Recall the situation, the Deep, feed forward neural net (also called MLP = multi layered perceptron) has the form

$$F(x, \eta) = F_L \circ \dots \circ F_1(x) \quad \text{with each}$$

$$F_i(x) = \sigma(A_L x + \vec{b}_L)$$

- The parameters are $\eta = A_1, \dots, A_L, b_1, \dots, b_L$
- The training data is $x_1, \dots, x_p$ with correct output $y_1, y_2, \dots, y_N$
- The loss or error or objective function is (simplest version)

$$\Phi(\eta) = \frac{1}{N} \sum_{i=1}^{N} \| F(x_i, \eta) - y_i \|^2$$

- Learning consists of adjusting the parameters η so that the error diminishes.

- So Φ depends on F which depends on $F_L \dots F_1$ as a composition
- This is a job for the chain Rule for multidimensional functions.
- First recall the derivative as a matrix

If $f : \mathbb{R}^n \to \mathbb{R}^m$ so

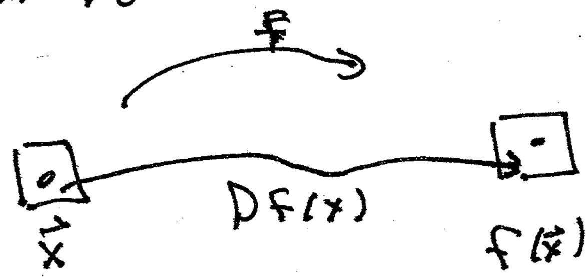
$$f(x_1, \dots, x_n) = (f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n))$$

Then

$$Df = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

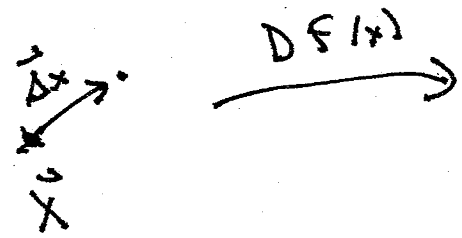
or $(Df)_{ij} = \frac{\partial f_i}{\partial x_j}$ Thus as a linear transformation $Df : \mathbb{R}^n \to \mathbb{R}^m$ also.

- The Derivative $Df(\vec{x})$ is the best local Linear approximation to f at $\vec{x}$



- Taylor's Theorem

$$f(\vec{x}+\Delta\vec{x}) = f(\vec{x}) + Df(\vec{x})\,\Delta x \quad + \text{h.o.t}$$



- example $f(x_1, x_2) = (x_1^2 x_2, x_1 + x_2)$

$$Df(x_1, x_2) = \begin{bmatrix} 2x_1 x_2 & x_1^2 \\ 1 & 1 \end{bmatrix}$$

- The Chain Rule says that when taking the derivative, composition of functions becomes multiplication of matrices

$$D(g \circ f)(x) = Dg(f(x)) \cdot Df(x)$$

note Dg is evaluated at $f(x)$.

example

$g(y_1, y_2) = y_1^2 + y_2^3$. Let $h = g \circ f$

$f(x_1, y_1) = (x_1^2 x_2, x_1 + x_2)$ $\qquad Dg(y) = [2y_1, 3y_2^2]$

$$\Rightarrow \quad Dh(x) = Dg(f(x)) \cdot Df(x)$$

$$= \left[2(x_1^2 x_2), 3(x_1 + x_2)^2\right] \begin{bmatrix} 2x_1 x_2 & x_1^2 \\ 1 & 1 \end{bmatrix}$$

$$= \left[4x_1^3 x_2^2 + 3(x_1 + x_2^2),\ 2x_1^4 x_2 + 3(x_1 + x_2)^2\right]$$

Now lets get back to the main job of understanding $\nabla \Phi$, which we want to write using composition of functions and the chain Rule

- For simplicity, let's work with just one pieces of training data at first so

$$\Phi(\eta) = \|F(x_1, \eta) - y_1\|^2$$

- To think about this using the chain rule we treat it as the composition of functions

$$\alpha(z) = \|z\|^2, \quad \beta(a) = a - y_1$$

So $\Phi(m) = \alpha \circ \beta \circ F(x_1, \eta)$

then $\nabla\Phi(\eta) = D\Phi(\eta) = D\alpha(\beta \circ F(x_1,\eta)) \cdot D\beta(F(x_1,\eta)) \cdot DF(x_1,\eta)$

- The first two pieces are easy

$$\alpha(z) = z_1^2 + \dots + z_p^2, \quad D\alpha = 2[z_1, z_2, \dots, z_n] = 2z$$

- $D\beta = Id$

- so we have

$$\nabla \Phi(\eta) = 2\left(F(x_1, \eta) - y_1\right) \cdot DF(x_1, \eta)$$

- So the real work is in computing DF.

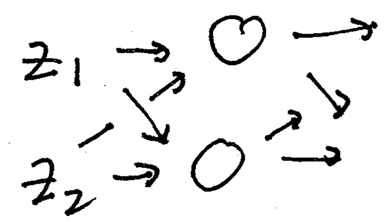
- Now recall $F = F_L \circ \dots \circ F_1$ and so (omitting arguments) $DF = DF_L \cdots DF_1$

or more properly

$$DF(\eta) = DF_L(F_{L-1} \circ \dots \circ F_1(\eta))\, DF_{L-1}(F_{L-2} \cdots F_1(\eta)) \cdots DF_1(\eta).$$

- Now the $F_j \circ \dots \circ F_1(\eta)$ are computed when we propagate x_1 through the network so the real issue is DF_j

To simplify even further, we focus on a layer with just two neurons and two inputs



so $F(z, \eta) = \sigma(Az+b)$

$= \left(\sigma(A_{11} z_1 + A_{12} z_2 + b_1), \sigma(A_{21} z_1 + A_{22} z_2 + b_2)\right)$

recalling that $\eta = A_{11}, A_{12}, A_{21}, A_{22}, A_{22}, b_1, b_2$

and letting $a_1 = A_{11} z_1 + A_{12} z_2 + b_1$ $a_2 = A_{21} z_1 + A_{22} z_2 + b_2$

$$DF = \begin{bmatrix} \sigma'(a_1) z_1 & \sigma'(a_1) z_2 & 0 & 0 & \cdots & \sigma'(a_1) & 0 \\ 0 & 0 & \sigma'(a_2) z_1 & \sigma'(a_2) z_2 & \cdots & 0 & \sigma'(a_2) \end{bmatrix}$$

$\underbrace{\qquad\qquad}_{\frac{\partial F}{\partial A_{ij}}}$ $\underbrace{\qquad}_{\frac{\partial F}{\partial b_i}}$

- This looks complicated, but is easy to compute since each term like $A_{11} z_1 + A_{12} z_2 + b_1$ has been computed already while forward propagating x_1 through the net

- Now we have a huge formula:

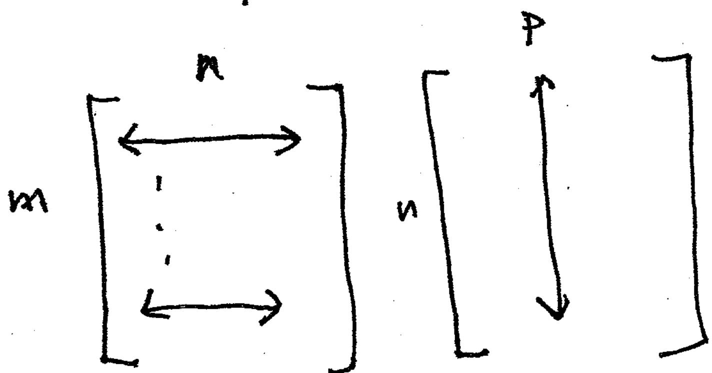
$$\nabla \Phi(\eta) = 2\,(F(x_1 - \eta) - y_1)\, DF_L\, DF_{L-1} \cdots DF_1$$

and how do we compute it?

- The answer is back propagation which means from the left to the right in the formula which is from the output to the input in the net.

- Let's see why that is by a simple example

First, if A is $m \times n$ and B is $n \times p$ how many multiplications are needed to compute AB?



each row & col is n multiplications and $n-1$ additions (so we just track the mult)

There are n rows in A and p col. in B so

TOT # MULT = $m n p$.

• Now, in analogy to our formula for $\nabla\Phi$, we examine ABC where

A is 1×100
B is 100×100
C is 100×100

then (AB) requires $1 \cdot 100 \cdot 100 = 10^4$ and is 1×100

so $(AB)C$ requires $1 \times 100 \times 100 = 10^4$

so computing from the left costs $10^4 + 10^4 = 2 \times 10^4$

on the other hand

(BC) requires $100 \cdot 100 \cdot 100 = 10^6$ and is 100×100

so $A(BC)$ requires $1 \cdot 100 \cdot 100 = 10^4$ so computing from the right costs 1.01×10^6

or about 50 times more than from the left

- The difference gets even more pronounced with larger matrices.

- Our whole discussion was based on a single input of training data, when there are N pieces it formally looks the same but ~~ever~~ things become vectors and matrices. We save for the HW.

There is lot's more to say even about just deep, feed forward nets, but we make just two more points.

- We have been using the quadratic, or least squares loss function. More common is to use the cross entropy

$$C(\eta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \ln(F(x_i,\eta)) + (1-y_i)\ln(1-F(x_i,\eta))$$

(after a normalization)

- There is a trick to make $Ax+\vec{b}$ into a single matrix multiplication. Let

$$\hat{A} = \begin{bmatrix} 1 & 0 \\ b & A \end{bmatrix}$$ and augment the input $\hat{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$

So $\hat{A}\hat{x} = \begin{bmatrix} 1 \\ Ax+b \end{bmatrix}$ So the 2nd components becon $Ax+b$ and the first remains 1

and note $\sigma_R(1) = \sigma_S(1) = 1$, so it stays 1 after activation.