Session 2

Linear Models and Their Limitations

Deep Learning course - SKEMA 2025

Mastère Spécialisé® Chef de Projet Intelligence Artificielle

Salem Lahlou

Some figures adapted from 3Blue1Brown (YouTube)

(Reminder) Understanding Linear Models

What is a linear model?

- The simplest form of machine learning
- Finds a straight line (or hyperplane) that best fits the data
- Makes predictions based on this line

Mathematical form:

- $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$
- Also written using the dot product: $\hat{y} = \langle \beta_{1:p}, x \rangle + \beta_0 = \beta_{1:p}^T x + \beta_0$
- A more concise notation requires adding a "fixed feature" to the vector x: if x is of dimension p, then $x' = \begin{pmatrix} x \\ 1 \end{pmatrix}$ is of dimension $p+1$, and $\hat{y} = \beta^T x'$.
- Also written as: $\hat{Y} = X\beta$ for n examples gathered in the same matrix X

(Reminder) Linear Regression: Mathematical Formulation

For a dataset with features X and target Y:

$$Y = X\beta + \epsilon$$

Where:

- X is the feature matrix (each row is a sample, each column a feature)
- β is the coefficient vector (parameters to learn)
- ϵ is the error term (noise)
- Y is the target vector

(Reminder) Linear Regression: Loss Function

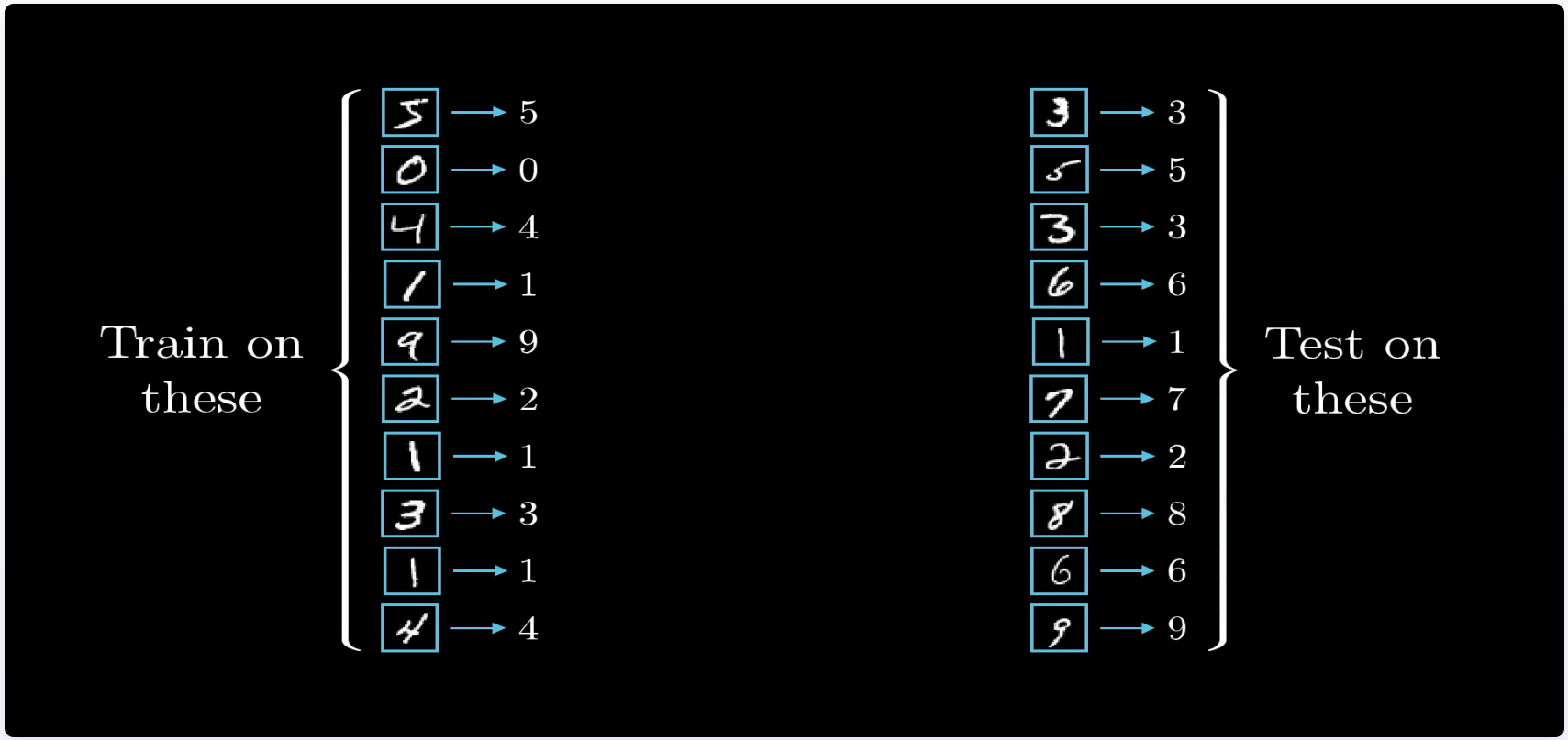
Objective: Find β that minimizes the sum of squared errors:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2$$

Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2$$

(Reminder) Train/Val/Test split



Closed-Form Solution for Linear Regression

Linear regression has a direct analytical solution:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Where:

- X^T is the transpose of X
- $(X^T X)^{-1}$ is the inverse of $X^T X$

Advantages:

- No iterative optimization needed
- Guaranteed to find global minimum
- Computationally efficient for small datasets

Gradient Descent for Linear Regression

When datasets are large, we can use gradient descent:

Compute the gradient:

$$\nabla_\beta \text{MSE} = -\frac{2}{n} X^T (y - X\beta)$$

Update rule:

$$\beta_{new} = \beta_{old} - \alpha \nabla_\beta \text{MSE}$$

$$\beta_{new} = \beta_{old} + \frac{2\alpha}{n} X^T (y - X\beta_{old})$$

Logistic Regression for Classification

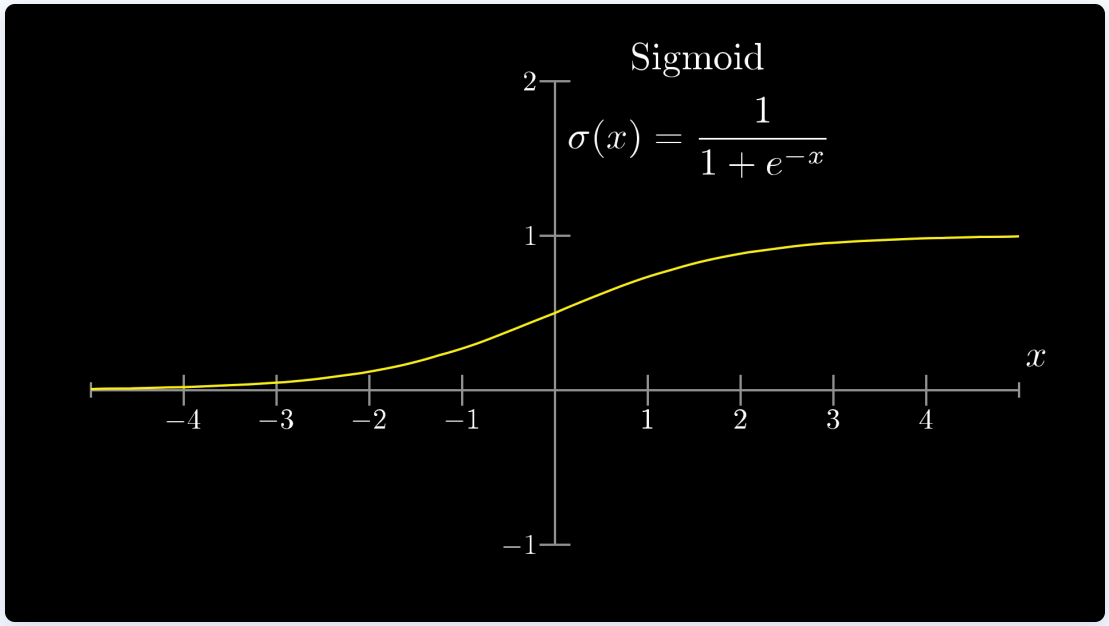
For binary classification, we model the probability of class 1:

$$P(y = 1|x) = \sigma(\beta^T x) = \frac{1}{1 + e^{-\beta^T x}}$$

Where:

- σ is the sigmoid function
- $\beta^T x$ is the linear predictor

Decision boundary: $\beta^T x = 0$ (where $P(y = 1|X) = 0.5$)



Logistic Regression: Loss Function

Loss function: Negative log-likelihood (Cross-entropy)

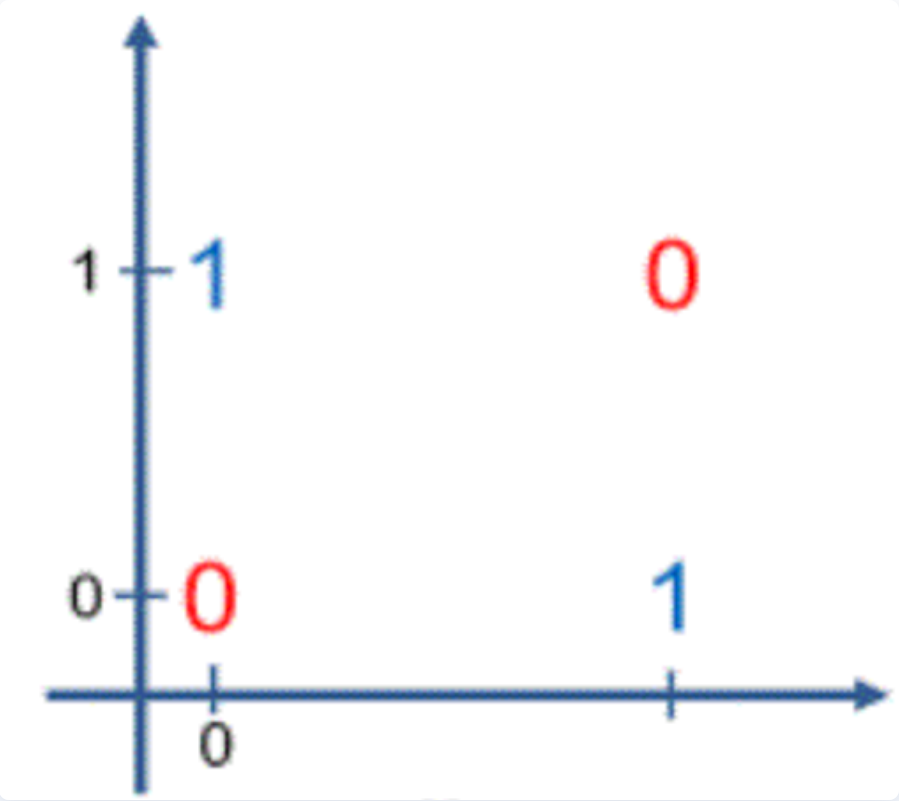
$$L(\beta) = -\sum_{i=1}^{n} [y_i \log(\sigma(\beta^T x_i)) + (1 - y_i) \log(1 - \sigma(\beta^T x_i))]$$

Interpretation:

- Penalizes confident incorrect predictions heavily
- Encourages confident correct predictions

The XOR Problem

A famous limitation of linear models:



Source: github Lavanyajoyce

- No single straight line can separate these points correctly
- This is a simple example of a non-linearly separable problem

Mathematical Proof of XOR Limitation

For a linear model to solve XOR, we need:

- $\beta_0 + \beta_1 \cdot 0 + \beta_2 \cdot 0 < 0$ (for point (0,0))
- $\beta_0 + \beta_1 \cdot 0 + \beta_2 \cdot 1 > 0$ (for point (0,1))
- $\beta_0 + \beta_1 \cdot 1 + \beta_2 \cdot 0 > 0$ (for point (1,0))
- $\beta_0 + \beta_1 \cdot 1 + \beta_2 \cdot 1 < 0$ (for point (1,1))

From the first equation: $\beta_0 < 0$
From the second equation: $\beta_0 + \beta_2 > 0$, so $\beta_2 > -\beta_0 > 0$
From the third equation: $\beta_0 + \beta_1 > 0$, so $\beta_1 > -\beta_0 > 0$
From the fourth equation: $\beta_0 + \beta_1 + \beta_2 < 0$

But if $\beta_1 > 0$ and $\beta_2 > 0$, then $\beta_0 + \beta_1 + \beta_2 > \beta_0$, which contradicts the fourth equation.

Beyond XOR: Real-World Limitations

Linear models struggle with:

- Complex relationships
- Interactions between features
- Non-linear patterns (most real-world data)

Interactive demonstration:

- [Instructor demonstration of interactive app]
- Observe how linear models fail on complex data patterns
- Try different datasets and see the limitations

Feature Engineering for Linear Models

To handle non-linearity with linear models, we can:

1. **Polynomial features:** Add powers of features
   - $x_1, x_1^2, x_1^3, \ldots$
2. **Interaction terms:** Add products of features
   - $x_1 \times x_2, x_1 \times x_3, \ldots$
3. **Basis functions:** Apply transformations
   - $\sin(x), \log(x), \exp(x), \ldots$

Limitations of Feature Engineering

Problems with manual feature engineering:

- Requires domain knowledge
- Prone to overfitting with too many features
- Doesn't scale well to high-dimensional data
- Need to decide which transformations to use beforehand

We need a more flexible approach!

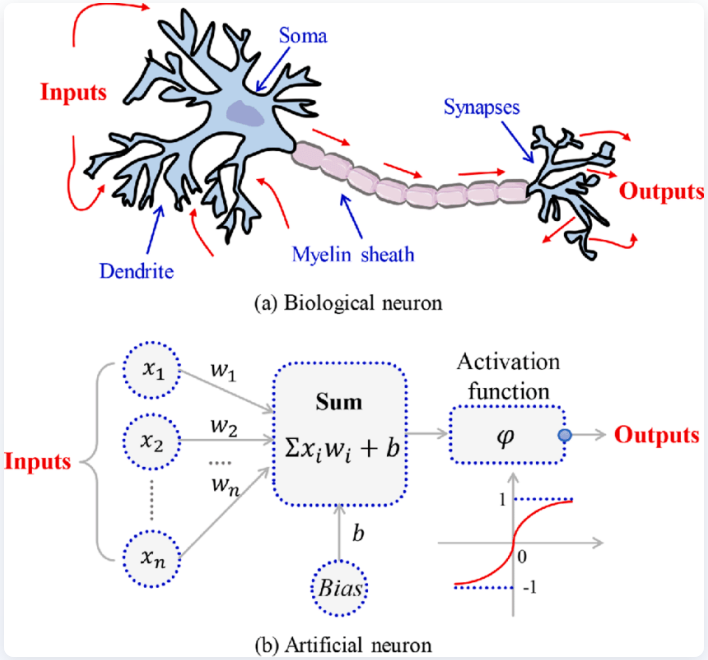
The Need for Non-Linearity

To solve complex problems, we need:

- Models that can create non-linear decision boundaries
- Ability to capture interactions between features
- Flexibility to represent complex patterns

Enter neural networks...

The Neuron Analogy



Source: Xianlin Wang

Biological Neuron:

- Receives signals from other neurons
- Processes these signals
- Fires if signals exceed a threshold

Artificial Neuron:

- Receives input values
- Applies weights to inputs
- Processes through an activation function
- Outputs a result

Mathematical Model of a Neuron

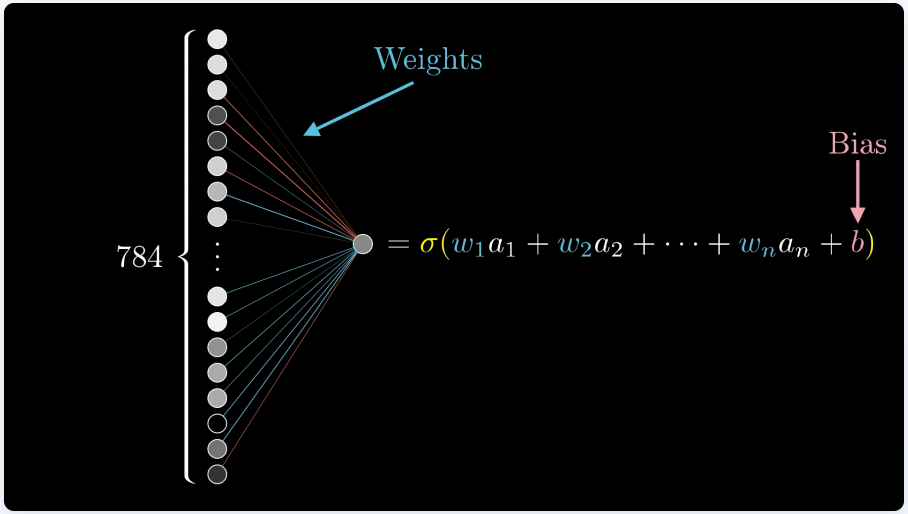
Artificial neuron computation:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$a = \sigma(z)$$

Where:

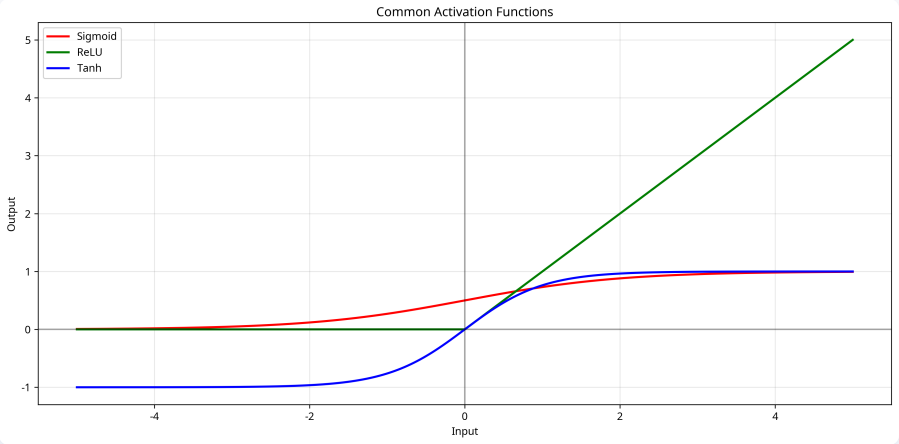
- x_i are the inputs
- w_i are the weights
- b is the bias term
- z is the weighted sum
- σ is the activation function
- a is the output (activation)



Activation Functions

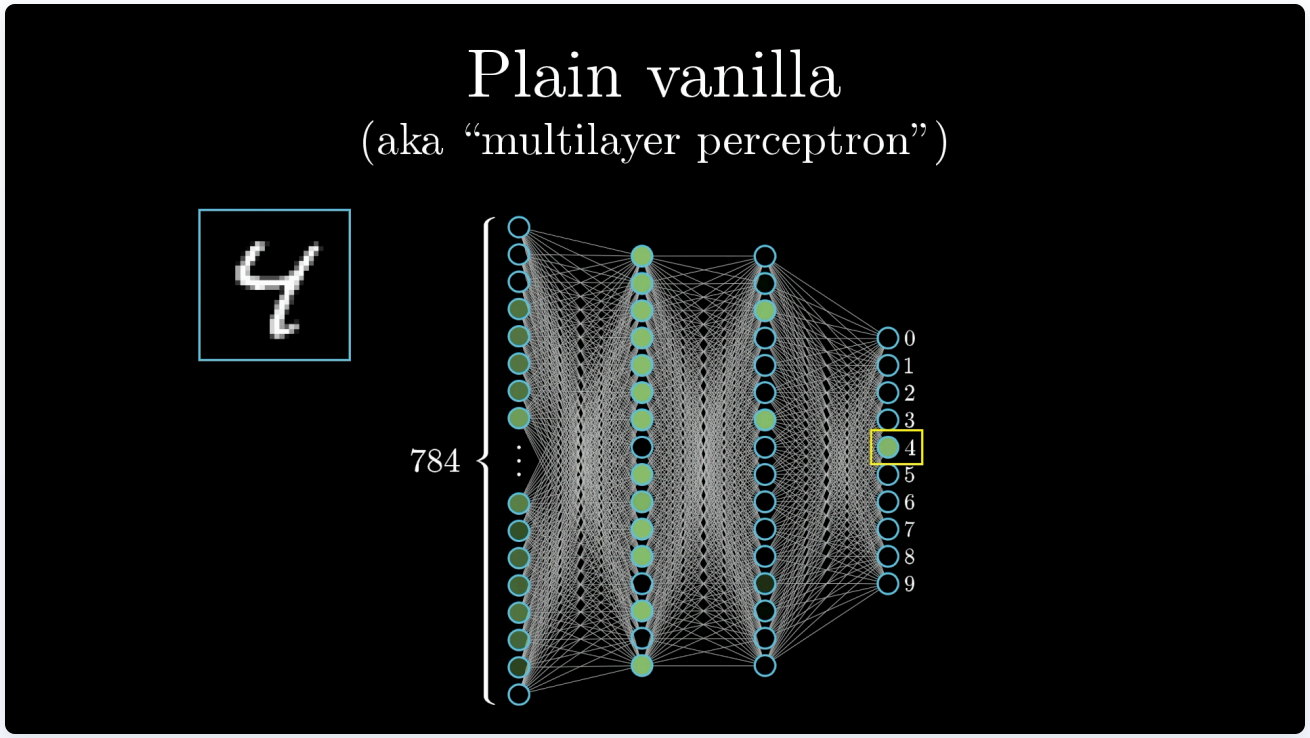
Purpose: Introduce non-linearity into the network

Common activation functions:



1. **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$
   - Range: (0, 1)
2. **Hyperbolic Tangent (tanh):** $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
   - Range: (-1, 1)
3. **Rectified Linear Unit (ReLU):** $\mathrm{ReLU}(z) = \max(0, z)$
   - Range: [0, ∞)

Building Neural Networks



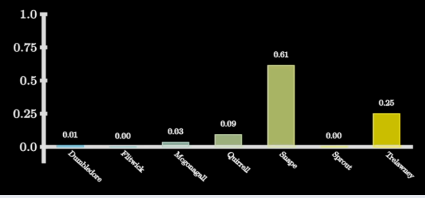
Components:

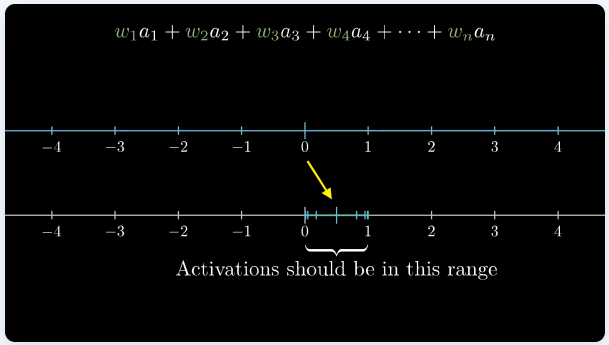
- **Input Layer:** Features from our data
- **Hidden Layer(s):** Where the "learning" happens
- **Output Layer:** Predictions or classifications
- **Weights:** Strength of connections between neurons
- **Activation Functions:** Introduce non-linearity

The need for an output layer

$$\begin{bmatrix} -0.04 & -0.04 & +0.01 & -0.03 & +0.00 & -0.01 & -0.04 \\ -2.99 & -2.88 & -3.35 & +2.32 & +1.94 & +2.58 & +3.34 \\ +0.12 & -0.02 & +0.12 & -0.16 & +0.06 & -0.15 & +0.19 \\ -0.01 & +0.06 & +0.16 & -0.18 & +0.03 & -0.05 & +0.32 \\ +0.10 & +0.09 & +0.10 & +0.37 & +0.15 & -0.12 & -0.05 \\ +0.10 & -0.22 & +0.09 & +0.09 & -0.15 & -0.19 & -0.09 \\ +0.11 & -0.06 & +0.05 & -0.41 & +0.33 & +0.24 & +0.28 \end{bmatrix} \begin{bmatrix} 5.4 \\ 7.1 \\ 6.0 \\ 5.4 \\ 4.2 \\ 6.4 \\ 4.3 \end{bmatrix} = \begin{bmatrix} -0.8 \\ -5.0 \\ +0.5 \\ +1.5 \\ +3.4 \\ -2.3 \\ +2.5 \end{bmatrix}$$

Not at all a probability distribution!





Classification vs Regression Outputs

Regression outputs:

- Direct values from final layer (often linear activation)
- Predict continuous values
- Loss functions: MSE, MAE, Huber loss

Binary classification outputs:

- Single node with sigmoid activation
- Output interpreted as probability of positive class
- Loss function: Binary cross-entropy

Multi-class classification outputs:

- Multiple nodes (one per class) followed by softmax
- Outputs form a probability distribution over classes
- Loss function: Categorical cross-entropy

Logits, Softmax, and Sigmoid Activation

Logits:

- Raw output values from a neural network before activation with K outputs
- Unbounded values that represent unnormalized predictions
- Must be transformed into probabilities for classification

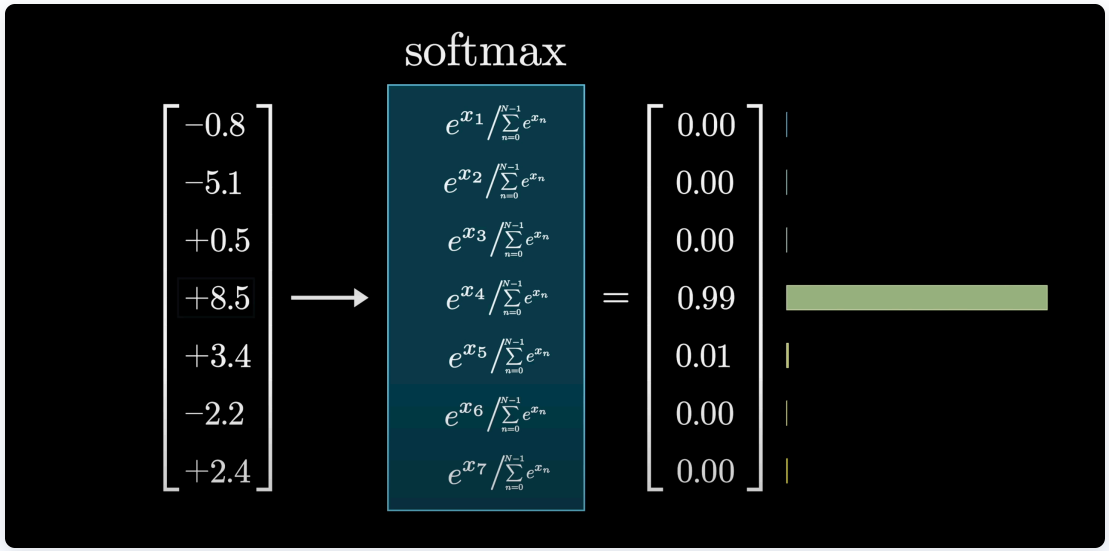
Sigmoid function:

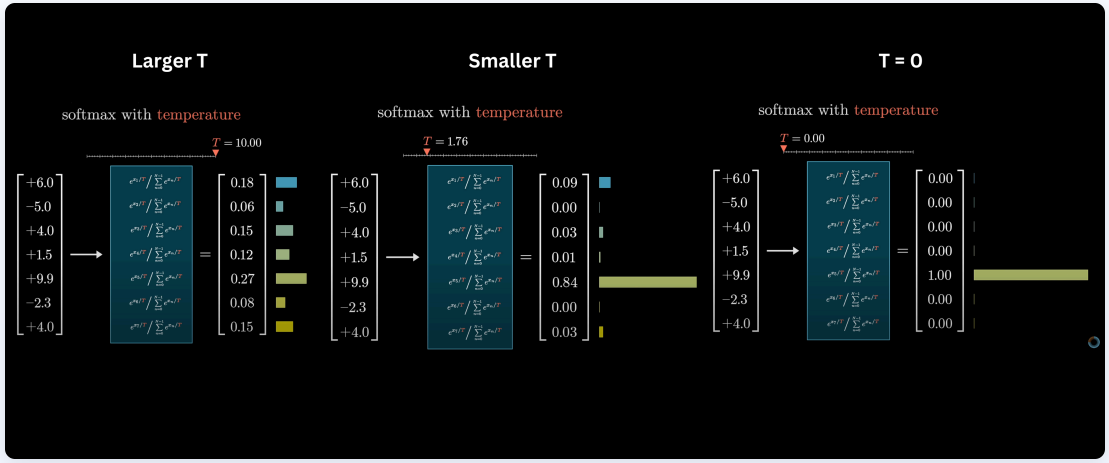
- Transforms a single value into a probability (0 to 1)
- $\sigma(z) = \frac{1}{1+e^{-z}}$
- Use case: Binary classification

Softmax function:

- Transforms a vector of values into a probability distribution
- $\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
- Use case: Multi-class classification
- Properties: Outputs sum to 1, enhances largest input

Softmax function





(Reminder) Detailed Examination of Loss Functions

Mean Squared Error (MSE):

- $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Penalizes larger errors more severely
- Sensitive to outliers

Mean Absolute Error (MAE):

- $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- More robust to outliers than MSE
- Constant gradient magnitude regardless of error size

Binary Cross-Entropy:

- $\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$
- Approaches infinity as prediction approaches wrong answer with high confidence
- Perfect for binary classification with sigmoid outputs

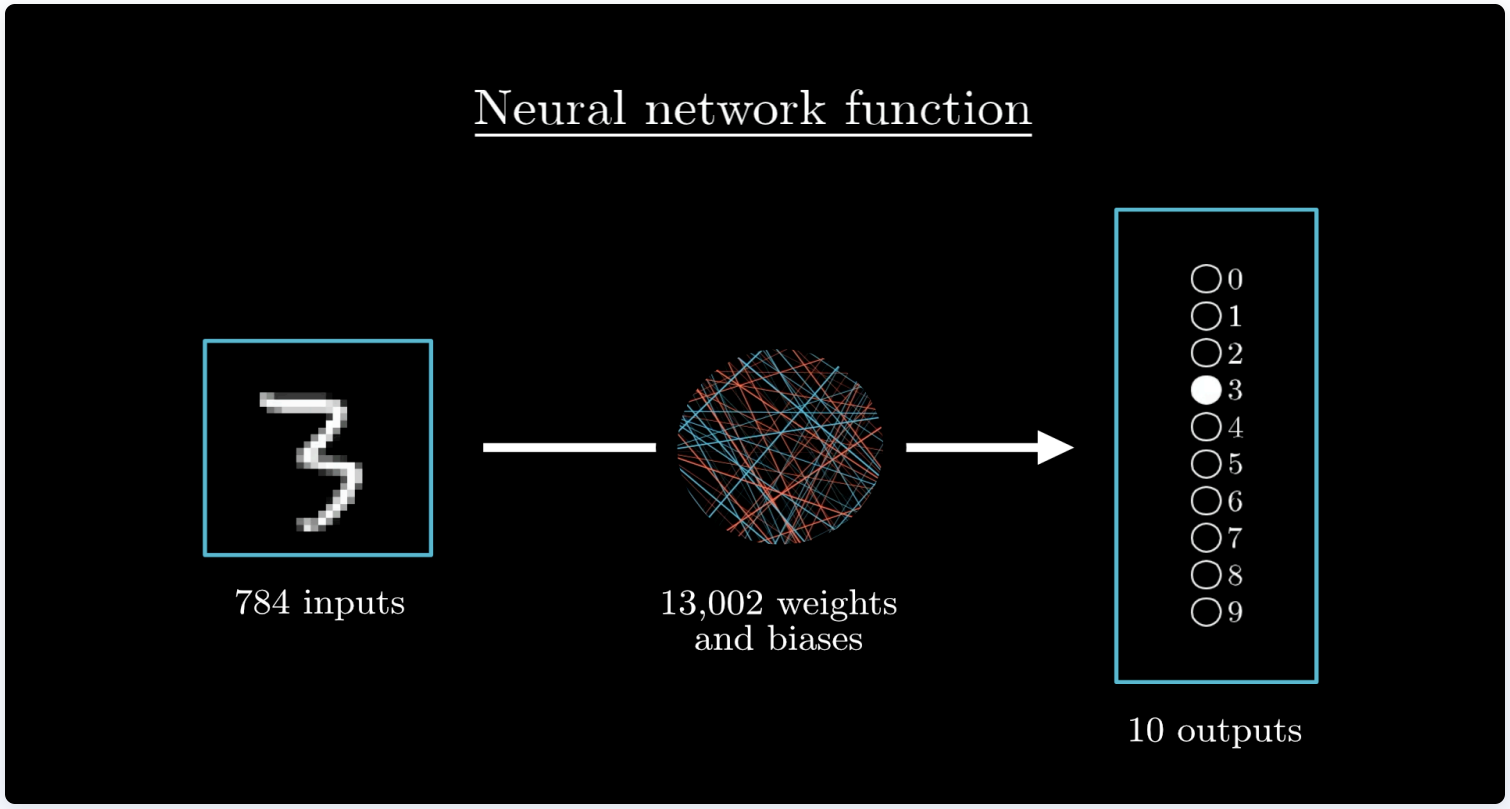
Universal Approximation Theorem

Key insight: A neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$, under mild assumptions on the **non-linear** activation function.

Implications:

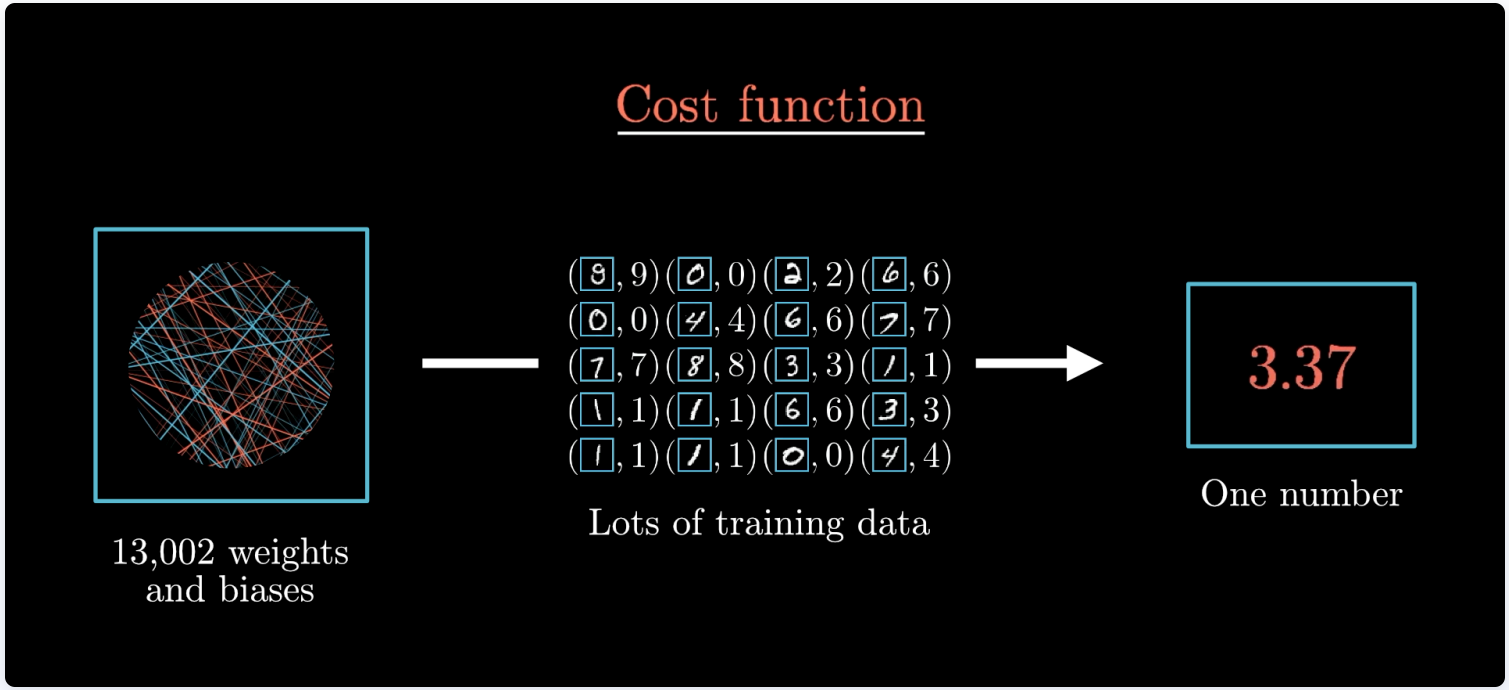
- Neural networks are theoretically capable of learning any function
- We don't need to manually choose the right function form
- The network will learn the appropriate function from data

A neural network is simply a big parametric model



Training a neural network requires tuning its parameters

Using a loss/cost function, as usual, then by doing gradient descent



Next, we will see how to evaluate $\nabla_\theta J$, where θ is a big vector containing all parameters of the neural network.

Key Takeaways

1. Linear models (Regression, Logistic) are simple and interpretable but limited to linear decision boundaries, failing on problems like XOR.
2. Manual feature engineering can extend linear models but requires domain expertise, is hard to scale, and may not capture complex interactions.
3. Artificial neurons compute a weighted sum of inputs plus a bias, followed by a non-linear activation function (e.g., Sigmoid, Tanh, ReLU) to introduce complexity.
4. Neural networks stack layers of neurons, enabling them to automatically learn hierarchical features and model complex, non-linear relationships in data.
5. The choice of output layer activation (e.g., Linear, Sigmoid, Softmax) and loss function (e.g., MSE, Binary/Categorical Cross-Entropy) is crucial and depends on the task (regression, binary/multi-class classification).
6. The Universal Approximation Theorem theoretically guarantees that neural networks are powerful function approximators.
7. Training involves finding optimal network parameters (weights/biases) by minimizing the chosen loss function, typically using gradient-based optimization.

Answers to your free comments

1. > Spend sometime again to explain bias-variance tradeoff

(Next slide)

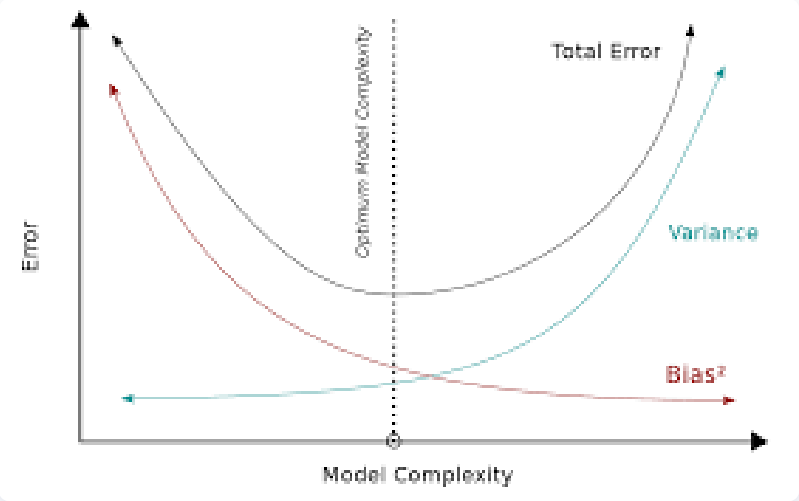
2. > The content is very interesting, but taking in everything at once can feel overwhelming. It's a lot of information to process in one go. I suggest adding a few short pauses between sections to make it easier to absorb.

- **You are not supposed to absorb all concepts.**
- Please take notes (handwritten, ideally), of the concepts that are new to you. e.g.:
  - If you had never heard of "gradient" before, you could note
    - The gradient is a vector that gives us a direction of increase of a function
    - They help for the gradient descent algorithm
    - There are some efficient ways of calculating them. I don't need to bother with the details for now
- **Interrupt me and ask me to re-explain some part if you feel it's interesting to you but I was too fast**

3. > rick ross

?

Understanding the Bias-Variance Tradeoff



Source: Wikipedia

When we build a model $\hat{f}$ to predict a target y based on features x, the expected squared error of our predictions can be decomposed.

Expected Prediction Error (MSE):

$$E[(y - \hat{f}(x))^2]$$

Decomposition (ignoring irreducible noise):

$$E[(y - \hat{f}(x))^2] \approx \underbrace{(E[\hat{f}(x)] - y)^2}_{\text{Bias}^2} + \underbrace{E[(\hat{f}(x) - E[\hat{f}(x)])^2]}_{\text{Variance}}$$

- **Bias:** Error from wrong assumptions. High bias leads to underfitting (oversimplification).
- **Variance:** Error from sensitivity to small fluctuations in the training set. High variance leads to overfitting (overly complex).

The Bias-Variance Tradeoff

Mathematical decomposition of error:

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Bias:

- Error from oversimplified models (underfitting)
- High bias = systematic error, missing important patterns
- Example: Linear model trying to fit non-linear data

Variance:

- Error from sensitivity to small fluctuations (overfitting)
- High variance = model changes drastically with small data changes
- Example: High-degree polynomial fitting to noisy data

Goal: Find optimal balance to minimize total error

Quiz time

Go to the course homepage: https://la7.lu/sk25

Click on "Quiz 2"