Spring 2025

The University of Texas at Austin

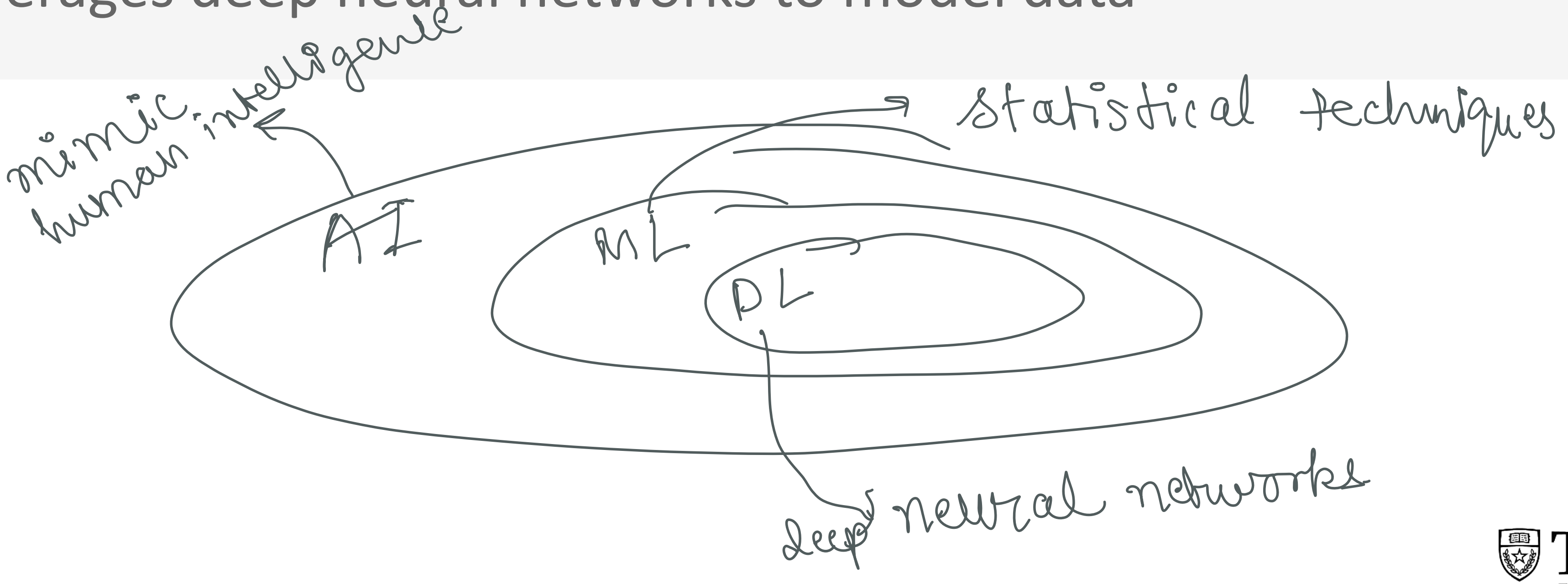
Machine Learning WB CS391L

Lecture 9 - Introduction to Deep Learning

Nilesh Gupta, Inderjit Dhillon

What is Deep Learning?

Deep Learning (DL) is a subfield of Machine Learning (ML) that leverages deep neural networks to model data



Why Deep Learning?

Automatically learn representations from data, eliminating the need for manual feature engineering

Deep Learning - Enablers

Hardware - GPUs happened!

Data - Internet happened!

Frameworks - Pytorch, Tensorflow, JAX happened!

Deep Learning - Applications

Any intelligent task!

- Object recognition
- Self driving
- Playing games
- Conversation agent
- Media (image/video) generation
- …
- Artificial general intelligence?

Deep Learning - Key Components

Model - a "neural network" to map input data to output prediction

Loss - a scalar that quantifies how well a model fits our data

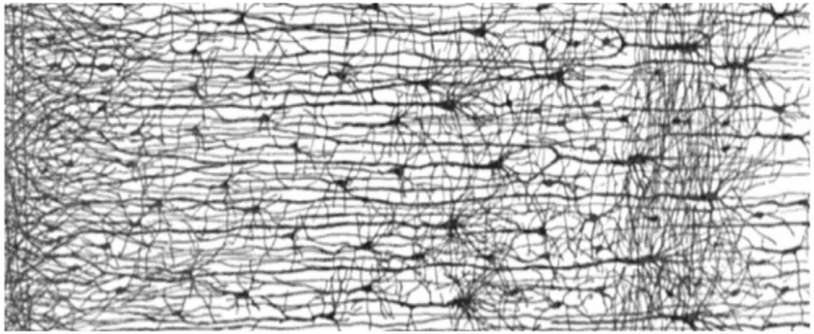
Optimization - a process (typically "gradient descent") to adjust the model parameters to minimize the loss

Model

Neural Networks

Neural Networks - Brief History

- Early work on perceptrons in the 1950s laid the foundation
- Modeled after the human brain's neurons



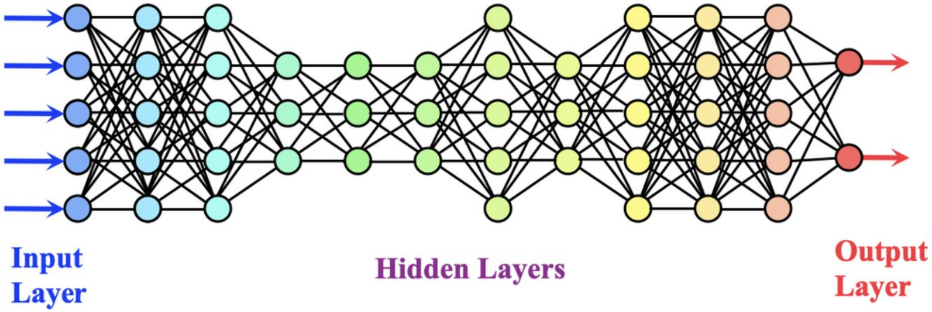


Image source - https://mriquestions.com/what-is-a-neural-network.html

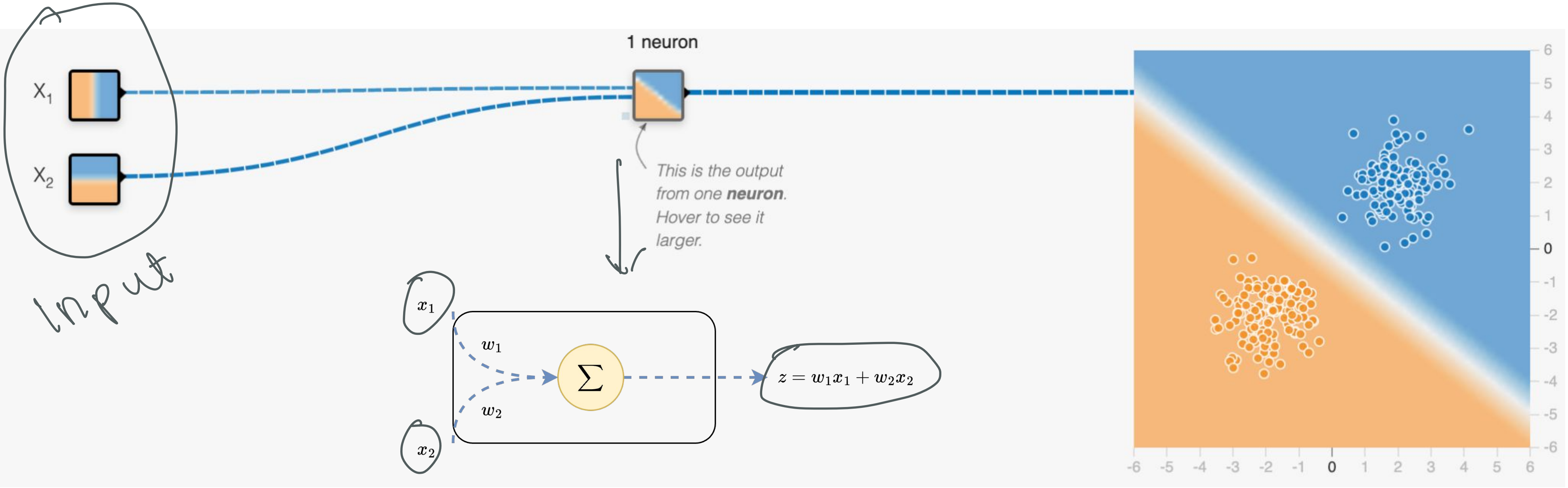
TEXAS
The University of Texas at Austin

Neural Networks - Building Block

Neuron/Perceptron a basic computational unit

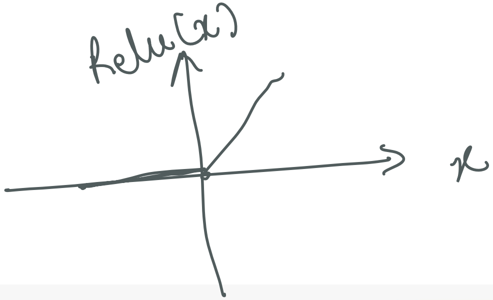
- **Inputs** - Receives multiple scalar inputs
- **Weights** - Each input has a weight, importance of that input
- **Summation** - The neuron adds up all the weighted inputs
- **Non-linearity** - A filter/activation function

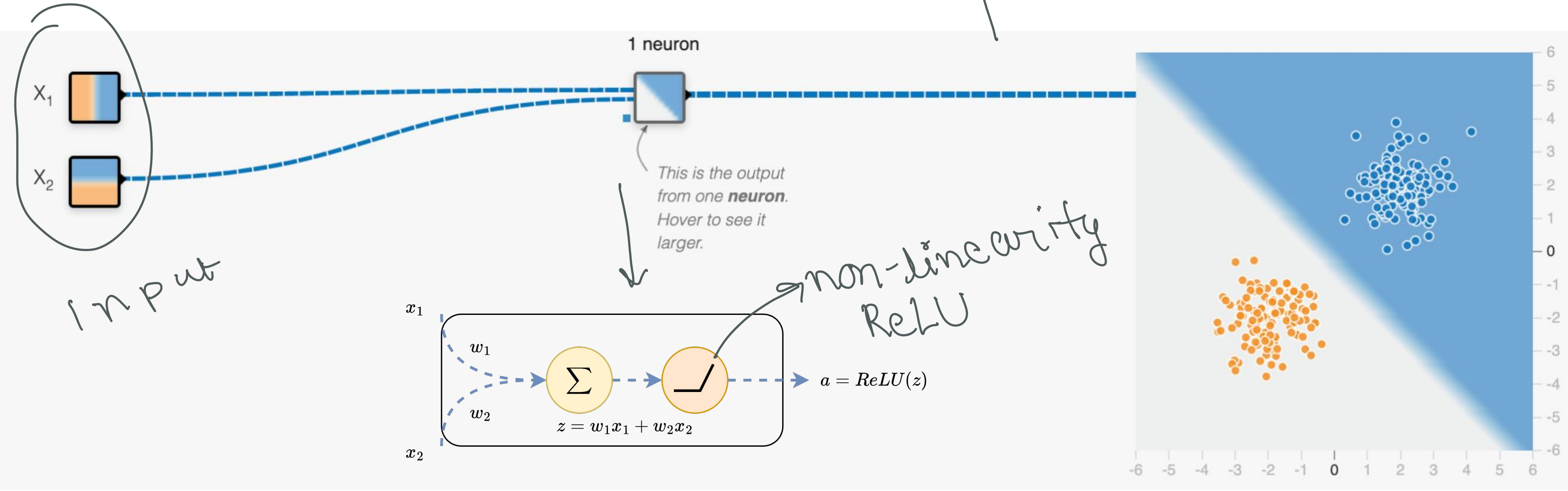
Linear Neuron



https://playground.tensorflow.org/

Non-linear Neuron





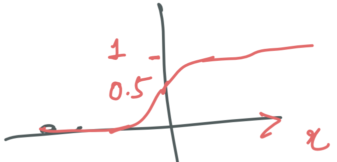
https://playground.tensorflow.org/

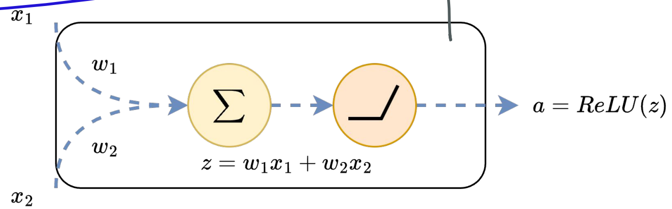
Non-linearities

$\mathbb{R}$

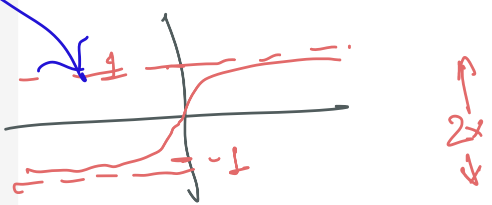
- ReLU $g(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$

- Sigmoid $g(x) = \dfrac{1}{1+e^{-x}}$





- Tanh $g(x) = \tanh(x)$



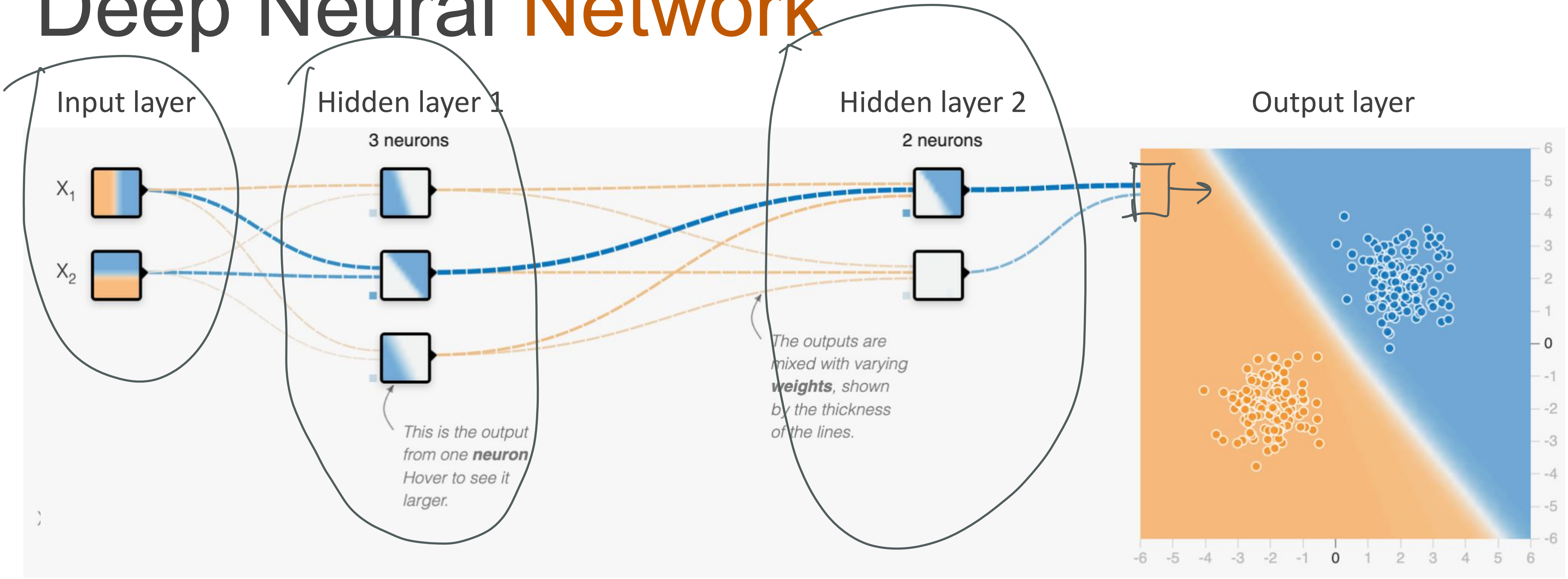
- Softmax $g(x)_i = \dfrac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$

$x \in \mathbb{R}^n$ $\quad g: \mathbb{R}^n \to \mathbb{R}^n$

$$\boxed{\begin{gathered}\sum_{i=1}^{n} g(x)_i = 1 \\ g(x)_i > 0\end{gathered}}$$

softmax

Deep Neural Network



A layered arrangement of neurons form a neural network

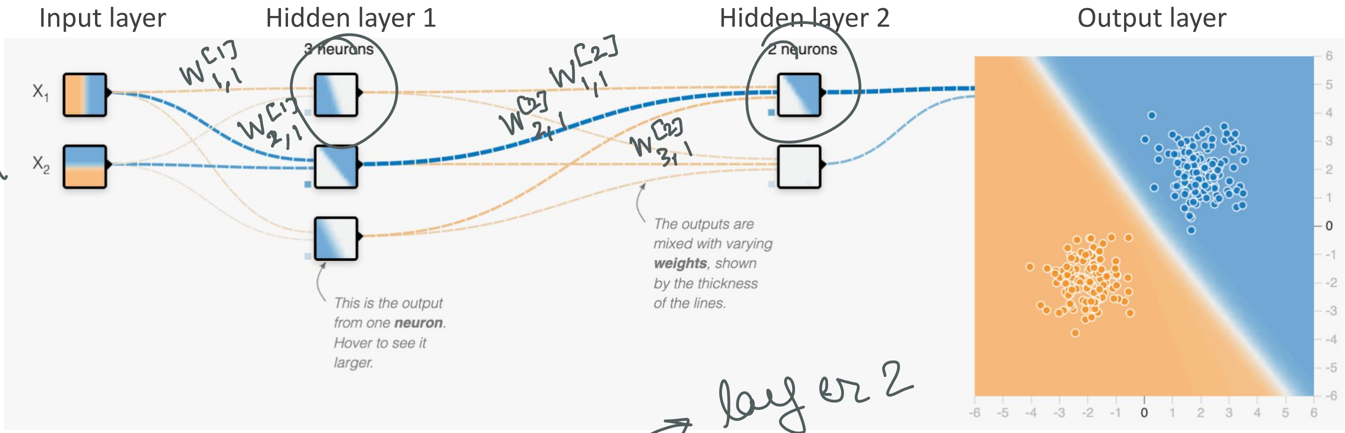
https://playground.tensorflow.org/

Deep Neural **Network**

$H^{[1]}_{1}$ → layer 1

1 → index of this neuron

$$\begin{bmatrix} W^{[1]}_{1,1} \\ W^{[1]}_{2,1} \end{bmatrix}$$



$H^{[2]}_{1}$ → layer 2

1 → index of this neuron

$$\begin{bmatrix} W^{[2]}_{1,1} \\ W^{[2]}_{2,1} \\ W^{[2]}_{3,1} \end{bmatrix}$$

https://playground.tensorflow.org/

batch-size

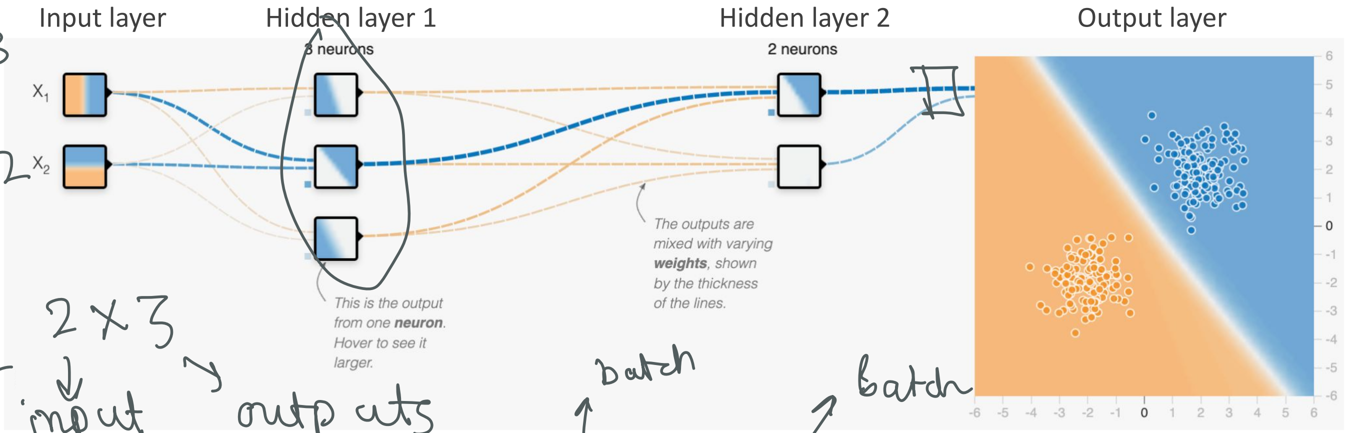
Matrix Representation

$X \in \mathbb{R}^{2\times 1}$

$H^{[1]} \rightarrow W^{[1]} \in \mathbb{R}^{2\times 3}$

$H^{[2]} \rightarrow W^{[2]} \in \mathbb{R}^{3\times 2}$

$O \rightarrow W^{[3]} \in \mathbb{R}^{2\times 1}$



2×3

input outputs

batch

batch

$Z^{[1]} = W^{[1]^T} X \in \mathbb{R}^{3\times 1}$

$A^{[1]} = \sigma(Z^{[1]}) \in \mathbb{R}^{3\times 1}$

$Z^{[2]} = W^{[2]^T} A^{[1]} \in \mathbb{R}^{2\times 1} \rightarrow$ batch

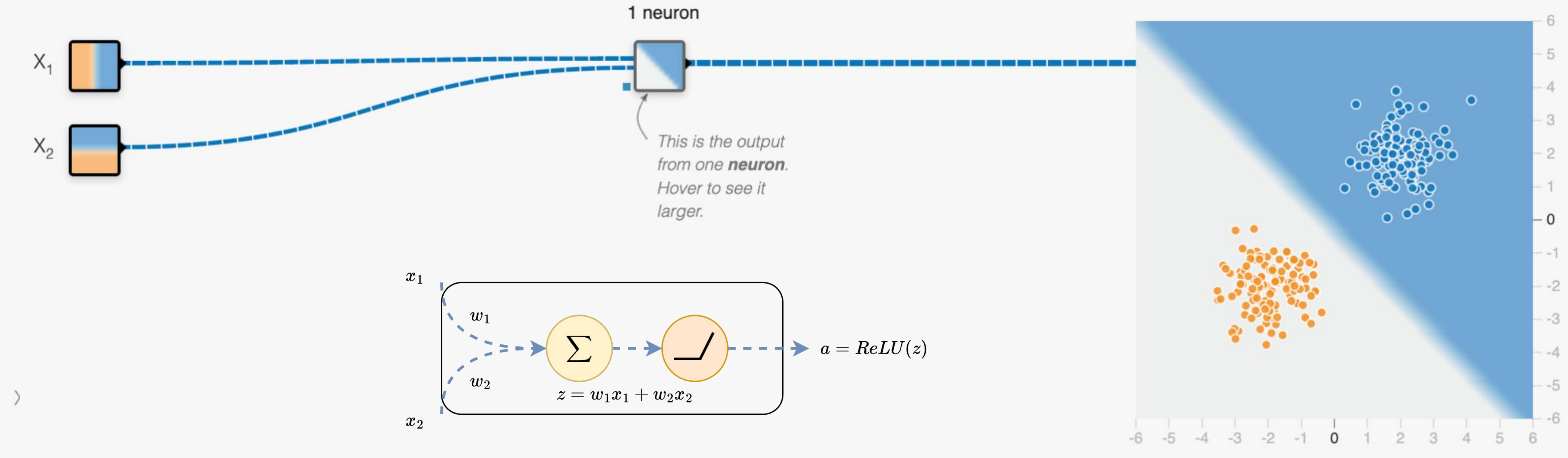
$A^{[2]} = \sigma(Z^{[2]})$

$Z^{[3]} = W^{[3]^T} A^{[2]} \in \mathbb{R}^{1\times 1} \rightarrow$ batch

$A^{[3]} = \sigma(Z^{[3]}) \in \mathbb{R}^{1\times 1}$

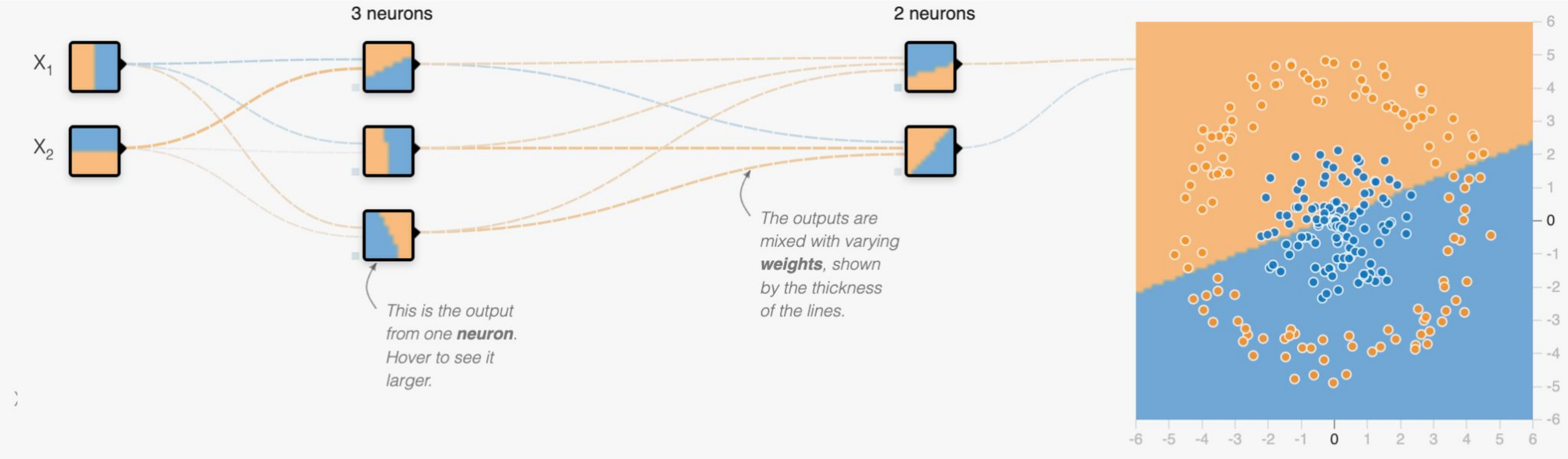
https://playground.tensorflow.org/

Why Non-linearity?



https://playground.tensorflow.org/

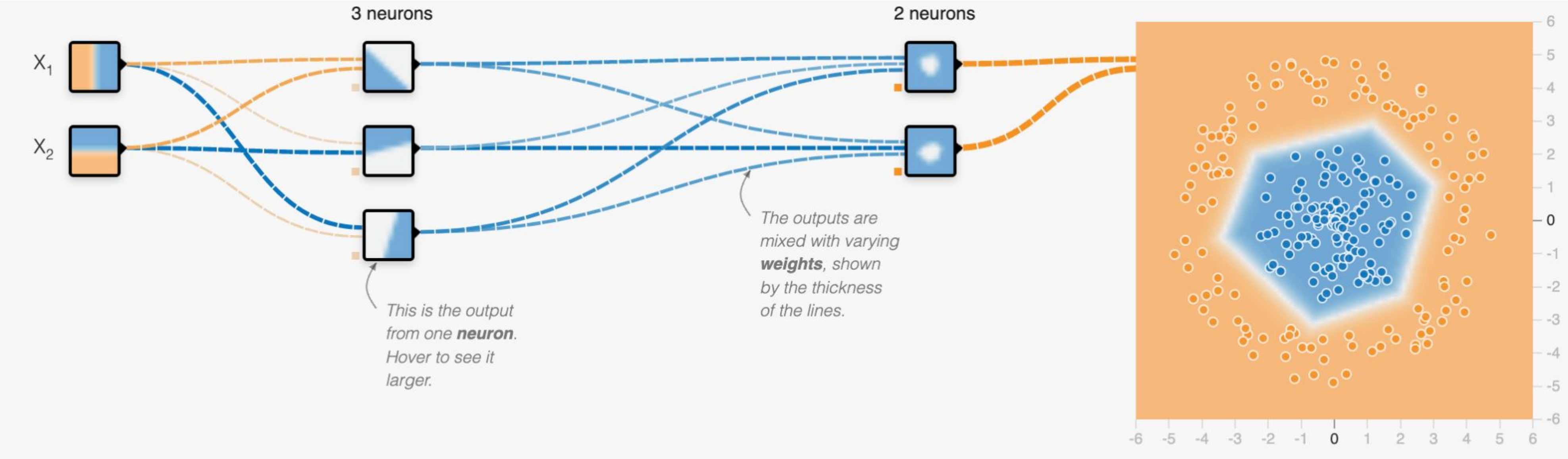
Why Non-linearity?



Combination of linear neurons is still linear

https://playground.tensorflow.org/ (link)

Why Non-linearity?



Why?

Combination of non-linear neurons is much more expressive!

https://playground.tensorflow.org/ (link)

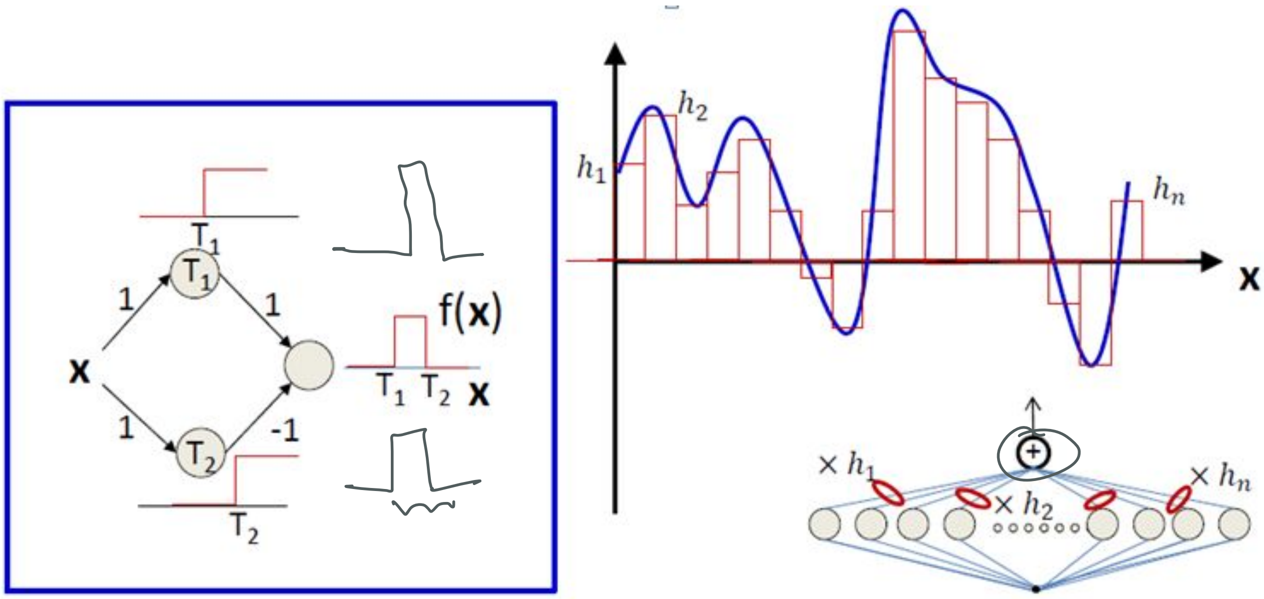
Universal Approximation Theorem

A neural network with

- at least one hidden layer,
- using a sufficiently large number of neurons and,
- an appropriate activation function

can approximate any continuous function on a compact subset of reals to any desired degree of accuracy.

Universal Approximation Theorem



Two neurons can be constructed to give a pulse function

https://medium.com/analytics-vidhya/neural-networks-and-the-universal-approximation-theorem-e5c387982eed

Loss Functions

Loss

- A measure of how “good” our model is for an input
- Usually quantified as the cost of an incorrect prediction

Empirical Loss

The total loss averaged over entire dataset

$n \rightarrow$ data points

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} l(\theta, x^{(i)})$$

Empirical loss | risk.

Loss Function MSE

Mean Squared Error (MSE)

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$\mathbb{R}$

$$\ell = (\hat{y}_i - y_i)^2$$

$\mathbb{R}$

Loss Function BCE

Binary Cross Entropy (BCE)

Logistic regression

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\right)$$

$y_i = \{0, 1\}$

$\hat{y}_i = P(\text{class being present}) \in (0, 1)$

Loss Function CE

Generalization to C class classification problem

Cross Entropy (CE)

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log \hat{y_{ij}}$$

C classes to classify into

$C=2 \rightarrow$ BCE

$$\hat{y}_i = \underbrace{[\qquad\qquad]}_{\text{probability dist.}} \in \mathbb{R}^C$$

probability dist. $\rightarrow$ softmax

TEXAS
The University of Texas at Austin

Example Task 1

Input

https://www.megavoxels.com/learn/how-to-make-a-pixel-art-cat/

Task - is there a cat in the image?

Loss - BCE

Example Task 2

Input

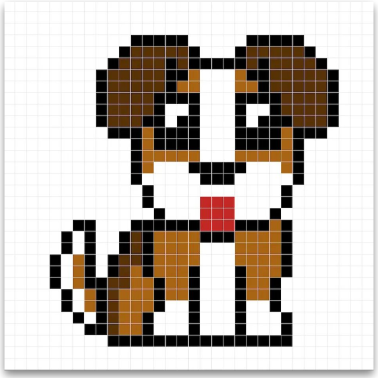
https://www.megavoxels.com/learn/how-to-make-a-pixel-art-cat/

Task - determine the age of the cat?

Loss - MSE

Example Task 3

Input



Task - determine cat vs dog vs bird?

Loss - CE softmax

3 - way

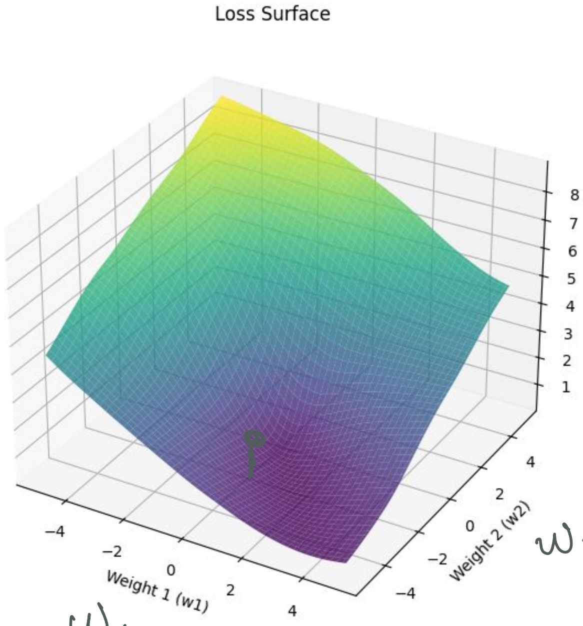
https://www.megavoxels.com/learn/how-to-make-a-pixel-art-cat/

Optimization

Gradient Descent & Backprop

Optimization Overview

Minimize the loss by adjusting the model parameters



loss

$= L(w_1, w_2, D)$

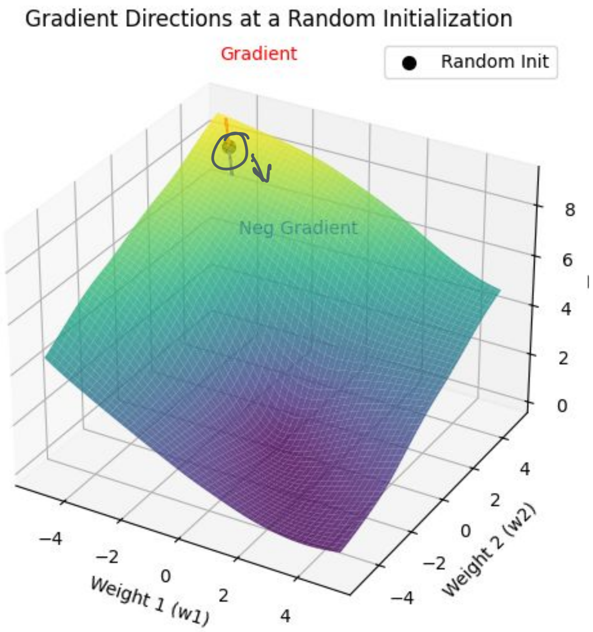
$\hookrightarrow$ scalar

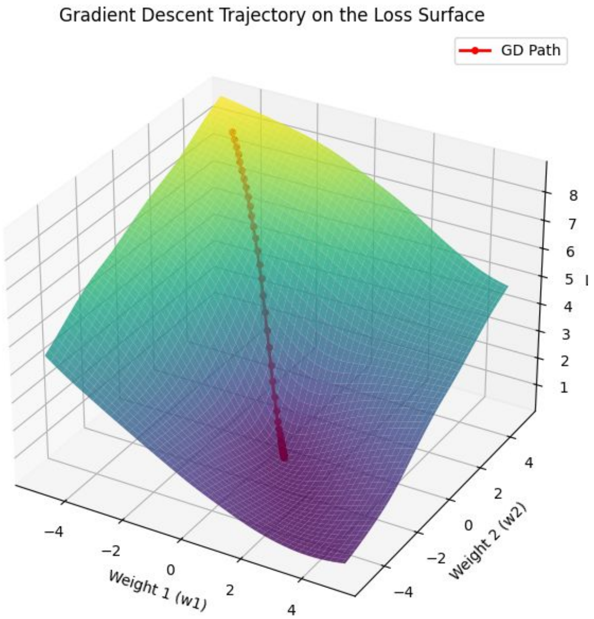
w_2

w_1

Gradient Descent

Iteratively update parameters in opposite direction of the gradient





TEXAS
The University of Texas at Austin

Gradient Descent

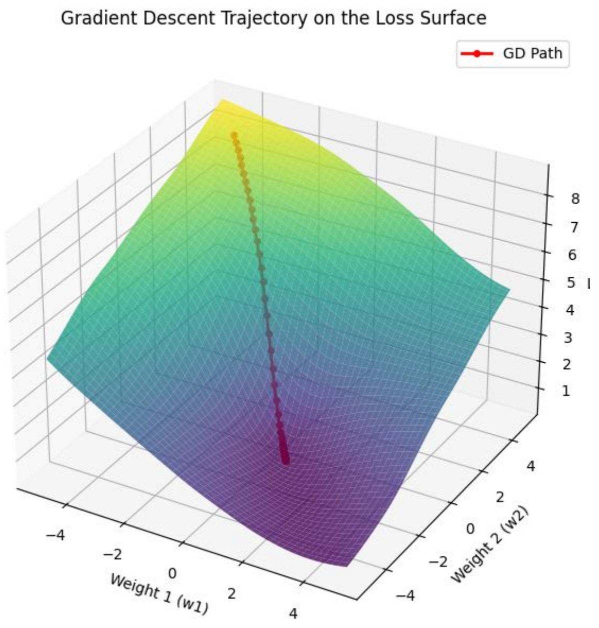
Initialize parameters (weights w)

For each iteration

1. Compute the loss L
2. Compute gradients dL/dw
3. Update parameters w = w - η * (dL/dw)

where η is the learning rate

scale



TEXAS
The University of Texas at Austin

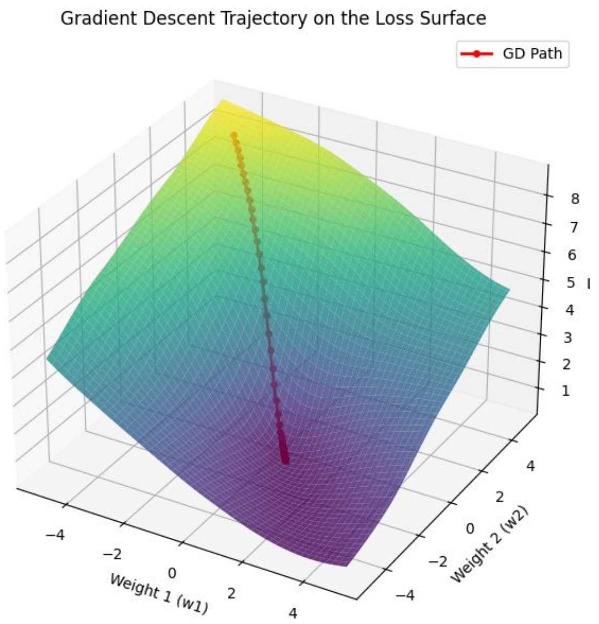
Gradient Descent

Initialize parameters (weights w)

For each iteration

1. Compute the loss L
2. Compute gradients dL/dw
3. Update parameters w = w - η * (dL/dw)

where η is the learning rate



TEXAS
The University of Texas at Austin

Backpropagation

- Neural networks require an efficient way to compute the gradients for all parameters
- **Backpropagation** - chain rule to propagate errors from the output back to the input layer

Backpropagation

Chain Rule - for a parameter θ that affects the loss $\mathcal{L}$ through an intermediate variable z

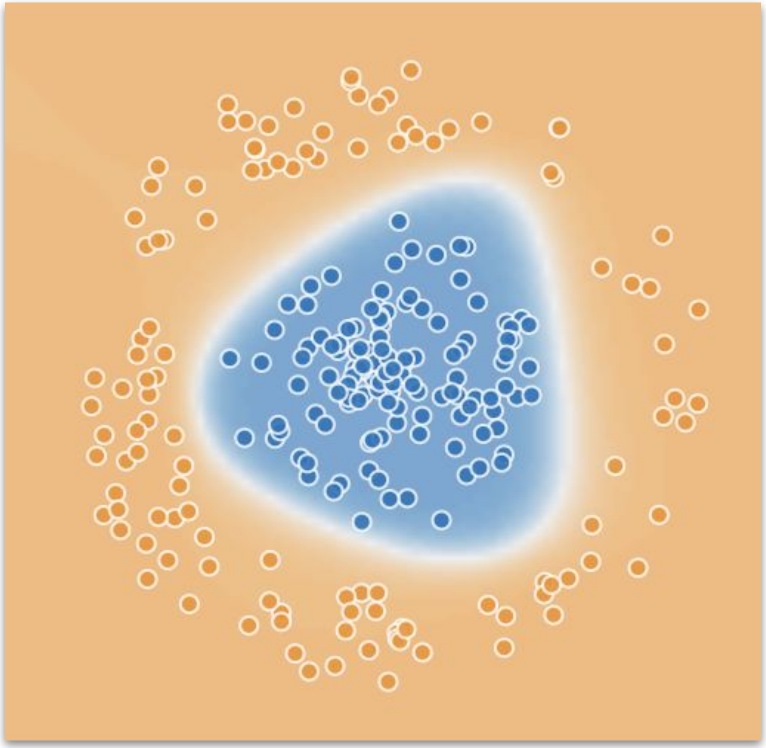
$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial \theta}$$

Forward Pass - Compute all intermediate activations $Z^{[l]}$ and $A^{[l]}$

Backward Pass - Starting at the output layer, compute the gradient and propagate it backwards

Sample DL Algorithm

Task



Task - classify orange from blue

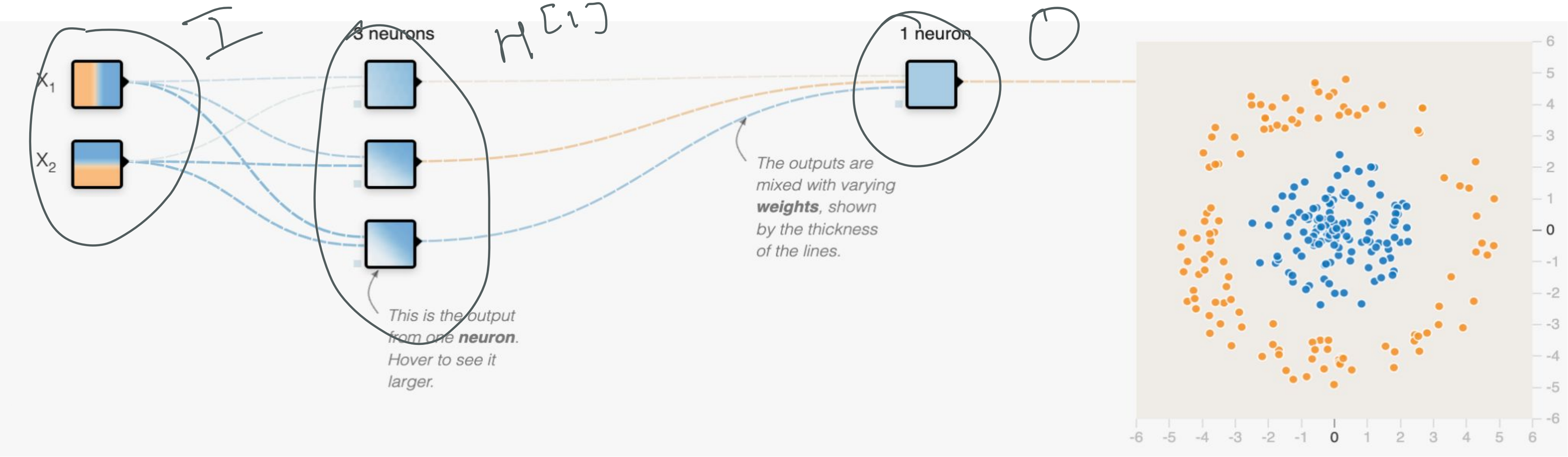
Loss

Binary Cross Entropy (BCE)

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\right)$$

Model

Sigmoid activation, 3 hidden neuron, 1 output neuron



Input layer Hidden layer Output layer

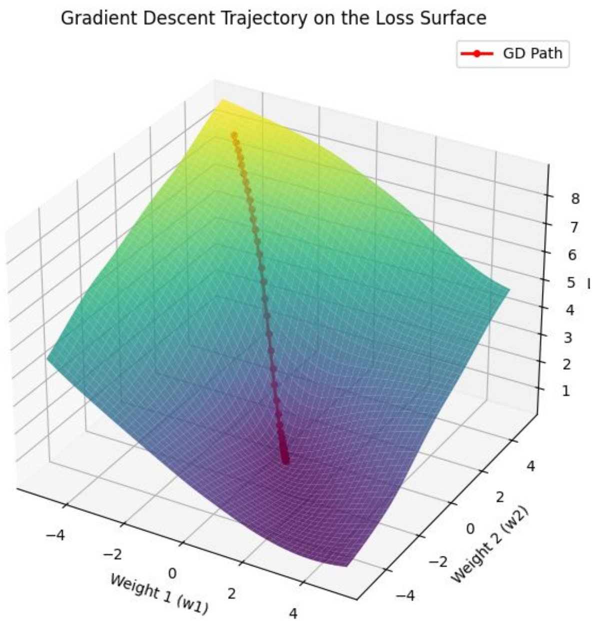
Optimization

Initialize parameters (weights w)

For each iteration

1. Compute the loss L
2. Compute gradients dL/dw
3. Update parameters w = w - η * (dL/dw)

where η is the learning rate.



Optimization - Forward Pass & Loss

$X \in \mathbb{R}^{2\times 1}$

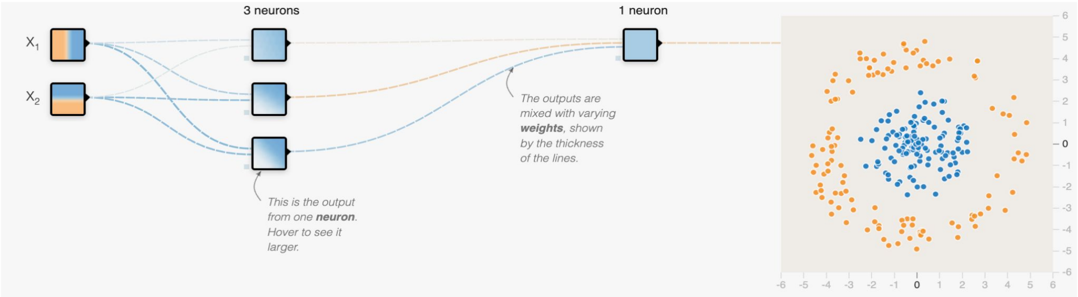
$H^{[1]} \rightarrow W^{[1]} \in \mathbb{R}^{3\times 2}$

$O \rightarrow W^{[2]} \in \mathbb{R}^{1\times 3}$

$Z^{[1]} = W^{[1]T} X \; ; \; A^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]T} A^{[1]} \; ; \; A^{[2]} = \hat{Y} = \sigma(Z^{[2]})$

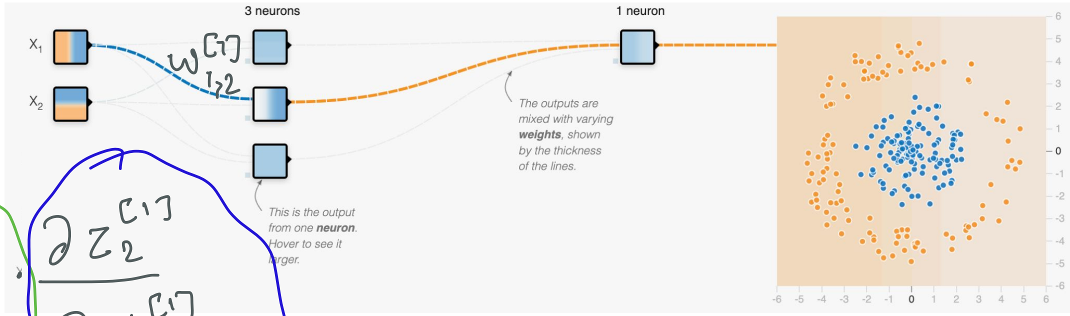
$\mathcal{L} = -(Y \cdot \log(\hat{Y}) + (1-Y) \cdot \log(1-\hat{Y}))$



TEXAS
The University of Texas at Austin

Optimization - Backward Pass

$$\boxed{\text{Compute } \frac{\partial L}{\partial w^{[1]}_{1,2}}}$$



Chain Rule

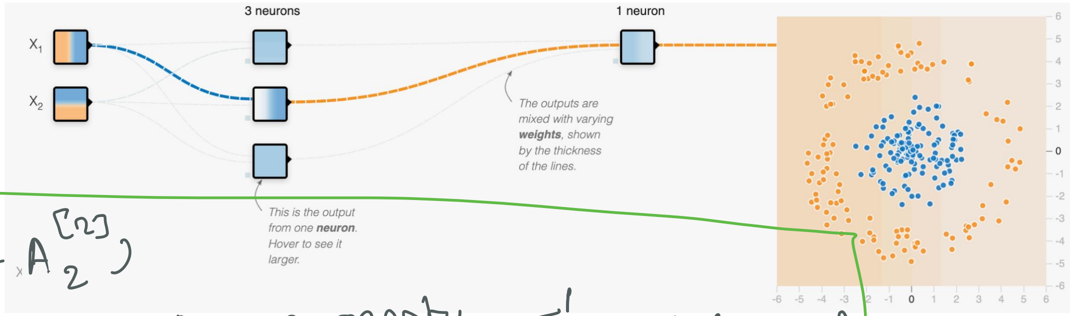
$$\frac{\partial L}{\partial w^{[1]}_{1,2}} = \frac{\partial L}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}_2} \cdot \frac{\partial A^{[1]}_2}{\partial Z^{[1]}_2} \cdot \frac{\partial Z^{[1]}_2}{\partial W^{[1]}_{1,2}}$$

Forward

$$L = Y \log(\hat{Y}) + (1-Y)\log(1-\hat{Y}) \quad ; \quad \hat{Y} = \sigma(Z^{[2]})$$

$$\Rightarrow \frac{\partial L}{\partial Z^{[2]}} = (\hat{Y} - Y) \quad \text{(after simplifying)}$$

Optimization - Backward Pass



forward

$$Z^{[2]} = W^{[2]T} A^{[1]} \Rightarrow \frac{\partial Z^{[2]}}{\partial A_2^{[1]}} = W_2^{[2]}$$

$$A_2^{[2]} = \sigma(Z_2^{[1]}) \Rightarrow \frac{\partial A_2^{[2]}}{\partial Z_2^{[1]}} = A_2^{[2]} \cdot (1 - A_2^{[2]})$$

$\hookrightarrow$ using property $\sigma' = \sigma \cdot (1 - \sigma)$

$$Z^{[1]} = W^{[1]T} X \Rightarrow Z_2^{[1]} = W_{1,2}^{[1]} X_1 + W_{2,2}^{[1]} X_2 \Rightarrow \frac{\partial Z_2^{[1]}}{\partial W_{1,2}^{[1]}} = X_1$$

$$\frac{\partial L}{\partial W_{1,2}^{[1]}} = (\hat{Y} - Y) \cdot W_2^{[2]} \cdot A_2^{[1]}(1 - A_2^{[1]}) \cdot X_1$$

$\hookrightarrow$ Final gradient for $W_{1,2}^{[1]}$

Full Algorithm

- **Step 1 (Forward Pass)**: Compute activations
- **Step 2 (Loss Computation)**: Evaluate the loss using the predicted probability and the true probability
- **Step 3 (Backward Pass)**: Use backpropagation to compute gradients for all parameters
- **Step 4 (Parameter Update)**: Adjust model weights using gradient descent

Repeat for multiple steps until convergence

In Practice

- **Efficiency** - mini-batch gradient descent
- **Learning Rate** - adaptive learning rate gradient descent
- **Regularization** - dropout, weight decay, early stopping, etc