Contextual Reinforcement Learning
for
System Identification
with
Noisy Data

Joshua Hughes
NW Computational Intelligence Laboratory (NWCIL)
Systems Science Graduate Program
Portland State University

Teuscher Lab
November 19, 2010
Presentation Outline

- Motivation, concepts, and definitions
- Background and previous work
- Conceptual overview
  - context discerning multi-function network (CDMFN)
  - contextual reinforcement learning (CRL)
- Experimental design
- Preliminary results
- Future work
Systems are becoming* increasing complex, requiring increasing complex control systems. (* or we are realizing they are more complex than we thought)

Humans are good at knowing how and when to change their actions relative to context, while machines are not.** (** generally speaking)

Still, humans are not able to control systems that
1) are too complex,
2) are unreachable, or
3) exceed our levels of perception.

If we want machines to manifest human-like control, we need to create machines that can discern context, select appropriate control policies, and switch between them efficiently and effectively (quickly and near-optimally).
“Black box” methods work, but we don't always know how or why.

We want to know what's in the black box.

If we examine a neural network (our black box) after it has solved a given problem, we may be able to identify its component parts and use that knowledge to structure our learning algorithm.

*We can break our learning system into learning subsystems.*
What do we mean by control?

Achieving and/or maintaining a desired state (or set of states) of a plant in some environment subject to design objectives.

plant + environment + objectives = context

What do we mean by human-like control?

1. After a human learns a set of related identification and/or control tasks, when presented with a novel task of the same genre, the human generates close-to-optimal performance on the new task (i.e. effective selection from experience).

2. The more knowledge a human attains, the speed and efficiency of performing a task are improved; in contrast to most AI systems thus far developed, wherein the more knowledge acquired (typically stored as “rules”), the slower the decision/action processing.

(Lendaris, 2009)
An approach to creating human-like control

Create machines that can learn *general* knowledge about a set of problems by first learning solutions for *specific* problems from that set, i.e., by developing “experience.” We characterize this process as *context discernment*.

A *context-discerning* machine is able to:

- monitor selected attributes of whatever is its *focus*,
- detect “large enough” changes in these attributes, and
- select an appropriate *solution* from its accumulated experience

Examples of focus are:

- what a controller in a control system pays attention to, and
- what an agent performing system identification pays attention to.

A solution is selected from a collection of previously developed solutions (“*experience repository*”). An objective of this line of research is to achieve an agent with the ability to “interpolate” solutions when an exact match is not in the repository, i.e., to solve a set of related problems.
Background and previous work


Main conclusions

1) Adaptation does not require weight changes.
2) Individual nodes of FWNN fill specialized roles.
3) FWNNs can be reformulated into two parts: a context-discerning network (CDN) and a multifunction network (MFN) to form a CDMN.
4) CDMNs can be trained via a DHP-derived process.
Universal Approximators

With enough hidden units, a neural network can approximate any function with arbitrary precision. (Proofs for MLP, RBF, and GRNN.)

\[
\begin{align*}
\frac{dQ_1}{dt} &= f_1 (Q_1, Q_2, \ldots Q_n) \\
\frac{dQ_2}{dt} &= f_2 (Q_1, Q_2, \ldots Q_n) \\
\vdots \\
\frac{dQ_n}{dt} &= f_n (Q_1, Q_2, \ldots Q_n)
\end{align*}
\]

If a system can be represented as a set of differential equations, it can be represented by a universal approximator, i.e. neural network.
Universal Approximator
MLP network with back-propagation

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$  \hspace{1cm} (1)

Hebbian rule

$$\Delta w_{ji} = \eta (t_j - o_j)i_i$$  \hspace{1cm} (2)

finite difference equivalent

$$\begin{bmatrix} \Delta w_{10} \\ \vdots \\ \Delta w_{NM} \end{bmatrix} = \begin{bmatrix} f_1(w_{10}, \ldots, w_{NM}, o_0, \ldots, o_N, t_0, \ldots, t_N, i_0, \ldots, i_M, \eta) \\ \vdots \\ f_{NM}(w_{10}, \ldots, w_{NM}, o_0, \ldots, o_N, t_0, \ldots, t_N, i_0, \ldots, i_M, \eta) \end{bmatrix}$$  \hspace{1cm} (3)

(compact version)

$$\Delta w = f(w, o, t, i, \eta)$$  \hspace{1cm} (4)
1. Eliminate output from weight update equations (network). Output of feedforward net depends only on inputs and weights.
2. Eliminate weight readouts from synapses (from the feedforward network) and replace by adding an accumulation unit (discrete-time integrator) to the weight update network.
Transforming Learning Algorithms to Fixed Weight Networks

3. Eliminate direct connections to synapses (in the feed forward network); add sigma-pi units (product units) with weight accumulators as inputs.

*Problem:* Some weights bypass layers—not strictly feedforward.
“All the changes in the network occur in the accumulation units of the weight update network. As these units play the role of synaptic weights, one might complain that our claim of learning with fixed weights amounts to little more than semantics—we have simply redrawn the boundaries of synapses. This claim is true.”

Lemma: Learning algorithms and nonlinear dynamic systems described by fixed coefficient equations are indistinguishable.
Transforming Learning Algorithms to Fixed Weight Networks

4. Replace feedforward sigma-pi network with a layered *universal approximation* network.

\[ o_j = \sum_i w_{ji} g \left( \sum_h w_{ih} h_h \right) \quad (5) \]

\[ o_j = h(i, w) \quad (6) \]

* May have to be a very large network.
5. Combine the universal approximation network with the weight update network to get a fixed-weight network with a uniform structure.
6. Consider all network outputs as potential weight updates. These potential weight updates can now be thought of as state variables.
Transforming Learning Algorithms to Fixed Weight Networks

7. Reduce number of weight vector components fed to the hidden units of the fixed-weight network.

\[ \omega_j = \sum_i w_{ji} w_i \]  \hspace{1cm} (7)

This network is a “threshold network.”

similar to Fourier series
The hidden-layer units compute a set of basis functions analogous to sine waves of fixed amplitude with adjustable phases.
Transforming Learning Algorithms to Fixed Weight Networks

Learning algorithms and nonlinear dynamic systems described by fixed coefficient equations are indistinguishable.

- Reasonably small networks can approximate the functions we require.
- Good learning algorithms do not require precise computation.
- Optimal learning algorithms can be found by a search of all possible fixed weights for a network structure.
- An integration mechanism can allow networks to learn without the correlated synaptic activity required by the Hebbian rule.
- Fixed parameters can be entirely responsible for defining learning algorithms.

A network whose only variable weights are hidden-layer thresholds is capable of universal approximation.
Meta-Learning

Two simultaneous processes during meta-learning

1) Supervisory system learns a set (or family) of mappings.
2) A subordinate system learns a specific mapping.

Both learners must leverage the regularities of their respective problems to efficiently solve them: the supervisor has a broader scope and must learn the common characteristics of all the problems, while the subordinate system must learn the specific characteristics of individual problems.

Implementation

Train the network to learn a set of mappings by using many examples from many different specific mappings within that set. (May often include data from previous time step during training.)
Context-Discerning Multi-Function Network (CDMFN)

Recurrence allows the context discerner (CD) and the multi-function network (MFN) to adapt *without weight changes*
Context-Discerning Multi-Function Network (CDMFN)

a simple example: interceptor
The sequence for training is $b = 0.5, 0, -0.5, 0.5, 0, -0.5$; for testing $b = 0.5, 0, -0.5, 0.25, -0.75$.

(a) By iteration 31 the perceptron has seen all three lines already, but it does not retain that any memory and so must relearn each one again.

(b) The perceptron does not generalize, it only maps the last mapping it was trained on, $b = -0.5$. 
The sequence for training is $b = 0.5, 0, -0.5, 0.5, 0, -0.5$; for testing $b = 0.5, 0, -0.5, 0.25, -0.75$.

(c) The CDMFN has higher errors than the perceptron at first, but then performs about as well.

(d) The CDMFN generalizes well, identifying all three lines it was trained on plus two new lines after only a few iterations.
The difference in performance lies with how $b$ is updated.

In the perceptron $b$ (bias) updated by learning rate $\eta$ times error $e$.

* Because the bias is fixed after training, the perceptron is capable of only one mapping.

The CD adjusts the weight $\eta$ of the error until it outputs the correct $b$. After training $\eta$ is fixed (like the bias in the perceptron).

* The CD receives an error signal during testing and adjusts $b$ (context parameter) until this error goes to zero.

Recurrence allows fixed-weight network to adapt.

Specific vs. general learning:
- the perceptron learns an intercept
- the CD learns how to adjust the intercept
Dual Heuristic Programming (DHP) is an adaptive critic implementation of non-linear sequential optimization using reinforcement learning.

DHP learns by changing the control in such a way as to minimize the expected “cost-to-go” (the sum of all future utilities).

The critic is used to navigate the “cost-to-go surface.”

The cost-to-go surface (or “J-surface”) is created from the matrix of partial derivatives for the control system (plant state, control, utility): the Jacobian.

The J-surface is updated throughout the learning process, changing with each update (iteration).

The goal is to navigate the surface the cost-to-go surface (even as it changes) in a gradient-descent type manner, always making each step optimal.

Contextual Reinforcement Learning repurposed DHP with signal substitution
Context Discerning Multi-Function Network (CDMFN) trained by Contextual Reinforcement Learning (CRL)
Contextual Reinforcement Learning

critic error signal

$\text{Error}_{\text{Critic}} = \left( \frac{\partial U(t)}{\partial C_D(t)} + \gamma \left( \lambda(t+1) \left( I + \frac{\partial \Delta C_D(t)}{\partial C_D(t)} \right) \right) - \lambda(t) \right)^2$

via primary utility function (output error)

via dual of CD

$\text{Error}_{\text{Controller}} = \lambda(t+1)$

$\nabla J(t+1)$

$\nabla J(t)$

via critic

secondary utility function ("cost-to-go")
trained

Context Discerning Multi-Function Network
System Identification (SID)

What system produced the observed data? (or, more generally, what system in what context?)
System Identification (SID)

What system produced the observed data? (or, more generally, what system in what context?)

INPUT
SYSTEM
OUTPUT

OBSERVER
	noisy

changing

noisy

INPUT
SYSTEM
OUTPUT
CRL SID task
MLP with 2 inputs, 1 output, and 1 context parameter

\[ C \]
\[ A \]
\[ x_1 \]
\[ x_2 \]
\[ \text{system} \]
\[ \rightarrow \]
\[ \rightarrow \]
\[ y \]
Experimental design
Noise

normally distributed noise
$\mu, \sigma, f$

\[ x = x + f \times \text{normrnd}(\mu, \sigma) \]

$\mu = \{0, \sim0\}$
$\sigma = 0.33$
$f = \{0, 0.25, 0.50, 1.00\}$
CDMFN
MLP with 2 inputs, 1 output, and 1 context parameter
CDMFN
MLP with 2 inputs, 1 output, and 1 context parameter.
Preliminary results
test results for trained CDMFN with no noise
test results for trained CDMFN with imperfect model
(linear output instead of tanh)
test results for CDMFN that didn't learn

*note y-scale
Conclusions

- Contextual Reinforcement Learning (CRL) appears to be somewhat robust to moderate amounts of noise, both random and systematic.
- The CD can learn to ignore input not relevant to the task.
- The CD can learn a set of mappings even when the model structure does not exactly match the system structure.
- Many additional experiments to run!
Questions or comments?
Adaptive Critic Methods

- Adaptive critic (AC) methods are implementations of adaptive dynamic programming (ADP) using reinforcement learning (RL).

- ADP is a general, non-linear sequential optimization method based on Bellman's Principle of Optimality:

  An optimal trajectory of control actions has the property that no matter how the current state was reached, the remainder of the trajectory is optimal.*

  *optimal is defined as minimizing (or maximizing) the sum of all the future utilities → the “cost-to-go”

- ADP is a “good” and computationally tractable approximation of dynamic programming—a mathematically proven method for designing optimal controllers, but too computationally expensive to be implemented for a “real-time” (online) controller.
Adaptive Critic Methods

- AC methods learn by changing the control in such a way as to minimize the expected cost-to-go (the sum of all future utilities).
- The critic is used to navigate the “cost-to-go surface.”
- The cost-to-go surface is created from the matrix of partial derivatives for the control system (plant state, control, utility): the Jacobian.
- Jacobian(s) are updated throughout the learning process → the cost-to-go surface is constantly changing with each update.
- The goal is to navigate the surface the cost-to-go surface in a gradient-descent type manner, always making each step optimal.
- Feedback can be qualitative (+ or -).
Adaptive Critic Methods

THREE MAIN LOOPS
(and possibly a fourth)

Each role can be filled by a machine learning algorithm.
Dual Heuristic Programming (DHP)
Dual Heuristic Programming (DHP)

plant \quad R(t+1)
Dual Heuristic Programming (DHP)
Dual Heuristic Programming (DHP)
Dual Heuristic Programming (DHP)

Diagram:
- **R(t)** enters the **action** block.
- **u(t)** moves from **action** to **plant**.
- **R(t+1)** leaves the **plant** block.
- **u(t)** connects from **action** to a **utility** block.
Dual Heuristic Programming (DHP)
Dual Heuristic Programming (DHP)
Dual Heuristic Programming (DHP)

- Plant
- Action
- Critic
- Controller feedback
- Utility

R(t), R(t+1), λ(R(t)), λ(R(t+1))
Dual Heuristic Programming (DHP)

- **Plant**
  - Action: \( R(t) \) → \( u(t) \) → **Plant**
  - Cost-to-Go: \( \lambda^*(R(t)) \)

- **Critic**
  - Feedback: \( \lambda(R(t+1)) \)
  - \( R(t+1) \) → **Critic**
  - \( \lambda(R(t)) \)

- **Controller**
  - Feedback: \( \lambda(R(t+1)) \)

- **Utility**
  - Feedback: \( \lambda(R(t)) \)

- **Utility**
  - Feedback: \( \lambda^*(R(t)) \)
Dual Heuristic Programming (DHP)

mathematics: Bellman equation

\[ J(t) = \sum_{k=0}^{\infty} \gamma^k U(t+k) \]

rewrite as recursive function

\[ J(t) = U(t) + \gamma J(t+1) \]

*secondary utility function*

*primary utility function*

*discount factor*
cost-to-go surface

find optimal trajectory
via critic

\[ J(t) \]

\[ R(t) \]

\[ J(0) \]

state

\[ R(0) \]
cost-to-go surface

find optimal trajectory
via critic

\[ J(t) \]

\[ R(t) \]

state

cost-to-go
cost-to-go surface

find optimal trajectory via critic

$J(t)$

$R(t)$

$J(0)$

$J(1)$

$R(0)$

$R(1)$

$R(2)$

state

cost-to-go

$J(t)$
cost-to-go surface
find optimal trajectory via critic
cost-to-go surface

find optimal trajectory via critic

J(t)

R(t)

J(0)

J(1)

J(2)

state

R(0)  R(3)
cost-to-go surface

find optimal trajectory via critic

\[ J(t) \]

\[ R(t) \]

\[ J(0) \]

\[ J(1) \]

\[ J(2) \]

\[ R(0) \]

\[ R(1) \]

\[ R(2) \]

\[ R(3) \]

state
Dual Heuristic Programming (DHP)

mathematics: chain rule

\[ \Delta w_{ij}(t) = lcoeff \cdot \frac{\partial}{\partial w_{ij}(t)} J(t) \]

\[ \frac{\partial}{\partial w_{ij}(t)} J(t) = \sum_{k=1}^{a} \frac{\partial}{\partial u_k(t)} J(t) \cdot \frac{\partial}{\partial w_{ij}(t)} u_k(t) \]

\[ \frac{\partial}{\partial u_k(t)} J(t) = \frac{\partial}{\partial u_k(t)} U(t) + \frac{\partial}{\partial u_k(t)} J(t+1) \]

\[ \frac{\partial}{\partial u_k(t)} J(t+1) = \sum_{s=1}^{n} \frac{\partial}{\partial R_s(t+1)} J(t+1) \cdot \frac{\partial}{\partial u_k(t)} R_s(t+1) \]

\[ \frac{\partial}{\partial R_s(t+1)} J(t+1) = \lambda_s(t+1) \]
Dual Heuristic Programming (DHP)

mathematics: chain rule

\[ \Delta w_{ij}(t) = l\text{coef} \cdot \frac{\partial}{\partial w_{ij}(t)} J(t) \]

\[
\frac{\partial}{\partial w_{ij}(t)} J(t) = \sum_{k=1}^{a} \frac{\partial}{\partial u_k(t)} J(t) \cdot \frac{\partial}{\partial w_{ij}(t)} u_k(t)
\]

\[
\frac{\partial}{\partial u_k(t)} J(t) = \frac{\partial}{\partial u_k(t)} U(t) + \frac{\partial}{\partial u_k(t)} J(t + 1)
\]

\[
\frac{\partial}{\partial u_k(t)} J(t + 1) = \sum_{s=1}^{n} \frac{\partial}{\partial R_s(t + 1)} J(t + 1) \cdot \frac{\partial}{\partial u_k(t)} R_s(t + 1)
\]

\[
\frac{\partial}{\partial R_s(t + 1)} J(t + 1) = \lambda_s(t + 1)
\]

weight update ✠

critic output

across plant
Dual Heuristic Programming (DHP)

mathematics: target for critic

critic target

\[ \lambda_s^o(t) = \frac{\partial}{\partial R_s(t)} J(t) = \frac{\partial}{\partial R_s(t)} (U(t) + J(t + 1)) \]

primary utility function

\[ U(t) = \sum_i \left( a_i \cdot R_i(t) \right) + \sum_j \left( c_j \cdot u_j^d(t) \right) \]

critic feedback

\[ e_s = (\lambda_s^o(t) - \lambda_s(t))^2 \]
Dual Heuristic Programming (DHP)
mathematics: target for critic

\[ \lambda_s^o(t) = \frac{\partial}{\partial R_s(t)} U(t) \]

\[ + \sum_{j=1}^{a} \left( \frac{\partial}{\partial u_j(t)} U(t) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right) \]

\[ + \sum_{k=1}^{n} \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \cdot \frac{\partial}{\partial R_s(t)} R_k(t+1) \right) \]

\[ + \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \cdot \frac{\partial}{\partial u_j(t)} R_k(t+1) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right) \right\} \]
Dual Heuristic Programming (DHP)

Mathematics: target for critic

\[
\lambda_s^o(t) = \frac{\partial}{\partial R_s(t)} U(t) \\
+ \sum_{j=1}^a \left( \frac{\partial}{\partial u_j(t)} U(t) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right) \\
+ \sum_{k=1}^n \left( \frac{\partial}{\partial R_k(t)} J(t+1) \cdot \frac{\partial}{\partial R_s(t)} R_k(t+1) \right) \\
+ \sum_{k=1}^n \left\{ \sum_{j=1}^a \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \cdot \frac{\partial}{\partial u_j(t)} R_k(t+1) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right) \right\}
\]
Dual Heuristic Programming (DHP)

mathematics: target for critic

\[ \lambda_s^c(t) = \frac{\partial}{\partial R_s(t)} U(t) \]

\[ + \sum_{j=1}^{a} \left( \frac{\partial}{\partial u_j(t)} U(t) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right) \]

\[ + \sum_{k=1}^{n} \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \cdot \frac{\partial}{\partial R_s(t)} R_k(t+1) \right) \]

\[ + \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \cdot \frac{\partial}{\partial u_j(t)} R_k(t+1) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right) \right\} \]
Dual Heuristic Programming (DHP)
mathematics: target for critic

\[
\lambda_s^c(t) = \frac{\partial}{\partial R_s(t)} U(t) + \sum_{j=1}^{a} \left( \frac{\partial}{\partial u_j(t)} U(t) \right) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) + \sum_{k=1}^{n} \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \right) \cdot \frac{\partial}{\partial R_s(t)} R_k(t+1) + \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \right) \cdot \frac{\partial}{\partial u_j(t)} R_k(t+1) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right\}
\]
Dual Heuristic Programming (DHP)

mathematics: target for critic

\[
\lambda_s^o(t) = \frac{\partial}{\partial R_s(t)} U(t) + \sum_{j=1}^{a} \left( \frac{\partial}{\partial u_j(t)} U(t) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right) + \sum_{k=1}^{n} \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \cdot \frac{\partial}{\partial R_s(t)} R_k(t+1) \right)
\]

controller

utility

plant

critic

\[
+ \sum_{k=1}^{n} \left\{ \sum_{j=1}^{a} \left( \frac{\partial}{\partial R_k(t+1)} J(t+1) \cdot \frac{\partial}{\partial u_j(t)} R_k(t+1) \cdot \frac{\partial}{\partial R_s(t)} u_j(t) \right) \right\}
\]
Dual Heuristic Programming (DHP)
example problem: pole cart

two to six state variables (translation and rotation), one control, $F$
GOAL: balance pole by applying horizontal force
Dual Heuristic Programming (DHP)
pole cart state equations

\[ \ddot{\theta}(t) = \frac{g \sin \theta_t + \cos \theta_t \left[ -F - ml \dot{\theta}_t^2 \sin \theta_t + \mu_c \text{sgn}(\dot{x}_t) \right]}{m_c + m} - \frac{\mu_p \dot{\theta}_t}{ml} \]

\[ l \left[ \frac{4}{3} - \frac{m(\cos \theta_t)^2}{m_c + m} \right] \]

\[ \ddot{x}_t(t) = \frac{F_t + ml \left[ \dot{\theta}_t^2 \sin \theta_t - \dot{\theta}_t \cos \theta_t \right] - \mu_c \text{sgn}(\dot{x}_t)}{m_c + m} \]
Dual Heuristic Programming (DHP)

pole cart state Jacobians

\[ \frac{\partial}{\partial R(t)} R(t+1) = \begin{bmatrix}
\frac{\partial x_{(t+1)}}{\partial x_t} & \frac{\partial x_{(t+1)}}{\partial \dot{x}_t} & \frac{\partial x_{(t+1)}}{\partial \ddot{x}_t} & \frac{\partial x_{(t+1)}}{\partial \theta_t} & \frac{\partial x_{(t+1)}}{\partial \dot{\theta}_t} & \frac{\partial x_{(t+1)}}{\partial \ddot{\theta}_t} \\
\frac{\partial \dot{x}_{(t+1)}}{\partial x_t} & \frac{\partial \dot{x}_{(t+1)}}{\partial \dot{x}_t} & \frac{\partial \dot{x}_{(t+1)}}{\partial \ddot{x}_t} & \frac{\partial \dot{x}_{(t+1)}}{\partial \theta_t} & \frac{\partial \dot{x}_{(t+1)}}{\partial \dot{\theta}_t} & \frac{\partial \dot{x}_{(t+1)}}{\partial \ddot{\theta}_t} \\
\frac{\partial \ddot{x}_{(t+1)}}{\partial x_t} & \frac{\partial \ddot{x}_{(t+1)}}{\partial \dot{x}_t} & \frac{\partial \ddot{x}_{(t+1)}}{\partial \ddot{x}_t} & \frac{\partial \ddot{x}_{(t+1)}}{\partial \theta_t} & \frac{\partial \ddot{x}_{(t+1)}}{\partial \dot{\theta}_t} & \frac{\partial \ddot{x}_{(t+1)}}{\partial \ddot{\theta}_t} \\
\frac{\partial \theta_{(t+1)}}{\partial x_t} & \frac{\partial \theta_{(t+1)}}{\partial \dot{x}_t} & \frac{\partial \theta_{(t+1)}}{\partial \ddot{x}_t} & \frac{\partial \theta_{(t+1)}}{\partial \theta_t} & \frac{\partial \theta_{(t+1)}}{\partial \dot{\theta}_t} & \frac{\partial \theta_{(t+1)}}{\partial \ddot{\theta}_t} \\
\frac{\partial \dot{\theta}_{(t+1)}}{\partial x_t} & \frac{\partial \dot{\theta}_{(t+1)}}{\partial \dot{x}_t} & \frac{\partial \dot{\theta}_{(t+1)}}{\partial \ddot{x}_t} & \frac{\partial \dot{\theta}_{(t+1)}}{\partial \theta_t} & \frac{\partial \dot{\theta}_{(t+1)}}{\partial \dot{\theta}_t} & \frac{\partial \dot{\theta}_{(t+1)}}{\partial \ddot{\theta}_t} \\
\frac{\partial \ddot{\theta}_{(t+1)}}{\partial x_t} & \frac{\partial \ddot{\theta}_{(t+1)}}{\partial \dot{x}_t} & \frac{\partial \ddot{\theta}_{(t+1)}}{\partial \ddot{x}_t} & \frac{\partial \ddot{\theta}_{(t+1)}}{\partial \theta_t} & \frac{\partial \ddot{\theta}_{(t+1)}}{\partial \dot{\theta}_t} & \frac{\partial \ddot{\theta}_{(t+1)}}{\partial \ddot{\theta}_t}
\end{bmatrix} \]
Dual Heuristic Programming (DHP)
pole cart state Jacobians

\[
\frac{\partial}{\partial R(t)} R(t+1) = \begin{bmatrix}
1 & \tau & 0.5\tau^2 & 0 & 0 & 0 \\
0 & 1 & \tau & 0 & 0 & 0 \\
0 & 0 & 0 & \partial\ddot{x}_{\theta},_t & \partial\ddot{x}_{\dot{\theta}},_t & \partial\ddot{x}_{\ddot{\theta}},_t \\
0 & 0 & 0 & 1 & \tau & 0.5\tau^2 \\
0 & 0 & 0 & 0 & 1 & \tau \\
0 & 0 & 0 & \partial\ddot{\theta}_{\theta},_t & \partial\ddot{\theta}_{\dot{\theta}},_t & 0
\end{bmatrix}
\]

\[
\frac{\partial}{\partial u(t)} R(t+1) = \begin{bmatrix}
\frac{\partial x_{(t+1)}}{\partial u_t} & \frac{\partial \dot{x}_{(t+1)}}{\partial u_t} & \frac{\partial \ddot{x}_{(t+1)}}{\partial u_t} & \frac{\partial \theta_{(t+1)}}{\partial u_t} & \frac{\partial \dot{\theta}_{(t+1)}}{\partial u_t} & \frac{\partial \ddot{\theta}_{(t+1)}}{\partial u_t}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\alpha_t (0.5)dt^2 & \alpha_t dt & \alpha_t & \beta_t (0.5)dt^2 & \beta_t dt & \beta_t
\end{bmatrix}
\]

\[
\alpha_t = \frac{1}{m_c + m}, \quad \beta_t = \frac{-\alpha_t \cos^2 \theta_t}{\frac{4l}{3} - ml \alpha_t \cos \theta_t}
\]