

第五节课习题作业



本讲习题共包含两部分：分别是算法理论题和代码实践题。同学们可以选择其一项完成并提交。（当然也欢迎大家将两部分全部完成，这样能够加深对课程的理解）

- 算法理论题提交方式：

发送邮件至 opendilab@pjlabor.org.cn

请同学们严格按照下方格式命名邮箱主题/标题：

【PPO × Family】+ 学生名 + vol.5 (第几节课) + 作业提交日期

示例：【PPO × Family】+ 喵小DI + vol.5 +20230223

- 代码实践题提交方式：

PPO × Family 官方GitHub 上发起 Pull Request

- 地址： [PP0xFamily/chapter5_time/hw_submission](https://github.com/PP0xFamily/chapter5_time/hw_submission)
- PR示例： <https://github.com/opendilab/PP0xFamily/pull/5>
- 命名规范：

hw_submission(学生名称): add hw5 (第几节课) +作业提交日期

示例：hw_submission(nyz): add hw5_20230104

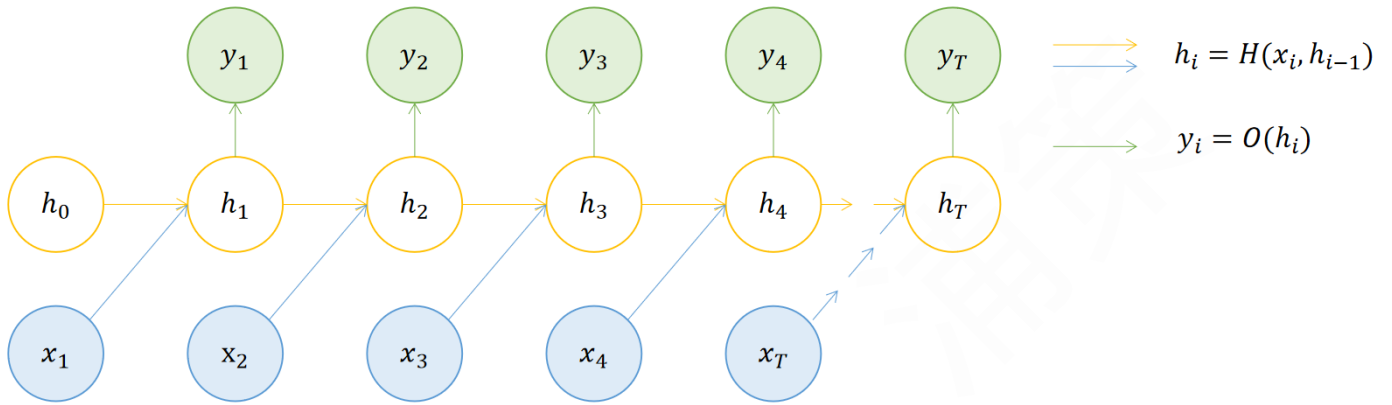
提交 截止时间为 2023.4.11 23:59 (GMT +8)，逾期作业将不会计入证书考量。

如果其他问题请添加官方课程小助手微信（vx: OpenDILab），备注「课程」，小助手将邀请您加入官方课程微信交流群；或发送邮件至 opendilab@pjlabor.org.cn

算法理论题

题目一（RNN梯度计算分析）

序列模型（Sequential model）在机器学习领域有非常广泛的应用，尤其是在时序信息处理，自然语言处理等领域。下图是一个由 Recurrent Neural Network（RNN）组成的序列到序列（Seq2Seq）模型的示意图，它的输入 x_t 是一个序列， $x_t \in \mathbb{R}^{d_x}$ ，输出也是一个序列， $o_t \in \mathbb{R}^{d_y}$ ，序列最大长度为 T ：



每一个时刻输入的信息 x_t ，都会与上一个时刻的隐藏状态 h_{t-1} ，一起通过转移函数 H ，生成当前时刻的隐藏状态：

$$h_t = H(x_t, h_{t-1} \mid \theta)$$
$$H : \mathbb{R}^{d_h \times d_x} \rightarrow \mathbb{R}^{d_h}$$

而每个时刻的隐藏状态 h_t ，通过一个输出函数 $o_t = O(h_t \mid \theta)$ ，输出当前时刻隐藏状态 h_t 的对应输出值 o_t 。从上图中，以及转移函数的定义式中，我们可以发现，各个时刻之间的信息是循环展开和传递的，在每一个时刻隐藏状态都会被当前时刻的输入所更新，并将这种变化的信息通过输出值传递出来。通过定义一些优化目标函数（比如各类损失函数），可以用于度量模型的输出 $o_{1:T}$ 和真实数据 $y_{1:T}$ 之间的差别。具体来说，将单个时刻的损失函数记为 $l(o_t, y_t)$ ，则所有时刻的平均损失为：

$$L = \frac{1}{T} \sum_{i=0}^T l(o_i(x_{1:i}), y_i)$$

在第 $t = i$ 时刻的损失关于模型参数的梯度，会通过时刻 $t = i - 1$ 的隐藏状态向过去传递。

问题1：请推导对于一组序列数据， $(x_i, y_i) \in \mathbb{R}^{(d_x+d_y) \times T}$ ，当 $d_x = 1$ ， $d_y = 1$ ， $T = 3$ 时这个 RNN 模型对其参数 θ 的梯度表达式。

(提示：可以将损失函数的梯度按照以下式子展开，并以此为基础续写和归纳至 $T = N$)

$$\begin{aligned}
\frac{\partial L}{\partial \theta} &= \frac{1}{T} \sum_{i=0}^T \frac{\partial l(o_i(x_{1:i}), y_i)}{\partial \theta} \\
&= \frac{1}{T} \sum_{i=0}^T \left[\frac{\partial l(o_i(x_{1:i}), y_i)}{\partial o_i} \frac{\partial O(h_i)}{\partial h_i} \frac{\partial h_i}{\partial \theta} \right] \\
&= \frac{1}{T} \sum_{i=0}^T \left[\frac{\partial l(o_i(x_{1:i}), y_i)}{\partial o_i} \frac{\partial O(h_i)}{\partial h_i} \left(\frac{\partial H(x_i, h_{i-1})}{\partial \theta} + \frac{\partial H(x_i, h_{i-1})}{\partial h_{i-1}} \frac{\partial h_{i-1}}{\partial \theta} \right) \right] \\
&= \frac{1}{T} \sum_{i=0}^T \left[\frac{\partial l(o_i(x_{1:i}), y_i)}{\partial o_i} \frac{\partial O(h_i)}{\partial h_i} \left(\frac{\partial H(x_i, h_{i-1})}{\partial \theta} + \frac{\partial H(x_i, h_{i-1})}{\partial h_{i-1}} \left(\frac{\partial H(x_{i-1}, h_{i-2})}{\partial \theta} + \frac{\partial H(x_{i-1}, h_{i-2})}{\partial h_{i-2}} \frac{\partial h_{i-2}}{\partial \theta} \right) \right) \right]
\end{aligned}$$

问题2：在实践中，简单的 RNN 模型比较容易因为出现梯度消失 [1] 或梯度爆炸 [2] 的现象而使得训练变得困难。请通过分析问题1中得到的 RNN 模型的梯度表达式，讨论梯度消失现象的原因。（此外，可以回顾课程内容，了解处理这类梯度问题的一些方法。）

题目二 (Belief MDP)

马尔可夫决策过程 (Markov decision process, MDP) 是一个可获得完全信息的决策过程，智能体在每个时刻可以观测到其真实的状态， $s_t \in S$ ，在经历行动 $a_t \in A$ 之后，可以观测到真实的状态转移

$$T(s_t, a_t, s_{t+1}) = P(s_{t+1} | s_t, a_t) : S \times A \times S \rightarrow \mathbb{R}^+,$$

并可以在每个时刻获得奖励的具体数值

$$r_t(s_t, a_t) : S \times A \rightarrow \mathbb{R}.$$

但对于那些由于各种客观原因的限制，**无法获取完全信息**的场景和其中的系统，假定其真实状态的动力学依然由一个真实不变的 MDP 所决定，只是智能体不能直接观察底层真实状态，仅能观察到其中一部分的信息 $o_t \in O$ （这里的记号 O 代表观测，即 *Observation*）。那么，则可以称这个决策过程为一个部分观测马尔可夫决策过程 (Partially observable Markov decision process, POMDP)。



相比于马尔可夫决策过程，部分观测马尔可夫决策过程可以额外引入观测信息 o_t 和真实状态 s_t 之间的函数关系，称为观测函数：

$$\Pi(o_{t+1}, s_t, a_t) = P(o_{t+1} | s_t, a_t) : S \times A \times O \rightarrow \mathbb{R}^+.$$

虽然智能体无法获知自己的真实状态，但是可以建立一种对真实状态的某种估计，称之为信念 (Belief State)：

$$b(s_t) = P(s_t) : S \rightarrow \mathbb{R}^+$$

也就是说，对于某一个时刻，智能体对当前时刻的真实状态的具体数值有一个实际的分布作为其估计。

通过更多次与环境的交互，可以利用交互带来的信息，结合贝叶斯定理逐步更新对当前真实状态信念的估计，一般将更新后的信念记为 $b'(s_t)$ 。比如对于离散的场景下，其利用贝叶斯定理的更新形式为：

$$b'(s_{t+1}) = P(s_{t+1}|o_{t+1}, a_t, b(s_t)) = \frac{P(o_{t+1}|s_{t+1}, a_t, b(s_t))P(s_{t+1}|a_t, b(s_t))}{\sum_{s_{t+1} \in S} [P(o_{t+1}|s_{t+1}, a_t, b(s_t))P(s_{t+1}|a_t, b(s_t))]}$$

上式中，由于真实状态具有观测信息的完备统计量，故有：

$$P(o_{t+1}|s_{t+1}, a_t, b(s_t)) = P(o_{t+1}|s_{t+1})$$

而对于下一时刻状态对于上一时刻信念和动作的条件先验，可以使用转移概率展开求解：

$$P(s_{t+1}|a_t, b(s_t)) = \sum_{s_t \in S} P(s_{t+1}|s_t, a_t)b(s_t)$$

这样一来，对于每一个时刻，我们都可以对当前的奖励做一个基于信念估计的函数计算：

$$R(a_t, b(s_t)) = \sum_{s_t \in S} r(a_t, s_t)b(s_t)$$

这样我们就让一个 POMDP 问题出现了一些熟悉的元素，比如我们用对于状态的信念，来替代 MDP 中的状态。用奖励函数来替代 MDP 中的奖励。然后我们就可以使用一些适用于 MDP 的算法来近似解决 POMDP 问题。

为了让大家熟悉 POMDP 中的信念更新，本次作业题将涉及一个相关入门小例子：

胡图图同学对 OpenDILab 开设的 PPO × Family 课程非常感兴趣，想要观看直播和回放，不过因为忘记了登录电子设备的密码，遇到了一点麻烦。由于多次尝试错误，系统需要每隔100分钟才能再次尝试（尝试出错的惩罚为 -100 ，成功打开设备的奖励为 $+10$ ）。他大概记得密码应该是 A 和 B 中的一个，都有可能。不过由于直播马上就要开始了，他现在比较苦恼。假设胡图图同学仔细冷静思考1分钟（思考的成本为 -1 ），可以大概确定个八九不离十，想起来具体是 A 还是 B 。假如标记想起来的密码为 o_t ，真实的密码为 s_t ，思考的动作为 $a_t = a_0$ ，输入密码的动作为 $a_t = a_A$ 或 $a_t = a_B$ ，即在这里我们可以认为：

$$P(o_{t+1} = A|s_t = A, a_t = a_0) = 0.85, \quad P(o_{t+1} = B|s_t = B, a_t = a_0) = 0.85$$

问题1：假如刚开始胡图图对真实密码是 A 还是 B 的信念**相同**，那么假如他冷静思考之后，信念如何更新？请根据贝叶斯定理，计算思考后的真实状态的信念的分布。

问题2：假如刚开始胡图图对真实密码是 A 还是 B 的信念**不相同**，为了让奖励期望最大，胡图图需要拥有多少程度的信念，才应该进行密码输入？

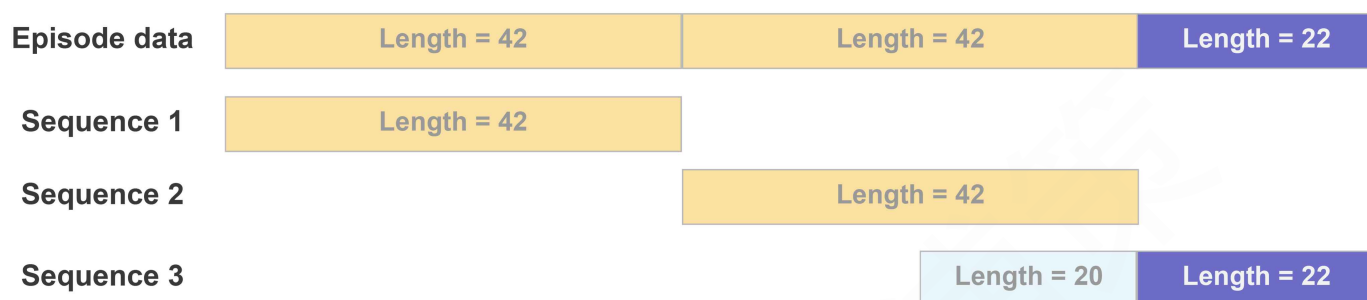
代码实践题

题目一（变长序列处理）

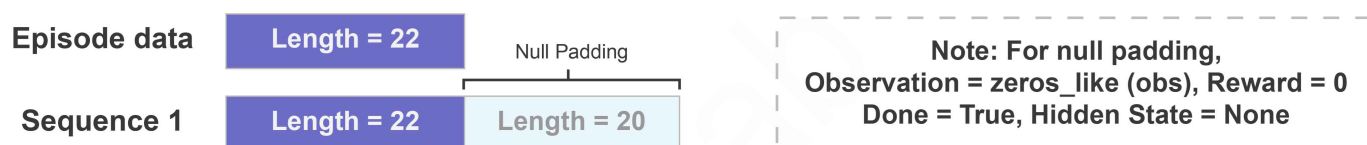
请根据课程第五讲 LSTM 部分讲到的变长序列处理方法，填补实现完成下方示例代码中的 `pack_data` 函数，并运行相应的测试函数，通过所有断言（Assertion）语句。

变长序列处理方法示意图如下：

Case 1: Episode length > Sequence length



Case 2: Episode length ≤ Sequence length



完整代码如下（也可从官网[链接](#)下载）：

```
1 """
2 Long Short Term Memory (LSTM) <link https://ieeexplore.ieee.org/abstract/documen
3 This document mainly includes:
4 - Pytorch implementation for LSTM.
5 - An example to test LSTM.
6 For beginners, you can refer to <link https://zhuanlan.zhihu.com/p/32085405 link
7 """
8 from typing import Optional, Union, Tuple, List, Dict
9 import math
10 import torch
11 import torch.nn as nn
12 from ding.torch_utils import build_normalization
13
14
15 class LSTM(nn.Module):
16     """
17     **Overview:**
18     Implementation of LSTM cell with layer norm.
19     """
```

```

20
21     def __init__(
22         self,
23         input_size: int,
24         hidden_size: int,
25         num_layers: int,
26         norm_type: Optional[str] = 'LN',
27         dropout: float = 0.
28     ) -> None:
29         # Initialize arguments.
30         super(LSTM, self).__init__()
31         self.input_size = input_size
32         self.hidden_size = hidden_size
33         self.num_layers = num_layers
34         # Initialize normalization functions.
35         norm_func = build_normalization(norm_type)
36         self.norm = nn.ModuleList([norm_func(hidden_size * 4) for _ in range(2 *
37         # Initialize LSTM parameters.
38         self.wx = nn.ParameterList()
39         self.wh = nn.ParameterList()
40         dims = [input_size] + [hidden_size] * num_layers
41         for l in range(num_layers):
42             self.wx.append(nn.Parameter(torch.zeros(dims[l], dims[l + 1] * 4)))
43             self.wh.append(nn.Parameter(torch.zeros(hidden_size, hidden_size * 4
44         self.bias = nn.Parameter(torch.zeros(num_layers, hidden_size * 4))
45         # Initialize the Dropout Layer.
46         self.use_dropout = dropout > 0.
47         if self.use_dropout:
48             self.dropout = nn.Dropout(dropout)
49         self._init()
50
51     # Dealing with different types of input and return preprocessed prev_state.
52     def _before_forward(self, inputs: torch.Tensor, prev_state: Union[None, List
53         seq_len, batch_size = inputs.shape[:2]
54         # If prev_state is None, it indicates that this is the beginning of a se
55         if prev_state is None:
56             zeros = torch.zeros(self.num_layers, batch_size, self.hidden_size, d
57             prev_state = (zeros, zeros)
58         # If prev_state is not None, then preprocess it into one batch.
59         else:
60             assert len(prev_state) == batch_size
61             state = [[v for v in prev.values()] for prev in prev_state]
62             state = list(zip(*state))
63             prev_state = [torch.cat(t, dim=1) for t in state]
64
65     return prev_state
66

```

```

67     def _init(self):
68         # Initialize parameters. Each parameter is initialized using a uniform a
69         gain = math.sqrt(1. / self.hidden_size)
70         for l in range(self.num_layers):
71             torch.nn.init.uniform_(self.wx[l], -gain, gain)
72             torch.nn.init.uniform_(self.wh[l], -gain, gain)
73             if self.bias is not None:
74                 torch.nn.init.uniform_(self.bias[l], -gain, gain)
75
76     def forward(
77         self,
78         inputs: torch.Tensor,
79         prev_state: torch.Tensor,
80     ) -> Tuple[torch.Tensor, Union[torch.Tensor, List]]:
81         # The shape of input is: [sequence length, batch size, input size]
82         seq_len, batch_size = inputs.shape[:2]
83         prev_state = self._before_forward(inputs, prev_state)
84
85         H, C = prev_state
86         x = inputs
87         next_state = []
88         for l in range(self.num_layers):
89             h, c = H[l], C[l]
90             new_x = []
91             for s in range(seq_len):
92                 # Calculate  $z_t, z^i, z^f, z^o$  simultaneously.
93                 gate = self.norm[l * 2](torch.matmul(x[s], self.wx[l])
94                                         ) + self.norm[l * 2 + 1](torch.matmul(h,
95                                     if self.bias is not None:
96                                         gate += self.bias[l]
97                 gate = list(torch.chunk(gate, 4, dim=1))
98                 i, f, o, z = gate
99                 #  $z^i = \sigma(Wx^i x^t + Wh^i h^{t-1})$ 
100                 i = torch.sigmoid(i)
101                 #  $z^f = \sigma(Wx^f x^t + Wh^f h^{t-1})$ 
102                 f = torch.sigmoid(f)
103                 #  $z^o = \sigma(Wx^o x^t + Wh^o h^{t-1})$ 
104                 o = torch.sigmoid(o)
105                 #  $z = \tanh(Wx x^t + Wh h^{t-1})$ 
106                 z = torch.tanh(z)
107                 #  $c^t = z^f \odot c^{t-1} + z^i \odot z$ 
108                 c = f * c + i * z
109                 #  $h^t = z^o \odot \tanh(c^t)$ 
110                 h = o * torch.tanh(c)
111                 new_x.append(h)
112             next_state.append((h, c))
113             x = torch.stack(new_x, dim=0)

```

```

114         # Dropout layer.
115         if self.use_dropout and l != self.num_layers - 1:
116             x = self.dropout(x)
117         next_state = [torch.stack(t, dim=0) for t in zip(*next_state)]
118         # Return list type, split the next_state .
119         h, c = next_state
120         batch_size = h.shape[1]
121         # Split h with shape [num_layers, batch_size, hidden_size] to a list wit
122         next_state = [torch.chunk(h, batch_size, dim=1), torch.chunk(c, batch_si
123         next_state = list(zip(*next_state))
124         next_state = [{k: v for k, v in zip(['h', 'c'], item)} for item in next_
125         return x, next_state
126
127
128 def pack_data(data: List[torch.Tensor], traj_len: int) -> Tuple[torch.Tensor, to
129     """
130     Overview:
131         You need to pack variable-length data to regular tensor, return tensor a
132         If len(data_i) < traj_len, use `null_padding`,
133         else split the whole sequences info different trajectories.
134     Returns:
135         - tensor (:obj:`torch.Tensor`): dtype (torch.float32), shape (traj_len,
136         - mask (:obj:`torch.Tensor`): dtype (torch.float32), shape (traj_len, B)
137     """
138     raise NotImplementedError
139
140
141 def test_lstm():
142     seq_len_list = [32, 49, 24, 78, 45]
143     traj_len = 32
144     N = 10
145     hidden_size = 32
146     num_layers = 2
147
148     variable_len_data = [torch.rand(s, N) for s in seq_len_list]
149     input_, mask = pack_data(variable_len_data, traj_len)
150     assert isinstance(input_, torch.Tensor), type(input_)
151     batch_size = input_.shape[1]
152     assert batch_size == 9, "packed data must have 9 trajectories"
153     lstm = LSTM(N, hidden_size=hidden_size, num_layers=num_layers, norm_type='LN
154
155     prev_state = None
156     for s in range(traj_len):
157         input_step = input_[s:s + 1]
158         output, prev_state = lstm(input_step, prev_state)
159
160     assert output.shape == (1, batch_size, hidden_size)

```



```
161     assert len(prev_state) == batch_size
162     assert prev_state[0]['h'].shape == (num_layers, 1, hidden_size)
163     loss = (output * mask.unsqueeze(-1)).mean()
164     loss.backward()
165     for _, m in lstm.named_parameters():
166         assert isinstance(m.grad, torch.Tensor)
167     print('finished')
168
169
170 if __name__ == '__main__':
171     test_lstm()
```

题目二（应用实践）

在课程第五讲（解密稀疏奖励空间）几个应用中任选一个

- Pong（使用叠帧和 PPO + LSTM）
- Memory Len（使用 PPO + GTrXL）

根据课程组给出的[示例代码](#)，训练得到相应的智能体。最终提交需要上传相关训练代码、日志截图或最终所得的智能体效果视频（replay），具体样式可以参考第五讲的[示例 ISSUE](#)。

计算资源共享指南

为了方便同学们完成代码题，探索更多强化学习的奥秘，我们提供了免费的计算资源（[AutoDL 平台](#)），具体的使用方法我们以第二节课的代码说明如下：

1. **登陆 AutoDL 平台：** <https://www.autodl.com/>
2. **账号信息：** 课程官方共享账号
 - UserName: 18016297479
 - Password: Opendilab1234567
3. **算力选择：** 点击左上角算力市场（如下图所示），并选择合适的算力资源。

AutoDL

算力市场

共享数据

算法社区

帮助文档

解决方案

面向AIGC的解决方案

弹性部署震撼上线

了解详情

计费方式: 按量计费 包日 包周 包月

选择地区: 北京A区 芜湖区 内蒙A区 北京C区 佛山区 毕业季A区 南京新手区 特惠/泉州A区 如何选择GPU?

GPU型号: ☐ 全部 ☐ RTX 3090 (1/984) ☐ V100-SXM2-32GB (0/80) ☐ RTX 3080 (0/632) ☐ RTX A4000 (0/962) ☒ RTX 2080 Ti (0/488) ☐ RTX 3070 (0/16)


GPU数量: 1 2 3 4 5 6 7 8

RTX 2080 Ti 北京A区 / 253机 可租用至: 2023-05-01

空闲GPU数量: 0 / 8卡 CPU: 12 核/GPU, Xeon(R) Platinum 8255C 内存: 43 GB/GPU
显存: 11 GB 系统盘: 25 GB 数据盘: 免费 50 GB SSD, 可扩容 4810 GB 支持最高CUDA版本: 11.7
浮点算力: 单精 13.45 TFLOPS / 半精 53.8 Tensor TFLOPS

3. 建议使用 网盘支持

¥0.88/时 ¥0.93/时 9.5折 会员最低享9.5折 ¥0.88/时 已租光



温馨提示:

由于课程的 demo 需要算力较小, 经过测试, 建议大家使用一张 2080Ti 即可顺利完成所有任务。

为了大家都可以方便地使用, 希望同学们可以珍惜计算资源, 不要浪费, 或选择更高配置。

4. 创建实例：在最下方的镜像处点击“我的镜像”，下拉框选择“di-engine”。这个镜像是助教老师为方便大家进行尝试，提前配置好的相关安装依赖。如果大家想要重新进行配置，可以选择合适版本的基础镜像，并在此基础上自行安装相关的依赖。

镜像:

基础镜像

算法镜像

我的镜像

如何保存我的镜像?

di-engine

创建完成后仍然可以更换其他镜像

5. 测试代码：如果配置无误，在点开生成的机器之后，在 terminal 中输入：

```
1 xvfb-run --auto-servernum --server-num=1 --server-args="-screen 1 1024x768x24" p
```

这是一段运行第二节课火箭回收（rocket landing）环境的简单代码，执行半分钟后如果**正常退出**，就证明整个流程无误。

其他示例镜像

上述步骤介绍了如何使用第二节课环境镜像进行基础的代码运行。除此之外，我们还针对不同节课提供了不同的镜像。这些镜像都安装了新版的 DI-engine，而主要的区别在于支持的环境不同，具体的对应关系如下表所示。如果大家有额外的需要，也可以在已有镜像的基础上自行安装。

针对的课程样例	镜像名称	支持的环境
---------	------	-------

第二节课	di-engine	lunarlander, rocket, gym_pybullet_drones, gym_hybrid
第三节课	ding_ch3	bipedalwalker, evogym, mario, di_sheep, procgen

注意事项:

因为官方共享账号，所有参与课程的同学均可以使用，因此希望大家节约资源，不要租用过贵的计算资源。

跑完代码之后，**一定要及时关机，避免持续扣费。**

尽量不要用公共资源做无关课程的事情、更不要自行更改密码，相关行为一经发现将被禁止使用。

更多关于 AutoDL 平台的使用问题，大家可以参考[文档](#)或是在微信群中提问。

参考文献

[1] https://en.wikipedia.org/wiki/Vanishing_gradient_problem

[2] <https://deeptai.org/machine-learning-glossary-and-terms/exploding-gradient-problem>