Runqiu Ye
Stanford CS229
Problem Set #4
08/07/2024

Problem Set #3: EM, Deep Learning, & Reinforcement Learning

Problem 1 Neural Networks: MNIST image classification
Implement a simple convolutional neural network to classify grayscale images of handwritten digits from the MNIST dataset. The starter code splits the set of 60000 training images and labels into a sets of 59600 examples as the training set and 400 examples for the dev set. To start, implement convolutional neural network and cross entropy loss, and train it with the provided dataset. The architecture is as follows:

(a) The first layer is a convolutional layer with 2 output channels with a convolution size of 4 by 4.

(b) The second layer is a max pooling layer of stride and width 5 by 5.

(c) The third layer is a ReLU activation layer.

(d) After the four layer, the data is flattened into a single dimension.

(e) The fifth layer is a single linear layer with output size 10 (the number of classes).

(f) The sixth layer is a softmax layer that computes the probabilities for each classes.

(g) Finally, we use a cross entropy loss as our loss function.

The cross entropy loss is

$$CE(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k,$$

where $\hat{y} \in \mathbb{R}^K$ is the vector of softmax outputs from the model for the training example x, and $y \in \mathbb{R}^K$ is the ground-truth vector for the training example X such that $y = [0, \ldots, 0, 1, 0, \ldots, 0]^T$ contains a single 1 at the position of the correct classes.
We also use mini-batch gradient descent with a batch size of 16. Normally we would iterate over the data multiple times with multiple epochs, but for this assignment we only do 400 batches to save time.

(a) Implement functions within `p01_nn.py`.

(b) Implement a function that computes the full backward pass. See training curve in `src/output/train.png`

■

Problem 2 Off Policy Evaluatino And Causal Inference

Need methods for evaluating policies without actually implementing them. This task is off-policy evaluation or causal inference.

For this problem, consider MDPs with a single timestep. Consider universe of states S, actions A, a reward function $R(s, a)$ where s is a state and a is an action. We often only have a subset of a in our dataset. For eaxmple, each state s could represent a patient, each action a could represent which drug we prescribe to that patient and $R(s, a)$ be their lifespan after prescribing that drug.

A policy is defined as $\pi_i(s, a) = p(a \mid s, \pi_i)$ We are given observational dataset of $(s, a, R(s, a))$ tuples. Let $p(s)$ be the density for the distribution of the state s values wihin the dataset. Let $\pi_0(s, a) = p(a \mid s)$ within our observational data. π_0 corresponds to the baseline policy in observational data. Also given target policy $\pi_1(s, a)$ which gives the conditional probability $p(a \mid s)$ in our optimal policy. Our goal is to computer expected value of $R(s, a)$ in the same population as our observatinoal data, but with a policy of π_1 instead of π_0. In other words, we are trying to compute

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)] = \sum_{(s,a)} R(s,a)p(s,a) = \sum_{(s,a)} R(s,a)p(s)p(a \mid s) = \sum_{(s,a)} R(s,a)p(s)\pi_1(s,a).$$

Simplifying Assumptions: We will assume that each action has a non-zero probability in the observed policy $\pi_0(s, a)$. In other words, for all actions a and states s, $\pi_0(s, a) > 0$.

Regression: The simplest possible estimator is to directly use our learned MDP parameters to estimate our goal. This is called regression estimator. While training our MDP, we learn an estimator $\hat{R}(s, a)$ that estimates $R(s, a)$. We can now directly estimate

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)]$$

with

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[\hat{R}(s,a)].$$

If $\hat{R}(s, a) = R(s, a)$, then this estimator is trivially correct. We will now consider alternative approaches and explore why you might use one estimator over another.

(a) **Importance Sampling:** Let $\hat{\pi}_0$ be an estimate of the true π_0. The *importance sampling estimator* uses that $\hat{\pi}_0$ and has the form

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)\right].$$

We now show that if $\hat{\pi}_0 = \pi_0$, then the importance sampling estimator is equal to

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)].$$

Proof. If $\hat{\pi}_0 = \pi_0$, then

$$\begin{aligned}\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)\right] &= \sum_{(s,a)}\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)p(s)\pi_0(s,a)\\ &= \sum_{(s,a)}\pi_1(s,a)R(s,a)p(s)\\ &= \mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)],\end{aligned}$$

as desired. □

Note that this estimator only requires us to model π_0 as we have the $R(s,a)$ values in the observational data.

(b) **Weighted Importance Sampling.** A variant of the importance sampling estimator is the *weighted importance sampling estimator*. The weighted importance sampling estimator has the from

$$\frac{\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)\right]}{\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}\right]}.$$

We now show that if $\hat{\pi}_0 = \pi_0$, then the weighted importance sampling estimator is equal to

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)].$$

Proof. We already showed that if $\hat{\pi}_0 = \pi_0$, then

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)\right] = \mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)].$$

For the denominator, if $\hat{\pi}_0 = \pi_0$, we have

$$\begin{aligned}\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}\right] &= \sum_{(s,a)}\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}p(s)\pi_0(s,a)\\ &= \sum_{(s,a)}\pi_1(s,a)p(s)\\ &= \sum_{(s,a)}p(s,a\mid\pi_1)\\ &= 1.\end{aligned}$$

Hence,

$$\frac{\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)\right]}{\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}\right]} = \mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)]$$

when $\hat{\pi}_0 = \pi_0$, as desired. □

(c) One issue with the weighted importance sampling estimator is that it can be biased in many finite smaple situations. In finite samples, we replace the expected value with a sum over the seen values in our observational dataset. Please show that the weighted importance sampling estimator is biased in these situations.

Hint: Consider the case where there is only a single data element in the observational dataset.

Proof. When there is only a single data element $(s^*, a^*, R(s^*, a^*))$ in the observational dataset, we have

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)\right] = \frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s^*,a^*),$$

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}\right] = \frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}.$$

The weighted importance estimator then gives

$$\frac{\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)\right]}{\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}\right]} = R(s^*,a^*) = \mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}[R(s,a)].$$

If $\pi_1 \neq \pi_0$, that is, if pi_1 has nonzero probability taking any action $a \neq a^*$ at state s^*, then the weighted importance sampling estimator

$$\frac{\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}R(s,a)\right]}{\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}\right]} = \mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}[R(s,a)] \neq \mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)],$$

causing a bias. □

(d) **Doulby Robust.** One final commonly used estimator is the doubly robust estimator. The doubly robust estimator has the form

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\mathop{\mathbb{E}}_{a\sim\pi_1(s,a)}\left[\hat{R}(s,a)\right] + \frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}(R(s,a) - \hat{R}(s,a))\right].$$

One advantage of the doubly robust estimator is that it works if either $\hat{\pi}_0 = \pi_0$ or $\hat{R}(s,a) = R(s,a)$.

i. First we show that the doubly robust estimator is equal to $\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)]$ when $\hat{\pi}_0 = \pi_0$.

Proof. First of all, we have

$$\mathbb{E}_{a\sim\pi_1(s,a)}[\hat{R}(s,a)] = \sum_a \pi_1(s,a)\hat{R}(s,a).$$

Note that this expression is independent of a. Hence,

$$\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\mathbb{E}_{a\sim\pi_1(s,a)}\left[\hat{R}(s,a)\right]\right] = \sum_s\sum_a \pi_1(s,a)\hat{R}(s,a)p(s) = \sum_{(s,a)} \pi_1(s,a)\hat{R}(s,a)p(s).$$

For the second term, when $\hat{\pi}_0 = \pi_0$, we have

$$\begin{aligned}
\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}(R(s,a) - \hat{R}(s,a)) &= \sum_{(s,a)}\frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}(R(s,a) - \hat{R}(s,a))p(s)\pi_0(s,a)\\
&= \sum_{(s,a)} \pi_1(s,a)R(s,a)p(s) - \pi_1(s,a)\hat{R}(s,a)p(s)\\
&= \mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)] - \sum_{(s,a)} \pi_(s,a)\hat{R}(s,a)p(s).
\end{aligned}$$

Therefore, the doubly robust estimator

$$\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\mathbb{E}_{a\sim\pi_1(s,a)}\left[\hat{R}(s,a)\right] + \frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}(R(s,a) - \hat{R}(s,a))\right] = \mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)].$$

when $\hat{\pi}_0 = \pi_0$. □

ii. Now we show the the doubly robust estimator is equal to $\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)]$ when $\hat{R}(s,a) = R(s,a)$.

Proof. As in the previous proof, we have

$$\mathbb{E}_{a\sim\pi_1(s,a)}[\hat{R}(s,a)] = \sum_a \pi_1(s,a)\hat{R}(s,a).$$

When $\hat{R}(s,a) = R(s,a)$,

$$\mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\mathbb{E}_{a\sim\pi_1(s,a)}\left[\hat{R}(s,a)\right]\right] = \sum_s\sum_a \pi_1(s,a)R(s,a)p(s) = \mathbb{E}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)].$$

The second term clearly vanishes when $\hat{R}(s,a) = R(s,a)$. Hence, the doubly robust estimator

$$\mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_0(s,a)}}\left[\mathop{\mathbb{E}}_{a\sim\pi_1(s,a)}\left[\hat{R}(s,a)\right] + \frac{\pi_1(s,a)}{\hat{\pi}_0(s,a)}(R(s,a) - \hat{R}(s,a))\right] = \mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_1(s,a)}}[R(s,a)].$$

when $\hat{R}(s,a) = R(s,a)$. □

(e) In some situations, we can choose between the importance sampling estimator and the regression estimator. Which one is better ni the following situatios? In all of these situations, the states s consist of patients, actions a represent the drug to give to certain patients and $R(s,a)$ represent the lifespan of the patient after receiving the drug.

i. Drugs are randomly assigned to patients, but the interaction between the drug, patient, and lifespan is very complicated.

In this case, importance sampling estimator is better, since it is easier to model an estimation $\hat{\pi}_0$ of π_0.

ii. Drugs are assigned to patients in a very complicated manner, but the interaction between the drug, patient and lifespan is very simple.

In this case, the regression estimator is better, since it is easier to learn an estimator $\hat{R}(s,a)$ that estimates $R(s,a)$.

■

Problem 3 PCA

In class, we showed that PCA finds the "variance maximizing" directions onto which to project the data. In this problem, we find another interpretation of PCA.

Suppose set of points $\{x^{(i)}\}_{i=1}^m$. Let us assume that we have as usual preprocessed the data to have zero mean and unit variance in each coordinate. For a given unit-length vector u, let $f_u(x)$ be the projection of point x onto the direction given by u. That is, if $\mathcal{V} = \{\alpha u : \alpha \in \mathbb{R}\}$, then

$$f_u(x) = \operatorname*{argmin}_{v \in \mathcal{V}} \|x - v\|^2.$$

Show that the unit-length vector i that minimizes the mean squared error between the projected points and original points corresponds to the first principle component of the data. That is, show that

$$\operatorname*{argmin}_{u:\|u\|=1} \sum_{i=1}^{m} \left\|x^{(i)} - f_u(x^{(i)})\right\|_2^2$$

gives the first principle component.

Proof. Let u be a vector with unit length. When $\|x - v\|^2$ attains its minimium value, we have $v = (x^T u)u$. Therefore

$$f_u(x) = \operatorname*{argmin}_{vin\mathcal{V}} \|x - v\|^2 = (x^T u)u.$$

For each x, we have

$$\|x - f_u(x)\|_2^2 = \left\|x - (x^T u)u\right\|_2^2 = x^T x - u^T x x^T u$$

It follows that

$$\sum_{i=1}^{m} \left\|x^{(i)} - f_u(x^{(i)})\right\|_2^2 = \sum_{i=1}^{m} (x^{(i)})^T x^{(i)} - u^T x^{(i)} (x^{(i)})^T u.$$

Hence,

$$\operatorname*{argmin}_{\|u\|=1} \sum_{i=1}^{m} \left\|x^{(i)} - f_u(x^{(i)})\right\|_2^2 = \operatorname*{argmax}_{\|u\|=1} u^T \left(\sum_{i=1}^{m} x^{(i)} (x^{(i)})^T\right) u.$$

This is exactly the same equation as in "variance maximizing", so this gives the first principal component. □

■

Problem 4 Independent Component Analysis

In this problem, understand why Gaussian distribution sources are a problme for ICA. We will also derive ICA with Laplace distribution and apply it to the cocktail party problem. Let $s \in \mathbb{R}^d$ be source data generated from d independent sources. Let $x \in \mathbb{R}^d$ be observed data such that $x = As$, where $A \in \mathbb{R}^{d \times d}$ is *mixing matrix*. Assume A invertible and $W = A^{-1}$ is called the *unmixing matrix*. It follows that $s = Wx$. The goal of ICA is to estimate W. Denote w_j^T to be the j-th row of W. Note that this implies that the j-th source can be reconstructed with w_j and x since $s_j = w_j^T x$. Given training set $\{x^{(i)}\}_{i=1}^m$ for the following sub-questions. Denote the training set by design matrix $X \in \mathbb{R}^{n \times d}$.

(a) **Gaussian source**

Assume sources are distributed according to a standard normal distribution, i.e. $s_j \sim \mathcal{N}(0, 1)$. The likelihood of our unmixing matrix is

$$\ell(W) = \sum_{i=1}^{n} \left(\log |W| + \sum_{j=1}^{d} \log g'\big(w_j^T x^{(i)}\big) \right),$$

where g is the CDF and g' is the PDF of the source distribution. Because the sources are Gaussian distribution, we can analytically reason about the resulting W.

Try derive a closed form expression for W interms of X when g is the standard normal CDF. Deduce the relation between W and X in the simplest terms and highlight the ambiguity (in terms of rotational invariance) in computing W.

To maximize the log-likelihood, we take the gradient of ℓ with respect to W and set it to 0. For the gradient, we have

$$\nabla_W \ell(W) = \sum_{i=1}^{n} (W^T)^{-1} + \begin{bmatrix} \frac{g''}{g'}(w_1^T x^{(i)}) \\ \vdots \\ \frac{g''}{g'}(w_d^T x^{(i)}) \end{bmatrix} (x^{(i)})^T.$$

Since $s_j \sim \mathcal{N}(0, 1)$, we have

$$g'(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}, \quad g''(z) = \frac{-z}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}.$$

It follows that

$$\frac{g''}{g'}(z) = -z.$$

The gradient then becomes

$$\nabla_W \ell(W) = \sum_{i=1}^{n} (W^T)^{-1} - W x^{(i)} (x^{(i)})^T = n(W^T)^{-1} - \sum_{i=1}^{n} W x^{(i)} (x^{(i)})^T.$$

Set the gradient to zero and use the design matrix, we have

$$n(W^T)^{-1} = WX^TX.$$

It follows that

$$W^TW = n(X^TX)^{-1}.$$

This solution is ambiguious. To be specific, consider a rotation given by the orthogonal matrix U. Now W becomes UW and

$$(UW)^T(UW) = W^TU^TUW = W^TW = n(X^TX)^{-1}.$$

Here, we can see that if W is a solution, any rotation of W is also a solution. In other words, the rotational invariance causes the unmixing matrix here to be ambigious.

(b) **Laplace source**

Assume sources are distributed according to a standard Laplace distribution. That is, $s_i \sim \mathcal{L}(0,1)$. The Laplace distribution $\mathcal{L}(0,1)$ has PDF $f(s) = \frac{1}{2}\exp(-|s|)$. With this assumption, derive the update rule for a single example.

To derive the update rule, we again consider the gradient of the log-likelihood

$$\nabla_W \ell(W) = \sum_{i=1}^{n} (W^T)^{-1} + \begin{bmatrix} \frac{g''}{g'}(w_1^T x^{(i)}) \\ \vdots \\ \frac{g''}{g'}(w_d^T x^{(i)}) \end{bmatrix} (x^{(i)})^T,$$

where g is the CDF for the distribution of the source. Here, g is the CDF for standard Laplace distribution. For the stochastic gradient ascent update, we calculate

$$g'(s) = f(s) = \frac{1}{2}e^{-|s|}, \quad g''(s) = f'(s) = -\frac{1}{2}\operatorname{sgn}(s)e^{-|s|}.$$

It follows that

$$\frac{g''}{g'}(s) = -\operatorname{sgn}(s).$$

Hence, the update rule is

$$W := W + \alpha \left((W^T)^{-1} - \begin{bmatrix} \operatorname{sgn}(w_1^T x^{(i)}) \\ \vdots \\ \operatorname{sgn}(w_d^T x^{(i)}) \end{bmatrix} (x^{(i)})^T \right).$$

(c) **Cocktail Party Problem**

Implement the Bel and Sejnowski ICA algorithm, but assuming a Laplace source, as derived in (b), in `p04_ica.py`. We can *anneal* the learning rate α (slowly decrease it

over time) to speed up laerning. In addition, we can also choose a random permutation of the training data, and running stochastic gradient ascent visiting the training data in that order (each of the specified learning rates was then used for one full pass through the data).

■

Problem 5 Markov Decision Processes

Consider an MDP with finite state and action spaces, and discount factor $\gamma < 1$. Let B be the Bellman update operator with V a vector of values for each state. That is, if $V' = B(V)$, then

$$V'(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s')V(s').$$

(a) Prove that for any two finite-valued vectors V_1, V_2, it holds true that

$$\|B(V_1) - B(V_2)\|_\infty \leq \gamma \|V_1 - V_2\|_\infty,$$

where

$$\|V\|_\infty = \max_{s \in S} |V(s)|.$$

This shows that the Bellman update operator is a γ-contraction in the max-norm.

Proof. For any two finite-valued vectors V_1, V_2 and state s, we have

$$B(V_1)(s) = R(s) + \gamma \max_a \sum_{s'} P_{sa}(s')V_1(s'),$$
$$B(V_2)(s) = R(s) + \gamma \max_a \sum_{s'} P_{sa}(s')V_2(s').$$

Consider the max-norm in state space S. By properties of norm, we have

$$\begin{aligned}\max_a \sum_{s'} P_{sa}(s')V_1(s') - \max_a \sum_{s'} P_{sa}(s')V_2(s') &\leq \max_a \sum_{s'} P_{sa}(s')(V_1(s') - V_2(s')) \\ &\leq \max_a \sum_{s'} P_{sa}(s')\|V_1(s') - V_2(s')\|_\infty \\ &= \|V_1(s') - V_2(s')\|_\infty.\end{aligned}$$

It follows that

$$\|B(V_1) - B(V_2)\|_\infty \leq \gamma \|V_1 - V_2\|_\infty,$$

as desried. □

(b) We say V is a *fixed point* of B if $B(V) = V$. Prove that B has at most one-fixed point. That is, there is at most one solution to the Bellman equations.

Proof. Suppose V_1 and V_2 are both fixed points of B, we want to show $\|V_1 - V_2\|_\infty = 0$. Since B is a *gamma*-contraction, we have

$$\|B(V_1) - B(V_2)\|_\infty \leq \gamma \|V_1 - V_2\|_\infty.$$

Since V_1 and V_2 are both fixed points of B, this means

$$\|V_1 - V_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty.$$

That is,

$$(1-\gamma)\|V_1 - V_2\|_\infty \leq 0.$$

As $\gamma < 1$, $\|V_1 - V_2\|_\infty = 0$. This means that B has at most one fixed point. □

■

Remark: The result shows that value iteraction converges geometrically to the optimal value function V^*. That is, after k iterations, the distance between V and V^* is at most γ^k. △

Problem 6 Reinforcement Learning: The inverted pendulum

Consider the inverted pendulum problem. The objective is to develop a controller to balance the pole with these constraints, by appropriately having the cart accelerate left and right. The state of the cart and pole at any time is completely characterized by 4 parameters: x, $\dot{x}$, θ, $\dot{\theta}$. We approximate the state space by a discretization that maps a state vector $(x, \dot{x}, \theta, \dot{\theta})$ into a numebr from 0 to `NUM_STATES - 1`.

At each time step, the controller must choose one of two actions - push the cart right, or push the cart left. (There is no do-nothing action.) These are represented as actions 0 and 1 in the code.

Assume reward function $R(s)$ is a current state only. When the pole angle goes beyond a certain limit or when the cart goes too far out, a negative reward is given and the system is reinitialized randomly. At all other time, the reward is zero. The program must learn to balance the pole using only the state transitions and rewards observed.

We will estimate a model (i.e. transition probabilities and rewards) for the underlying MDP, solve Bellman's equations for this estimated MDP to obtain a value functino, and act greedily with respect to this value function. Breifly, we will maintain current model of the MDP and a current estimate of the value function. Initially, each state has estimated reward zero, and the estimated transition probabilities are uniform.

Assume that the whole leraning procedure has converged once several consecutive attempts (defined by the parameters `NO_LAERNING_THRESHOLD`) to solve solve Bellman's equation all converge in the first iteraction. Intuitively, this indicates that the estimated model has stopped changing significantly.

(a) How many trials did it take before the algorithm convergeed?

(b) Plot a learning curve showing the number of time-steps for which the pole was balanced on each trial.

(c) Rerun the code with `np.random.seed` set to 1, 2, and 3. What do you observe? What does this imply about the algorithm?

■