방학 5주차

TRAINING & EVALUATION

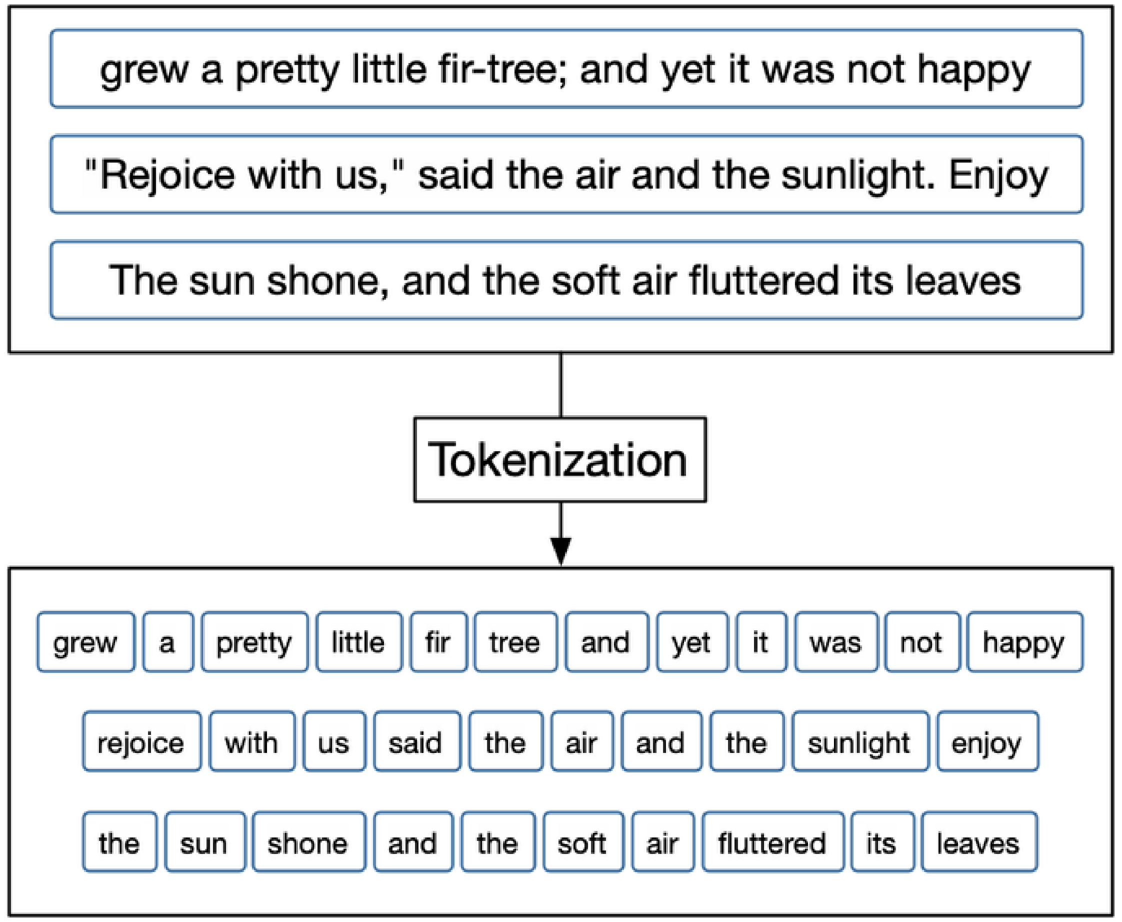
NEKA

Training의 과정

Tokenizer

주어진 코퍼스에서 토큰이라 불리는 단위로 나누는 작업

토큰은 단어나 형태소 단위로 구분하게 됨



Training의 과정

Tokenizer

```
from torchtext.data import get_tokenizer

tokenizer = get_tokenizer()
```

```
KoBertTokenizer(name_or_path='monologg/kobert', vocab_size=8002, model_max_length=512, is_fast=False, padding_side='right', truncation_side='right', special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]', 'mask_token': '[MASK]'}, clean_up_tokenization_spaces=True)
```

Training의 과정

Vocabulary

중복을 제거한 텍스트의 총 단어의 집합을 의미함

11205	prints
11206	bates
11207	reluctantly
11208	threshold
11209	algebra
11210	ira
11211	wherever
11212	coupled
11213	240
11214	assumption
11215	picks
11216	##air
11217	designers
11218	raids
11219	gentlemen
11220	##ean
11221	roller
11222	blowing
11223	leipzig
11224	locks

Training의 과정

Vocabulary

gluonnlp.vocab.Vocab(.from_sentencepiece)

```
Vocab(size=8002, unk="[UNK]", reserved="['[CLS]', '[SEP]', '[MASK]', '[PAD]']")
```

Training의 과정

Dataset 전처리 1 – 쓸모없는 데이터 제거하기

데이터를 딥러닝 모델에 넣을 수 있도록 만들어줌

id	중간고사	기말고사	나이	학점
1	87	63	20	A
2	56	65	20	C
3	25	73	21	B
4	46	57	22	B
5	64	86	21	C

id	중간고사	기말고사	학점
1	87	63	A
2	56	65	C
3	25	73	B
4	46	57	B
5	64	86	C

Training의 과정

Dataset 전처리 2 – 하나의 sequence로 만들기

데이터를 딥러닝 모델에 넣을 수 있도록 만들어줌

id	중간고사	기말고사	학점
1	87	63	A
2	56	65	C
3	25	73	B
4	46	57	B
5	64	86	C

→

sequence	학점
1 87 63	A
2 56 65	C
3 25 73	B
4 46 57	B
5 64 86	C

Training의 과정

Dataset 전처리 3 – SPECIAL TOKEN

데이터를 딥러닝 모델에 넣을 수 있도록 만들어줌

sequence	학점
1 87 63	A
2 56 65	C
3 25 73	B
4 46 57	B
5 64 86	C

sequence with special token	학점
[CLS] 1 [SEP] 87 [SEP] 63 [SEP]	A
[CLS] 2 [SEP] 56 [SEP] 65 [SEP]	C
[CLS] 3 [SEP] 25 [SEP] 73 [SEP]	B
[CLS] 4 [SEP] 46 [SEP] 57 [SEP]	B
[CLS] 5 [SEP] 64 [SEP] 86 [SEP]	C

Training의 과정

Dataset 전처리 3 – Label

데이터를 딥러닝 모델에 넣을 수 있도록 만들어줌

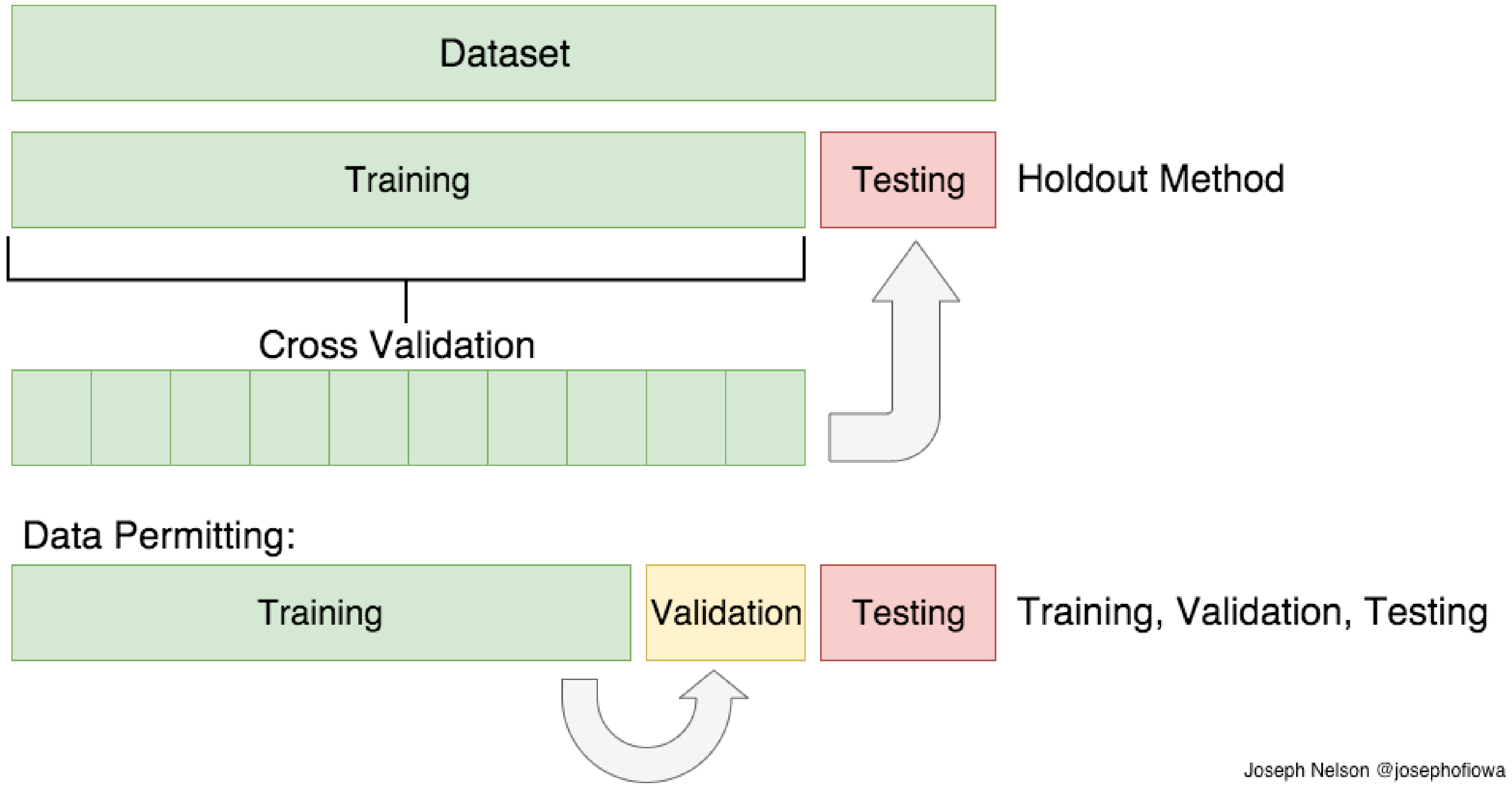
sequence with special token	학점
[CLS] 1 [SEP] 87 [SEP] 63 [SEP]	A
[CLS] 2 [SEP] 56 [SEP] 65 [SEP]	C
[CLS] 3 [SEP] 25 [SEP] 73 [SEP]	B
[CLS] 4 [SEP] 46 [SEP] 57 [SEP]	B
[CLS] 5 [SEP] 64 [SEP] 86 [SEP]	C

sequence with special token	학점
[CLS] 1 [SEP] 87 [SEP] 63 [SEP]	1
[CLS] 2 [SEP] 56 [SEP] 65 [SEP]	3
[CLS] 3 [SEP] 25 [SEP] 73 [SEP]	2
[CLS] 4 [SEP] 46 [SEP] 57 [SEP]	2
[CLS] 5 [SEP] 64 [SEP] 86 [SEP]	3

Training의 과정

데이터 분리하기

train : 학습용 데이터
validation : 검증용 데이터
test : 성능 평가용 데이터



Training의 과정

데이터 분리하기

sklearn.model_selection.train_test_split

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

Training의 과정

입력 데이터로 만들기

토큰화, 정수 인코딩하기, 패딩 작업하기 등

class `gluonnlp.data.` **`BERTSentenceTransform`**(*tokenizer*, *max_seq_length*, *vocab=None*, *pad=True*, *pair=True*) [source]

BERT style data transformation.

Parameters:
- **tokenizer** (*BERTTokenizer.*) – Tokenizer for the sentences.
- **max_seq_length** (*int.*) – Maximum sequence length of the sentences.
- **vocab** (*Vocab*) – The vocabulary which has cls_token and sep_token registered. If vocab.cls_token is not present, vocab.bos_token is used instead. If vocab.sep_token is not present, vocab.eos_token is used instead.
- **pad** (*bool, default True*) – Whether to pad the sentences to maximum length.
- **pair** (*bool, default True*) – Whether to transform sentences or sentence pairs.

Training의 과정

하라미터 파라미터 조정

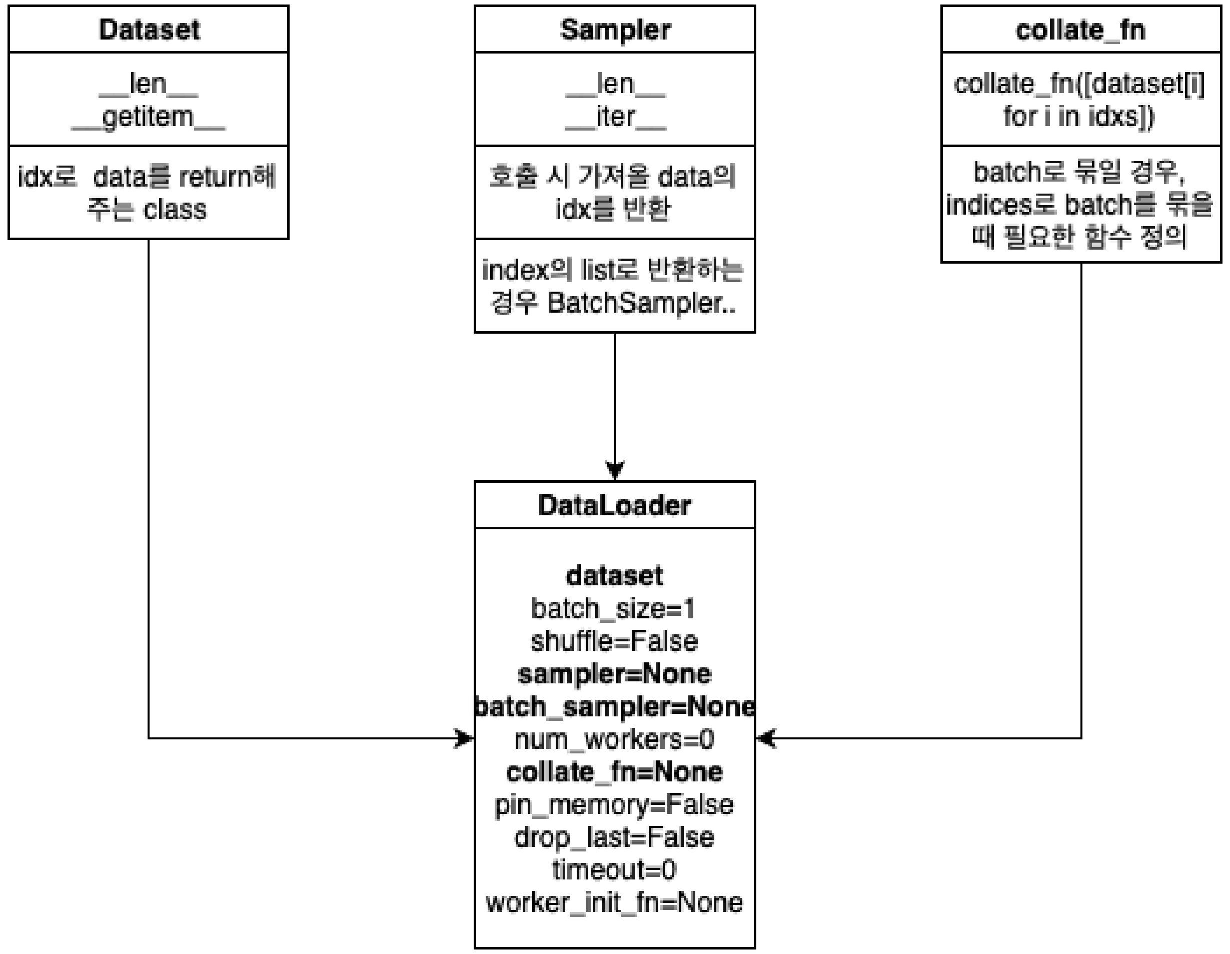
max_len : 한 sequence의 최대 길이
batch_size : 한번에 처리하는 sequence의 개수
num_epoch : 데이터셋 학습을 반복하는 횟수
log_interval : log를 띄워주는 간격
learning_rate : 학습률

```
# Setting parameters
max_len = 64
batch_size = 64
warmup_ratio = 0.1
num_epochs = 5
max_grad_norm = 1
log_interval = 200
learning_rate =  5e-5
```

Training의 과정

Dataloader

데이터셋을 미니배치 형태로 만들어주는 기능



Training의 과정

Classifier

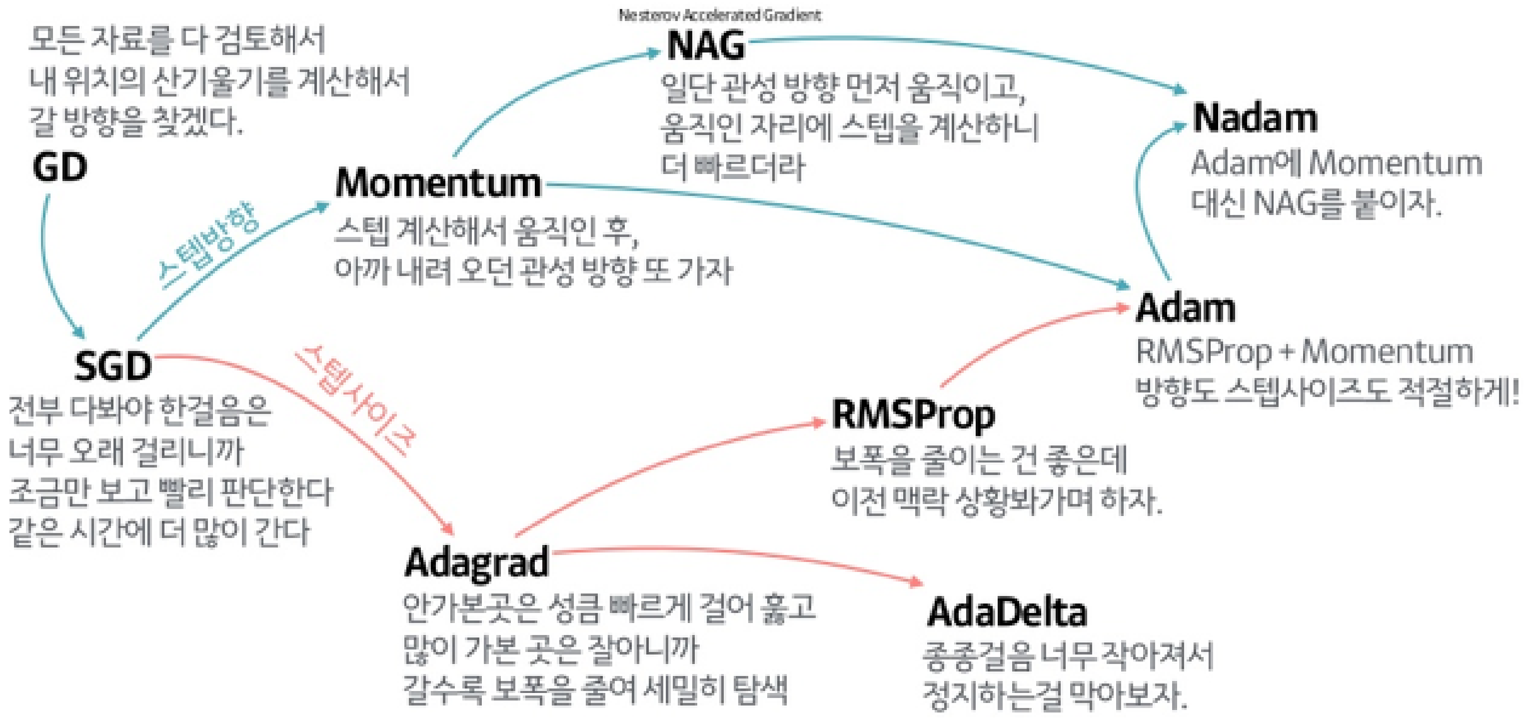
hidden_size : hidden state의 차원수
num_class : 분류되는 class의 개수
dr_rate : drop out 비율
attention mask : 패딩 토큰에 마스킹하기
forward : forweard propagation 진행

```python
class BERTClassifier(nn.Module):
    def __init__(self,
                 bert,
                 hidden_size = 768,
                 num_classes=60,   ##클래스 수 조정##
                 dr_rate=None,
                 params=None):
        super(BERTClassifier, self).__init__()
        self.bert = bert
        self.dr_rate = dr_rate

        self.classifier = nn.Linear(hidden_size , num_classes)
        if dr_rate:
            self.dropout = nn.Dropout(p=dr_rate)

    def gen_attention_mask(self, token_ids, valid_length):
        attention_mask = torch.zeros_like(token_ids)
        for i, v in enumerate(valid_length):
            attention_mask[i][:v] = 1
        return attention_mask.float()

    def forward(self, token_ids, valid_length, segment_ids):
        attention_mask = self.gen_attention_mask(token_ids, valid_length)

        _, pooler = self.bert(input_ids = token_ids, token_type_ids = segment_ids.long(), attention_mask =
        if self.dr_rate:
            out = self.dropout(pooler)
        return self.classifier(out)
```

Training의 과정

Optimizer

optimizer, loss function



출처 : https://www.slideshare.net/yongho/ss-79607172

Task	Error type	Loss function	Note
Regression	Mean-squared error	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Easy to learn but sensitive to outliers (MSE, L2 loss)
	Mean absolute error	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	Robust to outliers but not differentiable (MAE, L1 loss)
Classification	Cross entropy = Log loss	$-\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) +$ $(1 - y_i)\log(1 - \hat{y}_i)] =$	Quantify the difference between two probability

optimizer = AdamW(param, lr=)

loss_fn = nn.CrossEntropyLoss()

Training의 과정

training 진행

epoch과 accuracy, loss

```python
for e in range(num_epochs):
    train_acc = 0.0
    test_acc = 0.0
    model.train()
    for batch_id, (token_ids, valid_length, segment_ids, label) in enum
        optimizer.zero_grad()
        token_ids = token_ids.long().to(device)
        segment_ids = segment_ids.long().to(device)
        valid_length= valid_length
        label = label.long().to(device)
        out = model(token_ids, valid_length, segment_ids)

        #print(label.shape,out.shape)
        loss = loss_fn(out, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_nor
        optimizer.step()
        scheduler.step()  # Update learning rate schedule
        train_acc += calc_accuracy(out, label)
        if batch_id % log_interval == 0:
            print("epoch {} batch id {} loss {} train acc {}".format(e+
            train_history.append(train_acc / (batch_id+1))
            loss_history.append(loss.data.cpu().numpy())
    print("epoch {} train acc {}".format(e+1, train_acc / (batch_id+1))
    #train_history.append(train_acc / (batch_id+1))

    model.eval()
    for batch_id, (token_ids, valid_length, segment_ids, label) in enum
        token_ids = token_ids.long().to(device)
        segment_ids = segment_ids.long().to(device)
        valid_length= valid_length
        label = label.long().to(device)
        out = model(token_ids, valid_length, segment_ids)
        test_acc += calc_accuracy(out, label)
    print("epoch {} test acc {}".format(e+1, test_acc / (batch_id+1)))
    test_history.append(test_acc / (batch_id+1))
```

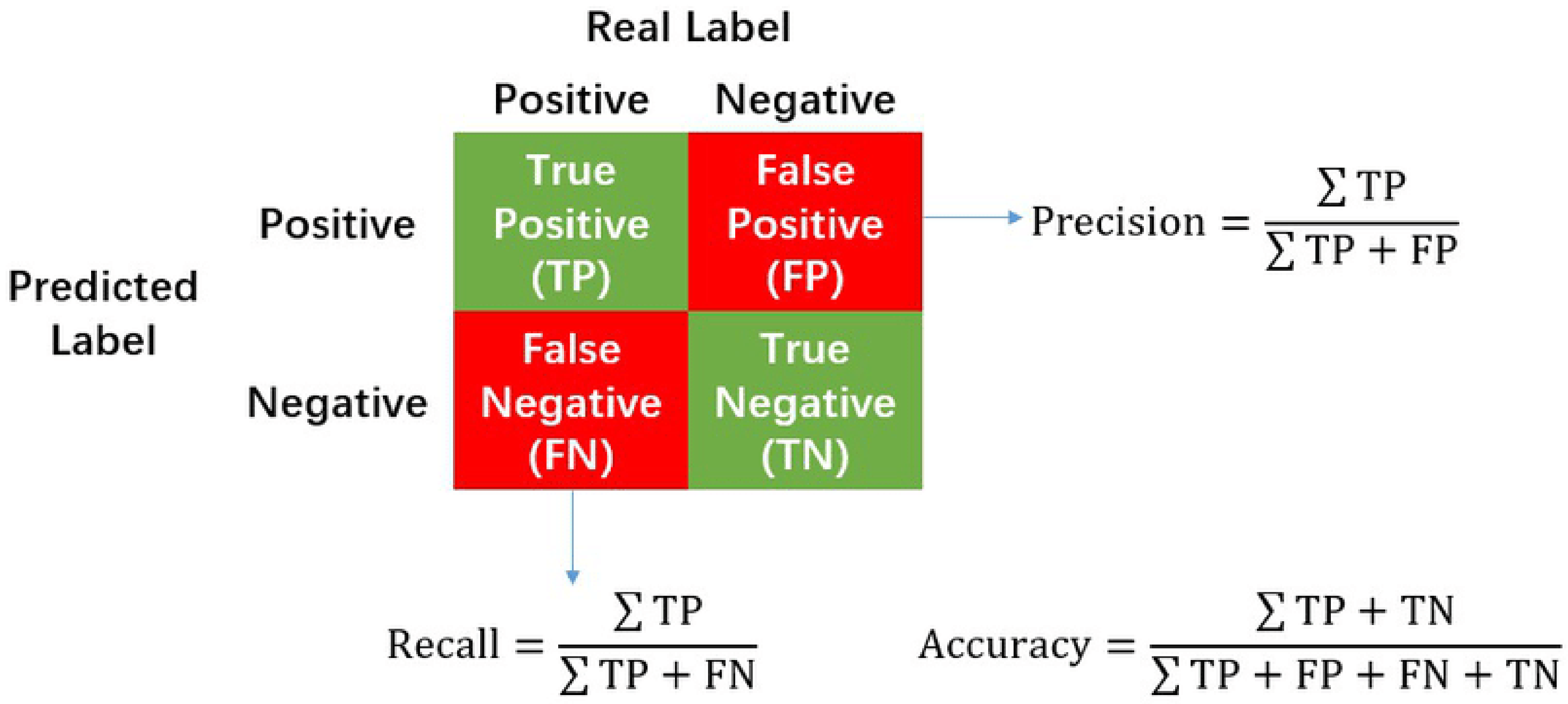
epoch : 반복하는 횟수

accuracy와 loss는 evaluation으로 사용

Evaluation

Metric 1

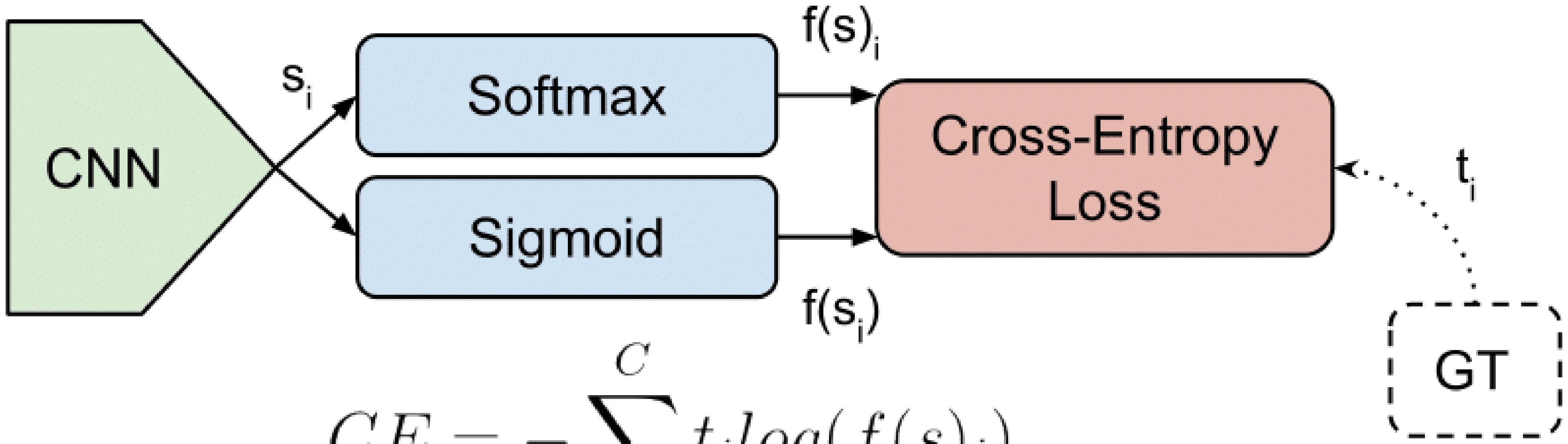
accuracy



Evaluation

Metric 2

Cross_entropy



$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

Evaluation

Metric 3

MSE

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

과제

오늘 배운 개념을 바탕으로
간단한 딥러닝 모델 구축하기

NEKA

THANK YOU