Section overview

Function Approximation

Approximating value functions

We want to scale up the model-free methods for prediction and control we learned previously.

- Situation: Up to now we have represented value function $V(s)$ or action-value function $Q(s, a)$ by a lookup table
- Complication: Large MDPs have many states (Curse of Dimensionality)
  Backgammon: 10^{20} states, Go: 10^{170}, Flying a helicopter: many continuous state spaces
  1. There are too many states and/or actions to store in memory (even with sparse storage approaches) as eventually we will have to fill them in
  2. Our agent will take a very, very long time learning the value of each state individually
- Solution for large MDPs: Estimate value function with function approximation

Approximating functions

Generalise from seen states to unseen states Update parameter w using MC or TD learning

- Estimate value function with function approximation

$$\begin{aligned} V^{\pi}(s) &\approx \hat{V}(s, \mathbf{w}) \\ Q^{\pi}(s, a) &\approx \hat{Q}(s, a, \mathbf{w}) \end{aligned}$$

- Generalise from seen states to unseen states (fundamental ability of any good machine learning system)
- Update function approximation parameter $\mathbf{w}$ using MC or TD learning

There are many function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases ...

1. We limit ourselves to differentiable functions, which simplifies learning
2. We require a training method that is suitable for non-stationary, non-identical-independent (iid) distributed data

Stochastic Gradient Descent

Goal: find parameter vector $\mathbf{w}$ minimising mean-squared error between approximate value function $\hat{V}(s, w)$ and true value function $V^{\pi}(s)$.

$$J(\mathbf{w}) = \mathbb{E}\left[\left(V^{\pi}(s) - \hat{V}(s, \mathbf{w})\right)^2\right] \tag{48}$$

Gradient decent finds a local minimum by sliding down the gradient

$$\begin{aligned} \Delta\mathbf{w} &= -\frac{1}{2}\alpha\nabla_{\mathbf{w}}J(\mathbf{w}) && (49)\\ &= \alpha\mathbb{E}\left[\left(V^{\pi}(s) - \hat{V}(s, \mathbf{w})\right)\nabla_{\mathbf{w}}\hat{V}(s, \mathbf{w})\right] && (50) \end{aligned}$$

The learning factor $-\frac{1}{2}\alpha$, where the learning rate α controls the step size. The term $\frac{1}{2}$ is chosen so that is cancels out with the derivative of the squared error. The term is negative as we want to perform gradient descent (the gradient points upwards otherwise).
Stochastic gradient descent samples the gradient and the average update is equal to the full gradient update:

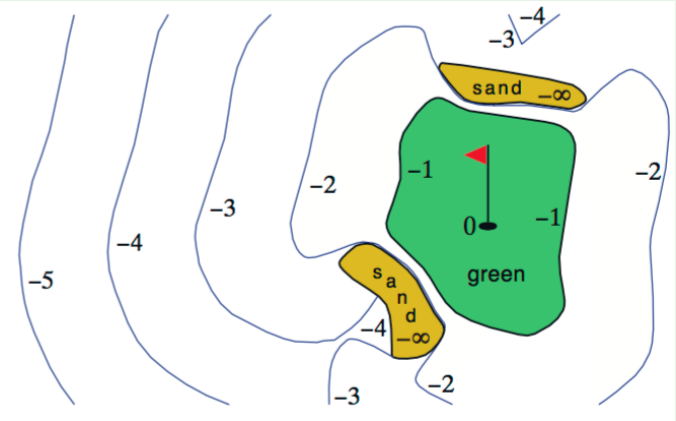
$$\Delta w = \alpha\left(V^{\pi}(s) - \hat{V}(s, \mathbf{w})\right)\nabla_{\mathbf{w}}\hat{V}(s, \mathbf{w}) \tag{51}$$

States from features vectors

Represent state by a feature vector $\mathbf{x}(s) = (x_1(s) \dots x_n(s))^\top$. The feature vector may be different from a precise state definition, e.g. because in many real world problems we do not have a strong definition of states.

Example

- Distance of robot from multiple landmarks

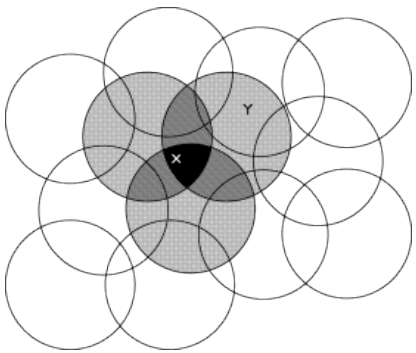


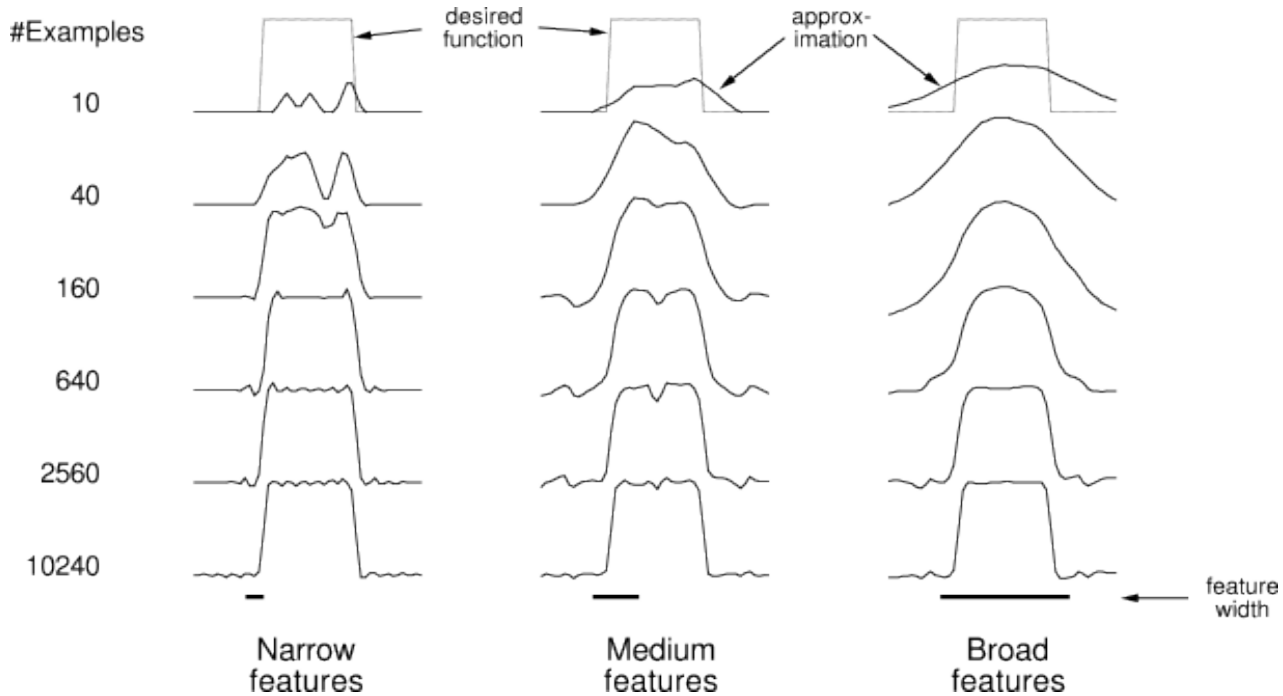
- Piece and pawn configurations in board game
- Cockpit camera

Note: All our previous derivations for RLs can be seen as a special case where the feature vector is the state vector and the function approximation would be the actual value.

Features (crafted not learned): Coarse coding

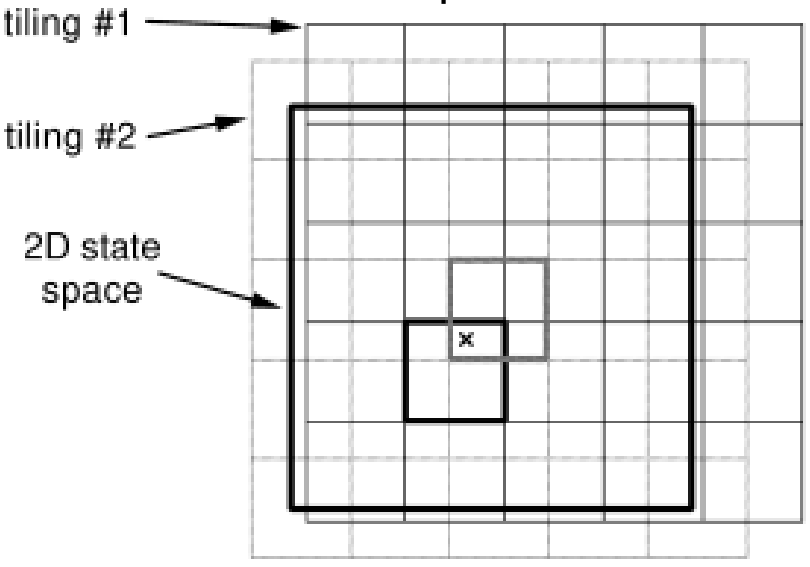
If the state is inside a circle, then the corresponding feature has the value 1 otherwise the feature is 0, a binary feature. Given a state, which binary features are present indicate within which circles the state lies, and thus coarsely code for its location. Representing a state with features that overlap in this way is known as coarse coding.





Features 2: Tile Coding

Tile coding is a form of coarse coding suited for computers and efficient on-line learning. In tile coding the receptive fields of the features are grouped into exhaustive partitions of the input space. Each such partition is called a tiling, and each element of the partition is called a tile, the overall number of features that are present at one time is strictly controlled and independent of the input state.



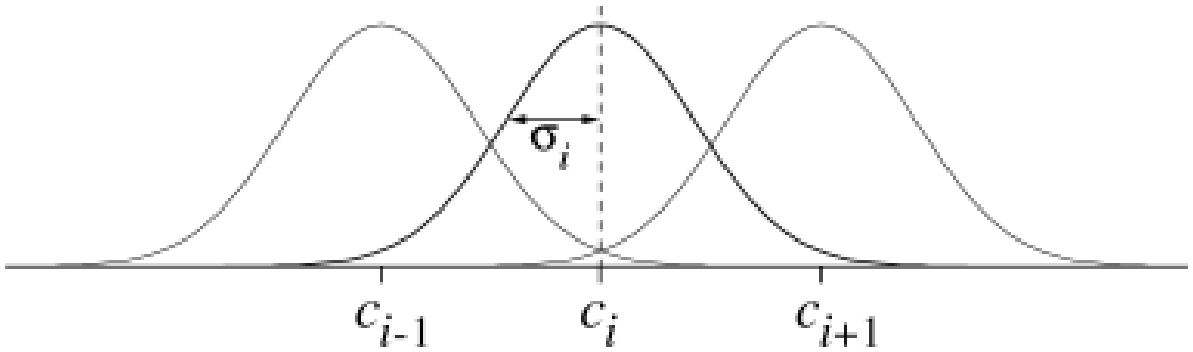
Shape of tiles ⇒ Generalization

#Tilings ⇒ Resolution of final approximation

Another trick for reducing memory requirements is hashing - a consistent pseudo-random collapsing of a large tiling into a much smaller set of tiles.

Features (crafted not learned): Radial Basis Functions I

Radial basis functions (RBFs) are the natural generalization of coarse coding to continuous-valued features. Rather than each feature being either 0 or 1, it is in $[0, 1]$, reflecting various degrees to which the feature is present.

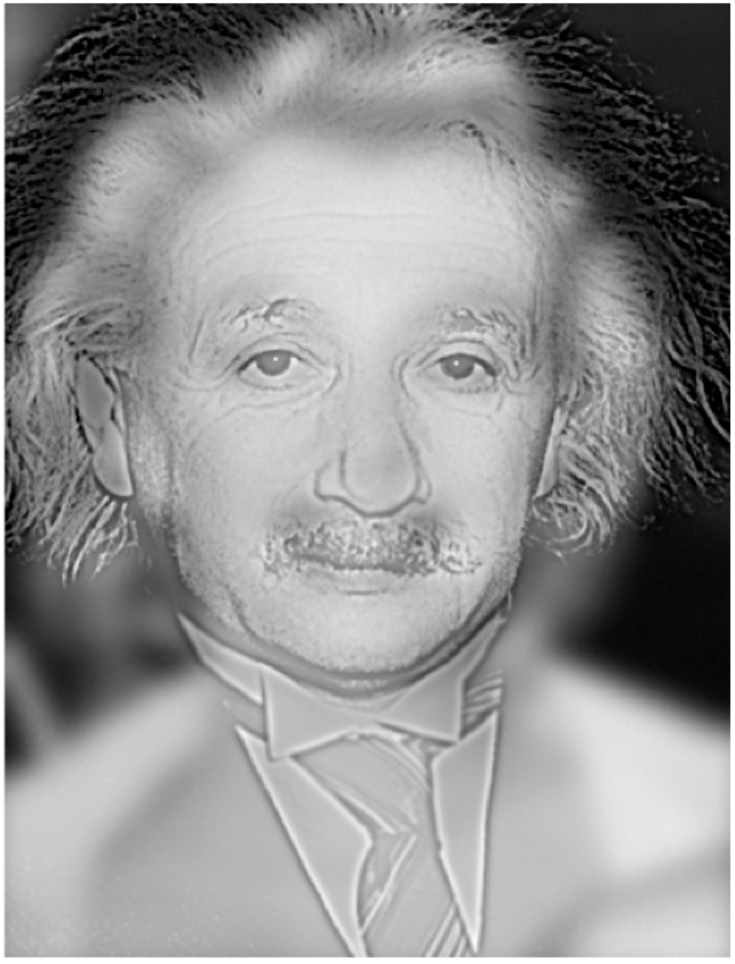


Features (crafted not learned): Radial Basis Functions II

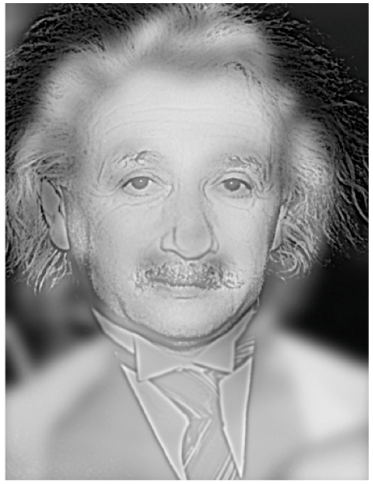
$$\phi_s(i) = \exp\left(-\frac{|s - c_i|^2}{2\sigma_i^2}\right) \tag{52}$$

where each $\phi_s(i)$ is the i-th basis function that maps a continous state space variable s. Each RBF is centered at location c_i and has width σ_i.

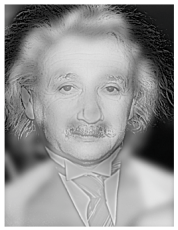
Example: Coarse coding with RBFs in your Brain



Example: Coarse Coding with RBFs in your Brain



Example: Coarse Coding with RBFs in your Brain



Features (crafted not learned): Learning representations

Mapping state to value as a function approximation problem.
There are many function approximations, e.g.

- Remove inductive bias (of the person designing the RL algorithm)
- Identify complex patterns that are not detectable to "see" as human developer
- Identify weak patterns in swaths of data that human cannot detect

Linear Value Function Approximation I

- Represent value function by a linear combination of features

$$\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^\top \mathbf{w} = \sum_{j=1}^{n} x_j(s) w_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = \mathbb{E}\left[(V^\pi(s) - \mathbf{x}(s)^\top \mathbf{w})^2\right]$$

- Stochastic gradient descent converges on global optimum

Linear Value Function Approximation II

- Update rule is particularly simple

$$\begin{aligned} \nabla_{\mathbf{w}} \hat{V}(s, \mathbf{w}) &= x(s) \\ \Delta \mathbf{w} &= \alpha(V^{\pi}(s) - \hat{V}(s, \mathbf{w}))\mathbf{x}(s) \end{aligned}$$

Definition

Update = learning step size $\times$ prediction error $\times$ feature value

Monte-Carlo with Value Function Approximation

- Return R_t is an unbiased, noisy sample of true value $V^\pi(s_t)$
- We apply supervised learning to "training data" of state return trace:

$$(s_1, r_1), (s_2, r_2), \ldots, (s_T, r_T) \tag{53}$$

- For example, using linear Monte-Carlo policy evaluation

$$\begin{aligned} \Delta \mathbf{w} &= \alpha(R_t - \hat{V}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w}) && (54) \\ &= \alpha(R_t - \hat{V}(s, \mathbf{w}))\mathbf{x}(s_t) && (55) \end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum, even when using non-linear value function approximation (provable)

TD Learning with Value Function Approximation I

- The TD-target $R_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w})$ is a biased (single) sample of the true value $V^{\pi}(s_t)$
- We still perform supervised learning on "digested" training data:

$$(s_1, r_2 + \gamma \hat{V}(s_2, \mathbf{w})), (s_2, r_3 + \gamma \hat{V}(s_3, \mathbf{w})), \ldots, (s_T, r_T) \tag{56}$$

- For example, using linear TD

$$\begin{aligned} \Delta \mathbf{w} &= \alpha(r + \gamma \hat{V}(s', \mathbf{w}) - \hat{V}(s, \mathbf{w})) \nabla_{\mathbf{w}} \hat{V}(s_t, \mathbf{w}) && (57) \\ &= \alpha(r + \gamma \hat{V}(s', \mathbf{w}) - \hat{V}(s, \mathbf{w})) \mathbf{x}(s_t) && (58) \end{aligned}$$

- Linear TD converges "close" to the global optimum (provable). This does not extend to non-linear TD (see Sutton & Barto, 2018).

TD Learning with Value Function Approximation I

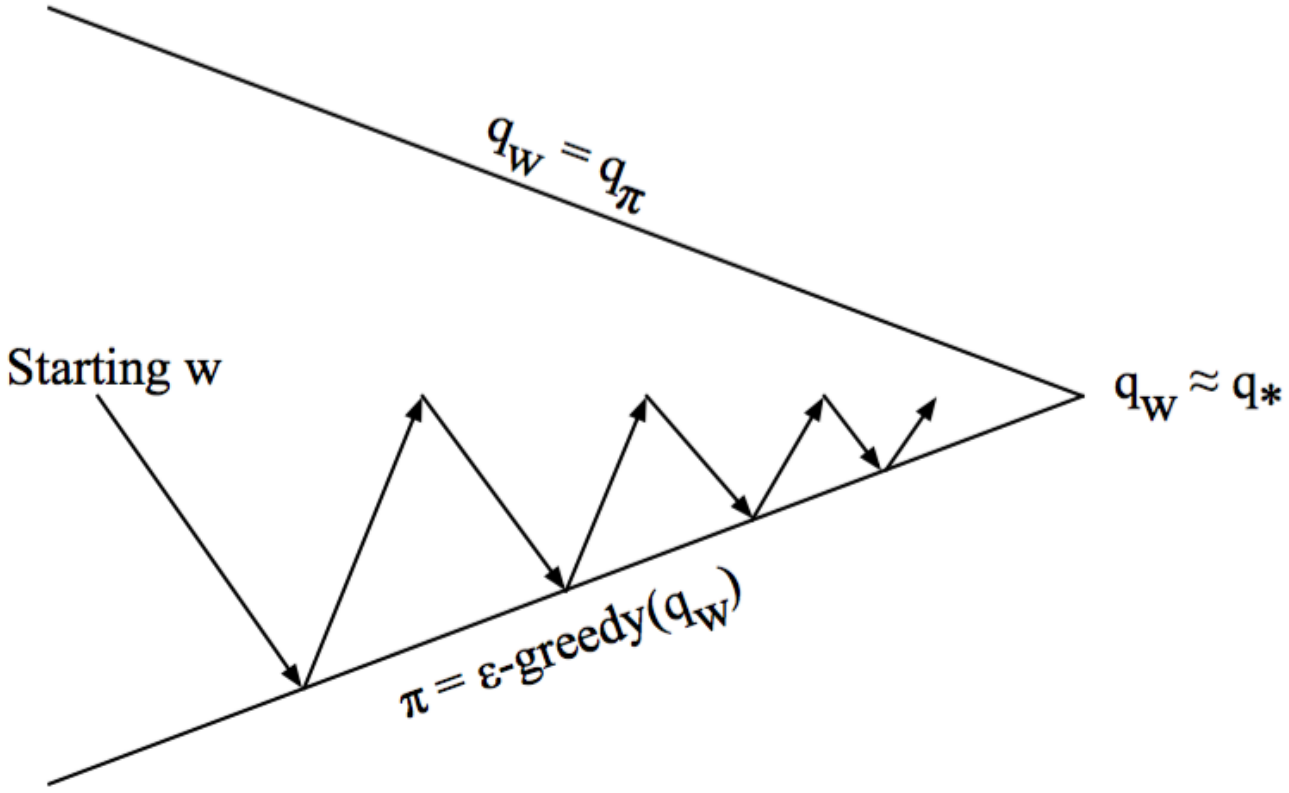
A simple algorithm implementing value function estimation using linear function approximation is the following.

1: Initialize w $= 0$, $k = 1$
2: **loop**
3: Sample tuple (s_k, a_k, r_k, s_{k+1}) given π
4: Update weights:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha(r + \gamma \boldsymbol{x}(s')^T \boldsymbol{w} - \boldsymbol{x}(s)^T \boldsymbol{w})\boldsymbol{x}(s)$$

5: $k = k + 1$
6: **end loop**

From evaluation to control: FA in GPI



1. Policy evaluation: Approximate policy evaluation $\hat{Q}(s, a, \mathbf{e}) \approx Q^p i(s, a)$
2. Policy improvement: ϵ-greedy policy improvement