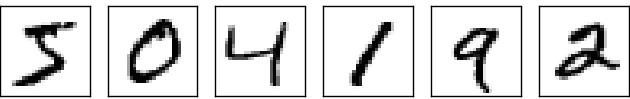
**A simple network to classify handwritten digits:**

We can split the problem of recognizing handwritten digits into two sub-problems.

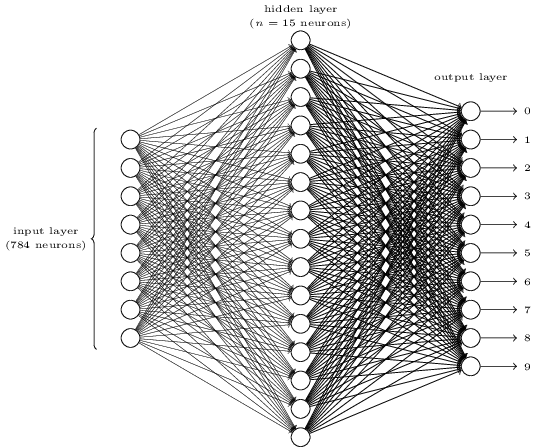
* First, we'd like a way of breaking an image containing many digits into a sequence of separate images, each containing a single digit.
  + Ex: image to separate images of each digit.
  +  to 
* Once the image has been segmented, the program then needs to classify each individual digit.

**Classifying individual digits using segmentation problem:**

There are many approaches to solving the segmentation problem.

* One approach is to Trial many different ways of segmenting the image, using the individual digit classifier to score each trial segmentation.
* A trial segmentation gets a high score if the individual digit classifier is confident of its classification in all segments.
* A trial segmentation gets a low score if the classifier is having a lot of trouble in one or more segments.
* The idea is that if the classifier is having trouble somewhere, then it's probably having trouble because the segmentation has been chosen incorrectly.

**To recognize individual digits we will use a three-layer neural network:**



* **Input layer:**
  + The input layer of the network contains neurons encoding the values of the input pixels.
  + As input is of 28\*28-pixel images of scanned handwritten digits, and so the input layer contains 28×28 = 784 neurons.
  + The input pixels are greyscaled, with a value of 0.0 representing white, a value of 1.0 representing black, and in between values representing gradually darkening shades of grey.
* **Hidden Layer(second layer):**
  + The second layer of the network is a hidden layer.
  + We denote the number of neurons in this hidden layer by n, and we'll experiment with different values for n.
* **output layer:**
  + The output layer of the network contains 10 neurons.
  + If the first neuron fires, then that will indicate that the network thinks the digit is a 0, Similarly If the second neuron fires digit is a 1. And so on.
  + A little more precisely, we number the output neurons from 0 through 9 and figure out which neuron has the highest activation value.
* Tnetworksork with 10 output neurons learns to recognize digits better than the network with 4 output neurons.
* There is a way of determining the bitwise representation of a digit by adding an extra layer to the neural network. The extra layer converts the output from the previous layer into a binary representation.

**Learning with gradient descent:**

* The first thing we'll need is a data set to learn from - a so-called training data set.
* We'll use the MNIST data set, which contains tens of thousands of scanned images of handwritten digits, together with their correct classifications.
* When testing our network we'll ask it to recognize images which aren't in the training set.
* We'll use the notation x to denote a training input. It is 784-dimensional vector. Each entry in the vector represents the grey value for a single pixel in the image.
* We'll denote the corresponding desired output by y=y(x), where y is a 10-dimensional vector.
* So we use cost function which lets us find weights and biases. C(w,b)≡1/2n(∑x(∥y(x)−a∥)2.
* The process of repeatedly nudging an input of a function using some multiple of the negative gradient is called gradient descent.
* By repeatedly applying this update rule we can "roll down the hill", and hopefully find a minimum of the cost function. In other words, this is a rule which can be used to learn in a neural network.