

# 60倍回报! AI工程师用OpenAI创建了一个比特币自动交易工具! 这里是详细做法 | 技术头条

原创 Adam King 区块链大本营 2019-05-13 18:06



作者 | Adam King

译者 | Guoxi

责编 | Aholiab

出品 | 区块链大本营 (blockchain\_camp)

炒股的人都知道，天天盯着大盘做决策不仅让人劳神，还让人秃头。所以一堆顶级的数学家开始用数学的手段进行股市预测。

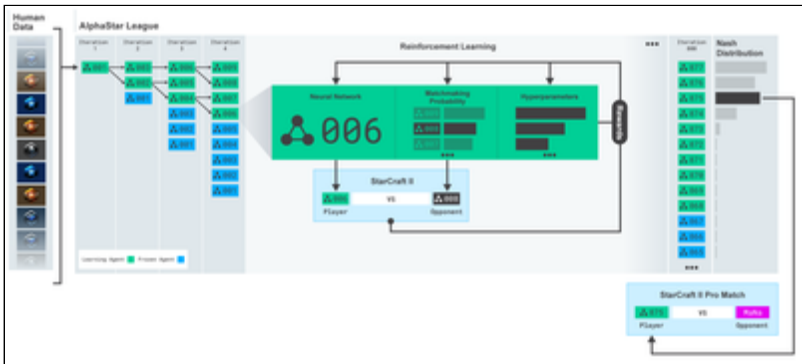
**加密货币市场也一样，而且加密货币市场波动更加频繁，更加剧烈。**对于这个问题，国外加密货币开发者 Adam King 提出了一种新的解决思路。

**结合人工智能在预测方面得天独厚的优势，Adam 提出了使用深度强化学习构建加密货币自动交易程序**，同时，这位小哥还做出了一个能够真正交易比特币的展示模型，他是怎么做到的？这个自动交易程序又能达到怎样的效果呢？让我们在文中一探究竟。

在本文中，我们将使用深度强化学习建立一个加密货币自动交易智能体（agent），并训练它通过交易比特币盈利。

为了避免重复造轮子，在本篇教程中我们将使用人工智能研究机构 OpenAI 开发的程序包。

目前人工智能在很多领域都已经超过了人类，从最初谷歌 DeepMind 团队开发的 AlphaGo 战胜围棋世界冠军李世石，到后来师出同门的 AlphaStar 在星际争霸中以 10：1 的大比分战胜两位职业玩家。近日，OpenAI 团队的 OpenAI Five 在 Dota2 游戏中以 2：0 的比分将世界冠军 OG 斩于马下。



谷歌 DeepMind 团队星际争霸人工智能产品 AlphaStar 的训练过程

人工智能给我们带来了很激动人心的结果，虽然我们不会构建像 AlphaGo 这样令人印象深刻的产品，但在日常的比特币交易中实现盈利也非易事。

因此，**与其冒着脱发的风险苦苦探索比特币币价的规律，何不让人工智能来一展身手？**

在本文中，我们将通过人工智能技术完成一下三个尝试：

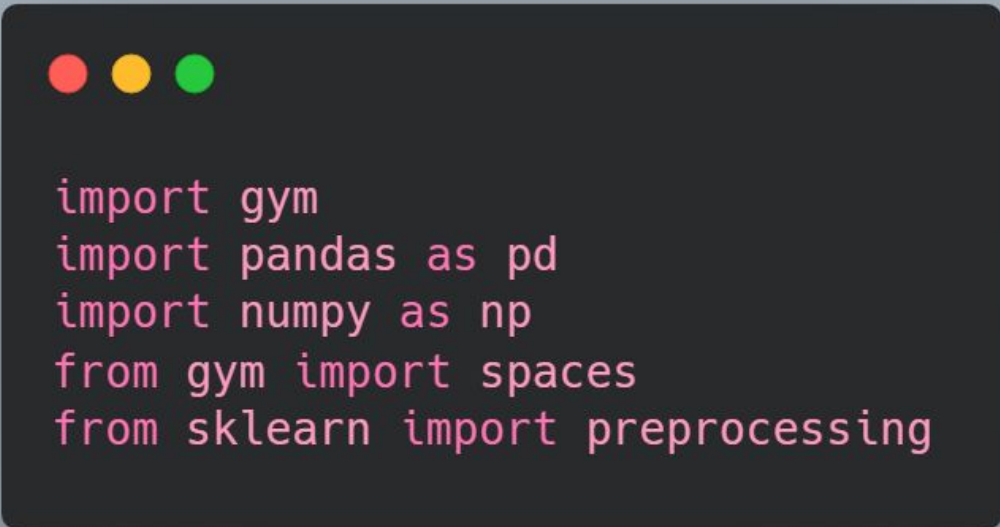
- 为我们的智能体（agent）创建一个测试强化学习的 gym 环境；
- 以一种简单、优雅的方式可视化我们的测试环境；
- 训练我们的智能体，让它学习到能获益的比特币交易策略。

这里有些操作可能会比较麻烦，就比如说从头开始构建 gym 测试环境并将测试环境可视化，不过不要担心，我会详细介绍这些细节，跟上我的节奏就好。

## 程序库安装

在本教程中，我们将使用 Zielak 提供的 Kaggle 数据集。如果你想要这些数据，你可以在我的 Github 仓库中下载 .csv 数据文件。

首先，我们来导入所有必要的 Python 程序库。如果你的电脑上还没有安装这些程序库，你可以使用 `pip install` 命令进行安装。



```
import gym
import pandas as pd
import numpy as np
from gym import spaces
from sklearn import preprocessing
```

接下来，我们创建一个比特币交易环境的类。我们需要向其中传入一个 pandas 数据帧，一个用于指示智能体在每一个时间步长（time step）需要分析前几个时间步长数据的回顾窗口大小（lookback\_window\_size），以及可选的智能体账户初始余额（initial\_balance）。

代码中我们将手续费（commission）设置为每笔交易的 0.075%，也就是加密货币期货交易所 Bitmex 当前的费率，同时，我们将序列运行（serial）参数默认为否（false），这意味着在默认情况下我们的数据帧将以随机的形式遍历各个片段。

除此之外，我们还在数据帧上分别调用了删除非数字（NaN, Not A Number）所在行的 `dropna` 函数以及在删除了数据之后重新设置数据帧索引的 `reset_index` 函数。

```

class BitcoinTradingEnv(gym.Env):
    """A Bitcoin trading environment for OpenAI gym"""
    metadata = {'render.modes': ['live', 'file', 'none']}
    scaler = preprocessing.MinMaxScaler()
    viewer = None
    def __init__(self, df, lookback_window_size=50,
                  commission=0.00075,
                  initial_balance=10000,
                  serial=False):
        super(BitcoinTradingEnv, self).__init__()
        self.df = df.dropna().reset_index()
        self.lookback_window_size = lookback_window_size
        self.initial_balance = initial_balance
        self.commission = commission
        self.serial = serial
        # Actions of the format Buy 1/10, Sell 3/10, Hold, etc.
        self.action_space = spaces.MultiDiscrete([3, 10])
        # Observes the OHCLV values, net worth, and trade history
        self.observation_space = spaces.Box(low=0, high=1, shape=(10,
                                                                lookback_window_size + 1), dtype=np.float16)

```

代码中 `action_space`（操作空间）的第一个数字表示可选的 3 个选项，即买入，卖出或持有，第二个数字表示所操作的比例，最小单位是 10%，也就是说这个数字中的 1, 2, 3 分别代表 10%, 20%, 30%。当选择买入操作时，具体买入的比特币数量将是第二个数字（`amount`）乘以当前账户的比特币余额（`self.balance`）。对于卖出操作，具体卖出的比特币数量也是第二个数字（`amount`）乘以当前账户的比特币余额（`self.balance`）。

当然了，如果选择持有操作，那么就不会买卖账户中的比特币，因而第二个数字就没有了意义。

我们的 `observation_space`（观察空间）被定义为 0 到 1 之间的连续浮点数集，它的大小为（10，回顾窗口大小（`lookback_window_size`）+ 1）。这里的 + 1 操作是考虑到了当前这一时间步长的操作。对于窗口中的每一步，我们都将观察它的收盘价位置价值（OHCLV），我们在那时的资产总价值、买入或卖出的比特币数量、以及我们在买入或卖出这些比特币时花费的美元数。

接下来，我们需要编写重新设置（`reset`）函数来初始化比特币交易环境。



```
def reset(self):
    self.balance = self.initial_balance
    self.net_worth = self.initial_balance
    self.btc_held = 0
    self._reset_session()
    self.account_history = np.repeat([
        self.net_worth,
        0,
        0,
        0,
        0
    ], self.lookback_window_size + 1, axis=1)
    self.trades = []
    return self._next_observation()
```

代码中我们使用了重新设置会话控制（`self._reset_session`）函数和下一次数据观察（`self._next_observation`）函数，不过这些函数都还没有被定义，接下来我们来定义它们。

## 交易会话控制

交易会话控制（`session`）是比特币交易环境中的一个重要组成部分。如果我们将这个智能体部署到外部，我们可能永远都不会一次让它运行几个月的时间。出于这个原因，我们将在数据帧参数（`self.df`）中限制智能体能够连续看到的数据帧数量。

在我们的重新设置会话控制（`_reset_session`）函数中，我们首先将当前的时间步长（`current_step`）重新设置为 0。接下来，我们将剩余时间步长（`steps_left`）设置为 1 到最大交易会话控制数（`MAX_TRADING_SESSION`）之间的随机数，当然了，最大交易会话控制数需要在文件的顶部定义。

接下来，如果需要连续遍历数据帧，那就应该设置遍历所有的数据帧，否则我们需要在数据帧参数（`self.df`）中设置一个随机的数据帧起始位置（`frame_start`），并创建一个名为激活数据帧（`active_df`）的新数据帧，它是数据帧（`self.df`）从起始位置（`frame_start`）到起始位置 + 剩余时间步长（`frame_start + steps_left`）这些连续帧组成的切片。

使用数据帧切片带来的一个重要影响就是，智能体将获得更多独一无二的的数据，以便进行长时间的训练。举个例子，如果我们只是按顺序来遍历数据帧（即按数据帧 0 到最后一帧（`len(df)`）的顺序），那么我们就只有数据帧个数这么多的唯一数据点。我们的观察空间在每一个时间步长只能观察区区几个状态。

但是，通过随机遍历数据帧的切片，我们有效地结合了原始数据集上每一个时间点的账户余额，交易数据以及当前比特币价格，从而创造出了更多独一无二的的数据点。接下来我们通过一个例子来说

明一下。

我们的智能体在每个时间步长中都有三种选择：买入，卖出或持有。对于这三种选择中的每一种，都还需指定操作比特币的数量，如操作当前比特币余额的 10%，20%，或是 100%。这意味着我们的智能体在每个时间步长中都有 30 种不同的选择（当然了，对于持有操作，这 10 种选择的效果是一样的），而它从中选出最好的一个。

回到我们随机切片后的比特币交易环境。在第 10 个时间步长中，我们的智能体可以处于数据帧内的任何数据帧长度（len（df））时间步长。考虑到每个时间步长智能体可以做 30 种选择，这意味着在任意 10 个时间步长的间隔时间中，该智能体可以经历数据帧长度（len（df））的 30 次方种可能的唯一状态。

虽然这样的操作可能会给大型数据集带来相当大的噪声，但我相信这是一把双刃剑，这样我们的智能体也会从有限的数据量中学到更多。不过，对于测试数据集，我们仍将按顺序来遍历，这样做更贴近于“实时”的交易数据，因而可以更好更精确地检测我们的智能体。

## 比特币交易智能体都学到了些什么

为了更好地了解智能体所看到并学习到的特征，我们需要将比特币交易环境的观察空间可视化。比如说，下面是使用 OpenCV 可视化渲染后的观察空间。

*OpenCV 可视化渲染后的观察空间*

图像中的每一行都代表我们观察空间（observation\_space）中的一行。**前4行类似于频率的红线代表了 OHCL 数据，下方的橙色和黄色的点代表着数量**，再下方这个起伏不定的蓝色长条是智能体所拥有资产的总价值，而下方颜色较浅的点代则表智能体的交易。

如果你眯着眼睛看这张图，你就可以看到一个 K 线图，下面有着代表数量的指示条以及一个显示交易历史的类似于莫尔斯电码的界面。**看起来我们的智能体应该能够在观察空间（observation\_space）的数据中学到一些东西**。在这里，我们将定义下一次观察（\_next\_observation）函数，在这个函数中我们要将观察到的数据缩放到 0 到 1 之间。

重要的一点是，**仅仅缩放智能体到目前为止所观察到的数据，以避免出现前视偏差**（ Look-ahead bias，前视偏差是指在策略的开发中，采取了未来的一些信息，而这些信息在实盘操作中是基本上不可能得到的）。

## 编写步骤

现在我们已经设置好了观察空间，是时候编写我们的操作步骤（ step ）函数了，这个函数可以指导智能体的行为。

每当当前交易时段的剩余操作步骤（ self.steps\_left ）等于 0 时，我们将卖出所持有的所有比特币并调用重新设置会话控制（ \_reset\_session ）函数。

否则，我们将智能体的奖励（ reward ）设置为当前所持有资产的总价值，如果智能体的资金用完了，则只会将完成（ done ）设置为真（ True ）。



其实，采取行动的过程十分简单，也就只有三步：

**第一步**，获取当前的比特币价格（ `current_price` ）；

**第二步**，确定该买入卖出还是持有，以及所要操作的份额；

**第三步**，就是真实买入或卖出这些比特币。现在我们来编写采取行动（ `_take_action` ）函数，以便于测试我们的比特币交易智能体。

最后，在这个函数中，我们将交易添加到交易记录参数（ `self.trades` ）中，并更新我们的资产总价值和账户交易历史。

到这里，我们的智能体就可以启动新环境，在新环境中学习比特币交易的特征，并采取行动以获得收益。是时候让比特币交易智能体一展身手了。

## 查看比特币交易智能体的交易记录

上文中说到了，我们需要将智能体的学习和决策过程可视化。当然了，仅仅使用最简单的方法，在智能体每次决策后输出智能体所持有资产的总价值（`print (self.net_worth)`）也不是不可以，不过这样做就少了很多的乐趣。因此，我们决定绘制一个简单的比特币价格数据 K 线图，其中包含数量栏和我们资产总价值的单独图表。

在代码中，我们需要定义一个用来可视化的资产交易图（`StockTradingGraph`）函数，在函数的初始化过程中，我们需要调用 python 可视化程序库 `matplotlib.pyplot`，并指出每一个需要可视化的数据。

为了更好地展现数据，在可视化方法中我们需要导入 Python 时间日期（datetime）处理模块，在数据上标注出人类可读的日期和时间。

在导入完成后，我们需要使用将时间戳转换为世界统一时间 UTC 的 `utcfromtimestamp` 函数，将每个时间戳转化为 UTC 时间，**然后用计算机时间函数（`strftime`）将这个 UTC 时间按照“年 - 月 - 日 小时：分钟”的格式展现出来。**

到这里，可视化函数的各个部分都已编写完成，回到比特币交易环境，我们现在可以汇总出一个可视化（`render`）函数来显示图形。

ok了！我们现在可以看到智能体正在交易比特币。



**图中绿色的竖线代表智能体在买入比特币，红色的竖线代表智能体在卖出比特币。** 右上角的白色方框是智能体所持有资产的总价值，在其下方的白色方框是当前比特币的价格。

这里容我自恋一下，我认为这个可视化的效果简单而又不失优雅。现在，是时候训练我们的比特币交易智能体了，看看它能帮我们赚到多少钱！

## 训练比特币交易智能体

由于我们训练智能体时使用的是时间序列数据，因此在交叉验证方面我们并没有太多的选择。

就拿一种常见的交叉验证形式： $k$ -fold ( $k$ 组) 交叉验证来举例，在  $k$ -fold 交叉验证中，你需要将数据拆分成  $k$  个相等的分组，将每一个分组分别做一次测试组，其余的  $k-1$  组数据用作训练组。

然而，时间序列数据与时间有着高度的依赖性，这意味着后面出现的数据高度依赖于先前出现的数据。所以在这种情况下  $k$ -fold 将不起作用，因为这样会让我们的智能体提前知道未来的数据，即使盈利了我们也不知道是得益于智能体精准的预测还是因为智能体作弊了。

当应用于时间序列数据时，大多数其他的交叉验证策略也都存在着同样的缺陷。因此，我们只需在完整数据帧中给定一个分界点，前一部分的数据用作训练集，其余的数据用作测试集。



接下来，由于我们的比特币交易环境被设置为仅处理单个数据帧，因此我们需要创建两个比特币交易环境，一个用于训练数据，一个用于测试数据。

到这里，我们就可以训练模型了。如下面的代码所示，我们只需要在比特币交易环境中创建智能体，然后调用 `model.learn` 命令开始训练。

在这里，我们会使用机器学习框架 `tensorflow` 的可视化工具 `tensorboard`，从而我们可以轻松地可视化 `tensorflow` 的数据流图并查看有关我们智能体的一些量化指标。

**比如说，下图展示了智能体在经过 200000 个时间步长后的盈利：**

---



看起来我们的智能体都获得了很多的收益！最好的一个智能体在 200,000 个时间步长后资产总价值提升了 1000 倍，而其余的智能体资产总价值平均提升了 30 倍以上！

不过，就在这时，我意识到比特币交易环境中存在一个错误.....在修复了该错误之后，这是新的收益图：

正如你所看到的，我们的一些智能体做得很好，而有一些则表现很差。**总的来说，表现良好的智能体最多能够实现资产总价值提升 10 倍甚至 60 倍。**

我必须承认，所有这些智能体都是在虚拟的比特币交易环境中训练和测试的，所以将这个比特币交易智能体直接应用于比特币区块链上还为时尚早。**但是至少这个结果告诉我们，使用人工智能来进行加密货币交易决策这条路是行得通的。**

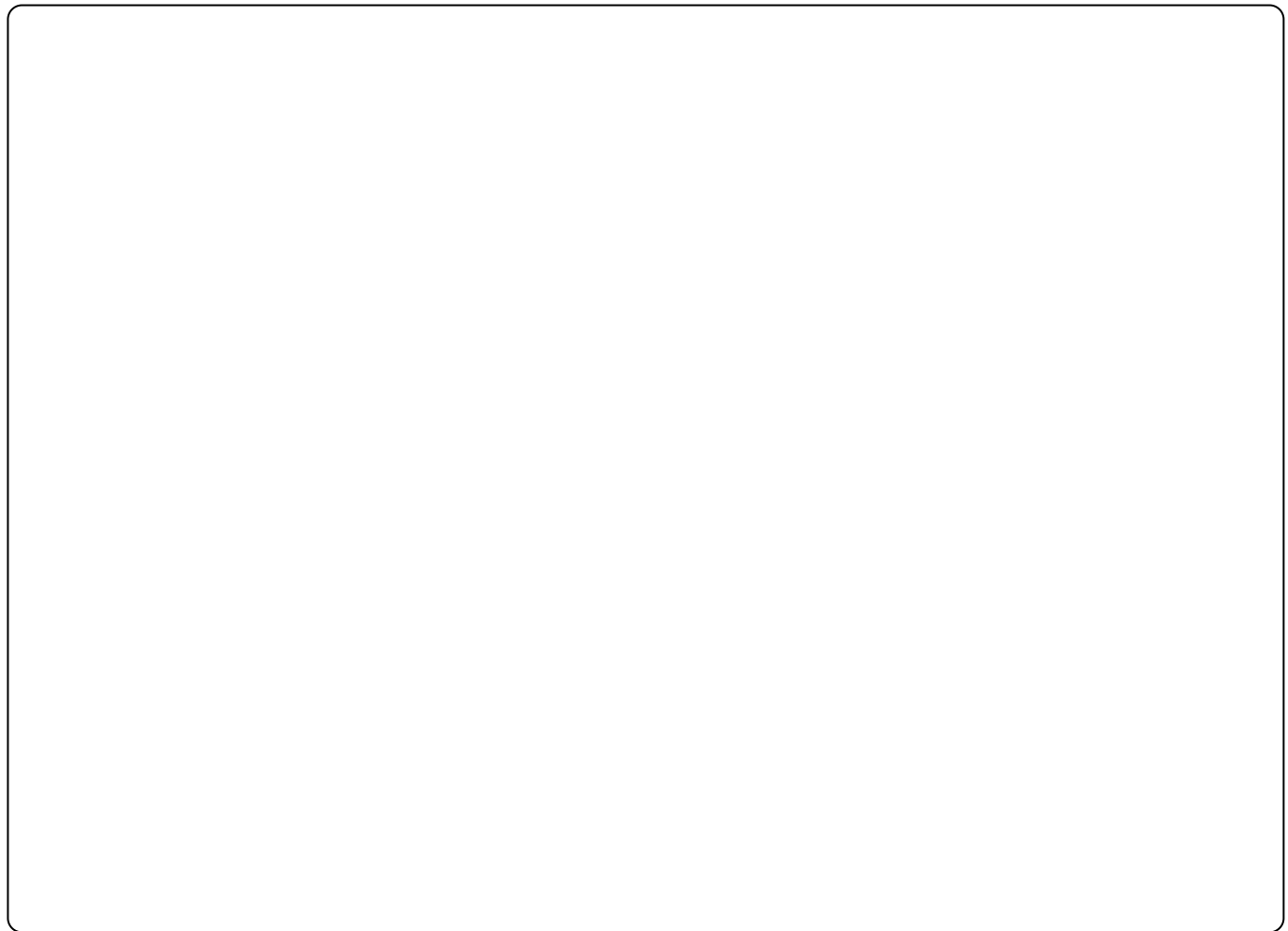
接下来，让我们在比特币交易测试环境中测试智能体，在测试环境中我们将使用智能体以前从未见过的全新数据，看看这些智能体是否学到了比特币的交易策略。

结果显示，我们训练出来的比特币交易智能体在新的测试环境中争相走向破产

这也并不意外，因为我们还有很多的工作要做。仅仅通过简单地将模型从当前的近端策略优化（**Proximal Policy Optimization**，**PPO2**）智能体切换到 **stable-baseline** 程序库中的 **A2C**（**Advantage Actor-Critic**）就可以大大提高我们在此数据集上的性能。

同时，我们也可以更新奖励函数，激励那些资产总价值不断增加的操作，防止有些比特币交易智能体在资产总价值达到高位时就消极怠工。

仅仅做出这两个改动就可以大幅度提高比特币交易智能体在当前数据集上的性能，正如下图所示，最终我们在数据全新的测试环境上成功实现了盈利。



除此之外，我们还可以做得更好。为了提升这些比特币交易智能体的准确度，我们可以优化超参数并训练智能体更长的时间。是时候给你的显卡（深度学习代码运行在显卡之上）一点压力了！

如果你想继续优化，这里可以给你提供些思路，你可以使用贝叶斯优化来在问题空间上寻找最佳的超参数，并使用显卡的 CUDA 运算平台优化训练环境和测试环境。

## 结论

在本教程中，我们使用深度强化学习从零开始创建了一个能够获得收益的比特币交易智能体。

### 具体而言，我们完成了以下的任务：

- 使用 OpenAI 团队开发的用于测试强化学习算法的工具包 gym 从零开始创建了一个比特币交易环境；
- 使用 Python 可视化程序库 Matplotlib 将比特币交易环境可视化；
- 使用简单的交叉验证对我们的比特币交易智能体进行了训练和测试；

- 虽然还有很多的工作需要完成，但现在我们已经可以看到成功的曙光。

虽然最后我们的比特币交易智能体在数据全新的测试环境中还不能保证总是盈利，但我们已经离成功不远了。

---

限时免费报名 | 区块链技术公开课精选

---

**5月22日晚8点，大咖与你一同探讨区块链扩展技术！**

**如何打造未来“黑科技”？**

**多方位+深度详解**

#### **推荐阅读：**

- [增长88%! 2019福布斯全球区块链50强榜单, 你未必看懂这3个细节](#)
- [威胁删库? 程序员: “呵呵, 一分都不给你!”](#)
- [储备金被暗中挪用? USDT信任危机再爆发! 拿什么拯救你我的稳定币?](#)
- [互联网行业人才格局大换血, BAT 已换位?](#)
- [数据库不适合上容器云? | 技术头条](#)
- [肖仰华: 知识图谱落地, 不止于“实现”](#)
- [一顿操作猛如虎! 云原生应用为何如此优秀?](#)
- [补偿100万? Oracle裁900+程序员, 新方案已出!](#)

猛戳"阅读原文"有惊喜哟

老铁在看了吗？👉

阅读原文

喜欢此内容的人还喜欢

IPFS的新移动应用程序——Durin  
IPFS中文资讯



Aleo最全资源大盘点（一）  
Aleo中文资讯



新维度体验混合现实(MR)与Avive  
VV官宣

