关于强化学习的探索

写在最前面的话

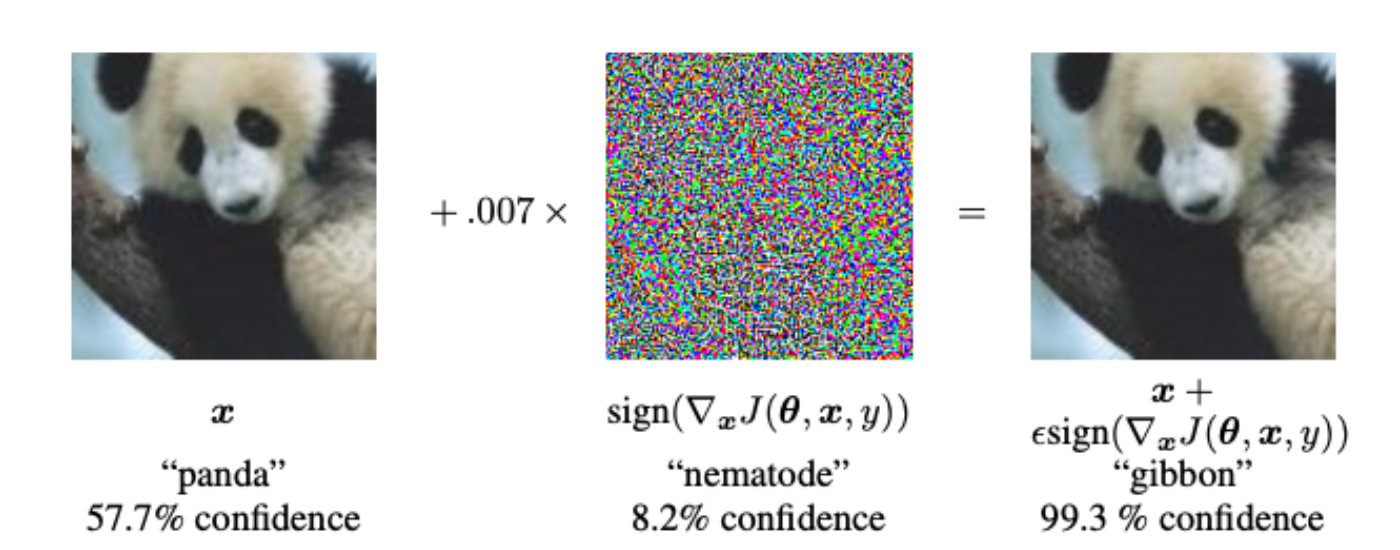
> 他山之石，可以攻玉。

基于上述原则，我们进行了以下探索。

一、背景介绍

在CIKM2020的对抗算法工作中，有任务如下：

> 对500*500图片生成一定的扰动，使目标模型失效。



类似如上，从熊猫的图像开始，攻击者向原始图像添加了较小的扰动，这导致模型以更高的置信度将该图像标记为长臂猿。

困难的是，任务对扰动的生成有一定的限制条件，如总面积等。

二、强化学习

（1）概念简介

- 强化学习（Reinforcement Learning, RL）又称再励学习、评价学习或增强学习，是机器学习的范式和方法论之一，用于描述和解决智能体在与环境的交互过程中通过学习策略以达成回报最大化或实现特定目标的问题。
- 强化学习理论受到行为主义心理学启发，侧重在线学习并试图在探索-利用间保持平衡。不同于监督学习和非监督学习，强化学习不要求预先给定任何数据，而是通过接收环境对动作的奖励（反馈）获得学习信息并更新模型参数。

（2）问题转化

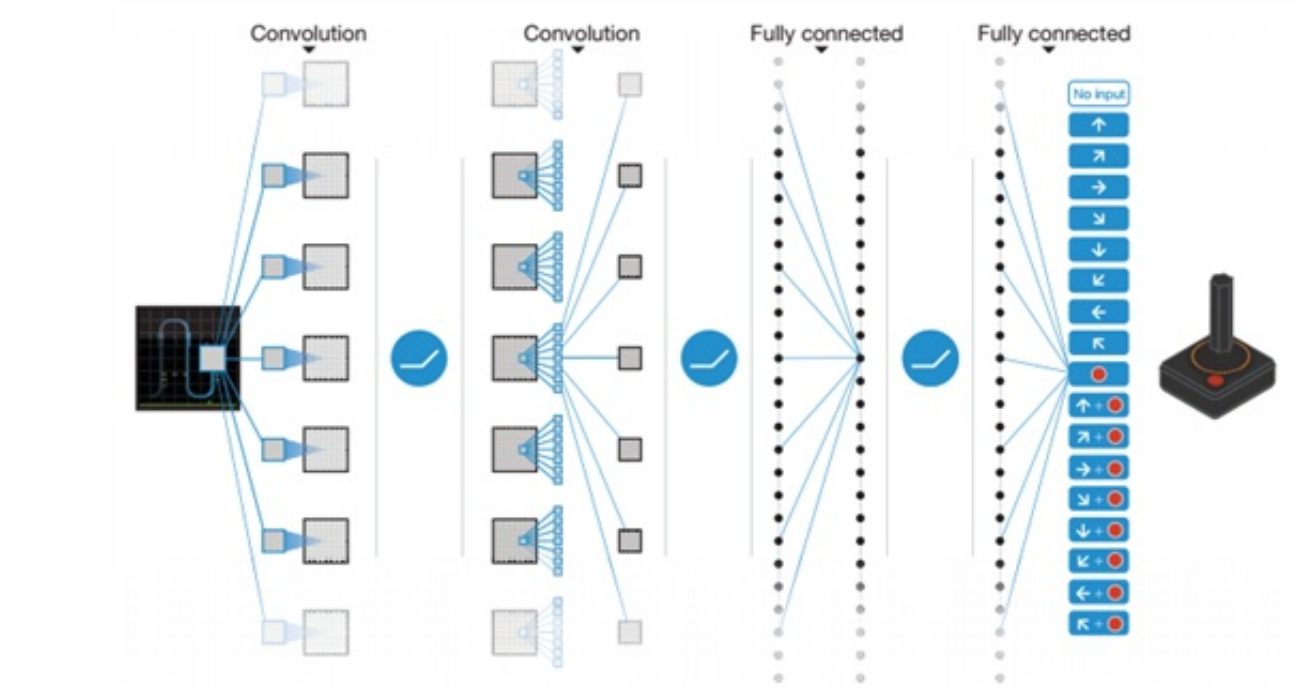
在这里，我们把扰动图生成变成像素点级的一个游戏。

即现在有一个500*500围棋棋盘，请随机在格子中下棋子，系统将返回对应奖励或惩罚。

据此，引入强化学习及其**策略梯度算法(Policy Gradient)**，并构建神经网络模型。

简单来说，神经网络的输入是原始的状态信息，优化即在该状态下执行动作的回报，即Q函数，输出是该状态下执行动作的概率。训练完成之后，神经网络逼近的是最优Q函数。

```
Q(S, a)
# 其中S为状态，a为动作
```



这样，我们就能知道处于S状态下怎么样的动作a是最优的。

三、过程细节

如果了解过强化学习的朋友会知道，几乎所有的示例代码都是基于gym实现某简单游戏，如flappybird。

以下我们将脱离gym游戏框架，直接上码：

（1）定义参数

```
N = 500
T = N ** 2
base = np.array([0 for _ in range(T)])
ACTION_DIM = T, T
```

（2）定义模型

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(8, (3, 3), activation='relu', input_shape=(N, N, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(ACTION_DIM, activation="softmax"),
])
model.compile(loss='categorical_crossentropy',
optimizer=tf.keras.optimizers.Adam(0.01))
print(model.summary())
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 498, 498, 8)       80
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 249, 249, 8)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 247, 247, 16)      1168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 123, 123, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 121, 121, 16)      2320
_________________________________________________________________
flatten (Flatten)            (None, 234256)            0
_________________________________________________________________
dense (Dense)                (None, 64)                14992448
_________________________________________________________________
dense_1 (Dense)              (None, 250000)            16250000
=================================================================
Total params: 31,246,016
Trainable params: 31,246,016
Non-trainable params: 0
```

（3）动作选择

```
def choose_action(s):
    prob = model.predict(np.array([s.reshape((N, N, 1))]))[0]
    return np.random.choice(len(prob), p=prob)
```

根据模型返回概率生成动作。

（4）模型迭代

```
def train(records):
    s_batch = np.array([record[0] for record in records])
    a_batch = np.array(
      [[1 if record[1] == i else 0 for i in range(ACTION_DIM)] for record in records]
    )
    prob_batch = model.predict(s_batch) * a_batch
    r_batch = np.array([record[2]*10000 for record in records])
    model.fit(s_batch, prob_batch, sample_weight=r_batch, verbose=0)
```

根据每次结果迭代训练模型。

解释一下

```
model.fit(s_batch, prob_batch, sample_weight=r_batch, verbose=0)
```

在这里通过sample_weight给loss加一个权重，进而改变**损失函数(loss function)**，使神经网络朝累加期望大的方向优化。

（5）奖励惩罚

最重要也是最困难的一步。需要定义出合适的奖励和惩罚函数，并给出限制条件。

也就是说游戏中，每画好一个点，都应该有奖励分数+10或者惩罚分数-10，并当所下棋子过多结束本轮游戏。

```
def sfun(arx):
   arr = cv.imread(f"{pdata0}/{idata}")
   _arr = arr.copy()
   for iarx in np.where(arx == 1)[0]:
      y_arx = iarx//N
      x_arx = iarx%N
      _arr[y_arx-M:y_arx+M, x_arx-M:x_arx+M] = datac
   cv.imwrite(f"{pdatat}/_{idata}", _arr)

   # .....省略 model0, modelt目标模型计算

   bb_score = [
     0 if np.isnan((_s0-_st)/_s0) or np.isinf((_s0-_st)/_s0) else (_s0-_st)/_s0
     for _s0, _st in zip(model0, modelt)
   ]
   rrr = np.sum(_bb_score) * _score # _score理解为限制条件分数

   return total_area_rate > 0.02, rrr
```

（6）开始游戏

```
episodes = 100 # 回合数
for i in range(episodes):
   s = base
   replay_records = []
   while True:
      a = choose_action(s)

      next_s = s.reshape(T).copy()
      next_s[a] = 1

      done, r = sfun(next_s)
      replay_records.append((s.reshape((N, N, 1)), a, r))

      s = next_s
```

```
        # 回合结束
        if done:
            print('episode:', i, 'training')
            train(replay_records)
            print('episode:', i, 'trainend')
            break
```

（7）结果展示

```
episode: 26 training
episode: 26 trainend
episode: 26 score:-0.000000 max:0.402377
0.000000, 0.000000, 0.000000, 0.000000, 0.000000
0.013311, -0.147544, -0.134232, 1.980000, -0.265780
0.121244, 0.001940, 0.123184, 1.960000, 0.241440
0.123301, 0.001600, 0.124900, 1.940000, 0.242307
0.122046, -0.094664, 0.027382, 1.920000, 0.052574
0.107073, -0.088503, 0.018569, 1.900000, 0.035282
0.106014, -0.092869, 0.013145, 1.880000, 0.024712
0.106435, -0.152741, -0.046306, 1.860000, -0.086129
0.108402, -0.152774, -0.044372, 1.840000, -0.081645
0.099727, -0.202831, -0.103104, 1.820000, -0.187649
0.093067, -0.192124, -0.099057, 1.800000, -0.178303
0.092921, -0.334464, -0.241543, 0.000000, -0.000000
episode: 27 training
episode: 27 trainend
episode: 27 score:-0.000000 max:0.402377
0.000827, 0.001682, 0.002508, 1.980000, 0.004967
-0.000493, 0.006273, 0.005780, 1.960000, 0.011328
-0.000279, -0.082330, -0.082609, 1.940000, -0.160262
0.023547, 0.063201, 0.086749, 1.920000, 0.166558
0.023583, 0.063201, 0.086785, 1.900000, 0.164891
0.023771, -0.013533, 0.010238, 1.880000, 0.019248
0.024863, -0.013493, 0.011370, 1.860000, 0.021149
0.024359, -0.031456, -0.007097, 1.840000, -0.013059
0.024146, -0.031480, -0.007334, 1.820000, -0.013348
0.024016, -0.031590, -0.007573, 1.800000, -0.013632
0.024213, -0.031590, -0.007377, 0.000000, -0.000000
```

（8）过程总结

- 其实可以看到，训练过程还是跟传统梯度下降训练网络概念相对一致，有趣的地方在于状态化和游戏化。
- 以探索经验来说，强化学习的关键在于反馈，试想怎么走都没有分数的话，模型无论如何学不会怎么走。
- 热启动机制设置，通过保存最优结果在下次重新开始游戏前进行预训练。

（9）TODO

- 游戏动作优化；
- 奖励或惩罚函数修正；
- 神经网络模型结构优化，对回合数据进行batch；

四、相关讨论

为加强理解，以下引用与某算法大佬的讨论过程：

> Q：先赞再评，个人感觉强化学习跟有监督学习相比，只是sample收集机制不同，能够边学边收集

> A：差不多，但不甚具体。
> 本质差异在于：
> 【1】强化学习数据更序列化且是需反馈；
> 【2】强化学习target是估计所得；
> 【3】强化学习强调过程化，更有生命；
>
> 引用周志华《机器学习》
> “但不同的是，在强化学习中并没有监督学习中的有标记的样本，换言之，没有人直接告诉机器在什么状态下应该做什么动作，只有等到最终结果揭晓，才能通过‘反思’之前的动作是否正确来进行学习，因此，强化学习在某种意义上可看作具有‘延迟标记信息’的监督学习问题”。

五、业务应用

以下我们讨论将**强化学习**概念引进响应型营销策略，如账单分期外呼策略，为解决实验设计自动化提供新思路。

（1）背景介绍

首先简化账单分期外呼策略，假定有策略分组ABCD及其分配外呼量，如下：

	组A	**组B**	**组C**	**组D**
外呼分配占比	25%	25%	25%	25%

但实际上，

	组A	**组B**	**组C**	**组D**
实验组实际响应率	32.00%	20.00%	8.00%	4.00%
空白组实际响应率	28.00%	10.00%	6.00%	3.98%
提升度	1.143	2.000	1.333	1.005

现为提升外呼渠道效应以及全渠道响应率，需要对策略组分配占比进行调整，使外呼资源往**高提升度组**上倾斜。

（2）方法应用

引入**强化学习**，定义关键参数：

- 调整系数：即占比调整系数，假定0.01
- 状态S：现分配系数矩阵[0.25, 0.25, 0.25, 0.25]
- 动作a：调整动作A>B，即分配占比A-0.01、B+0.01，得到[0.24, 0.26, 0.25, 0.25]，共计12种
- 奖惩：根据业务目标，可综合响应率、办理量、手续费收入的提升度
- 回合：结合策略迭代周期，可细化到月、周、日

最终将实现自动化调整分配占比。

[END]2020.08, IvanaXu.