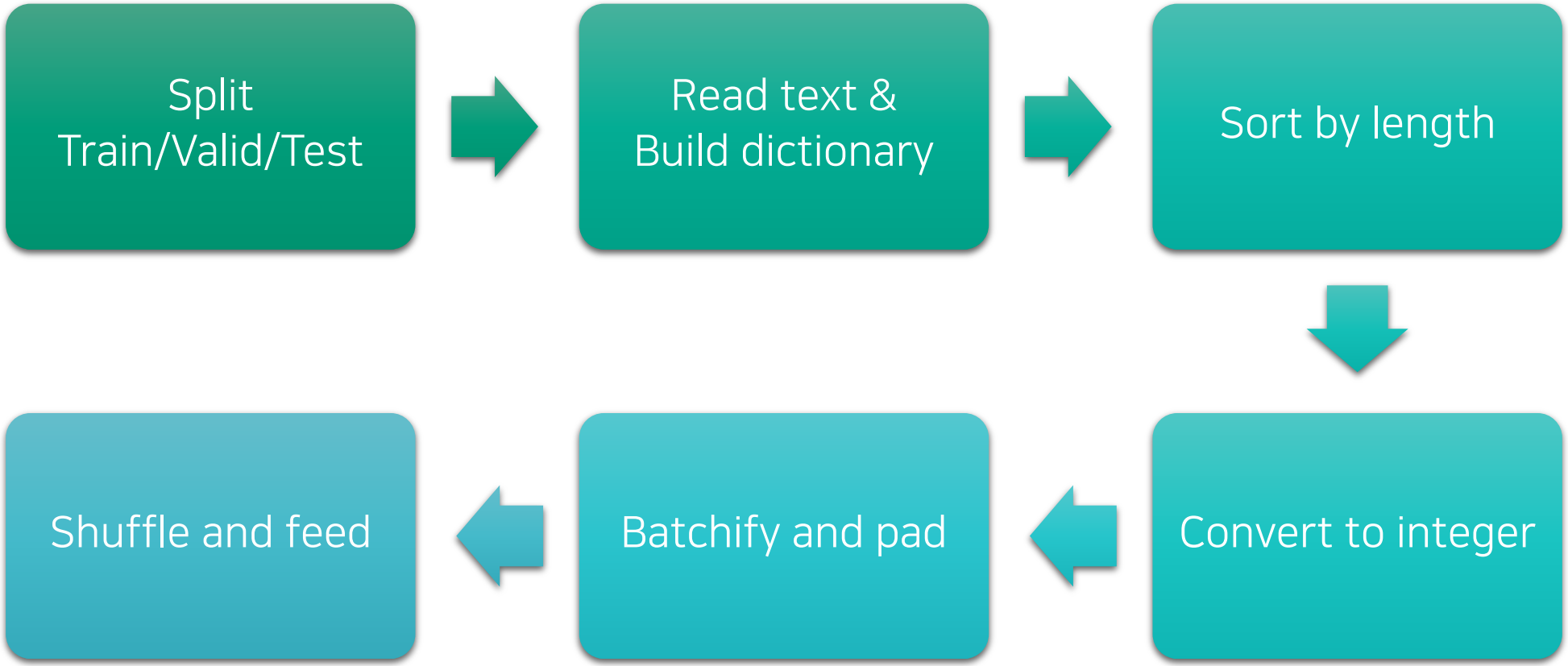
# TorchText: How to make a mini-batch

Ki Hyun Kim

nlp.with.deep.learning@gmail.com

# 모델에 넣기 위한 마지막 변환 과정

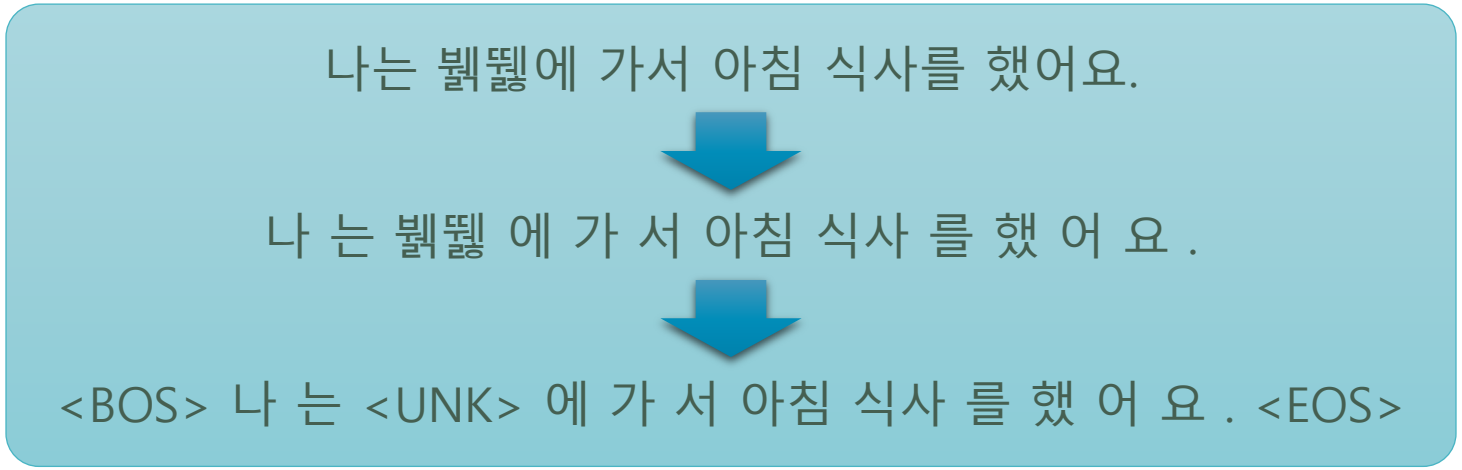


# Read Text & Build Dictionary

- 빈도 순으로 단어 사전 정렬

- 필요에 따라 min_count 보다 작은 빈도를 갖는 단어는 제외
  - 또는 max_vocab 에 따라 빈도순으로 어휘를 제외하기도 함

- 필요에 따라 특수 토큰도 어휘 사전에 포함
  - <BOS>, <EOS>, <UNK>, <PAD> 등



# Chunking and Padding

- 미니배치의 형태

$$(\text{batch\_size}, \text{length}, |V|)$$

- One-hot 벡터를 다 저장할 필요가 없음

$$(\text{batch\_size}, \text{length}, 1)$$
$$= (\text{batch\_size}, \text{length})$$

- Sequence 차원의 크기는 미니배치 내의 가장 긴 문장에 의해 결정됨
  - 각 샘플별 모자라는 부분은 padding으로 대체, 따라서 <PAD> 토큰이 필요.
  - PyTorch의 PackedSequence를 활용할 경우, <PAD> 생략 가능

- 문제점?

# Increase Training Efficiency

1) Sort by sequence length
2) Get chunks with similar length of sequences.

# 효율적인 학습이 가능한 미니배치 만들기

1) 코퍼스의 각 문장들을 길이에 따라 정렬

2) 각 token들을 사전을 활용하여 str→index 맵핑

3) 미니배치 크기대로 chunking

4) 각 미니배치 별 텐서 구성 및 padding

5) 학습 시 미니배치 shuffling하여 iterative하게 반환

# TorchText

- 앞서 소개한 작업들을 수행해주는 PyTorch 공식 텍스트 로딩용 라이브러리
  - https://github.com/pytorch/text

- 현재 버전 0.5.1

# Define Task: What you want

- **$f$(text) = class**
  - input: Text
  - output: Class (Numeric)
  - e.g. Text Classification

$$|\text{text}| = (\text{batch\_size}, \text{length}, |V|)$$

$$|\text{class}| = (\text{batch\_size}, \text{length}, |C|)$$

- **$f$(text) = word**
  - input : Text (words)
  - output : Word
  - e.g. Language Modeling

- **$f$(text) = text**
  - input : Text
  - output : Text
  - e.g. Machine Translation

# Step 1: Define Fields

```
# Define field of the input file.
# The input file consists of two fields.
self.label = data.Field(sequential=False,
                        use_vocab=True,
                        unk_token=None
                        )
self.text = data.Field(use_vocab=True,
                       batch_first=True,
                       include_lengths=False,
                       eos_token='<EOS>' if use_eos else None
                       )
```

# Step 2: Define Dataset with Fields

```
# Those defined two columns will be delimited by TAB.
# Thus, we use TabularDataset to load two columns in the input file.
# We would have two separate input file: train_fn, valid_fn
# Files consist of two columns: label field and text field.
train, valid = data.TabularDataset.splits(path='',
                                          train=train_fn,
                                          validation=valid_fn,
                                          format='tsv',
                                          fields=[('label', self.label),
                                                  ('text', self.text)
                                                  ]
                                          )
```

# Step 3: Get DataLoaders from Datasets

```
# Those loaded dataset would be feeded into each iterator:
# train iterator and valid iterator.
# We sort input sentences by length, to group similar lengths.
self.train_iter, self.valid_iter = data.BucketIterator.splits((train, valid),
                                                              batch_size=batch_size,
                                                              device='cuda:%d' % device if device >= 0 else 'cpu',
                                                              shuffle=shuffle,
                                                              sort_key=lambda x: len(x.text),
                                                              sort_within_batch=True
                                                              )

# At last, we make a vocabulary for label and text field.
# It is making mapping table between words and indice.
self.label.build_vocab(train)
self.text.build_vocab(train, max_size=max_vocab, min_freq=min_freq)
```

# Example Usage:

- Vocabulary (or Class) Size

```
vocab_size = len(dataset.text.vocab)
n_classes = len(dataset.label.vocab)
print('|vocab| =', vocab_size, '|classes| =', n_classes)
```

- Feeding with Ignite

```
trainer.run(train_loader, max_epochs=self.config.n_epochs)
```