郑不凡
码龄2年 西华大学

TextCNN pytorch实现

原创 郑不凡 于 2021-11-02 16:59:12 发布 256 收藏 2 版权
文章标签： pytorch cnn 深度学习

TextCNN结构

以下为原论文中的模型结构图：

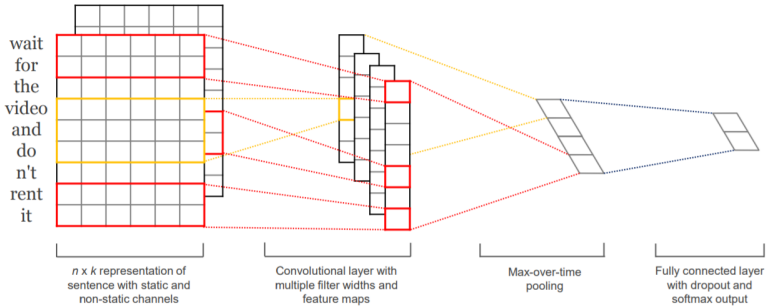


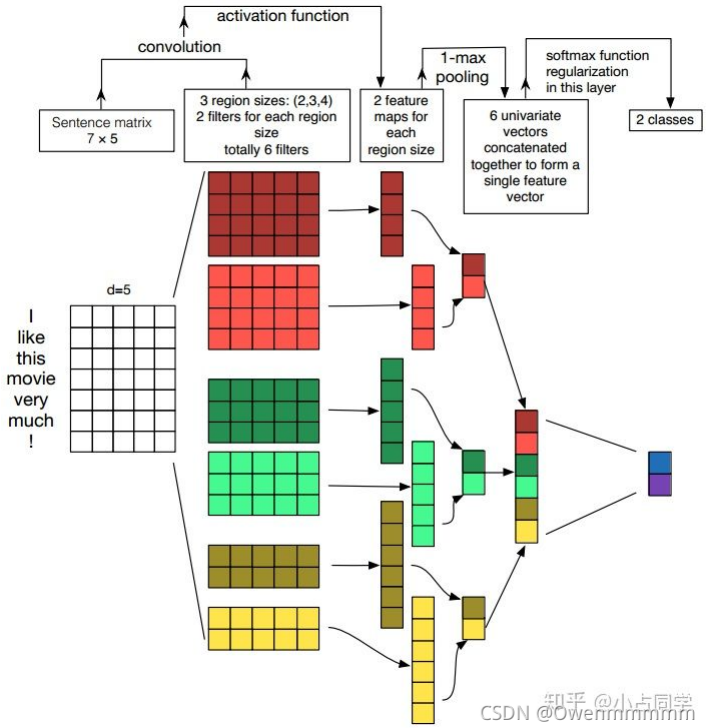
Figure 1: Model architecture with two channels for an example sentence.

1. embedding
2. 卷积层
3. MaxPooling
4. Flatten
5. 全连接层

卷积操作

- 在卷积神经网络中仅涉及**离散卷积**的情形。
- 卷积运算的作用就类似与滤波，因此也称卷积核为filter滤波器。
- 卷积神经网络的核心思想是**捕捉局部特征**（n-gram）。CNN的优势在于能够自动地对g-gram特征进行组合和筛选，获得不同抽象层次的语义信息。
- 下图为用于文本分类任务的TextCNN结构描述（这里详细解释了TextCNN架构以及词向量矩阵是如何做卷积的）

目录



1. 输入层：$n * k$的矩阵，n为句子中的单词数，k为embedding_size。（为了使向量长度一致，对原句进行了padding操作）
2. 卷积层：在NLP中输入层是一个由词向量拼成的词矩阵，且卷积核的宽和该词矩阵的宽相同，该宽度即为词向量大小，且卷积核只会在高度方向移动。输入层的矩阵与我们的filter进行convolution，然后经过激活函数得到feature map。filter这里有三种大小（3，4，5）。
3. 池化层：max-pooling
4. softmax输出结果。

简单代码实现

```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as Data
import torch.nn.functional as F

dtype = torch.FloatTensor
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# 3 words sentences (=sequence_length is 3)
sentences = ["i love you", "he loves me", "she likes baseball", "i hate you", "sorr
labels = [1, 1, 1, 0, 0, 0]  # 1 is good, 0 is not good.

embedding_size = 2
sequence_length = len(sentences[0])
num_classes = len(set(labels))
batch_size = 3
```

```
word2idx = {w:i for i,w in enumerate(vocab)}
vocab_size = len(vocab)
```

```
def make_data(sentences, labels):
    inputs = []
    for sen in sentences:
        inputs.append([word2idx[n] for n in sen.split()])

    targets = []
    for out in labels:
        targets.append(out)

    return inputs, targets
```

```
input_batch, target_batch = make_data(sentences, labels)
input_batch, target_batch = torch.LongTensor(input_batch), torch.LongTensor(target_

dataset = Data.TensorDataset(input_batch,target_batch)
loader = Data.DataLoader(dataset, batch_size, True)
```

```
class TextCNN(nn.Module):

    def __init__(self):
        super(TextCNN, self).__init__()
        self.W = nn.Embedding(vocab_size, embedding_size)
        output_channel = 3
        self.conv = nn.Sequential(nn.Conv2d(1, output_channel, (2,embedding_size)),
                                  nn.ReLU(),
                                  nn.MaxPool2d((2,1)))
        self.fc = nn.Linear(output_channel,num_classes)

    def forward(self, X):
      '''
      X: [batch_size, sequence_length]
      '''
      batch_size = X.shape[0]
      embedding_X = self.W(X) # [batch_size, sequence_length, embedding_size]
      embedding_X = embedding_X.unsqueeze(1) # add channel(=1) [batch, channel(=1),
      conved = self.conv(embedding_X) # [batch_size, output_channel,1,1]
      flatten = conved.view(batch_size, -1)# [batch_size, output_channel*1*1]
      output = self.fc(flatten)
      return output
```

维度变换

1. 输入X：[batch_size, sequence_length]
2. embedding：相当于把单词增加了一个维度。[batch_size, sequence_length, embedding_size]；然后我们对它做了一个unsqueeze（1）操作，原因是卷积操作的需要。[batch_size, channel(=1), sequence_length, embedding_size]
3. conved：我们这了进行了一个二维卷积，input_channel为1，output_channel为3，filter_size为（2，embedding_size），相当于bi-gram。
   [batch_size, output_channel,sequence_len-1, 1]

> 卷积输出的高和宽的计算公式：
>
> $$height_{out} = \frac{height_{in} - height_{kernel} + 2 \times padding}{stride} + 1$$
>
> $$widthout = \frac{width_{in} - width_{kernel} + 2 \times padding}{stride} + 1$$

5. Flatten: [batch_size,ouput_channel]

```
model = TextCNN().to(device)
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# Training
for epoch in range(5000):
  for batch_x, batch_y in loader:
    batch_x, batch_y = batch_x.to(device), batch_y.to(device)
    pred = model(batch_x)
    loss = criterion(pred, batch_y)
    if (epoch + 1) % 1000 == 0:
        print('Epoch:', '%04d' % (epoch + 1), 'loss =', '{:.6f}'.format(loss))

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
# Test
test_text = 'i hate me'
tests = [[word2idx[n] for n in test_text.split()]]
test_batch = torch.LongTensor(tests).to(device)
# Predict
model = model.eval()
predict = model(test_batch).data.max(1, keepdim=True)[1]
if predict[0][0] == 0:
    print(test_text,"is Bad Mean...")
else:
    print(test_text,"is Good Mean!!")
```

用TextCNN进行IMDB电影评论情感分析

1. 数据预处理

- 设置种子SEED，保证结果可复现
- 利用torchtext构建数据集

```
import torch
from torchtext.legacy import data
from torchtext.legacy import datasets
import random
import numpy as np

SEED = 1234

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True

TEXT = data.Field(tokenize = 'spacy',
                  tokenizer_language = 'en_core_web_sm',
                  batch_first = True)
LABEL = data.LabelField(dtype = torch.float)

train_data, test_data = datasets.IMDB.splits(TEXT, LABEL)

train_data, valid_data = train_data.split(random_state = random.seed(SEED))
```

- 构建vocab，加载预训练词嵌入：

```
MAX_VOCAB_SIZE = 25_000

```

```
                unk_init = torch.Tensor.normal_)

LABEL.build_vocab(train_data)
```

- 创建迭代器：

```
BATCH_SIZE = 64

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

train_iterator, valid_iterator, test_iterator = data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = BATCH_SIZE,
    device = device)
```

2. 构建模型

- nn.Embedding中padding_idx：当Embedding是随机初始化的矩阵时，**会对padding_idx所在的行进行填0**。保证了padding行为的正确性。

> 维度变换
>
> 1. 输入：text [batct_size, senquence_len]
> 2. embedding和unsqueeze(1): [batch_size,1,sequence_len,embedding_dim]
> 3. conved:[batch_size, output_channel,senquence_len-filter_size[n]+1,1]
> 4. pooled:[batch_size,output,channel,1,1]
> 5. Flatten: 这里的flatten在第三步和第四步进行了squeeze操作[batch_size,output_channel]
> 6. concat: [batch_size, output_channel*len(filter_sizes]

```
class CNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, n_filters, filter_sizes, output_d
                 dropout, pad_idx):

        super().__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx = pad_

        self.convs = nn.ModuleList([
                                    nn.Conv2d(in_channels = 1,
                                              out_channels = n_filters,
                                              kernel_size = (fs, embedding_dim))
                                    for fs in filter_sizes
                                    ])

        self.fc = nn.Linear(len(filter_sizes) * n_filters, output_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, text):

        #text = [batch size, sent len]

        embedded = self.embedding(text)

        #embedded = [batch size, sent len, emb dim]

        embedded = embedded.unsqueeze(1)

        #embedded = [batch size, 1, sent len, emb dim]

        conved = [F.relu(conv(embedded)).squeeze(3) for conv in self.convs]

        #conved_n = [batch size, n_filters, sent len - filter_sizes[n] + 1]

                                                                    (2) for conv in conved]
```

```
        #pooled_n = [batch size, n_filters]

        cat = self.dropout(torch.cat(pooled, dim = 1))

        #cat = [batch size, n_filters * len(filter_sizes)]

        return self.fc(cat)
```

- 设置超参数，实例化模型

```
INPUT_DIM = len(TEXT.vocab)
EMBEDDING_DIM = 100
N_FILTERS = 100
FILTER_SIZES = [3,4,5]
OUTPUT_DIM = 1
DROPOUT = 0.5
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]

model = CNN(INPUT_DIM, EMBEDDING_DIM, N_FILTERS, FILTER_SIZES, OUTPUT_DIM, DROPOUT,
```

- 查看参数

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f'The model has {count_parameters(model):,} trainable parameters')
```

- 加载预训练词嵌入

```
pretrained_embeddings = TEXT.vocab.vectors

model.embedding.weight.data.copy_(pretrained_embeddings)
```

- 将unk 和 pad 初始权重归零。

```
UNK_IDX = TEXT.vocab.stoi[TEXT.unk_token]

model.embedding.weight.data[UNK_IDX] = torch.zeros(EMBEDDING_DIM)
model.embedding.weight.data[PAD_IDX] = torch.zeros(EMBEDDING_DIM)
```

3. 训练模型

- 初始化优化器、损失函数

```
import torch.optim as optim

optimizer = optim.Adam(model.parameters())

criterion = nn.BCEWithLogitsLoss()

model = model.to(device)
criterion = criterion.to(device)
```

- 计算精度的函数

```
def binary_accuracy(preds, y):
    """
    Returns accuracy per batch, i.e. if you get 8/10 right, this returns 0.8, NOT 8
    """

    #round predictions to the closest integer
```

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

- train

```
def train(model, iterator, optimizer, criterion):

    epoch_loss = 0
    epoch_acc = 0

    model.train()

    for batch in iterator:

        optimizer.zero_grad()

        predictions = model(batch.text).squeeze(1)

        loss = criterion(predictions, batch.label)

        acc = binary_accuracy(predictions, batch.label)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

- eval（**注意**：同样，由于使用的是dropout，我们必须记住使用`model.eval()`来确保在评估时能够关闭dropout。）

```
def evaluate(model, iterator, criterion):

    epoch_loss = 0
    epoch_acc = 0

    model.eval()

    with torch.no_grad():

        for batch in iterator:

            predictions = model(batch.text).squeeze(1)

            loss = criterion(predictions, batch.label)

            acc = binary_accuracy(predictions, batch.label)

            epoch_loss += loss.item()
            epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

```
import time

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

- training

```
N_EPOCHS = 5

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss, train_acc = train(model, train_iterator, optimizer, criterion)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut4-model.pt')

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
```

- test

```
model.load_state_dict(torch.load('tut4-model.pt'))

test_loss, test_acc = evaluate(model, test_iterator, criterion)

print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
```

4. 模型验证

```
import spacy
nlp = spacy.load('en_core_web_sm')

def predict_sentiment(model, sentence, min_len = 5):
    model.eval()
    tokenized = [tok.text for tok in nlp.tokenizer(sentence)]
    if len(tokenized) < min_len:
        tokenized += ['<pad>'] * (min_len - len(tokenized))
    indexed = [TEXT.vocab.stoi[t] for t in tokenized]
    tensor = torch.LongTensor(indexed).to(device)
    tensor = tensor.unsqueeze(0)
    prediction = torch.sigmoid(model(tensor))
    return prediction.item()
```

- 输入需要评测的句子

```
predict_sentiment(model, "This film is terrible")
predict_sentiment(model, "This film is great")
```

优化

- 在TextCNN的实践中，有很多地方可以优化：

> Filter尺寸：这个参数决定了抽取n-gram特征的长度，这个参数主要跟数据有关，平均长度在50以内的话，用10以下就可以了，否则可以长一些。在调参时可以先用一个尺寸grid search，找到一个最优尺寸，然后尝试最优尺寸和附近尺寸的组合

> Filter个数：这个参数会影响最终特征的维度，维度太大的话训练速度就会变慢。这里在100-600之间调参即可

> 正则化：指对CNN参数的正则化，可以使用dropout或L2，但能起的作用很小，可以试下小的dropout率(<0.5)，L2限制大一点

> Pooling方法：根据情况选择mean、max、k-max pooling，大部分时候max表现就很好，因为分类任务对细粒度语义的要求不高，只抓住最大特征就好了

> Embedding表：中文可以选择char或word级别的输入，也可以两种都用，会提升些效果。如果训练数据充足（10w+），也可以从头训练
> 蒸馏BERT的logits，利用领域内无监督数据

> 加深全连接：原论文只使用了一层全连接，而加到3、4层左右效果会更好

TextCNN是很适合中短文本场景的强baseline，但不太适合长文本，因为 卷积核 尺寸通常不会设很大，无法捕获长距离特征。同时max-pooling也存在局限，会丢掉一些有用特征。另外再仔细想的话，TextCNN和传统的n-gram词袋模型本质是一样的，它的好效果很大部分来自于词向量的引入，解决了词袋模型的稀疏性问题。

> 参考文献
> TextCNN 的 PyTorch 实现
> 深入TextCNN（一）详述CNN及TextCNN原理
> 深度学习文本分类模型综述+代码+技巧

NLP学习之使用pytorch搭建textCNN模型进行中文文本分类 持久决心的博客 2万+
最近花周末两天时间利用pytorch实现了TextCNN进行了中文文本分类，在此进行记录。 数据获取 中文数据是从https://githu…

使用pytorch进行IMDB情感分析 最新发布 weixin_43991446的博客 282
使用pytorch进行IMDB情感分析 建议：将代码整合到main()函数中。 1. 配置 1.1 设置cuda和随机种子 # 设置cuda device = …

chinese_text_cnn：TextCNN Pytorch实现中文文本分类情感分析 02-03
TextCNN Pytorch实现中文文本分类 论文 参考 依赖项 python3.5 pytorch == 1.0.0 torchtext == 0.3.1 jieba == 0.39 词向量…

pytorch搭建TextCNN与使用案例 呆萌的代Ma 99
TextCNN算法流程 整体流程是将词拼接在一起，一句话构成一个特征图 根据卷积核得到多个特征向量 每个特征向量全局池…

pytorch构建LSTM分类器用于IMDB情感分类 Lyric1的博客 468
本文基于pytorch构建LSTM情感分类分类器，在IMDB数据集上进行测试，涉及文本预处理、数据集加载、模型训练、保存…

情感分析-火炬：IMDb数据集上的火炬情感分析 02-04
使用PyTorch进行情感分析 存储库将引导您完成构建完整的情感分析模型的过程，该模型将能够预测给定评论的极性（无论…

pytorch实现的TextCNN（Dataset, DataLoader的使用） SinGaln的博客 576
主要是Dataset, DataLoader的使用 （1）数据处理，生成Batch和向量化词表 import torch import numpy as np from tqdm i…

pytorch实现IMDB数据集情感分类（全连接层的网络、LSTM） Delusional的博客 2235
任务描述 使用Pytorch相关API，设计两种网络结构，一种网络结构中只有全连接层，一种使用文本处理中最为常用的LSTM…

pytorch使用套路（以RNN分析imdb文本情感为例） JJY的博客 556
这里写目录标题1.导入包2.预定义参数3.加载数据4.构建网络5.定义训练函数6.定义测试函数7.开始训练 在使用过keras和pyt…