

EE269

Signal Processing for Machine Learning

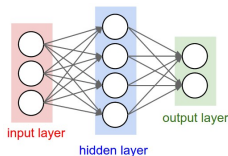
Lecture 15

Instructor : Mert Pilanci

Stanford University

March 10, 2019

Neural Network with a single hidden layer



- ▶ input variables x_1, \dots, x_d
- ▶ hidden variables z_1, \dots, z_m
- ▶ output variables y_1, \dots, y_K

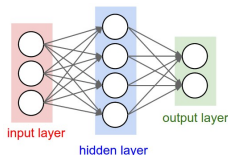
The output predicted by the network given input x is

$$f_1(x), f_2(x), \dots, f_K(x)$$

$$y_k = f_k(x) = g_k(\beta_k^T z) \quad \forall k \text{ and } z_m = \sigma(\alpha_m^T x)$$

- ▶ $\sigma(\cdot)$ and g_1, \dots, g_K are nonlinear functions

Neural Network with a single hidden layer



The output predicted by the network given input x is

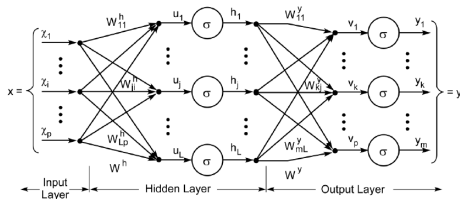
$$f_1(x), f_2(x), \dots, f_k(x)$$

$$y_k = f_k(x) = g_k(\beta_k^T z) \quad \forall k \text{ and } z_m = \sigma(\alpha_m^T x)$$

- ▶ $\sigma(\cdot)$ and g_1, \dots, g_K are nonlinear functions
- ▶ Regression (continuous output): $g_t(u) = u$ and $f(x) = \beta^T z$
- ▶ Multi-class classification $g_k(u_1, \dots, u_K) = \frac{e^{u_k}}{\sum_{j=1}^K e^{u_j}}$

One-hot encoding: x is class k $y = e_k$ (kth ordinary basis)

Multilayer Neural Networks

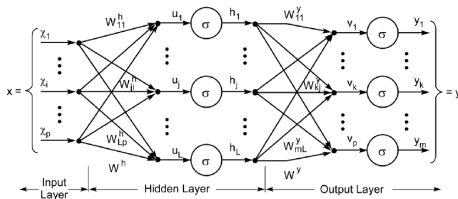


$$z^{(0)} = x \quad (\text{input})$$

$$a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \quad l = 1, \dots, L$$

$$z_j^{(l)} = \sigma(a_k^l) \quad l = 1, \dots, L$$

Multilayer Neural Networks



$$z^{(0)} = x \quad (\text{input})$$

$$a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \quad l = 1, \dots, L$$

$$z_j^{(l)} = \sigma(a_k^l) \quad l = 1, \dots, L$$

- Training: Parameters $\Theta = (W^{(1)}, W^{(2)}, \dots, W^{(L)})$
 regression: (squared loss) vs classification (cross-entropy loss)

$$\min_{\Theta} \sum_{n=1}^N (y_n - f(x_n))^2 \quad \min_{\Theta} - \sum_{n=1}^N \sum_{k=1}^K y_{nk} \log f_k(x_n)$$

Training Multilayer Neural Networks

- ▶ Training: Parameters $\Theta = (W^{(1)}, W^{(2)}, \dots, W^{(L)})$
regression: (squared loss) vs classification (cross-entropy loss)

$$\min_{\Theta} \sum_{n=1}^N \underbrace{(y_n - f(x_n))^2}_{R_n(\Theta)} \qquad \min_{\Theta} - \sum_{n=1}^N \underbrace{\sum_{k=1}^K y_{nk} \log f_k(x_n)}_{R_n(\Theta)}$$

- ▶ Gradient Descent
$$\Theta_{t+1} = \Theta_t - \sum_{i=1}^n \frac{\partial}{\partial \Theta} R_n(\Theta)$$
- ▶ Stochastic Gradient Descent
$$\Theta_{t+1} = \Theta_t - \frac{\partial}{\partial \Theta} R_{n_t}(\Theta)$$
where n_t is a random index
- ▶ non-convex optimization problem

Computing derivatives: Backpropagation Algorithm

$$z^{(0)} = x \quad (\text{input})$$

$$a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \quad l = 1, \dots, L$$

$$z_j^{(l)} = \sigma(a_k^l) \quad l = 1, \dots, L$$

$$\min_{\Theta} \sum_{n=1}^N \underbrace{(y_n - f(x_n))^2}_{R_n(\Theta)}$$

$$\frac{\partial R_n(\Theta)}{\partial W_{ij}^{(l)}} = \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial W_{ij}^{(l)}} = \delta_{nj}^{(l)} z_i^{(l-1)}$$

where we defined

$$\delta_{nj}^{(l)} \triangleq \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}}$$

Computing derivatives: Backpropagation Algorithm

$$z^{(0)} = x \quad (\text{input})$$


$$a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \quad l = 1, \dots, L$$

$$z_j^{(l)} = \sigma(a_k^l) \quad l = 1, \dots, L$$

$$\begin{aligned} \delta_{nj}^{(l)} &\triangleq \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} = \sum_k \frac{\partial R_n(\Theta)}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial a_j^{(l)}} \\ &= \sum_k \delta_{nk}^{(l+1)} W_{jk}^{(l+1)} \sigma'(a_j^{(l)}) \end{aligned}$$

last term follows from:

$$a_k^{(l+1)} = \sum_r W_{rk}^{(l+1)} z_r^{(l)} = \sum_r W_{rk}^{(l+1)} \sigma(a_r^{(l)})$$

► output layer: (e.g. $R_n(\Theta) = \|a^{(L)} - y_n\|^2$) $\delta_{nj}^{(L)} = 2(a_j^{(L)} - y_n)$ 

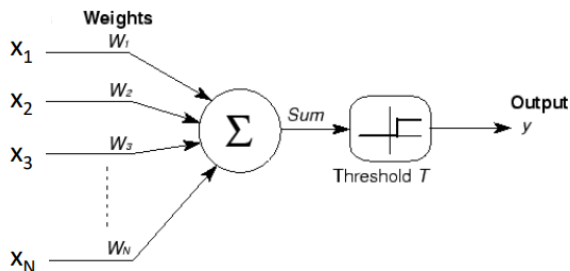
Backpropagation visually explained

$$\delta_{nj}^{(l)} \triangleq \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} = \sum_k \delta_{nk}^{(l+1)} W_{jk}^{(l+1)} \sigma'(a_j^{(l)})$$
$$\delta_{nj}^{(L)} = 2(a^{(L)} - y_n)$$

Backpropagation

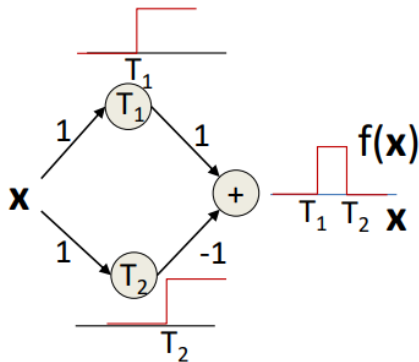
Backpropagation

Neural Networks: Expressive Power

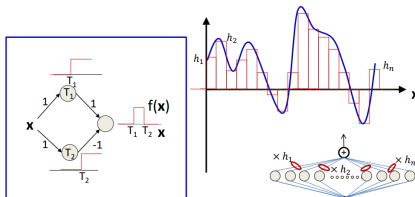


$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^d w_i x_i \geq T \\ 0 & \text{else} \end{cases}$$

Neural Networks: Expressive Power

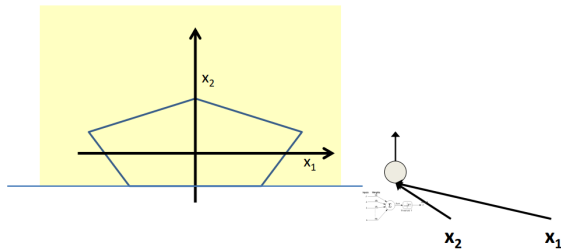


Neural Networks: Expressive Power

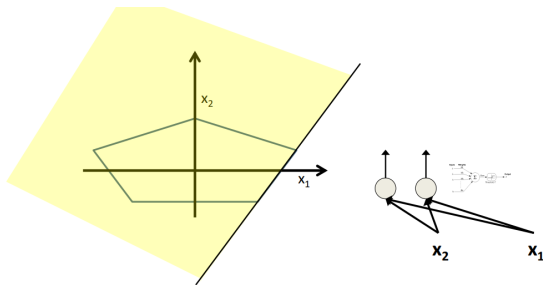


slide credit: Bhiksha Raj (<http://lxmls.it.pt/2018/Lecture.fin.pdf>)

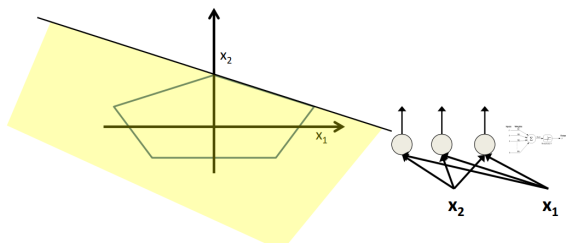
Neural Networks



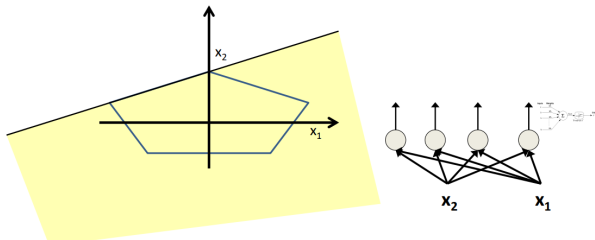
Neural Networks



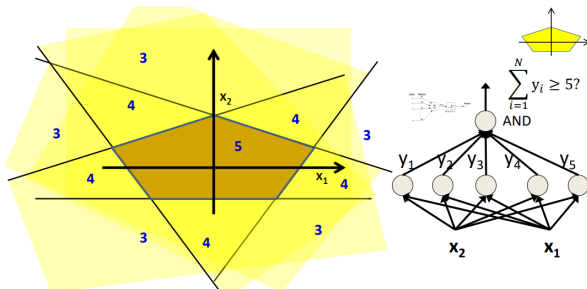
Neural Networks



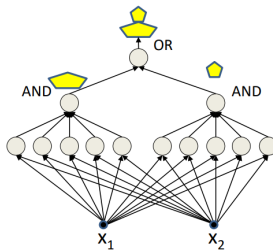
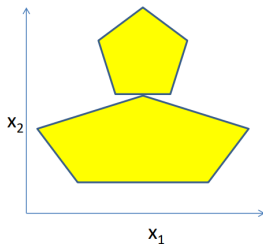
Neural Networks



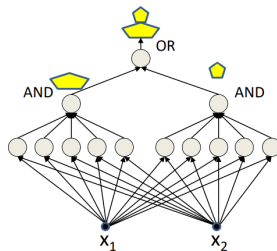
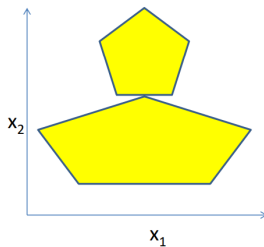
Neural Networks



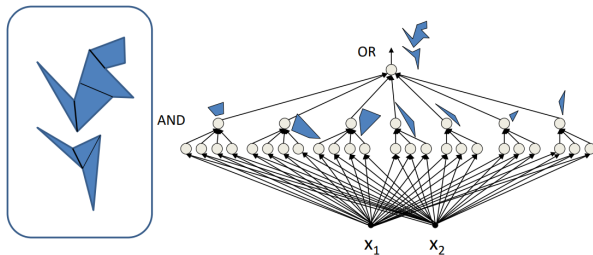
Neural Networks



Neural Networks

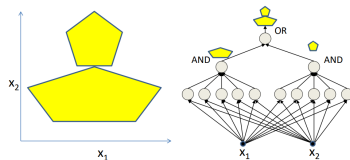


Neural Networks

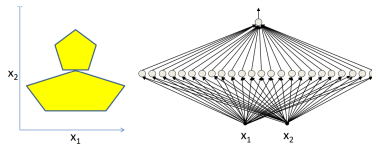


slide credit: Bhiksha Raj (<http://lxmls.it.pt/2018/Lecture.fin.pdf>)

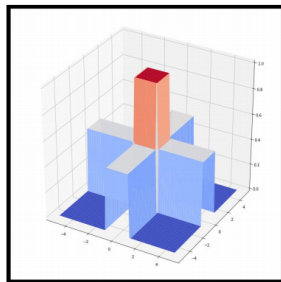
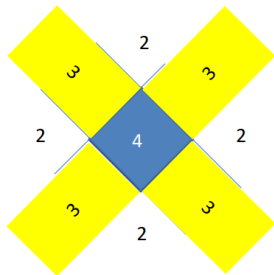
Neural Networks: Width vs Depth



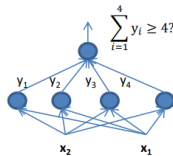
- Is it possible to obtain this decision region in 2 layers instead of 3 ?



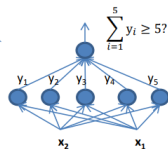
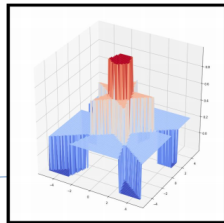
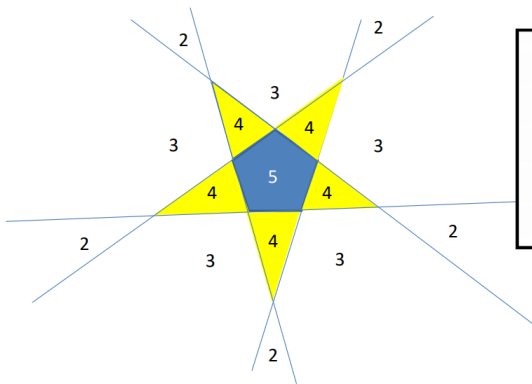
Neural Networks: Two layer



- The polygon net

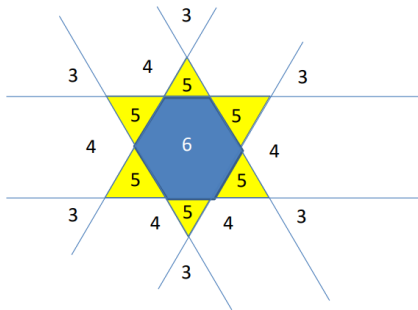


Neural Networks: Two layer

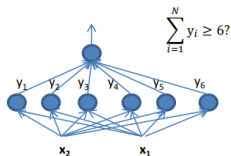
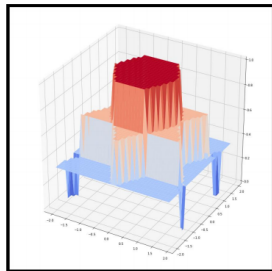


- The polygon net

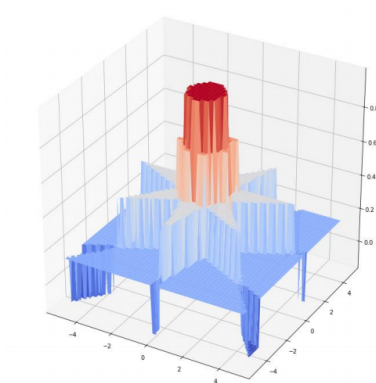
Neural Networks: Two layer



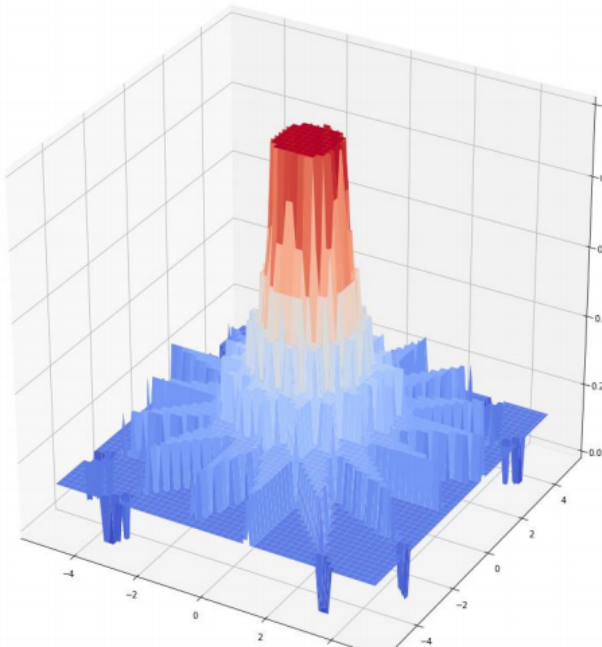
- The polygon net



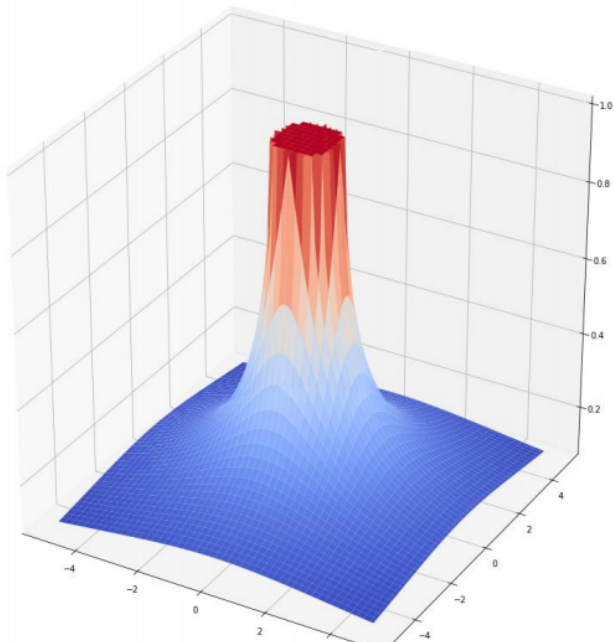
Neural Networks: Two layer



Neural Networks: Two layer

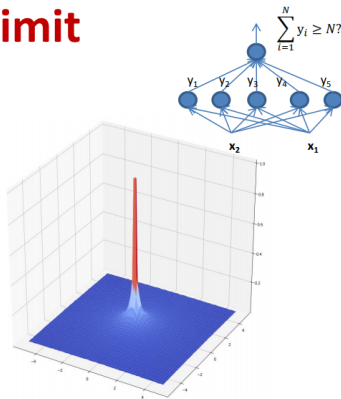
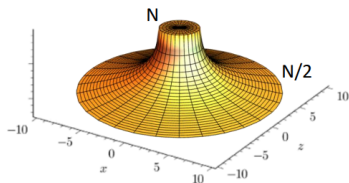


Neural Networks: Two layer



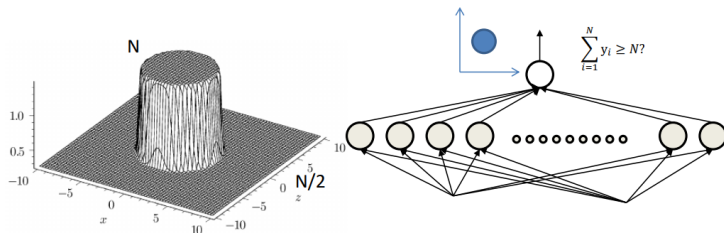
Neural Networks: Two layer

In the limit



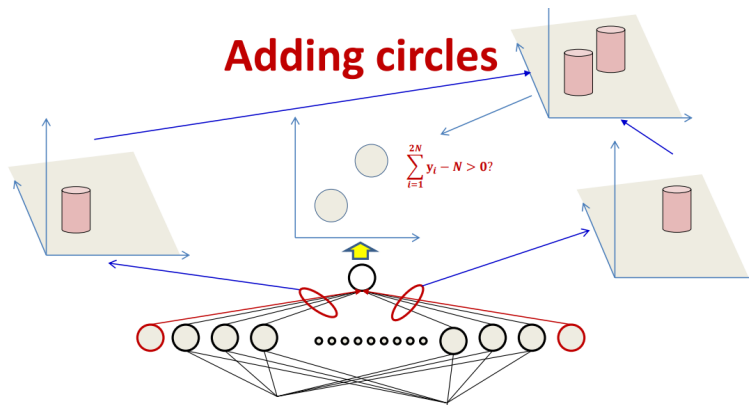
- $\sum_i y_i = N \left(1 - \frac{1}{\pi} \arccos \left(\min \left(1, \frac{\text{radius}}{|\mathbf{x} - \text{center}|} \right) \right) \right)$
- For small radius, it's a near perfect cylinder
 - N in the cylinder, N/2 outside

Neural Networks: Two layer

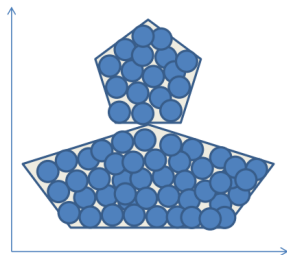
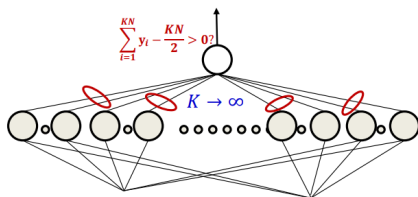


slide credit: Bhiksha Raj (<http://lxmls.it.pt/2018/Lecture.fin.pdf>)

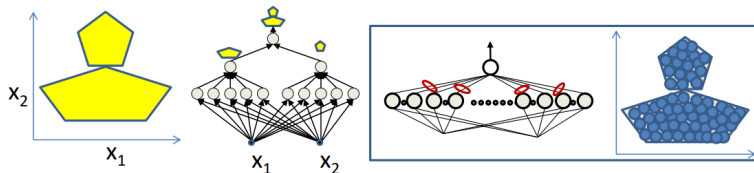
Neural Networks: Two layer



Neural Networks: Two layer

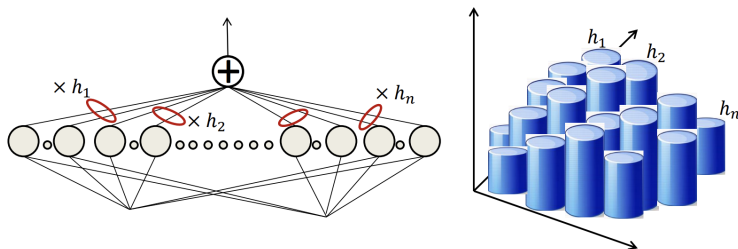


Neural Networks: Two layer



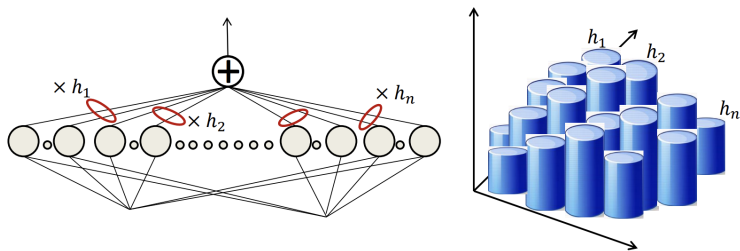
slide credit: Bhiksha Raj (<http://lxmls.it.pt/2018/Lecture.fin.pdf>)

Two layer networks are universal approximators



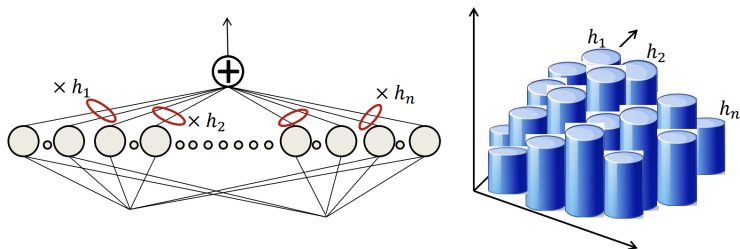
- ▶ Two layer networks (also called one hidden layer) can compose arbitrary functions to arbitrary precision

Two layer networks are universal approximators



- ▶ Two layer networks (also called one hidden layer) can compose arbitrary functions to arbitrary precision
- ▶ May require infinitely many neurons in the hidden layer

Two layer networks are universal approximators



- ▶ Two layer networks (also called one hidden layer) can compose arbitrary functions to arbitrary precision
- ▶ May require infinitely many neurons in the hidden layer
- ▶ Deeper networks require **fewer** neurons for the same approximation error

Neural Networks

