Multi-Layer vs. Single-Layer Networks

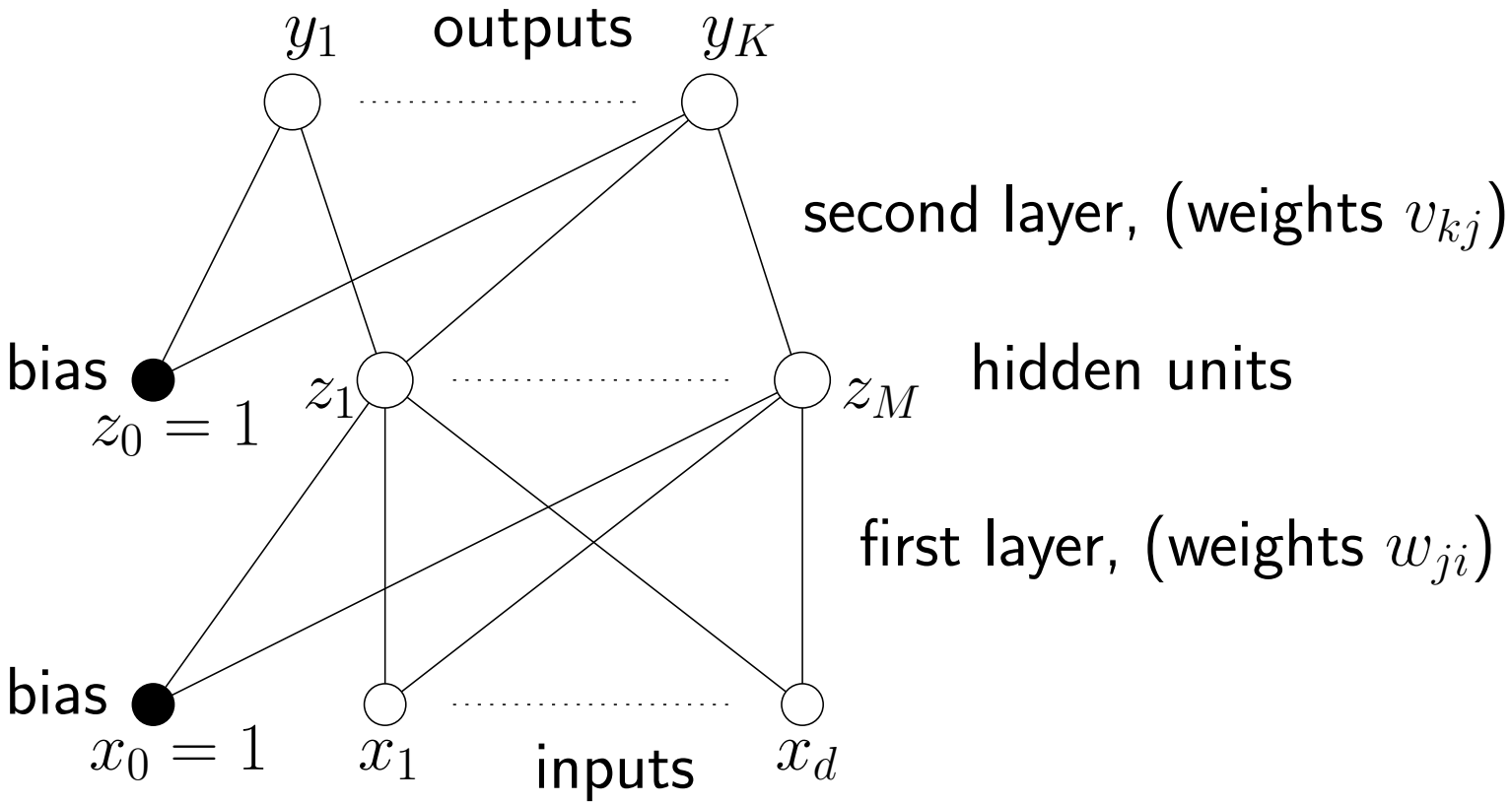
Single-layer networks

- based on a linear combination of the input variables which is transformed by linear/non-linear activation function
- are limited in terms of the range of functions they can represent

Multi-layer networks

- consist of multiple layers and are capable of approximating any continuous functional mapping
- are compared to single-layer networks not so straightforward to train

Multi-Layer Network



Connection in first layer from input unit i to hidden unit j is denoted as w_{ji}. Connection from hidden unit j to output unit k is denoted as v_{kj}.

Multi-Layer Network (cont.)

Hidden unit j receives input

$$a_j = \sum_{i=1}^{d} w_{ji} x_i + w_{j0} = \sum_{i=0}^{d} w_{ji} x_i$$

and produces output

$$z_j = g(a_j) = g\left(\sum_{i=0}^{d} w_{ji} x_i \right) .$$

Output unit k thus receives

$$a_k = \sum_{j=1}^{M} v_{kj} z_j + v_{k0} = \sum_{j=0}^{M} v_{kj} z_j$$

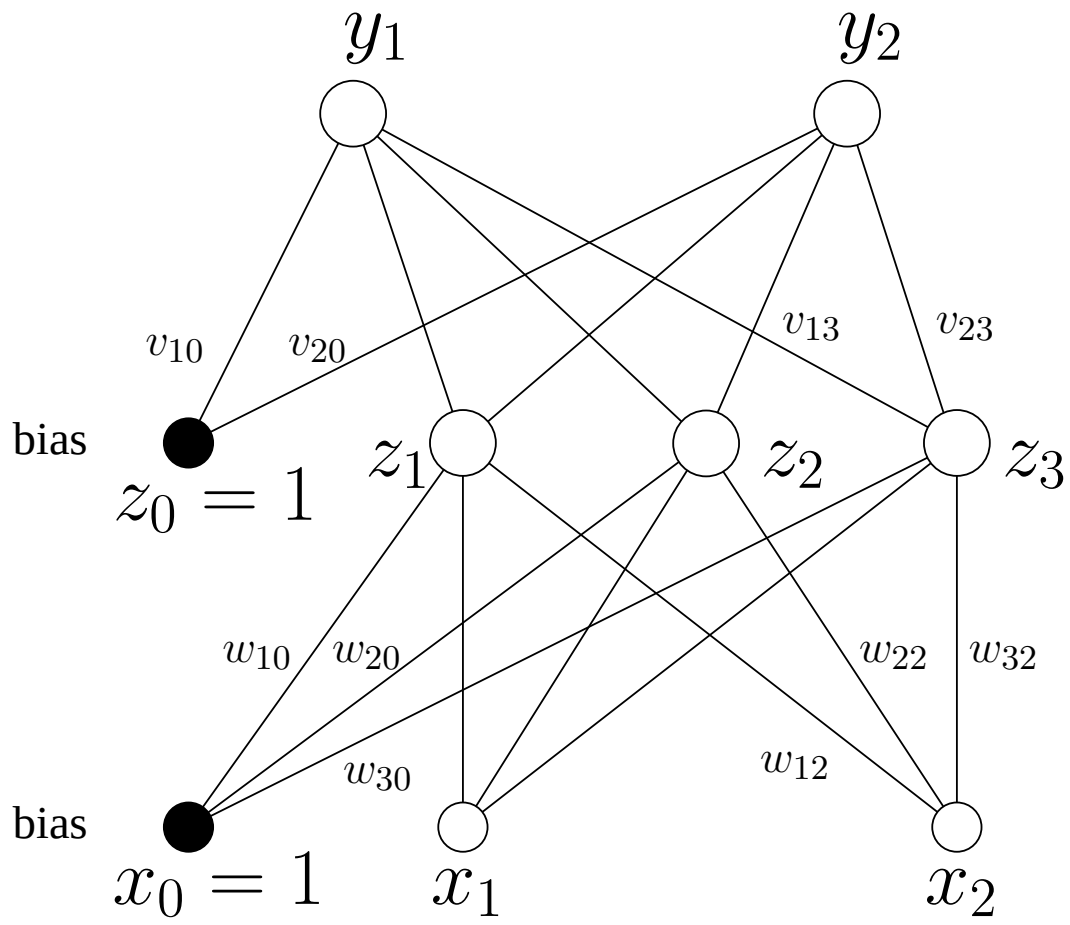
Multi-Layer Network (cont.)

and produces the final output

$$y_k = g(a_k) = g\left(\sum_{j=0}^{M} v_{kj} z_j\right) = g\left(\sum_{j=0}^{M} v_{kj}\, g\left(\sum_{i=0}^{d} w_{ji} x_i\right)\right)$$

Note that the activation function $g(\cdot)$ in the first layer can be different from those in the second layer (or other layers).

Multi-Layer Networks Example



Note: sometimes the layers of units are counted (here three layers), rather the layers of adaptive weights. In this course L-layer network is referred to a network with L layers of adaptive weights.

LMS Learning Rule for Multi-Layer Networks

- We have seen that the LMS learning rule is based on the gradient descent algorithm.
- The LMS learning rule worked because the error is proportional to the square difference between actual output y and target output t and can be evaluated for each output unit.
- In a multi-layer network we can use LMS learning rule on the hidden-to-output layer weights because target outputs are known.

Problem: we cannot compute the target outputs of the input-to-hidden weights because these values are unknown, or, to put it the other way around, how to update the weights in the first layer?

Backpropagation (Hidden-to-Output Layer)

Recall that we want to minimize the error on training patterns between actual output y_k and target output t_k:

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2.$$

Backpropagation learning rule is based on gradient descent:

$$\Delta\mathbf{w} = -\eta\frac{\partial E}{\partial \mathbf{w}}, \text{ component form } \Delta w_{st} = -\eta\frac{\partial E}{\partial w_{st}}$$

Apply chain rule for differentiation:

$$\frac{\partial E}{\partial v_{kj}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial v_{kj}}$$

Backprop. (Hidden-to-Output Layer) (cont.)

Gradient descent rule gives:

$$\begin{aligned} \Delta v_{kj} &= -\eta \frac{\partial E}{\partial v_{kj}} = -\eta (y_k - t_k) g'(a_k) z_j \\ &= -\eta \delta_k z_j \end{aligned}$$

where

$$\delta_k = (y_k - t_k) g'(a_k).$$

Observe that this result is identical to that obtained for LMS.

Backpropagation (Input-to-Hidden Layer)

For the input-to-hidden connection we must differentiate with respect to the w_{ji}'s which are deeply embedded in

$$E = \frac{1}{2}\sum_{k=1}^{K}\left[g\left(\sum_{j=0}^{M} v_{kj}\, g\left(\sum_{i=0}^{d} w_{ji}x_i\right)\right) - t_k\right]^2$$

Apply chain rule:

$$\begin{aligned}
\Delta w_{ji} &= -\eta\frac{\partial E}{\partial w_{ji}} = -\eta\frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \\
&= -\eta\sum_{k=1}^{K}\underbrace{(y_k - t_k)g'(a_k)}_{\delta_k} v_{kj}g'(a_j)x_i \\
&= -\eta\sum_{k=1}^{K}\delta_k v_{kj}g'(a_j)x_i
\end{aligned}$$

Backprop. (Input-to-Hidden Layer) (cont.)

$$\Delta w_{ji} = -\eta \delta_j x_i$$

where

$$\delta_j = g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k$$

Observe: that we need to propagate the errors (δ's) backwards to update the weights $\mathbf{v}$ and $\mathbf{w}$

$$\begin{aligned} \Delta v_{kj} &= -\eta \delta_k z_j \\ \delta_k &= (y_k - t_k) g'(a_k) \\ \Delta w_{ji} &= -\eta \delta_j x_i \\ \delta_j &= g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k \end{aligned}$$

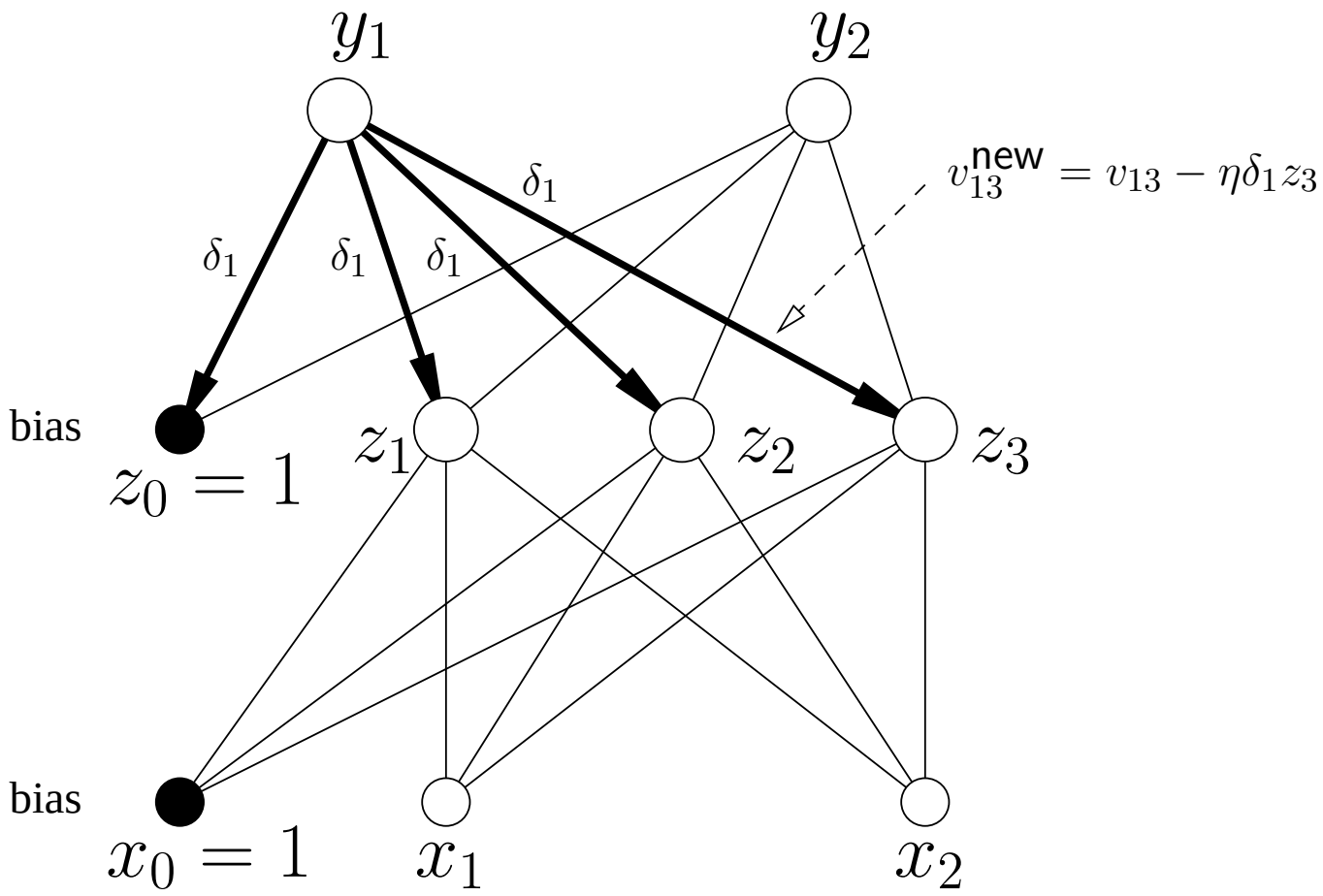
Error Backpropagation

- Apply input $\mathbf{x}$ and forward propagate through the network using $a_j = \sum_{i=0}^{d} w_{ji} x_i$ and $z_j = g(a_j)$ to find the activations of all the hidden and output units.
- Compute the deltas δ_k for all the output units using $\delta_k = (y_k - t_k) g'(a_k)$.
- Backpropagate the δ's using $\delta_j = g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k$ to obtain δ_j for each hidden unit in the network.

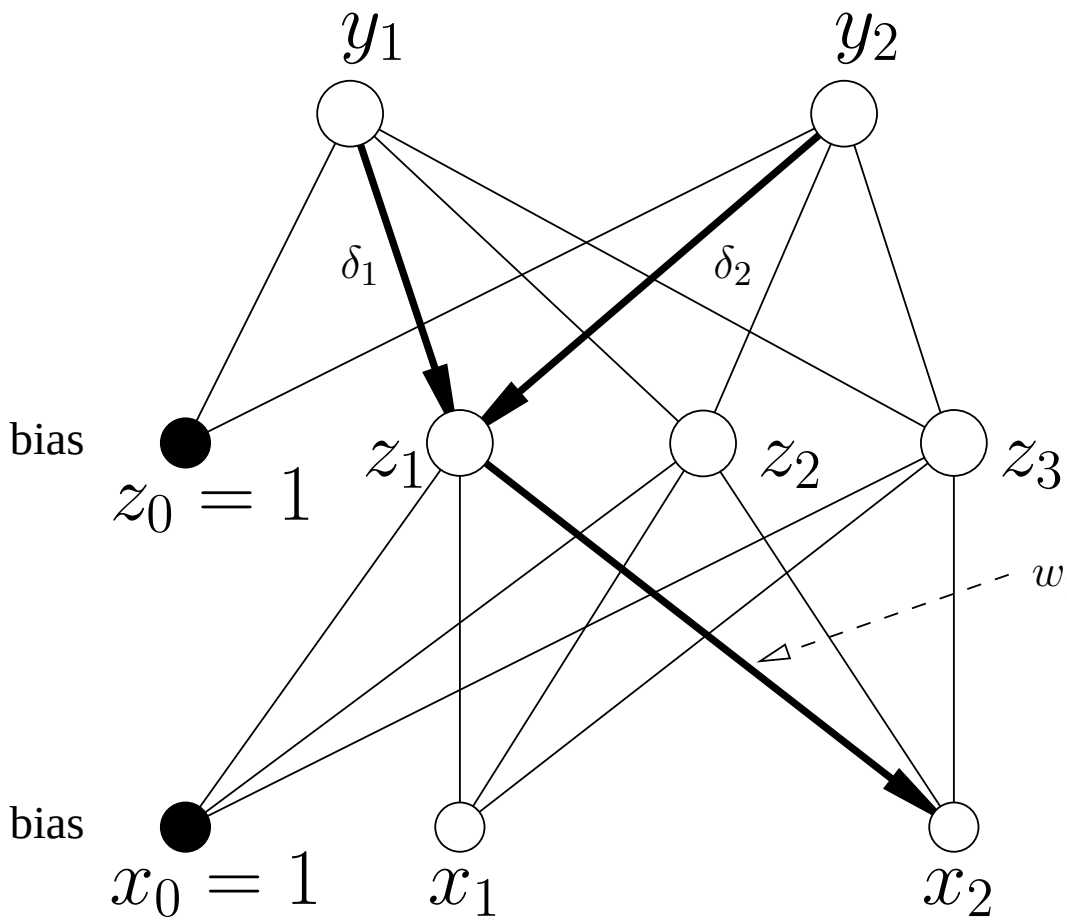
Time and space complexity:

d input units, M hidden units and K output units results in $M(d+1)$ weights in first layer and $K(M+1)$ weights in second layer. Space and time complexity is $\mathcal{O}(M\,(K+d))$. If e training epochs are performed, then time complexity is $\mathcal{O}(e\,M\,(K+d))$.

Backprop. (Output-to-Hidden Layer) Vis.



Backprop. (Hidden-to-Input Layer) Vis.



$$w_{12}^{\text{new}} = w_{12} - \eta \underbrace{[g'(a_1)(v_{11}\delta_1 + v_{21}\delta_2)]}_{\delta_j} x_2$$

Property of Activation Functions

- In the Backpropagation algorithm the derivative of $g(a)$ is required to evaluate the δ's.
- Activation functions

$$g_1(a) = \frac{1}{1+\exp(-\beta a)} \quad \text{and} \quad g_2(a) = \tanh(\beta a)$$

obey the property

$$\begin{aligned} g_1'(a) &= \beta\, g_1(a)(1-g_1(a)) \\ g_2'(a) &= \beta(1-(g_2(a))^2) \end{aligned}$$

Online Backpropagation Algorithm

input : $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^d \times \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K\}, \eta \in \mathbb{R}_+$, max.epoch $\in \mathbb{N}, \epsilon \in \mathbb{R}_+$

output: $\mathbf{w}, \mathbf{v}$

begin

  Randomly initialize $\mathbf{w}, \mathbf{v}$

  epoch $\leftarrow 0$

  repeat

    for $n \leftarrow 1$ **to** N **do**

      $\mathbf{x} \leftarrow$ select pattern $\mathbf{x}_n$

      $v_{kj} \leftarrow v_{kj} - \eta \delta_k z_j$

      $w_{ji} \leftarrow w_{ji} - \eta \delta_j x_i$

    epoch $\leftarrow$ epoch $+ 1$

  until $(\mathit{epoch} = \mathit{max.epoch})$ *or* $(\|\nabla E\| < \epsilon)$

  return $\mathbf{w}, \mathbf{v}$

end

Batch Backpropagation Algorithm

input : $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^d \times \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K\}, \eta \in \mathbb{R}_+$, max.epoch $\in \mathbb{N}, \epsilon \in \mathbb{R}_+$

output: $\mathbf{w}, \mathbf{v}$

begin

  Randomly initialize $\mathbf{w}, \mathbf{v}$

  epoch $\leftarrow 0$, $\Delta w_{ji} \leftarrow 0$, $\Delta v_{kj} \leftarrow 0$

  repeat

    for $n \leftarrow 1$ **to** N **do**

      $\mathbf{x} \leftarrow$ select pattern $\mathbf{x}_n$

      $\Delta v_{kj} \leftarrow \Delta v_{kj} - \eta \delta_k z_j$, $\Delta w_{ji} \leftarrow \Delta w_{ji} - \eta \delta_j x_i$

    $v_{kj} \leftarrow v_{kj} + \Delta v_{kj}$

    $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$

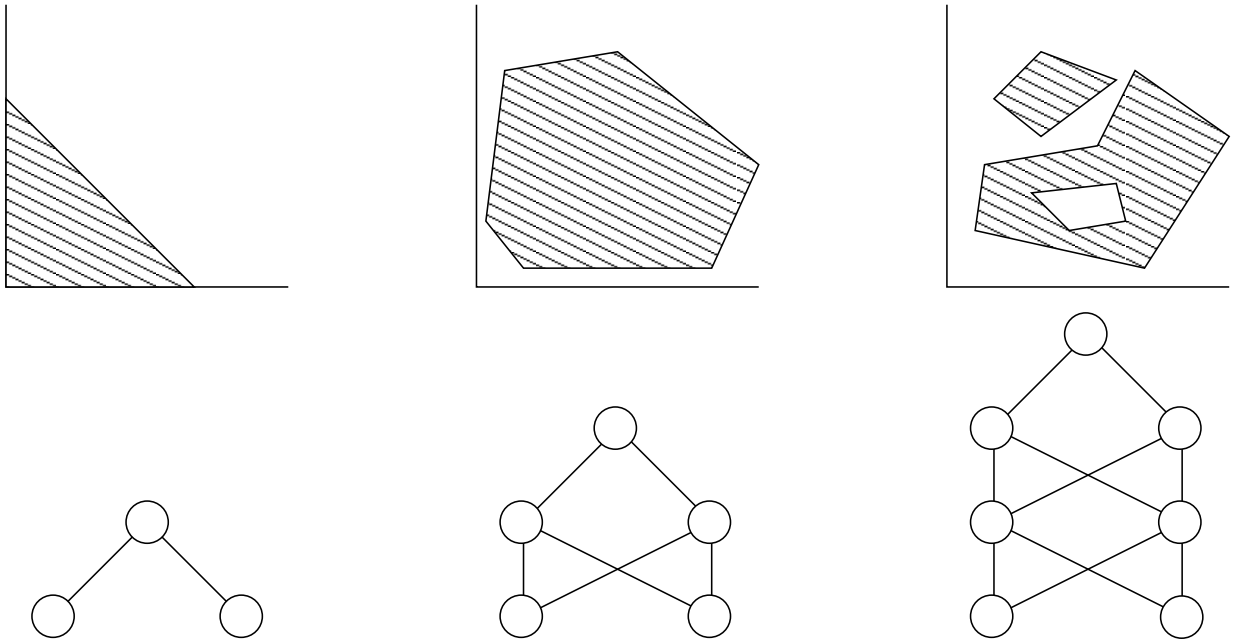
    epoch $\leftarrow$ epoch $+ 1$

  until (*epoch* $=$ *max.epoch*) *or* $(\|\nabla E\| < \epsilon)$

  return $\mathbf{w}, \mathbf{v}$

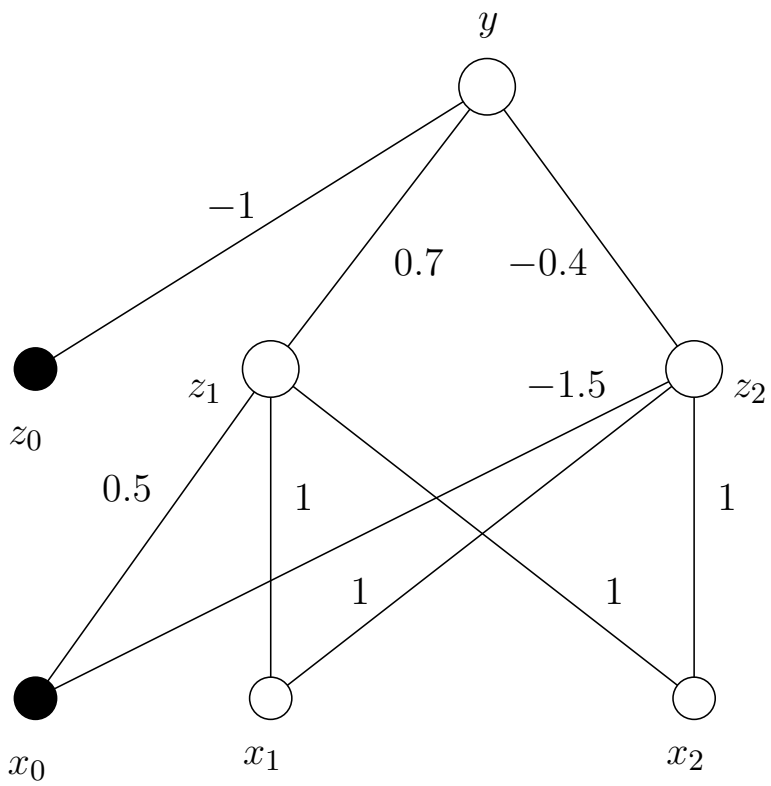
end

Multi-Layer Networks & Heaviside Step Func.



Possible decision boundaries which can be generated by networks having various numbers of layers and using Heaviside activation function.

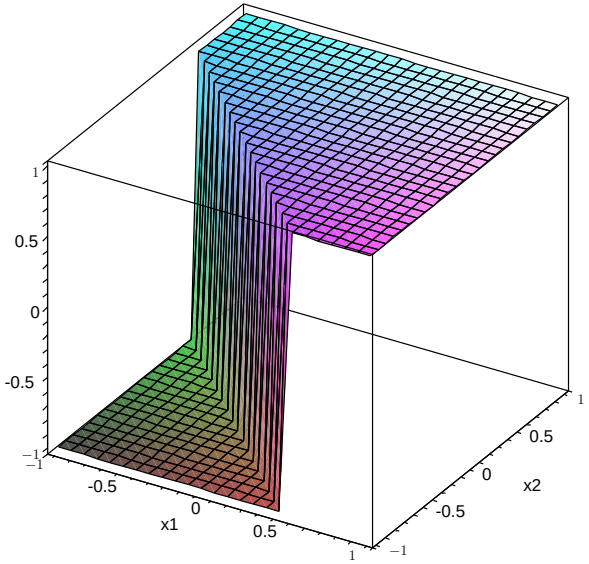
Multi-Layer NN for XOR Separability Problem

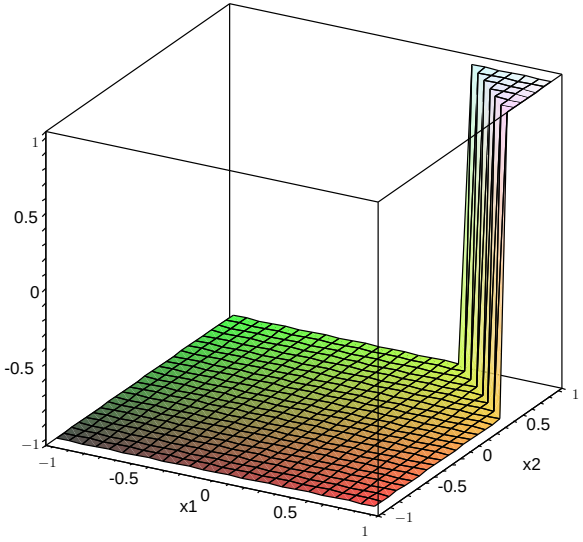


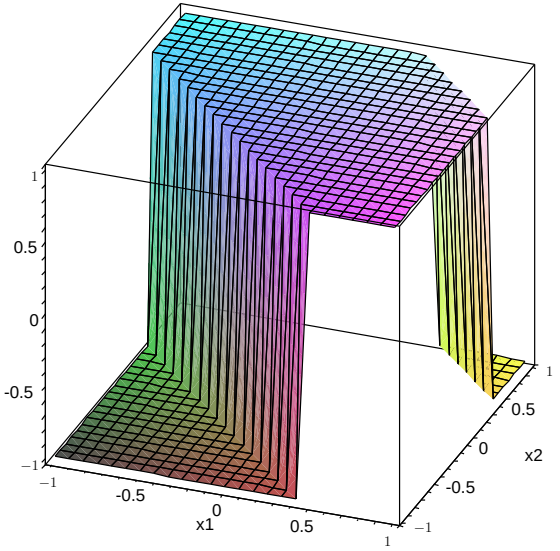
x_1	x_2	x_1 XOR x_2
-1	-1	-1
-1	$+1$	$+1$
$+1$	-1	$+1$
$+1$	$+1$	-1

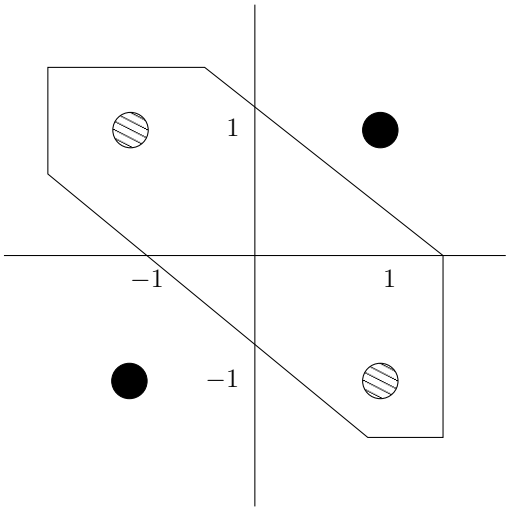
$$g(a) = \begin{cases} -1 & \text{if } a < 0 \\ +1 & \text{if } a \geq 0 \end{cases}$$

Multi-Layer NN for XOR Sep. Problem (cont.)









Expressive Power of Multi-Layer Networks

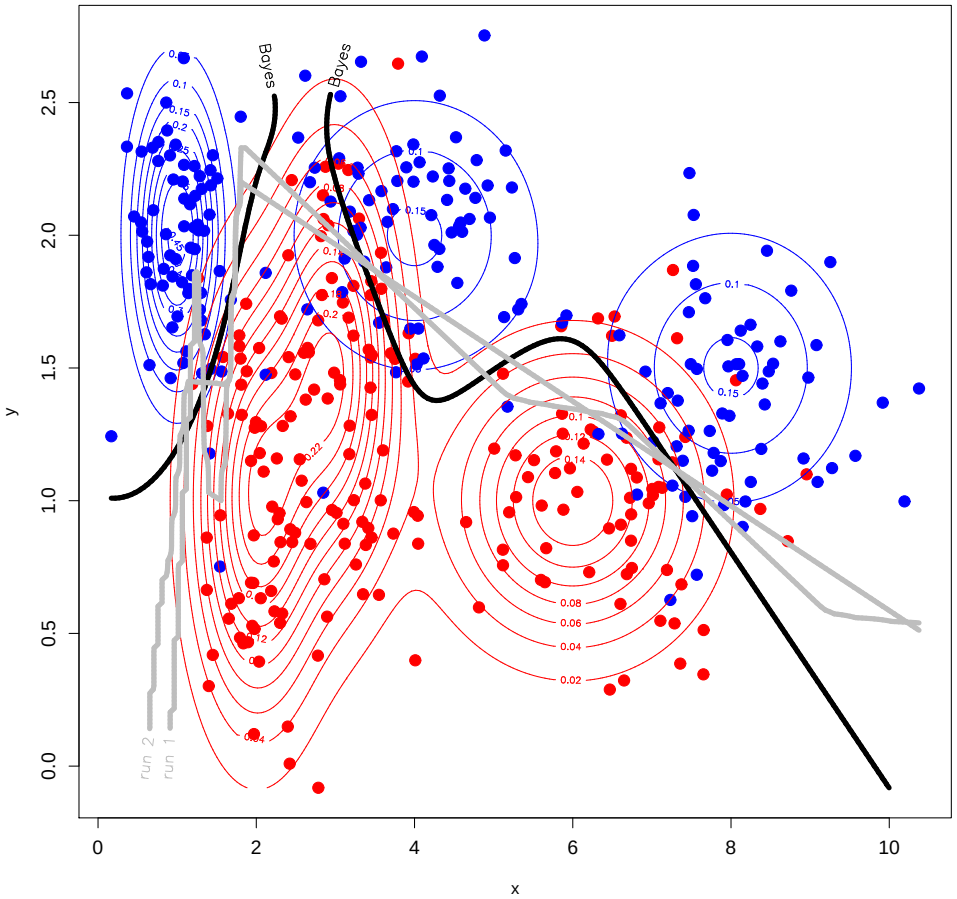
With a two-layer network and a sufficient number of hidden units, *any* type of function can be represented when given proper nonlinearities and weights.

The famous mathematician Andrey Kolmogorov proved that any continuous function $y(\mathbf{x})$ defined on the unit hypercube $[0,1]^n, n \geq 2$ can be represented in the form

$$y(\mathbf{x}) = \sum_{j=1}^{2n+1} \Xi_j \left(\sum_{i=1}^{d} \Psi_{ij}(x_i) \right)$$

for properly chosen Ξ_j and Ψ_{ij}.

Bayes Decision Region vs. Neural Network



Points from blue and red class are generated by a mixture of Gaussians. Black curve shows optimal separation in a Bayes sense. Gray curve shows neural network separation of two independent backpropagation learning runs.

Neural Network (Density) Decision Region