Homework 5

Deadline: Monday, April 6, at 11:59pm.

Submission: You must submit your solutions as a PDF file through MarkUs[1]. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.

See the syllabus on the course website[2] for detailed policies. You may ask questions about the assignment on Piazza[3]. *Note that 10% of the homework mark may be removed for a lack of neatness.*

Homework written by Harris Chan and Silviu Pitis.
mailto:csc413-2020-01-tas@cs.toronto.edu

[1] https://markus.teach.cs.toronto.edu/csc413-2020-01
[2] https://csc413-2020.github.io/assets/misc/syllabus.pdf
[3] https://piazza.com/class/k58ktbdnt0h1wx?cid=1

1 Reversible Architectures

Reversible architectures are based on a reversible block. In this section, we will investigate a variant for implementing reversible block with affine coupling layers.

Consider the following reversible affline coupling block:

$$\mathbf{y}_1 = \exp(\mathcal{G}(\mathbf{x}_2)) \circ \mathbf{x}_1 + \mathcal{F}(\mathbf{x}_2)$$
$$\mathbf{y}_2 = \exp(\mathbf{s}) \circ \mathbf{x}_2,$$

where $\circ$ denotes element-wise multiplication. The each inputs $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{d/2}$. The functions $\mathcal{F}, \mathcal{G}$ maps from $\mathbb{R}^{d/2} \to \mathbb{R}^{d/2}$. This modified block is identical to the ordinary reversible block, except that the inputs $\mathbf{x}_1$ and $\mathbf{x}_2$ are multiplied element-wise by vectors $\exp(\mathcal{F}(\mathbf{x}_2))$ and $\exp(\mathbf{s})$, where $\circ$ is the element-wise product.

You don't need to justify your answers for this question, but doing so may help you receive partial credit.

1.1 Inverting reversible block [1pt]

As a warm up, give equations for inverting this block, i.e. computing $\mathbf{x}_1$ and $\mathbf{x}_2$ from $\mathbf{y}_1$ and $\mathbf{y}_2$. You may use / to denote element-wise division.

1.2 Jacobian of the reversible block [1pt]

Give a formula for the Jacobian $\partial \mathbf{y} / \partial \mathbf{x}$, where $\mathbf{y}$ denotes the concatenation of $\mathbf{y}_1$ and $\mathbf{y}_2$. You may denote the solution as a block matrix, as long as you clearly define what the matrix for each block corresponds to.

Note: You may leave the derivative of $\mathcal{G}$ and $\mathcal{F}$ with respect to $\mathbf{x}_2$ as $\frac{\partial \mathcal{G}}{\partial \mathbf{x}_2}$ and $\frac{\partial \mathcal{F}}{\partial \mathbf{x}_2}$

1.3 Determinant of Jacobian [1pt]

Give a formula for the determinant of the Jacobian from previous part, i.e. compute $\det\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)$.

2 Policy gradients and black box optimization

Very often we have a function f that does not give us useful gradient information: input or output may be discrete; f may be piecewise constant, nowhere differentiable, or have pathological gradients (e.g., a discontinuous saw wave on an incline, whose gradient always points away from the global optimum); or f may be a black box that we cannot backpropagate through. For example, we may have a phone app that labels photos as cats or dogs. This situation is the default in Reinforcement Learning (RL), where we can execute the environment dynamics, but we cannot see or control their internals.

We still, however, want to optimize some score function $J[f] : X \to \mathbb{R}$. For example, in RL, we want to learn a policy that maximizes the non-differentiable environment reward.

When using the REINFORCE strategy, we replaced the θ optimization task with a Monte-Carlo approximation. One of the key factors for a successful REINFORCE application is the *variance*. The higher the variance, the more "noisy" the gradient estimates will be, which can slow down the optimization process. In this section we will derive the variance of the REINFORCE estimator for a simple toy task.

Consider a loss function, $f(\tilde{a})$ which is the zero-one loss of the logistic regression output, $p(a|\theta)$. The input vector has D *independent* scalar features, x_d. We evaluate the performance of the classifier by sampling from the output of the sigmoid μ. The loss function $J(\theta)$ can be written as:

$$\mu = \sigma\left(\sum_{d=1}^{D} \theta_d x_d\right), \tag{1}$$

$$p(a|\theta) = Bernoulli(\mu) = \begin{cases} \mu & a = 1 \\ 1 - \mu & a = 0 \end{cases}, \tag{2}$$

$$\tilde{a} \sim p(a|\theta), \tag{3}$$

$$f(\tilde{a}) = \begin{cases} 1 & \tilde{a} = 1 \\ 0 & \tilde{a} = 0 \end{cases}, \tag{4}$$

$$J(\theta) = \mathbb{E}_{\tilde{a} \sim p(a|\theta)}[f(\tilde{a})]. \tag{5}$$

2.1 Closed form expression for REINFORCE estimator [1pt]

Recall from above that the expression for REINFORCE estimator is:

$$\nabla_\theta J[\theta] = \mathbb{E}_{\tilde{a} \sim p(a|\theta)}\left[f(\tilde{a}) \frac{\partial}{\partial \theta} \log p(a = \tilde{a}|\theta)\right] \tag{6}$$

We can denote the expression inside the expectation as $g[\theta, \mathbf{x}]$:

$$g[\theta, \tilde{a}] = f(\tilde{a}) \frac{\partial}{\partial \theta} \log p(a = \tilde{a}|\theta), \quad \tilde{a} \sim p(a|\theta) \tag{7}$$

For this question, derive a closed form for the $g[\theta, \tilde{a}]$ as a deterministic function of $\tilde{a}$, μ, θ, and x_d.
Hint: Substitute in the log likelihood of the Bernoulli distribution.

2.2 Variance of REINFORCE estimator [1pt]

We will derive the variance of the REINFORCE estimator above. Since the gradient is is D-dimensional, the covariance of the gradients will be $D \times D$ matrix. In this question, we will only consider the variance with respect to the first parameter, i.e. $\mathbb{V}[\hat{g}[\theta, , \tilde{a}]_1]$ which scalar value corresponding to the first element in the diagonal of the covariance matrix. Derive the variance of the gradient estimator as a function of the first parameter vector: $\mathbb{V}[\hat{g}[\theta, , \tilde{a}]_1]$, as a function of μ, θ, and x_d.

Hint: The second moment of a Bernoulli random variable is $\mu(1 - \mu)$.

2.3 Variance of REINFORCE estimator with baselines [1pt]

Comment on the variance in Part 2.2. When do we expect learning to converge slowly in terms of the output of the logistic regression model, μ?

3 Approximate dynamic programming

Q-learning and Markov Decision Process are powerful tools to solve dynamic programming problems using neural networks. For this question we will study a simpler setup to understand the properties of these approximate dynamic programming algorithm. Consider a finite-state *Markov Reward Process* (MRP) induced by behavior policy π, which is described by a tuple (S, P^π, R, γ), where $s \in S$ is a state, $P^\pi(s, s')$ is the probability of transitioning from state s into state s' ($\sum_{s'} P^\pi(s, s') = 1$), $R(s)$ is the reward in state s, and $\gamma \in [0, 1)$ is a discount factor. We define the *value* $V^\pi(s)$ of state s under policy π as the expected sum of discounted rewards when starting in that state and following transition model P^π ad infinitum: $V^\pi(s) = \mathbb{E}_{s_t|P^\pi, s_0=s} \sum_{t=0} \gamma^t R(s_t)$.

By evaluating $V^\pi(s)$ we are doing what is called *policy valuation*, which is a critical component of many RL algorithms. For instance, in *policy iteration*, we interleave policy valuation steps with policy improvement steps, and in *Q-learning*, we are evaluating the greedy policy with respect to the current value function.

To simplify things we will assume that we know the transition model P^π (or at least, can compute dynamic programming estimates as if it were known), and consider solving this problem using synchronous 1-step dynamic programming or Bellman updates. Letting $\boldsymbol{v}^\pi$ be the true value vector, whose i-th entry is $V^\pi(s_i)$, $\boldsymbol{P^\pi}$ be the transition matrix, whose (i, j)th entry is $P^\pi(s_i, s_j)$, and $\boldsymbol{r}$ be the reward vector whose i-th entry is $R(s_i)$. We have the following dynamic programming relation (the *Bellman equation*):

$$\boldsymbol{v}^\pi = \boldsymbol{T}^\pi \boldsymbol{v}^\pi \triangleq \boldsymbol{r} + \gamma \boldsymbol{P}^\pi \boldsymbol{v}^\pi, \quad \text{(why?)}$$

where $\boldsymbol{T}^\pi$ is known as the "Bellman operator". Note that $\boldsymbol{v}^\pi$ is the only vector $\boldsymbol{v} \in \mathbb{R}^{|S|}$ that satisfies the Bellman relation, since:

$$\boldsymbol{v}^\pi + \boldsymbol{\epsilon} = \boldsymbol{T}^\pi(\boldsymbol{v}^\pi + \boldsymbol{\epsilon}) \tag{8}$$

$$\iff \boldsymbol{v}^\pi = \boldsymbol{r} + \gamma \boldsymbol{P}^\pi(\boldsymbol{v}^\pi + \boldsymbol{\epsilon}) - \boldsymbol{\epsilon} \tag{9}$$

$$\iff \gamma \boldsymbol{P}^\pi \boldsymbol{\epsilon} - \boldsymbol{\epsilon} = 0 \tag{10}$$

$$\iff \boldsymbol{\epsilon} = 0 \quad \text{(why?)} \tag{11}$$

To get a better understanding of the Bellman updates, let us consider a linear regression model to approximate the value function:

$$\hat{\boldsymbol{v}}^\pi = \boldsymbol{X}\boldsymbol{w} \approx \boldsymbol{v}^\pi. \tag{12}$$

where we represent the states using feature matrix $\boldsymbol{X} \in \mathbb{R}^{|S| \times d}$ whose ith row is the d-dimensional feature vector for the ith state, and model the value function as $\hat{\boldsymbol{v}}^\pi = \boldsymbol{X}\boldsymbol{w} \approx \boldsymbol{v}^\pi$.

In this case (and also when using neural networks to approximate value functions), $\boldsymbol{T}^\pi \boldsymbol{X}\boldsymbol{w}$ may not be representable as $\boldsymbol{X}\boldsymbol{w'}$ for any $\boldsymbol{w'}$. Thus, approximate dynamic programming must *project* $\boldsymbol{T}^\pi \boldsymbol{X}\boldsymbol{w}$ back into the rowspace of $\boldsymbol{X}$. This is done using an additional projection operator $\boldsymbol{\Pi}$, so that the Bellman operator becomes $\boldsymbol{\Pi T}^\pi$. Unfortunately, the projected Bellman operator does not guarantee convergence, as we will see in Part 3.2 below.

3.1 Approximate policy valuation [1pt]

The usual synchronous (update all states) policy valuation (dynamic programming) update is:

$$\boldsymbol{v}_{t+1} \leftarrow \boldsymbol{v}_t + \alpha(\boldsymbol{r} + \gamma \boldsymbol{P}^\pi \boldsymbol{v}_t - \boldsymbol{v}_t) \tag{13}$$

for some learning rate $\alpha \in (0, 1)$. As noted above Eq. 11, policy valuation converges.

When using a linear model to approximate the current value function $\hat{\boldsymbol{v}} = \boldsymbol{X}\boldsymbol{w}$, the approximate policy evaluation becomes minimizing the following objective function:

$$J(\boldsymbol{w}_t) = \frac{1}{2} \| \boldsymbol{T}^\pi \boldsymbol{X} \overline{\boldsymbol{w}_t} - \boldsymbol{X}\boldsymbol{w}_t \|_2^2, \tag{14}$$

where $\overline{\boldsymbol{w}}_t$ in (14) is a copy of $\boldsymbol{w}_t$ through which gradients do not flow (i.e., $\overline{\boldsymbol{w}}_t = \boldsymbol{w}_t$ but $\nabla_{\boldsymbol{w}_t} \overline{\boldsymbol{w}}_t = 0$).

Derive the gradient descent update rule for the weights $\boldsymbol{w_{t+1}} \leftarrow \boldsymbol{w_t} - \alpha \nabla_{\boldsymbol{w}_t} J$ in terms of γ, $\boldsymbol{P}^\pi$, $\boldsymbol{X}$, $\overline{\boldsymbol{w}}$ and learning rate α.

3.2 Divergence of approximate policy valuation [0pt]

In the previous Part we set $\boldsymbol{X} = \boldsymbol{I}$. In approximate RL and DP using , we use neural networks and parameterized models to enable generalization. Unfortunately, this can cause problems. The following is a minimal example to show that dynamic programming with linear function approximation can diverge. It is inspired by "Tsitsiklis and Van Roy's Counterexample" [2]. For a more classic (but also more complicated) example of divergence, see "Baird's counterexample" [1].

Consider approximate MRP with:

$$\boldsymbol{X} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \boldsymbol{P} = \begin{bmatrix} \epsilon & 1-\epsilon \\ \epsilon & 1-\epsilon \end{bmatrix}, \text{ and } \boldsymbol{r} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Here there are two states and $d = 1$ so we have parameter vector $\boldsymbol{w} \in \mathbb{R}^1$, and since $\boldsymbol{r} = 0$, it is clear that we have optimal $\boldsymbol{w}^* = \boldsymbol{0}$ for all possible γ. We use the same loss as above: $J(\boldsymbol{w}_t) = \frac{1}{2}\|\boldsymbol{r} + \gamma \boldsymbol{P}\boldsymbol{X}\overline{\boldsymbol{w}_t} - \boldsymbol{X}\boldsymbol{w}_t\|_2^2$.

Derive an expression for $\boldsymbol{w}_{t+1}$ in terms of $\boldsymbol{w}_t$, the learning rate $\alpha \in (0, 1)$, $\gamma \in (0, 1)$ and $\epsilon \in (0, 1)$—your answer should be in the form $\boldsymbol{w}_{t+1} = A\boldsymbol{w}_t$, where A is a function of α, γ, and ϵ.

Find one setting of α, γ, and ϵ, for which repeat updates will diverge (there are many) given any initialization $\boldsymbol{w}_0 \neq 0$.

Is $\boldsymbol{\Pi}\boldsymbol{T}^\pi$ a contraction in this case? (Note: since we are running one step of gradient descent rather than doing a full projection on the column space of $\boldsymbol{X}$, $\boldsymbol{\Pi}$ isn't a minimum norm projection. It turns out, however, that solving for the minimum norm solution here also results in divergence.)

References

[1] Baird, L. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[2] Tsitsiklis, J. N., and Van Roy, B. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems* (1997), pp. 1075–1081.