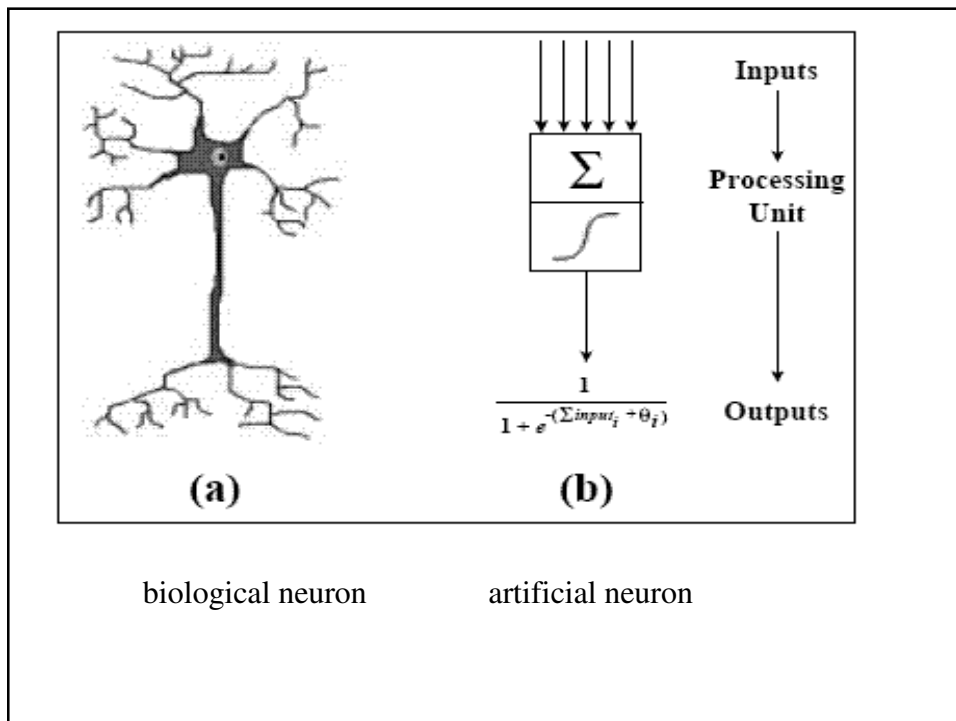
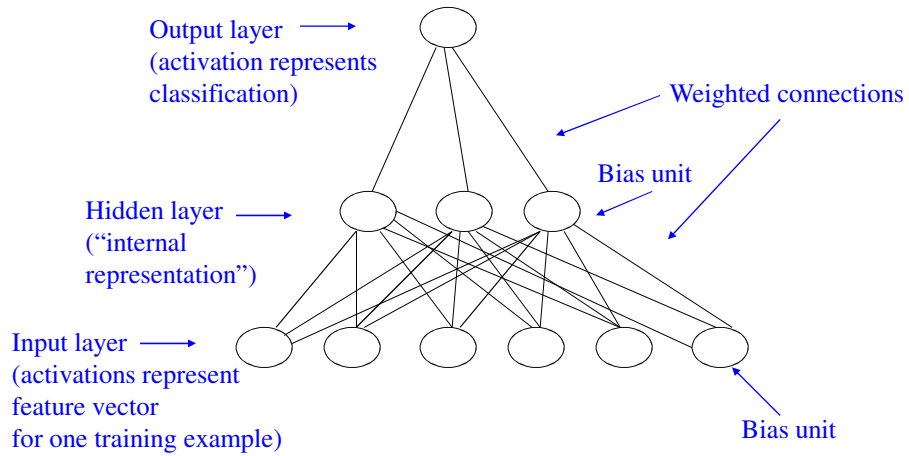


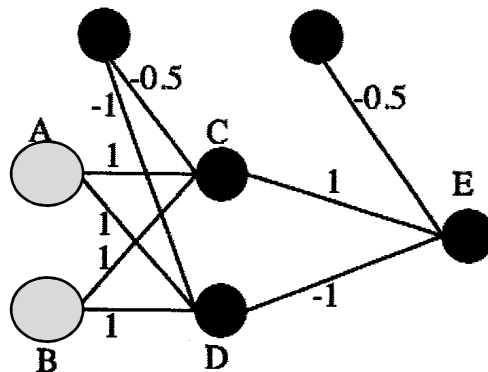
Neural Networks



A two-layer neural network

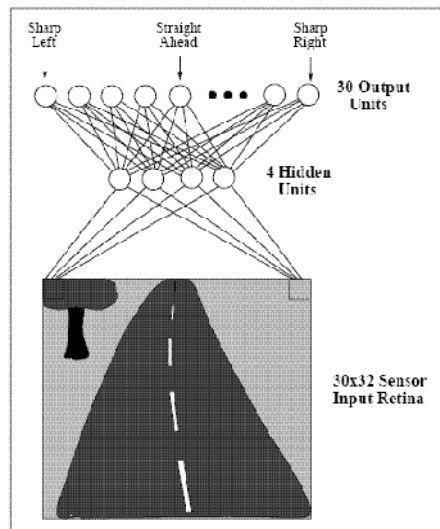
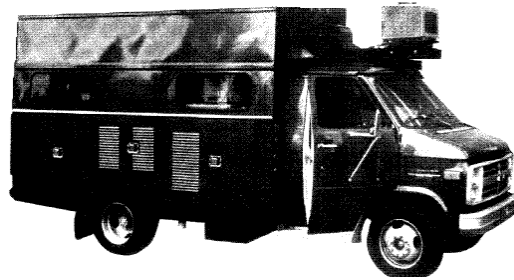


How to solve XOR with a multi-layer perceptron (MLP)



A Real-World Example: ALVINN (Pomerleau, 1993)

- ALVINN learns to drive an autonomous vehicle at normal speeds on public highways (!)
- Input: 30 x 32 grid of pixel intensities from camera



Each output unit correspond to a particular steering direction. The most highly activated one gives the direction to steer.

What kinds of problems are suitable for neural networks?

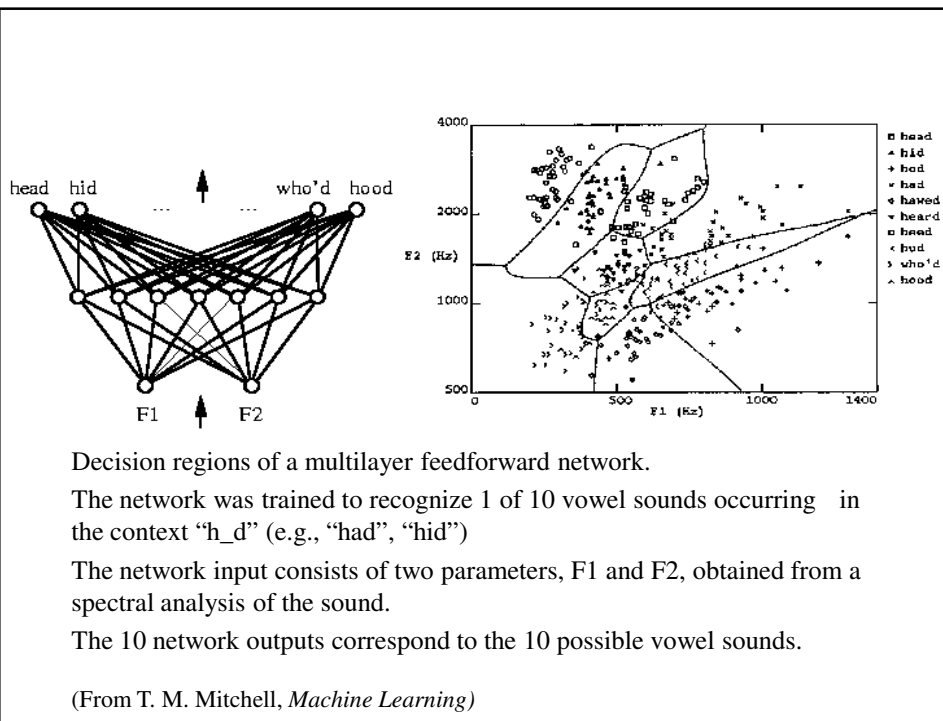
- Have sufficient training data
- Long training times are acceptable
- Not necessary for humans to understand learned target function or hypothesis

Advantages of neural networks

- Designed to be parallelized
- Robust on noisy training data
- Fast to evaluate new examples
- “Bad at logic, good at frisbee” (Andy Clark)

Multi-layer Perceptrons (MLPs)

- Single-layer perceptrons can only represent linear decision surfaces.
- Multi-layer perceptrons can represent non-linear decision surfaces



Differentiable Threshold Units

- In order to represent non-linear functions in general, need non-linear activation function at each unit.
- In order to do gradient descent on weights, need differentiable activation function.

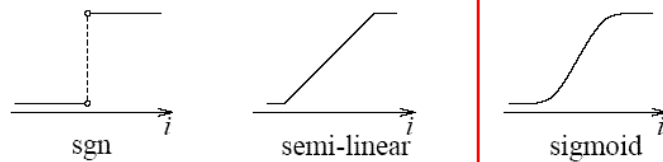


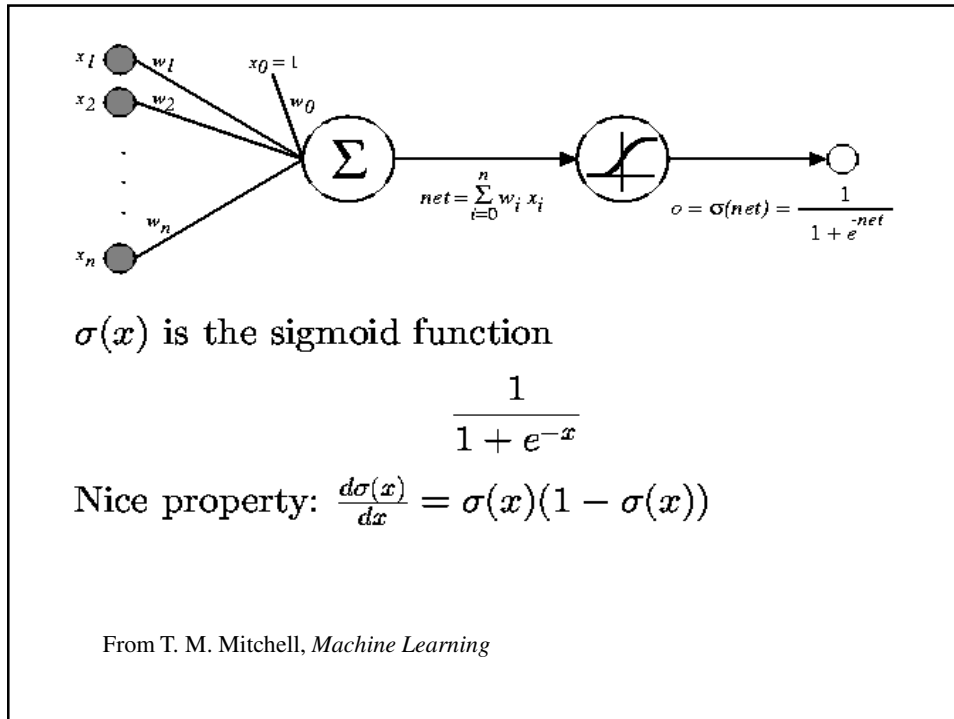
Figure 2.2: Various activation functions for a unit.

Sigmoid activation function:

$$o = \sigma(\mathbf{w} \cdot \mathbf{x}), \text{ where } \sigma(y) = \frac{1}{1 + e^{-y}}$$

To do gradient descent in multi-layer networks, need to generalize perceptron learning rule for non-linear activation functions.

(Recall, the linear activation function was what made the gradient of E turn out to be a simple, easy-to-calculate expression.)



Training multi-layer perceptrons

Assume two-layer networks:

I. For each training example:

1. Present input to the input layer.
2. Forward propagate the activations times the weights to each node in the hidden layer.

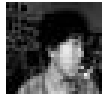
3. Forward propagate the activations times weights from the hidden layer to the output layer.
 4. At each output unit, determine the error E .
 5. Run the back-propagation algorithm to update all weights in the network.
- II. Repeat (I) for a given number of “epochs” or until accuracy on training or test data is acceptable.

Example: Face recognition

(From T. M. Mitchell, *Machine Learning*, Chapter 4)

- **Task:** classify camera images of various people in various poses.
- **Data:** Photos, varying:
 - Facial expression: *happy, sad, angry, neutral*
 - Direction person is facing: *left, right, straight ahead, up*
 - Wearing sunglasses?: *yes, no*

Within these, variation in background, clothes, position of face for a given person.



an2i_left_angry_open



an2i_right_sad_sunglasses



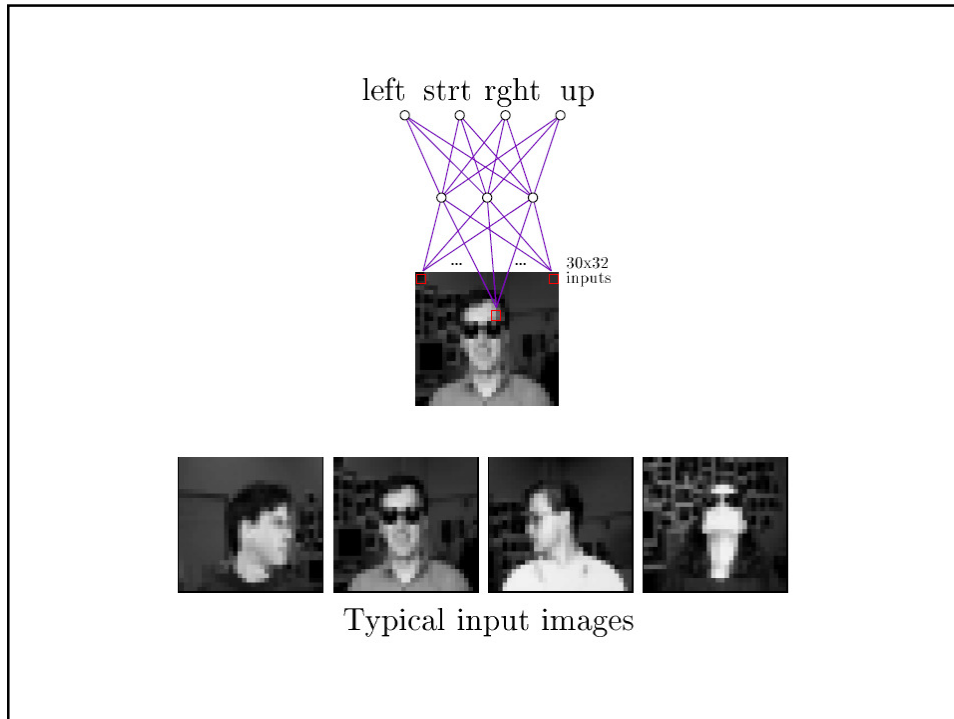
glickman_left_angry_open



phoebe_up_sad_sunglasses

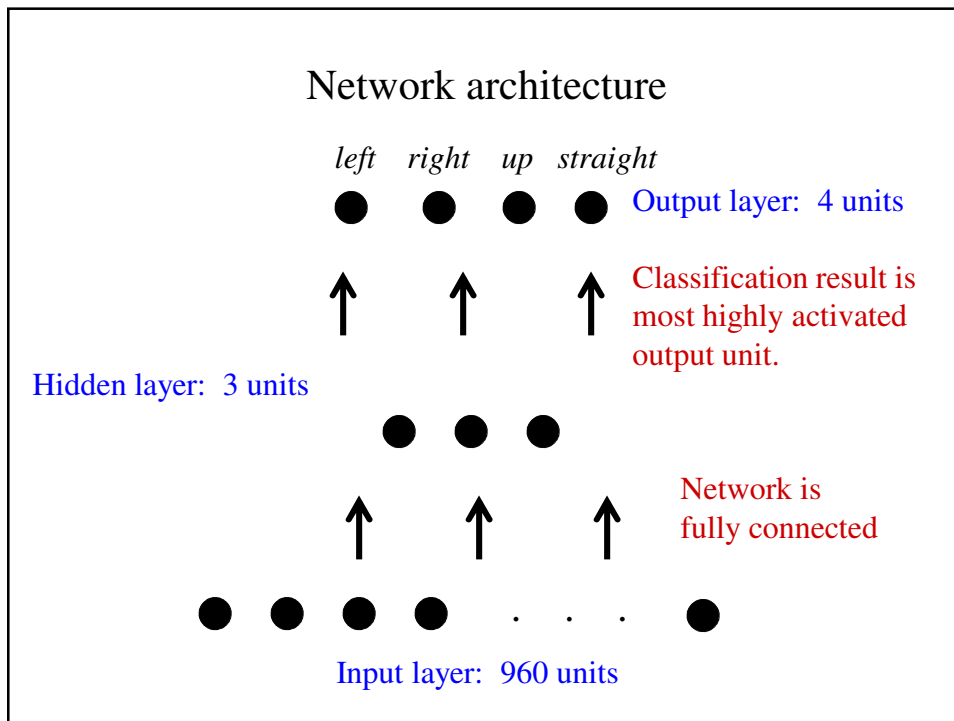
Design Choices

- Input encoding
- Output encoding
- Network topology:
 - How many hidden layers do we need? (Most of the time, just one.)
 - How many hidden nodes do we need? (Enough to represent the target function, but few enough to avoid overfitting. Hard to figure out what number to use!)
- Learning rate
- Momentum (add some contribution from previous weight change to help avoid local minima)



- Preprocessing of photo:
 - Create 30x32 coarse resolution version of 120x128 image
 - This makes size of neural network more manageable
- Input to neural network:
 - Photo is encoded as $30 \times 32 = 960$ pixel intensity values, scaled to be in $[0,1]$
 - One input unit per pixel
- Output units:
 - Encode classification of input photo

- Possible target functions for neural network:
 - Direction person is facing
 - Identity of person
 - Gender of person
 - Facial expression
 - etc.
- As an example, consider target of “direction person is facing”.



Target function

- Target function is:
 - Output unit should have activation 0.9 if it corresponds to correct classification
 - Otherwise output unit should have activation 0.1
- Use these values instead of 1 and 0, since sigmoid units can't produce 1 and 0 activation.

Other parameters

- Learning rate $\eta = 0.3$
- Momentum $\alpha = 0.3$
- If these are set too high, training fails to converge on network with acceptable error over training set.
- If these are set too low, training takes much longer.

Training

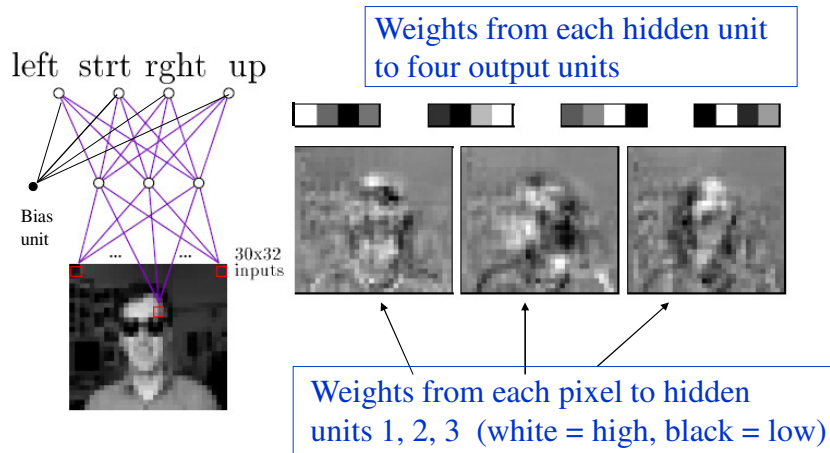
- For maximum of M epochs:
 - For each training example
 - Input photo to network
 - Propagate activations to output units
 - Determine error in output units
 - Adjust weights using back-propagation algorithm
 - Test accuracy of network on validation set. If accuracy is acceptable, stop algorithm.
- Demo of code

Examples of under/overfitting in this model

- overtrain
- undertrain
- model not complex enough
- model too complex

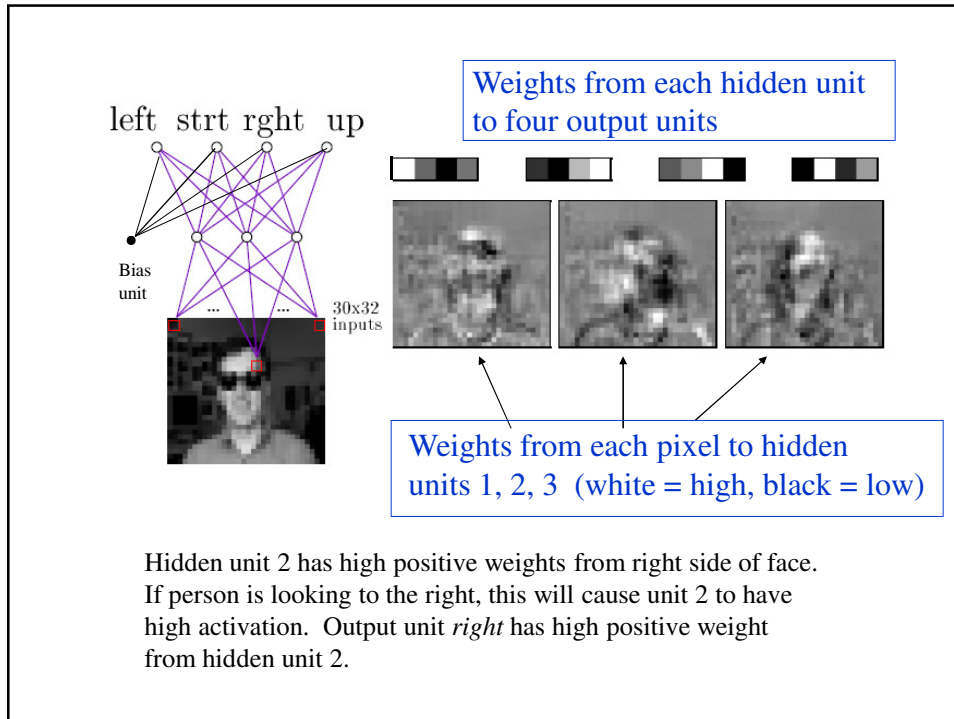
Understanding weight values

- After training:
 - Weights from input to hidden layer: high positive in certain facial regions
 - “Right” output unit has strong positive from second hidden unit, strong negative from third hidden unit.
 - Second hidden unit has positive weights on right side of face (aligns with bright skin of person turned to right) and negative weights on top of head (aligns with dark hair of person turned to right)
 - Third hidden unit has negative weights on right side of face, so will output value close to zero for person turned to right.



After 100 epochs of training.

(From T. M. Mitchell, Machine Learning)



Multi-layer networks can do everything

- **Universal approximation theorem:** One layer of hidden units suffices to approximate any function with finitely many discontinuities to arbitrary precision, if the activation functions of the hidden units are non-linear.
- In most applications, two-layer network with sigmoid activation function used.

Back-propagation

- Extends gradient descent algorithm to multi-layer network with (possibly) multiple output units.

Neural Network training algorithm (stochastic version)

- Assume you are given a set S of training examples (\mathbf{x}, \mathbf{t}) .
- Denote the weight from unit m to unit n by w_{mn} .
- Assume a network with one input layer, one hidden layer and one output layer.

In the following,

- input units will be indexed by i
- hidden units will be indexed by j
- output units will be indexed by k

- Initialize the network weights \mathbf{w} to small random numbers.
- For E epochs:
 - For each $(\mathbf{x}, \mathbf{t}) \in S$, Do:
 1. *Propagate the input forward:*
 - Input \mathbf{x} to the network and compute the output a_j of every hidden unit j in the network.
 - Input the hidden-layer activations a_j to the output layer, and compute the output y_k of every output unit k in the network

2. *Propagate the errors backward:*

- For each output unit k , calculate error term δ_{ok} :

$$\delta_{ok} \leftarrow y_k(1 - y_k)(t_k - y_k)$$

- For each hidden unit j , calculate error term δ_{hj} :

$$\delta_{hj} \leftarrow a_j(1 - a_j) \sum_{k \in \text{output units}} w_{jk} \delta_{ok}$$

3. *Update the weights :*

- For each weight in the network w_{mn} :

$$w_{mn} \leftarrow w_{mn} + \Delta w_{mn}$$

where:

for input-to-hidden-layer weights

$$\Delta w_{ij} = \eta \delta_{hj} x_i$$

for hidden-to-output-layer weights

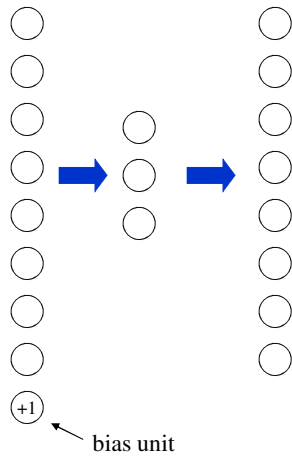
$$\Delta w_{jk} = \eta \delta_{ok} a_j$$

Will this algorithm converge on optimal weights?

- Not always, because of local minima in error surface.
- “In many practical applications the problem of local minima has not been found to be as severe as one might fear.”
- Especially true with large networks, since many dimensions give many “escape routes”.
- But “no methods are known to predict with certainty when local minima will cause difficulties.”

Hidden-layer representation in multi-layer perceptrons

“Auto-associator network”



Input	Hidden activations	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .15 .99 .99	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .01 .11 .88	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001

Results after network was trained for 5000 epochs. What is the encoding?

- One advantage of neural networks is that “high-level feature extraction” can be done automatically!

(Similar idea in face-recognition task. Note that you didn’t have to extract high-level features ahead of time.)

Heuristics for avoiding local minima in weight space

- Use stochastic gradient descent rather than true gradient descent. There is a different “error surface” for each training example. This adds “noise” to search process which can prevent it from getting stuck in local optima.
- Train multiple networks using the same data, but start each one with different random weights. If different weight vectors are found after training, select best one using separate validation set, or use ensemble of networks to vote on correct classification.
- Add “momentum” term to weight update rule.
- Simulated annealing
- Boosting

Learning rate and momentum

- Recall that stochastic gradient descent approaches “true” gradient descent as learning rate $\eta \rightarrow 0$.
- For practical purposes: choose η large enough so that learning is efficient, but small enough that a good approximation to true gradient descent is maintained, and oscillations are not reached.
- To avoid oscillations at large η , introduce *momentum*, in which change in weight is dependent on past weight change:

$$\Delta w_{i_j}(n) = \eta \delta_{ij} x_i + \alpha \Delta w_{i_j}(n-1)$$

where n is the iteration through the main loop of backpropagation.

The idea is to keep weight changes moving in the same direction.

Regularization (“Alternative Error Functions”)

- “Regularization” means to modify the error function to constrain the network in various ways.
- Examples:
 - Add a penalty term for the magnitude of the weight vector (to decrease overfitting):

$$E(\mathbf{w}) = \text{original squared error} + \gamma \sum_{i,j} w_{ij}^2$$

Data Sets

- **Training set:** Used to optimize the parameters, given a particular learning algorithm and model class
- **Validation set:** Used to optimize “hyper-parameters” (or “meta-parameters”) of the learning algorithm or the model class.
- **Test set:** Used at end, to report accuracy of resulting classifier.

Choosing Meta-Parameters using a validation data set

- how long to train
- how many hidden units to use
- what activation function to use
- other

Problem: Typically, not enough data for validation sets

One approach: k -fold cross-validation

1. Partition data D_0 into k disjoint validation sets, T_1, T_2, \dots, T_k of equal size, where this size is at least 30.
2. For i from 1 to k do
 - use T_i for the validation set, and the remaining data for training set S_i
3. Average the accuracy results from the different T_i

Typical value of k : 10

Neural Networks and Dimensionality Reduction

Example 1: Cascade Correlation Architecture

(Fahlman & Lebiere, 1991)

- Motivation: Speed up learning in neural networks, reduce dimensionality
- Why is back-propagation so slow?
 - One reason: “**moving target problem**”: Each hidden unit is trying to evolve (via changing weights) into a useful feature detector, but all other hidden units are changing their weights at the same time. Hidden units can’t communicate directly. Takes a long time to settle down.

- **Cascade correlation:** Allows only one hidden unit to evolve at any given time.
 1. Start with input, bias, and output units but no hidden units.
 2. Train weights as well as possible, using perceptron learning rule.
 3. When no significant error reduction after certain number of training cycles, run network one last time over training set to measure error. If still too high, continue, otherwise stop.

4. Add one new hidden unit to the network. This hidden unit receives a connection from each original input unit and from every existing hidden unit.
5. Before connecting it to the output units, go through several passes through training data, adjusting weights after every pass, in order to maximize S (using gradient ascent), the covariance between the hidden unit's value and errors observed at output units.

6. When S stops improving, freeze input weights, attach new unit to outputs.
 7. Now train hidden-to-output weights (using delta rule) and go to 2.
- Advantages of Cascade Correlation algorithm:
 - Builds smaller networks
 - Learns fast
 - Useful for incremental learning (new training examples added to already-trained net).
 - No need for back-propagation (biologically implausible)

Example 2:
Feature selection in cascade correlation networks
(C2FS)

Backstrom & Caruana, 2006

- Notion of “wrapper-based” feature selection:
 - **External:** Use different combinations of input units with validation tests. Quite slow.
 - **Internal:** Selects features at same time hidden units are being added. Much faster.

- **C2FS Algorithm:**

- Start with completely unconnected NN with k_i inputs and k_o outputs.
- Add single hidden node, along with edge from single input node, and edges from hidden node to each output node.
- Train weights using backprop, with a validation set for stopping.
- Repeat for each of the k_i inputs independently, training k_i separate networks.

- Subsequent iterations:
 - Select best network from previous iteration. Add hidden unit. Add edge from node i to new hidden unit. Repeat creating k_i networks. Select best and repeat.
- This is a kind of “hill-climbing”
- This algorithm uses four sets of data:
 - training
 - early stopping (validation)
 - feature selection (validation)
 - final test

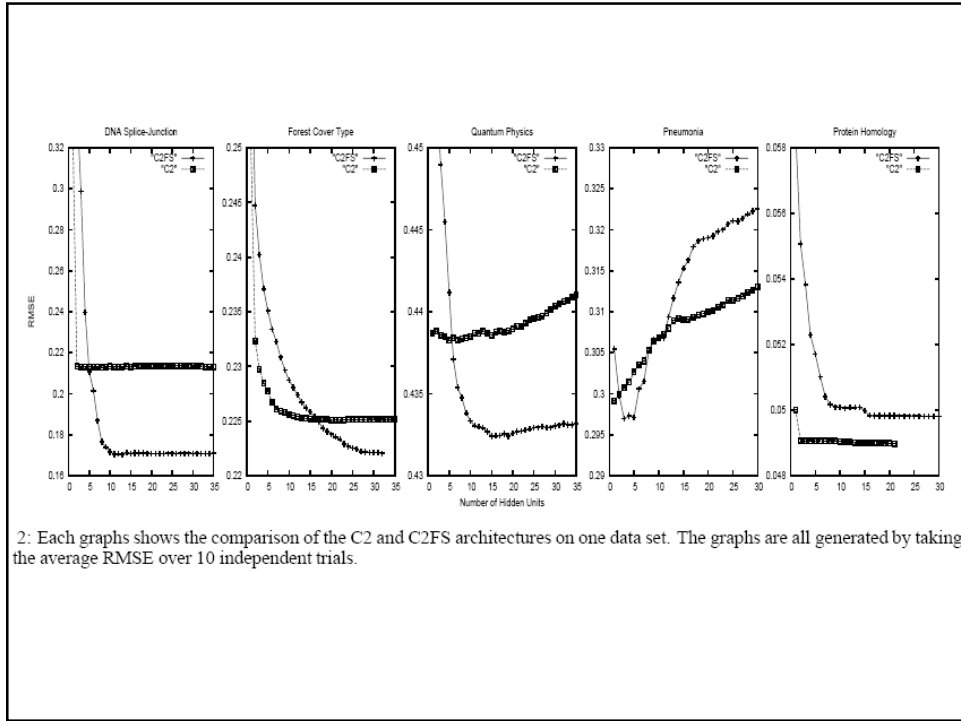
Results

1: Comparison of C2 and C2FS using t-Test for 10 trials. Each value is the average over 10 independent trials. Bold entries indicate differences that are different at $p = 0.05$. p values are also shown.

Data Set	Root-Mean-Squared-Error			ACCURACY		
	C2	C2FS	p-value	C2	C2FS	p-value
DNA Splice-Junction	0.2120	0.1667	0.000	0.9187	0.9525	0.000
Forest Cover Type	0.2249	0.2218	0.004	0.7585	0.7680	0.012
Quantum Physics	0.4374	0.4319	0.001	0.7040	0.7058	0.500
Pneumonia	0.2987	0.2920	0.187	0.8898	0.8932	0.358
Protein Homology	0.0489	0.0499	0.132	0.9974	0.9972	0.094

4: Mean number of features required by C2FS and C2FSGR. Each value is the average number of features required when the RMSE of the model stops improving. Models are evaluated separately in each trial, and then the mean is computed over the 10 trials.

Data Set	Tot # Feats	C2FS	C2FSGR	p-value
DNA	240	14.1	49.7	0.000
Forest	54	21.8	37.1	0.000
Quantum	78	7.4	52.6	0.000
Pneumonia	192	2.9	17.9	0.000
Protein	74	6.9	41.7	0.000
Mean	127.6	10.62	39.8	



Example 3 “Optimal Brain Damage” algorithm

Le Cun, Denker, & Solla, 1990

- Train large network, then prune inputs according to “saliency”

Other topics not covered here

- Other methods for training the weights
- Recurrent networks