Artificial Neural Networks

Reading:

• Neural nets: Mitchell chapter 4

Machine Learning 10-701

Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

Feb 1, 2010

Artificial Neural Networks to learn f: X → Y

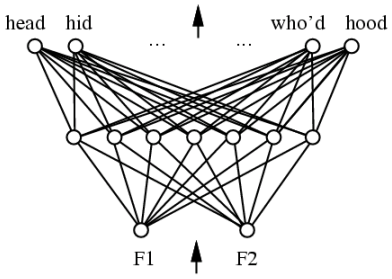
- f might be non-linear function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

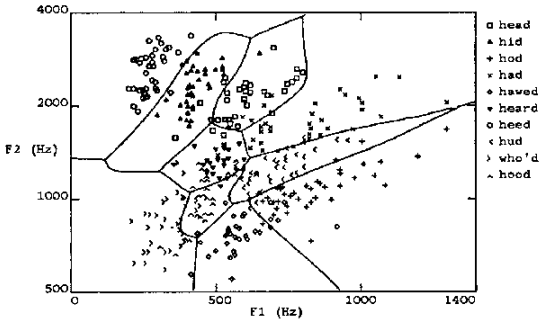
- Represent f by ***network*** of logistic units
- Each unit is a logistic function

$$unit\ output = \frac{1}{1 + exp(w_0 + \sum_i w_i x_i)}$$

- MLE: train weights of all units to minimize sum of squared errors of predicted network outputs

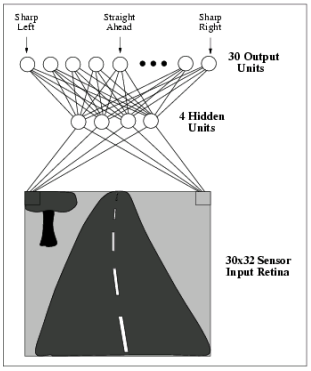
Multilayer Networks of Sigmoid Units

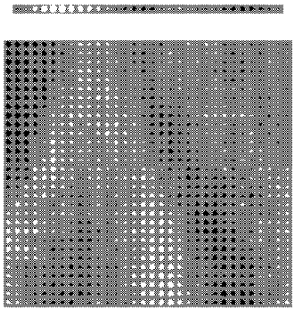




ALVINN

[Pomerleau 1993]





Connectionist Models

Consider humans:

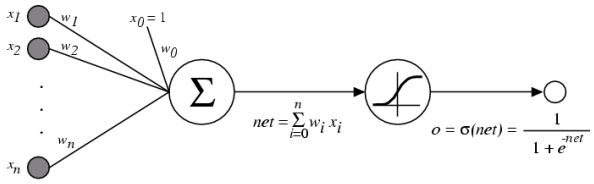
- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process

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

M(C)LE Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y

  $y = f(x) + \varepsilon$ ← assume noise N(0,σ_ε), iid

  deterministic

- Let's maximize the conditional data likelihood

$$W \leftarrow \arg\max_W \ln \prod_l P(Y^l | X^l, W)$$

$$W \leftarrow \arg\min_W \sum_l (y^l - \hat{f}(x^l))^2$$

Learned neural network

MAP Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y

  $y = f(x) + \varepsilon$ ← noise N(0,σ_ε)

  deterministic

Gaussian P(W) = N(0,σI)

$$W \leftarrow \arg\max_W \ln P(W) \prod_l P(Y^l | X^l, W)$$

$$W \leftarrow \arg\min_W \left[c \sum_i w_i^2 \right] + \left[\sum_l (y^l - \hat{f}(x^l))^2 \right]$$

ln P(W) ↔ c $\sum_i w_i^2$

Least Squares and MLE

Intuition: Signal plus (zero-mean) Noise model

$$Y = f^*(X) + \epsilon = X\beta^* + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$$

$$Y \sim \mathcal{N}(X\beta^*, \sigma^2 \mathbf{I})$$

$$\widehat{\beta}_{\text{MLE}} = \arg\max_{\beta} \underbrace{\log p(\{(X_i, Y_i)\}_{i=1}^n | \beta, \sigma^2)}_{\text{log likelihood}}$$

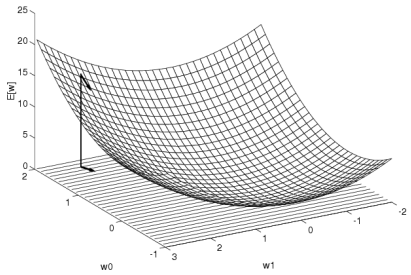
from previous lecture on linear regression

$$= \arg\min_{\beta} \sum_{i=1}^{n} (X_i\beta - Y_i)^2 = \widehat{\beta}$$

Least Square Estimate is same as Maximum Likelihood Estimate under a Gaussian model !

30

Gradient Descent



Gradient

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

Training rule:

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

i.e.,

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

Incremental (Stochastic) 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

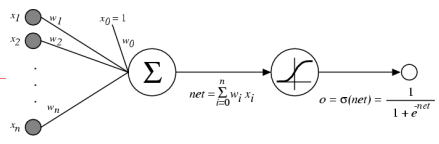
- For each training example d in D
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

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

$$E_d[\vec{w}] \equiv \frac{1}{2}(t_d - o_d)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

Error Gradient for a Sigmoid Unit



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

x_d = input

t_d = target output

o_d = observed unit output

w_i = weight i

But we know:

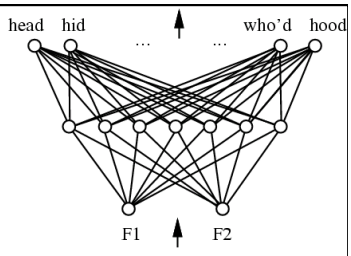
$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

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

So:

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

Backpropagation Algorithm (MLE)



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

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

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

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

where

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

x_d = input

t_d = target output

o_d = observed unit output

w_{ij} = wt from i to j

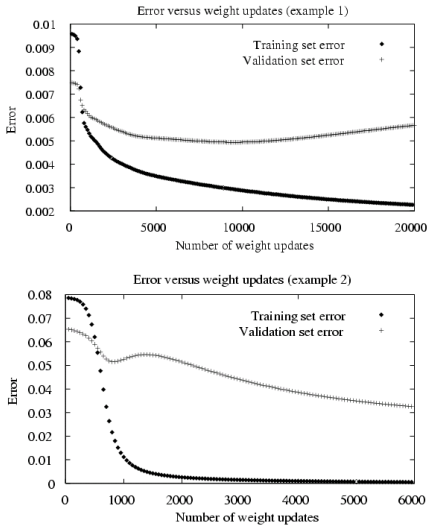
More on Backpropagation

- 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)
- Often include weight *momentum* α

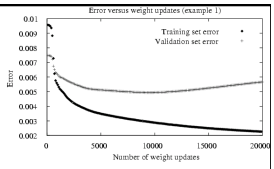
$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

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

Overfitting in ANNs



Dealing with Overfitting



Our learning algorithm involves a parameter

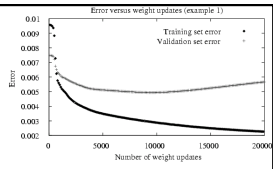
n=number of gradient descent iterations

How do we choose n to optimize future error?

(note: similar issue for logistic regression, decision trees, …)

e.g. the n that minimizes error rate of neural net over future data

Dealing with Overfitting



Our learning algorithm involves a parameter
n=number of gradient descent iterations
How do we choose n to optimize future error?

- Separate available data into training and validation set
- Use training to perform gradient descent
- n ← number of iterations that optimizes validation set error

→ *gives unbiased estimate of optimal n*
(but a biased estimate of future error)

K-Fold Cross Validation

Idea: train multiple times, leaving out a disjoint subset of data each time for validation. Average the validation set accuracies.

Partition data into K disjoint subsets
For k=1 to K
    validationData = kth subset
    h ← classifier trained on all data except for validationData
    accuracy(k) = accuracy of h on validationData
end
FinalAccuracy = mean of the K recorded accuracies

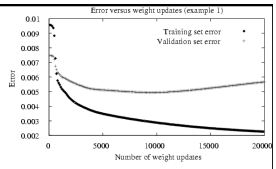
Leave-One-Out Cross Validation

This is just k-fold cross validation leaving out one example each iteration

---

Partition data into K disjoint subsets, *each containing one example*
For k=1 to K
    validationData = kth subset
    h ← classifier trained on all data except for validationData
    accuracy(k) = accuracy of h on validationData
end
FinalAccuracy = mean of the K recorded accuracies

Dealing with Overfitting



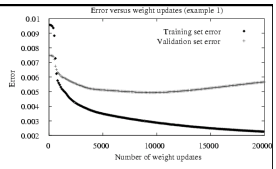
Our learning algorithm involves a parameter
    n=number of gradient descent iterations
How do we choose n to optimize future error?

- Separate available data into training and validation set
- Use training to perform gradient descent
- n ← number of iterations that optimizes validation set error

→ *gives unbiased estimate of optimal n*

How can we estimate the true error rate of the network trained with this choice of n?

Dealing with Overfitting



n=number of gradient descent iterations

Choosing n *and* obtaining unbiased est. of resulting true error:

- Separate available data into training, validation and test set
- Use training and validation sets to choose n
- Then use test set to obtain independent, unbiased estimate of the resulting true error rate

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].

Convergence of Backpropagation

Gradient descent to some local minimum

- Perhaps not global minimum...
- Add momentum
- Stochastic gradient descent
- Train multiple nets with different inital weights

Nature of convergence

- Initialize weights near zero
- Therefore, initial networks near-linear
- Increasingly non-linear functions possible as training progresses

Alternative Error Functions

Penalize large weights:

Original MLE error fn.

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

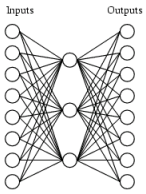
Train on target slopes as well as values:

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

Tie together weights:

- e.g., in phoneme recognition network

Learning Hidden Layer Representations



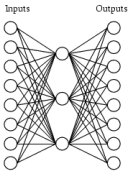
A target function:

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

Can this be learned??

Learning Hidden Layer Representations

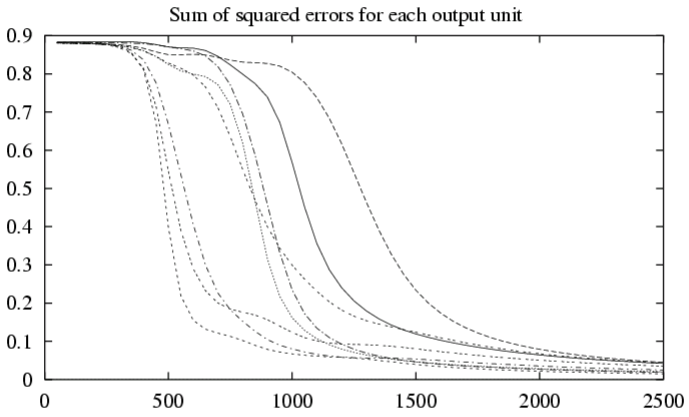
A network:



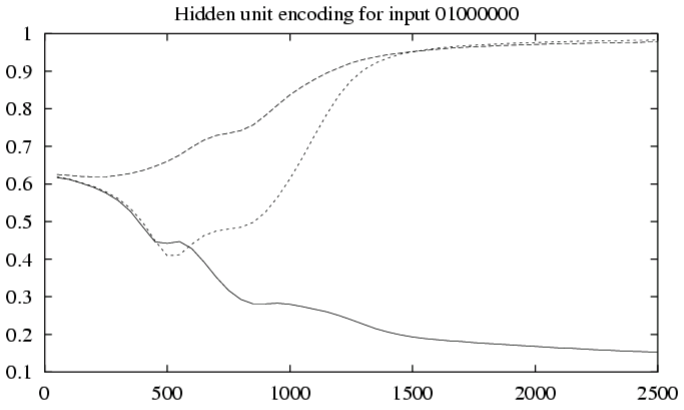
Learned hidden layer representation:

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

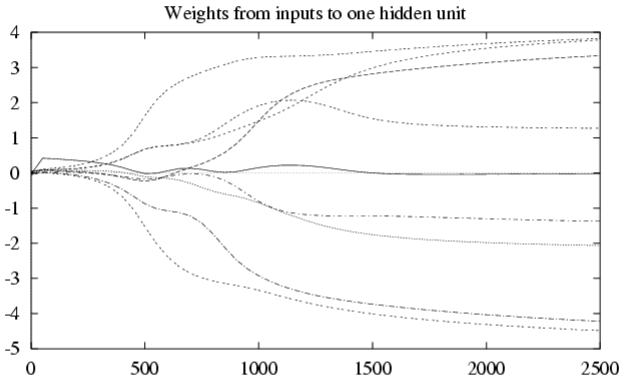
Training



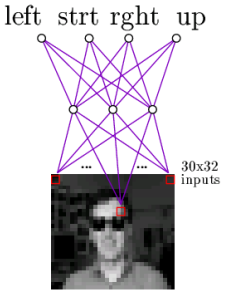
Training



Training



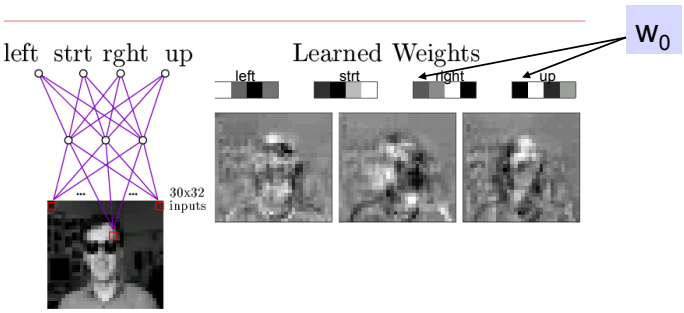
Neural Nets for Face Recognition



Typical input images

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

Learned Hidden Unit Weights

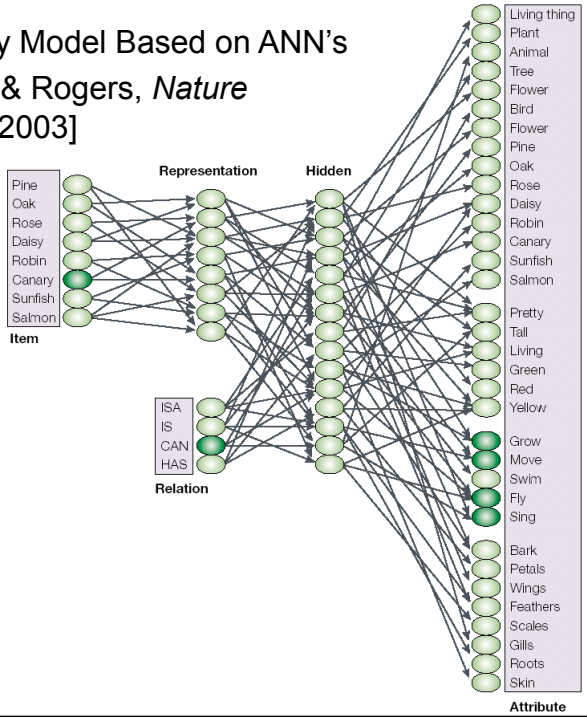


Typical input images

http://www.cs.cmu.edu/~tom/faces.html

Semantic Memory Model Based on ANN's

[McClelland & Rogers, *Nature* 2003]



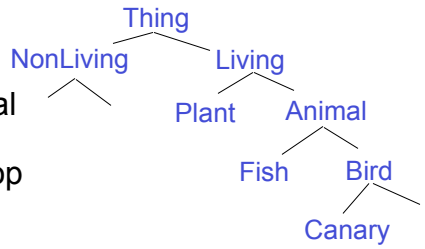
No hierarchy given.

Train with assertions,
e.g., Can(Canary,Fly)

Humans act as though they have a hierarchical memory organization

1. Victims of Semantic Dementia progressively lose knowledge of objects
   But they lose specific details first, general properties later, suggesting hierarchical memory organization

2. Children appear to learn general categories and properties first, following the same hierarchy, top down*.



Question: What learning mechanism could produce this emergent hierarchy?

* some debate remains on this.

Memory deterioration follows semantic hierarchy

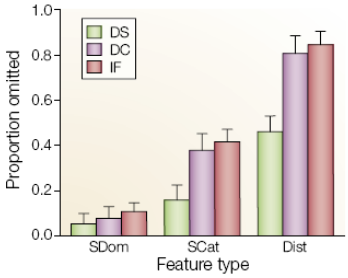
[McClelland & Rogers, *Nature* 2003]

a

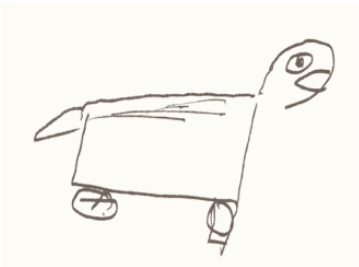
Picture naming responses for JL

Item	Sept. 91	March 92	March 93
Bird	+	+	Animal
Chicken	+	+	Animal
Duck	+	Bird	Dog
Swan	+	Bird	Animal
Eagle	Duck	Bird	Horse
Ostrich	Swan	Bird	Animal
Peacock	Duck	Bird	Vehicle
Penguin	Duck	Bird	Part of animal
Rooster	Chicken	Chicken	Dog

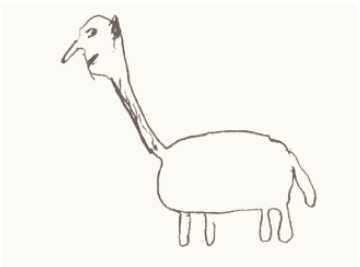
b



c IF's delayed copy of a camel



d DC's delayed copy of a swan



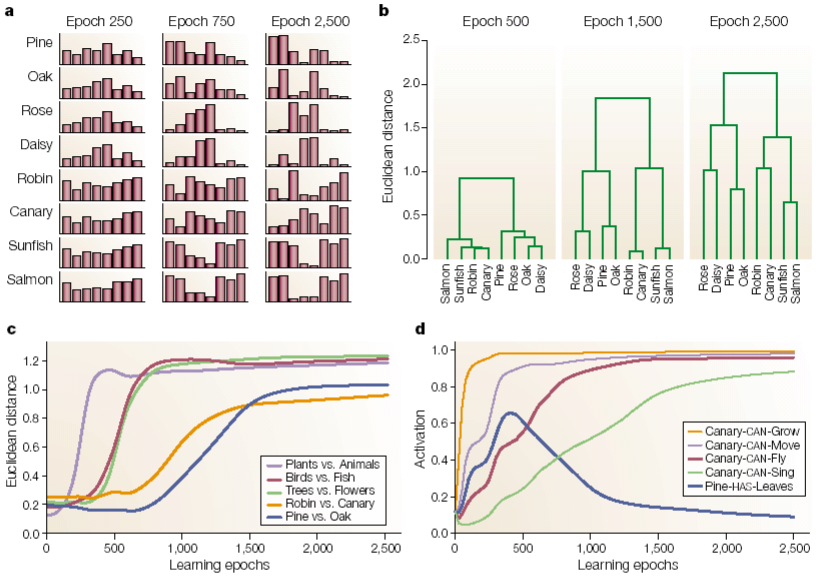
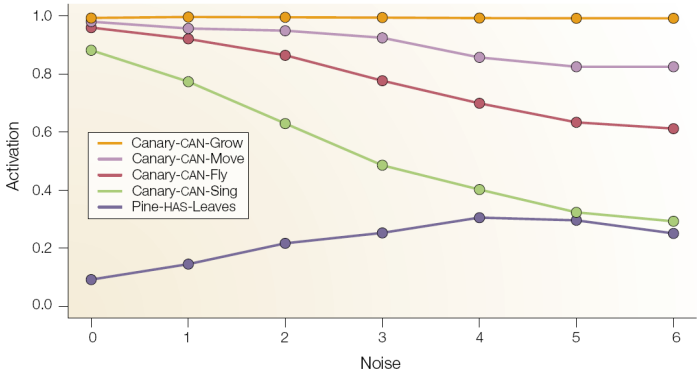


Figure 4 | **The process of differentiation of conceptual representations.** The representations are those seen in the feedforward network model shown in FIG. 3. **a** | Acquired patterns of activation that represent the eight objects in the training set at three points in the learning process (epochs 250, 750 and 2,500). Early in learning, the patterns are undifferentiated; the first difference to appear is between plants and animals. Later, the patterns show clear differentiation at both the superordinate (plant–animal) and intermediate (bird–fish/tree–flower) levels. Finally, the individual concepts are differentiated, but the overall hierarchical organization of the similarity structure remains. **b** | A standard hierarchical clustering analysis program has been used to visualize the similarity structure in the

ANN Also Models Progressive Deterioration

[McClelland & Rogers, *Nature* 2003]



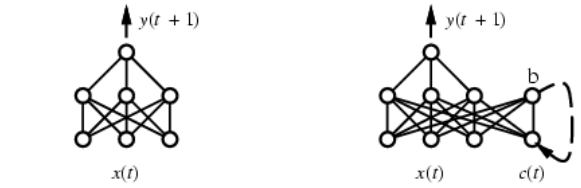
average effect of noise in inputs to hidden layers

Training Networks on Time Series

- Suppose we want to predict next state of world
  - and it depends on history of unknown length
  - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns

Training Networks on Time Series

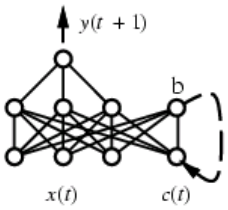
- Suppose we want to predict next state of world
  - and it depends on history of unknown length
  - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns
- Idea: use hidden layer in network to capture state history

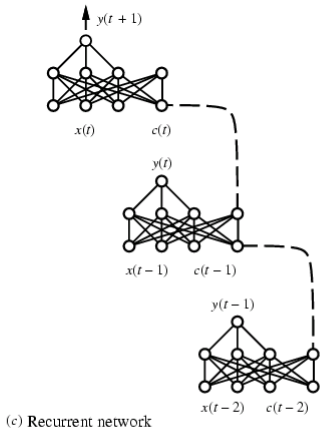


(a) Feedforward network (b) Recurrent network

Training Networks on Time Series

How can we train recurrent net??





(c) Recurrent network unfolded in time

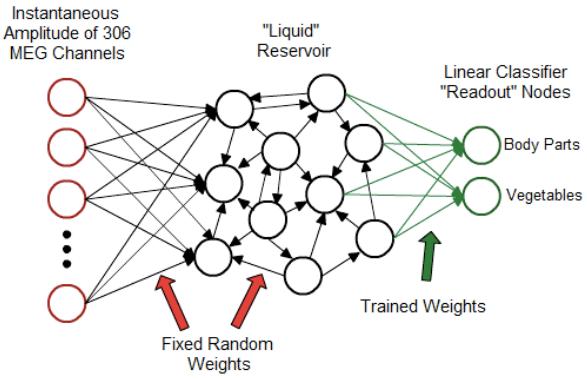
Training Networks on Time Series

- Training recurrent networks by unfolding in time is not very reliable
- Newer idea
  - randomly create a fixed (untrained) recurrent network
  - then train a classifier whose input is the network internal state, output is whatever you wish
  - suggested [Maas, et al., 2001] as model of cortical microcolumns
  - "liquid state" or "echo state" analogy to water reservoir
  - **NIPS 2007 Workshop on Liquid State Machines and Echo State Networks**

Example: LSM classifying MEG time series

[courtesy D. Pommerleau]

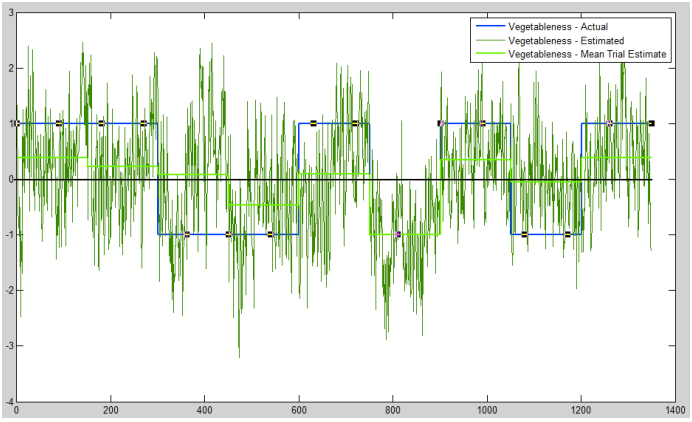
- Classify reading Vegetable words (e.g., corn) vs Body parts (e.g., arm)
- Input: 50 Hz, 100 fixed recurrent nodes in 'reservoir', train only outputs



Echo State Network Classifier
for 2 Category MEG Language Data

- Vegetable_out – Bodypart_out (green)
- Actual word type (blue)

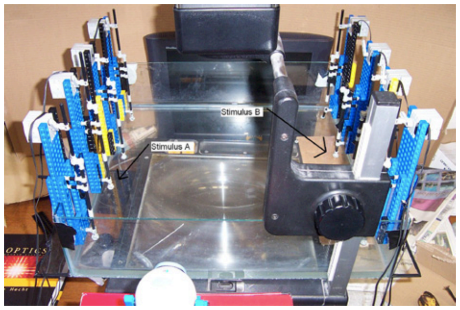
[courtesy D. Pommerleau]



Correct: 8/9 test cases

42

An Analog Implementation [Fernando&Sojakka, 2003]



Transparent water tank on overhead projector

Inputs: 8 motor-driven plungers (4 per side)

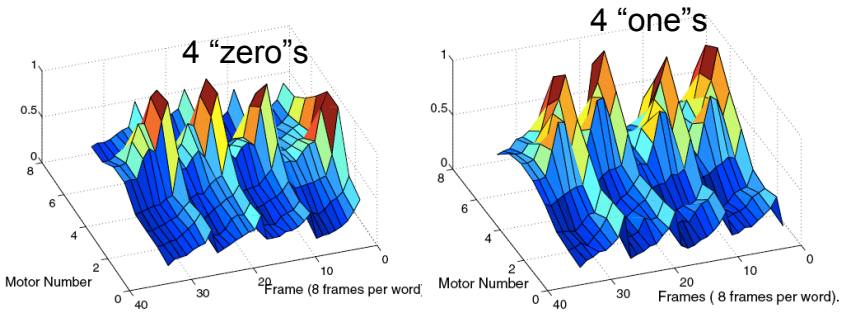
Reservoir output: image projected by overhead projector, captured by camera at 5 frames/second

LSM output: sum of outputs from 50 perceptrons trained to classify camera image

Speech recognition: words “one” vs. “zero”

[Fernando&Sojakka, 2003]

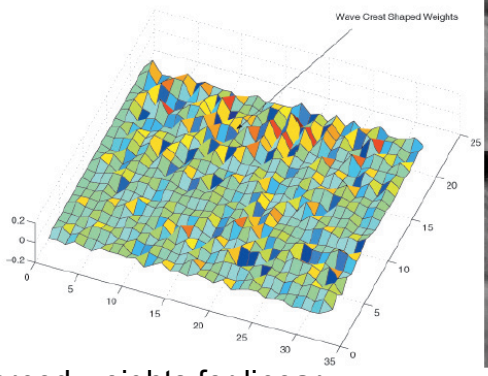
inputs to liquid state machine:



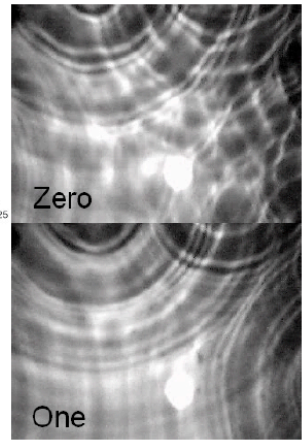
each motor a diff freq. band, updated every 500 msec

Speech recognition: “zero” vs. “one”

[Fernando&Sojakka, 2003]



learned weights for linear classifier of output image



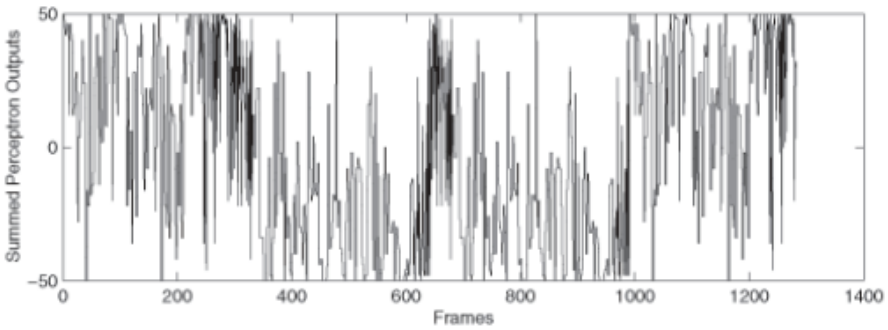
typical output images

An Analog Implementation

[Fernando&Sojakka, 2003]

Classifying individual output images (5/second, 20/word):

- training set: 2% error, test set: 35% error



test set

Artificial Neural Networks: Summary

- Highly non-linear regression/classification
- Vector-valued inputs and outputs
- Potentially millions of parameters to estimate
- Hidden layers learn intermediate representations
- Actively used to model distributed computation in brain

- Gradient descent, local minima problems
- Overfitting and how to deal with it

- Many extensions