Laporan Tutorial Deep Learning untuk Multimedia :
Text Classification with TorchText

Disusun Oleh:

Indiana Namaul Husnah 5024201061

DEPARTEMEN TEKNIK KOMPUTER

FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

2023

DAFTAR ISI

1. Text classification with the torchtext library
2. Access to the raw dataset iterators
3. Prepare data processing pipelines
4. Generate data batch and iterator
5. Define the model
6. Initiate an instance
7. Define functions to train the model and evaluate results.
8. Split the dataset and run the model
9. Evaluate the model with test dataset
10. Test on a random news

Text Classification with The Torchtext Library

Menggunakan torchtext library untuk membangun dataset text classification. Unuk melakukan klasifikasi text dengan torchtext, harus melalui step-step yaitu mengakses raw dataset, mempersiapkan pipelines, menentukan data batch, mendefinisikan model, fungsi training dan fungsi evaluasi, evaluasi model, dan melakukan tes pada text baru.

Pada laporan text classification ini menggunakan dataset dari `IMDB`. IMDB dataset merupakan dataset yang berisikan mengenai review film dan televisi yang terdapat pada IMDb (Internet Movie Database). IMDb berisikan informasi mengenai :

1. Informasi film dan televisi, seperti judul, tahun rilis, durasi, genre, rating, sinopsis, dan lain-lain
2. Informasi pemain film dan kru, seperti nama, peran, dan informasi lain mengenai karir pemeran
3. Informasi produksi film dan produksi, seperti studio produksi, lokasi syuting, dan lain-lain

Pada laporan text classification ini menggunakan model deep learning `'nn.Embeddingdbag'` atau Embedded Bag of Words untuk melakukan klasifikasi sentimen pada teks. Teks ini akan direpresentasikan dalam bentuk "bag-of-words", tiap kata pada dokumen direpresentasikan dalam bentuk one-hot encoding. Kemudian dokumen direpresentasikan dalam bentuk vektor dengan cara menjumlahkan seluruh vektor one-hot encoding dari setiap kata dalam dokumen.

Kemudian menggunakan tokenizer dari `Treebank Word Tokenizer`. Tokenizer ini merupakan tokenizer pada NLTK atau Natural Language Toolkit, yang memecah teks menjadi token-token dengan aturan tertentu.

Access to The Raw Dataset Iterators

Torchtext merupakan library untuk memproses data text memudahkan proses NLP (Neutral Language Programming). Library torchtext melakukan memuat, memproses, mengelola data text, serta mengkonverssi raw text menjadi representasi numerik, membagi data menjadi batch, membuat vocabulaties, dan membuat data dalam bentuk iterator untuk pelatihan model.

Untuk mengakses dataset, dapat menginstall torchdata di https://github.com/pytorch/data.

```
!pip uninstall -y torch torchdata torchvision torchtext torchaudio fastai
!pip install portalocker
!pip install --pre torch torchdata -f https://download.pytorch.org/whl/nightly/cpu/torch_nigh
!pip install torchtext
```

```
Requirement already satisfied: urllib3>=1.25 in /usr/local/lib/python3.9/dist-package
Requirement already satisfied: wheel in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: lit in /usr/local/lib/python3.9/dist-packages (from tr
Requirement already satisfied: cmake in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.9/dist-packages
Installing collected packages: nvidia-nvtx-cu11, nvidia-nccl-cu11, nvidia-cusparse-cu
  Attempting uninstall: torch
    Found existing installation: torch 2.1.0.dev20230404+cpu
    Uninstalling torch-2.1.0.dev20230404+cpu:
      Successfully uninstalled torch-2.1.0.dev20230404+cpu
  Attempting uninstall: torchdata
    Found existing installation: torchdata 0.7.0.dev20230404
    Uninstalling torchdata-0.7.0.dev20230404:
      Successfully uninstalled torchdata-0.7.0.dev20230404
```

Pada tahap penginstallan torchdata di atas, dilakukan tahap sebagai berikut.

- Menguninstall torch torchdata torchvision torchtext torchaudio fastai untuk memastikan instalasi yang digunakan adalah instalasi terbaru.
- Perintah !pip install portalocker akan menginstal perpustakaan portalocker yang digunakan untuk mengunci akses file pada sistem operasi Windows dan Linux.
- Pytorch biasa merupakan versi lebih stabil untuk dilakukan pengujian dan validasi, tidak memiliki kesalahan yang signifikan dalam penggunaannya. Versi nightly digunakan dalam menguji fitur baru dan perbaikan bug.

```
import os
from functools import partial
from pathlib import Path
from typing import Tuple, Union

from torchdata.datapipes.iter import FileOpener, IterableWrapper
from torchtext._download_hooks import HttpReader
from torchtext._internal.module_utils import is_module_available
from torchtext.data.datasets_utils import _create_dataset_directory
from torchtext.data.datasets_utils import _wrap_split_argument

URL = "http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"

MD5 = "7c2ac02c03563afcf9b574c7e56c153a"

NUM_LINES = {
    "train": 25000,
    "test": 25000,
```

```
}

MAP_LABELS = {"neg": 1, "pos": 2}

_PATH = "aclImdb_v1.tar.gz"

DATASET_NAME = "IMDB"
```

Tahap selanjutnya memuat dan mengolah dataset. Pada code di atas dapat dibedah library yang digunakan adalah sebagai berikut.

- `'os'` untuk mengakses fitur sistem direktori
- `'functools.partial'` untuk membuat versi baru dari fungsi yang sudah ada dengan mengganti beberapa nilai
- `'pathlib.Path'` mengakses dan manipulasi direktori
- `'typing.Tuple'` dan `'typing.Union'` menunjukkan jenis parameter
- `'FileOpener'` dan `'IterableWrapper'` adalah kelas-kelas yang disediakan oleh torchdata untuk membantu memuat dan mengolah data.
- `_create_dataset_directory` dan `_wrap_split_argument` adalah fungsi-fungsi utilitas internal torchtext yang digunakan untuk mengatur dataset dan memformat argumen split. Fungsi ini membantu memudahkan dan menstandardisasi pengaturan dataset dan pemrosesan data.

```
import torch
from torchtext.datasets import IMDB
train_iter = iter(IMDB(split='train'))
```

Prepare Data Processing Pipelines

Berikut merupakan mempersiapkan dataset. Pada code berikut ini menggunakan fungsi get_tokenizer. Get_tokenizer adalah fungsi dari torchtext.data.utils yang mengembalikan tokenizer yang telah ditentukan sebelumnya. Tokenizer digunakan untuk memisahkan teks menjadi token yang dapat diolah lebih lanjut.Pada code berikut kita menggunakan tokenizer `Treebank Word Tokenizer`. Tokenizer ini merupakan tokenizer pada NLTK atau Natural Language Toolkit, yang memecah teks menjadi token-token dengan aturan tertentu.

```
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator

!pip install nltk
import nltk
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
```

```
train_iter = IMDB(split='train')

def yield_tokens(data_iter):
    for _, text in data_iter:
        yield tokenizer.tokenize(text)

vocab = build_vocab_from_iterator(yield_tokens(train_iter), specials=["<unk>"])
vocab.set_default_index(vocab["<unk>"])
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: nltk in /usr/local/lib/python3.9/dist-packages (3.8.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: joblib in /usr/local/lib/python3.9/dist-packages (from nl
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages (from nlt
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from nltk
```

Dalam kode yang Anda tampilkan di atas digunakan untuk membangun kamus vocab dari dataset IMDB. Tahapnya yaitu pengambilan token dari dataset, kemudian dilakukan pembangunan vocab dari token tersebut.

`'build_vocab_from_iterator'` untuk mengembalikan objek vocab yang dibangun dari iterator token. `'yield_tokens'` digunakan untuk memperoleh token dari dataset dari 'train_iter'

```
vocab(['here', 'is', 'an', 'example'])
```

```
[177, 7, 44, 487]
```

Setelah dilakukan tokenizer di atas, selanjutnya vocabulary blocks bisa terconvert menjadi token-token dalam integer.

```
text_pipeline = lambda x: vocab(tokenizer.tokenize(x))
label_pipeline = lambda x: int(x) - 1
```

text_pipeline = lambda x: vocab(tokenizer(x)): Fungsi lambda ini menerima sebuah teks dan memprosesnya menjadi suatu daftar token menggunakan tokenizer yang telah ditentukan sebelumnya. Selanjutnya, fungsi ini akan mengonversi setiap token menjadi suatu bilangan bulat yang merepresentasikan indeks token tersebut pada kamus vocab yang telah dibangun sebelumnya. Oleh karena itu, output dari fungsi ini adalah suatu daftar bilangan bulat yang merepresentasikan teks yang telah diproses.

label_pipeline = lambda x: int(x) - 1: Fungsi lambda ini menerima sebuah label dan mengonversinya menjadi bilangan bulat dengan mengurangi 1 dari nilai label tersebut. Dalam konteks dataset IMDB

movie review, labelnya adalah 0 atau 1 yang merepresentasikan sentimen positif atau negatif dari review tersebut. Dengan demikian, fungsi ini akan mengonversi label 1 menjadi 0 dan label 2 menjadi 1.

Kedua fungsi lambda tersebut digunakan untuk memproses teks dan label pada dataset IMDB movie review, sehingga nantinya dapat diolah lebih lanjut untuk pelatihan model. Fungsi text_pipeline akan digunakan untuk mengonversi teks menjadi suatu representasi numerik yang dapat diproses oleh model, sementara fungsi label_pipeline akan digunakan untuk mengonversi label menjadi suatu bilangan bulat yang merepresentasikan kelas sentimen.

```
text_pipeline('here is the an example')
```

```
[177, 7, 1, 44, 487]
```

```
label_pipeline('10')
```

```
9
```

Generate Data Batch and Iterator

Setelah melakukan preparing data, selanjutnya melakukan pengolahan data batch untuk pelatihan model dengan memubuat variabel dataloader. Pada tahap ini menggunakan `'Data Loader'` library untuk memuat data dalam bentuk batch.

Pada proses ini, setiap label akan diproses dengan label_pipeline, yaitu diubah menjadi bilangan bulat dan dikumpulkan ke dalam sebuah list label_list. Selanjutnya, setiap teks akan diproses dengan text_pipeline, yaitu di-tokenisasi dan diubah menjadi urutan bilangan bulat sesuai dengan kamus yang telah dibangun sebelumnya. Setiap urutan tersebut akan dikumpulkan ke dalam sebuah list text_list.

Entri text pada data batch dikemas secara list dan digabungkan ke tensor tunggal untuk input dari `'nn.EmbeddingBag'`

```
from torch.utils.data import DataLoader
device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # pengecekan
# device CPU ataU CUDA

def collate_batch(batch): # pengolahan data batch
    label_list, text_list, offsets = [], [], [0]
    for (_label, _text) in batch:
         label_list.append(label_pipeline(_label))
         processed_text = torch.tensor(text_pipeline(_text), dtype=torch.int64)
         text_list.append(processed_text)
         offsets.append(processed_text.size(0))
```

```
    label_list = torch.tensor(label_list, dtype=torch.int64)
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0) # mengatasi perbedaan panjang text tia
    text_list = torch.cat(text_list)
    return label_list.to(device), text_list.to(device), offsets.to(device) # data diproses
    # dan dikumpulkan ke daam list ini

train_iter = IMDB(split='train')
dataloader = DataLoader(train_iter, batch_size=8, shuffle=False, collate_fn=collate_batch)
```

Double-click (or enter) to edit

Define The Model

Pada bagian ini, kita mendefinisikan arsitektur model yang akan digunakan untuk melakukan klasifikasi sentimen pada teks. Model yang digunakan terdiri dari lapisan nn.EmbeddingBag yang akan digunakan untuk mengekstraksi fitur dari teks, dan lapisan linear untuk tujuan klasifikasi.

Lapisan nn.EmbeddingBag dengan mode default "mean" akan menghitung nilai rata-rata dari "kantung" embedding. Meskipun entri teks memiliki panjang yang berbeda, modul nn.EmbeddingBag tidak memerlukan padding karena panjang teks disimpan dalam offsets.

Selain itu, karena nn.EmbeddingBag mengakumulasi rata-rata di seluruh embedding secara dinamis, nn.EmbeddingBag dapat meningkatkan kinerja dan efisiensi memori untuk memproses urutan tensor. Gambar arsitektur model juga diberikan untuk memberikan gambaran visual tentang bagaimana model ini bekerja.

```
from torch import nn

class TextClassificationModel(nn.Module):

    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=False)
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc(embedded)
```

Initiate an Instance

Implementasi model klasifikasi text pada dataset `IMDB` pada variabel model. Dengan mengambil data training, menghitung jumlah kelas pada dataset, mengukur ukuran vocalubary, menentukan emsize yaitu 64, dan jumlah kelas.

```
train_iter = IMDB(split='train') # merepresentasikan data training dari dataset
num_class = len(set([label for (label, text) in train_iter])) # menghitung jml
# kelas pada dataset
vocab_size = len(vocab) # ukuran vocabulary pada dataset
emsize = 64 # ukuran embedding
model = TextClassificationModel(vocab_size, emsize, num_class).to(device)
```

Define Functions to Train The Model and Evaluate Results.

Mendefinisikan fungsi untuk training model dan evaluasi model dataset IMDB. Pada training data, fungsi akan melakukan forward pass dan backward pass pada model, kemudian menggunakan loss function 'criterion', kemudian melakukan update pada parameter model dengan menggunakan optimizer.

Pada evaluasi model, fungsi akan melakukan forward pass dan menghitung loss function menggunakan fungsi 'criterion'

```
import time

def train(dataloader):
    model.train()
    total_acc, total_count = 0, 0
    log_interval = 500
    start_time = time.time()

    for idx, (label, text, offsets) in enumerate(dataloader):
        optimizer.zero_grad()
        predicted_label = model(text, offsets)
        loss = criterion(predicted_label, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
        optimizer.step()
        total_acc += (predicted_label.argmax(1) == label).sum().item()
        total_count += label.size(0)
        if idx % log_interval == 0 and idx > 0:
            elapsed = time.time() - start_time
            print('| epoch {:3d} | {:5d}/{:5d} batches '
                  '| accuracy {:8.3f}'.format(epoch, idx, len(dataloader),
                                              total_acc/total_count))
```

```
            total_acc, total_count = 0, 0
            start_time = time.time()

def evaluate(dataloader):
    model.eval()
    total_acc, total_count = 0, 0

    with torch.no_grad():
        for idx, (label, text, offsets) in enumerate(dataloader):
            predicted_label = model(text, offsets)
            loss = criterion(predicted_label, label)
            total_acc += (predicted_label.argmax(1) == label).sum().item()
            total_count += label.size(0)
    return total_acc/total_count
```

Split The Dataset and Run The Model

Dataset `IMDB` akan displit menjadi data train dan data valid. menggunakan library pada link berikut. torch.utils.data.dataset.random_split_

Untuk menghitung loss pada model klasifikasi, menggunakan CrossEntropyLoss_ yang mana class ini menggabungkan `nn.LogSoftmax()` and `nn.NLLLoss()`

Selanjutnya untuk optimizer menggunakan SGD (Stochastic Gradient Descent) SGD_ dan mengatur learning rate 5.0

Kemudian untuk menyesuaikan learning rate menggunakan StepLR_ melalui iterasi epoch.

```
from torch.utils.data.dataset import random_split
from torchtext.data.functional import to_map_style_dataset
# Hyperparameters
EPOCHS = 10 # epoch
LR = 5  # learning rate
BATCH_SIZE = 64 # batch size for training

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1.0, gamma=0.1)
total_accu = None
train_iter, test_iter = IMDB()
train_dataset = to_map_style_dataset(train_iter)
test_dataset = to_map_style_dataset(test_iter)
num_train = int(len(train_dataset) * 0.95)
split_train_, split_valid_ = \
    random_split(train_dataset, [num_train, len(train_dataset) - num_train])

train_dataloader = DataLoader(split_train_, batch_size=BATCH_SIZE,
                              shuffle=True, collate_fn=collate_batch)
```

```
valid_dataloader = DataLoader(split_valid_, batch_size=BATCH_SIZE,
                              shuffle=True, collate_fn=collate_batch)
test_dataloader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
                             shuffle=True, collate_fn=collate_batch)

for epoch in range(1, EPOCHS + 1):
    epoch_start_time = time.time()
    train(train_dataloader)
    accu_val = evaluate(valid_dataloader)
    if total_accu is not None and total_accu > accu_val:
      scheduler.step()
    else:
       total_accu = accu_val
    print('-' * 59)
    print('| end of epoch {:3d} | time: {:5.2f}s | '
          'valid accuracy {:8.3f} '.format(epoch,
                                           time.time() - epoch_start_time,
                                           accu_val))
    print('-' * 59)
```

```
-----------------------------------------------------------
| end of epoch   1 | time: 43.09s | valid accuracy    0.690
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   2 | time: 41.94s | valid accuracy    0.644
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   3 | time: 41.90s | valid accuracy    0.800
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   4 | time: 46.09s | valid accuracy    0.800
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   5 | time: 42.72s | valid accuracy    0.805
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   6 | time: 51.96s | valid accuracy    0.814
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   7 | time: 44.58s | valid accuracy    0.818
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   8 | time: 42.14s | valid accuracy    0.821
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   9 | time: 45.32s | valid accuracy    0.819
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  10 | time: 49.15s | valid accuracy    0.823
-----------------------------------------------------------
```

Evaluate The Model with Test Dataset

```
print('Checking the results of test dataset.')
accu_test = evaluate(test_dataloader)
print('test accuracy {:8.3f}'.format(accu_test))
```

```
Checking the results of test dataset.
test accuracy    0.834
```

Test on a Random News

```
IMDB_label = {1: "neg", 2: "pos"}

def predict(text, text_pipeline):
    with torch.no_grad():
        text = torch.tensor(text_pipeline(text))
        output = model(text, torch.tensor([0]))
        return output.argmax(1).item() + 1

ex_text_str1 = "I was fortunate enough to see this movie on pre-release last night and, thoug
ex_text_str2 = "not unexpected, this is too mainstream"

model = model.to("cpu")

print("This is a %s news" %IMDB_label[predict(ex_text_str1, text_pipeline)])
print("This is a %s news" %IMDB_label[predict(ex_text_str2, text_pipeline)])
```

```
This is a pos news
This is a neg news
```