基于MindSpore的自然语言处理套件

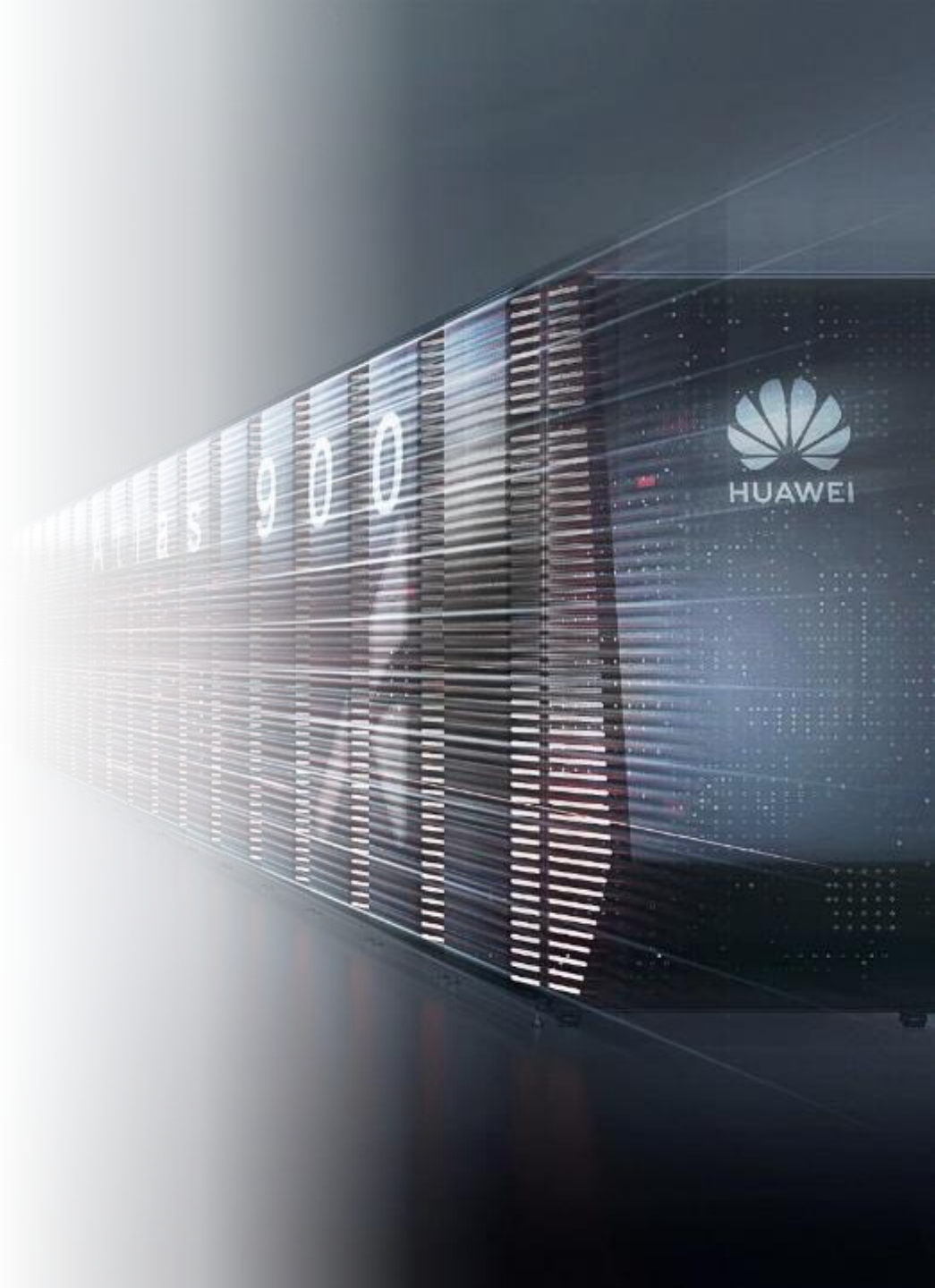
吕昱峰

MindSpore研发专家

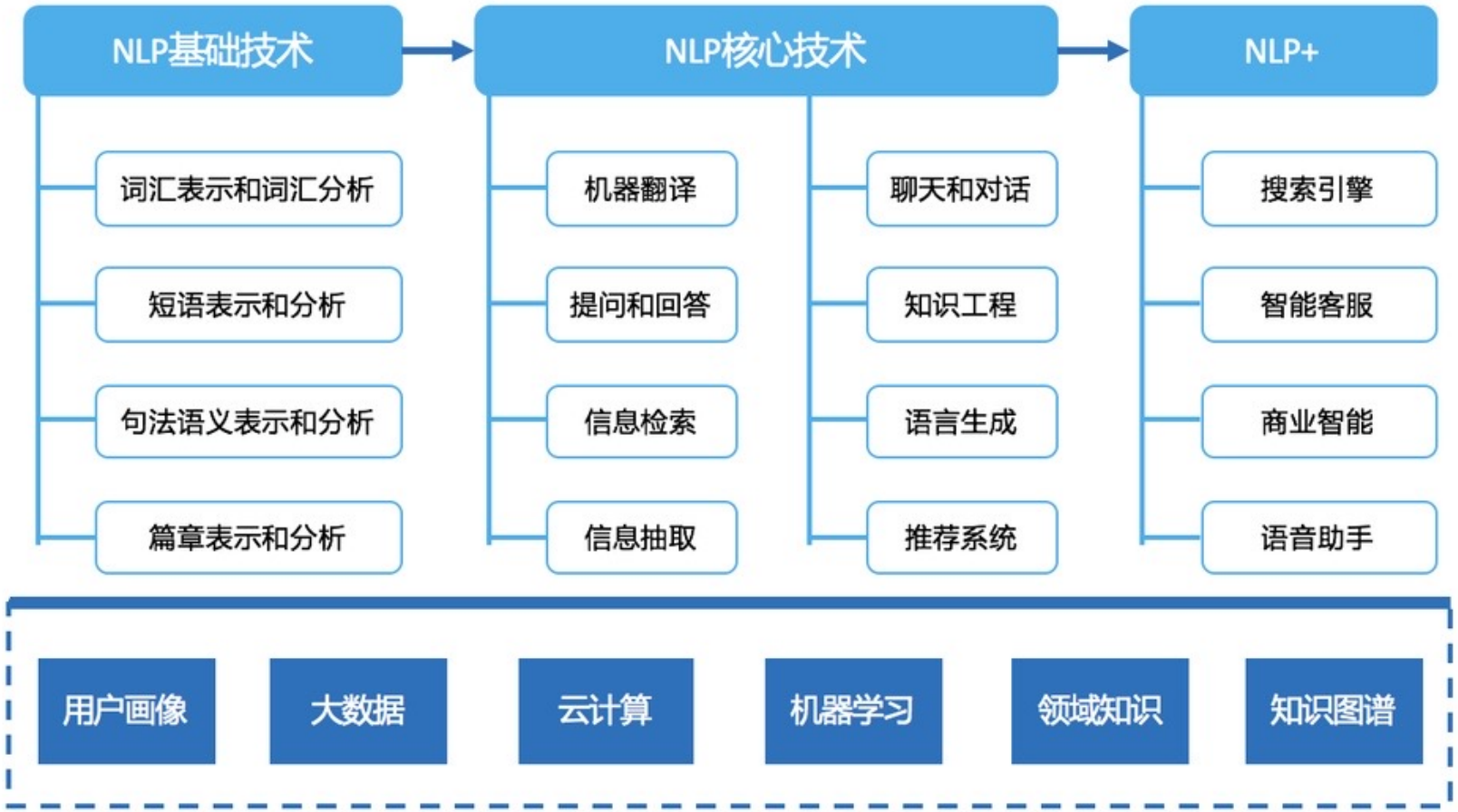
MindNLP套件负责人

HUAWEI

目录



多样化的自然语言处理任务



预训练语言模型逐步统一NLP范式

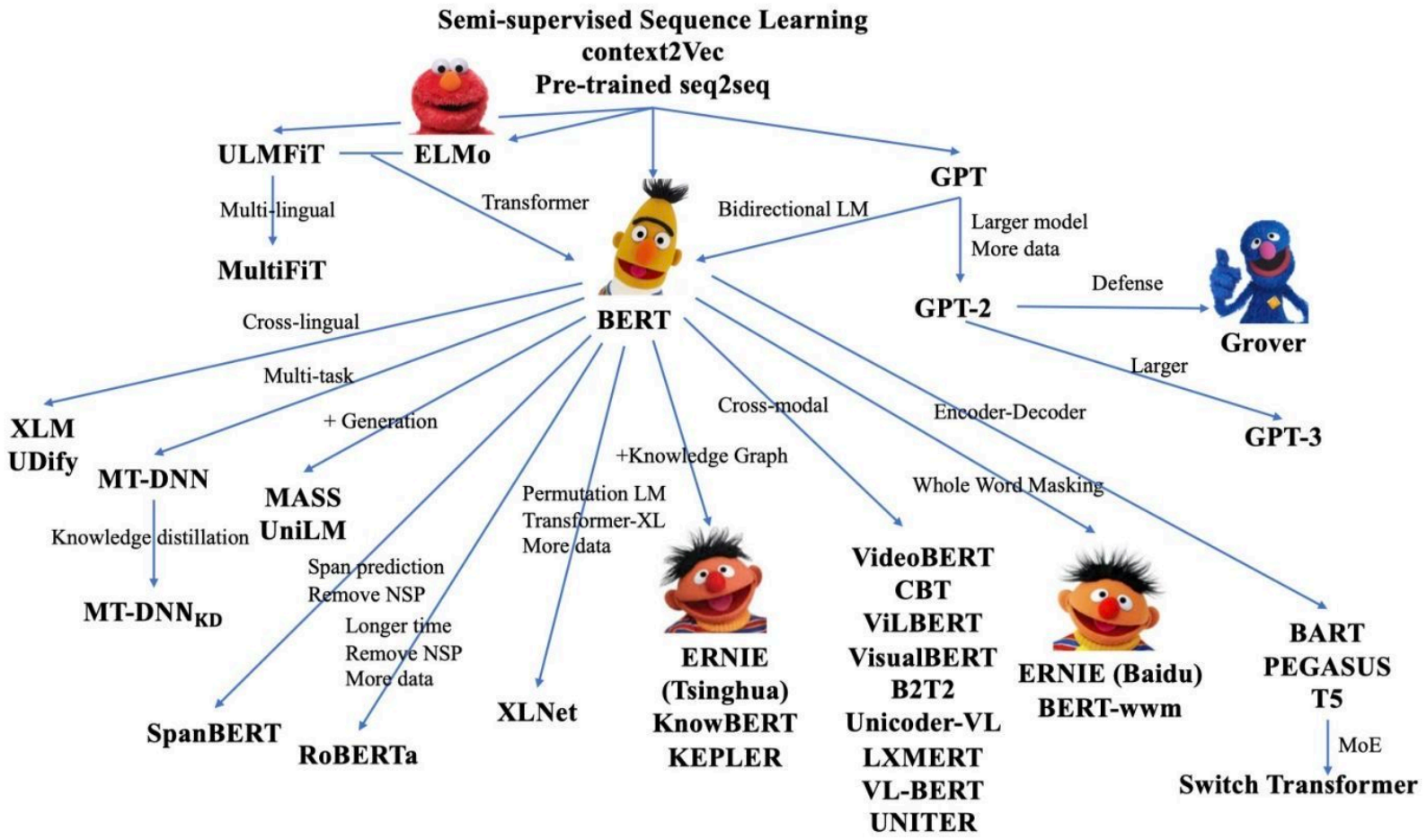
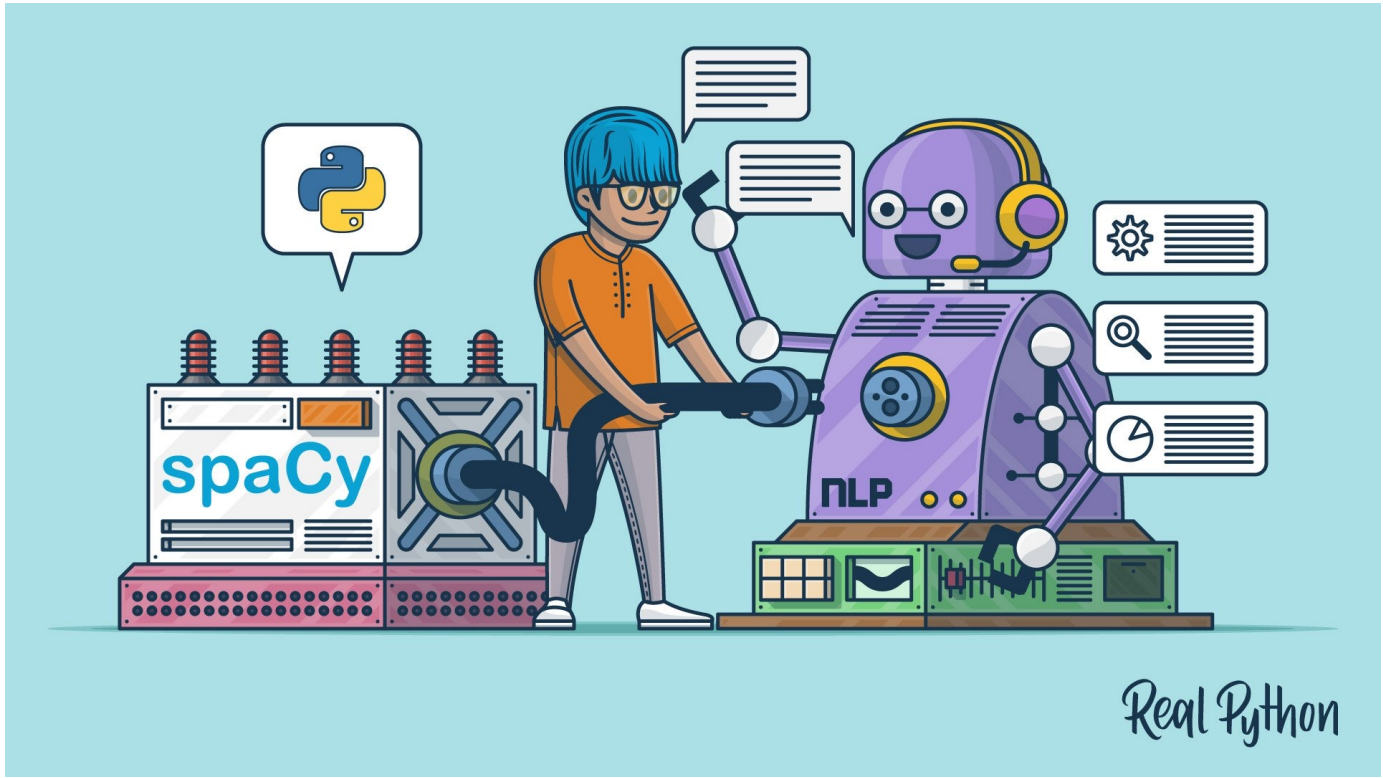


Figure 8: The family of recent typical PTMs, including both pre-trained language models and multimodal models.

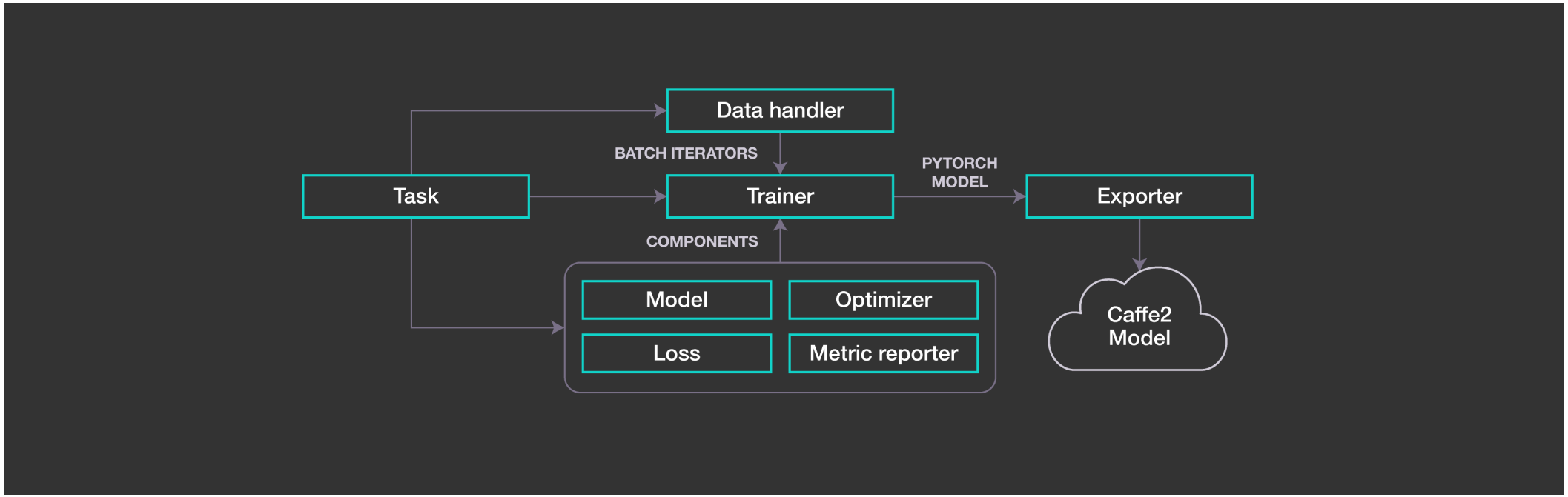
业界趋势



21 POPULAR NLP LIBRARIES OF 2022

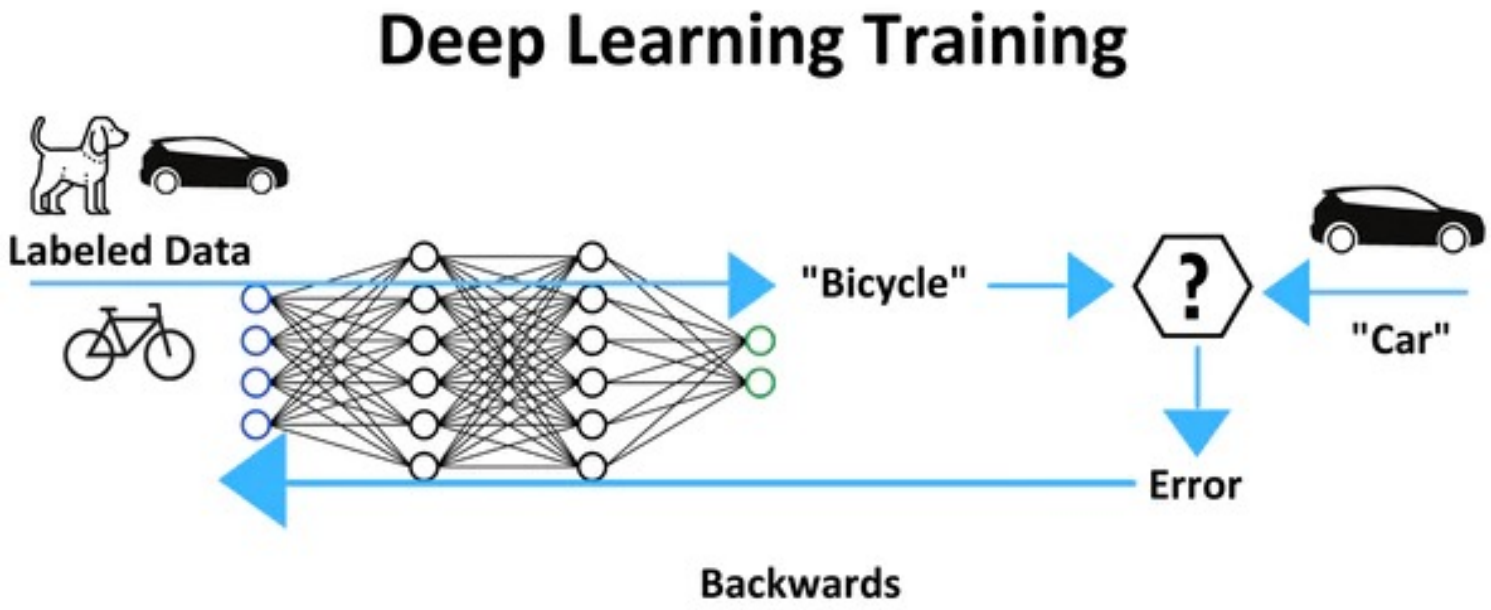
STARS	NAME	STARS	NAME
57.1K	HF Transformers	3.8K	Haystack
22.2K	spaCy	3.6K	Snips NLU
15.1K	Fairseq	2.8K	NLP Architect
12.8K	Gensim	2K	PyTorch-NLP
11.2K	Flair	1.9K	Polyglot
10.8K	AllenNLP	1.8K	TextAttack
10.4K	NLTK	513	WordForms
8.3K	CoreNLP	420	Rosetta
8.1K	Pattern		
8K	TextBlob	48.6K	scikit-learn
5.2K	HF Tokenizers	32.4K	pandas

自然语言处理系统化的必然选择



MindSpore的优势特性

- 函数式+面向对象融合编程
- 动静统一
- 数据处理引擎
- 自动并行



- 正向计算结果(logits)
- 计算结果(logits)与正确标签(targets)的误差
- 反向传播获取梯度
- 更新梯度到网络权重

OOP+FP融合编程

1. 用类构建神经网络
2. 实例化Network对象
3. Network+Loss直接构造正向函数
4. 函数变换，获得梯度计算（反向传播）函数
5. 构造训练过程函数
6. 调用函数进行训练

Pytorch vs MindSpore

```
net = Net()
loss_fn = nn.MSELoss()
optimizer = optim.Adam(net.parameters(), lr)

for i in range():
    for inputs, targets in dataset():
        optimizer.zero_grad()
        logits = net(inputs)
        loss = loss_fn(logits, targets)
        loss.backward()
        optimizer.step()
```

```
net = Net()
loss_fn = nn.MSELoss()
optimizer = nn.Adam(net.trainable_params(), lr)

def forword_fn(inputs, targets):
    logits = net(inputs)
    loss = loss_fn(logits, targets)
    return loss

grad_fn = value_and_grad(forward_fn, None, optim.parameters)

def train_step(inputs, targets):
    loss, grads = grad_fn(inputs, targets)
    optimizer(grads)
    return loss

for i in range():
    for inputs, targets in dataset():
        loss = train_step(inputs, targets)
```

即时编译(JIT) —— ms.jit

```
def train_step(inputs, targets):
    loss, grads = grad_fn(inputs, targets)
    optimizer(grads)
    return loss
```

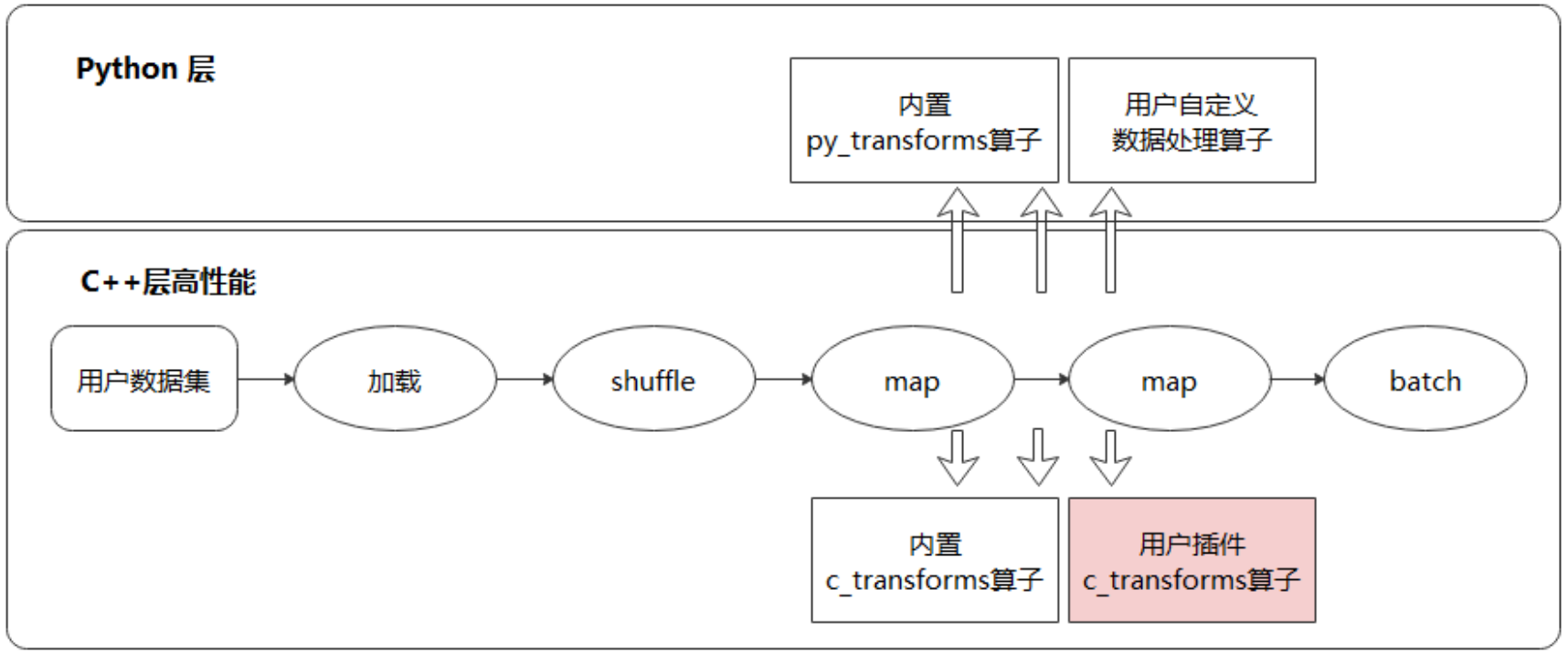
```
@ms_function
def train_step(inputs, targets):
    loss, grads = grad_fn(inputs, targets)
    optimizer(grads)
    return loss
```

ms.jit修饰器：

- 一行代码切换动静态图
- 即时编译，被修饰函数转为整图

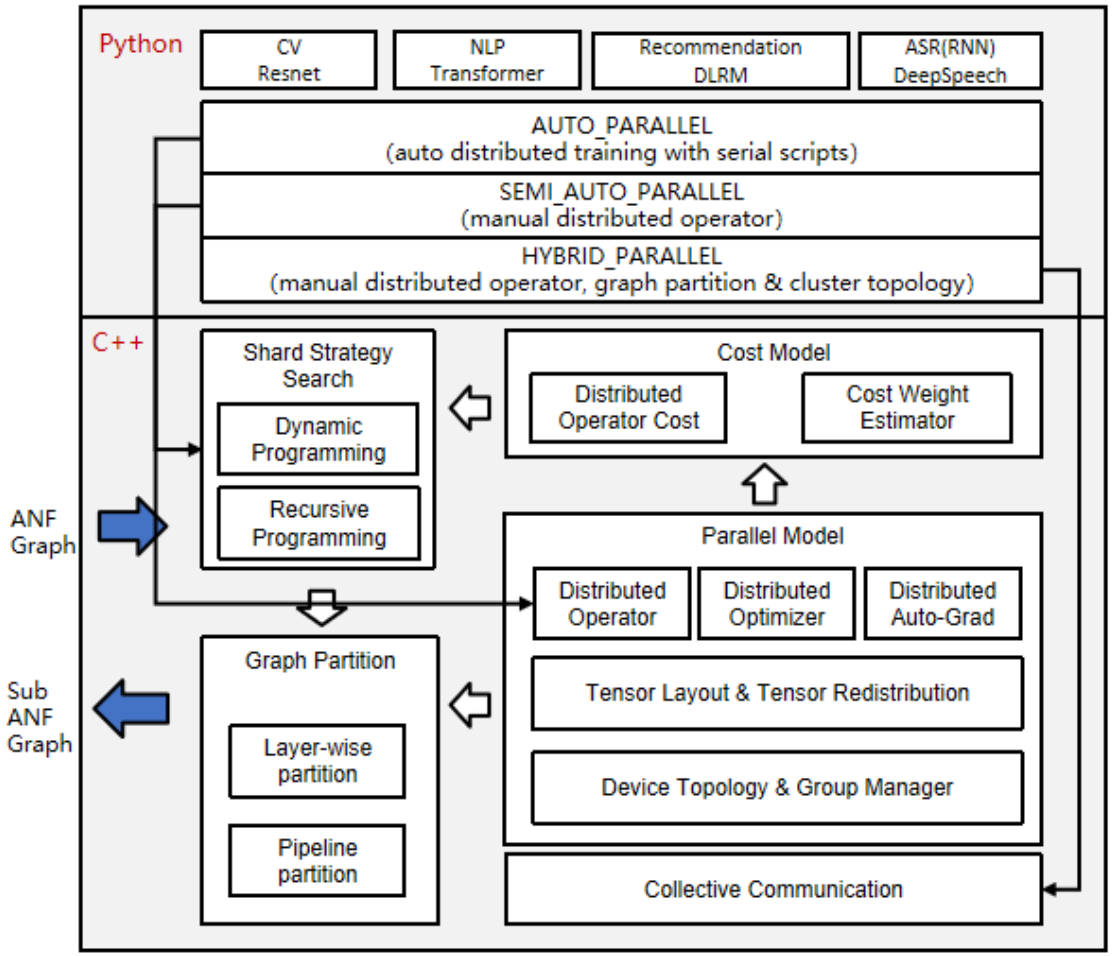
数据处理引擎

高效数据处理Pipeline，让数据在Pipeline内流动，实现高效处理能力



自动并行

- 自动并行流程会对单机的正向计算图（ANF Graph）进行遍历，以分布式算子（Distributed Operator）为单位对张量进行切分建模，表示一个算子的输入输出张量如何分布到集群各个卡上（Tensor Layout）。这种模型充分地表达了张量和设备间的映射关系，用户无需感知模型各切片放到哪个设备上运行，框架会自动调度分配。
- 为了得到张量的排布模型，每个算子都具有切分策略（Shard Strategy），它表示算子的各个输入在相应维度的切分情况。

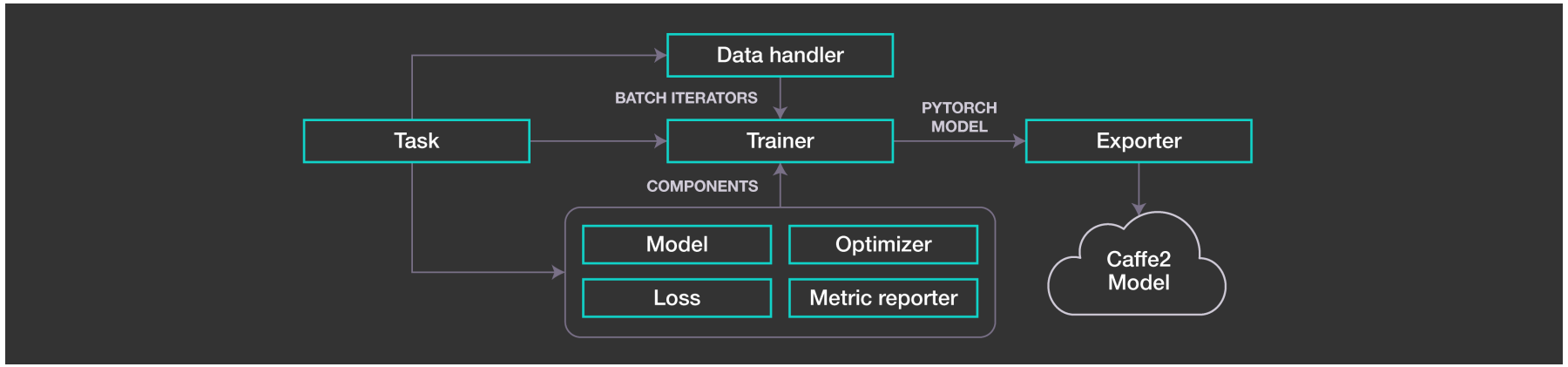


自然语言处理的需求

- 灵活编码
- 动态图+动态Shape支持
- 丰富的数据集
- 预训练语言模型支持

MindNLP特性一览

- 灵活编码 → 基于OOP+FP的Trainer
- 动态图+动态Shape支持 → 参数配置切换动静态图
- 丰富的数据集 → 5大类29个数据集
- 预训练语言模型支持 → 类Huggingface的PLM



基于OOP+FP的Trainer

- 灵活性强，分类问题、Seq2seq文本生成、序列标注等都可以使用同一套内置的训练逻辑控制，用户只需要关注模型构造
- 动静态图一键配置切换
- 与Dataset配合，灵活控制数据集中不同列数据的作用。

```
def forward_fn(inputs, labels):
    logits_list = ()
    logits = net(*inputs)
    if isinstance(logits, tuple):
        logits_list += logits
    else:
        logits_list += (logits,)

    loss = loss_fn(*logits_list, *labels)
    return_list = (loss,) + logits_list
    return return_list

def forward_without_loss_fn(inputs, labels):
    loss_and_logits = net(*inputs, *labels)
    return loss_and_logits

if self.loss_fn is None:
    grad_fn = value_and_grad(forward_without_loss_fn, None, optimizer.parameters, has_aux=True)
else:
    grad_fn = value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)

def _run_step(inputs, labels):
    """Core process of each step, including the forward propagation process and back propagation of data."""
    (loss, *_), grads = grad_fn(inputs, labels)
    optimizer(grads)
    return loss

@ms_jit
def _run_step_graph(inputs, labels):
    """Core process of each step, including the forward propagation process and back propagation of data."""
    (loss, _), grads = grad_fn(inputs, labels)
    loss = ops.depend(loss, optimizer(grads))
    return loss
```

```
def run(self, tgt_columns=None, jit=False):
    """
    Training process entry.

    Args:
        tgt_columns (Optional[list[str], str]): Target label column names for loss function.
        jit (bool): Whether use Just-In-Time compile.

    """

    args_dict = vars(self)
    run_context = RunContext(args_dict)
    self.callback_manager.train_begin(run_context)
    self._run(run_context, tgt_columns, jit)
    self.callback_manager.train_end(run_context)
```

基于OOP+FP的Trainer

```
from mindnlp.engine.metrics import Accuracy
from mindnlp.engine.trainer import Trainer

# define metrics
metric = Accuracy()

# define trainer
trainer = Trainer(network=net, train_dataset=imdb_train, eval_dataset=imdb_valid, metrics=metric,
                  epochs=5, loss_fn=loss, optimizer=optimizer)
trainer.run(tgt_columns="label", jit=False)
print("end train")
```

- 使用简单，将Network、Dataset、Loss、Optimizer传入即可。
- 配合Callback、Evaluator，方便用户进行边训练边推理

基于pipeline的数据预处理

```
from mindnlp.dataset import process

imdb_train = process('imdb', imdb_train, tokenizer=tokenizer, vocab=vocab, \
                     bucket_boundaries=[400, 500], max_len=600, drop_remainder=True)
imdb_train, imdb_valid = imdb_train.split([0.7, 0.3])
```

```
pad_value = vocab.tokens_to_ids('<pad>')

lookup_op = text.Lookup(vocab, unknown_token='<unk>')
type_cast_op = transforms.TypeCast(mindspore.float32)

dataset = dataset.map([tokenizer, lookup_op], 'text')
dataset = dataset.map([type_cast_op], 'label')
if bucket_boundaries is not None:
    if not isinstance(bucket_boundaries, list):
        raise ValueError(f"'bucket_boundaries' must be a list of int, but get {type(bucket_boundaries)}")
    trancate_op = TruncateSequence(max_len)
    dataset = dataset.map([trancate_op], 'text')
    if bucket_boundaries[-1] < max_len + 1:
        bucket_boundaries.append(max_len + 1)
    bucket_batch_sizes = [batch_size] * (len(bucket_boundaries) + 1)
    dataset = make_bucket(dataset, 'text', pad_value, \
                          bucket_boundaries, bucket_batch_sizes, drop_remainder)
else:
    pad_op = transforms.PadEnd([max_len], pad_value)
    dataset = dataset.map([pad_op], 'text')
    dataset = dataset.batch(batch_size, drop_remainder=drop_remainder)
```

基于MindSpore数据处理引擎进行二次封装，具备以下优势：

1. 内置Pipeline操作，用户只需简单调用。
2. 针对动静态图、NLP数据动态shape问题进行了多种处理逻辑的选择。
3. 与内置数据集load、Trainer协同，无需额外配置。

类Huggingface的预训练语言模型实现

```python
class BertModel(nn.Cell):
    r"""
    Bert Model
    """
    def __init__(self, config):
        super().__init__()
        self.embeddings = BertEmbeddings(config)
        self.encoder = BertEncoder(config)
        self.pooler = BertPooler(config)
        self.num_hidden_layers = config.num_hidden_layers

    def construct(self, input_ids, attention_mask=None, token_type_ids=None, \
        position_ids=None, head_mask=None):
        if attention_mask is None:
            attention_mask = ops.ones_like(input_ids)
        if token_type_ids is None:
            token_type_ids = ops.zeros_like(input_ids)

        extended_attention_mask = attention_mask.expand_dims(1).expand_dims(2)
        extended_attention_mask = (1.0 - extended_attention_mask) * -10000.0

        if head_mask is not None:
            if head_mask.ndim == 1:
                head_mask = head_mask.expand_dims(0).expand_dims(0).expand_dims(-1).expand_dims(-1)
                head_mask = mnp.broadcast_to(head_mask, (self.num_hidden_layers, -1, -1, -1, -1))
            elif head_mask.ndim == 2:
                head_mask = head_mask.expand_dims(1).expand_dims(-1).expand_dims(-1)
        else:
            head_mask = [None] * self.num_hidden_layers

        embedding_output = self.embeddings(input_ids, position_ids=position_ids, \
            token_type_ids=token_type_ids)
        encoder_outputs = self.encoder(embedding_output,
                                       extended_attention_mask,
                                       head_mask=head_mask)
        sequence_output = encoder_outputs[0]
        pooled_output = self.pooler(sequence_output)

        outputs = (sequence_output, pooled_output,) + encoder_outputs[1:]
        # add hidden_states and attentions if they are here
        return outputs
        # sequence_output, pooled_output, (hidden_states), (attentions)
```

```python
import mindspore
from mindnlp.models import BertModel
from mindnlp.dataset.transforms import BertTokenizer

tokenizer = BertTokenizer.load('bert-base-uncased')
model = BertModel.load('bert-base-uncased')
model.set_train(False)

# get tokenized inputs
text = tokenizer.encode("Here is some text to encode",
                        add_special_tokens=True)
inputs = mindspore.Tensor([text], mindspore.int32)
# run model inference
outputs = model(inputs)
```

常用的NLP功能模块支持

- Encoders
- Decoders
- Embeddings
- Attentions

Encoders

rnn_encoder

RNN encoder modules

class **mindnlp.modules.encoder.rnn_encoder.RNNEncoder**(*embedding*, *rnn*) [source]

Bases: `EncoderBase`

Seq2Seq Encoder.

Parameters:
- **embedding** (*Cell*) – The embedding layer.
- **rnn** (*Cell*) – The RNN Layer.

class **mindnlp.modules.encoder.cnn_encoder.CNNEncoder**(*embedding*, *convs*, *conv_layer_activation=Tanh<>*, *output_dim=None*) [source]

Bases: `EncoderBase`

CNN Encoder.

Convolutional encoder consisting of *len(convolutions)* layers.

Parameters:
- **embedding** (*Cell*) – The embedding layer.
- **convs** (*list[Cell]*) – The list of Conv Cell.
- **conv_layer_activation** (*Module*) – Activation to use after the convolution layers.
- **output_dim** (*int*) – The output vector of collected features after doing convolutions and pooling. If this value is *None*, return the result of the max pooling, an output of shape.

Decoders

RNN Decoder modules

class **mindnlp.modules.decoder.rnn_decoder.RNNDecoder**(*embedding*, *rnns*, *dropout_in=0*, *dropout_out=0*, *attention=True*, *encoder_output_units=512*, *mode='RNN'*) [source]

Bases: `DecoderBase`

Seq2Seq Decoder.

Parameters:

- **embedding** (*Cell*) – The embedding layer.
- **rnns** (*list*) – The list of RNN Layers.
- **dropout_in** (*Union[float, int]*) – If not 0, append *Dropout* layer on the inputs of each RNN layer. Default 0. The range of dropout is [0.0, 1.0).
- **dropout_out** (*Union[float, int]*) – If not 0, append *Dropout* layer on the outputs of each RNN layer except the last layer. Default 0. The range of dropout is [0.0, 1.0).
- **attention** (*bool*) – Whether to use attention. Default: True.
- **encoder_output_units** (*int*) – Number of features of encoder output. Default: 512.

Embeddings

```
class Fasttext(TokenEmbedding):
    r"""
    Create vocab and Embedding fr

    Args:
        vocab (Vocab): Passins ir
        init_embed (Tensor): Pass
        requires_grad (bool): Whe
        dropout (float): Dropout

    Examples:
        >>> vocab = Vocab.from_li
        >>> init_embed = Tensor(r
        >>> fasttext_embed = Fast
        >>> ids = Tensor([1, 2, 3
        >>> output = fasttext_emb

    """
    urls = {
        "1M": "https://dl.fbaipub
        "1M-subword": "https://dl
    }
```

```
class Word2vec(TokenEmbedding):
    r"""
    Create vocab and Embedding from a

    Args:
        vocab (Vocab): Passins into Vo
        init_embed (Tensor): Passing i
        requires_grad (bool): Whether
        dropout (float): Dropout of th

    Examples:
        >>> vocab = Vocab.from_list([
        >>> init_embed = Tensor(np.zer
        >>> word2vec_embed = Word2vec
        >>> ids = Tensor([1, 2, 3])
        >>> output = word2vec_embed(id

    """
    urls = {
        "google-news": "https://github
                       "word2vec-goog
    }
```

```
class Glove(TokenEmbedding):
    r"""
    Create vocab and Embedding from a given pre-trained vector file.

    Args:
        vocab (Vocab): Passins into Vocab for initialization.
        init_embed (Tensor): Passing into Tensor,use these values to initialize Embedding directly.
        requires_grad (bool): Whether this parameter needs to be gradient to update. Default: True.
        dropout (float): Dropout of the output of Embedding. Default: 0.5.

    Examples:
        >>> vocab = Vocab.from_list(['default','one','two','three'])
        >>> init_embed = Tensor(np.zeros((4, 4)).astype(np.float32))
        >>> glove_embed = Glove(vocab, init_embed)
        >>> ids = Tensor([1, 2, 3])
        >>> output = glove_embed(ids)

    """
    urls = {
        "42B": "http://nlp.stanford.edu/data/glove.42B.300d.zip",
        "840B": "http://nlp.stanford.edu/data/glove.840B.300d.zip",
        "twitter.27B": "http://nlp.stanford.edu/data/glove.twitter.27B.zip",
        "6B": "http://nlp.stanford.edu/data/glove.6B.zip",
    }
```

Attentions

class **mindnlp.modules.attentions.AdditiveAttention**(*hidden_dims, dropout=0.9*) [source]

Bases: `Cell`

Additive Attention Additive Attention proposed in “Neural Machine Translation by Jointly Learning to Align and Translate” paper

$$Attention(Q, K, V) = (W_v)T * (tanh(W_q * Q + W_k * K))$$

class **mindnlp.modules.attentions.BinaryAttention** [source]

Bases: `Cell`

Binary Attention, For a given sequence of two vectors : x_i and y_j, the BiAttention module will compute the attention result by the following equation:

$$e_{ij} = x_i^{\mathrm{T}} y_j$$
$$\hat{x}_i = \sum_{j=1}^{l_y} \frac{\exp(e_{ij})}{\sum_{k=1}^{l_y} \exp(e_{ik})} y_j$$
$$\hat{y}_j = \sum_{i=1}^{l_x} \frac{\exp(e_{ij})}{\sum_{k=1}^{l_x} \exp(e_{ik})} x_i$$

class **mindnlp.modules.attentions.CosineAttention**(*dropout=0.9*) [source]

Bases: `Cell`

Cosine Attention Cosine Attention proposed in “Neural Turing Machines” paper

$$Sim(Q, K) = (Q * (K)T)/|Q| * |K| Attention(Q, K, V) = softmax(Sim(Q, K)) * V$$

Parameters: **dropout** (*float*) – The keep rate, greater than 0 and less equal than 1. E.g. rate=0.9, dropping out 10% of input units. Default: 0.9.

class **mindnlp.modules.attentions.LinearAttention**(*query_dim, key_dim, hidden_dim, dropout=0.9*) [source]

Bases: `Cell`

Linear attention computes attention by concat the query and key vector.

class **mindnlp.modules.attentions.LocationAwareAttention**(*hidden_dim, smoothing=False*) [source]

Bases: `Cell`

Location Aware Attention Location Aware Attention proposed in “Attention-Based Models for Speech Recognition”

Parameters:
- **hidden_dim** (*int*) – The dimension of the hidden states
- **smoothing** (*bool*) – Smoothing label from “Attention-Based Models for

class **mindnlp.modules.attentions.MutiHeadAttention**(*heads=8, d_model=512, dropout=0.9, bias=False, attention_mode='dot'*) [source]

Bases: `Cell`

Muti-head attention is from the paper “attention is all you need” where heads == 1 Muti-head attention is normal self-attention

class **mindnlp.modules.attentions.ScaledDotAttention**(*dropout=0.9*) [source]

Bases: `Cell`

Scaled Dot-Product Attention Scaled Dot-Product Attention proposed in “Attention Is All You Need”

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

MindNLP案例——情感分析

模型构建

```python
import math
from mindspore import nn
from mindspore import ops
from mindspore.common.initializer import Uniform, HeUniform
from mindnlp.abc import Seq2vecModel

class Head(nn.Cell):
    """
    Head for Sentiment Classification model
    """
    def __init__(self, hidden_dim, output_dim, dropout):
        super().__init__()
        weight_init = HeUniform(math.sqrt(5))
        bias_init = Uniform(1 / math.sqrt(hidden_dim * 2))
        self.fc = nn.Dense(hidden_dim * 2, output_dim, weight_init=weight_init, bias_init=bias_init)
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(1 - dropout)

    def construct(self, context):
        context = ops.concat((context[-2, :, :], context[-1, :, :]), axis=1)
        context = self.dropout(context)
        return self.sigmoid(self.fc(context))
```

```python
class SentimentClassification(Seq2vecModel):
    """
    Sentiment Classification model
    """
    def __init__(self, encoder, head):
        super().__init__(encoder, head)
        self.encoder = encoder
        self.head = head

    def construct(self, text):
        _, (hidden, _), _ = self.encoder(text)
        output = self.head(hidden)
        return output
```

数据处理

Load dataset:

```
from mindnlp.dataset import load

imdb_train, imdb_test = load('imdb', shuffle=True)
```

Initializes the vocab and tokenizer for preprocessing:

```
from mindnlp.modules import Glove
from mindnlp.dataset.transforms import BasicTokenizer

embedding, vocab = Glove.from_pretrained('6B', 100, special_tokens=["<unk>", "<pad>"], dropout=drop
tokenizer = BasicTokenizer(True)
```

The loaded dataset is preprocessed and divided into training and validation:

```
from mindnlp.dataset import process

imdb_train = process('imdb', imdb_train, tokenizer=tokenizer, vocab=vocab, \
                     bucket_boundaries=[400, 500], max_len=600, drop_remainder=True)
imdb_train, imdb_valid = imdb_train.split([0.7, 0.3])
```

模型实例化

The following are some of the required hyperparameters in the model training process.

```
# define Models & Loss & Optimizer
hidden_size = 256
output_size = 1
num_layers = 2
bidirectional = True
drop = 0.5
lr = 0.001
```

```
from mindnlp.modules import RNNEncoder

lstm_layer = nn.LSTM(100, hidden_size, num_layers=num_layers, batch_first=True,
                     dropout=drop, bidirectional=bidirectional)
sentiment_encoder = RNNEncoder(embedding, lstm_layer)
sentiment_head = Head(hidden_size, output_size, drop)
net = SentimentClassification(sentiment_encoder, sentiment_head)
```

模型训练与评估

```
from mindnlp.engine.metrics import Accuracy
from mindnlp.engine.trainer import Trainer

# define metrics
metric = Accuracy()

# define trainer
trainer = Trainer(network=net, train_dataset=imdb_train, eval_dataset=imdb_valid, metrics=metric,
                  epochs=5, loss_fn=loss, optimizer=optimizer)
trainer.run(tgt_columns="label", jit=False)
print("end train")
```

欢迎使用MindNLP

master · 1 branch · 0 tags · Go to file

sugarfreeLiuYuXuan add MindNLP Bert modeling (#121) ✓ 0ac84c9

.github	add make_wheel_releases action (#118)
docs	add translation docs (#120)
examples	add BiLSTM-CRF sequence tagging model example (#116)
mindnlp	add MindNLP Bert modeling (#121)
requirements	add machine_translation, update multi30k process (#110)
scripts	move scripts (#22)
tests	add MindNLP Bert modeling (#121)
.gitignore	fix embedding and trainer problems (#71)
LICENSE	init text library
NOTICE	rename to mindnlp
README.md	add README introduction (#119)
setup.py	fix embedding and trainer problems (#71)

Q&A