Final Exam Reviews

Machine Learning (Regression)

Supervised learning

Data

$$\begin{tikzcd}
& \boxed{\text{Training data set } D} \arrow[d] & \\
& \boxed{\text{Learning Algorithm}} \arrow[d, "\hat{\theta}"] & \\
x_* \arrow[r] & \boxed{\text{Hypothesis } f(x;\theta^*)} \arrow[r] & y_*
\end{tikzcd}$$

$D = \{(x_i, y_i); i = 1, \dots, m\}$

$\boldsymbol{x}_i = (x_{i1}, \cdots, x_{in})$

$x_i = (x_{i1}, \cdots, x_{in})$

$$\begin{array}{c|cccc|c|c}
 & x_1 & x_2 & \cdots & x_n & y & \\
\hline
 & x_{11} & x_{12} & \cdots & x_{1n} & y_1 & \\
\text{m input Feature vectors} & x_{21} & x_{22} & \cdots & x_{2n} & y_2 & \text{m outputs} \\
 & \vdots & \vdots & & \vdots & \vdots & \\
 & x_{m1} & x_{m2} & \cdots & x_{mn} & y_m &
\end{array}$$

model

Functional form $f(x;\theta)$ Is usually given

Using training data set, a learning algorithm finds the best hypothesis function $h(x)$ that **is believed to accurately predict** the output y for a given query input x

prediction

Query input x_*

Input feature vector $x_* = (x_{*1}, x_{*2}, \dots x_{*n})$

Predicted output y_*

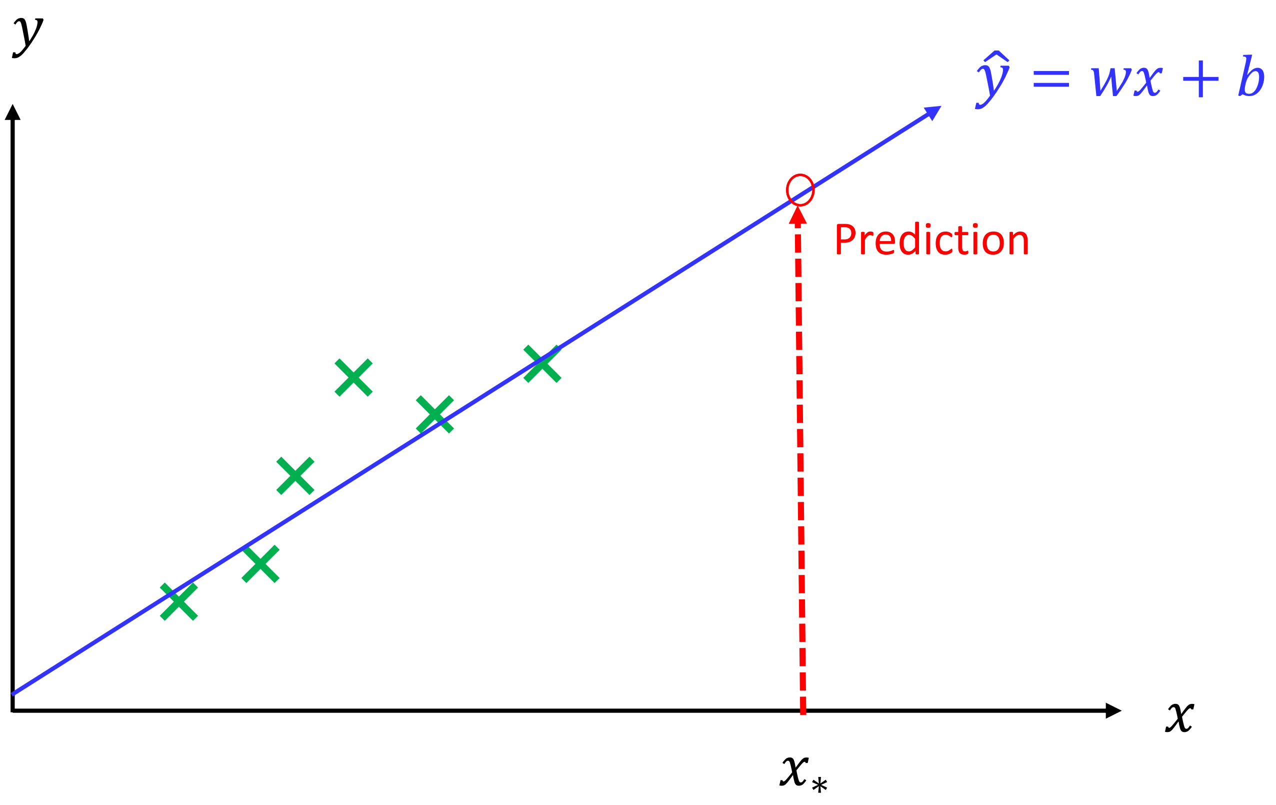
if $y_* \in \mathbb{R}$: **Regression**

if $y_* \in \{1, \dots, N\}$: **Classification**

Load Map

1. Optimization Approach (Normal Equation)
2. Maximum Likelihood Estimation (MLE) Approach
3. Maximum A Posteriori Estimation (MAP) Approach
4. Full Bayesian Approach
   - ✓ Analytical approach
   - ✓ Sampling approach
5. Regularization regression (Ridge and Lasso)
   - ✓ Optimization view
   - ✓ Bayesian View

1D Linear Regression

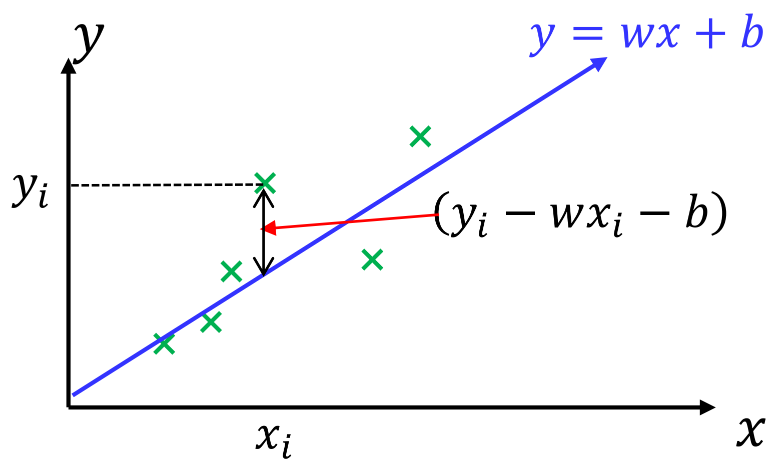


- **Data** : $(x_1, y_1), \ldots, (x_m, y_m)$
- **Model**: Linear model $\hat{y} = wx + b$ $\quad (y = w^T x + b$ for multidimensional)
- **Learning**: What are w and b?
- **Prediction** : What is $\hat{y}_* = wx_* + b$

Learning as optimization

- Define an objective (cost) function

$$J(w,b) = \sum_{i=1}^{m} (y_i - wx_i - b)^2$$



- Minimize the error function with respect to w and b

$$\frac{dJ(w,b)}{dw} = -2\sum_{i=1}^{m} x_i(y_i - wx_i - b) \;= 0 \rightarrow w^* = \frac{\sum_{i=1}^{m}(y_i - b^*)x_i}{\sum_{i=1}^{n} x_i^2} = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{m}(x_i - \bar{x})^2}$$

$$\frac{dJ(w,b)}{db} = -2\sum_{i=1}^{m} (y_i - wx_i - b) \;= 0 \rightarrow b^* = \frac{\sum_{i=1}^{m}(y_i - w^* x_i)}{n}$$

Learning as optimization

Notation for general cases $x_i \in \mathbb{R}^n$

- A linear regression model

$$\hat{y}_i = w_0 + w_1 x_{i1} + \cdots + w_n x_{in}$$

with $w = (w_0, w_1, \ldots, w_n)^T$ and $x_i = (x_{i1}, \ldots, x_{in})^T$

- If we introduce $x_{i0} = 1$,

$$\hat{y}_i = w^T x_i$$

with $w = (w_0, w_1, \ldots, w_n)^T$ and $x_i = (\underset{\substack{\nwarrow \\ 1}}{x_{i0}}, x_{i1}, \ldots, x_{in})^T$

- In a Matrix form

$$\begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_m \end{pmatrix} = \begin{pmatrix} -x_1^T - \\ \vdots \\ -x_m^T - \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} x_1^T w \\ \vdots \\ x_m^T w \end{pmatrix} \quad \to \hat{y} = Xw$$

m: # of data points

with $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)^T$, $\quad X = \begin{pmatrix} -x_1^T - \\ \vdots \\ -x_m^T - \end{pmatrix}$

Learning as optimization (Normal Equation)

- The cost function for the optimization can be defined as :

$$J(w) = \frac{1}{2}\sum_{i=1}^{m}\left(x_i^T w - y_i\right)^2 = \frac{1}{2}\|y - wX\|_2^2 = \frac{1}{2}(Xw - y)^T(Xw - y)$$

$$\sqrt{\Sigma_{i=1}^{m} z_i^2} = \|z\|_2 = \sqrt{z^T z}$$

- The optimum parameters $\hat{w}$ can be computed by minizing the cost function :

$$\hat{w} = \arg\min_w J(w) = \arg\min_w \frac{1}{2}\|y - wX\|_2^2$$

- For reference, other vector norms are summarized here :

$$\|z\|_1 = \sum_{i=1}^{n}|z_i|, \qquad \|z\|_p = \left(\sum_{i=1}^{n}|z_i|^p\right)^{\frac{1}{p}} (p \geq 1), \qquad \|z\|_\infty = \max_i |z_i|$$

Learning as optimization (Normal Equation)

- For an n-by-n (square) matrix A, the trace of A is defined to be the sum of its diagonal entries:

$$\mathrm{tr}A = \sum_{i=1}^{n} A_{ii}$$

$$\mathrm{tr}AB = \mathrm{tr}BA$$

$$\mathrm{tr}ABC = \mathrm{tr}CBA = \mathrm{tr}BCA$$

$$\mathrm{tr}A = \mathrm{tr}A^T$$

$$\mathrm{tr}(A + B) = \mathrm{tr}A + \mathrm{tr}B$$

$$\mathrm{tr}aA = a\mathrm{tr}A$$

- Trace operator associated with Matrix derivatives

$$\nabla_A \mathrm{tr}AB = B^T$$

$$\nabla_{A^T} f(A) = \left(\nabla_A f(A)\right)^T$$

$$\nabla_A \mathrm{tr}ABA^TC = CBA + C^TAB^T$$

$$\nabla_{A^T} \mathrm{tr}ABA^TC = (CBA + C^TAB^T)^T = B^TA^TC^T + BA^TC$$

Learning as optimization (Normal Equation)

Linear algebra approach for finding the optimum parameters:

$$\widehat{w} = \arg\min_{w} J(w) = \arg\min_{w} \frac{1}{2}\|y - wX\|_2^2$$

Since, $J(w)$ is differentiable in w, the optimality condition : $\nabla_w J(w) = 0$ at $w = \widehat{w}$

$$\begin{aligned}
\nabla_w J(w) &= \nabla_w \frac{1}{2}(Xw - y)^T(Xw - y) \\
&= \frac{1}{2}\nabla_w (w^T X^T X w - w^T X^T y - y^T X w + y^T y) \\
&= \frac{1}{2}\nabla_w \mathrm{tr}(w^T X^T X w - w^T X^T y - y^T X w + y^T y) \\
&= \frac{1}{2}\nabla_w \big(\mathrm{tr}(w^T X^T X w) - 2\mathrm{tr}(y^T X w)\big) \\
&= \frac{1}{2}(X^T X w + X^T X w - 2X^T y) \qquad \because \nabla_{A^T} \mathrm{tr} ABA^T C = (CBA + C^T AB^T)^T = B^T A^T C^T + BA^T C \\
&= X^T X w - X^T y
\end{aligned}$$

$$\nabla_w J(\widehat{w}) = X^T X\widehat{w} - X^T y = 0$$

$$\rightarrow X^T X \widehat{w} = X^T y$$

$\rightarrow \widehat{w} = (X^T X)^{-1} X^T y$ (when X is full column rank)

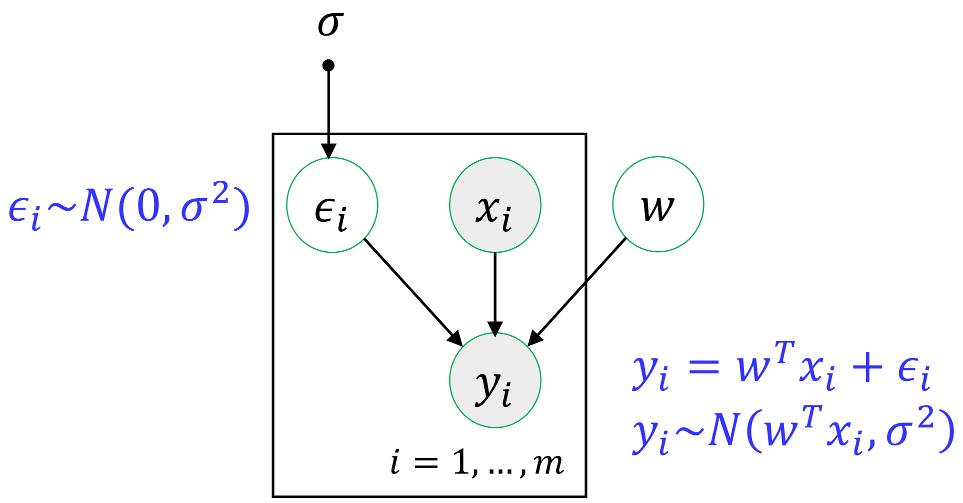
Probabilistic view on linear regression

- Assume there is uncertainty in the predicted value :

$$y_i = w^T x_i + \epsilon_i \text{ with } \epsilon_i \sim N(0, \sigma^2)$$

- Then the probabilistic model on output y_i can be represented as

$$y_i \sim N(w^T x_i, \sigma^2) \quad \text{or} \quad p(y_i | w^T x_i, \sigma) = N(y_i | w^T x_i, \sigma^2)$$


σ
$\epsilon_i \sim N(0, \sigma^2)$
ϵ_i
x_i
w
y_i
$i = 1, \dots, m$
$y_i = w^T x_i + \epsilon_i$
$y_i \sim N(w^T x_i, \sigma^2)$

An error ϵ_i is independently identically distributed (i.i.d assumption)

- The likelihood of the data is defined as

$$p(y|X, w, \sigma) = \prod_{i=1}^{m} N(y_i | w^T x_i, \sigma^2) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$$

Learning as probabilistic model (MLE Approach)

- The log likelihood is

$$
\begin{aligned}
L(w,\sigma) &= \log p(y|X,w,\sigma) \\
&= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right) \\
&= \log \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^m \exp\left(-\sum_{i=1}^{m} \frac{(y_i - w^T x_i)^2}{2\sigma^2}\right) \\
&= m \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{\sigma^2} \frac{1}{2} \sum_{i=1}^{m} (y_i - w^T x_i)^2 \\
&= m \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{\sigma^2} J(w)
\end{aligned}
$$

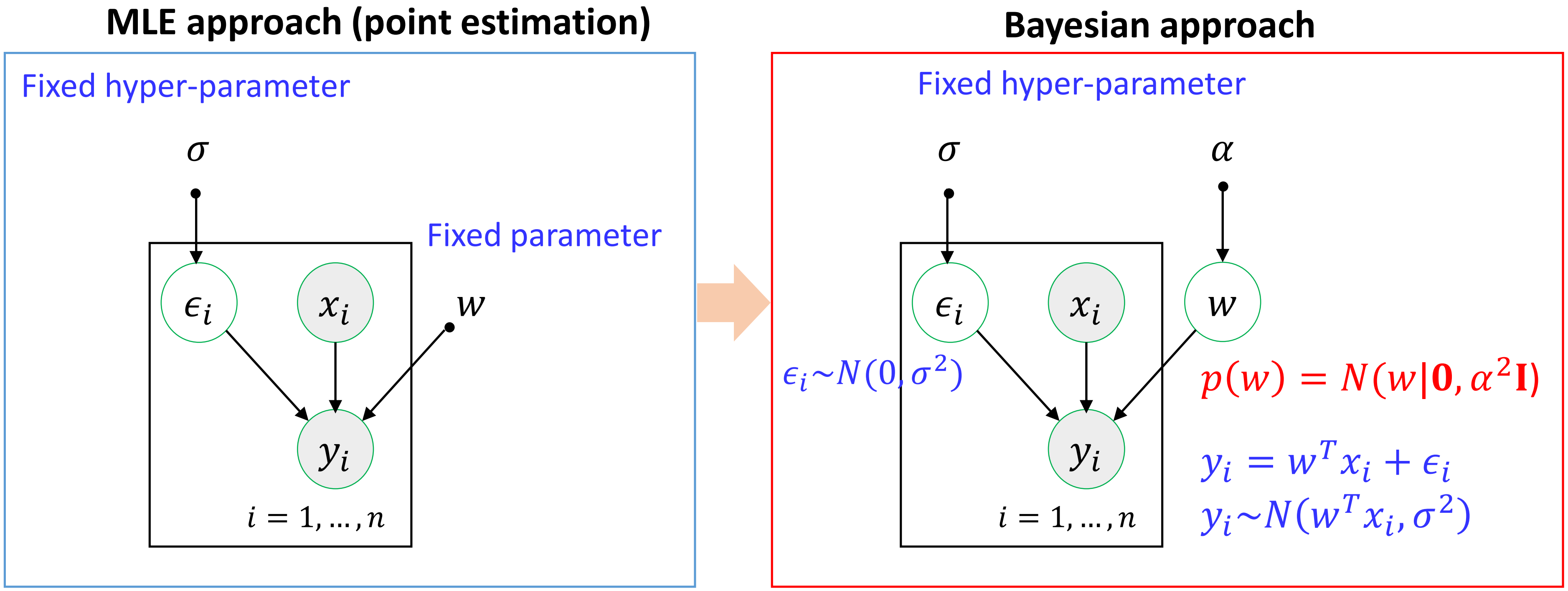
- The optimum parameters is determined by maximizing log likelihood

$$
(w^*, \sigma) = \max_{(w,\sigma)} L(w,\sigma) = \max_{(w,\sigma)} \log p(y|X,w,\sigma)
$$

Minimizing the square error sum $J(w) =$

Maximizing the log likelihood $\log p(y|X,w,\sigma)$ with respect to w

Learning as probabilistic model (Bayesian Approach)



- Consider the parameter w as stochastic variables (represented as a distribution)
- (Assume σ is known for simple derivation)
- Find the distribution on parameter w

$$p(w|y,X) = \frac{p(y|X,w)p(w|X)}{\int_w p(y|X,w)p(w|X)dw} \quad \rightarrow \quad p(w|y) = \frac{p(y|\, w)p(w)}{\int_w p(y|\, w)p(w)dw}$$

We will assume X is fixed for the data y

Learning as probabilistic model (Bayesian Approach)

- Multivariate regression likelihood is

$$\begin{aligned} p(y|w) &= \prod_{i=1}^{m} p(y_i|x_i, w) \\ &= \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - w^T x_i)^2\right) \end{aligned}$$

$m =$ # of data points

- Multivariate Gaussian prior on parameter w

$$p(w) = N(w|\mathbf{0}, \alpha^2\mathbf{I})$$

$$p(w) = \frac{1}{(2\pi\alpha^2)^{n/2}} \exp\left(-\frac{1}{2\alpha^2} w^T w\right)$$

$n =$ Dimension of w

Learning as probabilistic model (Bayesian Approach)

- **We want to find the posterior**

$$p(w|X,y) \propto p(y|X,w)p(w)$$
$$= \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - w^T x_i)^2\right) \frac{1}{(2\pi\alpha^2)^{n/2}} \exp\left(-\frac{1}{2\alpha^2} w^T w\right)$$

- **Take log:**

$$\log p(w|X,y) = -\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - w^T x_i)^2 - \frac{1}{2\alpha^2} w^T w + const$$
$$= -\frac{1}{2\sigma^2}\sum_{i=1}^{m} y_i^2 + \frac{1}{\sigma^2}\sum_{i=1}^{m} y_i x_i^T w - \frac{1}{2\sigma^2}\sum_{i=1}^{m} w^T x_i x_i^T w - \frac{1}{2\alpha^2} w^T w + const$$
$$= -\frac{1}{2\sigma^2} y^T y + \frac{1}{\sigma^2} y^T X w - \frac{1}{2\sigma^2} w^T X^T X w - \frac{1}{2\alpha^2} w^T w + const$$
$$= -\frac{1}{2\sigma^2} y^T y + \frac{1}{\sigma^2} y^T X w - \frac{1}{2} w^T \left[\frac{1}{\sigma^2} X^T X + \frac{1}{\alpha^2} I\right] w + const$$

- **Posterior distribution is**

$$p(w|X,y) = N(w|\mu_w, \Sigma_w)$$

$$\mu_w = \Sigma_w \left(\frac{1}{\sigma^2} X^T y\right) \qquad \Sigma_w = \left[\frac{1}{\sigma^2} X^T X + \frac{1}{\alpha^2} I\right]^{-1}$$

Learning as probabilistic model (Bayesian Approach)

- Predictive distribution

$$p(y_*|x_*, X, y) = \int_w p(y_*|x_*, w)p(w|X, y)dw$$

Jupyter Demo Simulation
Bayesian Regression Analytical

Learning as probabilistic model (Bayesian Approach)

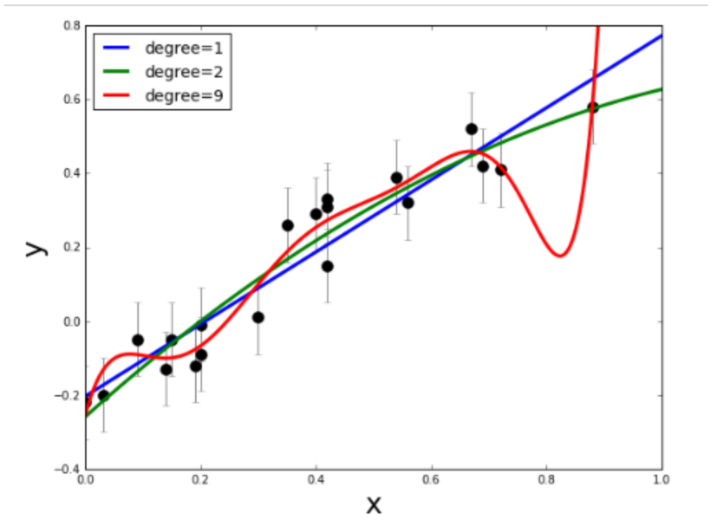
- Predictive distribution

$$p(y_*|x_*, X, y) = \int_w p(y_*|x_*, w)p(w|X, y)dw$$

Jupyter Demo Simulation

Bayesian Regression (Sampling using PyMC)

Regularized linear regression



What is a good regression function?

- The goal of regression is to come up with some good prediction function:

$$\hat{f}(x) = x^T \widehat{w}$$

- So far, we have found $\widehat{w}$ by finding (Ordinary Least Square Estimation)

$$\widehat{w} = \arg\min_{w} J(w) = \arg\min_{w} \frac{1}{2} \|y - wX\|_2^2$$

- To see whether $\hat{f}(x)$ is a good candidate, we need to check

  ✓ Is $\widehat{w}$ close to the true w?
  ✓ Will $\hat{f}(x)$ fit future observation well? (Generalization)

Regularized linear regression

Ridge regression

- Ridge regression introduces a regularization with the L-2 norm:

$$\hat{w} = \underset{w}{\operatorname{argmin}} \|y - wX\|_2^2 + \lambda_2 \|w\|_2^2 \qquad \|w\|_2 = \sqrt{\Sigma_{i=1}^{k} w_i^2,}$$

- Sacrifice a little of bias to reduce the variance of predicted values

→ More stable and generalize better

- Keep all the repressors in the model

→ Not easily interpretable model

Lasso (Least Absolute Shrinkage and Selection Operator)

- Lasso regression introduces a regularization with the L-1 norm:

$$\hat{w} = \underset{w}{\operatorname{argmin}} \|y - wX\|_2^2 + \lambda_1 \|w\|_1 \qquad \|w\|_1 = \Sigma_{i=1}^{k} |w_i|$$

- Only a small subset of features with $\hat{w}_i \neq 0$ are selected

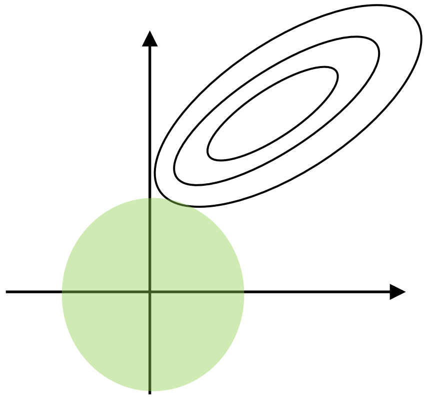
→ Increases the interpretability

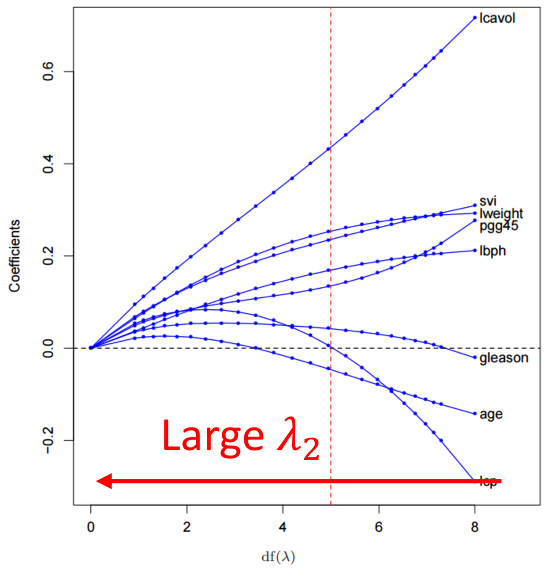
- More difficult to implement than Ridge Regression

Regularized linear regression

Ridge regression

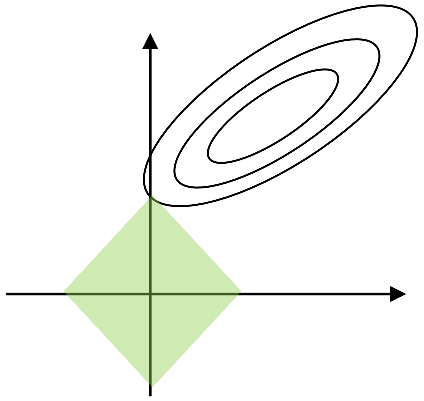
$$\hat{w} = \underset{w}{\operatorname{argmin}} \|y - wX\|_2^2 + \lambda_2 \|w\|_2^2$$

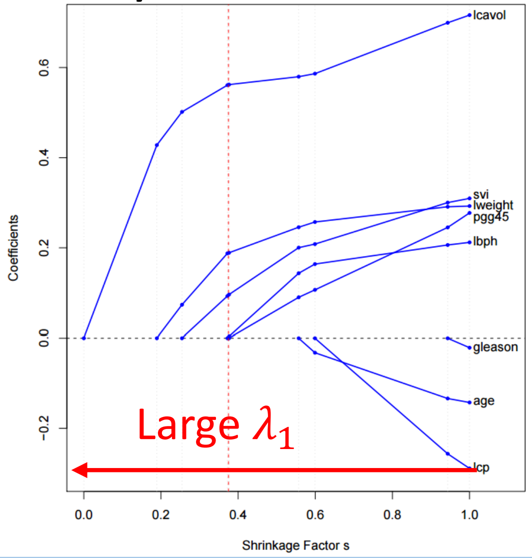




Lasso (Least Absolute Shrinkage and Selection Operator)

$$\hat{w} = \underset{w}{\operatorname{argmin}} \|y - wX\|_2^2 + \lambda_1 \|w\|_1$$



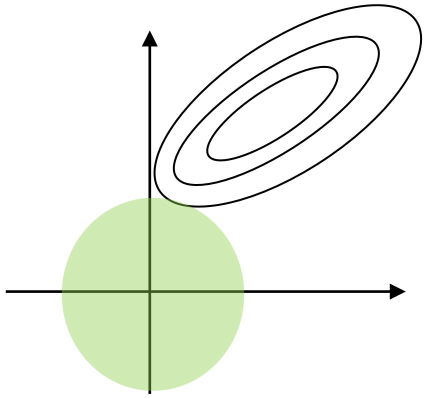


Bayesian view on Ridge regression

Ridge regression

$$\hat{w} = \underset{w}{\operatorname{argmin}} \|y - wX\|_2^2 + \lambda_2 \|w\|_2^2$$

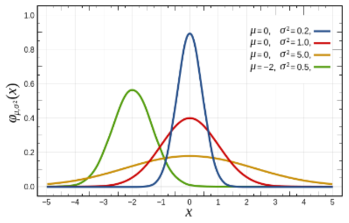
$$\hat{w} = \underset{w}{\operatorname{argmax}} \; \log p(w|X,y) = \log p(y|X,w)p(w)$$



MAP estimation view

Gaussian prior

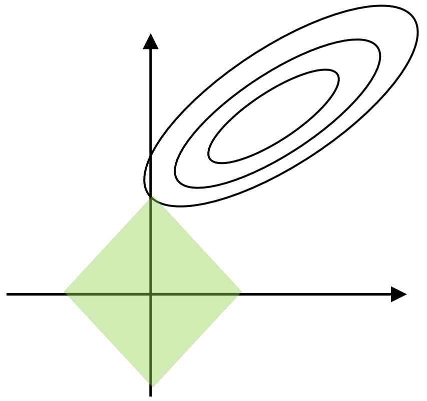
$$p(w) = \frac{1}{(2\pi\alpha^2)^{k/2}} \exp\left(-\frac{1}{2\alpha^2} w^T w\right)$$



Lasso (Least Absolute Shrinkage and Selection Operator)

$$\hat{w} = \underset{w}{\operatorname{argmin}} \|y - wX\|_2^2 + \lambda_1 \|w\|_1$$

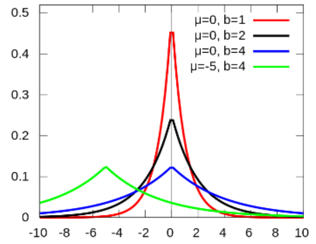
$$\hat{w} = \underset{w}{\operatorname{argmax}} \; \log p(w|X,y) = \log p(y|X,w)p(w)$$



MAP estimation view

Laplace prior

$$p(w) = \prod_{i=1}^{k} \frac{\lambda}{2\sqrt{\tau^2}} \exp\left(-\frac{\lambda|w_i|}{\sqrt{\tau^2}}\right)$$



Machine Learning (Classification)

Contents

- **Non-Bayesian approaches**
  - Discriminative model
    - ✓ Logistic regression
    - ✓ Neural Network
  - Generative model
    - ✓ Gaussian Discriminative Analysis
    - ✓ Naïve Bayes classification
- **Full Bayesian approach for classification**
  - Bayesian Logistic regression
  - Bayesian Neural Network

Non-Bayesian vs Bayesian

- **Non-Bayesian approaches**

✓ *discriminative* probabilistic classification

$$p(y|x) = f(w^T x)$$

Directly model posterior $p(y|x)$ using parametric form

✓ *Generative* probabilistic classification

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y\in Y} P(x|y)P(y)}$$

Model $P(x|y)$ and $P(y)$ and combined them in Bayes' rule

$$\hat{y} = \underset{y\in Y}{\operatorname{argmax}}\, p(y|x)$$

- **Full Bayesian approach for classification**

1. Construct prior $p(w)$
2. Construct likelihood $p(D|w)$, where $D = \{(x_i, y_i)\}_{i=1}^{m}$
3. Construct posterior $p(w|D) = \frac{p(D|w)p(w)}{p(D)}$
4. Posterior predictive distribution $p(y_*|x_*, D) = \int_w p(y_*|x_*, w)p(w|D)dw$

Logistic regression

- Logistic regression is *discriminative* probabilistic linear classification :$p(y|x) = g(w^T x)$

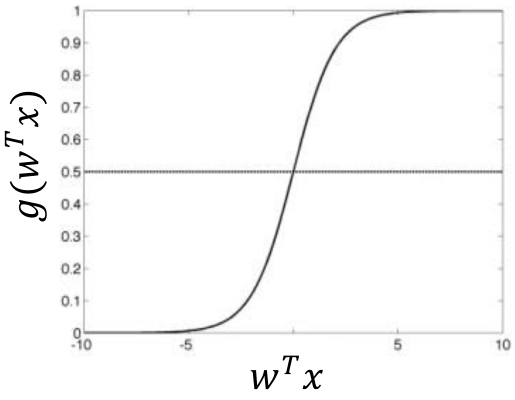
Let's denote p a probability of having $y = 1$

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = w^T x$$

$$\frac{p}{1-p} = \exp(w^T x\,)$$

$$p = \frac{\exp(w^T x\,)}{1+\exp(w^T x\,)} = \frac{1}{1+\exp(-w^T x\,)} = g(w^T x)$$

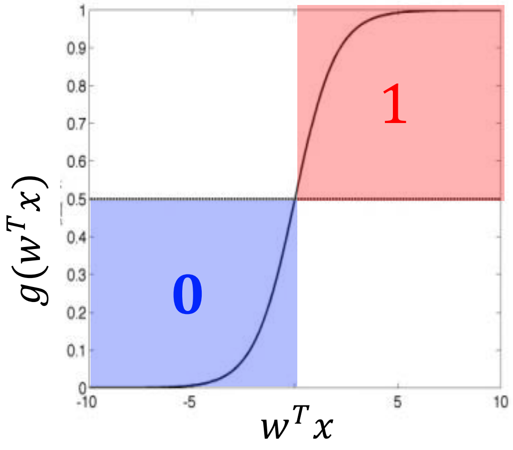
- Larger $w^T x$ →lareger→ $g(w^T x)$ → higher p for $y = 1$
- Smaller $w^T x$ →smaller→ $g(w^T x)$ → lower p for $y = 1$



$$g(z) = \frac{1}{(1+\exp(-w^T x))}$$

- Classification rule:

$$y = \begin{cases} 0, & \text{if } p(Y=1|x) = g(w^T x) < 0.5 \Leftrightarrow w^T x < 0 \\ 1, & \text{if } p(Y=1|x) = g(w^T x) \geq 0.5 \Leftrightarrow w^T x \geq 0 \end{cases}$$



Generative model

1. Define Class prior $p(y)$ and likelihood $P(x|y)$

2. Learn the parameters of the models, $P(y)$ and $P(x|y)$

3. Express posterior distribution on class y given the input vector x

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y \in Y} P(x|y)P(y)}$$

4. Prediction step: any new input feature vector x_{new} can be classified according to the maximum a posteriori detection principle (MAP)

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y|x_{new}) = \underset{y}{\operatorname{argmax}} \frac{P(x_{new}|y)P(y)}{\sum_y P(x_{new}|y)P(y)}$$

$$= \underset{y}{\operatorname{argmax}} P(x_{new}|y)P(y)$$

Generative model

1. Define prior and likelihood

$$p(x|y=\text{dog}), p(y=\text{dog})$$
$$p(x|y=\text{cat}),\ p(y=\text{cat})$$

2. Learn the parameters for the models

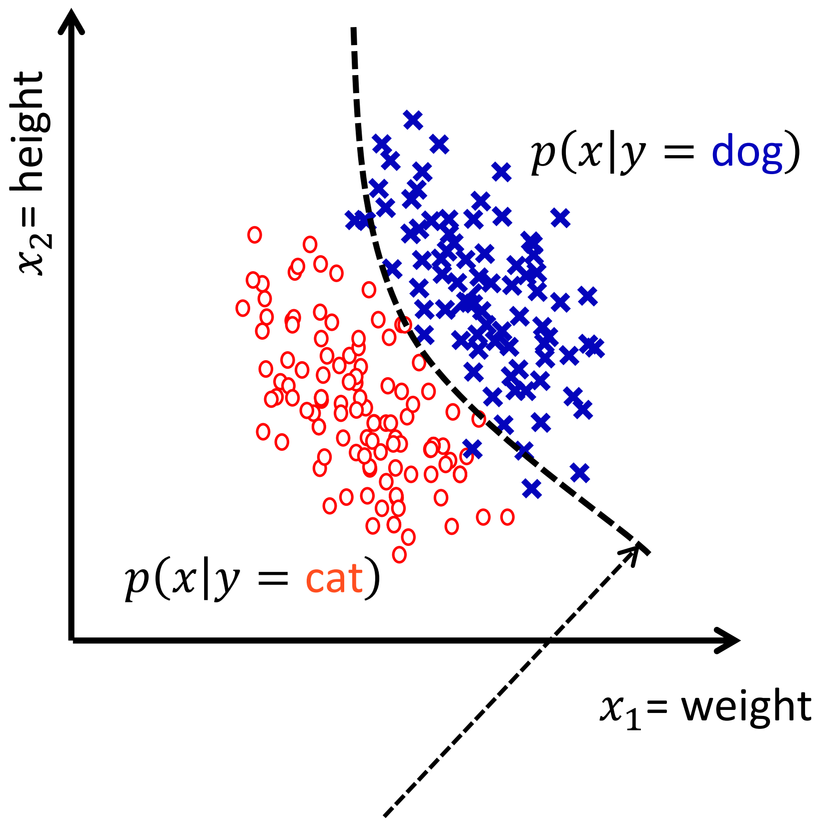
3. Construct the posterior distribution on class

$$P(y=\text{dog}|x) = \frac{p(x|y=\text{dog})p(y=\text{dog})}{\sum_{y\in\{\text{dog},\text{cat}\}} p(x|y)p(y)}$$

$$P(y=\text{cat}|x) = \frac{p(x|y=\text{cat})p(y=\text{cat})}{\sum_{y\in\{\text{dog},\text{cat}\}} p(x|y)p(y)}$$

4. Classify animal based on MAP estimation:

$$\hat{y} = \underset{y\in Y}{\operatorname{argmax}}\, P(y|x^{new})$$



Decision boundary

$$p(y=\text{dog}|x) = p(y=\text{cat}|x)$$

The shape of a decision boundary changes depending on the assumptions on the model (ex., linear, quadratic, …)

Gaussian Discriminant Analysis

Parameter learning for GDA

- Using the training data $\boldsymbol{D} = \{(x_i, y_i); i = 1, \dots, m\}$, the parameter sets for GDA are:

  $\boldsymbol{\phi} = \{\phi_1, \dots, \phi_N\}$: set of priors
  $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_N\}$: set of mean vectors
  $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_N\}$: set of covariance matrices

- The parameters are found as ones maximizing the log-likelihood of data

$$\log p(D|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\phi}) = \log \prod_{i=1}^{m} p(x_i|y_i, \boldsymbol{\mu_i}, \boldsymbol{\Sigma}_{\mathrm{i}}) P(y_i|\boldsymbol{\phi_i})$$
$$= \sum_{i=1}^{m} \log\, p(x_i|y_i, \boldsymbol{\mu_i}, \boldsymbol{\Sigma}_{\mathrm{i}}) P(y_i|\boldsymbol{\phi_i})$$

The log-likelihood function is **concave function** in terms of the parameters
→ the optimum parameters are analytically derived as

$$\phi_j = \frac{1}{m} \sum_{i=1}^{m} 1\{y_i = j\}$$

$$\boldsymbol{\mu}_j = \frac{\sum_{i=1}^{m} 1\{y_i = j\}\, x_i}{\sum_{i=1}^{m} 1\{y_i = j\}}$$

$$\boldsymbol{\Sigma}_j = \frac{1}{\sum_{i=1}^{m} 1\{y_i = j\}} \sum_{i=1}^{m} 1\{y_i = j\}(x_i - \boldsymbol{\mu}_j)(x_i - \boldsymbol{\mu}_j)^T$$

Indication function

$$1\{y_i = j\} = \begin{cases} 1, \text{if} \quad y_i = j \\ 0, \text{otherwise} \end{cases}$$

Gaussian Discriminant Analysis

Class Prediction

- The probability of class $y = j$ given the new input x^{new} can be computed

$$P(y = j\,|x^{new}) \sim P(x^{new}|y = j)P(y = j) \qquad p(y|x^{new}) = \frac{P(x^{new}|y)P(y)}{P(x^{new})}$$

$$= \frac{1}{\sqrt{(2\pi)^n|\mathbf{\Sigma}_j|}} \exp\left(-\frac{1}{2}\left(x^{new} - \boldsymbol{\mu}_j\right)^T \mathbf{\Sigma}_j^{-1}\left(x^{new} - \boldsymbol{\mu}_j\right)\right)\phi_j$$

- The class can be selected using MAP estimation:

$$\hat{y} = \underset{y \in Y}{\operatorname{argmax}}\, p(y\,|x^{new})$$

- The boundary surface between two neighboring classes i and j $\left(\text{i.e.}, P(y = i|\boldsymbol{x}) = P(y = j|\boldsymbol{x})\right)$

$$\frac{1}{\sqrt{(2\pi)^n|\mathbf{\Sigma}_i|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \mathbf{\Sigma}_i^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i)\right)\phi_i = \frac{1}{\sqrt{(2\pi)^n|\mathbf{\Sigma}_j|}} \exp\left(-\frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{\mu}_j\right)^T \mathbf{\Sigma}_j^{-1}\left(\boldsymbol{x} - \boldsymbol{\mu}_j\right)\right)\phi_j$$

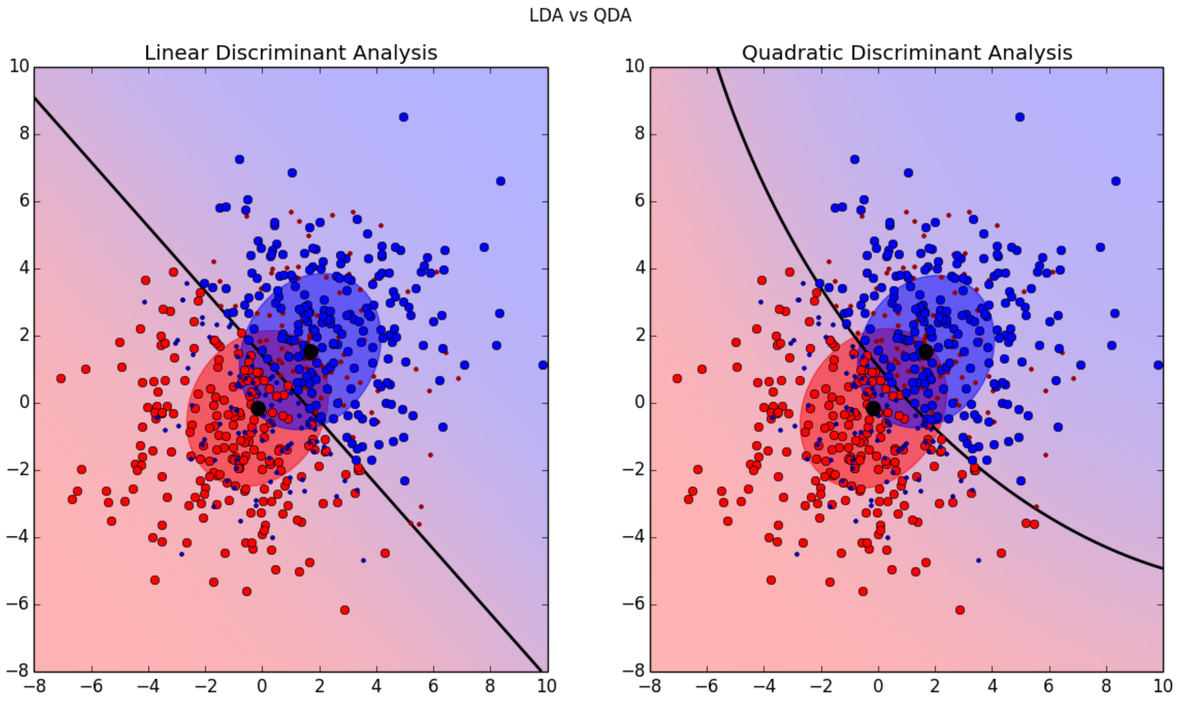
$$\rightarrow \boldsymbol{x}^T\left(\mathbf{\Sigma}_i^{-1} - \mathbf{\Sigma}_j^{-1}\right)\boldsymbol{x} - 2\left(\boldsymbol{\mu}_i^T\mathbf{\Sigma}_i^{-1} - \boldsymbol{\mu}_j^T\mathbf{\Sigma}_j^{-1}\right)\boldsymbol{x} + \boldsymbol{\mu}_i^T\mathbf{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \boldsymbol{\mu}_j^T\mathbf{\Sigma}_j^{-1}\boldsymbol{\mu}_j + \log\frac{\phi_j|\mathbf{\Sigma}_i|}{\phi_i|\mathbf{\Sigma}_j|} = 0$$

Gaussian Discriminant Analysis

Example (binary-classes)

The boundary surface between two neighboring classes i and j $\left(\text{i.e.}, P(y=i|\boldsymbol{x}) = P(y=j|\boldsymbol{x})\right)$

$$\boldsymbol{x}^T\left(\boldsymbol{\Sigma}_i^{-1} - \boldsymbol{\Sigma}_j^{-1}\right)\boldsymbol{x} - 2\left(\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}_i^{-1} - \boldsymbol{\mu}_j^T\boldsymbol{\Sigma}_j^{-1}\right)\boldsymbol{x} + \boldsymbol{\mu}_i^T\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \boldsymbol{\mu}_j^T\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\mu}_j + \log\frac{\phi_j|\boldsymbol{\Sigma}_i|}{\phi_i|\boldsymbol{\Sigma}_j|} = 0$$



Linear discriminant analysis

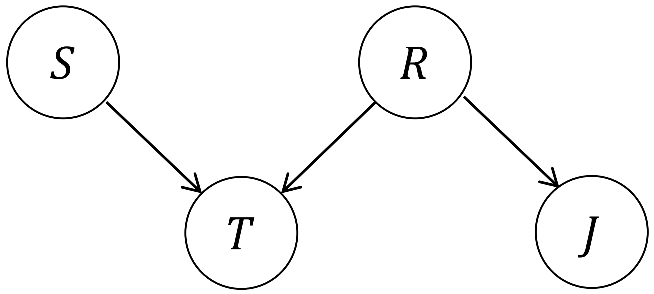
$\Sigma_i = \Sigma$ for all i

Quadratic discriminant analysis

Σ_i for each i

Bayesian Network

Example : Wet Grass



$R \in \{0,1\}$: $R = 1$ means that it has been raining
$S \in \{0,1\}$: $S = 1$ Sprinkler is turned on
$J \in \{0,1\}$: $J = 1$ Jack's grass is wet
$T \in \{0,1\}$: $T = 1$ Tracey's grass is wet

Joint distribution based on chain rule

$$\begin{aligned} p(T,J,R,S) &= p(T|J,R,S)p(J,R,S) \\ &= p(T|J,R,S)p(J|R,S)p(R,S) \\ &= p(T|J,R,S)p(J|R,S)p(R|S)p(S) \end{aligned}$$

$$8 + 4 + 2 + 1 = 2^4 - 1 = 15 \text{ parameters are required}$$

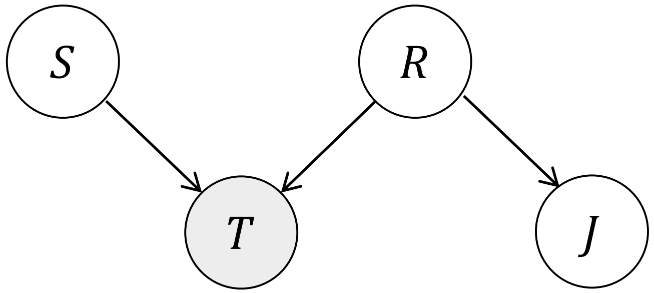
Joint distribution conditional independence

$$\begin{aligned} p(T,J,R,S) &= p(T|J,R,S)p(J|R,S)p(R|S)p(S) \\ &= p(T|R,S) \times p(J|R) \times p(R) \times p(S) \\ &= p(T|R,S)p(J|R)p(R)p(S) \end{aligned}$$

$$\begin{tikzcd} S \arrow[dr] & & R \arrow[dl] \arrow[dr] & \\ & T & & J \end{tikzcd}$$

Example : Wet Grass

Modeling



$R \in \{0,1\} : R = 1$ means that it has been raining
$S \in \{0,1\} : S = 1$ Sprinkler is turned on
$J \in \{0,1\} : J = 1$ Jack's grass is wet
$T \in \{0,1\} : T = 1$ Tracey's grass is wet

$$p(T, J, R, S) = p(T|R, S)p(J|R)p(R)p(S)$$

$p(T|S, E)$

Tracey's Grass wet=1	Rain	Sprinkler
1	1	1
1	1	0
0.9	0	1
0	0	0

$p(J|R)$

Jack's Grass wet=1	Rain
1	1
0.2	0

$p(S = 1) = 0.1$

$p(R = 1) = 0.2$

The tables and graphical structure fully specify the distribution

Example : Wet Grass

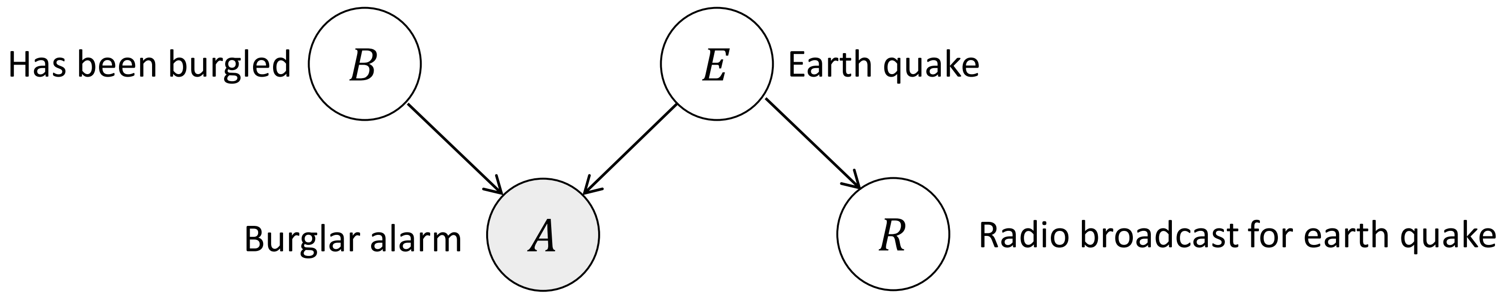
Inference

$$
\begin{aligned}
p(S=1|T=1) &= \frac{p(S=1,T=1)}{p(T=1)} = \frac{\sum_{J,R} p(T=1,J,R,S=1)}{\sum_{J,R,S} p(T=1,J,R,S)} \\
&= \frac{\sum_{J,R} p(J|R) p(T=1|R,S=1) p(R) p(S=1)}{\sum_{J,R,S} p(J|R) p(T=1|R,S) p(R) p(S)} \\
&= \frac{\sum_{R} p(T=1|R,S=1) p(R) p(S=1)}{\sum_{R,S} p(T=1|R,S) p(R) p(S)} \qquad \because \sum_J p(J|R) = 1 \\
&= \frac{0.9\times 0.8\times 0.1 + 1\times 0.2\times 0.1}{0.9\times 0.8\times 0.1 + 1\times 0.2\times 0.1 + 0\times 0.8\times 0.9 + 1\times 0.2\times 0.9} = 0.3382
\end{aligned}
$$

$$
\begin{aligned}
p(S=1|T=1,J=1) &= \frac{p(S=1,T=1,J=1)}{p(T=1,J=1)} \\
&= \frac{\sum_{R} p(T=1,J=1,R,S=1)}{\sum_{R,S} p(T=1,J=1,R,S)} \\
&= \frac{\sum_{R} p(J=1|R) p(T=1|R,S=1) p(R) p(S=1)}{\sum_{R,S} p(J=1|R) p(T=1|R,S) p(R) p(S)} \\
&= \frac{0.0344}{0.2144} = 0.1604
\end{aligned}
$$

The fact that Jack’s grass is also wet increases the chance that the rain has played a role in making Tracey’s grass wet

Example : Burglar Alarm



$$p(B,E,A,R) = p(A|B,E)p(R|E)p(E)p(B)$$

$p(A|B,E)$

Alarm=1	Burglar	Earthquake
0.9999	1	1
0.99	1	0
0.99	0	1
0.0001	0	0

$p(R|E)$

Radio=1	Earthquake
1	1
0	0

$p(E=1) = 0.01$

$p(E=1) = 0.000001$

$$p(B=1|A=1) = \frac{p(B,A=1)}{p(A=1)} = \frac{\sum_{E,R} p(B=1,E,A=1,R)}{\sum_{B,E,R} p(B,E,A=1,R)}$$

$$= \frac{\sum_{E,R} p(A=1|B=1,E)p(R|E)p(E)p(B=1)}{\sum_{B,E,R} p(A=1|B,E)p(R|E)p(E)p(B)} \approx 0.99$$

$p(B=1|A=1,R=1) \approx 0.01$

Conditional Independence

What causes the number of parameters to be reduced?

→ The conditional independence assumptions encoded by the structure of a Bayesian network

- X and Y are independent if and only if

$$P(X,Y) = P(X)P(Y)$$

or equivalently $P(X|Y) = P(X)$

$$\because P(X|Y) = \frac{P(X,Y)}{P(Y)} = \frac{P(X)P(Y)}{P(Y)} = P(X)$$

- X and Y are conditionally independent given Z if and only if

$$P(X,Y|Z) = P(X|Z)P(Y|Z)$$

or equivalently $P(X|Z) = P(X|Y,Z)$

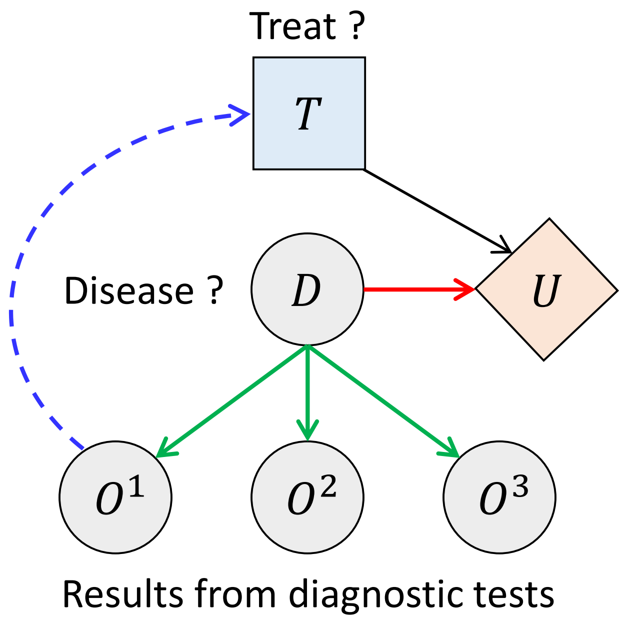
Independence assumptions reduce the number of parameters used to represent a joint pdf

Influential Diagram

Introduction

Bayesian Network + **Decision node** + **Utility node** = **Decision network (Influential Diagram)**

- make rational decisions based on a probabilistic model and utility function



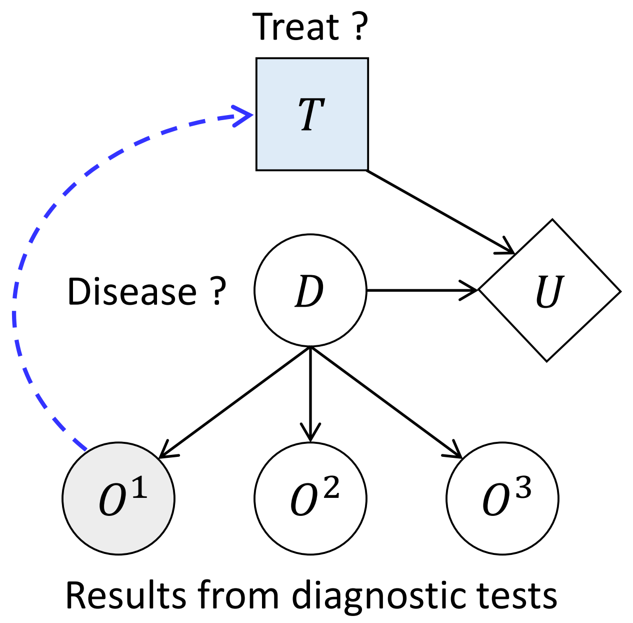
● **A chance node** corresponds to a random variable

■ **A decision node** corresponds to each decision to be made

◆ **A utility node** corresponds to an additive utility component

Decision Network

Assume we only have a single observation $O^1 = 1 (= o_1^1)$ from test 1



$$EU(t^1|o_1^1) = \sum_{o_3}\sum_{o_2}\sum_{d} P(d, o_2, o_3|t^1, o_1^1)U(t^1, d, o_1^1, o_2, o_3)$$

$$\approx \sum_{d} \underbrace{P(d|t^1, o_1^1)}_{\text{Can be computed using many inference methods}} U(t^1, d)$$

Compare $EU(t^0|o_1^1)$ and $EU(t^1|o_1^1)$ and chose the treatment that lead maximum EU

Value of Information

- It may be beneficial to administer additional diagnostic tests to reduce the uncertainty about the decease. Then, how to choose a test type to be conducted?

- Expected utility of optimal action given observation o *:*

$$EU^*(o) = \underset{a}{\operatorname{argmax}}\, EU(a|o)$$

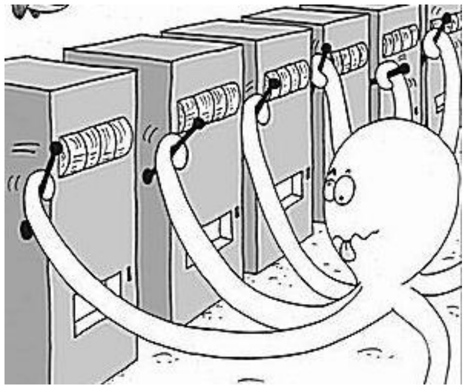
- The value of information (VPI) about new variable O^{new} (unobserved) given the current observation o (observed)*:*

$$\mathrm{VOI}(O^{new}|o) = \left(\sum_{o^{new}} P(o^{new}|o)EU^*(o^{new}, o)\right) - EU^*(o)$$

- The value of information about a variable is the increase in expected utility with the observation of that variable
- VPI can only captures the increase in expected utility→ need to consider the cost associated with observing the new information

Bandit Problem

An n-Armed Bandit Problem



- There are n machine
- Each machine i returns a reward $r \sim P(\theta_i)$: θ_i is unknown
- $a_t \in \{1, \dots, n\}$: the choice of machine at time t
- r_t : the reward at time t
- Policy π maps all the history to new action:

$$\pi: \ [(a_1, r_1), (a_2, r_2), \dots, (a_{t-1}, r_{t-1})] \to a_t$$

Find the optimal policy π^* that maximizes $\mathrm{E}[\sum_{t=1}^{T} r_t]$ or $\mathrm{E}[r_T]$

Acquiring new information
(exploration)

trade-off

capitalizing on the information available so far
(exploitation)

Internet add : Advertising showing strategy for users
Finance : Portfolio optimization under unknown return profiles (risk vs mean profit)
Health care : Choosing the best treatment among alternatives
Internet shopping : Choosing the optimum price (sales v.s. profits)
Experiment design : Sequential experimental design (or sequential simulation parameter)

Action-Value Methods

- If at tth play, action a has been chosen k_a times prior to t, yielding rewards, $r_1, r_2, \dots, r_{k_a}$, the estimated action value for a is defied as

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}$$

If $k_a = 0, Q_t(a) = 0$
If $k_a = \infty, Q_t(a) \to Q^*(a)$

greedy action selection rule

$$a_t = \underset{a}{\operatorname{argmax}}\, Q_t(a)$$

- ✓ Always exploits current knowledge to maximize immediate reward
- ✓ spends no time at all sampling apparently inferior actions to see if they might really be better

$\epsilon -$ *greedy* action selection rule

$$\pi(a) = \begin{cases} 1 - \epsilon + \epsilon/|A(s)| & \text{If } a = a* \\ \epsilon/|A(s)| & \text{If } a \neq a* \end{cases}$$

$$\left(1 - \epsilon + \frac{\epsilon}{|A(s)|}\right) \times 1 + \frac{\epsilon}{|A(s)|}(|A(s)| - 1) = 1$$

- ✓ In the limit as the number of plays increases, every a will be sampled an infinite number of times, guaranteeing $k_a \to \infty$ thus $Q_t(a) \to Q^*(a)$
- ✓ The probability of selecting the optimum action converges to greater than $1 - \epsilon$

Softmax Action Selection

Disadvantage in $\epsilon - greedy$ action selection:

- When it explores it chooses equally among all actions. That is it is as likely to chose the worst-appearing action as it is to choose the next-to best action

Solution:

- Vary the action probabilities as a graded function of estimated value
- The greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their values estimates

Softmax action selection rule

It chooses action a on the tth play with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

✓ τ is a positive parameter called the temperature:

- high $\tau \rightarrow$ all actions are equiprobable
- low $\tau \rightarrow$ greater difference in selection probability for action
- When $\tau = 0$, softmax selection rule become greedy one

Whether softmax action selection or greedy action selection rule is better is unclear and may depend on the task and on human factors (i.e., setting τ and ϵ)

Reinforcement Comparison

Reinforcement Comparison algorithm

- $p_t(a)$: The preference for each action at time t (not an actual action value)

- Action determination rule (soft max) :

$$\pi_t(a) = \frac{e^{p_t(a)/\tau}}{\sum_{b=1}^{n} e^{p_t(a)/\tau}}$$

- After selecting action and observing reward, the preference $p_t(a)$ is updated :

$$p_{t+1}(a_t) = p_t(a_t) + \beta[r_t - \bar{r}_t]$$

✓ High rewards should increase the probability of reselecting the action taken
✓ β is a positive step-size parameter
✓ The reference reward $\bar{r}_t$ (i.e., average reward) is updated using all recently received rewards whichever actions were taken:

$$\bar{r}_{t+1} = \bar{r}_t + \alpha[r_t - \bar{r}_t]$$

Reinforcement comparison method can be very effective sometimes outperforming $\epsilon - greedy$ method

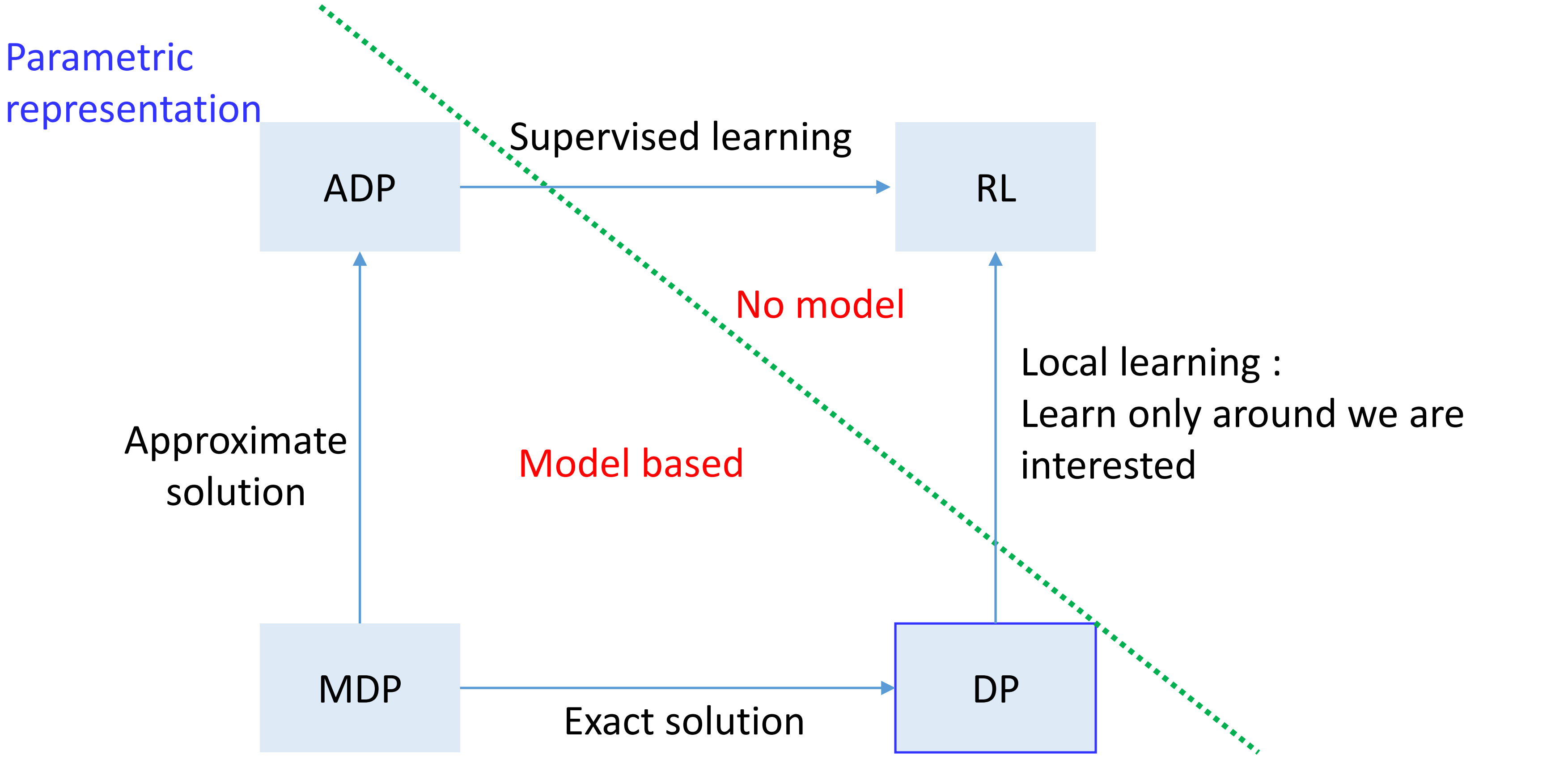
Relationship with model based approach

		Only Exploitation	Exploration vs Exploitation
		Model is known	Model is unknown
Optimum action	Single state	Optimization	Bandit
			Contextual Bandit
Optimum policy	Multiple state	Dynamic Programing	Reinforcement learning

- We going to learn Dynamic Programming approach first, and move to Reinforcement learning

Markov Decision Process

From MDP to RL



- Usually Tabular representation
- For special case, DP can be used for solving a MDP with continuous space and action

- All other methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment

Markov Decision Processes

Finite Markov Decision Process (MDP) :The state and action space are finite

An MDP is defined by:

- A set of states $s \in \mathcal{S}$
- A set of actions $a \in \mathcal{A}$
- A transition function $T(s, a, s') = P(S_{t+1} = s' | S_t = s, A_t = a) = P(s' | s, a)$
  - ✓ Probability that a from s leads to s' when taking action a
  - ✓ Also called the model or the dynamics
- A reward function $R(s, a, s')$
  - ✓ $r_{t+1} = R(s_t, a_t, s_{t+1})$ or $r_{t+1} = R(s_t, a_t)$
  - ✓ If stochastic, $R(s, a, s') = \mathbb{E}[r_t + r_{t+1}, \ldots | S_t = s, A_t = a, S_{t+1} = s']$
- A start state $s_0 \in \mathcal{S}$
- A terminal state $s_T \in \mathcal{S}^+$ (for episodic tasks)

Value Function & Q function

Value function (state value function for π)

"How good it is for the agent to be in a given state"

$V^\pi(s)$: The expected utility received by following policy π from state s

$$V^\pi(s) = \mathbb{E}_\pi(U_t | S_t = s) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \, | S_t = s\right)$$

$\mathbb{E}_\pi$: not expectation over policy π but all stochastic state transitions associated with π

Q-function (action-value function for π)

"How good it is for the agent to perform a given action in a given state"

$Q^\pi(s, a)$: The expected utility of taking action a from state s, and then following policy π

$$Q^\pi(s,a) = \mathbb{E}_\pi(U_t | S_t = s, A_t = a) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \, | S_t = s, A_t = a\right)$$

Because the agent can expect to receive in the future depend on what actions it will take
→ Value and Q functions are defined with respect to a particular policy mapping state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$

The Bellman Equation for Value Function

Recursive Formulation (The Bellman equation)

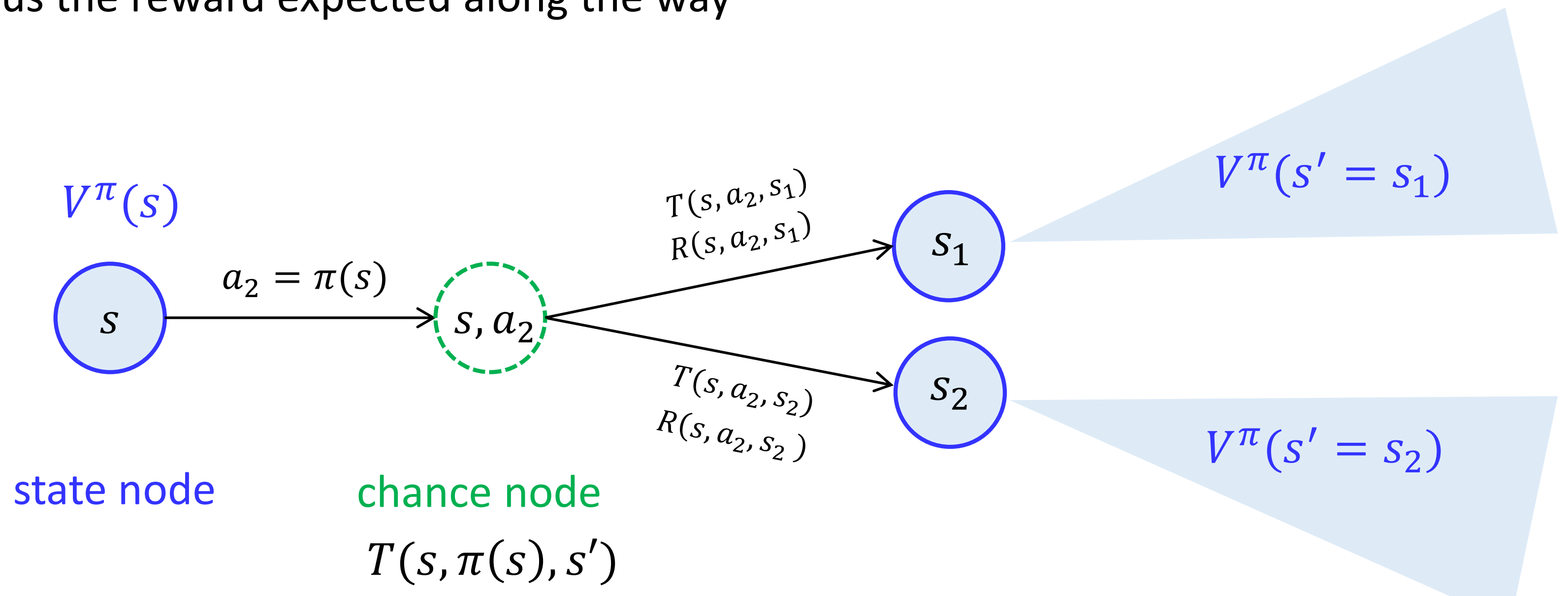
$$V^{\pi}(s) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|S_t = s\right)$$

$$= \mathbb{E}_{\pi}\left(r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \,|S_t = s\right)$$

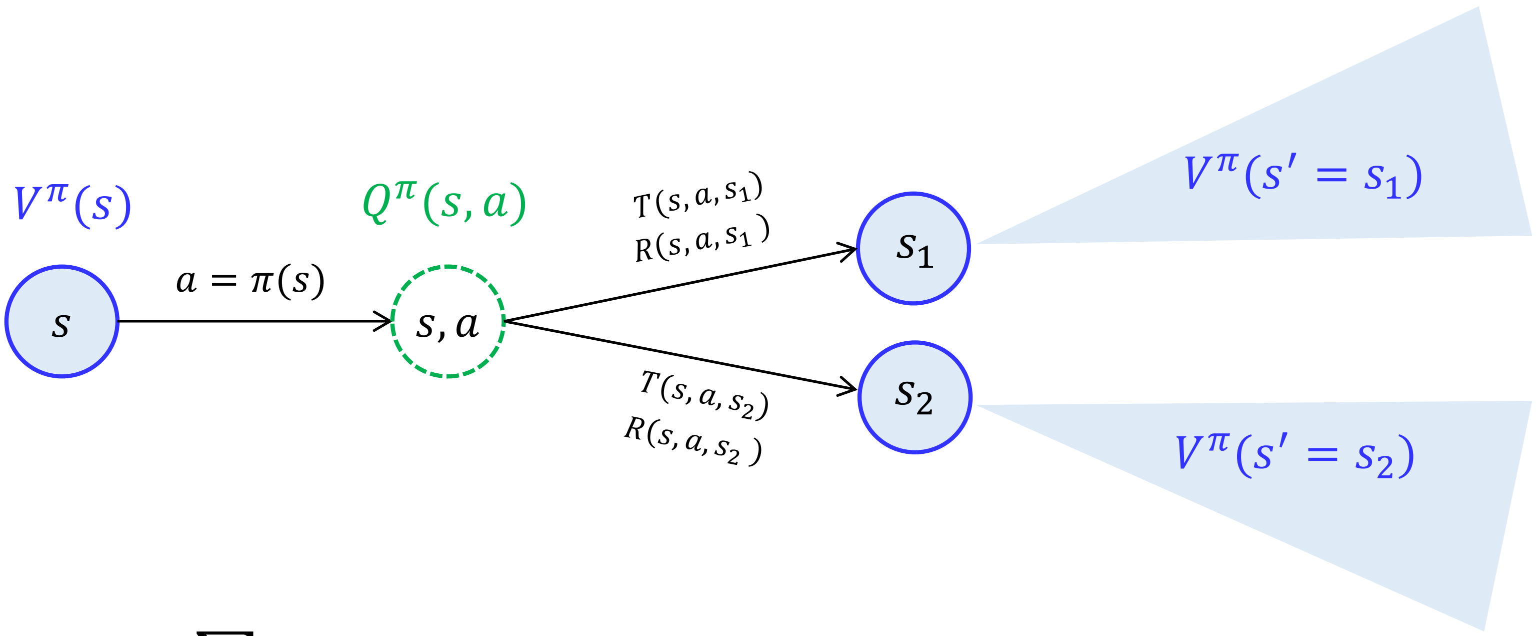
$$= \sum_{s'} T(s,\pi(s),s')\{R(s,\pi(s),s') + \gamma \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \,|S_{t+1} = s'\right)\}$$

$$= \sum_{s'} T(s,\pi(s),s')\{R(s,\pi(s),s') + \gamma V^{\pi}(s')\}$$

The value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way



Summary for Value function and Q function



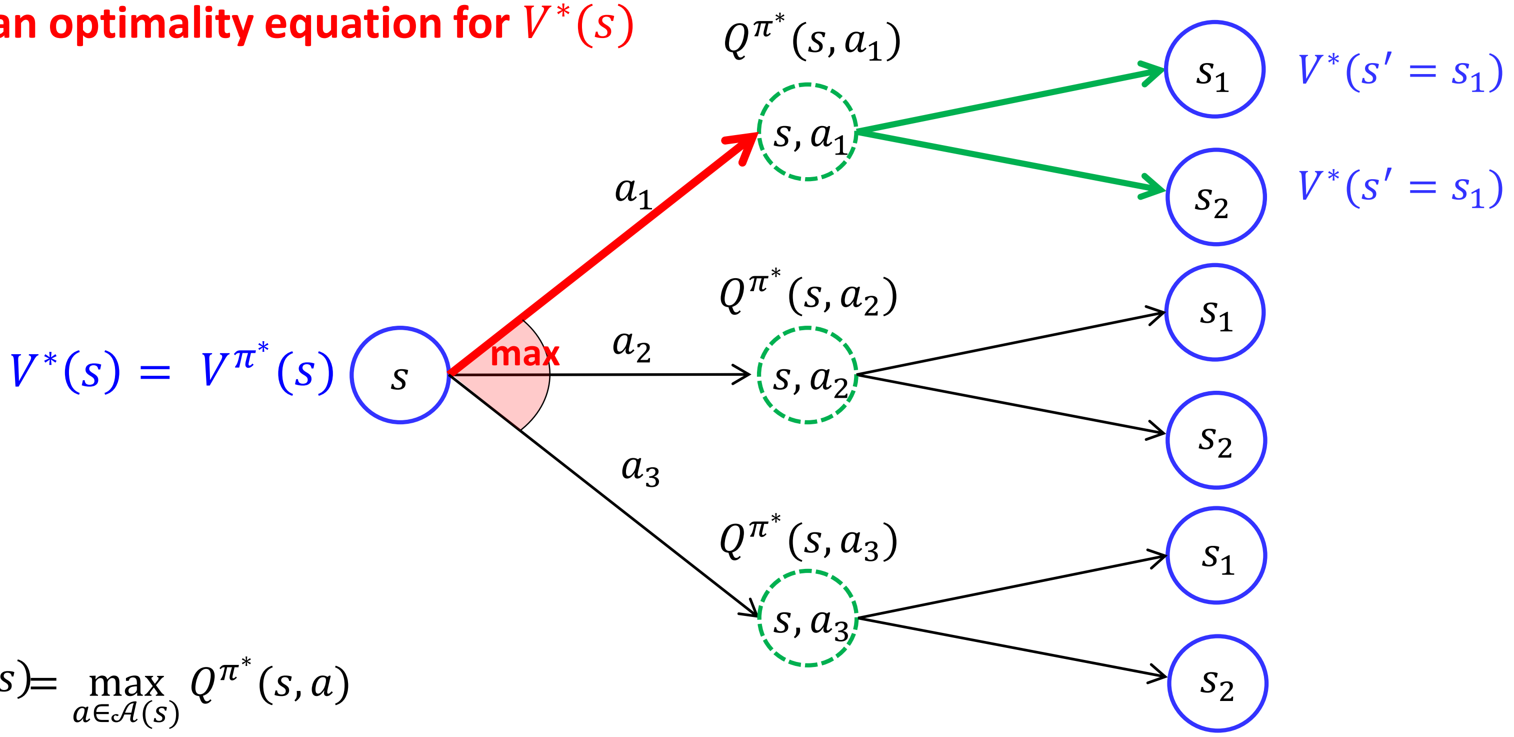
$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')\{R(s,\pi(s),s') + \gamma V^\pi(s')\}$$

$$Q^\pi(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^\pi(s')]$$

$$V^\pi(s) = Q^\pi(s,\pi(s))$$

Bellman Optimality Equation for State-Value Function

Bellman optimality equation for $V^*(s)$



$$V^{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s, a)$$

$$= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi^*}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, S_t = s, A_t = a\right)$$

$$= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi^*}\left(r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, S_t = s, A_t = a\right)$$

$\mathbb{E}$ is over transitions associated with π^*

$$= \max_{a \in \mathcal{A}(s)} \mathbb{E}\left(r_t + \gamma V^{\pi^*}(s_{t+1}) \,\middle|\, S_t = s, A_t = a\right)$$

$\mathbb{E}$ is over transitions

$$= \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s')\{R(s, a, s') + \gamma V^{\pi^*}(s')\}$$

First take optimum action and follow the optimum policy

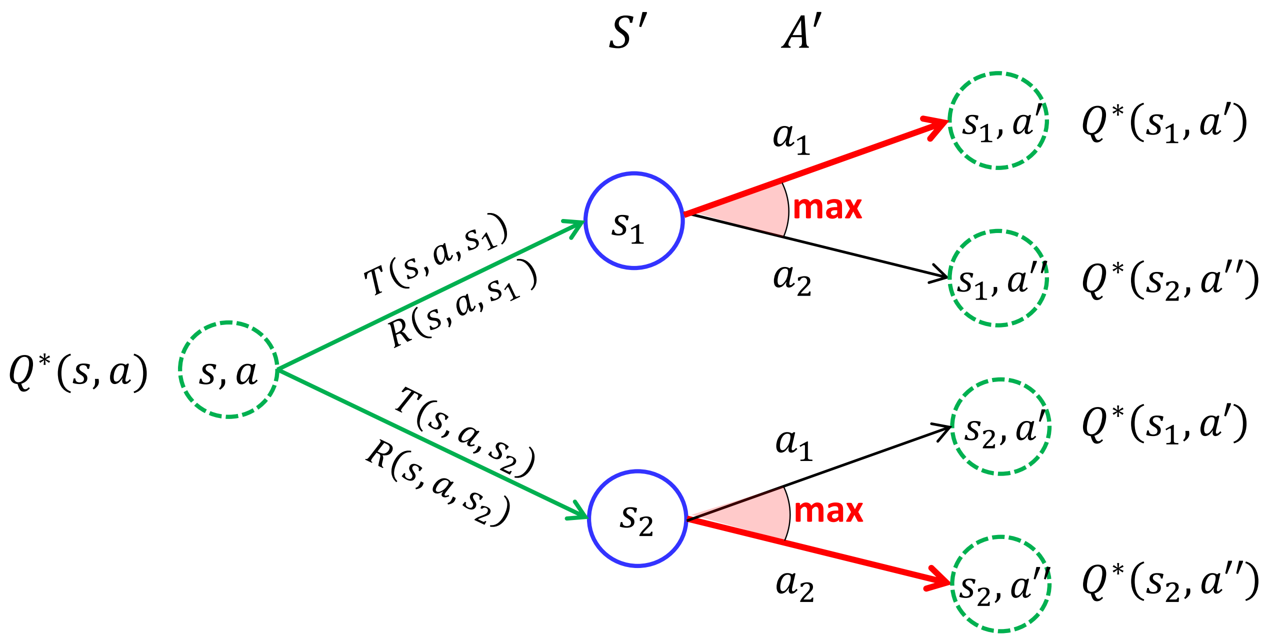
Optimal Value Function & Q function

Bellman optimality equation for $Q^*(s, a)$

$Q^*(s, a) = Q^{\pi^*}(s, a) = \max_{\pi} Q^{\pi}(s, a)$, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Bellman Optimality Equation for State-Action Value Function

Bellman optimality equation for $Q^*(s,a)$



$$Q^*(s,a) = \mathbb{E}\left\{r_t + \gamma \max_{a'} Q^*(s',a') | s_t = s, a_t = a\right\}$$
$$= \sum_{s'} T(s,a,s')\left\{R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right\}$$

$\mathbb{E}$ is over transitions $s \rightarrow s'$

First transits by transition probability and take the optimum action for each consequent states

Optimal Value Function & Q function

Relationships between $Q^*(s,a)$ and $V^*(s)$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s)$$

$$\because\ Q^\pi(s,a) = \mathbb{E}[R(s,a,s') + \gamma V^\pi(s') | s_t = s, a_t = a]$$

$$\begin{aligned} Q^*(s,a) &= \max_\pi Q^\pi(s,a) \\ &= \max_\pi \mathbb{E}[R(s,a,s') + \gamma V^\pi(s') | s_t = s, a_t = a] \\ &= \mathbb{E}\left[R(s,a,s') + \gamma \max_\pi V^\pi(s') \,| s_t = s, a_t = a\right] \\ &= \mathbb{E}[R(s,a,s') + \gamma V^*(s') | s_t = s, a_t = a] \end{aligned}$$

Summary

State-value function & State-action value function allows
Optimum Planning as a Greedy Search!

- **Reconstructing optimal policy with $Q^*(s,a)$ and $V^*(s)$**

$$
\begin{aligned}
a^* &= \underset{a}{\operatorname{argmax}}\, Q^*(s,a) \\
&= \underset{a}{\operatorname{argmax}}\, \mathbb{E}[R(s,a,s') + \gamma V^*(s') | s_t = s, a_t = a]
\end{aligned}
$$

- Any greedy policy with respect to the optimal value function $V^*(s)$ is an optimal policy

  → because $V^*(s)$ already takes into account the reward consequences of all possible future behavior

- The Q function effectively catches the results of all one-step-ahead search

MDP and Dynamic Programming Approach

Dynamic Programming

- The term **dynamic programming (DP)** refers to a collection of algorithms that can be used to compute optimal polices given a perfect model of the environment as a Markov decision process (MDP)
- The key idea of DP (and reinforcement learning) is the use of value functions to organize and structure the search for good policies
- Optimal policies can be derived from the optimal value functions that satisfy the Bellman optimality equations

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\}$$

$$\begin{aligned} Q^*(s,a) &= \sum_{s'} T(s,a,s')\left\{R(s,a,s') + \gamma \max_{a'} Q^*(s',a')0\right\} \\ &= \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\} \qquad \because V^*(s') = \max_{a'} Q^*(s',a') \end{aligned}$$

Optimal policy

$$\begin{aligned} \pi^*(s) &= \operatorname*{argmax}_a Q^*(s,a) \\ &= \operatorname*{argmax}_a \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\} \end{aligned}$$

DP Approaches

Policy Evaluation

For $t = 1, \ldots$

For each state s:

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

Policy Iteration

Policy Improvement

For each state s:

$$\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi}(s')]$$

Value Iteration

For each state s:

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s')\{R(s, a, s') + \gamma V_t(s)\}$$

Asynchronous Value iteration

For any single state s:

$$V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s')\{R(s, a, s') + \gamma V(s)\}$$

As long as both processes continue to update all states, the ultimate result is typically the same-convergence to the optimal value function and an optimal policy

Policy Evaluation

Policy evaluation :
A method to compute the state-value function $V^{\pi}(s)$ for an arbitrary policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V^{\pi}(s')\}$$

- A system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknown

Algorithm

```
Initialize $V_{t=0}^{\pi}(s) \leftarrow 0$ for all states $s \in S$
Repeat (iteration $t = 0, \ldots$):
    For each state $s$:
        $V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$
Until $\max_{s \in \mathcal{S}} |V_{t+1}^{\pi} - V_t^{\pi}(s)| \leq e$
```

Full backup:
Each iteration of iterative policy evaluation backs up the value of every state once to produce the new approximate value function V_{t+1}^{π}

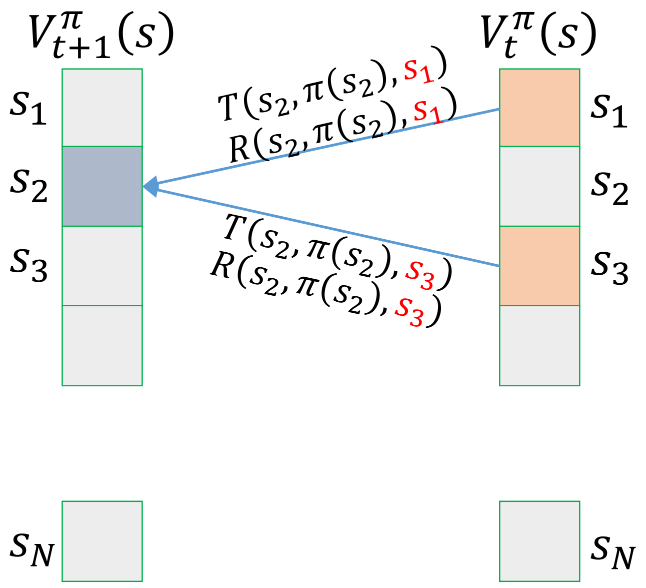
Policy Evaluation

$$V^{\pi}_{t+1}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')\{R(s,\pi(s),s') + \gamma V^{\pi}_{t}(s')\}$$

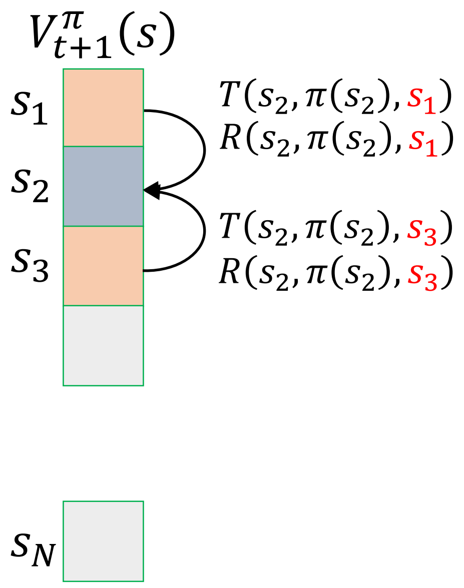
Example:

$$\begin{aligned} V^{\pi}_{t+1}(s_2) &= \sum_{s'} T(s_2,\pi(s_2),s')\{R(s_2,\pi(s),s') + \gamma V^{\pi}_{t}(s')\} \\ &= T(s_2,\pi(s_2),s_1)\{R(s_2,\pi(s),s_1) + \gamma V^{\pi}_{t}(s_1)\} + T(s_2,\pi(s_2),s_3)\{R(s_2,\pi(s_2),s_3) + \gamma V^{\pi}_{t}(s_3)\} \end{aligned}$$

"Two-arrays" update



"In place" update



Usually faster!
Less memory

Policy Improvement

$$\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \, Q^{\pi}(s, a)$$

$$\rightarrow Q^{\pi}(s, \pi'(s)) \geq Q^{\pi}\big(s, \pi(s)\big) = V^{\pi}(s)$$

$$x^* = \underset{x}{\operatorname{argmax}} \, f(x)$$

$\rightarrow f(x^*) \geq f(x)$ for all x

Improvement criterion =
Expected reward provided by changing one step action and following the original policy

Proof (Policy improvement Theorem)

Policy improvement must give us a strictly better policy $\pi'(s)$ than the older policy $\pi(s)$ except when the original policy is already optimal $\pi(s) = \pi^*(s)$

$$Q^{\pi}\big(s, \pi'(s)\big) \geq V^{\pi}(s) \quad \rightarrow \quad \underbrace{V^{\pi'}(s) \geq V^{\pi}(s) \text{ for all states } s \in \mathcal{S}}_{\pi' \geq \pi}$$

Policy Improvement

Proof (Policy improvement Theorem)

Policy improvement must give us a strictly better policy $\pi'(s)$ than the older policy $\pi(s)$ except when the original policy is already optimal $\pi(s) = \pi^*(s)$

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \rightarrow V^{\pi'}(s) \geq V^\pi(s) \quad \text{for all states } s \in \mathcal{S}$$

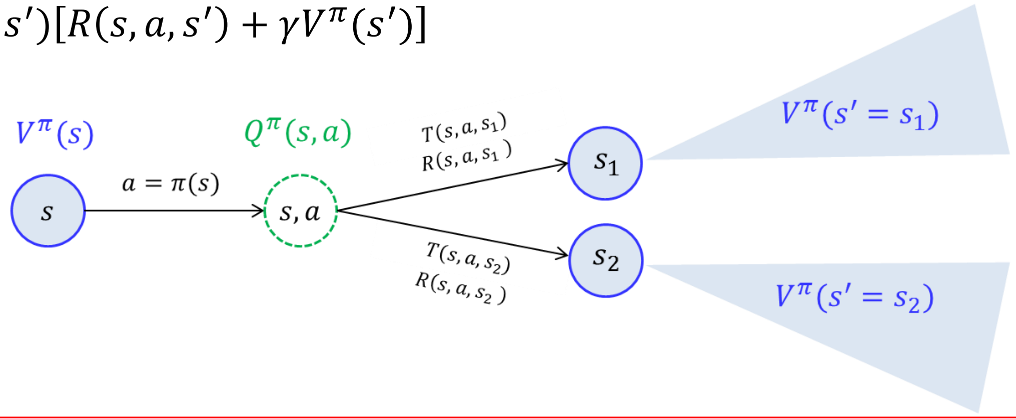
$$
\begin{aligned}
V^\pi(s) &\leq Q^\pi(s, \pi'(s)) && \text{Given} \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s] && \mathbb{E}_{\pi'} \text{ is expectation over } s_{t+1} \text{ induced by } \pi' \\
&\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) | s_t = s] && \because V^\pi(s_{t+1}) \leq Q^\pi(s_{t+1}, \pi'(s_{t+1})) \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \mathbb{E}_{\pi'}[r_{t+2} + \gamma V^\pi(s_{t+2})] | s_t = s] \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) | s_t = s] \\
&\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 Q^\pi(s_{t+2}, \pi'(s_{t+2})) | s_t = s] \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 \mathbb{E}_{\pi'}[r_{t+3} + \gamma V^\pi(s_{t+3})] | s_t = s] \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) | s_t = s] \\
&\quad \vdots \\
&= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots | s_t = s] \\
&= V^{\pi'}(s)
\end{aligned}
$$

Thus, $\pi \leq \pi'$

Policy Improvement

Policy improvement :
The process of making a new policy π^{new} that improves the original policy π, by making it greedy or nearly greedy with respect ot the value function of the original policy

Recall: $$Q^{\pi}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]$$



Algorithm

Input : value of policy $V^{\pi}(s)$

Output: new policy π'

For each state $s \in \mathcal{S}$

~~1. Compute $Q^{\pi}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]$ for each a~~

2. Compute $\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} Q^{\pi}(s,a)$

$$= \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]$$

Policy Iteration

Policy iteration :
Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement

Iteration

```
For $t = 0, \ldots$ until convergence
    For each state $s$:
        $V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')\{R(s,\pi(s),s') + \gamma V_t^{\pi}(s')\}$
```

Converged state value function $V^{\pi}(s)$

For each state s:

$$\pi'(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} Q^{\pi}(s,a)$$
$$= \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]$$

Value Iteration

Value Iteration:
A method to compute the optimum state-value function $V^*(s)$ by combining one sweep of policy evaluation and one sweep of policy improvement

Algorithm

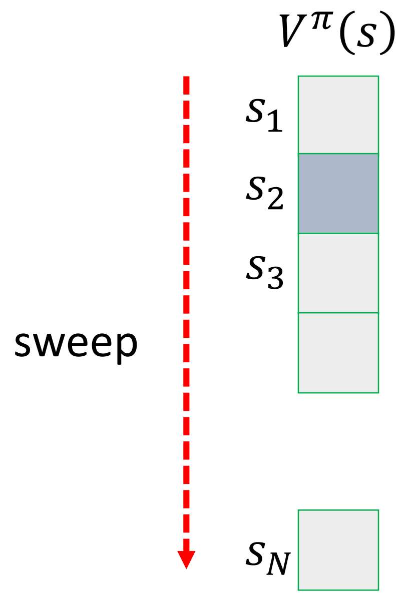
```
Initialize $V(s) \leftarrow 0$ for all states $s \in S$
Repeat
    For each state $s$:
        $V(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V(s)\}$
Until $\max_{s \in S} |V_t(s) - V_{t-1}(s)| \leq e$
```

Optimum policy can be obtained from the converged $V^*(s)$:

$$\pi^*(\mathrm{s}) = \operatorname*{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s,a,s')\{R(s,a,s') + \gamma V^*(s)\}$$

Asynchronous DP Algorithms

A major drawback to the DP methods is that they involve operations over the entire state set of the MDP

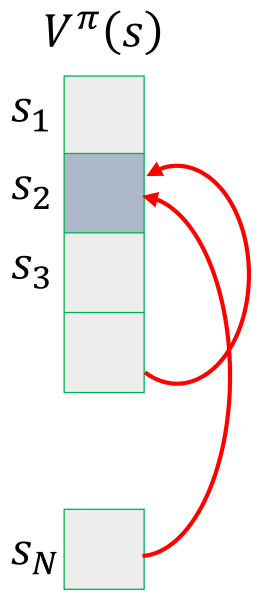


Conventional DP Algorithms

- Black mon game has 10^{20} states
- Go game has $3^{(19\times19)}$ states

…

- Take forever to sweep all states
- Does not improve policy until value functions are full backed up



Asynchronous DP Algorithms

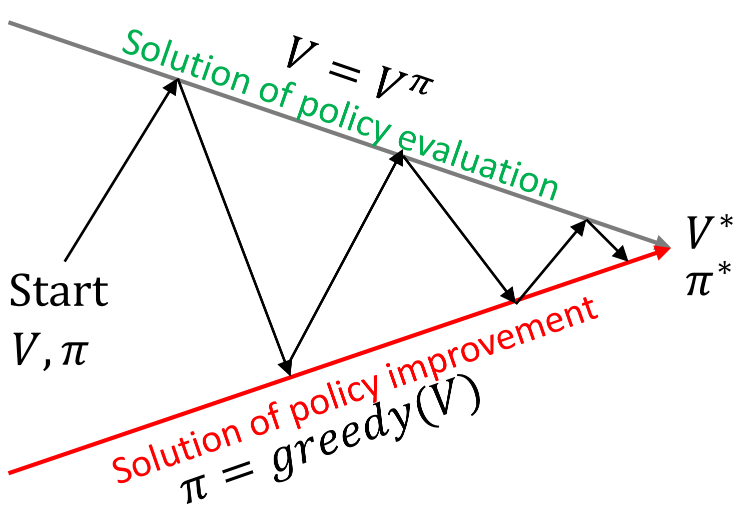
- Back up the values of states in any order whatsoever, using whatever values of other states happen to be available
- Allow great flexibility in selecting states to which backup operations are applied
- Make it easier to intermix computation with real-time interaction: To solve a given MDP, we can run iterative DP algorithm at the same time that an agent is actually experiencing the MDP (Reinforcement Learning !!!!)

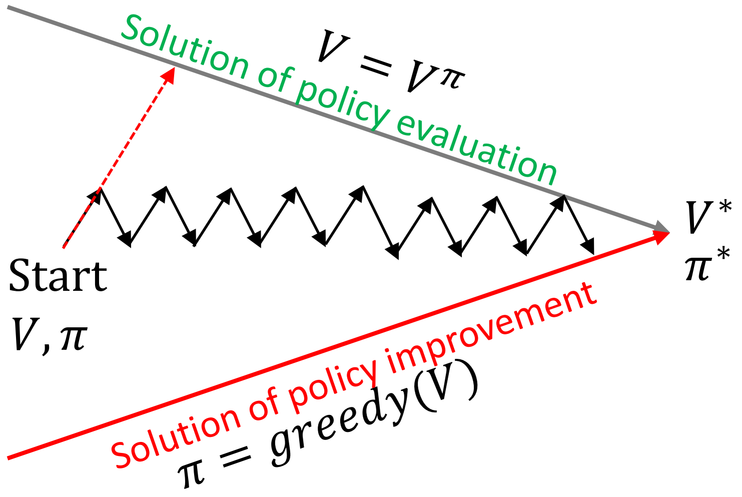
Generalized Policy Iteration

Generalized Policy Improvement (GPI)

Dynamic Programming
"full backup"

Asynchronous
Dynamic Programming





Asynchronous Dynamic Programming is a core concept in Reinforcement learning

Reinforcement Learning
(Monte Carlo Methods)

From MDP to Reinforcement Learning

Markov Decision Process (Offline)

- Have mental model of how the world works
- Find policy to collect the maximum rewards

Solve $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

Find $\pi^*(s) = \max_a Q^*(s,a)$

Reinforcement Learning (Offline & Online)

- Don’t know how the world works
- Perform a sequence of actions in the world to maximize the rewards

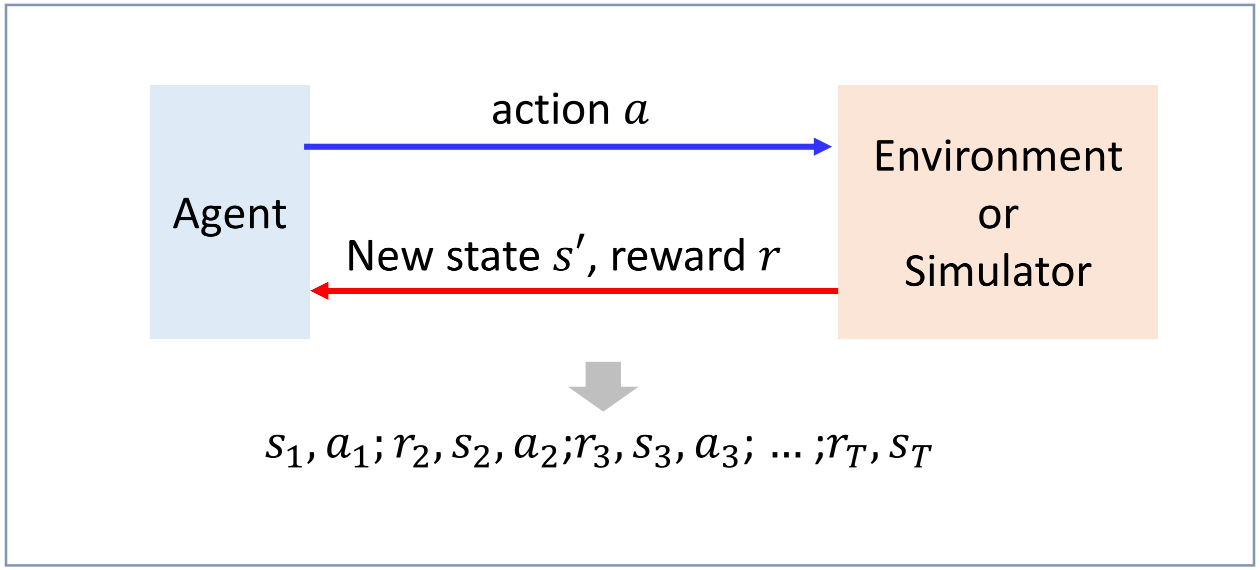
$$s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \ldots ; s_T, a_T, r_T \rightarrow Q^*(s,a) \rightarrow \pi^*(s)$$

- Reinforcement learning is really the way humans work:

→ we go through life, taking various actions, getting feedback.

→ We get rewarded for doing well and learn along the way.

Reinforcement Learning Template


Agent
action a
Environment or Simulator
New state s', reward r
$s_1, a_1; r_2, s_2, a_2; r_3, s_3, a_3; \ldots; r_T, s_T$

Template for Reinforcement Learning

```
For $t = 1, 2, 3, \ldots$
    Choose action $a_t = \pi(s_t)$ (how?) : Decision making
    Receive reward $r_{t+1}$ and observe new state $s_{t+1}$ (Environment)
    Update parameters associated with $V(t)$, $Q(s, a)$ (how?) : Learning
```

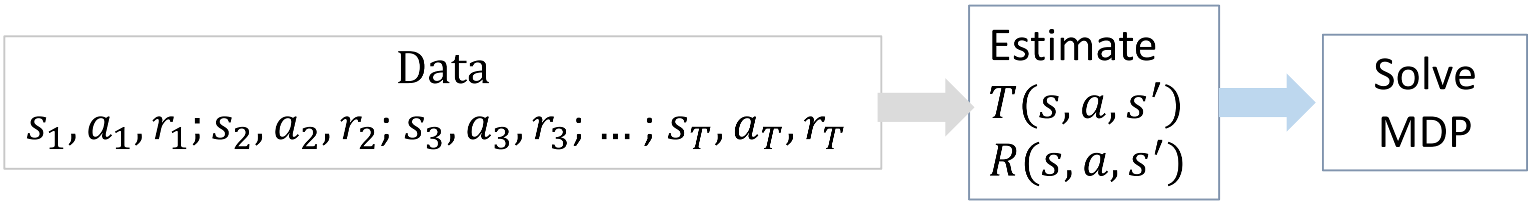
Road Map

- **Monte Carlo Method (Sutton & Barto Ch.5)**
  - Model-Based Monte Carlo method
  - Model-free Monte Carlo method
    - Policy Evaluation
    - Policy Improvement
    - Policy Iteration (Monte Carlo control)
      - ✓ On-policy
      - ✓ Off-policy

- **Temporal Difference Learning (Sutton & Barto Ch.6)**
  - SARSA
  - Q-Learning

Road Map

- **Model-Based Reinforcement learning**


Data
$s_1, a_1, r_1; s_2, a_2, r_2; s_3, a_3, r_3; \dots ; s_T, a_T, r_T$
Estimate
$T(s,a,s')$
$R(s,a,s')$
Solve
MDP

- **Model-FREE Reinforcement learning**

How to estimate $V^*(s)$ and $Q^*(s,a)$

Monte Carlo method | **Temporal Difference methods**

		Non-Bootstrap	Bootstrap
How to explore ?	On-policy	On-policy Monte Carlo Control	SARSA
	Off-policy	Off-policy Monte Carlo Control	Q-Learning

- Episodic based
- Single-data-point based

Model-Free Monte Carlo Based Methods

How to estimate $V^*(s)$ and $Q^*(s,a)$

How to explore ?		Monte Carlo method	Temporal Difference methods
		Non-Bootstrap	Bootstrap
	On-policy	On-policy Monte Carlo Control	SARSA
	Off-policy	Off-policy Monte Carlo Control	Q-Learning

- Episodic based
- Single-data-point based

Monte Carlo Policy Evaluation

Key Idea : Monte Carlo Policy Evaluation

- Learn the state-value function $Q^{\pi}(s,a)$ for a given policy π
- The value of action-state is the expected utility - expected accumulative future reward starting from s and following the policy π

$$s_t \in \mathcal{S} = \{s^1, s^2, s^3\},\ a_t \in \mathcal{A} = \{a^1, a^2, a^3\}$$

$(\text{State, Action, Reward })$ pairs generated by policy π

Episode 1: $(s^1, a^2, 1); (s^3, a^1, 5); (s^2, a^3, 3), (s^1, a^3, 10), (s^2, a^2, 2)$

Episode 2: $(s^1, a^1, 5); (s^2, a^2, 2); (s^1, a^2, 1); (s^2, a^3, 3), (s^1, a^3, 10)$

Episode 3: $(s^2, a^3, 3); (s^1, a^2, 1); (s^3, a^1, 5); (s^1, a^3, 10), (s^2, a^2, 2)$

$(\gamma = 1)$

Episode 1: $Q^{\pi}(s^1, a^2) = 1 + 5 + 3 + 10 + 2 = 21$

Episode 2: $Q^{\pi}(s^1, a^2) = 1 + 3 + 10 = 14$

Episode 3: $Q^{\pi}(s^1, a^2) = 1 + 5 + 10 + 2 = 19$

First visit to s

$Q^{\pi}(s^1, a^2) =$ average of accumulated reward over all episodes

$$= \frac{21+14+19}{3} = 14.6$$

Monte Carlo Control

Key Idea : Monte Carlo Control

- Idea of generalized policy iteration (GPI)
- Monte Carlo Policy Evaluation + Policy improvement

$$\begin{tikzcd}
\boxed{\pi} \arrow[r, shift left, "\substack{\text{Evaluation} \\ Q \to Q^{\pi}}"] & \boxed{Q} \arrow[l, shift left, "\substack{\text{Improvement} \\ \pi \to \text{greedy}(Q)}"]
\end{tikzcd}$$

- *Infinite* number of episodes
- Exploring starts

$$\begin{tikzcd}
\pi_0 \arrow[r, "\text{E}"] & Q^{\pi_0} \arrow[r, "\text{I}"] & \pi_1 \arrow[r, "\text{E}"] & Q^{\pi_1} \arrow[r, "\text{I}"] & \pi_2 \arrow[r, "\text{E}"] & \cdots \arrow[r, "\text{I}"] & \pi_* \arrow[r, "\text{E}"] & Q^*
\end{tikzcd}$$

- **Policy Evaluation**
  - ✓ The value function is repeatedly altered to more closely approximate the value function for the current policy π
- **Policy Improvement**
  - ✓ The policy is repeatedly improved with respect to the current action value function Q

Monte Carlo control algorithm assuming exploring starts

Algorithm : Monte Carlo ES Control

```
Initialize, for all $s \in S, a \in A(s)$
    $Q(s,a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $U(s,a) \leftarrow$ empty list

Repeat forever:
    (a) Generate *an episode* using exploring starts and $\pi$
    (b) For each pair $(s,a)$ appearing in the episode:
            $U \leftarrow$ utility following the first occurrence of $s, a$
            Append $U$ to $U(s,a)$
            $Q(s,a) \leftarrow$ average($U(s,a)$)

    (c) For each $s$ in the episode:
            $\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} Q(s,a)$
```

$\}$ Policy evaluation

$\}$ Policy improvement

In an single episode, both Policy evaluation and policy improvement proceeds together

On-policy Monte Carlo Control

Algorithm : $\epsilon - soft$ On-Policy Monte Carlo Control

```
Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
    $Q(s,a) \leftarrow$ arbitrary
    $\pi \leftarrow$ an arbitrary $\epsilon - soft\ policy$
    $U(s,a) \leftarrow$ empty list

Repeat forever:
    (a) Generate an episode using $\pi$
    (b) For each pair $(s,a)$ appearing in the episode:
        $U \leftarrow$ utility following the first occurrence of $s, a$
        Append $U$ to $U(s,a)$
        $Q(s,a) \leftarrow$ average($U(s,a)$)

    (c) For each $s$ in the episode:
        $a^* \leftarrow \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} Q(s,a)$
        For all $a \in \mathcal{A}(s)$
            $\pi(s,a) = \begin{cases} 1 - \epsilon + \epsilon/|A(s)| & \text{If } a = a* \\ \epsilon/|A(s)| & \text{If } a \neq a* \end{cases}$
```

$\left.\begin{matrix} (a) \\ (b) \end{matrix}\right\}$ Policy evaluation

$\left.\begin{matrix} (c) \end{matrix}\right\}$ Policy improvement

Model-Free Monte Carlo Based Methods

How to estimate $V^*(s)$ and $Q^*(s, a)$

How to explore ?

	Monte Carlo method	**Temporal Difference methods**
	Non-Bootstrap	Bootstrap
On-policy	On-policy Monte Carlo Control	SARSA
Off-policy	Off-policy Monte Carlo Control	Q-Learning

- Episodic based
- Single-data-point based

Off-policy Monte Carlo Control

Key Idea : Off-policy algorithm

- Follows the behavior policy while learning about and improving the estimation policy

- *Behavior* policy $\pi'(\mathrm{s,a})$
  - ✓ The policy used to generate behavior
  - ✓ Requires that the behavior policy have a nonzero probability of selecting all actions that might be selected by the estimation policy (e.g., $\epsilon - soft$ policy)

- *Estimation* policy $\pi(\mathrm{s,a})$
  - ✓ The policy that is evaluated and improved
  - ✓ π can be deterministic
  - ✓ π can be the greedy policy with respect to Q (an estimation Q^{π})

Disadvantages

→ Learning can be slow

Reinforcement Learning
(Temporal Difference)

Model-Free Monte Carlo Based Methods

How to estimate $V^*(s)$ and $Q^*(s,a)$

How to explore ?		**Monte Carlo method** Non-Bootstrap	**Temporal Difference methods** Bootstrap
	On-policy	On-policy Monte Carlo Control	SARSA
	Off-policy	Off-policy Monte Carlo Control	Q-Learning

- Episodic based
- Single-data-point based

Introduction

Dynamic Programming (DP) Methods

pros: Update estimates based in part on other learned estimates, without waiting for a final outcome (bootstrap)

cons: Need explicit model

Monte Carlo (MC) Methods

pros: Learn directly from raw experience without a model

cons: Need to wait until the end of episode to observe expected reward

Temporal-Difference (TD) Learning

pros: Learn directly from raw experience without a model MC

Model free

+

pros: Update estimates based in part on other learned estimates, without waiting for a final outcome (bootstrap) DP

On line Incremental

TD Generalized Policy iteration for

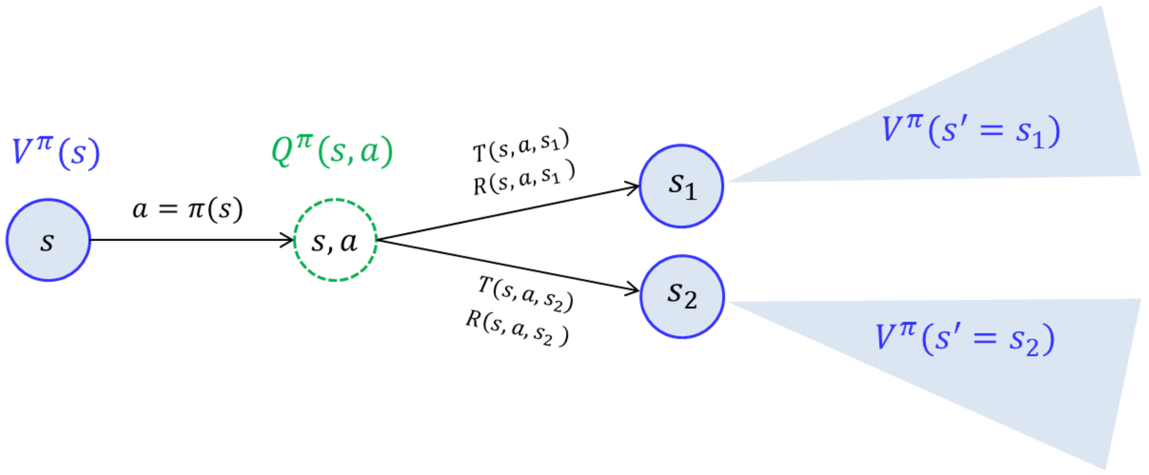
TD Policy Evaluation + **TD Policy Improvement**

On-Policy TD Control (SARSA)

Off-Policy Q-learning Control

Recall : Value function

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi(U_t | s_t = s) \\
&= \mathbb{E}_\pi(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s) \quad \text{Complete episode} \\
&= \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s\right) \\
&= \mathbb{E}_\pi\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,\middle|\, s_t = s\right) \\
&= \mathbb{E}_\pi(r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s) \quad \text{Bootstrapping}
\end{aligned}
$$



Temporal Difference Policy Evaluation

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}(U_t | s_t = s) \\
&= \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s\right) \\
&= \mathbb{E}_{\pi}\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s\right) \\
&= \mathbb{E}_{\pi}(r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s)
\end{aligned}
$$

Bootstrapping

Temporal Difference Policy Evaluation ;$TD(0)$:

After visiting s_t and transiting to s_{t+1} with a singe reward r_{t+1}

$$V(s_t) \leftarrow V(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{Target}} - V(s_t)]$$

- Bootstrapping: the TD method updates the state value using the previous estimations
- The TD target is an estimate because
  - ✓ it uses the current estimate of $V(s_t)$,
  - ✓ it samples the expected value

$$\mathbb{E}_{\pi}(r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s)$$

Temporal Difference Policy Evaluation

Algorithm : Tabular $TD(0)$ for estimating V^{π}

```
Initialize $V(s)$ arbitrarily, $\pi$ to the policy to be evaluated

Repeat (for each episode):
        Initialize $s$
        Repeat (for each step of episode)
        $a \leftarrow$ action given by $\pi$ for $s$
        Take action $a$; observe reward $r$ and next state $s'$
        $V(s) \leftarrow \mathrm{V(s)} + \alpha[r + \gamma V(s') - V(s)]$
        s ←s'
        Until $s$ is terminal
```

- Simple backups (MC method and TD methods) : Use a single sample success state

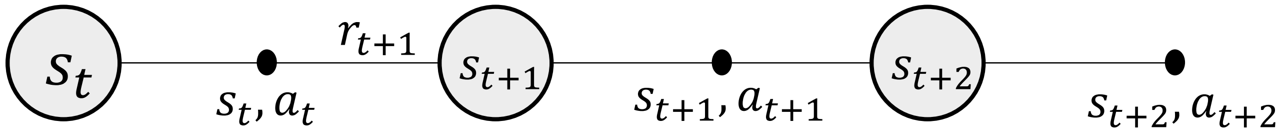
Recall:

- Full Backups (DP approach) : Use complete distribution of all possible successors

$$V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')\{R(s,\pi(s),s') + \gamma V^{\pi}(s')\}$$

Temporal Difference Policy Evaluation for Q function

As we estimate state value $V(s)$, we can estimate $Q(s, a)$ using a TD method



Temporal Difference Policy Evaluation for $Q(s, a)$ function

On each $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ for a single episode:

Note that the action taken is given as data

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\underbrace{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})}_{\text{Target}} - \underbrace{Q(s_t, a_t)}_{\text{Current estimate}}]$$

Model-Free Monte Carlo Based Methods

How to estimate $V^*(s)$ and $Q^*(s,a)$

How to explore ?		**Monte Carlo method**	**Temporal Difference methods**
		Non-Bootstrap	Bootstrap
	On-policy	On-policy Monte Carlo Control	SARSA
	Off-policy	Off-policy Monte Carlo Control	Q-Learning

- Episodic based
- Single-data-point based

Sarsa: On-Policy TD Control

SARSA Algorithm

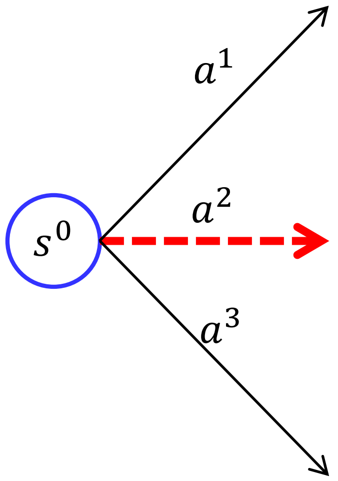
```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon - greedy$)
    Repeat (for **each time step** of episode):
        Take action $a$ given $s$, observe $r, s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\epsilon - greedy$)
        $Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \eta\big(r + \gamma Q_\pi(s',a') - Q_\pi(s,a)\big)$
        $s \leftarrow s'; a \leftarrow a';$
    Until $s$ is terminal
```

Behavioral policy
=
Estimation policy

- As in all on-policy methods, we continually estimate Q^π for the behavioral policy, and the same time change π toward greediness with respect to Q^π
- Converges with
  - ✓ All state-action pairs are visited an infinite number of times
  - ✓ The policy converges in the limit to the greedy policy (i.e., $\epsilon - greedy$ with $\epsilon = 1/t$)

Sarsa: On-Policy TD Control

S_t A_t R_{t+1} S_{t+1} A_{t+1} S_{t+2}

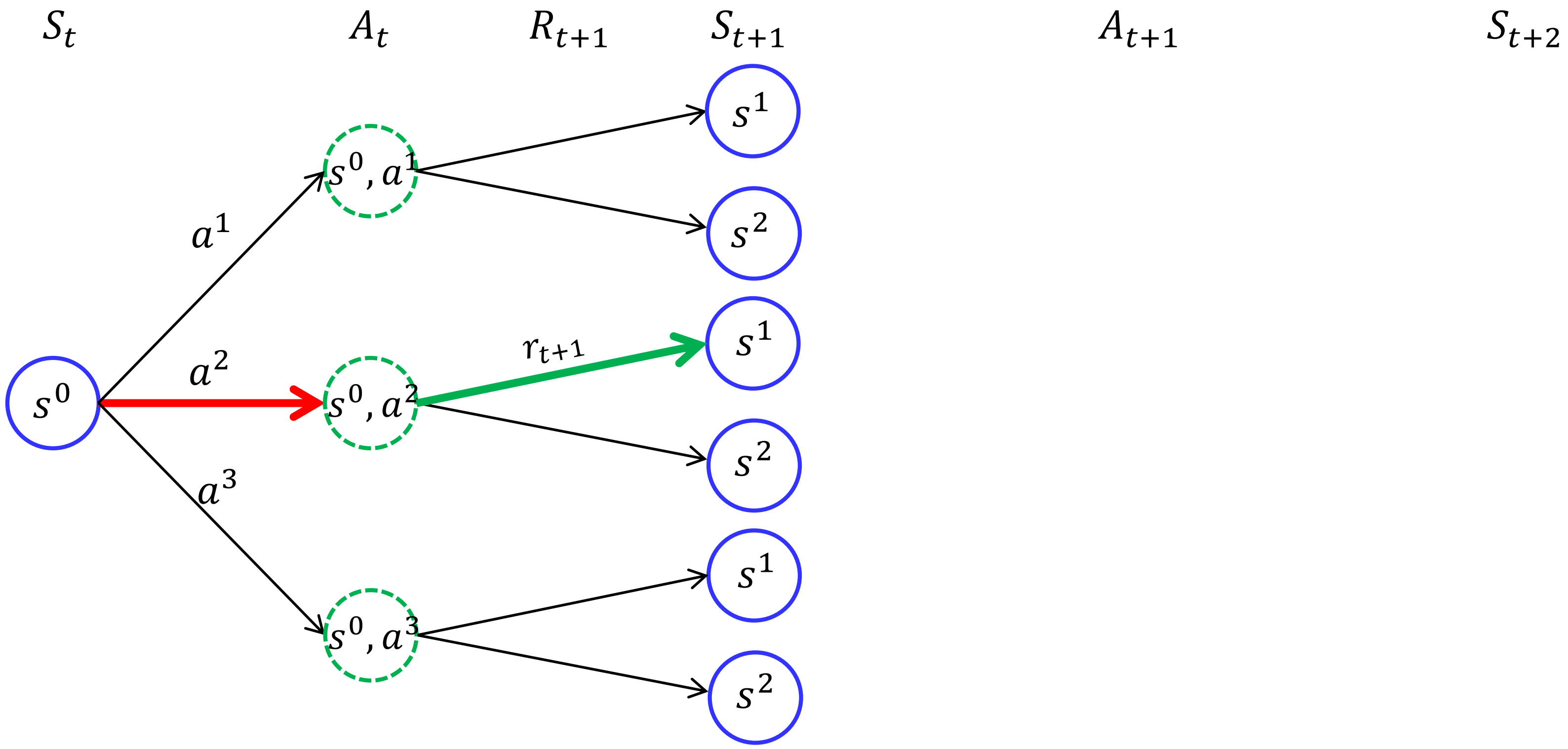


Choose a_t from $s_t = s^0$ using current Q

$$a_t = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(s_t = s^0, a) & \text{with prob } 1-\epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

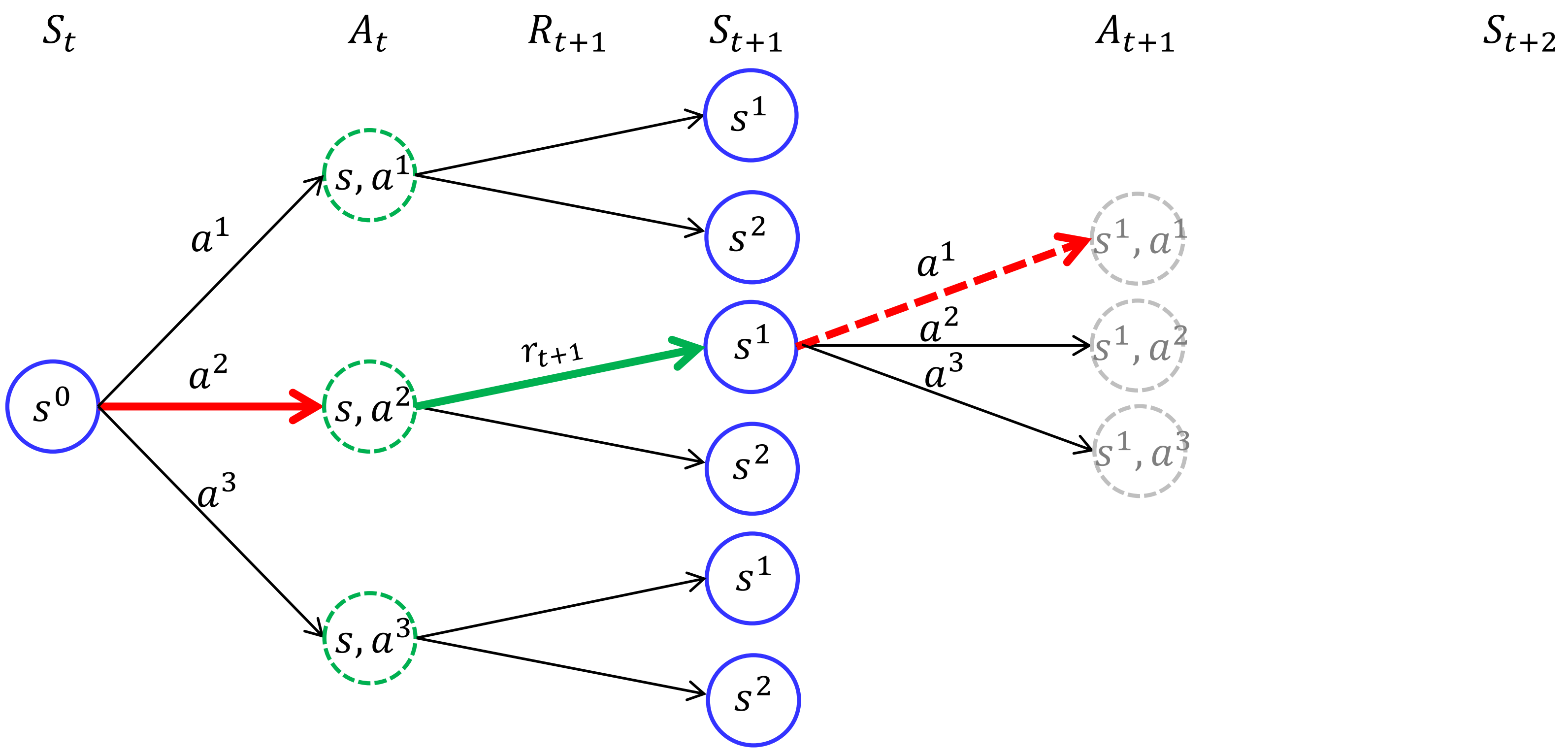
Assume a^2 is chosen

Sarsa: On-Policy TD Control



Take action $a_t = a^2$ given $s_t = s^0$ and observe r_{t+1} and $s_{t+1} = s^1$

Sarsa: On-Policy TD Control

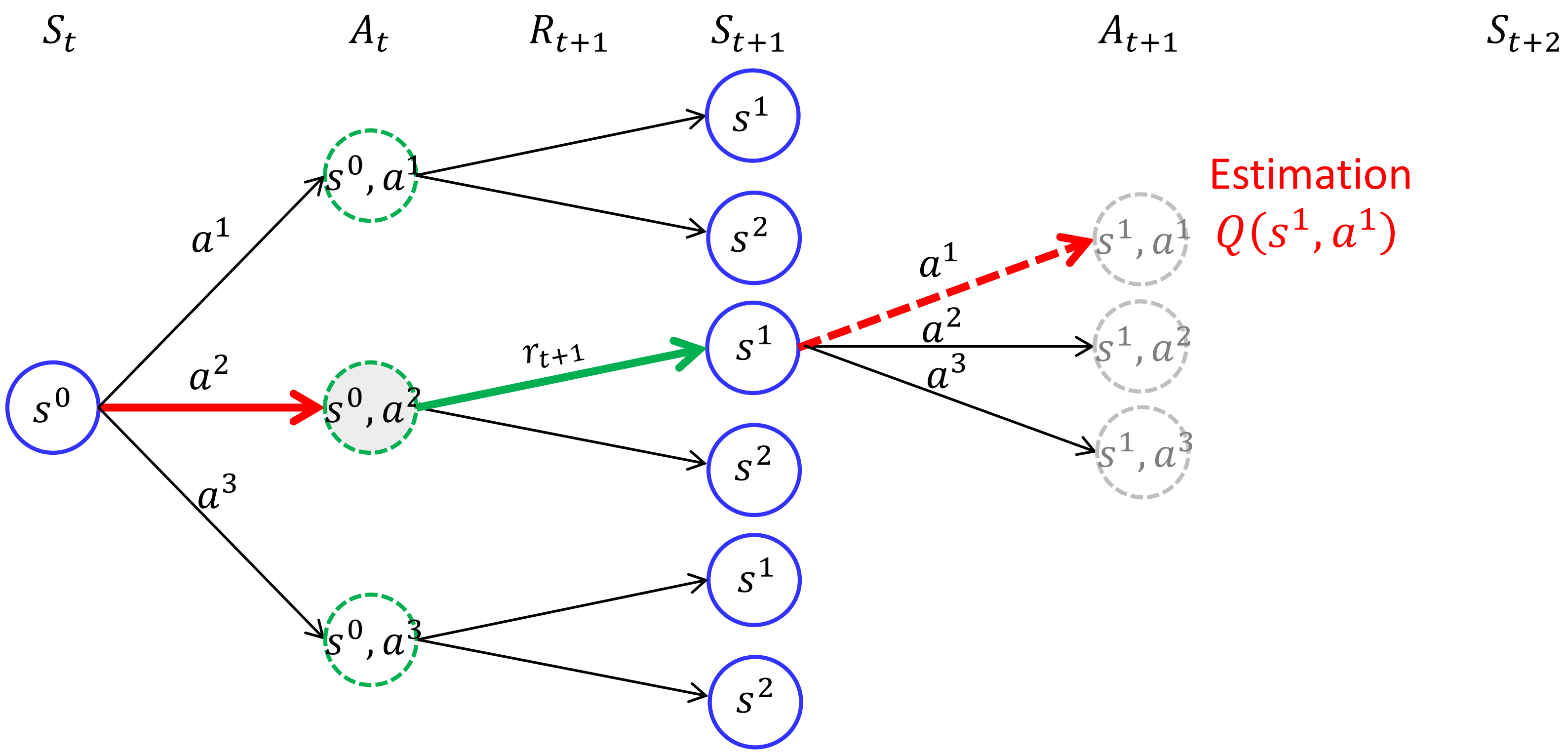


Choose a_{t+1} from $s_{t+1} = s^1$ using current Q

$$a_{t+1} = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(s_{t+1} = s^1, a) & \text{with prob } 1-\epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

Assume a^1 is chosen

Sarsa: On-Policy TD Control

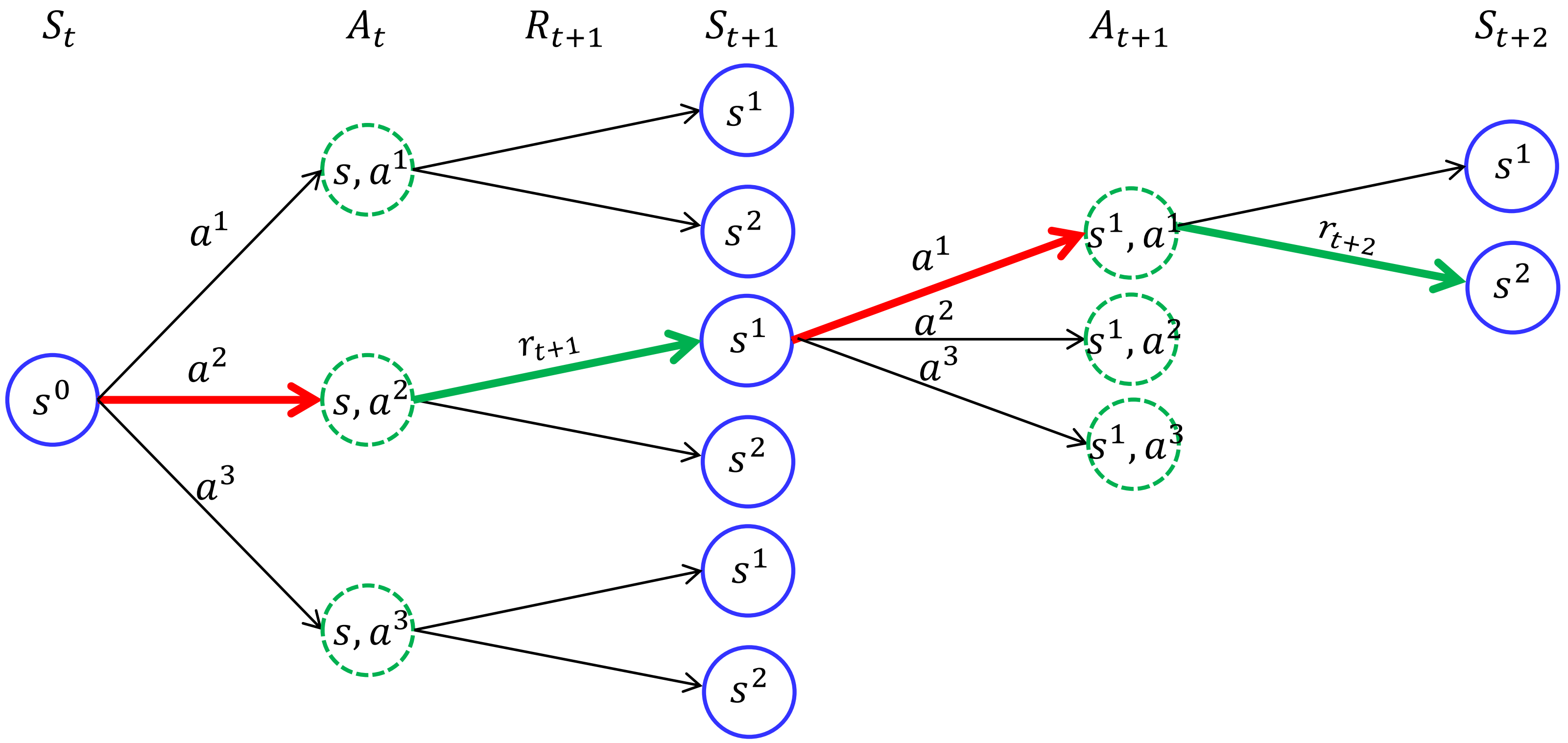


Update Q function with the estimation $Q(s_{t+1}, a_{t+1})$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

$$\rightarrow Q(s^0, a^2) \leftarrow Q(s^0, a^2) + \alpha[r_{t+1} + \gamma Q(s^1, a^1) - Q(s^0, a^2)]$$

Sarsa: On-Policy TD Control



Take action $a_{t+1} = a^1$ given $s_{t+1} = s^1$ and observe r_{t+2} and $s_{t+2} = s^2$

Why Q-learning is considered as Off-Policy method

$$\begin{tikzcd}
 & \pi(s) \arrow[d] & & & \pi(s') \arrow[d] & \\
(s \arrow[ur, bend left] & a & r & s' \arrow[ur, bend left] \arrow[d] & a' \arrow[d] &) \\
Q(s,a) \arrow[uur] & \leftarrow Q(s,a) + \eta\big(r + \gamma Q(& & s' & a' &) - Q_\pi(s,a)\big)
\end{tikzcd}$$

Classification of RL

How to estimate $V^*(s)$ and $Q^*(s,a)$

How to explore ?		Monte Carlo method	Temporal Difference methods
		Non-Bootstrap	Bootstrap
	On-policy	On-policy Monte Carlo Control	SARSA
	Off-policy	Off-policy Monte Carlo Control	Q-Learning (SARSmaxA)
		• Episodic based	• Single-data-point based

Q-Learning: Off-Policy TD Control

On-Policy TD Control (SARSA)

Choose a' from s' using policy derived from Q (e.g., $\epsilon - greedy$)

$$Q(s,a) \leftarrow Q(s,a) + \eta\left(r + \gamma Q(s',a') - Q(s,a)\right)$$

Off-Policy TD Control (Q-learning)

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

- The max over a rather than taking the a based on the current policy is the principle difference between Q-learning and SARSA.
- The learned action-value function Q directly approximates Q^*independent of the policy being followed
- Converges with
  - ✓ All state-action pairs are visited an infinite number of times
  - ✓ The policy converges in the limit to the greedy policy (i.e., $\epsilon - greedy$ with $\epsilon = 1/t$)

Q-Learning: Off-Policy TD Control

Q learning

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each time step of episode):
        Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon - greedy$)  Behavioral policy
        Take action $a$, observe $r$, $s'$
        $Q(s,a) \leftarrow Q(s,a) + \eta \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
        $s \leftarrow s'$
    Until $s$ is terminal
```

Estimation policy

(Always try to estimate the optimal policy)

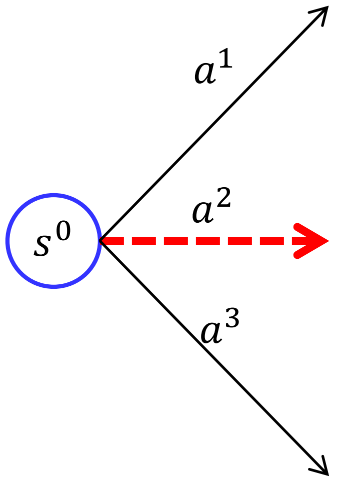
-Estimation can be greedy)

$a^* = \underset{a'}{\operatorname{argmax}} Q(s',a)$ is not used in the next state!!!

At the next state s', Choose a using policy derived from Q (e.g., $\epsilon - greedy$)

Q-Learning: Off-Policy TD Control

S_t A_t R_{t+1} S_{t+1} $\max A_{t+1}$ S_{t+2}

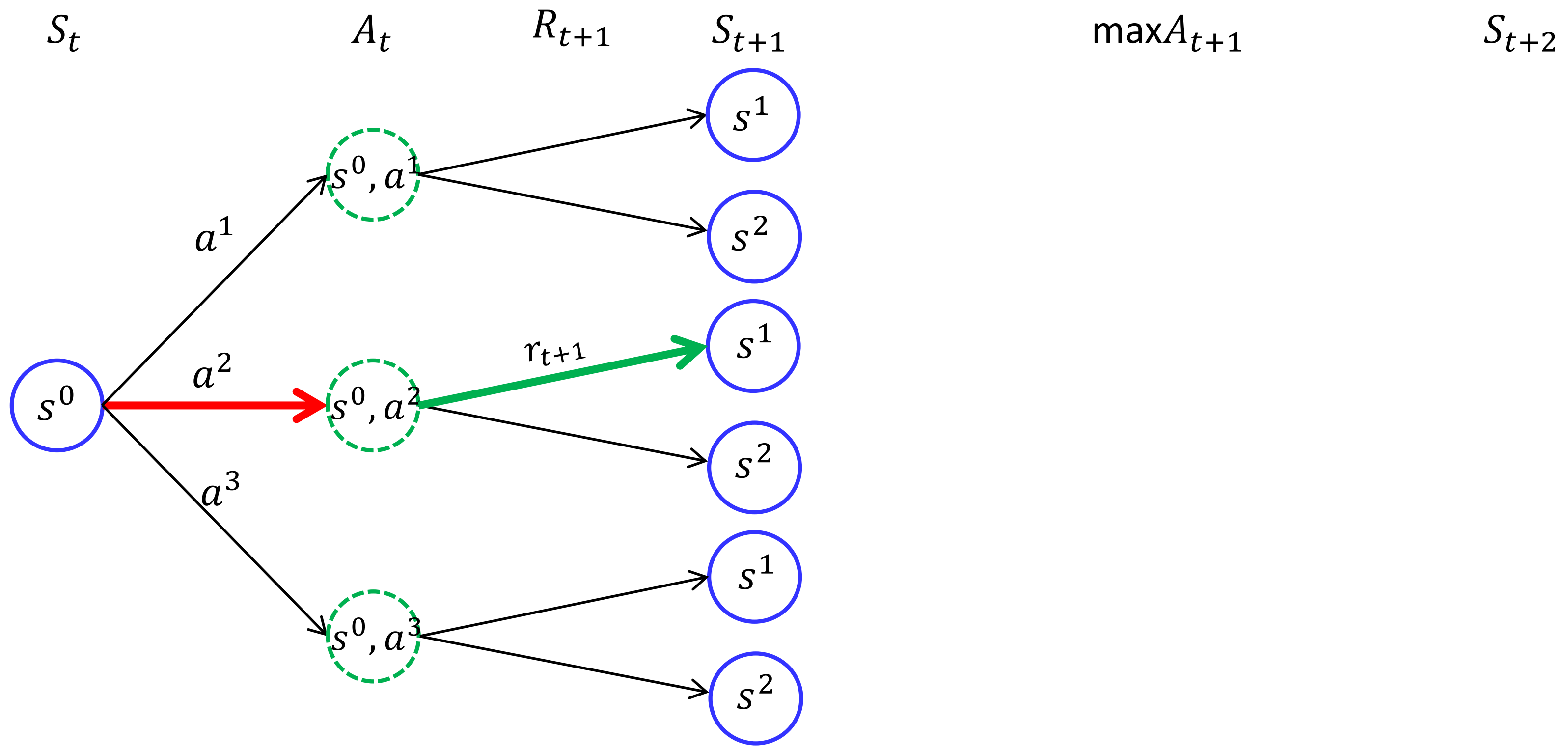


Choose a_t from $s_t = s^0$ using current Q

$$a_t = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(s_t = s^0, a) & \text{with prob } 1-\epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

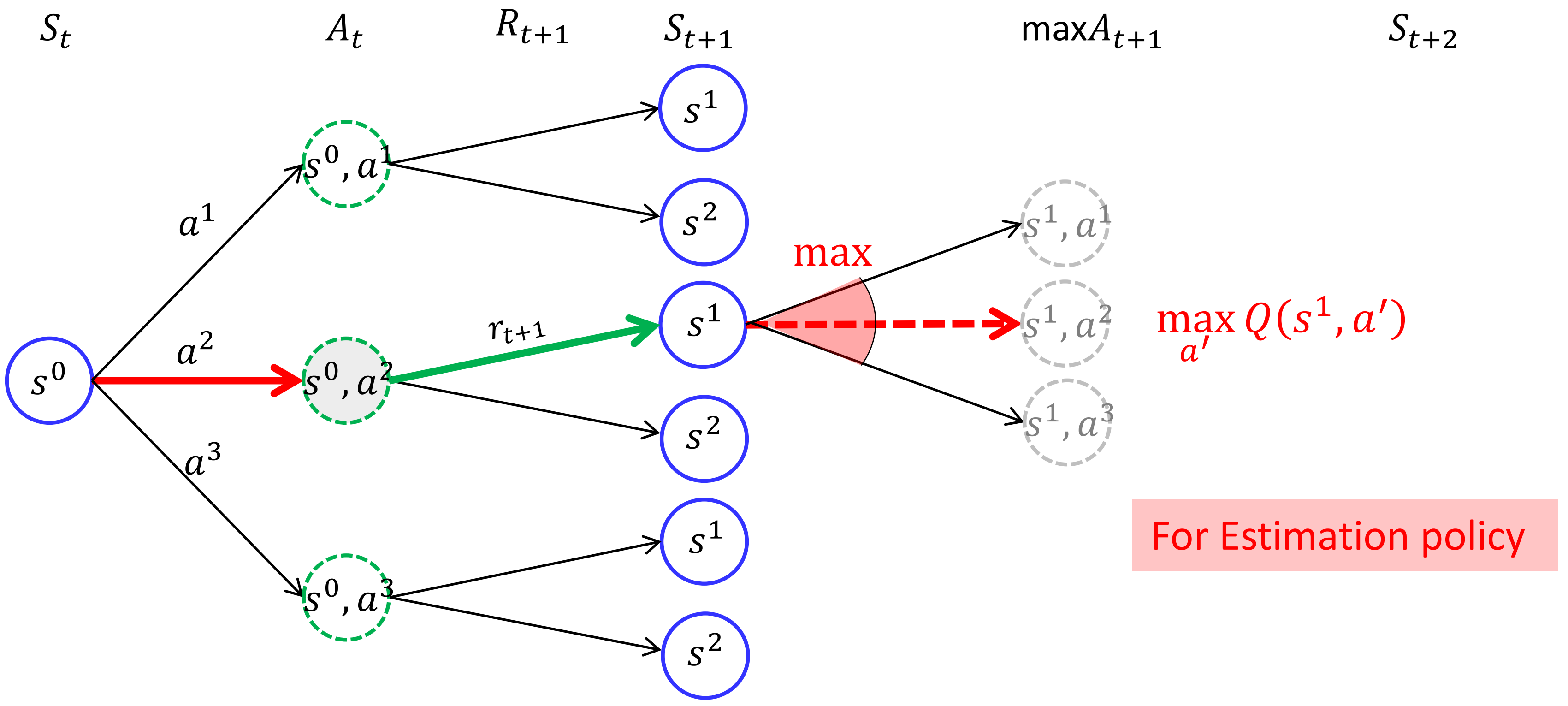
Assume a^2 is chosen

Q-Learning: Off-Policy TD Control



Take action $a_t = a^2$ given $s_t = s^0$ and observe r_{t+1} and $s_{t+1} = s^1$

Q-Learning: Off-Policy TD Control

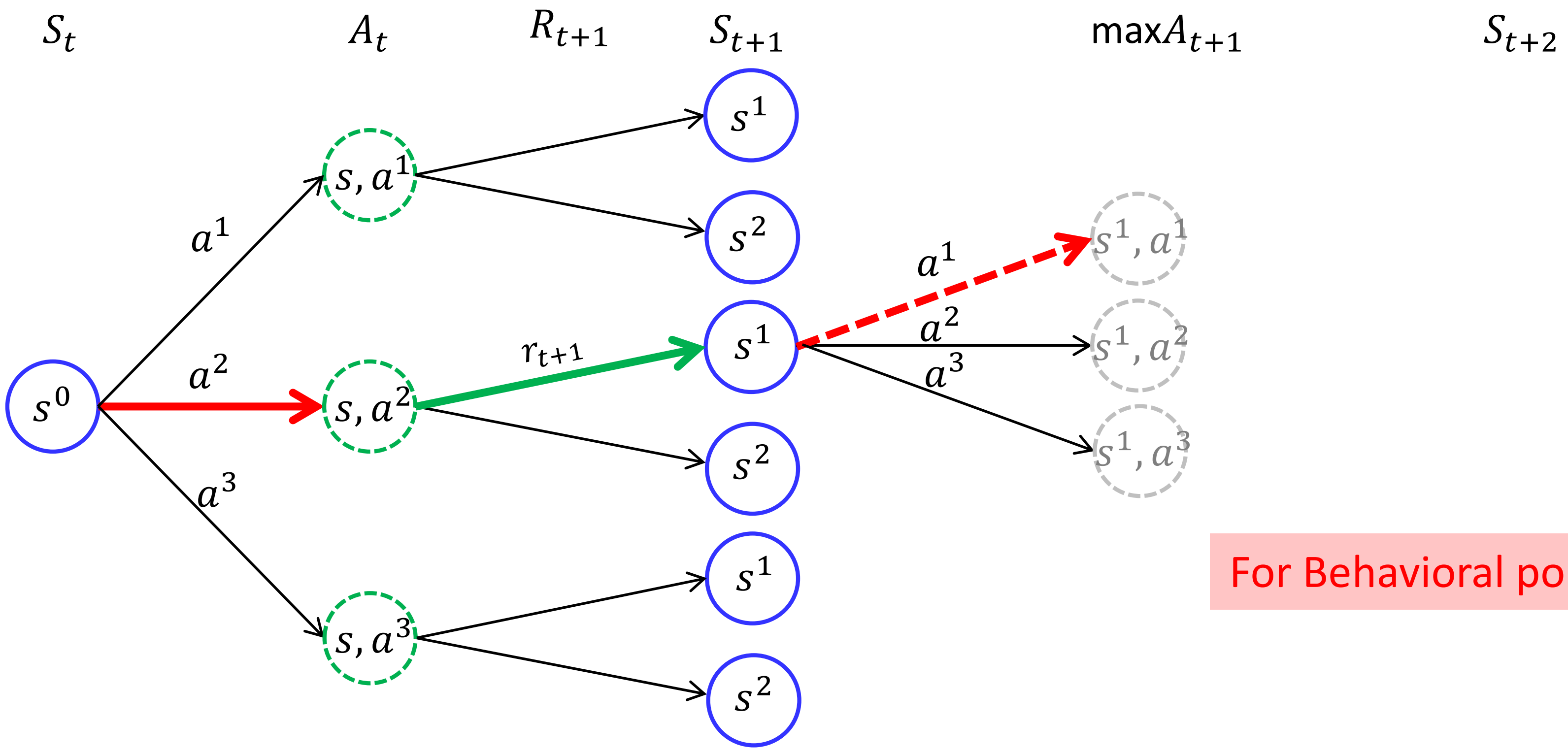


Update Q function with the $\max_{a'} Q(s^1, a')$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s, a') - Q(s_t, a_t) \right]$$

$$\rightarrow Q(s^0, a^2) \leftarrow Q(s^0, a^2) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s^1, a') - Q(s^0, a^2) \right]$$

Q-Learning: Off-Policy TD Control



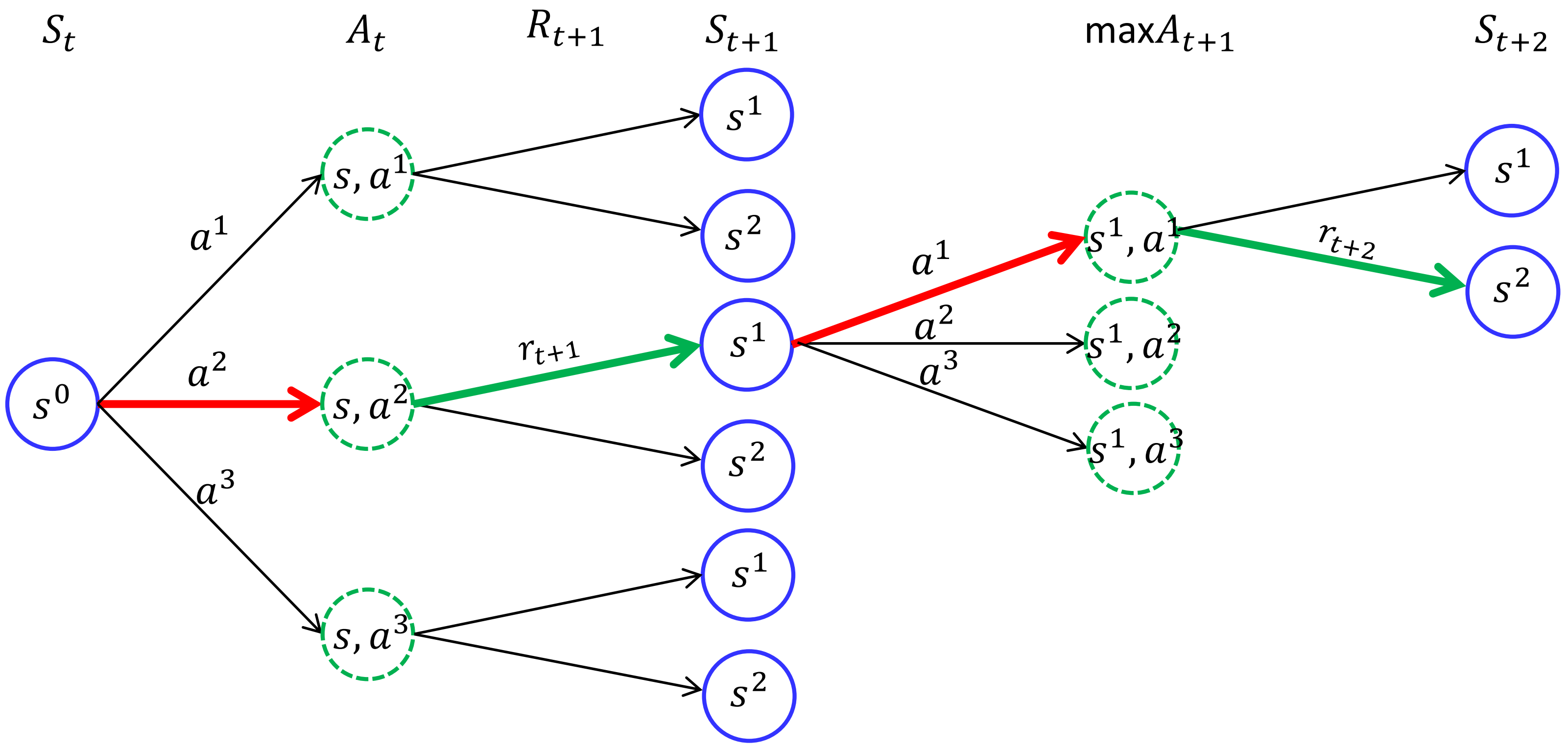
For Behavioral policy

Choose a_{t+1} from $s_{t+1} = s^1$ using current Q

$$a_{t+1} = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(s_{t+1} = s^1, a) & \text{with prob } 1 - \epsilon \\ \text{random action} & \text{with prob } \epsilon \end{cases}$$

Assume a^1 is chosen

Q-Learning: Off-Policy TD Control



Take action $a_{t+1} = a^1$ given $s_{t+1} = s^1$ and observe r_{t+2} and $s_{t+2} = s^2$