Lecture 9: Neural Networks III

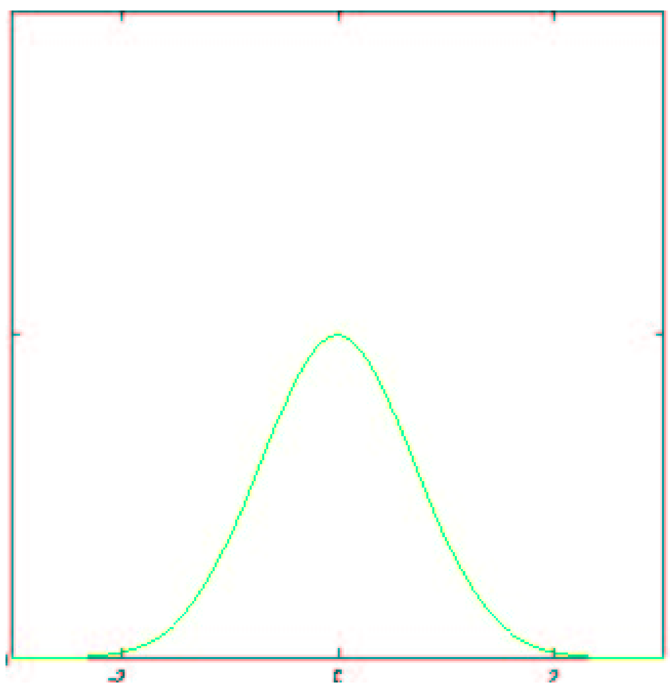
Radial Basis Functions

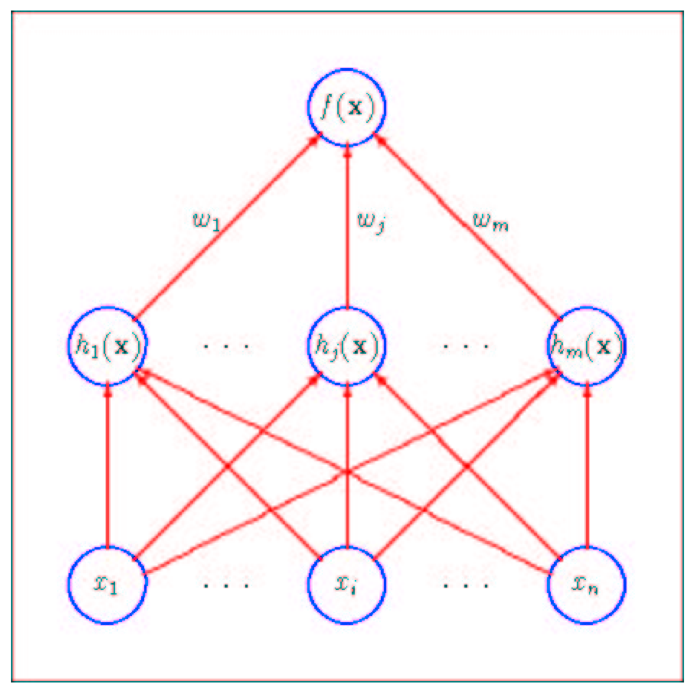
Using other functions besides sigmoids as bases, hidden activations (splines, wavelets, polynomials)

- instead of sigmoidal hidden units, we can use Gaussian shaped bumps (RBFs) as our basis functions:

$$h_j(\mathbf{x}) = g(\mathbf{x}, \mathbf{w}_j) = \exp[-\frac{1}{2\sigma^2}||\mathbf{x} - \mathbf{w}_j||^2]$$

- weights into hidden unit j describe the mean of that bump





Training Radial Basis Function Network

- learn the weights/positions using backprop
  - forward stage: different activation function $g()$ for hidden units
  - backprop stage: different derivatives of the activation function $g()$

$$\delta_j = g'(u_j) \sum_k w_{kj} \delta_k$$

  - need to set σ – learn or make a constant
  - also need to initialize weights - typically center on training examples

BackProp as ML's Holy Grail

BP: avoids process of articulating heuristics or rules for machines performing nontrivial tasks

Instead just learn by example – discovers appropriate heuristics and rules to the task at hand

Can also trivialize – just chain rule, albeit efficient implementation - $O(|W|)$ as opposed to $O(|W|^2)$

Now that have overcome perceptron limitations, solved hidden unit learning issue $\Rightarrow$ solve very hard problems

Kolmogorov's Theorem: any real-valued function of real inputs can be approximated in single-hidden-layer net with sufficient hidden units

But not so easy – same generalization issues apply

How to Maximize Generalization?

Need to match hypothesis complexity to amount of training data available

Goal: control complexity of network mapping

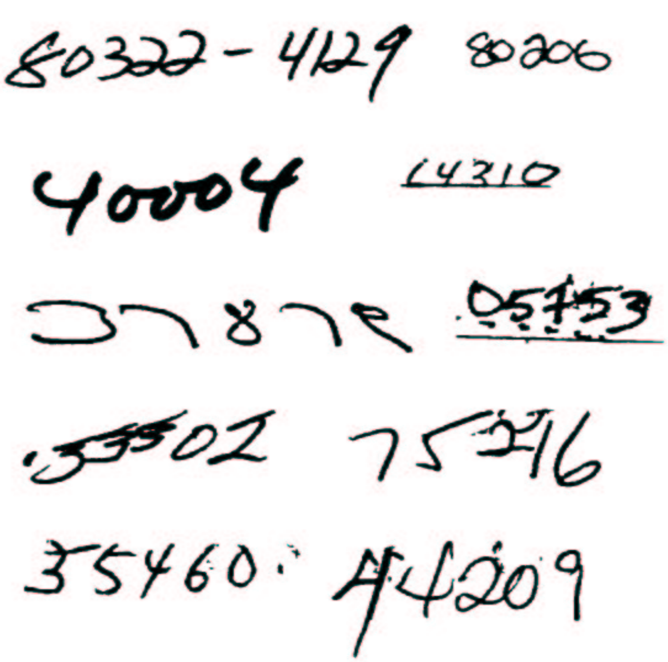
- tailor network – reduce free parameters
  1. select appropriate representation
  2. weight sharing
  3. weight pruning
- use validation set to avoid over-training
- more data (training examples)
  1. fabricate training data
  2. add noise to data

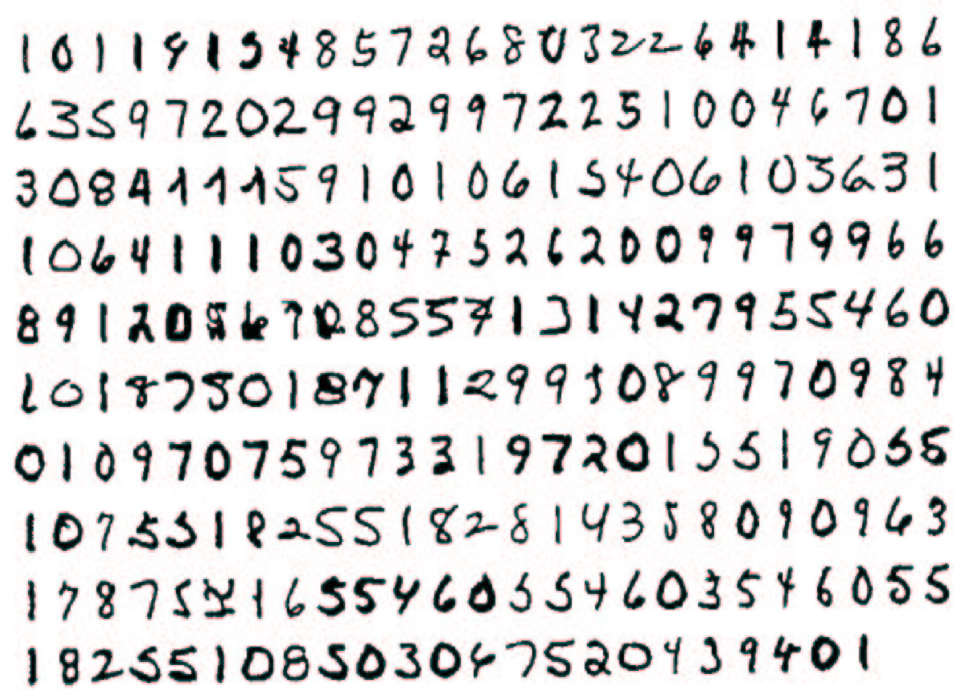
General point: building structure into solution can eliminate variance without eliminating the solution from possible net configurations

Weight Sharing – Example: LeNet

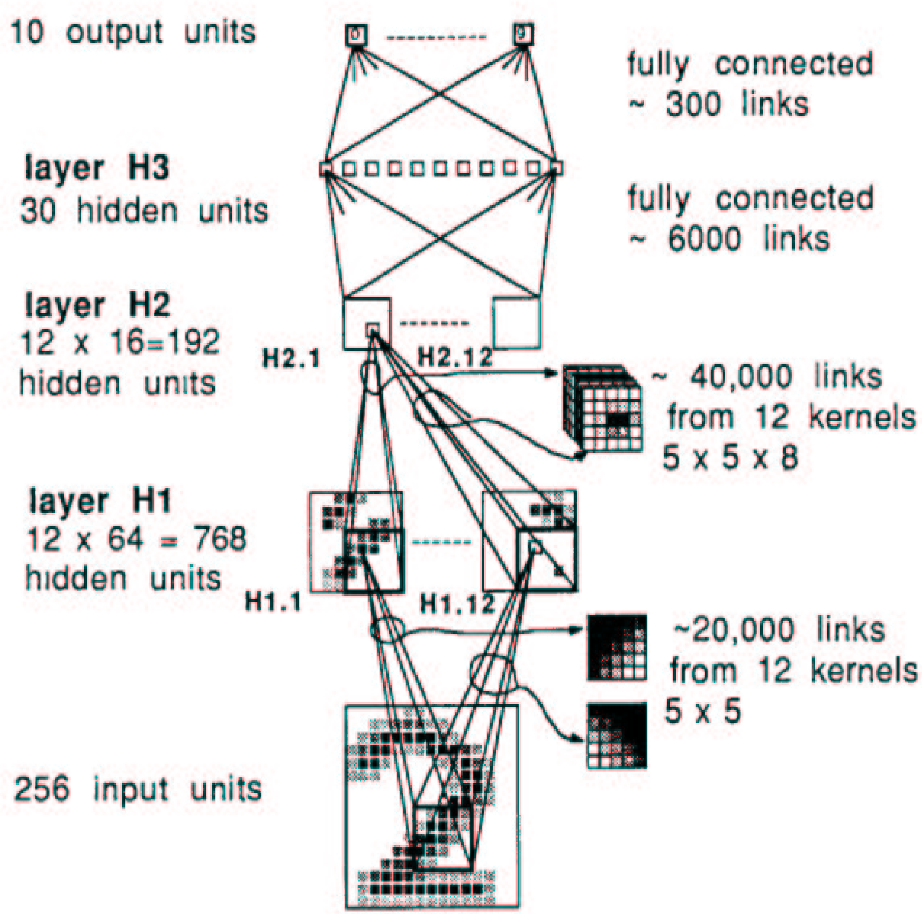
Hand-written digit recognition network: http://yann.lecun.com/exdb/lenet/index.html

- 7291 training examples + 2007 test examples
- Both contain ambiguous & misclassified examples
- Input pre-processed (segmented, normalized)
  – 16x16 gray level [-1,1],10 outputs





LeNet: Summary



Main ideas:

– local $\rightarrow$ global processing

– retain coarse posit. information

Main technique: *weight sharing*

– units arranged in featuremaps

Connections:

– 1256 units, 64,660 cxns, 9760 free parameters

Results:

– 0.14% (training) + 5.0% (test)

– 3-layer net (40 hidden units): 1.6% (training) + 8.1% (test)

Pruning Network Weights I – Weight Decay

Take net that works well – cut out half the weights and end up with net that works as well or better

Many methods for determining number of weights by introducing extra *regularizing* terms into objective function

Example: “weight decay”

$$E = \sum_{c,k} (y_k^c - t_k^c)^2 + \lambda \sum_{ij} w_{ij}^2$$

Here one component of $\frac{\partial E}{\partial w_{ij}}$ is $2\lambda w_{ij}$, so weight change is proportional to $-2\lambda w_{ij}$

Encourages small weights, or weight elimination

Pruning Network Weights II – Optimal Brain Damage

Weight saliency: analytical prediction of effectiveness of particular parameter wrt objective function

Use Taylor series approximation to predict effect of perturbing some parameter (under some approximations):

$$\delta E = 1/2 \sum_i \frac{\partial^2 E}{\partial w_{ij}^2} \delta w_{ij}$$

Algorithm (loop):

1. Train network to local minimum
2. Use back-prop like procedure to compute diagonal second derivatives
3. Delete some parameters with low saliency (little effect of perturbing it on E)

Validation set methods

estimate generalization error by examining performance on independent *validation* set

1. early stopping: stop training when validation error starts to increase (keeps weight small)

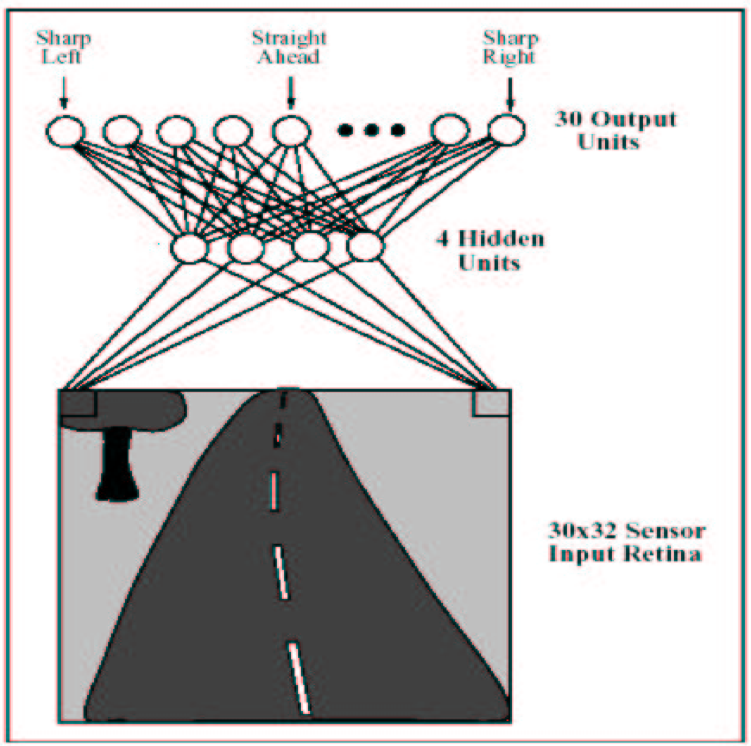
2. leave-out validation: divide training set into S segments, leave out 1 of those segments each training iteration, different segment each time: $T = [T_1, T_2, \cdots, T_S]$

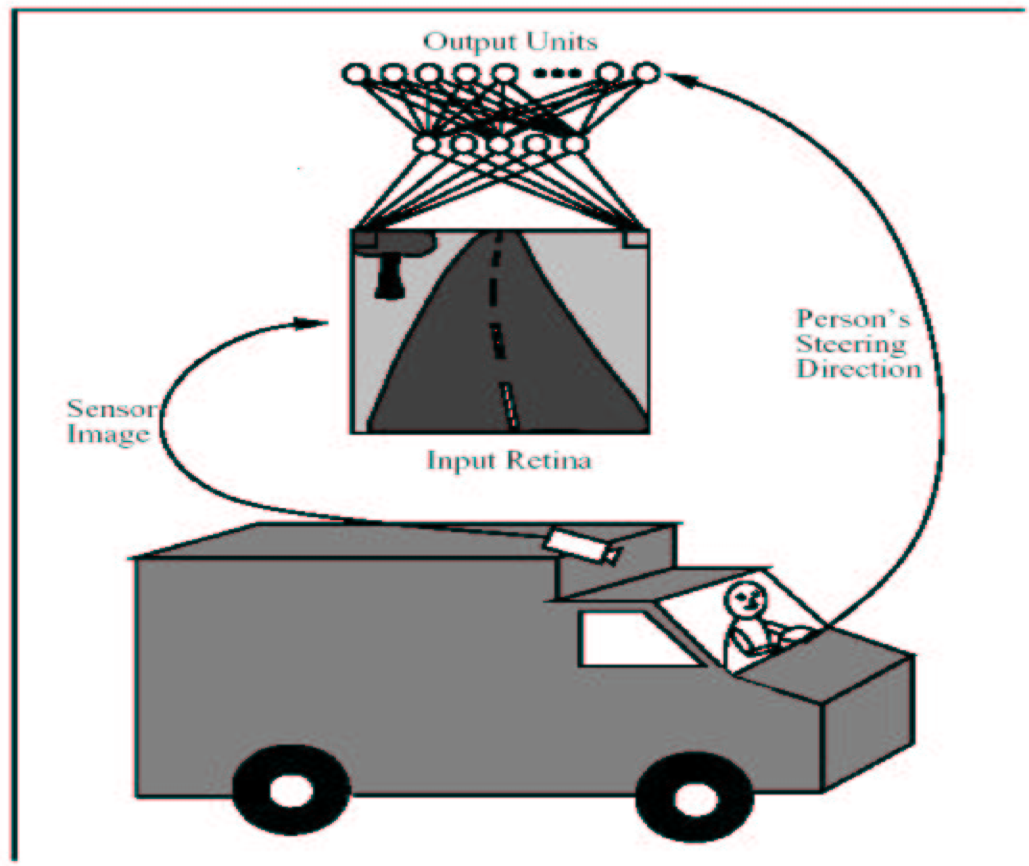
Fabricating Training Data

–Another way of overcoming lack of enough training data is to make it up.

–If good data can be constructed, we can approach optimal solutions.

Example: ALVINN - http://www.ri.cmu.edu/projects/project_160.html



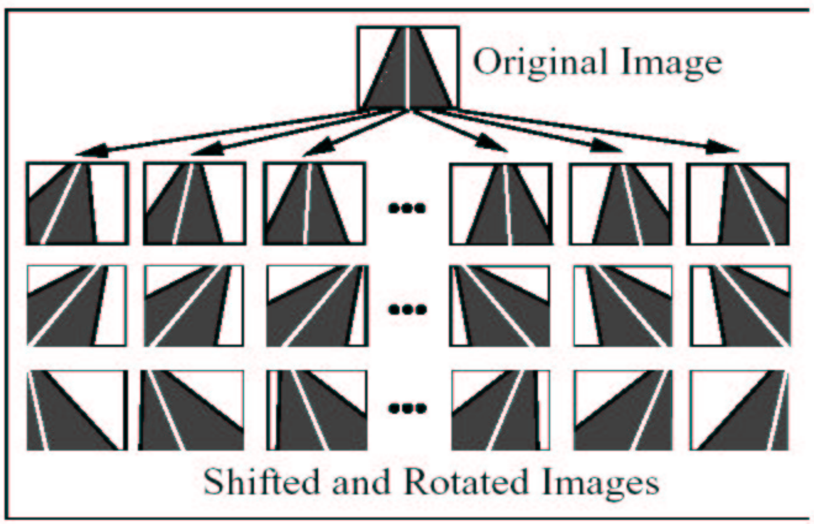


ALVINN: Simulating Training Examples

On-the-fly training: current video camera image as input, current steering direction as target

But:
– no experience with going off road
– over-training on same inputs

Method – generates 14 new training examples by shifting images



Replaces 10 low-error & 5 random training examples with 15 new ones

Key: relation between input and output known!

NO TUTORIAL THIS FRIDAY!