Bài 9 - Pytorch - Buổi 3 - torchtext module NLP

25 Aug 2019 - phamdinhkhanh

Menu

- 1. Giới thiệu về Torchtext
- 2. Khái quát
- 3. Khai báo trường.
- 3. Tạo tập dataset
- 4. Xây dựng các iterator
- 5. Đóng gói iterator
- 6. Huấn luyện mô hình
- 7. Tài liệu tham khảo

1. Giới thiệu về Torchtext

Như chúng ta đã biết, qui trình xây dựng một mô hình trong NLP sẽ đi qua các bước sau:

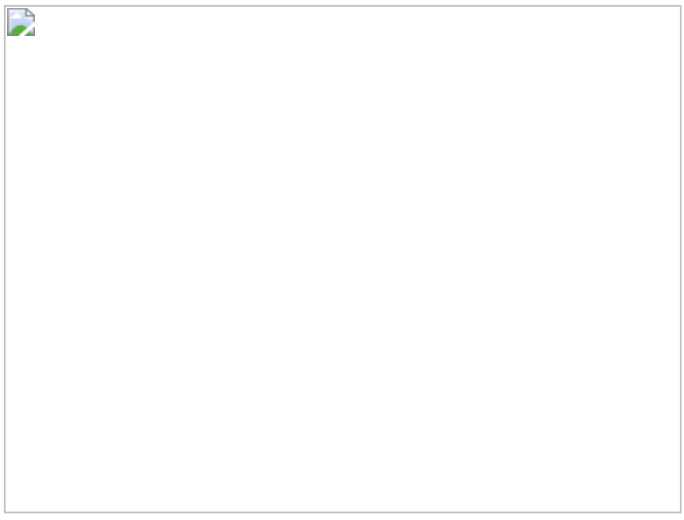
- Đọc dữ liệu văn bản từ ổ cứng.
- Tokenize dữ liệu text.
- Tạo từ điển mapping word sang index.
- Chuyển các câu sang list index.
- Padding dữ liệu bằng phần tử 0 để list các index về chung 1 độ dài.
- Xác định batch để truyền dữ liệu vào model.

Quá trình này đòi hỏi phải thực hiện tiền xử lý dữ liệu nhanh gọn và dễ dàng. Chính vì thế torchtext ra đời như là thư viện hỗ trợ quá trình tiền xử lý dữ liệu trở nên đơn giản hơn. Đặc biệt là các chức năng tạo batch và loading data lên GPU rất nhanh và tiện ích.

Trong ví dụ này chúng ta áp dụng torchtext để xử lý dữ liệu huấn luyện model phân loại văn bản. Dữ liệu được lấy tại practical torchtext data (https://github.com/keitakurita/practical-torchtext/blob/master/data) có nội dung về phân loại thái độ của comment. Dữ liệu này gồm 8 trường trong đó Id để xác định comment, comment_text là nội dung comment, 6 trường còn lại là mục đích của comment theo các loại (toxic: comment độc hại, severe toxic: cực kì độc hại, obscene: tục tĩu, threat: đe dọa, insult: lăng mạ, identity hate: ghét)

2. Khái quát

Hình bên dưới sẽ diễn tả quá trình mà torchtext hoạt động.



Hình 1: Qúa trình preprocessing data trên torchtext

Ta có thể hình dung torchtext như một preprocessing tool giúp chuyển hóa dữ liệu từ dạng thô nhất từ bất kì các nguồn nào: `txt, csv, json, tsv` để convert chúng sang Dataset.

Top

Dataset đơn giản là một khối dữ liệu với nhiều trường được load lên RAM để truyền vào model xử lý. Torchtext sẽ truyền những dataset này vào mỗi một vòng lặp (iterator). Trong một vòng lặp chúng ta sẽ thực hiện các biến đổi dữ liệu như: mã hóa số, padding data, tạo batch, và truyền dữ liệu lên GPU. Tóm lại torchtext sẽ thực hiện tất cả các biến đổi về dữ liệu để đưa chúng vào mạng nơ ron. Trong ví dụ bên dưới chúng ta cùng xem các quá trình dữ liệu hoạt động như thế nào.

3. Khai báo trường.

Khai báo trường nhằm mục đích nói cho dữ liệu biết chúng ta có những trường gì và được tạo ra từ dữ liệu như thế nào. Để khai báo trường chúng ta sử dụng class Field của torchtext. Xem ví dụ sau:

```
from torchtext.data import Field

tokenize = lambda x: x.split(' ')
TEXT = Field(sequential = True, tokenize = tokenize, lower = True)
LABEL = Field(sequential = False, use_vocab = False)

```

Trong tác vụ phân loại mục đích của comment, chúng ta có 6 nhãn (toxic, severe toxic, obscene, threat, insult, and identity hate).

Đầu tiên là trường LABEL. Chúng ta cần giữ nguyên các trường này và mapping chúng vào các số nguyên để tạo thành nhãn cho huấn luyện. Vì các nhãn này là các số nguyên chứ không phải list các index của nhãn nên sequential = False.

Tiếp theo TEXT sẽ là đoạn mô tả của sản phẩm. Do chúng là câu văn nên chúng ta phải mã hóa chúng về dạng list, do đó sequential = True. Hàm tokenize cho biết chúng ta tách câu sang token như thế nào. Khi áp dụng hàm x.split(‘’) có nghĩa rằng câu được chia thành các từ đơn. `lower = True` để chuyển chữ hoa thành chữ thường.

Bên dưới ta sẽ đọc dữ liệu: Mount folder trên google colab

```
from google.colab import drive
import os
drive.mount('/content/gdrive')
path = os.path.join('gdrive/My Drive/your_folder_path')
os.chdir(path)
```

Đọc dữ liệu

```
import pandas as pd

data = pd.read_csv('practical-torchtext/data/train.csv', header = 0, index_col = 0)
print('data.shape: ', data.shape)
data.head()
```

```
data.shape:  (25, 7)
```

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
id							
0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0

Top

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
id							
0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Thêm vào đó để trong xử lý ngôn ngữ chúng ta có thể áp dụng một số keyword đặc biệt. Khi đó class `Field` sẽ có một số tham số khai báo cho keyword như:

- `unk_token` : Token sử dụng cho các keyword không xuất hiện trong từ điển.
- `pad_token` : Token đại diện cho các vị trí padding câu.
- `init_token` : Đánh dấu bắt dầu câu.
- `eos_token` : Đánh dấu kết thúc câu.

Ngoài ra trong Field còn một số thuộc tính khác qui định dữ liệu là batch hay là sequence, khai báo độ dài câu được qui định trong thời gian chạy hay từ trước,….

Để hiểu thêm về các tham số của Field có thể tham khảo trong docstring (https://github.com/pytorch/text/blob/c839a7934930819be7e240ea972e4d600966afdc/torchtext/data/field.py#L61) của Field class đã được tác giả diễn giải rất chi tiết.

Có thể nói class Field chính là phần quan trọng nhất của torchtext có tác dụng giúp cho việc khởi tạo và xây dựng từ điển dễ dàng hơn.

Bên cạnh class Field, pytorch cũng hỗ trợ một vài dạng Field đặc biệt khác phù hợp với từng nhu cầu sử dụng khác nhau:

Dạng Field	**Mô tả**	**Trường hợp sử dụng**
Field	Là dạng field thông thường nhất áp dụng trong tiền xử lý dữ liệu	Sử dụng cho cả field dạng non-text dạng text trong TH chúng ta không cần map integers ngược lại các từ
ReversibleField	Mở rộng của Field cho phép map ngược lại từ index sang từ	Sử dụng cho text field khi ta muốn map ngược lại từ index sang từ
NestedField	Một trường biển đổi các văn bản sang tợp hợp nhỏ các Fields	Mô hình dựa trên character level
LabelField	Là một field thông thường trả về label cho trường	Sử dụng cho các trường Labels trong phân loại văn bản

3. Tạo tập dataset

Các fields sẽ cho ta biết chúng ta cần làm gì để biến đổi dữ liệu raw thành các trường. Còn dataset sẽ cho ta biết các trường dữ liệu được sử dụng như thế nào để huấn luyện mô hình.

Có rất nhiều các dạng Dataset khác nhau trong torchtext được sử dụng tương thích với các định dạng dữ liệu khác nhau. Chẳng hạn tsv/txt/csv file sẽ tương thích với class TabularDataset. Bên dưới chúng ta sẽ đọc dữ liệu từ csv file sử dụng TabularDataset.

Top

```
from torchtext.data import TabularDataset

# Khai báo thông tin fields thông qua các cặp ("field name", Field)
tv_datafields = [("id", None), # chúng ta không cần id nên gán trị của nó là None
                 ("comment_text", TEXT),
                 ("toxic", LABEL),
                 ("severe_toxic", LABEL),
                 ("threat", LABEL),
                 ("obscene", LABEL),
                 ("insult", LABEL),
                 ("identity_hate", LABEL)]

# Tạo dataset cho train và validation
train, valid = TabularDataset.splits(
               path="practical-torchtext/data", # root directory nơi chứa dữ liệu
               train='train.csv', validation="valid.csv",
               format='csv',
               skip_header=True, # khai báo header
               fields=tv_datafields # list các từ tương ứng với các Field được sử d
              )

# Khai báo test fields
test_datafields = [("id", None),
                   ("comment_text", TEXT)]

# Tạo dataset cho test
test = TabularDataset(
           path="practical-torchtext/data/test.csv",
           format='csv',
           skip_header=True,
           fields=test_datafields)
```

Chúng ta có 2 dạng biến đổi chính là LABEL và TEXT. Trong đó LABEL dành cho những biến category ở output và TEXT dành cho những biến dạng text cần được tokenize thành list các từ.

Kiểm tra kết quả được khởi tạo từ TabularDataset.

```
print('train[0]: ', train[0])
print('train[0].__dict__.keys(): ', train[0].__dict__.keys())
print('train[0].__dict__: ', train[0].__dict__)
```

```
train[0]:  <torchtext.data.example.Example object at 0x7fef75b8c0b8>
train[0].__dict__.keys():  dict_keys(['comment_text', 'toxic', 'severe_toxic', 'thre
train[0].__dict__:  {'comment_text': ['explanation\nwhy', 'the', 'edits', 'made', 'u
```

Example object là một tợp hợp các thuộc tính được tổng hợp trong dataset. Chúng ta thấy dataset đã được khởi tạo và các câu đã được tokenize thành các từ. Tuy nhiên chúng ta chưa thể map các câu thành từ và từ từ thành index do chưa khởi tạo mapping.

Torchtext sẽ quản lý map các từ với index tương ứng thông qua hàm `build_vocab()` tham số được truyền vào chính là các câu huấn luyện.

```
TEXT.build_vocab(train)
```

sau khi chạy hàm trên, torchtext sẽ duyệt qua toàn bộ các phần tử nằm trong train dataset, kiểm tra các dữ liệu tương ứng với `TEXT` field và thêm các từ vào trong từ điển của nó. Trong torchtext đã có class Vocab quản lý từ vựng. Vocab sẽ quản lý việc mapping các từ tới index thông qua tham số `stoi` và chuyển ngược mapping index sang từ bằng tham số `itos` . Ngoài ra Vocab cũng có thể xây dựng một ma trận embedding các từ từ rất nhiều các model pretrained như word2vec (http://mlexplained.com/2018/02/15/language-modeling-tutorial-in-torchtext-practical-torchtext-part-2/). Vocab cũng sử dụng các tham số như `max_size` và `min_freq` để xác định tối đa bao nhiêu từ trong từ điển và tần suất xuất hiện nhỏ nhất của 1 từ để nó được đưa vào từ điển. Những từ không xuất hiện trong từ điển sẽ được chuyển đổi thành `<unk>` .

Top

Bên dưới là danh sách loại Dataset khác nhau và định dạng dữ liệu mà chúng chấp nhận

Loại Dataset	**Mô tả**	**Trường hợp sử dụng**
TabularDataset	Lấy đường dẫn địa chỉ của các file csv/tsv và json files hoặc các python dictionaries	Cho bất kì trường hợp nào cần label các text
LanguageModelingDataset	Lấy đường dẫn địa chỉ của các file này như là input	Mô hình ngôn ngữ
TranslationDataset	Lấy đường dẫn có phần mở rộng của các file cho từng loại ngôn ngữ. Chẳng hạn nếu ngôn ngữ là tiếng anh thì file sẽ là 'hoge.en', French: 'hoge.fr', path='hoge', exts=('en','fr')	Mô hình dịch
SequenceTaggingDataset	Lấy đường dẫn tới 1 file với câu đầu vào và đầu ra tách biệt bởi các tabs	tagging câu

4. Xây dựng các iterator

Như chúng ta đã biết để truyền được các batch vào model chúng ta cần một class quản lý chúng. Trong torchvision và Pytorch sử dụng `DataLoaders` . Vì một số lý do mà torchtext đã đổi tên thành `Iterator` để phù hợp với đúng chức năng là tạo vòng lặp. Cả 2 class đều có tác dụng quản lý quá trình dữ liệu được truyền vào mô hình. Tuy nhiên `Iterator` của torchtext có một số chức năng được thiết kế đặc thù cho NLP.

Code bên dưới sẽ khởi tạo các `Iterators` cho dữ liệu train/test và validation.

```
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

import torch
from torch.jit import script, trace
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
import csv
import random
import re
import os
import unicodedata
import codecs
from io import open
import itertools
import math


USE_CUDA = torch.cuda.is_available()
device = torch.device("cuda" if USE_CUDA else "cpu")
```

```
from torchtext.data import Iterator, BucketIterator

train_iter, val_iter = BucketIterator.splits(
 (train, valid), # Truyền tập dữ liệu chúng ta muốn tạo vào iterator
 batch_sizes=(64, 64), # Kích thước batch size
 device=device, # Truyền vào device GPU được xác định thông qua hàm torch.device()
 sort_key=lambda x: len(x.comment_text), # sort dữ liệu theo trường nào
 sort_within_batch=False,
 repeat=False # Lấy dữ liệu không lặp lại dữ liệu
)

test_iter = Iterator(test, batch_size=64, device=device, sort=False, sort_within_ba
```

Top

Tham số `sort_within_batch` được thiết lập là True sẽ sắp xếp dữ liệu trong mỗi minibatch theo thứ tự giảm dần theo `sort_key` .

`BuckIterator` là một trong những `Iterator` mạnh nhất của Torchtext. Nó tự động shuffle và dồn các câu input thành các chuỗi có độ dài tương tự nhau bằng cách padding 0 vào bên phải. Độ dài của mỗi câu sẽ bằng với độ dài của câu lớn nhất.

Đối với dữ liệu testing, chúng ta không muốn trộn dữ liệu vì sẽ đưa ra các dự đoán khi kết thúc huấn luyên. Đây là lý do tại sao chúng ta sử dụng một `Iterator` tiêu chuẩn thay vì `BucketIterator` .

Dưới đây, một danh sách các Iterators mà Torchtext hiện đang hỗ trợ:

Tên Iterators	**Mô tả**	**Trường hợp sử dụng**
Iterator	Chạy vòng lặp qua toàn bộ dataset theo thứ tự của dataset	Dữ liệu test, hoặc các dữ liệu không cần xáo trộn thứ tự
BucketIterator	dồn dữ liệu về cùng 1 độ dài câu bằng nhau	Phân loại văn bản, tagging chuỗi,....
BPTTIterator	Được xây dựng cho các mô hình ngôn ngữ mà việc khởi tạo câu input bị trì hoãn theo từng timestep. Và đồng thời nó cũng biến đổi độ dài của BPTT (backpropagation through time). Xem thêm (http://mlexplained.com/2018/02/15/language-modeling-tutorial-in-torchtext-practical-torchtext-part-2/)	Mô hình ngôn ngữ

5. Đóng gói iterator

Hiện tại, iterator trả về một định dạng dữ liệu chuẩn là `torchtext.data.Batch` . Batch class có các đặc tính tương tự như Example với tợp hợp các dữ liệu từ mỗi field như là thuộc tính của nó. Điều này khiến chúng khó sử dụng khi tên trường thay đổi thì cần phải update lại code tương ứng.

Chính vì thế chúng ta sẽ sử dụng một tip nhỏ bằng cách wrap batch thành một tuple của 2 phần tử x và y. Trong đó x là biến độc lập và y là biến phụ thuộc.

```
class BatchWrapper:
      def __init__(self, dl, x_var, y_vars):
            self.dl, self.x_var, self.y_vars = dl, x_var, y_vars # we pass in the l

      def __iter__(self):
            for batch in self.dl:
                  # print('x_var: ', self.x_var)
                  # print('y_vars: ', self.y_vars)
                  x = getattr(batch, self.x_var) # we assume only one input in this
                  if self.y_vars is not None: # we will concatenate y into a single
                        y = torch.cat([getattr(batch, feat).unsqueeze(1) for feat i
                        # print('y size: ', y.size())
                  else:
                        y = torch.zeros((1))
                        # print('y size when y_vars is None: ', y.size())
                  yield (x, y)

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

train_dl = BatchWrapper(train_iter, "comment_text", ["toxic", "severe_toxic", "obsc
valid_dl = BatchWrapper(val_iter, "comment_text", ["toxic", "severe_toxic", "obscen
test_dl = BatchWrapper(test_iter, "comment_text", None)
```

Top

Những gì đã thực hiện ở đoạn code trên đó là chuyển hóa batch thành tuple của input và output

```
next(train_dl.__iter__())
```

```
(tensor([[ 63, 220, 368,  ..., 348,  81, 329],
         [552,  46,  61,  ..., 210, 674, 209],
         [  3,  37,   4,  ..., 541,  22,   6],
         ...,
         [  1,   1,   1,  ...,   1,   1,   1],
         [  1,   1,   1,  ...,   1,   1,   1],
         [  1,   1,   1,  ...,   1,   1,   1]], device='cuda:0'),
 tensor([[0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [1., 1., 0., 1., 1., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [1., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [1., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0.]], device='cuda:0'))
```

6. Huấn luyện mô hình

Bên dưới chúng ta sẽ cùng sử dụng model LSTM để huấn luyện mô hình phân loại văn bản. Trong module LSTM chúng ta cần xác định 3 tham số chính đó là:

- embedding size: Kích thước của embedding véc tơ để nhúng mỗi từ input.
- hidden_dim: Kích thước của hidden state véc tơ.
- number_layers: Một mạng LSTM sẽ bao gồm 1 chuỗi các layers liên tiếp nhau mà đầu ra của layer này là đầu vào của layer tiếp theo. Do đó chúng ta cần phải xác định số lượng các layer trong 1 mạng LSTM.

Đầu ra của mạng LSTM sẽ bao gồm:

- Encoder output: Là ma trận bao gồm các véc tơ hidden state tại layer cuối cùng được trả ra tại mỗi bước thời gian t và có kích thước (`max_length x batch_size x hidden_size`).
- hidden output: Là ma trận gồm các véc tơ hidden state của LSTM được trả ra tại mỗi layer có kích thước (`n_layers x batch_size x hidden_size`).
- cell output: Là ma trận của các cell state véc tơ được trả ra tại mỗi layer có kích thước (`n_layers x batch_size x hiden_size`).

Để hiểu rõ hơn về kiến trúc của mạng LSTM và đầu ra của mạng LSTM lại có kích thước như trên các bạn có thể tham khảo giới thiệu về mạng LSTM (https://phamdinhkhanh.github.io/2019/04/22/L%C3%BD_thuy%E1%BA%BFt_v%E1%BB%81_m%E1%BA%A1ng_LSTM.html

Top

```
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable

class SimpleLSTMBaseline(nn.Module):
    def __init__(self, hidden_dim, emb_dim=300, num_linear=1):
        super().__init__() # don't forget to call this!
        self.embedding = nn.Embedding(len(TEXT.vocab), emb_dim)
        self.encoder = nn.LSTM(emb_dim, hidden_dim, num_layers=1)
        self.linear_layers = []
        # Tạo 1 list gồm num_linear-1 các linear layer để project encoder output qu
        for _ in range(num_linear - 1):
            self.linear_layers.append(nn.Linear(hidden_dim, hidden_dim))
            self.linear_layers = nn.ModuleList(self.linear_layers)
        # Layer cuối cùng trả ra kết quả gồm 6 nodes.
        self.predictor = nn.Linear(hidden_dim, 6)

    def forward(self, seq):
        # encoder trả về 2 phần tử, dấu _ để gán cho các giá trị mà ta không sử dụn
        hdn, _ = self.encoder(self.embedding(seq))
        # Lấy feature là véc tơ hidden state tại bước cuối cùng.
        feature = hdn[-1, :, :]
        # project feature qua chuỗi layers và cuối cùng trả ra output dự báo.
        for layer in self.linear_layers:
          feature = layer(feature)
          preds = self.predictor(feature)
        return preds

em_sz = 100
nh = 500
nl = 3
model = SimpleLSTMBaseline(nh, emb_dim=em_sz, num_linear=nl)
model = model.to(device)
```

Bây h ta sẽ tạo một vòng lặp huấn luyện. Chúng ta có thể duyệt qua những Iterator được đóng gói và data sẽ được tự động truyền vào sau khi được đưa lên GPU và tham số hóa.

Top

```
import tqdm

opt = optim.Adam(model.parameters(), lr=1e-2)
loss_func = nn.BCEWithLogitsLoss()

epochs = 10

for epoch in range(1, epochs + 1):
    running_loss = 0.0
    running_corrects = 0
    model.train() # nhớ bật trạng thái là train. Khi đó mô hình có thể update các t
    for x, y in tqdm.tqdm(train_dl): # tạo vòng lặp đi qua wrapper của dữ liệu huấn
        # Nhớ đưa dữ liệu lên device để có thể training trên GPU
        x = x.to(device)
        y = y.to(device)
        # Cập nhật lại toàn bộ hệ số gradient về 0
        opt.zero_grad()

        preds = model(x)
        # Tính loss function
        loss = loss_func(y, preds).to(device)
        # Lan truyền ngược để cập nhật các tham số của mô hình
        loss.backward()
        # Cập nhật optimization sang bước tiếp theo
        opt.step()
        # Tổng của loss function qua các batch huấn luyện
        running_loss += loss.data * x.size(0)

    epoch_loss = running_loss / len(train)

    # Tính loss function trên tập validation
    val_loss = 0.0
    model.eval() # bật chế độ evaluation để tham số của mô hình không bị cập nhật.
    for x, y in valid_dl:
        preds = model(x)
        # Tính loss function
        loss = loss_func(y, preds)
        val_loss += loss.data * x.size(0)
    # Trả về giá trị loss function trung bình qua từng epoch huấn luyện.
    val_loss /= len(valid)
    print('Epoch: {}, Training Loss: {:.4f}, Validation Loss: {:.4f}'.format(epoch,

```

```
Epoch: 1, Training Loss: -17331.3613, Validation Loss: -12972.5557
Epoch: 2, Training Loss: -26293.1348, Validation Loss: -18848.7305
Epoch: 3, Training Loss: -38160.9180, Validation Loss: -26296.4727
Epoch: 4, Training Loss: -53191.8555, Validation Loss: -35586.1328
Epoch: 5, Training Loss: -71929.5703, Validation Loss: -47033.2656
Epoch: 6, Training Loss: -95008.3203, Validation Loss: -60955.9102
Epoch: 7, Training Loss: -123066.7891, Validation Loss: -77651.4453
Epoch: 8, Training Loss: -156701.6562, Validation Loss: -97493.2812
Epoch: 9, Training Loss: -196662.9688, Validation Loss: -120866.4688
Epoch: 10, Training Loss: -243723.7969, Validation Loss: -148217.6719
```

Tiếp theo chúng ta sẽ đánh giá mô hình

Top

```
import numpy as np

test_preds = []
for x, y in tqdm.tqdm(test_dl):
    preds = model(x)
    preds = preds.cpu().data.numpy()
    # Giá trị đầu ra thực tế của mô hình là logit nên ta sẽ pass giá trị dự báo vào
    preds = 1 / (1 + np.exp(-preds))
    test_preds.append(preds)
    test_preds = np.hstack(test_preds)
```

Kết quả dự báo

```
import pandas as pd
df = pd.read_csv("practical-torchtext/data/test.csv")
for i, col in enumerate(["toxic", "severe_toxic", "obscene", "threat", "insult", "id
    df[col] = test_preds[:, i]

df
```

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...	1.0	1.0	1.0	1.0	1.0	1.0
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...	1.0	1.0	1.0	1.0	1.0	1.0
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...	1.0	1.0	1.0	1.0	1.0	1.0
3	00017563c3f7919a	:If you have a look back at the source, the in...	1.0	1.0	1.0	1.0	1.0	1.0
4	00017695ad8997eb	I don't anonymously edit articles at all.	1.0	1.0	1.0	1.0	1.0	1.0

Như vậy qua bài hướng dẫn này chúng ta đã nắm được những kiến thức cơ bản về torchtext bao gồm:

- Cách thức biến đổi dữ liệu thông qua các Field.
- Khởi tạo một Dataset khai báo các trường thông tin, nguồn dữ liệu, tập train, tập test kèm theo cách thức biến đổi ở mỗi trường thông tin.
- Xây dựng một vocabulary map các keyword với index đối với các Field được tạo thành từ text để từ đó chuyển hóa câu văn sang list indexes phục vụ training.
- Khởi tạo 1 iterator quản lý quá trình truyền dữ liệu batch vào mô hình hồi qui.
- Xây dựng 1 baseline model LSTM nhằm phân loại các cảm xúc của comments. Khi xây dựng mô hình NLP sẽ có rất nhiều các tình huống chúng ta cần sử dụng torchtext để xử lý dữ liệu. Khi đó hi vọng bài hướng dẫn này sẽ phát huy tác dụng.

7. Tài liệu tham khảo

Và cuối cùng không thể thiếu là những tài liệu mà tôi đã sử dụng để tổng hợp lại thành bài viết này.

1. torchtext docs (https://torchtext.readthedocs.io/en/latest/data.html)
2. how to use torchtext for ML translation (https://towardsdatascience.com/how-to-use-torchtext-for-neural-machine-translation-plus-hack-to-make-it-5x-faster-77f3884d95)
3. torchtext sentiment analysis (https://medium.com/@sonicboom8/sentiment-analysis-torchtext-55fb57b1fab8)
4. Comprehensive tutorial torchtext (https://mlexplained.com/2018/02/08/a-comprehensive-tutorial-to-torchtext/)

Top