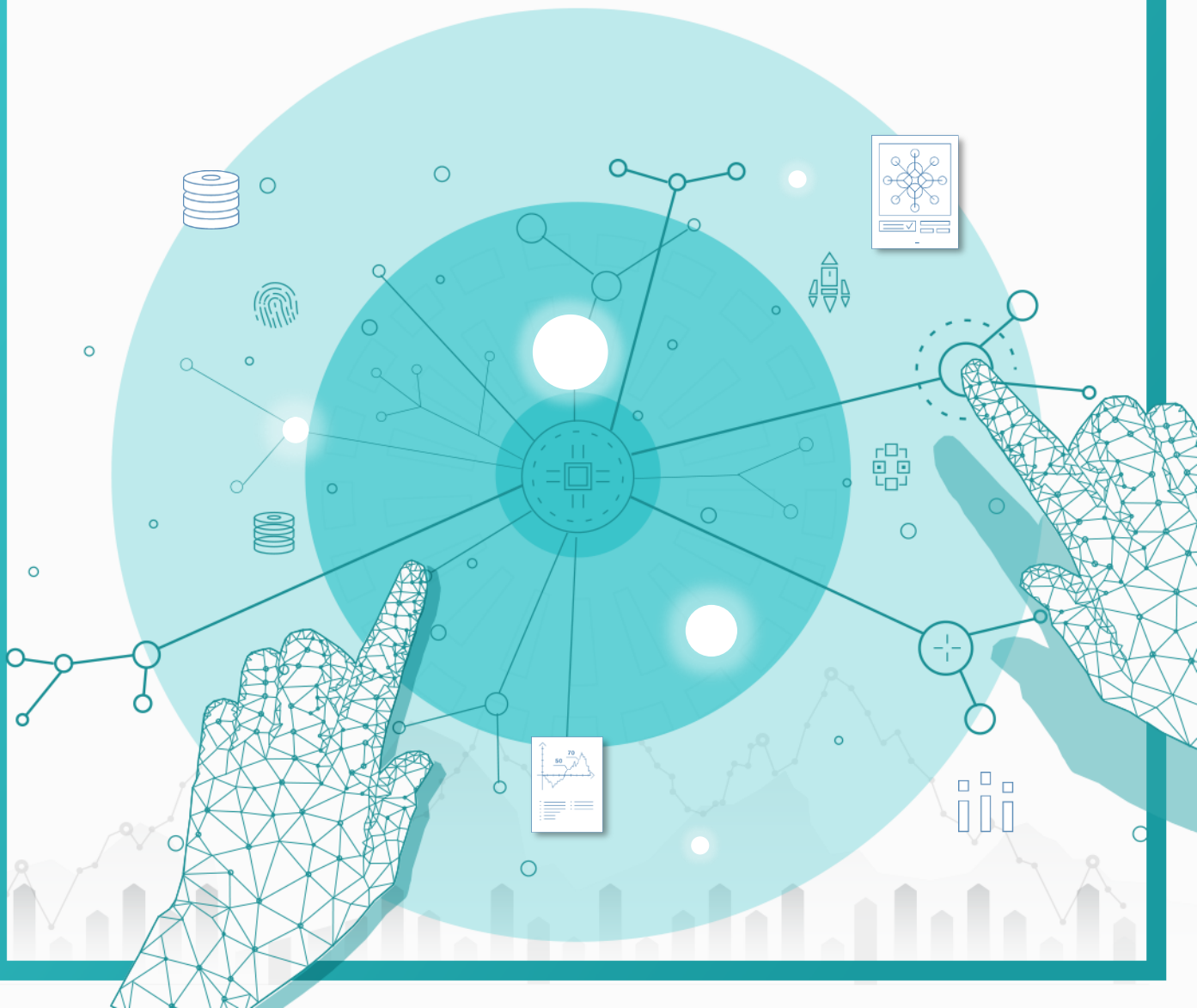
한국기술교육대학교
온라인평생교육원

파이썬을 활용한 인공지능 자연어 처리(실습)

자연어 처리를 위한 RNN



자연어 처리를 위한 RNN

학습 목표

1. RNN 데이터 입력을 위한 데이터 로더를 구현할 수 있다.
2. PyTorch를 이용하여 LSTM/GRU 모델을 활용할 수 있다.

학습 내용

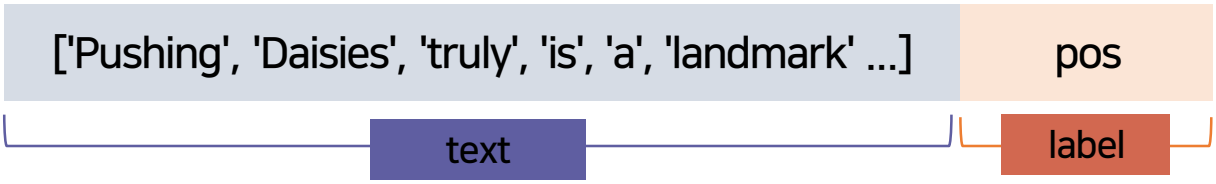
1. RNN 데이터 입력을 위한 데이터 로더 구현
2. LSTM/GRU 모델 활용

1. RNN 데이터 입력을 위한 데이터 로더 구현

1) field 지정

(1) torchtext를 이용한 field 객체 생성

- text field 객체 및 label field 객체 생성



- text: IMDB 데이터 세트 영화평
- label: IMDB 데이터 세트 긍정·부정

```
from torchtext.legacy.data import Field

text_field = Field(sequential=True, include_lengths=True, fix_length=200)
label_field = Field(sequential=False)
```

1. RNN 데이터 입력을 위한 데이터 로더 구현

2) 데이터 세트 생성

(1) torchtext를 이용한 IMDB 데이터 세트 로드

- IMDB 데이터 세트 로드
  - ① text_field와 label_field에 할당
  - ② split( ) 함수를 이용하여 train, test 데이터 세트로 분리

```
from torchtext.legacy.datasets import IMDB

train, test = IMDB.splits(text_field, label_field)
```

- train 데이터 세트의 데이터 및 레이블 출력
  - ① train 데이터 세트의 첫 번째 데이터 출력
  - ② train 데이터 세트의 첫 번째 레이블 출력

```
print(vars(train.examples[0]))
print(vars(train.examples[0])['label'])
```

```
{'text': ['Pushing', 'Daisies', 'truly', 'is', 'a', 'landmark', 'in', 'Te
pos
```

1. RNN 데이터 입력을 위한 데이터 로더 구현

3) vocabulary 생성

(1) text field와 label field vocabulary

- build_vocab( ) 함수를 이용하여 text field와 label field vocabulary 생성

```
text_field.build_vocab(train, vectors='fasttext.simple.300d')
label_field.build_vocab(train)
```

```
.vector_cache/wiki.simple.vec: 293MB [00:27, 10.7MB/s]
  0%|          | 0/111051 [00:00<?, ?it/s]WARNING:torchtext.vocab:Skipping
100%|██████████| 111051/111051 [00:14<00:00, 7659.41it/s]
```

- pre-trained 임베딩 모델 사용

```
vectors='fasttext.simple.300d'
```

1. RNN 데이터 입력을 위한 데이터 로더 구현

3) vocabulary 생성

(1) text field와 label field vocabulary

- 지원되는 pre-trained 임베딩 모델

pre-trained 임베딩 모델
charngram.100d
fasttext.en.300d
fasttext.simple.300d
glove.42B.300d
glove.840B.300d
glove.twitter.27B.25d
glove.twitter.27B.50d
glove.twitter.27B.100d
glove.twitter.27B.200d
glove.6B.50d
glove.6B.100d
glove.6B.200d
glove.6B.300d

→ torchtext.vocab.Vocab.set_vector 함수를 이용하여
사용자 정의 pre-trained 임베딩 모델 사용 가능

1. RNN 데이터 입력을 위한 데이터 로더 구현

4) 데이터 로더 구현

(1) BucketIterator 클래스를 이용한 데이터 로더 구현

- BucketIterator클래스의 splits( ) 함수를 이용하여 train 데이터 세트와 test 데이터 세트의 데이터 로더 구현

```
import torch
from torchtext.legacy.data import BucketIterator

device = 'cuda' if torch.cuda.is_available() else 'cpu'
batch_size = 32

train_iter, test_iter = BucketIterator.splits(
    (train, test),
    batch_size=batch_size,
    device=device
)
```

2. LSTM/GRU 모델 활용

1) PyTorch Lightning 개요

(1) PyTorch Lightning 정의

- PyTorch Lightning

PyTorch에 대한 High Level 인터페이스를 제공하는 오픈소스 Python 라이브러리

- PyTorch와 PyTorch Lightning의 관계
  - TensorFlow와 Keras의 관계와 유사

- PyTorch Lightning

PyTorch Deep Framework의 Training, Testing을 위해 복잡한 코드를 간결한 코드만으로도 사용할 수 있도록 구현

Bulid your neural network with PyTorch

Train it with PyTorch Lightning

2. LSTM/GRU 모델 활용

1) PyTorch Lightning 개요

(1) PyTorch Lightning 정의

- PyTorch Lightning

PyTorch

```
num_epochs = 1
for epoch in
range(num_epochs):

  # TRAINING LOOP
  for train_batch in
mnist_train:
    x, y = train_batch

    logits =
pytorch_model(x)
    loss =
cross_entropy_loss(logit
s, y)
    print('train loss: ',
loss.item())

    loss.backward() … …
```

»

PyTorch Lightning

```
model =
LightningMNISTClassifi
er()
trainer = pl.Trainer()

trainer.fit(model)
```

1) PyTorch Lightning 개요

(2) PyTorch Lightning 설치

- PyTorch Lightning 및 torchtext 설치

```
!pip install pytorch-lightning
!pip install torchtext
```

```
Requirement already satisfied: pyparsing>=2.0.2 in /
Requirement already satisfied: absl-py>=0.4 in /usr/
```

2. LSTM/GRU 모델 활용

2) PyTorch Lightning을 이용한 LSTM 구현

(1) LightningModule 클래스 상속

- LightningModule 클래스를 상속하여 사용자 정의 클래스 구현

```
import pytorch_lightning as pl

class RNNModel(pl.LightningModule):
```

(2) init 함수 재정의(Override)

- __init__( ) 함수를 이용하여 생성자 구현 및 주요 멤버 변수 초기화 수행

```
def __init__(self, embedding, lstm_input_size=300, lstm_hidden_size
    super().__init__()
    self.embedding = embedding
    self.lstm = nn.LSTM(lstm_input_size, lstm_hidden_size)
    self.lin = nn.Linear(lstm_hidden_size, output_size)
    self.loss_function = nn.CrossEntropyLoss()

    self.train_accuracy = pl.metrics.Accuracy()
    self.val_accuracy = pl.metrics.Accuracy()
```

2. LSTM/GRU 모델 활용

2) PyTorch Lightning을 이용한 LSTM 구현

(3) forward 함수 재정의(Override)

- forward( ) 함수를 이용하여 뉴럴 네트워크 구현

```
def forward(self, X: torch.Tensor):
    x = self.embedding[X].to(self.device).permute(1, 0, 2)
    x, _ = self.lstm(x)
    x = F.elu(x.permute(1, 0, 2))
    x = self.lin(x)
    x = x.sum(dim=1)
    return x
```

2) PyTorch Lightning을 이용한 LSTM 구현

(4) training, validation, test별 함수 재정의(Override)

- 학습 단계별로 step 함수를 이용하여 학습 및 평가 수행

```
def training_step(self, batch, batch_idx):
    x, y = batch.text[0].T, batch.label
    y_hat = self(x)
    loss = self.loss_function(y_hat, y)
    train_acc = self.val_accuracy(y_hat, y)
    self.log('train_acc', train_acc, prog_bar=True)
    return dict(loss=loss)
    )
```

```
def validation_step(self, batch, batch_idx):
    x, y = batch.text[0].T, batch.label
    y_hat = self(x)
    loss = self.loss_function(y_hat, y)
    val_acc = self.val_accuracy(y_hat, y)
    self.log('val_acc', val_acc, prog_bar=True)
    return dict(validation_loss=loss)
```

2) PyTorch Lightning을 이용한 LSTM 구현

(4) training, validation, test별 함수 재정의(Override)

- 학습 단계별로 데이터 로더 함수를 이용하여 데이터 로딩

```
def train_dataloader(self):
    return train_iter

def val_dataloader(self):
    return test_iter
```

(5) 기타 옵션 함수 재정의(Override)

- optimizer 및 learning rate 설정 등 기타 옵션 함수를 이용하여 처리

```
def configure_optimizers(self):
    return Adam(self.parameters(), lr=0.01)
```

3) 모델 객체 생성 및 학습 수행

(1) 사용자 정의 모델 클래스 객체 생성

- 생성자 함수를 이용하여 모델 클래스 객체 생성

```
model = RNNModel(text_field.vocab.vectors)
```

(2) trainer 클래스를 이용한 학습 수행

- trainer 객체 생성 및 fit( ) 함수를 이용한 학습 수행

```
trainer = pl.Trainer(
    gpus=1,
    max_epochs=3
)
trainer.fit(model)
```

```
INFO:pytorch_lightning.utilities.distributed:GPU available: True, used: True
INFO:pytorch_lightning.utilities.distributed:TPU available: False, using: 0 TPU cores
INFO:pytorch_lightning.utilities.distributed:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.accelerators.gpu:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.core.lightning:
  | Name           | Type             | Params
-------------------------------------------------
0 | lstm           | LSTM             | 160 K
1 | lin            | Linear           | 303
2 | loss_function  | CrossEntropyLoss | 0
3 | train_accuracy | Accuracy         | 0
4 | val_accuracy   | Accuracy         | 0
-------------------------------------------------
161 K     Trainable params
0         Non-trainable params
161 K     Total params
0.644     Total estimated model params size (MB)
Validation sanity check: 0%
Epoch 2: 100%
```

3) 모델 객체 생성 및 학습 수행

(3) 학습 수행 결과 평가

- train accuracy, validation accuracy 측정지표를 이용한 평가

```
1564/1564 [00:26<00:00, 59.57it/s, loss=0.396, v_num=2, train_acc=
```