# MA477: Data Science

**Lesson 24-25 Outline — 13 / 17 March 2026**

United States Military Academy, West Point
Instructor: MAJ Patrick Kuiper

---

# 1 Administrative

- Calendar review
- Student review
- Mentimeter
- NN Discussion 1 / 2
- Coding Exercise

# 2 Neural Network Lesson Objectives

- Understand the structure and notation of a multilayer neural network.
- Given inputs and the weights/biases of a multilayer neural network, calculate the predicted outputs for the network.
- Understand how the cost function and gradient descent are used to update the weights and biases through back propagation.
- Use functions in Python's keras module to fit multilayer neural networks.

# 3 Neural Networks and Deep Learning

## Overview

- A neural network is a model that learns a nonlinear function

$$f(X)$$

  to predict a response $Y$ from input features

$$X = (X_1, X_2, \ldots, X_p).$$

- Unlike linear regression or logistic regression, a neural network can automatically learn nonlinear transformations of the original predictors.
- Unlike decision trees and boosting, which create nonlinearity through recursive partitioning or ensembles, neural networks create nonlinearity through learned hidden-layer transformations.
- A useful way to think about a neural network is:

  it first builds new features from the inputs, and then uses those learned features to make a prediction.

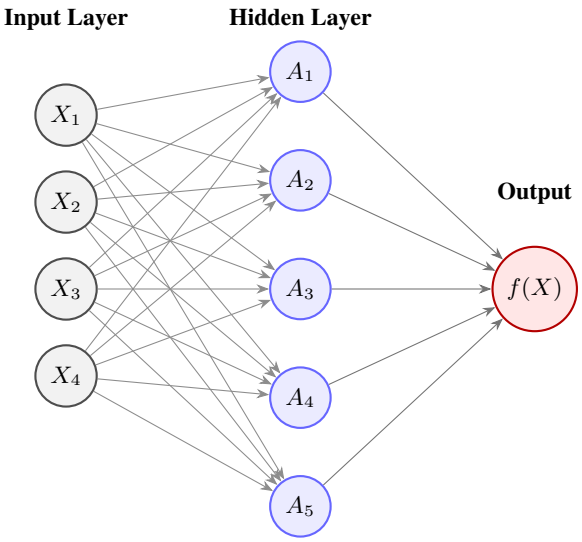


Figure 1: A feed-forward neural network with one hidden layer.

## Single Hidden Layer Neural Network

A simple feed-forward neural network has three parts:

- an **input layer**, containing the original features,
- one or more **hidden layers**, containing learned intermediate features,
- an **output layer**, producing the final prediction.

Figure 1 shows a small neural network with one hidden layer.
Each hidden unit computes a weighted sum of the inputs, followed by a nonlinear activation:

$$z_k = w_{k0} + \sum_{j=1}^{p} w_{kj} X_j, \qquad A_k = g(z_k).$$

The final output is then modeled as

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k.$$

## Activation Functions

$$\text{Sigmoid:} \qquad g(z) = \frac{1}{1+e^{-z}}$$

$$\text{ReLU:} \qquad g(z) = \begin{cases} 0, & z < 0, \\ z, & z \geq 0. \end{cases}$$

## Why Nonlinearity Matters

If the activation function were removed the model collapses into a linear model in the inputs.
Thus neural networks require nonlinear activation functions to capture complex relationships.

## Backpropagation: A Simple Worked Example

To train a neural network, we must compute how the loss changes with respect to each weight. This is done using **backpropagation**, which repeatedly applies the chain rule.

Consider a very small neural network:

$$X \to A \to \hat{y}$$

with

$$A = g(z), \qquad z = wX$$

and prediction

$$\hat{y} = \beta A$$

Suppose we use squared error loss

$$L = (y - \hat{y})^2.$$

**Forward Pass**

$$z = wX$$

$$A = g(z)$$

$$\hat{y} = \beta A$$

**Backward Pass**

First compute

$$\frac{\partial L}{\partial \hat{y}} = -2(y - \hat{y})$$

Then propagate through the network:

$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \beta} = -2(y - \hat{y})A$$

Next compute

$$\frac{\partial L}{\partial A} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial A}$$

Since

$$\hat{y} = \beta A$$

we have

$$\frac{\partial \hat{y}}{\partial A} = \beta$$

so

$$\frac{\partial L}{\partial A} = \frac{\partial L}{\partial \hat{y}} \cdot \beta$$

Finally propagate to the input weight

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial A} \cdot \frac{\partial A}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Since

$$A = g(z)$$

and

$$z = wX$$

we have

$$\frac{\partial A}{\partial z} = g'(z) \qquad \frac{\partial z}{\partial w} = X$$

Thus

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial A} \cdot g'(z) \cdot X$$

Substituting

$$\frac{\partial L}{\partial A} = -2(y - \hat{y})\beta$$

gives

$$\frac{\partial L}{\partial w} = -2(y - \hat{y})\beta g'(z)X.$$

Weights are updated using gradient descent. For each parameter in the network we update in the direction that reduces the loss.

For the hidden layer weights

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

For the output layer weights

$$\beta_{\text{new}} = \beta - \eta \frac{\partial L}{\partial \beta}$$

where $\eta$ is the learning rate. Each update moves the parameter in the direction that decreases the loss, and this process is repeated over many iterations during training.

### Simple Numerical Example

Suppose

$$X = 2, \quad w = 0.5, \quad \beta = 1, \quad y = 1$$

and activation $g(z) = z$.

Forward pass:

$$z = 0.5 \times 2 = 1$$

$$A = 1$$

$$\hat{y} = 1$$

Loss:

$$L = (1 - 1)^2 = 0$$

If the prediction had instead been

$$\hat{y} = 0.8$$

then

$$\frac{\partial L}{\partial w} = -2(1 - 0.8)(1)(1)(2) = -0.8$$

so the weight update becomes

$$w_{new} = w + 0.8\eta.$$

# 4 *Exercise

1. Compute the forward pass values $z, A, \hat{y}$.
2. Compute the loss $L$.
3. Compute the gradients
$$\frac{\partial L}{\partial \beta}, \quad \frac{\partial L}{\partial w}, \quad \frac{\partial L}{\partial b}.$$
4. Perform one gradient descent update with learning rate
$$\eta = 0.1.$$

**Solutions**

1. **Forward Pass**

$$z = wX + b = (0.4)(3) + 0.2 = 1.4$$

Since $z > 0$,

$$A = \text{ReLU}(z) = 1.4$$

$$\hat{y} = \beta A = (2)(1.4) = 2.8$$

2. **Loss**

$$L = (y - \hat{y})^2 = (2 - 2.8)^2 = (-0.8)^2 = 0.64$$

3. **Gradients**

   First compute

$$\frac{\partial L}{\partial \hat{y}} = -2(y - \hat{y}) = -2(2 - 2.8) = 1.6$$

   Output weight:

$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \beta} = 1.6(1.4) = 2.24$$

   Activation derivative:

$$g'(z) = 1 \quad (\text{since } z > 0)$$

   Hidden weight:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial A} \frac{\partial A}{\partial z} \frac{\partial z}{\partial w} = 1.6(2)(1)(3) = 9.6$$

   Bias:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial A} \frac{\partial A}{\partial z} \frac{\partial z}{\partial b} = 1.6(2)(1)(1) = 3.2$$

4. **Gradient Descent Update**

   Learning rate

$$\eta = 0.1$$

$$\beta_{new} = \beta - 0.1(2.24) = 1.776$$

$$w_{new} = w - 0.1(9.6) = -0.56$$

$$b_{new} = b - 0.1(3.2) = -0.12$$

## Cross-Entropy Loss

Cross-entropy loss is commonly used in classification problems to measure how well a model's predicted probabilities match the true labels. Cross-entropy measures how different a predicted probability distribution is from the true probability distribution.

Cross-entropy originates from **information theory**, where entropy measures the uncertainty of a probability distribution. Cross-entropy measures the expected amount of information required to encode outcomes drawn from a true distribution using probabilities from another distribution.

Suppose there are $M$ possible classes. Let

- $y_m$ denote the true label using one-hot encoding (1 for the correct class and 0 otherwise),
- $\hat{p}_m$ denote the predicted probability for class $m$.

The cross-entropy loss for a single observation is

$$L = -\sum_{m=1}^{M} y_m \log(\hat{p}_m).$$

Because only one value of $y_m$ equals 1, the expression simplifies to

$$L = -\log(\hat{p}_{\text{correct}}).$$

This means the loss depends only on the probability assigned to the correct class. The loss becomes small when the model assigns a high probability to the correct class, and large when it assigns a low probability.

## Example

Suppose we are classifying images into three classes: 0, 1, or 2.

If the true class is 1, the one-hot encoded label is

$$y = (0, 1, 0).$$

Suppose the model predicts the probabilities

$$\hat{p} = (0.1, 0.7, 0.2).$$

The cross-entropy loss becomes

$$L = -\log(0.7) \approx 0.357.$$

Now consider a worse prediction:

$$\hat{p} = (0.6, 0.2, 0.2).$$

The loss becomes

$$L = -\log(0.2) \approx 1.609.$$

The loss is much larger because the model assigned a low probability to the correct class. In this way, cross-entropy loss encourages the model to assign high probability to the correct class and low probability to the incorrect classes.

## Discussion Questions

1. Explain why a neural network with no hidden layer is closely related to linear regression or logistic regression.
2. Why is the activation function essential?
3. How does a neural network automatically learn nonlinear transformations compared with manually adding polynomial features?
4. Compare neural networks with decision trees and boosting. In what sense are all three nonlinear models?