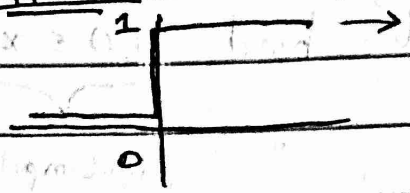
Discussion: backprop

Clarifications for HW10

Problem 1: approximation w/ ReLUs

- This is an example of universal approximators from last time (approximate any function with NN)

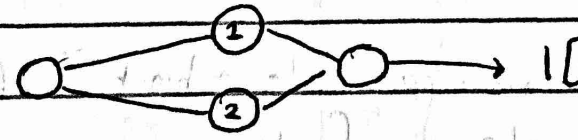
a) - You are asked to approximate a step function $1[x > 0]$



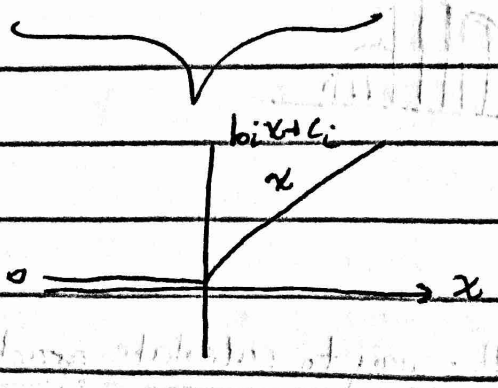
with 2 ReLU units,

$$\sum_{i=1}^{2} a_i \max(0, b_i x + c_i)$$

We use ReLU as our activation function in the following network (remember from HW9):



$$= a_1 \max(0, b_1 x + c_1) + a_2 \max(0, b_2 x + c_2)$$



ReLU = max(0, x)

in this case max(0, bix+ci)

In our case we want

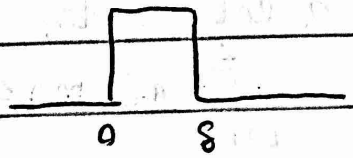
$$a_1(b_1 x + c_1) + a_2(b_2 x + c_2) = 1$$ (forget about ∅ case - ReLU takes care of that)

How can we choose our parameters such that this equation holds? (What is a, b, c)
(We can see we need to subtract one ReLU from another to approximate the step function...)
* Note that it will not be perfect!

b) Using 4 ReLU's, build $\mathbb{1}[0 \le x \le \delta]$

Unit impulse of width δ

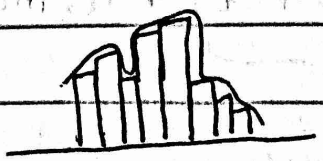
from part a



↑
One step function can be subtracted from another (at δ) to get a box function

In this way, if we approximate a bunch of step/impulse functions, we can approximate <u>any function</u>

i.e.



Backpropagation

Backpropagation is <u>the way to calculate gradients</u> so that we can update our weights. Numerical vs. analytic gradient

$$\frac{\text{error}(w+h) - \text{error}(w)}{h}$$

over millions of parameters = no

So for example, to train we would —

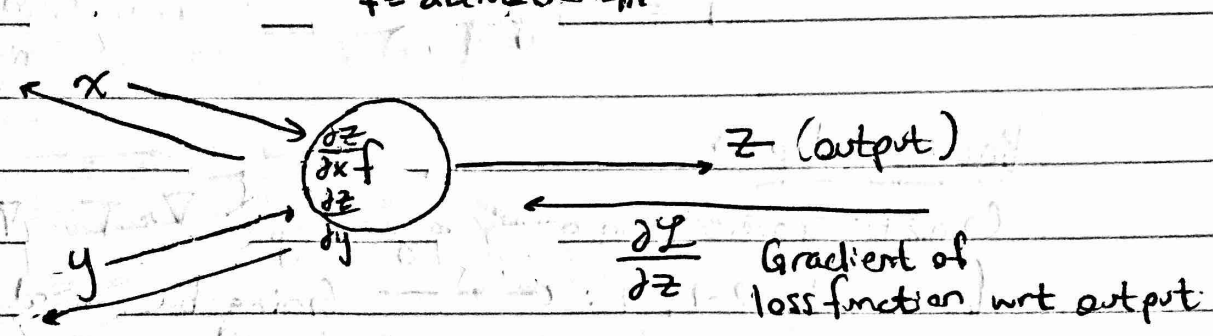
1. define model
2. define error

} any model/error that needs weight updates. not LDA, for example

3. Find analytical derivative of error for every parameter w_i (backpropagation)
4. update w_i (gradient descent)

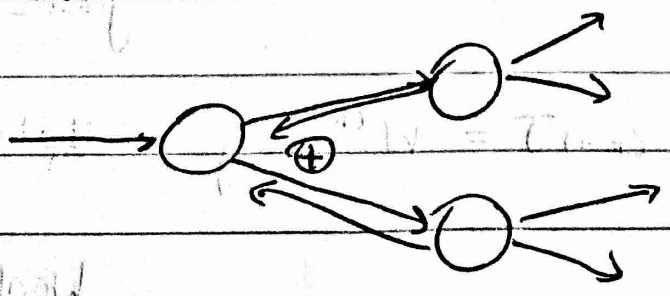
Backprop example (from class) Recursive application of the chain rule

f = activation fn



$\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}$ = local gradients

Gradients will add at branches —



To do this for every node we need some general function, described in HW10.

Forward pass

$h^{(0)} = x$ "Hidden layer output" on first run

for $k = 1$ to ℓ: ℓ = # of weights to update/calculate

$a^{(k)} = b^{(k)} + W^{(k)} h^{(k-1)}$ ← weight x hidden layer output

$h^{(k)} = f(a^{(k)})$ ← f = activation fn, $h^{(k)}$ = current hidden output

end

$\hat{y} = h^{(\ell)}$ ← prediction

$J = \mathcal{L}(y, \hat{y})$ ← Loss (did we get it right?)

Backward pass

Compute gradient on output layer $g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} \mathcal{L}(y, \hat{y})$

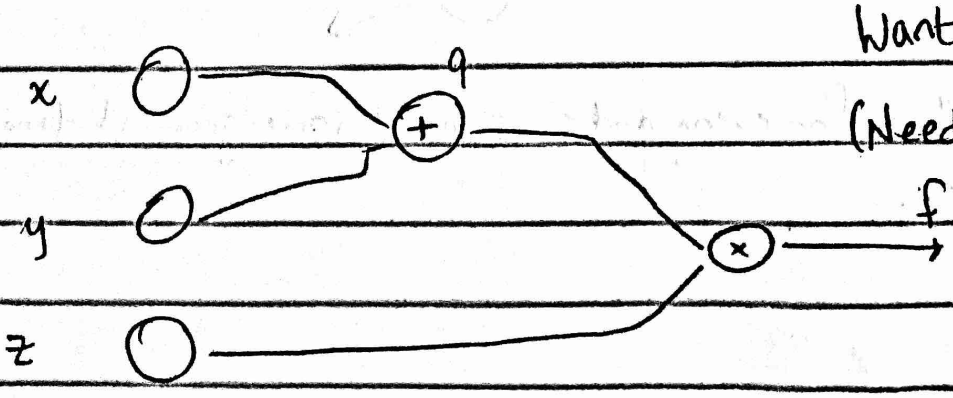
for $k = \ell, \ell-1, \ldots, 1$: ← Going backwards

$g \leftarrow \nabla_{a^{(k)}} J = g \odot f'(a^{(k)})$ ← How output of this layer should change to reduce error

$\nabla_{b^{(k)}} J = g$

$\nabla_{W^{(k)}} J = g \cdot h^{(k-1)T}$ ← We this to get gradient of parameters

$g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)T} g$ Update



Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

(Need $\frac{\partial f}{\partial q}$ to get it)