

## Introduction to Machine Learning

This is a summary of the 2026 IML Script (FS26). All content from the script is (briefly) covered. There may be errors. Images belong to the IML team.

Wherever  $\|\cdot\|_p$  isn't specified,  $p = 2$ .

# Regression

## 1.1 Supervised Learning

**Supervised Learning** is the task of 'learning' a function relationship, based on a given set of inputs/outputs.

Some terminology:

|                                          |                                |
|------------------------------------------|--------------------------------|
| $x \in \mathbb{R}^d$                     | Inputs (Attributes/Covariates) |
| $\phi(x) \in \mathbb{R}^p$               | Features                       |
| $y \in \mathbb{R}$                       | Outputs (Targets/Labels)       |
| $D = \{(x_i, y_i)\}_{i=1}^n$             | Training Set                   |
| $D'$                                     | Test Set                       |
| $f: \mathbb{R}^p \rightarrow \mathbb{R}$ | Predictor (Model)              |
| $l(f(x), y)$                             | Loss                           |

**Machine Learning Pipelines** can often be classified using:

|        |                     |
|--------|---------------------|
| $F$    | Function Class      |
| $L(f)$ | Training Loss       |
|        | Optimization Method |

The function class  $F$  is a set of parametrized functions. We are looking for the  $f \in F$  that minimizes  $L(f)$ .

### D. Training Loss

$$L(f) := \frac{1}{n} \sum_{i=1}^n l(f(x_i), y)$$

## 1.2 Multiple Linear Regression

**Multiple Linear Regression** directly uses the  $x \in \mathbb{R}^d$ . Here,  $F_{\text{affine}} = \{f(x) = w^\top x + w_0 \mid w \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$ .

**Rmk.** Why are we using linear functions instead?

Any estimator  $f \in F_{\text{affine}}$  can be rewritten as  $f((x, 1)) = (w, w_0)^\top \cdot (x, 1)$ , thus we can augment the inputs  $x \mapsto (x, 1)$  and instead search in  $F_{\text{linear}} = \{f(x) = \hat{w}^\top x \mid \hat{w} \in \mathbb{R}^{d+1}\}$

## 1.3 Loss Functions

**D. Squared Loss**  $l(f(x), y) := (f(x) - y)^2$

Most common Loss Function, but sensitive to outliers.

**D. Absolute Loss**  $l_{\text{abs}}(f(x), y) := |f(x) - y|$

Less sensitive to outliers, but not differentiable.

**D. Huber Loss**

$$l_{\text{huber}}(f(x), y) := \begin{cases} \frac{1}{2}(f(x) - y)^2 & |f(x) - y| \leq \delta \\ \delta(|f(x) - y| - \frac{1}{2}\delta) & |f(x) - y| > \delta \end{cases}$$

Using parameter  $\delta$ , the penalization of outliers can be controlled

**Assymetric Loss:** In some cases it is desirable to penalize overestimation harder than underestimation, or vice versa.

**D. Quantile Loss**

$$l_\tau(f(x), y) := \tau \max\{y - f(x), 0\} + (1 - \tau) \max\{f(x) - y, 0\}$$

Using parameter  $\tau$ , over/underestimation can be penalized

## 1.4 The Normal Equation

The normal equation is the basis for the closed form solution of linear regression. (square loss)

To find  $\hat{f} := \arg \min_{f \in F_{\text{linear}}} L(f)$  we look for ideal weights  $\hat{w} \in \mathbb{R}^d$ .

$$\hat{w} := \arg \min_{w \in \mathbb{R}^d} L(f_w) = \frac{1}{n} \sum_{i=1}^n \underbrace{(y_i - w^\top x_i)^2}_{l(f(x_i), y_i)}$$

A natural abuse of notation here is  $L(w) := L(f_w)$ .

This can be rewritten in matrix notation:

$$\sum_{i=1}^n (y_i - w^\top x_i)^2 = \|y - Xw\|^2$$

The factor  $\frac{1}{n}$  is irrelevant for Optimization, it doesn't depend on  $w$

So we find a problem familiar from linear algebra:

$$\hat{w} = \arg \min_{w \in \mathbb{R}^d} \|y - Xw\|^2$$

The solution is a stationary point, so:

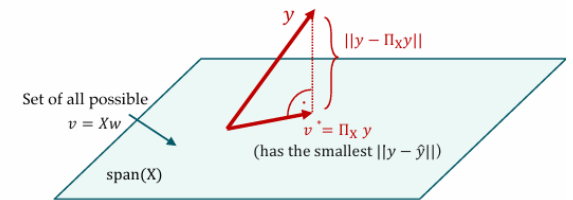
$$\nabla_w \|y - Xw\|^2 = 2X^\top(X\hat{w} - y) \stackrel{!}{=} 0$$

Which yields the **Normal Equation**.

$$\mathbf{X}^\top \mathbf{X} \hat{w} = \mathbf{X}^\top y$$

**Th. Geometric Interpretation**

$\hat{y} = \mathbf{X}\hat{w}$  for  $\hat{w}$  solving  $\mathbf{X}^\top \mathbf{X} \hat{w} = \mathbf{X}^\top y$  is the orthogonal projection of  $y$  onto  $\text{span}(\mathbf{X})$ .



Introduction to Machine Learning (2026), p. 74

## 1.5 Closed Form Solution

### Th. Minimum-Norm Solution

$$\hat{w} = \left( \mathbf{X}^\top \mathbf{X} \right)^\dagger \mathbf{X}^\top y = \mathbf{X}^\top \left( \mathbf{X}^\top \mathbf{X} \right)^{-1} \mathbf{X}^\top y$$

for  $\hat{w} = \arg \min_{w \in \mathbb{R}^d} \|w\|^2$

**Rmk.** The computational cost for this is  $\mathcal{O}(nd^2 + d^3)$ .

The closed form solution depends on  $\text{rank}(\mathbf{X})$ .

Assuming  $d \leq n$  and  $\text{rank}(\mathbf{X}) = d$ :  $(\mathbf{X}^\top \mathbf{X})^{-1}$  exists.

$$\hat{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top y \quad (\text{unique})$$

Assuming  $d > n$  or  $\text{rank}(\mathbf{X}) < d$  we have  $|\ker(\mathbf{X})| = \infty$  and there are infinite solutions. The pseudo-inverse provides the minimum-norm solution:

$$\hat{w} = \left( \mathbf{X}^\top \mathbf{X} \right)^\dagger \mathbf{X}^\top y$$

**Rmk.** If  $\text{rank}(\mathbf{X}) = d$ , then  $(\mathbf{X}^\top \mathbf{X})^\dagger = (\mathbf{X}^\top \mathbf{X})^{-1}$ .

## 1.6 Non-Linear Least Squares

To expand Linear Regression to Non-linear functions, feature maps are used on  $x$  of the form  $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^p$ .

$$f_w(x) = \sum_{j=1}^p w_j^\top \phi_j(x)$$

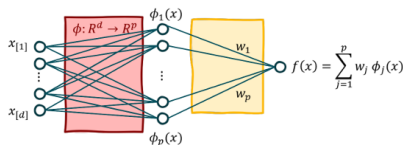
This induces a function class different from  $F_{\text{linear}}$ :

$$F_\phi = \left\{ f_w(x) = \sum_{j=1}^p w_j^\top \phi_j(x) \mid w \in \mathbb{R}^p \right\}$$

But the optimization problem remains the same:

$\Phi \in \mathbb{R}^{n \times p}$  now replaces  $\mathbf{X} \in \mathbb{R}^{n \times d}$ .

$$\hat{w} = \arg \min_{w \in \mathbb{R}^p} \|y - \Phi w\|^2$$



Introduction to Machine Learning (2026), p. 79

## 2 Optimization

**Problem:** Finding  $\hat{w}$  for  $l$  with no closed form solution.

or if the closed form solution is too expensive to compute.

**Solution:** Iterative optimization methods.

### Algorithm 1: Iterative Optimization

```

t ← 0 ;
w(0) ← winitial ;
repeat
    w(t+1) ← w(t) +  $\tilde{\eta}_t v^{(t)}$  ;
    t ← t + 1
until Stopping Criterion;
return w(t)

```

**Notation** The update takes the form  $\tilde{\eta}_t v^{(t)}$ .  $v^{(t)}$  is the update direction,  $\tilde{\eta}_t$  is the step size.

### 2.1 Gradient Descent

**Intuitively:** go in the direction  $v^{(t)}$  where  $L$  decreases most.

**L.**  $-\nabla L(w^{(t)})$  is the direction of steepest descent.

Assuming diff.-able  $L$ . Provable via Taylor expansion & Cauchy-Schwarz.

#### D. Gradient Descent Update Step

$$w^{(t+1)} = w^{(t)} - \tilde{\eta}_t \cdot \frac{\nabla L(w^{(t)})}{\|\nabla L(w^{(t)})\|} \quad (\text{Normalized})$$

$$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla L(w^{(t)}) \quad (\text{Unnormalized})$$

**Rmk.** Strictly,  $\eta$  is no longer the step size, but a step size factor. The actual step size is  $\tilde{\eta}_t = \eta \cdot \|\nabla L(w^{(t)})\|$ .

Unnormalized gradient descent takes advantage of  $\|\nabla L(w^{(t)})\|$ :

- $\|\nabla L(w^{(t)})\|$  small  $\mapsto$  close to stat. point  $\mapsto$  small steps.
- $\|\nabla L(w^{(t)})\|$  large  $\mapsto$  far from stat. point  $\mapsto$  large steps.

Stopping criterion uses the same idea:  $\|w^t - w^{t+1}\| < \epsilon$  or equivalently  $\|\nabla L(w^{(t)})\| < \frac{\epsilon}{\eta}$ , i.e. stop if little change is being made.

### Algorithm 2: Gradient Descent

```

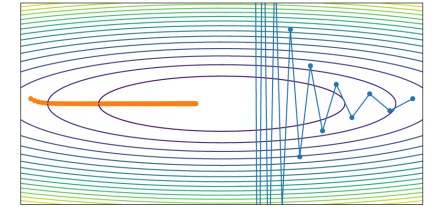
t ← 0 ;
w(0) ← winitial ;
repeat
    w(t+1) ← w(t) -  $\eta \nabla L(w^{(t)})$  ;
    t ← t + 1
until  $\|w^t - w^{t+1}\| < \epsilon$ ;
return w(t)

```

**D. Descent Direction:**  $v$  s.t. for a suitable  $\eta > 0$ , we have  $L(w - \eta v) < L(w)$ , i.e.  $L$  decreases.

**Th.** GD-Update is a *Descent Direction* for a suitable  $\eta$ .  
i.e. if we choose  $\eta$  well, GD decreases  $L$  at each step.

|                  |                              |
|------------------|------------------------------|
| $\eta$ too small | slow convergence             |
| $\eta$ too large | overshoots stationary points |



Introduction to Machine Learning (2026), p. 90

## 2.2 Convergence Analysis

**Question:** When & how fast does GD converge to  $\hat{w}$ ?

### D. Linear Convergence

$\forall t : \|w^{(t)} - \hat{w}\| \leq C \cdot \rho^t$  for some  $C > 0, 0 < \rho < 1$ .

Called linear, since we multiply with a fixed  $\rho < 1$  at each step.

**Rmk.** Linear convergence is exponential in terms of  $t$ .

### Th. Linear Convergence (GD)

GD converges linearly to the optimal solution  $\hat{w}$ :

$$\hat{w} = \arg \min_{w \in \mathbb{R}^d} L(w)$$

Where  $\text{rank}(\mathbf{X}^\top \mathbf{X})$  is full and  $\mu < \frac{2}{\lambda_{\max}(\mathbf{X}^\top \mathbf{X})}$

Convergence & convergence rate depend on  $\eta$ .

The optimal  $\eta$  depends on  $\mathbf{X}$ :

$$\eta_{\text{opt}} = \frac{2}{\lambda_{\max} + \lambda_{\min}}$$

Where  $\lambda_{\max} = \lambda_{\max}(\mathbf{X}^\top \mathbf{X})$  and  $\lambda_{\min} = \lambda_{\min}(\mathbf{X}^\top \mathbf{X})$

### D. Condition Number $\kappa$

$$\kappa := \frac{\lambda_{\max}}{\lambda_{\min}}$$

The optimal  $\rho$  is  $\rho_{\min} = \frac{\kappa-1}{\kappa+1}$ . Thus, the convergence rate is determined only by the eigenvalues of  $\mathbf{X}^\top \mathbf{X}$ , i.e.  $\kappa$ .

|                    |                  |                  |
|--------------------|------------------|------------------|
| $\kappa \approx 1$ | well-conditioned | fast convergence |
| $\kappa \gg 1$     | ill-conditioned  | slow convergence |

### Rmk. Geometrical Interpretation of $\kappa$

Considering the ellipsoids  $L^{-1}(c) = \{w \in \mathbb{R}^d : L(w) = c\}$ , i.e. the contour lines of  $L$ : For a large  $\kappa$ , the ellipsoids are distorted (large axis ratio). Equivalently for small  $\kappa$ , the ellipsoids are nearly spherical.

## 2.3 Stochastic Gradient Descent

### Method Minibatch Stochastic Gradient Descent

The main cost of GD stems from calculating & storing

$\nabla_w l(f_w(x_i), y_i)$  for all  $x_i$ , at every step.

We instead use only  $\mathcal{S} \subset \{1, 2, \dots, n\}$ , selected randomly:

$$\nabla L_{\mathcal{S}}(w) = \frac{1}{S} \sum_{i \in \mathcal{S}} \nabla_w l(f_w(x_i), y_i)$$

**Rmk.** if  $|\mathcal{S}| = 1$ , this is called *Stochastic Gradient Descent*.

### Algorithm 3: Stochastic Minibatch Gradient Descent

```

t ← 0 ;
w(0) ← winitial ;
repeat
    S ← S ⊂ {1, ..., n} randomly ;
    w(t+1) ← w(t) - η ∇LS(w(t)) ;
    t ← t + 1
until ||wt - wt+1|| < ε ;
return w(t)

```

Note how it no longer holds that the direction of SGD is a descent direction, however:

### L. Descent Direction in Expectation

Consider  $|\mathcal{S}| = k$  s.t.  $\mathcal{S} \stackrel{\text{i.i.d.}}{\subset} \{1, \dots, n\}$ . (duplicate selections possible)  
 $\mathcal{S} = \{I_1, \dots, I_k\}$  s.t.  $I_l = I_m$  is possible for  $l \neq m$

$$\begin{aligned}
 \mathbb{E}_{\mathcal{S}} [\nabla L_{\mathcal{S}}(w)] &= \mathbb{E}_{\mathcal{S}} \left[ \frac{1}{k} \sum_{j=1}^k \nabla_w l(f_w(x_{I_j}), y_{I_j}) \right] \quad (\text{def. } \nabla L_{\mathcal{S}}) \\
 &= \frac{1}{k} \sum_{j=1}^k \mathbb{E}_{I_j} [\nabla_w l(f_w(x_{I_j}), y_{I_j})] \quad (\text{lin. } \mathbb{E}) \\
 &= \frac{1}{k} \sum_{j=1}^k \left( \sum_{i=1}^n \frac{1}{n} \nabla_w l(f_w(x_i), y_i) \right) \quad (\text{def. } \mathbb{E}_{I_j}) \\
 &= \sum_{i=1}^n \frac{1}{n} \nabla_w l(f_w(x_i), y_i) \quad (\text{arith.}) \\
 &= \nabla L(w)
 \end{aligned}$$

## 2.4 Other Gradient Methods

### Method Momentum

The GD Update is modified to consider previous updates:

$$w^{(t+1)} = w^{(t)} - \eta \nabla L(w^{(t)}) + \underbrace{\alpha(w^{(t)} - w^{(t-1)})}_{\text{Momentum Term}}$$

Where  $\alpha = \eta \cdot \beta$  s.t.  $\beta \in [0, 1]$  is the *momentum coefficient*.

**Rmk. Intuition:** Dampens oscillations in the gradient direction, especially for ill-conditioned problems. The momentum term is a weighted average of previous updates.

**Rmk. Previous Gradients:** The influence of prev. gradients decreases exponentially. This can be seen by expanding the recursion above.

### Method Adaptive Methods

Adaptive methods use parameter-specific learning rates for each parameter  $w_j$ :

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{\delta_i^{(t)} + \gamma}} \cdot \frac{\partial L}{\partial w_i}(w^{(t)})$$

Where  $\delta_i^{(t)} = (w_i^{(t)} - w_i^{(t-1)})^2$  and  $\gamma > 0$ .

**Rmk. Intuition:** The idea above is that parameters that have changed significantly already should have lower learning rates.

**Rmk.** ADAM is an adaptive method.

### Method Second Order Methods

Methods which use the Hessian  $\mathbf{H}_L(w^{(t)})$ .

Computationally expensive and usually not done directly.

### 3 Model Selection

### 4 Regularization

### 5 Classification

In regression, we search an  $\hat{f} : \mathbb{R}^d \rightarrow \mathbb{R}$ , i.e.  $y, \hat{y} \in \mathbb{R}$ .  
In classification, we want  $\hat{y} \in \mathcal{Y} \subset \mathbb{R}$ , s.t.  $\mathcal{Y}$  is discrete.

#### 5.1 Binary Classification

We generally use  $\mathcal{Y} = \{+1, -1\}$  and set  $\hat{y} = \text{sgn}(\hat{f}(x))$ .  
So, a linear classifier where  $\hat{f}(x) = w^\top x$  takes the form:

$$x \mapsto \begin{cases} 1 & w^\top x > 0 \\ -1 & w^\top x < 0 \end{cases}$$

**D. Decision Boundary**  $\left\{ x \in \mathbb{R}^d \mid \hat{f}(x) = 0 \right\}$

Like in regression, using features is again possible.

#### 5.2 Surrogate Loss

We'd like to reuse the loss minimization from regression.  
A natural metric for accuracy is simply checking if  $\hat{y} = y$ .

**D. Zero-One Loss**

$$l_{0-1}(\hat{y}, y) := \mathbb{I}_{\hat{y} \neq y} = \begin{cases} 1 & \hat{y} \neq y \\ 0 & \hat{y} = y \end{cases}$$

We could try minimizing this:

$$\sum_{(x,y) \in \mathcal{D}} l_{0-1}(\hat{y}, y) = \sum_{(x,y) \in \mathcal{D}} \mathbb{I}_{f_w(x) \cdot y < 0}$$

Unfortunately,  $l_{0-1}$  is non-continuous and non-convex.  
We introduce *surrogate loss* to still apply GD.

**Rmk.** Note how  $\nabla l_{0-1}(\hat{y}, y) = 0$  everywhere.

Note how  $\mathbb{I}_{\hat{y} \neq y} = \mathbb{I}_{\hat{y} \cdot y < 0}$ , so  $l_{0-1}$  only depends on  $z := \hat{y} \cdot y$ .  
We thus define losses over  $z$ , that are cont. and convex.

**D. Surrogate Loss**

$$l_{\text{exp}} = e^{-z} \quad l_{\text{log}} = \log(1 + e^{-z})$$

A notable difference is that  $l'_{\text{exp}}$  is unbounded,  
while  $l'_{\text{log}} = \frac{1}{1+e^z} \in (-\frac{1}{2}, -1)$  for  $z < 0$ .  
This is better for outliers, thus  $l_{\text{log}}$  is usually preferred.

## 5.3 Logistic Regression

We assume  $w_0 = 0$

We try to minimize  $l_{\log} = \log(1 + e^{-z})$ , so:

$$L(w) = \frac{1}{n} \sum_{i=1}^n l_{\log}(z_i) = \frac{1}{n} \sum_{i=1}^n \log\left(1 + e^{-\overbrace{y_i \cdot w^\top x_i}^{z_i}}\right)$$

Assume  $\{x_i, y_i\}_{i=1}^n$  is linearly separable, i.e.

$$\exists w \in \mathbb{R}^d : \underbrace{y_i \cdot w^\top x_i}_{z_i} > 0 \quad \forall i \leq n$$

Then there are multiple valid decision boundaries. Additionally,  $L(w)$  is then convex.

the distance  $x_0$  to the decision boundary is:  $\|x_0\|_2 \cdot |\cos(\theta)|$ .  
 $\theta$  between  $w, x_0 \in \mathbb{R}^d$

$$\|x_0\|_2 \cdot |\cos(\theta)| = \|x_0\|_2 \cdot \frac{|w^\top x_0|}{\|w\|_2 \cdot \|x_0\|_2} = \frac{|w^\top x_0|}{\|w\|_2}$$

Note if  $w$  is a unit-vector, this is just  $|w^\top x_0|$

**D. Margin**  $\text{margin}(w) := \min_{1 \leq i \leq n} y_i \cdot w^\top x_i$

## 5.4 Solutions

### D. Maximum Margin Solution

$$w_{\text{MM}} := \max_{\|w\|_2=1} \min_{1 \leq i \leq n} (y_i \cdot w^\top x_i)$$

If  $\mathcal{D}$  is linearly separable, this is convex.

**Rmk.** Also called *hard-margin SVM*, assuming lin.-sep. data

### D. Support Vector Machine

$$w_{\text{SVM}} := \min_{w \in \mathbb{R}^d} \|w\|_2 \quad \text{s.t.} \quad y_i \cdot w^\top x_i \geq 1 \quad \forall i \leq n$$

Solving these problems is actually equivalent, up to scaling:

**L.**  $\frac{w_{\text{SVM}}}{\|w_{\text{SVM}}\|_2} = w_{\text{MM}}$  (This also holds for the case  $w_0 \neq 0$ )

By relaxing the SVM constraints, we can use the SVM problem on lin.-insep. data too:

$$y_i \cdot w^\top x_i \leq 1 \quad \rightarrow \quad y_i \cdot w^\top x_i \leq 1 - \zeta$$

$\zeta = (\zeta_1, \dots, \zeta_n)$  s.t.  $\zeta_i \geq 0$

### D. Soft-margin SVM

$$w_{\text{SM}} = \min_{w \in \mathbb{R}^d, \zeta \in \mathbb{R}^n} \left( \|w\|^2 + \lambda \sum_{i=1}^n \zeta_i \right) \quad \text{s.t.} \quad \begin{cases} y_i \cdot w^\top x_i \geq 1 - \zeta_i \\ \zeta_i \geq 0 \quad \forall i \leq n \end{cases}$$

$\lambda$  (hyperparam.) intuitively controls how much a violation is penalized

Another perspective: The optimal  $\zeta_i$  are:

$$\zeta_i = \begin{cases} 1 - y_i \cdot w^\top x_i & \text{if } y_i \cdot w^\top x_i \leq 1 \\ 0 & \text{else} \end{cases}$$

So the problem can be formulated without  $\zeta$  too:

### D. $l_2$ -penalized Hinge Loss Optimization

$$\min_{w \in \mathbb{R}^d} \left( \|w\|^2 + \lambda \underbrace{\sum_{i=1}^n \max(0, 1 - y_i \cdot w^\top x_i)}_{\text{Hinge Loss}} \right)$$

## 5.5 Gradient Descent for Classification

In practice, instead of explicitly solving  $w_{\text{SVM}}$  or  $w_{\text{MM}}$ , GD is usually applied on a diff.-able convex surrogate loss.

### 5.5.1 On linearly separable data

Assuming  $\{x_i, y_i\}_{i=1}^n$  is lin. separable,  $L(w) = \frac{1}{n} \sum_{i=1}^n l_{\log}(z_i)$  is convex, but no global optimum exists. Using GD,  $L(w)$  will approach 0, but the iterates  $\{w^t \mid t \in \mathbb{N}\}$  diverge. However,  $w^t$  may converge *in direction*, and interestingly:

**Th. GD converges to  $w_{\text{MM}}$  for lin.-sep. data** (log. loss)

$$\lim_{t \rightarrow \infty} \frac{w^t}{\|w^t\|} = w_{\text{MM}}$$

On  $L(w)$  (logistic regression),  $\mu = 1$

### 5.5.2 On linearly inseparable data

Assuming  $\{x_i, y_i\}_{i=1}^n$  is strictly lin. inseparable, i.e.

$$\forall w \neq 0 : \exists i \leq n : y_i \cdot w^\top x_i < 0$$

**Th. GD converges on lin.-insep. data** (log. loss)

$$\exists \hat{w} \in \mathbb{R}^d : \lim_{t \rightarrow \infty} w^t = \hat{w}$$

On  $L(w)$  (logistic regression),  $\mu = \frac{4}{\sigma_{\max}^2(X)}$

**Rmk.** Only holds for  $l_{\log}$ . In general, this is a hard problem.

## 5.6 Multiclass Classification

What if  $|\mathcal{Y}| > 2$ ? E.g.  $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{fish}\}$

Idea: Train  $K := |\mathcal{Y}|$  classifiers  $\hat{f}_1, \dots, \hat{f}_K \in F$ .

Why not one  $\hat{f}$ ? E.g. discretize further:  $1 \mapsto \text{cat}$ ,  $2 \mapsto \text{dog}$ ,  $3 \mapsto \text{fish}$ .

Problem: this assignment suggests cats are closer to dogs than fish.

We can then predict the class using these  $\hat{f}_k$ :

$$\hat{y}(x) = \arg \max_{1 \leq k \leq K} \hat{f}_k(x)$$

This leads to one decision boundary per class.

### D. One-vs-Rest Training

Train each model separately by relabeling for each  $\hat{f}_k$ :

1. Define  $\mathcal{D}_k = \{x_i, \tilde{y}_i\}$  where  $\tilde{y}_i := \begin{cases} -1 & y_i = k \\ 1 & y_i \neq k \end{cases}$
2. Run binary classification on  $\mathcal{D}_k$  to get  $\hat{f}_k$

This leads to  $K$  classification problems, which might be slow

**Rmk.** Visually, this leads to *convex* regions for each label.

Another way to reuse the existing methodology is to use a new loss:

### D. Cross-Entropy Loss

$$l_{\text{ce}}(\hat{f}_1(x), \dots, \hat{f}_K(x), y) = -\log\left(\frac{\exp(\hat{f}_y(x))}{\sum_{k=1}^K \exp(\hat{f}_k(x))}\right)$$

$y \in \{1, \dots, K\}$ ,  $\hat{f}_i(x) \in \mathbb{R}$

$l_{\text{ce}}$  encourages the *true class*  $\hat{f}_{y_i}(x_i)$  to be the largest  $\hat{f}_k(x_i)$ .

**Rmk.** For  $K = 2$ , if we use  $\mathcal{Y} = \{+1, -1\}$  then  $l_{\text{ce}} = l_{\log}$ .

The parametrized optimization problem then is:

$$\min_{w_1, \dots, w_K \in \mathbb{R}^d} \left( \sum_{i=1}^n l_{\text{ce}}(f_w(x_i), y_i) \right)$$

Here,  $w \in \mathbb{R}^{d \times K}$  is a matrix:  $w = (w_1, \dots, w_K)$

This then yields  $\hat{f}_k = f_{\hat{w}_k}$

**Rmk.** These methods may lead to very different decision boundaries!

## 5.7 Generalization

First, a lot of assumptions:

1. Inputs  $X \in \mathcal{X}$  come from a prob. distribution  $\mathbb{P}_X$
2. Training & Test set sampled i.i.d. from same distribution  
Note that in general, this is rarely true.
3. There exists a ground-truth  $y^*$
4. The observed labels are noisy:  $(y \mid x) = \epsilon \cdot y^*(x)$   
A *multiplicative* noise model, unlike lin.-reg. :  $\mathcal{Y} = f^*(X) + \epsilon$
5.  $\epsilon$  is also from a prob. distribution  $\mathbb{P}_\epsilon$   
Not necessarily indep. from  $x$ !

Focusing on  $y^*(x) \in \{+1, -1\}$ , we set  $\epsilon \in \{+1, -1\}$ .

Intuitively:  $\epsilon$  just flips the label

This allows defining a Joint-Distribution

$$\mathbb{P}[x, y] = \mathbb{P}_x[x] \cdot \mathbb{P}[y \mid x]$$

### 5.7.1 Evaluation

An intuitive metric to check is proximity to  $y^*$ :

Which can be done using the 0-1-loss

$$l(\hat{y}(x), y^*(x)) = \mathbb{I}_{\hat{y}(x) \neq y^*(x)}$$

Now, we can define the expected classification error:

$$\mathbb{E}_X[l(\hat{y}(x), y^*(x))] = \mathbb{E}_X[\mathbb{I}_{\hat{y}(x) \neq y^*(x)}] = \mathbb{P}[\hat{y}(x) \neq y^*(x)]$$

We can't compute or estimate this, since we don't have  $y^*$ . However, we can find an estimate of  $\mathbb{E}_{X,Y}[l(\hat{y}(X), Y)]$  using the observed  $X, Y$ .

Why is this useful? It approximates the generalisation error:

### D. Generalisation Error (0-1-Loss)

$$L(\hat{f}; \mathbb{P}_{X,Y}) = \mathbb{E}_{X,Y}[l(\hat{f}(X), Y)] = \mathbb{P}_{X,Y}(\hat{y} \neq y)$$

We can empirically evaluate this on a test set:

$$\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y) \in \mathcal{D}_{\text{test}}} \mathbb{I}_{\hat{y}(x) \neq y^*(x)}$$

## 5.8 Hypothesis Testing

Classical statistical methods can also be used.

### D. Asymmetric Errors

Misclassifications may be weighted differently.

Mistakenly classifying  $x$  with  $y^*(x) = 1$  as 2, may be worse than 3.

For binary classification, we may label:

$$+1 \mapsto \text{"Positive"} \quad -1 \mapsto \text{"Negative"}$$

This leads to the notion of confusion matrices.

|                | $y = -1$    | $y = +1$     |
|----------------|-------------|--------------|
| $\hat{y} = -1$ | TN          | FN (Type II) |
| $\hat{y} = +1$ | FP (Type I) | TP           |

### D. Empirical Measure

For an event  $A \subset \mathcal{X} \times \{+1, -1\}$

$$\mathbb{P}_n[A] := \frac{1}{n} \sum_{i=1}^n \mathbb{I}_{(x_i, y_i) \in A}$$

$$\mathcal{D} = \{(x_i, y_i) \mid i \leq n\} \subset \mathcal{X} \times \{+1, -1\}$$

$\mathbb{P}_n[A]$  is the percentage of  $(x, y) \in \mathcal{D}_{\text{train}}$  that belong to  $A$ .

**L.**  $\mathbb{P}_n[A]$  is an estimate of  $\mathbb{P}_{X,Y}[A]$ :

$$\lim_{n \rightarrow \infty} \mathbb{P}_n[A] = \mathbb{P}_{X,Y}[A] \quad (\text{Law of large numbers})$$

### Rmk. Asymmetric Loss in binary lassification

We can now weigh FP, FN differently in the 0-1-error:

$$\frac{c_{\text{FN}}}{|\{x \mid y = +1\}|} \underbrace{\sum_{(x,y), y=1} \mathbb{I}_{\hat{y} \neq -1}}_{\# \text{FN}} + \frac{c_{\text{FP}}}{|\{x \mid y = -1\}|} \underbrace{\sum_{(x,y), y=-1} \mathbb{I}_{\hat{y}(x) = +1}}_{\# \text{FP}}$$

Here  $c_{\text{FP}}, c_{\text{FN}}$  are the weights for penalization

Generally, reducing FP errors increases FN errors, and vice versa.

## 5.9 ROC Curves

**Rmk.** A side-effect of using  $\hat{y}(x) = \text{sign} \hat{f}(x)$  is that the magnitude  $|\hat{f}(x)|$  can be interpreted as *confidence*. We can set:

$$\hat{y}_\tau(x) = \text{sign}(\hat{f}(x) - \tau) = \begin{cases} +1 & \text{if } \hat{f}(x) > \tau \\ -1 & \text{if } \hat{f}(x) < \tau \end{cases}$$

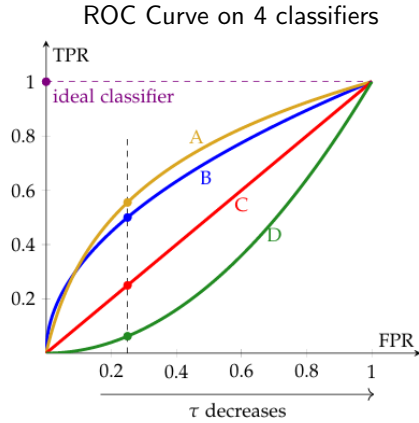
Now  $\tau$  can be used to penalize FP ( $\tau > 0$ ) or FN ( $\tau < 0$ ).

Note how this way, we don't modify the Optimization problem.

What if we don't know which FP/TP rate is desired?

Formally: which  $\tau$  should be used?

**D. ROC Curve** (Receiver Operating Characteristic) Plots TP Rate against FP Rate for different  $\tau$ .



Introduction to Machine Learning (2026), p. 160

**Rmk.** **How to read this?** A straight line is equivalent to random guessing, anything above is better.  $\tau$  isn't directly included in the curve, but it follows from the definition that  $\tau$  decreases as the FP rate increases.

How can we measure performance independent of  $\tau$ ?

**D. AUROC** (Area under ROC)

AUROC is 1 for the ideal classifier, and always in  $[0, 1]$ .

## 6 Kernels

**Motivation:** Regression using feature maps  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$ :

$$\min_{w \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n l(y_i, w^\top \cdot \phi(x_i))$$

What if computing/storing  $\phi(x)$  is expensive/infeasible?

e.g. if  $p$  is large, or infinite

**Rmk.** To store a poly.  $p(x) : \mathbb{R}^d \rightarrow \mathbb{R}$  with  $\deg(p) = m$  we require  $p = \mathcal{O}(d^m)$  features. Storing  $n$  data points requires  $\mathcal{O}(nd^m)$  memory. Not good.

### 6.1 Kernelization

By constraining  $w$  to  $\text{span}(\Phi^\top) \subset \mathbb{R}^p$  we can drastically improve memory usage. Since we know a minimizer exists here, we don't "lose anything".

#### D. Kernelization

1. **Reparametrization:** We assume  $w = \Phi^\top \alpha$  (i)

2. **Loss via Inner Products:** Observe:

$$f(x) = w^\top \phi(x) \stackrel{(i)}{=} (\Phi^\top \alpha)^\top \phi(x) = \sum_{i=1}^n \alpha_i (\phi(x_i)^\top \phi(x))$$

Note:  $x_i$  only appears in *inner products*  $\phi(x_i)^\top \phi(x_j)$

3. **Replace Inner Products:** We define:

$$k : \begin{cases} \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \\ k(x, x') = \phi(x)^\top \phi(x') \end{cases} \quad K : \begin{cases} K \in \mathbb{R}^{n \times n} \\ K_{ij} = k(x_i, x_j) \end{cases}$$

Now, we can reformulate the optimization problem:

$$\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n l\left(y_i, \sum_{j=1}^n \alpha_j k(x_i, x_j)\right) = \min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n l(y_i, (K\alpha)_i)$$

By storing  $K \in \mathbb{R}^{n \times n}$  instead of  $\phi(x) \in \mathbb{R}^p$  for  $i = 1, \dots, n$ , the memory usage is reduced:  $\mathcal{O}(np) \rightarrow \mathcal{O}(n^2)$ .

### 6.2 The Kernel Trick

Using  $k$ , the computation time is still  $\mathcal{O}(n^2 p)$  if

$$k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$$

So let's replace  $k$  with a simple function, which guarantees the existence of some  $\phi$  (which we never calculate).

**Rmk.** Since we only *implicitly* specify  $\phi$  via  $k$ , we can use  $\phi$  s.t.  $p = \infty$  now.

**D. Kernel Function**  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$

1.  $k$  is symmetric:  $\forall x, x' : k(x, x') = k(x', x)$

2.  $k$  is PSD:  $\forall n \in \mathbb{N}, \forall (x_1, \dots, x_n) \in \mathbb{R}^d$ :

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{bmatrix} \text{ is PSD}$$

**Th. Kernels guarantee existence of  $\phi$**

If  $k$  is a kernel, there exists a Hilbert Space  $(\mathcal{H}, \langle \cdot, \cdot \rangle_{\mathcal{H}})$  s.t.

$$\exists \phi : \mathbb{R}^d \rightarrow \mathcal{H} \text{ s.t. } k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}} \quad \forall x, x' \in \mathbb{R}^d$$

$\mathcal{H}$  may be, for example,  $\mathbb{R}^p$  with  $\|\cdot\|_2$ .

#### L. Properties of Kernels

1. Composed feature maps are Kernels

$$\left. \begin{array}{l} \phi : \mathbb{R}^d \rightarrow \mathbb{R}^p \\ \psi : \mathbb{R}^d \rightarrow \mathbb{R}^p \end{array} \right\} \quad k(x, x') = \langle \psi(\phi(x)), \psi(\phi(x')) \rangle$$

2. Kernels can be added in 2 ways, yielding a kernel

$$k_1 : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}, \quad k_2 : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}, \quad \text{(ii) assumes } d = d'$$

$$(i) \quad k((x, y), (x', y')) = k_1(x, x') + k_2(y, y')$$

$$(ii) \quad k(x, x') = k_1(x, x') + k_2(x, x')$$

3. Kernels can be multiplied in 2 ways, yielding a kernel

$$k_1 : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}, \quad k_2 : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}, \quad \text{(ii) assumes } d = d'$$

$$(i) \quad k((x, y), (x', y')) = k_1(x, x') \cdot k_2(y, y')$$

$$(ii) \quad k(x, x') = k_1(x, x') \cdot k_2(x, x')$$

## L. Non-negative Taylor Series

$g : \mathbb{R} \rightarrow \mathbb{R}$  s.t.  $g$  has only non-neg. coeffs. in its Taylor Series

$k(x, x') = g(\langle x, x' \rangle)$  is a valid kernel

### 6.2.1 Commonly used Kernels

#### D. Polynomial $g(\langle x, x' \rangle) = (1 + x)^m$

Where  $m$  decides the maximum polynomial degree.

#### D. Radial Base Function (RBF)

$p, \alpha$  are parameters,  $\tau$  is called *bandwidth parameter*.

$$k(x, x') = \exp\left(-\frac{\|x - x'\|_p^\alpha}{\tau}\right)$$

The RBF kernel for some special  $p, \alpha$  is named:

#### D. Gaussian

Sometimes used synonymously as just "RBF".

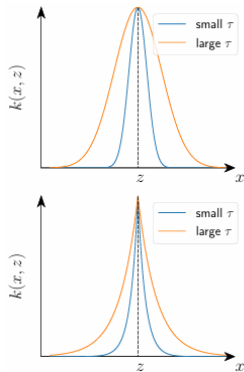
$$k(x, x') = \exp\left(-\frac{\|x - x'\|_2^2}{\tau}\right)$$

#### D. Laplacian

Sometimes defined with  $\alpha = 1$ , e.g. in *scikit-learn*.

$$k(x, x') = \exp\left(-\frac{\|x - x'\|_1^2}{\tau}\right)$$

Gaussian (top) and Laplacian (bottom):



Introduction to Machine Learning (2026), p. 173

**Rmk.** The feature space of the RBF kernels has  $\dim(\mathcal{S}_{\text{RBF}}) = \infty$ .

## 7 Neural Networks

**Motivation:** So far, when looking for  $\hat{f}(x)$  the form was  $\hat{f}(x) = w^\top x$ , or  $\hat{f}(x) = w^\top \phi(x)$ . Note how the features  $x, \phi(x)$  are predetermined. Why not learn them?

### New Optimization Problem:

The new joint-optimization problem, for  $w$  and  $\phi$ :

$\Theta$  is a set of parameters for  $\phi$

$$\hat{w} = \arg \min_{w \in \mathbb{R}^m, \Theta \in \mathbb{R}^{m \times d}} \left( \frac{1}{n} \sum_{i=1}^n l(w^\top \phi(x_i; \Theta), y_i) \right)$$

Where  $\phi(x, \Theta) = (\phi_1(x; \theta_1), \dots, \phi_m(x; \theta_m))$ .

$\theta_i$  is the  $i$ th row of  $\Theta$ , i.e.  $\theta_i := (\Theta)_{i,:}$

More compact, in terms of  $\Theta$ , which combines  $w, \phi$ :

$$\Theta^* := \arg \min_{\Theta} (L(\Theta; \mathcal{D})) = \arg \min_{\Theta} \left( \frac{1}{n} \sum_{i=1}^n l(\Theta; x_i, y_i) \right)$$

$\Theta$  may also encapsulate  $w, \phi$  for multiple layers, depending on definition

### 7.1 Definitions

#### D. Activation Function

We set  $\phi_i(x; \theta_i) = \psi(\theta_i^\top x)$ ,  $\psi$  is the activation function.

$\theta_i \in \mathbb{R}^d$ ,  $\psi : \mathbb{R} \rightarrow \mathbb{R}$

**Notation** More concisely,  $\phi(x; \Theta) = \psi(\Theta x)$

| Activation Function          | Definition                     |
|------------------------------|--------------------------------|
| Identity                     | $\psi(z) = z$                  |
| Sigmoid                      | $\psi(z) = \frac{1}{1+e^{-z}}$ |
| Hyperbolic tangent           | $\psi(z) = \tanh(z)$           |
| Rectified Linear Unit (ReLU) | $\psi(z) = \max(0, z)$         |

**Rmk.** **Derivative of Sigmoid**  $\sigma(x)' = \sigma(x) \cdot (1 - \sigma(x))$

## D. Artificial Neural Network

The output functions of the above problem take the form:

$$f(x; w, \theta) = \sum_{j=1}^m w_j \psi(\theta_j^\top x)$$

**Rmk.** Also called Multi-Layer Perceptron (MLP)

### What is happening here?

Explaining the calculation steps for such an  $f$  naturally leads to the common pictorial depiction of neural networks.

- (i)  $x = (x_1, \dots, x_n) \in \mathbb{R}^d$  (Input Vector)
- (ii)  $z = \Theta x$  (Linear transformation)
- (iii)  $h_i = \psi(z_i)$  (Activation function)
- (iv)  $f(x) = \sum_{j=1}^m w_j h_j$  (Output)

#### D. Hidden Layer $h = \psi(z)$

#### D. Bias Term $b \in \mathbb{R}^m$

Needed, as  $f$  might not pass through origin. Similar to using  $F_{\text{lin}}$  in regression, these can also be added by augmenting the input & hidden layers.

### Does this work at all?

Yes, for most functions this does work.

#### D. Sigmoidal Function

$$\sigma(t) \text{ s.t. } \begin{cases} \sigma : \mathbb{R} \rightarrow \mathbb{R} \\ \lim_{t \rightarrow \infty} = 1 \text{ and } \lim_{t \rightarrow -\infty} = 0 \end{cases}$$

### Th. Universal Approximation Theorem

$\hat{f}$ , that uniformly approximates  $f$ , exists and takes this form:

$$\hat{f}(x) = \mathbf{W}^{(2)} \psi(\mathbf{W}^{(1)} x + b)$$

$f : [0, 1]^d \rightarrow \mathbb{R}$  continuous,  $\psi$  sigmoidal

$\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}$ ,  $\mathbf{W}^{(2)} \in \mathbb{R}^{1 \times m}$ ,  $m \in \mathbb{N}$

Note how  $m$  could be very large.

$m$  can intuitively be understood as the "width" of the ANN

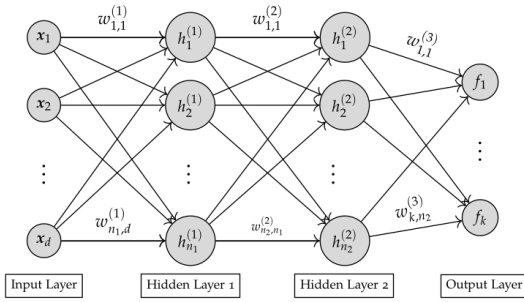
## D. Fully Connected Neural Network

More complex ANNs might have:

1. More hidden layers
2. Multiple outputs
3. Different activation functions across layers

These are called *fully connected*, since every node in a layer is connected to every node in the adjacent layers.

There are also more complex architectures.



*Introduction to Machine Learning (2026), p. 183*

**Notation** Weights:  $\mathbf{W}^{(i)} := [w_{k,l}^{(i)}]$ , Biases:  $b_k^{(i)}$   
and  $\Theta = (\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}, b^{(1)}, \dots, b^{(L)})$  (All parameters)

$w_{k,l}^{(i)}$ : "Weight at layer  $i$  to node  $k$  from node  $l$ "

## 7.2 Forward Propagation

How can we make predictions, i.e. how can  $\hat{f}$  be evaluated?

### D. Forward Propagation

This is just the computation for 1-layer ANN generalized for  $L$  layers

#### Algorithm 4: Forward Propagation

```

 $h^{(0)} \leftarrow x;$ 
for  $l = 1, \dots, L$  do
     $z^{(l)} = \mathbf{W}^{(l)} h^{(l-1)} + b^{(l)}$ 
     $h^{(l)} = \psi(z^{(l)})$ 
end
 $f \leftarrow \mathbf{W}^{(L)} h^{(L-1)} + b^{(L)}$ 
return  $f$ 

```

## 7.3 Backwards Propagation

How can we get all gradients needed for model training?

### D. Backwards Propagation

**Intuition:** An efficient way to get the gradients is to reuse results from forward prop. and previous steps. This works best when starting at the back, at  $\nabla_{\mathbf{W}^{(L)}} l$ .

**Goal:**  $\nabla_{\mathbf{W}^{(1)}} l, \dots, \nabla_{\mathbf{W}^{(L)}} l, \nabla_{b^{(1)}} l, \dots, \nabla_{b^{(L)}} l$

**Step 1:** Calculate  $\nabla_{\mathbf{W}^{(L)}} l$ , i.e. start from the back.

$$\begin{aligned}
 \nabla_{\mathbf{W}^{(L)}} l &= \frac{\partial l}{\partial \mathbf{W}^{(L)}} \\
 &= \frac{\partial l}{\partial f} \cdot \frac{\partial f}{\partial \mathbf{W}^{(L)}} \quad (\text{Chain Rule}) \\
 &= \frac{\partial l}{\partial f} \cdot \begin{bmatrix} (h^{(L-1)})^\top \\ \vdots \\ (h^{(L-1)})^\top \end{bmatrix} \quad (f = \mathbf{W}^{(L)} h^{(L-1)} + b^{(L)}) \\
 &= \nabla_f l \cdot \begin{bmatrix} (h^{(L-1)})^\top \\ \vdots \\ (h^{(L-1)})^\top \end{bmatrix} \quad \left( \frac{\partial l}{\partial f} = \nabla_f l \right)
 \end{aligned}$$

Notice how  $h^{(L-1)}$  was computed during forward prop.

**Step 2:** Calculate  $\nabla_{\mathbf{W}^{(L-1)}} l$ .

$$\nabla_{\mathbf{W}^{(L-1)}} l = \underbrace{\frac{\partial l}{\partial f}}_{(1)} \cdot \underbrace{\frac{\partial f}{\partial h^{(L-1)}}}_{(2)} \cdot \underbrace{\frac{\partial h^{(L-1)}}{\partial z^{(L-1)}}}_{(3)} \cdot \underbrace{\frac{z^{(L-1)}}{\partial \mathbf{W}^{(L-1)}}}_{(4)} \quad (\text{Chain Rule})$$

1. Already done in Step 1.
2. Already done in forward propagation, equal to  $\mathbf{W}^{(L)}$ :

$$f \stackrel{\text{def}}{=} \mathbf{W}^{(L)} h^{(L-1)} + b^{(L)} \implies \frac{\partial f}{\partial h^{(L-1)}} = \mathbf{W}^{(L)}$$

3. **Not done.** Needs to be calculated:

$$\begin{aligned}
 \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} &= \frac{\partial \psi(z^{(L-1)})}{\partial z^{(L-1)}} \\
 &= \text{diag}(\psi'(z^{(L-1)})) \\
 &= \begin{bmatrix} \psi'(z_1^{(L-1)}) & 0 & \dots & 0 \\ 0 & \psi'(z_2^{(L-1)}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \psi'(z_n^{(L-1)}) \end{bmatrix}
 \end{aligned}$$

4. Already done in forward propagation, analogous to step 1.

$$\frac{\partial z^{(L-1)}}{\partial \mathbf{W}^{(L-1)}} = \begin{bmatrix} (h^{(L-2)})^\top \\ \vdots \\ (h^{(L-2)})^\top \end{bmatrix}$$

**Step  $i \leq L$ :** Calculate  $\nabla_{\mathbf{W}^{(L-i)}} l$  Analogous to step 2.

The biases  $\nabla_{b^{(i)}} l$  are analogous.

## 7.4 Optimization

**Problem:** How can we train the model, i.e find  $\Theta^*$ ?

$$\Theta^* := \arg \min_{\Theta} (L(\Theta; \mathcal{D})) = \arg \min_{\Theta} \left( \frac{1}{n} \sum_{i=1}^n l(\Theta; x_i, y_i) \right)$$

**Rmk.**  $L(\Theta; \mathcal{D})$  is generally not convex.  
i.e. local minima, saddle points may exist

**Rmk.**  $\dim(\Theta)$  is the total param. count of NN, may be very large

**Solution:** Gradient Descent (with optimizations)

- Stochastic Gradient Descent  
(Why?  $\dim(\Theta)$  is very large,  $\nabla_{\Theta} l(\Theta; x_i, y_i)$  are expensive)
- Minibatch Gradient Descent  
(Why?  $\mathcal{D}$  may be very large, so there are *many* gradients)

The standard GD update for  $\Theta$  is:

$$\Theta^{t+1} = \Theta^t - \eta_t \cdot \nabla_{\Theta} L(\Theta; \mathcal{D})$$

In Minibatch GD, this becomes:

Where  $\mathcal{S} \subset \{1, \dots, n\}$

$$\Theta^{t+1} = \Theta^t - \eta_t \cdot \nabla_{\Theta^t} \left( \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} l(\Theta^t; x_i, y_i) \right)$$

**Rmk.** An advantage: If  $\Theta^t$  approaches a stat. point (which isn't the global minimum), GD will converge, but MB-GD may not converge.

The further subsections go into more details:

1. Preventing vanishing & exploding Gradients
2. Choice of initial  $w_i$
3. Choice of  $\mu_t$

### 7.4.1 Vanishing & Exploding Gradients

$$\nabla_{\Theta^t} \left( \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} l(\Theta^t; x_i, y_i) \right) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} (\nabla_{\Theta^t} l(\Theta^t; x_i, y_i))$$

The terms  $\nabla_{\Theta^t} l(\Theta^t; x_i, y_i)$  each contain  $\nabla_{\mathbf{W}^{(l)}} l(\mathbf{W}^{(l)}; x_i, y_i)$ .

**Problem:** Optimization might fail if:

$$\|\nabla_{\mathbf{W}^{(l)}} l\| \rightarrow \infty \quad \text{or} \quad \|\nabla_{\mathbf{W}^{(l)}} l\| \rightarrow 0$$

**Solution:**  $\|\nabla_{\mathbf{W}^{(l)}} l\|$  depends linearly on  $\text{diag}(\psi'(z^{(l)}))$ , so the choice of  $\psi$  ( $\psi'$ ) can be used to constrain  $\|\nabla_{\mathbf{W}^{(l)}} l\|$   
Generally, the gradient follows the behaviour of  $\psi'$

Which features do we want  $\psi$  ( $\psi'$ ) to fulfill?

- $\psi'$  should be fast to calculate
- $\psi'$  should be non-zero (and not get too close)

**Rmk.** The  $\|\nabla_{\mathbf{W}^{(l)}} l\|$  may still vanish for any  $\psi$ .

### 7.4.2 Random Weight Initialization

$$\begin{aligned} \nabla_{\mathbf{W}^{(l)}} l &= \frac{\partial l}{\partial f} \cdot \frac{\partial f}{\partial h^{(l)}} \cdot \frac{\partial h^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial \mathbf{W}^{(l)}} \\ &= \frac{\partial l}{\partial f} \cdot \frac{\partial f}{\partial h^{(l)}} \cdot \text{diag}(\psi'(z^{(l)})) \cdot \begin{bmatrix} (h^{(l-1)})^\top \\ \vdots \\ (h^{(l-1)})^\top \end{bmatrix} \end{aligned}$$

So, the gradient  $\|\nabla_{\mathbf{W}^{(l)}} l\|$  also depends on  $\|h^{(l-1)}\|$ .

For  $l \in \{1, \dots, L\}$

**Problem:** It might be that  $\|h^{(l-1)}\| \rightarrow \infty$  or  $\|h^{(l-1)}\| \rightarrow 0$ .

**Solution:** Set  $h^{(l-1)}$  randomly, bound mean  $\mu$  and var.  $\sigma^2$ .

There is no generally optimal bound for  $\sigma^2$ , it depends on the NN.

**Rmk.** Practical distributions for common  $\psi$ :

$n_{\text{out}}, n_{\text{in}}$  are the node counts of the layers adjacent to  $w_i$

| $\psi$ | Weights                                                                        |
|--------|--------------------------------------------------------------------------------|
| tanh   | $w_i \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}}\right)$                  |
| tanh   | $w_i \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right)$ |
| ReLU   | $w_i \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right)$                  |

### 7.4.3 Learning Rate & Weight Updates

$$\Theta^{t+1} = \Theta^t - \mu_t \cdot \nabla_{\Theta^t} \left( \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} l(\Theta^t; x_i, y_i) \right)$$

**Problem:** How to choose  $\mu$ ? (Learning Rate)

**Solution:** Heuristics.

There is no generally optimal  $\mu$ .

**Method** **Piecewise constant**  $\mu_t$

Intuitively, it makes sense to reduce  $\mu_t$  as optimization progresses, as the algorithm approaches the minimum.

Linear/cosine decay could also be used.

$$\mu_t = \begin{cases} 1 & 0 \leq t < 3 \\ 0.5 & 3 \leq t < 6 \\ 0.25 & 6 \leq t < 9 \\ \dots & \end{cases}$$

**Method** **Weight update indicator**

In practice, SGD often oscillates in finding the minimum. Then, a monotonic  $\mu_t$  doesn't make sense.

$$\frac{\|\nabla_{\Theta^t} L(\Theta^t; \mathcal{D})\|}{\|\Theta^t\|}$$

In general, if this indicator ratio is small, a higher learning rate makes sense (and vice versa).

Intuitively: How strongly is the weight change, relative to weight size.

**Method** **Momentum**

Combine the update direction with the previous update directions, for some weight  $m > 0$ , to stabilize.

## 7.5 Regularization

**Problem:** How can overfitting be avoided?

A few methods can be applied directly to SGD:

### Method Penalty Term

Similar to Ridge/LASSO Regression, a penalty term can be used, with some weight  $\lambda > 0$ .

$$\arg \min_{\Theta \in \mathbb{R}^d} (L(\Theta; \mathcal{D})) \rightarrow \arg \min_{\Theta \in \mathbb{R}^d} (L(\Theta; \mathcal{D}) + \lambda \|\Theta\|^2)$$

### Method Earlier stop

Choosing a different stop criterion for SGD, e.g. performance on the test set  $\mathcal{D}'$ .

### Rmk. Validation & Training Error

Overfitting occurs when the training error (on  $\mathcal{D}$ ) continues to fall, but the test error increases (on  $\mathcal{D}'$ ).



Introduction to Machine Learning (2026), p. 196

### 7.5.1 Dropout Regularization

This is a method specific to Neural Networks.

#### Method Dropout

Fix some  $p \in (0, 1)$ . For each SGD iteration in training: *Drop out* each hidden unit with probability  $1 - p$ , and skip their optimization for this iteration.

$$z_j^{(l)} = \sum_{i=0}^{n_{i-1}} (w_{j,i}^{(l)} h_i^{(l-1)}) + b_j^{(l)} \quad (\text{Regular Neuron})$$

$$z_j^{(l)} = \sum_{i=0}^{n_{i-1}} (w_{j,i}^{(l)} h_i^{(l-1)} \cdot \mathbb{I}_{C_i}) + b_j^{(l)} \quad (\text{With Dropout})$$

Where  $C_i := \{\text{"Unit } h_i^{(l-1)} \text{ is kept this iter."}\}$ , so  $\mathbb{P}[C_i] = p$

**Problem:** For  $\mathcal{D}'$ , we again want to use all layers.

**Solution:** Scale all weights with  $p$

For this, we use  $\mathbb{E}[z_j^{(l)}]$  instead of  $z_j^{(l)}$ .

$$\mathbb{E}[z_j^{(l)}] = (p \cdot w_j^{(l)})^\top \cdot h^{(l-1)} + b_j^{(l)}$$

By using  $\mathbb{E}[\mathbb{I}_{C_i}] = \mathbb{P}[C_i] = p$

### 7.5.2 Batch Normalization

During SGD, the weight's  $\sigma^2$  may again explode.

After few iterations,  $w_i$  may have changed completely from init.

**Problem:** Internal covariate shift.

The mean  $\mu$  deviates from 0 and  $\sigma^2$  might increase

**Solution:** Standardize  $\psi$  also during training.

#### Method Batch Normalization

In practice, only batches of  $\psi$  are normalized. The core idea is to set  $\mu \mapsto 0$  and  $\sigma^2 \mapsto 1$  within the batch.

This isn't optimal for all problems, and can be tweaked using  $\beta, \gamma$

The algorithm uses the parameters  $\beta, \gamma$  and buffers  $\mu_{\text{EMA}}, \sigma_{\text{EMA}}^2$ .  $\beta, \gamma$  are learnable and can also be optimized

#### Normalization Step (Training set)

For a minibatch  $\mathcal{S} = \{i_1, \dots, i_k\}$ , the batch is  $\{x_{i_1}, \dots, x_{i_k}\}$ .

**Step 1:** Find current values of  $\mu, \sigma^2$ .

$$\mu_{\mathcal{S}} := \frac{1}{|\mathcal{S}|} \sum_{j \in \mathcal{S}} x_j \quad (\text{minibatch mean})$$

$$\sigma_{\mathcal{S}}^2 := \frac{1}{|\mathcal{S}|} \sum_{j \in \mathcal{S}} (x_j - \mu_{\mathcal{S}})^2 \quad (\text{minibatch variance})$$

**Step 2:** Update the moving average:  $\mu_{\text{EMA}}, \sigma_{\text{EMA}}^2$ .

$$\mu_{\text{EMA}} = (1 - \alpha) \mu_{\text{EMA}} + \alpha \cdot \mu_{\mathcal{S}} \quad (\text{avg. mean update})$$

$$\sigma_{\text{EMA}}^2 = (1 - \alpha) \sigma_{\text{EMA}}^2 + \alpha \cdot \sigma_{\mathcal{S}}^2 \quad (\text{avg. variance update})$$

**Step 3:** Update the  $x_j$ .

$$\hat{x}_j = \frac{x_j - \mu_{\mathcal{S}}}{\sqrt{\sigma_{\mathcal{S}}^2 + \epsilon}} \quad (\text{point normalization})$$

$$\bar{x}_j = \gamma \cdot \hat{x}_j + \beta \quad (\text{scale \& shift})$$

#### Normalization Step (Test set)

Only apply step 3, now using the moving average values.

$$\hat{x}_j = \frac{x_j - \mu_{\text{EMA}}}{\sqrt{\sigma_{\text{EMA}}^2 + \epsilon}} \quad (\text{point normalization})$$

$$\bar{x}_j = \gamma \cdot \hat{x}_j + \beta \quad (\text{scale \& shift})$$

## 7.6 Convolutional Neural Networks

In fully connected NNs:

$$h^{(l)} = \psi(\mathbf{W}^{(l)} h^{(l-1)})$$

Each unit of layer  $l-1$  affects each unit of layer  $l$ .  
In CNNs, this is relaxed: not all nodes (must) interact.

### D. Convolutional Neural Network (CNN)

Layers are connected via convolutions.

$$h^{(l)} = \psi(w^{(l)} * h^{(l-1)})$$

**Rmk.** In CNNs, the weights are also called *filters*.

### D. Convolution (Discrete, 2D)

$w \in \mathbb{R}^k$ ,  $x \in \mathbb{R}^d$

$$w * x := \sum_{j=\max\{1, i-d+1\}}^{\min\{i, k\}} \left( w_j \cdot x_{i-j+1} \right)$$

Understanding this is easier by example:

**Example:**  $w = (w_1, w_2)^\top$ ,  $x = (x_1, x_2, x_3)^\top$

$$w * x = \begin{bmatrix} w_1 \cdot x_1 + w_2 \cdot 0 \\ w_1 \cdot x_2 + w_2 \cdot x_1 \\ w_1 \cdot x_3 + w_2 \cdot x_2 \\ w_1 \cdot 0 + w_2 \cdot x_3 \end{bmatrix}$$

**Example:** a CNN with 3 inputs and 1 hidden layer:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \underbrace{\begin{bmatrix} w_1 & 0 & 0 \\ w_2 & w_1 & 0 \\ 0 & w_2 & w_1 \\ 0 & 0 & w_2 \end{bmatrix}}_{\mathbf{W}^{(1)} \text{ in CNN}} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = w * x$$

## 7.7 Network sizes

**Example:** CNN vs FCNN (MLP) Parameter counts.

We can model an RGB Image as  $\mathbf{I} \in \mathbb{R}^{1920 \times 1080 \times 3}$  where  $(\mathbf{I}_{i,j,1}, \mathbf{I}_{i,j,2}, \mathbf{I}_{i,j,3})$  are the RGB values of pixel  $(i, j)$ . We use a CNN & FCNN to analyze.

For an FCNN  $N$  with 1 hidden layer with  $h$  nodes, and  $o$  output nodes:

$$\text{size}(N) = \underbrace{(1920 \cdot 1080 \cdot h)}_{\text{All-to-All}} + \underbrace{(h \cdot o)}_{\text{Output}} + \underbrace{(h + o)}_{\text{Bias}}$$

For a CNN  $N'$  with an  $n \times n$  Filter, padding  $p$  and stride  $s$ :

$$\text{size}(N') = \left( \frac{1920 + 2 \cdot p - n}{s} + 1 \right) \cdot \left( \frac{1080 + 2 \cdot p - n}{s} + 1 \right)$$

The advantage of CNNs becomes clear when plugging in values, e.g.

$$N : h = 256, o = 10 \quad N' : n = 4, p = 2, s = 2$$

$$\text{size}(N) = 1'592'527'626 \quad \text{size}(N') = 519'901$$

### 7.7.1 Multidimensional Convolution

**TODO** add explanation

## 8 Unsupervised Learning

In **Unsupervised Learning**,  $\mathcal{D}$  contains no labels. Models both define labels & assign inputs to labels.

$$\mathcal{D} = \{x_1, \dots, x_n\} \quad (\text{Dataset in unsupervised learning})$$

There are many use-cases:

1. Compression
2. Discovery of latent variables
3. Anomaly detection
4. Exploratory data analysis

### 8.1 Clustering

#### D. Clustering

The goal here is to group inputs into clusters, based on some definition of similarity, e.g.  $l_2$  distance for  $\mathcal{D} \subset \mathbb{R}^2$ .

This can be seen as the unsupervised analogy to classification

#### 8.1.1 Basic Methods

##### Method Hierarchical Clustering

A simple method, using the "similarity" measure directly.

1. Each  $x \in \mathcal{D}$  starts in its own cluster
2. Iteratively, the 2 "closest" clusters are merged

This results in a tree, thus *hierarchical* clustering.

##### Method Partitioning

In Partitioning methods, a weighted graph is constructed using  $\mathcal{D}$  and partitioned using graph theory approaches, i.e. using cuts or spectral analysis.

**Rmk.** Both Hierarchical and Partitioning do not give a natural way to deduce cluster membership for new datapoints.

### 8.1.2 $k$ -Means Clustering

In  $k$ -means, a cluster is represented by its center:  $\mu_j \in \mathbb{R}^d$ .  
The cluster assignment  $z_i$  for  $x_i \in \mathcal{D}$ :

$$z_i = \arg \min_{j=1,\dots,k} \|x_i - \mu_j\| \quad (\text{Closest center})$$

**Rmk.** This strategy induces a partition of  $\mathbb{R}^d$ . (Voronoi Pattern)

**Problem:** How to find  $\mu = (\mu_1, \dots, \mu_k)^\top$ ?

A new optimization objective:

$$\hat{R}(\mu) = \sum_{i=1}^n \min_{j \in \{1, \dots, k\}} \|x_i - \mu_j\|^2 = \sum_{i=1}^n \|x_i - \mu_{z_i}\|^2$$

(minimize the sum of sq. distances between points & their centers)

**Rmk.**  $\|\cdot\|_2$  corresponds to the *mean*.  $\|\cdot\|_1$  would use the *median*.

So we are searching: (non-convex & NP-hard)

$$\arg \min_{\mu} \left( \hat{R}(\mu) \right) \quad (\text{optimal } k\text{-means cluster})$$

#### Method Lloyd's Heuristic

This is an iterative method to find the cluster centers.

$$\text{D. } z^{(t)} = (z_1^{(t)}, \dots, z_n^{(t)})^\top \quad (\text{assignment of } x_i \text{ at iter. } t)$$

$$\text{D. } \mu^{(t)} = (\mu_1^{(t)}, \dots, \mu_k^{(t)})^\top \quad (\text{Cluster centers at iter. } t)$$

$$\text{D. } n_j^{(t)} = \left| \left\{ i = 1, \dots, n \mid z_j^{(t)} = j \right\} \right| \quad (\text{Size of cluster } j \text{ at iter. } t)$$

---

#### Algorithm 5: Lloyd's Heuristic

---

$$\mu^{(0)} \leftarrow (\mu_1^{(0)}, \dots, \mu_k^{(0)});$$

**repeat**

$$z_i^{(t)} \leftarrow \arg \min_{j \in \{1, \dots, k\}} \|x_i - \mu_j^{(t-1)}\| \quad \text{for } i = 1, \dots, n;$$

$$\mu_j^{(t)} \leftarrow \frac{1}{n_j^{(t)}} \sum_{i \text{ s.t. } z_i^{(t)} = j} x_i \quad \text{for } j = 1, \dots, k;$$

$$t \leftarrow t + 1;$$

**until convergence;**

---

**Rmk.** Each iteration is in  $\mathcal{O}(nkd)$ .

**Example:** Consider  $x_{1,2,3} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} -2 \\ -4 \end{bmatrix}$  for which we'd like  
 $k = 2$  clusters. We choose:

$$\mu_1^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \mu_2^{(0)} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

For Lloyd's Heuristic, the initial cluster assignments  $z_{1,2,3}^{(1)}$ :

$$z_1^{(1)} = \arg \min_{j \in \{1,2\}} \left( \left\| \begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|, \left\| \begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\| \right) = 1$$

$$z_2^{(1)} = \arg \min_{j \in \{1,2\}} \left( \left\| \begin{bmatrix} -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|, \left\| \begin{bmatrix} -1 \\ -1 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\| \right) = 2$$

$$z_3^{(1)} = \arg \min_{j \in \{1,2\}} \left( \left\| \begin{bmatrix} -2 \\ -4 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|, \left\| \begin{bmatrix} -2 \\ -4 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\| \right) = 2$$

And then the updated cluster centers:

$$\mu_1^{(1)} = \frac{1}{1} \left( \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\mu_2^{(1)} = \frac{1}{2} \left( \begin{bmatrix} -1 \\ -1 \end{bmatrix} + \begin{bmatrix} -2 \\ -4 \end{bmatrix} \right) = \frac{1}{2} \begin{bmatrix} -3 \\ -5 \end{bmatrix}$$

### 8.1.3 Convergence

$k$ -Means is guaranteed to converge to a local optimum:

**Th.** **Monotonically decreasing convergence**

$\forall t \geq 1$ :

$$\hat{R}(\mu^{(t)}, z^{(t)}) \geq \hat{R}(\mu^{(t+1)}, z^{(t+1)})$$

**Rmk.** For the global optimum, the initialization is critical.

**Rmk.**  $k$ -Means may produce bad results for non-spherical clusters.

(A consequence of using  $\|\cdot\|_2$ , kernels can overcome this)

### 8.1.4 initialization

**Problem:** How to choose  $\mu^{(0)} = (\mu_1^{(0)}, \dots, \mu_k^{(0)})$ ?

**Solution:** Heuristics.

A simple approach is sampling uniformly from  $\mathcal{D} = \{x_1, \dots, x_n\}$ .

However, This is problematic for unbalanced cluster sizes.

The chance that small clusters receive no initial  $\mu_i^{(0)}$  is high.

#### Method Furthest Point Heuristic

Select  $\mu_0^{(0)}$  randomly, then iteratively maximize distance to the nearest cluster center for subsequent  $\mu_{i \geq 1}^{(0)}$ .

#### Method $k$ -means++

More robust heuristic: more random factors against outliers.

**Step 1:** Pick  $\mu_0^{(0)}$  randomly.

$$\mu_0^{(0)} = x_i \in \mathcal{D}, \quad i \sim \mathcal{U}(\{1, \dots, n\})$$

**Step 2:** Pick  $\mu_{2, \dots, k}^{(0)}$  using this rule.

$$\mu_j^{(0)} = x_i \in \mathcal{D}, \quad i \sim p(i) \propto \min_{1 \leq m \leq j-1} \|x - \mu_m\|^2$$

**Th.**  $k$ -means++ is optimal up to  $\mathcal{O}(\log(k))$

$$\hat{R}(\mu_{k\text{-means}++}) \leq \mathcal{O}(\log(k)) \cdot \min_{\mu} \hat{R}(\mu)$$

### 8.1.5 Choosing $k$

**Problem:** How to choose  $k$ ?

**Rmk.** Unfortunately, cross-validation can't be used: Both the training & test loss will decrease as  $k$  increases, so the loss provides no good stopping criterion.

**Method** Increase  $k$  until  $\hat{R}$  yields diminishing returns.

Usually, plotting  $k$  against  $\hat{R}$  yields something like exp decay.

**Method** Penalize higher model complexity.

weight  $\lambda > 0$  is generally easier to choose than  $k$  directly.

$$\hat{R}' = \hat{R}(\mu) + \lambda \cdot k$$

There are several other methods to do this, based e.g. on concepts from information theory.

## 8.2 Principal Component Analysis

**Motivation:** For  $\mathcal{D} = \{x_i\}_{i=1}^n, x_i \in \mathbb{R}^d$ , we'd like a low-dimensional representation with  $k \ll d$ , which we call *embeddings*  $\{z_i\}_{i=1}^n, z_i \in \mathbb{R}^k$ .

e.g. for performance, memory, noise removal, visualizations...

**Intuition:** High dimensional data usually has many redundancies & highly correlated features. Dimensionality Reduction in practice preserves most substantial data. This idea is formalized as the *Manifold Hypothesis*.

**Rmk. Requirement:** We assume  $\mathbf{X}$  is centered. For general  $\mathbf{X}$ , we therefore use:

$$\bar{\mathbf{X}} = \mathbf{X} - \mathbf{I}\mu \quad \mu = \sum_{i=1}^n x_i$$

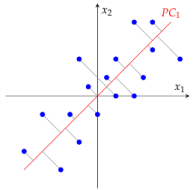
i.e. we subtract the mean  $\mu$  of all  $x_i$  from each  $x_i$ .

### 8.2.1 PCA in one dimension

**Method PCA**  $k = 1$

Find  $w^* \in \mathbb{R}^d$  and  $z_1^*, \dots, z_n^* \in \mathbb{R}$  for dimensionality reduction  $d = 1$ :

$$w^*, z_1^*, \dots, z_n^* = \min_{w \in \mathbb{R}^d: \|w\|_2=1} \sum_{i=1}^n \underbrace{\|x_i - \underbrace{z_i^* w}_{\approx x}\|}_{\text{Reconstruction Error}}$$



Introduction to Machine Learning (2026), p. 223

**L. PCA**  $k = 1$  (**Variance Matrix**)

Alternative formulation.

$$w^* = \arg \max_{\|w\|_2=1} w^\top \Sigma w$$

**Intuition:**  $w$  can be thought of as the subspace (a line for  $d = 1$ ) we want to project onto. The optimal  $w^*$  aligns with the direction maximizing the *empirical variance*  $\Sigma$  of the projected data.

**D. Empirical Covariance**  $\Sigma = \frac{1}{n} \sum_{i=1}^n x_i x_i^\top = \frac{1}{n} X^\top X$

**L. Eigendecomposition**  $\Sigma = \sum_{i=1}^d \lambda_i v_i v_i^\top$

Eigenvalues  $\lambda_1 > \dots > \lambda_d > 0$ ,

eigenvectors  $v_i$  forming an orthonormal Basis of  $\mathbb{R}^d$

**Th. PCA Solution for**  $k = 1$

$v$  is the Eigenvector associated with the largest  $\lambda$  for  $\Sigma$ .

$$w^* = v_1 \quad \text{solves} \quad w^* = \arg \max_{\|w\|_2=1} w^\top \Sigma w$$

### 8.2.2 Generalized PCA

Luckily, the case  $k = 1$  generalizes:

**Th. PCA General Solution**

Here,  $\mathbf{W} \in \mathbb{R}^{d \times k}$  and  $z_i^* \in \mathbb{R}^k$  for parameter  $k$

$$\mathbf{W}^*, z_1^*, \dots, z_n^* = \arg \min_{\mathbf{W} \in \mathbb{R}^{d \times k}: \mathbf{W}^\top \mathbf{W} = \mathbf{I}} \sum_{i=1}^n \|x_i - \mathbf{W} z_i^*\|^2$$

is solved using

$$\mathbf{W}^* = (v_1 \parallel \dots \parallel v_k), \quad z_i^* = \mathbf{W}^{*\top} x_i$$

**Intuition:** In words, the basis for the  $k$ -dim. subspace minimizing reconstruction error is given using the first  $k$  eigenvectors of  $\Sigma$ .

**L. PCA is an orthohgonal projection**

$$P: \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad x \mapsto \mathbf{W}^* \mathbf{W}^{*\top} x$$

**L. Connection to SVD**

The First  $k$  EV of  $\Sigma$  are the first  $k$  right singular vectors of  $\mathbf{X}$ .

$$\mathbf{X} = \mathbf{U} \mathbf{S} \mathbf{V}^\top \quad n \cdot \Sigma = \mathbf{V} \mathbf{S}^\top \mathbf{S} \mathbf{V}^\top$$

**Rmk.** Non-linear methods and kernels can be applied to PCA the same way they are applied to regression.

### 8.2.3 Connection to k-Means

k-Means (Clustering) can be reformulated:  $z_1, \dots, z_n \in E_k$  where  $E_k = \{e_1, \dots, e_n\}$  holds unit vectors  $e_i$ .

$$\mathbf{W}^*, z_1^*, \dots, z_n^* = \arg \min_{\mathbf{W} \in \mathbb{R}^{d \times k}: \mathbf{W}^\top \mathbf{W} = \mathbf{I}} \sum_{i=1}^n \|x_i - \mathbf{W} z_i^*\|^2$$

This is the same as PCA, only the space for  $z^*$  changes. Both PCA and k-Means are special forms of a more general technique:

**D. Matrix Factorization Methods**

General class of techniques for decomp. of  $\mathbf{X}$

$$\mathbf{X} \approx \mathbf{W} \mathbf{Z}$$

**Rmk. PCA.** For PCA,  $\mathbf{W}$  is an orthonormal Basis of the optimal  $k$ -dimensional subspace,  $\mathbf{Z}$  are the coefficients for the projection.

**Rmk. k-Means.** For k-means,  $\mathbf{W}$  contains cluster centroids and  $\mathbf{Z}$  the cluster assignments.

## 9 Probabilistic Modelling

A different approach: Try to learn  $\mathbb{P}^*$  directly.

$\mathbb{P}^*$  is the data-generating distribution

Some terminology:

|                                        |                                             |
|----------------------------------------|---------------------------------------------|
| $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ | Dataset, sampled i.i.d. from $\mathbb{P}^*$ |
| $\mathbb{P}^*$                         | Data-generating distribution                |
| $\mathcal{P}$                          | Family of potential distributions           |
| $\hat{\mathbb{P}} \in \mathcal{P}$     | Optimal model of $\mathbb{P}^*$             |

Some advantages & applications:

1. Allows assumptions about data-generating process  
e.g. what is the likelihood of sampling  $\mathcal{D}$  ?
2. Understand why some methods work  
e.g. on which distributions does the square loss work?
3. Encode prior knowledge into the model
4. Quantify uncertainty of predictions
5. Develop new decision rules
6. Generate entirely new samples

### 9.1 Assumptions

**Assumption 1:** We assume  $\mathcal{D}$  is i.i.d. sampled from  $\mathbb{P}_{X,Y}^*$ . Thus:

$$\mathbb{P}_{\mathcal{D}} = \prod_{i=1}^n \mathbb{P}_{X_i, Y_i} \quad (\text{Independence})$$

**Rmk.** i.i.d. is a strong assumption: often false in practice.  
e.g. sampling with temporal/spatial dependencies, bias, etc.

#### Method General-purpose Estimators

Methods that make no further assumptions, e.g. Histograms or Kernel Density Estimation (KDE).

Generally require large  $\mathcal{D}$  to be accurate, thus discouraged

**Assumption 2:**  $\mathbb{P}^* \in \mathcal{P}$  (some family of param. models  $\mathcal{P}$ )

#### D. Parametric family of distributions

$\theta \in \Theta \subset \mathbb{R}^p$  fully describes the distribution  $\mathbb{P}^\theta$

$$\mathcal{P} = \{\mathbb{P}^\theta \mid \theta \in \Theta\}$$

The art here, is to choose  $\mathcal{P}$  s.t.  $\mathbb{P}^* \in \mathcal{P}$  is likely. Then:

$$\exists \theta^* \in \Theta : \mathbb{P}^* = \mathbb{P}^{\theta^*} \in \mathcal{P}$$

$\theta \mapsto \mathbb{P}^\theta$  is assumed to be continuous. The advantage of this is that, if  $\theta$  is close to  $\theta^*$ , then  $\mathbb{P}^\theta$  is close to  $\mathbb{P}^{\theta^*} = \mathbb{P}^*$ .

## 9.2 Statistical Inference

**Problem:** How to choose  $\hat{\mathbb{P}}$  from  $\mathcal{P}$ , s.t.  $\hat{\mathbb{P}}$  is close to  $\mathbb{P}^*$ ?

If  $\mathcal{P}$  is parametric, this is the same as looking for  $\hat{\theta} \in \Theta$  close to  $\theta^*$

**Notation** if  $Z$  has  $\mathbb{P}_Z \in \mathcal{P} = \{\mathbb{P}_Z^\theta \mid \theta \in \Theta\}$

$$p(z; \theta) = p_Z^\theta(z)$$

**Notation** In the Bayesian context, where  $\theta^*$  is sampled from  $\mathbb{P}_\theta$ :

$$\begin{aligned} p(\theta) &= p_{\theta^*}(\theta) \\ p(z \mid \theta) &= p_{Z \mid \theta^* = \theta}(z) \\ p(\theta \mid z) &= p_{\theta^* \mid Z=z} \end{aligned}$$

Where  $p$  is either a density or mass function.

There are 2 paradigms:

1. **Frequentist:** Model only using observed data
2. **Bayesian:** Model also using prior beliefs

### 9.2.1 Bayesian Paradigm

**Further Assumption:**  $\theta^*$  is sampled from a distribution  $\mathbb{P}_{\theta^*}$

Note how  $\mathbb{P}_{\theta^*} \neq \mathbb{P}^{\theta^*}$ .

**Th. Bayes' Theorem** (Applied to Inference)

$$\underbrace{p(\theta \mid \mathcal{D})}_{\text{Posterior Belief}} = \underbrace{\frac{p(\mathcal{D} \mid \theta)}{p(\mathcal{D})}}_{\text{Update}} \cdot \underbrace{p(\theta)}_{\text{Prior Belief}}$$

$$p(\mathcal{D}) = \int p(\mathcal{D} \mid \theta) \cdot p(\theta) d\theta$$

### 9.2.2 Maximum Likelihood Estimator (MLE)

Frequentist Approach:  $\theta^*$  is considered fixed a priori.

**Method Maximum Likelihood Estimator**

Finds  $\hat{\theta}_{\text{MLE}}$ , which maximizes chance of observing  $\mathcal{D}$  over the possible distributions  $\mathcal{P} = \{\mathbb{P}_{X,Y}^\theta \mid \theta \in \Theta\}$ .

#### D. Maximum Likelihood Estimator

Corresponding to  $\hat{\mathbb{P}}_{X,Y} = \mathbb{P}_{X,Y}^{\hat{\theta}_{\text{MLE}}}$

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta \in \Theta} p(\mathcal{D}; \theta) \stackrel{\text{i.i.d.}}{=} \arg \max_{\theta \in \Theta} \prod_{i=1}^n p(x_i, y_i; \theta)$$

**Rmk.** Since log is strictly mon. increasing, the maximizer of the log-likelihood also maximizes the likelihood.

Applying several transformations:

$$\begin{aligned} \hat{\theta}_{\text{MLE}} &= \arg \max_{\theta \in \Theta} \prod_{i=1}^n p(x_i, y_i; \theta) \\ &= \arg \max_{\theta \in \Theta} \log \left( \prod_{i=1}^n p(x_i, y_i; \theta) \right) \\ &= \arg \max_{\theta \in \Theta} \sum_{i=1}^n \log(p(x_i, y_i; \theta)) \\ &= \underbrace{\arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(p(x_i, y_i; \theta))}_{\text{Generative Model}} \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(p(y_i \mid x_i; \theta) \cdot p(x_i)) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(p(y_i \mid x_i; \theta)) + \underbrace{\sum_{i=1}^n -\log(p(x_i))}_{\text{Indep. from } \theta} \\ &= \underbrace{\arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(p(y_i \mid x_i; \theta))}_{\text{Discriminative Model}} \end{aligned}$$

This has turned into an optimization problem. 2 approaches:

1. Analytically: insert  $p(x_i, y_i; \theta)$  or  $p(y_i \mid x_i; \theta)$ .  
There are closed-form expressions in this statistical model.

2. Numerically: Gradient Descent

**Rmk.** MLE is useful: It can be shown to converge to  $\theta^*$ .

### 9.2.3 Maximum A Posteriori Estimator (MAP)

Bayesian Approach:  $\theta^*$  is considered a random variable.

#### Method Maximum A Posteriori Estimator

Finds  $\hat{\theta}_{\text{MAP}}$ , which maximizes post. belief  $p(\theta | \mathcal{D})$ , i.e. it finds the  $\theta \in \Theta$  with the highest density *after* obtaining  $\mathcal{D}$ .

#### D. Maximum A Posteriori Estimator

Corresponding to  $\hat{\mathbb{P}}_{X,Y} = \mathbb{P}_{X,Y}^{\hat{\theta}_{\text{MAP}}}$

$$\hat{\theta}_{\text{MAP}} = \arg \max_{\theta \in \Theta} p(\theta | \mathcal{D})$$

Applying several transformations:

$$\begin{aligned} \hat{\theta}_{\text{MAP}} &= \arg \max_{\theta \in \Theta} p(\theta | \mathcal{D}) \\ &= \arg \max_{\theta \in \Theta} p(\mathcal{D} | \theta) \cdot p(\theta) \\ &\stackrel{\text{i.i.d.}}{=} \arg \max_{\theta \in \Theta} \underbrace{\left( \prod_{i=1}^n p(x_i, y_i | \theta) \right)}_{\text{Generative Model}} \cdot p(\theta) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(p(x_i, y_i | \theta)) - \log(p(\theta)) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(p(y_i | x_i, \theta)) \cdot p(x_i | \theta) - \log(p(\theta)) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(p(y_i | x_i, \theta)) + \underbrace{\sum_{i=1}^n -\log(p(x_i)) - \log(p(\theta))}_{\text{Indep. from } \theta} \\ &= \arg \min_{\theta \in \Theta} \underbrace{\sum_{i=1}^n -\log(p(y_i | x_i, \theta))}_{\text{Discriminative Model}} - \log(p(\theta)) \end{aligned}$$

**Rmk.** Intuitively, we can use  $p(\theta)$  as a weight for  $\theta$ , which can be used to introduce prior assumptions.

#### L. MAP without prior knowledge is MLE

Assume  $\mathbb{P}_{\theta^*} = \mathcal{U}(\Theta)$

$$\hat{\theta}_{\text{MAP}} = \arg \max_{\theta \in \Theta} \prod_{i=1}^n p(x_i, y_i | \theta) = \arg \max_{\theta \in \Theta} \prod_{i=1}^n p(x_i, y_i; \theta) = \hat{\theta}_{\text{MLE}}$$

Since  $p(\theta)$  can thus be eliminated

## 9.3 Bayes Optimal Predictor

What can we do once we have  $\hat{\mathbb{P}}$ ? We can estimate  $\mathbb{P}_{\mathcal{Y} | X=x}$  and thus derive a decision rule  $f^*(x)$ .

#### D. Bayes' Optimal Predictor

Best possible predictor when knowing  $\mathbb{P}_{\mathcal{Y}|X}$

$$f^*(x) = \arg \min_{a \in \mathcal{Y}} \mathbb{E} [l(a, \mathcal{Y}) | X = x] = \arg \min_{a \in \mathcal{Y}} \int p(y | x) \cdot l(a, y) dy$$

**Rmk.** In practice,  $\mathbb{P}_{\mathcal{Y}|X}$  is unknown, so  $\hat{\mathbb{P}}_{\mathcal{Y}|X}$  is used.

This is the theoretically best possible predictor when knowing  $\mathbb{P}_{\mathcal{Y}|X}$ , an optimal solution to supervised learning over all  $F$ :

$$\hat{f} = \arg \min_{f \in F} \sum_{i=1}^n l(f(x_i), y)$$

The conceptual difference between  $\hat{f}$  and  $f^*$  is that  $\hat{f}$  minimizes the *total* loss on  $\{x_1, \dots, x_n\}$  over a func. class  $F$ , while  $f^*$  minimizes the expected loss for each  $x_i$  separately, unconstrained by any  $F$ .

## 9.4 Probabilistic Perspective: Regression

How do we define a discriminative statistical model and use MLE/MAP for regression tasks?

**Problem:** We want to find  $\mathbb{P}_{\mathcal{Y}|X}^\theta$

**Assumption:** Outputs  $y$  have additive noise  $\epsilon$ :

$\epsilon$  indep. of  $x$ ,  $f \in F$  for some func. class

$$y = f(x; \theta^*) + \epsilon$$

#### D. Gaussian Noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$

This is the most common choice of  $\epsilon$

**Rmk.** Normally,  $\sigma^2$  is assumed to be known. If  $\sigma^2$  is unknown, we optimize for  $\theta = (\gamma, \sigma)$  s.t.  $\gamma$  are the parameters needed for  $f$ .

For Gaussian Noise, we can model  $\mathbb{P}_{X|Y}^\theta$  like this:

$$\begin{aligned} Y | X = x &\sim \mathcal{N}(f(x; \theta), \sigma^2) \\ p(y | x; \theta) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} (y_i - f(x_i; \theta))^2\right) \end{aligned}$$

### 9.4.1 Regression with MLE

**Solution:** Find  $\hat{\theta}_{\text{MLE}}$  for  $\mathbb{P}_{X|Y}^{\hat{\theta}_{\text{MLE}}}$ .

$$\begin{aligned} \hat{\theta}_{\text{MLE}} &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(p(y_i | x_i; \theta)) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} (y_i - f(x_i; \theta))^2\right)\right) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \underbrace{\left(\frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} (y_i - f(x_i; \theta))^2\right)}_{\text{Indep. from } \theta} \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n (y_i - f(x_i; \theta))^2 \end{aligned}$$

MLE using data with i.i.d. Gaussian Noise & constant, known  $\sigma^2$  is equivalent to minimizing the square loss. If we consider only linear  $f$ , this is equivalent to linear regression.

**Rmk.** Defining  $y_i = f(x_i; \theta) + \epsilon_i$  with  $\epsilon_i \sim \mathcal{N}(0, \sigma_i^2)$  (Different  $\sigma_i^2$  per  $\epsilon_i$ ) corresponds to weighted square loss: Higher  $\sigma_i^2$  means lower weight.

**Rmk.** Similar equivalences can be found for other losses too, using other distributions for  $\epsilon$ .

In general: the choice of likelihood function implicitly corresponds to the choice of loss function. So: Losses can now be chosen based on assumptions about the noise.

## 9.4.2 Regression with MAP

**Solution:** Find  $\hat{\theta}_{\text{MAP}}$  for  $\mathbb{P}_{X|Y}^{\hat{\theta}_{\text{MAP}}}$ .

$\theta^*$  is now considered a random variable, so we need a prior.

**D. Gaussian Prior**  $\theta^* \sim \mathcal{N}(0, \sigma_\theta^2 \cdot \mathbf{I}_d)$

$$p(\theta) = \frac{1}{(2\pi\sigma_\theta^2)^{\frac{d}{2}}} \exp\left(-\frac{1}{2\sigma_\theta^2} \|\theta\|^2\right)$$

Now applying the definition of the discriminative  $\hat{\theta}_{\text{MAP}}$ :

$$\begin{aligned} \hat{\theta}_{\text{MAP}} &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \left( -\log(p(y_i | x_i, \theta)) - \log(p(\theta)) \right) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \left( \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} (y_i - f(x_i, \theta))^2 \right) + \frac{d}{2} \log(\pi\sigma_\theta^2) + \frac{1}{2\sigma_\theta^2} \|\theta\|^2 \text{ So we get:} \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \frac{1}{2\sigma^2} (y_i - f(x_i, \theta))^2 + \frac{1}{2\sigma_\theta^2} \|\theta\|^2 \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n (y_i - f(x_i, \theta))^2 + \frac{\sigma^2}{\sigma_\theta^2} \|\theta\|^2 \end{aligned}$$

MLA using data with i.i.d. Gaussian Noise and Gaussian prior on  $\theta_i$  is equivalent to  $l_2$ -regularized regression ( $\lambda = \frac{\sigma^2}{\sigma_\theta^2}$ ) with square loss. If  $f$  is linear, this is ridge regression.

**Rmk.**  $\lambda$  implicitly corresponding to  $\frac{\sigma^2}{\sigma_\theta^2}$  justifies the idea of  $\lambda$  regularizing model complexity. High  $\lambda = \frac{\sigma^2}{\sigma_\theta^2}$  means the data is assumed to be noisy, and vice versa.

**D. Laplace Prior**  $\theta_i^* \sim \text{Laplace}(0, b)$

$$p(\theta_i) = \frac{1}{2b} \exp\left(-\frac{|\theta_i|}{b}\right)$$

$$\begin{aligned} \hat{\theta}_{\text{MAP}} &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \left( -\log(p(y_i | x_i, \theta)) - \log(p(\theta)) \right) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \left( \frac{1}{2\sigma^2} (y_i - f(x_i, \theta))^2 + \frac{1}{b} \|\theta\|_1 \right) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n (y_i - f(x_i, \theta))^2 + \frac{2\sigma^2}{b} \|\theta\|_1 \end{aligned}$$

MLA using the Laplace prior instead is equivalent to  $l_1$ -regularized regression ( $\lambda = \frac{2\sigma^2}{b}$ ) with square loss. If  $f$  is linear, this is LASSO regression.

## 9.4.3 Bayes Optimal Regression: Square Loss

**Problem:** What's an optimal decision rule for square loss?

$$f^*(x) = \arg \min_{a \in \mathcal{Y}} \mathbb{E}[l(a, Y) | X = x] \quad \text{s.t.} \quad l(f(x), y) = (f(x) - y)^2$$

**Solution:** Minimize analytically for  $a$ :

$$\begin{aligned} &\frac{\partial}{\partial a} \mathbb{E}[l(a, Y) | X = x] \\ &= \frac{\partial}{\partial a} \left( a^2 - 2a\mathbb{E}[Y | X = x] + \mathbb{E}[Y^2 | X = x] \right) \\ &= 2a - 2\mathbb{E}[Y | X = x] \\ &\stackrel{!}{=} 0 \implies a = \mathbb{E}[Y | X = x] \end{aligned}$$

$$f^*(x) = \arg \min_{a \in \mathcal{Y}} \mathbb{E}[l(a, Y) | X = x] = \underbrace{\mathbb{E}[Y | X = x]}_{\text{Mean of } \mathbb{P}_{Y|X=x}}$$

The Bayes opt. pred. for square loss is the conditional mean.

Intuitively, this makes sense since square loss penalizes larger deviations more.

## 9.5 Probabilistic Perspective: Classification

There are 2 approaches:

- Discriminative Modelling:** Model  $Y | X$  only:  $\mathbb{P}_{X|Y}$   
Explicitly model  $\mathbb{P}_{X|Y}$ , modelling  $\mathbb{P}_X$  is optional
- Generative Modelling:** Model the joint distr.  $\mathbb{P}_{X,Y}$   
Explicitly model both  $\mathbb{P}_{X|Y}$  and  $\mathbb{P}_Y$

Both can be used to model the joint distribution. Depending on the problem, one factorization may be more natural.

$$\mathbb{P}_{X,Y} = \underbrace{\mathbb{P}_{Y|X} \cdot \mathbb{P}_X}_{\text{discriminative}} = \underbrace{\mathbb{P}_{X|Y} \cdot \mathbb{P}_Y}_{\text{generative}}$$

**Rmk.** Discriminative M. is good if only predicting  $y$  for  $x$  is required.

**Rmk.** Generative M. is good if new  $x$  for a given  $y$  are required.

### 9.5.1 Classification: Discriminative

We first need a distribution, then we can use MLE/MAP.

**Problem:** Modelling binary Classification:  $y \in \{+1, -1\}$

**Assumption:** Labels are generated via some  $f(\cdot, \theta^*) \in F$ .  
 $f$  is called *discriminator*.

**Solution:** We choose a logistic model, using  $\sigma(z)$ :  
 $\sigma(z) = \frac{1}{1+\exp(-z)}$  (Logistic function)

$$Y | X = x \sim \text{Ber}\left(\sigma(f(x; \theta))\right)$$

Using  $p(x)$  for  $\text{Ber}(p)$ :

Note  $1 - \sigma(z) = \sigma(-z)$

$$\begin{aligned} p(y | x; \theta) &= \begin{cases} \sigma(f(x; \theta)), & y = +1 \\ 1 - \sigma(f(x; \theta)) & y = -1 \end{cases} \\ &= \sigma(y \cdot f(x; \theta)) \\ &= \frac{1}{1 - \exp(-y \cdot f(x; \theta))} \end{aligned}$$

We receive the decision criterion:

$$\hat{y} = \arg \max_{y \in \{+1, -1\}} p(y | x; \theta)$$

Now we can apply MLE/MAP to find a good  $\theta$ .

**Rmk.**  $X \sim \text{Ber}(p)$  is normally  $X \in \{0, 1\}$ , but here:  $X \in \{+1, -1\}$

**Rmk.** Similar to how  $\epsilon \sim \mathcal{N}(0, \sigma^2)$  was noise in regression, here the noisy outputs are simply  $\{+1, -1\}$ . ("Bernoulli Noise")

MLE corresponds to minimizing logistic loss:

$$\begin{aligned} \hat{\theta}_{\text{MLE}} &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n -\log(y_i | x_i; \theta) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \log\left(\underbrace{1 + \exp(-y_i \cdot f(x_i; \theta))}_{\text{Logistic Loss}}\right) \end{aligned}$$

MAP corresponds to  $l_2$ -regularized regression with sq. loss.  
Using a Gaussian prior:  $\theta^* \sim \mathcal{N}(0, \sigma_\theta^2 \cdot \mathbf{I}_d)$

$$\begin{aligned}\hat{\theta}_{\text{MAP}} &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \left( -\log(p(y_i | x_i, \theta)) - \log(p(\theta)) \right) \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \log(1 + \exp(-y_i \cdot f(x_i; \theta))) + \frac{d}{2} \log(2\pi\sigma_\theta^2) + \frac{1}{\sigma_\theta^2} \|\theta\|^2 \\ &= \arg \min_{\theta \in \Theta} \sum_{i=1}^n \log(1 + \exp(-y_i \cdot f(x_i; \theta))) + \frac{1}{\sigma_\theta^2} \|\theta\|^2\end{aligned}$$

**Rmk.** A Laplace prior corresponds to  $l_1$ -regularized log. regression

### 9.5.2 Classification: Generative

**Problem:** Estimate the joint distribution  $\mathbb{P}_{X,Y}$

An advantage of this is:  $Y | X$  can be estimated from  $\mathbb{P}_{X,Y}$ :

$$p(y | x) = \frac{p(y, x)}{p(x)} = \frac{p(y, x)}{\int_{y'} p(y', x) dy'}$$

However, estimating  $\mathbb{P}_{X,Y}$  from  $Y | X$  is impossible.  
We instead rely on:

$$p(y | x) \propto p(y) \cdot p(x | y) \quad (\text{Bayes})$$

#### Method Gaussian Naive Bayes

This model is used:

$$\mathbb{P}[Y = y] = p_y \quad \text{s.t.} \quad y \in \mathcal{Y} = \{1, \dots, c\} \wedge \sum_{y=1}^c p_y = 1$$

**Assumption:** Conditional independene.

i.e. every feature is independent, thus *naive* Bayes.

$$p(x_1, \dots, x_n | y) = \prod_{j=1}^n p(x_j | y)$$

For each  $x_i$  we use a Gaussian distribution:

$$X_j | Y = y \sim \mathcal{N}(\mu_{y,j}, \sigma_{y,j}^2)$$

#### L. MLE of the Gaussian Naive Bayes Model

$$\hat{p}_y = \frac{n_y}{n}, \quad \hat{\mu}_{y,j} = \frac{1}{n_y} \sum_{i:y_i=y} x_{i,j}, \quad \hat{\sigma}_{y,j}^2 = \frac{1}{n_y} \sum_{i:y_i=y} (x_{i,j} - \hat{\mu}_{y,j})^2$$

Here  $n_y$  is the frequency of label  $y$  in  $\mathcal{D}$

This yields the decision rule for Gaussian Naive Bayes:

$$\begin{aligned}\hat{y} &= \arg \max_y \log(\hat{p}(y | x)) \\ &\stackrel{\text{Bay.}}{=} \arg \max_y \log(\hat{p}(y)) + \sum_{j=1}^d \log(\hat{p}(x_j | y)) \\ &\stackrel{\text{Lem.}}{=} \arg \max_y \log(\hat{p}(y)) - \frac{1}{2} \sum_{j=1}^d \left( \log \hat{\sigma}_{y,j}^2 + \frac{1}{\hat{\sigma}_{y,j}^2} (x_j - \hat{\mu}_{y,j})^2 \right)\end{aligned}$$

And the special case for binary classification:

$$\hat{y} = \text{sign}(\hat{f}(x)) = \text{sign}\left(\log \frac{\hat{p}(+1 | x)}{\hat{p}(-1 | x)}\right)$$

#### Method Linear Discriminant Analysis

Special Case: GNB with  $\forall j \leq d: \sigma_{+1,j}^2 = \sigma_{-1,j}^2 = \sigma_j^2$ .

In this case  $\hat{f}$  is a linear decision boundary.

$$\hat{y} = \text{sign}(\hat{f}(x)) = \text{sign}(\mathbf{w}^\top \mathbf{x} + w_0)$$

Where  $\mathbf{w}, w_0$  can be derived like this:

$$\begin{aligned}\hat{f}(x) &= \log \frac{\hat{p}(+ | x)}{\hat{p}(- | x)} \\ &= \log \frac{\hat{p}_+ \prod_{j=1}^d p(x_j | +)/p(x)}{\hat{p}_- \prod_{j=1}^d p(x_j | -)/p(x)} \\ &= \log \frac{\hat{p}_+}{1 - \hat{p}_+} + \log \frac{\prod_{j=1}^d \frac{1}{\sqrt{2\pi\hat{\sigma}_j^2}} \exp\left(-\frac{(x_j - \hat{\mu}_{+,j})^2}{2\hat{\sigma}_j^2}\right)}{\prod_{j=1}^d \frac{1}{\sqrt{2\pi\hat{\sigma}_j^2}} \exp\left(-\frac{(x_j - \hat{\mu}_{-,j})^2}{2\hat{\sigma}_j^2}\right)} \\ &= \log \frac{\hat{p}_+}{1 - \hat{p}_+} + \sum_{j=1}^d \frac{1}{2\hat{\sigma}_j^2} \left( -(x_j - \hat{\mu}_{+,j})^2 + (x_j - \hat{\mu}_{-,j})^2 \right) \\ &= \underbrace{\sum_{j=1}^d \frac{1}{\hat{\sigma}_j^2} (\hat{\mu}_{+,j} - \hat{\mu}_{-,j}) x_j}_{=: \mathbf{w}^\top \mathbf{x}} + \underbrace{\log \frac{\hat{p}_+}{1 - \hat{p}_+} + \sum_{j=1}^d \frac{\hat{\mu}_{-,j}^2 - \hat{\mu}_{+,j}^2}{2\hat{\sigma}_j^2}}_{=: w_0}\end{aligned}$$

## 10 Gaussian Mixture Models

The unsupervised case: No labels given:  $\mathcal{D} = \{x_i\}_{i=1}^n$ .

**Notation**  $Z$  is used over  $Y$  as we assume the labels to be unknown.  
(Unsupervised Learning)

#### D. Gaussian Mixture Model

$$\textbf{Parameters: } \theta = \left( \overbrace{\mu_j, \Sigma_j}^{\mathcal{N}_j}, w_j \right)_{j=1}^k$$

**Assumption:**  $x$  is convex comb. of Gaussian distributions.  
i.e. a mixture of models

$$\begin{aligned}\mathbb{P}[x | \theta] &= \mathbb{P}[x | \mu, \Sigma, w] \quad (\text{Convex comb.}) \\ &= \sum_{j=1}^k \mathbb{P}[Z = j] \cdot \mathbb{P}[x | Z] \quad (\text{Gaussian}) \\ &= \sum_{j=1}^k w_j \cdot \mathcal{N}(x; \mu_j, \Sigma_j)\end{aligned}$$

$w_j = \mathbb{P}[Z = j]$  is the weight of class  $j$ ,  $\mathcal{N}(\mu_j, \Sigma_j)$  its distr.  
 $w_j \geq 0, \quad \sum_{j=1}^k w_j = 1$

#### L. Negative Log-Likelihood (GMM)

$$L(w_{1:k}, \mu_{1:k}, \Sigma_{1:k}) = - \sum_{i=1}^n \log \left( \sum_{j=1}^k w_j \cdot \mathcal{N}(x_i; \mu_j, \Sigma_j) \right)$$

**Rmk.** This is NP-hard for unobserved labels.

SGD can be used, but EM is preferred.

#### L. Classification Rule (GMM)

$$\hat{y} \stackrel{\text{def}}{=} \arg \max_y p(y | x)$$

$$\stackrel{\text{Bay.}}{=} \arg \max_y p(y) \cdot p(x | y)$$

$$\stackrel{\text{GMM}}{=} \arg \max_j \left( \log(w_j) + \frac{1}{2} \log(\det(2\pi\Sigma_j)) + \frac{1}{2} (x - \mu_j)^\top \Sigma_j^{-1} (x - \mu_j) \right)$$

## 10.1 Expectation Maximization

**Problem:** Fitting a GMM: unknown labels & missing data.

**Solution:** Iterative methods.

Intuitively: Impute labels using guessed  $\theta$ , then update guessed  $\theta$ .

### Method Hard Expectation Maximization

Assign a fixed class to each label.

**Initialisation:**  $\theta^{(0)} = (\mu_j^{(0)}, \Sigma_j^{(0)}, w_j^{(0)})_{j=1}^k$

**Iteration:**  $t \geq 1$

- **E-Step:** Find  $\mathcal{D}^{(t)} = \{(x_1, z_1^{(t)}), \dots, (x_n, z_n^{(t)})\}$  via:

$$\begin{aligned} z_i^{(t)} &= \arg \max_z \mathbb{P}[z \mid x_i, \theta^{(t-1)}] \\ &= \arg \max_z \mathbb{P}[z \mid \theta^{(t-1)}] \cdot \mathbb{P}[x_i \mid z, \theta^{(t-1)}] \end{aligned}$$

- **M-Step** Compute  $\hat{\theta}_{\text{MLE}}$  as for Gauss. Bayes Classifier:

$$\theta^{(t)} = \arg \max_{\theta} \mathbb{P}[\mathcal{D}^{(t)} \mid \theta]$$

### Method Soft Expectation Maximization

Find membership weights for each cluster.

**Initialisation:**  $\theta^{(0)} = (\mu_j^{(0)}, \Sigma_j^{(0)}, w_j^{(0)})_{j=1}^k$

**Iteration:**  $t \geq 1$

- **E-Step:** Find  $\gamma_j^{(t)}(x_i)$  for all  $i \leq n, j \leq k$ .

$\gamma_j^{(t)}(x_i)$  are the cluster membership weights

$$\gamma_j^{(t)}(x_i) = \frac{w_j^{(t-1)} \cdot \mathcal{N}(x_i; \mu_j^{(t-1)}, \Sigma_j^{(t-1)})}{\sum_{l=1}^k w_l^{(t-1)} \cdot \mathcal{N}(x_i; \mu_l^{(t-1)}, \Sigma_l^{(t-1)})}$$

- **M-Step** Update the model parameters (MLE):

$$\begin{aligned} w_j^{(t)} &\leftarrow \frac{1}{n} \sum_{i=1}^n \gamma_j^{(t)}(x_i), \quad \mu_j^{(t)} \leftarrow \frac{\sum_{i=1}^n \gamma_j^{(t)}(x_i) \cdot x_i}{\sum_{i=1}^n \gamma_j^{(t)}(x_i)} \\ \Sigma_j^{(t)} &\leftarrow \frac{\sum_{i=1}^n \gamma_j^{(t)}(x_i) \cdot (x_i - \mu_j^{(t)})(x_i - \mu_j^{(t)})^\top}{\sum_{i=1}^n \gamma_j^{(t)}(x_i)} \end{aligned}$$

**Example:** Let  $Y \in \{1, 2, 3\}$  be the clusters, we'd like to assign  $X \in \{0, 1\}^5$ . We're using *Soft Expectation Maximization* to find the cluster assignments  $\gamma_j(x_i)$ . We assume  $\mathbb{P}[X_i = 1 \mid Y]$  follows a  $\text{Ber}(p)$  distribution.

We'd like to find  $\gamma_3(x_2) = \mathbb{P}[Y = 3 \mid X = x_2]$ . The E-step is:

$$\mathbb{P}[Y = 3 \mid X = x_2] = \frac{\mathbb{P}[X = x_2 \mid Y = 3] \cdot \overbrace{\mathbb{P}[Y = 3]}^{w_3}}{\sum_{i=1}^3 \mathbb{P}[X = x_2 \mid Y = i] \cdot \mathbb{P}[Y = i]}$$

For the M-step, we update the assignments via MLE. For  $\text{Ber}(p)$ , this is the *sample mean*.

$$\mathbb{P}[X_i = 1 \mid Y = 1] = \frac{\sum_{k=1}^3 \mathbb{I}_{\{x_k^i=1\}} \cdot \mathbb{P}[Y = 1 \mid X = x_k]}{\sum_{k=1}^3 \mathbb{P}[Y = 1 \mid X = x_k]}$$

Where we use  $x \in \mathbb{R}^5$

### 10.1.1 Convergence

EM does not generally guarantee global convergence.

#### Th. Continuous Improvement of EM

EM guarantees continuous improvement for  $\{\theta^{(t)}\}_{t=1}^\infty$

$$\log(P(X \mid \theta^{(t+1)})) \geq \log(P(X \mid \theta^{(t)}))$$

Where  $P(X \mid \theta^{(t)})$  is the log-likelihood.

Further,  $\{\log(P(X \mid \theta^{(t)}))\}_{t=1}^\infty$  converges to a finite value and  $\{\theta^{(t)}\}_{t=1}^\infty$  converges to a stationary point of the log-likelihood.

## 10.2 Model Selection

**Problem:** How to choose  $k$ ?

**Rmk.** Increasing  $k$  also increases the likelihood (in general), as increasingly every point can be seen as a separate cluster.

**Solution:** Quantitative criteria & domain knowledge.

BIC & AIC are used: Lower is better. Both metrics balance model fit against complexity by penalizing large  $k$ .

#### D. Bayesian Information Criterion

$$\text{BIC}(k) = k \log(n) - \log(L)$$

#### D. Akaike Information Criterion

$$\text{AIC}(k) = k - \log(L)$$

$L$  is log-likelihood,  $n$  is number of datapoints

#### Method Model Selection

1. Find BIC & AIC for a range (e.g.  $1 \leq k \leq 100$ )
2. Choose  $k$  with lowest metric

#### Rmk. Elbow Methow

Alternatively, plot some performance metric (e.g. neg. log-likelihood) and find the inflection point indicating diminishing returns.



### 11.2.2 Multi-Head Attention

In practice, multiple attention "heads" are used to capture different aspects of the input.

Each head  $r = 1, \dots, h$  has its own  $\mathbf{W}_r^Q, \mathbf{W}_r^K, \mathbf{W}_r^V$  matrices.

#### D. Multi-Head Attention Update

Use  $h$  heads in parallel. The concatenation collects all heads' information, and  $\mathbf{W}^O$  projects it back to the original dimension  $d$ .

$$\mathbf{Z}'_r = \text{softmax}\left(\frac{\mathbf{Q}_r \mathbf{K}_r^\top}{\sqrt{d_k}} + \mathbf{M}\right) \mathbf{V}_r$$

$$\text{MultiHead}(\mathbf{Z}) = \text{Concat}(\mathbf{Z}'_1, \dots, \mathbf{Z}'_h) \mathbf{W}^O$$

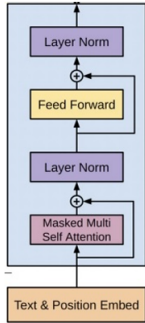
For  $\mathbf{Q}_r = \mathbf{Z} \mathbf{W}_r^Q, \mathbf{K}_r = \mathbf{Z} \mathbf{W}_r^K, \mathbf{V}_r = \mathbf{Z} \mathbf{W}_r^V$

### 11.2.3 Transformer

To build a network from multi-head attention, the transformer block is used.

#### D. Transformer Block

A Transformer block consists of a Multi-Head Attention layer followed by a Feed-Forward Neural Network layer. Each layer has residual connections ( $\oplus$ ) and layer normalization.



*Introduction to Machine Learning (2026), p. 285*

Normalization & residual connections help prevent vanishing gradients and stabilize training. Intuitively: only learn small corrections instead of relearning the whole token.

### 11.2.4 Decoding

This is the process of generating text from the model.

At each step, all tokens (so far) are used as input, returning a probability distribution over the vocabulary. The next token is sampled from it.

**Rmk.** During training, the entire sequence is used for the forward pass (with causal masking). At inference, each token requires a new forward pass.

#### Method Greedy Decoding

At each step, choose the token with the highest probability.

$$x_t = \arg \max_x P(X_t = x \mid X_{1:t-1} = x_{1:t-1}, \theta)$$

Fast, but leads to repetitive text.

#### Method Beam Search

Keep track of the  $b$  most likely sequences at each step, then expand by each by one token, keep the best  $b$  again.

#### Method Sampling

Sample the next token from the predicted distribution.

$$P(X_t = x \mid X_{1:t-1} = x_{1:t-1}) = \text{softmax}\left(\frac{f(x_{1:t-1})_x}{\tau}\right)_x$$

$\tau$  is the temperature parameter, controlling the sharpness of the distribution.  $\tau \rightarrow 0$  mimics greedy decoding,  $\tau \rightarrow \infty$  leads to uniform sampling.