SMLO 5: Connectionist Methods

Doug Aberdeen, October 19, 2004

National ICT Australia

NATIONAL ICT AUSTRALIA

Other Perceptron Rules

- Vector training pairs $\{\mathbf{x} = [x_1, \dots x_I], \mathbf{t} = [t_1, \dots, t_J]\}$
- t_j is target label for output y_j
- y_j is actual output k
- Assume that last $x_I = 1$, and last $w_{Ij} = b_j$ (bias term)

while Improvement **do**
  for each training point $\{\mathbf{x}, \mathbf{t}\}$ **do**
    Feed $\mathbf{x}$ forward to compute $\mathbf{y}$
    Compute weight update Δ_w using rules
    $\mathbf{w}' = \mathbf{w} + \alpha\Delta_w$
  end for
end while

NATIONAL ICT AUSTRALIA

Rules cont.

Apply one of the following rules to each training example

Name	Δ_w for w_{ij}	Comment
Widrow-Hoff	$(t_j - y_j)x_i$	Direct gradient descent
Delta	$(t_j - y_j)$ $\sigma'(\mathbf{xw}_j)x_i$	includes squashing function
Correlation	$t_j x_i$	Updates promote correlation
Outstar	$(y_j - w_{ij})x_i$	Move weights to match output (c)
Hebbian	$y_j x_i$	reinforcement (c)
WTA	$(x_i - w_{ij})$	Move only winner weights (c)

Multiply each rule by step size α to get weight increment

WTA=winner take all (Self Organising Feature Map)

Fisher Discriminant Function

- $J(\mathbf{w}) = \frac{\mathbf{w}^\top S_B \mathbf{w}}{\mathbf{w}^\top S_W \mathbf{w}}$
- S_B is class divergence

$$S_B = (\mathbf{c}_1 - \mathbf{c}_2)(\mathbf{c}_1 - \mathbf{c}_2)^\top$$

- S_W is variance in samples

$$S_W = \sum_{\{x:t=1\}} (\mathbf{x} - \mathbf{c}_1)(\mathbf{x} - \mathbf{c}_1)^\top + \sum_{\{x:t=2\}} (\mathbf{x} - \mathbf{c}_2)(\mathbf{x} - \mathbf{c}_2)^\top$$

- Maximising $J(\mathbf{w})$ minimises classification error
- Closed form $\mathbf{w}^* = S_W^{-1}(\mathbf{c}_1 - \mathbf{c}_2)$

NATIONAL ICT AUSTRALIA

Fisher Cont

- Works for non-separable data
- Interpretation: project all data onto a line
- Optimal for Gaussian classes of equal co-variance
- Often works well despite assumptions violated
- Dot product form makes *kernel trick* easy

$$\mathbf{x}^\top S \mathbf{x} = \; < \Phi(\mathbf{x}), \Phi(\mathbf{x}') > \; = k(\mathbf{x}, \mathbf{x}')$$

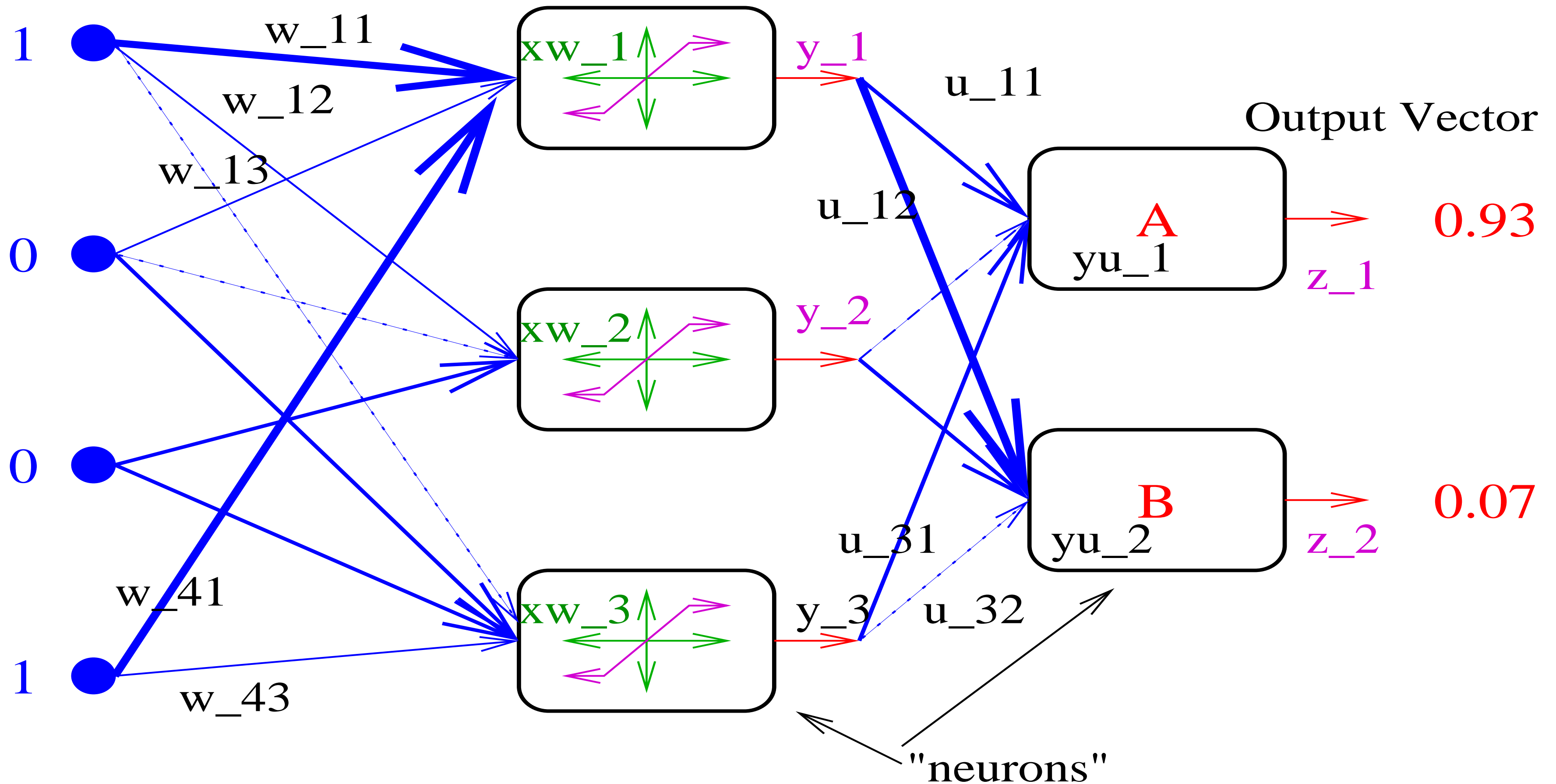
NATIONAL ICT AUSTRALIA

Multi-Class Data

- t is no longer a binary vector. C classes.
- Train C binary perceptrons with $t =$me and $t =$other:
  - winning class has maximum output
- Train all $\frac{C(C-1)}{2}$ binary classifiers
  - Use voting amongst classes
- Train perceptron with one output per class:
  - t is now a binary vector with 1 for true class
  - winning class is index of largest output node

NATIONAL ICT AUSTRALIA

Multi-Layer Perceptron



NATIONAL ICT AUSTRALIA

Multi-Layer Perceptron

- Outputs of individual perceptrons are fed into new perceptrons
- First layer of perceptrons is called “hidden layer”
- Two layers with non-linear squashing is sufficient to represent *any* continuous function arbitrarily closely (bad?)
- Choosing number of hidden units J is black-art
- Typical squashing function $y_j = \sigma(\mathbf{x}\mathbf{w}_j) = \frac{1}{1+exp(-\mathbf{x}\mathbf{w}_j)}$
- Output can be single real number (regression)
- Can be multiple outputs (now $\mathbf{z}$). Index of max output represents class

NATIONAL ICT AUSTRALIA

Error Back-Propagation Algorithm

- Essentially gradient descent over multiple layers
- Error at output nodes $E(\mathbf{x}, \mathbf{t}, \mathbf{w}, \mathbf{u}) = 0.5 \sum_{k=1}^{K} (z_k - t_k)^2$
- Compute gradient of E with respect to all w_{ij}, u_{jk}
- Sum gradients over samples, then update: *batch mode*
  - Use matrix-matrix multiply to speed up batch mode
- Update parameters on each sample: *online*

NATIONAL ICT AUSTRALIA

Multi-Layer Algorithm: Batch

1: **while** Error on withheld training data improves **do**
2: $\Delta_{WU} = 0$
3: $E = 0$
4: **for** each sample $\{\mathbf{x}, \mathbf{t}\}$ **do**
5: $\mathbf{y} = \sigma(\mathbf{x}W)$
6: $\mathbf{z} = \mathbf{y}U$
7: $E' = E + 0.5 \sum_k (t_k - z_k)^2$
8: $\Delta_{WU} = \Delta_{WU} + ErrorBackProb(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{t}, W, U)$
9: **end for**
10: $W' = W - \alpha\Delta_{WU}$ (or smarter grad descent)
11: **end while**

NATIONAL ICT AUSTRALIA

ErrorBackProb$(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{t}, W, U)$

Want to compute $\frac{\partial E}{\partial W}$ and $\frac{\partial E}{\partial U}$
Use repeated application of chain rule. To start

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

$$\frac{\partial z_k}{\partial u_{jk}} = \frac{\partial}{\partial u_{jk}} \sum_{j'=1}^{J} y_{j'} u_{j'k}$$

$$= y_j$$

$$\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial u_{jk}}$$

$$= y_j (t_k - z_k)$$

Looks like Widrow-Hoff Rule!

NATIONAL ICT AUSTRALIA

Cont...

Now propagate to hidden layer weights W

Let activation $a_j = \sum_{i'=1}^{I} x_{i'} w_{i'j}$

$$
\begin{aligned}
\frac{\partial z_k}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \sum_{j'=1}^{J} y_{j'} u_{j'k} \\
&= \frac{\partial}{\partial w_{ij}} y_j u_{jk} \\
&= \frac{\partial}{\partial w_{ij}} \sigma(a_j) u_{jk} \\
&= \frac{\partial \sigma}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} u_{jk}
\end{aligned}
$$

NATIONAL ICT AUSTRALIA

Cont...

$$\frac{\partial z_k}{\partial w_{ij}} = \sigma(a_j)(1-\sigma(a_j))\frac{\partial a_j}{\partial w_{ij}} u_{jk}$$

$$= \sigma(a_j)(1-\sigma(a_j))\left(\frac{\partial}{\partial w_{ij}}\sum_{i'=1}^{I} x_{i'}w_{i'j}\right) u_{jk}$$

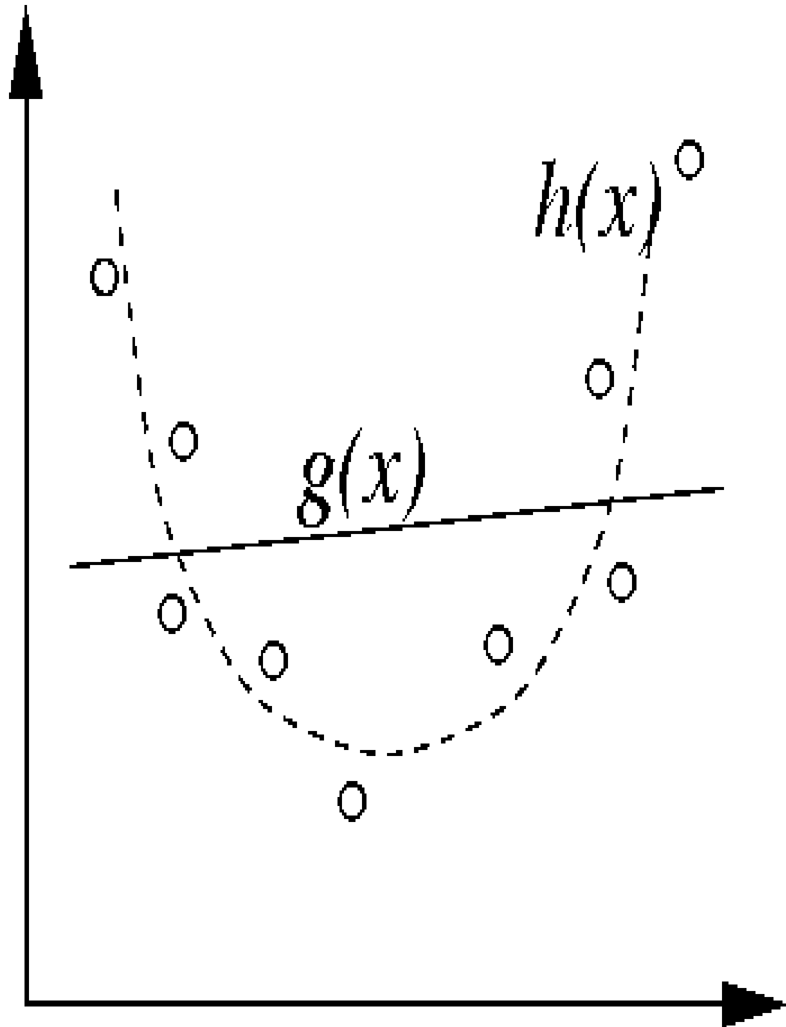
$$= \sigma(a_j)(1-\sigma(a_j))x_i u_{jk}$$

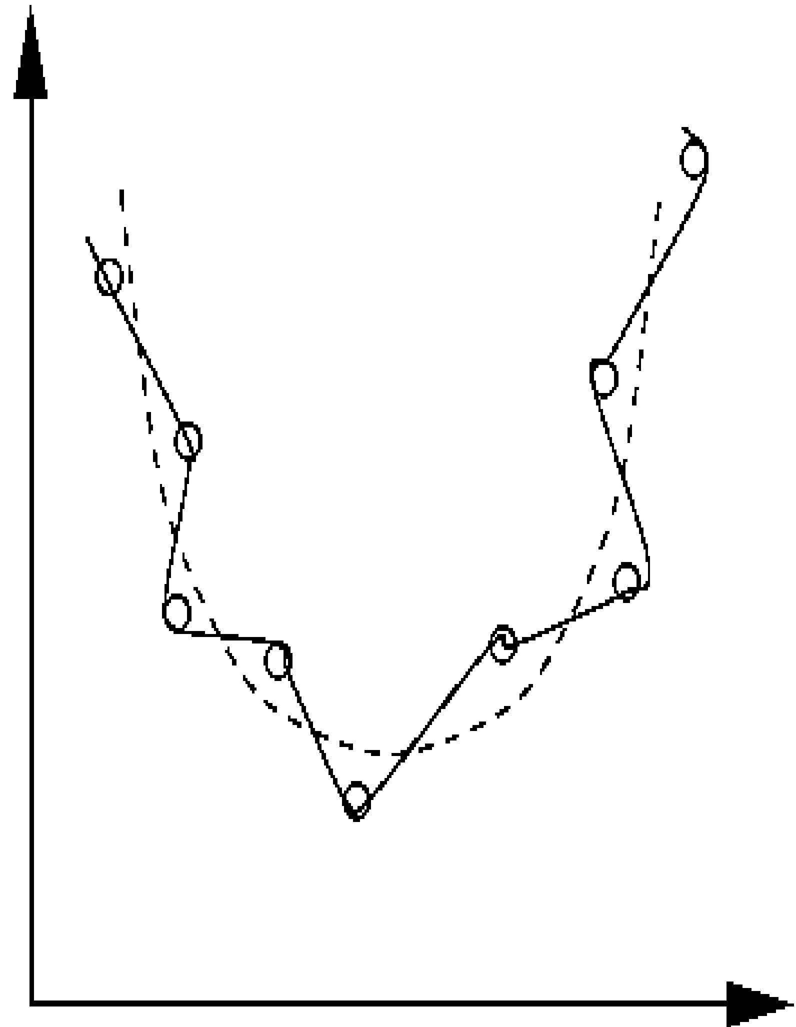
$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^{K} \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial w_{ij}}$$

$$= \sum_{k=1}^{K}(t_k - z_k)\sigma(a_j)(1-\sigma(a_j))x_i u_{jk}$$

NATIONAL ICT AUSTRALIA

Over fitting!





NATIONAL ICT AUSTRALIA

Over fitting Solutions

- Problem in words: function matches data too closely
- Fails to generalise to new points
- General machine learning problem!
- Solution? Regularise!
  - Regularisation means smoothing or simplifying function:
  - ...penalise large weights
  - ...reduce number of hidden units
- Deliberately add noisy training points
- Early stopping...

NATIONAL
ICT AUSTRALIA

Early Stopping

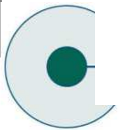
Error

validation

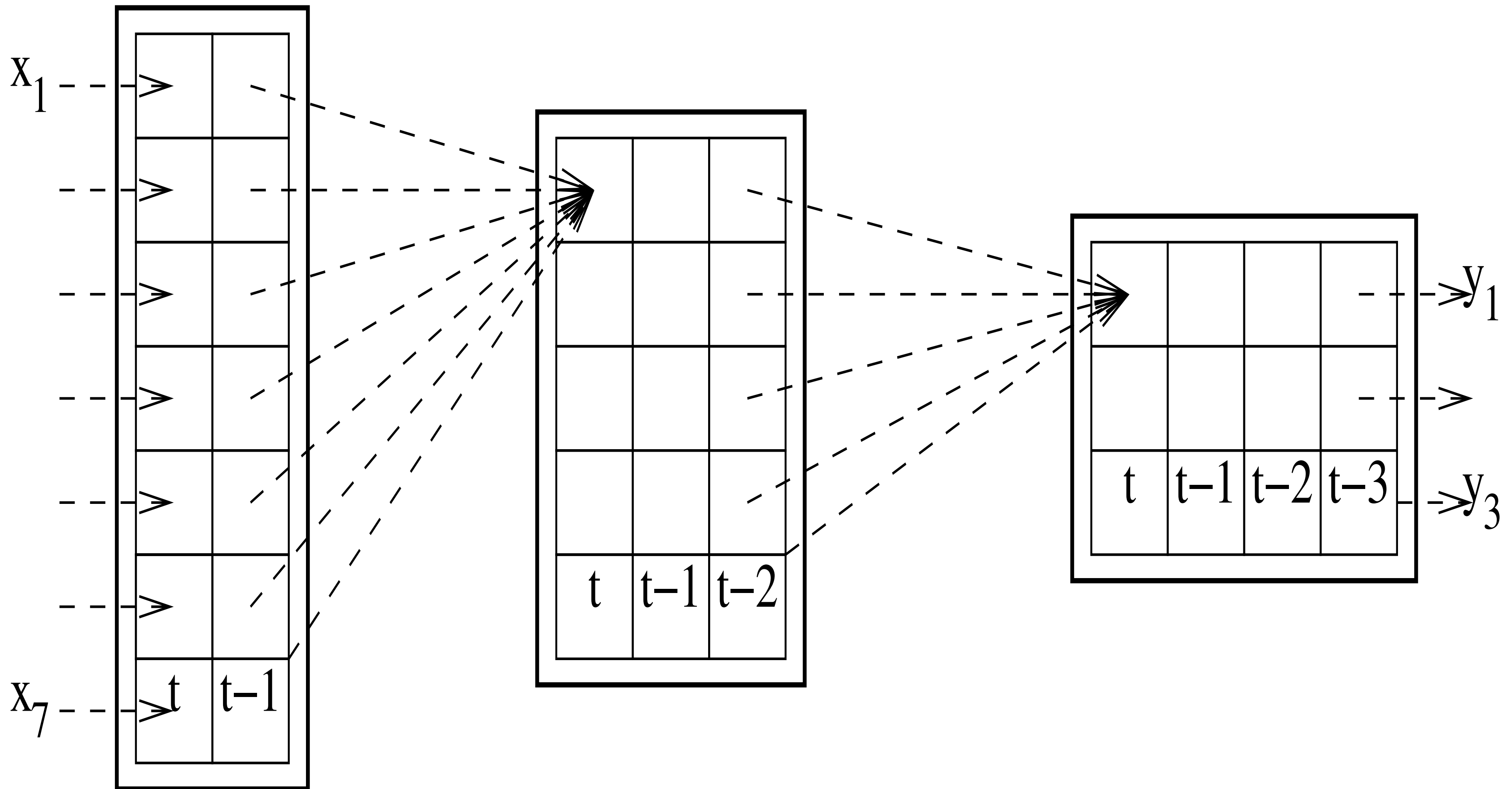
training

t

Time

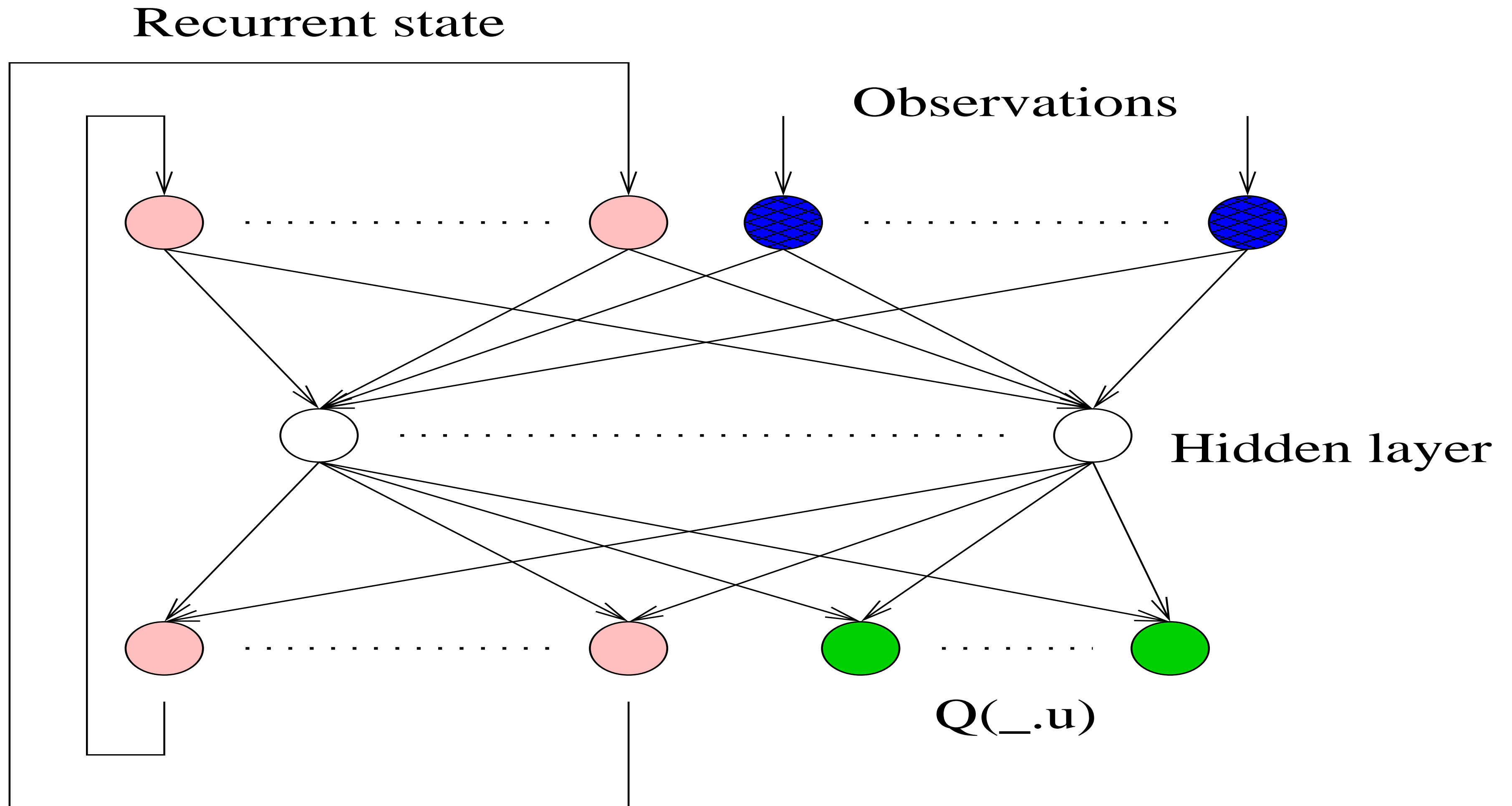


Time Series (TDNN)



NATIONAL ICT AUSTRALIA

Recurrent Neural Net



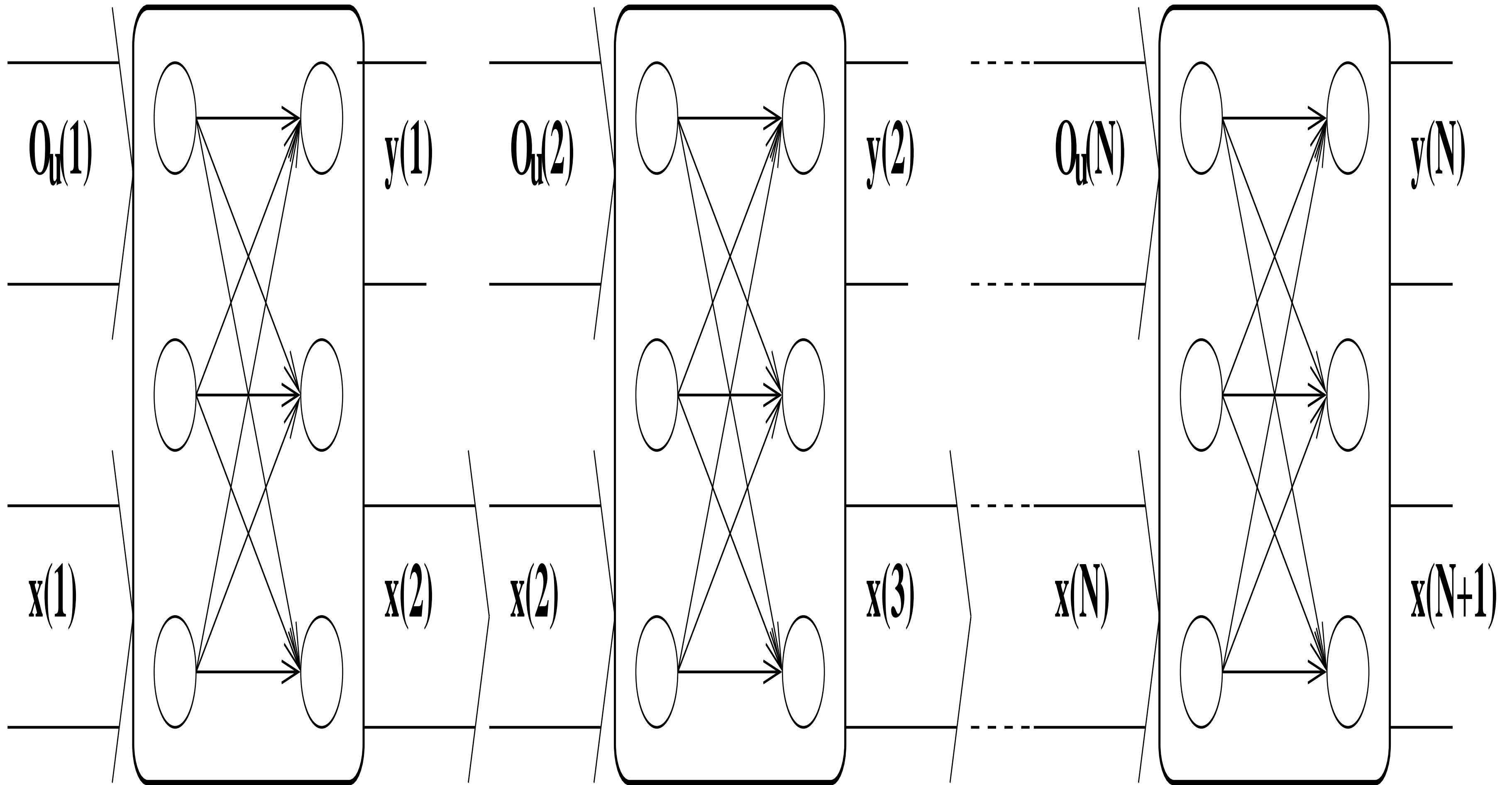
NATIONAL
ICT AUSTRALIA

Training RNNs

- Idea: unfold net over time and use back-prop
- State error = 0 at end
- Propagated errors tend to explode or vanish
- Trained memory only as far as unfolded
- Might work for longer time scales though
- Better algorithms now exist

NATIONAL ICT AUSTRALIA

Unfolding RNNs



NATIONAL ICT AUSTRALIA

Self Organising Feature Maps

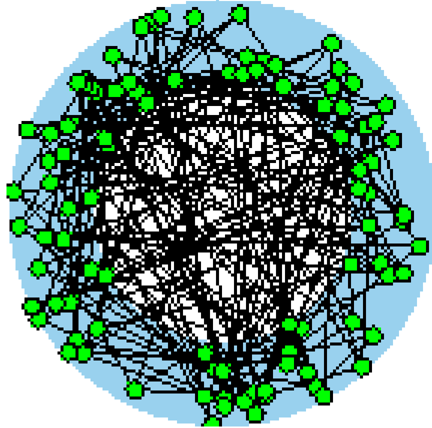
- Neural Net Clustering
- Similar to k-means
- Number of net outputs K equivalent to choice of k
- Outputs are conceptually linked in a graph
- Output neighbours computed from graph links
- Use Winner Takes All update rule
- Has cool ability to create maps with meaning

NATIONAL ICT AUSTRALIA

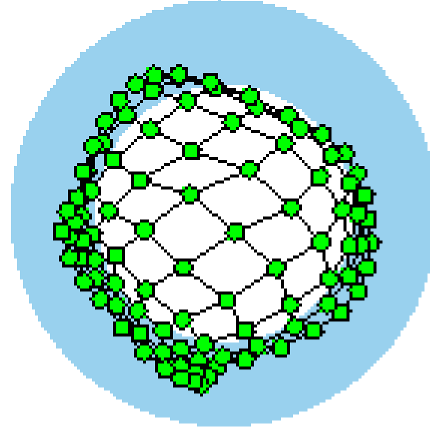
SOFM Algorithm

1: $N =$ initial neighbourhood size
2: **while** Centroids still changing **do**
3:     **for** Each data point $\mathbf{x}$ **do**
4:         Winner $k^* = \arg\max_k \mathbf{x}^\top \mathbf{w}_k$
5:         **for** Each set of weights w_k **do**
6:             **if** $d(k, k^*) < N$ **then**
7:                 Update $\mathbf{w}'_k = \mathbf{w}_k + \alpha(\mathbf{x} - \mathbf{w}_k)$
8:             **end if**
9:         **end for**
10:     **end for**
11: **end while**

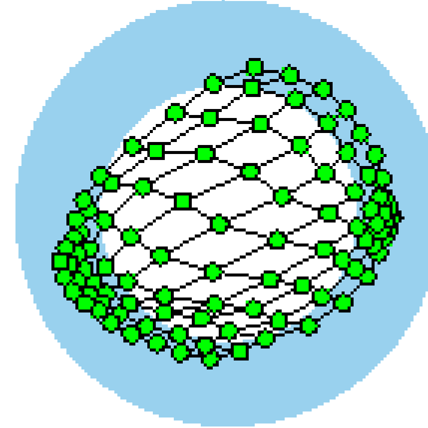
SOFM Example



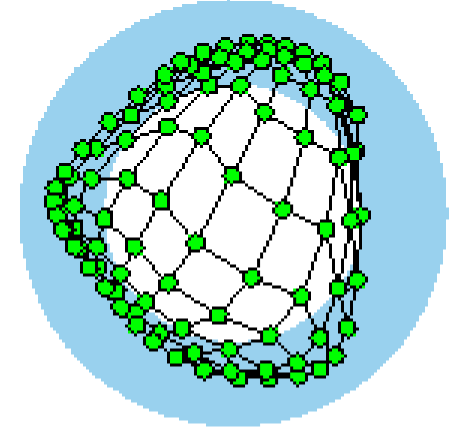
a) 0 signals



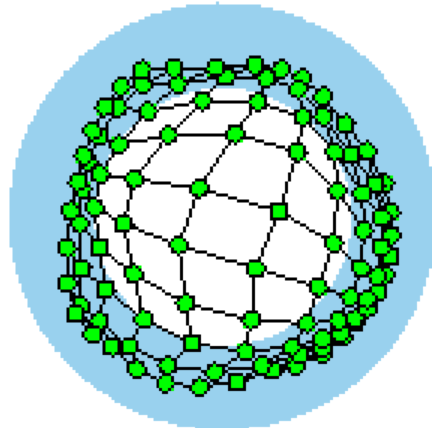
b) 100 signals



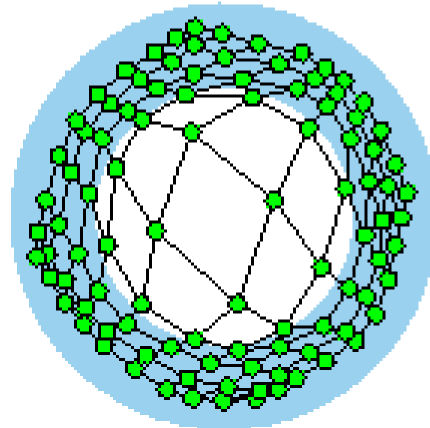
c) 300 signals



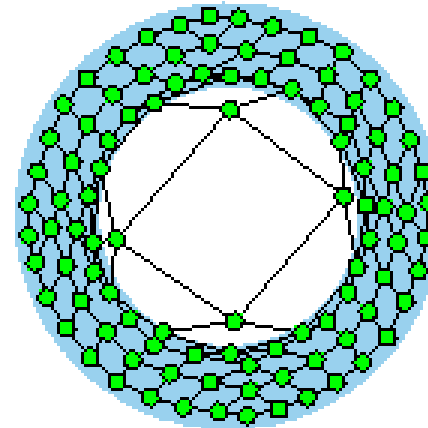
d) 1000 signals



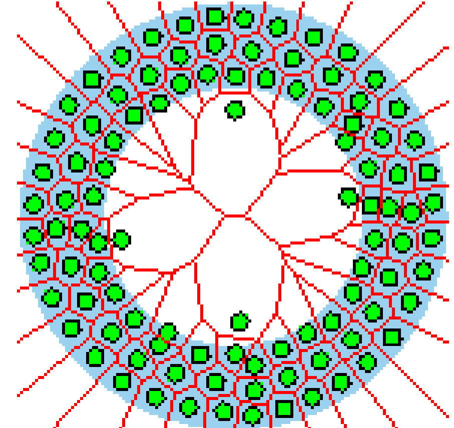
e) 2500 signals



f) 10000 signals



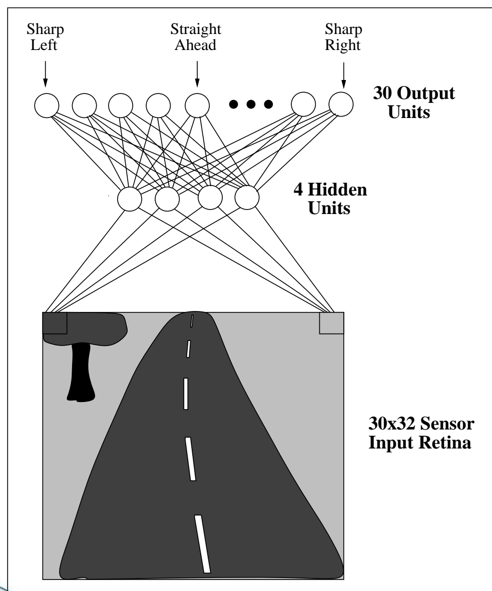
g) 40000 signals

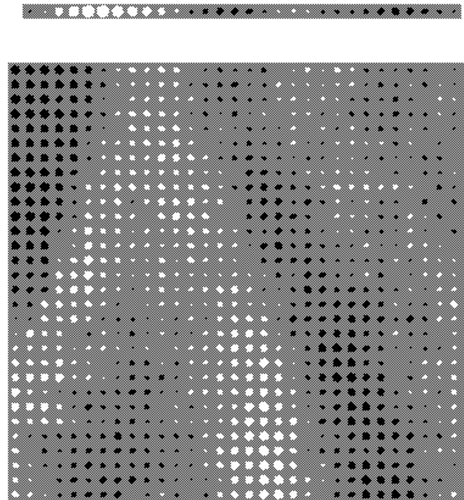


h) Voronoi regions

NATIONAL ICT AUSTRALIA

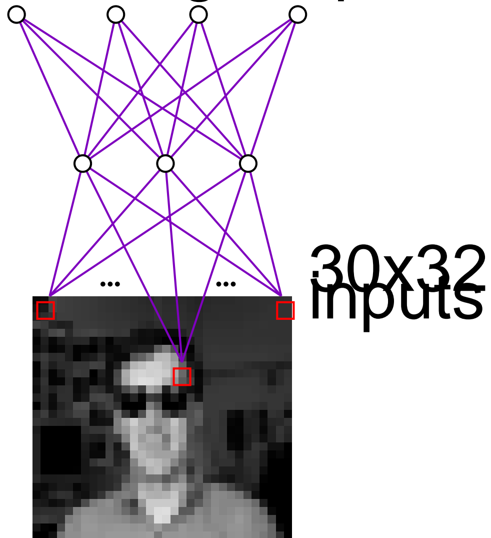
Alvinn





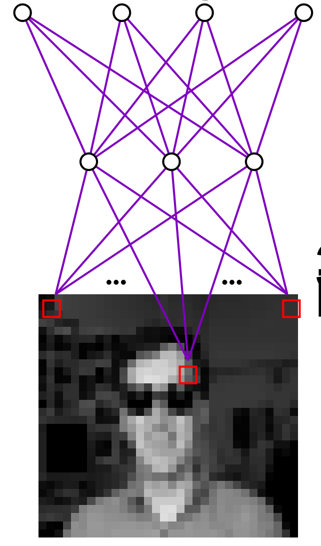
NATIONAL
ICT AUSTRALIA

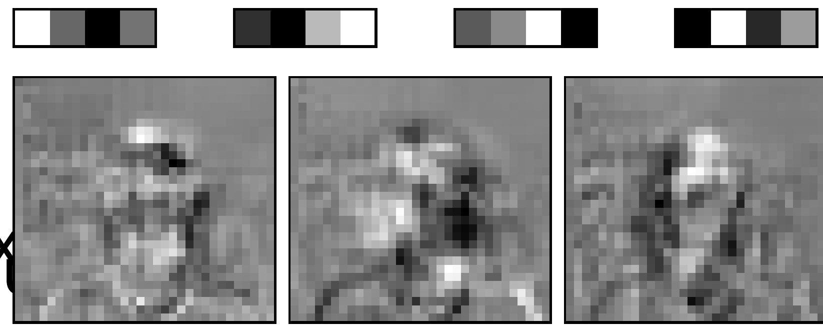
Face recognition



90% accurate learning head pose, and recognizing 1-of-20 faces

Weights





Typical input images