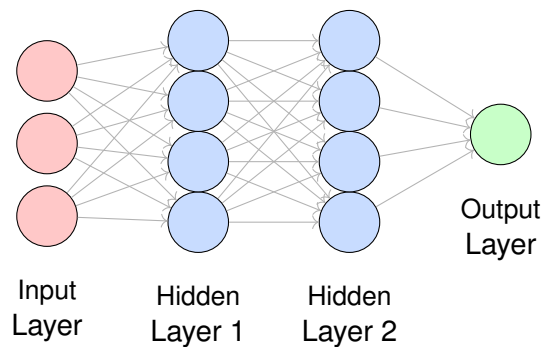


Feedforward Neural Networks

Comprehensive Lecture Notes

Deep Learning for Perception

Undergraduate Computer Science (BSCS)



Topics Covered:

- Motivation for Neural Networks: The Need for Non-Linear Models
- Neural Network Architecture: Hidden Layers
- Neural Network Architecture: Activation Functions
- Neural Network Architecture: Output Units (Softmax)

Contents

1	Motivation for Neural Networks: The Need for Non-Linear Models	2
1.1	Recall: The Perceptron	2
1.2	The XOR Problem: Perceptron's Limitation	3
1.3	From Biological to Artificial Neural Networks	4
2	Neural Network Architecture: Hidden Layers	4
2.1	What is a Hidden Layer?	4
2.2	Counting Parameters in a Neural Network	5
2.3	Fully Connected & Feedforward Networks	6
2.4	Why Hidden Layers Alone Are NOT Enough	6
3	Neural Network Architecture: Activation Functions	7
3.1	The Big Three: Sigmoid, Tanh, and ReLU	8
3.1.1	Sigmoid Function	8
3.1.2	Hyperbolic Tangent (Tanh)	9
3.1.3	Rectified Linear Unit (ReLU)	9
3.2	Summary: Activation Function Comparison	10
3.3	Solving XOR with Non-Linear Activation	10
3.4	Effect of Weights and Biases on Activation	11
4	Neural Network Architecture: Output Units	12
4.1	Linear Output (Regression)	12
4.2	Sigmoid Output (Binary Classification)	13
4.3	Softmax Output (Multiclass Classification)	13
4.4	Summary: Output Activation by Task	15
5	Putting It All Together	15
	Review Questions	16

1 Motivation for Neural Networks: The Need for Non-Linear Models

Key Definition

Neural Network: A computational model inspired by biological neurons, consisting of interconnected units (artificial neurons) organized in layers that can learn complex patterns from data.

Non-linear Model: A model where the output is *not* a simple linear combination (weighted sum) of inputs. Non-linear models can capture curved boundaries and complex relationships in data.

1.1 Recall: The Perceptron

Before understanding why we need neural networks, let us recall the **Perceptron** [a single artificial neuron that makes decisions based on weighted inputs].

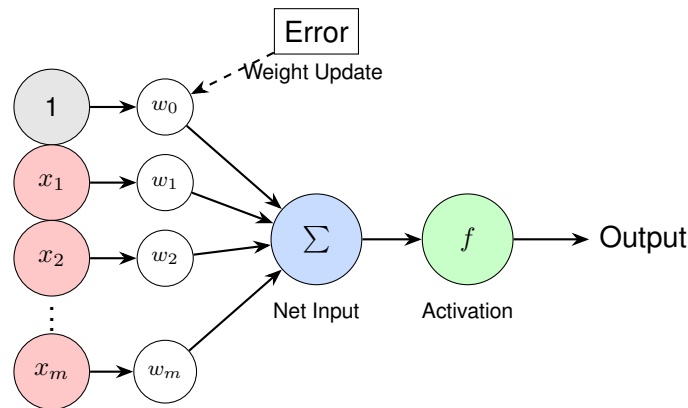


Figure 1: The Perceptron: A single artificial neuron. It computes a weighted sum of inputs, adds a bias (w_0), and applies an activation function to produce an output.

Analogy / Concrete Example

Think of a Perceptron like a **judge making a yes/no decision**. The judge considers multiple pieces of evidence (x_1, x_2, \dots), weighs each piece according to its importance (w_1, w_2, \dots), adds them up, and if the total exceeds a threshold, the verdict is “yes” (output = 1); otherwise, “no” (output = 0).

The mathematical formulation of a Perceptron is:

$$\hat{y} = f\left(\sum_{i=0}^m w_i x_i\right) = f(w_0 + w_1 x_1 + w_2 x_2 + \dots + w_m x_m) \quad (1)$$

where:

- x_i = input features (with $x_0 = 1$ for the bias term)
- w_i = weights [learnable parameters that determine the importance of each input]
- $f(\cdot)$ = activation function (e.g., step function for original Perceptron)
- \hat{y} = predicted output

1.2 The XOR Problem: Perceptron's Limitation

Key Definition

XOR (Exclusive OR): A logical operation that outputs 1 (true) when *exactly one* of the two inputs equals 1, and outputs 0 otherwise.

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: XOR Truth Table

Why It Matters

The XOR problem is historically significant because it demonstrated that single-layer perceptrons cannot solve all classification problems. This limitation, highlighted by Minsky and Papert in 1969, temporarily slowed neural network research for years. Understanding *why* XOR fails helps us understand *why* we need hidden layers.

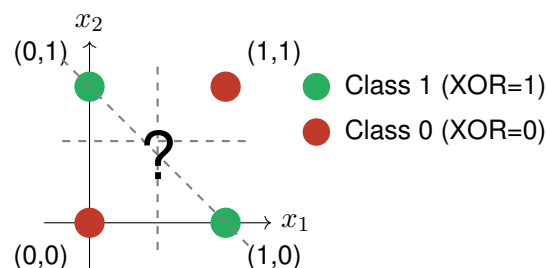


Figure 2: The XOR Problem: No single straight line can separate the green points (class 1) from the red points (class 0). The dashed lines show failed attempts at linear separation.

Common Mistake / Warning

A single Perceptron can only create a **linear decision boundary** (a straight line in 2D, a plane in 3D). Since XOR points cannot be separated by a single line, a single Perceptron **cannot solve the XOR problem**.

1.3 From Biological to Artificial Neural Networks

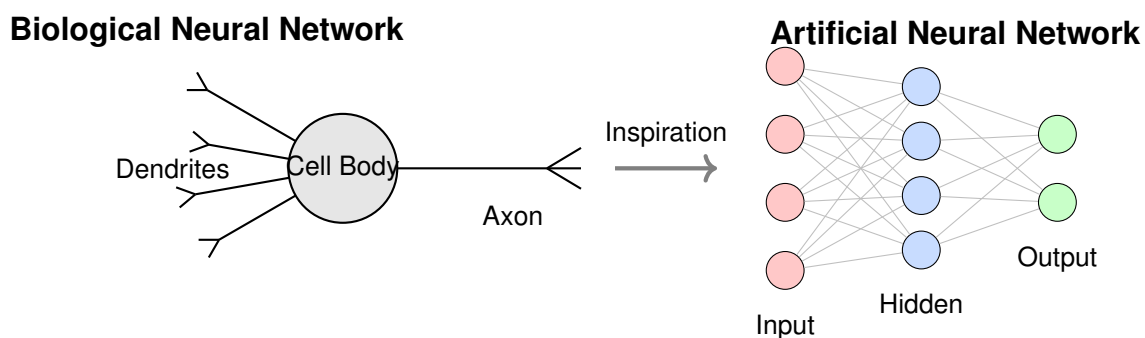


Figure 3: Biological neurons (left) inspired artificial neural networks (right). Just as biological neurons receive signals through dendrites, process them in the cell body, and transmit through axons, artificial neurons receive inputs, compute weighted sums with activation, and produce outputs.

Self-Test Question

Question: Why can't a single Perceptron solve the XOR problem?

Think about your answer before reading below...

Answer: A single Perceptron creates a linear decision boundary (a straight line in 2D). The XOR outputs form a pattern where the two classes (0 and 1) are positioned diagonally opposite to each other. No single straight line can separate these two classes—you would need a curved or multi-part boundary.

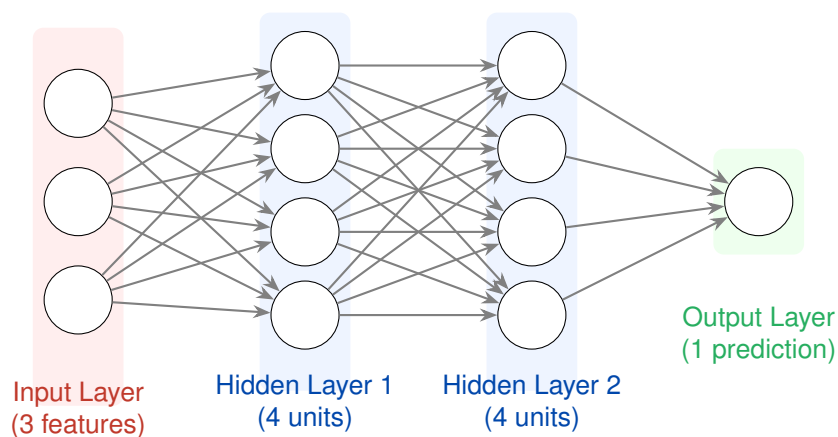
2 Neural Network Architecture: Hidden Layers

2.1 What is a Hidden Layer?

Key Definition

Hidden Layer: A layer of neurons between the input and output layers. It is called “hidden” because we don’t directly observe its values—the learning algorithm decides how to use each hidden unit to transform inputs into useful features for prediction.

Unit (Neuron): A single computational element that takes weighted inputs, sums them, and applies an activation function.



This is a **3-layer** neural network (count hidden layers + output layer)

Figure 4: A Feedforward Neural Network with 3 input features, two hidden layers (4 units each), and 1 output. The network is called “3-layer” because we count the hidden layers plus the output layer.

Analogy / Concrete Example

Think of hidden layers as **feature detectors** in an assembly line:

Input Layer: Raw materials (pixel values, sensor readings)

Hidden Layer 1: First-level inspectors who identify basic patterns (edges, simple shapes)

Hidden Layer 2: Second-level inspectors who combine basic patterns into complex features (eyes, wheels)

Output Layer: Final decision maker (“This is a cat” or “This is a car”)

2.2 Counting Parameters in a Neural Network

Understanding the number of parameters (weights and biases) is crucial for:

- Estimating computational requirements
- Understanding model complexity
- Avoiding overfitting [when a model memorizes training data instead of learning general patterns]

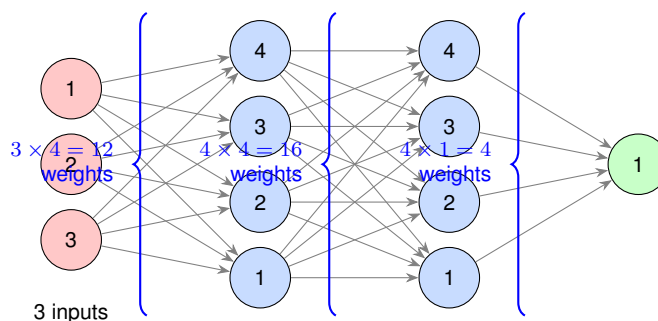


Figure 5: Counting weights in a neural network: multiply the number of neurons in consecutive layers.

Parameter Calculation for the network in Figure 5:

$$\text{Weights: } \underbrace{3 \times 4}_{\text{Input} \rightarrow \text{Hidden 1}} + \underbrace{4 \times 4}_{\text{Hidden 1} \rightarrow \text{Hidden 2}} + \underbrace{4 \times 1}_{\text{Hidden 2} \rightarrow \text{Output}} = 12 + 16 + 4 = 32 \quad (2)$$

$$\text{Biases: } \underbrace{4}_{\text{Hidden 1}} + \underbrace{4}_{\text{Hidden 2}} + \underbrace{1}_{\text{Output}} = 9 \quad (3)$$

$$\text{Total Parameters: } 32 + 9 = \boxed{41} \quad (4)$$

2.3 Fully Connected & Feedforward Networks

Key Definition

Fully Connected (Dense) Network: Each neuron in one layer is connected to *every* neuron in the next layer. Every input influences every output of the subsequent layer.

Feedforward Network: Information flows in one direction only—from input to output—with no loops or cycles. Each layer's output becomes the next layer's input.

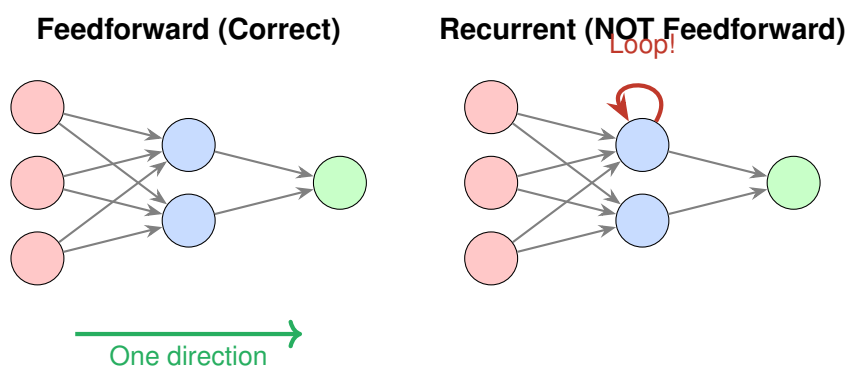


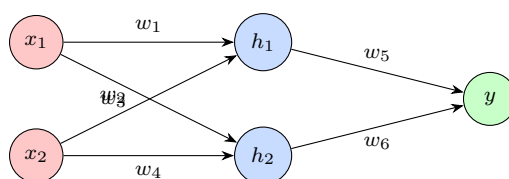
Figure 6: Left: A feedforward network where information flows strictly forward. Right: A recurrent network has loops (information can flow back), which is *not* feedforward. This course focuses on feedforward networks.

2.4 Why Hidden Layers Alone Are NOT Enough

Common Mistake / Warning

Critical Insight: Simply adding hidden layers does **not** automatically enable non-linear modeling. If we only use linear transformations, a chain of linear functions is still linear!

Consider a simple 2-layer network without activation functions:



Let's trace the computation (assuming no activation functions):

$$h_1 = w_1x_1 + w_3x_2 + b_1 \quad (5)$$

$$h_2 = w_2x_1 + w_4x_2 + b_2 \quad (6)$$

$$y = w_5h_1 + w_6h_2 + b_3 \quad (7)$$

Substituting h_1 and h_2 into y :

$$y = w_5(w_1x_1 + w_3x_2 + b_1) + w_6(w_2x_1 + w_4x_2 + b_2) + b_3 \quad (8)$$

$$= \underbrace{(w_1w_5 + w_2w_6)}_{\text{new weight}}x_1 + \underbrace{(w_3w_5 + w_4w_6)}_{\text{new weight}}x_2 + \underbrace{(w_5b_1 + w_6b_2 + b_3)}_{\text{new bias}} \quad (9)$$

Key Observation

The result is still a **linear function** of x_1 and x_2 ! We could achieve the same result with a single layer.

A chain of LINEAR functions at any depth is still a LINEAR function!

This is why we need **non-linear activation functions**.

Self-Test Question

Question: If you stack 100 hidden layers but use no activation functions (or only linear activations), what type of decision boundary can the network create?

Think about your answer...

Answer: Still only a linear (straight line/hyperplane) boundary! The composition of linear functions is always linear, regardless of depth. You need non-linear activation functions to create curved boundaries.

3 Neural Network Architecture: Activation Functions

Key Definition

Activation Function: A non-linear function applied to the weighted sum of inputs at each neuron. It introduces non-linearity, enabling the network to learn complex patterns that linear models cannot capture.

Non-linearity: A function where $f(ax + by) \neq af(x) + bf(y)$. Non-linear functions can create curved decision boundaries.

Why It Matters

Activation functions are the “magic ingredient” that transforms neural networks from fancy linear models into universal function approximators. Without them, deep networks would be no more powerful than single-layer perceptrons.

3.1 The Big Three: Sigmoid, Tanh, and ReLU

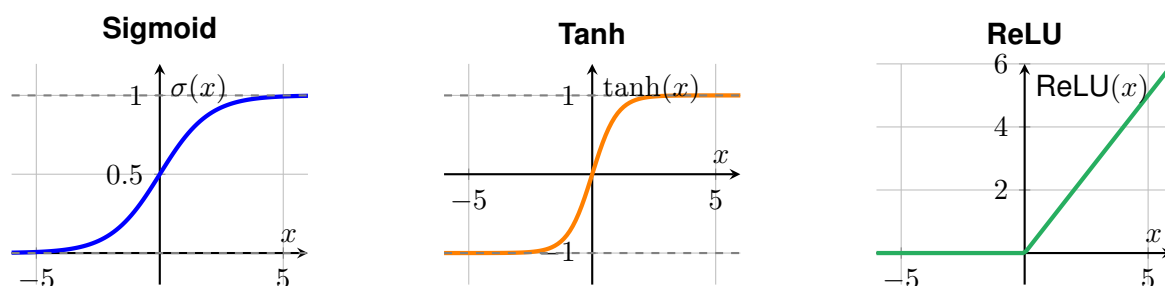


Figure 7: The three most common activation functions: Sigmoid (outputs 0 to 1), Tanh (outputs -1 to 1), and ReLU (outputs 0 or input value).

3.1.1 Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (10)$$

Properties:

- **Output Range:** $(0, 1)$ — useful for probability interpretation
- **Shape:** S-shaped curve (“squashes” any input to be between 0 and 1)
- **Historical Significance:** Mimics biological neuron “firing rate”

Problems with Sigmoid:

1. Vanishing Gradients [saturated neurons “kill” gradients]:

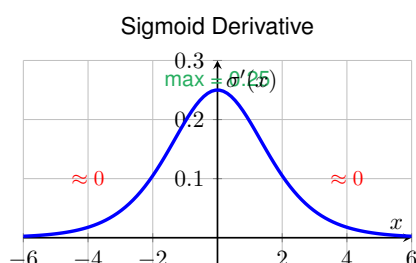


Figure 8: The sigmoid derivative is nearly zero for large positive or negative inputs, causing gradients to vanish during backpropagation.

When x is very large or very small:

- $x = -10 \Rightarrow \sigma(x) \approx 0$ and $\sigma'(x) \approx 0$
- $x = +10 \Rightarrow \sigma(x) \approx 1$ and $\sigma'(x) \approx 0$

Near-zero gradients mean weights barely update \rightarrow learning stops!

2. Not Zero-Centered:

Sigmoid outputs are always positive (between 0 and 1). This causes all gradients on weights to have the same sign, leading to inefficient “zig-zag” updates during training.

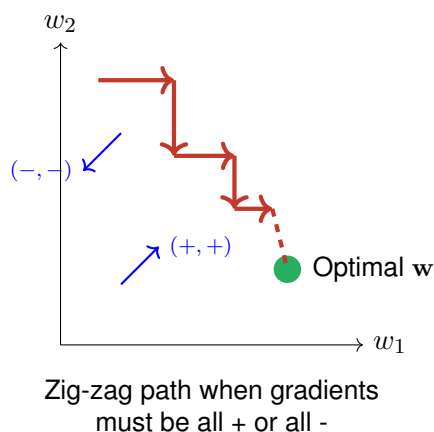


Figure 9: Non-zero-centered outputs force inefficient diagonal-only weight updates.

3. Computationally Expensive:

Computing e^{-x} (exponential) is slower than simpler operations.

3.1.2 Hyperbolic Tangent (Tanh)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1 \quad (11)$$

Properties:

- **Output Range:** $(-1, 1)$
- **Zero-Centered:** ✓ Outputs can be positive or negative
- **Still Saturates:** ✗ Still has vanishing gradient problem at extremes

Tanh vs. Sigmoid: Tanh is essentially a scaled and shifted sigmoid:

$$\tanh(x) = 2\sigma(2x) - 1$$

The zero-centered output makes tanh generally preferable to sigmoid for hidden layers.

3.1.3 Rectified Linear Unit (ReLU)

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad (12)$$

Advantages of ReLU:

- **No Saturation (for $x > 0$):** Gradient is always 1 for positive inputs
- **Computationally Efficient:** Just a comparison and selection—no exponentials!
- **Faster Convergence:** Empirically converges $\sim 6\times$ faster than sigmoid/tanh
- **Biologically Plausible:** More similar to how real neurons fire

The “Dead ReLU” Problem:

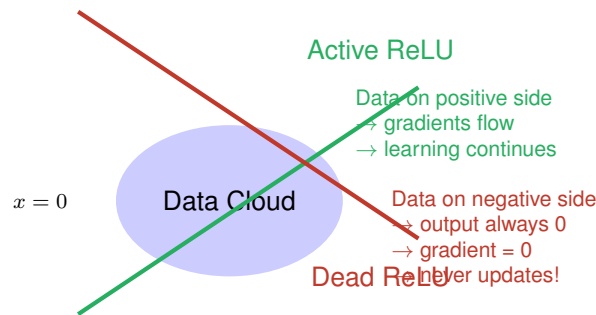


Figure 10: Dead ReLU Problem: If a ReLU unit's inputs are always negative (all data points on the “wrong” side of the boundary), it outputs 0 and receives no gradient. The neuron is effectively “dead” and stops learning.

Common Mistake / Warning

Dead ReLU Problem: If a neuron's weighted input is always negative, ReLU outputs 0, and the gradient is also 0. The neuron never updates and becomes permanently inactive.

Solution: Initialize ReLU neurons with slightly positive biases (e.g., 0.01) to give them a chance to activate initially.

3.2 Summary: Activation Function Comparison

Property	Sigmoid	Tanh	ReLU
Output Range	$(0, 1)$	$(-1, 1)$	$[0, \infty)$
Zero-Centered	No	Yes	No
Vanishing Gradient	Yes (both extremes)	Yes (both extremes)	No (for $x > 0$)
Computational Cost	High (e^x)	High (e^x)	Low (comparison)
Dead Neurons	No	No	Yes (Dead ReLU)
Convergence Speed	Slow	Slow	Fast ($\sim 6x$)

Table 2: Comparison of common activation functions.

3.3 Solving XOR with Non-Linear Activation

Now we can solve the XOR problem using hidden layers with ReLU activation!

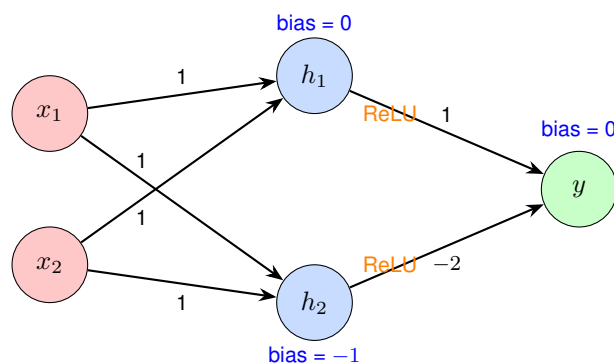


Figure 11: A neural network that solves XOR using ReLU activation.

Step-by-step verification:

For input (0,0):

$$h_1 = \text{ReLU}(0 \cdot 1 + 0 \cdot 1 + 0) = \text{ReLU}(0) = 0 \quad (13)$$

$$h_2 = \text{ReLU}(0 \cdot 1 + 0 \cdot 1 - 1) = \text{ReLU}(-1) = 0 \quad (14)$$

$$y = 0 \cdot 1 + 0 \cdot (-2) + 0 = \boxed{0} \quad \checkmark \quad (15)$$

For input (0,1):

$$h_1 = \text{ReLU}(0 + 1 + 0) = \text{ReLU}(1) = 1 \quad (16)$$

$$h_2 = \text{ReLU}(0 + 1 - 1) = \text{ReLU}(0) = 0 \quad (17)$$

$$y = 1 \cdot 1 + 0 \cdot (-2) + 0 = \boxed{1} \quad \checkmark \quad (18)$$

For input (1,0):

$$h_1 = \text{ReLU}(1 + 0 + 0) = 1 \quad (19)$$

$$h_2 = \text{ReLU}(1 + 0 - 1) = 0 \quad (20)$$

$$y = 1 \cdot 1 + 0 \cdot (-2) = \boxed{1} \quad \checkmark \quad (21)$$

For input (1,1):

$$h_1 = \text{ReLU}(1 + 1 + 0) = 2 \quad (22)$$

$$h_2 = \text{ReLU}(1 + 1 - 1) = 1 \quad (23)$$

$$y = 2 \cdot 1 + 1 \cdot (-2) = 2 - 2 = \boxed{0} \quad \checkmark \quad (24)$$

Success!

The neural network with ReLU activation correctly computes XOR! The hidden layer learns to represent the input in a way that makes the classes linearly separable.

3.4 Effect of Weights and Biases on Activation

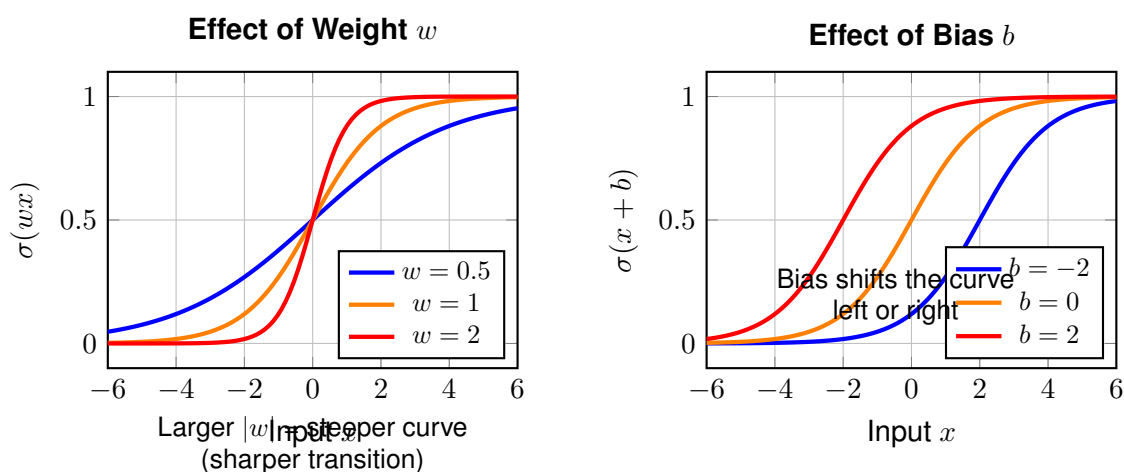


Figure 12: Left: Weights control the steepness of the sigmoid curve. Right: Biases shift the curve horizontally.

Self-Test Question

Question: Why is ReLU generally preferred over sigmoid for hidden layers in modern deep networks?

Think about your answer...

Answer: Three main reasons: (1) ReLU doesn't saturate for positive inputs, avoiding vanishing gradients that slow learning; (2) ReLU is computationally cheaper (just a max operation vs. exponentials); (3) ReLU empirically converges about 6x faster. However, you must watch for "dead ReLU" neurons.

4 Neural Network Architecture: Output Units

Key Definition

Output Unit: The final layer of a neural network that produces the prediction. The choice of output activation function depends on the task type (regression vs. classification).

The output layer activation is chosen based on what kind of prediction you need:

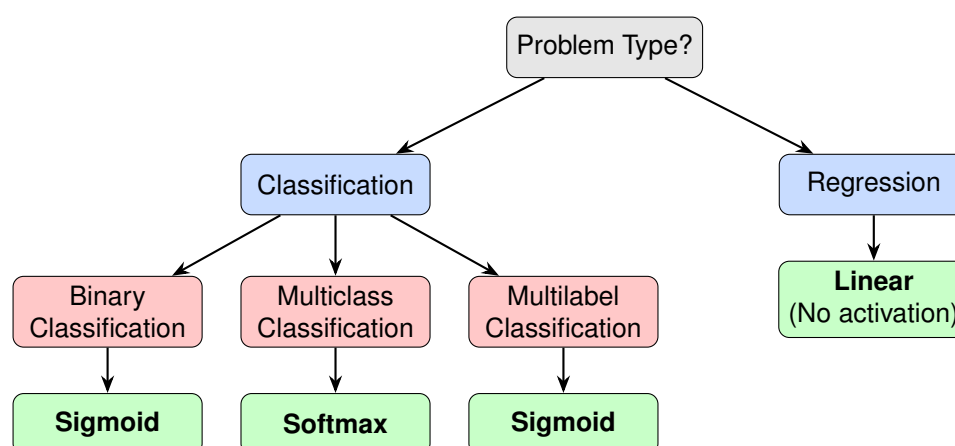


Figure 13: Decision tree for choosing output activation functions based on task type.

4.1 Linear Output (Regression)

Key Definition

Regression: Predicting a continuous numerical value (e.g., house price, temperature, stock value).

For regression, we use **no activation function** (or equivalently, a linear/identity activation):

$$\hat{y} = \sum_i w_i h_i + b \quad (25)$$

The output can be any real number, which is appropriate for continuous predictions.

Linear Activation (Identity)

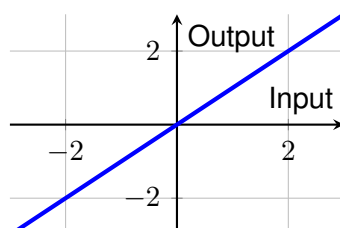


Figure 14: Linear activation: $f(x) = x$. The output equals the input—no transformation.

4.2 Sigmoid Output (Binary Classification)

Key Definition

Binary Classification: Predicting one of two classes (e.g., spam/not spam, cat/dog, positive/negative).

For binary classification, we use **sigmoid** to produce a probability:

$$P(\text{class} = 1|\mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (26)$$

Decision Rule:

- If $\sigma(z) \geq 0.5$, predict class 1
- If $\sigma(z) < 0.5$, predict class 0

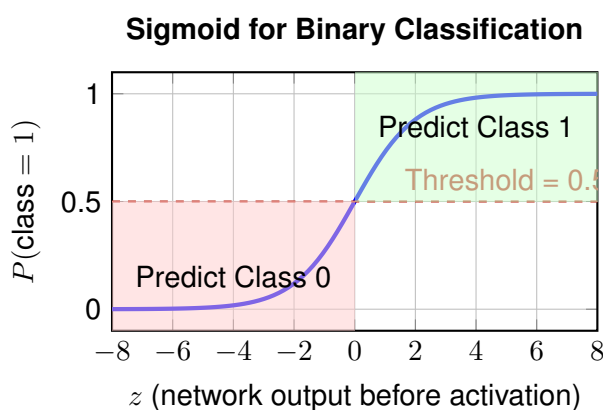


Figure 15: Sigmoid activation for binary classification. Values above 0.5 are classified as class 1.

4.3 Softmax Output (Multiclass Classification)

Key Definition

Multiclass Classification: Predicting one of K mutually exclusive classes (e.g., digit recognition 0-9, animal species classification).

Softmax Function: Converts a vector of K raw scores into a probability distribution where all values are positive and sum to 1.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, \dots, K \quad (27)$$

Why It Matters

Why use e^z (exponential)?

1. **Makes all values positive:** Even negative scores become positive after e^z
2. **Preserves ordering:** If $z_1 > z_2$, then $e^{z_1} > e^{z_2}$
3. **Amplifies differences:** Larger scores become much larger (exponentially)
4. **Simplifies math:** Using e makes the gradient calculations cleaner during training

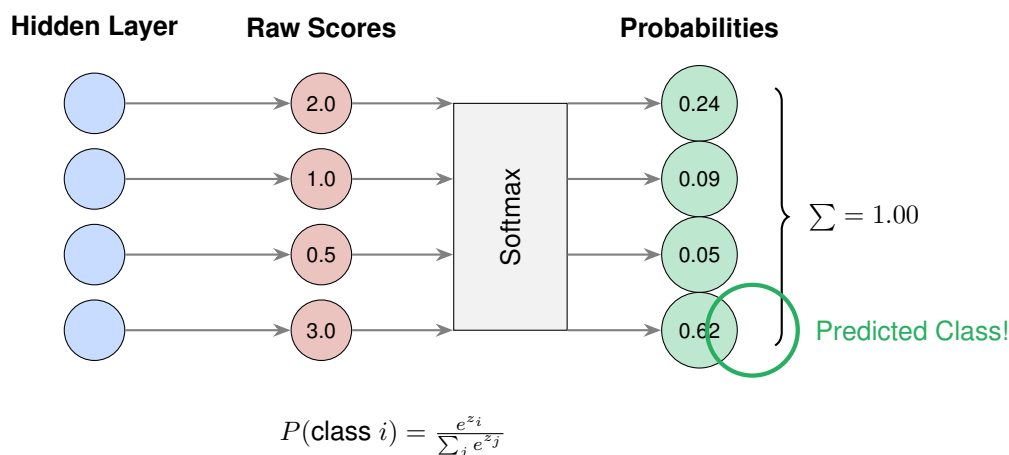


Figure 16: Softmax converts raw scores into probabilities that sum to 1. The class with the highest probability is the prediction.

Softmax Example:

Given raw scores $\mathbf{z} = [2.0, 1.0, 0.5, 3.0]$:

$$e^{z_1} = e^{2.0} = 7.39 \quad (28)$$

$$e^{z_2} = e^{1.0} = 2.72 \quad (29)$$

$$e^{z_3} = e^{0.5} = 1.65 \quad (30)$$

$$e^{z_4} = e^{3.0} = 20.09 \quad (31)$$

$$\sum_j e^{z_j} = 7.39 + 2.72 + 1.65 + 20.09 = 31.85 \quad (32)$$

$$P(\text{class 1}) = \frac{7.39}{31.85} = 0.23 \quad (33)$$

$$P(\text{class 2}) = \frac{2.72}{31.85} = 0.09 \quad (34)$$

$$P(\text{class 3}) = \frac{1.65}{31.85} = 0.05 \quad (35)$$

$$P(\text{class 4}) = \frac{20.09}{31.85} = \boxed{0.63} \quad \leftarrow \text{Highest! Predicted class.} \quad (36)$$

One-Hot Encoding

The ideal prediction is a **one-hot vector** [a vector with exactly one “1” and the rest “0”s]. For 4 classes where the true class is 4:

$$\mathbf{y}_{\text{true}} = [0, 0, 0, 1]$$

The softmax output $[0.23, 0.09, 0.05, 0.63]$ approximates this. Training adjusts weights to make the predicted probability closer to 1 for the correct class.

4.4 Summary: Output Activation by Task

Task	Output Activation	Output Range	Example
Regression	Linear (none)	$(-\infty, +\infty)$	House prices
Binary Classification	Sigmoid	$(0, 1)$	Spam detection
Multiclass Classification	Softmax	$(0, 1)^K$, sum=1	Digit recognition
Multilabel Classification	Sigmoid (each output)	$(0, 1)^K$	Image tagging

Table 3: Summary of output activation functions for different tasks.

Self-Test Question

Question: You're building a neural network to classify images into one of 10 categories (cat, dog, bird, ...). What activation function should you use for the output layer? Why?

Think about your answer...

Answer: Use **Softmax**. This is multiclass classification with mutually exclusive classes (each image is exactly one animal). Softmax converts the 10 raw output scores into 10 probabilities that sum to 1, and the class with the highest probability becomes the prediction.

5 Putting It All Together

Section Summary

Key Concepts from This Lecture:

1. Why Neural Networks?

- Single perceptrons can only create linear boundaries
- Real-world problems (like XOR) require non-linear boundaries
- Neural networks with hidden layers and activation functions can model any function

2. Hidden Layers:

- Transform raw inputs into useful features
- More layers = more complex transformations possible
- Parameters = weights + biases at each connection

3. Activation Functions:

- Introduce non-linearity (essential for deep networks)
- Sigmoid: $(0, 1)$, historical but has vanishing gradient
- Tanh: $(-1, 1)$, zero-centered but still saturates
- ReLU: $\max(0, x)$, fast and effective, watch for dead neurons

4. Output Units:

- Regression \rightarrow Linear (no activation)
- Binary classification \rightarrow Sigmoid
- Multiclass classification \rightarrow Softmax

Complete Feedforward Neural Network

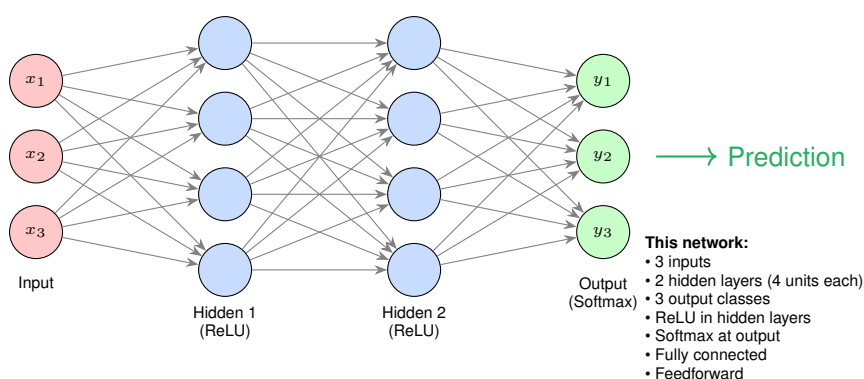


Figure 17: A complete feedforward neural network for 3-class classification with all components discussed in this lecture.

Review Questions for Self-Testing

Test your understanding by answering these questions without looking back:

1. What is the XOR problem, and why can't a single perceptron solve it?
2. What does "fully connected" mean in the context of neural networks?

3. Why is stacking linear layers (without activation functions) insufficient for modeling non-linear functions?
4. List three problems with using sigmoid activation in hidden layers.
5. What is the “dead ReLU” problem and how can it be mitigated?
6. Why is tanh often preferred over sigmoid for hidden layers?
7. When would you use softmax vs. sigmoid at the output layer?
8. Calculate the total number of parameters (weights + biases) in a network with: 5 inputs, one hidden layer with 10 units, and 3 outputs.
9. What property must the outputs of a softmax function satisfy?
10. Why does the softmax function use exponentials (e^{z_i})?

Spaced Repetition Reminder

For optimal retention, review these notes:

- Tomorrow (Day 1)
- In one week (Day 7)
- In one month (Day 30)

Research shows that spaced repetition significantly improves long-term memory compared to cramming!