JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

685.621 Algorithms for Data Science

Neural Networks: Mathematical Foundations

Forward Propagation Mechanics

Neuron Equation

$$\hat{y} = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Single Node Forward Propagation

$$a^{[l+1]} = f(W^{[l]} a^{[l]} + b^{[l]})$$

Matrix Form 1-Layer Forward Propagation

$$Z = WX + b$$

$$A = \sigma(Z)$$

Generalize for Multiple Layers

$$Z^{[l+1]} = W^{[l]} A^{[l]} + b^{[l]}$$

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

Loss Functions

Regression Problems

$$MSE = \frac{1}{m}\sum(y_i - \hat{y}_i)^2$$

Classification Problems

$$CrossEntropy = \sum y \log(\hat{y})$$

- **Loss functions** quantify the difference between the network's predictions and the actual target values
- **Mean Squared Error** penalizes larger errors more heavily because of the square term.
- **Cross Entropy** penalizes incorrect class probabilities and rewards confident correct predictions (multi-class classification)
- During training, our goal is to **minimize** this loss, step by step, until our model **generalizes well** to new, unseen data

Backpropagation Algorithm

- **Step 1**: Compute output layer error
- **Step 2**: Compute gradients of weights and biases (output layer)
- **Step 3**: Propagate error backwards
- **Step 4**: Compute gradients of weights and biases (hidden layers)
- **Step 5**: Update parameters using learning rate

Output Layer Error

$$dZ^{[L]} = A^{[L]} - Y$$

Propagate Error Backwards

$$dZ^{[l]} = (W^{[l+1]})^T dZ^{[l+1]} \circ \sigma'(Z^{[l]})$$

Compute Gradients

$$dW^{[L]} = \frac{1}{m} dZ^{[L]} (A^{[L-1]})^T$$

Update Parameters

$$W^{[l]} = W^{[l]} - \eta dW^{[l]}$$

Gradient Descent Variants

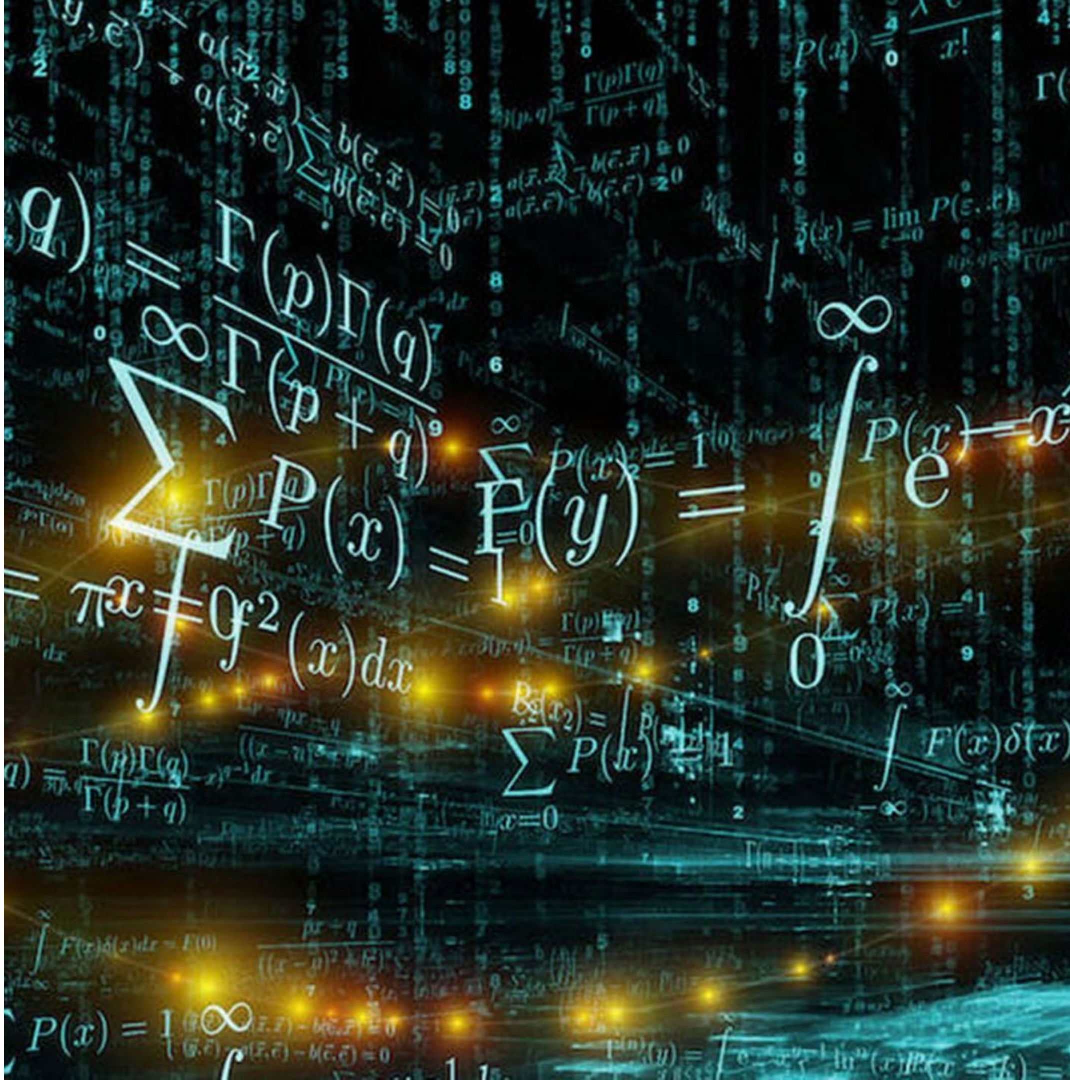
- The **gradient** is a vector (or matrix) of partial derivatives.
- **Backpropagation** efficiently computes all these partial derivatives using the **chain rule**.
- **Backpropagation** isn't just computing "a gradient" — it computes the entire set of partial derivatives (the full gradient) for the entire network.
- **Gradients** guide how we update parameters to **minimize** loss

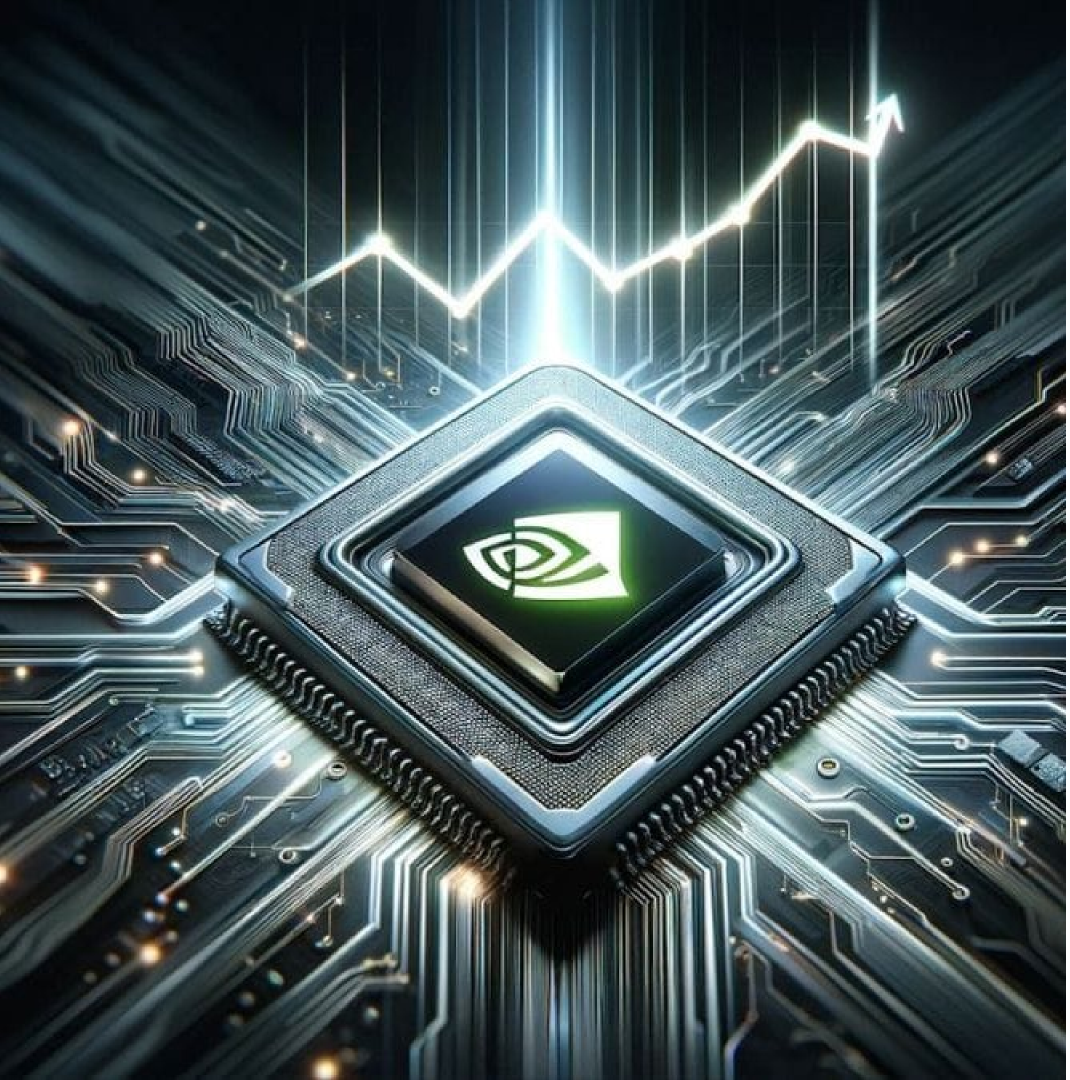
$$\nabla f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}]$$

$$\nabla \mathcal{L} = [\frac{\partial \mathcal{L}}{\partial W_1}, \frac{\partial \mathcal{L}}{\partial W_2}, \ldots, \frac{\partial \mathcal{L}}{\partial W_n}]$$

Computational Complexity

- Each **architecture** type has **different** training and testing complexity.
- Complexity mostly **scaled** with **depth and width** of the network.
- **Deep architectures** are **powerful** but come with cost and memory demands.

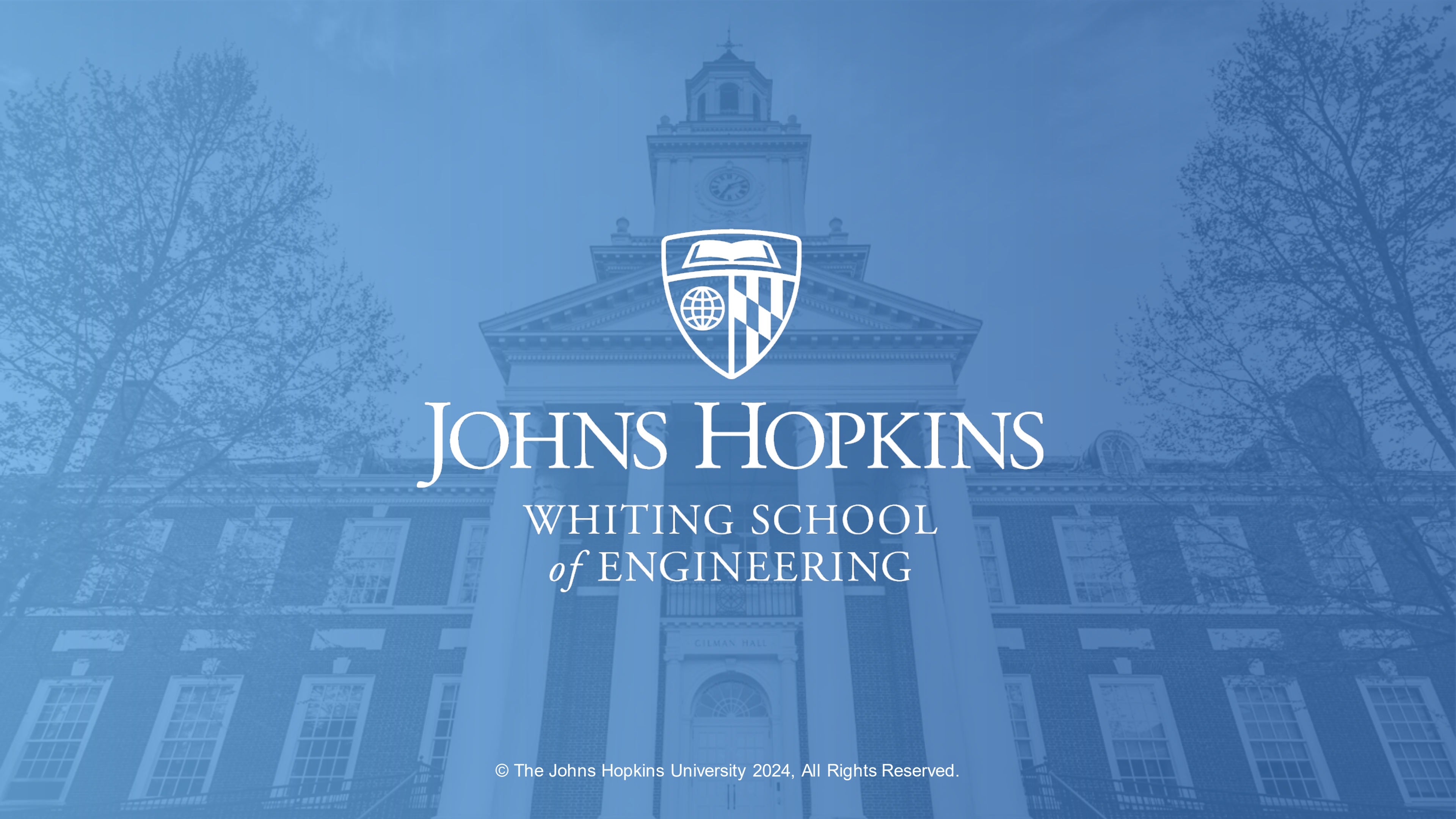




Hardware Acceleration for Training

- **Graphics Processing Units** (GPUs) have become the standard for accelerating deep learning workloads
- This has been a key enabler of modern **AI advancements**
- Although the cost of common GPUs has been going down the more **advanced cards** continue to be more **expensive**.

https://mpost.io/nvidia-launches-h200-gpu-to-ease-generative-ai-processing-workloads/



JOHNS HOPKINS

WHITING SCHOOL
of ENGINEERING