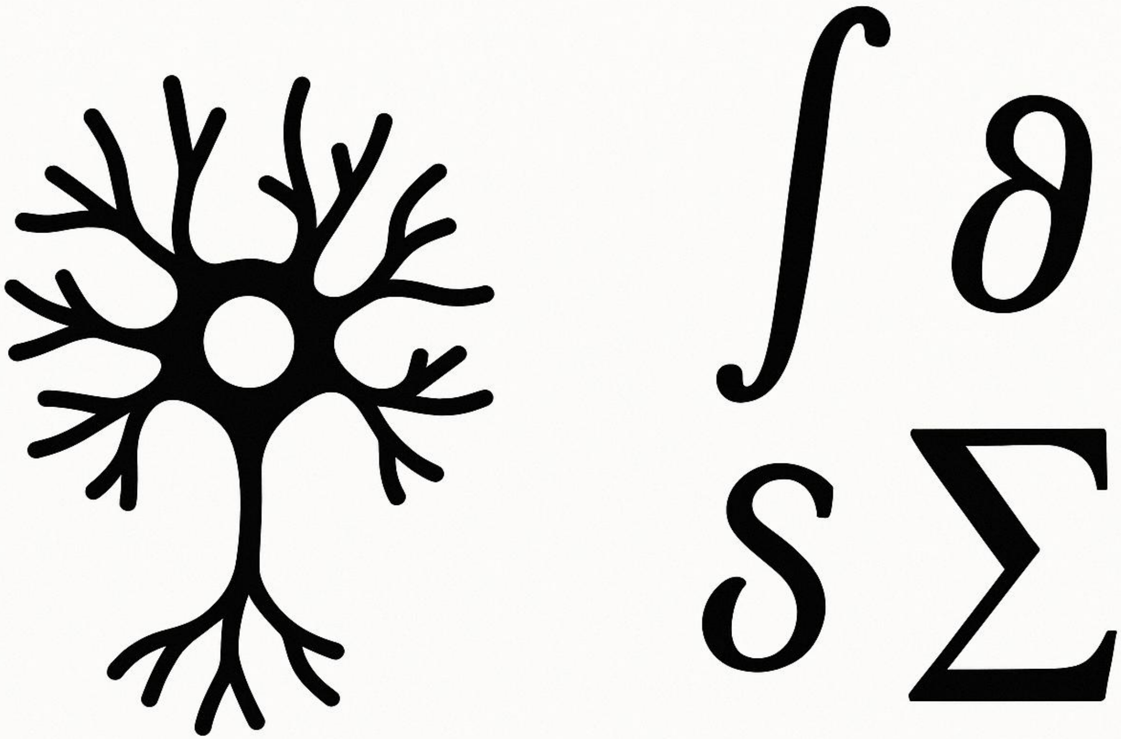
Unveiling Deep Learning: A Mathematical Journey for Educators

How Calculus Powers AI—From Neurons to LLMs



Presenter: Moez Ben-Azzouz

Institution: Sinclair Community College

Workshop Objectives

Key terms and definitions

Why mathematical concepts are crucial to AI and deep learning

How calculus concepts directly apply to neural networks

Step-by-step construction of a neural network

Mathematical foundations of training and prediction

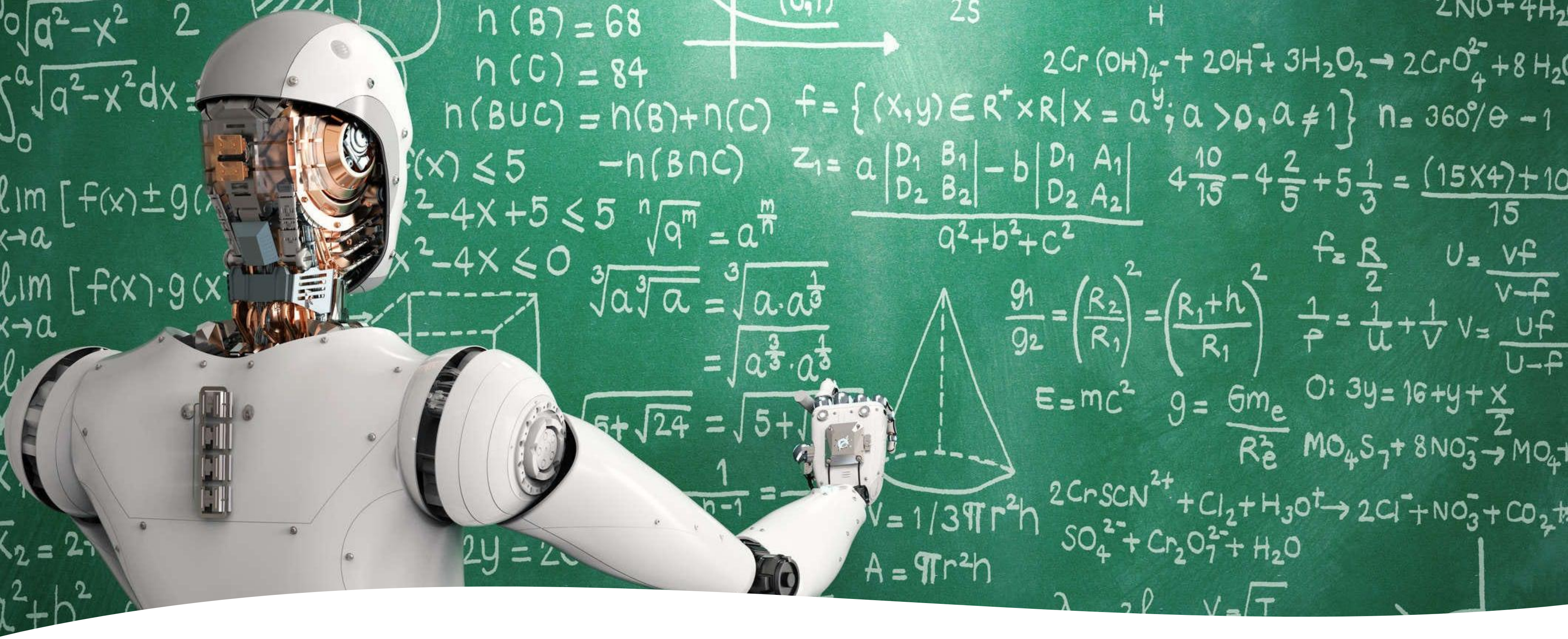
Practical applications and teaching strategies

Section 0

Key terms and definitions

What Is AI?

- **Artificial Intelligence (AI):** the simulation of human intelligence in machines that are programmed to think, learn, and solve problems like humans.
- **Goal:** create systems that can mimic human intelligence and behavior to some extent, and in some cases, surpass human capabilities.
- Applied to healthcare, finance, transportation, entertainment, etc.

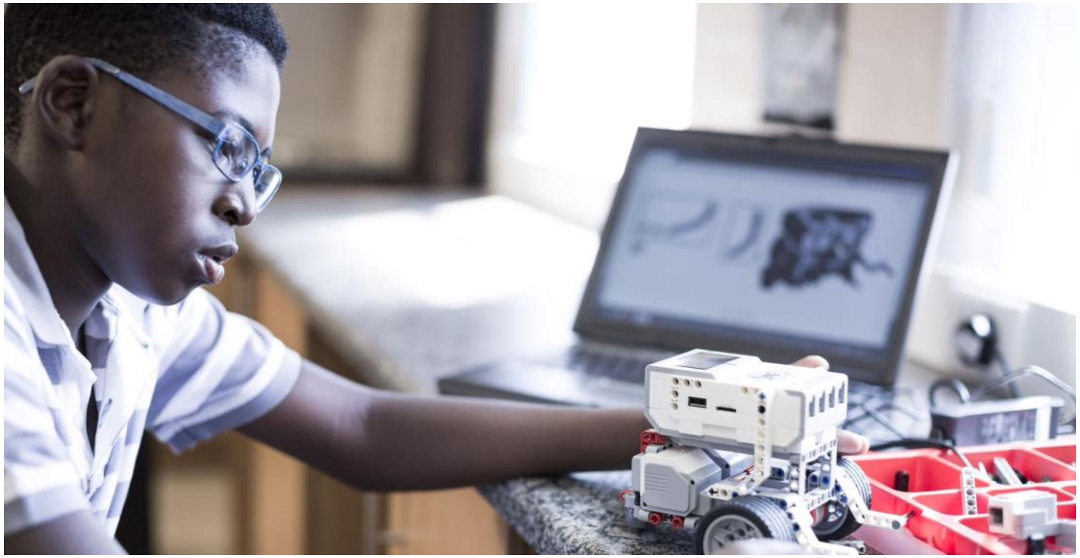


AI Concerns

- Ethical implications
- Privacy
- Bias
- Job displacement
- Control over autonomous systems
- Difficulty to analyze and interpret
- Safety

AI Tasks

- Machine Learning (ML)
- Deep Learning
- Natural Language Processing (NLP)
- Computer Vision
- Robotics
- Expert Systems
- Reinforcement Learning



What Is Machine Learning?

- **Machine Learning (ML):** the science and art of programming computers so they can learn from data

- The field of study that gives computers the ability to learn without being explicitly programmed.

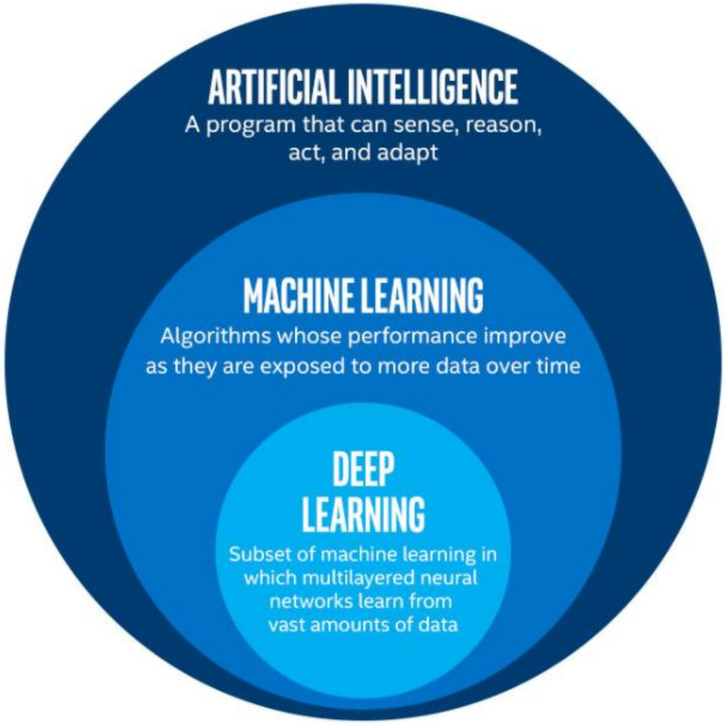
  \- Arthur Samuel, 1959

- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

  \- Tom Mitchell, 1997

AI, ML, DL, NLP, and GenAI

- **ML is a subset of AI** that enables machines to learn from data and improve their performance over time without being explicitly programmed.
- **Deep Learning:** a specific type of ML that uses artificial neural networks to model and process complex patterns in data.
- **Natural Language Processing (NLP):** enables computers to understand, interpret, and generate human language.
- **Generative AI (GenAI):** a type of AI that can create new content and ideas in various forms based on existing data it has been trained on.



AI Tasks

Computer Vision: techniques that enable machines to interpret and understand visual information: facial recognition, object detection, autonomous vehicles, etc.

Robotics: AI is used to enhance the capabilities of robots, enabling them to perform tasks autonomously and intelligently.

Expert Systems: AI systems that emulate the decision-making abilities of a human expert in a particular domain.

GenAI: text generation, image generation, video generation, music generation.

YOLO Multi-Object Detection and Classification.
Photo by Ilija Mihajlovic, Towards Data Science

Machine Learning vs. Traditional Programming

Machine Learning

- Pros
  - Complex problems
  - Scale
  - Adaptable
  - Personalization
  - Improves over time
- Cons
  - Slower to build
  - Harder to explain/interpret
  - Harder to debug

Traditional Programming

- Pros
  - Quicker to build
  - Easier to explain
  - Easier to debug
  - Easier to maintain
  - More consistent/stable
- Cons
  - Does not scale
  - Does not adapt to changes
  - Does not work for complex tasks

Types of Machine Learning

Supervised Learning: ML algorithms trained with human supervision. E.g., classification, regression

Unsupervised Learning: the training data is unlabeled. The system tries to learn without a teacher. Clustering, anomaly detection, visualization and dimensionality reduction, etc.

Reinforcement Learning: agents learn by interacting with an environment and receiving feedback in the form of rewards or penalties. The agent's goal is to maximize reward over time.

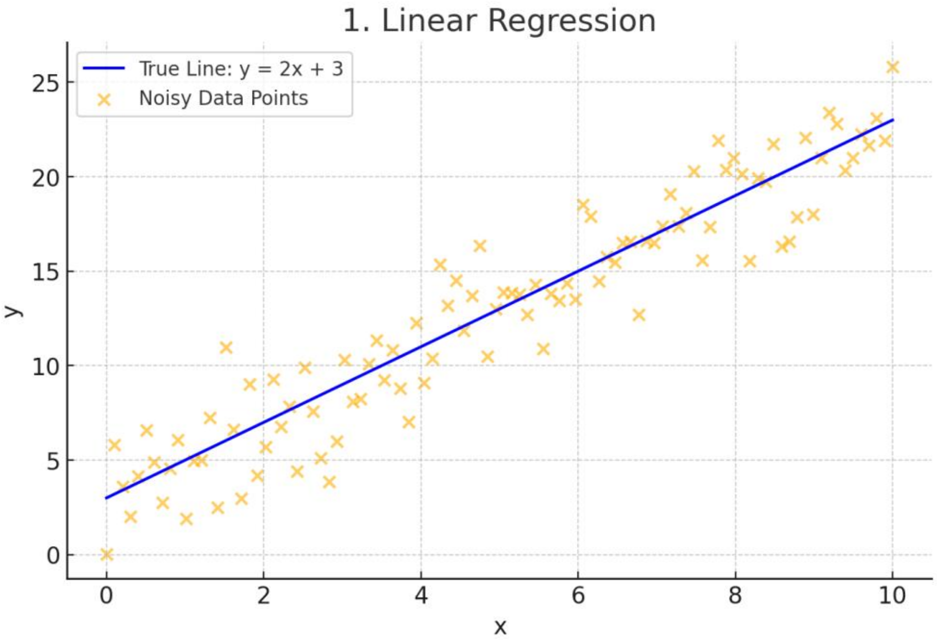
Section 1

Mathematical Foundations of Deep Learning

Linear Regression – The Starting Point

Equation: $y = mx + b$

- **Goal**: Predict y given x.
- **Concepts**:
  - Parameters: slope mmm, intercept b
  - Loss function: Mean Squared Error (MSE)
  - Optimization: Gradient Descent

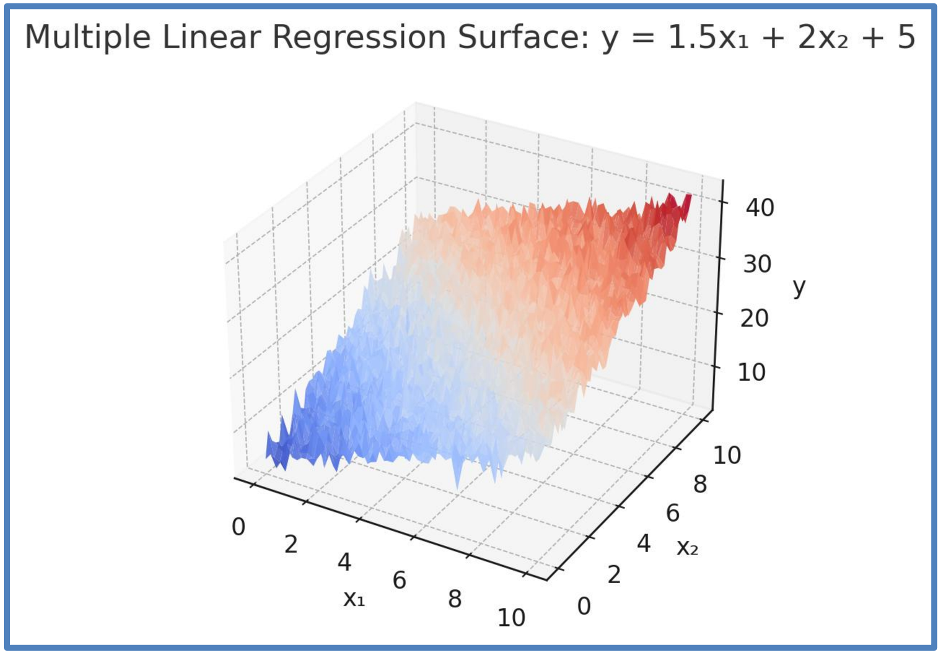


Multiple Linear Regression

Equation: $y = w_1x_1 + w_2x_2 + \ \dots\ + w_nx_n + b$

Now a vector space: $\vec{y} = \vec{w}^T\vec{x} + b$

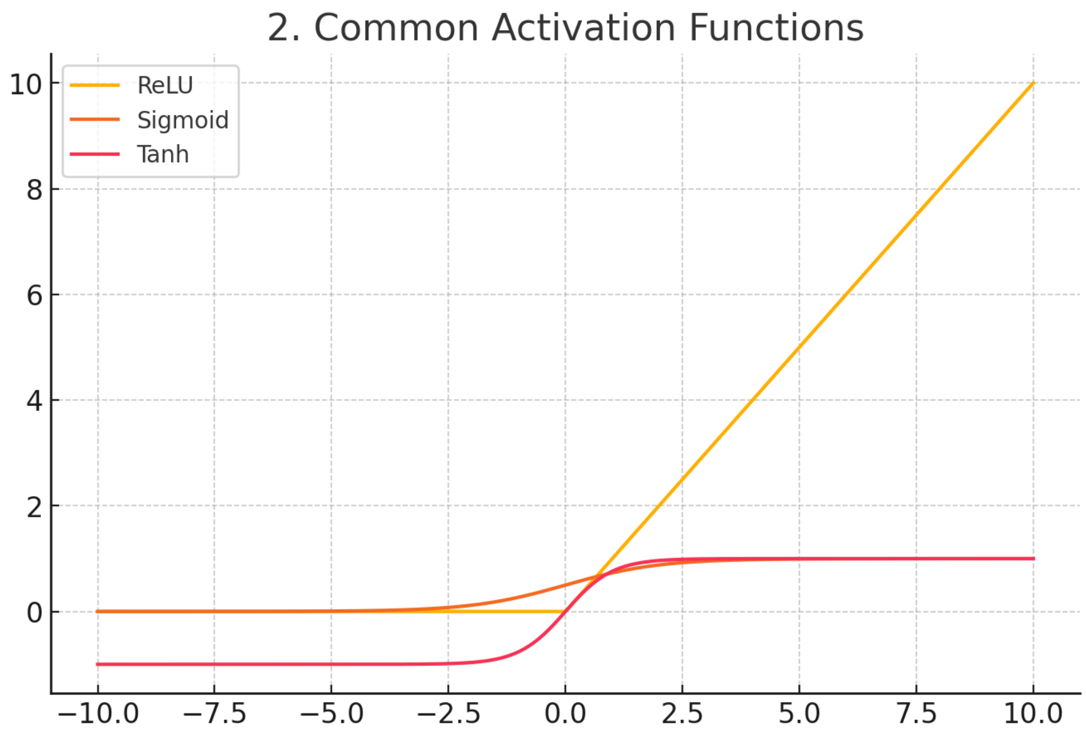
Matrix representation: $Y = XW + b$



Introducing Non-Linearity

- **Problem**: Linear models cannot capture non-linear patterns
- **Solution**: Introduce non-linear transformations (activation functions)
- **Now the model becomes:**

$$y = \sigma(w_1 x_1 + w_2 x_2 + \ \dots \ + b)$$

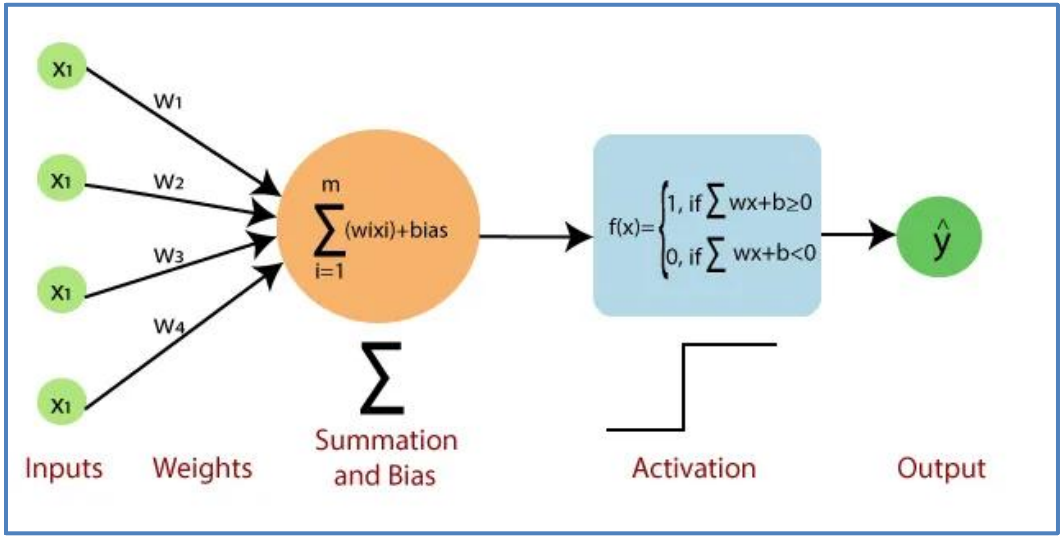


The Perceptron and Neural Networks

- **The Basic Unit**: A neuron

  $\text{output} = \sigma(w \cdot x + b)$

- **Stacking Neurons → Layers**
  - Input layer
  - Hidden Layer(s)
  - Output Layer
- **Deep Neural Network (DNN)**
  - NN with multiple hidden layers
  - Greater representational power



The perceptron is a neural network unit (an artificial neuron) introduced by Frank Rosenblatt in 1957.

Types of Neural Networks

Neural Networks	Applications
Convolutional Neural Networks (CNNs)	Image Classification
Recurrent Neural Networks (RNNs)	Time series, sequences, language modeling
Transformers and LLMs	Q&A, translation, chat, summarize documents, etc.
Generative Pre-Trained Transformers (GPTs)	Can produce new text, not just classify or summarize.

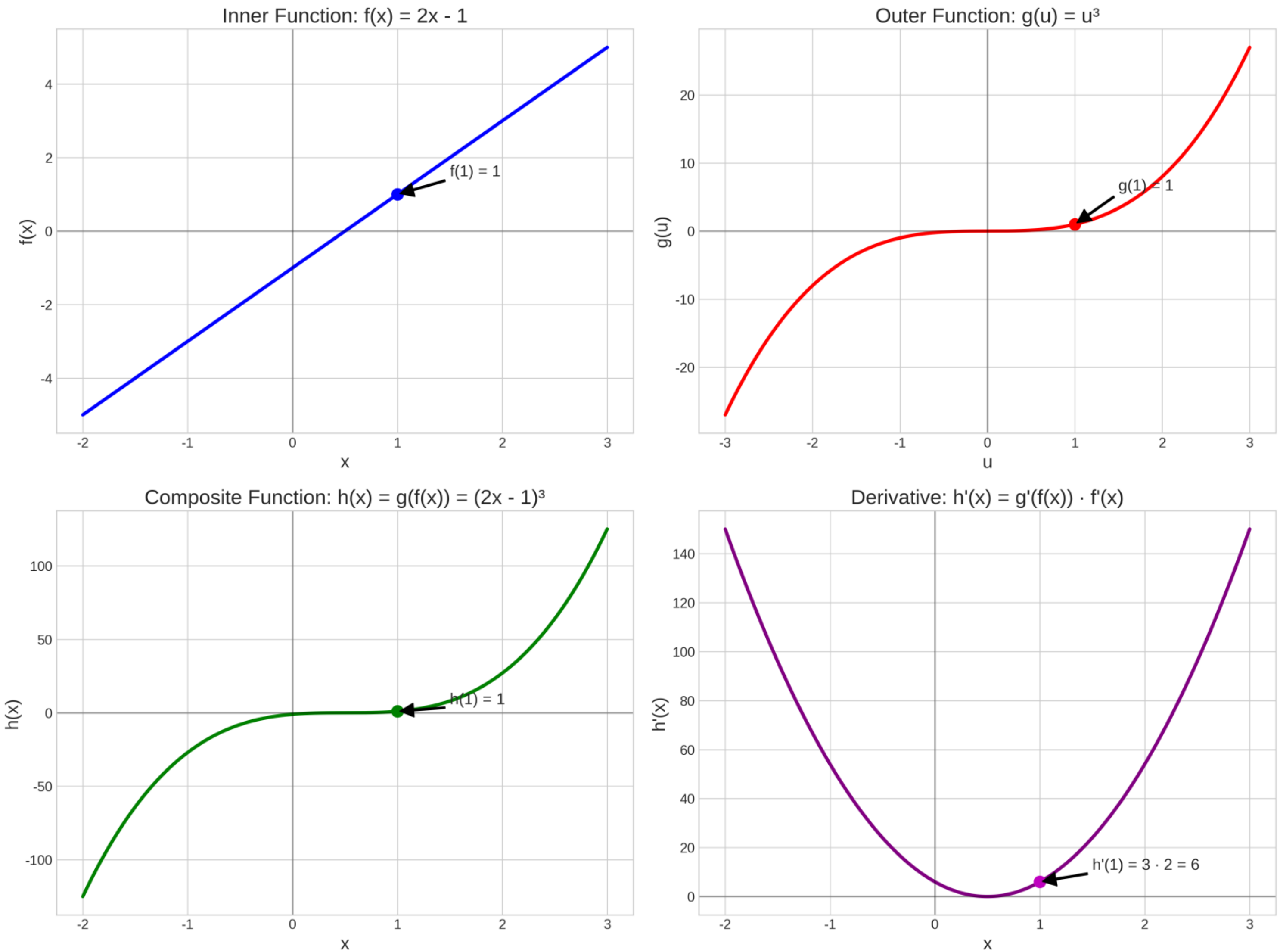
Key Mathematical Concepts:

- Composition of functions
- Optimization
- Calculus (partial differentiation, gradients, chain rule)
- Linear algebra
- Probability (language modeling, output distributions)

Key Mathematical Concepts in Deep Learning

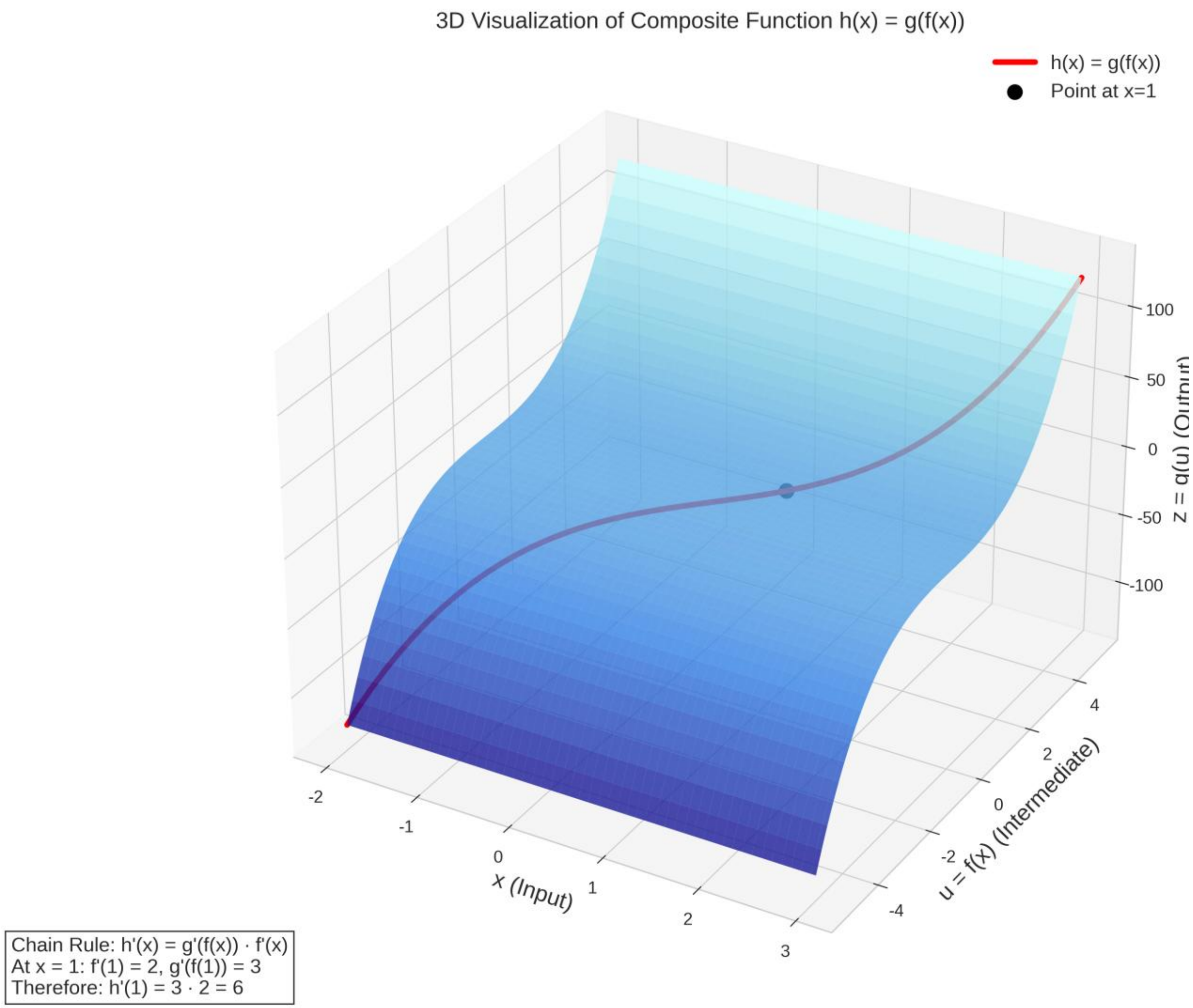
- Chain Rule: Core of backpropagation algorithm
- Partial Differentiation: Handling multiple variables
- Gradients: Direction of steepest ascent/descent
- Vectors and Matrices: Efficient computation
- Optimization Theory: Finding minima of functions

The Chain Rule in Neural Networks



The chain rule allows us to compute derivatives through compositions of functions

Chain Rule: 3D Visualization

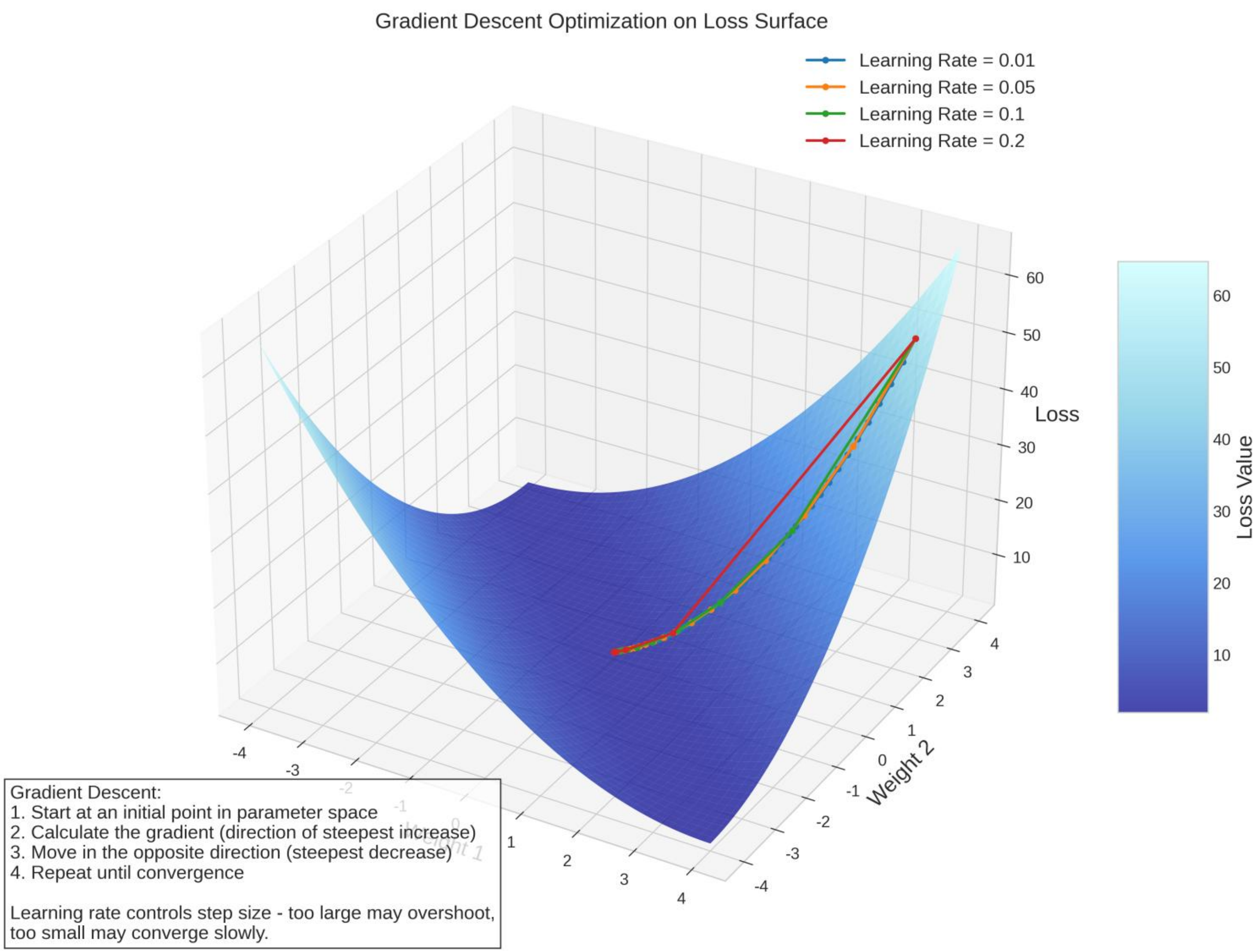


Visualizing the chain rule in three dimensions

Partial Derivatives and Gradients

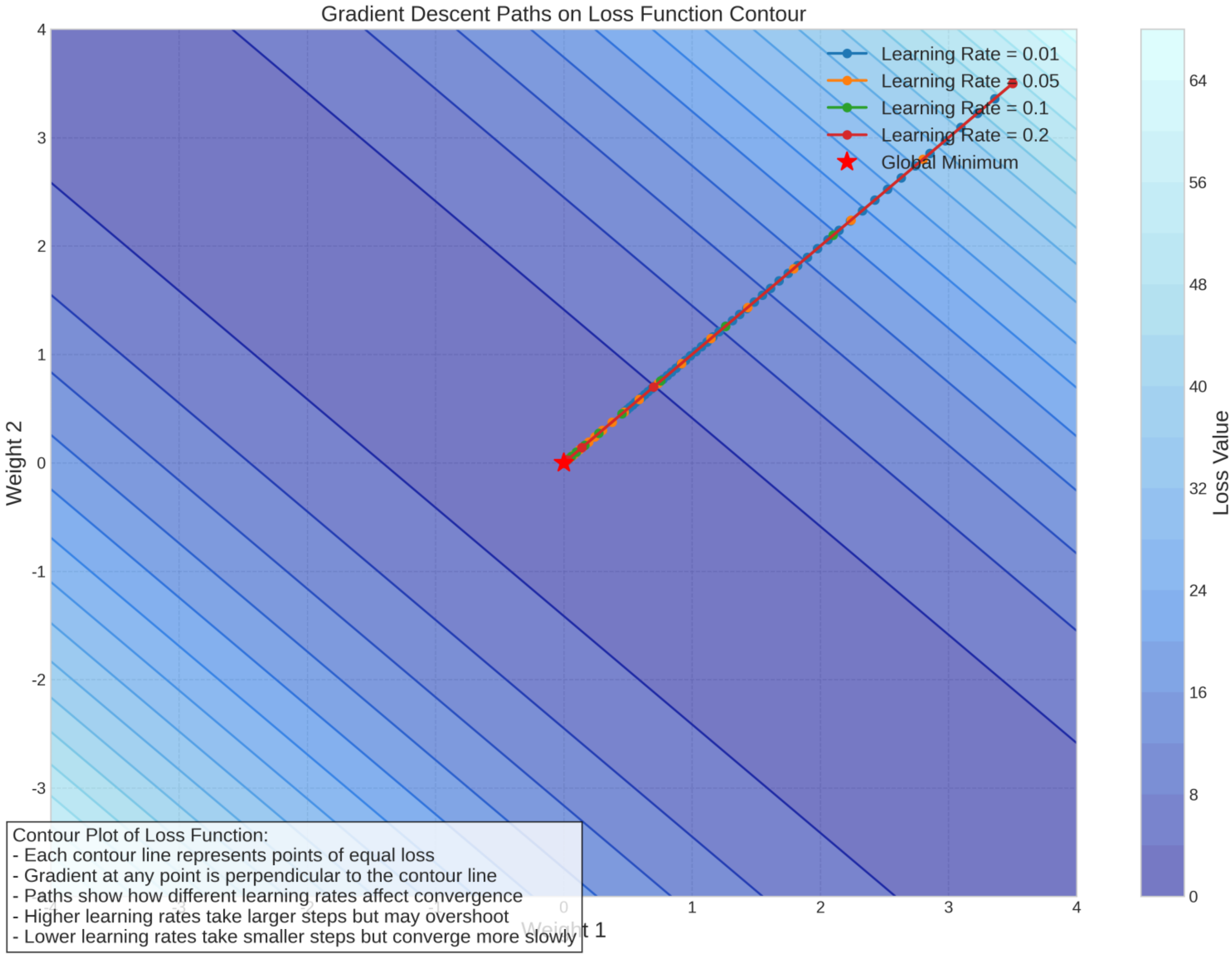
- Neural networks have many parameters (weights and biases)
- Need to compute how each parameter affects the output
- Partial derivatives measure rate of change with respect to one variable
- Gradient vector contains all partial derivatives
- Gradient points in direction of steepest increase

Gradient Descent Optimization



Gradient descent finds the minimum of the loss function by iteratively moving in the direction of steepest descent

Gradient Descent: Contour View

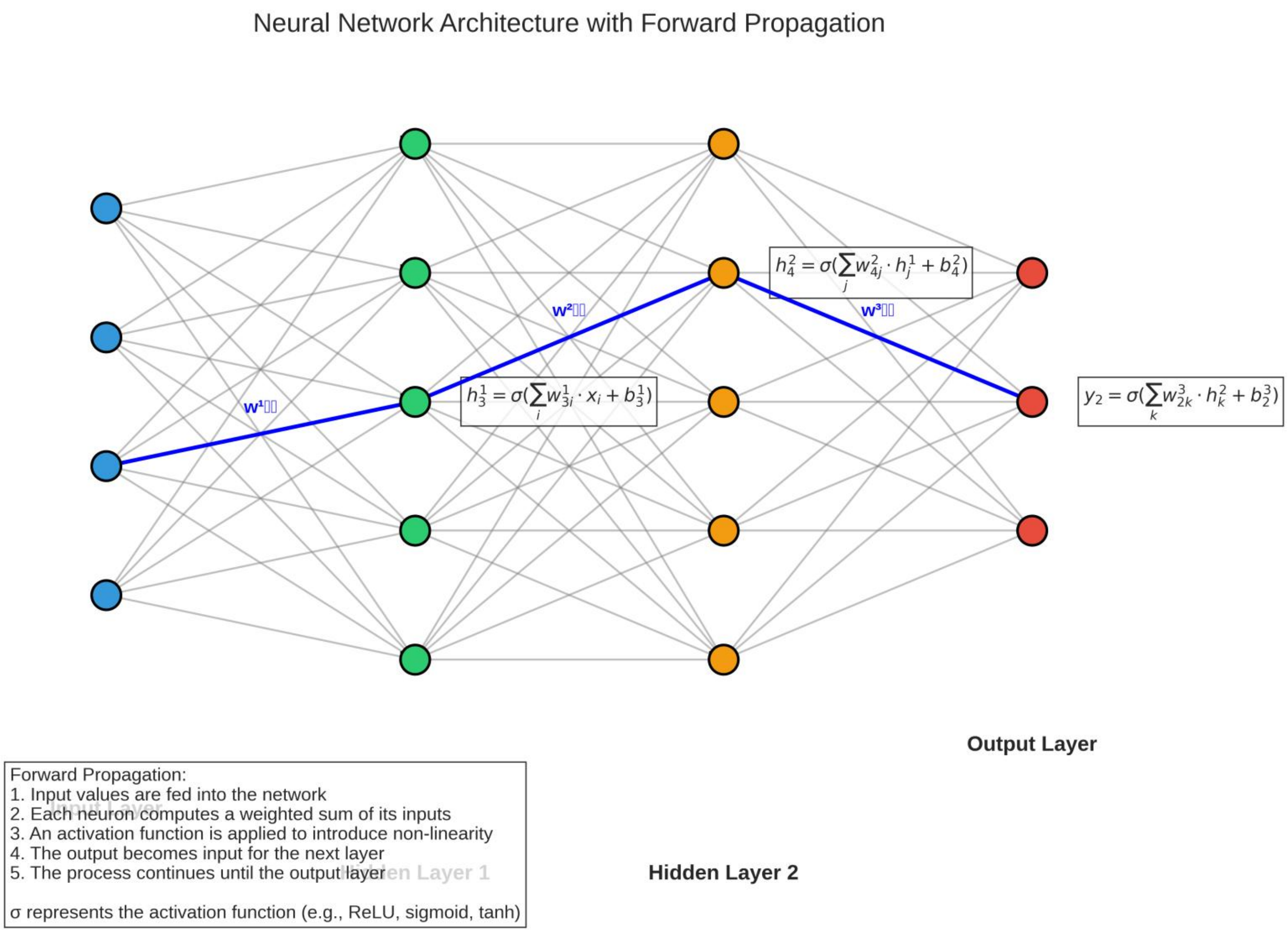


Contour plot showing gradient descent paths with different learning rates

Section 2

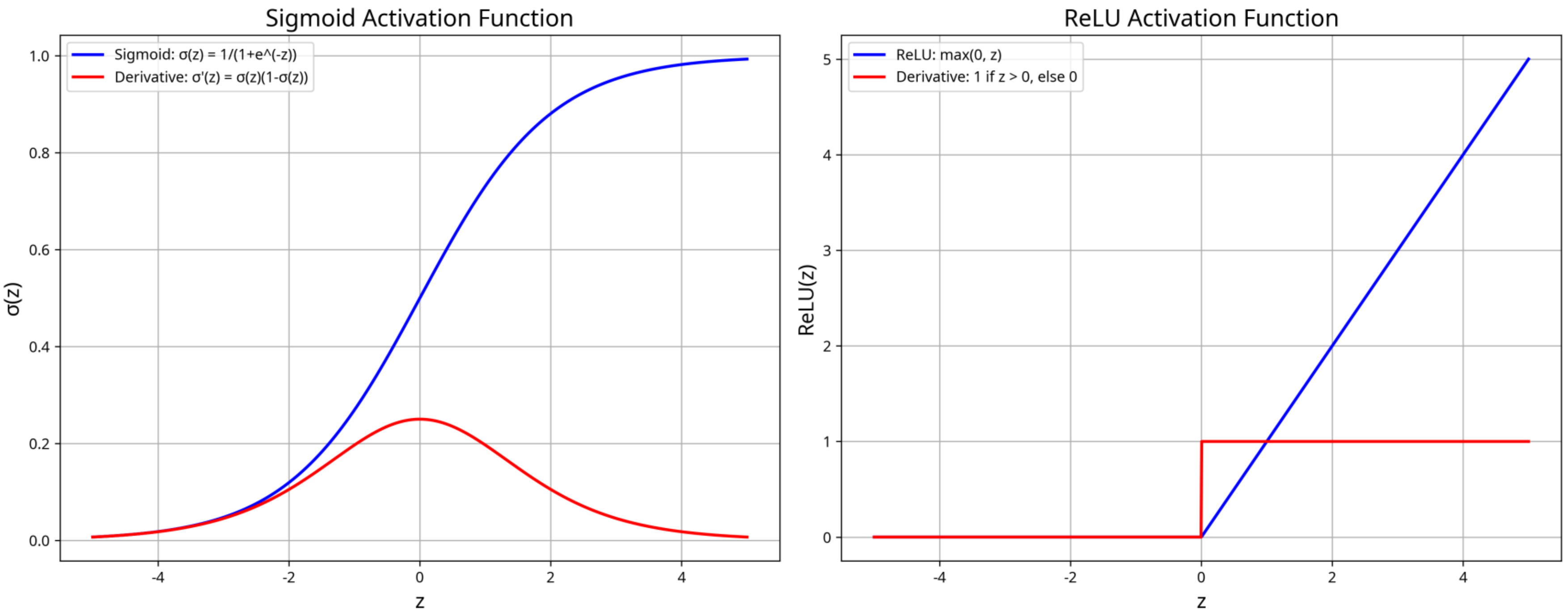
Neural Network Architecture and Forward Propagation

Neural Network Architecture



A neural network consists of layers of interconnected neurons

Activation Functions



Activation functions introduce non-linearity into the network

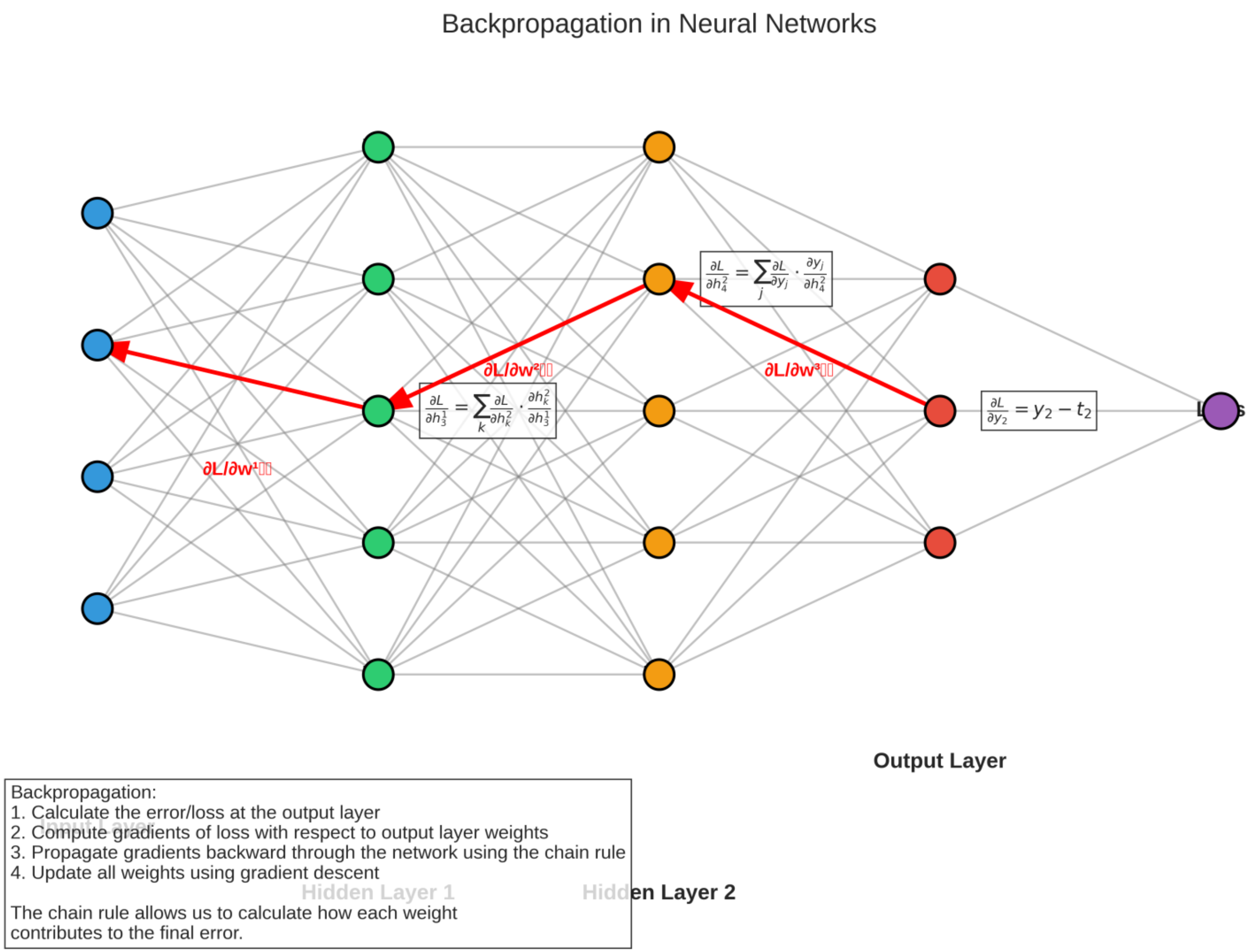
Section 3

Training Neural Networks: Backpropagation and Gradient Descent

Loss Functions

- Measure how far predictions are from actual values
- Mean Squared Error (MSE): $L = 1/n\ \Sigma(y\_pred - y\_true)^2$
- Binary Cross-Entropy: $L = -1/n\ \Sigma[y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})]$
- Goal: Find weights that minimize the loss function
- Calculus helps us find this minimum

Backpropagation

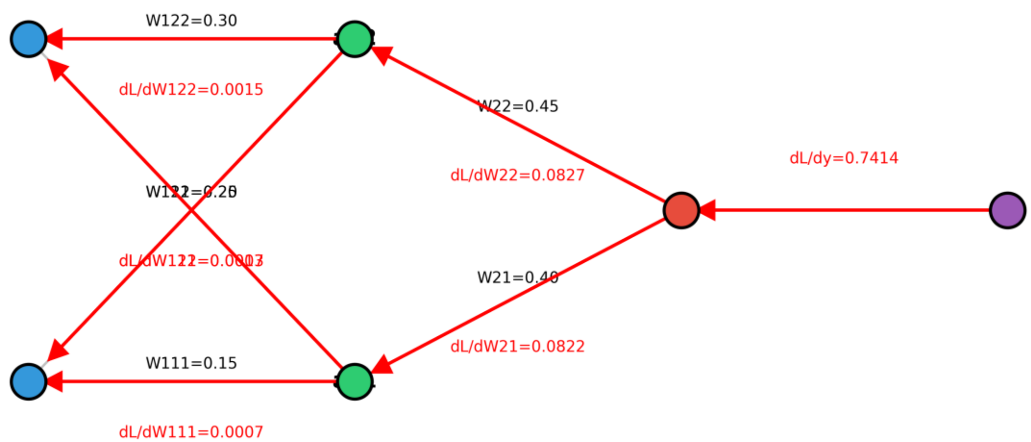


Backpropagation uses the chain rule to efficiently compute gradients

Gradient Calculation

Gradient Calculation in Neural Networks

Forward Pass:
z1 = W1·x + b1 = [0.3775, 0.3925]
a1 = sigmoid(z1) = [0.5933, 0.5969]
z2 = W2·a1 + b2 = [1.1059]
y_pred = sigmoid(z2) = [0.7514]
Loss = 0.5·(y_pred - y_true)² = 0.2748



Backward Pass (Chain Rule):
dL/dy_pred = (y_pred - y_true) = [0.7414]
dy_pred/dz2 = sigmoid'(z2) = [0.1868]
dL/dz2 = dL/dy_pred · dy_pred/dz2 = [0.1385]
dL/dW2 = a1 · dL/dz2 = [0.0822, 0.0827]ᵀ
dL/db2 = dL/dz2 = [0.1385]
dL/da1 = dL/dz2 · W2ᵀ = [0.0554, 0.0623]
da1/dz1 = sigmoid'(z1) = [0.2413, 0.2406]
dL/dz1 = dL/da1 · da1/dz1 = [0.0134, 0.0150]
dL/dW1 = x · dL/dz1ᵀ (outer product)

Detailed calculation of gradients using the chain rule

Backpropagation Steps

Step 1: Calculate Output Error

1. Compute the error at the output layer:
δ = y_pred - y_true

2. For Mean Squared Error (MSE) loss:
L = 0.5 * (y_pred - y_true)²
δ = ∂L/∂y_pred = (y_pred - y_true)

3. For Binary Cross-Entropy loss:
L = -[y_true*log(y_pred) + (1-y_true)*log(1-y_pred)]
δ = ∂L/∂y_pred = -y_true/y_pred + (1-y_true)/(1-y_pred)

Step 2: Output Layer Gradients

1. Apply the chain rule to get gradients for output layer:

∂L/∂w = ∂L/∂y_pred * ∂y_pred/∂z * ∂z/∂w

2. For sigmoid activation:
∂y_pred/∂z = sigmoid(z) * (1 - sigmoid(z))

3. The gradient for each weight:
∂L/∂w_ji = δ_j * sigmoid'(z_j) * a_i
where a_i is the activation from the previous layer

Step 3: Propagate Error to Hidden Layer

1. Propagate the error backward to hidden layer:

δ_hidden = (W_output^T * δ_output) ⊙ f'(z_hidden)

2. Where:
- W_output^T is the transpose of the output layer weights
- δ_output is the error at the output layer
- ⊙ represents element-wise multiplication
- f'(z_hidden) is the derivative of activation function

3. This is the chain rule in action, computing:
∂L/∂a_hidden = ∂L/∂z_output * ∂z_output/∂a_hidden = W_output^T * δ_output

Step 4: Update Weights Using Gradient Descent

1. Update all weights using gradient descent:

w_new = w_old - learning_rate * ∂L/∂w

2. The learning rate controls the step size:
- Too large: may overshoot the minimum
- Too small: slow convergence

3. Variants of gradient descent:
- Batch GD: update using all training examples
- Stochastic GD: update using one example at a time
- Mini-batch GD: update using a small batch of examples

Step-by-step process of backpropagation

Section 4

Practical Implementation and Code Examples

Activity (Handout)

Problem Setup

Task: Binary classification (e.g., "Pass (1) or Fail (0) based on study hours").

Network Architecture:

- **Input layer:** one neuron (study hours x)
- **Output layer:** one neuron (prediction $\hat{y} = \sigma(z)$, where $z = wx + b$)
- **Activation (Sigmoid):** $\sigma(z) = \frac{1}{1+e^{-z}}$
- **Loss (Mean Squared Error):** $L = \frac{1}{2}(y - \hat{y})^2$

Parameters (emerge from data):

- Initial weight $w = 0.6$
- bias $b = -0.3$

Hyperparameter (control training process):

- Learning rate $\alpha = 0.1$

Example Data Point

- **Input** $x = 2$, True label $y = 1$ (Pass)

Forward Propagation (Python)

```
# Forward propagation in a neural network
def forward(X, W1, b1, W2, b2):
    # First layer: input to hidden
    z1 = np.dot(X, W1) + b1  # Matrix multiplication
    a1 = sigmoid(z1)         # Apply activation function

    # Second layer: hidden to output
    z2 = np.dot(a1, W2) + b2 # Matrix multiplication
    y_pred = sigmoid(z2)     # Apply activation function

    return y_pred, (z1, a1, z2)
```

Backpropagation (Python)

```
# Backward propagation to compute gradients
def backward(X, y, y_pred, cache, W1, W2):
    z1, a1, z2 = cache
    m = X.shape[0]

    # Output layer gradient
    dz2 = y_pred - y                        # Derivative of loss w.r.t. z2
    dW2 = (1/m) * np.dot(a1.T, dz2)         # Gradient for W2
    db2 = (1/m) * np.sum(dz2, axis=0)       # Gradient for b2

    # Hidden layer gradient (chain rule)
    da1 = np.dot(dz2, W2.T)                   # dL/da1 = dL/dz2 · dz2/da1
    dz1 = da1 * sigmoid_derivative(z1)        # dL/dz1 = dL/da1 · da1/dz1
    dW1 = (1/m) * np.dot(X.T, dz1)            # Gradient for W1
    db1 = (1/m) * np.sum(dz1, axis=0)         # Gradient for b1

    return dW1, db1, dW2, db2
```

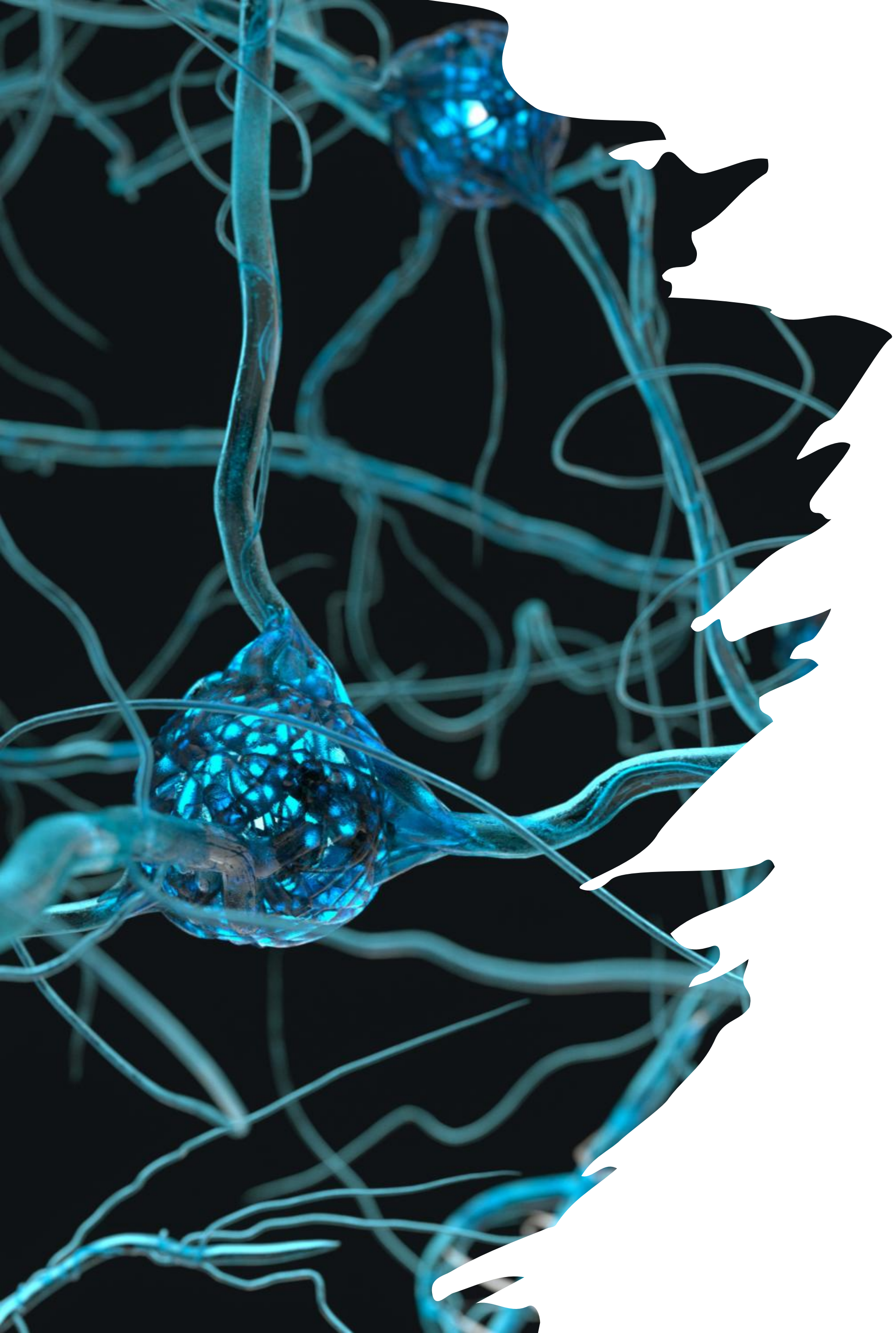
Gradient Descent Update (Python)

```
# Update parameters using gradient descent
def update_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2,
   learning_rate):
    W1 = W1 - learning_rate * dW1 # Update weights for layer 1
    b1 = b1 - learning_rate * db1 # Update biases for layer 1
    W2 = W2 - learning_rate * dW2 # Update weights for layer 2
    b2 = b2 - learning_rate * db2 # Update biases for layer 2

    return W1, b1, W2, b2
```

Section 5

Applications to Large Language Models (LLMs)



From Neural Networks to LLMs

- LLMs are extremely large neural networks
- Same mathematical principles apply at scale
- Transformer architecture uses attention mechanisms
- *Self-attention* involves matrix multiplications and *softmax*
- Training involves the same backpropagation and gradient descent

Mathematical Challenges in LLMs

- Computational efficiency: Optimizing matrix operations
- Numerical stability: Preventing exploding/vanishing gradients
- Optimization in high-dimensional spaces
- Regularization to prevent overfitting
- Probabilistic modeling of language

Section 6

Teaching Strategies and Classroom Applications

Incorporating AI Applications in Calculus Courses

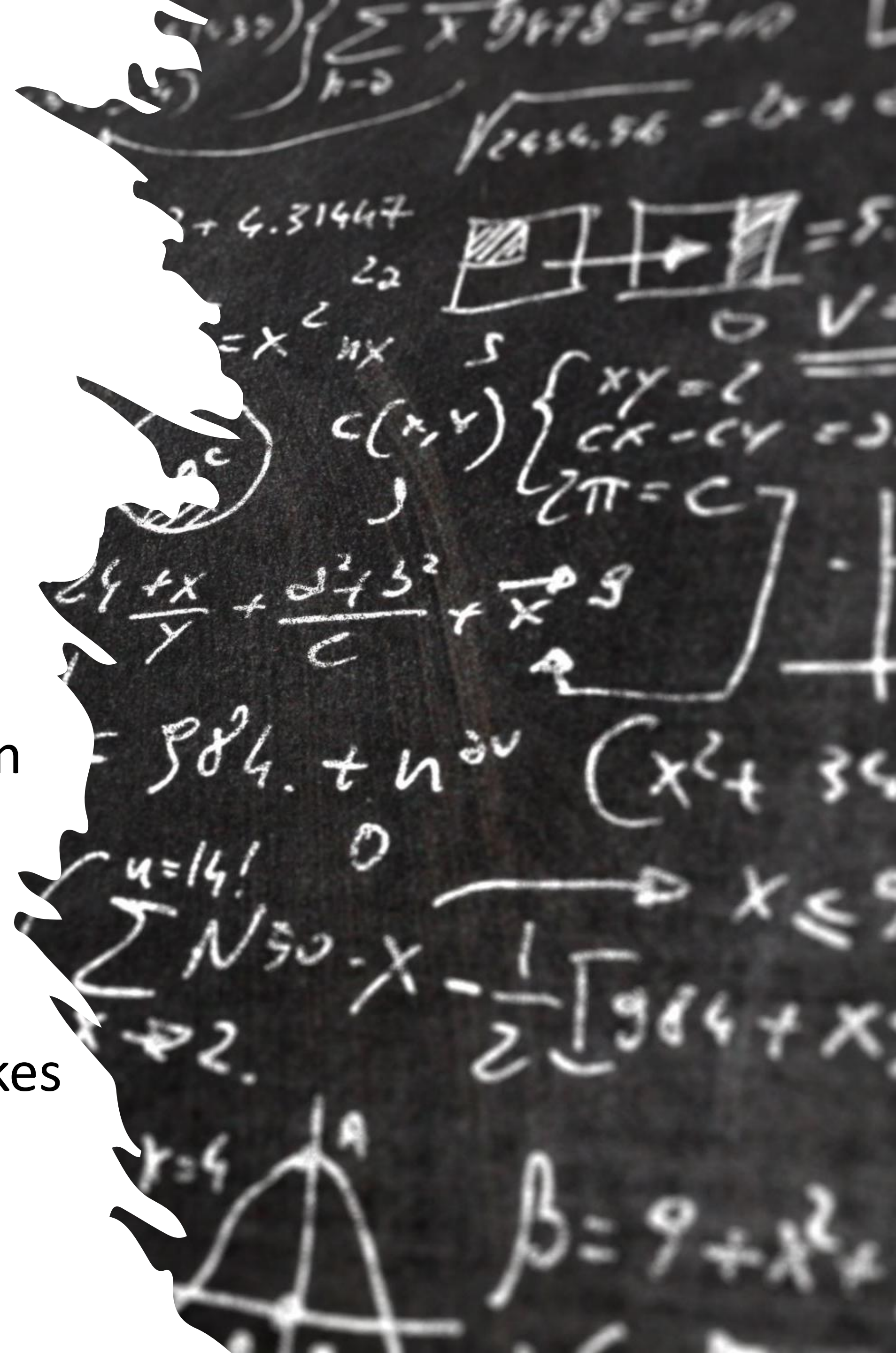
- Connect chain rule to backpropagation
- Use gradient descent to motivate partial derivatives
- Demonstrate real-world applications of optimization
- Show how linear algebra and calculus work together
- Use visualization tools to build intuition

Project Ideas for Students

- Implement a simple neural network from scratch
- Visualize gradient descent on different loss functions
- Explore the effects of different activation functions
- Analyze the impact of learning rate on convergence
- Apply neural networks to real-world datasets

Key Takeaways

- Calculus is the mathematical foundation of deep learning
- Chain rule enables backpropagation algorithm
- Gradients guide the optimization process
- Neural networks apply these concepts at scale
- Teaching these connections makes mathematics more relevant



Resources for Further Learning

- 'Deep Learning' by Goodfellow, Bengio, and Courville
- 3Blue1Brown's Neural Network series on YouTube
- Stanford's CS231n: Convolutional Neural Networks
- 'Neural Networks and Deep Learning' by Michael Nielsen
- TensorFlow and PyTorch tutorials

Questions and Discussion

- WHAT ASPECTS OF NEURAL NETWORKS INTEREST YOU MOST?

- HOW MIGHT YOU INCORPORATE THESE CONCEPTS IN YOUR TEACHING?

- WHAT CHALLENGES DO YOU ANTICIPATE?

- WHAT ADDITIONAL RESOURCES WOULD BE HELPFUL?

THANK YOU

- Moez Ben-Azzouz
- Mathematics Department, Sinclair Community College
- moez.ben-azzouz@sinclair.edu
- https://s013mmb.github.io/Math4DL/index.html