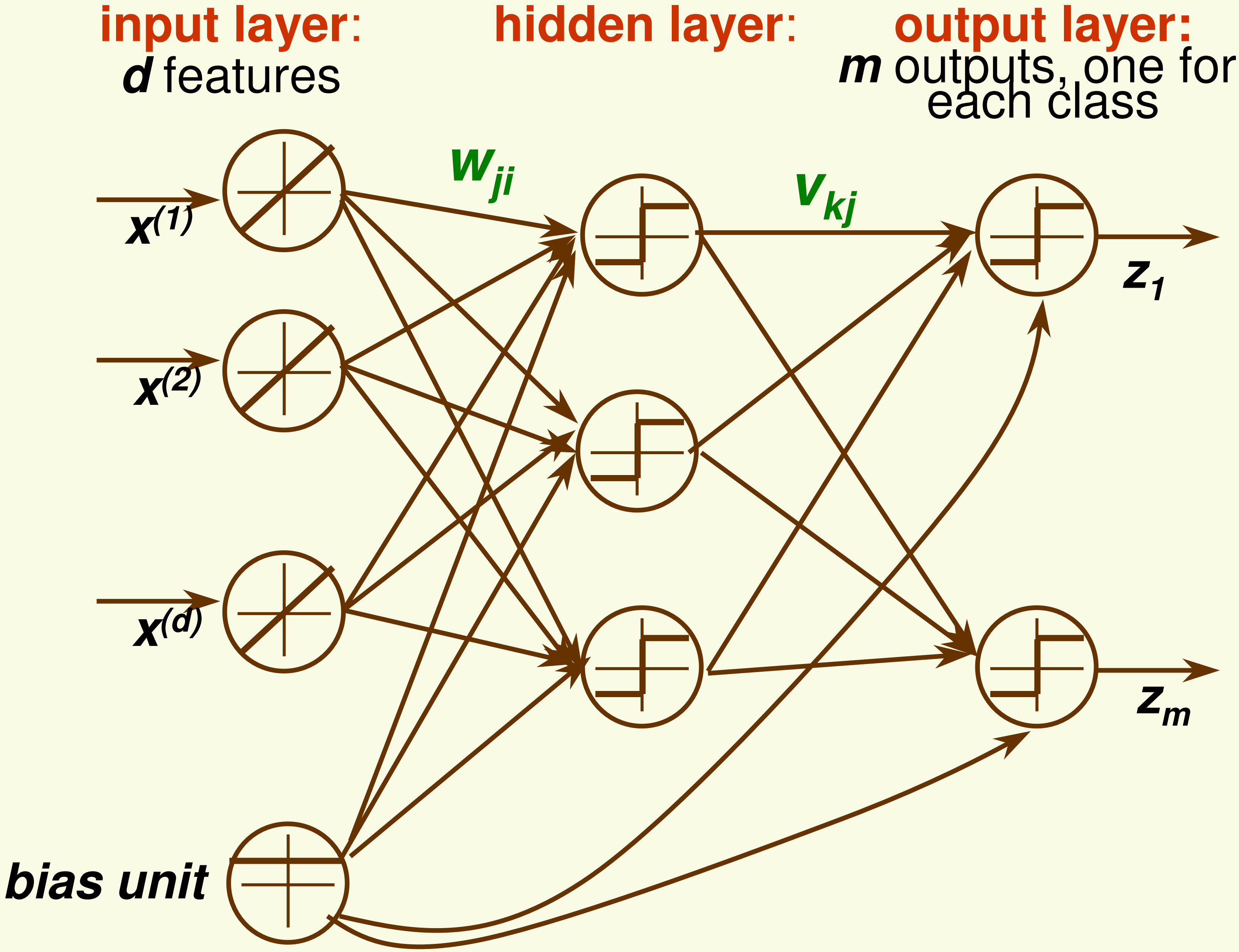
CS434a/541a: Pattern Recognition
Prof. Olga Veksler

Lecture 13

Today

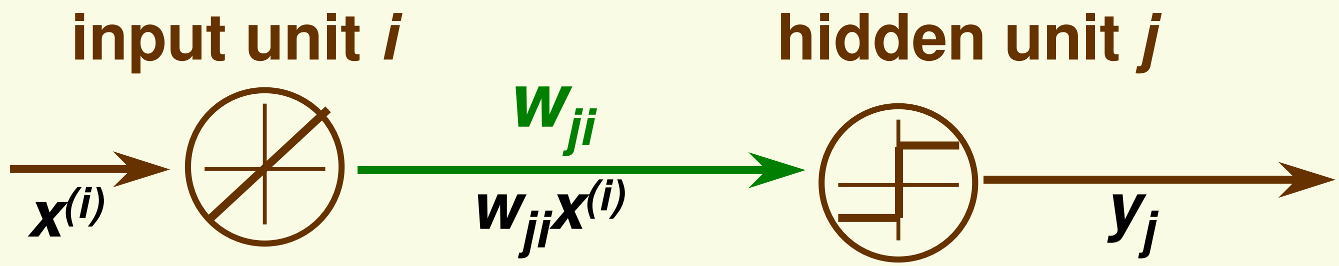
- Continue Multilayer Neural Networks (MNN)
  - Review MNN structure
  - Backpropagation
  - Training Protocols

MNN: Feed Forward Operation

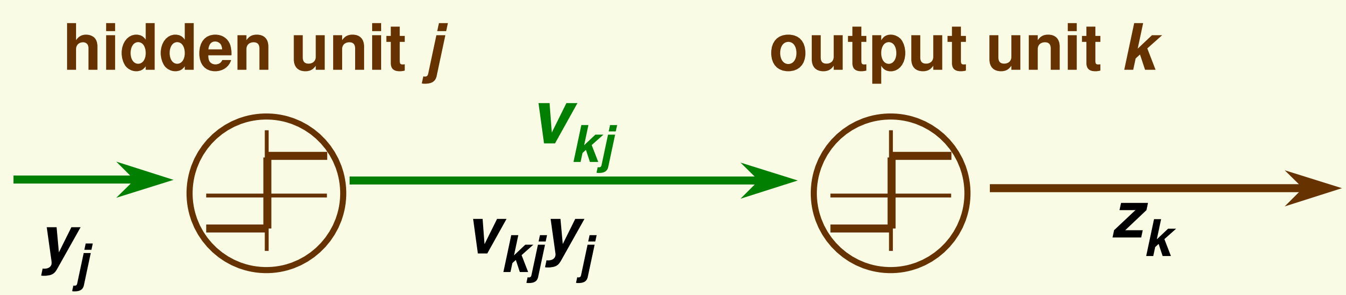


MNN: Notation for Weights

- Use w_{ji} to denote the weight between input unit i and hidden unit j



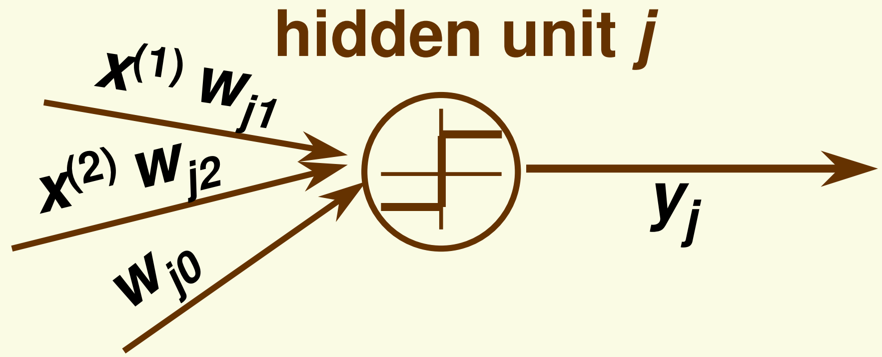
- Use v_{kj} to denote the weight between hidden unit j and output unit k



MNN: Notation for Activation

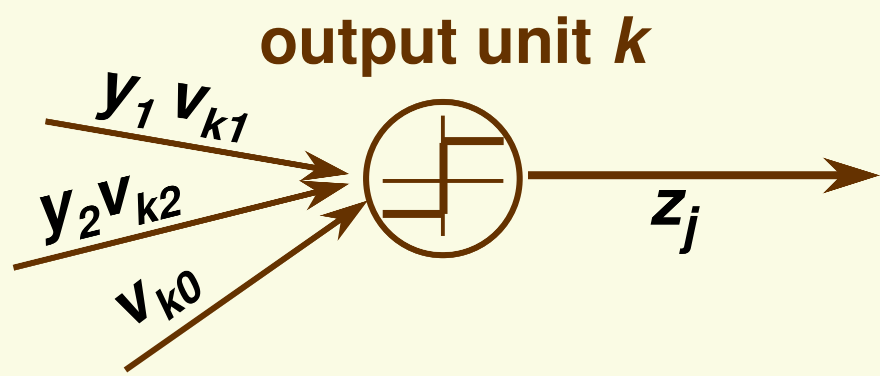
- Use $\boldsymbol{net_i}$ to denote the activation and hidden unit $\boldsymbol{j}$

$$net_j = \sum_{i=1}^{d} x^{(i)} w_{ji} + w_{j0}$$



- Use $\boldsymbol{net^*_k}$ to denote the activation at output unit $\boldsymbol{k}$

$$net^*_k = \sum_{j=1}^{N_H} y_j v_{kj} + v_{k0}$$



Discriminant Function

- Discriminant function for class ***k*** (the output of the ***k*** th output unit)

$$g_k(\boldsymbol{x}) = z_k =$$

$$= f\left(\sum_{j=1}^{N_H} v_{kj} f\left(\sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0}\right) + v_{k0}\right)$$

activation at jth hidden unit

activation at kth output unit

- Rich expressive power: every **continuous** discriminant function can be implemented with enough hidden units, 1 hidden layer, and proper nonlinear activation functions

Expressive Power

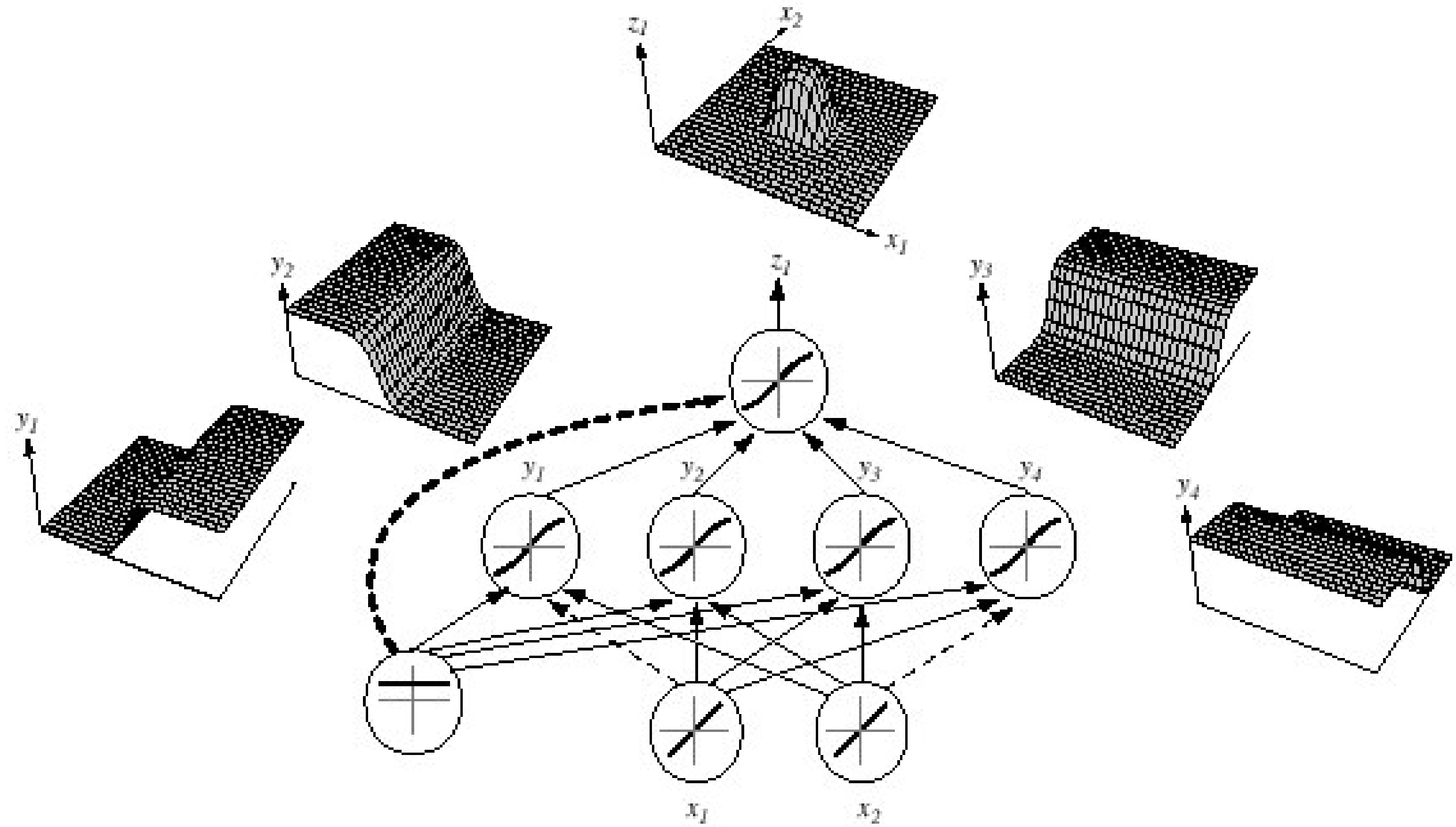


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

MNN Activation function

- Must be nonlinear for expressive power larger than that of perceptron
  - If use linear activation function at hidden layer, can only deal with linearly separable classes
  - Suppose at hidden unit $\boldsymbol{j}$, $\boldsymbol{h}(\boldsymbol{u}) = \boldsymbol{a}_j \boldsymbol{u}$

$$g_k(x) = f\left(\sum_{j=1}^{N_H} v_{kj} h\left(\sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0}\right) + v_{k0}\right)$$

$$= f\left(\sum_{j=1}^{N_H} v_{kj} a_j \left(\sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0}\right) + v_{k0}\right)$$

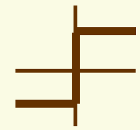
$$= f\left(\sum_{i=1}^{d} \sum_{j=1}^{N_H} \left(v_{kj} a_j w_{ji} x^{(i)} + w_{j0}\right) + v_{k0}\right)$$

$$= f\left(\sum_{i=1}^{d} x^{(i)} \underbrace{\sum_{j=1}^{N_H} v_{kj} a_j w_{ji}}_{w_i^{new}} + \underbrace{\left(\sum_{j=1}^{N_H} w_{j0} + v_{k0}\right)}_{w_0^{new}}\right)$$

MNN Activation function

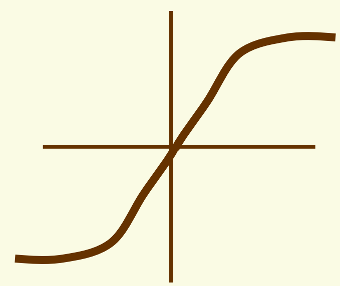
- In previous example, used discontinuous activation function

$$f(net_k) = \begin{cases} 1 & \text{if } net_k \geq 0 \\ -1 & \text{if } net_k < 0 \end{cases}$$



- We will use gradient descent for learning, so we need to use continuous activation function

sigmoid function



- From now on, assume ***f*** is a differentiable function

MNN: Modes of Operation

- Network have two modes of operation:

  - ***Feedforward***

    The feedforward operations consists of presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units (no cycles!)

  - ***Learning***

    The supervised learning consists of presenting an input pattern and modifying the network parameters (weights) to reduce distances between the computed output and the desired output

MNN: Class Representation

- Training samples $x_1, \dots, x_n$ each of class $1, \dots, m$
- Let network output z represent class c as ***target*** $t^{(c)}$

$$z = \begin{bmatrix} z_1 \\ \vdots \\ z_c \\ \vdots \\ z_m \end{bmatrix} = t^{(c)} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow c\text{th row}$$

Our Ultimate Goal For FeedForward Operation

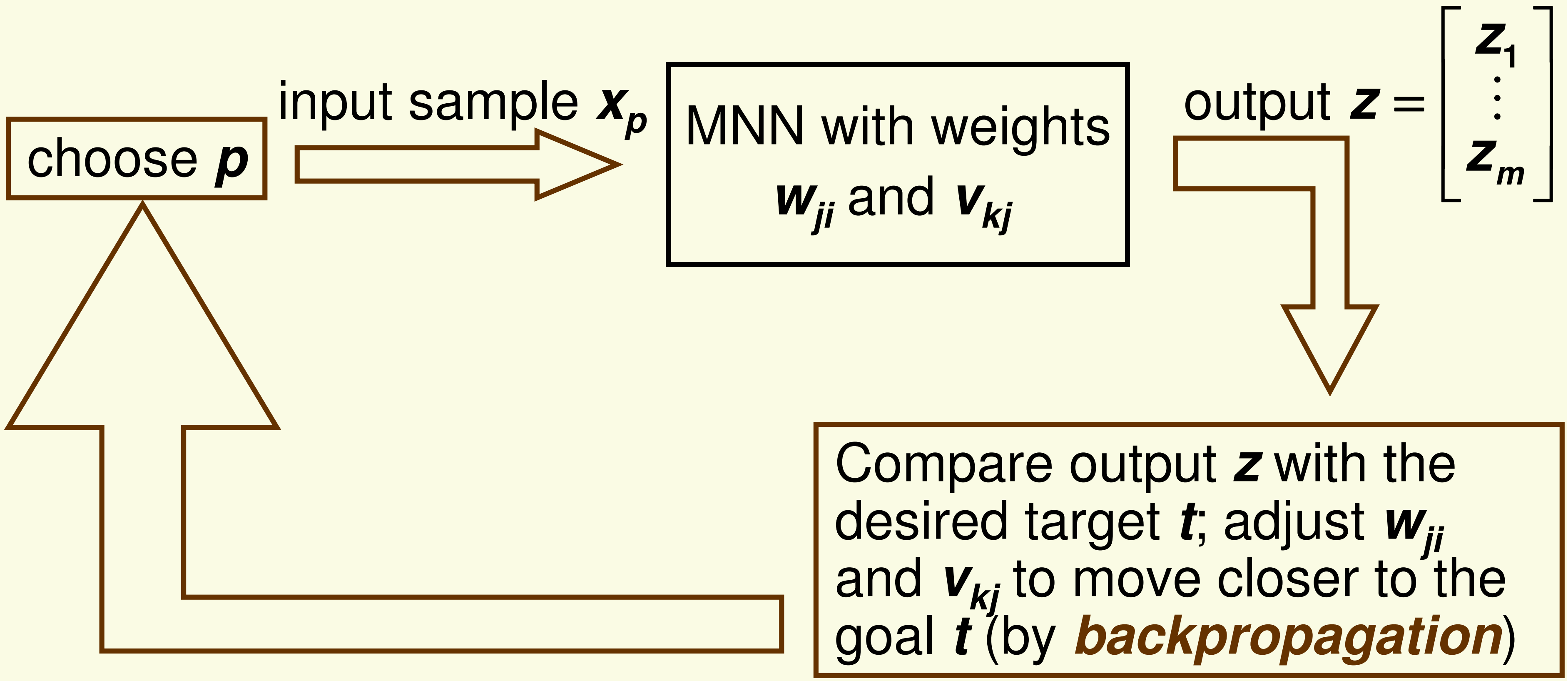
sample of class c ⇒ MNN with weights w_{ji} and v_{kj} ⇒ $t^{(c)}$

MNN training to achieve the Ultimate Goal

Modify (learn) MNN parameters w_{ji} and v_{kj} so that for each ***training*** sample of class c MNN output $z = t^{(c)}$

Network Training (learning)

1. Initialize weights $\boldsymbol{w}_{ji}$ and $\boldsymbol{v}_{kj}$ randomly
2. Iterate until a stopping criterion is reached



BackPropagation

- Learn w_{ji} and v_{kj} by minimizing the training error
- What is the training error?
- Suppose the output of MNN for sample x is z and the target (desired output for x) is t
- Error on one sample: $J(w,v) = \frac{1}{2}\sum_{c=1}^{m}(t_c - z_c)^2$
- Training error: $J(w,v) = \frac{1}{2}\sum_{i=1}^{n}\sum_{c=1}^{m}\left(t_c^{(i)} - z_c^{(i)}\right)^2$
- Use gradient descent:

$v^{(0)}, w^{(0)} =$ random

repeat until convergence:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w J\left(w^{(t)}\right)$$

$$v^{(t+1)} = v^{(t)} - \eta \nabla_v J\left(v^{(t)}\right)$$

BackPropagation

- For simplicity, first take training error for one sample $\boldsymbol{x}_i$

function of w,v

$$J(w,v) = \frac{1}{2}\sum_{c=1}^{m}\left(t_c - z_c\right)^2$$

fixed constant

$$z_k = f\left(\sum_{j=1}^{N_H} v_{kj} f\left(\sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0}\right) + v_{k0}\right)$$

- Need to compute
  1. partial derivative w.r.t. hidden-to-output weights $\frac{\partial J}{\partial v_{kj}}$
  2. partial derivative w.r.t. input-to-hidden weights $\frac{\partial J}{\partial w_{ji}}$

BackPropagation: Layered Model

activation at hidden unit ***j***

$$net_j = \sum_{i=1}^{d} x^{(i)} w_{ji} + w_{j0}$$

⇩

output at hidden unit ***j***

$$y_j = f\left(net_j\right)$$

⇩

activation at output unit ***k***

$$net_k^* = \sum_{j=1}^{N_H} y_j v_{kj} + v_{k0}$$

⇩

activation at output unit ***k***

$$z_k = f\left(net_k^*\right)$$

⇩

objective function

$$J(w,v) = \frac{1}{2}\sum_{c=1}^{m}\left(t_c - z_c\right)^2$$

chain rule ↑ $\dfrac{\partial J}{\partial v_{kj}}$

chain rule ↑ $\dfrac{\partial J}{\partial w_{ji}}$

BackPropagation

$$net_k = \sum_{j=1}^{N_H} y_j v_{kj} + v_{k0} \Rightarrow z_k = f\left(net_k^*\right) \Rightarrow J(w,v) = \frac{1}{2}\sum_{c=1}^{m}(t_c - z_c)^2$$

- First compute hidden-to-output derivatives $\frac{\partial J}{\partial v_{kj}}$

$$\frac{\partial J}{\partial v_{kj}} = \frac{1}{2}\sum_{c=1}^{m}\frac{\partial}{\partial v_{kj}}(t_c - z_c)^2 = \sum_{c=1}^{m}(t_c - z_c)\frac{\partial}{\partial v_{kj}}(t_c - z_c)$$

$$= (t_k - z_k)\frac{\partial}{\partial v_{kj}}(t_k - z_k) = -(t_k - z_k)\frac{\partial}{\partial v_{kj}}(z_k)$$

$$= -(t_k - z_k)\frac{\partial z_k}{\partial net_k^*}\frac{\partial net_k^*}{\partial v_{kj}}$$

$$= \begin{cases} -(t_k - z_k) f'\left(net_k^*\right) y_j & \text{if } j \neq 0 \\ -(t_k - z_k) f'\left(net_k^*\right) & \text{if } j = 0 \end{cases}$$

BackPropagation

Gradient Descent *Single Sample* Update Rule for hidden-to-output weights v_{kj}

j > 0: $$v_{kj}^{(t+1)} = v_{kj}^{(t)} + \eta(t_k - z_k)f'(net_k^*)y_j$$

j = 0 (bias weight): $$v_{k0}^{(t+1)} = v_{k0}^{(t)} + \eta(t_k - z_k)f'(net_k^*)$$

BackPropagation

- Now compute input-to-hidden $\dfrac{\partial J}{\partial w_{ji}}$

$$\frac{\partial J}{\partial w_{ji}} = \sum_{k=1}^{m}(t_k - z_k)\frac{\partial}{\partial w_{ji}}(t_k - z_k)$$

$$= -\sum_{k=1}^{m}(t_k - z_k)\frac{\partial z_k}{\partial w_{ji}} = -\sum_{k=1}^{m}(t_k - z_k)\frac{\partial z_k}{\partial net_k^*}\frac{\partial net_k^*}{\partial w_{ji}}$$

$$= -\sum_{k=1}^{m}(t_k - z_k)f'(net_k^*)\frac{\partial net_k^*}{\partial y_j}\frac{\partial y_j}{\partial w_{ji}}$$

$$= -\sum_{k=1}^{m}(t_k - z_k)f'(net_k^*)v_{kj}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$= -\sum_{k=1}^{m}(t_k - z_k)f'(net_k^*)v_{kj}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$= \begin{cases} -\sum_{k=1}^{m}(t_k - z_k)f'(net_k^*)v_{kj}f'(net_j)x^{(i)} & \text{if } i \neq 0 \\ -\sum_{k=1}^{m}(t_k - z_k)f'(net_k^*)v_{kj}f'(net_j) & \text{if } i = 0 \end{cases}$$

$$\Downarrow$$

$$net_h = \sum_{h=1}^{d} x^{(i)} w_{hi} + w_{h0}$$

$$\Downarrow$$

$$y_j = f(net_j)$$

$$\Downarrow$$

$$net_k^* = \sum_{s=1}^{N_H} y_s v_{ks} + v_{k0}$$

$$\Downarrow$$

$$z_k = f(net_k^*)$$

$$\Downarrow$$

$$J(w,v) = \frac{1}{2}\sum_{c=1}^{m}(t_c - z_c)^2$$

BackPropagation

$$\frac{\partial J}{\partial w_{ji}} = \begin{cases} -f'(net_j)\, x^{(i)} \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*)\, v_{kj} & \text{if } i \neq 0 \\ -f'(net_j) \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*)\, v_{kj} & \text{if } i = 0 \end{cases}$$

Gradient Descent *Single Sample* Update Rule for input-to-hidden weights w_{ji}

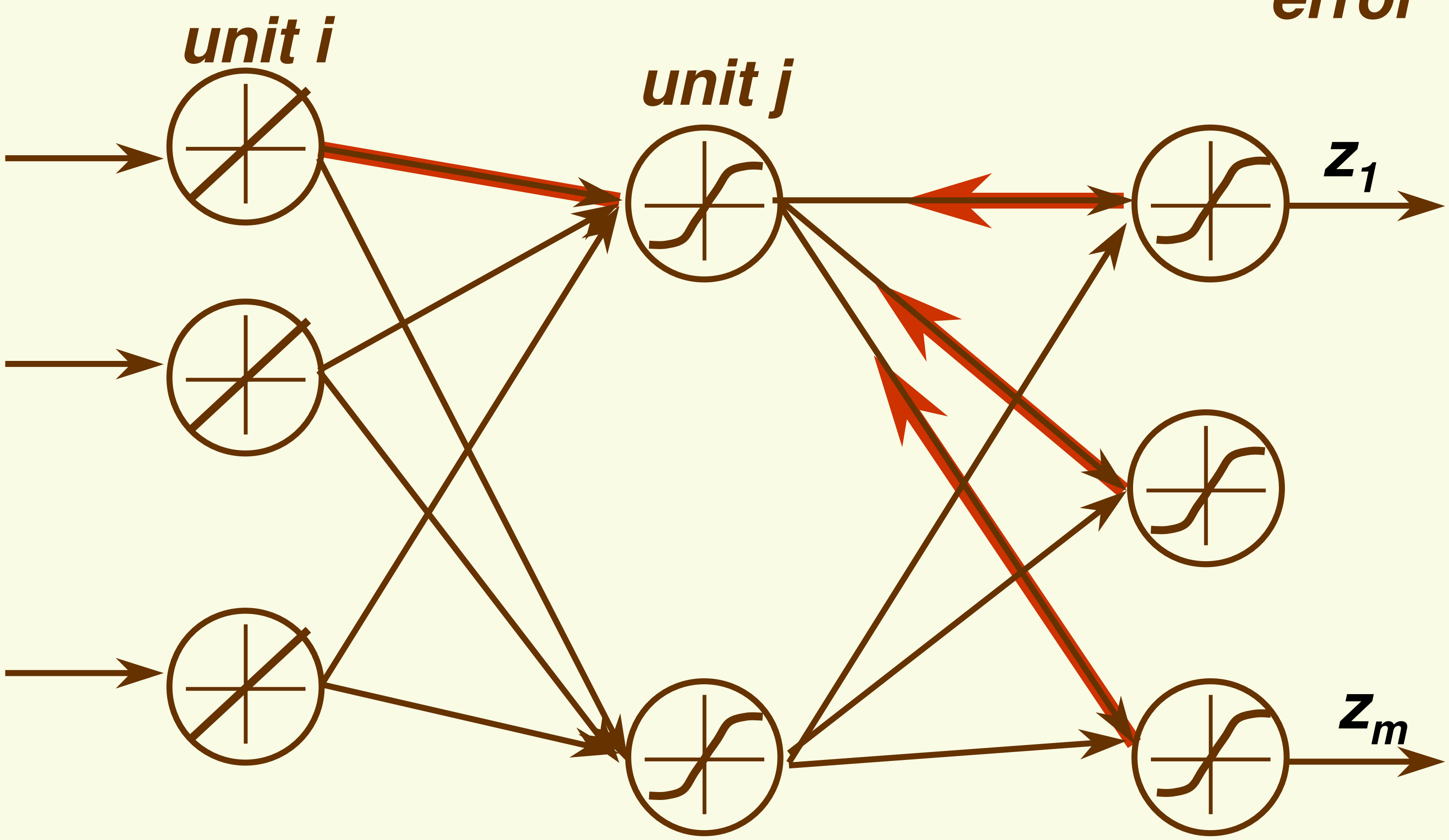
i > 0: $w_{ji}^{(t+1)} = w_{ji}^{(t)} + \eta f'(net_j)\, x^{(i)} \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*)\, v_{kj}$

i = 0 (bias weight): $w_{j0}^{(t+1)} = w_{j0}^{(t)} + \eta f'(net_j) \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*)\, v_{kj}$

BackPropagation of Errors

$$\frac{\partial J}{\partial w_{ji}} = -f'(net_j)\, x^{(i)} \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*)\, v_{kj}$$

$$\frac{\partial J}{\partial v_{kj}} = -\underbrace{(t_k - z_k)}_{error} f'(net_k^*)\, y_j$$



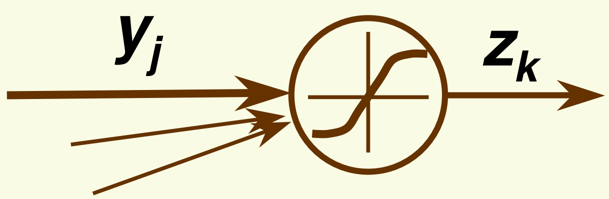
- Name “backpropagation” because during training, errors propagated back from output to hidden layer

BackPropagation

- Consider update rule for hidden-to-output weights:

$$v_{kj}^{(t+1)} = v_{kj}^{(t)} + \eta(t_k - z_k)f'(net_k^*)y_j$$

- Suppose $t_k - z_k > 0$
- Then output of the ***k***th hidden unit is too small: $t_k > z_k$
- Typically activation function ***f*** is s.t. ***f' > 0***
- Thus $(t_k - z_k)f'(net_k^*) > 0$



- There are 2 cases:
  1. $y_j > 0$, then to increase z_k, should increase weight v_{kj} which is exactly what we do since $\eta(t_k - z_k)f'(net_k^*)y_j > 0$
  2. $y_j < 0$, then to increase z_k, should decrease weight v_{kj} which is exactly what we do since $\eta(t_k - z_k)f'(net_k^*)y_j < 0$

BackPropagation

- The case $t_k - z_k < 0$ is analogous
- Similarly, can show that input-to-hidden weights make sense
- Important: weights should be initialized to random ***nonzero*** numbers

$$\frac{\partial J}{\partial w_{ji}} = -f'(net_j)\, x^{(i)} \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*)\, v_{kj}$$

- if $v_{kj} = 0$, input-to-hidden weights w_{ji} never updated

Training Protocols

- How to present samples in training set and update the weights?
- Three major training protocols:
  1. Stochastic
     - Patterns are chosen randomly from the training set, and network weights are updated after every sample presentation
  2. Batch
     - weights are update based on all samples; iterate weight update
  3. Online
     - each sample is presented only once, weight update after each sample presentation

Stochastic Back Propagation

1. Initialize
   - number of hidden layers n_H
   - weights w, v
   - convergence criterion θ and learning rate η
   - time $t = 0$
2. ***do***

   $x \leftarrow$ randomly chosen training pattern

   for all $0 \le i \le d,\ 0 \le j \le n_H,\ 0 \le k \le m$

$$w_{ji} = w_{ji} + \eta f'(net_j) x^{(i)} \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}$$

$$w_{j0} = w_{j0} + \eta f'(net_j) \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}$$

$$v_{kj} = v_{kj} + \eta (t_k - z_k) f'(net_k^*) y_j$$

$$v_{k0} = v_{k0} + \eta (t_k - z_k) f'(net_k^*)$$

   $t = t + 1$

   until $\| J \| < \theta$

3. ***return*** v, w

Batch Back Propagation

- This is the ***true*** gradient descent, (unlike stochastic propagation)
- For simplicity, derived backpropagation for a single sample objective function:

$$J(w,v)=\frac{1}{2}\sum_{c=1}^{m}(t_c-z_c)^2$$

- The full objective function:

$$J(w,v)=\frac{1}{2}\sum_{i=1}^{n}\sum_{c=1}^{m}\left(t_c^{(i)}-z_c^{(i)}\right)^2$$

- Derivative of full objective function is just a sum of derivatives for each sample:

$$\frac{\partial}{\partial w}J(w,v)=\frac{1}{2}\sum_{i=1}^{n}\frac{\partial}{\partial w}\left(\sum_{c=1}^{m}\left(t_c^{(i)}-z_c^{(i)}\right)^2\right)$$

already derived this

Batch Back Propagation

- For example,

$$\frac{\partial J}{\partial w_{ji}} = \sum_{p=1}^{n} - f'\left(net_j\right) x_p^{(i)} \sum_{k=1}^{m} \left(t_k - z_k\right) f'\left(net_k^*\right) v_{kj}$$

Batch Back Propagation

1. Initialize $\boldsymbol{n_H}$, $\boldsymbol{w}$, $\boldsymbol{v}$, θ, η, $\boldsymbol{t} = 0$
2. ***do***

$$\Delta v_{kj} = \Delta v_{k0} = \Delta w_{ji} = \Delta w_{j0} = 0$$

one epoch

for all $1 \le p \le n$

for all $0 \le i \le d,\ 0 \le j \le n_H,\ 0 \le k \le m$

$$\Delta v_{kj} = \Delta v_{kj} + \eta (t_k - z_k) f'(net_k^*) y_j$$

$$\Delta v_{k0} = \Delta v_{k0} + \eta (t_k - z_k) f'(net_k^*)$$

$$\Delta w_{ji} = \Delta w_{ji} + \eta f'(net_j) x_p^{(i)} \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}$$

$$\Delta w_{j0} = \Delta w_{j0} + \eta f'(net_j) \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}$$

$$v_{kj} = v_{kj} + \Delta v_{kj};\ v_{k0} = v_{k0} + \Delta v_{k0};\ w_{ji} = w_{ji} + \Delta w_{ji};\ w_{j0} = w_{j0} + \Delta w_{j0}$$

$$t = t + 1$$

until $\| J \| < \theta$

3. ***return*** $\boldsymbol{v}$, $\boldsymbol{w}$

Training Protocols

1. Batch
   - True gradient descent
2. Stochastic
   - Faster than batch method
   - Usually the recommended way
3. Online
   - Used when number of samples is so large it does not fit in the memory
   - Dependent on the order of sample presentation
   - Should be avoided when possible