

Intro. to machine learning (CSI 5325)

Lecture 9: neural networks

Greg Hamerly

Spring 2008

Some content from Tom Mitchell.

1 Example: Face Recognition

2 Advanced topics

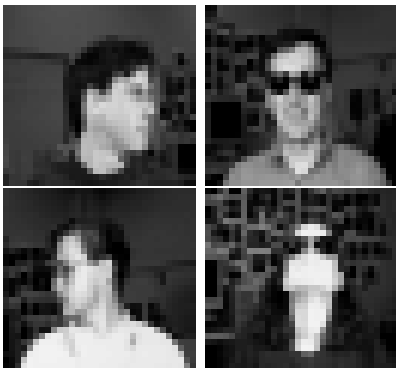
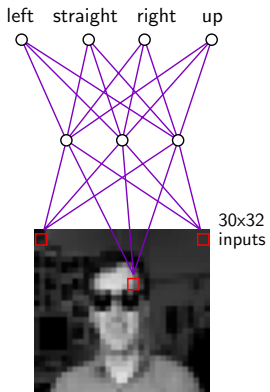
- Alternative error functions
- Recurrent Networks

3 Tricks of the trade

- Data scaling
- When to stop training?
- Choosing network topology
- Training faster

4 Next homework

Neural Nets for Face Recognition

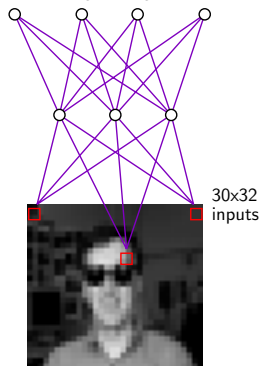


Typical input images

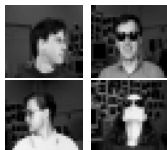
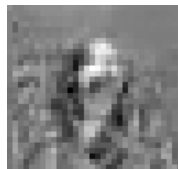
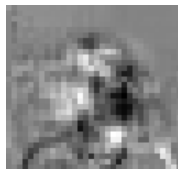
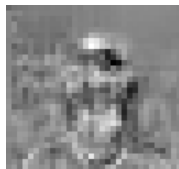
90% accurate learning head pose, and recognizing 1-of-20 faces

Learned Hidden Unit Weights

left straight right up



Learned Weights



Typical input images

Alternative error functions

Penalize large weights:

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

Gives an update rule for backpropagation that is identical, but each weight is multiplied by

$$(1 - 2\gamma\eta)$$

Alternative error functions

Maximizing the 'cross-entropy':

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

Cross-entropy assumes that the outputs are probabilities. Change \vec{w} to maximize E , the probability (likelihood) of the training set.

- not really an 'error' function... but the negative is!

All outputs and training labels must be in the range of $[0, 1]$.

Later we'll look at general 'maximum likelihood' methods.

Alternative error functions

Tie together weights

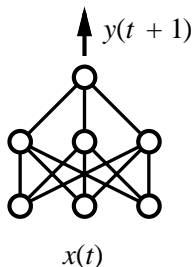
Based on prior knowledge, force certain weights to have the same values.

For example, phoneme recognition network:

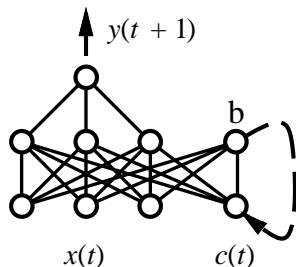
- certain weights representing the same function but at different times (e.g. different inputs)
- belief that they should behave the same way

Calculate weight updates independently, but then average the tied weights.

Recurrent Networks

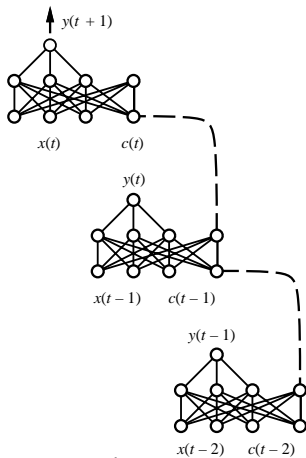


(a) Feedforward network



(b) Recurrent network

Recurrent Networks



(c) Recurrent network unfolded in time

Data scaling

Raw data can have values that slow backpropagation training.

For example, if all input values are between 1^9 and 2^9 , then...

- even small weights will produce large inputs to the sigmoid functions...
- therefore, most hidden-layer sigmoids will produce 0 or 1...
- therefore, the derivative of these sigmoids will be very small...
- therefore, training will be very slow.

If one input attribute has much larger values than another, the first will have a larger initial influence on the network.

Data scaling

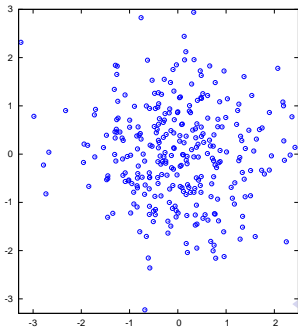
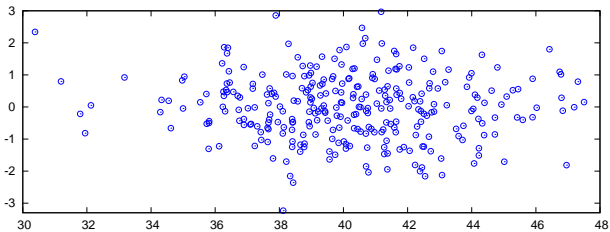
To avoid the problems of large values, a common trick is to recenter and z-score the data.

- Recenter: move the average value to be zero for each input attribute by subtracting the mean (over all the training examples).
- Z-score: divide each input value by the standard deviation of that attribute.

In Matlab:

$$x = (x - \text{mean}(x)) ./ \text{std}(x)$$

These two steps guarantee that input values will be within a reasonable range, and equalize the influence of each attribute.



How much training is enough?

We usually discuss training quantity in epochs.

One epoch is one loop over the training set.

Too much training leads to overfitting; not enough leads to underfitting.

- we can watch the validation set error

Single validation sets

Sometimes a single validation set is too limiting...

- not enough data
- aren't able to learn from the validation set
- can overfit the validation set! (how?)

Cross-validation is a preferred, more costly method.

Cross-validation

k -fold cross-validation partitions the training set X into k parts; call them X_1, X_2, \dots, X_k .

Partition: each part is disjoint, and together they form the original training set.

Do the following for $i = 1 \dots k$:

- train a (new) network on examples from $X \setminus X_i$ (all examples except X_i)
- evaluate the network on validation set from X_i

Compute the average error over all k folds.

Cross-validation

Doesn't give a single network, but does give more robust estimates of overfitting.

Can tell us how many training epochs to use.

When $k = n$, this is called 'leave-one-out cross-validation.'

Often use $k = 10$.

Validation sets and cross-validation are used throughout machine learning (next chapter!).

Choosing a network topology

How do we know what network topology we want?

- prior knowledge
- try lots of them + validation
- start small and grow (cascade-correlation algorithm)
- start large and shrink (optimal brain damage [LeCun], weight decay)
- other ideas?

Weight decay

A simple method of learning topology by

- training with backpropagation
- multiply each weight by a factor in $(0, 1)$ after each update
 - 'decaying' the weights so the unimportant ones get close to zero
- pruning connections with low weights after training
 - by removing them or setting their weights to zero

Equivalent to the alternative error function which penalizes large weights:

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

Other topologies

- Convolution networks
- Linear outputs for regression

Converging faster: line search

Once we compute ∇E , we have a vector $\Delta \vec{w} \propto -\nabla E$ in the weight space.

Move along that vector (line) until we find a minimum, then repeat.

Converging faster: second-order methods

Use the second order derivative, the Hessian.

The Hessian is the matrix of derivatives of the gradient.

$$H(E, w) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1 \partial w_1} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_n \partial w_1} & \frac{\partial^2 E}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_n} \end{bmatrix}$$

- second-order methods may train faster than first-order
- allows networks to re-train quickly
- inverse of the Hessian gives info on weight significance
- other useful properties

Still a cost of computing H .

Next homework

Implement backpropagation for 3-layer networks.

Learn AND, XOR, $8 \times 3 \times 8$ autoencoding, and shapes.

Focus on thorough experimentation and writeups.