Reinforcement Learning with Tensorflow

In exercise 11, we made use of the quadratic TD error as our loss function:

$$\begin{aligned}\mathcal{L} &= \delta^2\\ &= (\underbrace{r_{k+1} + \gamma\hat{q}(\boldsymbol{x}_{k+1}, u_{k+1}, \boldsymbol{w})}_{\text{target}} - \underbrace{\hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})}_{\text{prediction}})^2.\end{aligned}$$

The semi-gradient of this function is then be given by:

$$\nabla_{\boldsymbol{w}}\delta^2 = -2\,\delta\,\nabla_{\boldsymbol{w}}\hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w}).$$

> The true gradient would of course be $\nabla_{\boldsymbol{w}}\delta^2 = 2\,\delta\,\nabla_{\boldsymbol{w}}[\gamma\hat{q}(\boldsymbol{x}_{k+1}, u_{k+1}, \boldsymbol{w}) - \hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})]$. However, Tensorflow will not be able to compute this gradient, unless the prediction $\hat{q}(\boldsymbol{x}_{k+1}, u_{k+1}, \boldsymbol{w})$ happens within the same gradient tape as the prediction of $\hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})$. If we estimate the target without the use of a gradient tape it will be considered as a constant number, whose derivative will then be zero.

Tensorflow will inherently minimize loss, so it will perform a gradient **descent** with this gradient when the `apply_gradients` function is used:

$$\begin{aligned}\boldsymbol{w}_{\text{new}} &= \boldsymbol{w}_{\text{old}} - \alpha\nabla_{\boldsymbol{w}}\delta^2\\ &= \boldsymbol{w}_{\text{old}} + 2\,\alpha\,\delta\,\nabla_{\boldsymbol{w}}\hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w}).\end{aligned}$$

Which is the desired operation for Sarsa(0), when assuming the presence of the factor 2 to be only of minor importance, or by defining $\tilde{\alpha} = 2\alpha$.

Algorithmic Implementation: Semi-Gradient Sarsa

input: a differentiable function $\hat{q} : \mathbb{R}^\kappa \times \mathbb{R}^\varsigma \rightarrow \mathbb{R}$
input: a policy π (only if estimating q_π)
parameter: step size $\alpha \in \{\mathbb{R}|0 < \alpha < 1\}$, $\varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$
init: parameter vector $\boldsymbol{w} \in \mathbb{R}^\varsigma$ arbitrarily
for $j = 1, 2, \dots$ *episodes* **do**
  initialize $\boldsymbol{x}_0$;
  for $k = 0, 1, 2 \dots$ *time steps* **do**
    $u_k \leftarrow$ apply action from $\pi(\boldsymbol{x}_k)$ or ε-greedy on $\hat{q}(\boldsymbol{x}_k, \cdot, \boldsymbol{w})$;
    observe $\boldsymbol{x}_{k+1}$ and r_{k+1};
    if $\boldsymbol{x}_{k+1}$ *is terminal* **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\,[r_{k+1} - \hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})]\,\nabla_{\boldsymbol{w}}\hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})$;
      go to next episode;
    choose u' from $\pi(\boldsymbol{x}_{k+1})$ or ε-greedy on $\hat{q}(\boldsymbol{x}_{k+1}, \cdot, \boldsymbol{w})$;
    $\boldsymbol{w} \leftarrow$
     $\boldsymbol{w} + \alpha\,[r_{k+1} + \gamma\hat{q}(\boldsymbol{x}_{k+1}, u', \boldsymbol{w}) - \hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})]\,\nabla_{\boldsymbol{w}}\hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})$;

Algo. 10.2: Semi-gradient Sarsa action-value estimation (output: parameter vector $\boldsymbol{w}$ for $\hat{q}_\pi$ or $\hat{q}^*$)

Oliver Wallscheid | RL Lecture 10 | 10

However, for Sarsa(λ), we want a different gradient:

Algorithmic Implementation: Semi-Gradient Sarsa(λ)

input: a differentiable function $\hat{q}: \mathbb{R}^\kappa \times \mathbb{R}^\zeta \rightarrow \mathbb{R}$
input: a policy π (only if estimating q_π)
parameter: $\alpha \in \{\mathbb{R}|0 < \alpha < 1\}$, $\varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$, $\lambda \in \{\mathbb{R}|0 \leq \lambda \leq 1\}$
init: parameter vector $\boldsymbol{w} \in \mathbb{R}^\zeta$ arbitrarily
for $j = 1, 2, \ldots$ *episodes* **do**
  initialize $\boldsymbol{x}_0$ and set $\boldsymbol{z} = 0$;
  $u_0 \leftarrow$ choose action from $\pi(\boldsymbol{x}_0)$ or ε-greedy on $\hat{q}(\boldsymbol{x}_0, \cdot, \boldsymbol{w})$;
  for $k = 0, 1, 2 \ldots$ *time steps* **do**
    apply action u_k, observe $\boldsymbol{x}_{k+1}$ and r_{k+1};
    if $\boldsymbol{x}_{k+1}$ *is terminal* **then** $\delta \leftarrow r_{k+1} - \hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})$;
    else
      $u_{k+1} \leftarrow \pi(\boldsymbol{x}_{k+1})$ or ε-greedy on $\hat{q}(\boldsymbol{x}_{k+1}, \cdot, \boldsymbol{w})$;
      $\delta \leftarrow r_{k+1} + \gamma \hat{q}(\boldsymbol{x}_{k+1}, u_{k+1}, \boldsymbol{w}) - \hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})$;
    $\boldsymbol{z} \leftarrow \gamma\lambda\boldsymbol{z} + \nabla_{\boldsymbol{w}}\hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})$;
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta\boldsymbol{z}$;
    exit loop if $\boldsymbol{x}_{k+1}$ is terminal;

Algo. 11.3: Semi-gradient Sarsa(λ) (output: parameter vector $\boldsymbol{w}$ for $\hat{q}_\pi$ or $\hat{q}^*$)

Oliver Wallscheid | RL Lecture 11 | 27

In this case, it is easier to directly calculate the derivative of the action-value approximator:

$$\mathcal{L} = \hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w}).$$

This way, we receive $\nabla_{\boldsymbol{w}}\hat{q}(\boldsymbol{x}_k, u_k, \boldsymbol{w})$ without any scaling factors.