[**Exercises**](http://neuralnetworksanddeeplearning.com/chap1.html#exercises_191892)

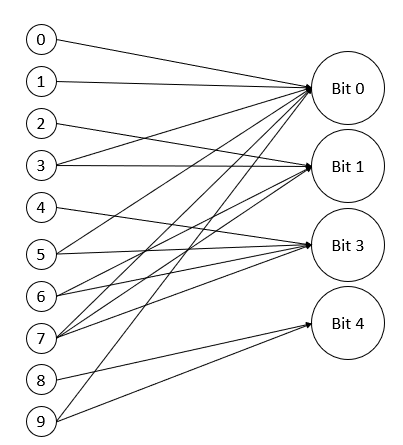
* **Sigmoid neurons simulating perceptrons, part I**   
  Suppose we take all the weights and biases in a network of perceptrons, and multiply them by a positive constant, c>0. Show that the behaviour of the network doesn't change.

[**Exercises**](http://neuralnetworksanddeeplearning.com/chap1.html#exercises_191892)

* **Sigmoid neurons simulating perceptrons, part II**   
  Suppose we have the same setup as the last problem - a network of perceptrons. Suppose also that the overall input to the network of perceptrons has been chosen. We won't need the actual input value, we just need the input to have been fixed. Suppose the weights and biases are such that w⋅x+b≠0 for the input x to any particular perceptron in the network. Now replace all the perceptrons in the network by sigmoid neurons, and multiply the weights and biases by a positive constant c>0. Show that in the limit as c→∞ the behaviour of this network of sigmoid neurons is exactly the same as the network of perceptrons. How can this fail when w⋅x+b=0 for one of the perceptrons?

[**Exercise**](http://neuralnetworksanddeeplearning.com/chap1.html#exercise_513527)

* There is a way of determining the bitwise representation of a digit by adding an extra layer to the three-layer network above. The extra layer converts the output from the previous layer into a binary representation, as illustrated in the figure below. Find a set of weights and biases for the new output layer. Assume that the first 3 layers of neurons are such that the correct output in the third layer (i.e., the old output layer) has activation at least 0.99, and incorrect outputs have activation less than 0.01.



It can be proved that the choice of Δv which minimizes ∇C⋅Δv is Δv=−η∇C, where η=ϵ/∥∇C∥ is determined by the size constraint ∥Δv∥=ϵ.

[**Exercises**](http://neuralnetworksanddeeplearning.com/chap1.html#exercises_647181)

* Prove the assertion of the last paragraph. *Hint:* If you're not already familiar with the [Cauchy-Schwarz inequality](http://en.wikipedia.org/wiki/Cauchy%E2%80%93Schwarz_inequality), you may find it helpful to familiarize yourself with it.

[**Exercises**](http://neuralnetworksanddeeplearning.com/chap1.html#exercises_647181)

* I explained gradient descent when C is a function of two variables, and when it's a function of more than two variables. What happens when C is a function of just one variable? Can you provide a geometric interpretation of what gradient descent is doing in the one-dimensional case?

.

[**Exercise**](http://neuralnetworksanddeeplearning.com/chap1.html#exercise_263792)

* An extreme version of gradient descent is to use a mini-batch size of just 1. That is, given a training input, xx, we update our weights and biases according to the rules wk→w′k=wk−η∂Cx/∂wk and bl→b′l=bl−η∂Cx/∂bl. Then we choose another training input, and update the weights and biases again. And so on, repeatedly. This procedure is known as *online*, *on-line*, or *incremental* learning. In online learning, a neural network learns from just one training input at a time (just as human beings do). Name one advantage and one disadvantage of online learning, compared to stochastic gradient descent with a mini-batch size of, say, 20.

Online learning allows real-time update to the model, and the more recent the input, the more relevant it becomes. This may be desirable if we are modeling a time-series.

Mini-batch size of 1 may lead to very slow learning and may achieve sub-optimal result, because only very small sample is exposed to parameter update in each step. This may lead to good approximation of local minimum but misses global minimum.

[**Exercise**](http://neuralnetworksanddeeplearning.com/chap1.html#exercise_717502)

* Write out Equation (22) in component form, and verify that it gives the same result as the rule (4) for computing the output of a sigmoid neuron.

Eq. 22

Rule 4

[**Exercise**](http://neuralnetworksanddeeplearning.com/chap1.html#exercise_420023)

* Try creating a network with just two layers - an input and an output layer, no hidden layer - with 784 and 10 neurons, respectively. Train the network using stochastic gradient descent. What classification accuracy can you achieve?

About 11% (which is not that different from just guessing)