Deep Learning

Lesson 4—Multilayer ANN

Learning Objectives

- ✓ Explore multilayer ANN
- ✓ Implement forward propagation in multilayer perceptron (MLP)
- ✓ Analyze how to regularize and minimize the cost function in a neural network
- ✓ Carry out backpropagation to adjust weights in a neural network
- ✓ Inspect convergence in a multilayer ANN
- ✓ Understand how the capacity of a model is affected by underfitting and overfitting

Multilayer ANN

Topic 1—Introduction

Single-layer ANN

A RECAP

- Perceptron rule and Adaline rule were used to train a single-layer neural network.
- Weights are updated based on a unit function in perceptron rule or on a linear function in Adaline rule.

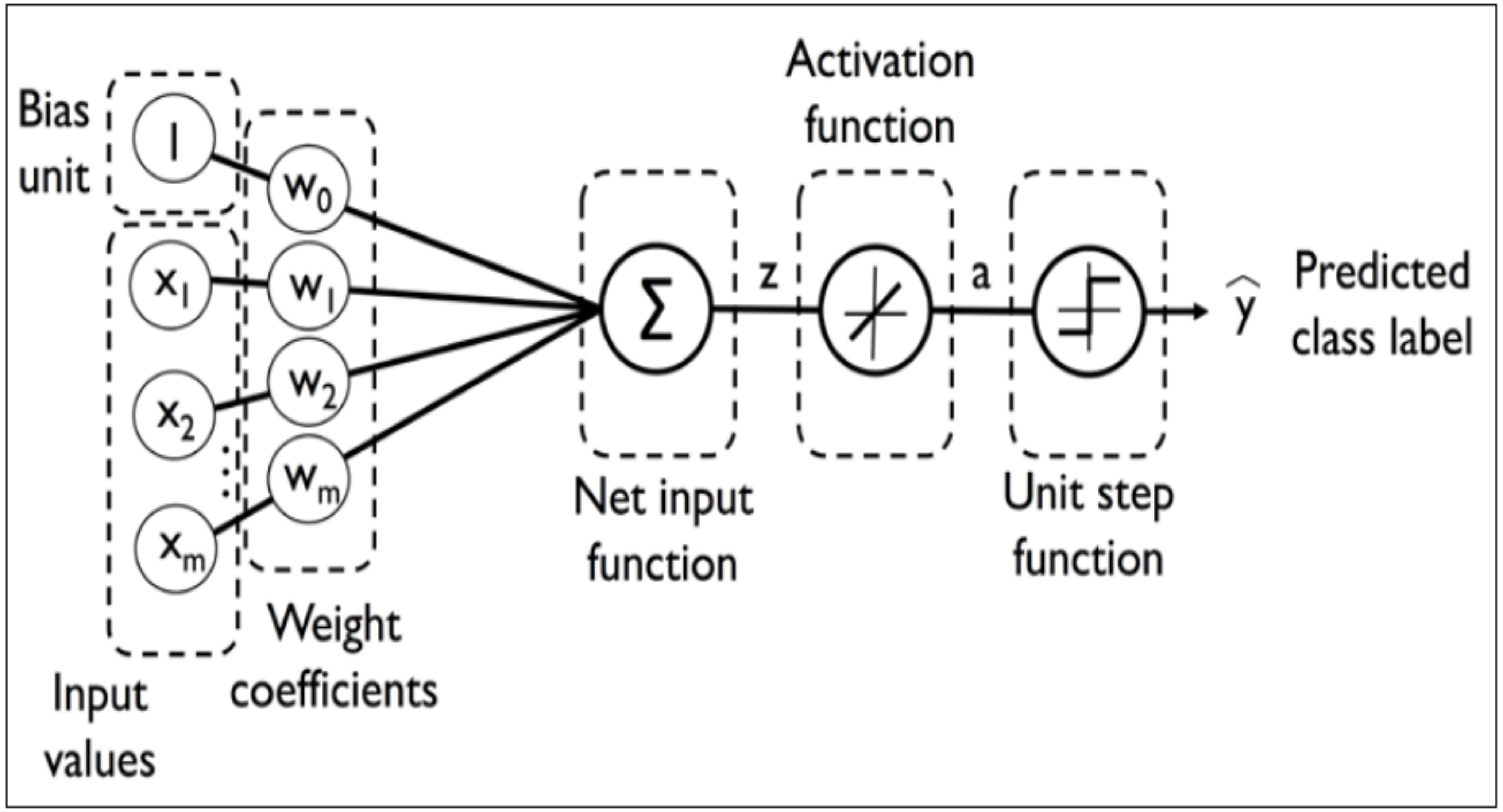


Image Source : "Python Machine Learning" by Sebastian Raschka

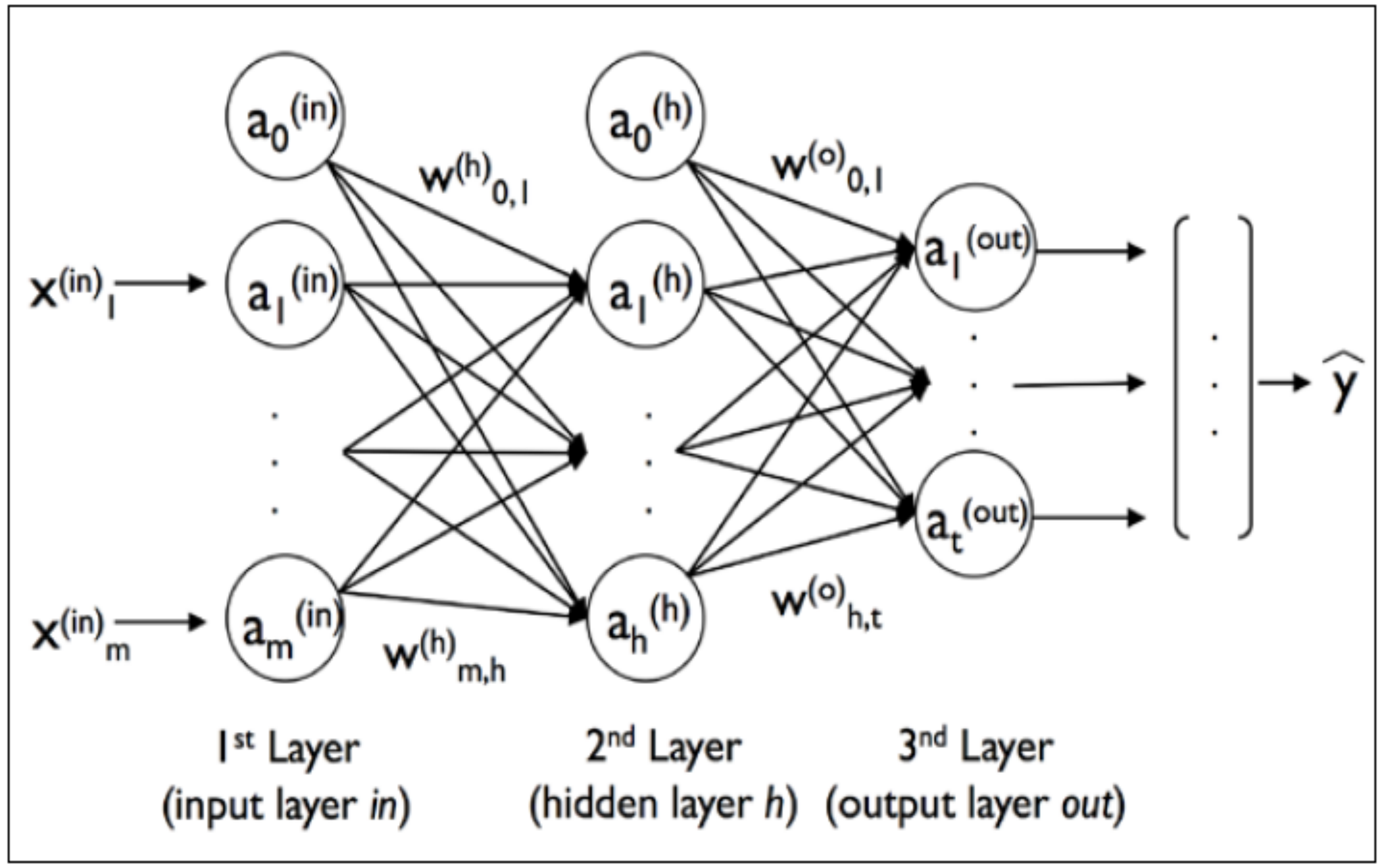
History of Multilayer ANN

Deep Learning deals with training multi-layer artificial neural networks, also called Deep Neural Networks.

- After Rosenblatt perceptron was developed in 1950s, there was lack of interest in neural networks until 1986, when Dr.Hinton and his colleagues developed the backpropagation algorithm to train a multilayer neural network.
- Today it is a hot topic with many leading firms like Google, Facebook and Microsoft which invest heavily in applications using deep neural networks.

Multi-layer ANN

- A fully connected multi-layer neural network is called a **Multilayer Perceptron (MLP)**.
- It has 3 layers including one hidden layer.
- If it has more than 1 hidden layer, it is called a deep ANN.
- An MLP is a typical example of a feedforward artificial neural network.



In this figure, the ith activation unit in the *l*th layer is denoted as $a_i^{(l)}$.

Image Source : "Python Machine Learning" by Sebastian Raschka

Multilayer ANN (Contd.)

- The number of layers and the number of neurons are referred to as hyper parameters of a neural network, and these need tuning. Cross-validation techniques must be used to find ideal values for these.

- The weight adjustment training is done via backpropagation.

- Deeper neural networks are better at processing data. However, deeper layers can lead to vanishing gradient problem. Special algorithms are required to solve this issue.

Image Credit : "Python Machine Learning" by Sebastian Raschka

Multilayer ANN

NOTATIONS

In the representation below:

$$a^{(in)} = \begin{bmatrix} a_0^{(in)} \\ a_1^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix}$$

- $a_i^{(in)}$ refers to the ith value in the input layer
- $a_i^{(h)}$ refers to the ith unit in the hidden layer
- $a_i^{(out)}$ refers to the ith unit in the output layer
- $a_0^{(in)}$ is simply the bias unit and is equal to 1; it will have the corresponding weight w_0
- The weight coefficient from layer *l* to layer *l*+1 is represented by $w_{k,j}^{(l)}$

Multilayer ANN (Contd.)

NOTATIONS

- A simplified view of the multilayer is presented here.
- This image shows a fully connected three-layer neural network with 3 input neurons and 3 output neurons. A bias term is added to the input vector.

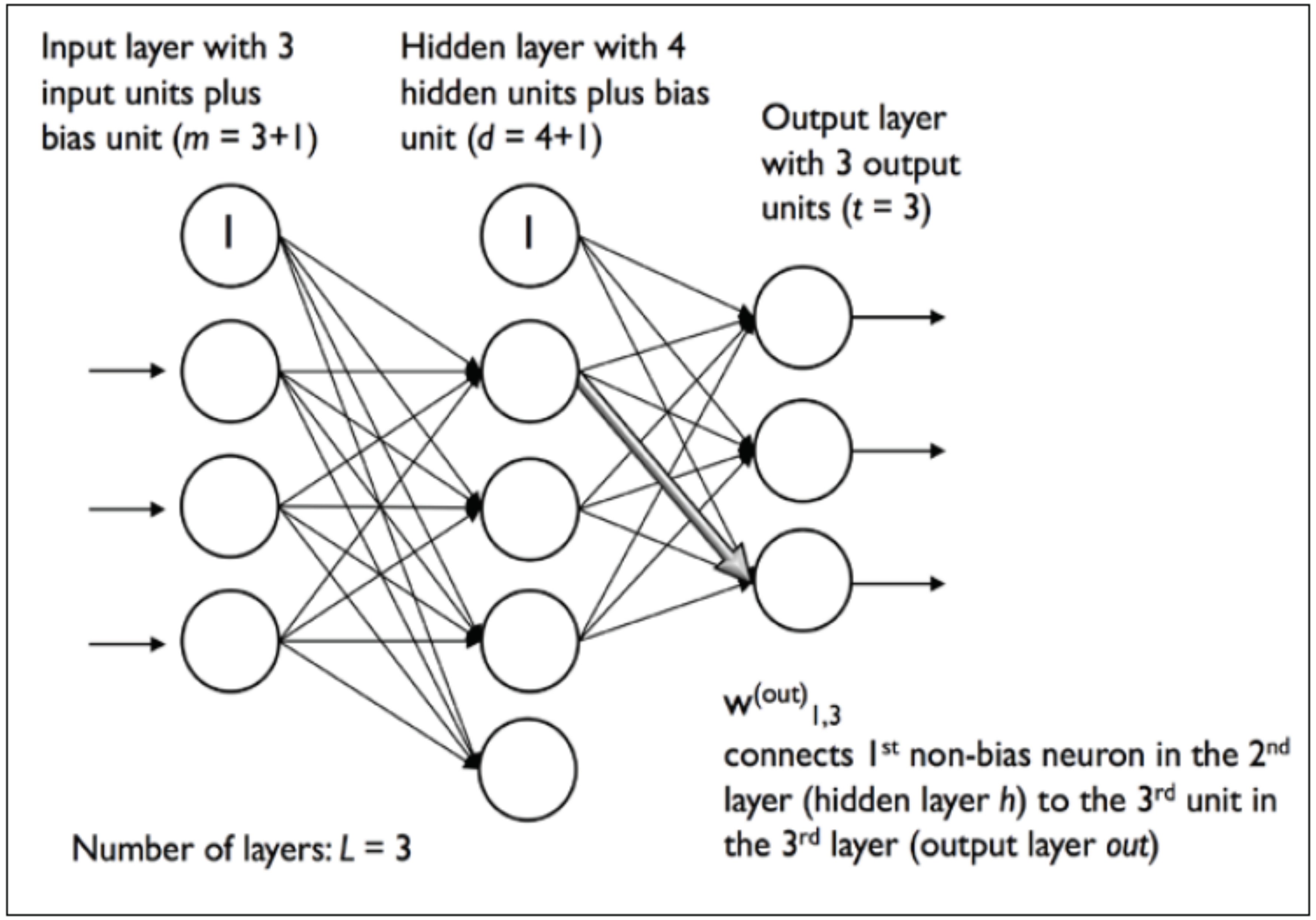

Input layer with 3 input units plus bias unit (m = 3+1)
Hidden layer with 4 hidden units plus bias unit (d = 4+1)
Output layer with 3 output units (t = 3)
1
1
$w^{(out)}_{1,3}$
connects 1st non-bias neuron in the 2nd layer (hidden layer h) to the 3rd unit in the 3rd layer (output layer out)
Number of layers: L = 3

Image Source : "Python Machine Learning" by Sebastian Raschka

Multilayer ANN

Topic 2—Forward Propagation

MLP Learning Procedure

The MLP learning procedure is as follows :

Starting with the input layer, propagate data forward to the output layer. This step is the forward propagation.

Based on the output, calculate the error (difference between predicted and known outcome). The error needs to be minimized.

Backpropagate the error. Find its derivative with respect to each weight in the network, and update the model.

Repeat the three steps given above over multiple epochs to learn ideal weights.

Finally, the output is taken via a threshold function to obtain the predicted class labels.

Forward Propagation in MLP

In the first step, calculate the activation unit $a_l^{(h)}$ of the hidden layer.

$$z_1^{(h)} = a_0^{(in)} w_{0,1}^{(h)} + a_1^{(in)} w_{1.1}^{(h)} + \cdots + a_m^{(in)} w_{m,1}^{(h)}$$

Activation unit is result of applying an activation function φ to the z value. It must be differentiable to be able to learn weights using gradient descent.

$$a_1^{(h)} = \phi\left(z_1^{(h)}\right)$$

The activation function φ is often the sigmoid (logistic) function. It allows nonlinearity needed to solve complex problems like image processing.

$$\phi(z) = \frac{1}{1+e^{-z}}$$

Forward Propagation in MLP

SIGMOID CURVE

The sigmoid curve is an S-shaped curve.

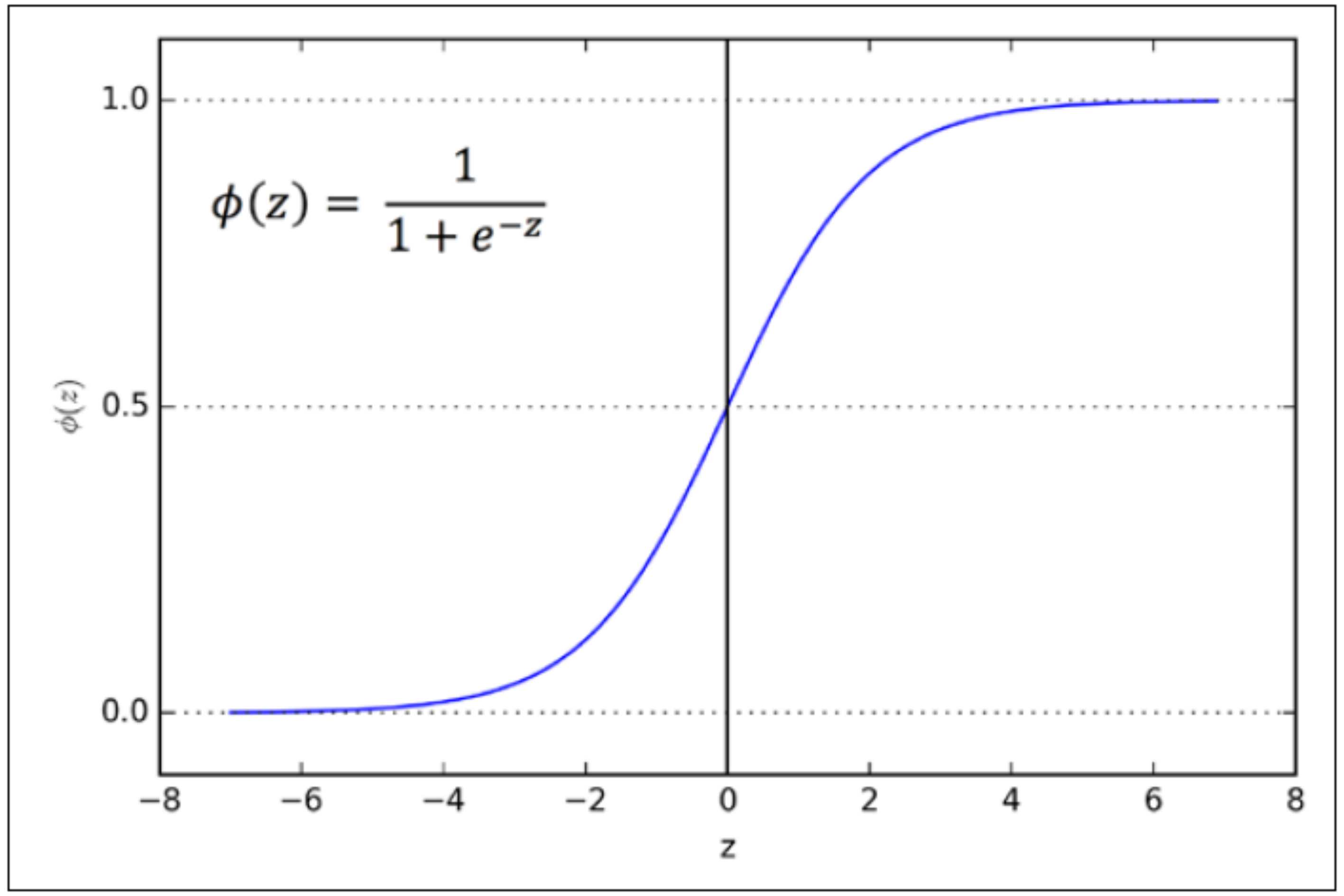


Image Source : "Python Machine Learning" by Sebastian Raschka

Forward Propagation in MLP

RECAP OF NOTATIONS

- The activation of the hidden layer is represented as:

$$\boldsymbol{z}^{(h)} = \boldsymbol{a}^{(in)}\boldsymbol{W}^{(h)}$$

$$\boldsymbol{a}^{(h)} = \phi\left(\boldsymbol{z}^{(h)}\right)$$

- For all n samples in the training set:

$$\boldsymbol{Z}^{(h)} = \boldsymbol{A}^{(in)}\boldsymbol{W}^{(h)} \qquad A^{(h)} = \phi\left(Z^{(h)}\right)$$

- For the output layer:

$$\boldsymbol{Z}^{(out)} = \boldsymbol{A}^{(h)}\boldsymbol{W}^{(out)} \qquad \boldsymbol{A}^{(out)} = \phi\left(\boldsymbol{Z}^{(out)}\right), \qquad \boldsymbol{A}^{(out)} \in \mathbb{R}^{n\times t}$$

Multilayer ANN

Topic 3– Cost Function in a Neural Network

Cost Function in a Neural Network

- To develop a multi-layer ANN, a cost function is chosen. This is minimized to arrive at the final network.
- This cost function is called the **Cross-entropy** cost function.

The logistic cost function used in a neural network is:

$$J(\boldsymbol{w}) = -\sum_{i=1}^{n} y^{[i]} log\left(a^{[i]}\right) + \left(1 - y^{[i]}\right) log\left(1 - a^{[i]}\right)$$

Here, "$a^{[i]}$" is the sigmoid activation of ith sample in the dataset:

$$a^{[i]} = \phi\left(z^{[i]}\right)$$

Cost Function in a Neural Network

REGULARIZATION PROCESS

Add an “L2” regularization term to prevent overfitting.

$$L2 = \lambda \|\boldsymbol{w}\|_2^2 = \lambda \sum_{j=1}^{m} w_j^2$$

Quick Recap: Regularization prevents overfitting by restricting the degrees of freedom of an algorithm. This is achieved by introducing a small amount of error in the model training process.

Cost Function in a Neural Network (Contd.)

REGULARIZATION PROCESS

The cost function now becomes:

$$J(\boldsymbol{w}) = -\left[\sum_{i=1}^{n} y^{[i]} log\left(a^{[i]}\right) + \left(1 - y^{[i]}\right) log\left(1 - a^{[i]}\right)\right] + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

The logistic cost function must be generalized to all “t” activation units in the network.

The cost function (without the regularization term) becomes:

$$J(\boldsymbol{W}) = -\sum_{i=1}^{n}\sum_{j=1}^{t} y_j^{[i]} log\left(a^{[i]}{}_j\right) + \left(1 - y_j^{[i]}\right) log\left(1 - a^{[i]}{}_j\right)$$

Cost Function in a Neural Network (Contd.)

REGULARIZATION PROCESS

Add regularizer as before:

$$J(\boldsymbol{W}) = -\left[\sum_{i=1}^{n}\sum_{j=1}^{t} y_j^{[i]} log\left(a^{[i]}{}_j\right) + \left(1 - y_j^{[i]}\right) log\left(1 - a^{[i]}{}_j\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{u_l}\sum_{j=1}^{u_{l+1}}\left(w_{j,i}^{(l)}\right)^2$$

Here, "u_l" represents the number of units in given layer "l."

Cost Function in a Neural Network

COST MINIMIZATION PROCESS

To minimize the cost function, calculate the partial derivative of the parameters “W” with respect to each weight for every layer in the network:

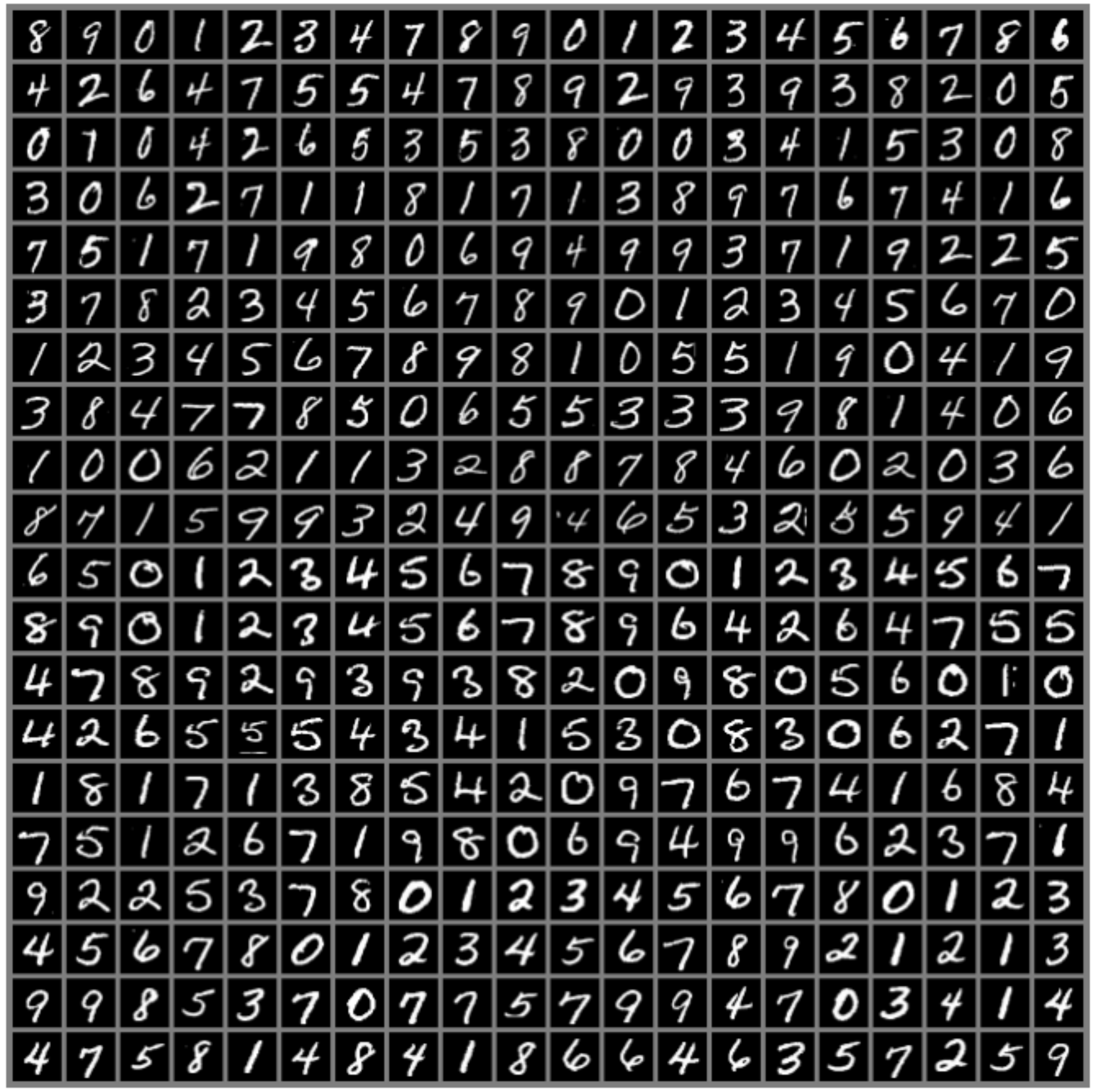
$$\frac{\partial}{\partial w_{j,i}^{(l)}} J(\boldsymbol{W})$$

Demo

MNIST Tutorial

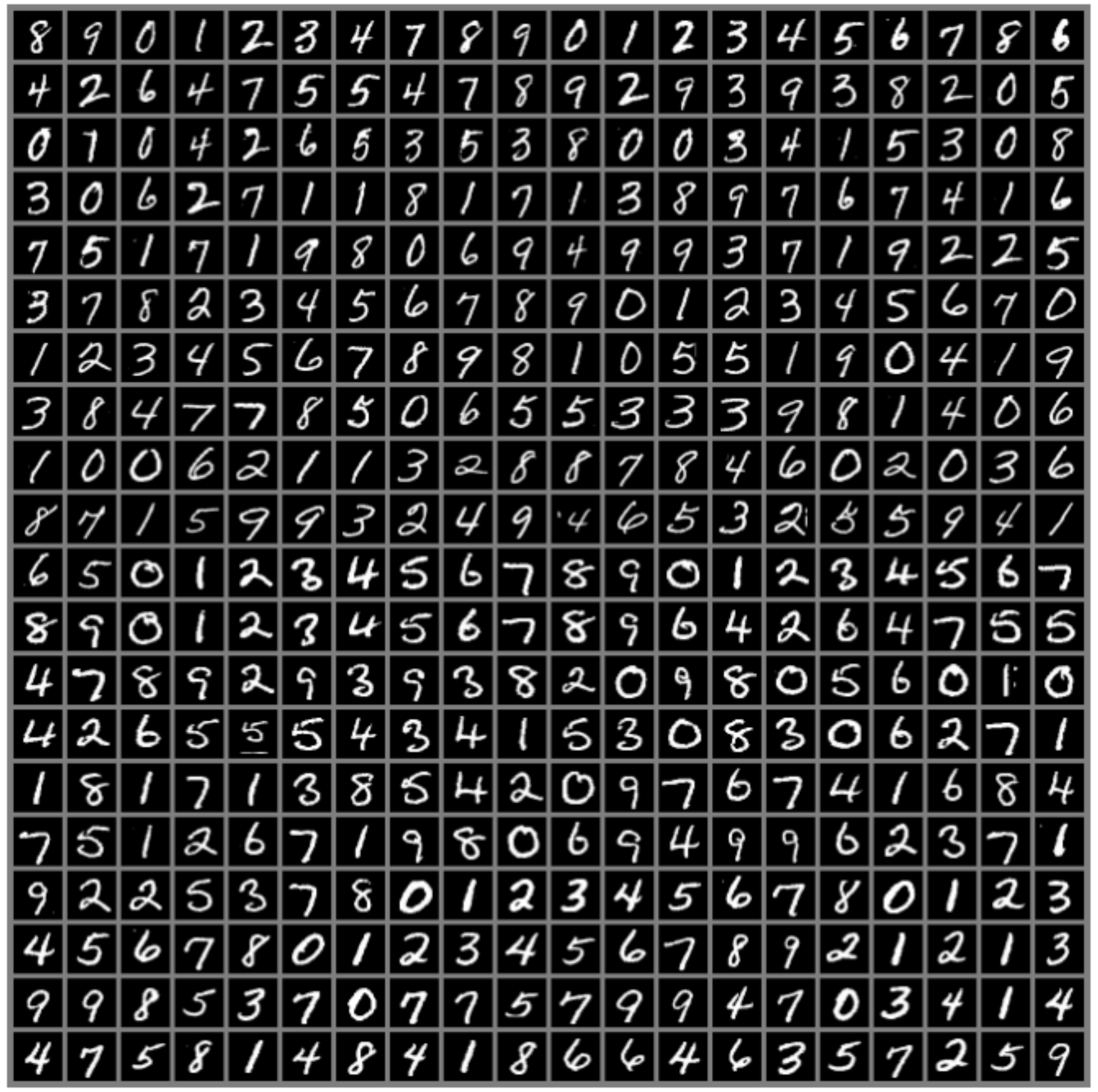
MNIST Dataset

Modified National Institute of Standards and Technology (MNIST) dataset is another popular dataset used in ML algorithms.



Source: "Deep Learning" book by Ian Goodfellow

MNIST Dataset



- National Institute of Standards and Technology (NIST) is a measurement standards laboratory and a nonregulatory agency of US Department of Commerce.
- Modified NIST (MNIST) database is a collection of 70,000 handwritten digits and corresponding digital labels.
- The digital labels identify each of the digits from 0 to 9.
- It is one of the most common datasets used by ML researchers to test their algorithms.

Source: "Deep Learning" book by Ian Goodfellow

Classifying Handwritten Digits

- The attached tutorial uses neural networks to learn how to classify the MNIST dataset.
- Each image 28x28 in size is reshaped to 784 pixel vectors.
- The dataset is split into 60000 training set and 10000 test set.
- Regularization as well as early stopping (of learning) may be deployed to prevent overfitting.

Classifying Handwritten Digits

EPOCHS Vs COST

- The figure shows how the cost goes down as training epochs increase.

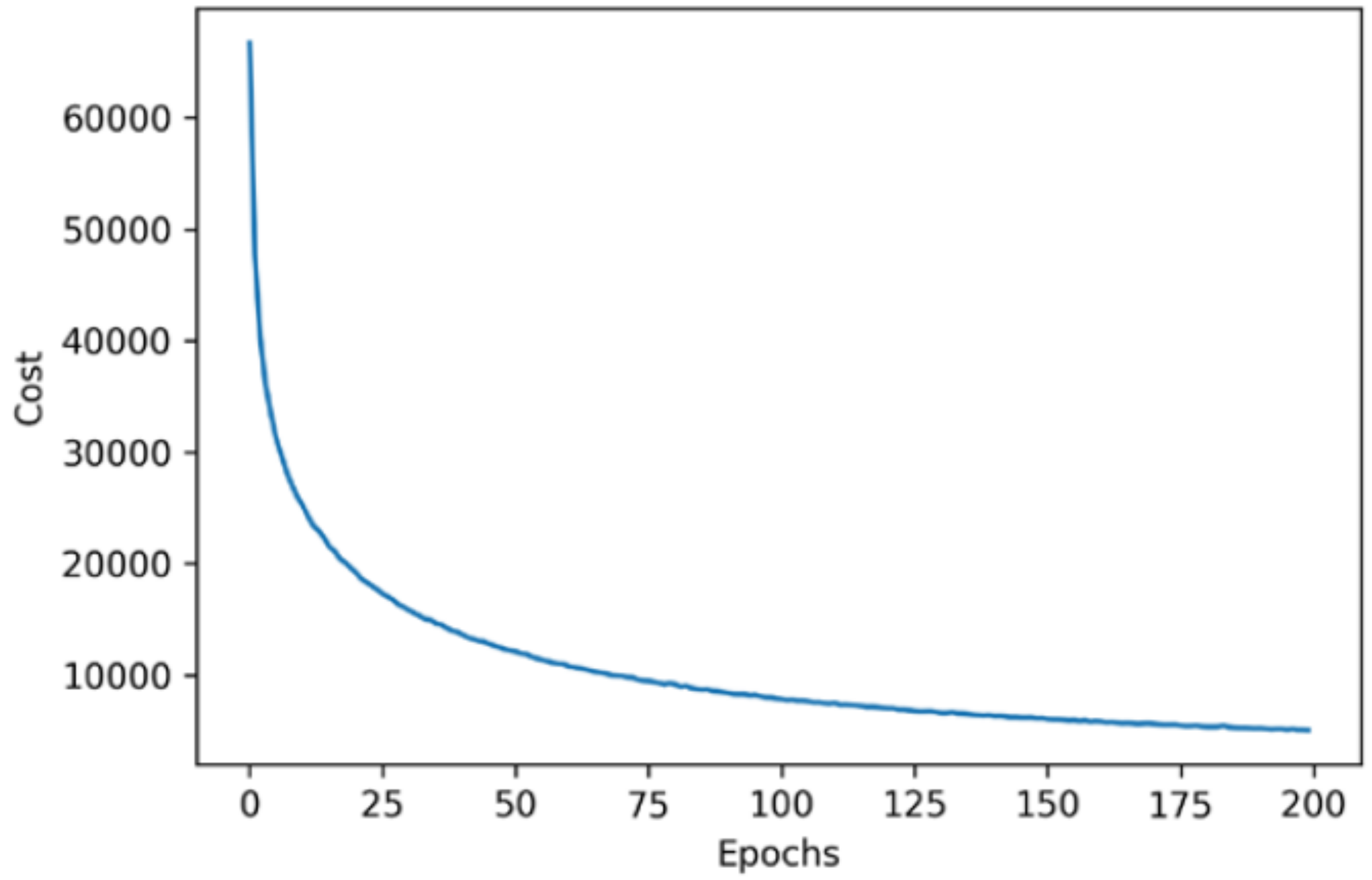


Image Source : "Python Machine Learning" by Sebastian Raschka

Classifying Handwritten Digits

CHECKING TRAINING AND VALIDATION ACCURACY

- At about 50 epochs, training and validation accuracy is equal. After that, the training accuracy starts getting higher, indicating overfitting.
- Overfitting can be decreased by increasing the regularization strength.
- To further fine-tune the model, changes can be made in the number of hidden units , values of the regularization parameters, and the learning rate.

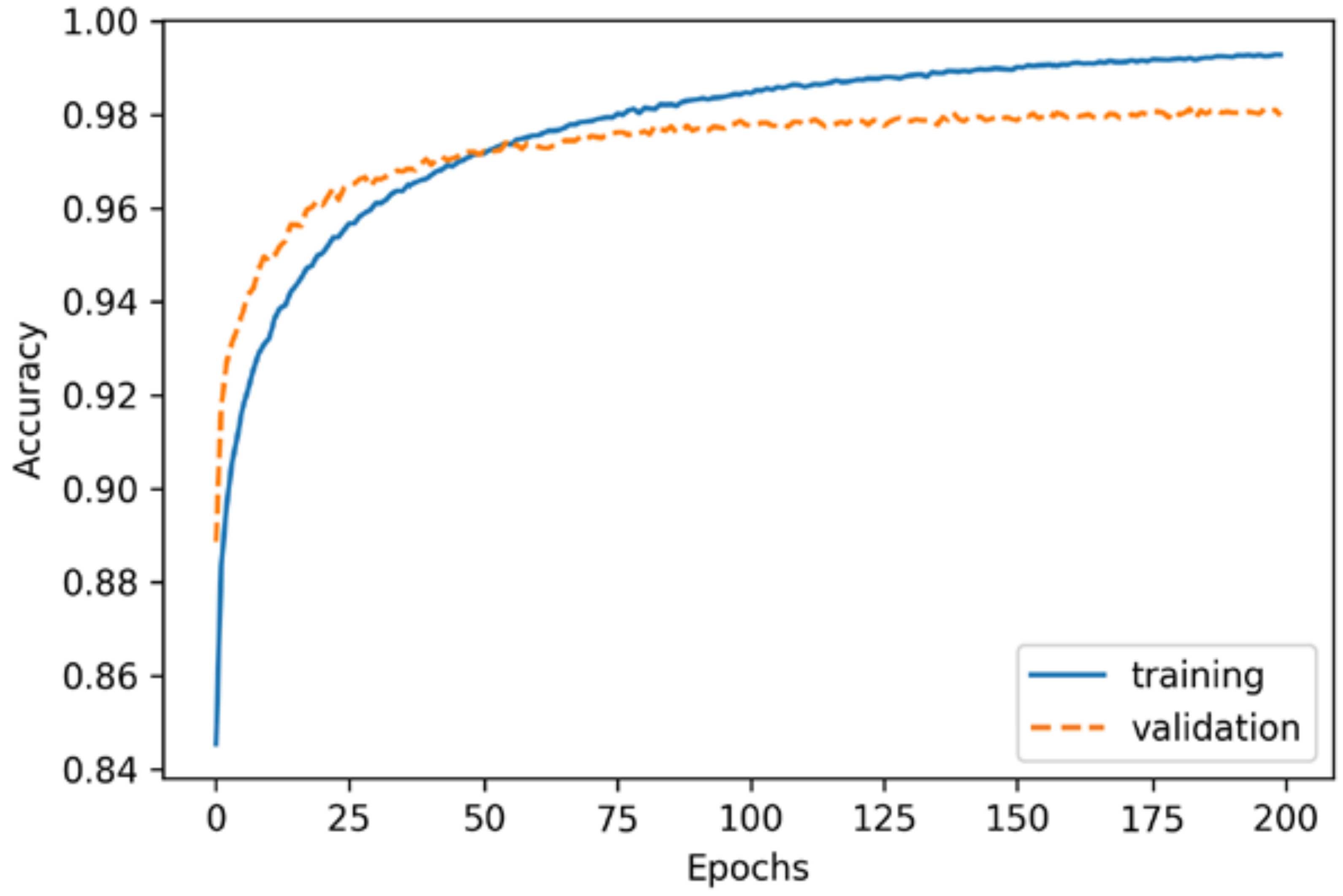


Image Source : "Python Machine Learning" by Sebastian Raschka

Demo 1

Perceptrons and Multilayer ANNs

- **Objective**: Demonstrate Perceptrons and Multilayer ANNs

- **Steps:**
  1. Implement multi-layer ANN (MLP) from scratch. Use low-level Python code for this instead of TensorFlow.
  2. Demonstrate building multi-layer ANN to classify MNIST digits.
  3. Plot the cost function with respect to number of Epochs.
  4. Plot the training and validation accuracy with respect to number of Epochs.
  5. Print the accuracy rate on test dataset.
  6. Print 25 images where the model did not predict correctly.

- **Dataset used:** MNIST dataset
- **Skills required:** Knowledge of multilayer neural network and its characteristics, such as cost function, learning rate, number of layers, accuracy, training vs test dataset, sigmoid activation, etc.

Multi-Layer ANN

Topic 4– Backpropagation in a Neural Network

Backpropagation in a Neural Network

DEFINITION

“Backpropagation is essentially a computationally efficient approach to compute the partial derivatives of a complex cost function in multilayer neural networks.”

Backpropagation in a Neural Network

The goal of the derivatives is to learn the weight coefficients.

- Multilayer perceptron has high-dimensional feature space. Also, the error surface of the cost function is not convex or smooth as in case of Adaline rule.
- There are many local minima to overcome to reach the global minimum.

Backpropagation in a Neural Network

FUNCTION AND DERIVATIVES

The chain rule of calculus is:

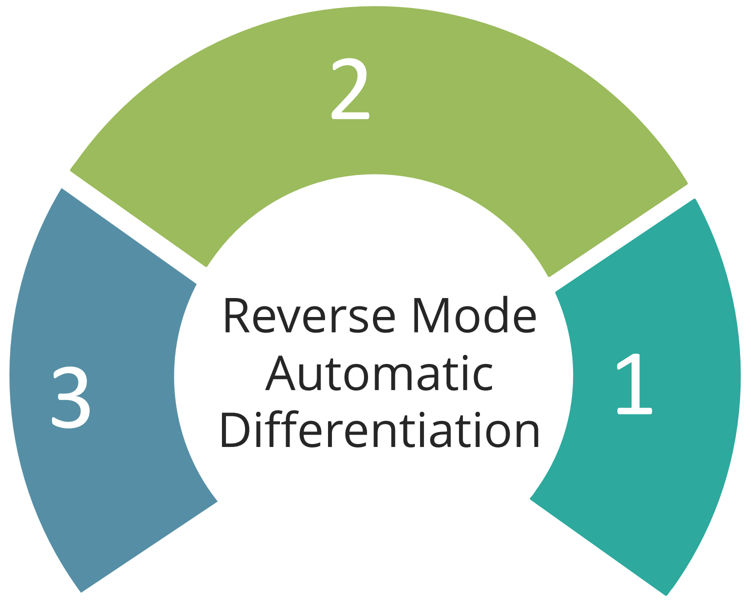
$$\frac{d}{dx}\left[f\left(g\left(x\right)\right)\right]=\frac{df}{dg}\cdot\frac{dg}{dx}$$

Assume there are five different functions and the derivative of their composition is:

$$\frac{dF}{dx}=\frac{d}{dx}F\left(x\right)=\frac{d}{dx}f\left(g\left(h\left(u\left(v\left(x\right)\right)\right)\right)\right)=\frac{df}{dg}\cdot\frac{dg}{dh}\cdot\frac{dh}{du}\cdot\frac{du}{dv}\cdot\frac{dv}{dx}$$

Backpropagation in a Neural Network

REVERSE MODE AUTOMATIC DIFFERENTIATION



Multiply a matrix by a vector

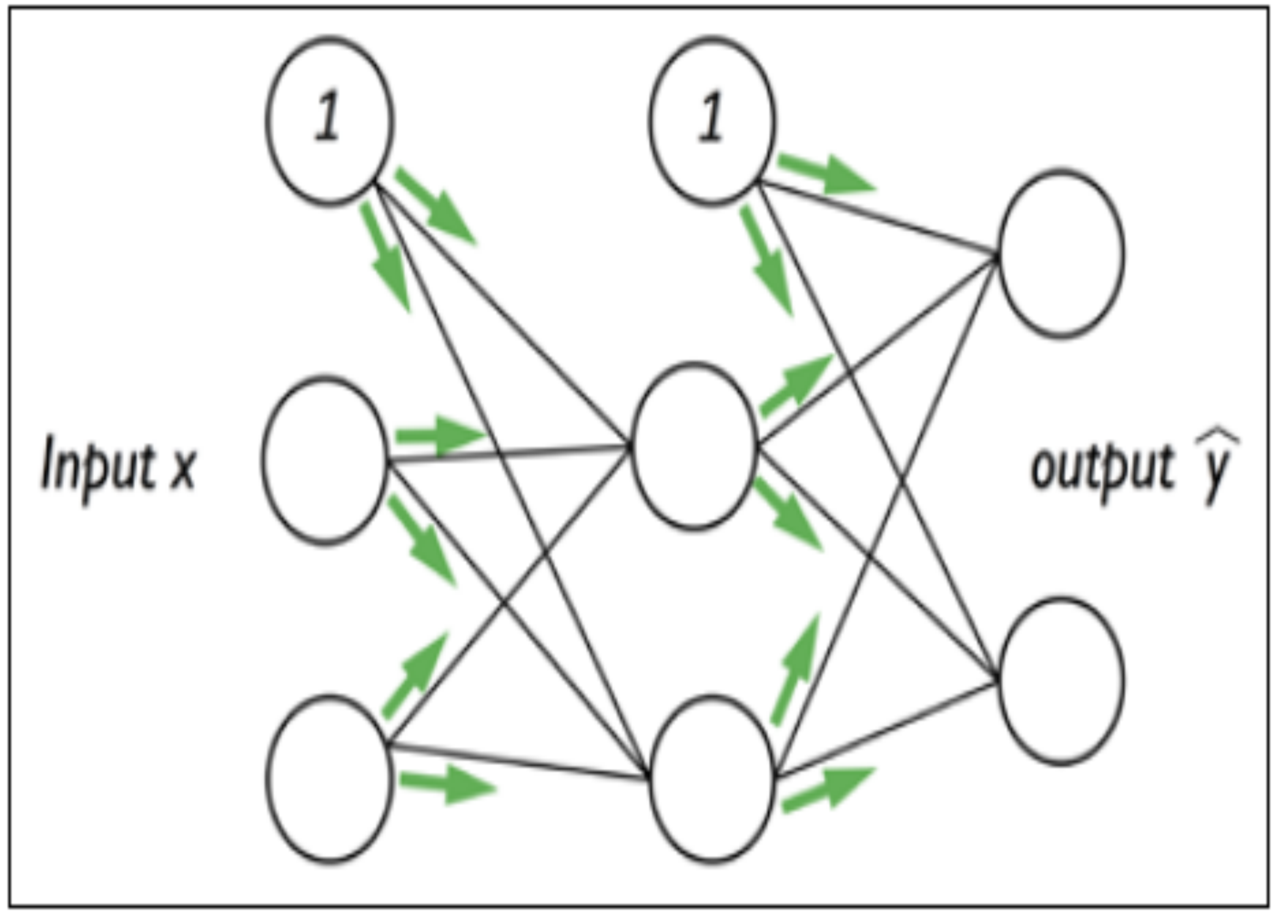
Vector yielded by step 2 is multiplied by next vector and so on

Start from right to left

Matrix-vector multiplication is computationally much cheaper than matrix-matrix multiplication.

Backpropagation in a Neural Network

RECAP OF FORWARD PROPAGATION (WITH ONE HIDDEN LAYER)



$$Z^{(h)} = A^{(in)} W^{(h)} \quad (\textit{net input of the hidden layer})$$

$$A^{(h)} = \phi\left(Z^{(h)}\right) \quad (\textit{activation of the hidden layer})$$

$$Z^{(out)} = A^{(h)} W^{(out)} \quad (\textit{net input of the output layer})$$

$$A^{(out)} = \phi\left(Z^{(out)}\right) \quad (\textit{activation of the output layer})$$

Image Source : "Python Machine Learning" by Sebastian Raschka

Backpropagation in a Neural Network

CALCULATION STEPS

Calculate error vector of output layer.
Here, y is the vector of ground truth labels.

$$\delta^{(out)} = a^{(out)} - y$$

Calculate error term of the hidden layer. Last term is the derivative of the sigmoid activation function.

$$\delta^{(h)} = \delta^{(out)} \left(W^{(out)}\right)^T \odot \frac{\partial \phi\left(z^{(h)}\right)}{\partial z^{(h)}}$$

The derivative of the sigmoid activation can be shown to be equal to:

$$\frac{\partial \phi(z)}{\partial z} = \left(a^{(h)} \odot \left(1 - a^{(h)}\right)\right)$$

This can also be written as given.

$$\delta^{(h)} = \delta^{(out)} \left(W^{(out)}\right)^T \odot \left(a^{(h)} \odot \left(1 - a^{(h)}\right)\right)$$

Note: $\odot$ symbol means element-wise multiplication in this context.

Backpropagation in a Neural Network

CALCULATION STEPS CONTINUED

The derivative of the sigmoid activation can be shown to be equal to:

$$\phi'(z) = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right)$$

$$= \frac{e^{-z}}{\left(1+e^{-z}\right)^2}$$

$$= \frac{1+e^{-z}}{\left(1+e^{-z}\right)^2} - \left(\frac{1}{1+e^{-z}}\right)^2$$

$$= \frac{1}{\left(1+e^{-z}\right)} - \left(\frac{1}{1+e^{-z}}\right)^2$$

$$= \phi(z) - \left(\phi(z)\right)^2$$

$$= \phi(z)\left(1-\phi(z)\right)$$

$$= a(1-a)$$

Backpropagation in a Neural Network

CALCULATION STEPS CONTINUED

After getting the error terms, the derivation of the cost function can be written as follows:

$$\frac{\partial}{\partial w_{i,j}^{(out)}} J(\boldsymbol{W}) = a_j^{(h)} \delta_i^{(out)}$$

$$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\boldsymbol{W}) = a_j^{(in)} \delta_i^{(h)}$$

$$\Delta^{(h)} = \Delta^{(h)} + \left(\boldsymbol{A}^{(in)}\right)^T \delta^{(h)}$$

$$\Delta^{(out)} = \Delta^{(out)} + \left(\boldsymbol{A}^{(h)}\right)^T \delta^{(out)}$$

Next, accumulate the partial derivative of every node in each layer and the error of the node in the next layer.

Add regularizer.

$$\Delta^{(l)} := \Delta^{(l)} + \lambda^{(l)}$$

$$(except\ for\ the\ bias\ term)$$

$$\boldsymbol{W}^{(l)} := \boldsymbol{W}^{(l)} - \eta\Delta^{(l)}$$

After the gradients are computed, the weights can be updated by taking an opposite step toward the gradient for each layer *l*.

Backpropagation in a Neural Network

WEIGHT ADJUSTMENT

The weight adjustment is implemented as follows:

Weight adjustment of hidden layer: $w\_hidden = w\_hidden - \eta * \Delta w\_hidden$

Bias weight adjustment of hidden layer: $b\_hidden = b\_hidden - \eta * \Delta b\_hidden$

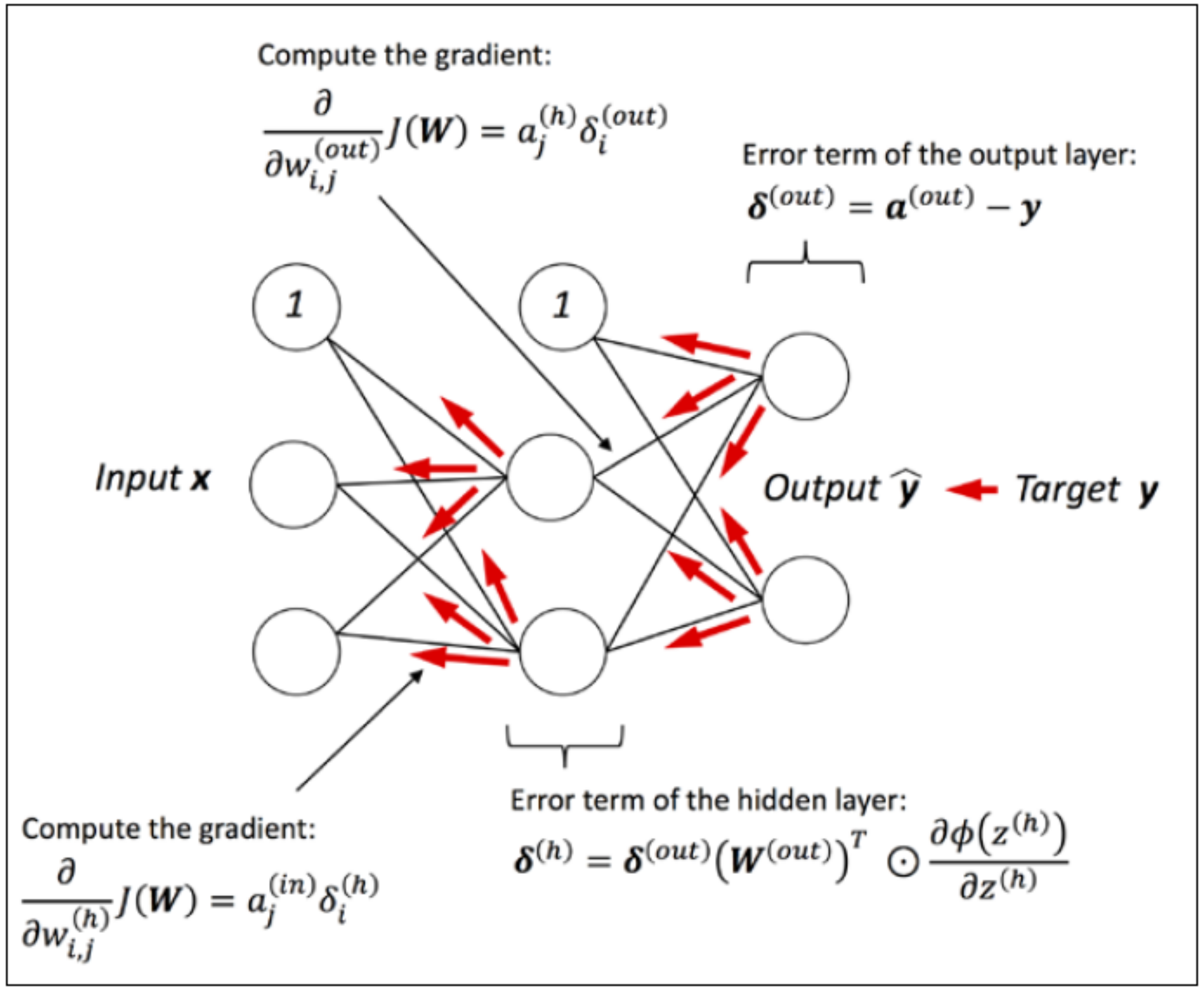
Weight adjustment of output layer: $w\_output = w\_output - \eta * \Delta w\_output$

Bias weight adjustment of output layer: $b\_output = b\_output - \eta * \Delta b\_output$

Backpropagation in a Neural Network

SUMMARY

Summarizing the backpropagation process:



Compute the gradient:
$\frac{\partial}{\partial w_{i,j}^{(out)}} J(\boldsymbol{W}) = a_j^{(h)} \delta_i^{(out)}$
Error term of the output layer:
$\boldsymbol{\delta}^{(out)} = \boldsymbol{a}^{(out)} - \boldsymbol{y}$
1
1
Input **x**
Output $\hat{\boldsymbol{y}}$
Target **y**
Compute the gradient:
$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\boldsymbol{W}) = a_j^{(in)} \delta_i^{(h)}$
Error term of the hidden layer:
$\boldsymbol{\delta}^{(h)} = \boldsymbol{\delta}^{(out)} \left(\boldsymbol{W}^{(out)}\right)^T \odot \frac{\partial \phi\left(z^{(h)}\right)}{\partial z^{(h)}}$

Image Source : "Python Machine Learning" by Sebastian Raschka

Multilayer ANN

Topic 5– Convergence

Convergence in a Multilayer ANN

- Mini-batch learning is generally used for convergence. It is a special form of stochastic gradient descent.
- The gradient is computed based on a subset k of the n training samples with $1 < k < n$.
- Mini-batch leads to faster convergence than batch gradient descent.

Convergence in a Multi Layer ANN (Contd.)

- Multi-layer ANNs can involve thousands, millions, or even billions of weights that we need to optimize. They are harder to train than Adaline, logistic regression, or SVMs.
- The output function in an ANN has a rough surface, and it may get stuck in the local minima as shown in the figure.
- One may try to increase learning rate to escape local minima and still settle at the global minima.
- In spite of this issue, Backpropagation works well in real-world applications.

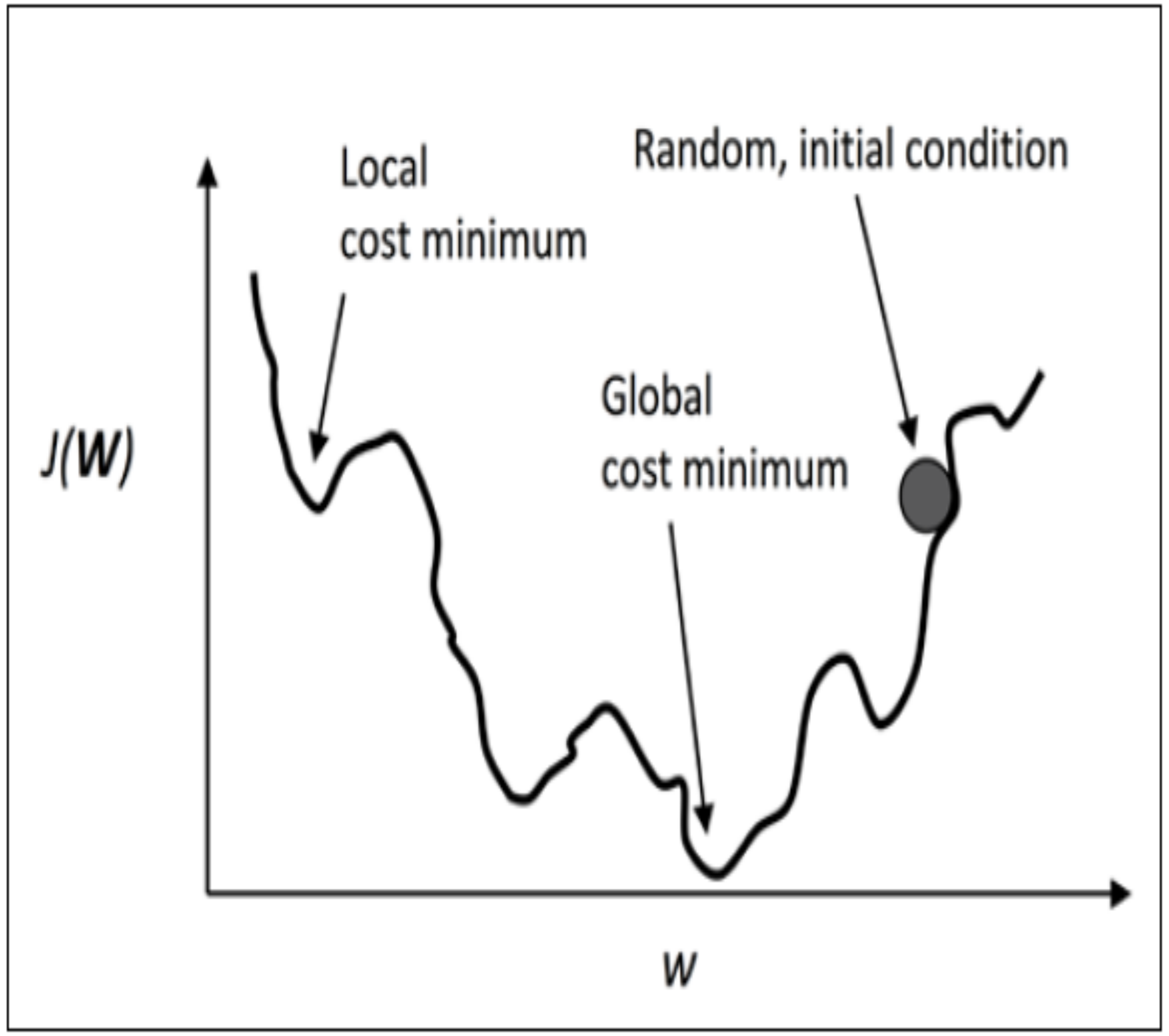


Image Source : "Python Machine Learning" by Sebastian Raschka

Multilayer ANN

Topic 6– Overfitting and Capacity

Overfitting

GENERALIZATION AND REGULARIZATION

- Overfitting indicates that network is trained at a granular level with training data.
- If the network is over-fitted, then it fails to account for generalized situations with new test data, which it has not seen before.
- The algorithm is said to “generalize” better if it can handle previously unseen test data accurately. This is called “**Generalization.**”
- The solution to achieve generalization is a technique called “**Regularization.**”

Overfitting

GRAPHICAL EXAMPLE

- In the graph shown, the green line indicates an overfitted model.
- Regularized model is represented by the black line.
- The green line follows the training data well but is too dependent on it. Also, there are chances of higher error rate on new unseen data in comparison with the black line.

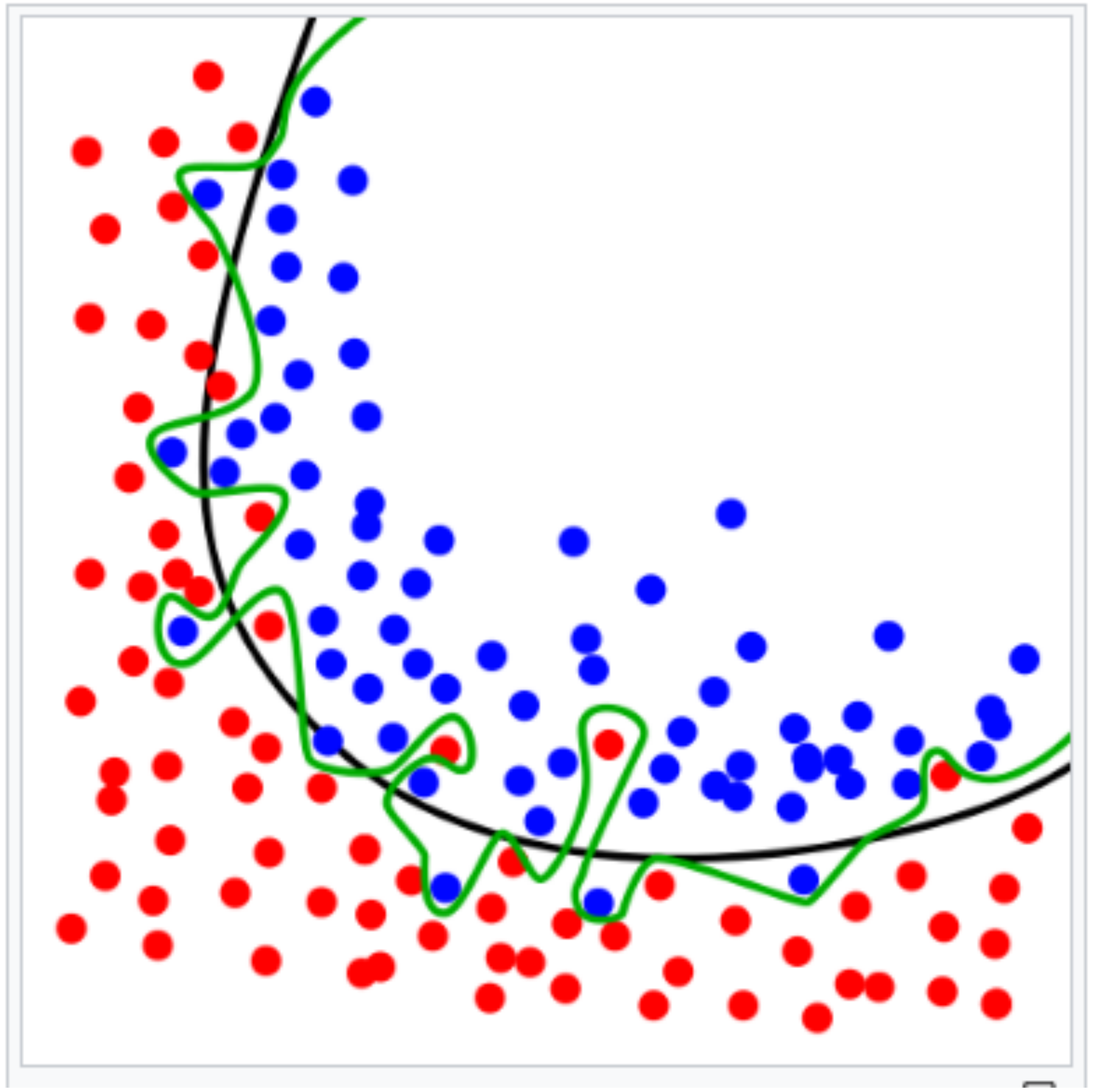


Image Source : https://en.wikipedia.org

Overfitting

DROPOUT

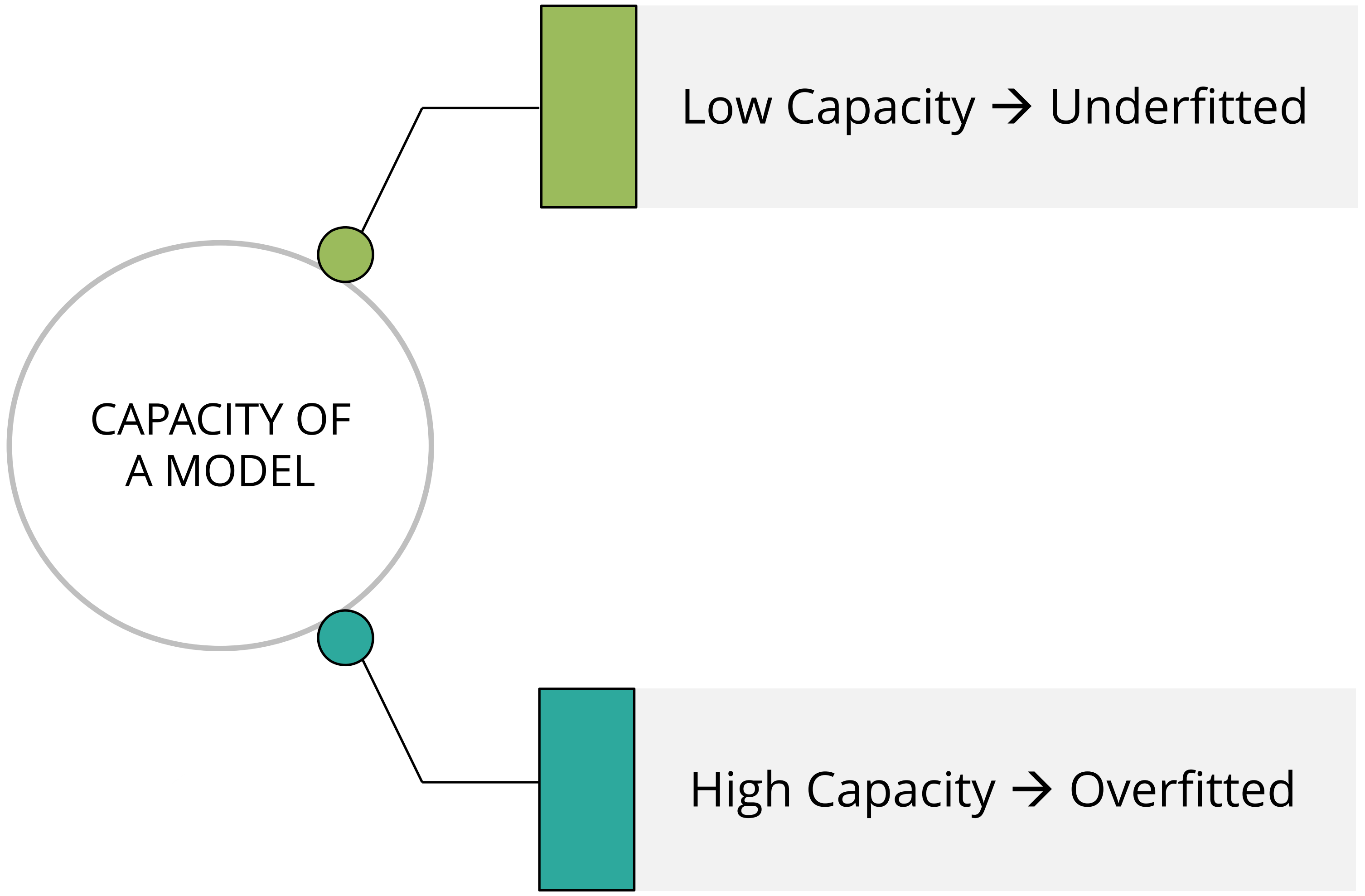
- One way to regularize is to resort to Dropouts.
- This means some neurons are shot in hidden layers in subsequent passes of learning iterations. Do not adjust biases and weights deliberately for some set of neurons in each learning pass.
- For example, 25% of neurons are shot after each iteration of activation functions only for middle layers. This keeps learning less “tightly fitted” to training data, so it can account for previously unseen types of test data with more accuracy.

Dropout is typically done during training to prevent overfitting but not during testing. This is to test the model precisely on the entire test data set.

Overfitting

CAPACITY OF MODEL

- Overfitting or underfitting can be controlled by altering the capacity of a model.



- One has to find the optimum balance.

Overfitting

CAPACITY OF MODEL

- The capacity of a model can be controlled by deciding on the set of functions and corresponding input features that the algorithm can use for its solution. This is called the hypothesis space of the algorithm.

- For example, linear regression uses linear functions as its hypothesis space. However, linear function has linear relationship with the parameters, but it might be a polynomial over the input itself.

Example:

$y = b + wx$ (or)

$y = b + w_1x + w_2x^2$ (or)

$y = b + w_1x + w_2x^2 + ... + w_9x^9$

Overfitting

REPRESENTATIONAL CAPACITY OF MODEL

- Polynomials over input increase the capacity of the model. However, this may cause overfitting beyond a point.

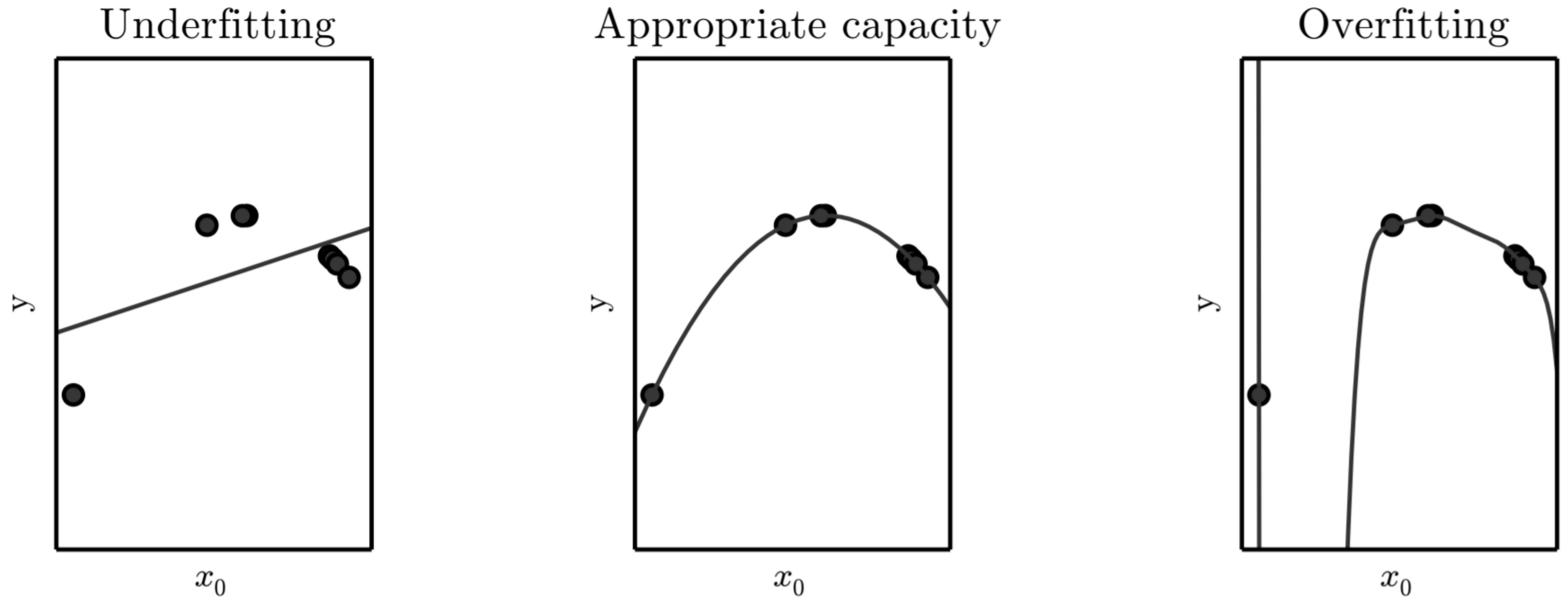
- Varying function choices involves not only using various polynomials but also different ML algorithms such as SVM, neural network, or logistic regression. One has to find optimum models from among all these.

- Hence, the capacity of a model can be changed by varying the function set or the number of input features along with corresponding new parameters. However, there are other means also to vary the model capacity.

- The capacity of the model along with its family of functions as the hypothesis space is called the "**Representational Capacity**" of the model.

Overfitting

CAPACITY OF MODEL



Underfitting happens when the Machine Learning algorithm is poorly trained on sample data.

A well-trained algorithm can generalize well. In other words, it can predict new test data well.

Overfitting means that algorithm is overly trained on test data and cannot analyze new test data well.

Image Source : www.healthiply.in

Overfitting

GENERALIZATION AND CAPACITY

- At lower capacity, the model does not fit well. It is underfitted.
- At higher capacity, it might fit well but might tend to overfit, causing a large generalization error.
- An optimum point for the capacity is found between underfitting and overfitting zones, where generalization error is close to minimum.

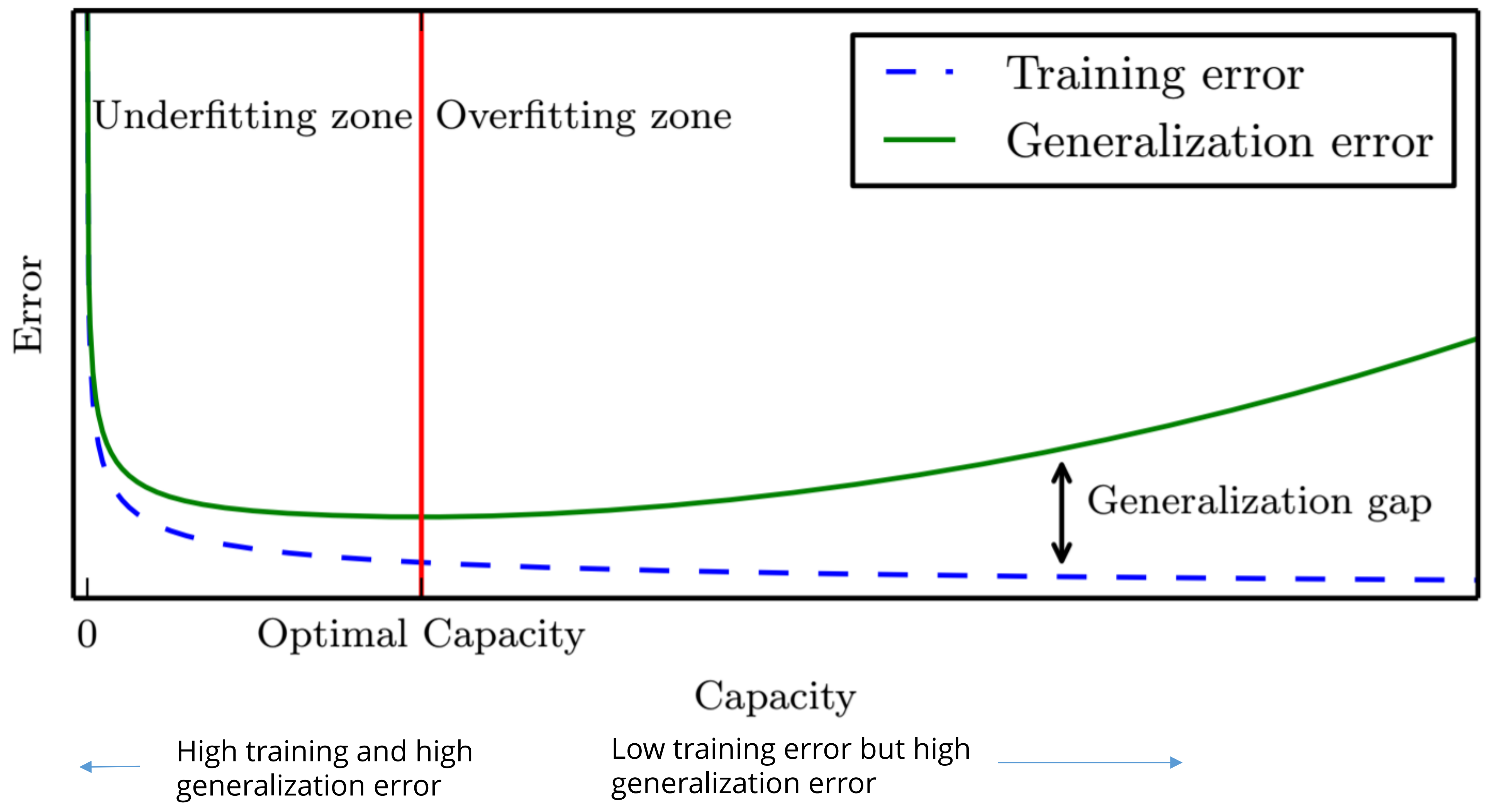


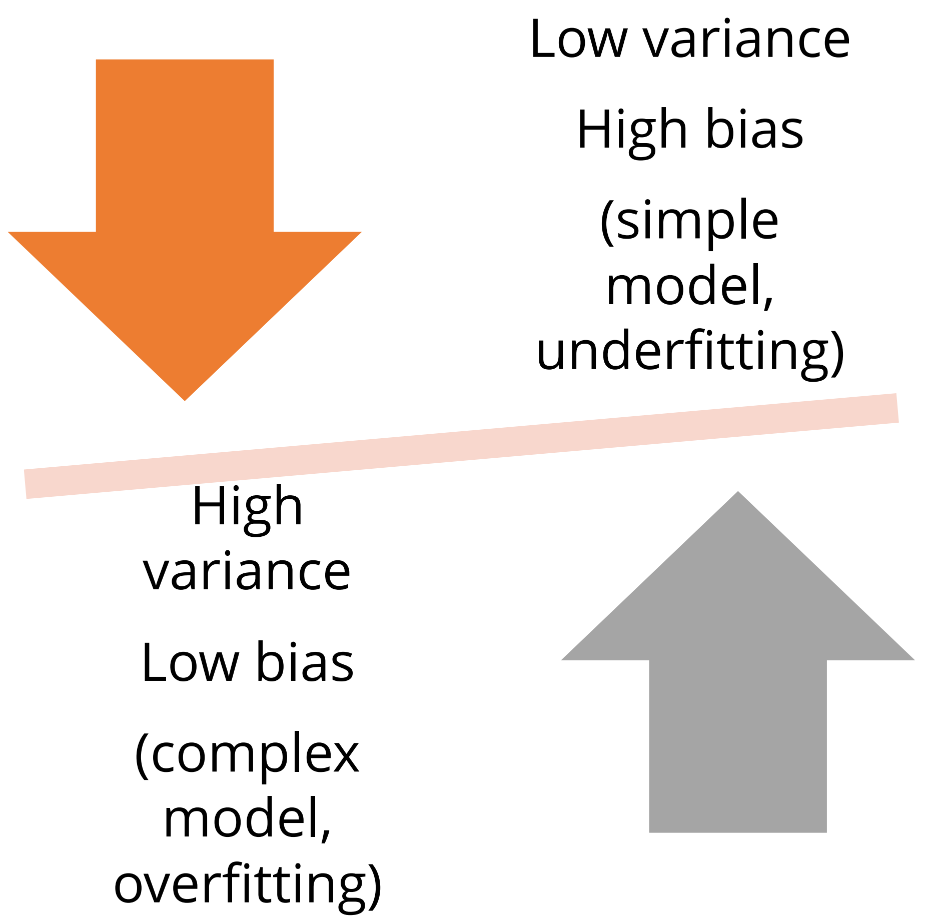
Image Source : https://buzzrobot.com

Bias vs. Variance Tradeoff

Bias

Error in the machine learning model due to wrong assumptions, for example, assuming data is linear when it is not

- A high-bias model will underfit the training data.



Variance

Problems caused by overfitting due to over-sensitivity of the model to small variations in the training data

- A model with many degrees of freedom such as a high-degree polynomial model is likely to have high variance, and thus to overfit the training data.

- Increasing a model's complexity will reduce its bias and increase its variance. Conversely, reducing a model's complexity will increase its bias and reduce its variance. This is called a tradeoff.

How to Train an Artificial Neural Network

SUMMARY

- Decide the input neurons.
- Decide the output neurons.
- Decide on hidden neuron layers.
- Start with arbitrary weights “w” and bias ”b” between each layer.
- Apply Gradient Descent Learning to optimize the weights and bias. This is done by minimizing the loss function using either the mean square error or the cross entropy loss function.

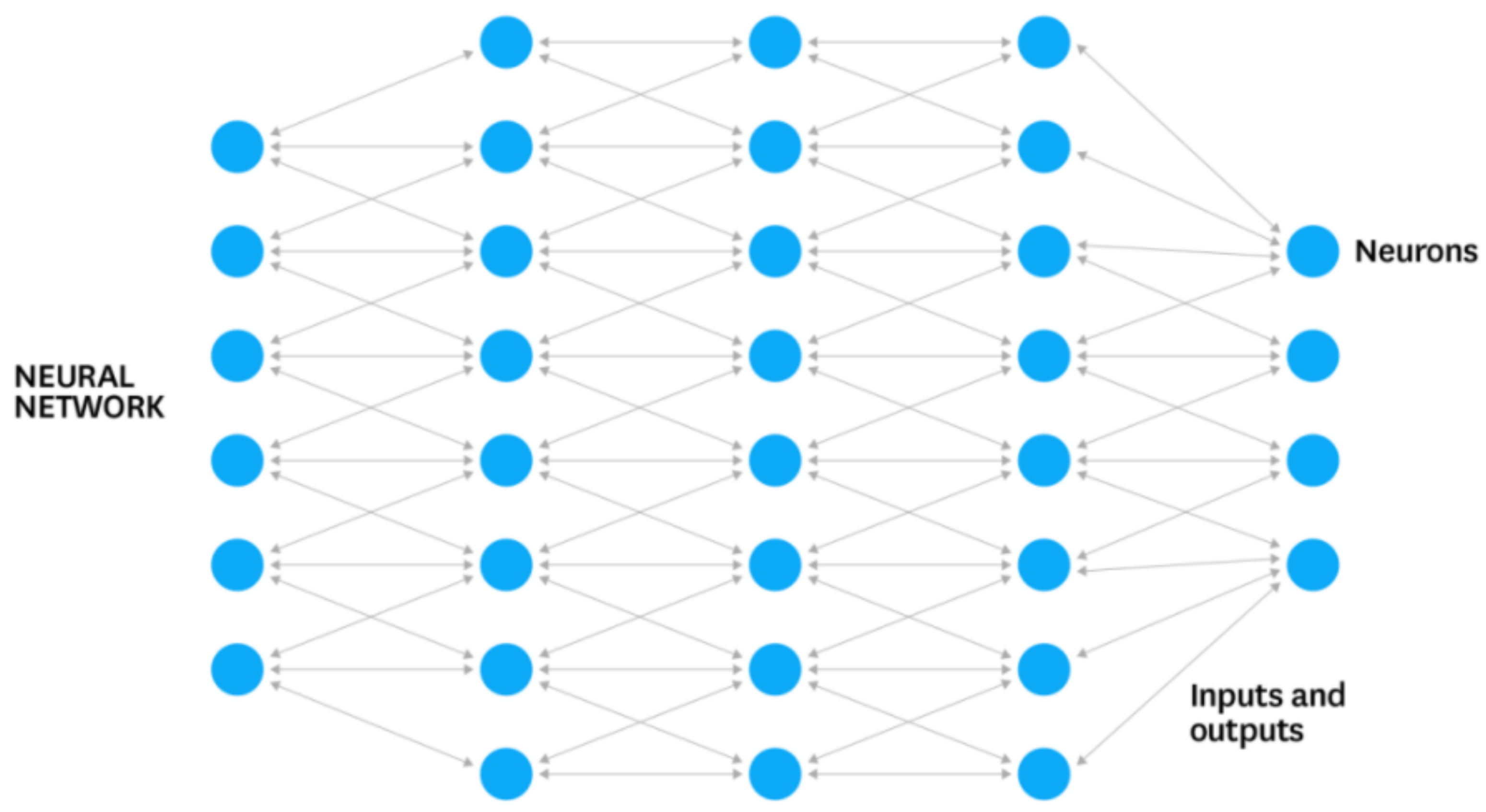
How to Design an Artificial Neural Network

- Once weights and bias are defined, you have a fully trained neural network with algorithms (between any two layers of neurons) such as:-

  y = f(x1*w1, x2*w2, b)

- Platforms like Google TensorFlow are typically used for programming ML. Such platforms offer pre-built APIs for machine learning.

- This is how the driverless cars, filtering people, image recognition, AI in retail, or healthcare diagnosis algorithms are designed broadly.



Key Takeaways

- ✓ A multilayer ANN with more than 1 hidden layer is called a deep ANN. Deeper neural networks are better at processing data.
- ✓ Forward propagation is done by propagating data forward and calculating the activation unit of each unit in each layer.
- ✓ Cross-entropy cost function is a popular cost function used in MLPs.
- ✓ Backpropagation is a mechanism to calculate partial derivatives of cost with respect to weights in each layer, going from right to left.
- ✓ Multilayer ANN can contain thousands of weights that need to be optimized. Finding the global minima of error or point of convergence is influenced by learning rate adjustment.
- ✓ Overfitting or underfitting can be controlled by altering the capacity of the model. Models with low capacity may be underfitted and those with high capacity might be overfitted.

This concludes "Multilayer ANN."

The next lesson is "Introduction to TensorFlow."