Inference and Learning

Author: Wenxiao Yang

Institute: Department of Mathematics, University of Illinois at Urbana-Champaign

Date: 2022

All models are wrong, but some are useful.

Contents

Chapter 1 Information-Theoretic Functional

Definitions

1.

Definition 1.1 (Entropy)

Entropy *of pmf* $\{p(y), y \in Y\}$

$$H(p) = -\sum_{y \in Y} p(y) \ln p(y)$$

(concave in p*)*

♣

2.

Definition 1.2 (KL divergence, Relative entropy)

The ***Kullback-Leibler divergence (or relative entropy)*** *of two pmf's* $p(x), x \in X$ *and* $q(x), x \in X$ *is defined as*

$$D(p\|q) = \sum_{y \in Y} p(y) \ln \frac{p(y)}{q(y)} \geq 0$$

with equality iff $p = q$*. (convex in* (p, q)*)*

♣

3.

Definition 1.3 (Cross-entropy)

The ***cross-entropy*** *of a pmf* $p(x), x \in X$ *relative to another pmf* $q(x), x \in X$

$$H(p, q) = -\sum_{y \in Y} p(y) \ln q(y)$$

$$= H(p) + D(p\|q)$$

$H(p, q) \geq H(p)$*, the lower bound is achieved by* $q = p$*.*

♣

4.

Definition 1.4 (Mutual Information)

Let (x, y) *be a pair of random variables with values over the space* $X \times Y$ *If their joint distribution is* $p_{X,Y}(x, y)$ *and the marginal distributions are* $p_X(x)$ *and* $p_Y(y)$*, the* ***mutual information*** *is defined as*

$$I(p_{X,Y}) = \sum_{x \in X} \sum_{y \in Y} p_{X,Y}(x, y) \log \left(\frac{p_{X,Y}(x, y)}{p_X(x) p_Y(y)} \right)$$

$$= H(p_X) + H(p_Y) - H(p_X, p_Y)$$

♣

Chapter 2 Learning Basics

2.1 Parameters and Hyperparameters

Definition 2.1

Parameters *are values learned by the model given the data.* ♣

e.g. β, W, b, θ.

Definition 2.2

Hyperparameters *are values supplied to tune the model and cannot be learned from data.* ♣

e.g. Number of Hidden Layers, Neurons, and Epochs to Train. Learning Rate, and Batch Size.

2.2 Neural Network: Back Propagation Algorithm

Neural Network

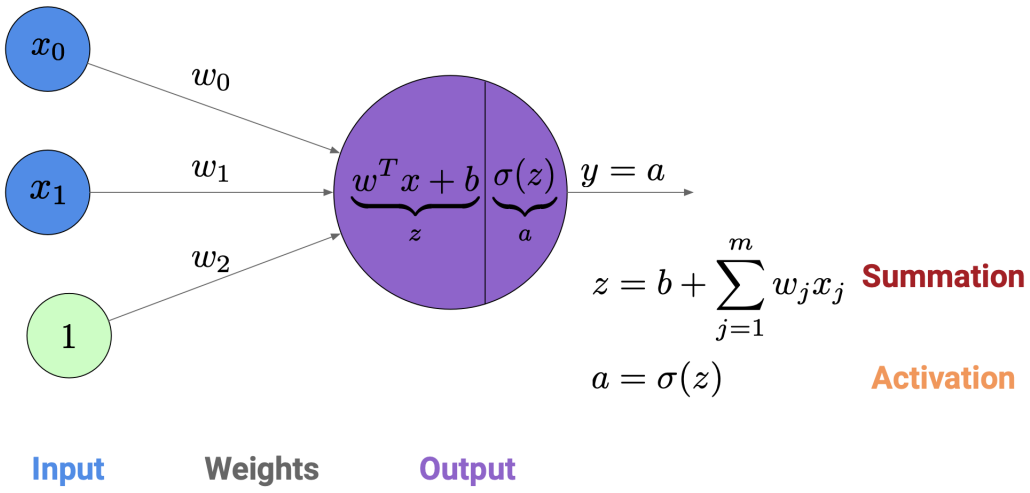


Figure 2.1: Simple Neural Network

Given a vector input x, we need to find the best estimator $\hat{y}$ which minimizes lost function. In the figure that has only one layer and one pathway, we find the parameter (ω, b) to form an input $\omega^T x + b$ to neuron σ (activation function). Then, the final output (estimator) of the network is $\hat{y} = \sigma(\omega^T x + b)$.

2.2.1 Activations

Definition 2.3

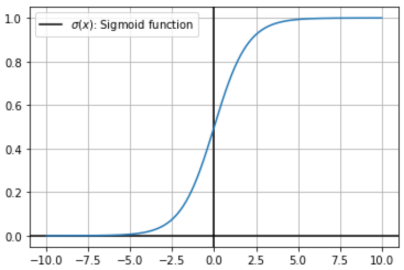
Activation functions are element-wise gates for letting information propagate to future layers either transformed with non-linearities or left as-is.

♣

Example of activation function:

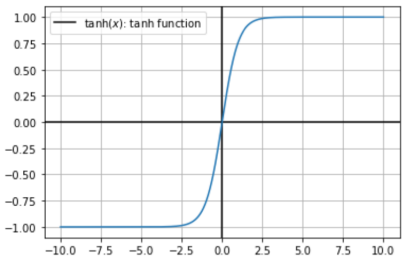
Sigmoid

$$\frac{1}{1+\exp(-x)}$$



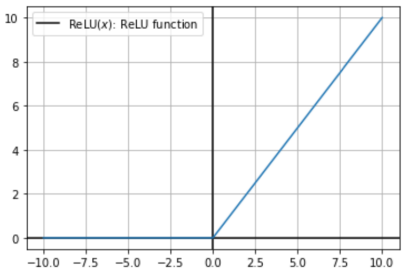
TanH

$$\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



ReLU

$$\max(0,x)$$



Leaky ReLU

$$\max(0.1x,x)$$

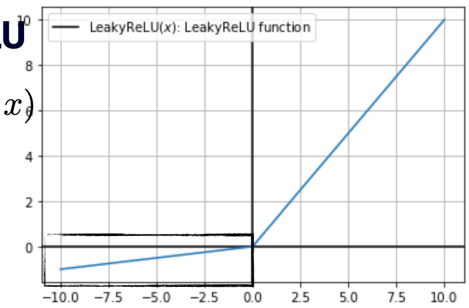


Figure 2.2: Activations

(1) **Identity:**

$$\text{identity}(x) = I(x) = x$$

(2) **Binary:**

$$\text{binary}(x) = \text{step}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

(3) **Sigmoid:**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma(-x) = 1-\sigma(x);\ \frac{d\sigma(x)}{dx} = \sigma(x)\cdot(1-\sigma(x))$$

$$\frac{\partial\sigma(\vec{x})}{\partial\vec{x}} = \begin{bmatrix} \sigma(x_1)(1-\sigma(x_1)) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma(x_n)(1-\sigma(x_n)) \end{bmatrix} = \text{diag}(\sigma(\vec{x}) \cdot (1-\sigma(\vec{x})))$$

(4) **TanH:**

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

(TanH is a rescaled sigmoid)

$$tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} = 1 - 2\sigma(-2x) = 2\sigma(2x) - 1$$

(5) **ReLU:**

$$g(x) = \max(0, x)$$

(6) **Leaky ReLU:**

$$g(x) = \max(0.1x, x)$$

(7) **Softmax:** $S_j(\vec{x}) = \frac{e^{x_j}}{\sum_{i=1}^n e^{x_i}}$,

$$S(\vec{x}) = \left[\frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^n e^{x_i}}, \cdots, \frac{e^{x_n}}{\sum_{i=1}^n e^{x_i}}\right]^T$$

$$\frac{\partial S_j(\vec{x})}{\partial x_j} = S_j(\vec{x})[1 - S_j(\vec{x})]$$

2.2.2 Multilayer Neural Network

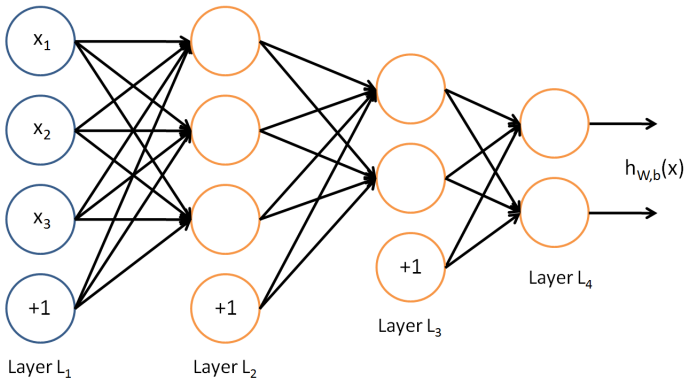


Figure 2.3: Multilayer Neural Network

- Number of neurons in each layer can be different.
- All weights on edge connecting layers $m-1$ and m is matrix $W^{(m)}$, with $w_{ij}^{(m)}$ being the weight connecting output j of layer $m-1$ with neuron i of layer m.

- Input to network is vector x; output of layer m is vector $y^{(m)}$

$$y_i^{(1)} = \sigma(x_i^{(1)}), \text{ with } x_i^{(1)} = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$y^{(1)} = \sigma(x^{(1)}), \text{ with } x^{(1)} = W^{(1)}x + b^{(1)}$$

$$y^{(2)} = \sigma(x^{(2)}), \text{ with } x^{(2)} = W^{(2)}y^{(1)} + b^{(2)}$$

$$\vdots$$

$$y^{(M)} = \sigma(x^{(M)}), \text{ with } x^{(M)} = W^{(M)}y^{(M-1)} + b^{(M)}$$

We want to find the weights $W^{(1)}, \cdots, W^{(M)}, b^{(1)}, \cdots, b^{(M)}$ so that the output of last layer

$$\hat{y} = y^{(M)} \approx f^*(x) = y$$

$f^*(x)$ is the unknown thing we need to predict.

We use labelled training data, i.e.

$$(x[1], y[1]), (x[2], y[2]), \cdots (x[N], y[N])$$

Minimize the "empirical" loss on training data.

$$J = \sum_{i=1}^{N} L(y[i], \hat{y}[i])$$

where $\bar{y}[i]$ is the output of NN whose input is $x[i]$.

- L is the function of $W^{(1)}, \cdots, W^{(M)}, b^{(1)}, \cdots, b^{(M)}$ to measure the loss. e.g. the square loss

$$L(y, \hat{y}) = (y - \hat{y})^2$$

- We wish to minimize J using a gradient descent procedure.
- To compute gradient we need:

$$\frac{\partial L}{\partial w_{ij}^{(l)}} \text{ for each } l, i, j; \quad \frac{\partial L}{\partial b_i^{(l)}} \text{ for each } l, i.$$

2.2.3 A Simple Example of Back Propagation Algorithm

We can consider a simple example (one layer, two pathways)

$W^{(1)} = [w_{1,1}^{[1]}, w_{2,1}^{[1]}]^T$, $b^{(1)} = [0, 0]^T$, $\sigma_1(x) = x$.
$W^{(2)} = [w_{1,1}^{[2]}, w_{1,2}^{[2]}]$, $b^{(2)} = 0$, $\sigma_2(x) = x$.

$$[a_1^{[1]}, a_2^{[1]}]^T = \sigma_1(W^{(1)}x_1 + b^{(1)}) = [w_{1,1}^{[1]}x_1, w_{2,1}^{[1]}x_1]^T$$

$$\hat{y} = \sigma_2(W^{(2)}[a_1^{[1]}, a_2^{[1]}]^T + b^{(2)}) = (w_{1,1}^{[1]}w_{1,1}^{[2]} + w_{2,1}^{[1]}w_{1,2}^{[2]})x_1$$

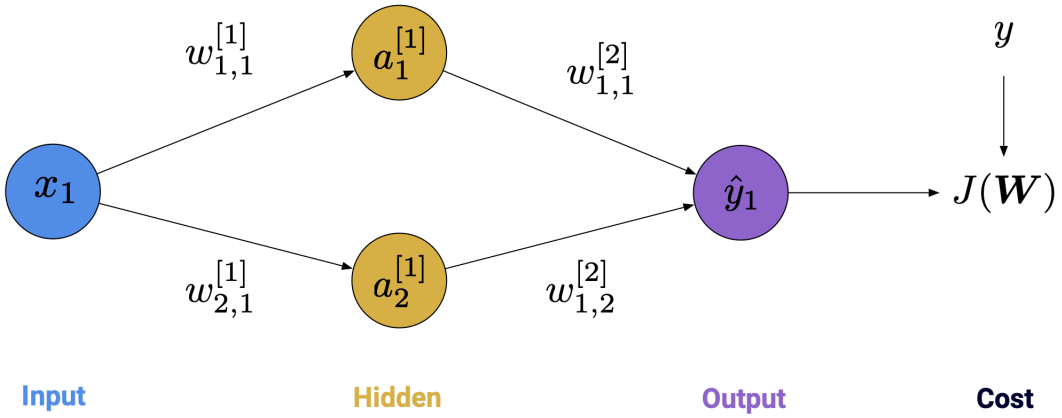


Figure 2.4: Two Independent Pathways

$$\frac{\partial J(\hat{y})}{\partial w_{2,1}^{[1]}} = \frac{\partial J(\hat{y})}{\partial \hat{y}} \cdot \frac{\hat{y}}{\partial a_2^{[1]}} \cdot \frac{a_2^{[1]}}{\partial w_{2,1}^{[1]}}$$

2.2.4 Back Propagation Algorithm

Recall $y_i^{(m)} = \sigma(x_i^{(m)})$, $x_i^{(m)} = \sum_j w_{ij}^{(m)} y_j^{(m-1)} + b_i^{(m)}$

$$\frac{\partial L}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial w_{ij}^{(m)}}$$

$$\frac{\partial L}{\partial b_i^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial b_i^{(m)}}$$

For large M,

- $\frac{\partial L}{\partial y_i^{(M)}}$ is easy to compute.
- $\frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} = \frac{\partial \sigma(x_i^{(M)})}{\partial x_i^{(M)}} = \sigma'(x_i^{(M)})$, (assuming σ differentiable).
- $\frac{\partial x_i^{(M)}}{\partial w_{ij}^{(M)}} = y_j^{(M-1)}$

Thus,

$$\frac{\partial L}{\partial w_{ij}^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} \cdot \sigma'(x_i^{(M)}) \cdot y_j^{(M-1)}$$

Similarly,

$$\frac{\partial L}{\partial b_i^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} \cdot \frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} \cdot \frac{\partial x_i^{(M)}}{\partial b_i^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} \cdot \sigma'(x_i^{(M)})$$

For $1 \leq m < M$, in this situation $\frac{\partial L}{\partial y_i^{(m)}}$ is not easy to compute. Note that $x^{(m+1)} = W^{(m+1)} y^{(m)} + b^{(m+1)}$.

$$\begin{aligned}
\frac{\partial L}{\partial y_i^{(m)}} &= \sum_k \frac{\partial L}{\partial x_k^{(m+1)}} \cdot \frac{\partial x_k^{(m+1)}}{\partial y_i^{(m)}} \\
&= \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \cdot \frac{\partial y_k^{(m+1)}}{\partial x_k^{(m+1)}} \cdot \frac{\partial x_k^{(m+1)}}{\partial y_i^{(m)}} \\
&= \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \cdot \sigma'(x_k^{(m+1)}) \cdot w_{ki}^{(m+1)}
\end{aligned}$$

Then use this form to compute,

(We can set $\delta^{(m)} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)})$ to avoid duplicate computation.)

$$\begin{aligned}
\frac{\partial L}{\partial w_{ij}^{(m)}} &= \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial w_{ij}^{(m)}} \\
&= \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)}) \cdot y_j^{(m-1)} \\
&= \delta^{(m)} \cdot y_j^{(m-1)}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\frac{\partial L}{\partial b_i^{(m)}} &= \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial b_i^{(m)}} \\
&= \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)}) \\
&= \delta^{(m)}
\end{aligned}$$

Summary

1. Compute $\frac{\partial L}{\partial y_i^{(M)}}$.
2. Use
$$\frac{\partial L}{\partial y_i^{(m)}} = \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \cdot \sigma'(x_k^{(m+1)}) \cdot w_{ki}^{(m+1)}$$
compute $\frac{\partial L}{\partial y_i^{(m)}}$ for $m = 1, 2..., M-1$.
3. Compute
$$\frac{\partial L}{\partial b_i^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)}) = \delta^{(m)}$$
for $m = 1, 2..., M$.
4. Compute
$$\frac{\partial L}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)}) \cdot y_j^{(m-1)} = \delta^{(m)} \cdot y_j^{(m-1)}$$
for $m = 1, 2..., M$.

2.2.5 Other Methods

Stochastic Gradient Descent (SGD)

Subgradient Method

2.3 Perceptron Algorithm

Definition 2.4

Binary linear classifiers distinguish between two categories through a linear function of the inputs.

Definition 2.5

Linearly separable refers to a line that can be drawn to perfectly split the two classes.

The Perceptron algorithm is an efficient algorithm for learning a **linear separator** in $d-$dimensional space, with a mistake bound that depends on the margin of separation of the data.

2.3.1 General Idea

Given the training data

$$D = \left\{ \left\langle x^{(i)}, y^{(i)} \right\rangle, i = 1, ..., n \right\} \in (\mathbb{R}^m \times \{0, 1\})^n$$

we want to know the exact value of $y \in \{0, 1\}$.

$$\hat{y} = g(b + \boldsymbol{w}^T \boldsymbol{X})$$

Output Activation Bias Weights Input

Summation

Figure 2.5: Perceptron Output

General idea:

- If the perceptron correctly predicts ($\hat{y} = y$):
  - Do nothing
- If the perceptron yields an incorrect prediction ($\hat{y} \neq y$):
  - If the prediction is 0 and truth is 1 ($\hat{y} = 0 | y = 1 \Rightarrow e = y - \hat{y} = 1$), add feature vector to weight vector.
  - If the prediction is 1 and truth is 0 ($\hat{y} = 1 | y = 0 \Rightarrow e = y - \hat{y} = -1$), subtract feature vector from the weight vector.

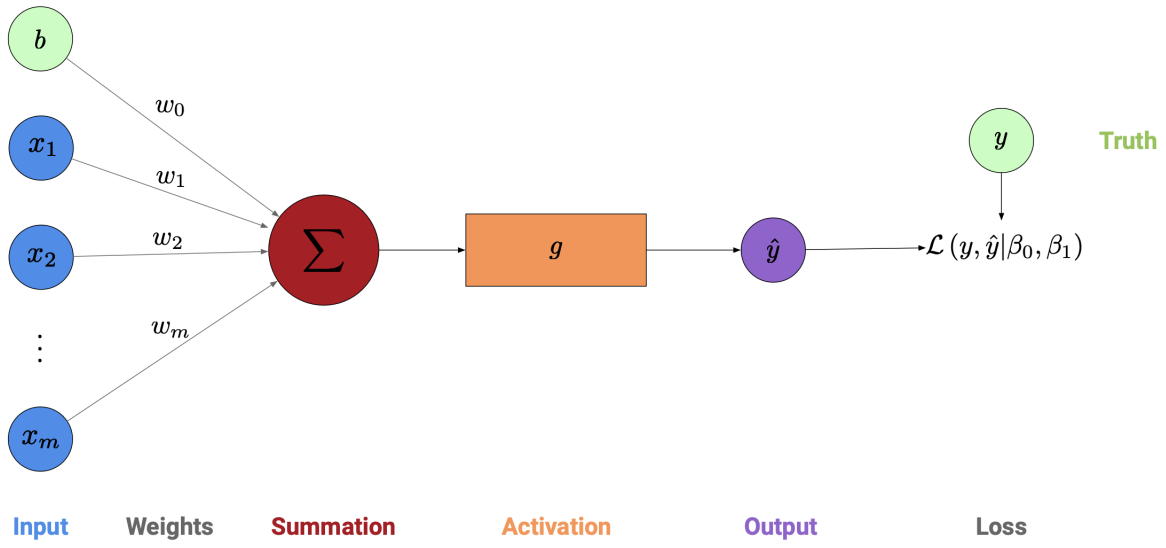


Figure 2.6: Perceptron

Since we want the prediction to be either 0 or 1, we usually use binary function as the activation function in perceptron.

2.3.2 Algorithm

Perceptron Algorithm:

- Initialize weights (including a bias term) to zero, e.g. $W = [w, b] = 0^{m+1}$.
- Under each training epoch: Compute for each sample $\langle x^{(i)}, y^{(i)} \rangle \in D$
  - A prediction $\hat{y}^{(i)} = g({x^{(i)}}^T W)$
  - Prediction error $e^{(i)} = y^{(i)} - \hat{y}^{(i)}$
  - Weighted update $W = W + \eta e^{(i)} x^{(i)}$

2.3.3 Limitations

(1) Only provides a linear classifier boundary.

(2) Only allows for **binary classifier** between two classes.

(3) **No convergence possible** if classes are not linearly separable.

(4) Perceptron will yield **multiple boundary/"optimal" solutions**.

(5) Boundaries found may **not** perform **equally well**.

2.4 ADAptive LInear NEuron (ADALINE)

2.4.1 General Idea

Except the activation function in perceptron, we can add a threshold function.

In perceptron, we generate the estimation $\hat{y}$ (after binary function) to help update weight $\{w_i\}_{i=0}^m$. However, in ADALINE, we minimize MSE $z = x^T W$ to update weight $\{w_i\}_{i=0}^m$ before output estimation $\hat{y}$ (before binary function).

Before entering threshold (binary function), we want to minimize a mean- squared error (MSE) loss function to estimate weights.

e.g. suppose $g(x) = x$, let $z = x^T W$ be the input of threshold, for each y,

$$W^* = \underset{W}{\operatorname{argmin}}\, L(z, y) = (y - z)^2$$

$$\frac{\partial L(z, y)}{\partial w_i} = -2(y - z)\frac{\partial z}{\partial w_i} = -2(y - z)x_i$$

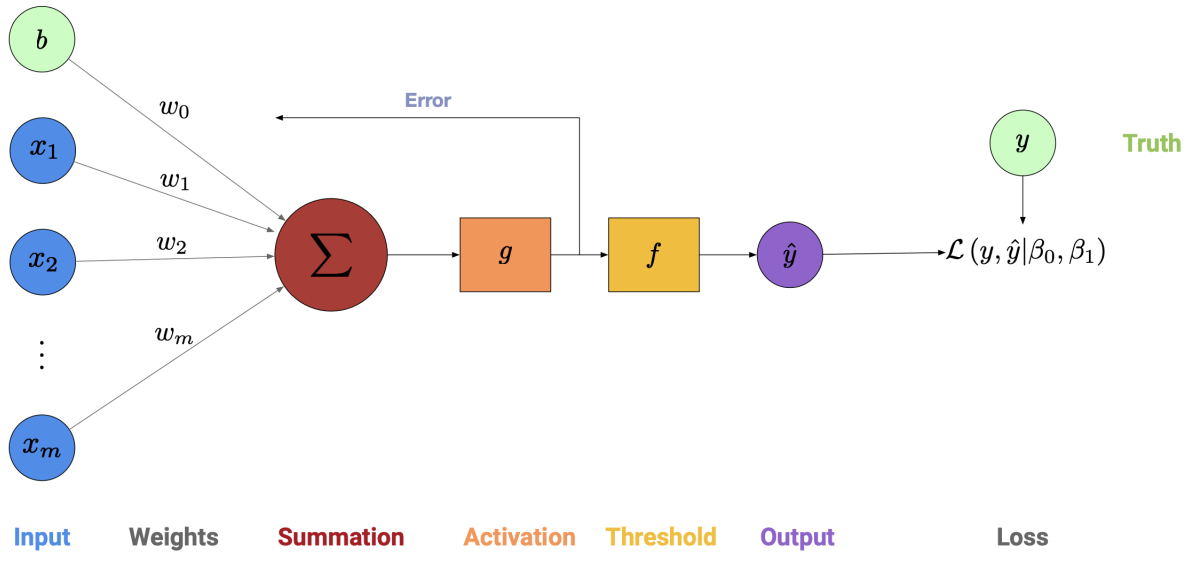


Figure 2.7: ADALINE

2.4.2 Widrow-Hoff Delta Rule

(Gradient Descent Rule for ADALINE)

- **Original:**

$$W = W + \eta(y^{(j)} - z)x^{(j)}$$

- **Unit-norm:**

$$W = W + \eta(y^{(j)} - z)\frac{x^{(j)}}{\|x^{(j)}\|}$$

where $\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_m^2}$

The Perceptron and ADALINE use variants of the delta rule!

(1) **Perceptron**: Output used in delta rule is $\hat{y} = g(x^T W)$; $W = W + \eta(y^{(j)} - \hat{y}^{(j)})x^{(j)}$

(2) **ADALINE**: Output used to estimate weights is $z = x^T W$. $W = W + \eta(y^{(j)} - z)x^{(j)}$

2.5 Logistic Regression (Binary-class Output)

2.5.1 Generative and Discriminative Classifiers

The most important difference between naive Bayes and logistic regression is that logistic regression is a **discriminative classifier** while naive Bayes is a **generative classifier**.

Suppose we want to classify class A (dogs) and class B (cats) (More genearl form: assign a class $c \in C$ to a document $d \in D$):

(1) **Generative model:** A generative model would have the goal of understanding what dogs look like and what cats look like. You might literally ask such a model to 'generate', i.e., draw, a dog.

e.g. **naive Bayes:** we do not directly compute the probability that the document d belongs to each class c, $P(c|d)$. We compute likelihood $P(d|c)$ and prior probability $P(c)$ to generate best estimation $\hat{c}$. (i.e., we want to know what should the distribution of a document d in class c be like.)

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

(2) **Discriminative model:** A discriminative model, by contrast, is only trying to learn to distinguish the classes (perhaps without learning much about them). That is we want to directly computing $P(c|d)$.

2.5.2 Sigmoid function

The goal of binary logistic regression is to train a classifier that can make a binary decision about the class of a new input observation.

The input observation is $x = [x_1, ..., x_m]^T$ and the output y is either 1 or 0. Instead of using the optimal weights of each feature x_i and binary activation function (**threshold**: $\hat{y} = 1$ if $z \geq 0$ and $\hat{y} = 0$ otherwise) to estimate in Perceptron and ADALINE, **we want to estimate the probability $P(y = 1|x)$.**

However, the weighted sum $z = x^T W = \sum_{i=1}^{m} w_i x_i + b$ ranges $-\infty$ to ∞. We want to **force the z to be a legal probability**, that is, to lie between 0 and 1.

The **sigmoid function** $\sigma(z) = \frac{1}{1+e^{-z}}$ can be used as acitivation for this purpose, $P(y = 1|x) = \sigma(x^T W)$. Since $1 - \sigma(x) = \sigma(-x)$, $P(y = 0|x) = \sigma(-x^T W)$.

2.5.3 Cross-entropy Loss Function

We choose the parameters W that maximize the log probability of the true y labels in the training data given the observations x. The conditional probability

$$p(y|x) = \begin{cases} \hat{y}, & y = 1 \\ 1 - \hat{y}, & y = 0 \end{cases} = \hat{y}^y (1 - \hat{y})^{1-y}$$

To maximize the probability, we log both sides:

$$\log p(y|x) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$

Then, we want the $\hat{y}$ to maximize the probability (also the logarithm of the probability):

$$\begin{aligned} \hat{y}^* &= \underset{\hat{y} \in [0,1]}{\operatorname{argmax}} \log p(y|x) \\ &= \underset{\hat{y} \in [0,1]}{\operatorname{argmin}} - \log p(y|x) \\ &= \underset{\hat{y} \in [0,1]}{\operatorname{argmin}} -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})) \end{aligned}$$

The right hand side is exactly the **cross-entropy loss function**:

$$L(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

where $\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$

$$\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial w_j} = \left(\sigma(w^T x^{(i)} + b) - y^{(i)}\right) x_j^{(i)} = (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

The risk (Binary Cross-Entropy Cost) of a weight W is

$$J(W) = -\frac{1}{n} \sum_{i=1}^{n} (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

$$\frac{\partial J(w,b)}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}\left(\sigma(w^T x^{(i)} + b) - y^{(i)}\right) x_j^{(i)}$$

2.5.4 Algorithm

- Initialize weights (including a bias term) to zero, e.g. $W = [w, b] = 0^{m+1}$.
- Under each training epoch: Compute for each sample $\langle x^{(i)}, y^{(i)} \rangle \in D$
  - A prediction $\hat{y}^{(i)} = g({x^{(i)}}^T W)$
  - Prediction error $e^{(i)} = y^{(i)} - \hat{y}^{(i)}$
  - Weighted update $W = W + \eta e^{(i)} x^{(i)} = W - \eta \nabla L(W)$

2.6 Softmax Regression (Multi-class Output)

2.6.1 Multi-Class Classification and Multi-Label Classification

Definition 2.6

Multi-Class Classification is a process for assigning each sample exactly one class. In this case, classes are considered mutually exclusive (no intersection). ♣

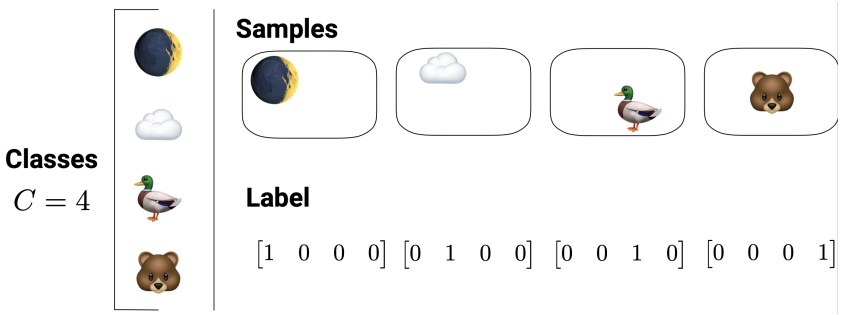


Figure 2.8: Multi-Class Classification

Definition 2.7

Multi-Label Classification or annotation allows for each sample to have 1 or more classes assigned to it. In this case, classes are mutually non-exclusive (one common element). ♣

We can show some examples of activation layer and loss choice in different probelms.

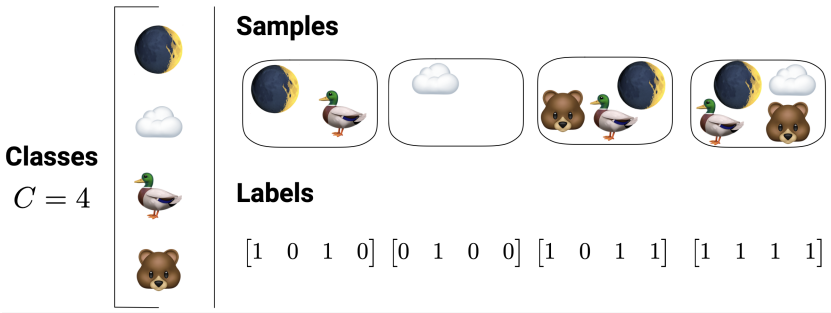


Figure 2.9: Multi-Label Classification

	Output Activation	Loss Function
Binary Classification	Sigmoid	Binary Cross-Entropy
Multi-Class Classification	Softmax	Categorical Cross-Entropy
Multi-Label Classification	Sigmoid	Binary Cross-Entropy
Linear Regression	Identity	Mean Squared Error
Regression to values in [0, 1]	Sigmoid	Mean Squared Error or Binary Cross-Entropy

Figure 2.10: Examples of Activation Layer and Loss Choice

2.6.2 One-hot Encoding

Definition 2.8

One-hot encoding is the process of assigning a single location within a vector to represent a given category.

Strength	High	Medium	Low
"High"	1	0	0
"Medium"	0	1	0
"Low"	0	0	1
...	...	...	...
"Med"	0	1	0

Figure 2.11: One-hot encoding

Examples:

1. $\mathbf{z} = [4]_{1\times 1} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{1\times 5}$

2. $\mathbf{y} = \begin{bmatrix} 3 & 0 & 2 \end{bmatrix}_{1\times 3} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}_{3\times 4}$

3. $\mathbf{v} = \begin{bmatrix} 5 & 0 & 4 & 4 & 3 \end{bmatrix}_{1\times 5} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}_{5\times 6}$

Usefulness of Encodings

1. Close to a traditional design matrix for linear regression that uses a codified dummy variable structure. e.g. FALSE (0) or TRUE (1)
2. Reduce the size of data stored by using numbers instead of strings.
3. Poor if there are too many unique values (e.g. text messages on a phone.)

2.6.3 Softmax function

Definition 2.9

Softmax function or normalized exponential function converts between real valued numbers to values between 0 and 1

♣

Softmax: $S_j(\vec{x}) = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}$,

$$S(\vec{x}) = \left[\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \cdots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}} \right]^T$$

We can show the softmax function has the following properties:

(1) First-derivative:

$$\frac{\partial S_i(\vec{x})}{\partial x_i} = S_i(\vec{x})[1 - S_i(\vec{x})]; \ \frac{\partial S_i(\vec{x})}{\partial x_j} = -S_i(\vec{x})S_j(\vec{x})$$

(2) Stabilizing softmax:

$$S_j(\vec{x} + c) = \frac{e^{x_j + c}}{\sum_{i=1}^{n} e^{x_i + c}} = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}} = S_j(\vec{x})$$

We can minus $\max_i x_i$ to avoid overflow in softmax function. (numerical issue) i.e., $S_j(\vec{x} - (\max_i x_i)) = S_j(\vec{x})$

2.6.4 Categorical Cross-entropy Loss Function

Definition 2.10

Categorical Cross-entropy Loss is a way to quantify the difference between a "true" values $\{y_c\}_{c\in C}$ *and an estimated* $\{\hat{y}_c\}_{c\in C}$ *across* C *categories.*

♣

Note: y needs one-hot encoding firstly, $\hat{y}$ are estimated probability.

$$L(y, \hat{y}) = -\sum_{c\in C} (y_c \cdot \log(\hat{y}_c))$$

Definition 2.11

Categorical Cross-entropy Cost is a way of quantifying the cost over multiple points from different categories.

♣

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c\in C} (y_{i,c} \cdot \log(\hat{y}_{i,c}))$$

2.7 Deep Feedforward Networks

2.7.1 Definition

In any neural network, a dense layer is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to *every neuron* of its preceding layer.

Definition 2.12

Deep feedforward networks, feedforward neural networks, multilayer perceptrons (MLPs), or dense neural networks form the foundations of deep learning models. Learning occurs in only one direction: ***forward****. There are* ***no feedback*** *connections in whichoutputs of the model are fed back into itself. There are no cycles or loops present. Information must flow from the input layer, through one or more hidden layers, before reaching the output.*

♣

A deep neural network contains *Input Layer*, *Hidden Layer*, and *Output Layer*.

2.7.2 Universal Approximation Theorem

Universal Approximation Theorem, in its lose form, states that a **feed-forward network** with a **single hidden layer** containing a finite number of neurons **can approximate any continuous function**. (Which is also

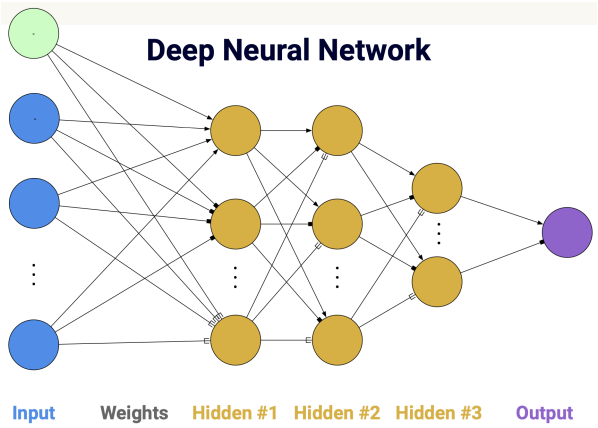


Figure 2.12: Deep Neural Network

equivalent to having a nonpolynomial activation function)

Theorem 2.1 (Universal approximation theorem)

Fix a continuous function $\sigma : \mathbb{R} \to \mathbb{R}$ *(activation function) and positive integers* $d, D \in \mathbb{Z}^+$. *The function* σ *is not a polynomial* $\Leftrightarrow$ *for every continuous function* $f : \mathbb{R}^d \to \mathbb{R}^D$ *(target function), every compact subset* K *of* $\mathbb{R}^d$, *and every* $\epsilon > 0$ *there exists a continuous function* $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ *(the layer output) with representation*

$$f_\epsilon = W_2 \circ \sigma \circ W_1$$

where W_2, W_1 *are composable affine maps and* o *denotes component-wise composition, such that the approximation bound*

$$\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \varepsilon$$

holds for any ϵ *arbitrarily small (distance from* f *to* f_ϵ *can be infinitely small).* ♡

2.8 Mini-batch Optimization

2.8.1 Stochastic Gradient Descent (SGD) and Batch Gradient Descent (BGD)

Stochastic Gradient Descent (SGD)

1. Start with a random guess.
2. For n epochs:
   1) Reorder data
   2) Retrieve an observation $i = 1, 2, ...$ one by one in reordered data:
      (1) Compute gradient on single data point i: $\frac{\partial J_i(W)}{\partial W}$

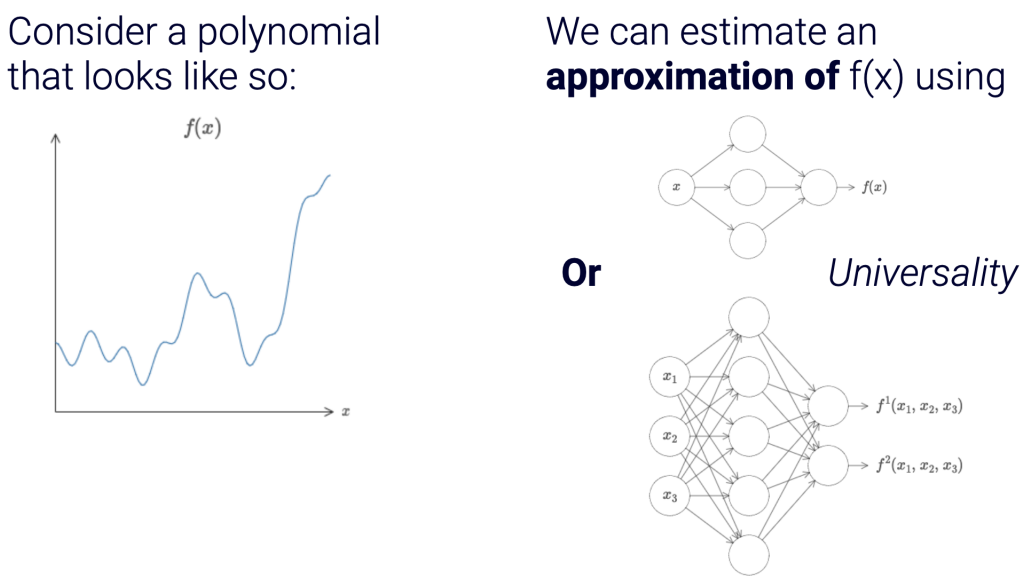


Figure 2.13: Universal Approximation Theorem

(2) Update parameters: $W = W - \alpha \frac{\partial J_i(W)}{\partial W}$

3. Output parameters

Note: ”On-line”/”Stochastic” **Single** Observation Updates

Batch Gradient Descent (BGD)

1. Start with a random guess.
2. For n epochs:
   1) Compute gradients on **all the data**: $\frac{\partial J(W)}{\partial W}$
   2) Update parameters: $W = W - \alpha \frac{\partial J(W)}{\partial W}$
3. Output parameters

Note: **All** data used in update

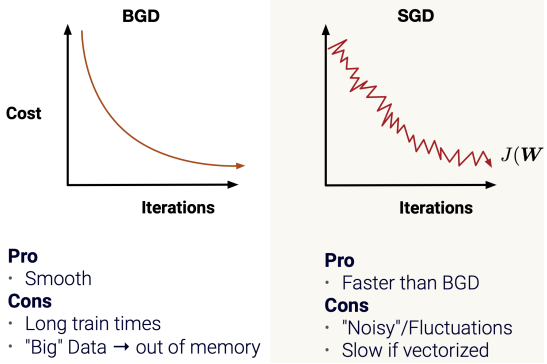


Figure 2.14: BGD and SGD

2.8.2 Mini-Batch Gradient Descent (MBGD)

We want a middle ground between SGD and BGD.

Mini-Batch Gradient Descent (MBGD)

1. Start with a random guess.
2. For n epochs:
    1) Reorder data and retrieve a subet of reordered data with size b (batch size)
    2) Compute gradient on subset: $\frac{\partial J(W)}{\partial W}$
    3) Update parameters: $W = W - \alpha \frac{\partial J(W)}{\partial W}$
3. Output parameters

If $b = n$, the algorithm is exactly BGD; If $b = 1$, the algorithm is exactly SGD.

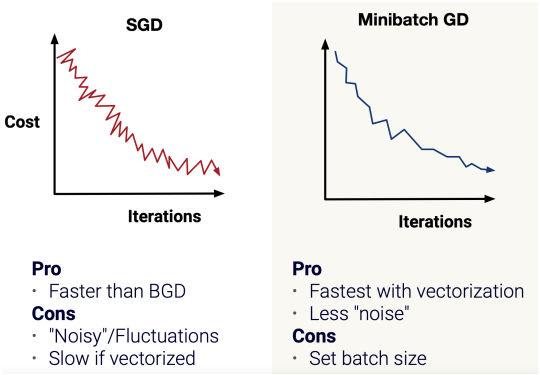


Figure 2.15: SGD and MBGD

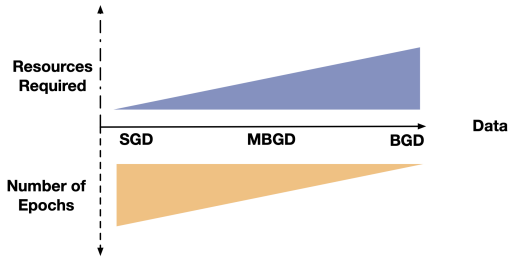


Figure 2.16: Comparison of Approaches

2.9 Weight Initialization

2.9.1 Xavier Initialization

Normal distribution with a scale variance by weights. Used on layers where either $TanH$ or $Sigmoid$ is present.

$$\text{Initialize weights for layer } l \text{ with: } W^{[l]} = W^{[l]} \sqrt{\frac{1}{n^{[l-1]}}}$$

where $n^{[l-1]}$ is the number of weights in the last layer. Each weight is sampled by $W_{j,i}^{[l]} \sim \mathcal{N}(0, 1)$

2.9.2 He Activation

Weight initialization for $ReLU$-powered network.

$$\text{Initialize weights for layer } l \text{ with: } W^{[l]} = W^{[l]} \sqrt{\frac{2}{n^{[l-1]}}}$$

where $n^{[l-1]}$ is the number of weights in the last layer. Each weight is sampled by $W_{j,i}^{[l]} \sim \mathcal{N}(0, 1)$

Chapter 3 Adaptive Optimization

3.1 Exponentially Weighted Moving Averages

How can we get an average across time?

1. (Simple) (weighted) Moving Average ((S)MA):

$$\bar{x}_{MA} = \frac{x_m + x_{m-1} + \cdots + x_{m-(n-1)}}{n} = \frac{1}{n}\sum_{i=0}^{n-1} x_{m-i}$$

2. Exponentially (weighted) Moving Averages (EMA):

$$EMA_t = \begin{cases} Y_1, & t = 1 \\ \beta \cdot Y_t + (1-\beta) \cdot EMA_{t-1}, & t > 1 \end{cases}$$

EMA is quicker to react: focuses more on recent events; SMA is slower to react: focuses on long series events.

3.2 Adaptive Learning Rates

Adaptive Learning Rate is a change to the learning rate while training a model to reduce the training time and improve output

3.2.1 Momentum

For a gradient descent with form:

$$\theta_{t+1} := \theta_t - v_t$$

where v_t is the **velocity** which is amplified gradient speed.

1. **SGD:**

$$v_t = \alpha \nabla_\theta J(\theta_t)$$

2. **SGD + Momentum:**

$$v_t = \rho v_{t-1} + \alpha \nabla_\theta J(\theta_t)$$

   where ρ is the **friction or momentum** which dampens the amount of the previous gradient included. Default: 0.9 for ~ 10 gradients.

With momentum, the learning rate is **decreased** if gradient direction changes and **increased** if gradient direction stays on same path.

3.2.2 Root Mean Square Propagation (RMSProp)

Decrease learning rate by EMA using squared gradient.

$$g_0 = 0 \quad \text{(Initial "gain")}$$

$$g_t = \alpha g_{t-1} + (1-\alpha)\nabla_\theta J(\theta_t)^2 \quad \text{(MA over gradient squared)}$$

$$\theta_t := \theta_{t-1} - \frac{\varepsilon}{\sqrt{g_t} + 1 \times 10^{-5}} v_t \quad (1 \times 10^{-5} \text{ is used to avoid division by } 0)$$

3.2.3 Adaptive Moment Estimation (ADAM)

Merges the momentum and RMSProp paradigms.

Novelty is a bias correction of 1st/2nd moments.

Focus is on learning rate annealing (start fast, decrease).

(1) **Initial:** $v_0 = 0, g_t = 0$.

(2) **Momentum-variant:** $v_t = \beta_1 v_{t-1} + (1-\beta_1)\nabla_\theta J(\theta_t)$

(3) **RMSProp:** $g_t = \beta_2 g_{t-1} + (1-\beta_2)\nabla_\theta J(\theta_t)^2$

(4) **Bias Correction:** $v'_t = \frac{v_t}{1-\beta_1^t}$, $g'_t = \frac{g_t}{1-\beta_2^t}$ (where β_i^t is the t^{th} power of β_i)

(5) **RMSProp + Momentum:** $\theta_t := \theta_{t-1} - \frac{\varepsilon}{\sqrt{g'_t}+1\times10^{-5}} v'_t$

Pytorch Code

```
torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08,
                 weight_decay=0, amsgrad=False, *, foreach=None,
                 maximize=False, capturable=False,
                 differentiable=False, fused=False)
```

Pseudocode

input : γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$ (objective)
λ (weight decay), *amsgrad*, *maximize*
initialize : $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$

for $t = 1$ **to** $\ldots$ **do**
  if *maximize* :
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
  else
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  if $\lambda \neq 0$
    $g_t \leftarrow g_t + \lambda \theta_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\widehat{m_t} \leftarrow m_t / \big(1 - \beta_1^t\big)$
  $\widehat{v_t} \leftarrow v_t / \big(1 - \beta_2^t\big)$
  if *amsgrad*
    $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
    $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t} / \big(\sqrt{\widehat{v_t}^{max}} + \epsilon\big)$
  else
    $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t} / \big(\sqrt{\widehat{v_t}} + \epsilon\big)$

return θ_t

Figure 3.1: Pseudocode of ADAM

Chapter 4 Convolutional Neural Network (CNN)

4.1 Convolution and Cross-correlation

Cross-correlation and convolution are both operations applied to images. Cross-correlation means sliding a kernel (filter) across an image. Convolution means sliding a flipped kernel across an image. **Most convolutional neural networks in machine learning libraries are actually implemented using cross-correlation**, but it doesn't change the results in practice because if convolution were used instead, the same weight values would be learned in a flipped orientation.

We have some *source pixels*, we use a *convolution kernel* (filter) to process the *source pixels*. The output is *destination pixels*.

Given the *source pixels* $\{\textbf{Image}(x, y) : x \in [-d_X, d_X], y \in [-d_Y, d_Y]\}$ and *convolution kernel* (filter) $\{K(i, j) : i, j \in [-d, d]\}$. $n_X = 2d_X + 1, n_Y = 2d_Y + 1$ are image dimension, $f = 2d + 1$ is the filter dimension. We can generate destination pixels in $(n_X - f + 1) \times (n_Y - f + 1) = (n_X - 2d) \times (n_Y - 2d)$

For $x \in [d + 1, n_X - d], y \in [d + 1, n_Y - d]$

1. **Convolution:**

$$\textbf{CONV}(x, y) = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \textbf{Image}(x - i, y - j)K(i, j)$$

2. **Cross-Correlation:**

$$\textbf{CrossCorrelation}(x, y) = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \textbf{Image}(x + i, y + j)K(i, j)$$

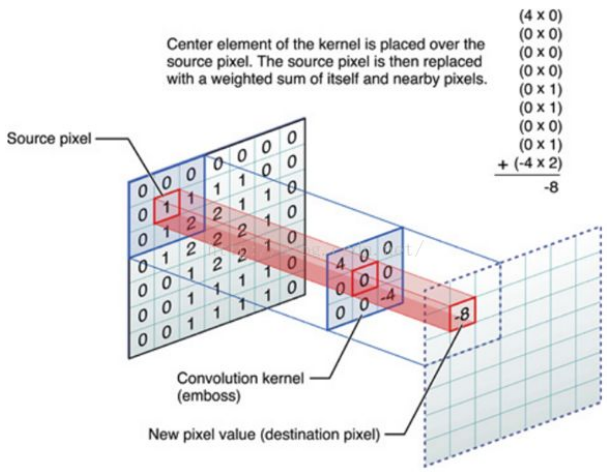


Figure 4.1: Convolution Used in CNN (actually cross-correlation)

Kernel

1. **Edge Detection:** vertical $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$; horizontal $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
2. Blur Pixel: $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$; Sharpen: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$; Identity: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$; Shift Pixel: $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

4.2 Padding (cover the border)

Definition 4.1

Padding refers to the extension of the input image by adding a border of pixels the image.

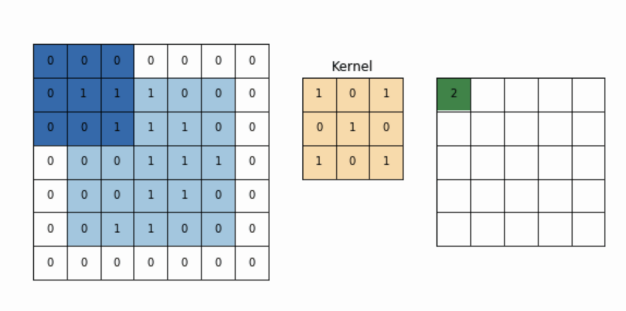


Figure 4.2: Padding: $p = 1$

For image $n \times n$ with filter $f \times f$ and padding p, the output has dimension

$$(n + 2p - f + 1) \times (n + 2p - f + 1)$$

In order for the output dimensions to be equivalent to the image dimension the padding value must be $p = \frac{f-1}{2}$

4.3 Stride

Definition 4.2

Stride refers to the sliding distance of the filter/kernel over spatial locations.

The default stride or strides in two dimensions is (1,1) for the height and the width movement.

For example, the stride can be changed to (2,2). This has the effect of moving the filter two pixels right for each horizontal movement of the filter and two pixels down for each vertical movement of the filter when creating the feature map.

The new dimension of the output would be

$$\left\lfloor \frac{n_X + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_Y + 2p - f}{s} + 1 \right\rfloor$$

4.4 Other Layer Types

4.4.1 Pooling

Definition 4.3

Pooling refers to the process of downsampling features by aggregating values at places in the feature map.

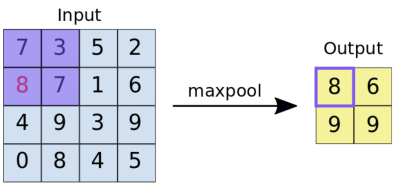


Figure 4.3: Example: maxpool

4.4.2 Unpooling

Definition 4.4

Unpooling refers to the process of upsampling features by recreating the dimensions of feature map pooled and placing the pooled values into their original location.

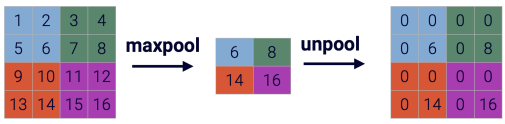


Figure 4.4: Example: maxpool+unpool

4.5 3D Convolution

Note the default filter has dimension $f \times f \times n'$ (n' is the number of filters in the previous layer)

SL.No		Activation Shape	Activation Size	# Parameters
1.	Input Layer:	(32, 32, 3)	3072	0
2.	CONV1 (f=5, s=1)	(28, 28, 8)	6272	608
3.	POOL1	(14, 14, 8)	1568	0
4.	CONV2 (f=5, s=1)	(10, 10, 16)	1600	3216
5.	POOL2	(5, 5, 16)	400	0
6.	FC3	(120, 1)	120	48120
7.	FC4	(84, 1)	84	10164
8.	Softmax	(10, 1)	10	850

Figure 4.5: 3D Convolution

Parameters Computing:

1. **CONV layer:** (shape of width of the filter * shape of height of the filter * number of filters in the previous layer+1)*number of filters

   (added 1 because of the bias term for each filter.)

$$(5 \times 5 \times 3 + 1) \times 8 = 608$$

2. **Fully Connected Layer (FC):** (current layer neurons number * previous layer neurons number)+1* current layer neurons number

$$120 \times 400 + 1 \times 120 = 48120$$

Chapter 5 Generative model

5.1 Autoencoders

Definition: *Self-supervised learning (SSL)* is a machine learning process where the model trains itself to learn one part of the input from another part of the input. It is a technique similar in scope to how humans learn to classify objects. SSL relies on unlabeled data to solve a task by splitting the task into at least two halves:

1. A decomposition into pseudo-labels by withholding some training data (self-supervised task/pretext task); and,
2. Reconstruction using either supervised or unsupervised learning.

For example, in natural language processing, if we have a few words, using self-supervised learning we can complete the rest of the sentence. Similarly, in a video, we can predict past or future frames based on available video data.

5.1.1 Basics

Definition 5.1

Autoencoders *are designed to take the input data, say x, and, then, predict the very same input x! In other words, the network trains itself to imitate its input so that its output is the same.* ♣

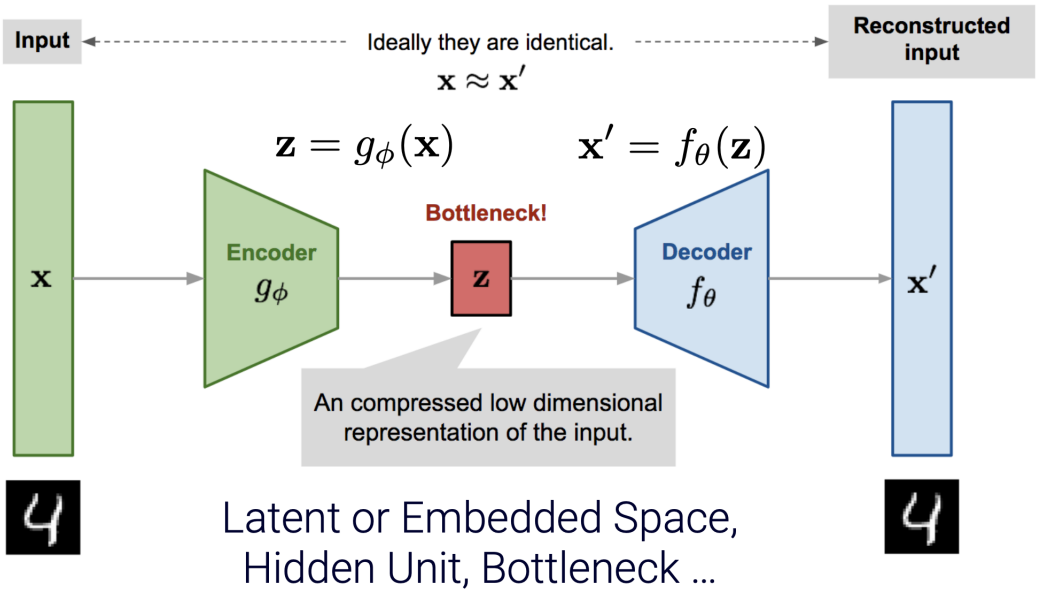


Figure 5.1: Autoencoders

Undercomplete autoencoders are defined as having a bottleneck layer dimension that is less than that of the input. e.g.

$$\dim(z) < \dim(x)$$

The lower dimention of bottleneck (hidden unit) avoids overfitting.

If the dimension is equal, the x will be completely transformed into x' which is often the same as the model learns nothing and may also be overfitted. Therefore, some other conditions are usually added to make the model only approximate its input, so that the model can really learn the hidden vector expression of the input samples and prevent overfitting.

5.1.2 PCA and Autoencoders

Principal Component Analysis (PCA) is using orthogonal basis to reduce dimensionality.

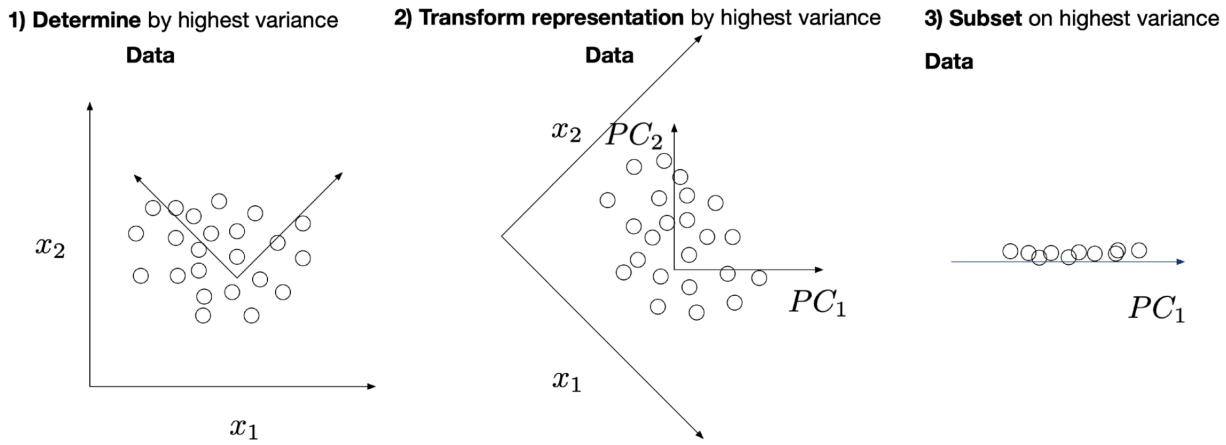


Figure 5.2: Principal Component Analysis (PCA)

Consider a single hidden layer linear autoencoder network with linear activations and MSE loss:

$$\vec{z} = W^{[1]}\vec{x} + \vec{b}^{[1]}$$

$$\vec{x}' = W^{[2]}\vec{z} + \vec{b}^{[2]}$$

$$L(\vec{x}, \vec{x}') = \|\vec{x} - \vec{x}'\|_2^2$$

If we have the $dim(m) < dim(n)$, then the problem will be a PCA without an orthogonality restriction on the weights.

Why bother with autoencoders if PCA exists? Dimensional Reductions (Addressing curse of dimensionality)

- Rarely are we seeking to use an auto encoder for solely a dimension reduction.

- In such cases, we probably would be better off with:

- Principal Component Analysis (PCA): If linear and desire global structure under a deterministic algorithm.
- T-distributed stochastic neighbour embedding (t-SNE): If non-linear and desire local structure under a randomized algorithm with dense structures.
- Uniform Manifold Approximation and Projection (UMAP): If non-linear and desire local structure under a randomized algorithm with sparse structures.

5.1.3 Transposed Convolutions (upscale method)

Definition: *Transposed Convolutions*, *uncov*, or *fractionally striped convolution* is a technique to upscale the feature map so that it matches in dimension with the input feature map.
(Sometimes erroneously called ”deconvolution”.)

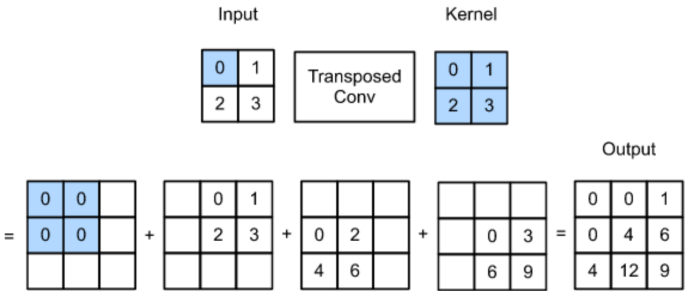


Figure 5.3: Transposed Convolutions

Comparison between Convolution and Transposed Convolution

Transposed Convolution with Stride

For a image $n \times n$ with filter $f \times f$, padding p and stride s, the dimension of the output is

$$(s(n-1)+f-2p) \times (s(n-1)+f-2p)$$

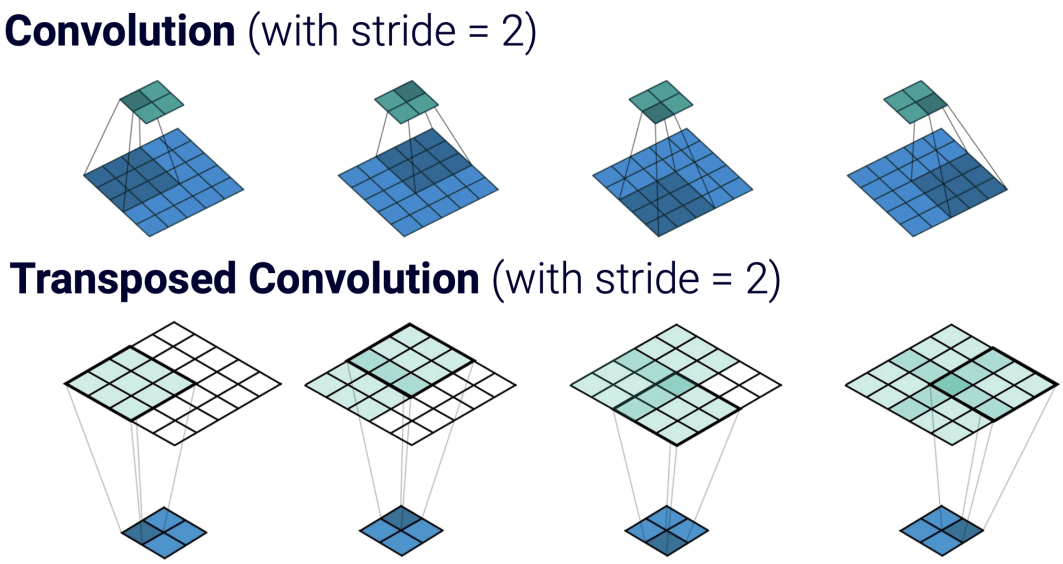


Figure 5.4: Comparison

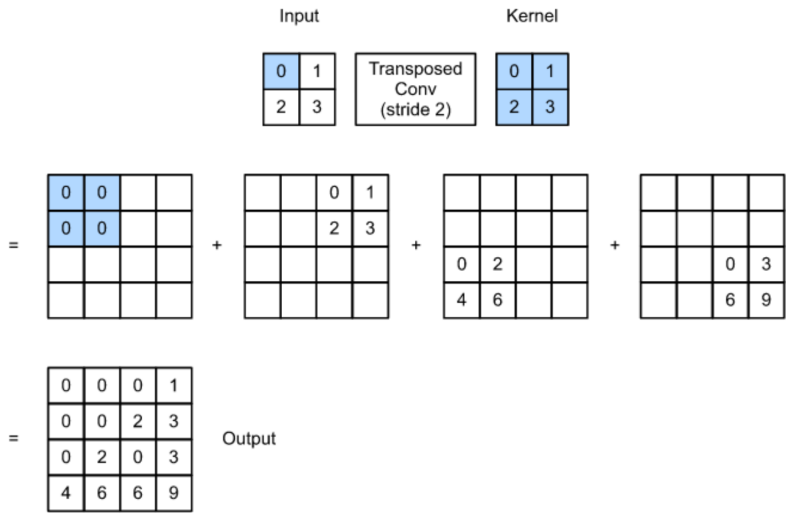


Figure 5.5: Transposed Convolution with Stride

Chapter 6 Statistical Inference

6.1 Basics

Given an observation $x \in X$, we want to estimate an unknown state $\theta \in S$ (not necessarily random). The θ can form x with $P_\theta(x)$. We use decision rule $\delta(x)$ to form an action (estimation of θ) $a = \hat{\theta}$.

Example:

(1) Binary hypothesis testing (detection) when $S = \{0, 1\}$ e.g. $P_0 \sim N(0, \sigma^2), P_1 \sim N(\mu, \sigma^2)$

(2) Multiple hypothesis testing (classification) when $S = \{1, 2, ..., n\}$

(3) (Estimation) when $S = \mathbb{R}$ e.g. $P_\theta \in N(\theta, \sigma^2)$

6.2 Decision Rule Examples

Binary HT Example

For the example Binary HT, $P_0 \sim N(0, \sigma^2), P_1 \sim N(\mu, \sigma^2)$: decision rule $\delta : \mathbb{R} \to \{0, 1\}$

We can find a τ such that $\delta(x) = \begin{cases} 1, & x \geq \tau \\ 0, & else \end{cases} = \mathbf{1}_{x \geq \tau}$. Howe to choose τ?

Type-I error probability: probability that θ is 0 but receive $\delta(x) = 1$.

$$P_I = P_0\{\delta(x) = 1\} = P_0\{x \geq \tau\} = Q\left(\frac{\tau}{\sigma}\right)$$

Type-II error probability: probability that θ is 1 but receive $\delta(x) = 0$.

$$P_{II} = P_1\{\delta(x) = 0\} = P_1(x < \tau) = Q(\frac{\mu - \tau}{\sigma})$$

Both P_I and P_{II} depends on τ. $Q(t) = \int_t^\infty \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} dx$

For $\tau = \frac{\mu}{2}$, $P_I = P_{II} = Q\left(\frac{\mu}{2\sigma}\right)$

Multiple HT Example

Consider three state $S = \{1, 2, 3\}$. We can find a τ such that $\delta(x) = \begin{cases} 1, & x < \tau_1 \\ 2, & \tau_1 \leq x \leq \tau_2 \\ 3, & x > \tau_2 \end{cases} = \mathbf{1}_{x \geq \tau}$.

Conditional Error Probabilities: probability that θ is i but receive $\delta(x) = j$ (6 types in this example)

$$P_i\{\delta(x) = j\}, \forall i \neq j$$

Estimation Example

Ex: $P_\theta \sim N(\theta, \sigma^2)$. Perform $\delta(x) = \hat{\theta}$ by using mean-squared error (MSE):

$$MSE = \mathbb{E}_\theta \left[(\delta(x) - \theta)^2 \right], \theta \in \mathbb{R}$$

6.3 Maximum-Likelihood Principle (state is norandom)

Maximum-Likelihood Principle

$$\hat{\theta} = \underset{\theta \in S}{\operatorname{argmax}} P_\theta(x) = \underset{\theta \in S}{\operatorname{argmax}} \ln P_\theta(x)$$

Applied to the binary example: $P_0 \sim N(0, \sigma^2), P_1 \sim N(\mu, \sigma^2)$.

$P_0(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, P_1(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. $\ln P_0(x) = c - \frac{x^2}{2\sigma^2}, \ln P_1(x) = c - \frac{(x-\mu)^2}{2\sigma^2}$.

Then, the rule can become

$$\hat{\theta} = \begin{cases} 0, & x^2 < (x-\mu)^2 \\ 1, & else \end{cases} = \mathbf{1}_{x^2 \geq (x-\mu)^2} = \mathbf{1}_{x \geq \frac{\mu}{2}}$$

Vector Observations

Observations $X = (x_1, x_2, ..., x_n)$, where i.i.d. $x_i \sim P_\theta$. Then

$$P_\theta(X) = \prod_{i=1}^{n} P_\theta(x_i), \ \ln P_\theta(X) = \sum_{i=1}^{n} \ln P_\theta(x_i)$$

$\ln P_0(x) = cn - \frac{\sum_{i=1}^n x_i^2}{2\sigma^2}, \ln P_1(x) = cn - \frac{\sum_{i=1}^n (x_i - \mu)^2}{2\sigma^2}$.

Then, the rule can become

$$\hat{\theta} = \begin{cases} 0, & \sum_{i=1}^n x_i^2 < \sum_{i=1}^n (x_i - \mu)^2 \\ 1, & else \end{cases} = \mathbf{1}_{\sum_{i=1}^n x_i^2 \geq \sum_{i=1}^n (x_i - \mu)^2} = \mathbf{1}_{\bar{x} \geq \frac{\mu}{2}}$$

where $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$. Under both H_0 and H_1, $\bar{x} \sim N(0, \frac{\sigma^2}{n})$.

Then, type I error prob and type II error prob are the same

$$P_I = P_0\{\bar{x} \geq \frac{\mu}{2}\} = P_{II} = P_1\{\bar{x} < \frac{\mu}{2}\} = Q\left(\frac{\mu\sqrt{n}}{2\sigma}\right)$$

Estimation $S = \mathbb{R}$

To estimate θ when $S = \mathbb{R}$

$$\begin{aligned}
&\max_{\theta\in\mathbb{R}} \sum_{i=1}^{n} \ln P_\theta(x_i) \\
\Leftrightarrow &\max_{\theta\in\mathbb{R}} \left[cn - \frac{\sum_{i=1}^{n}(x_i-\theta)^2}{2\sigma^2}\right] \\
\Leftrightarrow &\max_{\theta\in\mathbb{R}} \sum_{i=1}^{n}(x_i-\theta)^2 \Rightarrow \hat{\theta} = \bar{x}
\end{aligned}$$

Then, with $\bar{x} \sim N(\theta, \frac{\sigma^2}{n})$, the

$$MSE_\theta = \mathbb{E}_\theta\left(\bar{x}-\theta\right)^2 = \frac{\sigma^2}{n}$$

6.4 Bayesian Decision Rule (state is random)

6.4.1 Rules

Prior probability distribution $\pi(\theta)$,

Loss/cost function with action (estimation) a is $l(a,\theta)$. e.g.

1. (binary HT) Hamming/zero-one loss $l(a=\hat{\theta},\theta) = \mathbf{1}_{a\neq\theta}$
2. (estimation) Squared error loss $l(a=\hat{\theta},\theta) = (a-\theta)^2$; Absolute error loss $l(a,\theta) = |a-\theta|$.

Risk of decision rule δ:

$$R(\delta) = \mathbb{E}\left(l(\delta(X),\theta)\right)$$

where (X,θ) are random with prob $\pi(\theta), P_\theta$

Note: to help be consistent with machine learning notations, we use y to substitute θ.

The joint probability $P(x,y) = \pi(y)P_y(x) = P(x)\pi(y|x)$.

Example 6.1 The risk of decision $\delta(x)$ in Hamming/zero-one loss $l(a=\hat{y},y) = \mathbf{1}_{a\neq y}$

$$\begin{aligned}
R(\delta) &= \mathbb{E}(\mathbf{1}_{\delta(x)\neq y}) = \mathbb{E}[\delta(x)\neq y] \\
&= P(y=0)P[\delta(x)\neq 0|y=0] + P(y=1)P[\delta(x)\neq 1|y=1] \\
&= P(y=0)P[\delta(x)=1|y=0] + P(y=1)P[\delta(x)=0|y=1]
\end{aligned}$$

Bayes rule

$$\delta_B = \operatorname*{argmin}_\delta R(\delta)$$

Derive Bayes rule

$$\begin{aligned} R(\delta) &= \int_x \int_y P(x,y) l(\delta(x), y) dy dx \\ &= \int_x P(x) \int_y \pi(y|x) l(\delta(x), y) dy dx \end{aligned}$$

Solve optimization problem:

$$\min_{\delta} \int_x P(x) \int_y \pi(y|x) l(\delta(x), y) dy dx$$

this problem can be transformed into optimization problems for each $x \in S$

$$\min_{\delta(x)} \int_y \pi(y|x) l(\delta(x), y) dy$$

The problem becomes to compute $\pi(y|x)$. From $P(x,y) = \pi(y)P_y(x) = P(x)\pi(y|x)$, we know

$$\pi(y|x) = \frac{\pi(y)P_y(x)}{P(x)}$$

6.4.2 Maximum A Posteriori (MAP) Decision Rule (Binary example)

Example 6.2 Hamming/zero-one loss $l(a,y) = \mathbf{1}_{a \neq y}$

Maximum A Posteriori (MAP) Decision Rule:

Optimization problem is

$$\delta(x) = \operatorname*{argmin}_a \sum_{y=0,1} \pi(y|x) \mathbf{1}_{a \neq y} dy = \operatorname*{argmax}_{y \in \{0,1\}} \pi(y|x)$$

$$\Rightarrow \sum_{y=0,1} \pi(y|x) \mathbf{1}_{\delta(x) \neq y} dx = \min_a \sum_{y=0,1} \pi(y|x) \mathbf{1}_{a \neq y} dy = \min\{\pi(1|x), \pi(0|x)\}$$

Likelihood ratio: $L(x) = \frac{P_1(x)}{P_0(x)}$

Likelihood ratio test: threshold $\tau = \frac{\pi(0)}{\pi(1)}$. If $L(x) > \tau$ accept H_1 (equivalent to $P_1(x)\pi(1) > P_0(x)\pi(0)$ which is also equivalent to comparing $\pi(y|x)$).

In this rule the whole optimization problem also goes to

$$\begin{aligned} R(\delta_{MAP}) &= \int_x P(x) \sum_{y=0,1} \pi(y|x) \mathbf{1}_{\delta(x) \neq y} dx \\ &= \int_x P(x) \min\{\pi(1|x), \pi(0|x)\} dx \end{aligned}$$

6.4.3 Minimum Mean Squared Error (MMSE) Rule ($\mathbb{R}^n$ example)

Example 6.3 (estimation) Squared error loss $l(a,y) = (a-y)^2$.

Minimum Mean Squared Error (MMSE) Rule:

Optimization problem is $\delta(x) = \text{argmin}_a \int_y \pi(y|x)(a-y)^2 dy$

$$0 = \int_y \pi(y|x)(\delta_B(x) - y)dy = \delta_B(x) - \mathbb{E}\left[Y|X=x\right]$$

$$\Rightarrow \delta_B(x) = \mathbb{E}\left[Y|X=x\right]$$

which is called **conditional mean estimation**.

In this rule the whole optimization problem also goes to

$$R(\delta_{MMSE}) = \int_x P(x) \int_y \pi(y|x)(y - \mathbb{E}\left[Y|X=x\right])^2 dydx = \mathbb{E}_X Var\left[Y|X=x\right]$$

Gaussian case: If $X \in \mathbb{R}^n$ and (Y, X) are jointly Gaussian, then the conditional mean is a linear function of x, also called linear MMSE estimator.

$$\mathbb{E}\left[Y|X=x\right] = \mathbb{E}[Y] + Cov(Y,X)Cov(X)^{-1}(x - \mathbb{E}[X])$$

and the posterior risk is independent of x:

$$Var\left[Y|X=x\right] = Var[Y] - Cov(Y,X)Cov(X)^{-1}Cov(X,Y)$$

Note: MMSE estimator coincides with the MAP estimator for Gaussian Variables.

6.5 Comparison

Maximum-Likelihood Principle (state is nonrandom): $\delta_{ML}(x) = \text{argmax}_y P_y(x)$.

Maximum A Posteriori (MAP) Decision Rule (state is random): $\delta_{MAP}(x) = \text{argmax}_y \pi(y|x) = \text{argmax}_y\{\pi(y|x), P_y(x)\}$

Chapter 7 Machine Learning in Inference

Instead of given a prior distribution of Y, we are given a **training set** $T = (X_i, Y_i)_{i=1}^n$ where i.i.d. $(X_i, Y_i) \sim P$. (Distribution P is unknown).

Risk: $R(\delta) = \mathbb{E}_P [l(\delta(X), Y)]$

The ture optimal decision rule is

$$\delta_B = \underset{\delta}{\operatorname{argmin}} \mathbb{E}_P [l(\delta(X), Y)]$$

which is can't be computed since we don't know how actually P is.

7.1 Empirical Risk Minimization (ERM)

Instead of computing optimal decision rule with P, we compute the optimal decisioin rule in the training set:

$$\hat{\delta}_n = \underset{\delta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} l(\delta(X_i), Y_i)$$

The corresponding risk is $R(\hat{\delta}_n) = \mathbb{E}_P \left[l(\hat{\delta}_n(X), Y) \right]$. $\Delta R(\hat{\delta}_n) = R(\hat{\delta}_n) - R(\delta) > 0$ always holds.

Consistency: if $\Delta R(\hat{\delta}_n) \to 0$ as $n \to \infty$.

7.1.1 Example: Linear MMSE (LMMSE) estimator

Use the decision role in the class of $\delta(x) = wx$. To find the linear MMSE (LMMSE) estimation $\delta^*(x) = w^* x$:

$$w^* = \underset{w}{\operatorname{argmin}} \mathbb{E}_P \left[(wX - Y)^2 \right] = \frac{\mathbb{E}[XY]}{\mathbb{E}[X^2]}$$

The rule that minimizes the **empirical risk** is

$$\hat{w} = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (wX_i - Y_i)^2 = \frac{\frac{1}{n}\sum_{i=1}^n X_i Y_i}{\frac{1}{n}\sum_{i=1}^n X_i^2}$$

The risk of the optimal rule $\delta^* = w^* x$ is $R(\delta^*)$ and the empirical risk under rule $\hat{\delta}(x) = \hat{w}x$ is $R(\hat{\delta}(x))$. $R(\hat{\delta}) > R(\delta^*)$ always holds, and

$$R(\hat{\delta}) \to R(\delta^*) \text{ as } n \to \infty$$

According to CLT:

$$\sqrt{n} \left(\frac{1}{n} \sum_i X_i Y_i - \mathbb{E}(XY) \right) \xrightarrow{d} N(0, \sigma^2)$$

$$\sqrt{n} \left(\frac{1}{n} \sum_i X_i^2 - \mathbb{E}(X^2) \right) \xrightarrow{d} N(0, \sigma^2)$$

Then,

$$\frac{1}{n}\sum_i X_i^2 = \mathbb{E}(X^2) + O(\frac{1}{\sqrt{n}})$$
$$\frac{1}{n}\sum_i X_i Y_i = \mathbb{E}(XY) + O(\frac{1}{\sqrt{n}})$$

which means the error of estimators

$$\hat{w} = \frac{\mathbb{E}(X^2) + O(\frac{1}{\sqrt{n}})}{\mathbb{E}(XY) + O(\frac{1}{\sqrt{n}})} = w^* + O(\frac{1}{\sqrt{n}})$$
$$\hat{w} - w^* = O(\frac{1}{\sqrt{n}})$$

and the error of the risks:

$$\begin{aligned} R(\hat{\delta}) - R(\delta^*) &= \mathbb{E}_P[\hat{w}X - Y]^2 - \mathbb{E}_P[w^*X - Y]^2 \\ &= \mathbb{E}_P[(\hat{w} - w^*)X + w^*X - Y]^2 - \mathbb{E}_P[w^*X - Y]^2 \\ &= \mathbb{E}_P[(\hat{w} - w^*)X]^2 + 2(\hat{w} - w^*)\mathbb{E}_P[X(w^*X - Y)] \\ &= (\hat{w} - w^*)\mathbb{E}_P(X^2) = O\left(\frac{1}{n}\right) \end{aligned}$$

Complexity:

Definition 7.1
A sequence $f(n)$ *is* $O(1)$ *if* $\lim_{n\to\infty} f(n) < \infty$.

Definition 7.2
A sequence $f(n)$ *is* $O(g(n))$ *if* $\frac{f(n)}{g(n)}$ *is* $O(1)$.

Definition 7.3
A sequence $f(n)$ *is* $o(1)$ *if* $\lim_{n\to\infty} \sup f(n) = 0$.

Definition 7.4
A sequence $f(n)$ *is* $o(g(n))$ *if* $\lim_{n\to\infty} \sup \frac{f(n)}{g(n)} = 0$.

Definition 7.5
A sequence $f(n)$ *is asymptotic to* $g(n)$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$. *(This is denoted by* $f(n) \sim g(n)$ *as* $a \to \infty$*)*

7.1.2 Penalized ERM

$\delta(x) = \sum_{j=1}^{J} w_j x^j$

Pick $J = d$ and use ERM with d dimensional w:

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n}[w^T X_i - Y_i]^2$$

Approach 1: Fix $d << n$, use ERM.

Approach 2: (**Penalized ERM**)

$$\min_{\delta}[R_{emp}(\delta) + J(\delta)]$$

($J(\delta)$ is regularization (penalty) term)

7.2 Stochastic Approximation

Robbins and Monro (95)

Problem: Find a root of function $h(x)$. ($f(x) = 0$)

We do not observe $h(x)$ directly, but we observe $Y \sim P_x$ with

(1) $\mathbb{E}[Y|X = x] = h(x)$

(2) $(Y|X = x) - h(x)$ is bounded

Example 7.1 $Y = X + Z$ with $\mathbb{E}[Z] = 0$ and Z is bounded.

Assumptions: 1. $h'(x^*) > 0$; 2. x^* is the unique root of h.

SA Algorithm

- Pick Sequence $\{a_n\}$ such that $\sum_{n=1}^{\infty} a_n = \infty$ and $\sum_{n=1}^{\infty} a_n^2 < \infty$ (should converge to 0 but not too quick) e.g. $a_n = n^{-\alpha}$ when $\alpha \in (\frac{1}{2}, 1]$.
- Initialize X_1
- Update for $n = 1, 2, ..., Y_n \sim P(\cdot|X = X_n)$

  $$X_{n+1} = X_n - a_n Y_n$$

  until convergence.

Theorem 7.1

Under these assumptions

$$X_n \xrightarrow{m.s.} x^* \text{ as } n \to \infty$$

i.e., $\mathbb{E}(X_n - x^*)^2 \to 0$ *as* $n \to \infty$.

- **Performance Measure** (Convergence rate): the root mean squared error (RMSE) $e_n = \sqrt{\mathbb{E}[(X_n - x^*)^2]}$.
- Projections: If x is constrained to live in an interval I, the update rule becomes

  $$X_{n+1} = \text{Proj}_x[X_n - a_n Y_n]$$

- Averaging:

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i = \frac{X_n}{n} + \bar{X}_{n-1}\frac{n-1}{n}$$

(nicer graph) (The benefits of this smoothing operation are mostly seen in the initial stages of the SA recursion, and do not improve the convergence rate.)

Example 7.2 Let $h(x) = x$, in which case $x^* = 0$. $Y_n = h(X_n) + Z_n = X_n + Z_n$ where noise Z_n is independent of Y_n with $\mathbb{E}[Z_n] = 0, Var(Z_n) = 1$ and Z_n is bounded.

Then,

$$\begin{aligned} X_{n+1} &= X_n - a_n(X_n + Z_n) \\ &= (1 - a_n)X_n - a_n Z_n \end{aligned}$$

The MSE,

$$\begin{aligned} e_{n+1}^2 = \mathbb{E}(X_{n+1} - x^*)^2 &= \mathbb{E}(X_{n+1})^2 \\ &= \mathbb{E}[(1 - a_n)X_n - a_n Z_n]^2 \\ &= (1 - a_n)^2 \mathbb{E}X_n^2 + a_n^2 \mathbb{E}Z_n^2 \\ &= (1 - a_n)^2 e_n^2 + a_n^2 \end{aligned}$$

Pick $a_n = n^{-\alpha}$, where $\alpha \in (\frac{1}{2}, 1]$

$$\Rightarrow e_{n+1}^2 = (1 - n^{-\alpha})^2 e_n^2 + n^{-2\alpha}$$

Guess: $e_n = \sqrt{c}n^{-\beta} + H.O.T$

$$c(n+1)^{-2\beta} + H.O.T = (1-n^{-\alpha})^2 cn^{-2\beta} + n^{-2\alpha} + H.O.T$$

(where $(n+1)^{-2\beta} = n^{-2\beta}(1+\frac{1}{n})^{-2\beta} = n^{-2\beta}[1 - 2\beta n^{-1} + O(n^{-2})]$, by Taylor)

$$cn^{-2\beta} - 2c\beta n^{-1-2\beta} + H.O.T = (1-n^{-\alpha})^2 cn^{-2\beta} + n^{-2\alpha} + H.O.T$$

$$-2c\beta n^{-1-2\beta} + H.O.T = -2cn^{-\alpha-2\beta} + n^{-2\alpha} + H.O.T$$

(For $\alpha < 1$), $-2c\beta n^{-1-2\beta}$ is not dominant term.

$$H.O.T = -2cn^{-\alpha-2\beta} + n^{-2\alpha} + H.O.T$$

Identify Power: $2\alpha = \alpha + 2\beta \Rightarrow \beta = \frac{\alpha}{2}$ and $c = \frac{1}{2}$

(For $\alpha = 1$), there are three dominant terms.

$$-2c\beta n^{-1-2\beta} + H.O.T = -2cn^{-1-2\beta} + n^{-2} + H.O.T$$

Identify Power: $2 = 1 + 2\beta \Rightarrow \beta = \frac{1}{2}$ and $-2c\beta = -2c + 1 \Rightarrow c = 1$

$$e_n^2 \sim cn^{-2\beta}$$

To let the convergence rate as fast as possible, we want the β to be as large as possible. Since $\beta = \frac{\alpha}{2}$, we pick the highest $\alpha = 1 \Rightarrow \beta = \frac{1}{2}, c = 1$.

$$e_n = O(n^{-\frac{1}{2}}) \text{ with } a_n \sim \frac{1}{n}$$

Example 7.3 Let $h(x) = x^3$, in which case $x^* = 0$. $Y_n = h(X_n) + Z_n = X_n^2 + Z_n$ where noise Z_n is independent of Y_n with $\mathbb{E}[Z_n] = 0, Var(Z_n) = 1$ and Z_n is bounded.

Then,

$$\begin{aligned} X_{n+1} &= X_n - a_n(X_n^3 + Z_n) \\ &= X_n - a_n X_n^3 - a_n Z_n \end{aligned}$$

Pick $a_n = n^{-\alpha}, \alpha \in (\frac{1}{2}, 1] \Rightarrow \beta = \frac{1}{6}, \alpha = \frac{2}{3} \Rightarrow e_n \sim O(n^{-\frac{1}{6}})$

7.3 Stochastic Gradient Descent (SGD)

Solve $\min_{x \in \mathbb{R}^n} f(x)$.

We only use a **noisy version** $g(x, z)$ of $f(x)$, where $\mathbb{E}_z[g(x, z)] = f(x)$.

$$\mathbb{E}_z[\nabla_x g(x, z)] = \nabla_x \mathbb{E}_z[g(x, z)] = \nabla f(x)$$

Also pick sequence $\{a_n\}$ such that $\sum_{n=1}^{\infty} a_n = \infty$ and $\sum_{n=1}^{\infty} a_n^2 < \infty$.

SGD

- Initialize X_1
- Update for $n = 1, 2, ...,$

$$X_{n+1} = X_n - a_n \nabla g(X_n, Z_n)$$

Example 7.4 $f(x) = \frac{1}{2}x^2, x \in \mathbb{R}$. Let Z be a random variable with $\mathbb{E}(Z) = 0, Var(Z) = 1$.

$$g(x, Z) = \frac{1}{2}(x + Z)^2 - \frac{1}{2}$$

$$\mathbb{E}[g(x, Z)] = \frac{1}{2}x^2 = f(x)$$

$$\nabla_x g(x, Z) = x + Z \Rightarrow \mathbb{E}[\nabla_x g(x, Z)] = \nabla f(x)$$

$$X_{n+1} = X_n - a_n(X_n + Z_n)$$

which is the same as the stochastic approximation.

Main Results: (Suppose the unique minimum is x^*)

(1) Convergence: $e_n \to 0$ as $n \to \infty$.

(2) Convergence Rate: To achieve $\mathbb{E}[f(X_n)] - f(x^*) < \varepsilon$, we need $n = O(\frac{1}{\varepsilon})$ if f is twice continuously differentiable and strongly convex.

GD has linear convergence $\Rightarrow e_n = O(e^{-cn})$; Solve $\varepsilon = O(e^{-cn}) \Rightarrow n = O(\ln \frac{1}{\varepsilon})$. (**SGD is much worse than GD**, cost more.)

7.4 SGD Application to Empirical Risk Minimization (ERM)

ERM problem is

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} L(\delta_w(X_i), Y_i)$$

$R_{emp}(w) = \frac{1}{n} \sum_{i=1}^{n} L(\delta_w(X_i), Y_i)$ is the empirical risk (e.g. $\delta_w(x) = w^T x$, $L(\hat{y}, y) = (\hat{y} - y)^2$) To make w more visible, we can write

$$R_{emp}(w) = \frac{1}{n} \sum_{i=1}^{n} L(\delta_w(X_i), Y_i) = \frac{1}{n} \sum_{i=1}^{n} Q(X_i, Y_i, w)$$

For penalized ERM we would similarly have

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} Q(X_i, Y_i, w) + J(w)$$

$J(w)$ is the penalty (regularization) term.

In the problem $\min_{W \in \mathbb{R}^n} R_{emp}(w) = \frac{1}{n} \sum_{i=1}^{n} Q(X_i, Y_i, w)$

7.4.1 Different Gradient Descent for ERM

GD

- Initialize W_1
- Update for $k \geq 1$, Update: $W_{k+1} = W_k - a_k \frac{1}{n} \sum_{i=1}^{n} \nabla Q(X_i, Y_i, W_k)$

Computational cost: The computational cost of GD is $O(dn)$ operations per iteration. Since GD has exponential convergence, the number of iterations needed to reach an optimization error of ρ is $O(\log \frac{1}{\rho})$. Hence GD incurs a total computational cost of $O(dn \log \frac{1}{\rho})$ to reach a solution W_k such that $R_{emp}(W_k) \leq \min_W R_{emp}(W) + \rho$

SGD

- Initialize W_1
- Update for $k \geq 1$,

Step 1: Pick i uniformly over $\{1, ..., n\}$

Step 2: $W_{k+1} = W_k - a_k \nabla Q(X_i, Y_i, W_k)$

Computational cost: After k iterations, $\mathbb{E}[R_{emp}(W_k)] \leq \min_W R_{emp}(W) + \rho$, for $k = O(\frac{1}{\rho})$ and f twice differentiable and strongly convex. The cost per iteration is $O(d)$ (independent of n), so the total computational cost is $O(\frac{d}{\varepsilon})$.

7.4.2 Constraints on Learning Problem

Why achieving a low value of ρ is useful (low error of $R_{emp}(\cdot)$), since the cost function $R_{emp}(\cdot)$ is only a surrogate for the actual risk $R(\cdot)$?

Typically, $d = O(n^b)$ where $0 < b < 1$.

For any numerical algorithm producing a decision rule $\tilde{\delta_n}$, the excess risk (compared to Bayes rule δ_B) can be expressed as the sum of three terms:

$$\begin{aligned}\Delta R(\tilde{\delta_n}) &= R(\tilde{\delta_n}) - R(\delta_B) \\ &= [R(\tilde{\delta_n}) - R(\hat{\delta_n})] + [R(\hat{\delta_n}) - R(\delta^*)] + [R(\delta^*) - R(\delta_B)]\end{aligned}$$

where

$$\begin{aligned}\delta_B &= \text{ Bayes rule} \\ \delta^* &= \text{ best rule in } D = \operatorname*{argmin}_{\delta \in D} R(\delta) \\ \hat{\delta_n} &= \operatorname*{argmin}_{\delta \in D} R_{emp}(\delta) \\ \tilde{\delta_n} &= \text{ solution of the algorithm after } k \text{ iterations}\end{aligned}$$

(Note: D is the set of all available decision rule in approximation (e.g. all linear parameters $\{W, b\}$), which can't be better than Bayes rule)

The expected excess risk

$$\epsilon = \mathbb{E}[\Delta R(\tilde{\delta_n})] = \underbrace{\mathbb{E}[R(\tilde{\delta_n}) - R(\hat{\delta_n})]}_{\text{Comp. Error}=\rho} + \underbrace{\mathbb{E}[R(\hat{\delta_n}) - R(\delta^*)]}_{\text{Est. Error}=O(\frac{d}{n})} + \underbrace{\mathbb{E}[R(\delta^*) - R(\delta_B)]}_{\text{Approx. Error}=O(d^{-\beta})}$$

Estimation error increases as d increases, but *approximation error* decreases as d increases. To minimize the excess risk, we want to balance the last two items, that is $O(\frac{d}{n}) = O(d^{-\beta})$: solve

$$\frac{d}{n} = d^{-\beta} \Rightarrow d^{1+\beta} = n \Rightarrow d = n^{\frac{1}{1+\beta}}$$

$$\Rightarrow \text{ the last two items } O(\frac{d}{n}) = O(d^{-\beta}) = O(n^{-\gamma})$$

where $\gamma = \frac{\beta}{1+\beta} \in (0, 1]$ is a constant.

To balance the three items, we want

$$\rho = O(n^{-\gamma}) \Rightarrow n = O(\rho^{-\frac{1}{\gamma}}) \text{ and } d = O(n^{\frac{1}{1+\beta}})$$

The update rule $W_{k+1} = W_k - a_k \nabla Q(X_k, Y_k, W_k)$, where $i \sim \text{Uniform}\{1, 2, ..., n\}$

Relation to Online Learning: When the training data are made available sequentially (instead of in a batch as assumed here), online learning can be used to sequentially learn the decision rules (or the weights that parameterize the decision rule).

Variations on Basic SGD: mini batch: replace S by a subset B and n by $|B|$

$$\frac{1}{|B|} \sum_{i \in B} Q(X_i, Y_i, W_k)$$

Averaging SGD:

$$\bar{W}_n = \frac{1}{n} \sum_{i=1}^{n} W_i = \frac{W_n}{n} + \bar{W}_{n-1} \frac{n-1}{n}$$

SVRG (Stochastic Variance Randomed Gradient): R. Johnson and T. Zhang, "Accelerating Stochastic Gradient Descent using Predictive Variance Reduction," *Proc. NIPS* 2013.

Unsupervised learning: If no explanatory variable X is present, the problem reduces to

$$\min_w \frac{1}{n} \sum_{i=1}^{n} Q(Y_i, w)$$

which finds applications to various unsupervised learning problems. For instance the k-means clustering algorithm partitions n data points $y_i, 1 \le i \le n$ in $\mathbb{R}^d$ into k clusters with centroids $w_j, 1 \le j \le k$, in a way that minimizes the within-cluster sum-of-squares: WCSS $= \frac{1}{n} \sum_{i=1}^{n} \min_{1 \le j \le k} \|y_i - w_j\|^2$. Using the formalism of (7), we have $Q(y, w) = \min_{1 \le j \le k} \|y - w_j\|^2$ where $y \in \mathbb{R}^d$ and $w = \{w_j\}_{j=1}^{k} \in \mathbb{R}^{d \times k}$ is a matrix whose k columns are the centroid vectors.

Chapter 8 Stochastic Integration Methods

Integral $I = \int_X f(x)dx$.

8.1 Deterministic Methods (Better in Low Dimension)

8.1.1 Riemann Integration

Riemann integral: approximation integral I of $f(x)$ in $[a, b]$ with

$$\hat{I}_n = \sum_{i=1}^{n} \underbrace{(x_i - x_{i-1})}_{\frac{b-a}{n}} f(x_i)$$

where $x_i = a + \frac{b-a}{n} i$. We can also denote $\hat{f}(x) = f(x_i)$ if $x \in (x_{i-1}, x_i]$

The error $|\hat{I}_n - I| = \int_a^b |\hat{f}(x) - f(x)|dx$

Assume f is differentiable and $\max_x |f'(x)| = c < \infty$, then $|\hat{f}(x) - f(x)| \leq \frac{b-a}{n} c$

$$\Rightarrow |\hat{I}_n - I| \leq \int_a^b \frac{b-a}{n} c \, dx = \frac{(b-a)^2}{n} c$$

That is $n \sim O\left(\varepsilon^{-1}\right)$

8.1.2 Trapezoidal Rule

Using average can be better.

$$\hat{I}_n = \sum_{i=1}^{n} \underbrace{(x_i - x_{i-1})}_{\frac{b-a}{n}} \frac{f(x_i) + f(x_{i-1})}{2}$$

The upper bound of error is $|\hat{I}_n - I| \leq \frac{C}{n^2}$ for some constant C.

That is $n \sim O\left(\varepsilon^{-\frac{1}{2}}\right)$

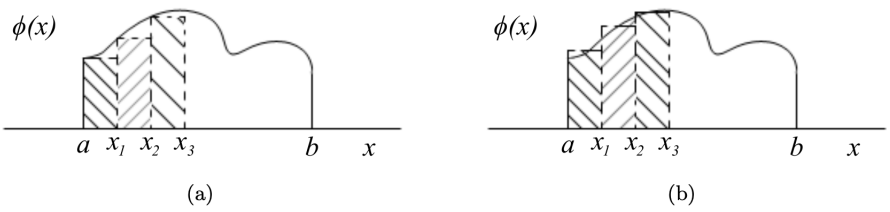


Figure 8.1: (a) Riemann approximation; (b) Trapezoidal approximation.

8.1.3 Multidimensional Integration

When we want to do intergral in high dimention, it will be really hard.

For $d-$dimensional integrals, the trapezoidal rule yields an approximation error $|\hat{I}_n - I| \leq \frac{C}{n^{\frac{2}{d}}}$ for some constant C. That is $n \sim O\left(\varepsilon^{-\frac{d}{2}}\right)$. n needs to increase exponentially with d to achieve a target approximation error ε. This phenomenon is known as the curse of dimensionality.

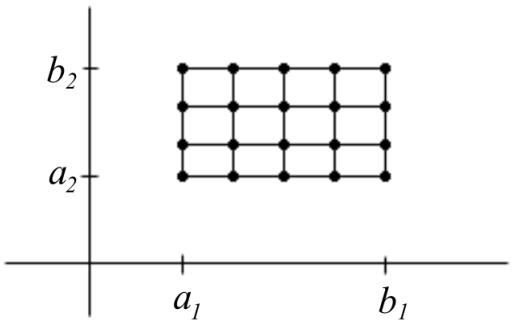


Figure 8.2: Two-dimensional integration using regular grid.

8.2 Stachastic Methods (Better in High Dimension)

8.2.1 Classical Monte Carlo Integration

Compute the expetation

$$\xi = \mathbb{E}_p[h(x)] = \int_X \underbrace{p(x)h(x)}_{f(x)} dx$$

The methods described below can be used to solve the following problems: (1) General $\int_X f$; (2) Compute the probability of falling into a subset $a \subset X : P(a) = \int_a p(x)dx$, where $h(x) = \mathbf{1}_{x \in a}$

The Monte Carlo approach is as follows: Given $X_1, X_2, \cdots, X_n$ drawn i.i.d from the pdf p, estimate ξ by the empirical average

$$\hat{\xi}_n = \frac{1}{n} \sum_{i=1}^{n} h(X_i)$$

$\mathbb{E}_p[\hat{\xi}_n] = \mathbb{E}_p[h(X)] = \xi$. $\hat{\xi}_n \xrightarrow{a.s.} \xi$ as $n \to \infty$ by SLLW.

$Var(\hat{\xi}_n - \xi) = Var(\hat{\xi}_n) = \frac{1}{n} Var[h(x)] = O\left(\frac{1}{n}\right) \Rightarrow sd(\hat{\xi}_n) = \frac{\sqrt{Var[h(x)]}}{\sqrt{n}}$

That is $n \sim O\left(n^{-\frac{1}{2}}\right)$

The stochastic methods **outperform** when the deterministic ones for dimensions $d > 4$ and are **worse** for $d < 4$.

8.2.2 Importance Sampling

Draw $X_i, i = 1, ..., n$ i.i.d from pdf q

$$\hat{\xi}_n = \frac{1}{n}\sum_{i=1}^{n}\frac{p(X_i)}{q(X_i)}h(X_i)$$

It is an unbiased estimator of ξ

$$\mathbb{E}[\hat{\xi}_n] = \mathbb{E}_q[\frac{p(X_i)}{q(X_i)}h(X_i)] = \int_X p(x)h(x)dx = \xi$$

$\hat{\xi}_n \xrightarrow{a.s.} \xi$ as $n \to \infty$ by SLLW.

Its variance is

$$\begin{aligned} Var_q(\hat{\xi}_n) &= \frac{1}{n}Var_q\left[\frac{p(X_i)}{q(X_i)}h(X_i)\right] \\ &= \frac{1}{n}\left(\int_X \frac{p^2(x)}{q(x)}h^2(x)dx - \xi^2\right) \end{aligned}$$

The idea of importance sampling is to find a good q such that

$$Var_q(\hat{\xi}_n) < Var_p(\hat{\xi}_n)$$

Error Meausre

The *relative error* of the importance-sampling estimator is defined as

$$\delta_{\text{rel}}\left(\hat{\xi}_n\right) \triangleq \frac{\sqrt{\text{Var}_q\left(\hat{\xi}_n\right)}}{\xi} = \sqrt{\frac{\text{Var}_q\left[\frac{p(X)}{q(X)}h(X)\right]}{\xi^2 n}}.$$

The *number* of simulations needed to achieve a relative error of δ is

$$n_{IS}(\delta) = \frac{\text{Var}_q\left[\frac{p(X)}{q(X)}h(X)\right]}{\xi^2\delta^2}.$$

The *gain* relative to a Monte Carlo simulation is defined as

$$\Gamma = \frac{n_{MC}(\delta)}{n_{IS}(\delta)} = \frac{\text{Var}_p[h(X)]}{\text{Var}_q\left[\frac{p(X)}{q(X)}h(X)\right]}$$

In the example of $\xi = P(a), h(x) = \mathbf{1}_{x\in a}$: Suppose $\xi = P(a) \approx 10^{-9}$ (small), $\hat{\xi}_n = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{X_i\in a}$. $\mathbf{1}_{X_i\in a}$ is $Bernoulli(\xi)$. We have $Var(\hat{\xi}_n) = \frac{\xi(1-\xi)}{n}$.

We can use relative error to measure

$$\delta_{\text{rel}}\left(\hat{\xi}_n\right) = \frac{\sqrt{Var(\hat{\xi}_n)}}{\xi} = \sqrt{\frac{1-\xi}{n\xi}}$$

and the number of simulation need to get relative error δ is

$$n_{IS}(\delta) = \frac{1-\delta}{\xi\delta^2}.$$

Find the optimal q:

$$\min_q \int_X \frac{p^2(x)}{q(x)} h^2(x) dx - \xi^2$$

write

$$\int_X \frac{p^2(x)}{q(x)} h^2(x) dx - \xi^2 = \mathbb{E}_q \left[\left(\underbrace{\frac{p(x)}{q(x)} h(x)}_{Z} \right)^2 \right]$$

Since x^2 is convex function, by Jensen's inequality

$$\mathbb{E}_q \left[\left(\underbrace{\frac{p(x)}{q(x)} h(x)}_{Z} \right)^2 \right] \geq \left(\mathbb{E}_q \left[\underbrace{\frac{p(x)}{q(x)} h(x)}_{Z} \right] \right)^2$$

This equality holds if and only if $\frac{p(x)}{q(x)} h(x) = \alpha, \forall x \in X$, α is a constant.

Since q is pdf., we can infer

$$q(x) = \frac{p(x)h(x)}{\int_X p(x)h(x)dx}$$

which is as hard as the original problem. In practice, one is content to find a "good" q that assigns high probability to the important region where $p(x)h(x)$ is large. Ideally the ratio $\frac{p(x)}{q(x)} h(x)$ would be roughly constant over X.

Chapter 9 Bootstrap (not enough data)

Problem: analyze the performance of an estimator $\hat{\theta}_n(\vec{Y})$, $\vec{Y} = (Y_1, Y_2, ..., Y_n)$ taken i.i.d. from distribution P.

e.g. $P_\theta = N(0,1), \hat{\theta}_n = \frac{1}{n}\sum_{i=1}^n Y_i$

Assume θ is a scalar parameter. Performance: (1) Bias $\mathbb{E}_\theta[\hat{\theta}_n(\vec{Y})] - \theta$; (2) Variance $\mathbb{E}_\theta[\hat{\theta}_n^2(\vec{Y})] - \mathbb{E}_\theta^2[\hat{\theta}_n(\vec{Y})]$; (3) CDF $G_n(t) = P(\hat{\theta}_n(\vec{Y}) < t), \forall t$

Approach #1 Monte-Carlo Simulations

Generate k vectors $\vec{Y}^{(i)}, i = 1, 2, ..., k$ (total kn random variables) (1) Bias $\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n(\vec{Y}^{(j)}) - \theta$; (2) Variance $\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n^2(\vec{Y}^{(j)}) - \left(\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n(\vec{Y}^{(j)})\right)^2$; (3) CDF $\hat{G}_n(t) = \frac{1}{k}\sum_{j=1}^k \mathbf{1}_{\hat{\theta}_n(\vec{Y}^{(j)})<t}, \forall t$

Approach #2 Bootstrap

(When data is not enough) Suppose we only have one data $\vec{Y} = (Y_1, ..., Y_n)$

Reuse $Y_1, ...Y_n$ to obtain resamples $\vec{Y}^* = (Y_1^*, ..., Y_n^*)$. Do this k times $\Rightarrow$ k resamples $\vec{Y^*}^{(1)}, ..., \vec{Y^*}^{(k)}$

(1) Bias $\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n(\vec{Y^*}^{(j)}) - \theta$; (2) Variance $\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n^2(\vec{Y^*}^{(j)}) - \left(\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n(\vec{Y^*}^{(j)})\right)^2$; (3) CDF $\hat{G}_n(t) = \frac{1}{k}\sum_{j=1}^k \mathbf{1}_{\hat{\theta}_n(\vec{Y^*}^{(j)})<t}, \forall t$

Example: $\theta = med\{P\}$, P is an unknown distribution over $\{0, 1, ..., 9\}$. $\vec{Y} = (4, 8, 9, 6, 2)$.

9.1 Residual Bootstrap

The bootstrap principle is quite general and may also be used in problems where the data Y_i, $1 \le i \le n$, **are not i.i.d**.

Example: Linear

Observation $Y_i = a + b\frac{i}{n} + Z_i$, where $Z_i \sim N(0, \sigma^2)$ (i.i.d.) for $i = 1, 2, ..., n$

Parameter $\theta = (a, b)$. Linear Least Square Estimator:

$$(\hat{a}_n, \hat{b}_n) = \underset{(a,b)}{\operatorname{argmin}} \sum_{i=1}^n (Y_i - a - b\frac{i}{n})^2$$

Given $\vec{Y}$, the residual (not i.i.d.)

$$E_i = Y_i = \hat{a}_n - \hat{b}_n\frac{i}{n} \approx Z_i$$

Generate k resamples of $\vec{E} = (E_1, E_2, ..., E_n)$

$\Rightarrow$ obtain $\vec{E^*}^{(1)}, \vec{E^*}^{(2)}, ..., \vec{E^*}^{(k)}$ by resampling

$\Rightarrow$ Compute pseudo-data $Y_i^{*(j)} = \hat{a}_n + \hat{b}_n \frac{i}{n} + E_i^{*(j)}$

$\Rightarrow$ Compute LS estimator

$$\hat{\theta}_n^{(j)} = (\hat{a}_n^{(j)}, \hat{b}_n^{(j)}) = \underset{(a,b)}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i^{*(j)} - a - b\frac{i}{n})^2$$

$\Rightarrow$ Evaluate bias

$$\widehat{Bias} = \frac{1}{k} \sum_{j=1}^{k} \hat{\theta}_n^{(j)} - \theta$$

Example: Nonlinear Markov Process

Observation $Y_i = F_\theta(Y_{i-1}) + Z_i$, where $Z_i \sim N(0, \sigma^2)$ (i.i.d.) for $i = 1, 2, ..., n$

Parameter $\theta = (a, b)$. Linear Least Square Estimator:

$$\hat{\theta}_n(\vec{Y}) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i - F_\theta(Y_{i-1}))^2$$

Given $\vec{Y}$, the residual (not i.i.d.)

$$E_i = Y_i = \hat{a}_n - F_{\hat{\theta}_n}(Y_{i-1}) \approx Z_i$$

Generate k resamples of $\vec{E} = (E_1, E_2, ..., E_n)$

$\Rightarrow$ obtain $\vec{E^*}^{(1)}, \vec{E^*}^{(2)}, ..., \vec{E^*}^{(k)}$ by resampling

$\Rightarrow$ Fix $Y_0^{*(j)} = Y_0$, compute pseudo-data $Y_i^{*(j)} = F_{\hat{\theta}_n}(Y_{i-1}^{*}{}^{(j)}) + E_i^{*(j)}$

$\Rightarrow$ Compute LS estimator

$$\hat{\theta}_n^{(j)} = \underset{(a,b)}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i^{*(j)} - F_{\hat{\theta}_n}(Y_{i-1}^{*}{}^{(j)}))^2$$

$\Rightarrow$ Evaluate bias

$$\widehat{Bias} = \frac{1}{k} \sum_{j=1}^{k} \hat{\theta}_n^{(j)} - \theta$$

Chapter 10 Particle Filtering

Kalman filtering is used in tracking problems (dynamic models). Particle Filtering is an extension of Kalman filtering.

10.1 Kalman Filtering (Linear Dynamic System)

1. Unknown state sequence $X_t \in \mathbb{R}^m, t = 0, 1, 2, ...$
2. Observations $Y_t \in \mathbb{R}^k, t = 0, 1, 2, ...$
3. $X_{t+1} = F_t X_t + U_t$, $F_t \in \mathbb{R}^{m \times m}, U_t \sim P_{U_t}$
4. $Y_t = H_t X_t + V_t$, $H_t \in \mathbb{R}^{k \times m}, V_t \sim P_{V_t}$

We want to solve two problems

1. **Estimation Problem:** Evaluate Linear MMSE (LMMSE) of X_t given $Y_{0:t}$.

$$\hat{X}_{t|t} = WY_{0:t} + b$$

2. **Prediction Problem:** Predict Linear MMSE (LMMSE) of $X_{t+1|t}$ given $Y_{0:t}$. (Really hard)

We can solve closed-form solutions.

10.2 Particle Filtering (Nonlinear Dynamic System)

Particle filtering is a nonlinear form of Kalman filtering, which doesn't have closed-from solutions.

We consider a Nonlinear Dynamic System

$$X_{t+1} \sim q(\cdot|X_t)$$

$$Y_t \sim r(\cdot|X_t)$$

$$t = 0, 1, 2, ...$$

where $q(X_{t+1}|X_t)$ is the transition probability distribution, and $r(Y_t|X_t)$ is the conditional probability distribution for the observations. Hence, X_t is a Markov process and Y_t follows a Hidden Markov Model (HMM).

We also consider these two probelms.

1. **Estimation Problem:** Evaluate X_t given $Y_{0:t}$.
2. **Prediction Problem:** Predict $X_{t+1|t}$ given $Y_{0:t}$.

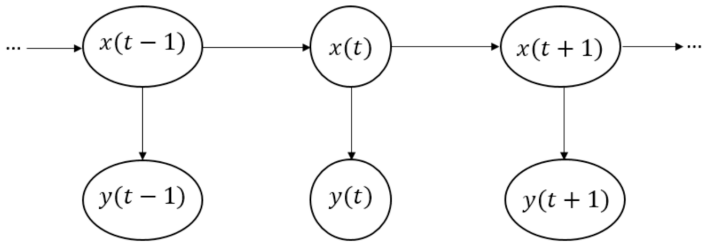


Figure 10.1: Hidden Markov Model

10.2.1 Bayesian Recursive Filtering

In this section we use Bayesian approach and use MMSE estimation $l(\hat{x}_t, x_t) = \|x_t - \hat{x}_t\|^2$

Estimation and prediction in conditional forms are:

$$\hat{X}_{t|t} = \mathbb{E}[X_t|Y_{0:t}] = \int_{\mathbb{R}^m} x_t P(X_t|Y_{0:t}) dx_t$$

$$\hat{X}_{t+1|t} = \mathbb{E}[X_{t+1}|Y_{0:t}] = \int_{\mathbb{R}^m} x_{t+1} P(X_{t+1}|Y_{0:t}) dx_{t+1}$$

Apparently the posterior p.d.f cannot be evaluated due to the curse of dimensionality as t increases. However, they can in principle be evaluated *recursively* using the following two-step procedure.

Step 1: Prediction. $P(X_{t+1}|Y_{0:t})$ can be expressed in term of $P(X_t|Y_{0:t})$:

$$\begin{aligned} P(X_{t+1}|Y_{0:t}) &= \int_{\mathbb{R}^m} P(X_{t+1}, X_t|Y_{0:t}) dx_t \\ &= \int_{\mathbb{R}^m} P(X_{t+1}|X_t, Y_{0:t}) P(X_t|Y_{0:t}) dx_t \\ &= \int_{\mathbb{R}^m} q(X_{t+1}|X_t) P(X_t|Y_{0:t}) dx_t \end{aligned}$$

Step 2: Update. We can also express $P(X_t|Y_{0:t})$ in terms of $P(X_t|Y_{0:t-1})$

$$\begin{aligned} P(X_t|Y_{0:t}) &= P(X_t|Y_t, Y_{0:t-1}) \\ &= \frac{P(Y_t|X_t, Y_{0:t-1}) P(X_t|Y_{0:t-1})}{P(Y_t|Y_{0:t-1})} \\ &= \frac{r(Y_t|X_t) P(X_t|Y_{0:t-1})}{\int_{\mathbb{R}^m} r(Y_t|X_t) P(X_t|Y_{0:t-1}) dx_t} \end{aligned}$$

10.2.2 Particle Filter (bootstrap filter)

Suppose we have n i.i.d. samples of X_t drawn from $p(x_t|Y_{0:t})$: $X_t(1), X_t(2), ..., X_t(n)$.

$$X_t(i) \sim p(\cdot|Y_{0:t}), 1 \leq i \leq n \qquad \text{(Sample 1)}$$

We can use above recursive filtering method to generate estimation of X_{t+1}.

Step 1: Prediction. Using the transition probability $q(\cdot|X_t(i)), 1 \leq i \leq n$ to generate n independent random variables

$$X^*_{t+1}(i) \sim q(\cdot|X_t(i)),\ 1 \leq i \leq n \qquad \text{(Sample 2)}$$

Step 2: Update. Upon **receiving a new measurement** y_{t+1}, evaluate the *importance weights* (nonnegative and summing to 1)

$$w_i = \frac{r(y_{t+1}|X^*_{t+1}(i))}{\sum_{j=1}^n r(y_{t+1}|X^*_{t+1}(j))},\ 1 \le i \le n$$

Then we resample n times from the set $\{X^*_{t+1}(i)\}_{i=1}^n$ with respective probabilities $\{w_i\}_{i=1}^n$, obtaining i.i.d samples $\{X_{t+1}(j)\}_{j=1}^n$ with probabilities

$$Pr[X_{t+1}(j) = X^*_{t+1}(i)] = w_i,\ 1 \le i, j \le n \qquad \text{(Sample 3)}$$

By the **weighted bootstrap theorem**, as $n \to \infty$, the distribution of the resampled $\{X_{t+1}(j)\}_{j=1}^n$ converges to the desired posterior.

Potential issues: 1. n is not large enough. 2. Sample impoverishment

Chapter 11 EM Algorithm

The ML estimator: $\hat{\theta}_{ML} = \text{argmax}_{\theta \in S} \ln p_\theta(y)$. Numerical evaluation of maximum-likelihood (ML) estimates is often difficult. The likelihood function may have multiple extreme and the parameter θ may be multidimensional, all of which are problematic for any numerical algorithm.

Maximum-Likelihood (ML) Estimation

Given a vector $\vec{y}$, find the θ that maximizes $p_\theta(\vec{y}) = \prod_{i=1}^{n} P(y_i|\theta)$

$$\hat{\theta}_{ML} = \underset{\theta \in S}{\text{argmax}} \ln p_\theta(\vec{y}) = \underset{\theta \in S}{\text{argmax}} \sum_{i=1}^{n} \ln P(y_i|\theta)$$

Solving the closed-form solution is quite hard sometimes, so we may use EM algorithm.

11.1 General Structure of the EM Algorithm

What we want to estimate:

$\theta \in S$ is an unknown parameter that we want to estimate.

What we know:

To help us solve the solution, we construct an unobservable vector $\vec{z}$ corresponding to $\vec{y}$.

1. There is a complete data space Z and an incomplete data space Y.
2. The reality is $z \in Z$, which has p.d.f $P(z|\theta)$. ($\ln P(z|\theta)$'s derivative should be constructed to be easy.)
3. **Instead of observing the z directly, we can observe $y = h(z) \in Y$ which has p.d.f $P(y|\theta)$.**
4. $h(z) = y$ is a many-to-one mapping.

$$P(y, z|\theta) = P(z|\theta), \quad \forall z \in h^{-1}(y)$$

5. We can infer that the relationship between $P(z|\theta)$ and $P(y|\theta)$ is

$$P(z|\theta) = P(z|y, \theta)P(y|\theta), \quad \forall z \in h^{-1}(y)$$

$$P(y|\theta) = \sum_{z \in h^{-1}(y)} P(z|\theta), \quad \forall y$$

6. For any function f, $\mathbb{E}_z[f(z)|y]$ depends on the p.d.f.

$$\mathbb{E}_{z|\theta}[f(z)|y] = \sum_{z \in h^{-1}(y)} P(z|y, \theta) f(z)$$

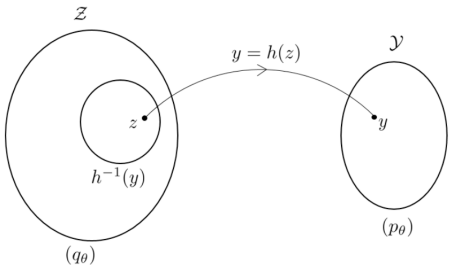


Figure 11.1: Complete and incomplete data spaces Z and Y.

EM Algorithm

Instead of computing the $P(\vec{y}|\theta)$ directly, we use the relationship $h(z) = y$ and $P(\vec{z}|\theta)$ to estimate θ.

Suppose we have a prior belief of the relationship between y and z: $P(z|y, \theta^{(k)})$. Given $\vec{y}$, since maximizing $\ln P(\vec{y}|\theta)$ is hard, we maximize the expected value of $\ln P(\vec{z}|\theta)|\vec{y}$ under the prior belief (i.e., finding the θ that can properly represent the relationship between $\vec{y}$ and $\vec{z}$), that is

$$\begin{aligned}\theta &= \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{z|y,\theta^{(k)}}[\ln P(\vec{z}|\theta)|\vec{y}] \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{z_i \in h^{-1}(y_i)} P(z_i|y_i, \theta^{(k)}) \ln P(z_i|\theta)\end{aligned}$$

EM algorithm alternates between Expectation (E) and Maximization (M) steps:

1. Initialize $\hat{\theta}^{(0)}$
2. For $k = 0, 1, 2, ...$

   Expectation (E)-Step: Compute

$$\begin{aligned}Q(\theta|\hat{\theta}^{(k)}) &= \mathbb{E}_{z|y,\hat{\theta}^{(k)}}[\ln P(\vec{z}|\theta)|\vec{y}] \\ &= \sum_{i=1}^{n} \sum_{z_i \in h^{-1}(y_i)} P(z_i|y_i, \theta^{(k)}) \ln P(z_i|\theta)\end{aligned}$$

   Maximization (M)-Step

$$\hat{\theta}^{(k+1)} = \underset{\theta \in S}{\operatorname{argmax}}\, Q(\theta|\hat{\theta}^{(k)})$$

Definition 11.1

θ^* is a stable point of the EM algorithm if $\exists$ subsequence that converges to θ^*.

e.g. $1, 3, \frac{1}{2}, 3, \frac{1}{3}, 3, ... \frac{1}{n}, 3, ...$

11.2 Example 1: Variance Estimation

Observation $Y = S + N$, $S \sim \mathcal{N}(0, \theta)$ is independent of $N \sim \mathcal{N}(0, \theta) \Rightarrow Y \sim \mathcal{N}(0, \theta + 1)$. $p_\theta(y) = \frac{1}{\sqrt{2\pi(\theta+1)}} e^{-\frac{y^2}{2(\theta+1)}}$. We want to estimate θ.

11.2.1 Maximum-Likelihood (ML) Estimation

$$\ln p_\theta(y) = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\theta+1) - \frac{y^2}{2(\theta+1)}$$

take derivation of θ to be equal to 0

$$-\frac{1}{2(\theta+1)} + \frac{y^2}{2(\theta+1)^2} = 0$$

We can get

$$\hat{\theta} = y^2 - 1$$

Then,

$$\hat{\theta}_{ML} = \begin{cases} 0, & y^2 \leq 1 \\ y^2 - 1, & y^2 > 1 \end{cases}$$

11.2.2 EM Algorithm

Let $Z = (S, N)$, $y = h(z) = s + n$.

$$q_\theta(z) = q_\theta(s, n) = \frac{1}{\sqrt{2\pi\theta}} e^{-\frac{s^2}{2\theta}} \frac{1}{\sqrt{2\pi}} e^{-\frac{n^2}{2}}$$

Then

$$\ln q_\theta(z) = \ln \frac{1}{\sqrt{2\pi}} e^{-\frac{n^2}{2}} - \frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\theta) - \frac{s^2}{2\theta}$$

E-Step: Compute

$$\begin{aligned} Q(\theta|\hat{\theta}^{(k)}) &= \mathbb{E}_{z|\hat{\theta}^{(k)}}[\ln q_\theta(z)|Y = y] \\ &= \sum_{z \in h^{-1}(y)} q_{\hat{\theta}^{(k)}}(z) \ln q_\theta(z) \\ &= \ln \frac{1}{\sqrt{2\pi}} e^{-\frac{n^2}{2}} - \frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\theta) - \frac{\mathbb{E}_{z|\hat{\theta}^{(k)}}(s^2)}{2\theta} \end{aligned}$$

M-Step

$$\hat{\theta}^{(k+1)} = \underset{\theta \in S}{\operatorname{argmax}}\, Q(\theta|\hat{\theta}^{(k)})$$

$$0 = -\frac{1}{2\hat{\theta}^{(k+1)}} + \frac{\mathbb{E}_{z|\hat{\theta}^{(k)}}(s^2)}{2(\hat{\theta}^{(k+1)})^2}$$

$$\hat{\theta}^{(k+1)} = \mathbb{E}_{z|\hat{\theta}^{(k)}}(s^2) = \frac{\hat{\theta}^{(k)}}{\hat{\theta}^{(k)}+1}\left(\frac{\hat{\theta}^{(k)}}{\hat{\theta}^{(k)}+1}y^2+1\right)$$

Then we can solve the stable point

$$\hat{\theta}^* = \frac{\hat{\theta}^*}{\hat{\theta}^*+1}\left(\frac{\hat{\theta}^*}{\hat{\theta}^*+1}y^2+1\right)$$

$$\Rightarrow \hat{\theta}^* = 0, \hat{\theta}^* = y^2 - 1$$

According to the relation between $\hat{\theta}^{(k)}$ and $\hat{\theta}^{(k+1)}$, we can infer

$$\hat{\theta}^* = \begin{cases} 0, & y^2 \le 1 \\ y^2 - 1, & y^2 > 1 \end{cases}$$

11.3 Example 2: Estimation of Gaussian Mixtures

Assume the data $\vec{Y} = \{Y_i, 1 \le i \le n\} \in \mathbb{R}^n$, are drawn iid from a pdf $p_\theta(y)$ which is the mixture of m univariate Gaussians with respective probabilities $\pi(j)$, means μ_j, and variances σ_j^2, for $1 \le j \le m$:

$$p_\theta(y|j) = \phi\left(y; \mu_j, \sigma_j^2\right)$$

$$p_\theta(y) = \sum_{j=1}^{m} \pi(j)\phi\left(y; \mu_j, \sigma_j^2\right), \quad y \in \mathbb{R}$$

where

$$\phi\left(y; \mu, \sigma^2\right) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y-\mu)^2}{2\sigma^2}\right\}$$

denotes the Gaussian pdf with mean μ and variance σ^2.

11.3.1 Unknown Means: ML estimation is hard

We initially assume that $\{\pi(j)\}$ and $\left\{\sigma_j^2\right\}$ are given and that we only need to estimate the means $\{\mu_j\}$. Thus, $\theta = \mu \in \mathbb{R}^m$.

Unfortunately the ML estimator cannot be derived in closed form. Indeed, the loglikelihood function for θ is

$$\ln \prod_{i=1}^{n} p_\theta(y_i) = \sum_{i=1}^{n} \ln p_\theta\left(y_i\right) = \sum_{i=1}^{n} \ln \sum_{j=1}^{m} \pi(j)\phi\left(y_i; \mu_j, \sigma_j^2\right)$$

and maximizing it is a m-dimensional, nonconcave maximization problem.

Taking the derivative of $\mu_j, j = 1, ..., m$,

$$\begin{aligned} 0 &= \frac{1}{\sigma_j^2} \sum_{i=1}^{n} (y_i - \mu_j) \frac{\pi(j)\phi(y_i; \mu_j, \sigma_j^2)}{\sum_{j=1}^{m} \pi(j)\phi(y_i; \mu_j, \sigma_j^2)} \\ &= \frac{1}{\sigma_j^2} \sum_{i=1}^{n} (y_i - \mu_j) \pi_\theta(j|y_i) \end{aligned}$$

where $\pi_\theta(j|y_i) \triangleq \frac{\pi(j)\phi(y_i;\mu_j,\sigma_j^2)}{\sum_{j=1}^{m} \pi(j)\phi(y_i;\mu_j,\sigma_j^2)}$. The system may have multiple solutions corresponding to local maxima or even local minima or saddle points of the likelihood function.

11.3.2 Unknown Means: EM Algorithm

There is a complete data $Z_i = (J_i, Y_i), = 1, ..., n$, where J_i is the random label that was drawn to produce Y_i. $z = \{j_i, y_i\}_{i=1}^n$ is the sample.

$$q_\theta(z) = \prod_{i=1}^{n} (\pi(j_i) p_\theta(y_i|j_i))$$

$$\ln q_\theta(z) = \sum_{i=1}^{n} [\ln \pi(j_i) + \ln p_\theta(y_i|j_i)]$$

Initialize $\hat{\theta}^{(0)}$

Iteration:

$$\begin{aligned} Q(\theta|\hat{\theta}^{(k)}) &= \sum_{i=1}^{n} \mathbb{E}_{\hat{\theta}^{(k)}} [\ln \pi(j_i) + \ln p_\theta(y_i|j_i)|Y_i = y_i] \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \pi_{\hat{\theta}^{(k)}}(j|y_i) [\ln \pi(j) + \ln p_\theta(y_i|j)] \\ &= cst - \sum_{i=1}^{n} \sum_{j=1}^{m} \pi_{\hat{\theta}^{(k)}}(j|y_i) \frac{(y_i - \mu_j)^2}{2\sigma_j^2} \\ &= cst - \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{\pi(j)\phi(y_i; \hat{\mu}_j^{(k)}, \sigma_j^2)}{\sum_{j=1}^{m} \pi(j)\phi(y_i; \hat{\mu}_j^{(k)}, \sigma_j^2)} \frac{(y_i - \mu_j)^2}{2\sigma_j^2} \end{aligned}$$

where $\ln p_\theta(y_i|j) = -\frac{1}{2} \ln(2\pi\sigma_j^2) - \frac{(y_i - \mu_j)^2}{2\sigma_j^2}$.

Take derivative of μ_j,

$$\begin{aligned} 0 = \frac{\partial Q(\theta|\hat{\theta}^{(k)})}{\partial \mu_j} &= \sum_{i=1}^{n} \pi_{\hat{\theta}^{(k)}}(j|y_i) \frac{(y_i - \mu_j)}{\sigma_j^2} \\ \hat{\mu}_j^{(k+1)} &= \frac{\sum_{i=1}^{n} \pi_{\hat{\theta}^{(k)}}(j|y_i) y_i}{\sum_{i=1}^{n} \pi_{\hat{\theta}^{(k)}}(j|y_i)} \end{aligned}$$

Recall the $\hat{\theta}_{ML}$

$$\hat{\theta}_{ML,j} = \frac{\sum_{i=1}^{n} \pi_{\hat{\theta}_{ML}}(j|y_i) y_i}{\sum_{i=1}^{n} \pi_{\hat{\theta}_{ML}}(j|y_i)}$$

$\hat{\theta}_{ML}$ is the stable point. (if exist)

11.3.3 Unknown Mixture Probabilities, Means and Variances

ML Estimation:

If $\theta \triangleq \left\{\pi(j), \mu_j, \sigma_j^2, 1 \le j \le m\right\}$ is unknown, the ML estimator $\hat{\theta}_{\text{ML}}$ satisfies the following nonlinear system of equations:

$$\hat{\mu}_{\text{ML},j} = \frac{\sum_{i=1}^{n} y_i \pi_{\hat{\theta}^{(k)}}(j \mid y_i)}{\sum_{i=1}^{n} \pi_{\hat{\theta}^{(k)}}(j \mid y_i)}$$

$$\hat{\sigma}^2_{\text{ML},j} = \frac{\sum_{i=1}^{n} (y_i - \hat{\mu}_{\text{ML},j})^2 \pi_{\hat{\theta}^{(k)}}(j \mid y_i)}{\sum_{i=1}^{n} \pi_{\hat{\theta}(k)}(j \mid y_i)}$$

$$\hat{\pi}_{\text{ML}}(j) = \frac{1}{n} \sum_{i=1}^{n} \pi_{\hat{\theta}^{(k)}}(j \mid y_i) \quad 1 \le j \le m$$

where

$$\pi_\theta(j \mid y_i) = \frac{\pi(j)\phi\left(y_i; \mu_j, \sigma_j^2\right)}{\sum_{j=1}^{m} \pi(j)\phi\left(y_i; \mu_j, \sigma_j^2\right)}, \quad 1 \le j \le m$$

E-step:

$$Q(\theta|\hat{\theta}^{(k)}) = cst - \sum_{i=1}^{n}\sum_{j=1}^{m} \pi_{\hat{\theta}^{(k)}}(j|y_i) \frac{(y_i - \mu_j)^2}{2\sigma_j^2}$$

$$= cst - \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{\hat{\pi}^{(k)}(j)\phi(y_i; \hat{\mu}_j^{(k)}, \hat{\sigma_j^2}^{(k)})}{\sum_{j=1}^{m} \hat{\pi}^{(k)}(j)\phi(y_i; \hat{\mu}_j^{(k)}, \hat{\sigma_j^2}^{(k)})} \frac{(y_i - \mu_j)^2}{2\sigma_j^2}$$

M-Step:

$$\hat{\mu}_j^{(k+1)} = \frac{\sum_{i=1}^{n} y_i \pi_{\hat{\theta}^{(k)}}(j \mid y_i)}{\sum_{i=1}^{n} \pi_{\hat{\theta}(k)}(j \mid y_i)}$$

$$\left(\hat{\sigma}_j^2\right)^{(k+1)} = \frac{\sum_{i=1}^{n} \left(y_i - \hat{\mu}_j^{(k+1)}\right)^2 \pi_{\hat{\theta}^{(k)}}(j \mid y_i)}{\sum_{i=1}^{n} \pi_{\hat{\theta}^{(k)}}(j \mid y_i)}$$

$$\hat{\pi}^{(k+1)}(j) = \frac{1}{n} \sum_{i=1}^{n} \pi_{\hat{\theta}^{(k)}}(j \mid y_i), \quad 1 \le j \le m.$$

11.4 Convergence of EM Algorithm

Theorem 11.1

The likelihood sequence $p_{\hat{\theta}^{(k)}}(y)$, $k = 0, 1, 2, \ldots$ is nondecreasing. ♡

Proof 11.1

Assume for notational simplicity that the random variables Y and Z are discrete. Hence, their joint

distribution is given by

$$P_\theta(y,z) = q_\theta(z)p_\theta(y|z) = q_\theta(z)\mathbf{1}_{\{y=h(z)\}}$$
$$= p_\theta(y)q_\theta(z|y)$$

Given y, the following identity holds for all $z \in h^{-1}(y)$:

$$p_\theta(y) = \frac{q_\theta(z)}{q_\theta(z|y)}$$

Taking the logarithm,

$$\ln p_\theta(y) = \ln q_\theta(z) - \ln q_\theta(z|y),\ \forall z \in h^{-1}(y)$$

Taking the conditional expectation with respect to $q_{\hat\theta}(z|y)$,

$$\ln p_\theta(y) = \sum_{z\in h^{-1}(y)} q_{\hat\theta}(z|y)\ln q_\theta(z) - \sum_{z\in h^{-1}(y)} q_{\hat\theta}(z|y)\ln q_\theta(z|y) \tag{1}$$

Expectation (E)-Step: *Compute*

$$Q(\theta|\hat\theta^{(k)}) = \sum_{z\in h^{-1}(y)} q_{\hat\theta^{(k)}}(z|y)\ln q_\theta(z)$$

Maximization (M)-Step

$$\hat\theta^{(k+1)} = \operatorname*{argmax}_{\theta\in S} Q(\theta|\hat\theta^{(k)})$$

According to (1),

$$\ln p_\theta(y) = Q(\theta|\hat\theta^{(k)}) - H(q_{\hat\theta^{(k)}}, q_\theta)$$

$$\ln p_{\hat\theta^{(k+1)}}(y) - \ln p_{\hat\theta^{(k)}}(y) = (Q(\hat\theta^{(k+1)}|\hat\theta^{(k)}) - Q(\hat\theta^{(k)}|\hat\theta^{(k)})) - (H(q_{\hat\theta^{(k)}}, q_{\hat\theta^{(k+1)}}) - H(q_{\hat\theta^{(k)}}, q_{\hat\theta^{(k)}}))$$

Since $\hat\theta^{(k+1)} = \operatorname{argmax}_{\theta\in S} Q(\theta|\hat\theta^{(k)})$, $Q(\hat\theta^{(k+1)}|\hat\theta^{(k)}) - Q(\hat\theta^{(k)}|\hat\theta^{(k)}) \geq 0$.

$$H(q_{\hat\theta^{(k)}}, q_{\hat\theta^{(k+1)}}) - H(q_{\hat\theta^{(k)}}, q_{\hat\theta^{(k)}}) = D(q_{\hat\theta^{(k)}} \| q_{\hat\theta^{(k+1)}}) \geq 0$$

Hence, we can conclude $\ln p_{\hat\theta^{(k+1)}}(y) - \ln p_{\hat\theta^{(k)}}(y) \geq 0$. Then $p_{\hat\theta^{(k)}}$ should be nondecreasing in k.

Corollary 11.1

Assume that S is a closed, bounded subset of Euclidean space, the functions $Q(\theta|\theta')$ and $H(\theta|\theta')$ are continuously differentiable, and the loglikelihood function $\ln p_{\hat\theta^{(k)}}$ is differentiable and bounded. Then the sequence $\ln p_{\hat\theta^{(k)}}$ converges, and any limit point $\theta^* \in$ interior(S) of the EM sequence is a solution of the likelihood equation $\nabla \ln p_\theta = 0$.

11.5 EM As an Alternating Maximization Algorithm

Define an *auxiliary cost function* $L(q,\theta)$.

Incomplete data Y; Complete data Z. Still $h(z) : Z \to Y$.

$\mathcal{Q}_y = \{q : q(z) = 0, \forall z \in h^{-1}(y)\}$

EM updates

1. E-Step:

Chapter 12 Hidden Markov model (HMM)

A Markov chain $X_{tt\geq 1}$ is observed as $\{Y_t\}_{t\geq 1}$. The state sets are finite sets S_x, S_y. Suppose the initial state distribution is π. The *transition probability matrix* of the MC is

$$A(i,j) = P(X_{t+1} = j | X_t = i),\ i,j \in S_x$$

and the *emission probability matrix* is

$$B(i,j) = P(Y_t = j | X_t = i),\ i \in S_x, j \in S_y$$

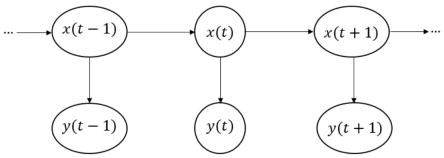


Figure 12.1: Hidden Markov Model (HMM)

Relative problems include

Problem 1: Estimate X_t given $Y_{1:t}$ (Using MAP or MMSE criterion: particle filtering)

Problem 2: Estimate X_{t+1} given $Y_{1:t}$ (Using MAP or MMSE prediction: particle filtering)

Problem 3: Estimate $X_{1:t}$ given $Y_{1:t}$ (MAP, MMSE)

Problem 4: Estimate the HMM parameters $\theta = (\pi, A, B)$ given $Y_{1:t}$ (learning)

12.1 Viterbi Algorithm: (MAP) estimate $X_{1:t}$ given $Y_{1:t}$

12.1.1 MAP estimation problem

The MAP estimation problem arises in a variety of applications, and Viterbi derived a remarkable algorithm for solving it exactly. The probability of state $\vec{x} \in S_x^n$ is given by

$$P(\vec{x}) = \pi(x_1) \prod_{t=1}^{n-1} A(x_t, x_{t+1})$$

and the conditional probability of the observed sequence $\vec{y}$ given the state sequence $\vec{x}$ is

$$P(\vec{y}|\vec{x}) = \prod_{t=1}^{n} B(x_t, y_t)$$

Hence, the joint probability of $\vec{x}$ and $\vec{y}$ is

$$P(\vec{x}, \vec{y}) = P(\vec{x})P(\vec{y}|\vec{x}) = \pi(x_1) \prod_{t=1}^{n-1} A(x_t, x_{t+1}) \prod_{t=1}^{n} B(x_t, y_t)$$

Then the MAP estimation problem is

$$\begin{aligned}\vec{x}^* &= \underset{\vec{x}}{\operatorname{argmax}}\, P(\vec{x}|\vec{y}) = \underset{\vec{x}}{\operatorname{argmax}}\, \frac{P(\vec{x},\vec{y})}{P(\vec{y})} = \underset{\vec{x}}{\operatorname{argmax}}\, P(\vec{x},\vec{y}) \\ &= \underset{\vec{x}}{\operatorname{argmax}}\ \pi(x_1)\prod_{t=1}^{n-1} A(x_t,x_{t+1})\prod_{t=1}^{n} B(x_t,y_t) \\ &= \underset{\vec{x}}{\operatorname{argmax}}\ \ln\pi(x_1) + \sum_{t=1}^{n-1}\ln A(x_t,x_{t+1}) + \sum_{t=1}^{n}\ln B(x_t,y_t)\end{aligned}$$

12.1.2 Viterbi Algorithm

Let $f(x_1) = \ln\pi(x_1) + \ln B(x_1,y_1)$, $g_t(x_t,x_{t+1}) = \ln A(x_t,x_{t+1}) + \ln B(x_{t+1},y_{t+1})$. Then the estimation problem is written in the form

$$\vec{x}^* = \underset{\vec{x}}{\operatorname{argmax}}\ \left[\varepsilon(\vec{x}) = f(x_1) + \sum_{u=1}^{n-1} g_u(x_u,x_{u+1})\right]$$

Let $V(1,x) = f(x)$ and

$$V(t,x_t=x) \triangleq \max_{x_1,x_2,\ldots,x_{t-1}}\left[\varepsilon([x_1,\ldots,x_t]) = f(x_1) + \sum_{u=1}^{t-1} g_u(x_u,x_{u+1})\right]$$

$$V(t,x_t=x) = \max_{x'}\left[V(t-1,x_{t-1}=x') + g_{t-1}(x',x)\right],\ t\geq 2$$

Then, when $t=n$ we have

$$\max_{\vec{x}}\varepsilon(\vec{x}) = \max_{x} V(n,x_n=x)$$

The complexity of the algorithm is $O(n|S_x|)$ storage and $O(n|S_x|^2)$ computation.

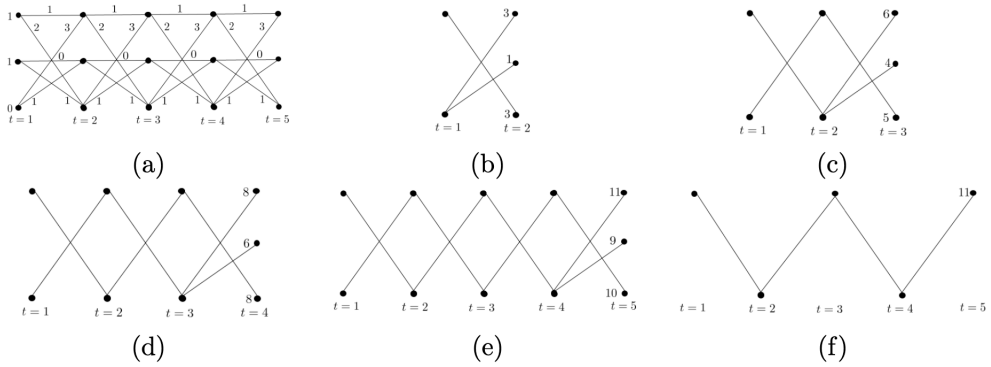


Figure 12.2: (a) Trellis diagram; (b)—(e) evolution of the Viterbi algorithm, showing surviving paths and values $V(t,x)$ at times $t=2,3,4,5$; (f) optimal path $\vec{x}^* = (0,2,0,2,0)$ and its value $\varepsilon(\vec{x}^*) = 11$.

12.2 Bayesian Estimation of a Sequence: Need (MMSE) estimate $X_{1:t}$ given $Y_{1:t}$

Consider Bayesian estimation under an additive squared-error loss function:

$$L(\vec{x}, \hat{\vec{x}}) = \sum_{t=1}^{n} L(x_t, \hat{x}_t) = \sum_{t=1}^{n} (x_t - \hat{x}_t)^2$$

The Bayesian estimator $\hat{\vec{x}}$ achieves

$$\min_{\hat{\vec{x}}} \sum_{\vec{x} \in X^n} L(\vec{x}, \hat{\vec{x}}) P(\vec{x}|\vec{y}) = \sum_{t=1}^{n} \min_{\hat{\vec{x}}_t} \sum_{x_t \in X} L(x_t, \hat{x}_t) P(x_t|\vec{y})$$

In particular, under squared-error loss, we obtain the conditional mean estimator

$$\hat{\vec{x}}_t = \sum_{x_t \in X} x_t P\left(X_t = x | Y = \vec{y}\right), \quad 1 \le t \le n$$

12.3 Forward-Backward Algorithm: (MMSE) estimate $X_{1:t+1}$ given $Y_{1:t}$

1. Evaluate $P\left(X_t = x | Y = \vec{y}\right)$ for $t = 1, 2, ..., n$ and $x \in \mathcal{X}$. (Used to (MMSE) estimate $X_{1:t}$ given $Y_{1:t}$)
2. Evaluate $P\left(X_t = x, X_{t+1} = x' | Y = \vec{y}\right)$ for $t = 1, 2, ..., n$ and $x, x' \in \mathcal{X}$ (Used to learn parameters $\theta = (\pi, A, B)$)

Define the shorthands

$$\gamma_t(x) \triangleq \mathrm{P}\left\{X_t = x \mid \vec{Y} = \vec{y}\right\}$$

$$\xi_t\left(x, x'\right) \triangleq \mathrm{P}\left\{X_t = x, X_{t+1} = x' \mid \vec{Y} = \vec{y}\right\}, \quad x, x' \in \mathcal{X}$$

Hence γ_t is the first marginal of ξ_t. The forward-backward algorithm allows efficient computation of these probabilities.

12.3.1 $\gamma_t(x) \triangleq \mathrm{P}\left\{X_t = x \mid \vec{Y} = \vec{y}\right\}$

We begin with

$$\gamma_t(x) = \mathrm{P}\left\{X_t = x \mid \vec{Y} = \vec{y}\right\} = \frac{\mathrm{P}\left\{X_t = x, \vec{Y} = \vec{y}\right\}}{\sum_{x \in \mathcal{X}} \mathrm{P}\left\{X_t = x, \vec{Y} = \vec{y}\right\}}, \quad 1 \le t \le n$$

Write the numerator as a product of two conditional distributions,

$$\mathrm{P}\left\{\vec{Y} = \vec{y}, X_t = x\right\} \overset{(a)}{=} \underbrace{\mathrm{P}\left\{Y_{1:t} = y_{1:t}, X_t = x\right\}}_{\mu_t(x)} \underbrace{\mathrm{P}\left\{Y_{t+1:n} = y_{t+1:n} \mid X_t = x\right\}}_{\nu_t(x)}$$

$$= \mu_t(x)\nu_t(x), \quad 1 \le t < n$$

where (a) follows from the Markov chain $Y_{1:t} \to X_t \to Y_{t+1:n}$. For $t = n$, we let $\nu_n(x) \equiv 1$. Combining above two equations we have

$$\gamma_t(x) = \frac{\mu_t(x)\nu_t(x)}{\sum_{x\in\mathcal{X}} \mu_t(x)\nu_t(x)}.$$

(1) The first factor in the product of $\mathrm{P}\left\{\vec{Y} = \vec{y}, X_t = x\right\}$ is

$$\mu_t(x) = \mathrm{P}\left\{Y_{1:t} = y_{1:t}, X_t = x\right\}, \quad x \in \mathcal{X}, 1 \le t \le n,$$

for which we derive a **forward recursion**. The recursion is initialized with

$$\mu_1(x) = \mathrm{P}\left\{Y_1 = y_1, X_1 = x\right\} = \pi(x)B\left(x, y_1\right).$$

For $t \ge 1$ we express μ_{t+1} in terms of μ_t as follows:

$$\begin{aligned}
\mu_{t+1}(x) &= \mathrm{P}\left\{Y_{1:t+1} = y_{1:t+1}, X_{t+1} = x\right\} \\
&\overset{(a)}{=} \mathrm{P}\left\{Y_{1:t} = y_{1:t}, X_{t+1} = x\right\} \mathrm{P}\left\{Y_{t+1} = y_{t+1} \mid X_{t+1} = x\right\} \\
&= B\left(x, y_{t+1}\right) \sum_{x'\in\mathcal{X}} \mathrm{P}\left\{Y_{1:t} = y_{1:t}, X_{t+1} = x, X_t = x'\right\} \\
&\overset{(b)}{=} B\left(x, y_{t+1}\right) \sum_{x'\in\mathcal{X}} \mathrm{P}\left\{Y_{1:t} = y_{1:t}, X_t = x'\right\} \mathrm{P}\left\{X_{t+1} = x \mid X_t = x'\right\} \\
&= B\left(x, y_{t+1}\right) \sum_{x'\in\mathcal{X}} \mu_t\left(x'\right) \mathrm{A}\left(x', x\right), \quad t = 1, 2, \cdots, n-1
\end{aligned}$$

where (a) holds because $Y_{1:t} \to X_{t+1} \to Y_{t+1}$ forms a Markov chain, and (b) because $Y_{1:t} \to X_t \to X_{t+1}$ forms a Markov chain.

(2) The second factor in the product of $\mathrm{P}\left\{\vec{Y} = \vec{y}, X_t = x\right\}$ is

$$\nu_t(x) = \mathrm{P}\left\{Y_{t+1:n} = y_{t+1:n} \mid X_t = x\right\}, \quad x \in \mathcal{X}, 1 \le t < n.$$

Starting from $\nu_n(x) \equiv 1$, we have the following **backward recursion**, expressing ν_{t-1} in terms of ν_t for $2 \le t \le n$:

$$\begin{aligned}
\nu_{t-1}(x) &= \mathrm{P}\left\{Y_{t:n} = y_{t:n} \mid X_{t-1} = x\right\} \\
&= \sum_{x'\in\mathcal{X}} \mathrm{P}\left\{Y_{t:n} = y_{t:n}, X_t = x' \mid X_{t-1} = x\right\} \\
&\overset{(a)}{=} \sum_{x'\in\mathcal{X}} \mathrm{P}\left\{Y_{t:n} = y_{t:n} \mid X_t = x'\right\} \mathrm{P}\left\{X_t = x' \mid X_{t-1} = x\right\} \\
&\overset{(b)}{=} \sum_{x'\in\mathcal{X}} \mathrm{P}\left\{Y_{t+1:n} = y_{t+1:n} \mid X_t = x'\right\} \mathrm{P}\left\{Y_t = y_t \mid X_t = x'\right\} \mathrm{P}\left\{X_t = x' \mid X_{t-1} = x\right\} \\
&= \sum_{x'\in\mathcal{X}} \nu_t\left(x'\right) \mathrm{B}\left(x', y_t\right) \mathrm{A}\left(x, x'\right), \quad t = n, n-1, \cdots, 2
\end{aligned}$$

where (a) holds because $X_{t-1} \to X_t \to Y_{t:n}$ forms a Markov chain, and (b) because $Y_{t+1:n} \to X_t \to Y_t$ forms a Markov chain.

12.3.2 $\xi_t(x,x') \triangleq \mathrm{P}\left\{X_t = x, X_{t+1} = x' \mid \vec{Y} = \vec{y}\right\}$

Next we derive an expression for

$$\xi_t(x,x') = \mathrm{P}\left\{X_t = x, X_{t+1} = x' \mid \vec{Y} = \vec{y}\right\} = \frac{\mathrm{P}\left\{\vec{Y} = \vec{y}, X_t = x, X_{t+1} = x'\right\}}{\sum_{x,x' \in \mathcal{X}} \mathrm{P}\left\{\vec{Y} = \vec{y}, X_t = x, X_{t+1} = x'\right\}}$$

We have

$$\begin{aligned}
&\mathrm{P}\left\{\vec{Y} = \vec{y}, X_t = x, X_{t+1} = x'\right\} \\
&\overset{(a)}{=} \mathrm{P}\left\{Y_{1:t+1} = y_{1:t+1}, X_t = x, X_{t+1} = x'\right\} \mathrm{P}\left\{Y_{t+2:n} = y_{t+2:n} \mid X_{t+1} = x'\right\} \\
&\overset{(b)}{=} \mathrm{P}\left\{Y_{1:t} = y_{1:t}, X_t = x\right\} \mathrm{P}\left\{X_{t+1} = x' \mid X_t = x\right\} \mathrm{P}\left\{Y_{t+1} = y_{t+1} \mid X_{t+1} = x'\right\} \nu_{t+1}(x') \\
&= \mu_t(x) \mathrm{A}(x,x') \mathrm{B}(x', y_{t+1}) \nu_{t+1}(x')
\end{aligned}$$

where (a) holds because $(Y_{1:t+1}, X_t) \to X_{t+1} \to Y_{t+2:n}$ forms a Markov chain, and (b) because $Y_{1:t} \to X_t \to X_{t+1} \to Y_{t+1}$ forms a Markov chain. Hence

$$\xi_t(x,x') = \frac{\mu_t(x) \mathrm{A}(x,x') \mathrm{B}(x', y_{t+1}) \nu_{t+1}(x')}{\sum_{x,x' \in \mathcal{X}} \mu_t(x) \mathrm{A}(x,x') \mathrm{B}(x', y_{t+1}) \nu_{t+1}(x')}, \quad 1 \le t \le n, x, x' \in \mathcal{X}$$

12.3.3 Scaling Factors

Unfortunately the recursions above are numerically unstable for large n because the probabilities $\mu_t(x)$ and $\nu_t(x)$ vanish exponentially with n and are sums of many small terms of different sizes. The following approach is more stable. Define

$$\begin{aligned}
\alpha_t(x) &= \mathrm{P}\left\{X_t = x \mid Y_{1:t} = y_{1:t}\right\}, \\
\beta_t(x) &= \frac{\mathrm{P}\left\{Y_{t+1:n} = y_{t+1:n} \mid X_t = x\right\}}{\mathrm{P}\left\{Y_{t+1:n} = y_{t+1:n} \mid Y_{1:t} = y_{1:t}\right\}}, \\
c_t &= \mathrm{P}\left\{Y_t = y_t \mid Y_{1:t-1} = y_{1:t-1}\right\}
\end{aligned}$$

Then

$$\begin{aligned}
\gamma_t(x) &= \alpha_t(x)\beta_t(x) \\
\xi_t(x,x') &= c_t \alpha_t(x) \mathrm{B}(x, y_t) \mathrm{A}(x,x') \beta_t(x')
\end{aligned}$$

A forward recursion can be derived for α_t and c_t, and a backward recursion for β_t.

The time and storage complexity of the algorithm is $O\left(n|\mathcal{X}|^2\right)$.

Chapter 13 Graphic Models

To compute $P(x_1, ..., x_d)$, we can utilize the chain rule $P(x_1, ..., x_d) = P(x_1) \prod_{i=2}^{d} P(x_i|x_{1:i-1})$. However, this approach becomes computationally expensive as the dimension d increases.

Fortunately, when there are conditionally independent relationships between variables, such as $x_A \perp x_C | x_B$, we can reduce the computational cost.

In this section, we can employ graphical models to represent probabilistic relationships between variables, particularly when there are conditionally independent relationships present.

13.1 Graph Theory

1. A graph (V, E), V is a set of *vertices*, $E \subseteq V \times V$ is a set of ordered pairs of vertices, called *edges*.

   An edge $(i, j) \in E$ is *directed* if $(i, j) \notin E$; otherwise the edge is *undirected*. We denote directed and undirected edges by the symbols $i \to j$ and $i \sim j$, respectively.
2. **Directed and Undirected Graphs:** Graphs in which *all* edges are directed (resp. undirected).
3. **Subgraph:** a subgraph (S, E_S) of (V, E) is a subset $S \subset G$ with edges that have both endpoints in S.
4. **Clique:** A set C of vertices in an undirected graph is a clique if either C is a singleton, or **each pair of vertices in C is linked by an edge**.

   That is, all vertices in C are neighbors. The clique is maximal if there is no larger clique that contains C.
5. **Parent, Child:** Vertex i is a parent of vertex j if $i \to j$, in which case j is also called a child of i. We denote by $\pi(j)$ the set of parents of j.
6. **Path:** A *path* of length n from i to j is a sequence $i = k_0, k_1, ..., k_n = j$ of distinct vertices such that $(k_{m-1}, k_m) \in E$ for all $m = 1, ..., n$. We designate such a path by $i \to j$.
7. **Connected Graph:** An undirected graph is connected if there is a path between any pair of nodes. In general, the connected components of a graph are those subgraphs which are connected.
8. **Cycle/Loop:** An $n-$cycle, or loop, is a path of length n $i \to j$ with $i = j$.

   A directed graph without cycle is also called Directed Acyclic Graph (DAG)
9. **Tree:** A tree is a connected, undirected graph without cycles; **it has a unique path between any two vertices**.
10. **Rooted Tree:** A rooted tree is the directed acyclic graph obtained from a tree by choosing as vertex as root and directing all edges away from this root. Each vertex of a rooted tree has at most one parent.
11. **Forest:** A forest is an undirected graph where all connected components are trees.

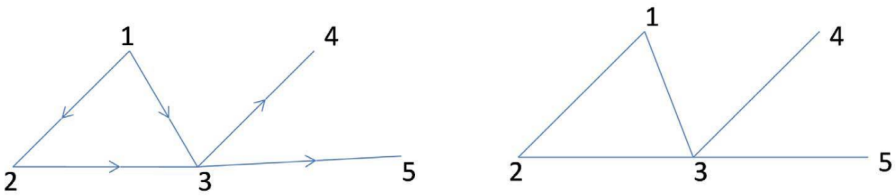


Figure 13.1: (a) Directed and (b) Undirected graph.

13.2 Bayesian Networks

A Bayesian network (or belief network) is a joint probability distribution associated with a *directed acyclic graph* (V, E) whose nodes $X_v, v \in V$ are random variables. The joint distribution is of the form

$$p(\vec{x}) = \prod_{v \in V} p(x_v | \pi(x_v))$$

$\pi(x_v)$ is the set of parents of vertices.

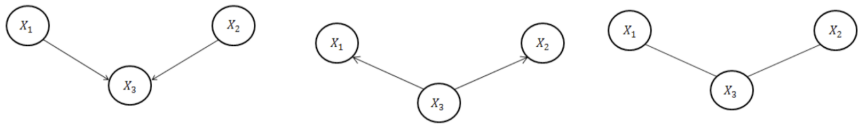
For instance a Markov chain is a chain-type directed acyclic graph where $V = \{1, 2, ..., n\}$, and $\pi(v) = v - 1$ for $v \geq 2$. The pmf for the sequence $\vec{x}$ is obtained from the chain rule

$$p(\vec{x}) = p(x_1)p(x_2|x_1) \cdots p(x_n|x_{n-1})$$

13.3 Markov Networks

13.3.1 General Form

We can use undirected graph to represent conditionally independent.



(a) $X_1 \perp\!\!\!\perp X_2$ (b) $X_1 \perp\!\!\!\perp X_2 | X_3$ (c) $X_1 \perp\!\!\!\perp X_2 | X_3$

Figure 13.2: (a) (b) Two Bayesian networks and (c) a Markov network.

More generally, if two nodes X_u and X_v in a Markov network are not connected by an edge, then the random variables X_u and X_v are conditionally independent given all the other random variables (denoted by $X_u \perp X_v \mid X_{\mathcal{V} \setminus \{u,v\}}$).

A Markov network is an undirected graph $G = (V, E)$ together with a collection $X = \{X_v, v \in V\}$ of random variables indexed by the nodes of G.

Since there is no direction, we use **clique** to help use represent probabilities. (*Review: clique is a set of vertices that each pair of vertices is linked*)

We use Ω let be collection of cliques in the graph and the functions $\psi_C(\cdot)$ be the ***clique potentials*, or *compatibility functions***.

The pmf of X takes the form

$$p(\vec{x}) = \frac{\prod_{C\in\Omega} \psi_C(\vec{x}_C)}{\sum_{\vec{x}} \prod_C \psi_C(\vec{x}_C)} = \frac{1}{Z} \prod_{C\in\Omega} \psi_C(\vec{x}_C)$$

where $Z = \sum_{\vec{x}} \prod_C \psi_C(\vec{x}_C)$ is a normalization constant.

Note: this is a form of factorization that can represent conditionally independent relationship among variables. $\psi_C(\cdot)$ are undefined functions.x

13.3.2 Hammersley-Clifford theorem

Theorem 13.1 (Hammersley-Clifford theorem)

Assume that $p(x_1, \ldots, x_n) > 0$ (positivity condition). Then,

$$p(\vec{x}) = \frac{1}{Z} \prod_{C\in\Omega} \phi_C(\vec{x}_C)$$

Thus, the following are equivalent (given the positivity condition):

1. ***Local Markov property:*** *$p(x_i \mid \vec{x} \setminus \{x_i\}) = p(x_i \mid \mathcal{N}(x_i))$, where $\mathcal{N}(x_i)$ is the neighboring set of x_i.*
2. ***Factorization property:*** *The probability factorizes according to the cliques of the graph.*
3. ***Global Markov property:*** *$p(\vec{x}_A \mid \vec{x}_B, \vec{x}_S) = p(\vec{x}_A \mid \vec{x}_S)$ whenever $\vec{x}_A$ and $\vec{x}_B$ are separated by $\vec{x}_S$ in G*

♡

13.3.3 Form of Gibbs distribution (Boltzmann distribution)

The factorization is not unique. We let $\psi(\vec{x}_C) = e^{-V_C(\vec{x}_C)}$, where $V_C(\cdot)$ are the so-called potential energy functions. In a pairwise Markov network, $p(\vec{x})$ can be expressed as a product of clique potentials involving either one or two random variables.

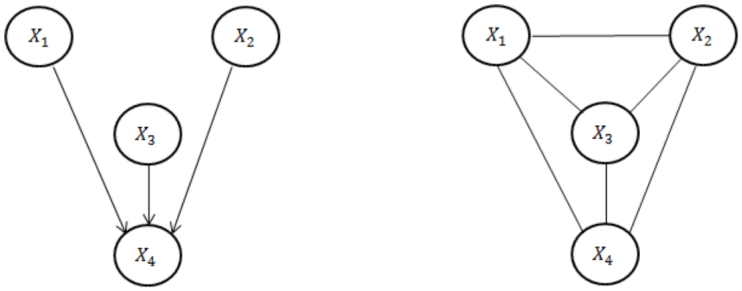
$$p(\vec{x}) = \frac{1}{Z} e^{-\sum_C V_C(x_C)}$$

This probability follows **Gibbs distribution (Boltzmann distribution)**. This distribution follows exponential families.

13.4 Conversion of directed graph to undirected graph

We can use a step known as *moralization*. Moralization of graph: connect two unmarried parents.

This is the process of "marrying" the parents of each node, i.e., adding an edge connecting any pair of parents if one did not exist. The figure illustrates this process for a node with three parents. In this case the undirected graph consists of a clique of size 4.



(a) A node with three unmarried parents (b) Moralized graph

Figure 13.3: Graph moralization

13.5 Inference and Learning

13.5.1 Inference on Trees

Consider the tree of the figure, which has 5 nodes and edges $1 \sim 2 \sim 3$ and $4 \sim 3 \sim 5$. We have

$$p(\vec{x}) = \frac{1}{Z}\psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)\psi_{34}(x_3, x_4)\psi_{35}(x_3, x_5)$$

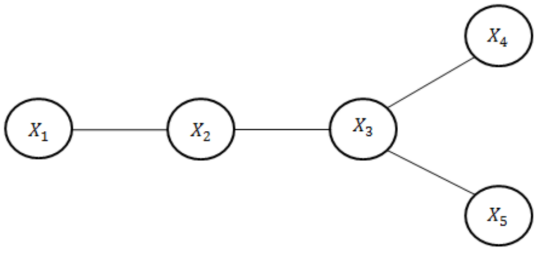


Figure 13.4: Example 1

1. **Marginal Inference:** As a first example of inference on trees, consider the problem of evaluating the marginal pmf $p(x_5)$. We explore two approaches: the **direct approach**, which is computationally

infeasible for large graphs (the number of items in the sum is $|\mathcal{X}|^4$);

$$p(x_5) = \sum_{x_1,\ldots,x_4} \frac{1}{Z}\psi_{12}(x_1,x_2)\psi_{23}(x_2,x_3)\psi_{34}(x_3,x_4)\psi_{35}(x_3,x_5)$$

and the **sum-product algorithm**, which exploits the graph structure.

$$\begin{aligned}
p(x_5) &= \frac{1}{Z}\sum_{x_3}\psi_{35}(x_3,x_5)\sum_{x_4}\psi_{34}(x_3,x_4)\sum_{x_2}\psi_{23}(x_2,x_3)\underbrace{\sum_{x_1}\psi_{12}(x_1,x_2)}_{m_{1\to 2}(x_2)} \\
&= \frac{1}{Z}\sum_{x_3}\psi_{35}(x_3,x_5)\sum_{x_4}\psi_{34}(x_3,x_4)\underbrace{\sum_{x_2}\psi_{23}(x_2,x_3)\,m_{1\to 2}(x_2)}_{m_{2\to 3}(x_3)} \\
&= \frac{1}{Z}\sum_{x_3}\psi_{35}(x_3,x_5)\,m_{2\to 3}(x_3)\underbrace{\sum_{x_4}\psi_{34}(x_3,x_4)}_{m_{4\to 3}(x_3)} \\
&= \underbrace{\frac{1}{Z}\sum_{x_3}\psi_{35}(x_3,x_5)\,m_{2\to 3}(x_3)\,m_{4\to 3}(x_3)}_{m_{3\to 5}(x_5)}.
\end{aligned}$$

In this derivation, nodes $1, 2, 4, 3$ are eliminated in that order. We think of each term $m_{i\to j}(x_j)$ as a message conveyed from node i to node j, just before elimination of j. Computing $m_{i\to j}(x_j)$ involves a summation over all possible values of x_i. This interpretation will be helpful in more complex problems.

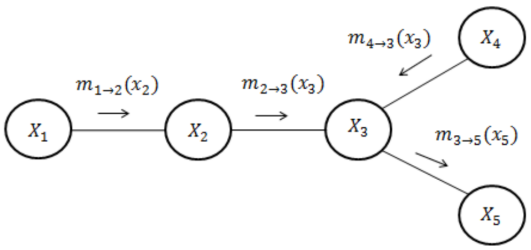


Figure 13.5: Belief propagation in a tree

As illustrated in the Figure, a node can send a message to a neighbor once it has received messages from all of its other neighbors. For a general tree, upon choosing an elimination order, we evaluate the following messages in the corresponding order:

$$m_{i\to j}(x_j) = \sum_{x_i}\psi_{ij}(x_i,x_j)\prod_{k\in\mathcal{N}(i)\setminus\{j\}} m_{k\to i}(x_i)$$

The marginal probability at any node i is the product of all incoming messages:

$$p(x_i) = \frac{1}{Z}\prod_{k\in\mathcal{N}(i)} m_{k\to i}(x_i).$$

We can also evaluate the 2D marginal $p(x_2, x_5)$

$$p(x_2, x_5) = \frac{1}{Z} \sum_{x_3} \psi_{23}(x_2, x_3) \psi_{35}(x_3, x_5) \underbrace{\sum_{x_4} \psi_{34}(x_3, x_4)}_{m_{4\to 3}(x_3)} \underbrace{\sum_{x_1} \psi_{12}(x_1, x_2)}_{m_{1\to 2}(x_2)}.$$

Finally, a conditional marginal such as $p(x_1 \mid x_5)$ is obtained as $p(x_1, x_5)/p(x_5)$, hence the problem is reduced to evaluating unconditional marginals.

The computational cost of the algorithm is $O\left(n|\mathcal{X}|^2\right)$ when the n random variables are defined over the same alphabet $\mathcal{X}$.

2. **Maximization:** A closely related problem is to find the most likely configuration, possibly by fixing some coordinates. For instance, evaluate

$$M(x_5) = \max_{x_1, x_2, x_3, x_4} p(x_1, x_2, x_3, x_4, x_5)$$

for the above Markov network. Direct calculation has exponential complexity. However, the more efficient max-product algorithm has the same structure as the sum-product algorithm:

$$\begin{aligned}
M(x_5) &= \max_{x_1, x_2, x_3, x_4} \frac{1}{Z} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \psi_{34}(x_3, x_4) \psi_{35}(x_3, x_5) \\
&= \frac{1}{Z} \max_{x_3} \psi_{35}(x_3, x_5) \max_{x_4} \psi_{34}(x_3, x_4) \max_{x_2} \psi_{23}(x_2, x_3) \underbrace{\max_{x_1} \psi_{12}(x_1, x_2)}_{m_{1\to 2}(x_2)} \\
&= \frac{1}{Z} \max_{x_3} \psi_{35}(x_3, x_5) \underbrace{\max_{x_4} \psi_{34}(x_3, x_4)}_{m_{4\to 3}(x_3)} \underbrace{m_{x_2} \psi_{23}(x_2, x_3) m_{1\to 2}(x_2)}_{m_{2\to 3}(x_3)} \\
&= \frac{1}{Z} \underbrace{\max_{x_3} \psi_{35}(x_3, x_5) m_{4\to 3}(x_3) m_{2\to 3}(x_3)}_{m_{3\to 5}(x_5)}
\end{aligned}$$

Chapter 14 Variational Inference, Mean-Field Techniques

Approximate complicated p.d.f. $p(\vec{x})$ with tractable $q(\vec{x})$, where $q \in Q =$ tractable set of distributions.

Use divergence to measure:

$$\min_{q \in Q} \quad D(q\|p)$$

note that D is convex in q.

14.1 Naive Mean-Field Methods

The *naive mean field method* approximates a distribution by a product distribution.

Assume the q has the form $q(\vec{x}) = \prod_{i=1}^{n} q_i(x_i)$. Assume $x_i \in X =$ finite set.

$$\begin{aligned} D(q\|p) &= \mathbb{E}_q \left[\ln \frac{q(\vec{X})}{p(\vec{X})} \right] \\ &= \sum_{i=1}^{n} \mathbb{E}_q \left[\ln q_i(X_i) \right] - \mathbb{E}_q \left[\ln p(\vec{X}) \right] \\ &= \sum_{i=1}^{n} \sum_{x_i \in X} q_i(x_i) \ln q_i(x_i) - \sum_{\vec{x} \in X^n} \left(\prod_{i=1}^{n} q_i(x_i) \right) \ln p(\vec{x}) \end{aligned}$$

Solve

$$\begin{aligned} &\min_{\{q_i\}} \quad D(\prod_{i=1}^{n} q_i \| p) \\ &s.t. \sum_{x_i \in X} q_i(x_i) = 1, i = 1, ..., n \end{aligned}$$

Using Lagrangian method:

$$L(q, \vec{\lambda}) = D(q\|p) + \sum_{i=1}^{n} \lambda_i \left(\sum_{x \in X} q_i(x_i) - 1 \right)$$

$$0 = \frac{\partial L(q, \vec{\lambda})}{\partial q_i(x_i)} = 1 + \ln q_i(x_i) - \sum_{\vec{x}' : x'_i = x_i} \left(\prod_{j \neq i}^{n} q_j(x'_j) \right) \ln p(\vec{x}') + \lambda_i$$

Hence, $q_i(x_i)$ should in the form:

$$\begin{aligned} q_i(x_i) &= \frac{1}{e^{1+\lambda_i}} e^{\sum_{\vec{x}' : x'_i = x_i} \left(\prod_{j \neq i}^{n} q_j(x'_j) \right) \ln p(\vec{x}')} \\ &= \frac{1}{Z_i} exp \left(\mathbb{E}_{\prod_{j \neq i}^{n} q_j} [\ln p(X_{1:i-1}, x_i, X_{i+1:n})] \right) \end{aligned}$$

Iteration Algorithm:

$$q_i^{(k+1)}(x_i) = \frac{1}{Z_i} exp \left(\mathbb{E}_{\prod_{j \neq i}^{n} q_j^{(k)}} [\ln p(X_{1:i-1}, x_i, X_{i+1:n})] \right)$$

14.1.1 Graphical Models

Consider $P =$ pairwise Markov model

$$p(\vec{x}) = \frac{1}{Z} \prod_{(i,j)\in E} \psi_{ij}(x_i, x_j)$$

$$\ln p(\vec{x}) = -\ln Z + \sum_{(i,j)\in E} \ln \psi_{ij}(x_i, x_j)$$

The expectation can be written as

$$\begin{aligned}
\mathbb{E}_{\prod_{j\neq i}^{n} q_j}[\ln p(X_{1:i-1}, x_i, X_{i+1:n})] &= \mathbb{E}_{\prod_{j\neq i}^{n} q_j}[\sum_{(i,j)\in E} \ln \psi_{ij}(x_i, x_j)] + cst \\
&= \sum_{j\in N(i)} \mathbb{E}_{q_j}[\ln \psi_{ij}(x_i, x_j)] + cst \\
&= \sum_{j\in N(i)} \sum_{x_j \in X} q_j(x_j) \ln \psi_{ij}(x_i, x_j) + cst
\end{aligned}$$

and thus,

$$q_i(x_i) = \frac{1}{Z_i} exp\left(\sum_{j\in N(i)} \sum_{x_j \in X} q_j(x_j) \ln \psi_{ij}(x_i, x_j) \right)$$

14.1.2 Ising Model

Consider a 2-D torus V with $|V| = n$ nodes, and $X = \{\pm 1\}$. Each node is connected to its upper, lower, right, and left neighbors. The distribution is of the form

$$p(\vec{x}) = \frac{1}{Z} exp\left(\beta \sum_{(i,j)\in E} x_i x_j \right), \quad \vec{x} \in \{\pm 1\}^n$$

with $\beta \geq 0$. The parameter β represents the inverse of a temperature. For $\beta = 0$ the distribution is uniform, hence fully factorized. For large positive values of β, configurations $\vec{x}$ with strong correlations are favored.

2-D Ising Model

$$\psi_{ij}(x_i, x_j) = e^{\beta x_i x_j}$$

$$\Rightarrow q_i(x_i) = \frac{1}{Z_i} \prod_{j\in N(i)} exp\left(- \sum_{x_j = \pm 1} q_j(x_j) \beta x_i x_j \right)$$

Since each X_i is a Bernoulli random variable, the decision varibale q_i can be represented by a single parameter which we choose to be the mean $m_i = q_i(1) - q_i(-1) \in [-1, 1]$. Equivalently,

$$\begin{aligned}
m_i = q_i(1) - q_i(-1) \Leftrightarrow q_i(1) = \frac{1 + m_i}{2},\ q_i(-1) = \frac{1 - m_i}{2} \\
\Rightarrow q_i(x_i) = \frac{1}{Z_i} \prod_{j\in N(i)} exp\left(-\beta x_i m_j\right)
\end{aligned}$$

Then, our problem is finding the optimal $\{m_i\}$.

$$q_i(1) = \frac{1}{Z_i} exp\left(-\beta \sum_{j \in N(i)} m_j\right) ;\ q_i(-1) = \frac{1}{Z_i} exp\left(\beta \sum_{j \in N(i)} m_j\right)$$

The normalization constant is given by

$$Z_i = exp\left(\beta \sum_{j \in N(i)} m_j\right) + exp\left(-\beta \sum_{j \in N(i)} m_j\right) = 2cosh\left(\beta \sum_{j \in N(i)} m_j\right)$$

Hence,

$$m_i = q_i(1) - q_i(-1) = tanh\left(\beta \sum_{j \in N(i)} m_j\right)$$

Convergence. We show that the algorithm always converges if a uniform initialization is used, i.e., $m_i^{(0)} = m^{(0)}$ for all $i \in V$. Then the (simultaneous) update equation for the means is

$$m^{(k+1)} = tanh\left(4\beta m^{(k)}\right)$$

where the factor of 4 arises because each vertex has 4 neighbors. Analysis of convergence depends on the value of β, and we need consider two cases.

(1) **Case I:** $\beta < \frac{1}{4}$**:** The mapping is a contraction mapping for $\beta < \frac{1}{4}$, and so the fixed point of this mapping is $\lim_{k \to \infty} m^{(k)} = 0$, for any initialization $m^{(0)}$. Hence, the variational approximation is uniform: $q(\vec{x}) = 2^{-n}$ for all $x \in \{\pm 1\}^V$.

(2) **Case II:** $\beta > \frac{1}{4}$**:** In this case, the equation $m = \tanh(4\beta m)$ has three possible solutions 0 and $\pm m^*$ where $m^* > 0$. If the algorithm is initialized with $m^{(0)} = 0$, then subsequent iterations do not change this value. If the algorithm is initialized with $m^{(0)} > 0$, it converges to m^*. Finally, if the algorithm is initialized with $m^{(0)} < 0$, it converges to $-m^*$. In the latter two cases (convergence to either m^* or $-m^*$), the variational approximations q_i are nonuniform.

(3) **Case III:** $\beta = \frac{1}{4}$**:** phase transition

The case $\beta > \frac{1}{4}$ is related to percolation theory in statistical physics. It may be shown that the distribution p favors configurations featuring large homogeneous regions. The correlation between any two nodes is significant, even for large graphs. This behavior is completely different from the case $\beta < \frac{1}{4}$, where the correlation between distant nodes dies out with distance (similarly to a homogeneous, irreducible Markov chain). The case $\beta = \frac{1}{4}$ is known as a phase transition.

14.2 Exponential Families of Probability Distributions

Definition 14.1 ($d-$dimensional exponential families)

In canonical form: $d-$dimensional exponential families

$$p_\theta(y) = \frac{h(y)}{Z(\theta)} e^{\sum_{k=1}^{d} \theta_k T_k(y)}$$

$T_k(\cdot)$ are called sufficient statistics. ***Partition function*** *$Z(\theta)$ is the normalization constant ensuring that the density p_θ integrates to 1.*

It can also be written

$$p_\theta(y) = e^{\theta^T T(y) - A(\theta)}$$

The ***log partition function*** *(aka cumulant function) $A(\theta) = lnZ(\theta)$*

♣

Example 1: $P_\theta = N(\theta, 1)$,

$$p_\theta(y) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(y-\theta)^2}{2}} = \frac{e^{\frac{y^2}{2}}}{\sqrt{2\pi} e^{-\frac{\theta^2}{2}}} e^{-\theta y}$$

Example 2: $P_\theta = N(0, \theta^{-1})$, θ is the inverse covariance matrix.

$$p_\theta(y) = \frac{|\theta|^{\frac{1}{2}}}{\sqrt{2\pi}} e^{-\frac{1}{2} y^T \theta y}$$

Example 3: 2D-Ising Model

$$p_\theta(y) = \frac{1}{Z(\theta)} e^{\theta y_i y_j}, \theta > 0$$

Generalized 2D-Ising Model

$$p_\theta(y) = \frac{1}{Z(\theta)} e^{\sum_{i \in V} \theta_i y_i + \sum_{i \sim j} \theta_{ij} y_i y_j}$$

The natural parameter set:

$$\Theta = \{\theta : \int_X e^{\theta^T T(y)} dy < \infty\}$$

The divergence in exponential form

$$\begin{aligned} D(P_\theta \| P_{\theta'}) &= \mathbb{E}_\theta \left[\ln \frac{P_\theta(Y)}{P_{\theta'}(Y)} \right] \\ &= \mathbb{E}_\theta [\theta^T T(Y) - A(\theta) - (\theta')^T T(Y) + A(\theta')] \\ &= -[A(\theta) - A(\theta')] + (\theta - \theta')^T \mathbb{E}_\theta [T(Y)] \end{aligned}$$

Cumulant-generating function (cgf)

$$\kappa(u) = \ln \mathbb{E}(e^{u^T T(Y)}) = \ln \int e^{u^T T(y)} (e^{\theta^T T(y) - A(\theta)}) dy$$

$$\nabla \kappa(0) = \mathbb{E}_\theta[T(Y)]$$

$$\nabla^2 \kappa(0) = \mathrm{Cov}_\theta[T(Y)]$$

We can compute

$$\int e^{\theta^T T(y) - A(\theta)} dy = 1$$

$$\int [T(y) - \nabla A(\theta)] e^{\theta^T T(y) - A(\theta)} dy = 0$$

$$\int [T(y) - \nabla A(\theta)] p_\theta(y) dy = 0$$

$$\mathbb{E}_\theta[T(Y)] = \nabla A(\theta)$$

Hence,

$$\nabla A(\theta) = \nabla \kappa(0) = \mathbb{E}_\theta[T(X)]$$

$$\nabla^2 A(\theta) = \nabla^2 \kappa(0) = \mathrm{Cov}_\theta[T(X)]$$

Definition. The set of realizable mean parameters $\mathcal{M}$ is the set of μ that are the expected value of $T(X)$ under some distribution p (not necessarily in the exponential family). Thus

$$\mathcal{M} \triangleq \left\{ \mu \in \mathbb{R}^d : \exists p : \mathbb{E}_p[T(X)] = \mu \right\}$$

which is a convex set.

Example 6. Consider Generalized 2D-Ising Model

$$p_\theta(y) = \frac{1}{Z(\theta)} e^{\sum_{i \in V} \theta_i y_i + \sum_{i \sim j} \theta_{ij} y_i y_j}$$

For $y \in \{0, 1\}$

$$\mu_i = \mathbb{E}_\theta \left[T_i(Y) \right] = P_\theta(Y_i = 1)$$

$$\mu_{ij} = \mathbb{E}_\theta \left[T_i(Y) T_j(Y) \right] = P_\theta(Y_i = Y_j = 1)$$

Example 7. If $T(x) = xx^\top \in \mathbb{R}^{n \times n}$ then μ is a correlation matrix, and so $\mathcal{M}$ is the set of all $n \times n$ symmetric nonnegative definite matrices.

14.3 ML Estimation

Consider n iid samples $X^{(i)}, 1 \leq i \leq n$ drawn from the exponential distribution p_θ. The ML estimator of θ given these n samples is obtained by solving

$$\begin{aligned}\hat{\theta}_{ML} &= \max_\theta \frac{1}{n} \sum_{i=1}^n \ln p_\theta \left(X^{(i)}\right) \\ &= \max_\theta \frac{1}{n} \sum_{i=1}^n \left[\theta^\top T\left(X^{(i)}\right) - A(\theta)\right] \\ &= \max_\theta \left[\theta^\top \hat{\mu} - A(\theta)\right]\end{aligned}$$

where $\hat{\mu}$ is the mean parameter

$$\hat{\mu} \triangleq \frac{1}{n} \sum_{i=1}^n T\left(X^{(i)}\right)$$

$A(\theta)$ is a convex function, we can solve optimal solution by solving critical point.

$$\nabla A(\hat{\theta}_{ML}) = \hat{\mu}$$

The gradient mapping could be hard to invert, however. For instance, for the Ising model example we easily obtain

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n \sum_{j \sim k} X_j^{(i)} X_k^{(i)}$$

but inverting the gradient mapping is a hard problem. Such is generally the case if p is a distribution over a Markov network with cycles.

14.4 Maximum Entropy

Consider a random variable X over a finite set $\mathcal{X}$. Its probability distribution p is unknown, however we are given the expected value $\mu_k = \mathbb{E}_p\left[T_k(X)\right]$ of d statistics $T_k(X), 1 \leq k \leq d$. A classical problem, which originates from statistical physics, is to find p that maximizes the entropy $H(p) = -\sum_x p(x) \ln p(x)$ subject to the d constraints above. Assuming the feasible set is nonvoid, the resulting distribution is called the maximum-entropy (or maxent()) distribution.

Since $H(p)$ is concave, the constraints are linear in p, and the probability simplex is a convex set, the maxent problem is concave. Its solution is obtained by introducing d Lagrange multipliers $\lambda_k, 1 \leq k \leq d$ associated with the mean constraints, and a Lagrange multiplier λ_{d+1} associated with the constraint $\sum_x p(x) = 1$. Ignoring momentarily the nonnegativity constraints, we maximize the Lagrangian

$$\mathcal{L}(p, \lambda) \triangleq -\sum_{x \in \mathcal{X}} p(x) \ln p(x) + \sum_{k=1}^d \lambda_k \left(\sum_{x \in \mathcal{X}} p(x) T_k(x) - \mu_k\right) + \lambda_{d+1} \left(\sum_{x \in \mathcal{X}} p(x) - 1\right)$$

over p, subject to the $d+1$ equality constraints

The first-order optimality conditions are given by

$$0 = \frac{\partial \mathcal{L}(p,\lambda)}{\partial p(x)} = -\ln p(x) - 1 + \sum_{k=1}^{d} \lambda_k T_k(x) + \lambda_{d+1}$$

Hence,

$$p(x) = \frac{1}{Z} e^{\sum_{k=1}^{d} \lambda_k T_k(x)}$$

where $Z = e^{1-\lambda_{d+1}}$

$$H(p) = \mathbb{E}_p[\ln p(X)] = -\theta^T \mathbb{E}_p[T(X)] + A(\theta) = -\max_{\theta}(\theta^T \mu - A(\theta))$$

Example. Let $X = (X_1, X_2) \in \{0,1\}^2$ and consider maximizing entropy subject to the constraint $\mathbb{E}[X_1 X_2] = \mu$ where $\mu \in (0,1)$. We obtain $p(x) = \frac{1}{Z}\exp\{\lambda x_1 x_2\}$. Since $\sum_x p(x) = 1$, the normalization constant is obtained as $Z = e^{\lambda} + 3$. We obtain λ from the constraint

$$\mu = \mathbb{E}_P[X_1 X_2] = \frac{e^{\lambda}}{e^{\lambda}+3} \quad \Rightarrow \quad \lambda = \ln \frac{3\mu}{1-\mu}$$

The maxent solution takes the form

$$p(x) = \begin{cases} \mu & (x_1, x_2) = (1,1) \\ \frac{1-\mu}{3} & \text{else} \end{cases}$$

and has entropy is $H(p) = -\mu \ln \mu - (1-\mu)\ln \frac{1-\mu}{3}$.

A similar version of the maxent problem exists for continuous random variables. The entropy function is replaced with the differential entropy functional $h(p) \triangleq -\int p \ln p$, and the maxent solution again takes an exponential form.

14.5

$$\begin{aligned} \min_{q \in Q} D(q\|p) &= \min_q \mathbb{E}_q \left[\ln \frac{q(x)}{p_\theta(x)}\right] \\ &= \min_q \left[A(\theta) - \theta^T \mathbb{E}_q[T(x)] - H(q)\right] \\ &= A(\theta) - \max_{\mu} \max_{q:\mathbb{E}_q[T(x)]=\mu} \left[\theta^T \mathbb{E}_q[T(x)] + H(q)\right] \\ &= A(\theta) - \max_{\mu \in M} \left[\theta^T \mu + \max_{q:\mathbb{E}_q[T(x)]=\mu} H(q)\right] \end{aligned}$$

Since $\max_{q:\mathbb{E}_q[T(x)]=\mu} H(q)$ is exactly an entropy maximum problem, we let $A^*(\mu) = \max_{q:\mathbb{E}_q[T(x)]=\mu} H(q)$.

As we showed: (1). $A^*(\mu) = \max_\theta[\theta^T\mu - A(\theta)]$, $A(\theta) = \max_\mu[\theta^T\mu - A^*(\mu)]$

$$\min_{q\in Q} D(q\|p) = A(\theta) - \max_{\mu\in M}\left[\theta^T\mu + A^*(\mu)\right]$$

14.6 Connection between Exponential Families and Graphic Models

Pairwise Markov network over $G(V, E)$

$$\begin{aligned} p(\vec{x}) &= \frac{1}{Z}\left(\prod_{i\in V}\psi_i(x_i)\right)\left(\prod_{(i,j)\in E}\psi_{ij}(x_ix_j)\right) \\ &= \frac{1}{Z}e^{\sum_{i\in V}\ln\psi_i(x_i)+\sum_{(i,j)\in E}\ln\psi_{ij}(x_ix_j)} \\ &= \frac{1}{Z}e^{\sum_{i\in V}\sum_{x\in X}\ln\psi_i(x_i)\mathbf{1}_{x=i}+\sum_{(i,j)\in E}\sum_{x,x'\in X}\ln\psi_{ij}(x_ix_j)\mathbf{1}_{x=i,x'=j}} \end{aligned}$$

We can let

$$\begin{aligned} T_{ix}(x) &= \mathbf{1}_{x_i=x}, && \forall i\in V, \forall x\in X \\ \theta_i(x) &= \ln\psi_i(x), && \forall i\in V, \forall x\in X \\ T_{ijxx'}(x,x') &= \mathbf{1}_{x_i=x,x_i=x'}, && \forall (i,j)\in E, \forall x,x'\in X \\ \theta_{ij}(x) &= \ln\psi_{ij}(x,x'), && \forall (i,j)\in E, \forall x,x'\in X \end{aligned}$$

The probability can be transformed into exponential families.

The dimension of this family is $d = |V||X| + |E||X|^2$.

14.6.1 Marginal polytope

Definition 14.2 (marginal polytope)

The mean parameters associated with the distribution p_θ are the 1-dimensional marginals for the vertices,

$$\mu_i(x) = \mathbb{E}_\theta[T_{ix}(x)] = P_\theta(x_i = x) = \textit{marginal distribution of } X_i$$

and the pairwise marginals associated with the edge set E,

$$\mu_{ij}(x,x') = \mathbb{E}_\theta[T_{ijxx'}(x,x')] = P_\theta(x_i = x, x_j = x') = \textit{2D marginal distribution of } (X_i, X_j)$$

*The set of realizable mean parameters, $\mathcal{M}$, is then called the **marginal polytope** and denoted by $\mathcal{M}(G)$.* ♣

14.6.2 Locally Consistent Marginal Distributions

Given a graph $G = (V, E)$, consider the set of marginal distributions τ_i on individual nodes $i \in V$ and pairwise marginals τ_{ij} on edges $(i, j) \in E$ that are locally consistent in the sense

$$\sum_x \tau_i(x) = 1,\ \tau_i(x) \geq 0, \qquad \forall i \in V, x \in X$$

$$\sum_{x,x'} \tau_{ij}(x,x') = 1,\ \tau_{ij}(x,x') \geq 0, \qquad \forall (i,j) \in E, x, x' \in X$$

$$\sum_{x_j \in X} \tau_{ij}(x_i,x_j) = \tau_i(x_i), \qquad \forall (i,j) \in E, x_i, x_j \in X$$

$$\sum_{x_i \in X} \tau_{ij}(x_i,x_j) = \tau_j(x_j), \qquad \forall (i,j) \in E, x_i, x_j \in X$$

Definition 14.3 (local marginal polytope)

*The **local marginal polytope** $\mathcal{L}(G)$ is the set of $\tau = \left(\{\tau_i\}_{i\in V}, \{\tau_{ij}\}_{(i,j)\in E}\right)$ that satisfy the above consistency conditions.*

♣

This is a fairly simple polytope defined by $|V| + \left(2|X| + |X|^2\right)|E|$ linear constraints. Clearly the marginal polytope $\mathcal{M}(G)$ is a subset of $\mathcal{L}(G)$, but is the converse true?

Proposition 14.1 (For foster $\mathcal{M}(G) = \mathcal{L}(G)$)

If $G = (V, E)$ is a forest then $\mathcal{M}(G) = \mathcal{L}(G)$. Any probability distribution on G can be expressed as follows in terms of its $1 - D$ and pairwise marginals:

$$p(\vec{x}) = \left(\prod_{i\in V} \mu_i(x_i)\right)\left(\prod_{(i,j)\in E} \frac{\mu_{ij}(x_i,x_j)}{\mu_i(x_i)\,\mu_j(x_j)}\right)$$

♠

Proof 14.1

We first prove the claim for a tree. Any tree can be generated starting from a single node and adding one edge at a time. The claim can be proven by induction. It clearly holds for a graph consisting of two nodes and a single edge (i, j), since $p(x) = \mu_{ij}(x_i, x_j)$ in this case. If a new node k and a new edge (j, k) are added to an existing tree $(\mathcal{V}', \mathcal{E}')$ where $j \in \mathcal{V}'$, we obtain a new tree $(\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \mathcal{V}' \cup \{k\}$ and $\mathcal{E} = \mathcal{E}' \cup \{(j, k\}$. If

$$p(\boldsymbol{x}_{\mathcal{V}'}) = \left(\prod_{i\in\mathcal{V}'} \mu_i(x_i)\right)\left(\prod_{(i,j)\in\mathcal{E}'} \frac{\mu_{ij}(x_i,x_j)}{\mu_i(x_i)\,\mu_j(x_j)}\right)$$

then

$$p(\boldsymbol{x}) = p(\boldsymbol{x}_{\mathcal{V}'}, x_k) = p(\boldsymbol{x}_{\mathcal{V}'})\,p(x_k \mid x_j) = p(\boldsymbol{x}_{\mathcal{V}'})\frac{\mu_{jk}(x_j,x_k)}{\mu_j(x_j)}$$

satisfies the claim. Next, if the forest contains more than one tree, the distribution $p(\boldsymbol{x})$ factors over the trees, and the claim still holds.

Finally, to any $\mu \in \mathcal{L}(\mathcal{G})$ we can associate a global distribution p using the claim. The marginals and

pairwise marginals of p are given by μ, hence $\mu \in \mathcal{M}(\mathcal{G})$. This proves the first part of the claim.

Example 14.1 However for a graph that is not a tree, $\mathcal{M}(G)$ is in general a strict subset of $\mathcal{L}(G)$. Consider for instance the 3-cycle with node set $V = \{1, 2, 3\}$ and edge set $E = \{(1; 2); (2; 3); (3; 1)\}$. Let $X = \{0, 1\}$ and consider τ_1, τ_2, τ_3 that are uniform over X, and

$$\tau_{12} = \tau_{23} = \begin{bmatrix} 0.5 - \varepsilon & \varepsilon \\ \varepsilon & 0.5 - \varepsilon \end{bmatrix}, \ \tau_{31} = \begin{bmatrix} \varepsilon & 0.5 - \varepsilon \\ 0.5 - \varepsilon & \varepsilon \end{bmatrix}$$

for some $\epsilon \in (0, 0.5)$. By inspection, $\tau \in \mathcal{L}(\mathcal{G})$. However, for ϵ small enough, the definitions of $\tau_{12}, \tau_{23}, \tau_{31}$ imply respectively that $X_1 = X_2, X_2 = X_3$, and $X_3 \neq X_1$ with high probability. These conditions are incompatible, hence $\tau \notin \mathcal{M}(\mathcal{G})$.

In this example, the edge set is small, and it is relatively easy to determine that $\tau \notin \mathcal{M}(\mathcal{G})$. For a large graph, this would generally not be computationally feasible. Since τ may not be the marginals of any joint distribution on $\mathcal{G}, \tau$ are often referred to as **pseudomarginals**.

Definition 14.4 (pseudomarginal)

*Marginal distribution τ such that $\tau \in \mathcal{L}(\mathcal{G})$ and $\tau \notin \mathcal{M}(\mathcal{G})$ is called **pseudomarginals**.* ♣

14.6.3 Entropy on Tree Graphs

Any distribution p defined on a tree graph is of the form $p(\vec{x}) = \left(\prod_{i \in V} \mu_i(x_i)\right) \left(\prod_{(i,j) \in E} \frac{\mu_{ij}(x_i, x_j)}{\mu_i(x_i)\mu_j(x_j)}\right)$. Hence, its entropy is given by

$$\begin{aligned} H(p) &= \mathbb{E}_p[-\ln p(\vec{X})] \\ &= \sum_{i \in V} \mathbb{E}_p\left[-\ln \mu_i(X_i)\right] - \sum_{(i,j) \in E} \mathbb{E}_p\left[\ln \frac{\mu_{ij}(X_i, X_j)}{\mu_i(X_i)\,\mu_j(X_j)}\right] \\ &= \sum_{i \in V} H(\mu_i) - \sum_{(i,j) \in E} I(\mu_{ij}) \end{aligned}$$

where

$$\begin{aligned} I(\mu_{ij}) &\triangleq \mathbb{E}_{\mu_{ij}}\left[\ln \frac{\mu_{ij}(X_i, X_j)}{\mu_i(X_i)\,\mu_j(X_j)}\right] \\ &= D(\mu_{ij} \| \mu_i \mu_j) \\ &= H(\mu_i) + H(\mu_j) - H(\mu_{ij}) \end{aligned}$$

is the **mutual information** associated with the pairwise marginal μ_{ij}. Since this is a Kullback-Leibler divergence, it is nonnegative. The mutual information is zero if X_i and X_j are independent random variables and is upper-bounded by both $H(\mu_i)$ and $H(\mu_j)$ (value achieved if X_j is a function of X_i, or vice-versa).

The entropy of p is easily computed but is not concave in μ. Equivalently, the set of distributions p on a tree

graph is generally nonconvex. (Consider a length-3 chain for instance.)

14.6.4 Naive Mean-Field Methods In Graph

Approximate complicated p.d.f. $p_\theta(\vec{x}) = e^{\theta^T I(\vec{x}) - A(\theta)}$ in $G = (V, E)$ with tractable $q(\vec{x}) \in G'(V, E')$, where $E' \subset E$. $q(\vec{x}) = \prod_{i \in V} q_i(x_i)$ (naive mean field method assumes fully factorized).

Minimizing divergence:

$$\begin{aligned} D(q\|p_\theta) &= \mathbb{E}_q \left[\ln \frac{q(\vec{x})}{p_\theta(\vec{x})} \right] \\ &= -\theta^T \mathbb{E}_q \left[T(\vec{x}) \right] + A(\theta) - H(q) \end{aligned}$$

$Q = \{q : \mathbb{E}_q[T(\vec{x})] = \mu, \mu \in M'\}$

$$\min_{q \in Q} D(q\|p_\theta) = A(\theta) - \max_{\mu \in M'} [\theta^T \mu - A^*(\mu)]$$

$$\max_{\{\mu_i\}_{i \in V}} [\theta^T \mu - A^*(\mu)] = \sum_{i \in V} \sum_{x \in X} \theta_{ix} \mu_i(x) + \sum_{(i,j) \in E} \sum_{x, x'} \theta_{ijxx'} \mu_{ij}(x, x') + \sum_{i \in V} H(\mu_i)$$

Taking Lagrangian and taking derivative

$$0 = \frac{\partial L(\mu, \lambda)}{\partial \mu_i(x)} \Rightarrow \mu_{i(x)} = \frac{1}{Z} e^{\theta_{ix} + \sum_{i \in N(i)} \sum_{x' \in X} \theta_{ijxx'} \mu_j(x')}$$

14.6.5 Structural Mean Field Optimization

$q(\vec{x}) = q_1(x_1) q_2(x_2|x_1) q_3(x_3|x_2)$ (Markov Chain in a tree $G' = (V, E')$)

$$\mu_{12}(x_1, x_3) = \sum_{x_2} p(x_1, x_2, x_3) = \sum_{x_2} p(x_1, x_2) p(x_3|x_2) = \sum_{x_2} p(x_1, x_2) \frac{p_{23}(x_2, x_3)}{p(x_2)}$$

14.6.6 Bethe Entropy Approximation

When we compute the entropy of a tree graph, the entropy equals to

$$H(p) \triangleq \sum_{i \in V} H(\mu_i) - \sum_{(i,j) \in E} I(\mu_{ij})$$

(**only holds in tree graph!**).

In a more general situation. For a distribution p that is not defined on a tree graph, $H(p)$ does not admit a simple expression, and cannot be expressed simply in terms of 1-D marginals and pairwise marginals. (Verify on a 3-cycle). However, if these marginals are known, one could use an approximation to $H(p)$.

Definition 14.5 (Bethe approximation)

We use the equation satisfied in tree graph to approximate entropy in general situations. This approxi-

mation is known as the ***Bethe approximation****, and the functional*

$$H_{Bethe}(\tau) \triangleq \sum_{i \in V} H(\tau_i) - \sum_{(i,j) \in E} I(\tau_{ij}), \quad \tau \in \mathcal{L}(\mathcal{G})$$

is known as the ***Bethe entropy****. This "entropy" is well defined for all pseudomarginals* $\tau \in \mathcal{L}(\mathcal{G})$*.*

The ***Bethe variational problem*** *is defined as*

$$A_{Bethe}(\theta) \triangleq \max_{\tau \in \mathcal{L}(\mathcal{G})} \left[\theta^\top \tau + H_{Bethe}(\tau)\right]$$

and is relatively tractable owing to the simple nature of $\mathcal{L}(\mathcal{G})$ *and the availability of a closed-form expression for* $H_{Bethe}(\tau)$*.*

♣

Compare with the expression

$$A(\theta) = \sup_{\mu \in \mathcal{M}(\mathcal{G})} \left[\theta^\top \mu + H\left(p_{\theta(\mu)}\right)\right]$$

that is unfortunately intractable because of the complex nature of $\mathcal{M}(\mathcal{G})$ and the lack of an explicit form for $H(p_\mu)$. For a general graph, $\mathcal{M}(\mathcal{G}) \subset \mathcal{L}(\mathcal{G})$ and Bethe entropy is an approximation to entropy; $A_{\text{Bethe}}(\theta)$ is not a bound on $A(\theta)$, only an approximation (see example below). For a tree graph however, $\mathcal{M}(\mathcal{G}) = \mathcal{L}(\mathcal{G})$ and $A_{\text{Bethe}}(\theta) = A(\theta)$

Example 14.2 (Inexactness of Bethe approximation) Consider a fully connected graph with four nodes, $V = \{1, 2, 3, 4\}$, uniform 1-D marginals $\mu_i, i \in V$, and pairwise marginals. $\mu_{ij} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \forall i, j \in V \Rightarrow$ $X_i = X_j$ w.p.1 . $\vec{x} = [0, 0, 0, 0]$ or $[1, 1, 1, 1]$ with probability 0.5 each.

We have $\mu \in \mathcal{M}(\mathcal{G})$; indeed the distribution p that places probability $\frac{1}{2}$ on the sequences $(0, 0, 0, 0)$ and $(1, 1, 1, 1)$ satisfies the marginal constraints above. We have $H(\mu_i) = \ln 2$ for all $i \in \mathcal{V}$ and $I(\mu_{ij}) = \ln 2$ for all $i \neq j \in \mathcal{V}$. Since there are 6 edges, we obtain

$$H_{\text{Bethe}}(\mu) = 4 \ln 2 - 6 \ln 2 = -2 \ln 2 < 0$$

which shows that the Bethe entropy does not satisfy the same properties as an entropy (it can be negative). The actual entropy $H(p) = \ln 2 > 0$.

Chapter 15 ℓ_1 Penalized Least Squares Minimization

We will focus on an ℓ_1-penalized least-squares problem where the objective function is the sum of a quadratic function representing "fit to the data" and a regularization term which is the l1 norm of an unknown signal to be recovered. The first term is smooth, the second is not.

15.1 Problem Statement

Given an observation vector $y \in \mathbb{R}^m$, a $m \times n$ matrix A, a constant $\lambda > 0$, find a vector $x \in \mathbb{R}^n$ that achieves the minimum of

$$f(x) \triangleq \frac{1}{2}\|y - \mathrm{A}x\|^2 + \lambda\|x\|_1.$$

The first component is half the squared ℓ_2 norm of $r \triangleq y - Ax$, which can be interpreted as an observation error. The second component, $\|x\|_1 \triangleq \sum_{i=1}^{n} |x_i|$, is the ℓ_1 norm of x.

The problem admits a Bayesian interpretation, in which the observations y are the sum of Ax and white Gaussian noise with mean zero and variance σ^2,

$$y = Ax + z, \text{ where } z \sim N(0, \sigma^2)$$

and x is a realization of a random vector with iid entries following a double-exponential (Laplace) distribution. In this case,

$$\ln p(y \mid x) + \ln p(x) = -\frac{n}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\|y - \mathrm{A}x\|^2 - n\ln 2 - \|x\|_1$$

hence minimizing f is equivalent to MAP estimation for the above Bayesian problem, with $\lambda = 2\sigma^2$.

The structure of A depends on the application. In signal processing and computer vision, A is usually related to a convolution operator, describing for instance motion blur in video. In compressive sensing, the entries of A are typically iid random variables. The algorithms we will focus on do not require A to have any special structure. We begin with two important simple cases (identity and orthonormal A), then move to the general problem.

15.2 Special Cases

15.2.1 Definition: Soft Threshold

Definition 15.1 (Soft Threshold)

$$S_\lambda(y) \triangleq \begin{cases} y-\lambda, & \textit{for } y \geq \lambda \\ y+\lambda & \textit{for } y \leq -\lambda \\ 0 & \textit{for } |y| < \lambda \end{cases}$$

♣

15.2.2 Identity A

Let $m = n$ and A be the identity matrix

$$f(x) \triangleq \sum_{i=1}^{n} \left(\frac{1}{2}(y_i - x_i)^2 + \lambda |x_i| \right)$$

$$f'(x) = \begin{cases} -y + x + \lambda \text{sign}(x), & x \neq 0 \\ \text{does not exist}, & x = 0 \end{cases}$$

The solution x is obtained by applying a soft threshold to each component y_i of the observations,

$$0 = f'(x) \Rightarrow x = S_\lambda(y) \triangleq \begin{cases} y-\lambda, & \text{for } y \geq \lambda \\ y+\lambda & \text{for } y \leq -\lambda \\ 0 & \text{for } |y| < \lambda \end{cases}$$

15.2.3 Orthonormal A

If $m = n$ and A is orthonormal, then $A^{-1} = A^T$ and

$$\|y - Ax\|^2 = \|A(A^T y - x)\|^2 = \|A^T y - x\|^2$$

$$\Rightarrow x = S_\lambda(A^T y)$$

15.2.4 Quadratic Optimization ($\lambda = 0$)

We now consider general A. It is useful to first study the case $\lambda = 0$, in which case f is quadratic and the optimization problem is smooth. The solution x satisfies the necessary first-order optimality condition

$$0 = \nabla f(x) = -\mathrm{A}^\top (y - \mathrm{A}x) \quad \in \mathbb{R}^n.$$

If $\text{rank}(\mathrm{A}) \geq n$ (which implies $m \geq n$), the unique solution is $x = \left(\mathrm{A}^\top \mathrm{A}\right)^{-1} \mathrm{A}^\top y$.

Otherwise, the solution is nonunique. Any $x = \mathrm{A}^+ y + z$ where $z \in \text{Null}(\mathrm{A})$ and $\mathrm{A}^+ \in \mathbb{R}^{n \times m}$ is the **Moore**

pseudo-inverse of A, is a solution. The minimum-norm solution is $x = \mathrm{A}^{+}y$.

Even though a closed-form solution exists, for large n one would avoid the computationally expensive matrix inverse and use an iterative algorithm such as gradient descent or conjugate gradient to derive the solution. The gradient descent update takes the form

$$x^{k+1} = x^k + \alpha \mathrm{A}^{\top}\left(y - \mathrm{A}x^k\right), \quad k = 1, 2, 3, \cdots$$

where α is the step size.

15.3 General Solution: Lasso

15.4 General Solution: Iterative Soft Thresholding Algorithm (ISTA)

The idea is to tackle the difficult optimization problem by solving a sequence of simple optimization problems. Often (as is the case here) the simple optimization problems will admit an easily-computable closed-form solution.

15.4.1 Proximal Minimization Algorithm

Definition 15.2 (Proximal Minimization Algorithm)

Consider a convex function F and a $n \times n$ positive definite matrix W. The iterative algorithm with update equation

$$x^{k+1} = \arg\min_{x \in \mathbb{R}^n} \left\{ F(x) + \frac{1}{2}\left(x - x^k\right)^T \mathrm{W}\left(x - x^k\right) \right\}$$

*is a **proximal minimization algorithm**.*

If W is suitably chosen, the algorithm is a majorization-minimization algorithm.

Lemma 15.1 (Nondecreasing and Convergence)

Such algorithms are of the form

$$x^{k+1} = \arg\min_{x \in \mathbb{R}^n} Q\left(x, x^k\right)$$

where the function $Q(\cdot, x')$ is easy to minimize, $Q(x, x) = F(x)$, and $Q(x, x') \geq F(x)$. The sequence $F\left(x^k\right)$ is nondecreasing because by application of the above properties and the definition of x^{k+1}

$$F\left(x^{k+1}\right) \leq Q\left(x^{k+1}, x^k\right) \leq Q\left(x^k, x^k\right) = F\left(x^k\right).$$

One may also show that the sequence x^k converges to a minimum of F.

15.4.2 Apply to ℓ_1-penalized least-squares

To apply this strategy to ℓ_1-penalized least-squares, let $F(x) = \frac{1}{2}\|y - \mathrm{A}x\|^2 + \lambda\|x\|_1$ and

$$Q\left(x, x'\right) = F(x) + \frac{c}{2}\left\|x - x'\right\|^2 - \frac{1}{2}\left\|\,\mathrm{A}\left(x - x'\right)\right\|^2$$

where c is larger than the maximum eigenvalue of $\mathrm{A}^\top\mathrm{A}$. Then $\mathrm{W} \triangleq c\mathbf{I}_n - \mathrm{A}^\top\mathrm{A}$ is a symmetric positive definite matrix and

$$\begin{aligned} Q\left(x, x^k\right) &= F(x) + \frac{c}{2}\left\|x - x^k\right\|^2 - \frac{1}{2}\left\|\,\mathrm{A}\left(x - x^k\right)\right\|^2 \\ &= F(x) + \frac{1}{2}\left(x - x^k\right)^\top \underbrace{\left(c\mathbf{I}_n - \mathrm{A}^\top\mathrm{A}\right)}_{=\mathrm{W}}\left(x - x^k\right) \end{aligned}$$

satisfies the properties of a majorizing function.

We now show that $Q\left(\cdot, x^k\right)$ is easy to minimize:

$$\begin{aligned} Q\left(x, x^k\right) &= \lambda\|x\|_1 + \frac{1}{2}\|y - \mathrm{A}x\|^2 + \frac{c}{2}\left\|x - x^k\right\|^2 - \frac{1}{2}\left\|\,\mathrm{A}\left(x - x^k\right)\right\|^2 \\ &= \lambda\|x\|_1 - x^\top \underbrace{\left[\mathrm{A}^\top\left(y - \mathrm{A}x^k\right) + cx^k\right]}_{=cu^k} + \frac{c}{2}\|x\|^2 + \text{constant} \\ &= \frac{c}{2}\left\|x - u^k\right\|^2 + \lambda\|x\|_1 + \text{constant} \end{aligned}$$

where $u^k \triangleq x^k + \frac{1}{c}\,\mathrm{A}^\top\left(y - \mathrm{A}x^k\right)$. The minimization of $Q\left(\cdot, x^k\right)$ takes the same form as *identity A situation*.

Hence, the solution is obtained in closed form using the componentwise soft threshold operator:

$$x^{k+1} = \arg\min_x Q\left(x, x^k\right) = S_{\frac{\lambda}{c}}\left(u^k\right) = S_{\frac{\lambda}{c}}\left(\frac{1}{c}\,\mathrm{A}^\top\left(y - \mathrm{A}x^k\right) + x^k\right)$$

This is the update equation for the **Iterative Soft Thresholding Algorithm (ISTA)**. Observe that the equation is an extension of the gradient descent update for the purely quadratic problem (with $\lambda = 0$ and step size $\alpha = \frac{1}{c}$).

15.5 Convergence Rate

We say **linear convergence** if

$$\|x^k - x^*\| \le ab^k$$

where $a > 0$ and $b \in (0, 1)$.

The sufficient condition of **linear convergence** is

$$\lim_{k\to\infty}\sup\frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} \le \beta$$

for some $\beta \in (0, 1)$.

As an application, we consider $\min_x\{f(x) = x^T Q x\}$ where $Q \succ 0$, achieves x^* at 0.

GD $\Rightarrow$ step size $\alpha < \frac{2}{\lambda_{\max}(Q)}$. The optimal step size $\alpha_{opt} = \frac{2}{\lambda_{\max}(Q)+\lambda_{\min}(Q)}$

Condition number of Q is $\kappa = \frac{\lambda_{\max}(Q)}{\lambda_{\min}(Q)} \geq 1$. With $\alpha = \alpha_{opt}$, $\beta = \frac{\|x^{k+1}\|}{\|x^k\|} = \frac{\kappa-1}{\kappa+1} < 1$

Heavy -Ball Method

$$x^{k+1} = x^k - \alpha \nabla f(x^k) + \beta(x^k - x^{k-1})$$

(with momentum), $\beta > 0$. For minimization of $\min f(x) = x^T Q x$, heavy ball can then be shown to be equivalent to conjugate gradient when α and β are optimized.

15.6 Fast Iterative Soft Thresholding Algorithm (FISTA)

In the ISTA, updates take the form of

$$x^{k+1} = S_{\lambda/c}\left(\frac{1}{c}\,\mathrm{A}^\top\left(y - \mathrm{A}x^k\right) + x^k\right)$$

In the FISTA, consider the sequence $t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$, initialized with $t_1 = 1$. It can easily be shown that $t_k = \frac{k}{2}[1 + o(1)]$

And the FISTA takes the form:

$$x^{k+1} = S_{\lambda/c}\left(\frac{1}{c}\,\mathrm{A}^\top\left(y - \mathrm{A}\tilde{x}^k\right) + \tilde{x}^k\right)$$
$$\tilde{x}^{k+1} = x^k + \frac{t_k - 1}{t_{k+1}}\left(x^k - x^{k-1}\right)$$

where $\tilde{x}^1 = x^{(0)}$ and the second term in the right side is a "momentum" term, as in the heavy ball algorithm. The constant c should be larger than the maximum eigenvalue of $A^\top A$.

Analysis of the convergence rate of FISTA-type algorithms is a current research topic.

15.7 Alternating Direction Method of Multipliers (ADMM)

The Alternating Direction Method of Multipliers (ADMM) is an augmented Lagrangian method.

The general idea is to reformulate the minimization problem

$$\min_x \{g(x) + h(x)\}$$

as

$$\min_{x,z} \{g(x) + h(z)\} \quad \text{subj. to} \quad x = z$$

which is solved using an augmented Lagrangian approach.

Identify $g(x) = \frac{1}{2}\|y - \mathrm{A}x\|^2$ and $h(x) = \lambda\|x\|_1$, fix some penalty parameter $\nu > 0$, and write the augmented

Lagrangian as

$$\mathcal{L}(x,z,u) = \frac{1}{2}\|y - \mathrm{A}x\|^2 + \lambda\|z\|_1 + \frac{\nu}{2}\|x - z\|^2 + u^\top(x - z)$$

which is linear in the vector of Lagrange multipliers u, strongly convex in (x, z), and nonsmooth in z but easy to minimize over z given (x, u). The update equations are

$$\begin{aligned}
x^{k+1} &= \arg\min_x \mathcal{L}\left(x, z^k, u^k\right) \\
&= \left(\mathrm{A}^\top\mathrm{A} + \nu\mathrm{l}\right)^{-1}\left(\mathrm{A}^\top y + \nu z^k - u^k\right) \\
z^{k+1} &= \arg\min_z \mathcal{L}\left(x^{k+1}, z, u^k\right) \\
&= S_{\lambda/\nu}\left(x^{k+1} - \nu^{-1}u^k\right) \\
u^{k+1} &= u^k + \nu\left(x^{k+1} - z^{k+1}\right), \quad k = 1, 2, 3, \cdots .
\end{aligned}$$

Chapter 16 Compressive Sensing

The problem is to recover a sparse signal $x \in \mathbb{R}^n$ by solving $y = Ax$, where $y \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and $x \in \mathbb{R}^n$ is a sparse vector. $m << n$. In general this would be a severely underdetermined linear system, but recovery is possible if the signal x is sparse (contains mostly zeroes), or at least approximately sparse.

16.1 Definitions related to Sparsity

Definitions:

1. **Definition 16.1 (ℓ_p norm)**

   The ℓ_p norm of a vector $x \in \mathbb{R}^n$ is

   $$\|x\|_p \triangleq \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \quad \text{for all } p \geq 1$$

2. **Definition 16.2 (ℓ_0 pseudonorm)**

   The ℓ_0 pseudonorm of a vector $x \in \mathbb{R}^n$ is

   $$\|x\|_0 \triangleq \sum_{i=1}^{n} \mathbf{1}_{\{x_i \neq 0\}}$$

   i.e., the number of nonzero components of x.

3. **Definition 16.3 (k-sparse)**

   A signal $x \in \mathbb{R}^n$ is k-sparse if

   $$\|x\|_0 \leq k$$

   i.e., the number of nonzero components of x is smaller than k.

4. **Definition 16.4 (set of k-sparse signals)**

   The set of k-sparse signals is

   $$\Sigma_k \triangleq \{x \in \mathbb{R}^n : \|x\|_0 \leq k\}$$

   This is a union of $\binom{n}{k}$-dimensional subspaces of $\mathbb{R}^n$, which is not a linear space.

   For instance let $n = 2$ and $k = 1$, then Σ_1 is the union of the horizontal and vertical axes in the 2-D plane.

   Many signals are sparse in a transform domain (for instance, Fourier-sparse) or approximately sparse.

For instance, the wavelet coefficients of an image or a speech signal are approximately sparse, and one can typically construct good approximations of such signals by using only the largest 5% of their components.

Claim 16.1 (Producing k-sparse signal)

One can go from an approximately sparse to an (exactly) k-sparse signal by applying a ***hard threshold operator,*** *$\hat{x} = H_k(x)$, producing a vector* ***in which all but the k largest (in magnitude) coefficients of x are set to zero.***

♠

5.

Definition 16.5 (ℓ_p approximation error)

The ℓ_p approximation error of $x \in \mathbb{R}^n$ in Σ_k is

$$e_{k,p}(x) \triangleq \min_{\hat{x} \in \Sigma_k} \|x - \hat{x}\|_p, \quad p \geq 1$$

♣

Lemma 16.1 (Properties)

(a). If x is k-sparse, i.e., $x \in \Sigma_k$, then $e_{k,p}(x) = 0$ for all p. (Optimal $\hat{x}$ is exactly the x)

(b). Otherwise, the approximation error typically vanishes geometrically with k, i.e.,

$$e_{k,p}(x) \leq ck^{-r}$$

for some $c, r > 0$, generally dependent on x.

(c). It is easily shown that the minimum of error is achieved by $\hat{x} = H_k(x)$.

♡

Other forms of sparsity are frequently encountered:

- **Low-dimensional manifolds:** let Θ be a compact subset of $\mathbb{R}^k$ and $f : \Theta \to \mathbb{R}^n$ a continuously differentiable mapping, then $x = f(\theta), \theta \in \Theta$ belongs to a k-dimensional manifold embedded in $\mathbb{R}^n$. This model applies to face images under varying illumination.
- **Low-rank matrices:** let x be a $n_1 \times n_2$ matrix of rank $r < \min(n_1, n_2)$. Then x can be represented using the singular value decomposition $x = \sum_{j=1}^{r} \sigma_j u_j v_j^\top$ where σ_j are the (positive) singular values, and $u_j \in \mathbb{R}^{n_1}$ and $v_j \in \mathbb{R}^{n_2}$ are the singular vectors for $1 \leq j \leq r$. This model finds applications to computer vision, geolocalization, and collaborative filtering (cf the Netflix recommender system).

16.2 Measurement Matrix

Definition 16.6 (Measurement Matrix)

The measurement matrix A is a $m \times n$ matrix where $m \ll n$ (fat matrix). The observations are given by $\vec{y} = A\vec{x} \in \mathbb{R}^m$ in case of noise-free measurements. ♣

In this section we consider two basic questions are:

1. What properties should $A \in \mathbb{R}^{m \times n}$ have so that $\vec{x} \in \mathbb{R}^n$ can be recovered from $\vec{y} \in \mathbb{R}^m$?
2. If A satisfies those properties, what recovery algorithm can be used?

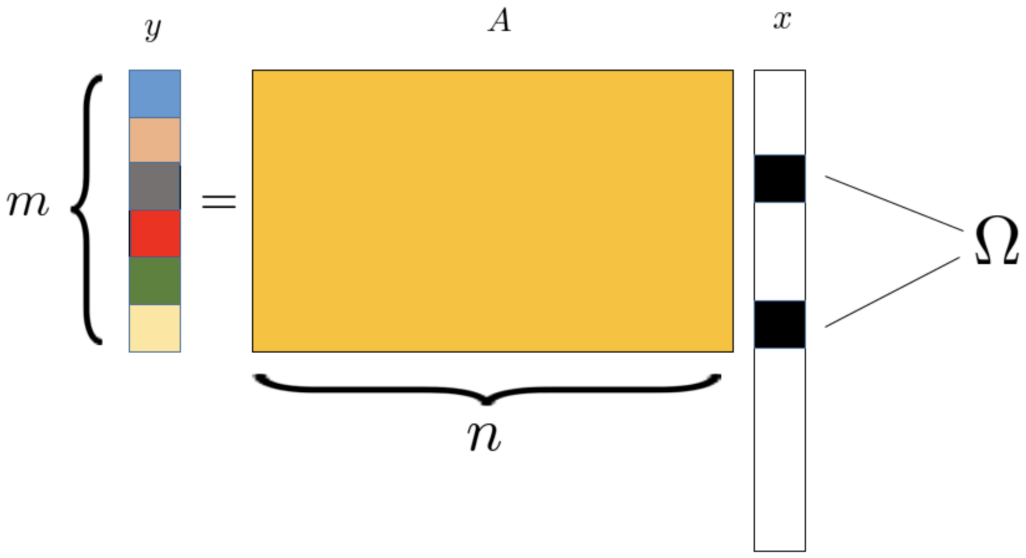


Figure 16.1: Measurement of sparse signal $\vec{x}$ with support set Ω of size k.

Denote by A_i, $1 \leq i \leq n$ the column vectors of A and let $\Omega \subset \{1, 2, ..., n\}$ be the support set of the vector $\vec{x}$, i.e., $\Omega = \{i : x_i \neq 0\}$. Define the reduced matrix $A_\Omega = \{A_i, i \in \Omega\}$ and the reduced vector $\vec{x}_\Omega = \{x_i, i \in \Omega\}$. Then

$$\vec{y} = A\vec{x} = A_\Omega \vec{x}_\Omega$$

16.2.1 Matrix Preliminaries

The SVD decomposition of A takes the form $A = U[D \mid 0]V^T$ where U and V are $m \times m$ and $n \times n$ unitary matrices (whose conjugate transpose equals to inverse), respectively, $\mathrm{D} = \mathrm{diag}\,(\sigma_k)_{k=1}^m$, and 0 is the $m \times (n - m)$ all-zero matrix. The m nonnegative entries $\sigma_k, 1 \leq k \leq m$, are the singular values of A. The

matrix A can be expanded as a sum of m rank-one matrices, $\mathrm{A} = \sum_{k=1}^{m} \sigma_k u_k v_k^\top$ where $\{u_k\}$ and $\{v_k\}$ are the columns of U and V, respectively.

The spectral norm of A is

$$\|\mathrm{A}\| \triangleq \sup_{\vec{x}:\|\vec{x}\|_2=1} \|\mathrm{A}\vec{x}\|_2$$

and is equal to the largest singular value of A.

The $m \times m$ Gram matrix $\mathrm{G} \triangleq \mathrm{A}\mathrm{A}^\top$ is symmetric and nonnegative definite and can be expressed as $\mathrm{G} = \mathrm{U}\mathrm{D}^2\mathrm{U}^\top$ hence its eigenvalues are given by $\lambda_k(\mathrm{G}) = \sigma_k^2(\mathrm{A}), 1 \le k \le m$.

Definition 16.7 (Null Space)

The null space of A *is*

$$N(\mathrm{A}) \triangleq \{\vec{x} \in \mathbb{R}^n : \mathrm{A}\vec{x} = 0\}$$

This is a subspace of $\mathbb{R}^n$*, whose dimension is at most* $n - m$*.*

Definition 16.8 (Spark)

The spark of A *is the smallest number of columns of* A *that are linearly dependent.*

If any set of q *columns of the matrix are linearly independent,* $\mathrm{spark}(A) = q + 1 \in [2, m + 1]$.

The minimum value of the spark of any matrix is 2, and is achieved by any matrix that has two identical columns. Computing the spark of a matrix in an NP-hard problem. In contrast, the rank of a matrix, which is the largest number of columns that are linearly independent, is easy to compute.

16.2.2 Recovery of k-Sparse Signals

Lemma 16.2

Unique recovery of $x \in \Sigma_k$ *given measurement matrix* A *is possible* $\Rightarrow \Sigma_{2k} \cap N(A) = \emptyset$.

Proof 16.1

To recover any k-sparse signal, we need that $A\vec{x} \neq A\vec{x}'$ *for any distinct* $\vec{x}, \vec{x}' \in \Sigma_k$*. Hence,* $\vec{x} - \vec{x}' \notin N(A)$*. Since* $\vec{x} - \vec{x}'$ *is a* $2k$*-sparse signal and in fact* $\Sigma_{2k} = \vec{x} - \vec{x}' : \vec{x}, \vec{x}' \in \Sigma_k$*, we need* $\Sigma_{2k} \cap N(A) = \emptyset$.

Theorem 16.1

Unique recovery of $\vec{x} \in \Sigma_k$ *given measurement matrix* A *is possible* $\Leftrightarrow \mathrm{spark}(A) > 2k$*.* $(q \ge 2k)$

Proof 16.2

Eldar, Y. C., & Kutyniok, G. (Eds.). (2012). Compressed sensing: theory and applications, Chapter 1.

16.2.3 Restricted Isometry Property

Definition 16.9 (Restricted Isometry Property (RIP))

The matrix A satisfies the RIP property of order k if there exists $\delta_k \in (0,1)$ such that

$$(1-\delta_k)\|\vec{x}\|_2^2 \leq \|A\vec{x}\|_2^2 \leq (1+\delta_k)\|\vec{x}\|_2^2, \quad \forall \vec{x} \in \Sigma_k$$

♣

If A satisfies the RIP property of order k, then A approximately preserves the ℓ_2 distance between any pair $\vec{x}, \vec{x}' \in \Sigma_k$. This provides a *stable embedding* of k-sparse signals in $\mathbb{R}^m$. This property will be key to derive a recovery algorithm that is robust to noise.

Claim 16.2 (Equivalent Formulation of RIP)

*Since $\vec{y} = A\vec{x} = A_\Omega \vec{x}_\Omega$, an **equivalent formulation** of the RIP is*

$$(1-\delta_k)\|\vec{x}_\Omega\|_2^2 \leq \|A_\Omega \vec{x}_\Omega\|_2^2 \leq (1+\delta_k)\|\vec{x}_\Omega\|_2^2, \quad \forall \Omega : |\Omega| = k$$

for all Ω of size k and for all $\vec{x}_\Omega \in \mathbb{R}^k$.

$$\frac{\|A_\Omega \vec{x}_\Omega\|_2^2}{\|\vec{x}_\Omega\|_2^2} - 1 = \frac{\vec{x}_\Omega^T \left(A_\Omega^T A_\Omega - I_k\right) \vec{x}_\Omega}{\vec{x}_\Omega^T \vec{x}_\Omega} \in [-\delta_k, \delta_k], \quad \forall \Omega : |\Omega| = k$$

$$\Rightarrow \delta_k = \max_{\Omega:|\Omega|=k} \|A_\Omega^T A_\Omega - I_k\|$$

♠

Theorem 16.2 (Measurement Bound)

Assume $A \in \mathbb{R}^{m \times n}$ satisfies the RIP of order $2k$ with RIP constant $\delta_{2k} \in (0, \frac{1}{2}]$. Then

$$m \geq ck \ln \frac{n}{k}$$

where the constant $c = \frac{1}{2}\ln(1+\sqrt{24}) \approx 0.28$.

♡

16.3 Robust Signal Recovery from Noiseless Observations

We now consider the so-called robust CS problem: the signal $\vec{x}$ is approximately sparse, and the observations are noiseless. A reasonable attempt to recover a sparse signal would be the so-called ℓ_0 recovery problem

$$\min_{\vec{x} \in \mathbb{R}^n} \|\vec{x}\|_0 \quad \text{subj. to} \quad A\vec{x} = \vec{y}$$

Unfortunately this problem is highly nonconvex. Solving it essentially requires evaluating all possible support sets of $\vec{x}$, which is a combinatorial problem.

A reasonable substitute is the so-called ℓ_1 recovery problem

$$\min_{\vec{x} \in \mathbb{R}^n} \|\vec{x}\|_1 \quad \text{subj. to} \quad A\vec{x} = \vec{y}$$

This procedure tends to produce sparse solutions, as illustrated in the Figure: the line $y = Ax$ is tangent to the

ℓ_1 ball $\|\boldsymbol{x}\|_1 = \text{cst}$ at the solution $\boldsymbol{x}$, producing a 1-sparse solution.

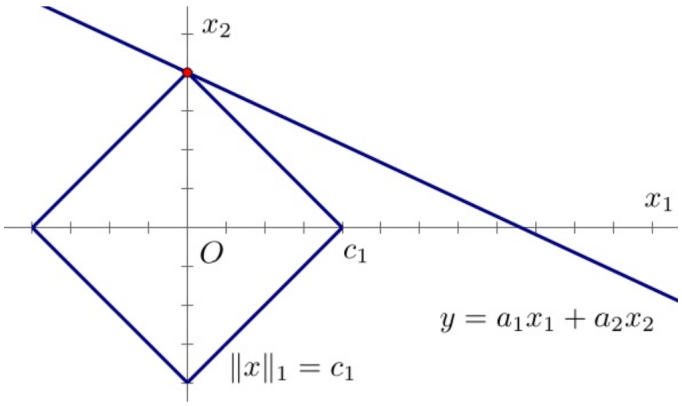


Figure 16.2: ℓ_1 recovery for $n = 2$ and $m = 1$

The following fundamental theorem shows that the ℓ_1 recovery procedure is remarkably good.

Theorem 16.3 (ℓ_1 recovery procedure is good)

Assume A satisfies the RIP of order $2k$ with constant $\delta_{2k} < \sqrt{2} - 1$. $\hat{x} = \text{argmin}_{\vec{x}\in\mathbb{R}^n : A\vec{x}=\vec{y}} \|\vec{x}\|_1$. Then

$$\|\hat{x} - x\|_2 \leq \frac{c}{\sqrt{k}} e_{k,1}(x)$$

and

$$\|\hat{x} - x\|_1 \leq c e_{k,1}(x)$$

with $c = 2\frac{1-(1-\sqrt{2})\delta_{2k}}{1-(1+\sqrt{2})\delta_{2k}}$. $e_{k,1}(x) \triangleq \min_{\hat{x}\in\Sigma_k} \|x - \hat{x}\|_1$ is the ℓ_1 approximation error of x in Σ_k. ♡

Proof 16.3

Based on the triangle inequality and the inequality $\frac{\|u\|_1}{\sqrt{k}} \leq \|u\|_2 \leq \sqrt{k}\|u\|_\infty$ for all $u \in \Sigma_k$.

Corollary 16.1

If $x \in \Sigma_k$ then $\hat{x} = x$ (exact recovery). ♡

If $x \notin \Sigma_k$ then the quality of the reconstruction is nearly as good as if an *oracle* gave us the location of the k largest absolute components and we measured those directly. (The oracle produces $\hat{x} = H_k(x)$, achieving $e_{k,p}(x)$ for all $p \geq 1$.)

Since $\delta_{2k} < \sqrt{2} - 1 < \frac{1}{2}$, the measurement bound $m \geq ck \ln \frac{n}{k}$ applies, and we need as few as $m = O\left(k \ln \frac{n}{k}\right)$ measurements to satisfy the conditions of the theorem.

16.4 Robust Signal Recovery from Noisy Observations

16.4.1 Bounded Noise

We consider observations corrupted by bounded noise: $y = \mathrm{A}x + z$ where $\|z\|_2 \leq \epsilon$. We study the ℓ_1 recovery problem

$$\min_{x \in \mathbb{R}^n} \|x\|_1 \quad \text{subj. to} \quad \|\mathrm{A}x - y\|_2 \leq \epsilon$$

which is closely related to the Lasso problem and can be solved using the algorithms introduced in the previous chapter.

Theorem 16.4

Assume A satisfies the RIP of order $2k$ with constant $\delta_{2k} < \sqrt{2} - 1$. Then

$$\|\hat{x} - x\|_2 \leq \frac{c_0}{\sqrt{k}} e_{1,k}(x) + c_1 \epsilon$$

with constants

$$c_0 = 2\frac{1 - (1 - \sqrt{2})\delta_{2k}}{1 - (1 + \sqrt{2})\delta_{2k}}, \quad \textit{and} \quad c_1 = 4\frac{\sqrt{1 + \delta_{2k}}}{1 - (1 + \sqrt{2})\delta_{2k}}.$$

♡

For $\delta_{2k} = \frac{1}{4}$ the theorem holds with $c_0 \leq 5.5$ and $c_1 \leq 6$. For $\epsilon = 0$, the result coincides with that given in the previous section (noise-free case).