# Data

## 字段Field

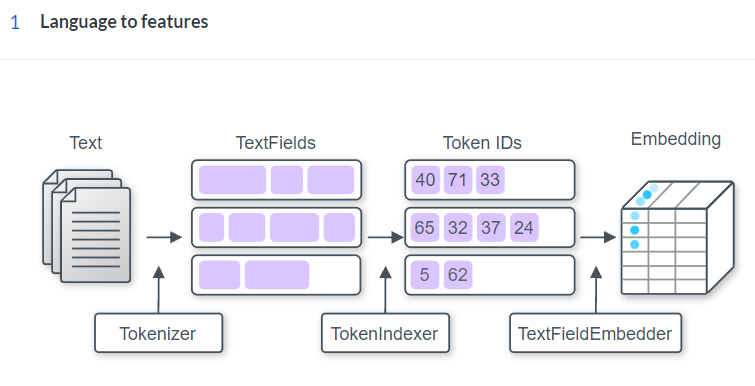
常用：

TextField

LabelField

SequenceLabelField

如何得到TextFiel并数值化



## 标记

Tokenizer (Text → Tokens)【标记器】

将字符串拆分为标记

常用：

WhitespaceTokenizer(): 把文本转换为单词的组合

CharacterTokenizer(): 把文本转换为字符的组合

SpacyTokenizer: 转换为单词同时获得额外属性

tokenize()方法返回a list of Tokens

## 标记索引

TextField, TokenIndexer, and Vocabulary (Tokens → Ids)

text\_field = TextField(tokens, token\_indexers)

TextField: 把 a list of Tokens 表示为an array

利用Vocabulary索引: text\_field.index(vocab)

然后提取padding信息: padding\_lengths = text\_field.get\_padding\_lengths()

最后得到填充后的TextFieldTensors:

tensor\_dict = text\_field.as\_tensor(padding\_lengths)

token\_indexers: Dict[str, Token\_Indexer]

TokenIndexer: 决定如何表示each Token【标记索引器】

0: padding

1: Vocabulary未知的标记

常用：

SingleIdTokenIndexer(): 把每个标记表示为一个ID（根据命名空间）

TokenCharactersIndexer(): 把每个标记表示成一串字符的ID

## 嵌入(Embedder)

TextFieldEmbedder (Ids → Vectors)

分别编码token\_indexers的输出，然后组合所有输出，使得每个标记对应一个向量

TextFieldTensors:

filed\_name[token\_indexer\_name][tokens]←→Dict[str, Dict[str, torch.Tensor]]

外部字典中的每个条目对应一个标记索引器

内部字典为索引tensor

常用: embedder = BasicTextFieldEmbedder(token\_embedders)

token\_embedders = Dict[str, Embddding]

embedded\_tokens = embedder(token\_tensor)

### TokenEmbedder

Takes as input a tensor with integer ids that have been output from a TokenIndexer and outputs a vector per token in the input.

Input: (batch\_size, num\_tokens) or (batch\_size, num\_tokens, num\_characters)

Output: (batch\_size, num\_tokens, output\_dim)

重要方法: get\_output\_dim

Returns the final output dimension that this TokenEmbedder uses to represent each token. This is not the shape of the returned tensor, but the last element of that shape.

### Embedding(嵌入)

**Registered as a TokenEmbedder with name "embedding".**

@TokenEmbedder.register("embedding")

class Embedding(TokenEmbedder):

Parameters

num\_embeddings: int

Size of the dictionary of embeddings (vocabulary size).

embedding\_dim: int

The size of each embedding vector. (get\_output\_dim()方法的返回值)

Return

An Embedding module.

### TokenCharactersEncoder

Takes the output of a TokenCharactersIndexer, which is a tensor of shape **(batch\_size, num\_tokens, num\_characters)**, embeds the characters, runs a token-level encoder, and returns the result, which is a tensor of shape **(batch\_size, num\_tokens, encoding\_dim)**.

@TokenEmbedder.register("character\_encoding")

class TokenCharactersEncoder(TokenEmbedder):

**Registered as a TokenEmbedder with name "character\_encoding"(注册为名为“character\_encoding”的标记嵌入程序)**

Parameters:

embedding: Embedding

encoder: Seq2vecEncoder

### TextFieldEmbedder

A Module that takes as input the DataArray produced by a TextField and returns as output an embedded representation of the tokens in that field.

### BasicTextFieldEmbedder

@TextFieldEmbedder.register("basic")

class BasicTextFieldEmbedder(TextFieldEmbedder):

def \_\_init\_\_(self, token\_embedders: Dict[str, TokenEmbedder]) -> None

This is a TextFieldEmbedder that wraps a collection of TokenEmbedder objects. Each TokenEmbedders embeds its input, and the result is concatenated in an arbitrary (but consistent) order.

**token\_embedders : Dict[str, TokenEmbedder]**

A dictionary mapping token embedder names to implementations. These names should match the corresponding indexer used to generate the tensor passed to the TokenEmbedder.

**Registered as a TextFieldEmbedder with name "basic", which is also the default.**

**Input:**

**text\_field\_input: TextFieldTensors**

**Output: torch.Tensor**

## 三部分组合

**1. Using a word-level tokenizer** (such as SpacyTokenizer or WhitespaceTokenizer):

I. SingleIdTokenIndexer → Embedding (for things like GloVe or other simple embeddings, including learned POS tag embeddings)

II. TokenCharactersIndexer → TokenCharactersEncoder (for things like a character CNN)

III. ElmoTokenIndexer → ElmoTokenEmbedder (for ELMo)

IV. PretrainedTransformerMismatchedIndexer

→ PretrainedTransformerMismatchedEmbedder(for using a transformer like BERT when you really want to do modeling at the word level, e.g., for a tagging task; more on what this does below)

**2. Using a character-level tokenizer** (such as CharacterTokenizer):

I. SingleIdTokenIndexer → Embedding

**3. Using a wordpiece tokenizer** (such as PretrainedTransformerTokenizer):

I. PretrainedTransformerIndexer → PretrainedTransformerEmbedder

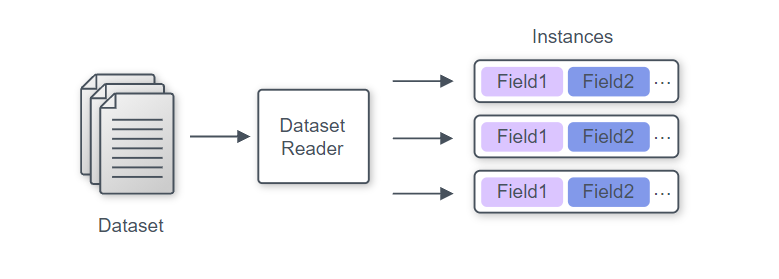
II. SingleIdTokenIndexer → Embedding (if you don't want contextualized wordpieces for some reason)

Contextualized representations in TextFields, like ELMo and BERT

Pretrained embeddings, like GloVe

# 实例Instance

字段→实例→数据集



实例为字典，其中key-value对应字段名-字段值

创建一个实例

instance = Instance(fields)

fields: Dict[str, Field] = {“f\_n-1”, field1, “f\_n-2”, field2}

Dataset readers 数据读取器

读取数据，得到实例迭代器（an iterable of Instances）

三步自定义

1. 继承DatasetReader并取名

2. 初始化\_\_init\_\_

3. 重写

text\_to\_instance()方法: fields→instance

\_read()方法: instance→instances

# Vocabulary

把为标记转化为整数ID提供映射，用**命名空间**（标记集合名）来区分，分为两类：Padded和Non-padded

按标记出现的频率索引

namespace一般取名: tokens和labels

创建方法: Vocabulary.from\_instances(instances)

from\_files ← “from\_files”

from\_files\_and\_instances ← “extend”

常用方法：

看标记的索引: get\_token\_index()

看索引所对应的标记: get\_token\_from\_index()

看索引列表get\_index\_to\_token\_vocabulary()

# Dataset Loader

→ An iterable of batched tensor dictionaries

把Instance打包后转换成数值（本质是把instances的Tokens换成IDs）

常用数据加载器:

1. MutiProcesssDataloader

建立方法: data\_loader = MultiProcessDataLoader(reader: DatasetReader, path\_to\_data: str, batch\_size: int)

封装: BatchSampler

data\_loader = MultiProcessDataLoader(reader: DatasetReader, path\_to\_data: str, batch\_sampler=BucketBatchSampler())

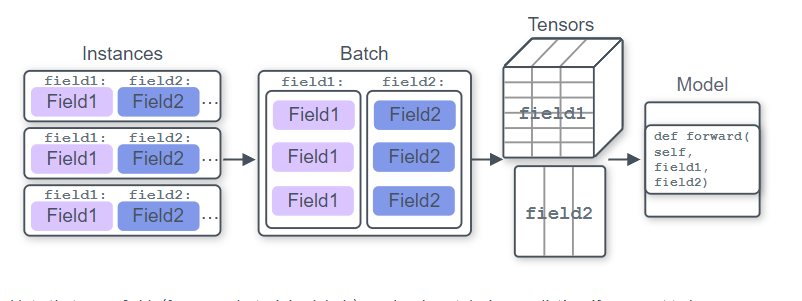
常用采样器: BucketBatchSample

2. SimpleDataLoader

利用Vocabulary数值化: data\_loader.index\_with(vocab: Vocabulary)

# Model

## 模型定义



1. 重写forward()方法

2. 定义get\_metrics()方法

常用方法：

make\_output\_human\_readable(): 结果可读↓

forward\_on\_instance()←Predictors

Metrics的get\_metric()方法: 自定义输出

常用: CategoricalAccuracy()

TextFieldTensors:

filed\_name[tokennizer\_name][tokens]←→Dict[str, Dict[str, torch.Tensor]]

## Saving Model

模型三要素：

1. Model config (specifications used to train the model)

Params class to\_file()方法

2. Model weights (trained parameters of the model)

model.state\_dict()方法（←torch.save()）

3. Vocabulary

Vocabulary.save\_to\_files()

进行封装: archive\_model()方法→tar.gz（自动调用）

## Loading Model

Model.load()方法

load\_archive()方法

# Common Architectures

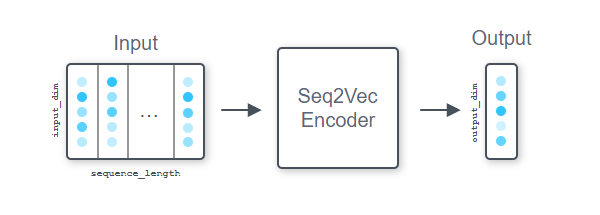
常用的neural architectures and AllenNLP abstractions

## Summarizing sequences

Take a sequence of vectors and summarize it to a single vector of fixed size.

Seq2VecEncoder:

(batch\_size, sequence\_length, input\_size) → (batch\_size, output\_size)



常用:

1. RNN: LstmSeq2VecEncoder and GruSeq2VecEncoder

2. CnnEncoder and BertPooler

### Seq2VecEncoder

A Seq2VecEncoder is a Module that takes as input a sequence of vectors and returns a single vector.

Input shape : (batch\_size, sequence\_length, input\_dim)

Output shape: (batch\_size, output\_dim).

**重要方法**

1. get\_input\_dim()

2. get\_output\_dim()

**CnnEncoder**

A CnnEncoder is a combination of multiple convolution layers and max pooling layers.

Input shape (batch\_size, num\_tokens, input\_dim)

Output shape (batch\_size, output\_dim)

@Seq2VecEncoder.register("cnn")

class CnnEncoder(Seq2VecEncoder):

Registered as a Seq2VecEncoder with name "cnn"

**Parameters:**

embedding\_dim : int = Input dimension to the encoder

num\_filters : int

the output dim for each convolutional layer = the number of "filters"

ngram\_filter\_sizes : Tuple[int], optional (default = (2, 3, 4, 5))

the number of convolutional layers and sizes.

(2, 3, 4, 5) corresponding to encoding ngrams of size 2 to 5 with some number of filters.

conv\_layer\_activation : Activation, optional (default = torch.nn.ReLU)

Activation to use after the convolution layers

output\_dim : Optional[int], optional (default = None)

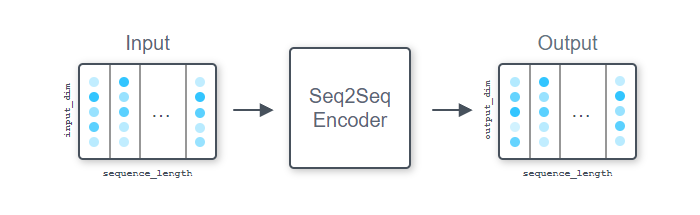
After doing convolutions and pooling, we'll project the collected features into a vector of this size. If this value is None, we will just return the result of the max pooling, giving an output of shape len(ngram\_filter\_sizes) \* num\_filters.

## Contextualizing sequences

Process a sequence of tokens and obtain another sequence of some embeddings.

Seq2SeqEncoder:

(batch\_size, sequence\_length, input\_dim) → (batch\_size, sequence\_length, output\_dim)



常用:

1. RNN: LstmSeq2SeqEncoder and GruSeq2SeqEncoder

2. FeedForwardEncoder

3. ComposeEncoder

4. PassThroughEncoder

## Similarities between sequences

Make use of attention to compute similarities (some sort of relatedness score) between sequences

AllenNLP provides two abstractions for attention—Attention and MatrixAttention.

### Attention

Attention modules compute similarities

**(each row of)** a matrix of size (batch\_size, sequence\_length, embedding\_dim)

a vector of size (batch\_size, embedding\_dim)

→a (typically normalized) similarity vector of size (batch\_size, sequence\_length)

常用:

1. DotProductAttention

Computes the dot product between the vector and each row of the matrix

2. BilinearAttention

Computes x^T W y + b for a given vector x and a matrix y using a matrix of weights W and a bias b ( The sizes of the embedding dimensions do not need to match)

3. LinearAttention

### MatrixAttention

MatrixAttention takes two matrices of size

(batch\_size, sequence\_length1, embedding\_dim)

(batch\_size, sequence\_length2, embedding\_dim)

→a (not normalized)matrix of size (batch\_size, sequence\_length1, sequence\_length2), which contains the similarity between each row of two input matrices

## Other common neural network building blocks

### FeedForward

### GatedSum and Highway layers

### TimeDistributed

### Conditional random field

### Activations