

# DSA5105 Principles of Machine Learning

AY2024/25 Sem1 By Zhao Peiduo

## Supervised Learning

### Lecture 1

**Empirical Risk Minimization (ERM)** The learning process aims to find a function  $f \in \mathcal{H}$  that minimizes the empirical risk,  $R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i))$ , where  $y_i = f^*(x_i)$ .

**Common Loss Function** Mean squared error (MSE) for regression tasks:  $L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$ , cross-entropy loss for classification task:  $L(y, p) = -\sum_i y_i \log(p_i)$ , and huber loss for robust regression:  $L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$

**Softmax Function** For a multi-class classification problem with  $K$  classes, the softmax function is defined as:  $\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)}$

**General Ordinary Least Squares Formula** Consider  $x \in \mathbb{R}^d$  and the new hypothesis space  $\mathcal{H}_M = \{f : f(x) = \sum_{j=0}^{M-1} w_j \varphi_j(x)\}$  Each  $\varphi_j : \mathbb{R}^d \rightarrow \mathbb{R}$  is called a **basis function** or **feature map**.

We can rewrite the ERM  $\min_{w \in \mathbb{R}^M} \frac{1}{2N} \sum_{i=1}^N \left( \sum_{j=0}^{M-1} w_j \varphi_j(x_i) - y_i \right)^2$  into  $\min_{w \in \mathbb{R}^M} \frac{1}{2N} \|\Phi w - y\|^2$ . Solving by setting  $\nabla R_{\text{emp}}(\hat{w}) = 0$ , we have  $\hat{w} = (\Phi^\top \Phi)^{-1} \Phi^\top y$ , given invertible  $\Phi^\top \Phi$ .

For cases where  $\Phi^\top \Phi$  is not invertible, the formula using the Moore-Penrose pseudoinverse is:  $\hat{w}(u) = \Phi^\dagger y + (I - \Phi^\dagger \Phi)u$   $u \in \mathbb{R}^M$

**Regularization:** To prevent overfitting, regularization techniques add a penalty to the loss function:  $\min_{w \in \mathbb{R}^M} \frac{1}{2N} \|\Phi w - y\|^2 + \lambda C(w)$   $\frac{1}{2N} \|\Phi w - y\|^2 + \lambda C(w)$ . Minimizing the ERM we get  $\hat{w} = (\Phi^\top \Phi + \lambda I_M)^{-1} \Phi^\top y$  which is always invertible for positive  $\lambda$ .

**Common regularization terms:** **L<sub>2</sub> (Ridge) regularization:**  $\lambda \sum_{j=1}^p w_j^2$ , **L<sub>1</sub> (Lasso) regularization:**  $\lambda \sum_{j=1}^p |w_j|$

### Lecture 2

**Reformulation of Ridge Regression** We rewrite the regularized least squares solution in another way:

$$\hat{w} = (\Phi^\top \Phi + \lambda I_M)^{-1} \Phi^\top y = \Phi^\top (\Phi \Phi^\top + \lambda I_N)^{-1} y$$

**Reformulation of Ridge Regression**

$\hat{f}(x) = \sum_{i=1}^N \alpha_i \varphi(x_i)^\top \varphi(x)$   $\alpha = (G + \lambda I_N)^{-1} y$  where  $G_{ij} = \varphi(x_i)^\top \varphi(x_j)$  is the gram matrix

**Mercer's Theorem and SPD Kernels** Suppose  $k$  is a SPD kernel. Then, there exists a feature space  $\mathcal{H}$  and a feature map  $\varphi : \mathbb{R}^d \rightarrow \mathcal{H}$  such that  $k(x, x') = \varphi(x)^\top \varphi(x')$

**SPD kernels properties:**  $K(x, x') = K(x', x)$  (**Symmetry**) For any  $n$  and  $\{x_1, \dots, x_n\}$ , the Gram matrix  $G_{ij} = k(x_i, x_j)$  is positive semi-definite. (Recall: a matrix  $G$  is positive semi-definite if  $c^\top G c \geq 0$  for any vector  $c$ ) (**Positive Semi-definiteness**)

**Examples of SPD Kernels:** **Linear Kernel:**  $K(x, x') = x^\top x'$  **Polynomial Kernel:**  $K(x, x') = (1 + x^\top x')^d$

**Gaussian RBF Kernel:**  $K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$

**Constructing kernels** Given valid kernels  $k_1(x, x')$  and  $k_2(x, x')$ , the following new kernels will also be valid:  
 $ck_1(x, x')$   $f(x)k_1(x, x')f(x')$   $q(k_1(x, x'))$   $\exp(k_1(x, x'))$   $k_1(x, x') + k_2(x, x')$   $k_1(x, x')k_2(x, x')$   
 $k_3(\varphi(x), \varphi(x'))$   $x^\top A x'$   $k_a(x_a, x'_a) + k_b(x_b, x'_b)$   $k_a(x_a, x'_a)k_b(x_b, x'_b)$

### Lecture 3

**SVM Max Margin Formulation**  $\max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \min_{i=1, \dots, N} |\mathbf{w}^\top \mathbf{x}_i + b|$  subject to  $y_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0 \quad \forall i$

**Optimization Problem**  $\min_{w, b} \frac{1}{2} \|w\|^2$  subject to:  $y_i(w^\top x_i + b) \geq 1 \quad \forall i$ . Introducing Lagrange multipliers  $\alpha_i \geq 0$ , the Lagrangian is:  $\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w^\top x_i + b) - 1]$

**KKT conditions for SVM**

- From Stationarity  $\hat{w} = \sum_{i=1}^N \hat{\mu}_i y_i x_i, \quad 0 = \sum_{i=1}^N \hat{\mu}_i y_i$
- From Dual Feasibility  $\hat{\mu}_i \geq 0$  for  $i = 1, \dots, N$

- From Complementary Slackness  $\hat{\mu}_i = 0$  or  $y_i(\hat{w}^\top x_i + \hat{b}) = 1$

- The multipliers  $\hat{\mu}$  can be found by the dual problem  $\max_{\mu \in \mathbb{R}^N} \sum_{i=1}^N \mu_i - \frac{1}{2} \sum_{i,j} \mu_i \mu_j y_i y_j (x_i^\top x_j)$

**Dual formulation of SVM**

$\hat{\mu} = \arg \max_{\mu \in \mathbb{R}^N} \sum_{i=1}^N \mu_i - \frac{1}{2} \sum_{i,j} \mu_i \mu_j y_i y_j (x_i^\top x_j)$  Subject to:  $\hat{\mu} \geq 0$  and  $\sum_{i=1}^N \hat{\mu}_i y_i = 0$

Decision function:  $\hat{f}(x) = \text{sgn}\left(\sum_{i=1}^N \hat{\mu}_i y_i x_i^\top x + \hat{b}\right)$  Complementary slackness:  $\hat{\mu}_i = 0$  or  $1 = y_i(\hat{w}^\top x_i + \hat{b})$

**Kernel SVMs**

$\hat{\mu} = \arg \max_{\mu \in \mathbb{R}^N} \sum_{i=1}^N \mu_i - \frac{1}{2} \sum_{i,j=1}^N \mu_i \mu_j y_i y_j k(x_i, x_j)$  Subject to:  $\hat{\mu} \geq 0$  and  $\sum_{i=1}^N \hat{\mu}_i y_i = 0$

Decision function:  $\hat{f}(x) = \text{sgn}\left(\sum_{i=1}^N \hat{\mu}_i y_i k(x_i, x) + \hat{b}\right)$

**Only** support vectors satisfying  $1 = y_i(\hat{w}^\top \varphi_i(x) + \hat{b})$  matter for predictions. This is a **sparse kernel method**.

### Lecture 4

**Classification and Regression Trees** Suppose that the input space is  $\mathcal{X}$ . A **partition** of  $\mathcal{X}$  is a collection of subsets  $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_J$  such that  $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$  for  $i \neq j$  and  $\bigcup_{j=1}^J \mathcal{R}_j = \mathcal{X}$

The general decision tree hypothesis space is:  $\mathcal{H} = \left\{f : f(x) = \sum_{j=1}^J a_j \mathbb{I}_{x \in \mathcal{R}_j}, \{\mathcal{R}_j\} \text{ is a partition of } \mathcal{X}, a_j \in \mathcal{Y}\right\}$  where  $\mathbb{I}_{x \in \mathcal{R}_j}$  is an indicator variable returning 1 if  $x$  is in  $\mathcal{R}_j$ .

A decision tree model  $f(x) = \sum_{j=1}^J a_j \mathbb{I}_{x \in \mathcal{R}_j}$  depends on both  $a_j$  and  $\mathcal{R}_j$ . For regression we take the **average** label

values  $a_j = y_j = \frac{\sum_i y_i \mathbb{I}_{x \in \mathcal{R}_j}}{\sum_i \mathbb{I}_{x \in \mathcal{R}_j}}$ ; For classification we take the **mode** label values  $a_j = \text{mode}\{y_i : x_i \in \mathcal{R}_j\}$

**Loss function for Decision Trees Classification** Entropy:  $-\sum_{k=1}^K \sum_{j=1}^J p_{jk} \log p_{jk}$

Gini Impurity:  $\sum_{k=1}^K \sum_{j=1}^J p_{jk} (1 - p_{jk})$  where  $p_{jk}$  is the proportion of samples in  $\mathcal{R}_j$  belonging to class  $k$ .

**Bagging** reduces the variance by aggregating predictions from multiple models trained on different random subsamples of the data: **Regression**  $\bar{f}(x) = \frac{1}{m} \sum_{j=1}^m f_j(x)$  **Classification**  $\bar{f}(x) = \text{Mode}\{f_j(x) : j = 1, \dots, m\}$

**Boosting** works by training weak learners sequentially, where each learner focuses on correcting the mistakes of the previous ones. The combined model is a weighted sum of the weak learners:  $f(x) = \sum_{t=1}^T \alpha_t f_t(x)$  where  $\alpha_t$  are coefficients based on each learner's performance. Boosting helps reduce bias.

**Key Ideas of AdaBoost**

- Initialize with uniform weight across all training samples
- Train a classifier/regressor  $f_1$
- Identify the samples that  $f_1$  got wrong (classification) or has large errors (regression)
- Weight these samples more heavily and train  $f_2$  on this reweighted dataset
- Repeat steps 3-5

**AdaBoost Implementation**

**Data:**  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ , Initialize  $w_i^{(1)} = \frac{1}{N}$  for all  $i = 1, \dots, N$ ;  
For  $j = 1, \dots, m$  do

- Obtain  $f_j$  from:  $f_j = \arg \min_{f \in \mathcal{H}} \sum_{i=1}^N w_i^{(j)} \mathbb{I}_{y_i \neq f(x_i)}$
- Compute combination coefficients:  $\delta_j = \frac{\sum_{i=1}^N w_i^{(j)} \mathbb{I}_{y_i \neq f_j(x_i)}}{\sum_{i=1}^N w_i^{(j)}} \quad \alpha_j = \log\left(\frac{1 - \delta_j}{\delta_j}\right)$
- Update weights:  $w_i^{(j+1)} = w_i^{(j)} \exp\left(\alpha_j \mathbb{I}_{y_i \neq f_j(x_i)}\right)$

Return:  $\bar{f}(x) = \text{Sign}\left(\sum_{j=1}^m \alpha_j f_j(x)\right)$

**Cross-Validation** In  $k$ -fold cross-validation, the data is split into  $k$  subsets. The model is trained on  $k - 1$  subsets and validated on the remaining one. This process is repeated  $k$ -times.

### Lecture 5

**Activation Functions**

**Sigmoid:**  $\sigma(z) = \frac{1}{1 + e^{-z}}, [0, 1]$  **ReLU**  $\sigma(z) = \max(0, z), [0, \text{inf}]$ , leaky ReLU:  $\delta \text{ if } z < 0$  instead of 0

**Gradient Descent**  $w^{(t+1)} = w^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial w}$

**Stochastic Gradient Descent (SGD):**  $\theta_{k+1} = \theta_k - \eta \frac{1}{J} \sum_{i \in I_B} \nabla \Phi_i(\theta_k)$ , less likely to be locally optimal.

**Deep Neural Networks (DNNs)** Deep neural networks are an extension of shallow networks. The idea is to stack  $n$  hidden layers together and forward pass the  $x$  sequentially.

**Back-propagation Algorithm**

Initialize  $x_0 = x \in \mathbb{R}^d$ .  
For  $t = 0, 1, \dots, T$ :  $x_{t+1} = g_t(x_t, W_t) = \sigma(W_t x_t)$   
Set  $p_{T+1} = \nabla_x L(x_{T+1}, y)$ .  
For  $t = T, T - 1, \dots, 1$ :

$$-\nabla W_t \Phi = p_{t+1}^\top \nabla W g_t(x_t, W_t)$$

$$-p_t = [\nabla g_t(x_t, W_t)]^\top p_{t+1}$$

Return  $\{\nabla W_t \Phi : t = 0, \dots, T\}$ .

**Neural Network Architecture Example**

**The architecture of the network consists of:**  $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ ,  $x_1 \in \mathbb{R}$ ,  $x_2 \in \mathbb{R}$  **Input layer:**  $x_0 = x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$

**1st hidden layer with 2 neurons. 2nd hidden layer with 3 neurons. Output:** Scalar.

**1st Hidden Layer**  $x_1 = W_0 x_0 + b_0$   $W_0 \in \mathbb{R}^{2 \times 2}$ ,  $b_0 \in \mathbb{R}^{2 \times 1}$   $x_1 = \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix}$ ,  $x_1 \in \mathbb{R}^{2 \times 1}$   $x_1 = \text{ReLU}(x_1)$

**2nd Hidden Layer**  $x_2 = W_1 x_1 + b_1$   $W_1 \in \mathbb{R}^{3 \times 2}$ ,  $b_1 \in \mathbb{R}^{3 \times 1}$   $x_2 = \text{ReLU}(x_2)$ ,  $x_2 \in \mathbb{R}^{3 \times 1}$

**Back-propagation Example Computation**

The model is defined as:  $y(x) = v \delta(w_1 \delta(w_0 x))$ ,  $x, w_0, w_1, v \in \mathbb{R}$ , where  $\delta$  is the identity function, i.e.,  $\delta(z) = z$ . The loss function is:  $L = (y(x) - y)^2$  We want to compute the gradients:  $\frac{\partial L}{\partial v}$ ,  $\frac{\partial L}{\partial w_1}$ ,  $\frac{\partial L}{\partial w_0}$

**Forward pass:**  $x_1 = w_0 x$   $x_2 = v w_1 x_1$   $L = (x_2 - y)^2$

**Backward pass:**

$$p_2 = \frac{\partial L}{\partial x_2} = 2(x_2 - y) \quad \frac{\partial L}{\partial v} = \frac{\partial L}{\partial x_2} \cdot \frac{\partial x_2}{\partial v} = 2(x_2 - y) w_1 x_1 \quad \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_1} = 2(x_2 - y) v x_1$$

$$p_1 = \frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial x_2} \cdot \frac{\partial x_2}{\partial x_1} = 2(x_2 - y) v w_1 \quad \frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_0} = 2(x_2 - y) v w_1 x$$

### Lecture 6

**PCA Algorithm Simplified Flow** Center the data  $\rightarrow$  compute sample covariance matrix  $S = \frac{1}{N} \sum_{i=1}^N x_i x_i^\top \rightarrow$  compute top  $k$  eigenvalues and retrieve their corresponding eigenvectors. In feature space we replace all  $x$  with  $\phi$ , the feature maps, and compute design matrix as  $\Phi_{ij} \leftarrow \Phi_{ij} - \frac{1}{N} \sum_{i=1}^N \Phi_{ij}$ , where  $\Phi_{ij} = \phi_j(x_i)$  is the raw design matrix. **PCA Whitening Transform** Principal component scores are given by  $Z = XU$  where  $X$  is the original features and  $U$  is the matrix of eigenvectors. The transformation  $X' = XU\Lambda^{-\frac{1}{2}}$ , where  $\Lambda$  is the matrix of eigenvalues makes  $\text{cov}(X') = I$ .

**Neural Network AutoEncoders** Choose encoder  $T_{\text{enc}}$  and decoder  $T_{\text{dec}}$  as:

$$T_{\text{enc}}(x; \theta) = A\sigma(Wx + b) \quad T_{\text{dec}}(c; \phi) = B\sigma(Vc + c)$$

$$\theta = (A, W, b) \in \mathbb{R}^{m \times q} \times \mathbb{R}^{q \times d} \times \mathbb{R}^q \quad \phi = (B, V, c) \in \mathbb{R}^{d \times q'} \times \mathbb{R}^{q' \times m} \times \mathbb{R}^{q'}$$

Given a dataset  $\mathcal{D} = \{x_i\}_{i=1}^N$ , we solve the empirical risk minimization to minimize the distance between  $x_i$  and  $x'_i$ :

$$\min_{\theta, \phi} \frac{1}{2N} \sum_i \|x_i - T_{\text{dec}}(T_{\text{enc}}(x_i; \theta); \phi)\|^2, \text{ where the input is used as labels.}$$

## Unsupervised Learning

### Lecture 7

**Clustering** partition a dataset  $\mathcal{D}$  into disjoint groups  $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \dots \cup \mathcal{D}_K$ , such that data grouped together are similar and dissimilar if they are in different groups.

**K-means Algorithm** To minimize the distortion  $\min_Z J(R, Z) = \frac{1}{2N} \sum_{i=1}^N \sum_{k=1}^K r_{ik} \|x_i - z_k\|^2$ :

**Data:**  $\mathcal{D} = \{x_i\}_{i=1}^N, x_i \in \mathbb{R}^d$  for all  $i$

**Hyperparameters:**  $K$  (number of clusters); stopping criterion (convergence of loss function  $j$  for instance)

Initialize  $Z \in \mathbb{R}^{K \times d}$ , that is, starting with some centroid set (which could determine algo performance)

While stopping criterion not reached do

Assign point to the nearest centroid:  $r_{ik} = \begin{cases} 1 & \text{if } k = \arg \min_j \|x_i - z_j\|^2 \\ 0 & \text{otherwise} \end{cases}, i = 1, \dots, N$

Recompute centroid by taking average of members of cluster:  $z_k = \frac{\sum_{i=1}^N r_{ik} x_i}{\sum_{i=1}^N r_{ik}}, k = 1, \dots, K$

**Return:** centroids  $Z$ , assignment matrix  $R$

Depending on the initial condition, we can end up with global/local optimum or saddle point.

**Maximum Likelihood Estimation** Given dataset  $x_i$ , model them as i.i.d. samples from  $p(x|\theta)$ ,  $\theta$  are parameters to be determined:  $\theta_{MLE} = \arg \max_{\theta} \log(\prod_i p(x_i|\phi)) = \sum_i \log(p(x_i|\phi))$

**Gaussian Mixture Models (GMMs)** Soft probabilistic labeling as compared to K-means (hard deterministic labeling): Model the data as samples from a linear convex combination of  $K$  Gaussian:

$p(x) = \sum_{k=1}^K \pi_k p_g(x; z_k, \Sigma_k)$ , where  $p_g(x_i|z, \Sigma) = (2\pi)^{-\frac{d}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-z)^T \Sigma^{-1}(x-z)\right\}$ ,

$\pi_k \geq 0, \sum_k \pi_k = 1$  (Mixture weights)

**Sampling from GMM**

- Sample  $r$  (one-hot) according to  $\pi_k$  - find a member Gaussian distribution
- Identify the "hot" coordinate of  $r$  - Go to that distribution
- Sample  $x$  from  $p_g(x; z_\ell, \Sigma_\ell)$  - sampling by its mean and covariance matrix

**MLE for GMM**

**Data:**  $\mathcal{D} = \{x_i\}_{i=1}^N, x_i \in \mathbb{R}^d$  for all  $i$  **Hyperparameters:**  $K$  (num.clusters); stopping criterion

**Initialize:**  $\pi_k = 1/K, z_k \in \mathbb{R}^d, \Sigma_k \in \mathbb{R}^{d \times d}$  for  $k = 1, \dots, K$

While stopping criterion not reached do

Update  $\gamma_{ik}$ , the responsibility term:  $r_{ik} = \frac{\pi_k p_g(x_i; z_k, \Sigma_k)}{\sum_{\ell=1}^K \pi_\ell p_g(x_i; z_\ell, \Sigma_\ell)}, i = 1, \dots, N, k = 1, \dots, K$

Compute  $\{N_k = \sum_{i=1}^N \gamma_{ik}\}$  and Update  $\{\pi_k, z_k, \Sigma_k\}$ :

$z_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} x_i, \Sigma_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} (x_i - z_k)(x_i - z_k)^T, \pi_k = \frac{N_k}{N}$  for  $k = 1, \dots, K$

**Return:** cluster centers, covariances, and mixture coefficients  $\{z_k, \Sigma_k, \pi_k\}$ , cluster responsibilities or soft assignments  $\{\gamma_{ik}\}$

**Relationship between K-means and GMM** Set  $\Sigma_k = \epsilon I_d$ , then the responsibility term becomes  $\gamma_{ik} = \frac{\pi_k \exp\left(-\frac{1}{2\epsilon} \|x_i - z_k\|^2\right)}{\sum_{\ell} \pi_\ell \exp\left(-\frac{1}{2\epsilon} \|x_i - z_\ell\|^2\right)}$ , and when we take  $\epsilon \rightarrow 0$  we will get the hard label assignment as K-means.

## Reinforcement Learning

### Lecture 8

**Key Elements of RL** Action (action space, policy/chain of actions leading to optimal reward), State (match with action to form transition function, terminal / goal state), Reward (reward function)

**Markov Property** The future does not depend on the past, i.e. for transition probability  $\mathbb{P}(S_{t+1} = s' | S_t = s, S_{t-1} = s_{t-1}, \dots, S_0 = s_0)$ , it is simply  $\mathbb{P}(S_{t+1} = s' | S_t = s)$ .

**Transition Matrix** In time homogeneous Markov Chain, The transition probability is independent of time:  $\mathbb{P}(S_{t+1} = s' | S_t = s) = P_{ss'}$

**Markov Decision Process (MDP)**  $p(s', r|s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ , which integrate reward  $R$  and action  $a$  to the transition probability. It is finite if the action space is discrete finite for each state.

**Policy**  $\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$ , and deterministic policy is a constant mapping from the state space to action space  $\pi: S \rightarrow A$

**(Discounted) Return**  $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \gamma \in (0, 1]$ , where  $\gamma$  is the discount factor. The goal is typically to maximize the expected return, and in the context of time homogeneous case,  $\mathbb{E}_\pi [G_t | S_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]$

**Dynamic Programming** Memoization / caching the result of overlapping optimal substructures. Trade space for cost of recomputation for naive implementations and space usage can be optimized through iterative approach.

**Bellman's Equation**  $v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma v_\pi(s')]$

This can be derived using DP to formulate a recursive solver for value function  $v_\pi(s) = \mathbb{E}_\pi [G_t | S_t = s]$  and action value function  $q_\pi(s, a) = \mathbb{E}_\pi [G_t | S_t = s, A_t = a]$ , and substitute the action value function to the value function.

**Bellman's Equation for Finite MDP**  $v_\pi = \gamma P(\pi) v_\pi + b(\pi)$ , where:

- $P(\pi)_{s,s'} = \sum_a \pi(a|s) \sum_r p(s', r|s, a)$  is the state transition matrix for the policy  $\pi$  ( $N * N$  matrix with  $N$  being number of states, so we only sum over  $r$ )
- $b(\pi)(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \cdot r$  is the expected immediate reward for the policy  $\pi$  ( $N * 1$  matrix)

The unique solution would thus be  $v_\pi = (I - \gamma P(\pi))^{-1} b(\pi)$ .

**Optimal Policy**  $\pi_* \geq \pi \quad \forall \pi \neq \pi_*$  Always exist but may not be unique.

**Policy Improvement** For any two policies  $\pi, \pi'$ , if  $\sum_a \pi'(a|s) q_\pi(s, a) \geq \sum_a \pi(a|s) q_\pi(s, a) \forall s$  then  $v_{\pi'}(s) \geq v_\pi(s) \forall s$ , and we assert  $\pi'$  is better than  $\pi$ .

**Bellman's Optimality Equations** Given an optimal policy  $\pi_*(s) \in \arg \max_a q_*(s, a)$ , we can recursively find optimal value and action value functions:

$v_*(s) = \max_a \sum_{s', r} p(s', r|s, a) [r + \gamma v_*(s')]$   $q_*(s, a) = \sum_{s', r} p(s', r|s, a) [r + \gamma \max_{a'} \sum_{s', r} p(s', r|s, a) q_*(s', a')]$

$v_*(s) = \max_a q_*(s, a)$  **Optimal Policy:**  $\pi_*(s) \in \arg \max_a q_*(s, a)$

### Lecture 9

**Model-Based vs Model-Free** Model-based assumes that the transition function is known: Value iteration, policy iteration. Model-free improves through sampling/ Monte Carlo simulations: Temporal Difference (TD), Q-learning.

**Value Iteration Algorithm** Start with some value function, update utility estimate iteratively and keep track of corresponding current optimal actions

## Reinforcement Learning

**Model Parameters:** MDP transition probability  $p(s', r|s, a)$ , discount rate  $\gamma < 1$

**Input:** Policy  $\pi$ , Stopping criterion

Initialize  $v \in \mathbb{R}^n$

While stopping criterion not reached do

Update value function:  $v \leftarrow F(v) = \max_\pi \{\gamma P(\pi) v + b(\pi)\}$  (Make use of P, model-based)

**Return**  $v$

**Contraction Mapping Theorem** Fix point iteration will lead to a unique  $v_* \in V$  such that  $v_* = F(v_*)$ , and  $v_*$  can be estimated by starting at arbitrary  $v_0$  and iterate for many times (towards infinity).

**Value Iteration Convergence Rate** Given error  $\epsilon$  being the difference between  $v_K$  and  $v_\infty$ , and number of iterations  $k$ , to achieve  $\mathcal{O}(\epsilon)$  we need  $k \sim \mathcal{O}(\log(1/\epsilon))$

**Policy Iteration Algorithm** Start with some policy, plug into value function and derive a better policy. Repeat until policy does not change.

**Model Parameters:** MDP transition probability  $p(s', r|s, a)$ , discount rate  $\gamma < 1$

**Initialize:** Deterministic policy  $\pi \leftarrow \pi_0$

While  $\pi \neq \pi'$  do

$\pi' \leftarrow \pi$  (Set the next policy for evaluation)

$v_\pi \leftarrow (I - \gamma P(\pi))^{-1} b(\pi)$  (Evaluate policy  $\pi$  using P, model-based)

$\pi' \leftarrow \arg \max_\pi \{b(\pi) + \gamma P(\pi) v_\pi\}$  (Retrieve subsequent best policy)

**Return**  $v$

Its convergence can be shown by  $v_{\pi_k} \leq v_{\pi_{k+1}} \leq v_*$  from the algorithm (argmax, minimally equivalent values) and  $\pi_{k+1} \neq \pi_k$  if and only if  $v_{\pi_{k+1}} > v_{\pi_k}$  (otherwise the while loop ends)

**Monte Carlo Method** IID sampling and compute expectation:  $\mathbb{E}_{x \sim \mu} f(x) \approx \frac{1}{N} \sum_{i=1}^N f(X_i)$  where  $X_i \sim \mu$  is IID

**Model-free Policy Iteration** Instead of relying on the transition matrix P we sample episodes according current  $\pi$  to get samples of state and reward sequence (S and R), and get the value by averaging discounted rewards. Concretely, update estimation for action-value function by:  $q(s, a) \leftarrow \frac{1}{N} \sum_{n=1}^N \left[ \sum_{k=0}^N \gamma^k R_{k+1}^{(n)} \middle| S_0^{(n)} = s, A_0^{(n)} = a \right]$  and

perform policy improvement based on this  $q(s, a)$ .

**Temporal Difference (TD) Algorithm**

**Input:** State-reward simulator, initial state sampler, Stopping criterion, Policy  $\pi$

**Initialize:** Initial value  $v_0$ , Episode length  $T$ , Start with  $v \leftarrow v_0$ ;

While stopping criterion not reached do

$s \leftarrow \text{InitialStateSampler}()$  (The start of the new episode)

For  $t = 0$  to  $T - 1$  do

$a \leftarrow a \sim \pi(\cdot|s)$

$s', r \leftarrow \text{StateRewardSimulator}(s, a)$

$v(s) \leftarrow (1 - \alpha)v(s) + \alpha (r + \gamma v(s'))$

**Return**  $v$

**Q-learning Algorithm**

**Input:** State-reward simulator, initial state sampler, Stopping criterion, Policy  $\pi$

**Initialize:** Initial value  $q_0$ , Episode length  $T$ , Start with  $q \leftarrow q_0$

While stopping criterion not reached do

$s \leftarrow \text{InitialStateSampler}()$

For  $t = 0$  to  $T - 1$  do

$a \leftarrow a \sim \pi(\cdot|s)$

$s', r \leftarrow \text{StateRewardSimulator}(s, a)$

$q(s, a) \leftarrow (1 - \alpha)q(s, a) + \alpha [r + \gamma \max_{a'} q(s', a')]$

**Return**  $q$

## Graph-Based Methods

### Lecture 10

**Page Rank** Let  $L_k \subset \{1, 2, \dots, n\}$  denote the set of pages with a link to page  $k$ ; that is,  $L_k$  is the set of page  $k$ 's backlinks. For each  $k$  we require  $x_k = \sum_{j \in L_k} \frac{x_j}{n_j}$  where  $n_j$  is the number of outgoing links from page  $j$ .

The resultant matrix A representing pairwise values of  $x_k$  is a column-stochastic matrix with an eigenvalue equal to

1, and 1 is also its largest eigenvalue. For instance:  $A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$

**Power Iteration** Let the eigenvalues of A be  $\lambda_1 > \lambda_2 > \dots > \lambda_n$  and the associated eigenvectors be  $v_1, v_2, \dots, v_n$ . Then any vector  $x_0$  can be written as  $x_0 = c_1 v_1 + c_2 v_2 + \dots + c_n v_n$ . Then

$A^k x_0 = c_1 \lambda_1^k v_1 + c_2 \lambda_2^k v_2 + \dots + c_n \lambda_n^k v_n = c_1 \lambda_1^k \left( v_1 + \frac{c_2}{c_1} \left( \frac{\lambda_2}{\lambda_1} \right)^k v_2 + \dots + \frac{c_n}{c_1} \left( \frac{\lambda_n}{\lambda_1} \right)^k v_n \right) \rightarrow c_1 \lambda_1^k v_1$  as  $k \rightarrow \infty$ .

### Lecture 11

**Normalized Laplacian**  $L_G := D - W = \sum_{i < j} w_{ij} (e_i - e_j)(e_i - e_j)^T$ , the diagl onl entries are the degree of each node and other entries are  $-w_{ij}$

**Spectral Clustering**

Given a graph  $G = (V, E, W)$ , let  $v_2$  be the eigenvector corresponding to the second smallest eigenvalue of the normalized Laplacian  $\mathcal{L}_G$ .

Let  $\phi_2 = D^{-\frac{1}{2}} v_2 \in \mathbb{R}^n$ , where D is the degree matrix with  $D_{ii} = \deg(i) = \sum_{j=1}^n w_{ij}$

Given a threshold  $\tau$  (one can try all different possibilities, or run  $k$ -means for  $k = 2$ ), set:  $S = \{i \in V : \phi_2(i) \leq \tau\}$

**Spectral Clustering for multi-clusters**

Given a graph  $G = (V, E, W)$ , let  $v_2, \dots, v_k$  be the eigenvectors corresponding to the second through  $(k-1)$ th eigenvalues of the normalized Laplacian  $\mathcal{L}_G = D^{-\frac{1}{2}} L_G D^{-\frac{1}{2}}$ .

Let  $\phi_m = D^{-\frac{1}{2}} v_m \in \mathbb{R}^n$ .

Consider the map  $\phi: V \rightarrow \mathbb{R}^{k-1}$  defined as  $\phi(v_i) = [\phi_2(i) \quad \dots \quad \phi_k(i)]^T$

Cluster the  $n$  points in  $k-1$  dimensions into  $k$  clusters using  $k$ -means.