ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

# Naive Augmenting Q-Learning to Process Feature-Based Representations of States

Janis Zuters *

University of Latvia - Faculty of Computing
19 Raina blvd., LV-1586 Riga - Latvia

**Abstract.** Temporal difference algorithms perform well on discrete and small problems. This paper proposes a modification of the Q-learning algorithm towards natural ability to receive a feature list instead of an already identified state in the input. Complete observability is still assumed. The algorithm, Naive Augmenting Q-Learning, has been designed through building a hierarchical structure of input features (a kind of feature-state mapping) to avoid a direct state identification, thus potentially optimizing the required resources for storing and processing action values.

## 1 Introduction and Motivation

Reinforcement learning (RL) [1] is one of main paradigms of machine learning (ML), the objective of which is to compute – how to map situations to actions.

In its simplest form, a RL task can be described as a Markov decision process (MDP). A finite state completely observable MDP is reasonably well understood, and for small and discrete state spaces the available algorithms are considered to be efficient and applicable enough. [2]

Most research however is concerned with various extensions of the 'pure' MDPs, such as continuous state-spaces and continuous action-spaces, searching directly in the policy space, partially observable MDPs (POMDPs).

This paper is positioned slightly towards POMDPs and introduces an extension to the classical Q-learning [3] – Naive Augmenting Q-Learning, which directly deals with feature based state representations. The most trivial example here is a state representation by its row and column numbers (Fig. 1c). The proposed algorithm does not claim to be able to cope with partial observability. It simply eliminates some pre-processing of the observation otherwise required by a pure Q-learning algorithm.

Instead of a look-up table to store action-values, the proposed algorithm builds a list of lookup trees, where a state is represented by several nodes. 'Naive' stands for the principle of computing the actual action-values as weighed average of the values of the active tree nodes. Augmenting is performed in the situation of suspicion for perceptual aliasing, i.e., if an active node tries to learn in the direction significantly different from the average of the recent history.

* The research reported in this paper was supported by ERDF Project 2010/0202/2DP/2.1.1.2.0/10/APIA/VIAA/013 "Support for the international cooperation projects and other international cooperation activities in research and technology at the University of Latvia" and Artificial Intelligence Foundation Latvia.

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

## 2 Related Work

### 2.1 Q-Learning

One of the "benchmark" RL algorithms is Q-learning [3]. It stores the value information in the look-up table $Q$ for each possible action at each (non-terminal) state, and the policy can be directly derived from the table. The Q-learning algorithm is designed as a sequence of episodes, where an episode is a sequence of moves until a terminal state is reached. At each step, the algorithm adjusts the action-value for the current (non-terminal) state $s$ and the current action $a$ – toward the maximum action-value of the next state $s2$ plus the reward $r$:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \cdot \max\left(Q(s2,\cdot)\right) - Q(s,a)\right],$$

where $\alpha$ – learning rate; $\gamma$ – discount rate; $[r + \gamma \cdot \max(Q(s2,\cdot))]$ – instant learning destination.

Classically the state information is passed as the serial number of the state (Fig. 1a), however it can be implemented otherwise, e.g., by a binary sequence (Fig. 1b, 1c).

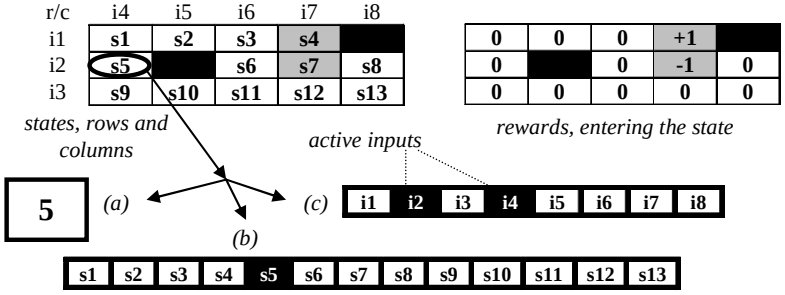


Fig. 1: A sample 3×5 gridworld (states 4 and 7 are terminal states) and a few possible kinds state representations (of the state #5): (a) state number, (b) state number in the form of a "simple" binary n-tuple (with exactly one 1), (c) feature-based representation in the form of a binary n-tuple (row and column).

### 2.2 Feature-Based Reinforcement Learning

Having classical RL algorithms on one's disposal, it's the responsibility of a designer to identify the state out of the bare observations. In general, this is the scope of POMDPs, where the history of observations also is to be considered. However, this is much more humble – about feature-based state representations, where a state passed to the algorithm is encoded as several numbers (instead of a single one). An example is to pass the global row and column numbers instead of the state number. The main problem with feature representations is that of curse of dimensionality.

1. One of the most obvious mechanisms to deal with features is to employ a **value-matrix in the feature space** [4]. The problem of directly using this approach is that the size of the value structure increases exponentially with the number of features. The linear architecture approximates a value function using a linear combination of feature values.

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

2. **Non-linear architectures** (e.g., with feed-forward neural networks to store weights) of the value function in the feature space [4] are preferred as number of features grows bigger.

3. The authors of [5] describe **policy gradient methods**, which approximate a stochastic policy directly and thus don't exploit value functions.

4. The authors of [6] introduce **Feature-SARSA algorithm**, which takes advantage of usefulness of local state features; "valuable" features are stored in a specific Example set and used to improve learning efficiency.

5. McCallum [7] presents algorithms that use **additional memory to prune and augment the state space** observed in the agent's input to cope with the problem of finding the right level of distinctions of observations ("too much distinctions" mean a lot of resources required by the agent to process the state space, "too few distinctions" can mean inability to obtain an adequate state representation to solve the task).

The proposed algorithm of Naive Augmented Q-learning exploits in part the ideas from approaches #1 and #5: a special value-structure is being built in the feature space to distinguish among states.

## 3 Naive Augmenting Q-Learning

If input from an environment is received in form of a binary n-tuple (Fig. 1c), a pre-processing module is needed to utilize standard Q-learning methods. Naive augmenting Q-learning has been designed to have the state distinction capability as part of it.

Instead of using a linear architecture of the value structure (approach #1, Section 2.2), the proposed method utilizes a list of trees (similar to decision trees). Unlike decision trees, (1) action-values are the ones stored in the nodes, (2) internal nodes (not only leaves) are also considered. A state is potentially represented by several nodes (from different trees) and action-values for the state are computed as a linear combination of action-values in these nodes. A deeper level of a node means more inputs associated with the node, thereby more specialization.

Naive augmenting Q-learning (Fig. 4) can be viewed as an extension to the classic Q-learning, and the main contribution of the method is automatic adaptation to the feature-based binary input. This is achieved by adding the following mechanisms:

(a) Collection of active nodes (those associated with active input channels) and computing action-values for the current state represented by the nodes (variables *qlist* and *qsum*). The impact of the node depends on the level (according to level impact rate $\lambda^{(imp)}$).

(b) The criterion and the mechanism to extend the value-structure (Q-tree) by adding more specialized nodes. For this purpose, the learning destination (see Section 2.1) statistics is cumulated, using statistics cumulation rate $\alpha^{(stat)}$, structure extension rate $\lambda^{(ext)}$ and destination difference threshold $\lambda^{(ext)}$. Extension is carried out only if the difference between the instant learning destination and the destination statistics (for a certain action-value) exceeds the threshold.

(c) The mechanism to prune tree structure by adding "fictive" nodes, which are not considered in computing action-values and are left without children.

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

Fig. 2 depicts a ready-made 2-level Q-tree structure for the 3×5 gridworld problem (Fig. 1). For example, states 1, 5 and 9 are represented as combinations of level 0 nodes, but for states 11 and 12 specialized nodes have been created.

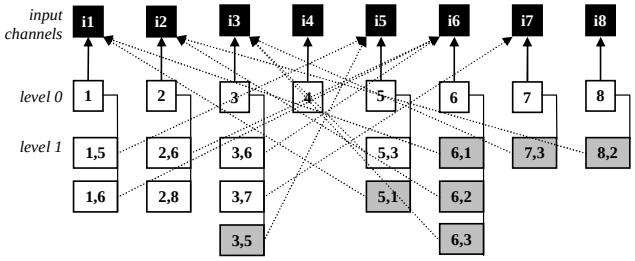


Fig. 2: A 2-level Q-tree structure built by the proposed algorithm (Fig. 4) for the 3×5 gridworld (Fig. 1). Fictive nodes are marked grey.

If there is only one feature in the input space (only one '1' for each input), the algorithm reduces to the standard Q-learning algorithm.

If there are more features, then the Q-tree structure can potentially grow exponentially with respect to the count of features, thus covering all the possible combinations of the values of the input features (this can be achieved with small values of parameters $\lambda^{(ext)}$ and $\varepsilon^{(ext)}$ for the proposed algorithm, see Fig. 4). All the input combinations represented in the tree structure guarantees the algorithm to work the same way as the standard Q-learning does, however much slower.

The sense to use the proposed algorithm is to exploit the potential "naiveness" of the input space, thus reducing the size of the tree structure but still being able to automatically deal with feature-based input.

## 4 Experimental Work

The goal of the experimental work was to show the ability of the Naive Augmenting Q-learning algorithm to cope with feature-based input, as well as its capability to reduce the size of the value tree structure.

The algorithm was tested on gridworlds (with rewards normalized to [-1..1]) up to 42 states and 4 features (see Fig. 1 and Fig. 3). In advance, the appropriate values of the meta-parameters were experimentally obtained (Fig. 4).

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -20 | -1 | -1 | 100 |
| -1 | | | | | -1 | | |
| -1 | -1 | -1 | -1 | | -1 | -1 | -1 |
| -1 | -1 | | -1 | | -1 | -1 | -1 |
| | | | -1 | | -5 | -5 | -1 |
| -1 | -1 | -1 | -1 | -5 | -8 | -1 | -1 |
| -1 | -1 | -1 | -1 | -10 | -1 | -1 | -1 |

Fig. 3: The 7×8 gridworld used in the experiments (the numbers indicate the rewards entering the states).

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

```
module naive_augmenting_qlearning (mdp,γ):
        acount – action count, here assumed to be fixed for all states
        I – input channels
        Q = {} # action-value tree structure
        for inp in the set of input channels I:  # create the level #0 structure of Q:
            q = node (Values=0.0, DestStat=0.0, Level=0, Input={inp}, IsFictive=False, Children={})
            append q to Q
            initialize_node(q,None,inp)
        repeat for each episode:
            Set internal state of mdp to some non-terminal state, observe input in I
            qlist = collect all deepest possible non-fictive nodes associated with "active" inputs
            qsum = cumulate_Qvalues (qlist)
            while current state of mdp is not terminal, for each time step t:
                choose action a derived from qsum
                apply a to mdp, observe input in I, observe reward r
                qlist0 = qlist  # input representation in time step (t-1)
                qlist = collect all deepest possible non-fictive nodes associated with "active" inputs
                qsum = cumulate_Qvalues (qlist)
                dest = r + γ * max(qsum)  # instant learning destination for q-learning
                for q in qlist0:
                    # (1/4). if structure extension condition is satisfied
                    if |q.DestinationStatistics[a]-dest| > ε^(ext)*(1 + q.Level * λ^(ext))):
                        from I choose an arbitrary channel inp, such that
                            (a) inp was active in time step (t-1)
                            (b) inp not in q.Input
                            (c) inp not in c.Input forall c in q.Children
                        newq = node (Values=0.0, DestStat=0.0, Level=q.Level+1,
                            Input=(q.Input union inp), IsFictive=False, Children={})
                        append newq to q.Children
                        # (2/4). Tree pruning #1: actual duplicate made fictive
                        if newq.Input == q0.Input for any q0 from qlist0:
                            newq.IsFictive = True
                        # (3/4). Tree pruning #2: potential duplicates created and made fictive
                        if (exist input channel inp2 and a node q2 from qlist0
                          such that (q2.Input union [inp2]) == newq.Input:
                            newq2 = node (Values=0.0, DestStat=0.0, Level=q2.Level+1,
                                Input=(q2.Input union inp2), IsFictive=True, Children={})
                            append newq2 to q2.Children
                    # (4/4). Adjusting action-values and destination statistics
                    q.Values[a] = q.Values[a] + α * (dest - q.Values[a])
                    q. DestStat[a] = q. DestStat[a] + α^(stat) * (dest - q. DestStat [a])

function cumulate_Qvalues (qlist):  # action-values for the current input
        values[1..acount] = 0.0
        if the current state of the mdp is non-terminal:
            total_impact = 0.0
            for q in qlist:
                impact = 1 + q.Level * λ^(imp)
                total_impact = total_impact + impact
                for a in [1..acount]: values[a] = values[a] + q.Values[a] * impact
            for a in [1..acount]: values[a] = values[a] ÷ total_impact
        return values
```

Fig. 4: Naive augmenting Q-learning algorithm. (The parameters and their fitted values. $\lambda^{(ext)}$=1.0: structure extension rate; $\varepsilon^{(ext)}$=0.5: destination difference threshold for structure extension; $\alpha$=0.01: learning rate; $\alpha^{(stat)}$=0.05: statistics cumulation rate; $\gamma$=0.9: discount rate; $\lambda^{(imp)}$=1.0: level impact rate)

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

With the fixed set of parameters, each task was run 10 times until the system achieved a suboptimal solution (one available in advance). Table 1 displays the results for each task: (a) approximate amount of iterations required; (b) the size of the generated Q-tree structure.

| Task ID (grid/features) | Amount of states | Iterations required | Q-tree size | |
|---|---|---|---|---|
| | | | actual | max possible[C] |
| 3×5 / 1[A] | 11 | 5,000 | 11 | 11 |
| 3×5 / 2[B] | 11 | 7,000 | 22..28 | 30 |
| 3×5 / 3 | 11 | 7,000 | 40..48 | ~400 |
| 3×5 / 4 | 11 | 20,000 | 55..73 | ~10,000 |
| 7×8 / 1[A] | 42 | 150,000 | 42 | 42 |
| 7×8 / 2 | 42 | 150,000 | 81..91 | 112 |

Table 1: Experimental results (A – similar to the classic Q-learning; B – as depicted in Fig. 2; C – an approximated estimate to cover all the combinations).

## 5 Conclusion

The algorithm presented here has been proposed to show an alternative approach how to deal with "specific" input – an extension to an RL-algorithm itself instead of using a separate preprocessing module. State distinction is performed jointly with the action-value learning. The value-structure is extended only if some states are not distinguishable in terms of tendency of changes to action-values. The state distinction mechanism has been designed to be task-dependent, so for "simpler" tasks the value-structure will automatically be smaller (i.e., more naive). The proposed algorithm extends the application area of the classical temporal difference algorithms.

## References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning. An introduction*, Cambridge, MA: MIT Press/A Bradford Book, 1998.

[2] M. Hutter. *Feature Markov Decision Processes*, Proceedings of the Second Conference on Artificial General Intelligence, AGI 2009, Arlington, Virginia, USA, March 6-9, 2009.

[3] C. J. C. H. Watkins, *Learning from Delayed Rewards*, PhD thesis, Cambridge University, Cambridge, England, 1989.

[4] J. N. Tsitsiklis and B. van Roy. *Feature-Based Methods for Large Scale Dynamic Programming*, Machine Learning, 22, 59-94, Kluwer Academic Publishers, Boston, 1996.

[5] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *In: Advances in Neural Information Processing Systems 12*, pp. 1057-1063, MIT Press, 2000

[6] Y.-P. Lin, X.-Y. Li. *Reinforcement learning based on local state feature learning and policy adjustment*, Information Sciences 154, 59–70, Elsevier Science Inc., 2003.

[7] A. K. McCallum, *Reinforcement Learning with Selective Perception and Hidden State*, PhD thesis, University of Rochester, Rochester, 1995.