**Bias**: Bias is introduced to make it easy for a perceptron to output 1. It helps to reduce the threshold value to zero. Bias prevents the network from outputting

**Sigmoid Neuron**: Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output. Just like a perceptron, the sigmoid neuron has inputs,,.. But instead of being just 0 or 1, these inputs can also take on any values between 00 and 11. So, for instance, 0.638, Also just like a perceptron, the sigmoid neuron has weights for each input, ,,… and an overall bias, b. But the output is not 0 or 1. Instead, it's σ(w⋅x+b), where σ is called the *sigmoid function* and is defined by:

**Feedforward Neural Network**: Up to now, we've been discussing neural networks where the output from one layer is used as input to the next layer. Such networks are called feedforward neural networks. This means there are no loops in the network - information is always fed forward, never fed back.

**The Problem with Gradient Descent and the Idea of Mini-Batch**: There are a number of challenges in applying the gradient descent rule. We'll look into those in depth in later chapters. But for now I just want to mention one problem. To understand what the problem is, let's look back at the quadratic cost in Equation (6). Notice that this cost function has the form C=1n∑xCxC=1n∑xCx, that is, it's an average over costs Cx≡∥y(x)−a∥22Cx≡‖y(x)−a‖22 for individual training examples. In practice, to compute the gradient ∇C∇C we need to compute the gradients ∇Cx∇Cx separately for each training input, xx, and then average them, ∇C=1n∑x∇Cx∇C=1n∑x∇Cx. Unfortunately, when the number of training inputs is very large this can take a long time, and learning thus occurs slowly.

An idea called *stochastic gradient descent* can be used to speed up learning. The idea is to estimate the gradient ∇C∇C by computing ∇Cx∇Cx for a small sample of randomly chosen training inputs. By averaging over this small sample it turns out that we can quickly get a good estimate of the true gradient ∇C∇C, and this helps speed up gradient descent, and thus learning.

To make these ideas more precise, stochastic gradient descent works by randomly picking out a small number mm of randomly chosen training inputs. We'll label those random training inputs X1,X2,…,XmX1,X2,…,Xm, and refer to them as a *mini-batch*. Provided the sample size mm is large enough we expect that the average value of the ∇CXj∇CXj will be roughly equal to the average over all ∇Cx∇Cx, that is,

∑mj=1∇CXjm≈∑x∇Cxn=∇C,