Metric Learning for Invariant Feature Generation in Reinforcement Learning

Evan Kriminger, Austin J. Brockmeier, Luis G. Sanchez Giraldo, and Jose C. Principe
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611
http://www.cnel.ufl.edu/

Abstract

The success of reinforcement learning in real-world problems depends on careful selection of features to represent the state. Proper feature selection results in the increased ability to approximate the value function and in quicker learning, as only relevant information is emphasized. It is desirable to generate such features automatically, as this would otherwise be a trial and error process requiring a human expert. We propose a method to automatically map states to feature vectors, which are not only sufficiently descriptive of the environment, but are invariant to information not relevant to the agent's goal. The mapping is based on the principle that if the Q-values for two states under the same action are similar, then the states themselves should be similar. This similarity is realized in a space defined by a metric computed using an information-theoretic approach to metric learning. Our method works in conjunction with a Q-learning algorithm of choice and is suitable for large and continuous state spaces. We test our algorithm on a non-sequential decision task in which the state is an image. While the problem is in fact linear, our method greatly outperforms linear Q-learning.

Keywords: Reinforcement learning, automatic feature generation, metric learning

1 Introduction

For many practical problems, the number of states in the environment is too large for a reinforcement learning agent to visit all of them. This is not only true in the case of a continuous state space, where there are a continuum of states, but even in discrete spaces where the number of states grows exponentially with the size of our state vector. This is the well-known "curse of dimensionality" [1]. In such cases, function approximation techniques are necessary to predict the value function for states that have not yet been visited. In this approach, a mapping from a feature vector representing the state to the value function is learned, as the agent interacts with the environment. The choice of the features is therefore a crucial factor in the success or failure of reinforcement learning. In RL problems each state is associated with raw measured data about the environment. This data might be the pixels of an image for instance. The feature vector is mapped from this raw data and should effectively represent the state in such a way that is invariant to distracting elements. For instance, when a baseball is approaching, the action of catching it is the same whether it occurs in a baseball stadium or in a backyard. Successful feature selection enables us to recognize the similarity between the two situations, despite the fact that the background in each case is completely different. While recognizing such invariants drastically reduces the size of the state space, the feature vector must be sufficiently descriptive such that the agent is able to learn complicated behaviors. In the first implementation of TD-gammon [2], the state of a game of backgammon was represented by a vector consisting of 198 elements. Even if the features intuitively capture the relevant information of the problem, it may not be the best way to represent the state to the function approximator that is learning the value function. Trial and error may be required to find a representation that leads to success.

Choosing the state representation is left to the 'designer', but in real-world problems this is unsatisfactory. Generating the feature mapping automatically, is a necessary step in extending RL to more complicated problems and to domains where experts cannot hand pick features. The literature on automatic feature generation is relatively sparse, owing partially to the difficulty of the problem. Many of the existing methods base the learning of features on the Bellman error of the value function estimates. These methods are known as Bellman Error Basis Function (BEBF) [3]. Other methods require estimates of information about the Markov Decision Process, such as the transition or adjacency matrix [4], which are difficult to acquire in many large problems. Additionally, most automatic feature generation methods are set in the domain of policy evaluation. Feature generation for control methods, such as Q-learning, is an even less studied problem. One such method [5] learns a metric for comparing states, such that states with similar transitions under the same action are close with respect to this metric. In this approach, two states x_i and x_j, at times i and j, respectively, are similar if their single step increments are similar, i.e. if $x_{i+1} - x_i$ and $x_{j+1} - x_j$ are similar. A potential problem with this method is that the differences between consecutive states may not reliably indicate similarity of states with respect to the goal of the agent. If distracting features are dominant, such as in visual systems, then the features of interest are not properly accounted for in the state transitions.

In this paper, we present an automatic feature generator for RL that is also based on metric learning. However, in our approach, the similarity between states is established with respect to the goal of our learning system, rather than temporal differences in the state vectors. We first present our method and the information-theoretic metric learning algorithm it utilizes. We then demonstrate the ability of our method to generate invariant features in a non-sequential decision problem, where the state is a black and white image.

2 Metric Learning for Reinforcement Learning

To create a feature representation that is invariant to information not relevant to the current decision problem, we use the following principle:

Two states are similar if the value of these states are similar under the same action.

Intuitively, the value function (specifically the Q-value) reflects the performance goals for the given problem independently of the state. If $Q(x, a) = Q(y, a)$ for states x and y and action a, then these states are similar, in the same way that a stadium and a backyard are similar when moving your arm to catch a baseball.

Given this principle, the features we choose to represent the state should exist in neighborhoods defined by their Q-values, not based on the Euclidean distances between the raw vectors that are associated with each state. The prospect of learning an approximation of Q-values supports the principle we are using. If we choose a representation where states with similar Q-values are clustered, the mapping from state/action pairs to Q-values can be smooth. In a representation where neighboring states may have wildly different Q-values, it may be difficult or even impossible to learn such a mapping. Imagine our baseball player is attempting first to catch a fly ball and then to field a ground ball. While the field looks identical in both cases, save for the small speck that is the ball, to hold his glove high in the air would be successful in the first case, and a failure in the second.

Consider a reinforcement learning agent that has interacted with the environment for N time steps to acquire tuples, $(x_i, a_i, Q(x_i, a_i))$, consisting of the state, action, and Q-value at each time i. We wish to group states based on the action

that was performed in that state, as these are the states for which we can compare the Q-values. Let $\mathcal{X}_a := \{x_i | a_i = a\}$, be the set of states for action a. Each pair of states in $\mathcal{X}_a$, for each a, provides a constraint on a metric m, such that if

$$|Q(x_i, a) - Q(x_j, a)| < |Q(x_i, a) - Q(x_k, a)| \quad \text{for } x_i, x_j, x_k \in \mathcal{X}_a,$$

then $m(x_i, x_j) < m(x_i, x_k)$. This is equivalent to considering a constraint on the mapping, ϕ, from states to feature vectors. In the latter case, $m(\phi(x_i), \phi(x_j)) < m(\phi(x_i), \phi(x_k))$, where m is simply the Euclidean distance. Learning the metric m or the mapping ϕ, that satisfies (or best satisfies) these constraints, provides the desired feature representation for our states.

Luckily, learning a metric is a well-investigated problem, as finding a 'good' similarity measure is common in pattern recognition and machine learning. The approach we consider is similar to metric-learning for classification [6, 7]. The concept of metric-learning is to parametrize a distance function such that states with similar Q-values are deemed close and states with very different Q-values are considered far apart. Most metric learning algorithms require hard information representing class membership. In our problem, the information in the form of the Q-values is soft, and we must find a metric on states which parallels the distances between Q-values. For this reason, we learn the metric via an information-theoretic optimization problem that optimizes the information between the state representation and the Q-values [8].

2.1 Distances and Similarity

The similarity between samples x and x' on the ith dimensions is $\kappa(x^{(i)}, x^{(i)\prime}) = \exp(-\theta d(x^{(i)}, x^{(i)\prime})^2)$, where $d(\cdot, \cdot)$ is the Euclidean distance and θ is a kernel size parameter. Changing the kernel size adjusts how close the samples must be in order to be considered similar.

In terms of a group of samples, the pairwise distance matrix for the ith dimension is denoted D_i where $(D_i)_{j,k} = d(x_j^{(i)}, x_k^{(i)})$ for $j, k \in \{1, \ldots, n\}$. Likewise, the corresponding kernel matrix with kernel size parameter θ is $K_i = \exp(-\theta D_i^2)$.

A similar quantity can be defined for the similarity between Q-values and the corresponding kernel-matrix is denoted L.

2.2 Entropy

Rényi's α-order entropy is an information measure for probability distributions. Recent work [8], has shown how a similar quantity can be defined in terms of the eigenvalues of a positive definite kernel matrix. Using the formulation of Rényi's entropy on the eigenvalues yields a matrix-based analog to entropy [8]:

$$S_\alpha(B) = \frac{1}{1-\alpha} \log [\operatorname{tr}(B^\alpha)]. \quad (1)$$
$$\text{where } \operatorname{tr}(B) = 1$$

Unlike standard approaches that require knowing the distributions of the data or estimating its density using methods such as Parzen windows, this approach directly estimates an entropy quantity without ever requiring an explicit density function. Another key benefit is optimization can be done in terms of the kernel and distance matrix using matrix calculus.

From this a measure of conditional entropy $S_\alpha(B|C) = S_\alpha(B, C) - S_\alpha(C)$ can be applied, and this form of conditional entropy was used in previous work for learning a Mahalanobis distance [8].

2.3 Tensor Product Kernel

The tensor product between kernel functions is a joint measure for multivariate data formed as the product of kernels for each dimension of the input.

Considering the Gaussian kernel, the tensor product kernel corresponds to using a non-negative combination of the different Euclidean metrics as the argument to the kernel function. (A series of Hadamard products is denoted $\prod_{i=1}^N A_i = A_1 \circ \cdots \circ A_N$, and entry-wise power as $A_i^{\circ r}$)

$$K_\theta = \prod_i K_i^{\circ \theta_i} = \prod_i \exp(-\theta_i D_i^2) = \exp(-\sum_i \theta_i D_i^2)$$

Adjusting the parameters of $\theta \geq 0$ changes the kernel size of each component kernel, and this is equivalent to scaling each dimension of the input so as to form a new metric $d_\theta^2(x, x') = \sum_i \theta_i d^2(x^{(i)}, x^{(i)\prime})$.

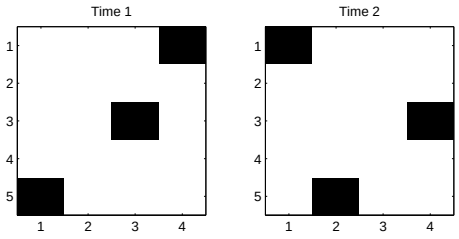

Figure 1: Example images for two consecutive time steps of the game.

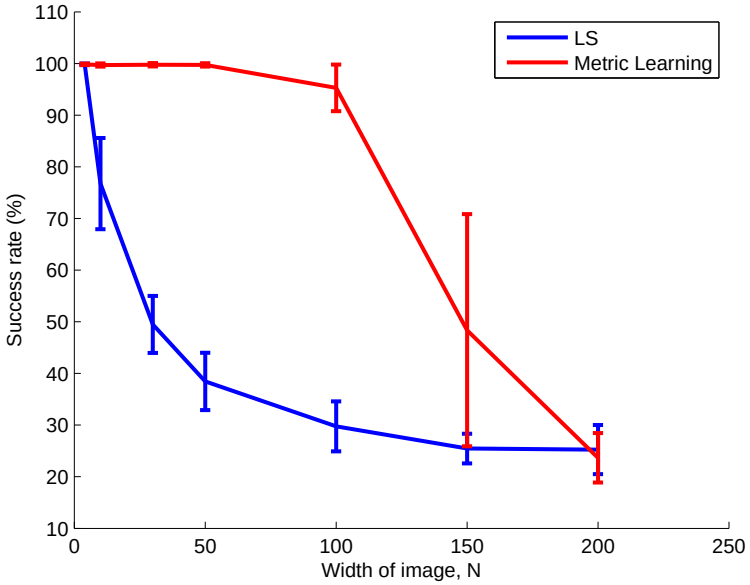

Figure 2: Success rate for least squares and metric learning methods. The error bars represent the interquartile range.

In order to learn this metric we consider the following information-theoretic optimization problem:

$$\begin{aligned} &\underset{\theta \geq 0}{\text{minimize}} \;\; S_\alpha(L, K_\theta) \\ &\text{subject to} \quad S_\alpha(K_\theta) = \eta \end{aligned} \tag{2}$$

This problem attempts to maximize the joint entropy of the state representation and the Q-values while constraining the marginal entropy. Allowing a large value of η allows the entropy of the data sample to increase, which is usually important for regularizing statistical estimates. However, if η is too high, a trivial solution is found where every sample is deemed different. A good choice of η is Rényi's entropy of the Q-values.

The relationship with the conditional entropy ($S_\alpha(L|K) = S_\alpha(L, K) - S_\alpha(K)$) can be seen by transforming the constraint into the Lagrangian formulation

$$\underset{\theta \geq 0}{\text{minimize}} \;\; S_\alpha(L, K_\theta) - \lambda \left(S_\alpha(K) - \eta \right) \tag{3}$$

with the Lagrange multiplier λ. However, the constraint on the non-negativity of the coefficients remains. As a simple approach, an unconstrained optimization is formed by re-parametrizing the function in terms of w where $\theta_i = 10^{w_i}$. By solving this optimization problem, a new metric can be learned as a linear combination of metrics.

3 Experiment

To test the ability of our generated features to represent the state, we consider a non-sequential decision task, in which the the state is a black and white image. By choosing a non-sequential task, the value of each action is known perfectly. This was done to isolate the problem of feature generation from value estimation. This problem consists of a image that is 5 pixels in height and N pixels wide. At most one pixel of each column is black, while the rest are white. At each time step, the black pixels shift one to the right, and with probability 0.5, a randomly selected row of the first column is set to black. When a pixel in the rightmost column is black, our RL agent must select the row in which it exists; thus there are 5 actions. If the agent selects the black pixel, a reward of +1 is received. If it misses the black pixel, a reward of -1 is received. If all pixels in the rightmost column are white, the reward is 0 for all actions. The reward in this case is the Q-value, as this is a non-sequential problem. See Figure 1 for an illustration of the game.

The state in this game is the image itself, which takes the form of a vector with $5 \cdot N$ binary elements, representing the values of each pixel. This is actually a problem for which a linear function of the raw state can perfectly approximate the value function. Consider the case where $N = 1$. Then as an example, $Q(x, a = 2) = \theta_2' x$, if $\theta_2 = [-1, 1, -1, -1, -1]'$. If $N > 1$, then all elements of θ_a not corresponding to the rightmost column are 0.

The standard procedure in RL would be to learn θ_a for each $a \in \{1, ..., 5\}$, which provides an estimate of the Q-values. While the Q-values are in fact a linear function of the state, there are many distracting elements in the image, namely every black pixel not in the rightmost column, which impede the learning of θ. We compare our method to this linear function approximation method, where the coefficients are with least squares.

The first 200 time steps serve as a training period, in which random actions are selected and stored with the associated states and Q-values (rewards). At that point, we learn the feature mapping based on the Q-values. For the next 300 time steps, the raw image vector is mapped to the feature vector, and compared to all previous feature vectors. We compare the current image to the sets of previous images grouped by action, and select the action for which the Q-value of the nearest neighbor image is greatest. For the product kernel, the goal is to learn a weighted combination of Euclidean distances, resulting in a projection that only weights the pixels of interest in the right column. The dimensionality is effectively reduced to 5.

For the linear function approximation, at every time step after 200, the coefficients are learned by finding the least squares solution to $X_a\theta_a = Q_a$, for each action, where X_a is the matrix of the past states in which action a was used and Q_a is the vector of associated rewards. For each of the 300 testing images, the Q-value is estimated for each action, and the action with highest Q is selected.

We evaluate both methods based on the rate of correct choices, relative to all instances in which a black pixel is in the right column, out of the 300 testing images. The results are averaged over 20 Monte Carlo trials. We run the test for 7 values of N, and the results are seen in Figure 2. Our metric learning approach remains nearly perfect until $N = 100$, while the least squares approach declines with increasing state space size. By $N = 200$, which corresponds to a 1000 element state vector, with 6^{200} possible states, the performance of both methods has declined to only slightly better than random chance. The linear Q-learning approach fails for large state spaces because not enough data is available for it to capture the elements of the image which correlate with successful runs. This occurs because the method attempts to linearly combine all pixels in the image into the observed Q-values, which becomes an increasingly more difficult task as the image grows. Our method simply observes Q-values, and figures out a way to compare the images that produced similar ones.

4 Conclusion

The Q-values serve as a valuable indicator of the inherent similarity between states. Using a metric learning approach we can capture this similarity in a feature mapping. The resultant features are invariant to the distracting elements of the raw state, and learning proceeds much faster, as RL agents are able to generalize to previously unseen states when the essential task is the same. Our method is shown to perform extremely well in a problem where Q-values are given following each decision. Even for a space with 6^{100} states, only 200 time steps of interaction are required to achieve a 95% success rate, while linear Q-learning operating on the raw image is only 30% successful, despite the fact that the optimal solution is linear. Future work will apply our automatic feature generator alongside Q-value approximation techniques.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[2] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[3] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. Littman, "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning," in *25th Int. Conf on Machine Learning*, 2008.

[4] M. Petrik, "An analysis of laplacian methods for value function approximation in mdps," in *Proc. of the 20th Int. Joint Conf. on Artifical intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, pp. 2574–2579.

[5] M. E. Taylor, B. Kulis, and F. Sha, "Metric learning for reinforcement learning agents," in *Int. Conf. on Autonomous Agents and Multiagent Systems*, 2011.

[6] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance metric learning, with application to clustering with side-information," in *Advances in Neural Information Processing Systems 15*, vol. 15, 2002, pp. 505–512.

[7] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, pp. 207–244, 2009.

[8] L. G. Sanchez Giraldo and J. C. Principe, "Information theoretic learning with infinitely divisible kernels," in *International Conference on Learning Representations*, May 2013.