Mathematical Background of Machine Learning/Deep Learning

Dongwoo Sheen

Department of Mathematics

Seoul National University

Seoul 08826, Korea

Email: dongwoosheen@gmail.com, sheen@snu.ac.kr

http://www.nasc.snu.ac.kr

December 17, 2020; 10:48

Chapter 1

DFFN

In this chapter we study the DFFN (Deep FeedForward Network) with an example of learning the XOR. The simple DFFN approach consists of one hidden layer plus an output unit. In particular we take the ReLU Hidden Units

- $\boxed{\text{feedforward + backpropagation}} = \boxed{\text{an epoch}}$
- $\boxed{\text{Deep feedforward network}} = \boxed{\text{MultiLayer Perceptrons(MLPs)}}$
- The goal of MLPs: approximate a function $f : \mathbb{R}^m \to \mathbb{R}^p$ that maps an input $\mathbf{x}$ to a category $\mathbf{y}$. We can write $f(\mathbf{x}; \boldsymbol{\theta})$.
- To approximate desire function, MLPs learns the value of $\boldsymbol{\theta}$ to improve approximation of f.
- MLPs are called networks since they are composed of many different functions, i.e. composed of many layers of function.
- Deep Learning: $f(\mathbf{x}; \boldsymbol{\theta}) \sim f_d \circ f_{d-1} \circ f_{d-2} \circ \cdots \circ f_1(\mathbf{x}; \boldsymbol{\theta})$
- f_d : output layer
- $f_j, j = 1, \cdots, d-1$: hidden layers
- The length of the chain of layers = the depth of the model = d
- The hidden layers and the output layer:
  - The training examples specify directly what the output layer must do at each point $\mathbf{x}^{(j)}$.
  - The behavior of the other layers is not directly specified by the training data.
  - The learning algorithm must decide how to use those layers to produce the desired output
  - The training data does not say what each individual layer should do.
  - The learning algorithm must decide how to use these layers to best implement an approximation of f^*.
  - Because the training data does not show the desired output for each of these layers, these layers are called hidden layers.

Example 1.1. *Consider the nonlinear function ϕ such that*

$$\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta}, \mathbf{w}) = \phi(\mathbf{x}; \boldsymbol{\theta})^T \mathbf{w}$$

- $\boldsymbol{\theta}$ *is parameter that we use to learn the desired output.*

- *ϕ defining a hidden layer.*
- *Human designer only needs to find proper general function family (e.g. tensors, polynomials, Fourier series, etc...) to approximate desired output rather than precisely right function.*
- *Training procedure is implemented by back-propagation algorithm commonly.*

1.1 Example: Learning XOR, "Exclusive OR"

- The XOR function f^* is and operation on $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

$$f^*(\mathbf{x}) = \begin{cases} 1 & \text{if } \|\mathbf{x}\|_1 = 1, \\ 0 & \text{otherwise.} \end{cases} .$$

- The XOR function is non-linear on $\mathbf{X} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ such that

$$f^*(\mathbf{x}_1) = f^*(\mathbf{x}_4) = 0; f^*(\mathbf{x}_2) = f^*(\mathbf{x}_3) = 1.$$

- The MSE loss function is defined as follows:

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\mathbf{x} \in \mathbf{X}} (f^*(\mathbf{x}) - f(\mathbf{x}; \boldsymbol{\theta}))^2$$

1.1.1 Linear model approach

- $f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b, \quad \boldsymbol{\theta} = \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ b \end{pmatrix}$. Hence,

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{4} \sum_{\mathbf{x} \in \mathbf{X}} \left(f^*(\mathbf{x}) - \mathbf{w}^T \mathbf{x} - b \right)^2 . \\ &= \frac{1}{4} \sum_{\mathbf{x} \in \mathbf{X}} \left[\mathbf{w}^T \mathbf{x} \mathbf{x}^T \mathbf{w} + 2(b - f^*(\mathbf{x}))(\mathbf{x}^T \mathbf{w}) + (b - f^*(\mathbf{x}))^2 \right] . \end{aligned}$$

- Seek $\tilde{\boldsymbol{\theta}}$ such that

$$J(\tilde{\boldsymbol{\theta}}) = \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- The gradient $\nabla_{\boldsymbol{\theta}} J(\tilde{\boldsymbol{\theta}}) = 0$ leads to
- The normal equation:

$$\frac{1}{2} \sum_{\mathbf{x} \in \mathbf{X}} \begin{pmatrix} (\mathbf{x}\mathbf{x}^T)\mathbf{w} + (b - f^*(\mathbf{x}))\mathbf{x} \\ \mathbf{x}^T \mathbf{w} + b - f^*(\mathbf{x}) \end{pmatrix} = \mathbf{0}, \tag{1.1}$$

which implies

$$\begin{pmatrix} 2 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 2 & 4 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ b \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} . \tag{1.2}$$

$$\begin{pmatrix} 2w_1 + w_2 + 2b - 1 \\ w_1 + 2w_2 + 2b - 1 \\ 2(w_1 + w_2) + 4b - 2 \end{pmatrix} = \mathbf{0} \tag{1.3}$$

Exercise 1.1. *Derive* (1.1) *and* (1.2)*.*

- The solution is given by $\mathbf{w} = 0$ and $b = \frac{1}{2}$.
- The obtained function $f(\mathbf{x}; \mathbf{w}, b)$ does not classify given data set $\mathbf{X}$ properly since $f(\mathbf{x}) = \frac{1}{2}$ for all $\mathbf{x} \in \mathbf{X}$.
- The nonlinear function XOR is not represented by the linear map $f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^T\mathbf{x} + b$.

Python Program 1.1: XOR

```
#
#Written by Dongwoo Sheen <sheen@snu.ac.kr>
#http://www.nasc.snu.ac.kr
#
from sympy import *

#x1,x2,w1,w2,b = symbols('x1,x2,w1,w2,b)
w1,w2,b = symbols('w1 w2 b')
x1,x2 = symbols('x1 x2')

def f(x1,x2):
    f=w1*x1+w2*x2+b
    return f
#fstar(0,0)=0; fstar(1,1)=0; fstar(0,1)=1; fstar(1,0)=1;

#J = mse(w1,w2,b)

def J(w1,w2,b):
    J=1/4*( (-f(0,0))**2 + (-f(1,1))**2 +(1-f(0,1))**2 + (1-f(1,0))**2)
    return J

print("J=J(w1,w2,b)= ", J(w1,w2,b))
J_w1=diff(J(w1,w2,b),w1); J_w2=diff(J(w1,w2,b),w2); J_b=diff(J(w1,w2,b),b)
print("J_w1=", J_w1)
print("J_w2=", J_w2)
print("J_w3=", J_b)

theta=solve([J_w1, J_w2, J_b], [w1, w2, b])

print("theta=",theta)
```

1.1.2 DFFN with one hidden layer nonlinear model approach

- Define

$$\begin{aligned} \mathbf{h} &= f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c}) \leftarrow \text{hidden layer} && (1.4a) \\ y &= f^{(2)}(\mathbf{h}; \mathbf{w}, b), \leftarrow \text{output layer} && (1.4b) \\ f(\mathbf{x}; \mathbf{W}, c, \mathbf{w}, b) &= (f^{(2)} \circ f^{(1)})(\mathbf{x}) \leftarrow \text{two layer DFFN} && (1.4c) \end{aligned}$$

- Try with $\mathbf{h} = g(\mathbf{W}^T\mathbf{x} + \mathbf{c})$

- For the Activation function g, choose a suitable nonlinear transformation
  - ReLU function:
    $$\mathbf{g}(\mathbf{z}) = \max\{0, \mathbf{z}\} = \mathbf{z}^+.$$
  - The step function
  - The logistic sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$
  - tanh function $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
  - ReLU function is used in the hidden layer while the sigmoid function is used for the output layer.
  - Drawback: for some example with zero activation, ReLU cannot learn via gradient-based methods
  - Some generalized ReLU functions
    - Absolute rectification: with $\alpha_j = -1$,
      $$\mathbf{g}(\mathbf{z}, \boldsymbol{\alpha})_j = \max\{0, z_j\} - \min\{0, z_j\} = z_j^+ - z_j^- = |z_j|;$$
    - Leaky ReLU: with small fixed α_j such as $\alpha_j = 0.01$,
      $$\mathbf{g}(\mathbf{z}, \boldsymbol{\alpha})_j = \max\{0, z_j\} + \alpha_j \min\{0, z_j\};$$
    - Parametric ReLU: α_j as a learnable parameter,
      $$\mathbf{g}(\mathbf{z}, \boldsymbol{\alpha})_j = \max\{0, z_j\} + \alpha_j \min\{0, z_j\};$$
    - Maxout units: Divide $\mathbf{z}$ into a group of k values, with $G_j = \{(j-1)k+1, \cdots, jk\}$, then
      $$g(\mathbf{z})_j = \max_{\ell \in G_j} z_\ell.$$
    - For more generalize ReLU functions see
      `https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e`

Python Program 1.2: activation functions

```
# The activation functions and its composition with a function
#
#Written by Dongwoo Sheen <sheen@snu.ac.kr>
#http://www.nasc.snu.ac.kr
#****************************************
import numpy as np #import numpy module
import matplotlib.pyplot as plt

def g(x):     return   x

def trim(t,low,upper):
    if t < low:
        return low
    elif t > upper:
        return upper
    else:
        return t

def relu(t): return max(0,t)
def leaky_relu(t): return max(0,t) + 0.01*min(0,t)
def para_relu(t,alpha): return max(0,t) + alpha*min(0,t)
def abs_rect(t): return abs(t)
def sigmoid(t): return 1/(1+np.exp(-t))
def tanh(t): return (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))

x = np.arange(-2.,2,0.02) #make an array between -2 and 2
# with 0.2 subinterval length for plotting

plt.legend('upper left')
plt.title('activation functions')
plt.xlabel('x'); plt.ylabel('y')

def f_trim(t): return trim(g(t),-15.,15)
for t in x:
    y_trim = f_trim(t)
#     print(t,y_trim)
    plt.plot(t,y_trim,'r+',label="Trim")

def f_relu(t): return relu(g(t))
for t in x:
    y = f_relu(t)
    plt.plot(t,y,'g+',label="ReLU")

def f_sig(t): return sigmoid(g(t))
for t in x:
    y = f_sig(t)
    plt.plot(t,y,'b+',label="Sigm")

    plt.savefig("activation.eps")
    plt.show()

def g(x):     return    (5*x**5 -3*x**3 + x)/(4*x**4 + 2*x**2+1)
def f_trim(t): return trim(g(t),-1.,1.5)
for t in x:
    y_trim = f_trim(t)
#     print(t,y_trim)
    plt.plot(t,y_trim,'r+',label="Trim")

def f_relu(t): return relu(g(t))
for t in x:
    y = f_relu(t)
    plt.plot(t,y,'g+',label="ReLU")

def f_sig(t): return sigmoid(g(t))
for t in x:
    y = f_sig(t)
    plt.plot(t,y,'b+',label="Sigm")

plt.savefig("activated.eps")
plt.show()
```

- The complete network is given as follows:

$$f(\mathbf{x};\mathbf{W},\mathbf{c},\mathbf{w},b) = \mathbf{w}^T \max\left\{0, \mathbf{W}^T\mathbf{x}+\mathbf{c}\right\} + b$$

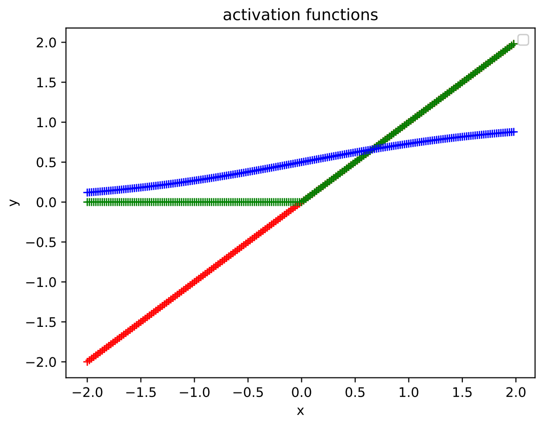

Figure 1.1.1: Some activation functions

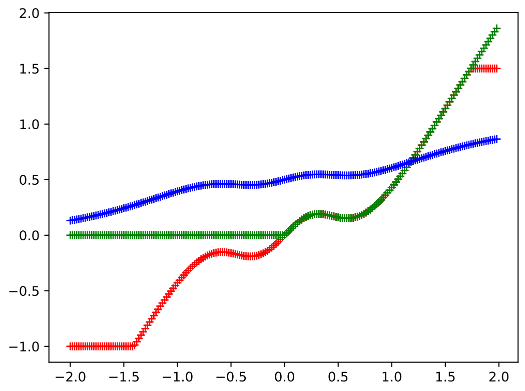

Figure 1.1.2: An example of nonlinear function with activation functions

- Our MLP for the XOR is the minimization problem given by

$$\boldsymbol{\theta}^* = (\mathbf{W}^*, \mathbf{c}^*, \mathbf{w}^*, b^*) \quad = \quad \operatorname{argmin} \sum_{j=1}^{4} |f(\mathbf{x}_j; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) - f^*(\mathbf{x}_j)|^2 \tag{1.5}$$

- $J(\boldsymbol{\theta}) = (\alpha_1 \max(c_1, 0) + \alpha_2 \max(c_2, 0) + b)^2 + (\alpha_1 \max(w_{11} + c_1, 0) + \alpha_2 \max(w_{12} + c_2, 0) + b - 1)^2 + (\alpha_1 \max(w_{21} + c_1, 0) + \alpha_2 \max(w_{22} + c_2, 0) + b - 1)^2 + (\alpha_1 \max(w_{11} + w_{21} + c_1, 0) + \alpha_2 \max(w_{12} + w_{22} + c_2, 0) + b)^2$, where $\mathbf{w} = (\alpha_1, \alpha_2)^T$.

- The solution of MLP is given by

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b = 0 \tag{1.6}$$

- To verify the solution, one sees that

$$\mathbf{W}^T\mathbf{x}+\mathbf{c} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} x_1+x_2 \\ x_1+x_2-1 \end{bmatrix}$$

- $f(\mathbf{x}) = \mathbf{w}^T \max\{0, \mathbf{W}^T\mathbf{x}+\mathbf{c}\} + b = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \cdot \max\left\{\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} x_1+x_2 \\ x_1+x_2-1 \end{bmatrix}\right\} = \max\{0, x_1+x_2\} - 2\max\{0, x_1+x_2-1\}$.
- It is easy to see that $f(\mathbf{x}_j) = f^*(\mathbf{x}_j), j = 1, \cdots, 4$. Thus the MLP learning gives a successful f.

Python Program 1.3: xor-mlp

```
#
#Written by Dongwoo Sheen <dongwoosheen@gmail.com>
#http://www.nasc.snu.ac.kr
#

from sympy import *
import numpy as np
import sympy as sp
from autograd import grad
from scipy.optimize import fsolve

#x1,x2,w1,w2,b = symbols('x1,x2,w1,w2,b)
w1,w2,b = symbols('w1 w2 b')
W = MatrixSymbol('W',2,2);   c = MatrixSymbol('c',2,1);
x = MatrixSymbol('x',2,1)
y = MatrixSymbol('y',2,1)
z = MatrixSymbol('z',9,1)

def relu(t): return Max(0,t)
def leaky_relu(t): return Max(0,t) + 0.01*Min(0,t)
def para_relu(t,alpha): return Max(0,t) + alpha*Min(0,t)
def abs_rect(t): return abs(t)
def sigmoid(t): return 1/(1+exp(-t))

def f(x1,x2):
    x = Matrix(2,1,[x1,x2])
    y = W.T*x+c
    y1 = relu(y[0,0]);       y2 = relu(y[1,0]);
#    y1 = Max(0,y[0,0]) # Max(0,y[0]) does not work
#    y2 = Max(0,y[1,0])
    f=w1*y1+w2*y2+b
    return f
#fstar(0,0)=0; fstar(1,1)=0; fstar(0,1)=1; fstar(1,0)=1;

#J = mse(w1,w2,b)

def J(W,c,w1,w2,b):
    J= (-f(0,0))**2 + (-f(1,1))**2 +(1-f(0,1))**2 + (1-f(1,0))**2
    return J
print("J=J(W,c,w1,w2,b)= ", J(W,c,w1,w2,b))
print("Remember that our J is 4*RMSE!")
```

```
heavi_modules = [{'Heaviside': lambda x: np.heaviside(x, 1)}, 'numpy']
def nonlinsys(theta):

#     W=Matrix(2,2,z[0:4]); c=Matrix(2,1,z[4:6])
#     W[0,0]=z[0]; W[0,1]=z[1]; W[1,0]=z[2]; W[1,1]=z[3];
#      c[0,0]=z[4]; c[0,0]=z[5];
#      w1=z[6];       w2=z[7];       b=z[8];
     F = np.empty((9))
#      ftn = (-b - w1*Max(0, W[0, 0] + W[1, 0] + c[0, 0])
# - w2*Max(0, W[0, 1] + W[1, 1] + c[1, 0]))**2
# + (-b - w1*Max(0, c[0, 0]) - w2*Max(0, c[1, 0]))**2
# + (-b - w1*Max(0, W[0, 0] + c[0, 0]) - w2*Max(0, W[0, 1]
# + c[1, 0]) + 1)**2 + (-b - w1*Max(0, W[1, 0] + c[0, 0])
# - w2*Max(0, W[1, 1] + c[1, 0]) + 1)**2
     def ftn(z): return (-z[8] - z[6]*Max(0, z[0] + z[2] + z[4])\
                              - z[7]*Max(0, z[1] + z[3] + z[5]))**2\
     + (-z[8] - z[6]*Max(0, z[4]) - z[7]*Max(0, z[5]))**2\
     + (-z[8] - z[6]*Max(0, z[0] + z[4])\
              - z[7]*Max(0, z[1] + z[5]) + 1)**2\
     + (-z[8] - z[6]*Max(0, z[2] + z[4])\
              - z[7]*Max(0, z[3] + z[5]) + 1)**2

     z = sp.IndexedBase('z')
     for i in range(9):
         func_val =  sp.diff(ftn(z),z[i])
         func = lambdify(z, func_val,modules=heavi_modules)
#          print("val=", func_val)
         F[i] = func(theta)

     return F

#theta_init = np.array([.5,.5,.5,.5,.5,.5,.5,.5,.5]) #initial guess
theta_init = np.array([.7,.7,.7,.7,.7,.7,.7,.7,.0]) #initial guess
theta_init = np.array([.9,.9,.9,.9,.09,-.9,.9,.9,.0]) #initial guess
theta_init = np.array([.8,.8,.8,.8,.09,-.8,.8,.8,.0]) #initial guess

theta=fsolve(nonlinsys, theta_init)

print("theta=",theta)

#theta = ([1, 1, 1, 1, 0, -1, 1, -2, 0]) #exact parameters

W=Matrix(2,2,theta[0:4]); c=Matrix(2,1,theta[4:6])
w1=theta[6];       w2=theta[7];       b=theta[8];
print("With these theta values J(W,c,w1,w2,b)=", J(W,c,w1,w2,b))

#With theta_init = np.array([.9,.9,.9,.9,.09,-.9,.9,.9,.0]) #initial guess
#theta= [ 2.03367777 -2.25684687 -1.96313146  2.193597    1.13317042  0.64571146
#  0.76133763  0.84916477 -1.4110407 ]
#With these theta values J(W,c,w1,w2,b)= 2.54208748405895e-23

#theta_init = np.array([.8,.8,.8,.8,.09,-.8,.8,.8,.0]) #initial guess
#theta= [ 1.42908227 -1.76372366 -2.81423044  3.08019683  2.07375326  0.24896974
#  0.85680641  0.90150248 -2.00125192]
#With these theta values J(W,c,w1,w2,b)= 2.34421549298756e-21
```

Interpretation with the first solution

- Let $\mathbf{X}$ be the design matrix containing all $\mathbf{x}_j, j = 1, \cdots, 4$.

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1\, \mathbf{x}_2\, \mathbf{x}_3\, \mathbf{x}_4 \end{bmatrix}.$$

- Recalling (1.6), we have

$$\mathbf{W}^T\mathbf{X} + \mathbf{c} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}.$$

- Observe that all data lie along the line $y = x - 1$.
- Recall that $f^*(x_1) = 0, f^*(x_2) = 1, f^*(x_3) = 1, f^*(x_4) = 0$, which means that the negative value does not appear. To cure this, apply the ReLU for each component to get

$$\mathrm{ReLU}\left(\mathbf{W}^T\mathbf{X} + \mathbf{c}\right) = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.7}$$

- Notice that $\left(\binom{0}{0}\right), \left(\binom{1}{0}\right), \left(\binom{1}{0}\right), \left(\binom{2}{1}\right)$ are not colinear, but we can apply linear regression to fit these data, which is the output layer operation!
- Recalling that the weight vector $\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ and $b = 0$, we have

$$\mathbf{w}^T \,\mathrm{ReLU}\left(\mathbf{W}^T\mathbf{X} + \mathbf{c}\right) + b = \begin{bmatrix} 1 & -2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}, \tag{1.8}$$

which is a correct classification of given set $\mathbf{X}$.

Interpretation with the 2nd solution

- $$\mathbf{W} = \begin{bmatrix} 2.03367777 & -2.25684687 \\ -1.96313146 & 2.193597 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 1.13317042 \\ 0.64571146 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} 0.76133763 \\ 0.84916477 \end{bmatrix}, \quad b = -1.4110407.$$
- Let $\mathbf{X}$ be the design matrix containing all $\mathbf{x}_j, j = 1, \cdots, 4$.

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1\, \mathbf{x}_2\, \mathbf{x}_3\, \mathbf{x}_4 \end{bmatrix}.$$

- Recalling (1.6), we have

$$\begin{aligned} \mathbf{W}^T\mathbf{X} + \mathbf{c} &= \begin{bmatrix} 2.03367777 & -1.96313146 \\ -2.25684687 & 2.193597 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1.13317042 \\ 0.64571146 \end{bmatrix} \\ &= \begin{bmatrix} 1.13317042 & -0.82996104 & 3.16684819 & 1.20371673 \\ 0.64571146 & 2.83930846 & -1.61113541 & 0.582461590 \end{bmatrix}. \end{aligned}$$

- Now apply the ReLU for each component to get

$$\mathrm{ReLU}\left(\mathbf{W}^T\mathbf{X}+\mathbf{c}\right) = \begin{bmatrix} 1.13317042 & 0 & 3.16684819 & 1.20371673 \\ 0.64571146 & 2.83930846 & 0 & 0.582461590 \end{bmatrix} \tag{1.9}$$

- Due to the output layer function

$$\begin{aligned} \mathbf{w}^T\,\mathrm{ReLU}\left(\mathbf{W}^T\mathbf{X}+\mathbf{c}\right)+b &= \begin{bmatrix} 0.76133763 & 0.84916477 \end{bmatrix} \\ &\quad \begin{bmatrix} 1.13317042 & 0 & 3.16684819 & 1.20371673 \\ 0.64571146 & 2.83930846 & 0 & 0.582461590 \end{bmatrix} - 1.4110407 \\ &= \begin{bmatrix} 5.366E-9 & 1.000 & 1.000 & 4.516E-9 \end{bmatrix} \end{aligned}$$

which is a correct classification of given set $\mathbf{X}$.

Chapter 2

Basic Linear Algebra

Notations and some useful formula.

- $A_{j,:}$ the j^{th} row (vector) of A.
- $A_{:,k}$ the k^{th} column (vector) of A.
- $\mathbf{e}^{(j)}$: j^{th} standard unit vector
- $Ax = \sum_{k=1}^{n} A_{:,k}x_k = \sum_{k=1}^{n} x_k A_{:,k}$
- $Ax = (A_{1,:}^T x, A_{2,:}^T x, \cdots, A_{m,:}^T x)^T$

2.1 Vector norms and matrix norms

2.1.1 Vector norms

Definition 2.1. *Let $F = \mathbb{R}$ or $F = \mathbb{C}$. A norm $\|\cdot\|$ on a vector space X over F is a nonnegative real-valued function, $\|\cdot\| : X \to \mathbb{R}^+$ is such that*

1. $\|x\| \geq 0$ *for* $\forall x \in X$,
2. $\|\alpha x\| = |\alpha|\|x\|$ *for* $\forall \alpha \in F, \qquad \forall x \in X$,
3. $\|x + y\| \leq \|x\| + \|y\|$ *for* $\forall x, y \in X$,
4. $\|x\| = 0$ *iff* $x = 0$.

If 1, 2, 3 are satisfied, $\|\cdot\|$ is called a seminorm.

We will implicitly assume that all the vector spaces are over the field $F = \mathbb{R}$ or $F = \mathbb{C}$.

Definition 2.2 (Hölder norm or p-norm on $\mathbb{R}^n$ or $\mathbb{C}^n$)**.** [*]

$$\begin{aligned} \|x\|_p &:= \left(\sum_{j=1}^{n} |x_j|^p\right)^{\frac{1}{p}} \qquad 1 \leq p < \infty, \\ \|x\|_\infty &:= \max_{1\leq j\leq n} |x_j|. \end{aligned}$$

The Hölder inequality ($p \geq 1$, $q \geq 1$, $\frac{1}{p} + \frac{1}{q} = 1$) [†]

$$|x^* y| = |\overline{x}^T y| \left(= \sum_{j=1}^{n} \bar{x}_j y_j \right) \leq \|x\|_p \|y\|_q \qquad \forall x, y \in \mathbb{C}^n.$$

[*] $\|x\|_\infty$ is called ∞-*norm* or *sup-norm*

[†] if $p = q = 2$, it is called *the Cauchy-Schwarz inequality*

Remark 2.1. *Unit circles in $\mathbb{R}^2$ with respect to p-norm*

$$\{x \in \mathbb{R}^2; \quad \|x\|_p = 1\}.$$

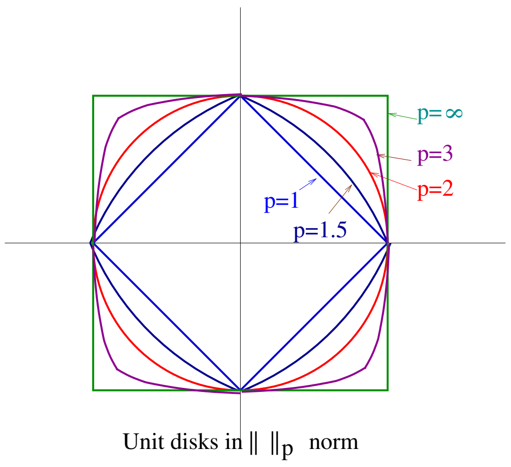

Figure 2.1.1: Unit disks in $\|\cdot\|_p$ norm in $\mathbb{R}^2$ for $1 \le p \le \infty$.

Python Program 2.1: mat-norm

```
#
#Written by Dongwoo Sheen <sheen@snu.ac.kr>
#http://www.nasc.snu.ac.kr
#
import numpy as np
from numpy import linalg as la
A = np.matrix([[2,-1,0],[-1,2,-1],[0,-1,2]])
print("A = ", A)

print("Try to see various norms of A")
normA=la.norm(A)
print("Frobenius-norm A = ", normA)
print("")
F_normA=la.norm(A,ord='fro')
print("Frobenius-norm A = ", F_normA)
print("")
l2_normA=la.norm(A,ord=2)
print("l2-norm A = ", l2_normA)
print("")
l1_normA=la.norm(A,ord=1)
print("l_1-norm A = ", l1_normA)
print("")
linf_normA=la.norm(A,ord=np.inf)
print("l_inf-norm A = ", linf_normA)
```

Definition 2.3. *A linear transformation (and a square matrix) Q is called unitary (or orthogonal if A is a transformation in an n-dimensional real vector space) if*

$$\bar{Q}^T Q = Q^* Q = I = QQ^* = Q\bar{Q}^T.$$

Remark 2.1. *If Q is unitary, then*

$$\|Qx\|_2 = \|x\|_2 \qquad \forall x \in X.$$

Indeed,

$$\|Qx\|_2{}^2 = (Qx)^*(Qx) = x^*Q^*Qx = x^*Ix = \|x\|_2.$$

Remark 2.2. *Let Q is an orthogonal linear transformation on $\mathbb{R}^n$ or an $n \times n$ orthogonal matrix. Then for any two nonzero vectors x and $y \in \mathbb{R}^n$, the angle θ between them is given by*

$$\theta = \cos^{-1}\left(\frac{x \cdot y}{\|x\|_2\|y\|_2}\right).$$

In the meanwhile, we have

$$(Qx) \cdot (Qy) = (Qy)^T Qx = y^T Q^T Qx = y^T x = x \cdot y,$$

and hence the angle between Qx and Qy is identical to θ since

$$\frac{(Qx) \cdot (Qy)}{\|Qx\|_2\|Qy\|_2} = \frac{x \cdot y}{\|x\|_2\|y\|_2}.$$

Hence the orthogonal matrix Q preserves the angles and lengths of two vectors.

If A is a real, $n \times n$ matrix, A is called orthogonal if

$$A^T A = AA^T = I. \tag{2.1}$$

If A is a complex, $n \times n$ matrix, A is called unitary if

$$A^* A = AA^* = I. \tag{2.2}$$

Thus if A is a complex, $n \times n$ matrix and A satisfies only (2.1) instead of (2.2), it is called complex orthogonal.

Remark 2.2. *Notice that a complex orthogonal matrix is not unitary. Recall that the complex matrices are applied to a complex vector. Let us try to have an example for this case. Consider the 2×2 matrix*

$$M(t) = \begin{pmatrix} \cosh(t) & i\sinh(t) \\ -i\sinh(t) & \cosh(t) \end{pmatrix}.$$

Then show that $M(t)$ is orthogonal, but not unitary. Then can you say that the matrix $M(t)$ has eigenvalues bounded?

Definition 2.4. *Two norms $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ on a vector space are called equivalent if there exist positive constants c_1 and c_2 such that*

$$c_1\|x\|_\beta \le \|x\|_\alpha \le c_2\|x\|_\beta \qquad \forall x \in X.$$

Proposition 2.1. *Let A_1 be an $(n-1) \times (n-1)$ principal submatrix of an $n \times n$ matrix A. Then,*

$$\|A_1\|_p \le \|A\|_p \text{ for any } p\text{–matrix norm } \|\cdot\|_p.$$

Proposition 2.2. *For $\forall x \in \mathbb{R}^n$, the following hold.*

$$\begin{aligned} \|x\|_2 &\le \|x\|_1 \le \sqrt{n}\|x\|_2, \\ \|x\|_\infty &\le \|x\|_2 \le \sqrt{n}\|x\|_\infty, \\ \|x\|_\infty &\le \|x\|_1 \le \ n\|x\|_\infty. \end{aligned}$$

When do the equalities hold?

Theorem 2.1. *Any two norms on $\mathbb{C}^n$ are equivalent. Indeed, any two norms on a finite-dimensional vector space are equivalent.*

2.1.2 Matrix norm

Let $\mathcal{M}(m,n)$ be the set of all $m \times n$ matrices over $\mathbb{C}$(or $\mathbb{R}$).

Definition 2.5. *A matrix norm $\|\cdot\| : \mathcal{M}(m,n) \longrightarrow \mathbb{R}^+$ is a nonnegative real-valued function such that*

1. $\|A\| \geq 0, \qquad \forall A \in \mathcal{M}(m,n)$
2. $\|\alpha A\| = |\alpha|\|A\|, \qquad \forall \alpha \in \mathbb{C}, \quad \forall A \in \mathcal{M}(m,n)$
3. $\|A+B\| \leq \|A\| + \|B\|, \qquad \forall A, B \in \mathcal{M}(m,n)$
4. $\|A\| = 0$ *if and only if* $A = 0$.

Example 2.1. *Let $\|A\|_\Delta$ be defined by*

$$\|A\|_\Delta = \max_{j,k} |A_{jk}|.$$

Then $\|\cdot\|_\Delta$ defines a norm.

But if $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $A^2 = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$, then $\|A^2\|_\Delta = 2, \quad \|A\|_\Delta = 1.$

$$\|A^2\|_\Delta \not\leq \|A\|_\Delta \|A\|_\Delta.$$

Definition 2.6. *A matrix norm on $\mathcal{M}(n,n)$ is* submultiplicative *if*

$$\|AB\| \leq \|A\|\|B\|, \qquad \forall A, B \in \mathcal{M}(n,n).$$

Definition 2.7. *A matrix norm $\|\cdot\|$ on $\mathcal{M}(m,n)$ is said to be* consistent *with respect to the vector norms $\|\cdot\|_\alpha$ on $\mathbb{C}^n$ and $\|\cdot\|_\beta$ on $\mathbb{C}^m$ if $\|Ax\|_\beta \leq \|A\|\|x\|_\alpha, \quad \forall x \in \mathbb{C}^n, \quad A \in \mathcal{M}(m,n)$.*

Definition 2.8. *A matrix norm $\|\cdot\|$* subordinate *to the vector norms $\|\cdot\|_\alpha$ on $\mathbb{C}^n$ and $\|\cdot\|_\beta$ on $\mathbb{C}^m$ is defined by*

$$\|A\|_{\alpha,\beta} = \sup_{x\neq 0, x\in\mathbb{C}^n} \frac{\|Ax\|_\beta}{\|x\|_\alpha} = \max_{\|x\|_\alpha = 1, x \in \mathbb{C}^n} \|Ax\|_\beta.$$

Note that a subordinate norm is consistent with respect to the associated norms.

Definition 2.9 (Matrix norms)**.** *Let A be an $m \times n$ matrix.*

1. *The Frobenius norm (F-norm, or Schur norm)*

$$\|A\|_F = \left(\sum_{j=1}^m \sum_{k=1}^n |A_{jk}|^2\right)^{\frac{1}{2}} = \left(\sum_{j=1}^m \|A_{j,:}\|_2^2\right)^{\frac{1}{2}} = \left(\sum_{k=1}^n \|A_{:,k}\|_2^2\right)^{\frac{1}{2}}$$

2. *p-norms ($p \geq 1$)*

$$\|A\|_p = \sup_{x\in\mathbb{C}^n, x\neq 0} \frac{\|Ax\|_p}{\|x\|_p},$$

(subordinate to $\|\cdot\|_p$ norms).

Let A be an $m\times n$ matrix. Then, let us try to compute the p–norms of A. Denote by $A_{j,:}$ the jth row of A. Then, by definition and Hölder's inequality,

$$\begin{aligned}\|A\|_p &= \max_{\|x\|_p=1}\|Ax\|_p\\ &= \max_{\|x\|_p=1}\Big(\sum_{j=1}^m |A_{j,:}^T x|^p\Big)^{\frac1p}\\ &\le \max_{\|x\|_p=1}\left[\sum_{j=1}^m \|A_{j,:}\|_q^p\|x\|_p^p\right]^{\frac1p} \qquad (2.3)\\ &= \left[\sum_{j=1}^m \|A_{j,:}\|_q^p\right]^{\frac1p},\end{aligned}$$

where $\frac1p+\frac1q=1$. The equality holds if and only if A is a rank 1 matrix. That is, all the row vectors $A_{j,:}$ of A forms a vector space of one dimension: $\mathrm{Span}\{A_{j_0,:}\}=\mathrm{Span}\{A_{1,:},A_{2,:},\cdots,A_{m,:}\}$ for some nonzero row vector $A_{j_0,:}$. In this case, by choosing x parallel to to $A_{j_0,:}$ we see that the equality holds in (2.3).

Theorem 2.2. *If A is an $m\times n$ matrix, then*

$$\max_{j,k}|A_{jk}|\le\|A\|_2\le\sqrt{mn}\max_{j,k}|A_{jk}|. \qquad (2.4)$$

Proof. Let $|A_{j_0k_0}|=\max_{j,k}|A_{jk}|$. Then, recalling $\|A\|_2=\max_{\|\mathbf{x}\|_2=1}\|A\mathbf{x}\|_2$, we have

$$|A_{j_0k_0}|\le\sqrt{\sum_{j=1}^m A_{jk_0}^2}=\|A\mathbf{e}^{(k_0)}\|_2\le\|A\|_2,$$

$\mathbf{e}^{(k_0)}$ being the k_0th standard unit vector in $\mathbb{R}^n$.

Conversely, let B be the $m\times n$ matrix such that $b_{jk}=1$ for all j,k. Certainly $\|A\|\le|A_{j_0k_0}|\|B\|$ for any matrix norm $\|\cdot\|$. By the Cauchy-Schwarz inequality, one has

$$\|B\|_2=\sup_{\mathbf{x}\neq\mathbf{0}}\frac{\|B\mathbf{x}\|_2}{\|\mathbf{x}\|_2}=\sup_{\mathbf{x}\neq\mathbf{0}}\frac{\sqrt{m(\sum_{k=1}^n x_k)^2}}{\|\mathbf{x}\|_2}\le\sup_{\mathbf{x}\neq\mathbf{0}}\frac{\sqrt{mn\sum_{k=1}^n x_k^2}}{\|\mathbf{x}\|_2}=\sqrt{mn}.$$

This proves the theorem. □

Proposition 2.3. *For the $\|\cdot\|_2$ matrix norm, we have the following identities:*

$$\begin{aligned}\|A\|_2 &= \sup_{\mathbf{x}\neq\mathbf{0}}\frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \qquad (2.5a)\\ &= \sup_{\mathbf{x}\neq\mathbf{0}}\sqrt{\frac{\mathbf{x}^TA^TA\mathbf{x}}{\mathbf{x}^T\mathbf{x}}} \qquad (2.5b)\\ &= \sup_{\mathbf{x}\neq\mathbf{0}}\sqrt{\rho(A^TA)} \qquad (2.5c)\\ &= \sigma_1, \qquad (2.5d)\end{aligned}$$

where $\rho(B)$ denotes the spectral radius of a square matrix B, and σ_1 is the largest singular value of A from the SVD (Singular Value Decomposition, see Theorem 2.7) of A.

Proposition 2.4. *Let $A=(A_{jk})$ be an $m\times n$ matrix over $\mathbb{C}$.*

$$\begin{aligned}\|A\|_1 &= \max_{k=1,\cdots,n}\sum_{j=1}^m|A_{jk}|, \qquad (2.6a)\\ \|A\|_\infty &= \max_{j=1,\cdots,m}\sum_{k=1}^n|A_{jk}|. \qquad (2.6b)\end{aligned}$$

That is, the 1*-norm is the maximum of* 1*-norms of all columns of the matrix* A*, while the* ∞*-norm is the maximum of* 1*-norms of all rows of the matrix* A*.*

Proposition 2.5. *Let* A *and* B *are an* $m \times n$ *matrix and an* $n \times k$ *matrix. Then we have that* $\|AB\|_p \le \|A\|_p \|B\|_p$ *for all* $1 \le p \le \infty$*.*

Proposition 2.6.

1. *The subordinate matrix norm is the smallest* consistent *norm with respect to the vector norms* $\|\cdot\|_\alpha$ *and* $\|\cdot\|_\beta$*.*
2. *Subordinate matrix norms are submultiplicative.*
3. *Let* $m \times m$ *matrix* Q *and* $n \times n$ *matrix* Z *be unitary transformations. Then for any* $m \times n$ *matrix* A*, we have*

$$\begin{aligned} \|QAZ\|_2 &= \|A\|_2, \\ \|QAZ\|_F &= \|A\|_F. \end{aligned}$$

Proposition 2.7. $\|A\|_2 \le \|A\|_F$.

2.1.3 Traces of matrices

Proposition 2.8. *The following trace formula holds:*

1. $\sqrt{\mathrm{Tr}(AA^T)} = \|A\|_F$,
2. $\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA) = \mathrm{Tr}(CAB)$, $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times \ell}, C \in \mathbb{R}^{\ell \times m}$,
3. $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$, $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times m}$.

Proof. Let $A \in \mathbb{R}^{m\times n}$ be arbitrary. Then

$$\mathrm{Tr}(AA^T) = (AA^T)_{jj} = A_{jk}(A^T)_{kj} = A_{jk}A_{jk} = \|A\|_F^2.$$

Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times m}$ be arbitrary. Then

$$\mathrm{Tr}(AB) = (AB)_{jj} = A_{jk}B_{kj} = B_{kj}A_{jk} = (BA)_{kk} = \mathrm{Tr}(BA).$$

□

2.1.4 Condition Number

$$\kappa(A) = \|A\|\,\|A^{-1}\|.$$

Thus the relative change in x is bounded by the relative change in b multiplied by $\kappa(A)$.

Theorem 2.3. *Let* $A \in \mathcal{M}(n,n)$ *be invertible and set* $B = A(I+F)$ *with* $\|F\| < 1$*. Consider* $Ax = b$*,* $B(x+\Delta x) = b$*. Then,*

$$\frac{\|\Delta x\|}{\|x\|} \le \frac{\|F\|}{1-\|F\|}.$$

Moreover,

$$\frac{\|\Delta x\|}{\|x\|} \le \frac{\kappa(A)\frac{\|B-A\|}{\|A\|}}{1-\kappa(A)\frac{\|B-A\|}{\|A\|}}.$$

Theorem 2.4. *Suppose that* $Ax = b$ *and* $(A+\Delta A)(x+\Delta x) = (b+\Delta b)$*. If* $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$*, then*

$$\frac{\|\Delta x\|}{\|x\|} \le \frac{\kappa(A)}{1-\kappa(A)\frac{\|\Delta A\|}{\|A\|}} \left\{ \frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right\}.$$

The following inequality is useful in estimation.

Lemma 2.1. *For any $\varepsilon > 0$ the Cauchy inequalities hold:*

$$xy \le \varepsilon x^2 + \frac{y^2}{4\varepsilon}, \quad \textit{and} \quad 2xy \le \varepsilon x^2 + \frac{y^2}{\varepsilon} \quad \forall x, y \in \mathbb{R}. \tag{2.7}$$

Theorem 2.5.

$$\|A\|_2 = \sup_{0 \ne x \in \mathbb{R}^n} \frac{x^T A x}{x^T x}$$

if A is a symmetric positive-definite matrix.

$$\|A^{-1}\|_2 = \frac{1}{\inf_{0 \ne x \in \mathbb{R}^n} \frac{x^T A x}{x^T x}}$$

if A is a symmetric positive-definite matrix.

2.2 Eigendecomposition

Suppose that A is an $n \times n$ matrix with n linearly independent eigenvectors.

Definition 2.10. *Let A be an $n \times n$ matrix over F, with $F = \mathbb{R}$ or $F = \mathbf{C}$. If there exists a pair $(\lambda, \boldsymbol{\xi}) \in \mathbf{C} \times \mathbf{C}^n$ such that*

$$A\boldsymbol{\xi} = \lambda \boldsymbol{\xi},$$

we call the pair is an eigenvalue and eigenvector of A.

Example 2.2. *Assume that $a, b \in \mathbb{R}, b \ne 0$. Let $A = \begin{bmatrix} a & -b \\ b & a \end{bmatrix}$. Then $\lambda = a \pm ib$ are eigenvalues with associated eigenvectors $\boldsymbol{\xi} = \begin{bmatrix} \pm i \\ 1 \end{bmatrix}$, respectively. That is,*

$$A \begin{bmatrix} \pm i \\ 1 \end{bmatrix} = (a \pm ib) \begin{bmatrix} \pm i \\ 1 \end{bmatrix}.$$

Remark 2.3. *In general, suppose that an $n \times n$ real matrix A has a complex eigenvalue $\lambda + i\mu$ with associated eigenvector $\mathbf{u} + i\mathbf{v}$. Then by taking the real and complex parts from*

$$A(\mathbf{u} + i\mathbf{v}) = (\lambda + i\mu)(\mathbf{u} + i\mathbf{v}),$$

the following relationship holds:

$$A \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \lambda & -\mu \\ \mu & \lambda \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}.$$

Definition 2.11. *Let*

$$p_A(\lambda) = \Pi_{j=1}^k (\lambda - \lambda_j)^{\mu_j}, \quad \sum_{j=1}^k \mu_j = n, \ \lambda_j \ne \lambda_\ell \text{ if } j \ne \ell.$$

be the characteristic equation for an $n \times n$ matrix A.

For each λ_j

$$E_{\lambda_j}(A) := \{v \in \mathbb{C}^n \mid Av = \lambda_j v\}$$

is called an "eigenspace" of A associated with λ_j, and $\nu_j := \dim(E_{\lambda_j}(A))$ is called the geometric multiplicity of λ_j for A, while μ_j is called the algebraic multiplicity of λ_j for A.

We have

Proposition 2.9. *For $j = 1, \cdots, k$, let μ_j and ν_j be the algebraic and geometric multiplicities of λ_j for an $n \times n$ matrix A. Then,*

$$1 \le \nu_j \le \mu_j, \quad j = 1, \cdots, k.$$

Theorem 2.6. *Let A be an $n \times n$ matrix over F. Suppose that the characteristic polynomial has the form*

$$p_A(\lambda) = \Pi_{j=1}^k (\lambda - \lambda_j)^{\mu_j}, \quad \sum_{j=1}^k \mu_j = n,$$

with distinct eigenvalues λ_j's. Then the following are equivalent:

(i) A is diagonalizable;

(i) the minimal polynomial is a product of linear factors

$$\mu_A(\lambda) = \Pi_{j=1}^k (\lambda - \lambda_j);$$

(ii) The algebraic multiplicities are equal to the geometric multiplicity:

$$\nu_j = \mu_j, \quad j = 1, \cdots, k.$$

Python Program 2.2: mat-eig

```
#
#Written by Dongwoo Sheen <sheen@snu.ac.kr>
#http://www.nasc.snu.ac.kr
#
import numpy as np
from numpy import linalg as la
A = np.matrix([[2,-1,0],[-1,2,-1],[0,-1,2]])
print("A = ", A)
print("Try to see the eigenvalues of A with la.eig(A)")
#Output: the eigenvalues, each repeated according to its multiplicity and
#the normalized eigenvectors, such that the column v[:,i] is the eigenvector corresponding to t

lam,V=la.eig(A)

print("lam[0]=",lam[0])
#lam_1=3.41421356
#V = np.matrix([[-5.00000000e-01, -7.07106781e-01,  5.00000000e-01],
#          [ 7.07106781e-01,  4.05925293e-16,  7.07106781e-01],
#          [-5.00000000e-01,  7.07106781e-01,  5.00000000e-01]])
#
Sig = np.diag(lam)
print("")
print("V=",V)
print("")
print("")
print("Sig=",Sig)
print("")
print("AV=",A*V)
print("")
print("V Sig=",V*Sig)
print("")
closeness=np.allclose(A*V,V*Sig)
print("A V-V Sig=0?",closeness)
```

2.3 Singular Value Decomposition(SVD)

Example 2.3. *Let* $A = \frac{1}{25}\begin{bmatrix} 24 & 43 \\ 57 & 24 \end{bmatrix}$, *and set* $E = \{y \in \mathbb{R}^2 : y = Ax, \|x\|_2 = 1\}$. *We will investigate in the image* E *of the unit sphere under the linear transformation* A. *Of course, since* A^{-1} *exists, one can write*

$$E = \{y \in \mathbb{R}^2 : \|A^{-1}y\|_2 = 1, \}$$

where $A^{-1} = \frac{1}{25}\begin{bmatrix} -8 & 43 \\ 19 & -8 \end{bmatrix}$. *Thus, we have* $(-8y_1 + 43y_2)^2 + (19y_1 - 8y_2)^2 = 25^2$, *or* $425y_1^2 - 496y_1y_2 + 1913y_2^2 = 25^2$ *which is an equation of ellipse. We now take a different approach for this problem. Notice that*

$$A = U\Sigma V^T = \frac{1}{25}\begin{bmatrix} 3 & -4 \\ 4 & 3 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 4 & 3 \\ 3 & -4 \end{bmatrix}^*. \tag{2.8}$$

where U *and* V *are orthogonal matrices. Notice that* U *is a rotation and* V *is a reflection. Thus, since* $\|V^Tx\|_2 = 1$, *we have*

$$\begin{aligned} E &= \{y \in \mathbb{R}^2 : y = U\Sigma V^T x, \|x\|_2 = 1, \} \\ &= \{y \in \mathbb{R}^2 : \Sigma^{-1}U^T y = V^T x, \|x\|_2 = 1, \} \\ &= \{y \in \mathbb{R}^2 : \|\Sigma^{-1}U^T y\|_2 = 1\} \\ &= \{y \in \mathbb{R}^2 : \|\Sigma^{-1}z\|_2 = 1, z = U^T y\} \\ &= \{y \in \mathbb{R}^2 : \|\Sigma^{-1}z\|_2 = 1, y = Uz\}. \end{aligned}$$

Thus, the equation $\frac{z_1^2}{3} + z_2^2 = 1$ *describes an ellipse, and then* E *is the rotation of it with the angle* $\theta = \operatorname{atan}(\frac{4}{3})$.

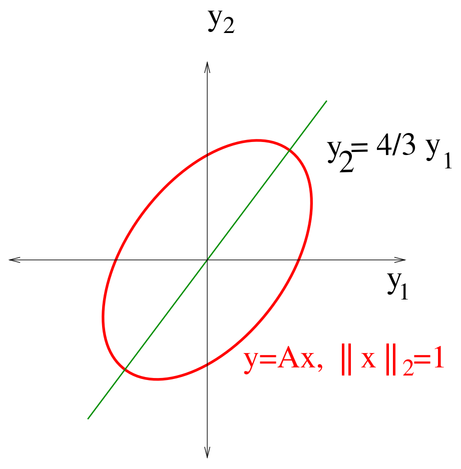

Figure 2.3.2: The ellipse E which is the image of the unit circle under the image of the matrix A

Here,

$$u_1 = \frac{1}{5}(3,4)^T, \quad u_2 = \frac{1}{5}(-4,3)^T.$$

In general in n*-dimensional vector space, a sphere is transformed by a nonsingular matrix* A *into an ellipsoid with singular values being the semi-axes of* E.

Theorem 2.7. *For $A \in \mathcal{M}(m,n)$, there exist unitary matrices $U = [u_1,\dots,u_m] \in \mathcal{M}(m,m)$, $V = [v_1,\dots,v_n] \in \mathcal{M}(n,n)$, and $\Sigma \in \mathcal{M}(m,n)$ such that*

$$\begin{aligned} U^*AV = \operatorname{diag}(\sigma_1,\dots,\sigma_p) =: \Sigma, \quad p = \min(m,n), \\ \text{where } \sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0. \end{aligned} \tag{2.9}$$

Proposition 2.10. $A = UU^*AVV^* = U\Sigma V^*$.

1. *From $AV = U\Sigma$, one has $Av_j = \sigma_j u_j, j = 1,\dots,p$.*

   If $U = V$, then this equation implies a eigenvalue problem. Hence it is called a generalized eigenvalue.

2. *From $U^*A = \Sigma V^*$, one has $A^*u_j = \sigma_j v_j, j = 1,\dots,p$.*

σ_j :	*jth singular value of A.*
u_j, v_j :	*jth left and right singular vectors of A.*

$$\sigma_1 = \|A\|_2 = \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2=1} \|Ax\|_2. \tag{2.10}$$

Corollary 2.1. *If $A = U\Sigma V^*$ is a SVD of A with $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0$, the following properties hold:*

1. $A = \sum_{j=1}^r \sigma_j u_j v_j^* = U_r\Sigma_r V_r^*$. *Hence $Av_k = \sigma_k u_k, k = 1,\cdots,r$.*
   $U_r = [u_1,\dots,u_r]$, $\quad V_r = [v_1,\dots,v_r]$, $\quad \Sigma_r = \operatorname{diag}\{\sigma_1,\dots,\sigma_r\}$.

2. $\operatorname{rank}(A) = r$.

3. $N(A) = \operatorname{Span}\{v_{r+1},\dots,v_n\}$.

4. $R(A) = \operatorname{Span}\{u_1,\dots,u_r\}$.

5. $\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$.

6. $\|A\|_2 = \sigma_1$.

7. $\sigma_j = \sqrt{\lambda_j(A^*A)}, \qquad j = 1,\dots,p,$
   where $\lambda_j(A^*A)$ is the jth largest eigenvalue of A^*A.

Proof Since $v = [v_1,\cdots v_n]$ is unitary and $\operatorname{span}\{v_1,\cdots,v_n\} = \mathbb{C}^n$, *we have $x = \sum \alpha_j v_j$, and therefore,*

$$\begin{aligned} \operatorname{Range}(A) &= \{Ax \mid x \in \mathbb{C}^n\} \\ &= \{\sum_{j=1}^r \sigma_j u_j v_j^* x \mid x \in \mathbb{C}^n\} \\ &= \{\sum_{j=1}^r \sigma_j u_j v_j^* (\sum_{k=1}^n \alpha_k v_k), \alpha_k \in \mathbb{C}\} \\ &= \{\sum_{j=1}^r \sigma_j \alpha_j u_j \mid \alpha_j \in \mathbb{C}\}. \end{aligned}$$

4.

$$\begin{aligned}
\mathrm{Null}(A) &= \{x \in \mathbb{C}^n \mid Ax = 0\} \\
&= \{x \in \mathbb{C}^n \mid \sum_{j=1}^{r} \sigma_j u_j v_j^* x = 0\} \\
&= \{x \in \mathbb{C}^n \mid \sum_{j=1}^{r} \sigma_j u_j v_j^* \sum_{k=r+1}^{n} \alpha_k v_k = 0\} \\
&= \{x \in \mathbb{C}^n \mid x = \sum_{k=r+1}^{n} \alpha_k v_k\}
\end{aligned}$$

where $x = \sum_{k=1}^{n} \alpha_k v_k$

5. $\|A\|_F = \|U_r \Sigma_r V_r^*\|_F = \|\Sigma_r\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$ *where* u, v^* *: unitary.*

note:: preserve F*-norm.*

Python Program 2.3: mat-svd

```
#
#Written by Dongwoo Sheen <sheen@snu.ac.kr>
#http://www.nasc.snu.ac.kr
#
import numpy as np
from numpy import linalg as la
A = np.matrix([[2,-1,0],[-1,2,-1],[0,-1,2]])
print("A = ", A)
print("Try to see the SVD of A with la.svd(A)")
#Output: U, the sigular values, V^T

U,sig,VT=la.svd(A)

Sig = np.diag(sig)
print("")
print("Sig=",Sig)
print("")
print("U=",U)
print("")
print("VT=",VT)
print("")
print("AV=",A*VT.T)
print("")
print("U Sig=",U*Sig)
print("")

closeness=np.allclose(A*VT.T,U*Sig)
print("A*V-U*Sig=0?",closeness)
```

Exercise 2.1. *Let* A *be an* $m \times n$ *matrix. Then show that the eigenvalues of* A^*A *are nonnegative with eigenvalues* $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$*. Also, denoting* $\lambda_j(A^*A) = \lambda_j, j = 1 \cdots, r$*, show that*

$$\sigma_j = \sqrt{\lambda_j(A^*A)}.$$

Exercise 2.2. *If A is an $m \times n$ matrix, find relationships between the eigenvalues and eigenvectors of*

$$B = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$$

and the singular values and singular vectors of A.

Exercise 2.3. *Show that if A is a positive-semidefinite matrix, then the singular values of A are the same as the eigenvalues of A.*

Exercise 2.4. *Prove that if A is a positive-definite matrix, and $A = U\Sigma V^*$ is a singular value decomposition of A, then $U = V$.*

Exercise 2.5. *Implement the SVD algorithm.*

Now we are ready to implement the SVD algorithm stated as in the proof of Theorem 2.7. We proceed as follows.

1. In order to compute the l^2-norm of a matrix A, recall that

$$\begin{aligned}\|A\|_2^2 &= \sup_{\|x\|_2=1} \|Ax\|_2^2 = \sup_{\|x\|_2=1} (Ax, Ax) = \sup_{\|x\|_2=1} (x, A^T Ax) \\ &= \sup_{\|\xi\|_2=1} (\xi, \lambda_{\max}(A^T A)\xi) = \lambda_{\max}(A^T A).\end{aligned}$$

   Hence, we identify the l^2-norm of A with $\sqrt{\lambda_{\max}(A^T A)}$.

2. $\lambda_{\max}(A^T A)$ can be approximated by the power iteration method.

Definition 2.12. *An $n \times n$ matrix P is called a projection matrix if $P^2 = P$. If, in addition, $P^* = P$, it is called an orthogonal projection matrix.*

2.4 Moore-Penrose pseudo inverse $A^\dagger$ of $A \in \mathcal{M}(m, n)$.

The SVD can be used to find a pseudo inverse of a matrix which is not of square size. The Moore-Penrose pseudo inverse $A^\dagger \in \mathcal{M}(n, m)$ of $A\mathcal{M}(m, n)$ is the unique solution $\chi \in \mathcal{M}(n, m)$ such that

1. $A\chi A = A$.
2. $\chi A\chi = \chi$
3. $(A\chi)^* = A\chi \quad \in \mathcal{M}(m, m)$
4. $(\chi A)^* = \chi A \quad \in \mathcal{M}(n, n)$

Example 2.4. *If $A = U\Sigma V^*$ is a SVD of A, then $A^\dagger = V\Sigma^\dagger U^*$ where*

$$\Sigma^\dagger = \begin{pmatrix} 1/\sigma_1 & & & & \\ & 1/\sigma_2 & & & \\ & & \ddots & & \\ & & & 1/\sigma_r & \\ & & & & 0 \end{pmatrix}. \tag{2.11}$$

Some useful and interesting properties of the pseudo-inverse are listed as follows.

Properties 2.1. *1. $A^{\dagger\dagger} = A$.*

2. $(A^\dagger)^* = (A^*)^\dagger$.

3. $\|Ax - b\|_2 \ge \left\|AA^\dagger b - b\right\|_2 \qquad x \in \mathbb{C}^n$.

4. $\|Ax-b\|_2 = \left\|AA^\dagger b - b\right\|_2$ *and* $x \neq A^\dagger b \quad \Longrightarrow \quad \|x\|_2 > \left\|A^\dagger b\right\|, x \neq x_{LS}$*. Thus* $A^\dagger b$ *is the solution of the least squares problem.*

Exercise 2.6. *Show that Properties 2.1 hold.*

Python Program 2.4: mat-pinv

```
#
#Written by Dongwoo Sheen <sheen@snu.ac.kr>
#http://www.nasc.snu.ac.kr
#
import numpy as np
from numpy import linalg as la
A = np.matrix([[2,-1,0],[-1,2,-1]])
print("A = ", A)
print("Try to see the pseudo-inverse of A with la.pinv(A)")
#Output: the pseudo-inverse of A

A_dag=la.pinv(A)
print("")
print("A^dag=",A_dag)
print("")
print("A*A^dag=",A*A_dag)
print("")
print("A^dag*A=",A_dag*A)
print("")
closeness=np.allclose(A*A_dag*A,A)
print("A*A^dag*A - A=0?",closeness)
print("")
closeness=np.allclose(A_dag*A*A_dag,A_dag)
print("A^dag*A*A^dag - A^dag=0?",closeness)
```

2.5 Least squares problem of matrices

Theorem 2.8. *Let* $A = U\Sigma V^*$ *be a SVD of* $A \in \mathcal{M}(m,n)$ *with* $r = \text{rank}\,(A)$*. Then for* $k < r$*,*

$$\min_{\text{rank}\,(B)=k, B\in\mathcal{M}(m,n)} \|A-B\|_2 = \left\| A - \sum_{j=1}^{k} \sigma_j u_j v_j^* \right\|_2 \tag{2.12}$$

To prove Theorem 2.8, we recall the following Dimension Theorem from Linear Algebra.

Theorem 2.9 (Dimension theorem)**.** *Let* B *be an* $m \times n$ *matrix. Then we have*

$$n = \text{rank}\,(B) + \text{nullity}(B) = \dim(R(B)) + \dim(N(B)). \tag{2.13}$$

In general, if $T : V \longrightarrow W$ *is a linear transformation from a finite dimensional vector space* V *into a vector space* W*, then we have*

$$n = \text{rank}\,(T) + \text{nullity}(T) = \dim(R(T)) + \dim(N(T)). \tag{2.14}$$

2.6 PCA (Principal Component Analysis)

- PCA is a machine learning algorithm which can be derived by using the linear algebra.
- Suppose that there are m data $S = \{x^{(1)}, \cdots, x^{(m)}\} \subset \mathbb{R}^n$.
- Lossy compression is to store these data in less memory in $\mathbb{R}^l, l \ll n$.
- Encoding $f : S \subset \mathbb{R}^n \to \mathbb{R}^l$ defined by $f(\mathbf{x}) = \mathbf{c}$. Let $C = f(S)$.

- Decoding $g : C \subset \mathbb{R}^l \to \mathbb{R}^n$ such that the reconstruction

$$r = (g \circ f) : S \to \mathbb{R}^n \approx I|_S. \tag{2.15}$$

- Target: try to find a suitable decoding matrix D which represents g as a linear transformation:

$$g(\mathbf{c}) = \mathbf{Dc}, \tag{2.16}$$

  where D is an $n \times l$ matrix whose columns are

  - orthogonal to each other;
  - of unit size.

Step 1 For the time being, let D be a fixed matrix.

(a) Then for each $\mathbf{x} \in S$, we have the least squares minimization problem:

$$c^* = \operatorname*{argmin}_{\mathbf{c}\in C} \|\mathbf{x} - g(\mathbf{c})\|_2^2 = \operatorname*{argmin}_{\mathbf{c}\in C} \|\mathbf{x} - D\mathbf{c}\|_2^2. \tag{2.17}$$

(b) The solution of the above LS minimization problem is obtained by differentiating $\|\mathbf{x} - D\mathbf{c}\|_2^2$ with respect to $\mathbf{c}$ and setting the result to be zero. Invoking that the columns of D are orthogonal, we have

$$\begin{aligned}
\nabla_{\mathbf{c}}\|\mathbf{x} - D\mathbf{c}\|_2^2 &= \nabla_{\mathbf{c}}(\mathbf{x} - D\mathbf{c})^T(\mathbf{x} - D\mathbf{c}) && \text{(2.18a)}\\
&= \nabla_{\mathbf{c}}\left(\mathbf{x}^T\mathbf{x} - \mathbf{x}^T D\mathbf{c} - (D\mathbf{c})^T\mathbf{x} + \mathbf{c}^T D^T D\mathbf{c}\right) && \text{(2.18b)}\\
&= \nabla_{\mathbf{c}}\left(-2\mathbf{x}^T D\mathbf{c} + \mathbf{c}^T\mathbf{c}\right) && \text{(2.18c)}\\
&= -2D^T\mathbf{x} + 2\mathbf{c} = 0, && \text{(2.18d)}
\end{aligned}$$

from which

$$\mathbf{c}^* = D^T\mathbf{x}. \tag{2.19}$$

(c) (2.19) suggests that the encoding map f should be chosen as the linear transformation

$$f(\mathbf{x}) = D^T\mathbf{x}, \tag{2.20}$$

from which the PCA reconstruction can be given as

$$r(\mathbf{x}) = DD^T\mathbf{x}. \tag{2.21}$$

Step 2 Next step is to choose a suitable encoding matrix D^* such that

$$D^* = \operatorname*{argmin}_{D\in\mathbb{R}^{n\times l}, D^T D = I_l} \sum_{j=1}^{m} \|\mathbf{x}^{(j)} - DD^T\mathbf{x}^{(j)}\|_2^2 \tag{2.22}$$

(a) Set $X = [\mathbf{x}^{(1)}\, \mathbf{x}^{(2)} \cdots \mathbf{x}^{(m)}] \in \mathbb{R}^{n\times m}$. Then (2.22) is equivalent to

$$D^* = \operatorname*{argmin}_{D\in\mathbb{R}^{n\times l}, D^T D = I_l} \|X - DD^TX\|_F^2 \tag{2.23}$$

(b) We have

$$\begin{aligned}
\|X - DD^TX\|_F^2 &= \operatorname{Tr}\left[(X^T - X^TDD^T)(X - DD^TX)\right]\\
&= \operatorname{Tr}(X^TX) - 2\operatorname{Tr}(X^TDD^TX) + \operatorname{Tr}(X^TDD^TDD^TX)\\
&= \operatorname{Tr}(X^TX) - \operatorname{Tr}(X^TDD^TX)\\
&= \operatorname{Tr}(X^TX) - \operatorname{Tr}(D^TXX^TD).
\end{aligned}$$

(c) Hence,

$$D^* = \underset{D\in\mathbb{R}^{n\times l}, D^TD=I_l}{\operatorname{argmin}} \|X - DD^TX\|_F^2 = \underset{D\in\mathbb{R}^{n\times l}, D^TD=I_l}{\operatorname{argmax}} \operatorname{Tr}(D^TXX^TD). \tag{2.24}$$

(d) Since XX^T is symmetric, it is diagonalizable and there exists an eigendecomposition of XX^T such that

$$XX^T = V\Sigma V^T,$$

where

$$\Sigma = \operatorname{diag}(\lambda_1, \lambda_2, \cdots, \lambda_l, \cdots, \lambda_n),\ \lambda_1 \geq \lambda_2 \geq \cdots \lambda_n \geq 0.$$

and the n column vectors $V_{:,1}, \cdots, V_{:,n}$ of V are the corresponding eigenvectors to λ_j's and V is orthogonal.

(e) We choose $D_{:,k} = V_{:,k}, k = 1, \cdots, l$. Then, it is easy to see that

$$\operatorname{Tr}(D^TXX^TD) = \sum_{k=1}^{l} \lambda_k.$$

Exercise 2.7. *Prove that the above choice of* **D** *solves the maximization problem:* (2.24).

Chapter 3

Numerical Computation

3.1 Overflow and underflow

- **Floating point representation** is commonly used to store real-valued number in the form of

$$\pm(mantissa) \times (base)^{(exponent)}.$$

- **Double precision**: uses 8 bytes (64 bits) to store numerical information. Using double precision we can store $2^{64} \simeq 10^{19}$ different informations.
- In double precision, 1 bit is used for $\pm$, 52 bits for mantissa, and 11 bits for exponent.
- There exists a smallest positive number($:= a$) and a largest number($:= b$) that can be expressed by floating point representation.
- **Underflow**: can occur in the case where positive numbers smaller than a is assumed to be 0. In many cases, underflow can be avoided by adequately modifying computation process.
- **Overflow** can occur in the case where numbers larger than b are assumed to be ∞.
- One way to avoid these errors is using known functions to crop very large data to be small enough.

3.2 Conditioning

Consider the problem: $Ax = b$ for x. (or finding $\text{argmin}_x ||Ax - b||_2$) A and b are stored as rounded data : $A + \Delta A$ and $b + \Delta b$. We can write the equation as

$$(A + \Delta A)(x + \Delta x) = b + \Delta b$$

where Δx is the error in solution.

We can estimate relative error in the solution with relative rounding errors using **condition numbers**. Assume that the matrix norm is subordinate to the vector norm.

Definition 3.1. *The condition number of A is defined as*

$$\kappa(A) := \|A\| \, \|A^{-1}\|.$$

1. For the change in the solution depending on that in the data

$$A(x + \Delta x) = b + \Delta b,$$

we have

$$\frac{\|\Delta x\|}{\|x\|} \le \frac{\|A\| \cdot \|A^{-1}\| \cdot \|\Delta b\|}{\|b\|} = \kappa(A)\frac{\|\Delta b\|}{\|b\|}.$$

2. For the change in the solution depending on those in the matrix and the data

$$(A + \Delta A)(x + \Delta x) = b + \Delta b,$$

we have if $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$, then

$$\frac{\|\Delta x\|}{\|x\|} \le \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\Delta A\|}{\|A\|}} (\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|}).$$

Problem happens when $\kappa(A)$ is very large. We call such A an **ill-conditioned matrix**. One way to avoid this situation is using an adequate **preconditioner matrix** M.
i.e.
We find an invertible matrix M with $\kappa(M^{-1}A) \ll \kappa(A)$, small enough such that we can ignore $\|\Delta x\|$. Then we can use $M^{-1}Ax = M^{-1}b$ instead of $Ax = b$ to get the same solution x.

By using the L^2-norm, the condition number of $A \in \mathbb{R}^n \times \mathbb{R}^n$ can be calculated as follows:

$$\kappa(A) = \|A\|_2 \, \|A^{-1}\|_2 = \rho(A)\rho(A^{-1}) = \max_{\lambda\in\sigma(A)} |\lambda| \frac{1}{\min_{\lambda\in\sigma(A)} |\lambda|} = \max_{i,j} \left|\frac{\lambda_i}{\lambda_j}\right|,$$

where $\sigma(A) = \{\lambda \in \mathbb{C} | Ax = \lambda x \text{ for some } x \neq 0\}$ is the spectrum of A, $\lambda_i \in \sigma(A)$,
$\rho(A) = \max\limits_{\lambda\in\sigma(A)} |\lambda|$ is the spectral radius of A. The last equality holds because for invertible A (full rank) and for $\lambda \neq 0$

$$Ax = \lambda x \iff \frac{1}{\lambda}x = A^{-1}x.$$

3.3 Gradient-based Optimization

3.4 Steepest descent method

Consider a given function $f : \mathbb{R}^n \to \mathbb{R}$ and find ξ such that $f(\xi) = \min_{x\in\mathbb{R}^n} f(x)$. Let $x^{(0)}$ be an initial approximation. Then, recursively, if $x^{(k)}$ is obtained, $x^{(k+1)}$ is sought in the direction of $-\nabla f(x^{(k)})$ with a step length $s^{(k)}$ such that

$$g(s^{(k)}) = \min_{s\in\mathbb{R}} g(s), \quad \text{where } g(s) := f\left(x^{(k)} - s\nabla f(x^{(k))}\right).$$

Remark 3.1. *The step length $s^{(k)}$ is called the learning rate*

Algorithm 4.1.

```
1 do k=0, max_it
2   compute $v^{(k)} = \nabla f(x^{(k)}) \in \mathbb{R}^n$
3   if $(|v^{(k)}| < \epsilon)$
4     stop
5   end if
6   find the step length $s^{(k)} > 0$, which minimizes $g(s) = f(x^{(k)} - sv^{(k)})$.
7   $x^{(k+1)} = x^{(k)} - s^{(k)}v^{(k)}$
8 enddo
```

By the chain rule, the local minimum of $g(s^{(k)})$ can be found by seeking the zero of

$$g'(s) = \nabla f\big(x^{(k)} - sv^{(k)}\big) \cdot (-v^{(k)}), \tag{3.1}$$

which can be obtained by the Newton method or any other root-finding scheme for the scalar function $\phi(s) := g'(s)$. For instance if the Newton method is applicable, one can iterate as follows:

$$\begin{aligned} s_{j+1} &= s_j - \phi(s_j)/\phi'(s_j) \\ &= s_j - g'(s_j)/g''(s_j), \end{aligned}$$

where $g'(s_j)$ is given by (3.1) and $g''(s_j)$ can be obtained by differentiating (3.1) once more:

$$\begin{aligned} g''(s_j) &= \nabla\left[\nabla f\big(x^{(k)} - s_j v^{(k)}\big)\cdot(-v^{(k)})\right]\cdot(-v^{(k)}) \\ &= [v^{(k)}]^T Hf\big(x^{(k)} - s_j v^{(k)}\big) v^{(k)}, \end{aligned}$$

where Hf denotes the Hessian matrix of f.

Remark 3.2. *The smallest-norm solution to the unconstrained least squares problem may be found using the Moore-Penrose pseudoinverse:* $x = \mathbf{A}^\dagger \mathbf{b}$.

3.5 Constrained optimization

Definition 3.2 (Constrained Optimization)**.** *Finding the maximum or minimum value of* $f(\mathbf{x})$ *for values of* $x \in S$.

Definition 3.3 (Feasible Points)**.** *A point* x *that lies in the set* S *are called* ***feasible points****.*

Karush-Kuhn-Tucker (KKT) Approach

Karush-Kuhn-Tucker (KKT) approach provides a very general solution to constrained optimization. With the KKT approach, we introduce a new function called the **generalized Lagrangian** or **generalized Lagrange function**.

The generalized Lagrangian is defined as

$$L(x, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(x) + \sum \lambda_i g^{(i)}(x) + \sum \alpha_j h^{(j)}(x). \tag{3.2}$$

Remark 3.3. *To define the Lagrangian, we first need to describe* S *in terms of equations and inequalities.* S *is described in terms of* m *function* $g^{(i)}$ *and* n *functions* $h^{(j)}$ *so that*

- $S = \{x \mid \forall i, g^{(i)}(x) = 0 \text{ and } \forall j, h^{(j)}(x) \le 0\}$
- $g^{(i)}$: *equality constraints*
- $h^{(j)}$: *inequality constraints*
- λ_i *and* α_j : *KKT multipliers .*

We can solve a constrained minimization problem using unconstrained optimization of the generalized Lagrangian.

$$\min_x \max_\lambda \max_{\alpha,\alpha\ge 0} L(x,\lambda,\alpha) \tag{3.3}$$

has the same optimal objective function value and set of optimal points x as the constrained min problem:

$$\min_{x\in S} f(x). \tag{3.4}$$

The reason is

$$\max_\lambda \max_{\alpha,\alpha\ge 0} L(x,\lambda,\alpha) = \begin{cases} f(x) & \text{if } x\in S \\ \infty & \text{if } x \notin S. \end{cases} \tag{3.5}$$

Definition 3.4 (Active Constraint)**.** $h^{(j)}(x^*)$ *is active if* $h^{(j)}(\mathbf{x}^*) = 0$ *If a constraint is not active, then the solution to the problem found using that constraint would remain at least a local solution if that constraint were removed.*

Remark 3.4. *The sign of the term for the equality constraints does not matter.*

Karush-Kuhn-Tucker (KKT) Conditions

1. The gradient of the generalized Lagrangian is zero.
2. All constraints on both x and the KKT multipliers are satisfied.
3. The inequality constraints exhibit "the complementary slackness": $\boldsymbol{\alpha} \cdot \mathbf{h}(x) = 0$

Remark 3.5. *They are necessary conditions, but not always sufficient conditions, for a point to be optimal.*

3.6 Example: Linear Least Squares

Suppose we want to find the value of x that minimizes

$$f(x) = \frac{1}{2}\|\mathbf{Ax} - \mathbf{b}\|_2^2$$

⇒ Using gradient-based optimization

1. Obtain the gradient:
$$\nabla_x f(x) = \mathbf{A}^T(\mathbf{Ax} - \mathbf{b}) = \mathbf{A}^T\mathbf{A}x - \mathbf{A}^T\mathbf{b}$$
2. Follow this gradient downhill, taking small steps.

3.6.1 (Unconstrained) Least squares problems

Given discrete data $(x_j, y_j), j = 1, \cdots n$ consider the problem of representing the data as a linear combination of reasonable functions $\phi_j(x), j = 1, \cdots, m$. For instance, Here, x_j's are not necessarily distinct; if they were distinct, then the usual interpolation can be a good representative.

Example 3.1. *Given discrete data $(x_j, y_j), j = 1, \cdots n$ consider the problem of finding a polynomial of degree $\le m-1$ with $m \le n$ which minimizes*

$$\sum_{j=1}^{n} \left| \sum_{k=1}^{m} \alpha_k \phi_k(x_j) - y_j \right|^2 .$$

In this case, the choice of $\phi_k(x) = x^{k-1},\ k = 1, \cdots, m$.

For a representative $\phi(x; \boldsymbol{\alpha}) = \sum_{k=1}^{m} \alpha_k \phi_k(x)$ with $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_m)^T$ to be determined. We now try to minimize $\sum_{j=1}^{n} |\phi(x_j; \boldsymbol{\alpha}) - y_j|^2$ over $\alpha \in \mathbb{R}^m$. Set

$$J(\boldsymbol{\alpha}) = \sum_{j=1}^{n} |\phi(x_j; \boldsymbol{\alpha}) - y_j|^2 = \sum_{j=1}^{n} \left| \sum_{k=1}^{m} \alpha_k \phi_k(x_j) - y_j \right|^2$$

for $\boldsymbol{\alpha} \in \mathbb{R}^m$. The least squares problem [*] is to find

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^m} J(\boldsymbol{\alpha}).$$

since $J(\boldsymbol{\alpha})$ is a quadratic polynomial in $\boldsymbol{\alpha}$, it has a minimum. To find this, one needs to find $\boldsymbol{\alpha}_* \in \mathbb{R}^m$ such that

$$\nabla J(\boldsymbol{\alpha}_*) = 0. \tag{3.6}$$

Notice that

$$\frac{\partial J}{\partial \alpha_p}(\boldsymbol{\alpha}) = 2 \sum_{j=1}^{n} \phi_p(x_j) \left(\sum_{k=1}^{m} \alpha_k \phi_k(x_j) - y_j \right),\ p = 1, \cdots, m. \tag{3.7}$$

[*]Gauss introduced the least squares method to predict the path of the planet *Ceres* in 1801.

Let

$$\mathbf{B} = (\beta_{pq}) = \left(\sum_{j=1}^{n} \phi_p(x_j)\phi_q(x_j) \right) \in \mathbb{R}^{m\times m}, \quad \mathbf{Y} = (y_1, \cdots, y_m)^T = \left(\sum_{j=1}^{n} y_j\phi_q(x_j) \right)^T.$$

Then from (3.7) and (3.7) one has

$$\mathbf{B}\boldsymbol{\alpha}_* = \mathbf{Y}. \tag{3.8}$$

Theorem 3.1. *If $n \ge m$ and $\phi_p(x) = x^{p-1}, p = 1, \cdots, m$, as in Example 3.1, the matrix $\mathbf{B}$ is a symmetric, positive-definite matrix.*

Proof. Consider for any $\boldsymbol{\alpha} \in \mathbb{R}^m$,

$$\begin{aligned}
\boldsymbol{\alpha}^T\mathbf{B}\boldsymbol{\alpha} &= \sum_{p=1}^{m}\sum_{q=1}^{m} \beta_{pq}\alpha_p\alpha_q \\
&= \sum_{p=1}^{m}\sum_{q=1}^{m}\sum_{j=1}^{n} \phi_p(x_j)\phi_q(x_j)\alpha_p\alpha_q \\
&= \sum_{j=1}^{n} \left[\sum_{p=1}^{m} \alpha_p\phi_p(x_j)\right] \left[\sum_{q=1}^{m} \alpha_q\phi_q(x_j)\right] \\
&= \sum_{j=1}^{n} \left[\sum_{p=1}^{m} \alpha_p\phi_p(x_j)\right]^2.
\end{aligned}$$

Thus $\boldsymbol{\alpha}^T\mathbf{B}\boldsymbol{\alpha} = 0$ if and only if $\sum_{p=1}^{m} \alpha_p\phi_p(x_j) = 0$ for all $j = 1, \cdots, n$. If $\phi_p(x) = x^{p-1}$, $\sum_{p=1}^{m} \alpha_p\phi_p(x)$ is a polynomial of degree $\le m-1$ which has at most $m-1$ distinct zeros. Hence if $n \ge m$, the least squares problem is always solvable. □

If $\mathbf{B}$ is symmetric, positive-definite, the Cholesky decomposition method or the conjugate gradient method can be easily applied to solve (3.8).

Remark 3.6. *For $p = 1, \cdots, m$, denote*

$$\mathbf{b}_p = (\phi_p(x_1), \cdots, \phi_p(x_n))^T \in \mathbb{R}^n, \quad \mathbf{y} = (y_1, \cdots, y_n)^T \in \mathbb{R}^n,$$

Then, (3.8) can be understood as

$$\sum_{q=1}^{m} (\mathbf{b}_p \cdot \mathbf{b}_q)\alpha_q = \mathbf{b}_p \cdot \mathbf{y}, \quad p = 1, \cdots, m.$$

or

$$\mathbf{b}_p \cdot \left(\sum_{q=1}^{m} \alpha_q \mathbf{b}_q - \mathbf{y} \right) = 0, \quad p = 1, \cdots, m.$$

Letting $\vec{f}(\alpha) = \frac{1}{2}\nabla J(\alpha)$, one gets

$$\frac{\partial f_p}{\partial \alpha_q}(\boldsymbol{\alpha}) = \sum_{j=1}^{n} \phi_p(x_j)\phi_q(x_j), \ p, q = 1, \cdots, m. \tag{3.9}$$

Now apply the multidimensional Newton method to find $\alpha \in \mathbb{R}^m$ such that

$$\vec{f}(\alpha) = \nabla\phi(\alpha) = 0.$$

For this, from (3.9) observe that the $(l,k)-$component of $\nabla \overrightarrow{f}(\alpha)$

$$\frac{\partial f_l}{\partial \alpha_k} = \sum_{j=1}^{n} \phi_k(x_j)\phi_l(x_j),$$

for $k, l = 1, \cdots, m$. Thus, with a suitable initial approximation, the Newton method to find the α is given as follows:

$$\alpha^{(iter+1)} = \alpha^{(iter)} - \left[\nabla \overrightarrow{f}(\alpha^{(iter)})\right]^{-1} \overrightarrow{f}(\alpha^{(iter)}), \quad \text{for } iter = 1, \cdots, max_{iter}.$$

3.6.2 Constrained least squares problems

Suppose we want to find the value of x that minimizes, but subject to the constraint $x^T x \le 1$
Define a Lagrangian by

$$L(x,\lambda) = f(x) + \lambda(x^T x - 1),$$

and then solve the minimax problem (saddle point problem)

$$\min_x \max_{\lambda,\lambda\ge 0} L(x,\lambda),$$

where

$$f(x) = \frac{1}{2}\|\mathbf{Ax} - \mathbf{b}\|_2^2.$$

Then we have

$$\nabla_x L(x,\lambda) = 0, \tag{3.10a}$$
$$\frac{\partial L(x,\lambda)}{\partial \lambda} = 0. \tag{3.10b}$$

From these we have

$$\mathbf{A}^T\mathbf{A}x - \mathbf{A}^T b + 2\lambda x = 0, \tag{3.11a}$$
$$x^T x - 1 = 0. \tag{3.11b}$$

The solution will take the form

$$x = (\mathbf{A}^T\mathbf{A} + 2\lambda I)^{-1}\mathbf{A}^T b$$

Chapter 4

Machine Learning Basics

4.1 Learning Algorithm

Definition 4.1. *[2] The definition of machine learning program is said*

- *to learn from experience E*
- *with respect to some class of tasks T,*
- *and performance measure P.*

if its performance in tasks in T, as measured by P, improves with experience E.

4.1.1 The Task, T

1. Classification

   In this type of task, the computer program is asked to specify which of k categories some input belongs to. To solve this task, the learning algorithm is usually asked to produce a function

   $$f : \mathbb{R}^n \to \{1, \cdots, k\}.$$

2. Classification with missing inputs

   When some of the inputs may be missing, rather than providing a single classification function, the learning algorithm must learn a set of functions. Each function corresponds to classifying x with a different subset of its inputs missing.

3. Regression

   In this type of task, the computer program is asked to predict a numerical value given some input. To solve this task, the learning algorithm is asked to output a function $f : \mathbb{R}^n \to \mathbb{R}$

4. Transcription

   In this type of task, the machine learning system is asked to observe a relatively unstructured representation of some kind of data and transcribe it into discrete, textual form.

5. Machine translation

   In a machine translation task, the input already consists of a sequence of symbols in some language, and the computer program must convert this into a sequence of symbols in another language.

6. Structured output

   Structured output tasks involve any task where the output is a vector (or other data structure containing multiple values) with important relationships between the different elements. This is a broad category, and subsumes the transcription and translation tasks described above, but also many other tasks.

7. Anomaly detection

   In this type of task, the computer program sifts through a set of events or objects, and flags some of them as being unusual or atypical.

8. Synthesis and sampling

   In this type of task, the machine learning algorithm is asked to generate new examples that are similar to those in the training data.

9. Imputation of missing values

   In this type of task, the machine learning algorithm is given a new example x $\in \mathbb{R}^n$ but with some entries x_i of x missing. The algorithm must provide a prediction of the values of the missing entries.

10. Denoising

    In this type of task, the machine learning algorithm is given in input a corrupted example $\tilde{x} \in \mathbb{R}^n$ obtained by an unknown corruption process from a clean example $\mathbf{x} \in \mathbb{R}^n$. The learner must predict the clean example x from its corrupted version $\tilde{x}$, or more generally predict the conditional prob dist'n $p(x \mid \tilde{x})$.

11. Density estimation or prob mass function estimation

    In the density estimation problem, the machine learning algorithm is asked to learn a function $p_{model} : \mathbb{R}^n \to \mathbb{R}$, where $p_{model}(x)$ can be interpreted as a prob density function (if x is continuous) or a prob mass function (if x is discrete) on the space that the examples were drawn from.

4.1.2 The Performance measure, P

- Usually we are interested in how well the machine learning algorithm performs on data that it has not seen before, since this determines how well it will work when deployed in the real world.
- We therefore evaluate these performance measures using a test set of data that is separate from the data used for training the machine learning system.
- Usually the performance measure P is specific to the task T being carried out by the system.
  - For tasks such as classification, classification with missing inputs, and transcription: measure the accuracy of the model.
    - Accuracy is just the proportion of examples for which the model produces the correct output.
    - measure the error rate, the proportion of examples for which the model produces an incorrect output.
      the expected 0-1 loss

$$\begin{cases} 0 & \text{if it is correctly classified,} \\ 1 & \text{if it is not.} \end{cases}$$

  - For tasks such as density estimation: use a performance metric that gives the model a continuous-valued score for each example. The most common approach is to report the average log-prob the model assigns to some examples.

4.1.3 The Experience, E

- Dataset: collection of many examples
- Iris dataset of 150 iris plants with features: the sepal length, sepal width, petal length and petal width.
- Unsupervised learning vs. Supervised learning

Unsupervised learning algorithms

- experience a dataset containing many features, then learn useful properties of the structure of this dataset.
- usually want to learn the entire prob dist'n that generated a dataset – synthesis or denoising
- dividing the dataset into clusters of similar examples– clustering
- observe several examples of a random vector x, and attempting to implicitly or explicitly learn the prob- dist'n $p(x)$, or some interesting properties of that dist'n,
- In unsupervised learning, there is no instructor or teacher, and the algorithm must learn to make sense of the data without this guide.
- regression, classification and structured output problems

Supervised learning algorithms

- experience a dataset containing features, but each example is also associated with a label or target.
- For example, the Iris dataset is annotated with the species of each iris plant. and learn to classify iris plants into three different species based on their measurements.
- observe several examples of a random vector x and an associated value or vector y, and learning to predict y from x, usually by estimating $p(y|x)$
- the target y being provided by an instructor or teacher who shows the machine learning system what to do.
- Density estimation in support of other tasks

Other variants of the learning paradigm

- semisupervised learning: some examples include a supervision target but others do not.
- multi-instance learning: an entire collection of examples is labeled as containing or not containing an example of a class, but the individual members of the collection are not labeled.
- Most machine learning algorithms simply experience a dataset
- Reinforcement learning algorithms interact with an environment, so there is a feedback loop between the learning system and its experiences

Design matrix

- A design matrix is a matrix containing a different example in each row.
- Each column of the matrix corresponds to a different feature.
- For instance, the Iris dataset contains 150 examples with four features for each example. For this dataset, a design matrix $X \in \mathbb{R}^{150\times 4}$.

4.1.4 Example: Linear Regression

Let

$$X = \{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\},$$

where $x^{(i)}$ and x may have different size.

1. In supervised learning: the example contains a label or target as well as a collection of features.
2. For object recognition from photographs, specify a numeric code, with 0 signifying a person, 1 signifying a car, 2 signifying a cat, etc.

3. If a dataset contains a design matrix of feature observations X, a vector of labels y, with $\mathbf{y}^{(i)}$ providing the label for example i.
4. The label may be more than just a single number. For example, if we want to train a speech recognition system to transcribe entire sentences, then the label for each example sentence is a sequence of words.
5. There is no formal definition of supervised and unsupervised learning.
6. There is no rigid taxonomy of datasets or experiences.

4.2 Capacity, Overfitting, and Underfitting

- Separate the training set and test set of examples
- Training error
- Test error (or generalization error): the expected value of the error on a new input. The expectation is taken across different possible inputs, drawn from the dist'n of inputs which the system encounters in practice.
- The $i.i.d$ assumptions: the data generating dist'n, denoted p_{data}, which is shared underlying dist'n:
  1. Each dataset are independent from each other
  2. The train set and test set are identically distributed, drawn from the same prob dist'n as each other.

Try to build a system $f : \mathbb{R}^n \to \mathbb{R}$ such that $y = f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w} \in \mathbb{R}^n$ is a parameter. Denote by $\hat{y}$ the value of our model predicts y should take on:

$$\hat{y} = \mathbf{w}^T\mathbf{x} = \sum_{j=1}^{n} w_j x_j \tag{4.1}$$

1. Define our task T: to predict y from $\mathbf{x}$ by outputting $\hat{y} = \mathbf{w}^T\mathbf{x}$.
2. Define our performance measure, P.
   - Suppose that we have a design matrix of m example inputs that we will not use for training, only for evaluating how well the model performs.
   - We also have a vector of regression targets providing the correct value of y for each of these examples.
   - the test set: $(\mathbf{X}^{(test)}, \mathbf{y}^{(test)}) \in \mathbb{R}^{n\times m} \times \mathbb{R}^m$ = (input design matrix, regression target)
   - In order to measure the performance of the model, compute the mean squared error of the model on the test set:

$$\text{MSE}_{\text{test}} = \frac{1}{m}\sum_{i=1}^{m}\left[(\hat{\mathbf{y}}^{(test)} - \mathbf{y}^{(test)})_i\right]^2 = \frac{1}{m}\|\hat{\mathbf{y}}^{(test)} - \mathbf{y}^{(test)}\|_2^2$$

3. The experience, E: improve the weights $\mathbf{w}$ to reduce MSE_{test} when the algorithm is allowed to gain experience by observing a training set $(\mathbf{X}^{(train)}, \mathbf{y}^{(train)}) \in \mathbb{R}^{n\times k} \times \mathbb{R}^k$
   - Minimize $\text{MSE}_{\text{train}}$:

$$\begin{aligned} 0 &= \nabla_{\mathbf{w}}\text{MSE}_{\text{train}} \\ &= \frac{1}{k}\nabla_{\mathbf{w}}\|\hat{\mathbf{y}}^{(train)} - \mathbf{y}^{(train)}\|_2^2 \\ &= \frac{1}{k}\nabla_{\mathbf{w}}\|\mathbf{X}^{(train)}\mathbf{w} - \mathbf{y}^{(train)}\|_2^2. \end{aligned}$$

- The normal equation:

$$\left[\mathbf{X}^{(train)}\right]^T \left[\mathbf{X}^{(train)}\right] \mathbf{w} = \left[\mathbf{X}^{(train)}\right]^T \mathbf{y}^{(train)}.$$

- Use the Cholesky decomposition or the Conjugate Gradient Method to get

$$\mathbf{w} = \left(\left[\mathbf{X}^{(train)}\right]^T \left[\mathbf{X}^{(train)}\right]\right)^{-1} \left[\mathbf{X}^{(train)}\right]^T \mathbf{y}^{(train)}.$$

Instead of (4.1), the linear regression uses an additional parameter, b, which is called a bias:

$$\hat{y} = \mathbf{w}^T\mathbf{x} = \sum_{j=1}^{n} w_j x_j + b \tag{4.2}$$

4.4 Estimators, Bias and Variance

4.4.1 Point Estimation

- An attempt to provide the single best prediction of some quantity of interest.
- Let $\{x^{(1)}, \cdots, x^{(m)}\}$ denote a set of m *i.i.d.* data points.
- A point estimator or statistics is any function of the data given as

$$\hat{\boldsymbol{\theta}}_m = g(x^{(1)}, \cdots, x^{(m)})$$

- "$\hat{\boldsymbol{\theta}}$" is called a point estimate of a parameter $\boldsymbol{\theta}$.
- No requirement that g return a value that is close to the true $\boldsymbol{\theta}$.
- No requirement that the range of g is the same as the set of allowable values of $\boldsymbol{\theta}$.
- This definition of a point estimator is very general and allows the designer of an estimator great flexibility.
- A good estimator is a function whose output is close to the true underlying $\boldsymbol{\theta}$ that generated the training data.
- The true parameter value $\boldsymbol{\theta}$ is assumed to be unknown but fixed, while the point estimate $\hat{\boldsymbol{\theta}}$ is a function of the data.
- Since the data is drawn from a random process, any function of the data is random. Therefore $\hat{\boldsymbol{\theta}}$ is a random variable.
- A point estimate is a function estimator that estimates the relationship between input and target variables.

Function estimation.

- To predict a variable $\mathbf{y}$ given an input vector $\mathbf{x}$.
- We assume that there is a function $f(\mathbf{x})$ that describes the approximate relationship between $\mathbf{x}$ and $\mathbf{y}$.
- we may assume that

$$\mathbf{y} = f(\mathbf{x}) + \boldsymbol{\epsilon},$$

  where $\boldsymbol{\epsilon}$ represents the part of $\mathbf{y}$ which is not predictable from $\mathbf{x}$.
- In function estimation, we are interested in approximating f with a model or estimate $\hat{f}$.
- Examples:
  1. $\mathbf{w}^T\mathbf{x} + \boldsymbol{\epsilon} \leftrightarrow$ estimate a parameter $\mathbf{w}$;
  2. Estimate $\hat{f}(\mathbf{x})$

4.4.2 Bias

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) := E(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta}$$

An estimator $\hat{\boldsymbol{\theta}}_m$ is

$$\begin{cases} \text{unbiased} & \text{if}\,\text{bias}(\hat{\boldsymbol{\theta}}_m) = 0, \\ \text{asymptotically unbiased} & \text{if } \lim_{m\to\infty} \text{bias}(\hat{\boldsymbol{\theta}}_m) = 0, \end{cases}$$

Each case means that

1. we can expect the expected value of estimator to be true $\boldsymbol{\theta}$ statistically.
2. if we choose sufficient number of data, the expected value of estimator come close to true $\boldsymbol{\theta}$

Example 4.1 (Bernoulli Dist'n)**.** *A Bernoulli Dist'n with set of samples* $\left\{x^{(1)}, \cdots, x^{(m)}\right\}$ *that are i.i.d. with mean* θ *is*

$$P(x^{(i)};\theta) = \theta^{x^{(i)}}(1-\theta)^{(1-x^{(i)})} \tag{4.3}$$

- *Choose the sample mean of the training data as an estimator for* θ *:*

$$\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^m x^{(i)}.$$

- $$\text{bias}(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta = E\left[\frac{1}{m}\sum_{i=1}^m x^{(i)}\right] - \theta = \frac{1}{m}\sum_{i=1}^m E\left[x^{(i)}\right] - \theta$$
$$= \frac{1}{m}\sum_{i=1}^m \sum_{x^{(i)}=0}^1 \left(x^{(i)}\theta^{x^{(i)}}(1-\theta)^{(1-x^{(i)})}\right) - \theta = \frac{1}{m}\sum_{i=1}^m(\theta) - \theta = \theta - \theta = 0$$

- *The sample mean estimator* $\hat{\theta}_m$ *is unbiased.*

Example 4.2 (Gaussian dist'n estimator of the mean)**.** *Consider* $\left\{x^{(1)}, \cdots, x^{(m)}\right\}$ *: i.i.d. according to a Gaussian dist'n*

- $p\left(x^{(i)}\right) = \mathcal{N}\left(x^{(i)};\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\frac{(x^{(i)}-\mu)^2}{\sigma^2}}$.
- *The sample mean estimator of Gaussian mean parameter:*

$$\hat{\mu}_m = \frac{1}{m}\sum_{i=1}^m x^{(i)}$$

- $$\begin{aligned}\text{bias}(\hat{\mu}_m) &= E(\hat{\mu}_m) - \mu = E\left[\frac{1}{m}\sum_{i=1}^m x^{(i)}\right] - \mu \\ &= \left(\frac{1}{m}\sum_{i=1}^m E\left[x^{(i)}\right]\right) - \mu = \left(\frac{1}{m}\sum_{i=1}^m \mu\right) - \mu = \mu - \mu = 0\end{aligned}$$

- *The sample mean* $\hat{\mu}_m$ *of Gaussian mean parameter is an unbiased estimator.*

Example 4.3 (Estimators of the variance of a Gaussian dist'n)**.** *For variance* σ^2 *of Gaussian dist'n.*

- *The sample variance estimator:*

$$\hat{\sigma}_m^2 := \frac{1}{m}\sum_{i=1}^m \left(x^{(i)} - \hat{\mu}_m\right)^2, \quad \hat{\mu}_m = \frac{1}{m}\sum_{i=1}^m x^{(i)}.$$

- *The sample variance estimator is biased:*

$$\begin{aligned} E(\hat{\sigma}_m^2) &= E\left[\frac{1}{m}\sum_{i=1}^m \left(x^{(i)} - \hat{\mu}_m\right)^2\right] = \frac{m-1}{m}\sigma^2, && (4.4a)\\ \text{bias}(\hat{\sigma}_m^2) &= E(\hat{\sigma}_m^2) - \sigma^2 = -\frac{1}{m}\sigma^2. && (4.4b)\end{aligned}$$

- *To prove (4.4), since $x^{(1)}, x^{(2)}, ..., x^{(n)} \sim \mathcal{N}(x; \mu, \sigma^2)$ are i.i.d., we get*

$$\begin{aligned} E(x^{(i)}) &= E(x) = \mu, && (4.5a)\\ \text{Var}(x^{(i)}) &= \text{Var}(x) = E(x^2) - \mu^2 = \sigma^2. && (4.5b)\end{aligned}$$

- *Since $\hat{\mu}_m = \frac{\sum_{i=1}^m x^{(i)}}{m}$, we have*

$$\begin{aligned} E(\hat{\mu}_m^2) &= \frac{1}{m^2}E((\sum_{i=1}^m x^{(i)})^2)\\ &= \frac{1}{m^2}\left[\sum_{i=1}^m E((x^{(i)})^2) + \sum_{j\neq k} E(x^{(j)})E(x^{(k)})\right]\\ &= \frac{1}{m^2}\left[mE(x^2) + (m^2 - m)E(x)E(x)\right]\\ &= \frac{1}{m}\left[E(x^2) + (m-1)E(x)^2\right]\\ &= \frac{1}{m}\left[(\sigma^2 + \mu^2) + (m-1)\mu^2\right] = \frac{\sigma^2}{m} + \mu^2.\end{aligned}$$

- *The unbiased sample variance estimator:*

$$(\tilde{\sigma}_m)^2 := \frac{1}{m-1}\sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2. \qquad (4.6)$$

Indeed, we have

$$\begin{aligned} E(\sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2) &= E(\sum_{i=1}^m (x^{(i)})^2 - 2\hat{\mu}_m \sum_{i=1}^m x^{(i)} + m\hat{\mu}_m^2)\\ &= \sum_{i=1}^m E((x^{(i)})^2) - mE(\hat{\mu}_m^2)\\ &= m(\sigma^2 + \mu^2) - (\sigma^2 + m\mu^2) = (m-1)\sigma^2\\ &= (m-1)E((\tilde{\sigma}_m)^2).\end{aligned}$$

- $E\left(\tilde{\sigma}_m^2\right) = E\left[\frac{1}{m-1}\sum_{i=1}^m \left(x^{(i)} - \hat{\mu}_m\right)^2\right] = \frac{m}{m-1}E(\hat{\sigma}_m^2) = \frac{m}{m-1}\left(\frac{m-1}{m}\sigma^2\right) = \sigma^2.$

Remark 4.1. Bessel's correction *In statistics, to correct the bias in the estimation of the population variance, use $(m-1)$ instead of m in the formula for the sample variance and sample standard deviation, where m is the number of observations in a sample.*

Notice that in the m independent observations in the sample,

$$\left(x^{(1)} - \hat{\mu}_m, x^{(2)} - \hat{\mu}_m, \cdots, x^{(m)} - \hat{\mu}_m\right)$$

are linearly dependent with rank $m-1$. since

$$\sum_{j=1}^{m} (x^{(j)} - \hat{\mu}_m) = 0.$$

Remark 4.2. *Note that unbiased estimator is not always the best estimator. And we often use biased estimator that possess other important properties.*

4.4.3 Variance and Standard Error

The variance of an estimator is defined as

$$\mathrm{Var}(\hat{\boldsymbol{\theta}})$$

and the standard error is defined as

$$SE(\hat{\boldsymbol{\theta}}) := \sqrt{\mathrm{Var}(\hat{\boldsymbol{\theta}})}.$$

.

The standard error of the mean is given as follows:

$$SE\left(\hat{\mu}_m\right) = \sqrt{Var\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}.$$

- The standard error of the mean is very useful in machine learning.
- The generalization errors are estimated by the sample mean of the error on the test set.
- The number of examples in the test set determines the accuracy of this estimate.
- By the central limit theorem, which tells us that the mean will be approximately distributed with a normal dist'n, with the mean $\hat{\mu}_m$ and the variance $SE(\hat{\mu}_m)^2$, we can use the standard error to compute the prob that the true expectation falls in any chosen interval.

Example 4.4 (Bernoulli dist'n)**.** *Assume that $x^{(1)}, x^{(2)}, ..., x^{(n)} \sim \mathcal{N}(x; \mu, \sigma^2)$ are i.i.d., from $P(x^{(i)}; \theta) = \theta^{x^{(i)}}\left(1-\theta)^{(1-x^{(i)})}\right)$. Invoking the sample mean estimator*

$$\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)},$$

we get

$$\mathrm{Var}(\hat{\theta}_m) = \frac{1}{m^2}\sum_{i=1}^{m} \mathrm{Var}(x^{(i)}) = \frac{1}{m}\theta(1-\theta).$$

4.4.4 Trading off Bias and Variance to minimize MSE

1. Bias and variance measure two different sources of errors in an estimator.
   (a) Bias: the expected deviation from the true value of the function or parameter.
   (b) Variance: the deviation from the expected estimator value that any particular sampling of the data is likely to cause.

2. Cross-validation : the trade-off to negotiate between large bias and large variance.

3. The mean squared error (MSE) of the estimates defined as follows:

$$MSE := E[(\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta})^2] = \text{bias}(\hat{\boldsymbol{\theta}}_m)^2 + \text{Var}(\hat{\boldsymbol{\theta}}_m) \tag{4.7}$$

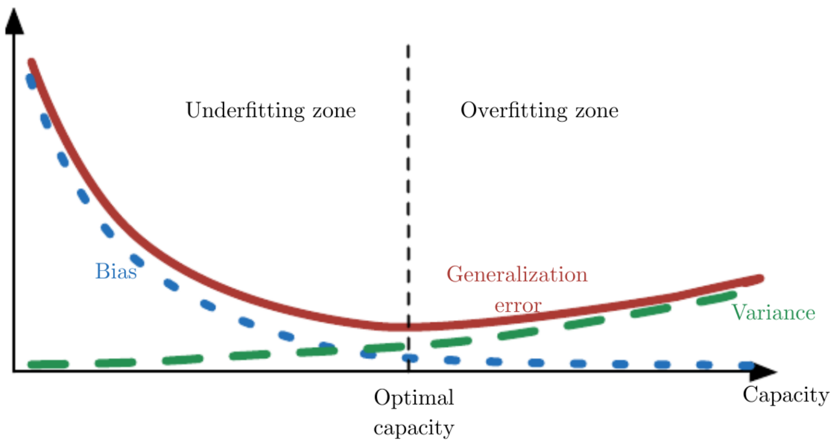

Figure 4.4.1: (This figure is Figure 5.6, p. 130 [GBC]): As capacity increases (x-axis), bias (dotted) tends to decrease and variance (dashed) tends to increase, yielding another U-shaped curve for generalization error (bold curve). If we vary capacity along one axis, there is an optimal capacity, with underfitting when the capacity is below this optimum and overfitting when it is above. This relationship is similar to the relationship between capacity, underfitting, and overfitting.

4.4.5 Consistency

1. The convergence in prob

$$\lim_{m\to\infty} \hat{\theta}_m \xrightarrow{p} \theta,$$

which means that $\forall \varepsilon > 0$, $P(|\hat{\theta}_m - \theta| > \varepsilon) \to 0$ as $m \to \infty$.

2. Almost sure convergence

$$p(\lim_{m\to\infty} \mathbf{x}^{(m)} = x) = 1.$$

4.5 Maximum likelihood estimation

- $\mathbb{X} := \left\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\right\}$: the set of m examples drawn independently from the true but unknown data-generating dist'n data $(\mathbf{x})$
- $p_{model}(\mathbf{x}; \boldsymbol{\theta})$: be a parametric family of prob dist'ns over the same space indexed by $\boldsymbol{\theta}$
- In other words, $p_{model}(\mathbf{x}; \boldsymbol{\theta})$ maps any configuration $\mathbf{x}$ to a real number estimating the true prob data $(\mathbf{x})$.
- The maximum likelihood estimator for $\boldsymbol{\theta}$ is then defined as follows:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{mode1}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{model}(\mathbf{x}^{(i)}; \boldsymbol{\theta}).$$

- Since the products of probs are prone to numerical underflow, we can take the log of the likelihood and get the following expression:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{model}(\mathbf{x}^{(i)}; \boldsymbol{\theta})$$

- Since argmax does not change while rescaling the cost function, we can divide m to obtain the following form which is expressed with respect to empirical dist'n $\hat{p}_{data}(\mathbf{x})$ defined by training data

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} E_{\mathbf{x}\sim\hat{p}_{data}} \log p_{model}(\mathbf{x};\boldsymbol{\theta}) \tag{4.8}$$

- One way to interpret likelihood estimation is to view it as minimizing the dissimilarity. And degree of dissimilarity between empirical dist'n $\hat{p}_{data}(\mathbf{x})$ defined by the training set and the model dist'n is measured by the KL divergence. And the corresponding KL divergence is given by

$$D_{KL}(\hat{p}_{data}||\, p_{model}) = E_{\mathbf{x}\sim\hat{p}_{data}}[\log \hat{p}_{data}(\mathbf{x}) - \log p_{model}(\mathbf{x})]$$

thus to get good model we only need to minimize the following part

$$-E_{\mathbf{x}\sim\hat{p}_{data}}[\log p_{model}(\mathbf{x})]$$

Note that minimizing this KL divergence corresponds exactly to maximization of (4.8).

4.5.1 Conditional Log-Likelihood and MSE

Note that maximum likelihood estimator can be readily generalized to estimate a conditional prob $P(y|\mathbf{x};\boldsymbol{\theta})$ in order to predict $\mathbf{y}$ given $\mathbf{x}$. It can be formulated as follows

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} P(Y|X;\boldsymbol{\theta})$$

where X represents all our inputs and Y represents all our observed target. And under assumption that each example is i.i.d., by taking log, we can get the following form

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P(y^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta})$$

Example 4.5 (Linear regression as Maximum Likelihood). • *$\hat{y} = b + \mathbf{w}^T\mathbf{x}$ by MSE;*

- *For the same input value $\mathbf{x}$, we may think of different values y. The learning algorithm is to fit the dist'n $p(y|\mathbf{x})$ to all possible y values that can be produced by $\mathbf{x}$*
- *Define $p(y|\mathbf{x}) := \mathcal{N}\left(y;\hat{y}(\mathbf{x},\mathbf{w}),\sigma^2\right)$,*
- *with $\mu = \hat{y}(\mathbf{x},\mathbf{w})$; σ fixed by the user.*
- *Then,*

$$\begin{aligned} p_{model}\left(y^{(i)};\mu,\sigma^2\right) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{\left(y^{(i)}-\mu\right)^2}{\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{\left(y^{(i)}-\hat{y}^{(i)}\right)^2}{\sigma^2}\right) \end{aligned}$$

- *We can get the following*

$$\begin{aligned} \boldsymbol{\theta}_{ML} &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p(y^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta}) \to \\ &= \sum_{i=1}^{m} \log p(y^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta}) = \sum_{i=1}^{m} \log\left(\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\|\frac{y^{(i)}-\hat{y}^{(i)}}{\sigma}\|^2\right)\right) \\ &= -m\log\sigma - \frac{m}{2}\log 2\pi - \sum_{i=1}^{m}\frac{\|y^{(i)}-\hat{y}^{(i)}\|^2}{2\sigma^2} \end{aligned}$$

- *From the above, note that maximizing log likelihood with respect to w equals to minimizing the mean squared error(MSE) which is*

$$MSE_{train} = \frac{1}{m}\sum_{i=1}^{m} \|y^{(i)} - \hat{y}^{(i)}\|^2$$

4.5.2 Support Vector Machine (SVM)

A hyperplane which is orthogonal to a unit vector $\mathbf{w}$ and passes $\mathbf{x}_0$ in $\mathbb{R}^n$ is given as follows:

$$H_{\mathbf{w},a} = \left\{\mathbf{x} \in \mathbb{R}^n | \mathbf{w}^T(\mathbf{x} - \mathbf{x}_0) = 0\right\},\ a = \mathbf{w}^T\mathbf{x}_0.$$

The distance between point $\mathbf{z}$ and the given hyperplane is defined by

$$d\left(H_{\mathbf{w},a}, \mathbf{x}_0\right) = \left|\mathbf{w}^T(\mathbf{z} - \mathbf{x}_0)\right|.$$

Note that if $a = 0$, then means $\mathbf{z}$ is orthogonal to $\mathbf{w}$.

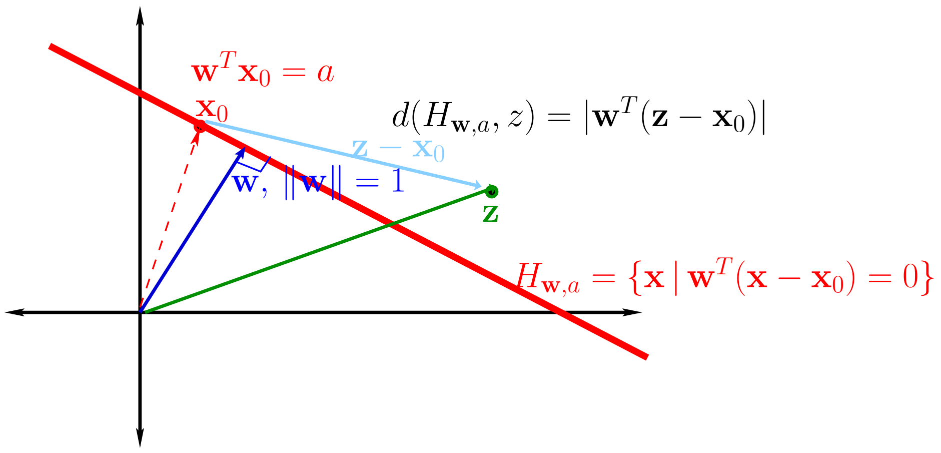
$\mathbf{w}^T\mathbf{x}_0 = a$
$\mathbf{x}_0$
$d(H_{\mathbf{w},a}, z) = |\mathbf{w}^T(\mathbf{z} - \mathbf{x}_0)|$
$\mathbf{z} - \mathbf{x}_0$
$\mathbf{w},\ \|\mathbf{w}\| = 1$
$\mathbf{z}$
$H_{\mathbf{w},a} = \{\mathbf{x} \,|\, \mathbf{w}^T(\mathbf{x} - \mathbf{x}_0) = 0\}$

Figure 4.5.2: The hyperplane $H_{\mathbf{w},a}$, where $\|\mathbf{w}\| = 1, a = \mathbf{w}^T\mathbf{x}_0$

Example 4.6. *Let* $\mathbf{w} = \frac{1}{\sqrt{2}}\binom{1}{1}$ *and* $a = \frac{1}{2}$. *Let* $\mathbf{x}_0 = \frac{1}{\sqrt{2}}\binom{1}{0}$. *Then,* $H_{\mathbf{w},a} = \{x_1 + x_2 = \frac{1}{\sqrt{2}}\}$.

Let $H_{\mathbf{w},0}$ be the homogeneous hyperplane passing through the origin. Then the hyperplane $H_{\mathbf{w},a} = H_{\mathbf{w},0} + \mathbf{x}_*$, where $\mathbf{x}_*$ lies on $H_{\mathbf{w},a}$ and parallel to $\mathbf{w}$.

Denote by $P\mathbf{z}$ the orthogonal projection of $\mathbf{z}$ on $H_{\mathbf{w},a}$, i.e. $P\mathbf{z} \in H_{\mathbf{w},a}$. Then $\mathbf{z} - P\mathbf{z} = \alpha\mathbf{w}$ for some α. We have $\mathbf{z} - P\mathbf{z} = \alpha\mathbf{w}$, from which $\rightarrow \langle\mathbf{w}, \mathbf{z} - P\mathbf{z}\rangle = \langle\mathbf{w}, \alpha\mathbf{w}\rangle = \alpha$. Hence,

$$d = |\alpha| = |\mathbf{w}^T\mathbf{z} - a|.$$

Using the SVM, we try to correctly classify a set of data to a known dataset (data sample). Let the data sample be of the form

$$\mathbf{S} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, 1\}, i = 1, \cdots, m.$$

Our aim is to separate the dataset $\mathbf{S}$ by the value of y by choosing a suitable hyperplane $H_{\mathbf{w},a}$. If such hyperplane(s) exists, the sample $\mathbf{S}$ is called **linearly separable**. We call a hyperplane $H_{\mathbf{w},a}$ is in **canonical form** relative to $\mathbf{S}$ if

$$\min_{(\mathbf{x},y)\in\mathbf{S}} |\mathbf{w}^T\mathbf{x} - a| = 1$$

Vectors that are closest to the separating hyperplane is called **support vectors**.

4.5.3 Other Simple Supervised Learning Algorithms

Break the input spaces into regions, with parameters associated with the regions.

1. The k-Nearest Neighbor regression
    - Nonparametric,
    - Store $(\mathbf{X}, \mathbf{y})$ from the training set
    - For a new test $\mathbf{x}$, find the nearest entry $\mathbf{X}_{i,:}$ such that $i = \text{argmin}\, \|\mathbf{X}_{i,:} - \mathbf{x}\|$ then return $\hat{y} = y_i$.
2. The kNN (k–Nearest Neighbors) algorithm
    - Nonparametric,
    - For a new test input $\mathbf{x}$, find the k–Nearest Neighbors $\mathbf{x}_j, j = 1, \cdots, k$ to $\mathbf{x}$; $\hat{y} = \frac{1}{k}\sum_{j=1}^{k} y_j$
    - high capacity: 1-Nearest Neighbor converges to 2 the Bayes error as $m \to \infty$ since it has to choose a single neighbor from the two possible equally distant neighbors.
    - high computational cost
    - One weakness: cannot learn that one feature is more discriminative than another. Example: $\mathbf{x} \in \mathbb{R}^{100}$, but $y = x_1$
3. Decision Tree algorithms
    - Nonparametric,

4.6 Unsupervised Learning Algorithms

- Unsupervised algorithms are those that experience only “features” but not a supervision signal.
- A classic unsupervised learning task: to find the “best” representation of the data.
- Preserves as much information about $\mathbf{x}$ as possible while obeying some penalty or constraint aimed at keeping the representation simpler or more accessible than $\mathbf{x}$ itself.
- By most common simpler representations,
    1. lower dimensional representations
    2. sparse representations
    3. independent representations

4.7 Basic optimization algorithms in machine learning

(This section is from pp.275-276 of [GBC].) $\boxed{\text{The cost function}}$: average over the (finite) training data

$$J(\boldsymbol{\theta}) = \mathbf{E}_{(\mathbf{x},y)\sim\hat{p}_{\text{data}}} L(f(\mathbf{x};\boldsymbol{\theta}), y) \tag{4.9}$$

- L: the per-example loss function;
    - arguments of L are $f(\mathbf{x};\boldsymbol{\theta})$ and y for unregularized supervised case
    - y: the target output in the case of supervised learning; dropped out in the unsupervised learning
    - arguments of L can include $\boldsymbol{\theta}$ or $\mathbf{x}$ for regularization
- $f(\mathbf{x};\boldsymbol{\theta})$: predicted output for the input $\mathbf{x}$
- $\hat{p}_{\text{data}}$: the empirical distribution

4.7.1 Empirical Risk Minimization

- $\boxed{\text{The cost function}}$: expectation across the data generating dist'n data

$$J^*(\boldsymbol{\theta}) = \mathbf{E}_{(\mathbf{x},y)\sim p_{\text{data}}} L(f(\mathbf{x};\boldsymbol{\theta}), y) \tag{4.10}$$

- $\boxed{\text{The goal of ML}}$ is to reduce the expected generalization error (4.11), which is known as the risk.
- Hence, $\boxed{\text{the goal of ML}}$ is risk minimization.
- In (4.11), the expectation is taken over the true underlying distribution p_{data}.
- If we knew the true distribution $p_{\text{data}}(\mathbf{x}, y)$, risk minimization would be an optimization task solvable by an optimization algorithm.
- However, when we do not know $p_{\text{data}}(\mathbf{x}, y)$ but only have a training set of samples, we have a machine learning problem.
- The simplest way to convert a machine learning problem back into an optimization problem is to minimize the expected loss on the training set.
- This means replacing the true distribution $p(\mathbf{x}, y)$ with the empirical distribution $\hat{p}(\mathbf{x}, y)$ defined by the training set.
- We now minimize the empirical risk:

$$\mathbf{E}_{(\mathbf{x},y)\sim p_{\text{data}}} L(f(\mathbf{x};\boldsymbol{\theta}), y) := \frac{1}{m}\sum_{j=1}^{m} L(f(\mathbf{x};\boldsymbol{\theta}), y) \tag{4.11}$$

- The training process based on minimizing this average training error is known as empirical risk minimization.
- In this setting, machine learning is still very similar to straightforward optimization. Rather than optimizing the risk directly, we optimize the empirical risk, and hope that the risk decreases significantly as well.

4.7.2 Minibatch

- Suppose the discrete case of samples:
- The generalization error (4.11) is given by

$$J^*(\boldsymbol{\theta}) = \sum_{\mathbf{x}}\sum_{y} p_{\text{data}}(\mathbf{x}, y) L(f(\mathbf{x};\boldsymbol{\theta}), y) \tag{4.12}$$

  with the gradient

$$\mathbf{g} = \nabla_{\boldsymbol{\theta}} J^*(\boldsymbol{\theta}) = \sum_{\mathbf{x}}\sum_{y} p_{\text{data}}(\mathbf{x}, y) \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x};\boldsymbol{\theta}), y). \tag{4.13}$$

- The gradient for the minibatch is

$$\hat{\mathbf{g}} = \frac{1}{m}\sum_{j=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(j)};\boldsymbol{\theta}), y^{(j)}). \tag{4.14}$$

4.7.3 Basic optimization algorithms in ML

- Steepest gradient algorithm
- Newton's method
- Conjugate gradient method

The SGD (Stochastic Gradient) method with batch

- Most widely used for ML/DL
- In each k step, sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$ from the training set $\mathbf{X}_{train}$ with corresponding targets $\{\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(m)}\}$
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon_k \frac{1}{m} \sum_{j=1}^m \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(j)}, \boldsymbol{\theta}), \mathbf{y}^{(j)})$
- The learning rate ε_k is decreasing;
- $\sum_{k=1}^\infty \varepsilon_k = \infty$
- $\sum_{k=1}^\infty \varepsilon_k^2 < \infty$
- $\varepsilon_k = \begin{cases} (1 - \frac{k-1}{\tau})\varepsilon_0 + \frac{k-1}{\tau}\varepsilon_\tau, & k = 1, \cdots, \tau \\ \varepsilon_\tau, & k = \tau + 1, \tau + 2, \cdots, \end{cases}$
- Usually $\varepsilon_\tau \sim \frac{\varepsilon_0}{100}$, $\mathcal{O}(\tau) = 100$;
- The excess error

$$J(\boldsymbol{\theta}^{(k)}) - \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{cases} \mathcal{O}(\frac{1}{\sqrt{k}}), & \text{for convex problems,} \\ \mathcal{O}(\frac{1}{k}), & \text{for strongly convex problems} \end{cases}$$

A function $f : \Omega \to \mathbf{C}$ is convex if

$$f((1-\alpha)\mathbf{x} + \alpha\mathbf{y}) \le (1-\alpha)f(\mathbf{x}) + \alpha f(\mathbf{y}) \quad \forall \mathbf{x}, \mathbf{y} \in \Omega,$$

A function $f : \Omega \to \mathbf{C}$ is strongly convex if

$$(\nabla f(\mathbf{x}) - \nabla f(\mathbf{y}))^T(\mathbf{x} - \mathbf{y}) \le c\|\mathbf{x} - \mathbf{y}\|_2^2 \quad \forall \mathbf{x}, \mathbf{y} \in \Omega,$$

or

$$f(\mathbf{y}) - f(\mathbf{x}) \ge \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{c}{2}\|\mathbf{x} - \mathbf{y}\|_2^2 \quad \forall \mathbf{x}, \mathbf{y} \in \Omega,$$

The SGD (Stochastic Gradient) method with momentum

- Update the parameter $\boldsymbol{\theta}$ and the velocity $\mathbf{v}$ at each iteration;
- In each k step, sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$ from the training set $\mathbf{X}_{train}$ with corresponding targets $\{\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(m)}\}$
- $\mathbf{g} \leftarrow \frac{1}{m} \sum_{j=1}^m \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(j)}, \boldsymbol{\theta}), \mathbf{y}^{(j)})$;
- $\mathbf{v} \leftarrow \alpha\mathbf{v} - \varepsilon\mathbf{g}$;
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$;
- If $\alpha > \varepsilon$, the effect of previous gradient is larger,
- Momentum = mass·velocity (mass is assumed to be unit);

- Set $\mathbf{v}^{(0)} = 0$.

$$\begin{aligned} \mathbf{v}^{(1)} &= \alpha \mathbf{v}^{(0)} - \varepsilon \mathbf{g}^{(0)}; \\ \mathbf{v}^{(2)} &= \alpha \mathbf{v}^{(1)} - \varepsilon \mathbf{g}^{(1)} = \alpha^2 \mathbf{v}^{(0)} - \varepsilon(\alpha \mathbf{g}^{(0)} + \mathbf{g}^{(1)}) \\ &\vdots \\ \mathbf{v}^{(k)} &= \alpha^k \mathbf{v}^{(0)} - \varepsilon \sum_{j=1}^{k} \alpha^{k-j} \mathbf{g}^{(j-1)} \end{aligned}$$

- Assume that $\mathbf{g}^{(j)} = \mathbf{g}$. With $\mathbf{v}^{(0)} = 0$, one has $\mathbf{v}^{(k)} = -\varepsilon \frac{1-\alpha^k}{1-\alpha} \mathbf{g}$. Hence, for sufficiently large k and $\alpha = 0.9$ for instance, $\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(k-1)} - \varepsilon \frac{1}{1-\alpha} \mathbf{g}$, which means about 10 times acceleration in the direction of $-\mathbf{g}$.

- Common choice of $\alpha = 0.5, 0.9, 0.99, \ldots$

The SGD (Stochastic Gradient) method with Nesterov momentum

- Update the parameter $\boldsymbol{\theta}$ and the velocity $\mathbf{v}$ at each iteration;

- In each k step, sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$ from the training set $\mathbf{X}_{train}$ with corresponding targets $\{\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(m)}\}$

- $\mathbf{g} \leftarrow \frac{1}{m} \sum_{j=1}^{m} \nabla_{\boldsymbol{\theta}+\alpha\mathbf{v}} L(f(\mathbf{x}^{(j)}, \boldsymbol{\theta} + \alpha \mathbf{v}), \mathbf{y}^{(j)})$;

- $\mathbf{v} \leftarrow \alpha \mathbf{v} - \varepsilon \mathbf{g}$;

- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$;

- Nesterov momentum is understood as the standard momentum method with a correction factor.

4.7.4 Adam (ADAptive Moments) by Kingma and Ba 2014)

- Use the first and second momentums;

- $\varepsilon = 0.001$; the step size

- $\rho_1 = 0.9; \rho = 0.999$ the exponential decay rates for moments estimates

- $\delta = 10^{-8}$ numerical stabilization parameter

- In each k step, sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$ from the training set $\mathbf{X}_{train}$ with corresponding targets $\{\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(m)}\}$

- $\mathbf{g} \leftarrow \frac{1}{m} \sum_{j=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(j)}, \boldsymbol{\theta}), \mathbf{y}^{(j)})$;

- $\mathbf{v} \leftarrow \rho_1 \mathbf{v} + (1 - \rho_1) \mathbf{g}$

- $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

- $\hat{\mathbf{r}} \leftarrow \frac{1}{1-\rho_1^k} \mathbf{r}; \quad \hat{\mathbf{v}} \leftarrow \frac{1}{1-\rho_2^k} \mathbf{v}$

- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon \frac{\hat{\mathbf{v}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$ (with component wise cal.)

- $A \odot B$ means the Hadamard product: $(A \odot B)_{jk} = A_{jk} B_{jk}$ for all j, k

4.7.5 BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm

- Newton's method:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}),$$

  $\mathbf{H}$ the Hessian of J wrt $\boldsymbol{\theta}$. Converges at a rate of second order, but expensive to compute $\mathbf{H}^{-1}$

- BFGS: a quasi-Newton method
  1. $\mathbf{p}^{(k)}$: by solving $\mathbb{B}^{(k)}\mathbf{p}^{(k)} = -\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k)})$
  2. Perform a line search to find $\varepsilon^{(k)}$ such that $\varepsilon^{(k)} = \arg\min J(\boldsymbol{\theta}^{(k)} + \varepsilon\mathbf{p}^{(k)})$
  3. $\mathbf{q}^{(k)} = \varepsilon^{(k)}\mathbf{p}^{(k)}$
  4. $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \mathbf{q}^{(k)}$
  5. $\mathbf{z}^{(k)} = \nabla J(\boldsymbol{\theta}^{(k+1)}) - \nabla J(\boldsymbol{\theta}^{(k)})$
  6. $\mathbb{B}^{(k+1)} = \mathbb{B}_k + \frac{\mathbf{z}^{(k)}\mathbf{z}^{(k)T}}{\mathbf{z}^{(k)T}\mathbf{q}^{(k)}} - \frac{B^{(k)}\mathbf{q}^{(k)}\mathbf{q}^{(k)T}B^{(k)T}}{\mathbf{q}^{(k)T}B^{(k)}\mathbf{q}^{(k)}}$

4.7.6 L-BFGS algorithm

Limited-memory BFGS algorithm

4.8 Optimization for parameter $\boldsymbol{\theta}$ in training deep models

4.8.1 Basic principles of gradient-based optimization

The steepest descent method:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

4.8.2 Optimization for training in machine learning

1. Consider the $\boxed{\text{optimization of the cost function}}$ as an average over the training set

$$J(\boldsymbol{\theta}) = \mathbf{E}_{(\mathbf{x},\mathbf{y})\sim \hat{\text{data}}} L(f(\mathbf{x},\boldsymbol{\theta}),\mathbf{y})$$

   - L : the loss function;
   - $f(\mathbf{x},\boldsymbol{\theta})$: the prescribed output with the input $\mathbf{x}$;
   - $\hat{\text{data}}$: the empirical dist'n
   - $\mathbf{y}$: the target output in the supervised learning

2. $\boxed{\text{The empirical risk minimization problem}}$: with m number of training examples,

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbf{E}_{(\mathbf{x},\mathbf{y})\sim \hat{\text{data}}} L(f(\mathbf{x},\boldsymbol{\theta}),\mathbf{y}) && (4.15)\\ &= \frac{1}{m}\sum_{j}^{m} L(f(\mathbf{x}^{(j)},\boldsymbol{\theta}),\mathbf{y}^{(j)}). && (4.16)\end{aligned}$$

   Most effective optimization algorithms are based on gradient descent methods, but useful loss functions have no useful derivatives...

3. Batch or deterministic optimization algorithms use the all training sets

4. Stochastic algorithms: use only a single example at a time;

5. Minibatch stochastic methods: use more than one but less than all of the training examples

Chapter 5

CNN (Convolution Neural Networks)

5.1 Convolutions

- Discrete convolution

$$(x * k)[n] = \sum_{j=-\infty}^{\infty} x(j)k(n-j).$$

- Cross-correlation

$$(x * k)(s) = \int x(t)k(s+t)\,dt.$$

- Discrete cross-correlation for two–dimensional input (e.g., images):

$$(x * k)[m,n] = \sum_{j=1}\sum_{k=1} x(m+j, n+k)k(j,k).$$

- Cross-correlation of random vectors $\mathbf{X} = (X_1, \cdots, X_m)^T$ and $\mathbf{Y} = (Y_1, \cdots, Y_m)^T$ is an $m \times n$ matrix whose jk-components are $E(X_j Y_k)$.

- Zero-normalized cross-correlation of vector-valued functions f and g is the cosine

$$\frac{\langle F, G\rangle}{\|F\|_{L^2(\Omega)}\|G\|_{L^2(\Omega)}},$$

  where

$$F(x) = f(x) - \mu_f,\; G(x) = g(x) - \mu_g$$

- Discrete zero-normalized cross-correlation

$$\frac{1}{n\sigma_f\sigma_g}\sum_{j,k}(f(j,k) - \mu_f)(g(j,k) - \mu_g)$$

- For $m \geq k-1$ and $n \geq \ell - 1$, the cross-correlation of an $m \times n$ matrix A and a $k \times \ell$ matrix B is an $(m-k+1) \times (n-\ell+1)$ matrix such that

$$(A * B)_{pq} = \sum_{r=1}^{k}\sum_{s=1}^{\ell} a_{p+r-1,q+s-1}b_{rs}.$$

A Convolutional Neural Network (CNN)

- Convolutional layers (often with a subsampling step) + fully connected layers

- Take advantage of the 2D structure of an input image (or other 2D input such as a speech signal).
- Local connections + tied weights followed by some form of pooling which results in translation invariant features.
- Easier to train and contain fewer parameters than fully connected networks with the same number of hidden units
- Back propagation algorithm to compute the gradient with respect to the parameters of the model in order to use gradient based optimization.

5.1.1 3-stage CNN

1. A set of linear activations are obtained by performing several convolutions in parallel
2. Detector stage: each linear activation is run through a nonlinear activation function, such as the ReLU activation function.
3. Pooling (a further modification the output layer by a pooling function): the max pooling within a rectangular nbd, the averaging within a rectangular nbd, the L^2-norm of a rectangular nbd, a weighted average, etc; invariance to translation which means that if the input is translated by a small amount, the values of most of the pooled outputs do not change.

Leverage of CNN

- Sparse interaction (or sparse connectivity, or sparse weights): making the kernel smaller than the inputs (An image may contains $10^3 - 10^6$ pixels, but edges may contains only $10 - 100$ pixels.
- Parameter sharing (or network has tied weight): in a traditional neural net, each element of the weight matrix is used only once to compute the output of a layer. But in a CNN, each member of the kernel is used at every position of the input
- Equivalent representation: a function f is equivalent to g if $(f \circ g)(x) = (g \circ f)(x)$. If T_a is a translate by a, i.e., $T_s(t) = t - a$, then

$$\begin{aligned}(x \circ T_a) * k(s) &= \int x(T_a(t))k(s+t)\,dt = \int x(t-a)k(s+t)\,dt \\ &= \int x(t-a)k((s+a)+t-a)\,dt \\ &= (x*k)(s+a) = (x*k)(T_a(s)).\end{aligned}$$

5.2 Neural Network with Scikit-Learn

We follow “Introduction to Neural Networks with Scikit-Learn” by Scott Robinson
`https://stackabuse.com/introduction-to-neural-networks-with-scikit-learn/#disqus_thread`
Prepare to install the following scikit-learn, NumPy, and SciPy

```
sudo pip install -U scikit-learn
sudo pip install -U NumPy
sudo pip install -U SciPy
```

Now open “python” in your terminal and proceed as follows.

Python Program 5.1: CNN for iris data

```
#sudo pip install -U scikit-learn
#sudo pip install -U NumPy
#sudo pip install -U SciPy
#sudo pip install -U pandas
#Now open ''python'' in your terminal and proceed as follows.
```

```
from numpy import array
import pandas as pd

# Location of dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign column names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
irisdata = pd.read_csv(url, names=names)
irisdata.head()
irisdata.tail()

# Assign data from first four columns to X variable
X = irisdata.iloc[:, 0:4]

# Assign data from the fifth column to y variable
y = irisdata.select_dtypes(include=[object])
y.head()

#convert these categorical values to numerical values
y.Class.unique()

#array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
y = y.apply(le.fit_transform)
y
array([0, 1, 2], dtype='int64')

# To create training and test splits, execute the following script:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.30)

#Feature scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

#Training and Predictions
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(X_train, y_train.values.ravel())

#The first parameter, hidden_layer_sizes, is used to set the size of the hidden layers.
#In our script we will create three layers of 10 nodes each.
# There is no standard formula for choosing the number of layers and
#nodes for a neural network
# and it varies quite a bit depending on the problem at hand.
#The best way is to try different combinations and see what works best.

#The second parameter to MLPClassifier specifies the number of
#iterations, or the epochs,
# that you want your neural network to execute.
# Remember, one epoch is a combination of one cycle of feed-forward
# and back propagation phase.

#By default the 'relu' activation function is used with 'adam' cost
#optimizer.
#However, you can change these functions using the activation and solver parameters, respective

#In the third line the fit function is used to train the algorithm
# on our training data i.e. X_train and y_train.

#Now make prediction as the final step as follows
predictions = mlp.predict(X_test)

#Evaluate the algorithm

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))

import readline
readline.write_history_file('/tmp/cnn.py')
```

Python Program 5.2: CNN for vector map

```
#https://scikit-learn.org/stable/modules/neural_networks_supervised.html
from sklearn.neural_network import MLPClassifier
X_train = [[0., 0.], [1., 1.]]
y_train = [0, 1]
mlp = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=1)
mlp.fit(X_train, y_train)
###MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
###              beta_1=0.9, beta_2=0.999, early_stopping=False,
###              epsilon=1e-08, hidden_layer_sizes=(5, 2),
###              learning_rate='constant', learning_rate_init=0.001,
###              max_iter=200, momentum=0.9, n_iter_no_change=10,
###              nesterovs_momentum=True, power_t=0.5, random_state=1,
###              shuffle=True, solver='lbfgs', tol=0.0001,
###              validation_fraction=0.1, verbose=False, warm_start=False)
###
X_test = [[2., 2.], [-1., -2.]]
mlp.predict(X_test)
#array([1, 0])

[coef.shape for coef in mlp.coefs_]
#[(2, 5), (5, 2), (2, 1)]

mlp.predict_proba([[2., 2.], [1., 2.]])
#array([[1.967...e-04, 9.998...-01],
#       [1.967...e-04, 9.998...-01]])

X_train = [[0., 0.], [1., 1.]]
y_train = [[0, 1], [1, 1]]
mlp = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15,), random_state=1)
mlp.fit(X_train, y_train)
#MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
#              beta_1=0.9, beta_2=0.999, early_stopping=False,
#              epsilon=1e-08, hidden_layer_sizes=(15,),
#                   learning_rate='constant', learning_rate_init=0.001,
#                   max_iter=200, momentum=0.9, n_iter_no_change=10,
#                   nesterovs_momentum=True, power_t=0.5,  random_state=1,
#                   shuffle=True, solver='lbfgs', tol=0.0001,
#                   validation_fraction=0.1, verbose=False, warm_start=False)
X_test=[[1.,  2.],[0.,  0.]]
y_test=[[1.,  2.],[0.,  0.]]
predictions=mlp.predict(X_test)
#array([[1,  1],[0,  1]])
print(predictions)
import readline
readline.write_history_file('/tmp/cnn-vector-map.py')
```

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning.* MIT press, 2016.

[2] T. M. Mitchell et al. *Machine learning.* McGraw-hill New York, 1997.