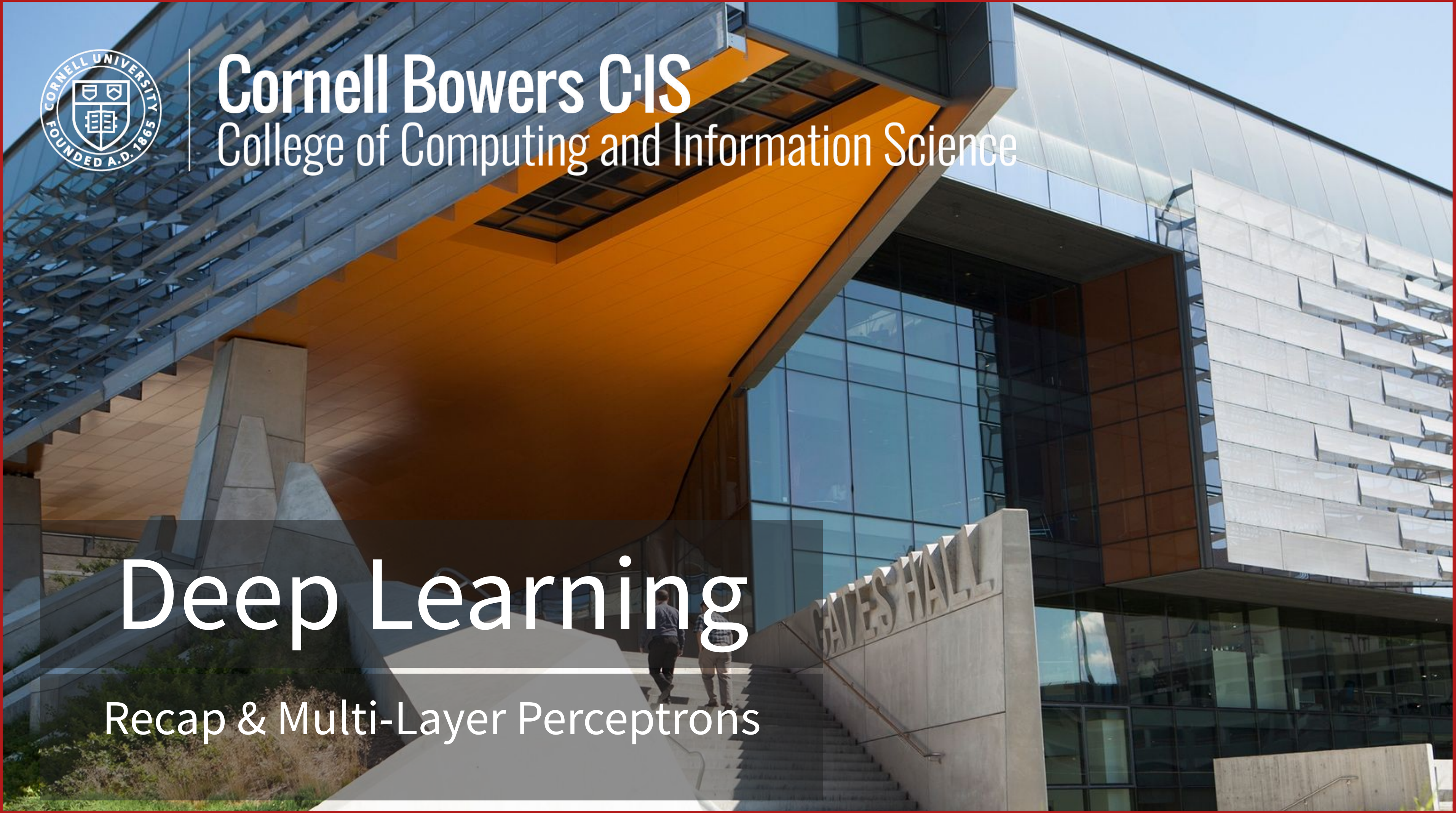


Cornell Bowers C·IS
College of Computing and Information Science

Deep Learning

Recap & Multi-Layer Perceptrons

Agenda

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

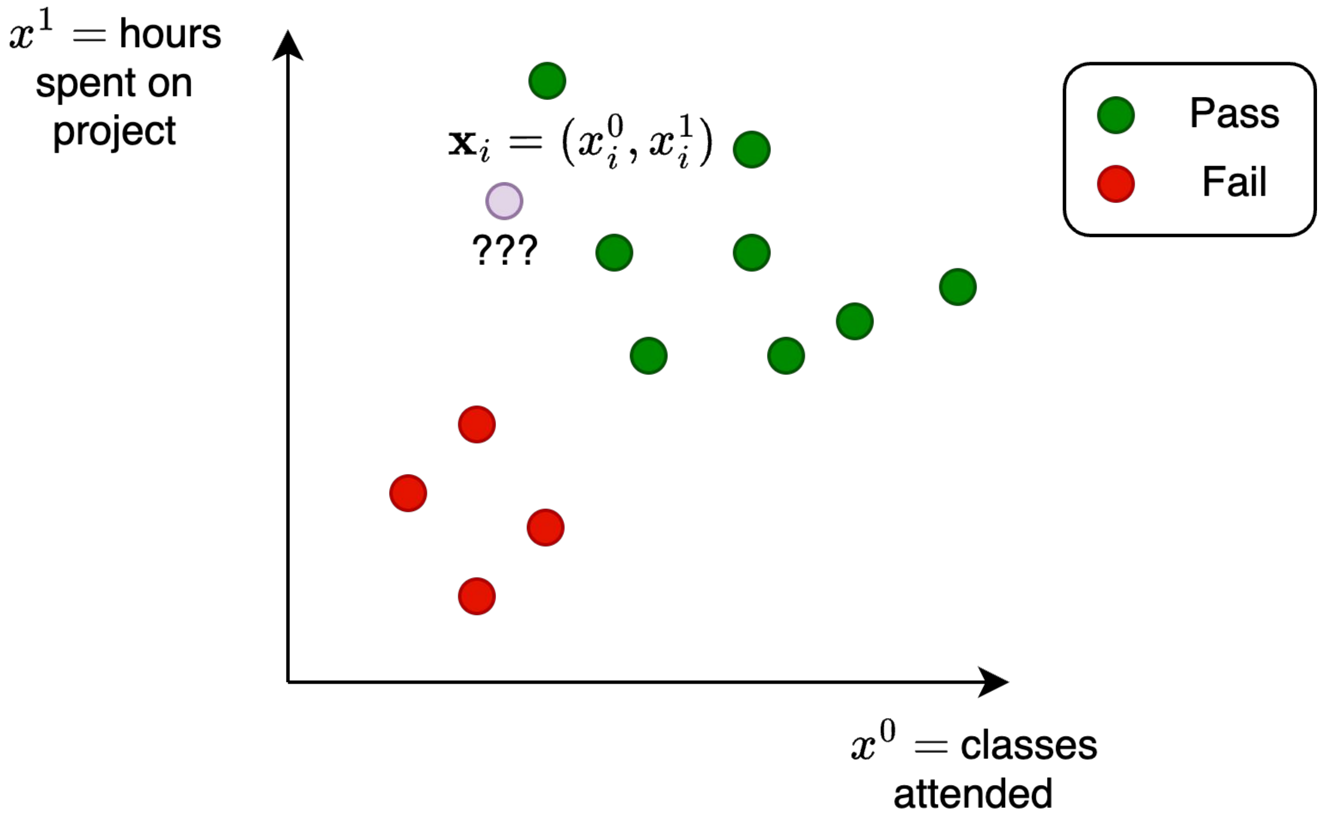
Cornell Bowers C·IS

A Classification Problem:

Will I Pass This Class?

Cornell Bowers CIS

A Classification Problem: Will I Pass This Class?



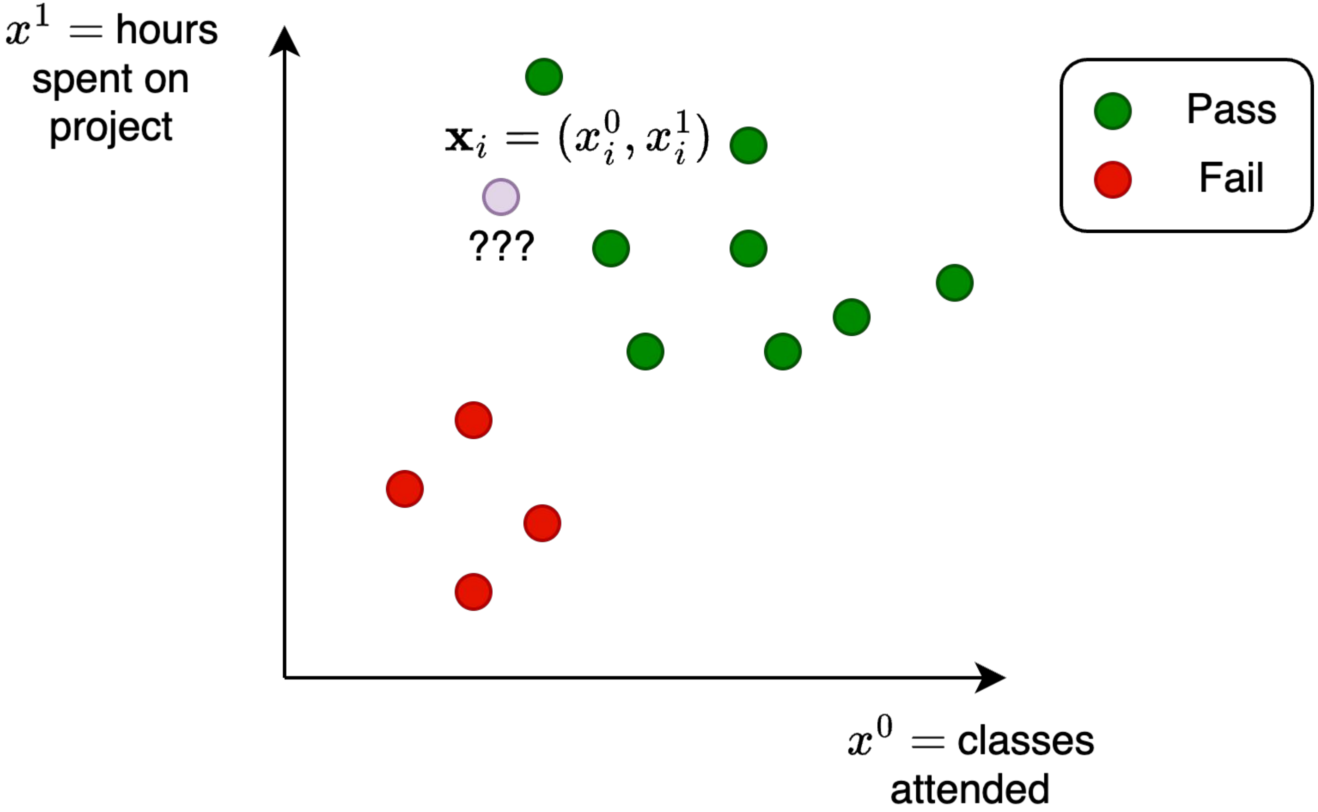
Cornell Bowers C·IS

What are key components in ML?

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

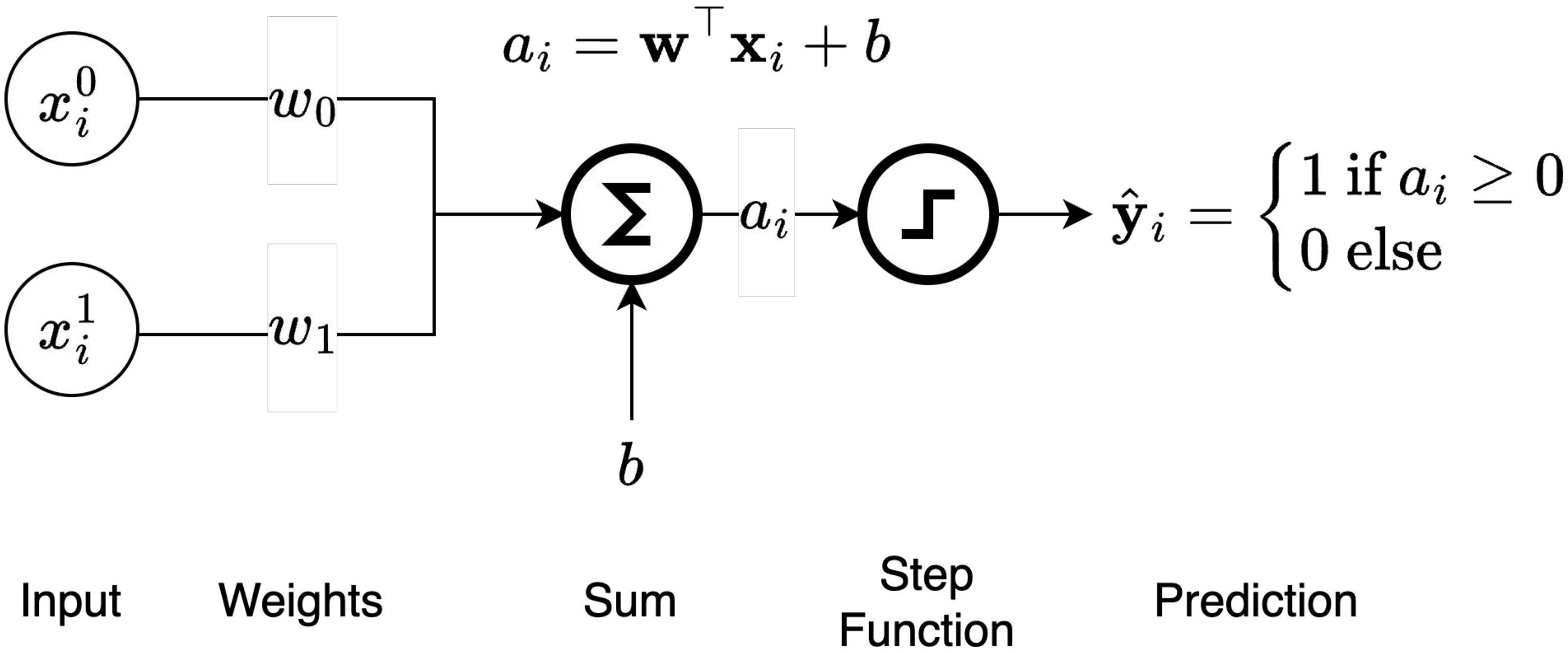
Cornell Bowers C·IS

A Classification Problem: Will I Pass This Class?



Perceptron

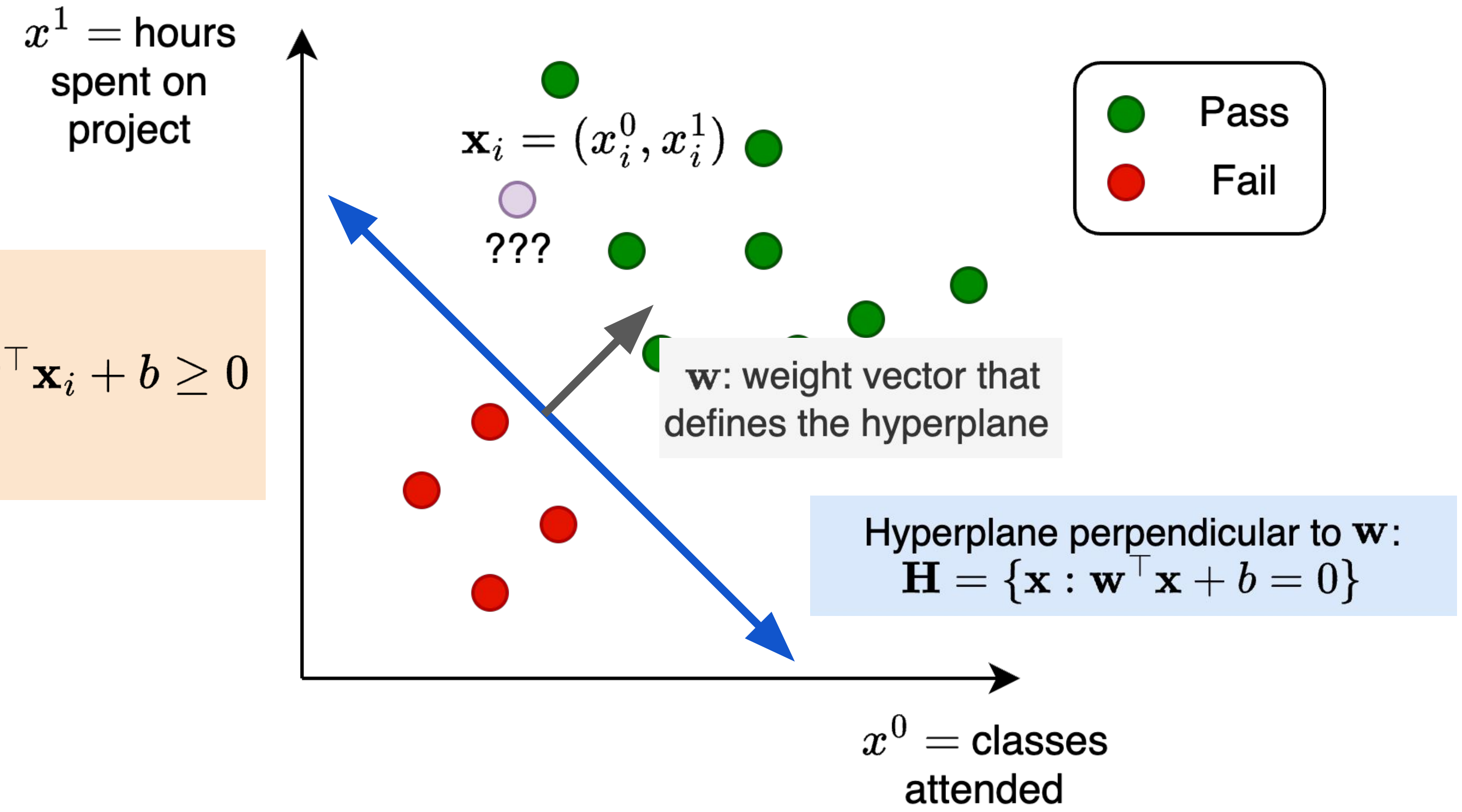
- Linear classifier
  - Predecessor to neural network



A Classification Problem: Will I Pass This Class?

Recall:

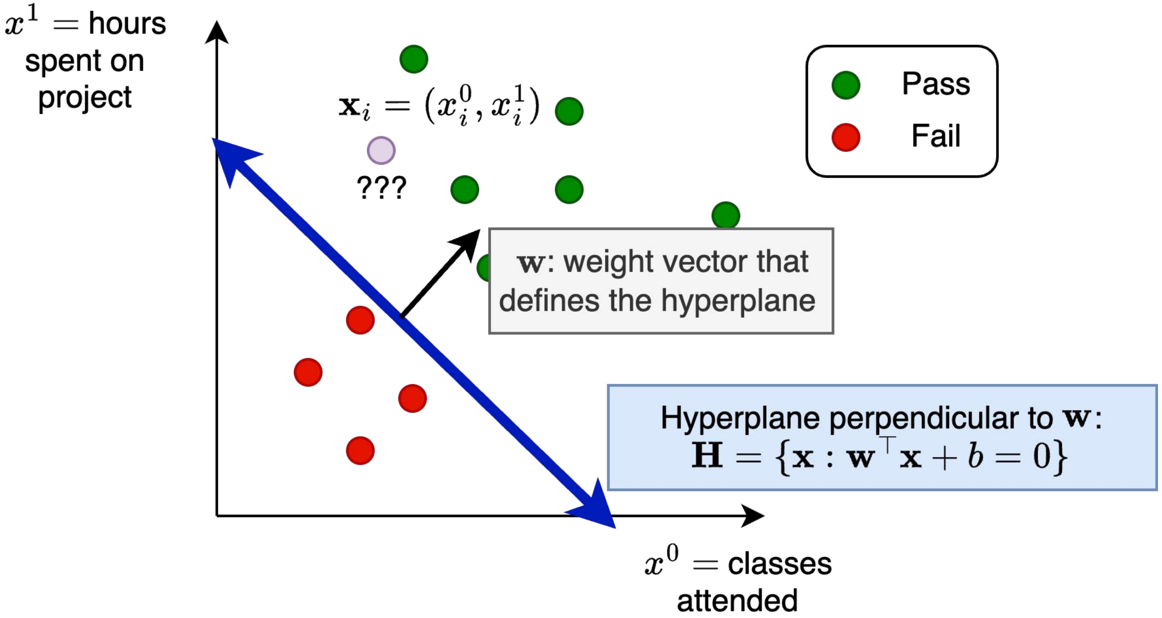
$$y_i = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$



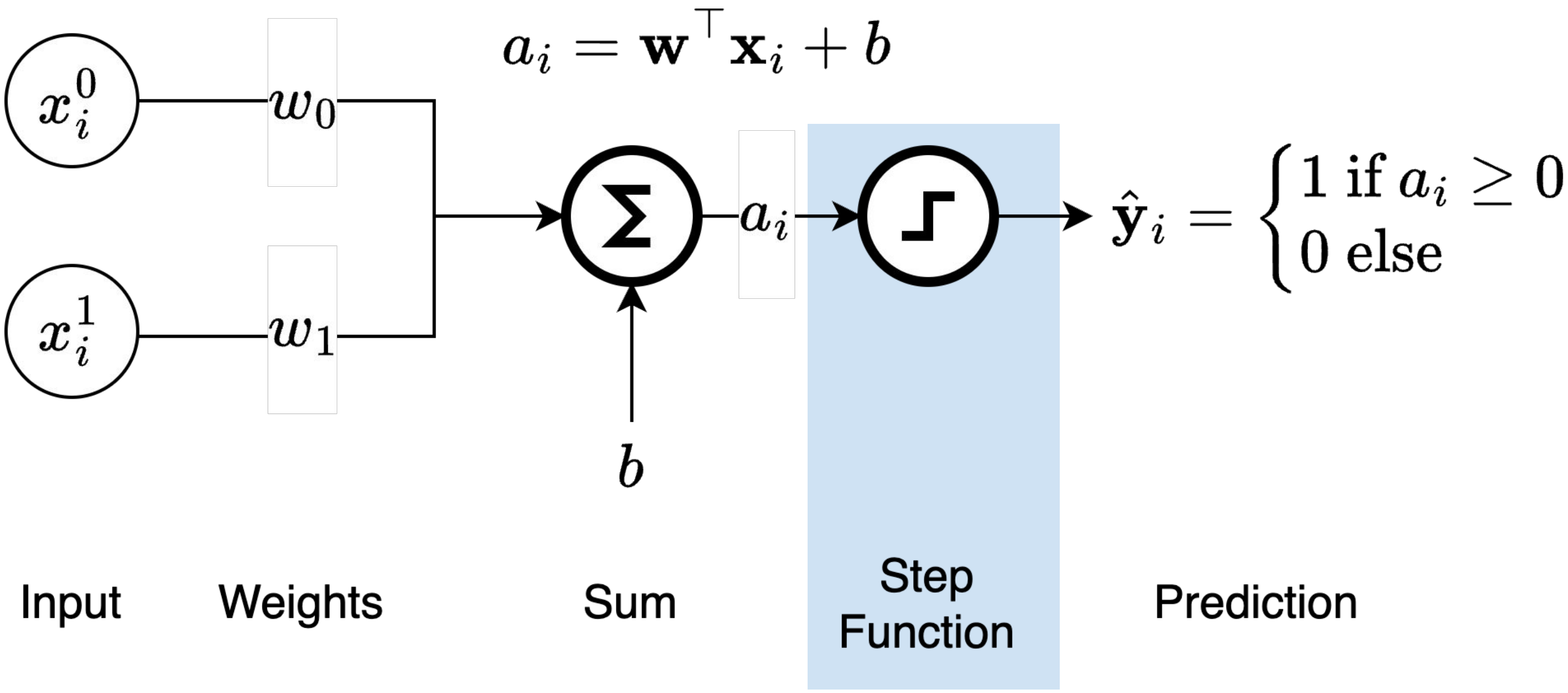
A Classification Problem: Will I Pass This Class?

- Perceptron defines a linear classification boundary

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



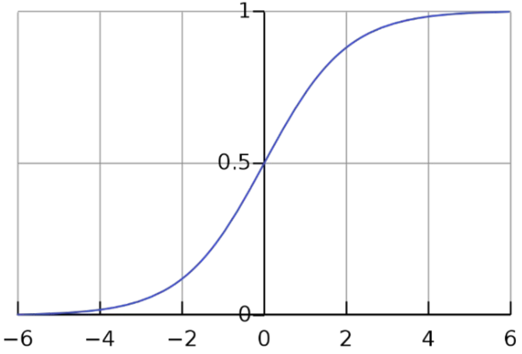
Perceptron


$a_i = \mathbf{w}^\top \mathbf{x}_i + b$
x_i^0
x_i^1
w_0
w_1
a_i
b
$\hat{\mathbf{y}}_i = \begin{cases} 1 \text{ if } a_i \geq 0 \\ 0 \text{ else} \end{cases}$
Input
Weights
Sum
Step Function
Prediction

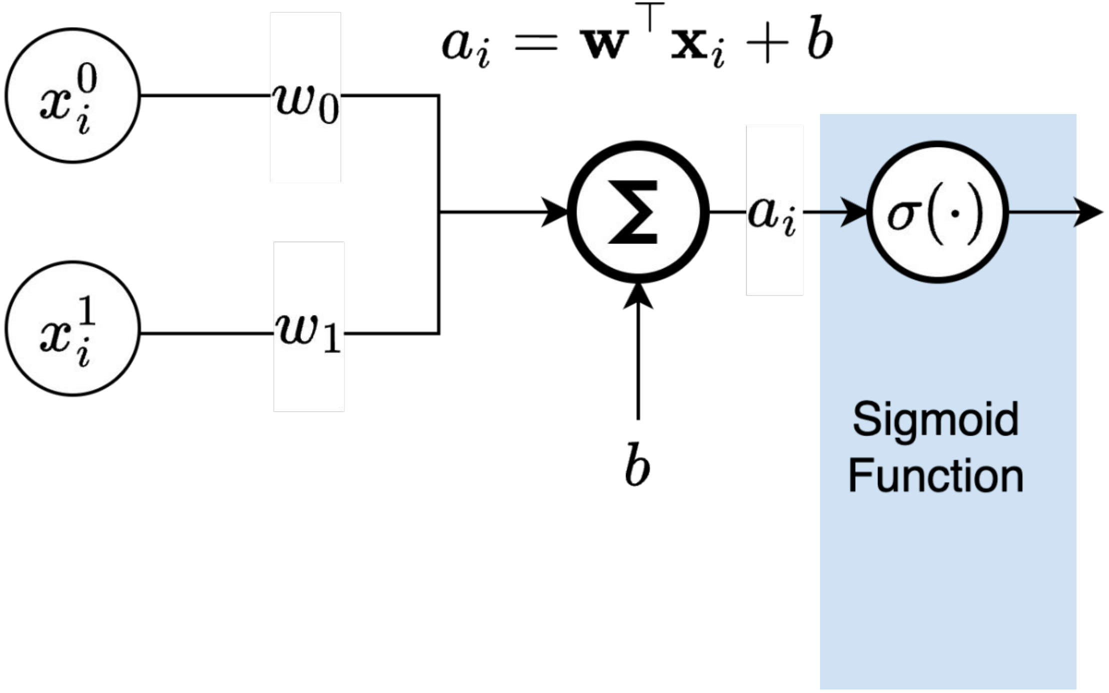
Cornell Bowers C·IS

The “Soft” Perceptron

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



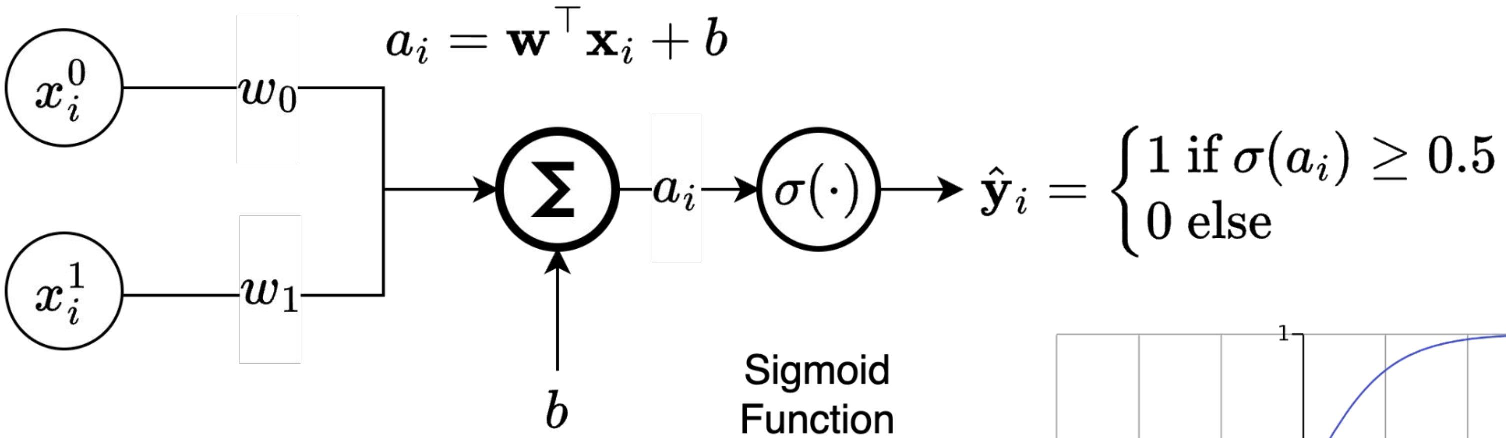
$$a_i = \mathbf{w}^\top \mathbf{x}_i + b$$



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

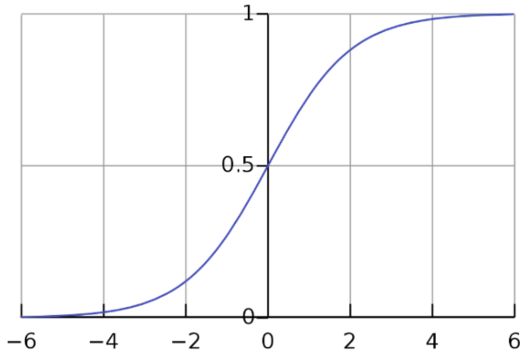
In other words… Logistic Regression

- A single-layer perceptron



Sigmoid Function

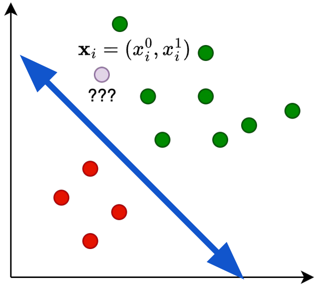
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



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

Absorb bias term into feature vector:

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

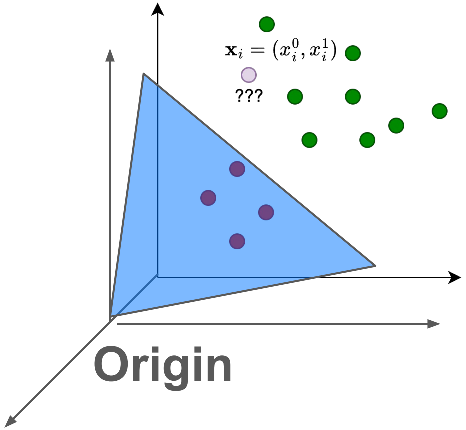


We can see that:

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

Can rewrite logistic regression as

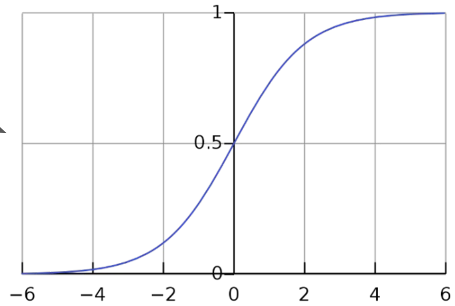
$$\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$



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 $\mathbf{y}_i \in \{0, 1\}$

$\rightarrow$ 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 CIS

Discuss: Why are they equivalent?

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

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

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

Conditional independent

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

monotonic

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

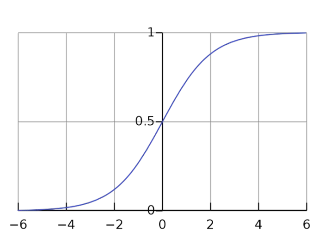
Flipping +/-

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

Cornell Bowers C·IS

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:

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

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

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

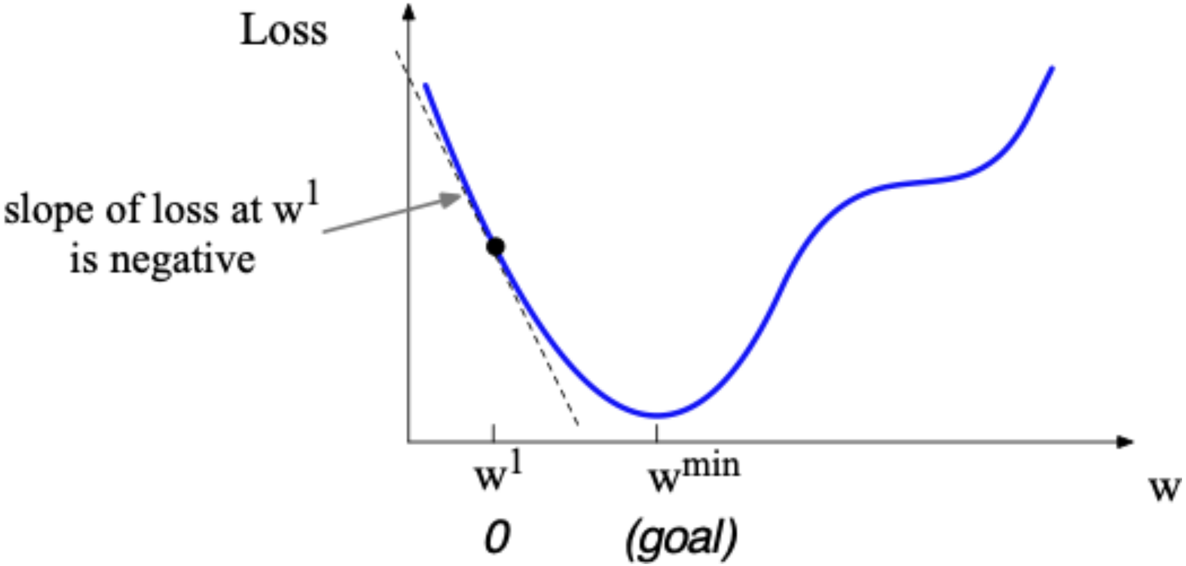
Cornell Bowers C·IS

Gradient Descent

Visualize Gradient Descent in 1-D

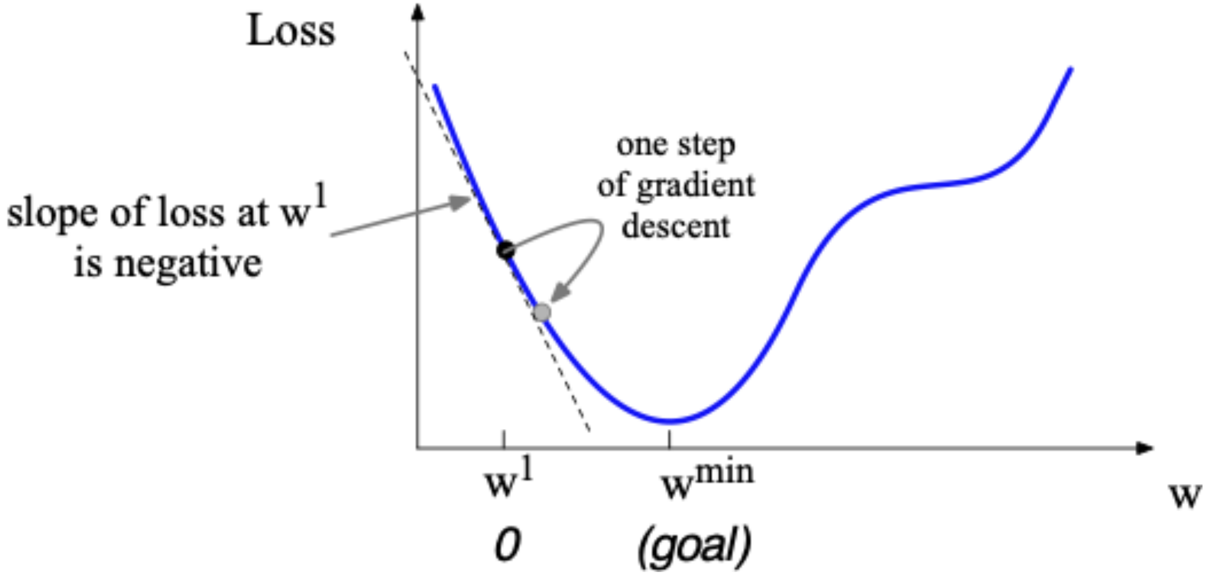
https://web.stanford.edu/~jurafsky/slp3/

Visualize Gradient Descent in 1-D



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

Visualize Gradient Descent in 1-D

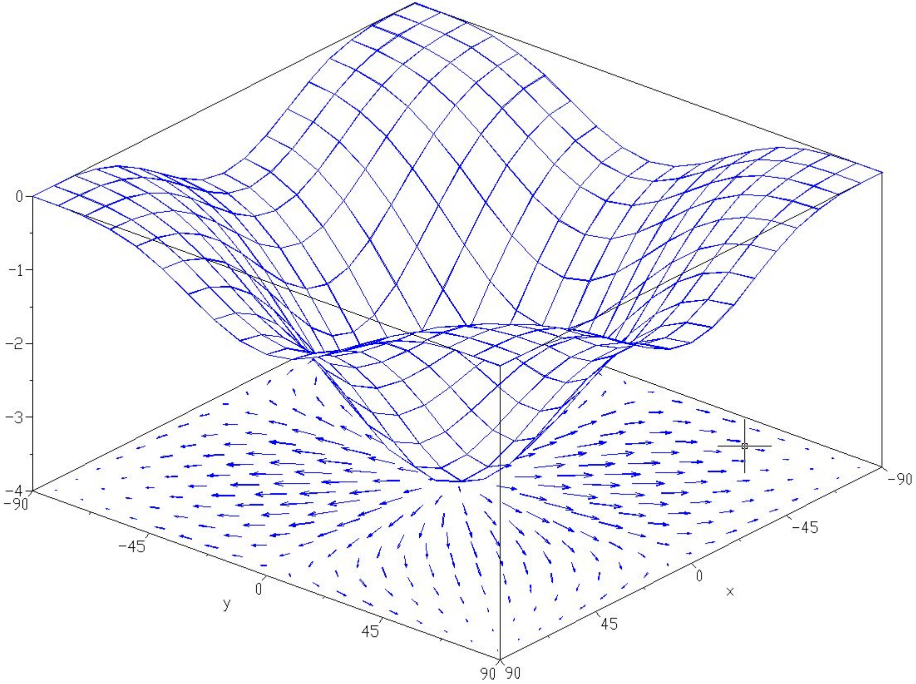


https://web.stanford.edu/~jurafsky/slp3/

Cornell Bowers C·IS

Gradients

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



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

Gradient Descent:

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

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

Cornell Bowers C·IS

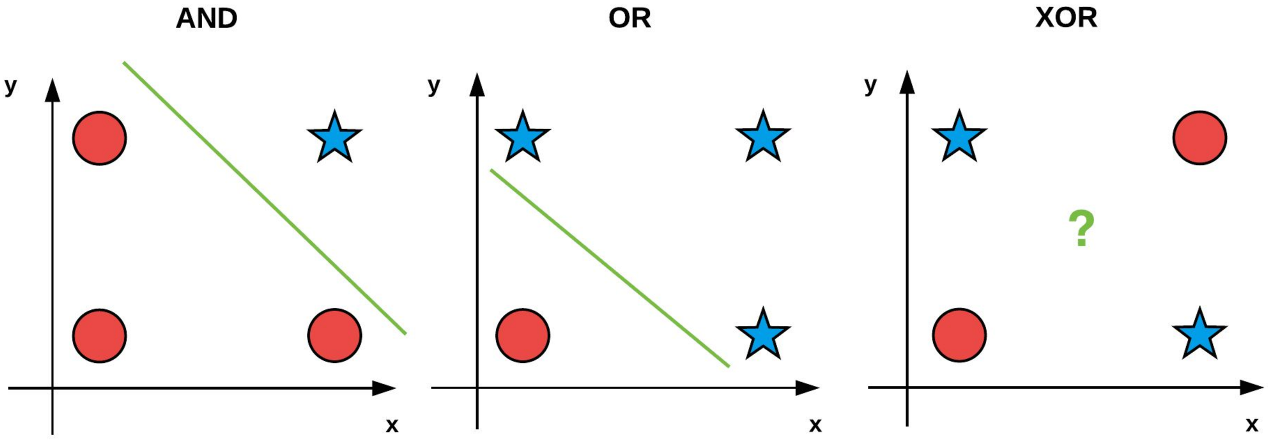
What are key components in ML?

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

Cornell Bowers C·IS

The XOR Problem

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

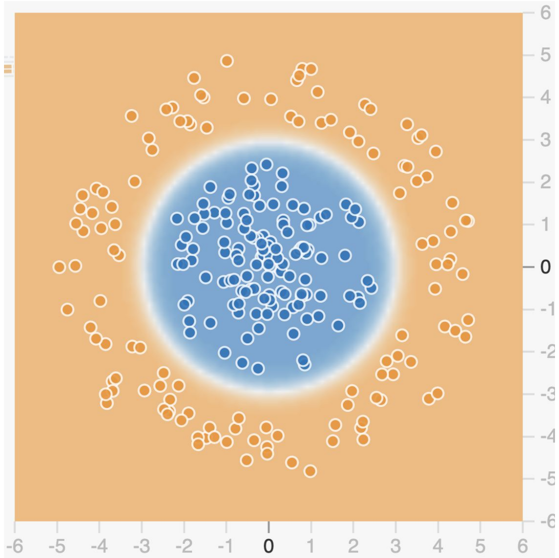


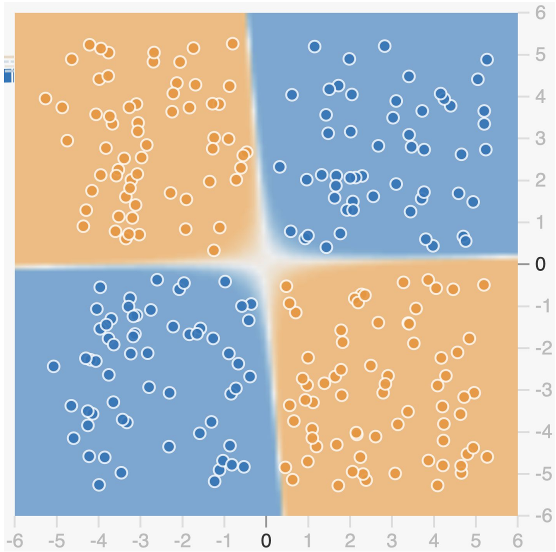
https://www.pyimagesearch.com/2021/05/06/implementing-the-perceptron-neural-network-with-python/

Cornell Bowers C·IS

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

Without deep learning!





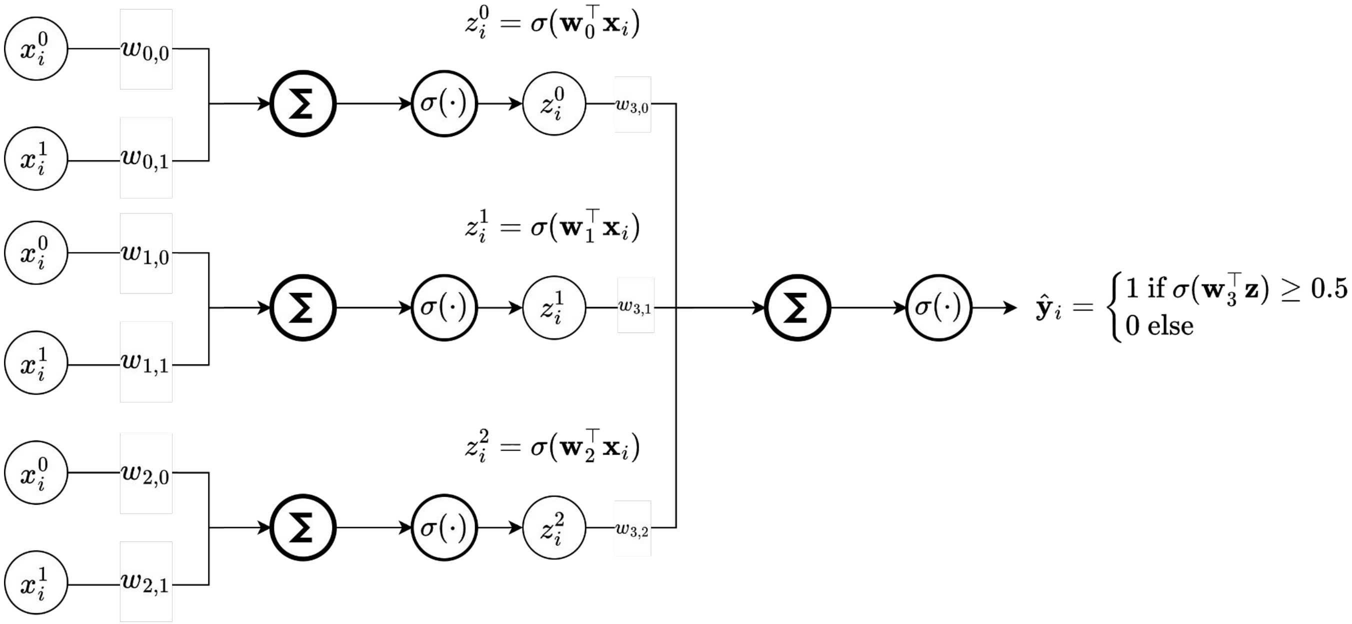
Agenda

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

Multi-Layer Perceptron (MLP)

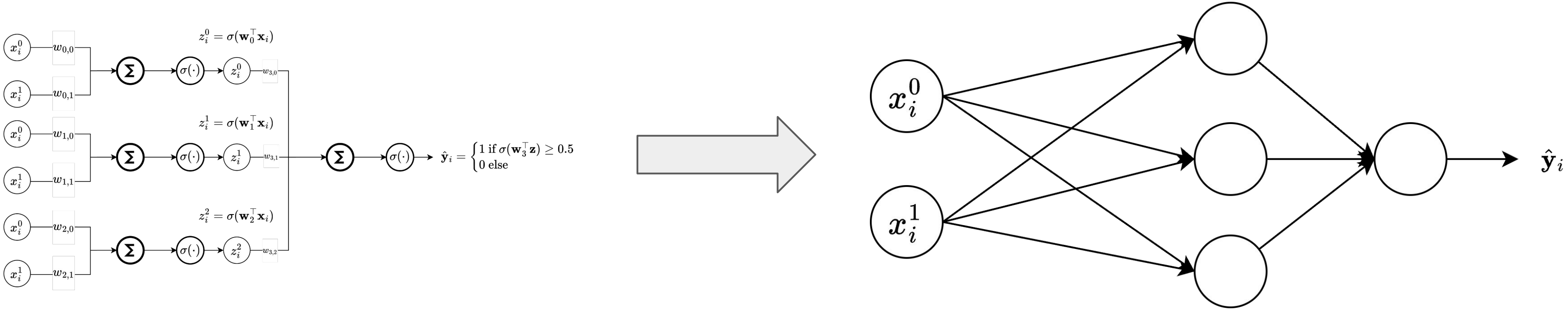
- Compose multiple perceptrons to **learn** intermediate features

An MLP with 1 hidden layer with 3 hidden units



A Simplified MLP Diagram

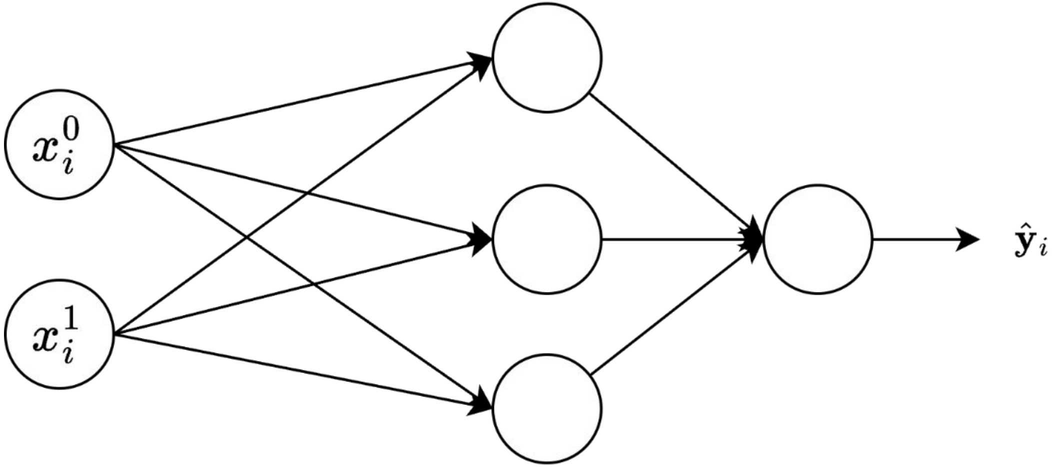
1 Hidden Layer,
3 Hidden Units

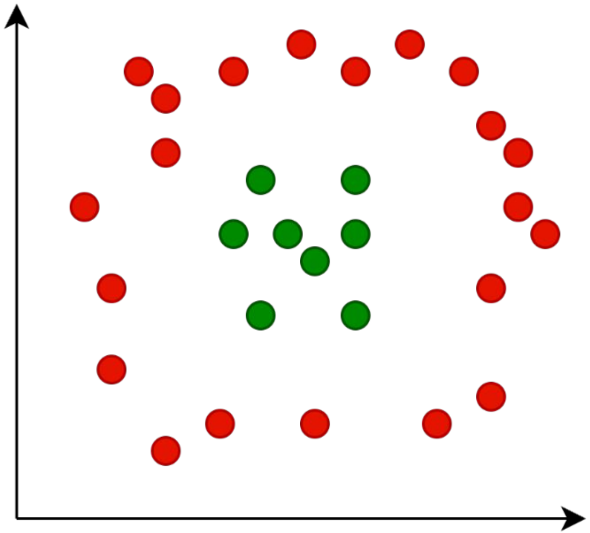


Cornell Bowers C·IS

Complex Decision Boundaries

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





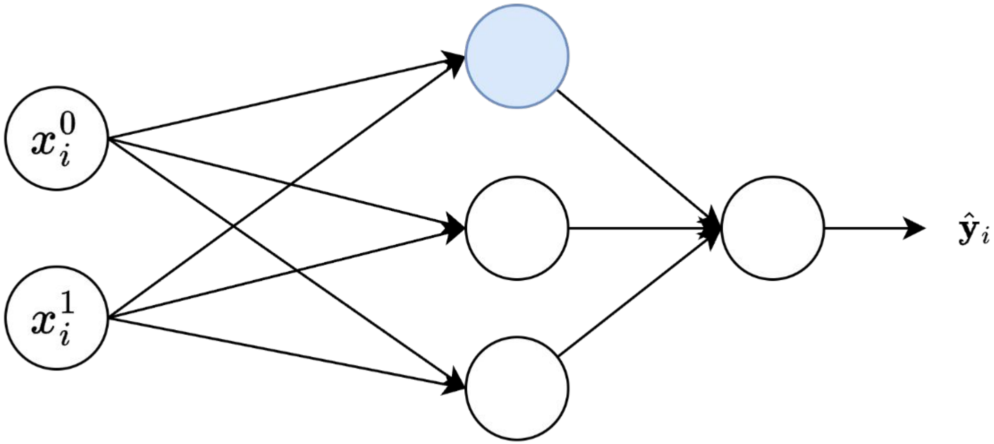
Cornell Bowers C·IS

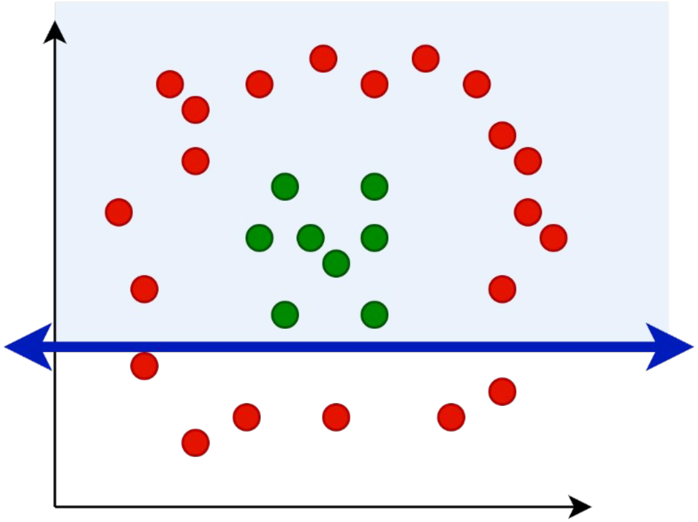
Complex Decision Boundaries

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

Recall:

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

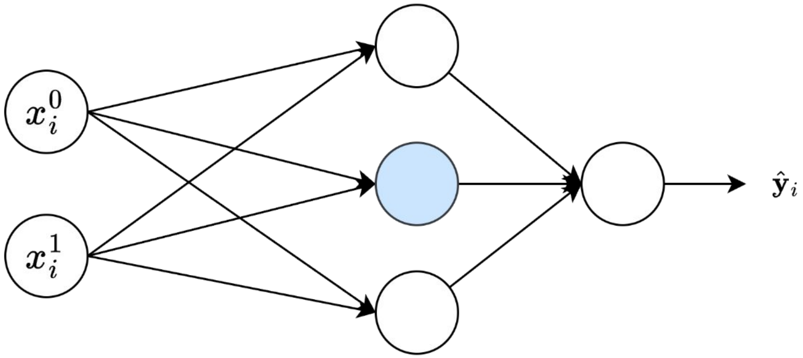


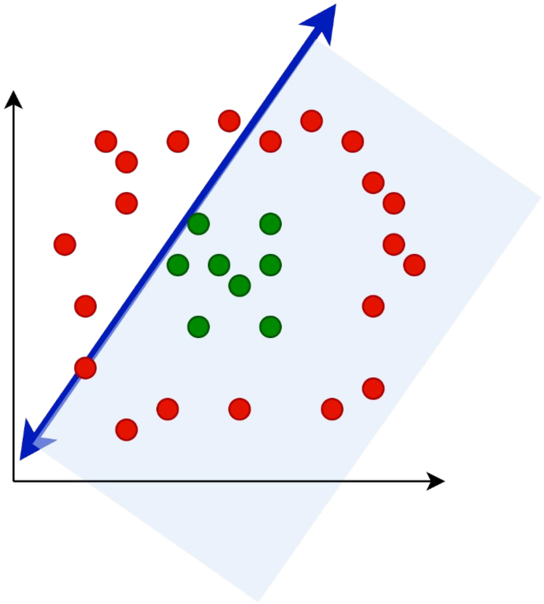


Cornell Bowers CIS

Complex Decision Boundaries

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

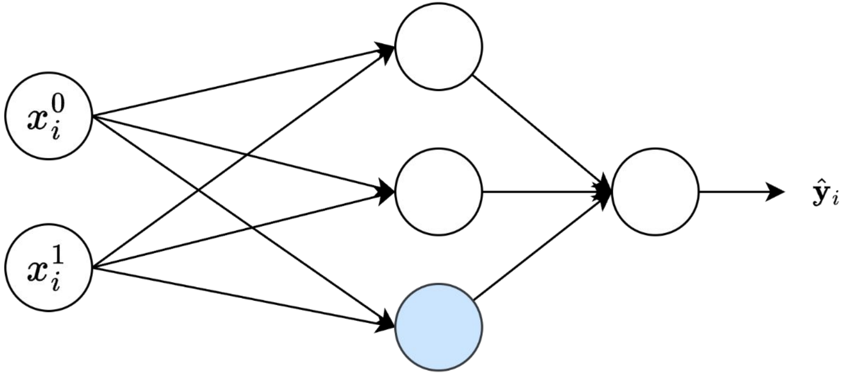


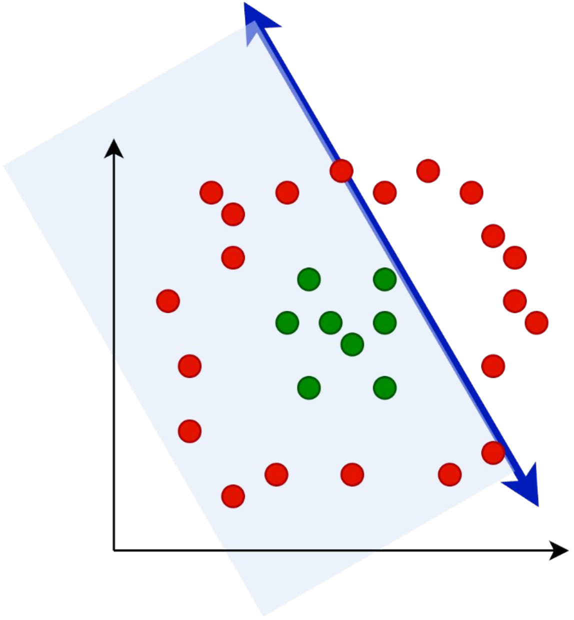


Cornell Bowers C·IS

Complex Decision Boundaries

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

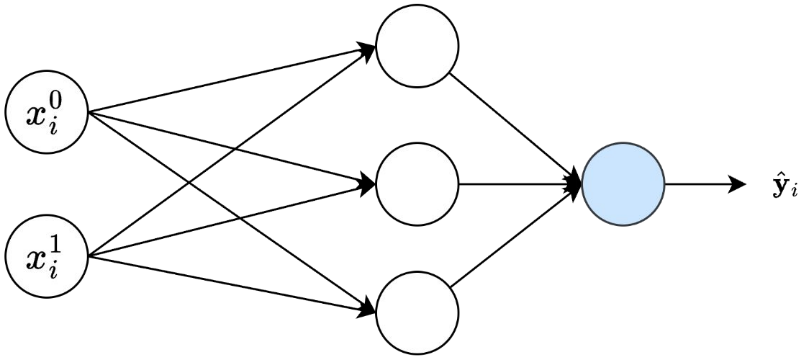


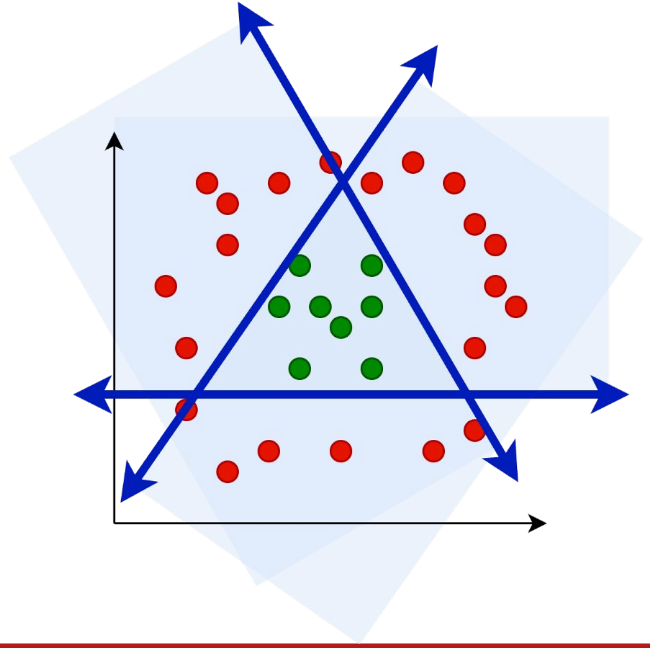


Cornell Bowers C·IS

Discuss: Why this works?

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

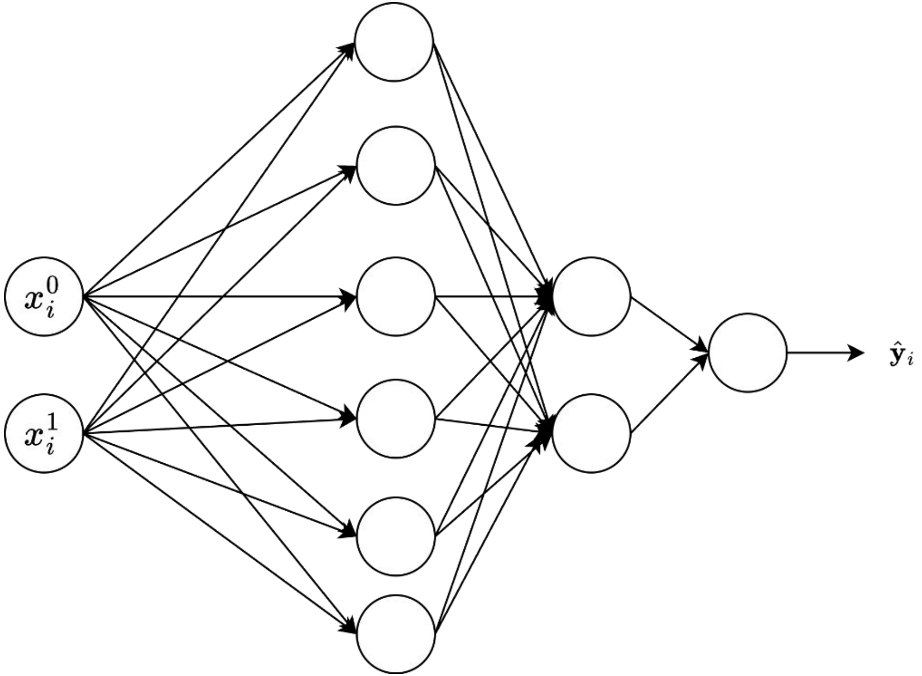


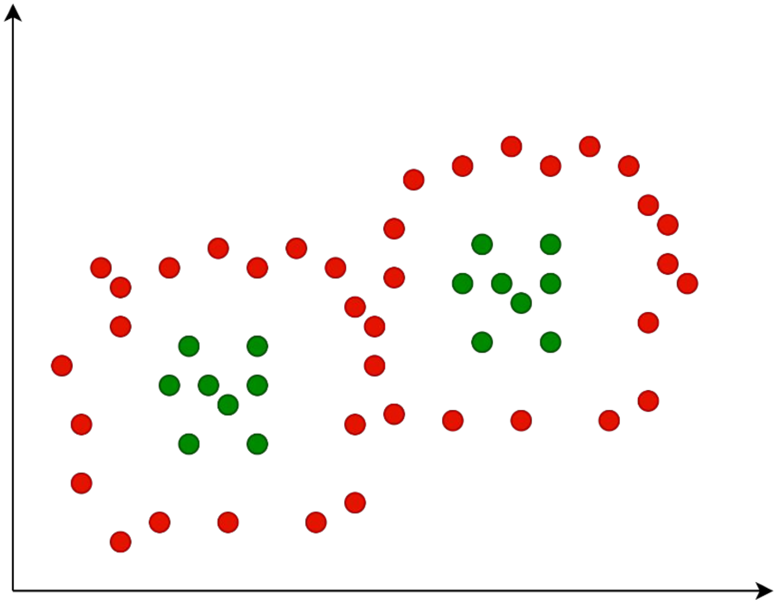


Cornell Bowers C·IS

Increasing Depth

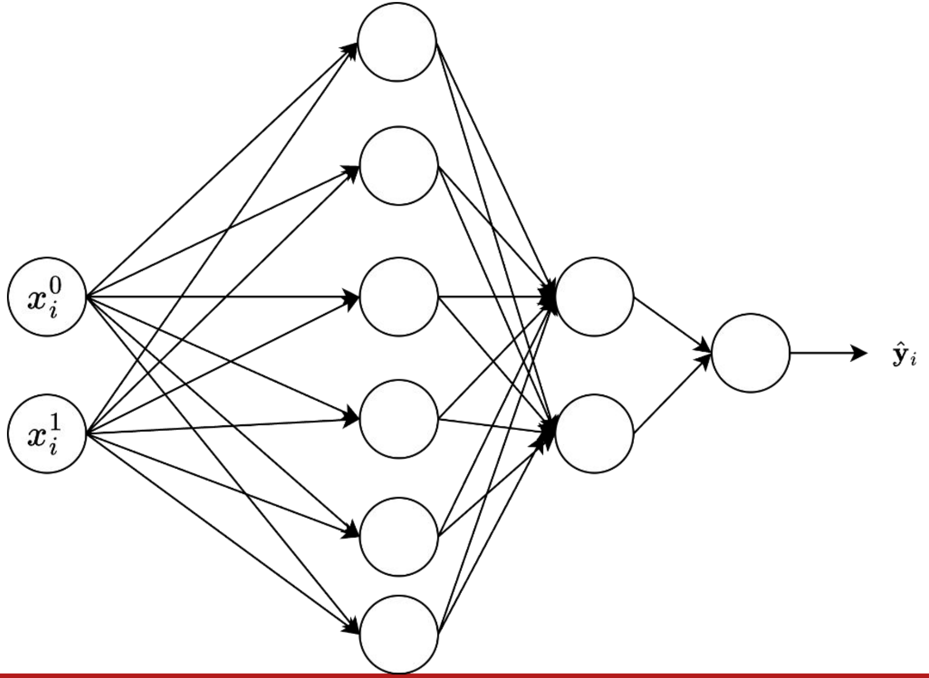
Discuss: How to construct the decision boundary?

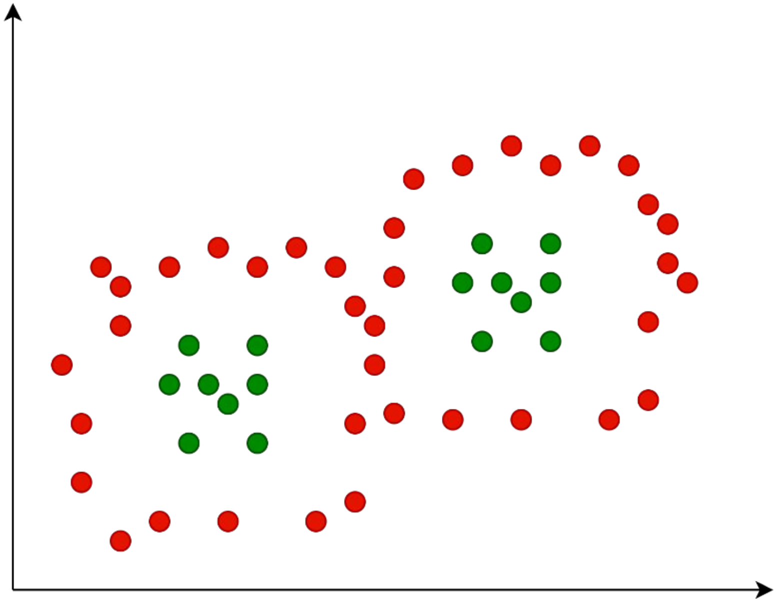




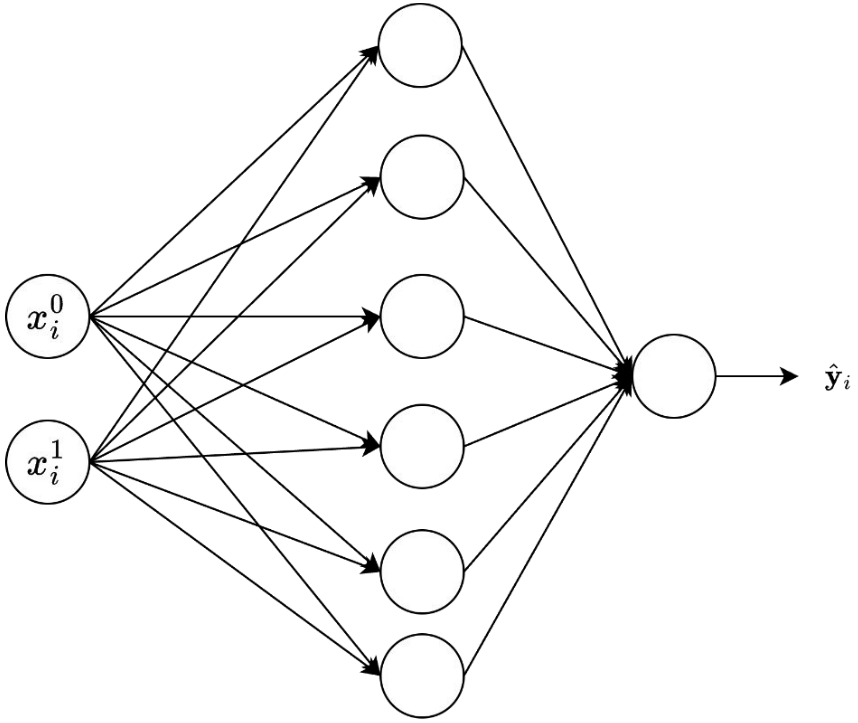
Increasing Depth

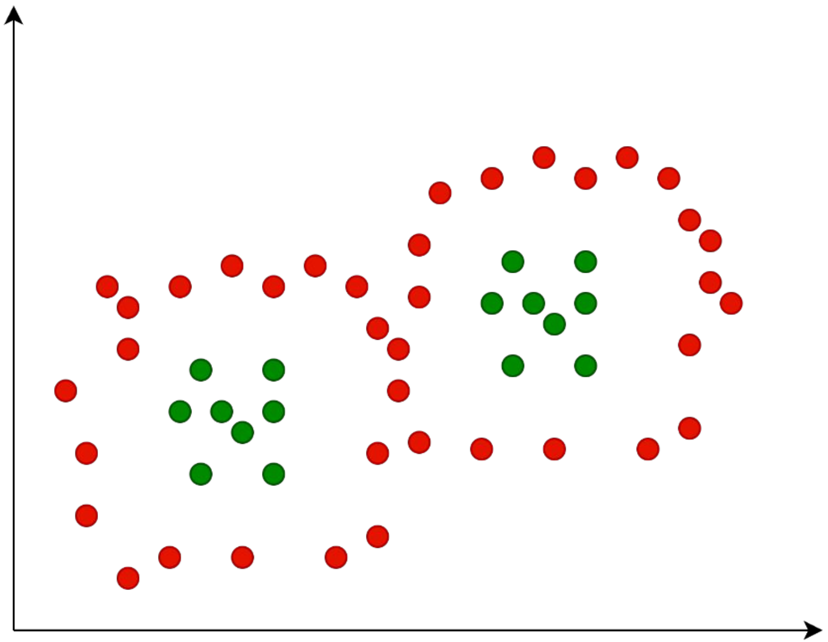
- MLP with 1 hidden layer composes linear classifiers
- MLP with 2 hidden layers can compose polygon classifiers





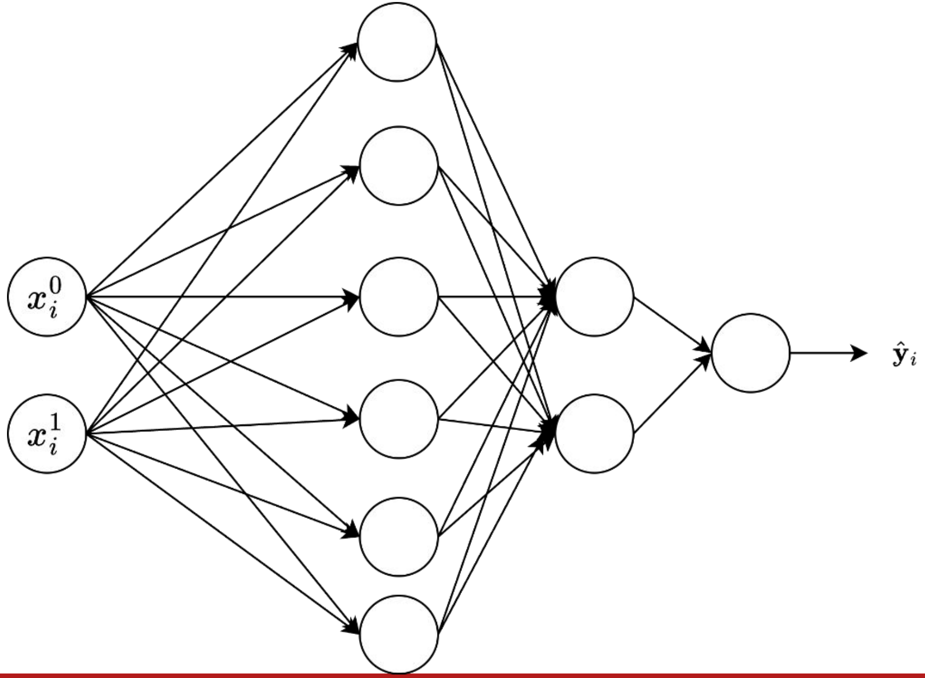
Discuss: What about just one layer?

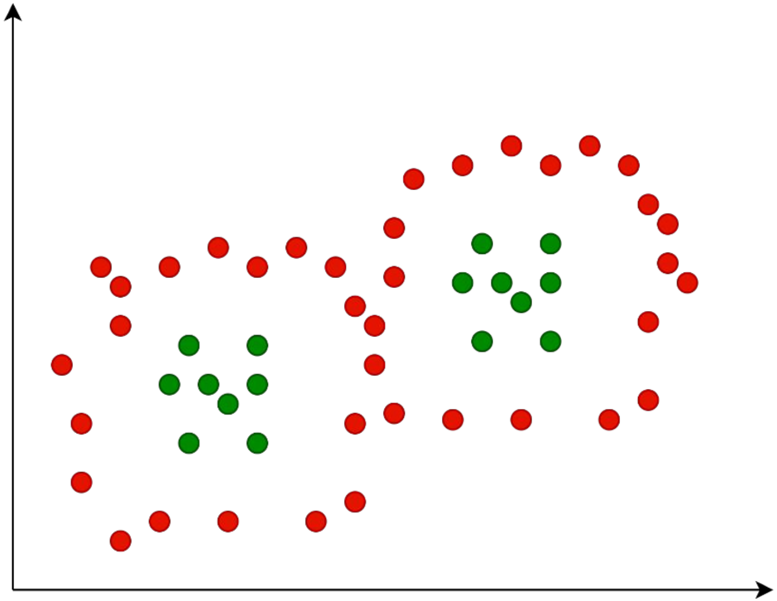




Increasing Depth

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

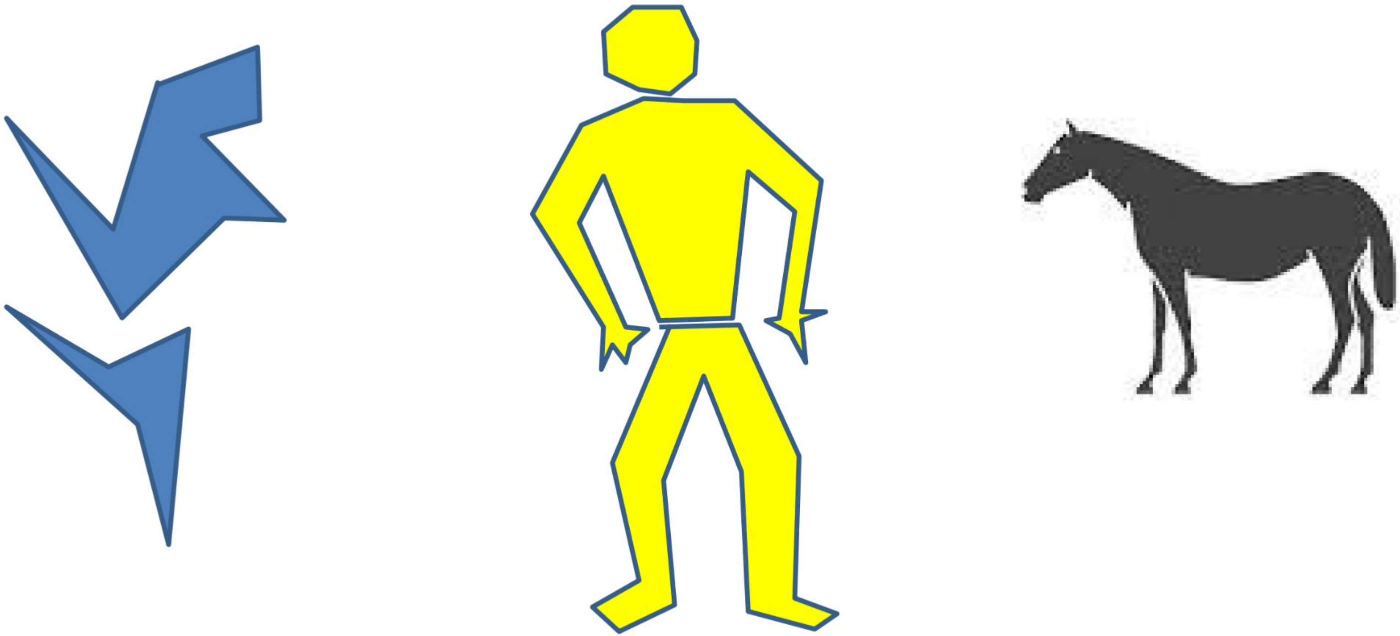




Cornell Bowers C·IS

Complex Decision Boundaries

- Can compose *arbitrarily* complex decision boundaries

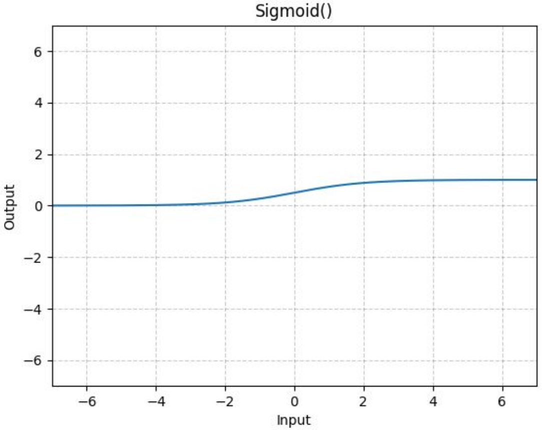


https://deeplearning.cs.cmu.edu/S24/document/slides/lec2.universal.pdf

Cornell Bowers C·IS

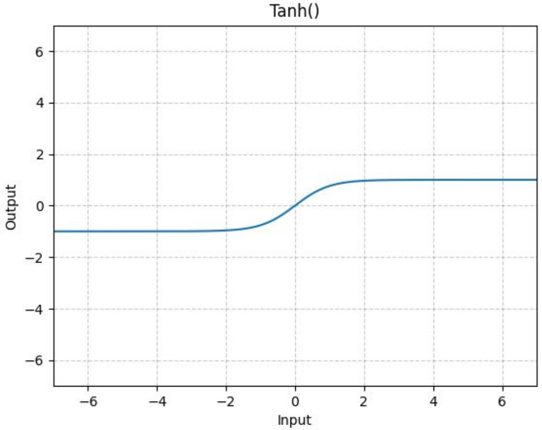
Activation Functions

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



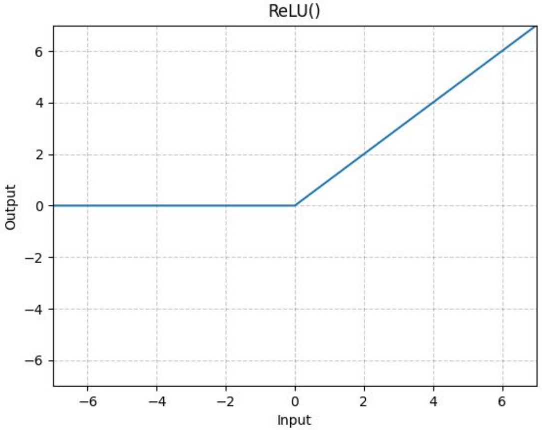
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

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

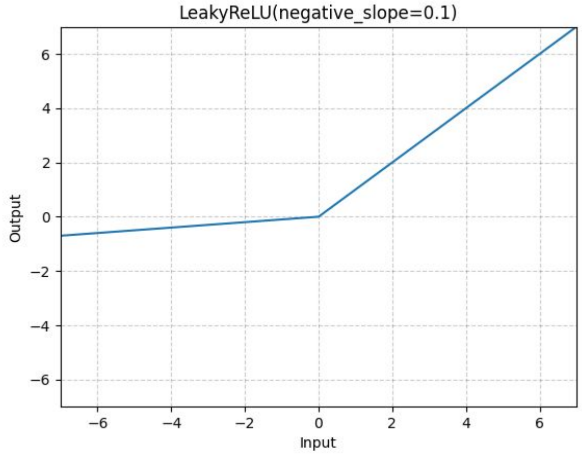
Cornell Bowers C·IS

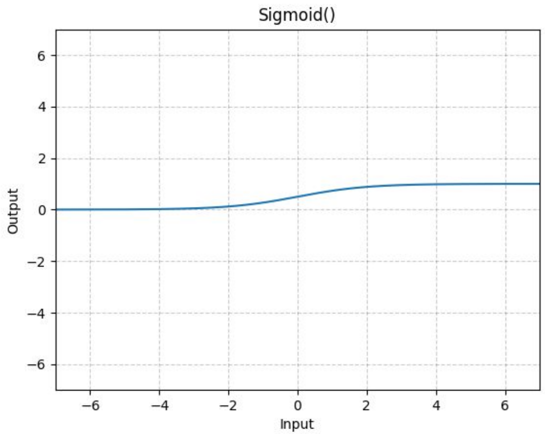
Activation Functions

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

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

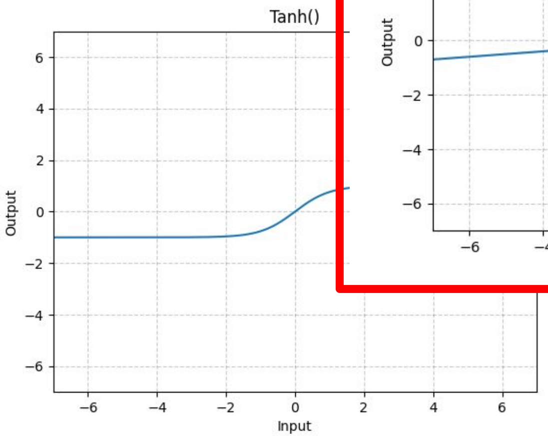
Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research





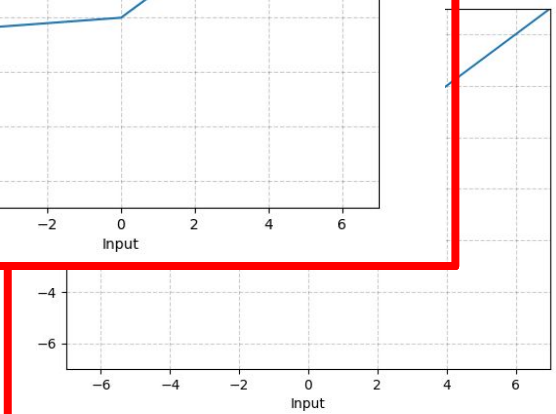
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

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



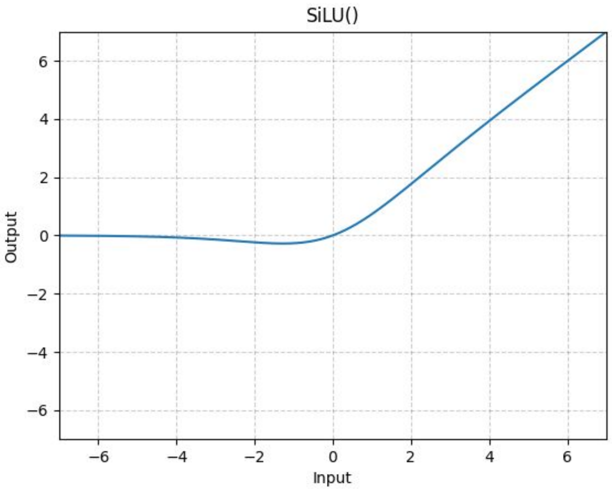
$$\mathrm{ReLU}(x) = \max(0, x)$$

Threshold at 0

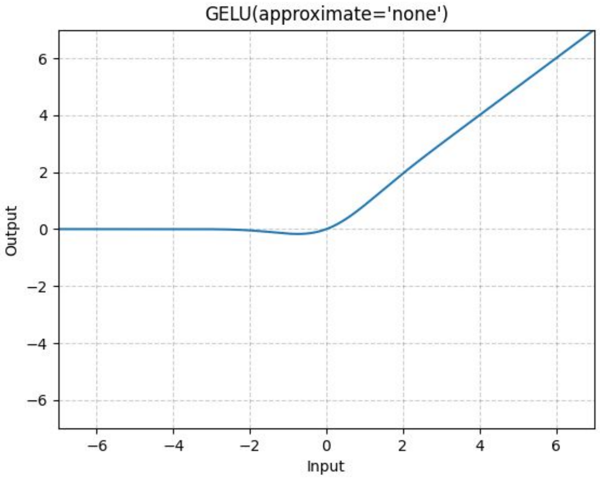
https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity

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

How to learn MLP weights?

Gradient descent!

Cornell Bowers C·IS

Calculus Review: The Chain Rule

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

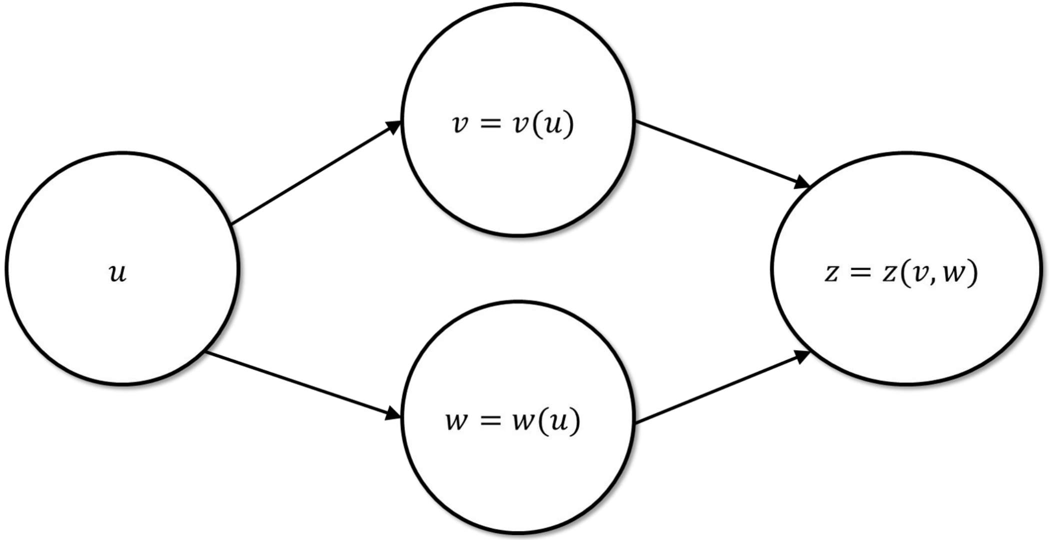
Leibniz's Notation: If $z = h(y), y = g(x)$, then $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

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

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

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

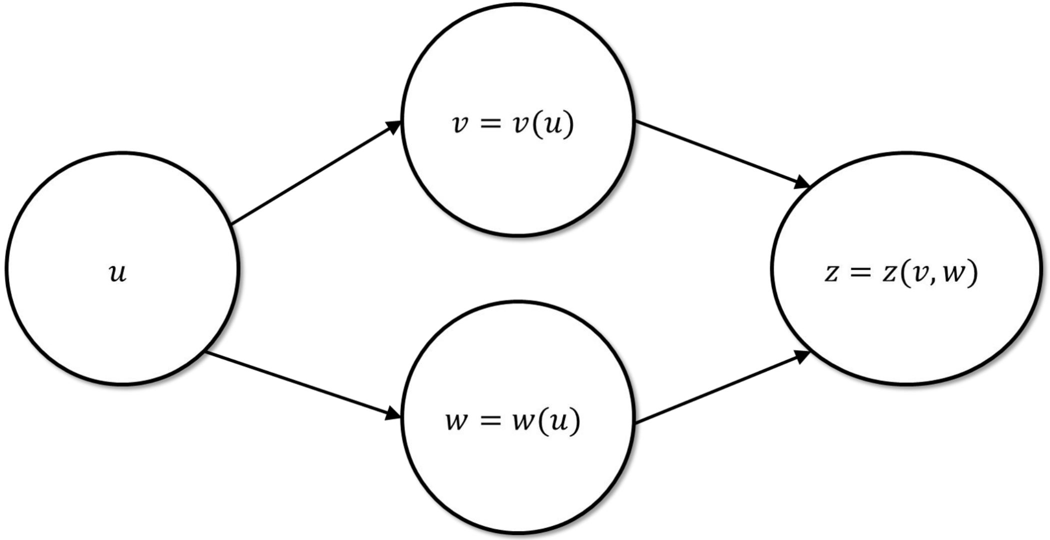
Multivariate Chain Rule



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

Cornell Bowers C·IS

Multivariate Chain Rule



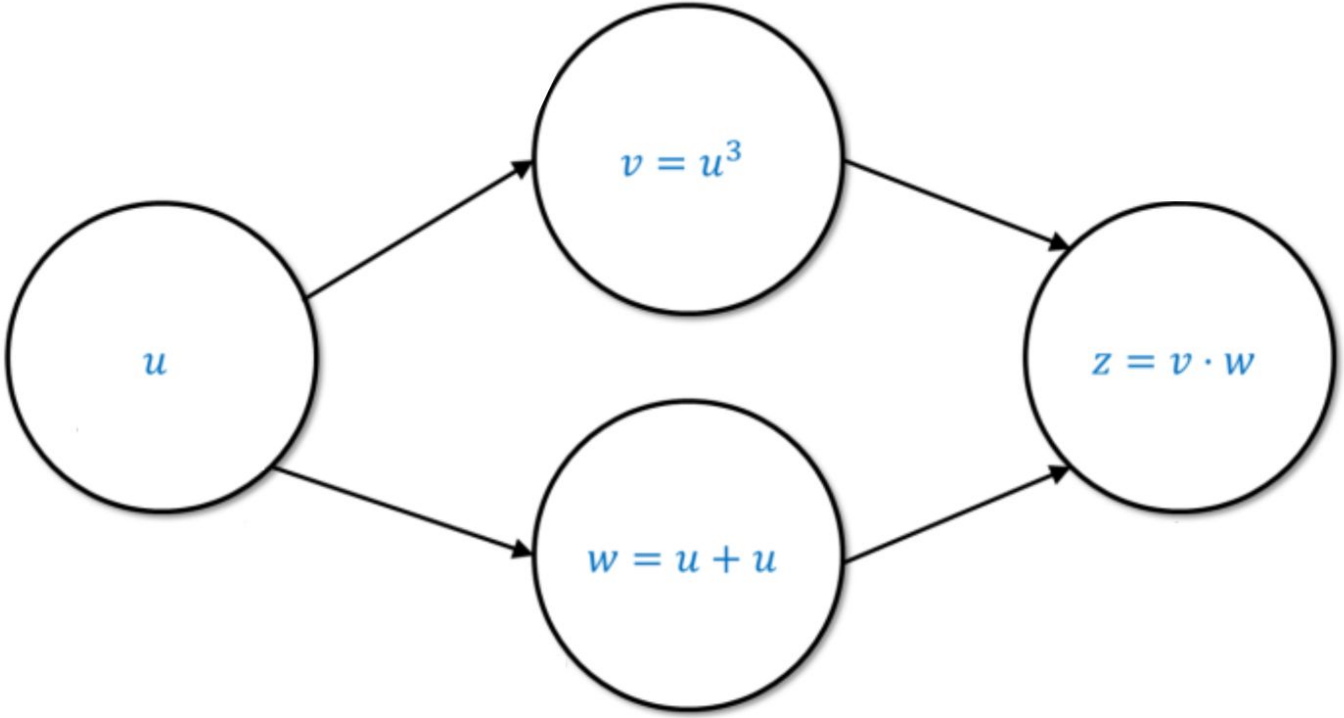
If $f(u)$ is $z = f(v(u), w(u))$, then

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

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

Cornell Bowers C·IS

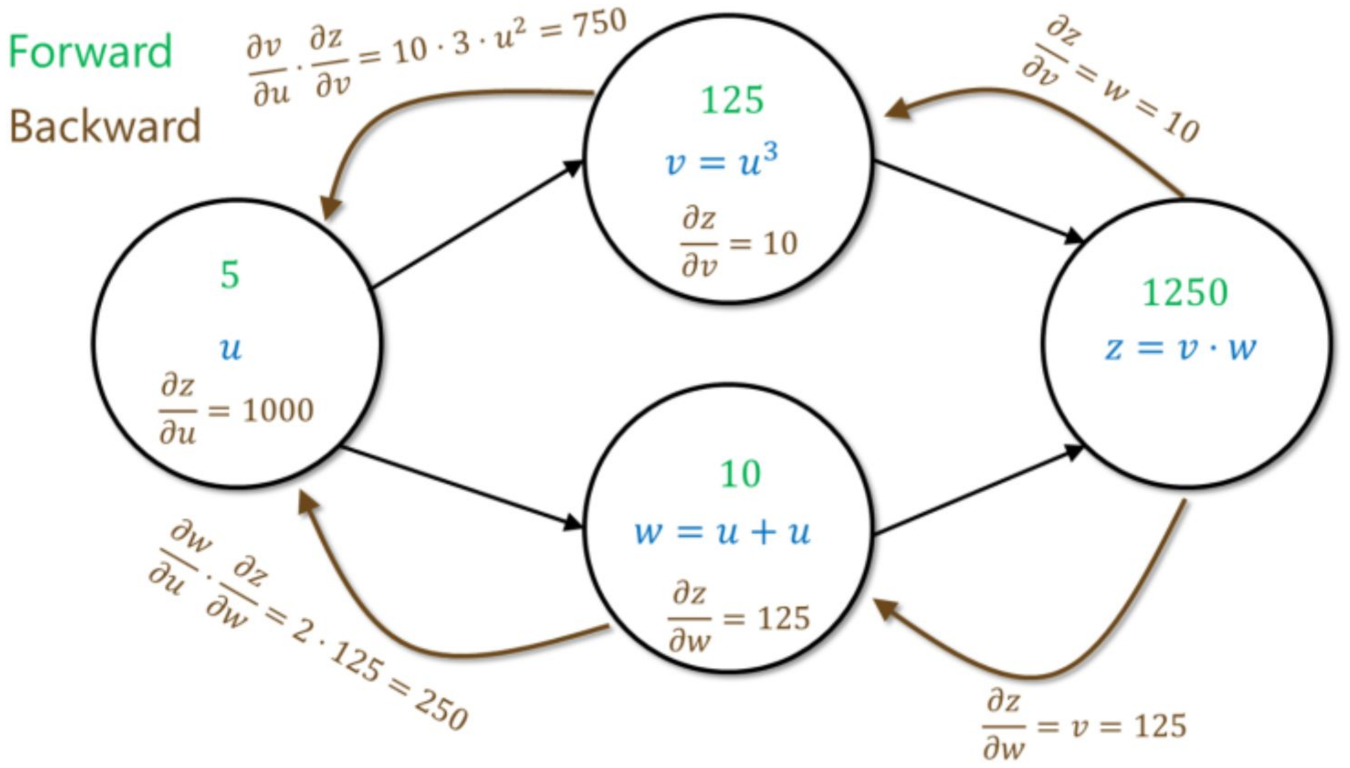
Backpropagation- An Example



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

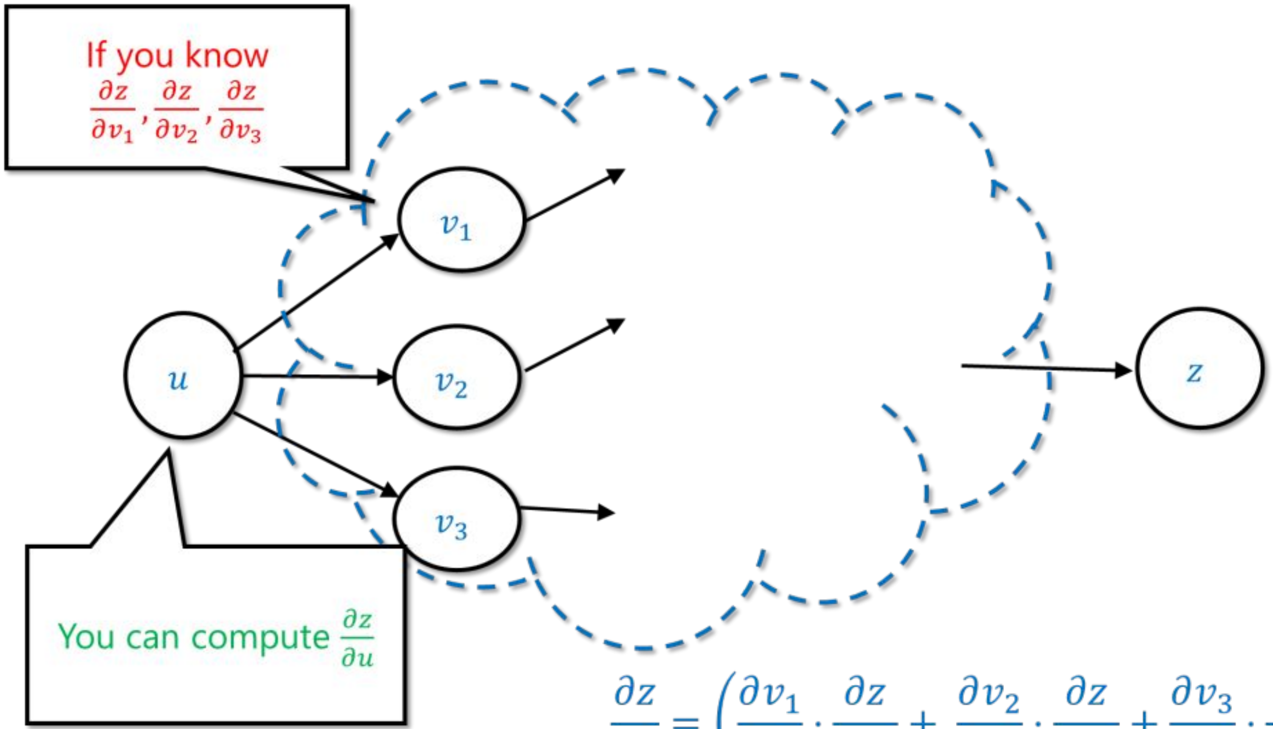
Backpropagation- An Example

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



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/

Cornell Bowers C·IS

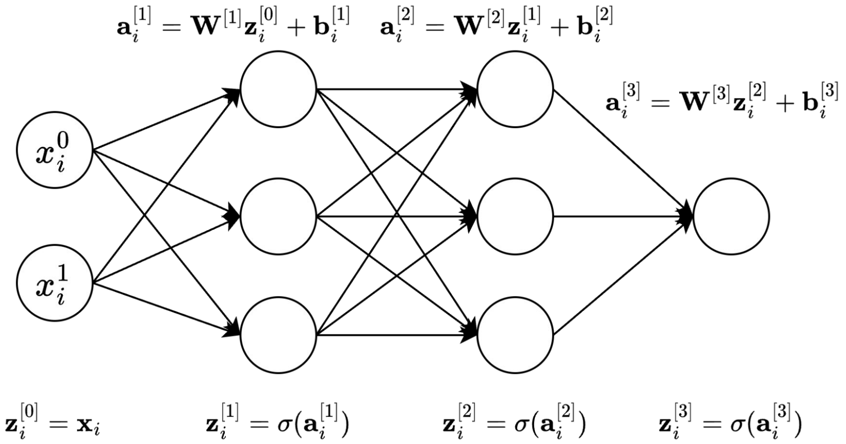
Preview

Cornell Bowers C·IS

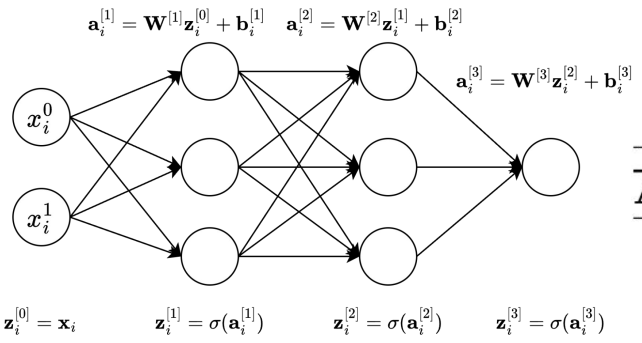
Backpropagation- MLPs

Algorithm Forward Pass through MLP

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



Backpropagation- MLPs



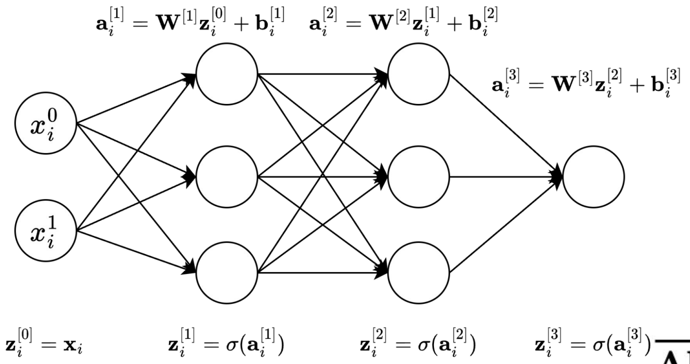
Algorithm Forward Pass through MLP

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 $\mathbf{W}^{[1]}, \dots, \mathbf{W}^{[L]}$, 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: $\mathbf{z}^{[L]}$

Algorithm Backward Pass through MLP (Detailed)

Input: $\{\mathbf{z}^{[1]}, \dots, \mathbf{z}^{[L]}\}$, $\{\mathbf{a}^{[1]}, \dots, \mathbf{a}^{[L]}\}$, 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: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$

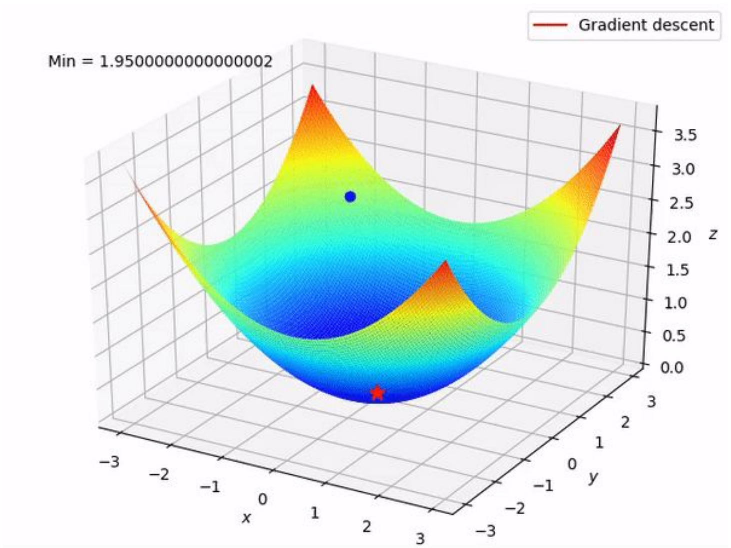
Takeaways

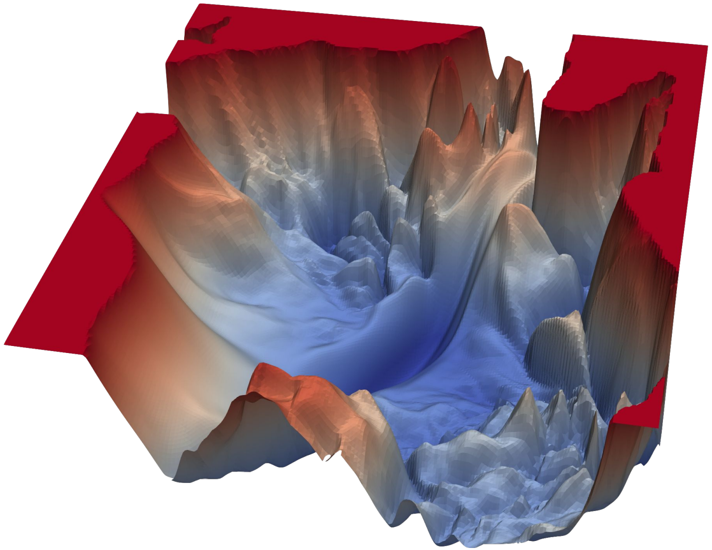
- MLPs consist of stacks of perceptron units
- MLPs can learn complex decision boundaries
  by composing simple features into more complex features
- Learn MLP weights with gradient descent
  - Backpropagation efficiently computes gradient

Cornell Bowers C·IS

Next Week

A deep dive into training neural networks!





https://arxiv.org/abs/1712.09913