Sigmoid neurons simulating perceptrons, part I

Suppose we take all the weights annd 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.

remember that a perceptron is defined as

$$\text{output} = \begin{cases} 0 \text{ if } w \cdot x + b \leq 0 \\ 1 \text{ if } w \cdot x + b > 0 \end{cases}$$

$$c(w \cdot x + b) \leq c0$$

$$cw \cdot cx + bc \leq 0$$

and

$$c(w \cdot x + b) > c0$$

$$cw \cdot cx + bc > 0$$

note we can simplify the above equation by defining $w' = cw, x' = cx, b' = bc$ then

$$\text{output} = \begin{cases} 0 \text{ if } c' \cdot x' + b' \leq 0 \\ 1 \text{ if } c' \cdot x' + b' > 0 \end{cases}$$

and we have the same form as before, so the behavior of the network doesn't change.

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

$w \cdot x + b \neq 0$ 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 weigths and biases by a positive constant $c > 0$. Show that in the limit as $c \to \infty$ the behaviour of this network of sigmoid neurons is exactly the same as the network of perceptrons. How can this fail when $w \cdot x + b = 0$ for one of the perceptrons?

remember the definition of the sigmoid neuron

$$\sigma(z) = \frac{1}{1+\exp^{-z}}$$

$$z = w \cdot x + b$$

now consider multiple neurons

$$\text{outputsys} = \Sigma_{j=0}^{n} \sigma_{j(z)}$$

$$\text{outputsys} = \Sigma_{j=0}^{n} \frac{1}{1+\exp^{-(w_j \cdot x_j + b_j)}}$$

$$\text{outputsys} = \Sigma_{j=0}^{n} \frac{1}{1+\exp^{-(c(w_j \cdot x_j + b_j))}}$$

$$\text{outputsys} = \lim_{c \to \infty} \left(\Sigma_{j=0}^{n} \frac{1}{1+\exp^{-(cw_j \cdot cx_j + cb_j)}} \right)$$

$$\text{outputsys} = \lim_{c \to \infty} \left(\Sigma_{j=0}^{n} \frac{1}{1+\exp^{\pm\infty}} \right)$$

$$\text{outputsys} = \lim_{c \to \infty} \left(\Sigma_{j=0}^{n-k} \frac{1}{\infty} = 0 \right) + \left(\Sigma_{j=n-k+1}^{k-1} 1 \right)$$

so indeed is the desired behavior, considering all perceptrons follow $w \cdot x + b \neq 0$.

if one of the perceptrons outputs 0, then

$$\text{outputsys} = \lim_{c \to \infty} \left(\Sigma_{j=0}^{n-1} 1 + \frac{1}{1+\exp^{-c(w_n \cdot x_n + b_n)}} \right)$$

$$\text{outputsys} = \lim_{c \to \infty} \left(\Sigma_{j=0}^{n-1} 1 + \frac{1}{1+\exp^{-0}} \right)$$

$$\text{outputsys} = \lim_{c \to \infty} \left(\left(\Sigma_{j=0}^{n-k} 1 \right) + \left(\Sigma_{j=(n-k)+1}^{k-2} 0 \right) + \frac{1}{2} \right)$$

one of the neurons output $\frac{1}{2}$ a value not valid for a perceptron, thus it won't be a valid perceptron emulation anymore.

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

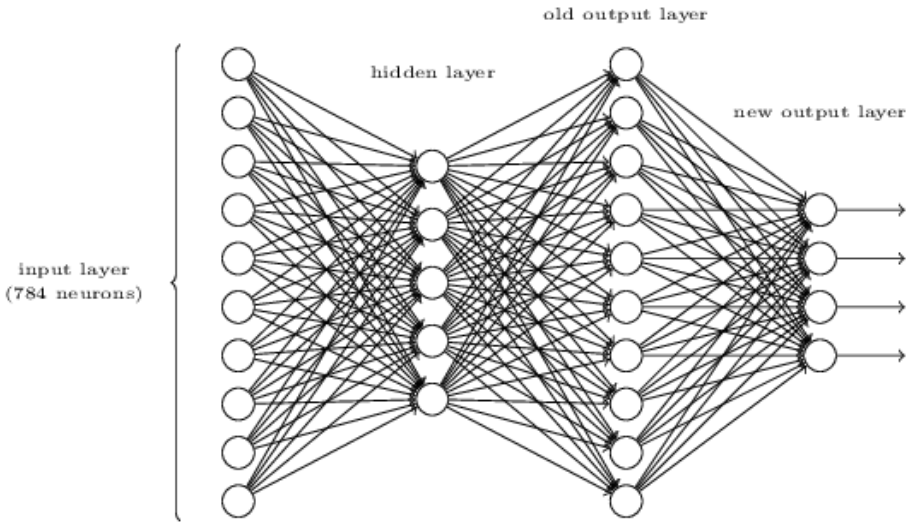


Figure 1: source: http://neuralnetworksanddeeplearning.com/chap1.html

let's review binary representation from 0 to 9, also consider the MSB(most significant bit) to be mapped to the top neuron in the output layer and the LSB(least significant bit) to be mapped to the bottom neuron in the output layer.

$$0_{10} = 0000_b$$

$$1_{10} = 0001_b$$

$$2_{10} = 0010_b$$

$$3_{10} = 0011_b$$

$$4_{10} = 0100_b$$

$$5_{10} = 0101_b$$

$$6_{10} = 0110_b$$

$$7_{10} = 0111_b$$

$$8_{10} = 1000_b$$

$$9_{10} = 1001_b$$

now the weights and biases

$$w_0 = \begin{bmatrix} \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.01} & \frac{1}{0.01} \end{bmatrix}^T, b_0 = -99$$

$$w_1 = \begin{bmatrix} \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.01} & \frac{1}{0.01} & \frac{1}{0.01} & \frac{0}{0.01} & \frac{1}{0.99} & \frac{1}{0.99} \end{bmatrix}^T, b_1 = -99$$

$$w_2 = \begin{bmatrix} \frac{1}{0.99} & \frac{1}{0.99} & \frac{1}{0.01} & \frac{1}{0.01} & \frac{0}{0.99} & \frac{1}{0.99} & \frac{1}{0.01} & \frac{1}{0.01} & \frac{1}{0.99} & \frac{1}{0.99} \end{bmatrix}^T, b_1 = -99$$

$$w_3 = \begin{bmatrix} \frac{1}{0.99} & \frac{1}{0.01} & \frac{1}{0.99} & \frac{1}{0.01} & \frac{1}{0.99} & \frac{1}{0.01} & \frac{1}{0.99} & \frac{1}{0.01} & \frac{1}{0.99} & \frac{1}{0.01} \end{bmatrix}^T, b_1 = -99$$

now we test

assume 0 triggered in the old layer

$$\frac{1}{0.99} * 0.99 + \left(\frac{0.01}{0.99}\right) * 7 + \left(\frac{0.01}{0.01}\right) * 2 = 1 + \frac{7}{99} + 2 - 99 < 0, \text{so the MSB bit is reset}$$

$$\frac{1}{0.99} * 0.99 + \left(\frac{0.01}{0.99}\right) * 5 + \left(\frac{0.01}{0.01}\right) * 4 = 1 + \frac{5}{99} + 4 - 99 < 0, \text{so the bit before the MSB is reset}$$

$$\frac{1}{0.99} * 0.99 + \left(\frac{0.01}{0.99}\right) * 5 + \left(\frac{0.01}{0.01}\right) * 4 = 1 + \frac{5}{99} + 4 - 99 < 0, \text{so this bit is also reset}$$

$$\frac{1}{0.99} * 0.99 + \left(\frac{0.01}{0.99}\right) * 5 + \left(\frac{0.01}{0.01}\right) * 5 = 1 + \frac{5}{99} + 5 - 99 < 0, \text{so the LSB is also reset}$$

meaning the output is 0000 as we expected, but we can also check 5 for instance, two bits are expected to be activated

assume 5 triggered in the old layer

$$\frac{1}{0.99} * 0.99 + \left(\frac{0.01}{0.99}\right) * 7 + \left(\frac{0.01}{0.01}\right) * 2 = 1 + \frac{7}{99} + 2 - 99 < 0, \text{so the MSB is reset, as expected}$$

$$\left(\frac{0.99}{0.01}\right) + \left(\frac{0.01}{0.99}\right) * 6 + \left(\frac{0.01}{0.01}\right) * 3 = 99 + \frac{6}{99} + 3 - 99 > 0, \text{bit set}$$

$$\left(\frac{0.99}{0.99}\right) + \left(\frac{0.01}{0.99}\right) * 5 + \left(\frac{0.01}{0.01}\right) * 4 = 1 + \frac{5}{99} + 4 - 99 < 0, \text{so bit reset}$$

$$\left(\frac{0.99}{0.01}\right) + \left(\frac{0.01}{0.99}\right) * 5 + \left(\frac{0.01}{0.01}\right) * 4 = 99 + \frac{5}{99} + 4 - 99 > 0, \text{so bit set}$$

we can see the output is 0101 as expected, analogous tests can be applied to other outputs in the "old output layer"

Prove the assertion of the last paragraph. *Hint*: If you're not already familiar with the *Cauchy-Schawarz inequality*, you may find it helpful to familiarize yourself with it.

We'll derive the proof without the inequality by applying analytic geometry techniques

$$\nabla C \cdot \Delta v$$

given the above expression we need to minimize it

note that both ∇C and Δv are vectors, thus we can start by projecting the Δv vector in the ∇C one.

$$\Delta v = \frac{< \nabla C, \Delta v >}{\|\nabla C\|\|\nabla C\|}\nabla C$$

we also know the dot product of both vectors can be expressed as

$$\cos\theta = \frac{< \nabla C, \Delta v >}{\|\nabla C\|\|\Delta v\|}$$

$$\cos\theta\|\nabla C\|\|\Delta v\| =< \nabla C, \Delta v >$$

thus substituting the last equation in the projection one

$$\Delta v = \frac{\cos\theta\|\Delta v\|}{\|\nabla C\|} = (\cos\theta)\frac{\varepsilon}{\|\nabla C\|}\nabla C$$

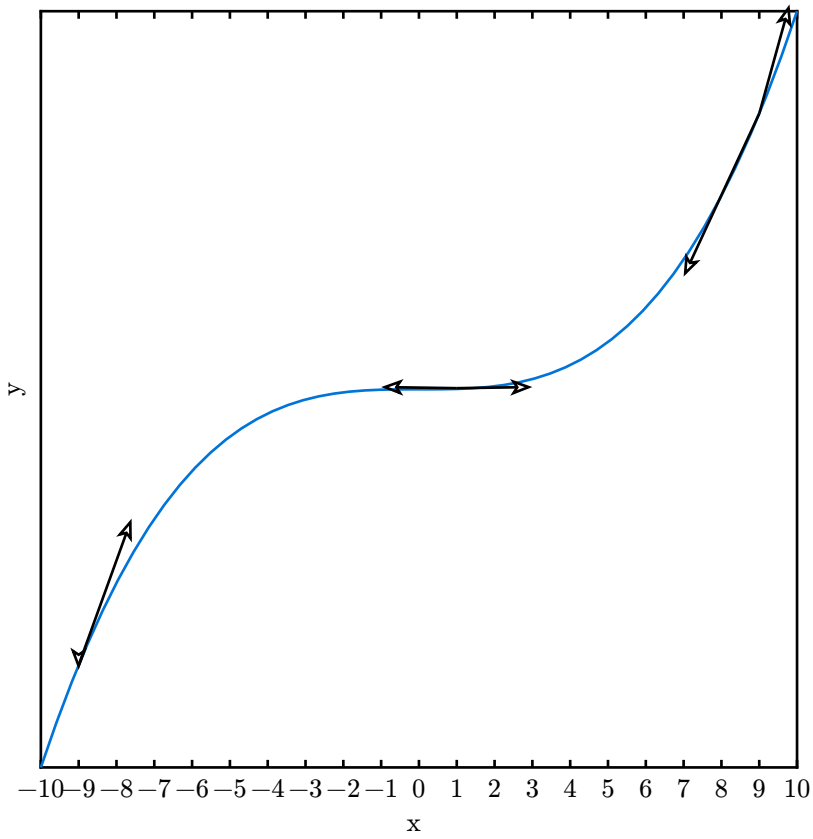
remember we want to minimize this function so we choose $\cos\theta = -1 \Rightarrow \theta = \pi$

and we arrive at the desired equation

$$\Delta v = -\eta\nabla C, \eta = \frac{\varepsilon}{\|\nabla C\|}$$

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 interpretaion of what gradient descent is doing in the one-dimensional case?

imagine a cubic curve



note that if we start from the leftmost point the algorithm would halt, due to the fact that, due to domain constraints, there's no where else to go, however if we start in the rightmost side, one can see that eventually the algoritm is going to halt in the region where $\frac{df}{dx} = 0$ a saddle point or maybe in the leftmost point, *depends on the learning rate*, meaning that the algorithm is effectively slicing a plane,if we consider this path to belong to a surface, and looking for points where $\frac{df}{dx} = 0$ that happens in sadddle points, global and local minima.

An extreme version of gradient descent is to use a mini-batch size of just 1. That is, given a training input, x, we update our weights and biases according to the rules $w_k \to w'_k = w_k - \eta \frac{\partial C_x}{\partial \omega_k}$ and $b_l \to b'_l = b_l - \eta \frac{\partial C_x}{\partial b_l}$. 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 learnig, a neural network learns from just one training input at a time (just as huamn beings do). Name one advantage and one disadvantage of online learning, compared to stochastic gradient descent with a mini-batch size of, say, 20.

a obvious advantage is a parameter update takes a single iteration, while in the mini-batch size 20 iterations, so a 20x speedup in the first case.

another not so obvious advantage is increased guarantees of cost function convergence. If during the algorithm execution a saddle point or local minima is found the extra ammount of noise, that won't be averaged when comparing to the mini-batch size case, allows to escape such points and thus might not need a momentum parameter for the algorithm.

a disadvantage is excess in noise thus increasing the variance in the performance metrics while training, however such case is minimized in the mini-batch size due to the larger batch size reducing noise per parameter,ω_i, b_i, update.