ROB 537: Learning-Based Control

Week 1, Lecture 1
Neural Network Basics

Announcements:
Project groups selected: Today
HW 1 Due on 10/2
Data sets for HW 1 are online
Robotics Seminars: 10 AM Fridays
Join Robotics mailing list (announcements page)

Reading assignment:
Two neural network papers (posted)

Learning From Data

\[ y = f(x) = a \cdot x + b \]

Linear regression (parametric)
Learning From Data

- \( y = f(x) = a \cdot x + b \) ??
- \( y = f(x) = a \cdot x^2 + b \cdot x + c \)

Polynomial regression (parametric)

Learning From Data

- \( y = f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_2 \cdot x^2 + a_1 \cdot x + a_0 \)

Polynomial regression (parametric)
Learning From Data

- $y = f(x)$

Neural Networks for Nonlinear Control

- Motivation:
  - Control a system with nonlinear dynamics
    - Robot
    - Satellite
    - Air vehicle
  - Do we know what the good control strategies are?
  - Yes: “teach” neural network those strategies
    - Drive a car and record good driver actions for each state
    - Fly a helicopter and record good pilot actions for each state
  - No: have a neural network discover those strategies
    - Let car drive around and provide feedback on performance
Neural Networks

- Why Neural Networks?
- Units or Neurons
- Neural Network Architectures
- Activation Functions
- Single Layer Feed Forward Networks
- Multi Layer Feed Forward Networks
- Error Backpropagation
- Implementation Issues

Why Neural Networks?

- Neural Network: A massively parallel distributed processor made up of simple processing units. It stores "knowledge".

- An artificial neural network is similar to the brain in that:
  - Knowledge is acquired by the network from its environment through a learning process
  - Interneuron connection strengths (synaptic weights) are used to store the acquired knowledge

- An artificial neural network is different from the brain in a thousand ways
  - ...

- Think of a neural network as a statistical tool.
Benefits of Neural Networks

- Performs an input/output mapping
- Can be trained from examples
- Is adaptive to changing environments
- Provides probabilistic response
- Yields fault tolerant computing

Nonlinear regression ++
Functional form of mapping need not be known
Track nonstationarity
Confidence in solution
Graceful degradation

Input / Output Mapping

- Supervised learning: learning with a set of labeled examples
- Each example has:
  - An input
  - A desired output
- Training:
  - Present input
  - Compute output
  - Compare network output to desired output
  - Update network weights to minimize error
- When weights are “stable” network has learned an input/output mapping
Classification: Dog vs. Cat

- Dog vs. Cat: Cat
Classification:

- Dog vs. Cat: Cat
- Movement vs. stationary: “Dog” maybe
- Indoor vs. outdoor: “Dog”
- Red vs. not red animal: “Dog”
Types of Learning

- Learning Rules:
  - Hebbian
  - Memory Based
  - Competitive
  - Gradient descent

- Learning Paradigms:
  - Supervised
  - Critic (Reinforcement Learning)
  - Unsupervised

Hebbian Learning

- If two neurons are activated at the same time, strengthen the weight between them (Hebb, 1949)

- Properties:
  - Highly Local
  - Time dependent
  - Interactive

- Appeal: Evidence for biological plausibility
Memory Based Learning

- Explicitly store experiences (patterns) in memory
- When a new pattern is observed:
  - Find stored patterns in neighborhood of test pattern
- Example: Nearest Neighbor algorithm
  - For each new unseen pattern, find closest (or closest K) patterns in memory
  - Assign new pattern to class most frequently represented in the neighborhood
- Slow recall (search through all stored patterns)

Competitive Learning

- Only neurons winning some “competition” are updated
- Basic elements:
  - All neurons start the same
  - There is a limit on the total strength of each neuron
  - A mechanism for neurons to compete. Winner is called winner-takes-all neuron
- Example: neurons represent concentrations of data
  - For each pattern, the winning neuron is modified to be closer to that particular pattern
  - Neurons form clumps to represent the different data clusters
Gradient Descent

- Update weights to minimize error
- Take steps proportional to the negative of the derivative
- More later ...

Model of a Neuron

- Each input is a product of some signal (output?) and a weight
- All incoming inputs are summed
- Sum goes through an “activation” function
- Output is sent out to the network
Activation Functions

**Sigmoid Functions**  These are smooth (differentiable) and monotonically increasing.

The logistic function

\[ \text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \]

Hyperbolic tangent

\[ \tanh(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \]

**Piecewise-Linear Functions**  Approximations of a sigmoid function.

\[ f(x) = \begin{cases} 
1 & \text{if } x \geq 0.5 \\
 x + 0.5 & \text{if } -0.5 \leq x \leq 0.5 \\
 0 & \text{if } x \leq 0.5 
\end{cases} \]

Neural Network Architectures

**Single Layer Feed-forward**

**Multi-Layer Feed-forward**

**Recurrent Network**
Neural Network Architectures

Single Layer Feed-forward

Multi-Layer Feed-forward

Recurrent Network

Single Layer Feed Forward Networks

- Input $x$ is an $m$ element input vector
- Target $t$ is the desired output (can be a vector)
- Output $y$ is response to $x$
- Error $e$ is difference between desired and network outputs

$$e = t - y$$
Single Layer Feed Forward Networks

- Linear Discrimination:
  \[ y = \sum_{k} w_k x_k + w_0 \]

- Logistic Discrimination:
  \[ y = f\left( \sum_{k} w_k x_k + w_0 \right) \]

Learning From Data: Representation

- \( y = f(x) = ax + b \)
Learning From Data: Representation

• \( y = f(x) = ax^2 + bx + c \)

• \( h(x) = \frac{1}{1+e^{-x}} \)

Learning From Data: Representation

• \( y = f(x) = ?? \)

• \( h(x) = \frac{1}{1+e^{-x}} \)
Single Layer Feed Forward Networks: Training

- n patterns \((x^n, t^n)\)

- Mean Square Error:
  \[
  E = \frac{1}{2} \sum_{n=1}^{N} (t^n - y^n)^2
  \]

- Error on pattern n:
  \[
  e^n = t^n - y^n
  \]

- Mean Square Error:
  \[
  E = \frac{1}{2} \sum_{n=1}^{N} (e^n)^2
  \]

- Least Mean Square algorithm:
  \[
  \frac{\partial E}{\partial w} = e^n \frac{\partial e^n}{\partial w}
  \]

- Gradient descent:
  \[
  \Delta w = -\eta \frac{\partial E}{\partial w}
  \]
Gradient Descent: Move in direction of negative derivative

\[ E(w) \]

\[ \frac{d E(w)}{dw_1} \]

Decreasing \( E(w) \)

\[ \frac{d E(w)}{dw_1} > 0 \]

\[ w_1 \leq w_1 - \eta \frac{d E(w)}{dw_1} \]

i.e., the rule decreases \( w_1 \)

Gradient Descent: Move in direction of negative derivative

\[ E(w) \]

\[ \frac{d E(w)}{dw_1} \]

Decreasing \( E(w) \)

\[ \frac{d E(w)}{dw_1} < 0 \]

\[ w_1 \leq w_1 - \eta \frac{d E(w)}{dw_1} \]

i.e., the rule increases \( w_1 \)
Single Layer Feed Forward Networks

- Linear activation function:
  \[ \frac{\partial E}{\partial w_{i,j}} = e_i^n \frac{\partial e_i^n}{\partial w_{i,j}} = -e_i^n \frac{\partial y_i^n}{\partial w_{i,j}} = -e_i^n x_j^n \]

- Weight update:
  \[ \Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}} = \eta \ e_i \ x_j \]

- Sigmoid activation function:
  \[ f(a) = \frac{1}{1 + e^{-a}} \]

- Derivative of sigmoid:
  \[ f'(a) = f(a)(1 - f(a)) \]

- Gradient descent:
  \[ \frac{\partial E}{\partial w_{i,j}} = e_i^n \frac{\partial e_i^n}{\partial w_{i,j}} = -e_i^n \frac{\partial}{\partial w_{i,j}} f(\sum_j w_{i,j} x_j^n) \]

- Weight update:
  \[ \Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}} = \eta \ e_i y_i(1 - y_i) x_j \]