Neural Networks

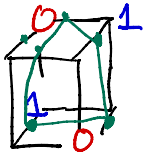
- Classification & regression
- uses ideas from perceptrons, logistic regression, ensemble learning...

Perceptrons: Can't perform

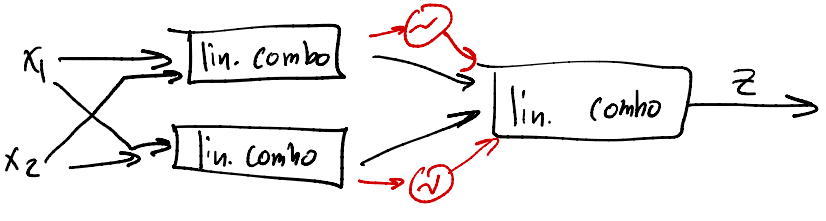
XOR	x_1	x_2
x_1	0	1
x_2	1	0

Can't separate these 4 points in 2D w/ a perceptron
↳ perceptron research halted

If you add a quadratic feature $x_1 x_2$, then XOR is linearly separable in 3D

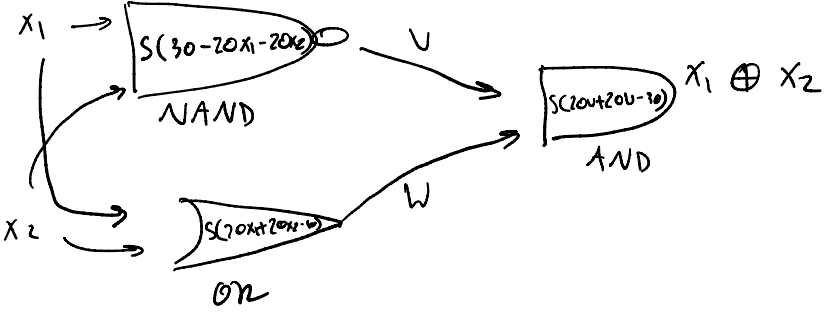


idea: use linear classifiers whose outputs are inputs to other lin. classifiers



$z > 1/2$ True
else 0

Still can't do XOR → lin. combo of lin. combos is still linear
Solution: add a nonlinearity



Neural Network w/ 1 hidden layer

typical architecture:

- Input layers: $x_1 \dots x_d$, $x_{d+1} = 1$
- hidden units: $h_1 \dots h_m$, $h_{m+1} = 1$
- output layers: $z_1 \dots z_k$

We add weights between each layer:

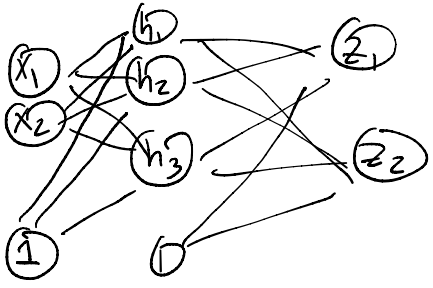
- between input & h: $m \times (d+1)$ matrix V, V_i^T is row i
- between h & output: $k \times (m+1)$ matrix W

recall $S(r) = \frac{1}{1+e^{-r}}$ activation function

$S_1(\vec{v}) = [S(v_1) \dots S(v_n) \quad 1]^T$

$\vec{h} = S_1(V\vec{x}) \rightarrow h_i = S(v_i^T x)$

$\vec{z} = S(W\vec{h}) = S(WS_1(V\vec{x}))$



- usually, NNs have more than one output

advantage of NNs: When training multiple classifiers through a single neural network, sometimes these classifiers take advantage of useful hidden units that support other classifiers.

Training Neural Networks

- SGD or BGD
- loss function $\mathcal{L}(\vec{z}, \vec{y})$ most common: Squared error
- cost function $J(h) = \frac{1}{n} \sum \mathcal{L}(h(\vec{x}_i), \vec{y}_i)$

goal: find W & V that minimize J

Usually, cost fn. isn't convex at all → many local minima

issue: NNs have symmetry — one hidden unit is indistinguishable from another
- ↳ gradient descent can't break the symmetry
  - ↳ might be in situation in which all weights of h_1 & h_2 are same

solution: random initialization of weights

Batch gradient descent:

$W \leftarrow$ random vector

repeat

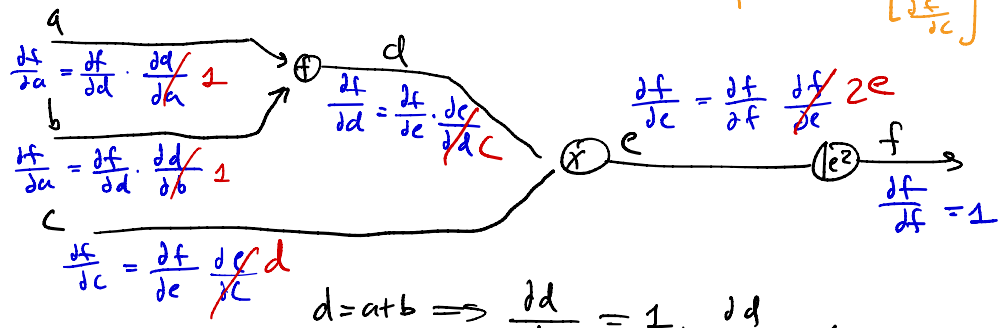
$W \leftarrow W - \varepsilon \nabla J(W)$ ← SGD

$\varepsilon \nabla L(z_i, y_i, W)$

Make sure random initialization isn't too big since $s'(\cdot) \to 0$ if $w_i >>> 0$

Computing Gradients for Arithmetic Expressions

goal: compute $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial a} \\ \frac{\partial f}{\partial b} \\ \frac{\partial f}{\partial c} \end{bmatrix}$



$d = a + b \Rightarrow \frac{\partial d}{\partial a} = 1, \frac{\partial d}{\partial b} = 1$

What if a unit's output goes to more then 1 unit?

Consider:

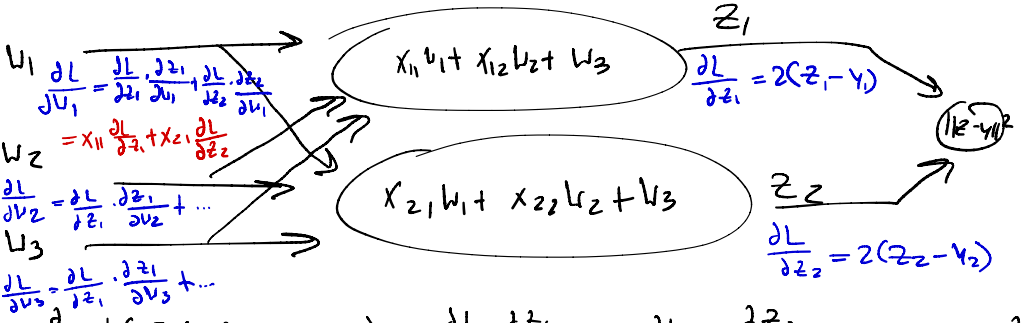


Figure 17.6: Gradient of an expression from least squares linear regression. Backpropagation arrows are omitted here, as they'd make the diagram quite messy.

The Backprop Algo.

v_i^T is row i of weight matrix V, same for U

recall $S'(x) = S(x)(1-S(x))$

$h_i = S(v_i^T \vec{x}) \rightarrow \nabla_{v_i} h_i = S'(\vec{v_i}^T \vec{x})\vec{x} = h_i(1-h_i)\vec{x}$

$z_j = S(u_j^T \vec{h}) \qquad = z_j(1-z_j)\vec{h}$