N-Gram Language Modeling

什么是 N-Gram 模型：

我们知道一篇文章，每句话都有很多单词组成，对于一句话，这些单词的组成顺序也是很重要的，如果我们想要知道一篇文章中，我们是否可以给出几个词然后预测这些词后面的一个单词，比如 I lived in France for 10 years, I can speak _____ .'那么我们想要做的就是预测最后这个词是 French。这就是 N-Gram 模型，模型的公式如下:

$$P(W_i|W_{i-1},W_{i-2},\cdots,W_{i-n+1})$$

其实就是个条件概率，即我们给定想要预测的单词的前面几个单词，然后最大化我们想要预测的这个单词的概率

Code 说明：

数据预处理部分:

这里我们设置 CONTEXT_SIZE = 2，表示我们想要由前面的 CONTEXT_SIZE 个单词来预测单词，这里设置为 2，并且设置 EMBEDDING_DIM = 100，表示 Embedding 后单词的词向量是 100 维。

```
trigram = [((test_sentence[i], test_sentence[i+1]),test_sentence[i+2]) for i in range(len(test_sentence) - 2)]
```

我们需要将单词三分组，每组的前两个单词作为传入的数据，最后一个作为预测的结果

Note:我们需要将每个单词进行编码，也就是用数字来表示每个单词，这样才能够传入 word embedding 得到词向量

定义模型：

```
class NgramModel(nn.Module):
    def __init__(self, vocb_size, context_size, n_dim):
        super(NgramModel,self).__init__()
        self.n_word = vocb_size
        self.embedding = nn.Embedding(self.n_word, n_dim)
        self.linear1 = nn.Linear(n_dim * context_size, 128)
        self.linear2 = nn.Linear(128, self.n_word)
    def forward(self, x):
        emb = self.embedding(x)
        emb = emb.view(1, -1)
        out = self.linear1(emb)
        out = F.relu(out)
        out = self.linear2(out)
        log_prob = F.log_softmax(out)
        return log_prob
print(len(vocb))
ngrammodel = NgramModel(len(vocb), CONTEXT_SIZE, EMBEDDING_DIM)
critrion = nn.NLLLoss()
optimizer = optim.SGD(ngrammodel.parameters(), lr = learning_rate)
```

这个模型需要传入的参数是所有的单词数，预测单词需要的前面单词数，即 CONTEXT_SIZE，词向量的维度。

然后在向前传播中，首先传入单词得到词向量，比如在该模型中传入两个词，得到的词向量是(2, 100)，然后将词向量展开成(1, 200)，然后传入一个线性模型，经过 relu 激活函数再传入一个线性模型，输出的维数是单词总数，可以看成一个分类问题，要最大化预测单词的概率，最后经过一个 log softmax 激活函数，然后定义好模型，loss 以及优化器

训练模型

```
for epoch in range(num_epoches):
    print('epoch {}'.format(epoch + 1))
    print('*'*100)
    running_loss = 0.0
    for data in trigram:
        word, label = data
        word = torch.LongTensor([word_to_idx[i] for i in word])
        label = torch.LongTensor([word_to_idx[label]])

        #forward
        out = ngrammodel(word)
        loss = critrion(out, label)
        running_loss += loss.item()

        # backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print("Loss:{:.6f}".format(running_loss/len(word_to_idx)))
```

模型预测

```
word, label = trigram[1]
word = torch.LongTensor([word_to_idx[i] for i in word])
out = ngrammodel(word)
_, predict_label = torch.max(out, 1)
predict_word = idx_to_word[predict_label.item()]
print('real word is {}, predict word is {}'.format(label, predict_word))
```

以上我们介绍了如何通过最简单的单边 N-Gram 模型预测单词，还有一种复杂一点的 N-Gram 模型通过双边的单词来预测中间的单词，这种模型有个专门的名字，叫 Continuous Bag-of-Words model (CBOW)，具体的内容差别不大，就不再细讲了