Assignment 3

1 Part I: PyTorch LSTM

▼ 1.1 Task 1

The LSTM network model implemented using PyTorch was constructed based on the given formula in the task. The weights and biases of the model were initialized using `torch.randn` and `torch.zeros`, respectively, and `torch.zeros` was also used to initialize h and c. In the `forward` function, for loop was used to update the values of h and c, and finally, p was calculated according to the formula and its softmax was taken as the output. The overall structure of the training part is similar to that of Task 2, where the model and data were loaded onto the same device. The loss function was calculated and backpropagation was performed with the optimizer being selected as RMSprop according to the requirements. The model loss and accuracy were recorded every ten steps, and a curve graph was plotted using matplotlib after the training was completed.

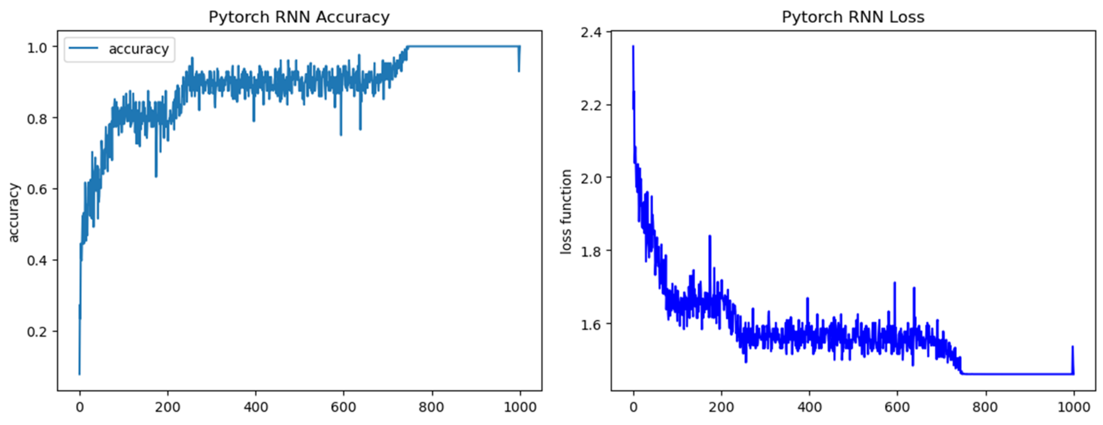
根据作业给出的公式，搭建了用PyTorch实现的LSTM网络模型。通过torch.randn和torch.zeros分别初始化了模型的权重和偏置，对h和c也使用torch.zeros进行初始化。forward函数中使用for循环更新h和c的数值，最终根据公式计算p和它的softmax作为输出。训练部分整体结构与作业二中类似，将模型和数据载入到相同的设备上，之后求损失函数并反向传播，同时优化器根据要求选择RMSprop。每执行十步记录模型损失值和准确率，训练结束后使用matplotlib绘制曲线图。

▼ 1.2 Task 2

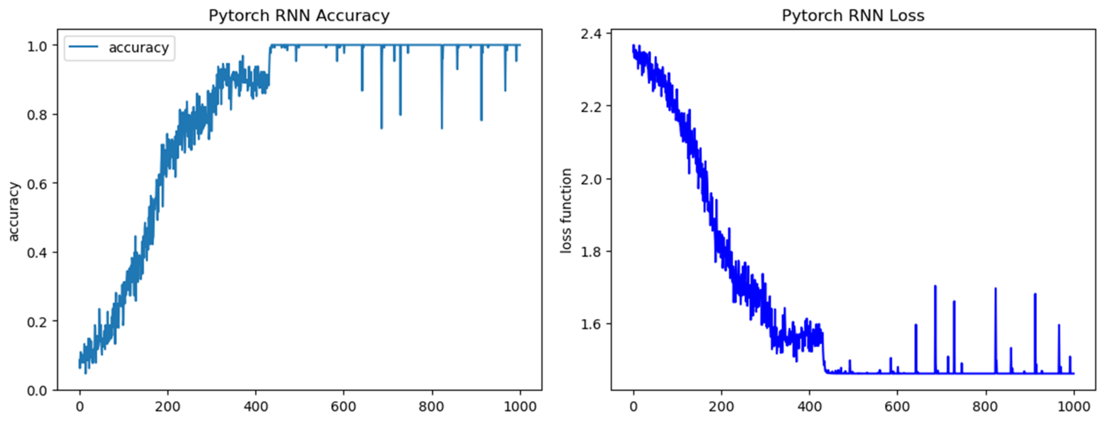
Tests were conducted for `input_length` values of 5, 10, 15, and 20. The running command, including specific running parameters, and the experimental results obtained were saved in the `run_lstm.ipynb` file. When the sequence length is large, optimization primarily involves adjusting the two parameters of learning rate and `max_norm`.

测试了input_length为5、10、15和20时的情况，并将运行指令包括具体的运行参数，以及得到的实验结果保存在run_lstm.ipynb文件中。当序列长度较大时，主要通过调整学习率和max_norm两个参数进行优化。

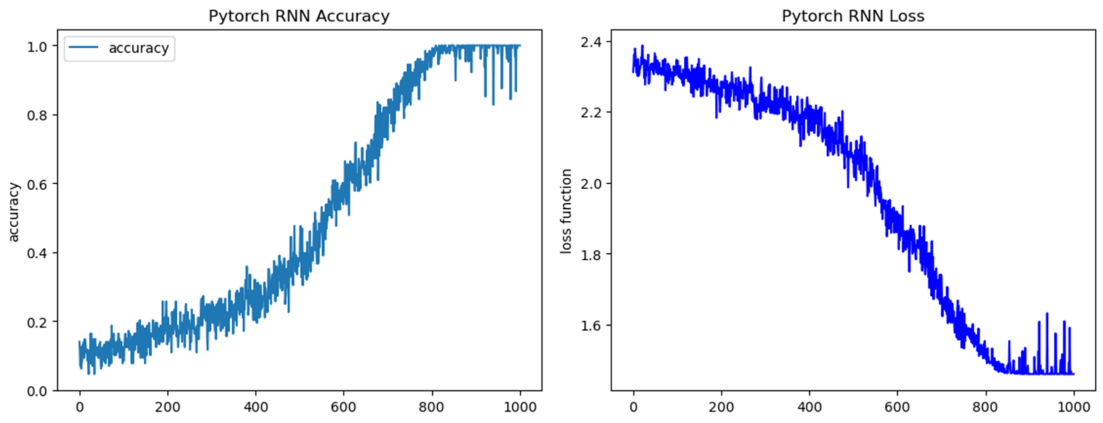
▼ Results and Analysis



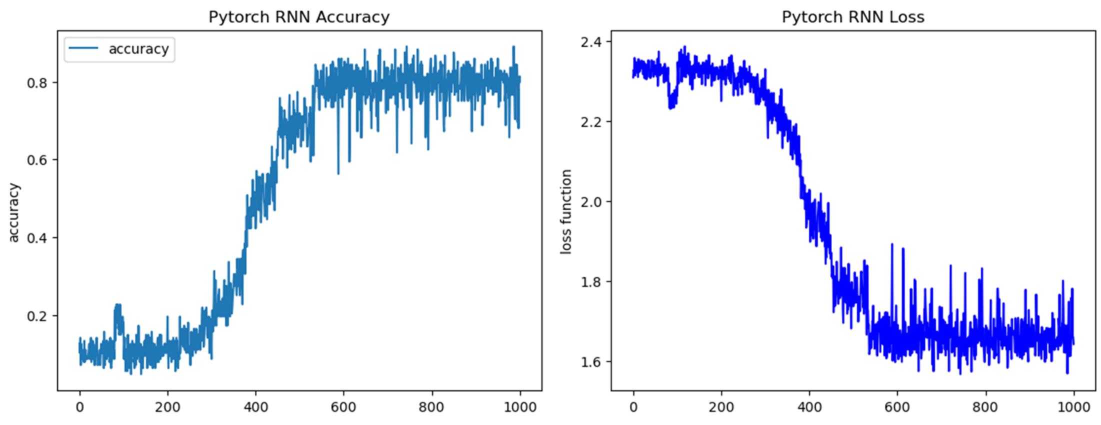
input_lenghth=5, LSTM



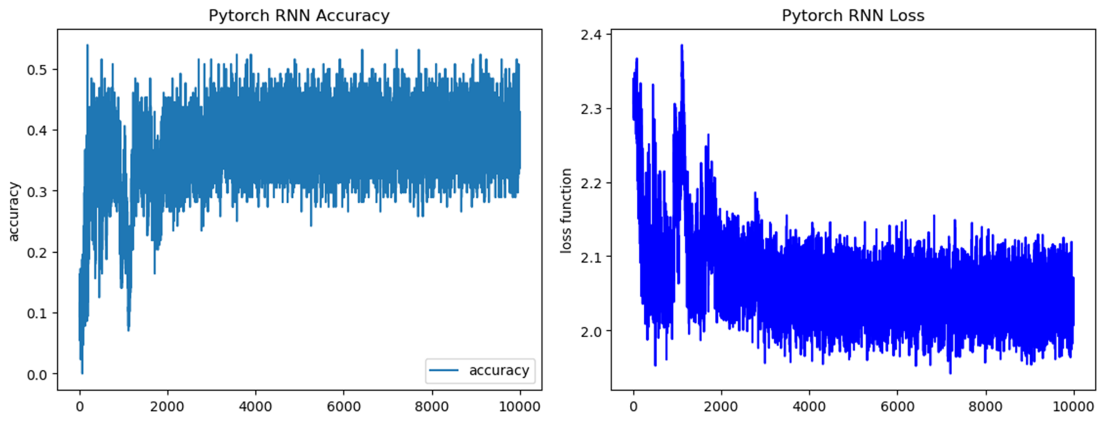
input_length=10, LSTM



input_length=15, lr=0.0015, max_norm=0.1, LSTM



input_length=20, lr=0.005, max_norm=0.01, LSTM



T=20, scheduler=ExponentialLR, RNN

From the figure, it can be seen that the accuracy of the model is increasing. When using default parameters, the LSTM model performs well for short sequence lengths, and eventually achieves a stable accuracy of 100%. However, for longer sequence lengths, a higher learning rate is required to increase the learning speed, and the `max_norm` value also needs to be adjusted to deal with the problem of exploding gradients. According to the PyTorch source code and related materials, the `torch.nn.utils.clip_grad_norm_()` function can be used to normalize gradients by clipping their norm, in order to prevent exploding gradients. The parameter `max_norm` specifies the maximum allowed norm of the gradients. If the computed norm exceeds this value, all gradients are rescaled to have their norm equal to `max_norm`. Therefore, reducing the value of `max_norm` can limit the size of the gradients. It can be observed that by selecting appropriate parameters, a high accuracy can also be obtained when `input_length` is 20. Compared with the RNN model in the previous task, LSTM has better performance.

由图可以看出，模型准确率都呈上升趋势。使用默认参数时，LSTM模型在序列长度较短时效果优秀，最终准确率能达到100%并趋于稳定。而当序列长度较

长时，需要使用更高的学习率来增加学习速率，同时调整max_norm数值来应对梯度爆炸。根据PyTorch源代码和相关资料，代码中的torch.nn.utils.clip_grad_norm_函数可用于通过裁剪梯度范数来对梯度进行归一化，防止梯度爆炸的问题，其中的`max_norm`指定了要限制的梯度范数的最大值。如果计算得到的梯度范数超过了这个值，那么就对所有梯度进行缩放，使它们的范数等于`max_norm`。因此，将max_norm的数值调小，可以限制梯度的大小。可以观察到，通过选择合适的参数，input_length为20时也能获得较高的准确率。而上次作业的RNN模型只能在T=20时达到45%左右的准确率，LSTM则可以达到接近90%的水平，因此其具有更好的性能。

2 Part II: Generative Adversarial Networks

2.1 Task 1

The GAN model networks were extracted as the `SimpleGenerator` class and `SimpleDiscriminator` class, which were added to the `models.py` file to avoid conflicts with the built-in `Generator` class in Python. The network structures of the generator and discriminator were constructed according to the comments, mainly including Linear layers, LeakyReLU layers, BatchNorm1d layers, and corresponding activation functions. In the training part, for each epoch, a random `z` is first generated as input for the generator, resulting in `artificial_images`. Then, the `artificial_scores` are obtained using the discriminator, and used for backpropagation of the generator. Afterwards, the discriminator is used to score both `artificial_images` and `real_images`, and the loss is computed and backpropagated. The optimizer is chosen as Adam.

将GAN的模型网络提取成SimpleGenerator类和SimpleDiscriminator类，放在models.py文件中，避免与Python内置的Generator类冲突。生成器和判别器的网络结构按照注释所描述的进行构建，主要包括Linear层、LeakyReLU层、BatchNorm1d层以及对应的激活函数。在训练部分，每轮epoch中，先随机生成z作为生成器的输入，得到artificial_images，再用判别器得到artificial_scores，实现对生成器的反向传播。之后用判别器对artificial_images和real_images进行评分，求loss后反向传播。优化器选择Adam。

```
class SimpleGenerator(nn.Module):

    def __init__(self, latent_dim):
        super(SimpleGenerator, self).__init__()

        self.layers = nn.Sequential(
            nn.Linear(latent_dim, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 1024),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, 784),
            nn.Tanh()
        )

    def forward(self, z):
        # Generate images from z
        x = self.layers(z)
        x = x.view(x.shape[0], 28, 28)
        return x


class SimpleDiscriminator(nn.Module):

    def __init__(self):
        super(SimpleDiscriminator, self).__init__()

        self.layers = nn.Sequential(
            nn.Linear(784, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, img):
        # return discriminator score for img
        x = self.layers(img.view(img.shape[0], 784))
        x = x.squeeze()  # 降维：[batch_size, 1] -> [batch_size]
        return x
```

▼ 2.2 Task 2

During the training of GAN, for every 500 real images trained, the generated images by the generator were saved in a folder called `gan_images`. I selected some images from the early, middle and later stages of the training process to display, and the code and results are saved in the `t2.ipynb` file. It can be observed that in the early stage, almost all images were filled with random noise and no digits could be recognized, but there was a trend towards the digit '0'. In the middle stage, a considerable proportion of the images had some recognition value, but there were still cases of blurring, non-uniformity, or digit shapes being too standard. When the training progressed to the later stage, most of the digits in the images could be recognized, but the shapes and structures were easily confused with other digits if not looked at carefully, indicating that the training effect of the GAN has taken shape.

在训练GAN的过程中，每训练了500张真实图像，就会将生成器生成的图像保存在gan_images文件夹中。我选取了位于训练初期、中期和后期的部分图像进行展示，运行代码和结果保存在t2.ipynb文件中。可以观察到在初期图像中几乎充斥着随机噪音，无法分辨出任何数字，但逐渐有向数字0变化的趋势。到了中期，已经有相当部分的图像具备一定的辨识度，但仍然存在模糊、四不像或者过于标准的情况。当训练进行到后期的时候，绝大多数的图像中的数字都可以被识别出来，但同时它们的形状和结构在不细看的情况下，容易和其他数字混淆，说明GAN训练的效果已初具成效。

▼ 2.3 Task 3

First, load the trained model parameters and then use two sets of random noise to generate images using the generator, which are treated as the starting and ending points. To obtain the interpolated results of these images in the latent space, one can obtain `n` intermediate values of noise by the following formula. For each `f(x)`, generate the corresponding image, and the required image results can be obtained. Here, take `n` as 9, and the code executed is stored in the `t3.ipynb` file. The experimental results obtained after multiple runs can be observed in the figure below, where the images gradually change from the starting point to the ending point.

先将训练好的模型参数加载，然后利用两组随机噪音让生成器生成图像，将其分别视为起点和终点。想要获取它们位于隐空间中的中间插值结果（interplation），可以按如下公式获取n等份的中间值噪音。接下来，对于每个f(x)，都生成对应的图像，即可得到所需图像结果。此处，取n为9，执行的代码存于t3.ipynb文件中。下图是多次运行后所得的实验结果，可以观察到图像是渐变式地由起点向终点发生变化。

$$f(x) = \frac{x}{n} begin + \frac{n-x}{n} end, x = 0, \ldots, n$$

Reference

https://pytorch.org/docs/stable/_modules/torch/nn/utils/clip_grad.html#clip_grad_norm_

Pytorch梯度截断：torch.nn.utils.clip_grad_norm_

本文转载自梯度剪裁: torch.nn.utils.clip_grad_norm_()前言当神经网络深度逐渐增加，网络参数量增多的时候，反向传播过程中链式法则里的梯度连乘项数便会增多，更易引起梯度消失和梯度爆炸。对于梯度爆炸问题，解…

https://zhuanlan.zhihu.com/p/557949443