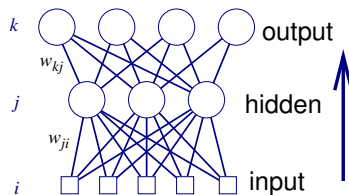


Neural Networks

- Threshold units
- Gradient descent
- Multilayer networks
- Backpropagation
- Hidden layer representations
- Example: Face Recognition
- Advanced topics
- And, more.

1

Artificial Neural Networks



- Many neuron-like threshold switching units (real-valued)
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically: New learning algorithms, new optimization techniques, new learning principles.

3

Biological Neurons and Networks

- Neuron switching time $\sim .001$ second (1 ms)
- Number of neurons $\sim 10^{10}$
- Connections per neuron $\sim 10^4-5$
- Scene recognition time $\sim .1$ second (100 ms)
- 100 processing steps doesn't seem like enough
[→] much parallel computation

2

Biologically Motivated (or Accurate) Neural Networks

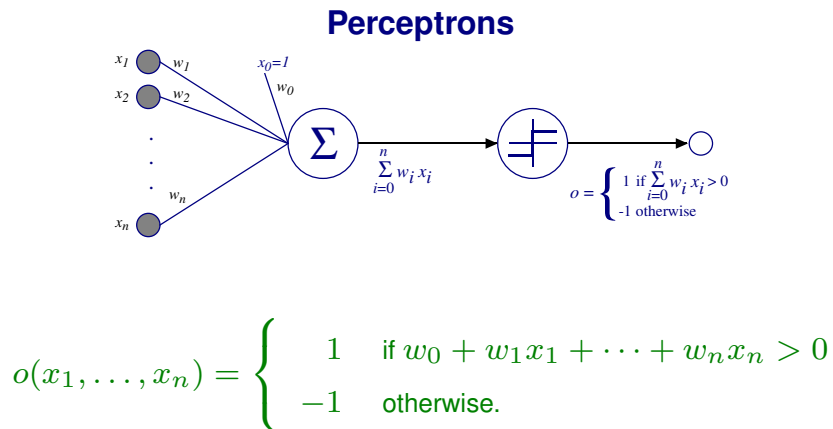
- Spiking neurons
- Complex morphological models
- Detailed dynamical models
- Connectivity either based on or trained to mimic biology
- Focus on **modeling** network/neural/subneural processes
- Focus on natural **principles** of neural computation
- Different forms of learning: spike-timing-dependent plasticity, covariance learning, short-term and long-term plasticity, etc.

4

When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Long training time (may need occasional, extensive **retraining**)
- Form of target function is unknown
- Fast evaluation of learned target function
- Human readability of result is unimportant

5

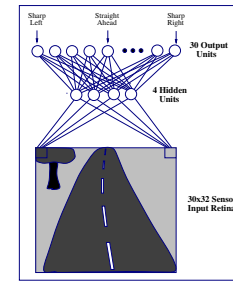


Sometimes we'll use simpler vector notation:

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

7

Example Applications (more later)



(a) ALVINN



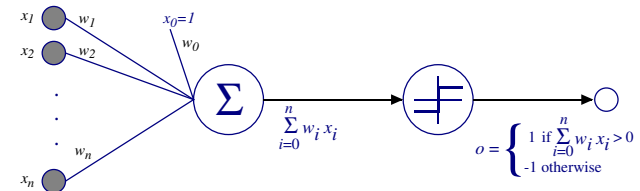
(b) <http://yann.lecun.com>

Examples:

- Speech synthesis
- Handwritten character recognition (from yann.lecun.com).
- Financial prediction, Transaction fraud detection (Big issue lately)
- Driving a car on the highway

6

Hypothesis Space of Perceptrons

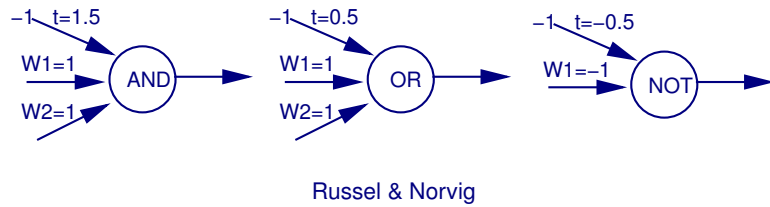


- The tunable parameters are the weights w_0, w_1, \dots, w_n , so the space H of candidate hypotheses is the set of **all possible combination of real-valued weight vectors**:

$$H = \{\vec{w} \mid \vec{w} \in \mathcal{R}^{(n+1)}\}$$

8

Boolean Logic Gates with Perceptron Units

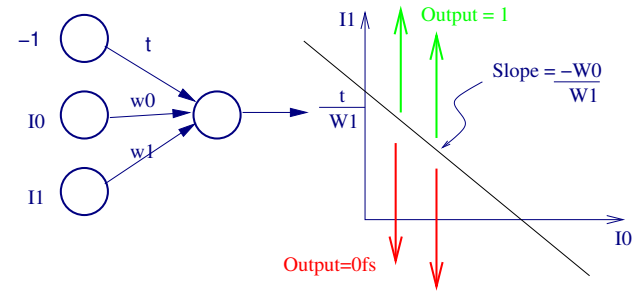


- Perceptrons can represent basic boolean functions.
- Thus, a network of perceptron units can compute any Boolean function.

What about XOR or EQUIV?

9

What Perceptrons Can Represent



Perceptrons can only represent **linearly separable** functions.

- Output of the perceptron:

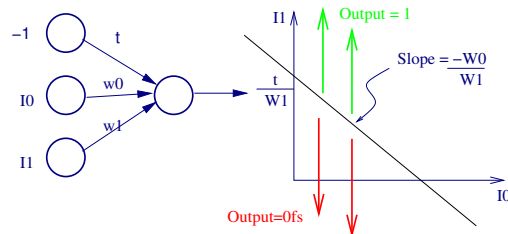
$$W_0 \times I_0 + W_1 \times I_1 - t > 0, \text{ then output is } 1$$

$$W_0 \times I_0 + W_1 \times I_1 - t \leq 0, \text{ then output is } -1$$

The hypothesis space is a collection of separating lines.

10

Geometric Interpretation



- Rearranging

$$W_0 \times I_0 + W_1 \times I_1 - t > 0, \text{ then output is } 1,$$

we get (if $W_1 > 0$)

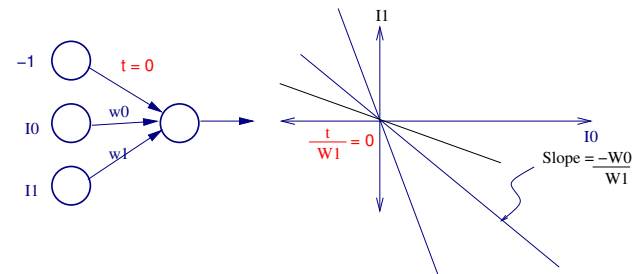
$$I_1 > \frac{-W_0}{W_1} \times I_0 + \frac{t}{W_1},$$

where points above the line, the output is 1, and -1 for those below the line.

Compare with

$$y = \frac{-W_0}{W_1} \times x + \frac{t}{W_1}.$$

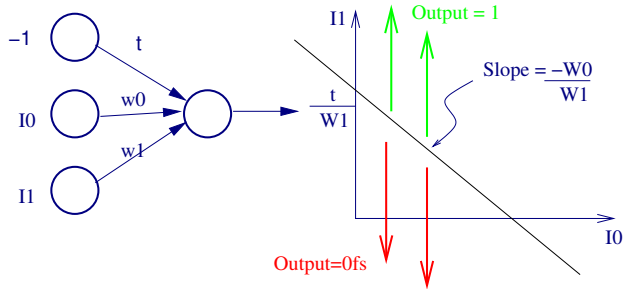
The Role of the Bias



- Without the bias ($t = 0$), learning is limited to adjustment of the slope of the separating line passing through the origin.
- Three example lines with different weights are shown.

12

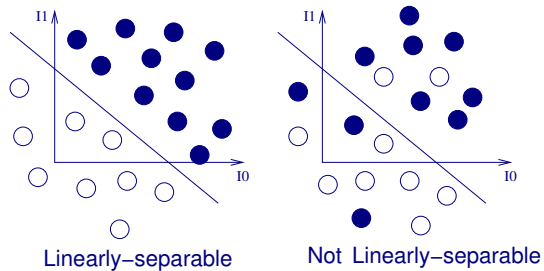
Limitation of Perceptrons



- Only functions where the -1 points and 1 points are clearly separable can be represented by perceptrons.
- The geometric interpretation is generalizable to functions of n arguments, i.e. perceptron with n inputs plus one threshold (or bias) unit.

13

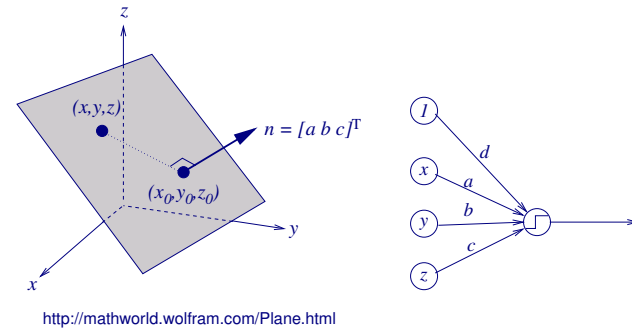
Linear Separability



- For functions that take integer or real values as arguments and output either -1 or 1.
- Left: linearly separable (i.e., can draw a straight line between the classes).
- Right: not linearly separable (i.e., perceptrons cannot represent such a function)

15

Generalizing to n -Dimensions

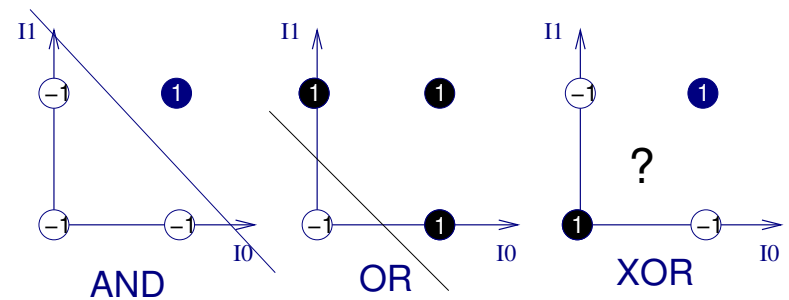


<http://mathworld.wolfram.com/Plane.html>

- $\vec{n} = (a, b, c)$, $\vec{x} = (x, y, z)$, $\vec{x}_0 = (x_0, y_0, z_0)$.
- Equation of a plane: $\vec{n} \cdot (\vec{x} - \vec{x}_0) = 0$
- In short, $ax + by + cz + d = 0$, where a, b, c can serve as the weight, and $d = -\vec{n} \cdot \vec{x}_0$ as the bias.
- For n -D input space, the decision boundary becomes a $(n - 1)$ -D hyperplane (1-D less than the input space).

14

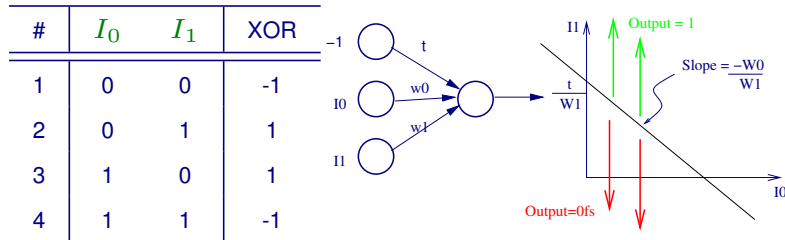
Linear Separability (cont'd)



- Perceptrons cannot represent XOR!
- Minsky and Papert (1969)

16

XOR in Detail



$W_0 \times I_0 + W_1 \times I_1 - t > 0$, then output is 1:

- 1 $-t \leq 0 \rightarrow t \geq 0$
- 2 $W_1 - t > 0 \rightarrow W_1 > t$
- 3 $W_0 - t > 0 \rightarrow W_0 > t$
- 4 $W_0 + W_1 - t \leq 0 \rightarrow W_0 + W_1 \leq t$

$2t < W_0 + W_1 < t$ (from 2, 3, and 4), but $t \geq 0$ (from 1), a contradiction.

17

Learning in Perceptrons (Cont'd)

$$w_i \leftarrow w_i + \eta(t - o)x_i$$

- When $t = o$, weight stays.
- When $t = 1$ and $o = -1$, change in weight is:

$$\eta(1 - (-1))x_i > 0$$

if x_i are all positive. Thus $\vec{w} \cdot \vec{x}$ will increase, thus eventually, output o will turn to 1.

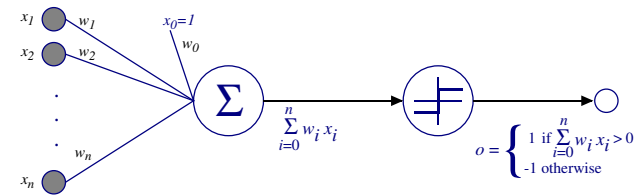
- When $t = -1$ and $o = 1$, change in weight is:

$$\eta(-1 - 1)x_i < 0$$

if x_i are all positive. Thus $\vec{w} \cdot \vec{x}$ will decrease, thus eventually, output o will turn to -1.

19

Learning: Perceptron Rule



- The weights do not have to be calculated manually.
- We can train the network with (input,output) pair according to the following weight update rule:

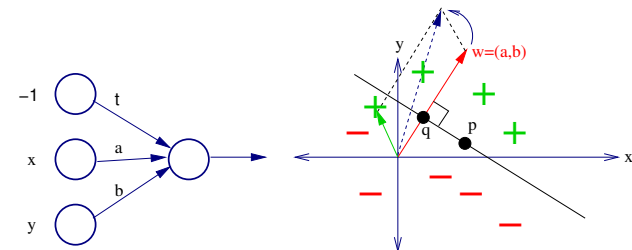
$$w_i \leftarrow w_i + \eta(t - o)x_i$$

where η is the learning rate parameter.

- Proven to converge **if input set is linearly separable** and η is small.

18

Learning in Perceptron: Another Look



- The perceptron on the left can be represented as a line shown on the right (why? see page 14).
- Learning can be thought of as adjustment of \vec{w} turning toward the input vector \vec{x} : $\vec{w} \leftarrow \vec{w} + \eta(t - o)\vec{x}$.
- Adjustment of the bias t moves the line closer or away from the origin.

20

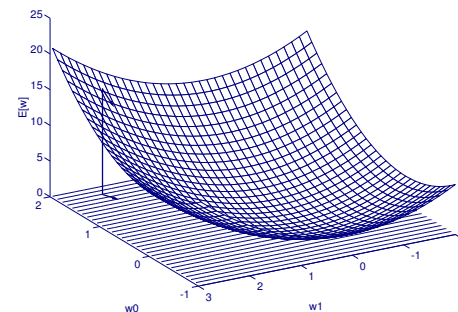
Another Learning Rule: Delta Rule

- The perceptron rule cannot deal with noisy data.
- The delta rule will find an approximate solution even when input set is not linearly separable.
- Use **linear unit** without the step function: $o(\vec{x}) = \vec{w} \cdot \vec{x}$.
- Want to reduce the error by adjusting \vec{w} :

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

21

Gradient Descent



- Want to minimize by adjusting \vec{w} : $E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$
- Note: the error surface is defined by the training data D . A different data set will give a different surface.
- $E(w_0, w_1)$ is the error function above, and we want to change (w_0, w_1) to position under a low E .

22

Gradient Descent (Cont'd)

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:

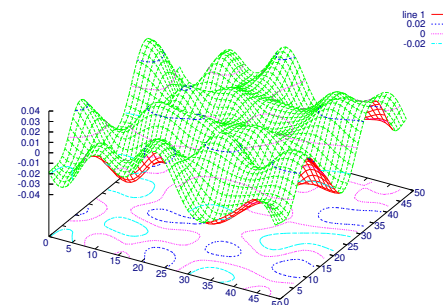
$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

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

23

Gradient Descent (Example)



- Gradient points in the **maximum increasing direction**.
- Gradient is perpendicular to the level curve (uphill direction).
- $E(w_0, w_1)$ is the error function above, so $\nabla E = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1} \right)$, a vector on a 2D plane.

24

Gradient Descent (Cont'd)

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

Since we want $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$, $\Delta w_i = \eta \sum_d (t_d - o_d) x_{i,d}$.

25

Gradient Descent Properties

Gradient descent is effective in searching through a large of infinite H :

- H contains continuously parameterized hypotheses, and
- the error can be **differentiated** wrt the parameters.

Limitations:

- convergence can be slow, and
- finds local minima (global minimum not guaranteed).

27

Gradient Descent: Summary

Gradient-Descent (*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - * Input the instance \vec{x} to the unit and compute the output o
 - * For each linear unit weight w_i , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

- For each linear unit weight w_i , Do

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

26

Stochastic Approximation to Grad. Desc.

Avoiding local minima: Incremental gradient descent, or stochastic gradient descent.

- Instead of weight update based on **all** input in D , immediately update weights after each input example:

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

instead of

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)x_i,$$

- Can be seen as minimizing error function

$$E_d(\vec{w}) = \frac{1}{2} (t_d - o_d)^2.$$

28

Standard and Stochastic Grad. Desc.: Differences

- In the standard version, error is defined over entire D .
- In the standard version, more computation is needed per weight update, but η can be larger.
- Stochastic version can **sometimes** avoid local minima.

29

Exercise: Implementing the Perceptron

- It is fairly easy to implement a perceptron.
- You can implement it in any programming language: C/C++, etc.
- Look for examples on the web, and JAVA applet demos.

31

Summary

Perceptron training rule guaranteed to succeed if

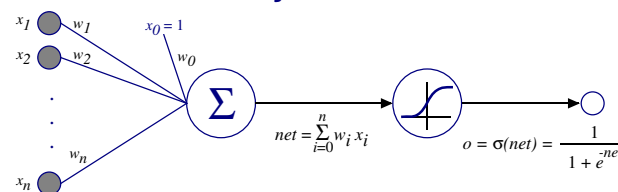
- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule using gradient descent

- Asymptotic convergence to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

30

Multilayer Networks



- Differentiable threshold unit: **sigmoid**

$$\sigma(y) = \frac{1}{1 + \exp(-y)}.$$

Interesting property: $\frac{d\sigma(y)}{dy} = \sigma(y)(1 - \sigma(y))$.

- Output:

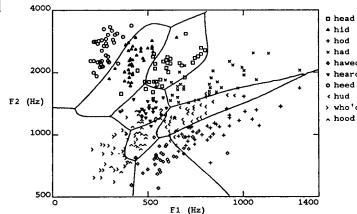
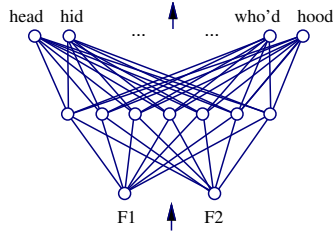
$$o = \sigma(\vec{w} \cdot \vec{x})$$

- Other functions:

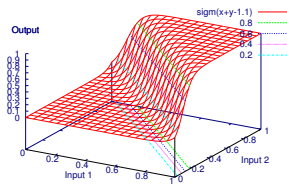
$$\tanh(y) = \frac{\exp(-2y) - 1}{\exp(-2y) + 1}$$

32

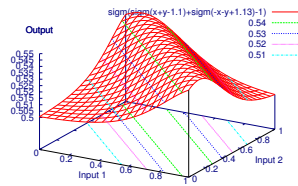
Multilayer Networks and Backpropagation



- Nonlinear decision surfaces.



(a) One output



(b) Two hidden, one output

- Another example: XOR

33

Error Gradient for a Sigmoid Unit

$$\begin{aligned}
 \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
 &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
 &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
 &= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right) \\
 &= - \sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}
 \end{aligned}$$

34

Error Gradient for a Sigmoid Unit

From the previous page:

$$\frac{\partial E}{\partial w_i} = - \sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}$$

But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d(1 - o_d) x_{i,d}$$

35

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
 $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k$
 4. Update each network weight $w_{i,j}$
 $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$ where
 $\Delta w_{ji} = \eta \delta_j x_i$.

Note: w_{ji} is the weight from i to j (i.e., $w_{j \leftarrow i}$).

36

The δ Term

- For output unit:

$$\delta_k \leftarrow \underbrace{o_k(1 - o_k)}_{\sigma'(net_k)} \underbrace{(t_k - o_k)}_{\text{Error}}$$

- For hidden unit:

$$\delta_h \leftarrow \underbrace{o_h(1 - o_h)}_{\sigma'(net_h)} \underbrace{\sum_{k \in \text{outputs}} w_{kh} \delta_k}_{\text{Backpropagated error}}$$

- In sum, δ is the derivative times the error.
- Derivation to be presented later.

37

Derivation of Δw : Output Unit Weights

From the previous page, $\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$

- First, calculate $\frac{\partial E_d}{\partial net_j}$:

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$\begin{aligned} \frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 \\ &= \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 \\ &= 2 \frac{1}{2} (t_j - o_j) \frac{\partial (t_j - o_j)}{\partial o_j} \\ &= -(t_j - o_j) \end{aligned}$$

39

Derivation of Δw

- Want to update weight as:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}},$$

where error is defined as:

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$$

- Given $net_j = \sum_j w_{ji} x_i$,

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

- Different formula for output and hidden.

38

Derivation of Δw : Output Unit Weights

From the previous page,

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} = -(t_j - o_j) \frac{\partial o_j}{\partial net_j}$$

- Next, calculate $\frac{\partial o_j}{\partial net_j}$: Since $o_j = \sigma(net_j)$, and $\sigma'(net_j) = o_j(1 - o_j)$,

$$\frac{\partial o_j}{\partial net_j} = o_j(1 - o_j).$$

Putting everything together,

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} = -(t_j - o_j) o_j(1 - o_j).$$

40

Derivation of Δw : Output Unit Weights

From the previous page:

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} = -(t_j - o_j) o_j (1 - o_j).$$

Since $\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial \sum_k w_{jk} x_k}{\partial w_{ji}} = x_i,$

$$\begin{aligned} \frac{\partial E_d}{\partial w_{ji}} &= \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \\ &= \underbrace{-(t_j - o_j) o_j (1 - o_j)}_{\delta_j = \text{error} \times \sigma'(net)} \underbrace{x_i}_{\text{input}} \end{aligned}$$

41

Derivation of Δw : Hidden Unit Weights

Start with $\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_i:$

$$\begin{aligned} \frac{\partial E_d}{\partial net_j} &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\ &= \sum_{k \in \text{Downstream}(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\ &= \sum_{k \in \text{Downstream}(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\ &= \sum_{k \in \text{Downstream}(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j} \\ &= \sum_{k \in \text{Downstream}(j)} -\delta_k w_{kj} \underbrace{o_j (1 - o_j)}_{\sigma'(net)} \end{aligned} \quad (1)$$

42

Derivation of Δw : Hidden Unit Weights

Finally, given

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_i,$$

and

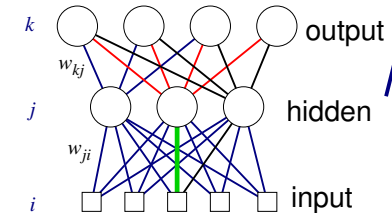
$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in \text{Downstream}(j)} -\delta_k w_{kj} \underbrace{o_j (1 - o_j)}_{\sigma'(net)},$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta \underbrace{[o_j (1 - o_j)]}_{\sigma'(net)} \underbrace{\left[\sum_{k \in \text{Downstream}(j)} \delta_k w_{kj} \right]}_{\text{error}} x_i$$

$\underbrace{\hspace{10em}}_{\delta_j}$

43

Extension to Different Network Topologies



- Arbitrary number of layers: for neurons in layer m :

$$\delta_r = o_r (1 - o_r) \sum_{s \in \text{layer } m+1} w_{sr} \delta_s.$$

- Arbitrary acyclic graph:

$$\delta_r = o_r (1 - o_r) \sum_{s \in \text{Downstream}(r)} w_{sr} \delta_s.$$

44

Backpropagation: Properties

- Gradient descent over entire *network* weight vector.
- Easily generalized to arbitrary directed graphs.
- Will find a local, not necessarily global error minimum:
 - In practice, often works well (can run multiple times with different initial weights).
- Often include weight *momentum* α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1).$$

- Minimizes error over *training* examples:
 - Will it generalize well to subsequent examples?
- Training can take thousands of iterations \rightarrow slow!
- Using the network after training is very fast.

45

H-Space Search and Inductive Bias

- H -space = n -D weight space (when there are n weights).
- The space is **continuous**, unlike decision tree or general-to-specific concept learning algorithms.
- Inductive bias:
 - Smooth interpolation between data points.

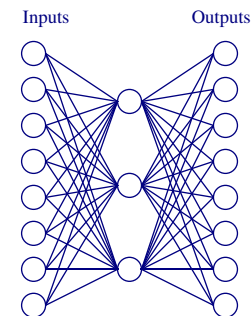
47

Representational Power of Feedforward Networks

- Boolean functions: every boolean function representable with two layers (hidden unit size can grow exponentially in the worst case: one hidden unit per input example, and “OR” them).
- Continuous functions: Every **bounded** continuous function can be approximated with an arbitrarily small error (output units are linear).
- Arbitrary functions: with three layers (output units are linear).

46

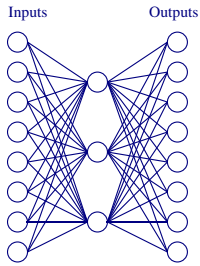
Learning Hidden Layer Representations



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

48

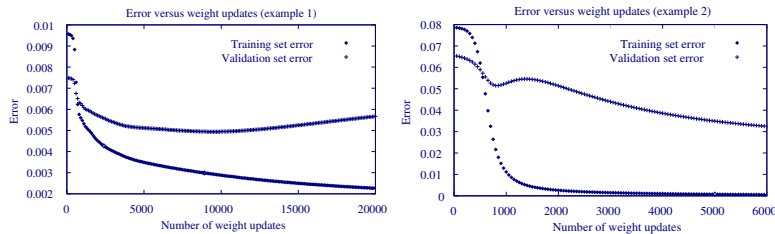
Learned Hidden Layer Representations



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

49

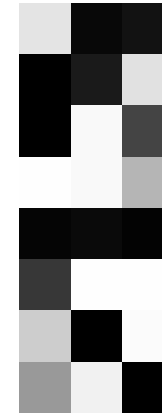
Overfitting



- Error in two different robot perception tasks.
- Training set and validation set error.
- Early stopping ensures good performance on unobserved samples, but must be careful.
- Weight decay, use of validation sets, use of k -fold cross-validation, etc. to overcome the problem.

51

Learned Hidden Layer Representations



- Learned encoding is similar to standard 3-bit binary code.
- Automatic discovery of **useful hidden layer representations** is a key feature of ANN.
- Note: The hidden layer representation is **compressed**.

50

Alternative Error Functions

Penalize large weights:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

Train on target slopes as well as values (when the slope is available):

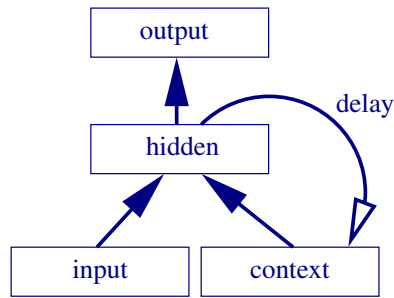
$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} \left[(t_{kd} - o_{kd})^2 + \mu \sum_{j \in \text{inputs}} \left(\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j} \right)^2 \right]$$

Tie together weights:

- e.g., in phoneme recognition network, or
- handwritten character recognition (weight sharing).

52

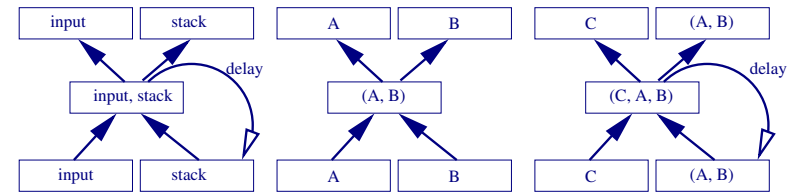
Recurrent Networks



- Sequence recognition.
- Store tree structure (next slide).
- Can be trained with plain backpropagation.
- Generalization may not be perfect.

53

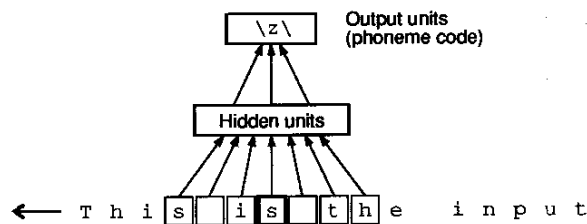
Recurrent Networks (Cont'd)



- Autoassociation (input = output)
- Represent a stack using the hidden layer representation.
- Accuracy depends on numerical precision.

54

Some Applications: NETtalk



- NETtalk: Sejnowski and Rosenberg (1987).
- Learn to pronounce English text.
- Demo
- Data available in UCI ML repository

55

NETtalk data

```
aardvark a-rdvark 1<<<>2<<0
aback xb@k-0>1<<0
abacus @bxkxs 1<0>0<0
abaft xb@ft 0>1<<0
abalone @bxloni 2<0>1>0 0
abandon xb@ndxn 0>1<>0<0
abase xbes-0>1<<0
abash xb@S-0>1<<0
abate xbet-0>1<<0
abatis @bxti-1<0>2<2
...
```

- Word – Pronunciation – Stress/Syllable
- about 20,000 words

56

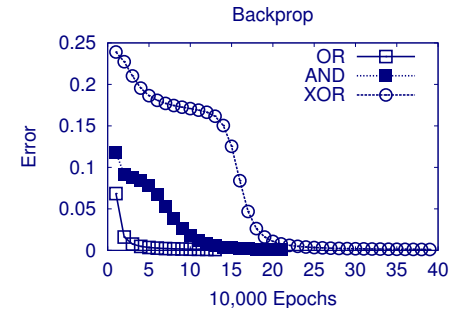
Backpropagation Exercise

- **URL:** <http://www.cs.tamu.edu/faculty/choe/src/backprop-1.6.tar.gz>
- Untar and read the README file:


```
gzip -dc backprop-1.6.tar.gz | tar
xvf -
```
- Run `make` to build (on departmental unix machines).
- Run `./bp conf/xor.conf` etc.

57

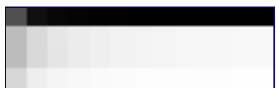
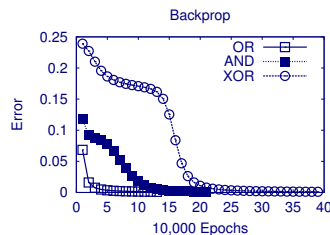
Backpropagation: Example Results



- Epoch: one full cycle of training through all training input patterns.
- OR was easiest, AND the next, and XOR was the most difficult to learn.
- Network had 2 input, 2 hidden and 1 output unit. Learning rate was 0.001.

58

Backpropagation: Example Results (cont'd)



OR



AND



XOR

Output to (0,0), (0,1), (1,0), and (1,1) form each row.

59

Backpropagation: Things to Try

- How does increasing the number of hidden layer units affect the (1) time and the (2) number of epochs of training?
- How does increasing or decreasing the learning rate affect the rate of convergence?
- How does changing the slope of the sigmoid affect the rate of convergence?
- Different problem domains: handwriting recognition, etc.

60

Summary

- ANN learning provides general method for learning real-valued functions over continuous or discrete-valued attributed.
- ANNs are robust to noise.
- H is the space of all functions parameterized by the weights.
- H space search is through gradient descent: convergence to local minima.
- Backpropagation gives novel hidden layer representations.
- Overfitting is an issue.
- More advanced algorithms exist.