Lecture 8: On-policy Prediction with Function Approximation

Tianpei Xie

Aug 10th., 2022

Contents

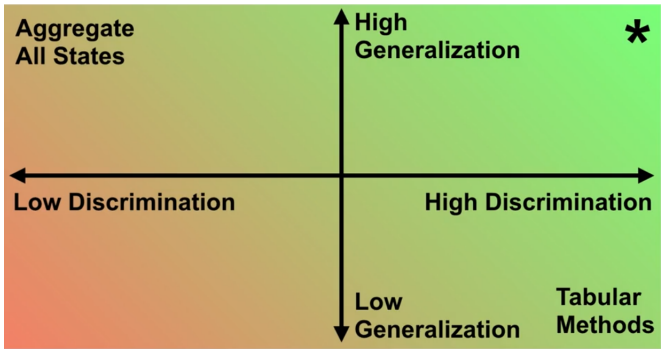

Figure 1: Generalization and discrimination are two independent dimensions to evaluate a value estimation strategy.

1 Introduction

In this chapter, we begin our study of function approximation in reinforcement learning by considering its use in estimating the state-value function from on-policy data, that is, in approximating v_π from experience generated using a known policy π.

There are two groups of RL methods based on how the value function are represented and stored

- ***Tabular*** *methods*: The value function is stored as a table, i.e. for each s, there is a value $v(s)$ (or $q(s,a)$ for each pair (s,a)) in the table. The tabular representation has good **discriminization** but no **generalization**. In other words, value for each state is stored and updated **independently**, so the dimension of the representation for the tabular value function is $|\mathcal{S}|$ for v and $|\mathcal{S}| \times |\mathcal{A}|$ for q. Also it cannot be applied to applications where the total set of states $|\mathcal{S}|$ is large.
- ***Function approximation*** *methods*: The value function is ***parameterized***. The function approximation allows us to represent the value function via its low dimensional paramater $d \ll |\mathcal{S}|$, which is more *efficient* in representation and storage, esp. when state-space $\mathcal{S}$ is very large. On the other hand, the value for *all* states are affected by the change of parameter. It will reduce the **discriminization** in order to gain **generalization**. As its name suggests, the value function estimated via function approximation is not exact but approximation.

In discussion above, we listed out two important concepts for value estimation:

- ***Generalization***, which means updating value of one state would affect the value of other states. That is, the change *generalizes* from that state to affect the values of many other states. Aggregate all states would easily achieve high generalization but with no discrimination.
- ***Discriminization***, which means the ability to make the value of two states different. If value of each states are represented independently, such as tabular methods, the discrimination is maximized but there is no generalization.

A good value estimation method need to have good generalization and good discrimination.

2 Value-function approximation as supervised learning

Value-function approximation problem can be treated as a **supervised learning problem**. Note that in supervised learning, we are given a set of covariates-target pairs $\{(s_t, G_t)\}_{t=1,\dots,T}$, the problem is to estimate the function $v : \mathcal{S} \to \mathbb{R}$ so that $\|G_t - v(s_t)\|_2^2$ is small in expectation.

In RL, the ***target*** is defined as the *expected returns* or its **approximation**. The **covariates** are states or state-action pairs. Given the functional form of value function $v(s, \boldsymbol{w})$, we listed out the target definition of several important algorithms

- **Expected returns (Ground Truth)**:

$$v_\pi(s_t) := \mathbb{E}_\pi \left[G_t | S_t = s_t \right].$$

  The ground truth target is the expected returns given state s_t and policy π and dynamic $p(s', r|s, a)$.

- **Boostrapped expected returns (Dynamic Programming)**:

$$v_\pi(s, \boldsymbol{w}_t) := \mathbb{E}_\pi \left[R_{t+1} + \gamma \, v_\pi(S_{t+1}, \boldsymbol{w}_t) | S_t = s \right].$$

  The target depends on expected estimate of next rewards, the expected estimate of value for all next states.

- **Sample returns (Monte Carlo methods)**: G_t, obtained by sampling the entire episode starting at S_t and averaging all rewards in the process.

- **Boostrapped sample returns (Temporal difference methods)**:

$$\hat{G}_t := R_{t+1} + \gamma v_\pi(S_{t+1}, \boldsymbol{w}_t),$$

  i.e. the target depends on both the next rewards and the value estimate of next state.

Using supervised learning for prediction/policy evaluation task has several challenges:

- Conventional supervised learning assumes the samples follow some **static distribution** and are **independent** identically distributed (i.i.d.). In general, this is not satisified in RL setting. Note that all sample pair (S_t, G_t) are generated by some Markove Decision Process, which are natrually **correlated**.

- The underlying **distribution of sample** in conventional supervised learning is fixed, **stationary**, unknown. But in RL, there is no assumption on the distribution for each state-action pair within the MDP. In fact, in typical use case of RL, the environment is **dynamic**, thus the underlying distribution of states is changing.

- The **target function** of conventional supervised learning is fixed. In RL such as TD methods or DP methods, the target function is determined by the value estimation of successor states, which is **non-stationary** and changing according to the change of value estimation.

- RL methods need to do **online learning**. In reinforcement learning, it is important that learning be able to occur online, while the agent interacts with its environment or with a model of its environment. To do this requires methods that are able to learn efficiently from *incrementally acquired* data.

2.1 The objective of prediction

The objective function for value approximation uses the **Mean Squared Value Error**, denoted $\overline{VE}$:

$$\overline{VE}(\boldsymbol{w}) = \sum_{s \in \mathcal{S}} \mu(s) \left\| v_\pi(s) - \hat{v}(s, \boldsymbol{w}) \right\|_2^2$$

where $v_\pi(s)$ is the target value function, and $\hat{v}(s, \boldsymbol{w})$ is the learned approximated value function. The ***state distribution*** $\{\mu(s)\} \subset \Delta$ helps to balance the generalization and discrimination. By assumption we have far more states than weights, so making one states estimate more accurate invariably means making others less accurate. We are obligated then to say which states we **care most** about. Often $\mu(s)$ is chosen to be the **fraction of time** spent in s. Under on-policy training this is called the **on-policy distribution**; we focus entirely on this case in this chapter. In continuing tasks, the on-policy distribution is the ***stationary distribution*** under π. In Markov chain theory, $\mu(s)$ is the **stationary distribution** of Markov Chain, or we called it **steady-state distribution**.

Note that in episodic task, the on-policy distribution $\boldsymbol{\mu}$ can be selected by equations:

$$\eta(s') = h(s') + \sum_s \eta(s) \sum_a \pi(a|s) p(s'|s, a), \quad \text{for all } s' \in \mathcal{S}$$
$$\mu(s') = \frac{\eta(s')}{\sum_s \eta(s)}$$

where $h(s)$ denote the probability that an episode **begins in each state** s, and $\eta(s)$ denote the number of time steps spent, on average, in state s in a single episode. This formular indicates that the on-policy distribution $\boldsymbol{\mu}$ is a function of policy $\pi(a|s)$. In this chapter, the policy π is fixed, therefore $\boldsymbol{\mu}$ is constant state weight. In future chapters for policy gradient, when the policy $\pi(a|s)$ is changing, this distribution will also change.

Unlike supervised learning, the **goal** of RL is to learn the **optimal policy** instead of optimal value function. For many nonlinear functions, finding the ideal **global optimal** $\boldsymbol{w}_*$ so that $\overline{VE}(\boldsymbol{w}_*) \leq \overline{VE}(\boldsymbol{w})$ for all $\boldsymbol{w}$ is not easy. Instead, the function approximation algorithm would converge to some **local minimal**.

3 Stochastic gradient and semi-gradient methods

To solve the minimization problem on $\overline{VE}(\boldsymbol{w})$, we simply apply the **stochastic gradient descent (SGD)** algorithm:

$$\begin{aligned}
\boldsymbol{w}_{t+1} &\leftarrow \boldsymbol{w}_t - \frac{1}{2}\alpha \nabla_{\boldsymbol{w}} \overline{VE}(\boldsymbol{w}_t) \\
&= \boldsymbol{w}_t + \alpha(v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)) \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t) \\
&= \boldsymbol{w}_t + \alpha \delta_t \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t)
\end{aligned}$$

The term $\delta_t := (v_\pi(s) - \hat{v}(s, \boldsymbol{w}_t))$ defines the **error** i.e. the difference between target value and the approximation function and the second term $\nabla_{\boldsymbol{w}} v(s, \boldsymbol{w}_t)$ finds the gradient of the value function. The SGD algorithm is preferred since in many cases, we do not seek or expect to find a value

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

Figure 2: Gradient Monte Carlo uses the SGD algorithm with Monte Carlo return estimate as target

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S is terminal

Figure 3: Semi-gradient TD(0) algorithm

function that has zero error for all states, but only an approximation that **balances the errors in different states**. SGD guaranteed to converge to local minimal given that α_t decreased over time and the standard stochastic approximation conditions are satisfied.

Note that the target output may not be the ground truth value of $v_\pi(S_t)$. We denote the output at S_t as $\hat{G}_t$. $\hat{G}_t$ is just an approximation of true value $v_\pi(S_t)$. This time the SGD becomes

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha\left[\hat{G}_t - \hat{v}(S_t, \boldsymbol{w}_t)\right]\nabla_{\boldsymbol{w}}\hat{v}(S_t, \boldsymbol{w}_t) \tag{1}$$

If $\hat{G}$ is an ***unbiased* estimate**, that is, if $\mathbb{E}\left[\hat{G}|S_t\right] = v_\pi(S_t)$, for each t, then $\boldsymbol{w}_t$ is guaranteed to converge to a local optimum under the usual stochastic approximation conditions for decreasing α. This is true for Monte Carlo methods since the sample return is equal to expected return asymptotically.

Figure 2 describes the ***Gradient Monte Carlo algorithm***, which uses the SGD algorithm in combination with sample returns obtained via Monte Carlo sampling.

On the other hand, one *does not obtain the same guarantees* if a **bootstrapping** estimate of $v_\pi(S_t)$ is used as the target $\hat{G}_t$ as discussed in DP and TD algorithms. Bootstrapping methods are biased

since they all depends on the estimation of value for successor states, which depends on the current value of the weight vector $\boldsymbol{w}_t$. Bootstrapping methods are **not** in fact instances of true gradient descent. They take into account the effect of changing the weight vector $\boldsymbol{w}_t$ on the estimate, but ignore its effect on the target. They include only a part of the gradient and, accordingly, we call them ***semi-gradient methods***.

The **semi-gradient methods** with **TD(0) updates**:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha \left[R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t)\right] \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t) \tag{2}$$

Here $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t)$ is the **TD error**. Figure 3 describes the semi-gradient TD(0) methods for value approximation. Although semi-gradient (bootstrapping) methods do not converge as robustly as gradient methods, they do converge reliably in important cases such as the linear case discussed in the next section. Moreover, semi-gradient methods typically enable significantly faster learning since they can make update at each of each step instead of at the end of whole episode.

Finally, we discuss a technique called ***state aggregation***. In state aggregation, states are grouped together, with one estimated value (one component of the weight vector $\boldsymbol{w}$) for each group. The value of a state is estimated as its groups component, and when the state is updated, that component alone is updated. This is a **binning method** to convert a large number of states into a lower dimensional weight vector $\boldsymbol{w}$ (dimension= no. of groups) in order to increase generalization. For SGD, note that $\left[\nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t)\right]_j = 1$ if the state S_t belongs to the bin j, otherwise is zero. (i.e. group indicator function). Thus (2) becomes conventional TD(0) for states in the bin.

4 Linear methods

A simple form of value function approximation is ***generalized linear approximation***.

$$\hat{v}(s, \boldsymbol{w}_t) = \sum_k w_{k,t} \phi_k(s) = \langle \boldsymbol{w}_t \,,\, \boldsymbol{\phi}(s) \rangle \tag{3}$$

where $\boldsymbol{w} = [w_{k,t}]$ is the function parameter, and $\{\phi_k\}$ is some ***basis function*** defined on state-value functional space $\mathcal{F} := \{f : \mathcal{S} \to \mathbb{R}\}$. $\boldsymbol{\phi}(s)$ is called a ***feature vector*** representing state s. WIth (3), the SGD in (1) can be formulated as

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha \left[\hat{G}_t - \hat{v}(S_t, \boldsymbol{w}_t)\right] \boldsymbol{\phi}(S_t) \tag{4}$$

since $\nabla \hat{v}(s, \boldsymbol{w}_t) = \boldsymbol{\phi}(s)$. Note that the **tabular methods** can be formulated as linear methods with **indicator feature** (**one-hot encoding**) $\phi_k(s) = \mathbb{1}\{s = s_k\}$.

Due to its simple form, linear methods have strong convergence guarantee. In fact, there is only one optimum (or, in degenerate cases, one set of equally good optima), and thus any method that is guaranteed to converge to or near a *local optimum* is automatically guaranteed to converge to or near the **global optimum**. In other words, the SGD method in (4) converges to global optimal solution given that the value target is *unbiased* and some stochastic conditions. The gradient Monte Carlo for linear approximation, for example, has the global optimality convergence results.

The generalized linear approximation has **low variance** but **high bias** (bias-variance tradeoff). This means that for semi-gradient (boostrapping) methods, which use the linear estimates in the

target function, do not converge to the global minimal. However, they do converge to a **fixed-point solution**

$$\begin{aligned}\boldsymbol{w}_{t+1} &\leftarrow \boldsymbol{w}_t + \alpha\left[R_{t+1} + \gamma\left\langle \boldsymbol{w}_t\,,\,\boldsymbol{\phi}(S_{t+1})\right\rangle - \left\langle \boldsymbol{w}_t\,,\,\boldsymbol{\phi}(S_t)\right\rangle\right]\boldsymbol{\phi}(S_t) \\ &= \boldsymbol{w}_t + \alpha\left[R_{t+1}\boldsymbol{\phi}_t - \boldsymbol{\phi}_t\left(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}\right)^T\boldsymbol{w}_t\right]\end{aligned} \tag{5}$$

In expectation, it becomes

$$\begin{aligned}\mathbb{E}\left[\boldsymbol{w}_{t+1}|\boldsymbol{w}\right] &= \boldsymbol{w}_t + \alpha\left(\boldsymbol{r} - \boldsymbol{A}\boldsymbol{w}_t\right) \\ &= \left(\boldsymbol{I} - \alpha\boldsymbol{A}\right)\boldsymbol{w}_t + \alpha\boldsymbol{r}\end{aligned}$$

where $\boldsymbol{r} = \mathbb{E}\left[R_{t+1}\boldsymbol{\phi}_t\right]$ and $\boldsymbol{A} = \mathbb{E}\left[\boldsymbol{\phi}_t\left(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}\right)^T\right]$. When it converges, the fixed-point solution is

$$\begin{aligned}\boldsymbol{0} &= \boldsymbol{r} - \boldsymbol{A}\boldsymbol{w}_t \\ \boldsymbol{w}_{TD} &= \boldsymbol{A}^{-1}\boldsymbol{r}\end{aligned}$$

which is a linear projection of expected rewards.

Theorem 4.1 *At the TD fixed point, the mean squared value error is within a bounded expansion of the lowest possible error:*

$$\overline{VE}(\boldsymbol{w}_{TD}) \leq \frac{1}{1-\gamma}\overline{VE}(\boldsymbol{w}_*)$$

Proof: Note that $\boldsymbol{A} = \mathbb{E}\left[\boldsymbol{\phi}_t\left(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}\right)^T\right]$ where $\boldsymbol{\phi}_{t+1} := \boldsymbol{\phi}(S_{t+1})$. Given the dynamic function and policy, we can be reformulated it as

$$\begin{aligned}\boldsymbol{A} &= \sum_s \mu(s)\sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\boldsymbol{\phi}(s)\left(\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s')\right)^T \\ &= \sum_s \mu(s)\sum_{s'} p(s'|s)\boldsymbol{\phi}(s)\left(\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s')\right)^T \\ &= \sum_s \mu(s)\boldsymbol{\phi}(s)\left[\boldsymbol{\phi}(s) - \gamma\sum_{s'} p(s'|s)\boldsymbol{\phi}(s')\right]^T \\ &= \boldsymbol{\Phi}^T\boldsymbol{D}\left[\boldsymbol{I} - \gamma\boldsymbol{P}\right]\boldsymbol{\Phi}\end{aligned}$$

where $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}(s_i)\right]_{1,\ldots,|\mathcal{S}|}^T \in \mathbb{R}^{|\mathcal{S}|\times d}$ using $\boldsymbol{\phi}(s_i)^T$ as its i-th row, $\boldsymbol{D} = \text{diag}\left\{\left[\mu(s_i)\right]_{1,\ldots,|\mathcal{S}|}\right\} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is the diagonal matrix of state distribution, $\boldsymbol{P} = \left[p(s'|s)\right]_{s,s'\in\mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is the state-transition matrix.

The key is to show that $\boldsymbol{D}\left[\boldsymbol{I} - \gamma\boldsymbol{P}\right] \succ \boldsymbol{0}$ is positive definite. Note that $\boldsymbol{D}$ is positive definite since its diagonal are all positive. The diagonal of $\boldsymbol{D}\left[\boldsymbol{I} - \gamma\boldsymbol{P}\right]$ is $\mu(s_i)(1 - p(s_i|s_i)) = \mu(s_i) > 0$. The off-diagonal is negative since $(-p(s_i|s_j)) < 0$ and so all we have to show is that each row sum plus the corresponding column sum is positive. This is true since the state-transition matrix $\boldsymbol{P}$ is a stochastic matrix so the row sum is positive. (because $\boldsymbol{D}\left[\boldsymbol{I} - \gamma\boldsymbol{P}\right]$ is symmetric real matrix and all of its diagonal entries are positive and greater than the sum of the absolute values of the corresponding off-diagonal entries) ■

Note that since the γ is closer to 1, the upper bound is very loose.

This fixed-point solution applies to other on-policy bootstrapping methods as well. We can show that **linear semi-gradient dynamic programming** with updates according to the on-policy distribution also converge to the same fixed-point solution $\boldsymbol{w}_{TD}$.

4.1 Choice of basis function / feature construction

The choice of different basis function or features $\phi(s)$ will affect the ability of generalization and discrimination for the function approximation. We listed several important basis functions and feature mapping:

- **Indicator function** (one-hot encoding): The simpliest feature mapping is one-hot encoding, which is essentially using each state value itself

$$\phi_k(\boldsymbol{s}) := \mathbb{1}\left\{\boldsymbol{s} = \boldsymbol{s}_k\right\}, \quad \boldsymbol{s}_k \in \mathcal{S}$$

  The main disadvantages for using one-hot encoding is that the **dimensionality** of the one-hot encoding increase exponentially when the state dimension increases. Suppose $\mathcal{S} \subset \mathcal{R}^d$ where $|\mathcal{R}| = k$, then the one-hot encoding has dimension k^d. Also one-hot encoding can only be applied to **discrete state space**. For continous space, we have to use coarse coding or tile coding.

  The **tabular methods** discussed before can be seen as **linear function approximation with one-hot encoding**. Due to sparisy of one-hot encoding, both SGD and semi-gradient methods can be easily computed by only updating weights corresponding to non-zero entries of one-hot encoding. This is essentially the same as MC, TD(0) etc.

$$\text{SGD: } \boldsymbol{w}_{t+1,k} \leftarrow \boldsymbol{w}_{t,k} + \alpha\left[\hat{G}_t - \hat{v}(S_t, \boldsymbol{w}_t)\right], \quad \text{only for k when } S_t = s_k$$

$$\text{TD(0): } \boldsymbol{w}_{t+1,k} \leftarrow \boldsymbol{w}_{t,k} + \alpha\left[R_{t+1} + \gamma \boldsymbol{w}_{t,j} - \boldsymbol{w}_{t,k}\right], \quad \text{when } S_t = s_k, S_{t+1} = s_j$$

- **Polynomials, Fourier basis and RBF basis function**: These are all different sets of classical functional basis for state function space $\mathcal{F}$. Using this basis as feature mapping, the function approximation for reinforcement learning is treated the same as the tasks of **interpolation** and **regression**. For $\boldsymbol{s} = (s_1, \ldots, s_d)$, coefficient $\boldsymbol{c}_k = (c_{1,k}, \ldots, c_{d,k})$

$$\text{order-n Polynomial-basis: } \phi_k(\boldsymbol{s}) := \prod_{j=1}^{d} s_j^{c_{j,k}},$$
$$s_i \in \mathbb{R} \text{ and } c_{j,k} \in \{0, \ldots, n\}, \; k = 0, 1, \ldots, (n+1)^d;$$
$$\text{Fourier: } \phi_k(\boldsymbol{s}) := \cos(\pi \boldsymbol{c}_k^T \boldsymbol{s}), \quad k = 0, 1, \ldots,$$
$$s_i \in [0, 1] \text{ and } c_{i,k} \in \{0, \ldots, n\}, k = 0, \ldots, (n+1)^d;$$
$$\text{Radical Basis Function (RBF): } \phi_k(\boldsymbol{s}) := \exp\left(-\frac{\|\boldsymbol{s} - \boldsymbol{c}_k\|_2^2}{2\sigma_k^2}\right),$$
$$s_i \in \mathbb{R} \text{ and } c_{i,k} \in \mathbb{R}, k = 0, \ldots, \infty$$

  In these cases, the feature vector $\boldsymbol{\phi}$ form a set of (linearly independent) basis for real-valued functional space $\mathcal{F} = \{f : \mathcal{S} \to \mathbb{R}\}$ and the weight $\boldsymbol{w}$ becomes the *coordinate* of the value function in the space $\mathcal{F}$.

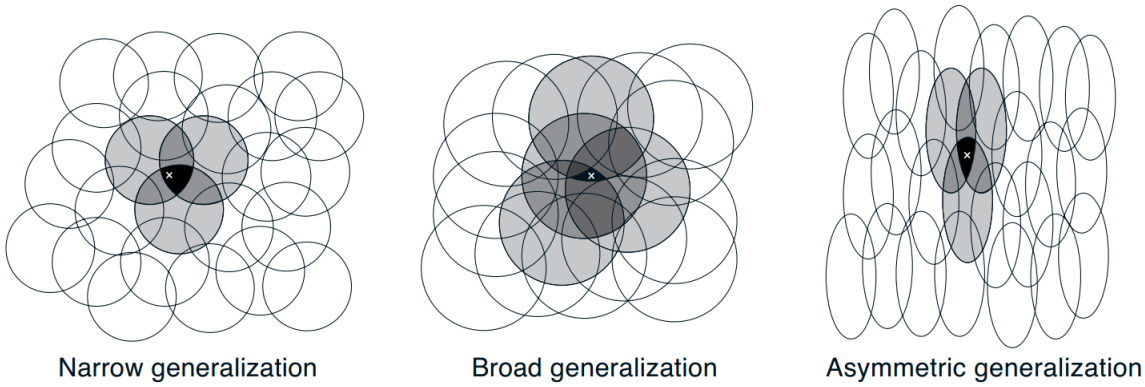

Figure 4: coarse coding

- **Coarse Coding**: Coarse Coding is an extension of one-hot encoding to groups of states. Specifically, define a collection of subset regions $\{U_1, \ldots, U_M\}$ that ***covers*** the state space so that $\mathcal{S} \subset \bigcup_i U_i$ and $U_i \cap U_j \neq \emptyset$ for some $i \neq j$. The Coarse Coding features are defined as

$$\phi_k(\boldsymbol{s}) := \mathbb{1}\left\{\boldsymbol{s} \in U_k\right\}, \quad k = 1, \ldots, M$$

  For feature vector $\boldsymbol{\phi}(\boldsymbol{s}) \in \{0,1\}^M$, the nonzero entries indicates the region index that covers the state $\boldsymbol{s}$.

  The **size** and the **shape** of collection of all region that covers the state $\boldsymbol{s}$, i.e. $\bigcup_{\{k:\boldsymbol{s}\in U_k\}} U_k$, measures the **generalization** of the coarse encoding and value approximation. Note that all states within the same region U_k shared the same feature ϕ_k and the same weight $\boldsymbol{w}_k$. So during the update, all of these states in the same region are updated simultaneously. These regions that cover $\boldsymbol{s}$ are seen as the receiptive fields of the state $\boldsymbol{s}$. In convolutional neural network, the region as well as the weight for each region are learned. CNN can be seen as a generalization of this idea.

  The **size** of non-empty overlapping of these regions $U_i \cap U_j \neq \emptyset$ measures the **discrimination** of the coarse coding and value approximation. Note that weights corresponding to U_i and U_j are updated independently. Thus, the smaller the overlapping region, the more **independent** the states in these two regions are.

  The **state aggregation** discussed above can be seen as a special case of coarse coding when $U_i \cap U_j = \emptyset$ for all i, j, i.e. $\{U_1, \ldots, U_M\}$ forms a **partition** of the state space.

- **Tile Coding**: Tile Coding is another way to ***generalize*** the *one-hot encoding* and *state aggregation* methods. Consider a set of H **tilings**. Each *tiling* defines a **partition** $\left\{U_1^h, \ldots, U_M^h\right\}$ of the state space $\mathcal{S} \subset \bigcup_i U_i^h$ and $U_i^h \cap U_j^h = \emptyset$ for all $i \neq j$. The element of tiling U_i^h is called a ***tile***. The Tile Coding features are defined as

$$\phi_{k,h}(\boldsymbol{s}) := \mathbb{1}\left\{\boldsymbol{s} \in U_k^h\right\}, \quad k = 1, \ldots, M, \quad h = 1, \ldots, H.$$

  The feature vector $\boldsymbol{\phi}(\boldsymbol{s}) := [\boldsymbol{\phi}_1(\boldsymbol{s}), \ldots, \boldsymbol{\phi}_h(\boldsymbol{s})] \in \{0,1\}^{M\,H}$ has dimension $M \times H$, where $\boldsymbol{\phi}_l := [\phi_{k,l}(\boldsymbol{s})]_{k=1,\ldots,M}$. Note that if $H = 1$, we have **state aggretation** method.

  From Figure 6, the tiles or receptive field here are **squares** rather than the circles in Figure 4. With just one tiling, the tile coding has less generalization compared to coarse coding, since

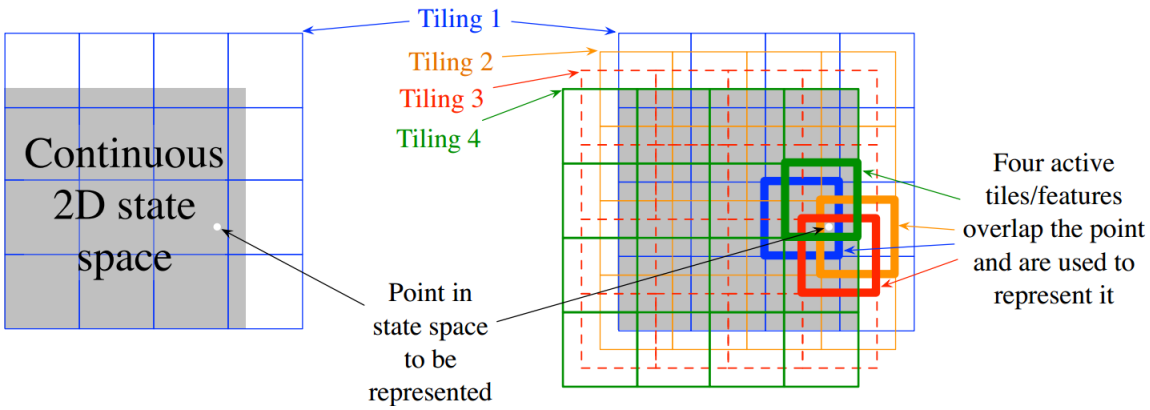

Figure 5: tile coding

two receptive fields within one tiling do not overlap. To increase overlapping, multiple tilings are used, each **offset** by a fraction of **a tile width**. Figure 6 shows the multiple tiling.

An immediate practical advantage of tile coding is that, because it works with ***partitions***, the overall number of features that are **active** at one time is **the same for any state**. (i.e. for given $\boldsymbol{s}$, there exists only one $U_{i_h}^h$ that contains it, so $\boldsymbol{\phi}(\boldsymbol{s})$ has **exactly** H non-zeros.) Exactly one feature is present in each tiling, so **the total number of features present is always the same as the number of tilings**. This allows the step-size parameter, α, to be set in an easy, intuitive way. For example, choosing $\alpha = \frac{1}{H}$, where H is the number of tilings, results in exact one-trial learning. If the example $s \to v$ is trained on, then whatever the prior estimate, $\hat{v}(\boldsymbol{s}, \boldsymbol{w}_t)$, the new estimate will be $\hat{v}(\boldsymbol{s}, \boldsymbol{w}_{t+1}) = v$.

Similar to one-hot encoding, tile coding also gains **computational advantages** from its use of **binary** feature vectors. Because each component is either 0 or 1, the weighted sum making up the approximate value function (4) or (5) is almost trivial to compute. Rather than performing d multiplications and additions, one simply computes the **indices** of the $n \ll d$ **active features** and then **adds up** the n corresponding components of the weight vector.

$$\hat{v}(\boldsymbol{s}, \boldsymbol{w}_t) = \sum_k w_{k,t}\phi_k(\boldsymbol{s}) = \sum_{\{i:\boldsymbol{s} \in U_i^h\}} w_{i,t}$$

Generalization occurs to states other than the one trained if those states fall within any of **the same tiles**, proportional to the *number of tiles in common*. Even the choice of **how to offset the tilings** from each other affects generalization. If w denotes the tile width and H the number of tilings, then $\frac{w}{H}$ is a ***fundamental unit***. Within small squares $\frac{w}{H}$ on a side, all states activate the same tiles, have the same feature representation, and the same approximated value. If a state is moved by $\frac{w}{H}$ in any cartesian direction, the feature representation changes by one *component*/tile.

We can also use non-square tiles. In choosing a tiling strategy, one has to pick the number of the tilings and the shape of the tiles. The number of tilings, along with the size of the tiles, determines the resolution or fineness of the asymptotic approximation, i.e. the **discrimination** power.

CNN is like a combination of coarse coding and tile coding since it has overlapping receiptive fields (coarse coding) and has stacked multiple filters (tile coding).

5 Function approximation via Artificial Neural Networks

Artificial Neural Networks (ANNs) are widely used to as (universal) nonlinear function approximator. In this case, a neural network can be applied to construct the feature representation

$$\phi_k(\boldsymbol{s}) := \boldsymbol{W}_k^{i+1}\sigma\left(\boldsymbol{W}^i\boldsymbol{s} + \boldsymbol{b}^i\right) + \boldsymbol{b}_k^{i+1}, i = 1, \ldots, h$$

Due to its complexity in structure, we only **highlights** the critical points when using neural networks as value function approximation $\hat{v}(s, \boldsymbol{w})$.

- Since a neural network consists of multiple layers, the weight $\boldsymbol{w} := (\boldsymbol{W}^i, \boldsymbol{b}^i, i = 1, \ldots, h)$ consists of all weight matrix and bias term at each layer.
- The stochastic gradient descent and sem-gradient descent above still holds:

$$\begin{aligned} \boldsymbol{w}_{t+1} &\leftarrow \boldsymbol{w}_t + \alpha \;\; \delta_t \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t) \\ \text{(SGD): } \delta_t &= \hat{G}_t - \hat{v}(S_t, \boldsymbol{w}_t) \\ \text{(Semi-gradient TD(0)): } \delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t) \end{aligned}$$

- Due to its complexity and layer-wise graph structure, the computation in **each step** of SGD/Semi-gradient has two **procedures** (mulitple-step loops):
  - **Forward pass** (propagation): given $\boldsymbol{w}_t$ and $\boldsymbol{s} = S_t$, to **compute** $\hat{v}(S_t, \boldsymbol{w}_t)$. This is the **inference** step. The error is computed via $\delta_t = \hat{G}_t - \hat{v}(S_t, \boldsymbol{w}_t)$ if the sample return $\hat{G}_t$ is known. For TD(0), we have to compute two forward passes **simultaneously** in order to obtain **TD error**, i.e. $\hat{v}(S_{t+1}, \boldsymbol{w}_t)$, $\hat{v}(S_t, \boldsymbol{w}_t)$, for the state S_t and its successor state S_{t+1}.

    The time complexity of forward pass is determined by the total number of layers in neural network as well as all hidden layer dimensions.
  - **Back-propagation**: In back-propagation, the algorithm need to compute the **gradient** term $\nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t)$ with respect to each weight matrices and bias terms in each layer.

    By using the **computation graph** and the **chain rule** of gradients, the algorithm can efficiently compute the gradient of each weight coefficient by one forward pass and one backward pass. Same as forward pass, the time complexity of forward pass is determined by the total number of layers in neural network as well as all hidden layer dimensions.
- For off-policy control with neural network, there is a significant issue with bootstrap **bias** and the ***positive feedback loop*** that can cause. Q-learning estimates can diverge because of this. Note that the weight update in SGD as (1) is **not the same** as the SGD used for training neural network in supervised learning. In the latter case, the target function is fixed, unknown and the inputs are independent, whereas the state S_t and S_{t+1} are correlated and the target $\hat{G}_t$ is dynamic, determined by samples or previous estimates. Thus the convergence of SGD/Semi-gradient algorithm for neural network function approximation is not guaranteed.

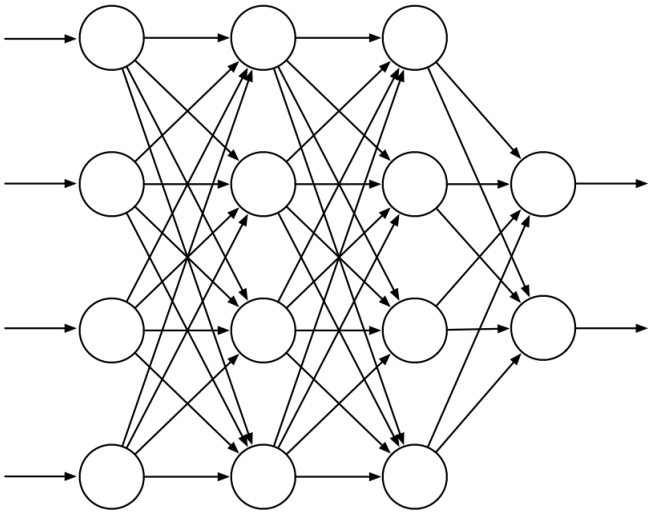

Figure 6: The artificial neural network

The **Off-policy Divergence** is the main challenge for off-policy learning with function approximationthat the distribution of updates does not match the onpolicy distribution.

- For TD error, we need to keep a copy old network $f(\boldsymbol{w}_{t-1})$ as well as the newly updated network $f(\boldsymbol{w}_t)$, since we need to use the old network for next state as well as old network for current state to construct the TD error.
- For Q learning, **maximisation bias** is a problem, whereby the action chosen is more likely to have an over-estimate of its true value. This can be fixed by **double Q-learning**.
- For Q-learning, using deep learning architecture we can learn complex value function. For instance the ***Deep Q-Network (DQN)*** [François-Lavet et al., 2018] algorithm introduced by Mnih et al. (2015) is able to obtain strong performance in an online setting for a variety of ATARI games, directly by learning from the pixels. Check other deep RL models in [François-Lavet et al., 2018]

References

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.