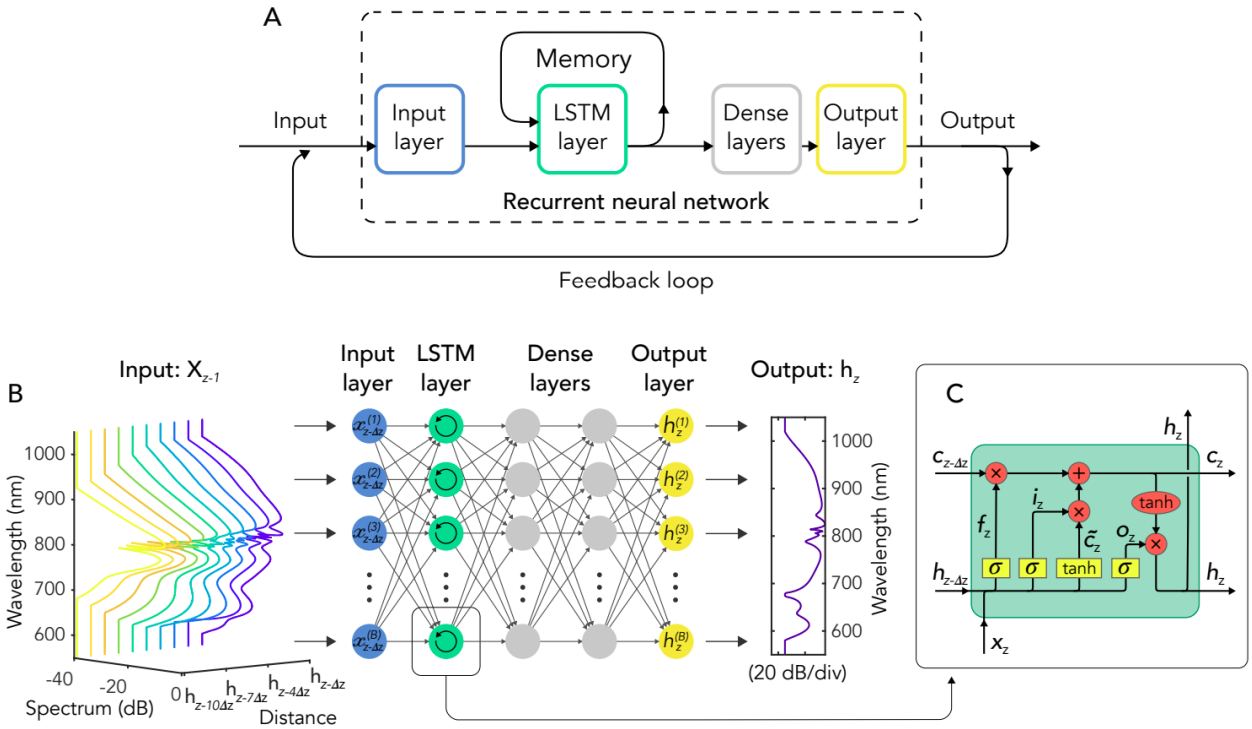
AI VIET NAM – COURSE 2024

Recurrent Neural Networks - Exercise

Khoa Tho-Anh Nguyen, Nha Dang Nguyen, Thang Dinh Duong

Ngày 14 tháng 12 năm 2024

Phần I: Giới thiệu



Data Cho đến nay, các kiến thức mà chúng ta đã học về các mô hình Machine Learning, chẳng hạn như Linear Regression, Logistic Regression, v.v., thường được áp dụng cho dữ liệu dạng bảng (tabular data), trong đó mỗi hàng đại diện cho một mẫu (sample) và mỗi cột đại diện cho một thuộc tính (attribute). Khi chuyển sang các mô hình như Multi-Layer Perceptron (MLP) hoặc Convolutional Neural Network (CNN), chúng ta làm việc với dữ liệu ảnh (image data), nơi đầu vào (input) là các pixel biểu diễn giá trị tại từng tọa độ trên hình ảnh.

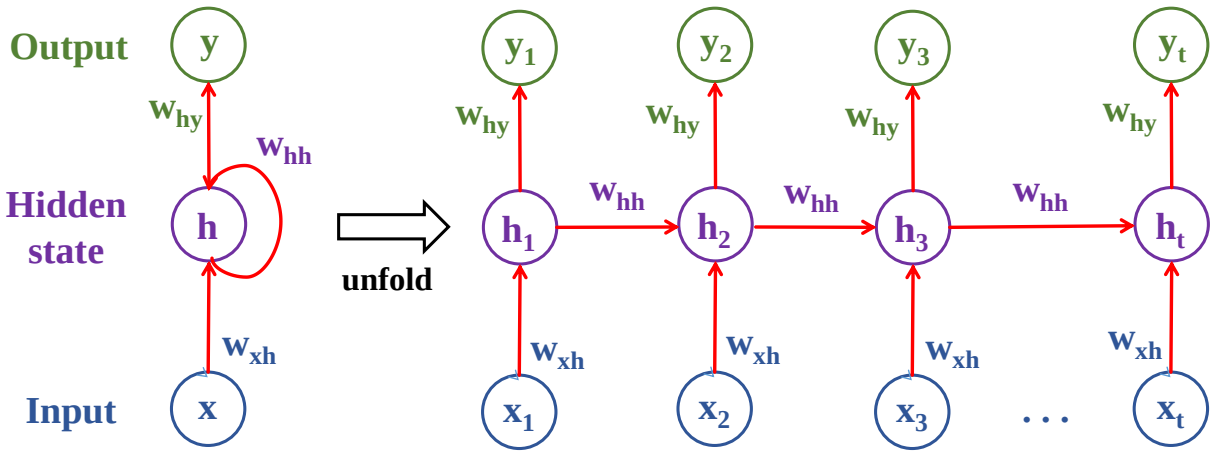
Điểm chung giữa dữ liệu dạng bảng và dữ liệu ảnh là chúng đều có số lượng đặc trưng (features) rõ ràng và cụ thể, chẳng hạn như số lượng thuộc tính trong dữ liệu bảng hoặc số lượng pixel của một bức ảnh (ví dụ: MNIST dataset chứa các hình ảnh có kích thước 28 x 28 pixel). Đồng thời, các mô hình được thiết kế cho các tác vụ regression hoặc classification cho những dữ liệu trên thường nhận input là các

features của một sample data và đưa ra dự đoán là một output đơn lẻ (ví dụ: phân loại hình ảnh chó hoặc mèo, dự đoán giá nhà, chẩn đoán bệnh, v.v.).
Tuy nhiên, có rất nhiều tác vụ học khác mà chúng ta sẽ phải làm việc với các data có độ dài linh hoạt như text, audio, time series. Đồng thời các models thiết kế cho các tác vụ trên cũng yêu cầu sau khi nhận vào input là sequential data, output dự đoán cũng sẽ có dạng sequential. Sẽ có nhiều sự thác mắc về việc như thế nào là data có fixed length và flexible length, khi mà ảnh cũng sẽ có nhiều ảnh lớn nhỏ, tabular data cũng sẽ có nhiều bảng với số lượng attribute khác nhau. Để hiểu rõ, chúng ta cần phân rõ sự khác biệt giữa chúng:

- **Image data:** Ảnh có rất nhiều size khác nhau như 28x28, 32x32, 128x128, ... nhưng nhìn chung chúng ta đều có thể resize về một size nhất định phù hợp với cấu trúc mô hình.
- **Tabular data:** Đối với cùng một mẫu, như chẩn đoán bệnh cho bệnh nhân, có thể sẽ có bệnh nhân có số lượng thông tin nhiều hơn hẳn bệnh nhân khác, nhưng chung quy chúng ta đều phải quy chúng về 1 bảng (thực hiện feature engineering) và thực hiện dự đoán trên cùng một số lượng features nhất định.
- **Sequential data:** Dữ liệu dạng chuỗi lại có tính chất rất khác so với 2 cái trên, khi mà từng mẫu có độ dài khác nhau, ví dụ như mỗi câu trong văn bản sẽ có độ dài khác nhau, việc chuẩn hóa (như padding, truncate) sẽ dẫn tới mất ngữ nghĩa hoặc tạo ra nhiễu.

Vì vậy, để xử lí cho data dạng time sequence, chúng ta cần mô hình riêng biệt xử lí hiệu quả cho chúng, sẽ được tìm hiểu ở phần tiếp theo.

Mô hình Recurrent Neural Networks (RNNs):



Recurrent Neural Networks (RNNs) là một mạng neural nhân tạo, được thiết kế đặc biệt để xử lý dữ liệu tuần tự (sequential data), ứng dụng rộng rãi đối với các tác vụ như Time Series Prediction, Speech Recognition and Synthesis. Điểm đặc trưng của RNN nằm ở khả năng ghi nhớ thông tin từ các bước trước đó bằng cách sử dụng **trạng thái ẩn** (hidden states). Tại mỗi thời điểm t, trạng thái h_t được cập nhật dựa trên trạng thái của bước trước đó h_{t-1} và dữ liệu đầu vào hiện tại x_t. Từ trạng thái ẩn này, RNN có thể đưa ra dự đoán tương ứng y_t, giúp mô hình học được mối quan hệ phụ thuộc trong chuỗi dữ liệu một cách hiệu quả.

RNN trong xử lý ngôn ngữ tự nhiên (NLP) Mô hình RNN là một công cụ mạnh mẽ trong việc xử lý ngôn ngữ tự nhiên (NLP), đặc biệt đối với các bài toán yêu cầu học mối quan hệ tuần tự giữa các từ trong câu. Với dữ liệu văn bản, mỗi từ trong chuỗi được biểu diễn bằng vector (word embedding), và RNN có thể mô hình hóa ngữ cảnh của từ dựa trên trạng thái ẩn h_t tại mỗi bước. Một ví dụ điển hình như sử dụng RNN trong bài toán dịch máy (machine translation), RNN có thể học cách mỗi từ phụ thuộc vào các từ trước đó, từ đó đưa ra kết quả chính xác hơn. Một trong những ưu điểm nổi bật là khả năng nắm bắt ngữ nghĩa của câu nhờ vào việc ghi nhớ trạng thái qua thời gian. Tuy nhiên, để xử lý các câu dài và mối quan hệ xa, các biến thể của RNN như LSTM và GRU thường được ưu tiên.

RNN trong bài toán Time Series Prediction (Forecasting) RNN cũng rất phù hợp để phân tích dữ liệu chuỗi thời gian, nơi các mẫu dữ liệu được ghi nhận tuần tự theo thời gian (ví dụ: giá cổ phiếu, nhiệt độ, lưu lượng mạng). Trong bài toán này, RNN sử dụng các hidden state để học mối quan hệ giữa các bước thời gian, giúp dự đoán các giá trị tương lai dựa trên các thông tin lịch sử. Thông thường, input của bài toán sẽ là các giá trị trong cửa sổ thời gian trước đó $\{y_{t-1}, y_{t-2}, ..., y_{t-n}\}$, output là giá trị tương lai tại thời điểm t (hoặc nhiều thời điểm tiếp theo nếu dự báo nhiều bước). RNN có thể nắm bắt các xu hướng ngắn hạn và dài hạn trong dữ liệu, nhưng tương tự bài toán text, nó cũng gặp khó khăn khi làm việc với chuỗi dài, yêu cầu các biến thể như LSTM hoặc GRU để tối ưu hóa.
Trong bài tập này ở phần lập trình, chúng ta sẽ thực hành xây dựng mô hình Classification và Regression có tích hợp các layer RNN, áp dụng vào giải quyết hai bài toán là Financial News Sentiment Analysis và Temperature Forecasting. Đồng thời, ôn tập một số lý thuyết về RNNs thông qua bài tập trắc nghiệm.

Phần II: Bài tập

A. Phần lập trình

- **Problem 01: Sentiment Analysis for Financial News** Trong bài tập này, chúng ta sẽ xây dựng một mô hình về Text Classfication dùng để phân loại tình hình tin tức tài chính là tích cực (positive), tiêu cực (negative) hay trung lập (neutral) dựa trên một đoạn văn có nội dung về tài chính cho trước. Các bạn sẽ thực hiện theo hướng dẫn sau:

1. **Import các thư viện cần thiết:**

```
import torch
import torch.nn as nn

seed = 1
torch.manual_seed(seed)

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import nltk
import unidecode

nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
```

2. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu tại đây. Ở đây, ta sẽ chỉ quan tâm đến file **all_data.csv**. Dưới đây là thông tin 4 hàng đầu tiên của bảng dữ liệu:

Index	sentiment	content
0	neutral	According to Gran , the company has no plans to move all production to Russia , although that is where the company is growing .
1	neutral	Technopolis plans to develop in stages an area of no less than 100,000 square meters in order to host companies working in computer technologies and telecommunications , the statement said .
2	negative	The international electronic industry company Elcoteq has laid off tens of employees from its Tallinn facility ; contrary to earlier layoffs the company contracted the ranks of its office workers , the daily Postimees reported .
3	positive	With the new production plant the company would increase its capacity to meet the expected increase in demand and would improve the use of raw materials and therefore increase the production profitability .

Dựa vào bảng dữ liệu trên, ta xác định được rằng mô hình ta xây dựng sẽ sử dụng thông tin từ cột **content** để dữ đoán nhãn tại cột **sentiment**.

3. **Đọc bộ dữ liệu:** Sử dụng thư viện pandas, các bạn đọc bộ dữ liệu lên như sau, lưu ý cần phải cài đặt tham số encoding='ISO-8859-1' để đưa về định dạng đúng như đoạn code như sau:

```
dataset_path = 'dataset/all-data.csv'
headers = ['sentiment', 'content']
df = pd.read_csv(
    dataset_path,
    names=headers,
    encoding='ISO-8859-1'
)
```

Từ đây, ta có thể xác định được danh sách các class của bài toán bằng phương thức unique() của Pandas, đồng thời ta cũng sẽ đổi class dạng string thành số nguyên đại diện cho ID của chúng:

```
classes = {
    class_name: idx for idx, class_name in enumerate(df['sentiment'].unique()
    .tolist())
}
df['sentiment'] = df['sentiment'].apply(lambda x: classes[x])
```

4. **Tiền xử lý dữ liệu:** Để tận dùng hàm apply của pandas, ta sẽ tiền xử lý văn bản cột **content** ngay tại đây. Đầu tiên, định nghĩa hàm chuẩn hóa, nhận tham số đầu vào là 1 text và trả về 1 text đã được chuẩn hóa:

```
english_stop_words = stopwords.words('english')
stemmer = PorterStemmer()

def text_normalize(text):
    text = text.lower()
    text = unidecode.unidecode(text)
    text = text.strip()
    text = re.sub(r'[^\w\s]', '', text)
    text = ' '.join([word for word in text.split(' ') if word not in
    english_stop_words])
    text = ' '.join([stemmer.stem(word) for word in text.split(' ')])

    return text
```

Các kỹ thuật chuẩn hóa được áp dụng trong bài bao gồm: Chuyển chữ viết thường (Lowercasing), Xóa dấu câu (Punctuation Removal), Xóa stopwords (Stopwords Removal), Stemming. Sau đó, áp dụng hàm text_normalize() vào cột **content**:

```
df['content'] = df['content'].apply(lambda x: text_normalize(x))
```

5. **Xây dựng bộ từ vựng:** Để huấn luyện dữ liệu văn bản, ta cần chuyển đổi các văn bản thành vector. Phương thức đơn giản nhất, ta sẽ thu thập toàn bộ các từ trong bộ dữ liệu thành một tập từ vựng, mỗi từ có một ID riêng. Từ đó, với một văn bản đầu, ta quy đổi sang ID tương ứng, như vậy sẽ được một vector số nguyên. Đoạn code tạo bộ từ vựng được thực hiện như sau:

```
vocab = []
for sentence in df['content'].tolist():
    tokens = sentence.split()
    for token in tokens:
        if token not in vocab:
            vocab.append(token)

vocab.append('UNK')
vocab.append('PAD')
word_to_idx = {word: idx for idx, word in enumerate(vocab)}
vocab_size = len(vocab)
```

Trong đó, **'UNK'** đại diện cho các từ không thuộc bộ từ vựng và **'PAD'** đại diện cho các ô trống được thêm vào để thỏa mãn độ dài tối thiểu của một văn bản mà ta quy định về sau. Với bộ từ vựng trên, ta xây dựng hàm transform() dùng để chuyển đổi văn bản thành danh sách các số nguyên như sau:

```
def transform(text, word_to_idx, max_seq_len):
    tokens = []
    for w in text.split():
        try:
            w_ids = word_to_idx[w]
        except:
            w_ids = word_to_idx['UNK']
        tokens.append(w_ids)

    if len(tokens) < max_seq_len:
        tokens += [word_to_idx['PAD']] * (max_seq_len - len(tokens))
    elif len(tokens) > max_seq_len:
        tokens = tokens[:max_seq_len]

    return tokens
```

Trong đó, **word_to_idx** đại diện cho bộ từ vựng và max_seq_len là một hằng số quy ước độ dài của các văn bản, đảm bảo các văn bản có cùng độ dài.

6. **Chia bộ dữ liệu train, val, test:** Ta dùng hàm **train_test_split** của thư viện scikit-learn để chia bộ dữ liệu thành 3 bộ train/val/test theo tỷ lệ 7:2:1:

```
val_size = 0.2
test_size = 0.125
is_shuffle = True
texts = df['content'].tolist()
labels = df['sentiment'].tolist()

X_train, X_val, y_train, y_val = train_test_split(
    texts, labels,
    test_size=val_size,
    random_state=seed,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=val_size,
    random_state=seed,
    shuffle=is_shuffle
)
```

7. **Xây dựng pytorch datasets:** Ta triển khai class tên FinancialNews với đầu vào gồm cặp dữ liệu đầu vào X - y, bộ từ điển cũng như hàm transform như sau:

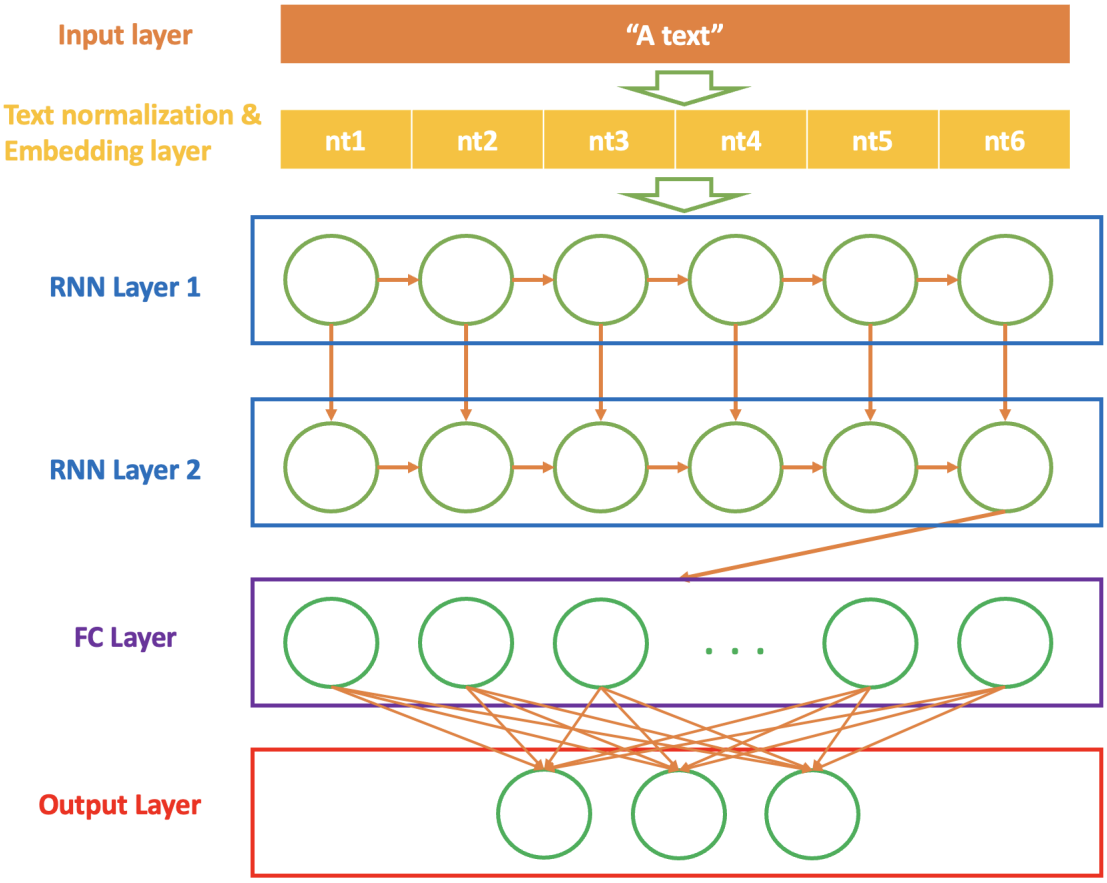
```
class FinancialNews(Dataset):
    def __init__(
        self,
        X, y,
        word_to_idx,
        max_seq_len,
        transform=None
    ):
        self.texts = X
        self.labels = y
        self.word_to_idx = word_to_idx
```

```
        self.max_seq_len = max_seq_len
        self.transform = transform

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]

        if self.transform:
            text = self.transform(
                text,
                self.word_to_idx,
                self.max_seq_len
            )
        text = torch.tensor(text)

        return text, label
```

8. **Khai báo dataloader:**

```
max_seq_len = 32

train_dataset = FinancialNews(
    X_train, y_train,
    word_to_idx=word_to_idx,
    max_seq_len=max_seq_len,
    transform=transform
)
val_dataset = FinancialNews(
    X_val, y_val,
    word_to_idx=word_to_idx,
    max_seq_len=max_seq_len,
    transform=transform
)
test_dataset = FinancialNews(
    X_test, y_test,
    word_to_idx=word_to_idx,
    max_seq_len=max_seq_len,
    transform=transform
)

train_batch_size = 128
test_batch_size = 8

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batch_size,
    shuffle=True
)
val_loader = DataLoader(
    val_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
test_loader = DataLoader(
    test_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
```

9. **Xây dựng mô hình:** Chúng ta sẽ xây dựng phân loại cảm xúc với 2 layer RNNs. Vector tại hidden state cuối cùng của RNN layer thứ 2 sẽ được đưa vào FC layer để thực hiện dự đoán. Các bạn có thể quan sát rõ hơn thông qua hình dưới đây:



Hình 1: Ảnh minh họa kiến trúc của mô hình

Như vậy, code cài đặt như sau:

```
class SentimentClassifier(nn.Module):
    def __init__(
        self, vocab_size, embedding_dim,
        hidden_size, n_layers, n_classes,
        dropout_prob
    ):
        super(SentimentClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.RNN(embedding_dim, hidden_size, n_layers, batch_first=
    True)
        self.norm = nn.LayerNorm(hidden_size)
        self.dropout = nn.Dropout(dropout_prob)
        self.fc1 = nn.Linear(hidden_size, 16)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(16, n_classes)

    def forward(self, x):
        x = self.embedding(x)
        x, hn = self.rnn(x)
        x = x[:, -1, :]
```

```
        x = self.norm(x)
        x = self.dropout(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)

        return x
```

Ta định nghĩa class với tên SentimentClassifier, nhận tham số đầu vào gồm: kích thước bộ từ vựng (vocab_size), kích thước không gian vector của mỗi từ trong chuỗi đầu vào (embedding_dim), kích thước không gian vector của hidden state (hidden_size), số lượng RNN layer (n_layers), số class dự đoán của bài toán (n_classes) và tỉ lệ dropout (dropout_prob). Với class trên, ta khai báo mô hình SentimentClassifier:

```
n_classes = len(list(classes.keys()))
embedding_dim = 64
hidden_size = 64
n_layers = 2
dropout_prob = 0.2
device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = SentimentClassifier(
    vocab_size=vocab_size,
    embedding_dim=embedding_dim,
    hidden_size=hidden_size,
    n_layers=n_layers,
    n_classes=n_classes,
    dropout_prob=dropout_prob
).to(device)
```

10. **Cài đặt hàm loss và optimizer:** Ta sử dụng hàm loss crossentropy cho bài toán phân loại và Adam optimizer.

```
lr = 1e-4
epochs = 50

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=lr
)
```

11. **Thực hiện huấn luyện:** Ta định nghĩa hàm fit() và evaluate() dùng để huấn luyện và đánh giá mô hình như sau:

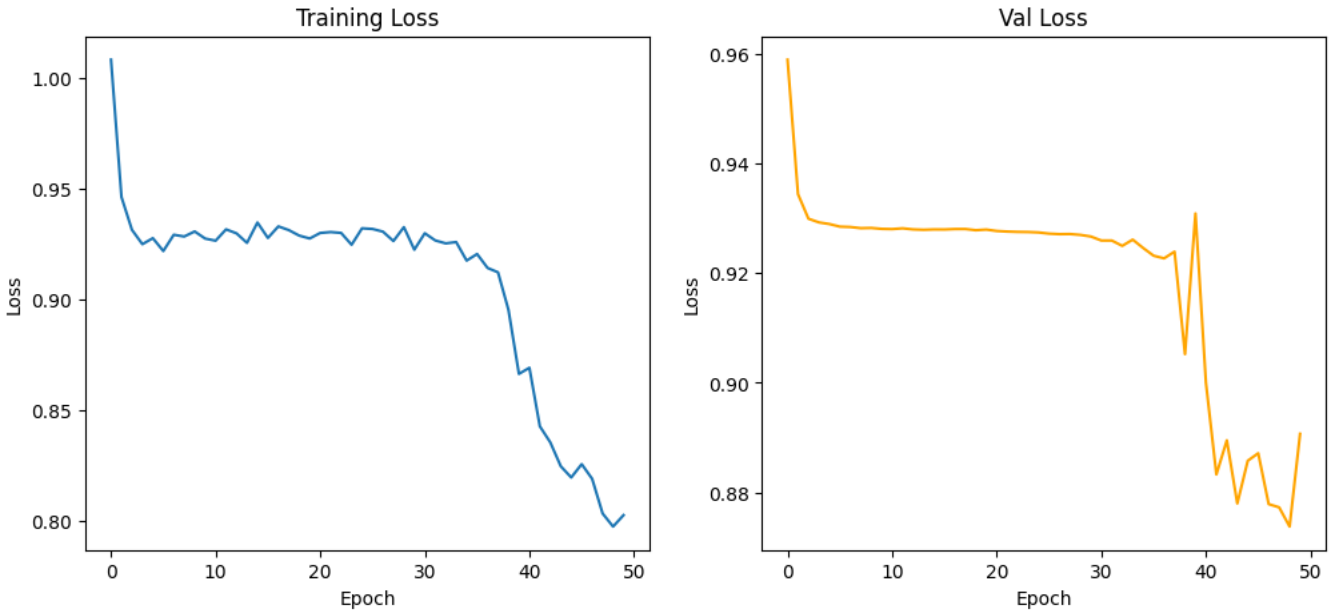
```
def fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs
):
    train_losses = []
    val_losses = []

    for epoch in range(epochs):
        batch_train_losses = []

```

```
        model.train()
        for idx, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            batch_train_losses.append(loss.item())

        train_loss = sum(batch_train_losses) / len(batch_train_losses)
        train_losses.append(train_loss)

        val_loss, val_acc = evaluate(
            model, val_loader,
            criterion, device
        )
        val_losses.append(val_loss)

        print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}')

    return train_losses, val_losses
```

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            losses.append(loss.item())
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    loss = sum(losses) / len(losses)
    acc = correct / total

    return loss, acc
```

Tổng hợp tất cả các dữ kiện trên, ta tiến hành huấn luyện mô hình SentimentClassifier:

```
train_losses, val_losses = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs
)
```



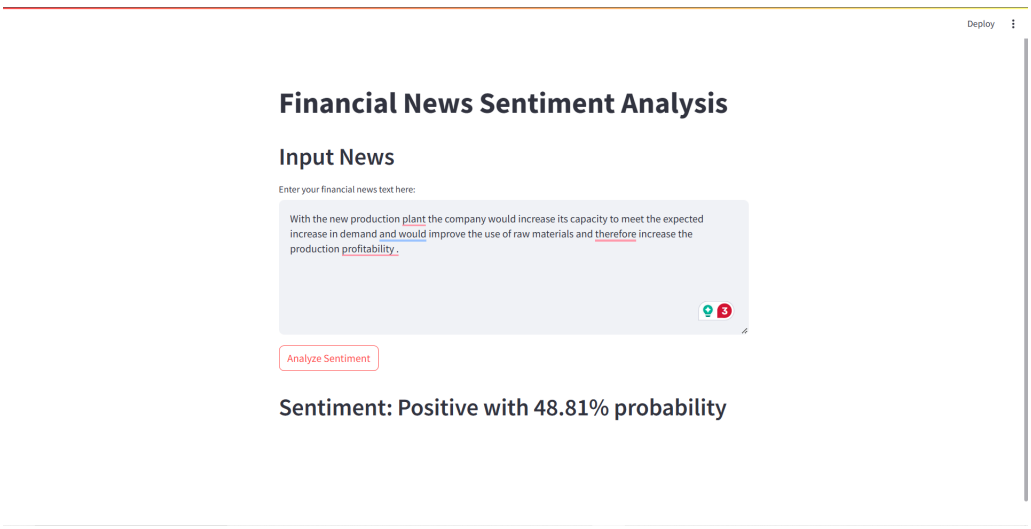
Hình 2: Kết quả huấn luyện mô hình SentimentClassifier

12. **Đánh giá mô hình:** Sử dụng hàm evaluate() đã xây dựng ở trên, ta đánh giá mô hình trên hai tập val và test như sau:

```
val_loss, val_acc = evaluate(
    model,
    val_loader,
    criterion,
    device
)
test_loss, test_acc = evaluate(
    model,
    test_loader,
    criterion,
    device
)

print('Evaluation on val/test dataset')
print('Val accuracy: ', val_acc)
print('Test accuracy: ', test_acc)
```

Lưu ý: Dựa vào phần code mẫu trên (dùng RNN), các bạn hãy thực hiện bài tập với các version bao gồm LSTM và BiLSTM (các bạn tham khảo cách sử dụng LSTM/BiLSTM trong pytorch tại đây) với các config về tham số không đổi.

13. **Triển khai mô hình**: Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:
    - Link Streamlit
    - Github
    - Giao Diện



Hình 3: Giao diện ứng dụng

- **Problem 02: Hourly Temperature Forecasting**

  Trong bài tập này, chúng ta sẽ xây dựng một mô hình về Times Series dùng để dự đoán nhiệt độ trong 1 giờ tiếp theo dựa trên nhiệt độ của 6 giờ trước đó. Các bạn sẽ thực hiện theo hướng dẫn sau:

  1. **Import các thư viện cần thiết:**

```python
import torch
import torch.nn as nn

seed = 1
torch.manual_seed(seed)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from torch.utils.data import Dataset, DataLoader
```

  2. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu tại đây. Dưới đây là thông tin 4 hàng đầu tiên của bảng dữ liệu:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.

  3. **Chuẩn bị dữ liệu:**

(a) **Đọc dữ liệu từ file .csv:**

```
dataset_filepath = 'dataset/weatherHistory.csv'
df = pd.read_csv(dataset_filepath)
```

Trong bài tập lần này, vì mô hình chỉ dự đoán nhiệt độ (Temperature (C)) nên ta sẽ loại bỏ đi các cột không cần thiết trong DataFrame trước khi đưa vào tiền xử lý:

```
univariate_df = df['Temperature (C)']
univariate_df.index = df['Formatted Date']
```

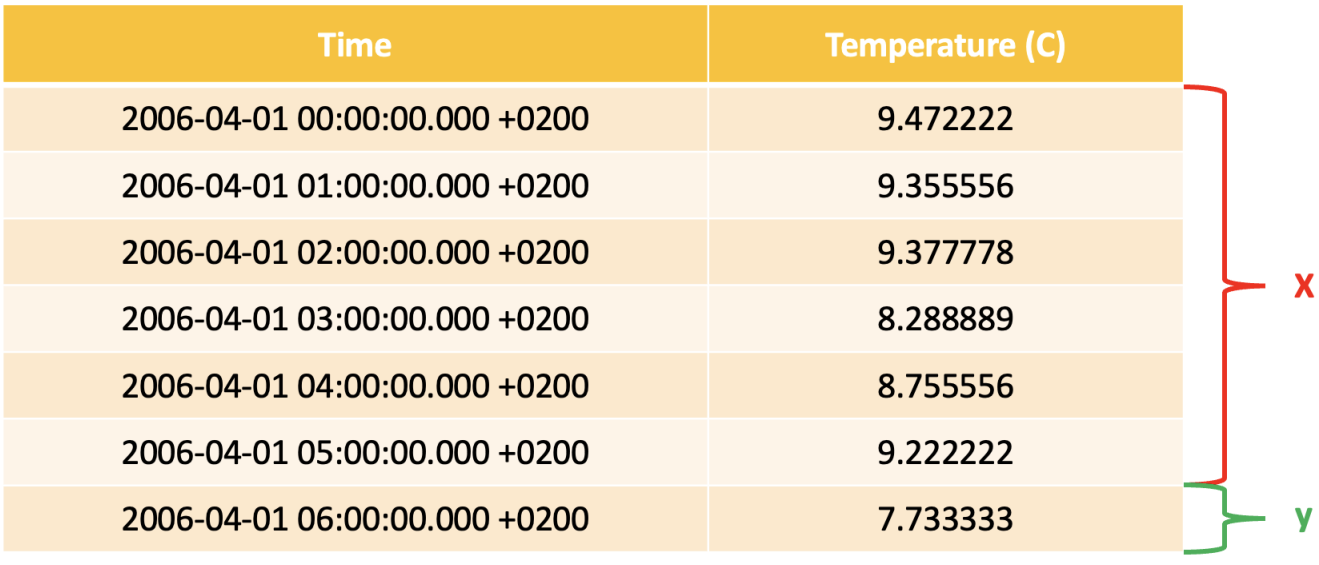
Như vậy, dữ liệu bảng của chúng ta sẽ có dạng:

Formatted Date	Temperature (C)
2006-04-01 00:00:00.000 +0200	9.472222
2006-04-01 01:00:00.000 +0200	9.355556
006-04-01 02:00:00.000 +0200	9.377778
2006-04-01 03:00:00.000 +0200	8.288889

Bảng 1: 4 hàng đầu tiên của bảng dữ liệu sau khi đã xóa đi các cột không cần thiết

(b) **Xây dựng hàm tạo cặp X, y:** Cũng như bất kì các bài toán thuộc nhánh học có giám sát trên dữ liệu dạng bảng khác, ta cần xác định X, y của bài toán sao cho phù hợp. Thông thường, X sẽ là các đặc trưng và y sẽ là nhãn tương ứng cho các đặc trưng X (thường được định nghĩa rõ ràng trong bảng dữ liệu). Tuy nhiên ở bài toán này, ta không có các nhãn cụ thể theo như đề bài đưa ra. Chính vì vậy, cần phải thực hiện chỉnh sửa Input/Output của dữ liệu sao cho phù hợp để ta có thể đưa vào mô hình.

Với việc đề bài yêu cầu sử dụng thông tin nhiệt độ của 6 giờ trước để dự đoán nhiệt độ của 1 giờ tiếp theo, ta có thể xác định được rằng X sẽ là nhiệt độ của 6 giờ, y là nhiệt độ của giờ tiếp theo, định nghĩa này có thể được thể hiện như hình dưới đây:



Hình 4: Cách xác định X, y dựa trên 7 mẫu của bảng dữ liệu cho trước

Từ đây, có thể thấy khi ta duyệt qua từng hàng trên bảng dữ liệu, ta có thể tạo ra các cặp X, y theo đúng ý của đề bài yêu cầu, nhờ đó có thể sử dụng để đưa vào huấn luyện mô hình. Kỹ thuật này được gọi là **Windowing** (coi các cặp X, y như các "cửa sổ"). Ta sẽ thực hiện triển khai hàm windowing này như sau:

```
1  input_size = 6
2  label_size = 1
3  offset = 1
4
5  def slicing_window(df, df_start_idx, df_end_idx, input_size, label_size,
       offset):
6      features = []
7      labels = []
8
9      window_size = input_size + offset
10
11     if df_end_idx == None:
12         df_end_idx = len(df) - window_size
13
14     for idx in range(df_start_idx, df_end_idx):
15         feature_end_idx = idx + input_size
16         label_start_idx = idx + window_size - label_size
17
18         feature = df[idx:feature_end_idx]
19         label = df[label_start_idx:(idx+window_size)]
20
21         features.append(feature)
22         labels.append(label)
23
24     features = np.expand_dims(np.array(features), -1)
25     labels = np.array(labels)
26
27     return features, labels
```

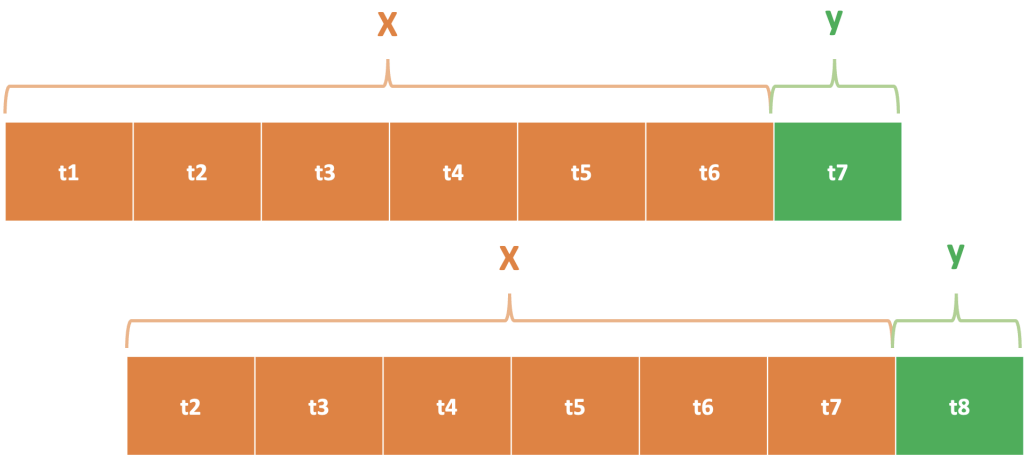
Ý nghĩa của các tham số đầu vào là:

- **df**: DataFrame của bảng dữ liệu.
- **df_start_idx:** Chỉ mục bắt đầu thực hiện "windowing" trong bảng dữ liệu.
- **df_end_idx:** Chỉ mục kết thúc thực hiện "windowing" trong bảng dữ liệu.
- **input_size:** Kích thước (số thời gian) của X.
- **label_size:** Kích thước (số thời gian) của y.
- **offset:** Khoảng cách về thời gian giữa X và y.

Đối với yêu cầu đề bài, có thể dễ dàng xác định được rằng input_size = 6, output_size = 1 và offset = 1.

Trong đó:

- **Dòng 6, 7, 9, 11, 12:** Tạo hai list dùng để chứa các mẫu dữ liệu X, y. Sau đó thực hiện tính kích thước của cửa sổ = input_size + offset.
- **Dòng 14, 15, 16, 18, 19, 21, 22:** Bắt đầu duyệt qua từng mẫu dữ liệu theo chỉ mục trong khoảng (df_start_idx, df_end_idx), ta sẽ thực hiện tìm X và y và thêm vào danh sách chứa features, labels đã khai báo phía trên.
- **Dòng 24, 25, 27:** Cuối cùng, ta chuyển đổi 2 list thành np.ndarray và sử dụng chúng làm kết quả trả về của hàm:



Hình 5: Kỹ thuật Windowing: duyệt qua từng chỉ mục trong bảng dữ liệu để xác định cặp X, y

4. **Chia bộ dữ liệu train, val, test:** Sử dụng hàm slicing_window() đã định nghĩa ở trên, ta tiến hành chia ba bộ dữ liệu train, val, test như sau:

```
dataset_length = len(univariate_df)
train_size = 0.7
val_size = 0.2
train_end_idx = int(train_size * dataset_length)
val_end_idx = int(val_size * dataset_length) + train_end_idx

X_train, y_train = slicing_window(
    univariate_df,
    df_start_idx=0,
    df_end_idx=train_end_idx,
    input_size=input_size,
    label_size=label_size,
    offset=offset
)

X_val, y_val = slicing_window(
    univariate_df,
    df_start_idx=train_end_idx,
    df_end_idx=val_end_idx,
    input_size=input_size,
    label_size=label_size,
    offset=offset
)

X_test, y_test = slicing_window(
    univariate_df,
    df_start_idx=val_end_idx,
    df_end_idx=None,
    input_size=input_size,
    label_size=label_size,
    offset=offset
)
```

Ở đây, ta chia bộ dữ liệu theo tỉ lệ 7/2/1, tương ứng cho train/val/test.

5. **Xây dựng pytorch datasets:**

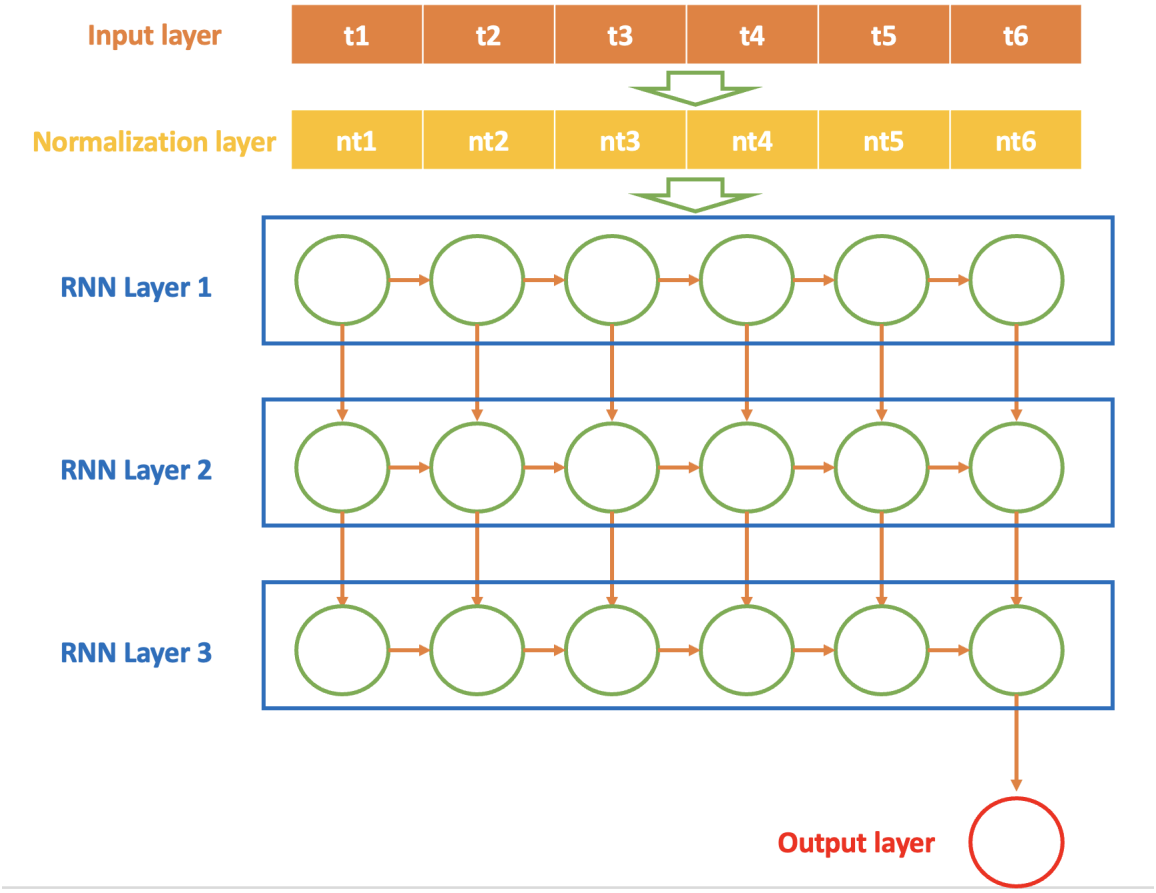
```
class WeatherForecast(Dataset):
```

```
    def __init__(
        self,
        X, y,
        transform=None
    ):
        self.X = X
        self.y = y
        self.transform = transform

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        X = self.X[idx]
        y = self.y[idx]

        if self.transform:
            X = self.transform(X)

        X = torch.tensor(X, dtype=torch.float32)
        y = torch.tensor(y, dtype=torch.float32)

        return X, y
```

6. **Khai báo dataloader:**

```
train_dataset = WeatherForecast(
    X_train, y_train
)
val_dataset = WeatherForecast(
    X_val, y_val
)
test_dataset = WeatherForecast(
    X_test, y_test
)

train_batch_size = 128
test_batch_size = 8

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batch_size,
    shuffle=True
)
val_loader = DataLoader(
    val_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
test_loader = DataLoader(
    test_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
```

7. **Xây dựng mô hình:** Ta xây dựng class WeatherForecastor dùng để dự đoán nhiệt độ ứng dụng 3 lớp RNN, vector hidden state cuối cùng của lớp RNN thứ 3 sẽ được đưa vào lớp FC để thực hiện dự đoán. Các bạn có thể quan sát rõ hơn ở hình sau:



Hình 6: Ảnh minh họa kiến trúc của mô hình

```python
class WeatherForecastor(nn.Module):
    def __init__(
        self, embedding_dim, hidden_size,
        n_layers, dropout_prob
    ):
        super(WeatherForecastor, self).__init__()
        self.rnn = nn.RNN(embedding_dim, hidden_size, n_layers, batch_first=True)
        self.norm = nn.LayerNorm(hidden_size)

        self.dropout = nn.Dropout(dropout_prob)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        x, hn = self.rnn(x)
        x = x[:, -1, :]
        x = self.norm(x)
        x = self.dropout(x)
        x = self.fc(x)

        return x
```

Sử dụng class đã định nghĩa ở trên, ta khai báo mô hình WeatherForecastor như sau:

```python
embedding_dim = 1
hidden_size = 8
```

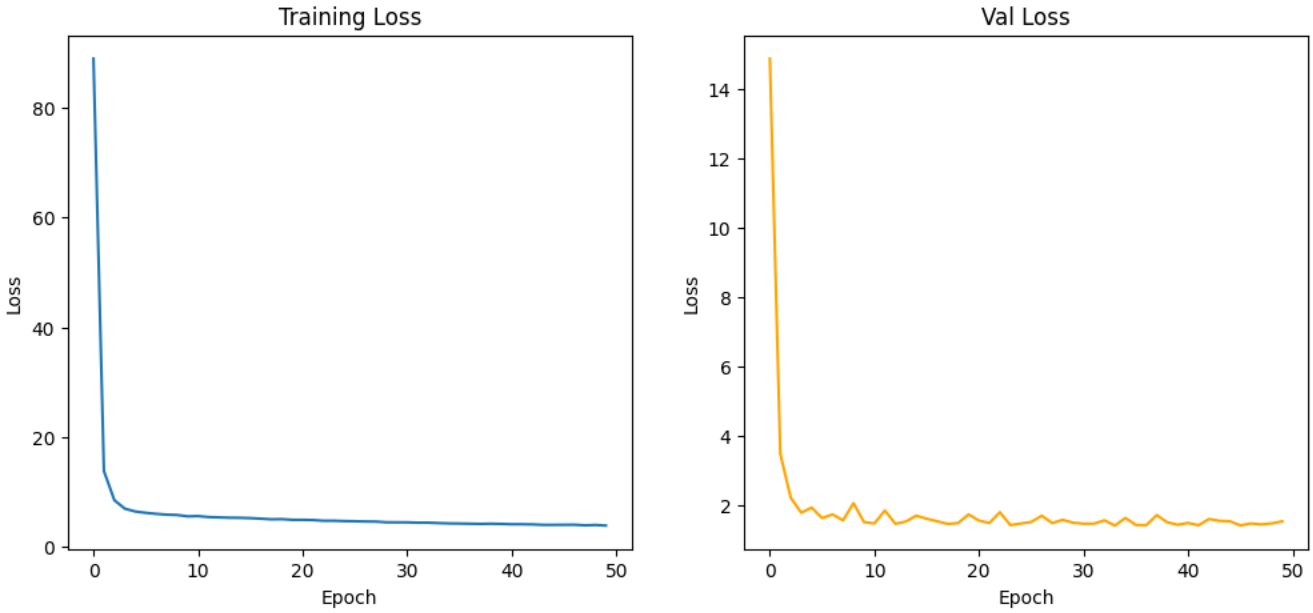
```
n_layers = 3
dropout_prob = 0.2
device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = WeatherForecastor(
    embedding_dim=embedding_dim,
    hidden_size=hidden_size,
    n_layers=n_layers,
    dropout_prob=dropout_prob
).to(device)
```

8. **Cài đặt hàm loss và optimizer:** Vì bài toán dự đoán nhiệt độ thời tiết này dưới dạng là bài Regression, ta sẽ sử dụng hàm loss là MSE và optimizer là Adam.

```
lr = 1e-3
epochs = 50

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=lr
)
```

9. **Thực hiện huấn luyện mô hình:** Ta sử dụng hàm fit() và evaluate() đã định nghĩa ở bài Sentiment Analysis trước, thực hiện huấn luyện mô hình dự báo nhiệt độ như sau:

```
train_losses, val_losses = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs
)
```



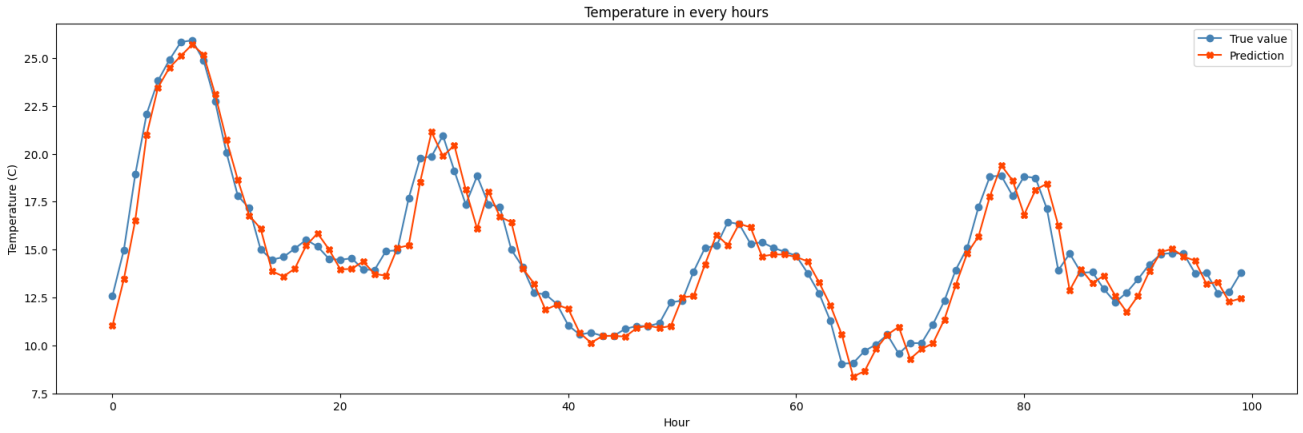
Hình 7: Trực quan hóa kết quả loss trong quá trình huấn luyện mô hình

10. **Đánh giá mô hình:** Ta sử dụng hàm evaluate() để đánh giá mô hình đã huấn luyện trên tập val và test như sau:

```
val_loss = evaluate(
    model,
    val_loader,
    criterion,
    device
)
test_loss = evaluate(
    model,
    test_loader,
    criterion,
    device
)

print('Evaluation on val/test dataset')
print('Val loss: ', val_loss)
print('Test loss: ', test_loss)
```

Bên cạnh đó, đối với bài toán time series, ta cũng có thể plot dự đoán của mô hình so với dữ liệu thực tế tromg một khoảng thời gian như sau:

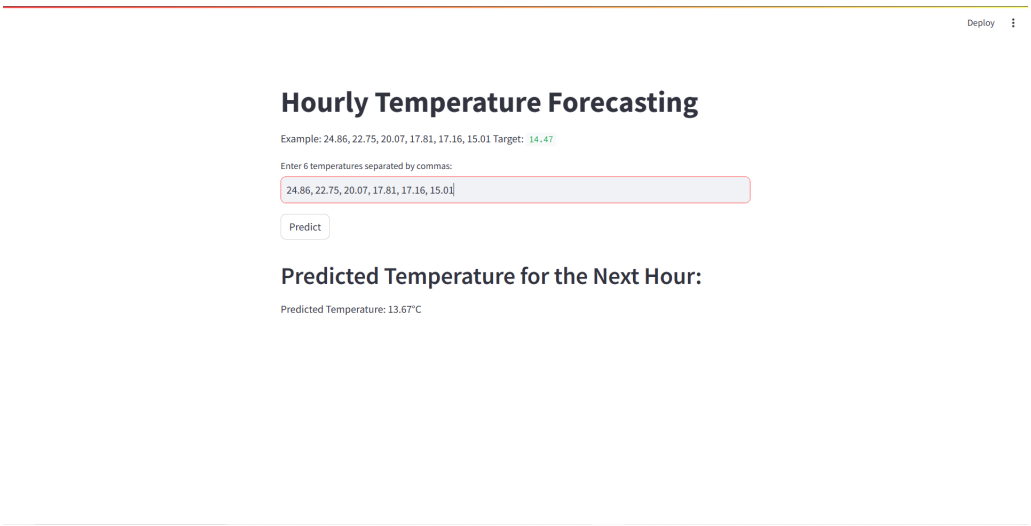
```
def plot_difference(y, pred):
    plt.figure(figsize=(20, 6))
    times = range(len(y))
    y_to_plot = y.flatten()
    pred_to_plot = pred.flatten()

    plt.plot(times, y_to_plot, color='steelblue', marker='o', label='True value')
    plt.plot(times, pred_to_plot, color='orangered', marker='X', label='Prediction')

    plt.title('Temperature in every hours')
    plt.xlabel('Hour')
    plt.ylabel('Temperature (C)')
    plt.legend()
    plt.show()

inputs = torch.tensor(X_test[:100], dtype=torch.float32).to(device)
model.eval()
with torch.no_grad():
    outputs = model(inputs).detach().cpu().numpy()
plot_difference(y_test[:100], outputs)
```



Hình 8: Kết quả dự đoán so với kết quả thực tế trong một khoảng thời gian

Lưu ý: Dựa vào phần code mẫu trên (dùng RNN), các bạn hãy thực hiện bài tập với các version bao gồm LSTM và BiLSTM (các bạn tham khảo cách sử dụng LSTM/BiLSTM trong pytorch tại đây) với các config về tham số không đổi.

- **Triển khai mô hình**: Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:
  - Link Streamlit
  - Github
  - Giao Diện



Hình 9: Giao diện ứng dụng

B. Phần trắc nghiệm

1. Đặc điểm của dữ liệu dạng chuỗi (sequential data) là gì?

(A). Mỗi mẫu có độ dài cố định
(B). Dữ liệu có tính chất không thay đổi theo thời gian
(C). Mỗi mẫu có độ dài khác nhau và có thể mất ngữ nghĩa khi chuẩn hóa
(D). Mỗi mẫu có số lượng thuộc tính cố định

2. Trong phần xử lí dữ liêu của **Problem 01** tại sao cần thêm token '**UNK**' và '**PAD**' vào bộ từ vựng?

(A). Token '**UNK**' dùng để thay thế các từ không có trong bộ từ vựng, còn token '**PAD**' giúp chuẩn hóa độ dài của các chuỗi đầu vào.
(B). Token '**UNK**' dùng để giảm độ dài của văn bản, còn token '**PAD**' dùng để tạo ra các từ mới.
(C). Token '**UNK**' dùng để xóa bỏ các từ không quan trọng, còn token '**PAD**' để thay thế từ khóa.
(D). Token '**UNK**' và '**PAD**' đều dùng để cải thiện hiệu suất mô hình học máy mà không thay đổi dữ liệu đầu vào.

3. "input_size + offset" trong phần Hourly Temperature Forecasting thể hiện điều gì?

(A). Số lượng dự đoán cần thiết cho mỗi bước tính toán
(B). Kích thước cửa sổ lấy cặp dữ liệu
(C). Khoảng cách giữa các mẫu dữ liệu trong bộ dữ liệu
(D). Kích thước của dữ liệu đầu vào

4. Mô hình RNN có bao nhiêu bộ tham số (không kể bias)?

(A). Có 1 bộ tham số
(B). Không xác định, tùy vào số lượng time step của mô hình RNN
(C). Có 3 bộ tham số
(D). Có 2 bộ tham số

5. Nhược điểm của mô hình RNN là?

(A). Mô hình RNN chỉ có thể xử lý dữ liệu chuỗi ngắn.
(B). RNN không thể học các mối quan hệ giữa các bước thời gian gần nhau.
(C). RNN không thể áp dụng cho các bài toán phân loại.
(D). Mô hình RNN gặp phải vấn đề vanishing gradient và không thể học các mối quan hệ dài hạn hiệu quả.

6. Cho đoạn code sau

```python
import torch
import torch.nn as nn

input_tensor = torch.randn(5, 10, 10)

lstm = nn.LSTM(input_size=10, hidden_size=20, num_layers=2, batch_first=True)
bilstm = nn.LSTM(input_size=10, hidden_size=20, num_layers=2, batch_first=True,
    bidirectional=True)

output_lstm, (hidden_state_lstm, cell_state_lstm) = lstm(input_tensor)

output_bilstm, (hidden_state_bilstm, cell_state_bilstm) = bilstm(input_tensor)

print(output_lstm.shape)
print(output_bilstm.shape)
```

Khi đưa một tensor có kích thước cố định (5, 10, 10) cho các lớp LSTM và BiLSTM được định nghĩa như code trên, kích thước đầu ra của mỗi lớp sẽ khác nhau như thế nào (xét theo biến trong hàm print)?

(A). LSTM sẽ tạo ra đầu ra với kích thước (5, 10, 20) và BiLSTM với kích thước (5, 10, 400).

(B). Cả LSTM và BiLSTM đều output ra tensor có shape là (5, 10, 20).

(C). Output của BiLSTM có số hidden state gấp đôi so với LSTM.

(D). BiLSTM sẽ tạo ra đầu ra với số sequence length nhỏ hơn so với LSTM do xử lý hai chiều.

7. Đâu là điểm mạnh của LSTM so với RNN?

(A). LSTM có thể xử lý tốt với chuỗi dài

(B). LSTM có tốc độ xử lý nhanh hơn

(C). LSTM yêu cầu ít dữ liệu huấn luyện hơn

(D). LSTM yêu cầu tài nguyên tính toán ít hơn

8. Dòng code ‘x = x[: , -1 , :]‘ ở trong class WeatherForecastor có ý nghĩa là gì?

(A). Lấy giá trị output của tất cả những time step trong chuỗi
(B). Lấy giá trị output của time step cuối cùng của chuỗi
(C). Lấy batch đầu tiên của dữ liệu
(D). Lấy giá trị hidden state cuối cùng của mô hình RNN

9. Đâu không phải là một ứng dụng chính của RNN?

(A). Text Classification

(B). Stock Trading Prediction

(C). Image Classification

(D). Machine Translation

10. Ý nghĩa chính của hidden state trong RNN là?

(A). Dùng để lưu trữ thông tin của các thời điểm trước

(B). Dùng để biểu diễn trọng số của input

(C). Dùng để tăng tốc độ tính toán của RNN so với CNN

(D). Dùng để chuẩn hóa dữ liệu đầu vào

Phần III: Phụ Lục:

1. **Hint:** Các file code template có thể tải tại Link
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể tải tại Link (Lưu ý Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)
3. **Rubric**:

Recurrent Neural Networks Exercise - Rubric		
Câu	**Kiến Thức**	**Đánh Giá**
II.A. 1, 2	- Kiến thức về lập trình pytorch. - Quy trình cơ bản trong việc xây dựng mô hình mạng nơ-ron sử dụng các layer thuộc họ RNN. - Cách xử lý dữ liệu văn bản và dữ liệu time series.	- Biết cách ứng dụng kỹ thuật lập trình pytorch để xây dựng một mô hình phân loại văn bản và mô hình time series.
II.B. 1, 2, 3	- Kiến thức cơ bản về RNN và LSTM.	- Hiểu được các tính chất cơ bản hai mô hình RNN và LSTM.
II.B. 4, 5, 6	- Kiến thức cơ bản về RNN và LSTM.	- Hiểu được các tính chất cơ bản hai mô hình RNN và LSTM.
II.B. 7, 8, 9	- Công thức tính toán hidden state, output và backpropagation trong RNN.	- Hiểu chi tiết các công thức trong RNN.

- Hết -