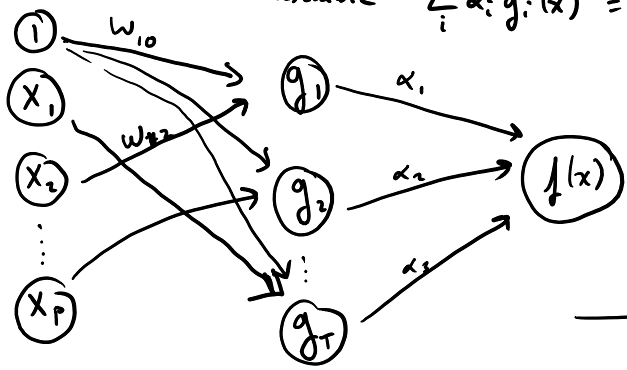
Neural Networks

Monday, May 20, 2019 12:02 PM

Neural nets use composition $(f \circ g(x) = f(g(x)))$ to build non-linear functions from linear ones & simple non-linear functions.

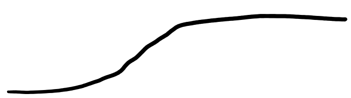
Like Adaboost: stumps $g_i(x) = \mathbb{1}\{w_i^T x > 0\}$ ← $(s_i, 0, \dots, w_{i,j}, \dots, 0)$

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



linear non-linear activation linear

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



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

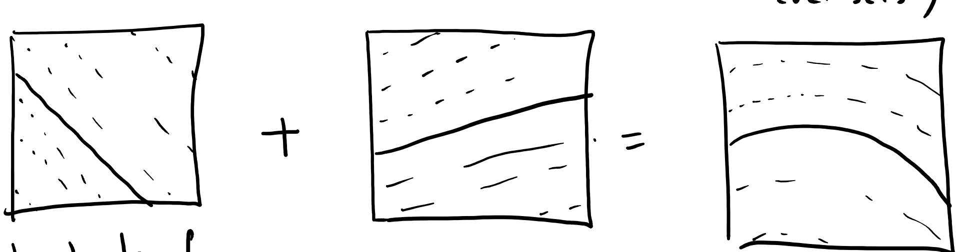
Hidden units: weight vector w_1 constuct feature $h_1 = \sigma(w_1^T x)$

w_2 ... $h_2 = \sigma(w_2^T x)$

$\vdots$

w_H .... $h_H = \sigma(w_H^T x)$

Combine $h_1, \dots, h_H$ we get $f(x) = \sum_{j=1}^{H} \beta_j h_j(x)$ can look non-linear (non-linear level sets)



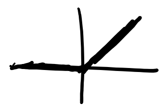
level sets of $\sigma(w_1^T x)$ + $\sigma(w_2^T x)$ =

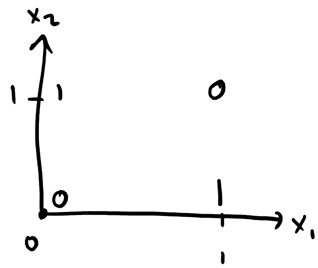
Why not $\sigma(z) = z$ (identity)

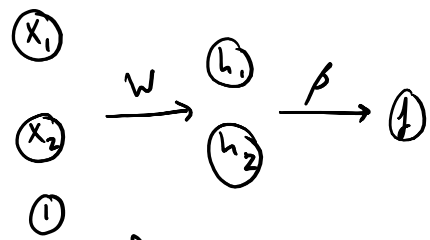
$$f(x) = \sum_{i=1}^{H} \beta_i h_i(x) = \sum_{i=1}^{H} \beta_i w_i^T x = \Big(\underbrace{\sum_{i=1}^{H} \beta_i w_i}_{\tilde{\beta}} \Big)^T x$$

Other common option: rectified linear unit, ReLU

$$\text{ReLU}(z) = z_+ = \begin{cases} z, & z > 0 \\ 0, & \ldots \end{cases}$$





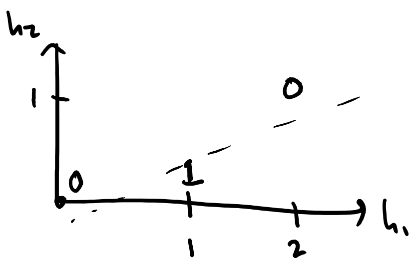


$$W^T x = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & -1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ x_1 + x_2 - 1 \end{pmatrix}$$

$h_1 = (x_1 + x_2)_+$ – use ReLU

$h_2 = (x_1 + x_2 - 1)_+$

$$\beta = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$



$$\underset{\substack{\uparrow \\ \text{vectorized}}}{\beta^T h} = (x_1 + x_2)_+ - 2\,(x_1 + x_2 - 1)_+$$

Optimization for Deep Learning

Monday, May 20, 2019 12:02 PM

Recall empirical risk: $R_n = \frac{1}{n} \sum_i \ell(y_i, f(x_i))$

Full gradient: $\frac{\partial R_n}{\partial w_j} = \frac{1}{n} \sum_i \frac{\partial}{\partial w_j} \ell(y_i, f(x_i))$

stochastic: $\frac{\partial}{\partial w_j} \ell(y_i, f(x_i))$

mini-batch: sample $i = 1, \dots, m$

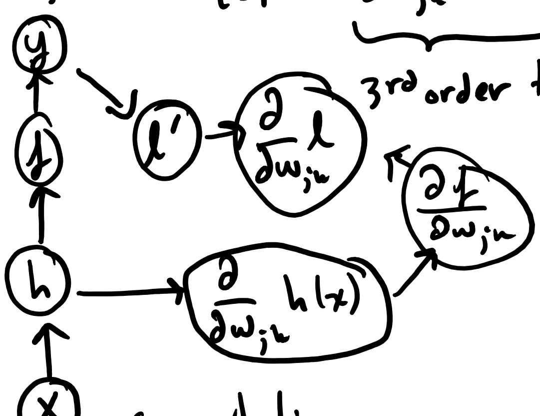
$$\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_j} \ell(y_i, f(x_i))$$

$$\frac{\partial}{\partial w_{jk}} \ell(y_i, f(x_i)) = \underset{\uparrow \frac{\partial}{\partial z} \ell(y_i, z)}{\ell'(y_i, f(x_i))} \cdot \frac{\partial f}{\partial w_{jk}}(x_i) \quad \text{(chain)}$$

$$\frac{\partial f(x_i)}{\partial w_{jk}} = \sum_{l=1}^{H} \beta_l \underbrace{\frac{\partial}{\partial w_{jk}} h_l(x_i)}_{\text{3rd order tensor}}$$

$h_l(x) = \sigma(w_l^T x)$

$\frac{\partial}{\partial w_{jk}} h_l(x) = \sigma'(w_l^T x) \cdot \cdot \begin{cases} x_k, & l = j \\ 0 & l \neq j \end{cases}$

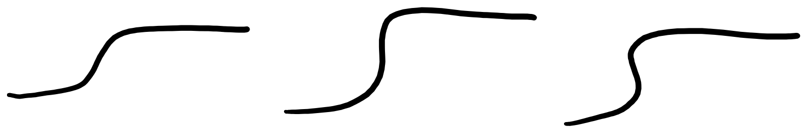


computation graph

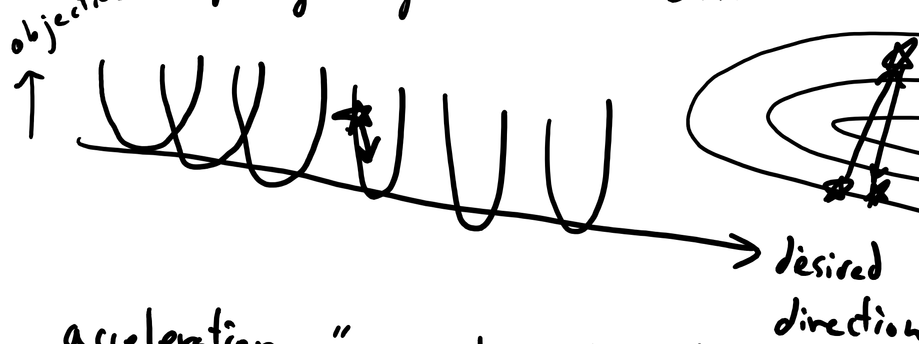
general procedure: backprop

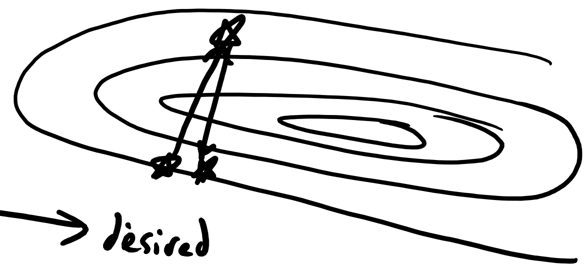
Deep Nnet: $H^{(l)}$ dim

$$x \to h^{(1)} \to h^{(2)} \to \dots \to h^{(L)} \to y$$



composing sigmoid $\to$ cliff's





acceleration "remembers" past directions in a momentum term

$$g \leftarrow \alpha g - \varepsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} \ell(y_i, f_\theta(x_i)) \right)$$

any parameter

$$\theta \leftarrow \theta + g$$

Nesterov acceleration:

idea: gradient is at θ but applied to $\theta + \alpha g$

$$g \leftarrow \alpha g - \varepsilon \nabla_\theta \left[\frac{1}{m} \sum_{i=1}^{m} \ell(y_i, f_{\theta + \alpha g}(x_i)) \right]$$

Adagrad / RMSprop: still have to choose η_t (learning schedule) track gradient in j^{th} direction accumulate variance of grad. make the step size be inversely prop. to var.

Adam: combines Nesterov accel. with RMSprop

Deep learning pipeline:

1. Specify Nnet "architecture": # and configuration of the layer, loss
2. Automatic differentiation: make gradients in closed form
3. Backprop: evaluate gradients
4. Use them in SGD solver (Adam)

* tensor operations are ideal for GPUs