

The problem is not coming from the above equation but rather arise for bootstrapping metod. [Sutton's reinforcement learning book (2nd)](http://incompleteideas.net/book/the-book-2nd.html) gave a pretty good explanation to the problem in Chapter 9.3.

The idea here is you cannot obtain **true value** for V\_pi, but you need to get an approximation to it. My understand is when bootstrapping, such as TD method, V\_pi depends on w. This will break the assumption that V\_pi is indenpendent of w and thus the gradient we get is not the true gradient and we call it semi-gradient. However, if you are not using bootstrapping, such as Monte Carlo, V\_pi will be unbias and you will not have this problem and the above will guarantee to converge to a local minimal.

In 2009, Maei et al. (prominent RL researchers) published Convergent temporal-difference learning with arbitrary smooth function approximation [1], which described "true" gradient descent variants of TD learning (normally, you don't backpropagate through the next-state value estimate, making conventional TD(0) a semi-gradient method).

Those variants are GTD (Gradient Temporal Differences), GTD2 (v2 of GTD), and TDC (TD with gradient Corrections), and the paper proved convergence even in the off-policy case with neural networks.

To quote:

In this paper, we solved a long-standing open problem in reinforcement learning, by establishing a family of temporal-difference learning algorithms that converge with arbitrary differentiable func- tion approximators (including neural networks). The algorithms perform gradient descent on a nat- ural objective function, the projected Bellman error. The local optima of this function coincide with solutions that could be obtained by TD(0). Of course, TD(0) need not converge with non-linear function approximation. Our algorithms are on-line, incremental and their computational cost per update is linear in the number of parameters

But I'm unable to find any studies that apply gradient TD methods to neural networks in modern Deep RL. Are there issues with convergence speed? Unscalable computation? Why are we still stabilizing off-policy TD with target networks?

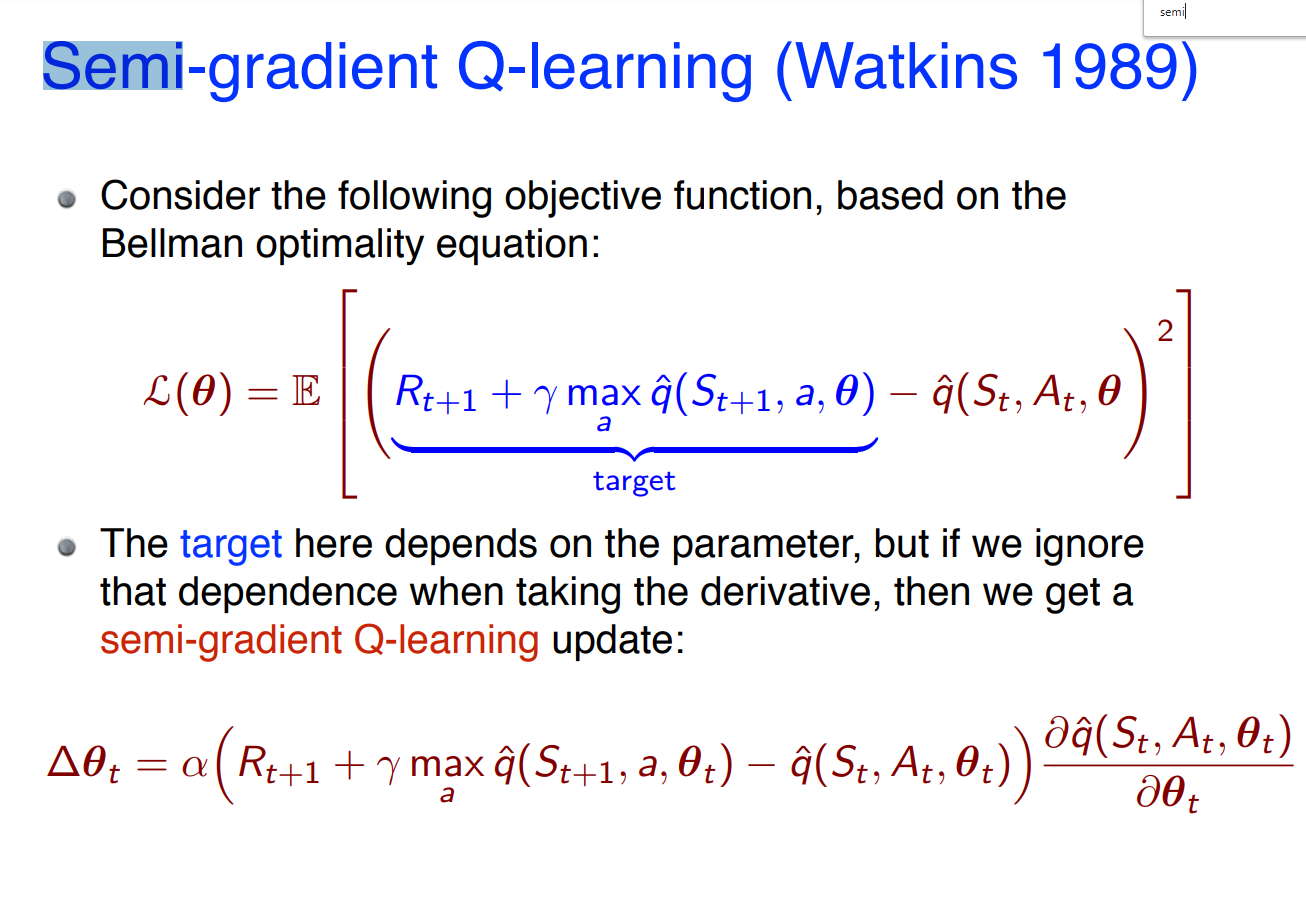
The Deepmind people are aware of these algorithms; the paper gets a passing mention in the Arxiv version of the DQN paper. Have people tried these out but just didn't publish negative results?

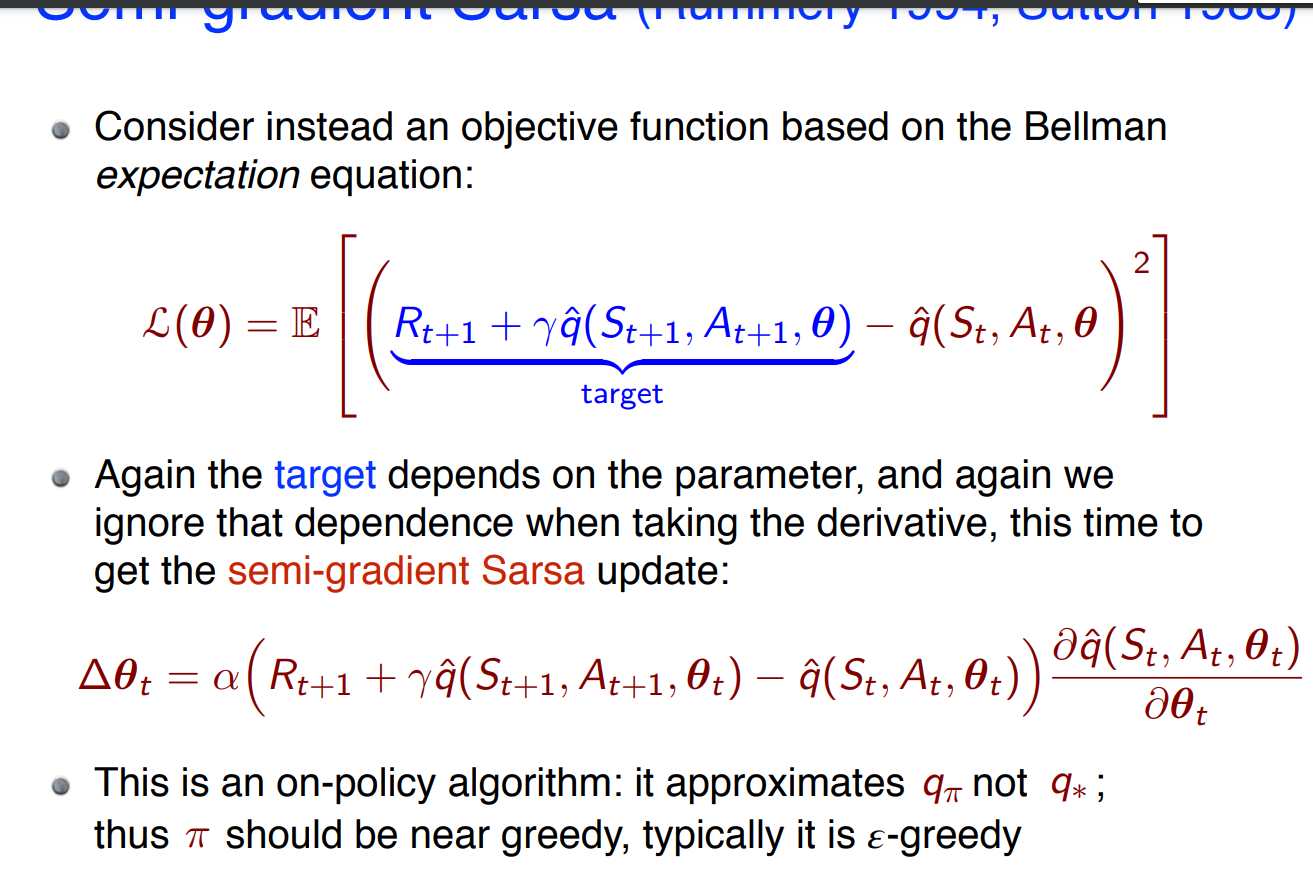
GTD and TDC cannot be applied directly to neural networks because they involve projection on to the space of the basis functions.

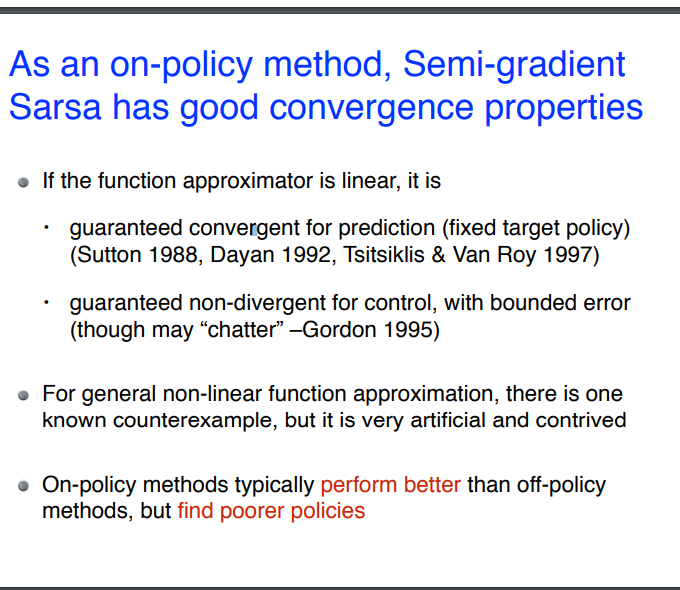
There is a follow up paper that manages to do this for neural networks be projecting on to the tangent plane of the manifold learnt by the neural network. (I'll add a link when I get a chance).

Moreover, GTD for neural networks has been done only for policy evaluation, not control.

But everyone I've spoken to says that "semi-gradient" TD works faster in practice on almost all problems, and the solution is reaches is not qualitatively worse than GTD.









7]. However, the empirical success of Gradient TD methods is limited to simple domains due to its large variance [48]. Semi-gradient algorithms are not convergent in general, e.g., the divergence of off-policy linear TD is well-documented [52]. Semi-gradient algorithms are fast but in general not conver

ν −. Both DQN and DDPG are semi-gradient algorithms. There are also true gradient methods for control, e.g., Greedy-GQ [32] and the residual version of Q-learning [2]. As with Gradient TD methods, the empirical success of Greedy-GQ is limited to simple domains due to its large v

<https://ins.sjtu.edu.cn/files/paper/20191021090916_Book%20(3).pdf>

6.2 Stochastic Gradient and Semi-Gradient Methods Before we can start to calculate the weights in the approximations of the value functions, we must specify the optimization objective or the error we seek to minimize. One of the most popular errors is the mean squared value error VE(w) := X s∈S µπ(s)(vπ(s) − vˆw(s))2 , where µπ is the discounted state distribution (see Definition 7.1). The discounted state distribution acts as weight for the squared differences in the true and approximated values. Of course, other weights can be used whenever it makes sense to assign different importance to the states. The goal is to find a global optimum w∗ , i.e., a weight vector w∗ such that VE(w∗ ) ≤ VE(w) for all w ∈ W ⊂ R d . It is sometimes possible to show that a global optimum is found when linear function approximations are used, but it becomes much harder in the case of nonlinear function approximation. Short of finding a global optimum, the goal is to find a local optimum, i.e., a weight vector w∗ such that VE(w∗ ) ≤ VE(w) holds for all w in a neighborhood of w∗ . The most popular method for function approximations is stochastic gradient descent (SGD), and it is very well suited for online learning. In SGD, it is assumed that the approximate value function vˆw is a differentiable function of the weight vector w. The weight vector calculated in each iteration is denoted by wt for t ∈ {0, 1, 2, . . .}. We assume for now that in each iteration a new sample vπ(St ) becomes available having reached state St . SGD means improving the weight vector wt by moving it slightly downhill with respect to the error VE in the direction of the greatest change in the error at wt . This direction of greatest change is the gradient, and minimizing the error means adding a small multiple of the negative gradient. This results in the iteration wt+1 := wt − 1 2 αt∇w(vπ(St ) − vˆwt (St ))2 (6.1a) = wt + αt (vπ(St ) − vˆwt (St ))∇wvˆwt (St ), (6.1b) where the learning rate αt ∈ R +. The sole purpose of the factor 1/2 in the first line is to not have a factor 2 in the second line. 54 6.2. Stochastic Gradient and Semi-Gradient Methods SGD is a stochastic gradient-descent method since the update is a random variable because it depends on the random variable St . Over many samples or iterations, the accumulated effect is to minimize the average of an objective function such as the mean squared value error. To ensure convergence, the learning rate αt must tend to zero. Unfortunately, the true value vπ(St ) is unknown, since vπ is to be calculated. Therefore, in fact, we can only use a random variable Ut instead of vπ(St ) in the iteration. Hence the general SGD method for the prediction of state values is the iteration wt+1 := wt + αt (Ut − vˆwt (St ))∇wvˆwt (St ). (6.2) If Ut is an unbiased estimate of vπ(St ), i.e., if E[Ut | St = s] = vπ(St ) for all times t and if the learning rate α satisfy the conditions (2.2) for stochastic approximation, then the wt converge to a local optimum. Equipped with the SGD method, we can now develop algorithms for calculating w∗ based on different choices for the target value Ut . Probably the most obvious choice for an unbiased estimate of vπ(St ) is the Monte-Carlo target Ut := Gt . Based on the convergence results just mentioned, the general SGD method in conjunction with this estimate converges to a locally optimal approximation of vπ(St ). In other words, the algorithm for the Monte-Carlo state-value prediction can be shown to always find a locally optimal solution. The resulting algorithm is shown in Algorithm 14. Note that the episode must have ended so that Gt can be calculated in each time step. Bootstrapping targets such as the n-step return Gt:t+n , which build on previously calculated approximations, do not provide the same convergence guarantees. By the definition of bootstrapping, the target Ut in a bootstrapping method depends on the current weight vector wt , which means that the estimate is biased. Bootstrapping methods are not even gradient-descent methods. This becomes clear by considering the derivative calculated in (6.1). While the derivative of vˆwt (St ) appears in the equation, in a bootstrapping method the derivative of Ut (wt ) ≈ vπ(St ) is nonzero, but does not appear in the equation. Because of this missing term, these methods are called semigradient methods. On the other hand, semigradient methods often learn significantly faster and they converge reliably in the important case of linear function approximation.

Additionally, they enable online learning in contrast to MC methods, which have to wait till the end of an episode. The most straightforward semigradient method is probably the semigradient TD(0) method, which uses the target Ut := Rt+1 + γvˆwt (St+1 ).

Q-learning is a bootstrapping method. The new value Qt+1 (st , at ) can also be written as Qt+1 (st , at ) | {z } new value := Qt (st , at ) | {z } old value +αt (rt+1 + γ max a∈A (st+1 ) Qt (st+1 , a) | {z } target value −Qt (st , at ) | {z } old value ), 87 Chapter 9. Convergence of Q-Learning which is the form of a semigradient SGD method with a certain linear function approximation. The learning rate αt must be chosen appropriately, and convergence results hold only under certain conditions on the learning rate. If the environment is fully deterministic, the learning rate αt := 1 is optimal. If the environment is stochastic, then a necessary condition for convergence is that limt→∞ αt = 0. If the initial approximation Q0 is defined to have large values, exploration is encouraged at the beginning of learning. This kind of initialization is known as using optimistic initial conditions.