Introduction to Deep Learning

MIT 6.S191

Alexander Amini

January 28, 2019

The Rise of Deep Learning

'Deep Voice' Software Can Clone Anyone's Voice With Just 3.7 Seconds of Audio

Using snippets of voices, Baidu's 'Deep Voice' can generate new speech, accents, and tones.

'Creative' AlphaZero leads way for chess computers and, maybe, science

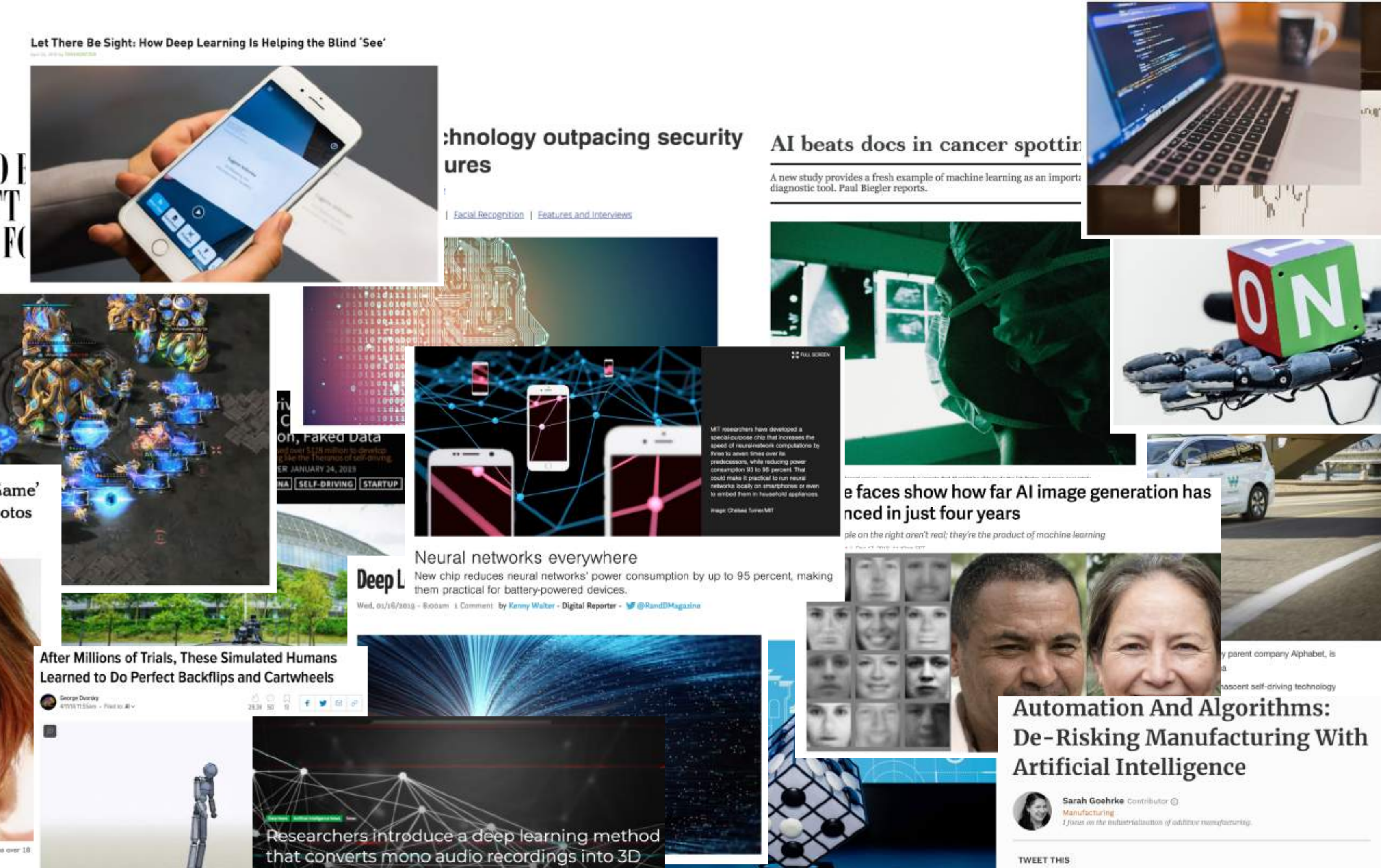
Former chess world champion Garry Kasparov likes what he sees of computer that could be used to find cures for diseases

Stock Predictions Based On AI: Is the Market Truly Predictable?

Complex of bacteria-infecting viral proteins modeled in CASP 13. The complex cont that were modeled individually. PROTEIN DATA BANK

Google's DeepMind aces protein folding

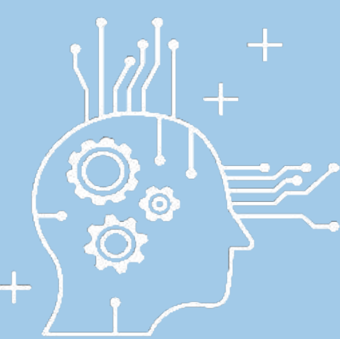
By Robert F. Service | Dec. 6, 2018, 12:05 PM



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

3 1 3 4 7 2
1 7 4 2 3 5

Lecture Schedule

Session	Part 1	Part 2	Lab
1	Introduction to Deep Learning [Slides] [Video] *coming soon!*	Deep Sequence Modeling [Slides] [Video] *coming soon!*	Intro to TensorFlow, Music Generation with RNNs [Code] *coming soon!*
2	Deep Computer Vision [Slides] [Video] *coming soon!*	Deep Generative Models [Slides] [Video] *coming soon!*	De-biasing Facial Recognition Systems [Code] *coming soon!*
3	Deep Reinforcement Learning [Slides] [Video] *coming soon!*	Limitations and New Frontiers [Slides] [Video] *coming soon!*	Model-Free Reinforcement Learning [Code] *coming soon!*
4	Data Visualization for Machine Learning [Info][Slides] [Video] *coming soon!*	Biologically Inspired Learning [Info][Slides] [Video] *coming soon!*	Work time for paper reviews/project proposals
5	Learing and Perception [Info][Slides] [Video] *coming soon!*	Final Project Presentations	Judging and Awards Ceremony

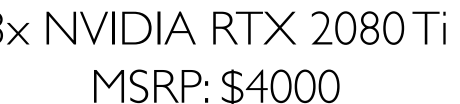
- Mon Jan 28 – Fri Feb 1
- 1:00 pm – 4:00pm
- Lecture + Lab Breakdown
- Graded P/D/F; 3 Units
- 1 Final Assignment

Final Class Project

Option 1: Proposal Presentation

- Groups of 3 or 4
- Present a novel deep learning research idea or application
- 3 minutes (strict)
- List of example proposals on website: introtodeeplearning.com
- Presentations on **Friday, Feb 1**
- Submit groups by **Wednesday 5pm** to be eligible
- Submit slide by **Thursday 9pm** to be eligible

- Judged by a panel of industry judges
- Top winners are awarded:


3x NVIDIA RTX 2080 Ti
MSRP: $4000

4x Google Home
MSRP: $400

Final Class Project

Option 1: Proposal Presentation

- Groups of 3 or 4
- Present a novel deep learning research idea or application
- 3 minutes (strict)
- List of example proposals on website: introtodeeplearning.com
- Presentations on **Friday, Feb 2**
- Submit groups by **Wednesday 5pm** to be eligible
- Submit slide by **Thursday 9pm** to be eligible

Option 2: Write a 1-page review of a deep learning paper

- Grade is based on clarity of writing and technical communication of main ideas
- Due **Friday 1:00pm** (before lecture)

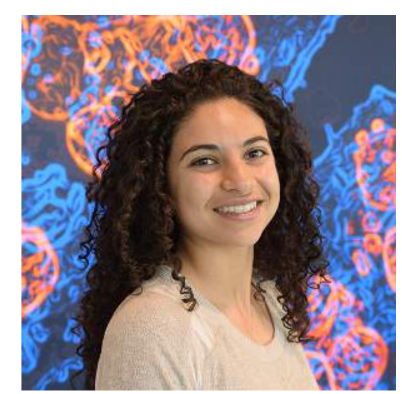
Class Support

- Piazza: http://piazza.com/mit/spring2019/6s191
  - Useful for discussing labs
- Course Website: http://introtodeeplearning.com
  - Lecture schedule
  - Slides and lecture recordings
  - Software labs
  - Grading policy
- Email us: introtodeeplearning-staff@mit.edu
- Office Hours by request

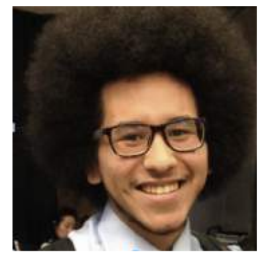
Course Staff

Alexander Amini
Lead Organizer

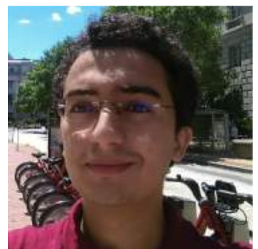
Ava Soleimany
Lead Organizer



Thomas


Mauri

Harini

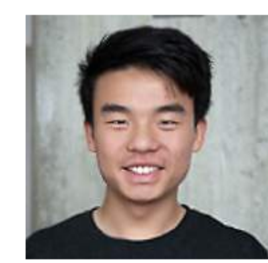

Houssam

Julia

Felix

Jacob

Rohil


Gilbert

\+ Ravi A.

introtodeeplearning-staff@mit.edu

Thanks to Sponsors!

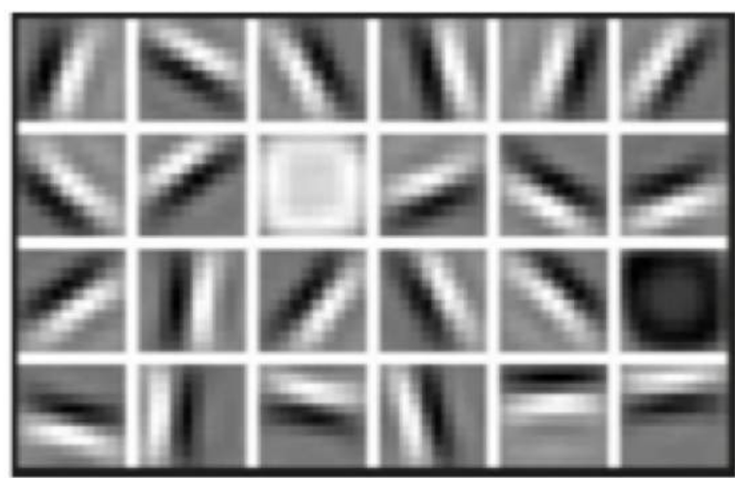
Why Deep Learning and Why Now?

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

Why Now?

Neural Networks date back decades, so why the resurgence?

1952	Stochastic Gradient Descent
1958	Perceptron • Learnable Weights
⋮	
1986	Backpropagation • Multi-Layer Perceptron
1995	Deep Convolutional NN • Digit Recognition
⋮	

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable

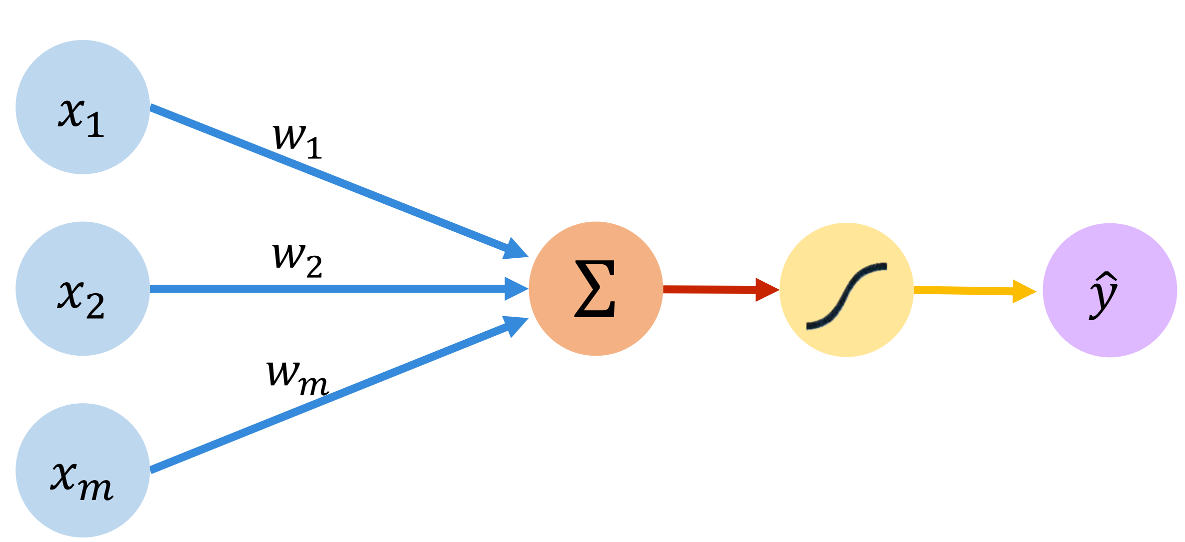
3. Software

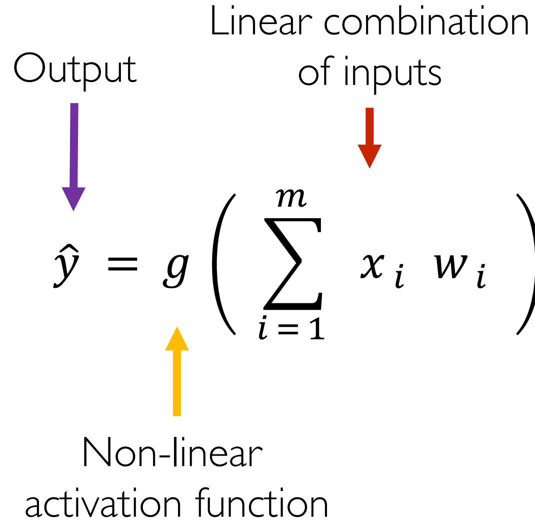
- Improved Techniques
- New Models
- Toolboxes

The Perceptron

The structural building block of deep learning

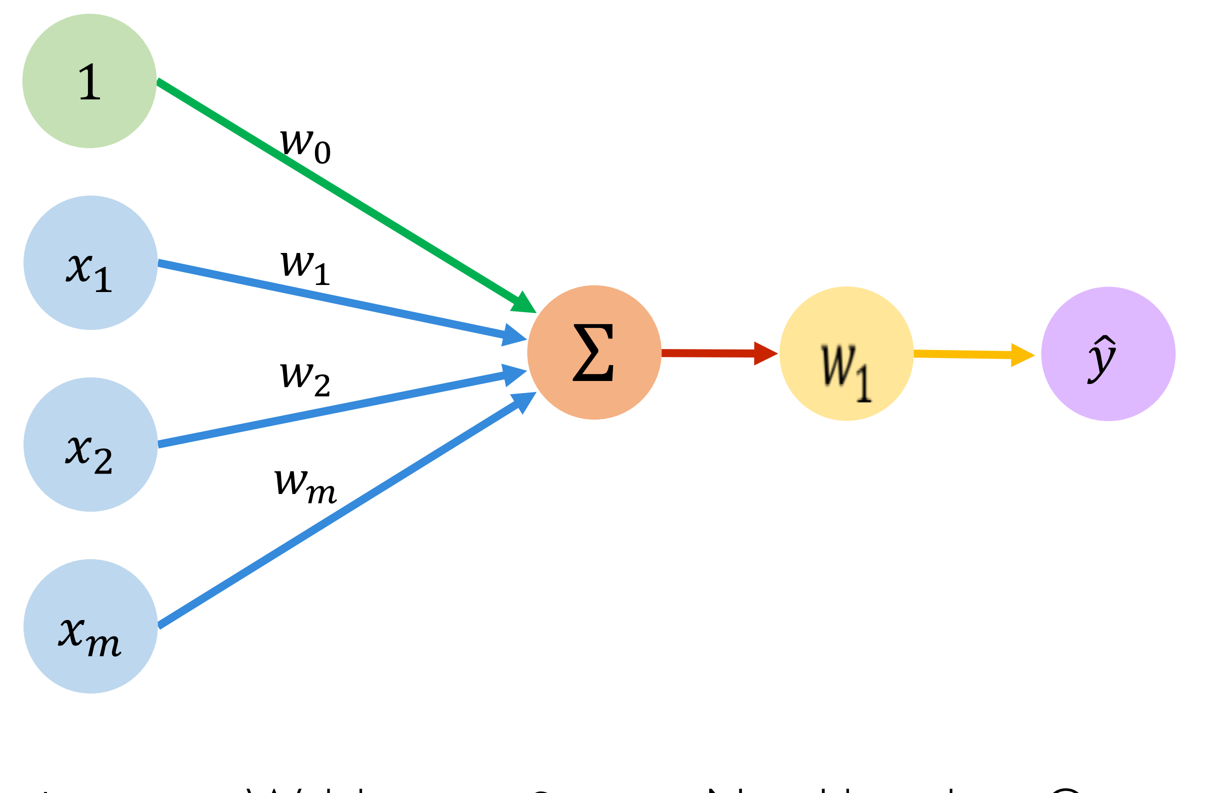
The Perceptron: Forward Propagation

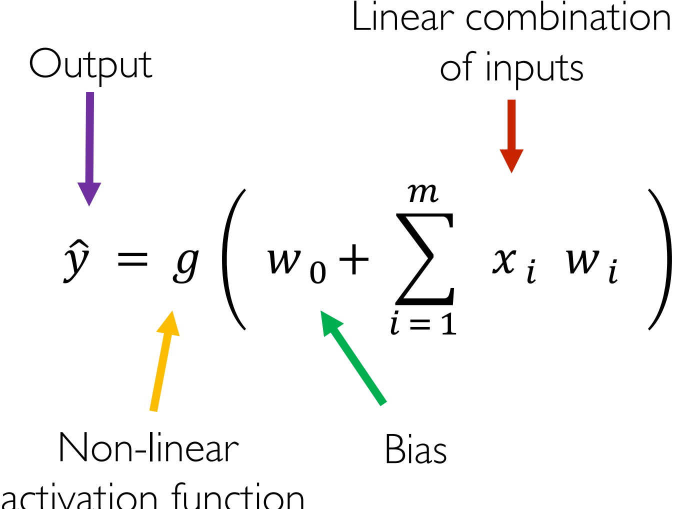




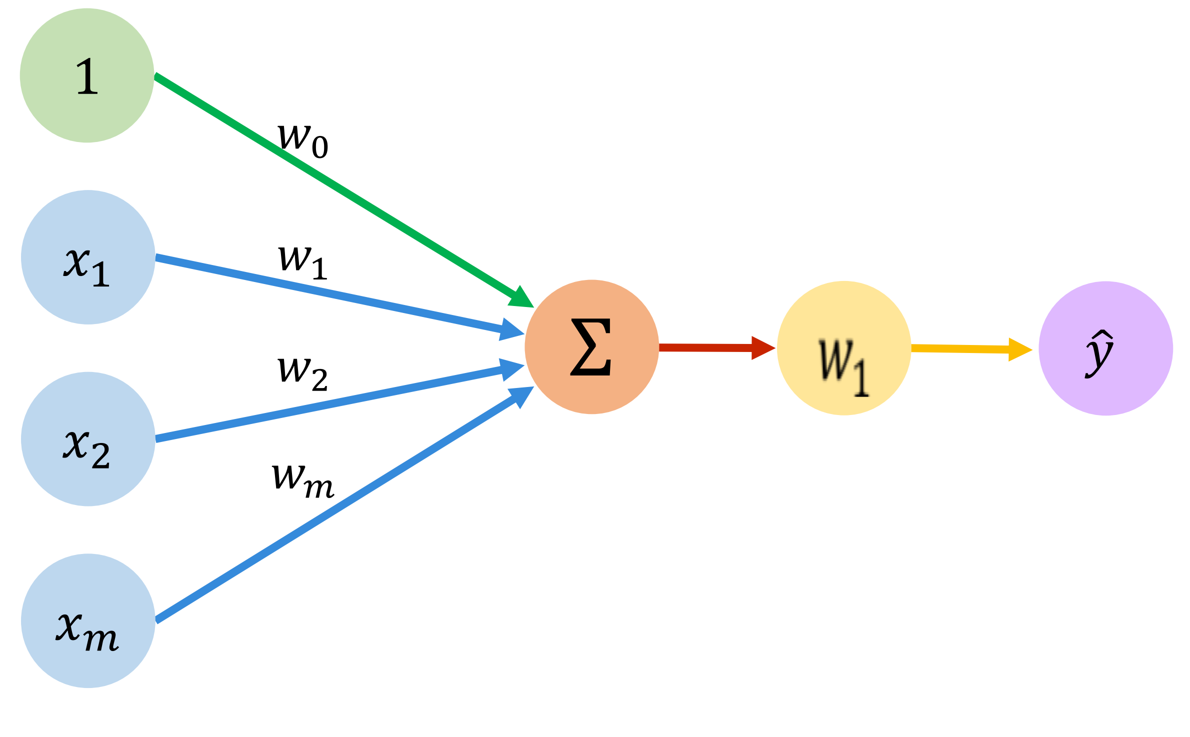
Inputs Weights Sum Non-Linearity Output

The Perceptron: Forward Propagation





The Perceptron: Forward Propagation

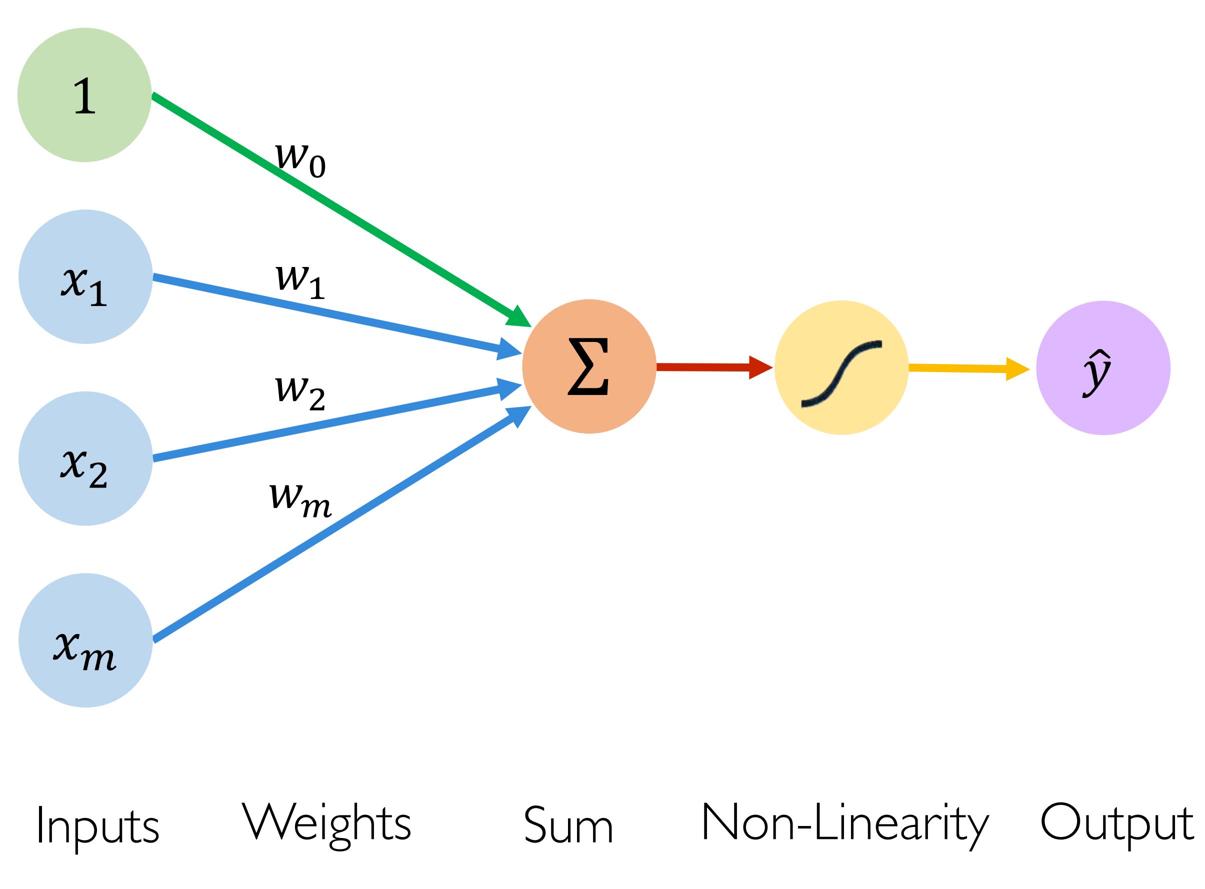


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

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

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

The Perceptron: Forward Propagation

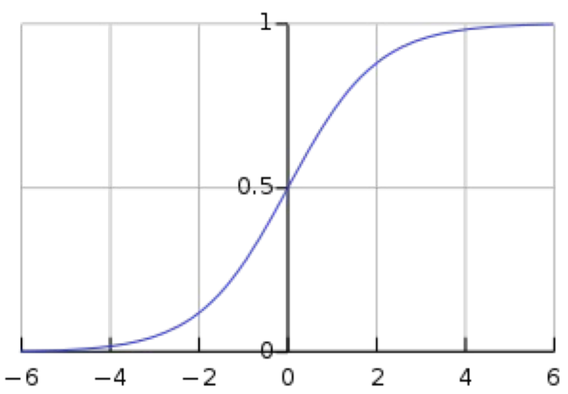


Activation Functions

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

- Example: sigmoid function

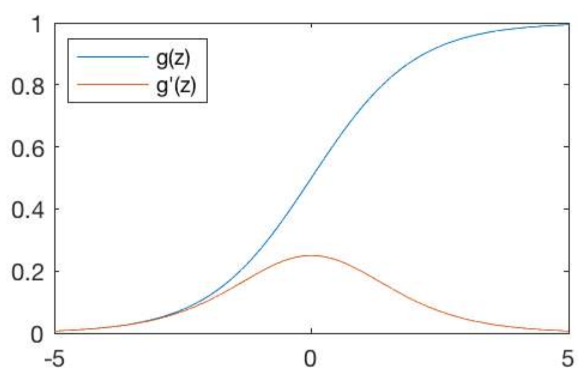
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



z

Common Activation Functions

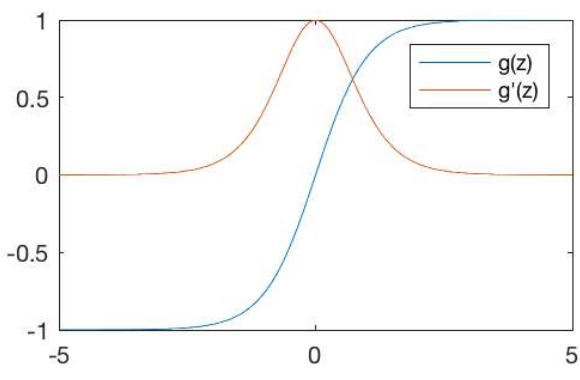
Sigmoid Function



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

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

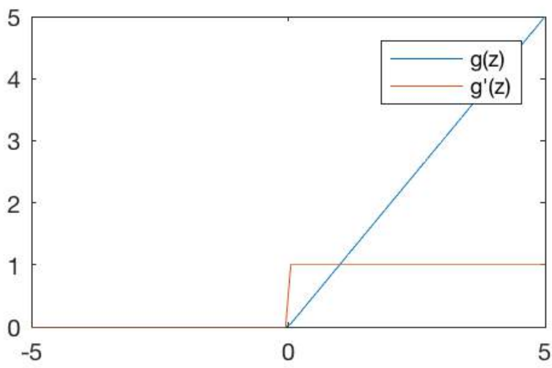
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)



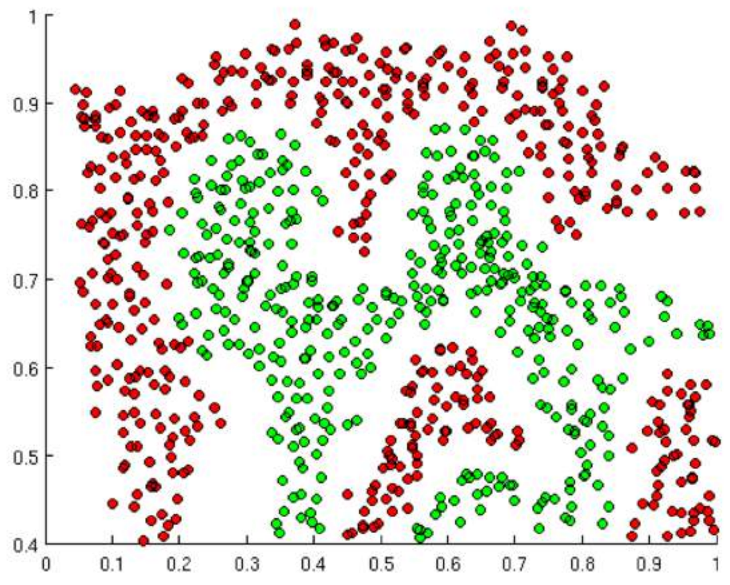
$$g(z) = \max(0, z)$$

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

NOTE: All activation functions are non-linear

Importance of Activation Functions

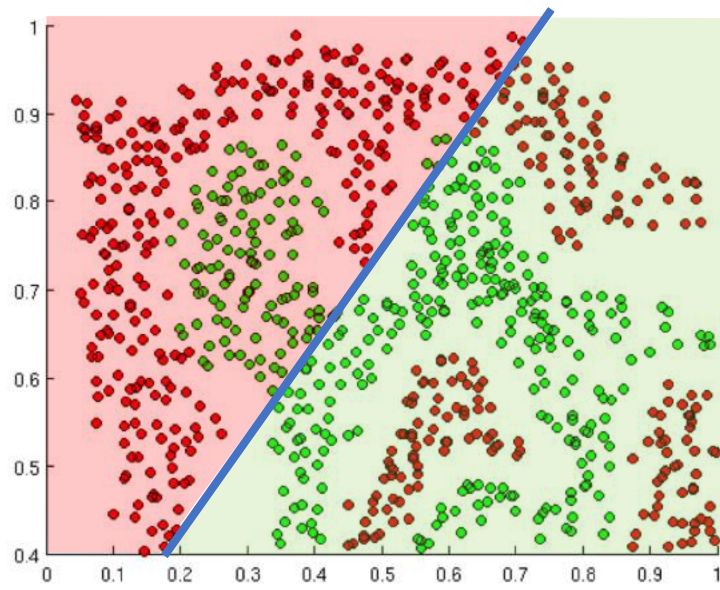
The purpose of activation functions is to ***introduce non-linearities*** *into the network*



What if we wanted to build a Neural Network to distinguish green vs red points?

Importance of Activation Functions

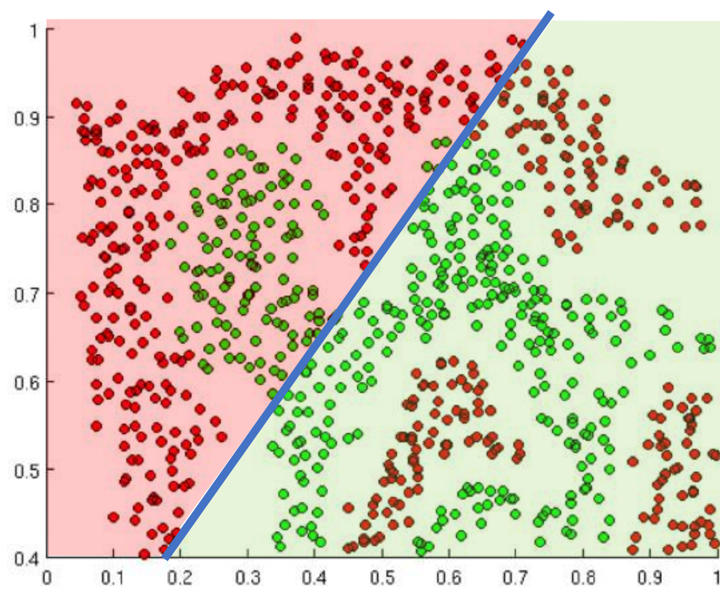
*The purpose of activation functions is to **introduce non-linearities** into the network*



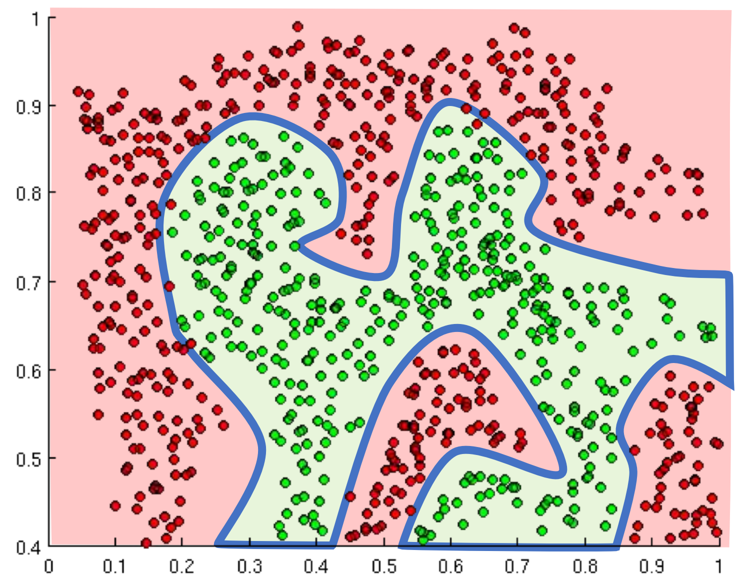
Linear Activation functions produce linear decisions no matter the network size

Importance of Activation Functions

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

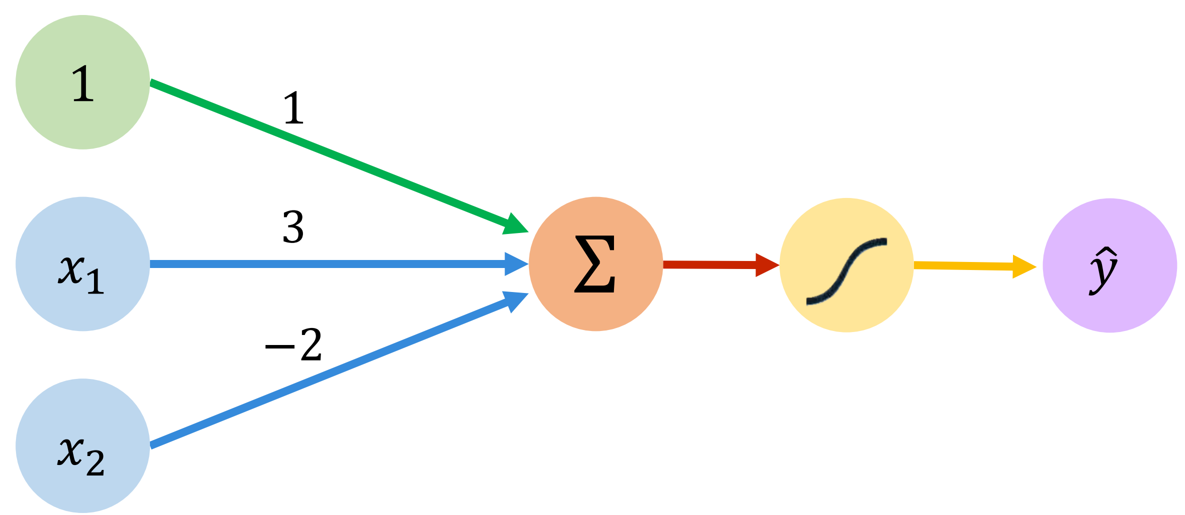


Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

The Perceptron: Example

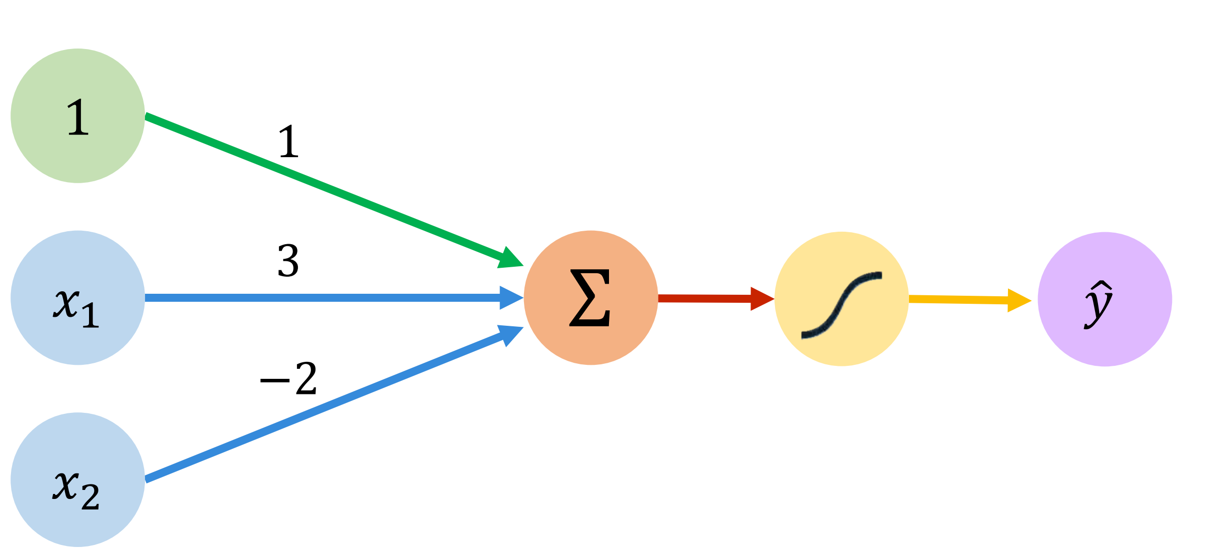


We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

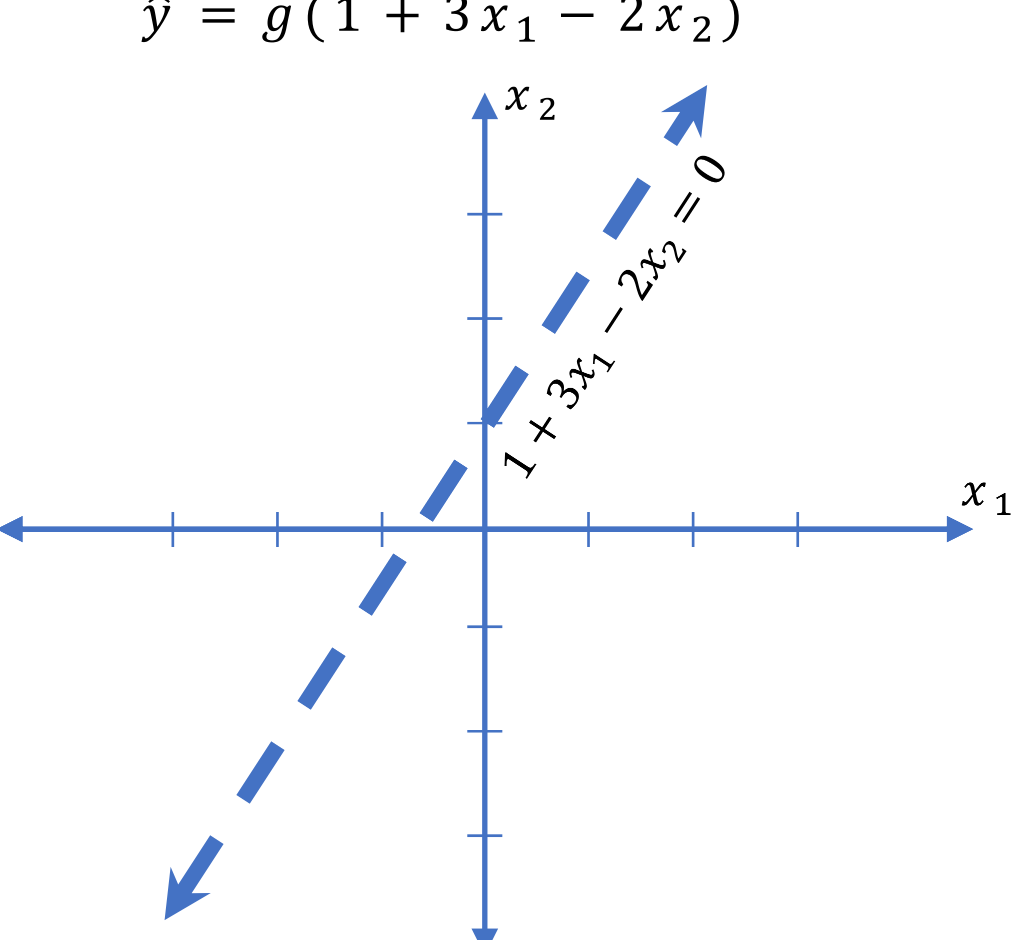
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$
$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$

This is just a line in 2D!

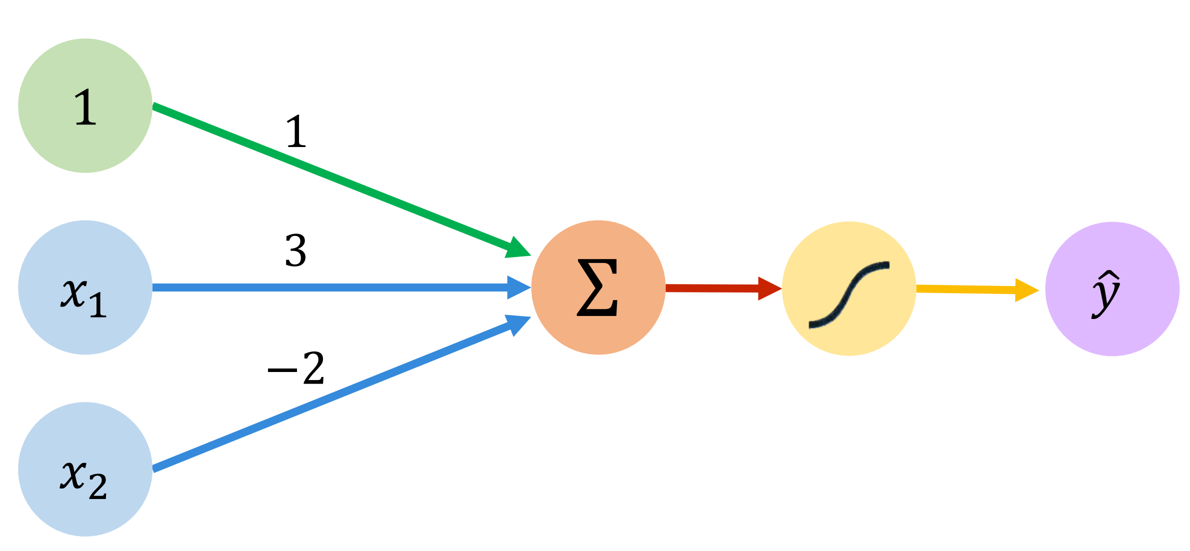
The Perceptron: Example



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



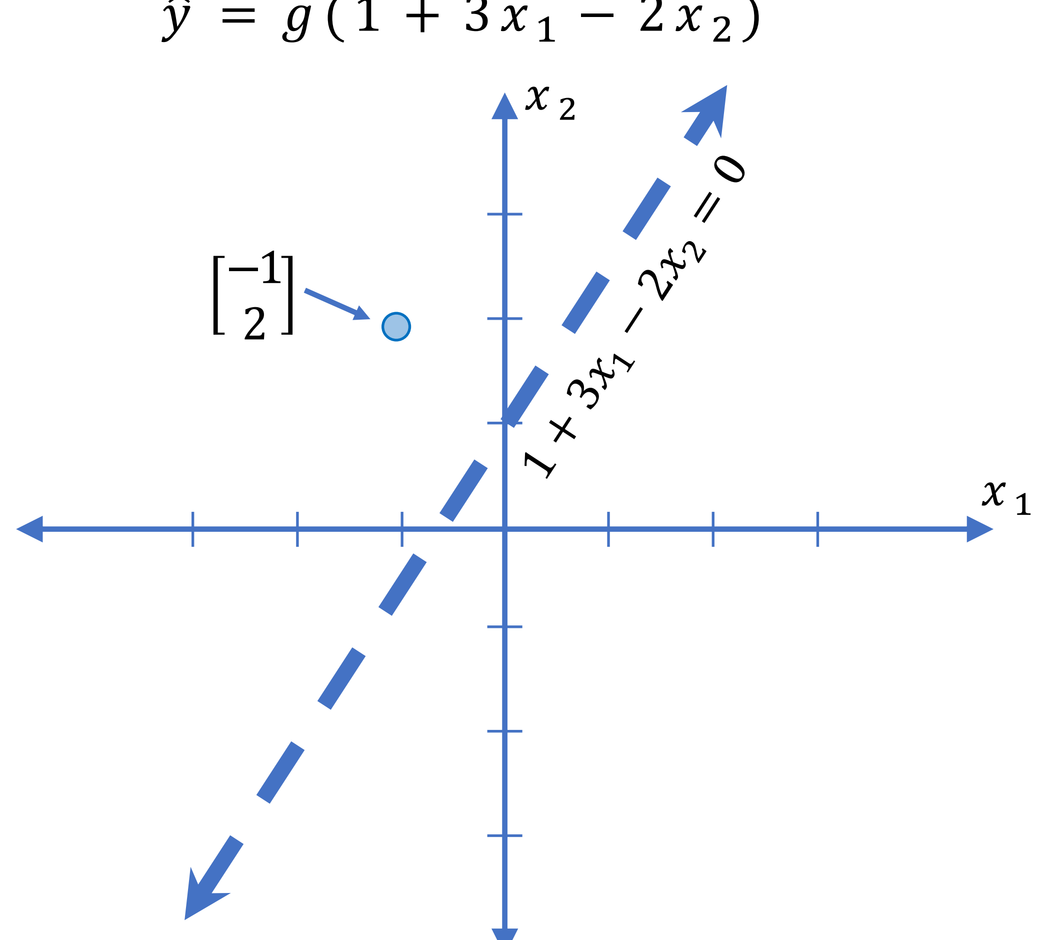
The Perceptron: Example



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

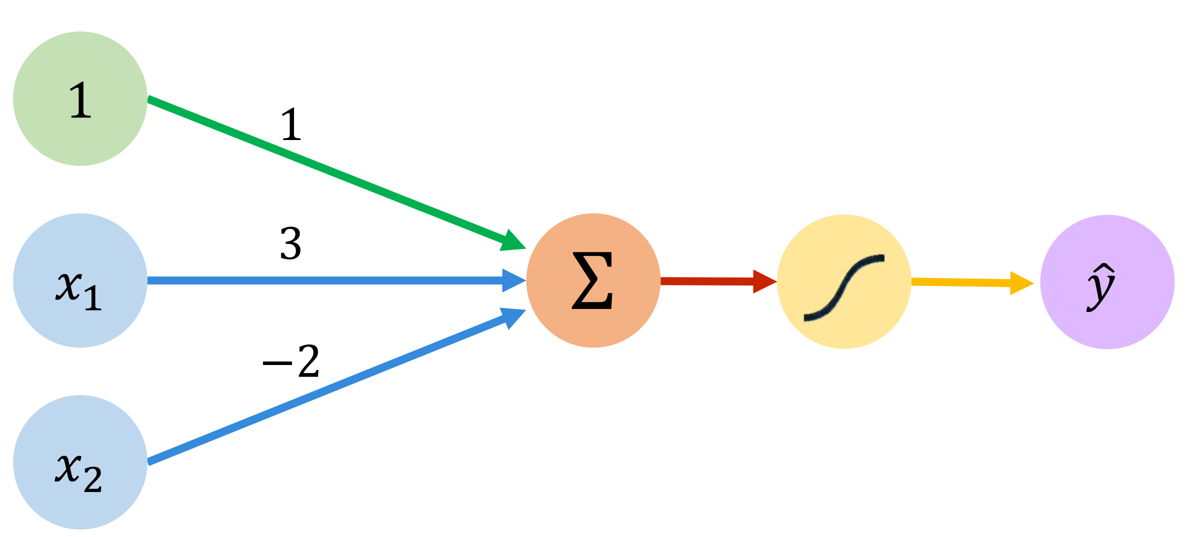
$$\hat{y} = g\big(1 + (3 * -1) - (2 * 2)\big)$$
$$= g(-6) \approx 0.002$$

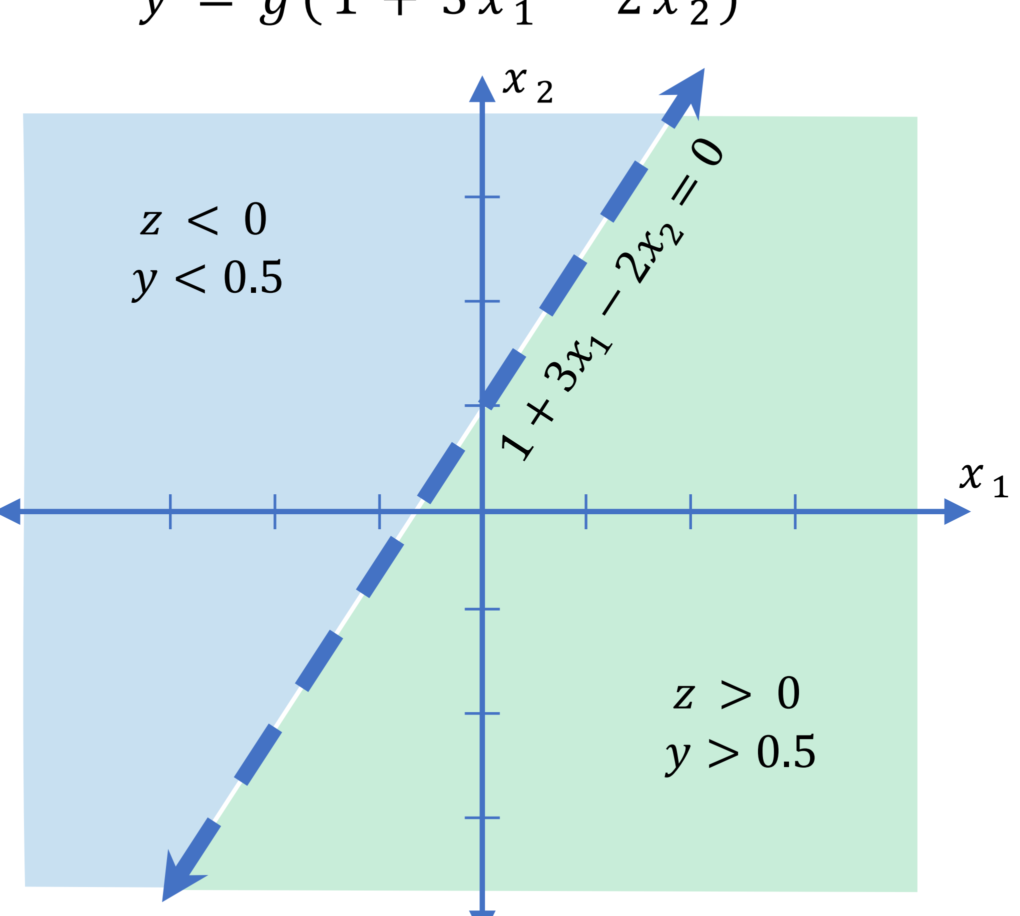
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



The Perceptron: Example

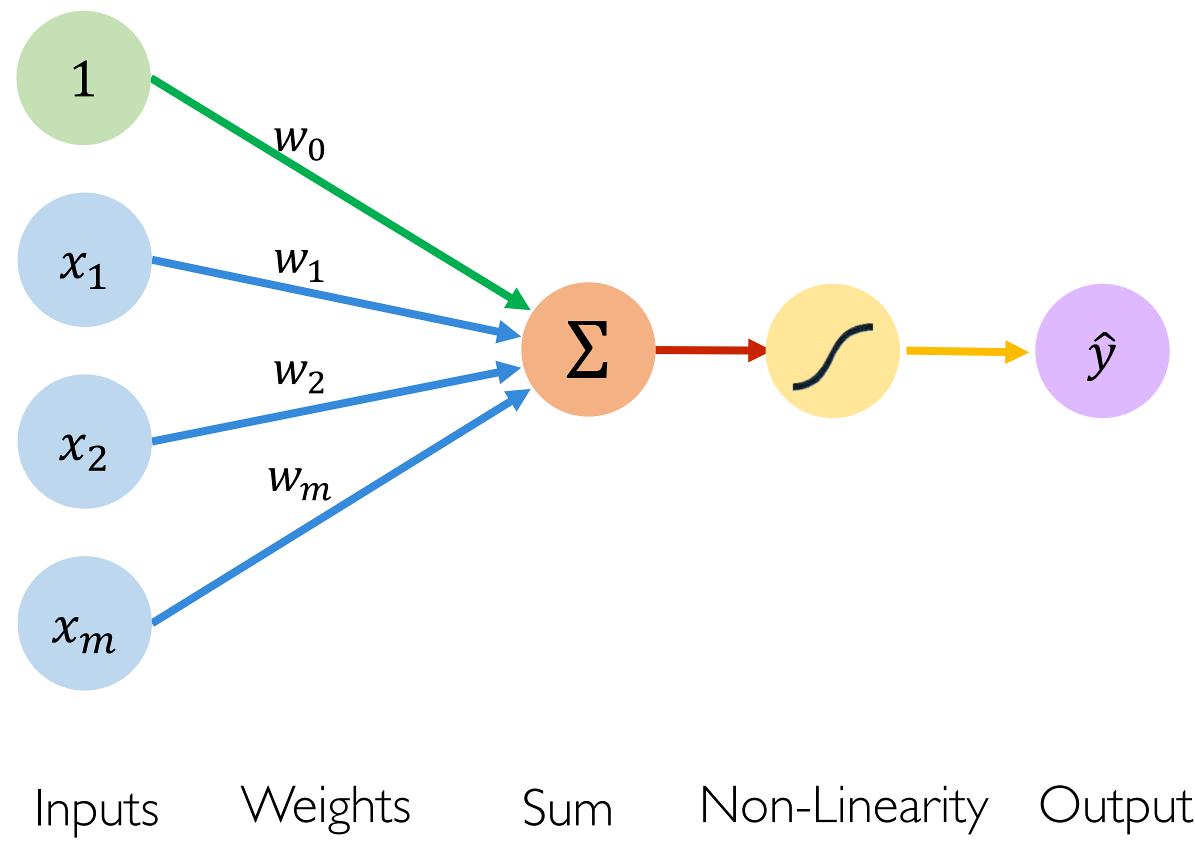
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



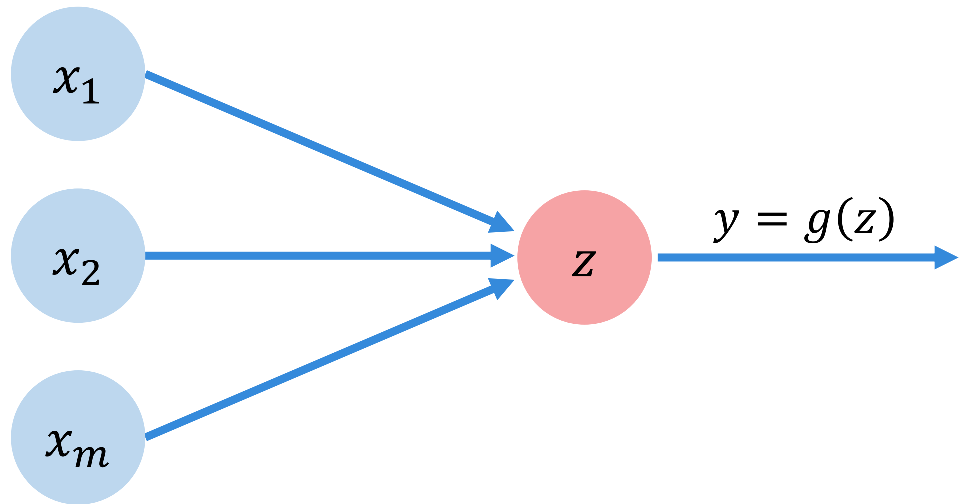


Building Neural Networks with Perceptrons

The Perceptron: Simplified

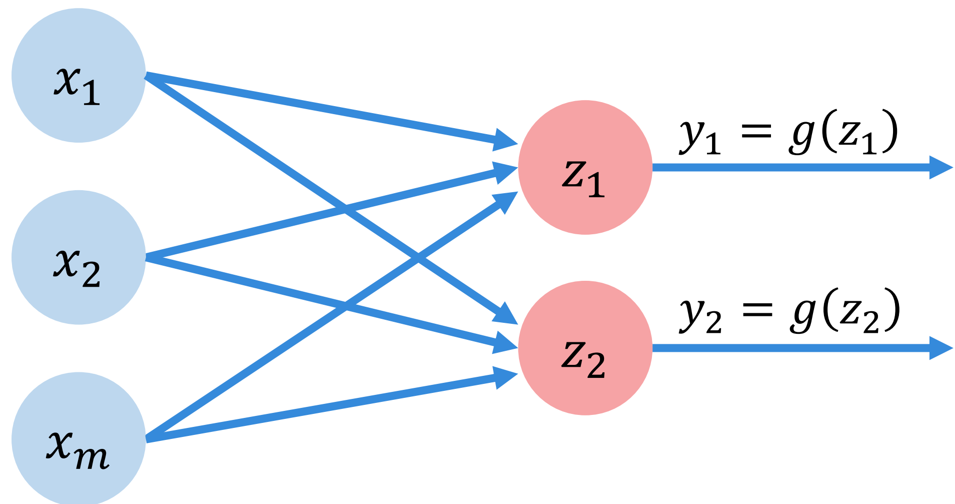


The Perceptron: Simplified



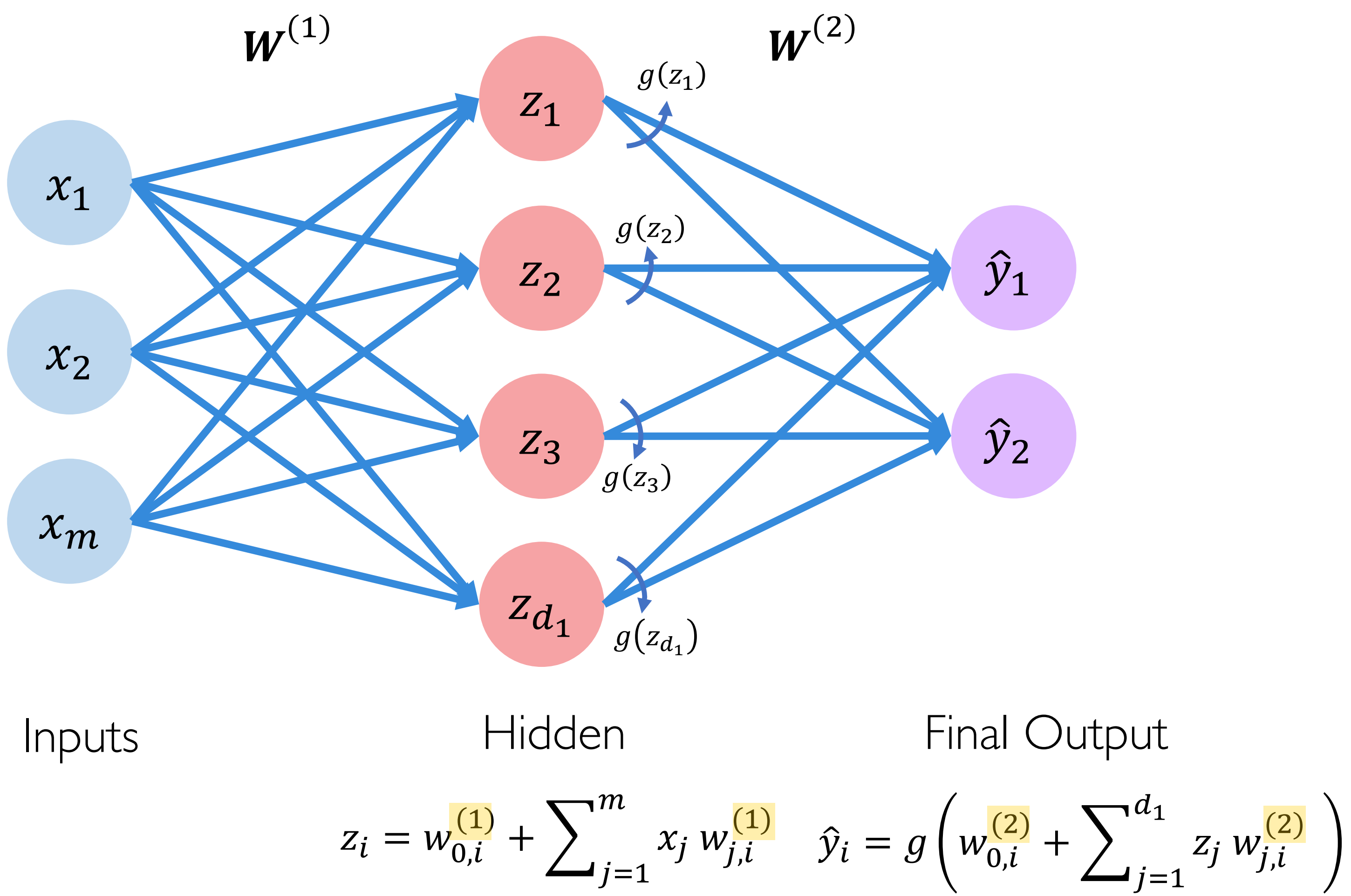
$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

Multi Output Perceptron

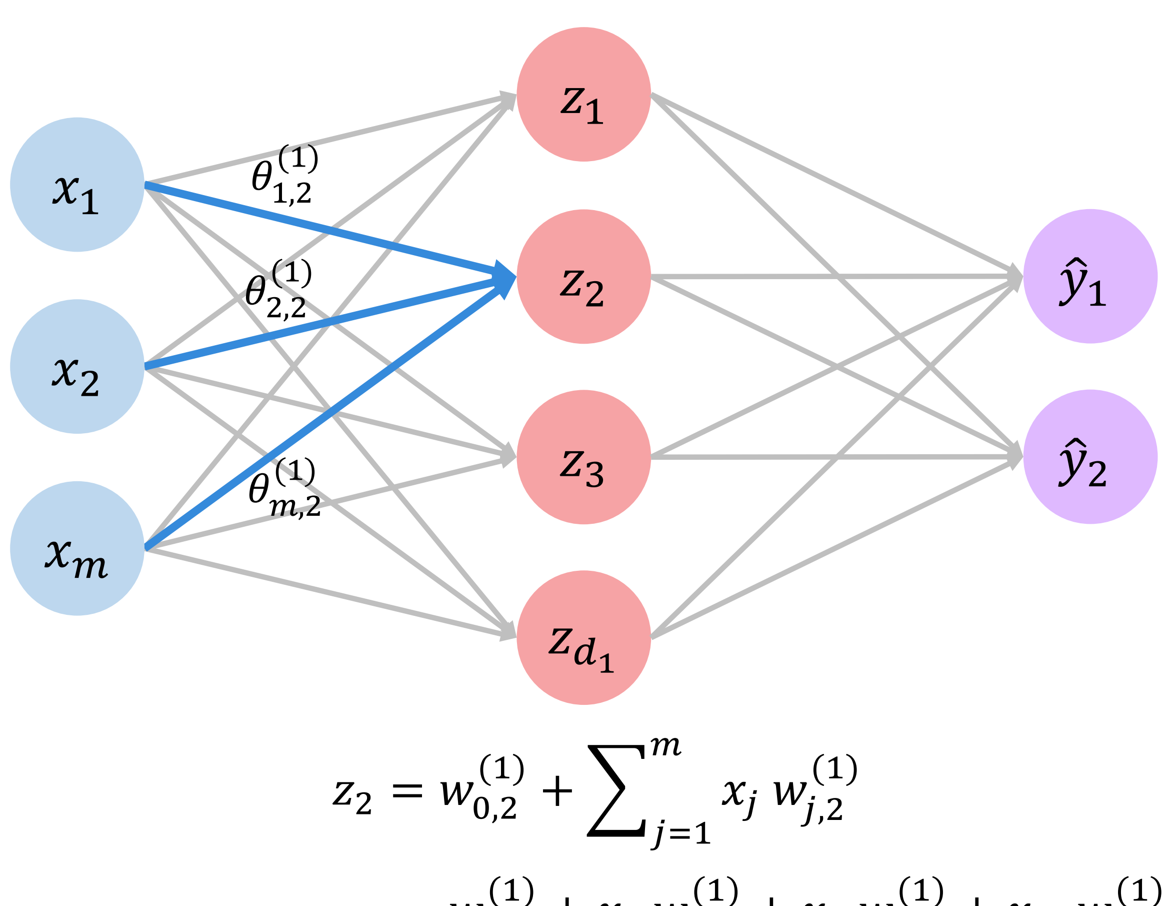


$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Single Layer Neural Network



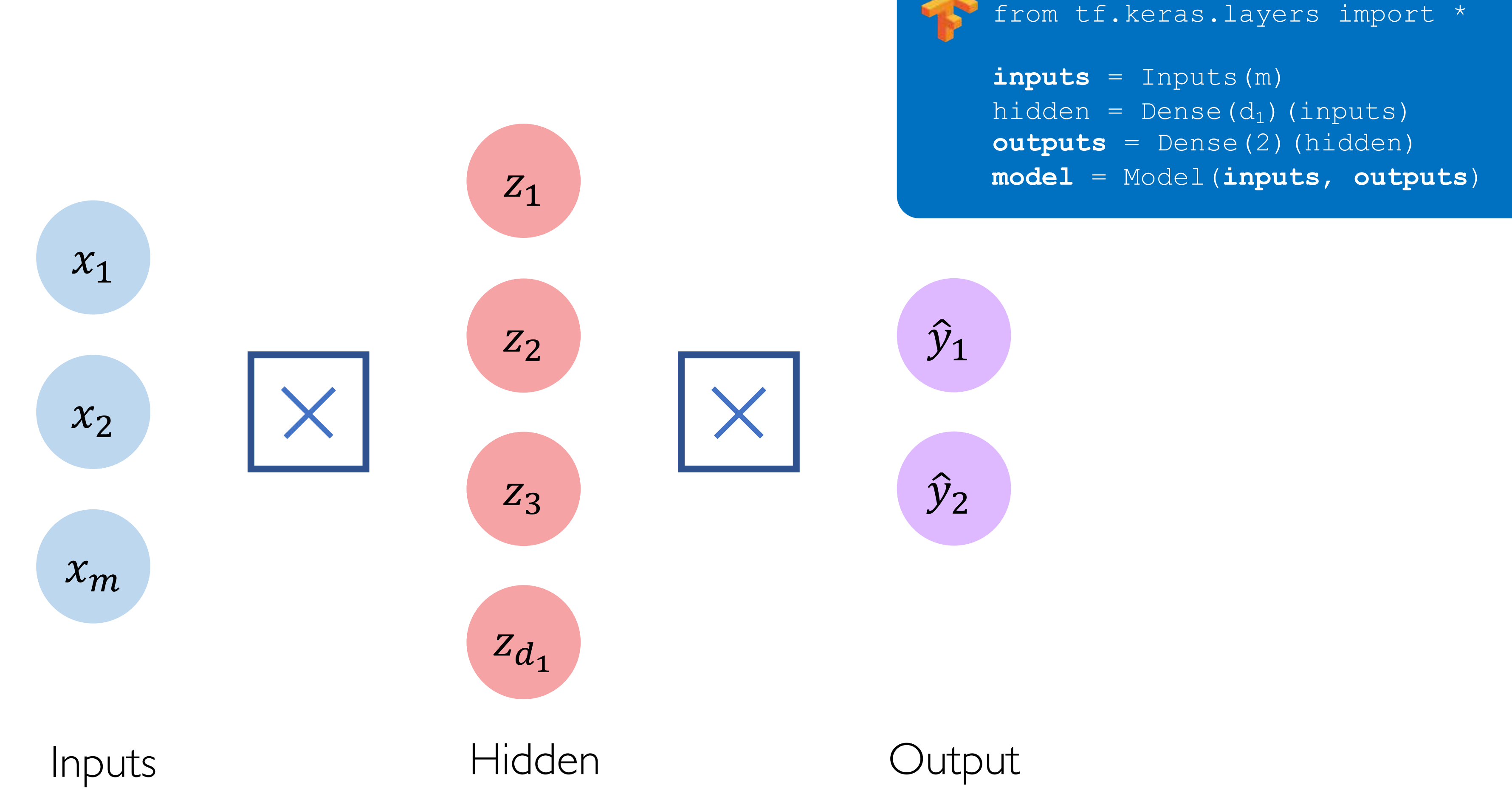
Single Layer Neural Network



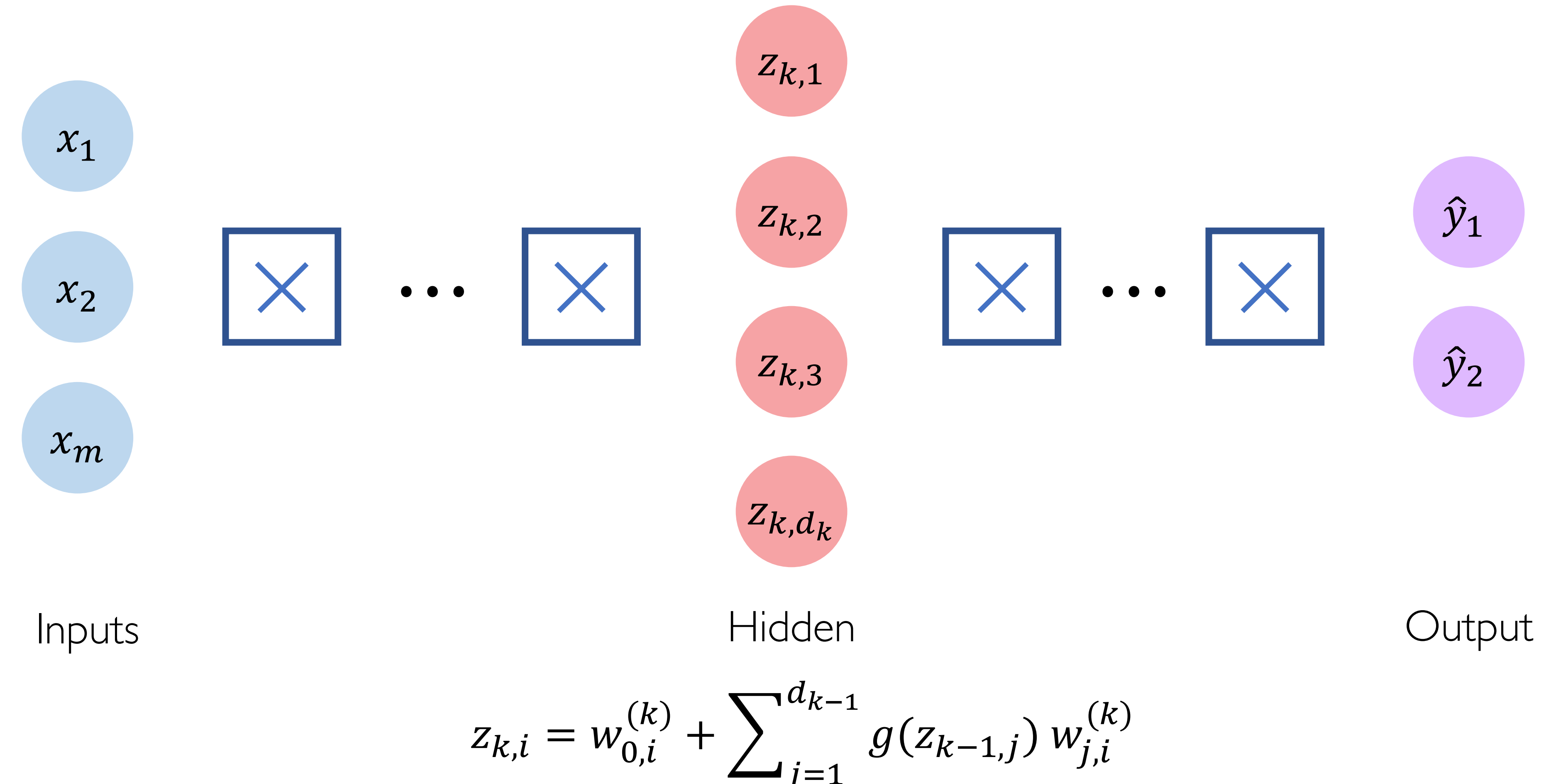
$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

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

Multi Output Perceptron



Deep Neural Network



Applying Neural Networks

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

Example Problem: Will I pass this class?

x_2 = Hours spent on the final project

Legend

Pass

Fail

x_1 = Number of lectures you attend

Example Problem: Will I pass this class?

x_2 = Hours spent on the final project

?

$\begin{bmatrix} 4 \\ 5 \end{bmatrix}$

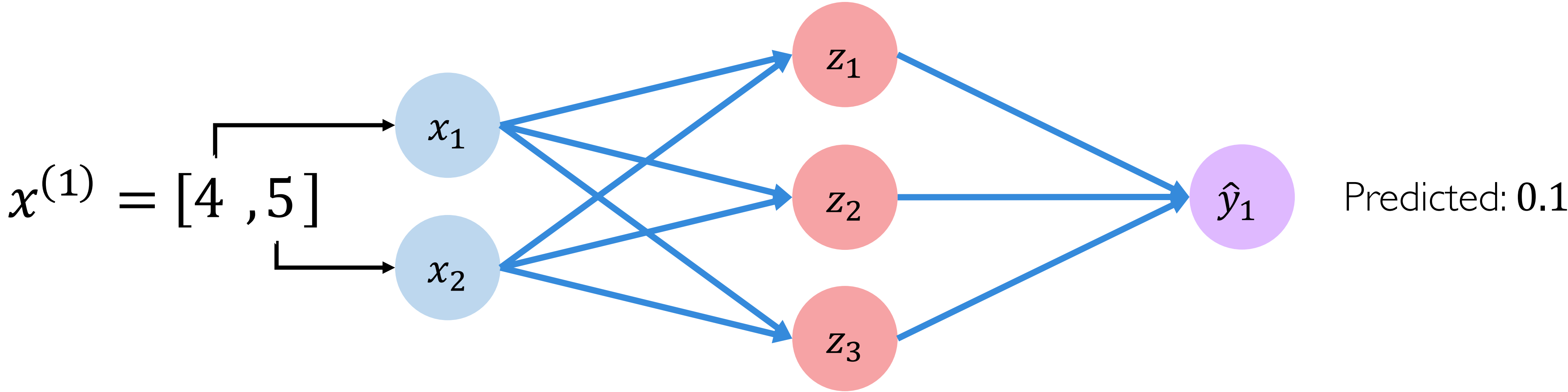
Legend

Pass

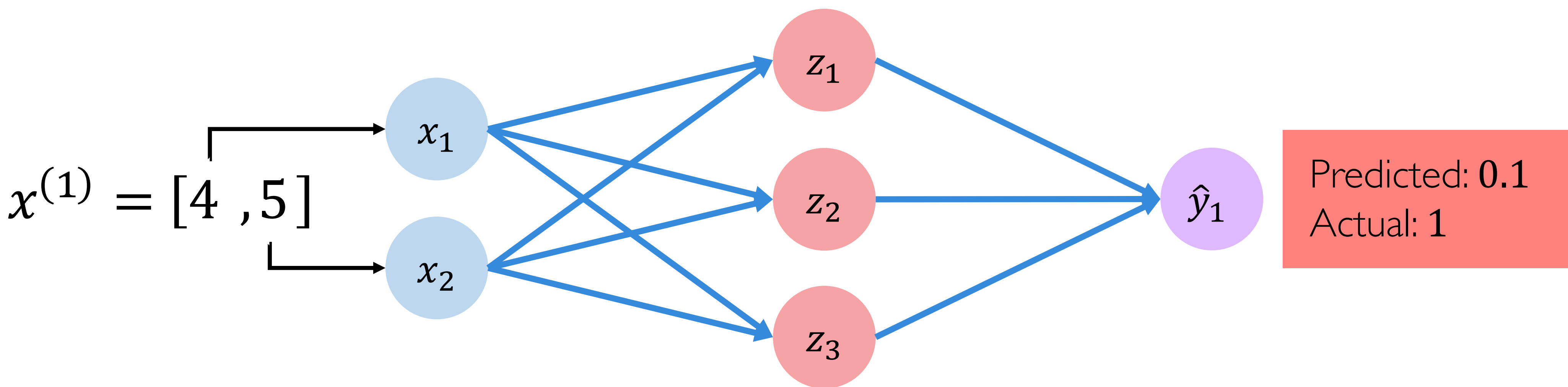
Fail

x_1 = Number of lectures you attend

Example Problem: Will I pass this class?

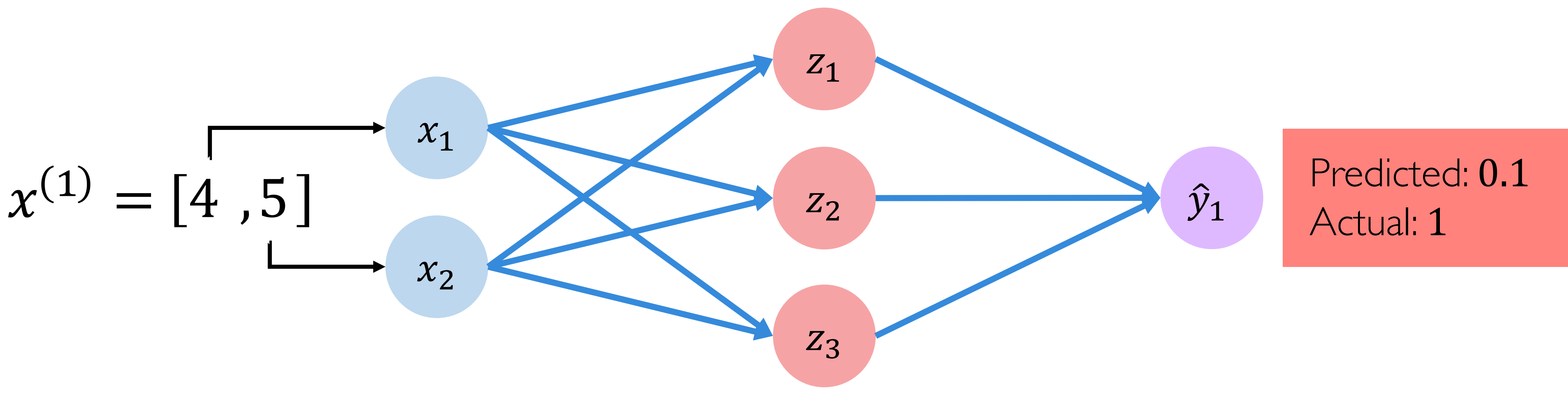


Example Problem: Will I pass this class?



Quantifying Loss

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



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

Predicted Actual

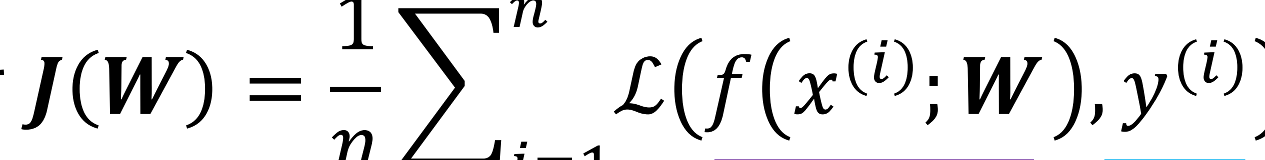
Empirical Loss

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

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

x_1, x_2 → z_1, z_2, z_3 → $\hat{y}_1$

$f(x)$: $\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix}$ y: $\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$



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

Predicted Actual

Also known as:
- Objective function
- Cost function
- Empirical Risk

Binary Cross Entropy Loss

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

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

x_1, x_2 → z_1, z_2, z_3 → $\hat{y}_1$

$f(x)$	y
0.1	1
0.8	0
0.6	1
⋮	⋮

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

Actual   Predicted   Actual   Predicted

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

Mean Squared Error Loss

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

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

x_1, x_2 → z_1, z_2, z_3 → $\hat{y}_1$

$f(x)$	y
30	90
80	20
85	95
⋮	⋮

Final Grades (percentage)

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

Actual Predicted

```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred) )
```

Training Neural Networks

Loss Optimization

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

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

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

Loss Optimization

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

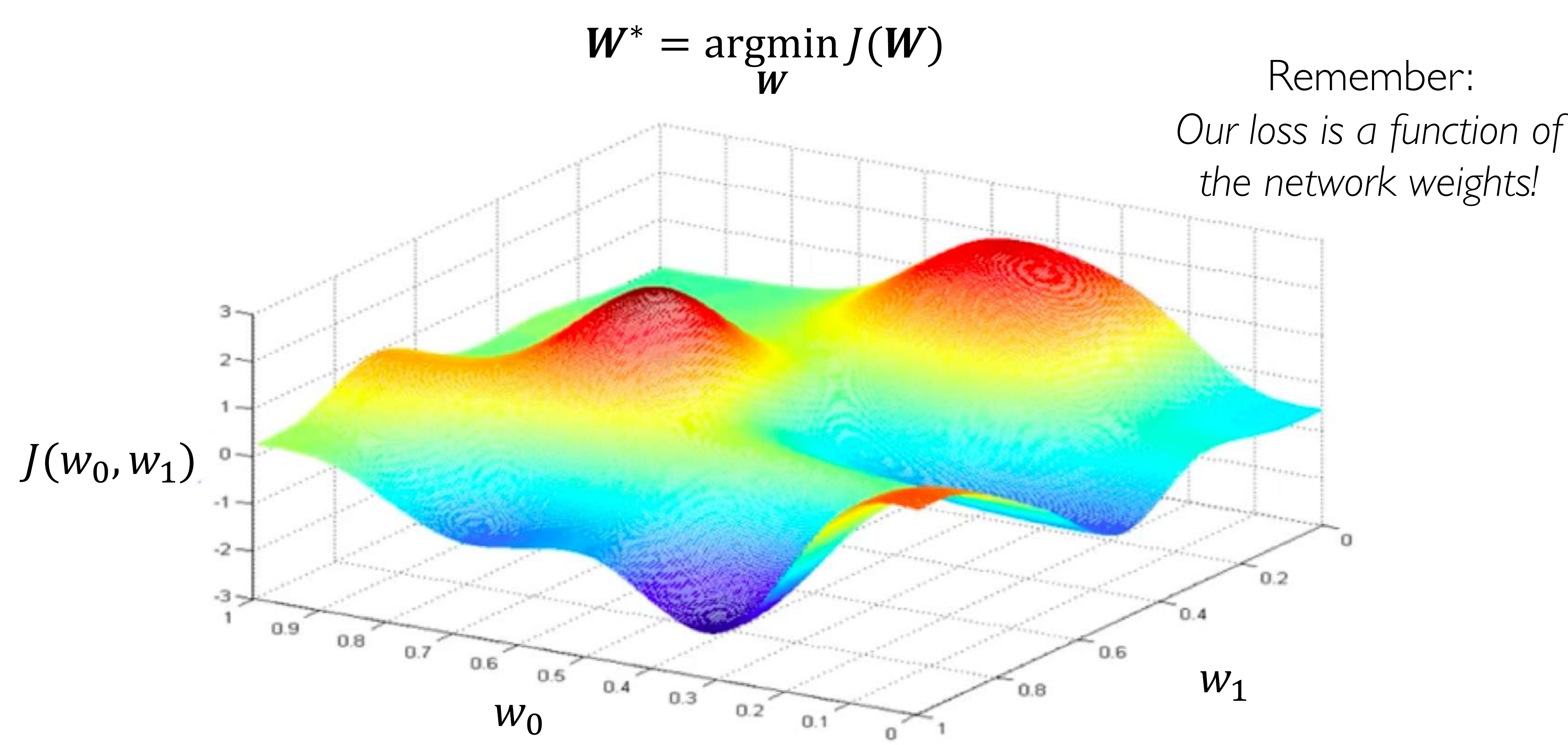
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

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

Remember:

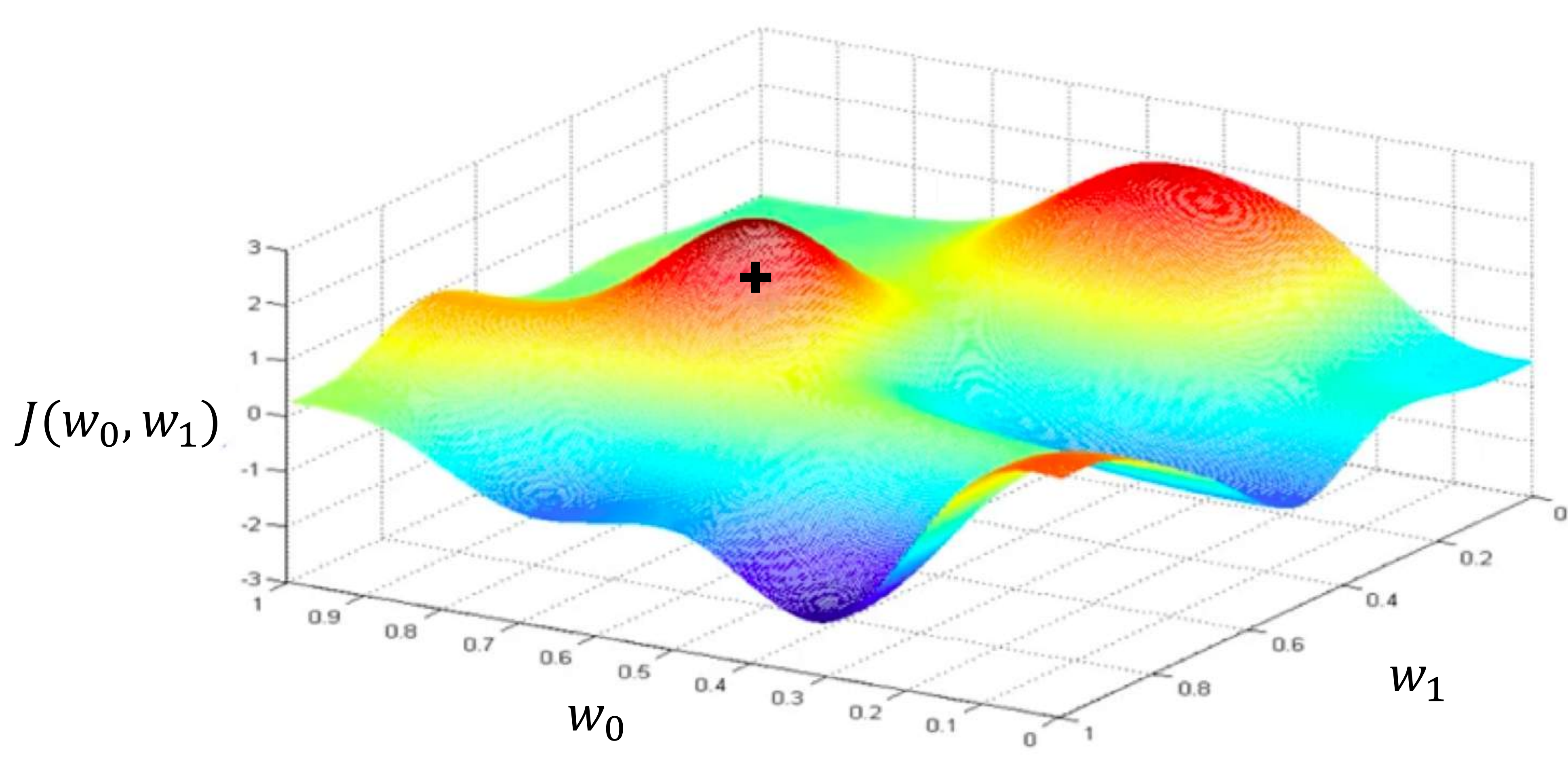
$$\boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\}$$

Loss Optimization



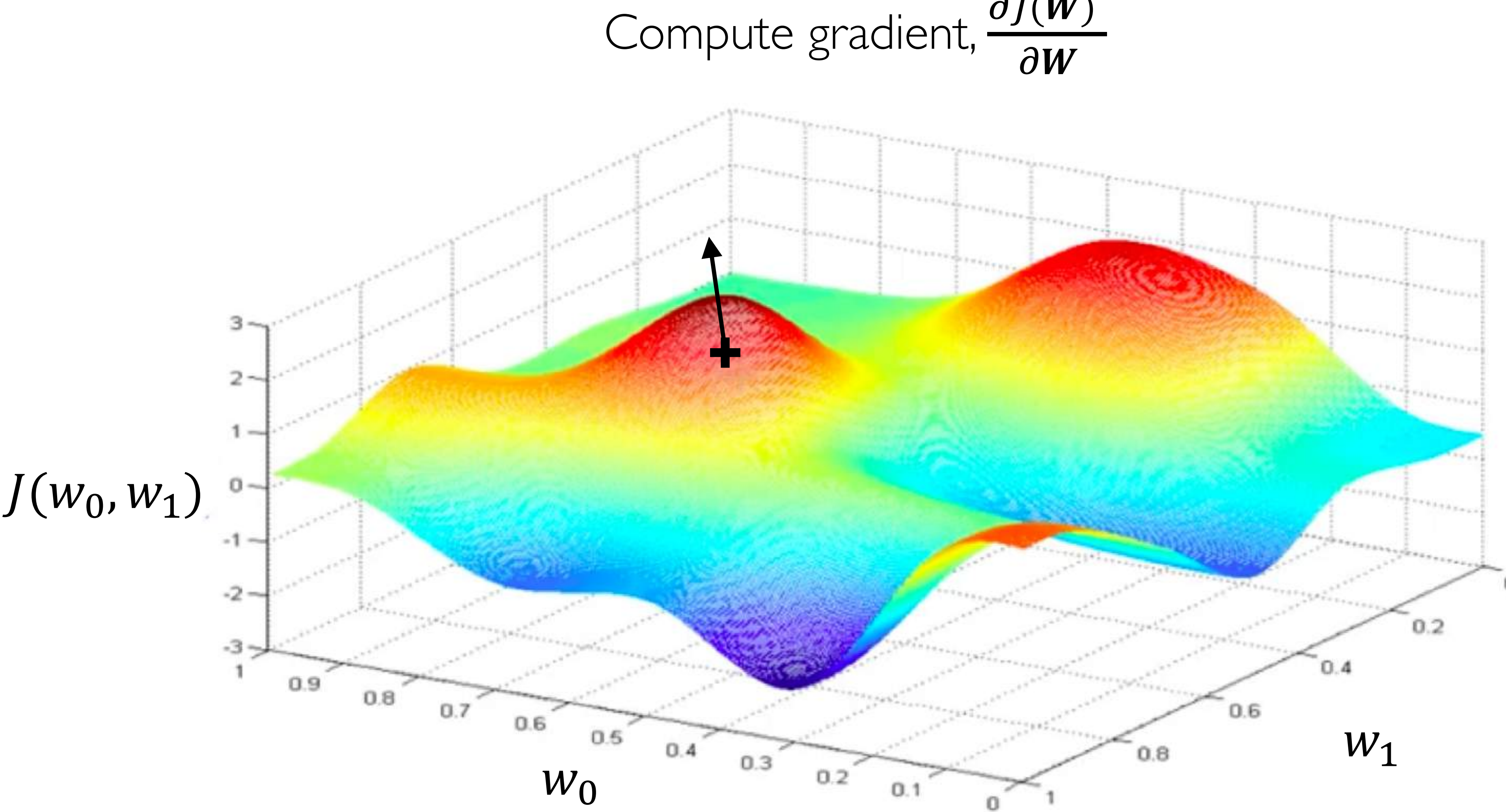
Loss Optimization

Randomly pick an initial (w_0, w_1)



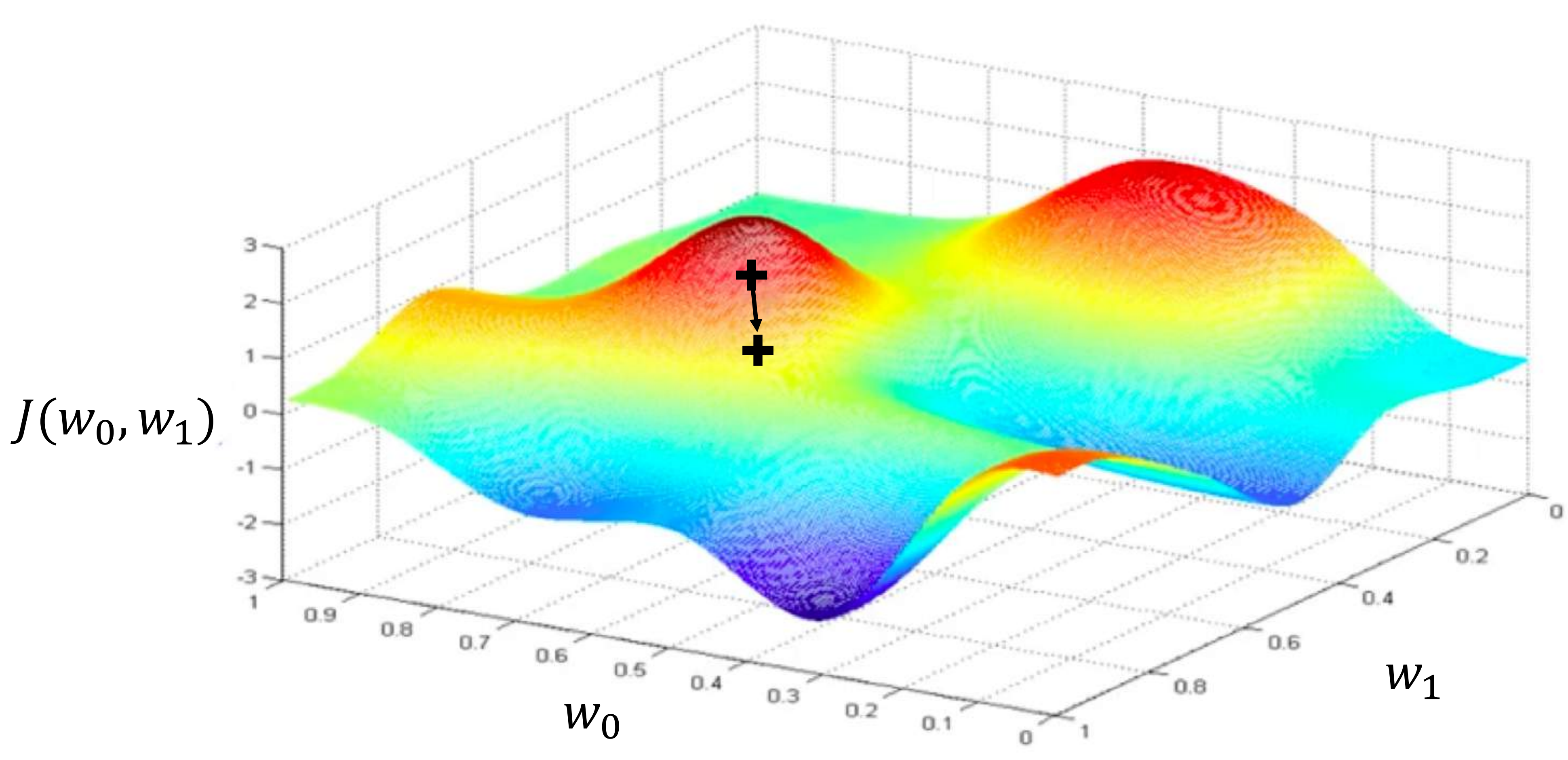
Loss Optimization

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



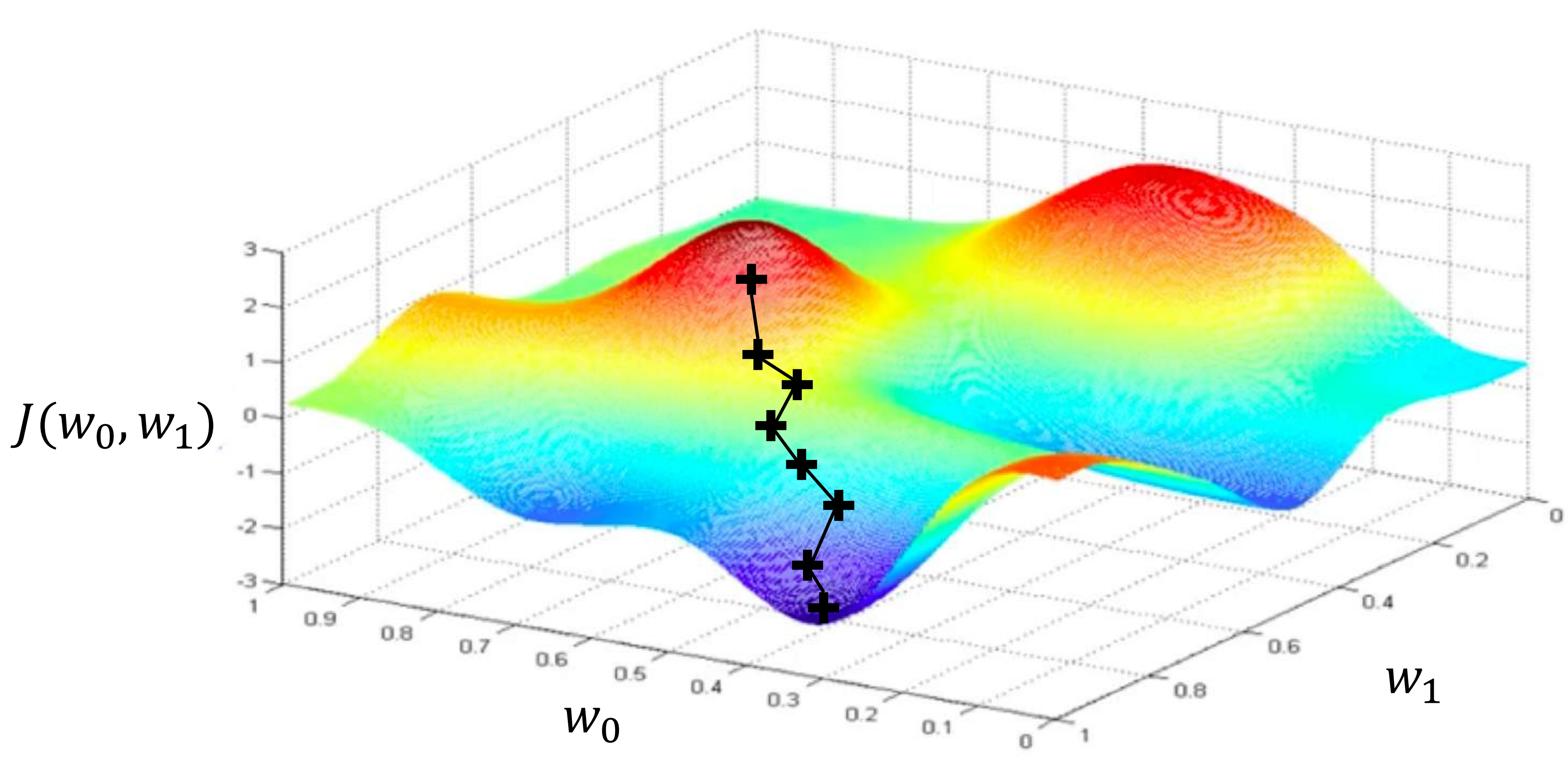
Loss Optimization

Take small step in opposite direction of gradient



Gradient Descent

Repeat until convergence



Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$

2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

```
grads = tf.gradients(ys=loss, xs=weights)
```

4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

```
weights_new = weights.assign(weights - lr * grads)
```

5. Return weights

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$

2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

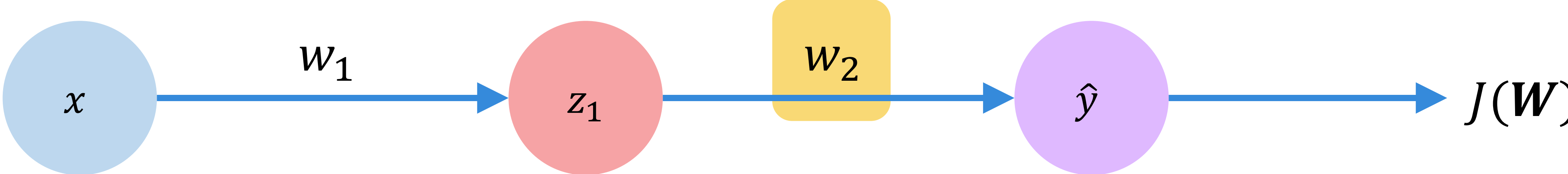
```
grads = tf.gradients(ys=loss, xs=weights)
```

4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

```
weights_new = weights.assign(weights - lr * grads)
```

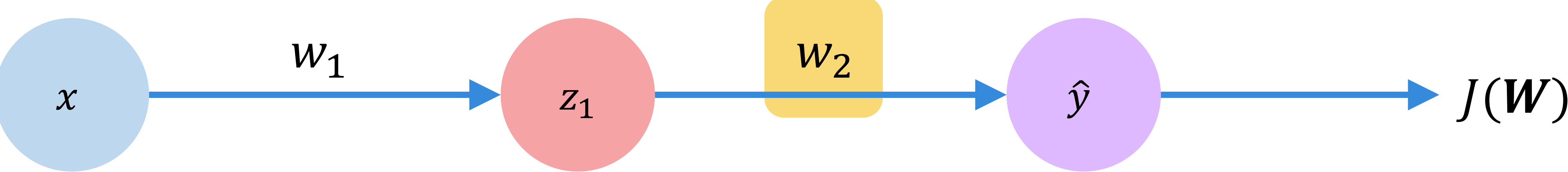
5. Return weights

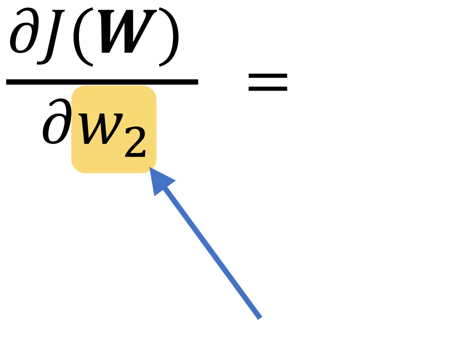
Computing Gradients: Backpropagation



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

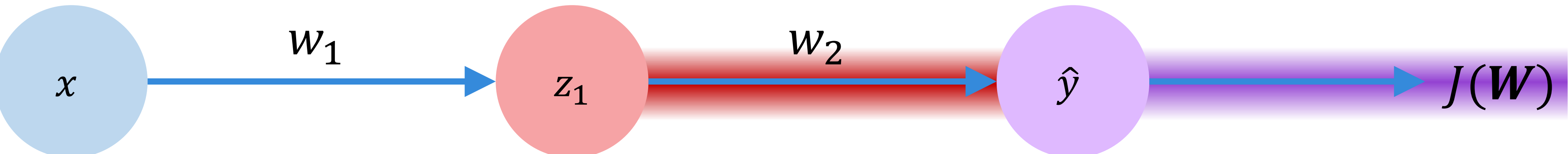
Computing Gradients: Backpropagation





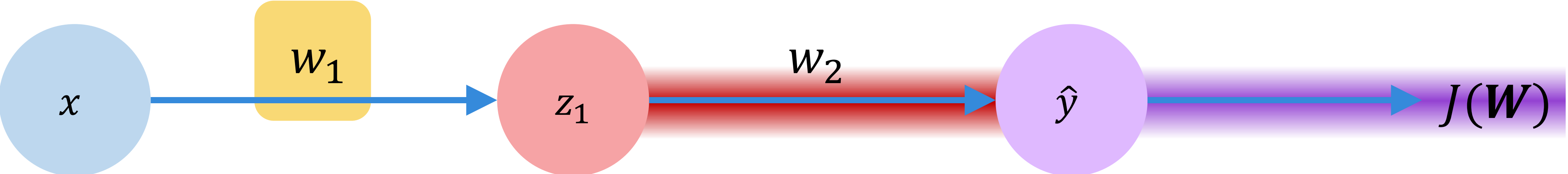
Let's use the chain rule!

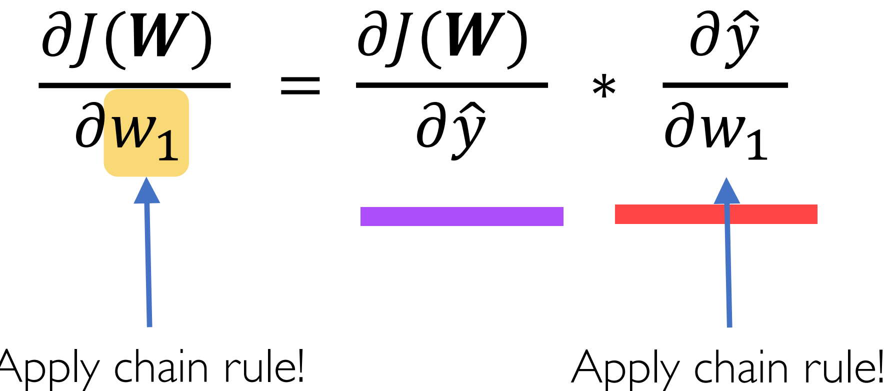
Computing Gradients: Backpropagation



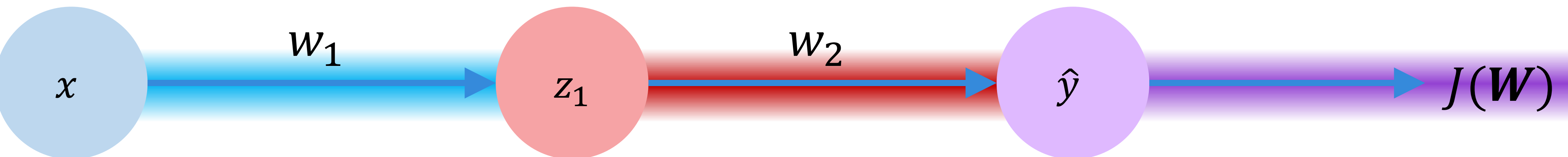
$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Computing Gradients: Backpropagation



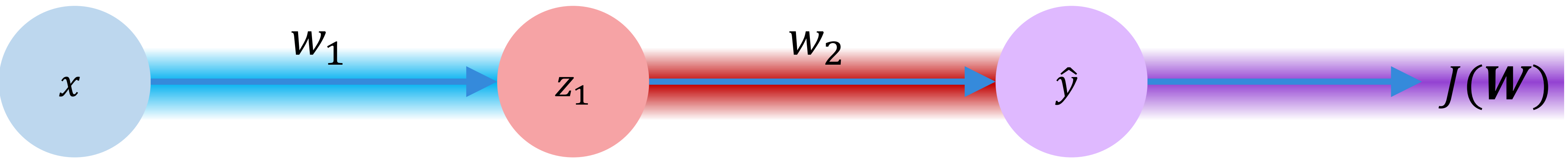


Computing Gradients: Backpropagation



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

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*

Neural Networks in Practice: Optimization

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

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

Loss Functions Can Be Difficult to Optimize

Remember:

Optimization through gradient descent

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

How can we set the learning rate?

Setting the Learning Rate

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

250
200
150
$J(\boldsymbol{W})$ 100
50
0
-50
-4 -3 -2 -1 0 1 2 3
$\boldsymbol{W}$

Initial guess

Setting the Learning Rate

Large learning rates *overshoot, become unstable and diverge*

$J(\boldsymbol{W})$

250
200
150
100
50
0
-50

-4 -3 -2 -1 0 1 2 3

$\boldsymbol{W}$

Initial guess

Setting the Learning Rate

Stable learning rates *converge smoothly and avoid local minima*

$J(\boldsymbol{\theta})$

250
200
150
100
50
0
-50
-4 -3 -2 -1 0 1 2 3

$\boldsymbol{W}$

Initial guess

How to deal with this?

Idea 1:

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

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

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...

Adaptive Learning Rate Algorithms

- Momentum
- Adagrad
- Adadelta
- Adam
- RMSProp

tf.train.RMSPropOptimizer

Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.

Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.

Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.

Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.

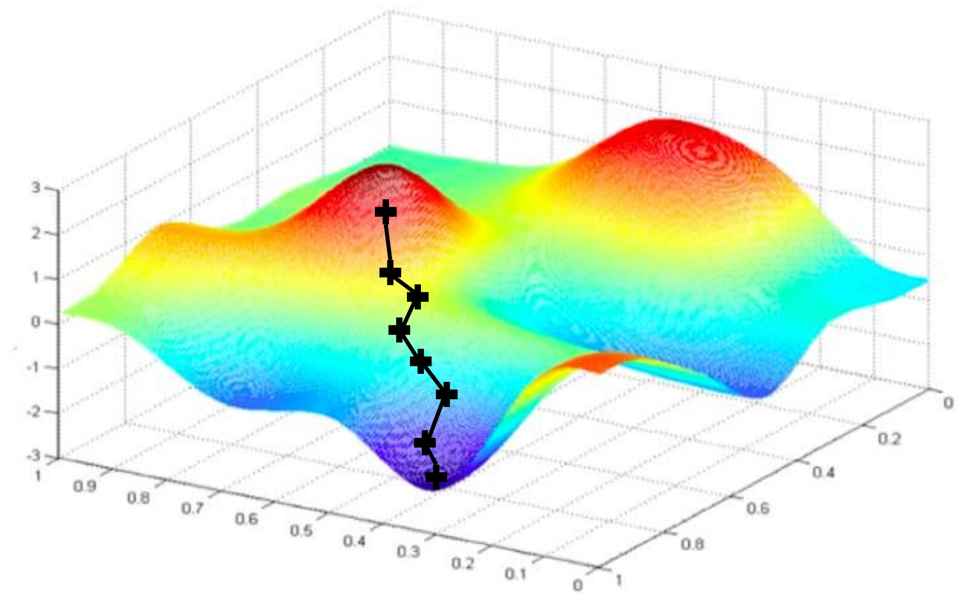
Additional details: http://ruder.io/optimizing-gradient-descent/

Neural Networks in Practice:
Mini-batches

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

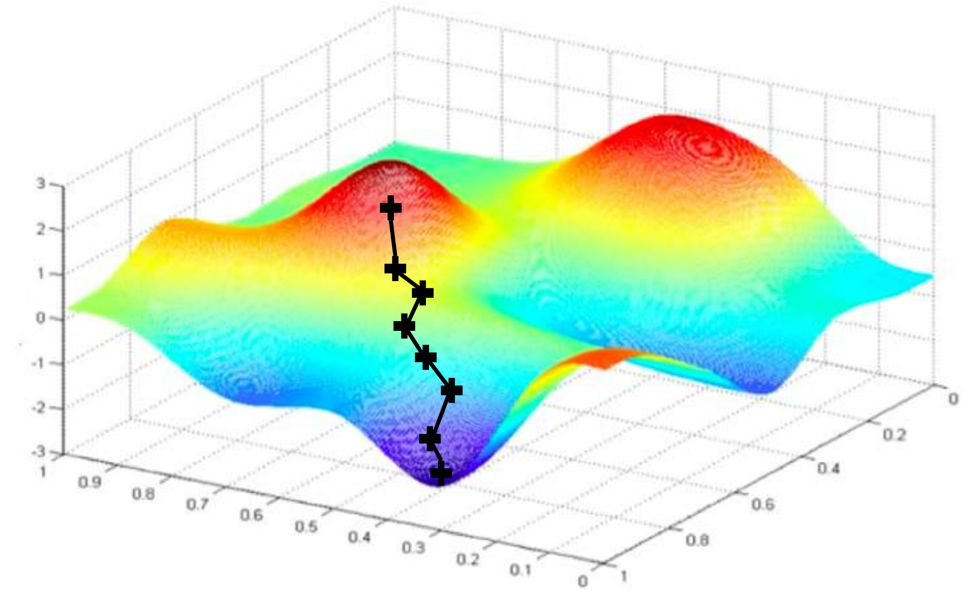


Gradient Descent

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

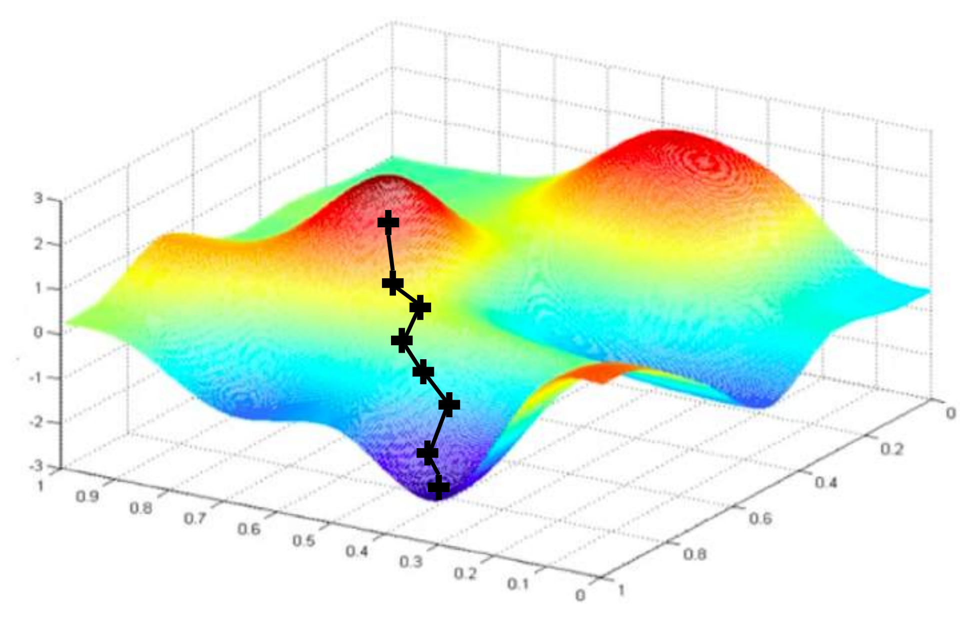
Can be very computational to compute!



Stochastic Gradient Descent

Algorithm

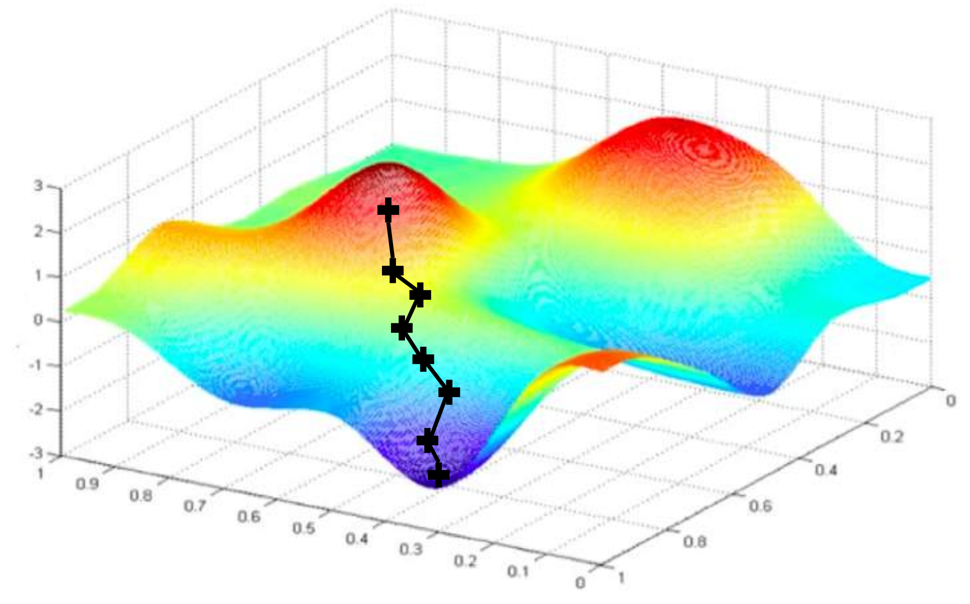
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point $\boldsymbol{i}$
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial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



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, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

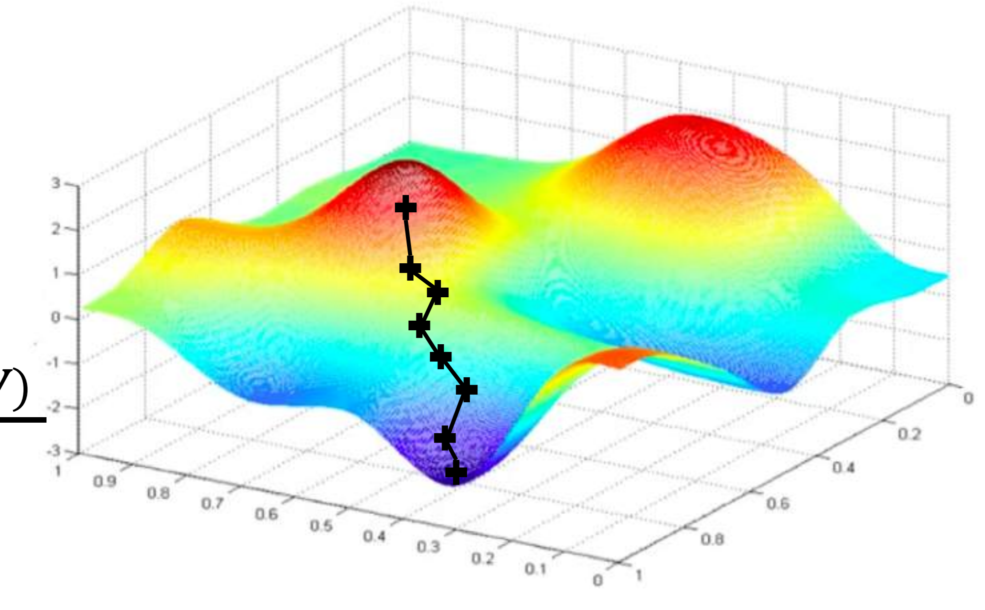


Easy to compute but
very noisy
(stochastic)!

Stochastic Gradient Descent

Algorithm

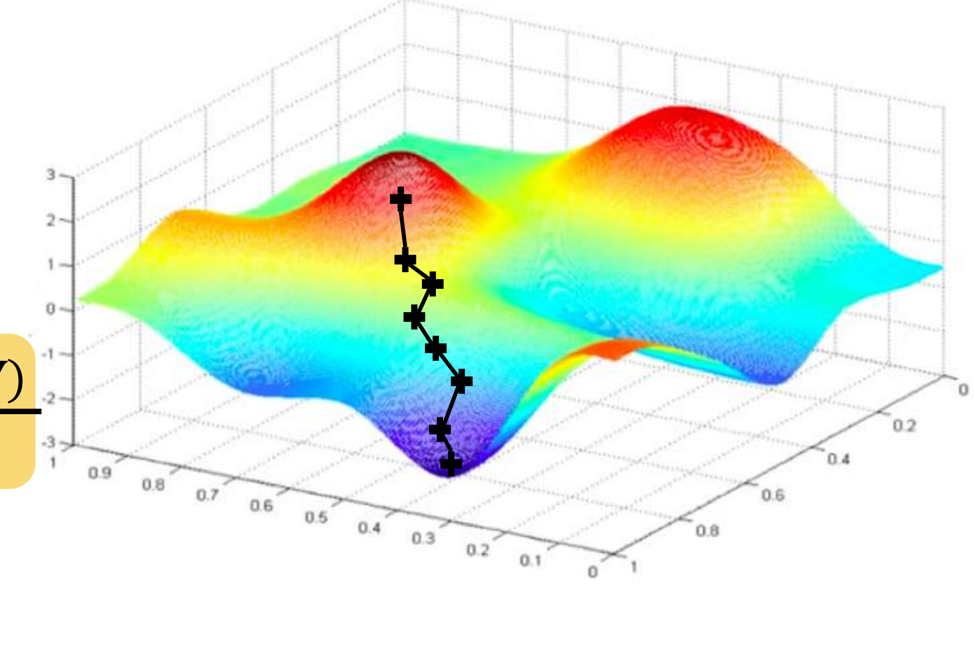
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update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



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, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
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!

Mini-batches while training

More accurate estimation of gradient

Smoother convergence

Allows for larger learning rates

Mini-batches while training

More accurate estimation of gradient

Smoother convergence

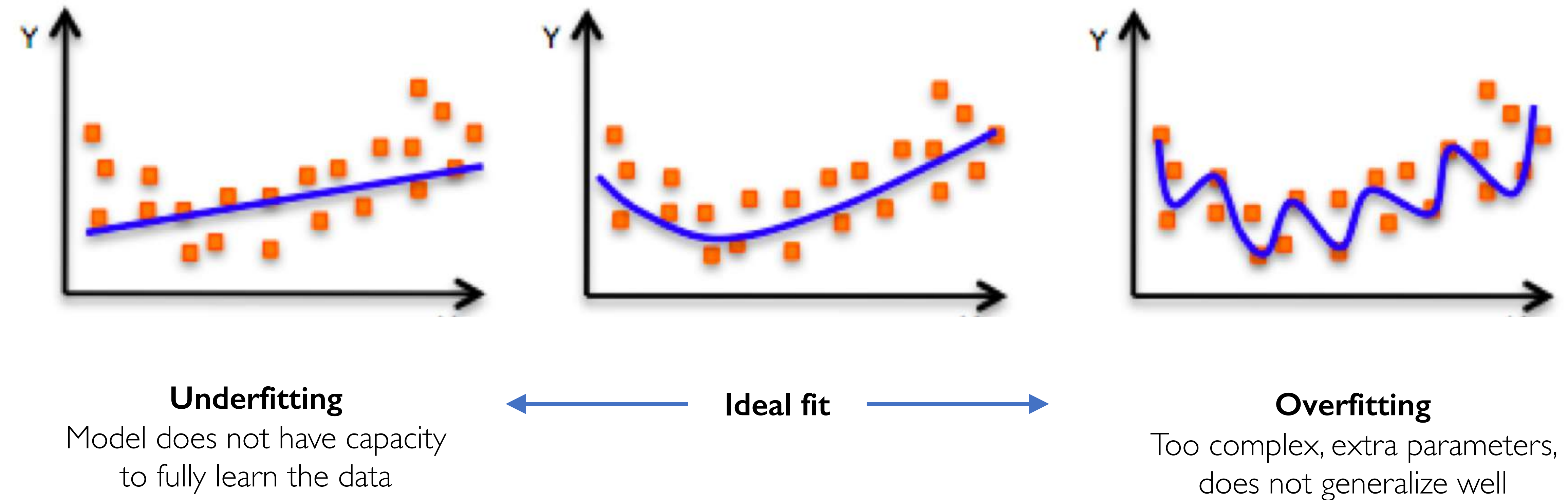
Allows for larger learning rates

Mini-batches lead to fast training!

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

Neural Networks in Practice: Overfitting

The Problem of Overfitting



Regularization

What is it?

Technique that constrains our optimization problem to discourage complex models

Regularization

What is it?

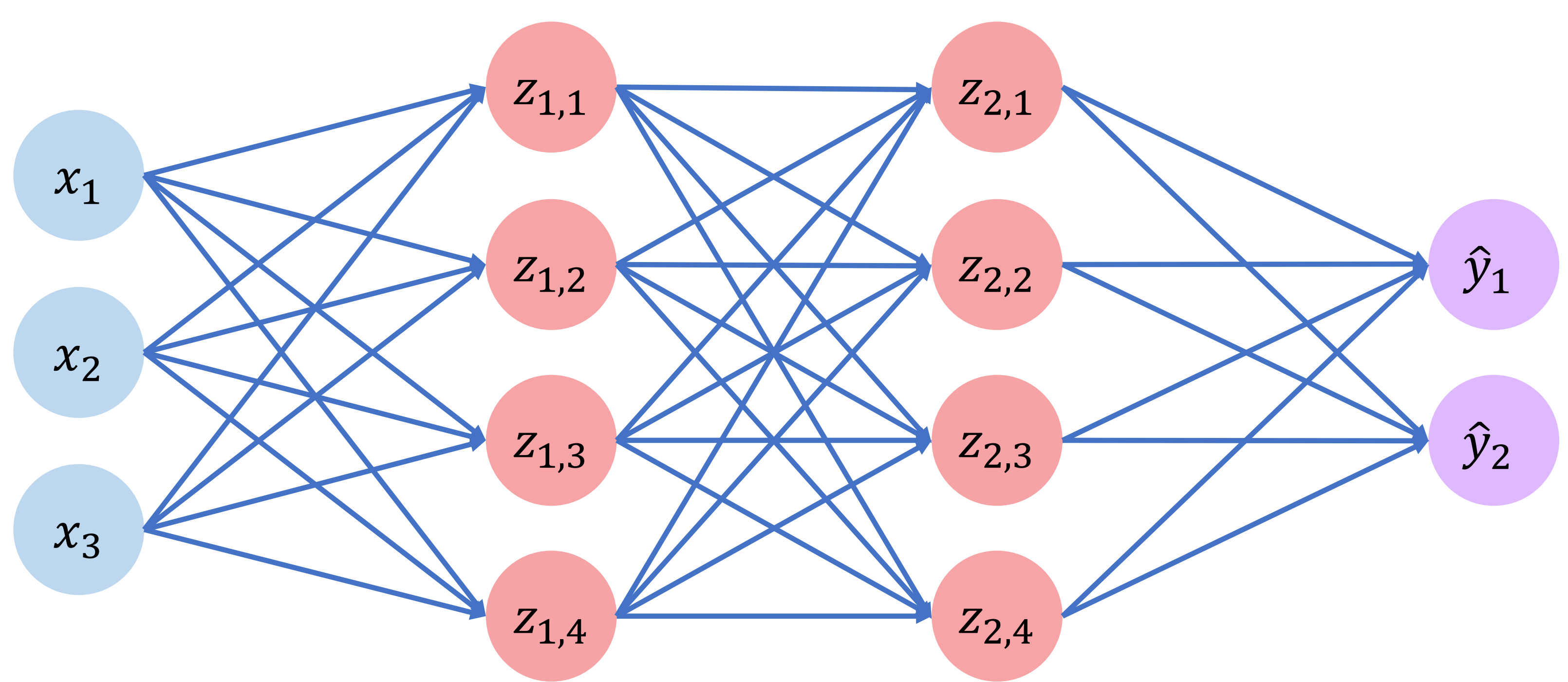
Technique that constrains our optimization problem to discourage complex models

Why do we need it?

Improve generalization of our model on unseen data

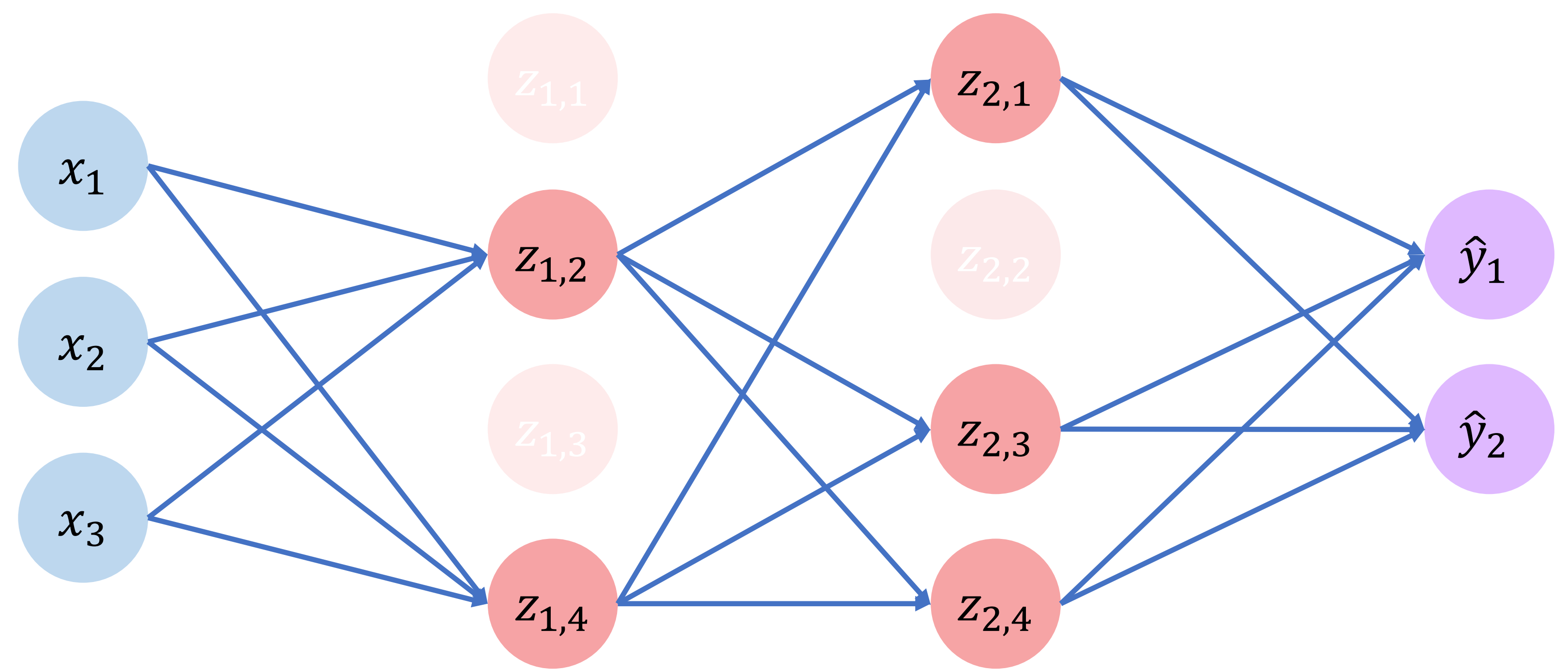
Regularization 1: Dropout

- During training, randomly set some activations to 0



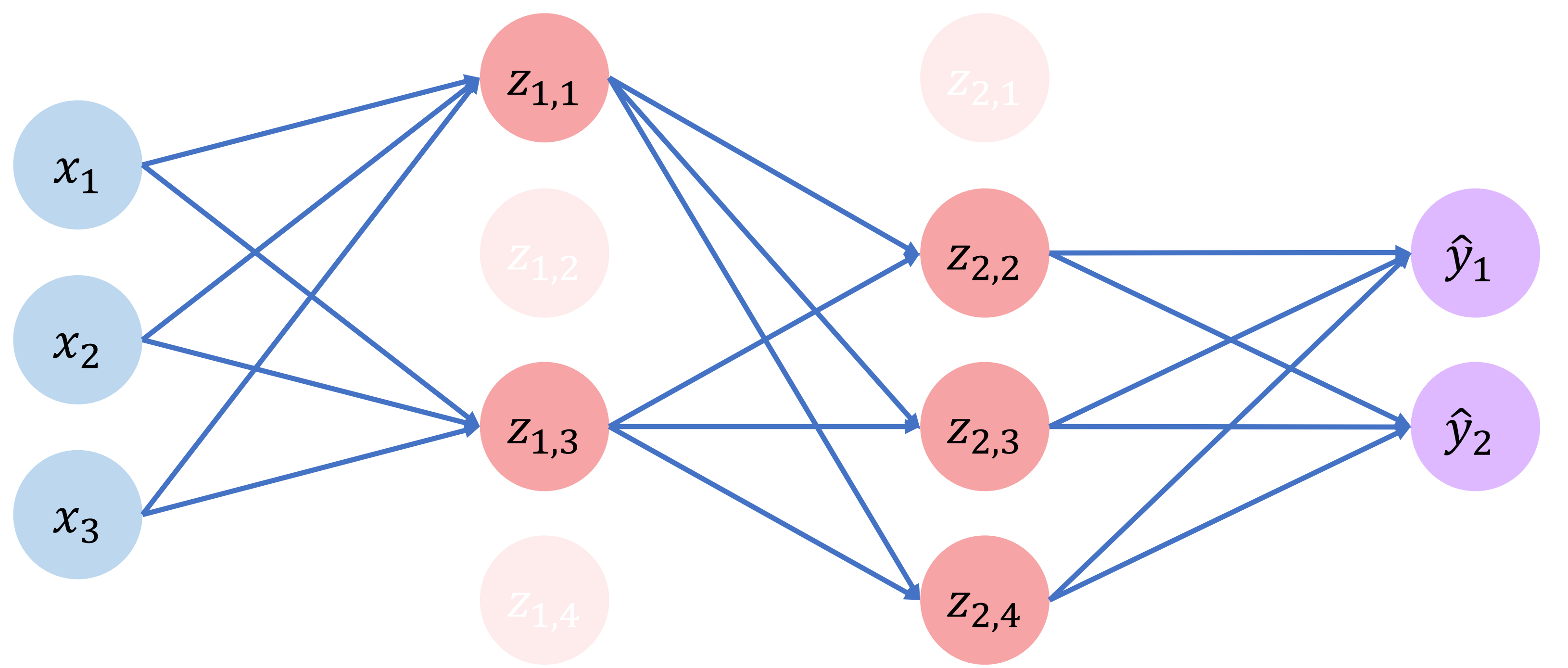
Regularization 1: Dropout

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



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



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

Loss

Training Iterations

Regularization 2: Early Stopping

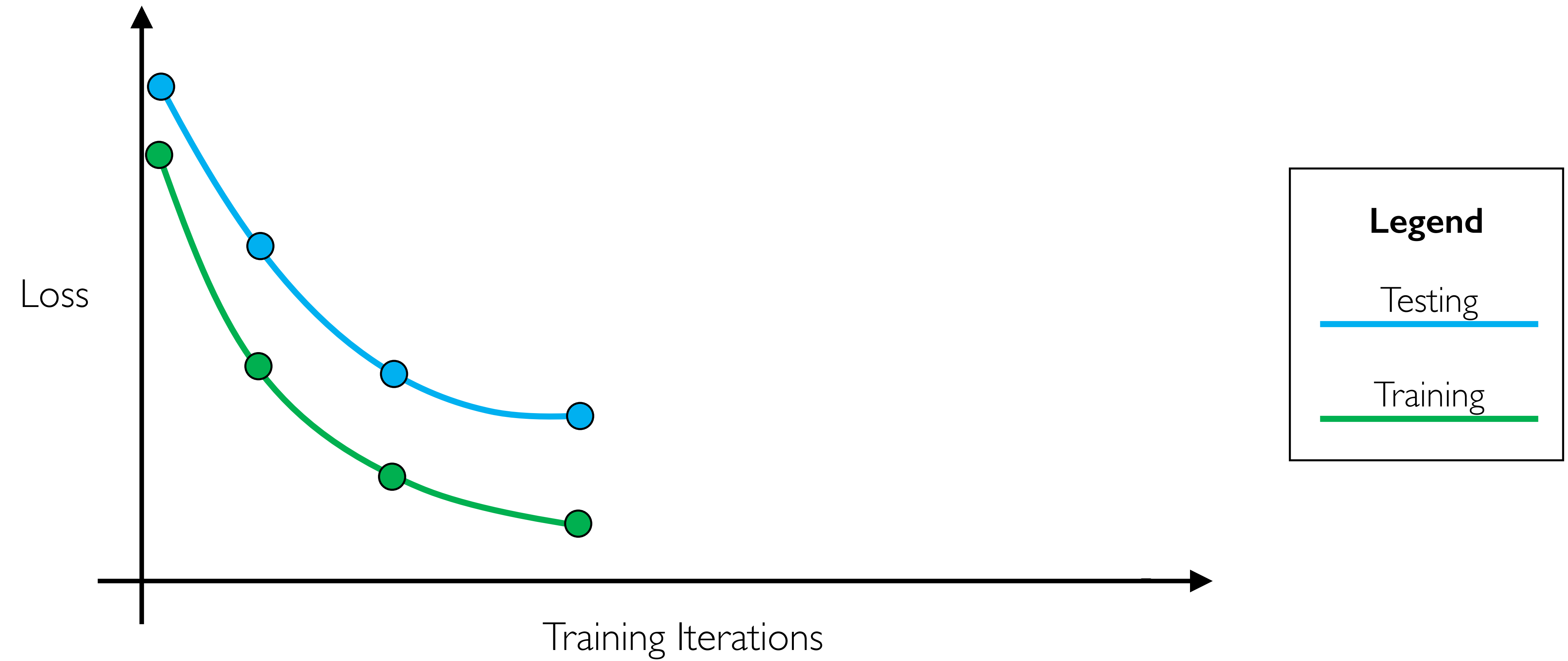
- Stop training before we have a chance to overfit

Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

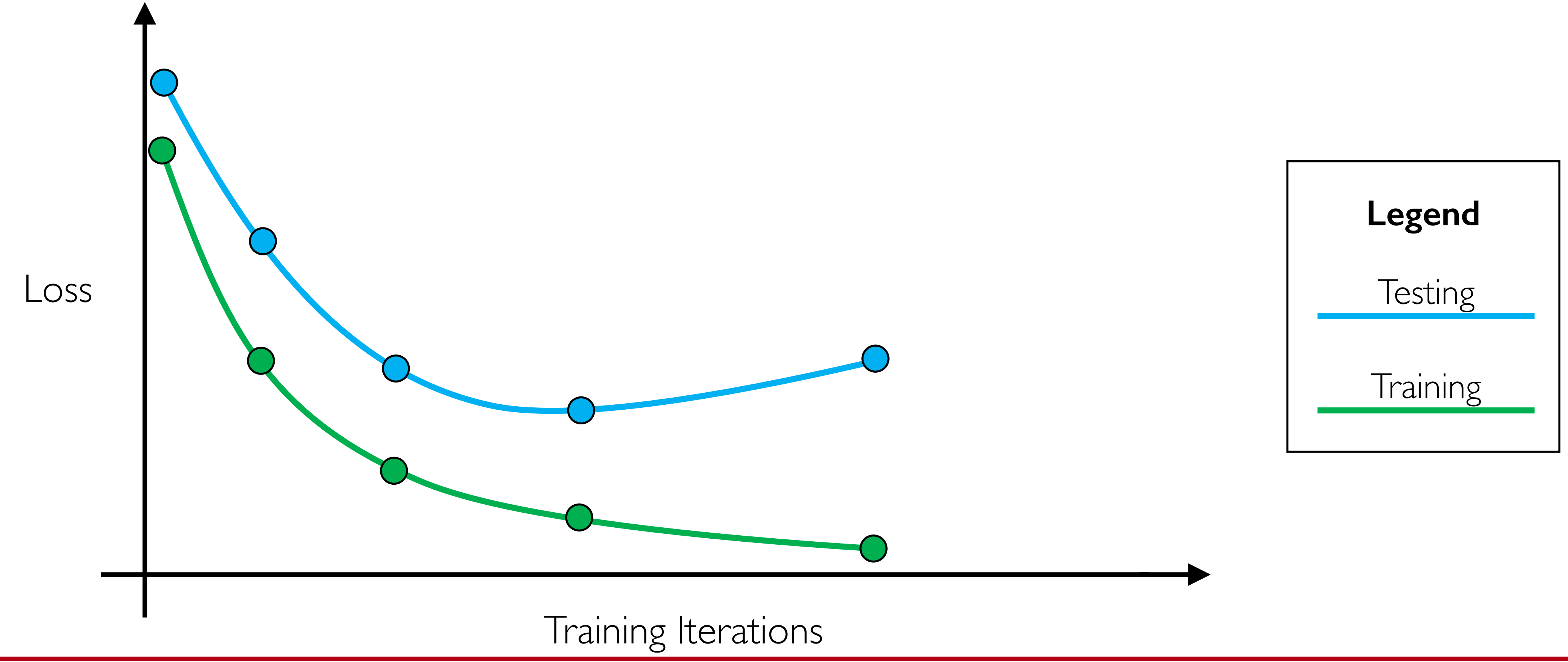
Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



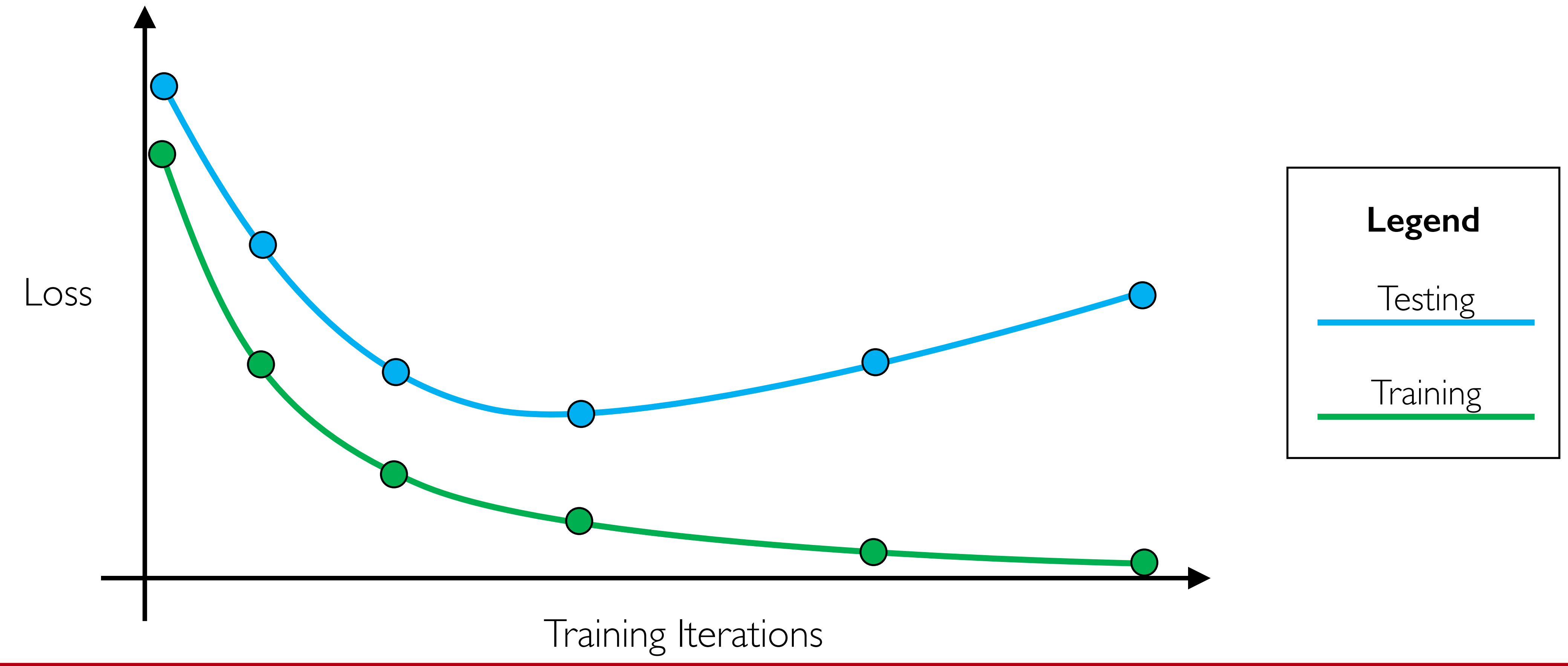
Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

Loss

Stop training here!

Training Iterations

Legend

Testing

Training

Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

Under-fitting

Over-fitting

Loss

Stop training here!

Training Iterations

Legend

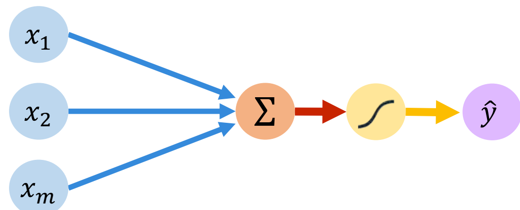
Testing

Training

Core Foundation Review

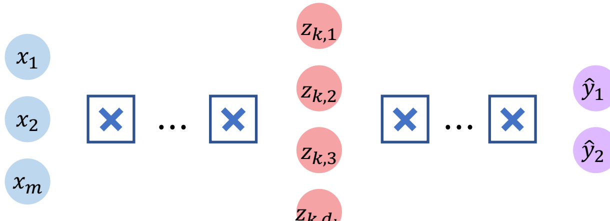
The Perceptron

- Structural building blocks
- Nonlinear activation functions



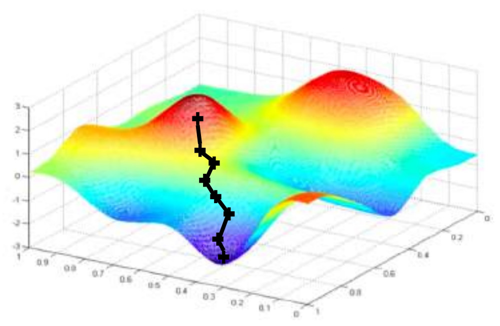
Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



Training in Practice

- Adaptive learning
- Batching
- Regularization



Questions?