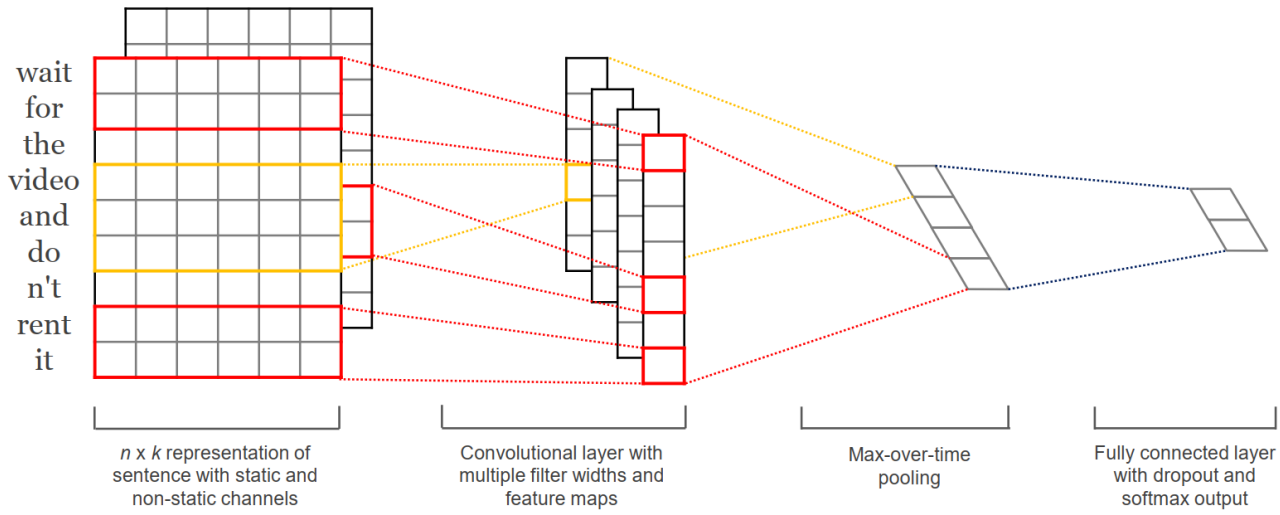
实验三：使用卷积神经网络TextCNN进行文本识别

实验内容

THUCNews是一个中文文本分类数据集，根据新浪新闻RSS订阅频道2005~2011年间的历史数据筛选过滤生成。在原始新浪新闻分类体系的基础上，标注者重新整合划分出14个候选分类类别：财经、彩票、房产、股票、家居、教育、科技、社会、时尚、时政、体育、星座、游戏、娱乐。

原始的THUCNew数据量庞大，难以处理和使用，故本次实验对数据进行了简单抽样，抽取出训练集180000条，验证集10000 条，测试集10000 条，将由助教统一下发

TextCNN是一种使用CNN进行文本分类的模型，其原理图如下所示



卷积神经网络的核心思想是捕捉局部特征，对于文本来说，局部特征就是由若干单词组成的滑动窗口，这在NLP任务中被称为N元语法(N-gram)， N-gram已经被证明是有用的文本特征。TextCNN的优势在于能够自动地对N-gram特征进行组合和筛选，获得不同抽象层次的语义信息。

实验要求

1. 使用Pytorch实现一个TEXTCNN，并完成对THUCNews数据集的训练和测试。
2. 在测试集上的准确率>75%

实验考察能力

1. 掌握卷积神经网络的实现

实验指导

准备工作

除torch, numpy等库外，本次实验需要安装sklearn和tqdm库，可以执行以下命令进行安装

```
pip install scikit-Learn
pip install tqdm
```

数据集处理

使用下列代码，读取助教提供的训练集和测试集文件即可。

```
def read_data(file):
    with open(file, encoding="utf-8") as f:
        all_data = f.read().split("\n")

    texts, labels = [], []
    for data in all_data:
        if data:
            text, label = data.split("\t")
            texts.append(text)
            labels.append(label)
    return texts, labels


def built_curpus(train_texts, embedding_num):
    word_2_index = {"<PAD>": 0, "<UNK>": 1}
    for text in train_texts:
        for word in text:
            word_2_index[word] = word_2_index.get(word, len(word_2_index))
    embedding = nn.Embedding(len(word_2_index), embedding_num)
    pkl.dump([word_2_index, embedding], open(parsers().data_pkl, "wb"))
    return word_2_index, embedding


class TextDataset(Dataset):
    def __init__(self, all_text, all_label, word_2_index, max_len):
        self.all_text = all_text
        self.all_label = all_label
        self.word_2_index = word_2_index
        self.max_len = max_len

    def __getitem__(self, index):
        text = self.all_text[index][:self.max_len]
        label = int(self.all_label[index])

        text_idx = [self.word_2_index.get(i, 1) for i in text]
        text_idx = text_idx + [0] * (self.max_len - len(text_idx))

        text_idx = torch.tensor(text_idx).unsqueeze(dim=0)

        return text_idx, label

    def __len__(self):
        return len(self.all_text)
```

卷积神经网络构建

我们可以使用如下代码构建TextCNN

```
class Block(nn.Module):
    def __init__(self, kernel_s, embeddin_num, max_len, hidden_num):
        super().__init__()
        # shape [batch *  in_channel * max_len * emb_num]
        self.cnn = nn.Conv2d(in_channels=1, out_channels=hidden_num, kernel_size=
(kernel_s, embeddin_num))
        self.act = nn.ReLU()
        self.mxp = nn.MaxPool1d(kernel_size=(max_len - kernel_s + 1))

    def forward(self, batch_emb):  # shape [batch *  in_channel * max_len *
emb_num]
        c = self.cnn(batch_emb)
        a = self.act(c)
        a = a.squeeze(dim=-1)
        m = self.mxp(a)
        m = m.squeeze(dim=-1)
        return m


class TextCNNModel(nn.Module):
    def __init__(self, emb_matrix, max_len, class_num, hidden_num):
        super().__init__()
        self.emb_num = emb_matrix.weight.shape[1]

        self.block1 = Block(2, self.emb_num, max_len, hidden_num)
        self.block2 = Block(3, self.emb_num, max_len, hidden_num)
        self.block3 = Block(4, self.emb_num, max_len, hidden_num)

        self.emb_matrix = emb_matrix

        self.classifier = nn.Linear(hidden_num * 3, class_num)  # 2 * 3
        self.loss_fun = nn.CrossEntropyLoss()

    def forward(self, batch_idx):   # shape torch.Size([batch_size, 1, max_len])
        batch_emb = self.emb_matrix(batch_idx)   # shape torch.Size([batch_size,
1, max_len, embedding])
        b1_result = self.block1(batch_emb)  # shape torch.Size([batch_size, 2])
        b2_result = self.block2(batch_emb)  # shape torch.Size([batch_size, 2])
        b3_result = self.block3(batch_emb)  # shape torch.Size([batch_size, 2])

        # 拼接
        feature = torch.cat([b1_result, b2_result, b3_result], dim=1)  # shape
torch.Size([batch_size, 6])
        pre = self.classifier(feature)  # shape torch.Size([batch_size,
class_num])
```

```
        return pre
```

构建损失函数，优化器

```
opt = torch.optim.AdamW(model.parameters(), lr=args.learn_rate)
loss_fn = nn.CrossEntropyLoss()
```

训练并测试模型

```
for epoch in range(args.epochs):
        model.train()
        loss_sum, count = 0, 0
        for batch_index, (batch_text, batch_label) in enumerate(train_loader):
            batch_text, batch_label = batch_text.to(device),
batch_label.to(device)
            pred = model(batch_text)

            loss = loss_fn(pred, batch_label)
            opt.zero_grad()
            loss.backward()
            opt.step()

            loss_sum += loss
            count += 1

            if len(train_loader) - batch_index <= len(train_loader) % 1000 and
count == len(train_loader) % 1000:
                msg = "[{0}/{1:5d}]\tTrain_Loss:{2:.4f}"
                print(msg.format(epoch + 1, batch_index + 1, loss_sum / count))
                loss_sum, count = 0.0, 0

            if batch_index % 1000 == 999:
                msg = "[{0}/{1:5d}]\tTrain_Loss:{2:.4f}"
                print(msg.format(epoch + 1, batch_index + 1, loss_sum / count))
                loss_sum, count = 0.0, 0

        model.eval()
        all_pred, all_true = [], []
        with torch.no_grad():
            for batch_text, batch_label in dev_loader:
                batch_text = batch_text.to(device)
                batch_label = batch_label.to(device)
                pred = model(batch_text)

                pred = torch.argmax(pred, dim=1)
                pred = pred.cpu().numpy().tolist()
                label = batch_label.cpu().numpy().tolist()
```

```
                all_pred.extend(pred)
                all_true.extend(label)

        acc = accuracy_score(all_pred, all_true)
        print(f"dev acc:{acc:.4f}")

        if acc > acc_max:
            acc_max = acc
            torch.save(model.state_dict(), args.save_model_best)
            print(f"已保存最佳模型")
        print("*"*50)
```

完整代码

```
import os
import torch
import torch.nn as nn
import argparse
import os.path
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
import pickle as pkl
import pickle as pkl
from sklearn.metrics import accuracy_score
import time
from test import test_data

def read_data(file):
    with open(file, encoding="utf-8") as f:
        all_data = f.read().split("\n")

    texts, labels = [], []
    for data in all_data:
        if data:
            text, label = data.split("\t")
            texts.append(text)
            labels.append(label)
    return texts, labels


def built_curpus(train_texts, embedding_num):
    word_2_index = {"<PAD>": 0, "<UNK>": 1}
    for text in train_texts:
        for word in text:
            word_2_index[word] = word_2_index.get(word, len(word_2_index))
    embedding = nn.Embedding(len(word_2_index), embedding_num)
    pkl.dump([word_2_index, embedding], open(parsers().data_pkl, "wb"))
    return word_2_index, embedding


class TextDataset(Dataset):
```

```
    def __init__(self, all_text, all_label, word_2_index, max_len):
        self.all_text = all_text
        self.all_label = all_label
        self.word_2_index = word_2_index
        self.max_len = max_len

    def __getitem__(self, index):
        text = self.all_text[index][:self.max_len]
        label = int(self.all_label[index])

        text_idx = [self.word_2_index.get(i, 1) for i in text]
        text_idx = text_idx + [0] * (self.max_len - len(text_idx))

        text_idx = torch.tensor(text_idx).unsqueeze(dim=0)

        return text_idx, label

    def __len__(self):
        return len(self.all_text)

class Block(nn.Module):
    def __init__(self, kernel_s, embeddin_num, max_len, hidden_num):
        super().__init__()
        # shape [batch *  in_channel * max_len * emb_num]
        self.cnn = nn.Conv2d(in_channels=1, out_channels=hidden_num, kernel_size=
(kernel_s, embeddin_num))
        self.act = nn.ReLU()
        self.mxp = nn.MaxPool1d(kernel_size=(max_len - kernel_s + 1))

    def forward(self, batch_emb):  # shape [batch *  in_channel * max_len *
emb_num]
        c = self.cnn(batch_emb)
        a = self.act(c)
        a = a.squeeze(dim=-1)
        m = self.mxp(a)
        m = m.squeeze(dim=-1)
        return m


class TextCNNModel(nn.Module):
    def __init__(self, emb_matrix, max_len, class_num, hidden_num):
        super().__init__()
        self.emb_num = emb_matrix.weight.shape[1]

        self.block1 = Block(2, self.emb_num, max_len, hidden_num)
        self.block2 = Block(3, self.emb_num, max_len, hidden_num)
        self.block3 = Block(4, self.emb_num, max_len, hidden_num)

        self.emb_matrix = emb_matrix

        self.classifier = nn.Linear(hidden_num * 3, class_num)  # 2 * 3
        self.loss_fun = nn.CrossEntropyLoss()

    def forward(self, batch_idx):   # shape torch.Size([batch_size, 1, max_len])
```

```
        batch_emb = self.emb_matrix(batch_idx)   # shape torch.Size([batch_size,
1, max_len, embedding])
        b1_result = self.block1(batch_emb)  # shape torch.Size([batch_size, 2])
        b2_result = self.block2(batch_emb)  # shape torch.Size([batch_size, 2])
        b3_result = self.block3(batch_emb)  # shape torch.Size([batch_size, 2])

        # 拼接
        feature = torch.cat([b1_result, b2_result, b3_result], dim=1)  # shape
torch.Size([batch_size, 6])
        pre = self.classifier(feature)  # shape torch.Size([batch_size,
class_num])

        return pre

def test_data():
    args = parsers()

    device = "cuda:0" if torch.cuda.is_available() else "cpu"

    dataset = pkl.load(open(args.data_pkl, "rb"))
    word_2_index, words_embedding = dataset[0], dataset[1]

    test_text, test_label = read_data(args.test_file)
    test_dataset = TextDataset(test_text, test_label, word_2_index, args.max_len)
    test_dataloader = DataLoader(test_dataset, batch_size=args.batch_size,
shuffle=False)

    model = TextCNNModel(words_embedding, args.max_len, args.class_num,
args.num_filters).to(device)
    model.load_state_dict(torch.load(args.save_model_best))
    model.eval()

    all_pred, all_true = [], []
    with torch.no_grad():
        for batch_text, batch_label in test_dataloader:
            batch_text, batch_label = batch_text.to(device),
batch_label.to(device)
            pred = model(batch_text)
            pred = torch.argmax(pred, dim=1)

            pred = pred.cpu().numpy().tolist()
            label = batch_label.cpu().numpy().tolist()

            all_pred.extend(pred)
            all_true.extend(label)

    accuracy = accuracy_score(all_true, all_pred)

    print(f"test dataset accuracy:{accuracy:.4f}")

def parsers():
    parser = argparse.ArgumentParser(description="TextCNN model of argparse")
    parser.add_argument("--train_file", type=str, default=os.path.join("data",
"train.txt"))
```

```
    parser.add_argument("--dev_file", type=str, default=os.path.join("data",
"dev.txt"))
    parser.add_argument("--test_file", type=str, default=os.path.join("data",
"test.txt"))
    parser.add_argument("--classification", type=str, default=os.path.join("data",
"class.txt"))
    parser.add_argument("--data_pkl", type=str, default=os.path.join("data",
"dataset.pkl"))
    parser.add_argument("--class_num", type=int, default=10)
    parser.add_argument("--max_len", type=int, default=38)
    parser.add_argument("--embedding_num", type=int, default=100)
    parser.add_argument("--batch_size", type=int, default=32)
    parser.add_argument("--epochs", type=int, default=30)
    parser.add_argument("--learn_rate", type=float, default=1e-3)
    parser.add_argument("--num_filters", type=int, default=2, help="卷积产生的通道
数")
    parser.add_argument("--save_model_best", type=str,
default=os.path.join("model", "best_model.pth"))
    parser.add_argument("--save_model_last", type=str,
default=os.path.join("model", "last_model.pth"))
    args = parser.parse_args()
    return args


if __name__ == "__main__":
    start = time.time()
    args = parsers()
    train_text, train_label = read_data(args.train_file)
    dev_text, dev_label = read_data(args.dev_file)

    device = "cuda:0" if torch.cuda.is_available() else "cpu"

    if os.path.exists(args.data_pkl):
        dataset = pkl.load(open(args.data_pkl, "rb"))
        word_2_index, words_embedding = dataset[0], dataset[1]
    else:
        word_2_index, words_embedding = built_curpus(train_text,
args.embedding_num)

    train_dataset = TextDataset(train_text, train_label, word_2_index,
args.max_len)
    train_loader = DataLoader(train_dataset, args.batch_size, shuffle=True)

    dev_dataset = TextDataset(dev_text, dev_label, word_2_index, args.max_len)
    dev_loader = DataLoader(dev_dataset, args.batch_size, shuffle=False)

    model = TextCNNModel(words_embedding, args.max_len, args.class_num,
args.num_filters).to(device)
    opt = torch.optim.AdamW(model.parameters(), lr=args.learn_rate)
    loss_fn = nn.CrossEntropyLoss()

    acc_max = float("-inf")
    for epoch in range(args.epochs):
        model.train()
```

```
        loss_sum, count = 0, 0
        for batch_index, (batch_text, batch_label) in enumerate(train_loader):
            batch_text, batch_label = batch_text.to(device),
batch_label.to(device)
            pred = model(batch_text)

            loss = loss_fn(pred, batch_label)
            opt.zero_grad()
            loss.backward()
            opt.step()

            loss_sum += loss
            count += 1

            # 打印内容
            if len(train_loader) - batch_index <= len(train_loader) % 1000 and
count == len(train_loader) % 1000:
                msg = "[{0}/{1:5d}]\tTrain_Loss:{2:.4f}"
                print(msg.format(epoch + 1, batch_index + 1, loss_sum / count))
                loss_sum, count = 0.0, 0

            if batch_index % 1000 == 999:
                msg = "[{0}/{1:5d}]\tTrain_Loss:{2:.4f}"
                print(msg.format(epoch + 1, batch_index + 1, loss_sum / count))
                loss_sum, count = 0.0, 0

        model.eval()
        all_pred, all_true = [], []
        with torch.no_grad():
            for batch_text, batch_label in dev_loader:
                batch_text = batch_text.to(device)
                batch_label = batch_label.to(device)
                pred = model(batch_text)

                pred = torch.argmax(pred, dim=1)
                pred = pred.cpu().numpy().tolist()
                label = batch_label.cpu().numpy().tolist()

                all_pred.extend(pred)
                all_true.extend(label)

        acc = accuracy_score(all_pred, all_true)
        print(f"dev acc:{acc:.4f}")

        if acc > acc_max:
            acc_max = acc
            torch.save(model.state_dict(), args.save_model_best)
            print(f"已保存最佳模型")
        print("*"*50)


    end = time.time()
    print(f"运行时间：{(end-start)/60%60:.4f} min")
    test_data()
```