ROB 537: Learning-Based Control

Guest Lecture: Mitch Colby

Week 7, Lecture 2
Advanced Topics:
1) Fitness modeling
2) Learning from demonstration

Announcements:
Project Due on 20 November
No Class on 25 November

Up Until Now...

• You have known $G(z)$
• You used $G(z)$ to assign fitness in an evolutionary algorithm
• What if you don’t have $G(z)$? Or what if computing $G(z)$ is too slow to run an EA? How do we provide feedback?
Up Until Now...

• You have had a ton of data
• You could use the data to perform gradient descent or Q-learning to find a policy.
• What if you don’t have a lot of data? Can you learn a policy from only a few examples?

Outline

• Fitness modeling in evolutionary algorithms
  – What if we can’t directly compute fitness values?
  – What if fitness assignment operator is prohibitively slow?

• Learning from demonstration:
  – If a task is non-trivial to encode, can we just have a robot watch us complete the task in order to learn how to do it?
Outline

- Fitness modeling in evolutionary algorithms
  - What if we can’t directly compute fitness values?
  - What if fitness assignment operator is prohibitively slow?

- Learning from demonstration:
  - If a task is non-trivial to encode, can we just have a robot watch us complete the task in order to learn how to do it?

Topic 1: Fitness Modeling

- Review of Evolutionary Algorithms
- EA operator analysis
- Fitness Assignment Operators Analysis
- Fitness Modeling Approach

Fitness modeling lecture based on:

Evolutionary Algorithms Review

Initialize $n$ random solutions

Create mutated solutions
Evolutionary Algorithms Review

1. Initialize $n$ random solutions
2. Create mutated solutions
3. Evaluate fitness of each solution
4. Select $n$ solutions to survive
Evolutionary Algorithms Review

1. Initialize $n$ random solutions
2. Create mutated solutions
3. Evaluate fitness of each solution
4. Select $n$ solutions to survive
5. Generation += 1

Evolutionary Algorithm Operator Analysis

1. Initialize $n$ random solutions
2. Create mutated solutions
3. Evaluate fitness of each solution
4. Select $n$ solutions to survive
5. Generation += 1

Computationally Cheap
Evolutionary Algorithm Operator Analysis

- Initialize $n$ random solutions
- Create mutated solutions
- Evaluate fitness of each solution
- Select $n$ solutions to survive

Example

- Problem: develop power plant controller via neuroevolution

EA Operators:
- Initialize solutions: initialize neural networks using RNG
- Mutate solutions: mutate neural networks using RNG
- Assign fitness: test neural network controller via
  - High fidelity simulation (computationally expensive)
  - Direct power plant implementation (slow, but also dangerous)
- Select solutions: select networks based on fitness values
Motivation for Fitness Modeling

There are two potential reasons fitness assignment causes problems in evolutionary algorithms:

1. Evaluating fitness via high fidelity simulations is too computationally costly.
   - Many simulators (e.g. computational fluid dynamics, finite element analysis) may run slower than real-time
   - Almost never run much faster than real time

2. Evaluating fitness online is too dangerous (and is typically costly in terms of time).
   - Randomized controllers could cause catastrophic system failure
   - Online fitness evaluation runs in exactly real-time

In any evolutionary algorithm, the cost of assigning fitness to solutions typically comprises most of the total cost of running the algorithm
• Suppose we have a simulator that operates in real-time
• We are evaluating plant controllers over 1 day (24 hours) of operation

• Consider an EA with:
  – Population size of 50
  – 50 child solutions created every generation
  – 1000 generations

In the best case scenario (static fitness assignment), this EA would require 1,201,200 hours for fitness assignment alone!
• Suppose we have a simulator that operates in real-time
• We are evaluating plant controllers over 1 day (24 hours) of operation

• Consider an EA with:
  – Population size of 50
  – 50 child solutions created every generation
  – 1000 generations

In the best case scenario (static fitness assignment), this EA would require 1,201,200 hours for fitness assignment alone!

For dynamic fitness assignment, this EA would require 2,400,000 hours for fitness assignment alone!

What Should We Do?

• We have to decrease the amount of time needed to assign fitness to a solution.

• Model/approximate the fitness assignment operator
Fitness Modeling

- Fitness modeling: develop an approximation/model of the system which is computationally efficient
  - Allows for an EA to be computationally tractable

- Common fitness model types:
  - Low fidelity simulations
  - Polynomial approximation
  - Kriging (Gaussian process regression): interpolated values are modeled via a Gaussian process with prior covariances
  - Neural network approximations

Fitness Modeling in an EA

1. Initialize $n$ random solutions
2. Create mutated solutions
3. Evaluate fitness of each solution
4. Select $n$ solutions to survive
5. Generation += 1
Fitness Modeling in an EA

1. Initialize $n$ random solutions
2. Create mutated solutions
3. Evaluate fitness of each solution
4. Select $n$ solutions to survive
5. Generation += 1

Approximate fitness of each solution

Fitness Modeling in an EA

1. Initialize $n$ random solutions
2. Create mutated solutions
3. Approximate fitness of each solution
4. Select $n$ solutions to survive
5. Generation += 1
Fitness Modeling Case Study

Optimizing the Ballast Configuration of a Wave Energy Converter

Columbia Power Technologies
Manta WEC

Problem Statement

Given:
• Known WEC geometry
• Known climate of operation (Oregon Coast) given by wave occurrence table

Find:
• Ballast configuration of WEC which optimizes energy capture in dynamic wave environment
Solution Representation

• Solution is a vector of values corresponding to where different ballast chambers are placed

• Different ballast chamber configurations:  
  – mass of water in each chamber:  
    • Affects center of mass position of each component  
    • Affects inertia of each component  
  – (CG, I): affect energy capture

How to Evaluate a Ballast Configuration?

• High Fidelity Simulator: ANSYS AQWA  
  – Hydrodynamic simulator of offshore and marine structures  
  – Method to analyze structure  
    • Define dynamic wave climate (dominate wave frequencies, directions, and heights)  
    • Define structure geometry (CAD model)  
    • Define structure mass parameters (inertia, center of gravity positions)  
    • Define mooring setup (i.e. is structure anchored)  
    • Create mesh along structure  
    • Numerically solve PDEs to find motion of structure
How to Evaluate a Ballast Configuration?

- High Fidelity Simulator: ANSYS AQWA
  - Hydrodynamic simulator of offshore and marine structures
  - Method to analyze structure
    - Define dynamic wave climate (dominate wave frequencies, directions, and heights)
    - Define structure geometry (CAD model)
    - Define structure mass parameters (inertia, center of gravity positions)
    - Define mooring setup (i.e. is structure anchored)
    - Create mesh along structure
    - Numerically solve PDEs to find motion of structure

Key Problem: AQWA takes about 24 hours to analyze a solution

Approach: Approximate Power Output

- Train a neural network to approximate power output of a WEC given its ballast configuration.
- Use this neural network to create a computationally efficient time domain simulator.
- Use the time domain simulator to approximate fitness.
Data Collection

- We ran 250 AQWA simulations, varying:
  - CG position of each component (spar and two floats)
  - Inertia of each component ($I_{xx}$, $I_{yy}$, $I_{zz}$, $I_{xy}$, $I_{xz}$, $I_{yz}$)

- For each simulation, we found:
  - Power output for CG/Inertia configuration in varying wave climates

Approximation Performance

- The AQWA simulations were used to train a neural network which completed the mapping:
  - $NN(I, CG, \text{wave height, wave frequency}) = \text{Power output}$

- After backpropagation, performance figures were:

<table>
<thead>
<tr>
<th>Data points</th>
<th>Hidden Units</th>
<th>Max Error</th>
<th>Mean Error</th>
<th>Portion of data set with &lt;5% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>10</td>
<td>42%</td>
<td>24%</td>
<td>22%</td>
</tr>
<tr>
<td>250</td>
<td>15</td>
<td>37%</td>
<td>18%</td>
<td>36%</td>
</tr>
<tr>
<td>250</td>
<td>20</td>
<td>34%</td>
<td>15%</td>
<td>42%</td>
</tr>
</tbody>
</table>
Active Data Collection

- The regions of the search space with highest error were found, and 150 more AQWA simulations were run in these regions.

- This *active data collection* allowed for the dataset to be refined without running unneeded simulations (e.g. in areas of the search space that already have low error).

Revised Approximation Performance

<table>
<thead>
<tr>
<th>Data points</th>
<th>Hidden Units</th>
<th>Max Error</th>
<th>Mean Error</th>
<th>Portion of data set with &lt;5% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>10</td>
<td>42%</td>
<td>24%</td>
<td>22%</td>
</tr>
<tr>
<td>250</td>
<td>15</td>
<td>37%</td>
<td>18%</td>
<td>36%</td>
</tr>
<tr>
<td>250</td>
<td>20</td>
<td>34%</td>
<td>15%</td>
<td>42%</td>
</tr>
<tr>
<td>400</td>
<td>10</td>
<td>14%</td>
<td>9%</td>
<td>89%</td>
</tr>
<tr>
<td>400</td>
<td>15</td>
<td>9%</td>
<td>5%</td>
<td>92%</td>
</tr>
<tr>
<td>400</td>
<td>20</td>
<td>6%</td>
<td>2%</td>
<td>98%</td>
</tr>
</tbody>
</table>
Simulator Based on Approximator

Neural Network Approximator:
• Inputs:
  – Center of mass positions
  – Inertia values
  – Dominant wave height
  – Dominant wave frequency
• Output:
  – Instantaneous power

We develop a time domain simulator based on this neural network.

Simulator Based on Approximator

**Given:** ballast configuration, ballast control policy, wave climate

1. \( Energy = 0; \)
2. For each time step \( t: \)
   – Find mass of seawater in each ballast chamber using ballast control policy;
   – Based on seawater mass, find inertia and CG values;
   – Find power: \( P = NN(CG, I, \text{wave height}, \text{wave frequency}); \)
   – \( Energy += P \times \Delta t; \)

Return \( Energy \)
Evolutionary Algorithm: Solution Representation

- Vector of values defining ballast chamber geometry

Evolutionary Algorithm: Mutation

- Add small random values to elements of solution vector
- Constrain values to ensure feasible solutions
Evolutionary Algorithm: Fitness Assignment and Selection

• Fitness assignment carried out with time domain simulator based on neural network
• Selection carried out with epsilon-greedy selection

Results

• Tested different control loop times, and compared EA with hill-climbing
Computational Cost Analysis

- Comparing time required to run EA with our fitness modeling approach vs. directly using AQWA:

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Population size</th>
<th>Generations</th>
<th>NN (hours)</th>
<th>AQWA (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Power Calculation</td>
<td>n/a</td>
<td>n/a</td>
<td>$10^{-3}$</td>
<td>24</td>
</tr>
<tr>
<td>EA</td>
<td>100</td>
<td>1000</td>
<td>4</td>
<td>$4 \times 10^5$ (~46 years)</td>
</tr>
<tr>
<td>EA</td>
<td>500</td>
<td>1000</td>
<td>6</td>
<td>$2 \times 10^6$ (~228 years)</td>
</tr>
</tbody>
</table>

- Key: using AQWA to assign fitness would have made it impossible to use an EA to optimize the WEC

Discussion

- Fitness assignment is typically the most computationally expensive element of an EA

- To allow for computational tractability, we use fitness modeling to approximate the fitness assignment operator
  - Can increase algorithm speed by many orders of magnitude
Up Until Now...

- You have known $G(z)$
- You used $G(z)$ to assign fitness in an evolutionary algorithm
- What if you don’t have $G(z)$? Or what if computing $G(z)$ is too slow to run an EA? How do we provide feedback?

Fitness Modeling
- We can approximate $G(z)$, which allows for:
  - Implementing EAs in cases where we don’t know $G(z)$
  - Implementing EAs in computationally efficient manner

Outline

- Fitness modeling in evolutionary algorithms
  - What if we can’t directly compute fitness values?
  - What if fitness assignment operator is prohibitively slow?

- Learning from demonstration:
  - If a task is non-trivial to encode, can we just have a robot watch us complete the task in order to learn how to do it?
Learning From Demonstration

Develop agent policies based on examples

Learning from Demonstration Lecture Based on:

Motivation

• One of the main goals of robotics is that robots be used in real-world domains, helping people to complete tasks.
• Advances in robotics have allowed for the concept of human-robotic interaction (e.g. robots helping out around the house).
• We want to design algorithms that allow robots to learn how to autonomously perform tasks in order to help humans.
• Key requirements:
  – Robots must be safe around humans
  – Robots should autonomously complete tasks
Motivation

- One of the main goals of robotics is that robots be used in real-world domains, helping people to complete tasks.
- Advances in robotics have allowed for the concept of human-robotic interaction (e.g. robots helping out around the house).
- We want to design algorithms that allow robots to learn how to autonomously perform tasks in order to help humans.
- Key requirements:
  - Robots must be safe around humans
  - Robots should autonomously complete tasks
  - Robots must be easy for humans to interact with (most people are not experts in robotics)

Motivation: Learning Policies

- Learning a policy that maps states to actions is at the heart of robotic applications.
- Typical methods to develop policies:
  - Deriving policies from physical principles (physics, kinematics)
  - Learn a policy using reinforcement learning
  - Learn a policy using evolutionary algorithms
Motivation: Learning Policies

• Learning a policy that maps states to actions is at the heart of robotic applications.
• Typical methods to develop policies:
  − Deriving policies from physical principles (physics, kinematics)
  − Learn a policy using reinforcement learning
  − Learn a policy using evolutionary algorithms

Problem with these methods: require an expert to design and implement them! Average (non-robotics) people can’t implement these algorithms, but we want them to be able to use robots too.

Motivation: Learning from Demonstration

Can we develop a framework so humans without expert robotics knowledge can teach robots to autonomously perform tasks?
Learning from Demonstration (LfD)

- Policies are learned from examples.
- Examples are defined as sequences of state-action pairs demonstrated by the teacher.
- Examples are used to derive a policy for the robot.

LfD Problem Statement

- World consists of states $S$ and actions $A$, with a mapping between states defined by transition function $T(s'|s,a)$
- State is not fully observable, and the learner has access to an observed state $Z$ defined by the mapping $Z = M(s)$
- A policy $\pi$ maps a state to an action: $\pi(s) = a$
LfD: Key Issues

• How to gather examples?
• How to derive a policy based on examples?
• What are the limitations of the demonstration dataset?
How to Gather Examples

Two main classes of how to gather examples
• Demonstration: teleoperation, shadowing
• Imitation: sensors on teacher, external observation

Regardless of the way we get examples, they are of the form (state, action)

Record vs. Embodiment Mapping

• Record mapping: teacher execution $\rightarrow$ recorded execution
  – Refers to whether the exact states/actions experienced by the teacher during the demonstration are recorded within the dataset
• Embodiment mapping: recorded execution $\rightarrow$ learner
  – Refers to whether the states/actions recorded in the dataset are exactly those that the learner would observe/execute

Mappings denoted by $g(z,a)$
Demonstration: Teleoperation

• During teleoperation, a robot is operated by the teacher while recording from its own sensors.
• As the robot directly records the states/actions experienced during execution, the record mapping is direct: \( g(z,a) = I(z,a) \) (where \( I \) is the identity mapping)

Teleoperation Advantages

• Most direct method for information transfer
• Controlling robot with joystick applies to many domains: flying robotic UAVs, grasping objects, obstacle avoidance and navigation.
Teleoperation Disadvantages

• Often, external operation of the robot is not manageable.
  – For example, low-level motion demonstrations are difficult on systems with complex motor control, such as high degree of freedom humanoids.

Demonstration: Shadowing

• Shadowing: the robot learner records the execution using its own sensors while attempting to match or mimic the teacher motion as the teacher executes the task.
• There exists a non-direct record mapping: $g(z,a) \neq I(z,a)$ because the states/actions of the true demonstration execution are not recorded, but the states/actions of the robot’s mimicking execution are.
Shadowing Advantages

• Can represent more complex tasks that teleoperation, such as complex arm gestures

Shadowing Disadvantages

• Requires extra algorithmic step allowing the robot to actively shadow (rather than be teleoperated by) the teacher
Imitation: Sensors on Teacher

- Sensors located on the executing body are used to record the teacher execution.
- Record mapping is direct: \( g(z,a) = l(z,a) \)

Sensors on Teacher Advantages

- Allow for the same complex tasks to be encoded as in shadowing, but without the algorithmic overhead requiring real-time mimicking
Sensors on Teacher Disadvantages

• High overhead attached to specialized wearable sensors: require human wearable sensor suits, and customized surroundings such as rooms outfitted with cameras. This is often non-trivial and impractical

Imitation: External Observation

• Sensors external to the executing body are used to record the execution. The sensors may or may not be located on the robot learner.
• As the sensing is indirect, there exists a non-direct record mapping $g(z,a) \neq l(z,a)$
External Observation Advantages

• More general than sensors on teacher, as it is not limited by the overhead of specialized sensors and settings.

External Observation Disadvantages

• Less reliable data than sensors on teacher, because the states/actions of the teacher must be inferred.
Gathering Examples Summary

• Demonstration:
  – Teleoperation provides direct mapping, but is not practical in high degree of freedom systems.
  – Shadowing provides indirect mapping, but allows for more complex tasks than teleoperation.

• Imitation:
  – Sensors on teacher provides direct mapping, but requires overhead in equipment.
  – External observation provides indirect mapping, but is easier and more practical to implement.

None of these is the “right” approach, one of these is simply selected based on the task to be completed as well as the resources available.

LfD: Key Issues

• How to gather examples?
• How to derive a policy based on examples?
• What are the limitations of the demonstration dataset?
Deriving a Policy

- Mapping function: classification, regression
- System models: engineered reward functions, learned reward functions
- Plans

Mapping Function: Classification

- Categorize dataset into discrete classes, grouping examples with similar inputs together.
  - Used in cases where actions should be discrete (e.g. placing items in different boxes)
- Input to classifier: continuous state
- Output from classifier: discrete action

- Types of classifiers: Gaussian mixture models, decision trees, Bayesian networks
Mapping Function: Regression

- Mapping states to continuous action spaces
- Types of regression: neural networks with backprop, parametric regression, PCHIP approximation, etc.

System Models: Engineered Reward Functions

- Uses a state transition model of the world $T(s' | s, a)$, and from this derives a policy $\pi(z)=a$ (formulated within RL)
- Set of examples $(s, a)$ seed the policy
- User designs a reward function to fine tune the policy via reinforcement learning.
- Key idea: example set is sparse, so we want to fine tune policy in areas outside of example set.
System Models: Learned Reward Functions

- Defining an effective reward is often a non-trivial issue.
- Inverse reinforcement learning: learn (rather than hand-define) a reward function.
- In LfD, inverse reinforcement learning typically carried out by associating greater reward with states similar to those encountered during demonstration. Using transition model $T(s'|s,a)$, high rewards are given to states which lead to these high value states.

Plans

- Rather than mapping states directly to actions, represent the desired robot behavior as a plan.
  - Works well in tasks requiring sequential actions
- A planning framework represents the policy as a sequence of actions that lead from the initial state to the goal state.
Deriving Policies Summary

- Mapping function: develop a policy based only on examples
  - Classification for discrete actions
  - Regression for continuous actions
- System models: develop a policy that is seeded from examples (more generalizable)
  - Engineered rewards when we can hand-define reward functions
  - Learned rewards when we can’t
- Plans: assume examples are coming from a sequential task
  - Represent the policy as a sequence of actions from an initial state to a goal state.

LfD: Key Issues

- How to gather examples?
- How to derive a policy based on examples?
- What are the limitations of the demonstration dataset?
Limitations of Demonstration Dataset

- Undemonstrated states
- Poor quality data

Undemonstrated States

- In all but the most simple domains, the teacher is unable to demonstrate the correct action for every state.
- What should the robot do in unencountered states?
  - Generalize from existing demonstrations: interpolation, k-nearest neighbor
  - RL: learn a policy in unencountered states, use demonstrations as policy in encountered states
  - Acquire new demonstrations: re-engage the teacher with a request for additional demonstrations when robot reaches novel states.
Poor Quality Data

• LfD typically assumes the dataset contains high quality demonstrations.
• In reality, teacher demonstrations may be:
  − Ambiguous
  − Unsuccessful
  − Suboptimal
• How do we deal with poor quality data?
  − Eliminate examples that are obviously suboptimal or ambiguous
  − Let the learner optimize the policy learned from demonstration via RL.

Up Until Now...

• You have had a ton of data
• You could use the data to perform gradient descent or Q-learning to find a policy.
• What if you don’t have a lot of data? Can you learn a policy from only a few examples?

Learning from Demonstration
• A teacher provides examples (limited in size)
• Policies can be derived from these examples
  − We may refine policies afterwards using reinforcement learning