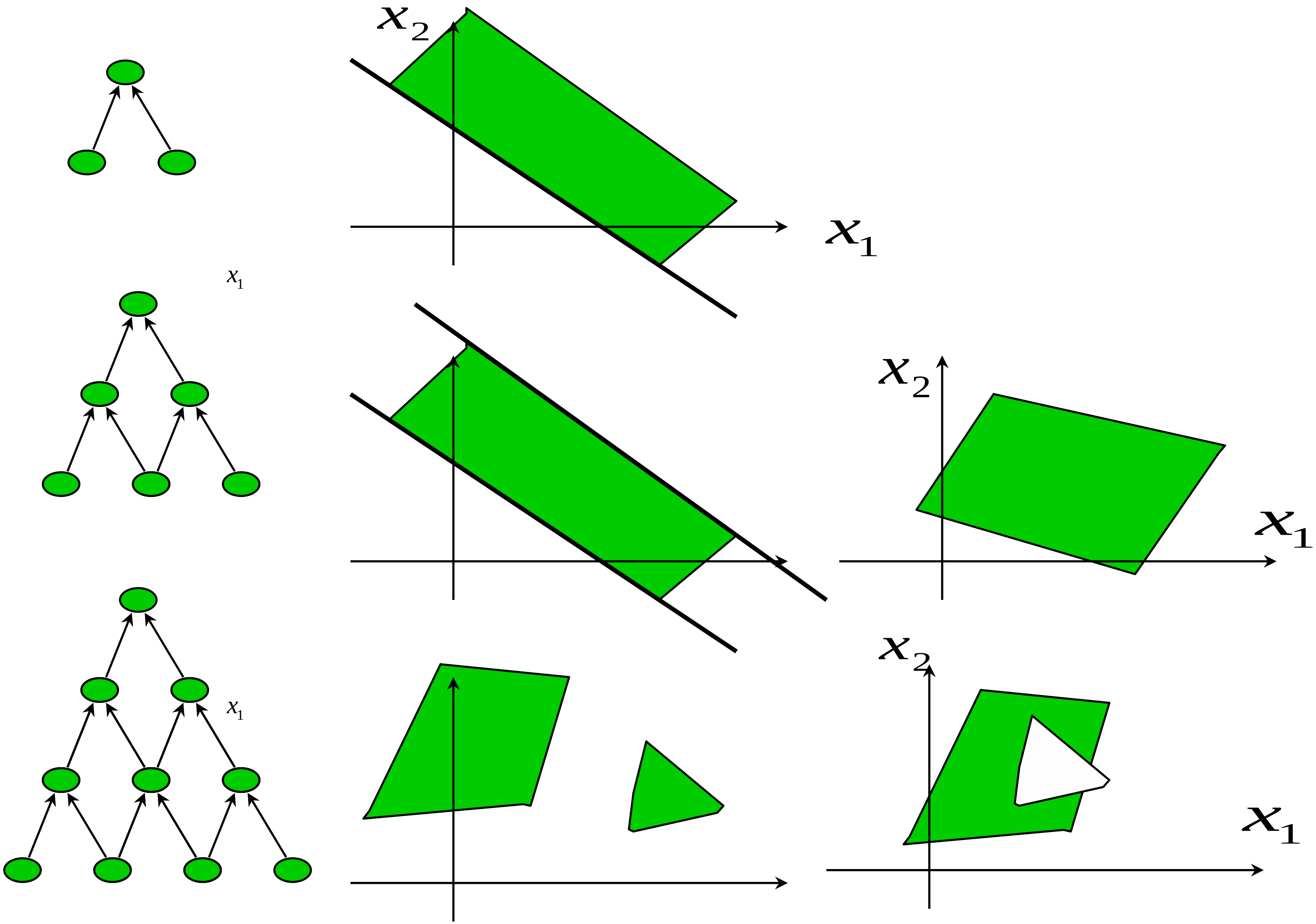
Multi-Layer Perceptrons

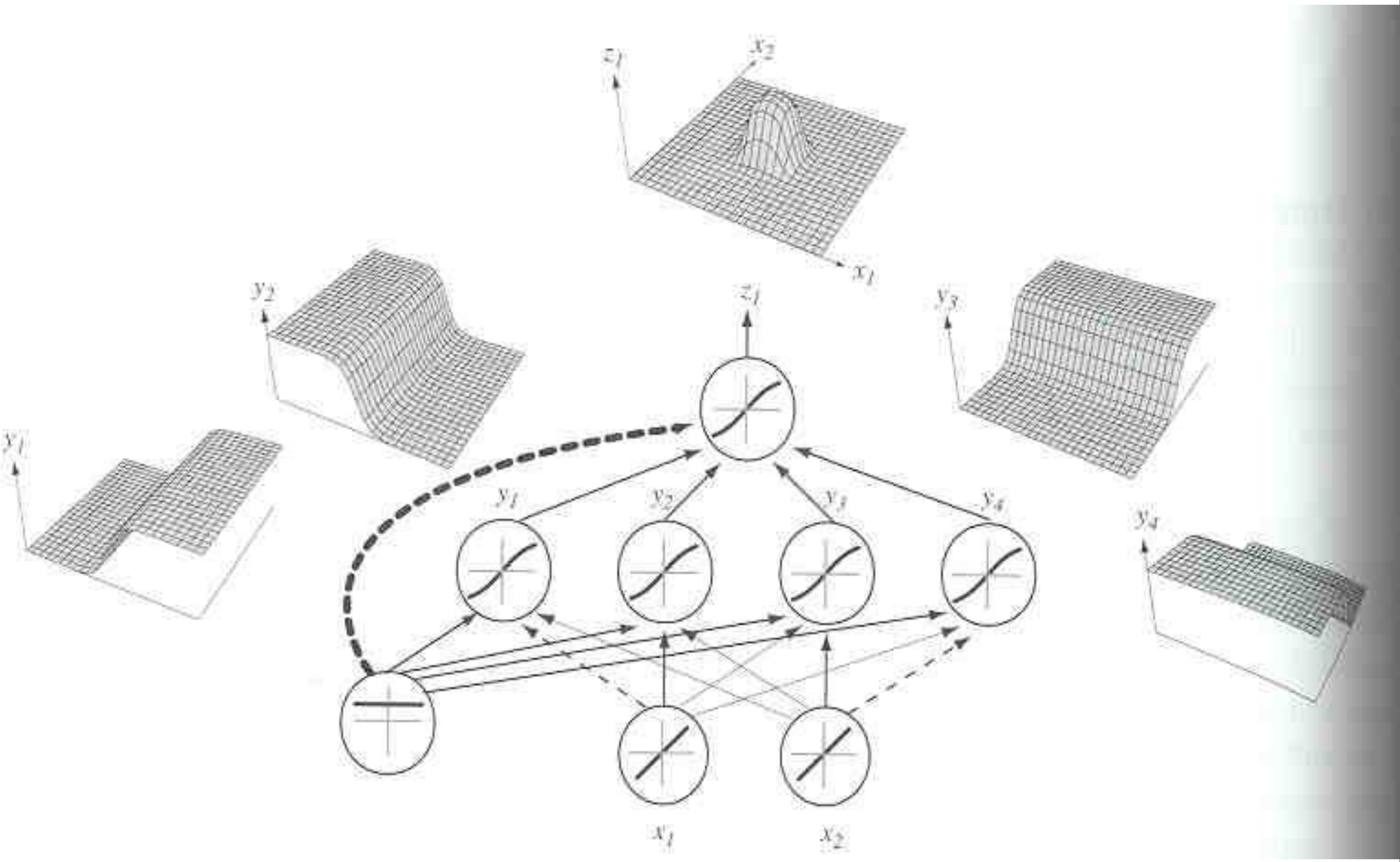
Multi-Layer Perceptrons

- With “hidden” layers
- One hidden layer - any Boolean function or convex decision regions
- Two hidden layers - arbitrary decision regions

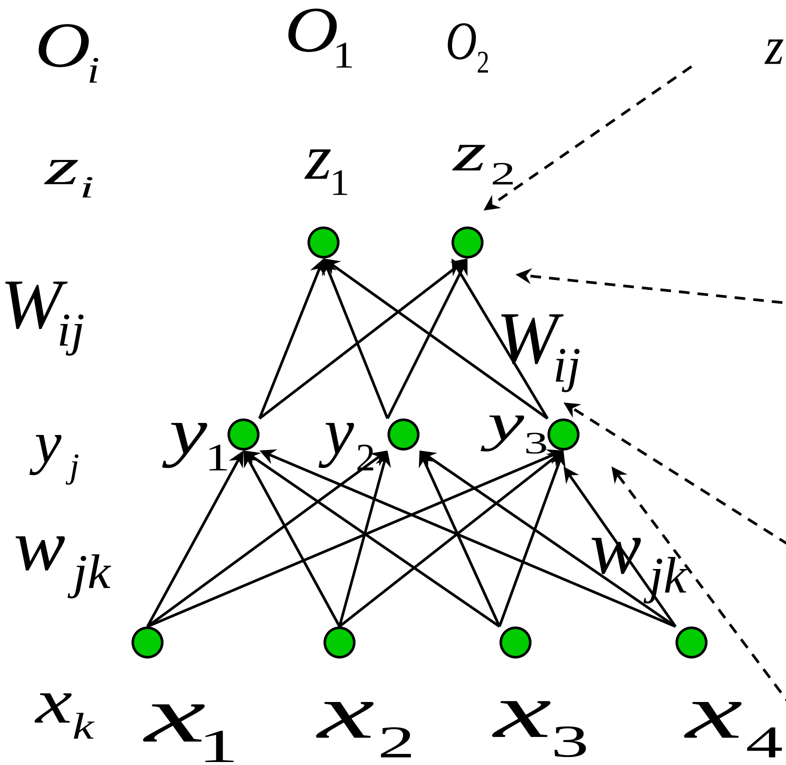
Decision boundaries



Decision Boundaries



Backpropagation Learning rule



$$z_i^{\,u} = g(NET_i^{\,u}) = g(\sum_j W_{ij} y_j^{\,u}) = g(\sum_j W_{ij} g(\sum_k w_{jk} x_k^{\,u}))$$

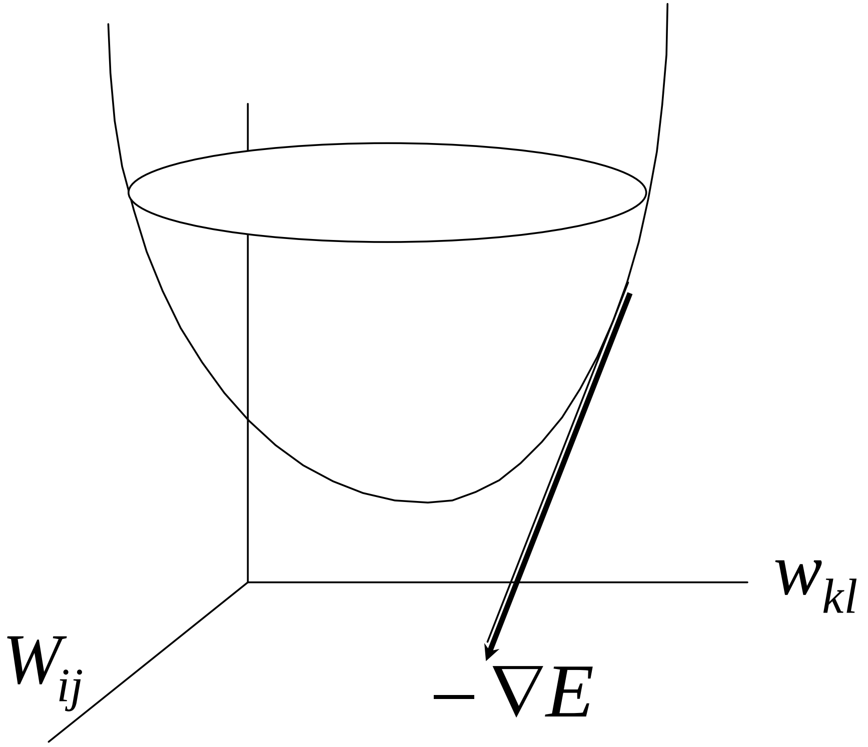
$$NET_i^{\,u} = \sum_j W_{ij} y_j^{\,u} = \sum_j W_{ij} g(\sum_k w_{jk} x_k^{\,u})$$

$$y_j^{\,u} = g(net_j^{\,u}) = g(\sum_k w_{jk} x_k^{\,u})$$

$$net_j^{\,u} = \sum_k w_{jk} x_k^{\,u}$$

Cost function

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_{u,i}(O_i^{\ u} - z_i^{\ u})^2 = \frac{1}{2}\sum_{u,i}(O_i^{\ u} - g(\sum_j W_{ij}\, g(\sum_k w_{jk} x_k^{\ u}))^2$$



Change w.r.t. W_{ij}

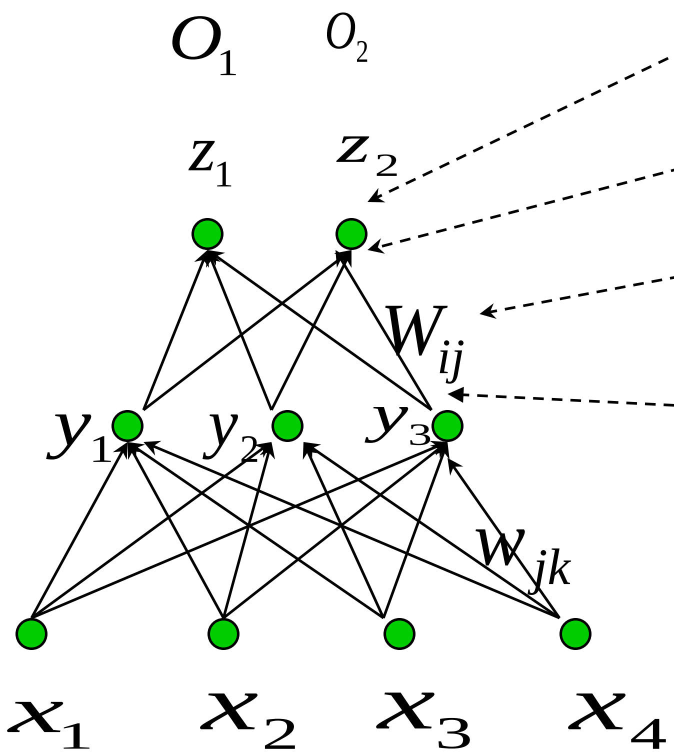
$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = -\eta \frac{\partial \frac{1}{2}(O_i^u - g(\sum_j W_{ij} y_j^u))^2}{\partial W_{ij}}$$

$$= -\eta \frac{1}{2} \frac{\partial (O_i^u - z_i^u)^2}{\partial (O_i^u - z_i^u)} \frac{\partial (O_i^u - g(NET_i^u))}{\partial NET_i^u} \frac{\sum_j W_{ij} y_j^u}{\partial W_{ij}}$$

$$= \eta \sum_u \underline{(O_i^u - z_i^u) g'(NET_i^u)} y_j^u$$

$$= \eta \sum_u \underline{\delta_i^u} y_j^u \qquad \delta_i^u = (O_i^u - z_i^u) g'(NET_i^u)$$

Interpretation



$$(O_i^u - z_i^u)$$

$$g'(NET_i^u)$$

$$\delta_i^u = (O_i^u - z_i^u) g'(NET_i^u)$$

$$\eta \sum_u \delta_i^u y_j^u$$

$$y_j^u$$

- A form of Hebb's rule
  - If error, reward mutual excitation (large feedback output and large input)

Change w.r.t. w_{ij}

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \frac{\partial \sum_{u,i} \frac{1}{2}\left(O_i^u - g\left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} x_k^u\right)\right)\right)^2}{\partial w_{jk}}$$

$$= -\eta \frac{\partial \sum_{u,i} \frac{1}{2}\left(O_i^u - g\left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} x_k^u\right)\right)\right)^2}{\partial \left(O_i^u - g\left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} x_k^u\right)\right)\right)} \frac{\partial \sum_{u,i} \left(O_i^u - g\left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} x_k^u\right)\right)\right)}{\partial \left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} x_k^u\right)\right)}$$

$$\frac{\partial \sum_{u,i} \left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} x_k^u\right)\right)}{\partial g\left(\sum_k w_{jk} x_k^u\right)} \frac{\partial g\left(\sum_k w_{jk} x_k^u\right)}{\partial \sum_k w_{jk} x_k^u} \frac{\partial \sum_k w_{jk} x_k^u}{\partial w_{jk}}$$

$$= \eta \sum_{u,i} (O_i^u - z_i^u)\, g'(NET_i^u)\, W_{ij}\, g'(net_j^u)\, x_k^u$$

$$= \eta \sum_{u,i} \delta_i^u W_{ij}^u\, g'(net_j^u)\, x_k^u$$

$$= \eta \sum_u \delta_j^u x_k^u \qquad \delta_j^u = g'(net_j^u) \sum_i \delta_i^u W_{ij}$$

Change w.r.t. w_{ij}

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \frac{\partial \sum_{u,i} \frac{1}{2}(O_i^u - g(\sum_j W_{ij} g(\sum_k w_{jk} x_k^u))^2}{\partial w_{jk}}$$

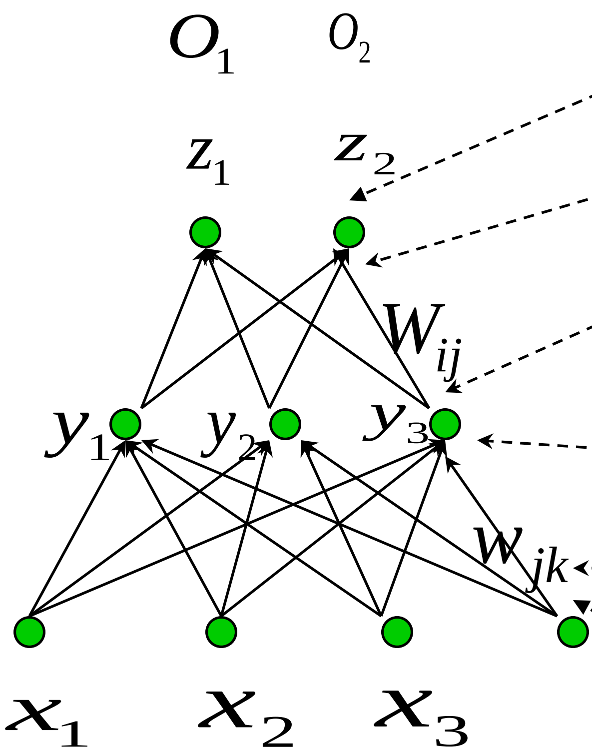
$$= -\eta \frac{\partial E}{\partial y_j^u} \frac{\partial y_j^u}{\partial w_{jk}}$$

$$= \eta \sum_{u,i} (O_i^u - z_i^u) g'(NET_i^u) W_{ij} g'(net_j^u) x_k^u$$

$$= \eta \sum_{u,i} \delta_i^u W_{ij}^u g'(net_j^u) x_k^u$$

$$= \eta \sum_{u} \delta_j^u x_k^u \qquad \delta_j^u = g'(net_j^u) \sum_i \delta_i^u W_{ij}$$

Interpretation (cont.)



$(O_i^u - z_i^u)$ Error

$\delta_i^u = (O_i^u - z_i^u) g'(NET_i^u)$ backpropagated error

$\sum_i \delta_i^u W_{ij}$ Weighted backpropagated error

$\delta_j^u = g'(net_j^u) \sum_i \delta_i^u W_{ij}$

$g'(net_j^u)$ Weighted backpropagated error backpropagated again

$\eta \sum_u \delta_j^u x_k^u$ Weighted backpropagated error backpropagated again and weighted again

x_k^u

- A two-level Hebb's rule
  - If error, reward mutual excitation (large feedback output and large input)

Interpretation (cont.)

$$\Delta w_{pq} = \eta \sum_{patterns} \delta_{output} \times V_{input}$$

- Hebb’s learning
- Error at the output end
- Activation at the input end
- Learning rate

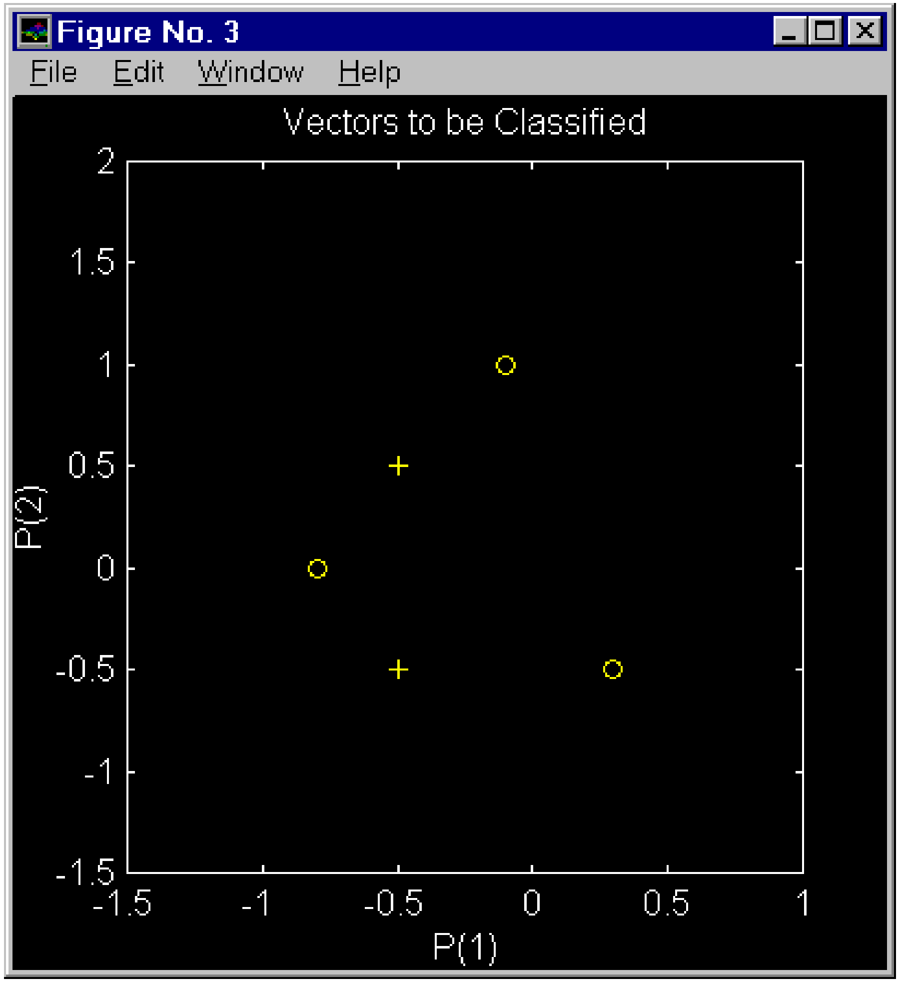


Figure No. 3
File Edit Window Help
Vectors to be Classified
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5 -1 -0.5 0 0.5 1
P(1)
P(2)

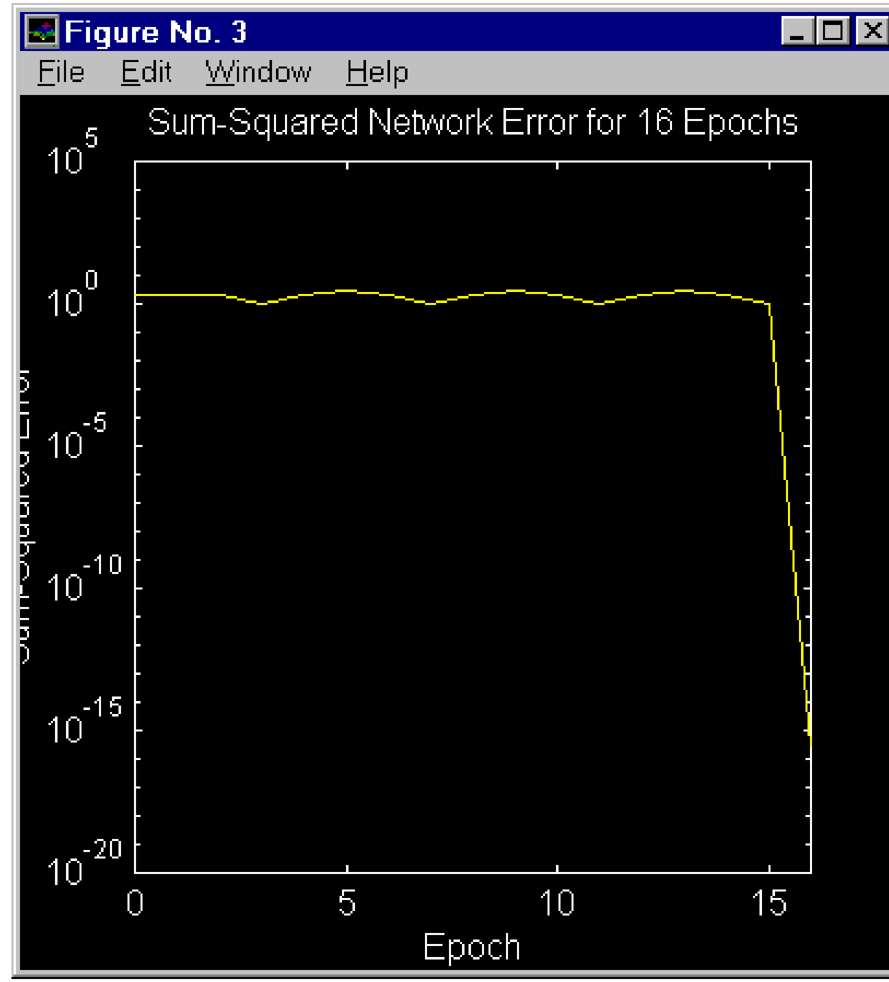
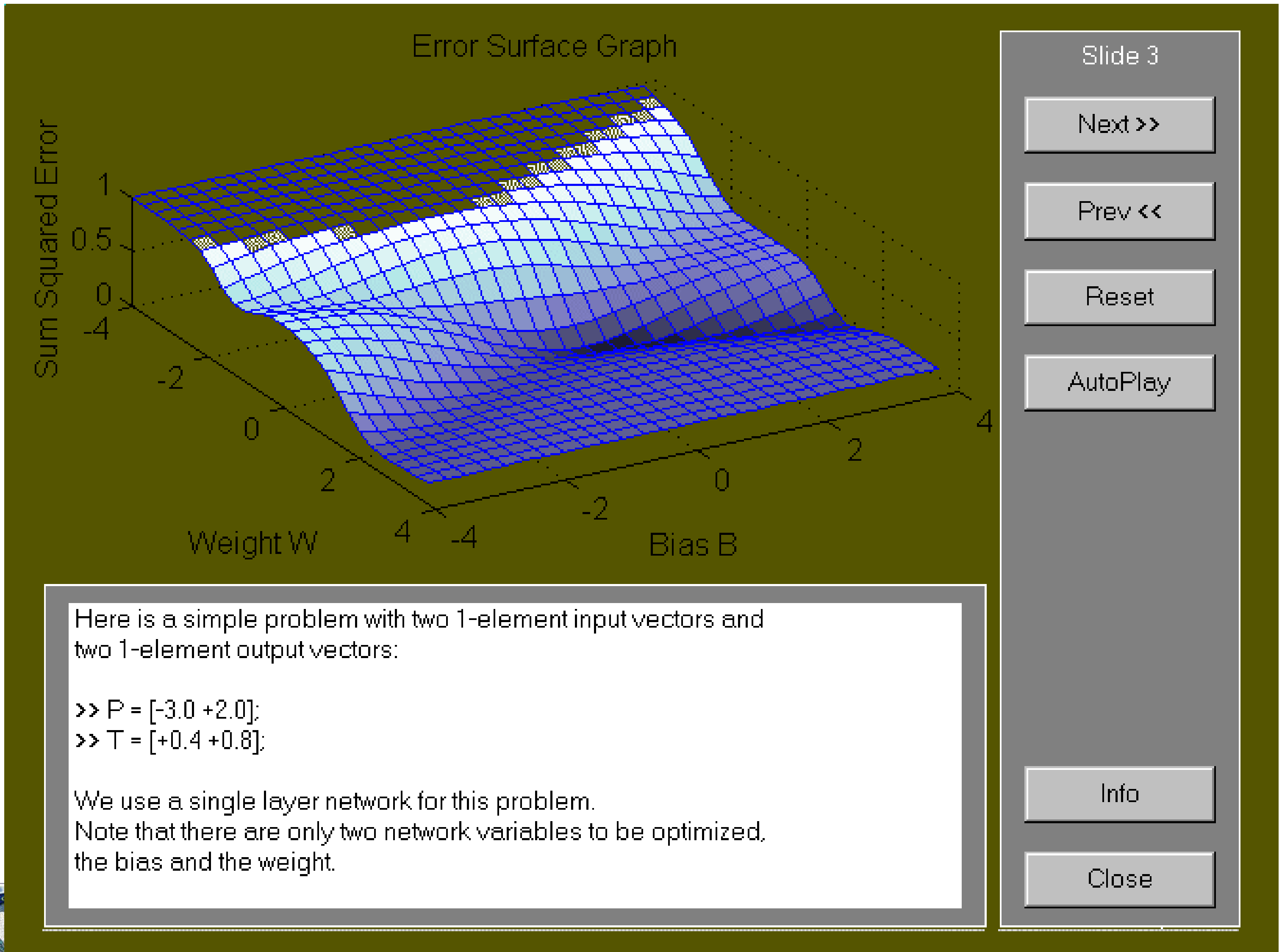
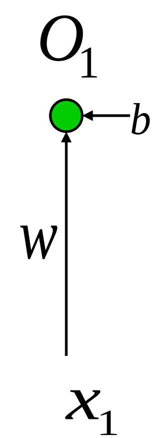
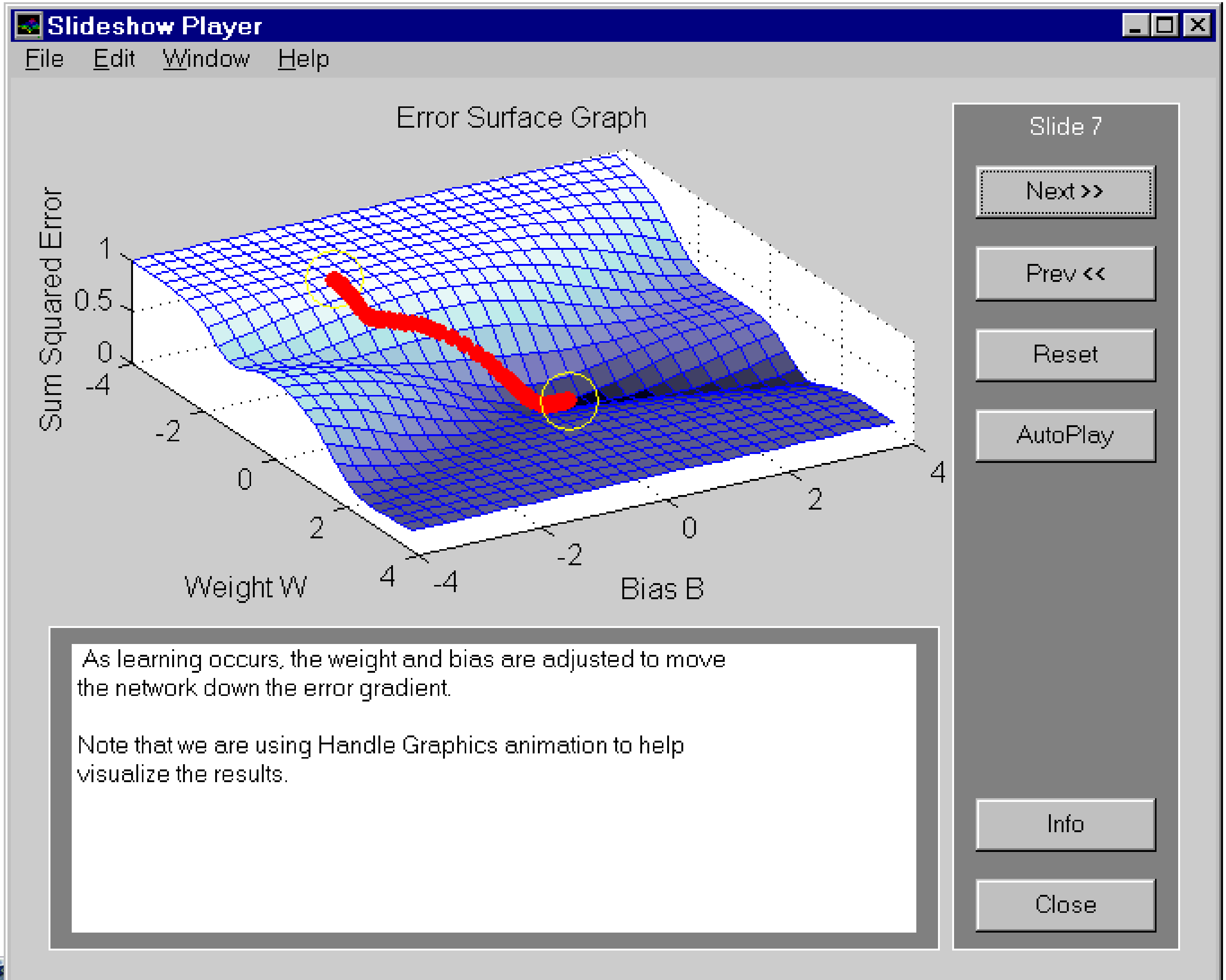


Figure No. 3
File Edit Window Help
Sum-Squared Network Error for 16 Epochs
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
0 5 10 15
Epoch

Graphics Illustration of Backpropagation






Slideshow Player
File Edit Window Help
Error Surface Graph
Sum Squared Error
1
0.5
0
-4
-2
0
2
4
Weight W
-4
-2
0
2
4
Bias B
As learning occurs, the weight and bias are adjusted to move the network down the error gradient.
Note that we are using Handle Graphics animation to help visualize the results.
Slide 7
Next >>
Prev <<
Reset
AutoPlay
Info
Close

Caveats on Backpropagation

- Slow
- Network Paralysis
  - if weights become large
  - operates at limits of squash (transfer) functions
  - derivatives of squash function (feedback) small
- Step size
  - too large may lead to saturation
  - too small cause slow convergence

Caveats on Backpropagation

- Local minima
  - many different initial guesses
  - momentum
  - varying step size (large initially, getting small as training goes on)
  - simulated annealing
- Temporal instability
  - learn B and forgot about A

Other than BackPropagation

- In reality, gradient descent is slow and highly dependent on initial guess
- More sophisticated numerical methods exist Trust region methods, combination of
  - Gradient descent
  - Newton's methods

Caveats

- Error backpropagation is the work horse of all such learning algorithms
- In reality, "hodge-podge" of hacks, tweaks and trials and errors are needed
- Experience and intuition (or dumb luck) are keys

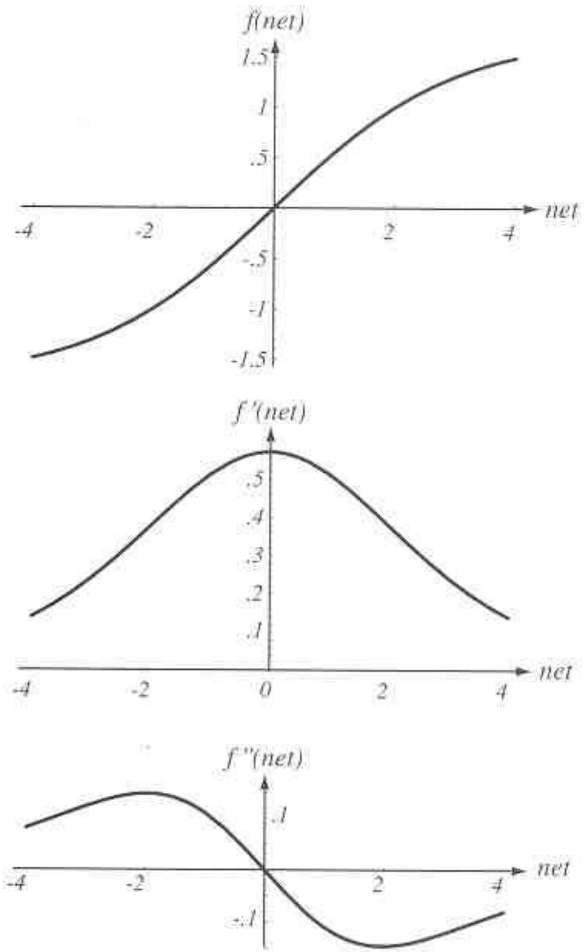
Other Practical Issues

- Which transfer function (g)?
  - g must be nonlinear

$$net_j^u = \sum_k w_{jk} x_k^u \Rightarrow \mathbf{H} = \mathbf{WX}$$

  - g should be continuous and smooth
    - So g and g’ are defined
  - g should saturate
    - Biologically (electronically) plausible

Sigmoid Function

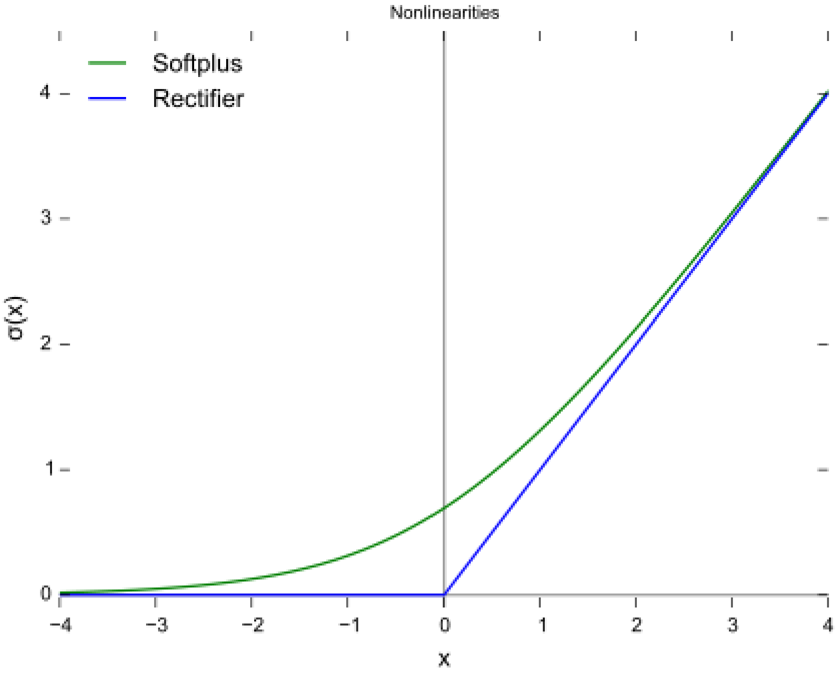


$$g = a\tanh(b \cdot net) = a\frac{1-e^{-b \cdot net}}{1+e^{-b \cdot net}} = \frac{2a}{1+e^{-b \cdot net}} - a$$

$$a = 1.716$$
$$b = 2/3$$

Trend

- Sigmod is replaced by ReLu (rectified linear unit) or soft plus in many applications



Input Scaling

- Inputs (weight, size, etc.) have different units and dynamic range and may be learned at different rates
- Small input ranges make small contribution to the error and are often ignored
- Normalization to same range and same variance (similar to Whitening transform)

Weight initialization

- Don’t set the initial weights to zero, the network is not going to learn at all
- Don’t set the initial weights too high, that leads to paralysis and slow learning (with sigmoid function)
- Don’t set the initial weight too small, output signal shrinkage is a problem

Weight initialization

- Random initialization
  - both positive and negative random weights to insure uniform learning
- Xaiver initialization
  - Certain variance of the weight distribution should be maintained (to avoid shrinkage and blowup problems)

Xavier Initialization

- A single neuron $$Y = W_1X_1 + W_2X_2 + \cdots + W_nX_n$$
- Variance f a single term

$$\text{Var}(W_iX_i) = E[X_i]^2\text{Var}(W_i) + E[W_i]^2\text{Var}(X_i) + \text{Var}(W_i)\text{Var}(i_i)$$

- Assume zero mean $$\text{Var}(W_iX_i) = \text{Var}(W_i)\text{Var}(X_i)$$
- Output variance $$\text{Var}(Y) = \text{Var}(W_1X_1 + W_2X_2 + \cdots + W_nX_n) = n\text{Var}(W_i)\text{Var}(X$$
  - n var(w_i) input variance
- Maintain same variance $$\text{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$
- if n_{in} and n_{out} are different $$\text{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

Output Scaling

- Rule of thumb: Avoid operating neurons in the saturation (tail) regions
  - Tendency for weight saturation
  - g' is small, learning is very slow
  - For sigmoid function as shown before, use range (-1, 1) instead of (-1.716, 1.716)

Output Scaling: Batch Normalization

- Maintain mean and variance of not just input, but also output
- Xavier initialization (?)
  - Too many assumptions (independence, zero mean, etc.) not holding
- Forced renormalization after each layer
  - Zero mean and unit variance
  - Done batch by batch before ReLu

Error Functions

- Autoencoder
  - Reproducing output automatically
  - No single feature is more or less important than others
  - RMS error

Classifier

- Outputs untrimmed indicator scores
- Two cases:
  - One-hot encoding: a dog, a cat, a vehicle, a person, etc.
  - General encoding: President Obama predicting final-4 outcome. Political? Sports? Comedy?
    - A probability function

Classifier Error Func

- Two components:
  - Forced normalization: e.g. softmax

$$\sigma : \mathbb{R}^K \to [0,1]^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Error: cross entropy

$$H(p,q) = -\sum_x p(x)\, \log q(x).$$

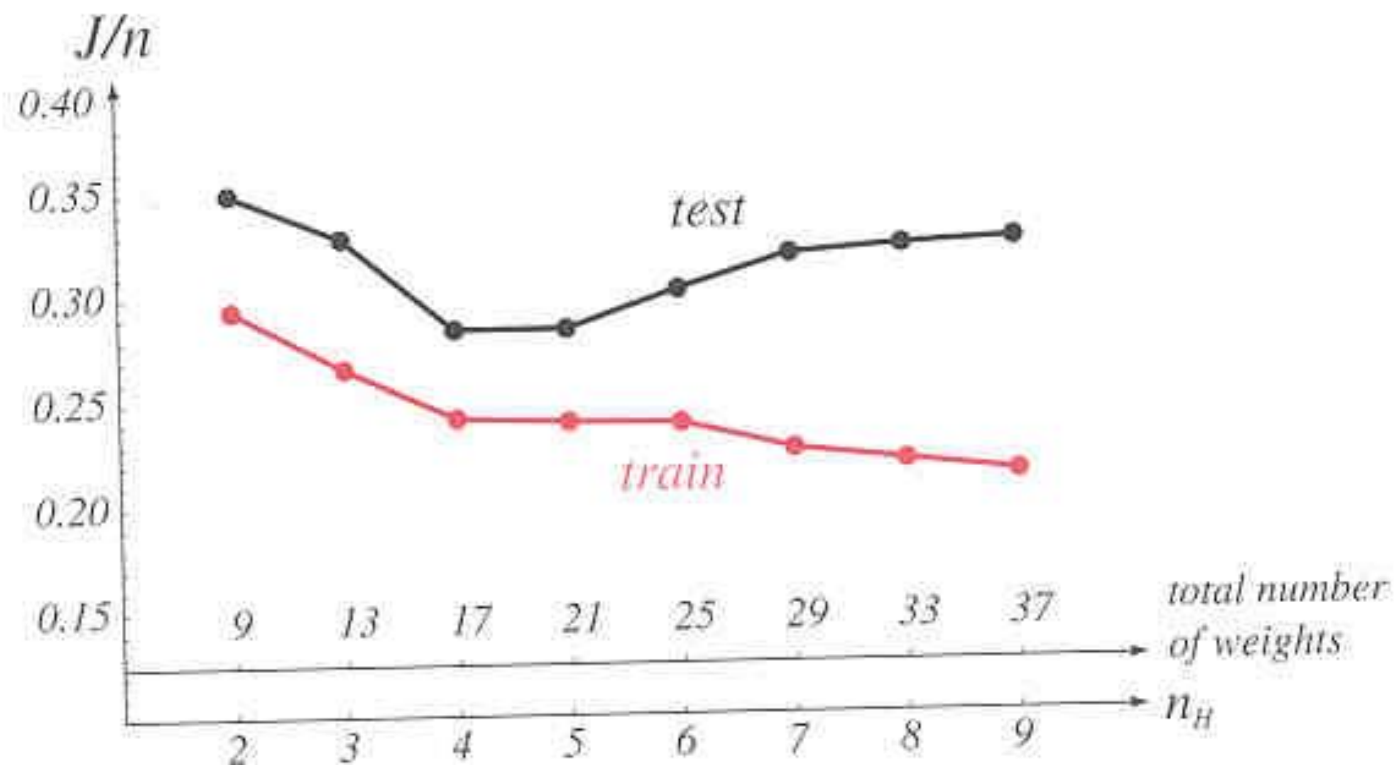
- E.g., in tensorflow

tf.nn.softmax_cross_entropy_with_logits

```
softmax_cross_entropy_with_logits(
    _sentinel=None,
    labels=None,
    logits=None,
    dim=-1,
    name=None
)
```

Number of Hidden Layers

- Too few – poor fitting
- Too many – over fitting, poor generalization



Numerical Stability – step size

- Adaptive

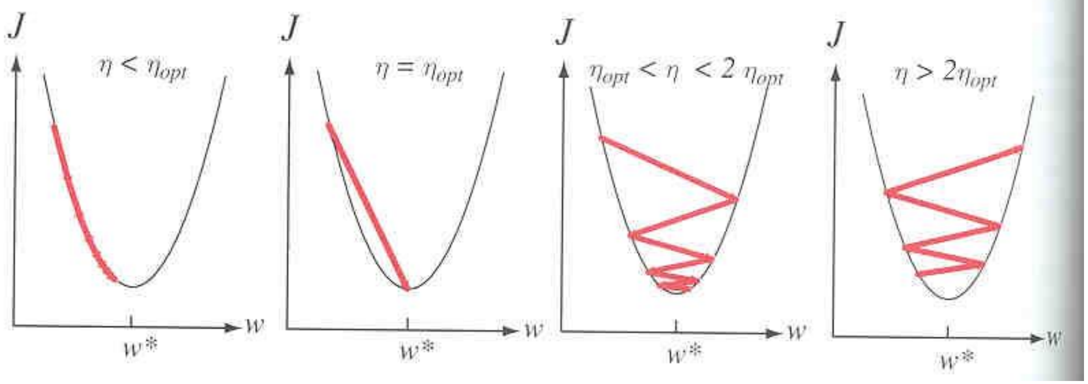
$$J(\mathbf{w}+\Delta\mathbf{w}) = J(\mathbf{w}) + \frac{\partial J}{\partial \mathbf{w}}\Delta\mathbf{w} + \frac{1}{2}\frac{\partial^2 J}{\partial \mathbf{w}^2}\Delta\mathbf{w}^2$$

$$\frac{J(\mathbf{w}+\Delta\mathbf{w}) - J(\mathbf{w})}{\Delta\mathbf{w}} \approx 0 = \frac{\partial J}{\partial \mathbf{w}} + \frac{\partial^2 J}{\partial \mathbf{w}^2}\Delta\mathbf{w}$$

$$\eta_{opt} < \eta < 2\eta_{opt}$$

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{\partial^2 J}{\partial \mathbf{w}^2}\Delta\mathbf{w}$$

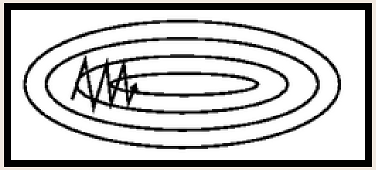
$$\eta_{opt} = \left(\frac{\partial^2 J}{\partial \mathbf{w}^2}\right)^{-1}$$



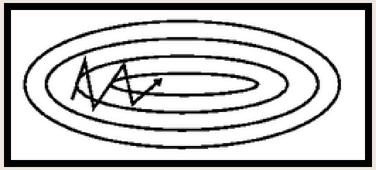
Numerical Stability - momentum

$$\mathbf{w}^{new} = \mathbf{w}^{curr} + (1-\alpha)\Delta\mathbf{w}_{bp}^{curr} + \alpha\Delta\mathbf{w}^{prev}$$

$\alpha \approx 0.9$

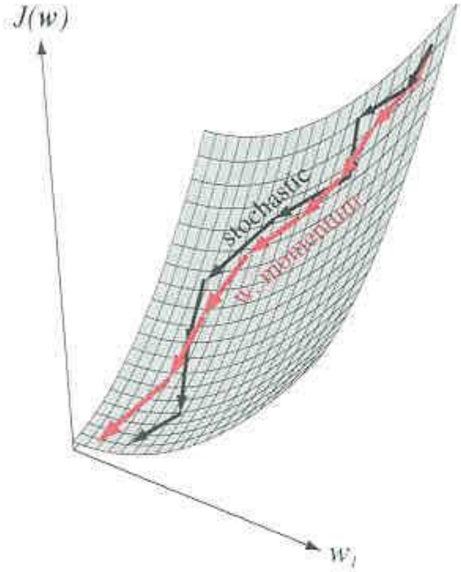


(Fig. 2a)



(Fig. 2b)

Without w. momentum



- Red: as computed from current back propagation
- Blue: as computed from previous back propagation

Numerical Stability

- Weight decay
  - To ensure no single large weight dominates the training process

$$\mathbf{w}^{new} = \mathbf{w}^{old}(1-\xi)$$

Optimizers

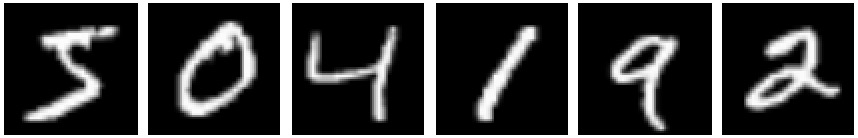
- Wrapper around error backpropagation
  - Stochastic GD, Moment, adaptive stepsize (advanced line search), and decay are often there
  - E.g., Adam Optimizer (adaptive and time varying learning rate for all parameters)
  - Not for fainted heart, ask around!

Essentially

- Yes, multi-layer perceptrons can distinguish classes even when they are not linearly separable?
- Questions: How many layers? How many neurons per layers?
- Can # layers/# neurons per layer be *learned* too? (in addition to weights)

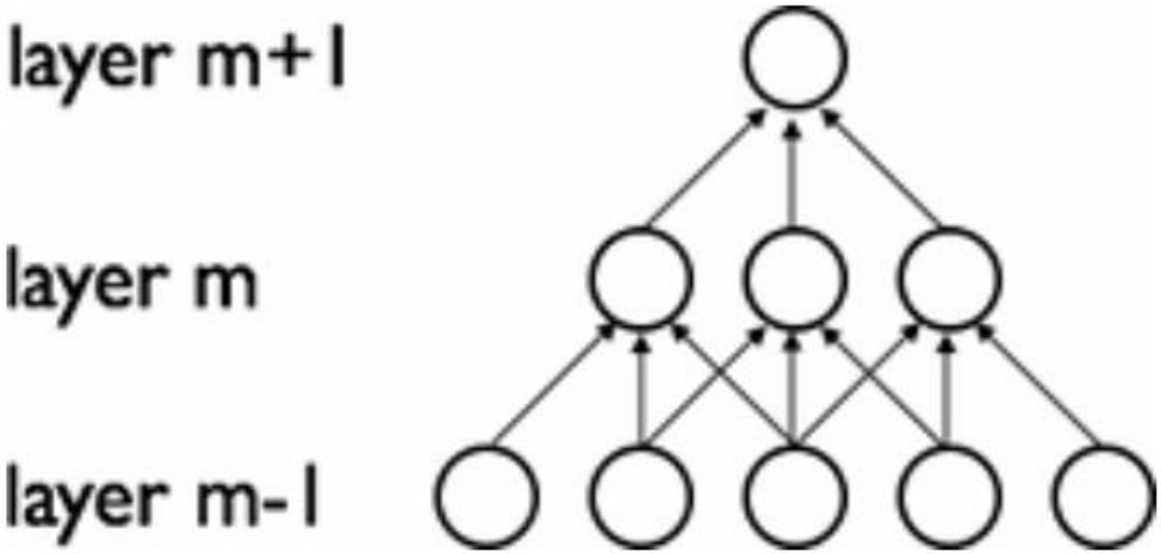
Easier Said than Done

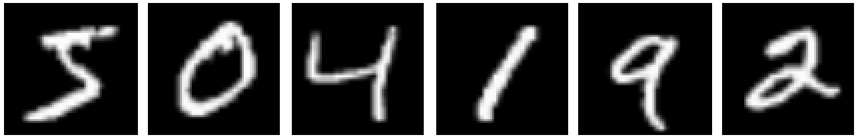
- Blind learning with large number of parameters is numerically impossible
- Major recent advance
  - Reduced number of parameters
  - Layered learning



Emulation of Human Vision

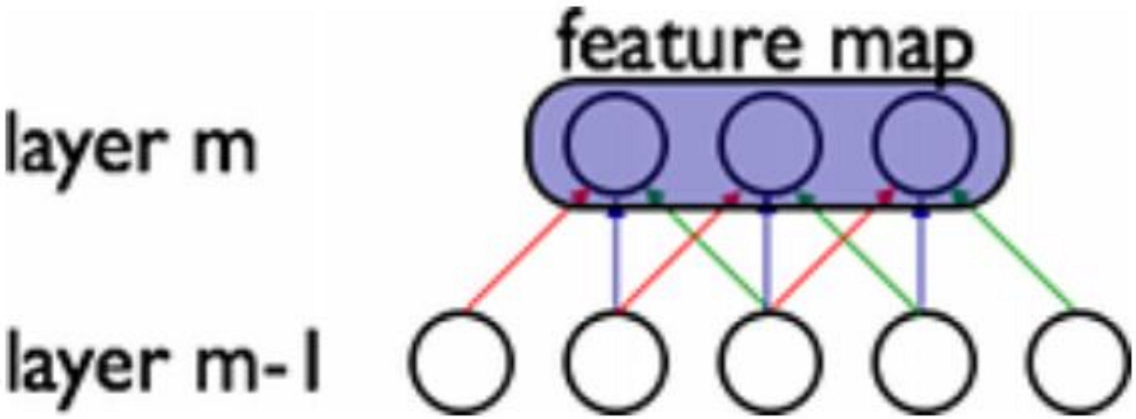
- Sparsity of connection

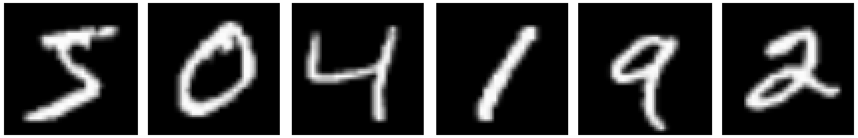




Emulation of Human Vision

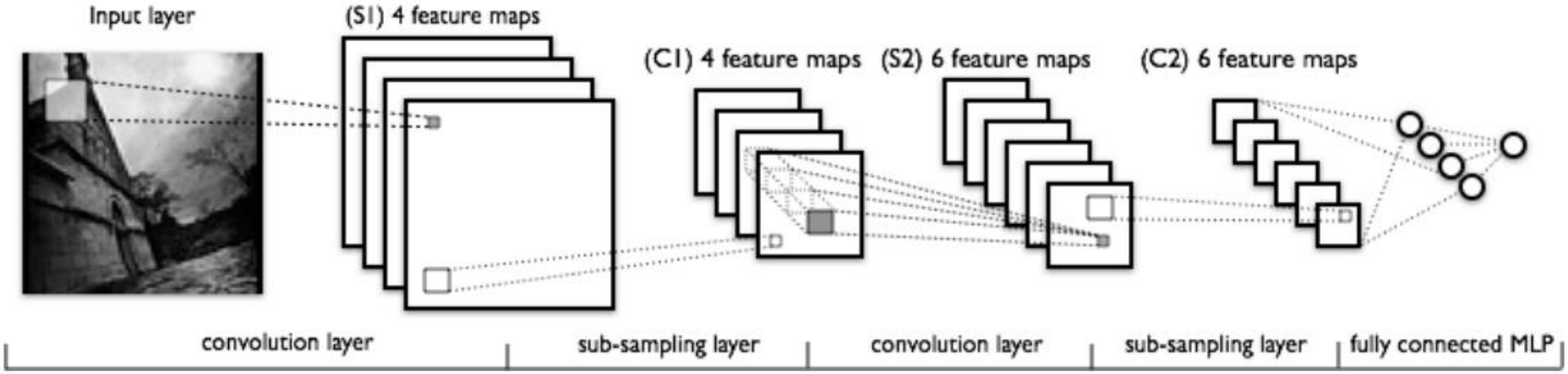
- Shared weight





Layered Learning

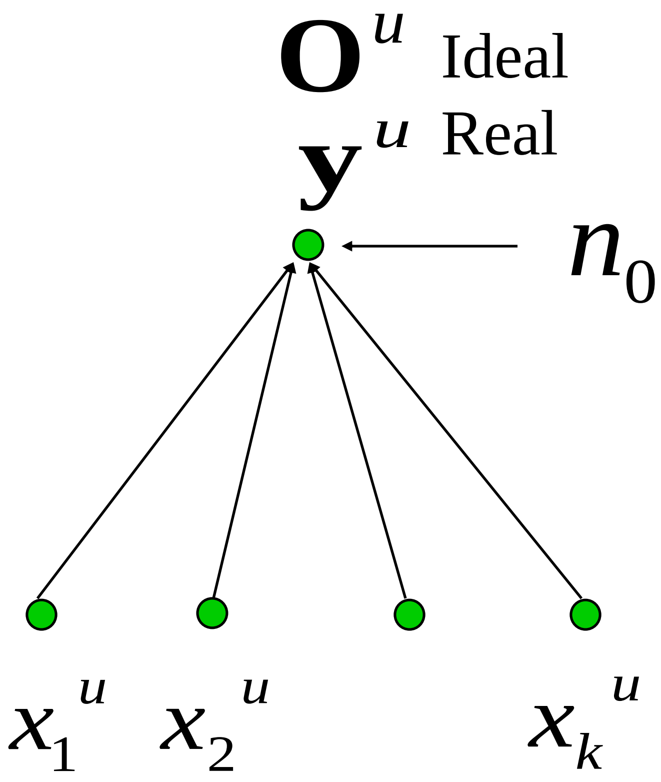
- A hierarchical "feature descriptor"
- Learning automatically from input data
- Layer-by-layer learning with auto encoder
- Partition:
  - CNN: feature detection
  - Fully-connected network: recognition


Input layer
(S1) 4 feature maps
(C1) 4 feature maps
(S2) 6 feature maps
(C2) 6 feature maps
convolution layer
sub-sampling layer
convolution layer
sub-sampling layer
fully connected MLP

Adaptive Networks

- Network size/layer is not fixed initially
- Layer/size are added when necessary (or when a large number of epochs progress without finding suitable weights)
- Assumptions:
  - two classes (1,0)
  - may not be linearly separable (e.g., multiple concave regions)

Initially one neuron



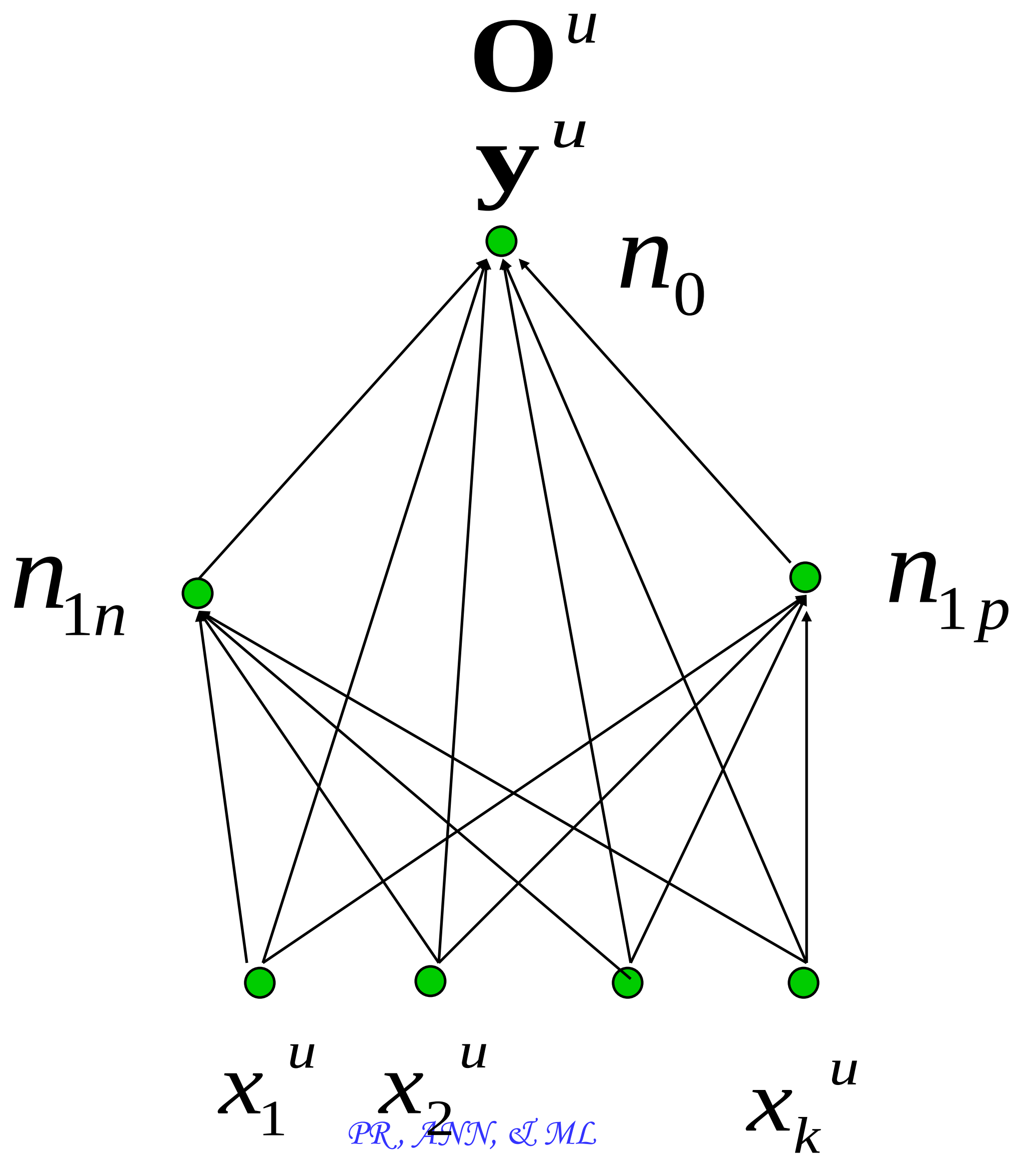
wrongly on $\quad O^u = 0 \quad y^u = 1$

wrongly off $\quad O^u = 1 \quad y^u = 0$

Refinement with more neurons

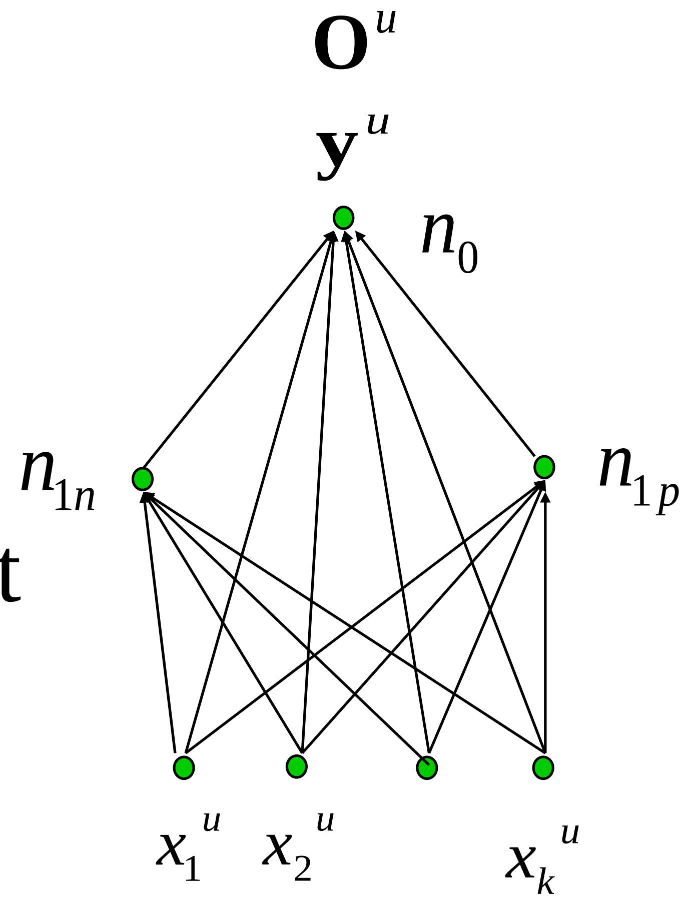
- Train through a number of epochs
- if no wrongly on/off cases, the two classes are linearly separable, stop
- if there are wrongly on/off cases, the two classes are not linearly separable, then
  - remember the best weights (the weights that cause the less number of misclassification)
  - introduce more units (instead of throwing away everything and restarting from scratch with a larger network)

Increase Network Complexity



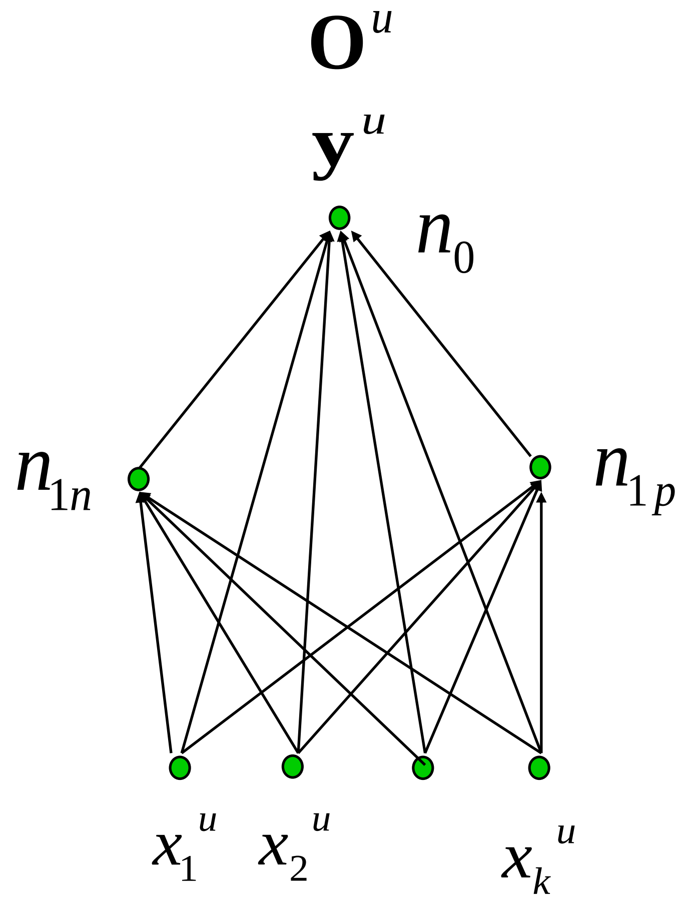
N_{1n}: correct wrongly-on error fire negative feedback only

O	y	*action*
0	0	don't care
1	1	*off*
0	1	large negative output
1	0	*off*

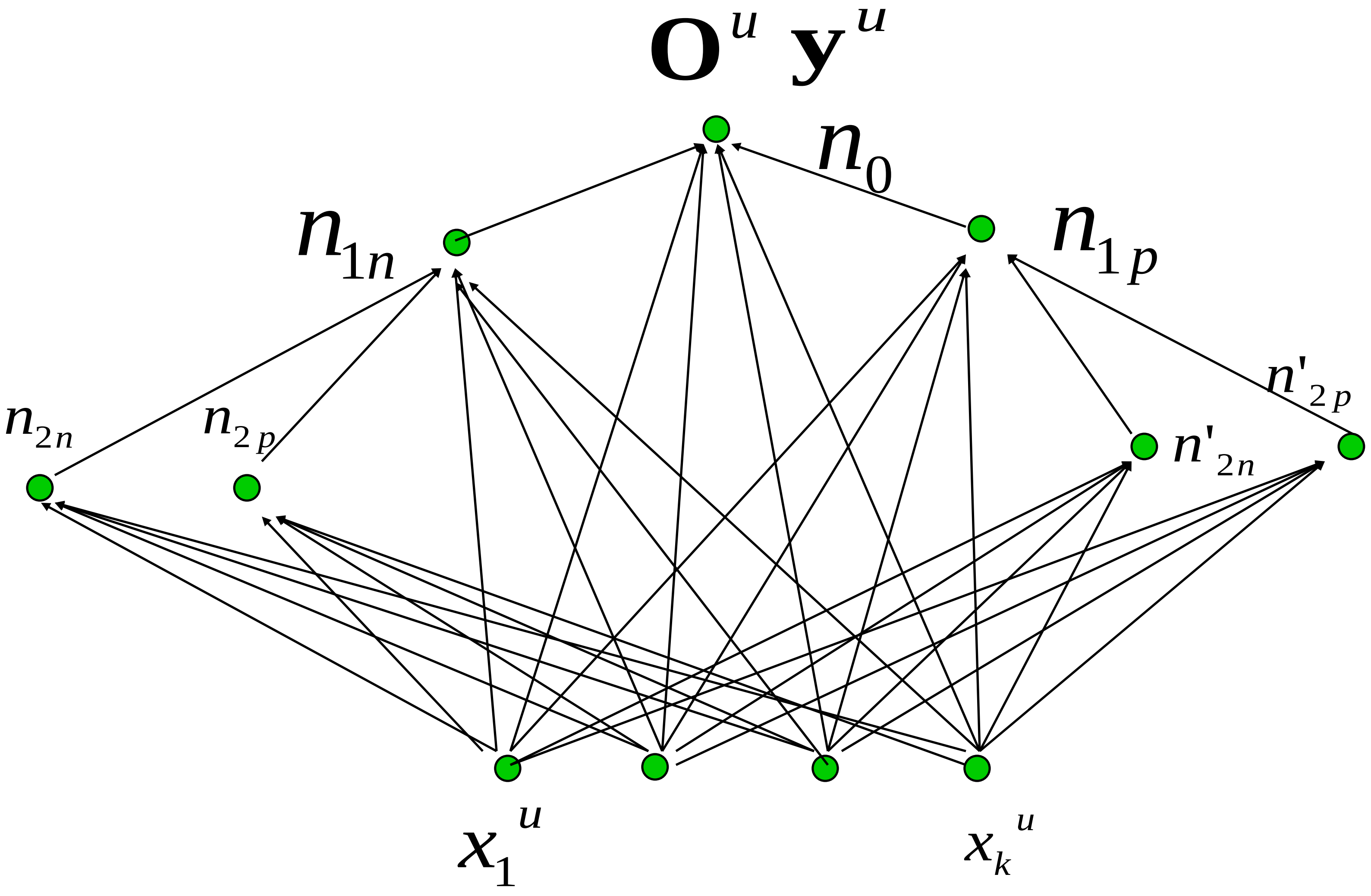


N_{1p}: correct wrongly-off error fire positive feedback only

O	y	*action*
0	0	off
1	1	don' t care
0	1	*off*
1	0	large positive output



Further Refinement

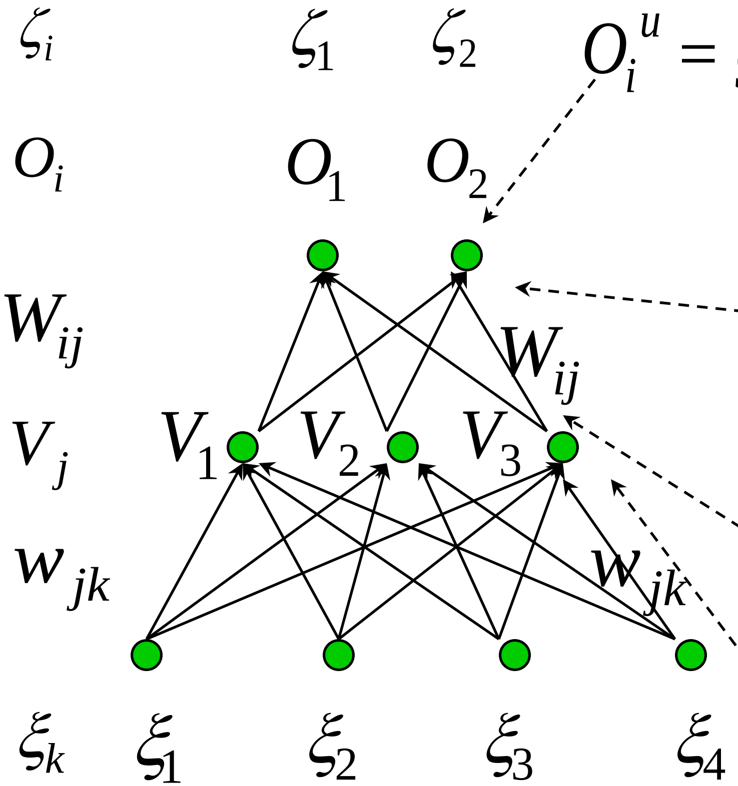


This image cannot currently be displayed.

General Learning Rule

- N_{xn}
  - Fire negative impulse
  - Correct wrongly on cases
  - Turn off if O=1 (no matter what y is)
  - Don't care if O=0 and y=0
- N_{xp}
  - Fire positive impulse
  - Correct wrongly off cases
  - Turn off if O=0 (no matter what y is)
  - Don't care if O=1 and y=1

Backpropagation Learning rule



$$O_i^u = g(h_i^u) = g\left(\sum_j W_{ij} V_j^u\right) = g\left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} \xi_k^u\right)\right)$$

$$h_i^u = \sum_j W_{ij} V_j^u = \sum_j W_{ij}\, g\left(\sum_k w_{jk} \xi_k^u\right)$$

$$V_j^u = g(h_j^u) = g\left(\sum_k w_{jk} \xi_k^u\right)$$

$$h_j^u = \sum_k w_{jk} \xi_k^u$$

Change w.r.t. w_ij

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = -\eta \frac{\partial(\zeta_i^u - g(\sum_j W_{ij} V_j^u))^2}{\partial W_{ij}}$$

$$= \eta \sum_u (\zeta_i^u - O_i^u) g'(h_i^u) V_j^u$$

$$= \eta \sum_u \delta_i^u V_j^u \qquad \delta_i^u = (\zeta_i^u - O_i^u) g'(h_i^u)$$

This image cannot currently be displayed.

Change w.r.t. w_ij

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \frac{\partial \sum_{u,i} (\zeta_i^{u} - g(\sum_j W_{ij} g(\sum_k w_{jk} \xi_k^{u}))^2}{\partial w_{jk}}$$

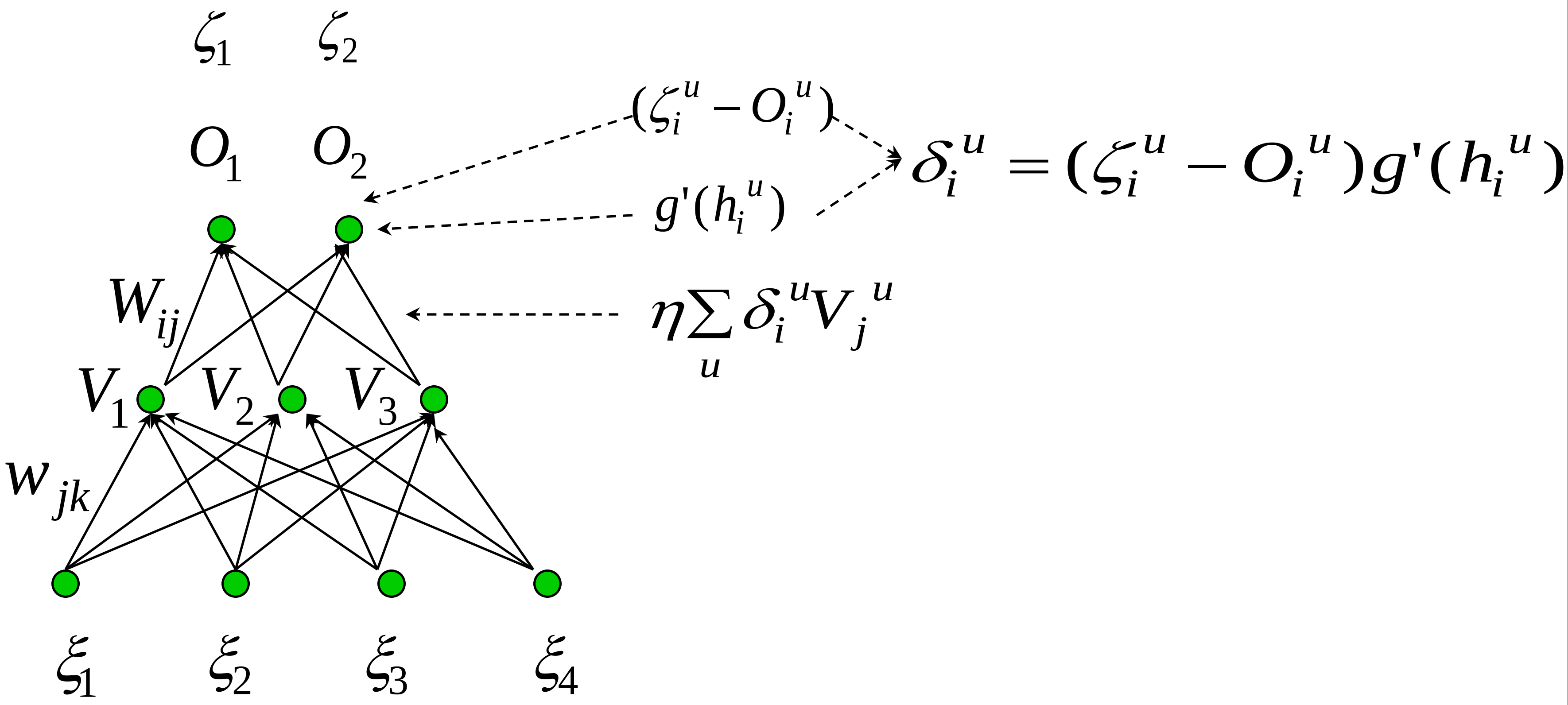
$$= -\eta \frac{\partial E}{\partial V_j^{u}} \frac{\partial V_j^{u}}{\partial w_{jk}}$$

$$= \eta \sum_{u,i} (\zeta_i^{u} - O_i^{u}) g'(h_i^{u}) W_{ij} g'(h_j^{u}) \xi_k^{u}$$

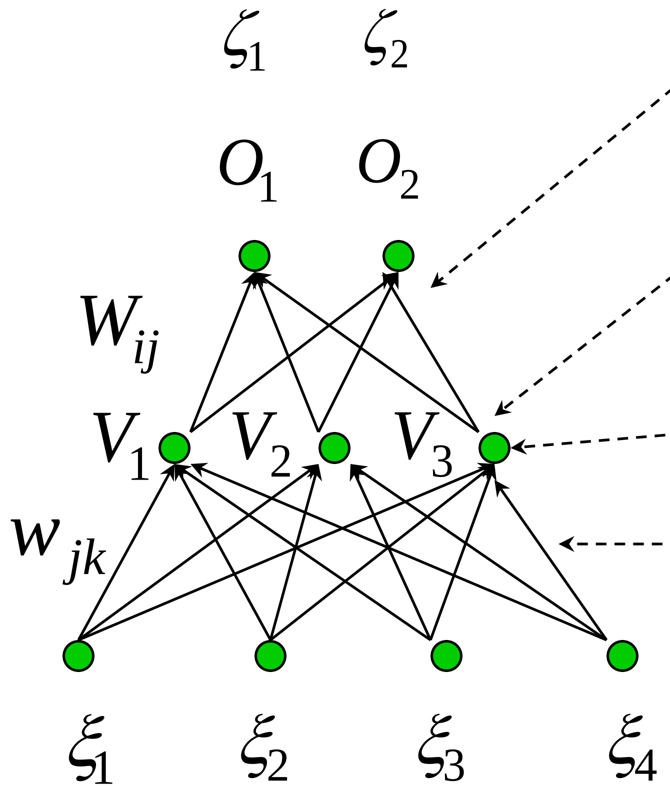
$$= \eta \sum_{u,i} \delta_i^{u} W_{ij}^{u} g'(h_j^{u}) \xi_k^{u}$$

$$= \eta \sum_{u} \delta_j^{u} \xi_k^{u} \qquad \delta_j^{u} = g'(h_j^{u}) \sum_i \delta_i^{u} W_{ij}$$

Interpretation



Interpretation (cont.)



$$\delta_i^u = (\zeta_i^u - O_i^u) g'(h_i^u)$$

$$\sum_i \delta_i^u W_{ij}$$

$$g'(h_j^u)$$

$$\delta_j^u = g'(h_j^u) \sum_i \delta_i^u W_{ij}$$

$$\eta \sum_u \delta_j^u \xi_k^u$$