IEOR 8100: Reinforcement learning

Lecture 3: Large-scale RL: function approximation

By Shipra Agrawal

Q-learning and TD learning algorithms discussed in the last lecture are an instance of dynamic programming based algorithms for RL. These are iterative algorithms that try to find fixed point of Bellman equations. When the state space is large, tabular methods that enumeratively learn Q-values for every state and action (or value function for every state) are no longer useful. In such settings, we approximate the value-function/Q-function as a parametric function of the state/action features, which allows scalability and transfer learning. Here, we will examine some versions of these algorithms and their convergence properties.

1 TD-learning and Q-learning with function approximation

1.1 Q-learning with function approximation

The tabular Q-learning does not scale with increase in the size of state space. In most real applications, there are too many states too keep visit, and keep track of. For scalability, we want to *generalize*, i.e., use what we have learned about already visited (relatively small number of) states, and generalize it to new, similar states. A fundamental idea is to use 'function approximation', i.e., use a lower dimensional feature representation of the state-action pair s, a and learn a parametric approximation $Q_\theta(s,a)$. For example, the function $Q_\theta(s,a)$ could simply be represented as a linear function in θ and features $x(s,a)$: $Q_\theta(s,a) = \theta_0 + \theta_1 x_1(s,a) + \ldots + \theta_n x_n(s,a)$; or through a deep neural net, $Q_\theta(s,a) = f_\theta(x(s,a))$ with θ being the parameters of the DNN. Under such representations, given parameter θ, the Q-function can be computed for an unseen s, a. Instead of learning the $|S| \times |A|$ dimensional Q-table, the Q-learning algorithm will learn the parameter θ. Here, on observing sample transition to s' from s on playing action a, instead of updating the estimate of $Q(s,a)$ in the Q-table, the algorithm updates the estimate of θ. Intuitively, we are trying to find a θ such that for every s, a, the Bellman equation,

$$Q_\theta(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s,a,s') + \gamma \max_{a'} Q_\theta(s',a')]$$

can be approximated well for all s, a. Or, in other words, we are trying to minimize some loss function like squares loss:

$$\ell_\theta(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}(Q_\theta(s,a) - R(s,a,s') - \gamma \max_{a'} Q_\theta(s',a'))^2 =: \mathbb{E}_{s' \sim P(s,a,\cdot)}\left[\ell_\theta(s,a,s')\right]$$

where

$$\ell_\theta(s,a,s') = \left(Q_\theta(s,a) - target(s')\right)^2$$
$$target(s') = R(s,a,s') + \gamma \max_{a'} Q_\theta(s',a')$$

Fundamentally, Q-learning uses gradient descent to optimize this loss function given sample (target) observations at s'.

Q-learning Algorithm.

Start with initial state $s = s_0$. In iteration $k = 1, 2, ...,$

- Take an action a.
- Observe reward r, transition to state $s' \sim P(\cdot|s,a)$.

- $\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla_{\theta_k} \ell_{\theta_k}(s, a, s')$, where

$$\nabla_\theta \ell_{\theta_k}(s, a, s') = -\delta_k \nabla_{\theta_k} Q_{\theta_k}(s, a)$$

$$\delta_k = r + \gamma \max_{a'} Q_{\theta_k}(s', a') - Q_{\theta_k}(s, a)$$

- $s \leftarrow s'$,

If s' reached at some point is a terminal state, s is reset to starting state.

Effectively, in the tabular Q-learning method discussed in the previous lecture, after taking action a_t and observing s_{t+1}, replace Step 8 and Step 9 by:

8: Let $\delta_t := r_t + \gamma \max_{a'} Q_{\hat{\theta}}(s_{t+1}, a') - Q_{\hat{\theta}}(s, a)$.

9: Update $\hat{\theta} \leftarrow \hat{\theta} + \alpha_t \delta_t \nabla_{\hat{\theta}} Q_{\hat{\theta}}(s, a)$.

Challenge: Exploration. Note that we are trying to minimize many loss functions (one for each s, a) simultaneously. At any point, the number of samples available for different state and action pairs are different, and depend on how much we explore a given state and that action. Therefore, depending on the both the transition dynamics and our exploration scheme, at any point, we will have different level of accuracies for different state and actions. Note that not all s, a need to have high accuracy in order to find optimal policy. For example, if some states are rare and have low value, in those states loss function does not need to be optimized. Further, if two states/action pairs have similar features, then both don't need to be explored in order to learn a good parameter θ. This points to significance of an adaptive exploration scheme which manages appropriate accuracy levels across state and actions in order to quickly converge to a good θ.

A simple exploration scheme often utilized is ϵ-greedy strategy, possibly with decreasing ϵ to adapt to less exploration requirement in the later part of execution. We will study more advanced adaptive exploration methods later in the course.

Challenge: Instability. When comparing to supervised learning, we may view each tuple (s, a, r, s'), constructed using observed reward and next state on taking action a in state s, as one row in training data; with the loss in the component s, a for parameter θ for this row being $(Q_\theta(s, a) - target(s'))^2$. However, there are several challenges compared to supervised learning. Firstly, this is different from minimizing loss compared to 'true' labels in the training data, as in supervised learning. Here, the **target** $target(s') = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')]$ that plays the role of labels is in fact also an estimate. This can make the learning unstable, as the target label may change too quickly as θ changes. Also, the training data rows are not independent of each other: the "next" row (s', a', r'', s'') depends on the previous row.

Boyan and Moore [1994] provide several examples where Q-learning with function approximation diverges due to these challenges.

There are two main ideas utilized to make the Q-learning stable.

- Batch learning (experience replay): Experience Replay (introduced in Lin [1992]) stores experiences including state transitions, rewards and actions, the sample observations – the necessary data to perform Q learning updates. Then, these experiences are used in mini-batches to update neural networks. This increases speed of update due to batch updates, as well reduces correlation between samples used to update the parameters with decisions made from those updates.

- Lazy update of target network: The target function may change frequently with updates of Q-network parameters. Unstable target function can make training difficult. Lazy update method fixes the parameters of target function and replaces them with the latest Q-network parameters occasionally, every thousands or so steps.

Refer to Mnih et al. [2013] for an interesting (and famous) application of Deep Q networks for superhuman performance on Atari breakout games. The paper discusses practical implications and approaches for handling the challenges mentioned above, in context of lerge-scale deep reinforcement learning.

1.2 TD-learning with function approximation

In TD-learning we are only interested in evaluating a policy given π – you can think of this a special case of Q-learning on an MDP with only one action available for every state. Therefore, we are interested in computing value function $V^\pi(s)$ for all s. In the function approximation version, we learn a parametric approximation $V_\theta(s)$. For example, the function $V_\theta(s,a)$ could simply be a linear function in θ and features $V_\theta(s) = \theta_0 f_0(s) + \theta_1 f_1(s) + \ldots + \theta_n f_n(s)$, or output of a deep neural net. The parameter θ can map the feature vector $f(s)$ for any s to its value-function, thus providing a compact representation. Instead of learning the $|S|$ dimensional value vecor TD-learning algorithm will learn the parameter θ.

Here is a least squares approximation based modification (similar to Q-learning with function approximation) of TD(0) and TD(1) respectively.

Algorithm 1 Approximate TD(0) method for policy evaluation

1: Initialization: Given a starting state distribution D_0, policy π, the method evaluates $V^\pi(s)$ for all states s.
   Initialize θ.
   Initialize episode $e = 0$.
2: **repeat**
3: $e = e + 1$.
4: Set $t = 1, s_1 \sim D_0$. Choose step sizes $\alpha_1, \alpha_2, \ldots$.
5: Perform TD(0) updates over an episode:
6: **repeat**
7: Take action $a_t \sim \pi(s_t)$. Observe reward r_t, and new state s_{t+1}.
8: (*) **Let** $\delta_t := r_t + \gamma \hat{V}_\theta(s_{t+1}) - V_\theta(s_t)$**.**
9: (*) **Update** $\theta \leftarrow \theta + \alpha_t \delta_t \nabla_\theta V_\theta(s_t)$**.**
10: $t = t + 1$
11: **until** until episode terminates
12: **until** termination condition

Recall in $TD(1)$, the target is computed using infinite lookahead as $target(s_{t+1}) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1}, \ldots =: \mathcal{R}_t$. Effectively, the only difference is to replace Step 8 in TD(0) by:

TD(1) algorithm

8: Let $\delta_t := \mathcal{R}_t - V_\theta(s_t)$.

In tabular case, both these algorithms converge to the true value (they are both instances of the stochastic approximation method); we saw that depending on the MDP, one may be more efficient than the other. However, these tradeoffs change when we use function approximation. Below is any example where these methods do converge *but* some times to a function approximation that is far from true value function! The examples show that this can happen even if there exists some value of θ for which the function approximation V_θ is close to V^π.

1.3 Convergence: Counter-example

The following example has been taken from Bertsekas [1995].

Consider an MDP with states indexed by $i = 1, 2, \ldots, n$. Under the actions taken by a fixed policy π, the (deterministic) reward in state i is given by $g_i = 1, i \neq n$, and $g_n = -(n+1)$. Starting state is n, and from state i there is a deterministic transition to $i-1$ under this policy. State 1 is terminal state, the rewards are 0 after state 1. Therefore, (for $\gamma = 1$), $V(i) = i$ for all $i \neq n$, and $V(n) = -2$. Consider linear approximation (single parameter) $V_\theta(i) = \theta i$, where $\theta \in \mathbb{R}$. A desirable approximator in this family is $V_\theta(i) = i$, i.e., $\theta = 1$. For large n, $\theta = 1$ is in fact also close to the least square estimator $\arg\min_\theta \frac{1}{n} \sum_{i=1}^n (V(i) - i\theta)^2$ We show that $TD(0)$-learning converges to $\theta \approx -1$ instead.

For simplicity we consider lazy update of target in TD learning. We update the target only at the end of every episode. Let $\hat{\theta}$ is the parameter value at the beginning of an episode, then the update on observing reward-state

transition $(i, g_i, i-1)$ is given by:

$$\theta \leftarrow \theta + \alpha(g_i + V_\theta(i-1) - V_\theta(i))\frac{\partial V_\theta(i)}{\partial \theta} = \theta + \alpha(g_i + \theta(i-1) - \theta i)i = \theta + \alpha(g_i - \theta)i$$

For simplicity suppose that the updates are applied in batch at the end of every episode: Then, at the end of an episode, θ would be updated as

$$\theta \leftarrow \theta + \alpha \sum_{i=1}^{n} (g_i - \theta)i$$

At convergence the update should converge to 0, i.e., θ would be solution of

$$\sum_{i=1}^{n} (g_i - \theta)i = 0$$

giving

$$\theta = \frac{\sum_{i=1}^{n} g_i i}{\sum_{i=1}^{n} i} = \frac{-(n+1)n + (n-1)n/2}{n(n+1)/2} = -1 - \frac{2}{(n+1)} \approx -1$$

In fact, if one uses TD(1) (full lookhead) instead in this case, the convergence is to the least square estimator: Per step update in that case is given by

$$\begin{aligned} \theta &\leftarrow \theta + \alpha(g_i + g_{i-1} + \cdots + g_1 - V_\theta(i))\frac{\partial V_\theta(i)}{\partial \theta} \\ &= \theta + \alpha(g_i + g_{i-1} + \cdots + g_1 - \theta i)i \end{aligned}$$

So that the batch update at the end of the episode is given by:

$$\theta \leftarrow \theta + \alpha \sum_{i=1}^{n} (g_i + g_{i-1} + \cdots + g_1 - \theta i)i$$

At convergence:

$$\sum_{i=1}^{n} (g_i + g_{i-1} + \cdots + g_1 - \theta i)i = 0$$

giving

$$\theta = \frac{\sum_{i=1}^{n} (g_i + g_{i-1} + \cdots + g_1)i}{\sum_{i=1}^{n} i^2} = \frac{-(n+1)n + \sum_{i=1}^{n} i^2 - n}{\sum_{i=1}^{n} i^2} = 1 - O(1/n) \approx 1$$

2 Fitted Value iteration (Approximate dynamic programming)

Let's take a step back and consider an easier case: the problem of solving a known MDP instead of reinforcement learning. That is, we know the MDP model or can simulate the MDP model in any state and action. If state space was small, we could use value iteration for computing optimal policy. However, to handle large state space, we consider *approximate* value iteration, also known as 'fitted' value iteration. This is an instance of approximate dynamic programming algorithms.

Recall that the tabular value iteration algorithm starts with an initial value vector v^0, and in every iteration i, simply performs the update $v^i = Lv^{i-1}$ (see (1)).

Tabular value-iteration (Recall)

1. Start with an arbitrary initialization $\mathbf{v}^0$. Specify $\epsilon > 0$
2. **Repeat** for $k = 1, 2, \ldots$ **until** $\|\mathbf{v}^k(s) - \mathbf{v}^{k-1}(s)\|_\infty \leq \epsilon$:

- for every $s \in S$, improve the value vector as:

$$\mathbf{v}^k(s) = [Lv^{k-1}](s) := \max_{a \in A} R(s,a) + \gamma \sum_{s'} P(s,a,s') v^{k-1}(s'), \quad (1)$$

In the fitted value iteration, we replace this to a scalable update, by fitting a compact representation V_θ to v_i in every iteration.

Fitted value-iteration

1. Start with an arbitrary initialization θ^0.
2. **Repeat** for $k = 1, 2, 3, \ldots$:
   - Generate a sample subset of states S. For each sampled state s and action a, generate samples of next state and reward to get an estimate $[\hat{L}V_{\theta^{k-1}}](s)$ of $[LV_{\theta^{k-1}}](s)$.
   - Use some regression technique to find a θ to fit data $(V_\theta(s), \hat{L}V_{\theta^{k-1}}(s))_{s \in S_0}$. Set this θ as θ_k.

Example 1: Sampling + Least squares fitting [Munos and Szepesvári, 2008]. (This assumes number of actions is small, number of states is large. And, the MDP model is large but can be simulated for any arbitrary state s and action a).
In iteration k:

1. Sample states $X_1, X_2, \ldots, X_N$ from the state space S, using some distribution ν.
2. For each action a, and state X_i, take multiple samples of next state and rewards $\{Y_{i,a,j}, R_{i,a,j}\}_{j=1,\ldots,M}$.
3. Approximate $[LV_{\theta^{k-1}}](s)$ for $s \in \{X_1, X_2, \ldots, X_N\}$ as

$$\text{target}_i = \max_{a \in A} \frac{1}{M} \sum_{j=1}^{M} \left(R_{i,a,j} + \gamma V_{\theta^{k-1}}(Y_{i,a,j}) \right), \forall i = 1, \ldots, N$$

4. Use least squares new θ as:

$$\theta_k := \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} (V_\theta(i) - \text{target}_i)^2$$

2.1 Convergence: Counter-example [Tsitsiklis and Van Roy, 1996, Bertsekas and Tsitsiklis, 1996]

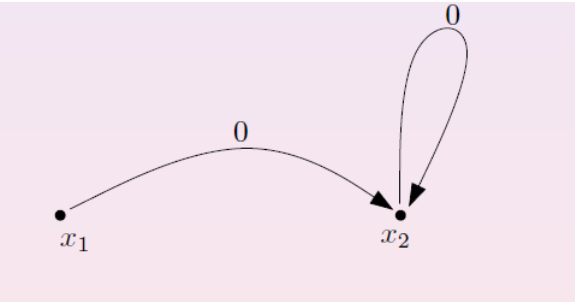
Unfortunately, the above method may not converge for linear function approximation even if infinite number of samples are available. and even if optimal value function is indeed linear. Consider an MDP with two states x_1, x_2, 1-d features for the two states: $f_{x_1} = 1, f_{x_2} = 2$. (It is a trivial setting with single action on all states).

We do a Linear Function Approximation with $V_\theta(x) = \theta f_x$. We want θ to converge to 0.

Suppose we use fitted value iteration with samples collected using uniform distribution ν over states.

$$\begin{aligned}
\theta_k &:= \arg\min_\theta \frac{1}{2} \left((\theta - \text{target}_1)^2 + (2\theta - \text{target}_2)^2 \right) \\
&= \arg\min_\theta \frac{1}{2} \left((\theta - \gamma\theta^{k-1} f_{x_2})^2 + (2\theta - \gamma\theta^{k-1} f_{x_2})^2 \right) \\
&= \arg\min_\theta \frac{1}{2} \left((\theta - \gamma 2\theta^{k-1})^2 + (2\theta - \gamma 2\theta^{k-1})^2 \right) \\
(\theta - \gamma 2\theta^{k-1}) + 2(2\theta - \gamma 2\theta^{k-1}) &= 0 \Rightarrow 5\theta = 6\gamma\theta^{k-1} \\
\theta_k &= \frac{6}{5}\gamma\theta_{k-1}
\end{aligned}$$

This diverges if $\gamma \geq 5/6$. A similar result can be derived for any sampling distribution ν with a constant support on each state.



2.2 Convergence under non-expansive approximations

Operator view of Fitted value-iteration. A more general way to interpret fitted value iteration is that you have an operator M_A that takes a value vector estimate v^i and projects it into the function space formed by functions of form $V_\theta \in \mathcal{F}$.

1. Start with an arbitrary initialization $v^0, V_{\theta^0} := M_A(v^0)$.
2. **Repeat** for $k = 1, 2, 3, \ldots$:
   - $V_{\theta^i} = M_A(L(V_{\theta^{i-1}})$.

Now, to match the description earlier, consider operator M_A defined as follows: Fit a V_θ to $LV_{\theta^{i-1}}$ by comparing its values on a subset S_0 of states, using a regression technique. And, then return this V_θ as new function V_{θ^i} in the output space of M_A. Thus, M_A is effectively an approximation operator.

Equivalently,

1. Start with an arbitrary initialization v^0.
2. **Repeat** for $k = 1, 2, 3, \ldots, K$:
   - $v^i = (L \circ M_A)v^{i-1}$.
3. Output $M_A v^K$

(In an efficient implementation, $u^{i-1} = M_A v^{i-1}$ probably has a more compact representation, so the first view may be better for implementation)

Effectively, in above the value function estimate at the end of iteration i is $v^i = L \circ M_A \circ L \circ M_A \cdots L \circ M_A(v^0)$ The above view allows us to view fitted value iteration as just value iteration with a different operator: $v^i = Lv^{i-1}$ is replaced by $v^i = (L \circ M_A)v^{i-1}$ Therefore, as long as the new operator $(L \circ M_A)$ is also γ-contraction, the results proven earlier for convergence of value iteration will hold.

A sufficient condition is that the operator M_A is non-expansive:

$$\|M_A v - M_A v'\|_\infty \leq \|v - v'\|_\infty$$

Then,

$$\begin{aligned} \|(L \circ M_A)v - (L \circ M_A)v'\|_\infty &\leq \gamma \|M_A v - M_A v'\|_\infty \\ &\leq \gamma \|v - v'\|_\infty \end{aligned}$$

and

$$\begin{aligned} \|(M_A \circ L)V_\theta - (M_A \circ L)V_{\theta'}\|_\infty &\leq \|LV_\theta - LV_{\theta'}\|_\infty \\ &\leq \gamma \|V_\theta - V_{\theta'}\|_\infty \end{aligned}$$

Converges to what The fitted value iteration converges with exponential rate under above condition, but to what? Let the fited iteration converges to V_θ where $V_\theta = M_A L V_\theta$. Is that close to V^*? Indeed:

$$\begin{aligned}
\|V_\theta - V^*\|_\infty &\leq \|V_\theta - M_A V^*\|_\infty + \|M_A V^* - V^*\|_\infty \\
&= \|M_A L V_\theta - M_A L V^*\|_\infty + \|M_A V^* - V^*\|_\infty \\
&\leq \gamma \|V_\theta - V^*\|_\infty + \|M_A V^* - V^*\|_\infty \\
\|V_\theta - V^*\|_\infty &\leq \frac{\gamma}{(1-\gamma)} \|M_A V^* - V^*\|_\infty
\end{aligned}$$

Therefore, if $V^* \in \mathcal{F}$, V_θ converges to V^*.

Non-expansive projections In the fitted value iteration algorithm above, the operator M_A corresponds to the linear regression using least-square loss function over data collected from ν. Therefore the contraction/expansion properties of $L \circ M_A$ depend both on the distribution ν used to collect sample states as well as the function class $\mathcal{F}$ (linear in the case) used to represent the value function. [Gordon, 1995] gave one specific case in which case M_A is always non-expansive.

Sufficient condition for convergence: Averager [Gordon, 1995].:

Lemma 1. *The opertor M_A is non-expansive if it is an averager, that is,*

$$[M_A v^k](s) = \sum_{i \in S} w_{i,s} v^k(i) + w_{0,s}$$

where

$$\sum_i w_{i,s} + w_{0,s} = 1, w_{i,s} \geq 0$$

Proof. This is non-expansive because

$$M_A v = W v + w_0$$

$$\|M_A v - M_A v'\|_\infty = \max_s |\sum_{i \in S} w_{i,s}(v(i) - v'(i))| \leq \max_s \max_i |v(i) - v'(i)| = \|v - v'\|_\infty$$

□

However, this specific case is not widely applicable.

More useful results consider inherent Bellman error of the function class $\mathcal{F} = \{V_\theta, \forall \theta \in \Theta\}$, defined as

$$\epsilon = \sup_{\theta \in \Theta} \inf_{\theta' \in \Theta} \|V_{\theta'} - L V_\theta\|_\infty$$

In other words, measure the worst Bellman error in the function class when considering the best function approximation for each target. Then, define M_A as the finding the closest function approximation for any vector in ∞ norm:

$$M_A v = V_{\theta'}$$

where

$$\theta' = \inf_{\theta \in \Theta} \|V_\theta - v\|_\infty$$

Then, we have

$$\sup_{V_\theta \in \mathcal{F}} \|M_A \circ L V_\theta - L V_\theta\|_\infty \leq \epsilon$$

Intuitively, in this case the contraction property of L holds approximately for $M_A \circ L$. Then, assuming $V^* \in \mathcal{F}$, it can be shown that (exercise)

$$\lim_{k \to \infty} \|V_{\theta_k} - V^*\|_\infty \leq \frac{\epsilon}{1-\gamma}$$

3 Discussion

What next? So far the methods we have seen are based on the value function approach, where all the function approximation effort went into scalable estimation of value function. The action selection policy was represented implicitly as greedy policy with respect to the estimated value functions. This approach has several limitations. First, it is oriented toward finding deterministic policies, whereas the optimal policy is often stochastic, selecting different actions with specific probabilities. Second, an arbitrarily small change in the estimated value of an action can cause it to be, or not be, selected. Such discontinuous changes have been identified as a key obstacle to establishing convergence assurances for algorithms following the value-function approach (Bertsekas and Tsitsiklis, 1996). For example, Q-learning, Sarsa, and dynamic programming methods have all been shown unable to converge to any policy for simple MDPs and simple function approximators (Gordon, 1995, 1996; Baird, 1995; Tsitsiklis and van Roy, 1996; Bertsekas and Tsitsiklis, 1996). This can occur even if the best approximation is found at each step before changing the policy, and whether the notion of "best" is in the mean-squared-error sense or the slightly different senses of residual-gradient, temporal-difference, and dynamic-programming methods.

Next, we will look at policy gradient methods that use function approximation for directly approximating the optimal policy and eliminate some of the above limitations.

References

Dimitri P. Bertsekas. A counterexample to temporal differences learning. *Neural Computation*, 7(2):270–279, 1995.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, page 369–376, Cambridge, MA, USA, 1994. MIT Press.

Geoffrey J. Gordon. Stable function approximation in dynamic programming. In *Machine Learning Proceedings 1995*, pages 261 – 268. Morgan Kaufmann, San Francisco (CA), 1995.

Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Pittsburgh, PA, USA, 1992. UMI Order No. GAX93-22750.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

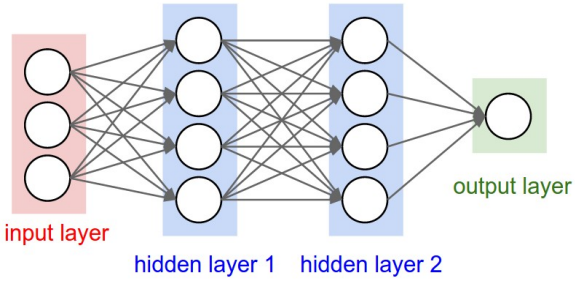
Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *J. Mach. Learn. Res.*, 9:815–857, June 2008. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1390681.1390708`.

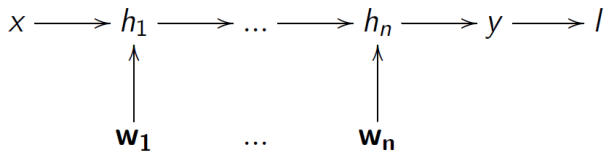
John N. Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1):59–94, Mar 1996.

Appendix

Deep Q-learning Networks (DQN)

Deep Q-learning Networks (DQN) use deep neural network for function approximation, with θ being the parameters of the neural network.





Architecture A deep representation is composed of many functions, typically linear transformations alternated by non-linear activation functions: $h_1 = W_1 x, h_2 = \sigma(h_1), \ldots, h_{k+1} = W_k h_k, h_{k+2} = \sigma(h_{k+1}),$, where $\sigma(\cdot)$ is a nonlinear function. Two popular functions are sigmoid and Relu.

The parameters pf this network are $\theta = (\mathbf{W}_1, \ldots, \mathbf{W}_n)$. The parameter θ maps any given input x to output $f_\theta(x) = g(h_n)$, where $g(\cdot)$ is a 'scoring function'. The scoring function could yield a scalar prediction $\hat{y} = g(h_n)$, or scores/probabilities over different possible predictions/labels. For example, softmax scoring function maps $h_n \in \mathbb{R}^k$ to probabilities $\hat{p} = g(h_n) = [p_1, \ldots, p_k]$, where $\hat{p}_i = \frac{e^{h_n[i]}}{(e^{h_n[1]} + \ldots + e^{h_n[k]})}$. Then, this score is evaluated against target prediction y^* using a loss function $\ell(\cdot, y^*)$. Examples are squared loss function, common in regression $\ell(\hat{y}, y^*) = (\hat{y} - y^*)^2$ for evaluating a prediction against a given target, and log-likelihood for evaluating probability scores $\ell(\hat{p}, y^*) = -\log(\hat{p}_{y^*})$ common in multi-label classification. A neural network is trained using labeled data (x_i, y_i) to learn parameters $\theta = (\mathbf{W}_1, \ldots, \mathbf{W}_n)$ that minimize loss $\sum_i \ell(f_\theta(x_i), y_i)$.

In DQN, the input x is formed by the feature vector for (s_t, a_t) pairs, and for any given θ the output $f_\theta(x)$ is a prediction for $Q_\theta(s_t, a_t)$. However, to evaluate these prediction we do not directly have a target prediction/label y^*. Instead, we bootstrap to use current Q-network (i.e., current value of parameters θ) to generate a target as: $r_t + \gamma \max_a Q_\theta(s_{t+1}, a)$. Then, to update θ we use a gradient descent step for squared loss function.

The second form of output (probability scores) will be more relevant in policy gradient methods, discussed later in the course. There, we will use deep neural networks to learn "policies", i.e., probability scores for different actions in a given state.

Backpropagation for gradient computation The independent-layer structure of deep neural network allows the gradient computations efficiently through **backpropagation**. (In the reinforcement learning context) Backpropagation refers to simply a computational implementation of the chain rule : an algorithm that determines the relevant partial derivatives by means of the backward pass.

Suppose $Q_\theta(s, a) = f_\theta(\mathbf{x}(s, a))$ is represented by a deep neural network with $K - 1$ hidden layers and weights $\theta = (W_1, \ldots, W_K)$. Its input $\mathbf{x}$ is representation of a state action pair s, a, and output

$$\hat{y} = f_\theta(\mathbf{x}) = g(h^K) = g(W^K(\sigma(\cdots W^3 \sigma(W^2 \cdot \sigma(W^1 \mathbf{x})))))$$

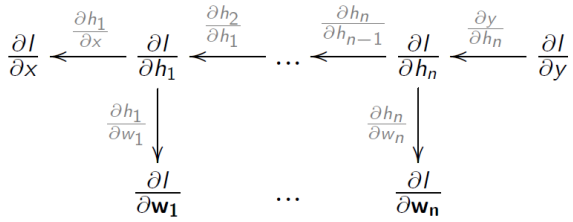
is a representation of action Q-values $Q_\theta(s, a)$. Given, a target observation $y = target(s')$, we are interested in updating θ by computing gradient of loss function

$$\ell_\theta(\hat{y}, y) = (\hat{y} - y)^2 = (g(h^K) - y)^2$$

where

$$h^k = \sigma(z^k), z^k = W^k h^{k-1}$$
$$h^{k-1} = \sigma(z^{k-1}), z^{k-1} = W^{k-1} h^{k-2},$$

$$\dots,$$
$$h^1 = \sigma(z^1), z^1 = W^1\mathbf{x}$$



Now, gradient with respect of ℓ_θ with respect to parameter W^ℓ_{ab} is:

$$\frac{\partial \ell_\theta(\cdot)}{\partial W^\ell_{ab}} = 2(g(h^K) - y)\nabla g(h^K)\left[\frac{\partial h^K}{\partial W^\ell_{ab}}\right]_r$$

Then, basic observation is as follows. For a neuron r in layer k, $\ell < k$:

$$\begin{aligned}\frac{\partial h^k_r}{\partial W^\ell_{ab}} &= \sigma'(z^k_r)\frac{\partial z^k_r}{\partial W^\ell_{ab}} \\ &= \sigma'(z^k_r)\frac{\partial W^k_r h^{k-1}}{\partial W^\ell_{ab}} \\ &= \sigma'(z^k_r)\sum_{i\in\text{parents}(r)} W^k_{ri}\frac{\partial h^{k-1}_i}{\partial W^\ell_{ab}}\end{aligned}$$

Thus, the gradients can be back propagated over the network, until we reach layer ℓ where, from layer structure of the neural network:

$$\begin{aligned}\frac{\partial z^\ell_a}{\partial W^\ell_{ab}} &= \frac{\partial \sum_i W^\ell_{ai} h^{\ell-1}_i}{\partial W^{\ell-1}_{ab}} \\ &= h^{\ell-1}_b + \sum_{i\neq b} W^\ell_{ai}\frac{\partial h^{\ell-1}_i}{\partial W^\ell_{ab}} \\ &= h^{\ell-1}_b\end{aligned}$$

as layer $h^{\ell-1}$ is independent of layer ℓ parameters; so that

$$\frac{\partial h^\ell_i}{\partial W^\ell_{ab}} = \begin{cases}\sigma'(z^\ell_a)h^{\ell-1}_b, & \text{if } i = a \\ 0 & \text{otherwise}\end{cases}$$