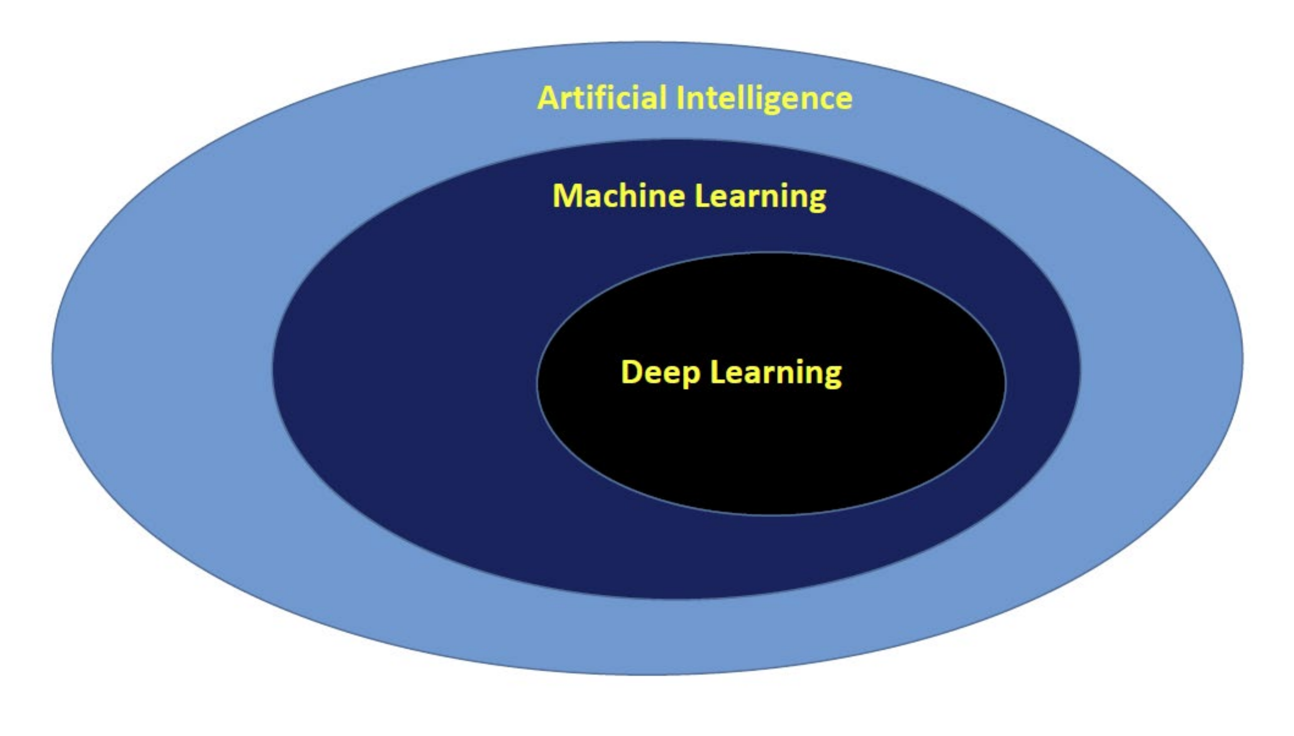
Artificial Neural Networks

Chapter 10: pp 279 – 294, 323 – 329

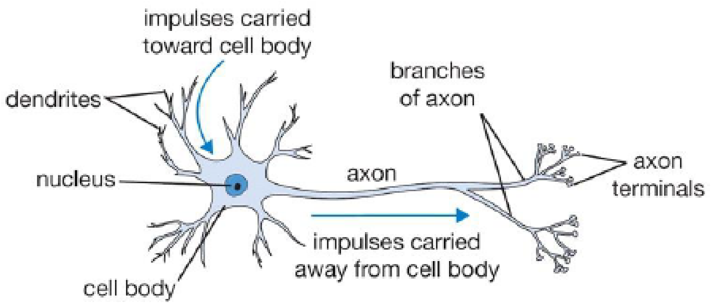
Artificial Neural Networks (ANNs)

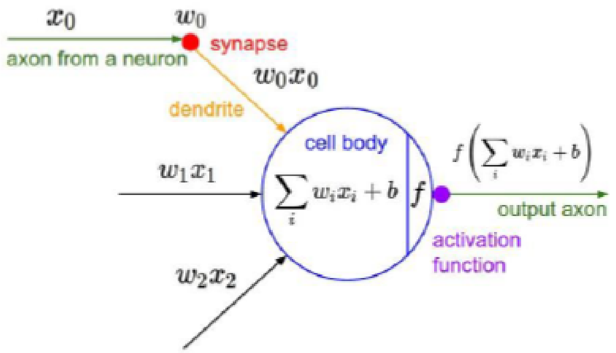
- Versatile, powerful, scalable
- The essence of Deep Learning
- Used to tackle highly complex ML tasks
  - Speech recognition
  - Object detection
  - Self-driving car
  - Recommendation systems
  - Machine translation
  - . . .

- Popular Deep Learning Terms
  - Deep Learning = Neural Networks with multiple hidden layers
  - DNN = Deep Neural Network



From Biological to Artificial Neurons





- **Neuron:** computational building block for the brain
- Human brain:
  - ~100-1,000 trillion synapses

- **(Artificial) Neuron:** computational building block for the "neural network"
- (Artificial) neural network:
  - ~1-10 billion synapses

Human brains have ~10,000 computational power than computer brains.

Logic Computations with Neurons

- Simple logical computations

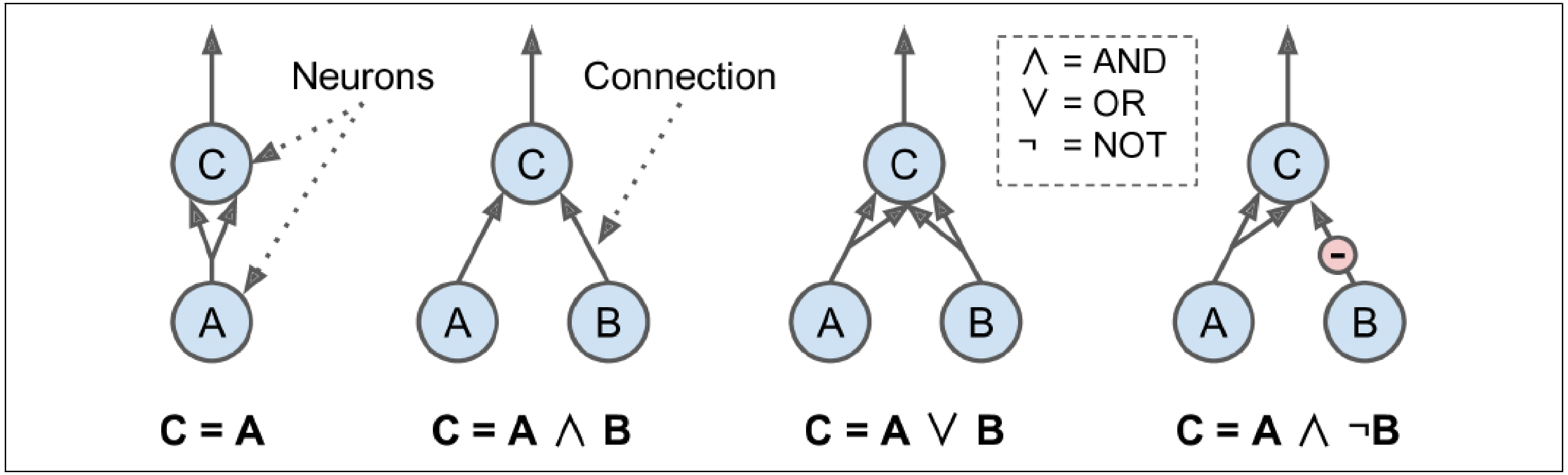


Figure 10-3. ANNs performing simple logical computations

Perceptron (Rosenblatt's Perceptron)

- a.k.a Threshold Logic Unit (TLU), Linear Threshold Unit (LTU)
  - Inputs & outputs are real numbers instead of binary values
  - Each input connection is weighted
  - Output is the result of a step function of sum of the weighted inputs

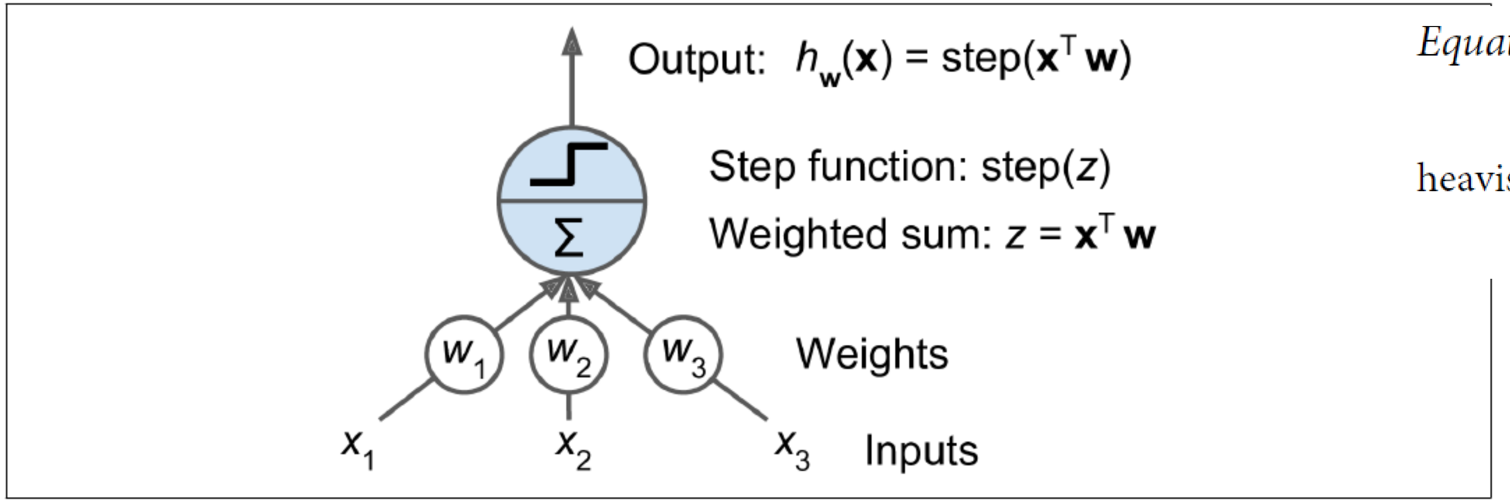


Figure 10-4. Threshold logic unit

Equation 10-1. Common step functions used in Perceptrons

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

- A single TLU can do simple linear binary classification
- A perceptron is a single layer *fully-connected* TLU’s
- Every neuron is connected to the inputs

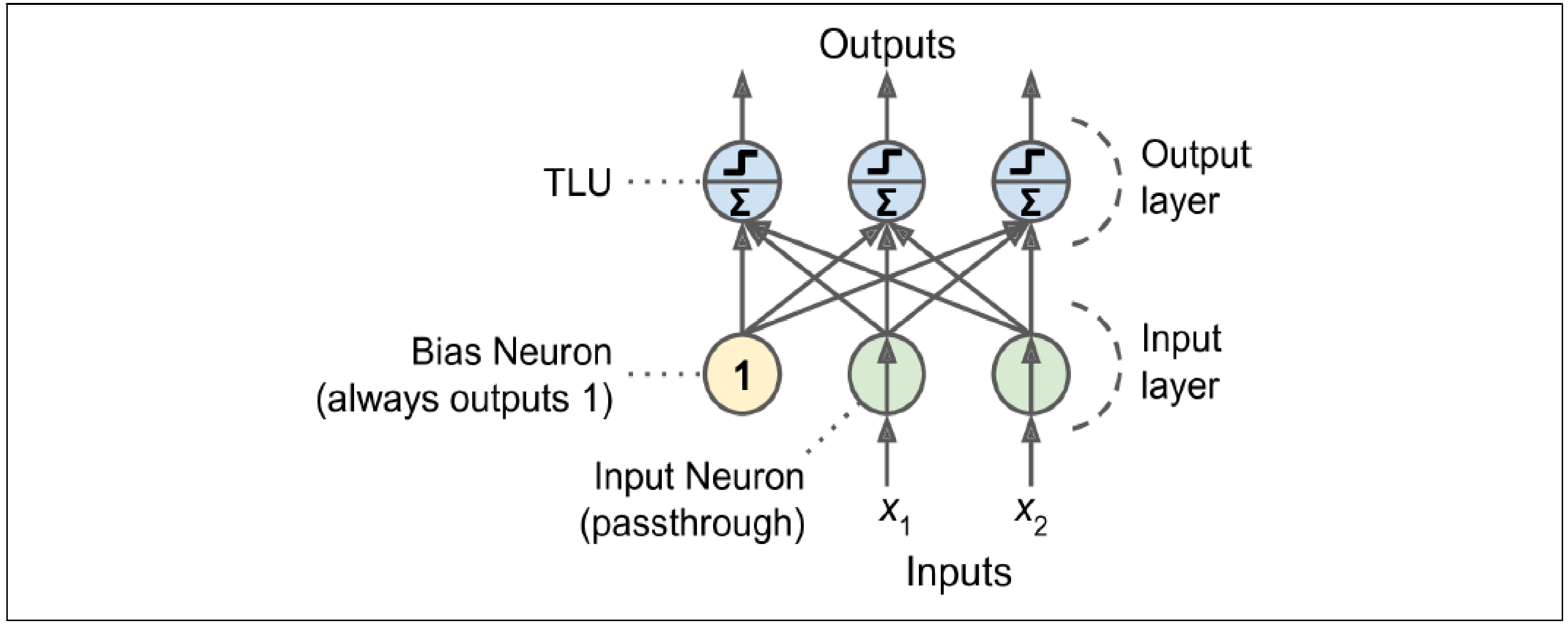


Figure 10-5. Perceptron diagram

- $\mathbf{X}$ is the input feature matrix.
  - One row per instance, one column per feature
- $\mathbf{W}$ is the connection weight matrix
  - Contains all connection weights except for the ones from bias neuron
  - One row per input, one column per neuron in the layer
- $\mathbf{b}$ is the bias vector
  - Contains all connection weights between the bias neuron and the other neurons
  - One bias term per neuron
- ϕ is the activation function
  - A step function

Training Perceptrons

- Hebb's Rule: the connection weight between two neurons is increased if they have the same output

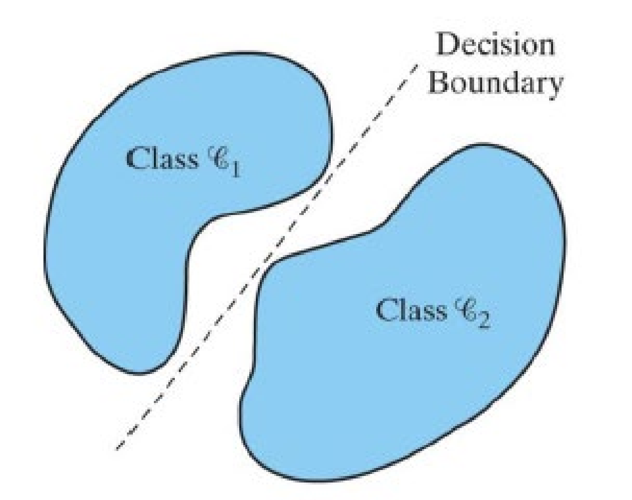
Equation 10-3. Perceptron learning rule (weight update)

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$$

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- $\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.

- Perceptron Convergence Theorem
  - Algorithm will converge well for linearly separable classes

- Comparison with Logistic Regression
  - Perceptron outputs a class label based on a hard threshold
  - Logistic Regression outputs class probability



(a)

- Perceptron cannot solve XOR classification problem but Multi-Layer Perceptron (MLP) can

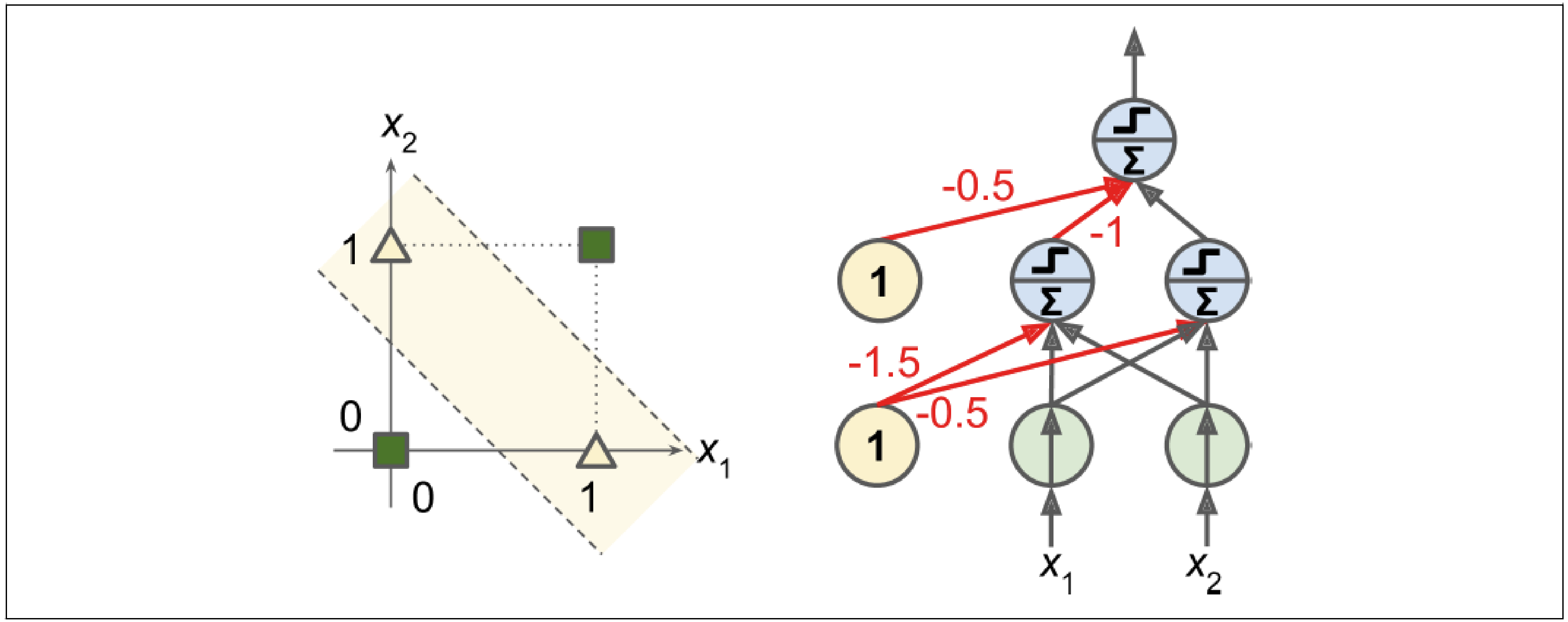


Figure 10-6. XOR classification problem and an MLP that solves it

Multi-Layer Perceptron

- MLP consists of input layer, at least one hidden layer, output layer
- Signal flows from inputs to outputs → feedforward NN
- Deep Neural Network (DNN) has many hidden layers

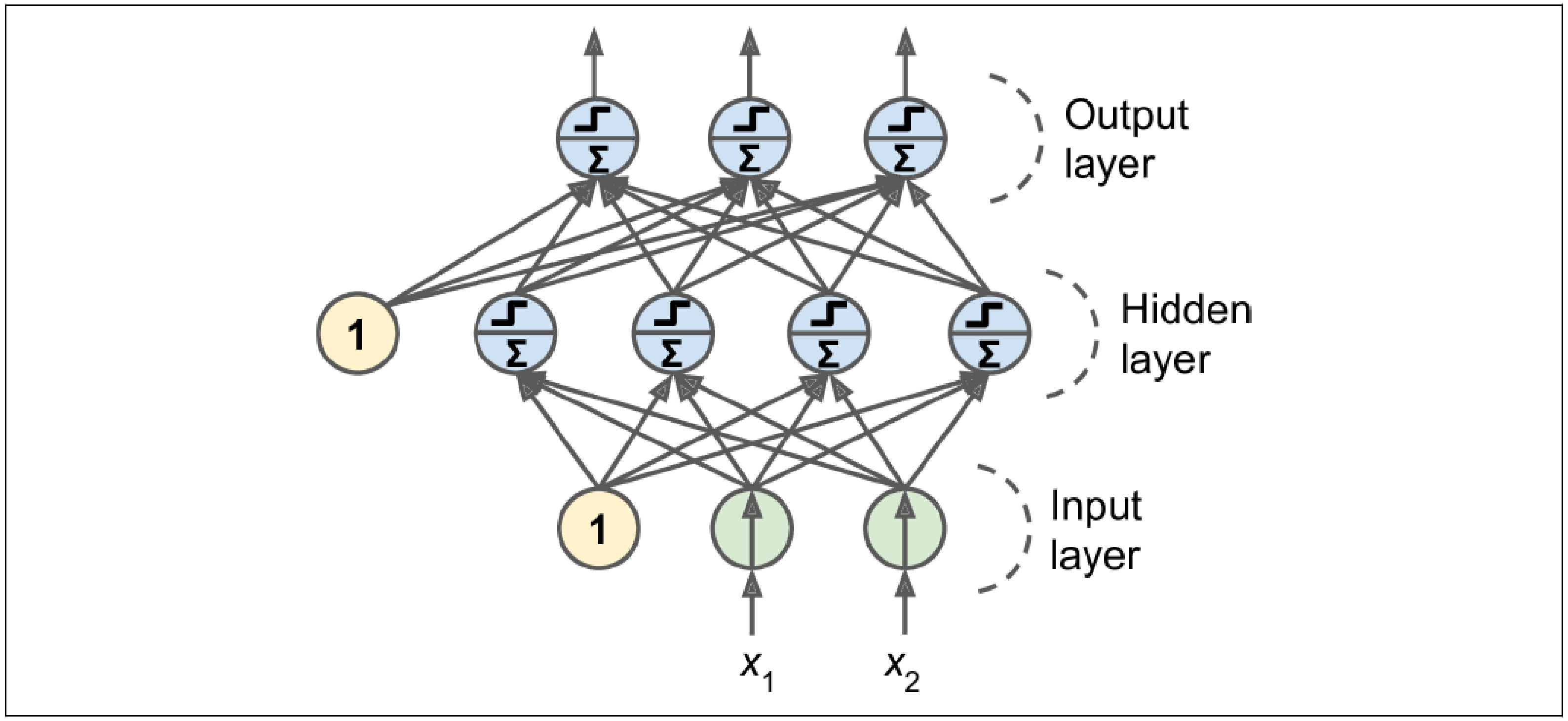


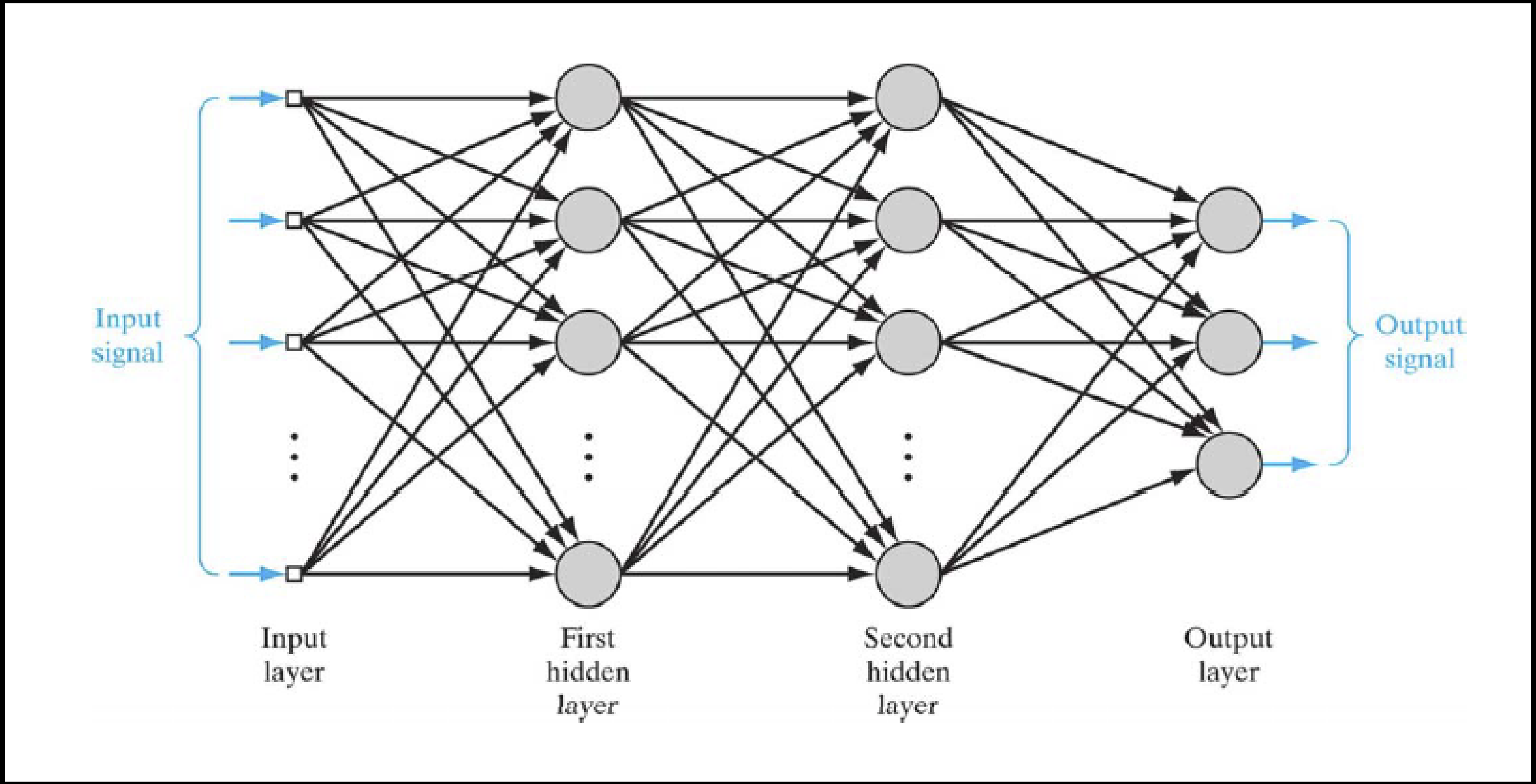
Figure 10-7. Multi-Layer Perceptron

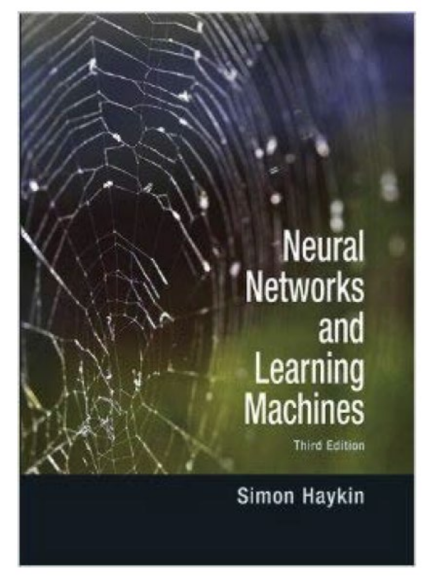
Function of the Hidden Neurons

- The hidden neurons act as *feature detectors*.
- As the learning process progresses across the multilayer perceptron, the hidden neurons begin to gradually "discover" the salient features that characterize the training data by performing a nonlinear transformation on the input data into a new space called *feature space*.
- In the feature space, the classes of interest in a pattern-classification task, for example, may be more easily separated from each other than could be in the original input data space.

- Neural Networks and Learning Machines by Simon Haykin

Architectural graph of MLP with 2 hidden layers

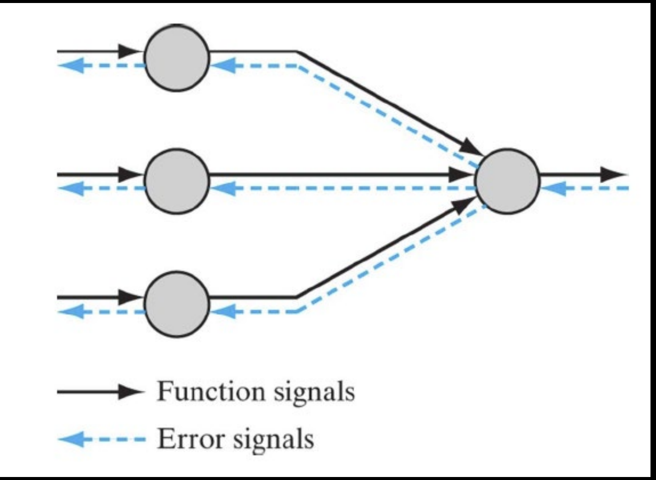




- This is called a *three-layer* neural network
- # of input connections = # features
- # output neurons = # dimensions (regression) or # of labels/classes (classification)

Backpropagation Algorithm

- How do you train MLP?
  - Automatically find the error gradients with the training data and adjust the connection weights



- Forward pass
  - Makes predictions
  - Measures the error
- Reverse pass
  - Goes through each layer in reverse
  - Measures the error contribution from each connection
  - Modifies connection weights layer by layer to reduce the error

The Back-Propagation Algorithm

For an output neuron j,

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \qquad y_j(n) = \varphi(v_j(n))$$

$$\mathcal{E}(n) = \frac{1}{2}\sum_j e_j^2(n) = \frac{1}{2}\sum_j \left[d_j(n) - y_j(n)\right]^2$$

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$
$$= -e_j(n)\varphi'(v_j(n))y_i(n)$$

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \eta \delta_j(n) y_i(n) \quad \text{where } \delta_j(n) = e_j(n)\varphi'(v_j(n))$$

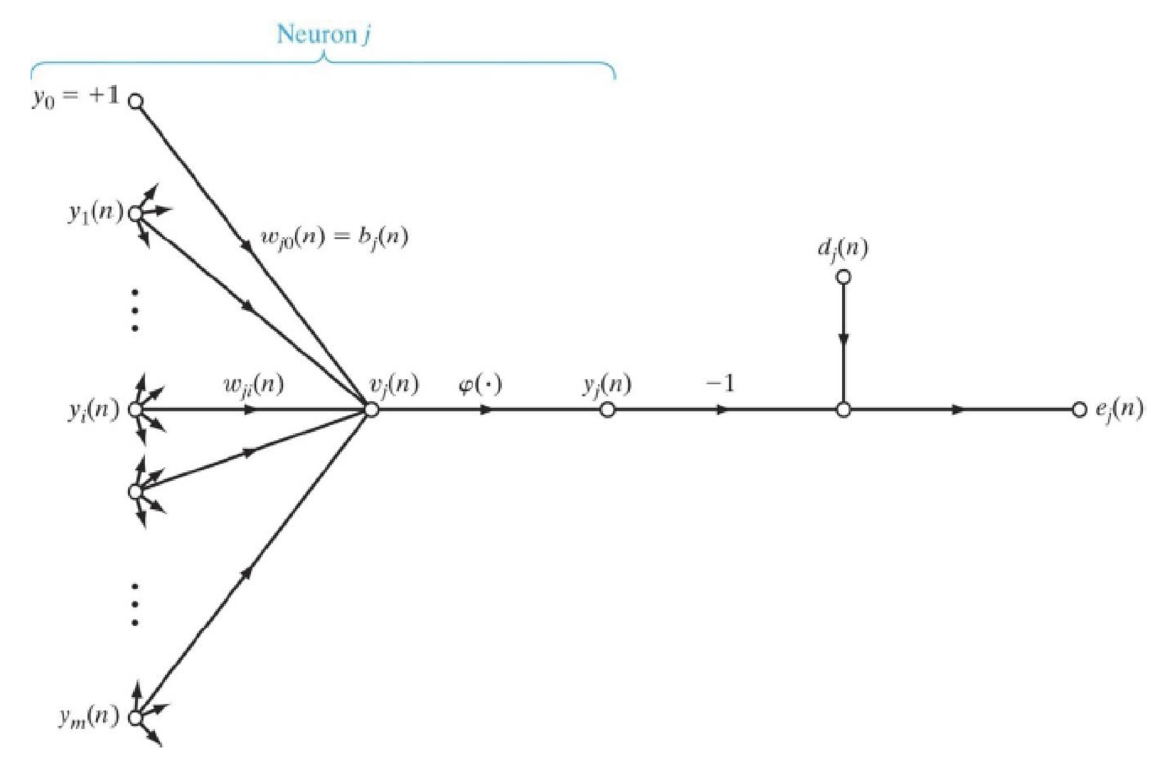


FIGURE 4.3 Signal-flow graph highlighting the details of output neuron j.

- Neural Networks and Learning Machines by Simon Haykin

For a hidden neuron j,

$$\mathcal{E}(n)=\frac{1}{2}\sum_k e_k^2(n)=\frac{1}{2}\sum_k\left[d_k(n)-\varphi\left(\sum_j w_{kj}^{out}(n)\,y_j^h(n)\right)\right]^2$$

$$y_j^h(n)=\varphi\left(\sum_i w_{ji}^h(n)\,y_i^{in}(n)\right)$$

$$\frac{\partial\mathcal{E}(n)}{\partial w_{ji}(n)}=\sum_k e_k\,\frac{\partial e_k(n)}{\partial v_k(n)}\,\frac{\partial v_k(n)}{\partial y_j^h(n)}\,\frac{\partial y_j^h(n)}{\partial w_{ji}(n)}$$

$$=-\sum_k e_k\varphi'(v_k(n))w_{kj}(n)\varphi'(v_j(n))y_i(n)$$

$$=-\sum_k \delta_k(n)w_{kj}(n)\varphi'(v_j(n))y_i(n)$$

$$\Delta w_{ji}(n)=\eta\,\delta_j(n)\,y_i(n)\quad\text{where}\quad \delta_j(n)=\underline{\sum_k \delta_k(n)w_{kj}(n)}\;\varphi'(v_j(n))$$

Backpropagated gradient
from the layer to the
immediate right

$$\begin{pmatrix}\text{weight}\\\text{correction}\\\Delta w_{ji}(n)\end{pmatrix}=\begin{pmatrix}\text{learning}\\\text{rate}\\\eta\end{pmatrix}\times\begin{pmatrix}\text{overall}\\\text{local gradient}\\\delta_j(n)\end{pmatrix}\times\begin{pmatrix}\text{input signal}\\\text{of neuron } j\\y_i(n)\end{pmatrix}$$

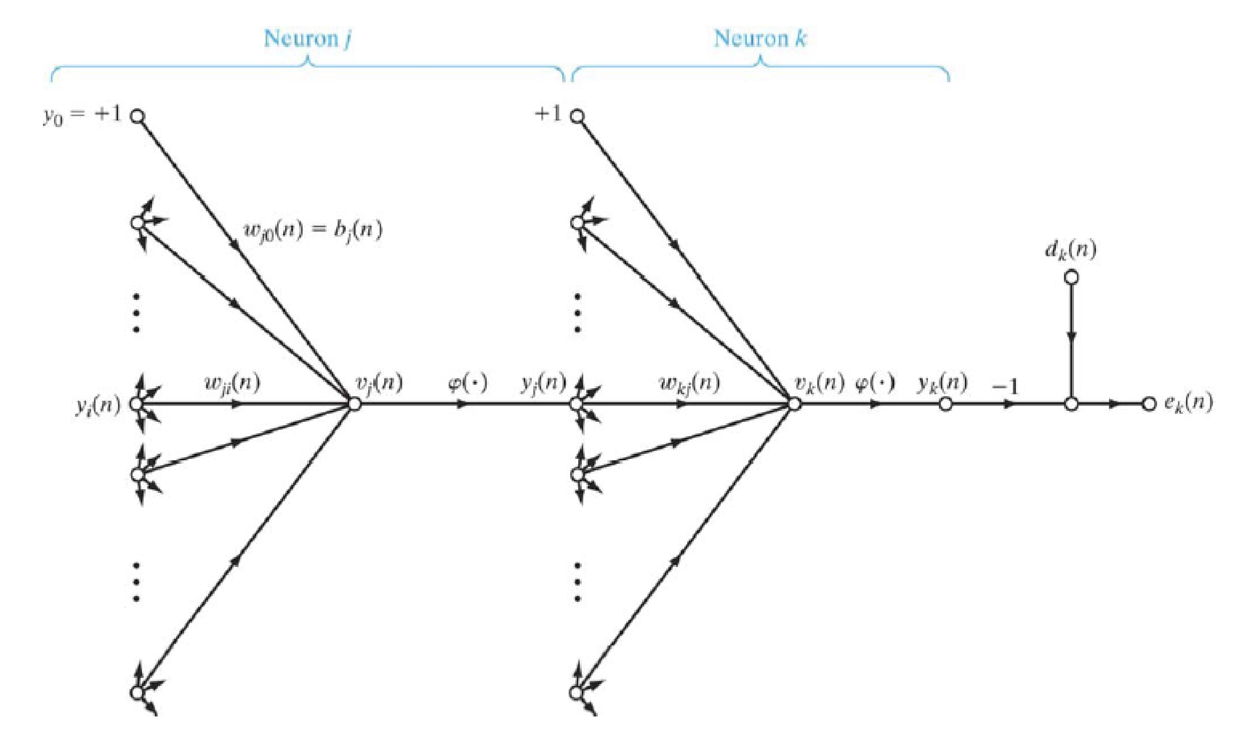
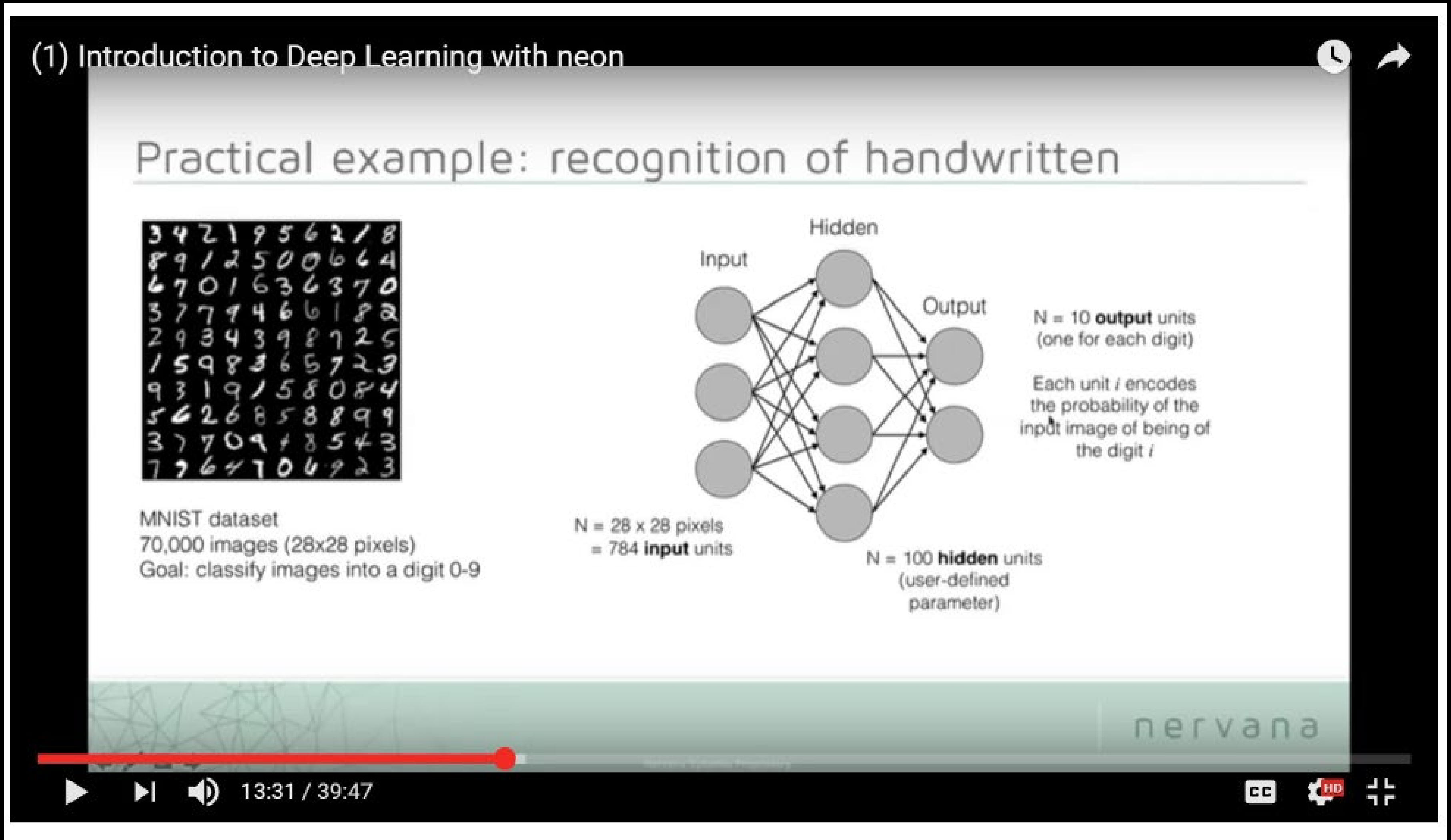
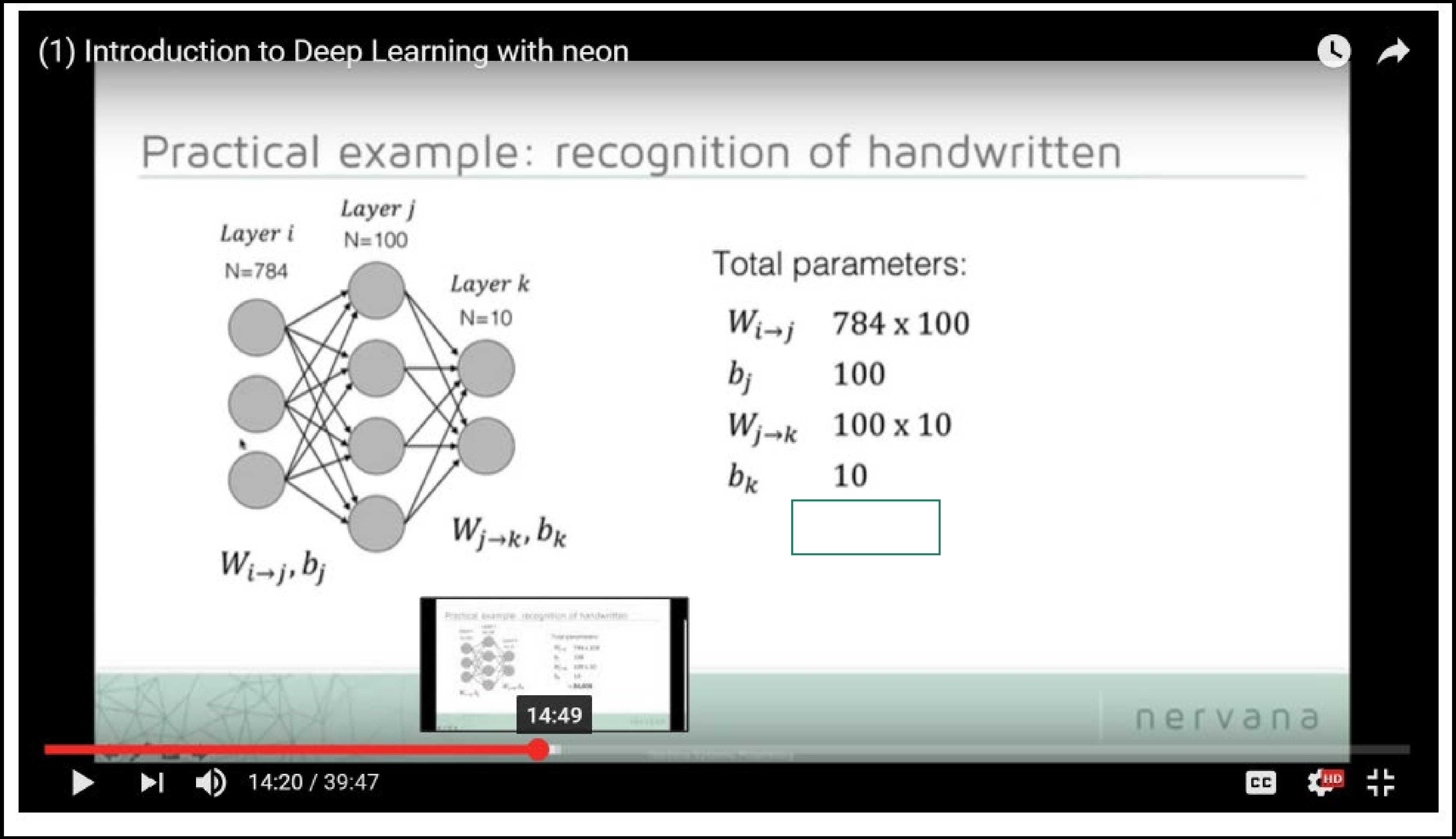


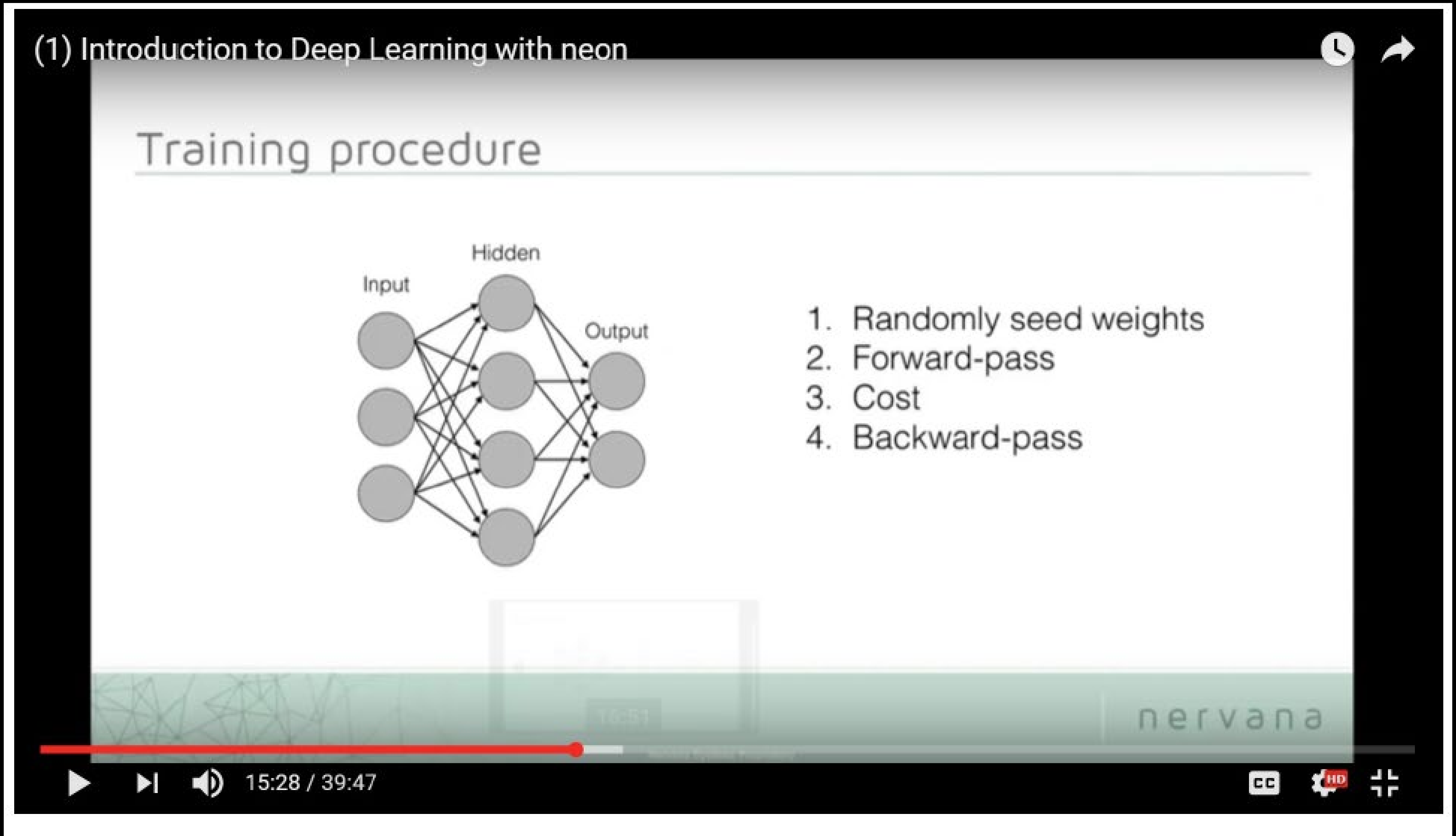
FIGURE 4.4 Signal-flow graph highlighting the details of output neuron k connected to hidden neuron j.

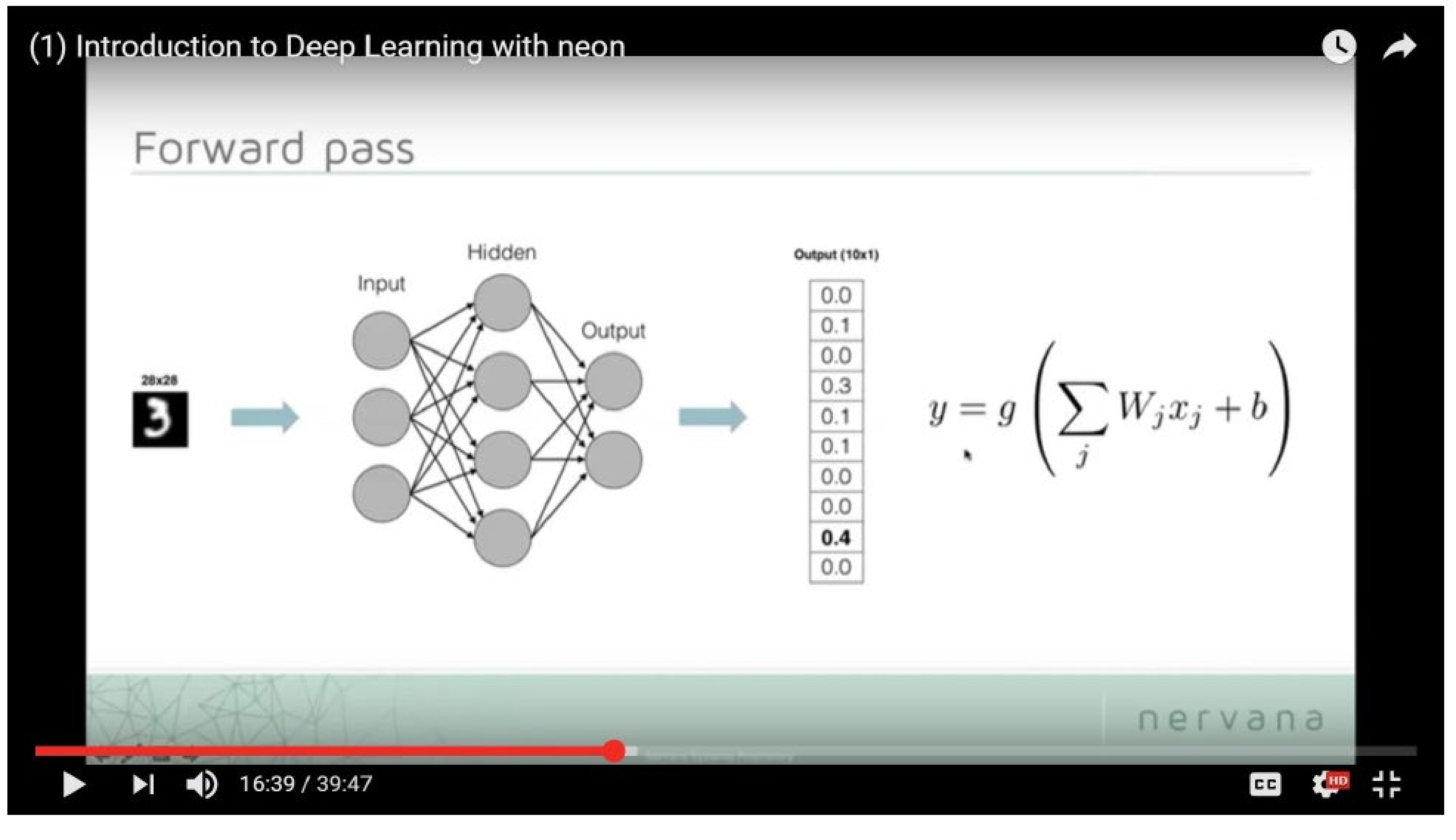
- Neural Networks and Learning Machines by Simon Haykin

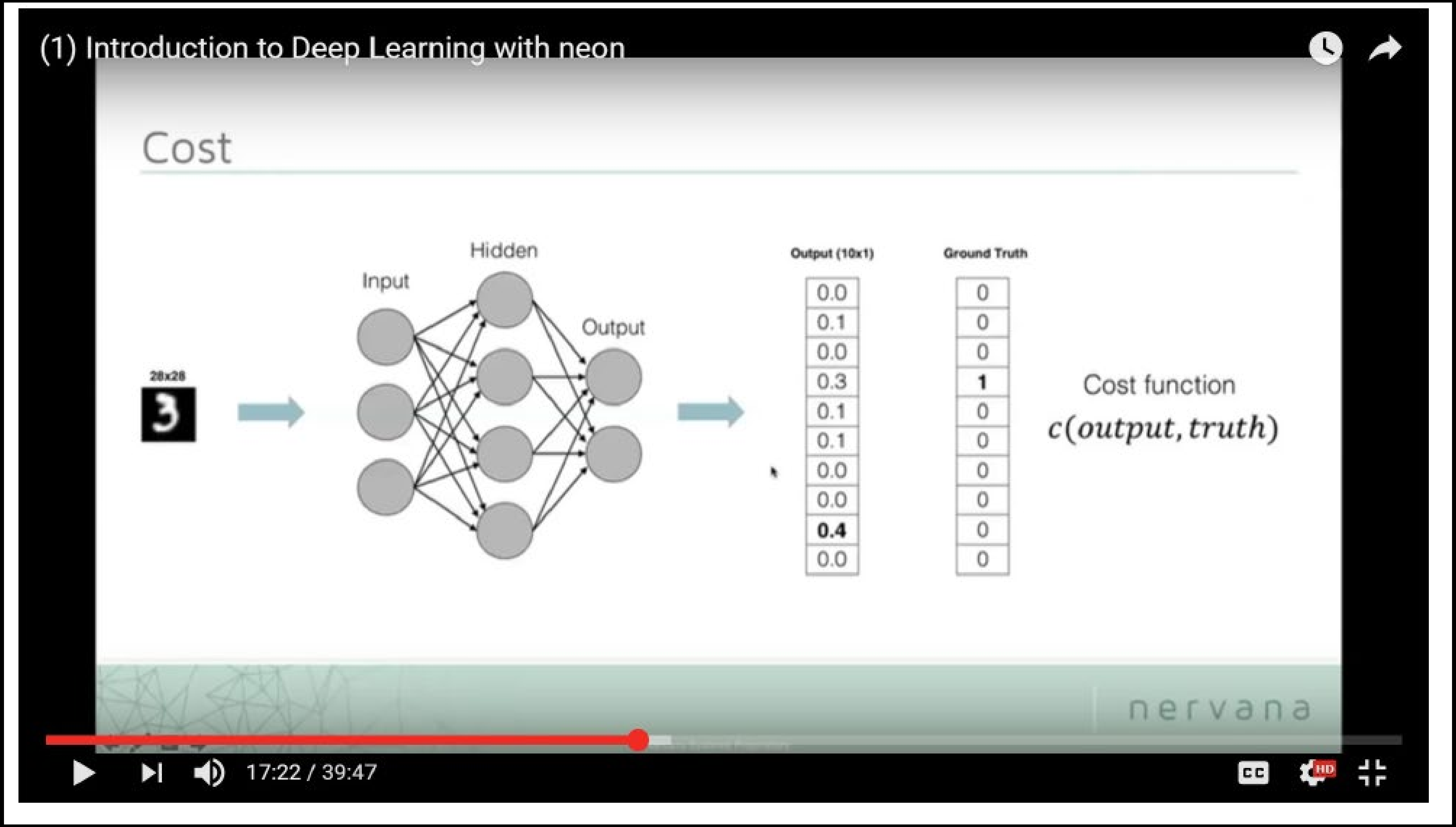


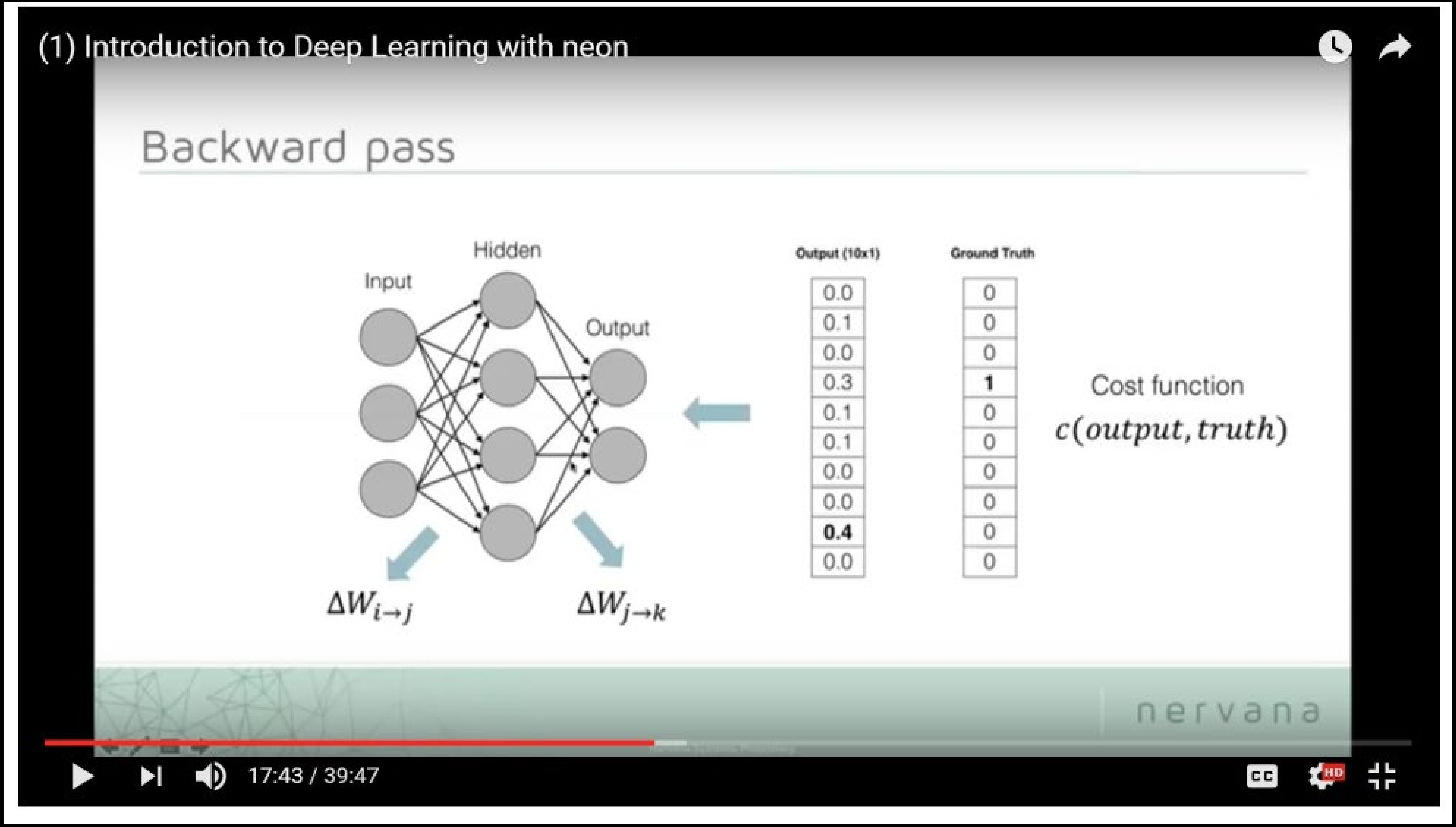
- Reference: Nervana/Intel


(1) Introduction to Deep Learning with neon
Practical example: recognition of handwritten
Layer i
N=784
Layer j
N=100
Layer k
N=10
$W_{i\to j}, b_j$
$W_{j\to k}, b_k$
Total parameters:
$W_{i\to j}$ 784 x 100
b_j 100
$W_{j\to k}$ 100 x 10
b_k 10
nervana
14:49
14:20 / 39:47


(1) Introduction to Deep Learning with neon
Training procedure
Hidden
Input
Output
1. Randomly seed weights
2. Forward-pass
3. Cost
4. Backward-pass
nervana
15:28 / 39:47


(1) Introduction to Deep Learning with neon
Forward pass
28x28
Input
Hidden
Output
Output (10x1)
0.0
0.1
0.0
0.3
0.1
0.1
0.0
0.0
0.4
0.0
$y = g\left(\sum_j W_j x_j + b\right)$
nervana
16:39 / 39:47


(1) Introduction to Deep Learning with neon
Cost
28x28
Input
Hidden
Output
Output (10x1)
0.0
0.1
0.0
0.3
0.1
0.1
0.0
0.0
0.4
0.0
Ground Truth
0
0
0
1
0
0
0
0
0
0
Cost function
c(output, truth)
nervana
17:22 / 39:47


(1) Introduction to Deep Learning with neon
Backward pass
Input
Hidden
Output
$\Delta W_{i \to j}$
$\Delta W_{j \to k}$
Output (10x1)
0.0
0.1
0.0
0.3
0.1
0.1
0.0
0.0
0.4
0.0
Ground Truth
0
0
0
1
0
0
0
0
0
0
Cost function
$c(output, truth)$
nervana
17:43 / 39:47

- Initialize connection weights randomly
- Use differentiable activation function
  - Sigmoid
  - Hyperbolic tangent (tanh)
  - Rectified Linear Unit (ReLu)

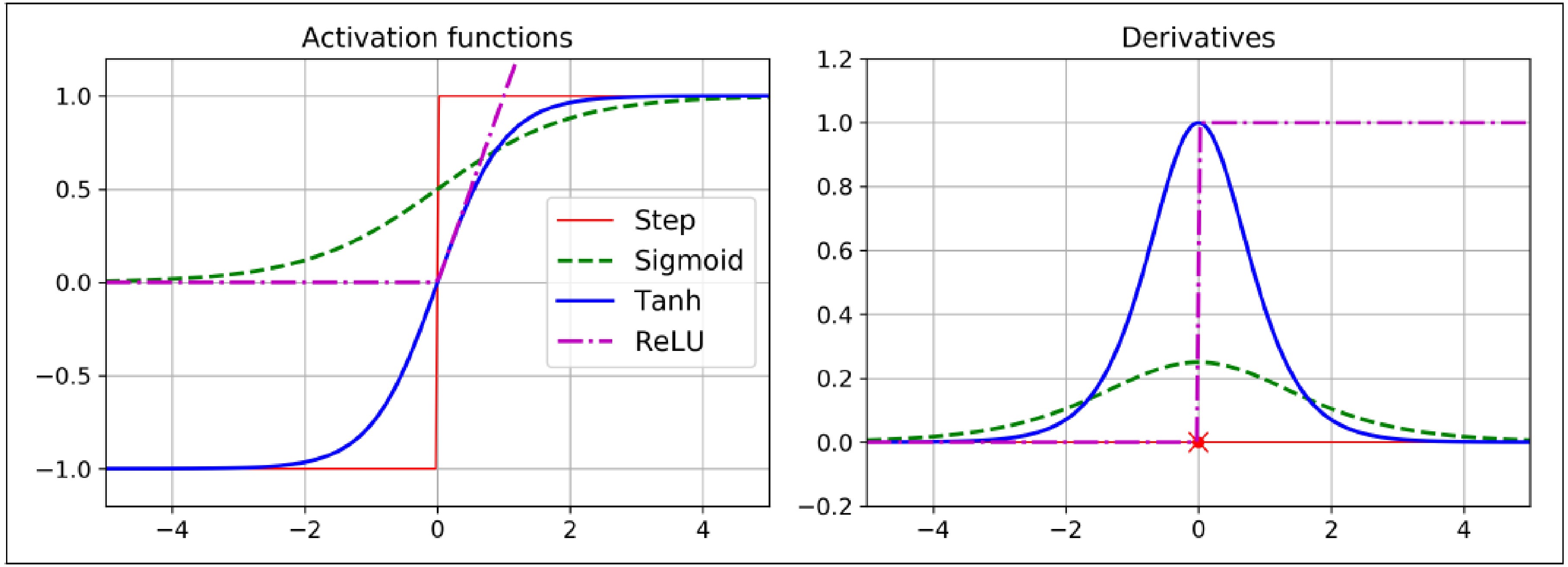


Figure 10-8. Activation functions and their derivatives

MLP for Regression

Table 10-1. Typical Regression MLP Architecture

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

MLP for Classification

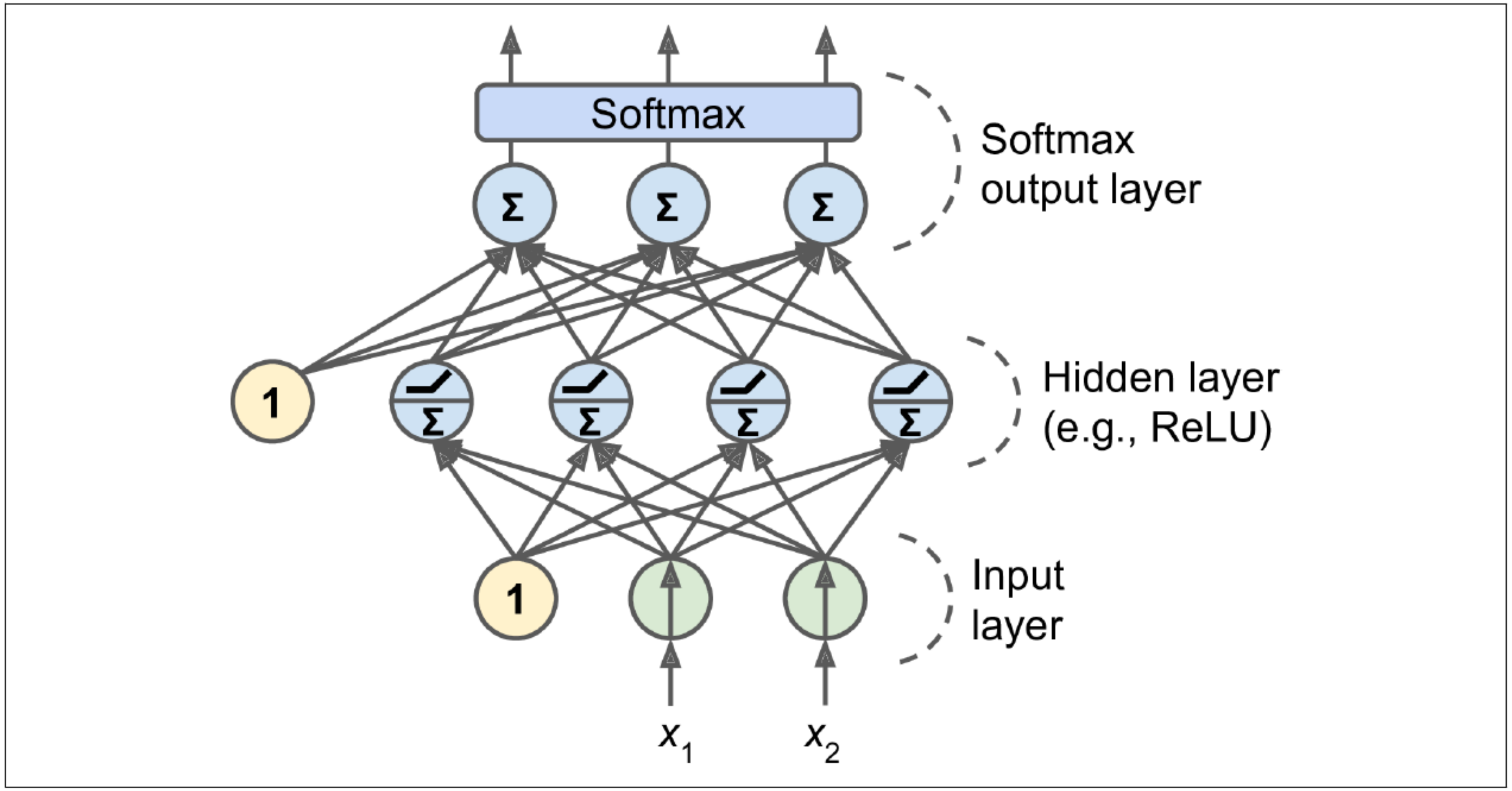


Figure 10-9. A modern MLP (including ReLU and softmax) for classification

Table 10-2. Typical Classification MLP Architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

Fine-Tuning NN Hyperparameters

- A lot of hyperparameters to tweak
  - Grid Search and Randomized Search
- Many companies offer services for hyperparameter optimization
- Active area of research
  - Evolutionary algorithms

- Some guidelines on:
  - Number of hidden layers
  - Number of neurons per hidden layer
  - Other hyperparameters

Number of Hidden Layers

- Start with a single hidden layer, gradually increase the number of layers. Stop just before overfitting
- For complex problems, higher number of hidden layers may be needed
  - Lower layers model low-level structures (line segments, shapes, orientations)
  - Intermediate layers model intermediate structures (squares, circles)
  - Higher layers model high-level structures (faces)
- Transfer learning
  - Start with pre-trained lower layers
  - Train higher layers with domain specific data

Number of Neurons per Hidden Layer

- Data determines the size of input and output layers
  - For MNIST, 28x28 = 784 input neurons and 10 output neurons
- Common practice: form a *pyramid* with fewer neurons at each layer
  - No longer standard practice
- Finding the perfect number of neurons is not well understood
- Stretch pants approach
  - Build a model with more layers and neurons than needed
  - Use early stopping or dropout to avoid overfitting

Other Hyperparameters

- Learning Rate
  - Start with a large value, then divide by 3 and repeat
- Batch Size
  - Not too large, not too small
  - Typically smaller than 32 but greater than 10
- Activation Function
  - Good choice to choose ReLU activation function as default activation function for all hidden layers.
  - Use early stopping