Off-Policy Methods with Approximation

Monday, June 22, 2020 8:52 AM

Two main problems in off-policy learning

- Effectively changing the update targets
- The distribution of updates doesn't match the on-policy distribution
  - This is only for function approximation methods

Per-Step Importance Sampling Ratio:

$$\rho_t \equiv \rho_{t:t} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Semi-Gradient Off-Policy TD(0):

$$\vec{w}_{t+1} \equiv \vec{w}_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t,\vec{w}_t)$$

- TD Error:
  - Episodic and Discounted:

    $$\delta_t \equiv R_{t+1} + \gamma\hat{v}(S_{t+1},\vec{w}_t) - \hat{v}(S_t,\vec{w}_t)$$

  - Continuing and Undiscounted:

    $$\delta_t \equiv R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1},\vec{w}_t) - \hat{v}(S_t,\vec{w}_t)$$

Semi-Gradient Expected Sarsa:

$$\vec{w}_{t+1} \equiv \vec{w}_t + \alpha\delta_t\nabla\hat{q}(S_t,A_t,\vec{w}_t)$$

- TD Error:
  - Episodic and Discounted:

    $$\delta_t \equiv R_{t+1} + \gamma\sum_a \pi(a|S_{t+1})\hat{q}(S_{t+1},a,\vec{w}_t) - \hat{q}(S_t,A_t,\vec{w}_t)$$

  - Continuing and Undiscounted:

    $$\delta_t \equiv R_{t+1} - \bar{R}_t + \sum_a \pi(a|S_{t+1})\hat{q}(S_{t+1},a,\vec{w}_t) - \hat{q}(S_t,A_t,\vec{w}_t)$$

n-Step Semi-Gradient Sarsa:

$$\vec{w}_{t+n} \equiv \vec{w}_{t+n-1} + \alpha\cdot\prod_{i=0}^{n-1}\rho_{t+i}\cdot\left[G_{t:t+n} - \hat{q}(S_t,A_t,\vec{w}_{t+n-1})\right]\nabla\hat{q}(S_t,A_t,\vec{w}_{t+n-1})$$

- Return:
  - Episodic and Discounted:

    $$G_{t:t+n} \equiv \sum_{k=0}^{n-1}\gamma^k R_{t+k+1} + \gamma^n\hat{q}(S_{t+n},A_{t+n},\vec{w}_{t+n-1})$$

  - Continuing and Undiscounted:

    $$G_{t:t+n} \equiv \sum_{i=0}^{n} R_{t+i+1} - \bar{R}_{t+i} + \hat{q}(S_{t+n},A_{t+n},\vec{w}_{t+n-1})$$

    $$= R_{t+1} - \bar{R}_t + \cdots + R_{t+n} - \bar{R}_{t+n-1} + \hat{q}(S_{t+n},A_{t+n},\vec{w}_{t+n-1})$$

n-Step Tree Backup:

$$\vec{w}_{t+n} \equiv \vec{w}_{t+n-1} + \alpha\left[G_{t:t+n} - \hat{q}(S_t,A_t,\vec{w}_{t+n-1})\right]\nabla\hat{q}(S_t,A_t,\vec{w}_{t+n-1})$$

$$G_{t:t+n} \equiv \hat{q}(S_t,A_t,\vec{w}_{t-1}) + \sum_{k=t}^{t+n-1}\delta_k\prod_{i=t+1}^{k}\gamma\pi(A_i,S_i)$$

- Where δ_t is as defined above

Deadly Triad:

- Instability and divergence can result from combining all of the following elements in one system
  - Function Approximation
  - Bootstrapping

Off-Policy Training

Function approximation cannot easily be given up since it generalizes the best to large problems

Giving up bootstrapping incurs losses in computational and data efficiency, and generally slower learning

Giving up off-policy training may cause problems for general intelligence if the task is considered as learning the results of our actions when we act in different ways

A value function can be thought of as corresponding to a dimension preserving transformation of state space

The space of possible weight vectors used in function approximation is then a subspace of this space of value functions

A weight vector then represents a point in this subspace

With linear function approximation, this subspace is a plane

Distance Between Value Functions:

$$\|v\|_\mu^2 \equiv \sum_{s\in S} \mu(s)v(s)^2$$

This is used instead of the Euclidean norm to take into account some states being more important than others

Recall that μ is often the on-policy distribution

Mean Squared Error Loss Using This:

$$\overline{VE}(\vec{w}) \equiv \left\|v_{\vec{w}} - v_\pi\right\|_\mu^2$$

Projection Operator Taking any Value Function to the Closest Representation:

$$\prod v \equiv v_{\vec{w}}$$

$$\vec{w} \equiv \underset{\vec{w}\in R^d}{\operatorname{argmin}} \left\|v - v_{\vec{w}}\right\|_\mu^2$$

Where $\prod$ is here the projection operator being defined

Note that the representable value function closest to the true value function is the projection of the true value

This is the solution found by MC methods

Projection Matrix:

$$\prod \equiv \vec{X}\left(\vec{X}^T\vec{D}\vec{X}\right)^{-1}\vec{X}^T\vec{D}$$

Where $\vec{D}$ represents a diagonal matrix with $\mu(s)$ on the diagonal and $\vec{X}$ represents a matrix whose rows are the feature vectors $\vec{x}(s)^T$

Distance with this Matrix:

$\|v\|_\mu^2 \equiv \vec{v}^T\vec{D}\vec{v}$

Closest Value Function:

$\vec{v}_{\vec{w}} = \vec{X}\vec{w}$

Bellman Error:

$$\bar{\delta}_{\vec{w}}(s) \equiv \left(\sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\left[r + \gamma v_{\vec{w}}(s')\right]\right) - v_{\vec{w}}(s)$$

$$= E\left[R_{t+1} + \gamma v_{\vec{w}}(S_{t+1}) - v_{\vec{w}}(S_t)\middle|S_t = s, A_t{\sim}\pi\right]$$

Note that this is the expectation of the TD error

Bellman Error Vector:

$\bar{\delta}_{\vec{w}}$

A vector containing the Bellman error at every state

Mean Squared Bellman Error:

$$\overline{BE}(\vec{w}) = \left\|\bar{\delta}_{\vec{w}}\right\|_\mu^2$$

For linear function approximation there is a unique value of $\vec{w}$ that minimizes this

Bellman Operator:

$$\left(B_\pi(s)\right) \equiv \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$$

This transforms a value function to its Bellman error vector, if it is applied to a function already in the representable subspace it will generally produce a new value function outside the subspace

The true value function is the only stationary point of the Bellman Operator (i.e. $v_\pi = B_\pi v_\pi$)

Note that this implies that the true value function can be found through repeated application of the Bellman Operator (this process is similar to DP)

Note that this process will alternate between representable value functions and non-representable value functions

This is more commonly called a dynamic programming operator and denoted T^π

A generalized form called the $TD(\lambda)$ operator also exists and is denoted $T^{(\lambda)}$

Bellman Error Vector Using Operator:

$\bar{\delta}_{\vec{w}} = B_\pi v_{\vec{w}} - v_{\vec{w}}$

Mean Square Projected Bellman Error:

$$\overline{PBE}(\vec{w}) = \left\| \prod \bar{\delta}_{\vec{w}} \right\|_\mu^2$$

Recall that here $\prod$ is the projection operator defined above

The TD fixed point is the point where this is 0

Matrix Form:

$$\overline{PBE}(\vec{w}) = \bar{\delta}_{\vec{w}}^T \vec{D}\vec{X}\left(\vec{X}^T\vec{D}\vec{X}\right)^{-1}\vec{X}^T\vec{D}\bar{\delta}_{\vec{w}} = \left(\vec{X}^T\vec{D}\bar{\delta}_{\vec{w}}\right)^T\left(\vec{X}\vec{D}\vec{X}\right)^{-1}\left(\vec{X}^T\vec{D}\bar{\delta}_{\vec{w}}\right)$$

Gradient:

$$\nabla\overline{PBE}(\vec{w}) = 2\nabla\left[\vec{X}\vec{D}\delta_{\vec{w}}\right]^T\left(\vec{X}\vec{D}\vec{X}\right)^{-1}\left(\vec{X}\vec{D}\bar{\delta}_{\vec{w}}\right)$$

In terms of the behavior policy distribution:

$$\nabla\overline{PBE}(\vec{w}) = 2E\left[\rho_t(\gamma\vec{x}_{t+1} - \vec{x}_t)\vec{x}_t^T\right]E\left[\vec{x}_t\vec{x}_t^T\right]^{-1}E\left[\rho_t\delta_t\vec{x}_t\right]$$

In practice, methods learn and store a vector $\vec{v} \approx E\left[\vec{x}_t\vec{x}_t^T\right]^{-1}E\left[\rho_t\delta_t\vec{x}_t\right]$

Iterative Form:

$$\vec{v}_{t+1} \equiv \vec{v}_t + \beta\rho_t(\delta_t - \vec{v}_t^T\vec{x}_t)\vec{x}_t$$

Where β is another step-size parameter

This is possible because the above definition of is the solution to a linear least-squares problem (a problem involving minimizing the least squares error)

MC methods are a form of SGD

Mean Squared TD Error:

$$\overline{TDE}(\vec{w}) = \sum_{s\in S}\mu(s)E\left[\delta_t^2|S_t = s, A_t{\sim}\pi\right] = \sum_{s\in S}\mu(s)E\left[\rho_t\delta_t^2|S_t = s, A_t = b\right] = E_b\left[\rho_t\delta_t^2\right]$$

Note that this is for the off-policy case

Naïve Residual Gradient Algorithm:

$$\vec{w}_{t+1} = \vec{w} + \alpha\rho_t\delta_t\left(\nabla\hat{v}(S_t, \vec{w}_t) - \gamma\nabla\hat{v}(S_{t+1}, \vec{w})\right)$$

Note that this is the same as the semi-gradient TD algorithm but with an extra term

Note that this converges to the values that minimize the mean squared TD error, which are not necessarily the true value

Residual Gradient Algorithm:

$$\vec{w}_{t+1} = \vec{w}_t + \alpha\left[E_b\left[\rho_t\left(R_{t+1} + \gamma\hat{v}(S_{t+1}, \vec{w})\right)\right] - \hat{v}(S_t, \vec{w})\right]\left[\nabla\hat{v}(S_t, \vec{w}) - \gamma E_b\left[\rho_t\nabla\hat{v}(S_{t+1}, \vec{w})\right]\right]$$

This uses the Bellman error, so it converges to the true values

Sampling the values in the expectation values leads almost exactly to the naïve residual gradient algorithm above

Getting an unbiased sample requires two independent samples of the next state, which is obviously not possible in a real environment but is possible in a simulated environment

This converges very slowly

Learnable:

Quantities that can be computed or estimated from the observed sequence of feature vectors, actions, and rewards

Note that quantities that are not learnable can still be computed given knowledge of the internal structure of the environment or if the state sequence was observed instead of the feature vector representations

The mean squared Bellman error is not learnable

The mean squared error is also not learnable since it can be different for two processes that generate the same observable sequence

However, the parameter that optimizes it is learnable so the mean squared error is a useful objective

Mean Square Return Error:

$$\overline{RE}(\vec{w}) = E\left[\left(G_t - \hat{v}(S_t, \vec{w})\right)^2\right] = \overline{VE}(\vec{w}) + E\left[\left(G_t - v_\pi(S_t)\right)^2\right]$$

This is always learnable

Note that this is for the on-policy case

Also note that the second form is the mean squared error but with and extra term that doesn't depend on the optimal parameter

This is how the parameter that optimizes the mean squared error can be learned

The mean squared projected Bellman error $\overline{PBE}$ and mean squared TD error $\overline{TDE}$ are learnable

Because the Bellman error is not learnable, no algorithm can minimize it without access to the underlying states, not just feature vectors

Note that this means it can still be used in model-based systems

GTD2:

$$\vec{w}_{t+1} = \vec{w}_t + \alpha E\left[\rho_t(\vec{x}_t - \gamma\vec{x}_{t+1})\vec{x}_t^T\right]E\left[\vec{x}_t\vec{x}_t^T\right]^{-1}E\left[\rho_t\delta_t\vec{x}_t\right] \approx \vec{w}_t + \alpha\rho_t(\vec{x}_t - \gamma\vec{x}_{+1})\vec{x}_t^T\vec{v}_t$$

Note that the last form involves sampling and uses definitions from the matrix form of the projected Bellman error given above

TD(0) with Gradient Correction (TDC):

$$\vec{w}_{t+1} = \vec{w}_t + \alpha\left(E\left[\vec{x}_t\rho_t\delta_t\right] - \gamma E\left[\rho_t\vec{x}_{t+1}\vec{x}_t^T\right]E\left[\vec{x}_t\vec{x}_t^T\right]^{-1}E\left[\rho_t\delta_t\vec{x}_t\right]\right) \approx \vec{w}_t + \alpha\rho_t\left(\delta_t\vec{x}_t - \gamma\vec{x}_{t+1}x_t^T\vec{v}_t\right)$$

Note that the last form involves sampling

This is also known as GTD(0)

Cascade:

A dependence where one part of a learning process assumes that another part of a learning process has finished

Note that this occurs for the above algorithms, which assume that $\vec{v}$ has been fully learned

Two-Time Scale Proofs:

Proofs that assume that the secondary learning process in a cascade has reached its asymptotic value before being applied to the primary learning process

The name comes from the secondary process being on a "fast" time scale and the primary process being on a "slow" time scale

If α is the step-size for the primary process and β is the step-size for the secondary process, these proofs typically require that $\beta \to 0$ and $\frac{\alpha}{\beta} \to 0$ as time goes to infinity

Pseudo Termination:

Termination that doesn't affect the sequence of state transitions, but does affect the learning process and quantities being learned

This is related to thinking of discounting as probabilistic termination

One-Step Emphatic TD:

$$\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \vec{w}_t) - \hat{v}(S_t, \vec{w}_t)$$
$$\vec{w}_{t+1} = \vec{w}_t + \alpha M_t\rho_t\delta_t\nabla\hat{v}(S_t, \vec{w}_t)$$
$$M_t = \gamma\rho_{t-1}M_{t-1} + I_t$$

Where I_t is the interest (arbitrary) and M_t (the emphasis) is initialized to $M_{t-1} = 0$

Off-policy learning is inherently of greater variance than on-policy learning