Neural Nets

Tuesday, May 30, 2017 9:51 AM

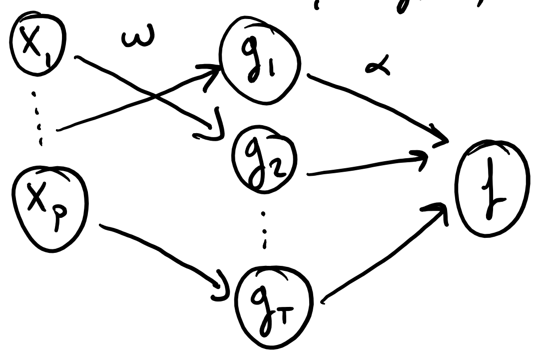
Software: theano (conda install)

Neural nets use composition to build non-linear fncts from linear (+ a few simple non-linear)

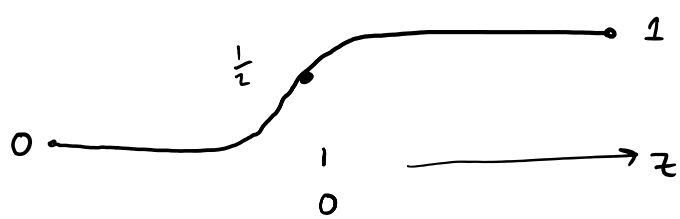
Adaboost:

stump decision tree $g_i(x) = \mathbb{1}\{w_i^T x > 0\}$

ensemble $\sum_i \alpha_i g_i(x) = f(x)$



Define the sigmoid: $\sigma(z) = \dfrac{1}{1+e^{-z}}$



is a smooth surrogate for $\mathbb{1}\{z > 0\}$

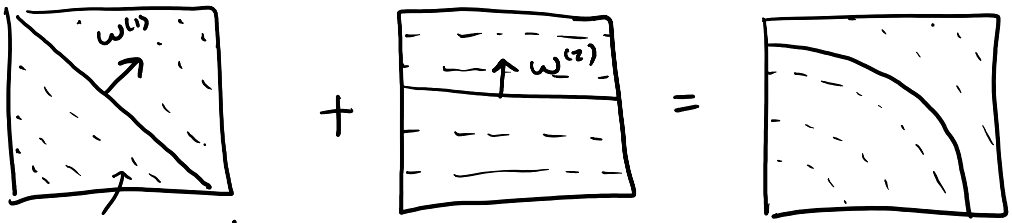
For a weight vector $w^{(1)}$ construct feature $h_1 = \sigma(w^{(1)T} x)$

$w^{(2)}$ . . . $h_2 = \sigma(w^{(2)T} x)$

⋮

$w^{(H)}$ . . . . $h_H = \sigma(w^{(H)T} x)$

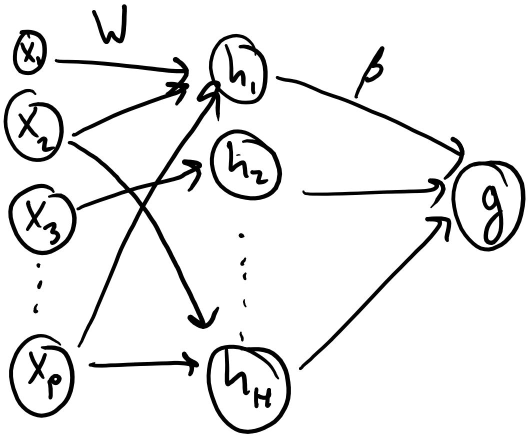
Combine $h_1, \dots, h_H$ w/ linear classifier $g(\beta^T h) > 0$?



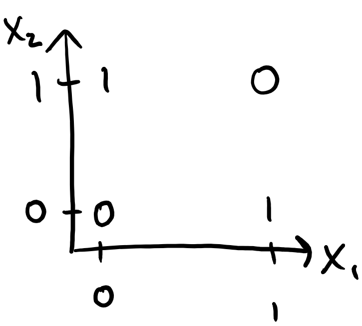
level sets of $\sigma(w^{(1)T} x)$

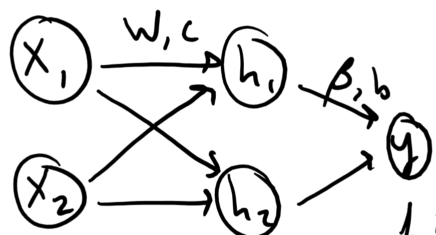
¿ σ = identity? $\quad g(\beta^T h) = g\left(\beta^T \underbrace{\begin{pmatrix} w^{(1)T} x \\ \vdots \\ w^{(H)T} x \end{pmatrix}}_{W^T}\right) = g\left((W\beta)^T x\right)$

Composing linear functions is linear.



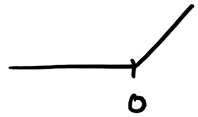
XOR





f is output

$$\sigma(z) = z_+ = \begin{cases} z, & z > 0 \\ 0, & \dots \end{cases}$$



Rectified linear unit ReLU

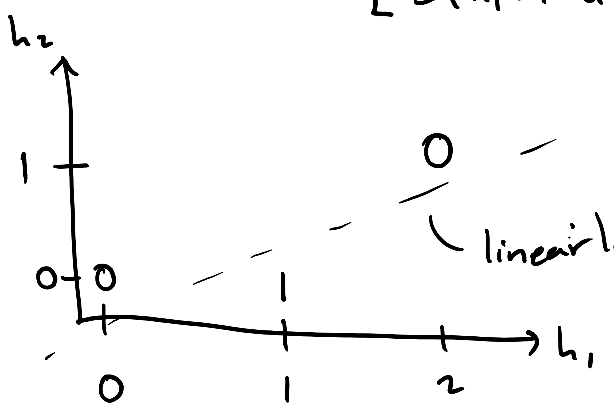
$$f(x \mid W, c, \beta, b) = \beta^T h + b$$

$$= \beta^T (W^T x + c)_+ + b$$

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad \beta = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$W^T x = \begin{bmatrix} x_1 + x_2 \\ x_1 + x_2 \end{bmatrix} \qquad W^T x + c = \begin{bmatrix} x_1 + x_2 \\ x_1 + x_2 - 1 \end{bmatrix}$$

$$(W^T x + c)_+ = \begin{bmatrix} x_1 + x_2 \\ \mathbb{1}\{x_1 = 1 \ \& \ x_2 = 1\} \end{bmatrix}$$



Backprop.

Tuesday, May 30, 2017 10:56 AM

Fit the n-net as if you are fitting a linear classifier using last hidden layer...

$$\text{Emp. Risk} \equiv R_n = \frac{1}{n}\sum_i \ell(y_i, \beta^T h(x_i))$$

$$x \xrightarrow{W^{(1)}} g^{(1)}(x) \xrightarrow{W^{(2)}} \dots \longrightarrow g^{(L)}(x) \xrightarrow{W^{(L+1)}} h(x) \xrightarrow{\beta} y$$

$$\frac{\partial}{\partial \beta} R_n = \frac{1}{n}\sum_i \frac{\partial}{\partial \beta} \ell(y_i, \beta^T h(x_i)) = \frac{1}{n}\sum_i \ell'(y_i, \beta^T h(x_i)) \cdot h(x_i)$$

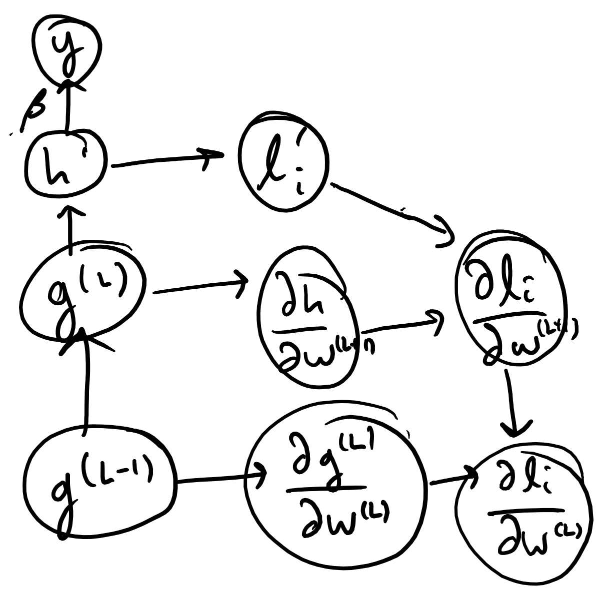
▷ to compute grad wrt. β need $h(x)$ and access to ℓ'

$$\frac{\partial}{\partial W^{(L+1)}_{jk}} \underbrace{\ell(y_i, \beta^T h(x_i))}_{\ell_i(\beta^T h)} = \sum_{\substack{\text{hidden}\\ m}} \frac{\partial \ell_i(\beta^T h)}{\partial h_m} \cdot \underbrace{\frac{\partial h_m}{\partial W^{(L+1)}_{j,k}}}_{0 \text{ if } k \neq m}$$

$$h(x) = \sigma\left(W^T g^{(L)}(x)\right)$$

$$\nabla_W \ell_i = \left(\frac{\partial h}{\partial W}\right)^T \underbrace{\nabla_h \ell_i}_{\ell_i'}$$

↑ Jacobian



feed forward:

Apply N-net to x_i

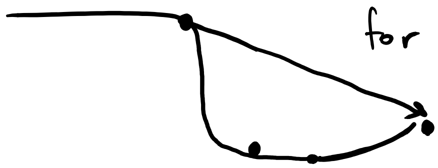
backprop.

Apply chainrule backwards

Apply chainrule backwards

Adding other things:

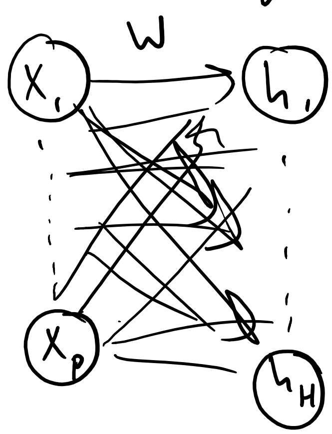
- regularization terms to R_n for parameters
  $\|\cdot\|_1$, $\|\cdot\|_2^2$, etc.
- deep networks have many layers: with enough layers ReLU's can approx. large classes of functions



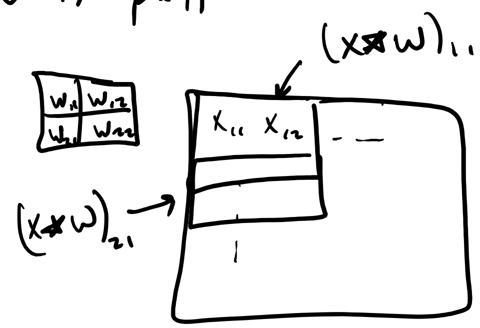
for deep nnets gradients can have "cliffs"

- clipped gradient: $\|g\|_2 > v$: $g \leftarrow \frac{g}{\|g\|} \cdot v$

X is an image (p is # of pixels)



W is $p \times H$



Convolution: $(x * w)_i = \sum_j x_j w_{i-j}$ 1-dim

$$(x * w)_{ij} = \sum_{k,l} x_{kl} w_{i-k,j-l} \quad \text{2-dim}$$

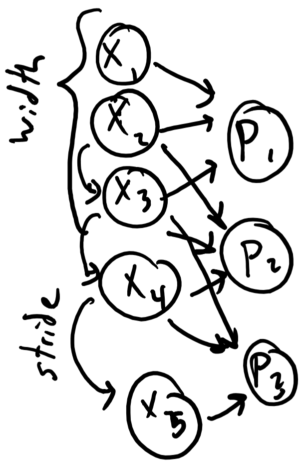
$X \xrightarrow{w} C$ $c = x * w$

▷ usually ω is supported over fewer pixels ($K << p$)

params K vs. $p \cdot H$ (hidden layer)

▷ translation invariant prediction

▷ pooling layers - max, average



$$P_1 = \max(x_1, x_2, x_3) \quad (\max)$$

$$= \frac{1}{3}(x_1 + x_2 + x_3) \quad (\text{ave})$$

▷ process is called downsampling

change "scale"

$$\omega \longrightarrow \beta \longrightarrow y$$

$$\frac{\partial}{\partial \omega_i} y = \sum_j \frac{\partial y}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial \omega_i}$$