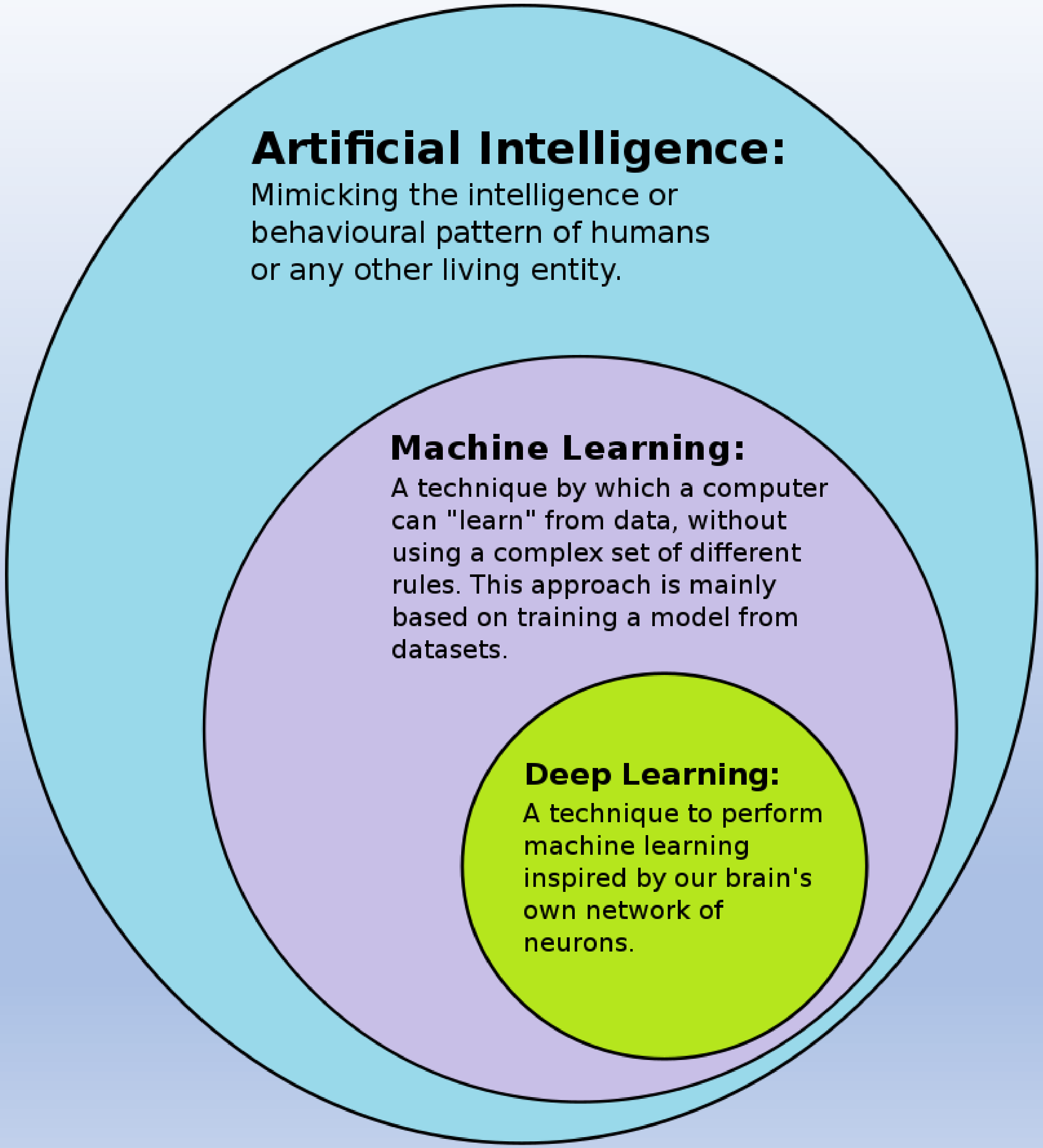
Deep Learning

Unit 1
Fundamentals of Deep Learning

AI Vs ML Vs DL

AI	ML	DL
1956 **<u>John McCarthy</u>** **Machine to Mimic Human Behavior.**	**1959** **<u>Arthur Samuel</u>** **Machine to Learn.** **Training Time Lass** **Testing Time More**	**2000** **<u>Igor Aizenberg</u>** **Algo. Inspired by Structure & Functions of Human Brain.** **Training Time More** **Testing Time Less**


Artificial Intelligence:
Mimicking the intelligence or behavioural pattern of humans or any other living entity.
Machine Learning:
A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.
Deep Learning:
A technique to perform machine learning inspired by our brain's own network of neurons.

What is Deep Learning ?

- A type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data.
- Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning.

Deep Learning

Why ?	What ?	Where ?
Huge Amount of Data.	Handle huge amount of Structured and Unstructured data.	Medical field
Complex problems	Complex Operations, Problems Solved	Self driving Cars
Feature Extraction		Translation

Multilayer Perceptron

- Perceptron is the most commonly used to learn Machine Learning and Deep Learning technologies.
- Which consists of : a set of weights, input values or scores, and a threshold.
- ***Perceptron is a building block of an Artificial Neural Network***.
- Mid of 19th century, **Mr. Frank Rosenblatt** invented the Perceptron for performing certain calculations to detect input data capabilities or business intelligence.

- Perceptron is a linear Machine Learning algorithm used for supervised learning for various binary classifiers.
- This algorithm enables neurons to learn elements and processes them one by one during preparation.
- Basic Components of Perceptron:

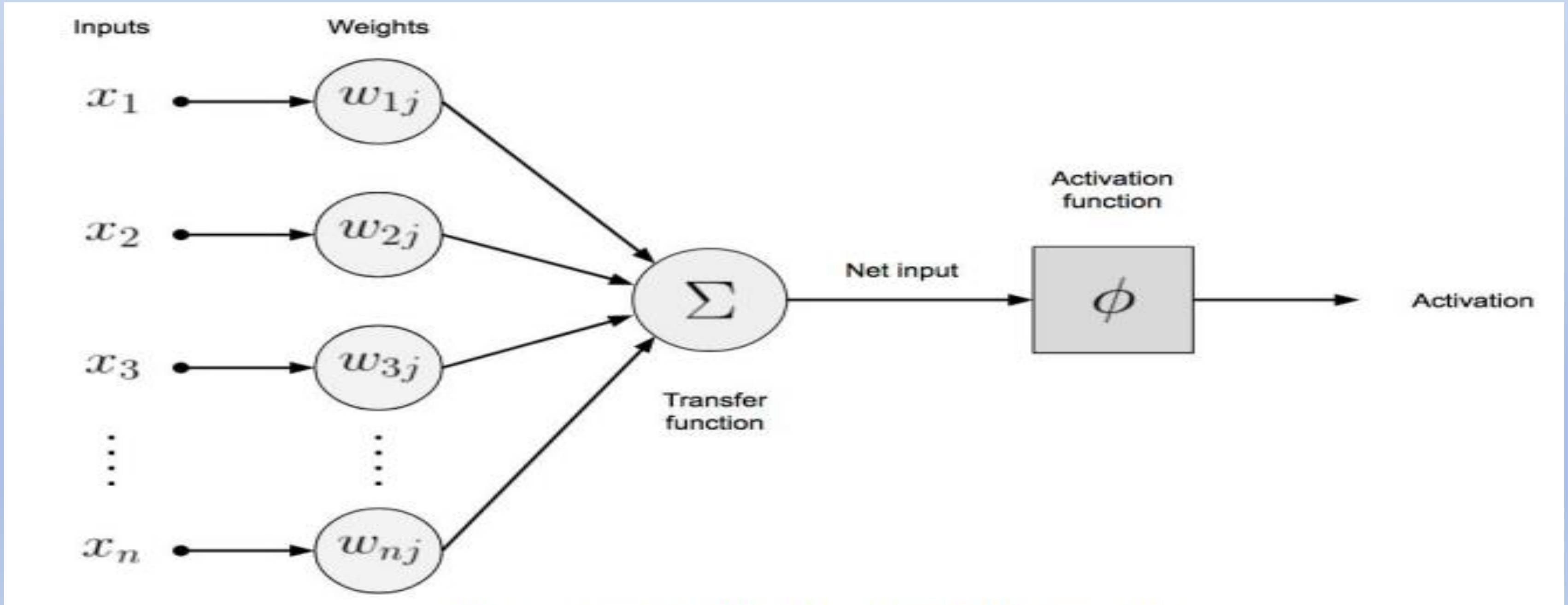
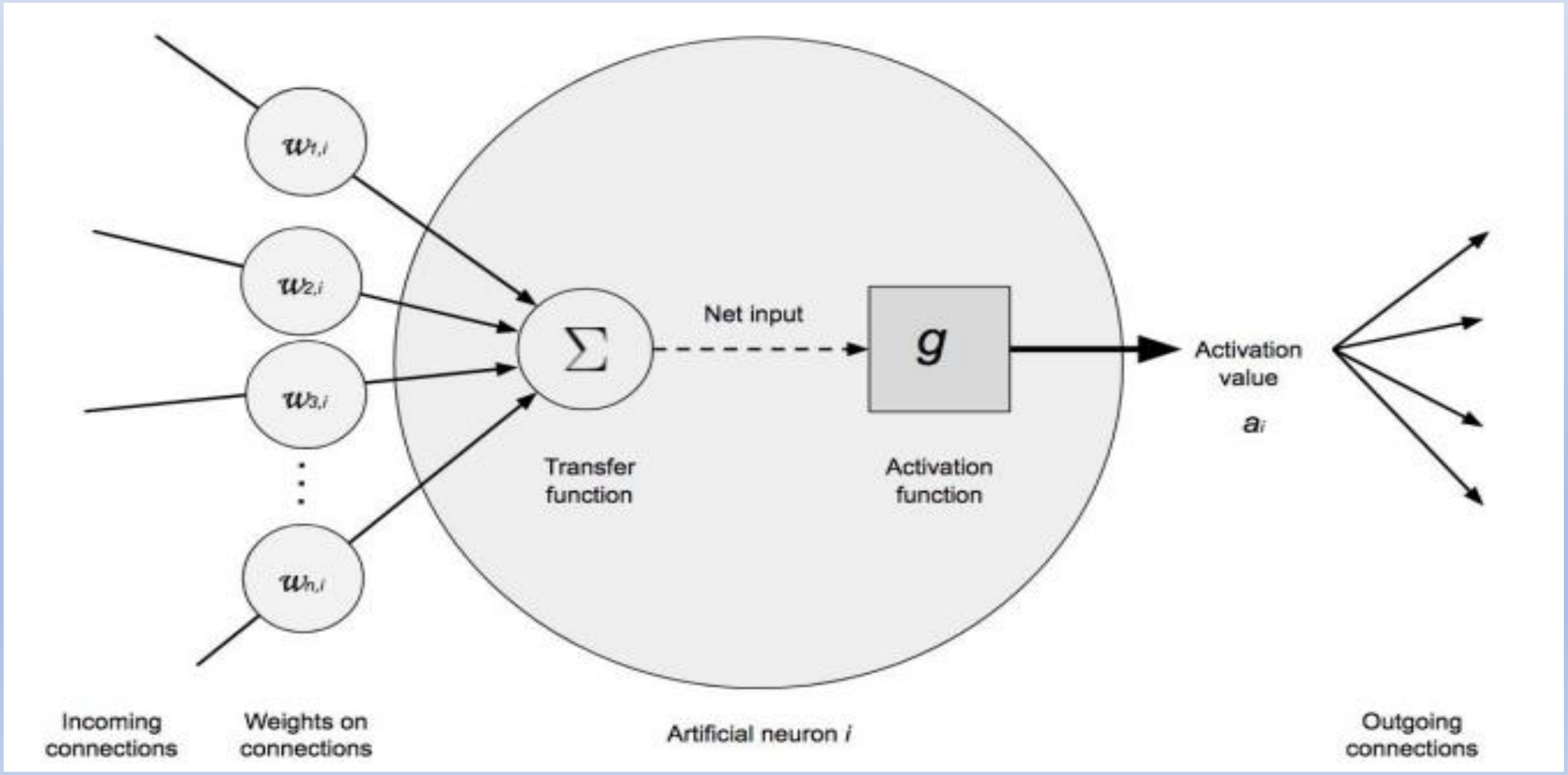


Figure 2-5. Artificial neuron for a multilayer perceptron

- **Input Nodes or Input Layer:**
- This is the primary component of Perceptron.
- Each input node contains a real numerical value.
- **Wight and Bias:**
- Weight parameter represents the strength of the connection between units.
- Weight is directly proportional to the strength of the associated input neuron in deciding the output.
- Biases are scalar values added to the input to ensure that at least a few nodes per layer are activated regardless of signal strength.

- **Activation functions**:
- The functions that govern the artificial neuron’s behavior are called activation functions.
- The transmission of that input is known as forward propagation.
- Activation functions transform the combination of inputs, weights and biases.
- When an artificial neuron passes on a nonzero value to another artificial neuron, it is said to be activated.

- Types of Activation functions:
- Sign function,
- Step function, and
- Sigmoid function.



- The perceptron model begins with the multiplication of all input values and their weights.
- Then adds these values together to create the weighted sum.
- Then this weighted sum is applied to the activation function 'g' to obtain the desired output.
- This activation function is also known as the **step function** and is represented by **'g'**.
- Perceptron model works in two important steps as follows:

- **Step-1**
- In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:
- ∑wi*xi = x1*w1j + x2*w2j +…wn*xn
- Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.
- **∑wij*xi + b**
- **Step-2**
- An activation function is applied ,output either in binary form or a continuous value as follows:
- **Y = g(∑wij*xi + b)**

- A multi-layer perceptron model has a greater number of hidden layers.
- The multi-layer perceptron model is also known as the Backpropagation algorithm.
- Which executes in two stages as follows:
- **Forward Stage:**
- Activation functions start from the input layer in the forward stage and terminate on the output layer.
- **Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement.

- **Advantages of Multi-Layer Perceptron:**
- Can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

- **Disadvantages of Multi-Layer Perceptron:**
- Computations are difficult and time-consuming.
- Difficult to predict how much the dependent variable affects each independent variable.
- Model functioning depends on the quality of the training.

Feed-forward neural network

- Multilayer feed-forward neural networks, have artificial neurons arranged into groups called layers.
- Multilayer neural network has the following:
- **A single input layer.**
- **One or many hidden layers, fully connected**
- **A single output layer.**

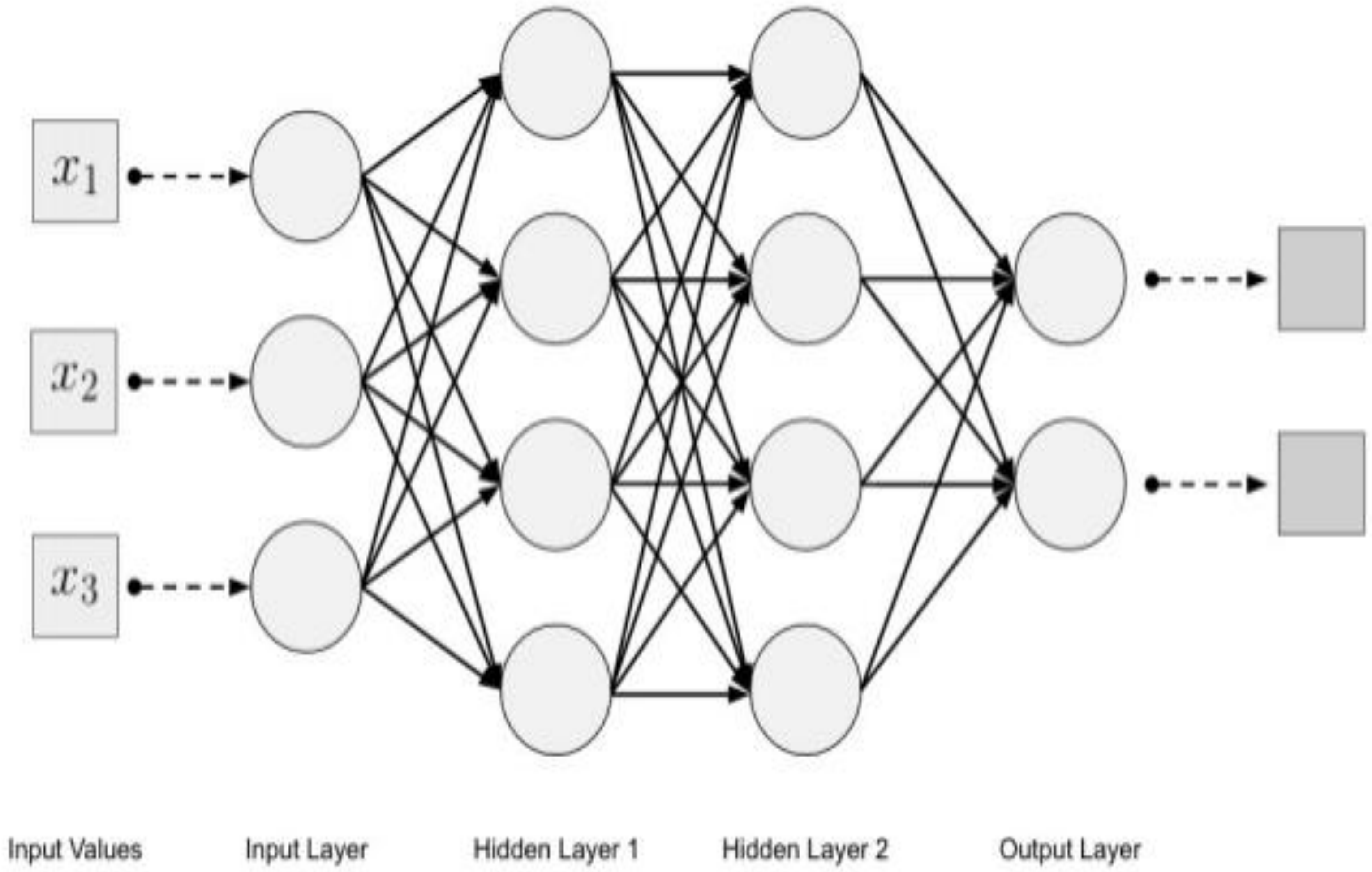


Figure 2-7. Fully connected multilayer feed-forward neural network topology

- **Input layer:**
- The number of neurons in an input layer is typically the same number as the input feature to the network.
- Input layers are followed by one or more hidden layers.
- Input layers in classical feed-forward neural networks are fully connected to the next hidden layer.
- **Hidden layer:**
- There are one or more hidden layers in a feed-forward neural network.

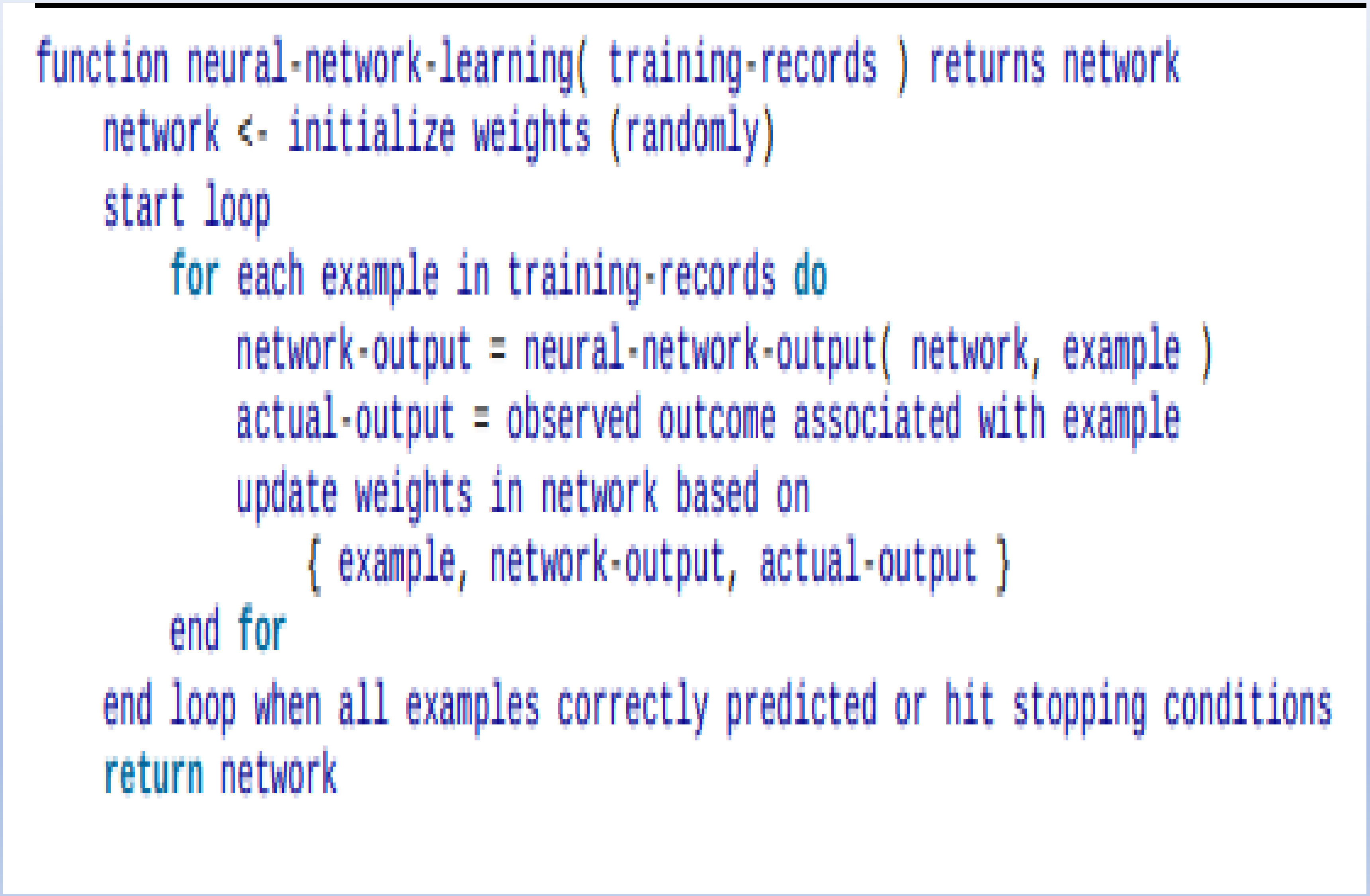
- The weight values on the connections between the layers are how neural networks encode the learned information extracted from the raw training data.
- Hidden layers are the key to allowing neural networks to model nonlinear functions.
- **Output layer:**
- We get the answer or prediction from our model from the output layer.
- Depending on the setup of the neural network, the final output may be a real-valued output (regression) or a set of probabilities (classification).

- This is controlled by the activation function.
- The output layer typically uses either a softmax or sigmoid activation function for classification.
- <u>**Connections between layers :**</u>
- In a fully connected feed-forward network, the connections between layers are the outgoing connections from all neurons, previous layer to all next layer.
- We change these weights progressively as our algorithm finds the best solution it can with the backpropagation learning algorithm.

Backpropagation Learning

- Backpropagation is an important part of reducing error in a neural network model.
- Backpropagation, how information circulates within a feed-foward neural network.
- **Algorithm intuition :**
- Backpropagation learning is similar to the perceptron learning algorithm.
- Compute the input example’s output with a forward pass through the network.
- If the output matches the label, we don’t do anything.
- If the output does not match the label, we need to adjust the weights on the connections in the neural network.

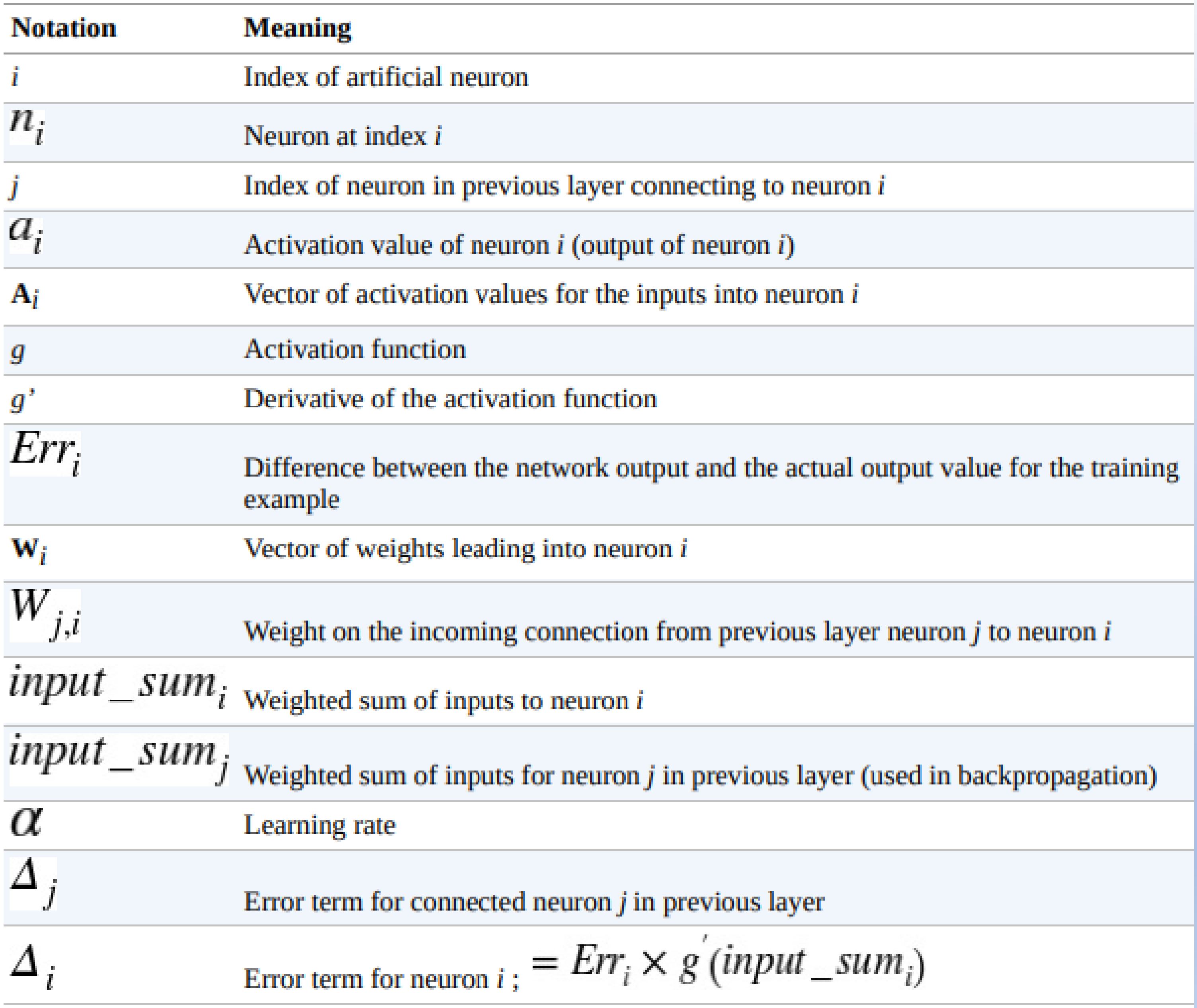
- General neural network training pseudo code :



```
function neural-network-learning( training-records ) returns network
    network <- initialize weights (randomly)
    start loop
        for each example in training-records do
            network-output = neural-network-output( network, example )
            actual-output = observed outcome associated with example
            update weights in network based on
                { example, network-output, actual-output }
        end for
    end loop when all examples correctly predicted or hit stopping conditions
    return network
```

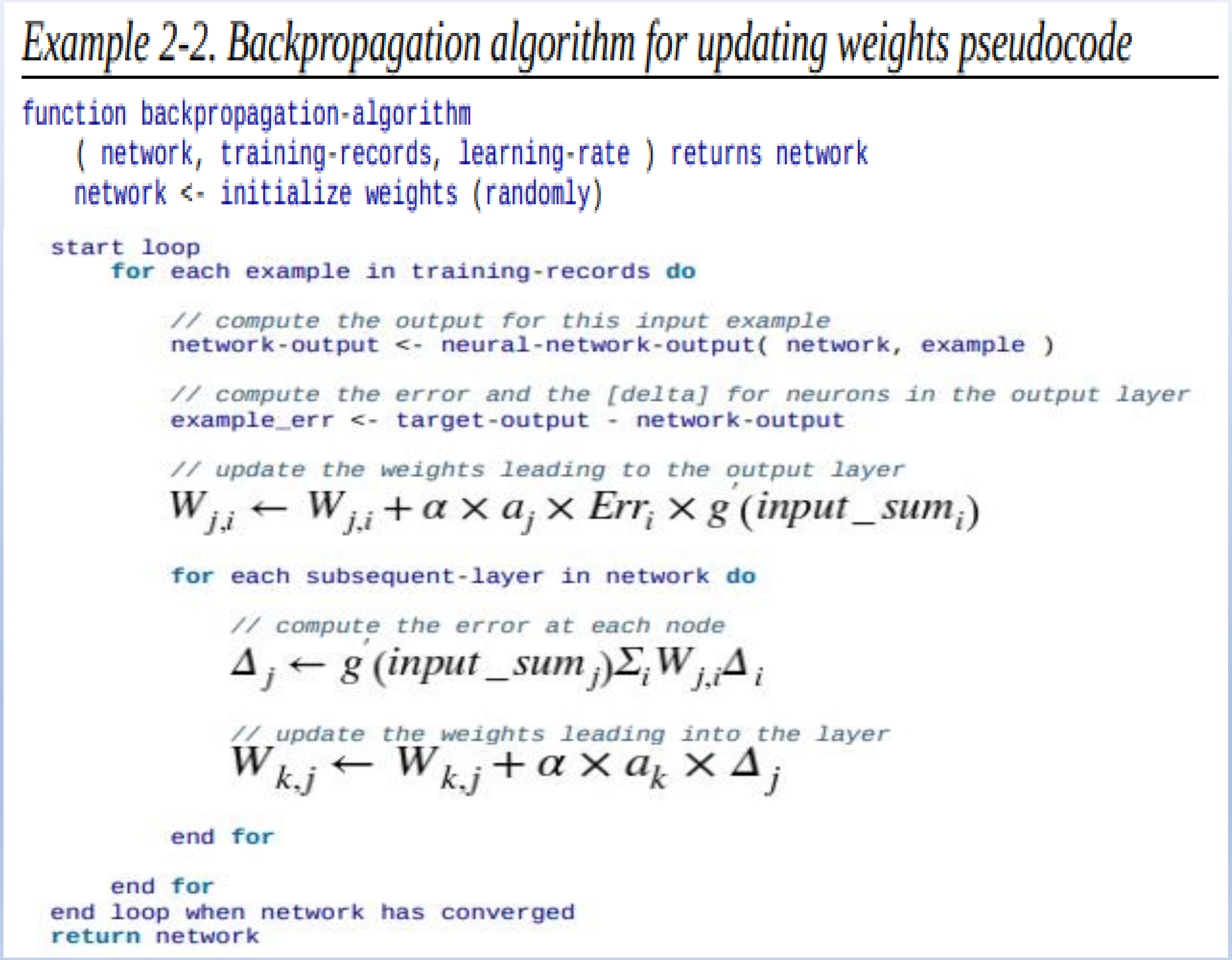
- With the perceptron learning algorithm, it’s easy because there is only one weight per input to influence the output value.
- With feed forward multilayer networks learning algorithms, many weights connecting each input to the output, so it becomes more difficult.
- Each weight contributes to more than one output.
- With backpropagation,minimize the error between the label (or “actual”) output associated with the training input and the value generated from the network output.

Table 2-2. Neural network notation



Notation	Meaning
i	Index of artificial neuron
n_i	Neuron at index *i*
j	Index of neuron in previous layer connecting to neuron *i*
a_i	Activation value of neuron *i* (output of neuron *i*)
$\mathbf{A}_i$	Vector of activation values for the inputs into neuron *i*
g	Activation function
g'	Derivative of the activation function
Err_i	Difference between the network output and the actual output value for the training example
$\mathbf{W}_i$	Vector of weights leading into neuron *i*
$W_{j,i}$	Weight on the incoming connection from previous layer neuron *j* to neuron *i*
$input\_sum_i$	Weighted sum of inputs to neuron *i*
$input\_sum_j$	Weighted sum of inputs for neuron *j* in previous layer (used in backpropagation)
α	Learning rate
Δ_j	Error term for connected neuron *j* in previous layer
Δ_i	Error term for neuron *i* ; $= Err_i \times g'(input\_sum_i)$

Backpropagation algorithm for updating weights pseudo code



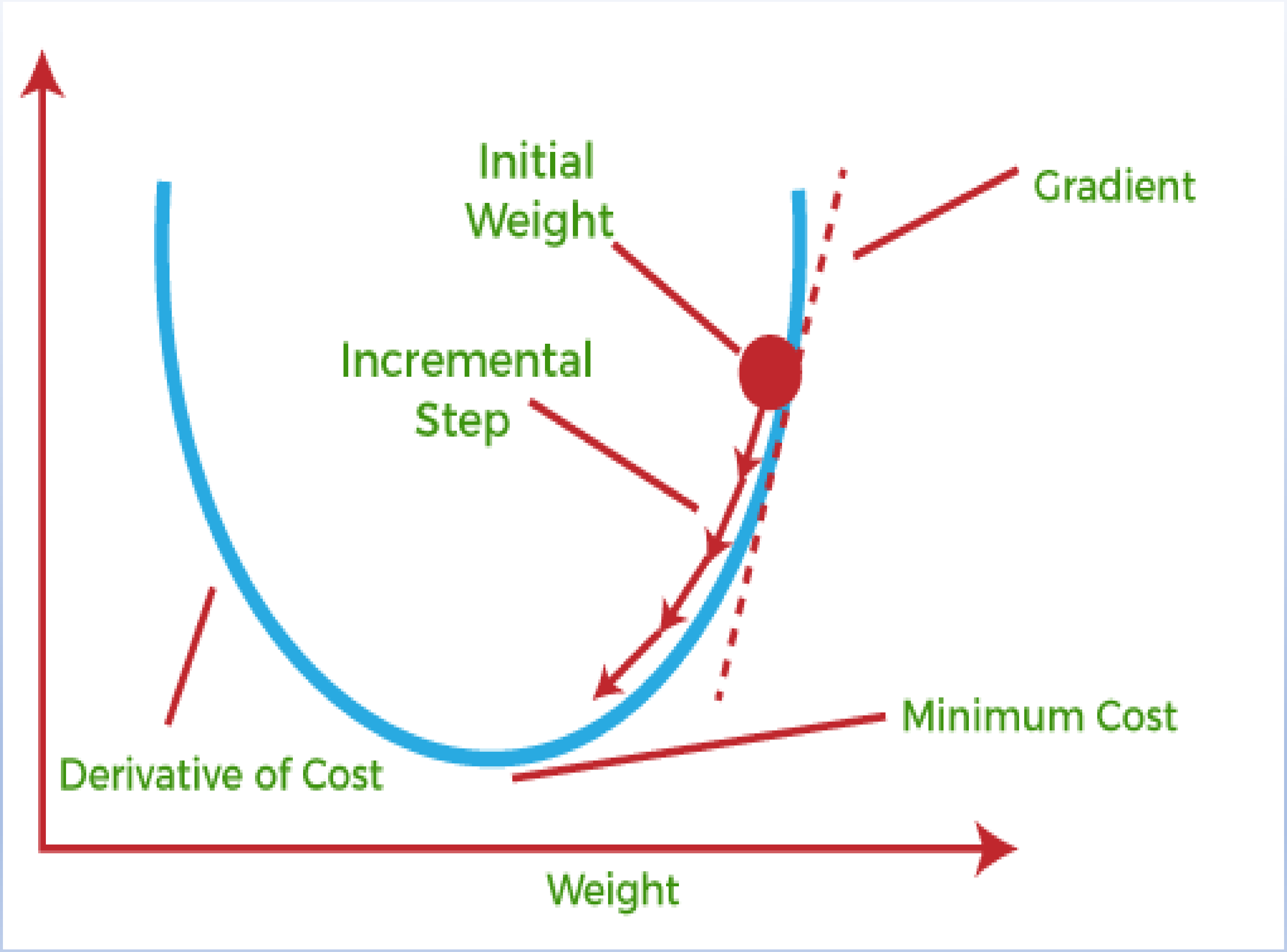
Example 2-2. Backpropagation algorithm for updating weights pseudocode

```
function backpropagation-algorithm
    ( network, training-records, learning-rate ) returns network
    network <- initialize weights (randomly)

  start loop
      for each example in training-records do

          // compute the output for this input example
          network-output <- neural-network-output( network, example )

          // compute the error and the [delta] for neurons in the output layer
          example_err <- target-output - network-output

          // update the weights leading to the output layer
          $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err_i \times g'(input\_sum_i)$

          for each subsequent-layer in network do

              // compute the error at each node
              $\Delta_j \leftarrow g'(input\_sum_j)\Sigma_i W_{j,i}\Delta_i$

              // update the weights leading into the layer
              $W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$

          end for

      end for
  end loop when network has converged
  return network
```

Gradient Descent

Gradient Descent

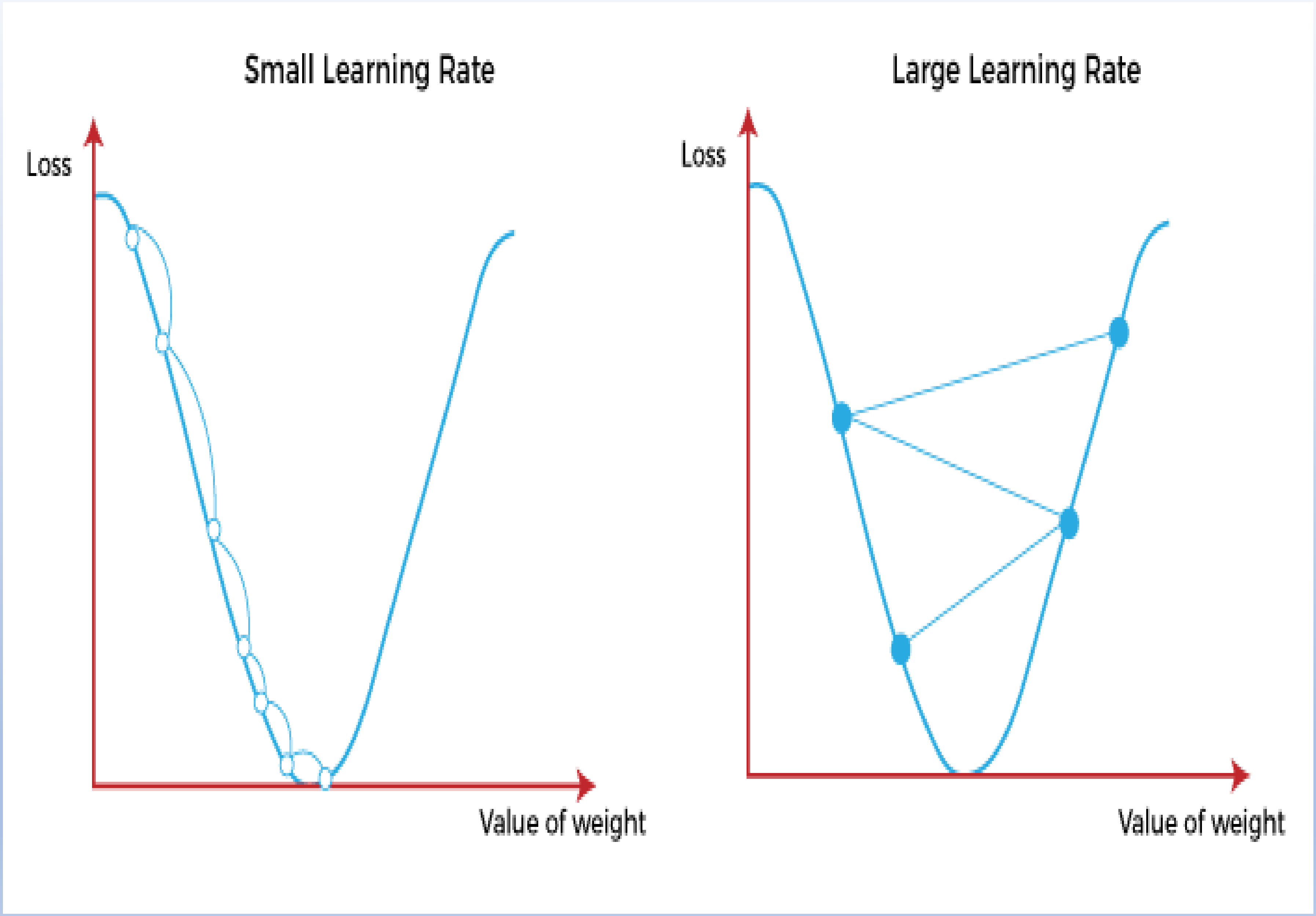
- Gradient descent (**steepest descent**) discovered by **"Augustin-Louis Cauchy"** in mid of 18th century.
- ***Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.***
- What is Cost-function?
- ***The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number.***


Initial Weight
Gradient
Incremental Step
Minimum Cost
Derivative of Cost
Weight

- Minimize error and find the local or global minimum.
- Imagine the quality of our network’s predictions.
- **Hills** represent locations (parameter values or weights) lot of prediction error; **valleys** represent locations with less error.
- Choose one point on that landscape -initial weight.
- Select the initial weight based on domain knowledge.(initial weights randomly).
- Move weight downhill, areas of lower error, as quickly as possible.
- Gradient descent can sense the actual slope of the hills with regard to each weight.

- GD measures the slope (the change in error caused by a change in the weight) and takes the weight one step toward the bottom of the valley.
- Derivative of the loss function to produce the gradient.
- Derivative measures “rate of change” of a function.
- Stop in the point of greatest accuracy.
- Convex loss function has only a global minimum.
- Process of measuring loss and changing the weight by one step in the direction of less error is repeated until the weight arrives at a point beyond which it cannot go lower.

- **Learning Rate:**
- Step size taken to reach the minimum or lowest point.
- Typically a small value that is evaluated and updated based on the behavior of the cost function.
- High Learning rate is, results in larger steps but leads to risks of overshooting the minimum.
- Low learning rate shows small step sizes, advantage of more precision.


Small Learning Rate
Loss
Value of weight
Large Learning Rate
Loss
Value of weight

Types of Gradient Descent

- **Batch Gradient Descent.**
- **Stochastic Gradient Descent.**
- **Mini-Batch Gradient Descent.**

Batch Gradient Descent

- Used to find the error for each point in the training set and update the model after evaluating all training examples.
- Procedure is known as **training epoch.**
- **Advantages:**
- Produces less noise.
- Produces stable gradient descent convergence.
- Computationally efficient as all resources are used for all training samples.

Stochastic Gradient Descent

- Runs one training example per iteration.
- Requires only one training example at a time.
- Easier to store in allocated memory.
- Frequent updates, treated as a noisy gradient.
- **Advantages :**
- Easier to allocate in desired memory.
- Relatively fast to compute.
- More efficient for large datasets.

MiniBatch Gradient Descent:

- Combination of both batch gradient descent and stochastic gradient descent.
- Divides the training datasets into small batch sizes.
- Performs the updates on those batches separately.
- MiniBatch Gradient Descent tends to be more stable and less noisy.
- Operations on mini-batches can be parallelized, leading to faster computations.
- **Advantages:**
- Converges faster.

- Requires less memory than BGD - makes it feasible to <u>train large models on large datasets</u>.
- Noise introduced by mini-batches can help in avoiding local minima and finding better general solutions.

VANISHING GRADIENT PROBLEM

- **Recurrent Neural Networks (RNN) :**
- A superset of feed-forward neural networks but they add the concept of recurrent connections.
- Type of artificial neural network which uses sequential data or time series data.
- Vanishing Gradient Problem occurs when the gradients become too large or too small .
- Make it difficult to model long-range dependencies (10 time-steps or more) in the structure of the input dataset.

- The most effective way, use the LSTM variant of Recurrent Neural Networks.
- LSTM(**Long short-term memory)** Artificial neural network.
- LSTM Networks :
- The critical component of the LSTM is the memory cell and the gates (including the forget gate, but also the input gate).
- The contents of the memory cell are modulated by the input gates and forget gates.

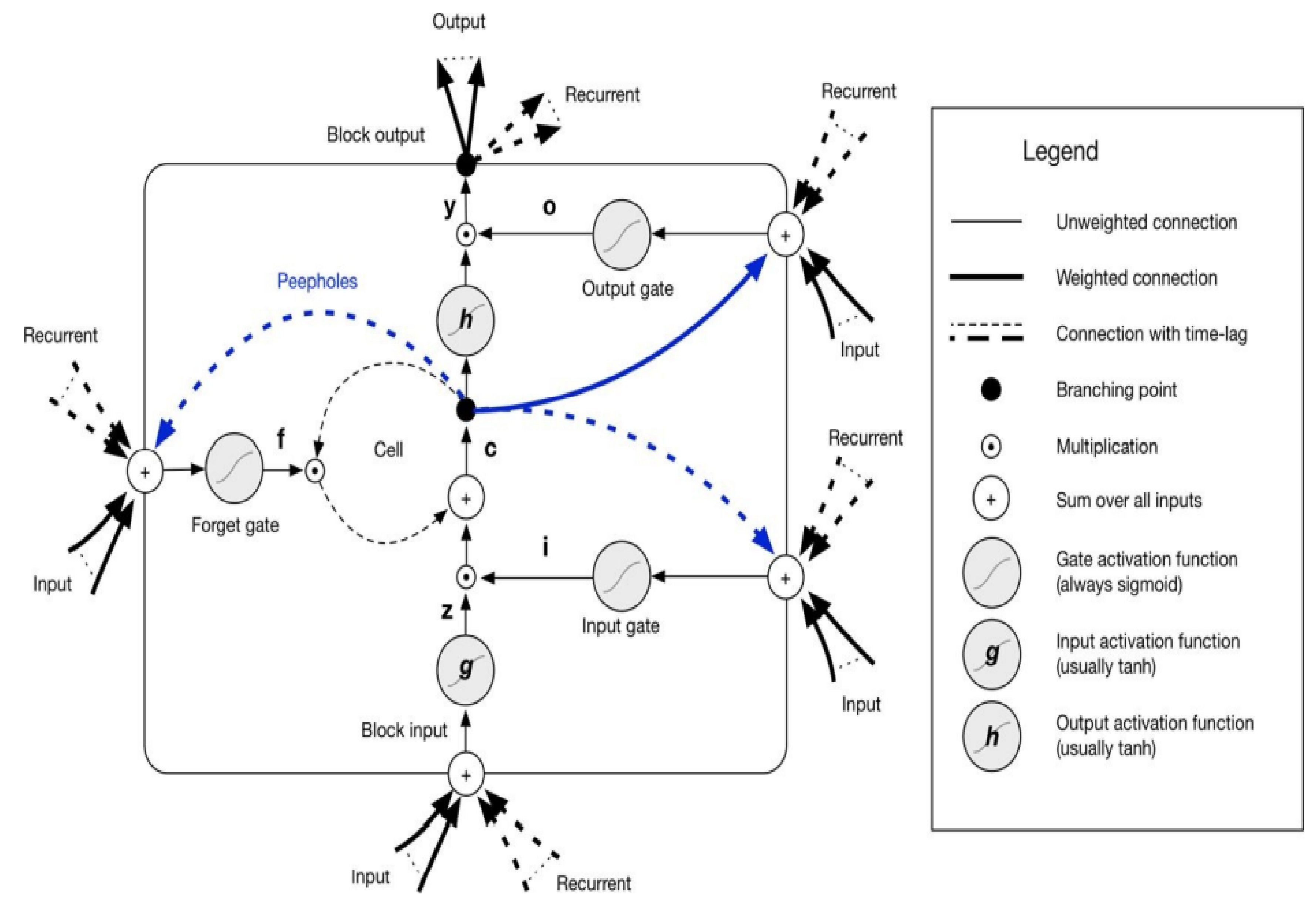


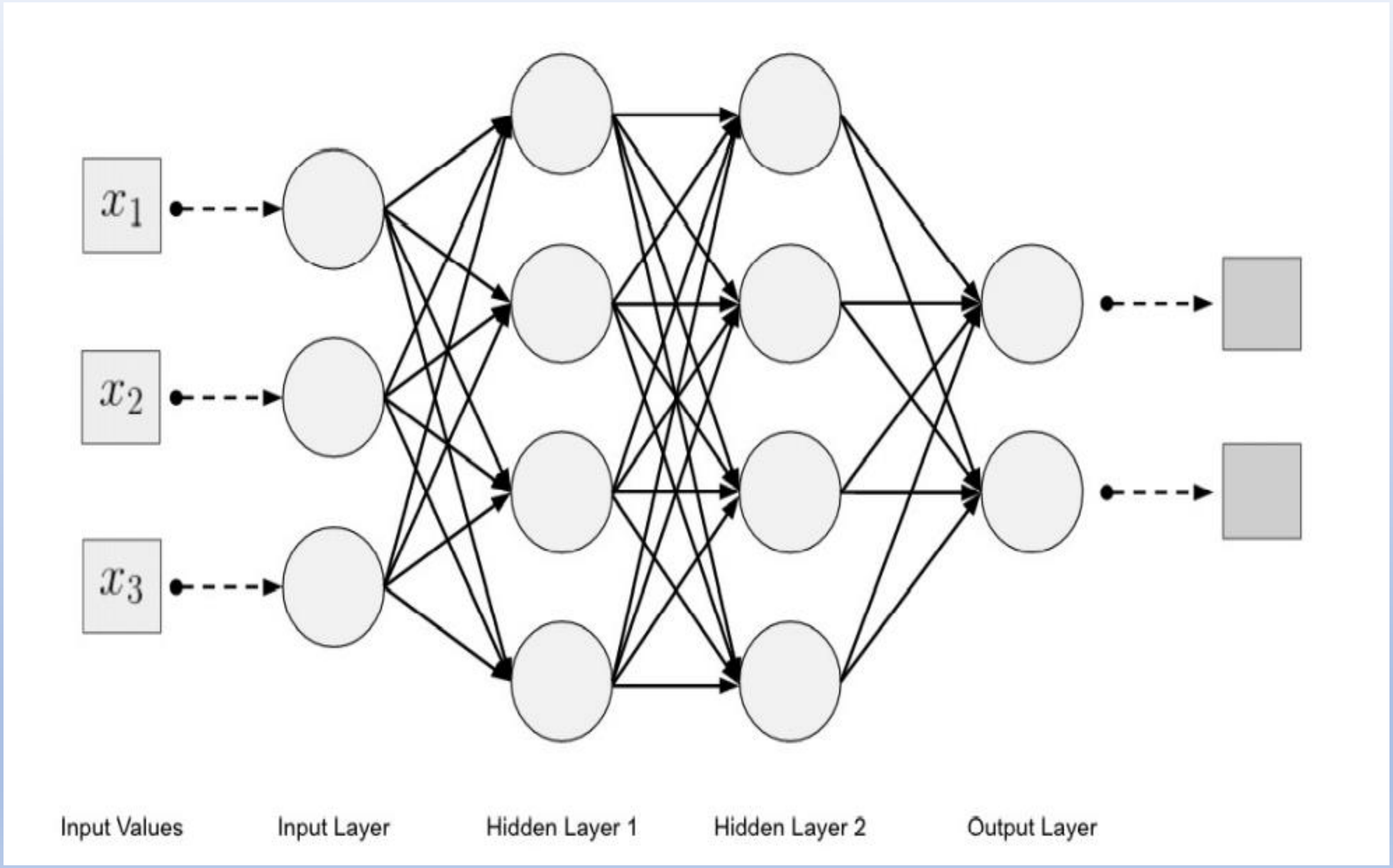
Figure 4-24. LSTM block diagram

- **LSTM unit:**
- Three gates
- input gate (input modulation gate)
- forget gate
- output gate
- Block input
- Memory cell.
- Output activation function
- Peephole connections

- Assuming that both of these gates are closed, the contents of the memory cell will remain unmodified between one time-step and the next.
- The gating structure allows information to be retained across many time-steps, and consequently also allows gradients to flow across many time-steps.
- This allows the LSTM model to overcome the vanishing gradient problem , occurs with most Recurrent Neural Network models.

- <u>Properties of LSTM networks :</u>
- Better update equations
- Better backpropagation
- <u>Example use cases of LSTMs:</u>
- Generating sentences (e.g., character-level language models)
- Classifying time-series
- Speech recognition
- Handwriting recognition
- Polyphonic music modeling

LSTM network architecture



- If we were to show each layer from the network as a single node in a flattened representation.

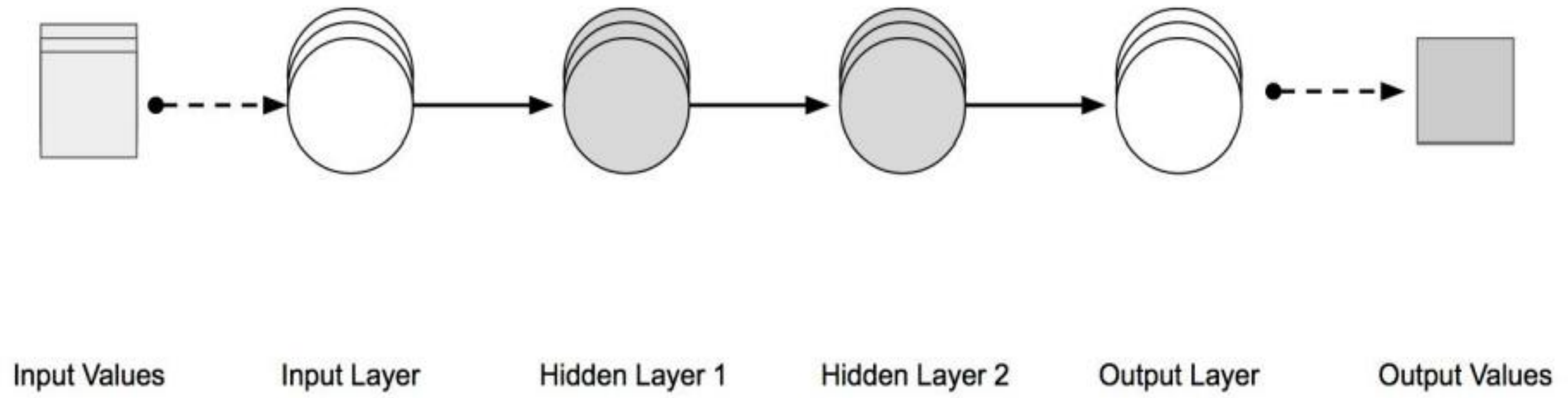


Figure 4-21. Visually feed-forward multilayer network

- With Recurrent Neural Networks, we introduce the idea of a type of connection that connects the output of a hidden-layer neuron as an input to the same hidden layer neuron.
- With this recurrent connection, we can take input from the previous time-step into the neuron as part of the incoming information.

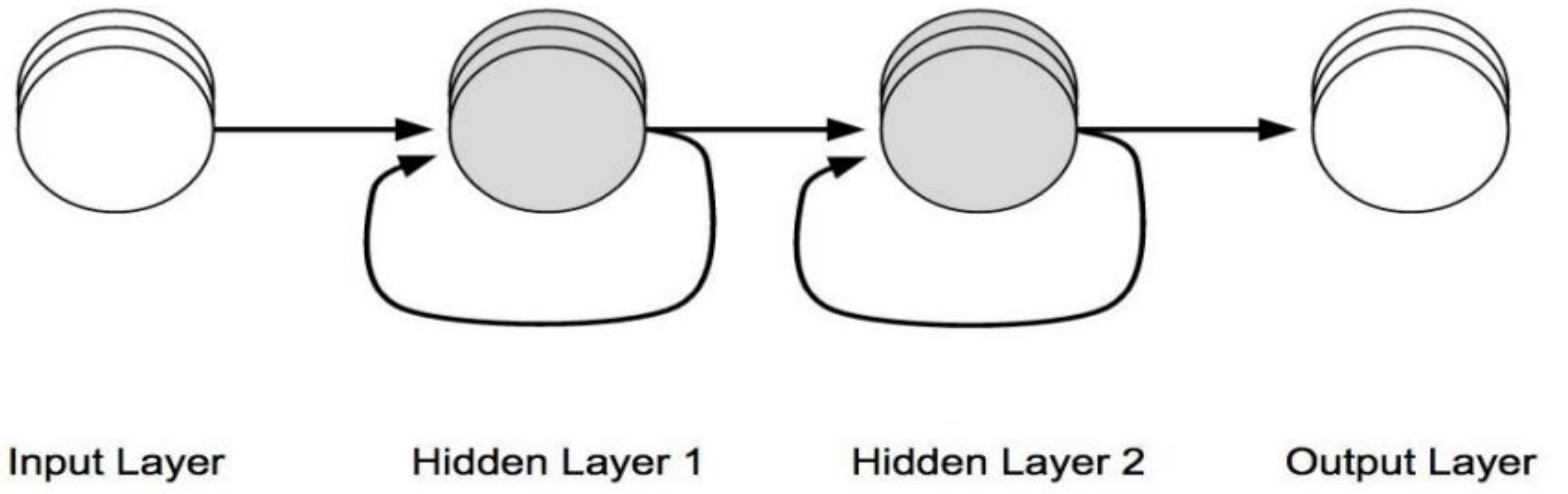
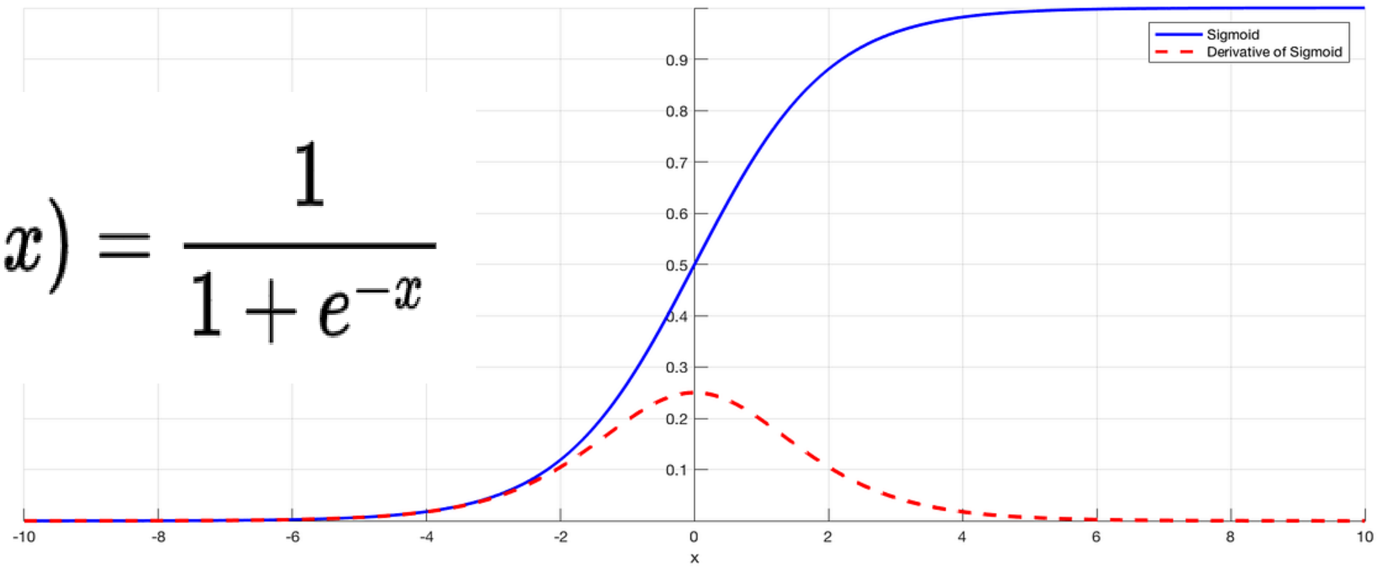


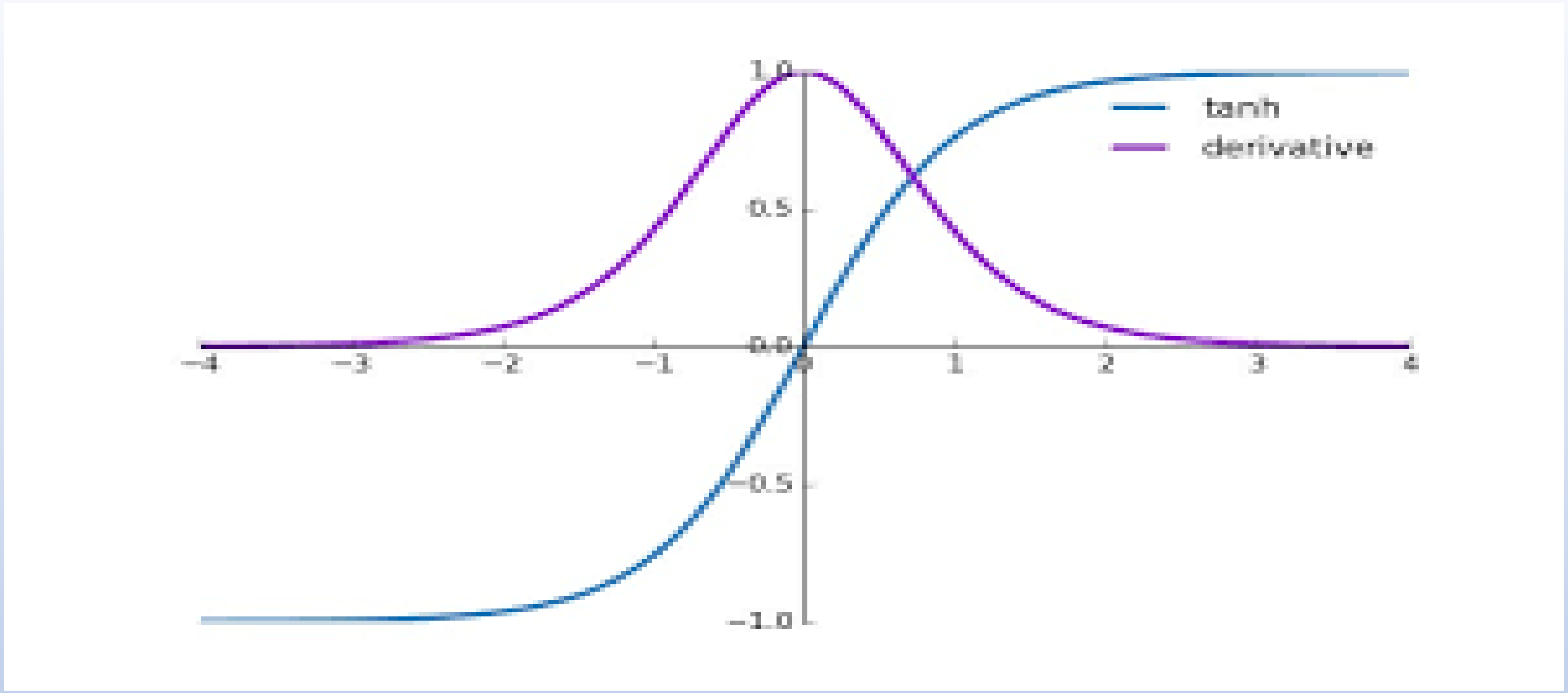
Figure 4-22. Showing recurrent connections on hidden-layer nodes

Sigmoid & Threshold Activation Function



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

- When we pass the value of X to equation then sigmoid A.F. transform the values bet 0 to 1.
- In case backpeapogation we need to calculate derivate of activation function.
- Derivative of sigmoid activation function ranging from 0 to 0.25 (mathematical proven).



tanh
derivative
1.0
0.5
0.0
−0.5
−1.0
−4
−3
−2
−1
0
1
2
3
4

Optimization Algorithms

Optimization Algorithms

- **Optimization :**
- Process of adjusting weights to produce more and more accurate guesses about the data is known as **parameter optimization**.
- Hypothesis, test it against reality, and refine or replace that hypothesis again and again to better describe events in the world.
- Training a model in machine learning involves finding the best set of values for the **parameter vector** of the model.

Optimization Algorithms

- **First – Order.**
- **Second – Order.**

First – Order

- First-order optimization algorithms calculate the Jacobian matrix.
- Jacobian : Matrix of partial derivatives of loss function values with respect to each parameter.
- Algorithm takes one step in the direction specified by the Jacobian.
- First-order methods calculate a gradient (Jacobian) at each step to determine which direction to go in next.
- Optimization algorithm - "search."
- Finding a path toward minimal error.

Jacobian Matrix

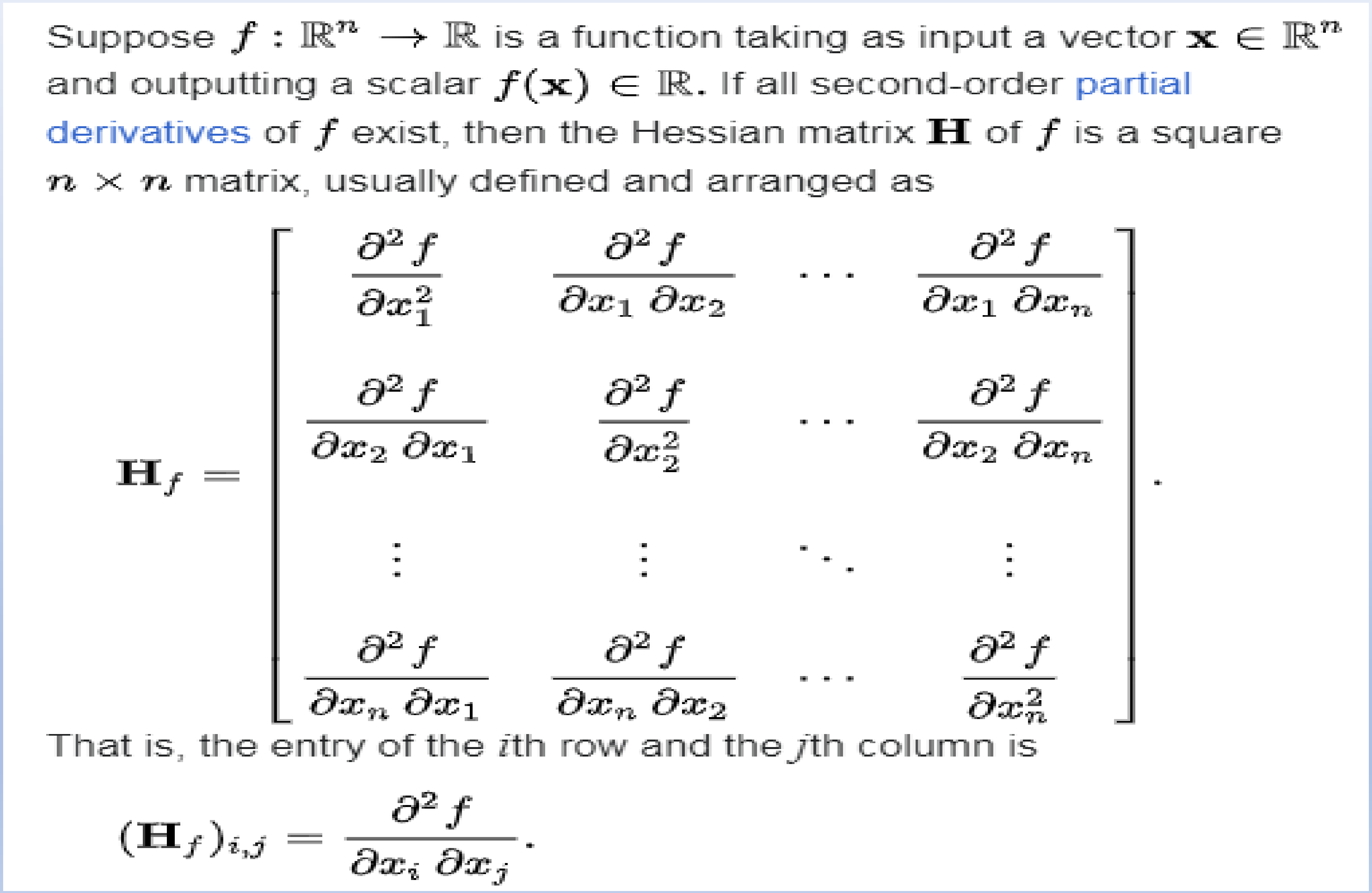
Suppose we have a vector-valued function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ that maps an n-dimensional input vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ to an m-dimensional output vector $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))$. The Jacobian matrix J of $\mathbf{f}$ is an $m \times n$ matrix where each element J_{ij} represents the partial derivative of the i-th output function with respect to the j-th input variable:

$$\mathbf{J} = \frac{df(x)}{dx} = \left[\frac{\partial f(x)}{\partial x_1} \cdots \frac{\partial f(x)}{\partial x_u} \right]$$

Second-order methods

- Second-order methods calculate or approximate the Hessian, the derivative of the Jacobian.
- A matrix of second-order- "tracking acceleration rather than speed."
- Hessian's job is to describe the curvature of each point of the Jacobian.
- Second-order methods include:
- Limited-memory BFGS (L-BFGS).
- Conjugate gradient
- Hessian-free

Hessian Matrix



Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a function taking as input a vector $\mathbf{x} \in \mathbb{R}^n$ and outputting a scalar $f(\mathbf{x}) \in \mathbb{R}$. If all second-order partial derivatives of f exist, then the Hessian matrix $\mathbf{H}$ of f is a square $n \times n$ matrix, usually defined and arranged as

$$\mathbf{H}_f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\[2.2ex] \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\[2.2ex] \vdots & \vdots & \ddots & \vdots \\[2.2ex] \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

That is, the entry of the ith row and the jth column is

$$(\mathbf{H}_f)_{i,j} = \frac{\partial^2 f}{\partial x_i\,\partial x_j}.$$

Limited-memory BFGS

- A specialized version of the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.
- Used for solving unconstrained optimization problems.
- Useful for large-scale optimization tasks -it reduces the memory requirements.
- **Memory Efficiency**:
- L-BFGS only storing a limited number of vectors that represent updates to the position and gradient, rather than the entire Hessian matrix.

- L-BFGS stores information from the last ***m*** iterations, where ***m*** is a small integer (e.g., 5 or 10).
- This significantly reduces memory usage, making it feasible to handle large datasets or high-dimensional parameter spaces.
- **Quasi-Newton Method**:
- **(**Class of iterative optimization algorithms used to find the local maxima or minima of functions.**)**
- L-BFGS approximates the Hessian matrix.
- Approximation is used to determine the direction and size of the steps taken towards the minimum of the objective function.

- Approximation is updated iteratively using the BFGS update formula.
- Only the limited recent history of the gradients and positions is maintained.

- L-BFGS typically converges faster than simple gradient descent methods.
- **Applications**:
- For training models like
- logistic regression, support vector machines, and neural networks.

- **Algorithm Outline**:
- **Initialization**: Start with an initial guess for the parameters and choose the number of vectors ***m*** to store.
- **Iteration**:
  - Calculate the gradient of the objective function at the current position.
  - Update the position using the approximate Hessian.
  - Store the new position and gradient difference vectors if they are within the last ***m*** iterations.
- **Convergence Check**: The process repeats until convergence criteria, such as a small gradient norm or minimal change in the objective function, are met.

Conjugate Gradient

- The Conjugate Gradient (CG) method is an iterative algorithm used to solve large-scale linear systems of equations and unconstrained optimization problems.
- Highly efficient for problems where the matrix involved is too large to be stored or manipulated directly.
- Find the minimum of the quadratic function with an iterative method called conjugate gradient.

- The conjugate gradient method (CG) was originally invented to minimize a quadratic function:

$$F(x) = \frac{1}{2}x^T A x - bx$$

- where A is an n×n symmetric positive definite matrix, x and b are n×1 vectors.
- The solution to the minimization problem is equivalent to solving the linear system, i.e. determining x when ∇F(x)=0, i.e. Ax−b=0.

- CG being between:
- **Newton’s method** a second-order method that incorporates Hessian and gradient,
- and the method of **steepest descent**, a first-order method that uses gradient.
- To avoid the high computational cost of Newton’s method and to accelerate the convergence rate of steepest descent, the conjugate gradient method was developed.
- The idea of the CG method is to pick n orthogonal search directions first.
- In each search direction, take exactly one step such that the step size is orthogonal to the proposed solution x at that direction.

- The solution is reached after n steps.
- Theoretically, the number of iterations needed by the CG method is equal to the number of different eigenvalues of A, i.e. at most n.

- **Hessian-free**
- Hessian-free optimization is related to Newton's method, but it better minimizes the quadratic function.
- It is a powerful optimization method adapted to neural networks by James Martens in 2010.
- Technique that avoids direct computation and storage of the Hessian matrix.
- In Hessian-free optimization, the algorithm uses a technique called **conjugate gradient** to approximate the inverse Hessian-vector product.
- Efficient - It does not require the full Hessian matrix but only the product of the Hessian with a vector.

Hyperparameters

- Hyperparameters are parameters that control the learning process of a machine learning model.
- They influence how the model learns and performs but are not adjusted through the training process.
- Model parameters (weights and biases) ,which are learned during training.
- Hyperparameters need to be set manually or optimized using specific techniques.

- **Hyper parameters fall into several categories:**
- Layer size .
- Magnitude (momentum, learning rate)
- Regularization (dropout, drop connect, L1, L2)
- Activations (and activation function families)
- Weight initialization strategy
- Loss functions
- Settings for epochs during training (mini-batch size)
- Normalization scheme for input data (vectorization)

Layer Size

- Layer size is defined by the number of neurons in a given layer.
- Input and output layers are relatively easy to figure out.
- Deciding neuron counts for each hidden layer is a challenge.
- Complex problem -Directly correlated to how many neurons are in the hidden layers.
- Neurons come with a cost.
- *Deep network architecture, connection schema between layers can vary.*

- Weights on the connections, are the parameters we must train.
- More parameters in our model, increase the amount of effort needed to train the network.
- Long training times and models struggle to find convergence.

Magnitude

- Magnitude group involve the gradient, step size, and momentum.
- **Momentum** is a method that helps speed up the convergence of gradient descent by incorporating the concept of **inertia** into the weight updates.
- **Momentum** helps to know the direction of the next step with the knowledge of the previous steps.
- It helps to prevent oscillations.
- Speed up training by increasing momentum.
- Typically, the value for momentum between **0.9 and 0.99.**

Learning Rate

- The learning rate defines how quickly a network updates its parameters.
- **Low learning rate** slows down the learning process but converges smoothly.
- **Larger learning rate** speeds up the learning but may not converge.
- AdaGrad (Adaptive Gradient Algorithm) -technique to finding the “right” learning rate.
- **Features :**
- Monotonically decreasing and never increases the learning rate.

- Square root of the sum of squares of the history of gradient computations.
- Speeds our training in the beginning and slows it appropriately toward convergence.
- RMSprop is a very effective, but currently unpublished adaptive learning rate method.
- AdaDelta is a variant of AdaGrad that keeps only the most recent history.
- ADAM (Adaptive Moment Estimation) derives learning rates from estimates of first and second moments of the gradients.

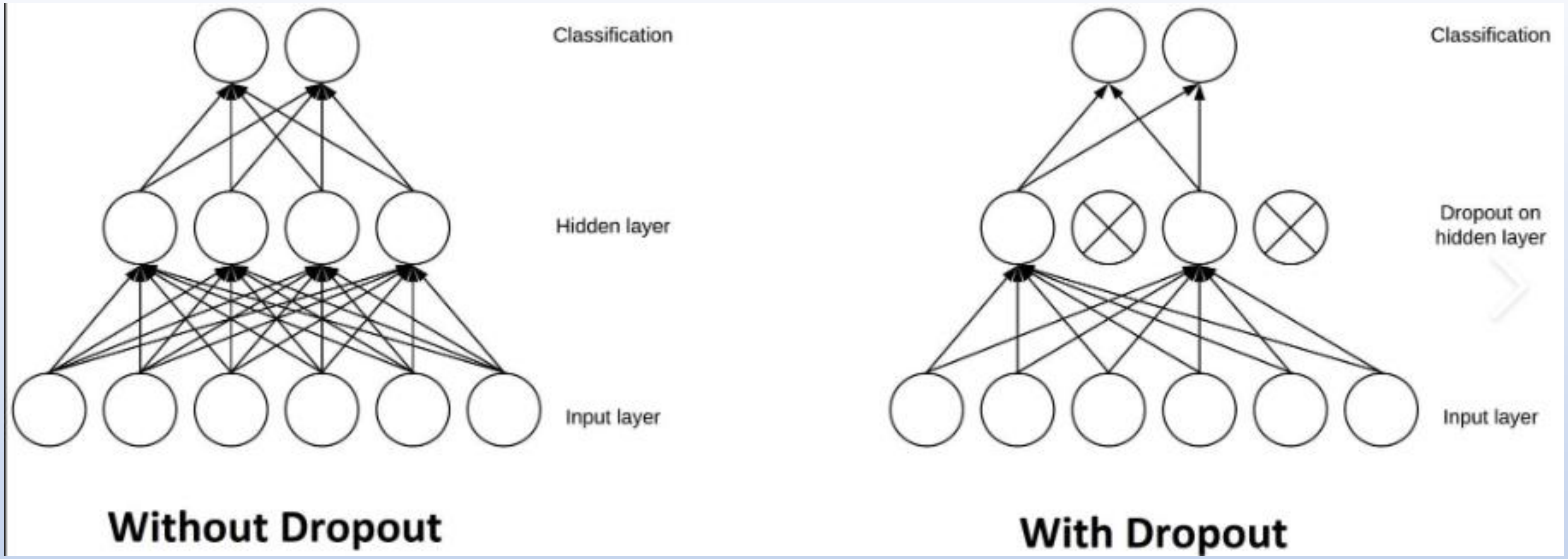
- **First Moment (Mean)**: Adam computes a moving average of the gradients.
- Helps to smooth out the noise and accelerate convergence.
- **Second Moment (Uncentered Variance)**: Adam also computes a moving average of the squared gradients.
- Scales the learning rates according to the variance of the gradients.

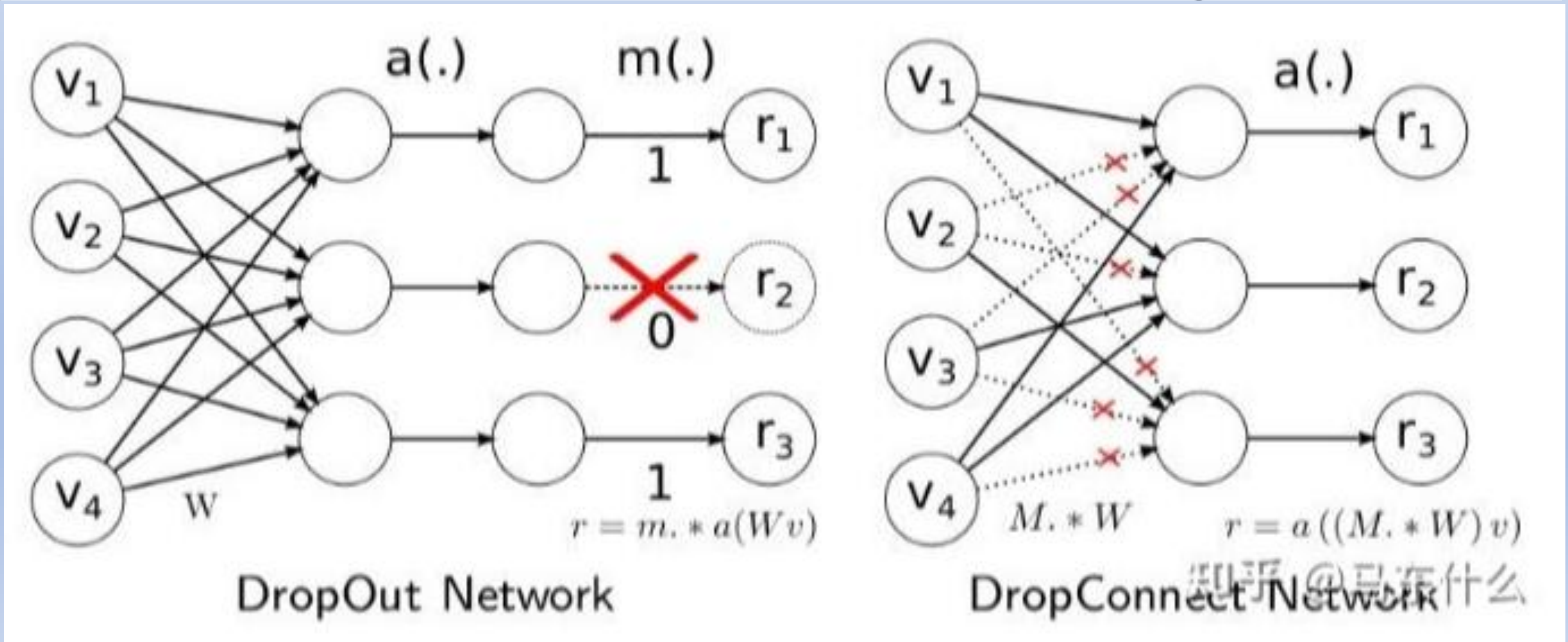
Regularization

- Regularization is a measure taken against overfitting.
- Overfitting : when a model describes the training set but cannot generalize well over new inputs.
- Overfitted models have no predictive capacity for data that they haven’t seen.
- Geoffery Hinton described the best way to build a neural network model:
- Cause it to overfit, and then regularize it to death.

- Regularization, modify the gradient so that it doesn’t step in directions that lead it to overfit.
- <u>Regularization includes :</u>
- Dropout
- Drop Connect
- L1 penalty
- L2 penalty

- Dropout :
- Dropout is a mechanism used to improve the training of neural networks by omitting a hidden unit.
- It also speeds training.
- Dropout is driven by randomly dropping a neuron so that it will not contribute to the forward pass and back propagation.
- DropConnect :
- Mutes the connection between two neurons.


Classification
Hidden layer
Input layer
Without Dropout
Classification
Dropout on hidden layer
Input layer
With Dropout


v_1
v_2
v_3
v_4
a(.)
m(.)
1
0
1
r_1
r_2
r_3
W
$r = m .* a(Wv)$
DropOut Network
v_1
v_2
v_3
v_4
a(.)
r_1
r_2
r_3
$M .* W$
$r = a((M .* W)v)$
DropConnect Network

- **Penalty Methods :**
- L1 and L2 penalty methods are regularization techniques used in machine learning to prevent overfitting by adding a penalty to the loss function.
- **L1 Penalty (Lasso Regularization):**
- Lasso (Least Absolute Shrinkage and Selection Operator) regularization.
- Adds a penalty equal to the absolute value of the magnitude of coefficients.
- **L2 Penalty (Ridge Regularization):**
- Adds penalty equal to the square of the magnitude of coefficients.

L1 Regularization

$$\text{Modified loss function} = \text{Loss function} + \lambda \sum_{i=1}^{n} |W_i|$$

L2 Regularization

$$\text{Modified loss function} = \text{Loss function} + \lambda \sum_{i=1}^{n} {W_i}^2$$

- Loss is the original loss function (e.g., mean squared error for regression).
- λ\lambda is the regularization parameter (controls the strength of the penalty).
- Wi are the model parameters.

Mini-batching

- Batch size always seems to affect training.
- Using a very small batch size can lead to slower convergence of the model.
- Too small or a too large batch size can both affect training badly.
- A batch size of 32 or 64 almost always seems like a good option.