COMP5328 Course Notes

Jazlyn Lin

July, 2021

Contents

1 Intro to ML Problems

1.1 5 Key Elements of Algorithms

1. **Input training data**
2. **Pre-defined hypothesis class**
3. **Objective function**
4. **Optimisation method**
   - both objective function and optimisation method form a mapping $\mathcal{A}$ that maps samples S to the output hypothesis h_S
5. **Output hypothesis**
   - h_S which is dependent on samples S

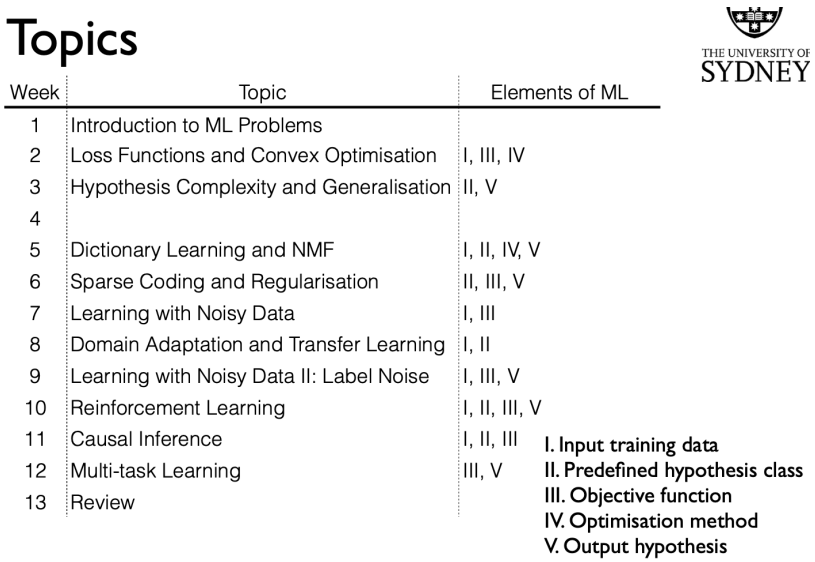


Figure 1: Relationship between weekly topic and the five elements

1.2 Input training data

- $S = \{(x_1, y_1), ..., (x_n, y_n)\}$
- can suffer from either feature noise (from X) or label noise (from Y).
  - feature noise: use robust algorithms to outliers such as Truncated Cauchy NMF
  - label noise: use robust algorithms to label noise

1.3 Pre-defined hypothesis class

- $\mathcal{H} = \{h_1, ...\}$ is a class of functions, can be infinite or finite
- find $\mathcal{H}$ through deep learning
  - each feature map $\phi_i(.)$ is a feature extractor. $\phi_L(.)$ is the last non-prediction layer. Since it passes through all L layers of feature extraction, it learns the best representation of feature X which is $\phi_L(X)$
  - $h(.)$ is the last layer (aka fully-connected prediction layer) of the neural network. This layer outputs prediction $h(\phi_L(X))$, just like the colorful viz.
  - if we rewrite prediction as $h(\phi_L(X))$ a new function $h'(X) = h(\phi_L(X))$ in terms of input feature X, then h' is basically a hypothesis. Essentially, the whole NN is a complex function/a hypothesis h' that leads to desirable prediction.

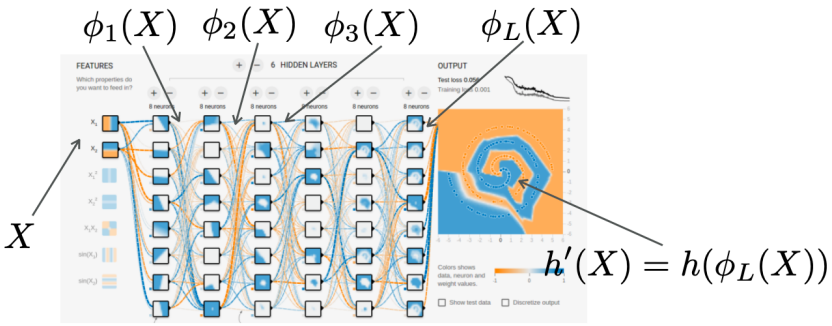


Figure 2: Caption

1.4 Objective function

- know what's the best classifier so we can work towards it
- **best classifier (accuracy)**: the classifier that has minimum **classification error** on *all* possible data of the task
  - for each data point (x_i, y_i), the classification error of h is measured by 0-1 loss function $1_{\{y_i \neq sgn(h(x_i))\}}$, which either 0 or 1.
  - the best classifier is then the classifier that minimises the classification error over all possible data points of the task such as

    $$\text{argmin}_h \mathbb{E}[1_{\{Y \neq sgn(h(X))\}}]$$

  - Best classifier is usually unobtainable as
    1. unknown distribution D -> unknown P -> can't calculate expected risk
    2. the objective function $\text{argmin}_h \mathbb{E}[1_{\{Y \neq sgn(h(X))\}}]$ contains 0-1 loss function, which makes it non-convex and non-smooth -> hard to optimise
    3. the predefined hypothesis class we pick may not necessarily contain the best classifier, which makes it unreachable.
- obtain best classifier through empirical 0-1 risk
  - Though unknown D, we have samples S drawn from D which we can utilise
  - under LLN, empirical 0-1 risk converges to the expected 0-1 risk such as

    $$\text{argmin}_h \frac{1}{n} \sum_{i=1}^{n} 1_{\{y_i \neq sgn(h(x_i))\}} \xrightarrow{n->\infty} \text{argmin}_h \mathbb{E}[1_{\{Y \neq sgn(h(X))\}}]$$

- empirical risk is an unbiased estimator

1.5 Optimisation Method

- once we have the right objective function (minimise empirical risk), we use optimisation method to *find* a hypothesis out of predefined hypothesis class $h \in \mathcal{H}$ that minimises the objective function
- note that usually we replace 0-1 loss with convex and smooth surrogate loss function that can be easily optimised.

  $$\text{argmin}_{h \in \mathcal{H}} f(h)$$

  where $f(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, h)$. Here h is what we want to optimise for as we want to find the h that minimise the empirical risk $f(h)$.
- use iterative updating method such as gradient descent to update h such that for each updated h (i.e. h_{k+1}), the objective function is minimised further $f(h_{k+1}) < f(h_k)$ when $\nabla f(h_k) \neq 0$

1.6 Output Hypothesis

- h_S
- generalisation error

2 Convex Optimisation and Loss Function

2.1 Convex Optimisation

Convex optimisation or convex analysis is crucial to the optimisation of our loss function.

- easy to obtain global minimum (as local minimum must be global minimum)
- easy to check for optimality through first-order and second-order condition

Definition 2.1. a minimyConvex Set

A set C is convex if for any two points in C, the line segment between them lies in C. Mathematically, for any $x, y \in C$ and for any $\theta \in [0, 1]$, if $\theta x + (1 - \theta)y \in C$ then C is convex.

Definition 2.2. Hyperplane and Halfspace

1. **Hyperplane**: A hyperplane of an n-dimensional space V is a subspace of dimension n-1. So a hyperplane in a 3D space should be 2D. Mathematically, $\{x|a^T x = b \text{ where } a \neq 0\}$.
2. **Half-Space** One of the two spaces a hyperplane divides an affine space into.
   (a) An open half-space doesn't include the hyperplane while the closed one include the hyperplane.
   (b) Property: A half space is a convex set.

Definition 2.3. Supporting Hyperplane and Supporting Hyperplane Theorem

1. **Supporting Hyperplane**: Given b_0 is the boundary point of set S, if for all $x \in S$, $\{a^T x \leq a^T b_0 \text{ where } a \neq 0\}$, then the hyperplane $\{x|a^T x = a^T b_0\}$ is called a supporting hyperplane to S at the point b_0. Essentially, a supporting hyperplane of set S is a hyperplane that satisfies two properties:
   (a) the set S is *fully* contained in one of the two closed half-spaces bounded by the hyperplane.
   (b) S has at least one boundary point on the hyperplane

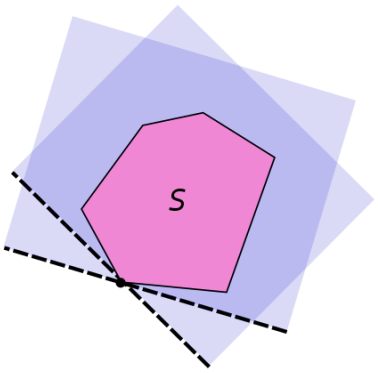


Figure 3: Convex set S can have multiple supporting hyperplanes on the same boundary point

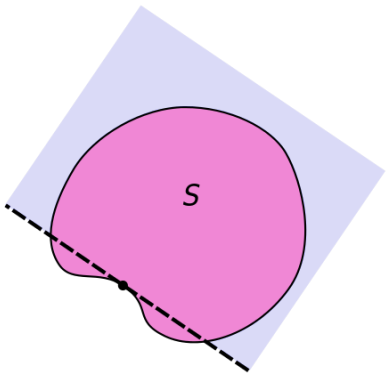


Figure 4: If property (b) is satisfied but not property (a). That is, S is not convex, then SH does not exist

2. **Supporting Hyperplane Theorem** If S is a *convex* set and b_0 is a boundary point of S, then there exists a supporting hyperplane containing b_0. (Shown as Figure 1). This theorem is proved from the separating hyperplane theorem.

Definition 2.4. Convex Functions

1. **Definition**
   (a) Mathematically, function f is *a convex function* if
      i. dom f is convex (i.e. an interval, as Figure 5)
      ii. for $\theta \in [0, 1]$ and all x, y $\in$ dom f, $f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$ (Jensen's Inequality)
   (b) Intuitively or geometrically, it means that the line segment between $(x, f(x))$ and $(y, f(y))$ lies above the function segment between $(x, f(x))$ and $(y, f(y))$. (Figure 6)

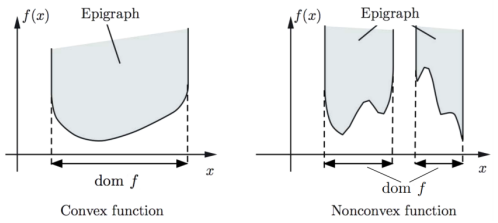


Figure 5: Convex and Non-convex Domain f

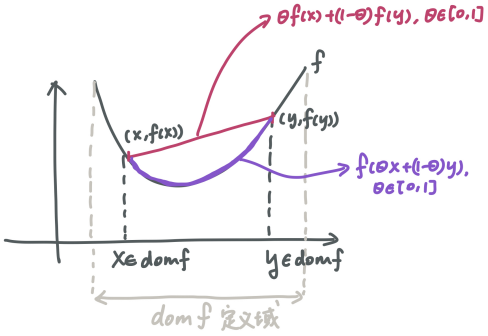


Figure 6: Convex Function Illustration

2. **Property**

   (a) f is concave if $-f$ is convex

   (b) f is strictly convex if there is a strict inequality ($<$) whenever $x \neq y$ and $\theta \in (0, 1)$

   (c) f is strictly concave if $-f$ is strictly convex

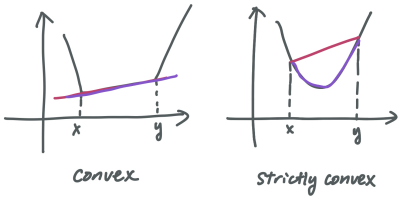


Figure 7: Convex and Strictly Convex

3. **Operations**

   (a) Non-negative scaling: convex f and $\alpha > 0 \Rightarrow$ convex αf

   (b) Addition: convex f_1 and $f_2 \Rightarrow$ convex $f_1 + f_2$

   (c) Non-negative weighted sum (combining above 2): convex f_1 and f_2, $\theta_1, \theta_2 \geq 0 \Rightarrow$ convex $\theta_1 f_1 + \theta_2 f_2$.

   (d) Function composition with affine mapping: convex f and $Ax + b \in$ dom $f \Rightarrow g(x) = f(Ax + b)$ is also convex

   (e) Point-wise maximum: convex $f_1, f_2 \Rightarrow$ convex $f(x) = max\{f_1(x), f_2(x)\}$

   Proof. Given convex function f_1 and f_2 and another point-wise maximum function $f(x) = max\{f_1(x), f_2(x)\}$, we have

$$\begin{aligned} f(\theta x + (1-\theta)y) &= max\{f_1(\theta x + (1-\theta)y), f_2(\theta x + (1-\theta)y)\} \\ &\leq max\{\theta f_1(x) + (1-\theta)f_1(y), \theta f_2(x) + (1-\theta)f_2(y)\} \\ &\leq \theta max\{f_1(x), f_2(x)\} + (1-\theta)max\{f_1(y), f_2(y)\} \\ &= \theta f(x) + (1-\theta)f(y) \end{aligned}$$

   □

4. **First-order condition**
   Suppose f is differentiable, hence ∇f exists for all $x \in$ dom f(open set since it's differentiable).
   Then f is a convex function $\Leftrightarrow$

   (a) dom f is convex

   (b) $f(y) \geq f(x) + \nabla f(x)^T (y - x)$ for all $x, y \in$ dom f

   Geometrically, this means that for any points in dom f, the graph lies above ($\geq$) its tagent line.

5. **Second-order condition**
   Suppose f is now twice-differentiable (Hessian Matrix H exists for each point in dom f).
   Then f is a convex function $\Leftrightarrow$

   (a) dom f is convex

   (b) Hessian matrix H is positive semidefinite for *all* points in dom f

Alternatively, f is strictly convex when Hessian is positive definite.

.

Remarks:

(a) Positive semidefinite geometrically means that f has a positive curvature over dom f (i.e. a bowl). In 1-d space, this condition is equivalent to $f'' \geq 0$

(b) Twice differentiable $f \Leftrightarrow \nabla^2 f$/Hessian matrix exists for all points in convex open set dom f

(c) Hessian matrix positive definite $\Leftrightarrow x^T Hx > 0$ or all eigenvalues > 0

(d) Hessian matrix positive semi-definite $\Leftrightarrow x^T Hx \geq 0$ or all eigenvalues ≥ 0

(e) f convex $\Leftrightarrow$ Hessian matrix is positive semi-definite (over all points in dom f)

(f) f strictly convex $\Leftrightarrow$ Hessian matrix is positive definite (over all points in dom f)

6. **Check if f is a convex function: 3 ways**

   (a) By definition - Jensen's equality
      i. dom f is convex
      ii. $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$ for $\theta \in [0,1]$ and all x, y $\in$ dom f

   (b) First-order condition
      i. (assume differentiable everywhere)
      ii. dom f is convex
      iii. $f(y) \geq f(x) + \nabla f(x)^T (y - x)$ for all $x, y \in$ dom f

   (c) Second-order condition
      i. (assume twice-differentiable everywhere)
      ii. dom f is convex
      iii. Hessian matrix is positive semi-definite for all $x \in$ dom f

7. **Optimality criterion for f: check if x is x^***
   point x is optimal(aka attains the global min) $\Leftrightarrow$

   (a) x is feasible

   (b) $f(y) \geq f(x) + \nabla f(x)^T (y - x)$ for all feasible $y \in$ dom f

Definition 2.5. Convex Optimisation

1. **Convex Optimisation Problem in Standard Form**

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0,\ i = 1, \ldots, m \\ & h_i(x) = 0,\ i = 1, \ldots, p \end{array}$$

   Remarks:

   - The objective function f_0 and the inequality constraint functions $f_1, ..., f_m$ must be convex
   - The equality constraint functions $h_i(x) = a_i^T x - b_i = 0$ must be affine (linear transformation + translation)
   - The domain/feasible set $\mathcal{D}$ of the convex optimisation problem is defined as $\mathcal{D} = f_0 \cap \bigcap_{i=1}^{m} f_i \cap \bigcap_{i=1}^{p} h_i$. $\mathcal{D}$ is convex as convex intersects convex is convex, and the remaining are just p hyperplanes.

2. **Optimality Condition for a feasible x and convex f**

   Suppose we have a convex optimisation problem as the problem above. Also suppose that $f_0(x)$ is differentiable and x is feasible. Then point x is optimal *iff* for all $y \in$ dom f we have $\nabla f_0(x)^T (y - x) \geq 0$. For unconstrained problem (there are no constraints to filter the complete curvature of convex function), this equation can simply reduce to $\nabla f_0(x) = 0$. That is, $\nabla f_0(x) = 0 \Leftrightarrow x$ is optimal.

   Remarks:

   (a) The first order condition $\nabla f_0(x) = 0$ only works when we also have second order condition - $\nabla f_0^2(x) \succ 0$ aka convex f, increase curvature or positive definite Hessian. Both conditions are sufficient for an optimal x to minimise f. Missing the second order will not be sufficient and a counter example would be the saddle point.

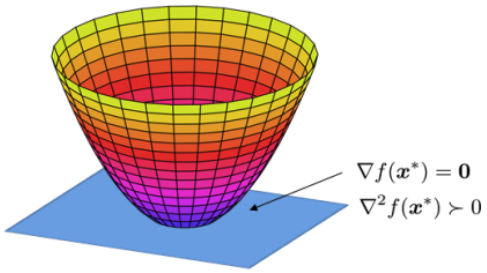


Figure 8: A Minimiser for a 3d Convex Function

Definition 2.6. Unconstrained Optimisation

$$\text{minimize} \quad f(x)$$

where f is convex and twice-differentiable.
Solving this equation has two ways

1. *Analytically* through $\nabla f(x) = 0$. For example, least square method.
2. *Iteratively* through descent methods such as gradient descent

Definition 2.7. Constrained Optimisation

Suppose we have a convex optimisation problem in standard form as follows

$$\begin{aligned} &\text{minimize} && f_0(x) \\ &\text{subject to} && f_i(x) \leq 0, \ i = 1, \ldots, m \\ &&& h_i(x) = 0, \ i = 1, \ldots, p \end{aligned}$$

where

- The objective function f_0 and the inequality constraint functions $f_1, ..., f_m$ are convex
- The equality constraint functions $h_i(x) = a_i^T x - b_i$ must be affine (linear transformation + translation)
- The domain/feasible set $\mathcal{D}$ of the convex optimisation problem is defined as $\mathcal{D} = f_0 \cap \bigcap_{i=1}^{m} f_i \cap \bigcap_{i=1}^{p} h_i$. $\mathcal{D}$ is convex as convex intersects convex is convex, and the remaining are just p hyperplanes.

Definition 2.8. Lagrangian and Lagrange Dual Function

Given a *general* constrained optimisation problem where f is NOT necessarily convex, the **Lagrangian** L for the optimisation problem can be written as

$$L(x, \lambda, v) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} v_j h_j(x) \tag{1}$$

where λ_i and v_j are the Lagrange multipliers/dual variables associated with contraints $f_i(x) \leq 0$ and $h_i(x) = 0$ respectively.

The **Lagrangian dual function/dual function** is then defined as the infimum (similar to minimum but can be $-\infty$) of this Lagrangian L over primal variables x

$$g(\lambda, v) = \inf_{x \in D} L(x, \lambda, v) = \inf_{x \in D} (f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} v_j h_j(x)) \tag{2}$$

where $\mathcal{D} = f_0 \cap \bigcap_{i=1}^{m} f_i \cap \bigcap_{i=1}^{p} h_i$. This dual function is *concave* because it is the pointwise infimum of affine functions.

Definition 2.9. Primal and Dual

Note that if we take the supremum over the Lagrangian L it gives back our original objective of the optimisation problem we want to solve. This is because f_i and h_i are non-positive while the Lagrange multipliers are non-negative so

the weighted constraint terms are always non-positive if all constraints are satisfied. Naturally, in order to maximize L we have to force the weighted constraints to be 0, which brings us back the objective function $f_0(x)$. If any constraints are not satisfied here, say $f_1(x) \geq 0$, we will try to maximize L by taking advantage of this opportunity and choose a λ_1 such that $\lambda_1 f_1(x) = \infty$. And this would lead to $L = \infty$.

$$p(x) = \sup_{\lambda,v:\lambda_i,v_i\geq 0} L(x,\lambda,v) = \begin{cases} f_0(x), & \text{if all constraints are satisfied} \\ \infty, & \text{otherwise} \end{cases} \tag{3}$$

Then we take the infimum over p as follows.

$$p^* = \inf_x p(x) = f_0(x^*) = \inf_x \sup_{\lambda,v} L(x,\lambda,v) \tag{4}$$

This is called the **Primal Problem**, which is equivalent to the original optimisation problem but in the primal form. By comparison, the **Lagrange Dual Problem** is basically swapping the inf and sup where $g(\lambda, v)$ is the (Lagrange) dual function.

$$d^* = \sup_{\lambda,v} d(\lambda,v) = g(\lambda^*) = \sup_{\lambda,v} g(\lambda,v) = \sup_{\lambda,v} \inf_x L(x,\lambda,v) \tag{5}$$

where x^* and λ^* are primal optimal point and dual optimal point respectively. To interpret this, the Lagrange dual function $g(\lambda, v)$ or $d(\lambda, v)$ is basically one lower bound of the optimal solution p^*. **The Lagrange Dual Problem** is basically a search for "the best lowerbound" d^*, which is the largest Lagrange dual function.

$$\begin{aligned} p^* &\geq g(\lambda,v) = d(\lambda,v) \\ d^* &= g(\lambda^*) = \sup_{\lambda,v} g(\lambda,v) \end{aligned}$$

The reason we introduce the dual problem for the primal/original problem is because it is usually easier to solve the dual problem. Also, d^* can be used as stopping criterior for primal optimization.

Definition 2.10. Weak Duality

For *any* optimisation problems that are not just convex, we have

$$p^* \geq d^* \text{ or } \inf_x \sup_{\lambda,v} L(x,\lambda,v) \geq \sup_{\lambda,v} \inf_x L(x,\lambda,v) \tag{6}$$

we call such inequality **Weak Duality** and the gap $p^* - d^*$ the duality gap. This inequality idea is from the **Weak Max-Min Inequality**, and we can memorize the order by "the one who max/sup first" is larger.

Proof.
$\forall \lambda_0, v_0 \geq 0, x_0 \in \text{dom} f$, we obviously have

$$\begin{aligned} \sup_{\lambda,v\geq 0} L(x_0,\lambda,v) &\geq L(x_0,\lambda_0,v_0) \geq \inf_{x\in \text{dom} f} L(x,\lambda_0,v_0) \\ \sup_{\lambda,v\geq 0} L(x_0,\lambda,v) &\geq \inf_{x\in \text{dom} f} L(x,\lambda_0,v_0) \end{aligned}$$

Since LHS holds for all fixed x_0 aka all x and RHS holds for all fixed λ_0 and v_0 aka for all λ and v, then the infimum of LHS is definitely greater than or equal to the supremum of RHS.

$$\begin{aligned} \inf_{x\in \text{dom} f} \sup_{\lambda,v\geq 0} L(x,\lambda,v) &\geq \sup_{\lambda,v\geq 0} \inf_{x\in \text{dom} f} L(x,\lambda,v) \\ p^* \geq d^* &= \sup_{\lambda,v} g(\lambda,v) \end{aligned} \tag{7}$$

□

We can interpret the RHS $d^* = \sup_{\lambda,v} g(\lambda, v)$ as the best lower bound.

Definition 2.11. Strong Duality

When optimisation problems are convex, we usually have **Strong Duality**, that is,

$$\begin{aligned} p^* &= d^* \\ f_0(x^*) &= g(\lambda^*) \end{aligned} \tag{8}$$

Definition 2.12. Slater's Constraint Qualifications for Strong Duality

For convex optimisation problem, most of time it will have strong duality but it is not always. In order to guarantee

strong duality, it needs to satisfy additional constraint called **constraint qualifications** and **Slater's Constraint** is one of them. It is sufficient condition for strong duality in a convex optimisation problem.

Slater's constraint qualification basically requires the problem to be **strictly feasible**, that is, find *one* solution that strictly satisfies the inequality constraint (e.g. $>$ instead of $\geq$). If the inequality constraints are *affine*, then being feasible instead of strictly feasible is enough (i.e. $\geq$ is okay too).

Definition 2.13. Complementary Slackness

Given a general optimisation problem, if we have strong duality (i.e. $p^* = d^*$) then we can find an interesting relationship between the optimal Lagrange multiplier λ_i^* and the optimum of the primal problem $f_0(x_i^*)$ and we call such relationship **Complementary Slackness**. This basically means that we always have *at least one* term in the equation that is 0.

$$\lambda_i^* f_0(x_i^*) = 0$$

Proof.

$$\begin{aligned} f_0(x^*) &= g(\lambda^*) \text{.... By definition of Strong Duality} \\ &= \inf_x L(x, \lambda^*) \\ &\leq L(x^*, \lambda^*) \\ &= f_0(x^*) + \sum_{i=0}^{m} \lambda_i^* f_i(x^*) \text{.... Expand Lagrangian} \\ &\leq f_0(x^*) \text{.... As inequality constraint } f_i(x) \leq 0 \end{aligned}$$

Hence $\sum_{i=0}^{m} \lambda_i^* f_i(x^*) \equiv 0$. Since each $f_i(x^*) \leq 0$ then each term $\lambda_i^* f_0(x_i^*) \equiv 0$ for any i in the range. □

Definition 2.14. KKT Conditions

Suppose we have a constrained optimisation problem that's NOT necessarily convex and the objective $f_0(x)$ is differentiable. Let x^*, λ^* and v^* be primal and dual optimal points where strong duality obtains (i.e. no duality gaps, $p^* = d^*$). Then those optimal points satisfy the following *necessary condition* called **KKT Conditions**

1. **Stationarity** The Lagrangian L has 0 first-order derivative at (x^*, λ^*, v^*).
$$\frac{\partial L(x^*, \lambda^*, v^*)}{\partial x} = \nabla f_0(x^*) + \lambda^T f(x^*) + v^T h(x^*) = 0$$
2. **Feasibility** Both inequality and equality constraints have to satisfy
$$f(x^*) \leq 0 \text{ and } h(x^*) = 0$$
3. **Non-negativity**
$$\lambda \geq 0$$
4. **Complementary Slackness** This actually indicates that $f(x^*) = 0 \forall i$ due to the non-negativity constraints.
$$\lambda^T f(x^*) = 0$$

If the optimisation problem is convex, then KKT conditions are not only necessary but also sufficient.

Definition 2.15. Taylor's Theorem

Taylor's Theorem basically provides an approximation of a k-th differentiable function $f(x)$ around point a given point a using a k-th order Taylor polynomial, assuming that $f(x)$ is k-th differentiable at $x = a$.

The following equation is the k-th order Taylor expansion of f(x) around point a

$$\begin{aligned} f(x) &= f(a) + f'(a)(x-a) + ... + \frac{f^k(a)}{k!}(x-a)^k + h_k(x)(x-a)^k \\ &= f(a) + f'(a)(x-a) + ... + \frac{f^k(a)}{k!}(x-a)^k + o((x-a)^k) \end{aligned}$$

where

- $P_k(x) = f(a) + f'(a)(x-a) + ... + \frac{f^k(a)}{k!}(x-a)^k$ is the k-th order Taylor polynomial
- $R_k(x) = f(x) - P_k(x) = h_k(x)(x-a)^k$ is the approximation error/remainder term.
- $\lim_{x \to a} h_k(x) = 0$.

Example. see w2 lecture notes

Definition 2.16. Small-o Notation

From Taylor's Theorem, we have $\lim_{x\to a} h_k(x) = 0$. Now we use the small-o $o((x-a)^k)$ to denote the remainder term such as $o((x-a)^k) = h_k(x)(x-a)^k$. Both equations are actually equivalent as

- $o((x-a)^k)$ means that when when x approaches a, $o((x-a)^k)$ converges to 0 faster than $(x-a)^k$ does.
- The ratio of small-o and the content inside the bracket also equals $h_k(x)$. The limiting behaviors of both equations are actually equivalent, such as

$$\lim_{x\to a} \frac{o((x-a)^k)}{(x-a)^k} = \lim_{x\to 0} h_k(x) = 0$$

Definition 2.17. Descent Method

Descent methods are proposed to solve problems where analytical method doesn't work well. Here we want to find the best hypothesis h^* that minimises the objective function in a convex function, so every step we take must lead to a smaller value until we find one. The general descent method is basically an iterative updating method.

In order for this algorithm to perform well, we need good descent direction d and step size η. But how can we find/choose them?

Definition 2.18. W2 Lec: Gradient Descent, d_k and η

Note that what we optimise here is h the hypothesis instead of x. Let $h_{k+1} = h_k + \eta d_k$ and then plug it back in first-order Taylor Series we have

$$\begin{aligned} f(h_{k+1}) &= f(h_k) + \nabla f(h_k)^T (h_{k+1} - h_k) + h_1'(h_k)(h_{k+1} - h_k) \\ &= f(h_k) + \nabla f(h_k)^T (\eta d_k) + o(\eta d_k) \\ &= f(h_k) + \eta \nabla f(h_k)^T d_k + o(\eta) \dots \ d_k \text{ usually a constant?} \end{aligned}$$

Remarks

- η: we can set η to be sufficiently small so that $o(\eta)$ is even smaller than η, which can be ignored.
- d_k: in order to make $f(h_{k+1}) - f(h_k) = \eta \nabla f(h_k)^T d_k < 0$ where $\eta > 0$, we should find some descent direction such that $\nabla f(h_k)^T d_k < 0$. For example, $d_k = -\nabla f(h_k)$.

Find d_k

- If we set $d_k = -D^k \nabla f(h_k)$ then

$$\begin{aligned} h_{k+1} &= h_k + \eta d_k \\ &= h_k - \eta D^k \nabla f(h_k) \\ f(h_{k+1}) &= f(h_k) - \eta \nabla f(h_k)^T D^k \nabla f(h_k) \end{aligned}$$

- Gradient/Steepest Descent with $D^k = I$

$$\begin{aligned} h_{k+1} &= h_k - \eta \nabla f(h_k) \\ f(h_{k+1}) &= f(h_k) - \eta \nabla f(h_k)^T \nabla f(h_k) \end{aligned}$$

- Newton's Method with $D^k = [\nabla^2 f(h)]^{-1}$

Find η

Definition 2.19. Find d the Descent Direction

1. **Gradient Descent Method**
   Given point x
   - **If x^* is optimal** then for convex f, for all d we have
   $$\lim_{\epsilon\to 0}[f(x^* + \epsilon d) - f(x^*)] = d\nabla f(x^*) \geq 0$$
   - **If x^* is not optimal** then we want to choose some search direction d to keep decreasing the f, i.e.
   $$\lim_{\epsilon\to 0}[f(x^{(k+1)} + \epsilon d) - f(x^{(k)})] = d\nabla f(x^{(k)}) \leq 0$$
   In this case, the decrease $d\nabla f(x^{(k)})$ would be the largest when we choose d in the opposite of $\nabla f(x^{(k)})$. Also we need to upperbound the $\|d\| \leq \delta$ so that the decrease $d\nabla f(x^{(k)})$ wouldn't overshoot. Here for convenience we choose $d = -\nabla f(x^{(k)})$ and then set $\delta = \left\|\nabla f(x^{(k)})\right\|_2$ to satisfy both conditions.

2. **Stochastic Gradient Descent**
   The gradient of loss function is approximated by a single example? make learning algorithsm more stable FILLL IT INNNNNNN LATER

3. **Steepest Descent Method**
   Note that this method will be the same as gradient descent if we take Euclidean Norm as the normalization style, that is, $d_{sd} = d_{gd} = -\nabla f(x)$. Otherwise (e.g. quadratic, l_1 norms), they are different.

4. **Newton's Method**
   Newton's Method is a second-order method (i.e. used $\nabla^2 f(x)$) while gradient descent method is a first-order method (i.e. only used $\nabla f(x)$). It basically uses the second-order Taylor approximation to approximate $f(x+d)$ at x,

$$\hat{f}(x+d) = f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x) d$$

   This is a convex quadratic equation and it is minimised at $d = d_{nt}$, just as illustrated as Figure 9. As we keep updating x and x is closer to x^*, the quadratic approximation would be better and better.

   Note that the Lipschitz Constant L evaluates how good Newton's Method is. For more detials, check **Definition Lipschitz Continuity** below.

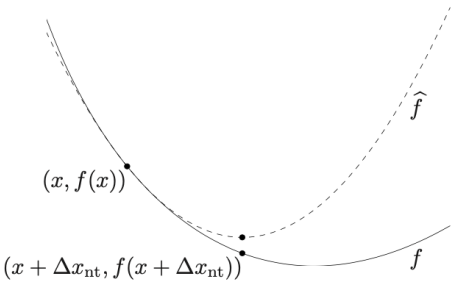


Figure 9: $\hat{f}$ is the second-order Taylor approximation of f at x, which is minimised at $x + d_{nt}$

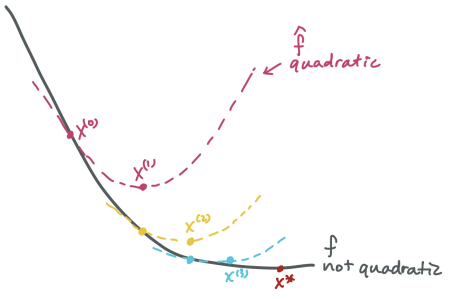


Figure 10: As x approaches x^*, $\hat{f}$'s minimum approaches f's minimum

Algorithms 2.20. Overview of Different Descent Methods

Algorithm 1: General Descent Method

Given Starting point $x^{(0)} \in \text{dom } f$
Iterate

1. Choose a descent direction d
2. Perform line search and choose a step size η
3. Update x: $x + \eta d$

Terminate Until the stopping criterion is satisfied (e.g. $\|f\|_2 \leq \eta$)

Algorithm 2: Gradient Descent Method

Given Starting point $x^{(0)} \in \text{dom } f$
Iterate

1. Descent direction $d_{gd} \equiv -\nabla f(x)$
2. Choose a step size η either through exact line search or backtracking line search
3. Update x: $x + \eta d_{gd}$

Terminate Until the stopping criterion is satisfied. Usually $\|\nabla f(x)\|_2 \leq$ some small and positive value η

Algorithm 3: (Damped/Guarded) Newton's Method

Given Starting point $x^{(0)} \in \text{dom } f$ and tolerance $\epsilon > 0$
Iterate

1. Compute **Newton Step** and **Newton Decrement** where
   (a) Newton step d_{nt} aka the descent direction
   $$\begin{aligned} d_{nt} &= -\nabla^2 f(x)^{-1} \nabla f(x) \\ &= -H^{-1} \nabla f(x) \end{aligned}$$
   (b) Newton Decrement λ^2 and $\lambda^2/2$ is used to express the closeness between f and $\hat{f}$
   $$\begin{aligned} \lambda^2 &= \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x) \\ &= \nabla f(x)^T H^{-1} \nabla f(x) \end{aligned}$$
2. Check stopping criterion: $\frac{\lambda^2}{2} \leq \epsilon$
3. Choose a step size backtracking line search
4. Update x: $x + \eta d_{nt}$

ProsCons 2.21. Pros & Cons of Different Descent Methods

Gradient Descent Method

- Pros: Simple as it's first-order method
- Cons: Convergence rate depends critically on condition number of sublevel sets (i.e. the contour's shape)

Newton's Method

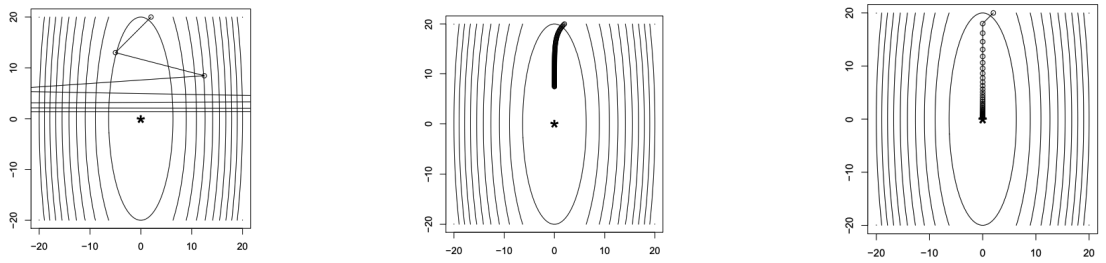
- Pros:
  - much faster convergence compared to GD. For example, superlinear convergence rate if objective function is strongly convex and has Lipschitz gradient.
  - scale well with problem size
- Cons:
  - calculate H^{-1} is expensive (i.e. $O(n^3)$)
  - cost of storing H

- Practical approximation methods for Newton's approach (https://www.cs.ubc.ca/ schmidtm/Courses/540-W18/L5.pdf)
  1. Only use the diagonals of Hessian matrix, which is cheap to store and invert.
  2. Only compute the Hessian every m iterations
  3. Modify Hessian matrix to be positive-definite
  4. Quasi-Newton Approximation, which approximates Hessian matrix by a diagonal plus low-rank approximation $B^t = D + UV^T$. This supports fast multiplication and inversion. Common choices are BFGS, L-BFGS.

Definition 2.22. Find η the Step Size

For step size η, it can either be fixed or varied during different iterations.
Line search refers to the procedure of choosing a step size η along the line $x + \eta d$ after choosing a descent direction d.

1. **Fixed** η. Really need to find the "right" fixed step size to make this approach work.



(a) η too big, 8 steps and diverging (b) η too small, 100 steps (c) η choice just right and 40 steps

Figure 11: Fixed η

2. **Exact line search** $\eta = \text{argmin}_{t \geq 0} f(x+td)$, where η is chosen to *exactly* minimise f along the line $\{f(x+td) | t \geq 0\}$. As Figure 12, for a given x and d, $x + td$ is a ray and each iteration we need to choose a t as η to minimise f. In gradient descent case, each iteration it will choose a t along the opposite direction of gradient to minimise f as much as possible. For a 2d convex function such as Figure 12(1), it would be easier to conduct binary search.

   Note that in practice, exact line search usually not work as well as backtracking line search.

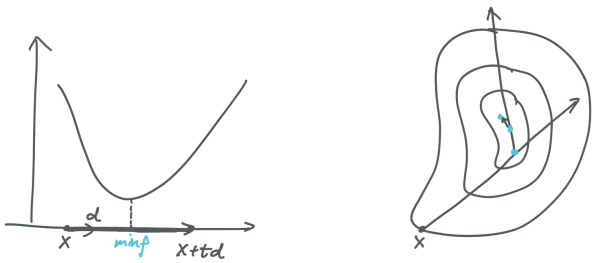


Figure 12: Exact Line Search Illustration

3. **(Armijo) Backtracking Line Search** One of the *inexact* line search methods that is simple and effective, where the step size is chosen to *approximately* minimise f along the line.

   As shown in figure below, $f(x) + t\nabla f(x)^T d$ is the linear extrapolation aka "Lower Bound" and $f(x) + \alpha t \nabla f(x)^T d$ is the "Upper Bound" that we are happy with, decrease measured as a factor of linear extrapolation's decrease. By keep iterating and shrinking t until $f(x + td)$ is below the UB, we can basically attain the minimal result we are happy with (as its below the UB we set).

   When t decreases to be small enough, by Taylor expansion, we have the first equation; The second equation is because $\nabla f(x)^T d \leq 0$.

$$f(x + td) \approx f(x) + t\nabla f(x)^T d \leq f(x) + \alpha t \nabla f(x)^T d$$

Algorithms 2.23. (Armijo) Backtracking Line Search

Given a point $x \in \text{dom } f$, a descent direction d, $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$
Initialise $t = 1$
While $f(x + td) > f(x) + \alpha t \nabla f(x)^T d$

(a) Shrink t: $t = \beta t$

Terminate Until $f(x + td) \leq f(x) + \alpha t \nabla f(x)^T d$ (equivalent to $t \leq t_0$)

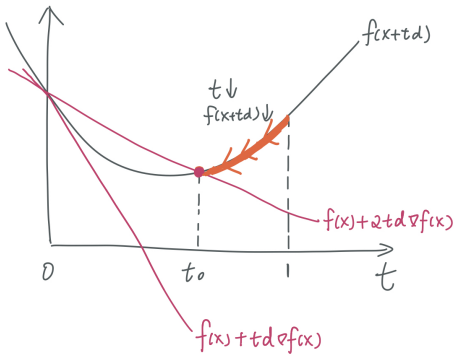


Figure 13: Keep shrinking t until below UB

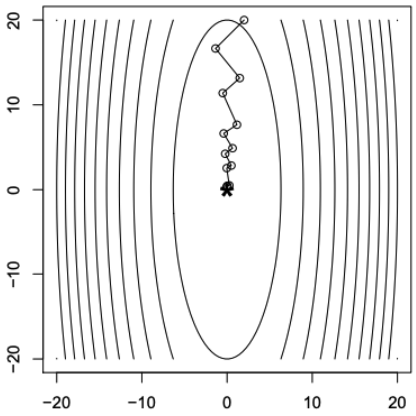


Figure 14: $\beta = 0.8$, 13 steps

In practice, α is typically chosen within (0.01, 0.3), meaning that we are happy with (1%, 30%) the decrease prediction of the linear extrapolation. β is typically chose within (0.1,0.8). The larger the value, the more detailed the search is.

It also shows that in practice given huge number of variables this method works better than exact line search.

Definition 2.24. Lipschitz Continuity and Quadratic Upperbound

We call *a function* f Lipschitz Continuous if $\forall x_1, x_2 \in \text{dom } f$, we have

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|$$

Note that here we didn't assume f is convex. L is called Lipschitz constant for function f. This means that for any pair of points in dom f, the slope connecting them will not be greater than L. This equation basically defines the function smoothness and limits how fast it can change.

Intuitively, if f is the $\nabla^2 f(x)$, we have $\left\|\nabla^2 f(x_1) - \nabla^2 f(x_2)\right\| \leq L\|x_1 - x_2\|$. L basically defines the magnitude of 3rd-derivative. The smaller the L, the *smaller* the 3rd-derivative, and the *better* the 2nd-order approximation/Newton's Method is.

Lipschitz Continuous Gradient
When the gradient function f is Lipschitz continuous, then $\forall x, y \in \text{dom } f$ we have

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \tag{9}$$

By Generalized Cauchy–Schwartz Inequality, we have

$$\|\nabla f(x) - \nabla f(y)\|^T (x - y) \leq L\|x - y\|^2 \tag{10}$$

Theorem 2.25. Suppose gradient function f is not only Lipschitz continuous but also *convex*, then we will have equation 10 equivalent to

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}\|x - y\|^2 \tag{11}$$

Where the RHS is the quadratic upperbound. In summary, equation $9 \Rightarrow 10 \equiv 11$

Proof (of the equivalence of (1) and (2) if dom f is convex)

- consider arbitrary $x, y \in \mathbf{dom}\, f$ and define $g(t) = f(x + t(y - x))$
- $g(t)$ is defined for $t \in [0, 1]$ because dom f is convex
- if (1) holds, then

$$g'(t) - g'(0) = (\nabla f(x + t(y - x)) - \nabla f(x))^T (y - x) \leq tL\|x - y\|^2$$

integrating from $t = 0$ to $t = 1$ gives (2):

$$\begin{aligned} f(y) = g(1) = g(0) + \int_0^1 g'(t)\, dt &\leq g(0) + g'(0) + \frac{L}{2}\|x - y\|^2 \\ &= f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}\|x - y\|^2 \end{aligned}$$

- conversely, if (2) holds, then (2) and the same inequality with x, y switched, *i.e.*,

$$f(x) \leq f(y) + \nabla f(y)^T (x - y) + \frac{L}{2}\|x - y\|^2,$$

can be combined to give $(\nabla f(x) - \nabla f(y))^T (x - y) \leq L\|x - y\|^2$

Figure 15: Proof of $10 \equiv 11$

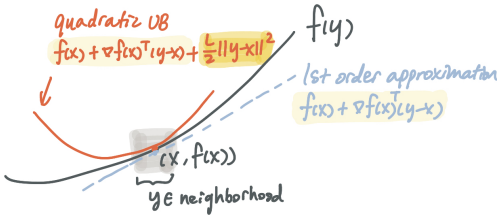


Figure 16: Quadratic Upperbound

Now in equation 11, replace x and y as x^k and $y = x^{k+1}$. Also set $x^{k+1} = x^k - \frac{1}{2L}\left\|\nabla f(x^k)\right\|^2$, then we have

$$f(x^{(k+1)}) \leq f(x^{(k)}) - \frac{1}{2L}\left\|\nabla f(x^{(k)})\right\|^2 \tag{12}$$

This basically means that we can decrease f by least $\frac{1}{2L}\left\|\nabla f(x^k)\right\|^2$ if we choose another iteration with step size $\frac{1}{L}$.

Definition 2.26. Strong Convexity and Quadratic Lowerbound

Recall that to tell if a function f is convex we use 3 ways including 1st-order and 2nd-order condition. Here we strengthen the condition a bit more but the logic is similar.

A function is called *m-strongly convex* if

1. "1st-order condition"
   (a) similar to the 1st-order condition of convex function one but with extra term $\frac{m}{2}\|x - y\|^2$

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2}\|x - y\|^2 \text{ for all x, y}$$

2. "2nd-order condition"
   (a) first $H - mI$ is positive semi-definite
   (b) second for all x eigenvalues of H must $\geq m$

$$\nabla^2 f(x) \succeq mI \Leftrightarrow \nabla^2 f(x) - mI \succeq 0 \text{ for all } x$$

Note that strong convexity $\rightarrow$ strict convexity but not other way around. From this strong convexity equation, we can also yield an equivalent equation for all x and y in dom f, where the RHS is the quadratic lowerbound.

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2}\|y - x\|^2 \tag{13}$$

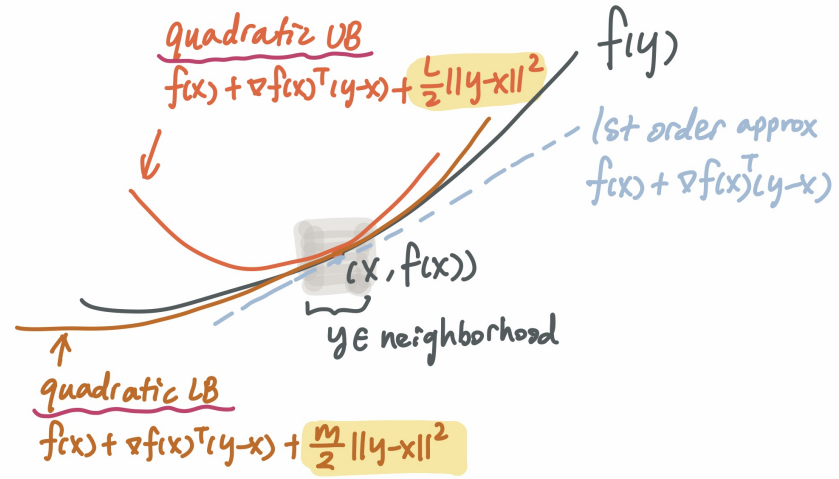


Figure 17: Quadratic Upperbound and Lowerbound of $f(y)$

Noticed that the RHS is actually a convex quadratic function in terms of y for fixed x and the minimiser is $\hat{y} = x - \frac{1}{m}\nabla f(x)$. Hence by plugging this minimiser in the RHS we have

$$\begin{aligned} f(y) &\geq f(x) + \nabla f(x)^T(y-x) + \frac{m}{2}\|y-x\|^2 \\ &\geq min(f(x) + \nabla f(x)^T(y-x) + \frac{m}{2}\|y-x\|^2) \\ &= f(x) + \nabla f(x)^T(\hat{y}-x) + \frac{m}{2}\|\hat{y}-x\|^2 \\ &= f(x) - \frac{1}{2m}\|\nabla f(x)\|^2 \end{aligned} \tag{14}$$

Hence $\forall x, y \in \text{dom } f$, we have

$$f(y) \geq f(x) - \frac{1}{2m}\|\nabla f(x)\|^2 \tag{15}$$

which also implies

$$p^* = f(x^*) \geq f(x) - \frac{1}{2m}\|\nabla f(x)\|^2 \tag{16}$$

where x^* is the global minimiser and p^* is the global minimum for f.

Definition 2.27. Gradient Convergence Rate

By rearranging Equation 12 and Equation 16, we have

$$\begin{aligned} &f(x^*) \geq f(x) - \frac{1}{2m}\|\nabla f(x)\|^2 \\ &f(x^{(k+1)}) \leq f(x^{(k)}) - \frac{1}{2L}\left\|\nabla f(x^{(k)})\right\|^2 \\ &m(f(x^{(k)}) - f(x^*) \leq \left\|\nabla f(x^{(k)})\right\|^2 \leq L(f(x^{(k)}) - f(x^{(k+1)})) \\ &m(f(x^{(k)}) - f(x^*)) \leq L(f(x^{(k)}) - f(x^{(k+1)})) \\ &f(x^{(k+1)}) - f(x^*) \leq (1 - \frac{m}{L})(f(x^{(k)}) - f(x^*)) \end{aligned} \tag{17}$$

By the last equation of 17, we have

$$\begin{aligned} f(x^{(k+1)}) - f(x^*) &\leq (1 - \frac{m}{L})(f(x^{(k)}) - f(x^*)) \\ &\leq (1 - \frac{m}{L})^2(f(x^{(k-1)}) - f(x^*)) \\ &\leq (1 - \frac{m}{L})^3(f(x^{(k-2)}) - f(x^*)) \\ &... \\ &\leq (1 - \frac{m}{L})^k(f(x^{(1)}) - f(x^*)) \end{aligned} \tag{18}$$

Since $c = 1 - \frac{m}{L} \leq 1$, we know that as $k \to \infty$,

- $c^k \to 0$
- error $f(x^{(k+1)}) - f(x^*) \to 0$
- $f(x^{(k+1)}) \to f(x^*) = p^*$

Gradient convergence rate basically talks about how many **iteration steps** the optimisation algorithms needs to iterate before we converge to the optimal solution h_S.

Linear convergence rate

If first objective function f is μ-strongly convex second gradient of f is L-Lipschitz continuous then we know that the current solution will converge to the optimal solution in a **linear convergence rate**. Mathematically,

$$f(h_{k+1}) - f(h_S) \leq (1 - \frac{\mu}{L})^k(f(h_1) - f(h_S))$$

This basically means that when $k \to \infty$, $(1 - \frac{\mu}{L})^k \to 0$ and so is the RHS. This makes $f(h_{k+1}) \to f(h_S)$ which is the optimal solution. That is, we've iterated enough steps (k) so that $f(h_{k+1})$ converges to the optimal solution.

1. Gradient Descent Algorithms
   (a) gradient is Lipschitz continuous + objective function is convex → convergence rate is $O(\frac{1}{k})$, which is sublinear (aka slower than linear convergence)

(b) gradient is Lipschitz continuous + objective function is strongly convex → convergence rate is $O((1-\frac{m}{L})^k)$, which is linear (aka linear convergence)

2. Newton Method:

   - gradient is Lipschitz continuous + objective function is strongly convex → convergence rate $O(\prod_1^k \rho_k)$ where $\rho_k \to 0$, which is superlinear (aka faster than linear convergence)

Remarks Note that algorithms with faster convergence rate (e.g. Newton's Method w superlinear convergence rate) does not necessarily take less *time* to reach the optimal solution. This is because here the whole convergence rate is with respect to *the number of iterations* required to converge instead of the time required to converge. It's "iteration"-complexity instead of time complexity. Also, the each iteration can take different time, we can't say much about total time required purely based on the number of iterations.

Algorithm	Assumption	Convergence rate
Gradient	Lipshitz Gradient, Convex	$O(1/k)$
Gradient	Lipshitz Gradient, Strongly-Convex	$O(1-\mu/L)^k$
Newton	Lipshitz Gradient, Strongly-convex	$\prod_{i=1}^k \rho_k$, $\rho_k \to 0$

Figure 18: Iteration Complexity of Different Algorithms

2.2 Loss Functions

Definition 2.28. 0-1 Loss

0-1 Loss is one form of loss functions for binary classification. To express in terms of margin m

$$\ell_{0-1}(m) = 1[m \leq 0]$$

As the loss plot shown, this loss is indeed

- Non-convex
- Not differentiable as it's not smooth at 0
- Not continuous at 0 (the plot curve is actually not accurate, at 0 shouldn't have vertical line)

which basically ended up to be NP-hard to solve.

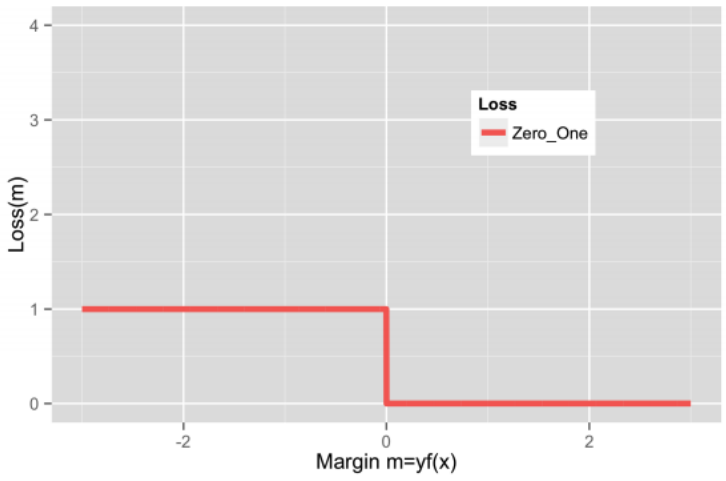


Figure 19: 0-1 Loss Illustration

Definition 2.29. Margin and Margin-Based Loss

Suppose all we deal with are classification problem and we use 0-1 loss

$$1_{\{y \neq sgn(f(x))\}}$$

then **margin** or functional margin m is defined as follows where $y \in \{-1, 1\}$. If our prediction is correct, the margin will be positive. Negative otherwise. And we want to *maximize our margin.*

$$m = \hat{y}y$$

Since most classification losses only depend on the margin m, we call such a loss **Margin-Based Loss**.

Definition 2.30. Surrogate Loss

As mentioned in previously, 0-1 loss function has multiple cons such as non-convex, non-smooth and NP-hard to optimise. A common solution is to use (convex and smooth) surrogate loss function, which upper bounds the 0-1 loss and also are easy to optimise.

One thing to note is that, although most common surrogate loss functions are convex and smooth (usually we'd also like to choose those), not all surrogate loss functions have to be convex and smooth everywhere as long as they can be optimised in some way. Some non-convex surrogate loss functions can actually have some nice properties which are desirable. So we should choose surrogate loss based on the desirable properties instead of purely the convexity and smoothness.

For example, TL mentioned in lecture that SVM uses Hinge loss and gradient descent even when Hinge loss is non-smooth at m=1. This is because the dual is smooth and can still be optimised.

Example. Common Surrogate Loss Functions

1. **Hinge Loss (convex)**

$$\ell_{hinge} = \phi_{hinge}(m) = \max\{1 - m, 0\}$$

   - Pros: convex, continuous, upper bound on 0-1 loss
   - Cons: not differentiable at $m = 1$
   - Remarks
     - Though the function is not differentiable at $m = 1$, it can still be optimized due to its convexity.
     - have "margin error" when $m \in (0, 1)$. During this interval, the prediction is correct but there is still error. Intuitively, this margin error pushes the function to obtain higher margin with higher confidence.
     - Unlike 0-1 loss, it penalises margin proportionally. The smaller the margin, the larger the loss is.
     - SVM uses Hinge Loss as the surrogate loss

2. **Logistic Loss (convex)**

$$\ell_{logistic} = \phi_{logistic}(m) = \log_2(1 + e^{-m})$$

   - Pros: differentiable everywhere
   - Remarks:
     - The viz is incorrect, should go through (0,1). Unlike Hinge loss, the loss is never 0. And it will always push for larger margin hence less loss.
     - For classification problem, cross-entropy loss and logistic loss (i.e. log loss) are equivalent.

3. **Exponential Loss (convex)**

$$\ell_{exp} = \phi_{exp}(m) = e^{-m}$$

   - Remarks
     - Boosting algorithms use it as its surrogate loss function.

4. **Least Square Loss (convex)**

$$\begin{aligned} \ell_{least-square} &= (f(x) - y)^2 \\ &= f(x)^2 + y^2 - 2f(x)y \\ &= 2 - 2f(x)y \\ &= (1 - f(x)y)^2 \\ &= (1 - m)^2 \end{aligned}$$

   - Remarks
     - It heavily penalizes outliers or mislabelled examples
     - May have higher sample complexity and need larger sample size than Hinge or Logistic loss

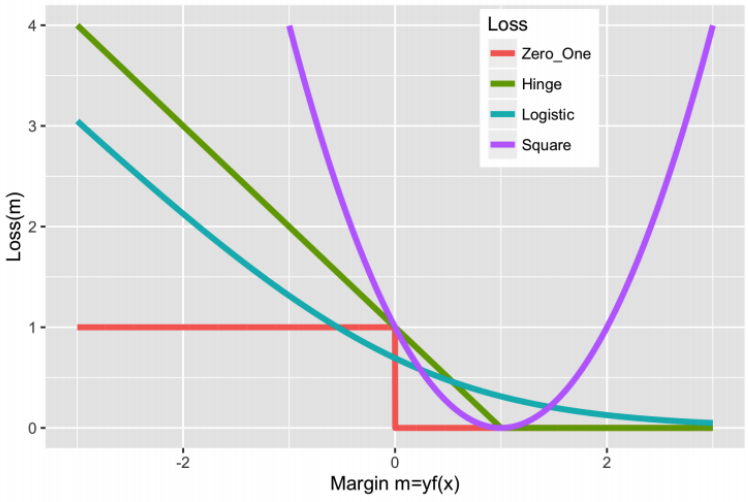


Figure 20: Convex Surrogate Loss Functions (Not all smooth)

5. **Cauchy Loss (non-convex)**

$$\ell_{cauchy} = \log_2(1 + (\frac{1-m}{\delta})^2)$$

6. **Correntropy Loss/Welsch Loss (non-convex)**

$$\ell_{correntropy} = 1 - e^{-(\frac{1-m}{\delta})^2}$$

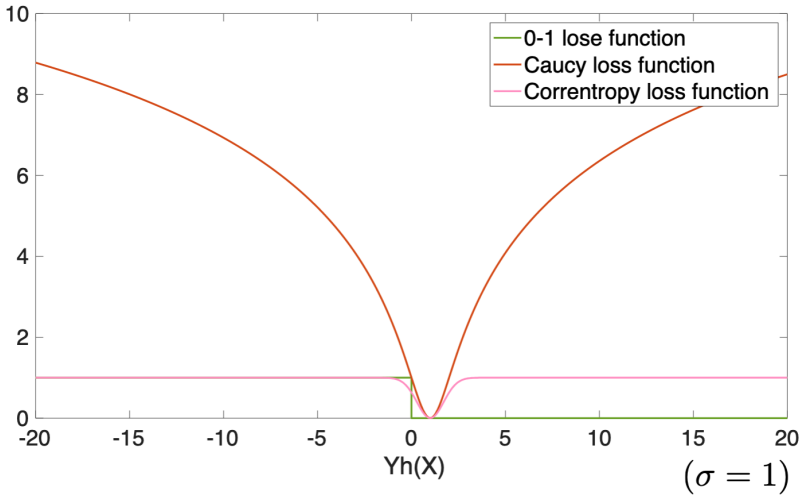


Figure 21: Non-convex Surrogate Loss

Definition 2.31. Classification Calibrated: Definition and Checking Method

A surrogate loss function ℓ is **classification calibrated/Bayes consistent** if given enough training data, using the loss function will result in a classifier with the same risk as Bayes classifier. Formally, ℓ is classification calibrated/Bayes consistent if

$$\mathbb{E}[1_{\{y \neq sgn(h_1(x))\}}] \xrightarrow{N \to \infty} \mathbb{E}[1_{\{y \neq sgn(h_2(x))\}}]$$

where

- $h_1 = \text{argmin}_h \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i)$, which is the empirical risk minimiser using surrogate loss function
- $h_2 = \text{argmin}_h \mathbb{E}[1_{\{y \neq sgn(h(x))\}}]$, which is the Bayes classifier using 0-1 loss function

Remarks

- the consistency is defined over the expected risk using *0-1 loss function*, after h_1 was substituted into the equation
- most popular surrogate loss functions are classification-calibrated

Checking Method

Given $\phi(.)$ is a convex function and rewrite the loss function as $\phi(m) = \ell(X, Y, h)$

surrogate loss ℓ is classification-calibrated $\Leftrightarrow \phi^{'}(0)$ exists (aka differentiable at 0) and $\phi^{'}(0) < 0$

Note that this checking method only works if $\phi(.)$ is convex.

3 Hypothesis Complexity and Generalisation Error

3.1 Background Knowledge: Risk

Definition 3.1. Predefined Hypothesis Class and Universal Function Space

- Universal Function Space $\mathcal{U}$: includes all the possible hypotheses or all sorts of functions
- Predefined Hypothesis Class $\mathcal{H}$: $\mathcal{U}$'s subset and where we choose our hypotheses from.

> **Examples.** For Linear SVM model, $\mathcal{H}$ we choose will be a set of linear functions (in Euclidean space). For kernel SVM model, $\mathcal{H}$ we choose will be a set of non-linear functions (in Reproducing kernel Hilbert space (RKHS)). Which one to choose? Depends on the problem.

Definition 3.2. Risk and Empirical Risk

Suppose there's some unknown distribution P_{XY} and n sample points $S = \{(x_1, y_1), ..., (x_n, y_n)\}$ were generated from this distribution. Also, suppose we have some hypothesis h.

Then **Risk, or Expected Risk, True Error** is defined as

$$R(h) = \mathbb{E}[\ell(X, Y, h)]$$

The expected risk is incalculable as P_{XY} is unknown. But we can always estimate it through the **Empirical Risk** using the observed n samples such as

$$R_S(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i, h)$$

And we know that

$$\mathbb{E}[R_S(h)] = R(h)$$

Definition 3.3. Three Risk Minimisers and their Relationship

.

Three Minimisers

1. **Target Concept** c, also called the Bayes classifier, is the best expected risk minimiser over the Universal Function Space $\mathcal{U}$.
$$c = \arg\min_{h \in \mathcal{U}} R(h)$$
Remarks:
   - Defined over expected risk
   - When $\mathcal{H}$ equals to $\mathcal{U}$, c in $\mathcal{H}$; When $\mathcal{H}$ not equals to $\mathcal{U}$, c *could* be in $\mathcal{H}$ or out. If in, c will be hard to learn as it requires much more data to feed the class complexity.
   - $R(c)$ the Bayes Risk can be 0 if the data is *separable*, though most cases it is *non-separable* which makes $R(c) > 0$.
2. **Expected Risk Minimiser** h^* is the best expected risk minimiser over the predefined hypothesis space $\mathcal{H}$
$$h^* = \arg\min_{h \in \mathcal{H}} R(h)$$
Remarks:
   - Defined over expected risk, too
   - "best in class"
   - h^* and c can equate depending on how big $\mathcal{H}$ is.
3. **Emipirical Risk Minimiser (ERM)** h_S is the best empirical risk minimiser (i.e. lowest training error) over the predefined hypothesis space $\mathcal{H}$.
$$h_S = \arg\min_{h \in \mathcal{H}} R_S(h)$$
Remarks:
   - Defined over empirical risk instead
   - something we can *actually learn* by using the samples S

.

Relationships Between the Three Minimisers

1. **Estimation Error** the error caused by estimating h^* using h_S
$$R(h_S) - R(h^*)$$
2. **Approximation Error** the difference between best in class expected risk and Bayes risk
$$R(h^*) - R(c)$$
3. **Excess Risk**
$$\begin{aligned}\text{Excess Risk} &= \text{Estimation Error} + \text{Approximation Error} \\ &= [R(h_S) - R(h^*)] + [R(h^*) - R(c)]\end{aligned}$$

Remarks:

- Goal is to carefully choose $\mathcal{H}$ such that the $h_S \in \mathcal{H}$ learned from samples S can minimise the excess risk (hence h_S is closest to c risk-wise). This minimisation is done by carefully balancing between the estimation error and the approximation error.
- If we purposely choose a big and complex $\mathcal{H}$ such that $c \in \mathcal{H}$, we can shrink approximation error to be 0. However it would make it very difficult to learn h_S as now there are many more parameters in such hypothesis under the complex $\mathcal{H}$. Given the same amount of training data, the estimation error would be huge and we need much more training data to decrease estimation error.

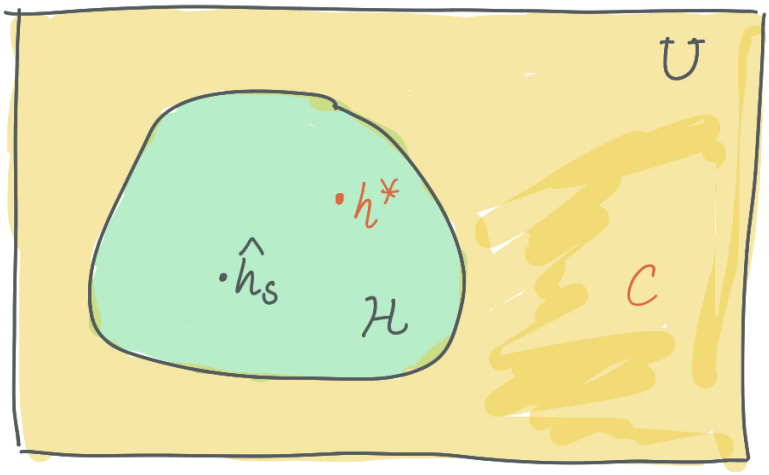


Figure 22: Excess Risk Decomposition: Relationship Between Three Minimisers

3.2 PAC Learning Framework

When it comes to a hypothesis h, there are few more fundamental questions

- What can be learned efficiently (i.e. requires only poly-many samples)?
- What is hard to learn (i.e. requires expo-many samples)?
- How many training samples do we need so we can learn the best possible hypothesis in class h^* through h_S? (i.e. sample complexity, poly-many samples)
- Is there a general model of learning?

The **Probably Approximately Correct(PAC) Learning Framework** helps answer those questions. Essentially, this framework defines a set of learnable hypothesis classes or learnable function classes (i.e. a bunch of $\mathcal{H}$s) by their *sample complexity*.

Definition 3.4. PAC Learnable

we call the hypothesis class $\mathcal{H}$ **PAC-learnable** if

1. For polynomially-many samples S

2. For any distribution D (i.e. task) where S were sampled from

there exists an learning algorithm $\mathcal{A}$ such that any h_S learned by $\mathcal{A}$ through S satisfies

$$P[R(h_S) - R(h^*) \leq \epsilon] \geq 1 - \delta \ . \forall \delta, \epsilon$$

Remarks

- PAC-learnable is about the hypothesis class or function class $\mathcal{H}$
- The essence of $\mathcal{H}$'s PAC-learnability is the best-in-class hypothesis h^*'s learnability through h_S. If $\mathcal{H}$'s h^* can be learned "probably approximately correctly" and "efficiently" using h_S, i.e.
  1. probably approximately correctly: with at least $1-\delta$ probability (probably correct), h_S is only ϵ-different from h^* (approximately correct)
  2. efficiently: it only requires poly-many samples for an learning algorithm to learn such h_S which satisfies "probably approximately correctly" inequality.
- **Sample Complexity** is the minimum number of samples required for the "probably approximately correctly" inequality to hold. If sample complexity is poly-many, then such sample complexity is considered efficient/desirable under the PAC learning framework.
- PAC-learning framework defined in this course only cares about sample complexity instead of time or space complexity. Sample complexity cares about accuracy while time and space complexity cares about computational efficiency. That is, if h^* of $\mathcal{H}$ can be learned accurately (i.e. probably approximately correctly) using polynominally-many samples. If $\mathcal{H}$ is too complex and requires exponentially many samples (i.e. $n > exp(\frac{1}{\epsilon}, \frac{1}{\delta})$ instead of (i.e. $n > poly(\frac{1}{\epsilon}, \frac{1}{\delta})$)), then $\mathcal{H}$ is *not* PAC-learnable.
- Note that the gap between h_S and h^* is defined by the estimation error $(R(h_S) - R(h^*))$, where both errors defined using expected risk. Please don't confuse with the generalisation error $R(h_S) - R_S(h_S)$ or $R(h_S)$.

Definition 3.5. Check if $\mathcal{H}$ is PAC-learnable

Given a hypothesis $\mathcal{H}$, if we want to know if it's PAC-learnable, we need to first find a learning algorithm $\mathcal{A}$ and also a polynomial function $poly()$.

1. $\mathcal{A}$:
   - $\mathcal{A}$ is a mapping of objective function and optimisation method to the output hypothesis h_S.
   - For example, $\mathcal{A}$ can be the empirical risk minimisation algorithm and h_S the empirical risk minimiser will be selected from all the hypotheses from $\mathcal{H}$
2. $poly()$:
   - $poly()$ is a polynomial function, usually as $poly(\frac{1}{\delta}, \frac{1}{\epsilon})$
   - used to check if the sample complexity n is in this format

After obtaining h_S as the output of $\mathcal{A}$, we apply this h_S to the PAC inequality and verify if it holds for requirements such as if n is in poly() format, where n is usually rearranged from $\delta =$ some UB.

There are two approaches to check if $\mathcal{H}$ is PAC-learnable.

1. Approach 1: if $\mathcal{H}$ contains finite hypotheses (i.e. $|\mathcal{H}| < \infty$), then $\mathcal{H}$ is PAC-learnable. Check proof in Proof 3.4
2. Approach 2: if the VC dimension of $\mathcal{H}$ is finite (i.e. $VCdim(\mathcal{H}) = d < \infty$), then $\mathcal{H}$ is PAC-learnable.

The proof is in the end of this week.

Definition 3.6. Generalisation Error and Generalisation Error Bound

.

Generalisation Error examines how well the classifier h generalises over unseen data outside the observed sample S. There are two common definitions of generalisation error. The first one is the difference between testing error and training error, and the second one is just the testing error.

1. $R(h_S) - R_S(h_S) =$ Out-Sample Testing Error - In-Sample Training Error
2. $R(h_S) =$ Out-Sample Testing Error

We want the generalisation error of learned hypothesis h_S to be small or upperbounded by some value, so we know h_S can generalise well.

Generalisation Error Bound is also defined correspondingly in two ways, depending on which Generalisation Error definition we use.

1. $R(h_S) - R_S(h_S) \leq \sup_{h\in\mathcal{H}} |R(h) - R_S(h)|$
2. $R(h_S) \leq R_S(h_S) + \sup_{h\in\mathcal{H}} |R(h) - R_S(h)|$ More common definition

$R_S(h_S)$ is the training error and we can optimise by ourselves. $\sup_{h\in\mathcal{H}} |R(h) - R_S(h)|$ is the maximum generalisation error in the class which we don't know about. So naturally we want to upperbound it so we can then upperbound the generalisation error of h_S altogether.

Asymptotically, we can upperbound $\sup_{h\in\mathcal{H}} |R(h) - R_S(h)|$ by 0 when $n \to \infty$. However, when n is not infinitely-large, how can we find the upperbound of $\sup_{h\in\mathcal{H}} |R(h) - R_S(h)|$? The answer is that we use some non-asymptotical measurements such as concentration inequalities. Below are the two popular concentration inequalities and in this course we focus on the first one.

- Hoeffding's Inequality
- McDiarmid's Inequality

Definition 3.7. Bound Generalisation Error using Hoeffding's Inequality

Hoeffding's Inequality Suppose $X_1, ..., X_n$ are n independent variables such and $X_i \in [a_i, b_i]$. Also let S_n be the sample average of those n independent variables. Then $\forall \epsilon > 0$ we have

$$P(|\mathbb{E}[S_n] - S_n| \geq \epsilon) \leq 2\exp\left(\frac{-2n^2\epsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$
$$\Leftrightarrow P(|\mathbb{E}[S_n] - S_n| \leq \epsilon) \geq 1 - 2\exp\left(\frac{-2n^2\epsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

This is the general definition of Hoeffding Inequality. It basically provides a way for us to upperbound the error gap between the sample average and the expectation in probability. In our case, since generalisation error is the error gap between the testing error (expectation) and the training error (sample average), we can use this definition to upperbound the generalisation error. Suppose $\ell(X_1, Y_1, h), ..., \ell(X_n, Y_n, h)$ are independent since (X_i, Y_i) are independent. Suppose $\forall i, \ell(X_i, Y_i, h) \in [0, M]$. Then $\forall \epsilon > 0$, by Hoeffding's Inequality we have

$$P(|R(h) - R_S(h)| \geq \epsilon) \leq 2\exp\left(\frac{-2n\epsilon^2}{M^2}\right)$$
$$\Leftrightarrow P(|R(h) - R_S(h)| \leq \epsilon) \geq 1 - 2\exp\left(\frac{-2n\epsilon^2}{M^2}\right)$$

where $R_S(h)$ is the empirical risk and $R(h)$ is the expected risk.

Definition 3.8. Upperbound "Estimation Error"

.

Recall that under PAC framework, our goal is to upperbound estimation error $R(h_S) - R(h^*)$ within the probability bracket. We basically upperbound this error in two steps.

Theorem 3.9. With at least $1 - \delta$ probability we have

$$R(h_S) - R(h^*) \leq 2M\sqrt{\frac{\log|\mathcal{H}| + \log\frac{2}{\delta}}{2n}}$$

where

- $\epsilon = M\sqrt{\frac{\log|\mathcal{H}| + \log\frac{2}{\delta}}{2n}}$
- $\delta = 2|\mathcal{H}|\exp\left(\frac{-2n\epsilon^2}{M^2}\right)$
- $n = \frac{M^2}{2\epsilon^2}\log\frac{2|\mathcal{H}|}{\delta}$.

Proof. Derivation Logic of this Estimation Error Upperbound

Step 1 Find the intermediate upper bound of estimation error, which 2 × generalisation error bound.

$$R(h_S) - R(h^*) \leq 2 \sup_{h \in \mathcal{H}} |R(h) - R_S(h)|$$

Step 2 Find the upper bound for the intermediate upperbound through Hoeffding's Inequality

$$\sup_{h \in \mathcal{H}} |R(h) - R_S(h)| \leq M \sqrt{\frac{\log |\mathcal{H}| + \log \frac{2}{\delta}}{2n}}$$

Then link two steps together, we have

$$P\Big(R(h_S) - R(h^*) \leq 2M \sqrt{\frac{\log |\mathcal{H}| + \log \frac{2}{\delta}}{2n}}\Big) \geq 1 - \delta$$

□

Corollary 3.10. Step 1 Find the intermediate upper bound of estimation error, which 2 × generalisation error bound.

$$R(h_S) - R(h^*) \leq 2 \sup_{h \in \mathcal{H}} |R(h) - R_S(h)|$$

Proof.

$$\begin{aligned}
&R(h_S) - R(h^*) \\
&= (R(h_S) - R_S(h_S)) + (R_S(h_S) - R_S(h^*)) + (R_S(h^*) - R(h^*)) \\
&\leq (R(h_S) - R_S(h_S)) + (R_S(h^*) - R(h^*)) \text{....[Note: } R_S(h_S) = \arg\min R_S \leq R_S(h^*)] \\
&\leq |R(h_S) - R_S(h_S)| + |R_S(h^*) - R(h^*)| \\
&\leq \sup_{h \in \mathcal{H}} |R(h) - R_S(h)| + \sup_{h \in \mathcal{H}} |R(h) - R_S(h)| \text{....[Note: since both } h_S, h^* \in \mathcal{H} \text{ and upper bounded by sup]} \\
&= 2 \sup_{h \in \mathcal{H}} |R_S(h) - R(h)|
\end{aligned}$$

□

Corollary 3.11. Step 2 Find the final upper bound for $\sup_{h \in \mathcal{H}} |R_S(h) - R(h)|$ through Hoeffding's Inequality. With at least $1 - \delta$ probability we have

$$\sup_{h \in \mathcal{H}} |R(h) - R_S(h)| \leq M \sqrt{\frac{\log |\mathcal{H}| + \log \frac{2}{\delta}}{2n}}$$

Proof.

$$\begin{aligned}
P(\sup_{h \in \mathcal{H}} |R(h) - R_S(h)| \geq \epsilon) &\leq P(\bigcup_{h \in \mathcal{H}} |R(h) - R_S(h)| \geq \epsilon) \\
&\leq \sum_{h \in \mathcal{H}} P(|R(h) - R_S(h)| \geq \epsilon) \\
&\leq \sum_{h \in \mathcal{H}} 2 \exp\Big(\frac{-2n\epsilon^2}{M^2}\Big) \text{....[Note: by Hoeffding's Inequality]} \\
&= 2|\mathcal{H}| \exp\Big(\frac{-2n\epsilon^2}{M^2}\Big) \\
\Leftrightarrow P(\sup_{h \in \mathcal{H}} |R(h) - R_S(h)| \geq \epsilon) &\leq 2|\mathcal{H}| \exp\Big(\frac{-2n\epsilon^2}{M^2}\Big) = \delta \\
\Leftrightarrow P(\sup_{h \in \mathcal{H}} |R(h) - R_S(h)| \leq \epsilon) &\geq 1 - 2|\mathcal{H}| \exp\Big(\frac{-2n\epsilon^2}{M^2}\Big) = 1 - \delta
\end{aligned}$$

Hence with at least $1 - \delta$ probability we have

$$\sup_{h \in \mathcal{H}} |R(h) - R_S(h)| \leq \epsilon = M \sqrt{\frac{\log |\mathcal{H}| + \log \frac{2}{\delta}}{2n}}$$

where $|\mathcal{H}|$ denotes the hypothesis complexity

where

- $\delta = 2|\mathcal{H}| \exp\Big(\frac{-2n\epsilon^2}{M^2}\Big)$
- $\epsilon = M \sqrt{\frac{\log |\mathcal{H}| + \log \frac{2}{\delta}}{2n}}$
- $n = \frac{M^2}{2\epsilon^2} \log \frac{2|\mathcal{H}|}{\delta}$

.

The first line holds because

1. LHS event $\sup_{h\in\mathcal{H}} |R(h)-R_S(h)|$ is one of the union of events ($|R(h_1)-R_S(h_1)| \geq \epsilon$ or ... or $|R(h_k)-R_S(h_k)| \geq \epsilon$)
2. If event $\sup_{h\in\mathcal{H}} |R(h)-R_S(h)|$ holds, the union of events must hold as one of them is satisfied through "or".
3. Hence $\sup_{h\in\mathcal{H}} |R(h)-R_S(h)|$ implies ($|R(h_1)-R_S(h_1)| \geq \epsilon$ or ... or $|R(h_k)-R_S(h_k)| \geq \epsilon$), which means LHS has a smaller probability as it's more strict.

□

.

Remarks From the inequality below we can also see that

$$P\Big(R(h_S) - R(h^*) \leq 2M\sqrt{\frac{\log|\mathcal{H}| + \log\frac{2}{\delta}}{2n}}\Big) \geq 1-\delta$$

1. when $|\mathcal{H}| < \infty$, from approach 1 we know that $\mathcal{H}$ is PAC-learnable and we can derive the generalisation error bound
$$R(h_S) - R_S(h_S) \leq \sup_{h\in\mathcal{H}} |R(h) - R_S(h)| \leq M\sqrt{\frac{\log|\mathcal{H}| + \log\frac{2}{\delta}}{2n}}$$
2. when $|\mathcal{H}| = \infty$, is $\mathcal{H}$ still PAC-learnable? Can we derive a generalisation error bound too? Here we need to introduce VC-dimension to help answer the question. (reduce the class into finite hypothesis groups where hypotheses in each group have the same classifications)

3.3 Dichotomy, Growth Function and Shattering

Previously we have yield the bound for $\sup_{h\in\mathcal{H}} |R(h) - R_S(h)|$ under finite hypotheses. However, when it comes to infinite hypotheses ($|\mathcal{H}| \to \infty$), the previous bound becomes infinity and not working anymore. Is there any way to bound $|\mathcal{H}|$? Can we reduce it?

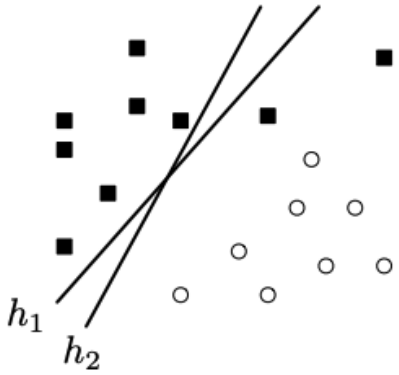


Figure 23: h_1 and h_2 result in the same labelling and empirical risks over the sample data

Just as Figure 23 shown, many hypotheses in $\mathcal{H}$ actually ended up having the same labelling and empirical risks over the sample data. $|\mathcal{H}|$ actually overestimates how complex our hypothesis space is. This motivates us to introduce the concept of dichotomy and growth function, which better captures the richness of $\mathcal{H}$ and enable us to find the bound under infinite hypotheses as well.

Definition 3.12. Dichotomy

Let $x_1, ..., x_n$ be n training samples from $\mathcal{X}$. Then the **dichotomies** generated by $\mathcal{H}$ over all those samples are basically just a set of unique classification outputs. Mathematically, it is defined as

$$\begin{aligned}\mathcal{H}(x_1, ..., x_n) &= \{(h(x_1), ..., h(x_n)) | \forall h \in \mathcal{H}\} \\ &= \{(h^1(x_1), ..., h^1(x_n)), (h^2(x_1), ..., h^2(x_n)), (h^3(x_1), ..., h^3(x_n)), ..., (h^G(x_1), ..., h^G(x_n))\} \\ \mathcal{H}' &= \{h_1, h_2, ..., h_G\}\end{aligned} \tag{19}$$

assuming $\mathcal{H}$ can have G unique classification outputs over all training data. This means we can group hypotheses (especially when infinite) into G group without changing the classification output or empirical risk, where $h^1, h^2, h^3, ...$ are the representative hypothesis from all G hypothesis groups. All hypothese inside each group i end up with the same

classification output for all training data.

For example, given $\mathcal{H}$ is a set for linear hypotheses only and there are huge number of even infinite hypotheses inside the set. Suppose we only consider binary classification problem and the case when $n = 3$. If the sample points are given as Figure 24 (not in straight line), then by applying all hypotheses inside $\mathcal{H}$ to the sample points we end up having the following 8 distinct dichotomies. We can think each yellow line as the representative hypothesis of 8 hypothesis groups.

$$\mathcal{H}(x_1, x_2, x_3) = \{(-1,-1,-1), (+1,-1,-1), (+1,-1,+1), (+1,+1,-1), (-1,+1,-1), (-1,-1,+1), (-1,+1,+1), (+1,+1,+1)\}$$

However, the same set of hypotheses in $\mathcal{H}$ can end up having different sets of dichotomies depending on the position of sample data points. For example, by applying the same hypotheses inside $\mathcal{H}$ to the sample points given as Figure 25 we end up having 6 distinct dichotomies.

$$\mathcal{H}(x_1', x_2', x_3') = \{(-1,-1,-1), (+1,-1,-1), (+1,+1,-1), (-1,-1,+1), (-1,+1,+1), (+1,+1,+1)\}$$

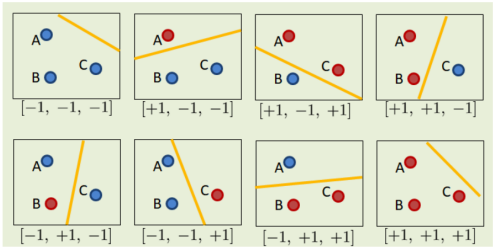


Figure 24: 3 points not in line

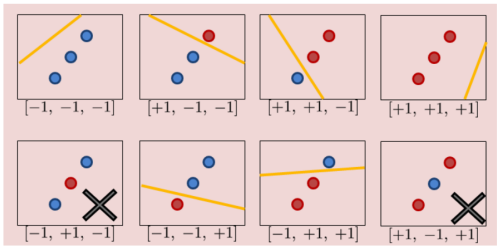


Figure 25: 3 points in line

Definition 3.13. Growth Function

In order to overcome the "same hypotheses different dichotomies" caused by different sample arrangements and find G and $\mathcal{H}' = \{h_1, h_2, ..., h_G\}$ (so we can simplify infinite $\mathcal{H}$ then apply upperbound equation), the concept of **Growth Function** is introduced to better measure the richness of $\mathcal{H}$. The growth function for a hypothesis set $\mathcal{H}$ over n sample points is defined as

$$\Pi_{\mathcal{H}}(n) = \max\{|\mathcal{H}(x_1, ..., x_n)| : \forall (x_1, ..., x_n) \in \mathcal{X}\}$$

which is the maximum number of dichotomies/maximum number of groups G $\mathcal{H}$ can have over all possible arranges of n samples. The growth function is about exploring the possibilities of sample points and it is dependent on n instead of one exact arrangement. For example, given 3 data points as in previous sections, we can either arrange them inline or not, but the growth function for such 3-point scenario is the same, which is $\Pi_{\mathcal{H}}(3) = 8$.

Definition 3.14. Shattering

A set S with $n \geq 1$ sample points is said to be **shattered** by hypothesis set $\mathcal{H}$ if $\mathcal{H}$ realises all possible dichotomies in S. That is,

$$\Pi_{\mathcal{H}}(n) = 2^n$$

For binary classification, the most dichotomies generated by $\mathcal{H}$ on n points are $\Pi_{\mathcal{H}}(n) = 2^n$. For example, in Figure 24, S is shattered by $\mathcal{H}$ as $\Pi_{\mathcal{H}}(3) = 2^3 = 8$.

3.4 VC-Dimension

Definition 3.15. VC-Dimension

The **VC-Dimension** of hypothesis set $\mathcal{H}$ is the size/n of largest set S that can be shattered by $\mathcal{H}$. Though generalisation exists, VC-Dimension is usually only defined for binary classifiers. Hence formally we have

$$\text{VCdim}(\mathcal{H}) = d_{VC} = \max\{n : \Pi_{\mathcal{H}}(n) = 2^n\}$$

This means if $d_{VC} = d$, there must exist a set S with size d that can be shattered by $\mathcal{H}$.

Examples.

1. When predefined hypothesis class is an interval function class, i.e.
$$\mathcal{H} = \{x \to 1_{\{x \in (a,b)\}} : a < b \in \mathbb{R}\}$$
according to the following derivation, the largest number of samples that can be fully shattered by the interval function is 2, hence $VCdim_{\mathcal{H}} = 2$

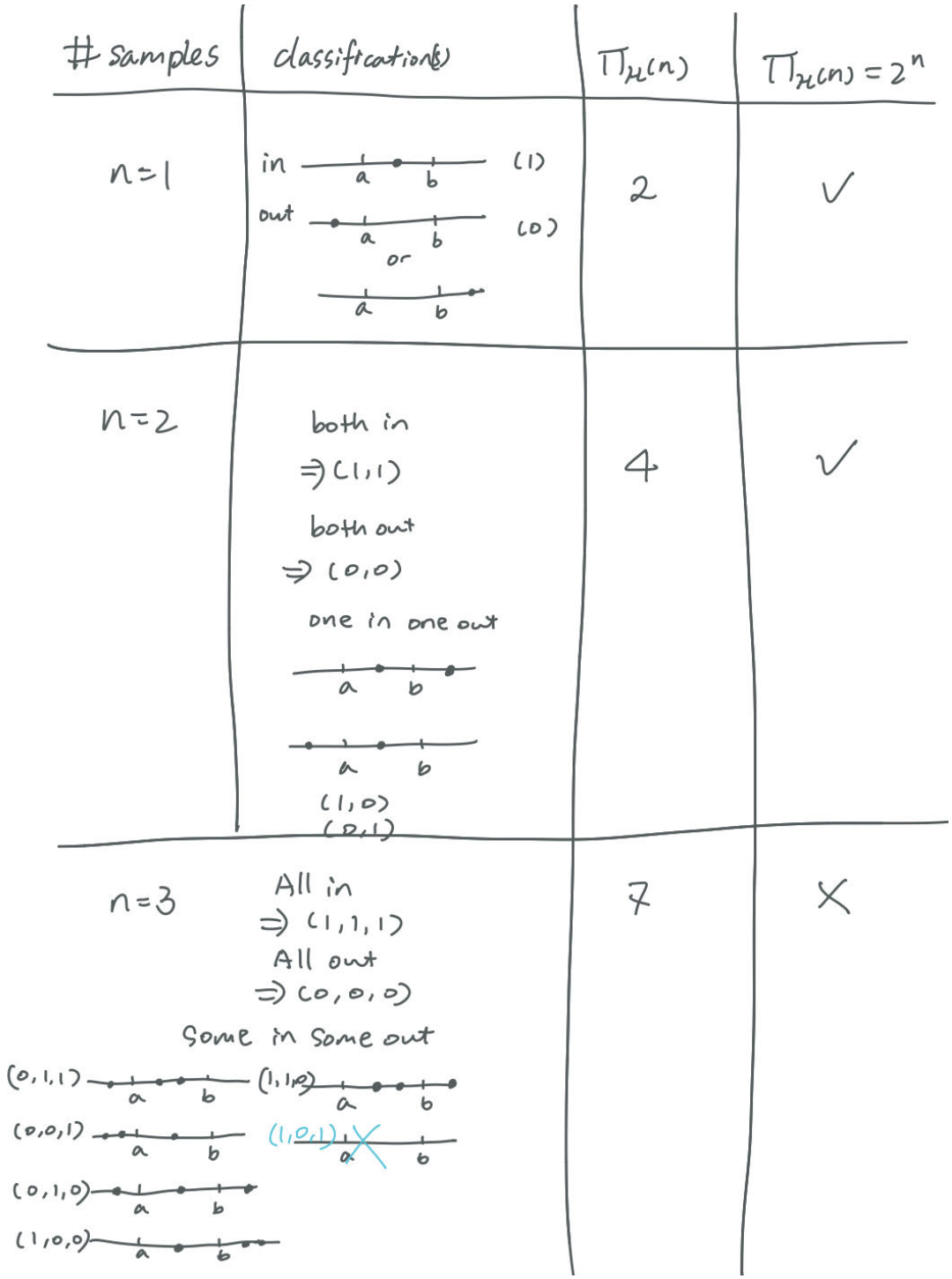


Figure 26: Finding VC dimension of an interval function class

2. When predefined hypothesis class is a linear function class in $\mathbb{R}^2$ (two features), i.e.
$$\mathcal{H} = \{(x_1, x_2) \to 1_{\{w_1 x_1 + w_2 x_2 + b \geq 0\}} : w_1, w_2, b \in \mathbb{R}\}$$
According to the following derivation, the largest size of samples that can be fully shattered by $\mathcal{H}$ is 3. Hence $VCdim(\mathcal{H}) = d_{VC} = 3$

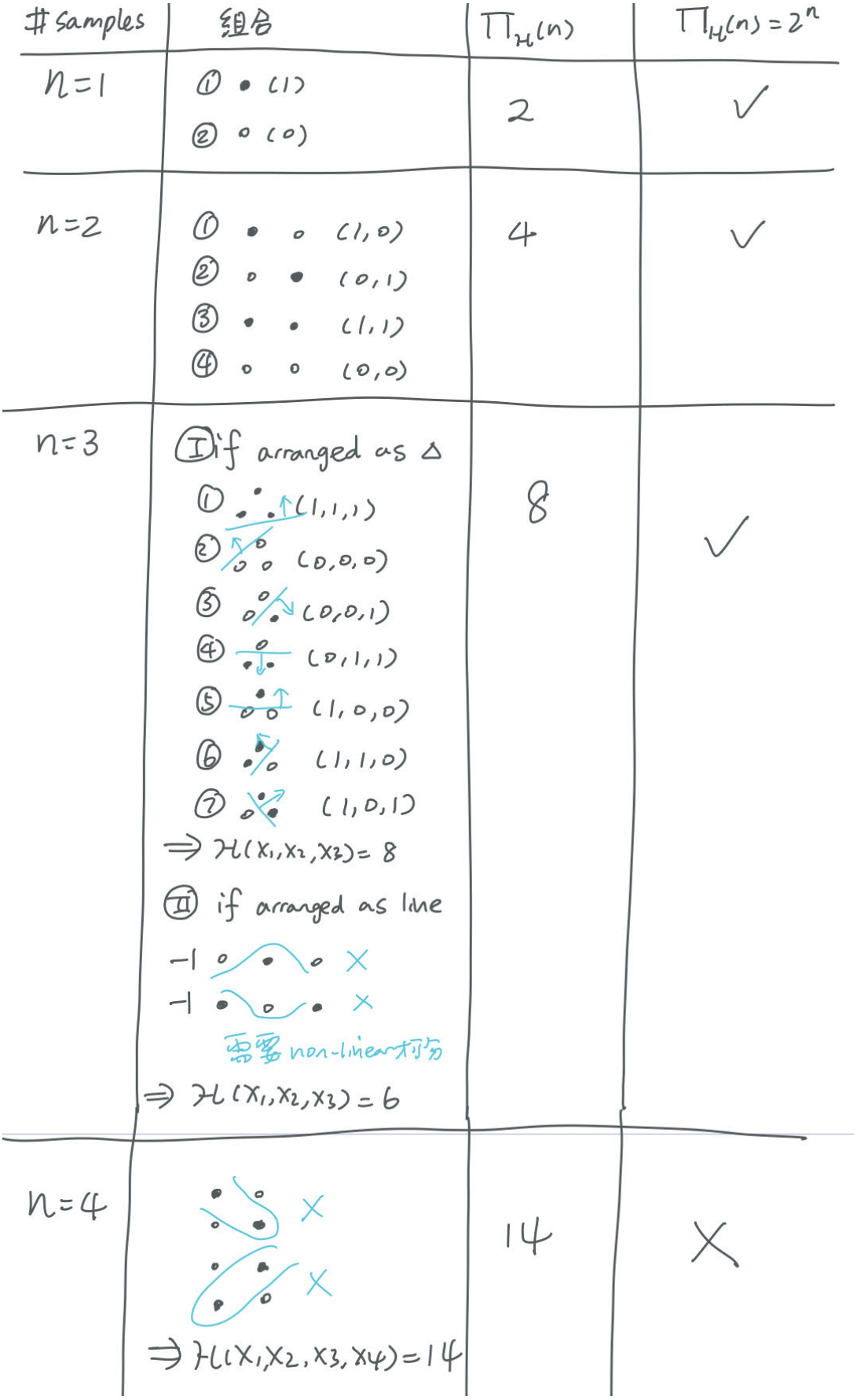


Figure 27: Finding VC dimension of a linear function class

Definition 3.16. Upperbound Growth Function using VC dimension d

For any $\mathcal{H}$, if we have its VC dimension d, then we can upperbound the growth function of $\mathcal{H}$ using d.

Claim Consider a $\mathcal{H}$ with $d_{VC}(\mathcal{H}) = d$, then $\forall n \geq d$ we have

$$\begin{aligned} &\text{Sauer's Lemma: } \Pi_{\mathcal{H}}(n) \leq \sum_{i=1}^{d} \binom{n}{i} \\ &\text{Corollary: } \Pi_{\mathcal{H}}(n) \leq (\frac{en}{d})^d \end{aligned} \tag{20}$$

Proof. of the Corollary

$$\begin{aligned} \Pi_{\mathcal{H}}(n) &\leq \sum_{i=0}^{d} \binom{n}{i} \text{... By Sauer's Lemma} \\ &\leq \sum_{i=0}^{d} \binom{n}{i} (\frac{n}{d})^{d-i} \text{... as } n \geq d \\ &\leq \sum_{i=0}^{n} \binom{n}{i} (\frac{n}{d})^{d-i} \text{... as } n \geq d \\ &\leq (\frac{n}{d})^d \sum_{i=0}^{n} \binom{n}{i} (\frac{d}{n})^i \\ &= (\frac{n}{d})^d (1 + \frac{d}{n})^n \text{... By Binomial Theorem} \\ &\leq (\frac{n}{d})^d e^d \text{... as } (1 + (-x)) < e^{-x} \text{ hence } (1 + (\frac{d}{n}))^n < (e^{\frac{d}{n}})^n = e^d \\ &= (\frac{en}{d})^d \end{aligned}$$

□

Definition 3.17. Check if $\mathcal{H}$ is PAC-learnable: $|\mathcal{H}| < \infty$

Theorem 3.18. If $|\mathcal{H}| < \infty$ then $\mathcal{H}$ is PAC-learnable.

Proof. From step 2 we have

$$P\Big(R(h_S) - R(h^*) \leq 2M\sqrt{\frac{\log|\mathcal{H}| + \log\frac{2}{\delta}}{2n}}\Big) \geq 1 - \delta$$

Where

- $\delta = 2|\mathcal{H}| \exp\Big(\frac{-2n\epsilon^2}{M^2}\Big)$
- $n = \frac{M^2}{2\epsilon^2} \log\frac{2|\mathcal{H}|}{\delta} = poly(\frac{1}{\delta}, \frac{1}{\epsilon})$ when $|\mathcal{H}| < \infty$

Though from the above inequality we know that h^* can be learned "probably approximately correctly", we still need to check if it can be learned "efficiently" through the sample complexity test. Since $|\mathcal{H}| < \infty$ (aka $|\mathcal{H}|$ is a constant) we can rewrite n as $poly(\frac{1}{\delta}, \frac{1}{\epsilon})$ format. This then satisfies the efficiency requirement (poly-many samples) hence $\mathcal{H}$ is PAC-learnable. □

Definition 3.19. Check if $\mathcal{H}$ is PAC-learnable: $VCdim(\mathcal{H}) = d < \infty$

Theorem 3.20. If $VCdim(\mathcal{H}) = d < \infty$, then $\mathcal{H}$ is PAC-learnable.

Proof. From "generalisation error upperbound" section we have

$$
\begin{aligned}
&P(\sup_{h\in\mathcal{H}}|R_S(h)-R(h)|\geq\epsilon)\\
&\leq 2P(\sup_{h\in\mathcal{H}}|R_S(h)-R_{S'}(h)|\geq\frac{\epsilon}{2})\\
&\leq 4P(\sup_{h\in\mathcal{H}}\frac{1}{n}\left|\sum_{i=1}^{n}\delta_i\ell(X_i,Y_i,h)\right|\geq\frac{\epsilon}{4}) \text{ ... By Glivenko-Cantelli Theorem}\\
&\leq 4P(\cup_{h\in\mathcal{H}'}\frac{1}{n}\left|\sum_{i=1}^{n}\delta_i\ell(X_i,Y_i,h)\right|\geq\frac{\epsilon}{4}) \text{ ... same idea as "implies"; H' the regrouped set}\\
&\leq 4\Pi_{\mathcal{H}}(n)\{\frac{1}{n}\left|\sum_{i=1}^{n}\delta_i\ell(X_i,Y_i,h)\right|\geq\frac{\epsilon}{4}\}\\
&\leq 8\Pi_{\mathcal{H}}(n)exp(\frac{-n\epsilon^2}{32M^2})\\
&\leq 8(\frac{en}{d})^d\exp\left(\frac{-n\epsilon^2}{32M^2}\right)
\end{aligned}
\tag{21}
$$

$$
\begin{aligned}
&\Leftrightarrow P(\sup_{h\in\mathcal{H}}|R_S(h)-R(h)|\geq\epsilon)\leq 8(\frac{en}{d})^d\exp\left(\frac{-n\epsilon^2}{32M^2}\right)\\
&\Leftrightarrow P(2\sup_{h\in\mathcal{H}}|R_S(h)-R(h)|\leq 2\epsilon)\geq 1-8(\frac{en}{d})^d\exp(\frac{-n\epsilon^2}{32M^2})\\
&\Leftrightarrow P(2\sup_{h\in\mathcal{H}}|R_S(h)-R(h)|\leq 2\epsilon)\\
&\Leftrightarrow P(R(h_S)-R(h^*)\leq 2\sup_{h\in\mathcal{H}}|R_S(h)-R(h)|\leq 2\epsilon)\geq 1-8(\frac{en}{d})^d\exp(\frac{-n\epsilon^2}{32M^2})\\
&\Leftrightarrow P(R(h_S)-R(h^*)\leq 2\epsilon)\geq 1-8(\frac{en}{d})^d\exp(\frac{-n\epsilon^2}{32M^2})
\end{aligned}
$$

where

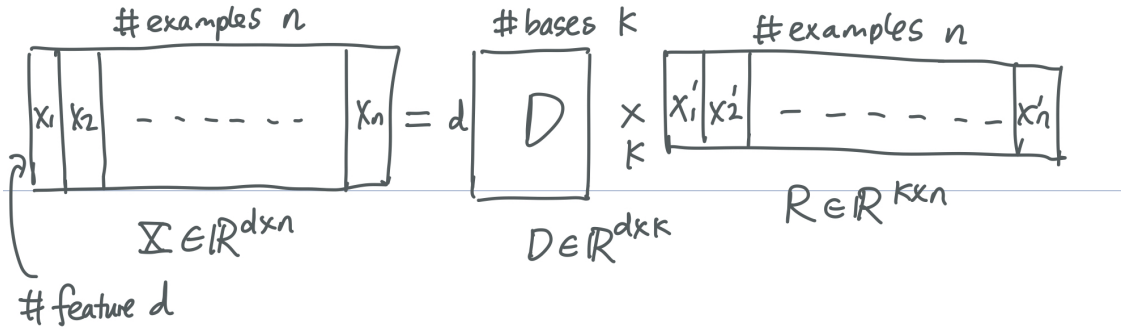
- $\delta=8(\frac{en}{d})^d\exp(\frac{-n\epsilon^2}{32M^2})$
- $\epsilon=M\sqrt{\frac{32(d\log\frac{en}{d}+\log\frac{2}{\delta})}{n}}$
- $n=\frac{32M^2}{\epsilon^2}(d\log\frac{en}{d}+\log\frac{8}{\delta})=poly(\frac{1}{\delta},\frac{1}{\epsilon})$ when $d<\infty$

Since $\mathcal{H}$ satisfies the "probably approximately correctly" through the inequality and also the poly-many sample size $n=poly(\frac{1}{\delta},\frac{1}{\epsilon})$ when $d<\infty$, $\mathcal{H}$ is PAC-learnable. □

4 Dictionary Learning and Non-negative Matrix Factorisation (NMF)

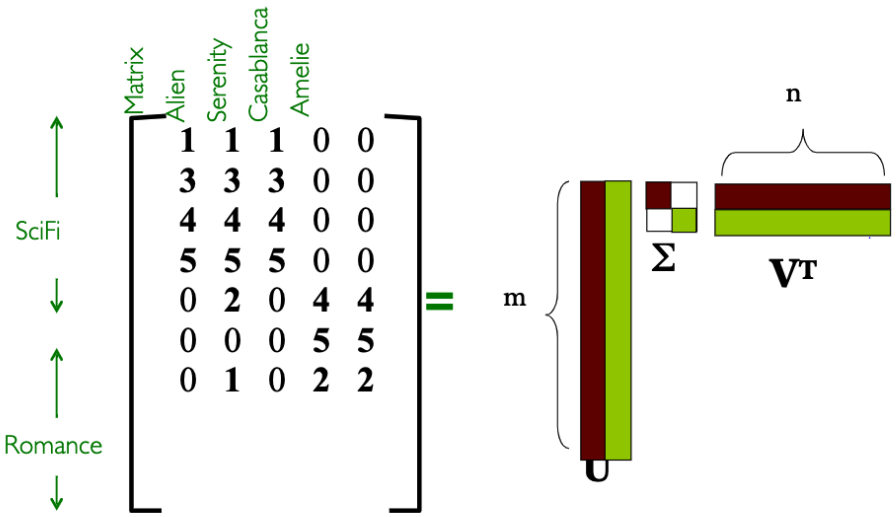
4.1 Dictionary Learning

1. Unsupervised learning algorithms
2. Dictionary
   - Definition: a matrix containing a collection of bases which is used to reconstruct the data matrix
3. Dictionary Learning



   - Goal: to learn D and R to reconstruct the original data matrix X with the least reconstruction errors.
   - Objective function:
     - $\min_{D,R} \|X - DR\|_F^2$
     - where D and R are specific to the domains/applications with different requirements. E.g. PCA, SVD and NMF have different constraints/requirements on D and R.
   - Optimisation method:
     - Multiplicative Update Rule (MUR)
     - Alternatively update D and R. For example, first fix D and optimise for R and then fix R and then optimise D. Stop until convergence.
     - Cons: only convex for either D or R but not both. Hence solution is locally optimal and can have many local solutions.
4. Dictionary Applications
   (a) Dimension reduction:
   - if k«d then we can transform X from a higher dimensional space (i.e. d dimensions) to a lower dimensional space (i.e. k dimensions)

   (b) SVD



   - Type: A type of dimension reduction method, a special case of dictionary learning

- Formula
$$X = U\Sigma V^T$$
where U=D and $\Sigma V^T = R$
- Special requirements (same as PCA)
  i. columns of U are normalised
  ii. columns of U are orthogonal
- Remarks: X is non-square (while in PCA X has to be square)

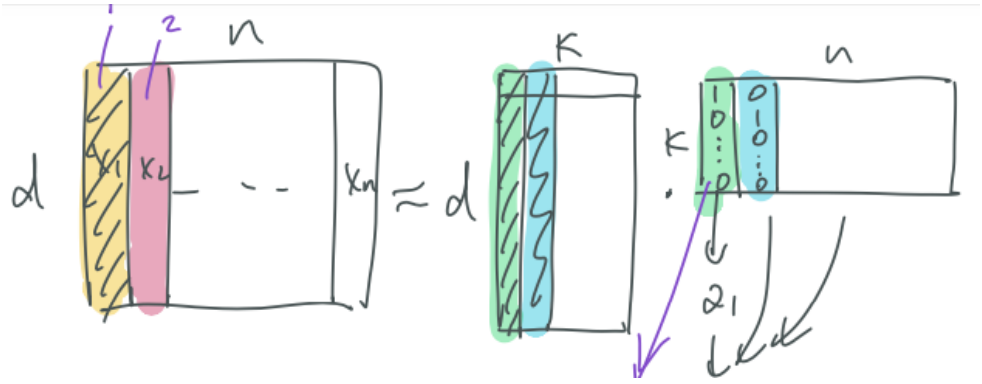
(c) PCA

$$A = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \begin{bmatrix} — & \mathbf{u}_1^T & — \\ — & \mathbf{u}_2^T & — \\ & \vdots & \\ — & \mathbf{u}_n^T & — \end{bmatrix}$$

D R

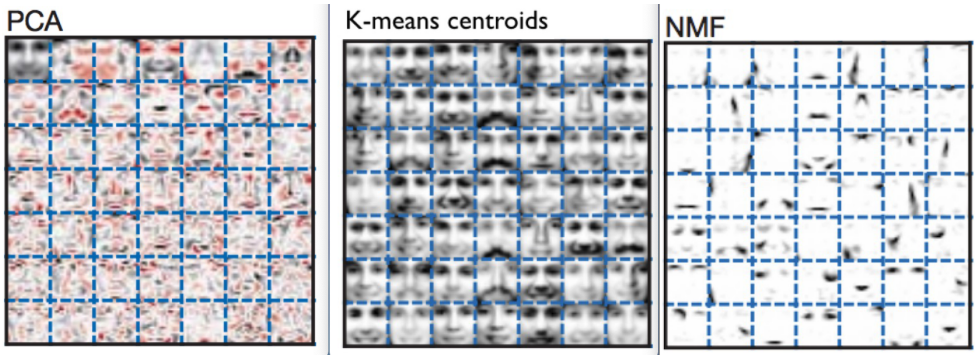
- Type: A type of dimension reduction method, a special case of dictionary learning
- Formula:
$$X = U\Lambda U^T$$
where $U = D$ and $\Lambda U^T = R$, where left and right matrix are the same
- Special requirements (same as SVD)
  i. columns of U are normalised
  ii. columns of U are orthogonal
- Remarks
  – X is square
  – Columns of D or U are eigenvectors

(d) K-Means Clustering



- Type: special case of dictionary learning
- Special requirements
  – All columns of R must be one-hot vector (i.e. sparse R)
- Remarks
  – Each column of D: centroid
  – Each column of R: denote which centroid to assign X's column (i.e. one sample)
- Example: if R_1 has the 1st entry as 1, all other 0s, then it means that X_1 should be assigned to the cluster D_1 centroid sits or X_1 should be represented by the D_1 centroid.

(e) Basis Visualisation of PCA and K-Means



- The dictionary D for both PCA and K-Means stores faces as the bases.
- In NMF, D stores parts of faces.

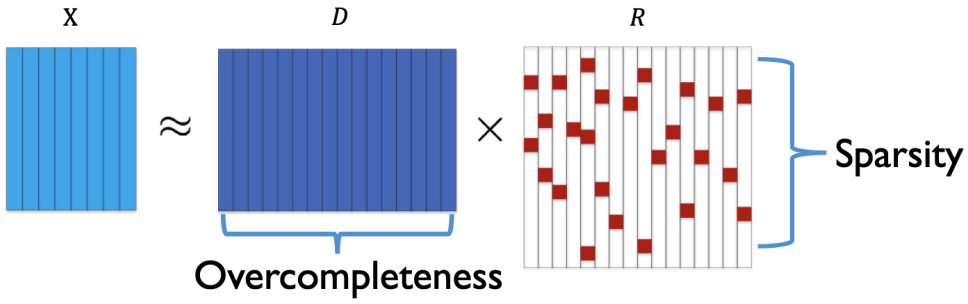
4.2 Non-Negative Matrix Factorisation

1. Type
   (a) unsupervised learning algorithm
   (b) special cases of dictionary learning
2. Special requirements
   - Both D and R are non-negative
3. Key feature: Non-negativity
   - Motivation:
     – many real world data are non-negative, such as image pixels, stock prices and ratings.
     – humans tend to perceive objects by parts
   - Pros
     (a) Allow part-based and additive-only representation of the data, which is semantic
     (b) Nice geometric interpretation. Non-negative D forms a cone and non-negative R restricts all points within the cone.
   - Cons
     (a) Factorisation of data matrix X is not unique and there can be many local minimum solutions. The reason is that the objective function is not convex for both D and R.
     (b) Many basis columns in D can be redundant
   - NMF variants
     (a) Norms. Use different norms such as ℓ_2, ℓ_1 etc to improve the NMF robustness.
     (b) Redundancy. Add sparsity regularisation to reduce the redundancy of D.
     (c) To improve generalisation capability of NMF through upperbound analysis.
   - Optimisation: Multiplicative Update Rule (MUR) for the l2 NMF
   - Applications
     (a) Image processing: remove sunglasses and scaves
     (b) Text mining and topic discovery
   - Others
     – Calculate F-norm: $\|X\|_F = \sqrt{\text{trace}(X^T X)}$

5 Sparse Coding and Regularisation

5.1 Sparse Coding

1. Motivation: sparsity is natural and many data (i.e. signals) are sparse in some transformed domains. If dictionary is well-defined, often time R is naturally sparse.
2. Sparsity Definition
   - Most elements of a vector are zeros
3. How to Ensure Sparsity?
   - Choose a large k
     - X: dxn, D: dxk, R: kxn
     - such that D is overcomplete and then R is sparse



4. Measure Sparsity
   - Objective Function

$$\min_{D\in\mathcal{D},R\in\mathcal{R}} \underbrace{\|X-DR\|_F^2}_{\text{Data fitting}} + \underbrace{\lambda\psi(R)}_{\text{Sparse regularisation}}$$

     ψ is the sparsity measure and $\lambda\psi(R)$ the sparsity regularisation term forces R to be sparse
   - Choice of *psi*: overall ℓ_0 and ℓ_1 are most popular.
     - ℓ_0 norm:
       * Informally denotes the count of non-zero elements in a vector. The lower the norm, the more sparse R is.
       * Pros: most effective in measuring sparsity
       * Cons: non-convex and non-smooth (i.e. non-differentiable), hard to optimise
     - ℓ_1 norm:
       * A surrogate for ℓ_0 norm
       * Pros:
         · Convex (compared to ℓ_0)
         · More sparse than ℓ_2 (constraint space: square for ℓ_1 while circle for ℓ_2. Using ℓ_1 norm can help hit the axis more hence more 0 and more sparse)
       * Cons:
         · Not as effective as ℓ_0 norm
         · Non-smooth
5. Two Popular Formulations of Sparse-coding Problem

(a) ℓ_0-based Approach

- $\min_{\alpha} \|X - D\alpha\|_F^2 \ s.t. \forall i, \|\alpha\|_0 < L.$
- $\min_{\alpha} \|\alpha\|_0 \ s.t. \|X - D\alpha\|_F^2 \leq \epsilon.$

where ℓ_0 norm of R_i (i.e. α) is upperbounded by a very small number L

(b) ℓ_1-based Approach

- $\min_{\alpha} \|\alpha\|_1 \ s.t. \|X - D\alpha\|_F^2 \leq \epsilon.$
- $\min_{\alpha} \|X - D\alpha\|_F^2 + \lambda\|\alpha\|_1.$

6. Sparse Coding Examples:
   - K-Means Clustering
     - Special requirements: R_i is one-hot vector (i.e. $\|R_i\|_0 = 1$ and $\|R_i\|_1 = 1$)
   - K-SVD
     - Type: A generalisation of K-means clustering, nothing to do with SVD
     - Special requirements: the number of non-zero elements are way less than k (i.e. $\|R_i\|_0 << k$)
     - Comparison to K-Means: R is less sparse but still sparse
7. Applications
   (a) Image Compression
   - Compress image X by storing D and R only
   - Good compression: small memory and small reconstruction error
   - Classic algorithms: K-SVD, PCA, JPEG2000
   - Preprocessing images before K-SVD
     - Align facial features
     - Uniform slicing

   (b) Image recovery
   - Recovery images with missing pixels
   - Reason: can learn missing pixels from images with similar pixels. These common information is captured in D.

   (c) Image Text Removal
   - Remove added texts from the images
   - Reason: other images don't have the same text so during reconstruction, the text information wont remain.

6 Feature Noise

6.1 Background: Bayesian Statistics

Definition 6.1. Bayesian Theorem

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{22}$$

where

- H is one of the competing hypotheses and we want to select one *most probable* hypothesis based on observed evidence E.
- $P(H)$ aka **Prior Probability or Prior**, is the *probability estimate* of hypothesis H before we observe any evidence or data. It's a guess based on previous related events or knowledge.
- $P(E)$ aka **Model Evidence or Marginal Likelihood**
- $P(E|H)$ aka **Likelihood**, is the probability of observing evidence E given that we have hypothesis H. For example, given we know that evidence/sample follows Poisson(3), we can calculate the probability of observing the current evidence/samples.
- $P(H|E)$ aka **Posterior Probability**, is the probability of a hypothesis after observing data/evidence E. It is a more rational and more updated belief as we observed evidence and gained more information. Under Bayesian Approach we should always value posterior probability more than prior probability and use it to guide our decision-making in choosing the best estimator.

Assume we have some sample data $S = ((x_1, y_1), ..., (x_n, y_n))$ that independently and identically follows some unknown distribution P_θ and θ is our hypothesis here. Our goal is to find the θ that make the event of observing current samples S most probable.

Definition 6.2. Estimate unknown θ

1. **Maximum Likelihood Estimation (MLE)**:
   - Goal: find θ that maximises the (log)likelihood function $P(S|\theta)$
   - Max likelihood = Min -likelihood = Min $R_S(h)$
   - It is some θ that makes the current sample data most probable.
   - a frequentist's approach
   - Assumption: data are iid
2. **Maximum A Posteriori Probability Estimation (MAP)**:
   - Goal: use the *mode/maximum* of the posterior probability $P(\theta|S)$ as the point estimator.
   - MAP = Min $R_S(h) + \lambda\|w\|_2^2$ (regularised empirical risk)
   - It is the current sample data that makes some θ most probable.
   - More of a Bayesian approach.
3. **Posterior Mean**
   - use the expected value of posterior probability distribution $P(\theta|S)$ as the point estimate of θ.

$$\begin{aligned}\hat{\theta}_{\text{MLE}}(S) &= \arg\max_\theta P(S|\theta) \\ &= \arg\max_\theta \Pi_{i=1}^n p_\theta(x_i, y_i)\end{aligned}$$

$$\begin{aligned}\hat{\theta}_{\text{MAP}}(S) &= \arg\max_\theta P(\theta|S) \\ &= \arg\max_\theta \frac{P(S|\theta)P(\theta)}{P(S)} \quad \text{.... By Bayes Theorem} \\ &= \arg\max_\theta P(S|\theta)P(\theta) \quad \text{.... P(S) independent of } \theta\end{aligned}$$

$$\hat{\theta}_{\text{PM}}(S) = \mathbb{E}[\theta|S]$$

Definition 6.3. Bias, Variance and Bias-Variance Trade-off

Assume we have some linear regression classifier $\hat{f}$ and we want to apply it to the test data and evaluate the performance based on MSE (i.e. choose square error as the loss function). Also assume the test data and training data follow the same distribution where

$$y_i = f(x_i) + \epsilon_i$$

where noise $\epsilon_i \sim N(0, \sigma^2)$. Then the Mean-Square Error (MSE) over the test data (x, y) is

$$\begin{aligned}
\text{MSE}_{\text{test}} &= \mathbb{E}[(y - \hat{f}(x))^2] \\
&= \mathbb{E}[(f(x) + \epsilon - \hat{f}(x))^2] \\
&= \mathbb{E}[(f(x) - \hat{f}(x))^2] + \mathbb{E}[\epsilon^2] - 0 \\
&= \mathbb{E}^2[f(x) - \hat{f}(x)]] + \text{Var}(f(x) - \hat{f}(x)) + \sigma^2 \text{.... By Variance's definition} \\
&= \text{Bias}^2(f(x), \hat{f}(x)) + \text{Var}(\hat{f}(x)) + \sigma^2 \text{.... as } f(x) \text{ is fixed}
\end{aligned} \tag{23}$$

Bias of the estimator $\hat{f}(x)$ is defined as $\mathbb{E}[f(x) - \hat{f}(x)]]$ and it evaluates how well on average the predictor $\hat{f}(x)$ predicts $f(x)$. Usually if $\hat{f}(x)$ didn't capture the relationship between x and y in training data, it would lead to **high bias** in the test stage and we say the predictor $\hat{f}(x)$ is **underfitting**.

Variance of the estimator $\hat{f}(x)$ is evaluated based on how fluctuating the estimator is. For a predictor that is **overfitting** the data, usually there is **high variance** since the predictor is over-"fit" on the specific training set and doesn't generalise well on new examples in test data.

Usually there's a trade-off between bias and variance. When we reduce one, we increase the other. Since we want to minimise the loss (here MSE_{test}) as much as possible in equation 23 and we assume δ^2 is fixed, we should keep a good balance between those two terms. That is, reduce the bias without increasing too much of the variance, and reduce the variance without increasing too much of the bias.

Remarks. Bias-Variance Trade-off

- λ is the regularisation parameter. Larger $\lambda \equiv$ more regularisation $\equiv$ more underfitting; smaller $\lambda \equiv$ less regularisation $\equiv$ more overfitting
- The gap between black and pink lines is fixed as it represents σ^2
- The opposite trends of blue and red lines indicate the trade-off between bias and variance

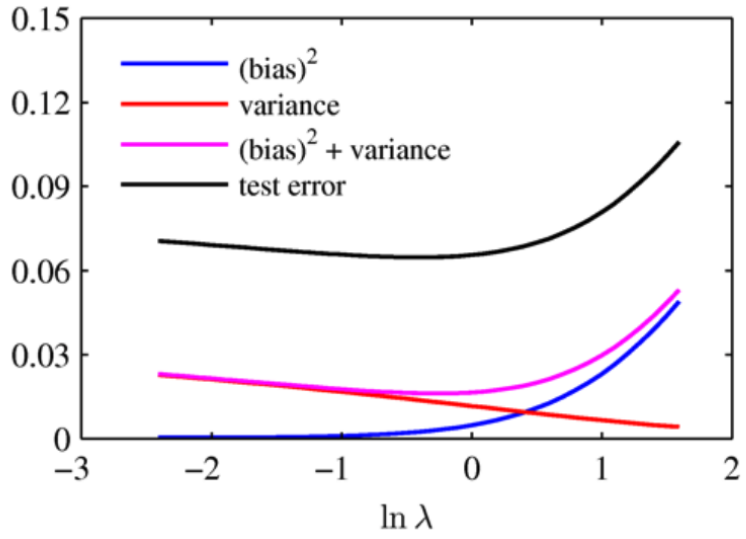


Figure 28: Bias-Variance Trade-off

Ways to avoid overfitting

1. Reduce $\mathcal{H}$ complexity. For example, use less complex model.
2. Increase sample data. The reason is by LLN with more sample data, the empirical risk will be closer to the expected risk (i.e. training error will be closer to test error hence less overfitting)

6.2 Robustness of Surrogate Loss Functions

1. Derive surrogate loss
   (a) Step 1: assume noise distribution
   (b) Step 2: calculate negative log likelihood of $P(S|\theta)$
   (c) Step 3: find loss representation in the equation
2. Example:
   - When noise follows Gaussian distribution, we have least square loss
   - When noise follows Laplacian distribution, we have absolute loss
   - When noise follows Cauchy distribution, we have Cauchy loss
3. Noise distribution
   - Cauchy > Laplacian > Gaussian in modelling large noise
   - Reason: when noise is large (x), p(x) is largest under Cauchy while under Gaussian it almost disappears. This indicates Cauchy is capable of modelling noise.

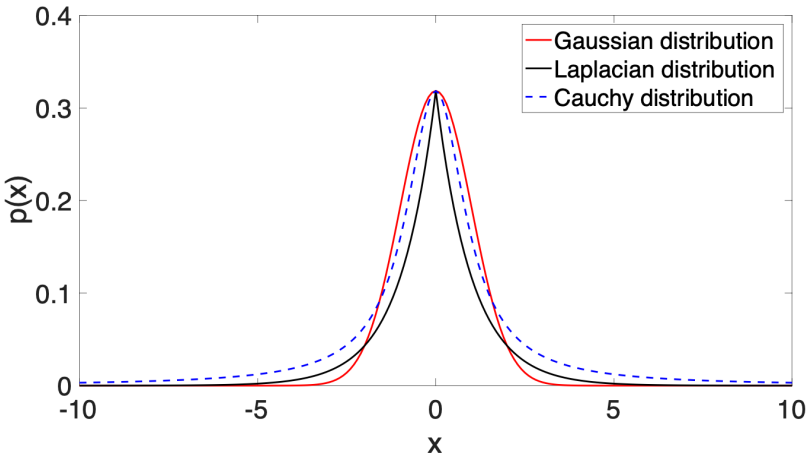


Figure 29: Noise Distributions

4. Compare Robustness of Surrogate Loss
   (a) Goal: find some optimal h such that the empirical risk using the surrogate loss can be minimised
   $$R_S(h) = f(h) = \frac{1}{n}\sum \ell(x_i, y_i, h)$$
   where ℓ is the surrogate loss, could be square loss, absolute loss, Cauchy loss or Correntropy loss.
   (b) Old way: through $\nabla f(h)$
   - if dom f is bounded: h is optimal $\Leftrightarrow \nabla f(h)^T(h' - h) \geq 0$
   - if dom f is not bounded: h is optimal $\Leftrightarrow \nabla f(h) = 0$ (vector)

   (c) New way: through $g(t) = f(h * t)$
   - Motivation: previous $\nabla f(h)$ is a vector and it is hard to compare.
   - Key: construct $g(t) = f(h * t)$ and $g'(t) = \nabla f(h * t)^T h$ where $g(t)$ and $g'(t)$ are both scalars
     – h is optimal $\Leftrightarrow g'(1) = 0$ (equivalent to $\nabla f(h) = 0$)
   - Pros: much easier to check optimality of h through comparing scalars than vectors
   - Examples: compare $g'(1)$ of different surrogate loss

- **Least squares loss:** $g'(1) = \frac{1}{n}\sum_{i=1}^{n} 2(y_i - h(x_i))(-h(x_i))$
- **Absolute loss:** $g'(1) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{|y_i - h(x_i)|}(y_i - h(x_i))(-h(x_i))$
- **Cauchy loss:**

$$g'(1) = \frac{1}{n}\sum_{i=1}^{n} \frac{2}{\gamma^2 + (y_i - h(x_i))^2}(y_i - h(x_i))(-h(x_i))$$

- **Correntropy loss (Welsch loss):**

$$g'(1) = \frac{1}{n}\sum_{i=1}^{n} \frac{2}{\sigma^2 \exp\left(\frac{y_i - h(x_i)}{\sigma}\right)^2}(y_i - h(x_i))(-h(x_i))$$

– Notations
  * Each term within the sum: contribution to optimising the empirical risks
  * Blue boxes: weight
  * Red boxes: c_i, the bases of contribution
– Criterion: a surrogate loss is more robust if it assigns smaller weight to the base as the noise goes larger
– Robustness Rank: Correntropy > Cauchy > Absolute > Square
– Reason:
  * More robust loss tends to assign smaller weight for larger noise $|y_i - h(x_i)|$. This helps decrease the contribution of large noise term.
  * Square loss is least robust as it has constant weight regardless of noise scale.
  * Corretropy loss is the most robust as it decays the weight exponentially with the size of noise.
– If we don't know the noise distribution of data, we should try all surrogate loss and see which one is the most robust.

7 Domain Adaptation and Transfer Learning

7.1 Context

1. Domain: means distribution or joint distribution in ML, different from the maths definition
2. Source domain:
   - distribution $P_s(X, Y)$
   - often with sufficient data (i.e. features and labels)
3. Target domain:
   - distribution $P_t(X, Y)$
   - often with insufficient data or unavailable labels
     - insufficient data (small n): empirical risk does not estimate expected risk well, h_S^T will be poorly learned
     - unavailable labels: can't even calculate empirical risk, hence can't even learn h_S^T
4. Difference between Transfer Learning and Domain Adaption
   (a) Domain Adaptation:
   - More theoretical
   - Goal: reduce the distribution difference between source domain data and target domain data through distribution modification

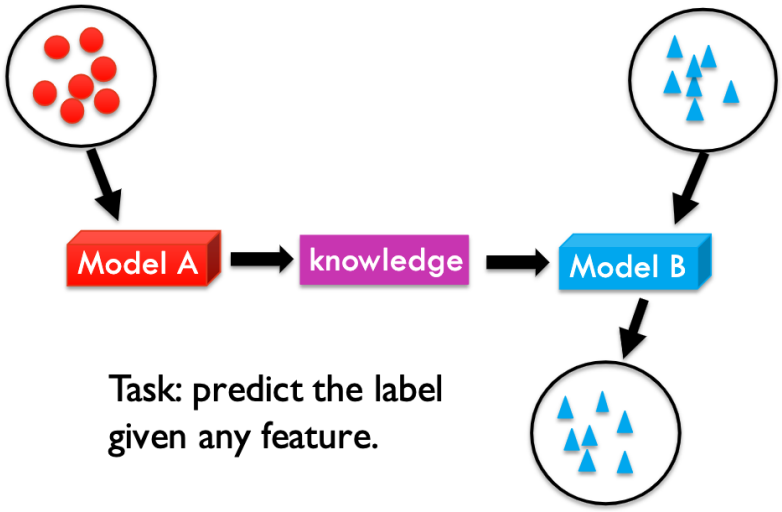
   (b) Transfer Learning
   - More empirical
   - Goal: improve classifier's performance in target domain by making use of the source domain knowledge

7.2 Domain Adaptation

1. Motivation: $P_t(X, Y) \neq P_s(X, Y)$, we can't directly apply model trained over source domain to the target domain.
2. Solution: modify the source domain distribution such that the modified distribution is similar or same as the target domain distribution
3. Focus: make source and target distributions the same
4. When to Adapt/Modify?
   - If $P_t(X, Y) = P_s(X, Y)$, then directly adapt the classifier trained over source domain to the target domain
   - Otherwise, need to first reduce the distribution difference between $P_t(X, Y)$ and $P_s(X, Y)$ by modifying the source domain distribution.
5. Distribution Similarity Measure
   - Kernel Mean Matching
   - μ operator: bijective function. If $\mu(\beta(X)P_s(X)) = P_t(X)$ then distribution $\beta(X)P_s(X)$ is the same as $P_t(X)$

7.3 Transfer Learning

1. Definition: extract knowledge from the source domain to improve the classifier performance in the target domain
2. Motivation:
   - Insufficient data in the target domain (i.e. labels, features or both) while abundant data in the related source domain. For example, self-driving domain and simulated driving domain.
   - In real world, can't guarantee sufficient data for all tasks and such insufficient cases happen very often.
3. Focus: improve target domain classifier's performance
4. Feature: Train on source domain, test on target domain



5. Different TL Methods
   - Objective perspective: modify objective functions like KMM
   - Hypothesis perspective: apply h_S to target domain and fine tune it
   - 2 Examples: Deep Adaption Network (hypothesis); Deep Correlation Alignment (loss correlated)
6. Method: Importance Reweighting (use $\beta(X, Y)$)
   - Motivation:
     - For target domain data, we want expected risk $R^T(h)$ but distribution $P_t(X, Y)$ is unknown
     - Instead, try use empirical risk but y_i^T are unavailable
   - Solution: **Assume the weight $\beta(X, Y)$ is known**. Then we can approximate the expected risk in target domain using source domain data.

$$R^T(h) = E_{(X,Y)\sim P_t(X,Y)}[\ell(X, Y, h)]$$

$$= \int \ell(x, y, h) P_t(x, y)\, dx dy$$

$$= \int \ell(x, y, h) \frac{P_t(x, y)}{P_s(x, y)} P_s(x, y)\, dx dy$$

$$= \int \ell(x, y, h)\, \beta(x, y)\, P_s(x, y)\, dx dy \quad \text{where } \beta(x, y) = \frac{P_t(x, y)}{P_s(x, y)}$$

$$= E_{(X,Y)\sim P_S(X,Y)}[\beta(X, Y)\, \ell(X, Y, h)]$$

Assume $\beta(X, Y)$ is known, since $P_S(X, Y)$ is also unknown, we estimate the expected risk with empirical risk

$$\Rightarrow E_{(X,Y)\sim P_S(X,Y)}[\beta(X, Y)\, \ell(X, Y, h)]$$

$$\approx \frac{1}{n_S} \sum_{i=1}^{n_S} \beta(x_i^S, y_i^S)\, \ell(x_i^S, y_i^S, h) \quad \text{with sufficient } n_S$$

   - Pros: enable us to learn best hypothesis h^* in the target domain even without sufficient data or labels
7. Learn $\beta(X, Y)$
   (a) Motivation: importance reweighting only works when beta is given. We need to learn beta such that we can estimate $R^T(h)$ through importance reweighting.

(b) Context: $\beta(X,Y)$ defines the distribution change between target and source domains, which can be split into two cases, depending on it's covariate shift on target shift. In covariate shift model, $\beta(X,Y) = \beta(X)$ while in target shift model, $\beta(X,Y) = \beta(Y)$.

$$P_t(X,Y) \neq P_s(X,Y)$$
$$P_t(X|Y)P_t(Y) \neq P_s(Y|X)P_s(X)$$

(c) Kernel Mean Matching (learn $\beta(X)$ or $\beta(Y)$)

i. $\mu()$ operator:

$$\mu(P_s(X)) = \mathbb{E}_{X\sim P_s(X)}[\phi(X)]$$

where

- inside must be distributions, such as $P_s(X)$ or $\beta(X)P_s(X)$
- only bijective when K() is a univeral kernel

ii. Same distribution $P_t(X) = \beta(X)P_s(X)$.
If we have

A. $\mu(P_t(X)) = \mu(\beta(X)P_s(X)) = \mathbb{E}_{X\sim P_s(X)}[\beta(X)\phi(X)]$

B. $\beta(X) \geq 0$

C. $\mathbb{E}_{X\sim P_s(X)}[\beta(X)] = 1$

Then $P_t(X) = \beta(X)P_s(X)$

iii. Proof

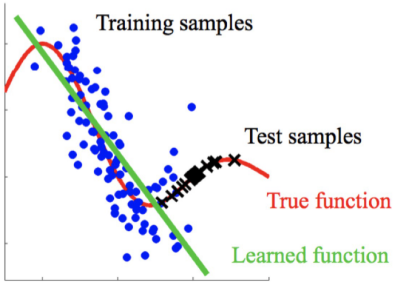
① $\mu(P_t(X)) = \mathbb{E}_{X\sim P_t(X)}[\phi(X)]$ by definition

$$= \int \phi(x) P_t(x)\, dx$$
$$= \int \phi(x) \frac{P_t(x)}{P_s(x)} P_s(x)\, dx$$
$$= \int \phi(x) \beta(x) P_s(x)\, dx$$
$$= \mathbb{E}_{X\sim P_s(X)}[\beta(X)\phi(X)]$$
$$= \mu(\beta(X)P_s(X))$$

② $\beta(X) = \frac{P_t(X)}{P_s(X)} \geq 0$

③ $\mathbb{E}_{X\sim P_s(X)}[\beta(X)] = \int \beta(x) P_s(x)\, dx$
$$= \int \frac{P_t(x)}{P_s(x)} P_s(x)\, dx$$
$$= 1$$

(d) **Approach 1: Covariate Shift Model**



- Assumption: $P_s(Y|X) = P_t(Y|X)$ but $P_s(X) \neq P_t(X)$
- $\beta(X,Y) = \beta(X)$ under such assumption
- Learn $\beta(X)$ with Kernel Mean Matching

objective

$$\min_{\beta} \| \mu(P_t(X)) - \mu(\beta(X) P_s(X)) \|^2$$

$$= \min \| E_{X \sim P_t(X)}[\phi(X)] - E_{X \sim P_s(X)}[\beta(X)\phi(X)] \|^2$$

s.t. $\beta(X) \geq 0$

$E_{X \sim P_s(X)}[\beta(X)] = 1$

only have samples $\{x_1^S, x_2^S, \ldots x_{n_S}^S\}$

$\{x_1^T, x_2^T, \ldots x_{n_T}^T\}$

unknown $P_t(X), P_s(X)$

↓

Empirically

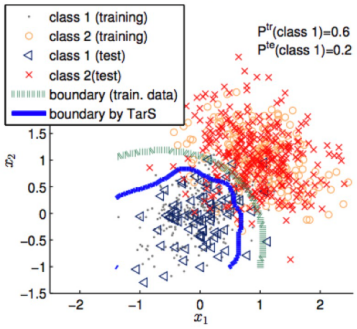
obj

$$\min_{\beta} \| \frac{1}{n_T} \sum_{i=1}^{n_T} \phi(x_i^T) - \frac{1}{n_S} \sum_{i=1}^{n_S} \beta(x_i^S) \phi(x_i^S) \|^2$$

s.t. $\beta(x_i^S) \geq 0$

$\frac{1}{n_S} \sum_{i=1}^{n_S} \beta(x_i^S) = 1$

(e) **Approach 2: Target Shift Model**



- Assumption: $P_s(X|Y) = P_t(X|Y)$ but $P_s(Y) \neq P_t(Y)$
- $\beta(X, Y) = \beta(Y)$ under such assumption
- Still learn $\beta(Y)$ with Kernel Mean Matching but with a different objective function
  - Issue: unlike covariate shift's case where x_i^T is available, here y_i^T is unavailable. So we can't use the similar objective function as covariate shift and kernel mean match $P_t(Y)$ and $\beta(Y)P_s(Y)$ to optimise for β.
  - Solution: find other two distribution with one contain $\beta(Y)$ and then kernel mean match them to optimise for beta.
  - Derivation

$$
\begin{aligned}
P_t(X) &= \int P_t(X,Y)\,dy \\
&= \int P_t(X|Y)\,\underbrace{P_t(Y)}_{\text{unavailable}}\,dy \\
&= \int P_t(X|Y)\,\underbrace{P_s(Y)\,\beta(Y)}_{\text{available}}\,dy \quad \text{by } \beta(Y) = \frac{P_t(Y)}{P_s(Y)} \\
&= \int P_s(X|Y)\,P_s(Y)\,\beta(Y)\,dy \quad \text{by Target shift assumption} \\
&= E_{Y\sim P_s(Y)}[\,\beta(Y)\,P_s(X|Y)]
\end{aligned}
$$

Then we match $P_t(X)$ & $E_{Y\sim P_s(Y)}[\,\beta(Y)\,P_s(X|Y)]$

use kMM

$$
\Rightarrow
$$

$$
\min_{\beta} \left\| \mu(P_t(X)) - \mu[E_{Y\sim P_s(Y)}[\,\beta(Y)\,P_s(X|Y)]] \right\|^2
$$

$$
= \min_{\beta} \left\| \mu(P_t(X) - E_{Y\sim P_s(Y)}[\,\mu(\beta(Y)\,P_s(X|Y))] \right\|^2
$$

Hence

$$
\text{obj} \quad \min_{\beta} \left\| \mu(P_t(X)) - E_{Y\sim P_s(Y)}[\,\mu(P_s(X|Y))\,\beta(Y)] \right\|^2
$$

$$
\text{s.t} \quad \beta(Y) \geq 0
$$

$$
E_{Y\sim P_s(Y)}[\,\beta(Y)] = 1
$$

Empirically,

$$
\text{obj} \quad \min \left\| \frac{1}{n_T}\sum_{i=1}^{n_T} \phi(x_i^T) - \frac{1}{n_S}\sum_{i=1}^{n_S} \beta(y_i^S)\,\hat{\mu}(P_s(x_i^S|y_i^S)) \right\|^2
$$

$$
\text{s.t} \quad \beta(y_i^S) \geq 0
$$

$$
\frac{1}{n_S}\sum_{i=1}^{n_S} \beta(y_i^S) = 1
$$

8 Label Noise

Definition 8.1. Noise Model

General Model

$$\rho_Y(X) = P(\tilde{Y} \mid Y, X)$$

flip rate

$$\begin{cases} \rho_{+1}(X) = P(\tilde{Y} = -1 \mid Y = +1, X = X) \\ \rho_{-1}(X) = P(\tilde{Y} = +1 \mid Y = -1, X = X) \end{cases}$$

RCN Model

$$\rho_Y(X) = P(\tilde{Y} \mid Y, X) = P(\tilde{Y} \mid Y)$$

$$\rho_{+1}(X) = \rho_{-1}(X) = \rho$$

CCN Model

$$\rho_Y(X) = P(\tilde{Y} \mid Y)$$

$$\rho_{+1}(X) = \rho_{+1}$$

$$\rho_{-1}(X) = \rho_{-1}$$

ILN Model

$$\rho_Y(X) = P(\tilde{Y} \mid Y, X)$$

$\rho_{+1}(X)$

$\rho_{-1}(X)$

Definition 8.2. General Noise: Relationship between Noisy and Clean Class Posterior

General noise's case

$$\begin{aligned} &P(\tilde{Y} = 1 \mid X = X) \\ &= P(\tilde{Y} = 1, Y = 1 \mid X = X) + P(\tilde{Y} = 1, Y = -1 \mid X = X) \\ &= P(\tilde{Y} = 1 \mid Y = 1, X = X) P(Y = 1 \mid X = X) + P(\tilde{Y} = 1 \mid Y = -1, X = X) P(Y = -1 \mid X = X) \\ &= [1 - \rho_{+1}(X)] P(Y = 1 \mid X = X) + \rho_{-1}(X) [1 - P(Y = 1 \mid X = X)] \\ &= [1 - \rho_{+1}(X) - \rho_{-1}(X)] P(Y = 1 \mid X = X) + \rho_{-1}(X) \end{aligned}$$

Similarly

$$P(\tilde{Y} = -1 \mid X = X) = [1 - \rho_{+1}(X) - \rho_{-1}(X)] P(Y = -1 \mid X = X) + \rho_{+1}(X)$$

Relationship: noisy & clean class posterior

$$\Rightarrow \begin{cases} P(\tilde{Y} = 1 \mid X = X) = [1 - \rho_{+1}(X) - \rho_{-1}(X)] P(Y = 1 \mid X = X) + \rho_{-1}(X) \\ P(\tilde{Y} = -1 \mid X = X) = [1 - \rho_{+1}(X) - \rho_{-1}(X)] P(Y = -1 \mid X = X) + \rho_{+1}(X) \end{cases}$$

Definition 8.3. Random Classification Noise Model (RCN)

1. Issue: negative effect of RCN
   - If we perform classification with convex potential loss over linear function class, then the result is equal to random guessing.
2. How to Deal with RCN?
   (a) use symmetric loss instead of convex potential loss
   (b) use universal function class instead of linear function class
   (c) ensure that noisy training samples are sufficiently large

   then the classifier will be robust to RCN and expected loss over noisy and clean data space are the same.

3. Proof

Claim:

if use ① symmetric loss L
② universal function class
③ sufficient large n

then

$$R_{D_\rho,L}(f) = (1-2\rho) R_{D,L}(f) + \rho C$$

$$\operatorname*{argmin}_{f\in U} R_{D_\rho,L}(f) = \operatorname*{argmin}_{f\in U} R_{D,L}(f)$$

proof

$$R_{D_\rho,L}(f) = E_{(X,\tilde{Y})\sim D_\rho}[\ell(f(X),\tilde{Y}]$$

$$= \int_{(X,\tilde{Y})} \left[\ell(f(x),1)\, P(X=x,\tilde{Y}=1) + \ell(f(x),-1) P(X=x,\tilde{Y}=-1)\right] dx$$

$$= \int \left[\ell(f(x),1)\cdot P(\tilde{Y}=1|X=x)\, P(X=x) + \ell(f(x),-1)\, P(\tilde{Y}=-1|X=x) P(X=x)\right] dx$$

$$= \int \ell(f(x),1)\left[(1-2\rho) P(Y=1|X=x)+\rho\right] P(X=x)dx$$

$$+ \int \ell(f(x),-1)\left[(1-2\rho) P(Y=-1|X=x)+\rho\right] P(X=x)\,dx$$

$$= (1-2\rho)\int \left[\ell(f(x),1)\cdot P(Y=1|X=x) + \ell(f(x),-1)\cdot P(Y=-1|X=x)\right] P(X=x)\,dx$$

$$+ \int \rho \underbrace{\left[\ell(f(x),1)+\ell(f(x),-1)\right]}_{C} P(X=x)\,dx$$

$$= (1-2\rho)\int \left[\ell(f(x),1)\, P(Y=1,X=x) + \ell(f(x),-1) P(Y=-1,X=x)\right] dx + \rho C$$

$$= (1-2\rho)\cdot E_{(X,Y)\sim D}[\ell(f(X),Y)] + \rho C$$

$$= (1-2\rho)\cdot R_{D,L}(f) + \rho C$$

Hence $R_{D_\rho,L}(f) = (1-2\rho) R_{D,L}(f) + \rho C$

Hence $\operatorname*{argmin}_{f\in U} R_{D_\rho,L}(f) = \operatorname*{argmin}_{f\in U} R_{D,L}(f)$

it is robust

4. Remarks:

- Convex potential loss ℓ
  - Convex, non-decreasing, differentiable at 0 with $\phi'(0) < 0$ and $\phi(\infty) = 0$
- Symmetric loss: $\ell(f(x),-1) + \ell(f(x),+1) = C$

Definition 8.4. Class Conditional Noise Model (CCN)

1. How to Deal with CCN?
   (a) Assume beta is given, use Importance Reweighting to modify the loss such that the expected risk using original loss over clean data is same as the expected risk using modified loss over noisy data
   (b) Proof

Technique:

modify L as $\tilde{L}(\beta L)$

Claim:

$$\operatorname*{argmin}_{f \in \mathcal{F}} R_{D_\rho, \tilde{L}}(f) = \operatorname*{argmin}_{f \in \mathcal{F}} R_{D, L}(f)$$

Proof

$$\begin{aligned}
R_{D,L}(f) &= \mathbb{E}_{(X,Y)\sim D}\left[L(f(X), Y)\right] \\
&= \int L(f(x), y)\, P_D(x, y)\, dx\, dy \\
&= \int L(f(x), y)\, \frac{P_D(x, y)}{P_{D_\rho}(x, y)}\, P_{D_\rho}(x, y)\, dx\, dy \\
&= \int L(f(x), y)\, \beta(x, y)\, P_{D_\rho}(x, y)\, dx\, dy \\
&= \mathbb{E}_{(X,Y)\sim D_\rho(X,Y)}\left[\beta(X, Y) L(f(X), Y)\right] \\
&= \mathbb{E}_{(X,Y)\sim D_\rho(X,Y)}\left[\tilde{L}(f(X), Y)\right] \\
&= R_{D,\tilde{L}}(f)
\end{aligned}$$

Hence $\operatorname*{argmin}_{f \in \mathcal{F}} R_{D,\tilde{L}}(f) = \operatorname*{argmin}_{f \in \mathcal{F}} R_{D,L}(f)$

2. Learn $\beta(X, Y)$, which depends on
   (a) Noisy class posterior (learn from noisy data)
   (b) Two flip rates ρ_{+1} and ρ_{-1} (aka transition matrix, both binary and multiclass case learn from anchor points)
3. Proof

$$\beta(x,y) = \frac{P_D(x,y)}{P_{D_\rho}(x,y)}$$

$$= \frac{P_D(y|x)\,P_D(x)}{P_{D_\rho}(y|x)\,P_{D_\rho}(x)} \quad \text{as features are clean}$$

Recall General noise

$$\begin{cases} P(\tilde{Y}=1|X=x) = [1-\rho_{+1}(x)-\rho_{-1}(x)]\,P(Y=1|X=x) + \rho_{-1}(x) \\ P(\tilde{Y}=-1|X=x) = [1-\rho_{+1}(x)-\rho_{-1}(x)]\,P(Y=-1|X=x) + \rho_{+1}(x) \end{cases}$$

CCN:

$$P_{D_\rho}(\tilde{Y}=y|X=x) = (1-\rho_{+1}-\rho_{-1})P(Y=y|X=x) + \rho_{-y}(x)$$

Sub in

$$\beta(x,y) = \frac{P_D(Y=y|X=x)}{P_{D_\rho}(\tilde{Y}=y|X=x)}$$

$$= \frac{P_{D_\rho}(\tilde{Y}=y|X=x) - \rho_{-y}}{(1-\rho_{+1}-\rho_{-1})\,P_{D_\rho}(\tilde{Y}=y|X=x)}$$

$$\beta(x,y) = \frac{P_{D_\rho}(\tilde{Y}=y|X=x) - \rho_{-y}}{(1-\rho_{+1}-\rho_{-1})\,P_{D_\rho}(\tilde{Y}=y|X=x)}$$

depends on

1. $P_{D_\rho}(\tilde{Y}=y|X=x)$ noisy posterior
2. ρ_{+1}
3. ρ_{-1}

} flip rates

4. Estimate Flip Rates (transition matrix)
   (a) Binary case

Binary Case Estimator

(I) Normal Estimator

$$\rho_{-y} = \min P(\tilde{Y}=y \mid X=x) = \max P(\tilde{Y}=-y \mid X=x)$$

(II) if anchor points exist

$$\rho_{+1} = P(\tilde{Y}=-1 \mid X=x_{+1})$$
$$\rho_{-1} = P(\tilde{Y}=+1 \mid X=x_{-1})$$

Proof

$$\begin{cases} P(\tilde{Y}=1 \mid X=x) = [1-\rho_{+1}-\rho_{-1}]\, P(Y=1 \mid X=x) + \rho_{-1} \\ P(\tilde{Y}=-1 \mid X=x) = [1-\rho_{+1}-\rho_{-1}]\, P(Y=-1 \mid X=x) + \rho_{+1} \end{cases}$$

(I) Assume $\rho_{+1}+\rho_{-1} < 1$

$$\Rightarrow P(\tilde{Y}=1 \mid X=x) \geq \rho_{-1}$$
$$P(\tilde{Y}=-1 \mid X=x) \geq \rho_{+1}$$

$$\Rightarrow \rho_{-1} = \min P(\tilde{Y}=1 \mid X=x) = \max P(\tilde{Y}=-1 \mid X=x)$$
$$\rho_{+1} = \min P(\tilde{Y}=-1 \mid X=x) = \max P(\tilde{Y}=+1 \mid X=x)$$

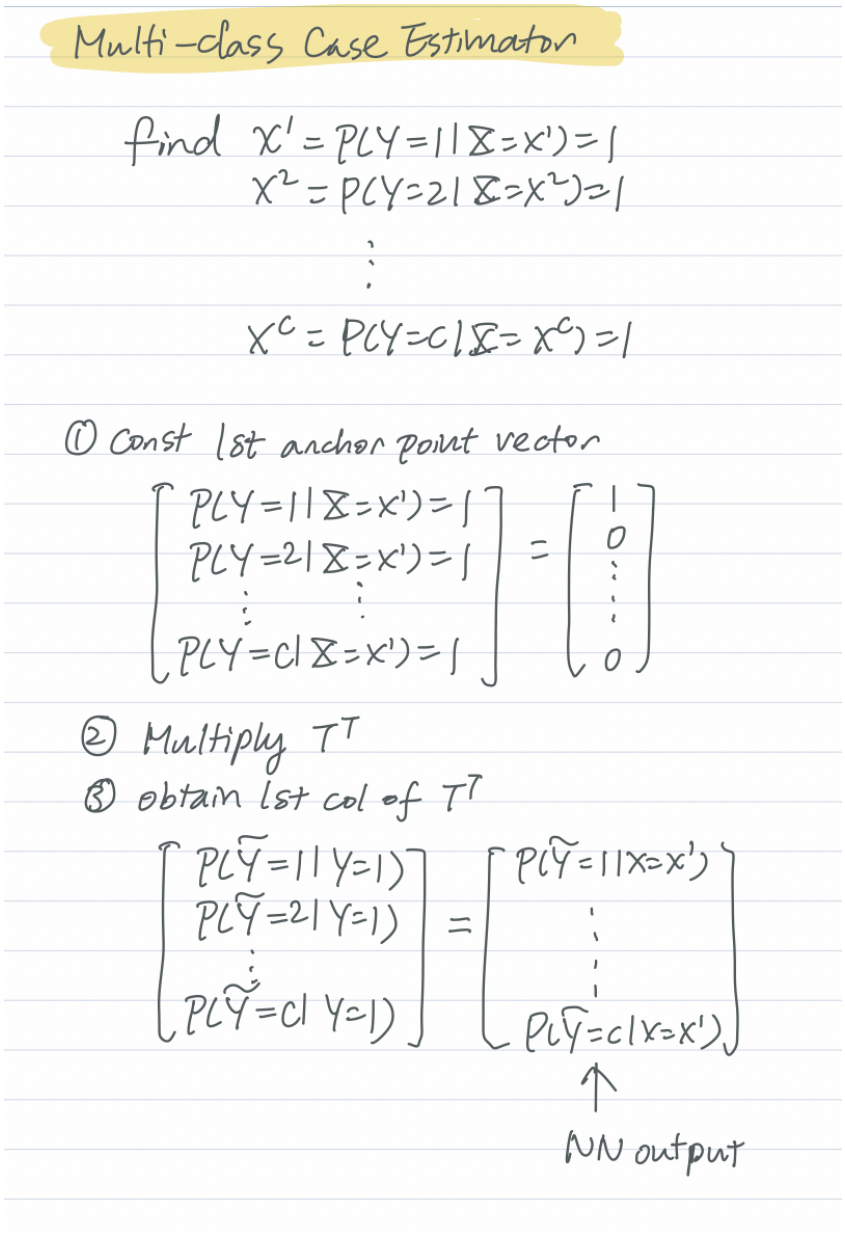
$$\Leftrightarrow \rho_{-y} = \min P(\tilde{Y}=y \mid X=x) = \max P(\tilde{Y}=-y \mid X=x)$$

(II) if $\rho_{-1} = P(\tilde{Y}=1 \mid X=x)$

$$P(Y=1 \mid X=x) = 0$$
$$P(Y=-1 \mid X=x) = 1$$
$$\Rightarrow x \text{ is } x_{-1}$$
$$\Rightarrow \rho_{-1} = P(\tilde{Y}=1 \mid X=x_{-1})$$

Similarly $\rho_{+1} = P(\tilde{Y}=-1 \mid X=x_{+1})$

(b) Multiclass case



Definition 8.5. **Instance- and Class-dependent Label Noise Model (ILN)**

9 Week 10 Reinforcement Learning

Definition 9.1. Reinforcement Learning

1. Definition: one kind of machine learning paradigms where an agent in an environment aims to learn the best policy to follow (i.e. to guide its actions) through trial-and-error in order to maximise its expected cumulative rewards.
2. Elements
   (a) Policy: determines the agent's behavior
   (b) Reward: a function of (s, a). Immediate and short-term reward. Agent adjusts its policy depending on the reward it receives (e.g. too low this time, next time select other action)
   (c) Value function: the expected cumulative reward since the current state. Long-term and over the entire lifetime of the agent. Need to be estimated.
   (d) Model: a model of the environment, which mimics the behavior of the environment (i.e. given action, provide reward and new state). Markov Decision Process is a type of model. Depending on whether model is used, we have model-based methods and model-free methods.
3. Difference from other two ML paradigms (data, goal)
   - Supervised Learning
     - Data: each example is correctly labelled.
     - Goal: learn the correct mapping between example and its correct label that generalises over unseen examples
   - Unsupervised Learning
     - Data: examples without labels
     - Goal: learn hidden structure in the data
   - Reinforcement Learning
     - Data: state-action pairs
     - Goal: maximise the expected cumulative rewards/future rewards
     - Learn from interactions

9.1 Markov Decision Process (MDP)

1. Definition: an extension of Markov Process or Markov Chains, where there are more than one actions and there is reward associated with each action.
2. Purpose: used to model the RL environment
3. Assumption: Markov Property (since MARKOV decision process). The current state s' only depends on the previous state s and the previous action a.
4. Notations
   - $\mathcal{S}$: a set of all possible states
   - $\mathcal{A}$: a set of all possible actions
   - r_t: immediate reward at time step t to reward the action a_{t-1}
   - π: agent's policy
     - for each state s, $\pi(|s)$ is a probability distribution over action. $\pi(a_{11}|s_1)$, $\pi(a_{12}|s_1)$ and $\pi(a_{13}|s_1)$ are probabilities over three possible actions under s_1 and they sum to 1. The probability distribution $\pi(|s)$ is specific to the state s and varies with s.
     - $\pi(a_t|s_t)$ represents that if the agent follows π as a policy, the agent will have $\pi(a_t|s_t)$ probability to take an action a_t given state s_t
   - π^*: agent's optimal policy
   $$\pi^* = \text{argmax}_\pi \mathbb{E}[\sum_{t\geq 0} \gamma^t r_t|\pi] \tag{24}$$
   - γ: discount factor
     - obtain the present value of all rewards; small=short-sighted; large=long-sighted
5. Interacting Procedure:
   (a) At some time step t, the agent receives a state $S_t = s_t$ (from the environment) and a reward $R_t = r_t$ for its previous action. The agent interacts with the state by making an action $A_t = a_t$.

(b) At next time step $t+1$, based on s_t and a_t, the environment sends the new reward r_t (the lecture notation) as well as the new state s_{t+1} for the agent to interact with.

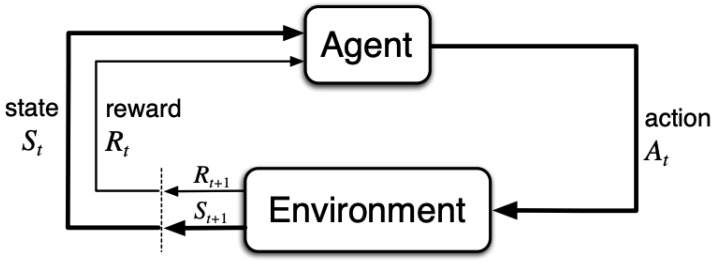


Figure 3.1: The agent–environment interaction in a Markov decision process.

6. Distributions
   (a) Initial state s_0
      - follows some random distribution
      - $s_0 \sim P(s_0)$
   (b) Action a_t
      - only depends on the current state s_t. NOT dependent on the reward as r_t is irrelevant and only for previous action a_{t-1}. r_t is obtained only after the action a_t.
      - Perform action according to the policy (i.e. probability distribution) which is state specific $\pi(a_t|s_t)$
   (c) Reward r_t (this notation is different from the book but consistent with 5318 and 5328 lectures)
      - depend on s_t and a_t. It's for the action a_t in previous time step
      - $P(r_t|s_t, a_t)$
   (d) Next state s_{t+1}
      - depends on s_t and a_t (e.g. robot rides bike example)
      - $P(s_{t+1}|s_t, a_t)$, which is also a transition probability
7. Trajectory
   - by following a policy, produces state-action-reward sequences
   - $s_0, a_0, r_0, s_1, a_1, r_1, ...$
8. Episodic Vs. Continuing Tasks
   (a) Episodic Tasks: the agent-environment interaction can be broken down into different subsequences called episodes. For example, each new round of the Pong game is an episode, and the final time step of an episode occurs when a player scores a point. The environment then will be reset to some standard starting state or to a random sample from a distribution of possible starting states.
   (b) Continuing Tasks: the agent-environment interactions don't break up naturally into episodes, but instead continue without limit. If we don't discount future rewards, the agent will keep doing the task as the reward goes to infinity.
9. Discounted Expected Accumulative Reward (i.e. Expected Accumulative Reward)
   - to overcome the infinite accumulative reward problem from continuing tasks. Makes the expected accumulative reward converge to a finite number even the rewards are infinite
   - makes agent to focus more on immediate reward than future rewards

9.2 Q-Learning

Definition 9.2. Value function

1. Definition: given a state s or a state-action pair (s, a), if the agent starts to follow the policy π *from now on* (i.e. the policy does not influence s or (s,a)), then the output of the value function is the expected cumulative reward.
2. Application: use to evaluate the expected cumulative reward of scenario when the agent starts to follow the policy π from now on after being given state s or given state-action pair (s, a)
3. **State-value function** $V^{\pi}(s)$

- Input: state s only
- $V^{\pi}(s) = \mathbb{E}[\sum_{t\geq 0}\gamma^t r_t|s,\pi]$, where $s \sim P(S_{t+1}|s_t,a_t)$, $r_t \sim P(R_t|s_t,a_t)$ and $\gamma \in (0,1)$
- Interpretation: given state s, if the agent follows π from now on, it can receive $V^{\pi}(s)$ expected cumulative reward
- Application: use to evaluate how good state s is
- **Optimal V-value** $V^*(s)$: for a given state s, the optimal state-value function gives the largest expected cumulative reward under any policy π

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

4. **Action-value function** or **Q-value Function** $Q^{\pi}(s,a)$
   - Input: state s and action a
   - $Q^{\pi}(s,a) = \mathbb{E}[\sum_{t\geq 0}\gamma^t r_t|s,a,\pi]$, where $s \sim P(S_{t+1}|s_t,a_t)$, $a \sim P(A_t|s_t)$, $r_t \sim P(R_t|s_t,a_t)$ and $\gamma \in (0,1)$
   - Interpretation: given state s and action a, if the agent follows π from now on, it can receive $Q^{\pi}(s,a)$ expected cumulative reward
   - Application: use to evaluate how good the state-action pair s and a are
   - **Optimal Q-value** $Q^*(s,a)$: for a given state-action pair (s,a), the optimal action-value function gives the largest expected cumulative reward under any policy π

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

Definition 9.3. Policy

1. Definition: policy π is a function that maps a given state s to a probability of taking each action a that is possible to that state. For each state s, $\pi(|s)$ is a probability distribution over action and the probability distribution is specific to the state.
2. Example: $\pi(a_t|s_t)$ represents that at time step t, given state s_t, if the agent follows π as a policy/function, then agent will have $\pi(a_t|s_t)$ probability to take an action a_t
3. π^*: agent's optimal policy
4. Difference between Policy and Value Function
   - Policy cares about the probabilities of each action. Following some policy π, given some state s, the policy $\pi(|s)$ evaluates how **probable** it is for an agent to take any action that is possible given s.
   - Value function cares about the expected cumulative rewards. Following some policy π, the function evaluates how **valuable** the state (if V-function) or state-action pair (if Q-function) is for an agent
5. Optimal Policy
   - Definition: Given any state s, optimal policy gives the largest (could tie) V-value (i.e. expected cumulative reward) among all other policies.

     That is, $\forall s \in \mathcal{S}$, we have $\pi \geq \pi' \Leftrightarrow V^{\pi}(s) \geq V^{\pi'}(s)$, which is equivalent to

$$\begin{aligned}\pi^* &= \text{argmax}_{\pi} V^{\pi}(s) \\ &= \text{argmax}_{\pi}\mathbb{E}[\sum_{t\geq 0}\gamma^t r_t|s,\pi]\end{aligned}$$

Definition 9.4. Bellman Optimality Equation

One property of the optimal action-value function $Q^*(s,a)$ is that it has to satisfy **Bellman Optimality Equation**, which can be defined as

$$Q^*(s,a) = r + \gamma\mathbb{E}_{s'\sim\mathcal{S}}[\max_{a'} Q^*(s',a')|s,a]$$

where the r is the expected immediate reward for r_t (rewarding current state-action (s,a). That is, $\mathbb{E}(r_t) = r$.

It states that for any state-action pair (s,a) (i.e. s_t and a_t to be exact) at time t, the optimal Q-value $Q^*(s,a)$ we can get from such pair can be considered as the sum of two following components

1. the expected value of immediate reward r (i.e. $\mathbb{E}(r_t) = r$ is a constant) for taking action a in state s at the current time t
2. the expected value of maximum discounted optimal Q-value from any next state-action pair (s',a'), We can't control s' as it is given by the environment following some distribution S, but we can pick the action a' that maximises the Q-value.

Definition 9.5. Q-Learning

1. Goal: find optimal Q-function Q^* or Optimal Q-value $Q^*(s,a)$ for any given state-action pair. Afterwards, find the optimal policy
2. Philosophy: use the Bellman optimality equation to iteratively update Q-value for each state-action pair and eventually make every Q(s,a) converges to $Q^*(s,a)$. Once all estimated optimal Q-values are obtained, the optimal policy is also ready as we can simply lookup from the Q-table.
3. Value Iteration
   - Motivation: we want to know the optimal Q-function Q^* such that given any state s we can know what action a to take that comes with the highest Q^*. Even though Q^* satisfies Bellman Equation, we actually can't obtain Q^* directly. Both LHS and RHS of Bellman Equation depends on the unknown Q^* and the only reliable number we have is the immediate reward.
   - Goal: Estimate unknown $Q^*(s,a)$ iteratively
   - Solution: use **value iteration** method. Use the Bellman equation to iteratively updates the Q-values for each state-action pair until every $Q(s,a)$ converges to the optimal Q-function $Q^*(s,a)$.
   - Update Rule:

$$\begin{aligned} Q(s,a) &\leftarrow r + \gamma \mathbb{E}_{s' \sim \mathcal{S}}[\max_{a'} Q(s',a')] \\ &\leftarrow Q(s,a) + \mathbb{E}_{s' \sim \mathcal{S}}[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \\ &\leftarrow Q(s,a) + \{\mathbb{E}_{s' \sim \mathcal{S}}[r + \gamma \max_{a'} Q(s',a')] - Q(s,a)\} \\ &\leftarrow Q(s,a) + \{Q^*(s,a) - Q(s,a)\} \text{ where Q(s,a) is known but not the } Q^*(s,a) \end{aligned}$$

     Empirically, it can be written as

$$\begin{aligned} &\leftarrow Q(s,a) + \{\hat{Q}^*(s,a) - Q(s,a)\} \\ &\leftarrow Q(s,a) + \eta\{[r + \gamma \max_{a'} Q(s',a')] - Q(s,a)\} \\ Q_{i+1}(s,a) &\leftarrow Q_i(s,a) + \eta\{[r + \gamma \max_{a'} Q_i(s',a')] - Q_i(s,a)\} \\ &\leftarrow (1-\eta) Q_i(s,a) + \eta[r + \gamma \max_{a'} Q_i(s',a')] \end{aligned}$$

   - Q-table:
     - Definition: a look-up table used to update and track q-values for each state-action pair
     - Each table entry is the estimation of $Q^*(s,a)$ and we iteratively update the estimation to make it more accurate

     Some remarks:
     - Key equation:

$$\begin{aligned} Q_{i+1}(s,a) &\leftarrow Q_i(s,a) + \eta\{[r + \gamma \max_{a'} Q_i(s',a')] - Q_i(s,a)\} \\ &\leftarrow (1-\eta) Q_i(s,a) + \eta[r + \gamma \max_{a'} Q_i(s',a')] \end{aligned}$$

     - The difference between Q can be regarded as loss. Ideally, after many iterations, when the Q(s,a) is close enough to $Q^*(s,a)$, the loss will converge to 0 or equivalently value update difference will be very small between iterations. That's how we know we can stop updating and take it as optimal value.
     - The updated Q value in the (i+1) iteration is the weighted sum of old Q-value $Q_i(s,a)$ and **learned Q-value or target** $[r + \gamma \max_{a'} Q_i(s',a')]$. We don't want to discard the old value as there may be useful information to take from. By using the learning rate η and gradually decay it, we can adjust how much old information to keep as we learn more and more about the environment.
     - when i is sufficiently large, the Q value would converge to the optimal one.

Definition 9.6. Q-learning Algorithm and Sarsa Algorithm

1. Both are temporal difference methods
2. Target Policy vs Behavior Policy
   - Motivation: All learning control methods face a dilemma: They seek to learn action values conditional on subsequent optimal behavior, but they need to behave non-optimally in order to explore all actions to find the optimal actions. How can they learn about the optimal policy while behaving according to an exploratory policy?
   - One Solution: use two policies

  - Target Policy:
    * Update Q-value with the new chosen action but not update action
    * Obtain action through Q (i.e. argmaxQ)
    * Exploitation
    * Target policy is used as optimal policy
    * Example: **Q-Learning**, **Sarsa**
  - Behavior Policy:
    * Updates action after every state such that it can generate more behavior to explore around
    * Exploration
    * Behavior policy is not used as optimal policy
    * Example: **Sarsa**

3. On-Policy vs Off-Policy

- On-policy
  - Target policy is consistent with behavior policy
  - Example: **Sarsa**, which is both target and bahavior policy. Sarsa is not only a target policy by updating the Q-value with the new chosen action a', but also a behavior policy by updating the new action a' as next action.
- Off-policy
  - target policy is inconsistent with behavior policy
  - Example: **Q-Learning**, which is target policy only. It only updates the Q-value function with the newly chosen action a from argmax, but it does not update the action as next action.

Algorithm 1 Q-Learning

Initialize the Q value $Q(s,a)$ for all state-action pairs arbitrarily
for *each episode* **do**
  Initialize s
  for s *is not a terminal state* **do**
    Choose $a = \arg\max_a Q(s,a)$
    Substitute action a and s into the environment and observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \eta\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
    $s \leftarrow s'$ //only update the state but not action
  end
end

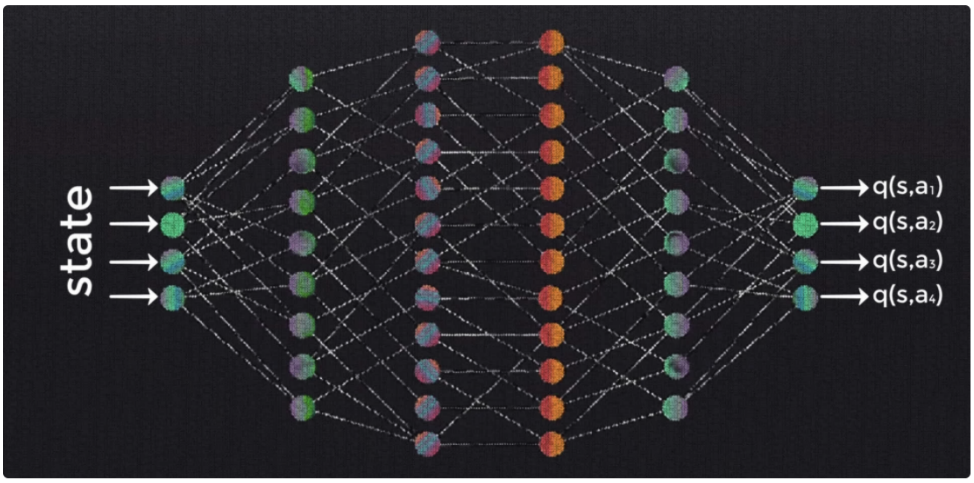
Algorithm 2 Sarsa

Initialize the Q value $Q(s,a)$ for all state-action pairs arbitrarily
for *each episode* **do**
  Initialize s
  Choose $a = \arg\max_a Q(s,a)$
  for s *is not a terminal state* **do**
    Substitute action a and s into the environment and observe r, s'
    Choose $a' = \arg\max_{a'} Q(s',a')$
    $Q(s,a) \leftarrow Q(s,a) + \eta\left(r + \gamma Q(s',a') - Q(s,a)\right)$
    $s \leftarrow s'$
    $a \leftarrow a'$ //action a is also updated
  end
end

9.3 Deep Q-Learning

Definition 9.7. **Deep Q-Networks (DQN)**

1. Definition: a deep neural network (DNN) that approximates a Q-function
2. Motivation: some tasks have continuous state-action pairs (e.g. games) and it would be infeasible and inefficient to compute, store and update all those pairs in a Q-table
3. Solution: rather than using value iteration method to iteratively calculate the optimal Q-value and then obtain the optimal Q-function, we can use deep neural network (DNN) as a function approximation to estimate the optimal Q-function (Recall that a 3-layer DNN can approximate almost any functions). For example, $Q(s, a, w) \approx Q^*(s, a)$.
4. Input: a stack of preprocessed consecutive frames. This is because a single frame is not enough for the network to fully understand the state and more frames provide more information.
5. Output: Q-value for each possible action a_i for the given state (the output layer is a fully connected layer. No softmax or any activation function as we want the raw Q-values)
6. The loss
$$L(w) = (r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$$
7. Update the weight through GD and BP.



Definition 9.8. DQN with Experience Replay and Prioritised Experience Replay

1. Motivation: learning from consecutive training examples (i.e. state-action pairs) can be problematic as there may be strong temporal relationship between those examples. DNN can capture them and then degrades the performance.
2. Approaches
   (a) **DQN with Experience Replay**
   - Solution: randomly sample examples from the **experience bank** for training.
   - Procedure

1. Initialize replay memory capacity.
2. Initialize the network with random weights.
3. *For each episode:*
    1. Initialize the starting state.
    2. *For each time step:*
        1. Select an action.
            - *Via exploration or exploitation*
        2. Execute selected action in an emulator.
        3. Observe reward and next state.
        4. Store experience in replay memory.
        5. Sample random batch from replay memory.
        6. Preprocess states from batch.
        7. Pass batch of preprocessed states to policy network.
        8. Calculate loss between output Q-values and target Q-values.
            - Requires a second pass to the network for the next state
        9. Gradient descent updates weights in the policy network to minimize loss.

   - Loss calculation: For each sample, two forward passes of states. First pass with state s. second pass with next state s' to obtain max term in the RHS of Bellman Equation. Then calculate the loss between the output $Q(s)$

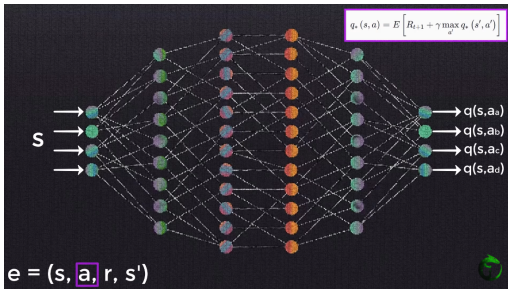


Figure 30: First Forward Pass

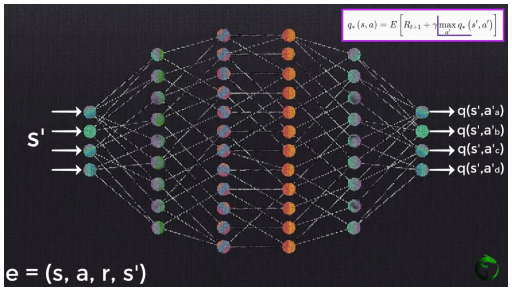


Figure 31: Second Forward Pass

- Pros: breaks the temporal correlation between consecutive samples and therefore makes the learning more effective.
- Cons: random sampling ignores the importance (i.e. update contribution) of individual samples

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t ϵ-greedy
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$ experience
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for update w. loss
end for

From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

(b) **DQN with Prioritised Experience Replay**

- Solution: weight example accordingly to their DNN weight update contribution
- Pros: take the sample update contribution into consideration and more contributing samples are more likely to be sampled
- Procedure: store experiences in a priority queue according to their error
  - DNN weight update is positively correlated with the loss value $([r + \gamma \max_{a'} Q_w(s', a')] - Q_w(s, a))^2$
  - example with a larger loss also has a larger DNN weight update and should be "prioritised" with higher assigned weight in sampling. In such a way they are more likely to be sampled.

  correlated with error

Algorithm 3 DQN with Prioritised Experience Replay (simplified)

Initialize the Q value $Q(s, a)$ for all state-action pairs arbitrarily
Initialize experience bank $B = \emptyset$
Observe s_0
for $t = 1$ *to* T **do**
  Choose $a = \arg\max_a Q_w(s_{i-1}, a)$
  Observe s_t, r_t and γ_t
  Store the transition $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$ in B with maximal priority
  if $t \equiv 0 \mod K$ *to* **then**
    Sample a transition $(s, a, r, \gamma, s', a')$ according to the priority
    Compute error $\delta = (r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$
    Update the priority of current transition according to $|\delta|$
    Update weights $w \leftarrow w + \eta \nabla (r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a))^2$
  end
end

3. Notations and Terminology

- Experience: $e_t = (s, a, r, s') = (s_t, a_t, r_t, s_{t+1})$ represents a summary of agent's experience at time t
- Experience Replay: a technique

- Experience Bank: also called replay memory, is a database where agent's experiences are stored
- N: in practice only last N experiences are stored in the experience bank

Definition 9.9. Target Network

1. Purpose: deal with non-stationary learning targets (i.e. the learned Q-value $[\gamma + max_{a'}Q(s', a')]$) and improve the learning stability
2. Procedure: Fixed the parameter $\hat{w}$ for target network $\hat{w}$

Definition 9.10. Double DQN

1. Motivation: The max operator in DQN uses the same network to select and evaluate, which makes it more likely to overestimate target value (i.e. learnt Q-value) and create biased estimation.
2. Solution: decouple the selection from evaluation. Current Q-network is used to select actions and the older Q-network is used to evaluate actions.

Definition 9.11. Policy Gradient

1. Motivation: Q-function can be really complicated especially given high-dimensional state (i.e. images). Hard to learn every state-action pair for such scenario.
2. Solution: Instead of learn state-action pair in Q-learning, learn policy directly. Use DNN to model policy π_θ.
3. Solve for the Gradient Estimator $\nabla\hat{J}(\theta)$
   (a) Expected cumulative reward for policy θ is $J(\theta)$ and we want to find the optimal policy θ^* that maximises $J(\theta)$.
   $$J(\theta) = \mathbb{E}[\sum_{t\geq 0}\gamma^t r_t|\pi_\theta]$$
   $$\theta^* = \text{argmax}_\theta J(\theta)$$
   (b) Define trajectory $\tau = (s_0, a_0, r_0, ...)$ and $r(\tau)$ as the cumulative reward of trajectory τ, then we can rewrite

$J(\theta)$ as

$$\begin{aligned}
J(\theta) &= \mathbb{E}[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta] \\
&= \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] \\
&= \int_{\tau \sim p(\tau;\theta)} r(\tau) p(\tau;\theta)\, d\tau
\end{aligned}$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int_{\tau \sim p(\tau;\theta)} r(\tau) \nabla_\theta p(\tau;\theta)\, d\tau \\
&= \int_{\tau \sim p(\tau;\theta)} r(\tau) p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)}\, d\tau \\
&= \int_{\tau \sim p(\tau;\theta)} r(\tau) p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)\, d\tau \\
&= \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau) \nabla_\theta \log p(\tau;\theta)]
\end{aligned}$$

Suppose we have n trajectories τ^1, τ^2, ..., τ^n then by plugging them in

$$\nabla_\theta \hat{J}(\theta) = \frac{1}{n} \sum_{i=1}^{n} r(\tau^i) \nabla_\theta \log p(\tau^i;\theta)$$

$$\begin{aligned}
p(\tau^i;\theta) &= [\pi_\theta(a_0^i|s_0^i) p(s_1^i|s_0^i, a_0^i)] \times [\pi_\theta(a_1^i|s_1^i) p(s_2^i|s_1^i, a_1^i)] ... \\
&= \Pi_{t\geq 0} \pi_\theta(a_t^i|s_t^i) p(s_{t+1}^i|s_t^i, a_t^i) \\
\log p(\tau^i;\theta) &= \sum_{t\geq 0} [\log \pi_\theta(a_t^i|s_t^i) + \log p(s_{t+1}^i|s_t^i, a_t^i)] \\
\nabla_\theta \log p(\tau^i;\theta) &= \nabla_\theta \sum_{t\geq 0} [\log \pi_\theta(a_t^i|s_t^i) + \log p(s_{t+1}^i|s_t^i, a_t^i)] \\
&= \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) + \sum_{t\geq 0} \nabla_\theta \log p(s_{t+1}^i|s_t^i, a_t^i) \\
&= \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) + 0 \\
&= \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \\
\nabla_\theta \hat{J}(\theta) &= \frac{1}{n} \sum_{i=1}^{n} r(\tau^i) \nabla_\theta \log p(\tau^i;\theta) \\
&= \frac{1}{n} \sum_{i=1}^{n} r(\tau^i) \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \\
&= \frac{1}{n} \sum_{i=1}^{n} \sum_{t\geq 0} r(\tau^i) \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)
\end{aligned}$$

Algorithm 4 Policy Gradient (simplified)

Initialize θ arbitrarily

for *each episode* $(s_0, a_0, r_0, \ldots, s_T, a_T, r_T) \sim \pi_\theta$ **do**

  $\Delta\theta \leftarrow 0$

  $r \leftarrow 0$

  for $t = 0$ *to* T **do**

    $\Delta\theta \leftarrow \Delta\theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t)$

    $r \leftarrow r + \gamma r_t$

  end

  $\theta \leftarrow \theta + \eta\Delta\theta$

end

9.4 RL Application

1. Alpha Go
2. Self-play
   - Produce a large amount of training data for training DNN
   - Provide auto-curriculum for the agent to learn, from simple to hard opponents

10 Causal Inference

Definition 10.1. Context

1. Simpsons Paradox
   - the trends appear in several groups but disappear when combine groups together
   - examples:
     - Age, biking, cholesterol
     - Sunny, sunburn, ice-cream sales
2. DAG: directed acyclic graph
3. Path: undirected
4. Dependence:
   - Causal
   - Correlation

Definition 10.2. Markov Condition

1. Definition: in DAG, if some certain graph property holds true then some certain statistical independence holds true too.
2. Types:
   (a) Local Markov Condition
      i. Definition: In a DAG, every variable X_i is independent of its non-descendent Y conditional on its parents, i.e. $(X_iY|PA_i)_G$.
      ii. Implication: Since this graph property holds true, it also implies the statistical property. I.e. $(X_iY|PA_i)_G \rightarrow (X_iY|PA_i)_p$
      iii. Example
      iv. Application: factorise a joint distribution based on a causal graph
   (b) Global Markov Condition (D-separation)
      i. Definition: X is d-separated from Y conditional on S if
         A. X, Y and S are three disjoint sets of variables
         B. all paths between any member of X and any member of Y are **blocked** by S
      ii. Implication: Again, $(XY|S)_G \rightarrow (XY|S)_p$
      iii. Application:
         A. factorise a joint distribution based on a causal graph
         B. determine the dependence between group variables
      iv. Procedure:
         A. Check disjoint
         B. Find all paths between X and Y and check if every path is blocked S (when not many paths)
         C. Find one counterexample that one path is not blocked by S
         D. When no nodes qualified to block any single path, S can be a empty set
   (c) Blocked
      i. Definition: a path q is said to be blocked by S as long as one of three cases happens
         A. q contains **a** chain and the middle node $m \in S$
         B. q contains **a** fork and the middle node m
         C. q contains **a** collider and the middle node $m \notin S$
      ii. If a path contains both collider and fork, as long as one of them blocks the path, the path is blocked!

Definition 10.3. Causal Faithfulness Assumption

1. Definition: this assumption ensures that the distribution does not have additional conditional dependent relations that are **NOT** entailed or included by d-separation applied to a graph
2. Related to d-separation. If holds, we say distribution p is faithful to graph G or $(XY|S)_p \rightarrow (XY|S)_G$
3. Importance: ensure the statistical property is consistent with the graph property (d-separation)

4. Application: when we don't have causal graph and we want to recover G, we can first assume p is faithful to graph G and then recover the graph G from the distribution p
5. Violation: does not always hold. counterexample in the slide, when distribution-wise independence of X and Y does not imply graph-wise independence of X and Y.

Definition 10.4. Causal Bayesian Net

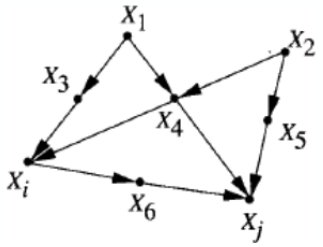
1. Definition: a causal DAG where every directed edge denotes a causal relationship
2. Application: can be used to denote external changes such as interventions

Definition 10.5. Conditioning, Intervention and Counterfactual

Definition 10.6. Identification of Causal Effectes

1. want to know the causal effect of using treatment A on recovery rate, but has another factors such as kidney stone also influences the outcome recovery rate
2. Control variable: Treatment, Outcome: Recovery rate, Other factor: kidney stone size
3. Solution: randomised control experiments, to fix all other impacting factors. In this way, any changes in the outcome variable must be due to the controlled variable
4. Cons: expensive and sometimes infeasible

Definition 10.7. Backdoor Criterion



- What if $Z = \{X_3, X_4\}$?
  $Z = \{X_4, X_5\}$?
  $Z = \{X_4\}$?
- What if there is a confounder?

1. Definition: Set Z satisfies backdoor criterion relative to an ordered pair (X_i, X_j) if two conditions are satisfied
   (a) No node in Z is descendent of X_i
   (b) All paths between X_i and X_j that contains an arrow into X_i (**Backdoor paths**) are blocked by Z
2. Examples
   (a) set X3 X4 satisfies backdoor criterion
   (b) set X4 X5 satisfies backdoor criterion
   (c) set X4 does not satisfies backdoor criterion as X4 contains a collider where middle node X4 also belongs to Z
3. When there's a confounder, use frontdoor criterion

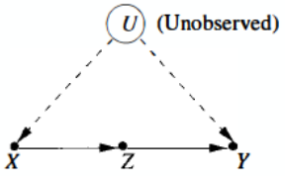
Definition 10.8. Backdoor Adjustment

Theorem 3.3.2 (Back-Door Adjustment)
If a set of variables Z satisfies the back-door criterion relative to (X, Y), then the causal effect of X on Y is identifiable and is given by the formula

$$P(y \mid \hat{x}) = \sum_z P(y \mid x, z) P(z).$$

1. Definition: used to adjust the causal effect of X_i on X_j with the help of Z
2. Application: used to identify the causal effect
3. Procedure: Once Z satisfies backdoor criterion relative to (X_i, X_j) in DAG G, then the causal effect of X_i on X_j is identifiable.

Definition 10.9. Frontdoor Criterion



Definition 3.3.3 (Front-Door)
A set of variables Z is said to satisfy the front-door *criterion relative to an ordered pair of variables (X, Y) if:*

(i) *Z intercepts all directed paths from X to Y;*
(ii) *there is no back-door path from X to Z; and*
(iii) *all back-door paths from Z to Y are blocked by X.*

Theorem 3.3.4 (Front-Door Adjustment)
If Z satisfies the front-door criterion relative to (X, Y) and if $P(x, z) > 0$, then the causal effect of X on Y is identifiable and is given by the formula

$$P(y \mid \hat{x}) = \sum_{z} P(z \mid x) \sum_{x'} P(y \mid x', z) P(x'). \tag{3.29}$$

1. Definition: Z satisfies frontdoor criterion relative to an ordered pair (X_i, X_j) if three conditions are satisfied
   (a) Z intercepts all directed paths from X to Y (i.e. all causal paths from X to Y have to go through Z)
   (b) No backdoor paths available from X to Z
   (c) All backdoor paths from Z to Y are blocked by X

Definition 10.10. Frontdoor Adjustment

1. Definition: another way of identifying causal effect of X on Y if Z satisfies front door criterion
2. Example: in slide

Definition 10.11. Causality

1. Causal Inference
   (a) Assumption: causal graph G is given
   (b) Goal: identify the causal effect of X on Y with intervention
   (c) Two Methods:
      i. Backdoor criterion and adjustment
      ii. Frontdoor criterion and adjustment
2. Causal Discovery
   (a) Goal: identify the causal structure or causal graph between variables from the observational data

Definition 10.12. Causal Sufficiency

1. A set of variable V is causally sufficient if it contains all directed cause of any pairs of variables in V
2. Example: in slide

11 Multi-task Learning

Definition 11.1. Multi-task Learning Context

1. Definition: a learning approach that learns multiple tasks at the same time assuming most individual tasks are related
2. Assumption: most individual tasks are related
3. Goal: improve the performance of each individual task
4. Suitable cases
   (a) Related individual tasks such as spam filtering
   (b) Insufficient data for individual task such as medical image classification
5. Parameter-based MTL models
   (a) Low-rank based model 1
   (b) Low-rank based model 2
6. Feature-based MTL models
   (a) Feature-based model 1
   (b) Feature-based model 2
7. Feature- and Parameter-based MTL models

Definition 11.2. Difference between Multi-task Learning and Transfer Learning

1. Different goals: MLT improves performance of all individual tasks while TL only improves performance of target tasks
2. Learning sources: MLT individual tasks learn from other related tasks while TL target domain task learns from source domain

Definition 11.3. Parameter-based MTL Models

1. Definition: assume tasks are related by parameters where w_0 is the shared parameter learnt from all datasets and Δw^i is the task specific parameter learnt from individual datasset D^i
2. Technique: add some extra regularisation term to shrink task-specific parameters, which forces individual tasks to have stronger relatedness

$$\min_{w_0, \Delta W=[\Delta w^1, \ldots, \Delta w^m]} \frac{1}{m} \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} \ell(x_j^i, y_j^i, w_0 + \Delta w^i) + \lambda \|\Delta W\|_F^2.$$

3. Low-rank based model 1

Low-rank based MTL model (I):

$$\min_{W=[w^1, \ldots, w^m]} \frac{1}{m} \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} \ell(x_j^i, y_j^i, w^i) + \lambda \ \mathrm{rank}(W).$$

   - the rank regularisation term forces the rank of parameter matrix W to be small such that individual tasks (i.e. columns of W) are more related

4. Low-rank based model 2

Low-rank based MTL model (II):

$$\min_{U,V,\Theta} \frac{1}{m} \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} \ell(x_j^i, y_j^i, u^i + \Theta^\top v^i) + \lambda \|U\|_F^2,$$

$$s.t.\ \Theta^\top \Theta = I.$$

- U and V are task-specific while Θ is the low-rank subspace shared by multiple tasks.
- The orthogonal Θ constraint ensures that each column of Θ is very different and they share little redundant information
- only regularise U here as we assume $W = U + \Theta^T V$ is fixed. Enforcing U to be small makes the shared part large hence ensures more relatedness between individual tasks.

Definition 11.4. Feature-based MTL Models

1. Definition: assume tasks are related by features
2. Trick: maps the hypotheses from the original space where hypotheses are far away from each other to a new feature space where hypotheses are close and more related to each other
3. Feature-based model 1

Feature-based MTL model (I):

$$\min_{W,P} \frac{1}{m} \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} \ell(P^\top x_j^i, y_j^i, w^i) + \lambda \text{rank}(W),$$

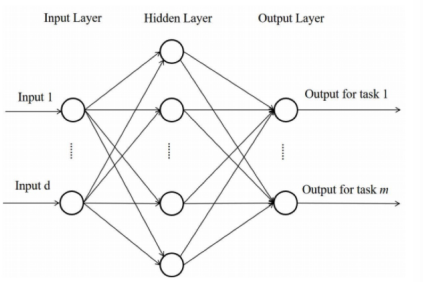
$$s.t.\ PP^\top = I.$$

Note that P is a projection matrix.

- Features are transformed by P the projection matrix into the new feature space

4. Feature-based model 2

Feature-based MTL model (II):



Shared Hidden nodes in a Neural Network. Note that neural network can be regarded as feature extractors.

- Input of NN are the old features x_j^i for each task i while output of NN are the new feature $P^T x_j^i$ for each task i
- the NN acts as a feature extractor
- Each task share hidden layers in the NN

Definition 11.5. Feature- and Parameter-based MTL Models

$$\min_{w_0, \Delta W, P} \frac{1}{m} \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} \ell(P^\top x_j^i, y_j^i, w_0 + \Delta w^i) + \lambda \|\Delta W\|_F^2,$$

$$s.t.\ PP^\top = I.$$

1. An extension of feature-based models
2. Learns feature projection P as well as commonly shared parameter w_0
3. Pros: further enhance the relatedness between tasks