114-1 Machine Learning
Week 4 Assignment

Ming Hsun Wu

September 30, 2025

Problem 1. Unanswered Questions

Nope!

Problem 2.

(a) Classification Dataset

Format: (Longitude, Latitude, Label)

Rules:

- If the temperature observation value is an invalid value of -999, then label $= 0$.
- If the temperature observation value is valid, then label $= 1$.

(b) Regression Dataset

Format: (Longitude, Latitude, Value)

Rules:

- Only valid temperature observation values are retained (all values of -999 are removed).
- The Value corresponds to the temperature in degrees Celsius.

Note of Problem 2.

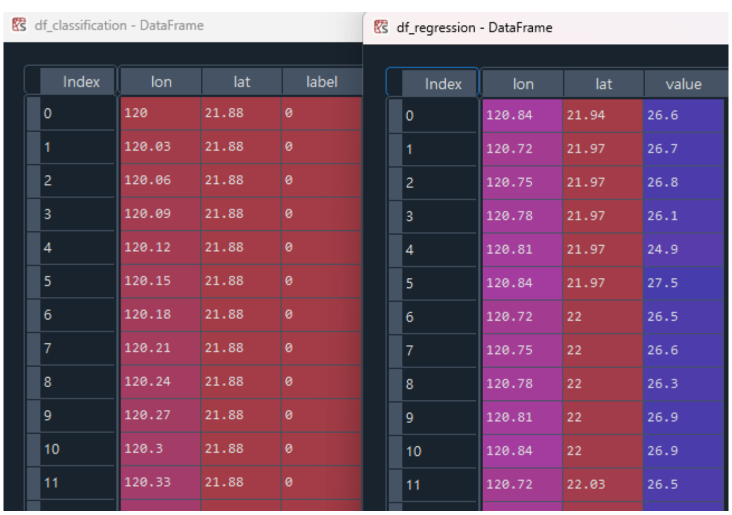
The raw data comes from gridded observations provided by the Central Weather Bureau, consisting of 67×120 grid points. Invalid values are marked as -999.0.

Two datasets were constructed:

- **Classification dataset:** $(\text{lon}, \text{lat}, \text{label})$, where $\text{label} = 0$ represents invalid points and $\text{label} = 1$ represents valid points.
- **Regression dataset:** $(\text{lon}, \text{lat}, \text{value})$, retaining only valid points, where *value* corresponds to the temperature in degrees Celsius.

After processing:

- The classification dataset contains $8,040$ entries.
- The regression dataset contains $3,495$ entries.



df_classification - DataFrame

Index	lon	lat	label
0	120	21.88	0
1	120.03	21.88	0
2	120.06	21.88	0
3	120.09	21.88	0
4	120.12	21.88	0
5	120.15	21.88	0
6	120.18	21.88	0
7	120.21	21.88	0
8	120.24	21.88	0
9	120.27	21.88	0
10	120.3	21.88	0
11	120.33	21.88	0

df_regression - DataFrame

Index	lon	lat	value
0	120.84	21.94	26.6
1	120.72	21.97	26.7
2	120.75	21.97	26.8
3	120.78	21.97	26.1
4	120.81	21.97	24.9
5	120.84	21.97	27.5
6	120.72	22	26.5
7	120.75	22	26.6
8	120.78	22	26.3
9	120.81	22	26.9
10	120.84	22	26.9
11	120.72	22.03	26.5

Problem 3.

Model Training

Using the two datasets prepared in problem 2, two simple machine learning models were trained separately:

- **Classification model:** Predicts whether a grid point is valid (1) or invalid (0) based on $(\text{longitude}, \text{latitude})$.
- **Regression model:** Predicts the corresponding temperature observation value based on $(\text{longitude}, \text{latitude})$.

NOTE OF PROBLEM 3.

Model design

(a) Classification Model

Input: (lon, lat)

Output: Binary classification

Architecture: A fully connected neural network (MLP) with 2 input features, followed by 4 hidden layers $(128, 128, 64, 32)$ units. Each layer uses ReLU activation, combined with Batch Normalization and Dropout (0.3). The output layer is 2-dimensional with softmax activation.

Loss function: CrossEntropyLoss

Optimizer: Adam ($lr = 0.001$)

(b) Regression Model

Input: (lon, lat)

Output: Temperature value (°C)

Architecture: A fully connected neural network with 5 hidden layers $(64, 128, 256, 128, 32)$ units. Each layer uses ReLU activation. The final output is a single real-valued unit.

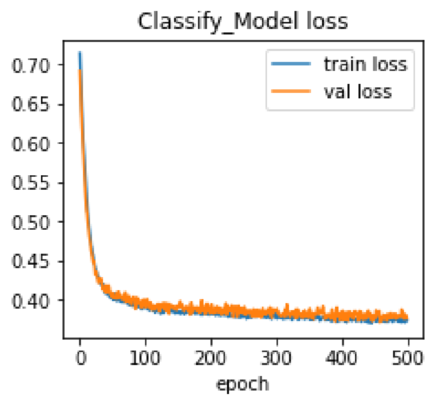
Loss function: MSELoss

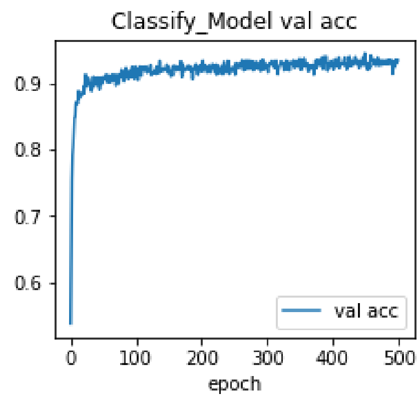
Optimizer: Adam ($lr = 0.001$)

Training Process and Results

(a) Classification Model Results

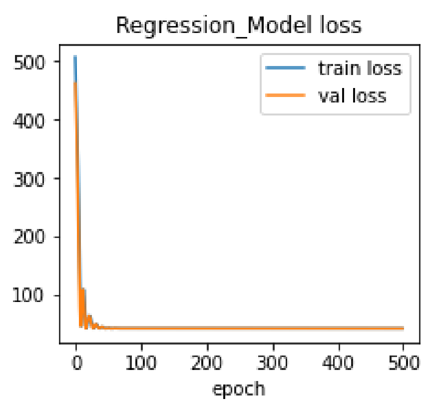
During training, the loss gradually converged, and the validation accuracy reached above 0.9 (as shown in the figure). The trained classification model achieved a prediction accuracy of approximately 0.84 on the test set.





(b) Regression Model Results

During training, the MSE decreased from around 500 to double digits. The trained regression model achieved a mean squared error of 40.7 on the test set when comparing predictions with the ground truth.



Discussion and Analysis

(a) Classification Model

At the beginning, I designed the model in a relatively simple way, but the training results were very poor (the predictions on the test set all defaulted to class 0). Therefore, I adjusted the model architecture and depth, which led to the current results. I suspect the poor initial performance was mainly due to limitations and deficiencies inherent in the dataset. Despite the data imbalance, the model still maintained high accuracy, suggesting that it has learned the decision boundary effectively.

(b) Regression Model

I believe the current regression model results are unsatisfactory. On the test set, the predictions are mostly concentrated around 21. While the training loss did converge, the model seems to have simply found a constant value that minimizes the loss, rather than capturing the true variation in the data.

```python
#%% source code
import json
import re
import numpy as np
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
#%%
'load data'
path = r"./O-A0038-003.json"
with open(path, "r", encoding="utf-8") as f:
    data = json.load(f)

content = data['cwaopendata']["dataset"]["Resource"]["Content"]

nums = re.findall(r'[-+]?\d+\.\d+E[+-]\d+', content)
vals = np.array([float(x) for x in nums], dtype=float)


ncols, nrows = 67, 120
grid = vals.reshape((nrows, ncols))
```

```
bl_lon, bl_lat = 120.0, 21.88
res_lon, res_lat = 0.03, 0.03


lon = bl_lon + np.arange(ncols) * res_lon
lat = bl_lat + np.arange(nrows) * res_lat
lon_grid, lat_grid = np.meshgrid(lon, lat)


lon_flat = lon_grid.ravel()
lat_flat = lat_grid.ravel()
val_flat = grid.ravel()


labels = np.where(val_flat == -999.0, 0, 1)
df_classification = pd.DataFrame({"lon": lon_flat, "lat":
    lat_flat, "label": labels})


mask = val_flat != -999.0
df_regression = pd.DataFrame({
    "lon": lon_flat[mask],
    "lat": lat_flat[mask],
    "value": val_flat[mask]
})

```

```

#%%
'data preprocessing'
x_cls = df_classification[["lon", "lat"]].values
y_cls = df_classification["label"].values

x_cls_train, x_cls_test, y_cls_train, y_cls_test =
    train_test_split(x_cls, y_cls, test_size=0.3, random_state=40)
x_cls_val, x_cls_test, y_cls_val, y_cls_test =
    train_test_split(x_cls_test, y_cls_test, test_size=0.3,
    random_state=40)



x_cls_train = torch.tensor(x_cls_train, dtype=torch.float32)
y_cls_train = torch.tensor(y_cls_train, dtype=torch.long)

x_cls_val = torch.tensor(x_cls_val, dtype=torch.float32)
y_cls_val = torch.tensor(y_cls_val, dtype=torch.long)

x_cls_test = torch.tensor(x_cls_test, dtype=torch.float32)
y_cls_test = torch.tensor(y_cls_test, dtype=torch.long)



x_reg = df_regression[["lon", "lat"]].values
y_reg = df_regression["value"].values
```

```python

x_reg_train, x_reg_test, y_reg_train, y_reg_test = train_test_split(x_reg, y_reg, test_size=0.3, random_state=42)
x_reg_val, x_reg_test, y_reg_val, y_reg_test = train_test_split(x_reg, y_reg, test_size=0.3, random_state=42)



x_reg_train = torch.tensor(x_reg_train, dtype=torch.float32)
y_reg_train = torch.tensor(y_reg_train, dtype=torch.float32).view(-1,1)
x_reg_val = torch.tensor(x_reg_val, dtype=torch.float32)
y_reg_val = torch.tensor(y_reg_val, dtype=torch.float32).view(-1,1)
x_reg_test = torch.tensor(x_reg_test, dtype=torch.float32)
y_reg_test = torch.tensor(y_reg_test, dtype=torch.float32).view(-1,1)

#%%
'Classify_Model Training'
# compile model
class Classify_Model(nn.Module):
    def __init__(self):
        super(Classify_Model, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(2, 128),
            nn.ReLU(),
            nn.BatchNorm1d(128),
```

```
            nn.Dropout(0.3),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.BatchNorm1d(128),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 2),
            nn.Softmax()
        )
    def forward(self, x):
        return self.layers(x)

Classify_Model = Classify_Model()
optimizer_cls = optim.Adam(Classify_Model.parameters(), lr=0.001)

# train model
epochs = 500
cls_train_losses, cls_val_losses, cls_val_acc = [], [], []
for epoch in range(epochs):
    Classify_Model.train()
    optimizer_cls.zero_grad()
    outputs = Classify_Model(x_cls_train)
    loss = nn.CrossEntropyLoss()(outputs, y_cls_train)
    cls_train_losses.append(loss.item())
    loss.backward()
    optimizer_cls.step()
```

```python

    with torch.no_grad():
        y_pred_val = Classify_Model(x_cls_val)
        val_loss = nn.CrossEntropyLoss()(y_pred_val, y_cls_val)
        cls_val_losses.append(val_loss.item())

        val_acc = accuracy_score(y_cls_val, y_pred_val.argmax(dim=1).numpy())
        cls_val_acc.append(val_acc)
# evaluate model
Classify_Model.eval()
with torch.no_grad():
    Classify_preds = Classify_Model(x_cls_test).argmax(dim=1).numpy()
acc = accuracy_score(y_cls_test, Classify_preds)
print("Classification accuracy:", acc)

#%%
'Regression_Model Training'
# compile model
class Regression_Model(nn.Module):
    def __init__(self):
        super(Regression_Model, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(2, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
```

```
            nn.ReLU(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 32),
            nn.ReLU(),
            nn.Linear(32, 1)


        )
    def forward(self, x):
        return self.layers(x)

Regression_Model = Regression_Model()
criterion_reg = nn.MSELoss()
optimizer_reg = optim.Adam(Regression_Model.parameters(), lr=0.001)

# train model
reg_train_losses, reg_val_losses = [], []
epochs = 500
for epoch in range(500):
    Regression_Model.train()
    optimizer_reg.zero_grad()
    outputs = Regression_Model(x_reg_train)
    loss = criterion_reg(outputs, y_reg_train)
    reg_train_losses.append(loss.item())
```

```python
    loss.backward()
    optimizer_reg.step()

    with torch.no_grad():
        y_pred_val = Regression_Model(x_reg_val)
        val_loss = nn.MSELoss()(y_pred_val, y_reg_val)
        reg_val_losses.append(val_loss.item())


# evaluate model
Regression_Model.eval()
with torch.no_grad():
    Regression_preds = Regression_Model(x_reg_test).numpy()
mse = mean_squared_error(y_reg_test, Regression_preds)
print("Regression MSE:", mse)
```