1 Linear Regression

(a) For the $d = 1$ case, we take the derivative of the loss function L w.r.t. w_0 and w_1 and set them to 0 to get the following two equations:

$$\sum_k^N y^{(k)} = Nw_0 + w_1 \sum_k^N x_1^{(k)}$$
$$\sum_k^N x_1^{(k)} y^{(k)} = w_0 \sum_k^N x_1^{(k)} + w_1 \sum_k^N \left(x_1^{(k)}\right)^2.$$

In general for $d > 1$, the second equation is similar for the other derivatives.

$$\sum_k^N y^{(k)} = Nw_0 + \sum_{j=1}^d w_j \sum_k^N x_j^{(k)}$$
$$\sum_k^N x_i^{(k)} y^{(k)} = w_0 \sum_k^N x_i^{(k)} + \sum_{j=1}^d w_j \sum_k^N x_j^{(k)} x_i^{(k)} \qquad \text{for } 1 \le i \le d$$

To solve for $\boldsymbol{w}$, we can write the above equations in matrix form and factor out $\boldsymbol{w}$.

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} = \begin{bmatrix} N & \sum_k^N x_1^{(k)} & \cdots & \sum_k^N x_d^{(k)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$

$$\begin{bmatrix} x_i^{(1)} & x_i^{(2)} & \cdots & x_i^{(N)} \end{bmatrix} \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} = \begin{bmatrix} \sum_k^N x_i^{(k)} & \sum_k^N x_1^{(k)} x_i^{(k)} & \cdots & \sum_k^N x_d^{(k)} x_i^{(k)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \quad 1 \le i \le d$$

$$\therefore \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ x_d^{(1)} & x_d^{(2)} & \cdots & x_d^{(N)} \end{bmatrix} \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} = \begin{bmatrix} N & \sum_k^N x_1^{(k)} & \cdots & \sum_k^N x_d^{(k)} \\ \sum_k^N x_1^{(k)} & \sum_k^N x_1^{(k)} x_1^{(k)} & \cdots & \sum_k^N x_d^{(k)} x_1^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_k^N x_d^{(k)} & \sum_k^N x_1^{(k)} x_d^{(k)} & \cdots & \sum_k^N x_d^{(k)} x_d^{(k)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$

This can be succinctly expressed as

$$\boldsymbol{X}^T \boldsymbol{y} = \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{w}$$

where

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_d^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & \cdots & x_d^{(N)} \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}, \qquad \text{and } \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}.$$

Assuming $\boldsymbol{X}^T\boldsymbol{X}$ is invertible, we get the least squares estimate:

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

(b) The weight update rule can be obtained by taking the negative gradient of the loss function. From slide 9, the gradient of $L(\boldsymbol{w}, \boldsymbol{S})$ is

$$\frac{\partial L}{\partial \boldsymbol{w}} = 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - 2\boldsymbol{X}^T\boldsymbol{y}$$

Thus, we have the gradient update rule

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\left(\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{y}\right)$$

where η is a tunable learning rate. (The 2 doesn't matter since it's a constant factor.)

2 Logistic Regression

(a) With binary classification, we use the sigmoid σ activation function:

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

with $z = \boldsymbol{w}^T \hat{\boldsymbol{x}},\ \hat{\boldsymbol{x}} = (x_0, x_1, \dots x_d)$.

With multiclass classification, we have a set of weights *for each class*. This means we can apply sigmoid with $z = \boldsymbol{w}_i^T \hat{\boldsymbol{x}}$ where each $\boldsymbol{w}_i$ are the weights for class C_i.

Applying sigmoid for each class of weights, we get

$$\sigma(\boldsymbol{W}\hat{\boldsymbol{x}}) = \boldsymbol{r} \in (0, 1)^K$$

where K is the number of classes and $\boldsymbol{W}$ are the weights, with the k-th row being the weights of class C_i.

Intuitively, we want the activation output of $\boldsymbol{w}_i\hat{\boldsymbol{x}}$ to be close to the actual class C_i. That is, $\sigma(\boldsymbol{w}_i^T \hat{\boldsymbol{x}}^{(\ell)}) = y_i^{(\ell)}$.

Taking a hint from the likelihood function for binary classification, we derive a similar likelihood.

$$\prod_{\ell}^{N} \prod_{i}^{K} f(\boldsymbol{w}_i; \boldsymbol{x}^{(\ell)})^{y_i^{(\ell)}} (1 - f(\boldsymbol{w}_i; \boldsymbol{x}^{(\ell)}))^{1 - y_i^{(\ell)}}$$

with the log likelihood (our loss function) being

$$L(\boldsymbol{W}, \boldsymbol{x}) = -\sum_{\ell}^{N} \sum_{i}^{K} y_i^{(\ell)} \log f(\boldsymbol{w}_i; \boldsymbol{x}^{(\ell)}) + (1 - y_i^{(\ell)}) \log(1 - f(\boldsymbol{w}_i; \boldsymbol{x}^{(\ell)}))$$

(b) With the formulation from slide 17, the sum of the K outputs equal 1. The formulation considers the total ratio of output and is biased towards dominant values (since we take the exponent e^{z_j}). With the formulation presented above, *each* output is

in $(0, 1)$, so the sum is not fixed.

(c) With multiclass classification, since we want to find one and only label which the data belongs to, it's more appropriate to use softmax, since usually one class will dominate the loss.

The alternative formulation presented here may be more suitable for multilabel classification, which aims to identify multiple labels (i.e. not limited to one) applicable to a sample. For example, we can set some threshold value so that the output classes surpassing that threshold threshold are used as labels.

3 Feedforward Neural Network

(a)

$$\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} \\
\tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\
&= \frac{e^x + e^{-x} - 2e^{-x}}{e^x + e^{-x}} && \text{add an imaginary } e^{-x} \text{ term} \\
&= 1 - \frac{2e^{-x}}{e^x + e^{-x}} \\
&= 1 - \frac{2}{e^{2x}+1} && \text{multiply numerator and denominator by } e^x \\
&= 1 - 2\sigma(-2x)
\end{aligned}$$

(b)

$$\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} \\
\sigma'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \sigma(x)\frac{e^{-x}}{1+e^{-x}} \\
&= \sigma(x)\frac{1+e^{-x}-1}{1+e^{-x}} \\
&= \sigma(x)\left(1 - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x)(1-\sigma(x)) \\
\tanh(x) &= 1 - 2\sigma(-2x) \\
&= 4\sigma'(-2x) \\
&= 4\sigma(-2x)(1-\sigma(-2x))
\end{aligned}$$

(c) The weight update rules for three layers use the previous weights with

$$\Delta w^{[2]} = -\eta \sum_q z^{[1](q)} \sum_\ell \cdots w^{[3]} \cdots$$

$$\Delta w^{[1]} = -\eta \sum_q x^{(q)} \sum_\ell \cdots w^{[2]} \cdots$$

where $\cdots$ represent a bunch of other factors.

The issue is that the weights are all multiplied onto each other, meaning the weights will blow up and changes will likely be drastic. Additionally, the weights in each layer will fluctuate immensely as we apply these weights.

Since we're considering units with tanh, another downside is that the large values will saturate, so it becomes hard to differentiate between two nodes. This means we don't get too much meaningful value out of activation and it's difficult to differentiate between different nodes.

(d) In the forward pass, we compute $z = g(\sum wx)$ (a lot of details are stripped, but the general idea remains the same). Since we're supposing $g = \tanh$ and have $w = 0$, so $z = 0$ as well.

In the weight update rules, we have $\Delta w = \eta \sum \delta z$. Since all $z = 0$, so all $\Delta w = 0$ as well, and so the weights don't change at all, and the model doesn't learn.

4 Convolutional Neural Networks

(a) (i) The feature map would have size $\lfloor \frac{227+2p-w}{s} + 1 \rfloor = \lfloor \frac{227+4-7}{2} + 1 \rfloor = 113$.

(ii) $(128 \times 7 \times 7 + 1) \times 64 = 401472$ parameters.

(b) (i) Size $\lfloor \frac{227+0-1}{1} + 1 \rfloor = 227$.

(ii) $(128 \times 1 \times 1 + 1) \times 16 = 2064$ parameters.

(iii) Unchanged, with size $\lfloor \frac{227+4-7}{2} + 1 \rfloor = 113$.

(iv) $(16 \times 7 \times 7 + 1) \times 64 = 50240$ parameters.

(v) $1 - \frac{50240+2064}{401472} \approx 86.97\%$ reduction, 13.03% kept.

5 Principal Component Analysis

(a)

$$\boldsymbol{\mu} = \begin{bmatrix} \frac{2+3+4+5+6+7}{6} \\ \frac{1+5+3+6+7+8}{6} \end{bmatrix} = \begin{bmatrix} 4.5 \\ 5 \end{bmatrix}$$

(b)

$$\mathcal{S} - \boldsymbol{\mu} = \left\{ \begin{bmatrix} -2.5 \\ -4 \end{bmatrix}, \begin{bmatrix} -1.5 \\ 0 \end{bmatrix}, \begin{bmatrix} -0.5 \\ -2 \end{bmatrix}, \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, \begin{bmatrix} 1.5 \\ 2 \end{bmatrix}, \begin{bmatrix} 2.5 \\ 3 \end{bmatrix} \right\}$$

(c)

$$\boldsymbol{\Sigma} = \begin{bmatrix} 3.5 & 4.4 \\ 4.4 & 6.8 \end{bmatrix}$$

(d)

$$\boldsymbol{W} = \begin{bmatrix} 0.450798 & 9.849203 \end{bmatrix}$$

```python
import numpy as np
from math import *

S = np.array([(2, 1), (3, 5), (4, 3), (5, 6), (6, 7), (7, 8)])
mu = S.mean(axis=0)

print('covariance:')
cov = np.cov(S.T)
print(cov)

print('eigenvalues:')
w, _ = np.linalg.eig(cov)
print(w)
```

(e) After taking the first principal component and projecting $\mathcal{S}$ to one dimension, our proportion of variance is

$$\frac{9.849203}{9.849203 + 0.450798} \approx 95.6\%$$

6 Clustering – Partitional Clustering

(a)
- p_1: **distance to (4, 2): 5.000**; distance to (11, 3): 11.180340
- p_2: **distance to (4, 2): 3.162**; distance to (11, 3): 10.000000
- p_3: **distance to (4, 2): 2.828**; distance to (11, 3): 9.055385
- p_4: **distance to (4, 2): 2.000**; distance to (11, 3): 5.099020
- p_5: **distance to (4, 2): 3.606**; distance to (11, 3): 5.000000
- p_6: distance to (4, 2): 4.123; **distance to (11, 3): 3.000000**
- p_7: distance to (4, 2): 5.099; **distance to (11, 3): 2.828427**

(b) C_1 points: $\{(0,5),(1,3),(2,4),(6,2),(7,0)\}$.

C_1 median: $(2,4)$.

C_2 points: $\{(8,3),(9,1)\}$.

C_2 median: $(8.5,2)$.

(c)
- p_1 **distance to (2, 4): 2.236**; distance to (8.5, 2): 9.013878
- p_2 **distance to (2, 4): 1.414**; distance to (8.5, 2): 7.566373
- p_3 **distance to (2, 4): 0.000**; distance to (8.5, 2): 6.800735
- p_4 distance to (2, 4): 4.472; **distance to (8.5, 2): 2.500000**
- p_5 distance to (2, 4): 6.403; **distance to (8.5, 2): 2.500000**
- p_6 distance to (2, 4): 6.083; **distance to (8.5, 2): 1.118034**
- p_7 distance to (2, 4): 7.616; **distance to (8.5, 2): 1.118034**

(d) C_0 points: {(0, 5), (1, 3), (2, 4)}

C_0 median: (1, 3)

C_1 points: {(6, 2), (7, 0), (8, 3), (9, 1)}

C_1 median: (7.5, 1.5)

(e) (1, 3) and (7.5, 1.5).

7 Clustering – Hierarchical Clustering

(a) $5 - 1 = 4$ merging steps.

(b) $C_1 \leftarrow \{(8, 8), (9, 8)\}$, with distance 1.

(c) $C_2 \leftarrow \{(1, 0), (2, 1)\}$, with distance 1.414.

(d) $C_3 \leftarrow \{(9, 6), \ldots C_1\}$, with distance 2.

(e) $C_4 \leftarrow \{\ldots C_2, \ldots C_3\}$, with distance 8.6023.

(f) For this particular dataset, 2 clusters would be appropriate. The distance between points within clusters C_2 and C_3 are usually small (1-2), but the distance between the two clusters is large (8.6).