Chapter 1

Introduction

Artificial neural networks are mathematical inventions inspired by observations made in the study of biological systems, though loosely based on the actual biology. An artificial neural network can be described as mapping an input space to an output space. This concept is analogous to that of a mathematical function. The purpose of a neural network is to map an input into a desired output. While patterned after the interconnections between neurons found in biological systems, artificial neural networks are no more related to real neurons than feathers are related to modern airplanes. Both biological systems, neurons and feathers, serve a useful purpose, but the implementation of the principles involved has resulted in man-made inventions that bear little resemblance to the biological systems that spawned the creative process.

This text starts with the aim of introducing the reader to many of the most popular artificial neural networks while keeping the mathematical gymnastics to a minimum. Many excellent texts, which have detailed mathematical descriptions for many of the artificial neural networks presented in this book, are cited in the bibliography. Additional mathematical background for the neural-network algorithms is provided in the appendixes as well. Artificial neural networks are modeled after early observations in biological systems: myriads of neurons, all connected in a manner that somehow distributes the necessary signals to various parts of the body to allow the biological system to function and survive. No one knows exactly how the brain works or what is happening in the nervous system all of the time, but scientists and medical doctors have been able to uncover the inner workings of the mind and the nervous system to a degree that allows them to completely describe the operation of the basic computational building block of the nervous system and brain: the neuron.

1.1 The Neuron

The human body is made up of a vast array of living cells. Certain cells are interconnected in a way that allows them to communicate pain, or to actuate fibers or tissues. Some cells control the opening and shutting of minuscule valves in the veins and arteries. Others tell the brain that they are experiencing cold, heat, or any number of sensations. These specialized communication cells are called neurons. Neurons are equipped with long tentacle-like structures that stretch out from the cell body, permitting them to communicate with other neurons. The tentacles that take in signals from other cells and the environment itself are called dendrites, while the tentacles that carry signals from the neuron to other cells are called axons. The interaction of the cell body itself with the outside environment through its dendritic connections and the local conditions in the neuron itself cause the neuron to pump either sodium or potassium in and out, raising and lowering the neuron's electrical potential. When the neuron's electrical potential exceeds a threshold, the neuron fires, creating an action potential that flows down the axons to the synapses and other neurons. The action potential is created when the voltage across the cell membrane of the neuron becomes too large and the cell "fires," creating a spike that travels down the axon to other neurons and cells. If the stimulus causing the buildup in voltage is low, then it takes a long time to cause the neuron to fire. If it is high, the neuron fires much faster.

The firing rate of the neuron can thus be close to zero, as in a case involving no stimulus, to a maximum of approximately 300 pulses per second, as in the case of a stimulus that causes the neuron to fire as fast as possible. Because a neuron is a physical system, it takes time to build up enough charge to cause it to fire. This is where adrenaline comes into play. Adrenaline acts as a bias for the neuron, making it much more likely to fire in the presence of a stimulus. In this book the reader will see how man-made neural networks mimic this potential through weighted interconnections and thresholding terms that allow the artificial neurons to fire more rapidly, just as a biological cell can be biased to fire more rapidly through the introduction of adrenaline. Figure 1.1 is a simplified depiction of a neuron.

The neuron has a central cell body, or soma, with some special attachments, dendrites and axons. The dendrites are special nodes and fibers that receive electrochemical stimulation from other neurons. The axon allows the neuron to communicate with other neighboring neurons. The axon connects to a dendrite fiber through an electrochemical junction known as a synapse. For simplicity, the neuron is depicted with a handful of connections to other cells. In reality, there can be from tens to thousands of interconnections between neurons. The key concept to remember is that neurons receive inputs from other neurons and send outputs to other neurons and cells.

1.2 Modeling Neurons

From the previous section, the reader learned that neurons are connected to other neurons. A simple model of the neuron that shows inputs from other neurons and a corresponding output is depicted in Fig. 1.2. As can be seen in the figure, three neurons feed the single neuron, with one output emanating from the single neuron.

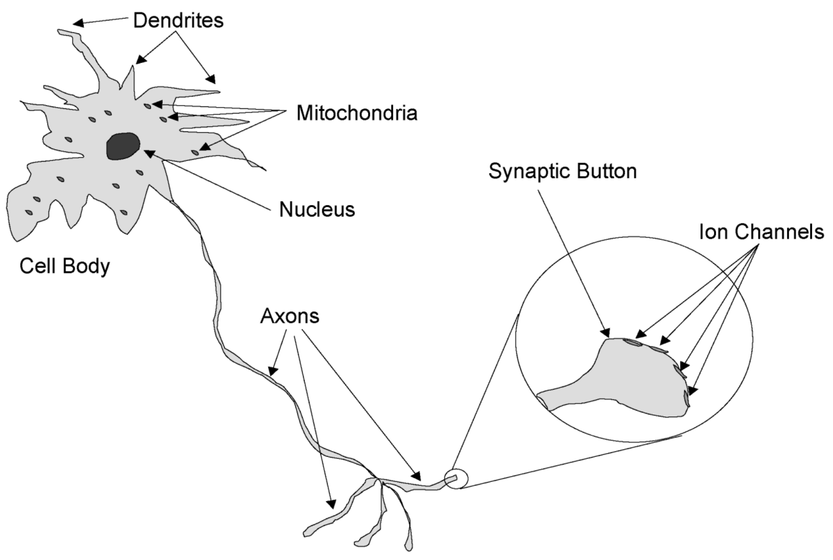


Figure 1.1 Simplified diagram of a neuron.

The reader may recall that neurons send signals to other neurons by sending an action potential down the axon. This is modeled through the use of a transfer function that mimics the firing rate of the neuron action potential, as shown in Fig. 1.3.

Some inputs to the neuron may have more relevance as to whether the neuron should fire, so they should have greater importance. This is modeled by weighting the inputs to the neuron. Thus, the neuron can be thought of as a small computing engine that takes in inputs, processes them, and then transmits an output. The artificial neuron with weighted inputs and a corresponding transfer function is shown in Fig. 1.4. The output for the neuron in Fig. 1.4 is given by

$$z = f\left(\sum_{i=0}^{3} w_i x_i\right). \tag{1.1}$$

The reader has probably already determined that Eq. (1.1) lacks a definition for the transfer function. Many different transfer functions are available to the neural network designer, such as those depicted in Fig. 1.5.

The most commonly used transfer function is the sigmoid or logistic function, because it has nice mathematical properties such as monotonicity, continuity, and differentiability, which are very important when training a neural network with gradient descent. Initially, scientists studied the single neuron with a hard-limiter or step-transfer function. McCulloch and Pitts used an “all or nothing” process to describe neuron activity [McCulloch, 1949]. Rosenblatt [Rosenblatt, 1958] used a hard limiter as the transfer function and termed the hard-limiter neuron a perceptron because it could be taught to solve simple problems. The hard-limiter is an example of a linear equation solver with a simple line forming the decision boundary, as shown in Fig. 1.6.

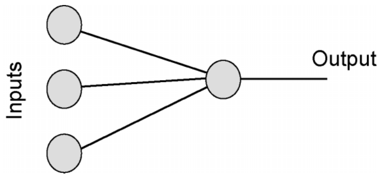


Figure 1.2 Artificial neuron with inputs and a single output.

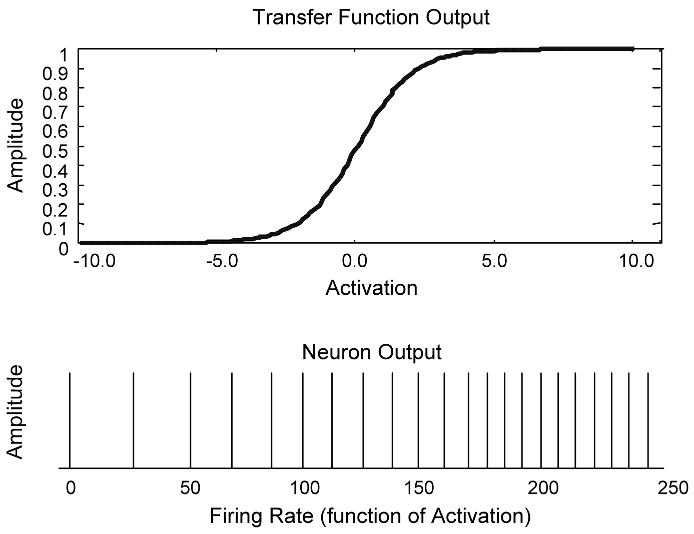


Figure 1.3 Transfer function representation of neuron firing rate.

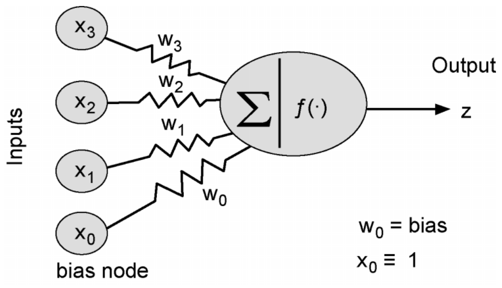


Figure 1.4 Neuron model with weighted inputs and embedded transfer function.

The most common transfer function is the logistic sigmoid function, which is given by the following equation:

$$output = \frac{1}{1 + e^{-(\sum_i w_i x_i + w_0)}}, \tag{1.2}$$

where i is the index on the inputs to the neuron, x_i is the input to the neuron, w_i is the weighting factor attached to that input, and w_0 is the bias to the neuron.

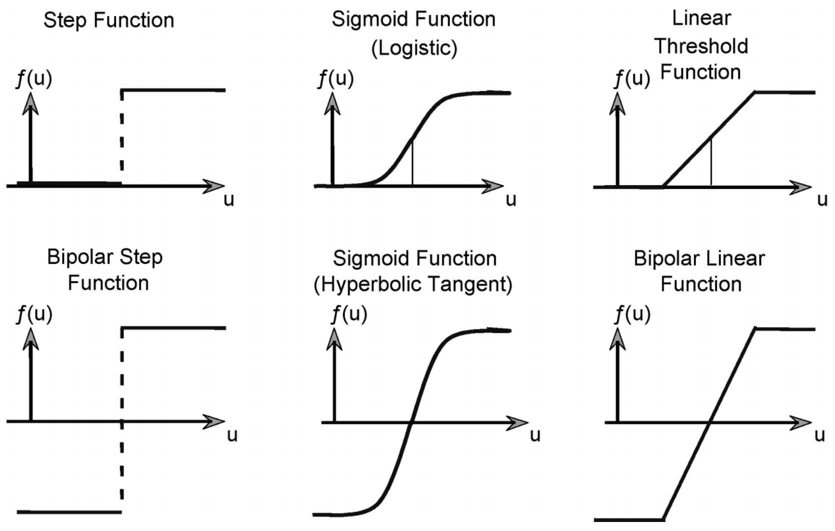

Step Function
f(u)
u
Sigmoid Function (Logistic)
f(u)
u
Linear Threshold Function
f(u)
u
Bipolar Step Function
f(u)
u
Sigmoid Function (Hyperbolic Tangent)
f(u)
u
Bipolar Linear Function
f(u)
u

Figure 1.5 Three different types of transfer function—step, sigmoid, and linear in unipolar and bipolar formats.

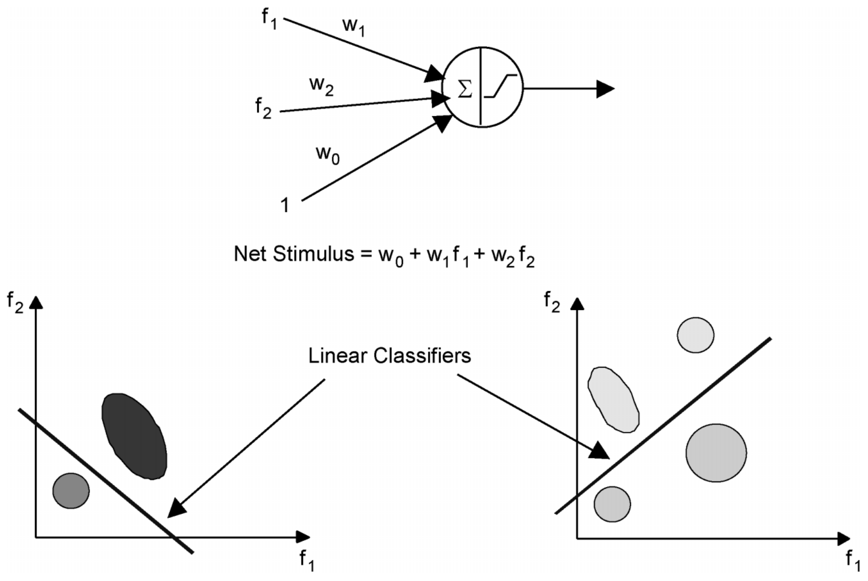


Figure 1.6 Simple neuron model with decision boundaries.

The aggregation and selective use of these decision boundaries is what makes artificial neural networks interesting. Artificial neural networks were created to permit machines to form these decision boundaries with their associated class regions as derived from the data. In fact, a more proper name for artificial neural networks may very well be data-driven computational engines. Neural networks can be used to combine these decision regions together to form higher and higher levels of abstraction, which can result in neural networks with some amazing properties.

To illustrate the how a decision boundary can be formed, consider the neuron in Fig. 1.6 and the associated subfigures. Looking at the subfigure on the lower left, the reader will see a small circle and an ellipse. Each shape represents objects that belong to the same set or class; e.g., the circle could represent all mammals (M) and the ellipse could represent all birds (B). The neuron, combined with a hard-limiter transfer function, can find a line that will separate the two classes {M, B} as depicted by the drawn lines in Fig. 1.6. Each line represents the locus of neuron activation values possible for a given set of weights associated with the neuron. For example, in Fig. 1.6, the reader will see an equation for the neuron net stimulus, or activation, given as

$$\textit{Net Stimulus} = \sum_{i=0}^{2} w_i f_i = w_2 f_2 + w_1 f_1 + w_0, \tag{1.3}$$

which can be rearranged to form

$$f_2 = \frac{w_1 f_1}{w_2} + \frac{(w_0 - \textit{Net Stimulus})}{w_2}, \tag{1.4}$$

which is an equation of a line with a slope of w_1/w_2 and an intercept of $(w_0 - \textit{Net Stimulus})/w_2$ on the f_2 axis. Thus, for a given node with weighted inputs, the activation must lie on the line described by Eq. (1.4).

Decision engines that follow the form of Eq. (1.4) are called linear classifiers. This important observation, that all possible neuron activation values lie on a line, cannot be emphasized enough, because by changing the weights and using a nonlinear transfer function a decision engine can be produced that places one set of objects on one side of the line and other objects or classes on the other side of the line. Thus, by changing w_0, w_1, w_2 and the net stimulus value, the position of the line can be moved until one side of the line contains all the examples of a given class and the other side contains all examples of other classes, as shown in Fig. 1.6. When more than two feature dimensions are used, the equation becomes that of a separating hyperplane. In principle, the concept is similar to the line described previously: On one side of the hyperplane is one class, and on the other side, all the other classes. Whenever a decision engine can produce such a line or hyperplane and separate the data, the data is called a linearly separable set.

One of the best-known linear classifiers is the perceptron, first introduced by Rosenblatt [Rosenblatt, 1958]. The perceptron adjusts its weights using the error vector between a data point and the separating decision line. Changes are made to the weights of the network until none of the training-set data samples produces an error. The perceptron algorithm is presented in simplified form in Table 1.1. The reader will find Matlab code in the appendixes that implements the perceptron algorithm.

The perceptron has the ability to form separating lines and hyperplanes to permit linearly separable classification problems to be solved. Many examples of data exist in the real world in which the classes are not linearly separable, such as that shown in Fig. 1.7. This demonstrates that no single line can separate the green circle from the red kidney-shaped object. It will be shown later that multiple linear classifiers can be combined to produce separable decision regions with a single classifier.

Researchers discovered that combining neurons into layers permits artificial neural networks to solve highly complex classification problems. The next section

Table 1.1 Perceptron Learning Algorithm

```
1. Assign the desired output of the perceptron $\{-1, 1\}$ to each data point in the training set
2. Augment each set of features for each data point with one to form $\mathbf{x} = \{1, x_1, x_2, \ldots, x_n\}$
3. Choose an update step size (eta $\in (0, 1)$) usually eta $= 0.1$
4. Randomly initialize weights $\mathbf{w} = \{w_0, w_1, w_2, \ldots, w_n\}$
5. Execute perceptron weight adjustment algorithm
error_flag = 1
while error_flag = 1
   error = 0
   for ii = 1 to total number of data points in training set
      grab a feature vector ($\mathbf{x}$) and its desired output $\{1, -1\}$
      output = signum($\mathbf{w}\mathbf{x}^T$)
      if output not equal to desired
         $\mathbf{w} = \mathbf{w} - \text{eta} \cdot \mathbf{x}$
error = error + output − desired
      end
   end
   if error = 0
      error_flag = 0
   end
end
```

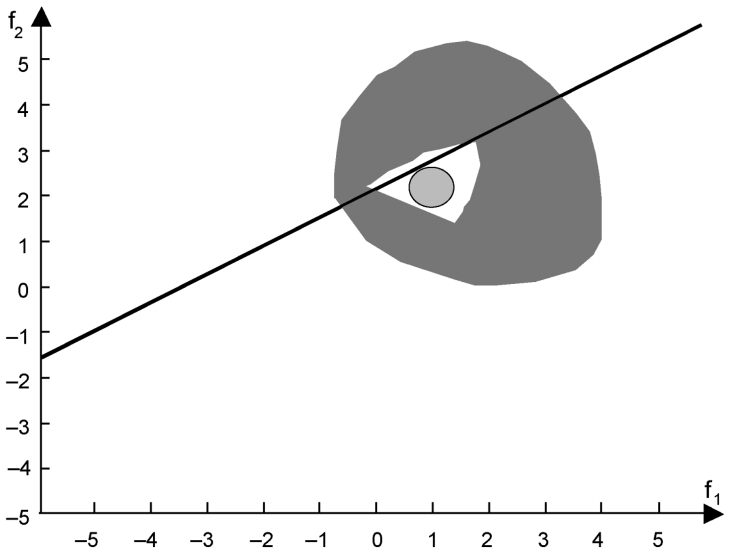


Figure 1.7 Example of a nonlinearly separable classification problem.

introduces the feedforward neural network, which is composed of layers of neurons.

1.3 The Feedforward Neural Network

A depiction of a simple feedforward neural network is given in Fig. 1.8. On the left portion of the figure are inputs to the first layer of neurons, followed by interconnected layers of neurons, and finally with outputs from the final layer of neurons. Note that each layer directly supplies the next layer in the network, feeding the inputs forward through the network, earning this network architecture the feedforward network label. The transfer functions of the neurons do not affect the feedforward behavior of the network. The reader will see many feedforward networks with sigmoid transfer functions, but the neurons in feedforward networks can be any transfer function the designer wishes to use. In addition, the reader should note that a neuron on the first layer could feed a neuron on the third as well as the second layer. Feedforward networks feed outputs from individual neurons forward to one or more neurons or layers in the network. Networks that feed outputs from a neuron back to the inputs of previous layers themselves or other neurons on their own layer are called recurrent networks. Suffice it to say that neural network topologies can vary widely, resulting in differences in architecture, function, and behavior.

Just as the perceptron allowed machines to form decision regions with a single hyperplane, the multilayer feedforward neural network allows machines to form arbitrarily complex decision regions with multiple hyperplanes to solve classification problems as shown in Fig. 1.9.

The process of modifying the weights in a neuron or network to correctly perform a desired input-to-output mapping is termed learning in the neural-network community. As the reader will discover, many methods exist for training neural net-

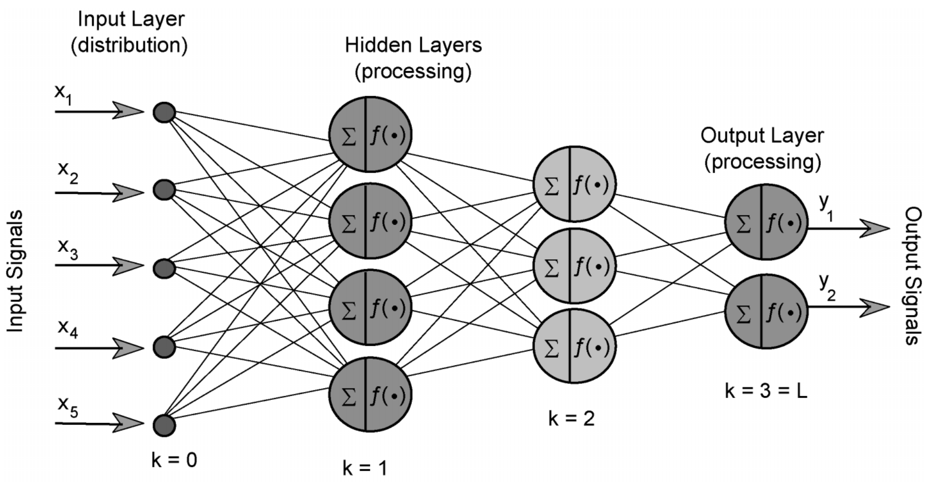
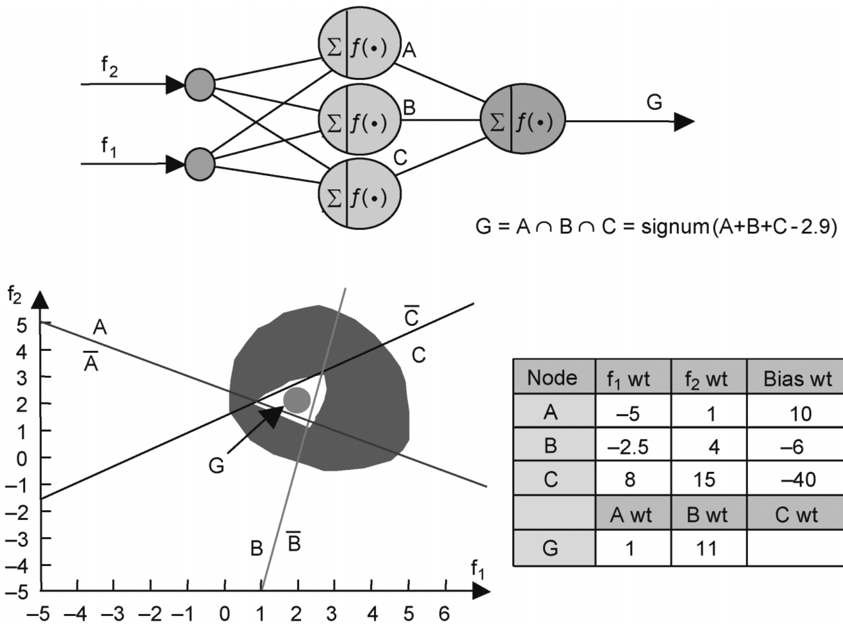


Figure 1.8 Multilayer feedforward neural network.



Node	f_1 wt	f_2 wt	Bias wt	
A	–5	1	10	
B	–2.5	4	–6	
C	8	15	–40	
	A wt	B wt	C wt	Bias wt
G	1	11		–2.9

Figure 1.9 Multilayer feedforward neural network forms complex decision regions to solve nonlinearly separable problem.

works, but all learning is based upon adapting the weights between interconnected neurons. Once a particular network architecture is chosen, the designer must determine which weight to modify to effect the desired network behavior. This is discussed in the next section.

1.3.1 The Credit-Assignment Problem

Once a set of neurons is interconnected, how does the designer train the neurons to do what he/she wants? Which weights should be modified and by how much? These questions are all related to what is often called the credit-assignment problem, which can be defined as the process of determining which weight should be adjusted to effect the change needed to obtain the desired system performance. If things go well, which weight gets the pat on the back with reinforcement of the desired response? Likewise, when things go wrong, which weight gets identified as the guilty party and is penalized? Other questions that need to be answered concern the number of neurons to put in each layer, the number of hidden layers, and the best way to train the network. The answers to all of these questions would fill volumes.

Rosenblatt, with his perceptron, showed some interesting results as long as the classes were linearly separable. Minsky and Papert [Minsky, 1969] showed quite conclusively that there were many classes of problems ill-suited for linear classifiers, so the neural-network community went back to work to find a better solution for the nonlinearly separable class of problems. Werbos showed that by adding layers of neurons and carefully defining their transfer functions to be sigmoids, the credit-assignment problem could be solved and the issues pointed out by Minsky and Pappert overcome. Thus, by using monotonic continuous-transfer functions, Werbos solved the credit-assignment problem—which weight is the slacker or conversely the worker bee—by taking the derivative of the mean squared error with respect to a given weight (see Appendix A for the derivation). This is often referred to as gradient descent, as depicted in Fig. 1.10. Werbos' result allowed researchers to add any number of hidden layers between the input and output. Researchers discovered that problems occurred with the gradient-descent approach because of the time required to train the network as well as a propensity to find local minima in the overall solution space. Local minima are locations in the solution space that form a minimum, which causes network convergence to a non-optimal solution.

1.3.2 Complexity

Some people argue that a neural network is a black box, because either they cannot understand its inner workings or they believe it is too complex a system for the problem. They would prefer a simpler solution that can be easily dissected. A neural network is a complex structure used to solve complex data-analysis problems. Henry Louis Mencken [Mencken, 1917], an early twentieth-century editor, author, and critic, once wrote, "... there is always an easy solution to every human problem—neat, plausible, and wrong."

While this statement was intended as social commentary, it does indicate mankind's desire for easy solutions. Usually, it is best to find the simplest solution to a problem, but the complexity of the solution must be comparable to the complexity of the problem. This does not mean that a designer should build the most

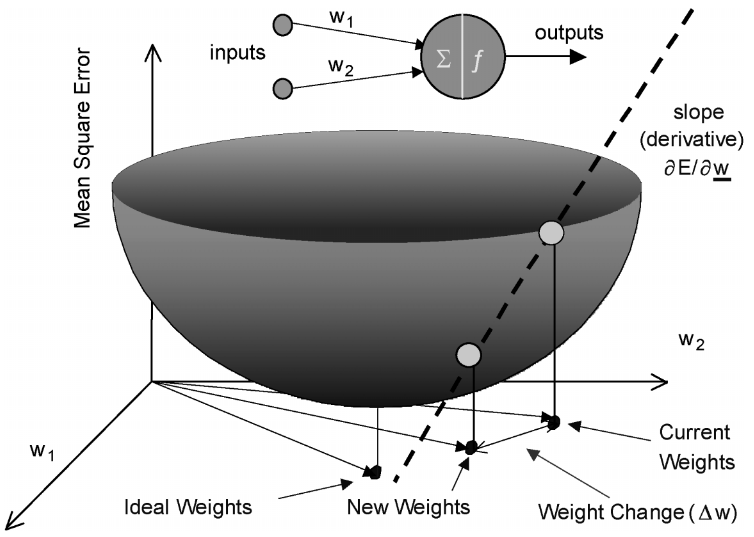


Figure 1.10 Gradient descent for mean-squared-error cost function.