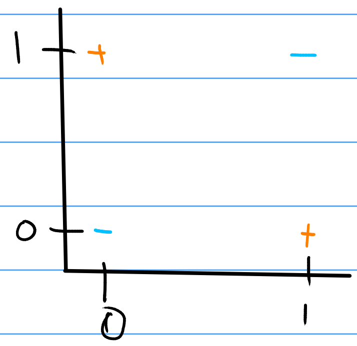
Multilayer Perceptron

XOR Problem

$\hat{y} = \langle w, x \rangle + b$



$x_1=(0,0),\ y_1=-1.\ \ b<0$

$x_2=(0,1),\ y_2=1 \Rightarrow w_2+b>0$

$x_3=(1,0),\ y_3=1 \Rightarrow w_1+b>0$

$x_4=(1,1),\ y_4=-1. \Rightarrow w_1+w_2+b<0$

$\Rightarrow w_1+w_2+2b<0 \qquad \Rightarrow w_1+w_2+2b>0.$

How to learn XOR?

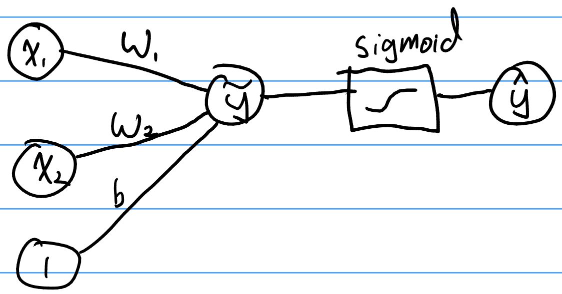
Kernels

- Apply mapping to data, use linear model on top
  representation
- Generic kernel. "Hand-crafted features"
- Neural Network: learn representation of data, from data.

Then simple linear model.

Drawings

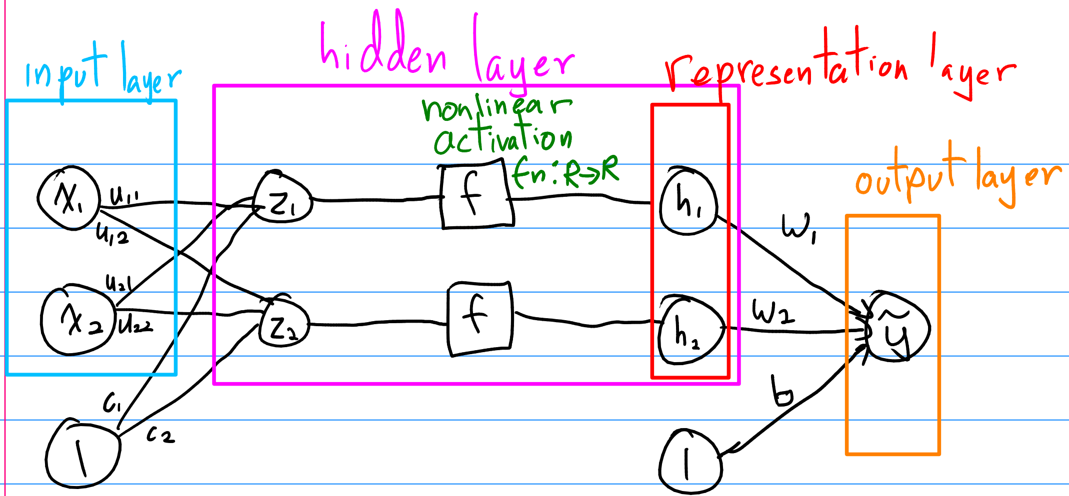
$x \in \mathbb{R}^2$. $\quad \tilde{y} = x_1 w_1 + x_2 w_2 + b = \langle x, w \rangle + b$



$\text{Sigmoid}(t) = \sigma(t) = \dfrac{1}{1+e^{-t}}$

$\hat{y} = \dfrac{1}{1+\exp(-\langle w, x \rangle - b)}$

↳ logistic regression



$$z = Ux + c$$
$$h = f(z)$$
$$\tilde{y} = \langle h, w\rangle + b$$

$U = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $w = \begin{bmatrix} 2 \\ -4 \end{bmatrix}$, $b = -1$

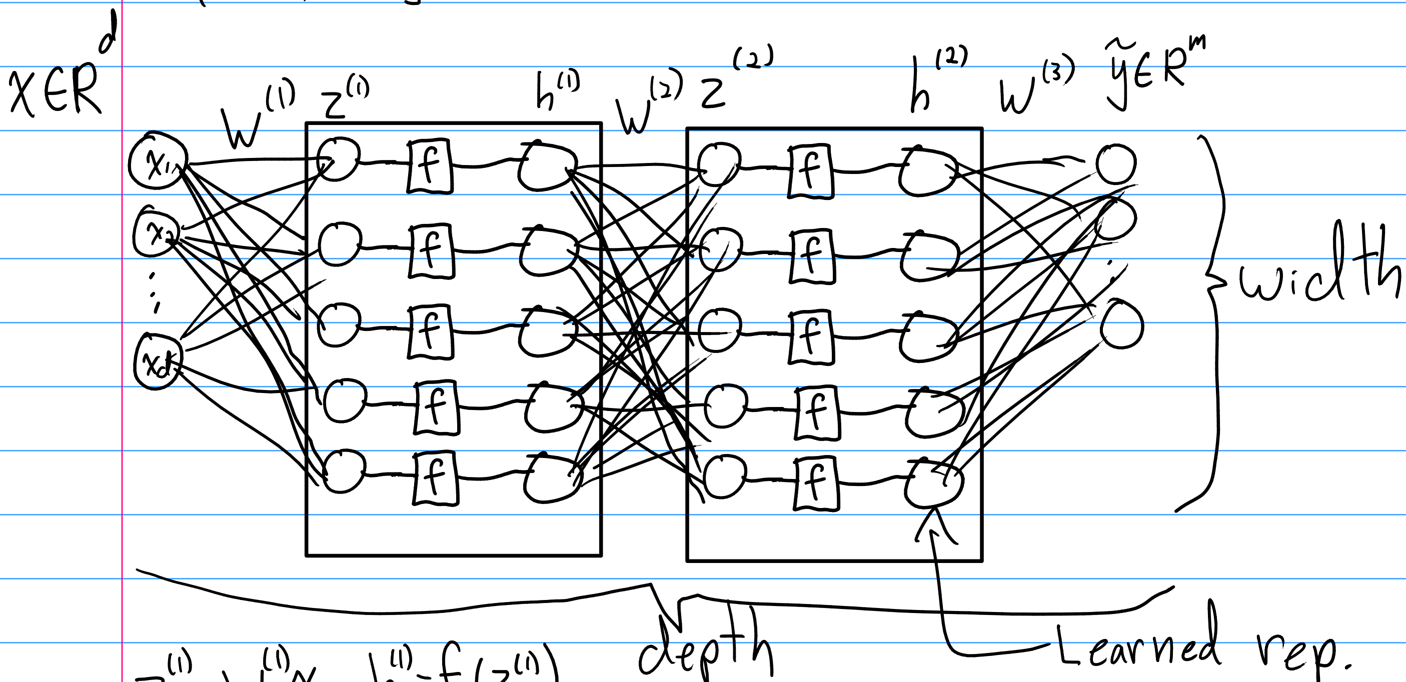
$f(t) = \max(0, t)$ (ReLU)

$x_1 = (0,0)^T$, $y_1 = -1$. $z = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$. $h = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. $\tilde{y} = -1$

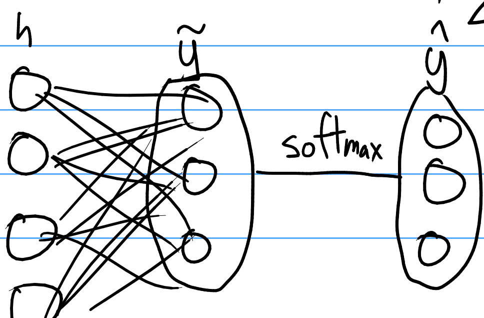
$x_2 = (0,1)^T$, $y_2 = 1$. $z = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. $h = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\tilde{y} = 2 - 1 = 1$.

$x_3 = (1,0)^T$, $y_3 = 1$

x_4 $(1,1)^T$ $y_4 = 1$



$z^{(1)} = W^{(1)}x$, $h^{(1)} = f(z^{(1)})$, $z^{2} = W^{(2)}h^{(1)}, \ldots$



$$\hat{y}_i = \frac{\exp(\tilde{y}_i)}{\sum_{i=1}^{m} \exp(\tilde{y}_i)}$$

$\hat{y} = g_\theta(x)$

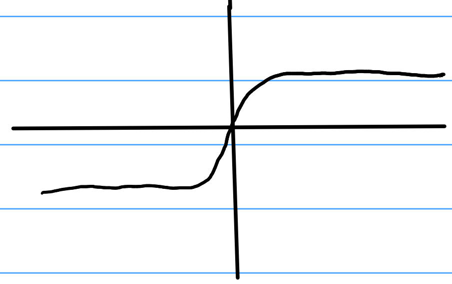
$$\ell_\theta(x, y) = -\sum_{i=1}^{m} y_i \log \hat{y}_i$$

one-hot encoding of y

Activation functions

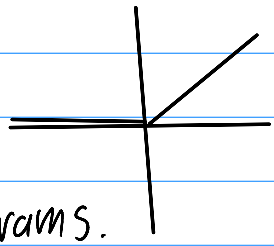
- Non-linear

Sigmoid: $\sigma(t) = \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t}$



Tanh: $\tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$

ReLU: $\mathrm{relu}(t) = \max(0, t)$



Training?

Have to determine params.

Loss fn: $\underset{\theta}{\operatorname{argmin}}\ L = \frac{1}{n}\sum_{i=1}^{n} \ell_\theta(x_i, y_i)$

Params

Gradient descent. $\theta^t = \theta^{t-1} - \eta_t \nabla L_{\theta^{t-1}}$

Auto. diff. Backpropagation

↳ Chain rule + dynamic programming

$\nabla_\theta L$

Simple case: $x \in \mathbb{R},\ f, g : \mathbb{R} \to \mathbb{R}$

$y = g(x),\ z = f(y) = f(g(x))$.

$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$ (chain rule)

$x \xrightarrow{u} z - \boxed{f} - h \xrightarrow{w} y - L$

Calculate:

$\frac{dL}{dy}$ $\quad \frac{dL}{dw} = \frac{dL}{dy} \cdot \frac{dy}{dw}$

$\frac{dy}{dw}$ $\quad \frac{dL}{du} = \frac{dL}{dy} \cdot \frac{dy}{dh} \cdot \frac{dh}{dz} \cdot \frac{dz}{du}$

$\frac{dy}{dh}$

$\frac{dh}{dz}$

$\frac{dz}{du}$

$x \in R^m, y \in R^n, z \in R.\quad g: \mathbb{R}^m \to R^n, f: \mathbb{R}^n \to \mathbb{R}$

$z = f(y) = f(g(x))$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \Rightarrow \nabla_x z = \underbrace{\left(\frac{\partial y}{\partial z}\right)^T}_{\substack{\text{Jacobian matrix} \\ n\times m \text{ matrix}}} \nabla_y z$$

Universal.
Any continuous fn $g: [0,1]^d \to \mathbb{R}$
can be approx arbitrarily well by some
2 layer NN, with arbitrary non-polynomial activation