Machine Learning

313704071 陳安定 Assignment 1

1.

Since SGD, m=1

$$\theta^{n+1} = \theta^n - \alpha\nabla_\theta Loss = \theta^n + 2\alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(y^i - h\left(x_1^i, x_2^i\right)\right)\nabla_\theta h\right]$$

$$\theta^1 = \theta^0 + 2\alpha[(3 - h(1,2))\nabla_\theta h]$$
$$= (4,5,6) + 2\alpha[(3 - \sigma(b + w_1 + 2w_2))\nabla_\theta\sigma(b + w_1 + 2w_2)]$$

2(a).

$$\sigma(x) = sigmoid\ function$$

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

$$s' = -1(1 + e^{-x})^{-2}(-e^{-x}) = \frac{e^{-x}}{(1 + e^{-x})^2}$$
$$= s \times \frac{e^{-x}}{1 + e^{-x}} = s(1 - s)$$

$$s'' = \left(s(1 - s)\right)' = s'(1 - s) + s(-s')$$
$$= s'(1 - s - s) = s'(1 - 2s)$$
$$= s(1 - s)(1 - 2s)$$

$$s''' = \left(s'(1 - 2s)\right)' = s''(1 - 2s) + s'(-2s')$$
$$= s'(1 - 2s)^2 - 2s'(s')$$
$$= s'(1 - 4s + 4s^2 - 2s')$$
$$= s(1 - s)(1 - 6s + 6s^2)$$

2(b).

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

$$= \frac{1}{2}\left(\frac{e^{\frac{x}{2}} + e^{-\frac{x}{2}}}{e^{\frac{x}{2}} + e^{-\frac{x}{2}}} + \frac{e^{\frac{x}{2}} - e^{-\frac{x}{2}}}{e^{\frac{x}{2}} + e^{-\frac{x}{2}}}\right)$$

$$= \frac{1}{2}\left(1 + \tanh\left(\frac{x}{2}\right)\right)$$

$$\tanh\left(\frac{x}{2}\right) = 2\sigma(x) - 1$$

3.

The sigmoid saturates for large-magnitude inputs: if $x \gg 0$, $\sigma(x) = \frac{1}{1+e^{-x}} \approx 1$; if $x \ll 0$, $\sigma(x) \approx 0$. Since $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, the previous situation will make $\sigma'(x)$ become near zero. During backpropogation, $\delta^{[i]} = \sigma'(z^{[i]}) \circ (W^{[i+1]})^T \delta^{[i+1]}$ Hence saturation makes $\delta^{[i]}$ tiny, leading to vanishing gradients and slow or stalled learning in lower layers.

Machine Learning

313704071 陳安定 Assignment 2

1. Consider a network as defined in (3.1) and (3.2). Assume that $n_L = 1$, find an algorithm to calculate $\nabla a^{[L]}(x)$.

(3.1) $a^{[1]} = x \in \mathbb{R}^{n_1}$

(3.2) $a^{[l]} = \sigma\left(W^{[l]}a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l}$ for $l = 2, 3, \ldots, L$.

$$\nabla a^{[L]}(x) = \begin{bmatrix} \frac{\partial a^{[l]}}{\partial W^{[l]}} \\ \frac{\partial a^{[l]}}{\partial b^{[4]}} \\ \frac{\partial a^{[l]}}{\partial a^{[l-1]}} \end{bmatrix}$$

$$\frac{\partial a^{[l]}}{\partial W^{[l]}} = \sigma'\left(W^{[l]}a^{[l-1]} + b^{[l]}\right)a^{[l-1]}$$

$$\frac{\partial a^{[l]}}{\partial b^{[4]}} = \sigma'\left(W^{[l]}a^{[l-1]} + b^{[l]}\right)$$

$$\frac{\partial a^{[l]}}{\partial a^{[l-1]}} = \sigma'\left(W^{[l]}a^{[l-1]} + b^{[l]}\right)W^{[l]}$$

2. Use a neural network to approximate the Runge function.

$$f(x) = \frac{1}{1+25x^2},\ x \in [-1,1].$$

The $f(x)$ is an even function, in $[-1,1]$, $0 < f(x) \leq 1$, maximum value $f(0) = 1$, minimum value $f(\pm 1) = \frac{1}{26}$. Moreover, $f(x)$ is decreasing when $x > 0$, and increasing when $x < 0$.

$$f'(x) = -\frac{50x}{(1+25x^2)^2}$$

***Hypothesis**:* An MLP trained with Chebyshev-like sampling and endpoint reweighting in the loss (weighted MSE); hidden layers use **tanh** and the output layer is linear. Expect to achieve $RMSE \leq \varepsilon$ and $L_\infty \leq \delta$, where ε, δ is the pre-set tolerance limits. ($RMSE$: root mean square error, L_∞: Maximum absolute error).

***Evaluation criteria**:* The reasons why we use these two criteria as follows, RMSE focus on the overall average error, it can reflect the approximate quality of the model in most intervals, and L_∞ check the worst-case performance of the model within the interval, especially the areas where the Runge function has large curvature at the endpoints and is most prone to oscillation.

***Chebyshev-like sampling**:* In the interval $[-1, 1]$, samples are distributed more densely at the endpoints to reduce the approximation error caused by the Runge function when the endpoint curvature is large.

***Model**:* Two hidden layers are sufficient to represent smooth functions. 64–128 neurons are recommended per layer to ensure approximation capability. (we use 64 neurons). About the Weight setting, adopt Xavier/Glorot-uniform and follow tanh activation function to set appropriate **gain** (about 5/3). The initial value of Bias is set to 0 to facilitate symmetric training.

$$W(x) = \frac{1}{\sqrt{1 - x^2 + \varepsilon}}$$

When x reaches ± 1, $1 - x^2$ approach 0, W(x) will increase. Used to strengthen the impact of endpoints

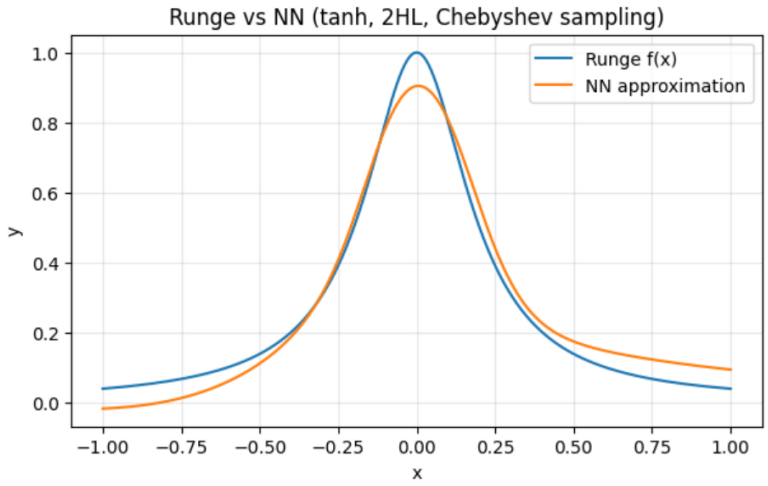
> *Note:* Xavier/Glorot-uniform initialization is a weight initialization method that aims to balance the signal variance in forward propagation and back propagation to avoid gradient explosion or disappearance.

***Forward / Backpropagation**:* By inputting x through (1-64-64-1)

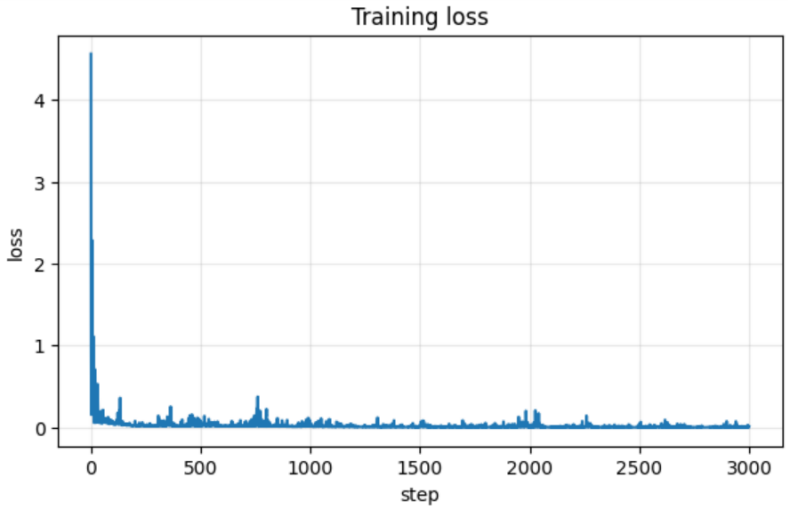
$$W^{[4]}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\boldsymbol{x} + b^{[2]}\right) + b^{[3]}\right) + b^{[4]}$$

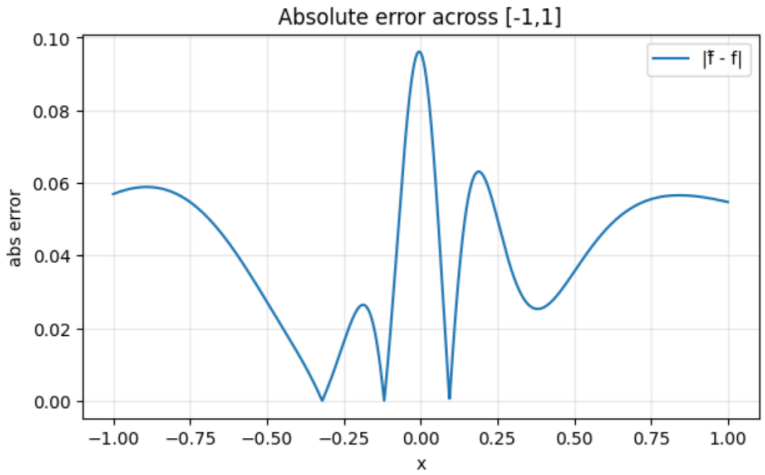
to produce approximate value $f'(x)$, then return error and calculate the gradient according to the loss function and update the weights and biases.

```
step     1 | loss=4.5633e+00 | RMSE=4.6317e-01 | L∞=1.0489e+00
step   500 | loss=8.2160e-02 | RMSE=1.7904e-01 | L∞=4.7691e-01
step  1000 | loss=1.6874e-02 | RMSE=1.4317e-01 | L∞=3.5808e-01
step  1500 | loss=2.3310e-02 | RMSE=9.2911e-02 | L∞=2.6680e-01
step  2000 | loss=2.9639e-03 | RMSE=6.1346e-02 | L∞=2.1472e-01
step  2500 | loss=2.9682e-03 | RMSE=4.2217e-02 | L∞=1.3388e-01
step  3000 | loss=1.1849e-02 | RMSE=4.6706e-02 | L∞=9.6132e-02
```



At the center point $x = 0$, NN's hotspot is slightly lower than the true value. Near the endpoints$|x| \approx 1$, the function value of NN is slightly higher than the true value.





The error for x>0 is larger than that for x<0.

Machine Learning

313704071 陳安定 Assignment 3

Lemma 3.1. Let $k \in \mathbb{N}_0$ *and* $s \in 2\mathbb{N} - 1$

$$\max_{\substack{p \le s \\ p \text{ odd}}} \left\| f_p - (\Psi_{s,\epsilon})_{\frac{p+1}{2}} \right\|_{W^{k,\infty}} \le \epsilon$$

This lemma shows how to approximate all odd degree monomials y^p with tanh neural network, and the error from the actual value is approached to an arbitrarily small error ϵ in $[-M, M]$ and under norm $W^{k,\infty}$.

The author first define the pth order central finite difference (中央有限差分) δ_h^p, which is used to exlicit the pth derivatives of tanh. Let $p \le s$.

$$\delta_h^p[f](x) = \sum_{i=0}^{p} (-1)^i \binom{p}{i} f\left(x + \left(\frac{p}{2} - i\right)h\right)$$

Use $\hat{f}_{q,h}(y)$ as an approximator, the purpose is to approach y^p, for any *p* is odd and lower than *s*. The proof will be described step by step.

$$\hat{f}_{q,h}(y) := \frac{\delta_{hy}^q[\sigma](0)}{\sigma^{(q)}(0)h^q}$$

With $0 \le m \le \min\{k, p+1\}$, by chain rule, the mth dirivitive of $\delta_{hx}^q[\sigma](0)$ is:

$$\frac{d^m}{dx^m}\delta_{hx}^p[\sigma](0) = \sum_{i=0}^{p} (-1)^i \binom{p}{i} \left(\frac{p}{2} - i\right)^m h^m \cdot \sigma^{(m)}\left(\left(\frac{p}{2} - i\right)hx\right)$$

Then, do Taylor expansion (for part of $\sigma^{(m)}$), and the main body is the finite sum from $l = m \; to \; p+1$ plus a remainder containing $\sigma^{(p+2)}(\xi_{x,i})$.

$$\frac{d^m}{dx^m}\delta_{hx}^p[\sigma](0) = \sum_{i=0}^{p} (-1)^i \binom{p}{i} \left(\frac{p}{2} - i\right)^m h^m \left(\sum_{l=m}^{p+1} \frac{\sigma^{(l)}(0)}{(l-m)!}\left(\frac{p}{2} - i\right)^{l-m} (hx)^{l-m}\right)$$

$$+ \sum_{i=0}^{p} (-1)^i \binom{p}{i} \left(\frac{p}{2} - i\right)^m h^m \left(\frac{\sigma^{(p+2)}(\xi_{x,i})}{(p+2-m)!}\left(\frac{p}{2} - i\right)^{p+2-m} (hx)^{p+2-m}\right)$$

By key Identity (Katsuura 2009):

$$\sum_{i=0}^{p}(-1)^i\binom{p}{i}\left(\frac{p}{2}-i\right)^l = p!\,\delta(1-p) = \begin{cases} p!\,, (\boldsymbol{l}=\boldsymbol{p}), \\ 0, \quad (l \neq p). \end{cases}$$

We can do some technical algebraic operations and rewrite blue part

$$\sum_{i=0}^{p}(-1)^i\binom{p}{i}\left(\frac{p}{2}-i\right)^m h^m\left(\sum_{l=m}^{p+1}\frac{\sigma^{(l)}(0)}{(l-m)!}\left(\frac{p}{2}-i\right)^{l-m}(hx)^{l-m}\right)$$

$$= h^m\sum_{l=m}^{p+1}\frac{\sigma^{(l)}(0)}{(l-m)!}(hx)^{l-m}p!$$

$$= h^m\frac{\sigma^{(p)}(0)}{(p-m)!}(hx)^{p-m}p!$$

$$= h^p\sigma^{(p)}(0)\boldsymbol{f_p^{(m)}}(\boldsymbol{x}), \qquad (0 \le m \le p)$$

Now we know $\boldsymbol{f_p^{(m)}}(\boldsymbol{x})$.

Then, we calculate $\hat{f}_{p,h}^{(m)}(x) - \boldsymbol{f_p^{(m)}}(\boldsymbol{x})$

$$= \frac{1}{h^p\sigma^{(p)}(0)}(red\ part)$$

$$= \sum_{i=0}^{p}(-1)^i\binom{p}{i}\frac{1}{(p+2-m)!}\frac{\sigma^{(p+2)}(\xi_{x,i})}{\sigma^{(p)}(0)}\left(\frac{p}{2}-i\right)^{p+2}h^2x^{p+2-m}$$

With the lower and upper bound on the derivatives of σ from Lemma A.1 and A.4, which yield for $m \le \min(k, p+1)$, and we should only know that error will be bound by an arbitrary number $(k \in \mathbb{N})$. And $h > 0$, such that for all $p < s$.

$$\left\|f_p - \hat{f}_{p,h}\right\|_{W^{m,\infty}} \le ((2(p+2)pM)^{p+3} + (2ph)^{k+1}))h^2 =: \boldsymbol{\epsilon}$$

The shallow tanh neural network $\Psi_{s,\epsilon}$, by $\left(\Psi_{s,\epsilon}\right)_p = \hat{f}_{p,h}$ such that it only has $\frac{s+1}{2}$ neurons in hidden layer. The width follows directly from its definition and the fact that σ is an odd function.

Lemma 3.2.

In Lemma 3.1., we made a monomial approximator with odd degree $\hat{f}_{p,h}$, and control $W^{k,\infty}$ error. The even power y^{2n} is not derived directly from difference but is expressed as **"The difference of two odd powers minus the linear combination of lower-order even powers"** using an algebraic equation. In this way, the higher-order even power can be constructed. To any $n \in \mathbb{N}, \alpha > 0$, it holds that

$$y^{2n} = \frac{1}{2\alpha(2n+1)}\left((y+\alpha)^{2n+1} - (y-\alpha)^{2n+1} - 2\sum_{k=0}^{n-1}\binom{2n+1}{2k}\alpha^{2(n-k)+1}y^{2k}\right)$$

Define $(\Psi_{s,\epsilon}(y))_{2n}$ as approximator of y^{2n}, and when $p = odd$, we can use Lemma3.1., so the above formula can be rewritten as:

$$(\Psi_{s,\epsilon}(y))_{2n} = \frac{1}{2\alpha(2n+1)}\left(\hat{f}_{2n+1,h}(y+\alpha) - \hat{f}_{2n+1,h}(y-\alpha)\right.$$

$$\left. - 2\sum_{k=0}^{n-1}\binom{2n+1}{2k}\alpha^{2(n-k)+1}(\Psi_{s,\epsilon}(y))_{2k}\right)$$

Introducing the notation $E_p = \left\| f_p - \left(\Psi_{s,\epsilon}\right)_p \right\|_{W^{m,\infty}}$, then

$$E_p \le E_p^* := \frac{2^{p/2}(1+\alpha)^{(p^2+p)/2}}{\alpha^{p/2}} \cdot \epsilon$$

Note that choosing h as in Lemma 3.1. implies that $\max_{\substack{p\le s \\ p\ odd}} E_p \le \epsilon$

The author uses induction steps to obtain that $E_{2n} \le E_{2n}^*$ for all $p \le s$.

By Lemma A.2, it proves that the optimal choice of α is $1/s$. It concludes that for any $\epsilon \ge 0$, there exist a shallow tanh neural network $\Psi_{s,\epsilon}$, such that

$$\max_{p\le s} \left\| f_p - \left(\Psi_{s,\epsilon}\right)_p \right\|_{W^{k,\infty}} \le \epsilon$$

Since it needs to use three inputs $(y-\alpha), y, (y+\alpha)$, the hidden layer width is $\frac{3(s+1)}{2}$ (three per position). The entire structure is still a single hidden layer, but it can compress all $0 \le p < s$ monomials to the specified accuracy at once.

Programming assignment

Review: Approximation of Runge function

Data:

In previous assignment, the sample data (training points) are distributed using **Chebyshev samples**. This sampling method will make the data points denser at the end points (x≈±1) and sparser in the middle (x≈0). It can reduce the error caused by Runge phenomenon (endpoint oscillation). However, since this method is ChatGPT teaches us, I skipped trying to use uniform-sampling. I found that because we use MSE regression training, the model is not forced to "step through" the training samples. Therefore, we don't need to use Chebyshev samples.

Hypothesis:

# of hidden layer: **2**

# of neurons in each hidden layer: **64**

Activation function: **tanh**

Two hidden layers can express the required curvature and nonlinear relationship, and about number of neurons, the scale (1-64-64-1) has achieved low RMSE and good L_∞ performance in a one-dimensional smooth approximation. As for the activation function, tanh is a smooth and oddly symmetric nonlinearity that can approximate the curvature with fewer nodes.

Loss function: MSE

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The reason why use MSE is that the model output is smooth, differentiable everywhere, and has simple gradients. Also, it is convenient for back propagation and learning rate setting. By the way, we report RMSE and L_∞ to intuitively interpret the average error size and avoid ignoring large local errors.

1. ***Use the same code to calculate the error in approximating the derivative of the given function.***

$$f'(x) = -\frac{50x}{(1+25x^2)^2}$$

Also use MSE to evaluate error between $f'(x)$ and $\hat{f}'(x)$, using PyTorch automatic differentiation to find the derivative of the model output with respect to the input.

```
dhat_dx = torch.autograd.grad(yhat.sum(), xs, create_graph=False)[0]
```

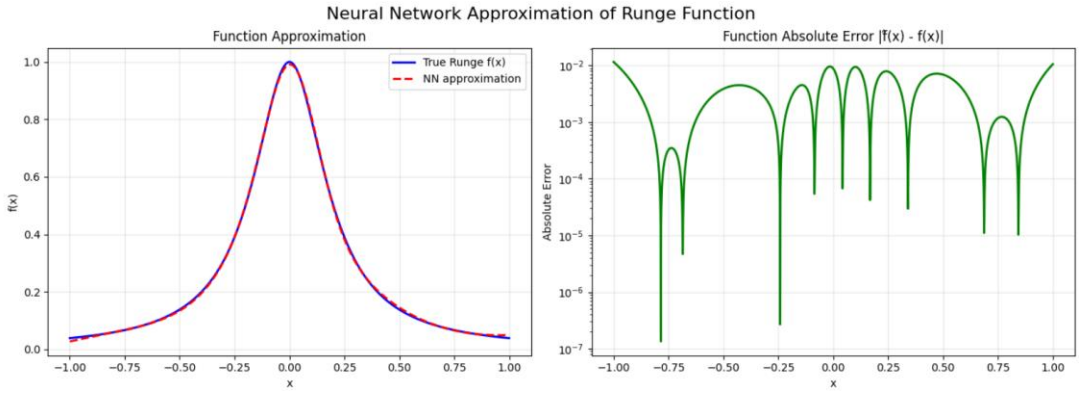
yhat.sum() is a n by 1 matrix, **xs** are independent variables, and **autograd.grad** means automatically differentiating each sample.

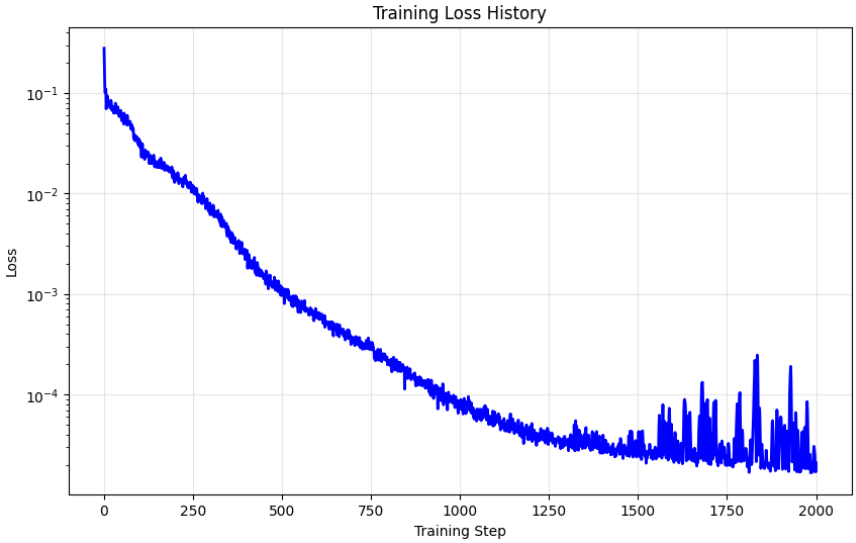
```
Mean absolute error: 5.596297e-02
Max absolute error: 2.636853e-01
Min absolute error: 1.087785e-05
Std of absolute error: 5.460906e-02
Median absolute error: 3.865539e-02
```

2. Use a neural network to approximate both the Runge function and its derivative.

Loss function of the function and the derivative function: both are **MSE**.

(1) Function

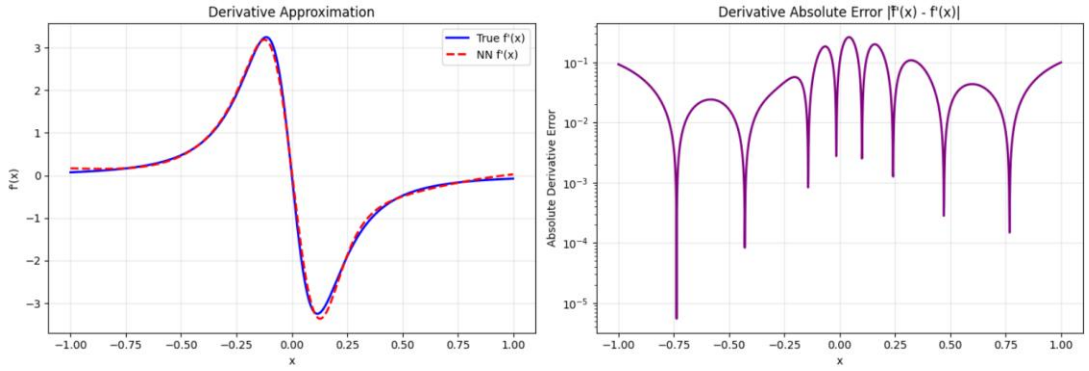


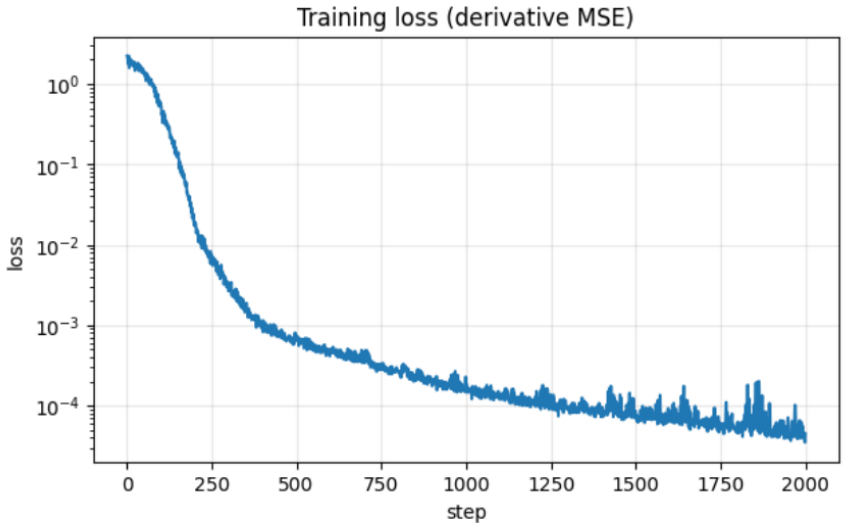


```
Neural Network Approximation of Runge Function
============================================================
Starting training with 2000 steps...
Sampling: Uniform
Sobolev regularization: False
------------------------------------------------------------
Step     1 | Loss: 2.7948e-01 | RMSE: 4.0390e-01 | L∞: 9.5313e-01
Step   500 | Loss: 1.1610e-03 | RMSE: 3.2524e-02 | L∞: 1.0493e-01
Step  1000 | Loss: 7.1744e-05 | RMSE: 8.9276e-03 | L∞: 3.0416e-02
Step  1500 | Loss: 2.9275e-05 | RMSE: 6.1242e-03 | L∞: 1.2912e-02
Step  2000 | Loss: 2.0908e-05 | RMSE: 4.6107e-03 | L∞: 1.1655e-02

============================================================
FINAL EVALUATION
============================================================
Function approximation:
  RMSE  = 4.607288e-03
  L_inf = 1.165528e-02
```

(2) Derivative function






```
Neural Network Approximation of Runge Function (Derivative-only)
============================================================
Starting training with 2000 steps...
Sampling: Uniform
Loss: derivative MSE (Sobolev style)
------------------------------------------------------------
Step     1 | Loss: 2.2285e+00 | RMSE_d: 1.4258e+00 | L∞_d: 3.5678e+00
Step   500 | Loss: 7.3314e-04 | RMSE_d: 2.6129e-02 | L∞_d: 8.0776e-02
Step  1000 | Loss: 1.4590e-04 | RMSE_d: 1.3840e-02 | L∞_d: 4.0524e-02
Step  1500 | Loss: 6.9204e-05 | RMSE_d: 8.5979e-03 | L∞_d: 2.3589e-02
Step  2000 | Loss: 4.4568e-05 | RMSE_d: 6.6351e-03 | L∞_d: 1.5527e-02

============================================================
FINAL EVALUATION
============================================================
Derivative approximation:
  RMSE_d  = 6.633780e-03
  L_inf_d = 1.552677e-02
```

Machine Learning

313704071 陳安定 Assignment 4

1. Data conversion

Original data from: https://opendata.cwa.gov.tw/dataset/observation/O-A0038-003

The original data is organized into two new data sets, namely (longitude, latitude, label) and (longitude, latitude, value), where the label is 0 or 1 and the value is the corresponding temperature value. The following is example.

```
lon,lat,label
120.0000,21.8800,0
120.0300,21.8800,0
120.0600,21.8800,0
120.0900,21.8800,0
120.1200,21.8800,0
120.1500,21.8800,0
120.1800,21.8800,0
```

Label data

```
lon,lat,value
120.8400,21.9400,29.80
120.7200,21.9700,30.70
120.7500,21.9700,30.70
120.7800,21.9700,30.00
120.8100,21.9700,30.20
120.8400,21.9700,29.10
120.7200,22.0000,30.80
```

Value data

2. Classification model – Logistic Regression

Hypothesis function:

$$h_\theta(x) = \hat{p} = P(y = 1|x)$$

$$h_\theta(x) = \sigma(\theta^T \tilde{x}) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

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

Convert the input features into the probability of belonging to a certain category, and use the sigmoid function to compress the output to between 0 and 1.

$$\hat{p} = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)}}$$

$\hat{p}$: The probability of predicted $y = 1$

θ_0, θ_1, θ_2: Parameters that the model to learn

x_1, x_2: Input features (x_1: longitude, x_2: latitude)

Threshold:

if $p \geq 0.5$, $\hat{y} = 1$;

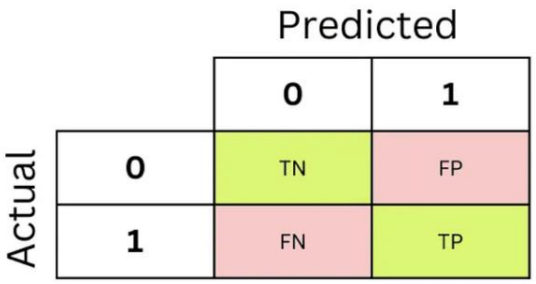
if $p < 0.5$, $\hat{y} = 0$

(1) Directly use the suite in Scikit-learn

```
Accuracy: 0.5733830845771144
Confusion matrix:
 [[922   7]
 [679   0]]
Report:
              precision    recall  f1-score   support

           0       0.58      0.99      0.73       929
           1       0.00      0.00      0.00       679

    accuracy                           0.57      1608
   macro avg       0.29      0.50      0.36      1608
weighted avg       0.33      0.57      0.42      1608
```



Logistic Regression defaults to "**maximizing overall accuracy.**" It finds that "always guessing 0" gives it 57% accuracy, so it simply stops guessing 1.

Meteorological data has strong spatial continuity, but it may be difficult to classify "missing vs. valid" based solely on latitude and longitude distribution.
We try to adjust the model in code: **dealing with class imbalance**.

```
model = LogisticRegression(class_weight="balanced")
```

Then,

```
Accuracy: 0.5472636815920398
Confusion matrix:
 [[503 426]
 [302 377]]
Report:
              precision    recall  f1-score   support

           0       0.62      0.54      0.58       929
           1       0.47      0.56      0.51       679

    accuracy                           0.55      1608
   macro avg       0.55      0.55      0.54      1608
weighted avg       0.56      0.55      0.55      1608
```

(2) Custom Loss

Binary Cross-Entropy Loss:

$$Loss = -y \ln h_\theta(x) - (1-y)\ln\left(1 - h_\theta(x)\right)$$

$$y \in \{0,1\},\ h_\theta(x) = \hat{p}$$

Gradient descent method:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta Loss$$

$$= \theta_n - \eta \left\{ \sum_i \frac{y_i - h_\theta(x_i)}{h_\theta(x_i)(1 - h_\theta(x_i))} \nabla_\theta h_\theta(x_i) \right\}$$

Through cross-entropy loss + gradient descent, θ_1, θ_2, and θ_0 are continuously adjusted, ultimately obtaining an optimal set of parameters.

1st try:

```
[001] Loss=nan | Train Acc=0.481 | Test Acc=0.750 | Confusion Matrix: [[TN=1206, FP=0], [FN=402, TP=0]]
[020] Loss=nan | Train Acc=0.519 | Test Acc=0.750 | Confusion Matrix: [[TN=1206, FP=0], [FN=402, TP=0]]
[040] Loss=nan | Train Acc=0.519 | Test Acc=0.750 | Confusion Matrix: [[TN=1206, FP=0], [FN=402, TP=0]]
```

Problem: $Loss = NaN$

Possible causes:

log(0)、sigmoid overflow、Learning rate too large…

Solution: change to use **log-sum-exp trick**, avoiding computing **log(sigmoid)**

2nd try:

```
[001] Loss=19.3766 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[020] Loss=17.9454 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[040] Loss=16.4388 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[060] Loss=14.9323 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[080] Loss=13.4257 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[100] Loss=11.9191 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[120] Loss=10.4125 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[140] Loss=8.9059 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[160] Loss=7.3994 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[180] Loss=5.8928 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
[200] Loss=4.3865 | Train Acc=0.481 | Test Acc=0.250 | Confusion Matrix: [[TN=0, FP=1206], [FN=0, TP=402]]
```

Loss decreases, but accuracy remains unchanged. This means the prediction probability is moving in the right direction, but has not yet crossed the threshold of 0.5, so the binarization result (二值化結果) is the same.

Logistic Regression is too weak.

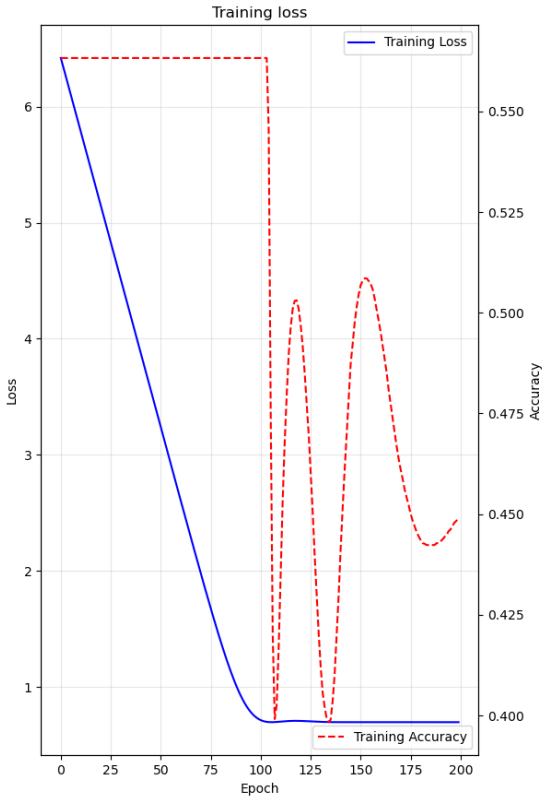
Essentially, it is a "linear decision boundary" (a slash). In this data, the "missing value distribution" on the map may not be cut by a straight line, and there may be large blocks and complex areas.

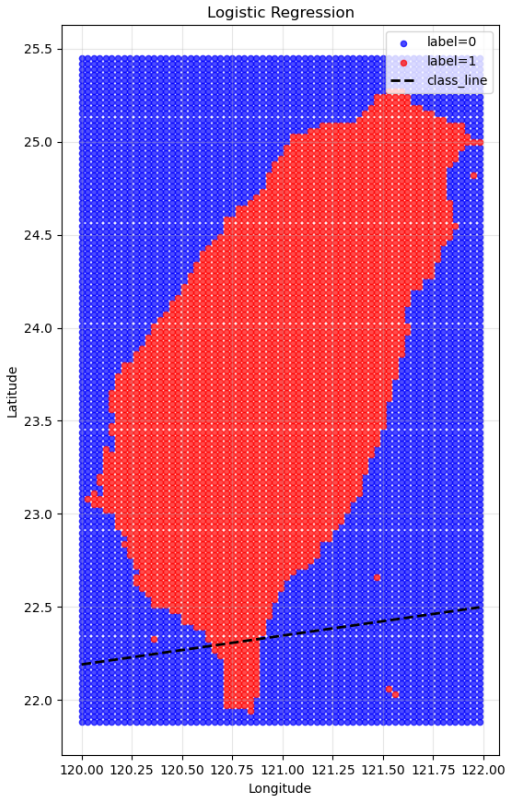
From the previous confusion matrix, we can see that the model achieves around 70% accuracy when choosing to “guess all 0s.” This suggests that 0s are more likely to learn patterns than 1s in the data, while the distribution of 1s may be scattered and irregular.

3rd try:

Inherited and used nn.module of Pytorch, which provides functions such as automatic gradient calculation and parameter management; loss function: BCE.

Function **nn.BCEWithLogitsLoss()** avoids numerical instability (when logits are large or small) More efficient than using sigmoid + BCELoss separately.






```
[001] Loss=6.4185 | Train Acc=0.563
[050] Loss=3.3013 | Train Acc=0.563
[100] Loss=0.7227 | Train Acc=0.563
[150] Loss=0.6965 | Train Acc=0.505
[200] Loss=0.6962 | Train Acc=0.449
```

Although the loss can be reduced to around 0.6, the prediction accuracy has decreased instead of increased. This is because accuracy is a step function that depends on whether the prediction probability crosses the 0.5 threshold. Even a slight parameter update, such as one that causes the prediction probability of certain points to fluctuate between 0.49 and 0.51, will cause the accuracy to fluctuate.

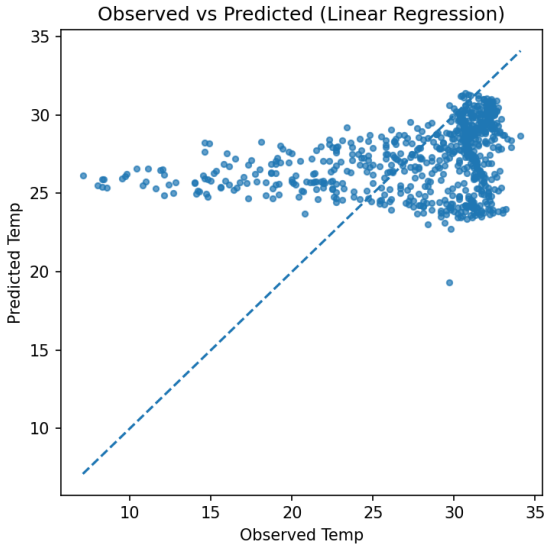
This is not a purely linearly separable data type, so a straight line cannot consistently separate the red and blue areas.

3. Regression model - Linear regression

$$h_\theta(x) = \hat{T} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

x_1: longitude, x_2: latitude

Result:



Bash line: Observed Temp = Predicted Temp **(perfect prediction)**

Most points do not follow the dotted line, and the predicted range is much narrower than the actual temperature (mostly around 24–31°C). When the temperature is low (x is small), the prediction is high; when the temperature is high (x is large), the prediction is low.

Typical “Mean Reversion” behavior of linear, low-feature models.

```
Coefficients of lon & lat : [-5.1195173   2.93689361]
Intercept : 576.9318
MAE : 4.0329
RMSE: 5.2543
R^2  : 0.0843
```

$$\hat{T} = 576.9 - 5.1x_1 + 2.9x_2$$

Machine Learning

313704071 陳安定 Assignment 5

1. Given

$$\boldsymbol{f(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}}$$

Where $x, \mu \in \mathbb{R}^k, \Sigma$ is a k-by-k positive definite matrix and $|\Sigma|$ is its determinant. Show that $\int_{\mathbb{R}^k} f(x)dx = 1$.

Proof:

Σ can be transformed into $\Sigma = \mathrm{Q}\Lambda\mathrm{Q}^{\mathrm{T}}$, where Q is k-by-k orthogonal matrix, Λ is k-by-k diagonal matrix. Let $y = x - \mu$, then $(x-\mu)^T\Sigma^{-1}(x-\mu) = y^T\mathrm{Q}\Lambda^{-1}\mathrm{Q}^{\mathrm{T}}y$. And let $u = \mathrm{Q}^{\mathrm{T}}y$, so $(x-\mu)^T\Sigma^{-1}(x-\mu) = u^T\Lambda^{-1}u$.

Continuously, let $z = \Lambda^{-\frac{1}{2}}u$, $u = \Lambda^{\frac{1}{2}}z$, then $u^T\Lambda^{-1}u = \|z\|^2$.

Since $z = \Lambda^{-\frac{1}{2}}\mathrm{Q}^{\mathrm{T}}(x-\mu)$, $\frac{dz}{dx} = \left|\Lambda^{-\frac{1}{2}}\mathrm{Q}^{\mathrm{T}}\right|$, $dx = \left|\Lambda^{\frac{1}{2}}\mathrm{Q}^{\mathrm{T}}\right| dz = \sqrt{|\Sigma|}\, dz$

Then,

$$f(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} e^{-\frac{1}{2}\|z\|^2},$$

$$\int f(x)dx = \int \frac{1}{\sqrt{(2\pi)^k}} e^{-\frac{1}{2}\|z\|^2} dz = 1$$

Notice that $\frac{1}{\sqrt{(2\pi)^k}} e^{-\frac{1}{2}\|z\|^2}$ is the density of k independent $N(0,1)$variables, hence its integral is equal to 1.

2. Let A, B be n-by-n matrices and x be a n-by-1 vector.

(a) Show that $\frac{\partial}{\partial A}\boldsymbol{trace}(\boldsymbol{AB}) = \boldsymbol{B^T}$.

Assume A, B is 2-by-2 matrices, $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $B = \begin{bmatrix} p & q \\ r & s \end{bmatrix}$

$AB = \begin{bmatrix} ap + br & \# \\ \# & cq + ds \end{bmatrix}$, $trace(AB) = \boldsymbol{a}p + \boldsymbol{b}r + \boldsymbol{c}q + \boldsymbol{d}s$.

and derived by every element of A (a, b, c, d) respectively, then it will be

$\frac{\partial}{\partial A} trace(AB) = \begin{bmatrix} p & r \\ q & s \end{bmatrix}$, which is B^T.

$trace(AB)_{n\times n} = \sum_i (AB)_{ii} = \sum_i \sum_j A_{ij} B_{ji}$, and derive it by A_{kl},

The result will be $B_{lk} = B^{-1}$.

(b) Show that $\boldsymbol{x^T Ax} = \boldsymbol{trace}(\boldsymbol{xx^T A})$.

$x^T Ax$ is **scalar**, so $x^T Ax = trace(x^T Ax)$,

by $trace(AB) = trace(BA)$,

$trace(x^T Ax) = trace(Axx^T) = trace(Axx^T)$.

(c) Derive the maximum likelihood estimators for a multivariate Gaussian.

$L(\mu, \Sigma) = \prod_i^n f(x) = \left(\frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}}\right)^N e^{-\frac{1}{2}\sum (x-\mu)^T \Sigma^{-1}(x-\mu)}$,

$\ln L(\mu, \Sigma) = -\frac{\mathrm{nk}}{2}\ln(2\pi) - \frac{n}{2}\ln|\Sigma| - \frac{1}{2}\sum (x_i - \mu)^T \Sigma^{-1}(x_i - \mu)$,

$\frac{\partial \ln L(\mu,\Sigma)}{\partial \mu} = -\frac{1}{2}\Sigma^{-1}\left(-2(\sum x_i - n\mu)\right) = 0$, $\boxed{\widehat{\boldsymbol{\mu}} = \frac{1}{n}\sum x_i.}$

$\frac{\partial \ln L(\mu,\Sigma)}{\partial \Sigma}$, first compute $\frac{d}{d\Sigma}\ln|\Sigma|$, $\frac{d}{d\Sigma}\ln|\Sigma| = (\Sigma^{-1})^T = \Sigma^{-1}$,

$\frac{\partial \ln L(\mu,\Sigma)}{\partial \Sigma} = -\frac{n}{2}\Sigma^{-1} + \frac{1}{2}\Sigma^{-2}\sum (x_i - \mu)^T (x_i - \mu) = 0$,

$-\frac{n}{2} + \frac{1}{2}\Sigma^{-1}\sum (x_i - \mu)^T (x_i - \mu) = 0$, $\boxed{\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum (x_i - \mu)^T (x_i - \mu)}$.

3. Unanswered Questions

上一個作業：台灣氣候資料分類問題，很難用一條線去區分有無資料，因為台灣是有許多離島且外圍是一個番薯形狀，那我的問題是，假設圖畫出來，有資料的地方僅僅是一個圓，包圍的部分就是無資料，這樣又如何去分類呢? 問題會變得比較簡單嗎?

還是最大的問題是在離島分布不規則的狀況?

Machine Learning

313704071_Assignment 6.

1. Classification using GDA

From the assignment 4, we know that since the dataset includes outlying islands and the boundary between the two categories (0 and 1) is irregular—where class 1 is surrounded by class 0—the Taiwan climate dataset is **a linearly inseparable problem**, making LDA unsuitable. Taiwan's main island has a sweet-potato-like shape; therefore, we apply Quadratic Discriminant Analysis (QDA), where the decision boundary is a quadratic surface, and maximum likelihood estimation (MLE) is used for parameter estimation.

Hypothesis: for any class $k \in \{0,1\}$,

$$x \mid y = k \sim N(\mu_k, \Sigma_k) \text{ , where}$$

μ_k: mean, Σ_k: covariance matrix.

QDA allows $\Sigma_0 \neq \Sigma_1$, so the decision boundary is quadratic curve.

By Bayes' Theorem, $P(y = k \mid x) \propto P(x \mid y = k)\pi_k$, where $\pi_k = P(y = k)$.|

Hence, use likelihood function,

$$L(\theta) = P(x \mid y = k)\pi_k = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)} \pi_k$$

$$\ln L(\theta) = \sum_{i=1}^{n} \left[-\frac{1}{2} \ln|\Sigma_k| - \frac{1}{2}(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k) + \ln\pi_k \right] + C$$

$$\ln L(\theta)_k = \delta_k(\theta) = \sum_{i=1}^{n} \left[-\frac{1}{2} \ln|\Sigma_k| - \frac{1}{2}(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k) \right] + n_k \ln\pi_k$$

$$\theta^* = arg \max_{\theta} \delta_k(\theta)$$

We know that by maximum likelihood estimation (MLE), for each class k,

$$\hat{\mu} = \frac{1}{n}\sum x_i, \ \hat{\Sigma} = \frac{1}{n}\sum (x_i - \mu)^T (x_i - \mu)$$

There are three steps in the prediction process:

(1) **Split** the data into training and test sets;

(2) **Fit** the training set to estimate the parameters for each class (μ, Σ, π).

(3) **Predict** – Given a test sample x, determine the optimal parameters(θ^*) and

output the predicted value y.

Performance index: Confusion Matrix

TN: prediction 0 when y is 0; **FP**: prediction 1 when y is 0;

FN: prediction 0 when y is 1; **TP**: prediction 1 when y is 1.

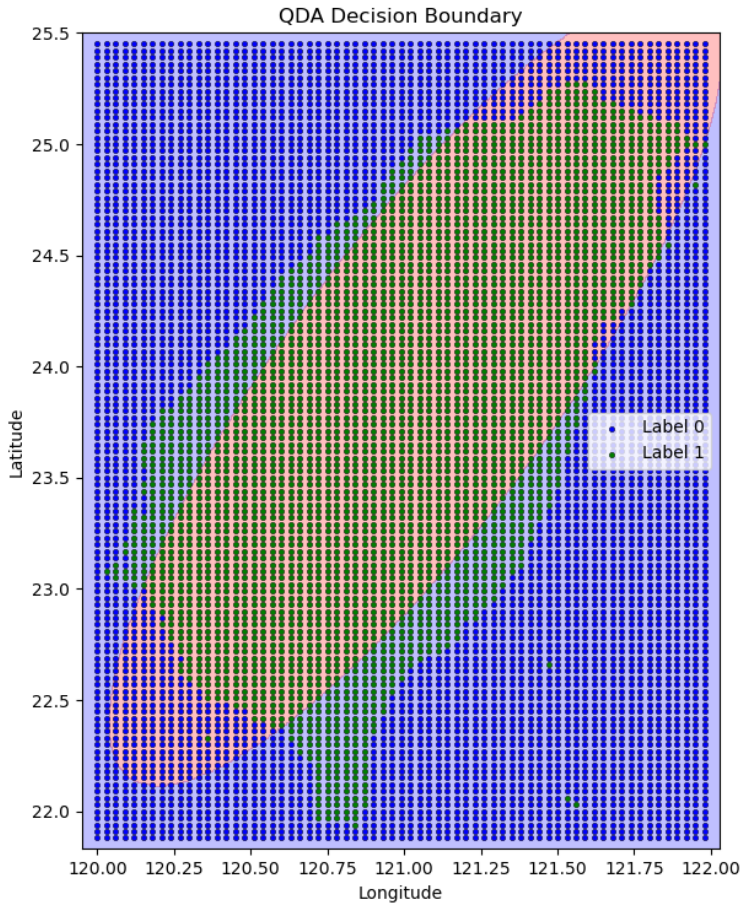
$$Accuracy = \frac{TN + TP}{sum\ of\ y_{test}}$$

```
Accuracy : 0.839
Precision: 0.839
Recall   : 0.778
F1-score : 0.807
Confusion Matrix: [[TN=806, FP=104], [FN=155, TP=543]]
```

The other three index respectively is precision, recall, and F1-score, where

$Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, $F1\_score = 2 \times \frac{Precision \times Recall}{Precision+Recall}$.

Decision boundary: using QDA.



2. Regression – piecewise smooth function

Combine the two models from Assignment 4 into a single function.

$$h(\vec{x}) = \begin{cases} R(\vec{x}), & if\ C(\vec{x}) = 1 \\ -999, & if\ C(\vec{x}) = 0 \end{cases}.$$

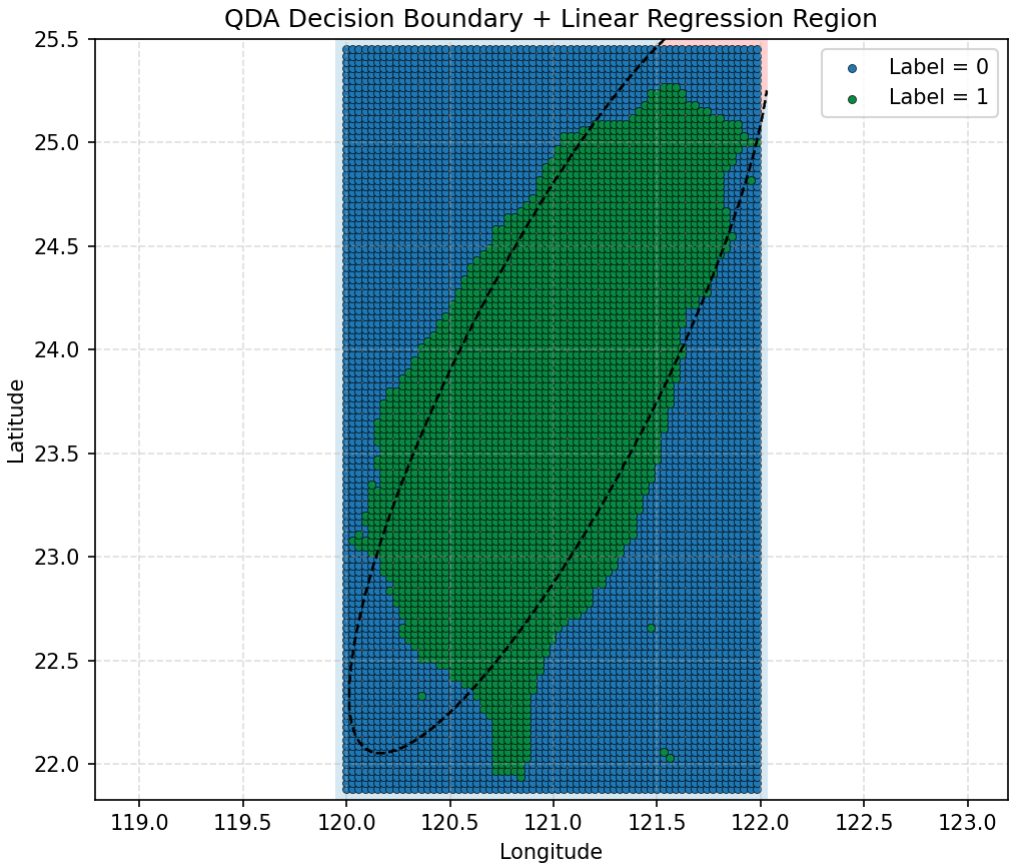
$C(\vec{x})$: classification model, $R(\vec{x})$: regression model.

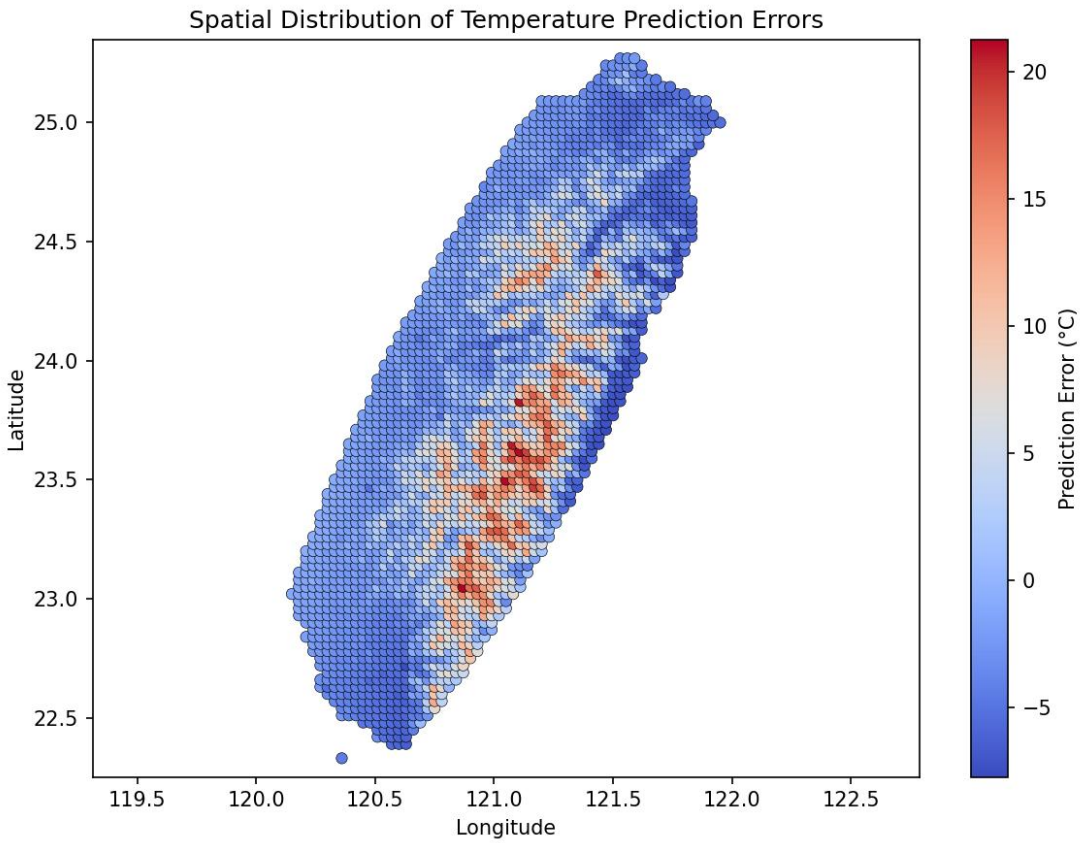
Input: labels.csv, values.csv

Output*:* Prediction that whether having data using $C(\vec{x})$，if yes (label_pred=1)，continue to predict temperature value using $R(\vec{x})$，Finally, output the predicted values of the temperature of these points.

Brief explanation:

This problem doesn't involve **data splitting**, as the goal isn't generalization evaluation. Instead, we ensure that the classification boundaries are intact, telling the regression model which locations are worth predicting temperature for and which should be skipped. Include plots or tables that demonstrate the behavior of your model. Next, we use linear regression to predict temperature based on latitude and longitude.





Simple linear regression:

$$Temperature = \beta_0 + \beta_1(lon) + \beta_2(lat)$$

This regression assumes that the relationship between temperature and (lon, lat) is roughly linear, where β_1 represents the temperature change for every 0.03 degrees eastward, and β_2 represents the temperature change for every 0.03 degrees northward.

Result:

$$Temperature = 566.540 - 4.999(lon) + 2.767(lat)$$

The equation suggests that temperature decreases eastward and increases northward. However, this contradicts the general expectation that temperatures decrease as latitude increases.

Error in prediction: MSE

As shown in **figure** above, red dots represent locations with larger error values. It's not hard to see that the pattern of red dots resembles the distribution of mountain ranges in Taiwan. Because the error is the difference between the predicted value and the actual value, this result is reasonable.

In the future, nonlinear relationships should be considered, for example using polynomial regression.

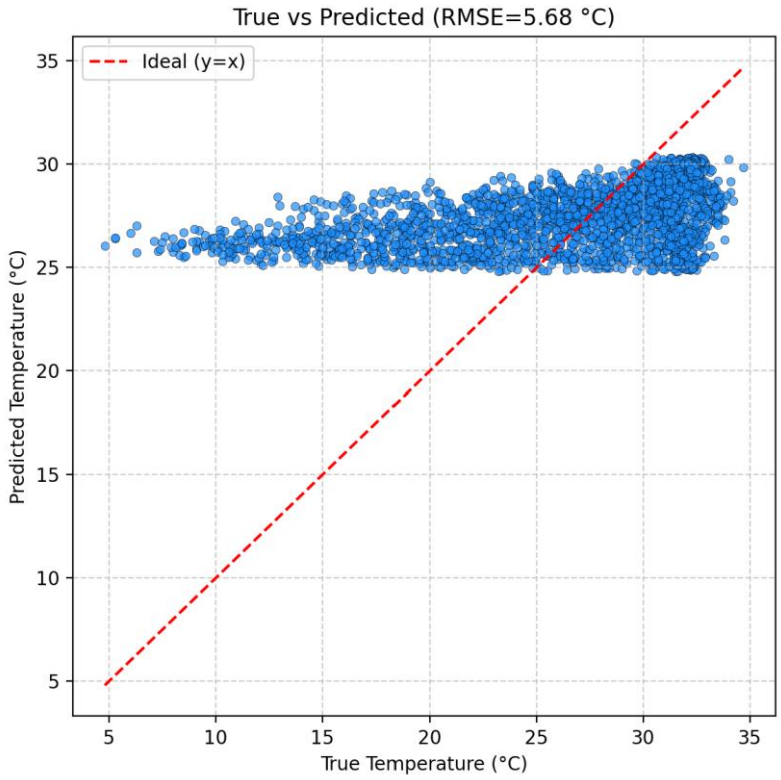
```
=== Linear Regression Parameters ===
Equation: y = 566.540 + -4.999·lon + 2.767·lat

=== Classification (QDA) ===
Accuracy : 0.836 | Precision : 0.828 | Recall : 0.785 | F1 : 0.806
Confusion Matrix: [[TN=3974, FP=571], [FN=751, TP=2744]]

=== Regression (Linear) ===
MSE=32.2723 | RMSE=5.6809 | MAE=4.2995 | N=2744
```

Scatter plot:

Red line: true temp = predicted temp



The performance of prediction model is not good enough.

Machine Learning

313704071 _HW7

1. **Explain the concept of score matching and describe how it is used in score-based (diffusion) generative models.**

Because it is difficult to learn the probability density function of a data distribution which is one of the goals of a generative model, we try to learn the score function.

$$S(x) \approx \nabla_x \log p(x)$$

Score matching minimizes the difference between the model's score $s_\theta(x)$ and the true data score:

$$L_{ESM}(\theta) = \mathbb{E}_{x\sim p(x)} \|S(x;\theta) - \nabla_x \log p(x)\|^2$$

$\mathbb{E}_{x\sim p(x)}$: expectation of sample data.

Since the true score $\nabla_x \log p(x;\theta)$ is usually intractable, **implicit forms** (via integration by parts) or **denoising versions** are used to make it computable.

$$L_{ISM}(\theta) = \mathbb{E}_{x\sim p(x)}[\|S(x;\theta)\|^2 + 2\nabla_x \cdot S(x;\theta)]$$

Using derivation, we know that

$$\mathbb{E}_{x\sim p(x)} \|S(x;\theta) - \nabla_x \log p(x;\theta)\|$$
$$= \mathbb{E}_{x\sim p(x)}[\|S(x;\theta)\|^2 + 2\nabla_x \cdot S(x;\theta)] + \mathbb{E}_{x\sim p(x)}[\|\nabla_x \log p(x)\|^2]$$

Hence, $$arg\min_\theta L_{ESM} = arg\min_\theta L_{ISM}$$

Denoising score matching (DSM)

Instead of modeling $p(x)$ directly, we model how data changes under noise and learn the score function of the *noisy data distributions* at different noise levels.

Let x_0: original data, $p_0(x)$: original data

x: perturbed data, and $x = x_0 + \epsilon_\sigma$; $p(x)$: pdf of perturbed data.

$$L_{DSM}(\theta) = \mathbb{E}_{x_0\sim p_0(x)} \mathbb{E}_{(x|x_0)\sim p(x|x_0)} \|S_\sigma(x;\theta) - \nabla_x \log p(x|x_0)\|^2$$

where, $p(x|x_0) = \frac{1}{(2\pi)^{d/2}\sigma^d} e^{-\left(\frac{\|x-x_0\|}{2\sigma^2}\right)^2}$

$$\nabla_x \log p(x|x_0) = -\frac{1}{\sigma^2}(x - x_0)$$

Add Gaussian noise to the data:

$x = x_0 + \sigma\epsilon,\ \epsilon \in N(0, I)$

$\quad = x_0 + \epsilon_\sigma,\ \epsilon_\sigma \in N(0, \sigma^2 I)$

$$L_{DSM}(\theta) = \mathbb{E}_{x_0 \sim p_0(x)} \mathbb{E}_{(x|x_0) \sim p(x|x_0)} \left\| S_\sigma(x;\theta) + \frac{1}{\sigma^2}(x - x_0) \right\|^2$$

$$= \mathbb{E}_{x_0 \sim p_0(x)} \mathbb{E}_{(x|x_0) \sim p(x|x_0)} \frac{1}{\sigma^2} \|\sigma S_\sigma(x;\theta) + \epsilon\|^2$$

We can conclude that, unlike ESM, DSM does not require the true gradient of the data log-density, and unlike ISM, it avoids complex integration by parts by formulating a computable "denoising" objective. It allows the model to efficiently learn stable score functions across multiple noise levels and has become the foundation of score-based (diffusion) generative models, where the learned scores guide the reverse process that gradually transforms pure noise back into realistic data.

2. Unanswered questions

將資料加噪，訓練 score function 去近似 $\nabla_x \log p(x)$，最後再反向擴散把純噪聲逐步還原成資料。那 DSM 主要有什麼用途? 我能想到的是通常衛星遙測拍攝的相片都會相當模糊，但透過一些除雜訊的技術，就能得到相對清晰的地球表面影像。而 GPT 告訴我更多應用，像是 MRI 影像復原、語音生成、文字生成、分子結構生成等等…

Machine Learning

313704071_HW8

1. Show that the sliced score matching (SSM) loss can also be written as

$$\boldsymbol{L_{SSM}} = \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p}(\boldsymbol{x})}\mathbb{E}_{\boldsymbol{v}\sim\boldsymbol{p}(\boldsymbol{v})}\left[\|\boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{S}(\boldsymbol{x};\boldsymbol{\theta})\|^2 + \boldsymbol{2v}^{\boldsymbol{T}}\boldsymbol{\nabla}_{\boldsymbol{x}}\left(\boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{S}(\boldsymbol{x};\boldsymbol{\theta})\right)\right]$$

We know that $\nabla \cdot S(x;\theta) = trace(\nabla S)$.

Hutchinson's trace estimator:

Let $v \in \mathbb{R}^d$ is a random vector, such that $E[vv^T] = 1$, then

$$tr(A) = \mathbb{E}_{v\sim p(v)}(v^T A v)$$

We also know that

$L_{ISM}(\theta) = \mathbb{E}_{x\sim p(x)}[\|S(x;\theta)\|^2 + 2\nabla_x \cdot S(x;\theta)]$

Hence, the $\nabla_x \cdot S(x;\theta)$ in ISM loss can be rewritten as

$$\nabla_x \cdot S(x;\theta) = trace(\nabla S) = \mathbb{E}_{v\sim p(v)}(v^T(\nabla S)v) = \mathbb{E}_{v\sim p(v)}(v^T\nabla(vS))$$

Then, combine it with ISM loss and become SSM loss.

$$\begin{aligned} L_{SSM} &= \mathbb{E}_{x\sim p(x)}\|S(x;\theta)\|^2 + \mathbb{E}_{x\sim p(x)}\mathbb{E}_{v\sim p(v)}\left(2v^T\nabla(vS)\right) \\ &= \mathbb{E}_{x\sim p(x)}\mathbb{E}_{v\sim p(v)}\left[\|v^T S(x;\theta)\|^2 + 2v^T\nabla_x\left(v^T S(x;\theta)\right)\right] \end{aligned}$$

2. Briefly explain SDE

The Stochastic Differential Equation:

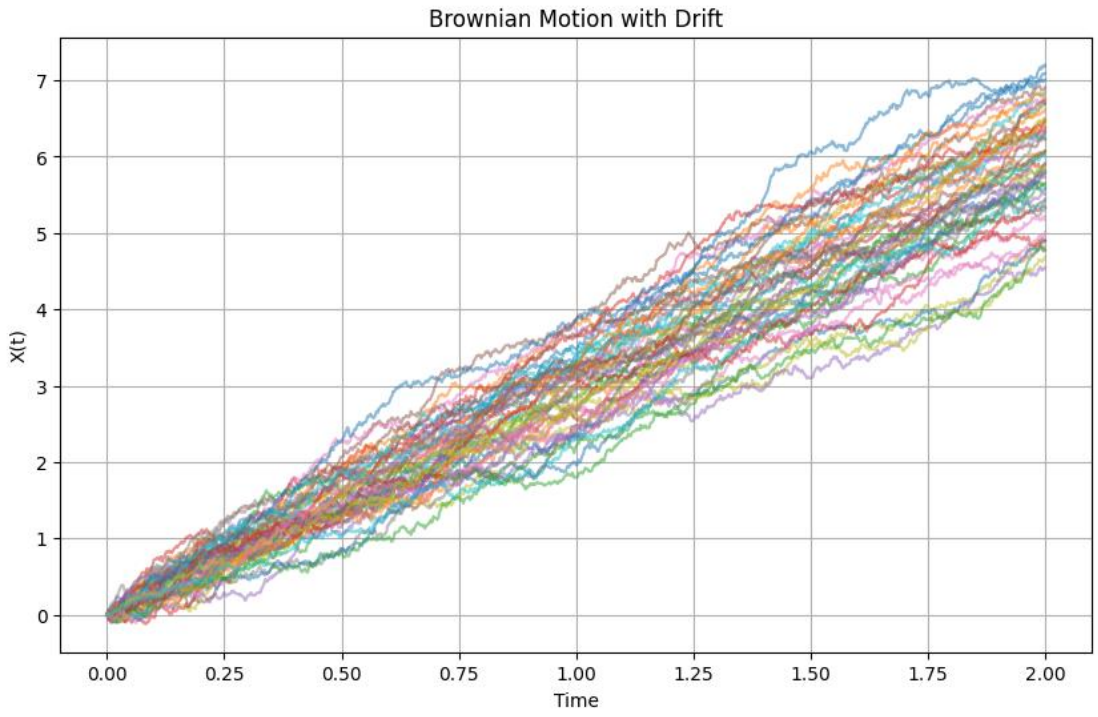
$$dX_t = f(X_t,t)dt + G(X_t,t)dW_t$$
$$X(0) = X_0$$

where X_t is stochastic process, W_t is standard Brownian motion.

The forward SDE is a process of adding noise, and drift part $f(X_t,t)$ control the average trend, on the other hand, diffusion part $G(X_t,t)$ control the strength of noise.

The figure shows sample paths of a Brownian motion with drift. For any fixed time t, the random variable $X(t)$ is normally distributed with mean μt and variance $\sigma^2 t$.

The empirical histogram across paths at time t approximates this PDF.



3. Unanswered Questions

(1)為什麼需要 $f(X_t, t)$?

GPT: 沒有 f 就沒有系統性的趨勢；只會靠雜訊擴散、均值不動、難以把系統帶往指定區域。

(2)加噪的起點可以不同?

GPT: 影像加噪（同一張圖內）：每個像素/通道的起點就是它自己的像素值 $x_0[i]$（彼此通常不同），然後各自加上獨立高斯噪聲沿時間演化。所以起點不同是正常的。