

Basic RL.4

Judy Tutorial

Recall Q-learning:

Value Update

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

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

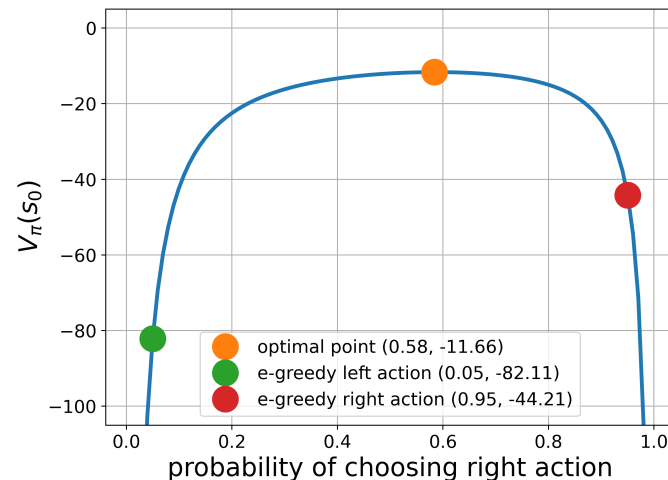
Policy Update
for sampling
(s,a,r,s')

Problems w/ Value-based methods

- If state and action spaces are large:
 $\pi(a|s) \leftarrow \operatorname{argmax}_a Q(s, a)$ 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:

$$\pi(a|s; \theta)$$

└ parameters

updated using gradient ascent:

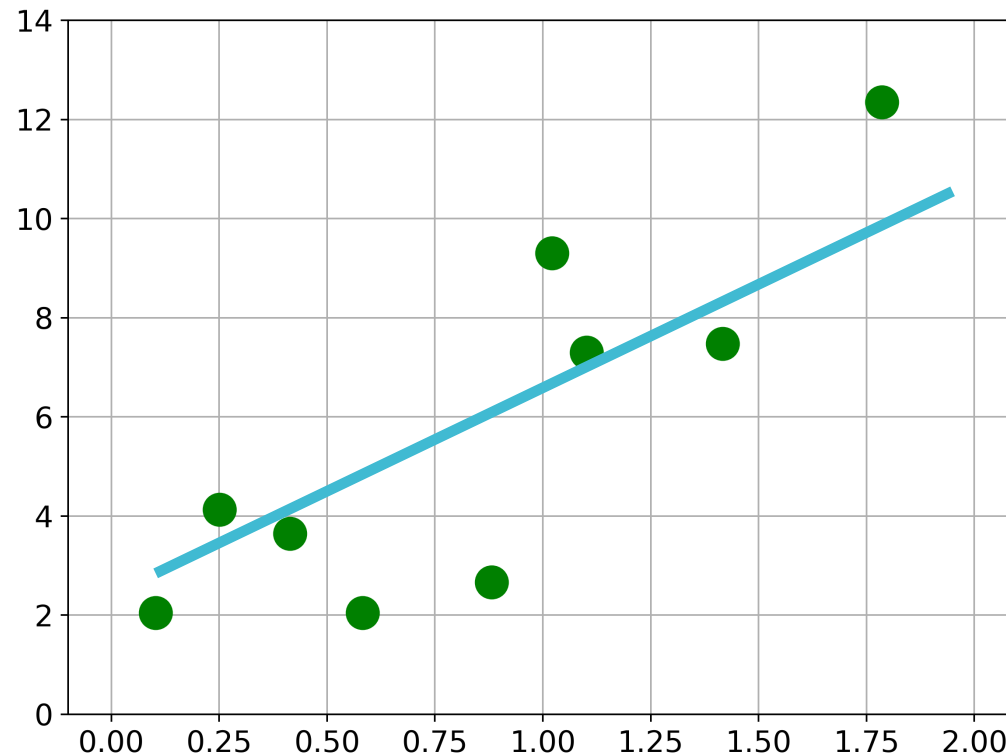
$$\theta_{t+1} \leftarrow \theta_t + \alpha \widehat{\nabla J(\theta_t)}$$

where $J(\theta)$ 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!

Hypothesis function

$$h(\theta) = \theta_1 x + \theta_0 = \theta^T X$$

inputs

dot product

parameters/weights

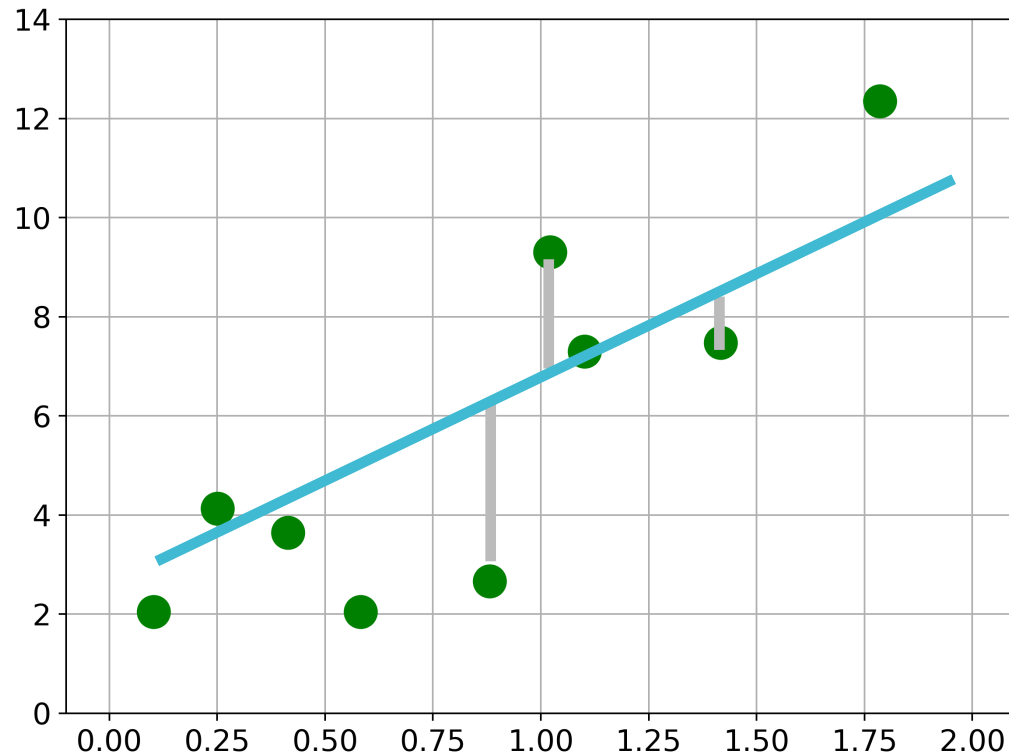
$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}, X = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

Objective function

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

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



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

error

Hypothesis function

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

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

Objective function

$$J(\theta) = \frac{1}{2N} \| h_{1 \times 10} - Y_{1 \times 10} \|^2$$

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

Objective function

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

Gradient

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

Element-wise

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

Need to take gradient w.r.t θ_j one by one

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

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=200

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
```

$X - (2,10)$

$\theta - (2,1)$

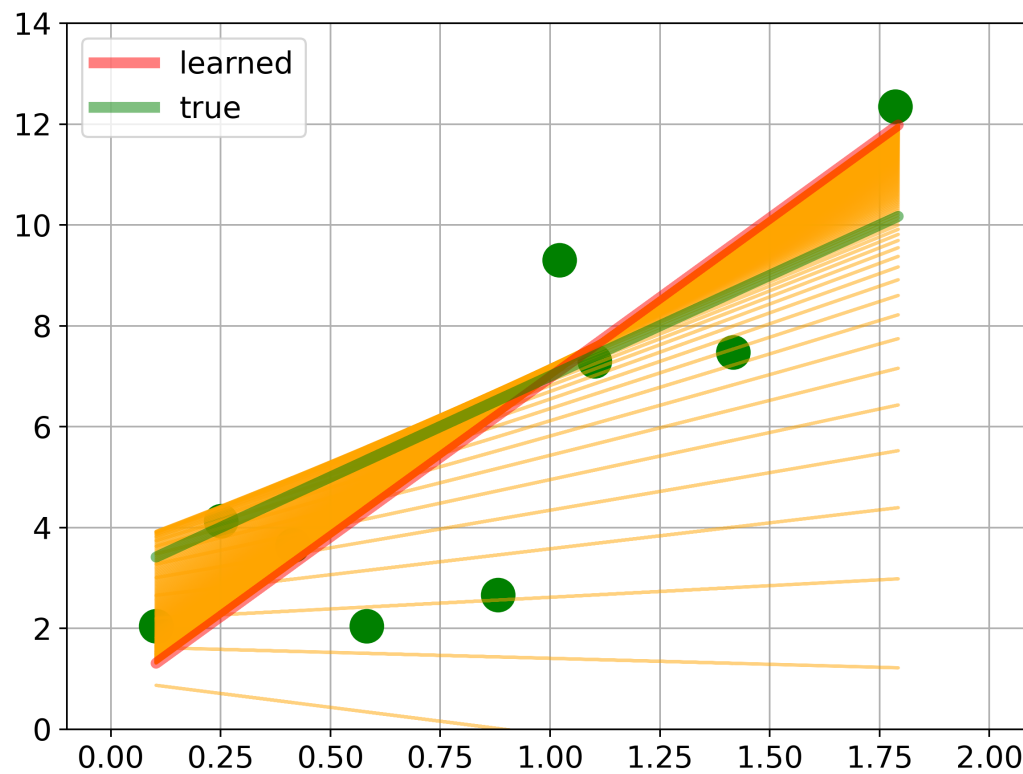
$h = \theta^T X - (1,10) \times -(1,2)(2,10)$

$\text{grad} = X(h - Y)^T - (2,1) \times -(2,10)(10,1)$

$\theta = \theta - lr * \text{grad} - (2,1)$

Data shape is
important in
implementation!

learning behavior



$$\theta_0 = \begin{bmatrix} 0.982 \\ -1.1 \end{bmatrix} \rightarrow \theta_f = \begin{bmatrix} 0.658 \\ 6.317 \end{bmatrix} \quad ? \quad \theta_f? = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Gradient is the direction of steepest descent

