

Ch4. Artificial Neural Networks

4.1 - 4.7

S. Visa

Biological neural systems

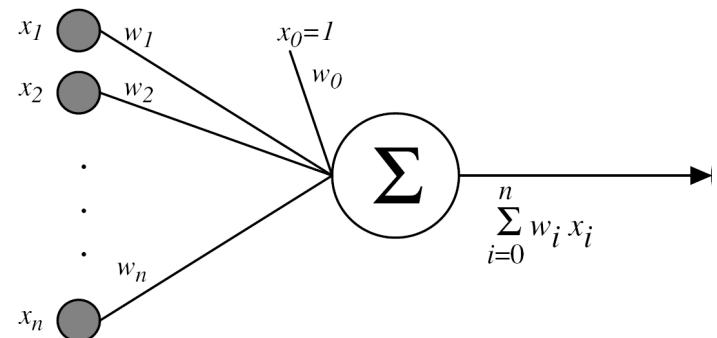
- Each neuron has
 - Soma = cell body
 - Dendrites = multiple inputs
 - Axon = output
- Synapse
 - Connects an axon to a dendrite
 - Might increase (excite) or decrease (inhibit) a signal
 - When input signal sufficiently strong → neuron fires (= propagates signal)
- No. of neurons in human brain $\sim 10^{10}$
- Connections per neuron $\sim 10^4$
- Face recognition ~ 0.1 sec
- Neuron switching time $\sim 10^{-3}$ sec
- Highly parallel & distributed processing



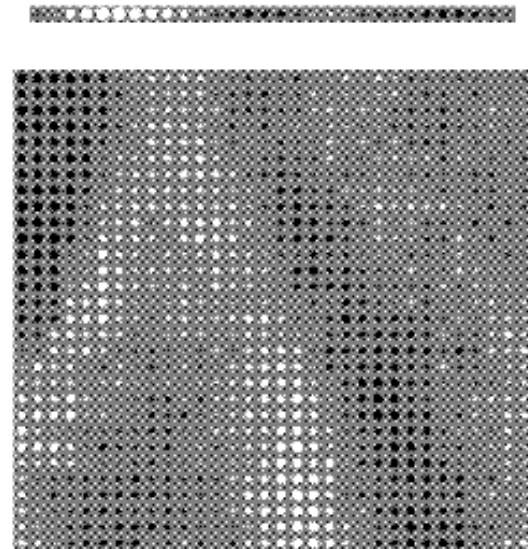
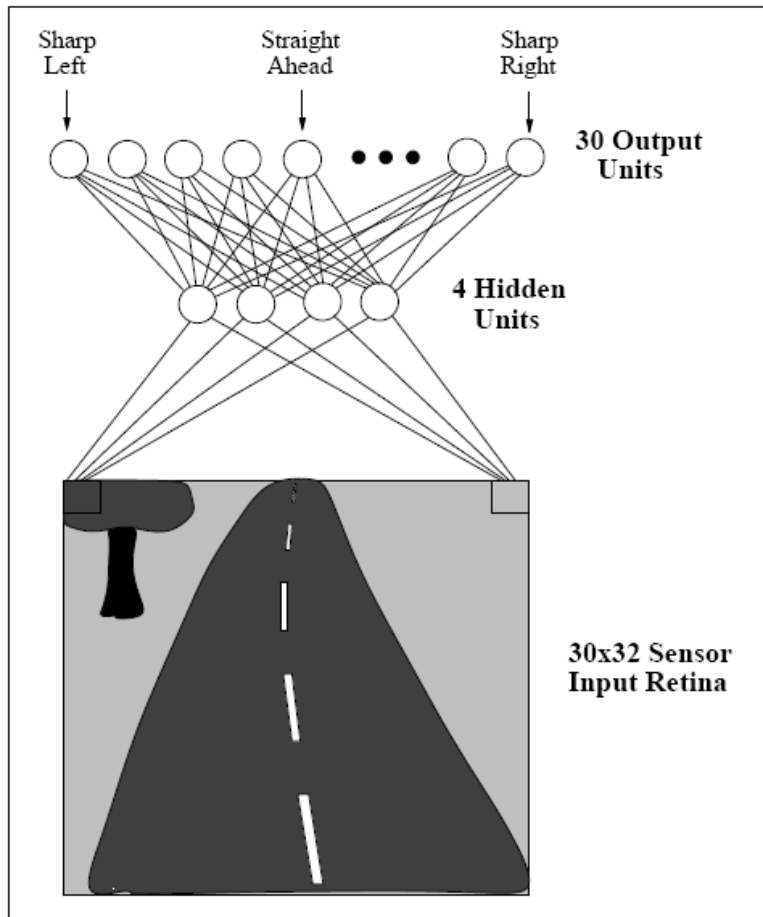
Artificial neural networks (ANN)

- Consist of
 - Units
 - Connections
 - Weights
- Learn to associate inputs to outputs by tuning the weights
- E.g.
 - Input = pixels of photo
 - Output = classification of photo (landscape?, car?,...)
- Highly parallel & distributed processing

Biological NN	Artificial NN
Soma	Unit
Axon, dendrite	Connection
Synapses	Weights
Threshold	Bias
Signal	Activation function



Ex. – ALVINN [Pomerleau 1989] drives 70mph on highway

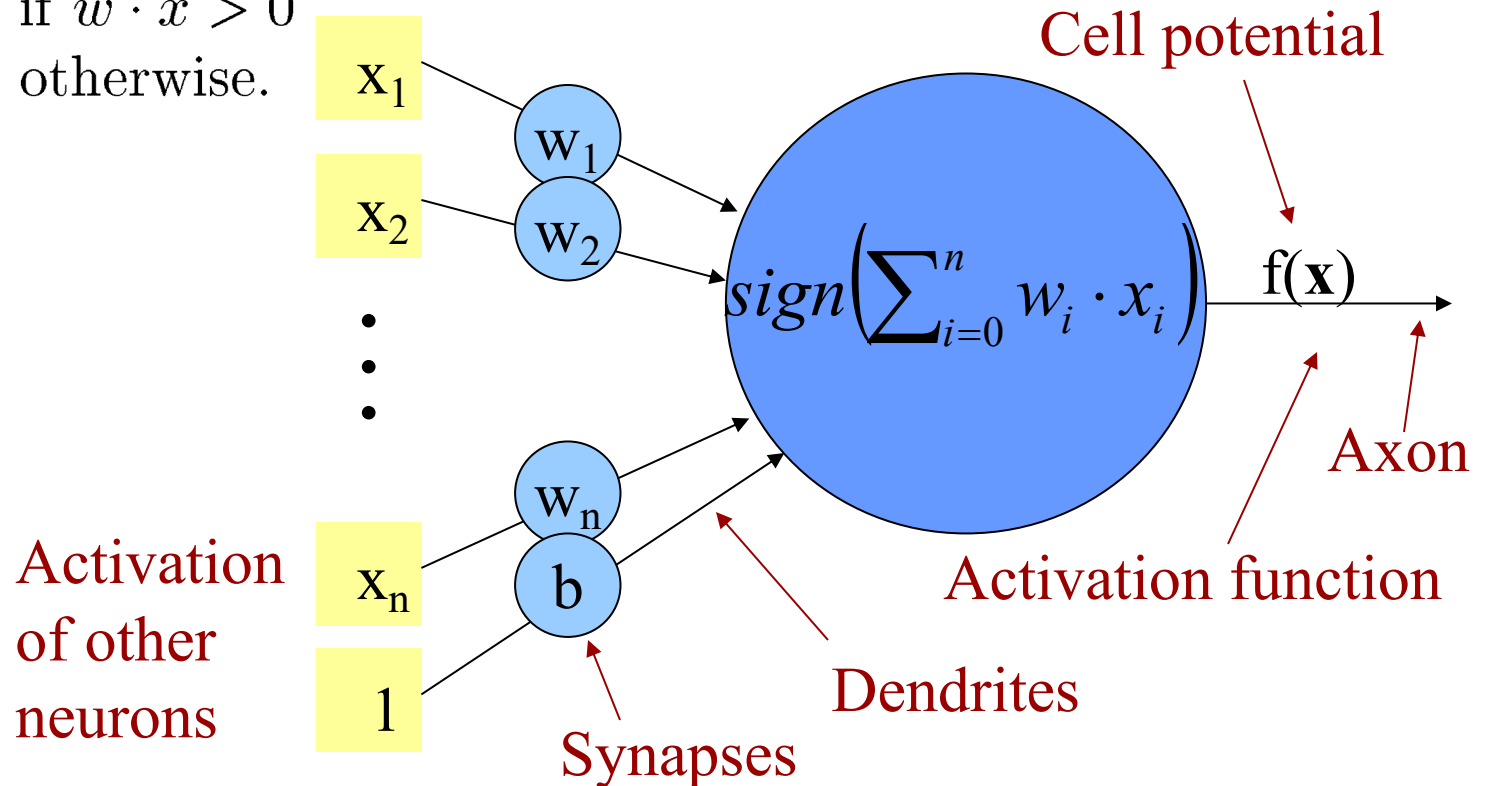


Autonomous Land Vehicle in a Neural Net

Perceptron

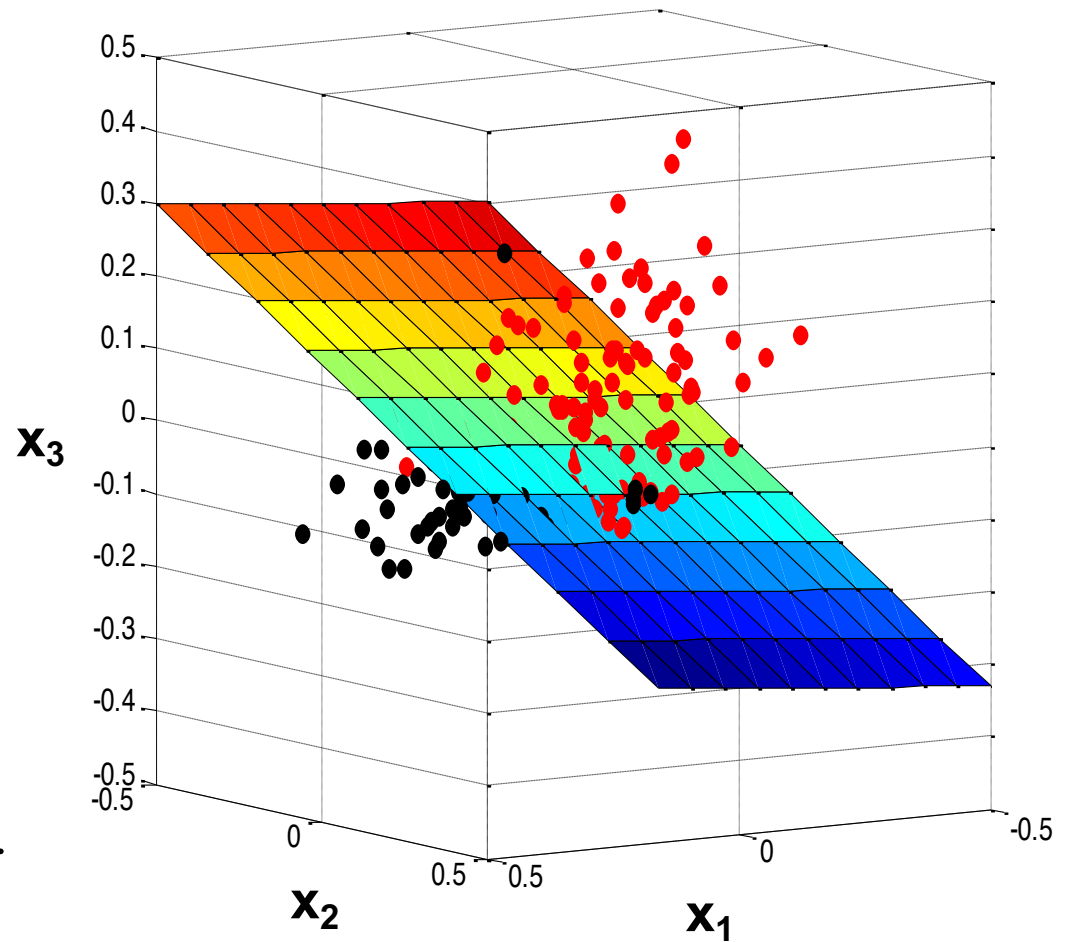
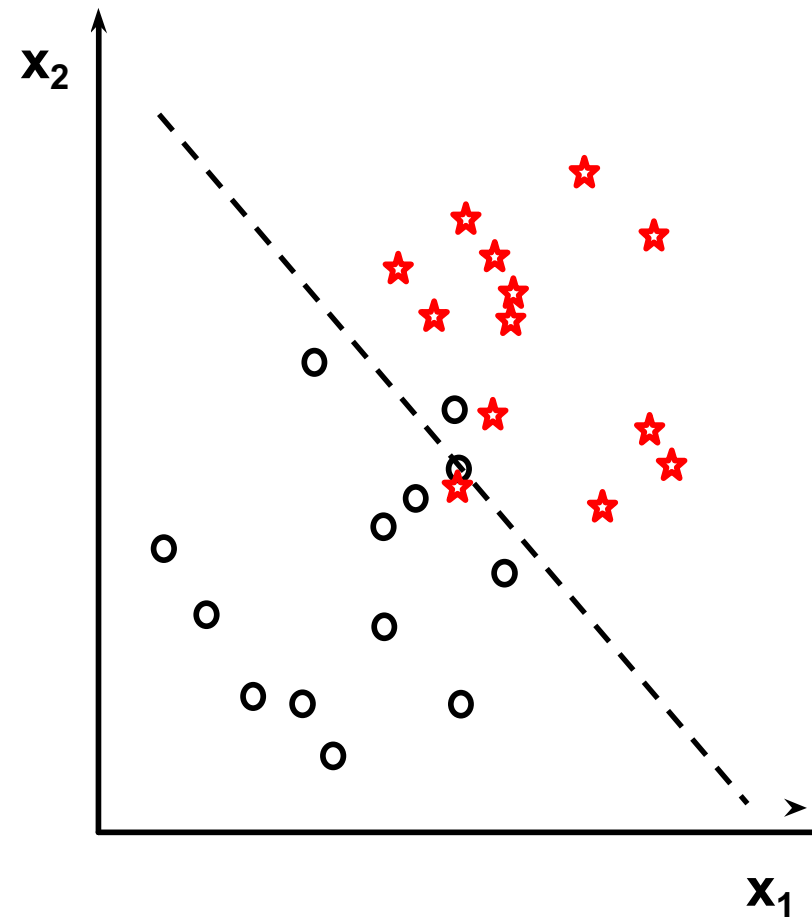
- Simplest NN \rightarrow simulates 1 neuron
- $o(\mathbf{x}) = \text{sign}(\sum_{i=0}^n w_i \cdot x_i)$

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

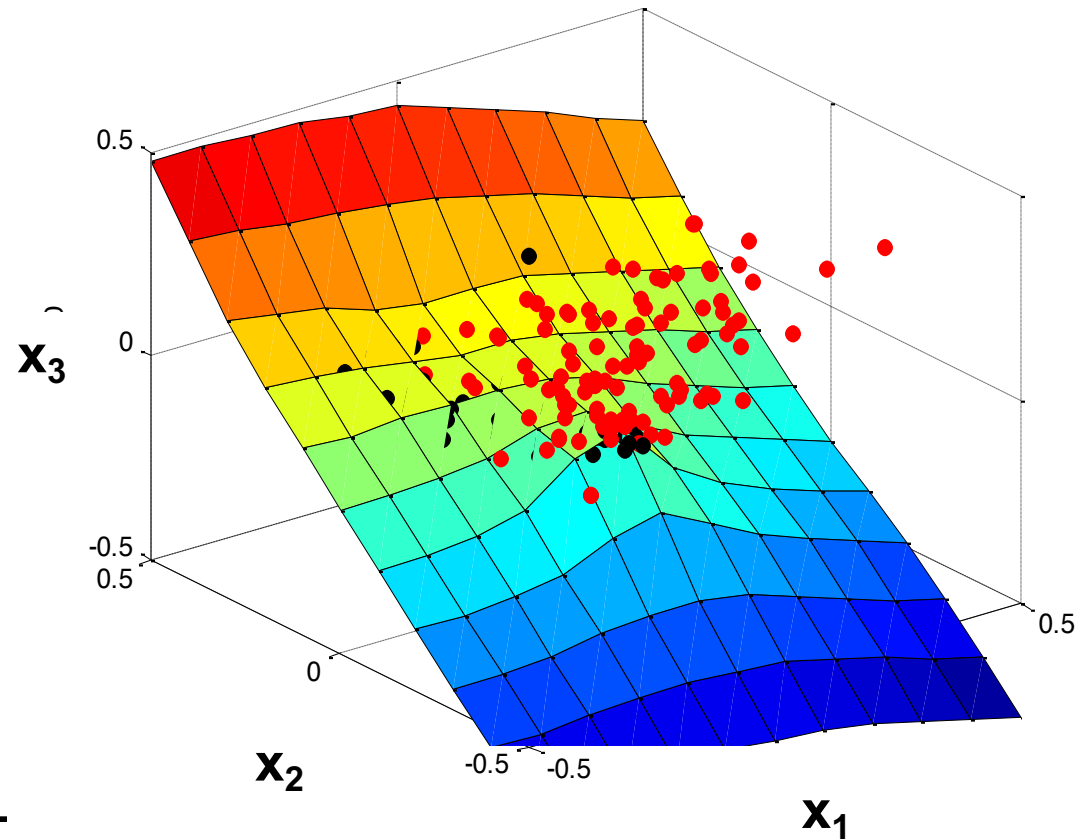
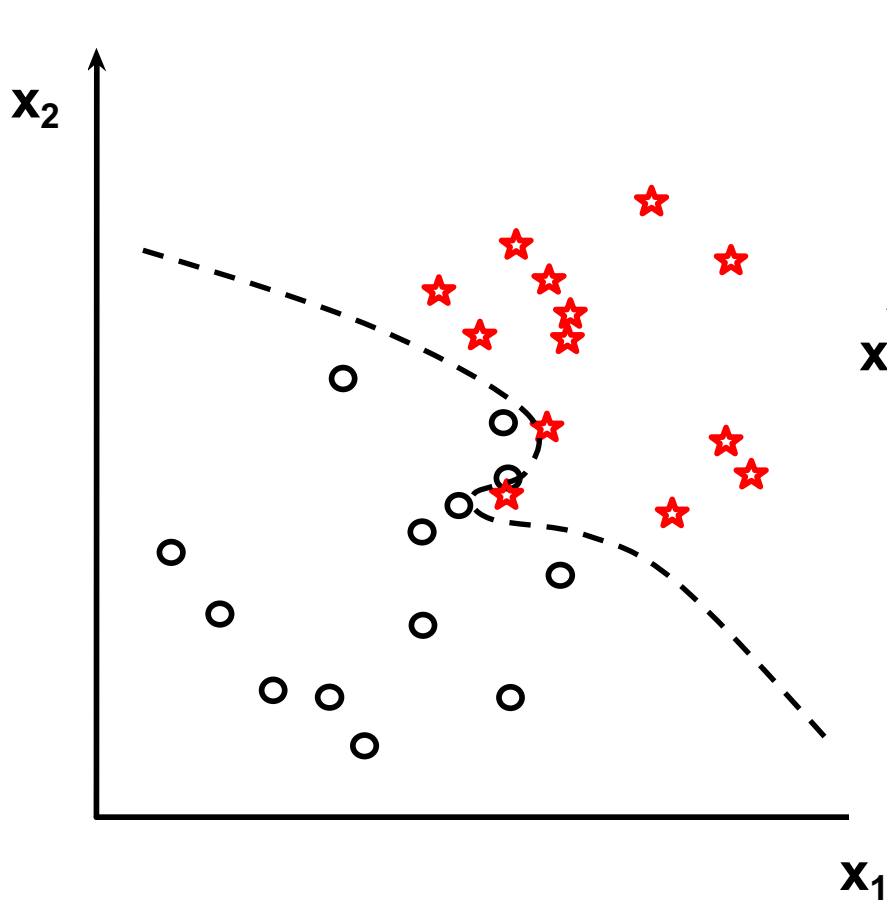


Linear decision boundary

hyperplane

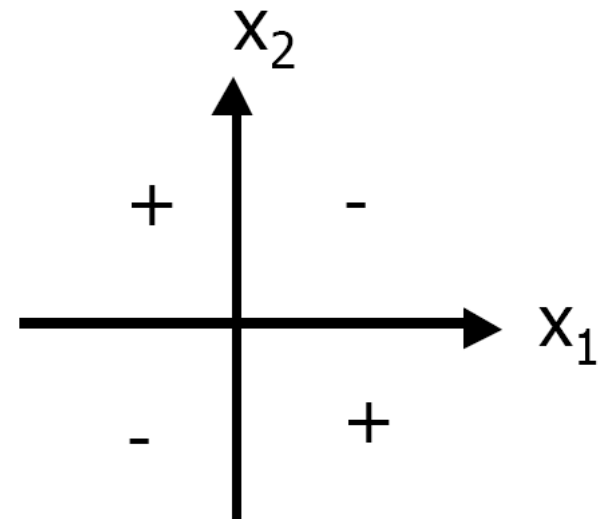
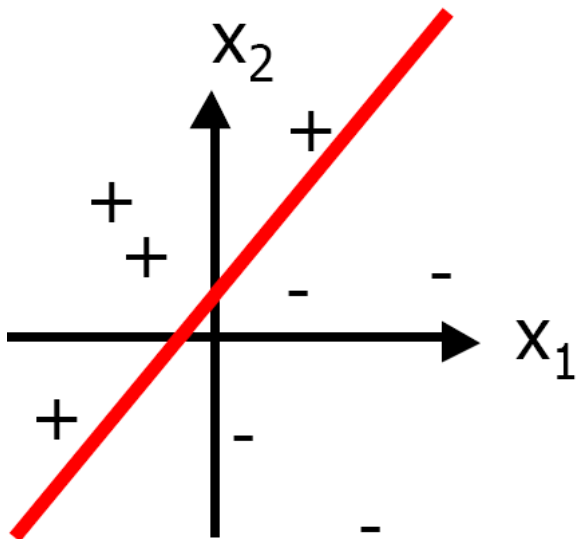


Non-linear decision boundary



Perceptron – decision surface

- Hyperplane (“line”) in an n-dimensional space
- Find a Perceptron to solve the AND pb. for two inputs x_1 , $x_2 \rightarrow w_i=?$
- Functions not linearly separable (e.g. XOR) \rightarrow not representable with only one neuron \rightarrow use more neurons \rightarrow neural network (NN)



Perceptron – learning rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
 - o is perceptron output
 - η is small constant (e.g., .1) called *learning rate*
-
- If o correct ($t = o$) \rightarrow weights w_i are not changed
 - If o incorrect ($t \neq o$) \rightarrow weights w_i are changed s.t o is closer to t
 - Algorithm converges to correct classification if
 - Training data is linearly separable
 - Learning rate is sufficiently small

Gradient descent – learning rule

- Consider perceptron without threshold (i.e. no $\text{sign}(o)$) and continuous outputs o (not just 1, -1)
- Train w_i s.t. they min the squared err (LMS-least mean squared error)

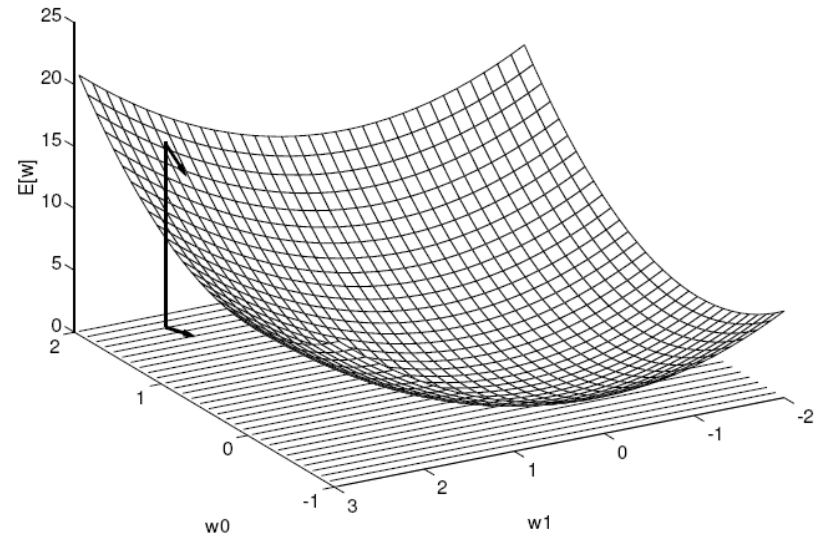
$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

where D is the set of training data

Gradient descent

- = steepest descent rule = delta rule = LMS rule = Widrow-Hoff rule
- Name “delta rule” (Widrow-Hoff rule) comes from

$$\Delta w_i = \eta(t - o)x_i$$



- Finds local minima by taking steps opposite to the gradient
- Gradient = vector with the greatest rate of increase
- Small learning rate \rightarrow algorithm converges

Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Training rule:

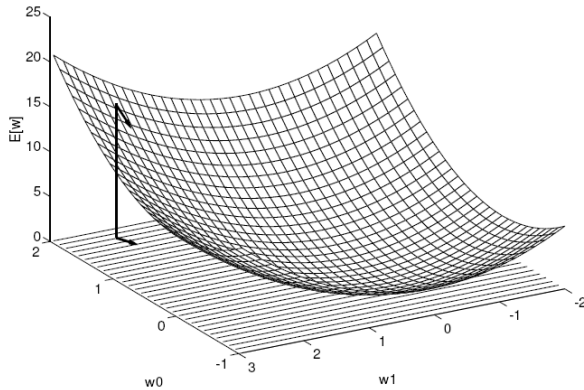
$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradient descent

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Training rule:

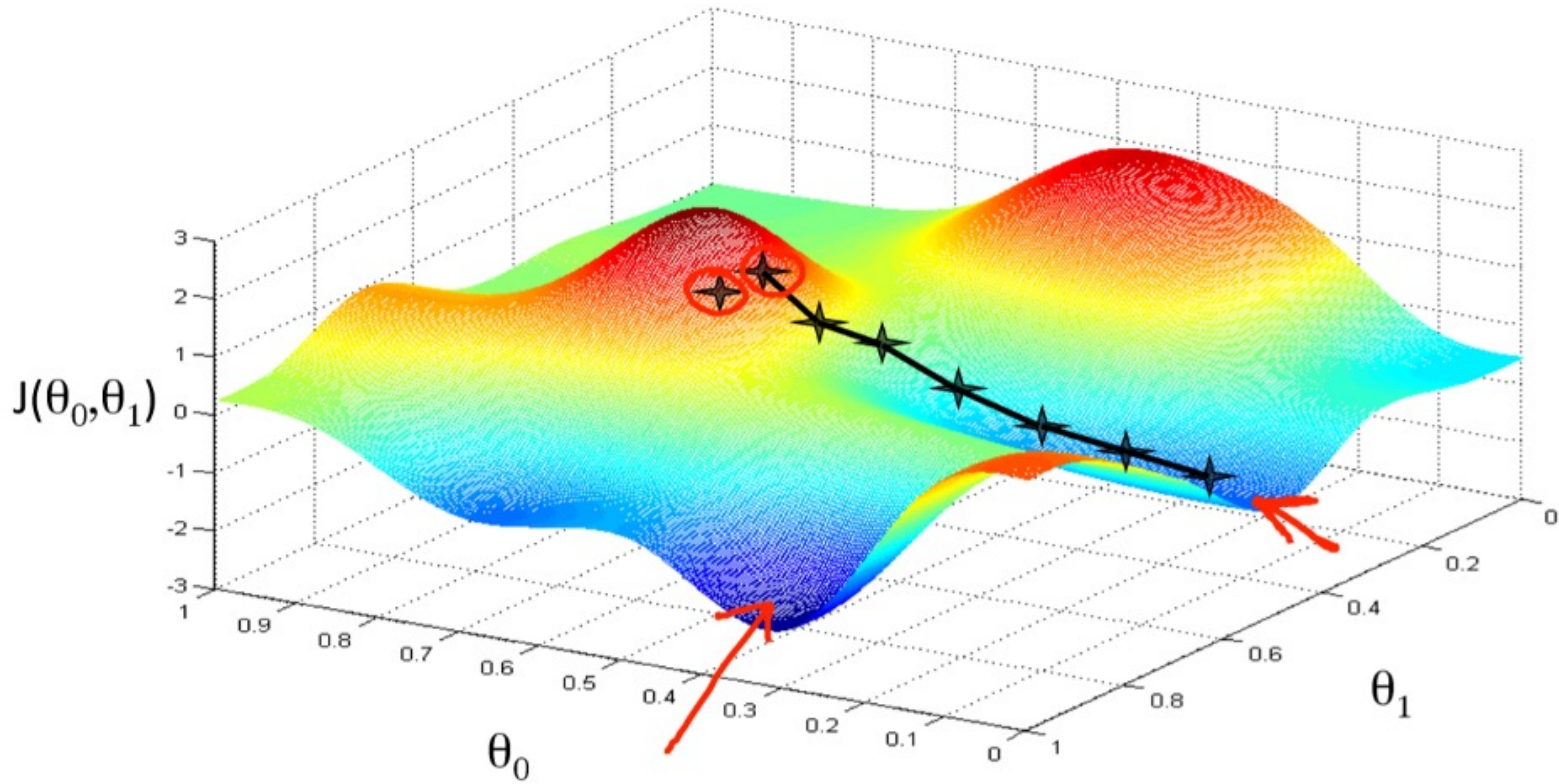
$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ \frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d) (-x_{i,d}) \end{aligned}$$

Gradient descent illustration



You might miss the global minima on error surface.

Gradient descent algorithm

- Ip: tr. ex., η
 - Each tr. ex is a pair $\langle(x_1, \dots, x_n), t\rangle$
 - Op: w
1. w init. with small random values
 2. Until termination cond. met do
 1. each $\Delta w_i = 0$
 2. For each tr. ex. $\langle(x_1, \dots, x_n), t\rangle$ do
 1. Input the instance (x_1, \dots, x_n) to the neuron and compute o
 2. For each w_i do
$$\Delta w_i = \Delta w_i + \eta(t-o)x_i$$

%accumulates change from each tr.ex
 3. For each w_i do
$$w_i = w_i + \Delta w_i$$

%update w only once \rightarrow batch mode

Gradient descent

Batch mode Gradient Descent:

Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
 2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$
-

Incremental mode Gradient Descent:

Do until satisfied

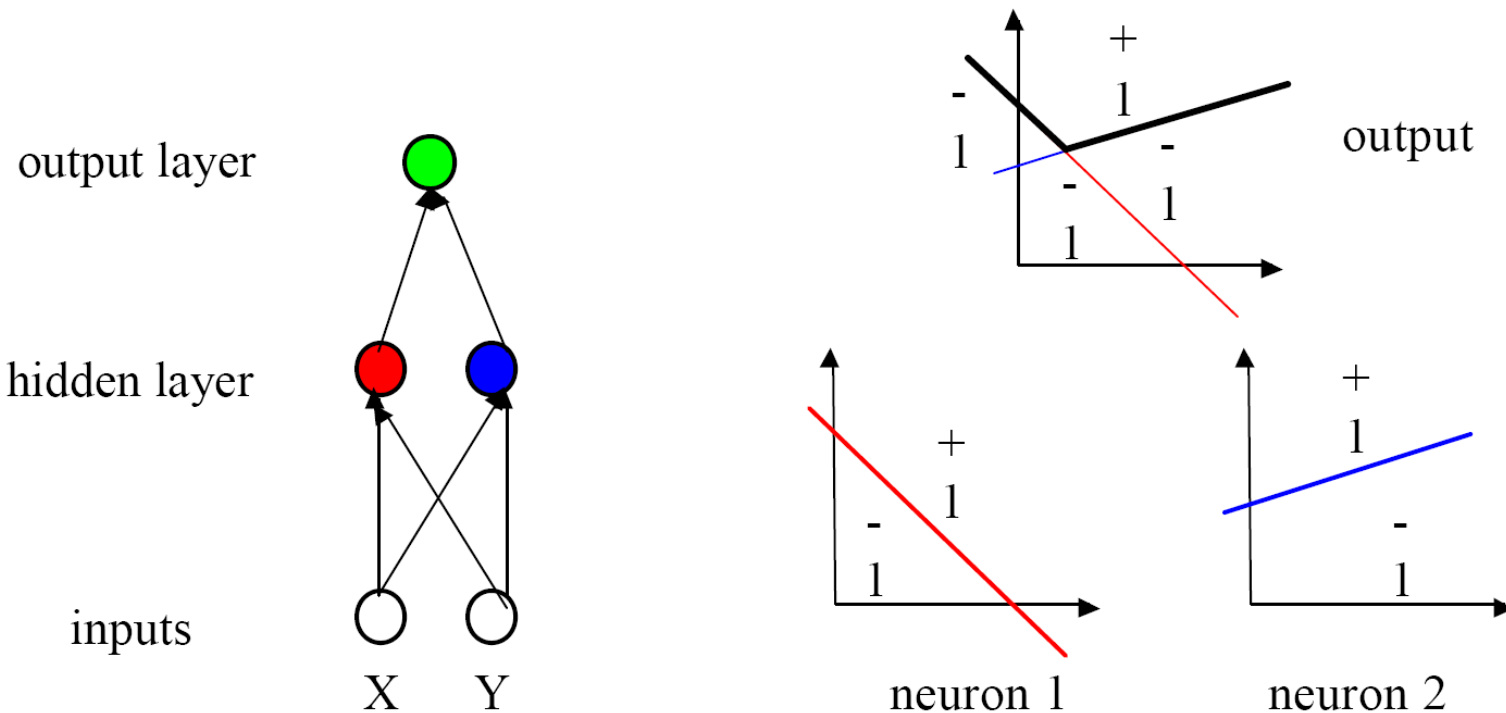
- For each training example d in D
 1. Compute the gradient $\nabla E_d[\vec{w}]$
 2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$
-

Gradient descent - conclusions

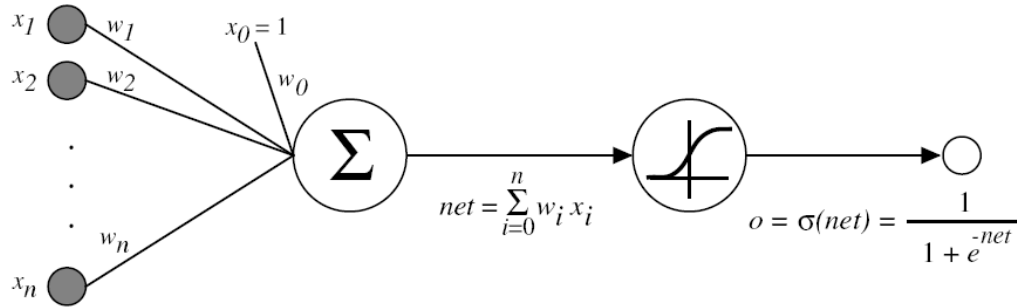
- Finds a solution that minimizes the error
 - works also for non-linearly separable data (unlike perceptron!)
 - tolerates noisy data
- Local minima (= minimum error) obt. by taking steps opposite to the gradient
- Small learning rate → algorithm converges
- Weaknesses
 - Slow convergence
 - If not small enough learning rate → might miss the min
- Limitations for both perceptron and gradient descent
 - Solve only a small class of pb.
 - → Combine many neurons in a network

Multilayer networks

- Increase representation power



Sigmoid unit



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units \rightarrow Backpropagation

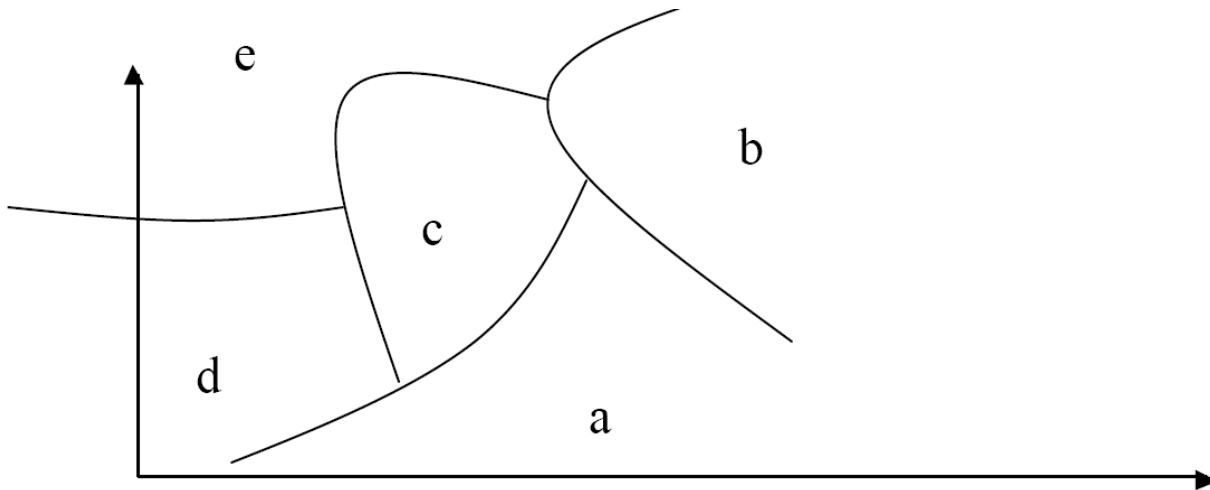
- Graph σ

```
x=(-10:0.1:10);  
y=1./(1+(2.71).^(-x));  
plot(x,y);
```

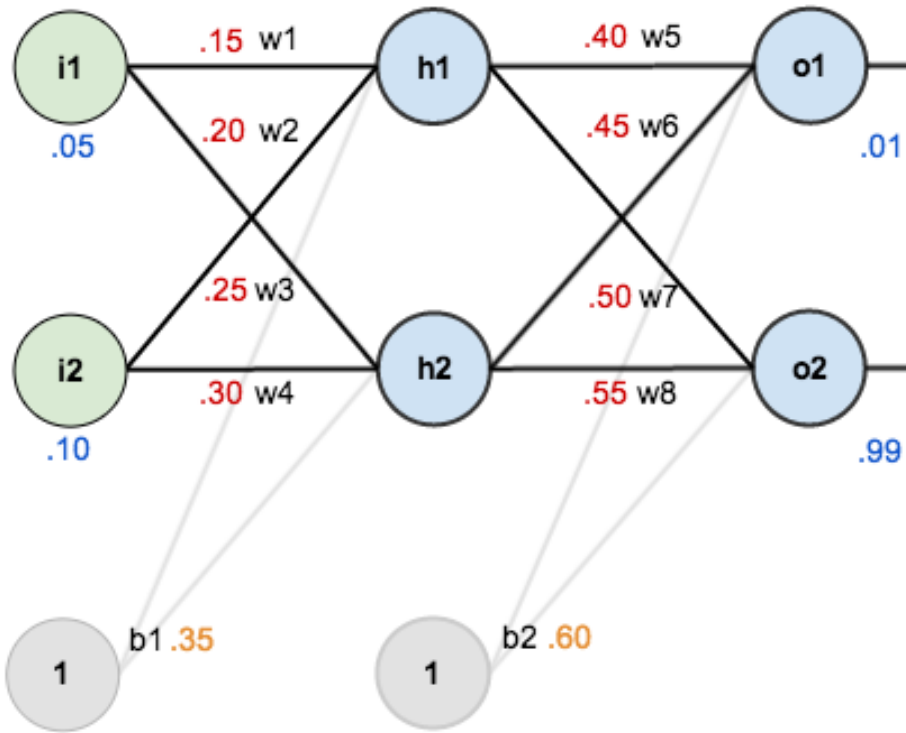
- Can you understand why is called squashing function?
- aka. logistic function

Sigmoid unit

- Causes non-linear decision surface
- Very powerful representation
- OBS. Multiple layers of linear units still produce only linear functions → use non-linear activation fct.



Examples of NN and 1-to-N encoding



5) Ex. of 1-to-N

Class label =

[2
3
1
3
...]

Class label encoded =

[0 1 0
0 0 1
1 0 0
0 0 1
...]

1) If using sigmoid activation fct. → output in [0,1]

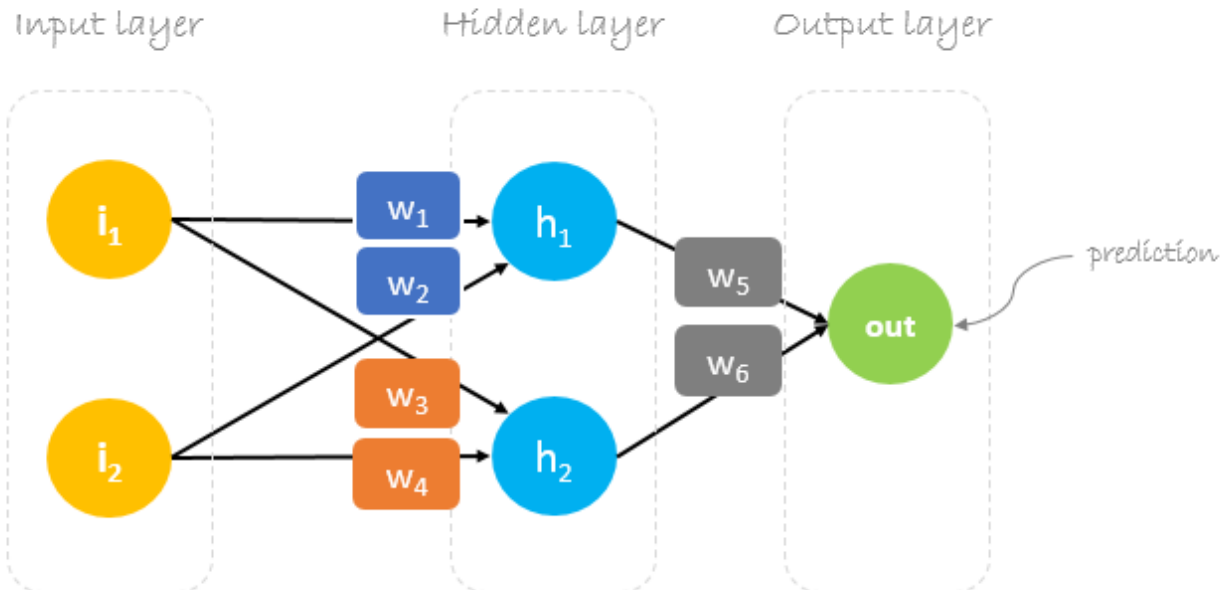
2) What is the difference between using 1 vs 2 output neurons?

3) Assume, a 2-class classification problem. It is easier to learn outputs

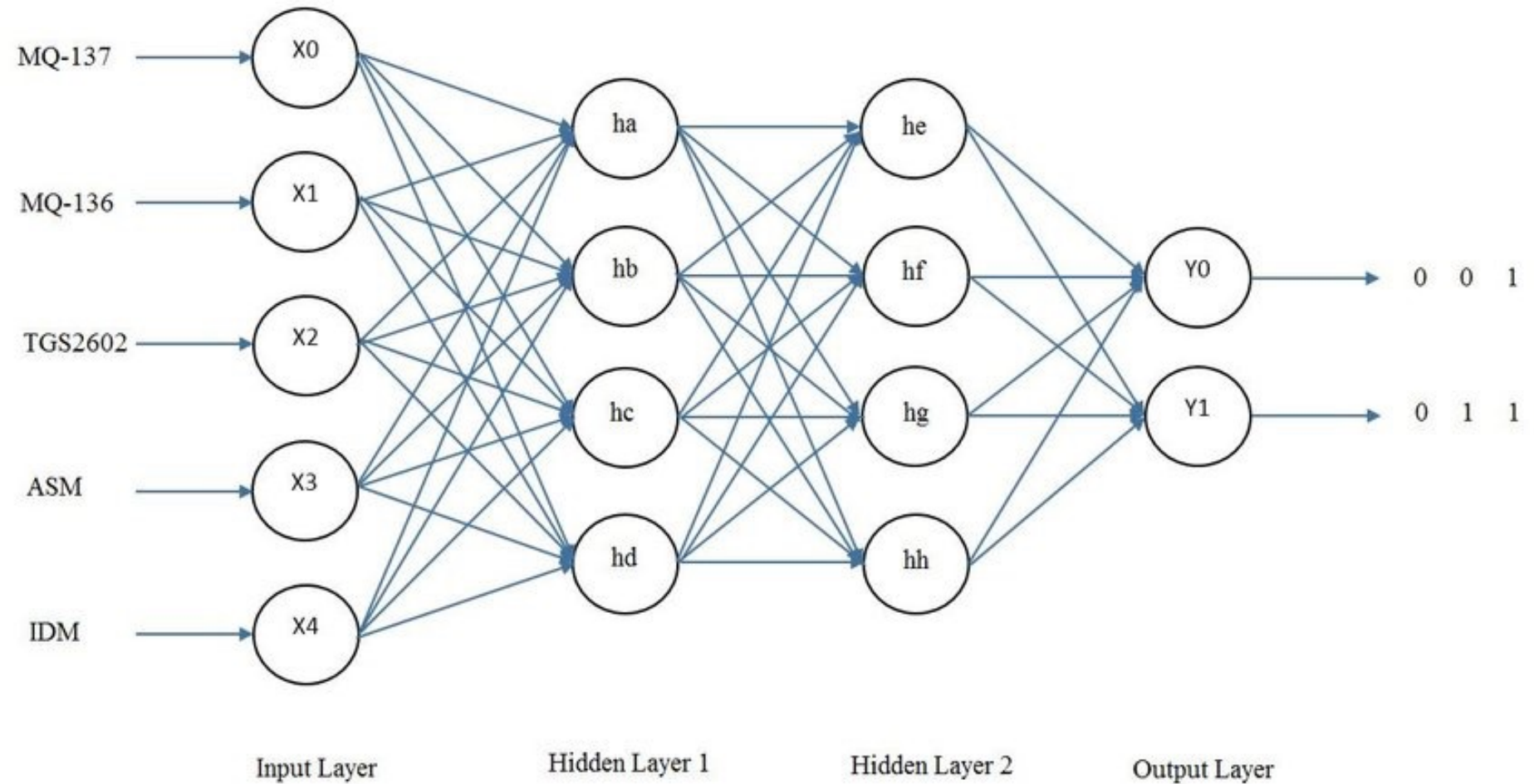
[1 0] as class 1
[0 1] as class 2

4) For 3-class classification this is even more obvious:

[1 0 0] is output for class 1, etc.

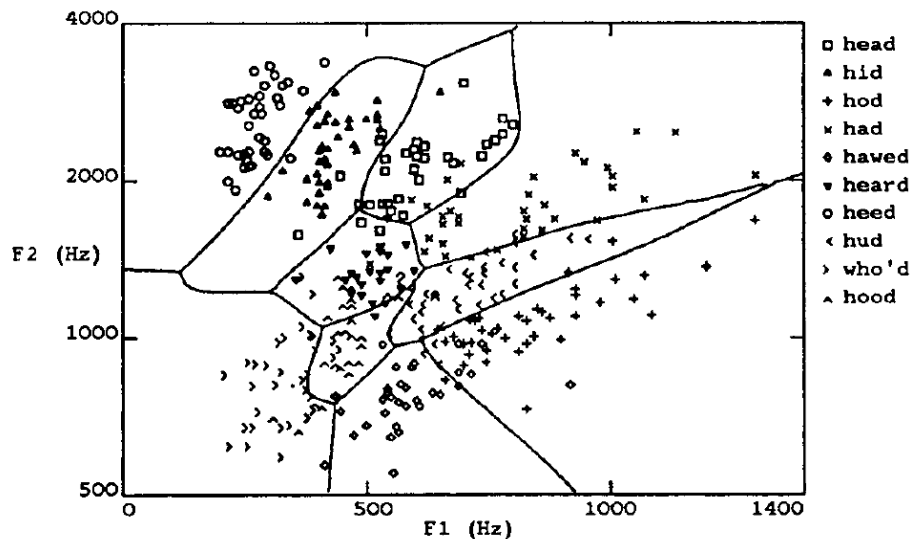
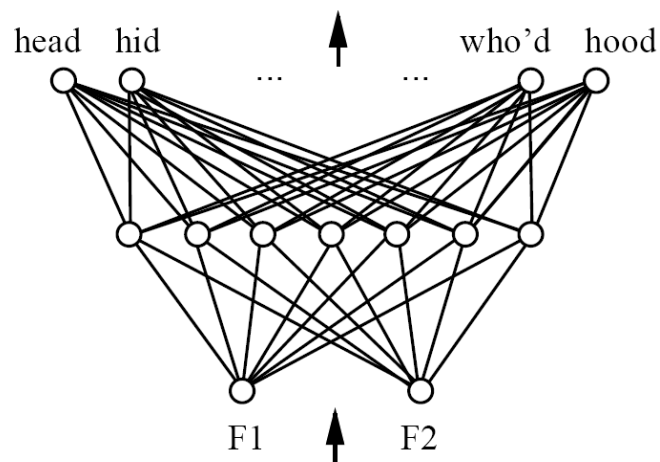


Examples of NN



Multilayer NN with sigmoid units

- Speech recognition
- Data from spectral analysis of the sound
- 10 outputs



Backpropagation algorithm

Initialize all weights to small random numbers.

Until satisfied, Do

- For each training example, Do

1. Input the training example to the network and compute the network outputs
2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k$$

4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

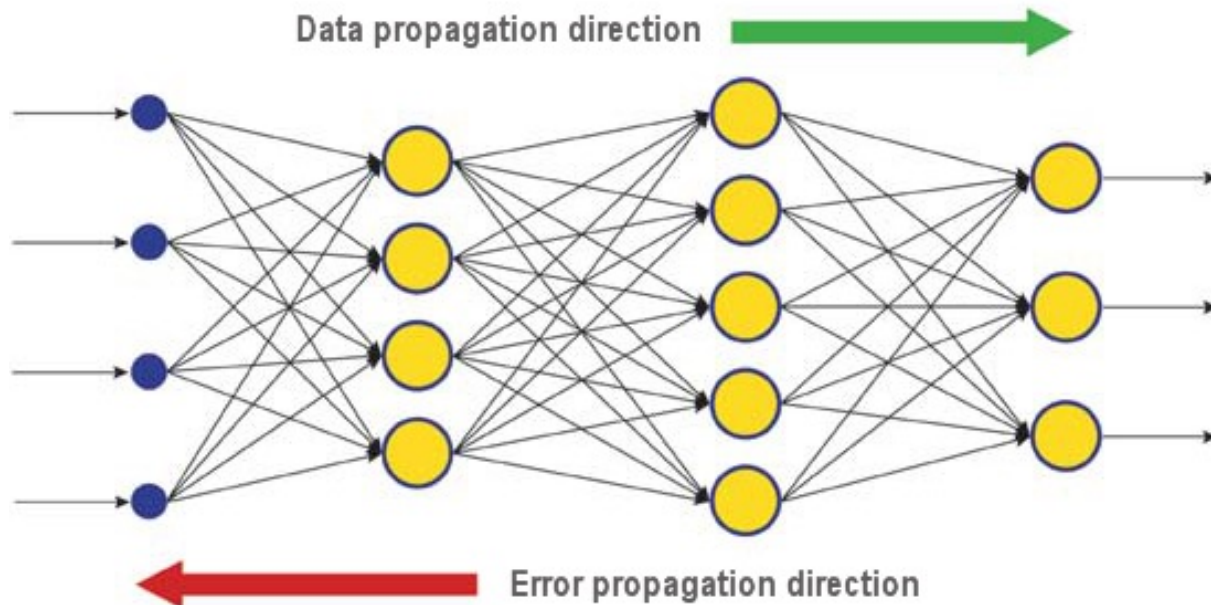
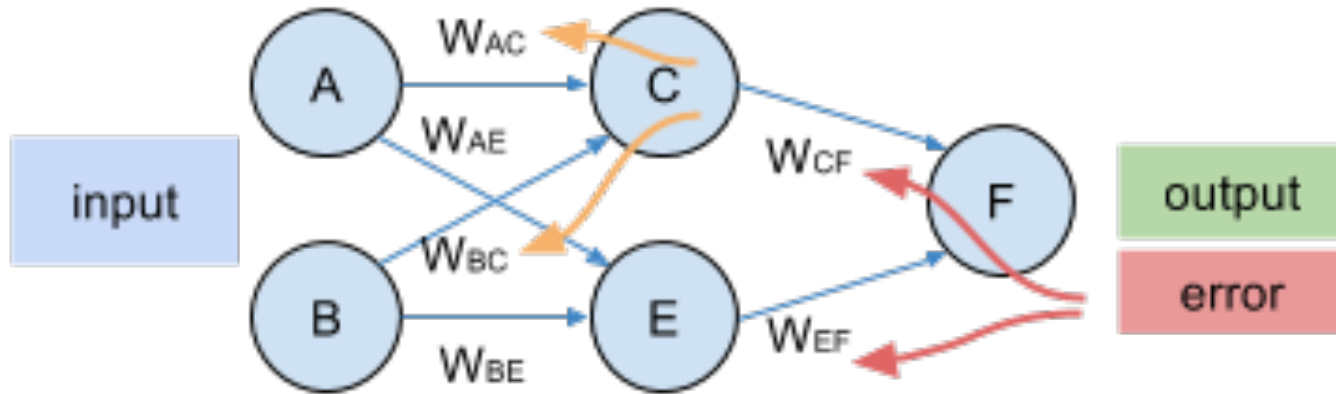
where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

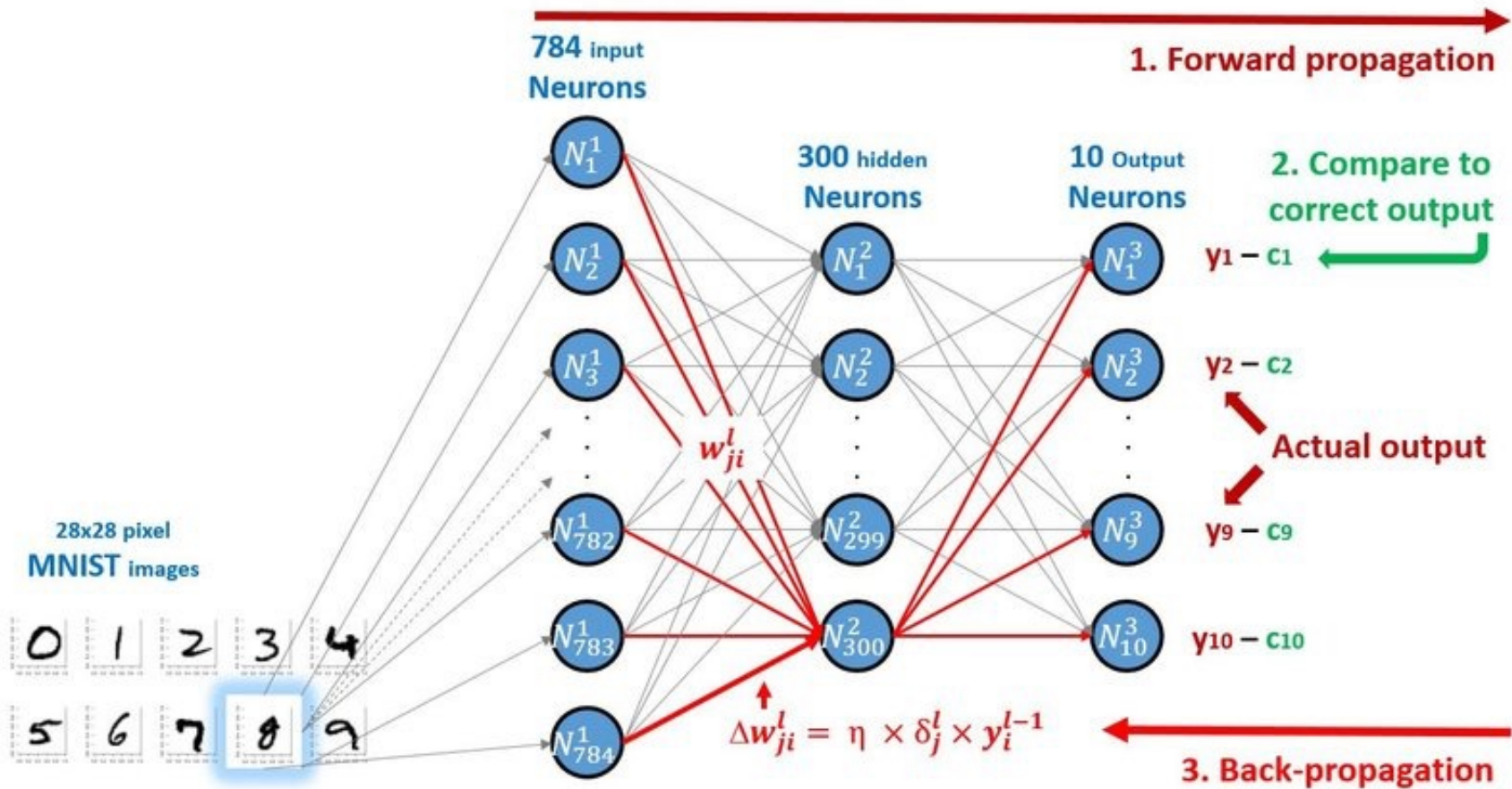
Obs. 1 Steps 2, 3 and 4 propagate the err backward through NN

Obs. 2 Initial w near zero \rightarrow init. net near-linear (see logistic fct. around 0) \rightarrow increasingly non-linear functions possible as training progresses

Backpropagation illustration



NN for classifying hand-written digits



Bias in Backpropagation FNN

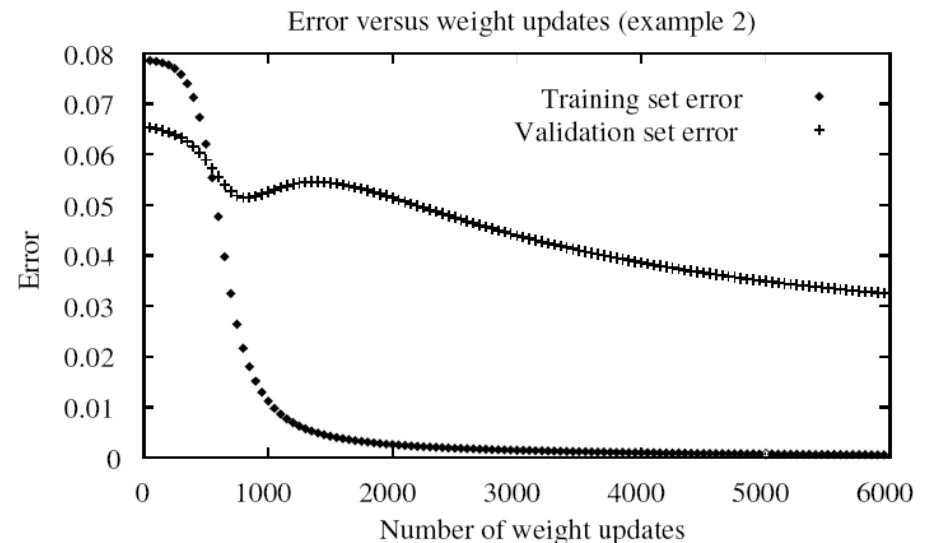
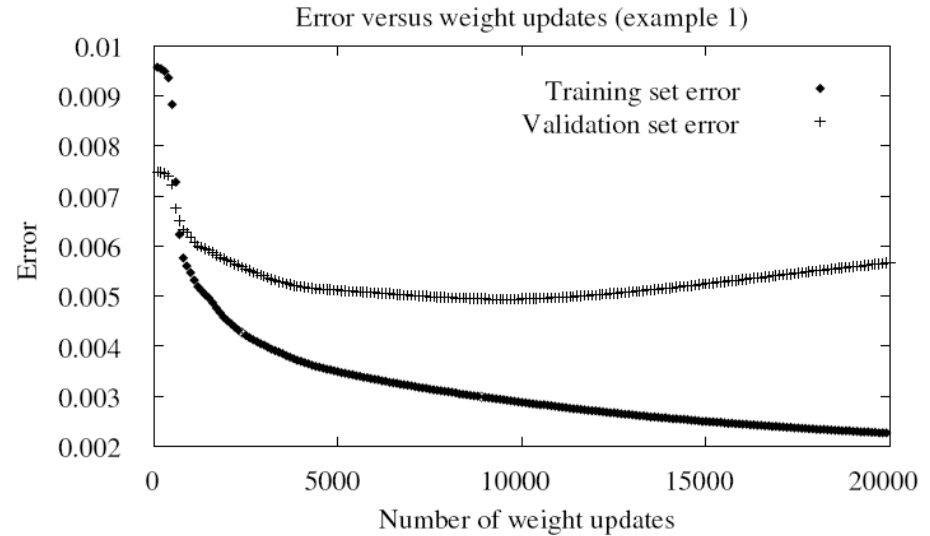
- Mitchel: interpolates two pos. ex. that do not have a intervening negative, with a pos.
- Net topology chosen by trainer
 - # of layers
 - # of neurons
 - transfer fct.
 - 1. many hidden layers and neurons
 - powerful net
 - can approx. many hypotheses
 - weak inductive bias → poor generalization
 - 2. smaller hidden layers and neurons
 - weak net
 - can approx. fewer hypotheses
 - stronger inductive bias
 - **PREFERRED**: an h that approx. well t from training has higher probability of well approx. the actual (TRUE) t
- GOAL: find the weakest topology to learn the training data → strongest inductive bias → best generalization

Overfitting in Backpropagation FNN

- “memorize” training data, but cannot generalize
- Choice of too powerful a net provides with excessive # of h , thus making available h that fit tr. data but do not match t well
- May use a powerful net + add some bias
 - weight-decay
 - adds bias by decreasing all w by a small amount at each iteration
→ non-reinforced weights get smaller
 - k-cross validation
 - split tr. data in k subsets, train k different nets by using one of the k parts for test and remaining $k-1$ for training → select the net that generalizes best
- OBS. More than 1 or 2 layers on neurons leads to overfitting

More about overfitting

- Tr. data is not representative of general distribution of examples
- After many iterations, Backpropagation will create overly complex dec. surface that fits noise
- Solution:
 - Use validation set
 - k-cross validation (if little data)
- One can discover the best net at unpredictable time (keep a running w of min err), e.g. top figure shows best net at epoch~9000



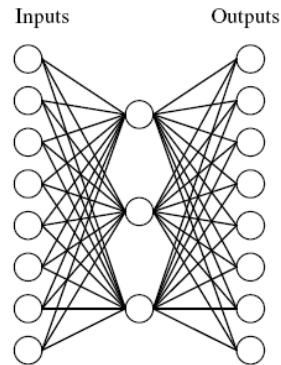
Backpropagation algorithm

- Gradient descent over entire network weight vector
- Min error over training ex.
- Finds local (not always global) min – err surface has multiple local min!!
- However, in practice works well – run it multiple times with different random initial weights
- Slow training (1,000 – 10,000 iterations using same tr. examples)
- Using net after training is fast
- Can overfitt

Representation power of FFNN

- Every boolean fct. can be repres. by a NN with one hidden layer (input x hidden x output → 2 layers in total)
- NN with one hidden layer (input x hidden x output → 2 layers in total) can approximate ANY continuous function (Cybenko'89, Hornik'89)
- NN with two hidden layers (input x hidden x hidden x output → 3 layers in total) can approximate ANY function (Cybenko'88)

Ex.1 8-3-8 Binary encoder - decoder



- Hidden layer representation

→ essential info from 8 ip. captured by 3 learned hidden units

→ ability to invent features not explicitly introduced by humans

A target function:

Input	Output
10000000	→ 10000000
01000000	→ 01000000
00100000	→ 00100000
00010000	→ 00010000
00001000	→ 00001000
00000100	→ 00000100
00000010	→ 00000010
00000001	→ 00000001

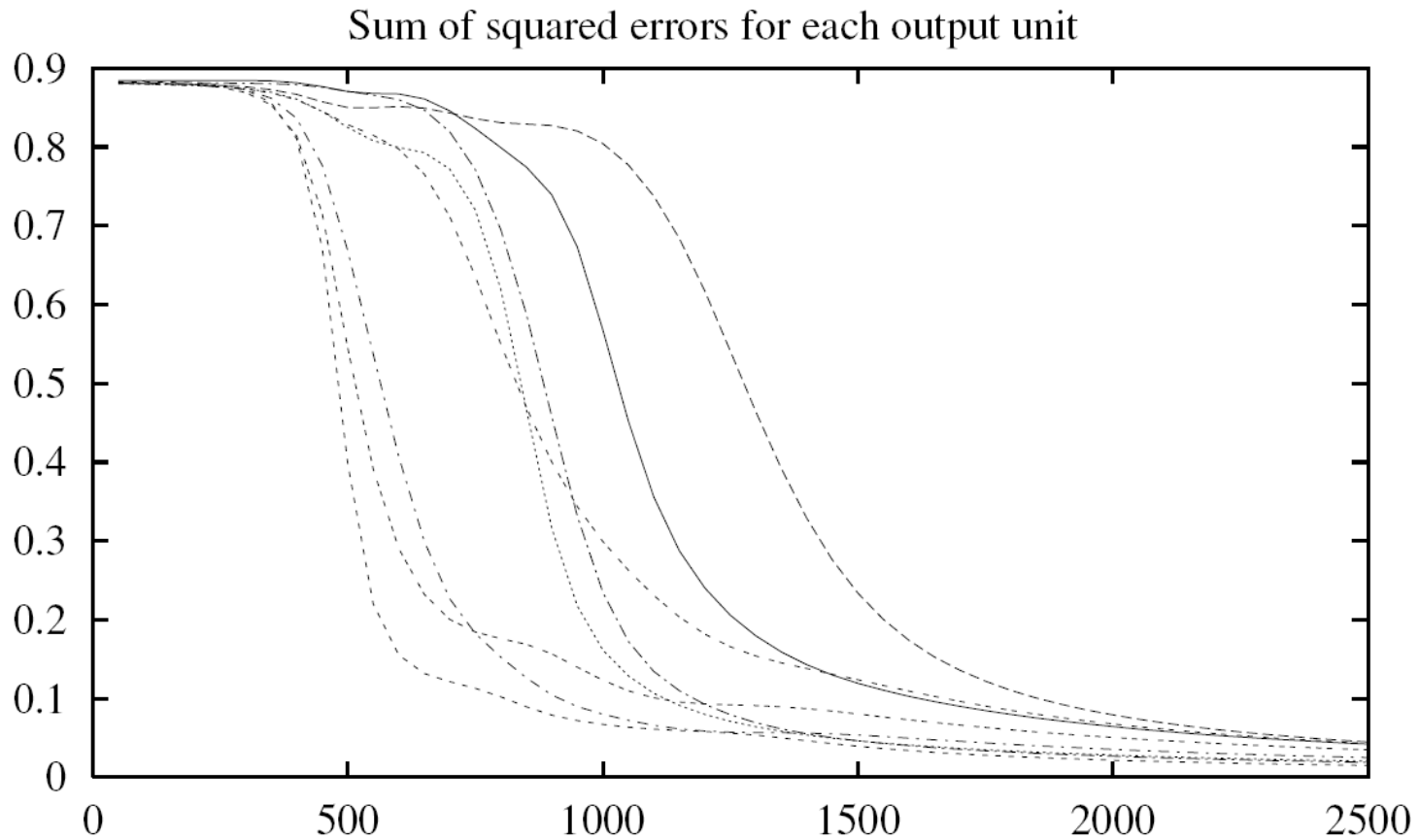
Learned hidden layer representation:

Input	Hidden Values	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .01 .11 .88	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .22 .99 .99	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001

Can this be learned??

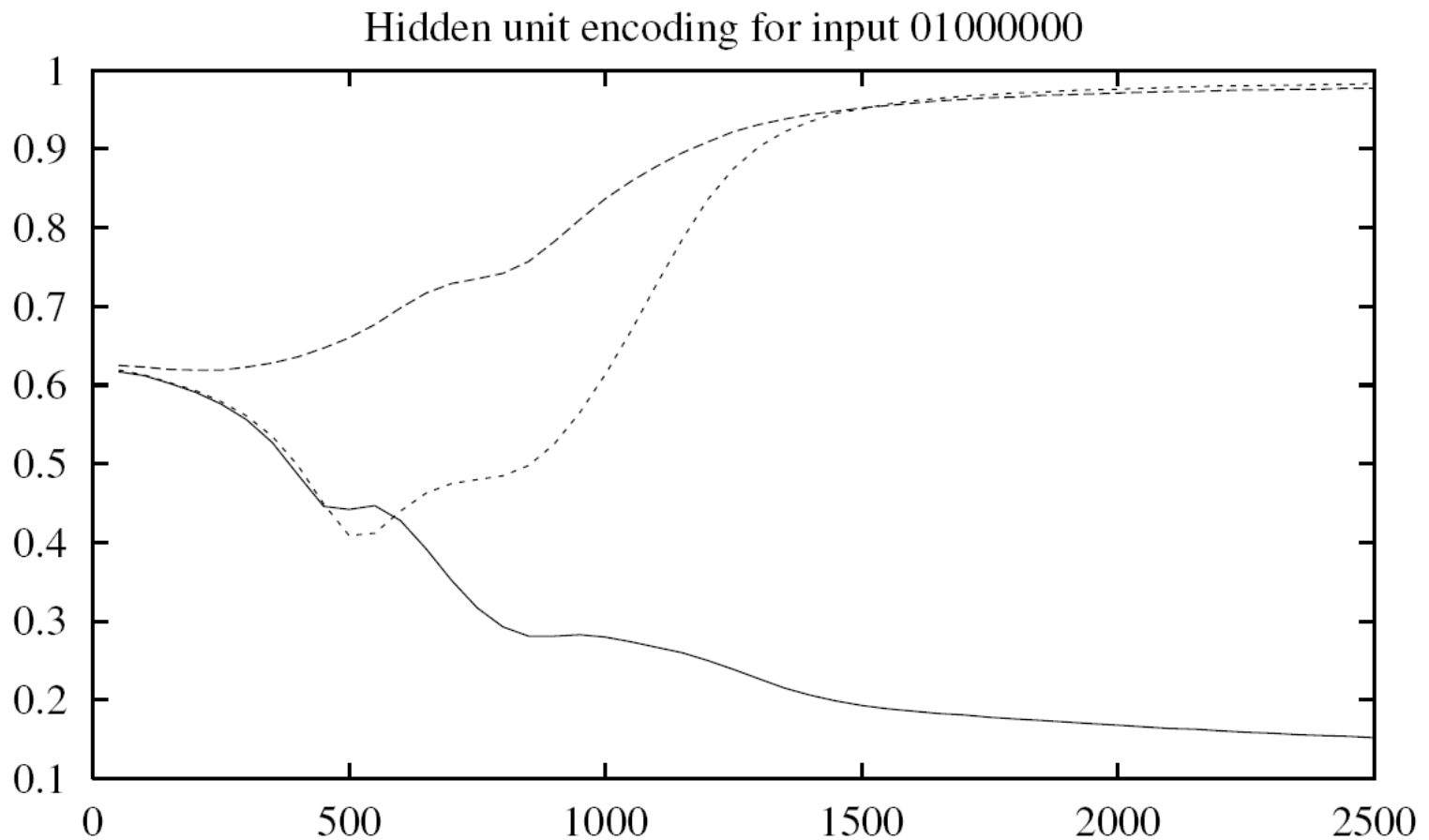
Training

- Sum of squared err for the 8 output units



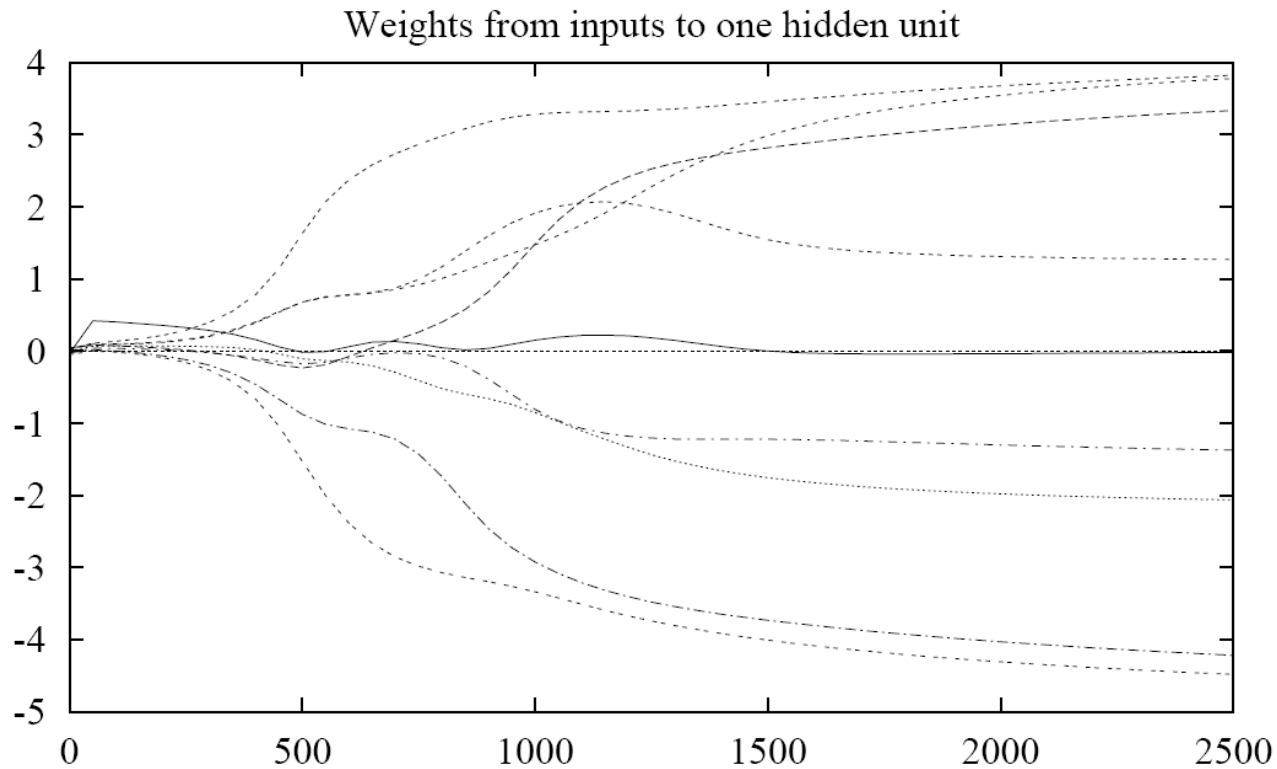
Training

- Hidden unit encoding for the 3 hidden units



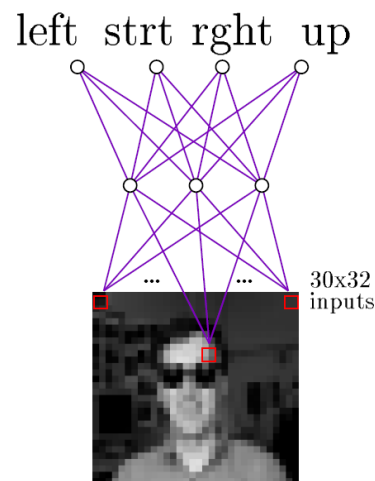
Training

- 9 weights from 8 ip. to 1 hidden unit



Ex.2 NN for face recognition

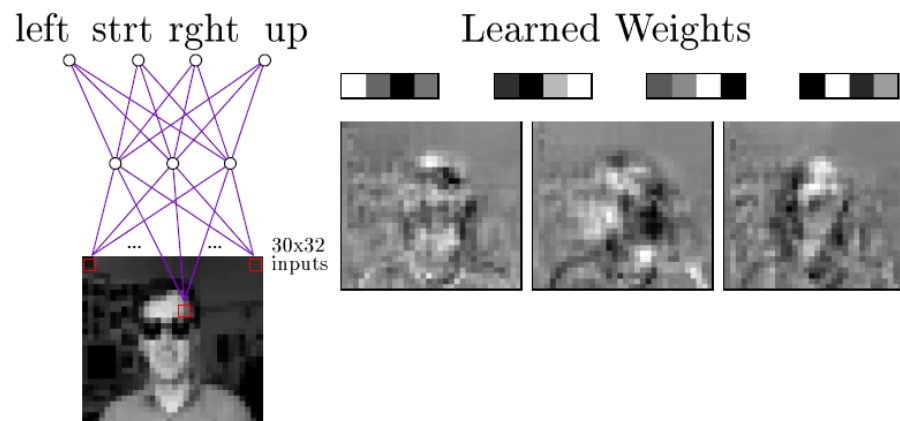
- Data
 - 624 greyscale images 120x128
 - Pixel intensity 0-255
- Task
 - predict forward, left, right, up
- Net
 - Ip: 30x32 pixel intensities
 - Op: 4 nodes (1-of-n op. coding) e.g. (.1,.1,.9,.1)
 - One hidden layer: 3 nodes
 - Tr. Time: 5 min to achieve 90% acc. (vs. 60 min for 30 hidden nodes which performs just slightly better (~92%))



Typical input images

Ex.2 NN for face recognition

- Weights into the three hidden layer nodes after 100 epochs
- Weights from image pixels into each hidden unit, plotted in the pos. of the corresp. pixel



Typical input images

Resources

- NN for Pattern Recognition, Bishop C.M., 1996

- Stuttgart NN Simulator (SNNS)

<http://www.ra.cs.uni-tuebingen.de/SNNS/>

- NN for face recognition

<http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitcheII/ftp/faces.html>