Charles Desjardins

Computer Science and Software Engineering Department
Laval University
desjardins@damas.ift.ulaval.ca
http://damas.ift.ulaval.ca

October 12th, 2007
Plan

1. Introduction
2. Reinforcement Learning and Function Approximation
3. Neural Fitted Q-Iteration Algorithm
4. Empirical Results
5. Conclusion
6. Questions
Sommaire

1 Introduction
   - Overview

2 Reinforcement Learning and Function Approximation
   - Reinforcement Learning
   - Function Approximation

3 Neural Fitted Q-Iteration Algorithm

4 Empirical Results

5 Conclusion
   - Related Work
   - Article Review

6 Questions
Efficient value function representation techniques are important in order to solve complex problems using reinforcement learning.

**Discretization of the state space**

For most real world problems, discretizing the state space to use classical algorithms, for which theoretical convergence proofs exist, cannot efficiently solve the problem:

- State space grows exponentially with the number of state variables.
- Difficult to consider problems that need a continuous state space.
Function Approximation Techniques

Advantages:
- No issues with state space dimensionality.
- Can work efficiently with continuous state variables.
- Generalization of experience.

Disadvantages:
- Still an approximation.
- Side effect of generalization: Weight updates induced by the visit of a \((s, a)\) pair affect the value of many unrelated \((s, a)\) pairs, resulting in long learning times.
- Convergence issues with function approximation.
Neural Fitted Q-Iteration (NFQ) Algorithm

We wish to:
- keep the advantages of value function approximation.
- reduce the negative effects brought by global changes to the approximate function.

Neural Fitted Q-Iteration (NFQ) Algorithm Characteristics

- Model-free Reinforcement Learning.
- Function approximation based on a multi-layer perceptron.
- Training of the neural network by storing and reusing transitions.

In brief, the article presents an efficient Q-Value approximation method using neural networks and an enhanced weight update.
Sommaire

1 Introduction
   - Overview

2 Reinforcement Learning and Function Approximation
   - Reinforcement Learning
   - Function Approximation

3 Neural Fitted Q-Iteration Algorithm

4 Empirical Results

5 Conclusion
   - Related Work
   - Article Review

6 Questions
Markov Decision Process

**MDP Framework**

- \( S \) : a set of states.
- \( A \) : a set of actions.
- \( T(s, a, s') \) : a transition function.
- \( r(s, a) \) : a reward/cost function.

The goal is to find the optimal policy that minimizes the expected cost/maximizes the expected reward for every state.
Q-Learning

Based on the MDP framework, the Q-Learning algorithm uses the following update rule:

Q-Learning update rule

\[
Q_{t+1}(s, a) = Q_t(s, a) + \alpha (r(s, a) + \gamma \max_{a'} Q_t(s', a') - Q_t(s, a))
\]

Convergence to the optimal value function has been proven for finite state and action spaces, when the state action pairs are visited infinitely often.

**Curse of dimensionality**: Instead of tabular representations, we are looking for different value function approximation techniques.
Parametrized Value Functions

For example, many approaches use a linear parametrized value function similar to:

\[ F(x) = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n \]

where \( x_1, \ldots, x_n \) are the state variables and \( \theta_1, \ldots, \theta_n \) their respective weights. We want to learn the value of the parameters that best represent the value function throughout the entire state space.

Example of Methods

- Tile Coding.
- Multi-Layered Perceptron.
Learning with function approximators: Comes down to calculating the gradient of an error function and updating the weights in the opposite direction of the gradient.

### Gradient-Descent Methods

1. Calculate the squared approximation error:

\[
e(t) = \frac{(d(t) - a_\theta(t))^2}{2}
\]

- \(d(t)\) is the desired value.
- \(a(t)\) is the current estimation according to the approximator.

2. Calculate the gradient of the error function:

\[
\frac{\partial e(t)}{\partial \theta_i}
\]

3. Move the weights in the direction that reduces the error:

\[
\theta_i \leftarrow \theta_i - \alpha \frac{\partial e(t)}{\partial \theta_i}
\]
Q-Learning with NN

Why?
- Neural networks have the advantages of value function approximation methods.
- Neural networks can approximate non-linear functions.

How?
1. Compute the error function, which is given in this case by the TD-Error:

\[ E = \frac{((r(s, a) + \gamma \max_{a'} Q(s', a')) - Q(s, a))^2}{2} \]

2. Use the backpropagation algorithm to compute the gradient of the error according to the weights.
3. Modify the network’s weights to reduce the error.
Q-Learning with NN

Convergence issues

- Updates are done on-line, after each new visit of an \((s, a)\) pair. Changing the value function for a \((s, a)\) pair modifies the function for many other pairs, resulting in slow learning.
- There are no convergence proofs for non-linear approximators.
- Simple linear function approximation examples have been shown to diverge.
Sommaire

1 Introduction
   - Overview

2 Reinforcement Learning and Function Approximation
   - Reinforcement Learning
   - Function Approximation

3 Neural Fitted Q-Iteration Algorithm

4 Empirical Results

5 Conclusion
   - Related Work
   - Article Review

6 Questions
Rather than updating the value function on-line, after each visit of an 
(s, a) pair, the NFQ update is done off-line, using a set of transition
experiences.

NFQ steps

1. Generation of the training set.
2. Training of the neural network.
Generation of the training set

We want to generate transition samples of the form \(((s, a), \text{target})\).

**Target value**

- Computed using the current value function \(Q_t\).
- Corresponds to \(r(s, a) + \gamma \max_{a'} Q_t(s', a')\).
- This target is used to compute the error.

**Samples are generated by interaction with the environment:**

- by generating transition by taking random actions.
- by adding incrementally samples to the training set using greedily the current value function estimate.
Using a training set $P$ containing $D$ transitions and target values, we can train the network.

A training set enables the use of **batch learning methods** that sweep through the set many times, until the corresponding value function has been learned.

Some batch neural network learning algorithms have better convergence properties than normal gradient-descent based algorithms. The author uses the RPROP algorithm to train its network.
Algorithm

function NEURAL FITTED Q-ITERATION(D) return $Q_N$, the current Q-Value function estimates

Inputs: $D$: A set of transition samples.
Statics: $N$: The number of episodes to execute.

$K = 0$
$Q_0 \leftarrow \text{INIT}NEURAL\text{NETWORK}()$

repeat

Generate training set $P = \{(input^l, target^l), l = 1, \ldots, \#D\}$ where:

$\text{input}^l = s^l, a^l$
$\text{target}^l = r(s, a) + \gamma \max_{a'} Q(s, a')$
$Q_{K+1} \leftarrow \text{TRAIN}NEURAL\text{NETWORK}(P)$

$k = k + 1$

until $k < N$ return $Q_N$

end function
Sommaire

1 Introduction
   • Overview

2 Reinforcement Learning and Function Approximation
   • Reinforcement Learning
   • Function Approximation

3 Neural Fitted Q-Iteration Algorithm

4 Empirical Results

5 Conclusion
   • Related Work
   • Article Review

6 Questions
Pole Balancing Task

Task Description

- **Goal**: Balance a pole in the upright position by applying appropriate forces.
- **State**: Angle of the pole and angular velocity (both continuous values).
- **Actions**: Left, right and no force.

NFQ Neural Network Configuration

- **Input**: 3 neurons.
- **Hidden Layers**: 2 layers of 5 neurons.
- **Output**: 1 neuron.
Learning Methodology

- Comparison with LSPI and Q-Learning with Exp. Replay.
- NFQ generates samples by taking random actions.
- Average training episode length of 6 steps.

Controller Performance Test Methodology

- The resulting controller was evaluated using 50 test episodes for different numbers of random learning episodes.
Pole Balancing Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Training Episodes</th>
<th>Balancing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSPI</td>
<td>1000</td>
<td>Average of 285 secs.</td>
</tr>
<tr>
<td>QL with Exp. Replay</td>
<td>750</td>
<td>About 300 secs.</td>
</tr>
<tr>
<td>NFQ</td>
<td>200</td>
<td>300 secs.</td>
</tr>
</tbody>
</table>

Using 200 learning episodes, NFQ can generate a policy that result in balancing the pole for 300 seconds. This shows the generalization efficiency of this method.
Mountain Car Task

Task Description

- **Goal**: Reach the top of the mountain.
- **State**: Position and velocity.
- **Actions**: Go left or right.

NFQ Neural Network Configuration

- **Input**: 3 neurons.
- **Hidden Layers**: 2 layers of 5 neurons.
- **Output**: 1 neuron.
Mountain Car Results

Learning Methodology

- NFQ generates samples incrementally, as random actions cannot help us travel the entire state space.
- Training episodes length of 50 steps.

Controller Performance Test Methodology

- Learning controller results averaged over 20 experiments of 500 episodes.
- The resulting controllers are tested using 1000 random starting positions.
- Successful policy: policy that reaches the goal state for all 1000 random starting positions.
### Mountain Car Results

#### NFQ Results - First Successful Policy

<table>
<thead>
<tr>
<th>First Successful Policy</th>
<th>Training Episodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>71</td>
</tr>
<tr>
<td>Best</td>
<td>10</td>
</tr>
<tr>
<td>Worst</td>
<td>243</td>
</tr>
</tbody>
</table>

#### NFQ Results - Best Policy

<table>
<thead>
<tr>
<th>Best Policy</th>
<th>Training Episodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>297</td>
</tr>
<tr>
<td>Best</td>
<td>101</td>
</tr>
<tr>
<td>Worst</td>
<td>444</td>
</tr>
</tbody>
</table>

**Note**: Q-Learning with 300,000 episodes yields an average of 29 steps to reach the goal (this might be considered the optimal policy).
Sommaire

1 Introduction
   - Overview

2 Reinforcement Learning and Function Approximation
   - Reinforcement Learning
   - Function Approximation

3 Neural Fitted Q-Iteration Algorithm

4 Empirical Results

5 Conclusion
   - Related Work
   - Article Review

6 Questions
Conclusion

NFQ Algorithm

- Model-free Reinforcement Learning.
- Memory-based approach.
- Stores and re-uses transitions in order to do batch neural network learning.
- Batch learning reduces the negative effects of online learning that cause long learning times.
Least-Squares Policy Iteration

- Approximate Policy-Iteration algorithm.
- Linear approximation architecture.
- Batch processing for efficient use of training data.
Experience Replay


- Idea: Experience is costly, and only using it once is waste.

- Experience Replay: Re-use of experience that still matches the current policy since sampling from past policies will disturb learning.
Article Review

Pros

- Good application of techniques from different fields.
- Good description of benchmarking and types of learning tasks.

Cons

- Poor comparison with other algorithms.
- No details about algorithm convergence.
Sommaire

1 Introduction
   • Overview

2 Reinforcement Learning and Function Approximation
   • Reinforcement Learning
   • Function Approximation

3 Neural Fitted Q-Iteration Algorithm

4 Empirical Results

5 Conclusion
   • Related Work
   • Article Review

6 Questions