Neural Networks
(Deep Learning)

In kernel method we made linear methods non linear by implicit (fixed) feature mapping $x \mapsto \phi(x)$ (which was possibly infinite dimensional)

Neural networks, can think of the method as explicitly learning this feature map ϕ. Invented by Frank Rosenblatt in 1963 at Cornell!

A single neuron: $\sigma(u^T x + c)$

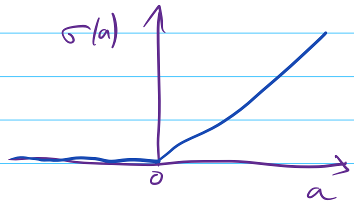
$\sigma(a)$ is a non-linear activation function

u are the weights of the neuron(s) we will learn

Eg: Activation functions:

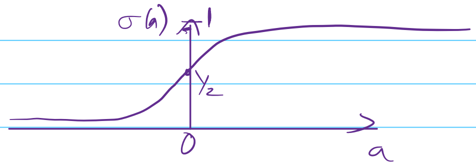
Relu: Rectified linear unit

$$\sigma(a) = \max\{a, 0\}$$



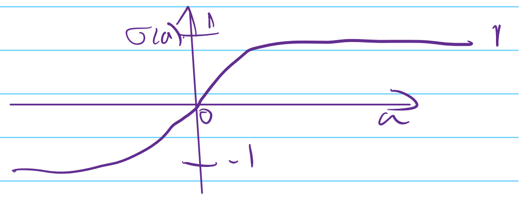
Sigmoid:

$$\sigma(a) = \frac{e^a}{1+e^a}$$

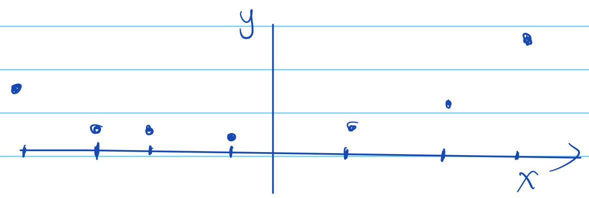


tanh:

$$\sigma(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$



Regression with a single neuron:

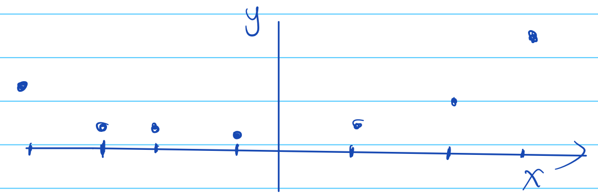


A single hidden layer of neurons: (regression with k neurons)

$$\phi(x) = \sigma(U\vec{x} + \vec{c})$$

$\vec{c}$ is a k dimensional vector

U is a kxd matrix



eg ReLU $\quad h(x) = w^T\phi(x) + b = w^T\sigma(Ux + \vec{c}) + b$

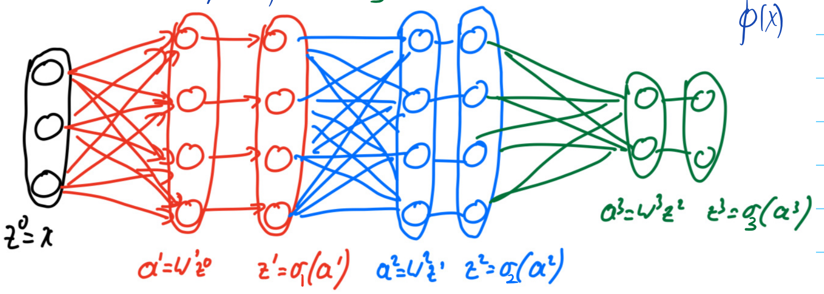
$$= w^T \max(Ux + \vec{c}, \vec{0}) + b$$

What happens to the regression if $\sigma(a) = a$ (linear activation)?

Now lets try regression with RELU.

Deep Neural Network: We can stack multiple hidden layers one after the other, first layer gives linear boundaries, second layer combines corresponding non-linear functions, third layer combines non-linear functions output in second layer and so on. Depth quickly gives high expressibility.

$$\phi(x) = \sigma_3(C\sigma_2(B\sigma_1(Ax + \vec{a}) + \vec{b}) + \vec{c})$$



In above example, how many neurons, how many parameters?

Forward Pass (prediction):

$z_0 = x$ (Input layer)

Typically σ_L is different from others

FOR $i = 1$ to L

$\quad a_i = W_i z_{i-1} + \vec{b}_i$

$\quad z_i = \sigma_i(a_i)$

END

Return z_L

As usual bias can be aborted in each layer by inserting 1 to each layer as extra feature

We need to learn the weights and biases of each layer.
We will use gradient descent (or its cousin)!

How easy is it to compute gradients?

Warmup with 1 dimensional example:

$x \in \mathbb{R} \qquad C, c, B, b, A, a \in \mathbb{R}$

$$\ell(h(x), y) = \frac{1}{2}(h(x) - y)^2$$

$$h(x) = \underbrace{C\,\overbrace{\sigma(\overbrace{B\,\underbrace{\sigma(\overbrace{A\,\underbrace{x}_{z_0} + a}^{a_1})}_{z_1} + b}^{a_2})}^{z_2} + c}_{z_3}$$

ℓ' derivative of Loss

σ' derivative of σ

gradients w.r.t.

C, c:

$$\frac{\partial \ell(h(x), y)}{\partial C} = \ell'(h(x), y) \times \frac{\partial z_3}{\partial C} = \ell'(h(x), y) \times z_2$$

$$\frac{\partial \ell(h(x), y)}{\partial c} = \ell'(h(x), y) \times \frac{\partial z_3}{\partial c} = \ell'(h(x), y)$$

Already computed

B, b:

$$\frac{\partial \ell(h(x), y)}{\partial B} = \ell'(h(x), y) \times \frac{\partial z_3}{\partial B} = \ell'(h(x), y)\, \sigma'(a_2) \frac{\partial a_2}{\partial B}$$

$$= \ell'(h(x), y)\, \sigma'(a_2)\, z_1$$

$$\frac{\partial \ell(h(x), y)}{\partial b} = \ell'(h(x), y)\, \sigma'(a_2)$$

$$\vec{a} \odot \vec{r} = \begin{bmatrix} a_1 r_1 \\ a_2 r_2 \\ a_3 r_3 \\ \vdots \end{bmatrix}$$

elementwise product

Backward pass (gradient step)

$$\vec{\delta}_L = \nabla \ell(z_L, y) \odot \sigma_L'(a_L)$$

For $j = L-1$ to 1

$$W_j = W_j - \eta \vec{\delta}_j z_{j-1}^T$$

$$b_j = b_j - \eta \vec{\delta}_j$$

$$\vec{\delta}_{j-1} = \sigma'(a_{j-1}) \odot W_j^T \delta_j$$

End

1. Compute a's and z's from forward pass (store them)
2. Run backward pass using a's and z's from forward pass (computes gradients)