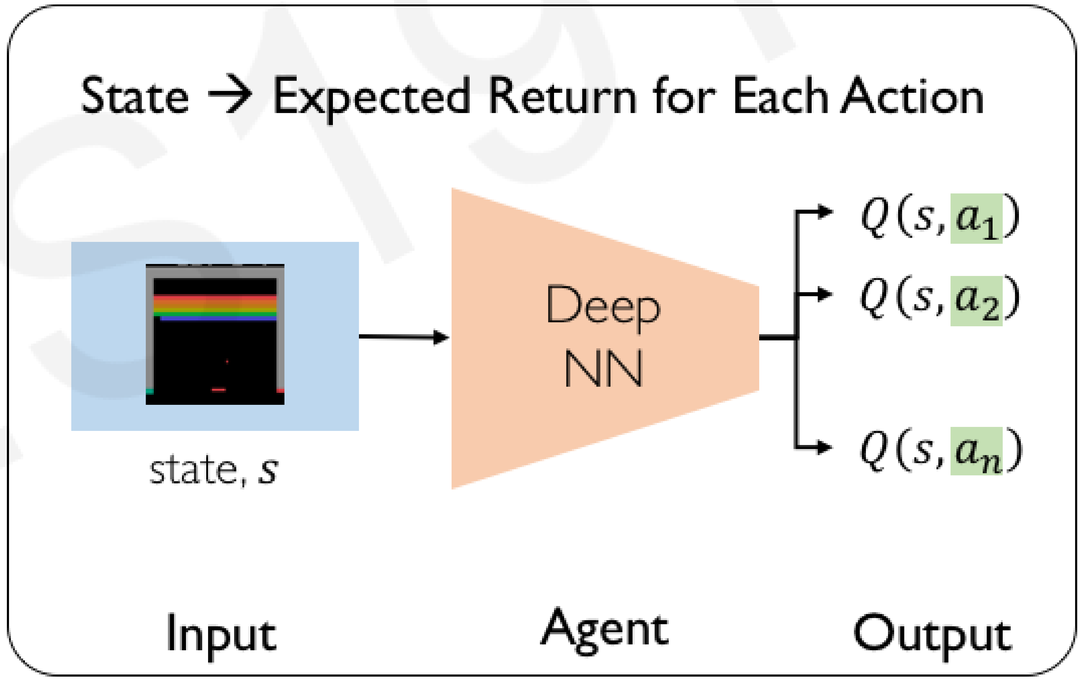


Which (**s**, **a**) pair has a higher Q-value?

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?

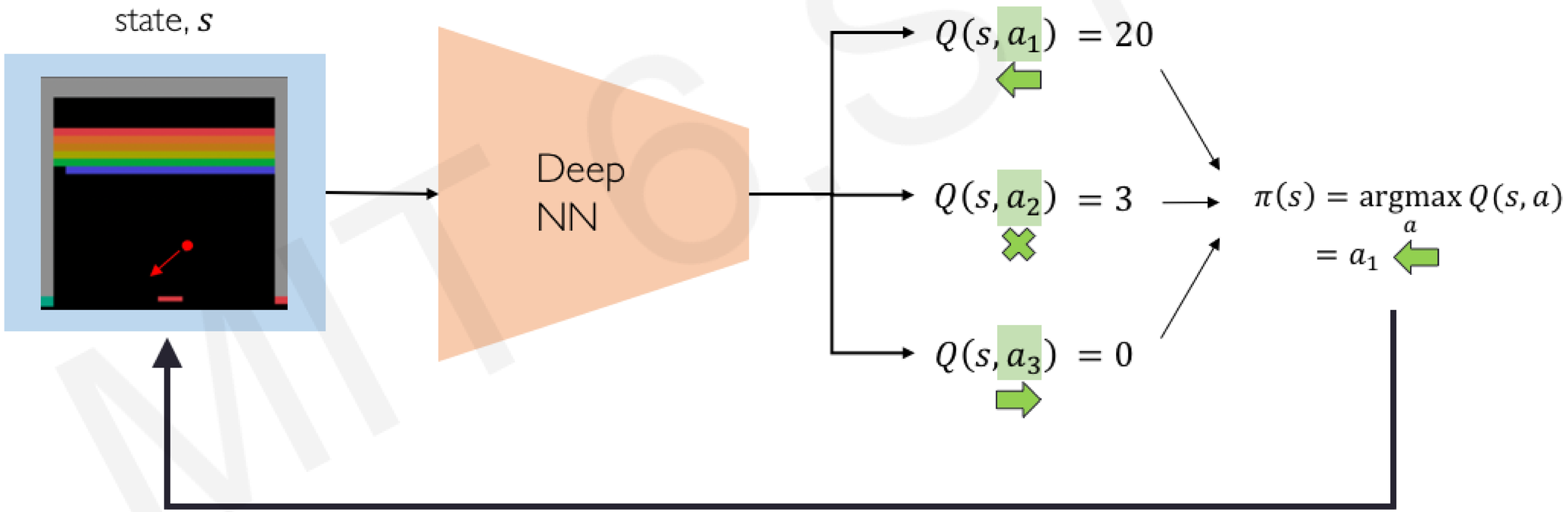




$$\mathcal{L} = \mathbb{E}\left[\left\|\overbrace{\left(r + \gamma \max_{a'} Q(s',a')\right)}^{\text{target}} - \overbrace{Q(s,a)}^{\text{predicted}}\right\|^2\right] \qquad \textbf{Q-Loss}$$

Deep Q Network Summary

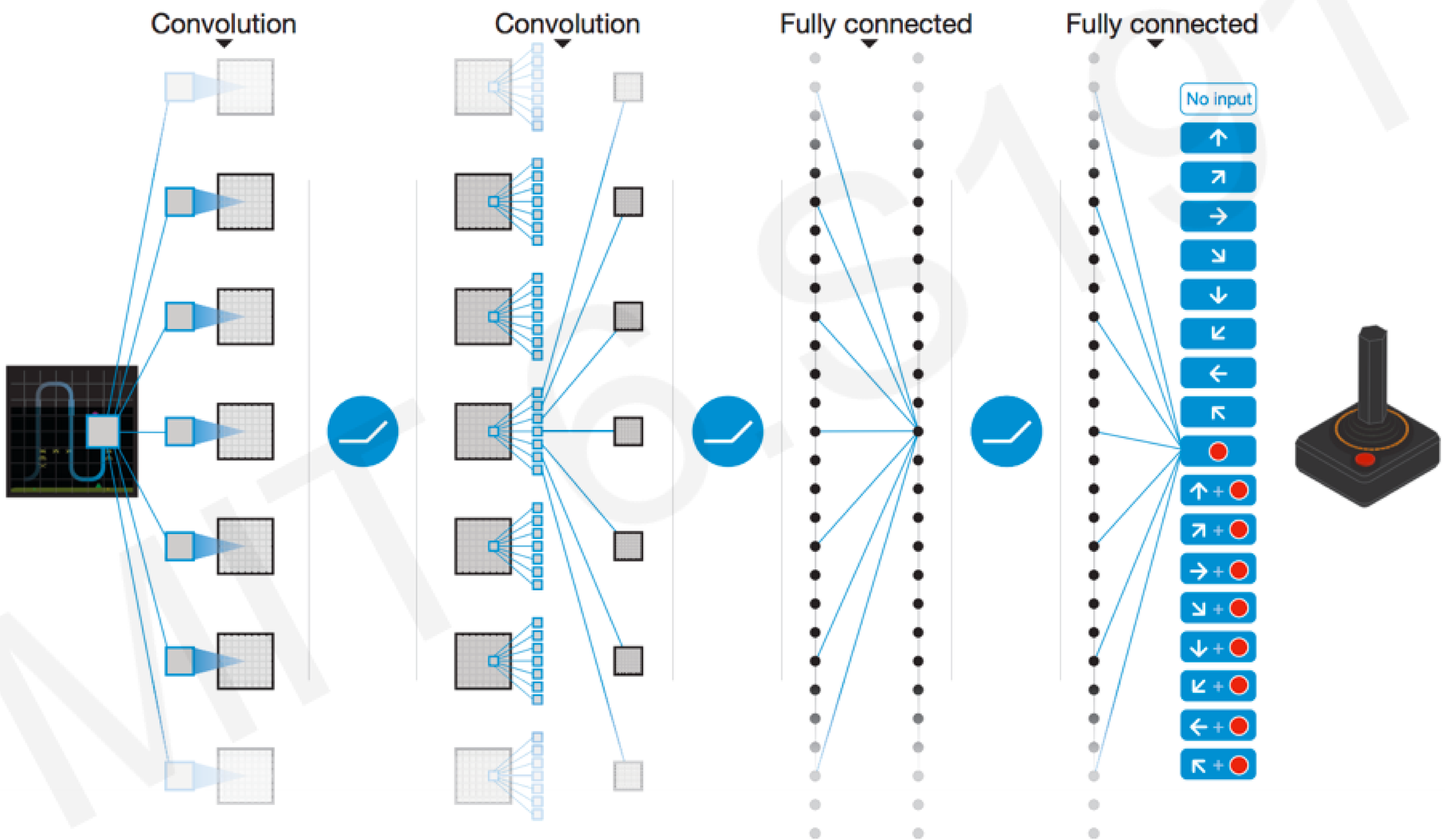
Use NN to learn Q-function and then use to infer the optimal policy, $\pi(s)$



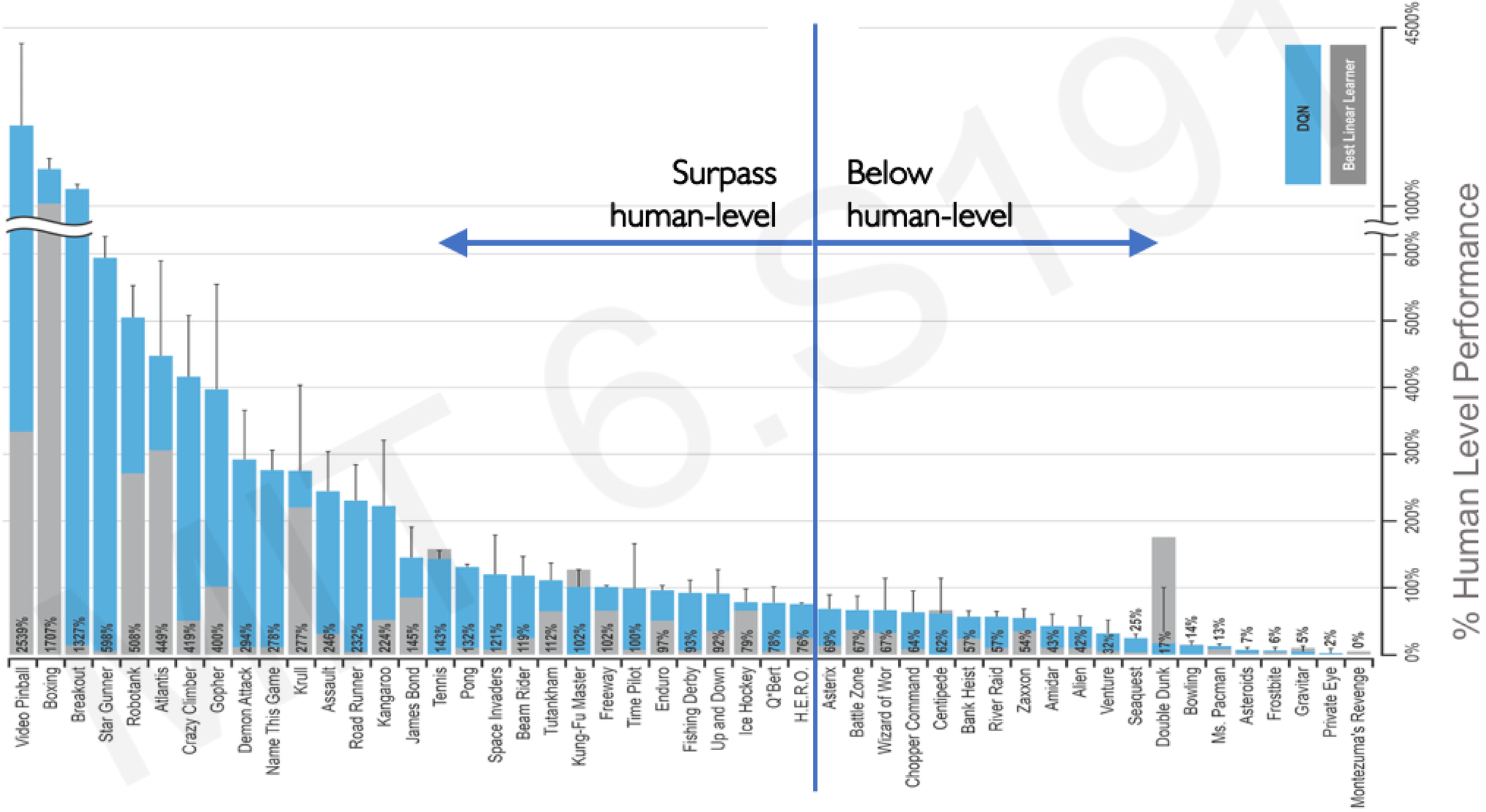
Send action back to environment and receive next state

MIT 6.S191

DQN Atari Results



DQN Atari Results



MIT 6.S191

Downsides of Q-learning

Complexity:

- Can model scenarios where the action space is discrete and small
- Cannot handle continuous action spaces

Flexibility:

- Policy is deterministically computed from the Q function by maximizing the reward → cannot learn stochastic policies

To address these, consider a new class of RL training algorithms:
Policy gradient methods

MIT 6.S191

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy Learning

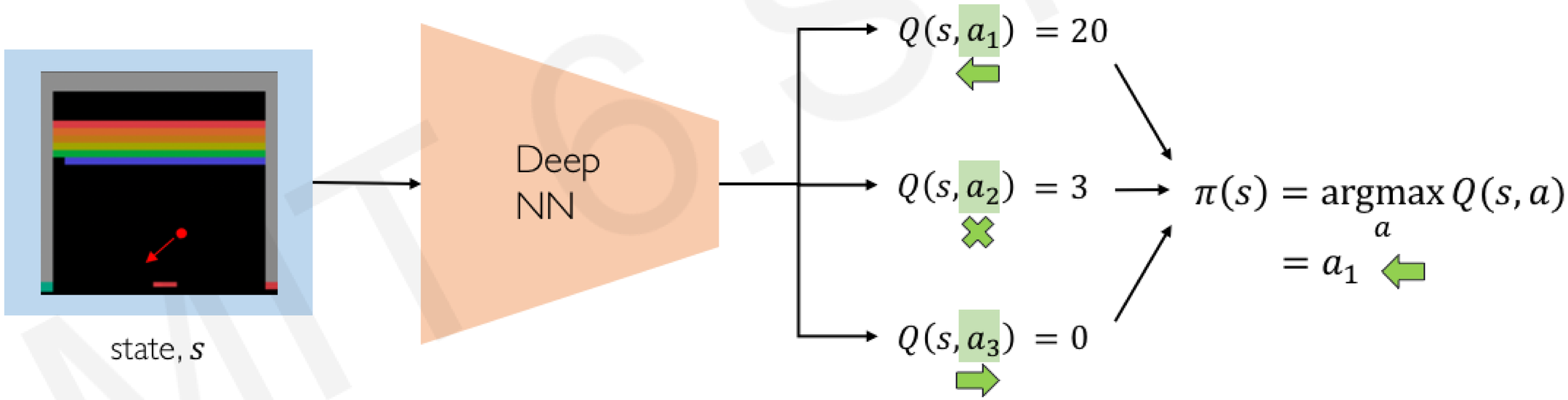
Find $\pi(s)$

Sample $a \sim \pi(s)$

MIT 6.S191

Deep Q Networks (DQN)

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

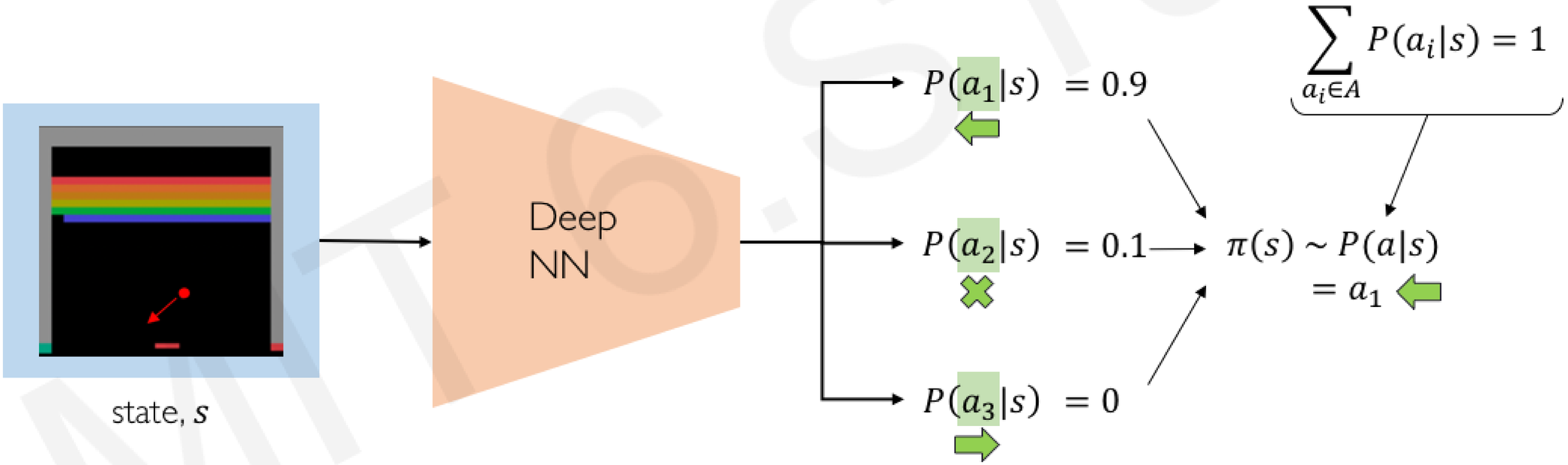


MIT 6.S191

Policy Gradient (PG): Key Idea

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

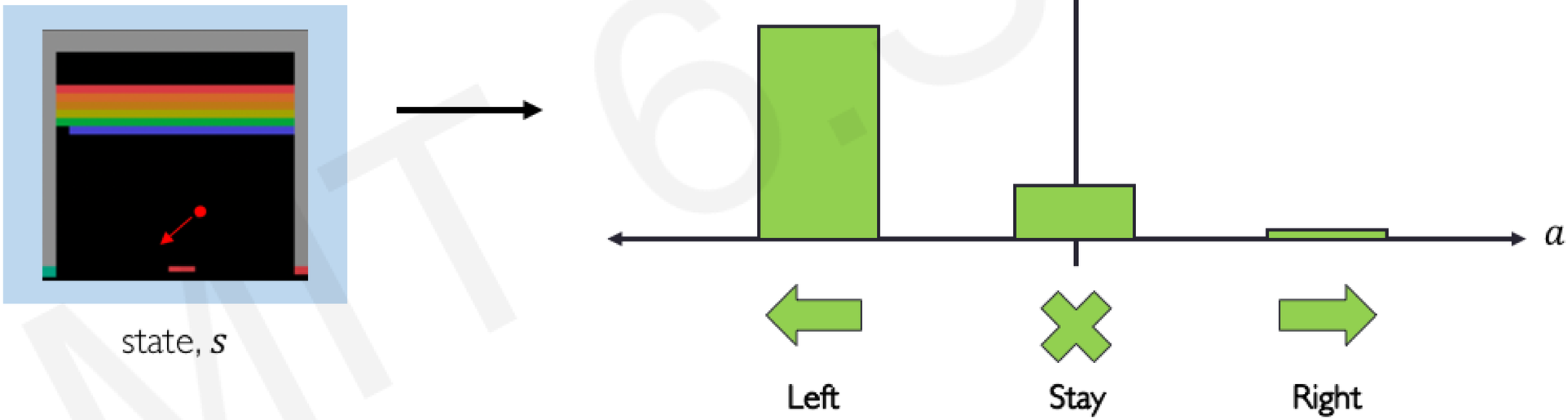
Policy Gradient: Directly optimize the policy $\pi(s)$



What are some advantages of this formulation?

Discrete vs Continuous Action Spaces

Discrete action space: which direction should I move?

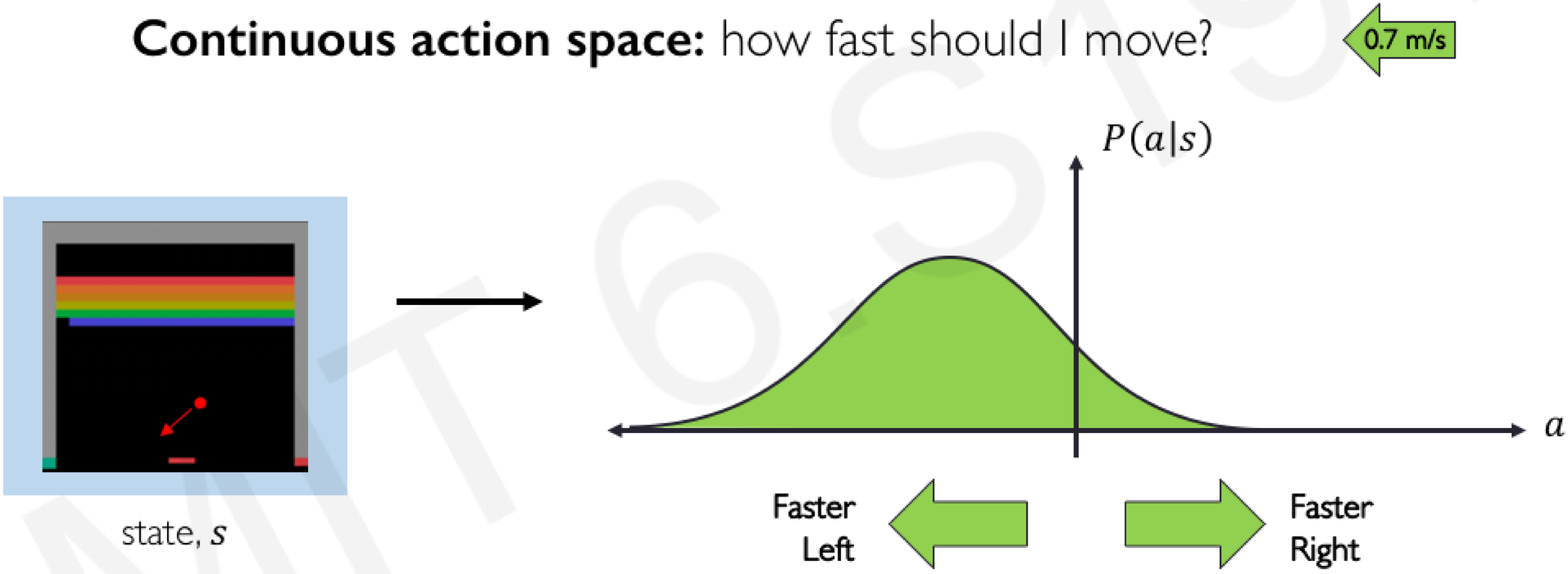


MIT 6.S191

Discrete vs Continuous Action Spaces

Discrete action space: which direction should I move?

Continuous action space: how fast should I move?

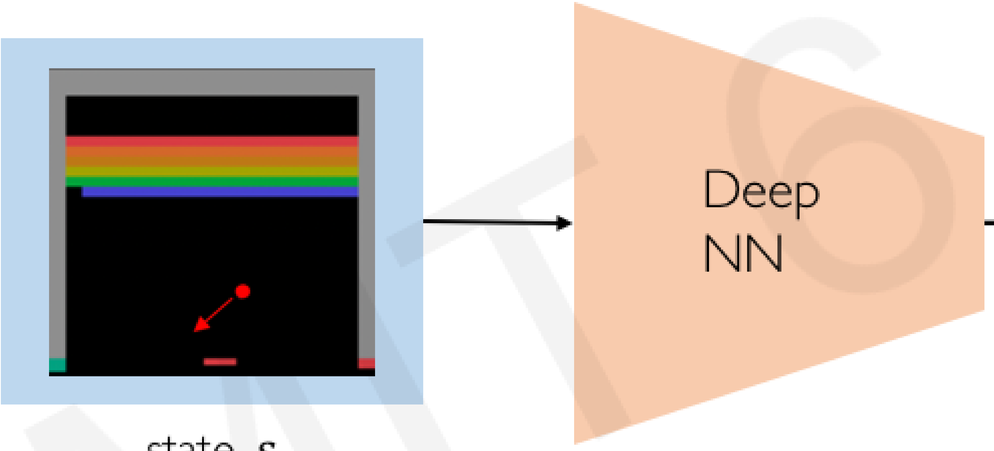


MIT 6.S191

Policy Gradient (PG): Key Idea

Policy Gradient: Enables modeling of continuous action space

$$\underbrace{\int_{a=-\infty}^{\infty} P(a|s) = 1}$$



state, s

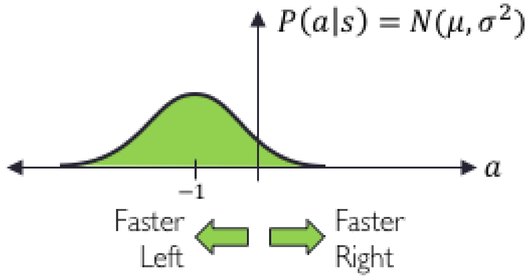
Mean, $\mu = -1$

Variance, $\sigma^2 = 0.5$

$$P(a|s) = N(\mu, \sigma^2)$$

$$\pi(s) \sim P(a|s)$$

$$= -0.8 \text{ [m/s]}$$



MIT 6.S191

Training Policy Gradients: Case Study

Reinforcement Learning Loop:

Case Study – Self-Driving Cars

Agent: vehicle

State: camera, lidar, etc

Action: steering wheel angle

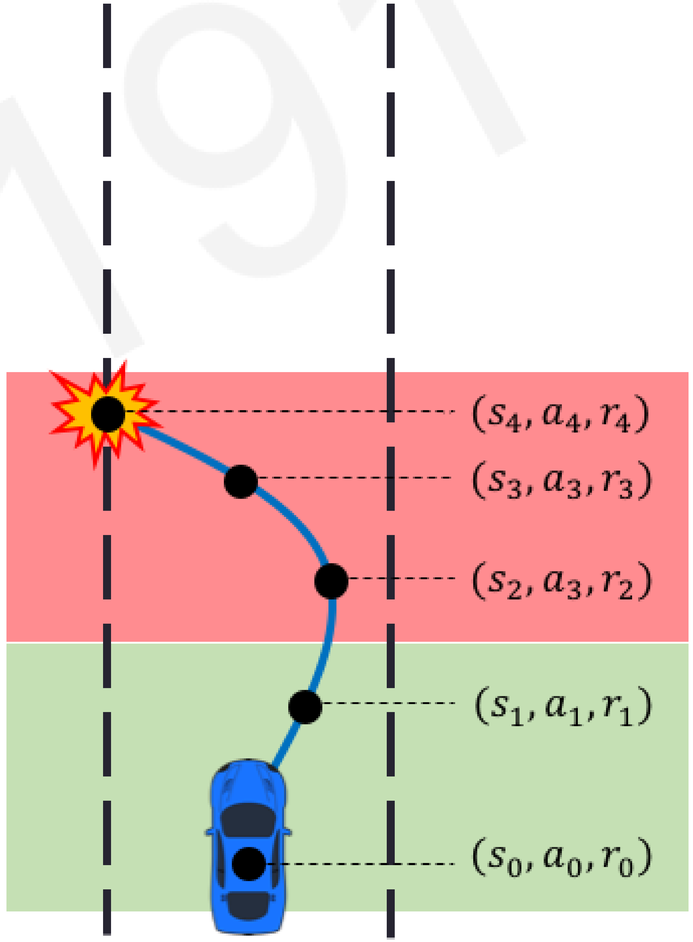
Reward: distance traveled

MIT 6S191

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

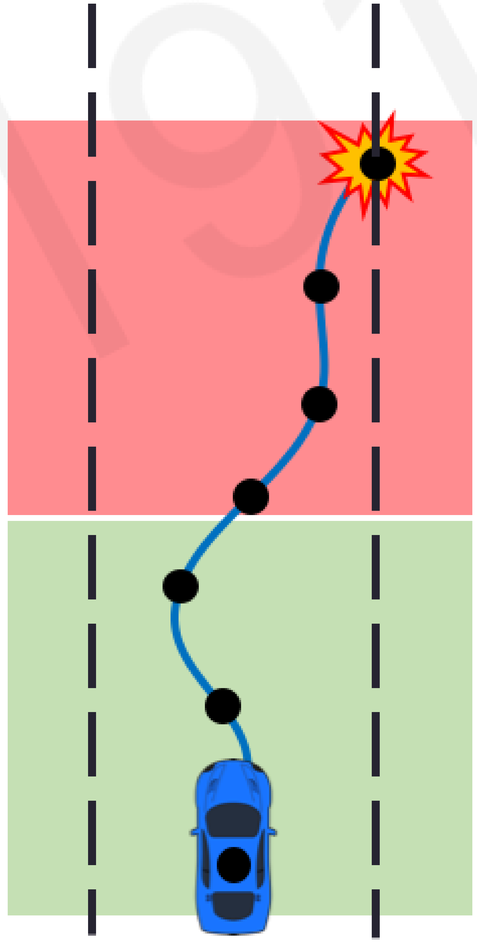


MIT 6.S191

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

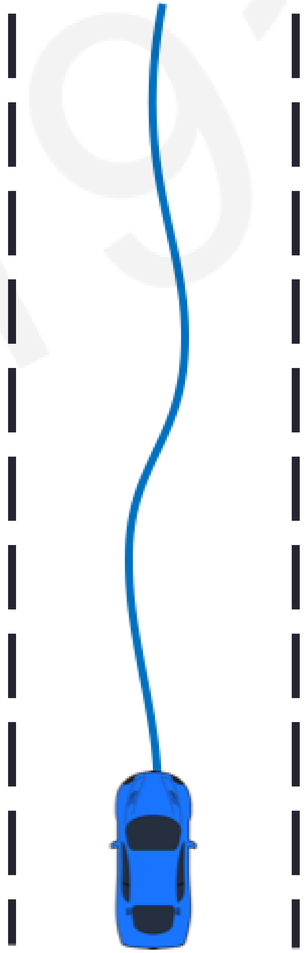


MIT 6.S191

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



MIT 6.S191

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

log-likelihood of action

$$\mathbf{loss} = -\log \mathrm{P}(a_t|s_t)\, R_t$$

reward

Gradient descent update:

$$w' = w - \nabla\, \mathbf{loss}$$

$$w' = w + \boxed{\nabla \log \mathrm{P}(a_t|s_t)\, R_t}$$

Policy gradient!

MIT 6.S191

Reinforcement Learning in Real Life

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

MIT 6.S191

Data-driven Simulation for Autonomous Vehicles

VISTA: Photorealistic and high-fidelity simulator for training and testing self-driving cars

Amini+ *IEEE RA-L* 2020. 1/29/20

Deploying End-to-End RL for Autonomous Vehicles

Policy Gradient RL agent trained entirely within VISTA simulator

End-to-end agent directly deployed into the real-world

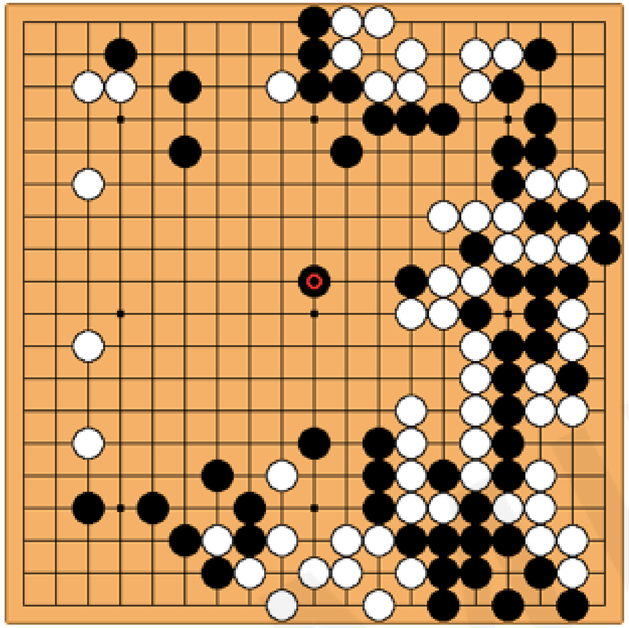
First full-scale autonomous vehicle trained using RL entirely in simulation and deployed in real life!

Deep Reinforcement Learning Applications

MIT 6.S191

The Game of Go

Aim: Get more board territory than your opponent.

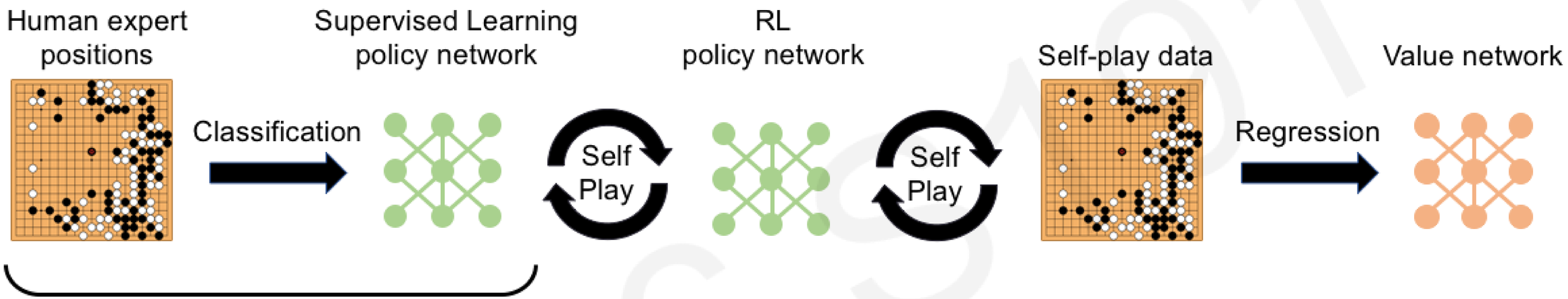


Board Size $n \times n$	Positions 3^{n^2}	% Legal	Legal Positions
1×1	3	33.33%	1
2×2	81	70.37%	57
3×3	19,683	64.40%	12,675
4×4	43,046,721	56.49%	24,318,165
5×5	847,288,609,443	48.90%	414,295,148,741
9×9	$4.434264882 \times 10^{38}$	23.44%	$1.03919148791 \times 10^{38}$
13×13	$4.300233593 \times 10^{80}$	8.66%	$3.72497923077 \times 10^{79}$
19×19	$1.740896506 \times 10^{172}$	1.20%	$2.08168199382 \times 10^{170}$

Greater number of legal board positions than atoms in the universe.

MIT 6.S191

AlphaGo Beats Top Human Player at Go (2016)



1) Initial training: human data

2) Self-play and reinforcement learning
→ super-human performance

3) "Intuition" about board state

MIT 6.S191

Deep Reinforcement Learning: Summary

Foundations

- Agents acting in environment
- State-action pairs → maximize future rewards
- Discounting

Q-Learning

- Q function: expected total reward given s, a
- Policy determined by selecting action that maximizes Q function

Policy Gradients

- Learn and optimize the policy directly
- Applicable to continuous action spaces

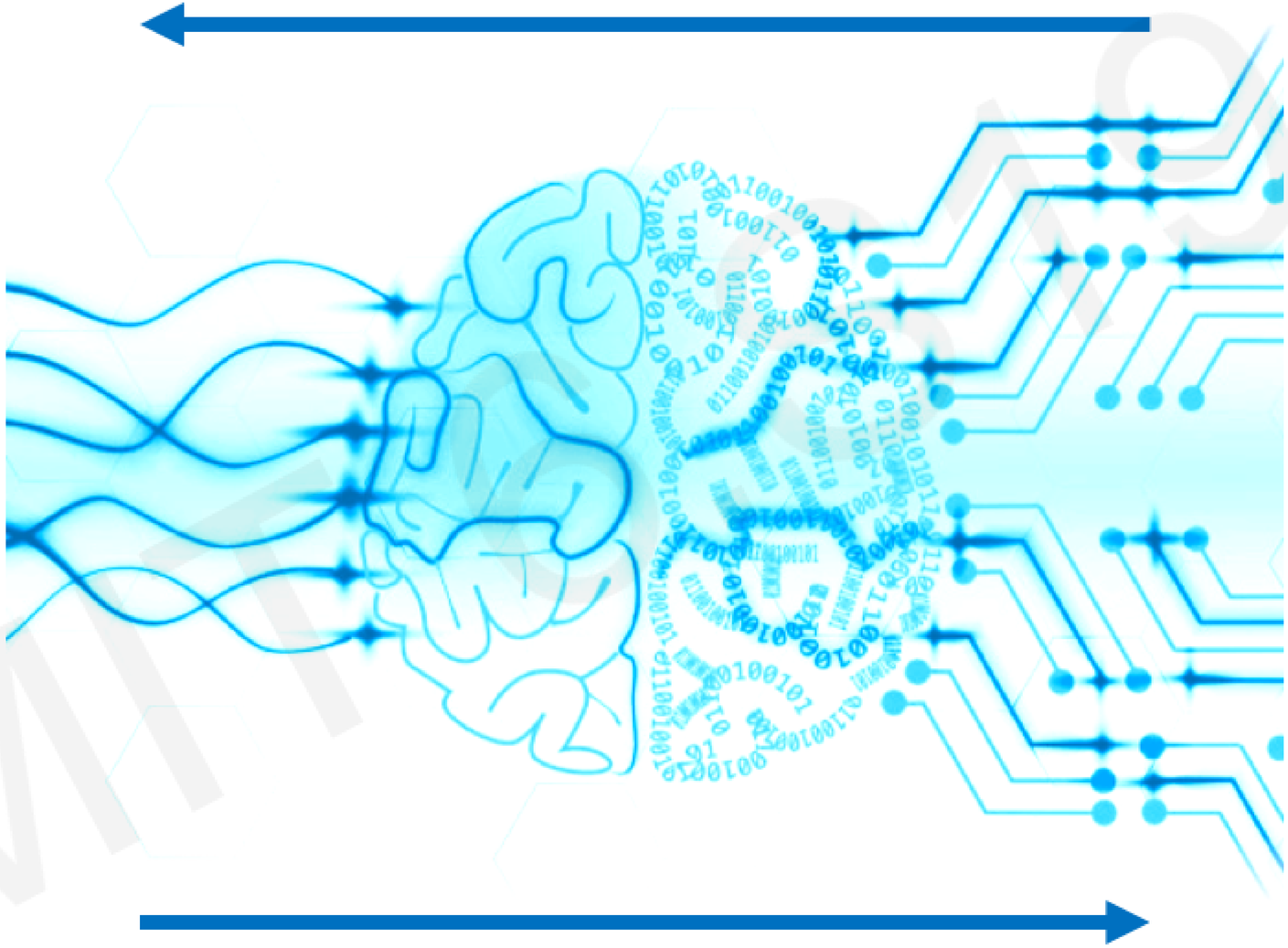
MIT 6.S191

So far in 6.S191…

Data

- Signals
- Images
- Sensors

…



Decision

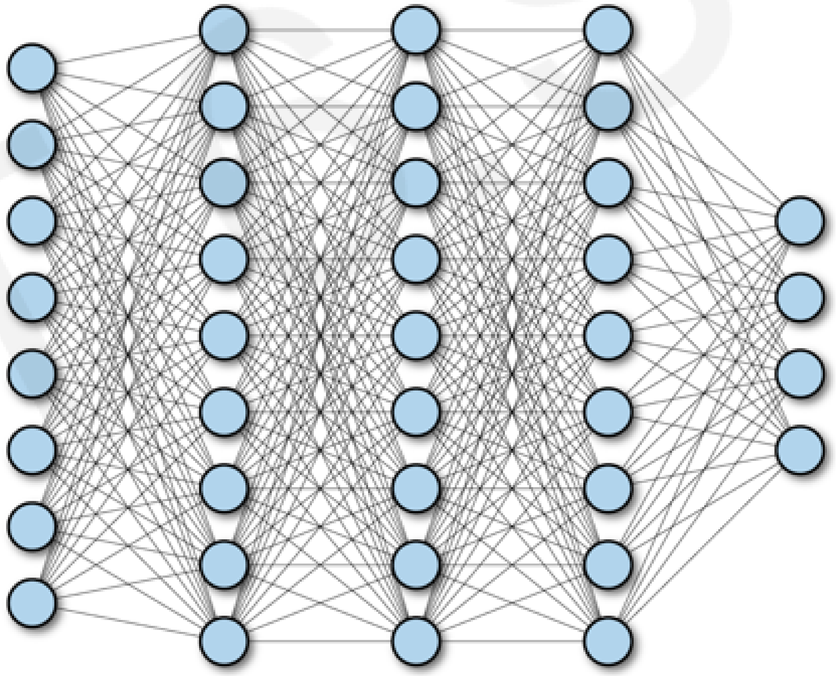
- Prediction
- Detection
- Action

…

Power of Neural Nets

Universal Approximation Theorem

A feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function.



MIT 6.S191

Power of Neural Nets

Universal Approximation Theorem

A feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function.

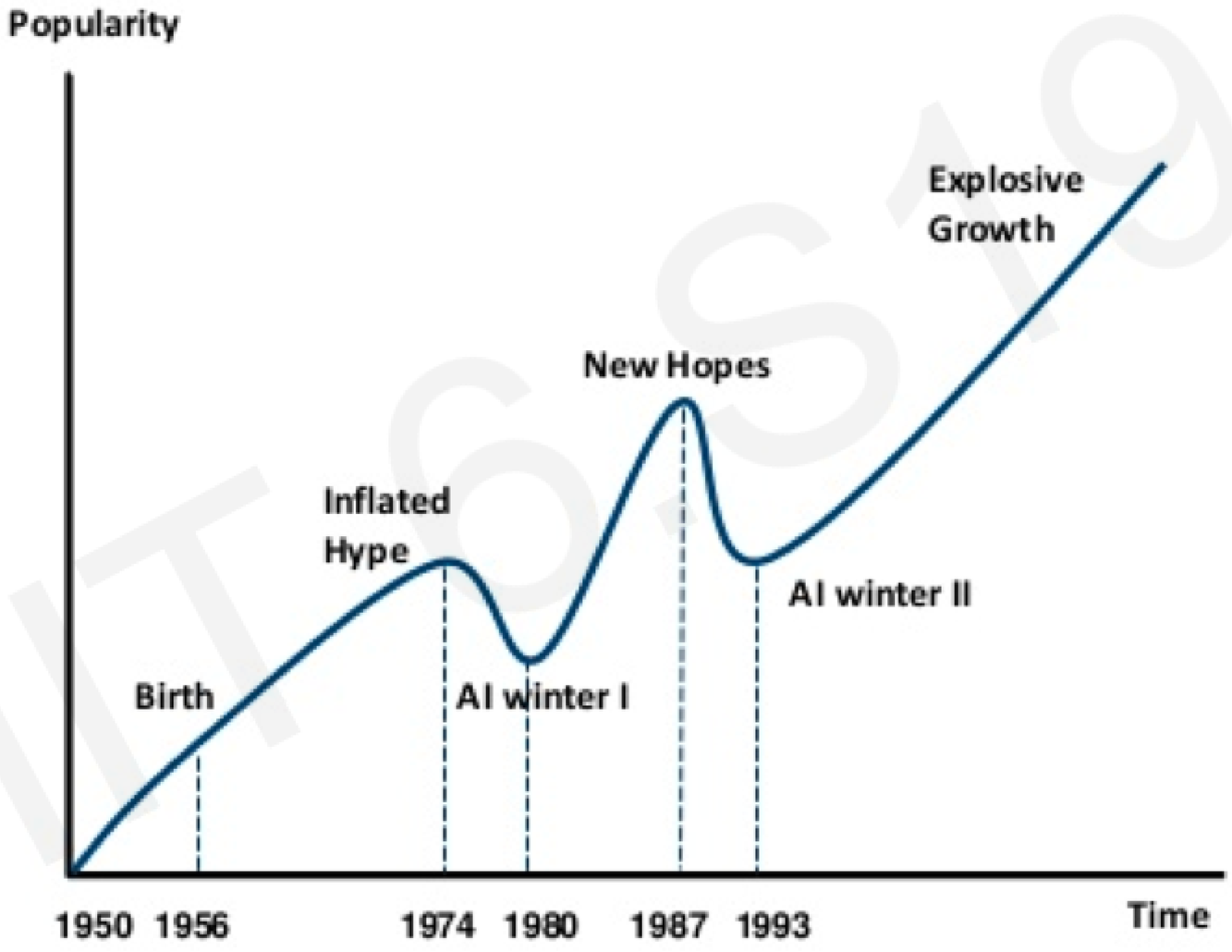
Caveats:

The number of hidden units may be infeasibly large

The resulting model may not generalize

MIT 6.S191

Artificial Intelligence "Hype": Historical Perspective



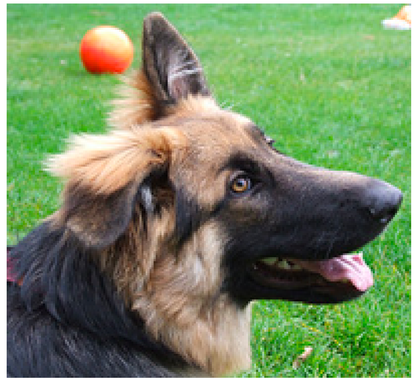
MIT 6.S191

Limitations

MIT 6.S191

Rethinking Generalization

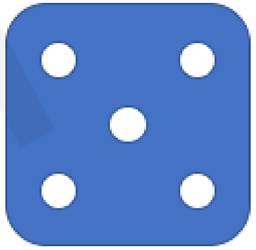
"Understanding Deep Neural Networks Requires Rethinking Generalization"



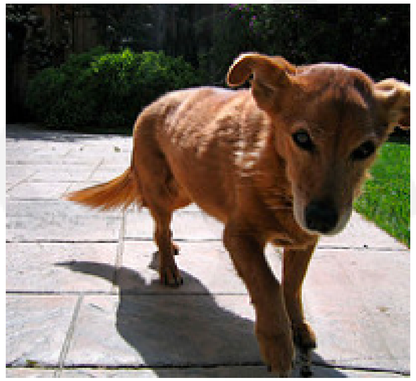
dog

banana

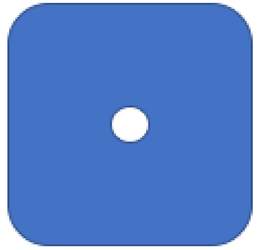
banana



dog

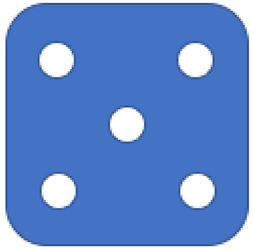


dog



tree

tree

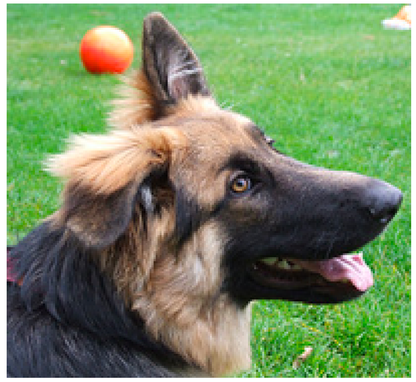


dog

MIT 6.S191

Rethinking Generalization

"Understanding Deep Neural Networks Requires Rethinking Generalization"

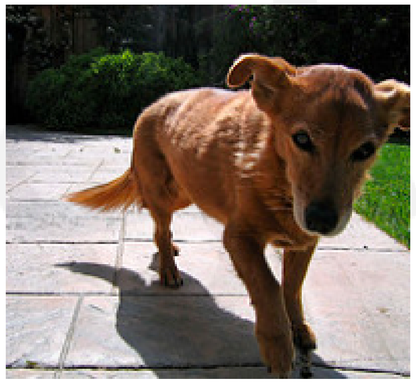


~~dog~~

banana

~~banana~~

dog



~~dog~~

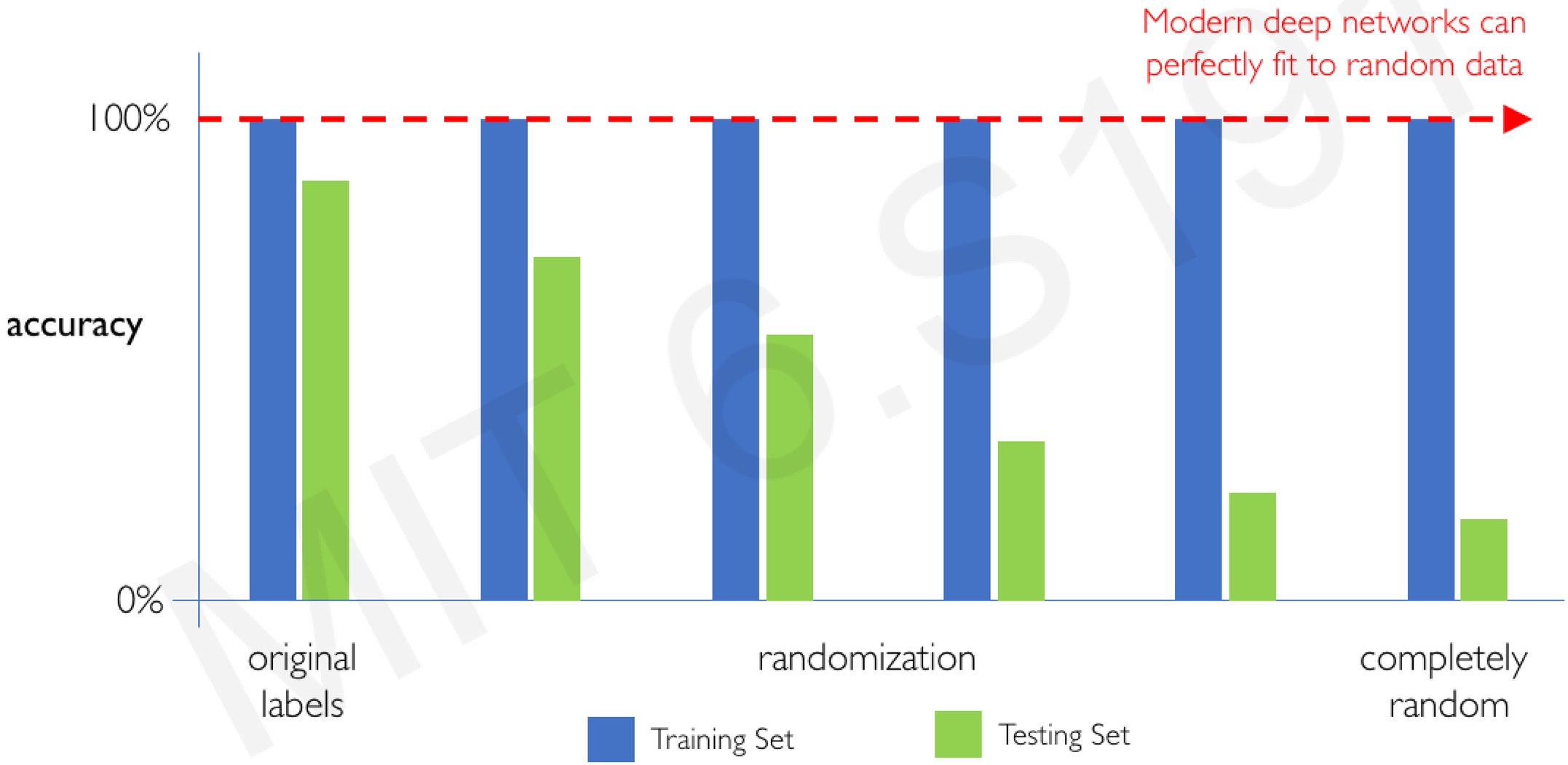
tree

~~tree~~

dog

MIT 6.S191

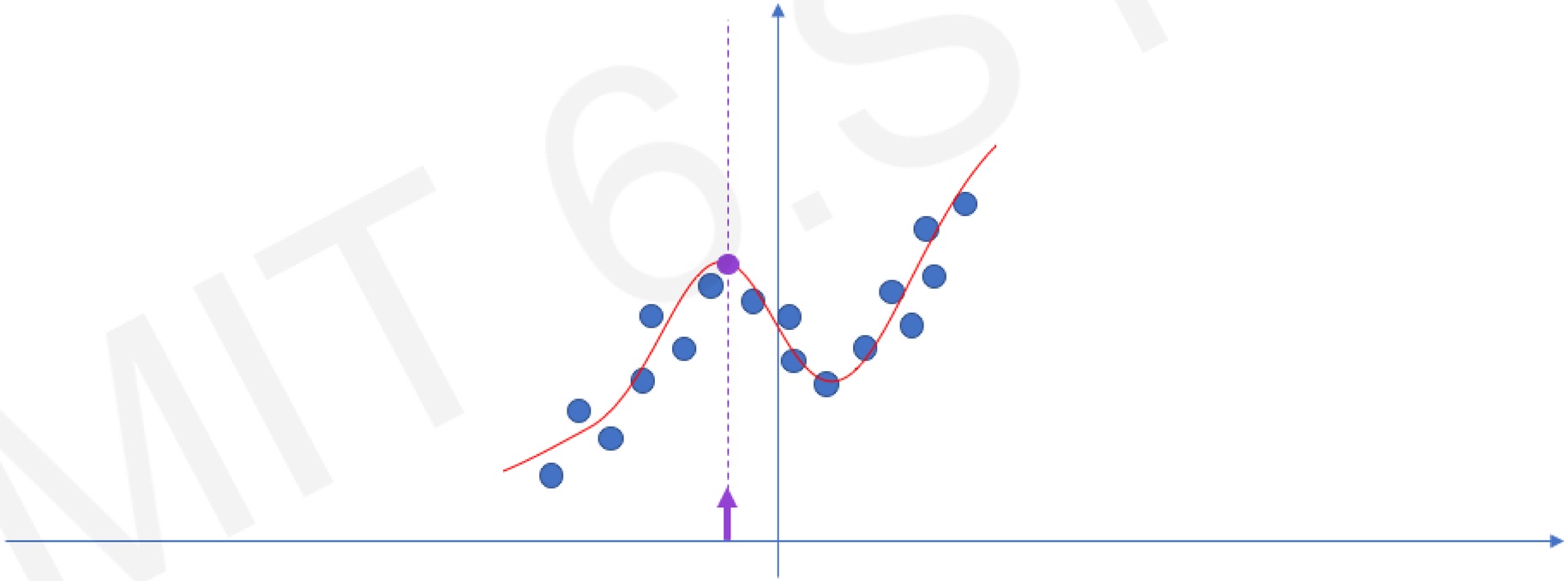
Capacity of Deep Neural Networks



MIT 6.S191

Neural Networks as Function Approximators

Neural networks are **excellent** function approximators

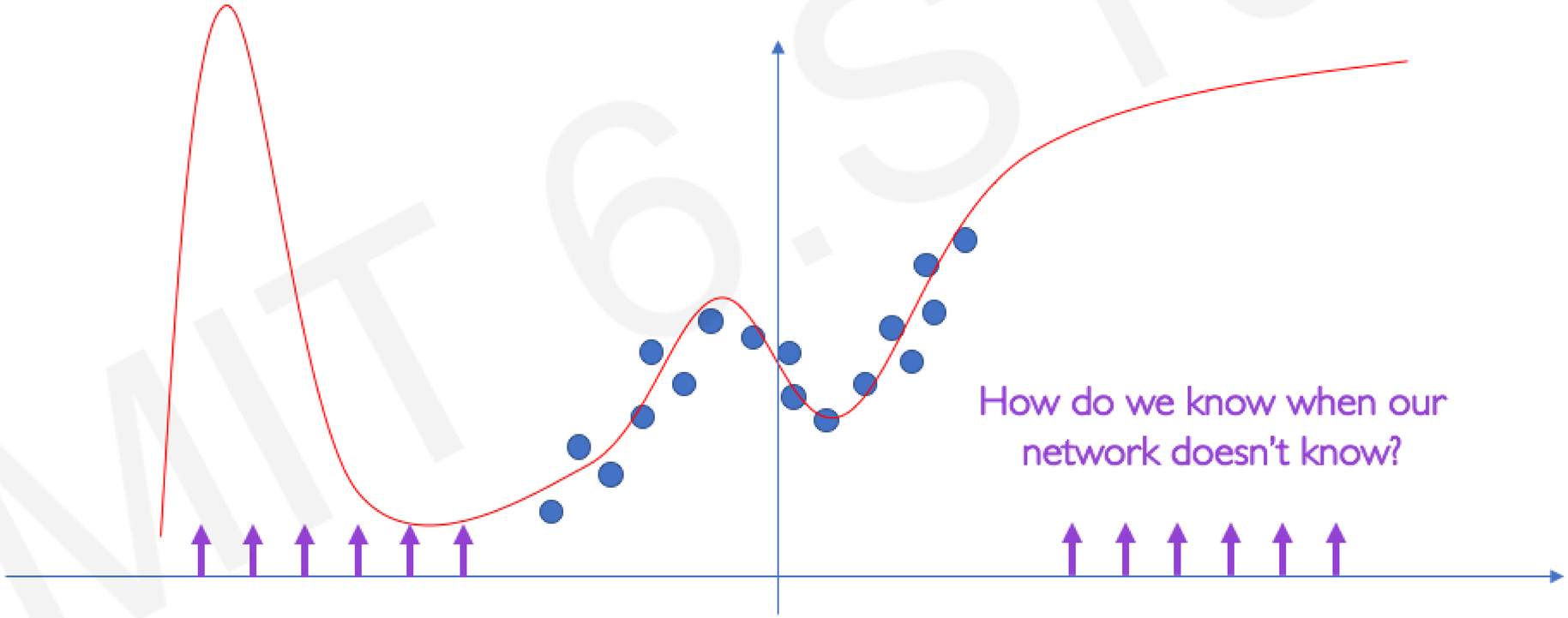


MIT 6.S191

Neural Networks as Function Approximators

Neural networks are **excellent** function approximators

…when they have training data

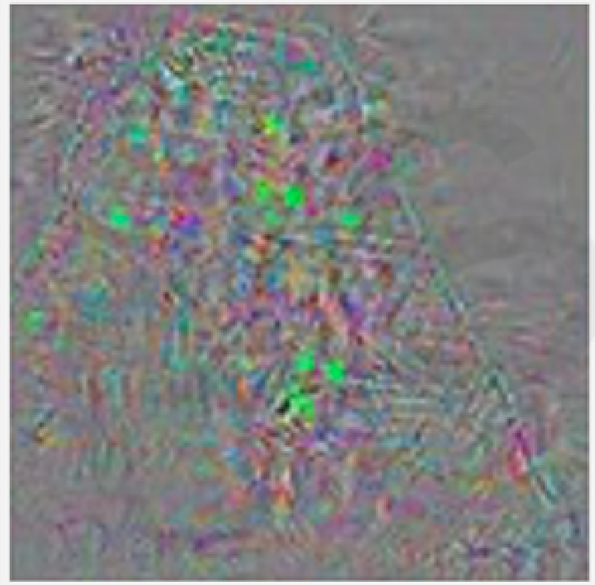


MIT 6.S191

Adversarial Attacks on Neural Networks

Original image

Temple (97%)



Perturbations

Adversarial example

Ostrich (98%)

MIT 6.S191

Adversarial Attacks on Neural Networks

Remember:

We train our networks with gradient descent

$$W \leftarrow W - \eta \frac{\partial J(W, x, y)}{\partial W}$$

Fix your image x, and true label y

"How does a small change in weights decrease our loss"

MIT 6.S191

Adversarial Attacks on Neural Networks

Adversarial Image:

Modify image to increase error

$$x \leftarrow x + \eta \frac{\partial J(W, x, y)}{\partial x}$$

Fix your weights θ, and true label y

"How does a small change in the input increase our loss"

MIT 6.S191

Synthesizing Robust Adversarial Examples

classified as turtle classified as rifle
classified as other

Neural Network Limitations…

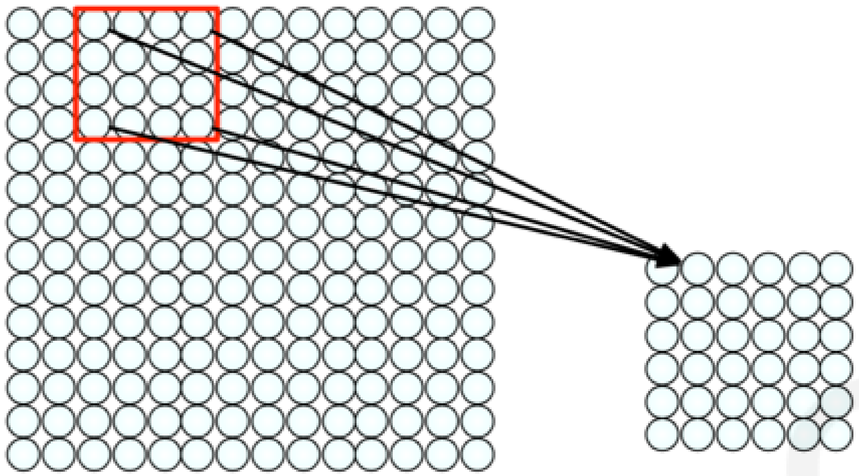
- Very **data hungry** (eg. often millions of examples)
- **Computationally intensive** to train and deploy (tractably requires GPUs)
- Easily fooled by **adversarial examples**
- Can be subject to **algorithmic bias**
- Difficult to **encode structure** and prior knowledge during learning
- Poor at **representing uncertainty** (how do you know what the model knows?)
- Uninterpretable **black boxes**, difficult to trust
- **Finicky to optimize**: non-convex, choice of architecture, learning parameters
- Often require **expert knowledge** to design, fine tune architectures

MIT 6.S191

New Frontiers 1:
Encoding Structure into Deep Learning

MIT 6.S191

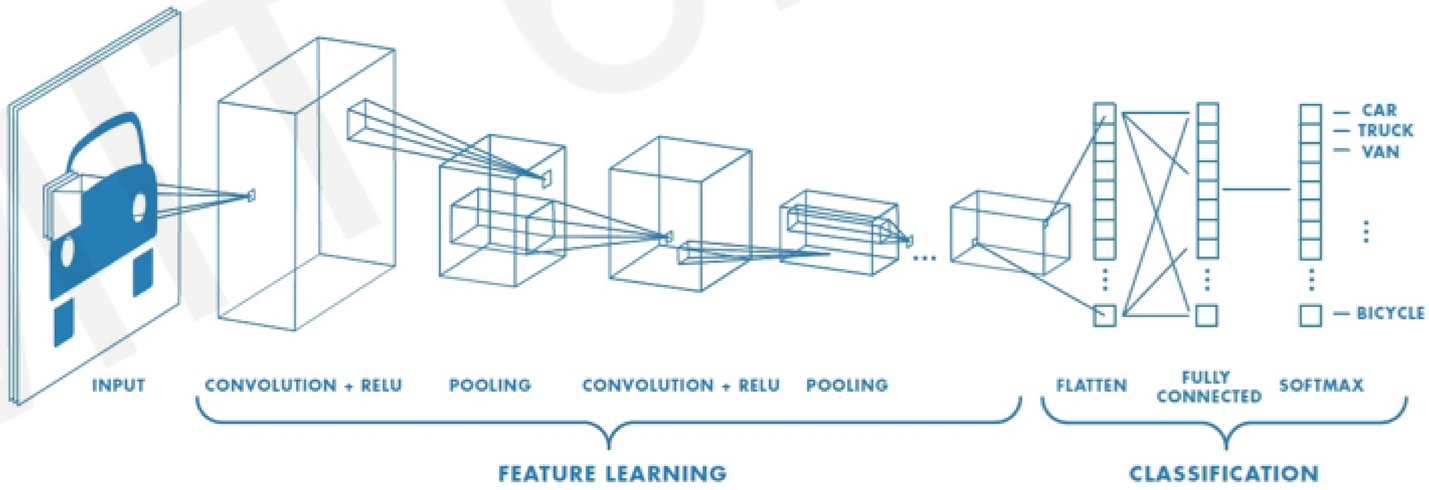
CNNs: Using Spatial Structure



1) Apply a set of weights to extract **local features**

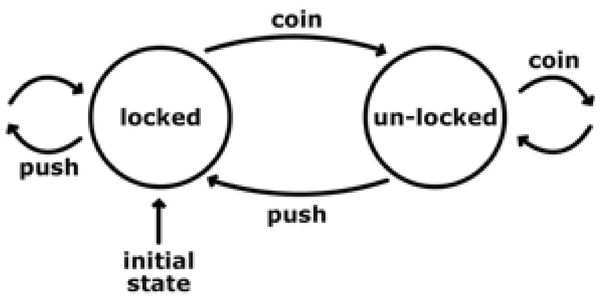
2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

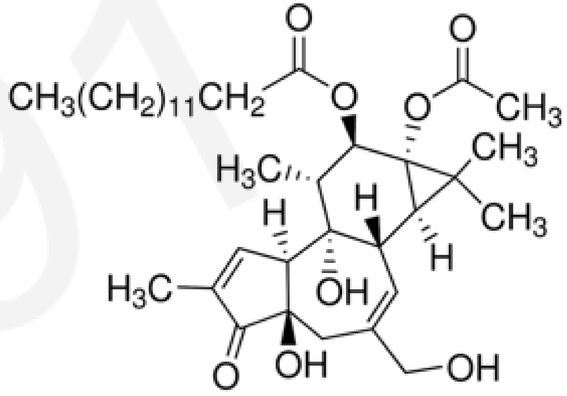


MIT 6.S191

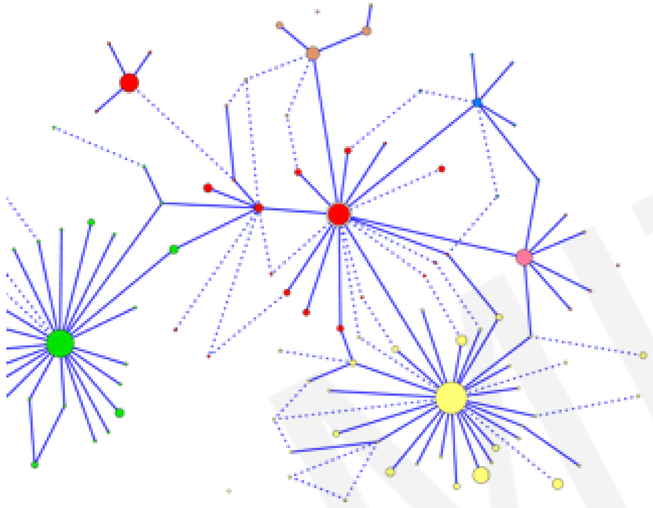
Graphs as a Structure for Representing Data



State Machines



Molecules



Biological Networks

Social Networks

Mobility & Transport

Graph Convolutional Networks

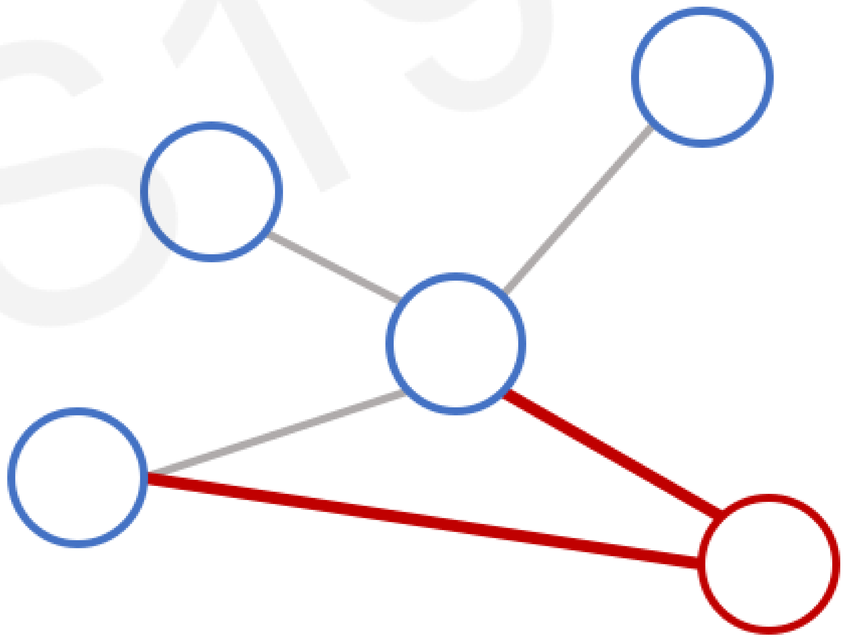
Convolutional Networks

Graph Convolutional Networks (GCNs)

Graph Convolutional Networks

Convolutional Networks

Graph Convolutional Networks (GCNs)

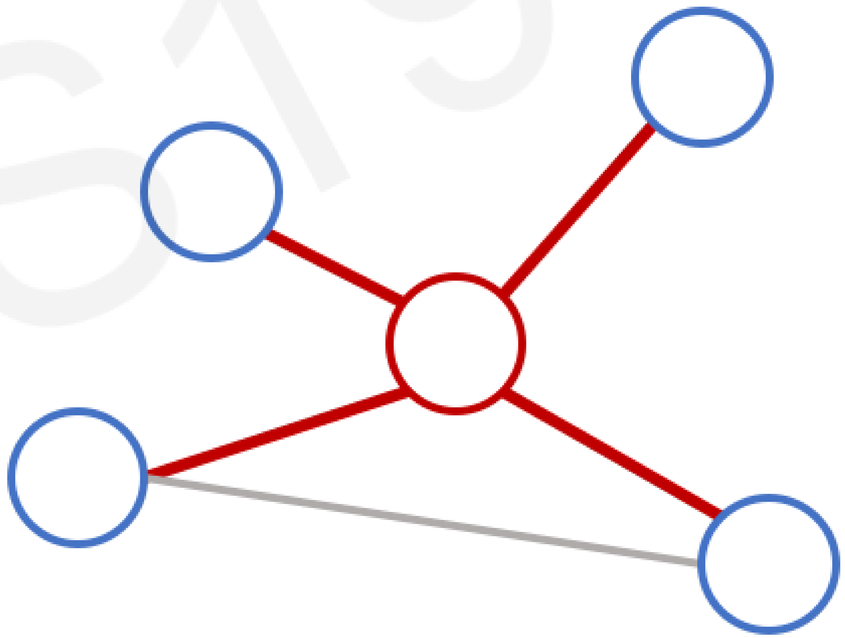


Duvenaud+ *NeurIPS* 2015; Kipf & Welling *ICLR* 2017. 1/29/20

Graph Convolutional Networks

Convolutional Networks

Graph Convolutional Networks (GCNs)



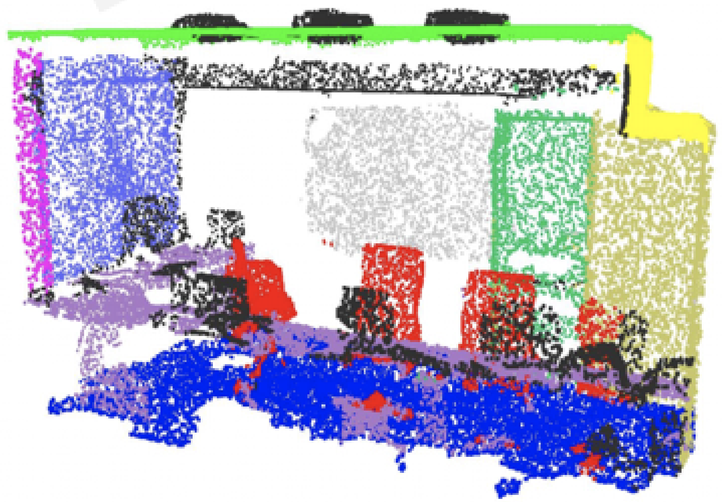
MIT 6.S191

Learning From 3D Data

Point clouds are **unordered sets** with **spatial dependence** between points

Classification

Part Segmentation

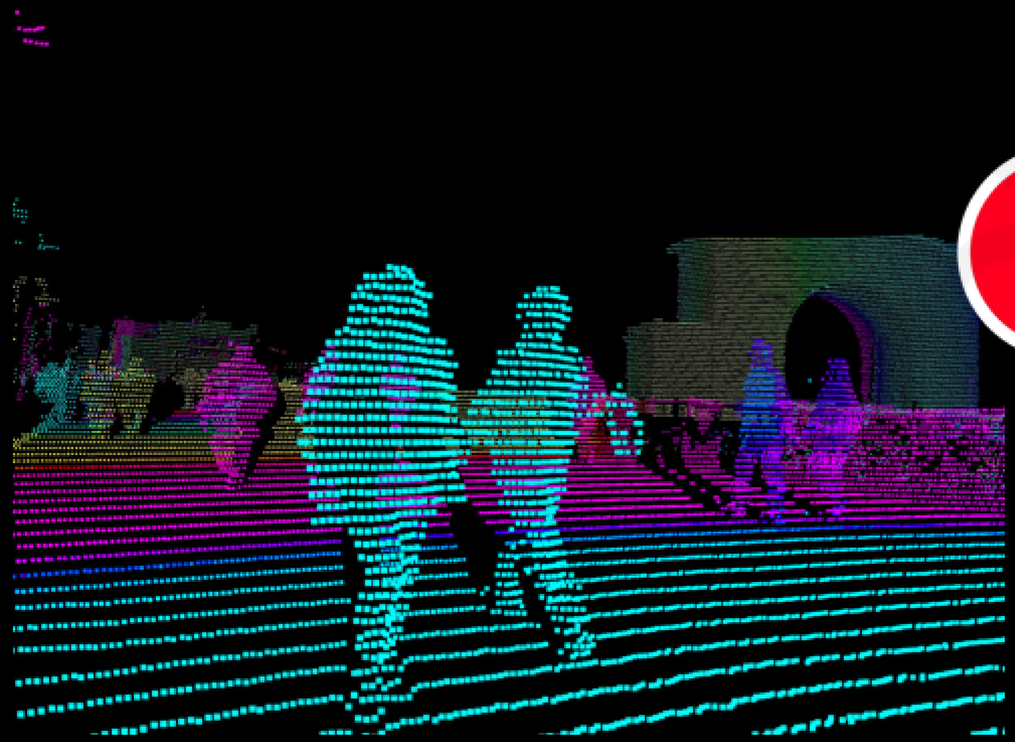


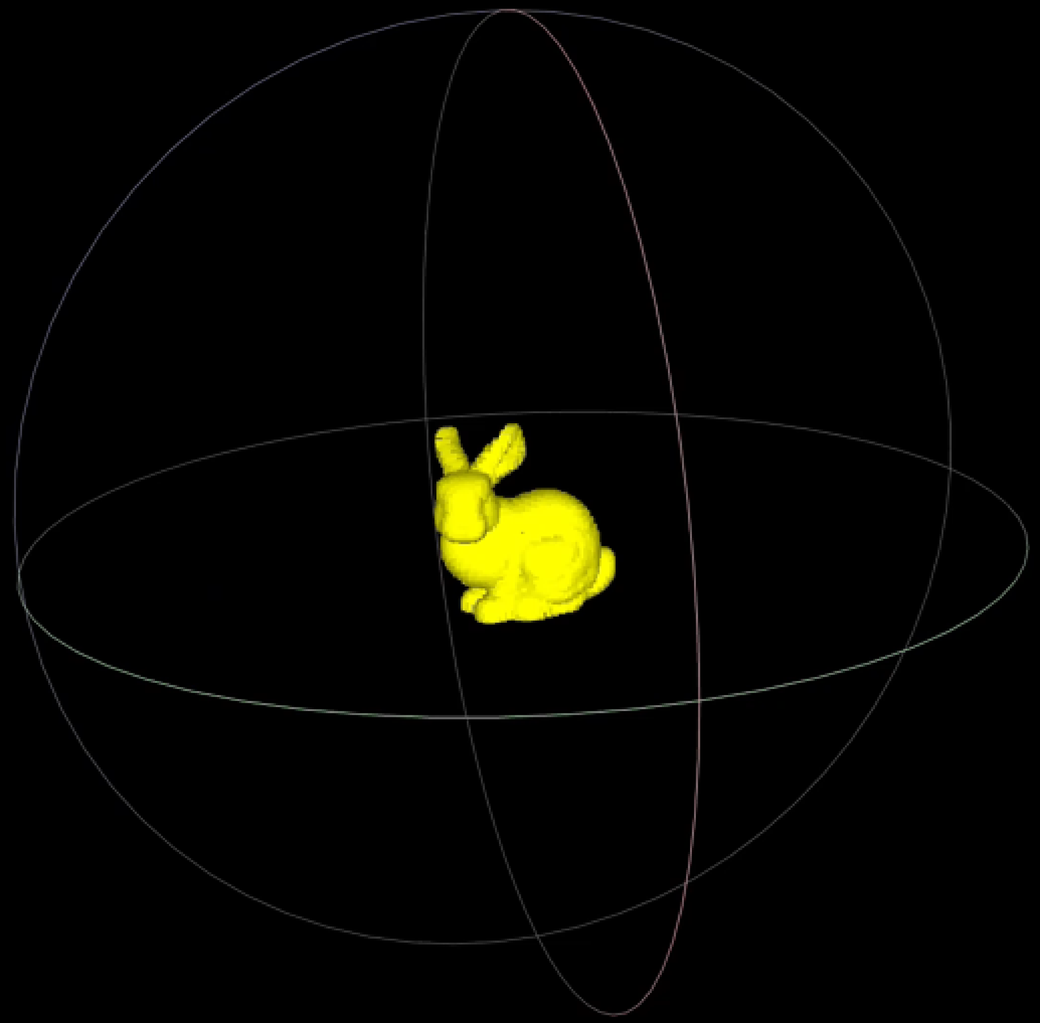
Semantic Segmentation

MIT6.S191

Extending Graph CNNs to Pointclouds

Capture local geometric features of point clouds while maintaining order invariance



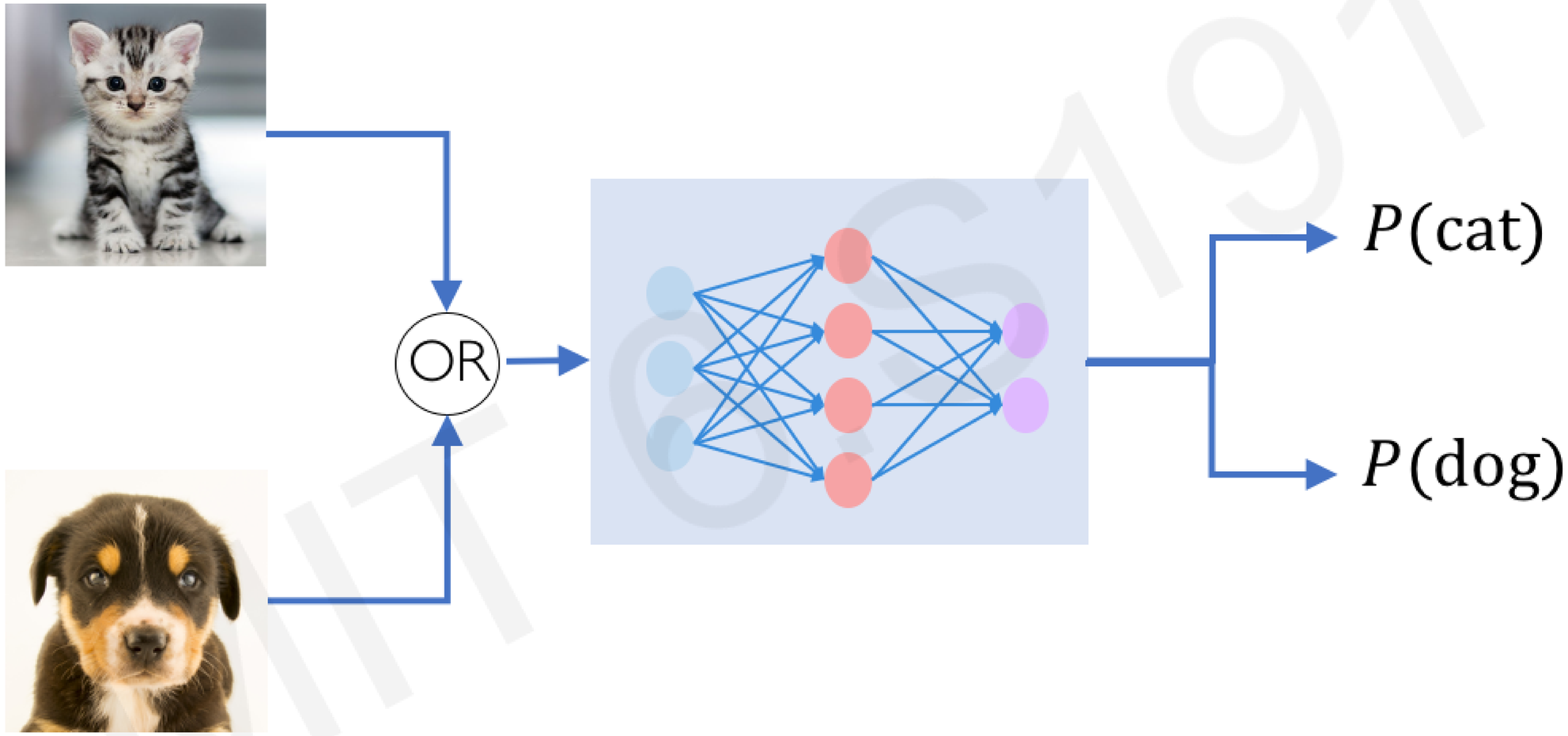


Wang+ *TOG* 2019. 1/29/20

New Frontiers II:
Uncertainty Estimation & Bayesian Deep Learning

MIT 6.S191

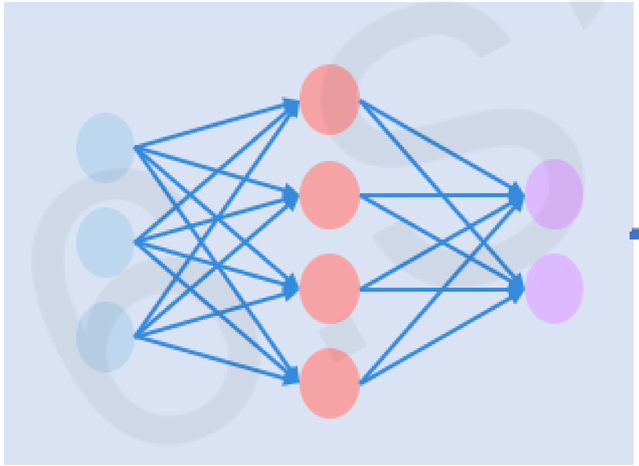
Why care about uncertainty?



MIT 6.S191

Why care about uncertainty?

We need **uncertainty** metrics to assess the network's **confidence** in its predictions.



$P(\text{cat}) = 0.2$

$P(\text{dog}) = 0.8$

Remember: $P(\text{cat}) + P(\text{dog}) = 1$

MIT 6.S191

Bayesian Deep Learning for Uncertainty

Network tries to learn output, $\boldsymbol{Y}$, directly from raw data, $\boldsymbol{X}$

Find mapping, f, parameterized by weights $\boldsymbol{W}$ such that

$$\min \mathcal{L}(\boldsymbol{Y}, f(\boldsymbol{X}; \boldsymbol{W}))$$

Bayesian neural networks aim to learn a posterior over weights, $P(\boldsymbol{W}|\boldsymbol{X}, \boldsymbol{Y})$:

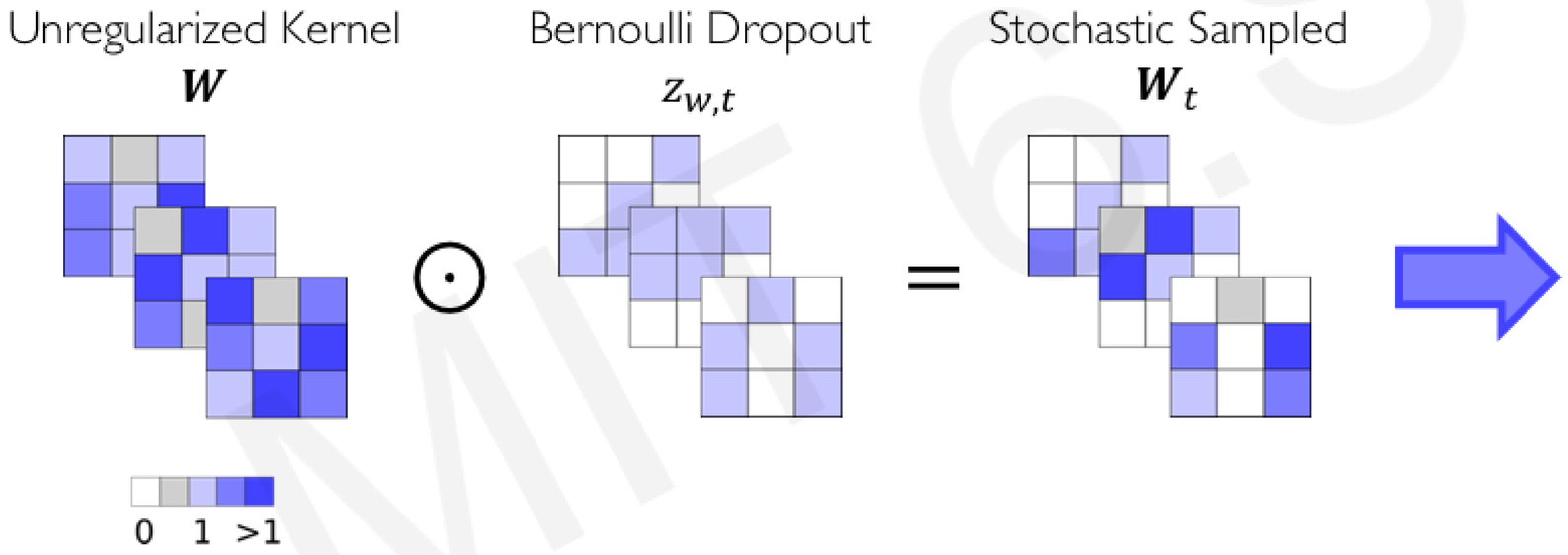
Intractable! $$P(\boldsymbol{W}|\boldsymbol{X}, \boldsymbol{Y}) = \frac{P(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{W})P(\boldsymbol{W})}{P(\boldsymbol{Y}|\boldsymbol{X})}$$

MIT 6.S191

Dropout for Uncertainty

Evaluate T stochastic forward passes through the network $\{\boldsymbol{W}_t\}_{t=1}^{T}$

Dropout as a form of stochastic sampling $z_{w,t} \sim Bernoulli(p) \quad \forall\, w \in \boldsymbol{W}$



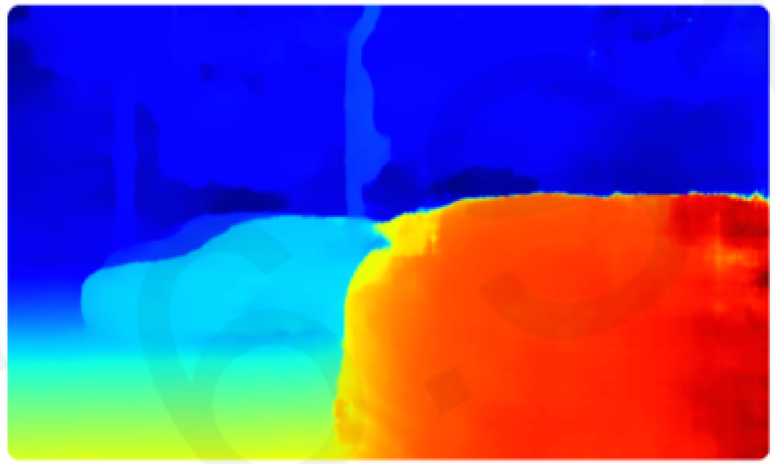
$$\mathbb{E}(\hat{\boldsymbol{Y}}|\boldsymbol{X}) = \frac{1}{T}\sum_{t=1}^{T} f(\boldsymbol{X}|\boldsymbol{W}_t)$$

$$Var(\hat{\boldsymbol{Y}}|\boldsymbol{X}) = \frac{1}{T}\sum_{t=1}^{T} f(\boldsymbol{X})^2 - \mathbb{E}(\hat{\boldsymbol{Y}}|\boldsymbol{X})^2$$

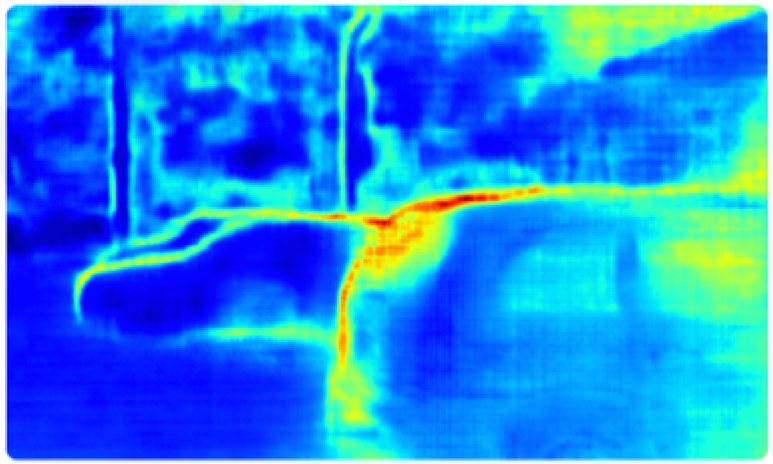
MIT 6.S191

Model Uncertainty Application

Input Image



Predicted Depth

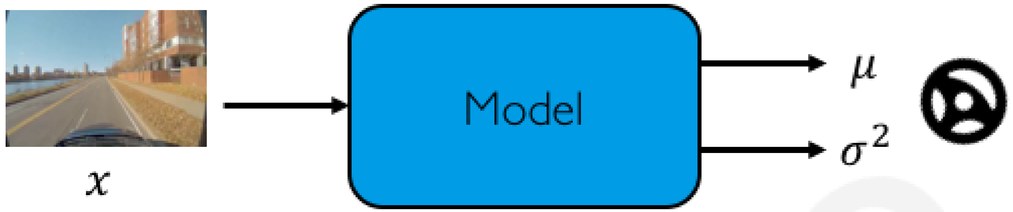


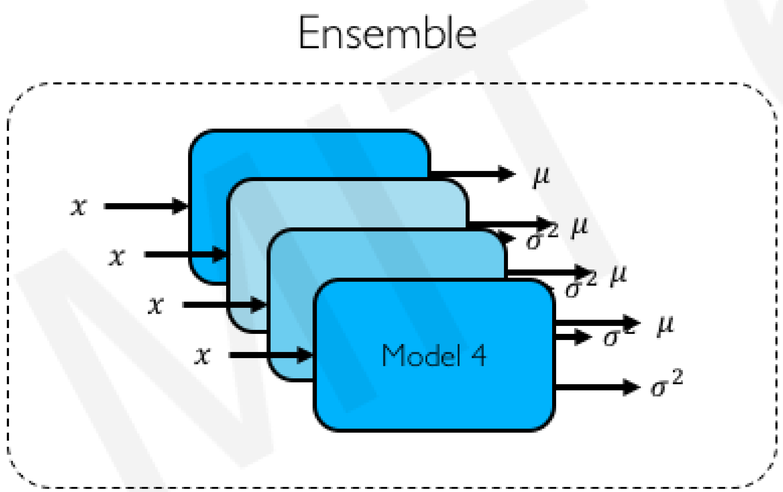
Model Uncertainty

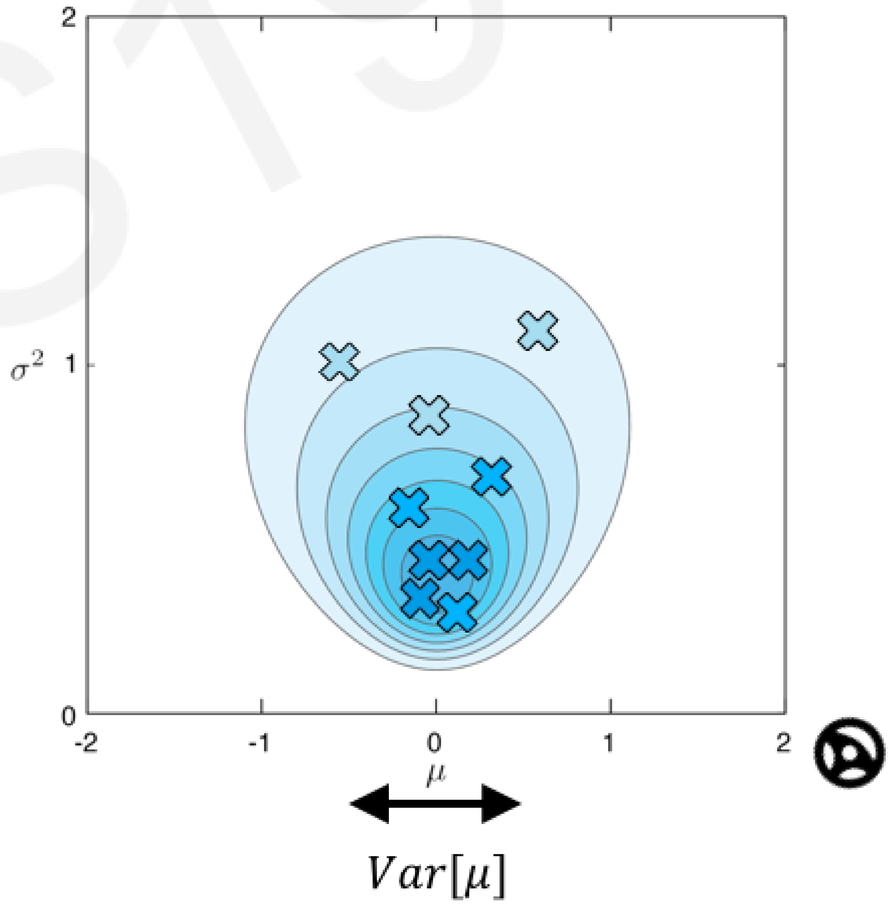
MIT 6.S191

Uncertainty Estimation via Ensembling

Model ensembling for estimating uncertainty





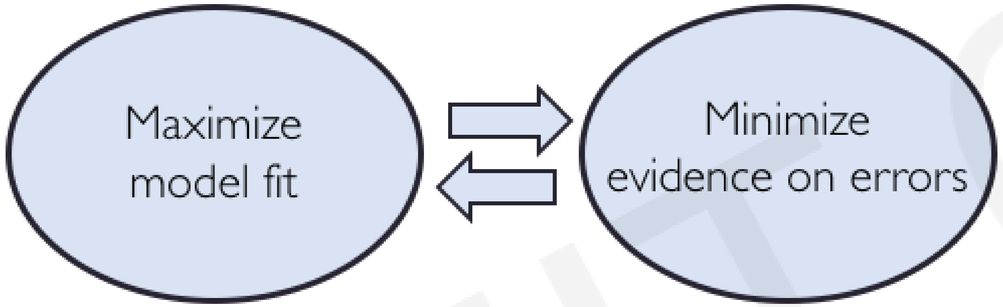


MIT 6.S191

Evidential Deep Learning

Directly learn the underlying uncertainties using **evidential distributions**

Competing loss training:

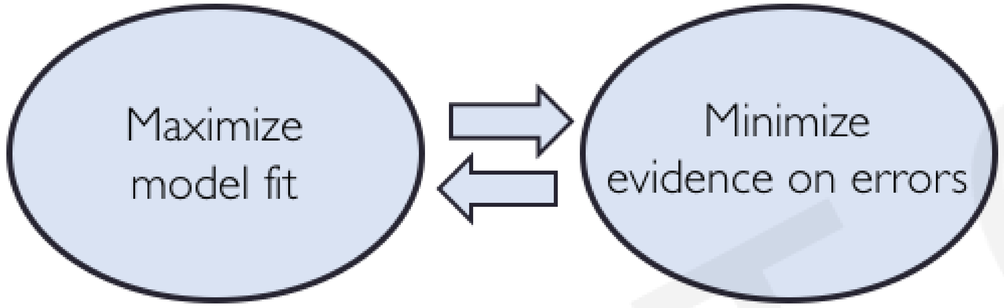


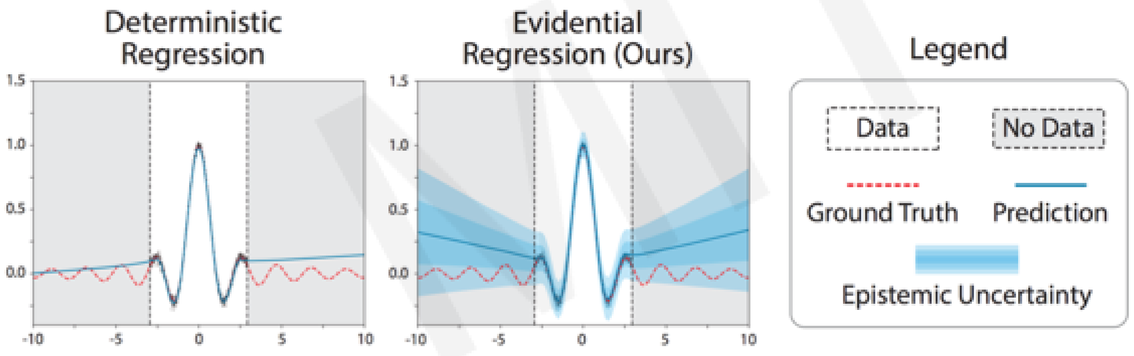
MIT 6.S191

Evidential Deep Learning

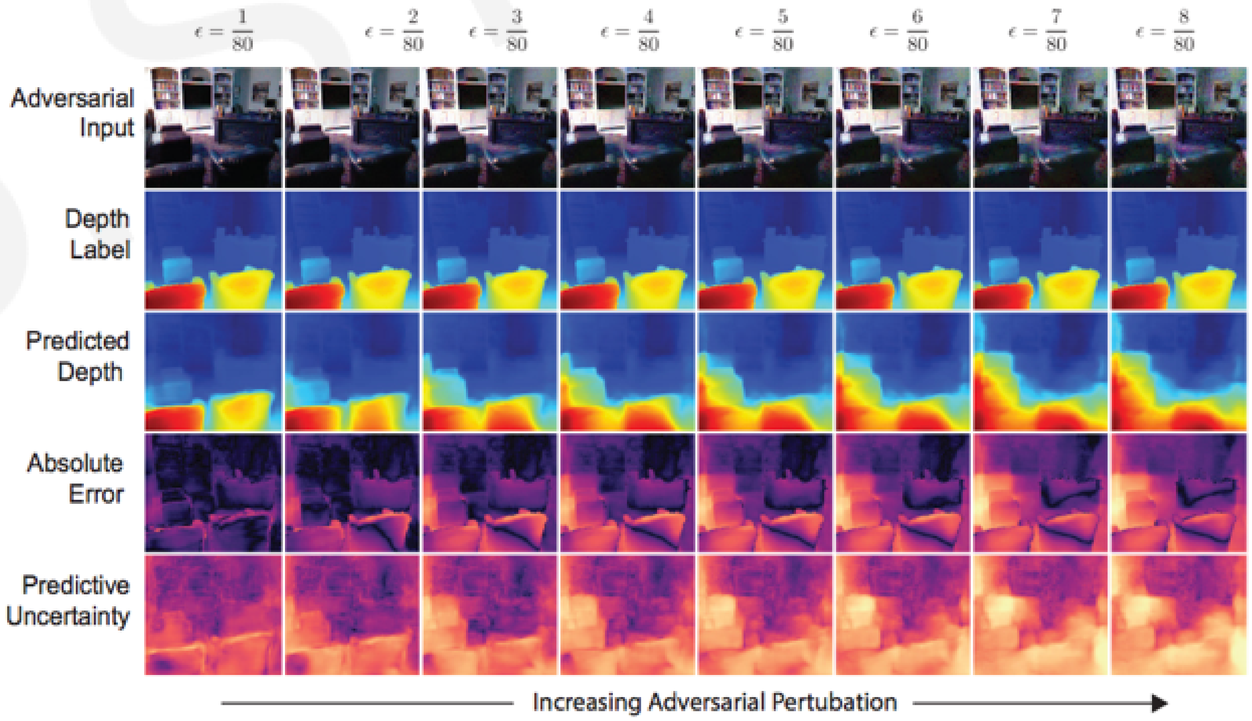
Directly learn the underlying uncertainties using **evidential distributions**

Competing loss training:

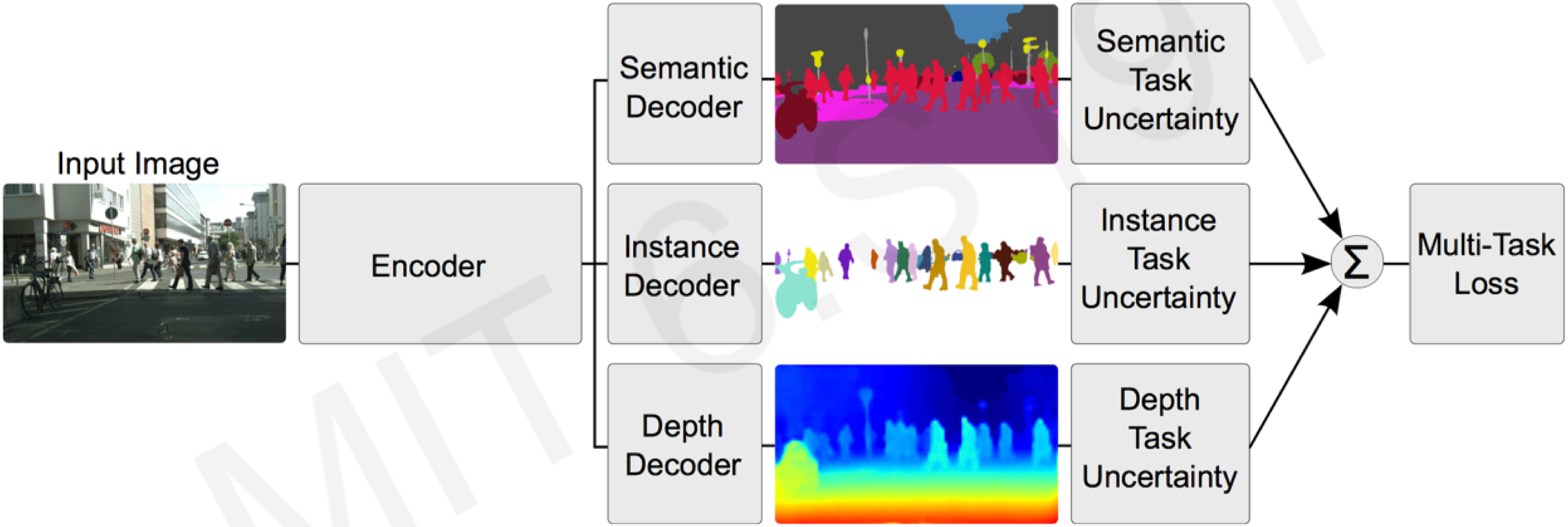




Robustness to adversarial perturbation

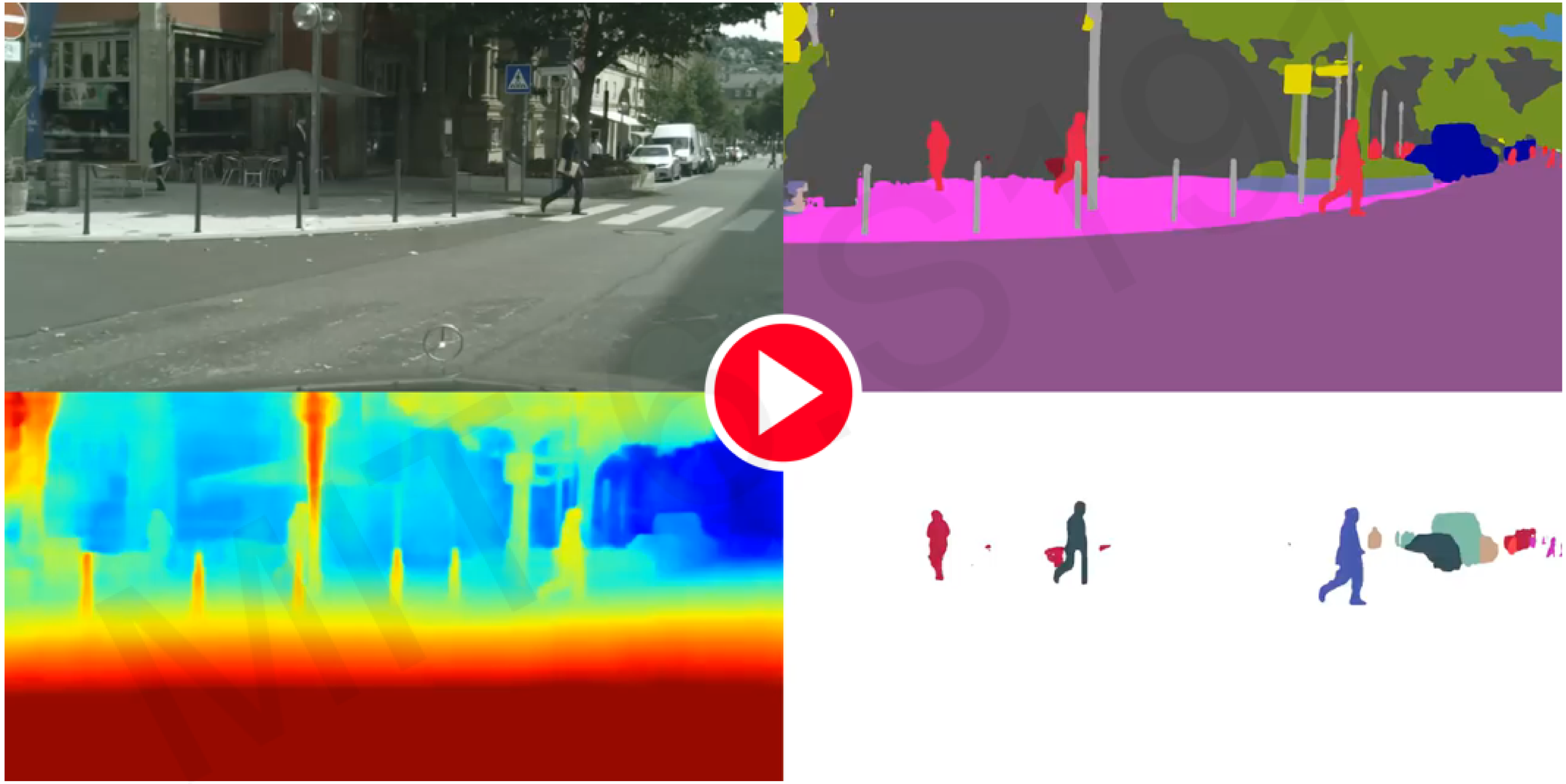


Multi-Task Learning Using Uncertainty



MIT 6.S191

Multi-Task Learning Using Uncertainty



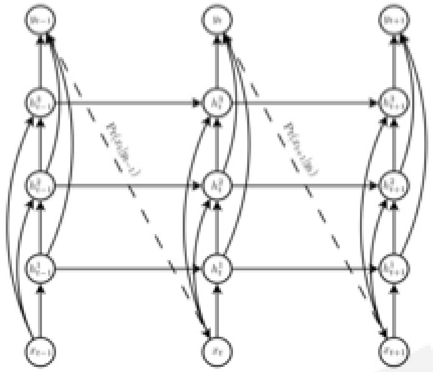
Kendall+ *CVPR* 2018. 1/29/20

New Frontiers III:
Automated Machine Learning

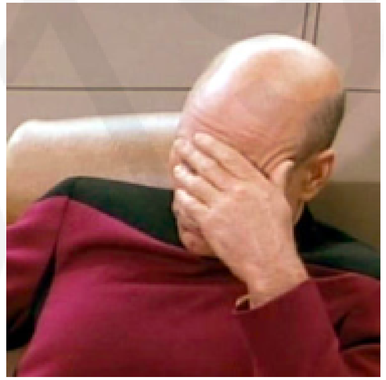
MIT 6.S191

Motivation: Automated Machine Learning

Standard deep neural networks are optimized for **a single task**



Complexity of models increases



Greater need for specialized engineers

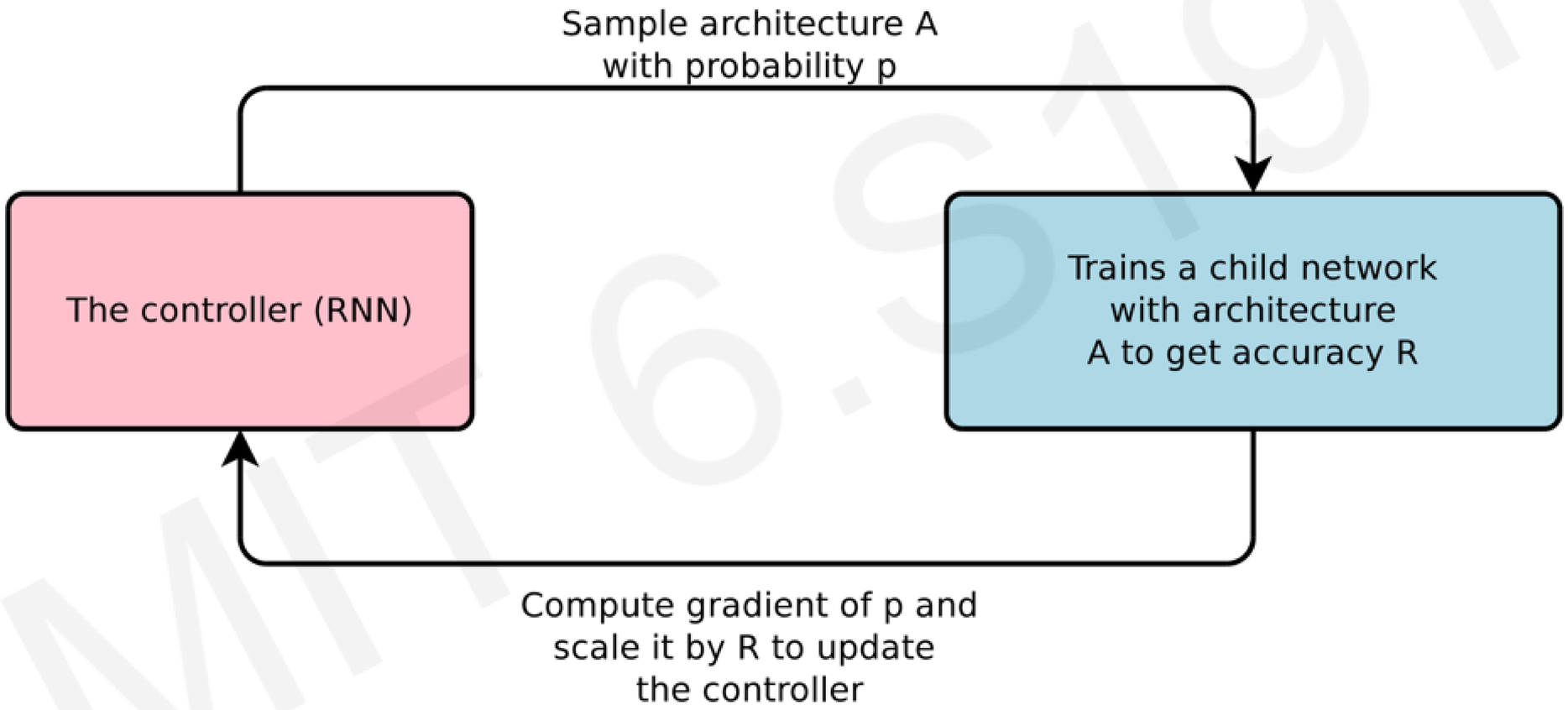
Often require **expert knowledge** to build an architecture for a given task

Build a learning algorithm that **learns which model** to use to solve a given problem

AutoML

MIT 6.S191

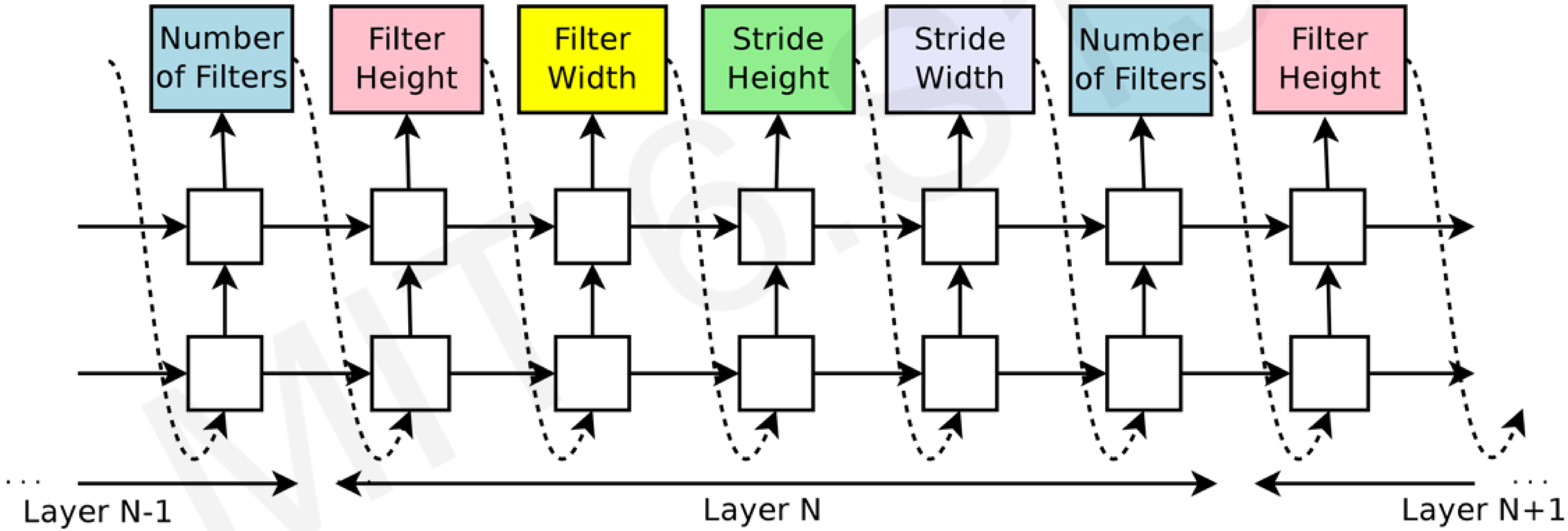
Automated Machine Learning (AutoML)



MIT 6.S191

AutoML: Model Controller

At each step, the model samples a brand new network



MIT 6.S191

Zoph and Le *ICLR* 2017. 1/29/20

AutoML: The Child Network

Training Data → Sampled network from RNN → Prediction

Compute final accuracy on this dataset.
Update RNN controller based on the accuracy of the child network after training.

MIT 6.S191

AutoML on the Cloud

AutoML Vision[BETA]

Start with as little as a few dozen photographic samples, and Cloud AutoML will do the rest.

AutoML Natural Language[BETA]

Automatically predict text categories through either single or multi-label classification.

AutoML Translation[BETA]

Upload translated language pairs to train your own custom model.

MIT 6.S191

AutoML Spawns a Powerful Idea

- Design an AI algorithm that can build new models capable of solving a task
- Reduces the need for experienced engineers to design the networks
- Makes deep learning more accessible to the public

Connections and distinctions between artificial and human intelligence