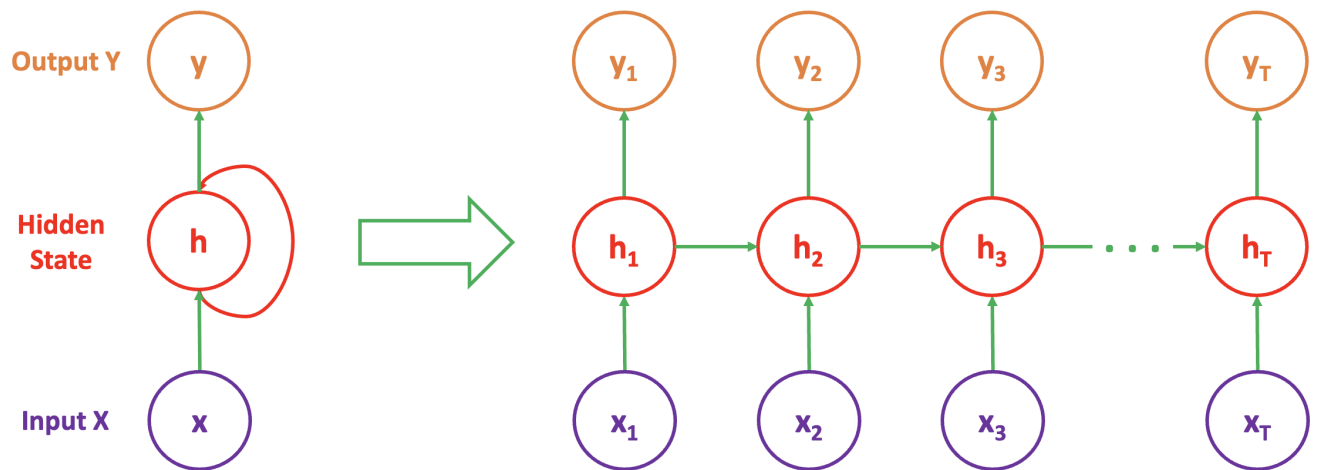


# Recurrent Neural Networks - Exercise

Ngày 24 tháng 11 năm 2023

## Phần I: Giới thiệu

**Recurrent Neural Networks (RNNs)** là một trong các loại mạng nơ-ron thường được dùng để giải quyết các bài toán có dữ liệu dạng time sequence (dạng chuỗi thời gian) như time series, text... Với cơ chế sử dụng output của thời điểm  $t - 1$  để dự đoán cho input của thời điểm  $t$  hiện tại, RNN duyệt qua từng thời điểm  $t$  trong mẫu dữ liệu, thực hiện tính toán theo cơ chế trên dẫn đến khả năng ghi nhớ được các thông tin trước đó.



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 Classification 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:

```
1 import torch
2 import torch.nn as nn
3
4 seed = 1
5 torch.manual_seed(seed)
6
7 import os
8 import numpy as np
9 import pandas as pd
10 import matplotlib.pyplot as plt
11 import re
12 import nltk
13 import unicodedata
14
15 nltk.download('stopwords')
16 from nltk.corpus import stopwords
17 from nltk.stem.porter import PorterStemmer
18
19 from torch.utils.data import Dataset, DataLoader
20 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 dự 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:

```

1 dataset_path = 'dataset/all-data.csv'
2 headers = ['sentiment', 'content']
3 df = pd.read_csv(
4     dataset_path,
5     names=headers,
6     encoding='ISO-8859-1'
7 )

```

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:

```

1 classes = {
2     class_name: idx for idx, class_name in enumerate(df['sentiment'].unique().tolist())
3 }
4 df['sentiment'] = df['sentiment'].apply(lambda x: classes[x])

```

4. **Tiền xử lý dữ liệu:** Để tiện 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:

```

1 english_stop_words = stopwords.words('english')
2 stemmer = PorterStemmer()
3
4 def text_normalize(text):
5     text = text.lower()
6     text = unicode.unidecode(text)
7     text = text.strip()
8     text = re.sub(r'^\w\s', '', text)
9     text = ' '.join([word for word in text.split(' ') if word not in
10                     english_stop_words])
11     text = ' '.join([stemmer.stem(word) for word in text.split(' ')])
12
13     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**:

```

1 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:

```

1 vocab = []
2 for sentence in df['content'].tolist():
3     tokens = sentence.split()
4     for token in tokens:
5         if token not in vocab:
6             vocab.append(token)
7
8 vocab.append('UNK')
9 vocab.append('PAD')
10 word_to_idx = {word: idx for idx, word in enumerate(vocab)}
11 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:

```

1 def transform(text, word_to_idx, max_seq_len):
2     tokens = []
3     for w in text.split():
4         try:
5             w_ids = word_to_idx[w]
6         except:
7             w_ids = word_to_idx['UNK']
8         tokens.append(w_ids)
9
10    if len(tokens) < max_seq_len:
11        tokens += [word_to_idx['PAD']] * (max_seq_len - len(tokens))
12    elif len(tokens) > max_seq_len:
13        tokens = tokens[:max_seq_len]
14
15    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:

```

1 val_size = 0.2
2 test_size = 0.125
3 is_shuffle = True
4 texts = df['content'].tolist()
5 labels = df['sentiment'].tolist()
6
7 X_train, X_val, y_train, y_val = train_test_split(
8     texts, labels,
9     test_size=val_size,
10    random_state=seed,
11    shuffle=is_shuffle
12 )
13
14 X_train, X_test, y_train, y_test = train_test_split(
15     X_train, y_train,
16     test_size=val_size,
17     random_state=seed,
18     shuffle=is_shuffle
19 )

```

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:

```

1 class FinancialNews(Dataset):
2     def __init__(
3         self,
4         X, y,
5         word_to_idx,
6         max_seq_len,
7         transform=None
8     ):
9         self.texts = X
10        self.labels = y
11        self.word_to_idx = word_to_idx

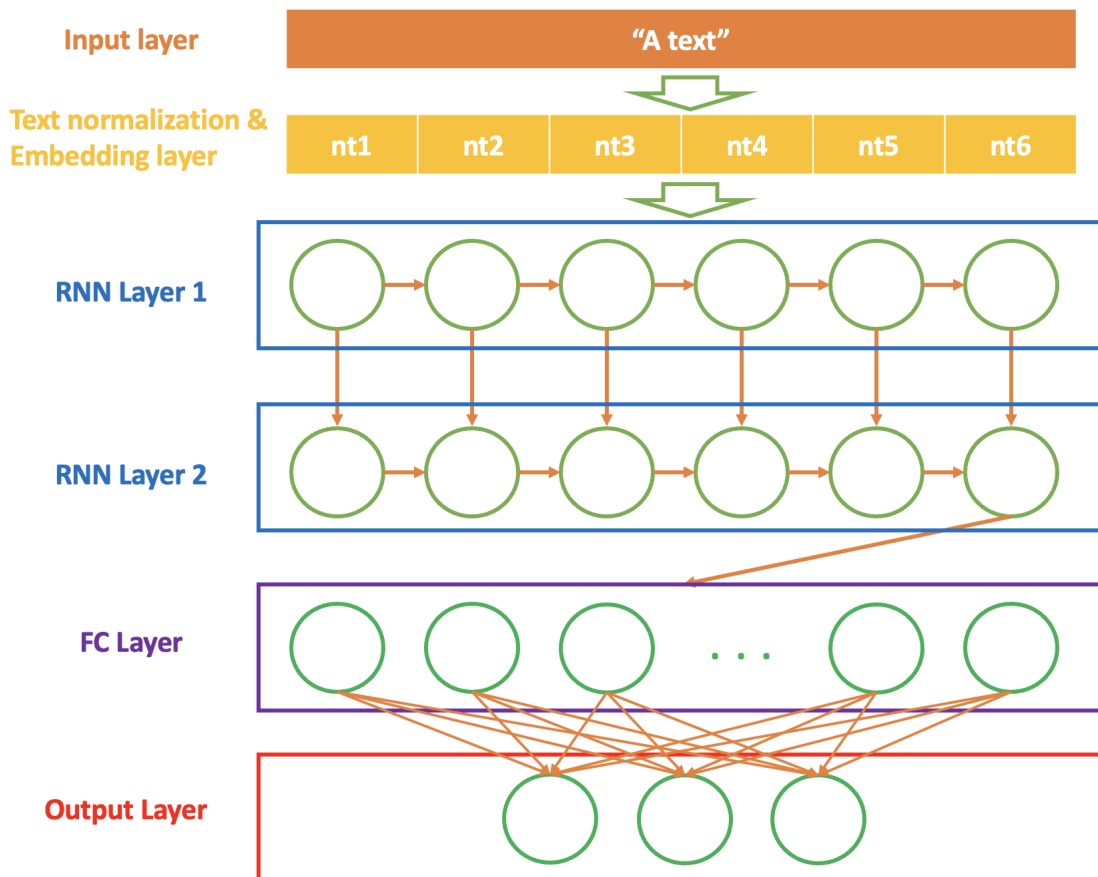
```

```
12         self.max_seq_len = max_seq_len
13         self.transform = transform
14
15     def __len__(self):
16         return len(self.texts)
17
18     def __getitem__(self, idx):
19         text = self.texts[idx]
20         label = self.labels[idx]
21
22         if self.transform:
23             text = self.transform(
24                 text,
25                 self.word_to_idx,
26                 self.max_seq_len
27             )
28         text = torch.tensor(text)
29
30         return text, label
```

## 8. Khai báo dataloader:

```
1 max_seq_len = 32
2
3 train_dataset = FinancialNews(
4     X_train, y_train,
5     word_to_idx=word_to_idx,
6     max_seq_len=max_seq_len,
7     transform=transform
8 )
9 val_dataset = FinancialNews(
10     X_val, y_val,
11     word_to_idx=word_to_idx,
12     max_seq_len=max_seq_len,
13     transform=transform
14 )
15 test_dataset = FinancialNews(
16     X_test, y_test,
17     word_to_idx=word_to_idx,
18     max_seq_len=max_seq_len,
19     transform=transform
20 )
21
22 train_batch_size = 128
23 test_batch_size = 8
24
25 train_loader = DataLoader(
26     train_dataset,
27     batch_size=train_batch_size,
28     shuffle=True
29 )
30 val_loader = DataLoader(
31     val_dataset,
32     batch_size=test_batch_size,
33     shuffle=False
34 )
35 test_loader = DataLoader(
36     test_dataset,
37     batch_size=test_batch_size,
38     shuffle=False
39 )
```

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:

```

1 class SentimentClassifier(nn.Module):
2     def __init__(
3         self, vocab_size, embedding_dim,
4         hidden_size, n_layers, n_classes,
5         dropout_prob
6     ):
7         super(SentimentClassifier, self).__init__()
8         self.embedding = nn.Embedding(vocab_size, embedding_dim)
9         self.rnn = nn.RNN(embedding_dim, hidden_size, n_layers, batch_first=
10             True)
11         self.norm = nn.LayerNorm(hidden_size)
12         self.dropout = nn.Dropout(dropout_prob)
13         self.fc1 = nn.Linear(hidden_size, 16)
14         self.relu = nn.ReLU()
15         self.fc2 = nn.Linear(16, n_classes)
16
17     def forward(self, x):
18         x = self.embedding(x)
19         x, hn = self.rnn(x)
20         x = x[:, -1, :]

```

```

20         x = self.norm(x)
21         x = self.dropout(x)
22         x = self.fc1(x)
23         x = self.relu(x)
24         x = self.fc2(x)
25
26         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`:

```

1 n_classes = len(list(classes.keys()))
2 embedding_dim = 64
3 hidden_size = 64
4 n_layers = 2
5 dropout_prob = 0.2
6 device = 'cuda' if torch.cuda.is_available() else 'cpu'
7
8 model = SentimentClassifier(
9     vocab_size=vocab_size,
10    embedding_dim=embedding_dim,
11    hidden_size=hidden_size,
12    n_layers=n_layers,
13    n_classes=n_classes,
14    dropout_prob=dropout_prob
15 ).to(device)

```

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

```

1 lr = 1e-4
2 epochs = 50
3
4 criterion = nn.CrossEntropyLoss()
5 optimizer = torch.optim.Adam(
6     model.parameters(),
7     lr=lr
8 )

```

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

```

1 def fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 ):
10     train_losses = []
11     val_losses = []
12
13     for epoch in range(epochs):
14         batch_train_losses = []
15

```

```

16     model.train()
17     for idx, (inputs, labels) in enumerate(train_loader):
18         inputs, labels = inputs.to(device), labels.to(device)
19
20         optimizer.zero_grad()
21         outputs = model(inputs)
22         loss = criterion(outputs, labels)
23         loss.backward()
24         optimizer.step()
25
26         batch_train_losses.append(loss.item())
27
28     train_loss = sum(batch_train_losses) / len(batch_train_losses)
29     train_losses.append(train_loss)
30
31     val_loss, val_acc = evaluate(
32         model, val_loader,
33         criterion, device
34     )
35     val_losses.append(val_loss)
36
37     print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {
38         val_loss:.4f}')
39
40     return train_losses, val_losses

```

```

1 def evaluate(model, dataloader, criterion, device):
2     model.eval()
3     correct = 0
4     total = 0
5     losses = []
6     with torch.no_grad():
7         for inputs, labels in dataloader:
8             inputs, labels = inputs.to(device), labels.to(device)
9             outputs = model(inputs)
10            loss = criterion(outputs, labels)
11            losses.append(loss.item())
12            _, predicted = torch.max(outputs.data, 1)
13            total += labels.size(0)
14            correct += (predicted == labels).sum().item()
15
16     loss = sum(losses) / len(losses)
17     acc = correct / total
18
19     return loss, acc

```

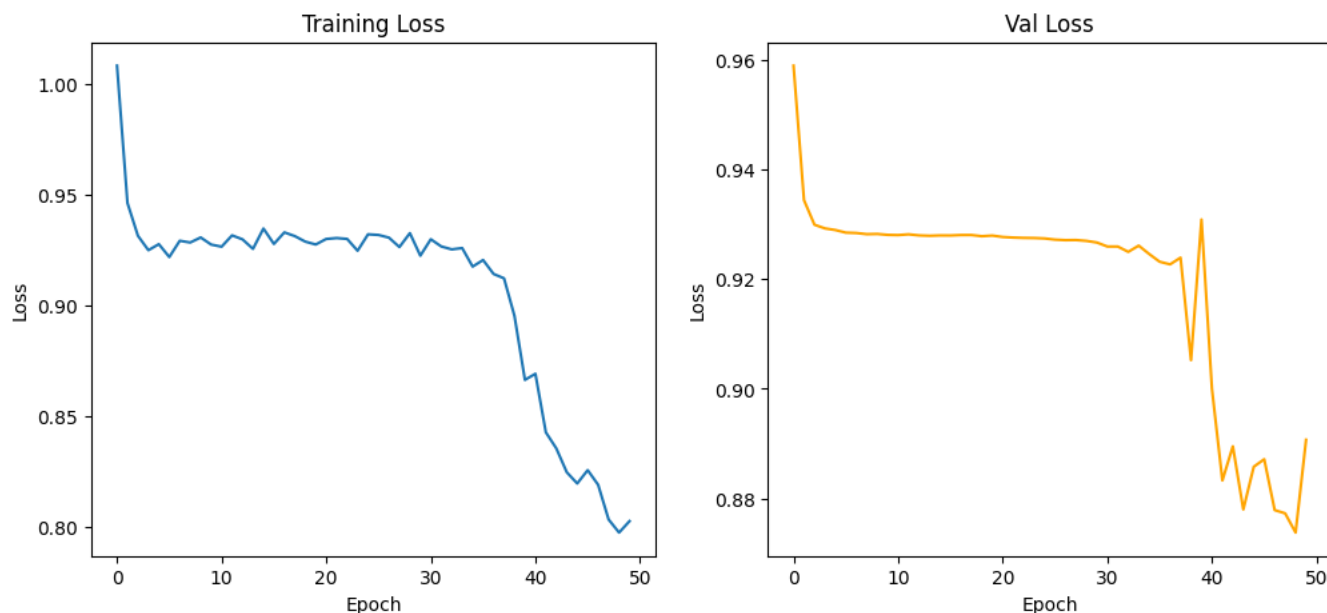
Tổng hợp tất cả các dữ kiện trên, ta tiến hành huấn luyện mô hình SentimentClassifier:

```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 )

```





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:

```
1 val_loss, val_acc = evaluate(  
2     model,  
3     val_loader,  
4     criterion,  
5     device  
6 )  
7 test_loss, test_acc = evaluate(  
8     model,  
9     test_loader,  
10    criterion,  
11    device  
12 )  
13  
14 print('Evaluation on val/test dataset')  
15 print('Val accuracy: ', val_acc)  
16 print('Test accuracy: ', test_acc)
```

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

## • 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:

```
1 import torch
2 import torch.nn as nn
3
4 seed = 1
5 torch.manual_seed(seed)
6
7 import numpy as np
8 import pandas as pd
9 import matplotlib.pyplot as plt
10
11 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:

```
1 dataset_filepath = 'dataset/weatherHistory.csv'
2 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:

```
1 univariate_df = df['Temperature (C)']
2 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
2006-04-01 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:

Time	Temperature (C)	
2006-04-01 00:00:00.000 +0200	9.472222	X
2006-04-01 01:00:00.000 +0200	9.355556	
2006-04-01 02:00:00.000 +0200	9.377778	
2006-04-01 03:00:00.000 +0200	8.288889	
2006-04-01 04:00:00.000 +0200	8.755556	
2006-04-01 05:00:00.000 +0200	9.222222	y
2006-04-01 06:00:00.000 +0200	7.733333	

Hình 3: 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,
6   offset):
7     features = []
8     labels = []
9
10    window_size = input_size + offset
11
12    if df_end_idx == None:
13        df_end_idx = len(df) - window_size
14
15    for idx in range(df_start_idx, df_end_idx):
16        feature_end_idx = idx + input_size
17        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

```

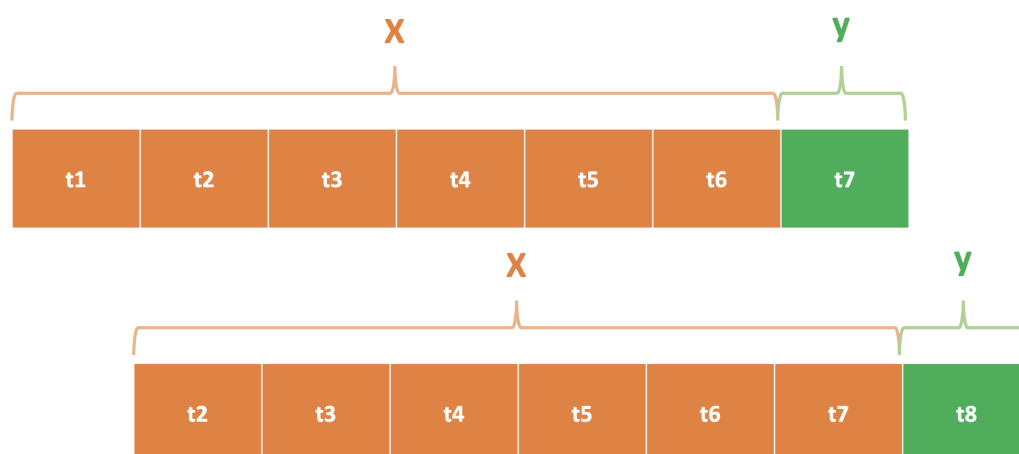
Ý nghĩa của các tham số đầu vào là:

- **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  $= \text{input\_size} + \text{offset}$ .
- **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 4: 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:

```

1 dataset_length = len(univariate_df)
2 train_size = 0.7
3 val_size = 0.2
4 train_end_idx = int(train_size * dataset_length)
5 val_end_idx = int(val_size * dataset_length) + train_end_idx
6
7 X_train, y_train = slicing_window(
8     univariate_df,
9     df_start_idx=0,
10    df_end_idx=train_end_idx,
11    input_size=input_size,
12    label_size=label_size,
13    offset=offset
14 )
15
16 X_val, y_val = slicing_window(
17     univariate_df,
18     df_start_idx=train_end_idx,
19     df_end_idx=val_end_idx,
20     input_size=input_size,
21     label_size=label_size,
22     offset=offset
23 )
24
25 X_test, y_test = slicing_window(
26     univariate_df,
27     df_start_idx=val_end_idx,
28     df_end_idx=None,
29     input_size=input_size,
30     label_size=label_size,
31     offset=offset
32 )

```

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

## 5. Xây dựng pytorch datasets:

```

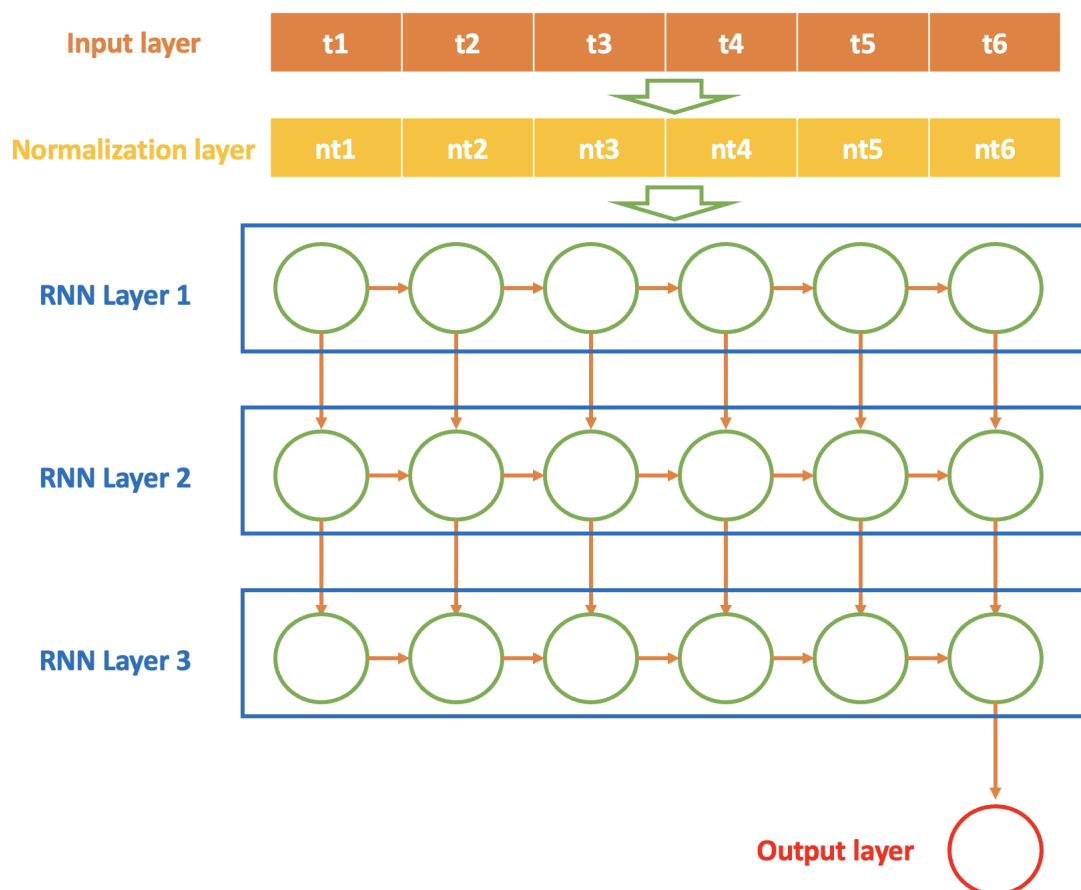
1 class WeatherForecast(Dataset):
2     def __init__(
3         self,
4         X, y,
5         transform=None
6     ):
7         self.X = X
8         self.y = y
9         self.transform = transform
10
11     def __len__(self):
12         return len(self.X)
13
14     def __getitem__(self, idx):
15         X = self.X[idx]
16         y = self.y[idx]
17
18         if self.transform:
19             X = self.transform(X)
20
21         X = torch.tensor(X, dtype=torch.float32)
22         y = torch.tensor(y, dtype=torch.float32)
23
24         return X, y

```

## 6. Khai báo dataloader:

```
1 train_dataset = WeatherForecast(  
2     X_train, y_train  
3 )  
4 val_dataset = WeatherForecast(  
5     X_val, y_val  
6 )  
7 test_dataset = WeatherForecast(  
8     X_test, y_test  
9 )  
10  
11 train_batch_size = 128  
12 test_batch_size = 8  
13  
14 train_loader = DataLoader(  
15     train_dataset,  
16     batch_size=train_batch_size,  
17     shuffle=True  
18 )  
19 val_loader = DataLoader(  
20     val_dataset,  
21     batch_size=test_batch_size,  
22     shuffle=False  
23 )  
24 test_loader = DataLoader(  
25     test_dataset,  
26     batch_size=test_batch_size,  
27     shuffle=False  
28 )
```

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 5: Ảnh minh họa kiến trúc của mô hình

```

1 class WeatherForecaster(nn.Module):
2     def __init__(
3         self, embedding_dim, hidden_size,
4         n_layers, dropout_prob
5     ):
6         super(WeatherForecaster, self).__init__()
7         self.rnn = nn.RNN(embedding_dim, hidden_size, n_layers, batch_first=
True)
8         self.norm = nn.LayerNorm(hidden_size)
9
10        self.dropout = nn.Dropout(dropout_prob)
11        self.fc = nn.Linear(hidden_size, 1)
12
13    def forward(self, x):
14        x, hn = self.rnn(x)
15        x = x[:, -1, :]
16        x = self.norm(x)
17        x = self.dropout(x)
18        x = self.fc(x)
19
20    return x

```

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

```

1 embedding_dim = 1
2 hidden_size = 8

```

```

3 n_layers = 3
4 dropout_prob = 0.2
5 device = 'cuda' if torch.cuda.is_available() else 'cpu'
6
7 model = WeatherForecaster(
8     embedding_dim=embedding_dim,
9     hidden_size=hidden_size,
10    n_layers=n_layers,
11    dropout_prob=dropout_prob
12 ).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.

```

1 lr = 1e-3
2 epochs = 50
3
4 criterion = nn.MSELoss()
5 optimizer = torch.optim.Adam(
6     model.parameters(),
7     lr=lr
8 )

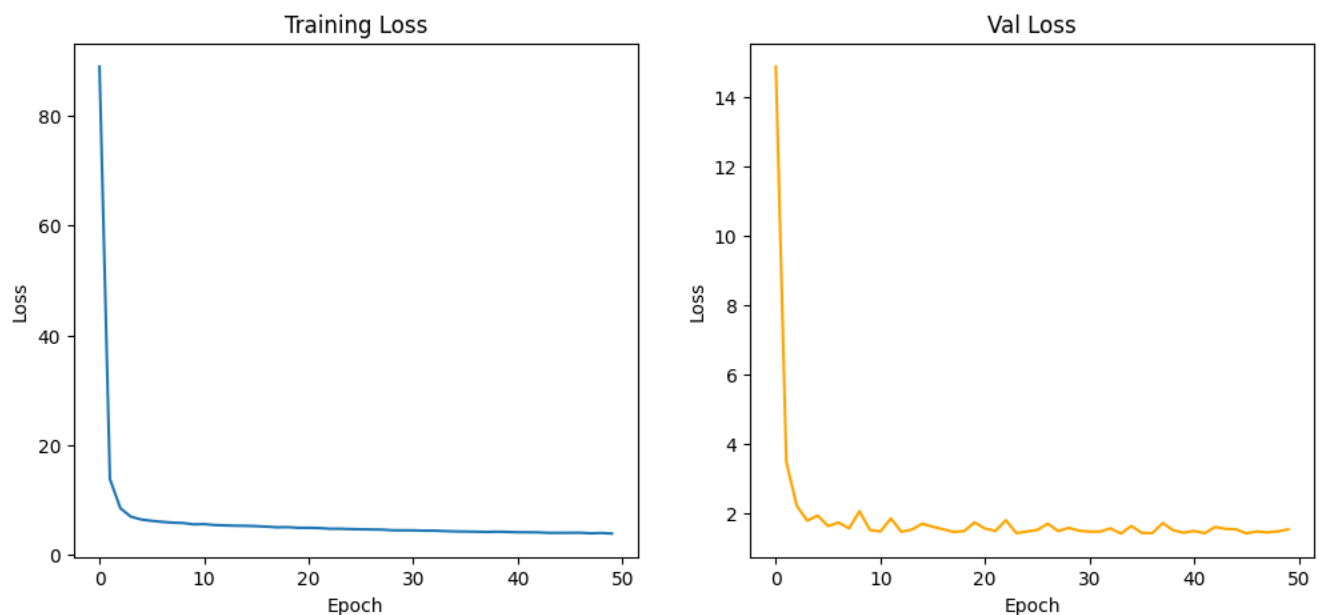
```

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:

```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 )

```



Hình 6: Trực quan hóa kết quả loss trong quá trình huấn luyện mô hình

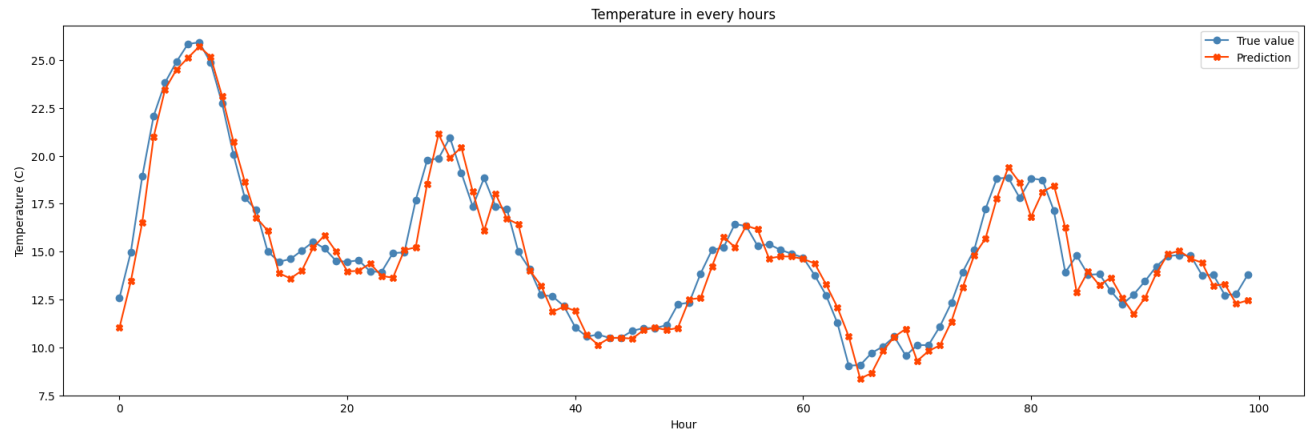


10. **Đánh giá mô hình:** Ta sử dụng hàm `evaluate()` để đánh giá mô hình đã huấn luyện trên tập val và test như sau:

```
1 val_loss = evaluate(  
2     model,  
3     val_loader,  
4     criterion,  
5     device  
6 )  
7 test_loss = evaluate(  
8     model,  
9     test_loader,  
10    criterion,  
11    device  
12 )  
13  
14 print('Evaluation on val/test dataset')  
15 print('Val loss: ', val_loss)  
16 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 trong một khoảng thời gian như sau:

```
1 def plot_difference(y, pred):  
2     plt.figure(figsize=(20, 6))  
3     times = range(len(y))  
4     y_to_plot = y.flatten()  
5     pred_to_plot = pred.flatten()  
6  
7     plt.plot(times, y_to_plot, color='steelblue', marker='o', label='True  
8     value')  
9     plt.plot(times, pred_to_plot, color='orangered', marker='X', label='  
10    Prediction')  
11  
12     plt.title('Temperature in every hours')  
13     plt.xlabel('Hour')  
14     plt.ylabel('Temperature (C)')  
15     plt.legend()  
16     plt.show()  
17  
18 inputs = torch.tensor(X_test[:100], dtype=torch.float32).to(device)  
19 model.eval()  
20 with torch.no_grad():  
21     outputs = model(inputs).detach().cpu().numpy()  
22 plot_difference(y_test[:100], outputs)
```



Hình 7: 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 ba version bao gồm RNN, LSTM và BiLSTM (các bạn tham khảo cách sử dụng LSTM/BiLSTM trong pytorch tại [đây](#)) với các config về tham số không đổi.

## B. Phần trắc nghiệm

1. Đâu không phải là một ứng dụng chính của RNN?
  - (a) Text Classification
  - (b) Stock Trading Prediction
  - (c) Image Classification
  - (d) Machine Translation
2. 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
3. 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
4. Đâu là điểm khác biệt chính giữa LSTM so với RNN?
  - (a) LSTM có ít hidden unit hơn RNN
  - (b) LSTM sử dụng activation function khác so với RNN
  - (c) LSTM sử dụng loss function khác so với RNN
  - (d) LSTM sử dụng cơ chế cổng để điều phối luồng thông tin xuyên suốt các unit
5. Phát biểu nào sau đây là hạn chế của RNN so với LSTM?
  - (a) Thời gian huấn luyện lâu
  - (b) Kiến trúc phức tạp hơn
  - (c) Vanishing Gradient
  - (d) Yêu cầu tài nguyên tính toán lớn
6. Phương án nào sau đây là khả thi để cải thiện hiệu năng của RNN?
  - (a) Dùng một optimizer khác
  - (b) Dùng một hàm loss khác
  - (c) Tăng số hidden units
  - (d) Tăng số lượng token biểu diễn input
7. Gọi  $h^t$ ,  $x^t$  lần lượt là hidden state và input token tại thời điểm thứ  $t$ .  $W_h$ ,  $W_x$  là trọng số của hidden state và input token.  $a_h()$  là ký hiệu của activation function. Khi đó, công thức tính hidden state  $h^t$  là:
  - (a)  $h^t = a_h(\sum_{i=1}^{t-1} W_h h^i + W_x x^t + b_h)$
  - (b)  $h^t = a_h(W_h h^{t-1} + W_x x^t + b_h)$

- (c)  $h^t = a_h(W_h h^{t-1} + W_x x^{t-1} + b_h)$   
 (d)  $h^t = a_h(W_h h^{t-1} + W_x x^t)$
8. Gọi  $h^t$  là hidden state tại thời điểm thứ  $t$ .  $W_y$  là trọng số của output.  $a_y()$  là ký hiệu của activation function. Khi đó, công thức tính output  $y$  tại thời điểm thứ  $t$  là:
- (a)  $y^t = a_y(W_y h^t)$   
 (b)  $y^t = a_y(W_y h^{t-1} + b_y)$   
 (c)  $y^t = a_y(\prod_{i=1}^{t-1} W_y h^i + b_y)$   
 (d)  $y^t = a_y(W_y h^t + b_y)$
9. Gọi  $y^t$  là output tại thời điểm thứ  $t$ ,  $L^t$  là hàm loss tính tại thời điểm thứ  $t$  và  $W_y$  là trọng số của output  $y$ . Khi đó, công thức tính backpropagation cho  $W_y$  tại thời điểm  $t = 5$  được tính bằng công thức:
- (a)  $\frac{\partial L^5}{\partial W_y} = \frac{\partial L^5}{\partial y^5} \frac{\partial y^5}{\partial W_y}$   
 (b)  $\frac{\partial L^5}{\partial W_y} = \frac{\partial L^5}{\partial y^5} \frac{\partial h^5}{\partial W_y} \frac{\partial h^5}{\partial W_y}$   
 (c)  $\frac{\partial L^5}{\partial W_y} = \sum_{i=1}^5 \frac{\partial L^i}{\partial y^i} \frac{\partial y^i}{\partial W_y}$   
 (d)  $\frac{\partial L^5}{\partial W_y} = \prod_{i=1}^5 \frac{\partial L^i}{\partial y^i} \frac{\partial y^i}{\partial W_y}$
10. Cho đoạn code sau:

```

1 import torch
2 import torch.nn as nn
3
4 input_tensor = torch.randn(5, 10, 10)
5
6 lstm = nn.LSTM(input_size=10, hidden_size=20, num_layers=2, batch_first=True)
7 bilstm = nn.LSTM(input_size=10, hidden_size=20, num_layers=2, batch_first=True,
8                 bidirectional=True)
9
10 output_lstm, (hidden_state_lstm, cell_state_lstm) = lstm(input_tensor)
11 output_bilstm, (hidden_state_bilstm, cell_state_bilstm) = bilstm(input_tensor)
12
13 print(output_lstm.shape)
14 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.

- Hết -