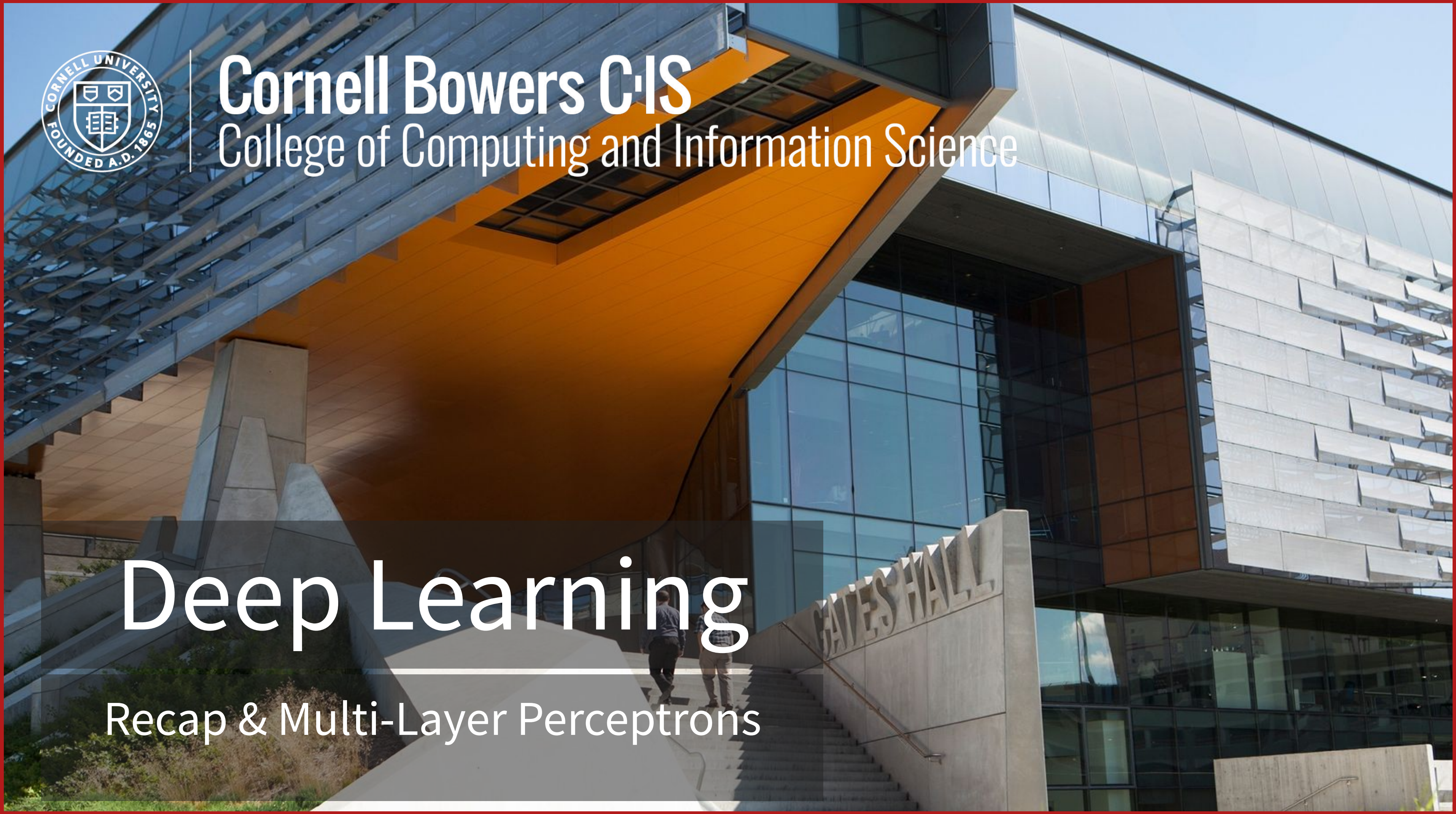
Cornell Bowers C·IS
College of Computing and Information Science

Deep Learning

Recap & Multi-Layer Perceptrons



Cornell Bowers C·IS

Quick Recap- Logistics

CS 4782: Intro to Deep Learning, Spring 2026

Overview Assignments Schedule References Policies

Instructors: Kilian Q. Weinberger and Wei-Chiu Ma

Office hours:

Kilian Weinberger : Mondays 9:45 - 10:30 am (Booking Link) in 410 Gates Hall
Wei-Chiu Ma: Wednesday 14:30 - 15:30 (Booking Link) in 416A Gates Hall

Lectures: Tuesday and Thursday from 2:55 to 4:10 pm.

Course staff office hours: Calendar Link (TBD)

Course overview: This class is an introductory course to deep learning. It covers the fundamental principles behind training and inference of deep networks, deep reinforcement learning, the specific architecture design choices applicable for different data modalities, discriminative and generative settings, and the ethical and societal implications of such models.

Prerequisites: Fundamentals of Machine Learning (CS4780 , ECE4200 , STCSI4740), Python fluency (CS1110), and programming proficiency (e.g. CS 2110).

Course logistics: For enrolled students the companion Canvas page serves as a hub for access to Ed Discussions (the course forum) and Gradescope (for HWs). If you are enrolled in the course you should automatically have access to the site. Please let us know if you are unable to access it.

https://www.cs.cornell.edu/courses/cs4782/2026sp/

Cornell Bowers CIS

Quick Recap- Logistics

CS 4782: Intro to Deep Learning, Spring 2026

Overview Assignments Schedule References Policies

Instructors: Kilian Q. Weinberger and Wei-Chiu Ma

Office hours:

Kilian Weinberger : Mondays 9:45 - 10:30 am (Booking Link) in 410 Gates Hall

Wei-Chiu Ma: Wednesday 14:30 - 15:30 (Booking Link) in 415A Gates Hall

Lectures: Tuesday and Thursday from 2:55 to 4:10 pm.

Course staff office hours: Calendar Link (TBD)

Course overview: This class is an introductory course to deep learning. It covers the fundamental principles behind training and inference of deep networks, deep reinforcement learning, the specific architecture design choices applicable for different data modalities, discriminative and generative settings, and the ethical and societal implications of such models.

Prerequisites: Fundamentals of Machine Learning (CS4780 , ECE4200 , STCSI4740), Python fluency (CS1110), and programming proficiency (e.g. CS 2110).

Course logistics: For enrolled students the companion Canvas page serves as a hub for access to Ed Discussions (the course forum) and Gradescope (for HWs). If you are enrolled in the course you should automatically have access to the site. Please let us know if you are unable to access it.

https://www.cs.cornell.edu/courses/cs4782/2026sp/

No laptops/mobiles/smart devices in class please!

Agenda

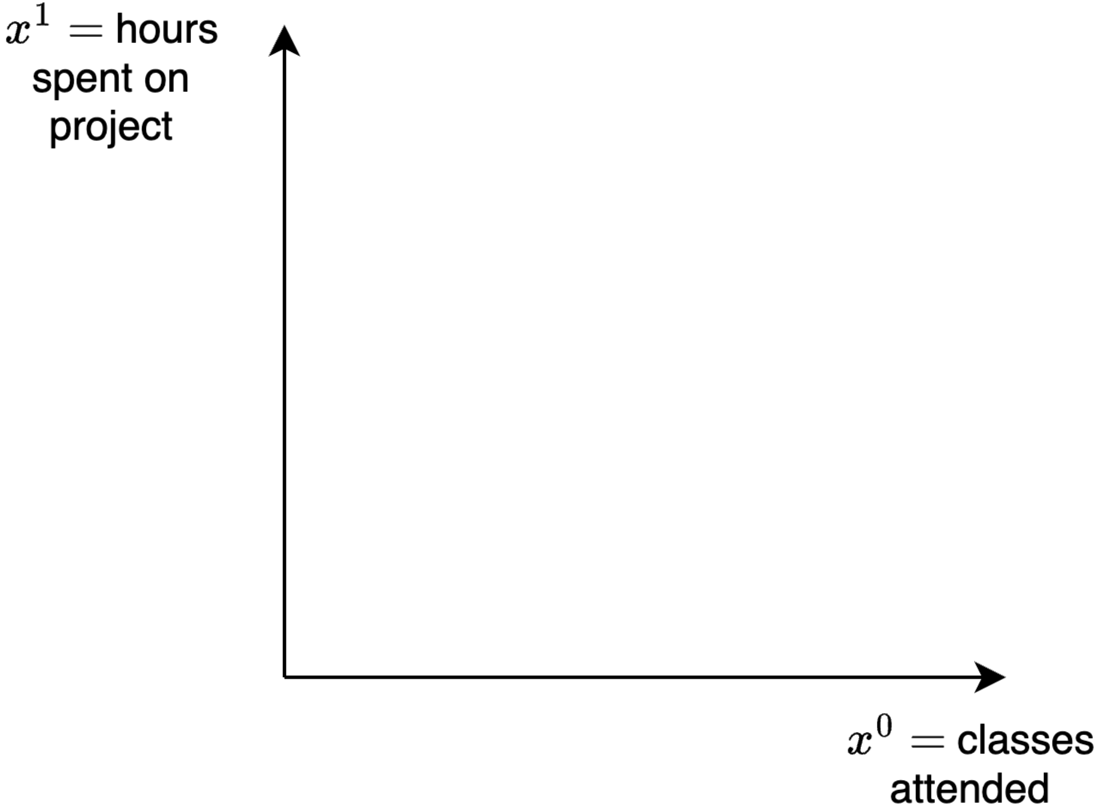
- Perceptron
- Logistic Regression
- Gradient Descent
- Multi-Layer Perceptrons (MLPs)
- Backpropagation

A Classification Problem:

Will I Pass This Class?

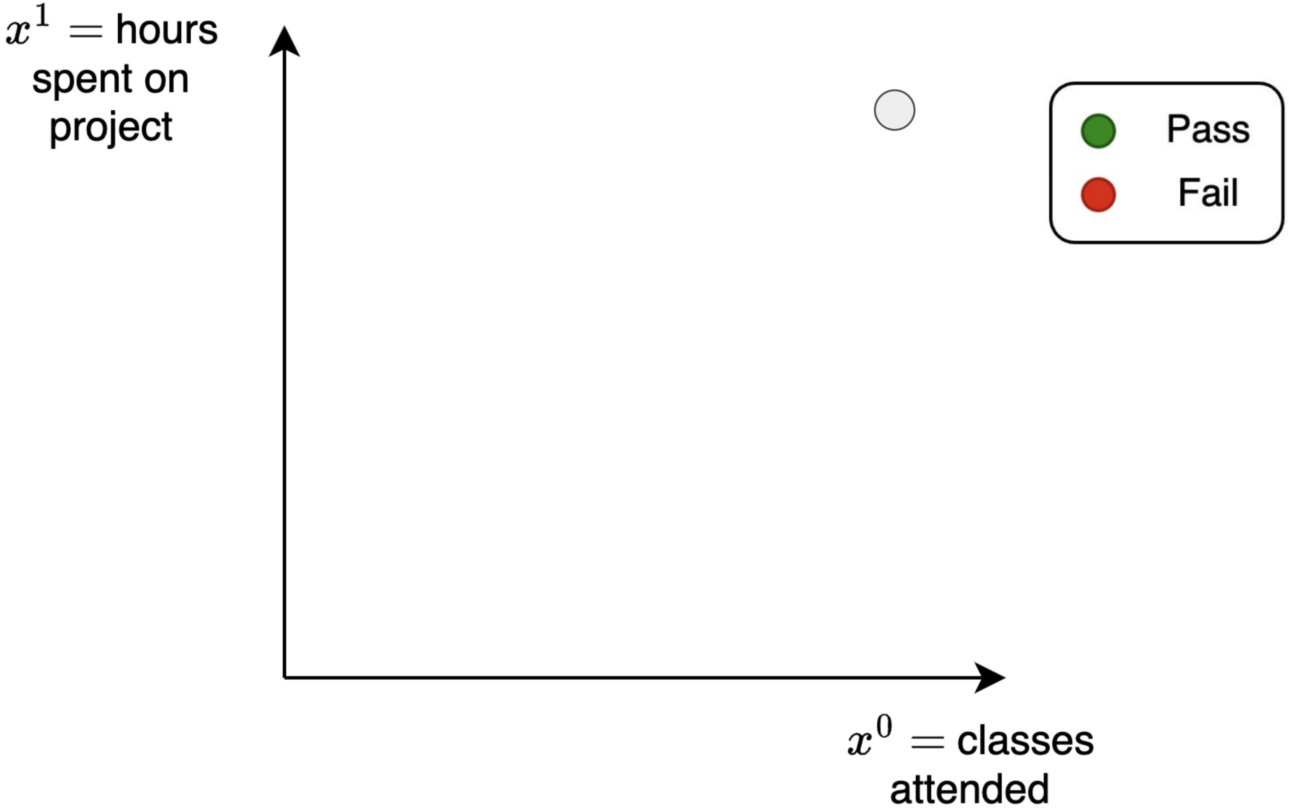
Cornell Bowers C·IS

A Classification Problem: Will I Pass This Class?



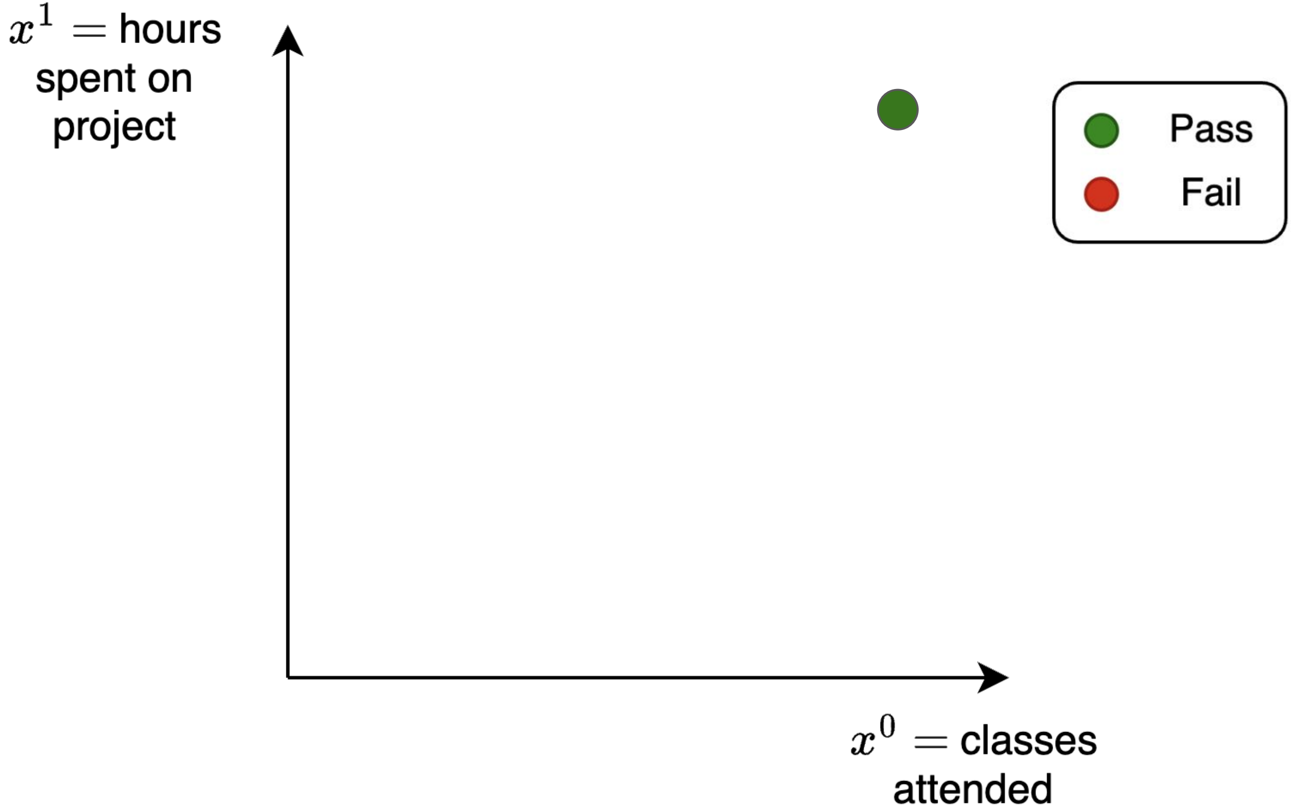
Cornell Bowers CIS

A Classification Problem: Will I Pass This Class?

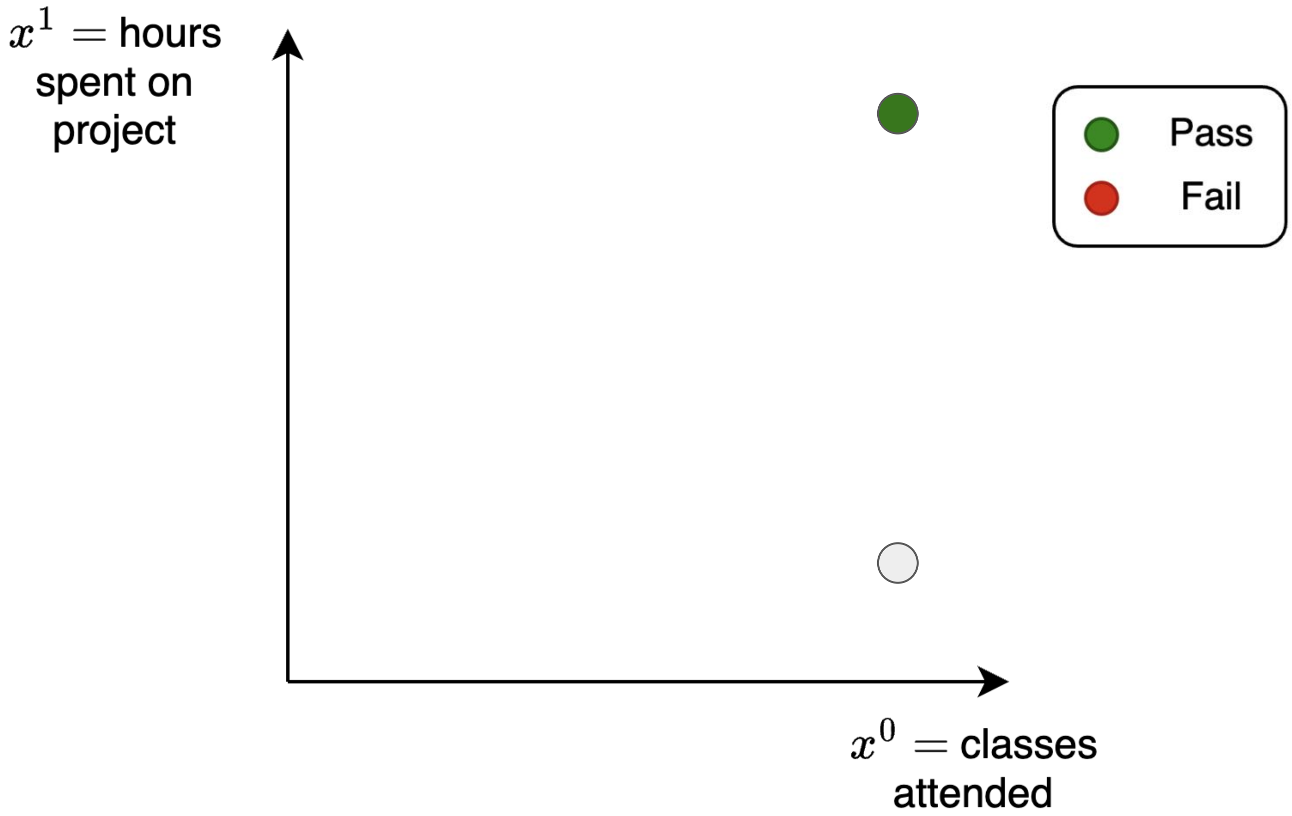


Cornell Bowers C·IS

A Classification Problem: Will I Pass This Class?

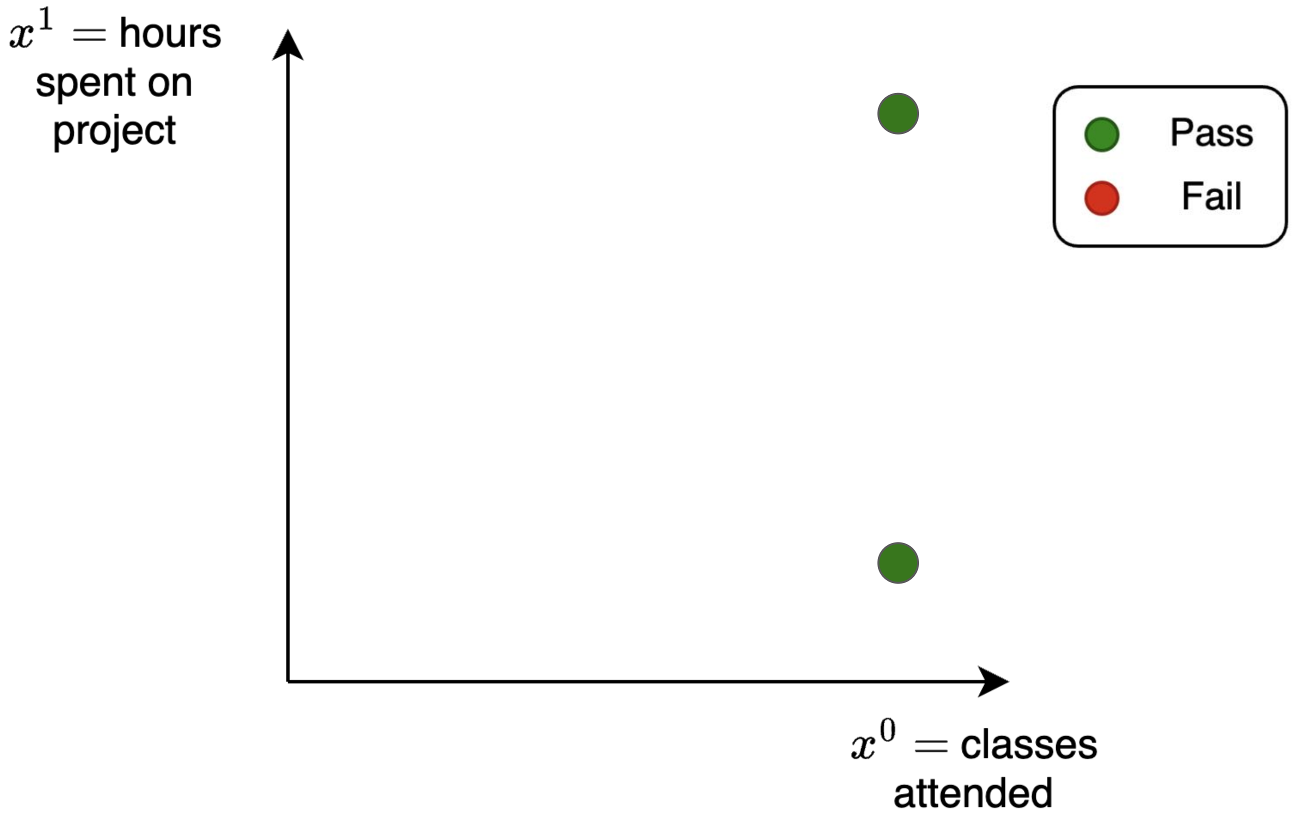


A Classification Problem: Will I Pass This Class?



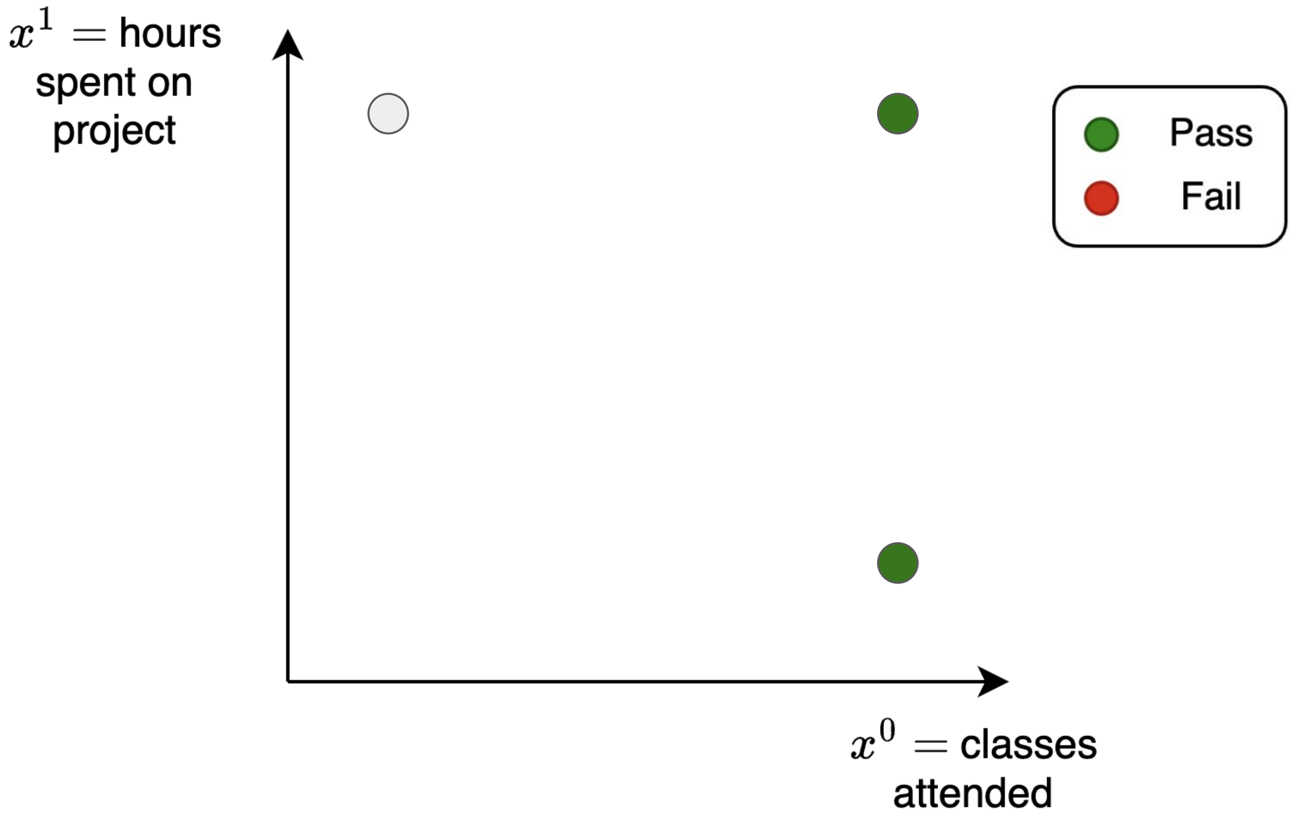
Cornell Bowers C·IS

A Classification Problem: Will I Pass This Class?

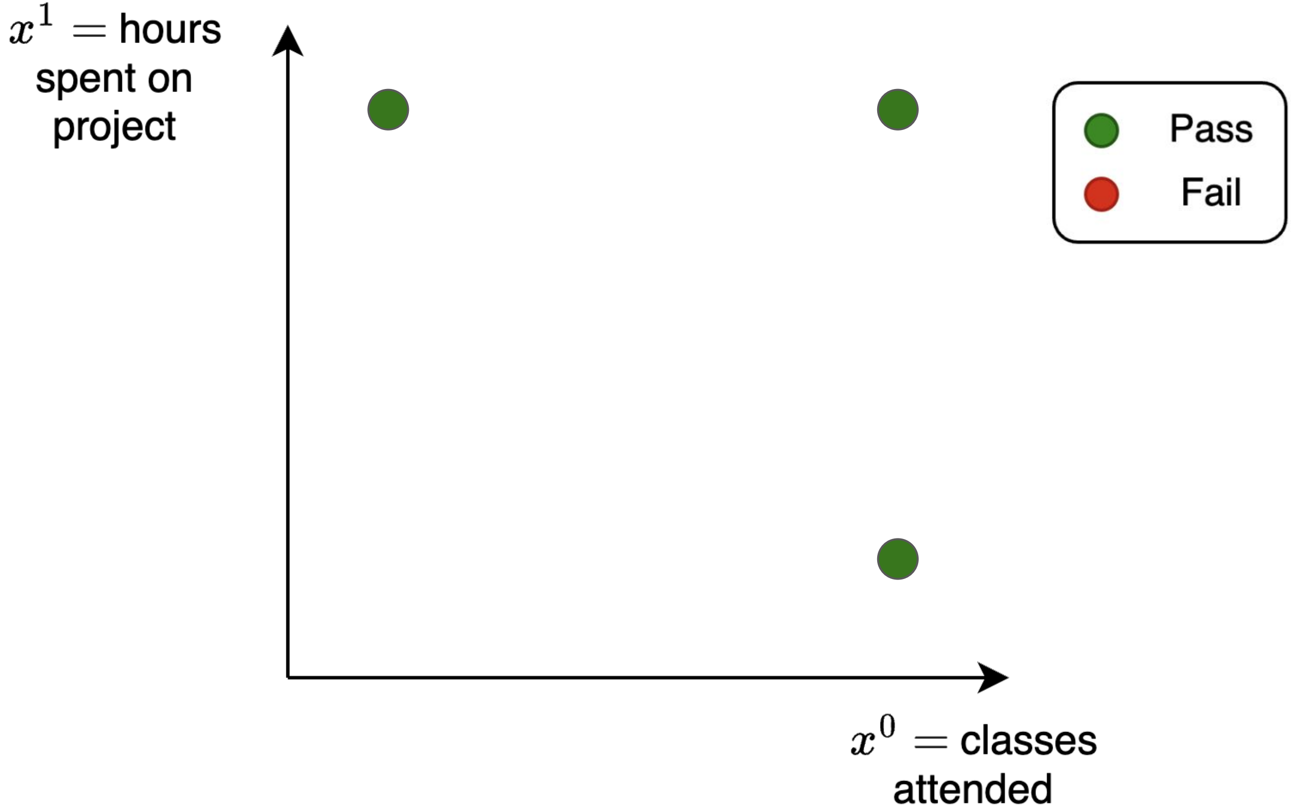


Cornell Bowers CIS

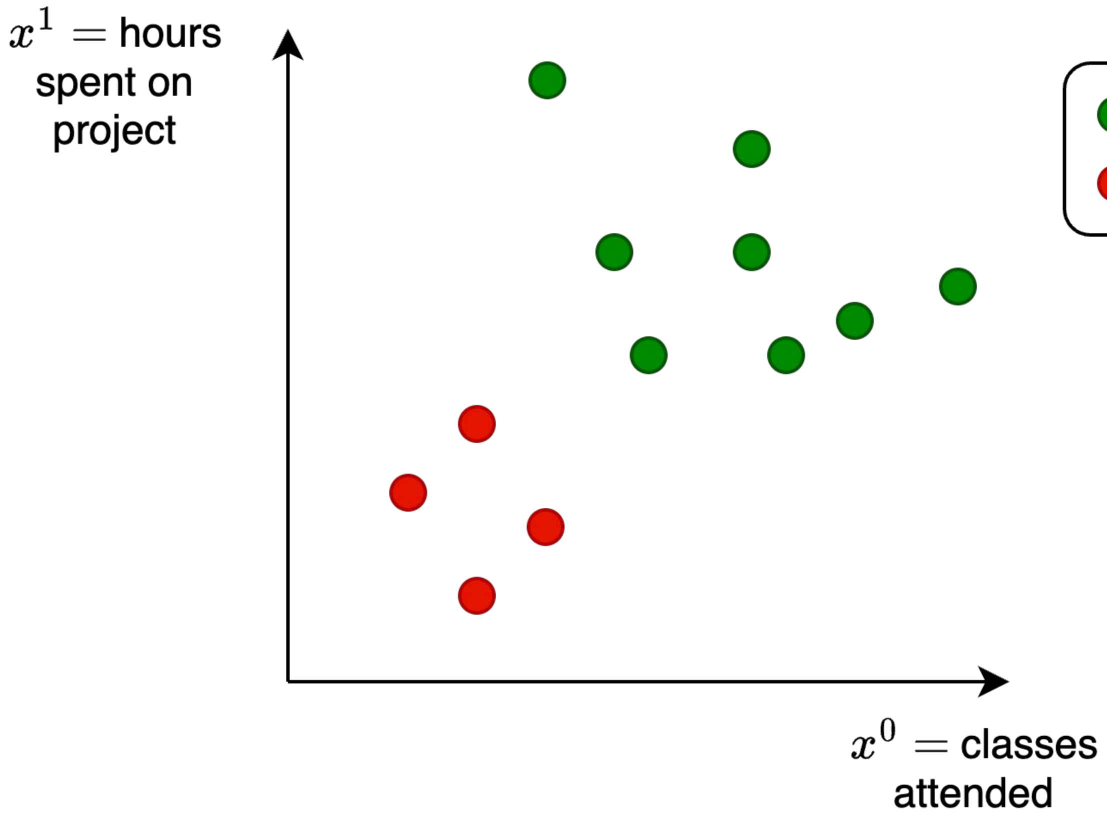
A Classification Problem: Will I Pass This Class?



A Classification Problem: Will I Pass This Class?

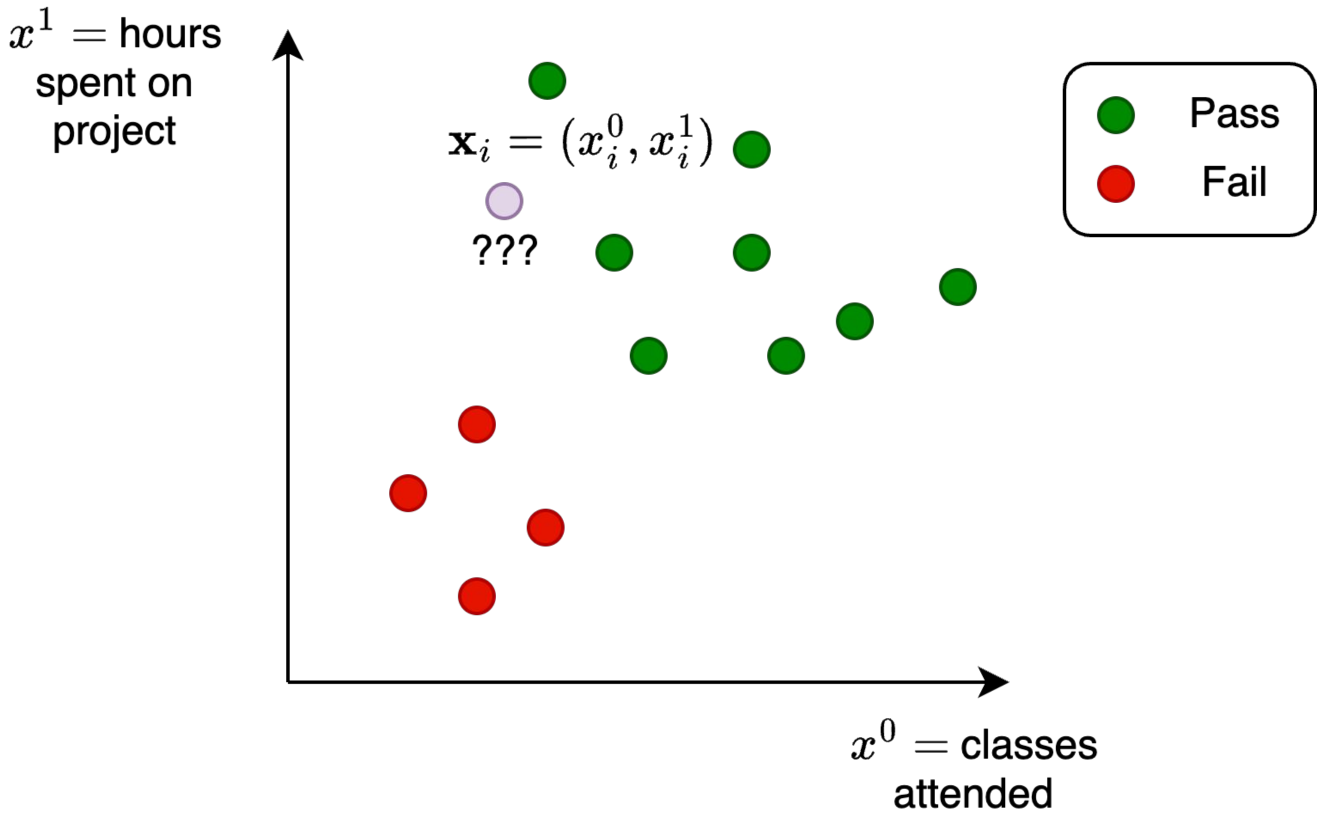


A Classification Problem: Will I Pass This Class?



Cornell Bowers C·IS

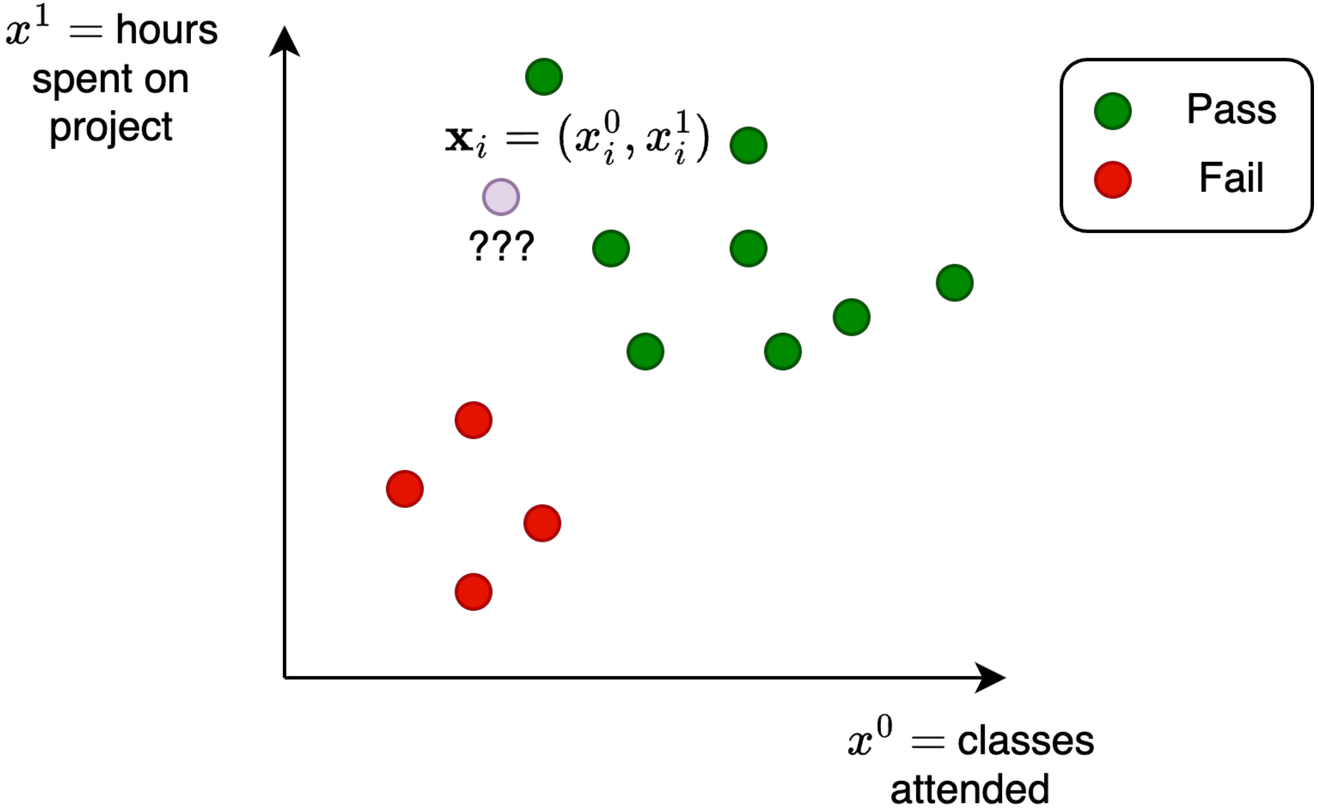
A Classification Problem: Will I Pass This Class?



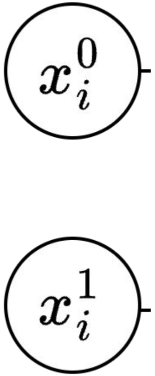
What are key components in ML?

- Training data
- Model Class / Hypothesis space
- Loss function
- Optimization

A Classification Problem: Will I Pass This Class?



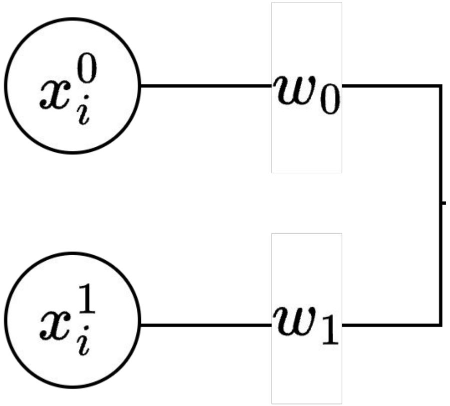
Perceptron



Input

Cornell Bowers C·IS

Perceptron

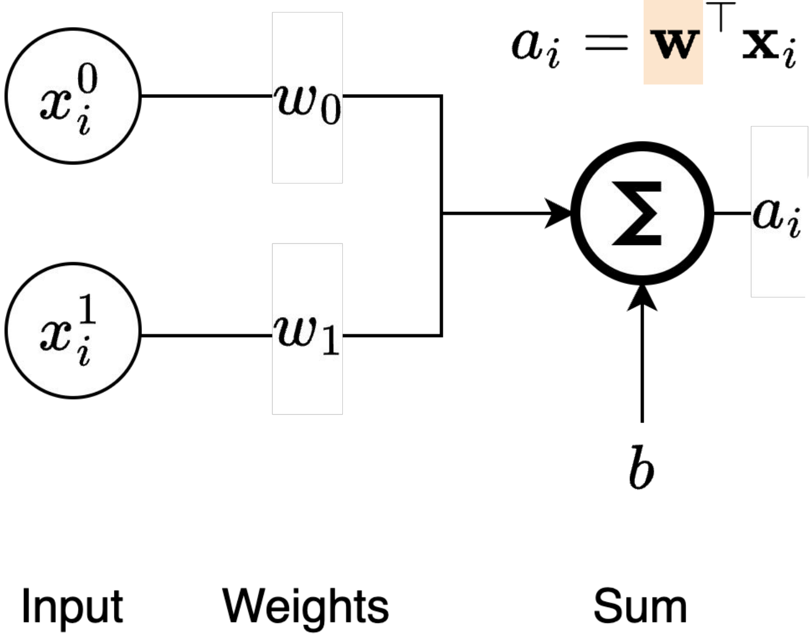


Input Weights

Perceptron

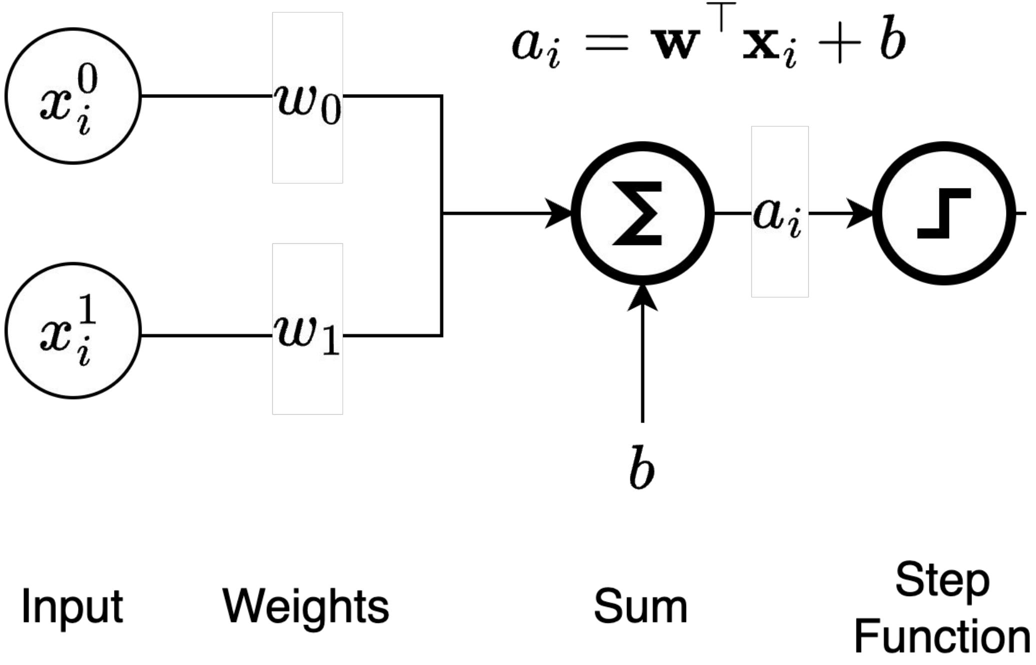
Learned from training data

$$a_i = \mathbf{w}^\top \mathbf{x}_i + b$$



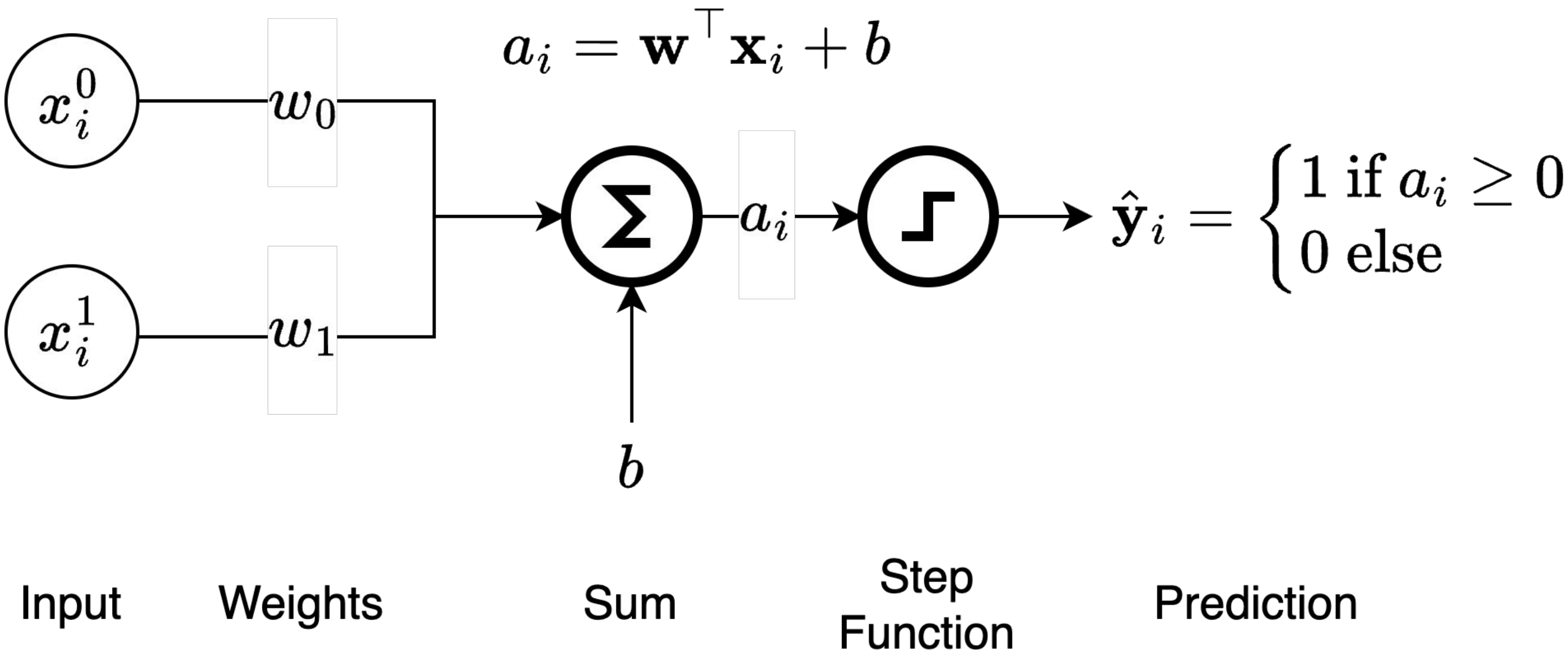
Perceptron

$$a_i = \mathbf{w}^\top \mathbf{x}_i + b$$



Cornell Bowers C·IS

Perceptron

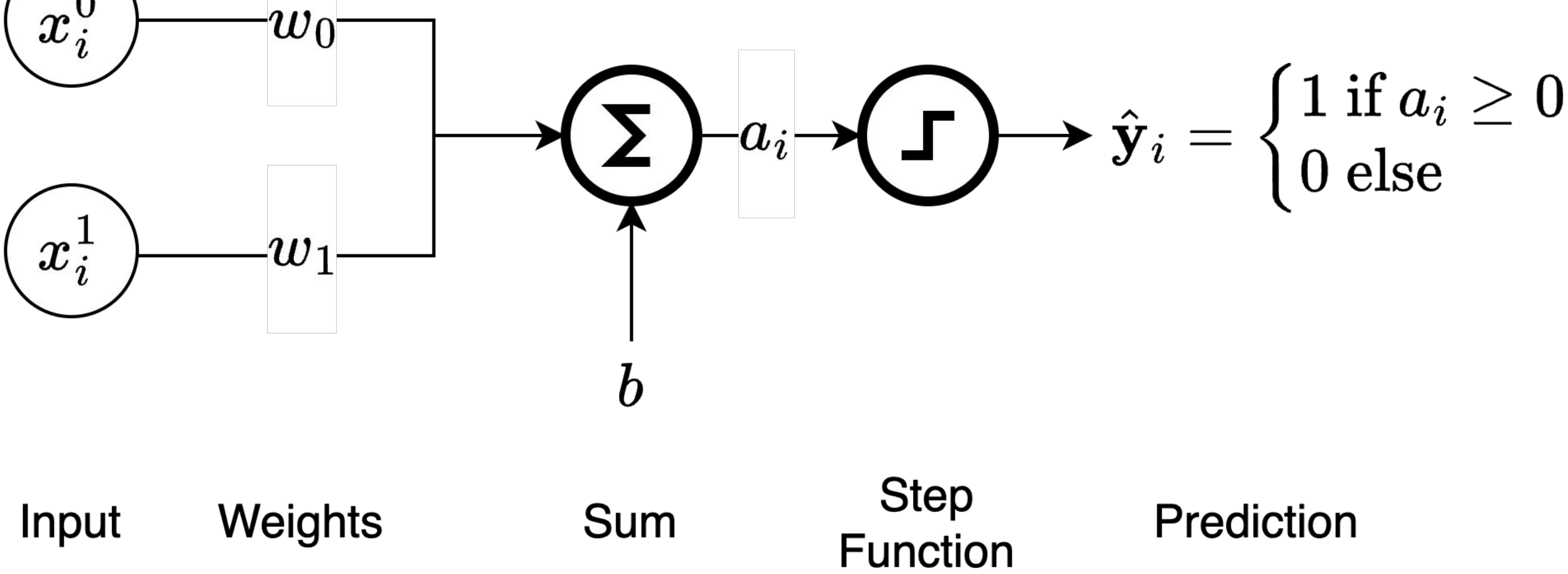


Cornell Bowers C·IS

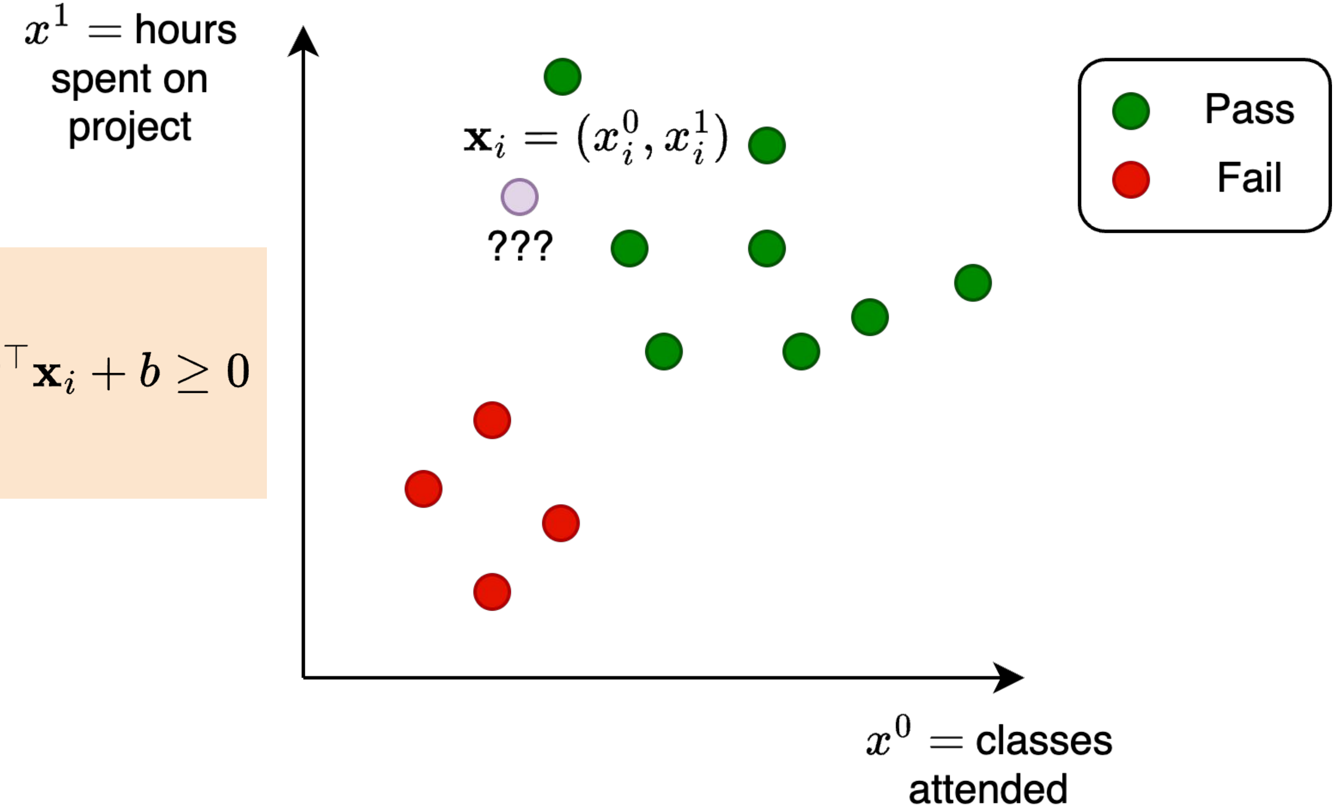
Perceptron

- Linear classifier
  - Predecessor to neural network

$$a_i = \mathbf{w}^\top \mathbf{x}_i + b$$



A Classification Problem: Will I Pass This Class?



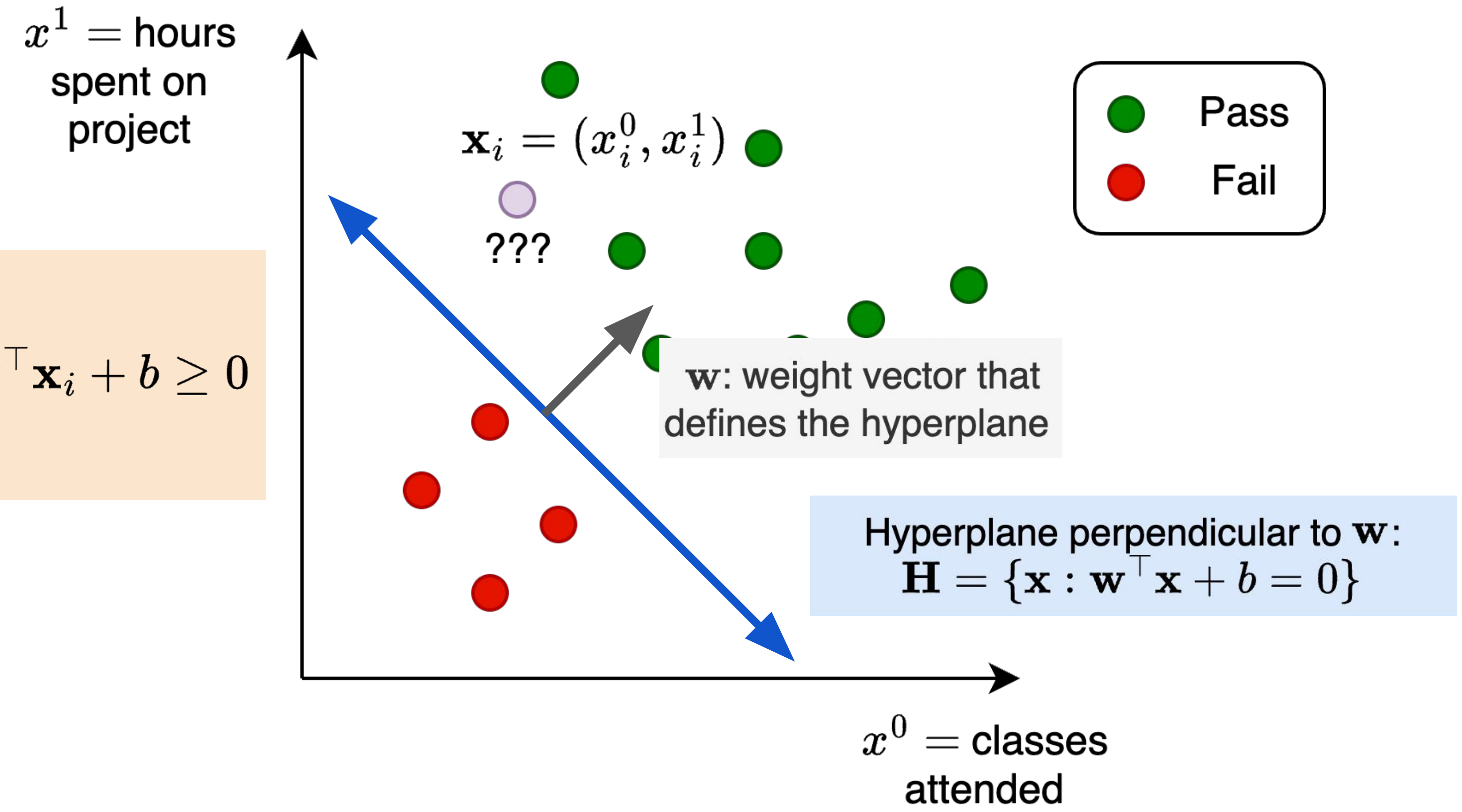
Recall:

$$y_i = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$

A Classification Problem: Will I Pass This Class?

Recall:

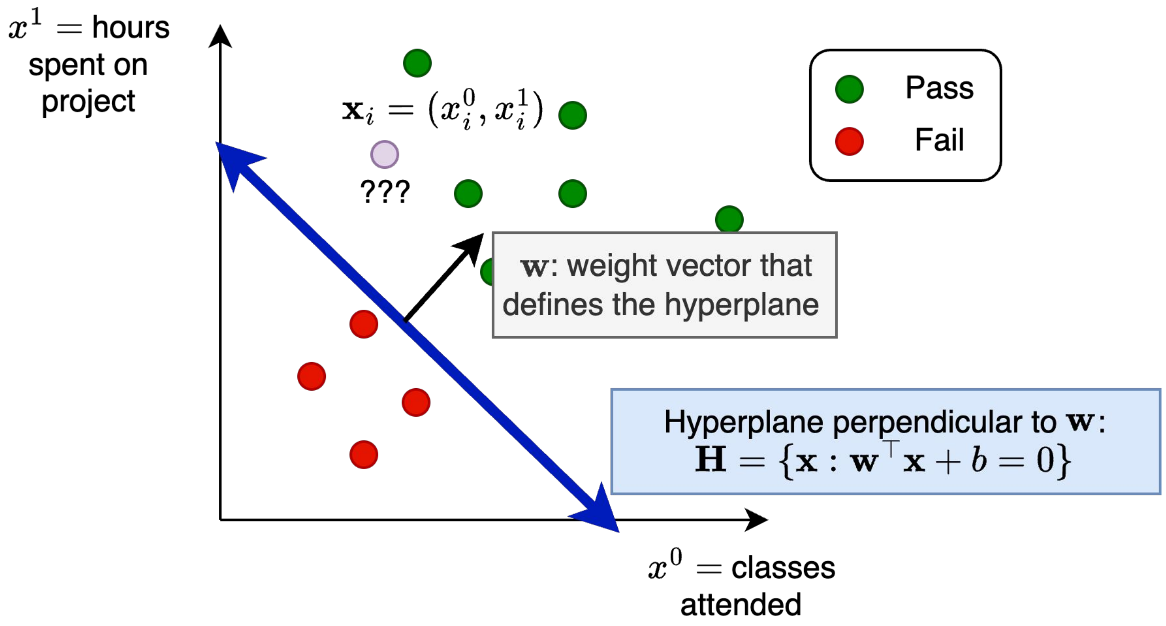
$$y_i = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$



A Classification Problem: Will I Pass This Class?

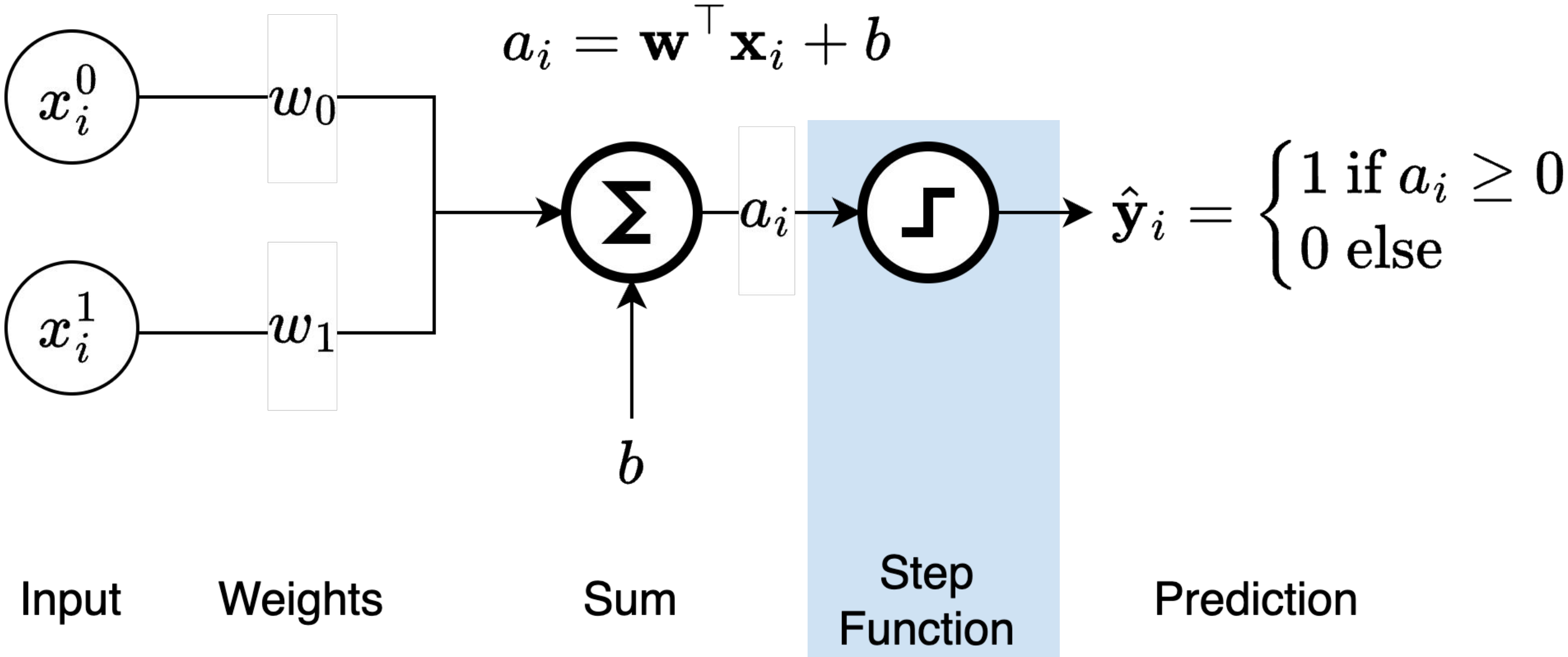
- Perceptron defines a linear classification boundary

$$y_i = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$

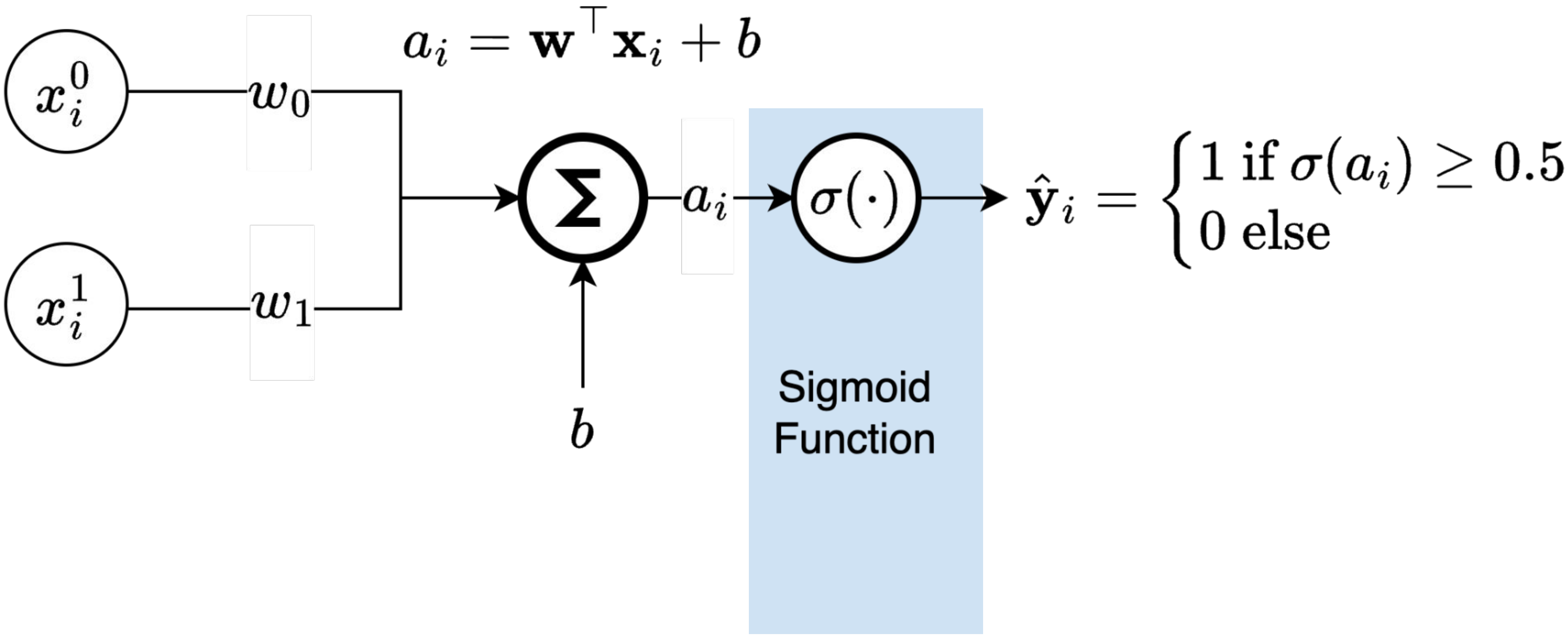


Cornell Bowers C·IS

Perceptron

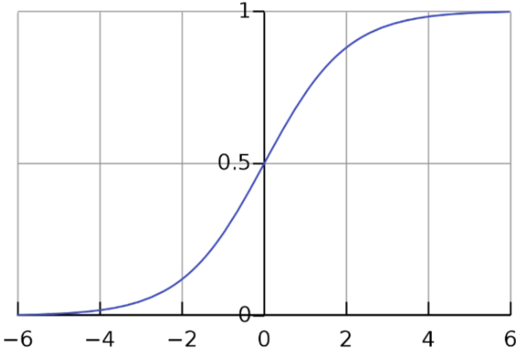


The “Soft” Perceptron

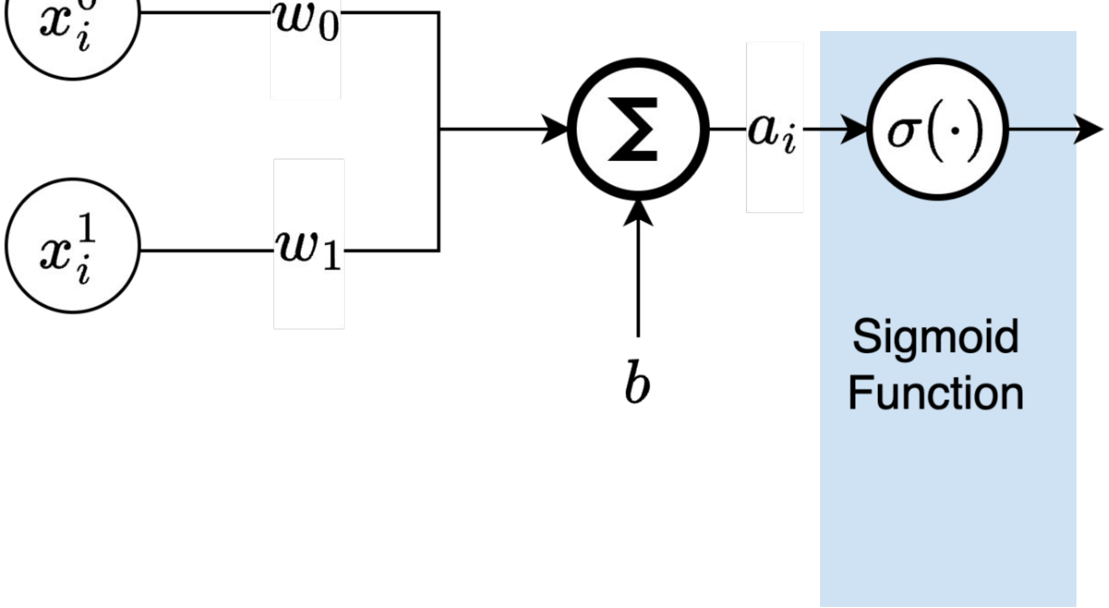


The "Soft" Perceptron

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



$$a_i = \mathbf{w}^\top \mathbf{x}_i + b$$

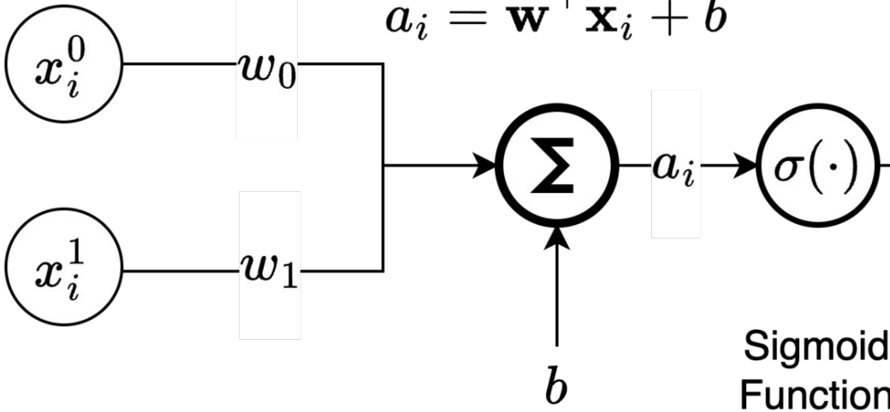


$$\hat{\mathbf{y}}_i = \begin{cases} 1 \text{ if } \sigma(a_i) \geq 0.5 \\ 0 \text{ else} \end{cases}$$

In other words… Logistic Regression

- A single-layer perceptron

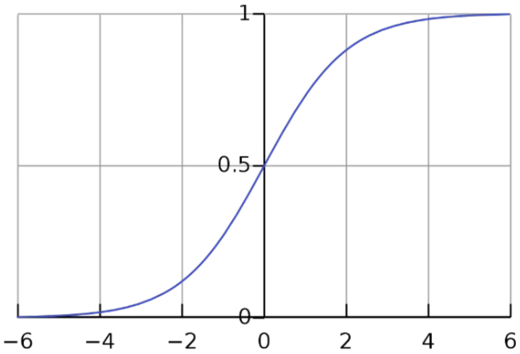
$$a_i = \mathbf{w}^\top \mathbf{x}_i + b$$



$$\hat{\mathbf{y}}_i = \begin{cases} 1 \text{ if } \sigma(a_i) \geq 0.5 \\ 0 \text{ else} \end{cases}$$

Sigmoid Function

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



Clean Up Bias Term $\mathbf{w}^\top \mathbf{x}_i + b$

Absorb bias term into feature vector:

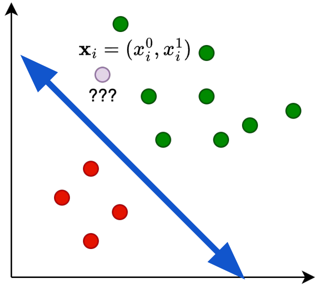
$$\mathbf{x}_i \text{ becomes } \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} \text{ and } \mathbf{w} \text{ becomes } \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$$

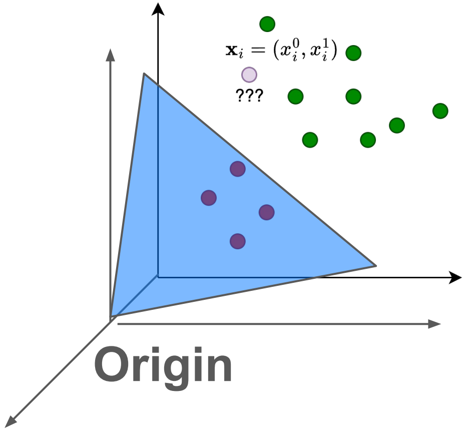
We can see that:

$$\begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}^\top \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} = \mathbf{w}^\top \mathbf{x}_i + b$$

Can rewrite logistic regression as

$$\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

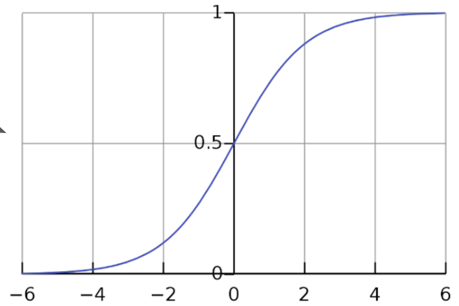




Cornell Bowers C·IS

Maximum Likelihood Estimation

$$\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$



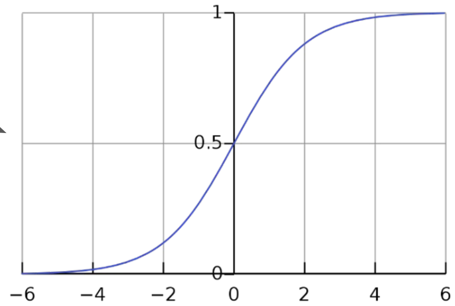
We want to find w to maximize the likelihood of the observed data

$(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{y}_i \in \{0, 1\}$

Cornell Bowers C·IS

Maximum Likelihood Estimation

$$\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$



We want to find w to maximize the likelihood of the observed data

$(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{y}_i \in \{0, 1\}$

$\rightarrow$ Minimize negative log likelihood loss (NLL loss)

Cornell Bowers C·IS

What are key components in ML?

- Training data
- Model Class / Hypothesis space
- Loss function
- Optimization

Cornell Bowers C·IS

Discuss: Why are they equivalent?

$$\max P_w(\text{Data})$$

data= $(\mathbf{x}_i, \mathbf{y}_i)$

$$\max P_w(y_1, y_2, \ldots, y_n | x_1, x_2, \ldots, x_n)$$

Conditional independent

$$\max \Pi_i P_w(y_i | x_i)$$

monotonic

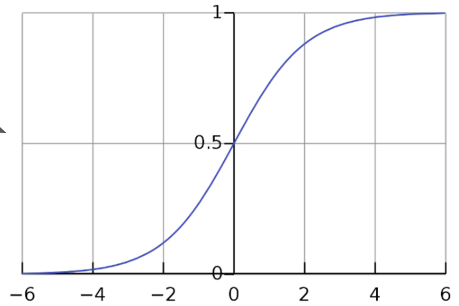
$$\max \sum_i \log P_w(y_i | x_i)$$

Flipping +/-

$$\min - \sum_i \log P_w(y_i | x_i)$$

Maximum Likelihood Estimation

$$\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$



We want to find w to maximize the likelihood of the observed data

$(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{y}_i \in \{0, 1\}$

Start by writing $p(\mathbf{y}_i|\mathbf{x}_i)$ using $\hat{\mathbf{y}}_i$ and $\mathbf{y}_i$

Derive the "Log Loss":

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -[\mathbf{y}_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i \qquad) + (1 - \mathbf{y}_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i \qquad))]$$

Maximum Likelihood Estimation

Maximize the likelihood of the observed data $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{y}_i \in \{0, 1\}$:

$$p(\mathbf{y}_i|\mathbf{x}_i) = \hat{\mathbf{y}}_i^{\mathbf{y}_i}(1 - \hat{\mathbf{y}}_i)^{1-\mathbf{y}_i}$$

Note that if $\mathbf{y}_i = 1$, then

$$p(\mathbf{y}_i|\mathbf{x}_i) = \hat{\mathbf{y}}_i$$

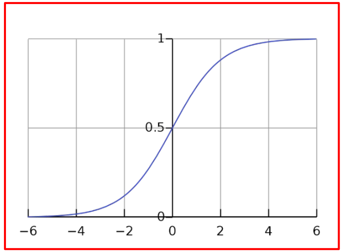
and if $\mathbf{y}_i = 0$, then

$$p(\mathbf{y}_i|\mathbf{x}_i) = 1 - \hat{\mathbf{y}}_i$$

Cornell Bowers CIS

Negative log-likelihood loss (NLL loss)

$$\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$



Maximizing the likelihood is equivalent to maximizing the log-likelihood:

$$\begin{aligned}\log p(\mathbf{y}_i|\mathbf{x}_i) &= \log[\hat{\mathbf{y}}_i^{\mathbf{y}_i}(1-\hat{\mathbf{y}}_i)^{1-\mathbf{y}_i})\\ &= \mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))\end{aligned}$$

Add a negative sign to turn it into a loss, i.e. something to minimize:

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

We can plug in our definition of $\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i + b)$:

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -[\mathbf{y}_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i + b) + (1-\mathbf{y}_i)\log(1-\sigma(\mathbf{w}^\top \mathbf{x}_i + b))]$$

Our Goal: Minimize the Loss

Given some training dataset:

$$\mathcal{D}_{\text{TR}} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=0}^{n}$$

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}; \mathcal{D}_{\text{TR}}) = \frac{1}{n}\sum_{i}^{n} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i)$$

$$= \frac{1}{n}\sum_{i}^{n} \ell(\sigma(\mathbf{w}^\top \mathbf{x}_i), \mathbf{y}_i)$$

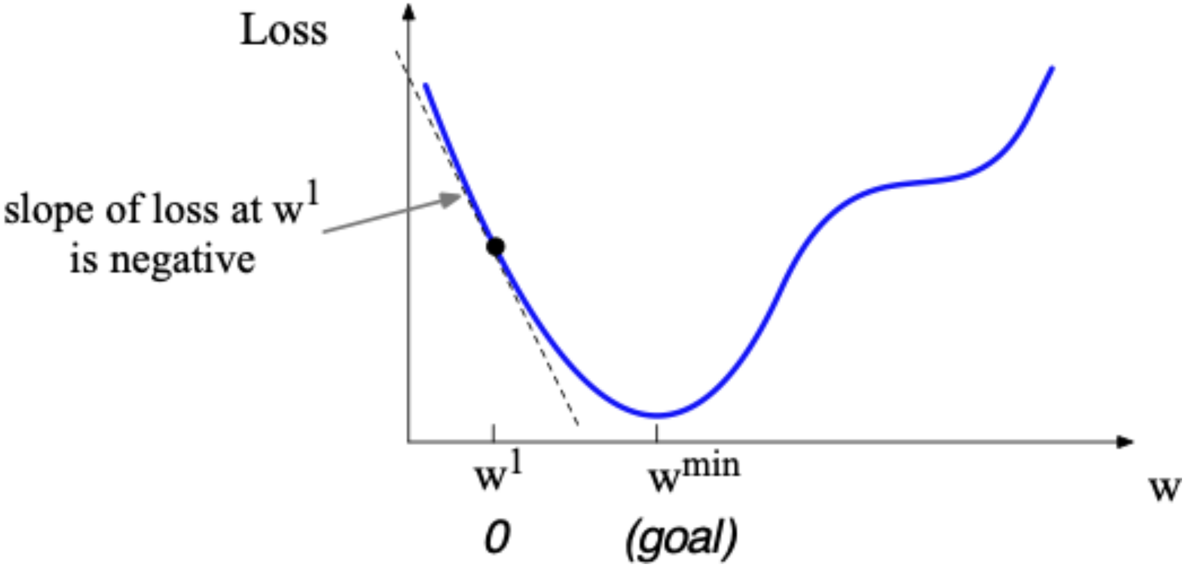
Gradient Descent

Visualize Gradient Descent in 1-D

https://web.stanford.edu/~jurafsky/slp3/

Cornell Bowers C·IS

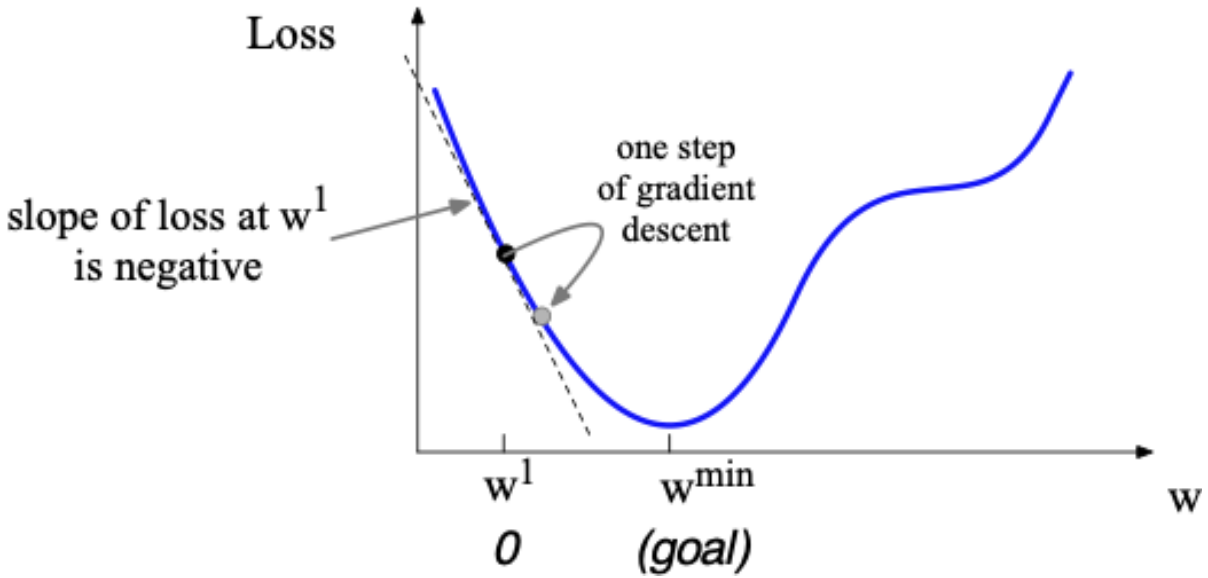
Visualize Gradient Descent in 1-D



https://web.stanford.edu/~jurafsky/slp3/

Cornell Bowers C·IS

Visualize Gradient Descent in 1-D

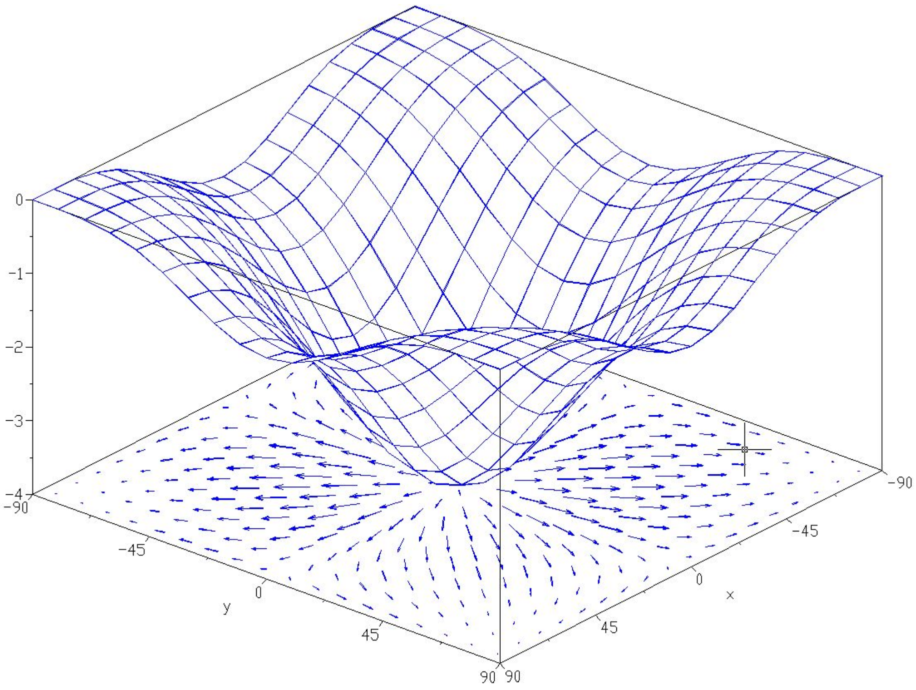


https://web.stanford.edu/~jurafsky/slp3/

Cornell Bowers C·IS

Gradients

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w};\mathcal{D}_{\mathrm{TR}}) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w^{(0)}}(\mathbf{w};\mathcal{D}_{\mathrm{TR}}) \\ \frac{\partial \mathcal{L}}{\partial w^{(1)}}(\mathbf{w};\mathcal{D}_{\mathrm{TR}}) \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial w^{(m)}}(\mathbf{w};\mathcal{D}_{\mathrm{TR}}) \end{bmatrix}, \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w};\mathcal{D}_{\mathrm{TR}}) \in \mathbb{R}^m$$



https://www.ml-science.com/gradients

Gradient Descent:

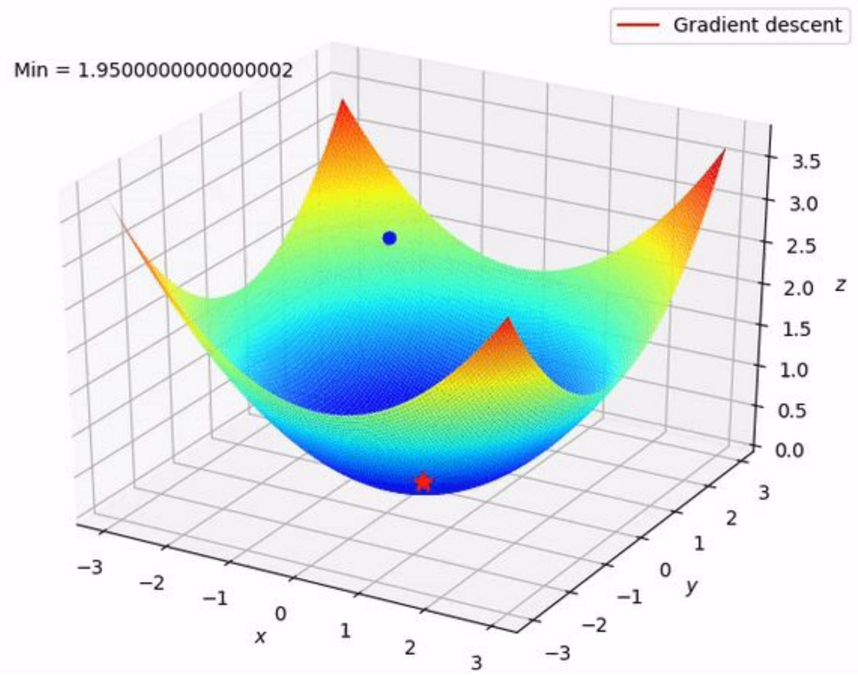
- Find the gradient at current point
- Move in **opposite** direction with learning rate α

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}_t}\mathcal{L}(\mathbf{w}_t;\mathcal{D}_{\mathrm{TR}})$$

Cornell Bowers C·IS

Gradient Descent (GD)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}_t} \mathcal{L}(\mathbf{w}_t; \mathcal{D}_{\mathrm{TR}})$$



Cornell Bowers C·IS

What are key components in ML?

- Training data
- Model Class / Hypothesis space
- Loss function
- Optimization

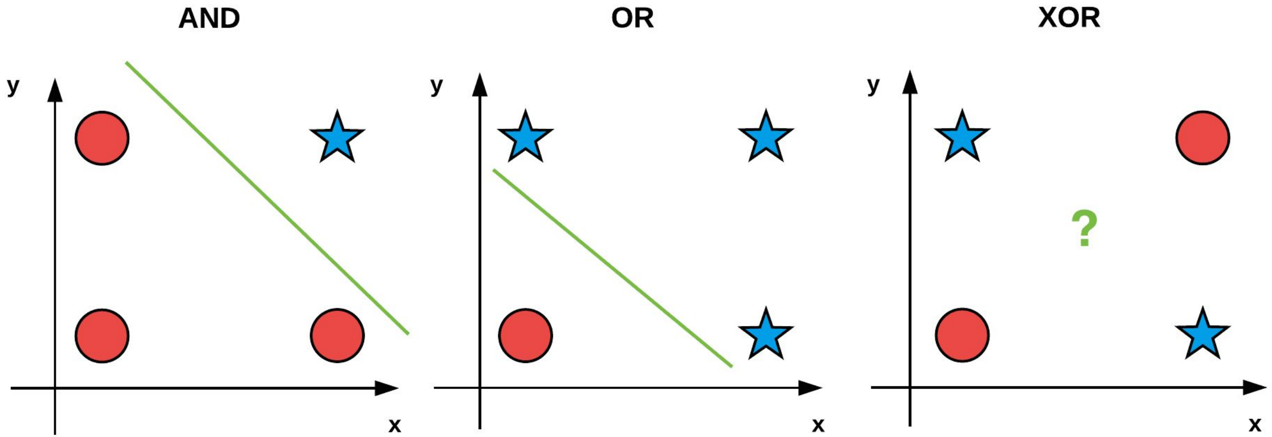
Demo: Logistic Regression

- Tensorflow Playground

Cornell Bowers C·IS

Demo: The XOR Problem

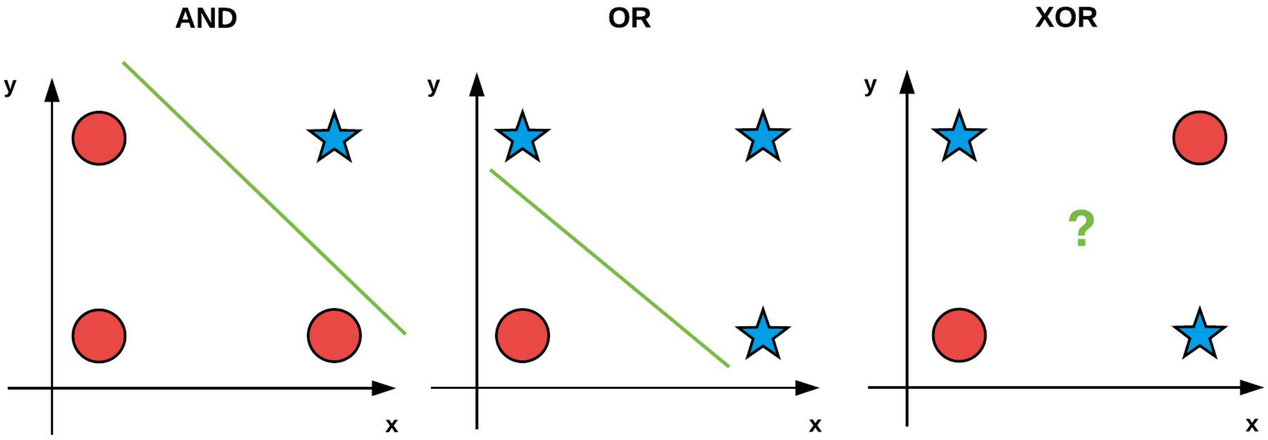
- Tensorflow Playground



Cornell Bowers C·IS

The XOR Problem

- Perceptron can’t learn the XOR function
  - Simple logical operation
- Data is not linearly separable

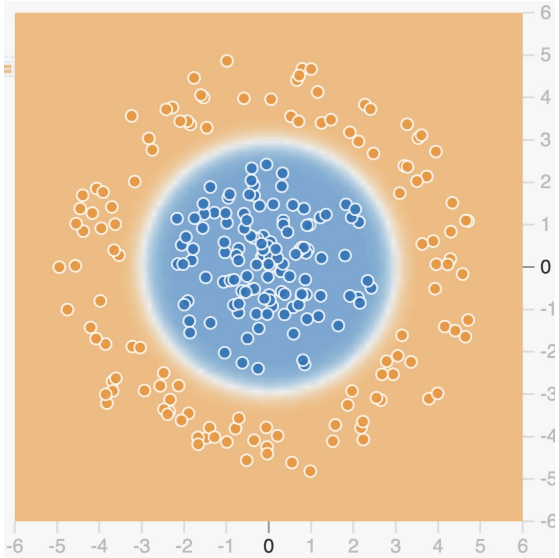


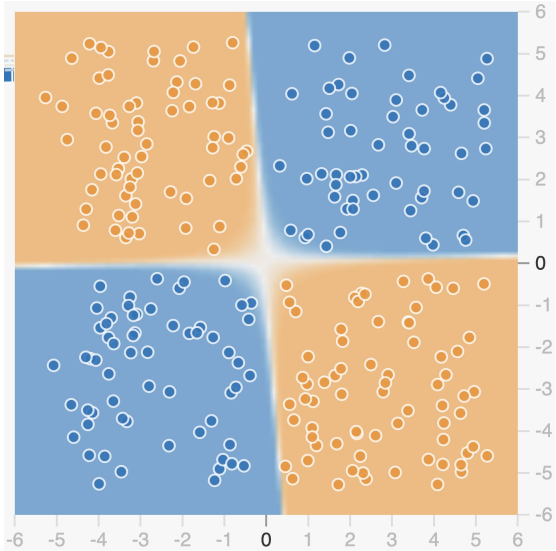
https://www.pyimagesearch.com/2021/05/06/implementing-the-perceptron-neural-network-with-python/

Cornell Bowers C·IS

Discuss: What are some ways to handle data that is not linearly separable?

Without deep learning!





Possible Solutions

- Feature engineering
  - Construct a feature space where the data is linearly separable
- Kernel methods
  - Implicitly project the data into a higher-dimensional space where it is linearly separable
- Non-linear classifiers
  - E.g. Nearest neighbor, decision tree algorithms

Demo: Feature Engineering

[Tensorflow Playground]

Cornell Bowers C·IS

Feature Engineering


input image

classification →

"dog"


input image

classification →

"cat"

Cornell Bowers C·IS

Agenda

- Perceptron
- Logistic Regression
- Gradient Descent
- **Multi-Layer Perceptrons (MLPs)**
- Backpropagation

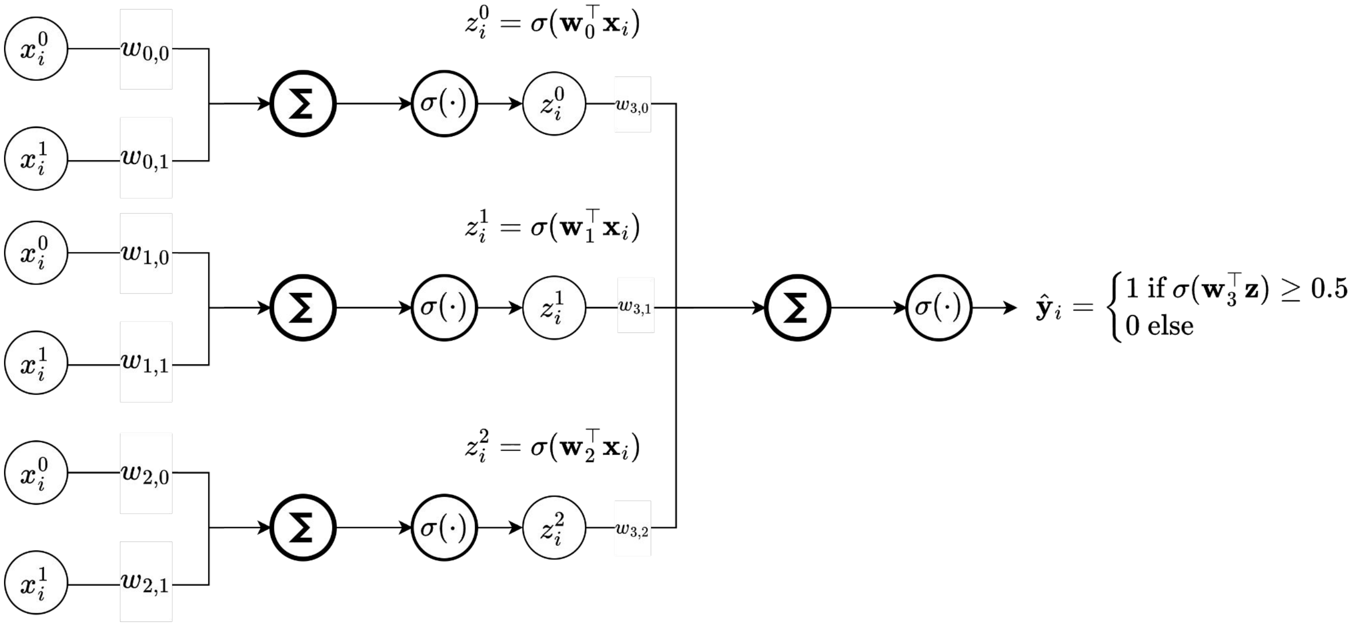
Multi-Layer Perceptron (MLP)

- Compose multiple perceptrons to **learn** intermediate features

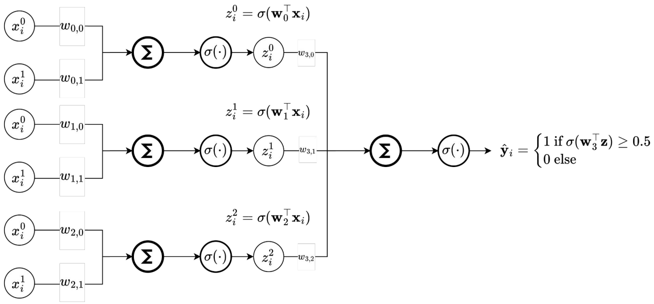
Multi-Layer Perceptron (MLP)

- Compose multiple perceptrons to **learn** intermediate features

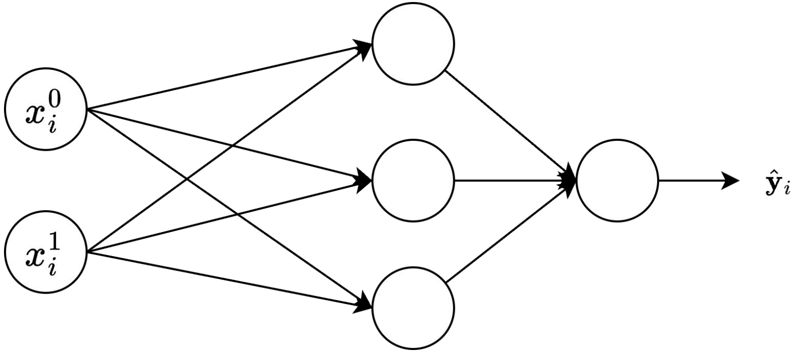
An MLP with 1 hidden layer with 3 hidden units



A Simplified MLP Diagram

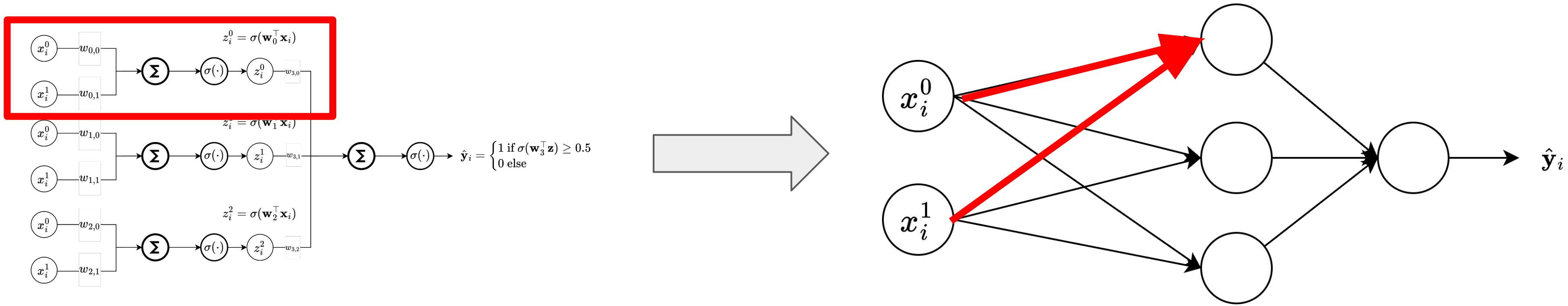


1 Hidden Layer,
3 Hidden Units



A Simplified MLP Diagram

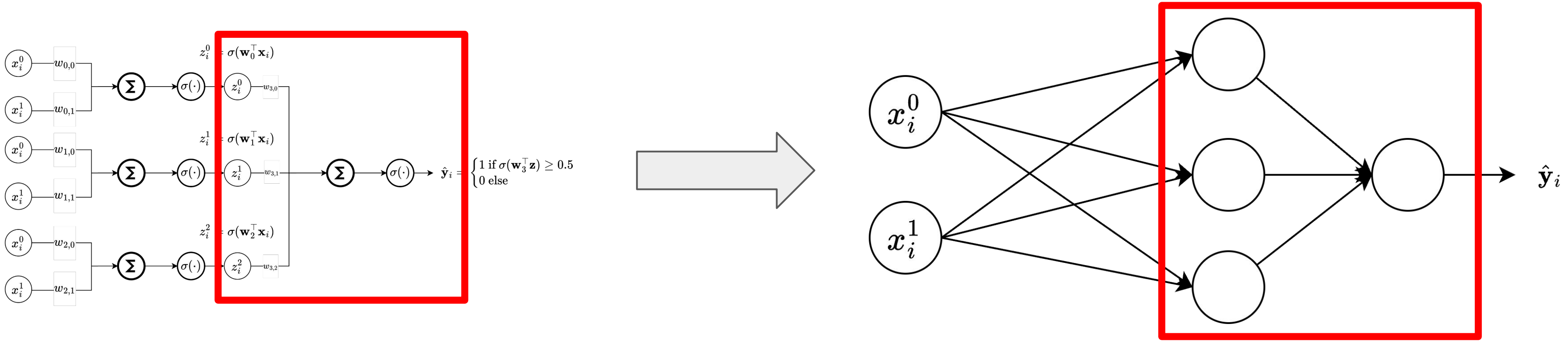
1 Hidden Layer,
3 Hidden Units



Cornell Bowers C·IS

A Simplified MLP Diagram

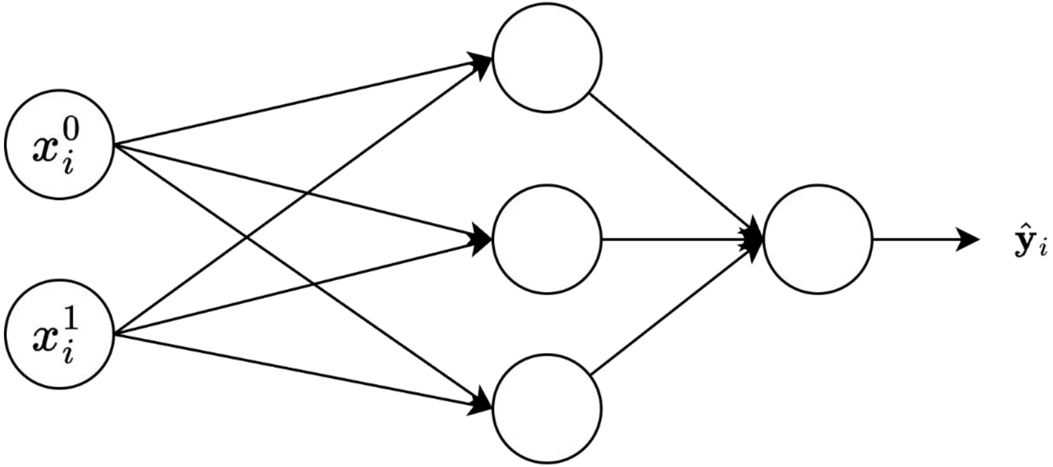
1 Hidden Layer,
3 Hidden Units

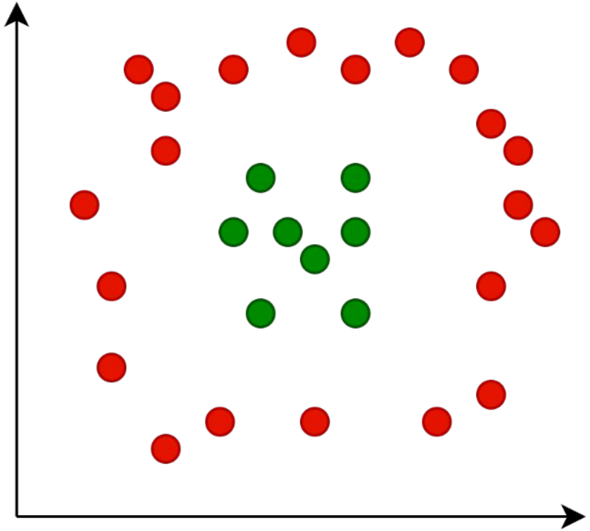


Cornell Bowers C·IS

Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers



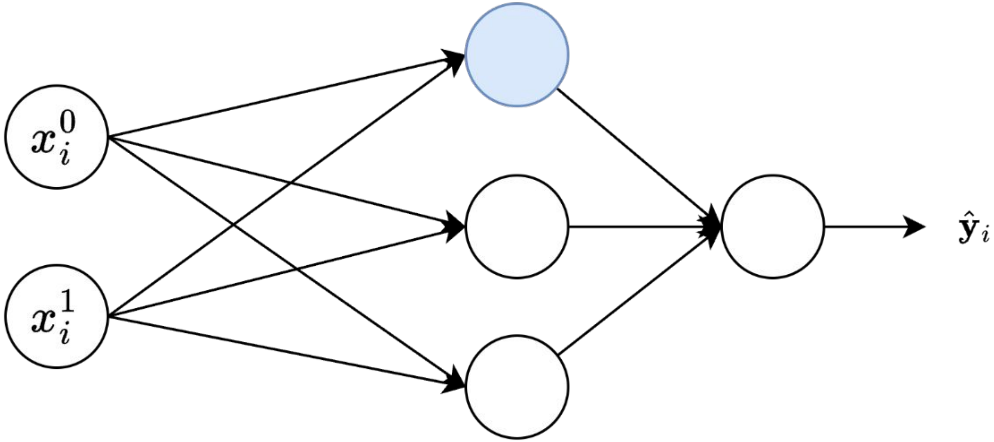


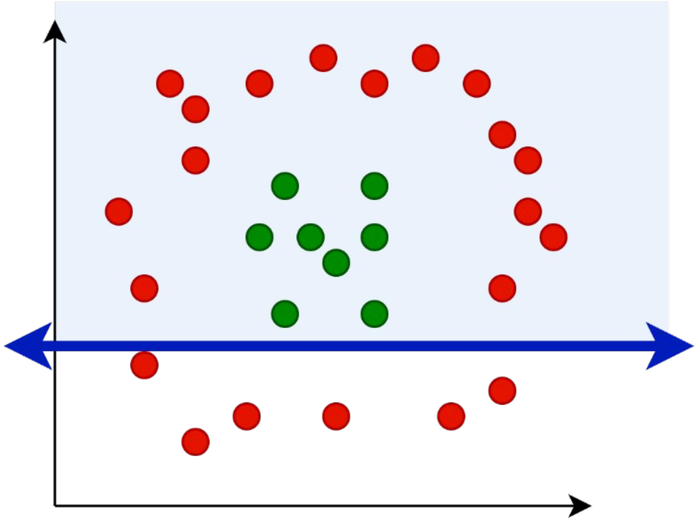
Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers

Recall:

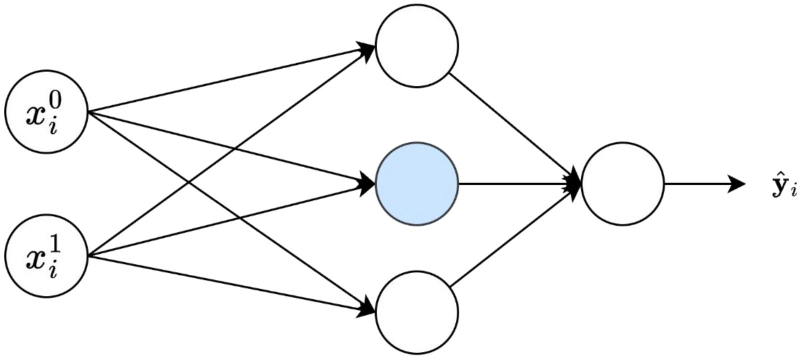
$$y_i = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$

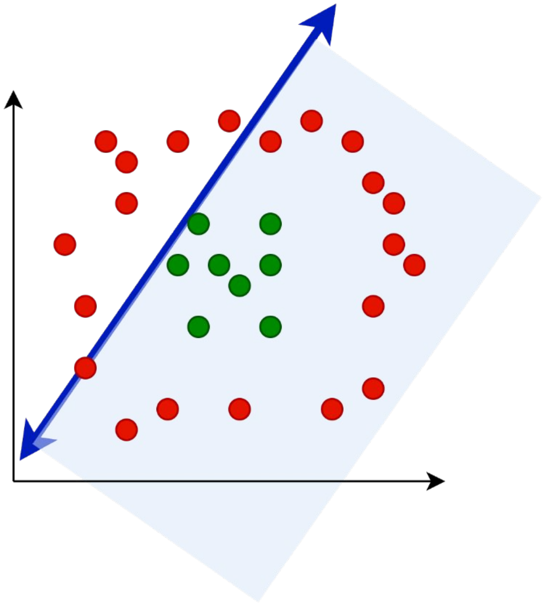




Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers

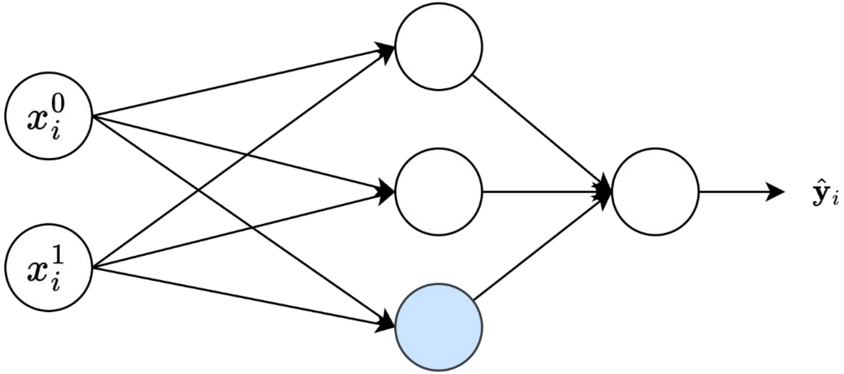


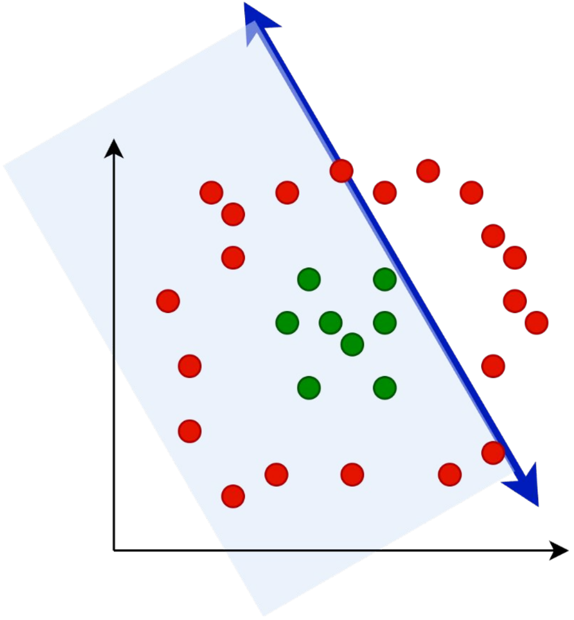


Cornell Bowers C·IS

Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers

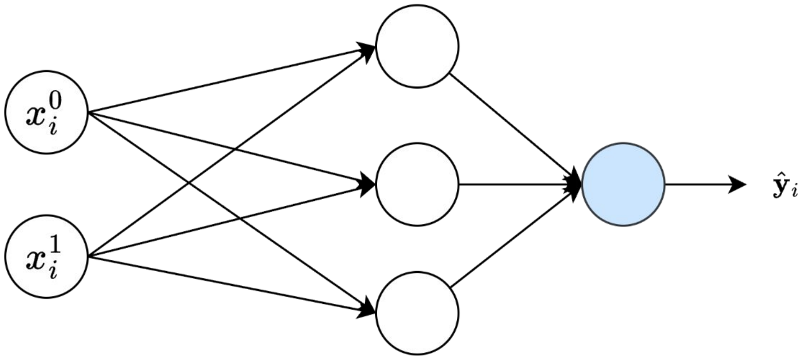


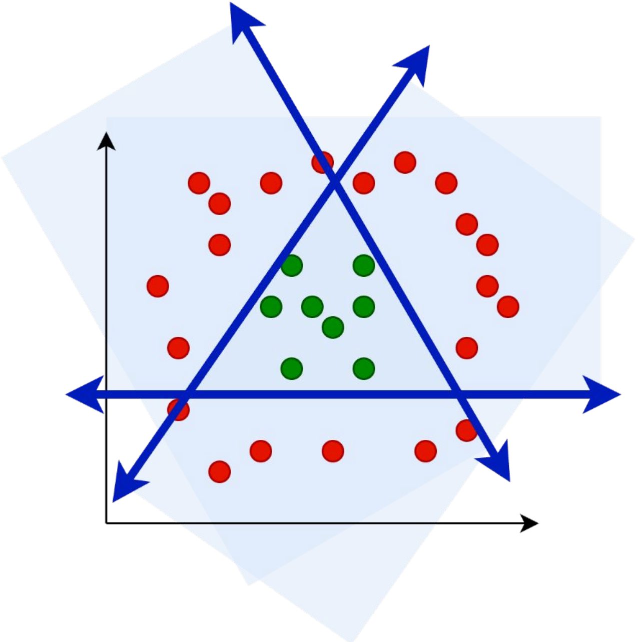


Cornell Bowers C·IS

Discuss: Why this works?

- What does this extra layer give us?
    - Can compose multiple linear classifiers





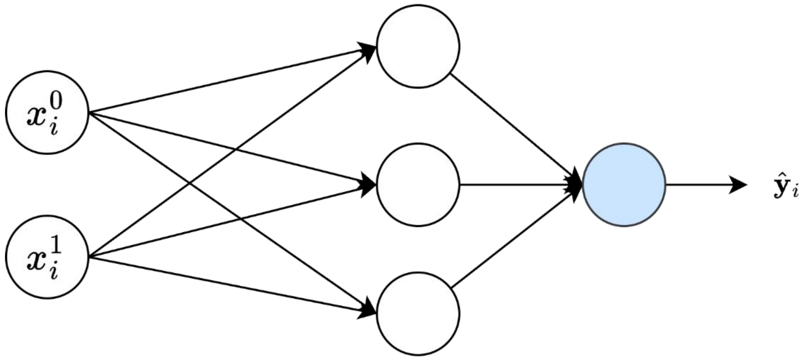
Cornell Bowers C·IS

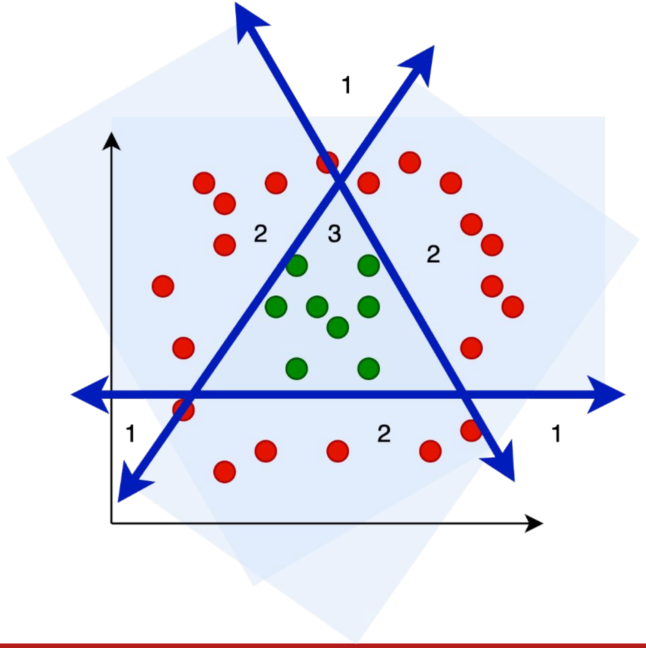
Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers

Recall:

$$y_i = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$





Complex Decision Boundaries

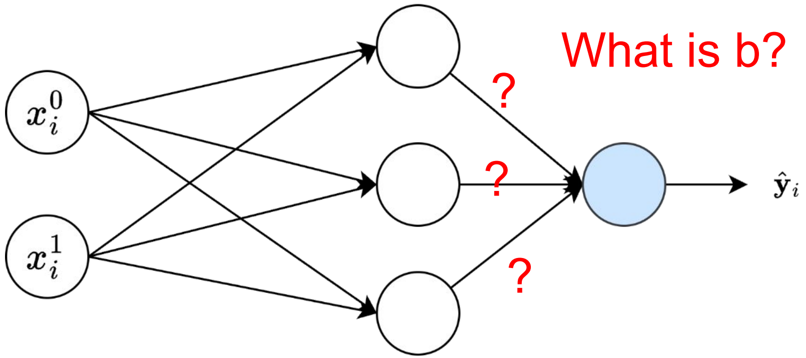
- What does this extra layer give us?
  - Can compose multiple linear classifiers

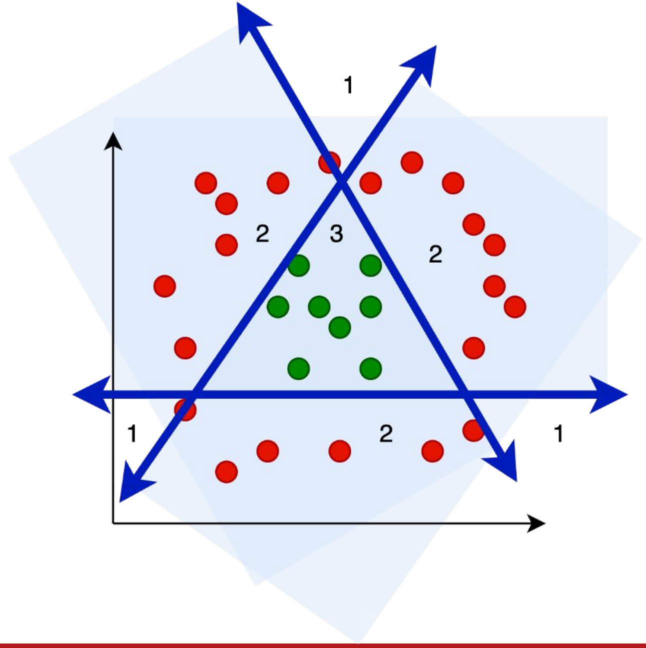
Recall:

$$y_i = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$

What is w?

What is b?





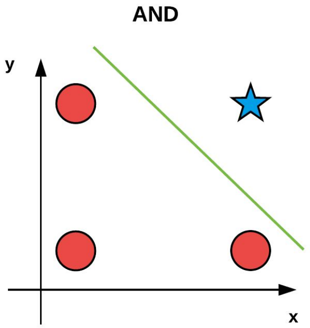
Cornell Bowers C·IS

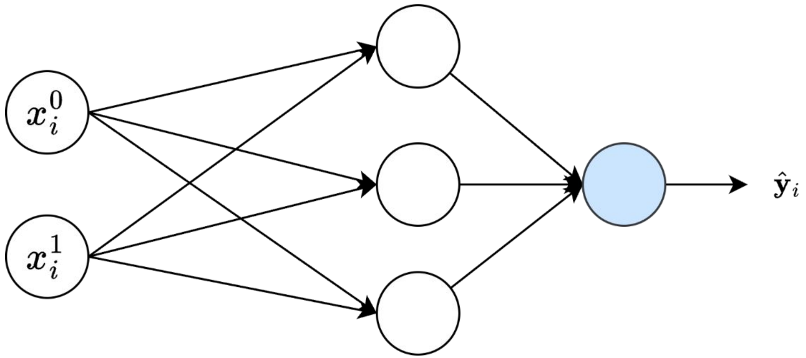
Complex Decision Boundaries

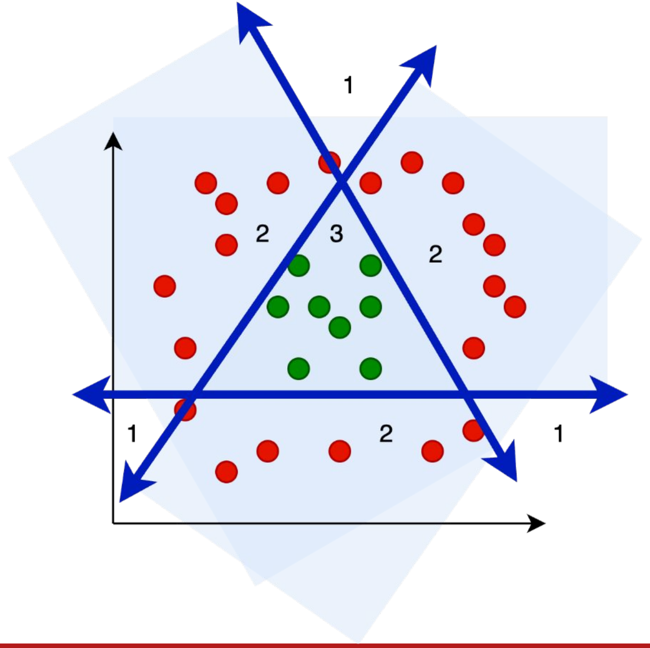
- What does this extra layer give us?
  - Can compose multiple linear classifiers

Recall:

$$y_i = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$





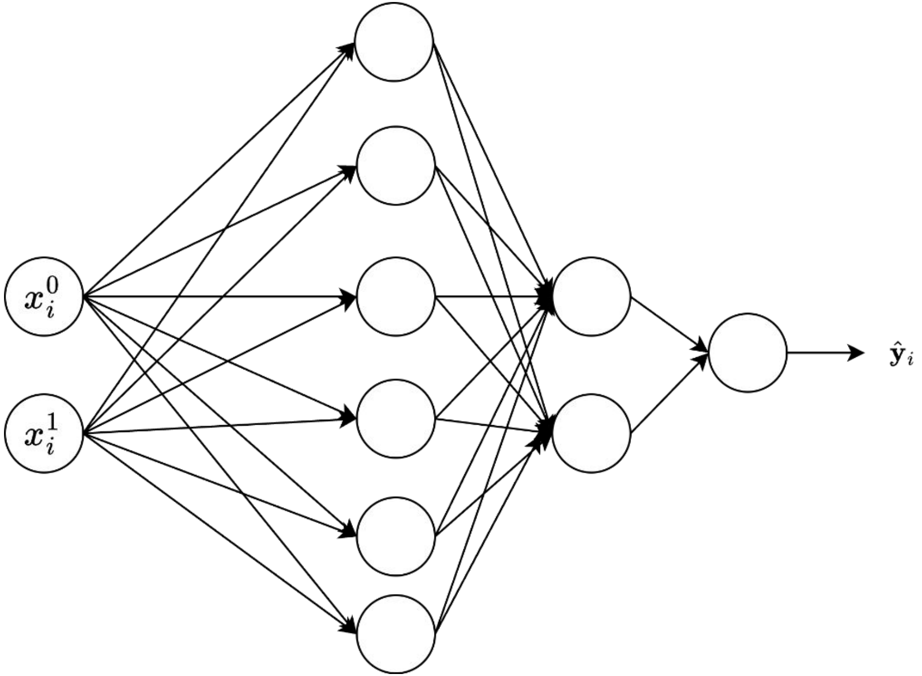


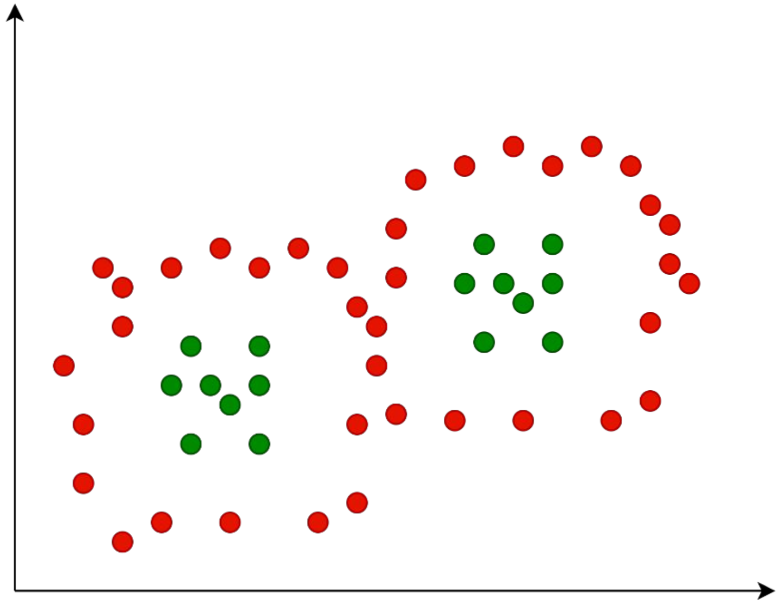
MLP Demo (1 Hidden Layer)

[Tensorflow Playground]()

Increasing Depth

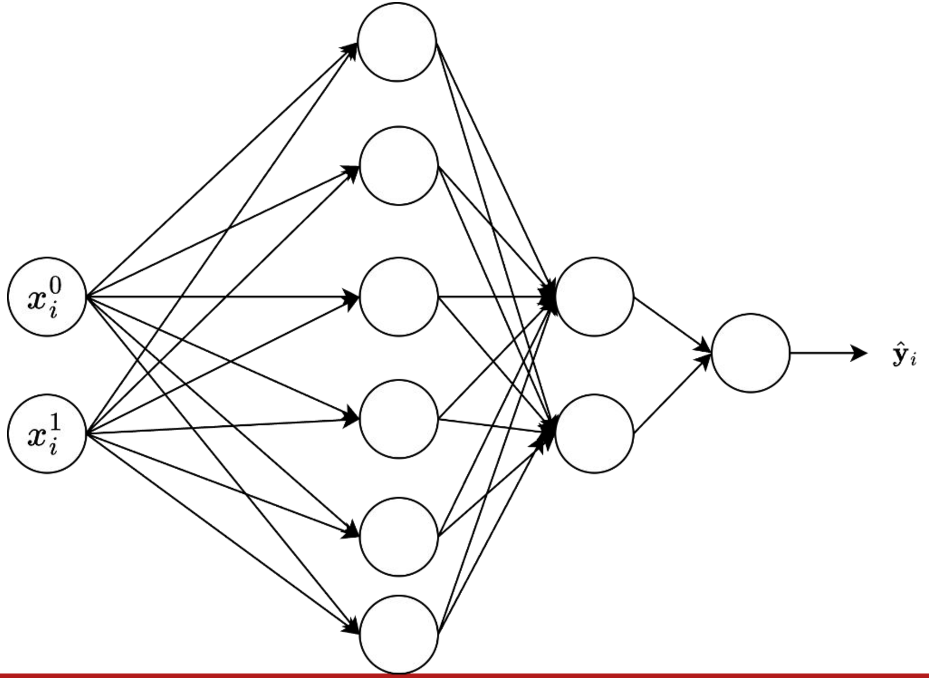
Discuss: How to construct the decision boundary?

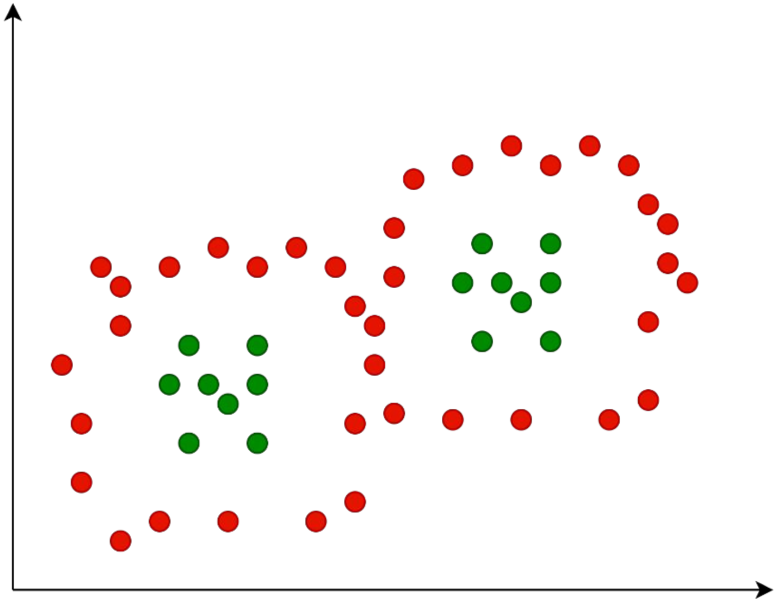




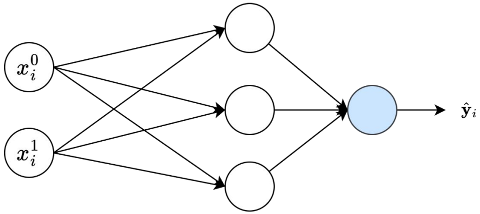
Increasing Depth

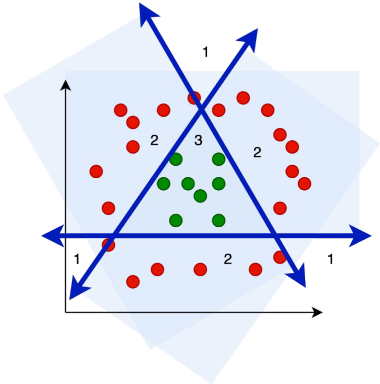
- MLP with 1 hidden layer composes linear classifiers
- MLP with 2 hidden layers can compose polygon classifiers

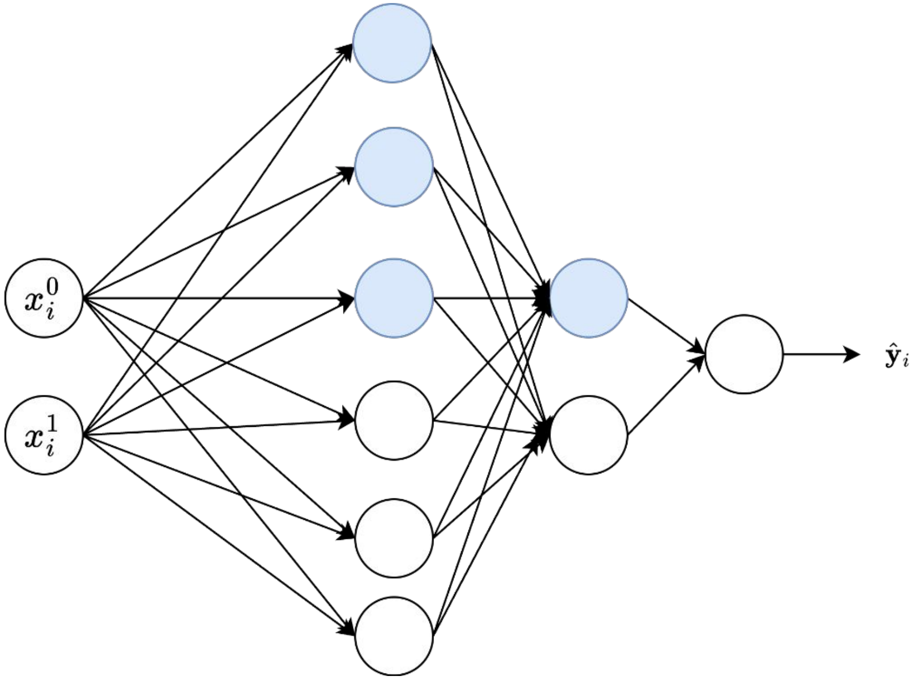


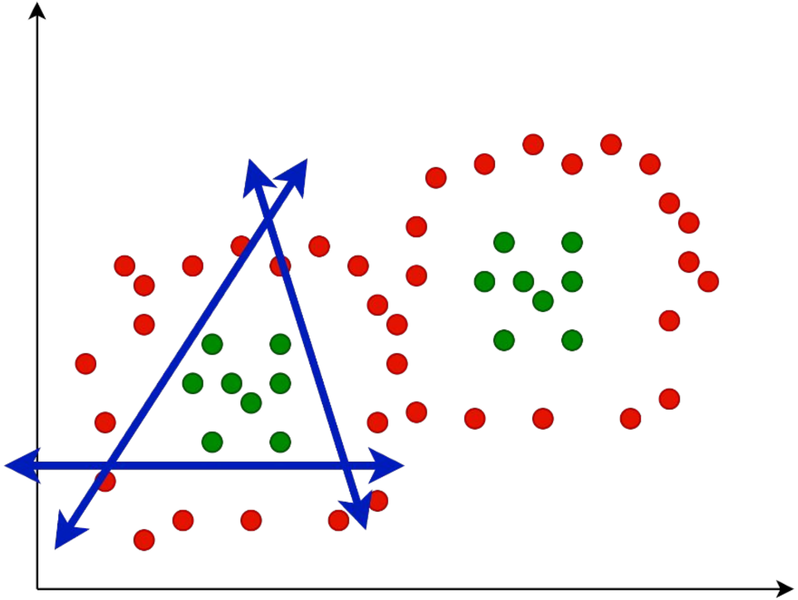


Increasing Depth

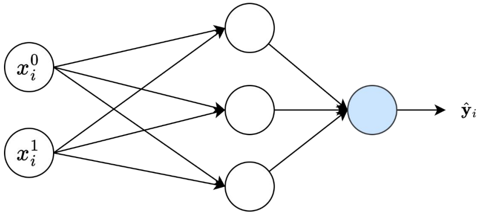


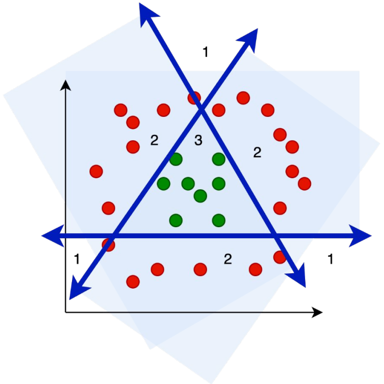


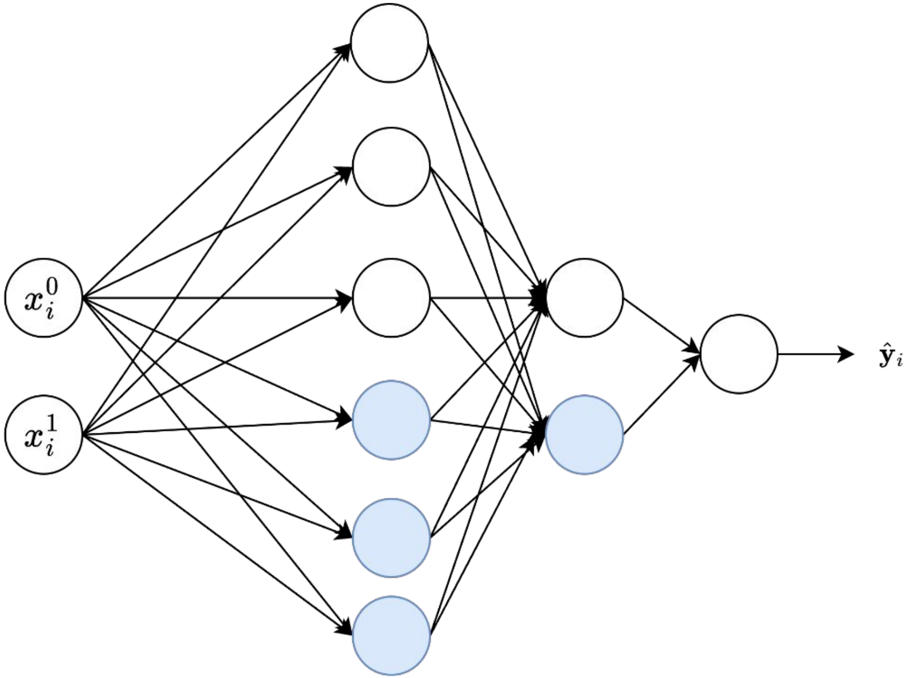


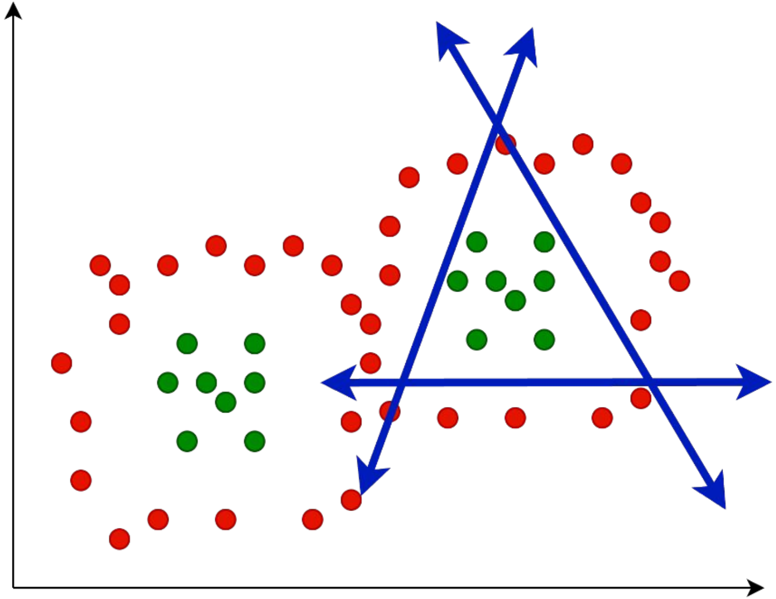


Increasing Depth

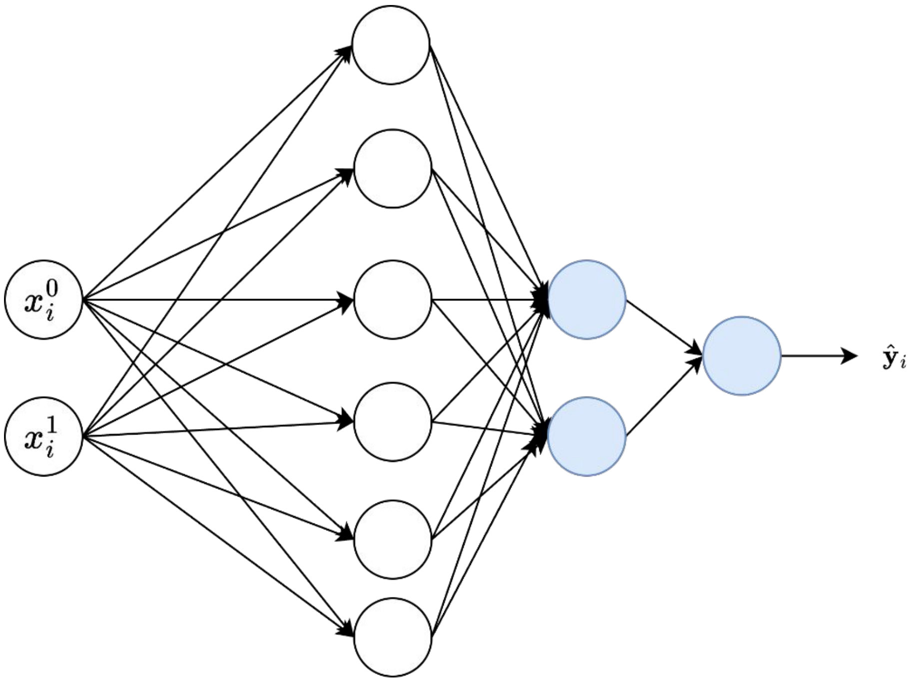


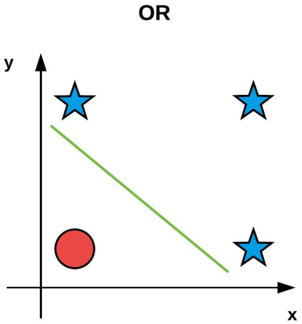


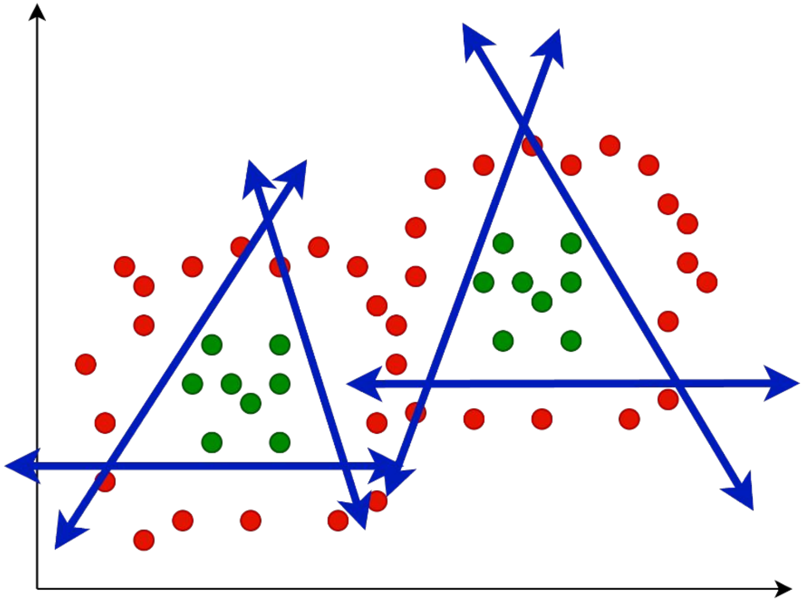




Increasing Depth

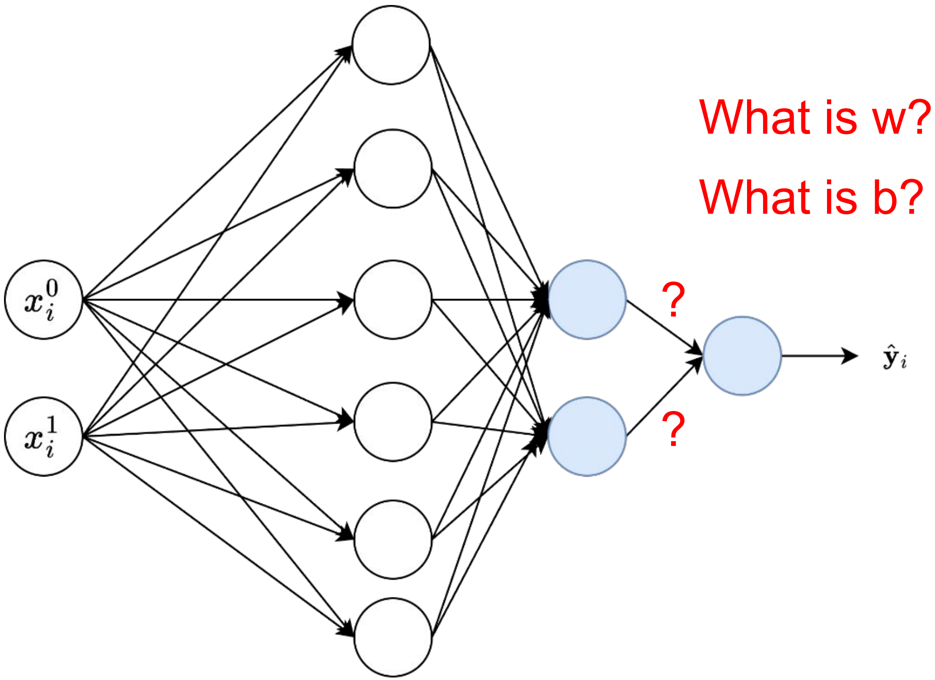






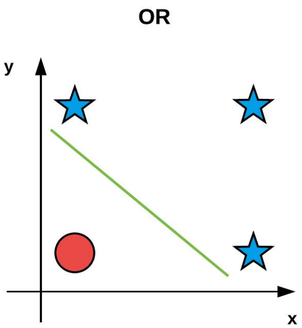
Cornell Bowers C·IS

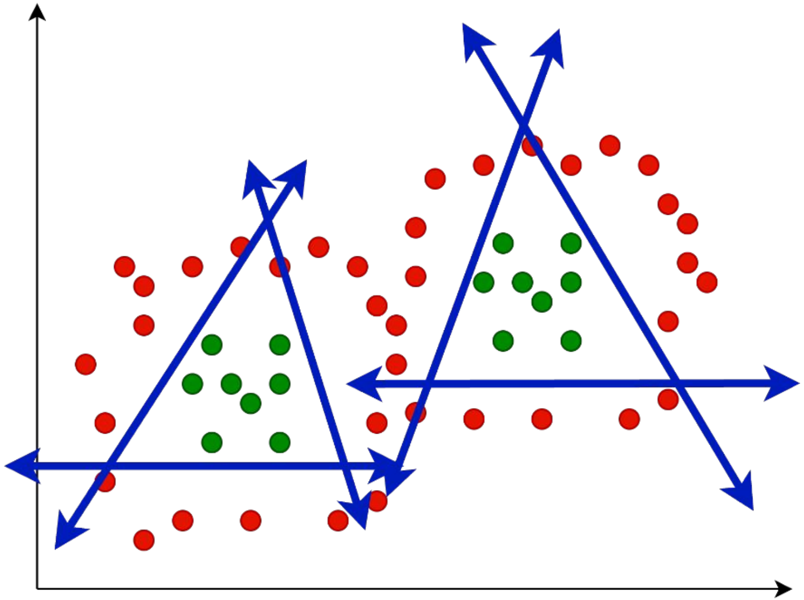
Increasing Depth



What is w?

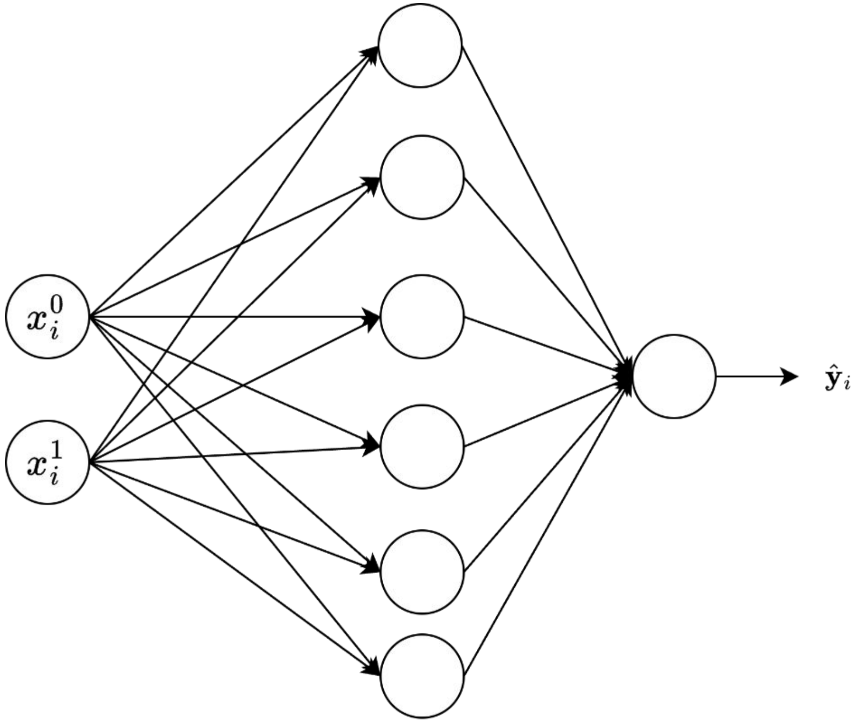
What is b?

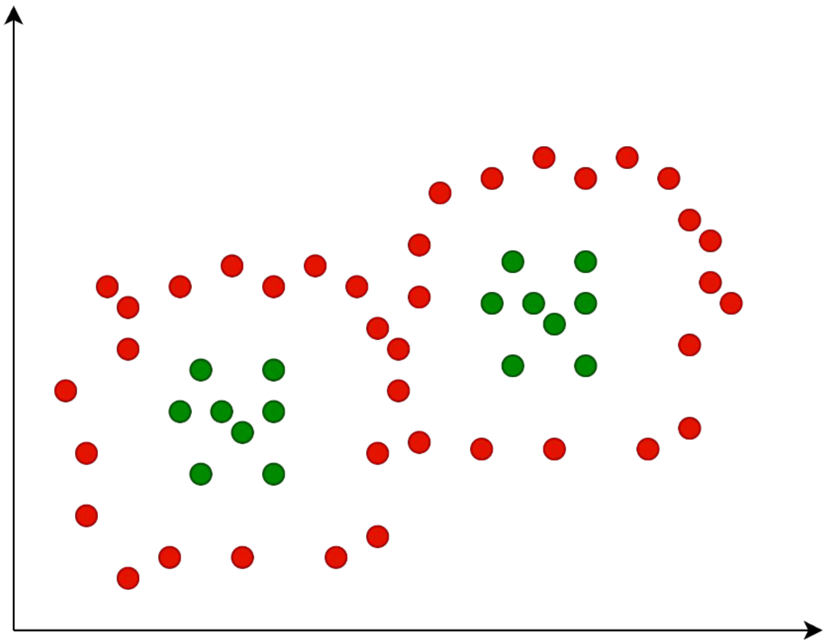




Cornell Bowers C·IS

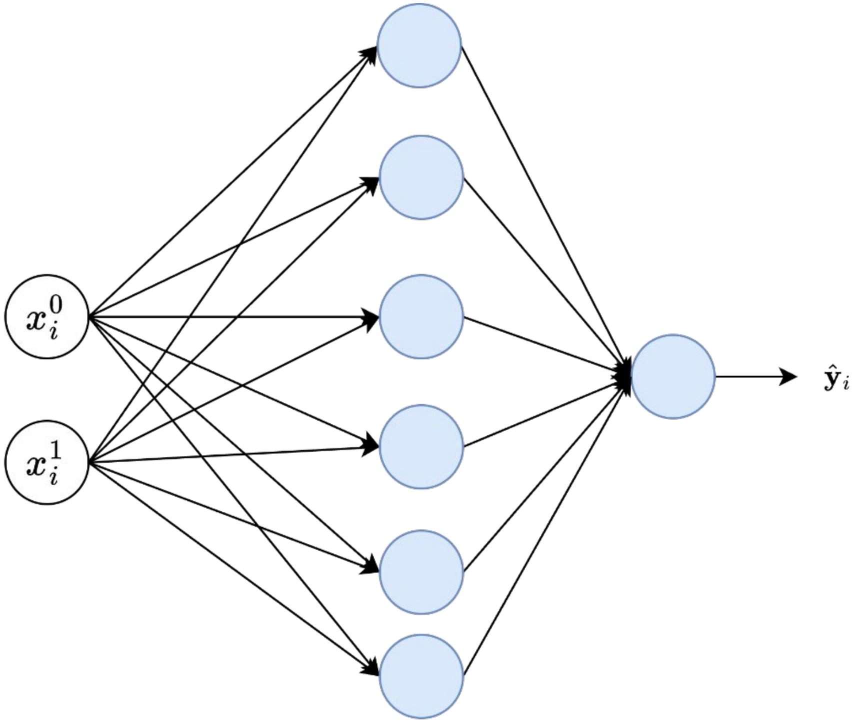
Discuss: What about just one layer?

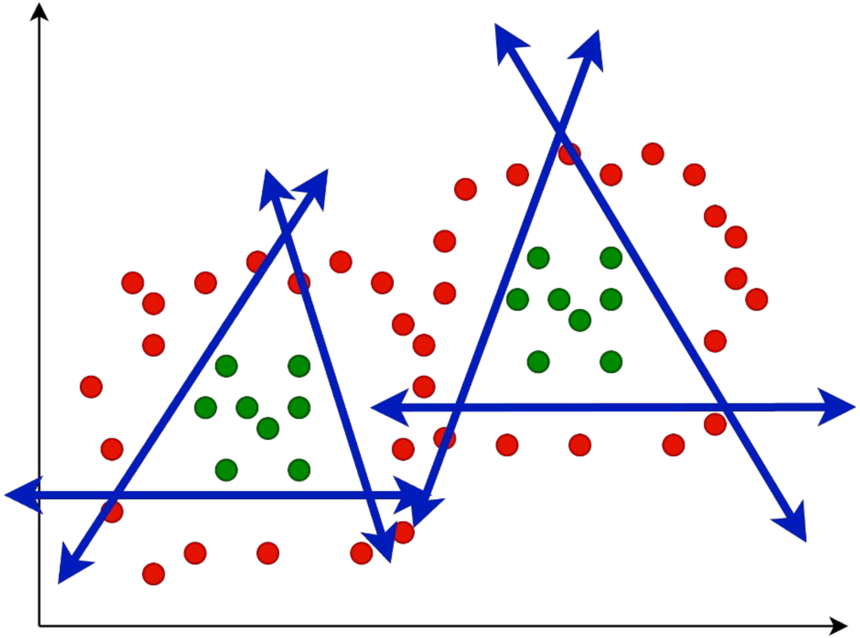




Cornell Bowers CIS

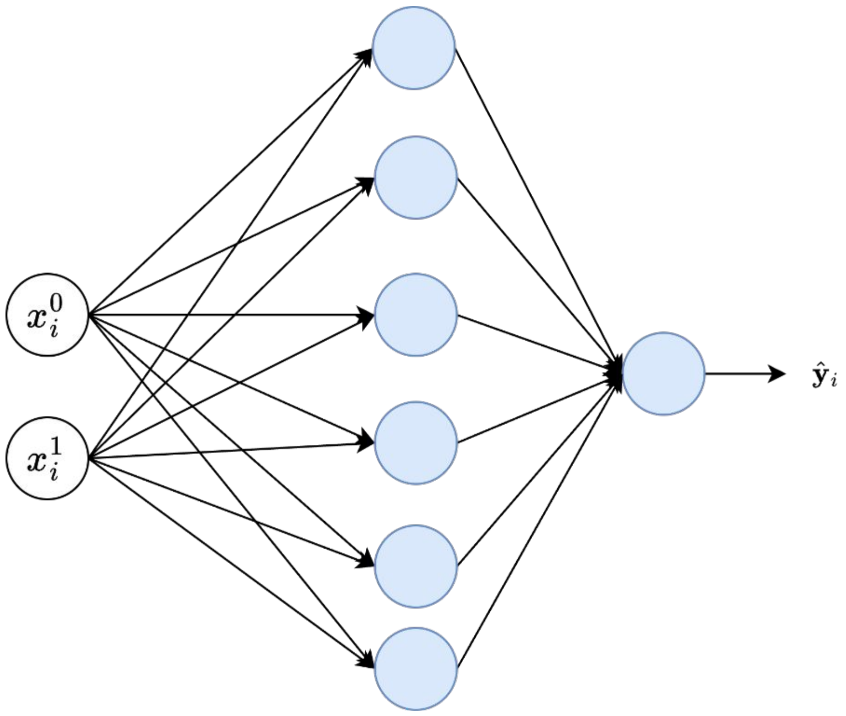
What about just one layer?

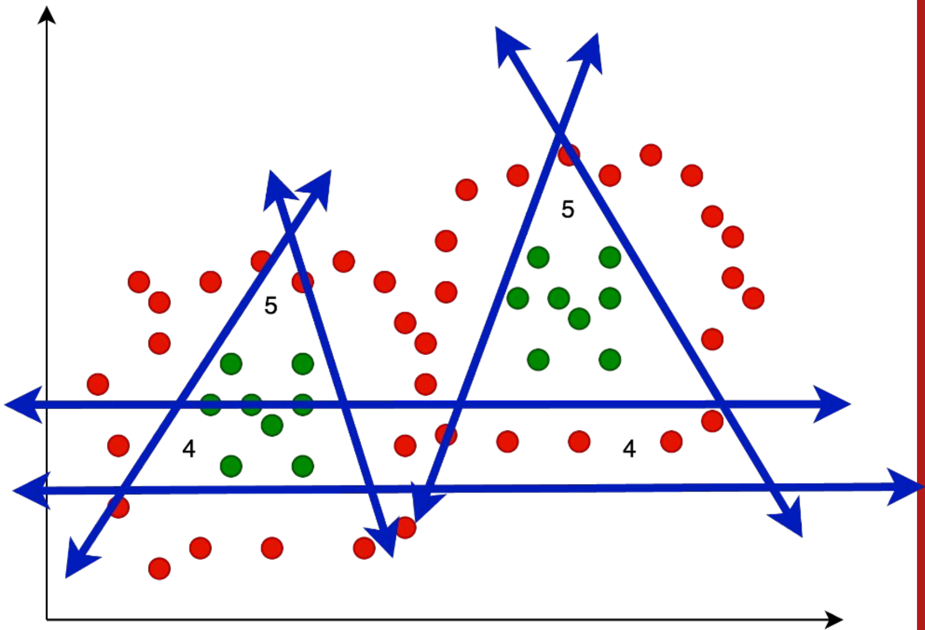




Cornell Bowers C·IS

What about just one layer?

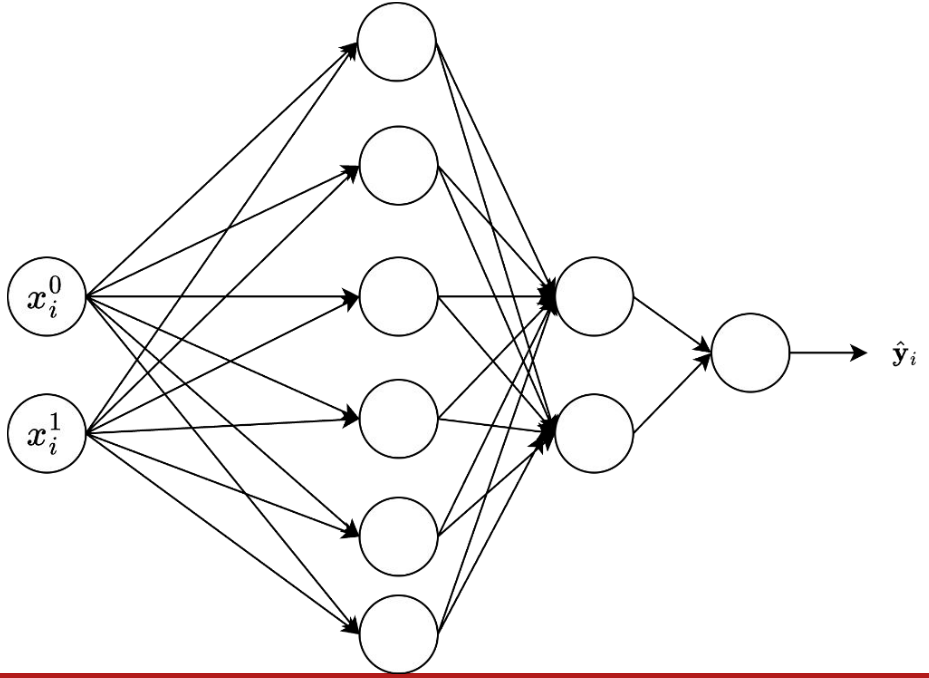


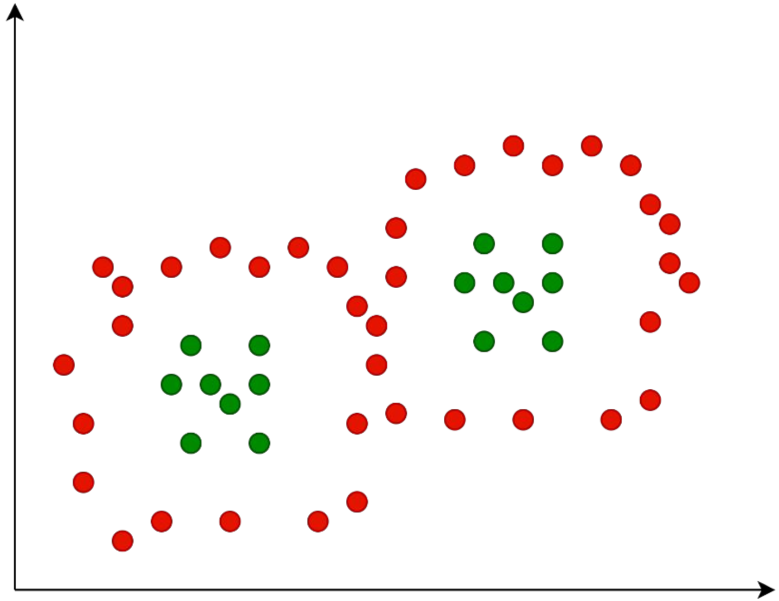


Cornell Bowers C·IS

Increasing Depth

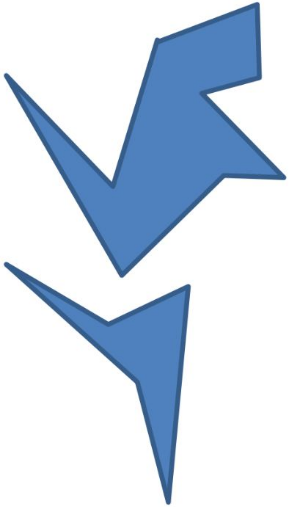
- MLP with 1 hidden layer composes linear classifiers
- MLP with 2 hidden layers can compose polygon classifiers

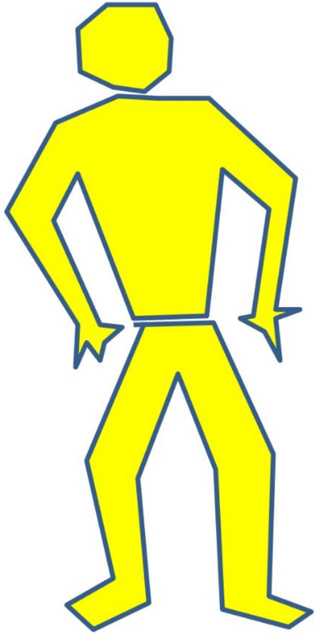


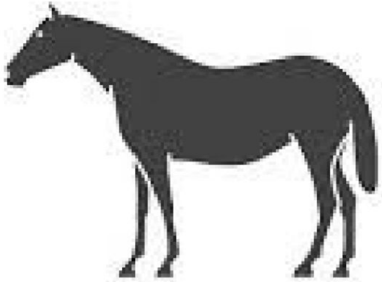


Complex Decision Boundaries

- Can compose *arbitrarily* complex decision boundaries





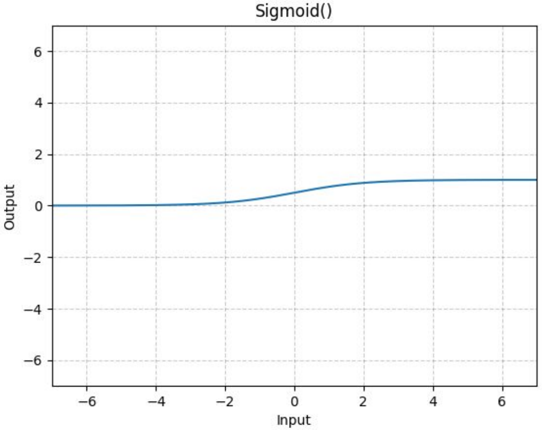


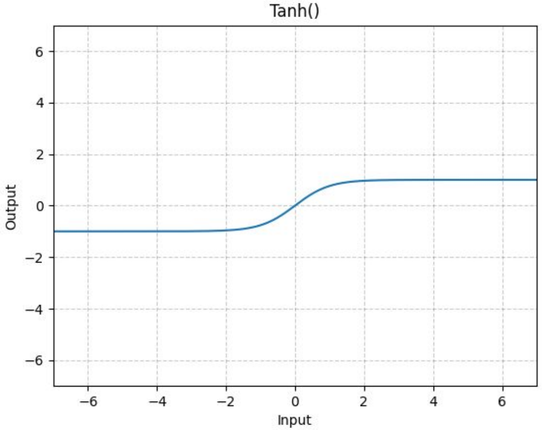
https://deeplearning.cs.cmu.edu/S24/document/slides/lec2.universal.pdf

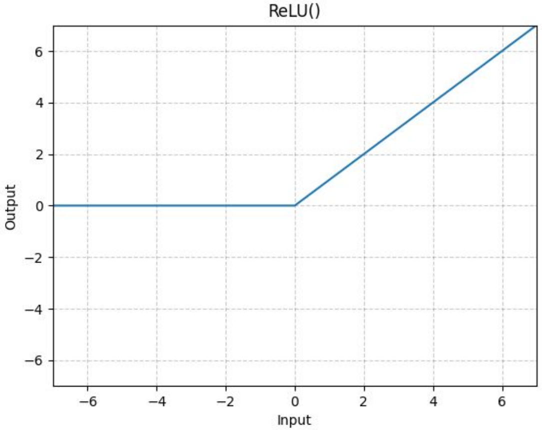
Cornell Bowers C·IS

Activation Functions

- Can replace the sigmoid with other nonlinear functions
  - Still universal approximators!







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

Squash between 0 and 1

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$$

Squash between -1 and 1

$$\mathrm{ReLU}(x) = \max(0, x)$$

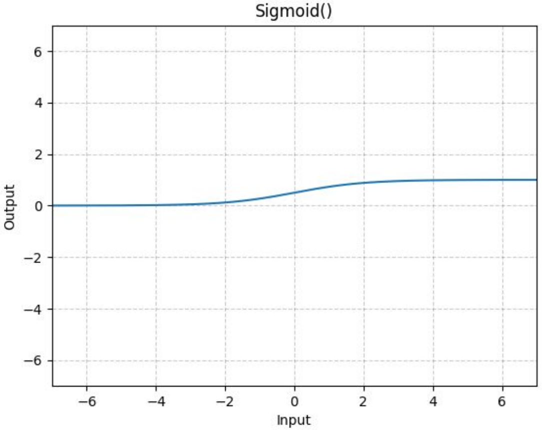
Threshold at 0

https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity

Cornell Bowers C·IS

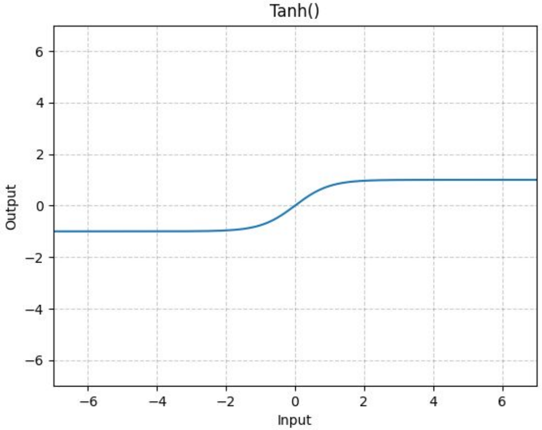
Activation Functions

- Can replace the sigmoid with other nonlinear functions
    - Still universal approximators!



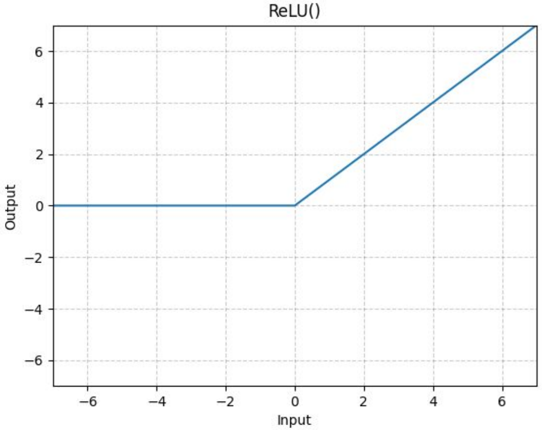
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Squash between 0 and 1



$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$$

Squash between -1 and 1



$$\mathrm{ReLU}(x) = \max(0, x)$$

Threshold at 0

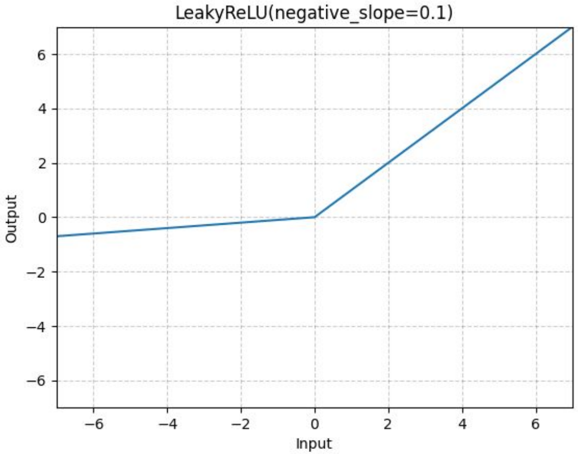
https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity

Cornell Bowers CIS

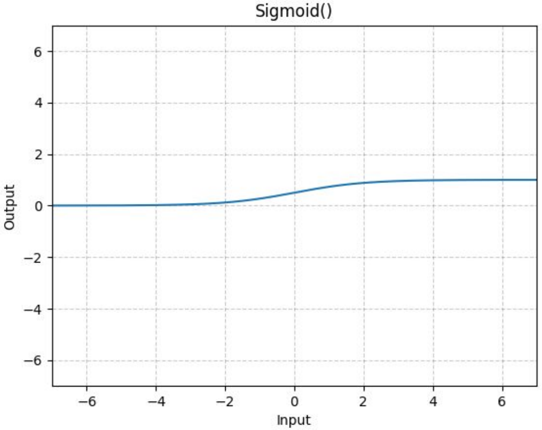
Activation Functions

Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research

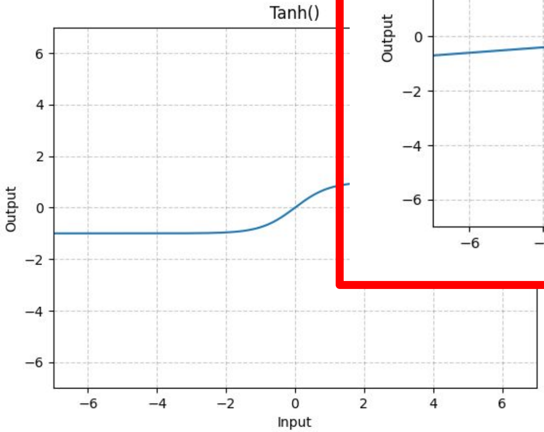


- Can replace the sigmoid with other nonline[...]
  - Still universal approximators!



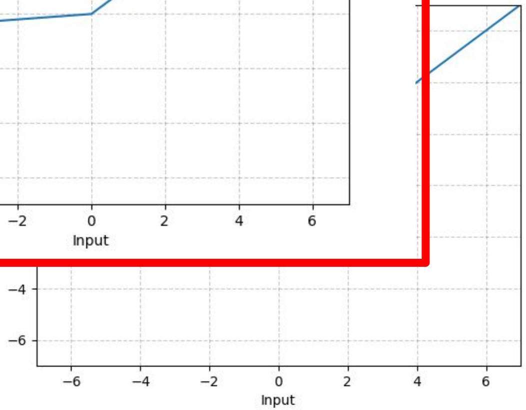
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Squash between 0 and 1



$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$$

Squash between -1 and 1



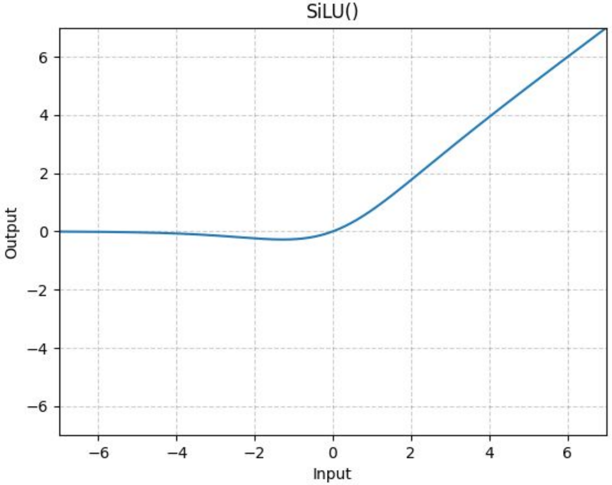
$$\mathrm{ReLU}(x) = \max(0, x)$$

Threshold at 0

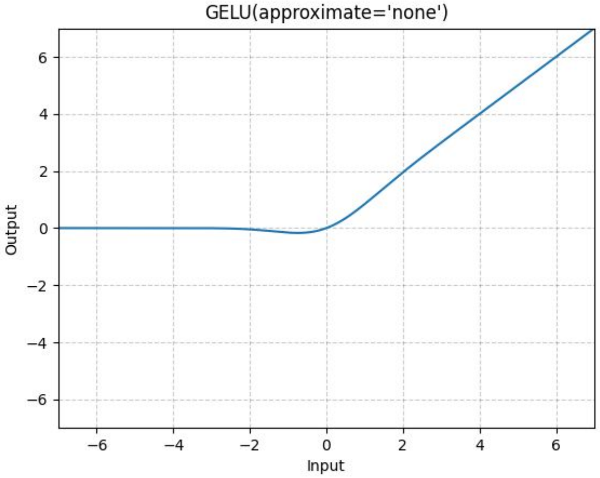
https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity

Activation Functions

- Can replace the sigmoid with other nonlinear functions
  - Still universal approximators!



$\mathrm{silu}(x) = x * \sigma(x)$



$\mathrm{GELU}(x) = x * \Phi(x)$

https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity

MLP Demo (3 Hidden Layers)

Tensorflow Playground

How to learn MLP weights?

Gradient descent!

Calculus Review: The Chain Rule

Lagrange's Notation: If $h(x) = f(g(x))$, then $h' = f'(g(x))g'(x)$

Leibniz's Notation: If $z = h(y), y = g(x)$, then $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

Calculus Review: The Chain Rule

Lagrange's Notation: If $h(x) = f(g(x))$, then $h' = f'(g(x))g'(x)$

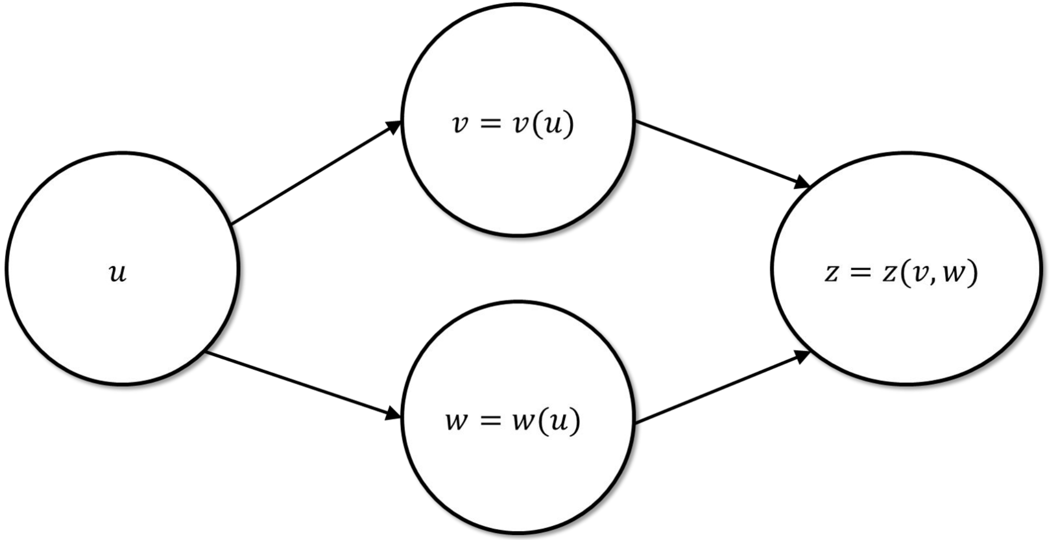
Leibniz's Notation: If $z = h(y), y = g(x)$, then $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

Example: If $z = \ln(y), y = x^2$, then

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$
$$= (\frac{1}{y})(2x) = (\frac{1}{x^2})(2x) = \frac{2}{x}$$

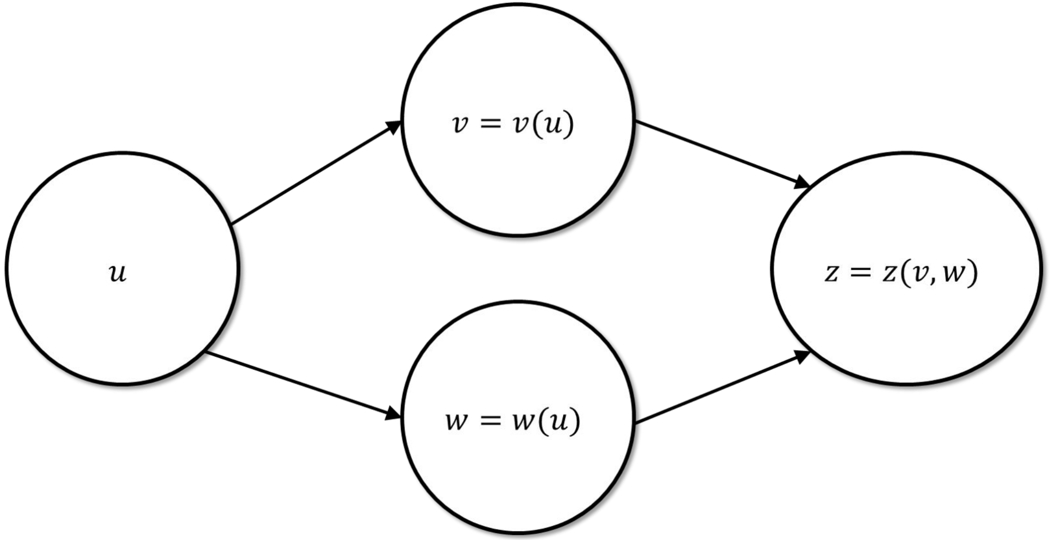
Cornell Bowers CIS

Multivariate Chain Rule



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Multivariate Chain Rule



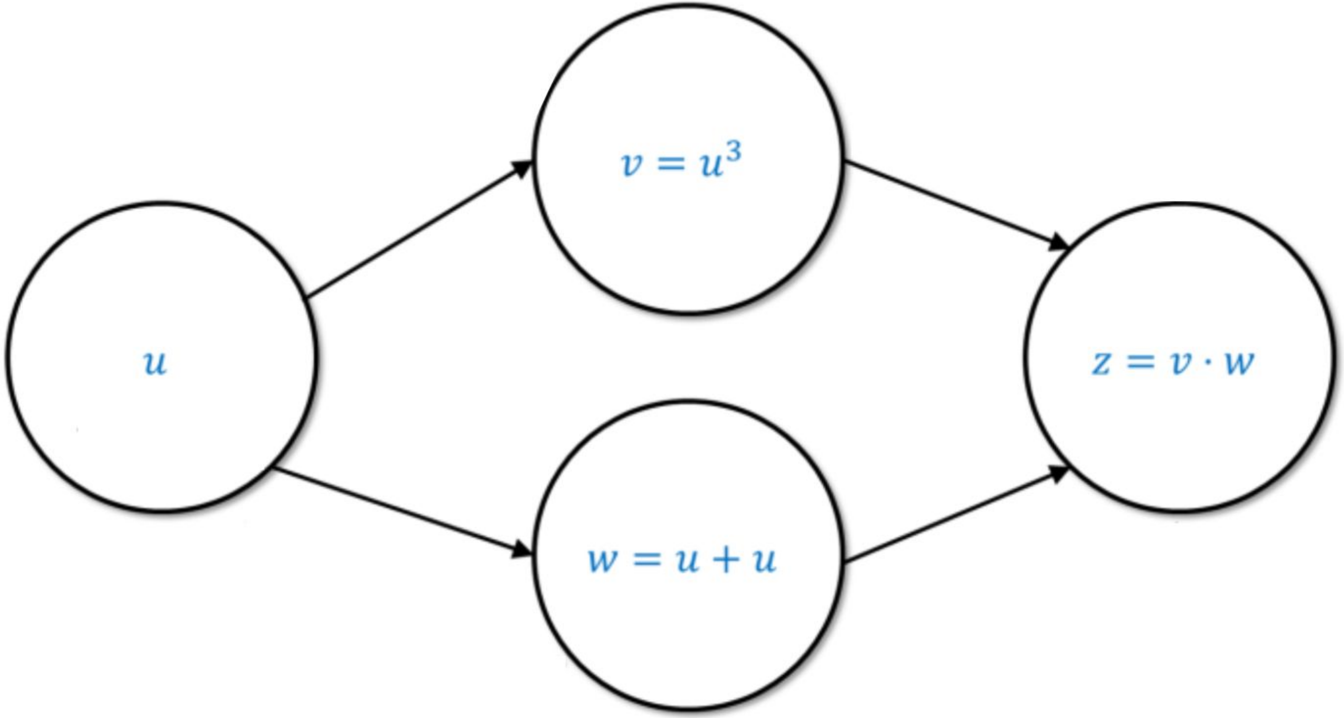
If $f(u)$ is $z = f(v(u), w(u))$, then

$$\frac{\partial f}{\partial u} = (\frac{\partial v}{\partial u}\frac{\partial z}{\partial v} + \frac{\partial w}{\partial u}\frac{\partial z}{\partial w})$$

https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Cornell Bowers C·IS

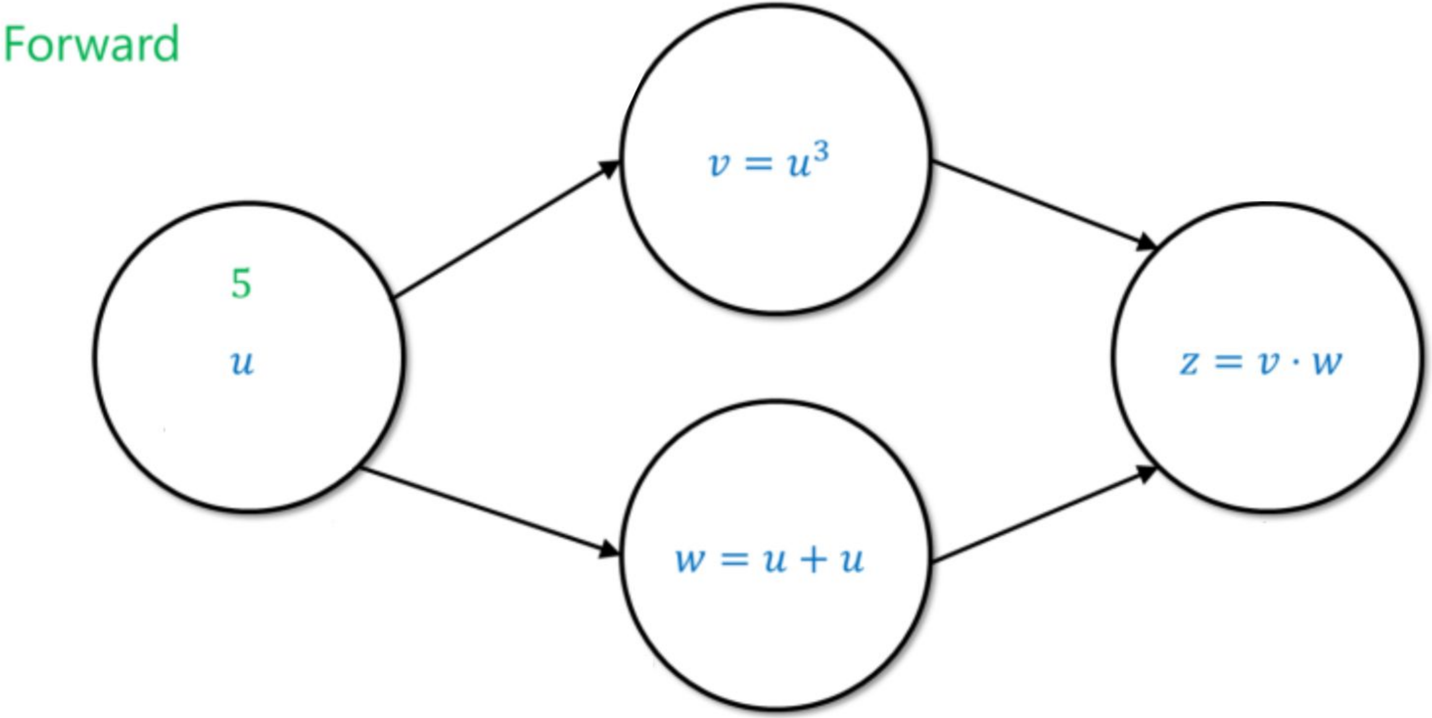
Backpropagation- An Example



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

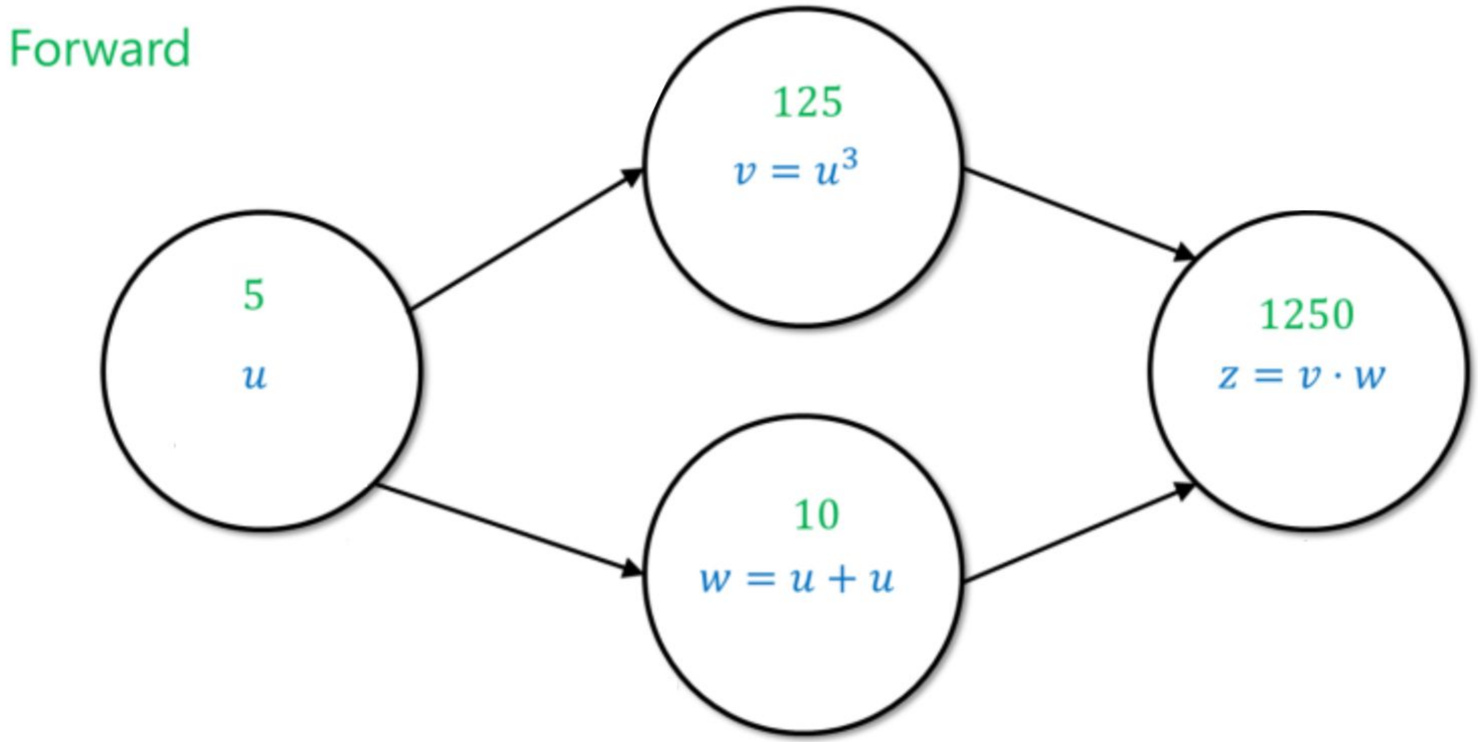
Cornell Bowers C·IS

Backpropagation- An Example



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

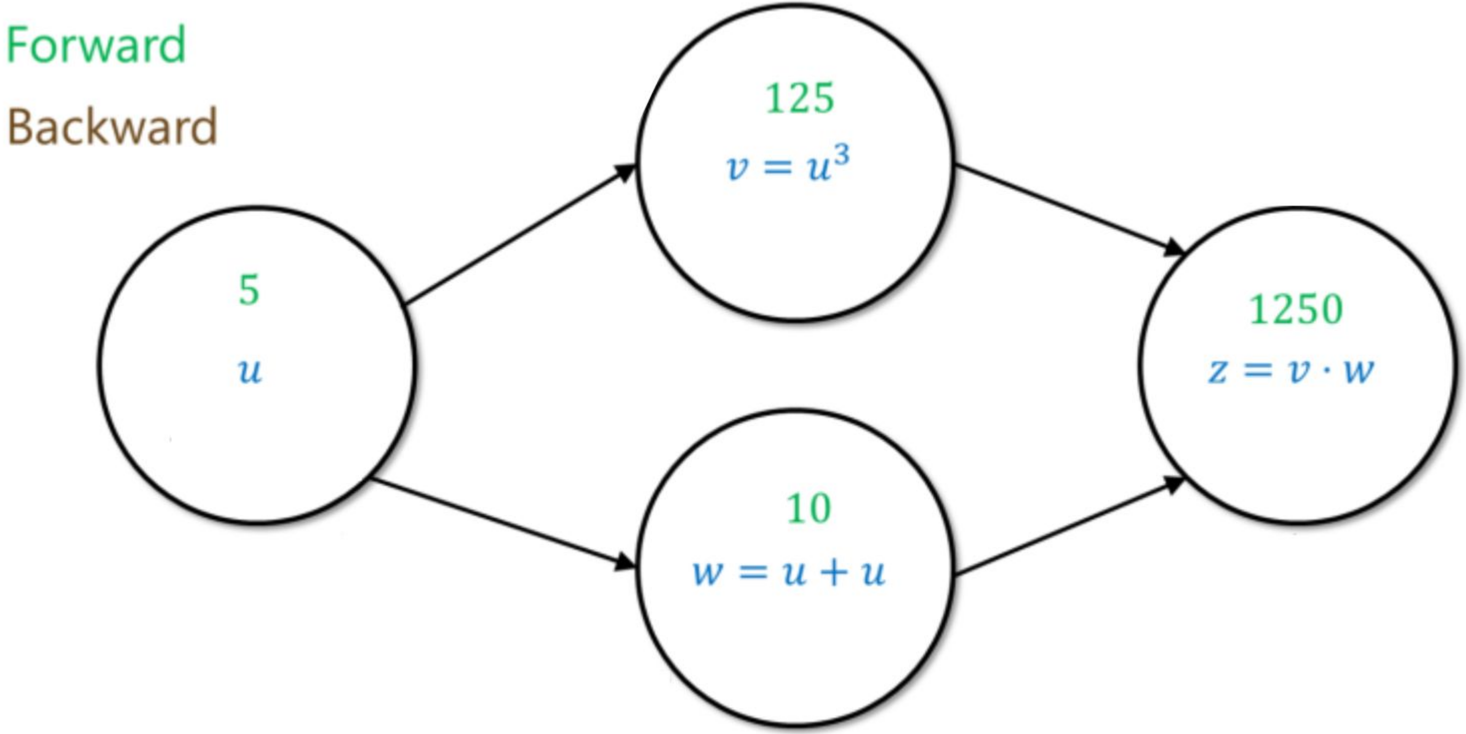
Backpropagation- An Example



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Backpropagation- An Example

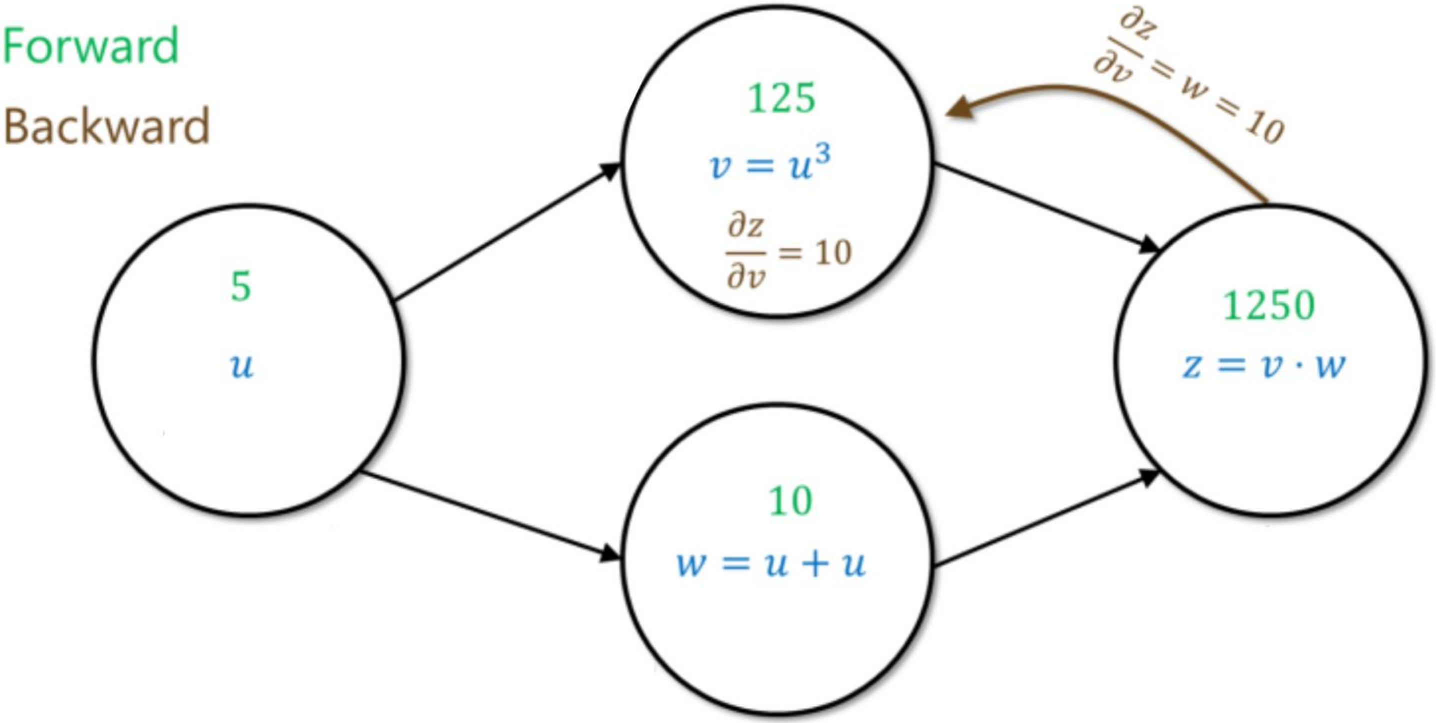
$$\frac{\partial z}{\partial u} = \left(\frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w}\right)$$



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Backpropagation- An Example

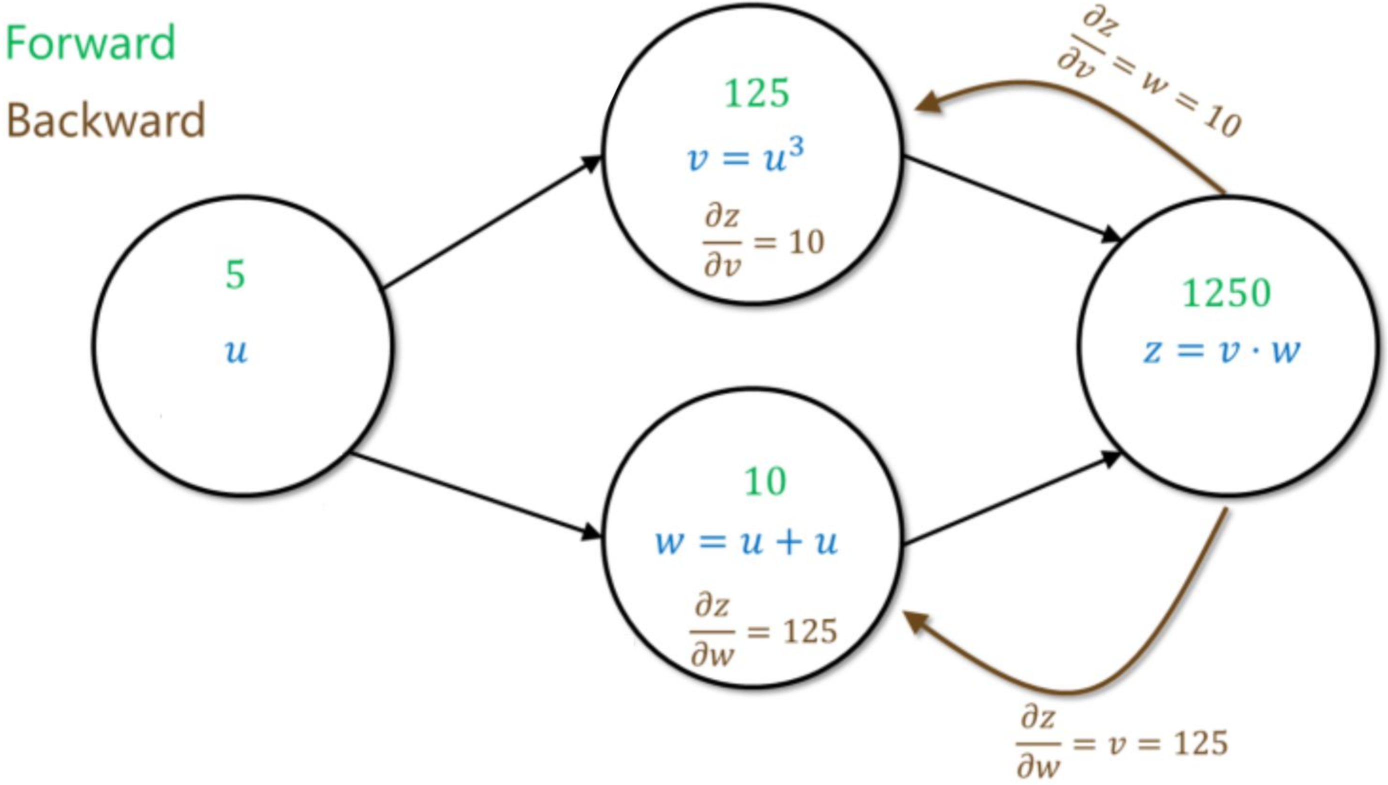
$$\frac{\partial z}{\partial u} = \left(\frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w}\right)$$



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Backpropagation- An Example

$$\frac{\partial z}{\partial u} = \left(\frac{\partial v}{\partial u}\cdot\frac{\partial z}{\partial v} + \frac{\partial w}{\partial u}\cdot\frac{\partial z}{\partial w}\right)$$

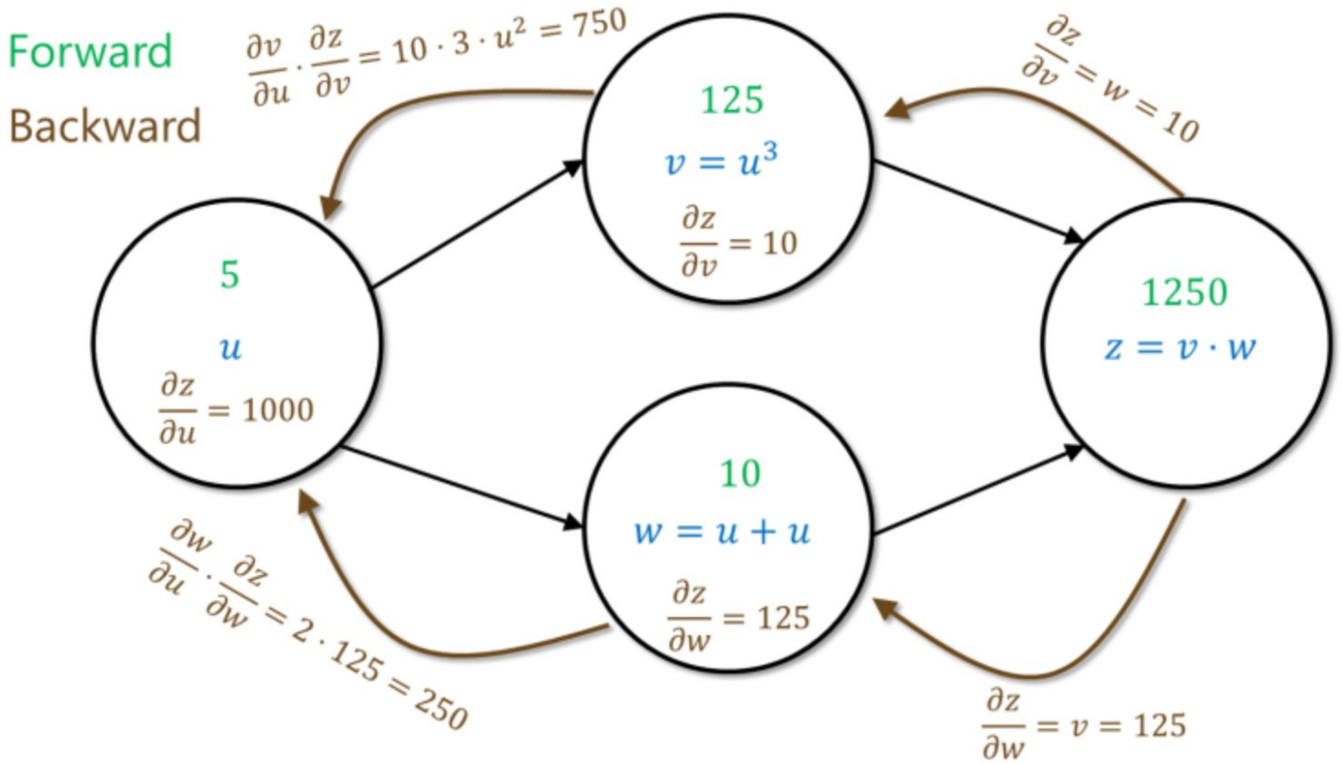


https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Cornell Bowers C·IS

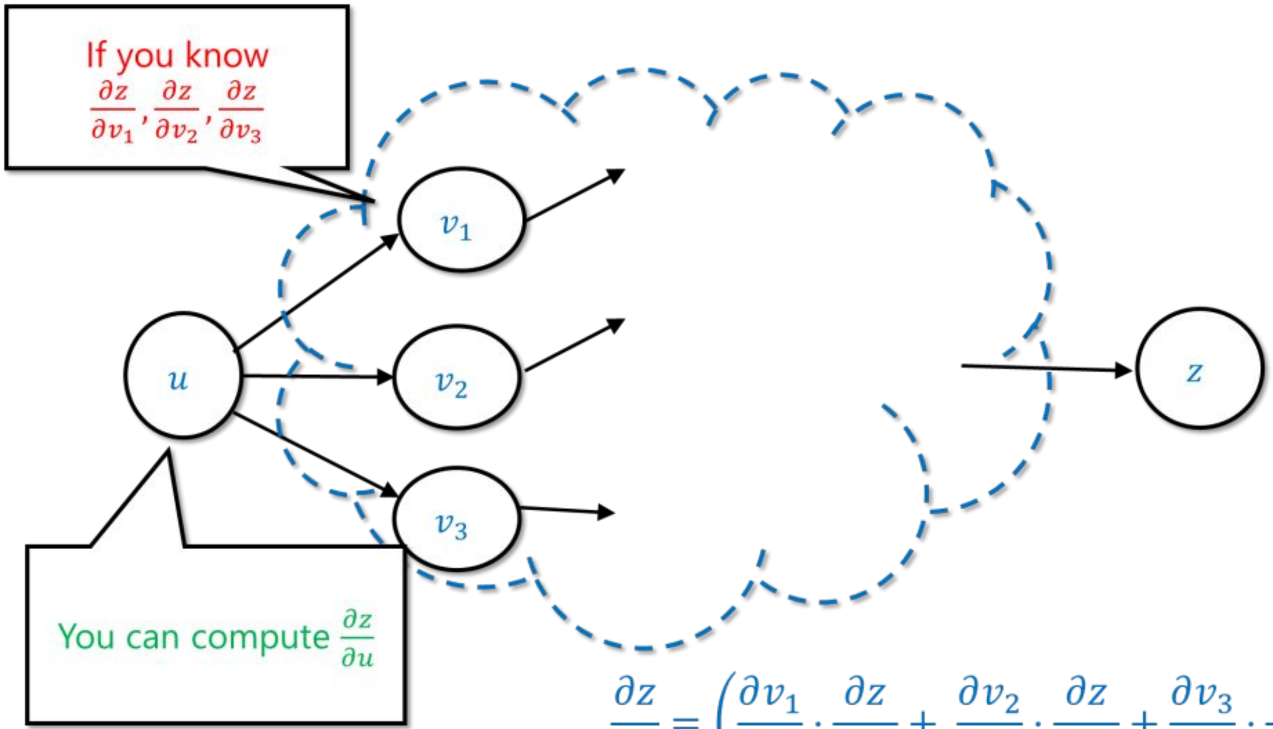
Backpropagation- An Example

$$\frac{\partial z}{\partial u} = \left(\frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w}\right)$$



https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Backpropagation- Key Idea



$$\frac{\partial z}{\partial u} = \left(\frac{\partial v_1}{\partial u} \cdot \frac{\partial z}{\partial v_1} + \frac{\partial v_2}{\partial u} \cdot \frac{\partial z}{\partial v_2} + \frac{\partial v_3}{\partial u} \cdot \frac{\partial z}{\partial v_3} \right)$$

https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/

Preview

Backpropagation- MLPs

Algorithm Forward Pass through MLP

Input: input $\mathbf{x}$, weight matrices $\mathbf{W}^{[1]}, \dots, \mathbf{W}^{[L]}$, bias vectors $\mathbf{b}^{[1]}, \dots, \mathbf{b}^{[L]}$

$\mathbf{z}^{[0]} = \mathbf{x}$ ▷ Initialize input

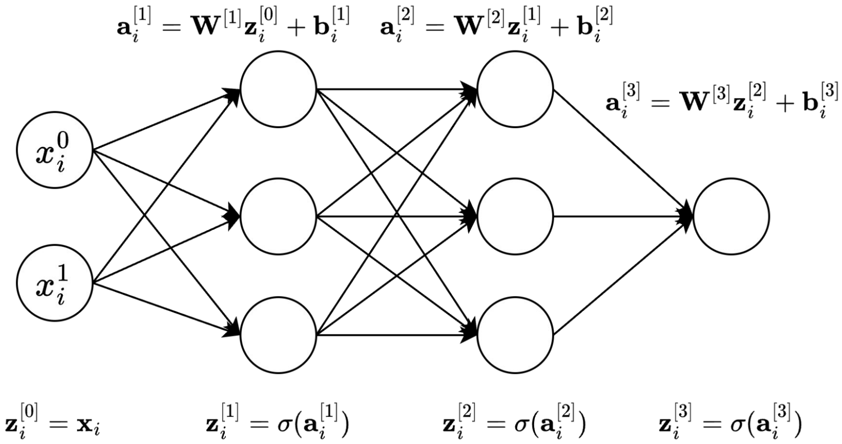
for $l = 1$ **to** L **do**

  $\mathbf{a}^{[l]} = \mathbf{W}^{[l]}\mathbf{z}^{[l-1]} + \mathbf{b}^{[l]}$ ▷ Linear transformation

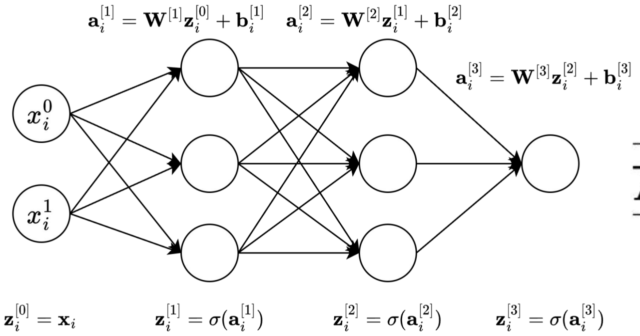
  $\mathbf{z}^{[l]} = \sigma^{[l]}(\mathbf{a}^{[l]})$ ▷ Nonlinear activation

end for

Output: $\mathbf{z}^{[L]}$



Backpropagation- MLPs



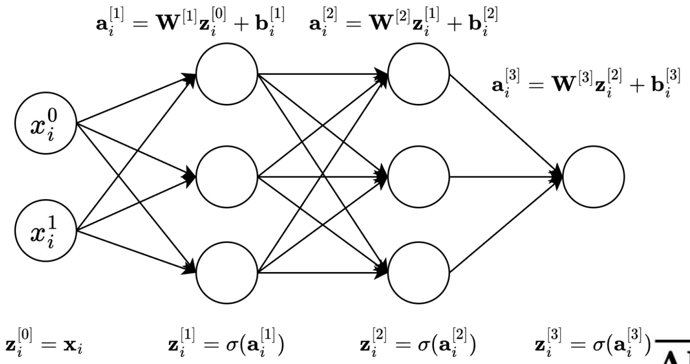
Algorithm Forward Pass through MLP

Input: input $\mathbf{x}$, weight matrices $\mathbf{W}^{[1]}, \ldots, \mathbf{W}^{[L]}$, bias vectors $\mathbf{b}^{[1]}, \ldots, \mathbf{b}^{[L]}$
$\mathbf{z}^{[0]} = \mathbf{x}$ ▷ Initialize input
for $l = 1$ **to** L **do**
    $\mathbf{a}^{[l]} = \mathbf{W}^{[l]}\mathbf{z}^{[l-1]} + \mathbf{b}^{[l]}$ ▷ Linear transformation
    $\mathbf{z}^{[l]} = \sigma^{[l]}(\mathbf{a}^{[l]})$ ▷ Nonlinear activation
end for
Output: $\mathbf{z}^{[L]}$

Algorithm Backward Pass through MLP

Input: $\{\mathbf{z}^{[1]}, \ldots, \mathbf{z}^{[L]}\}$, $\{\mathbf{a}^{[1]}, \ldots, \mathbf{a}^{[L]}\}$, loss gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}}$
$\delta^{[L]} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} \odot \sigma^{[L]'}(\mathbf{a}^{[L]})$ ▷ Error term
for $l = L$ **to** 1 **do**
    $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[l]}} = \delta^{[l]}(\mathbf{z}^{[l-1]})^T$ ▷ Gradient of weights
    $\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[l]}} = \delta^{[l]}$ ▷ Gradient of biases
    $\delta^{[l-1]} = ((\mathbf{W}^{[l]})^T\delta^{[l]}) \odot \sigma^{[l-1]'}(\mathbf{a}^{[l-1]})$
end for
Output: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$

Backpropagation- MLPs



Algorithm Forward Pass through MLP

Input: input $\mathbf{x}$, weight matrices $\mathbf{W}^{[1]}, \ldots, \mathbf{W}^{[L]}$, bias vectors $\mathbf{b}^{[1]}, \ldots, \mathbf{b}^{[L]}$
$\mathbf{z}^{[0]} = \mathbf{x}$ ▷ Initialize input
for $l = 1$ **to** L **do**
  $\mathbf{a}^{[l]} = \mathbf{W}^{[l]}\mathbf{z}^{[l-1]} + \mathbf{b}^{[l]}$ ▷ Linear transformation
  $\mathbf{z}^{[l]} = \sigma^{[l]}(\mathbf{a}^{[l]})$ ▷ Nonlinear activation
end for
Output: $\mathbf{z}^{[L]}$

Algorithm Backward Pass through MLP (Detailed)

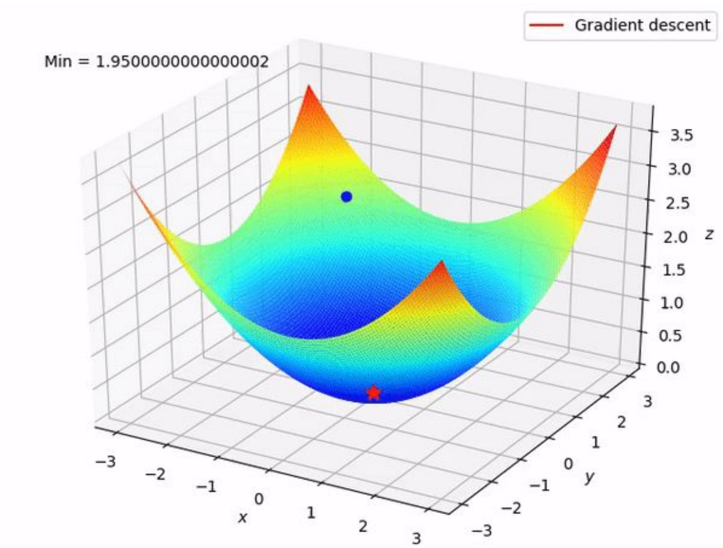
Input: $\{\mathbf{z}^{[1]}, \ldots, \mathbf{z}^{[L]}\}$, $\{\mathbf{a}^{[1]}, \ldots, \mathbf{a}^{[L]}\}$, loss gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}}$
$\delta^{[L]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} \frac{\partial \mathbf{z}^{[L]}}{\partial \mathbf{a}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} \odot \sigma^{[L]\prime}(\mathbf{a}^{[L]})$ ▷ Error term
for $l = L$ **to** 1 **do**
  $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{W}^{[l]}} = \delta^{[l]}(\mathbf{z}^{[l-1]})^T$ ▷ Gradient of weights
  $\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{b}^{[l]}} = \delta^{[l]}$ ▷ Gradient of biases
  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{z}^{[l-1]}} = (\mathbf{W}^{[l]})^T \delta^{[l]}$
  $\delta^{[l-1]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l-1]}} \frac{\partial \mathbf{z}^{[l-1]}}{\partial \mathbf{a}^{[l-1]}} = ((\mathbf{W}^{[l]})^T \delta^{[l]}) \odot \sigma^{[l-1]\prime}(\mathbf{a}^{[l-1]})$
end for
Output: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$

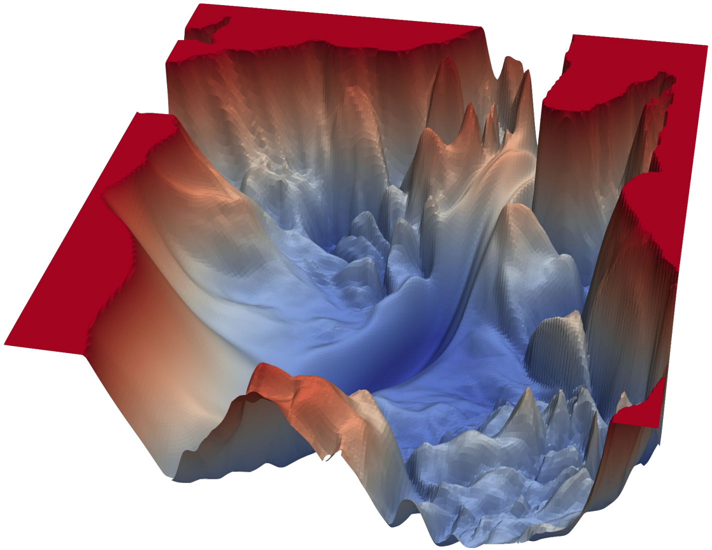
Takeaways

- MLPs consist of stacks of perceptron units
- MLPs can learn complex decision boundaries
  by composing simple features into more complex features
- Learn MLP weights with gradient descent
  - Backpropagation efficiently computes gradient

Next Week

A deep dive into training neural networks!





https://arxiv.org/abs/1712.09913