Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

October 10, 2024

1 Gradient Descent

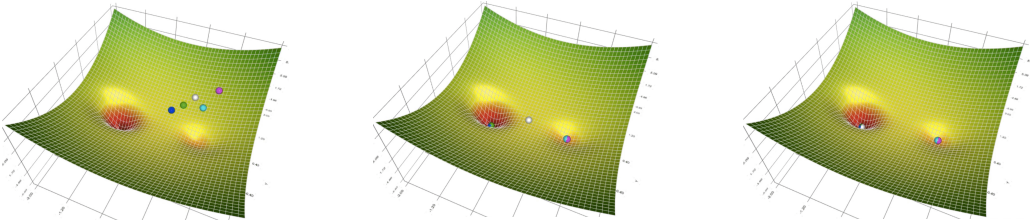
2 Backpropagation

3 Foundations in Detail: Initialization, Loss, and Activation

4 References

How to Update Weights?

- Imagine training a large model like ChatGPT. It has billions of parameters that need to be adjusted.
- If we used **random search** to update these weights, it would take an astronomical number of trials to find good parameters.
- How to make training feasible at this massive scale?



How to Update Weights?

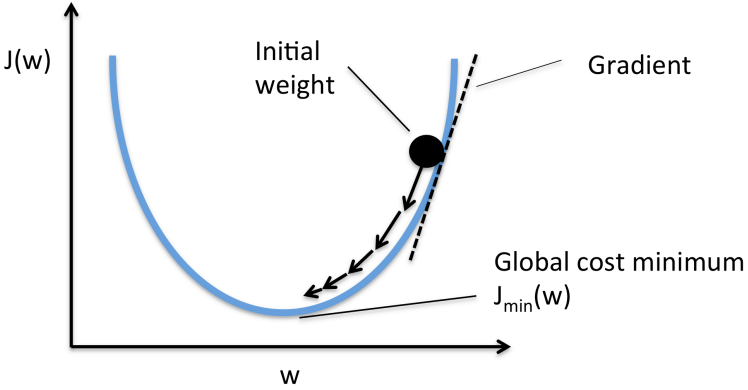
Options for Updating Weights:

- **Random Search:** Tries values randomlyinefficient and impractical.
- Gradient Descent: Follows the slope of the loss functionefficient and guided.

Why Gradient Descent?

- It updates weights by following the slope, reducing error with each step.
- Controlled, stepwise updates ensure we move closer to minimizing the loss effectively.

How to Update Weights?



Gradient Descent: Concept and Weight Updates

Gradient Descent: Minimizes the loss function by updating weights based on the gradient.
Weight Update Rule:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

Where:

- η is the learning rate (step size).
- $\frac{\partial L}{\partial w}$ is the gradient of the loss function with respect to w.

Example: Gradient Descent and Updating Weights

Example Problem:

- Initial weight: $w_0 = 2$
- Learning rate: $\eta = 0.1$
- Loss function: $L(w) = (y - wx)^2$

Gradient Calculation:

$$\frac{\partial L}{\partial w} = -2x(y - wx)$$

Example: For $x = 3$, $y = 10$, and $w_0 = 2$,

$$\frac{\partial L}{\partial w} = -24, \quad w_{\text{new}} = 5.6$$

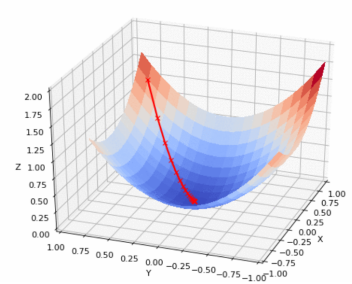
Gradient Descent: Formula and Process

Weight Update Formula:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

Steps in Gradient Descent:

- Compute the gradient of the loss function.
- Update the weights using the update rule.
- Repeat until convergence.



1 Gradient Descent

2 Backpropagation
- Forward and Backward Passes
- Vectorized Backpropagation
- Chain Rule

3 Foundations in Detail: Initialization, Loss, and Activation

4 References

1 Gradient Descent

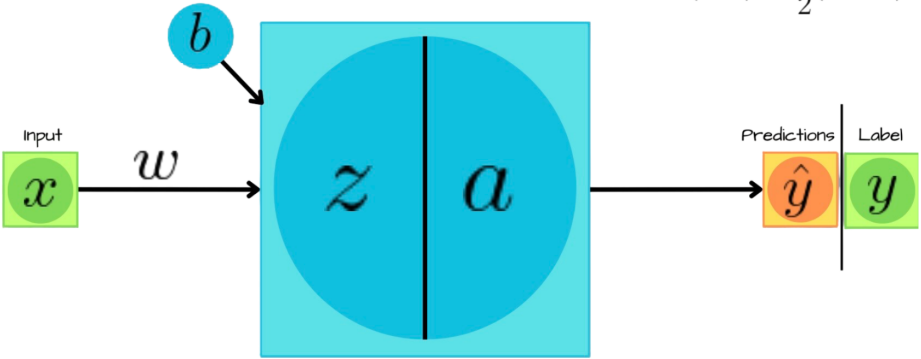
2 Backpropagation
- Forward and Backward Passes
- Vectorized Backpropagation
- Chain Rule

3 Foundations in Detail: Initialization, Loss, and Activation

4 References

Example: One Neuron

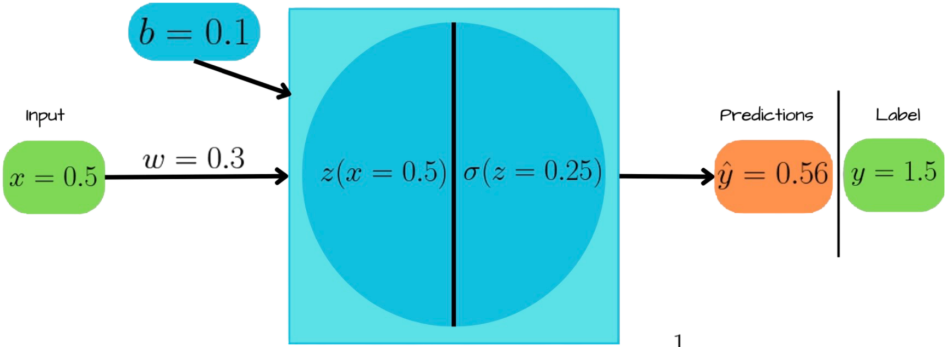
Total Error $L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$



$$z(x) = wx + b \quad a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

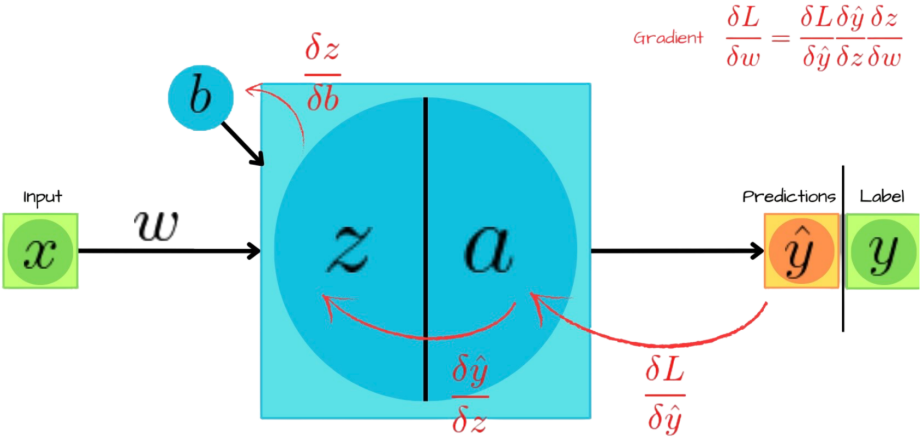
Forward Pass

Total Error $L(1.5, 0.56) = \frac{1}{2}(1.5 - 0.56)^2 = 0.44$



$$z(0.5) = 0.3 \cdot 0.5 + 0.1 = 0.25 \quad \sigma(z = 0.25) = \frac{1}{1 + e^{-0.25}} = 0.56$$

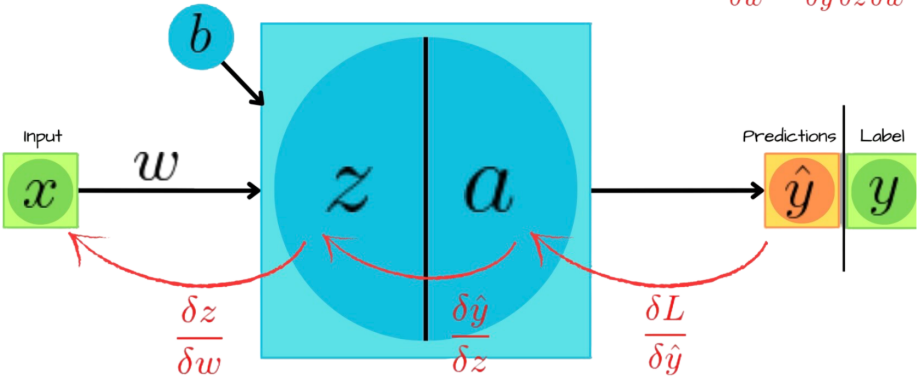
Backward Pass



$$z(x) = wx + b \quad a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

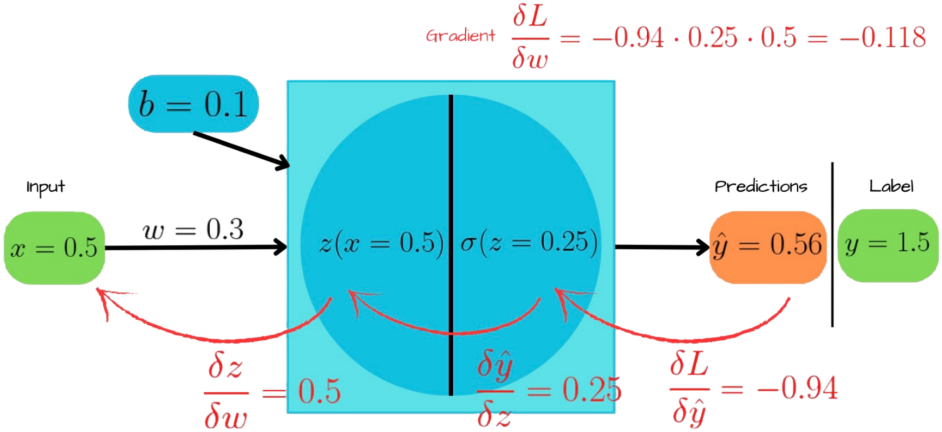
Backward Pass

Gradient $\frac{\delta L}{\delta w} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z} \frac{\delta z}{\delta w}$



$$z(x) = wx + b \quad a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

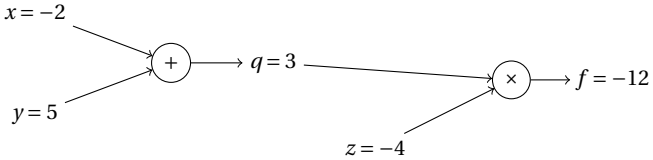
Backward Pass



Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$



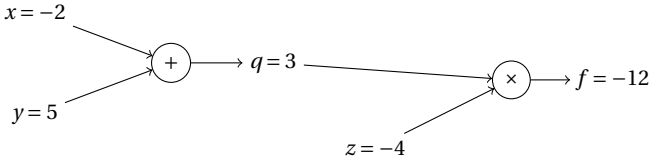
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



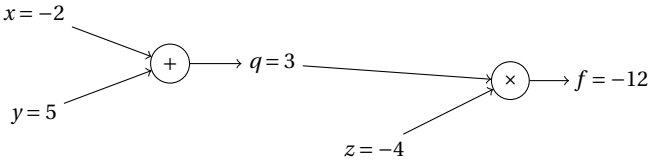
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

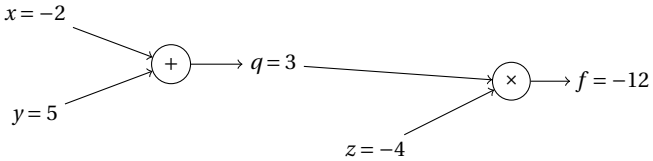
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

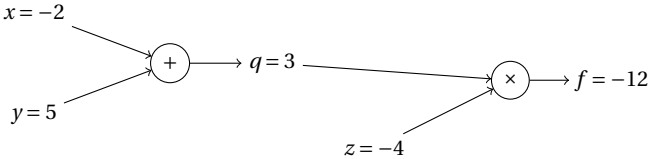
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

want $\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$

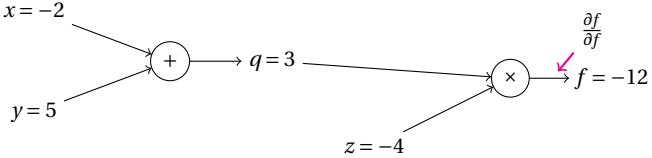
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

want $\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$

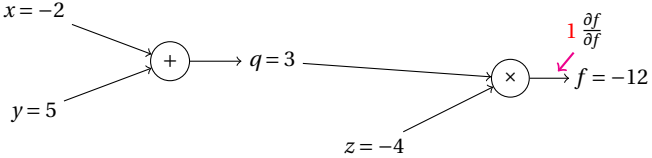
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

want $\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$

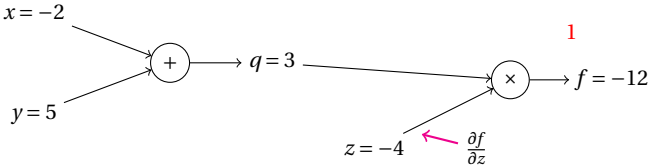
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

want $\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$

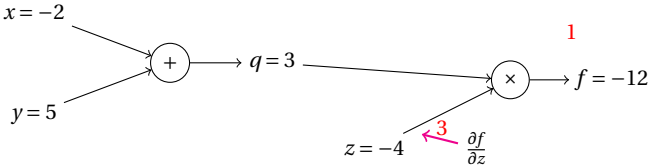
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

want $\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$

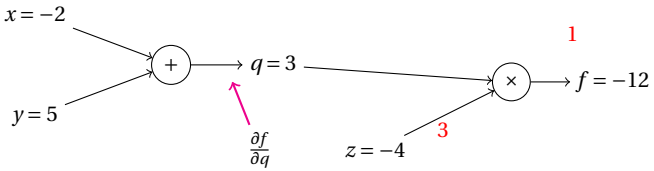
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

$$\textit{want} \quad \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$$

Backpropagation: a simple example

Function:

$$f(x,y,z) = (x+y)z$$

Example:

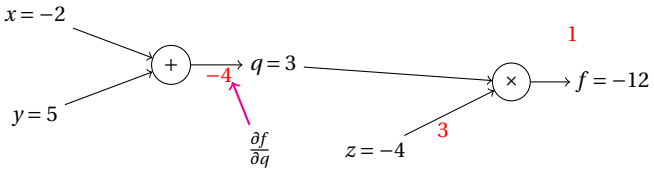
$$x = -2, \quad y = 5, \quad z = -4$$

Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$



want $\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$

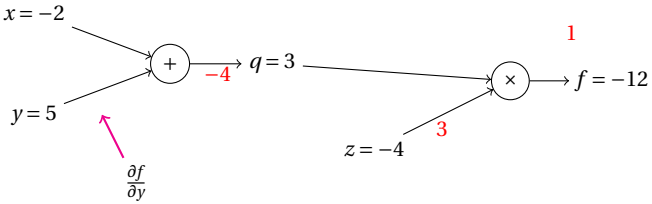
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

$$\textit{want} \quad \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}, \quad \textit{Chain rule} \quad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

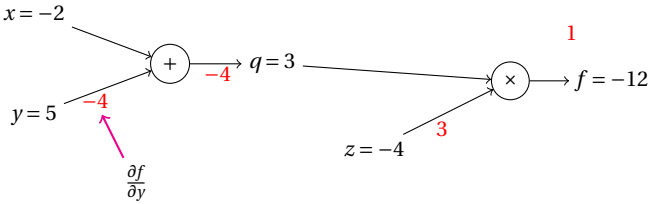
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

$$\textit{want} \quad \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}, \quad \textit{Chain rule} \quad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

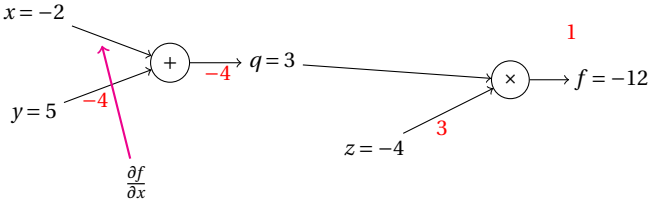
Backpropagation: a simple example

Function:

$$f(x, y, z) = (x + y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

$$\textit{want} \quad \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}, \quad \textit{Chain rule} \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

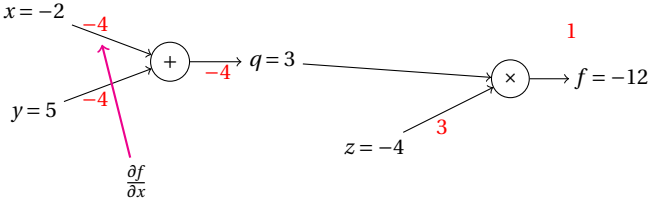
Backpropagation: a simple example

Function:

$$f(x,y,z) = (x+y)z$$

Example:

$$x = -2, \quad y = 5, \quad z = -4$$



Step 1:

$$q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$$

Step 2:

$$f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q$$

$$\textit{want} \quad \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}, \quad \textit{Chain rule} \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

1 Gradient Descent

2 Backpropagation
- Forward and Backward Passes
- Vectorized Backpropagation
- Chain Rule

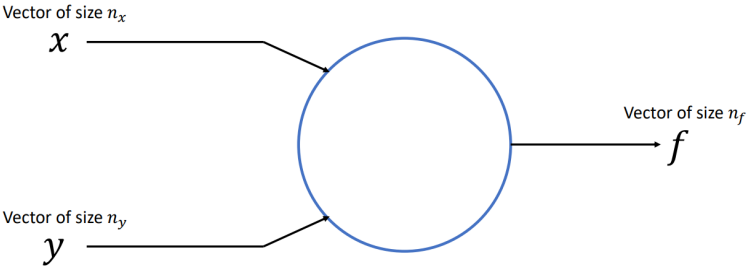
3 Foundations in Detail: Initialization, Loss, and Activation

4 References

Vectorized Backpropagation

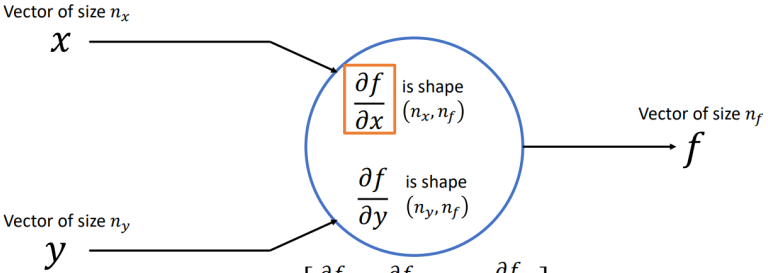
The faster you compute gradients, the quicker each parameter update in gradient descent.

Derivative of a Vector by a Vector: leveraging matrix operations for faster computation.



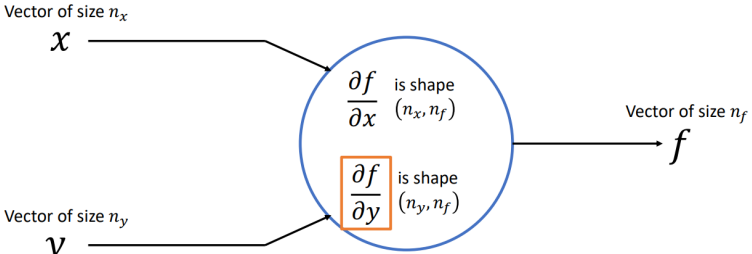
Vectorized Backpropagation

Local Derivatives are Jacobian Matrices



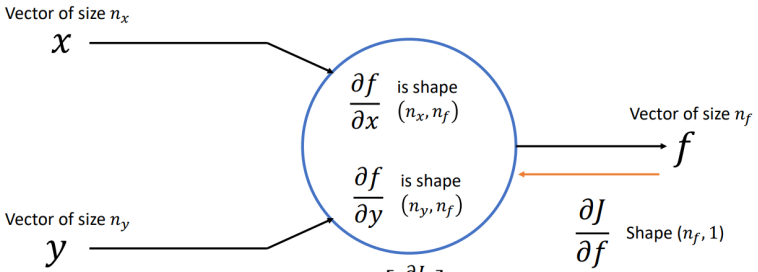
$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_{n_f}}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & & \frac{\partial f_{n_f}}{\partial x_2} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_{n_x}} & \frac{\partial f_2}{\partial x_{n_x}} & \cdots & \frac{\partial f_{n_f}}{\partial x_{n_x}} \end{bmatrix}$$

Vectorized Backpropagation



$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_2}{\partial y_1} & \cdots & \frac{\partial f_{n_f}}{\partial y_1} \\ \frac{\partial f_1}{\partial y_2} & \frac{\partial f_2}{\partial y_2} & & \frac{\partial f_{n_f}}{\partial y_2} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_1}{\partial y_{n_y}} & \frac{\partial f_2}{\partial y_{n_y}} & \cdots & \frac{\partial f_{n_f}}{\partial y_{n_y}} \end{bmatrix}$$

Vectorized Backpropagation

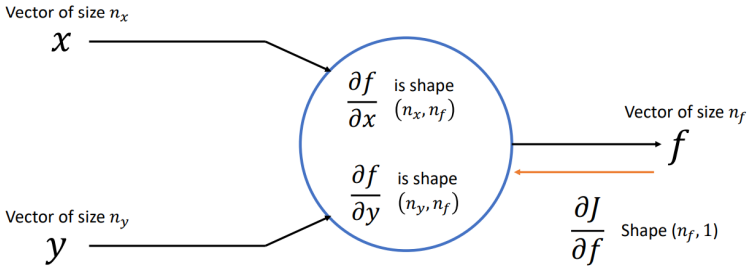


$$\frac{\partial J}{\partial f} = \begin{bmatrix} \frac{\partial J}{\partial f_1} \\ \frac{\partial J}{\partial f_2} \\ \vdots \\ \frac{\partial J}{\partial f_{n_f}} \end{bmatrix}$$

Upstream gradient is with respect to Cost J (a **scalar**)

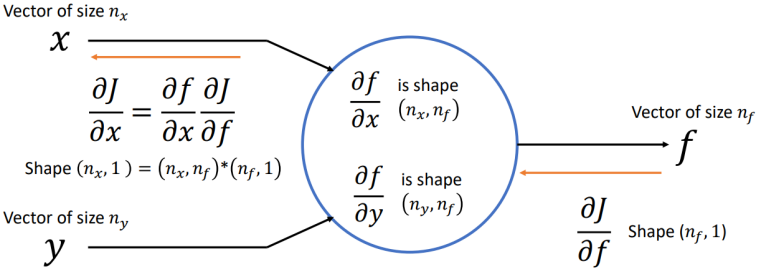
i.e. How does each output f_i affect the Cost

Vectorized Backpropagation



Apply chain rule like before!

Vectorized Backpropagation



Applying the chain rule involves matrix-vector multiplication

Vectorized Backpropagation

Vector of size n_x

x

$$\frac{\partial J}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial J}{\partial f}$$

Shape $(n_x, 1) = (n_x, n_f)^*(n_f, 1)$

Vector of size n_y

y

$\frac{\partial f}{\partial x}$ is shape (n_x, n_f)

$\frac{\partial f}{\partial y}$ is shape (n_y, n_f)

Vector of size n_f

f

$\frac{\partial J}{\partial f}$ Shape $(n_f, 1)$

$$\frac{\partial J}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial J}{\partial f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, & \frac{\partial f_1}{\partial x_2}, & \cdots & \frac{\partial f_1}{\partial x_{n_x}} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_{n_f}}{\partial x_1}, & \frac{\partial f_{n_f}}{\partial x_2} & \cdots & \frac{\partial f_{n_f}}{\partial x_{n_x}} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial f_1} \\ \frac{\partial J}{\partial f_2} \\ \vdots \\ \frac{\partial J}{\partial f_{n_f}} \end{bmatrix}$$

1 Gradient Descent

2 Backpropagation
- Forward and Backward Passes
- Vectorized Backpropagation
- Chain Rule

3 Foundations in Detail: Initialization, Loss, and Activation

4 References

Chain Rule Matrix-Vector Multiply

$$\frac{\partial J}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial J}{\partial f} \rightarrow \begin{bmatrix} \frac{\partial J}{\partial x_1} \\ \frac{\partial J}{\partial x_2} \\ \vdots \\ \frac{\partial J}{\partial x_{n_x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, \frac{\partial f_2}{\partial x_1}, & \cdots & \frac{\partial f_{n_f}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_{n_x}}, \frac{\partial f_2}{\partial x_{n_x}} & \cdots & \frac{\partial f_{n_f}}{\partial x_{n_x}} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial f_1} \\ \frac{\partial J}{\partial f_2} \\ \vdots \\ \frac{\partial J}{\partial f_{n_f}} \end{bmatrix}$$

Shape $(n_x, 1) = (n_x, n_f)^*(n_f, 1)$

Chain Rule Matrix-Vector Multiply

$$\frac{\partial J}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial J}{\partial f} \rightarrow \begin{bmatrix} \frac{\partial J}{\partial x_1} \\ \frac{\partial J}{\partial x_2} \\ \vdots \\ \frac{\partial J}{\partial x_{n_x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, \frac{\partial f_2}{\partial x_1}, & \cdots & \frac{\partial f_{n_f}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_{n_x}}, \frac{\partial f_2}{\partial x_{n_x}} & \cdots & \frac{\partial f_{n_f}}{\partial x_{n_x}} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial f_1} \\ \frac{\partial J}{\partial f_2} \\ \vdots \\ \frac{\partial J}{\partial f_{n_f}} \end{bmatrix}$$

Jacobian

Shape $(n_x, 1) = (n_x, n_f)^*(n_f, 1)$

Chain Rule Matrix-Vector Multiply

$$\frac{\partial J}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial J}{\partial f} \rightarrow \begin{bmatrix} \frac{\partial J}{\partial x_1} \\ \frac{\partial J}{\partial x_2} \\ \vdots \\ \frac{\partial J}{\partial x_{n_x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, \frac{\partial f_2}{\partial x_1}, & \cdots & \frac{\partial f_{n_f}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_{n_x}}, \frac{\partial f_2}{\partial x_{n_x}} & \cdots & \frac{\partial f_{n_f}}{\partial x_{n_x}} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial f_1} \\ \frac{\partial J}{\partial f_2} \\ \vdots \\ \frac{\partial J}{\partial f_{n_f}} \end{bmatrix}$$

Jacobian

Shape $(n_x, 1) = (n_x, n_f) * (n_f, 1)$

Upstream Gradient

Chain Rule Matrix-Vector Multiply

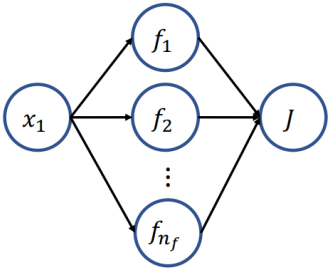
$$\frac{\partial J}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial J}{\partial f} \rightarrow \begin{bmatrix} \boxed{\frac{\partial J}{\partial x_1}} \\ \frac{\partial J}{\partial x_2} \\ \vdots \\ \frac{\partial J}{\partial x_{n_x}} \end{bmatrix} = \boxed{\begin{bmatrix} \frac{\partial f_1}{\partial x_1}, \frac{\partial f_2}{\partial x_1}, & \cdots & \frac{\partial f_{n_f}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_{n_x}}, \frac{\partial f_2}{\partial x_{n_x}} & \cdots & \frac{\partial f_{n_f}}{\partial x_{n_x}} \end{bmatrix}} \boxed{\begin{bmatrix} \frac{\partial J}{\partial f_1} \\ \frac{\partial J}{\partial f_2} \\ \vdots \\ \frac{\partial J}{\partial f_{n_f}} \end{bmatrix}}$$

Shape $(n_x, 1) = (n_x, n_f)^*(n_f, 1)$

$$\frac{\partial J}{\partial x_1} = \frac{\partial f_1}{\partial x_1}\frac{\partial J}{\partial f_1} + \frac{\partial f_2}{\partial x_1}\frac{\partial J}{\partial f_2} + \cdots + \frac{\partial f_{n_f}}{\partial x_1}\frac{\partial J}{\partial f_{n_f}}$$

Chain Rule Matrix-Vector Multiply

$$\frac{\partial J}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial J}{\partial f} \rightarrow \begin{bmatrix} \frac{\partial J}{\partial x_1} \\ \frac{\partial J}{\partial x_2} \\ \vdots \\ \frac{\partial J}{\partial x_{n_x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}, & \frac{\partial f_2}{\partial x_1}, & \cdots & \frac{\partial f_{n_f}}{\partial x_1} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_{n_x}}, & \frac{\partial f_2}{\partial x_{n_x}} & \cdots & \frac{\partial f_{n_f}}{\partial x_{n_x}} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial f_1} \\ \frac{\partial J}{\partial f_2} \\ \vdots \\ \frac{\partial J}{\partial f_{n_f}} \end{bmatrix}$$



Shape $(n_x, 1) = (n_x, n_f)^*(n_f, 1)$

$$\frac{\partial J}{\partial x_1} = \frac{\partial f_1}{\partial x_1}\frac{\partial J}{\partial f_1} + \frac{\partial f_2}{\partial x_1}\frac{\partial J}{\partial f_2} + \cdots + \frac{\partial f_{n_f}}{\partial x_1}\frac{\partial J}{\partial f_{n_f}}$$

Chain Rule Matrix-Vector Multiply

Vector of size n_x

x

$$\frac{\partial J}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial J}{\partial f}$$

Shape $(n_x, 1) = (n_x, n_f)*(n_f, 1)$

Vector of size n_y

y

$$\frac{\partial J}{\partial y} = \frac{\partial f}{\partial y}\frac{\partial J}{\partial f}$$

Shape $(n_y, 1) = (n_y, n_f)*(n_f, 1)$

$\frac{\partial f}{\partial x}$ is shape (n_x, n_f)

$\frac{\partial f}{\partial y}$ is shape (n_y, n_f)

Vector of size n_f

f

$\frac{\partial J}{\partial f}$ Shape $(n_f, 1)$

Upstream gradient is with respect to Cost J (a **scalar**)

i.e. How does each output f_i affect the Cost

Chain Rule application is Matrix-Vector Multiply

1 Gradient Descent

2 Backpropagation

3 Foundations in Detail: Initialization, Loss, and Activation
- Weight Initialization
- Loss Functions
- Activation Functions

4 References

1 Gradient Descent

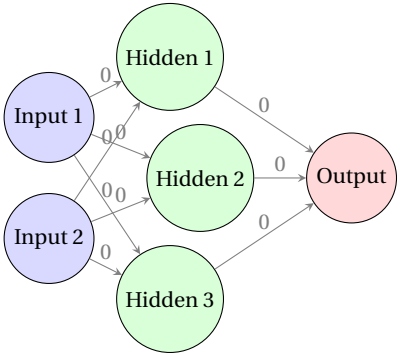
2 Backpropagation

3 Foundations in Detail: Initialization, Loss, and Activation
- Weight Initialization
- Loss Functions
- Activation Functions

4 References

Weight Initialization

Example: Imagine a network where all weights are initialized to zero.
Issue: If all weights are zero, each neuron in a layer will produce identical outputs. This symmetry prevents the network from learning distinct features, as every neuron updates identically.
Solution: To break this symmetry, weights need to be initialized with small random values, allowing neurons to learn unique features and avoid identical updates.



All weights initialized to zero

Why Weight Initialization Matters

Importance:

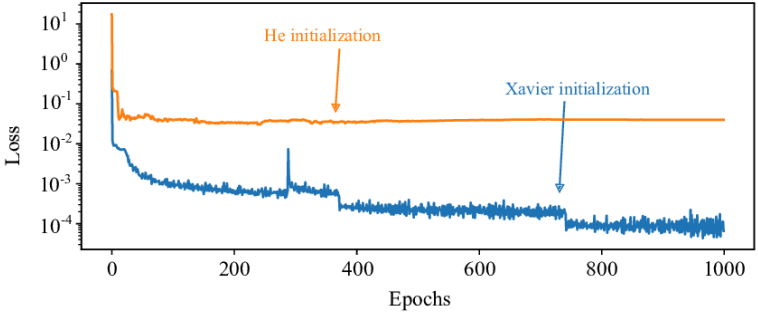
- Proper initialization ensures faster convergence and improves training stability.
- Prevents issues like vanishing or exploding gradients, which can make training slow or unstable.

Question: How can we initialize weights to maximize learning efficiency and prevent gradient problems?

Key Initialization Techniques

- **Zero Initialization:** Set all weights to zero.
  - Rarely used, as it leads to identical updates for all neurons.
- **Random Initialization:** Assign small random values to weights.
  - Helps break symmetry, but can still cause issues with gradient magnitudes.
- **Xavier Initialization:**
  - Scales weights by $\sqrt{\frac{1}{\text{number of input neurons}}}$.
  - Maintains variance across layers for efficient learning, suitable for sigmoid and tanh activations.
- **He Initialization:**
  - Scales weights by $\sqrt{\frac{2}{\text{number of input neurons}}}$.
  - Optimized for ReLU activation functions to prevent vanishing/exploding gradients.
- **Key Point:** Proper initialization reduces the risk of gradient issues and helps the network converge faster.

Xavier vs He



Evolution of loss term for Xavier weight initialization and He weight initialization.

Choosing the Right Initialization Examples

- **Scenario 1:** Using ReLU activation functions in a deep network.
  - **Best Choice:** He Initialization.
  - **Reason:** Helps maintain gradient flow through the layers.
- **Scenario 2:** Using Sigmoid activation functions in a shallow network.
  - **Best Choice:** Xavier Initialization.
  - **Reason:** Keeps variance balanced, which is crucial for non-ReLU activations.
- **Experiment:** Try initializing with zeros and random weights to see how it impacts training speed and performance.

Transition to Loss and Activation Functions

Recap: Proper weight initialization:

- Ensures stability during training by maintaining gradient magnitudes.
- Helps the network converge faster and learn more effectively.

Next Steps:

- Once weights are initialized, the network needs a measure of error this is where **loss functions** come in.
- After initializing weights, **activation functions** determine the output of each neuron, enabling the network to learn complex patterns.

Question: How do we measure the error in predictions and adjust our weights to minimize it?

1 Gradient Descent

2 Backpropagation

3 Foundations in Detail: Initialization, Loss, and Activation
- Weight Initialization
- Loss Functions
- Activation Functions

4 References

Overview of Loss Functions

Types of Loss Functions:

- **Mean Squared Error (MSE):** Used in regression to minimize the squared difference between predicted and true values.
- **Mean Absolute Error (MAE):** Also for regression, minimizing absolute differences between predicted and true values.
- **Binary Cross-Entropy:** Used for binary classification to minimize the difference between predicted probabilities and binary labels.
- **Categorical Cross-Entropy:** For multi-class classification, comparing predicted probabilities across multiple classes.

Mean Squared Error (MSE) Multiple Samples

Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2$$

Example:

- Predicted values: $\hat{y} = [4.2, 3.8, 5.1]$
- True values: $y = [5.0, 4.0, 4.9]$
- Calculation:

$$\text{MSE} = \frac{1}{3}\left[(5.0-4.2)^2 + (4.0-3.8)^2 + (4.9-5.1)^2\right]$$

$$\text{MSE} = \frac{1}{3}[0.64 + 0.04 + 0.04] = \frac{0.72}{3} = 0.24$$

Mean Absolute Error (MAE) Multiple Samples

Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|y_i - \hat{y}_i\right|$$

Example:

- Predicted values: $\hat{y} = [4.2, 3.8, 5.1]$
- True values: $y = [5.0, 4.0, 4.9]$
- Calculation:

$$\text{MAE} = \frac{1}{3}\left(|5.0 - 4.2| + |4.0 - 3.8| + |4.9 - 5.1|\right)$$

$$\text{MAE} = \frac{1}{3}(0.8 + 0.2 + 0.2) = \frac{1.2}{3} \approx 0.4$$

Binary Classification Loss Binary Cross-Entropy

Binary Cross-Entropy:

$$\mathscr{L}_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

Example:

- Predicted probabilities: $\hat{y} = [0.7, 0.3, 0.9]$
- True labels: $y = [1, 0, 1]$
- Calculation:

$$\mathscr{L}_{\text{BCE}} = -\frac{1}{3}\left[\log(0.7) + \log(0.7) + \log(0.9)\right]$$

$$\approx -\frac{1}{3}(-0.35 + -0.357 + -0.105) \approx 0.273$$

Used to minimize the error between predicted probabilities and binary labels.

Multi-Class Classification Loss Categorical Cross-Entropy

Categorical Cross-Entropy:

$$\mathscr{L}_{\text{CCE}} = -\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

Example (3-Class):

- True labels (One-hot): Class 2, Class 1, Class 3
- Predicted probabilities:

$$\hat{y} = [[0.1, 0.7, 0.2], [0.6, 0.3, 0.1], [0.1, 0.6, 0.3]]$$

- Calculation:

$$\mathscr{L}_{\text{CCE}} = -\frac{1}{3}\big(\log(0.7) + \log(0.6) + \log(0.3)\big)$$

$$\approx -\frac{1}{3}(-0.357 - 0.511 - 1.204) \approx 0.691$$

Optimizes multi-class classification by matching predicted probabilities to true classes.

1 Gradient Descent

2 Backpropagation

3 Foundations in Detail: Initialization, Loss, and Activation
- Weight Initialization
- Loss Functions
- Activation Functions

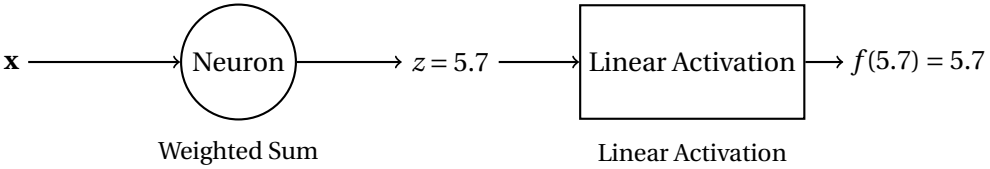
4 References

Linear Activation - A Limitation

Linear Activation:

$$f(z) = z$$

- Example: If a neuron produces a raw output $z = 5.7$, linear activation would pass this unchanged.



Limitation of Linear Activation

Why Transform Outputs? Raw outputs need to be transformed into meaningful values, such as probabilities.

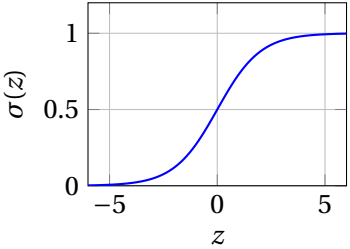
The Problem: Linear activation lacks non-linearity, restricting the model to simple linear relationships.

Sigmoid

Characteristics of Sigmoid:

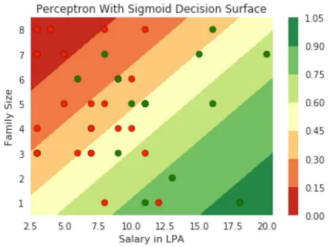
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Squashes the input between 0 and 1, which makes it useful in probabilistic interpretations (e.g., logistic regression).
- Often used in output layers for binary classification problems.



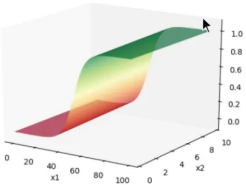
Model: Sigmoid Decision Surface

Model



$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

	Salary in LPA	Family Size	Buys Car?
0	11	8	1
1	20	7	1
2	4	8	0
3	8	7	0
4	11	5	1

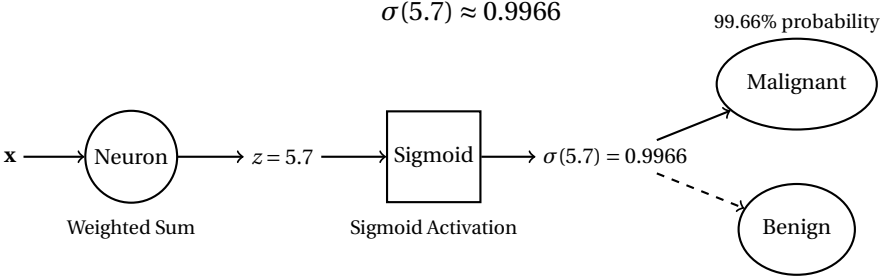


Classification: Tumor Detection (Malignant vs. Benign)

Sigmoid Activation: Useful for binary classification!

- Example: For $z = 5.7$,

$$\sigma(5.7) \approx 0.9966$$



Sigmoid

Limitations of Sigmoid:

- **Gradient Saturation:** When z is very large or very small, the gradient becomes nearly zero, causing slow learning (vanishing gradient problem).
- **Not Zero-Centered:** The output is not zero-centered, which can make optimization more difficult.

Question:

- Why does the vanishing gradient problem occur with Sigmoid during backpropagation? (To be discussed in more detail later)

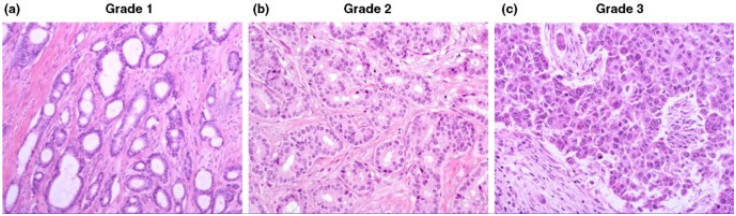
Problem: Multi-Class Tumor Classification

Scenario: We want to classify a tumor into one of three categories:

- **Class 0**: Benign
- **Class 1**: Malignant
- **Class 2**: Pre-cancerous

Goal: Given a set of tumor features, predict which class the tumor belongs to.

This is a **multi-class classification problem**, and we will use the **Softmax activation function** to assign probabilities to each class.



Softmax Activation: The Model

In multi-class classification, Softmax is used to convert raw outputs (logits) into probabilities for each class.

Softmax Function:

$$P(y = i|X) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where:

- z_i is the raw output (logit) for class i.
- K is the number of classes (in this case, 3: benign, malignant, pre-cancerous).

The Softmax function ensures that the sum of the probabilities for all classes is 1, and the class with the highest probability is chosen as the prediction.

Example: Softmax Calculation

Consider a tumor with the following logits from a neural network:

- Logit for Benign (Class 0): $z_0 = 1.5$
- Logit for Malignant (Class 1): $z_1 = 0.8$
- Logit for Pre-Cancerous (Class 2): $z_2 = -0.5$

Step 1: Exponentiate the logits

$$e^{z_0} = e^{1.5} \approx 4.48, \quad e^{z_1} = e^{0.8} \approx 2.23, \quad e^{z_2} = e^{-0.5} \approx 0.61$$

Step 2: Compute the sum of exponentials

$$\text{Sum} = e^{z_0} + e^{z_1} + e^{z_2} = 4.48 + 2.23 + 0.61 = 7.32$$

Example: Softmax Probabilities

Step 3: Calculate Softmax probabilities for each class

$$P(\text{Benign}) = \frac{4.48}{7.32} \approx 0.612, \quad P(\text{Malignant}) = \frac{2.23}{7.32} \approx 0.305, \quad P(\text{Pre-Cancerous}) = \frac{0.61}{7.32} \approx 0.083$$

Step 4: Make a classification decision

- The highest probability is 0.612 for the **Benign** class.
- Therefore, the model predicts that the tumor is **Benign** (Class 0).

Conclusion: Softmax Activation for Classification

Key Points:

- Softmax is used in the output layer for **multi-class classification**.
- It converts logits into a **probability distribution** across classes.
- The class with the highest probability is selected as the prediction.

Main Idea: Softmax ensures all outputs sum to 1, making it ideal for choosing one class out of multiple options.

Tanh (Hyperbolic Tangent)

Characteristics of Tanh:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- Squashes input between -1 and 1, making it zero-centered (Balanced Updates → Reduced Bias in Gradient Descent → Faster Convergence)

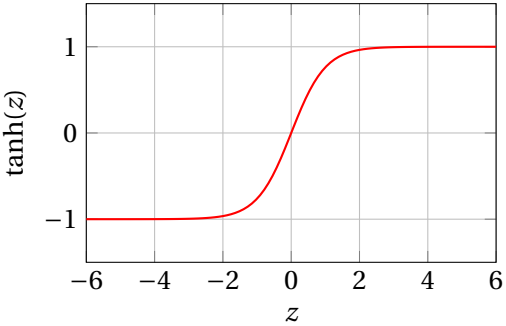
Advantages of Tanh:

- **Zero-Centered**: Output ranges from -1 to 1, making optimization easier.
- Better for **hidden layers** than Sigmoid due to zero-centered output.

Limitations:

- Similar saturation issues as Sigmoid: large input values push gradients towards zero (vanishing gradient problem).

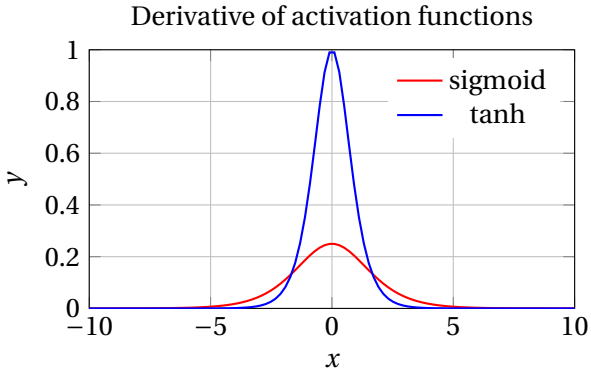
Tanh (Hyperbolic Tangent)



Question:

- How does Tanh help with faster convergence compared to Sigmoid?

Comparison: Sigmoid vs Tanh

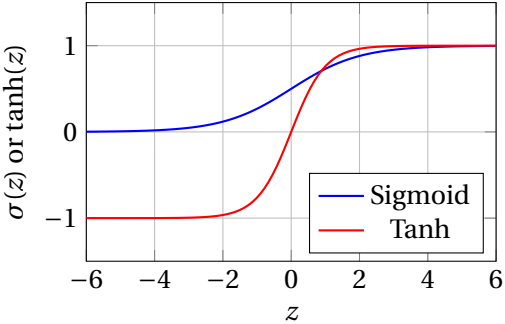


- The derivative of the Tanh function has a much steeper slope at $x = 0$, meaning it provides a larger gradient for backpropagation compared to the Sigmoid function.

Comparison: Sigmoid vs Tanh

Key Differences:

- **Sigmoid**: Maps input to [0, 1]. Output is not zero-centered.
- **Tanh**: Maps input to [-1, 1]. Output is zero-centered, leading to easier optimization.



Comparison: Sigmoid vs Tanh

When to Use:

- **Sigmoid**: Best for binary classification tasks, particularly in the output layer.
- **Tanh**: More suitable for hidden layers due to its centered output, allowing faster training.

Question:

- In what scenario might Sigmoid be preferred over Tanh, despite its limitations?

Neural Networks: Why is the Max Operator Important?

- **Before:** Linear score function:

$$f = Wx$$

- **Now:** 2-layer Neural Network:

$$f = W_2 \max(0, W_1 x)$$

- The function $\max(0, z)$ is called an activation function (in this case, ReLU).
- Q: What if we try to build a neural network without an activation function?

$$f = W_2 W_1 x$$

$$W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, \quad f = W_3 x$$

- **A:** We end up with a linear classifier again!

ReLU

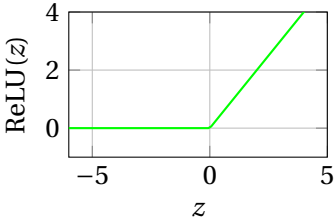
Characteristics of ReLU:

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

- Faster convergence: Efficient computation, especially for deep networks.

Advantages of ReLU:

- Does not saturate for positive values, helping to avoid the vanishing gradient problem.
- Computationally efficient (simpler than Sigmoid/Tanh).



ReLU

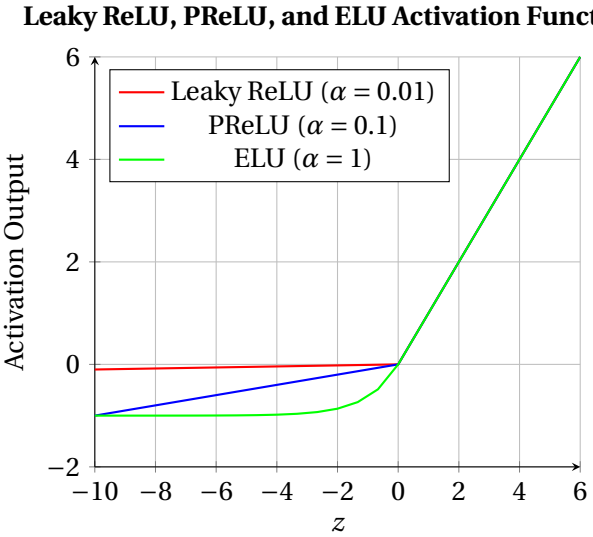
Limitation:

- **Dead ReLU Problem**: Neurons can become inactive during training, outputting 0 for all inputs if they receive negative values consistently.

Question:

- Why does ReLU lead to faster training in deep networks?

Variants of ReLU: Leaky ReLU, PReLU, ELU

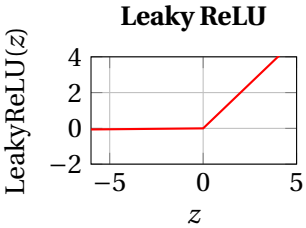


Variants of ReLU: Leaky ReLU

- Allows a small, non-zero gradient for negative inputs.

$$\text{LeakyReLU}(z) = \max(\alpha z, z), \quad \alpha = 0.01$$

- Helps prevent the "dead ReLU" problem, where neurons stop updating.



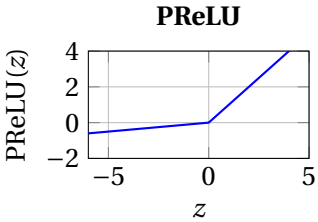
Leaky ReLU: Allows a small, non-zero gradient for negative inputs.

Variants of ReLU: PReLU (Parametric ReLU)

- Similar to Leaky ReLU, but the slope for negative inputs (α) is learned during training.

$$\mathrm{PReLU}(z) = \max(\alpha z, z), \quad \alpha \text{ is learned}$$

- Provides more flexibility by adjusting the slope for negative inputs based on data.



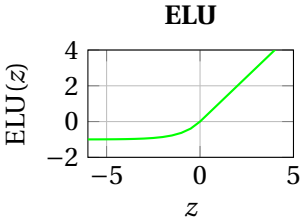
PReLU: Similar to Leaky ReLU, with a learnable slope.

Variants of ReLU: ELU (Exponential Linear Unit)

- Similar to ReLU for positive values but smoother for negative inputs.

$$\mathrm{ELU}(z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha(e^z - 1), & \text{if } z \leq 0 \end{cases}, \quad \alpha = 1$$

- Provides faster convergence and reduces bias shift by smoothing negative values.



ELU: Smoother than ReLU for negative values.

1 Gradient Descent

2 Backpropagation

3 Foundations in Detail: Initialization, Loss, and Activation

4 References

[1] T. Networks, "Benign vs malignant tumors." *Web article*, 2023.
Available: https://www.technologynetworks.com/cancer-research/articles/benign-vs-malignant-tumors-364765.

[2] B. Central, "Breast cancer research article." *Journal article*, 2010.
Available: https://breast-cancer-research.biomedcentral.com/articles/10.1186/bcr2607.

[3] S. Raschka, "Gradient optimization image." *Image*, 2020.
Available: https://sebastianraschka.com/images/faq/gradient-optimization/ball.png.

[4] Xnought, "Backpropagation explainer." *Website*, 2022.
Available: https://xnought.github.io/backprop-explainer/.

[5] Datamapu, "Deep learning backpropagation." *Web article*, 2021.
Available: https://datamapu.com/posts/deep_learning/backpropagation/.

[6] B. Quinton, "Elec502 vectorized backpropagation slides." *Lecture slides*, 2021.
Available: https://people.ece.ubc.ca/bradq/ELEC502Slides/ELEC502-Part5VectorizedBackpropagation.pdf.

[7] D. D. Investor, "Simplified sigmoid neuron: A building block of deep neural networks." *Web article*, 2019.
Available: https://medium.datadriveninvestor.com/simplified-sigmoid-neuron-a-building-block-of-deep-neural-network-5bfa

[8] ResearchGate, "Evolution of loss term for xavier and he weight initialization." *ResearchGate image*, 2023.
Available: https://www.researchgate.net/publication/363843256/figure/fig7/AS:1143128108618505 6@1664161711792/Evolution-of-loss-term-for-Xavier-weight-initialization-and-He-weight-
png.

Any Questions?