Connectionist Models of Learning

Neural Networks

Characterized by:

- A large number of very simple neuronlike processing elements.
- A large number of weighted connections between the elements.
- Highly parallel, distributed control.
- An emphasis on learning internal representations automatically.

Slide CS478–1

Why Neural Nets?

Solving problems under the constraints similar to those of the brain may lead to solutions to AI problems that would otherwise be overlooked.

- Individual neurons operate very slowly.
- Neurons are failure-prone devices.
- Neurons promote approximate matching.

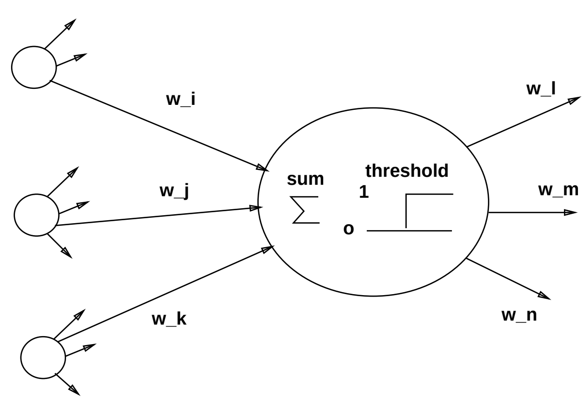
Slide CS478–2

Real Neurons

1. Threshold unit
2. Fire
3. Excitatory and inhibitory connections

Slide CS478–3

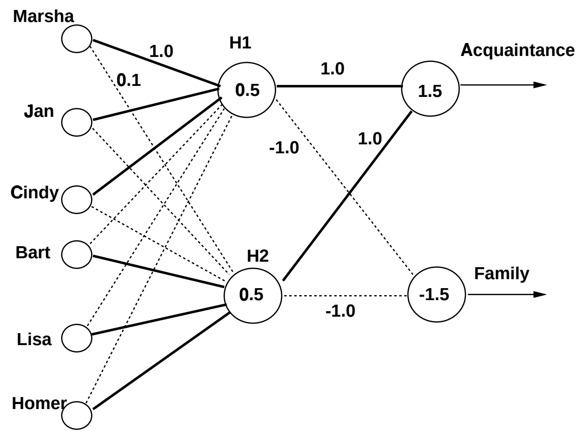
Simulated Neurons



Slide CS478–4

Using Feedforward Nets for Classification

Feedforward, layered, fully connected



Slide CS478–5

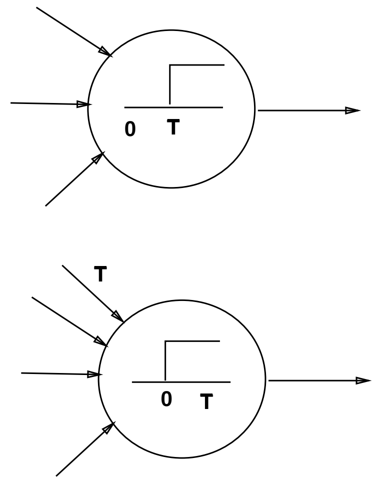
Backpropagation Procedure

Initialize weights. Until performance is satisfactory*,

1. Present all training instances. For each one,
   (a) Calculate actual output. (forward pass)
   (b) Compute the weight changes. (backward pass)
      i. Calculate error at output nodes. Compute adjustment to weights from hidden layer to output layer accordingly.
      ii. Calculate error at hidden layer. Compute adjustment to weights from initial layer to hidden layer accordingly.
2. Add up weight changes and change the weights.

Slide CS478–6

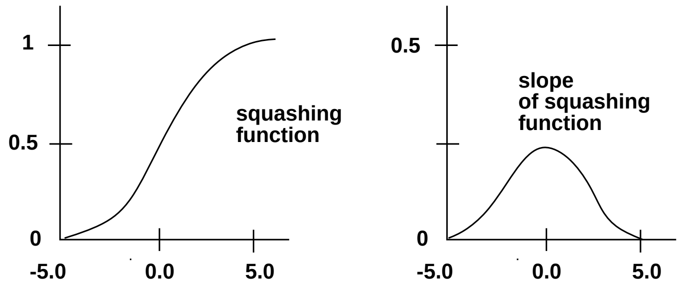
Learning Threshold Values



Slide CS478–7

Requires a Smooth Threshold Function

Because backpropagation updates all weights simultaneously, stair-step threshold function won't work.



Slide CS478–8

Adjusting the Weights

Make a large change to a weight, w, if the change leads to a large reduction in the errors observed at the output nodes.

d = desired value at output nodes
o = actual value at output nodes
error = $d - o$

Slide CS478–9

Adjusting the Weights

Let change in $w_{i \to j}$ be proportional to

- the slope of the threshold function at j
- the output at node i
- degree of error at j *(benefit)*
  - $\beta_z = d_z - o_z$
  - $\beta_j = \sum_k w_{j \to k} o_k (1 - o_k) \beta_k$
- learning rate r

Change to $w_{i \to j}$ should be proportional to $o_i o_j (1 - o_j) \beta_j$.

Slide CS478–10

The Backpropagation Procedure

Pick a rate parameter r.
Until performance is satisfactory,
For each training instance,

- Compute the resulting output.
- Compute $\beta = d_z - o_z$ for nodes in the output layer.
- Compute $\beta = \sum_k w_{j \to k}\ o_k(1 - o_k)\beta_k$ for all other nodes.
- Compute weight changes for all weights using

$$\Delta w_{i \to j} = r\ o_i\ o_j(1 - o_j)\beta_j$$

Add up weight changes for all training instances, and change the weights.

Slide CS478–11

Backpropagation Algorithm (Mitchell)

Initialize all weights to small random numbers. Until satisfied, do

For each training example, do

- Input the training example to the network and compute the network outputs
- For each output unit k

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

- For each hidden unit h

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

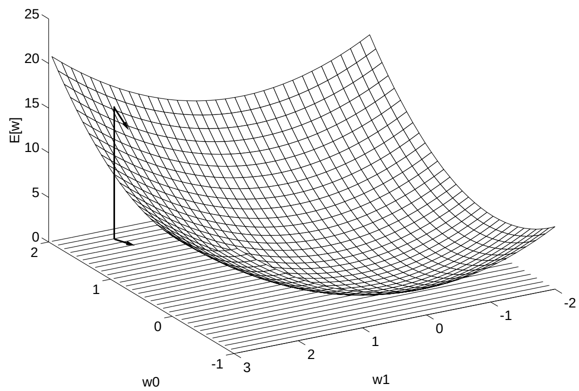
- Update each network weight $w_{i,j}$

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

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

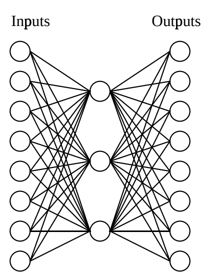
Slide CS478–12

Gradient Descent



Slide CS478–13

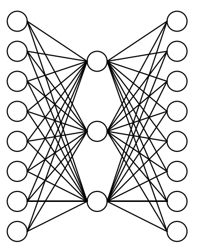
Learning Hidden Layer Representations



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

Slide CS478–14

Learning 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

Hidden Units

- **Hidden units** are nodes that are situated between the input nodes and the output nodes.
- Hidden units allow a network to learn non-linear functions.
- Hidden units allow the network to represent combinations of the input features.
- Given too many hidden units, a neural net will simply memorize the input patterns.
- Given too few hidden units, the network may not be able to represent all of the necessary generalizations.

When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete, real-valued, or a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

Slide CS478–17

More on Backpropagation

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

Slide CS478–18

Expressive Capabilities of ANNs

Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Slide CS478–19

Momentum

$$\Delta w_{ij}(t+1) = \eta \delta_j x_i + \alpha [w_{ij}(t) - w_{ij}(t-1)]$$

- A momentum factor, α, makes the n^{th} weight change partially dependent on the $(n-1)^{th}$ weight change. α ranges between 0 and 1.
- Momentum tends to keep the weight moving in the same direction, thereby improving convergence.
- Tends to increase the step size in regions where the gradient is unchanging, speeding convergence.
- Tends to avoid getting caught in small local minima and in oscillations about local minima.

Slide CS478–20

How many training pairs are needed?

This is a difficult question and depends on the problem, the training examples, and the network architecture. But a good rule of thumb is:

$$\frac{W}{P} = e$$

where W=# weights; P=# training pairs; e=error rate

For example, for $e = 0.1$, a net with 80 weights will require 800 training patterns to be assured of getting 90% of the test patterns correct (assuming it got 95% of the training patterns correct).

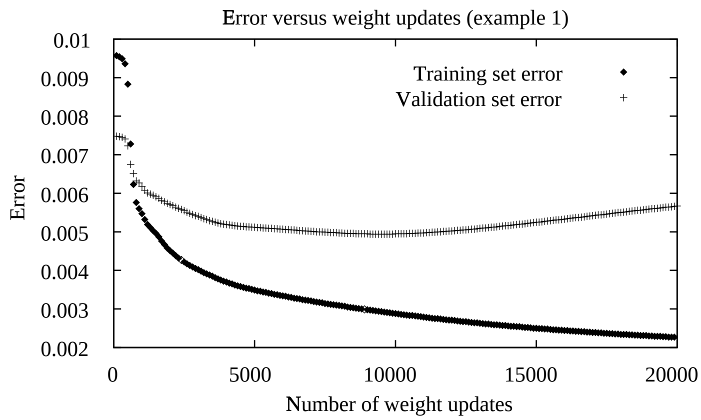
Slide CS478–21

How long should you train the net?

- The goal is to achieve a balance between correct responses for the training patterns and correct responses for new patterns. (That is, a balance between memorization and generalization.)
- If you train the net for too long, then you run the risk of overfitting.
- In general, the network is trained until it reaches an acceptable error rate (e.g., 95%).

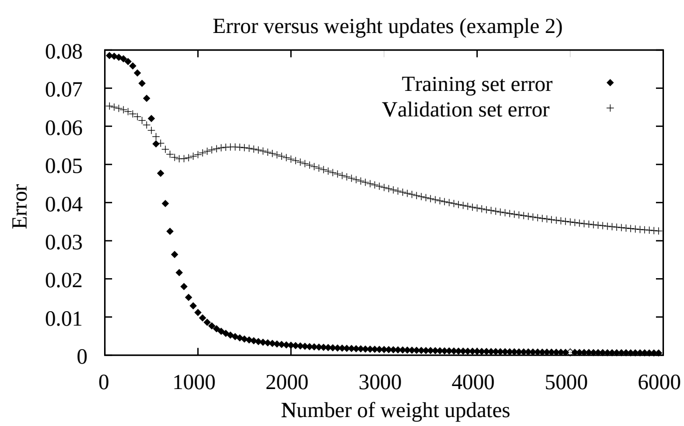
Slide CS478–22

Overfitting in ANNs



Slide CS478–23

Overfitting in ANNs



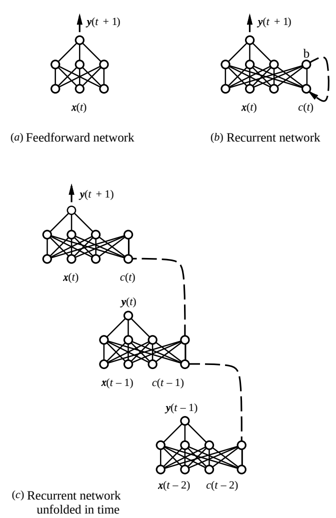
Slide CS478–24

Implementing Backprop – Design Decisions

1. Choice of r
2. Network architecture
   (a) How many hidden layers? how many hidden units per layer?
   (b) How should the units be connected? (Fully? Partial? Use domain knowledge?)
3. Stopping criterion – when should training stop?

Slide CS478–25

Recurrent Networks



(a) Feedforward network

(b) Recurrent network

(c) Recurrent network unfolded in time

Slide CS478–26