Deep Learning and Neural Networks Basics Perceptron: Single vs Multi-Layer Perceptron (MLP) Lecture-4

CS4152: Deep Learning and Neural Networks

Dr. Jameel Ahmad
CS-SST-UMT,
Fall 2025

Lecture Overview

This Lecture covers:

- Understanding perceptrons and multilayer perceptrons
- Activation functions and their types
- Training networks: feedforward, error functions, optimization
- Backpropagation and gradient descent

Traditional ML vs. Deep Learning

Traditional ML:

- Manual feature extraction
- Feature vector $\rightarrow$ ML algorithm

Deep Learning:

- Automatic feature extraction
- Input $\rightarrow$ DL algorithm $\rightarrow$ Output

What is a Perceptron?

- Single neuron model
- Inspired by biological neurons
- Inputs: $x_1, x_2, \ldots, x_n$
- Weights: $w_1, w_2, \ldots, w_n$
- Output: $\hat{y} = \text{activation}(\sum w_i x_i + b)$

Weighted Sum and Bias

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

- Bias b allows shifting the decision boundary
- Without bias, line must pass through origin
- Implemented as an extra input with fixed value 1

Step Activation Function

$$\text{Output} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Used for binary classification
- Output is 0 or 1

How Perceptron Learns

1. Feedforward: compute $\hat{y}$
2. Calculate error: error $= y - \hat{y}$
3. Update weights: $w_i = w_i + \alpha \cdot \text{error} \cdot x_i$
4. Repeat until error is small

Limitations of Single Perceptron

- Only works for linearly separable data
- Fails on nonlinear datasets
- Example: XOR problem

Multilayer Perceptron (MLP)

- Multiple layers of neurons
- Input layer, hidden layers, output layer
- Fully connected layers
- Can model nonlinear relationships

MLP Architecture

- Input layer: feature vector
- Hidden layers: learn features
- Output layer: prediction
- Weights: connections between neurons

Hidden Layers

- Learn hierarchical features
- Early layers: edges, lines
- Later layers: shapes, objects
- "Hidden" because we don't see their outputs

Designing MLP

- Number of layers and neurons are hyperparameters
- Too few $\rightarrow$ underfitting
- Too many $\rightarrow$ overfitting
- Start small and increase complexity

Fully Connected Layers

- Each neuron in layer L is connected to all neurons in layer $L + 1$
- Total weights $=$ (input_size $\times$ hidden_size) $+$ biases
- Example: 4 inputs, 5 neurons $\rightarrow 45 + 5 = 25$ weights

Activation Functions Introduction

- Introduce nonlinearity
- Without them, MLP = single perceptron
- Common types: Step, Sigmoid, Tanh, ReLU, Softmax

Linear Activation

$$f(x) = x$$

- No nonlinearity
- Useless in hidden layers
- Derivative $= 1$

Sigmoid Function

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

- Output between 0 and 1
- Used for probability
- Suffers from vanishing gradients

Tanh Function

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

- Output between -1 and 1
- Zero-centered
- Better than sigmoid for hidden layers

ReLU Function

$$\text{ReLU}(x) = \max(0, x)$$

- Most popular for hidden layers
- Fast computation
- Avoids vanishing gradient for positive inputs

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Prevents "dead neurons"
- Small slope for negative inputs
- Hyperparameter: slope value

Softmax Function

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}}$$

- Used for multi-class classification
- Outputs sum to 1
- Generalization of sigmoid

Activation Function Cheat Sheet

Function	**Range**	**Use Case**
Step	$\{0,1\}$	Binary classification
Sigmoid	$(0,1)$	Probability
Tanh	$(-1,1)$	Hidden layers
ReLU	$[0,\infty)$	Hidden layers
Softmax	$(0,1)$	Multi-class output

Feedforward Process

- Forward pass through the network
- Compute weighted sum + activation
- Repeat for each layer
- Final output: prediction $\hat{y}$

Matrix Form of Feedforward

$$\hat{y} = \sigma(W^{(3)} \cdot \sigma(W^{(2)} \cdot \sigma(W^{(1)} \cdot X)))$$

- Efficient computation using matrices
- Each layer: $A^{(l)} = \sigma(W^{(l)} \cdot A^{(l-1)})$

Error Functions

- Measure of "wrongness" of prediction
- Also called loss or cost function
- Must be always positive
- Common: MSE, Cross-Entropy

Mean Squared Error (MSE)

$$E(W, b) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

- Used for regression
- Sensitive to outliers
- Always positive

Cross-Entropy Loss

$$E = -\sum_{j=1}^{m} y_j \log(\hat{p}_j)$$

- Used for classification
- Measures difference between distributions
- y = true label, $\hat{p}$ = predicted probability

Optimization Introduction

- Goal: minimize error function
- Adjust weights to reduce error
- Brute force is infeasible for large networks

Gradient Descent

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}$$

- α: learning rate
- $\frac{\partial E}{\partial w_i}$: gradient
- Update: $w_{\text{new}} = w_{\text{old}} + \Delta w$

Learning Rate

- Step size in gradient descent
- Too small $\rightarrow$ slow convergence
- Too large $\rightarrow$ oscillation
- Typical values: 0.1, 0.01, 0.001

Batch Gradient Descent

- Uses entire training set for each update
- Computationally expensive
- Smooth convergence

Stochastic Gradient Descent (SGD)

- Uses one sample per update
- Faster but noisy
- Can escape local minima

Mini-Batch Gradient Descent

- Compromise between batch and SGD
- Uses small batches (e.g., 32, 64, 128)
- Most commonly used in practice

Backpropagation

- Core of neural network learning
- Propagates error backward
- Uses chain rule to compute gradients

Chain Rule in Backpropagation

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a^{(l+1)}} \cdot \frac{\partial a^{(l+1)}}{\partial z^{(l+1)}} \cdot \frac{\partial z^{(l+1)}}{\partial w_{ij}^{(l)}}$$

- Multiply local gradients backward
- Efficient computation using dynamic programming

Summary

- Perceptrons for linear problems
- MLPs for nonlinear problems
- Activation functions introduce nonlinearity
- Training: feedforward, error, backpropagation
- Optimization: gradient descent variants

Parameters vs. Hyperparameters

Parameters:

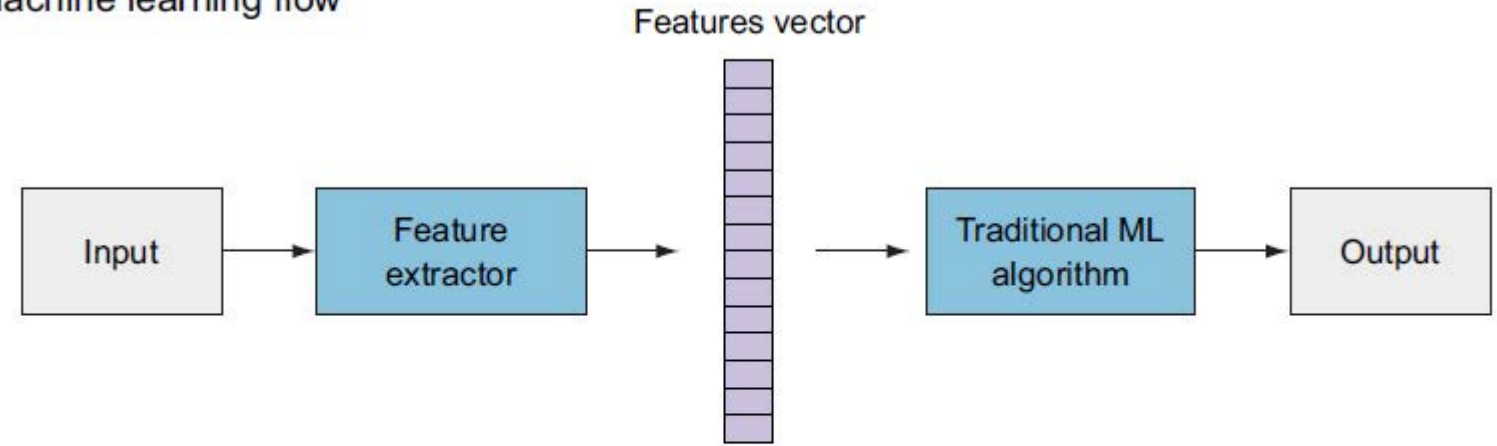
- Weights, biases
- Learned during training

Hyperparameters:

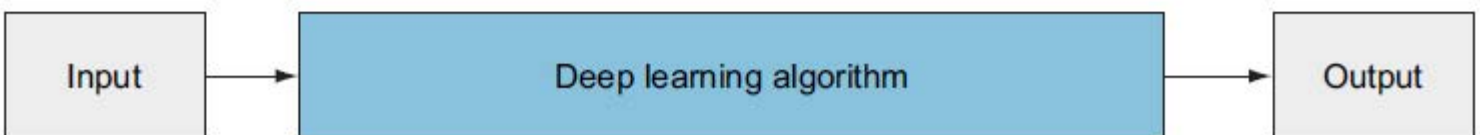
- Learning rate, batch size, layers
- Tuned by the ML or DL engineer

Thank You!
More Exciting Deep Learning in Next Weeks

Traditional machine learning flow

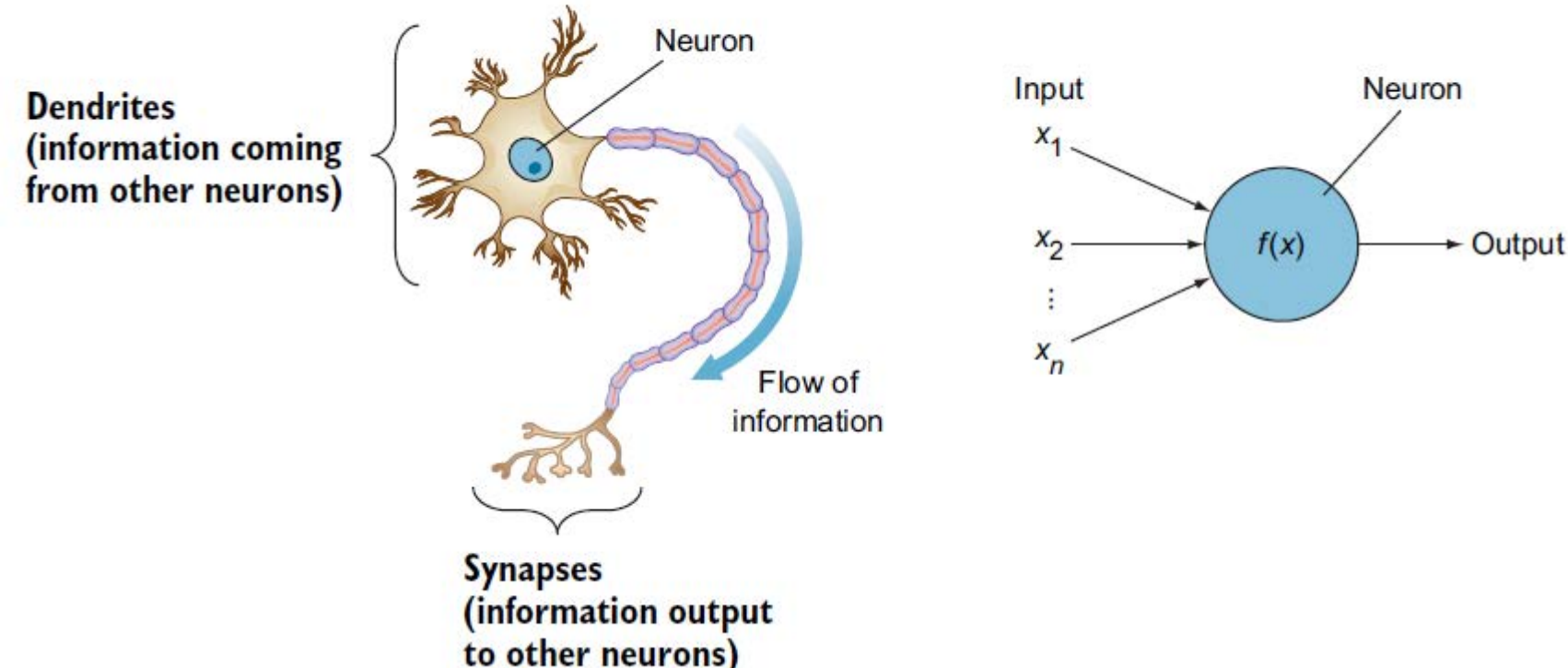


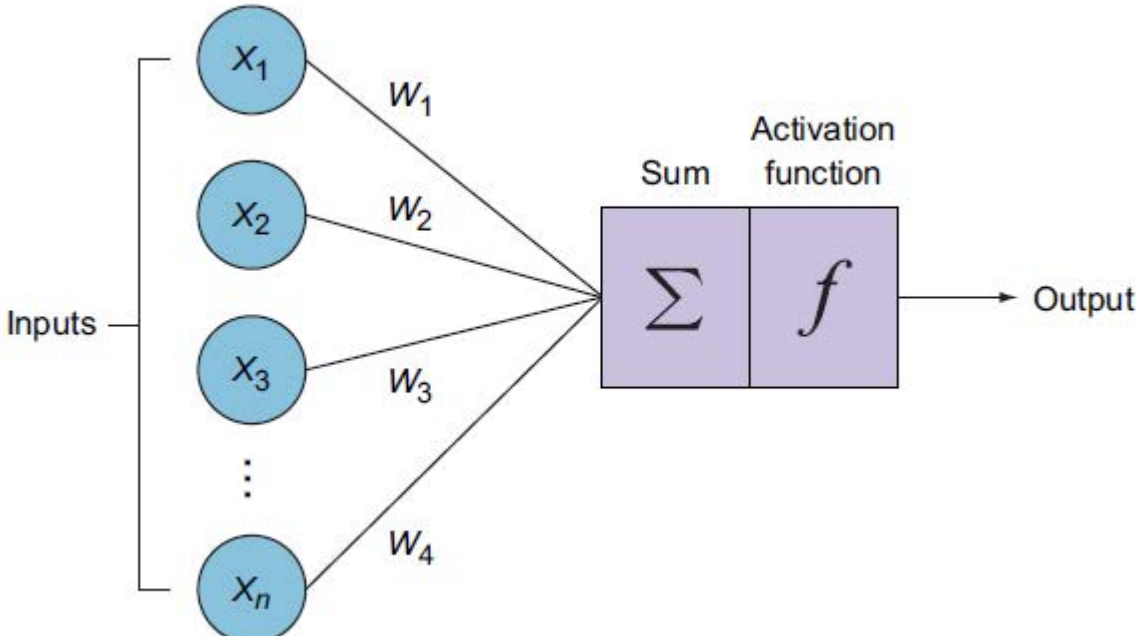
Deep learning flow



Biological neuron

Artificial neuron




Inputs
x_1
x_2
x_3
x_n
w_1
w_2
w_3
w_4
Sum
Activation function
Σ
f
Output

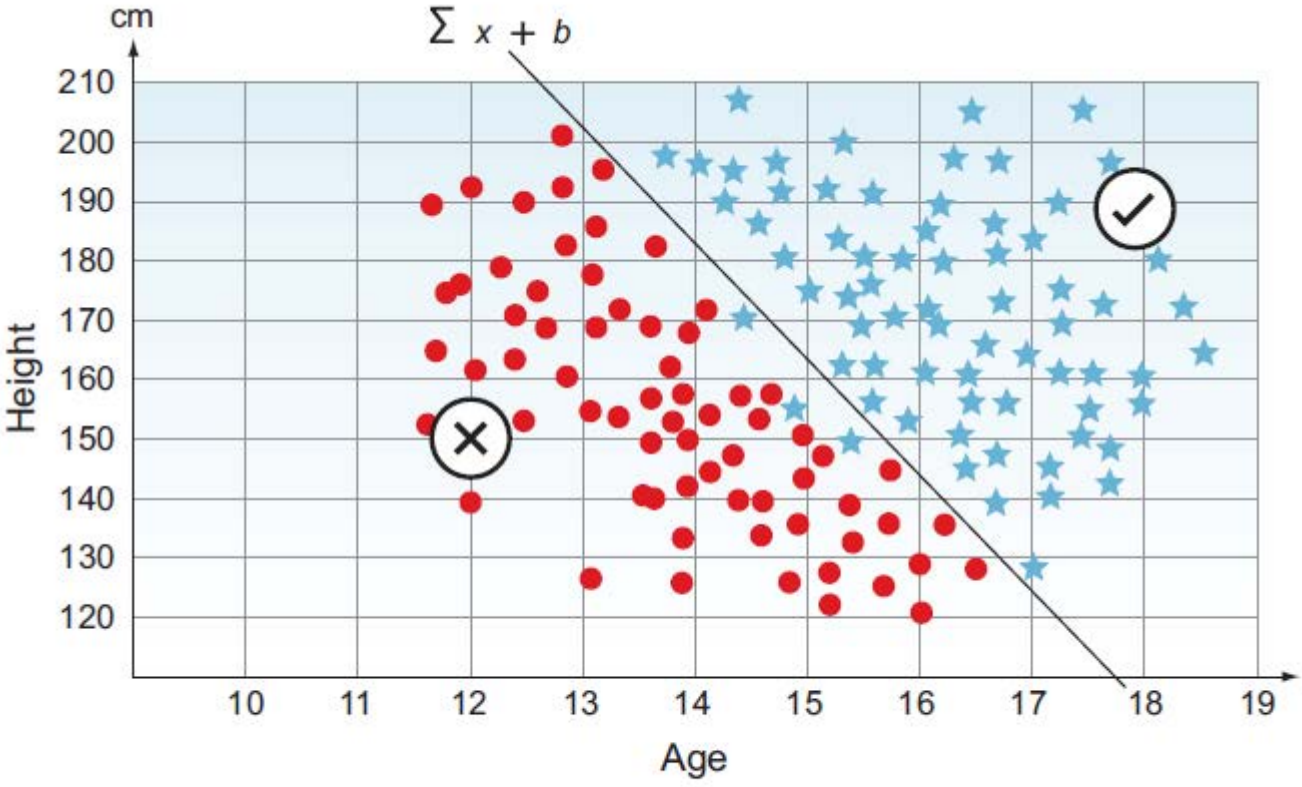
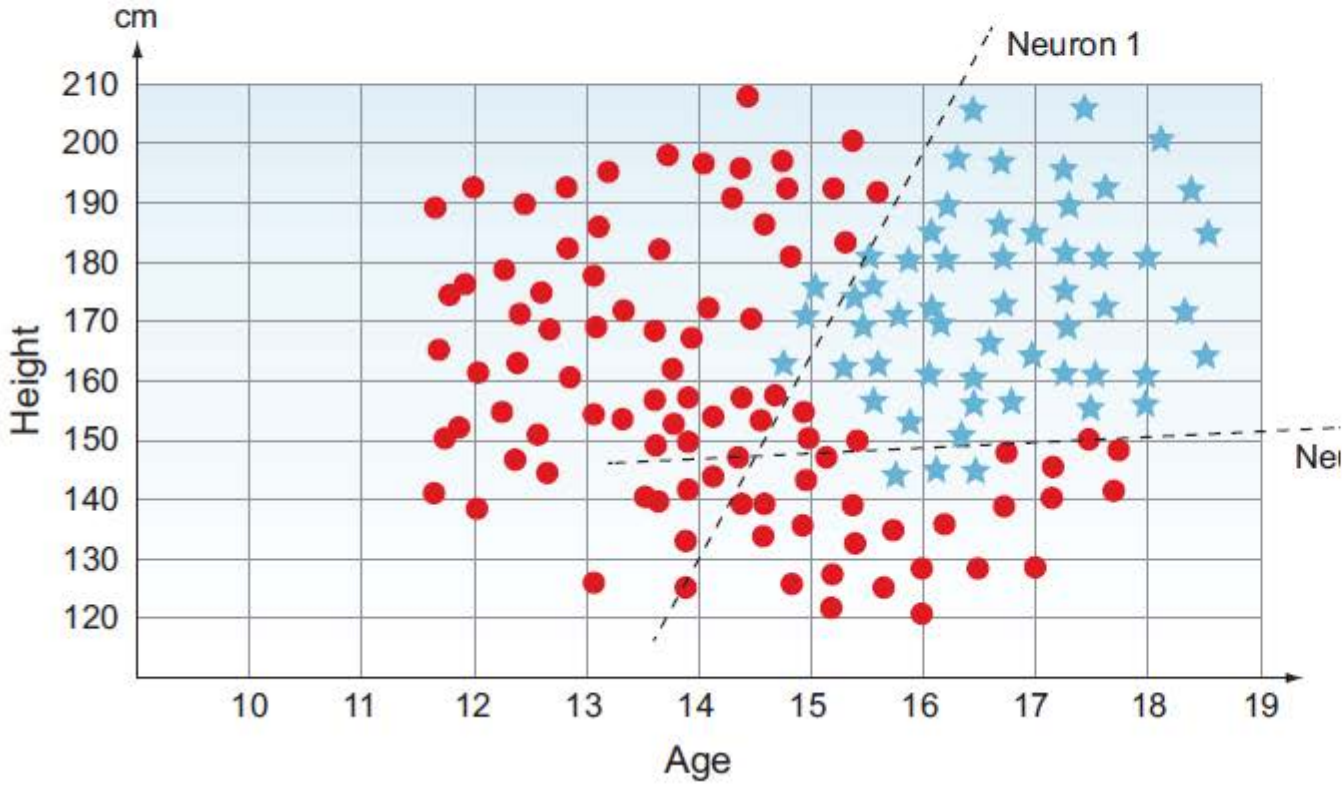
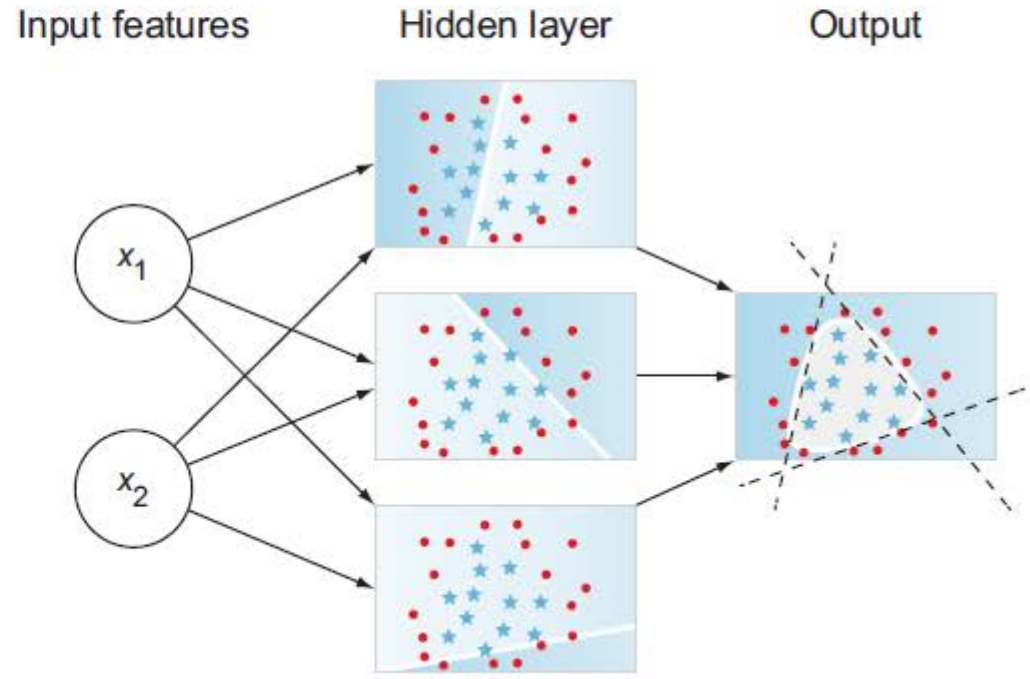
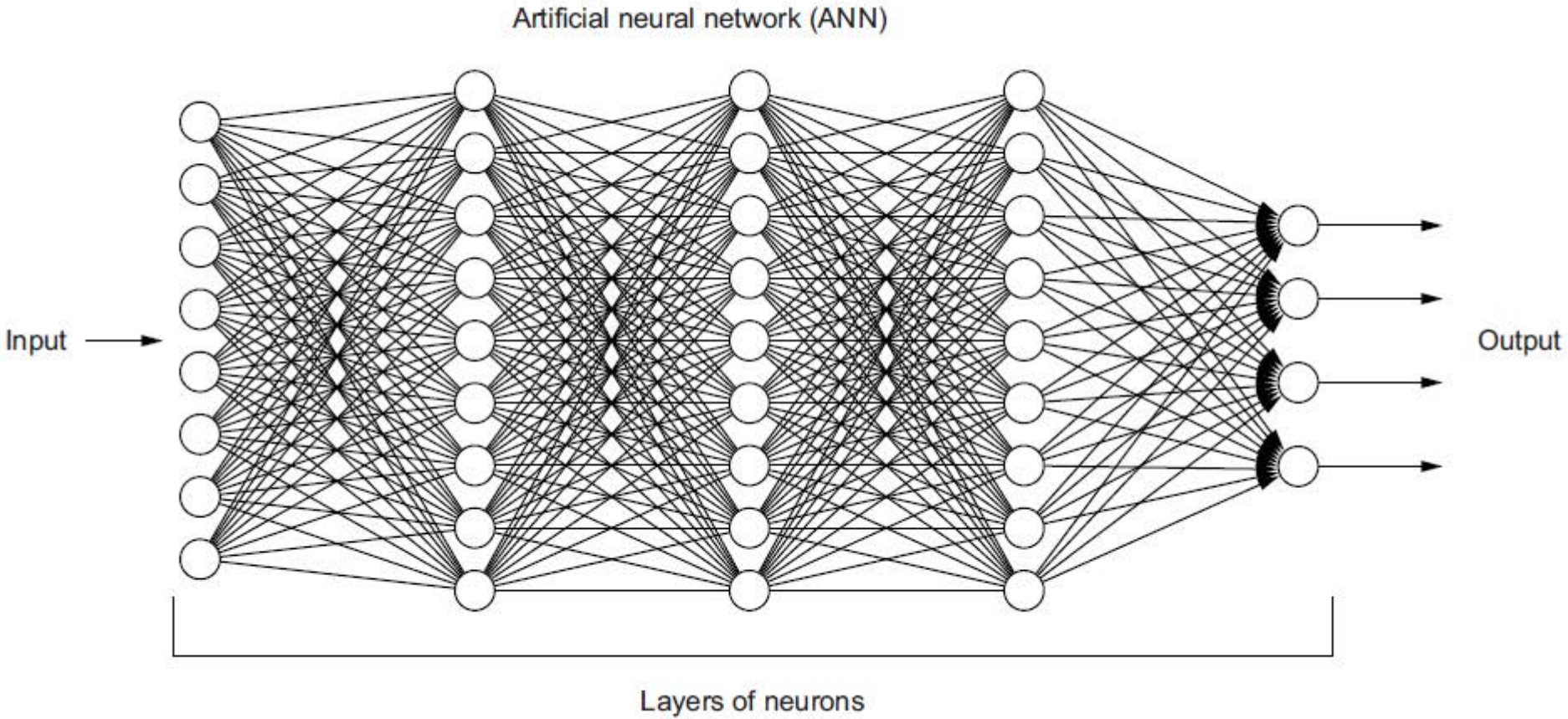


Figure **Linearly separable data can be separated by a straight line.**

Figure In a nonlinear dataset, a single straight line cannot separate the training data. A network with two perceptrons can produce two lines and help separate the data further in this example.






Artificial neural network (ANN)
Input
Output
Layers of neurons

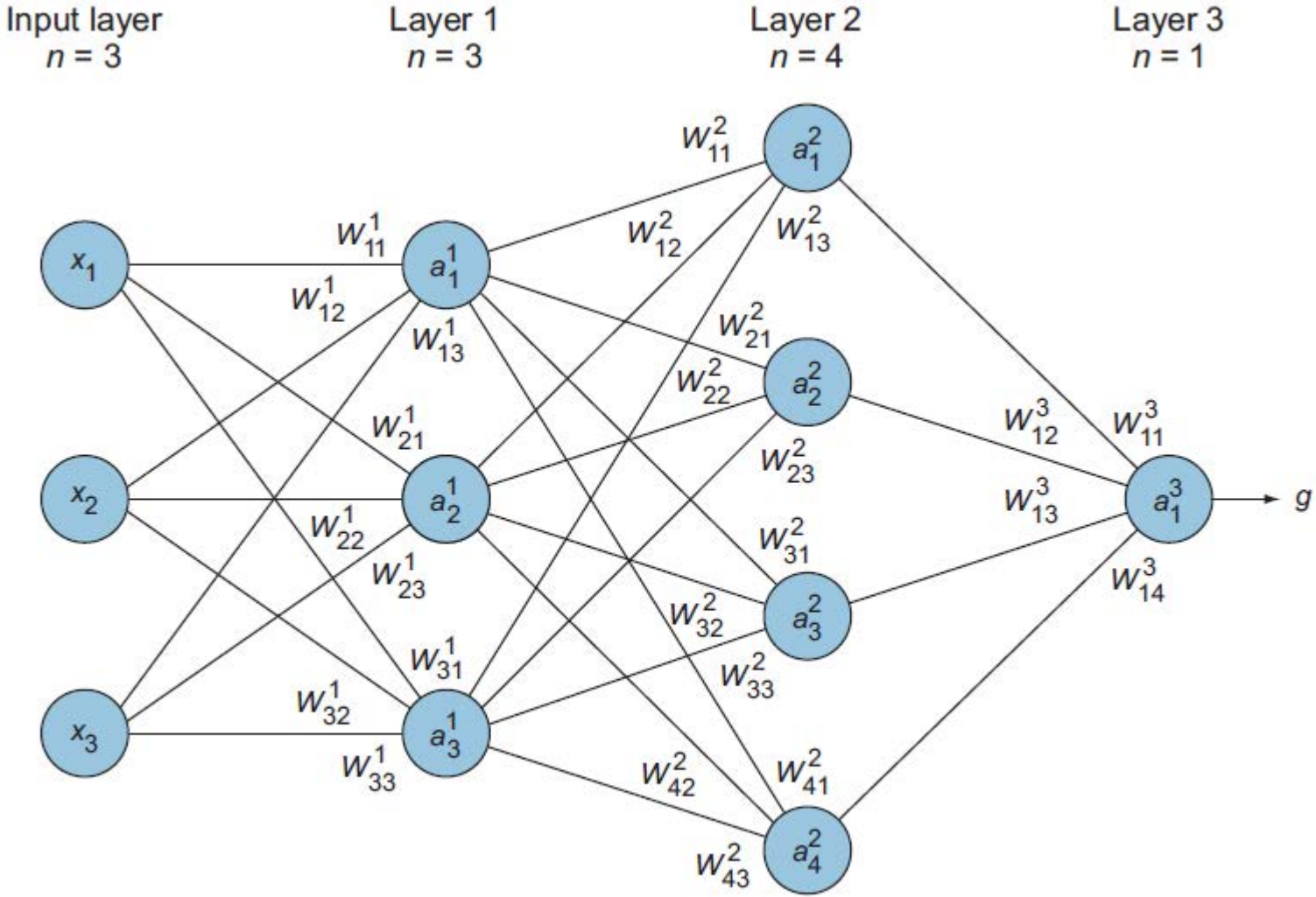


Figure 2.20 A simple three-layer neural network

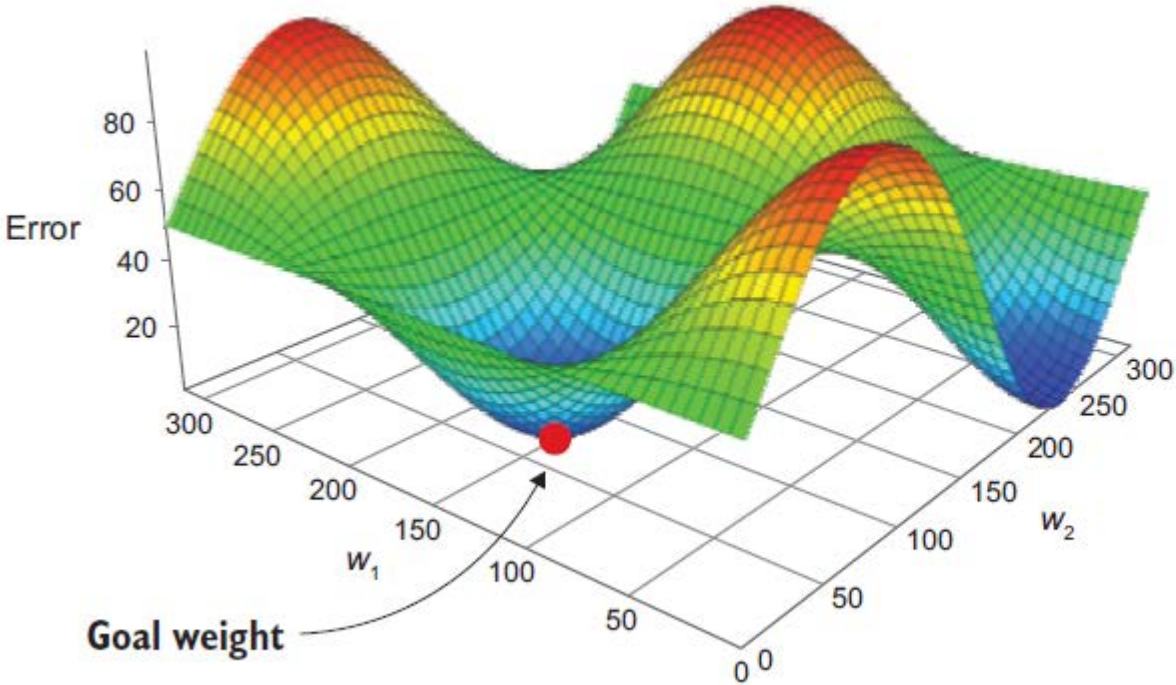


Figure 2.27 Graphing all possible values of two weights gives a 3D error plane.

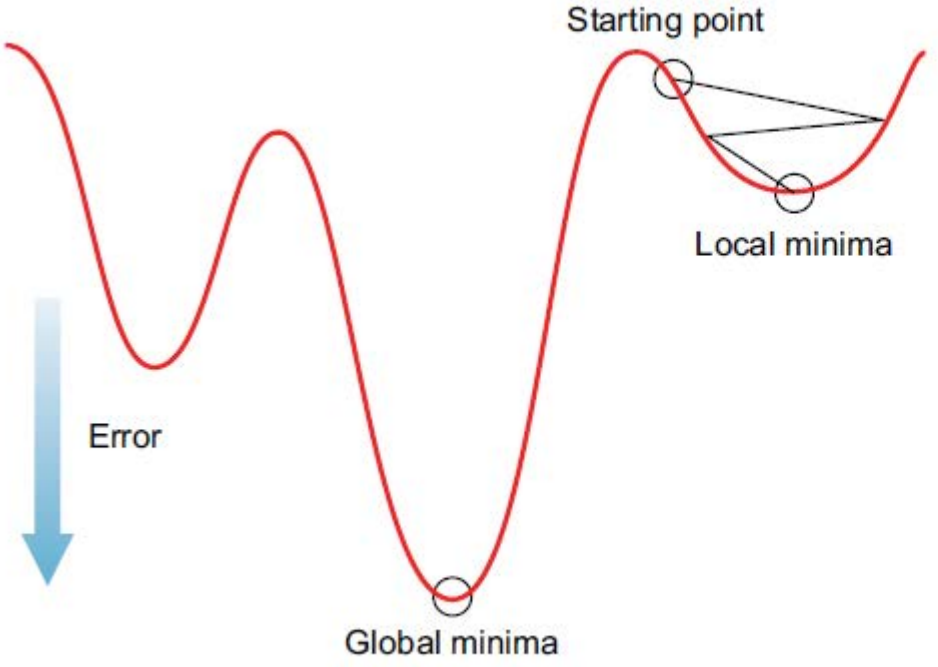
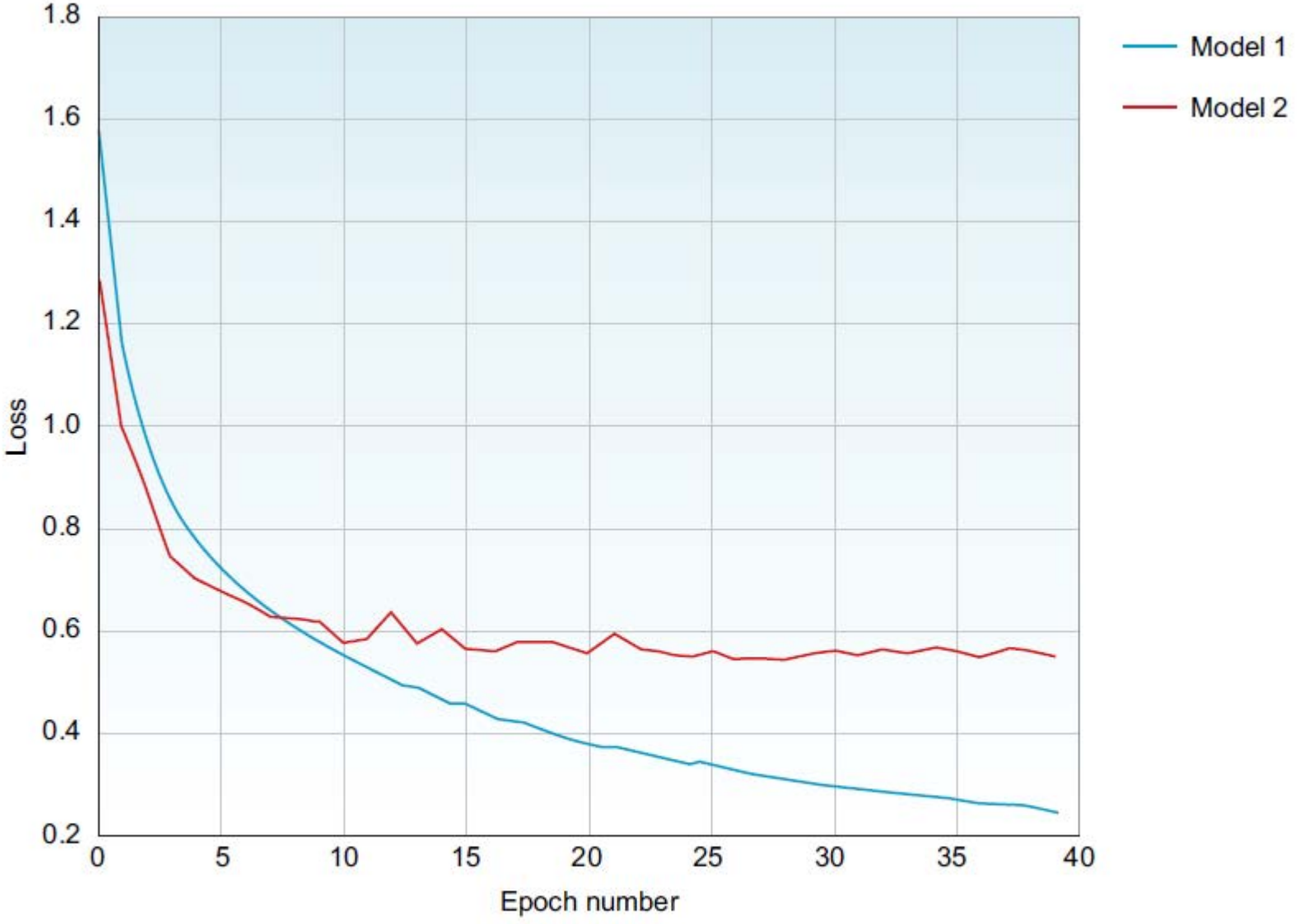
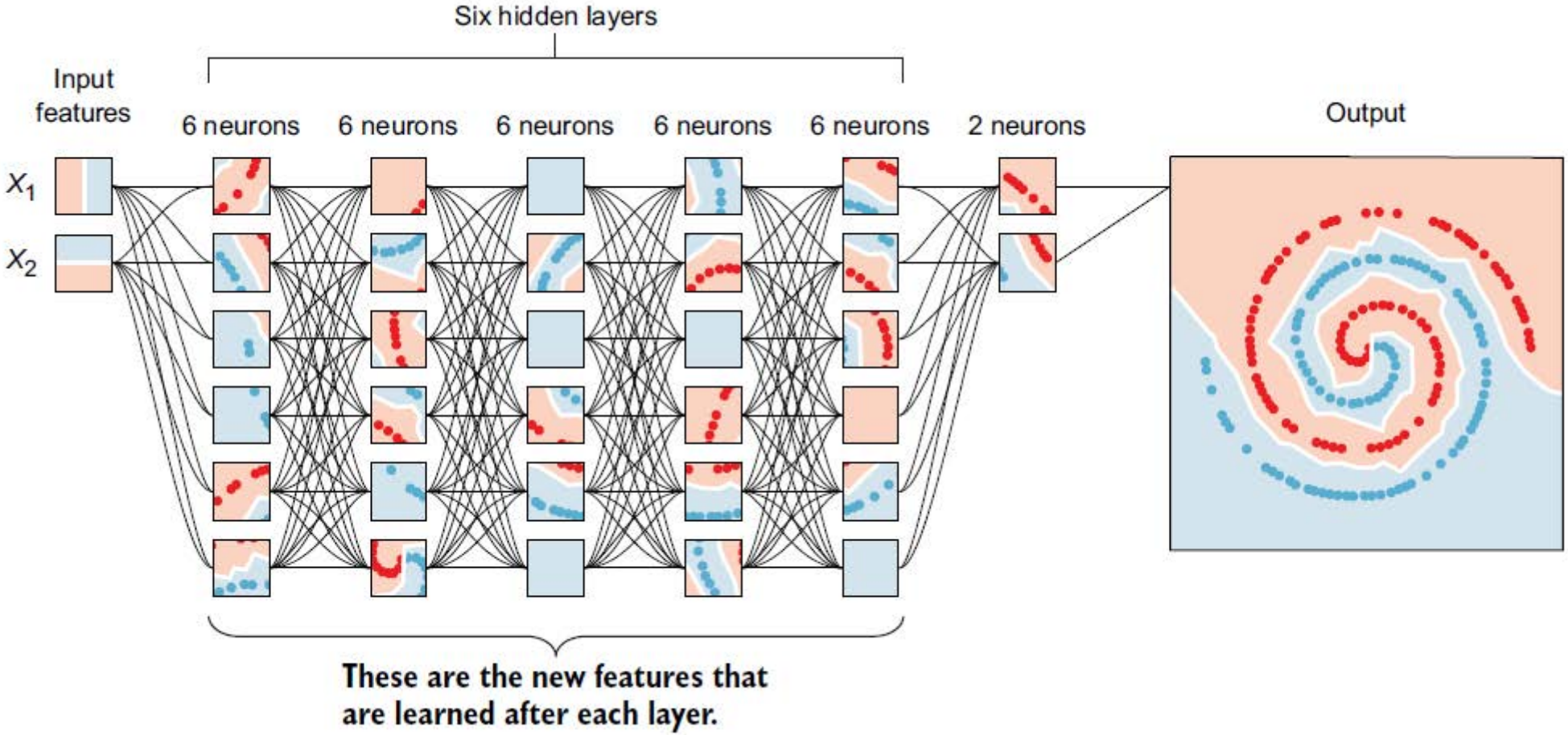


Figure 2.33 Complex error functions are represented by more complex curves with many local minima values. Our goal is to reach the global minimum value.


Model 1
Model 2
Loss
Epoch number
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
35
40


Six hidden layers
Input features
6 neurons
6 neurons
6 neurons
6 neurons
6 neurons
2 neurons
Output
X_1
X_2
These are the new features that are learned after each layer.

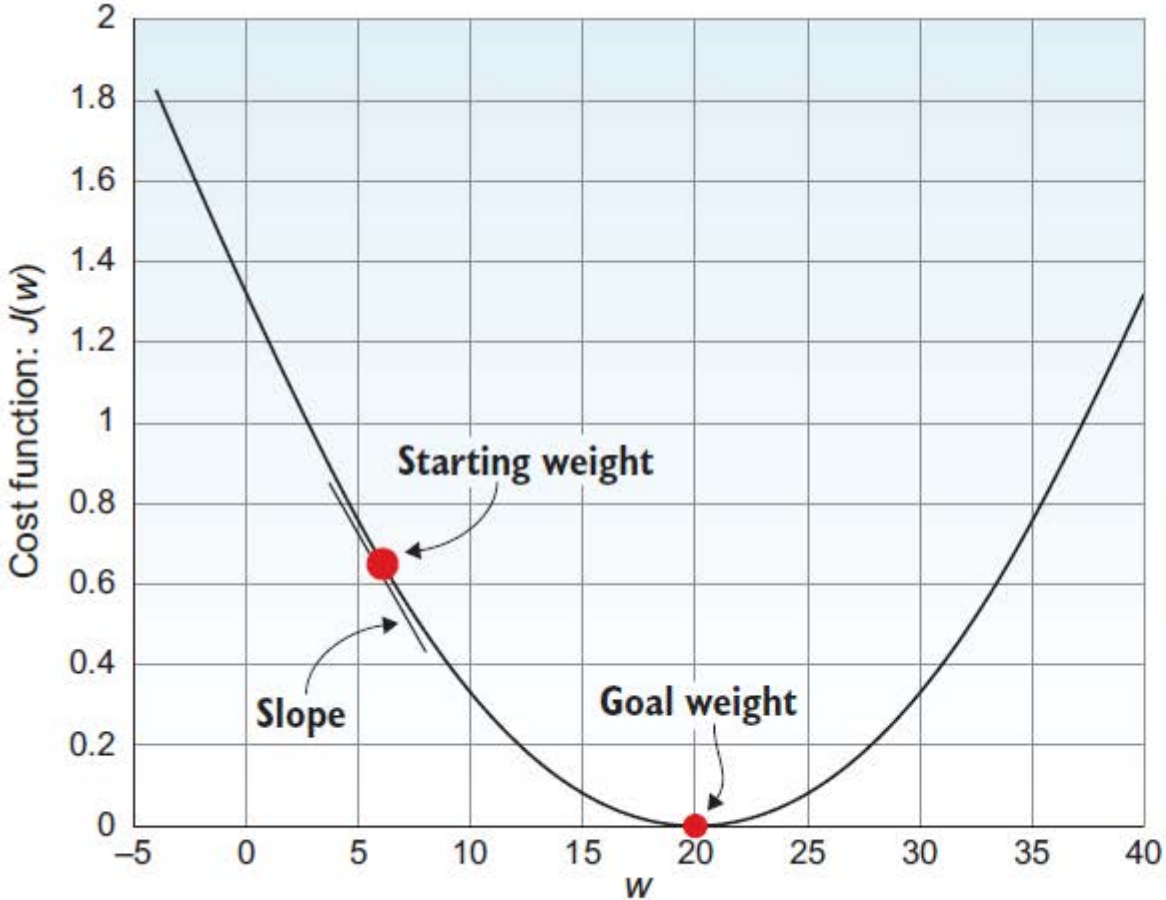


Figure 2.25 The network learns by adjusting weight. When we plot the error function with respect to weight, we get this type of graph.