

# 732A96/TDDE15 Advanced Machine Learning

## Reinforcement Learning

Jose M. Peña  
IDA, Linköping University, Sweden

Lectures 9: REINFORCE and Deep Q-Learning Algorithms

# Contents

- ▶ REINFORCE
- ▶ Neural Networks
- ▶ Example: Grid Worlds
- ▶ Deep Q-Learning
- ▶ Example: CartPole

# Literature

- ▶ Main source

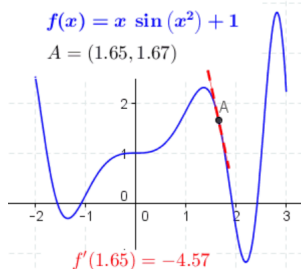
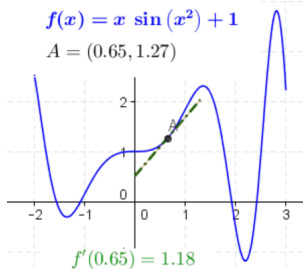
- ▶ Sutton, R. S. and Barto, A. G. [Reinforcement Learning: An Introduction](#). The MIT Press, 2018. Chapters 13 and 16.
- ▶ Mnih, V. et al. [Playing Atari with Deep Reinforcement Learning](#). *NIPS Deep Learning Workshop*, 2013.

- ▶ Additional source

- ▶ Russel, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson, 2010. Chapters 16, 17 and 21.
- ▶ Mnih, V. et al. [Human-level Control through Deep Reinforcement Learning](#). *Nature* 518, 529–533, 2015.

# REINFORCE

- ▶ Consider learning a **parameterized policy**  $\pi(a|s, \theta)$  to select actions without consulting state or action values.
- ▶ Advantages:
  - ▶ The policy function may be simpler to learn than the state and action value functions.
  - ▶ Possibility to inject prior knowledge through the parameterization chosen.
  - ▶ Possibility to model deterministic and stochastic policies. The latter may be interesting in problems with imperfect knowledge, e.g. bluffing in poker. Learning is also easier as the gradient is smoother.
- ▶ Choose the parameter values that maximize the following function for **episodic** tasks:  $J(\theta) = v_{\pi_\theta}(S_0)$  where  $S_0$  is the initial state. It can also be adapted to continuing tasks.
- ▶ Stochastic **gradient ascent**:  $\theta^{i+1} \leftarrow \theta^i + \alpha \nabla_{\theta^i} J(\theta^i)$ .



# REINFORCE

- Theorem:

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} v_{\pi_{\theta}}(S_0) \propto E_{\pi_{\theta}} \left[ \sum_a \gamma^t q_{\pi}(S_t, a) \nabla_{\theta} \pi(a|S_t, \theta) \right] \text{ at time } t$$

and thus

$$\begin{aligned} \nabla_{\theta} J(\theta) &\propto E_{\pi_{\theta}} \left[ \sum_a \pi(a|S_t, \theta) \gamma^t q_{\pi}(S_t, a) \frac{\nabla_{\theta} \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \right] = E_{\pi_{\theta}} \left[ \gamma^t q_{\pi}(S_t, A_t) \frac{\nabla_{\theta} \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \\ &= E_{\pi_{\theta}} \left[ \gamma^t G_t \frac{\nabla_{\theta} \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] = E_{\pi_{\theta}} [\gamma^t G_t \nabla_{\theta} \ln \pi(A_t|S_t, \theta)]. \end{aligned}$$

- All this results in the REINFORCE algorithm, which asymptotically converges to a local optimum of  $J(\theta)$ .

## REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for $\pi_*$

Input: a differentiable policy parameterization  $\pi(a|s, \theta)$

Algorithm parameter: step size  $\alpha > 0$

Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$  (e.g., to  $\mathbf{0}$ )

Loop forever (for each episode):

Generate an episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , following  $\pi(\cdot|\cdot, \theta)$

Loop for each step of the episode  $t = 0, 1, \dots, T-1$ :

$$\begin{aligned} G &\leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \\ \theta &\leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \theta) \end{aligned} \tag{G_t}$$

# REINFORCE

- ▶ REINFORCE has one major disadvantage: It updates the policy only **at the end of each episode** and, thus, it is not suitable for incremental online learning and may show slow convergence and high variance.
- ▶ A particularly interesting advantage of REINFORCE is that updating the policy parameters may change the policy for every action-state pair, and not only for the pairs visited, i.e. **generalization**.
- ▶ Another interesting feature of REINFORCE is that it can be applied to **continuous state and action spaces** by choosing an appropriate parametric policy model, e.g.  $\mathcal{N}(\mu s, \sigma)$ :

$$\pi(a|s, \theta) = \pi(a|s, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(a - \mu s)^2}{2\sigma^2}\right)$$

which implies that

$$\nabla_{\mu} \ln \pi(a|s, \mu, \sigma) = \frac{a - \mu s}{\sigma^2} s$$

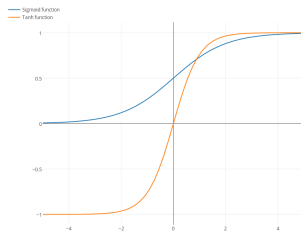
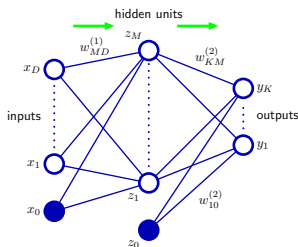
and

$$\nabla_{\sigma} \ln \pi(a|s, \mu, \sigma) = \left(\frac{(a - \mu s)^2}{\sigma^2} - 1\right) s$$

and, thus, the REINFORCE update has a simple form.

- ▶ For discrete state and action spaces, we can easily do better than this as we show next.

# Neural Networks



- ▶ Consider a multiclass classification problem, i.e.  $C \in \{1, 2, \dots, K\}$ .
- ▶ Activations:  $a_j = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}$  for all  $j = 1, \dots, M$ .
- ▶ Hidden units and activation function:  $z_j = h(a_j)$  for all  $j = 1, \dots, M$ .
- ▶ Output activations:  $a_k = \sum_{j=1}^M w_{kj}^{(2)} z_j + w_{k0}^{(2)}$  for all  $k = 1, \dots, K$ .
- ▶ Output :  $y_k(\mathbf{x}) = a_k$  for all  $k = 1, \dots, K$ .
- ▶ Two-layer NN:  $y_k(\mathbf{x}) = \sum_{j=1}^M w_{kj}^{(2)} h\left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}$ .
- ▶ Class probabilities:  $p(C = k|\mathbf{x}) = \frac{\exp(y_k(\mathbf{x}))}{\sum_{k=1}^K \exp(y_k(\mathbf{x}))}$ .
- ▶ All the previous is generalizable to several hidden layers.

## Neural Networks

- ▶ Consider some training data  $\{(\mathbf{x}_n, c_n)\}$  of size  $N$ . Consider minimizing the cross-entropy or negative log-likelihood loss function:

$$L(\mathbf{w}) = \sum_{n=1}^N L_n(\mathbf{w}) = - \sum_{n=1}^N \log p(c_n | \mathbf{x}_n, \mathbf{w}).$$

- ▶ The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- ▶ Stochastic gradient descent:

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \alpha \nabla_{\mathbf{w}^i} L_n(\mathbf{w}^i) = \mathbf{w}^i + \alpha \nabla_{\mathbf{w}^i} \log p(c_n | \mathbf{x}_n, \mathbf{w}^i)$$

where  $n$  is chosen randomly, and  $\nabla_{\mathbf{w}^i} L_n(\mathbf{w}^i)$  can be computed efficiently via **backpropagation**.

- ▶ Note the similarities between the update above and the REINFORCE update:

$$\theta^{i+1} = \theta^i + \alpha \gamma^t G \nabla_{\theta^i} \ln \pi(A_t | S_t, \theta^i).$$

- ▶ Therefore, REINFORCE is straightforward to implement when a NN is used to represent the parameterized policy. We only need to set the sample weights  $\gamma^t G$ . In other words, the policy can be seen as a classifier whose training involves certain peculiarities: Incremental online, and **delayed labels** via sample weights.



# Example: Grid Worlds



- ▶ Now, consider a unidimensional regression problem. Consider also some training data  $\{(\mathbf{x}_n, c_n)\}$  of size  $N$ . Consider minimizing the mean squared-error loss function:

$$L(\mathbf{w}) = \sum_{n=1}^N L_n(\mathbf{w}) = \sum_{n=1}^N \frac{1}{2} (c_n - y(\mathbf{x}_n))^2.$$

- ▶ The above implies the following stochastic gradient descent update:

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \alpha \nabla_{\mathbf{w}^i} L_n(\mathbf{w}^i) = \mathbf{w}^i + \alpha (c_n - y(\mathbf{x}_n)) \nabla_{\mathbf{w}^i} y(\mathbf{x}_n)$$

where  $n$  is chosen randomly, and  $\nabla_{\mathbf{w}^i} y(\mathbf{x}_n)$  can be computed efficiently via **backpropagation**.

## Deep Q-Learning

- ▶ If the transition model were so that  $s$  and  $a$  are always followed by  $s'$  and  $r$ , then  $q_*(s, a) = r + \gamma \max_{a'} q_*(s', a')$  by the Bellman optimality equation and, thus,  $0 = r + \gamma \max_{a'} q_*(s', a') - q_*(s, a)$ . We can try to enforce this constraint by executing  $\pi$  one step from  $s$  and  $a$  and, then, updating the estimate of  $q_*(s, a)$  as

$$q_*(s, a) \leftarrow q_*(s, a) + \alpha(r + \gamma \max_{a'} q_*(s', a') - q_*(s, a)).$$

where  $\alpha > 0$  is the learning rate.

- ▶ The reasoning above also applies to stochastic transition models, since the number of times that  $s$  and  $a$  are followed by  $s'$  and  $r$  in the sampled episodes is proportional to the transition probability.

### Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

        Take action  $A$ , observe  $R, S'$

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

    until  $S$  is terminal

- ▶ Q-learning has a major disadvantage: Representing the action value function  $q(s, a)$  as a look-up table is not feasible for **large state and/or action spaces**, due to storage space and time to convergence.
- ▶ A solution is to represent it as a **parameterized function**  $q(s, a|\theta)$ . This brings advantages such as fewer parameters than table entries and thus easier to learn and reach convergence, less storage space, **generalization to unvisited state-action pairs**, etc.
- ▶ It also comes with the challenge of selecting an appropriate parameterized function: It has to be expressive and allow for incremental online training. A NN is typically used.

## Deep Q-Learning

- ▶ For the  $i$ -th iteration (i.e., episode step), the NN is trained by minimizing the mean squared-error loss function:

$$L(\theta^i) = E_{s,a \sim \rho(s,a)} [(y^i - q(s, a|\theta^i))^2]$$

where  $\rho(s, a)$  is the behaviour distribution induced by  $\epsilon$ -greedy, and

$$y^i = E_{s' \sim p(s'|s,a)} [r + \gamma \max_{a'} q(s', a'|\theta^{i-1}) | s, a].$$

- ▶ The gradient is given by

$$\nabla_{\theta^i} L(\theta^i) = -E_{s,a \sim \rho(s,a); s' \sim p(s'|s,a)} [(r + \gamma \max_{a'} q(s', a'|\theta^{i-1}) - q(s, a|\theta^i)) \nabla_{\theta^i} q(s, a|\theta^i)]$$

and stochastic gradient descent is typically considered:

$$\theta^{i+1} = \theta^i + \alpha (r + \gamma \max_{a'} q(s', a'|\theta^{i-1}) - q(s, a|\theta^i)) \nabla_{\theta^i} q(s, a|\theta^i).$$

- ▶ Note the similarities between the update above and the NN update:

$$\mathbf{w}^{i+1} = \mathbf{w}^i + \alpha (c_n - y(\mathbf{x}_n)) \nabla_{\mathbf{w}^i} y(\mathbf{x}_n)$$

which implies that deep Q-learning is straightforward to implement when a NN is used to represent the parameterized state value function. In other words, the action value function can be seen as a regressor whose training involves certain peculiarities: Incremental online, and **delayed targets** via bootstrapping on the model from the previous iteration.

### Deep Q-learning with experience replay for estimating $\pi \approx \pi_*$

Algorithm parameters: small  $\epsilon > 0$

Initialize  $Q(S, A|\theta)$  with random weights, and initialize  $\mathcal{D}$  with  $N$  random experiences

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

        Take action  $A$ , observed  $R, S'$

        Store transition  $(S, A, R, S')$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(S_j, A_j, R_j, S'_j)$  from  $\mathcal{D}$

$$Y_j \leftarrow \begin{cases} R_j & \text{for terminal } S'_j \\ R_j + \gamma \max_a Q(S'_j, a|\theta) & \text{for non-terminal } S'_j \end{cases}$$

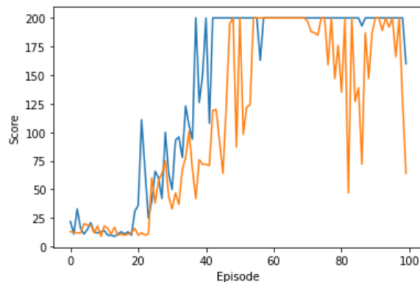
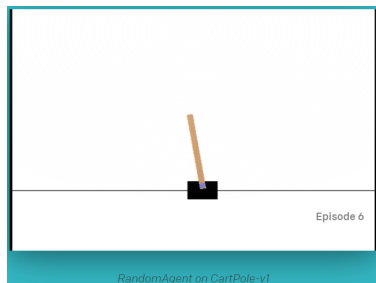
        Perform a gradient descent step on  $(Y_j - Q(S_j, A_j|\theta))^2$

$S \leftarrow S'$

    until  $S'$  is terminal

- ▶ Learning a parameterized state value function may resemble supervised learning but the learning data is **not i.i.d.** due to highly correlated states and changing agent behaviour. Experience replay alleviates these problems by smoothing the training distribution over many past behaviours.
- ▶ All action values at once:  $S \rightarrow NN \rightarrow (Q(S, A_1), \dots, Q(S, A_k))$ .

## Example: CartPole



- ▶ Open AI's gym
- ▶ PilcoLearner

# Summary

- ▶ REINFORCE
- ▶ Neural Networks
- ▶ Example: Grid Worlds
- ▶ Deep Q-Learning
- ▶ Example: CartPole
- ▶ Interested in more ? Check out [AlphaGo - The Movie](#).

Thank you