基于MindSpore的RNN情感分类实验手册



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2025。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验介绍 2](#_Toc173082587)

[1.1 实验目的 2](#_Toc173082588)

[1.2 实验清单 2](#_Toc173082589)

[1.3 开发平台介绍 3](#_Toc173082590)

[2 基于MindSpore的RNN情感分类 4](#_Toc173082591)

[2.1 实验环境 4](#_Toc173082592)

[2.2 数据准备 5](#_Toc173082593)

[2.3 数据集预处理 10](#_Toc173082594)

[2.4 模型构建 11](#_Toc173082595)

[1.1.1 Embedding 11](#_Toc173082596)

[2.4.2 RNN（循环神经网络） 11](#_Toc173082597)

[2.4.3 Dense 13](#_Toc173082598)

[2.4.4 损失函数与优化器 14](#_Toc173082599)

[2.4.5 训练逻辑 14](#_Toc173082600)

[2.4.6 评估指标和逻辑 15](#_Toc173082601)

[2.5 模型训练与保存 16](#_Toc173082602)

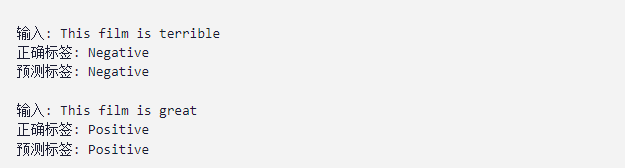
[2.6 模型加载与测试 17](#_Toc173082603)

[2.7 自定义输入测试 17](#_Toc173082604)

[2.8 实验小结 18](#_Toc173082605)

# 实验介绍

情感分类是自然语言处理中的经典任务，是典型的分类问题。本实验通过MindSpore实现一个基于RNN网络的情感分类模型，实现如下的效果：



情感分类任务实现的效果

## 实验目的

本案例使用MindSpore框架进行深度学习网络RNN搭建，利用IMDB影评数据集进行模型训练、准确率验证，最后对英文的影评文本进行情感分类（Positive：积极；Negative：消极）。通过本实验可以了解到深度学习任务开发的具体流程，包括：数据收集、数据处理、模型搭建、模型训练及文本内容情感分类；了解经典循环神经网络RNN的结构特点和不同RNN模型的改进方式；以及深度学习自然语言处理任务中的重要概念，包括：词向量、词嵌入、时序模型、RNN中cell单元的结构、LSTM模型中的门控机制等。

## 实验清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| 基于MindSpore的RNN情感分类 | 本实验使用MindSpore框架搭建RNN网络，利用IMDB影评数据集进行模型训练和文本分类 | 中级 | Python3.9  MindSpore2.2 | ModelArts、 Ascend |

## 开发平台介绍

昇腾（Ascend）是华为自研的一款基于达芬奇架构的AI处理器，具有超高算力的和极高的能效比，最高可达320TFLOPS（FP16）的浮点计算能力。昇腾的片上系统（SoC）还集成了多个CPU、DVPP和任务调度器，因而具有自我管理能力，可以充分发挥其高算力的特点。强大的矩阵、向量运行能力，在进行海量数据运算的神经网络训练场景有巨大的优势。

昇思MindSpore（官方网站：<https://www.mindspore.cn/>）是一种适用于端边云场景的新型开源深度学习训练/推理框架。 MindSpore提供了友好的设计和高效的执行，旨在提升数据科学家和算法工程师的开发体验，并为Ascend AI处理器提供原生支持，以及软硬件协同优化。同时，MindSpore作为全球AI开源社区，致力于进一步开发和丰富AI软硬件应用生态。

ModelArts（官方网站：<https://console.huaweicloud.com/modelarts/>）是面向AI开发者的一站式开发平台，提供海量数据预处理及半自动化标注、大规模分布式训练、自动化模型生成及模型按需部署能力，帮助用户快速创建和部署AI应用，管理全周期AI工作流。

# 基于MindSpore的RNN情感分类

## 实验环境

在开始本实验前，需要完成实验环境搭建工作。

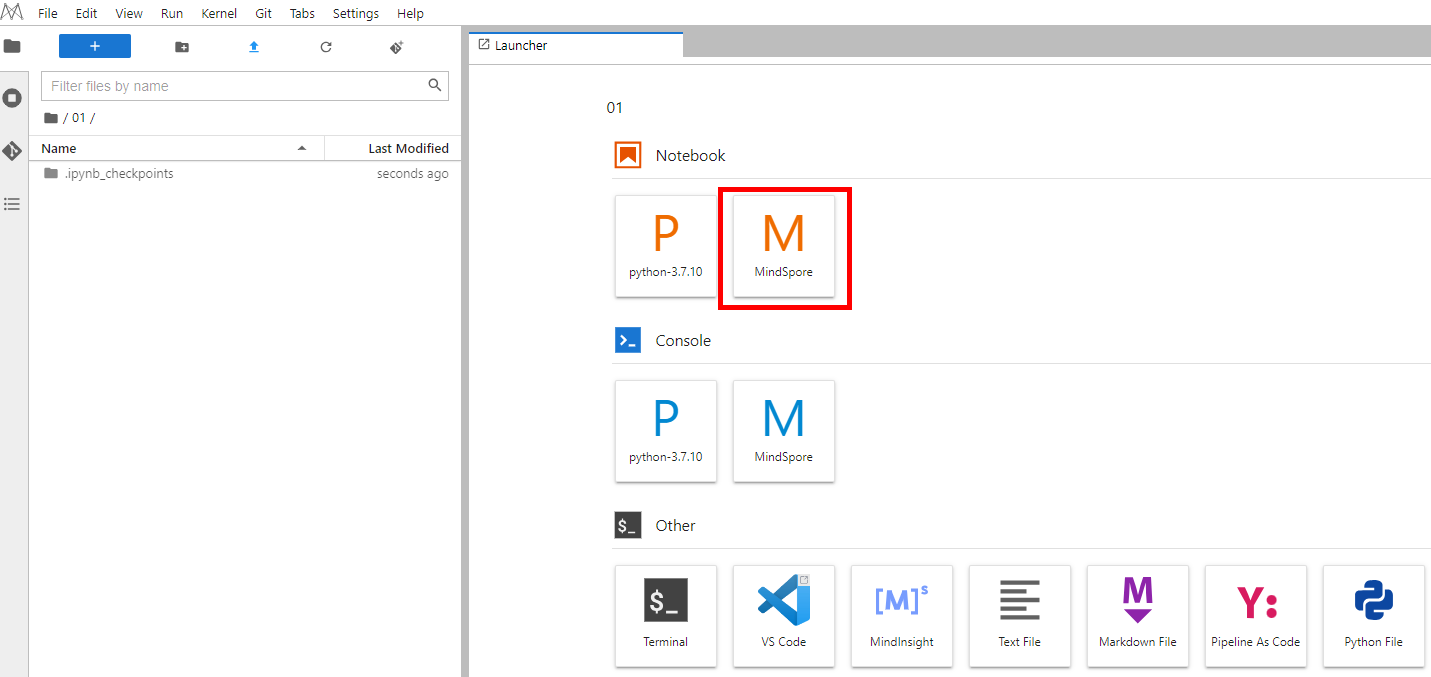
进入ModelArts开发环境

环境搭建方式可参考下方环境搭建手册中 “2.华为云ModelArts环境搭建（训练环境）”部分内容。注意此实验因数据处理量比较大，创建环境时，磁盘规格自定义为200GB。



打开Notebook

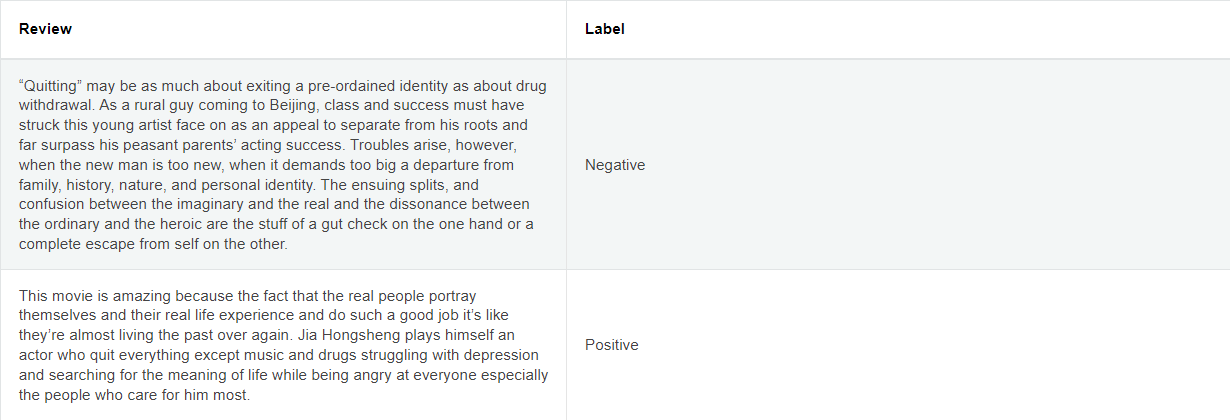
打开Notebook控制台后，新建或打开ipynb文件，选择MindSpore环境作为Kernel，即可开始编辑实验代码。



创建ipynb文件

## 数据准备

本实验使用情感分类的经典数据集[IMDB影评数据集](https://ai.stanford.edu/~amaas/data/sentiment/)，数据集包含Positive和Negative两类，下面为其样例：



IMDB数据集内容示例

此外，需要使用预训练词向量对自然语言单词进行编码，以获取文本的语义特征，本节选取[Glove](https://nlp.stanford.edu/projects/glove/)词向量作为Embedding。

导入依赖库

为了方便数据集和预训练词向量的下载，首先设计数据下载模块，实现可视化下载流程，并保存至指定路径。数据下载模块使用requests库进行http请求，并通过tqdm库对下载百分比进行可视化。此外针对下载安全性，使用IO的方式下载临时文件，而后保存至指定的路径并返回。

import os

import shutil

import requests

import tempfile

from tqdm import tqdm

from typing import IO

from pathlib import Path

from mindspore import context

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend", device\_id=0)

# 指定保存路径为 `home\_path/.mindspore\_examples`

cache\_dir = Path.home() / '.mindspore\_examples'

def http\_get(url: str, temp\_file: IO):

"""使用requests库下载数据，并使用tqdm库进行流程可视化"""

req = requests.get(url, stream=True)

content\_length = req.headers.get('Content-Length')

total = int(content\_length) if content\_length is not None else None

progress = tqdm(unit='B', total=total)

for chunk in req.iter\_content(chunk\_size=1024):

if chunk:

progress.update(len(chunk))

temp\_file.write(chunk)

progress.close()

def download(file\_name: str, url: str):

"""下载数据并存为指定名称"""

if not os.path.exists(cache\_dir):

os.makedirs(cache\_dir)

cache\_path = os.path.join(cache\_dir, file\_name)

cache\_exist = os.path.exists(cache\_path)

if not cache\_exist:

with tempfile.NamedTemporaryFile() as temp\_file:

http\_get(url, temp\_file)

temp\_file.flush()

temp\_file.seek(0)

with open(cache\_path, 'wb') as cache\_file:

shutil.copyfileobj(temp\_file, cache\_file)

return cache\_path

完成数据下载模块后，下载IMDB数据集进行测试(此处使用华为云的镜像用于提升下载速度)。下载过程及保存的路径如下：

#设置no\_proxy环境变量

%env no\_proxy='a.test.com,127.0.0.1,2.2.2.2'

imdb\_path = download('aclImdb\_v1.tar.gz', 'https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/aclImdb\_v1.tar.gz')

imdb\_path

加载IMDB数据集

下载好的IMDB数据集为tar.gz文件，我们使用Python的tarfile库对其进行读取，并将所有数据和标签分别进行存放。原始的IMDB数据集解压目录如下：

├── aclImdb

│ ├── imdbEr.txt

│ ├── imdb.vocab

│ ├── README

│ ├── test

│ └── train

│ ├── neg

│ ├── pos

数据集已分割为train和test两部分，且每部分包含neg和pos两个分类的文件夹，因此需分别train和test进行读取并处理数据和标签。

import re

import six

import string

import tarfile

class IMDBData():

"""IMDB数据集加载器

加载IMDB数据集并处理为一个Python迭代对象。

"""

label\_map = {

"pos": 1,

"neg": 0

}

def \_\_init\_\_(self, path, mode="train"):

self.mode = mode

self.path = path

self.docs, self.labels = [], []

self.\_load("pos")

self.\_load("neg")

def \_load(self, label):

pattern = re.compile(r"aclImdb/{}/{}/.\*\.txt$".format(self.mode, label))

# 将数据加载至内存

with tarfile.open(self.path) as tarf:

tf = tarf.next()

while tf is not None:

if bool(pattern.match(tf.name)):

# 对文本进行分词、去除标点和特殊字符、小写处理

self.docs.append(str(tarf.extractfile(tf).read().rstrip(six.b("\n\r"))

.translate(None, six.b(string.punctuation)).lower()).split())

self.labels.append([self.label\_map[label]])

tf = tarf.next()

def \_\_getitem\_\_(self, idx):

return self.docs[idx], self.labels[idx]

def \_\_len\_\_(self):

return len(self.docs)

完成IMDB数据加载器后，加载训练数据集进行测试，输出数据集数量：

imdb\_train = IMDBData(imdb\_path, 'train')

len(imdb\_train)

输出信息如下：

25000

将IMDB数据集加载至内存并构造为迭代对象后，可以使用mindspore.dataset提供的Generatordataset接口加载数据集迭代对象，并进行下一步的数据处理，下面封装一个函数将train和test分别使用Generatordataset进行加载，并指定数据集中文本和标签的column\_name分别为text和label:

import mindspore.dataset as ds

def load\_imdb(imdb\_path):

imdb\_train = dataset.GeneratorDataset(IMDBData(imdb\_path, "train"), column\_names=["text", "label"], shuffle=True)

imdb\_test = dataset.GeneratorDataset(IMDBData(imdb\_path, "test"), column\_names=["text", "label"], shuffle= False)

return imdb\_train, imdb\_test

加载IMDB数据集，可以看到imdb\_train是一个GeneratorDataset对象。

imdb\_train, imdb\_test = load\_imdb(imdb\_path)

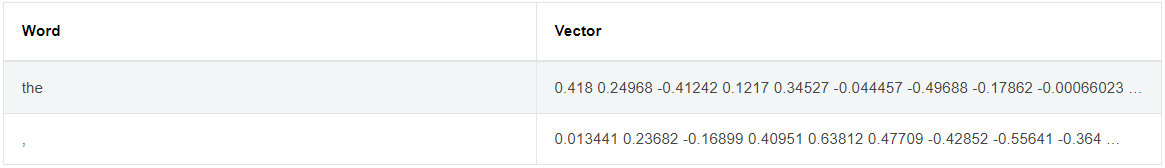
imdb\_train

输出如下信息：

<mindspore.dataset.engine.datasets\_user\_defined.GeneratorDataset at 0x7f3885733710>

加载预训练词向量

预训练词向量是对输入单词的数值化表示，通过nn.Embedding层，采用查表的方式，输入单词对应词表中的index，获得对应的表达向量。 因此进行模型构造前，需要将Embedding层所需的词向量和词表进行构造。这里我们使用Glove(Global Vectors for Word Representation)这种经典的预训练词向量， 其数据格式如下：



预训练词向量格式

我们直接使用第一列的单词作为词表，使用dataset.text.Vocab将其按顺序加载；同时读取每一行的Vector并转为numpy.array，用于nn.Embedding加载权重使用。具体实现如下：

import zipfile

import numpy as np

def load\_glove(glove\_path):

glove\_100d\_path = os.path.join(cache\_dir, 'glove.6B.100d.txt')

if not os.path.exists(glove\_100d\_path):

glove\_zip = zipfile.ZipFile(glove\_path)

glove\_zip.extractall(cache\_dir)

embeddings = []

tokens = []

with open(glove\_100d\_path, encoding='utf-8') as gf:

for glove in gf:

word, embedding = glove.split(maxsplit=1)

tokens.append(word)

embeddings.append(np.fromstring(embedding, dtype=np.float32, sep=' '))

# 添加 <unk>, <pad> 两个特殊占位符对应的embedding

embeddings.append(np.random.rand(100))

embeddings.append(np.zeros((100,), np.float32))

vocab = dataset.text.Vocab.from\_list(tokens, special\_tokens=["<unk>", "<pad>"], special\_first=False)

embeddings = np.array(embeddings).astype(np.float32)

return vocab, embeddings

由于数据集中可能存在词表没有覆盖的单词，因此需要加入<unk>标记符；同时由于输入长度的不一致，在打包为一个batch时需要将短的文本进行填充，因此需要加入<pad>标记符。完成后的词表长度为原词表长度+2。

加载Glove生成词表和词向量权重矩阵。

glove\_path = download('glove.6B.zip', 'https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/glove.6B.zip')

vocab, embeddings = load\_glove(glove\_path)

len(vocab.vocab())

输出信息如下：

400002

使用词表将the转换为index id，并查询词向量矩阵对应的词向量：

idx = vocab.tokens\_to\_ids('the')

embedding = embeddings[idx]

idx, embedding

输出信息如下：

(0,

array([-0.038194, -0.24487 , 0.72812 , -0.39961 , 0.083172, 0.043953,

-0.39141 , 0.3344 , -0.57545 , 0.087459, 0.28787 , -0.06731 ,

0.30906 , -0.26384 , -0.13231 , -0.20757 , 0.33395 , -0.33848 ,

-0.31743 , -0.48336 , 0.1464 , -0.37304 , 0.34577 , 0.052041,

0.44946 , -0.46971 , 0.02628 , -0.54155 , -0.15518 , -0.14107 ,

-0.039722, 0.28277 , 0.14393 , 0.23464 , -0.31021 , 0.086173,

0.20397 , 0.52624 , 0.17164 , -0.082378, -0.71787 , -0.41531 ,

0.20335 , -0.12763 , 0.41367 , 0.55187 , 0.57908 , -0.33477 ,

-0.36559 , -0.54857 , -0.062892, 0.26584 , 0.30205 , 0.99775 ,

-0.80481 , -3.0243 , 0.01254 , -0.36942 , 2.2167 , 0.72201 ,

-0.24978 , 0.92136 , 0.034514, 0.46745 , 1.1079 , -0.19358 ,

-0.074575, 0.23353 , -0.052062, -0.22044 , 0.057162, -0.15806 ,

-0.30798 , -0.41625 , 0.37972 , 0.15006 , -0.53212 , -0.2055 ,

-1.2526 , 0.071624, 0.70565 , 0.49744 , -0.42063 , 0.26148 ,

-1.538 , -0.30223 , -0.073438, -0.28312 , 0.37104 , -0.25217 ,

0.016215, -0.017099, -0.38984 , 0.87424 , -0.72569 , -0.51058 ,

-0.52028 , -0.1459 , 0.8278 , 0.27062 ], dtype=float32))

## 数据集预处理

通过加载器加载的IMDB数据集进行了分词处理，但不满足构造训练数据的需要，因此要对其进行额外的预处理。其中包含的预处理如下：

* 通过Vocab将所有的Token处理为index id。
* 将文本序列统一长度，不足的使用<pad>补齐，超出的进行截断。

这里我们使用mindspore.dataset中提供的接口进行预处理操作。这里使用到的接口均为MindSpore的高性能数据引擎设计，每个接口对应操作视作数据流水线的一部分，详情请参考MindSpore数据引擎。 首先针对token到index id的查表操作，使用text.Lookup接口，将前文构造的词表加载，并指定unknown\_token。其次为文本序列统一长度操作，使用PadEnd接口，此接口定义最大长度和补齐值(pad\_value)，这里我们取最大长度为500，填充值对应词表中<pad>的index id。除了对数据集中text进行预处理外，由于后续模型训练的需要，要将label数据转为float32格式。

import mindspore as ms

lookup\_op = dataset.text.Lookup(vocab, unknown\_token='<unk>')

pad\_op = dataset.transforms.c\_transforms.PadEnd([500], pad\_value=vocab.tokens\_to\_ids('<pad>'))

type\_cast\_op = dataset.transforms.c\_transforms.TypeCast(mindspore.float32)

完成预处理操作后，需将其加入到数据集处理流水线中，使用map接口对指定的column添加操作。

imdb\_train = imdb\_train.map(operations=[lookup\_op, pad\_op], input\_columns=['text'])

imdb\_train = imdb\_train.map(operations=[type\_cast\_op], input\_columns=['label'])

imdb\_test = imdb\_test.map(operations=[lookup\_op, pad\_op], input\_columns=['text'])

imdb\_test = imdb\_test.map(operations=[type\_cast\_op], input\_columns=['label'])

由于IMDB数据集本身不包含验证集，我们手动将其分割为训练和验证两部分，比例取0.7, 0.3。

imdb\_train, imdb\_valid = imdb\_train.split([0.7, 0.3])

最后指定数据集的batch大小，通过batch接口指定，并设置是否丢弃无法被batch size整除的剩余数据。

imdb\_train = imdb\_train.batch(64, drop\_remainder=True)

imdb\_valid = imdb\_valid.batch(64, drop\_remainder=True)

## 模型构建

完成数据集的处理后，我们设计用于情感分类的模型结构。首先需要将输入文本(即序列化后的index id列表)通过查表转为向量化表示，此时需要使用nn.Embedding层加载Glove词向量；然后使用RNN循环神经网络做特征提取；最后将RNN连接至一个全连接层，即nn.Dense，将特征转化为与分类数量相同的size，用于后续进行模型优化训练。整体模型结构如下：

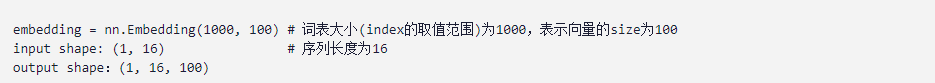


模型结构

这里我们使用能够一定程度规避RNN梯度消失问题的变种LSTM(Long short term memory)做特征提取层。下面对模型进行详解：

### Embedding

Embedding层又可称为EmbeddingLookup层，其作用是使用index id对权重矩阵对应id的向量进行查找，当输入为一个由index id组成的序列时，则查找并返回一个相同长度的矩阵，例如：

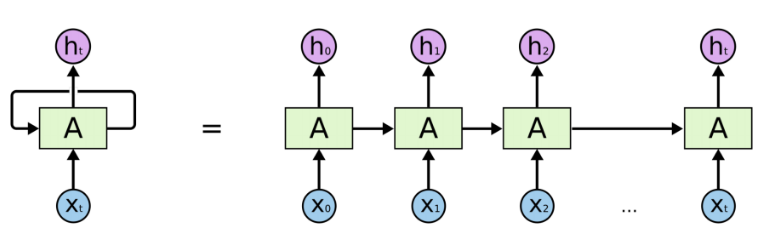


词向量接口使用示例

这里我们使用前文处理好的Glove词向量矩阵，设置nn.Embedding的embedding\_table为预训练词向量矩阵。对应的vocab\_size为词表大小400002，embedding\_size为选用的glove.6B.100d向量大小，即100。

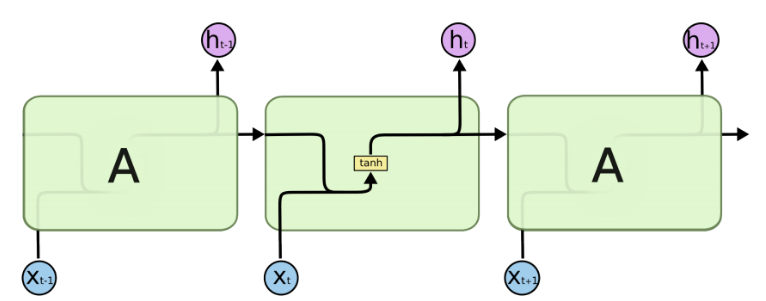
### RNN（循环神经网络）

循环神经网络（Recurrent Neural Network, RNN）是一类以序列（sequence）数据为输入，在序列的演进方向进行递归（recursion）且所有节点（循环单元）按链式连接的神经网络。下图为RNN的一般结构：



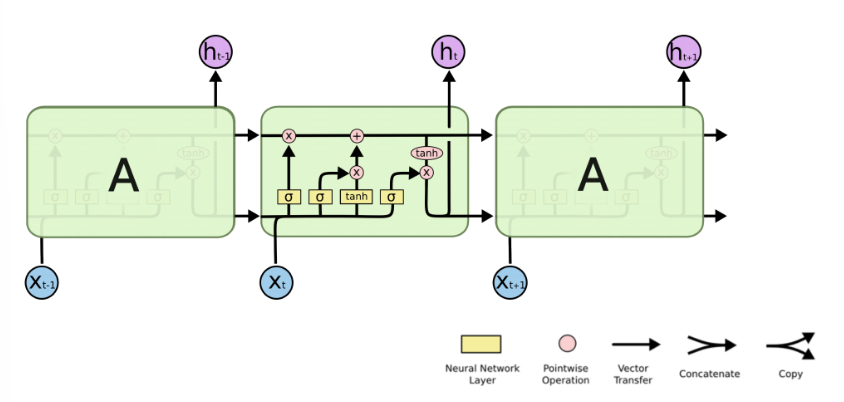
RNN结构特点

由于RNN的循环特性，和自然语言文本的序列特性(句子是由单词组成的序列)十分匹配，因此被大量应用于自然语言处理研究中。下图为RNN的结构拆解：



RNN计算单元内部结构

RNN单个Cell的结构简单，因此也造成了梯度消失(Gradient Vanishing)问题，具体表现为RNN网络在序列较长时，在序列尾部已经基本丢失了序列首部的信息。为了克服这一问题，LSTM(Long short term memory)被提出，通过门控机制(Gating Mechanism)来控制信息流在每个循环步中的留存和丢弃。下图为LSTM的结构拆解：



LSTM单元内部结构

本节我们选择LSTM变种而不是经典的RNN做特征提取，来规避梯度消失问题，并获得更好的模型效果。下面来看MindSpore中nn.LSTM对应的公式：

这里nn.LSTM隐藏了整个循环神经网络在序列时间步(Time step)上的循环，送入输入序列、初始状态，即可获得每个时间步的隐状态(hidden state)拼接而成的矩阵，以及最后一个时间步对应的隐状态。我们使用最后的一个时间步的隐状态作为输入句子的编码特征，送入下一层。

### Dense

在经过LSTM编码获取句子特征后，将其送入一个全连接层，即nn.Dense，将特征维度变换为二分类所需的维度1，经过Dense层后的输出即为模型预测结果。

import math

import mindspore as ms

import mindspore.nn as nn

import mindspore.ops as ops

from mindspore.common.initializer import Uniform, HeUniform

class RNN(nn.Cell):

def \_\_init\_\_(self, embeddings, hidden\_dim, output\_dim, n\_layers,

bidirectional, dropout, pad\_idx):

super().\_\_init\_\_()

vocab\_size, embedding\_dim = embeddings.shape

self.embedding = nn.Embedding(vocab\_size, embedding\_dim, embedding\_table=Tensor(embeddings), padding\_idx=pad\_idx)

self.rnn = nn.LSTM(embedding\_dim,

hidden\_dim,

num\_layers=n\_layers,

bidirectional=bidirectional,

dropout=dropout,

batch\_first=True)

self.fc = nn.Dense(hidden\_dim \* 2, output\_dim)

self.dropout = nn.Dropout(1 - dropout)

self.sigmoid = ops.Sigmoid()

def construct(self, inputs):

embedded = self.dropout(self.embedding(inputs))

\_, (hidden, \_) = self.rnn(embedded)

hidden = self.dropout(mnp.concatenate((hidden[-2, :, :], hidden[-1, :, :]), axis=1))

output = self.fc(hidden)

return self.sigmoid(output)

### 损失函数与优化器

完成模型主体构建后，首先根据指定的参数实例化网络；然后选择损失函数和优化器，并使用nn.TrainOneStepCell对其进行封装。针对本节情感分类问题的特性，即预测Positive或Negative的二分类问题，我们选择nn.BCELoss(二分类交叉熵损失函数)，这里也可以选择``nn.BCEWithLogitsLoss``, 其包含sigmoid运算，即：



损失计算公式

这里使用BECLoss需要设置reduction参数为均值。而后使用nn.WithLossCell将其与实例化网络对象进行关联。

选择合适的损失函数后，选择Adam优化器，并将二者传入TrainOneStepCell中。

hidden\_size = 256

output\_size = 1

num\_layers = 2

bidirectional = True

lr = 0.001

pad\_idx = vocab.tokens\_to\_ids('<pad>')

model = RNN(embeddings, hidden\_size, output\_size, num\_layers, bidirectional, pad\_idx)

loss\_fn = nn.BCEWithLogitsLoss(reduction='mean')

optimizer = nn.Adam(model.trainable\_params(), learning\_rate=lr)

### 训练逻辑

在完成模型构建，进行训练逻辑的设计。一般训练逻辑分为以下步骤：

* 读取一个Batch的数据；
* 送入网络，进行正向计算和反向传播，更新权重；
* 返回loss。

下面按照此逻辑，使用tqdm库，设计训练一个epoch的函数，用于训练过程和loss的可视化。

def forward\_fn(data, label):

logits = model(data)

loss = loss\_fn(logits, label)

return loss

grad\_fn = ms.value\_and\_grad(forward\_fn, None, optimizer.parameters)

def train\_step(data, label):

loss, grads = grad\_fn(data, label)

optimizer(grads)

return loss

def train\_one\_epoch(model, train\_dataset, epoch=0):

model.set\_train()

total = train\_dataset.get\_dataset\_size()

loss\_total = 0

step\_total = 0

with tqdm(total=total) as t:

t.set\_description('Epoch %i' % epoch)

for i in train\_dataset.create\_tuple\_iterator():

loss = train\_step(\*i)

loss\_total += loss.asnumpy()

step\_total += 1

t.set\_postfix(loss=loss\_total/step\_total)

t.update(1)

### 评估指标和逻辑

训练逻辑完成后，需要对模型进行评估。即使用模型的预测结果和测试集的正确标签进行对比，求出预测的准确率。由于IMDB的情感分类为二分类问题，对预测值直接进行四舍五入即可获得分类标签(0或1)，然后判断是否与正确标签相等即可。下面为二分类准确率计算函数实现：

def binary\_accuracy(preds, y):

"""

计算每个batch的准确率

"""

# 对预测值进行四舍五入

rounded\_preds = np.around(preds)

correct = (rounded\_preds == y).astype(np.float32)

acc = correct.sum() / len(correct)

return acc

有了准确率计算函数后，类似于训练逻辑，对评估逻辑进行设计, 分别为以下步骤：

* 读取一个Batch的数据；
* 送入网络，进行正向计算，获得预测结果；
* 计算准确率。

同训练逻辑一样，使用tqdm进行loss和过程的可视化。此外返回评估loss至供保存模型时作为模型优劣的判断依据。

在进行evaluate时，使用的模型是不包含损失函数和优化器的网络主体； 在进行evaluate前，需要通过model.set\_train(False)将模型置为评估状态，此时Dropout不生效。

def evaluate(model, test\_dataset, criterion, epoch=0):

total = test\_dataset.get\_dataset\_size()

epoch\_loss = 0

epoch\_acc = 0

step\_total = 0

model.set\_train(False)

with tqdm(total=total) as t:

t.set\_description('Epoch %i' % epoch)

for i in test\_dataset.create\_tuple\_iterator():

predictions = model(i[0])

loss = criterion(predictions, i[1])

epoch\_loss += loss.asnumpy()

acc = binary\_accuracy(predictions.asnumpy(), i[1].asnumpy())

epoch\_acc += acc

step\_total += 1

t.set\_postfix(loss=epoch\_loss/step\_total, acc=epoch\_acc/step\_total)

t.update(1)

return epoch\_loss / total

## 模型训练与保存

前序完成了模型构建和训练、评估逻辑的设计，下面进行模型训练。这里我们设置训练轮数为5轮。同时维护一个用于保存最优模型的变量best\_valid\_loss，根据每一轮评估的loss值，取loss值最小的轮次，将模型进行保存。

num\_epochs = 5

best\_valid\_loss = float('inf')

ckpt\_file\_name = os.path.join(cache\_dir, 'sentiment-analysis.ckpt')

for epoch in range(num\_epochs):

train\_one\_epoch(model, imdb\_train, epoch)

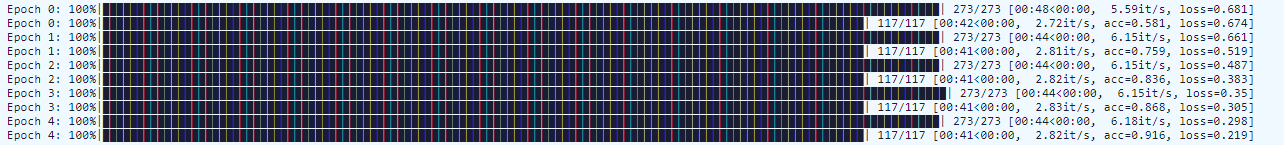
valid\_loss = evaluate(model, imdb\_valid, loss\_fn, epoch)

if valid\_loss < best\_valid\_loss:

best\_valid\_loss = valid\_loss

ms.save\_checkpoint(model, ckpt\_file\_name)

训练过程输出如下信息：



模型训练过程输出信息

## 模型加载与测试

模型训练完成后，一般需要对模型进行测试或部署上线，此时需要加载已保存的最优模型(即checkpoint)，供后续测试使用。这里我们直接使用MindSpore提供的Checkpoint加载和网络权重加载接口：

* 将保存的模型Checkpoint加载到内存中，
* 将Checkpoint加载至模型，
* 对测试集按照batchsize切分，然后使用evaluate方法进行评估，得到模型在测试集上的效果。

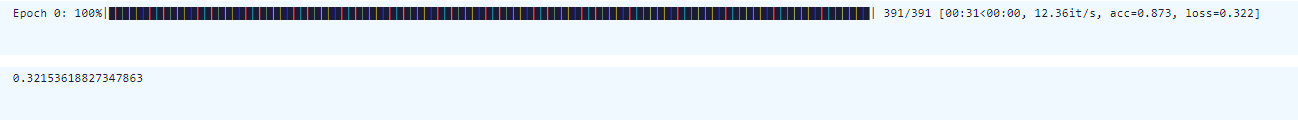
param\_dict = ms.load\_checkpoint(ckpt\_file\_name)

ms.load\_param\_into\_net(model, param\_dict)

imdb\_test = imdb\_test.batch(64)

evaluate(model, imdb\_test, loss\_fn)

输出信息如下：



模型评估过程及输出信息

## 自定义输入测试

最后我们设计一个预测函数，实现开头描述的效果，输入一句评价，获得评价的情感分类。具体包含以下步骤:

* 将输入句子进行分词；
* 使用词表获取对应的index id序列；
* index id序列转为Tensor；
* 送入模型获得预测结果；
* 打印输出预测结果。

具体实现如下

score\_map = {

1: "Positive",

0: "Negative"

}

def predict\_sentiment(model, vocab, sentence):

model.set\_train(False)

tokenized = sentence.lower().split()

indexed = vocab.tokens\_to\_ids(tokenized)

tensor = ms.Tensor(indexed, ms.int32)

tensor = tensor.expand\_dims(0)

prediction = model(tensor)

return score\_map[int(np.round(ops.sigmoid(prediction).asnumpy()))]

最后我们预测开头的样例，可以看到模型可以很好地将评价语句的情感进行分类。

测试样例1：

predict\_sentiment(model, vocab, "This film is terrible")

输出信息如下：

'Negative'

测试样例2：

predict\_sentiment(model, vocab, "This film is great")

输出信息如下：

'Positive'

## 实验小结

本案例通过在ModelArts平台创建拥有昇腾AI加速芯片的NoteBook开发环境，使用MindSpore深度学习框架来搭建RNN循环神经网络，采用IMDB影评数据集进行模型训练、准确率验证，最后我们通过两个测试样例对模型的预测效果进行了演示。通过这个实验，我们可以了解MindSpore框架和ModelArts平台的使用，了解RNN循环神经网络模型的结构和设计特点，了解词嵌入、经典RNN网络结构LSTM搭建、门控单元等深度学习自然语言处理技术，了解如何使用一个深度学习模型来解决自然语言处理的具体任务。