Basic RL.4

Judy Tutorial

Recall Q-learning:

Value Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

$$\pi(a|s) \leftarrow \mathrm{argmax}_a Q(s,a)$$

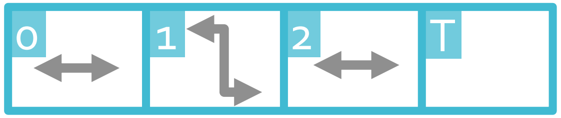
Policy Update for sampling (s,a,r,s')

Problems w/ Value-based methods

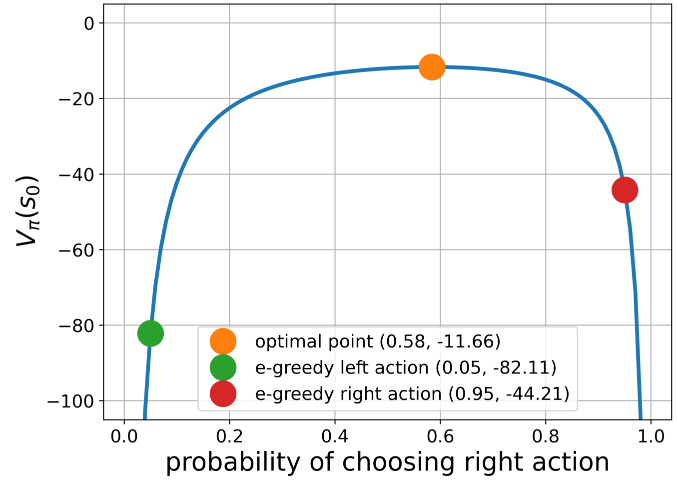
- If state and action spaces are large:

$\pi(a|s) \leftarrow \mathrm{argmax}_a Q(s,a)$ becomes impossible

- There is no natural way for value-based method to find a stochastic optimal policy



-1 reward at all states



A parameterized policy:

$$\pi(a|s;\boldsymbol{\theta})$$

parameters

updated using gradient ascent:

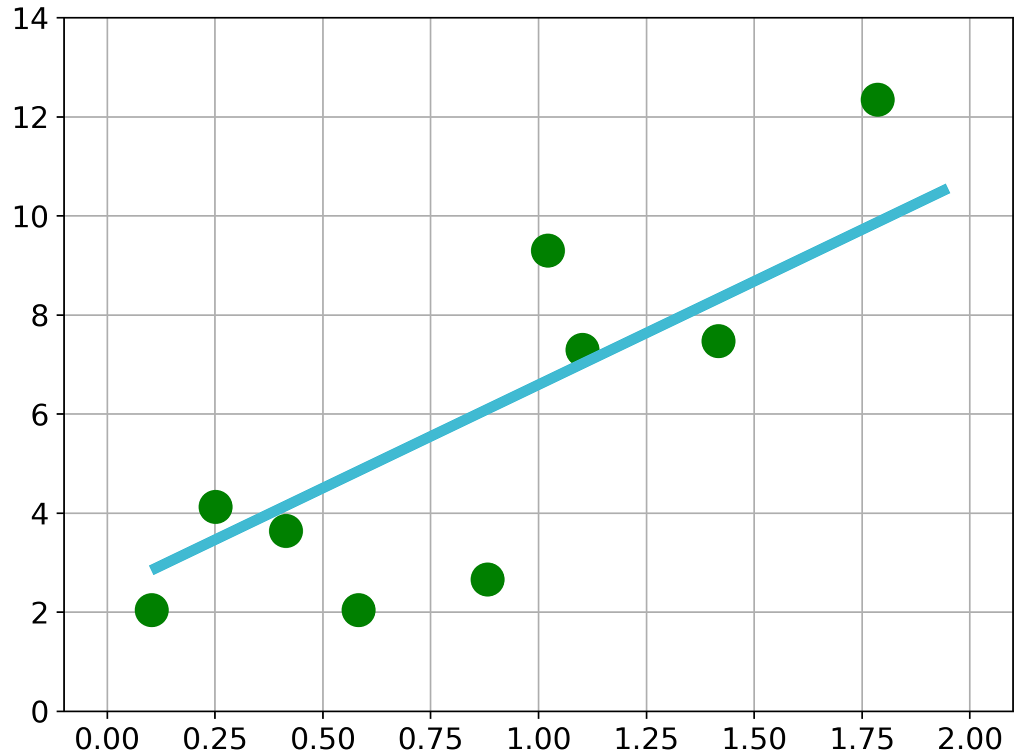
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \nabla \widehat{J(\boldsymbol{\theta}_t)}$$

where $J(\boldsymbol{\theta})$ is a scalar performance measure

To understand what are $J(\boldsymbol{\theta})$ and its **gradient**

Let's start with some basic concepts and examples in machine learning

Linear Regression



- Model the pattern of the dots
- Use the model to predict any new dots

Generalizability!

Element-wise

Hypothesis function

$$h(\boldsymbol{\theta}) = \theta_1 x + \theta_0 = \boxed{\boldsymbol{\theta}^{\mathrm{T}} X}$$

inputs

dot product

parameters/weights

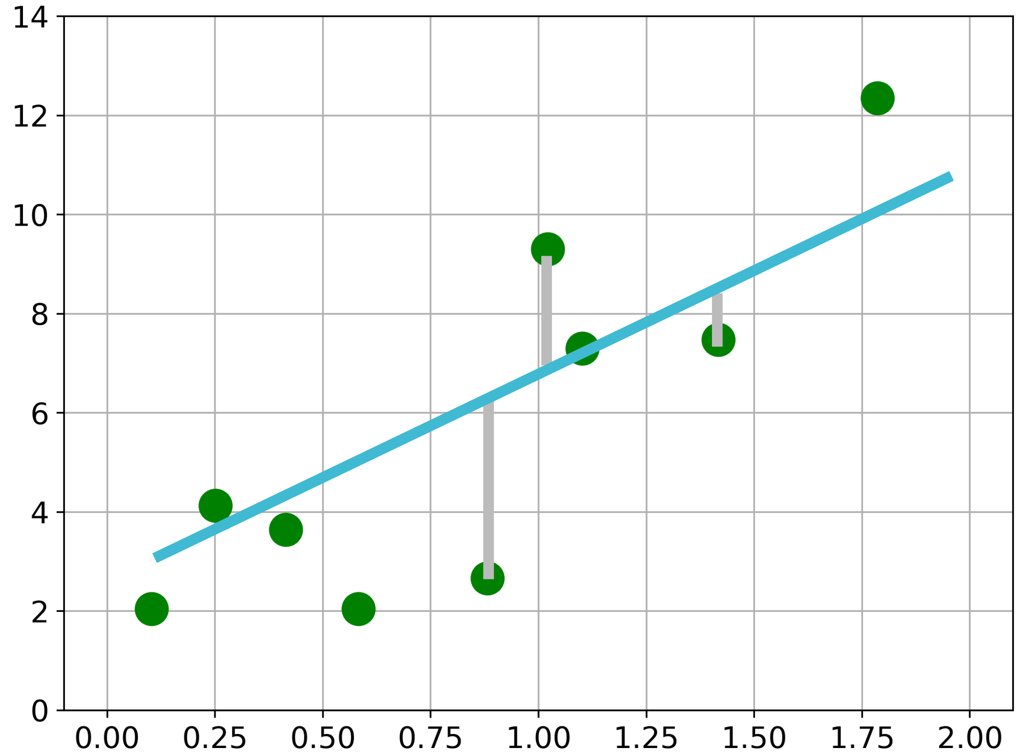
$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}, X = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

Objective function

$$J(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} \left[h(\boldsymbol{\theta})^i - y^i \right]^2$$

x	y
(0.103 ,	2.048)
(0.251 ,	4.13)
(0.414 ,	3.646)
(0.582 ,	2.043)
(0.882 ,	2.66)
(1.022 ,	9.298)
(1.102 ,	7.297)
(1.416 ,	7.472)
(1.786 ,	12.347)
(1.793 ,	14.444)

Objective



$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}\big[\underbrace{h(\boldsymbol{\theta})^i - y^i}_{\text{error}}\big]^2$$

Vector-wise

Hypothesis function

$$\boldsymbol{h}(\boldsymbol{\theta})_{1\times 10} = \boldsymbol{\theta}^{T}_{2\times 1}\boldsymbol{X}_{2\times 10}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \qquad \boldsymbol{X}^T = \begin{bmatrix} 1.0 , 0.103 \\ 1.0 , 0.251 \\ 1.0 , 0.414 \\ 1.0 , 0.582 \\ 1.0 , 0.882 \\ 1.0 , 1.022 \\ 1.0 , 1.102 \\ 1.0 , 1.416 \\ 1.0 , 1.786 \\ 1.0 , 1.793 \end{bmatrix}$$

Objective function

$$J(\boldsymbol{\theta}) = \frac{1}{2N}\|\boldsymbol{h}_{1\times 10} - \boldsymbol{Y}_{1\times 10}\|^2$$

$$\boldsymbol{Y}^T = \begin{bmatrix} 2.048 \\ 4.13 \\ 3.646 \\ 2.043 \\ 2.66 \\ 9.298 \\ 7.297 \\ 7.472 \\ 12.347 \\ 14.444 \end{bmatrix}$$

Objective function

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}\left[h(\boldsymbol{\theta})^i - y^i\right]^2 = \frac{1}{2N}\|\boldsymbol{h}(\boldsymbol{\theta}) - \boldsymbol{Y}\|^2$$

Gradient

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \frac{1}{2N}\sum_{i=1}^{N}\left[h(\boldsymbol{\theta})^i - y^i\right]^2$$

Element-wise

$$= \frac{1}{2N} \cdot 2 \sum_{i=1}^{N}\left[h(\boldsymbol{\theta})^i - y^i\right]\nabla_{\boldsymbol{\theta}} h(\boldsymbol{\theta})^i$$

Need to take gradient w.r.t $\boldsymbol{\theta}_j$ one by one

$$= \frac{1}{N}\boldsymbol{X}[\boldsymbol{h}(\boldsymbol{\theta}) - \boldsymbol{Y}]^T$$

Vector-wise

Gradient Descent

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

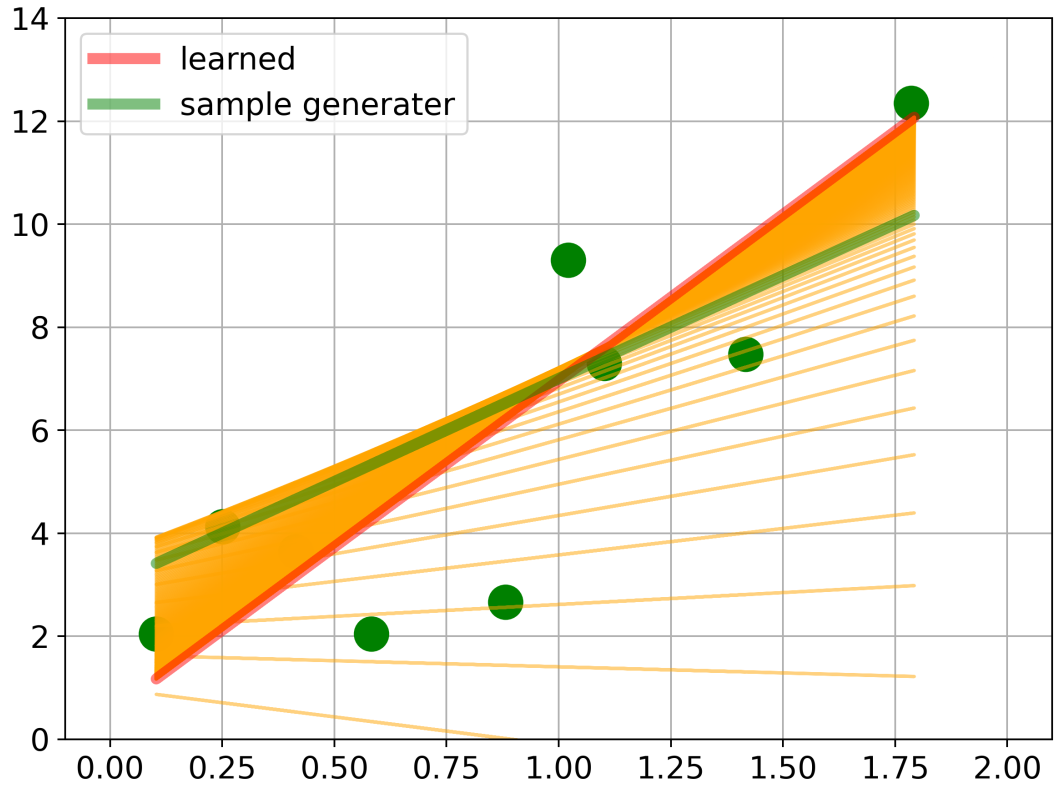
in python

```
np.random.seed(3)
x=2*np.random.rand(10)
y=3+4*x+np.random.randn(10)*2.5
X=np.ones((2,10))
X[1,:]=x

theta=np.random.randn(2,1)
lr=0.01
N=400

for i in range(N):

    h=np.dot(theta.T,X)
    grad=np.dot(X,(h-y.reshape(1,10)).T)
    theta=theta-lr*grad
```

X - (2,10)
theta - (2,1)
h=theta.T*X - (1,10)<-(1,2)(2,10)
grad=X*(h-Y).T - (2,1)<-(2,10)(10,1)
theta-=lr*grad - (2,1)

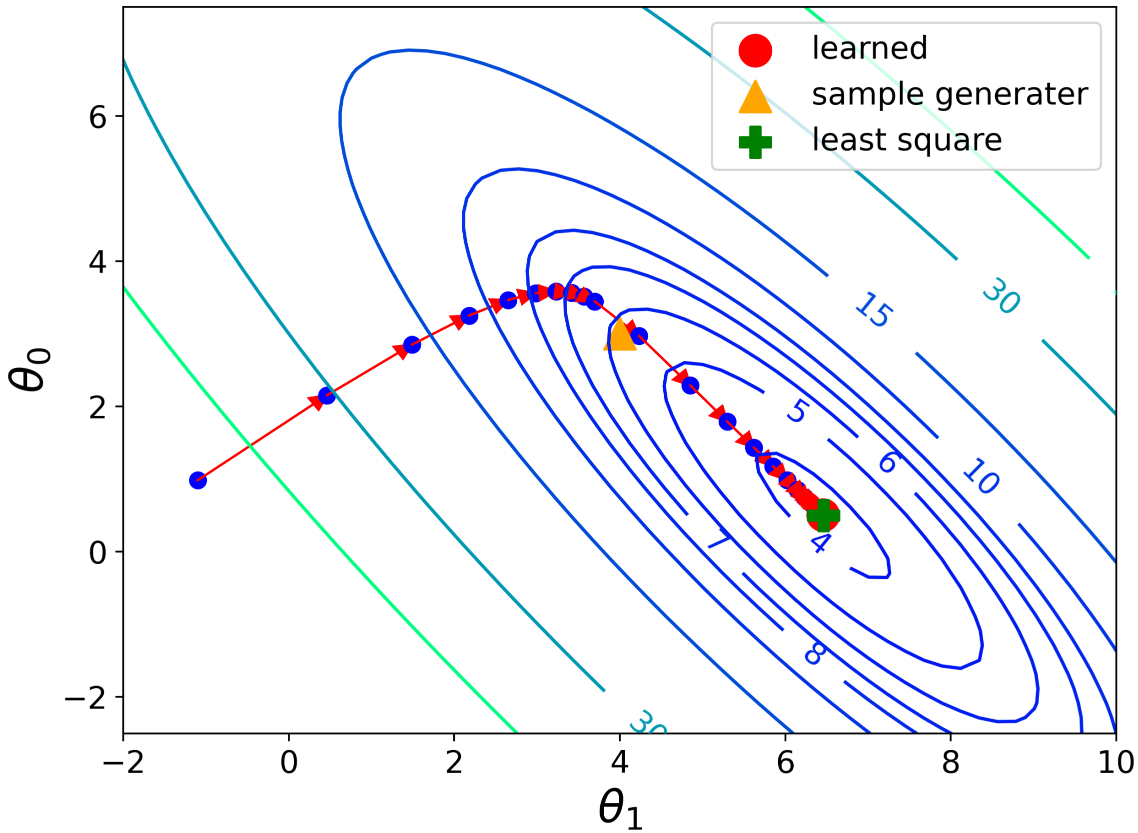
Data shape is important in implementation!

learning behavior



$$\boldsymbol{\theta}_0 = \begin{bmatrix} 0.982 \\ -1.1 \end{bmatrix} \rightarrow \boldsymbol{\theta}_f = \begin{bmatrix} 0.505 \\ 6.455 \end{bmatrix} \quad ? \quad \boldsymbol{\theta}_f ? = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Gradient is the direction of steepest descent



We can also directly calculate $\boldsymbol{\theta} = \left(\boldsymbol{X}\boldsymbol{X}^T\right)^{-1}\boldsymbol{X}\boldsymbol{Y}^T$ using least square method

Back to our policy-based RL

$$\boldsymbol{J(\theta)}, \pi(a|s;\boldsymbol{\theta}), \widehat{\nabla \boldsymbol{J}(\boldsymbol{\theta_t})}$$

How do we define these items?

in episodic case

Assume: every episode starts in some particular non-random state s_0

$$\boldsymbol{J(\theta)} \triangleq V_{\pi_{\boldsymbol{\theta}}}(s_0)$$

Recall: $V_\pi(s_t) \triangleq \mathbb{E}_\pi[R_t | s_t = s]$

Policy Gradient Theorem

$$\mu(s) \geq 0, \sum_{s \in \mathcal{S}} \mu(s) = 1$$

state distribution

$$\nabla J(\theta) \propto \sum_{s} \mu(s) \sum_{a} Q_{\pi}(s, a) \nabla_{\theta} \pi(a|s; \theta)$$

$$= \mathbb{E}_{\pi} \left[\sum_{a} Q_{\pi}(s_t, a) \nabla_{\theta} \pi(a|s_t; \theta) \right]$$

If we follow the policy π,
we can sample $s_t \sim \pi$

REINFORCE
[Willams, 1992]

$$\nabla J(\theta) \propto \mathbb{E}_\pi\left[\sum_a Q_\pi(s_t, a)\,\nabla_\theta \pi(a|s_t;\theta)\right]$$

$$= \mathbb{E}_\pi\left[\sum_a \pi(a|s_t;\theta) Q_\pi(s_t, a)\,\frac{\nabla_\theta \pi(a|s_t;\theta)}{\pi(a|s_t;\theta)}\right]$$

Sample $a_t \sim \pi$

$$= \mathbb{E}_\pi\left[Q_\pi(s_t, a_t)\,\frac{\nabla_\theta \pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta)}\right]$$

$$= \mathbb{E}_\pi\left[R_t\,\frac{\nabla_\theta \pi(a|s;\theta)}{\pi(a|s;\theta)}\right]$$

Recall: $Q_\pi(s_t, a_t) \triangleq \mathbb{E}_\pi[R_t | s_t = s, a_t = a]$

$$= \mathbb{E}_\pi[R_t \nabla_\theta \log \pi(a|s;\theta)]$$

$$\nabla \log x = \frac{\nabla x}{x}$$

Log-derivative trick

*We have to wait for the ending of one episode to get the complete return

Monte Carlo

looking deeper

In the direction for
higher action probability

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left[R_t \frac{\nabla_\theta \pi(a|s;\theta)}{\pi(a|s;\theta)} \right]$$

In the direction for
higher return

To prevent frequently
selected action to be at
an advantage

update rule

Note:
learning rate can absorb "∝"

$$\theta_{t+1} \leftarrow \theta_t + \alpha\underbrace{[R_t \nabla_{\theta} \log \pi(a|s; \theta_t)]}_{\widehat{\nabla J(\theta_t)}}$$

REINFORCE baseline

a baseline to reduce variance

- $\boldsymbol{b(s) = \widehat{V}(s)}$
- $\boldsymbol{b(s) = \overline{R}}$
- $\boldsymbol{b^*(s)}$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha[\underbrace{(R_t - b(s))\nabla_{\boldsymbol{\theta}} \log \pi(a|s; \boldsymbol{\theta}_t)}_{\widehat{\nabla J(\boldsymbol{\theta}_t)}}]$$

formulate the policies

Discrete action

hypothesis function $h(\theta)$

- $\pi(a|s;\boldsymbol{\theta}) \triangleq \frac{\exp[\boldsymbol{\theta}^T \phi(s,a)]}{\sum_b \exp[\boldsymbol{\theta}^T \phi(s,b)]}$

feature of states and actions i.e. $\boldsymbol{\phi}(\boldsymbol{s}) = \boldsymbol{s}$

Continuous action

- $\pi(a|s;\boldsymbol{\theta}) \triangleq \mathcal{N}(\underbrace{\boldsymbol{\theta}_\mu^T \phi_\mu(s)}_{\boldsymbol{\mu}}, \underbrace{\exp^2[\boldsymbol{\theta}_\sigma^T \phi_\sigma(s)]}_{\boldsymbol{\sigma}^2})$

for the gradient

Discrete action

- $$\nabla_{\boldsymbol{\theta}} \log \pi(a|s;\boldsymbol{\theta}) = \phi(s,a) - \mathbb{E}_{\pi}\left[\phi(s,\cdot)\right]$$

Continuous action

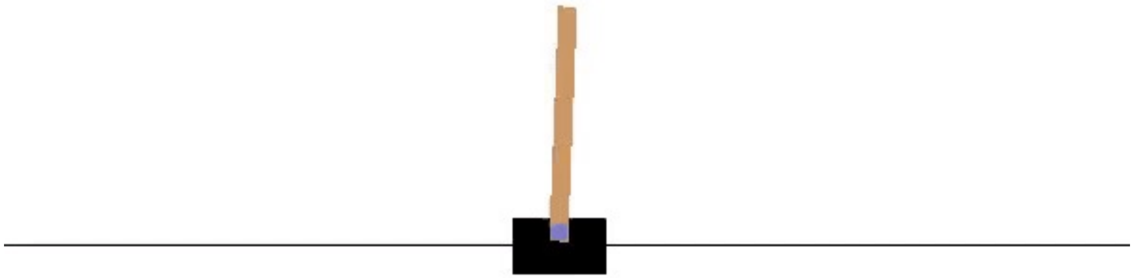
- $$\nabla_{\boldsymbol{\theta}_{\mu}} \log \pi\left(a\middle|s;\boldsymbol{\theta}_{\mu}\right) = \frac{[a-\mu(s)]\phi_{\mu}(s)}{\sigma^2(s)}$$
- $$\nabla_{\boldsymbol{\theta}_{\sigma}} \log \pi(a|s;\boldsymbol{\theta}_{\sigma}) = \left[\frac{\left(a-\mu(s)\right)^2}{\sigma^2(s)} - 1\right]\phi_{\sigma}(s)$$

Cartpole
[gym]

Gaussian
policy

r +1 all time

T: out of range



4 states $\boldsymbol{X^T = [x, \dot{x}, \theta, \dot{\theta}]}$

- x - cart position
- $\dot{x}$ - cart velocity
- θ - pole angle
- $\dot{\theta}$ - pole angular velocity

1 continuous action

- F - [-10, 10]N
- $a \sim \mathcal{N}(\theta_\mu^T X, (\exp \theta_\sigma^T X)^2)$

in python

```
theta_mu=np.zeros((4,1))
theta_sig=np.zeros((4,1))

for ep in range(n_eps):
    stp,r_sum,done=0,0,False
    states,actions,rewards,mus,sigs=[],[],[],[],[]
    s=env.reset().reshape((4,1))

    for stp in range(n_stps):

        a,mu,sig=Gaussian_policy(theta_mu,theta_sig,s)
        s_,r,done,_=env.step(a)
        s_=s_.reshape((4,1))

        states.append(s)
        actions.append(a)
        rewards.append(r)
        mus.append(mu)
        sigs.append(sig)

        s=s_
        stp+=1

        if done:
            break

    R=get_return(rewards,gm)

    gmt=1
    for i in range(len(rewards)):
        dlog_mu,dlog_sig=get_dlog(mus[i],sigs[i],states[i],actions[i])
        theta_mu=theta_mu+lr*gmt*R[i]*dlog_mu
        theta_sig=theta_sig+lr*gmt*R[i]*dlog_sig
        gmt*=gm
```

Need to do memory cache for calculating return for each step at the end of the episode

Policy parameter update

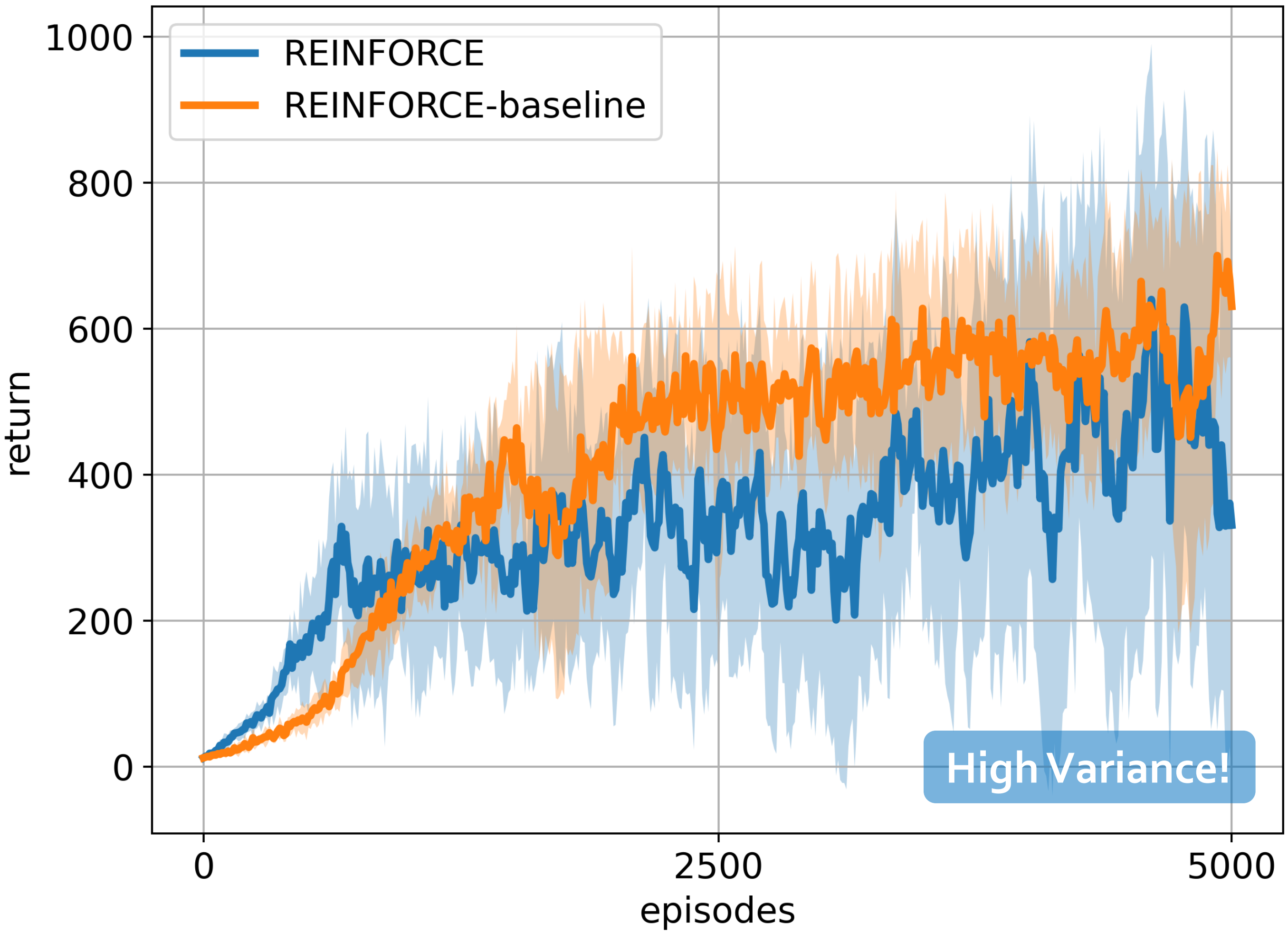
in python

```python
def Gaussian_policy(theta_mu,theta_sig,s):

    mu=theta_mu.T.dot(s)[0]
    upper=theta_sig.T.dot(s)
    sig=np.exp(upper-np.max(upper))[0]

    return np.random.normal(mu,sig)[0],mu[0],sig[0]

def get_dlog(mu,sig,s,a):

    dlog_mu=((a-mu)/(sig**2))*s
    dlog_sig=(((a-mu)**2/sig**2)-1)*s

    return dlog_mu,dlog_sig

def get_return(rewards,gm):

    R=np.zeros(len(rewards))
    R[-1]=rewards[-1]
    for i in range(2,len(R)+1):
        R[-i]=gm*R[-i+1]+rewards[-i]

    return R
```

avoiding np.exp overflow

a reversive return for each step

results



1000
800
600
400
200
0
return
0
2500
5000
episodes
REINFORCE
REINFORCE-baseline
High Variance!