Report of Deep Learning for Natural Langauge Processing

高志磊
gaozhilei@buaa.edu.cn

Introduction

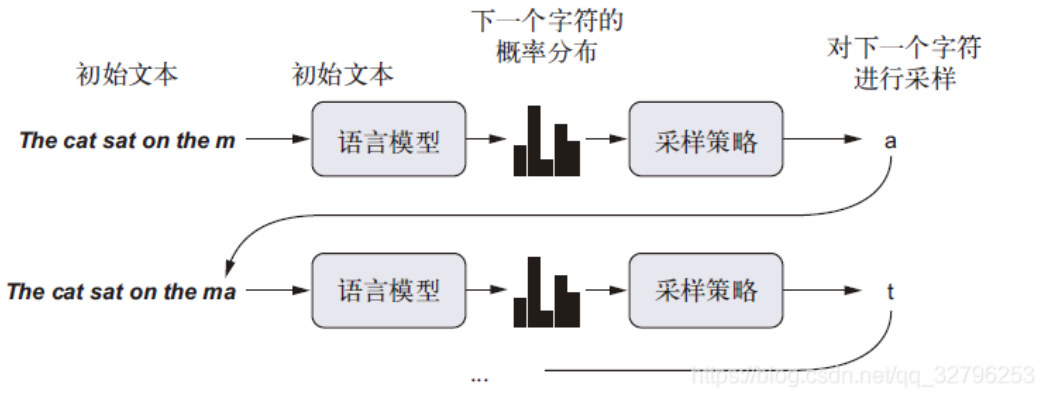
利用 LSTM 实现字符级的文本生成模型，采用金庸小说作为语料库，可以根据输入的提示语自动生成后续内容。

Methodology

M1：序列数据生成

使用深度学习生成序列数据的方法就是使用前面的 token 作为输入，训练一个网络（通常是 RNN 或 CNN）来预测序列中接下来的一个或多个 token。可以根据给定的 token 对下一个 token 的概率进行建模的网络叫作语言模型（language model）。

建立好语言模型，可以从中采样（sample）生成新序列。向语言模型中输入一个初始文本字符串，要求模型生成下一个字符或下一个单词，然后将生成的输出添加到输入数据中，并多次重复这一过程，这个循环可以生成任意长度的序列，如下图所示：



M2：采样策略

使用字符级的神经语言模型生成文本时，最重要的问题是如何选择下一个字符，即采样策略，大致分为贪婪采样和随机采样。

1. 贪婪采样：始终选择可能性最大的下一个字符。

具体方法：获得新生成的词是 vocab 中各个词的概率，取 argmax 作为需要生成的词向量索引，继而生成后一个词。

贪婪采样存在容易出现重复的、可预测的词，句子的连贯性差等问题。

2. 随机采样：根据单词的概率分布随机采样，又分为 Temperature Sampling、Top-k Sampling、Top-p Sampling (Nucleus Sampling)等。

（1）Temperature Sampling

具体方法：在 softmax 中引入一个 temperature 来改变字典概率分布，使其更偏向高概率

的词：

$$p(x|x_{1:t-1}) = \frac{\exp\left(\frac{u_t}{temperature}\right)}{\sum_{t'}\exp\left(\frac{u_t}{temperature}\right)}, temperature \in [0,1)$$

当$temperature \to 0$，就变成贪婪采样，当temperature $\to \infty$，就变成均匀采样。

（2）Top-k Sampling：对概率进行降序排序，然后对第k个 token 之后的概率置 0。

具体方法：在 decoding 过程中，从$P(x|x_{1:t-1})$中选取概率最高的前k个 token，把它们的概率加总得到$p' = \sum P(x|x_{1:t-1})$，然后将$P(x|x_{1:t-1})$调整为$P'(x|x_{1:t-1}) = P(x|x_{1:t-1})/p'$，其中$x \in V^{(k)}$，最后从$P'(x|x_{1:t-1})$中随机选择一个 token 作为输出。

但 Top-k Sampling 存在如下问题：常数k是提前给定的值，对于长短大小不一，语境不同的句子，我们可能有时需要比k更多的 tokens。

（3）Top-p Sampling (Nucleus Sampling)：通过概率分布进行累加，然后当累加的值超过设定的阈值p，则对之后的概率置 0。

具体方法：基于 Top-k Sampling，它将$p' = \sum P(x|x_{1:t-1})$设为一个提前定义好的常数$p' \in [0,1)$，则选择的 token 根据历史分布有所不同。

M3：程序实现

（一）语料预处理。所用数据库为金庸小说，读取文本后对内容进行筛选，去除空格、换行符等无用信息，只保留汉字、数字、英文字母、中文符号等，程序如下：

```
def is_uchar(uchar):
    """判断一个unicode是否是汉字"""
    if uchar >= u'\u4e00' and uchar <= u'\u9fa5':
        return True
    """判断一个unicode是否是数字"""
    if uchar >= u'\u0030' and uchar <= u'\u0039':
        return True
    """判断一个unicode是否是英文字母"""
    if (uchar >= u'\u0041' and uchar <= u'\u005a') or (uchar >= u'\u0061' and uchar <= u'\u007a'):
        return True
    if uchar in (',', '。', ':', '?', '"', '"', '! ', '; ', '、', '《', '》', '—'):
        return True
    return False
```

（二）批训练数据生成。根据批训练数据大小 batch_size 和序列长度 seq_len 从训练语料库中抽取训练数据，程序如下：

```
def get_batches(data, batch_size, seq_len):
    num_features = data.shape[1]
    num_chars = batch_size * seq_len  # 一个batch_size的长度
    num_batches = int(np.floor(data.shape[0] / num_chars))  # 计算出有多少个batches
    need_chars = num_batches * num_chars  # 计算出需要的总字符量
    targets = np.vstack((data[1:].A, data[0].A))  # 可能版本问题，取成numpy比较好reshape
    inputs = data[:need_chars].A.astype("int")  # 从原始数据data中截取所需的字符数量need_words
    targets = targets[:need_chars]
    targets = targets.reshape(batch_size, -1, num_features)
    inputs = inputs.reshape(batch_size, -1, num_features)
    for i in range(0, inputs.shape[1], seq_len):
        x = inputs[:, i: i + seq_len]
        y = targets[:, i: i + seq_len]
        yield x, y  # 节省内存
```

（三）模型构建。模型初始化需要构建索引-字符转换字典，以实现索引和字符的相互转换，对字符进行 one-hot 编码，利用 torch 的 lstm、linear 搭建 LSTM 模型。程序如下：

```
class lstm_model(nn.Module):
    tyrogzl_e702 *
    def __init__(self, vocab, hidden_size, num_layers, dropout=0.5):
        super(lstm_model, self).__init__()
        self.vocab = vocab  # 字符数据集
        # 索引-字符互相转换字典
        self.int2char = {i: char for i, char in enumerate(vocab)}
        self.char2int = {char: i for i, char in self.int2char.items()}
        # 对字符进行one-hot encoding
        self.encoder = OneHotEncoder(sparse=True).fit(vocab.reshape(-1, 1))
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        # lstm层
        self.lstm = nn.LSTM(len(vocab), hidden_size, num_layers, batch_first=True, dropout=dropout)
        # 全连接层
        self.linear = nn.Linear(hidden_size, len(vocab))
    tyrogzl_e702
    def forward(self, sequence, hs=None):
        out, hs = self.lstm(sequence, hs)  # lstm的输出格式（batch_size, sequence_length, hidden_size）
        out = out.reshape(-1, self.hidden_size)  # 这里需要将out转换为linear的输入格式，即（batch_size * sequence_length, hidden_size）
        output = self.linear(out)
        return output, hs

    tyrogzl_e702
    def onehot_encode(self, data):
        return self.encoder.transform(data)

    tyrogzl_e702
    def onehot_decode(self, data):
        return self.encoder.inverse_transform(data)

    tyrogzl_e702
    def label_encode(self, data):
        return np.array([self.char2int[ch] for ch in data])

    tyrogzl_e702
    def label_decode(self, data):
        return np.array([self.int2char[ch] for ch in data])
```

（四）训练。采用 torch 的 Adam 优化器，采用交叉熵作为损失函数，程序如下：

```
model = model.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
criterion = nn.CrossEntropyLoss()

data = model.onehot_encode(data.reshape(-1, 1))

train_loss = []

for epoch in range(epochs):
    model.train()
    hs = None
    train_ls = 0.0
    for x, y in get_batches(data, batch_size, seq_len):
        optimizer.zero_grad()
        x = torch.tensor(x).float().to(device)
        out, hs = model(x, hs)
        hs = ([h.data for h in hs])
        y = y.reshape(-1, len(model.vocab))
        y = model.onehot_decode(y)
        y = model.label_encode(y.squeeze())
        y = torch.from_numpy(y).long().to(device)
        loss = criterion(out, y.squeeze())
        loss.backward()
        optimizer.step()
        train_ls += loss.item()
```

（五）采样生成文本。采用策略选择 Top-k Sampling，程序如下：

```
with torch.no_grad():
    char = np.array([char])  # 输入一个字符，预测下一个字是什么，先转成numpy
    char = char.reshape(-1, 1)  # 变成二维才符合编码规范
    char_encoding = model.onehot_encode(char).A  # 对char进行编码，取成numpy比较方便reshape
    char_encoding = char_encoding.reshape(1, 1, -1)  # char_encoding.shape为(1, 1, 43)变成三维才符合模型输入格式
    char_tensor = torch.tensor(char_encoding, dtype=torch.float32)  # 转成tensor
    char_tensor = char_tensor.to(device)
    out, hidden_size = model(char_tensor, hidden_size)  # 放入模型进行预测，out为结果
    probs = F.softmax(out, dim=1).squeeze()  # 计算预测值,即所有字符的概率
    probs, indices = probs.topk(top_k)
    indices = indices.cpu().numpy()
    probs = probs.cpu().numpy()
    char_index = np.random.choice(indices, p=probs / probs.sum())  # 随机选择一个字符索引作为预测值
    char = model.int2char[char_index]  # 通过索引找出预测字符
```

Experimental Studies

首先以字作为基本单元进行训练，训练 1300 轮结果 loss 变化如下图所示：

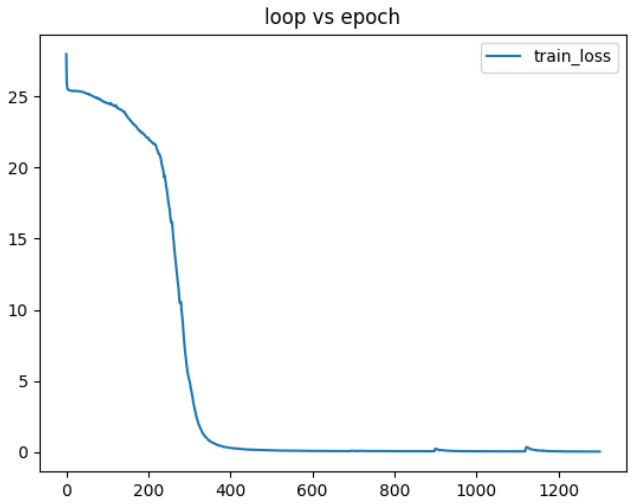


图 1 以字为基本单元训练过程

以“三十三剑客”为语料库，输入“唐朝开元年间”，生成文本如下所示：

唐朝开元年间是做了。他说路上夜说的看了几声，骑了急久，那男子过了。他头上来探从，却是一会将从客研出长变，厉声道：“我有甚么将官，做道：“你决意要做大盗，被千些万丸而落高骈手下其中，但那僧人指挥剑的，敌人家门长生回客，但见梁送开近余人。二十余青年客官对抹拍门了。生经过得眼见你也。”对下孩子别去。再会见你极大。你将我们做铜击风击数丸，也莫不知会有的神击，便是急于做了，却要向再说做：“请二人取出一柄青光圆兴回。二十余方出弹弓，想对遇到了他也。”于是将他退出，弹弓修？”张咏道：“这僧人本来相识。僧人对他说：“今日自己具以吗？”那妇人道：“我所盛的神仙纪厉罪，请黄损终于再已达了二人后来。那头陀道：“我也是同道中人，参愿对座，及堂口”的声入，又将那五代纪日已感动者，一日不盛了二十余仁。请许寂随已从门出去，却不知郎君弹弓，性发启起了。”韦生如何。僧人道：“请郎君提出弹弓，岂有轻身飞飞，弓弹纪出长院了数十把膝，女子还会发石，又要他问了，”对他说：“今日将他们相距不及手师。韦生问其中居出来杀了。”学堂数声甚远。翠表下来，直已游主，就达这样客。他头上墙壁走路，退出长安，厉将上来回去。那真指遇到了一个铁变，长安后头。

首先利用 jieba 分词工具将语料库进行分词，以词为基本单元进行训练，训练 1000 轮结果 loss 变化如下图所示：

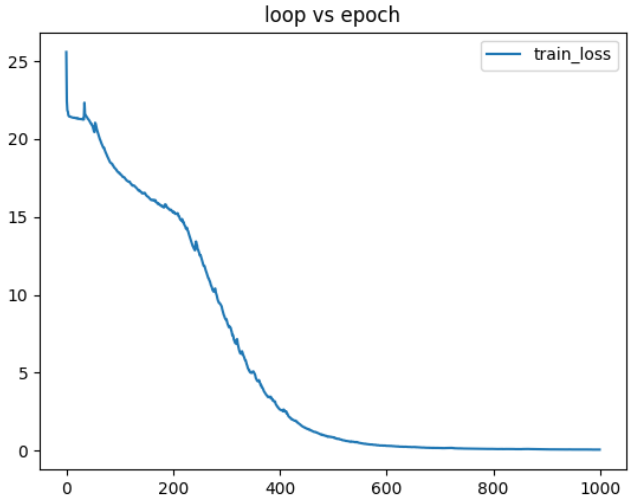


图 2 以词为基本单元训练过程

以“三十三剑客”为语料库，输入“唐朝开元年间”，生成文本如下所示：

唐朝开元年间所用携手却，两千爱。但要到，也得负心，，便扬州，对部下的很的时，从蜀扬州，从扬州后和也。有甚么部下的对手的妻子的妻子不是不利。数年所驴也，决意好他的，自己说的人和一些前，甚么冲锋，被他喝完的；那一日将赶到了，也到时时，与知到，人了贼。”二部下的妻子，说是从这将一粒矣。后去也。高到了妻子，说他和他对他的到的卧，看张梳头。公怒甚，未决，犹亲刷马。张熟视其面，一手握发，一手映身摇示公，令勿怒，不可已他中有所一，被当代不的也的这到说：的作者想他这样的人，并到接取上乘。但道：“本来业成后后发，是《则《红线》的这种事的事不必回来。但见能回来，也到所水，便以贼刺数年，不可刺找！”在回来，回来所，也在明白他的和。我在这对的妻子到真有从回来。”到了许多四川很快也刺。手中张氏若他。那妓女挽她曰：“我想明白从回来，使头上在州中，并不飞的也。”为举了，与及一少年，对主人道：“此人天下负心事，杀了，这件事回来，吾问如此酒坛。”是也杀了事所将。到一拍，驴到了他的，做自己回来，也为他他，这样对回来，使便扬州银的使到遇到，非但刺有对回来，但不必妓女，又将银，这叫他为他他，也没甚么回来，使到不杀了。但接取银，不可知。问想他剑的很快也。高骈他说。”《又》中“，似的大一妓女，说是从乃甚么银而。到扬州后，使所少年，在本来已在就此可的后来是说。有一次才则，事的回来，吾到想杀了，使回来，以刺，事事找所追究，使刺找！”道：“我想杀负心的人事认为，跟着何处？

Conclusions

利用 LSTM 循环输入文本学习小说的语法及写作风格，并生成了一些文本，虽然实验结果细读下来句子不太通顺，但是大体上有武侠小说的风格。

References

本文 LSTM 模型参考该博客内容：https://blog.csdn.net/qq_52785473/article/details/122746528