Introduction to Deep Learning

Démarrage 9h10

Vincent Havard,
Enseignant-chercheur, CESI LINEACT,
Rouen

2022

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

« le projet DEFI&Co est cofinancé au titre du Programme d’Investissements d’Avenir lancé par l’Etat »

Outline

1. What and Why Deep Learning?
2. The Perceptron
3. Going Deep
4. Applying Neural Networks
5. Training Neural Networks
6. Backpropagation
7. Neural Networks in Practice
8. Mini Batches
9. Conclusion

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

What is deep Learning ?

Artificial Intelligence

Any technique that enables computers to mimic human behaviour

Machine Learning

Ability to learn without explicitly being programmed

Deep Learning

Extract patterns from data using neural networks

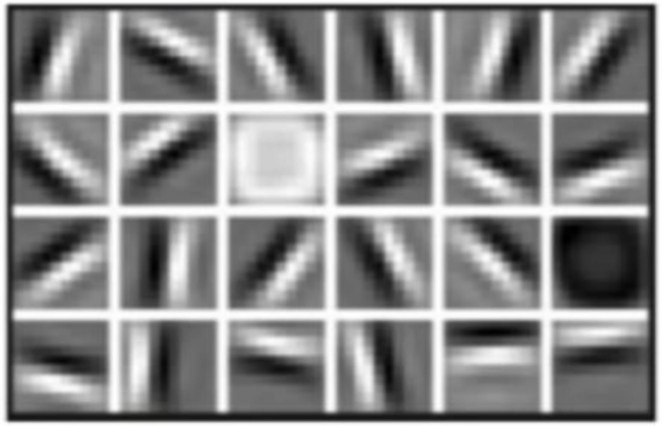
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Why deep Learning?

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

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

Low level features



Lines & edges

Mid level features

Eyes, Nose, Ears

High level features

Facial structure

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Why deep Learning ?

Rule-based programmation paradigm

Data → **Rules** → **Answers**

Machine learning / Deep learning paradigm

Data → **Rules** → **Answers**

Why now?

1952		Stochastic Gradient Descent
1958		Perceptron • Learnable weights
⋮		
1986		Backpropagation • Multi-layer perceptron
⋮		
1995		Deep Convolutional NN • Digit recognition
⋮		
2017		Tensorflow • 1st release

Neural networks date back decades, so why the resurgence?

1. Big Data

- Larger datasets
- Easier collection and storage

2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parrallelizable

3. Software

- Improved Techniques
- New Models
- Toolboxes

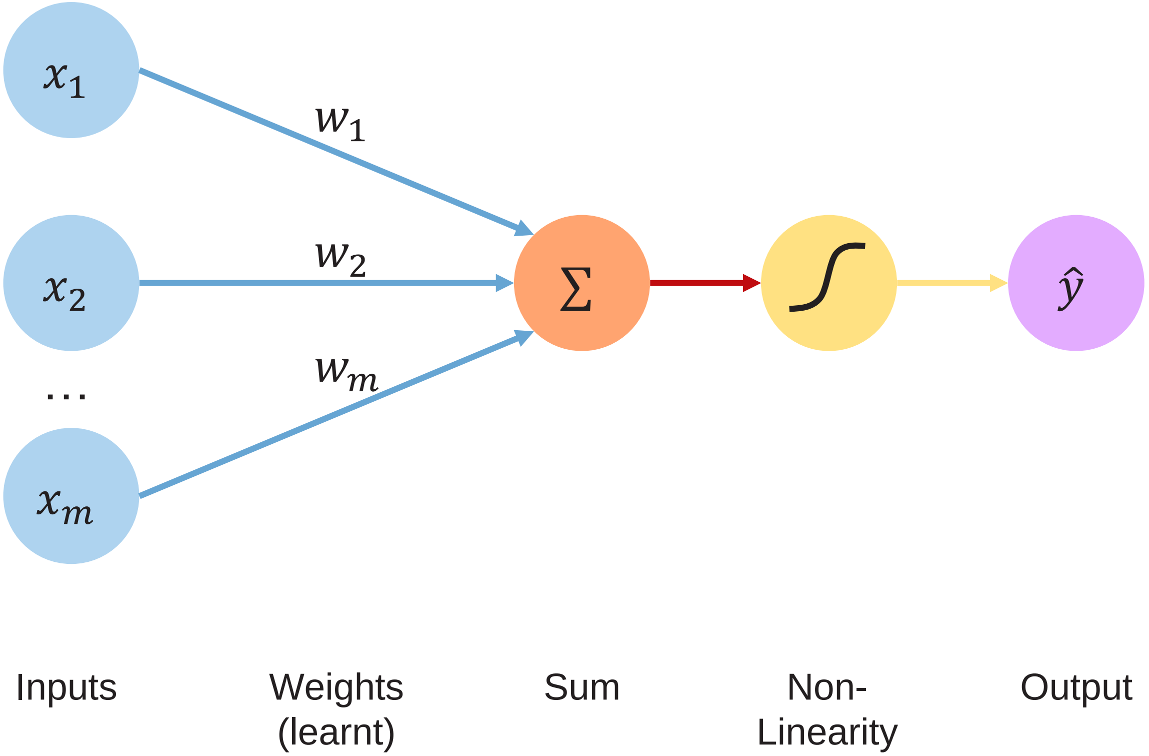
TensorFlow

PyTorch

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron

The Perceptron: Forward Propagation



Linear combination of inputs

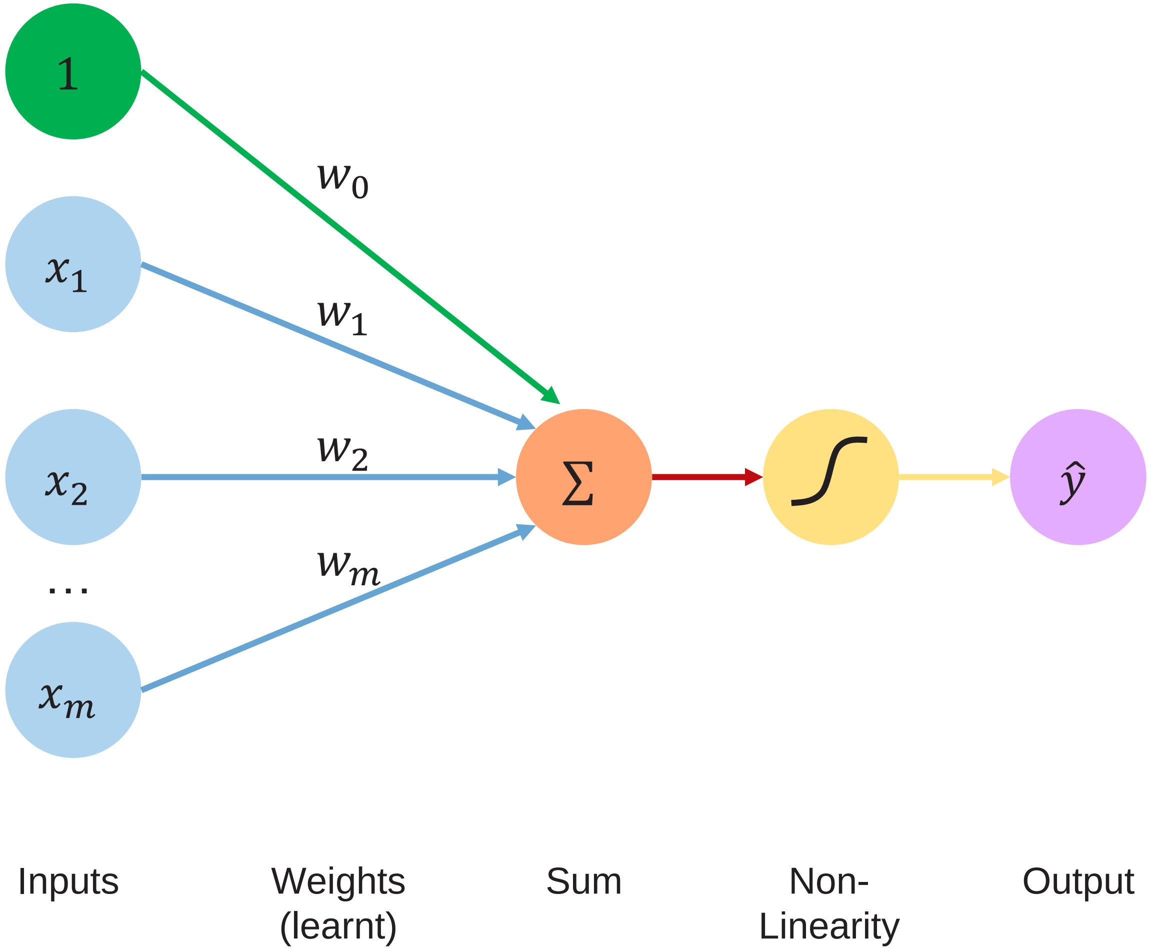
Output

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

Non-Linear activation function

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron: Forward Propagation



Output

Bias

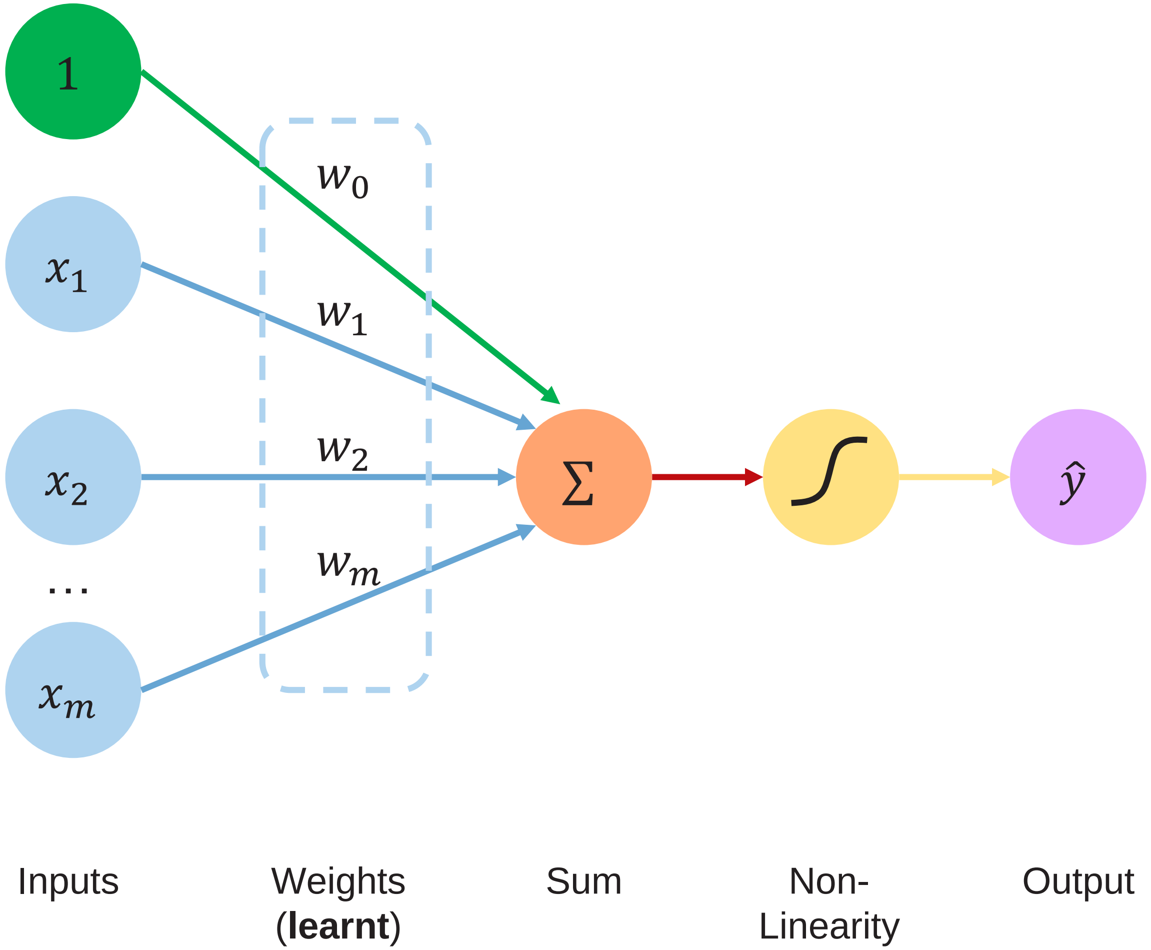
Linear combination of inputs

$$\hat{y} = g\left(w0 + \sum_{i=1}^{m} x_i \cdot w_i\right)$$

Non-Linear activation function

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron: Forward Propagation



$$\hat{y} = g\left(w0 + \sum_{i=1}^{m} x_i \cdot w_i\right)$$

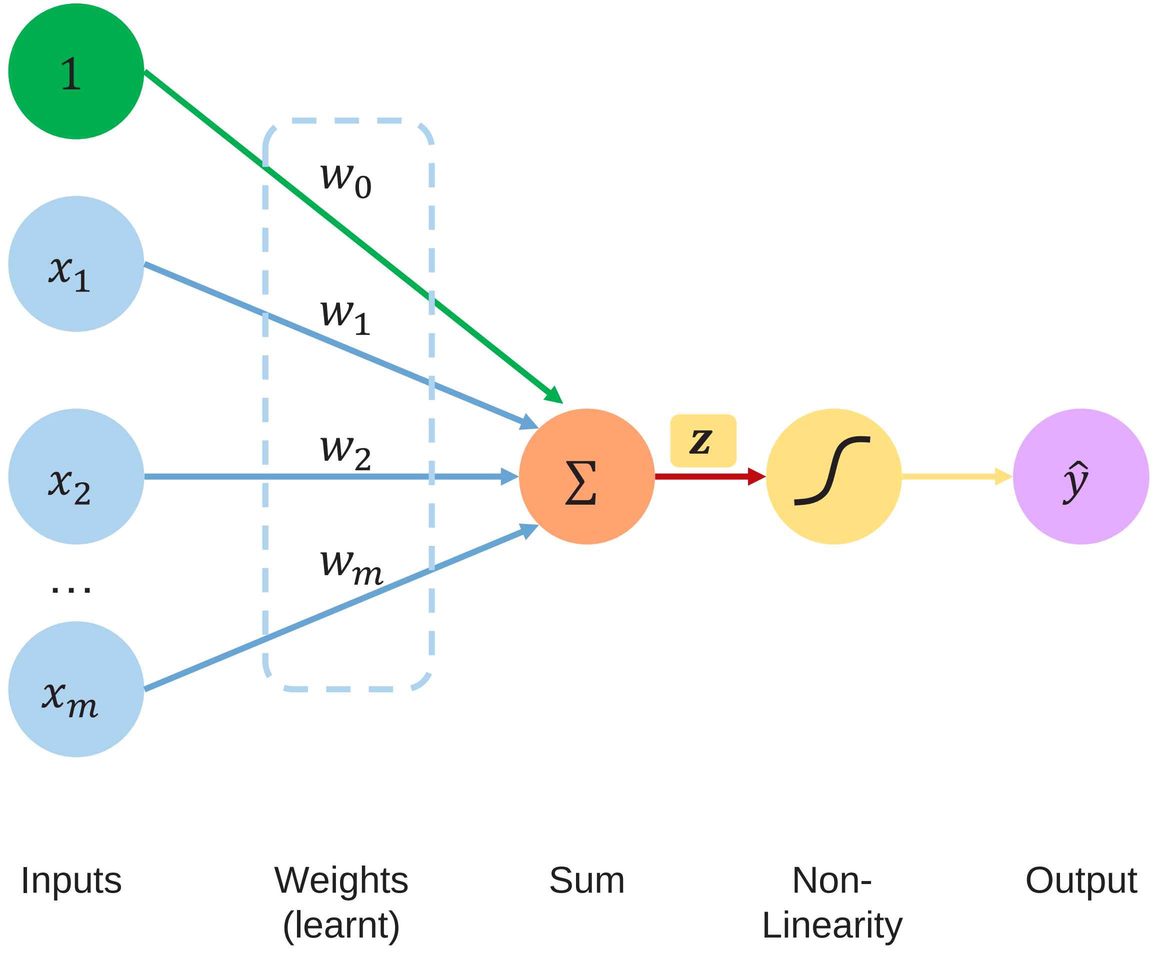
$$\hat{y} = g(w0 + X^T W)$$

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

where $\boldsymbol{X^T} = [\boldsymbol{x_1} \quad \dots \quad \boldsymbol{x_m}]$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron: Forward Propagation



$$\hat{y} = g\left(w0 + \sum_{i=1}^{m} x_i \cdot w_i\right)$$

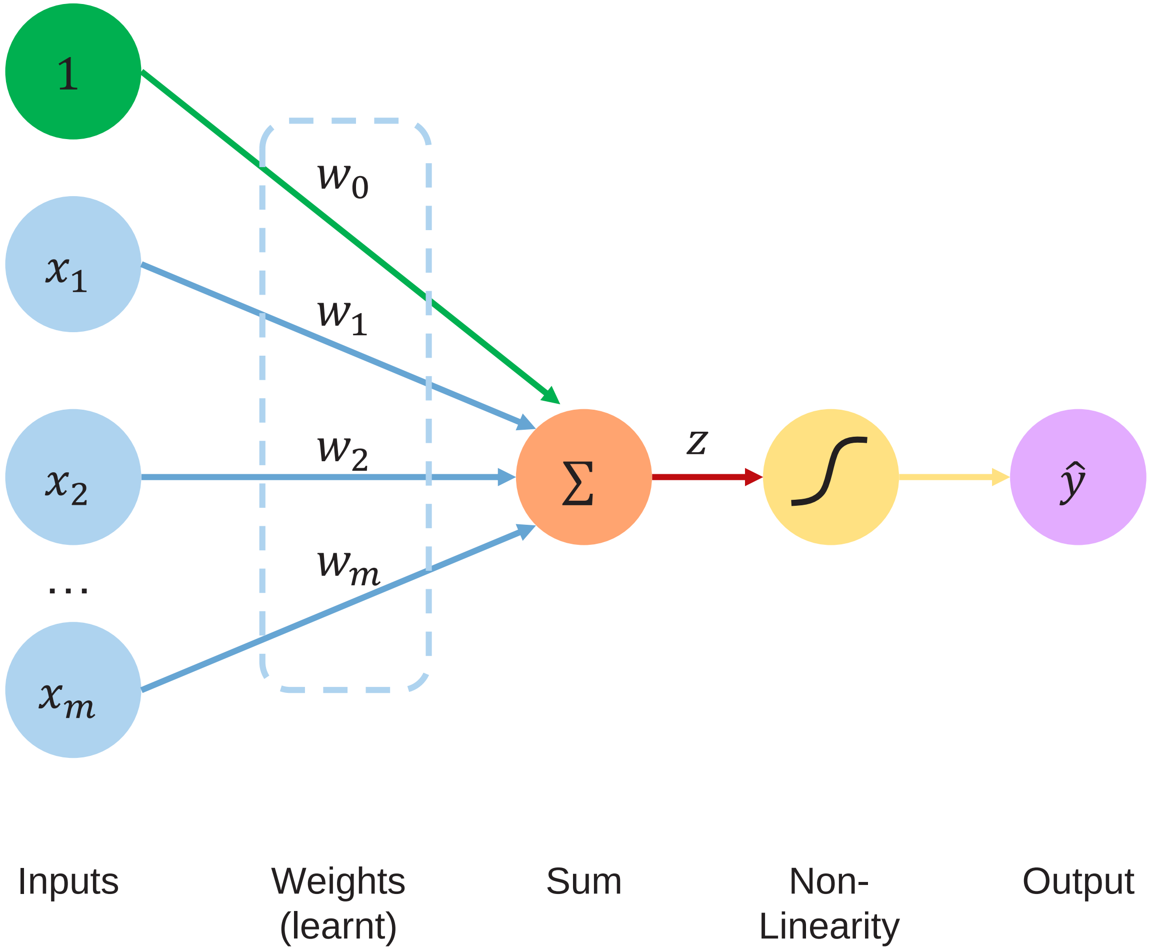
With, $\boldsymbol{z = w0 + X^T W}$

$\hat{y} = g(z)$

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

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron: Forward Propagation

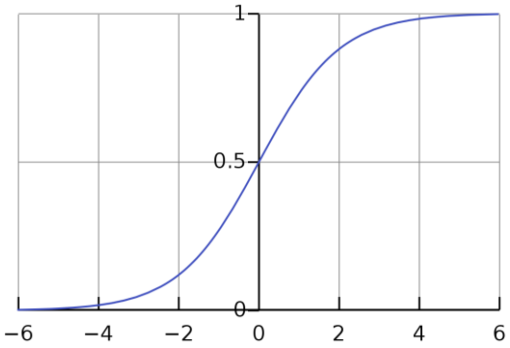


Activation functions

$$\hat{y} = g(w0 + X^T W) = g(z)$$

Example of the sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \in]0,1[$$

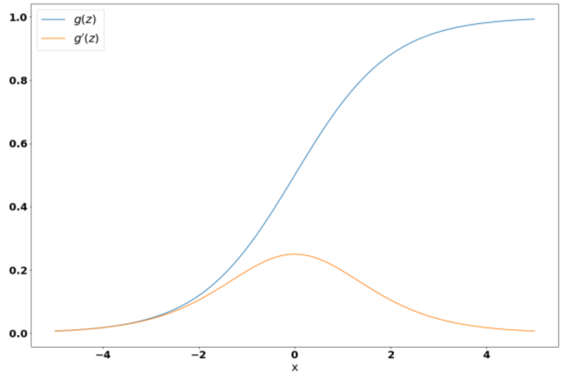


$$g'^{(z)} = \sigma(z) \cdot (1 - \sigma(z))$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Common Activation Functions

Sigmoid Function

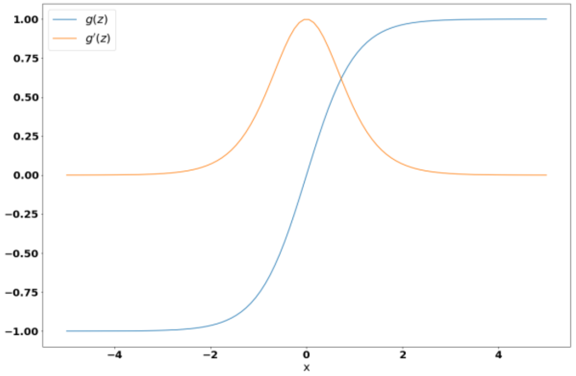


$$g(z) = \frac{1}{1+e^{-z}} \quad \in \,]0,1[$$

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

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

Hyperbolic Tangent (tanh)



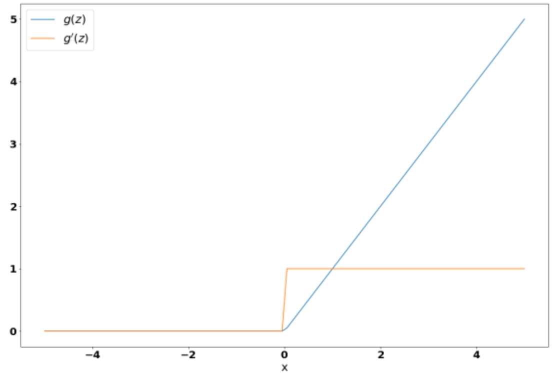
$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \quad \in \,]-1,1[$$

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

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

```
np.tanh(z)
```

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z) \quad \in [0, +\infty[$$

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

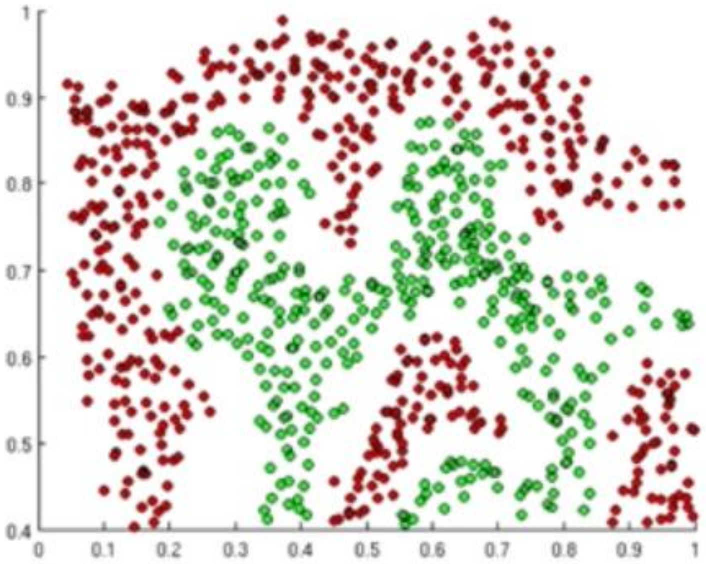
```
tf.nn.relu(z)
```

All activation functions are non-linear

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

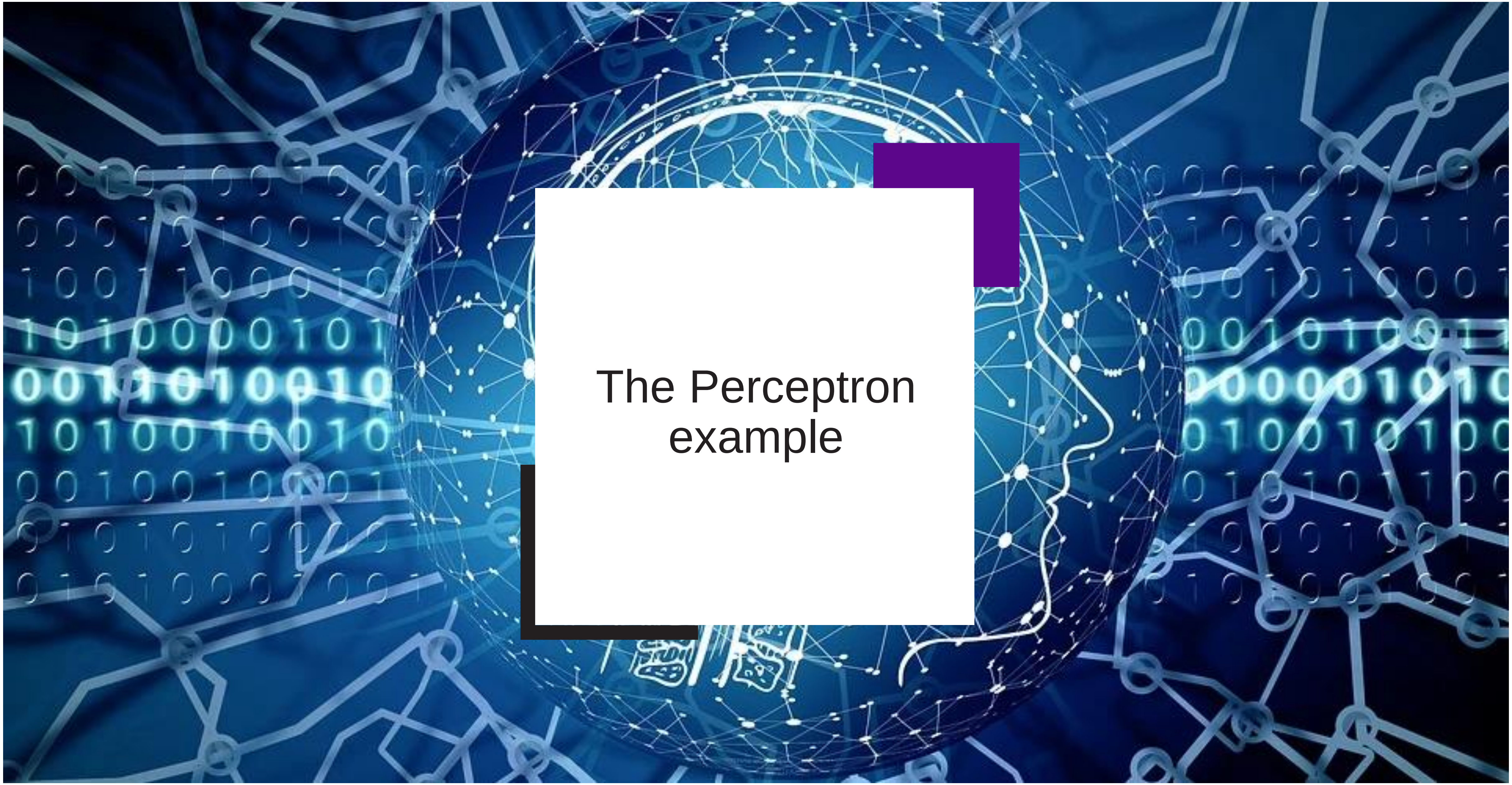
Importance of Activation Functions

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



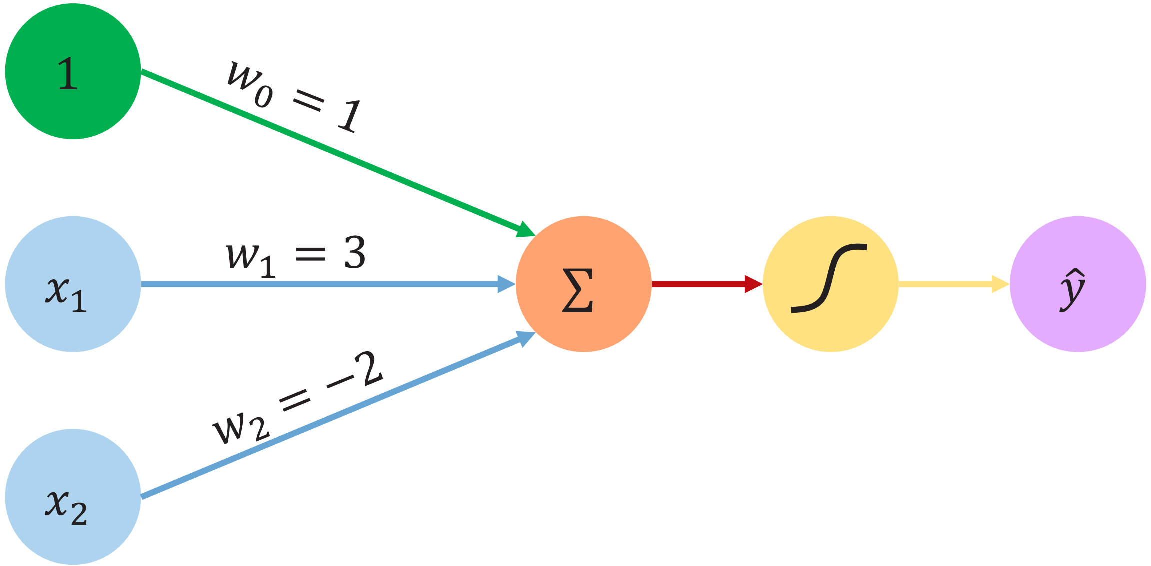
How to split red points with green ones with a neural network
Linear problem: How to split points with a straight line?

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning



The Perceptron example

The Perceptron: example

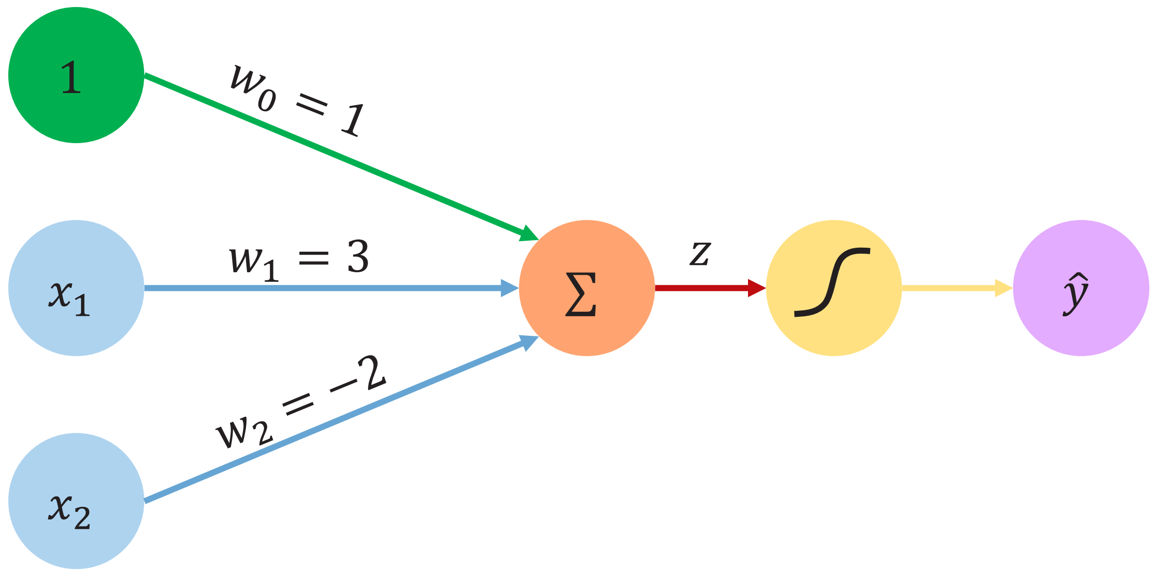


In this example, we have:

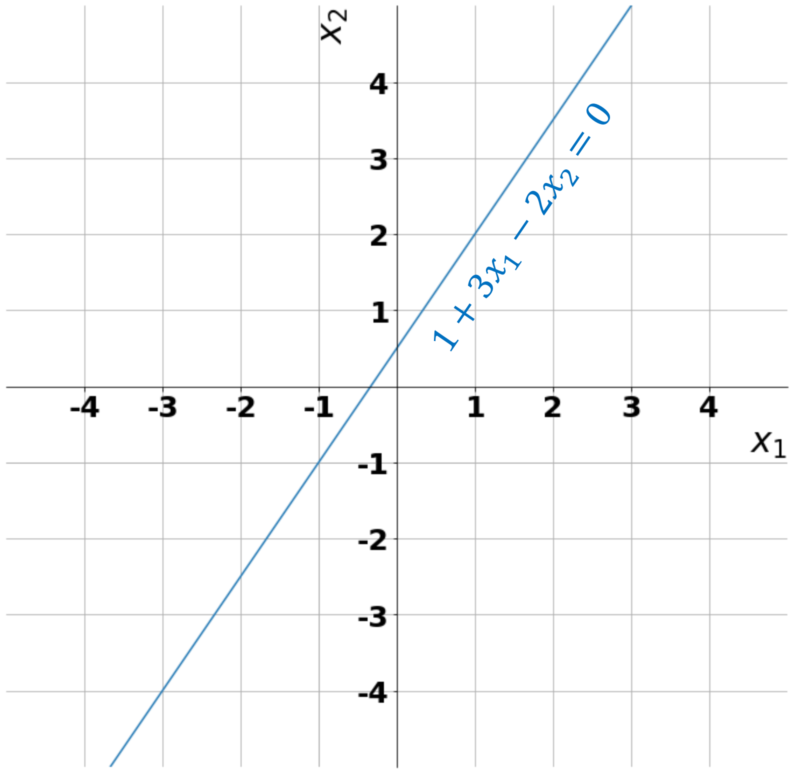
where $\boldsymbol{w_0} = \mathbf{1}$ and $\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w_1} = & \mathbf{3} \\ \boldsymbol{w_2} = & \mathbf{-2} \end{bmatrix}$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron: example



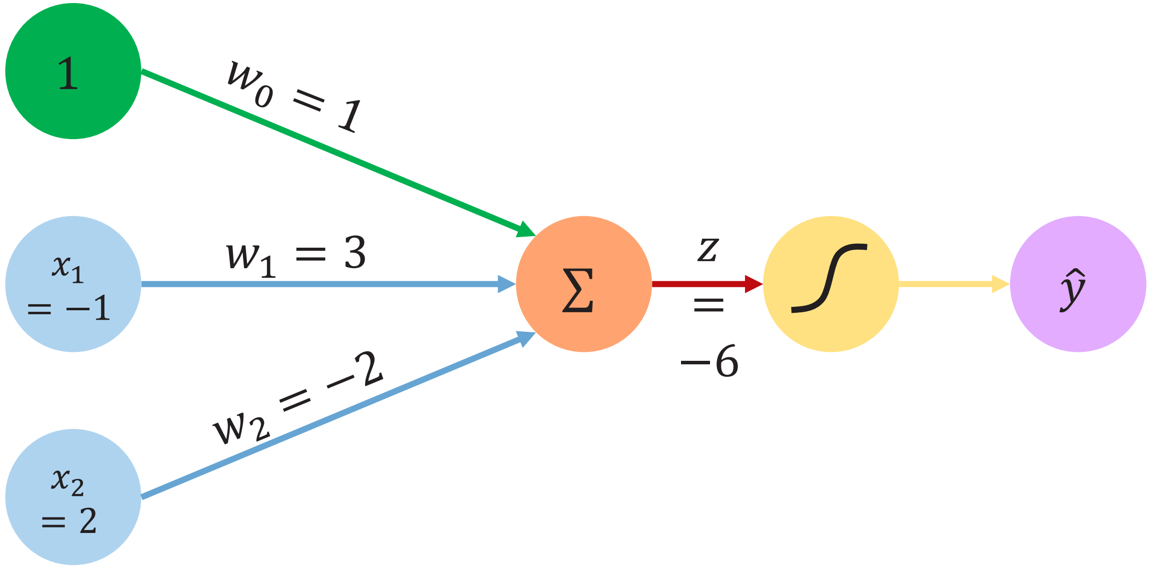
$$\hat{y} = g(z)$$
$$= g(1 + 3x_1 - 2x_2)$$

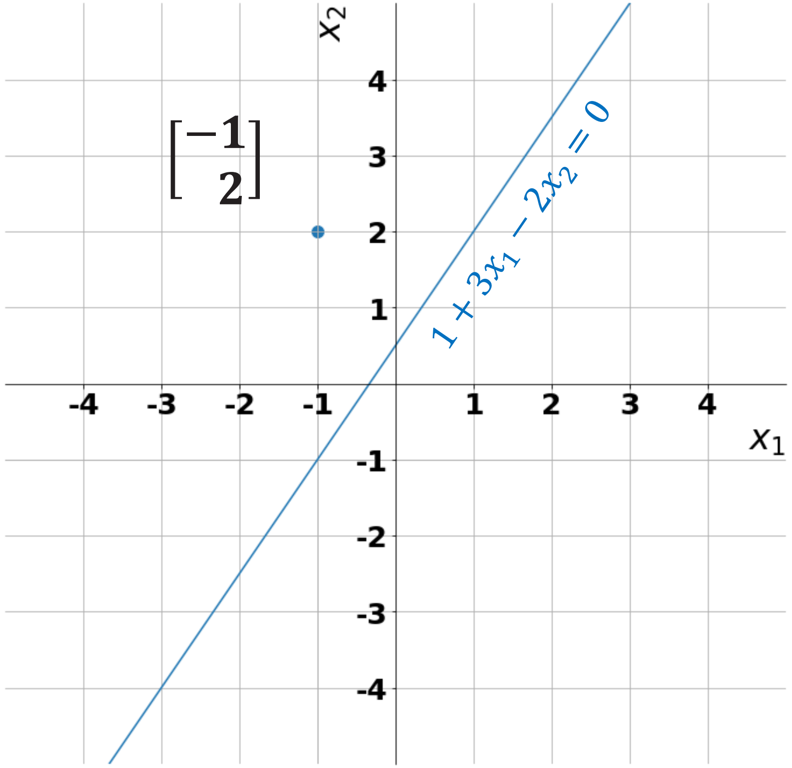


© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron: example

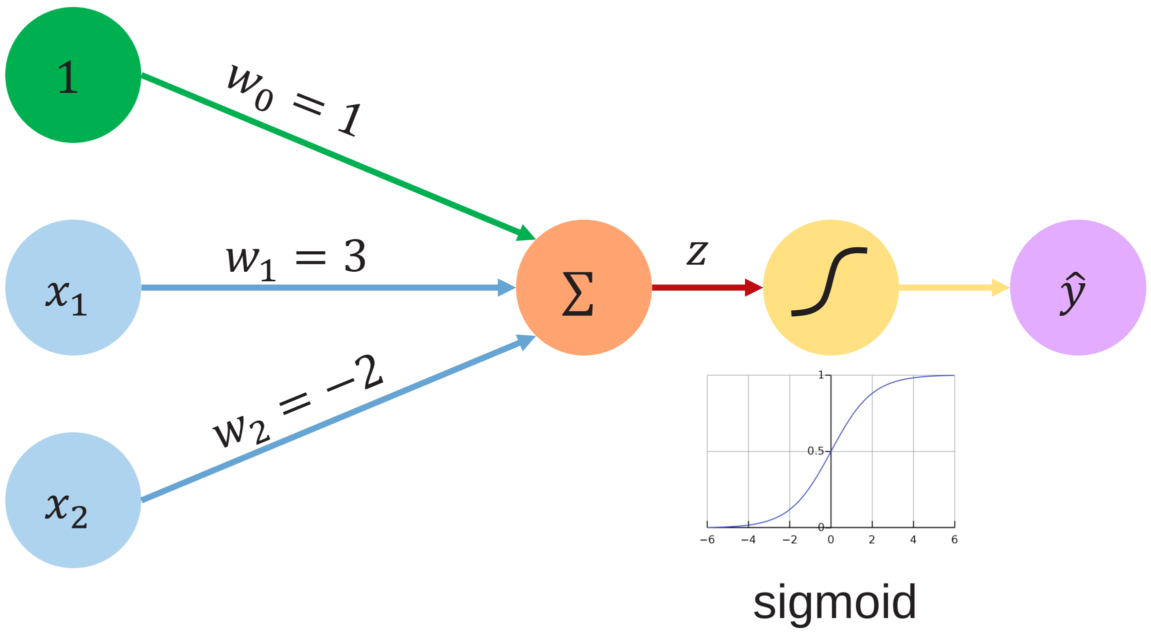
$$
\begin{aligned}
\hat{y} &= g(z) \\
&= g(1 + 3x_1 - 2x_2) \\
&= g(1 + 3 * (-1) - 2 * 2) \\
&= g(-6) \\
\hat{y} &\approx 0.002
\end{aligned}
$$



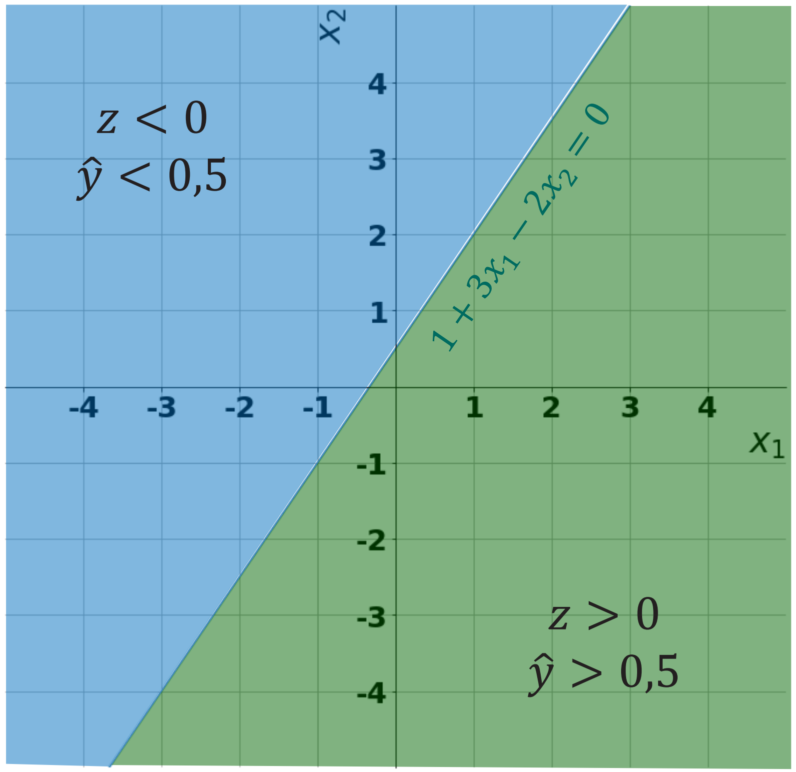


© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron: example



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

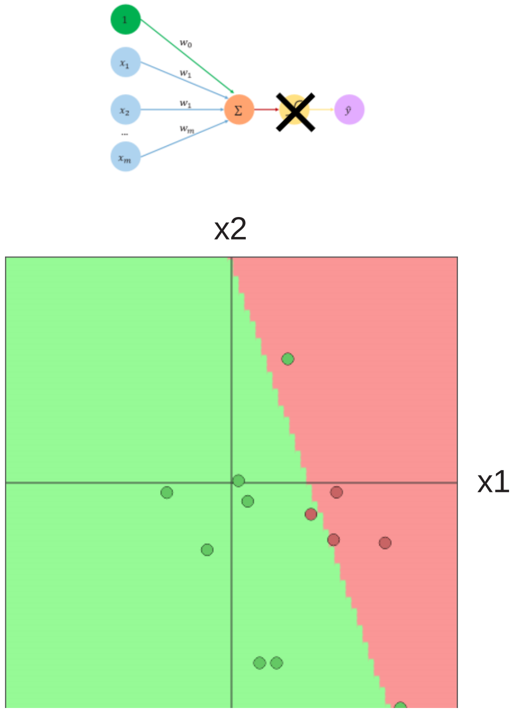


© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

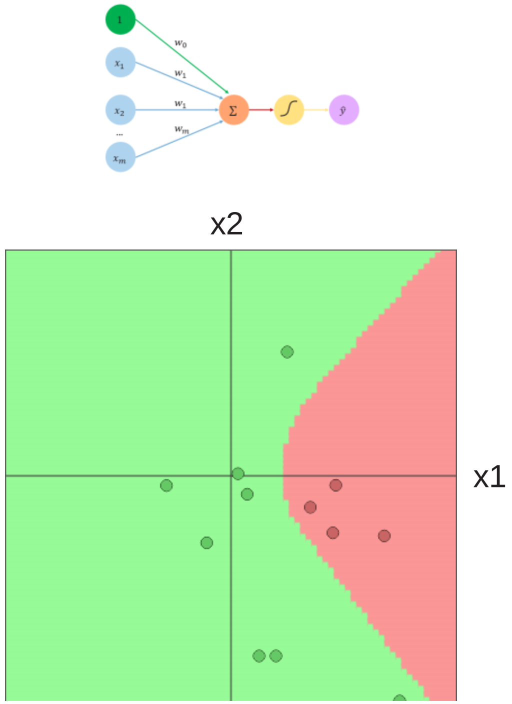
Importance of Activation Functions

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

Without Activation Function



With Activation Function



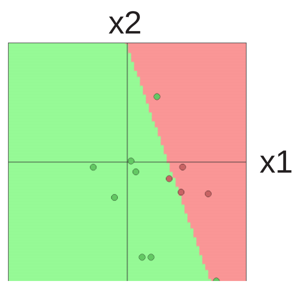
Demo from https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Importance of Activation Functions

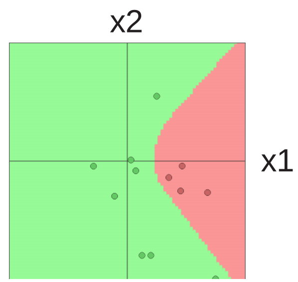
Simple data No activation function can solve linear problem

```
layer_defs = [];
layer_defs.push({type:'input', out_sx:1, out_sy:1, out_depth:2});
layer_defs.push({type:'fc', num_neurons:2});
layer_defs.push({type:'fc', num_neurons:2});
layer_defs.push({type:'softmax', num_classes:2});
net = new convnetjs.Net();
net.makeLayers(layer_defs);
trainer = new convnetjs.SGDTrainer(net, {learning_rate:0.01, momentum:0.1,
batch_size:10, l2_decay:0.001});
```



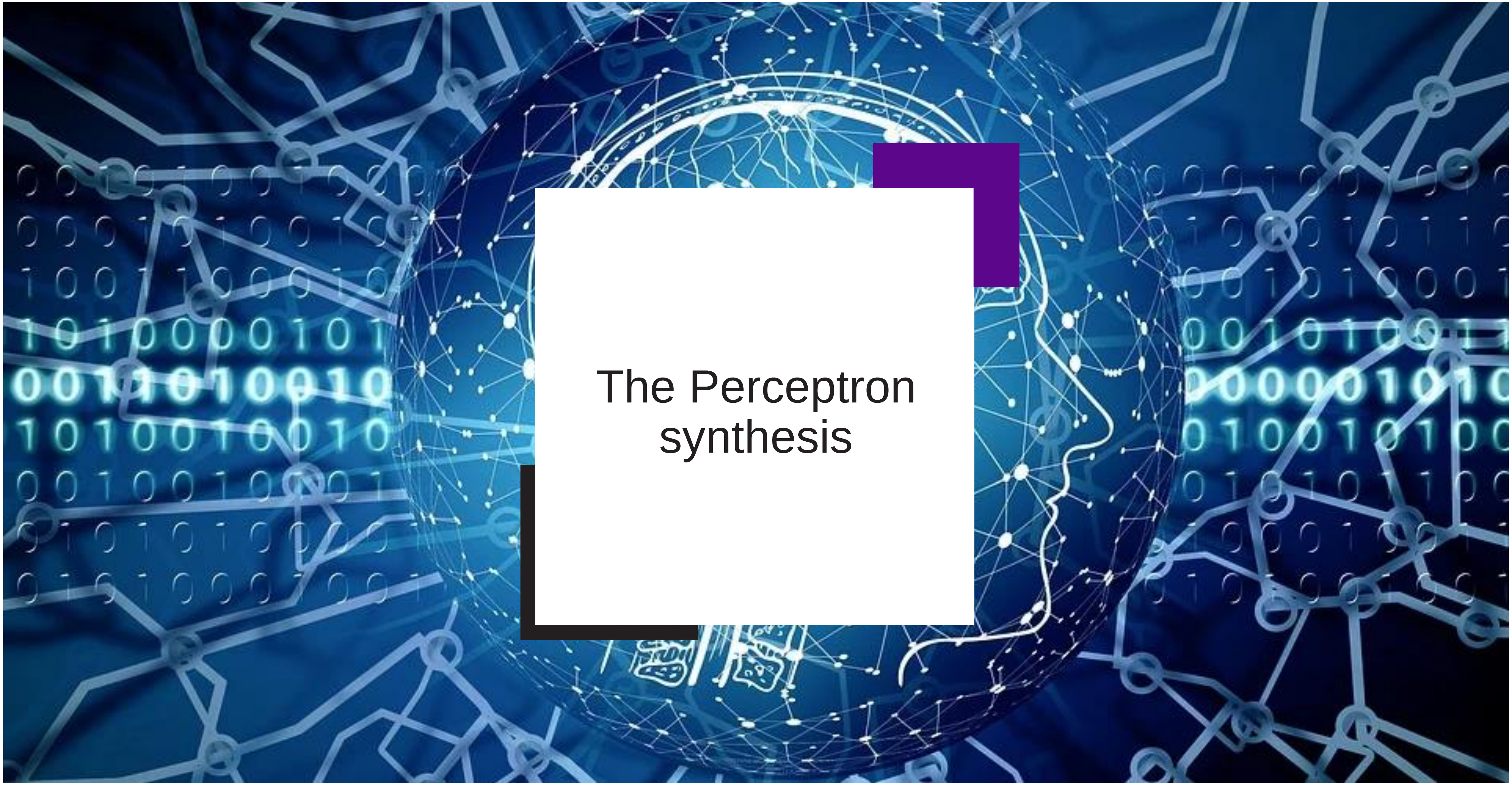
Simple data with sigmoid activation function can solve non linear problem

```
layer_defs = [];
layer_defs.push({type:'input', out_sx:1, out_sy:1, out_depth:2});
layer_defs.push({type:'fc', num_neurons:2, activation: 'sigmoid'});
layer_defs.push({type:'softmax', num_classes:2});
net = new convnetjs.Net();
net.makeLayers(layer_defs);
trainer = new convnetjs.SGDTrainer(net, {learning_rate:0.01, momentum:0.1,
batch_size:10, l2_decay:0.001});
```



Demo from https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

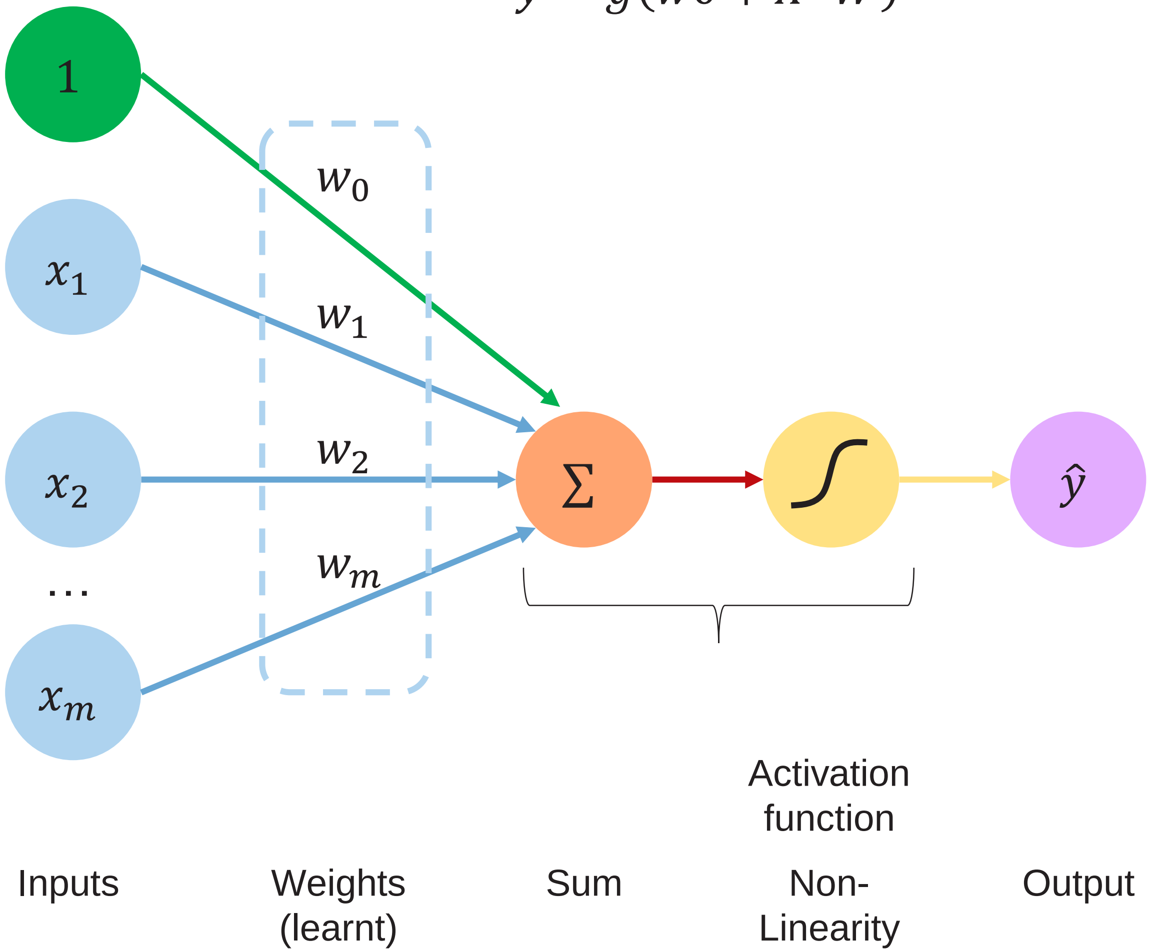
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning



The Perceptron synthesis

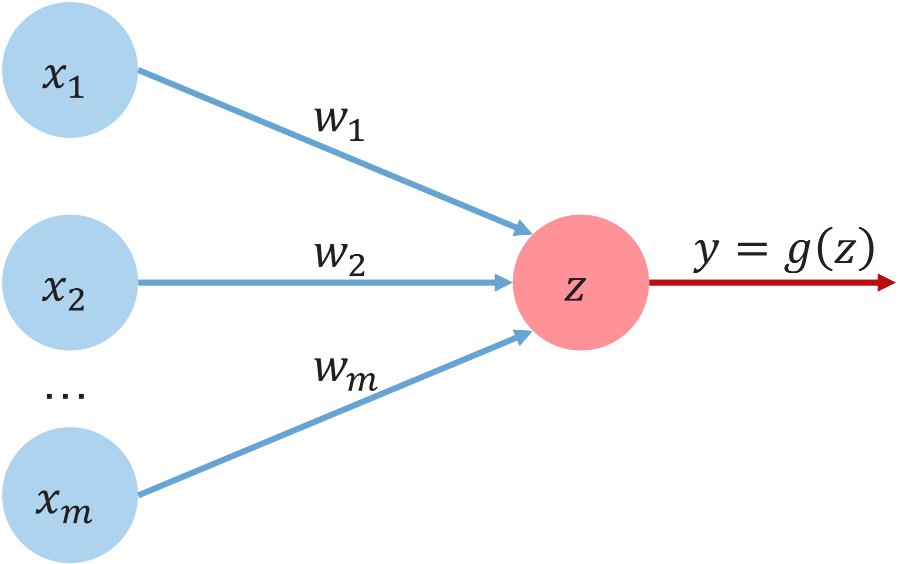
The Perceptron: detailed and…

$$\hat{y} = g(w0 + X^T W)$$



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

The Perceptron: …simplified representation

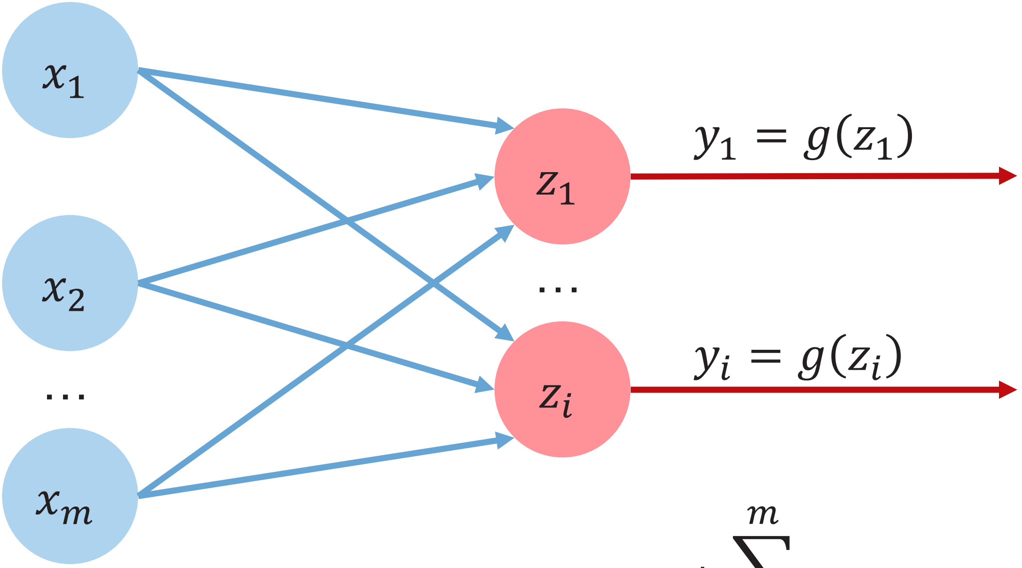


$$z = w0 + X^T W$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Multi Output Perceptron

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



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

$$g(z) = \sigma(z), \tanh(z) \text{ or } ReLU(z)$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

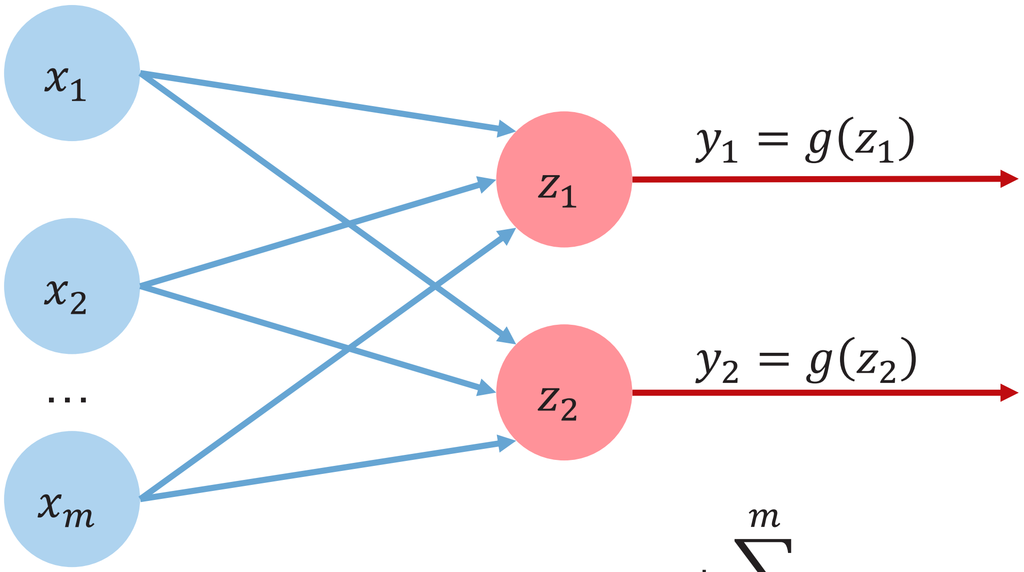
Dense Layer in Tensorflow

```python
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 function
        output = tf.math.sigmoid(z)

        return output
```


© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Multi Output Perceptron

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




```python
import tensorflow as tf
tf.keras.layers.Dense(
    units=2,
    activation='sigmoid')
```

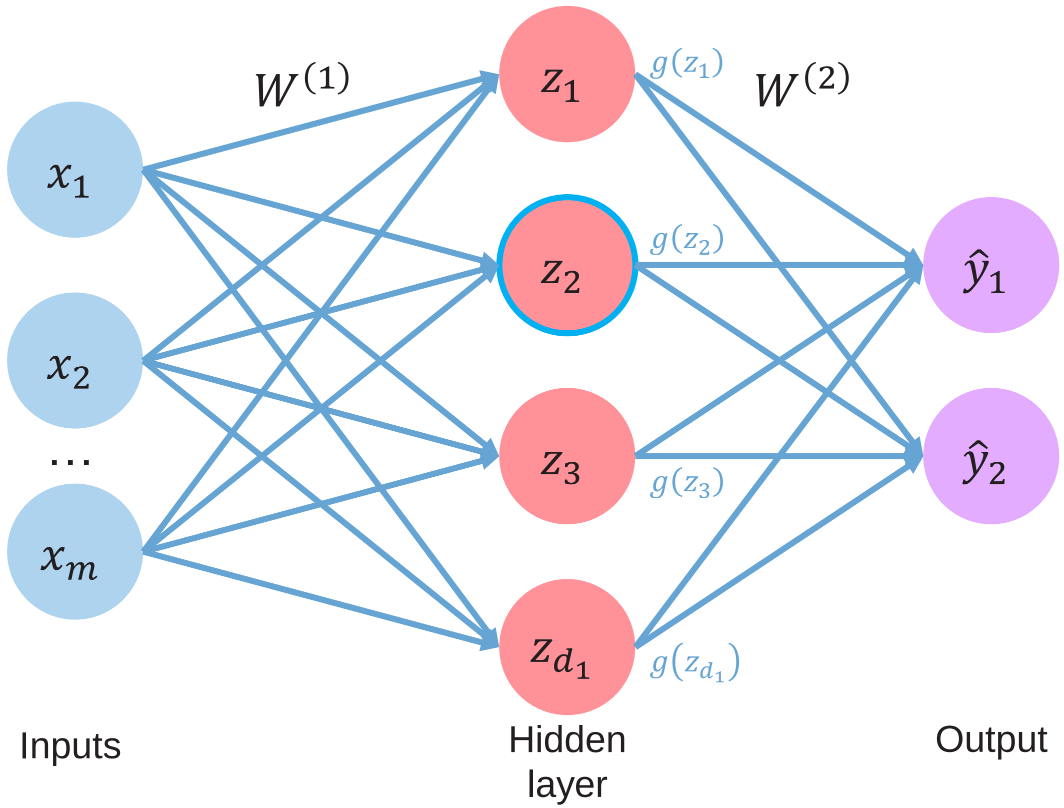
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$

$$g(z) = \sigma(z), \tanh(z) \text{ or } ReLU(z)$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Going Deep

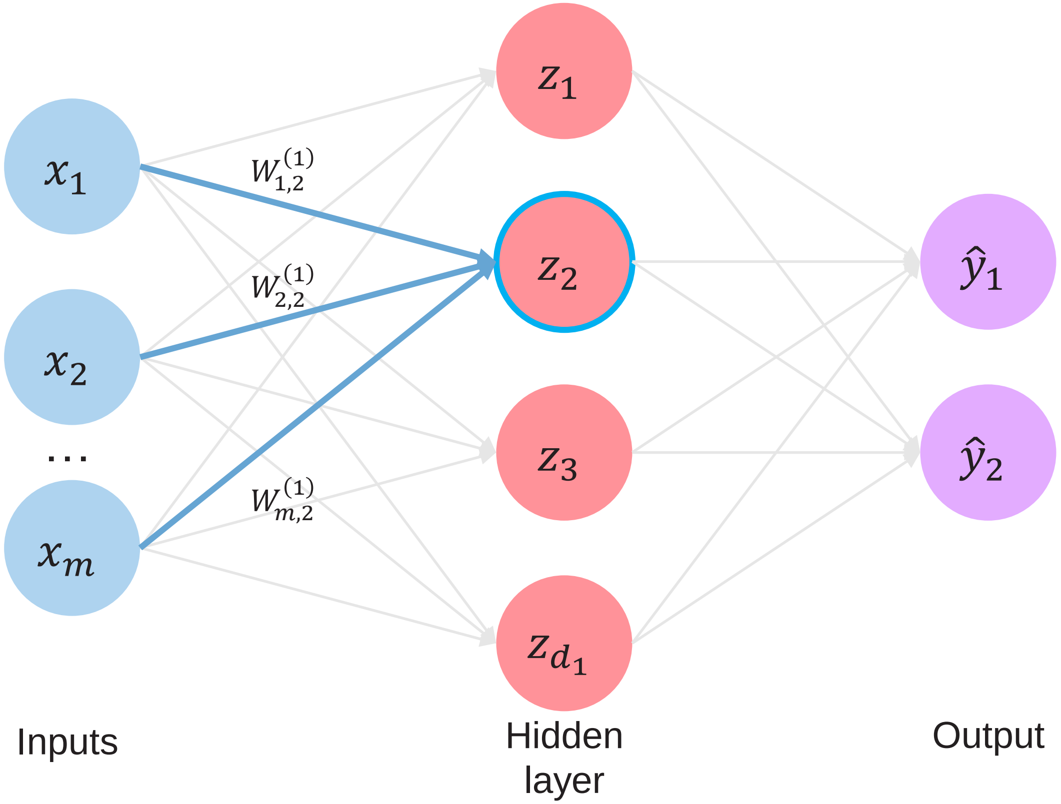
Single Hidden Layer Neural Network



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

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

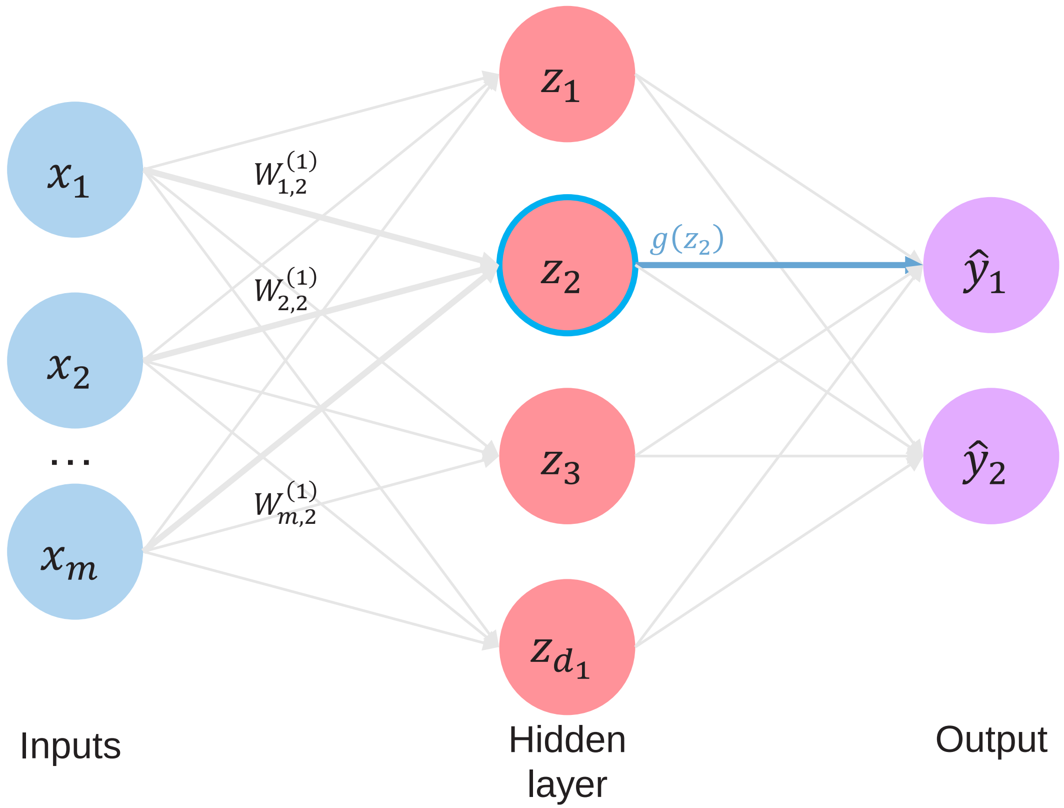
Single Hidden Layer Neural Network



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j w_{j,2}^{(1)} = w_{0,2}^{(1)} + x_1 * w_{1,2}^{(1)} + x_2 * w_{2,2}^{(1)} + \cdots + x_m * w_{m,2}^{(1)}$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

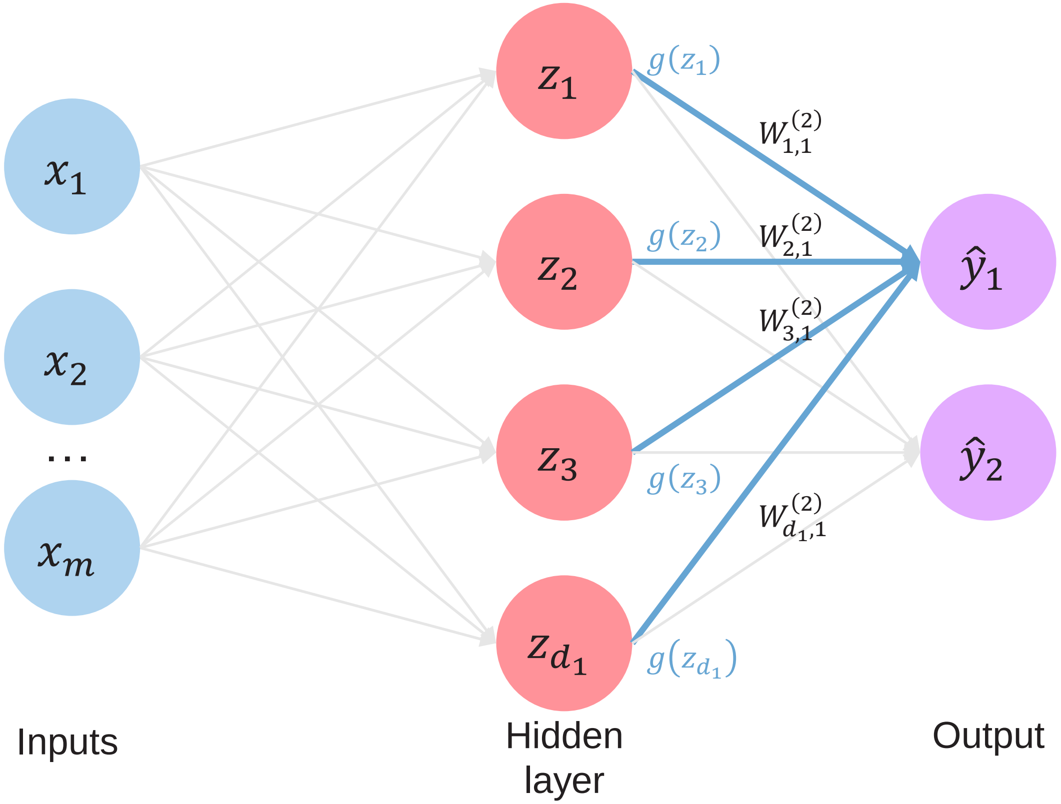
Single Hidden Layer Neural Network



$$g(z_2) = g\left(w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j w_{j,2}^{(1)} = w_{0,2}^{(1)} + x_1 * w_{1,2}^{(1)} + x_2 * w_{2,2}^{(1)} + \cdots + x_m * w_{m,2}^{(1)}\right)$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

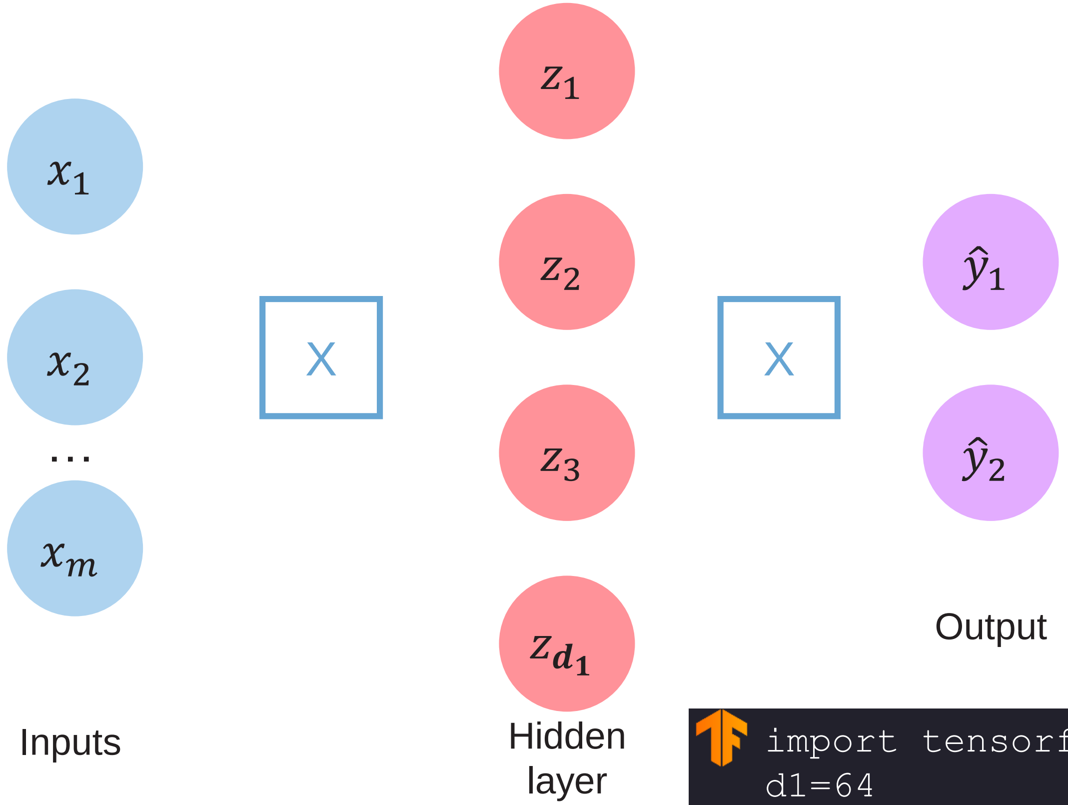
Single Hidden Layer Neural Network



$$\hat{y}_1 = g\left(w_{0,k}^{(2)} + \sum_{j=1}^{d_1} g(z_j) w_{j,k}^{(2)}\right) = g\left(w_{0,2}^{(2)} + w_{1,2}^{(2)} * g(z_1) + w_{2,2}^{(2)} * g(z_2) + w_{3,2}^{(2)} * g(z_3) + w_{d_1,2}^{(2)} * g(z_{d_1})\right)$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

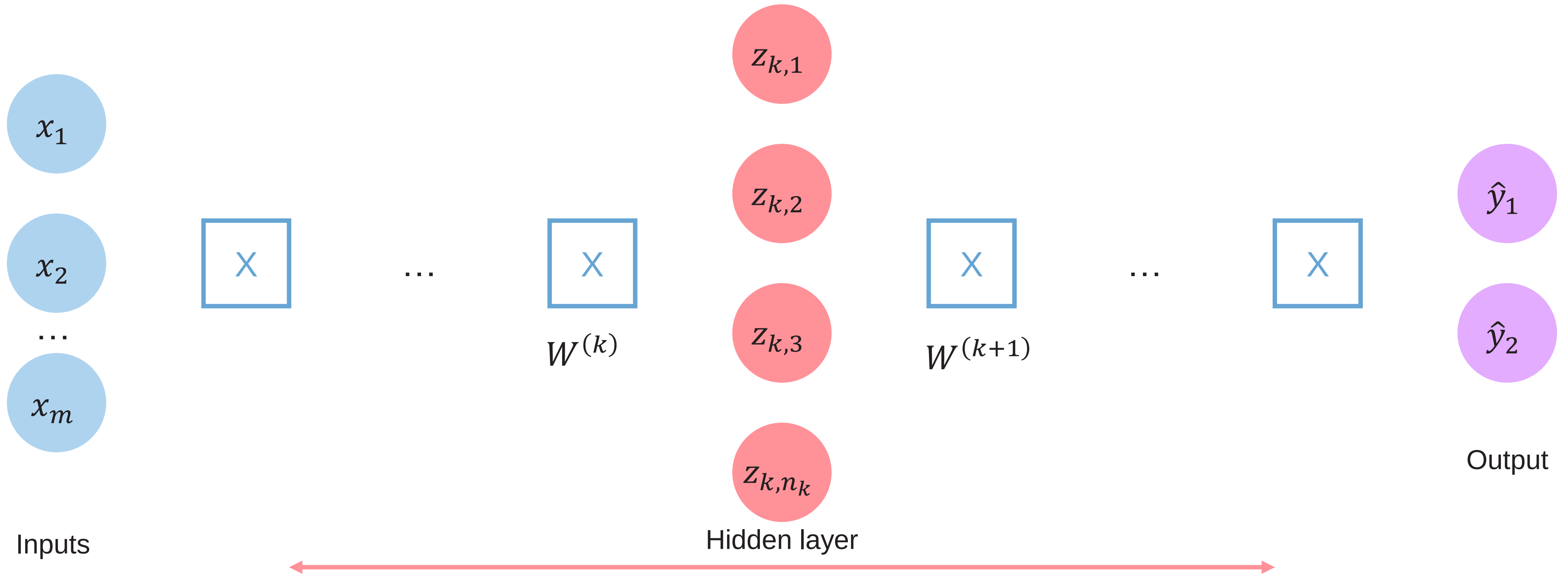
Single Hidden Layer Neural Network




```
import tensorflow as tf
d1=64
model = tf.keras.Sequential([
  tf.layers.Dense(units=d1, activation='sigmoid'),
  tf.layers.Dense(units=2, activation='sigmoid')
])
```

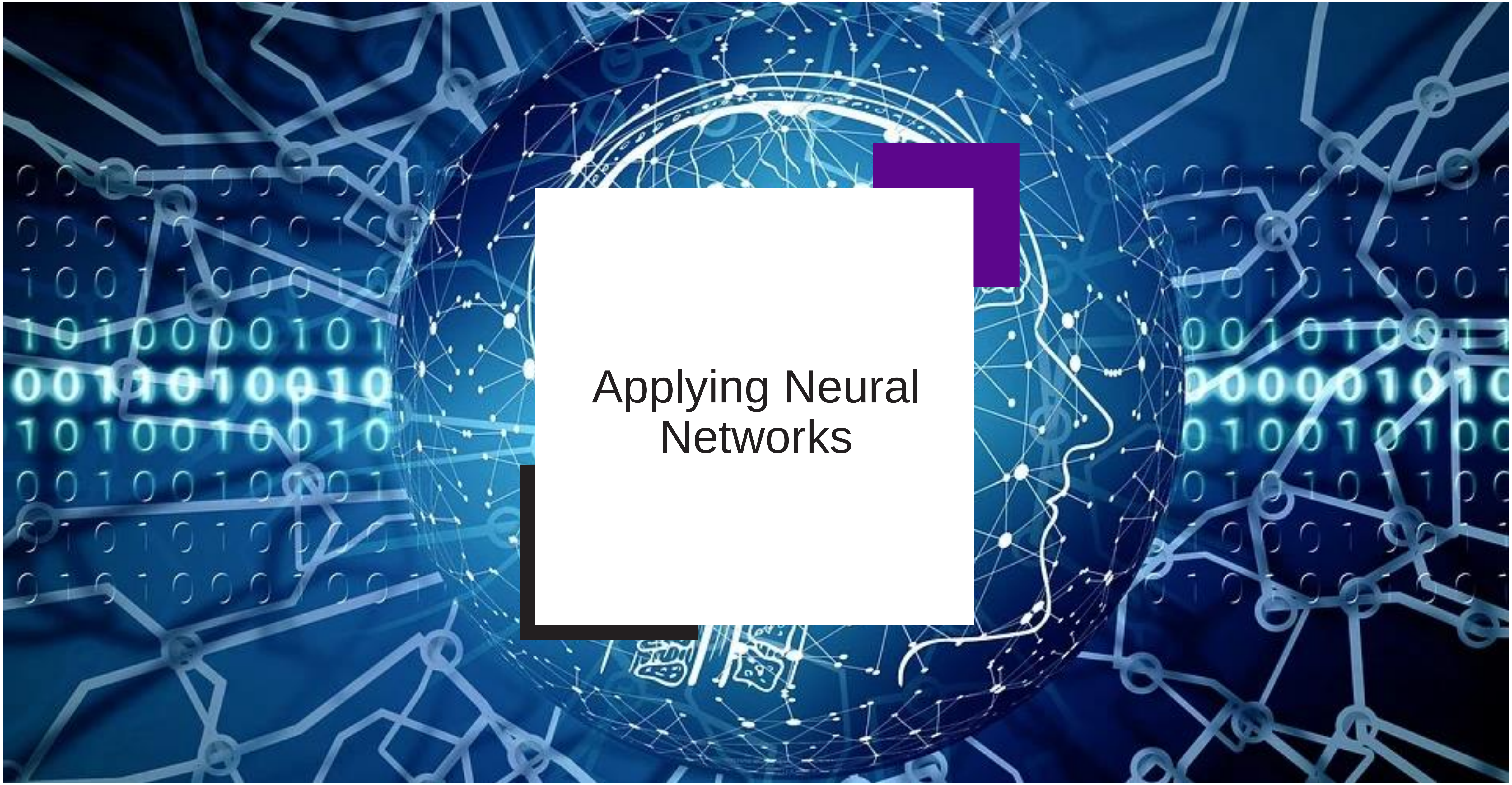

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Deep Neural Network



$$z_i^{(k)} = w_{0,i}^{(k)} + \sum_{j=1}^{d_k} g(z_j^{(k-1)}) w_{j,i}^{(k)}$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning



Applying Neural Networks

Example problems

Will I pass this class? (yes, no)

- x_1=Number of lectures you attend
- x_2=Hours spent on the final project

Should I go to a kitesurfing session? (yes, no)

- x_1=Kite dimension
- x_2=Wind Force

Should I buy this car? (yes, no)

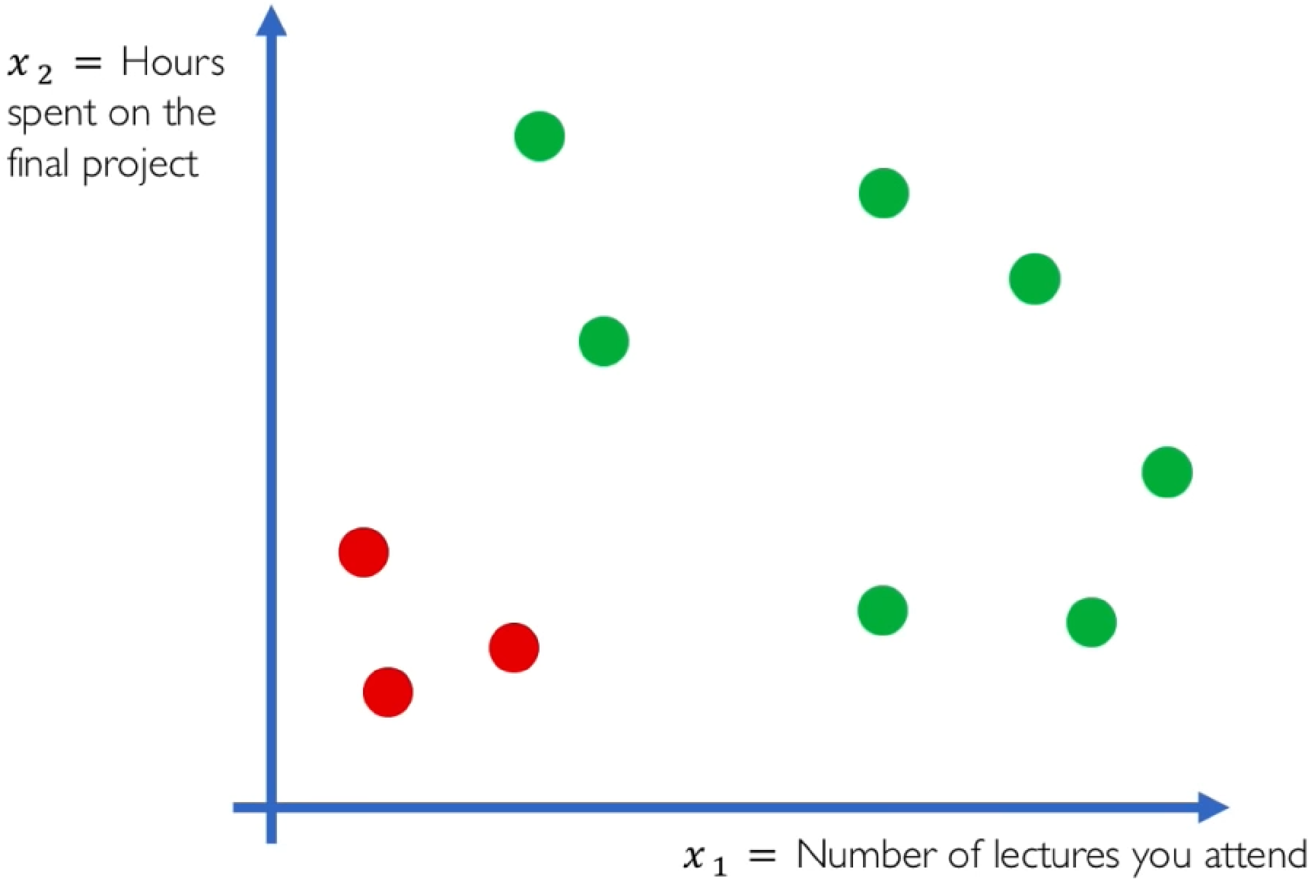
- x_1= number of options
- x_2= ecological ranking

Definitions

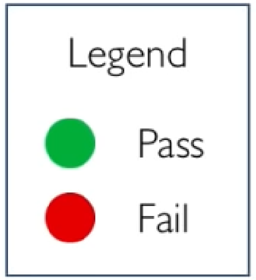
- $\hat{\boldsymbol{y}}$ **Network prediction**
- $\boldsymbol{y}$ **Ground truth**

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Example problem: Will I pass this class?

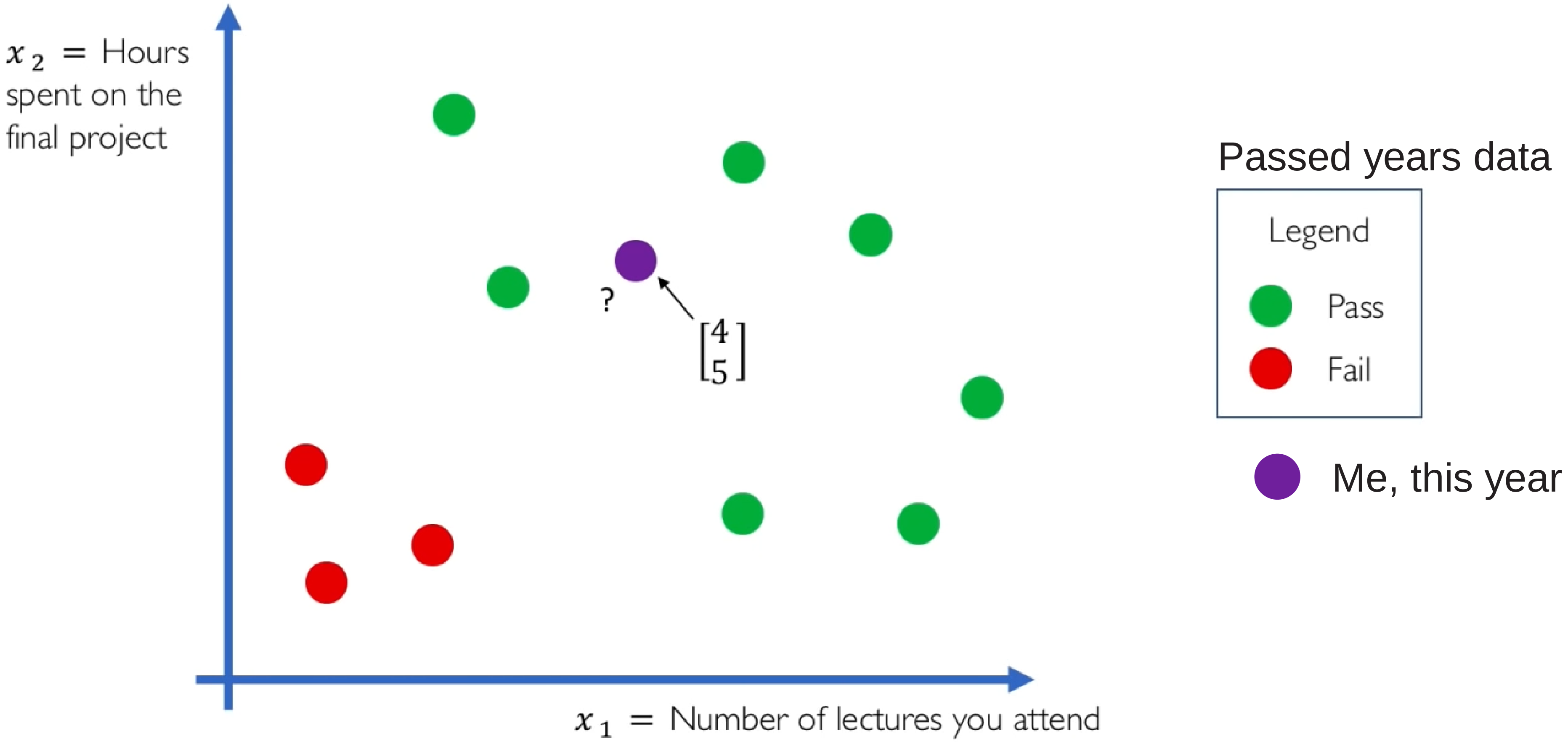


Passed years data



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

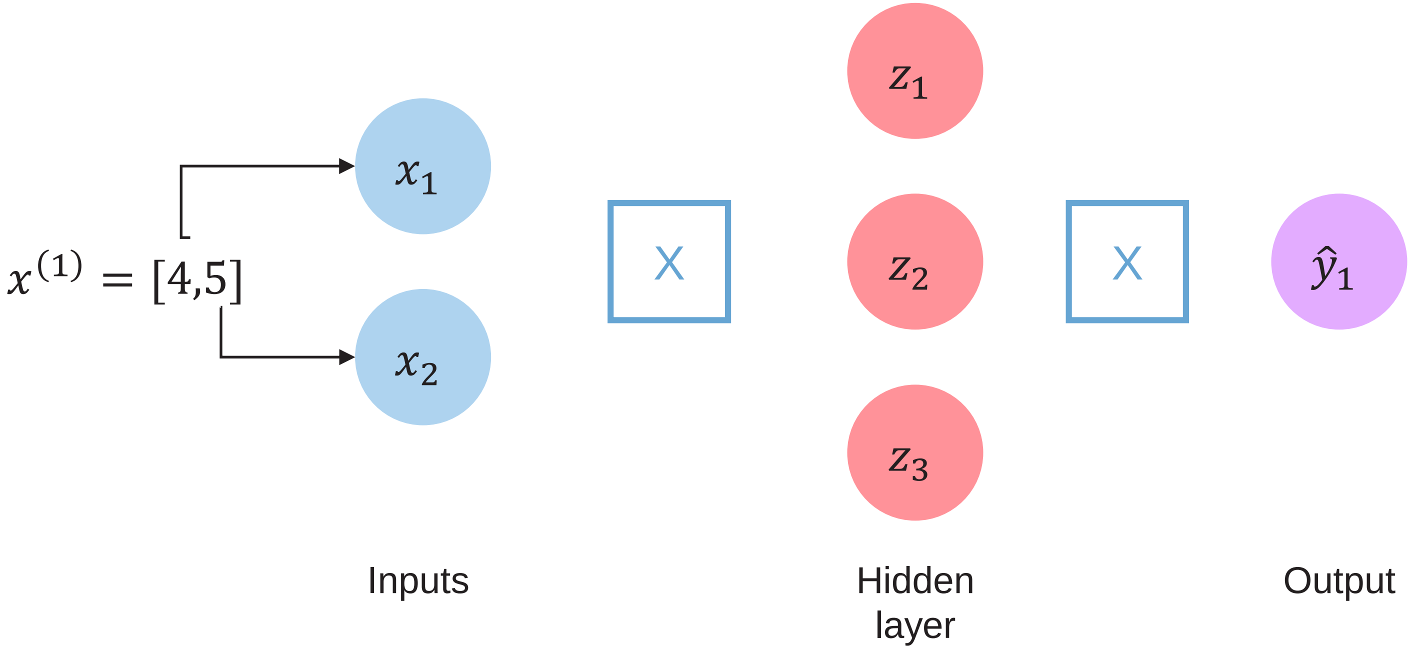
Example problem: Will I pass this class?



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Example problem: Will I pass this class?

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



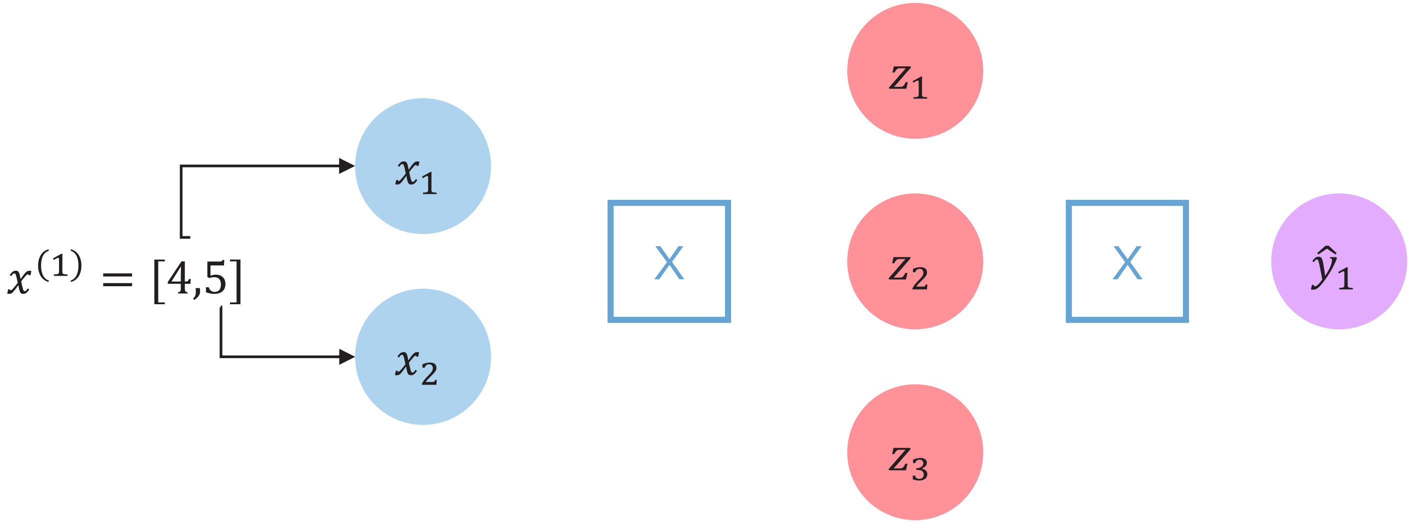
Predicted: **0.1**

Actual or Ground Truth: **1**

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Quantifying Loss

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



Predicted: **0.1**

Actual or Ground Truth: **1**

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

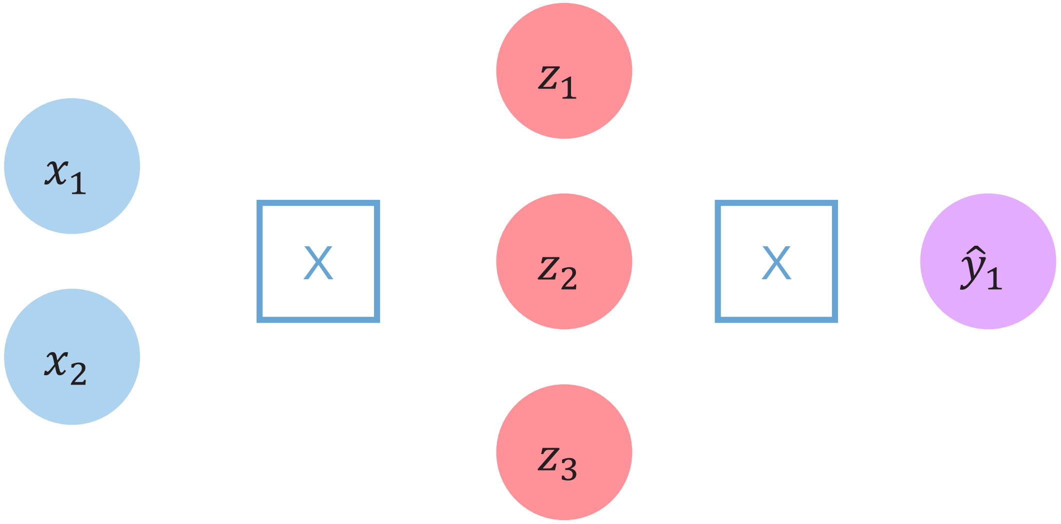
$$\mathcal{L}\left(\underbrace{\hat{y}^{(i)},}_{\text{Predicted}} \underbrace{y^{(i)}}_{\text{Actual}}\right)$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Empirical Loss

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

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



$$\begin{array}{ccc} f(x) & & \\ \hat{y} & & y \\ \begin{bmatrix} 0.9 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{array}{c} \checkmark \\ \times \\ \checkmark \\ \end{array} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

Called:

- **Objective function**
- **Cost function**
- Empirical risk

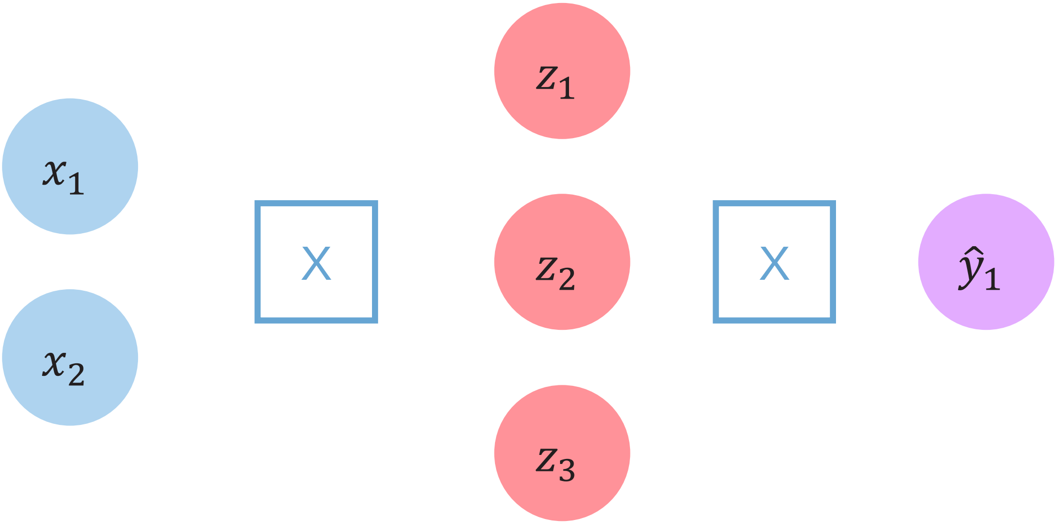
$$\boldsymbol{J(W)} = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\big(\underbrace{f\big(x^{(i)}, \boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\big)$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Binary Cross Entropy Loss

Cross entropy loss for model that output a probability between 0 and 1

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

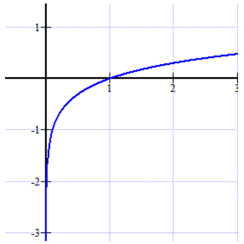


$$\begin{array}{cc} f(x) & \\ \hat{y} & y \\ \begin{bmatrix} 0.9 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \begin{array}{c} ✓ \\ ✗ \\ ✓ \\ \end{array} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

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

Reminder:

$$\log(x) = \begin{cases} < 0, si\ x < 1 \\ 0, si\ x = 1 \\ > 0, si\ x > 1 \end{cases}$$

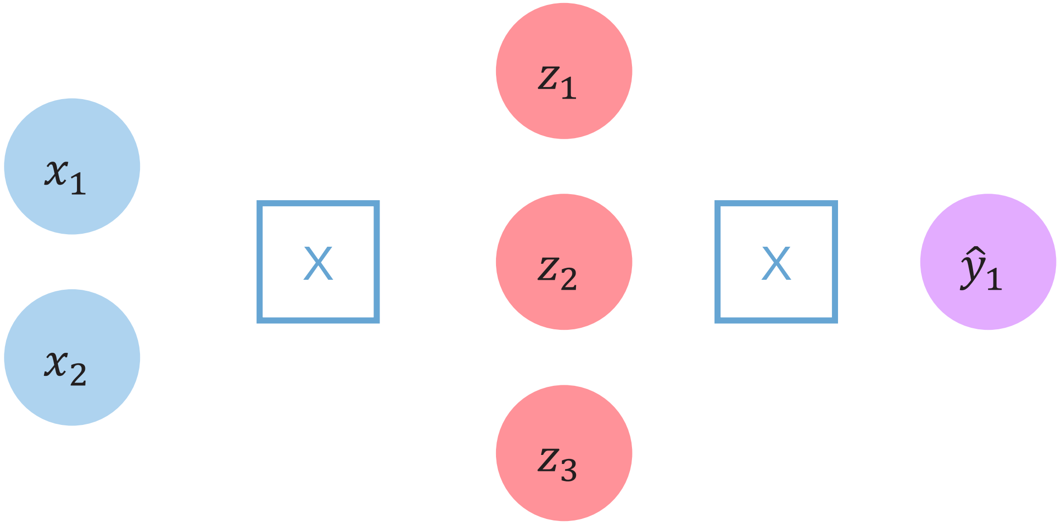


© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Binary Cross Entropy Loss

Cross entropy loss for model that output a probability between 0 and 1

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



$$\begin{array}{ccc} f(x) & & \\ \hat{y} & & y \\ \begin{bmatrix} 0.9 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{matrix} ✓ \\ ✗ \\ ✓ \\ \ \end{matrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

$\boldsymbol{J}(\boldsymbol{W})$

$y \cdot \log(\hat{y})$

or

$(1 - y) \cdot \log(1 - \hat{y})$

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

Reminder :

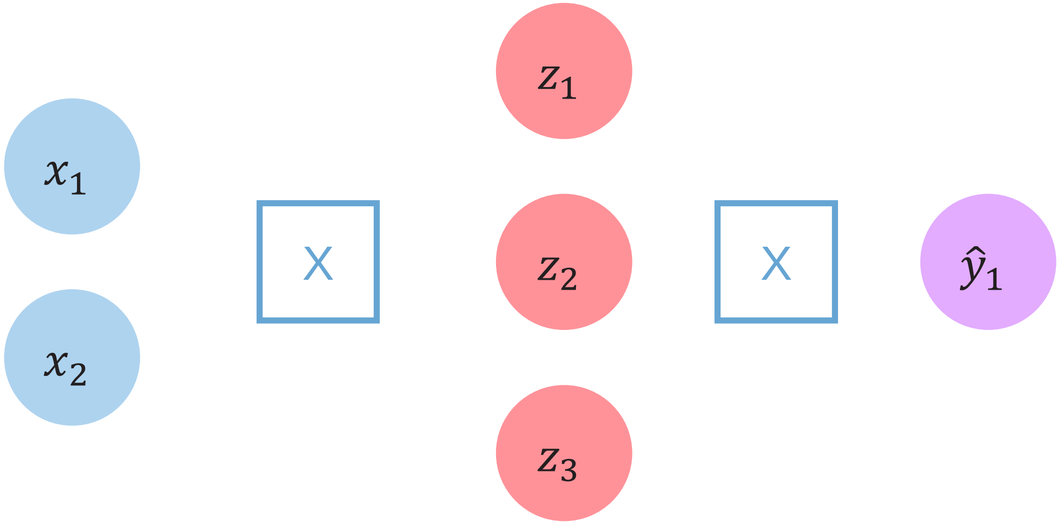
$$\log(x) = \begin{cases} < 0, si\ x < 1 \\ 0, si\ x = 1 \\ \cancel{> 0, si\ x > 1} \end{cases}$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Binary Cross Entropy Loss

Cross entropy loss for model that output a probability between 0 and 1

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



$$\begin{array}{ccc} f(x) & & \\ \hat{y} & & y \\ \begin{bmatrix} 0.9 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{matrix} \checkmark \\ \times \\ \checkmark \\ \end{matrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

$$\begin{array}{c} \boldsymbol{J}(\boldsymbol{W}) \\ y \cdot \log(\hat{y}) \\ \text{or} \\ (1-y) \cdot \log(1-\hat{y}) \\ \begin{bmatrix} -1 \cdot \log(0.9) - (1-1) \cdot \log(1-0.9) = 0.046 \\ 0 - (1-0) \cdot \log(1-0.8) = 0.70 \\ -1 \cdot \log(0.6) - (1-1) \cdot \log(1-0.6) = 0.22 \\ \vdots \end{bmatrix} \end{array}$$

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

Remember:

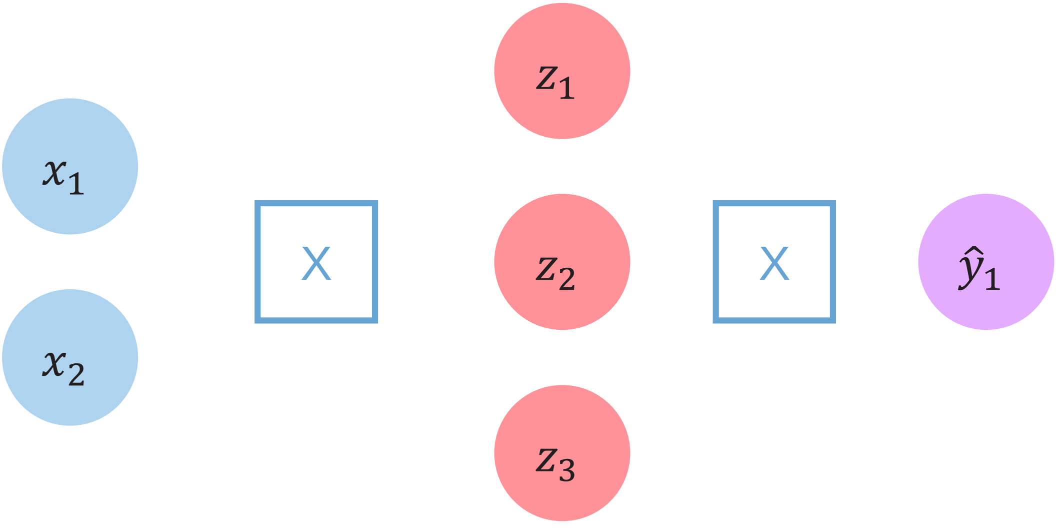
$$\log(x) = \begin{cases} < 0, si\ x < 1 \\ 0, si\ x = 1 \\ \cancel{> 0, si\ x > 1} \end{cases}$$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Mean Squared Error (MSE) Loss

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

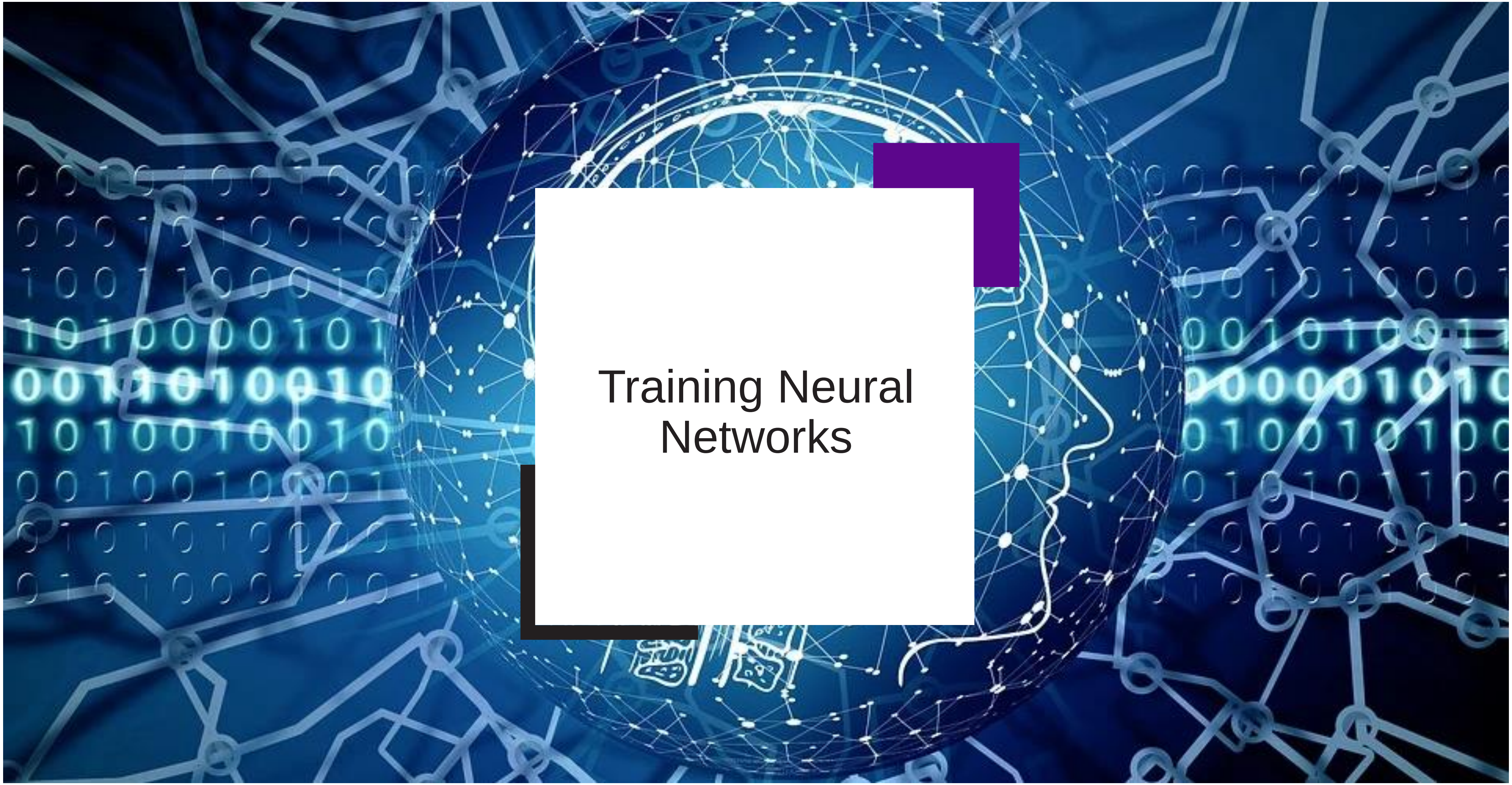
$$X = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



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

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

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning



Training Neural Networks

Loss Optimization

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

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

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

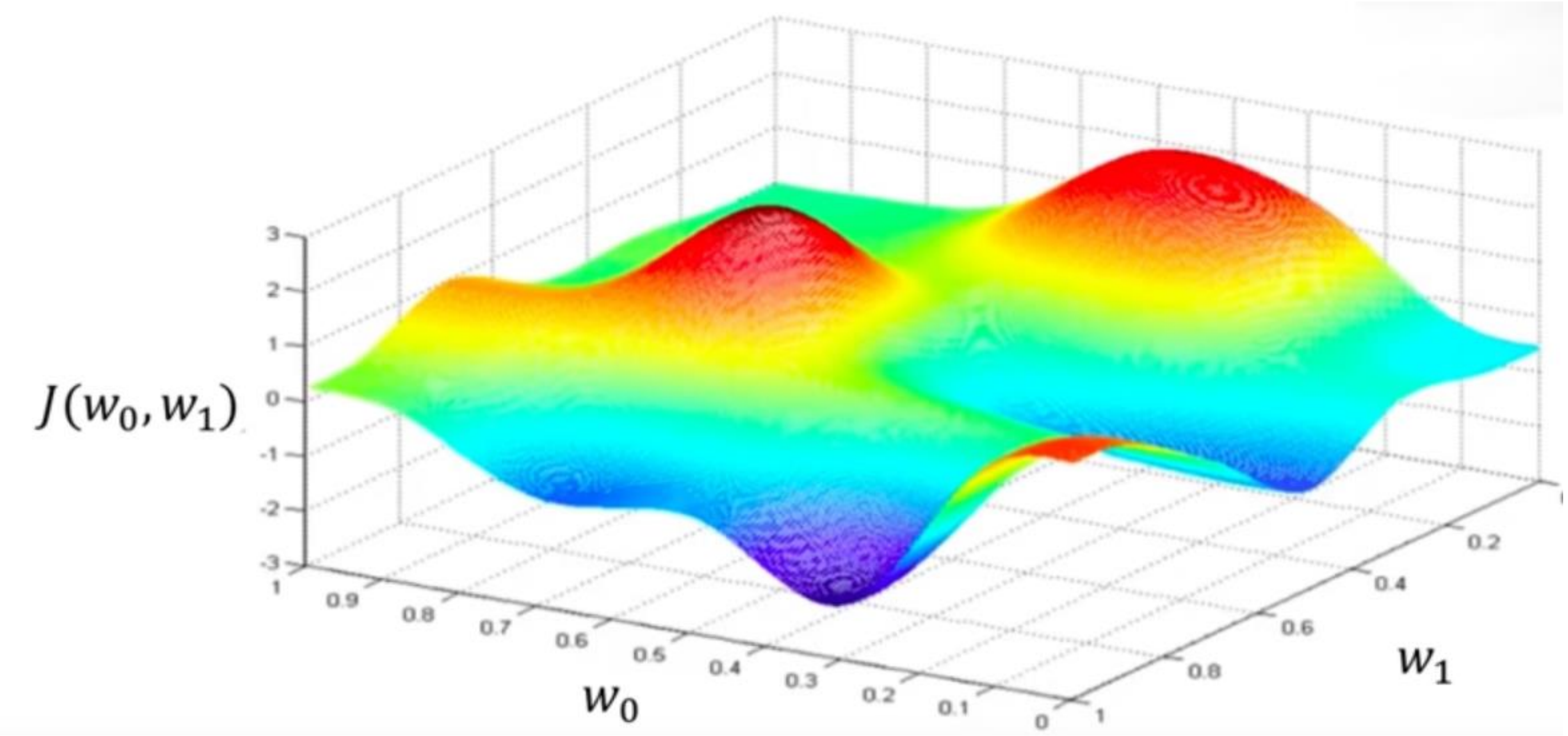
Remember that:

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

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Loss Optimization

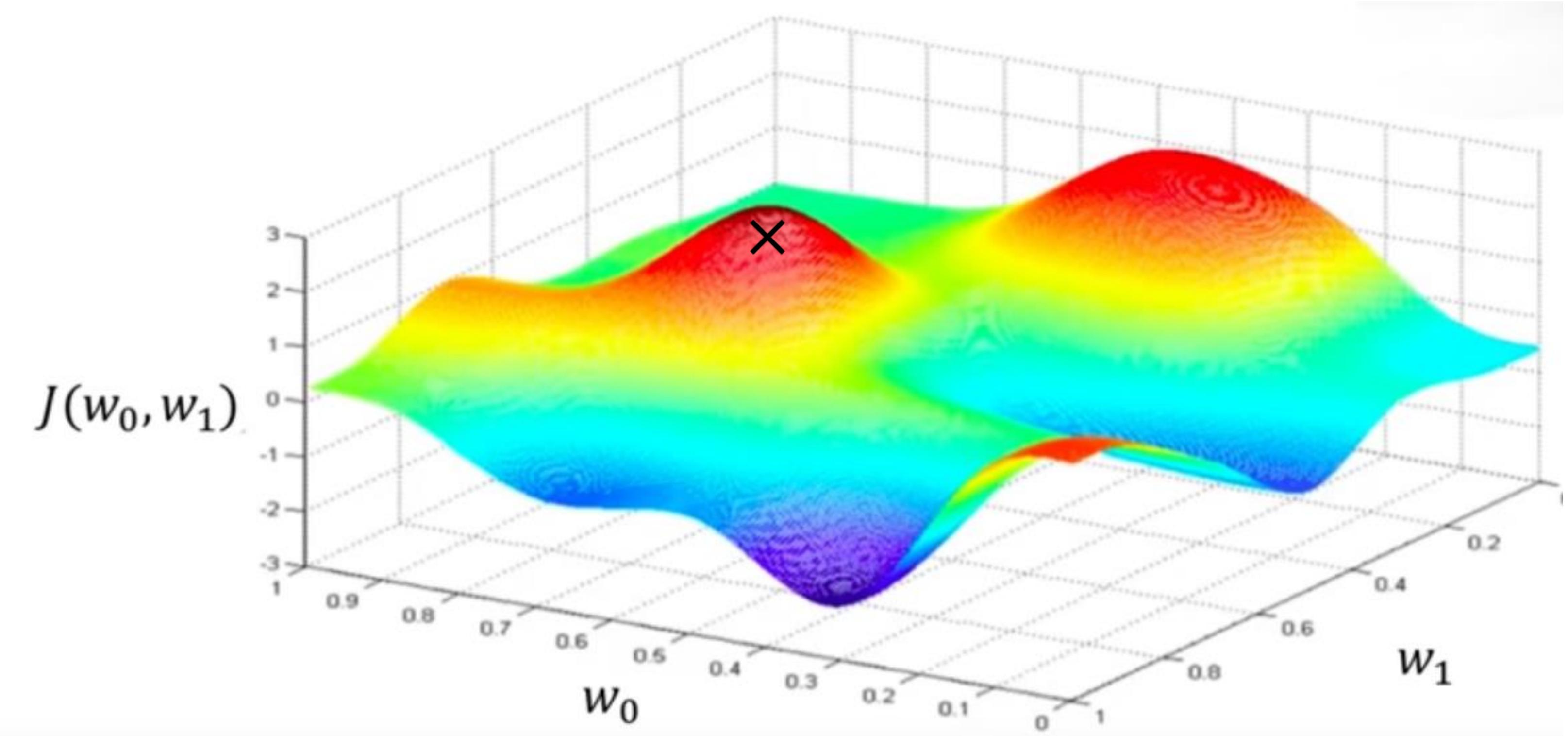
$$\boldsymbol{W}^* = \underset{W}{\operatorname{argmin}}\, \boldsymbol{J}(\boldsymbol{W})$$



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Loss Optimization

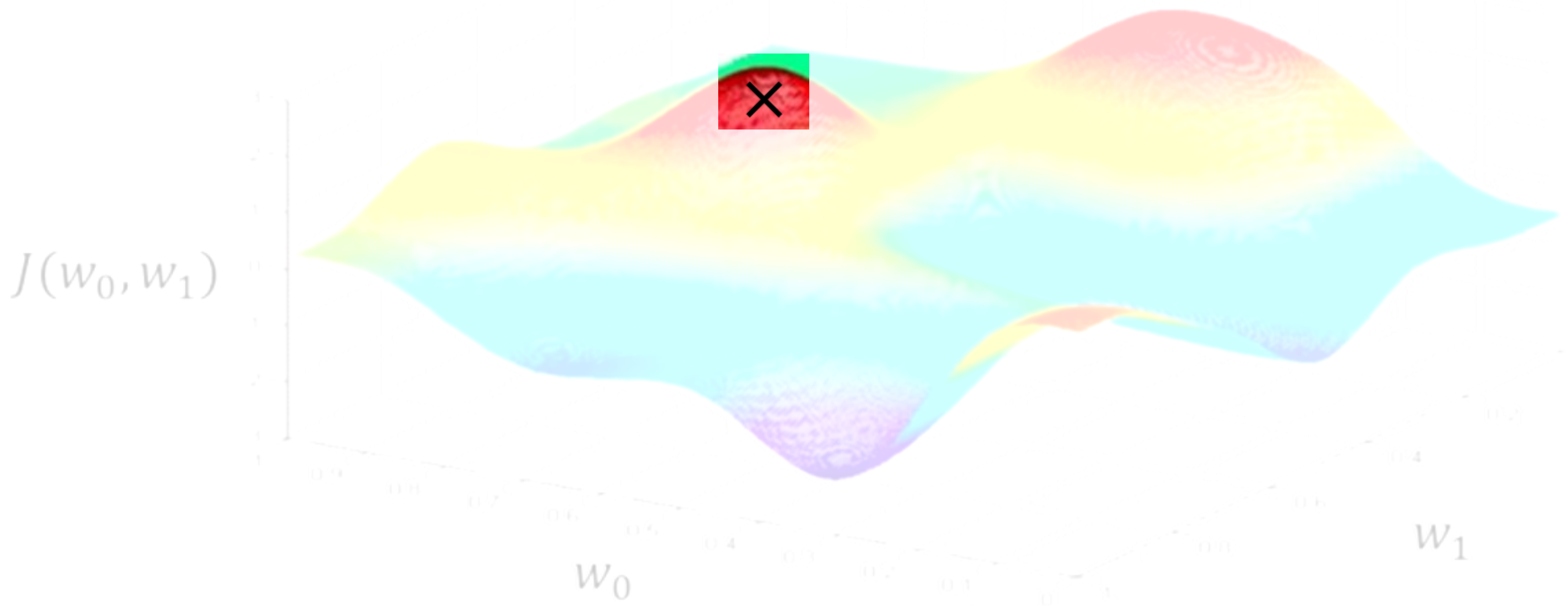
Randomly pick an initial value of (w_0, w_1)



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Loss Optimization

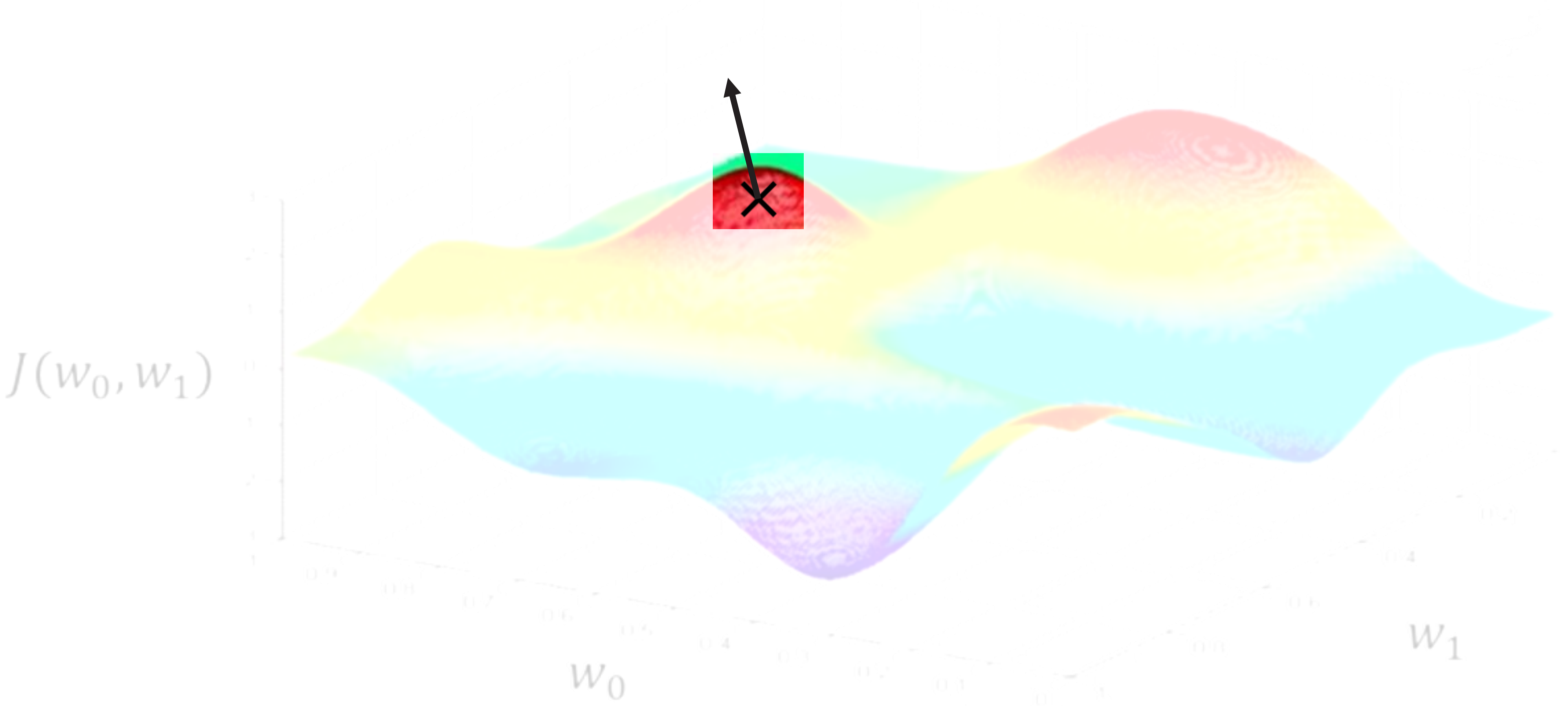
Randomly pick an initial value of (w_0, w_1)



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Loss Optimization

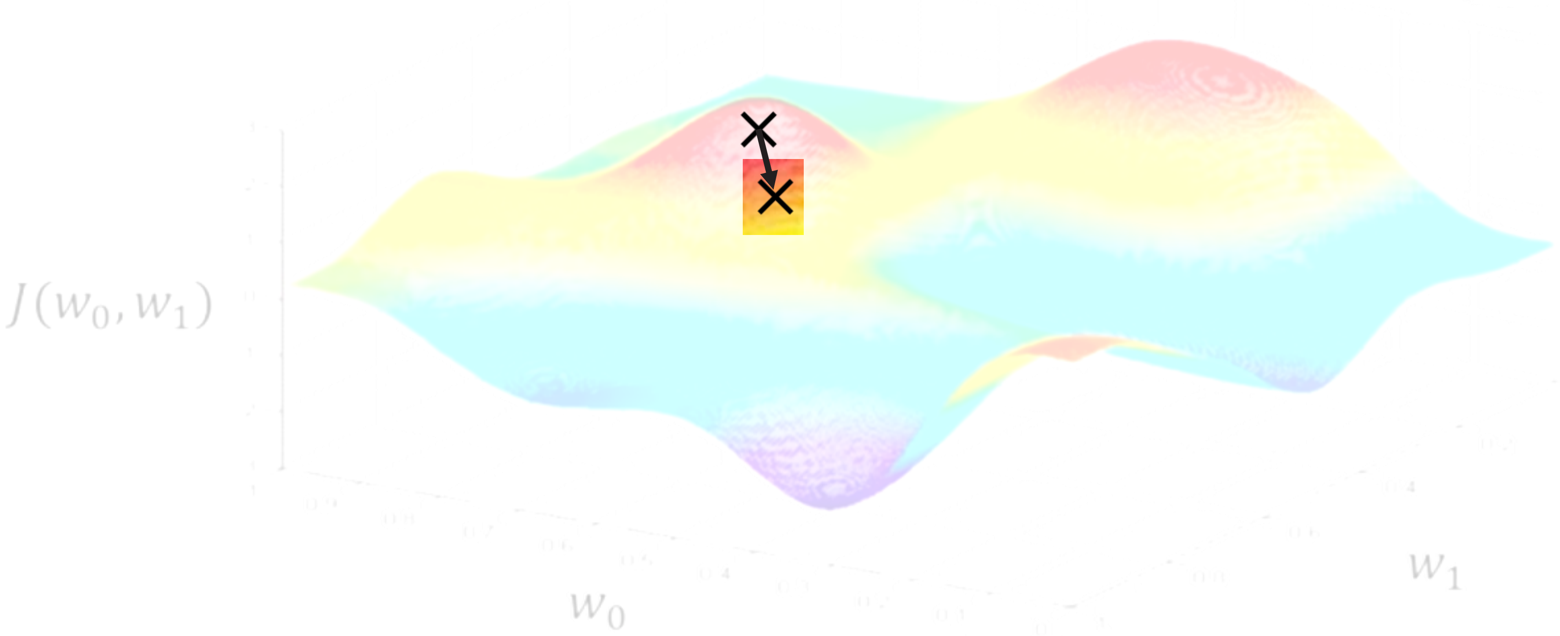
Compute the gradient $\frac{\partial J(W)}{\partial W}$



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning
ÉCOLE D'INGÉNIEURS LINEACT

Loss Optimization

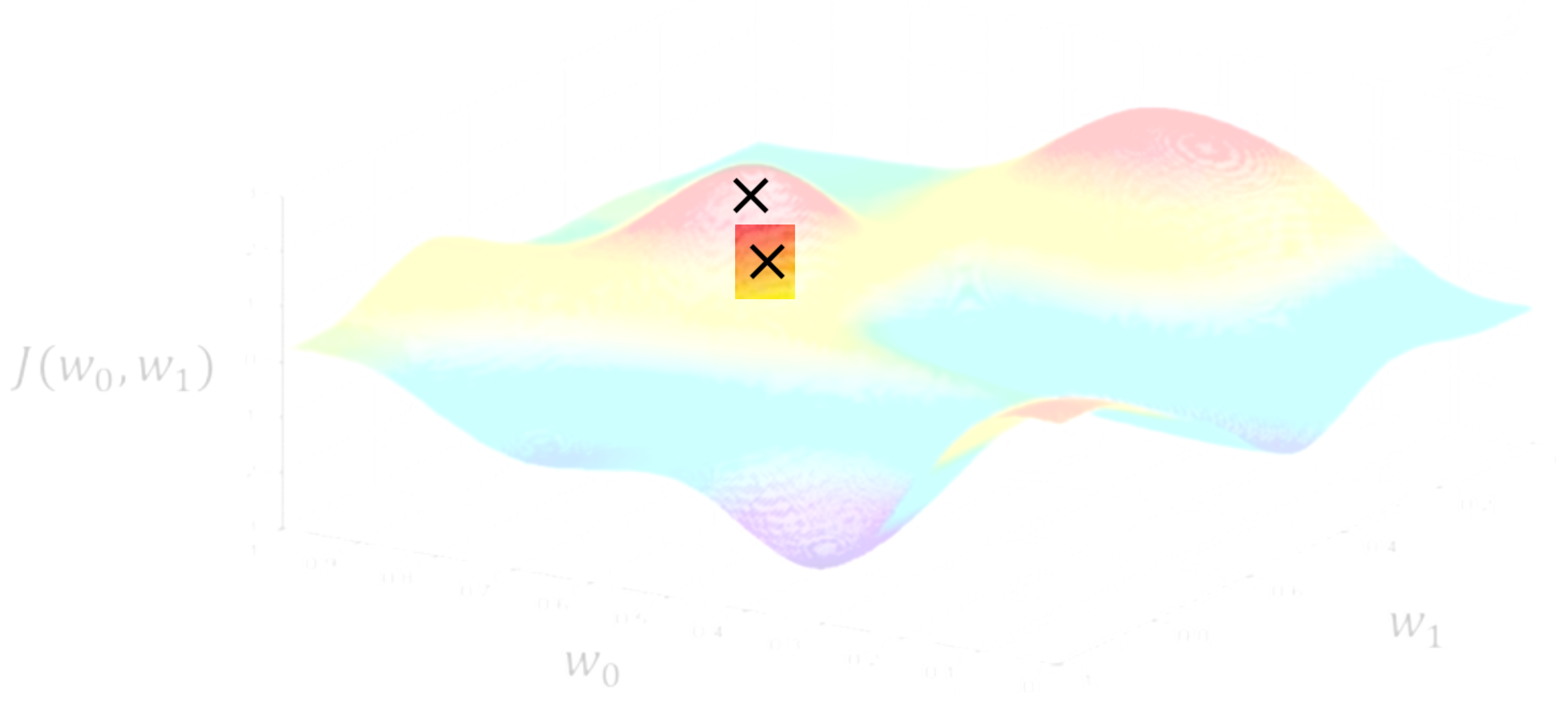
Take a small step in the opposite direction



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Loss Optimization

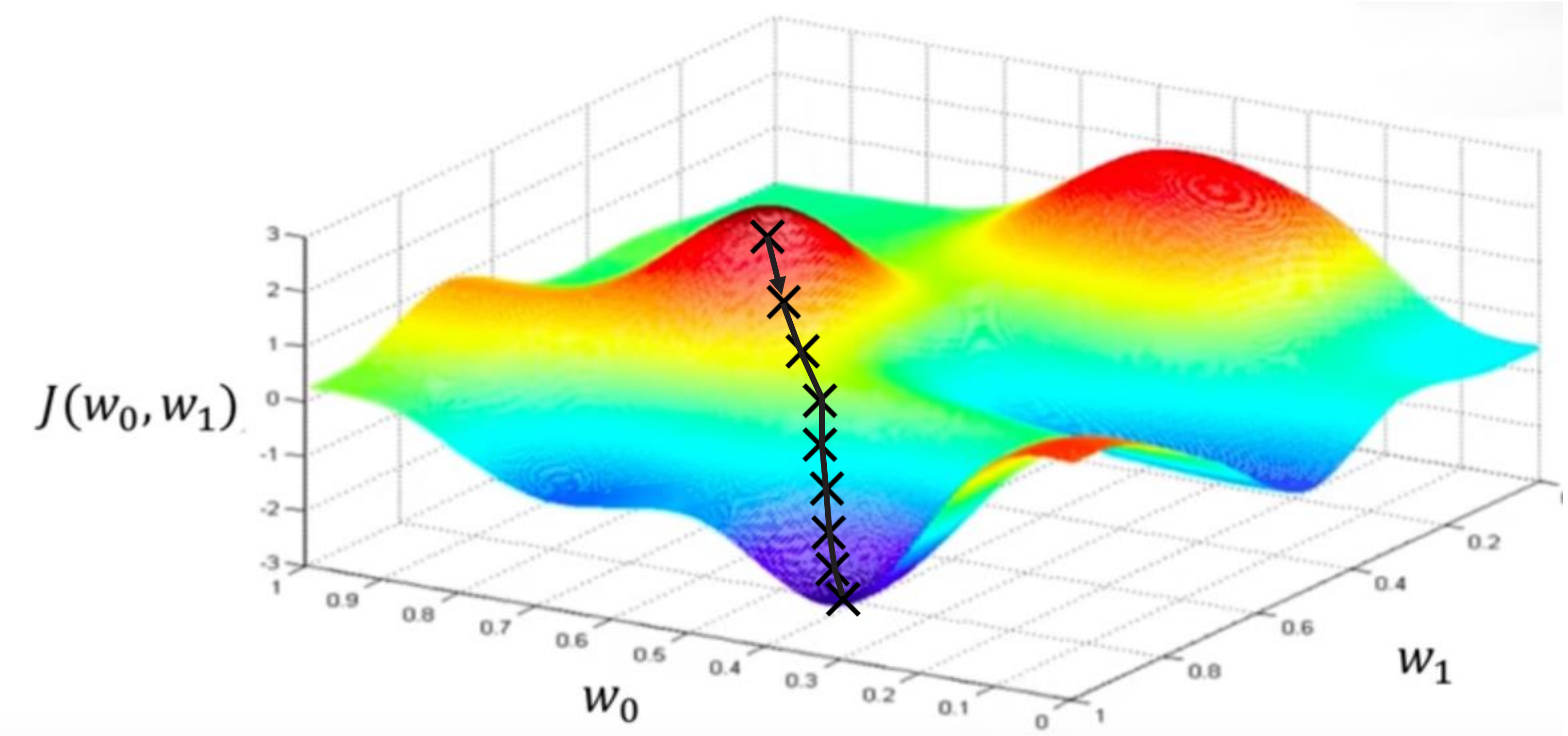
Take a small step in the opposite direction



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Loss Optimization

Repeat until convergence

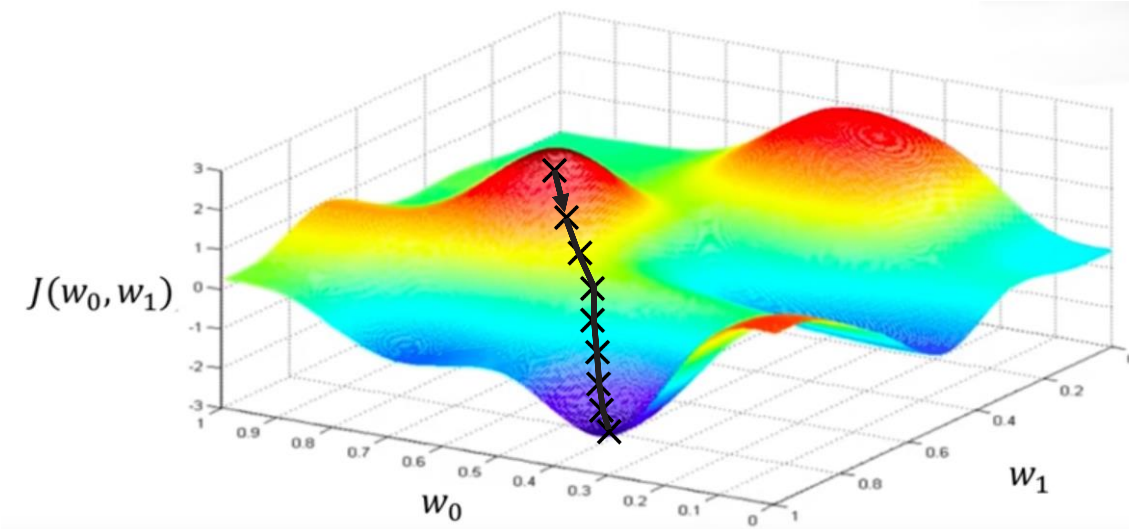


© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Gradient Descent

Algorithm

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



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

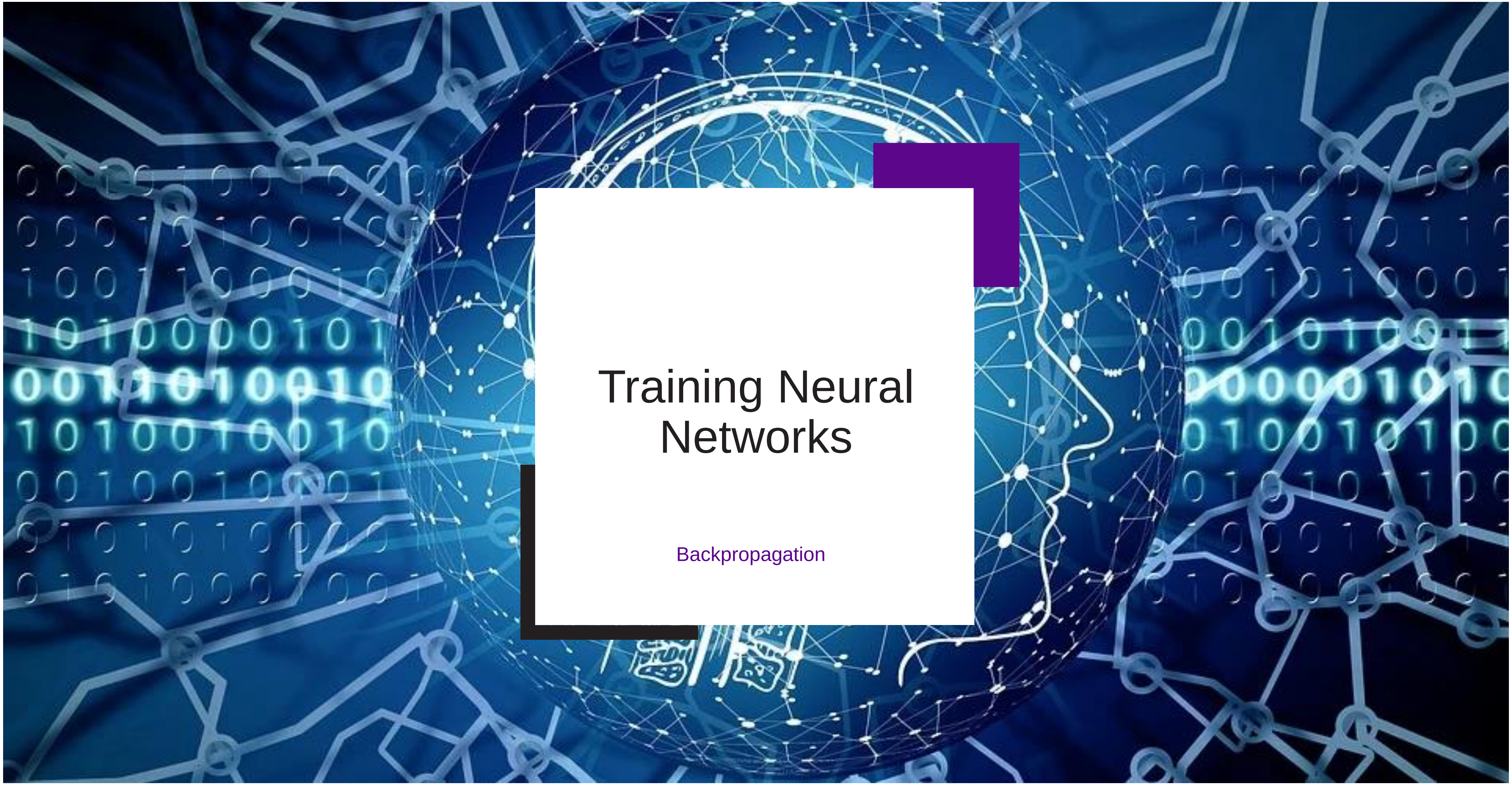
Gradient Descent

Algorithm

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

```python
import tensorflow as tf

lr = 0.001
weight = tf.Variable([tf.random.normal()])

while True: # must be replace by a convergence condition
    with tf.GradientTape() as g:
        loss = compute_loss(weights)
        gradient = g.gradient(loss, weights)

    weights = weights - lr * gradient
```

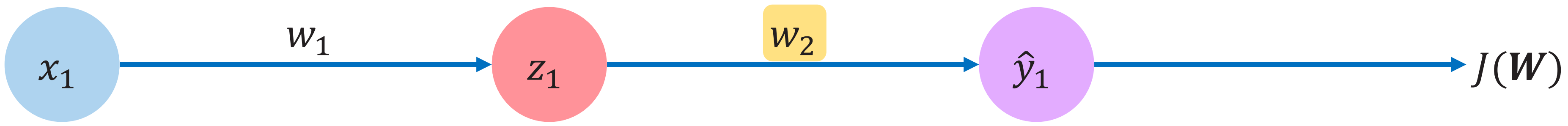

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning



Training Neural Networks

Backpropagation

Computing Gradients: Backpropagation

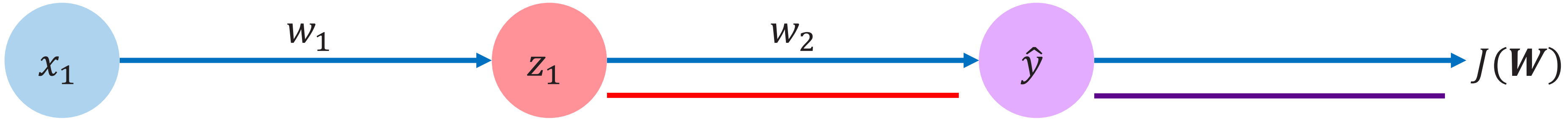


How does a small change in one weight (ex. w_2) affect the final loss $J(\boldsymbol{W})$?

It is represented by $\frac{\partial J(W)}{\partial w_2} = \nearrow or \searrow$

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

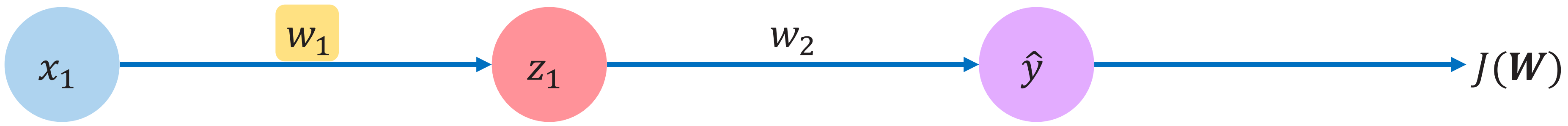
Computing Gradients: Backpropagation



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

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

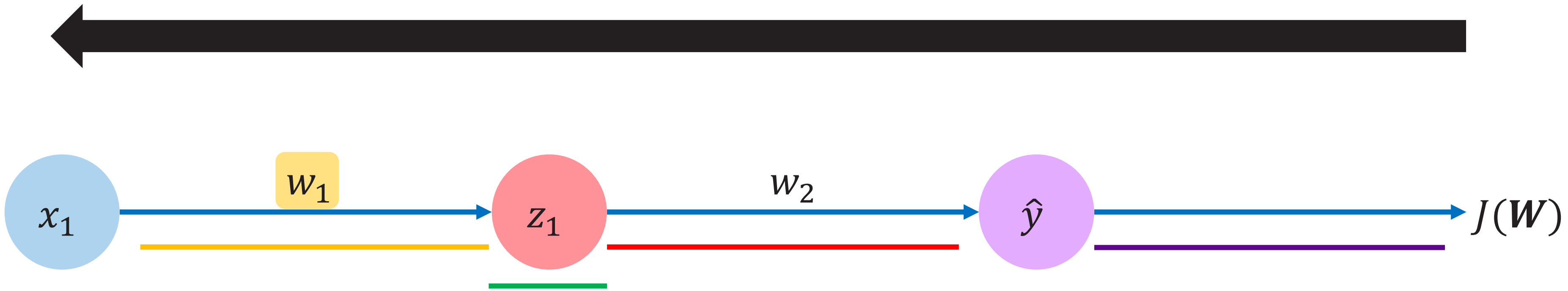
Computing Gradients: Backpropagation



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

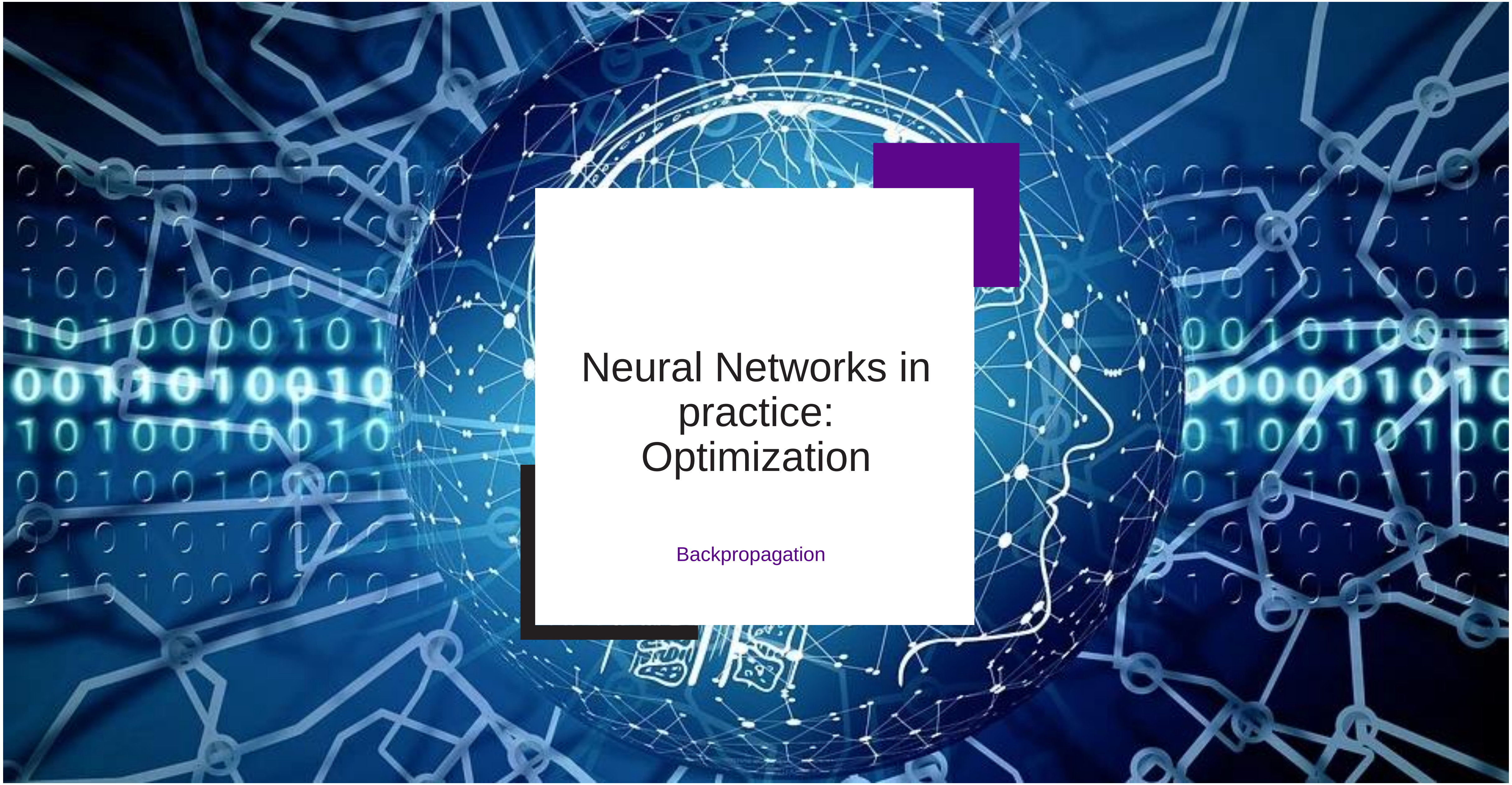
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2} * \frac{\partial w_2}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

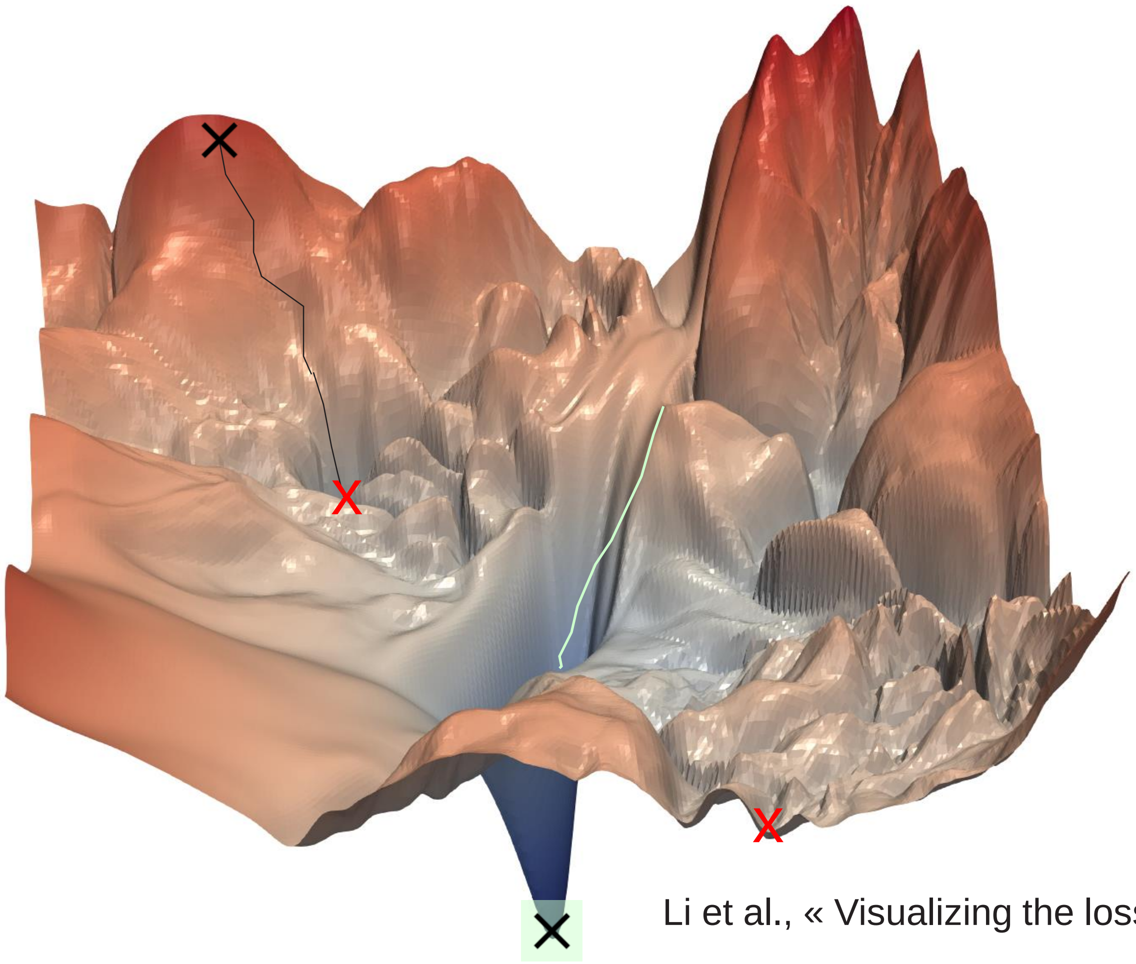
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning



Neural Networks in practice: Optimization

Backpropagation

Training Neural Networks is Difficult



Li et al., « Visualizing the loss landscape of neural nets », 2017

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Loss function can be difficult to optimize

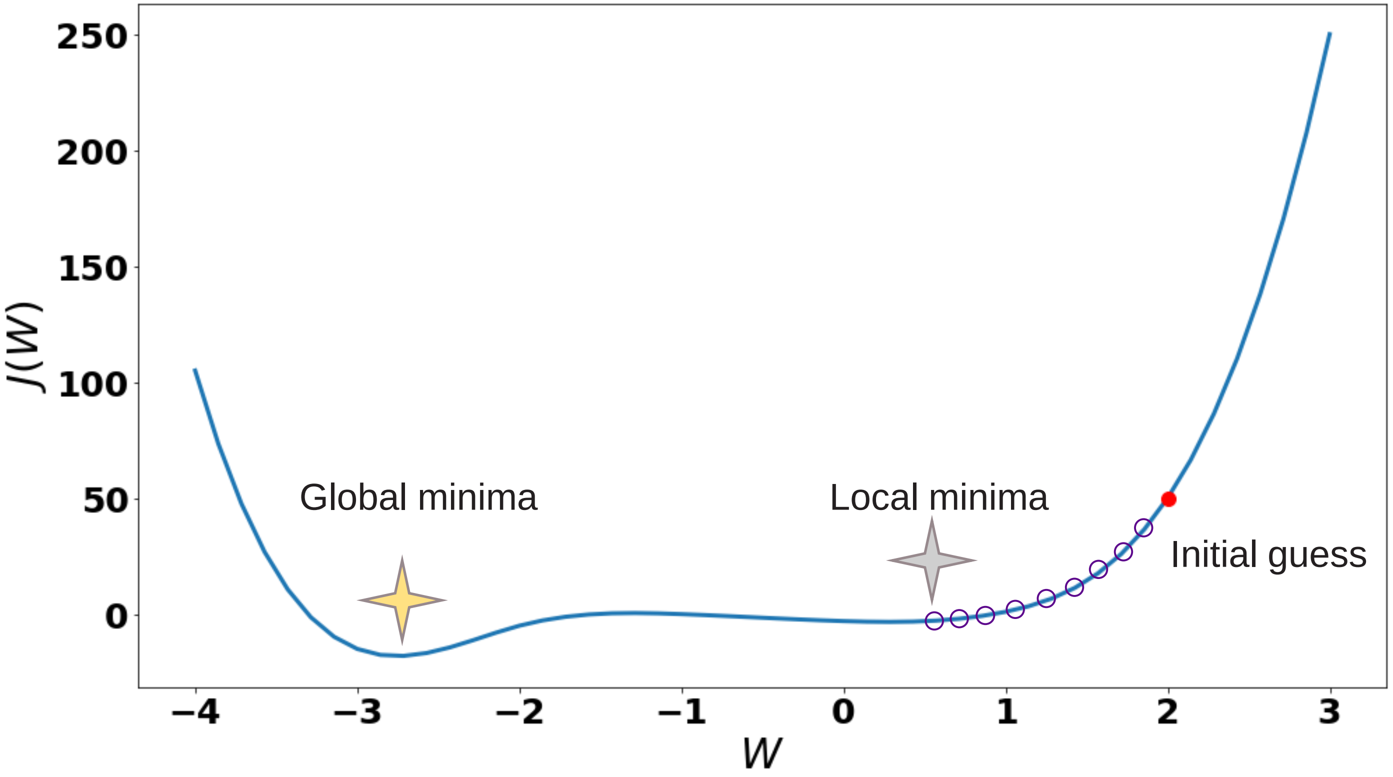
Remember that optimization is done thanks to **gradient descent** algorithm

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

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Setting the Learning Rate η

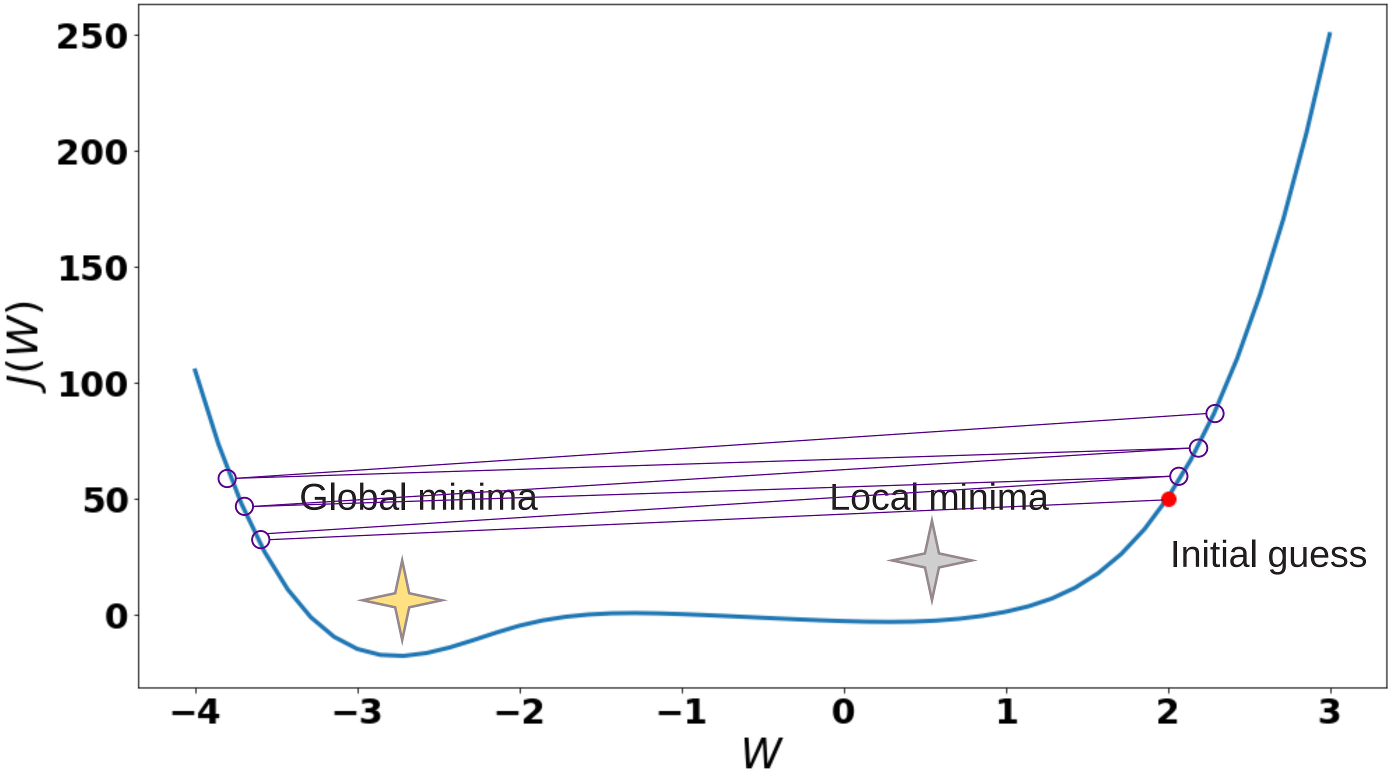
Small learning rates converge slowly and/or get stick in local minima



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Setting the Learning Rate η

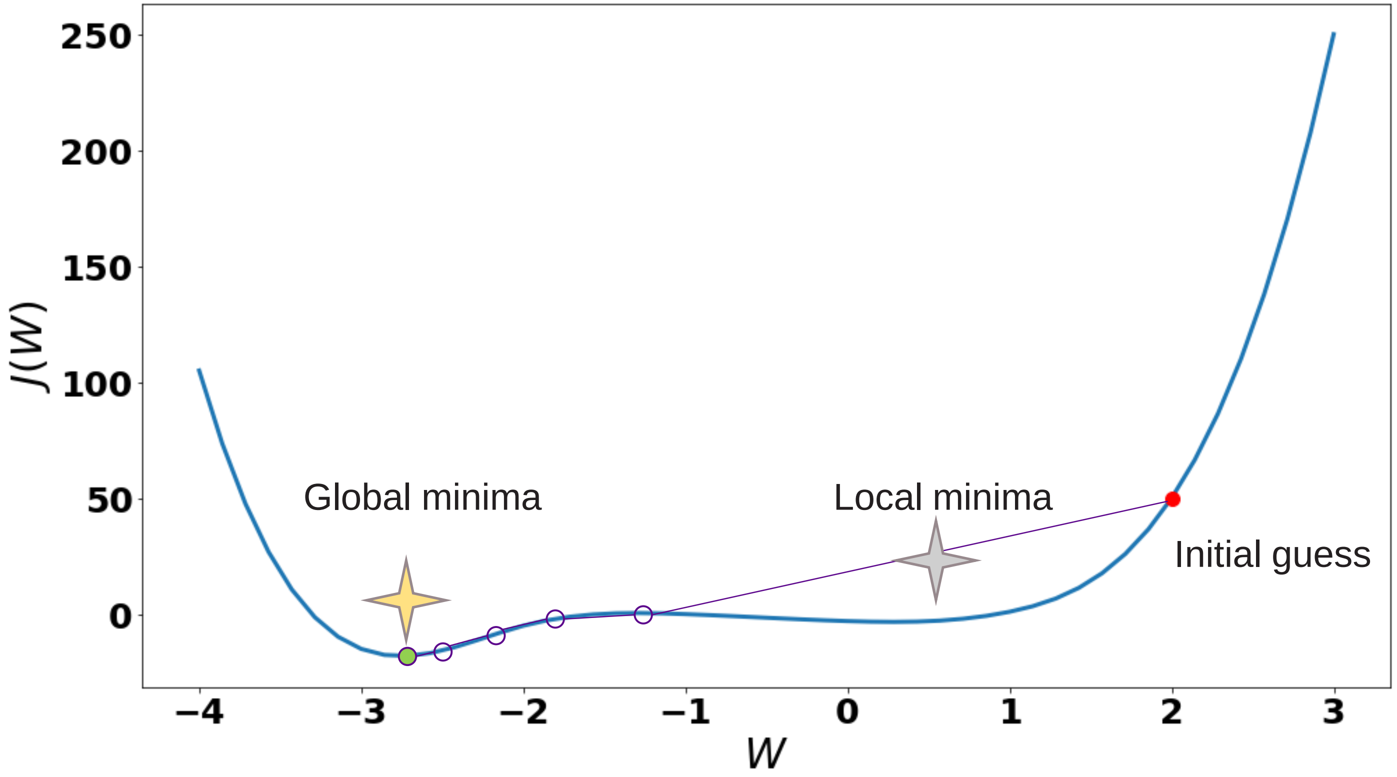
Large learning rates overshoot, become unstable and diverge



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Setting the Learning Rate η

Stable learning rates converge smoothly and avoid local minima



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

How to smartly Choose the Learning Rate η

Idea 1 (empirically):

Try a lots of different learning rates and choose the more efficient one

Idea 2 (adaptive one):

Design an adaptive learning rate that « adapts » to the landscape

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Setting the Learning Rate η

Don’t use a fixed learning rate over the training

Choose depending on:

- How large the gradient is
- How fast the learning is occuring
- Size of a particular weights
- …

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Gradient Descent Algorithms

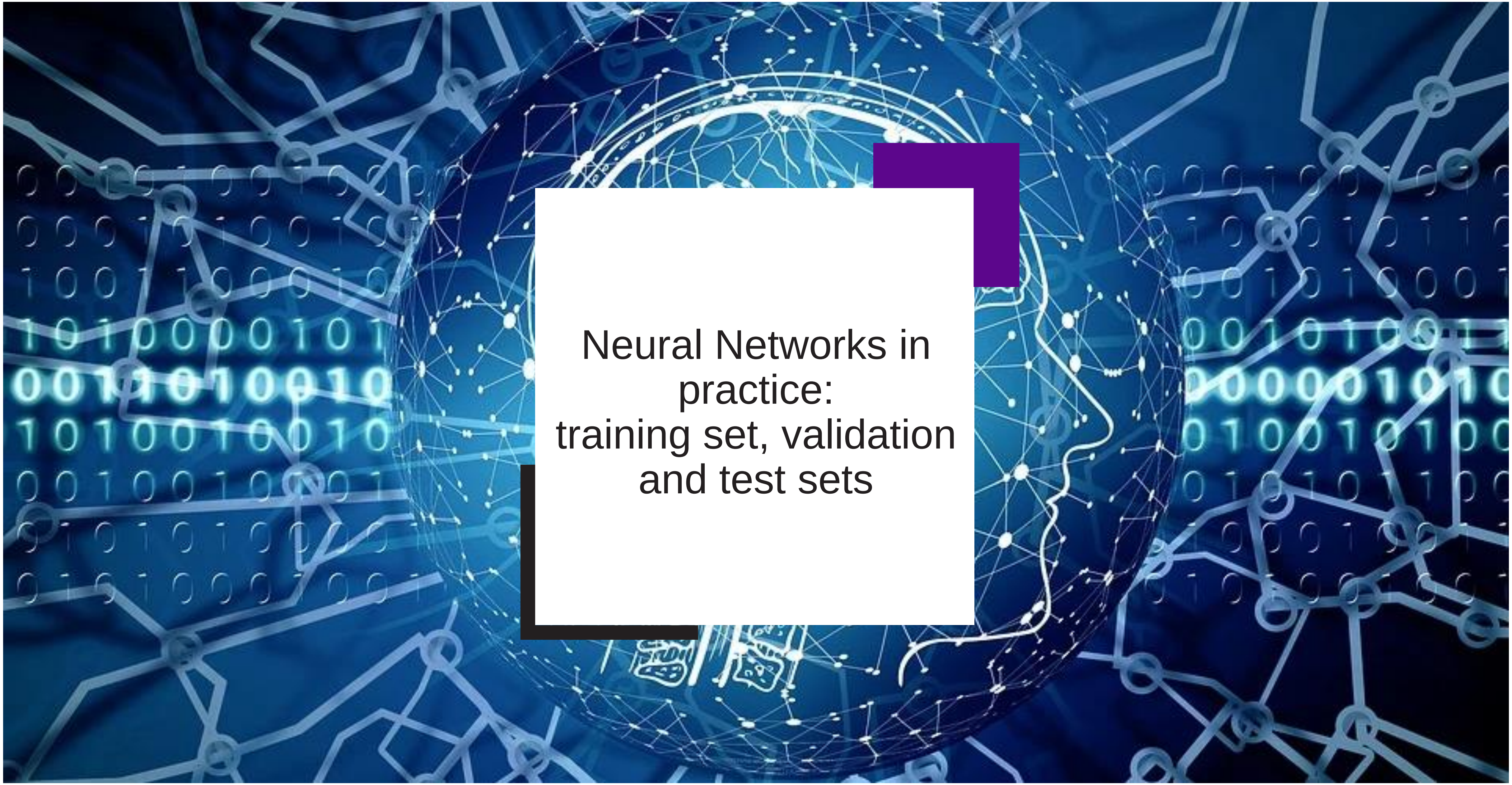
Algorithm	Tensorflow	Reference
SGD (Stochastic Gradient Descent)	tf.keras.optimizers.SGD	Kiefer, J., & Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. The Annals of Mathematical Statistics, 23(3), 462-466.
Adam	tf.keras.optimizers.Adam	Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
Adadelta	tf.keras.optimizers.Adadelta	Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.
Adagrad	tf.keras.optimizers.Adagrad	Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).
RMSProp	tf.keras.optimizers.RMSProp	Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Gradient Descent in Tensorflow

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

# model to define
model = model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=64, activation='sigmoid'),
    tf.keras.layers.Dense(units=10, activation='sigmoid')
    ])
# Instantiate an optimizer and the loss function to use (pointer on function)
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-3)
compute_loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
# train
for epoch in range(500):
    # Open a GradientTape to record the operations for computing gradient
    with tf.GradientTape() as tape:
        # Run the forward pass of the layer.
        prediction = model(x_train, training=True)  # Logits for this minibatch
        # Compute the loss value for this minibatch.
        loss_value = compute_loss(y_train, prediction)
    # update the weights using the gradient
    grads = tape.gradient(loss_value, model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))
```


© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning



Neural Networks in practice: training set, validation and test sets

Training, validation and test sets

Among the whole label data, we need to define several sets

Basic approach

Whole Labeled Dataset

Training set 70% | Test set 30%

Training set: used to train the algorithm

Test set: used to evaluate your algorithm performance

Each set MUST HAVE the **same distribution** as the Whole Labelled Dataset

Training, validation and test sets

Among the whole label data, we need to define several sets

Advanced approach (for testing several Neural Network architectures or hyper parameters)

Whole Labeled Dataset

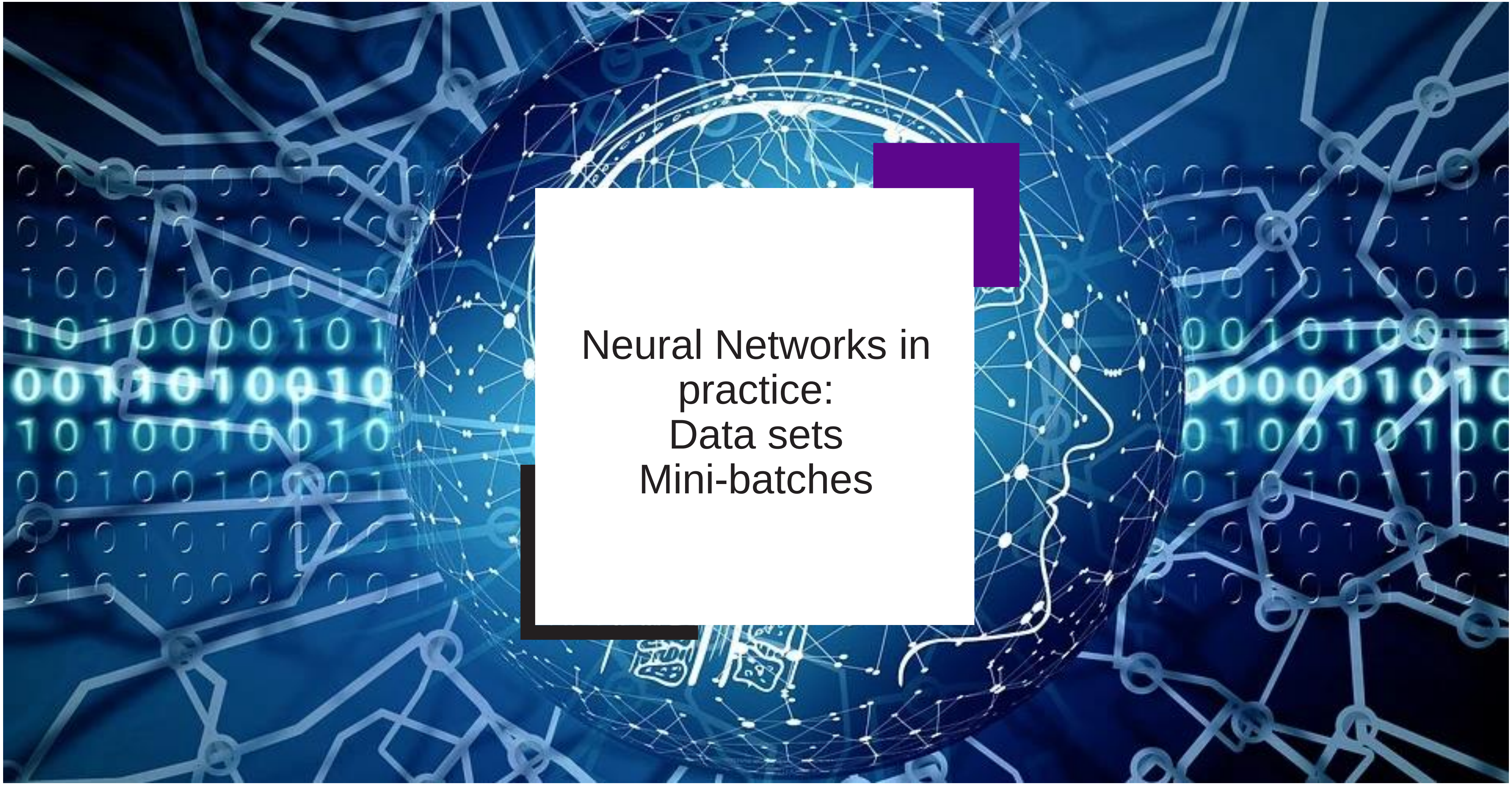
Training set 60% | Validation set 20% | Test set 20%

Training set: used to train the algorithm
Validation set: used to tune networks hyperparameters or select among several networks architecture

Test set: used to evaluate your algorithm performance

Each set MUST HAVE the **same distribution** as the Whole Labelled Dataset

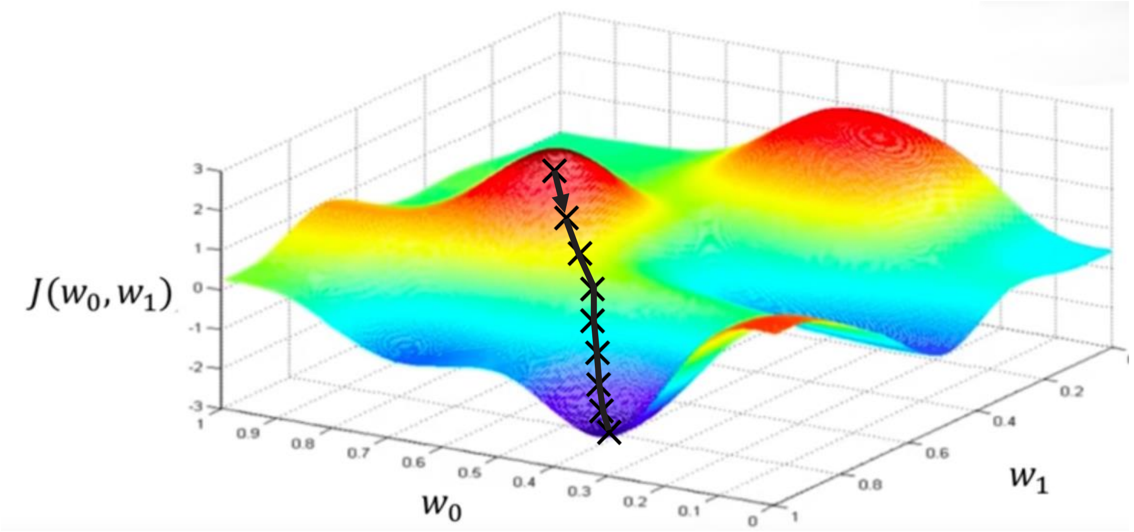
Neural Networks in practice:
Data sets
Mini-batches



Gradient Descent

Algorithm

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



Labelled Data

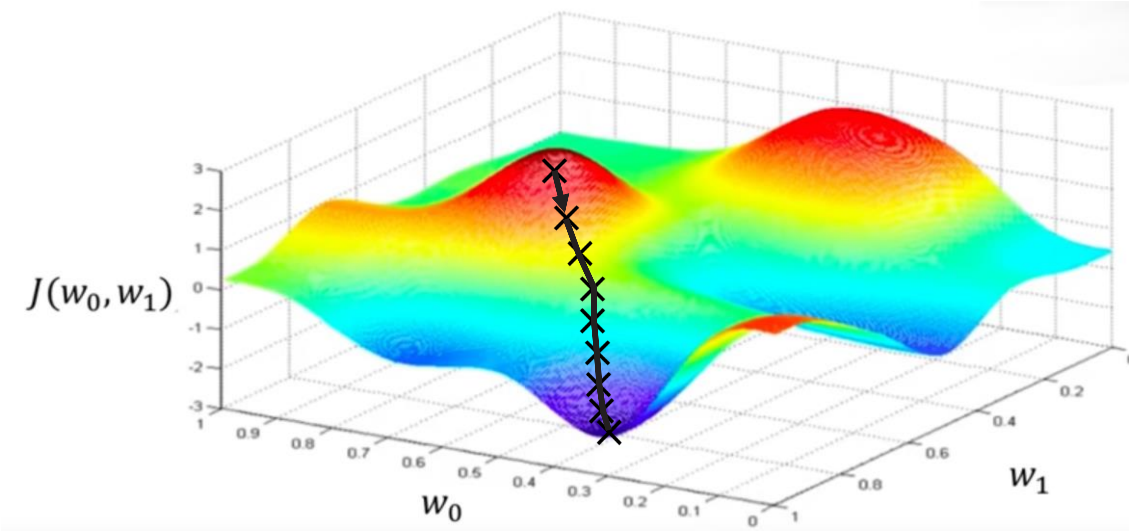
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Stochastic Gradient Descent

Algorithm

1. Initialize the weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick only <u>1 sample</u>
4. Compute gradient, $\boxed{\frac{\partial J(W)}{\partial W} = \frac{\partial J_i(W)}{\partial W}}$
5. Update weights, $W = W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights

Easy to compute but **Very noisy**
(too stochastic)



Labelled Data

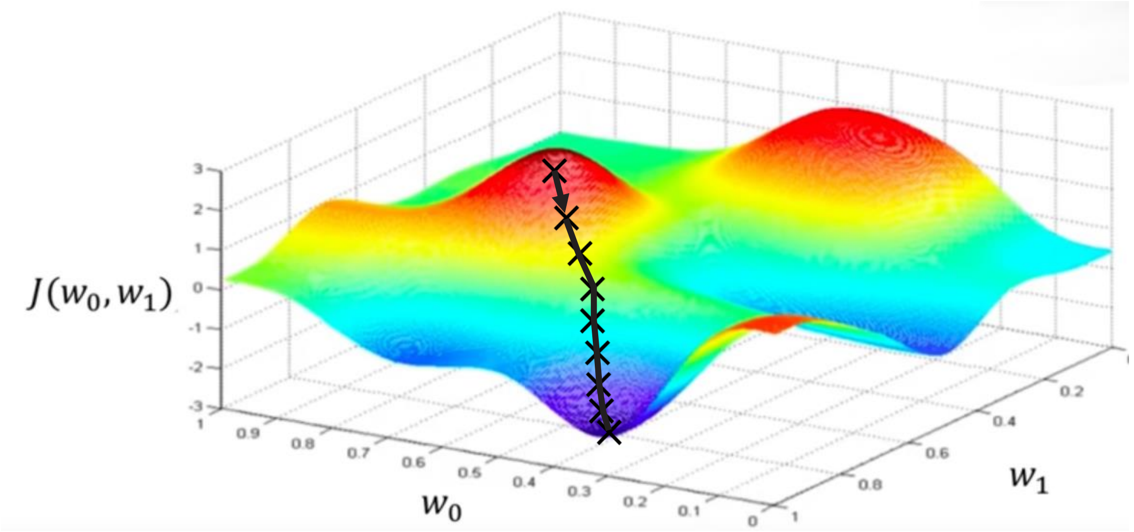
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Stochastic Gradient Descent

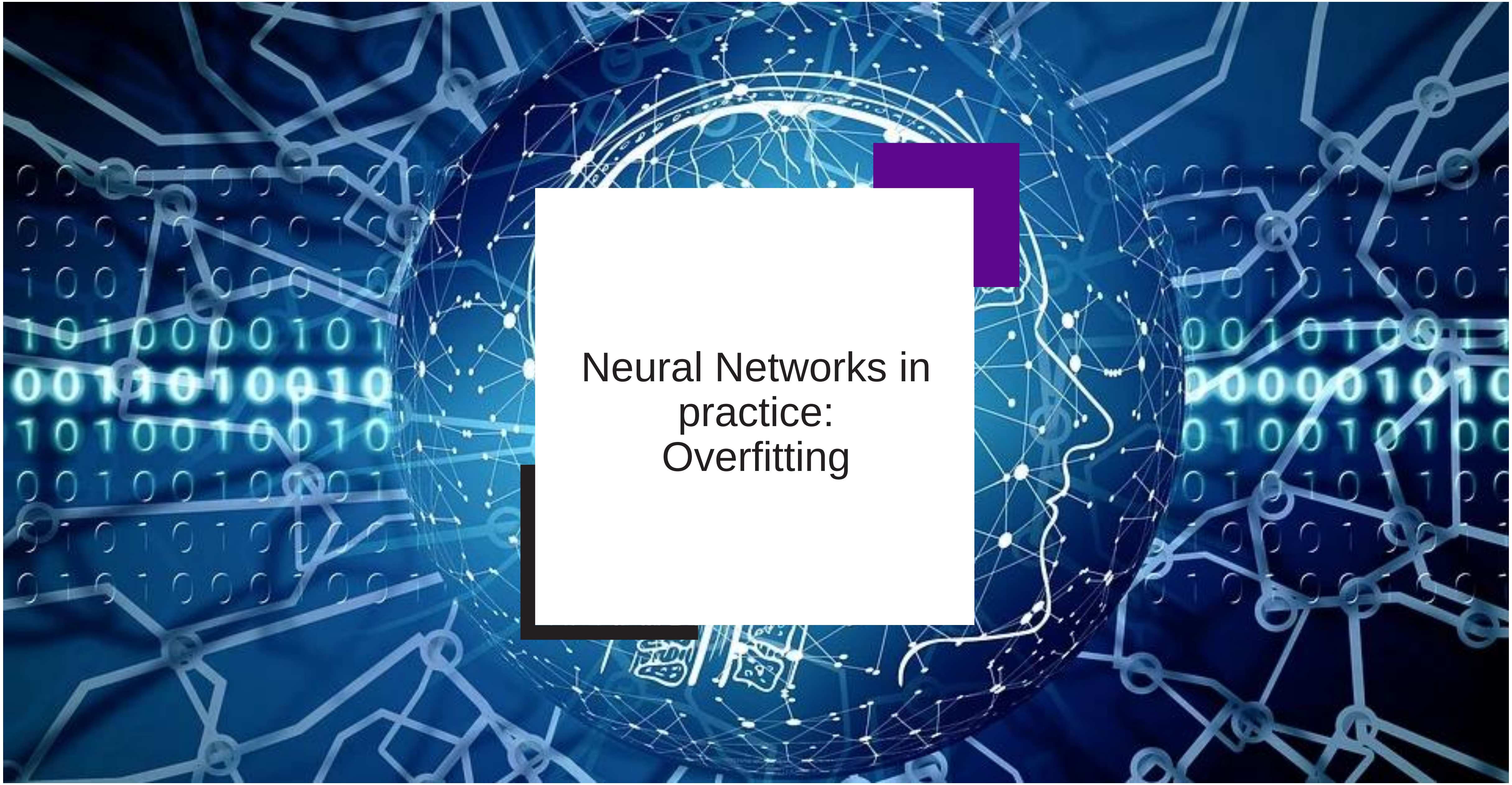
Algorithm

1. Initialize the weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick a batch of B samples
4. Compute gradient, $\boxed{\frac{\partial J(W)}{\partial W} = \frac{1}{B}\sum_{i=1}^{B}\frac{\partial J_i(W)}{\partial W}}$
5. Update weights, $W = W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights

More accurate estimation of gradient
Smoother convergence
Allows **larger learning rates**
Parrallelizable on GPUs
Scalable to huge data

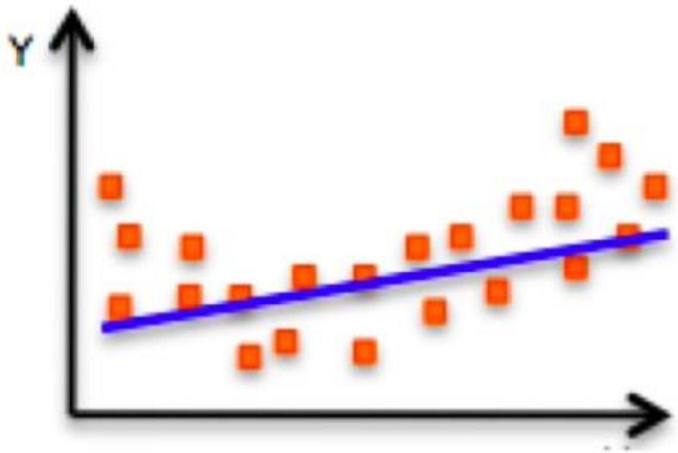


© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning



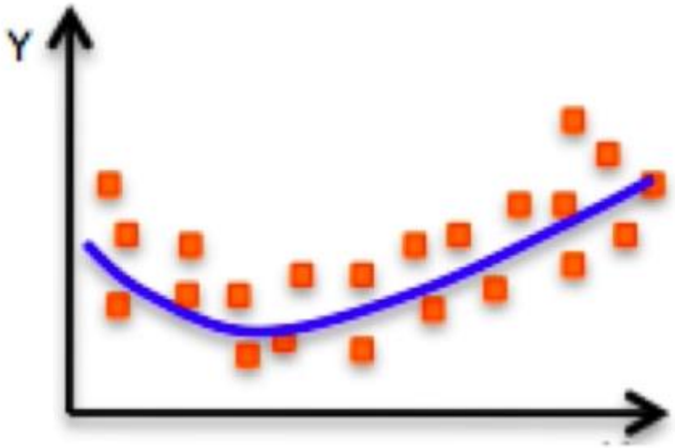
Neural Networks in practice: Overfitting

The Problem of Overfitting

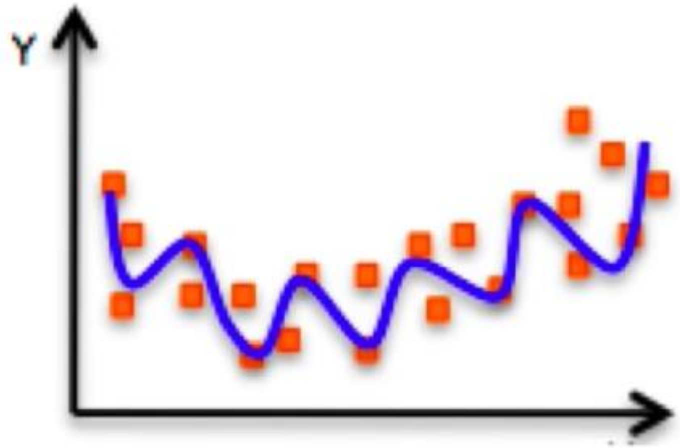


Underfitting = <u>High bias</u>

Model does not have capacity to fully learn the data



Ideal fit



Overfitting = <u>High variance</u>

Learn by heart
Model does not have capacity to generalize well

Too complex representation / too many parameters

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization

The **regularization** is a technique that constraints our optimization problem to **discourage too complex models**.

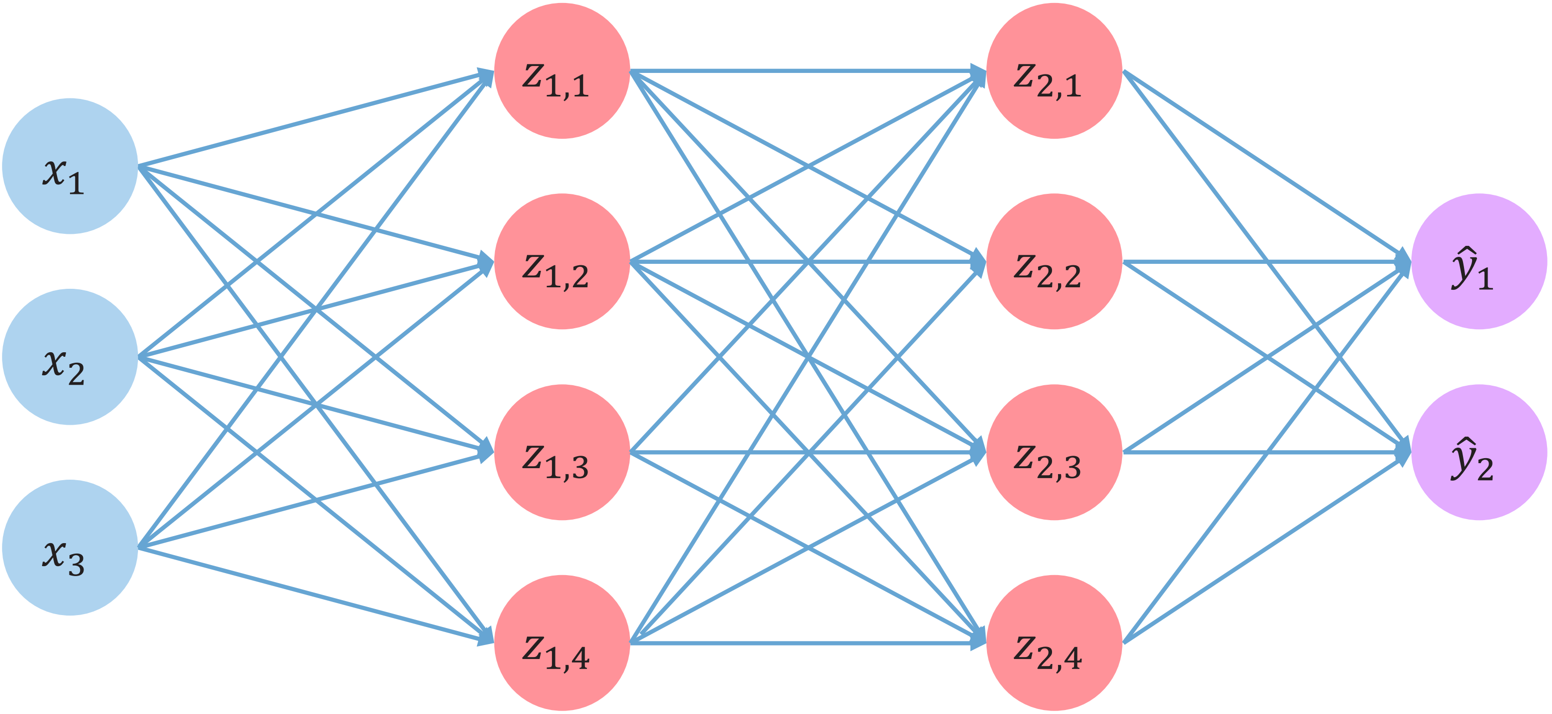
Why do we need it?

Improve our generalization of our model on unseen data

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization I: Dropout

During training, randomly set activations to 0



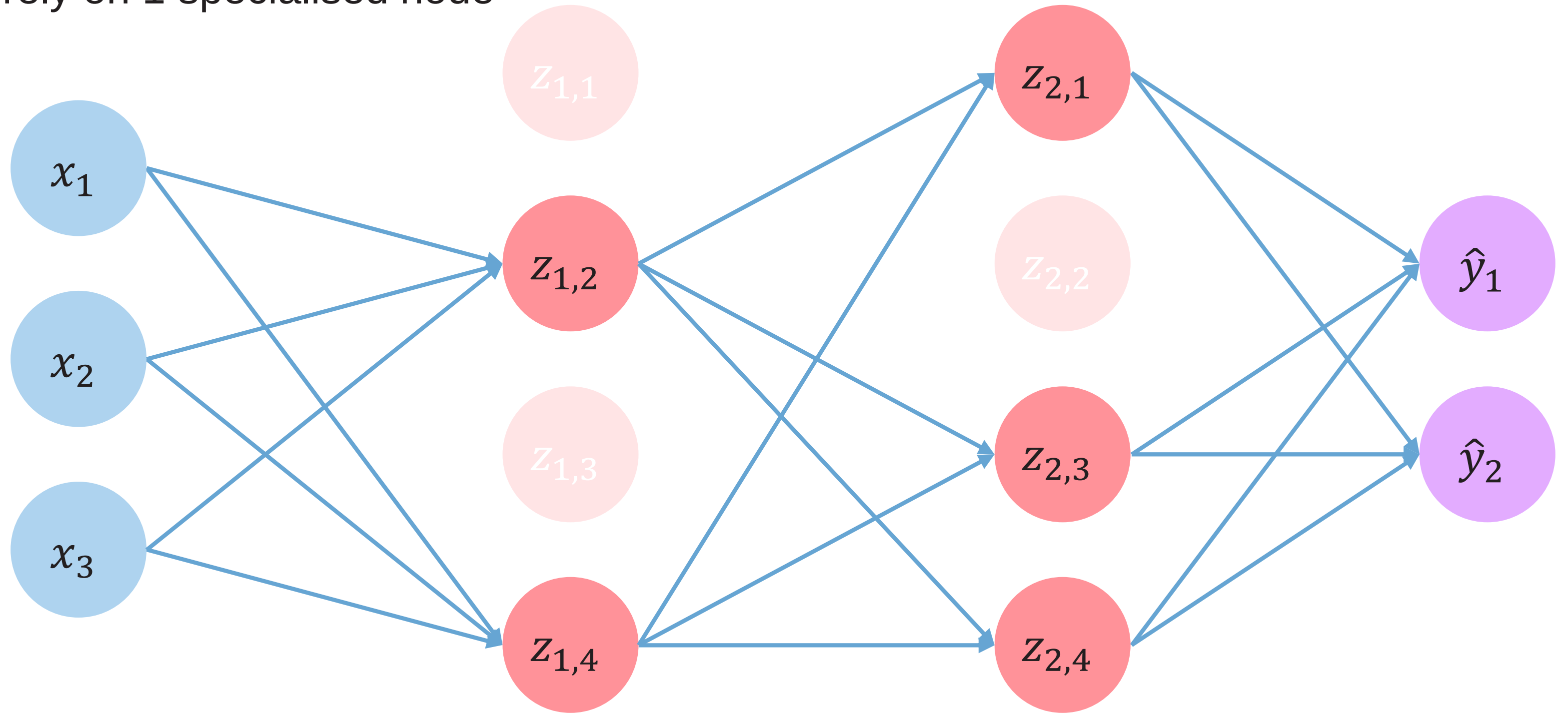
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization I: Dropout

During training, randomly set activations to 0

- Typically, drop 50% of activations layer
- Forces network to not rely on 1 specialised node

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



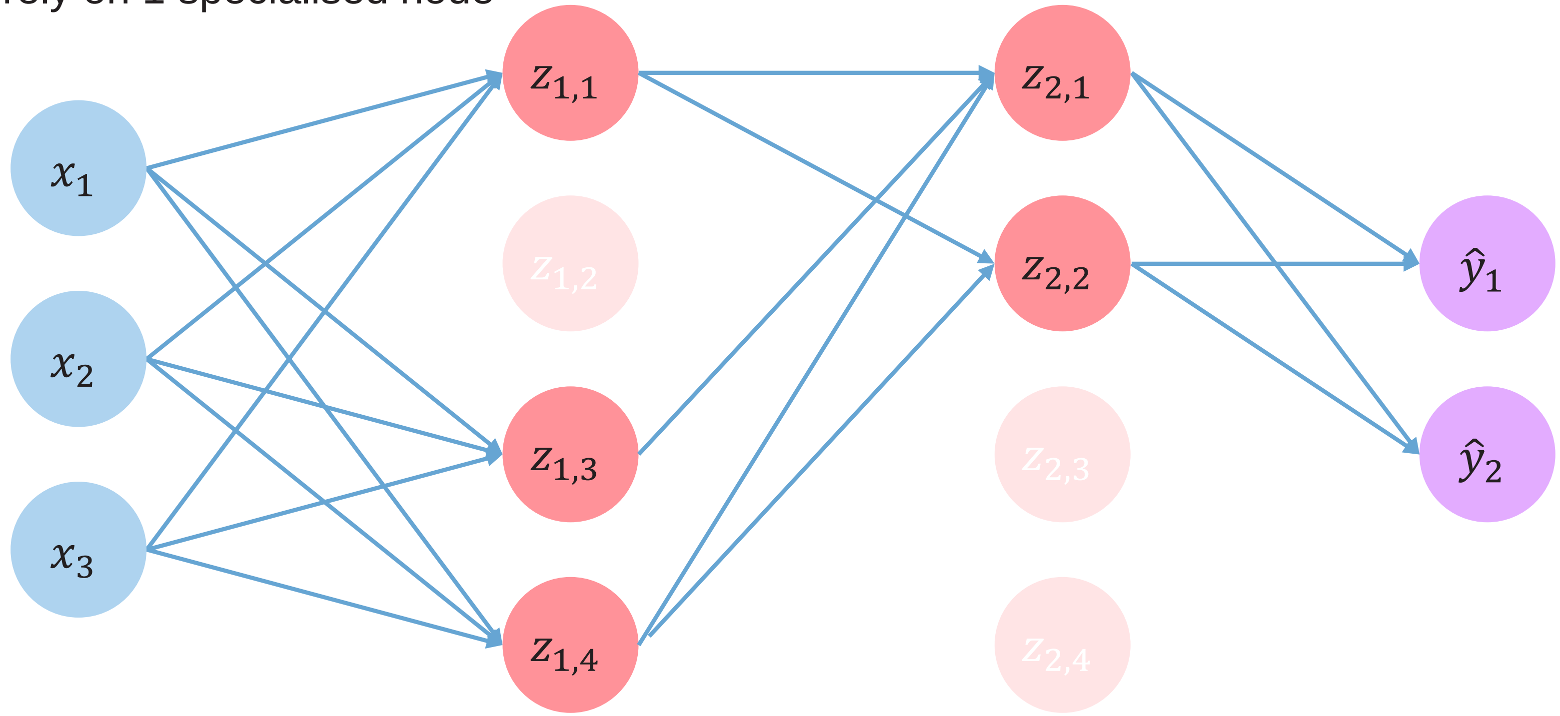
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization I: Dropout

During training, randomly set activations to 0

- Typically, drop 50% of activations layer
- Forces network to not rely on 1 specialised node

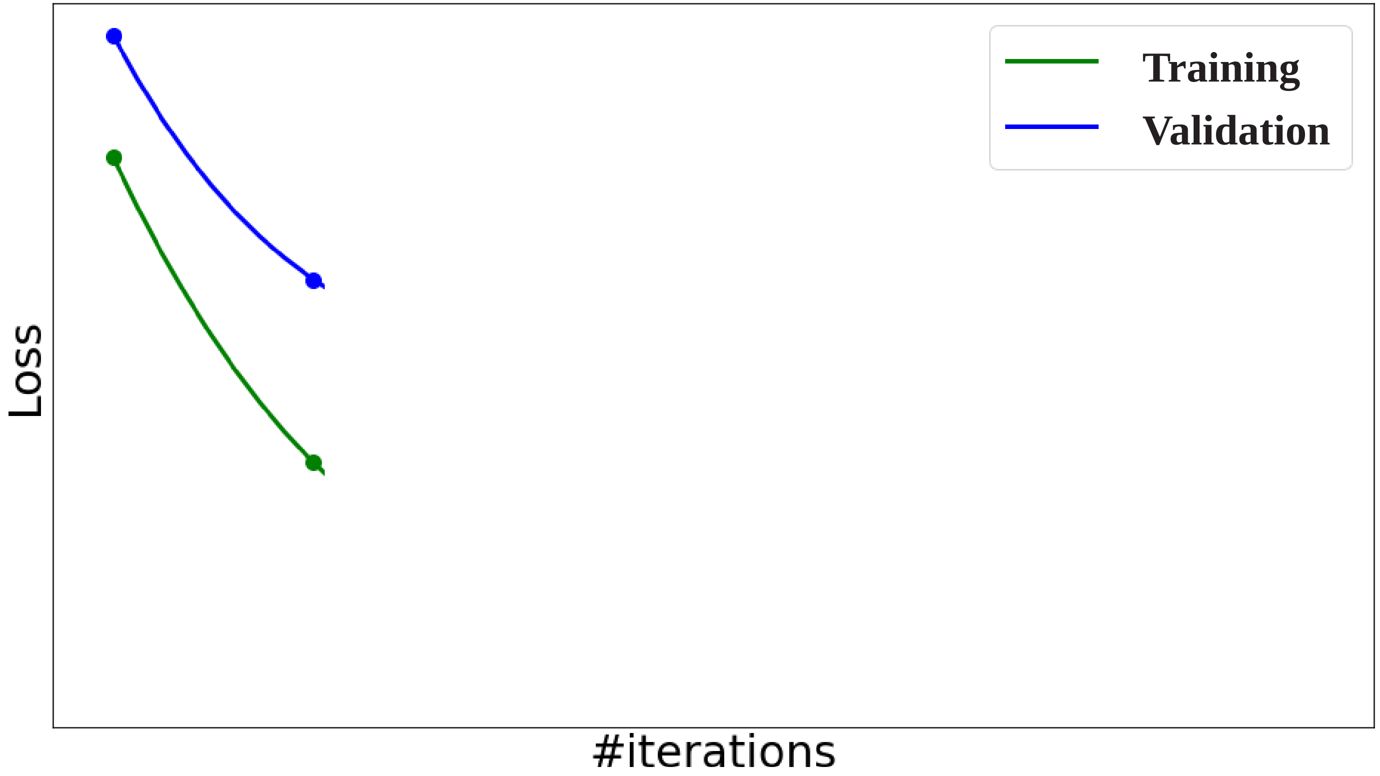
```
tf.keras.layers.Dropout(p=0.5)
```



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization II: Early stopping

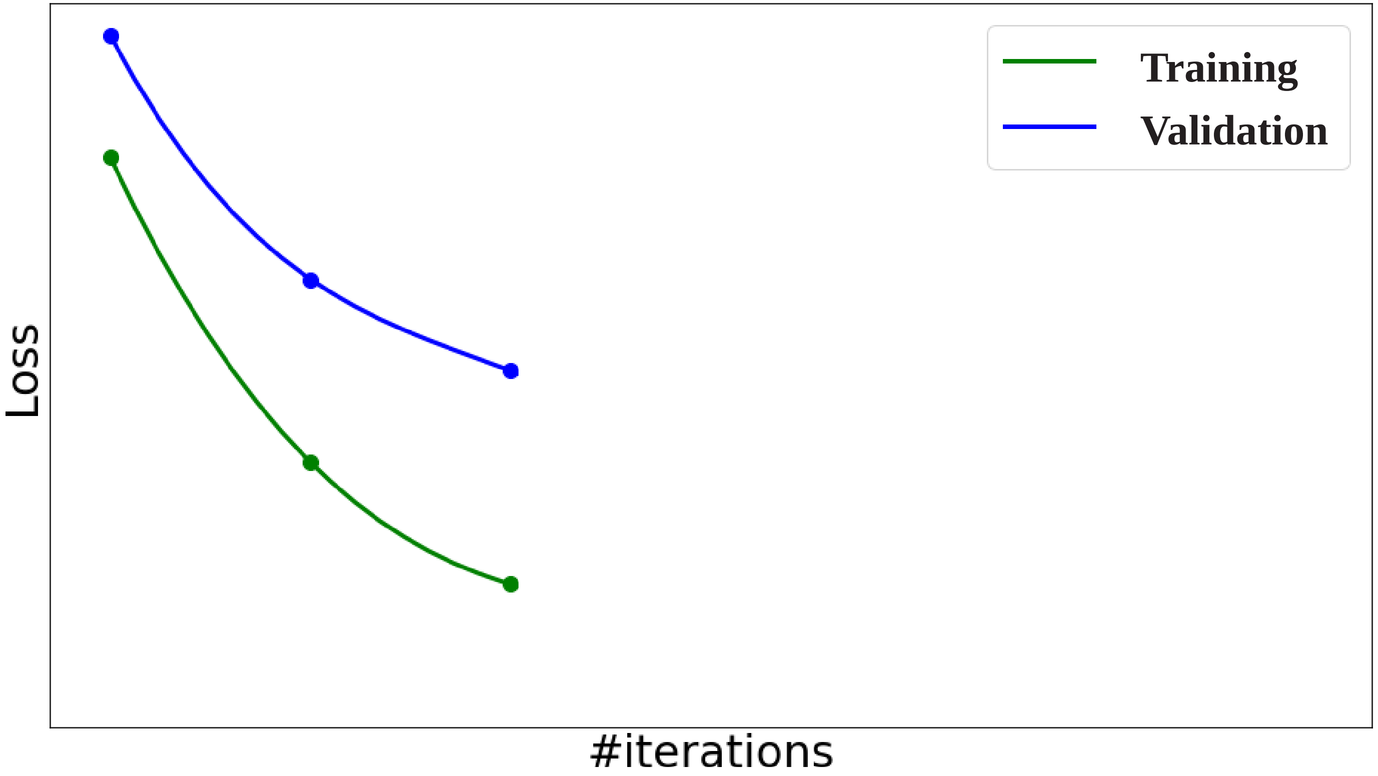
Stop training before we have a chance to overfit



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization II: Early stopping

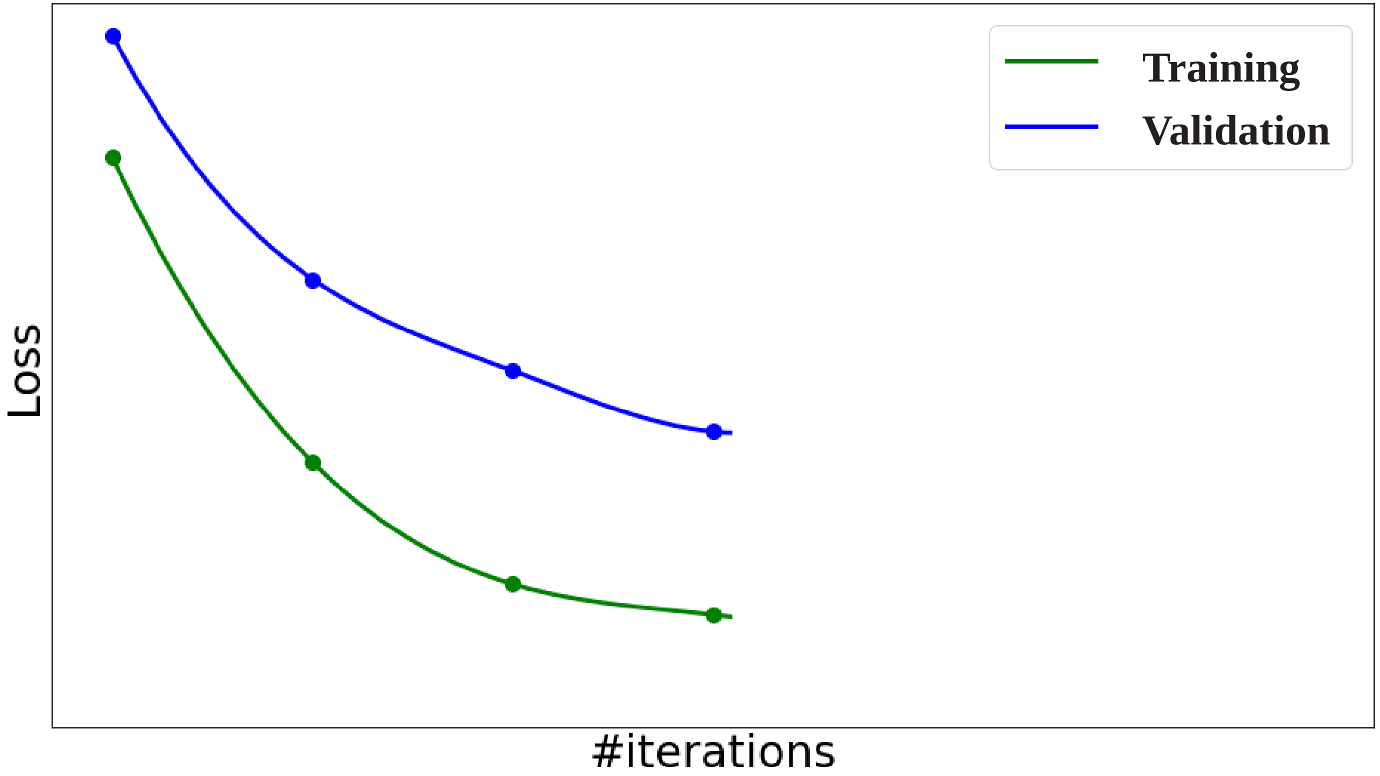
Stop training before we have a chance to overfit



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization II: Early stopping

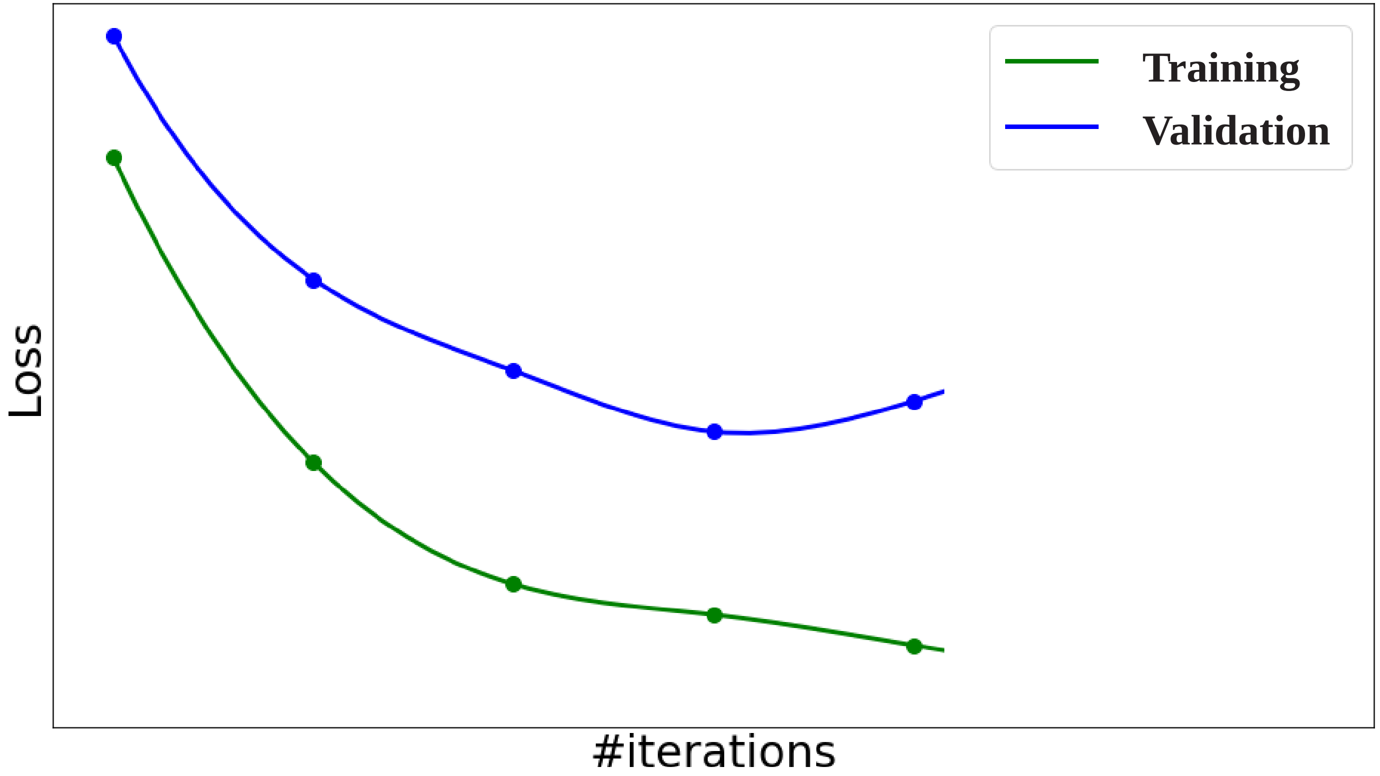
Stop training before we have a chance to overfit



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization II: Early stopping

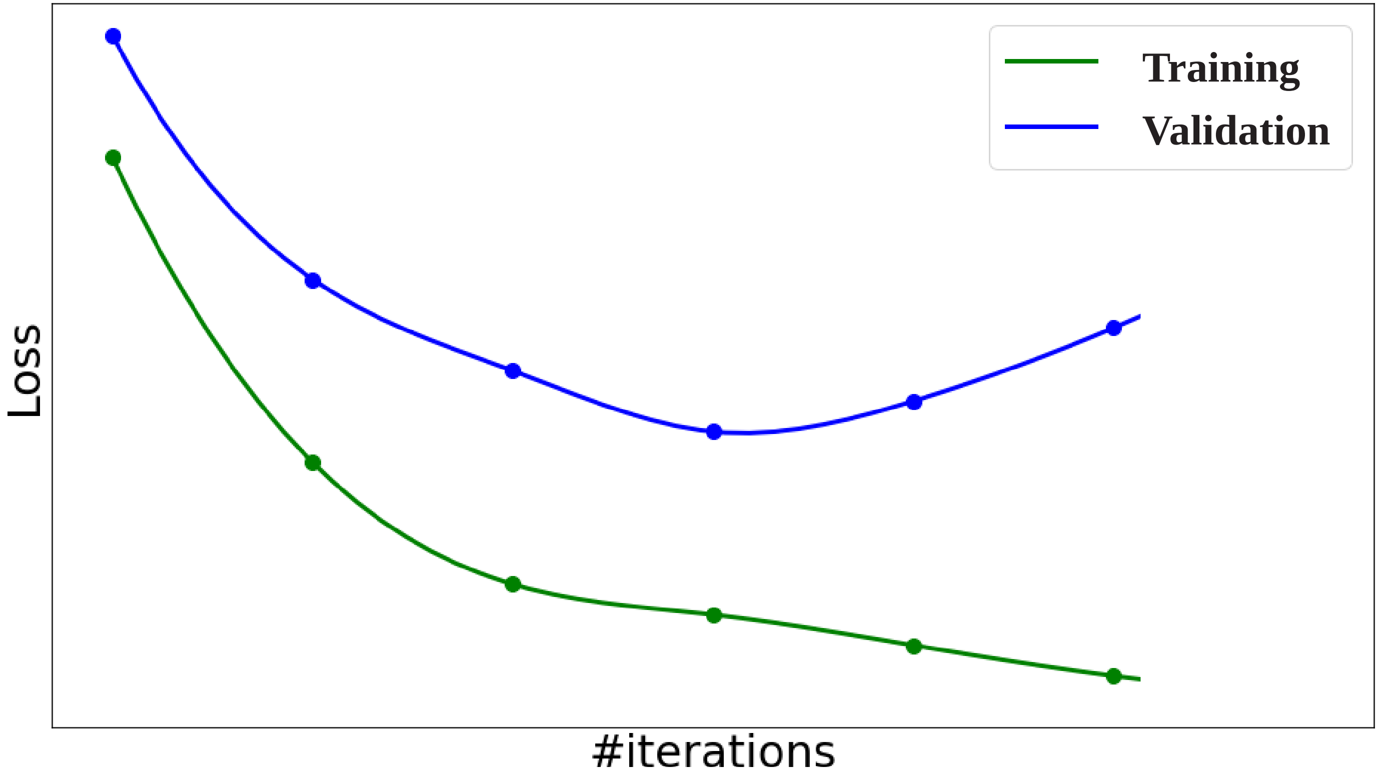
Stop training before we have a chance to overfit



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization II: Early stopping

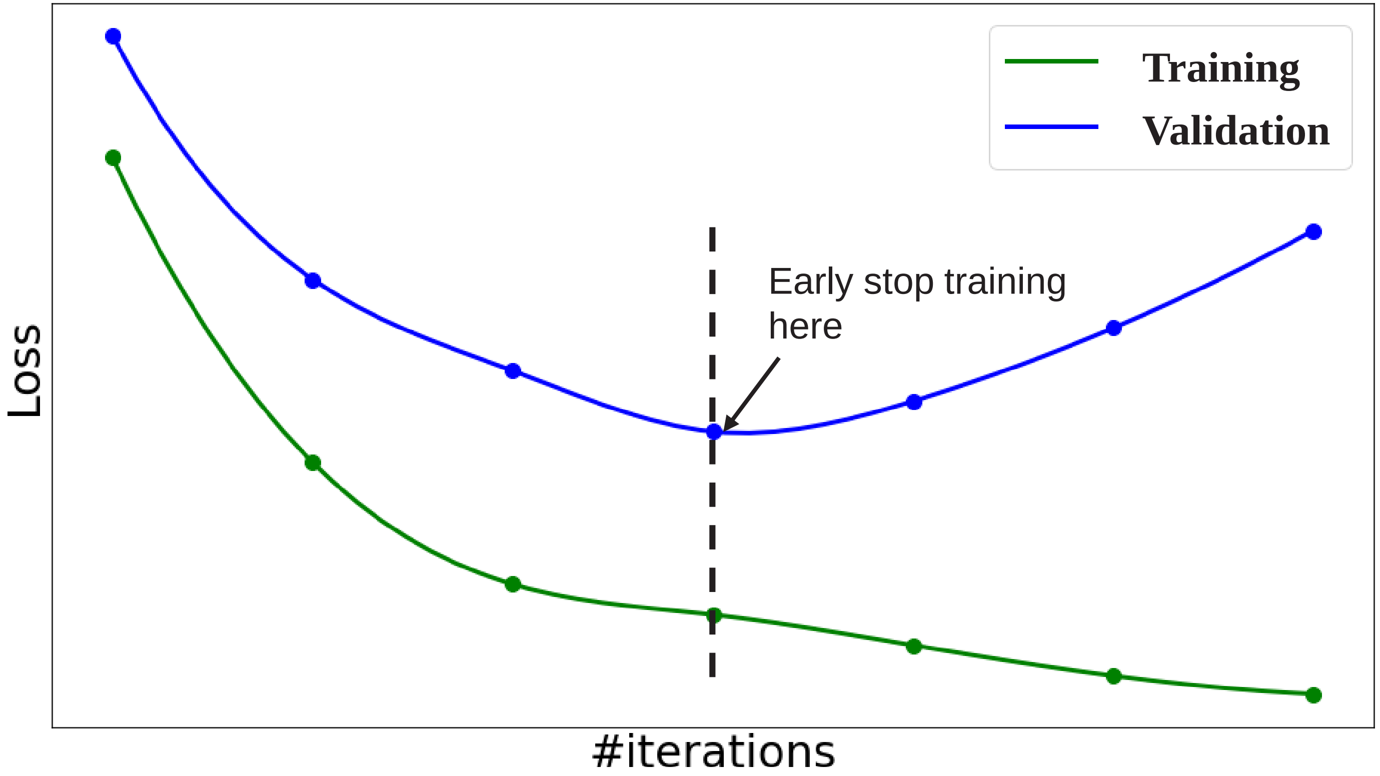
Stop training before we have a chance to overfit



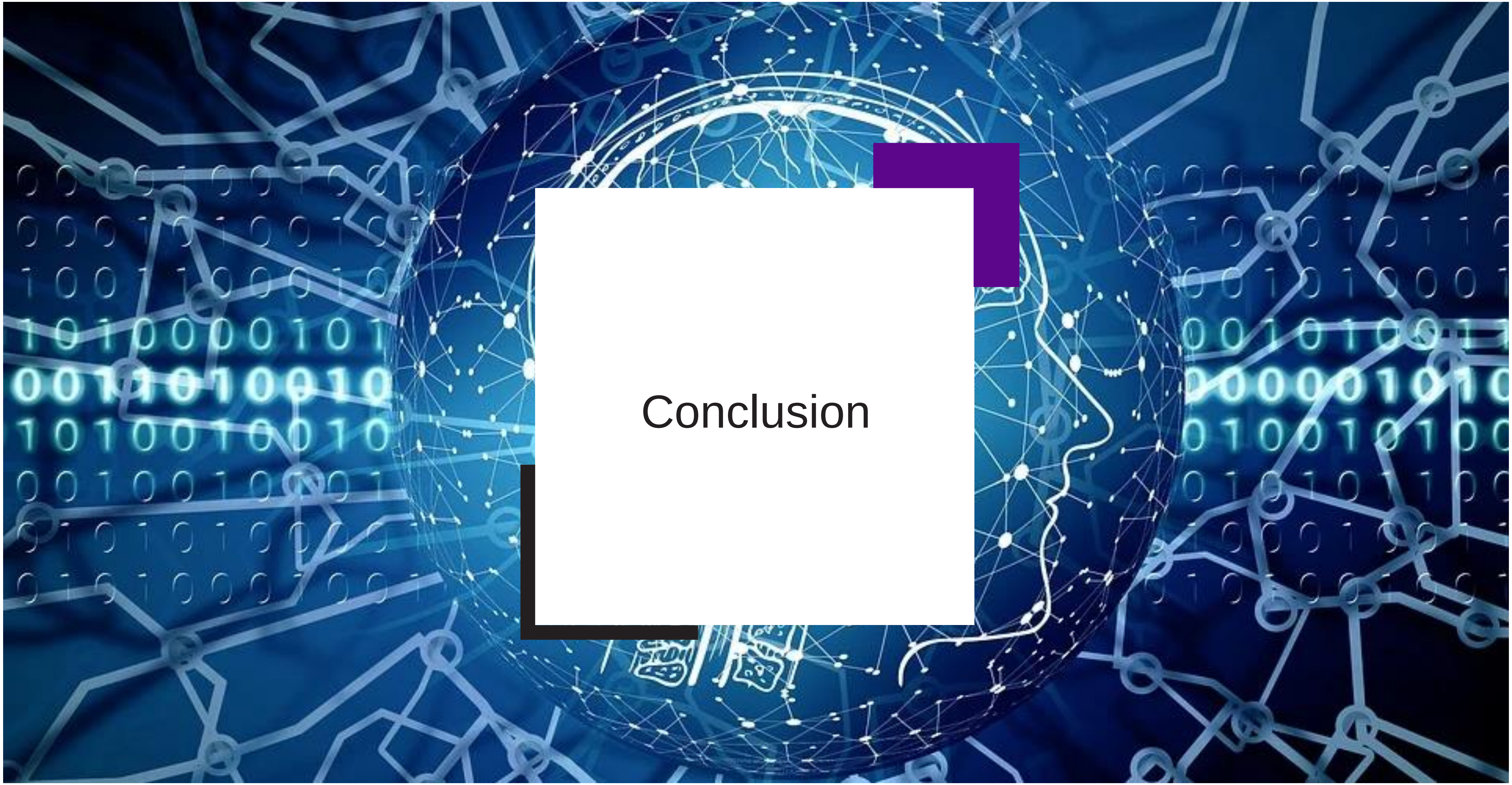
© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Regularization II: Early stopping

Stop training before we have a chance to overfit



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

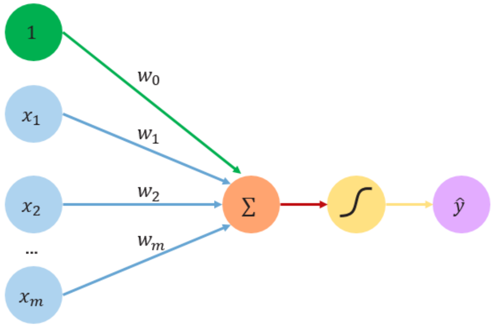


Conclusion

Conclusion about Neural Networks

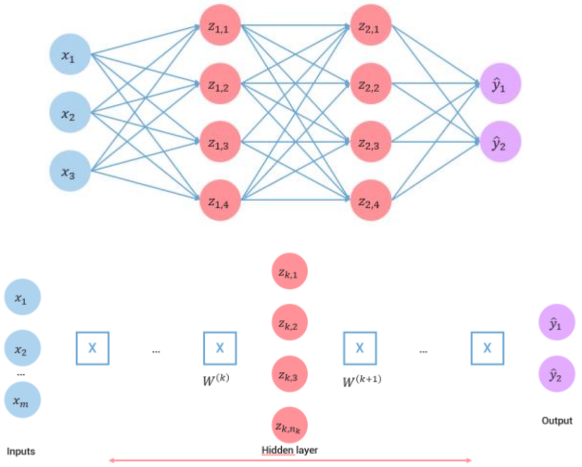
The Perceptron

- Structural building blocks
- Nonlinear activation functions



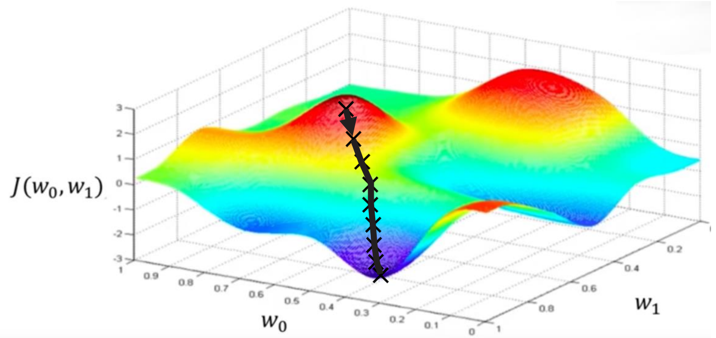
Neural Networks (NN)

- Stacking Perceptrons to form Deep Neural networks
- Optimization through Backpropagation



Training NN in Practice

- Stochastic Gradient Descent
- Adaptive learning rate
- Mini-Batching
- Regularization



© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Next topics

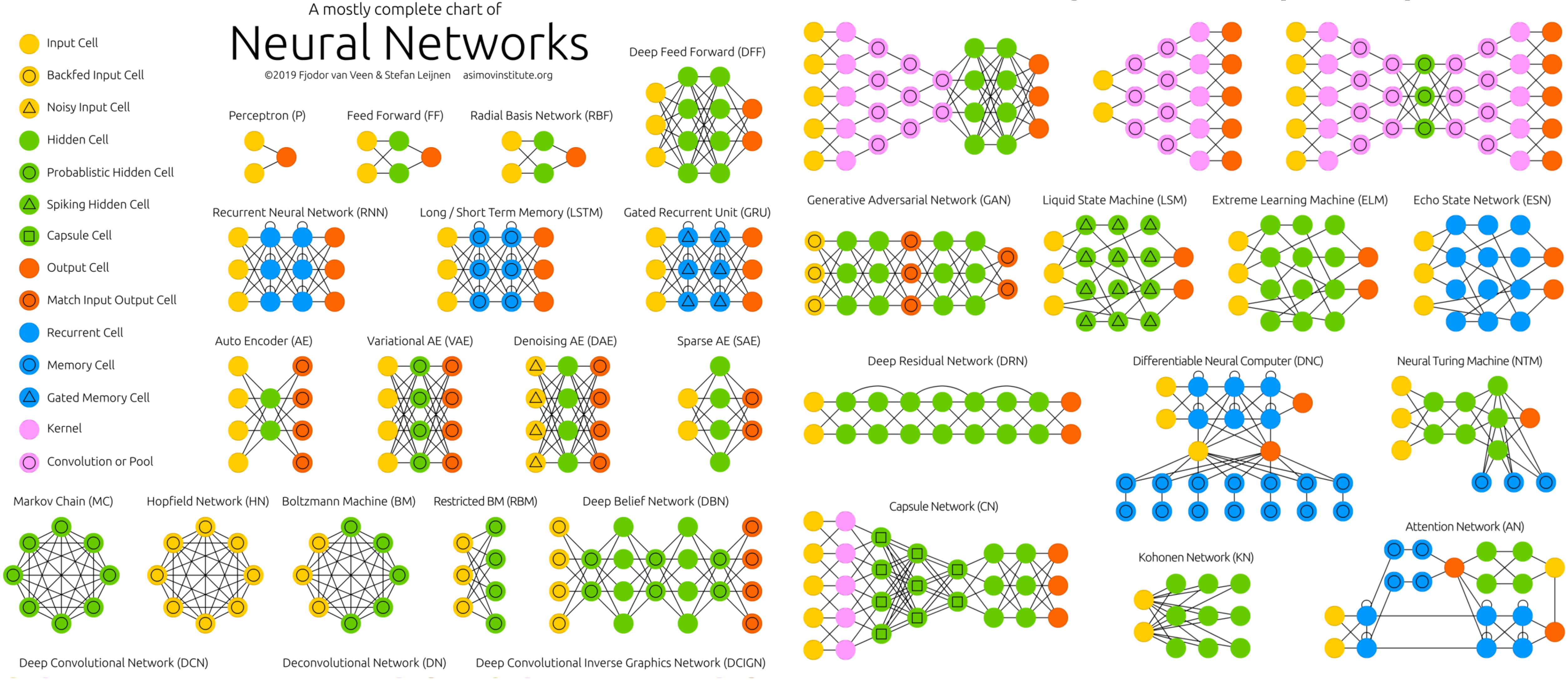
Recurrent neural networks

- Adapted for time series
- Gated Recurrent Unit (GRU),
- Long Short Term Memory (LSTM)

Convolutional neural network (CNN)

- Adapted to computer vision
- AlexNet
- resNet
- Inception v3, v4

The Neural network Zoo, Van Veen, F. & Leijnen, S. ([2019](2019)).



Références

© Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Van Veen, F. & Leijnen, S. (2019). The Neural Network Zoo. Retrieved from https://www.asimovinstitute.org/neural-network-zoo

A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A Survey of the Recent Architectures of Deep Convolutional Neural Networks,” *Artif Intell Rev*, vol. 53, no. 8, pp. 5455–5516, Dec. 2020, doi: 10.1007/s10462-020-09825-6.

"Anatomy and Physiology" by the US National Cancer Institute's Surveillance, Epidemiology and End Results (SEER) Program . Neuron description, Licence CC BY-SA 3.0

H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, “Visualizing the Loss Landscape of Neural Nets,” in *Advances in Neural Information Processing Systems*, 2018, vol. 31, pp. 6389–6399, [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf.

Références

Datasets:

A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. IJCV, 2020.

R. Benenson, S. Popov, and V. Ferrari. Large-scale interactive object segmentation with human annotators. CVPR, 2019.

Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) **ImageNet Large Scale Visual Recognition Challenge**. *IJCV,* 2015

J. Carreira, E. Noland, C. Hillier, and A. Zisserman, *A Short Note on the Kinetics-700 Human Action Dataset*. 2019.

Ambika Choudhury, 10 Open Datasets You Can Use For Computer Vision Projects, available at https://analyticsindiamag.com/10-open-datasets-you-can-use-for-computer-vision-projects/

J. Fritsch, T. Kuehnl, and A. Geiger, “A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms,” 2013.

A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite,” 2012.

M. Menze and A. Geiger, “Object Scene Flow for Autonomous Vehicles,” 2015.

A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets Robotics: The KITTI Dataset,” *International Journal of Robotics Research (IJRR)*, 2013.

Google Research, available at https://research.google/tools/datasets/, access on Nov. 10, 2019