1.3-宋阳-测试词向量小结

任务：

1. 封装以下两种词向量，方便以后调用：

（1） one normal embedding, finetune, dense, cpu gpu both；

（2） the other one, const embedding, static, sparse, only in cpu because the size may be huge。

2. 在情感分类任务上进行测试

情感分类测试

测试1：normal embedding, random init, finetune, dense, both cpu and gpu

（1）实现

因为用的是一套的随机种子，所以weight放在不同位置随机，会得到不同结果，所以设置了以下几组实验进行对比。

Method 1:

nn.Embedding内部调用一次reset_parameters()进行初始化，然后外部调用rand_init()又一次初始化，所以用了一整套的随机种子的前面2部分，起作用的是第2次用的随机种子部分。

```
def rand_init(self, input_emb):
    print("Use uniform to initialize the embedding")
    input_emb.weight.data.uniform_(-0.01, 0.01)
    # print(input_emb.padding_idx)
    if input_emb.padding_idx is not None:
        input_emb.weight.data[input_emb.padding_idx].fill_(0)
self.word_embeddings = nn.Embedding(args.embed_num, args.embedding_dim, padding_idx=args.padID)
self.rand_init(self.word_embeddings)
```

Method 2:

封装的类继承了nn.Embedding，所以一共只随机了一次，用的是一整套随机种子的第1部分，这里就与Method 1中不同了。

```
class Embedding(nn.Embedding):
    def reset_parameters(self):
        print("Use uniform to initialize the embedding")
        self.weight.data.uniform_(-0.01, 0.01)
        # print(self.padding_idx)
        if self.padding_idx is not None:
            self.weight.data[self.padding_idx].fill_(0)
self.word_embedding = Embedding.Embedding(args.embed_num, args.embedding_dim, padding_idx=args.padID)
```

Method 3:

封装的类继承了nn.Embedding，调用一次reset_parameters()进行初始化，然后外部又调用了一次reset_parameters()，所以一共用了一整套的随机种子的前面2部分，起作用的是第2次用的随机种子部分，理论上与Method 1中相同了，但是结果不对，所以有了后面的Method 4。

```
self.word_embedding = Embedding.Embedding(args.embed_num, args.embedding_dim, padding_idx=args.padID)
self.word_embedding.reset_parameters()
```

Method 4:

封装的类里面的初始化是正态分布，所以为了2次的初始化完全一样，所以对Method 2进行改进。

```
class Embedding(nn.Embedding):
    def reset_parameters(self):
        print("Use uniform to initialize the embedding")
        self.weight.data.normal_(0, 1)
        if self.padding_idx is not None:
            self.weight.data[self.padding_idx].fill_(0)

        self.weight.data.uniform_(-0.01, 0.01)
        # print(self.padding_idx)
        if self.padding_idx is not None:
            self.weight.data[self.padding_idx].fill_(0)
self.word_embedding = Embedding.Embedding(args.embed_num, args.embedding_dim, padding_idx=args.padID)
```

（2）测试结果

model	acc
Method 1	81.27%
Method 2	82.70%
Method 3	81.44%
Method 4	81.27%

注：特别感谢师兄的启发和指导，收获很多，谢谢师兄。

测试2：const embedding, static, sparse, only in cpu

（1）实现

Method 1: sparse=True

```
self.word_embeddings = nn.Embedding(args.embed_num, args.embedding_dim, padding_idx=args.padID, sparse=True)
if args.use_pretrained_emb:
    pretrained_weight = np.array(loader.vector_loader(args.word_dict, set_padding=True))
    self.word_embeddings.weight.data.copy_(torch.from_numpy(pretrained_weight))
    self.word_embeddings.weight.requires_grad = False
```

Method 2: sparse=False

```
self.word_embeddings = nn.Embedding(args.embed_num, args.embedding_dim, padding_idx=args.padID, sparse=False)
if args.use_pretrained_emb:
    pretrained_weight = np.array(loader.vector_loader(args.word_dict, set_padding=True))
    self.word_embeddings.weight.data.copy_(torch.from_numpy(pretrained_weight))
    self.word_embeddings.weight.requires_grad = False
```

Method 3:

```
class ConstEmbedding(nn.Module):
    def __init__(self, pretrained_embedding, padding_idx=0):
        super(ConstEmbedding, self).__init__()
        self.vocab_size = pretrained_embedding.size(0)
        self.embedding_size = pretrained_embedding.size(1)
        self.embedding = nn.Embedding(self.vocab_size, self.embedding_size, padding_idx=padding_idx, sparse=True)
        self.embedding.weight = nn.Parameter(pretrained_embedding, requires_grad=False)
if args.use_pretrained_emb:
    pretrained_weight = loader.vector_loader(args.word_dict, set_padding=True)
    pretrained_weight = torch.FloatTensor(pretrained_weight)
    self.word_embedding_static = Embedding.ConstEmbedding(pretrained_weight, padding_idx=args.padID)
```

（2）测试结果

Model	Acc	Time
Method 1	84.95%	31.2350597381591s
Method 2	84.95%	32.8334894180297s
Method 3	84.95%	32.0523877143859s

注：编码时注意到model文件中尽量不出现cuda的设置，在main文件、train文件和类封装文件内部实现好cpu和gpu的转换，所以模型的forward部分的调用的一样的。