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

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

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

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

$$\text{and}$$

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

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

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

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

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

$$\text{now consider multiple neurons}$$

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

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

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

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

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

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

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

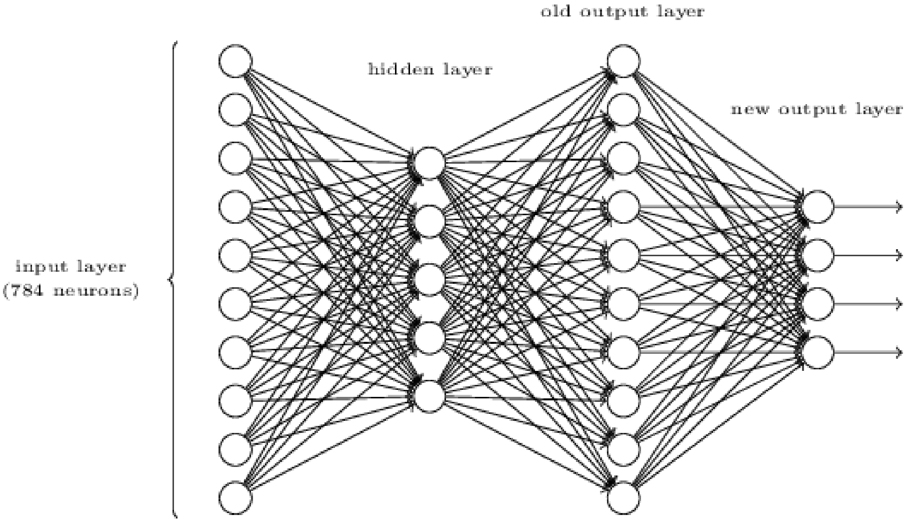


Figure 1: source: http://neuralnetworksanddeeplearning.com/chap1.html

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

$$\text{assume 0 triggered in the old layer}$$

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

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

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

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

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

$$\text{assume 5 triggered in the old layer}$$

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

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

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

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

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

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

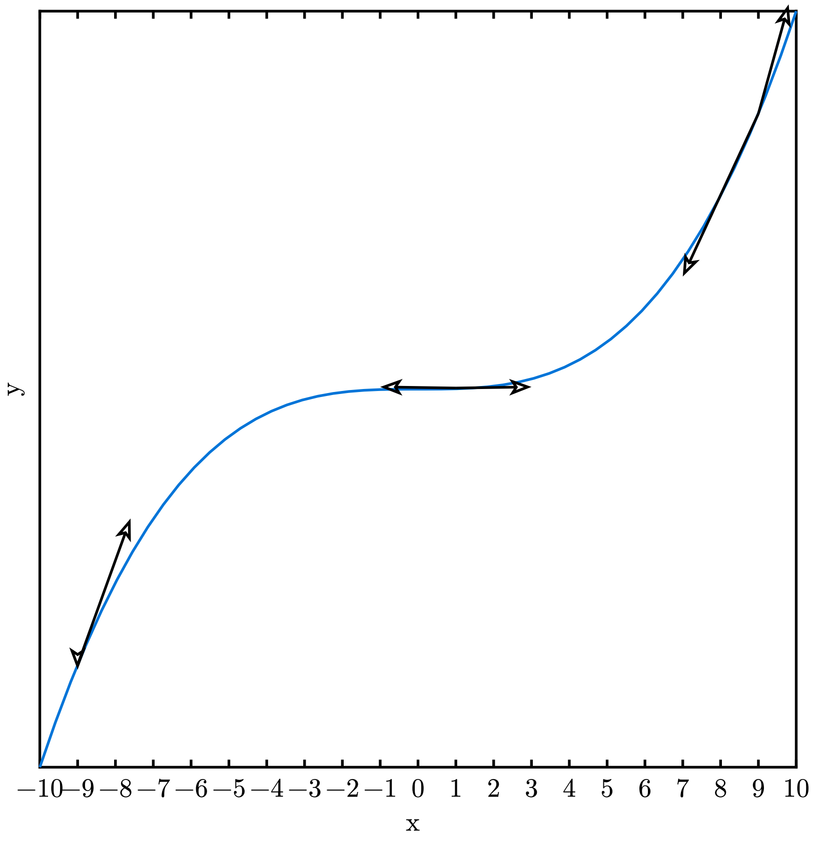
remember we want to minimize this function so we choose $\cos\theta = -1 \Rightarrow \theta = \pi$

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.

Prove equations BP3 and BP4

remember the following, already proved by the author

$$\delta_j^l = \frac{\partial C}{\partial z_k^l} \text{ , error due to weighted sum in the j-th neuron in the l-th layer}$$

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \text{ , j-th neuron in the l-th layer weighted sum}$$

now we need to prove the following

$$\frac{\partial C}{\partial b_j} = \delta_j^l$$

this can be easily be proven by chain rule application

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l$$

the $\frac{\partial z_j^l}{\partial b_j^l} = 1$

this proves BP3

now we prove BP4, again by applying the chain rule

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial w_{jk}^{l}}$$

and the partial derivative for $\frac{\partial z_{j}^{l}}{\partial w_{jk}^{l}} = a_{k}^{l-1}$

thus we have $\frac{\partial C}{\partial w_{jk}^{l}} = a_{k}^{l-1} \delta_{j}^{l}$ as desired.

Backpropagation with a single modified neuron Suppose we modify a single neuron in a feedforward network so that the output from the neuron is given by $f\left(\Sigma_{j} w_{j} x_{j} + b\right)$, where f is some function other than the sigmoid. How should we modify the backpropagation algorithm in this case?
First we start by computing the error due to the weighted sum in the last layer, the output layer.

$$\frac{\partial C}{\partial z_{j}^{L}} = \frac{\partial C}{\partial a_{j}^{L}} \frac{\partial a_{j}^{L}}{\partial z_{j}^{L}}$$

note the modification in this case, the second term will change for $j = h$

thus

$$\frac{\partial C}{\partial z_{h}^{L}} = \frac{\partial C}{\partial a_{h}^{L}} \left(\partial a_{h}^{L}\right) f'\left(z_{h}^{L}\right)$$

and for the other case, j $\neq$ h

$$\frac{\partial C}{\partial z_{j}^{L}} = \frac{\partial C}{\partial a_{j}^{L}} \left(\partial a_{j}^{L}\right) \sigma'\left(z_{j}^{L}\right)$$

now we consider how this impacts the backpropagate step

$$\frac{\partial C}{\partial z_{j}^{l}} = \Sigma_{k} \frac{\partial C}{\partial z_{k}^{l+1}} \frac{\partial z_{k}^{l+1}}{\partial z_{j}^{l}}$$

remembering the notation for $\frac{\partial C}{\partial z_{k}^{l}} \equiv \delta_{k}^{l}$

$$\delta_{l}^{j} = \delta_{k}^{l+1} \Sigma_{k} \left(\Sigma_{j \neq h} w_{kj}^{l+1} a_{j}^{l} + b_{k}^{l+1} + w_{kh}^{l+1} f'\left(z_{h}^{l}\right)\right)$$

the updates won't change their form, only the output error and backpropagate

Backpropagation with linear neurons Suppose we replace the usual non-linear σ function with $\sigma(z) = z$ throughout the network. Rewrite the backpropagation algorithm for this case.
again we start by computing the output error

$$\delta_{j}^{L} \equiv \frac{\partial C}{\partial z_{j}^{L}}$$

and by chain rule

$$\frac{\partial C}{\partial z_{j}^{L}} = \frac{\partial C}{\partial a_{j}^{L}} \frac{\partial a_{j}^{L}}{\partial z_{j}^{L}}$$

note that the activation function changed, thus the partial derivative is 1

$$\frac{\partial C}{\partial z_{j}^{L}} = \frac{\partial C}{\partial a_{j}^{L}}$$

now we consider the backpropagate task

$$\delta_j^l \equiv \Sigma_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

thus we can simplify this to

$$\delta_j^l = \delta_k^{l+1} \frac{\partial\left(\Sigma_j w_{kj}^{l+1} f\left(z_j^l\right) + b_k^{l+1}\right)}{\partial z_j^l}$$

and we simplify this again to

$$\delta_j^l = \Sigma_k w_{kj}^{l+1} \delta_k^{l+1}$$