Supervised Learning – Neural Networks (1)

COMP9417 Machine Learning and Data Mining

March 21, 2017

Acknowledgements

Material derived from slides for the book
"Elements of Statistical Learning (2nd Ed.)" by T. Hastie,
R. Tibshirani & J. Friedman. Springer (2009)
`http://statweb.stanford.edu/~tibs/ElemStatLearn/`

Material derived from slides for the book
"Machine Learning: A Probabilistic Perspective" by P. Murphy
MIT Press (2012)
`http://www.cs.ubc.ca/~murphyk/MLbook`

Material derived from slides for the book
"Machine Learning" by P. Flach
Cambridge University Press (2012)
`http://cs.bris.ac.uk/~flach/mlbook`

Material derived from slides for the book
"Bayesian Reasoning and Machine Learning" by D. Barber
Cambridge University Press (2012)
`http://www.cs.ucl.ac.uk/staff/d.barber/brml`

Material derived from slides for the book
"Machine Learning" by T. Mitchell
McGraw-Hill (1997)
`http://www-2.cs.cmu.edu/~tom/mlbook.html`

Material derived from slides for the course
"Machine Learning" by A. Srinivasan

BITS Pilani, Goa, India (2016)

Aims

This lecture will enable you to describe and reproduce machine learning approaches to the problem of numerical prediction. Following it you should be able to:

- define the perceptron
- describe the perceptron training algorithm
- describe the dual version of perceptron training
- reproduce the method of gradient descent for linear models
- define the problem of non-linear models for classification
- describe back-propagation training of a multi-layer perceptron neural network for non-linear classification

Introduction

We start by revisiting class of linear models that predict a numeric output . . .

. . . and move from there to consider scaling these up using networks of such models both for regression and classification

Artificial Neural Networks

- Gradient descent
- Multilayer networks
- Backpropagation
- Hidden layer representations
- Example: Face Recognition
- Advanced topics

Connectionist Models

Consider humans:

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

$\rightarrow$ much parallel computation

Connectionist Models

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

When to Consider Neural Networks

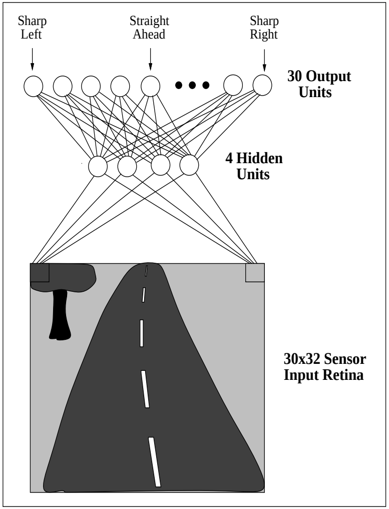
- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

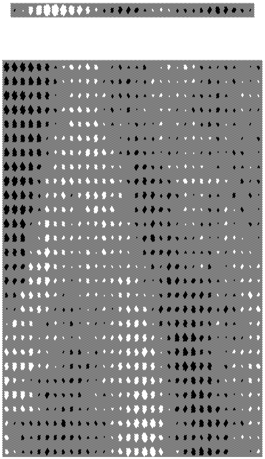
Examples:

- Speech phoneme recognition (NetTalk)
- Image classification (see face recognition data)
- many others . . .

ALVINN drives 70 mph on highways

ALVINN


Sharp Left
Straight Ahead
Sharp Right
30 Output Units
4 Hidden Units
30x32 Sensor Input Retina



Gradient Descent

To understand, consider simpler *linear unit*, where

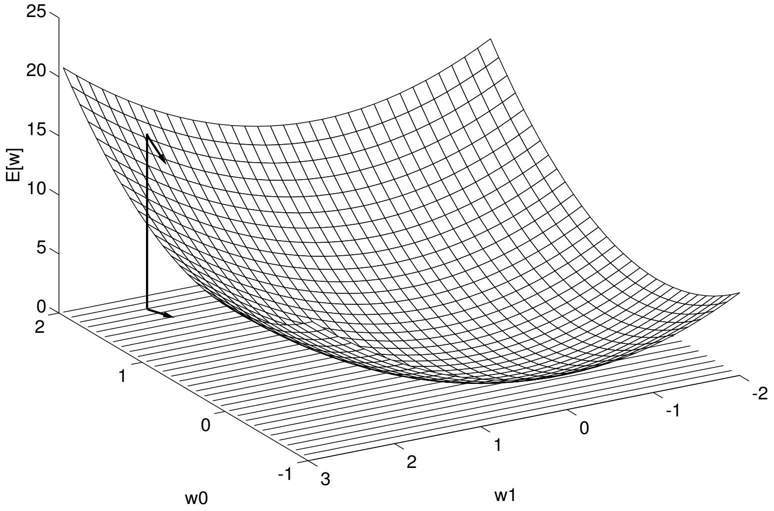
$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

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

Where D is set of training examples

Gradient Descent



Gradient Descent

Gradient

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

Training rule:

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

i.e.,

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

Gradient Descent

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

Gradient Descent

```
Gradient-Descent($training\_examples, \eta$)
Each training example is a pair $\langle \mathbf{x}, t \rangle$, where $\mathbf{x}$ is the vector of input values, and $t$ is the target output value. $\eta$ 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 $\Delta w_i$ to zero
    For each $\langle \mathbf{x}, t \rangle$ in $training\_examples$, Do
        Input the instance $\mathbf{x}$ to the unit and compute the output $o$
        For each linear unit weight $w_i$
            $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$
    For each linear unit weight $w_i$
            $w_i \leftarrow w_i + \Delta w_i$
```

Training Perceptron vs. Linear unit

Perceptron training rule guaranteed to succeed if

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

Linear unit training rule uses gradient descent

- Guaranteed to converge 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

Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied

- Compute the gradient $\nabla E_D[\mathbf{w}]$
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_D[\mathbf{w}]$

Incremental mode Gradient Descent:
Do until satisfied

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

Incremental (Stochastic) Gradient Descent

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

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

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

Very useful for online learning from data streams

Multilayer Networks of Sigmoid Units

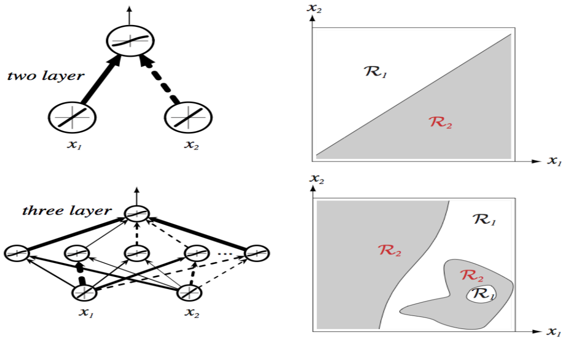
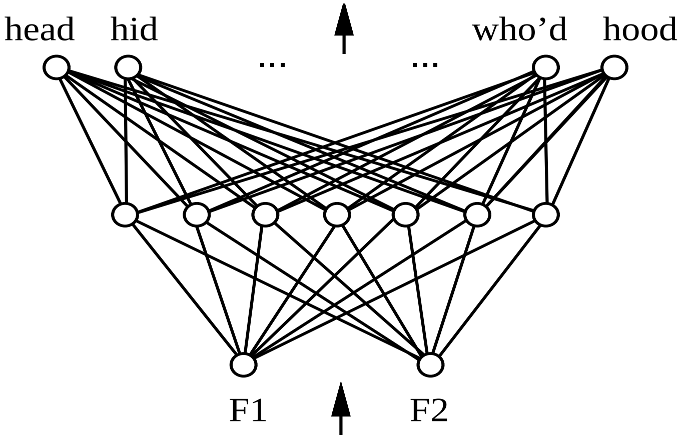
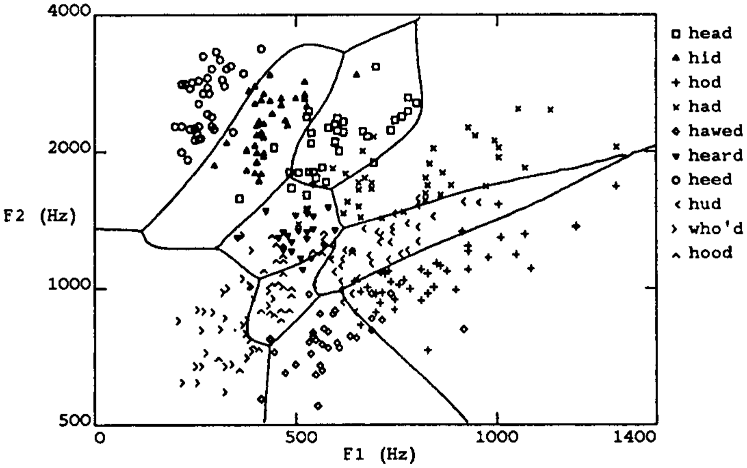


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

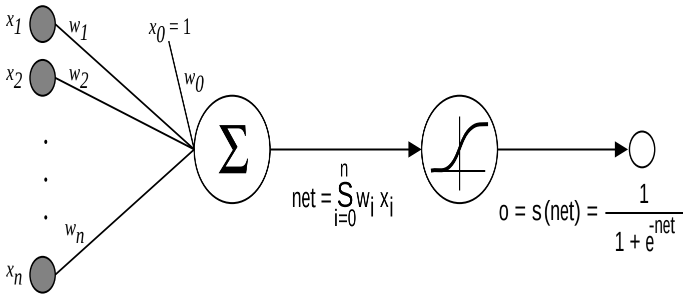
A Multilayer Network of Sigmoid Units for Speech Recognition: Model



A Multilayer Network of Sigmoid Units for Speech Recognition: Decision Boundaries



Sigmoid Unit


x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

$\sigma(x)$ is the sigmoid function

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

Sigmoid Unit

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

We can derive gradient descent rules to train

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

Error Gradient for a Sigmoid Unit

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

Error Gradient for a Sigmoid Unit

But we know:

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

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

So:

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

Backpropagation Algorithm

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

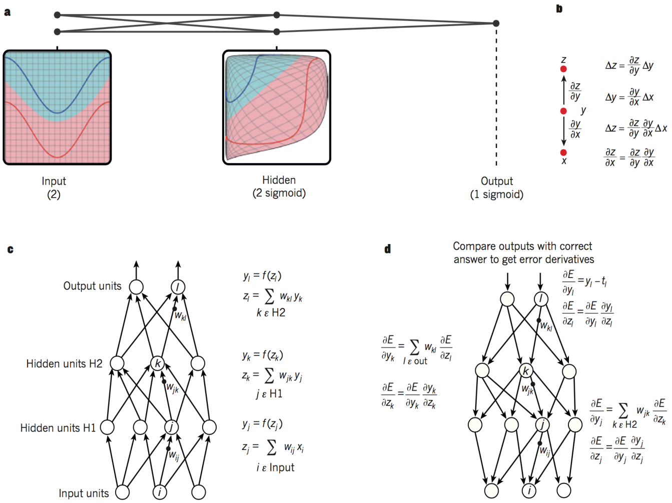
More on Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Add a term to error that increases with the magnitude of the weight vector
- 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 $\rightarrow$ slow!
- Using network after training is very fast

Deep Learning


a
Input (2)
Hidden (2 sigmoid)
Output (1 sigmoid)
b
$\Delta z = \frac{\partial z}{\partial y} \Delta y$
$\Delta y = \frac{\partial y}{\partial x} \Delta x$
$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$
$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$
c
Output units
Hidden units H2
Hidden units H1
Input units
$y_l = f(z_l)$
$z_l = \sum_{k \in H2} w_{kl} y_k$
$y_k = f(z_k)$
$z_k = \sum_{j \in H1} w_{jk} y_j$
$y_j = f(z_j)$
$z_j = \sum_{i \in \text{Input}} w_{ij} x_i$
d
Compare outputs with correct answer to get error derivatives
$\frac{\partial E}{\partial y_l} = y_l - t_l$
$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$
$\frac{\partial E}{\partial y_k} = \sum_{l \in \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$
$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$
$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk} \frac{\partial E}{\partial z_k}$
$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$

Y. Lecun et al. (2015) Nature (521) 436–444.

Convergence of Backpropagation

Gradient descent to some local minimum

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

Nature of convergence

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

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].

How complex should the model be ?

With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.

John von Neumann

Overfitting in ANNs

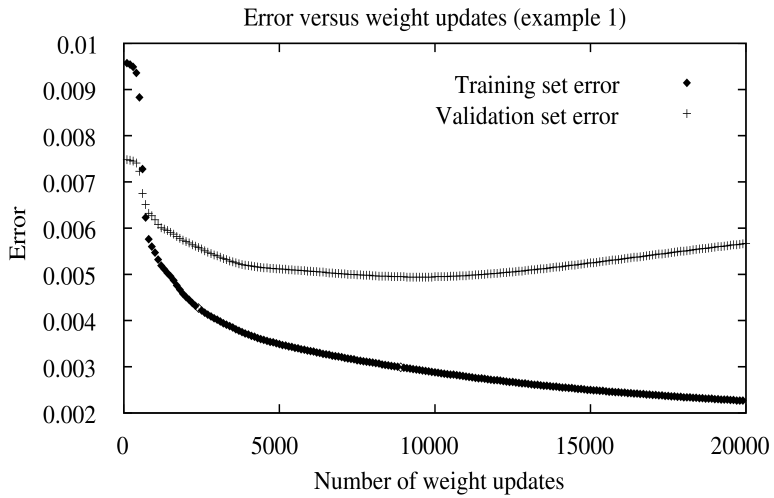
Can neural networks overfit ?

Next two slides: plots of “learning curves” for error as the network learns (shown by number of weight updates) on two different robot perception tasks.

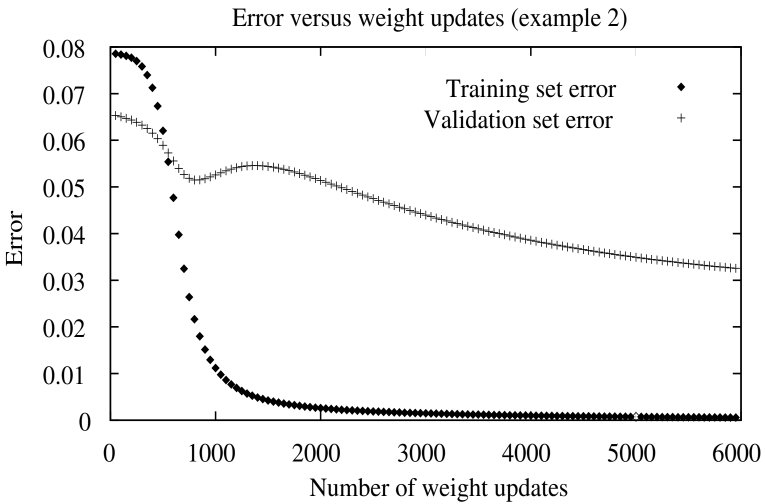
Note difference between training set and off-training set (validation set) error on both tasks !

Note also that on second task validation set error continues to decrease after an initial increase — any regularisation (network simplification, or weight reduction) strategies need to avoid early stopping

Overfitting in ANNs



Overfitting in ANNs



A practical application: Face Recognition

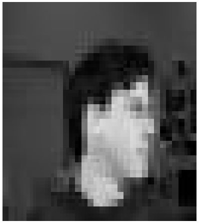
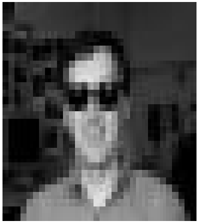
Dataset: 624 images of faces of 20 different people.

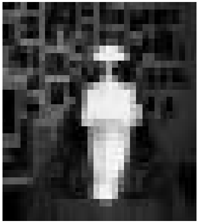
- image size 120x128 pixels
- grey-scale, 0-255 pixel value range
- different poses
- different expressions
- wearing sunglasses or not

Raw images compressed to 30x32 pixels, each is mean of 4x4 pixels.

MLP structure: 960 inputs $\times$ 3 hidden nodes $\times$ 4 output nodes.

Neural Nets for Face Recognition - Task

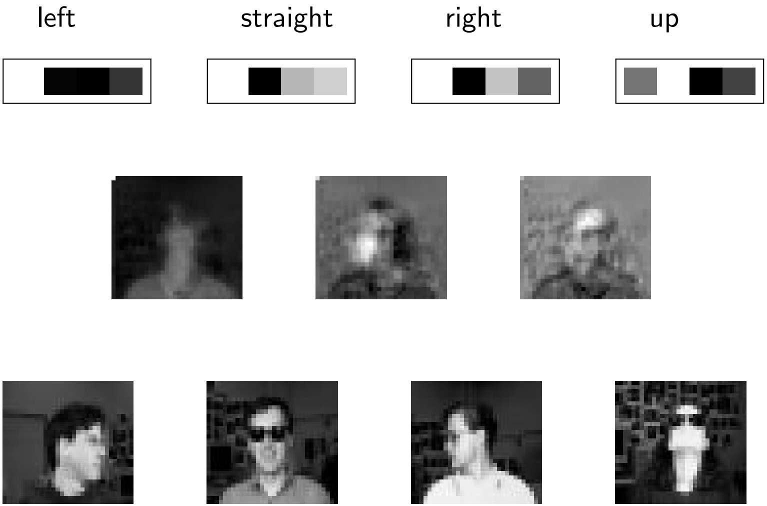





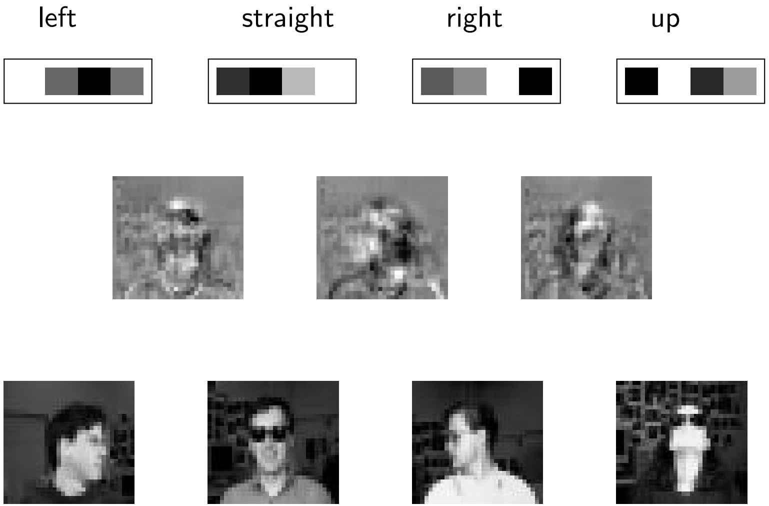
left straight right up

Four pose classes: looking left, straight ahead, right or upwards.
Use a 1-of-n encoding: more parameters; can give confidence of prediction.
Selected single hidden layer with 3 nodes by experimentation.

Neural Nets for Face Recognition - after 1 epoch



Neural Nets for Face Recognition - after 100 epochs



Neural Nets for Face Recognition - Results

Each output unit (left, straight, right, up) has four weights, shown by dark (negative) and light (positive) blocks.
Leftmost block corresponds to the bias (threshold) weight
Weights from each of 30x32 image pixels into each hidden unit are plotted in position of corresponding image pixel.
Classification accuracy: 90% on test set (default: 25%)
Question: what has the network learned ?
For code, data, etc. see `http://www.cs.cmu.edu/~tom/faces.html`

Alternative Error Functions

Penalize large weights:

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

Train on target slopes as well as values:

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

Tie together weights, e.g., in phoneme recognition network

And many more . . .

Evaluating numeric prediction

- Same strategies: independent test set, crossvalidation, significance tests, etc.
- Difference: error measures
- Actual target values: $a_1, a_2, \ldots, a_n$
- Predicted target values: $p_1, p_2, \ldots, p_n$
- Most popular measure: mean-squared error

$$\frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{n}$$

Easy to manipulate mathematically.

Other measures

The root mean-squared error:

$$\sqrt{\frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{n}}$$

The average (mean) absolute error is less sensitive to outliers than the mean-squared error:

$$\frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{n}$$

Sometimes relative error values are more appropriate (e.g. 10% for an error of 50 when predicting 500)

Summary

- Two frameworks for classification by linear models

  Distance-based. The key ideas are geometric. prediction error.
  Probabilistic. The key ideas are Bayesian.