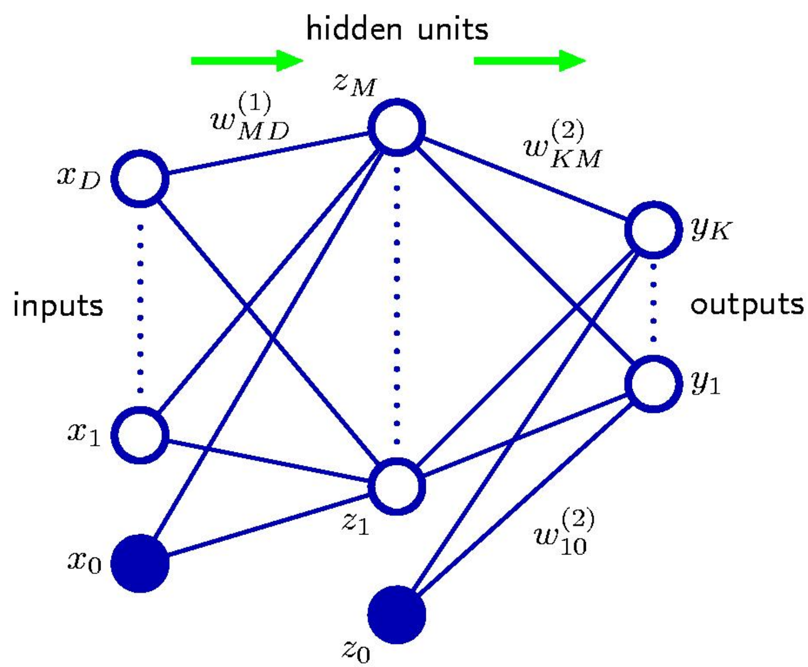
Artificial Neural Networks: Introduction

Sargur Srihari

Topics

1. Introduction
    1. As an extension of linear models
    2. Feed-forward Network Functions
    3. Weight-space symmetries
2. Autonomous Vehicle Navigation Example
3. When to use ANNs

A Neural Network



Can be viewed as a generalization of linear models

$$y_k(\boldsymbol{x},\boldsymbol{w}) = \boldsymbol{\sigma}\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

Linear Models

- Linear Models for Regression and Classification have form

$$y(\boldsymbol{x},\boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right)$$

where $\boldsymbol{x}$ is a D-dimensional vector

$\phi_j(\boldsymbol{x})$ are fixed nonlinear *basis* functions

e.g., Gaussian, sigmoid or powers of $\boldsymbol{x}$

For Regression f is identity function

Linear Regression

- For Classification f is a nonlinear *activation* function

Generalized Linear Regression

  - If f is sigmoid it is called *logistic regression*

$$f(a) = \frac{1}{1+e^{-a}}$$

Extending Linear Models

$$y(\boldsymbol{x},\boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right)$$

- Linear models have limited applicability
  - Due to curse of dimensionality
    - E.g., no of polynomial coeffts needed, finding means of Gaussians
- Extend scope by adapting basis functions ϕ_j to data
  - Become useful in large scale problems
- Both SVMs and Neural Networks address this limitation
  - SVM
    - Varying number of basis functions M
      - centered on training data points
    - Select subset of these during training
  - Neural Network
    - Number of basis functions M fixed in advance
      - But the ϕ_j have their own parameters $\{w_{ji}\}$
    - Adapt all parameter values during training

$$y_k(\boldsymbol{x},\boldsymbol{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

SVM versus Neural Networks

- SVM
  - Involves non-linear optimization
  - Objective function is convex
    - Leads to straightforward optimization, *e.g.,* single minimum
  - Number of basis functions is much smaller than number of training points
    - Often large and increases with size of training set
  - RVM results in sparser models
    - Produces probabilistic outputs at expense of non-convex optimization
- Neural Network
  - Fixed number of basis functions, parametric forms
    - Multilayer perceptron uses layers of logistic regression models
    - Also involves non-convex optimization during training (many minima)
    - At expense of training, get a more compact and faster model

Origin of Neural Networks

- To find information processing models of biological systems
- Term covers wide range of models
  - Exaggerated claims of biological plausibility
- Biological realism imposes unnecessary constraints
- Neural networks are efficient models for machine learning
  - Particularly multilayer perceptrons
- Network parameters are obtained in maximum likelihood framework
  - A nonlinear optimization problem
  - Requires evaluatiing derivative of log-likelihood function wrt network parameters
    - Done efficiently using error back propagation

Feed Forward Network Functions

A neural network can also be represented similar to linear models but basis functions are generalized

$$y(\boldsymbol{x},\boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right)$$

activation function

For regression: identity function

For classification: a non-linear function e.g., sigmoid

Coefficients w_j adjusted during training

There can be several activation functions

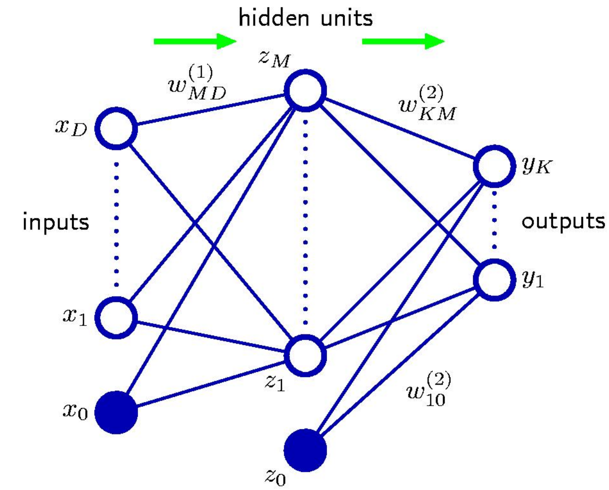
Basis functions

$\boldsymbol{\phi}_j(\boldsymbol{x})$ a nonlinear function of a linear combination of D inputs

its parameters are adjusted during training

First Layer: Basis Functions

- D input variables $x_1,.., x_D$
- M linear combinations in the form



$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \quad \text{where} \quad j = 1,..,M$$

  - Superscript (1) indicates parameters are in first layer of network
  - Parameters w_{ji} are referred to as weights
  - Parameters w_{j0} are biases, with $x_0=1$
  - Quantities a_j are known as *activations* (which are input to activation functions)
- No of hidden units M can be regarded as no. of basis functions
  - Where each basis function has parameters which can be adjusted

First Layer: Basis Functions

- Each activation a_j is transformed using differentiable nonlinear activation functions

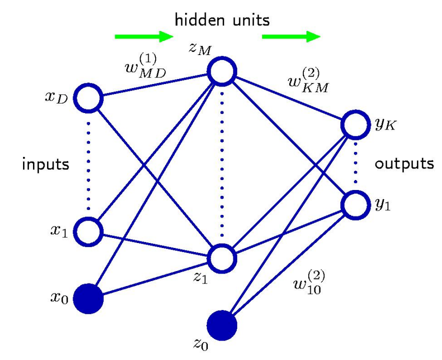
$$z_j = h(a_j)$$

- The z_j correspond to *outputs of basis functions* $\phi_j(\boldsymbol{x})$
  - *or first layer of network* or <u>*hidden units*</u>
- Nonlinear functions h
- Two examples of activation functions
  1. Logistic sigmoid

$$\boxed{\sigma(a) = \frac{1}{1+e^{-a}}}$$

  2. Hyperbolic tangent

$$\boxed{\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}}$$

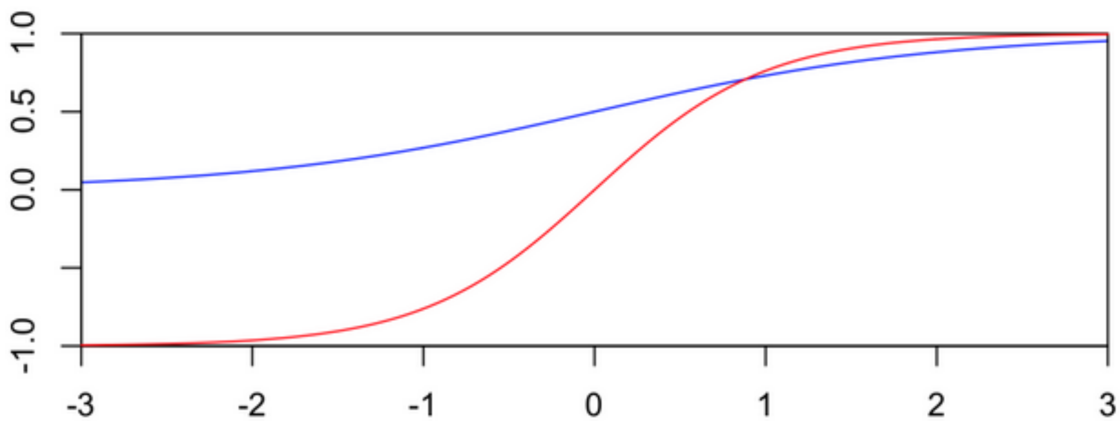


tanh is a rescaled sigmoid function

- The logistic sigmoid function is $\boxed{\sigma(a) = \frac{1}{1+e^{-a}}}$
  - The outputs range from 0 to 1 and are often interpreted as probabilities
- Tanh is a rescaling of logistic sigmoid, such that outputs range from -1 to 1. There is horizontal stretching as well.
  - It leads to the standard definition $\boxed{\tanh(a) = 2\sigma(2a) - 1}$

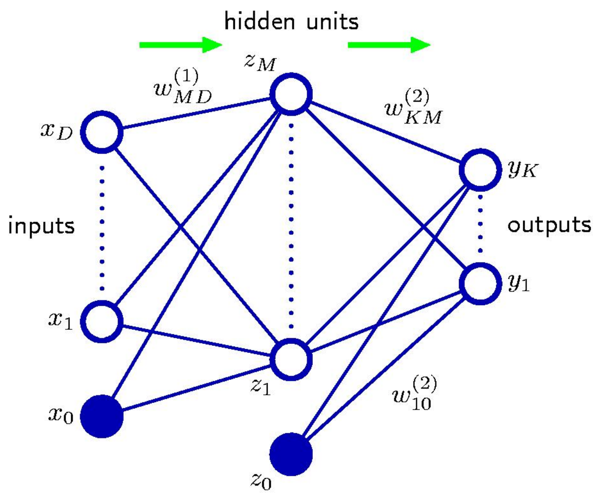
$$\boxed{\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}}$$

  - The $(-1,+1)$ output range is more convenient for neural networks.
  - The two functions are plotted below: blue is the logistic and red is tanh



Second Layer: Activations

- Values z_j are again linearly combined to give output unit activations



$$a_k = \sum_{i=1}^{M} w^{(2)}_{ki} x_i + w^{(2)}_{k0} \quad \text{where} \quad k = 1,..,K$$

- Where K is the total number of outputs
- Output unit activations are transformed by using appropriate activation function to give network outputs y_k

Choice of output activation

- Standard regression problems
  - identity function

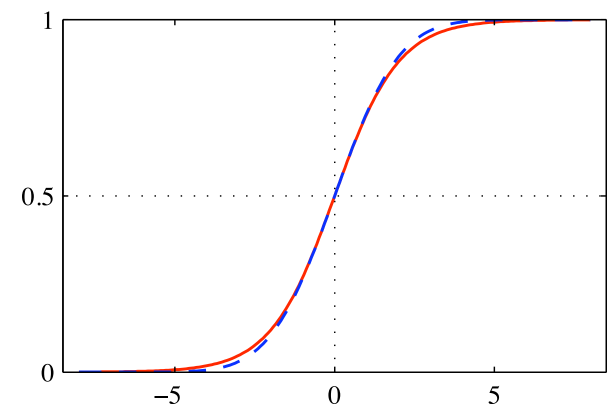
$$y_k = a_k$$

- For binary classification
  - logistic sigmoid function

$$y_k = \sigma(a_k)$$

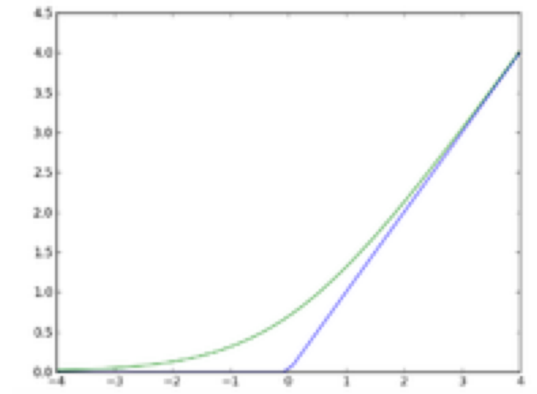
- For multiclass problems
  - a softmax activation function

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$



Note that logistic sigmoid also involves an exponential making it a special case of softmax

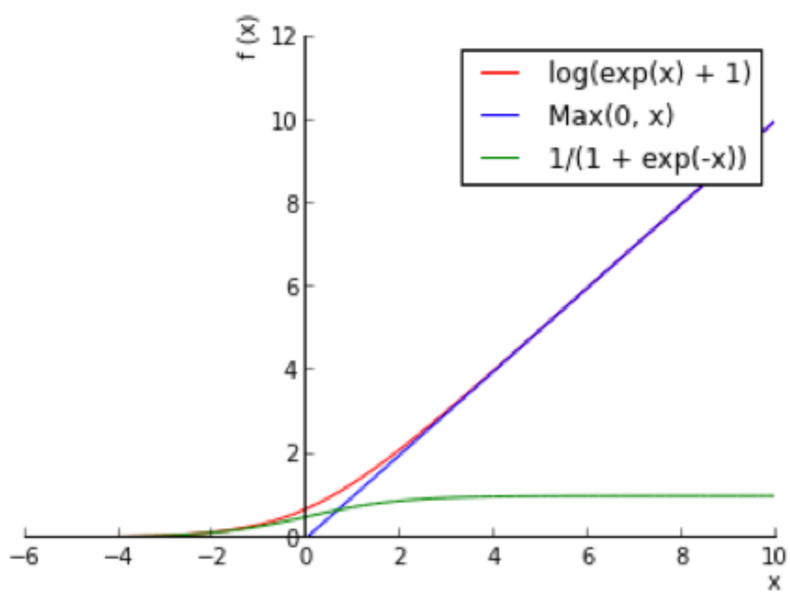
Activation function: Rectifier Linear Unit (ReLU)



- $f(x) = \max(0, x)$
- Where x is input to activation function
  - This is a ramp function
  - Argued to be more biological than widely used logistic sigmoid and its more practical counterpart hyperbolic tangent
  - Popular activation function in 2015 for *deep neural networks* since function does not quickly saturate (leading to vanishing gradients)
- Smooth approximation is $f(x) = \ln(1 + e^x)$ called the *softplus*
- Derivative of softplus is $f'(x) = e^x/(e^x + 1) = 1/(1 + e^{-x})$ i.e., logistic sigmoid
- Can be extended to include Gaussian noise

Comparison of ReLU and Sigmoid

- Sigmoid has gradient vanishing problem as we increase or decrease x
- Plots of softplus, hardmax (Max) and sigmoid:



Overall Network Function

- Combining stages of the overall function with sigmoidal output

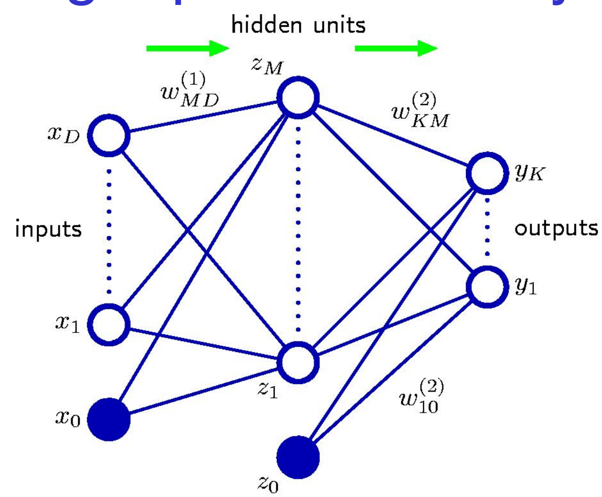
$$y_k(\boldsymbol{x},\boldsymbol{w}) = \boldsymbol{\sigma}\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

- Where $\boldsymbol{w}$ is the set of all weights and bias parameters
- Thus a neural network is simply
  - a set of nonlinear functions from input variables $\{x_i\}$ to output variables $\{y_k\}$
  - controlled by vector $\boldsymbol{w}$ of adjustable parameters
- Note presence of both σ and h functions

Forward Propagation

- Bias parameters can be absorbed into weight parameters by defining a new input variable x_0

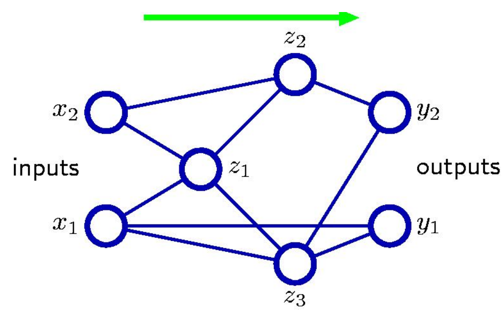
$$y_k(\boldsymbol{x},\boldsymbol{w}) = \boldsymbol{\sigma}\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right)\right)$$



- Process of evaluation is forward propagation through network
- Multilayer perceptron is a misnomer
  - since only continuous sigmoidal functions are used

Feed Forward Topology

- Network can be sparse with not all connections being present



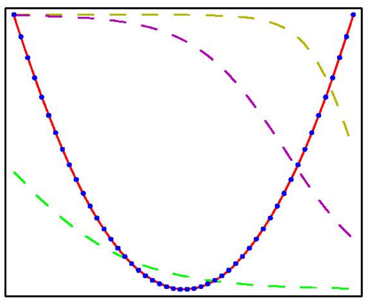
- More complex network diagrams
- But restricted to feed forward architecture
  - No closed cycles ensures outputs are deterministic functions of inputs
- Each hidden or output unit computes a function given by

$$z_k = h\left(\sum_j w_{kj} z_j\right)$$

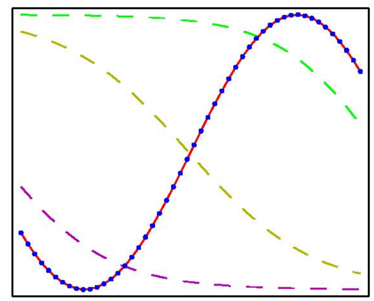
Examples of Neural Network Models of Regression

- Two-layer network with three hidden units
- Three hidden units collaborate to approximate the final function
- Dashed lines show outputs of hidden units
- 50 data points (blue dots) from each of four functions (over $[-1,1]$)

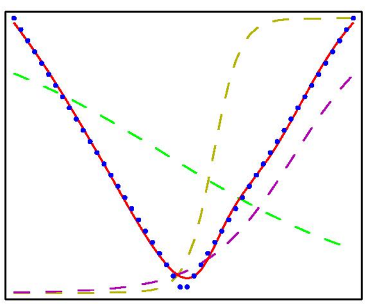
$$f(x) = x^2$$



$$f(x) = sin(x)$$

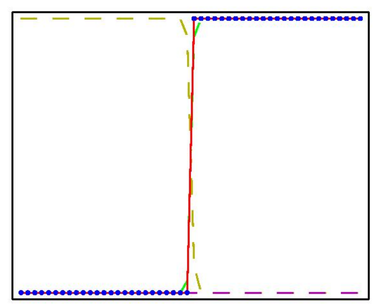


$$f(x) = |x|$$



$$f(x) = H(x)$$

Heaviside
Step Function

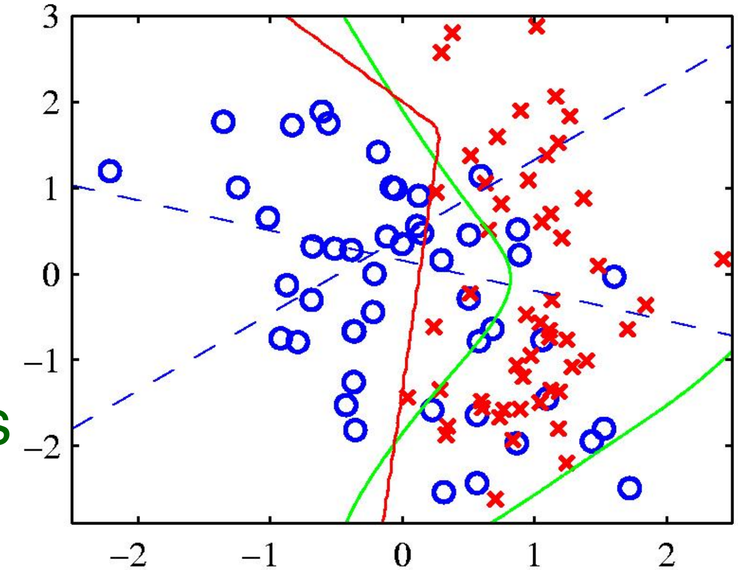


Example of Two-class Classification

Neural Network:

Two inputs, two hidden units with $\tanh$ activation functions

And single output with sigmoid activation



Red line:

decision boundary $y=0.5$ for network

Dashed blue lines:

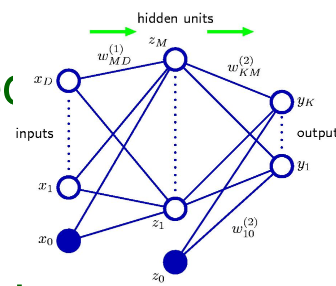
contours for $z=0.5$ of two hidden units

Green line:

optimal decision boundary from distributions used to generate the data

Weight-Space Symmetries

- There are multiple distinct choices for the weight vector $\boldsymbol{w}$
  - All give rise to the same mapping function from inputs to outputs
- Easily shown in fully connected network
  - M hidden units having $\tanh$ activation function change sign of all weights and bias feeding to particular hidden node
  - Since $\tanh\ (\text{-}a) = \text{-}\tanh(a)$ this change can be compensated changing sign of all weights leading out of that hidden unit



For M hidden units M such *sign-flip* symmetries

Thus any given weight vector will be one of 2^M equivalent weight vectors

Since the values of all weights and bias of a node can be interchanged for M hidden units there are $M!$ equivalent weight vectors

- Network has overall weight space symmetry factor of $M!2^M$
- No practical consequence other than in Bayesian neural networks

Applications of ANNs

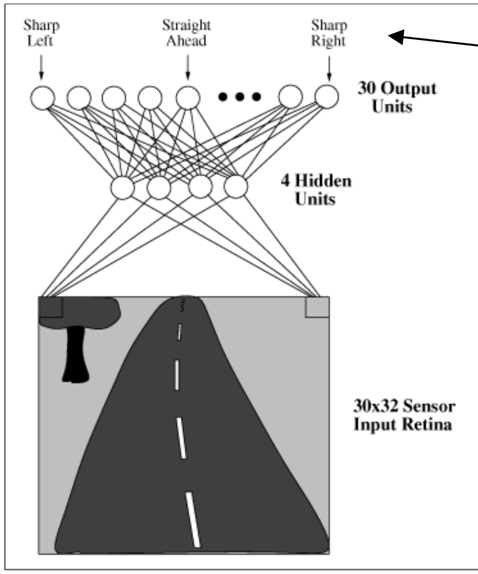
- Most effective learning method known for learning to interpret complex-real world data
- BACKPROPAGATION Algorithm
  - Learning to recognize handwritten characters
  - Learning to recognize spoken words
  - Learning to recognize faces

Neural Network Representations

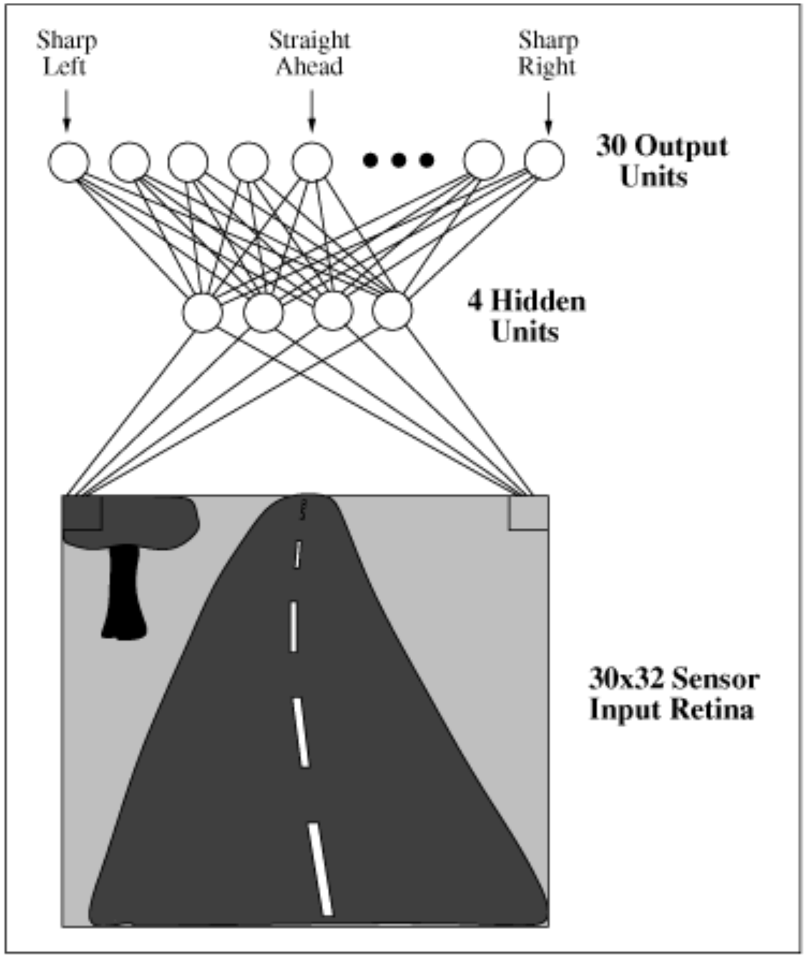
- Autonomous Vehicle is an example of a system that uses an ANN to steer an vehicle on a public high way

Output:
Direction in which vehicle is steered

Input:
30 x 32 grid of pixel intensities



ANN for Steering



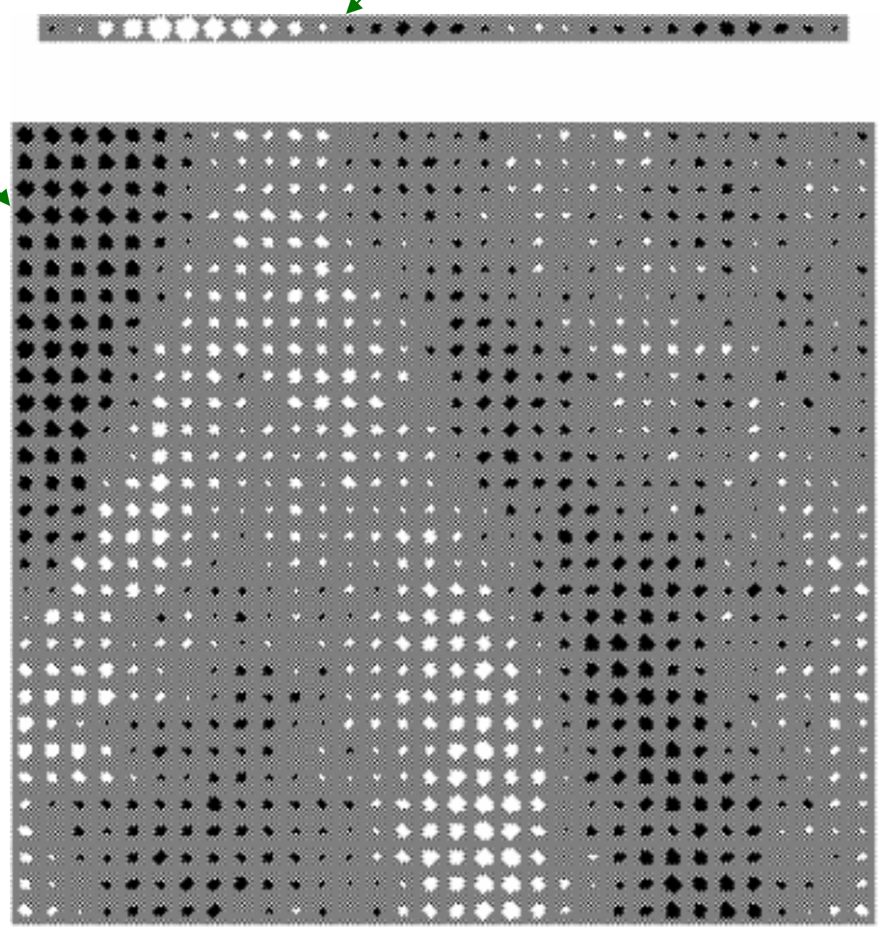
Each hidden unit has 960 inputs and produces 4 outputs connected to 30 output units

Each output unit corresponds to a steering direction

Learned Weights

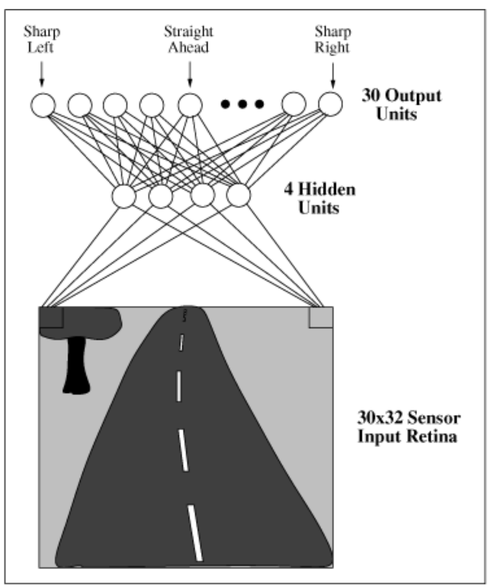
Weights of inputs to one of four hidden units,white is positive, black is negative;
size indicates wt magnitude

Weights from hidden unit to 30 output units. This hidden unit prefers left turn



4 x 32 per row

30 rows



Appropriate Problems for Neural Network Learning

- Well suited for complex sensor data such as from cameras and microphones
- Also applicable to problems where symbolic representations are used, as in decision tree learning
  - Decision trees and ANNs produce results of comparable accuracy
- Backpropagation is most commonly used for ANN learning

When to use ANNs

- Instances represented by attribute-value pairs
- Vector of predefined features
    - Pixel values (as in ALVINN)
    - Input attributes can be highly correlated or independent
    - Real values
- Target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes
    - ALVINN output is a vector of 30 attributes, each corresponding to a recommendation on steering direction
    - Value of each output is a real number between 0 and 1, corresponding to confidence in steering direction
    - Can train a single neural network to output both steering command and acceleration, by concatenating these two output predictions

When to use ANNs (2)

- The training examples may contain errors
  - robust to noise
- Long training times are acceptable
- Fast evaluation of the learning target function may be required
- The ability of humans to understand the learning target function is not important