

Introduction to Deep Learning

Lecture 14 – HCCDA-AI

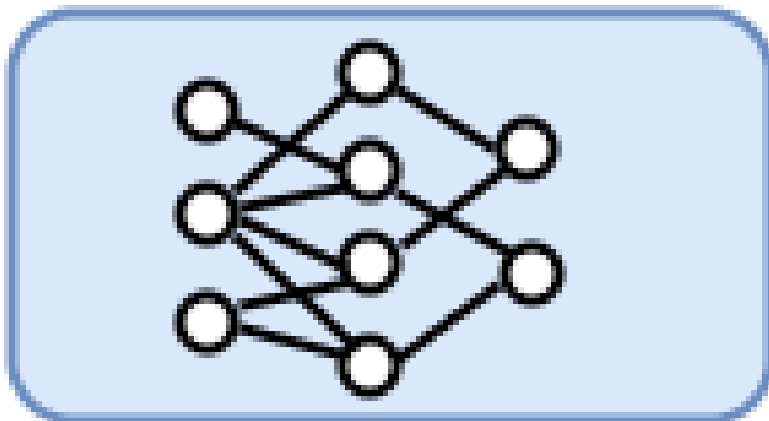
Dr. Muhammad Sajjad

R.A: Imran Nawar

20 July 2025

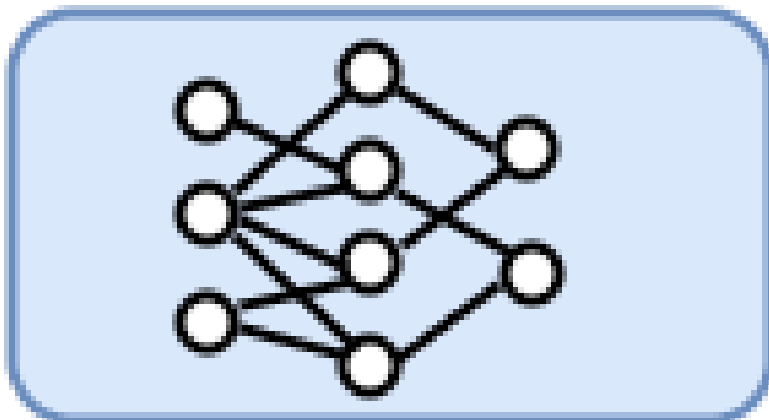
Generating Images from Natural Language


“A photo of an astronaut riding a horse.”



Generating Language from Natural Language

“Write code in TensorFlow to train a neural network”



 Certainly! Here is an example of how you can use TensorFlow to train a neural network for binary classification:

[Copy code](#)

```
import tensorflow as tf

# Load the data
(x_train, y_train), (x_test, y_test) = load_data()

# Define the model
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=x_train.shape[1:]),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
```

This code assumes that you have a function `load_data()` that returns the training and test datasets as tuples of NumPy arrays. The input shape of the model is determined by the shape of the training data, and the model has two dense layers with 16 and 1 units, respectively, with ReLU and sigmoid activations. The model is then compiled with an Adam optimizer and a binary cross-entropy loss function, and is trained using the `fit()` method.

I hope this helps! Let me know if you have any questions.

Deep Fake Video

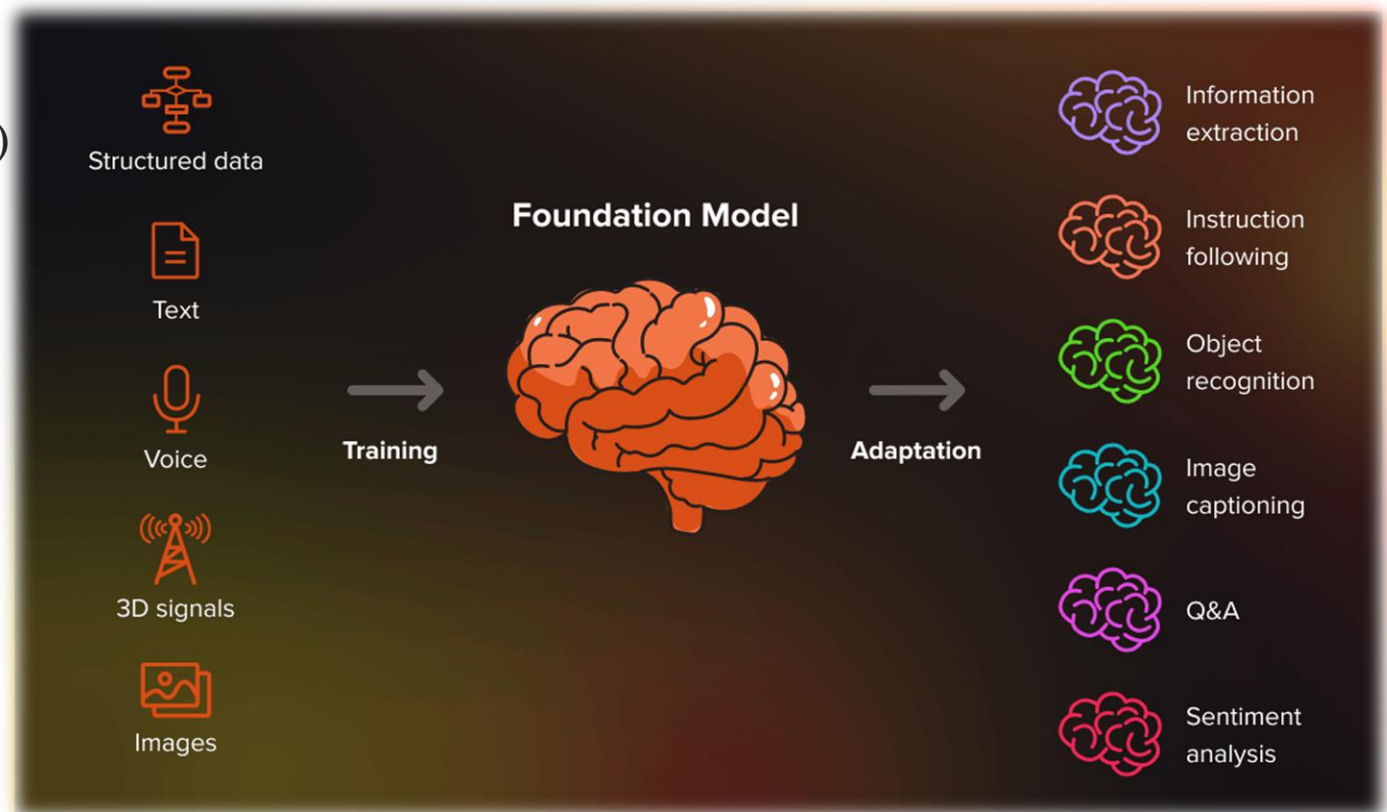


[Courtesy of Alexander Amini]

Deep Learning in 2025:

Multimodal Foundation Models

- Modern AI systems can understand and generate text, images, audio, and video, enabling more natural and human-like interactions across formats.
- **Examples:**
 - GPT-4o (OpenAI)
 - Claude Sonnet 4 (Anthropic)
 - Gemini 2.5 (Google DeepMind)
 - Grok3 (xAI)

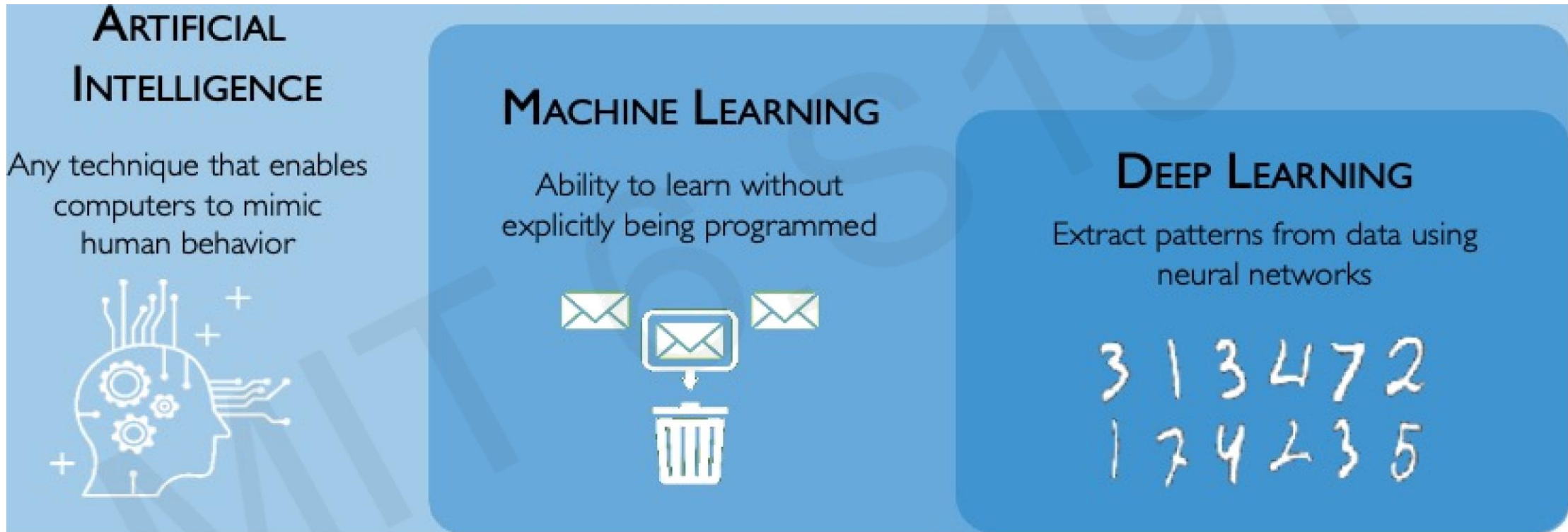


Where are Deep Learning and AI Headed?

Deep Learning is **revolutionizing** so many fields.

Healthcare, education, entertainment and beyond.

What is Deep Learning?



Teaching computers how to **learn a task** directly from **raw data**

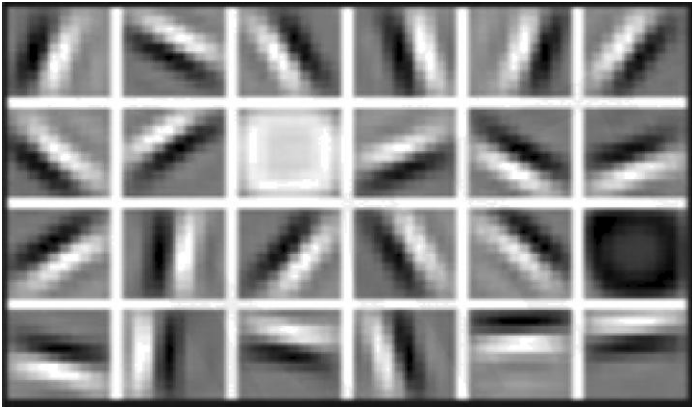
Why Deep Learning and Why Now?

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

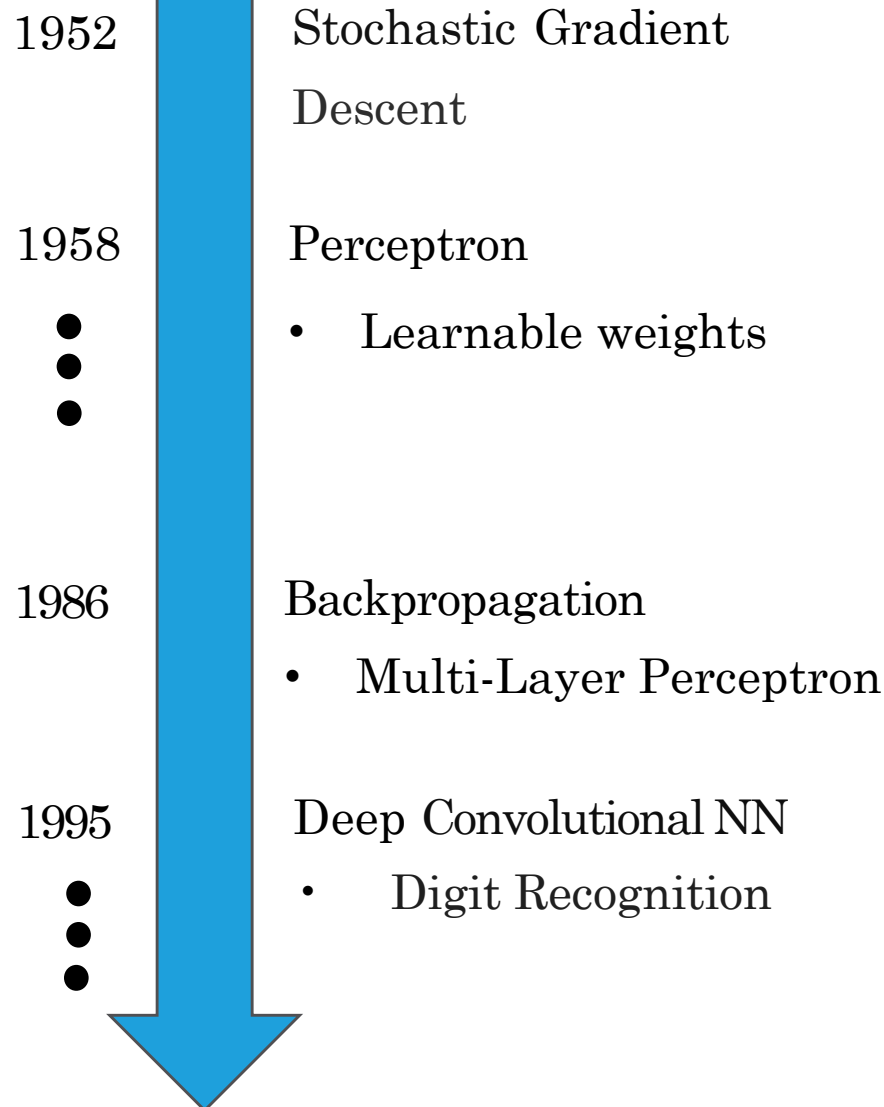
High Level Features



Facial Structure

Why Now?

Neural Networks date back decades, so why the dominance?



1) Big Data

- Larger Datasets
- Easier Collection & Storage

2) Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable

3) Software

- Improved Techniques
- New Models
- Toolboxes

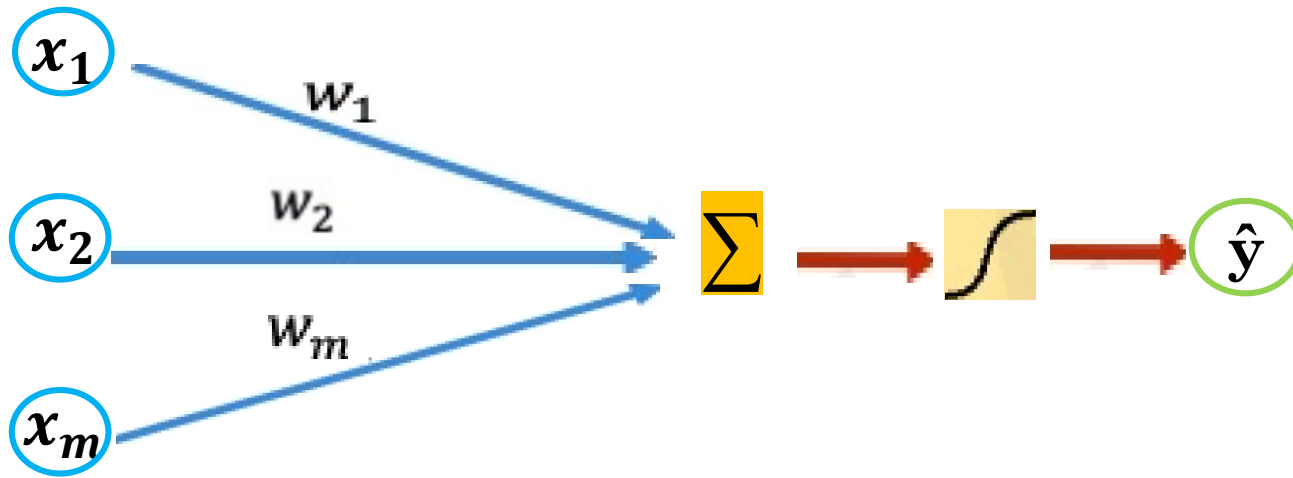
IMAGENET



The Perceptron

The structural building block of deep learning

The Perceptron Forward Propagation



Linear combination of inputs

Output

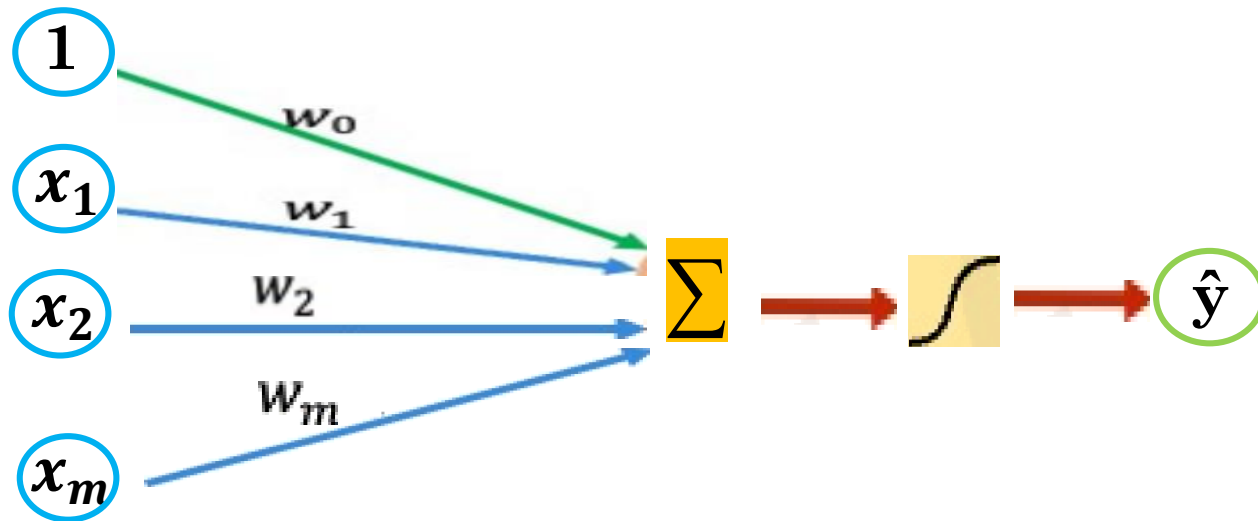
$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

Non-linear activation function

Bias

Inputs Weights Sum Non-Linearity Output

The Perceptron Forward Propagation



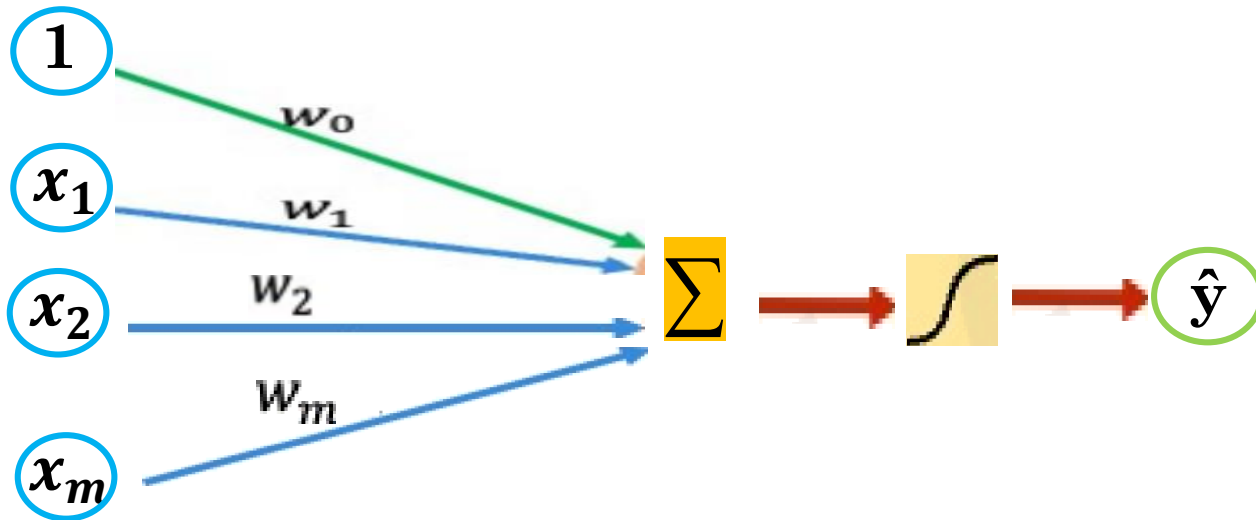
$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

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

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

Inputs Weights Sum Non-Linearity Output

The Perceptron Forward Propagation



Inputs

Weights

Sum

Non-Linearity

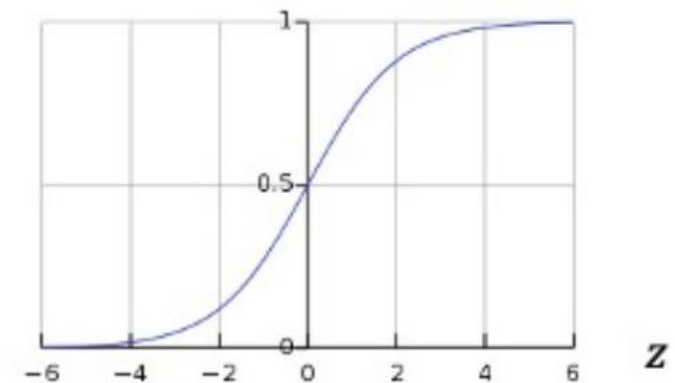
Output

Activation Functions

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

- Example: sigmoid function

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



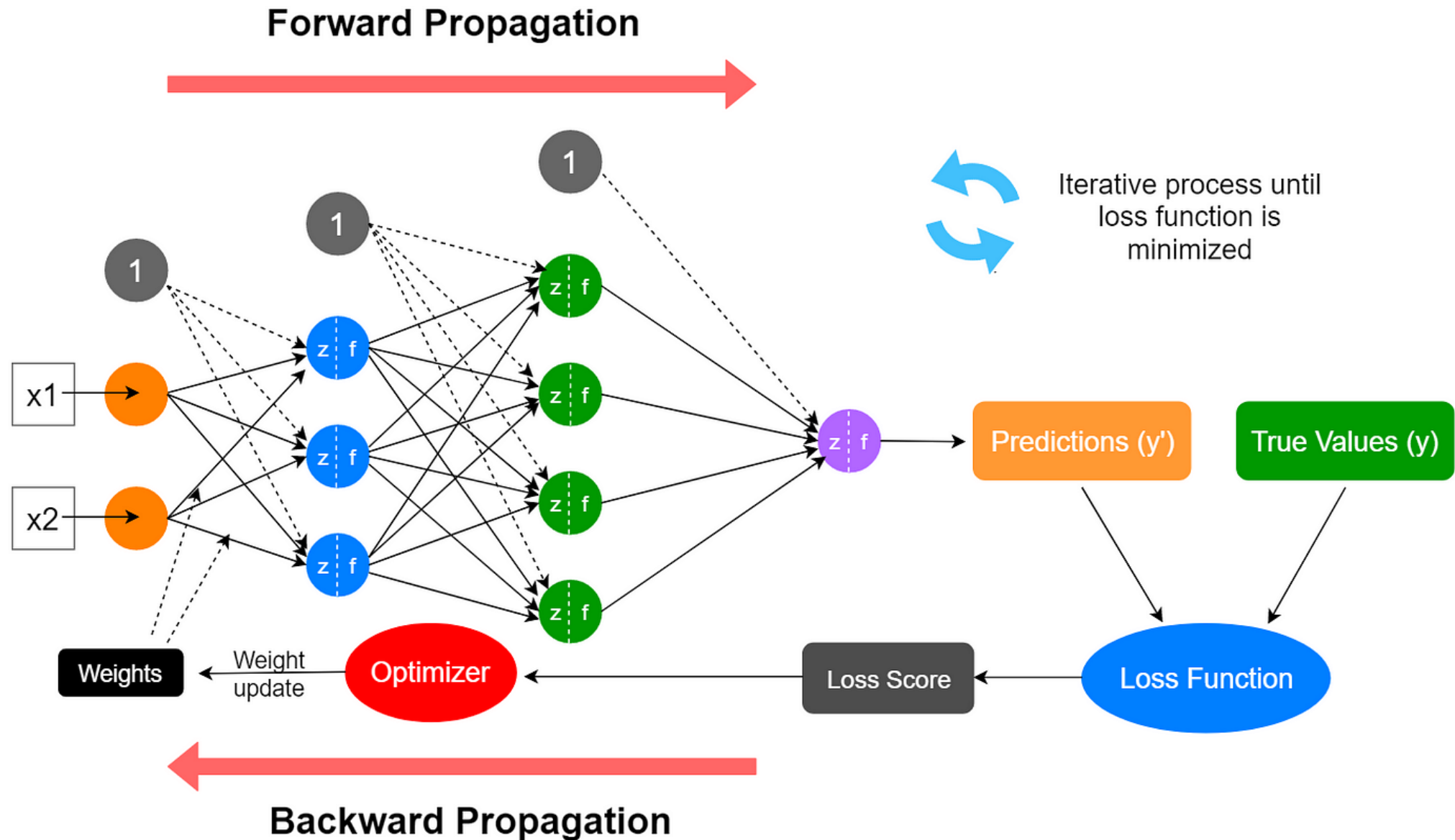
Neural Networks Training Process – Overview

- To train a neural network, we optimize its parameters using **backpropagation** and **gradient descent**.

Steps:

1. **Initialize weights & biases**
2. **Forward pass** (compute predictions)
3. **Compute loss** – Measure how far predictions are from actual values.
4. **Backpropagation** – Calculate gradients to adjust parameters.
5. **Update weights** using gradient descent
6. **Repeat** until convergence
7. **Evaluate** and adjust hyperparameters

Neural Networks Training – Visual Summary

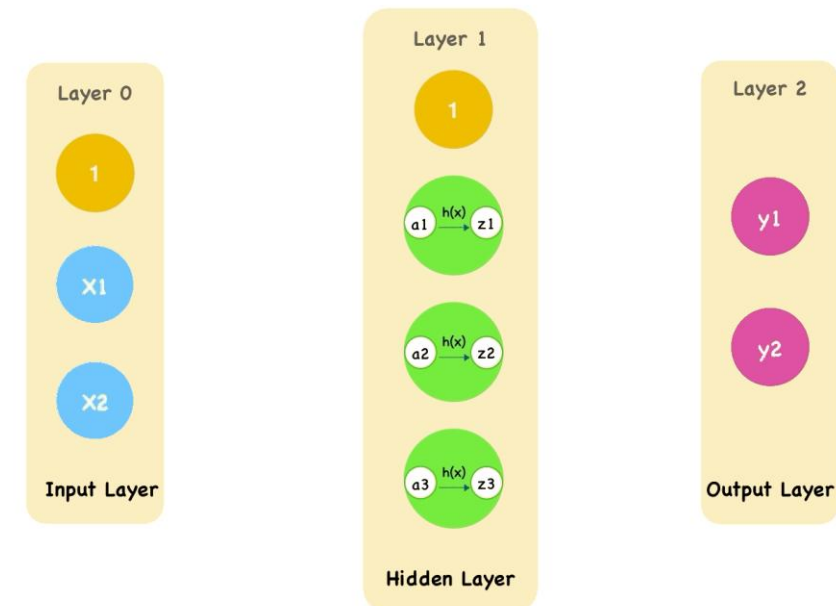
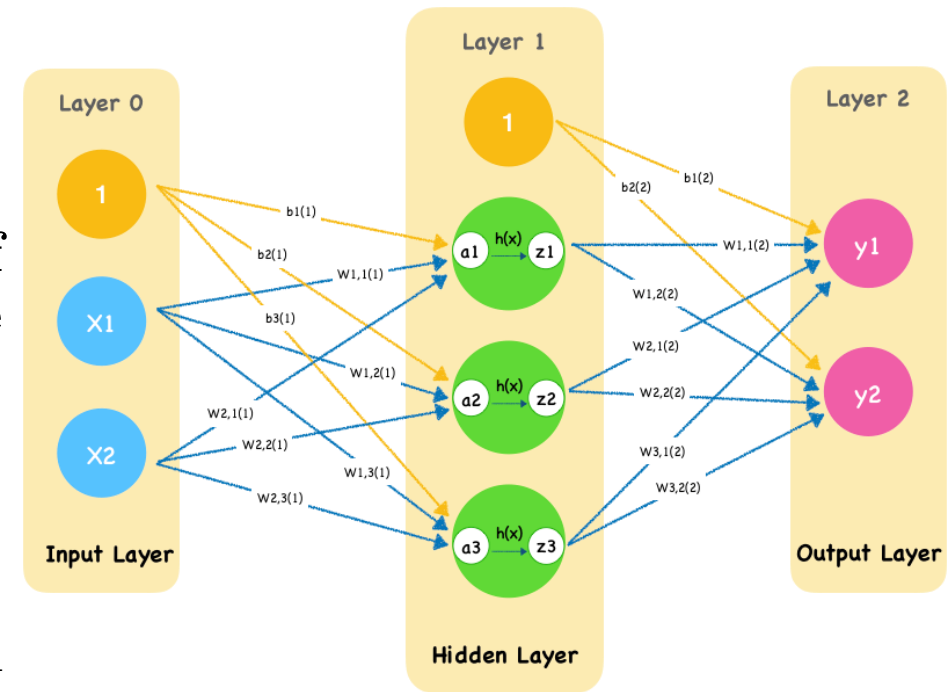


Forward Propagation

- Forward propagation refers to the process of feeding input through the network to generate predictions.

Steps:

- 1) Input data is provided to the input layer.
- 2) Weighted sum of inputs is calculated at each neuron.
- 3) Activation function is applied.
- 4) The output of each layer serves as input to the next layer.
- 5) The final layer produces predictions.



Python Code: Forward Propagation (Simple Example)

```
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Input values
x = np.array([0.5, 0.8])

# Initialize weights and bias
w = np.array([0.2, 0.4])
b = 0.1

# Compute weighted sum
z = np.dot(x, w) + b

# Apply activation function
output = sigmoid(z)
print("Final Output:", output)
```

Activation Functions

- Activation functions introduce non-linearity into the neural network, enabling it to learn and model complex patterns beyond linear relationships.

- **Common activation functions:**

- **Sigmoid:**

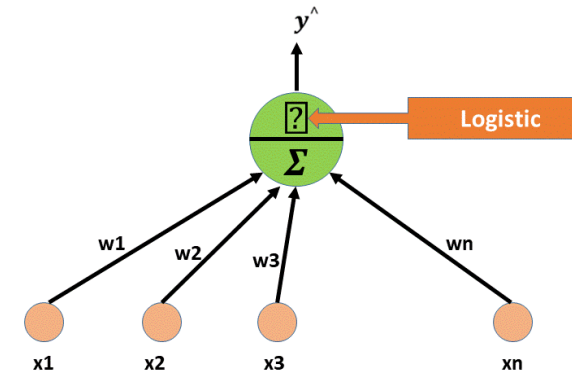
- **Output range:** $(0, 1)$
 - Good for probability based outputs (e.g., binary classification)
 - Can suffer from vanishing gradients, especially in deep networks

- **Tanh (Hyperbolic Tangent):**

- **Output range:** $(-1, 1)$
 - Centered around zero \rightarrow better for optimization than sigmoid
 - Still prone to vanishing gradient issues.

- **ReLU: (Rectified Linear Unit):**

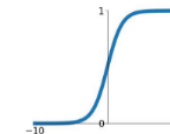
- **Output:** $\max(0, x)$
 - Most widely used due to computational efficiency
 - Helps reduce vanishing gradient problem
 - Can lead to “dead neurons” (zero gradients) if inputs are always negative



Activation Functions

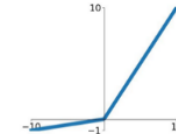
Sigmoid

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



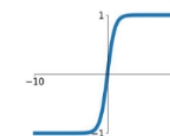
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

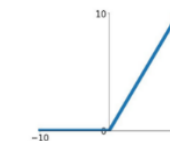


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

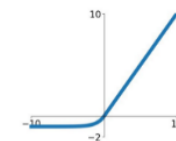
ReLU

$$\max(0, x)$$



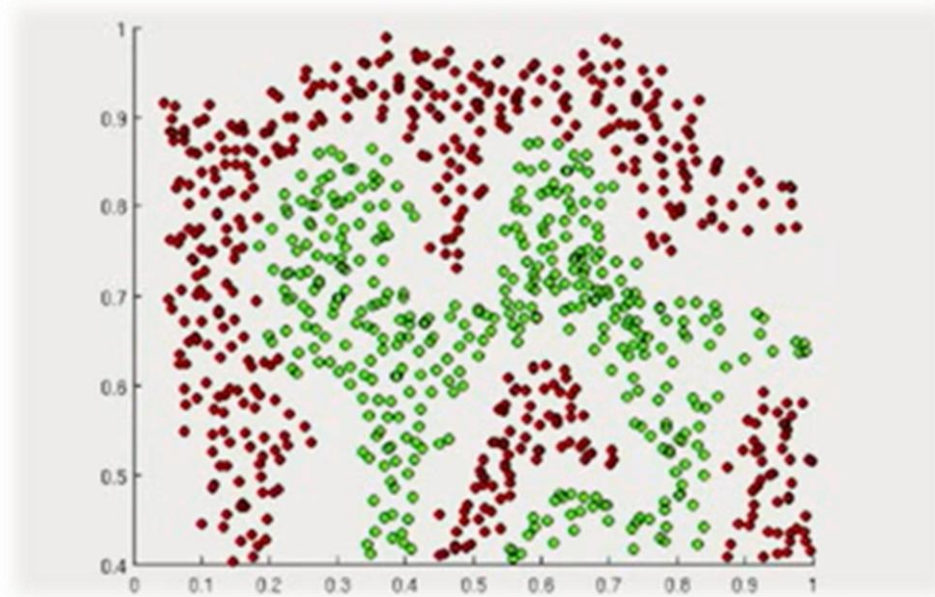
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Importance Activation Functions

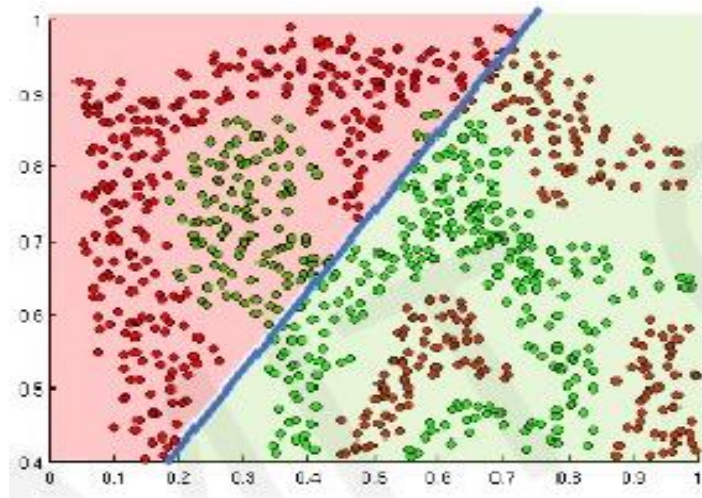
The purpose of activation functions is **to introduce non-linearity** into the network



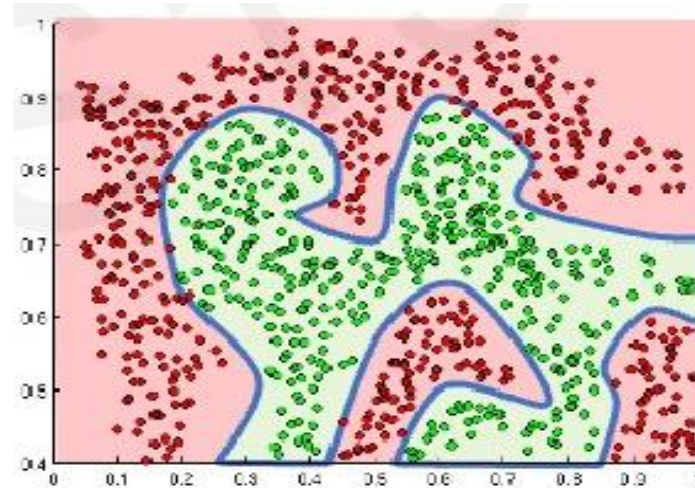
What if we wanted to build a neural network
to distinguish green vs red points?

Importance Activation Functions

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



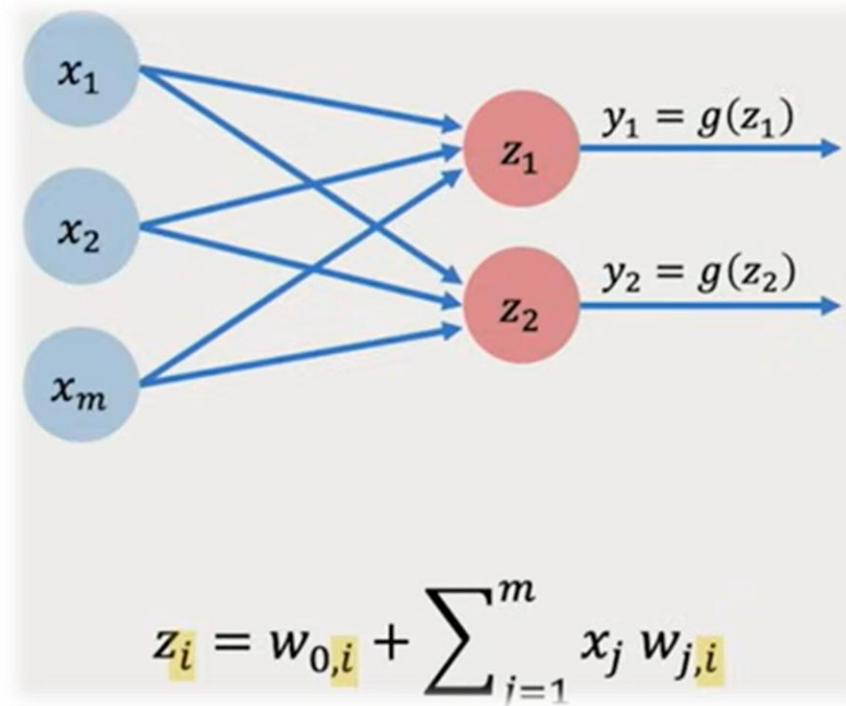
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Multi Output Perceptron

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



Dense layer from scratch

```
class MyDenseLayer(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(MyDenseLayer, self).__init__()

        # Initialize weights and bias
        self.W = nn.Parameter(torch.randn(input_dim,
                                             output_dim, requires_grad=True))
        self.b = nn.Parameter(torch.randn(1, output_dim,
                                             requires_grad=True))

    def forward(self, inputs):
        # Forward propagate the inputs
        z = torch.matmul(inputs, self.W) + self.b

        # Feed through a non-linear activation
        output = torch.sigmoid(z)
        return output
```



Dense layer from scratch

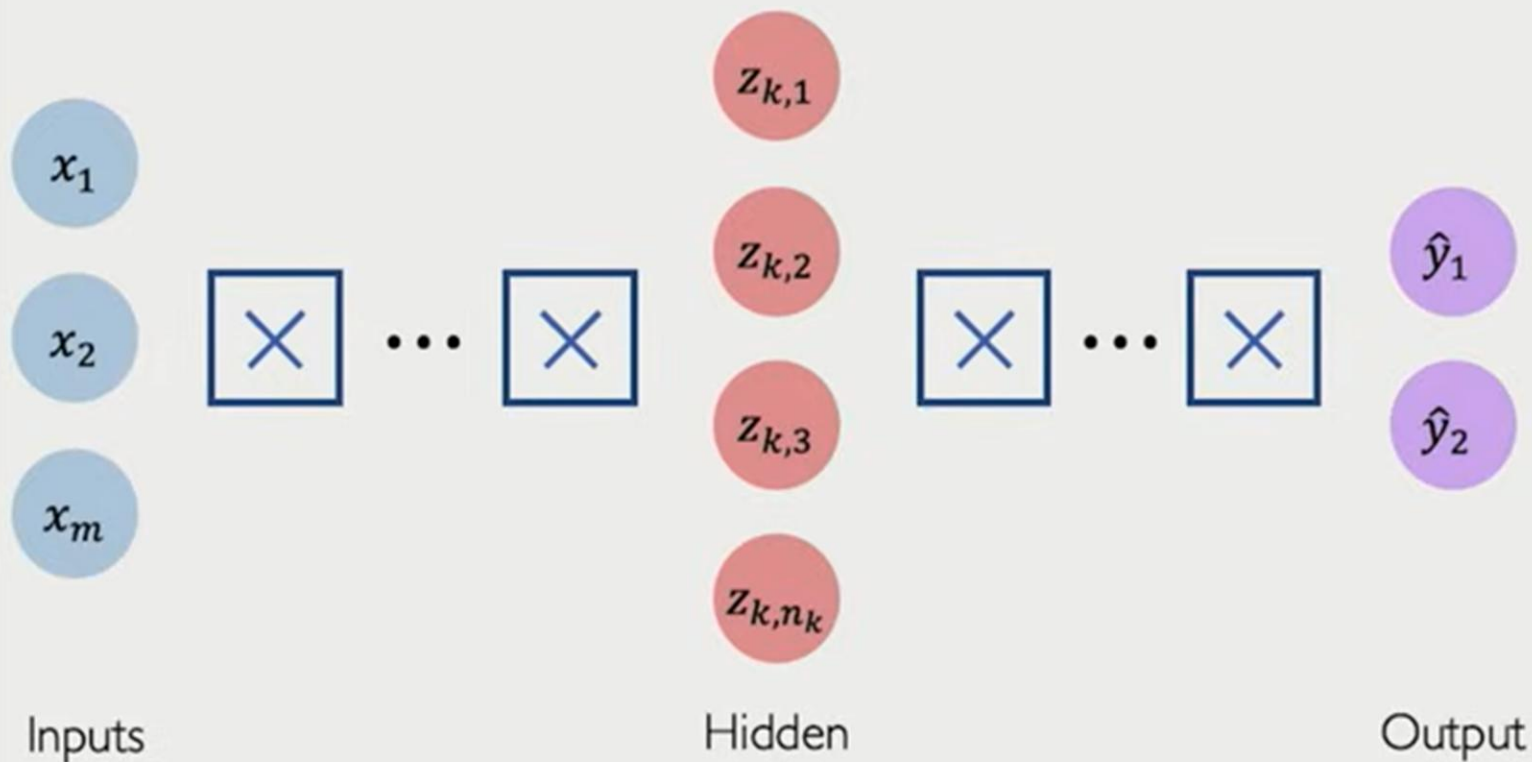
```
import torch.nn as nn  
layer = nn.Linear(in_features=m, out_features=2)
```



```
import tensorflow as tf  
layer = nn.keras.layers.Dense( units=2 )
```



Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$



```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n1),
    tf.keras.layers.Dense(n2),
    ...,
    tf.keras.layers.Dense(2)
])
```



```
from torch import nn

model = nn.Sequential(
    nn.Linear(m, n1),
    nn.ReLU(),
    ...,
    nn.ReLU(),
    nn.Linear(nK, 2)
)
```

Loss Function / Cost Function in Deep Learning

Loss Function:

- A loss function measures the difference between the predicted output and the actual target.
- It helps the neural network adjust weights during training to minimize errors.

```
loss = torch.nn.functional.mse_loss(predicted, y )
```



Cost Function:

- The cost function is the average of the loss function over the entire dataset.
- It provides a single scalar value to optimize the model.

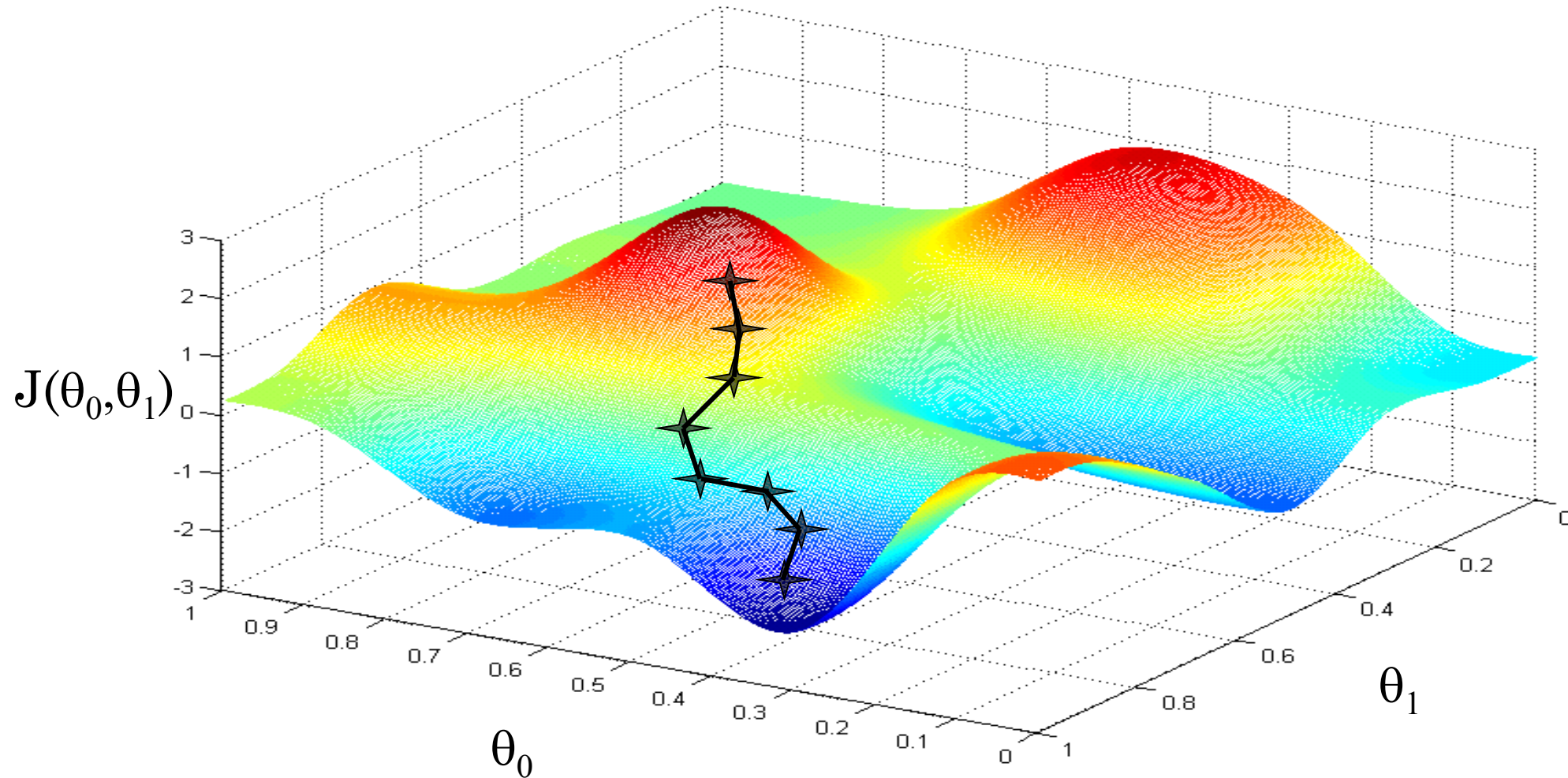
Types of Loss Functions in Neural Networks

Loss Function	Type	Use Case
Mean Squared Error (MSE)	Regression	Penalizes large errors, used for continuous values
Mean Absolute Error (MAE)	Regression	More robust to outliers than MSE
Binary Cross-Entropy	Classification	For binary classification (e.g., Yes/No, 0/1)
Categorical Cross-Entropy	Classification	For multi-class classification (e.g., Image Recognition)
Huber Loss	Regression	Combines MSE and MAE, handles outliers well

Training Neural Networks

Loss Optimization

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



Backpropagation – Intuition

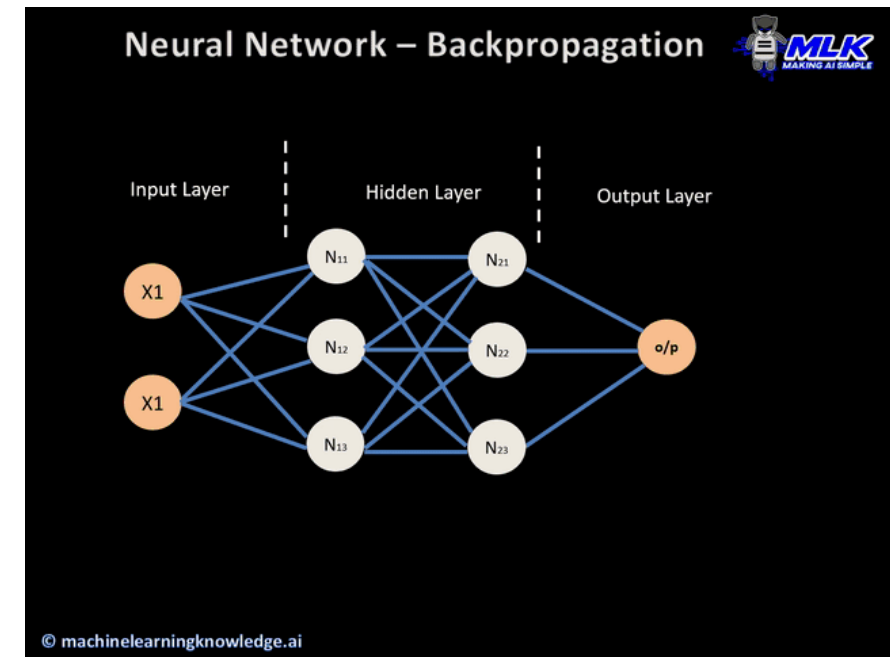
Backpropagation is an algorithm used to minimize the cost function by updating weights based on the error obtained at the output layer.

Key Steps:

- Perform forward pass
- Compute error/loss
- Compute gradients of loss with respect to weights using chain rule.
- Propagate gradients backward layer by layer
- Update weights using gradient descent
- Repeat

Important Notes:

- Backpropagation adjusts weights using partial derivatives.
- It enables efficient training by distributing errors backward.



Optimization Algorithms for Training Neural Networks

Optimization Algorithms:

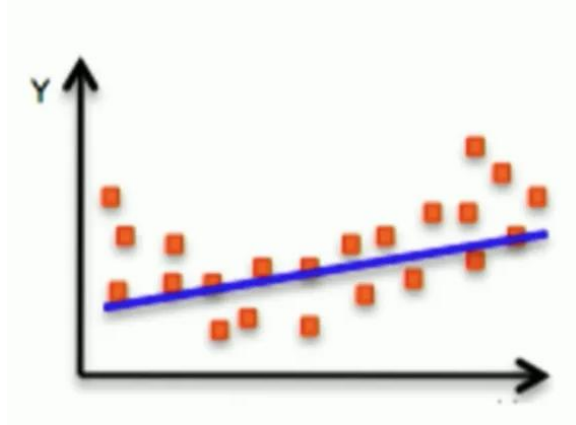
- Optimization algorithms adjust model parameters (weights) to minimize the loss function.
- Efficient optimization ensures faster convergence and better model performance.

Popular Optimization Algorithms

Algorithm	Type	Characteristics
Stochastic Gradient Descent (SGD)	First-order	Updates weights using a single data point, faster but noisy
Momentum-based SGD	First-order	Adds momentum to reduce oscillations and speed up convergence
Adam (Adaptive Moment Estimation)	Adaptive	Combines momentum and adaptive learning rates, widely used
RMSprop	Adaptive	Adjusts learning rates for each parameter, effective for deep networks
Adagrad	Adaptive	Adapts learning rate based on past gradients, suitable for sparse data

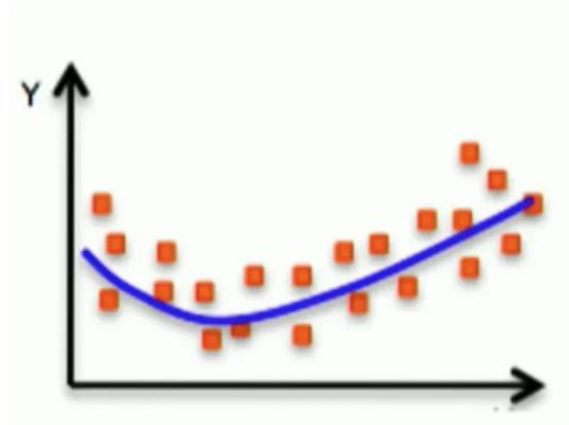
Neural Networks in Practice: **Overfitting**

The Problem of Overfitting

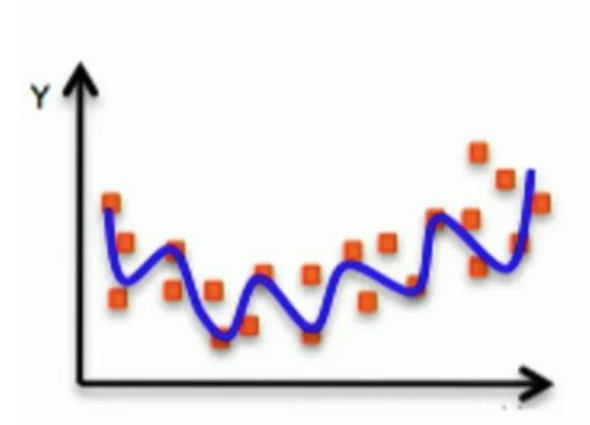


Under fitting

Model does not have capacity to fully learn the data



Ideal fit



Overfitting

Too complex, extra parameters, does not generalize well

Overfitting

- Overfitting occurs when a model learns noise and patterns specific to the training data but fails to generalize to new data.
- It leads to high accuracy on training data but poor performance on test data.

Causes of Overfitting

- Too complex a model with excessive parameters.
- Insufficient training data.
- Lack of regularization techniques.

Techniques to Prevent Overfitting

Method	Description
L1 Regularization (Lasso)	Adds absolute values of weights to the loss function, promotes sparsity by reducing less important weights to zero.
L2 Regularization (Ridge)	Adds squared values of weights to the loss function, prevents large weights and improves generalization.
Dropout	Randomly disables a fraction of neurons during training to reduce reliance on specific features.
Early Stopping	Stops training when validation performance starts to degrade, preventing overfitting.
Data Augmentation	Expands training data by applying transformations (rotation, flipping, scaling) to improve generalization.

Regularization

What is it?

Technique that constraint our optimization problem to discourage complex models

Why do need it?

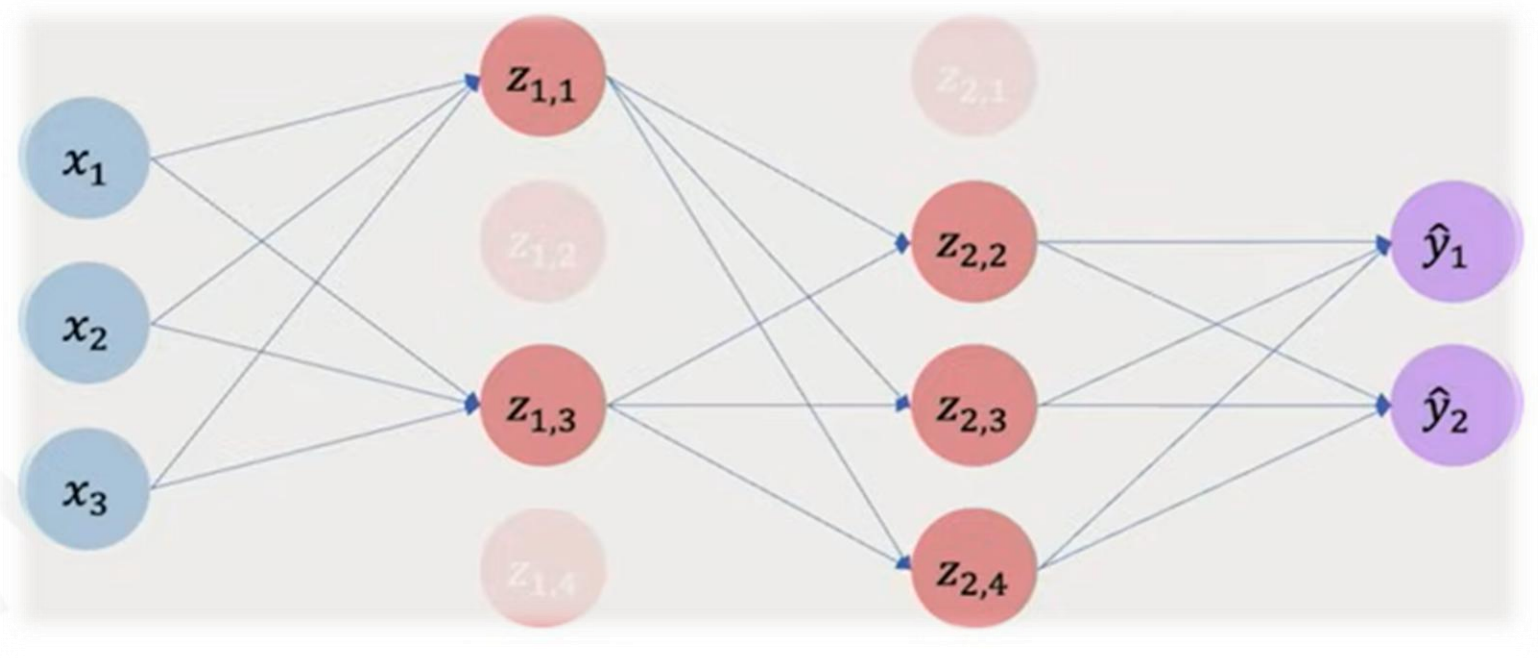
Improve generalization of our model on unseen data

Regularization I: Dropout

During training, randomly set some activations to 0



`torch.nn.Dropout(p=0.5)`



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



Modern Deep Learning Architectures

Modern Deep Learning Architectures

Deep learning has evolved into powerful, specialized architectures:

Architecture	Purpose	Example Use
CNNs (Convolutional Neural Networks)	Extract patterns from images using filters	Image classification, object detection
RNNs (Recurrent Neural Networks)	Handle sequential data with memory	Language modeling, time series
Transformers	Attention-based, parallelized learning for sequences	ChatGPT, translation, summarization
GANs (Generative Adversarial Networks)	Learn to generate realistic data by competition	Deepfakes, image synthesis
Multimodal Models	Process multiple input types (text, image, audio) together	Text-to-image, video Q&A, agents

Core Foundation Review

The Perceptron

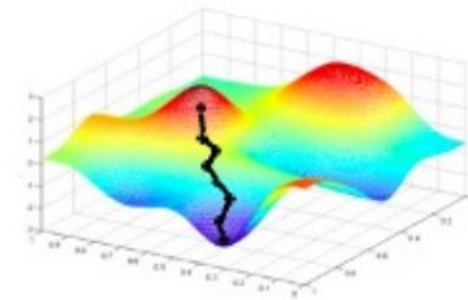
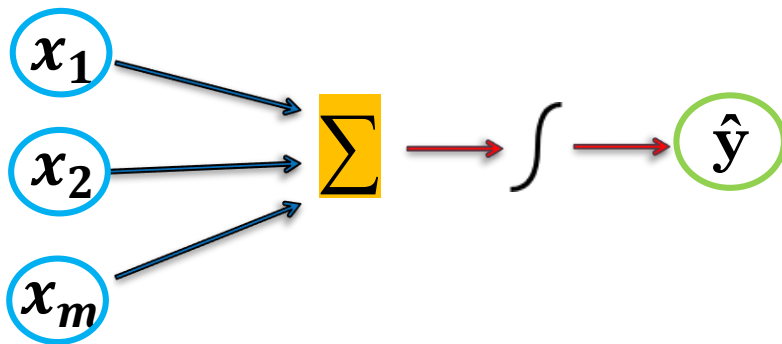
- Structural building blocks
- Nonlinear activation functions

Neural Networks

- Stacking Perceptrons to form neural network
- Optimization through backpropagation

Training in Practice

- Adaptive learning
- Batching
- Regularization



Thank You