Published in Towards Data Science

Bruce Yang Follow

Aug 25, 2020 · 15 min read · ✨ · Listen

Deep Reinforcement Learning for Automated Stock Trading

Using reinforcement learning to trade multiple stocks through Python and OpenAI Gym | Presented at ICAIF 2020

Image by Chris on Unsplash

Note from Towards Data Science's editors: *While we allow independent authors to publish articles in accordance with our rules and guidelines, we do not endorse each author's contribution. You should not rely on an author's works without seeking professional advice. See our Reader Terms for details.*

This blog is based on our paper: ***Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy***, presented at **ICAIF 2020**: ACM International Conference on AI in Finance.

Our codes are available on Github.

GitHub ...

This repository provides codes for ICAIF 2020 paper This ensemble strategy is reimplemented in a Jupiter Notebook at...

github.com

Our paper is available on SSRN.

Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy

Date Written: September 11, 2020 Stock trading strategies play a critical role in investment. However, it is...

papers.ssrn.com

If you want to cite our paper, the reference format is as follows:

> *Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. 2020. Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. In ICAIF '20: ACM International Conference on AI in Finance, Oct. 15–16, 2020, Manhattan, NY. ACM, New York, NY, USA.*

A most recent DRL library for Automated Trading-**FinRL** can be found here:

GitHub - AI4Finance-Foundation/FinRL: A Deep Reinforcement Learning Framework for Automated Trading...

News: we plan to share our codes for both paper trading and live trading. Please actively share your interests with our...

github.com

FinRL for Quantitative Finance: Tutorial for Single Stock Trading

FinRL for Quantitative Finance: Tutorial for Multiple Stock Trading

FinRL for Quantitative Finance: Tutorial for Portfolio Allocation

ElegantRL supports state-of-the-art DRL algorithms and provides user-friendly tutorials in Jupyter notebooks. The core codes <1,000 lines, using PyTorch, OpenAI Gym, and NumPy.

ElegantRL: A Lightweight and Stable Deep Reinforcement Learning Library

Mastering deep reinforcement learning in one day.

towardsdatascience.com

Overview

One can hardly overestimate the crucial role stock trading strategies play in investment.

Profitable automated stock trading strategy is vital to investment companies and hedge funds. It is applied to optimize capital allocation and maximize investment performance, such as expected return. Return maximization can be based on the estimates of potential return and risk. However, it is challenging to design a profitable strategy in a complex and dynamic stock market.

Every player wants a winning strategy. Needless to say, a profitable strategy in such a complex and dynamic stock market is not easy to design.

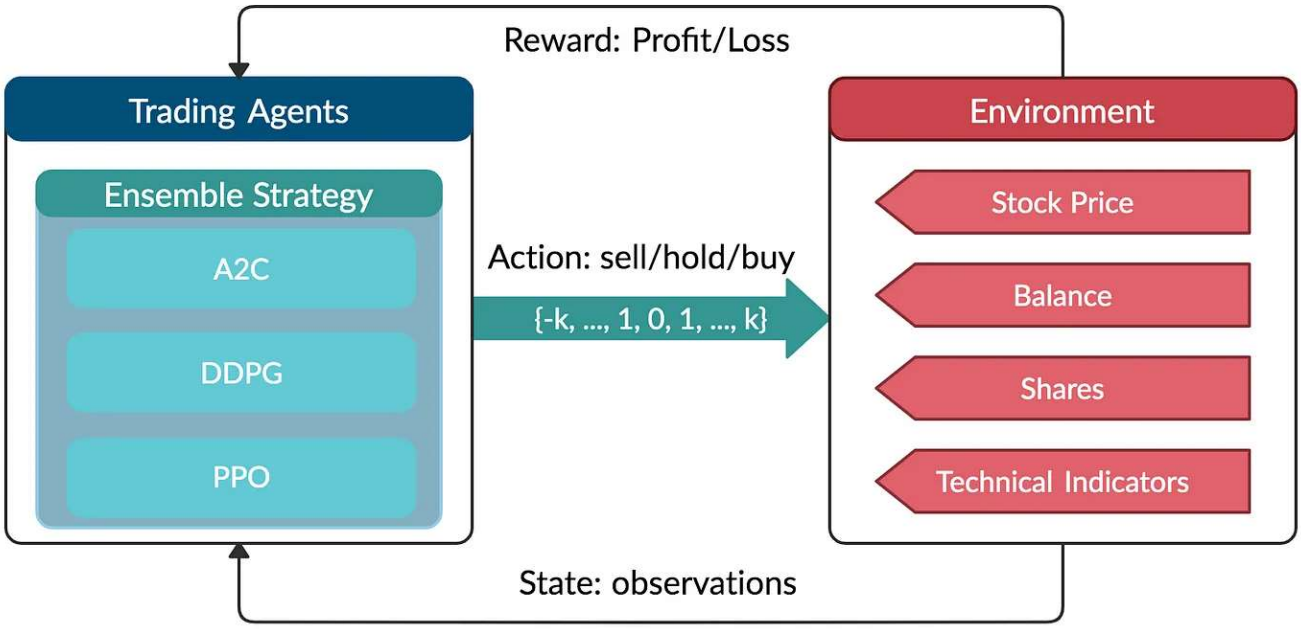
Yet, we are to reveal a deep reinforcement learning scheme that automatically learns a stock trading strategy by maximizing investment return.

Image by **Suhyeon** on Unsplash

Our Solution: Ensemble Deep Reinforcement Learning Trading Strategy
This strategy includes three actor-critic based algorithms: Proximal Policy Optimization (PPO), Advantage Actor Critic (A2C), and Deep Deterministic Policy Gradient (DDPG).

It combines the best features of the three algorithms, thereby robustly adjusting to different market conditions.

The performance of the trading agent with different reinforcement learning algorithms is evaluated using Sharpe ratio and compared with both the Dow Jones Industrial Average index and the traditional min-variance portfolio allocation strategy.



Part 1. Why do you want to use Deep Reinforcement Learning (DRL) for stock trading?

Existing works are not satisfactory. Deep Reinforcement Learning approach has many advantages.

1.1 DRL and Modern Portfolio Theory (MPT)

1. MPT performs not so well in **out-of-sample data**.
2. MPT is very **sensitive to outliers**.
3. MPT is calculated **only based on stock returns**, if we want to **take other relevant factors** into account, for example some of the technical indicators like **Moving Average Convergence Divergence (MACD)**, and **Relative Strength Index (RSI)**, MPT may not be able to combine these information together well.

1.2 DRL and supervised machine learning prediction models

1. DRL doesn't need **large labeled training datasets**. This is a significant advantage since the amount of data grows exponentially today, it becomes very time-and-labor-consuming to label a large dataset.

2. DRL uses a **reward function** to optimize future rewards, in contrast to an ML regression/classification model that predicts the probability of future outcomes.

1.3 The rationale of using DRL for stock trading

1. The goal of stock trading is to **maximize returns**, while avoiding risks. DRL solves this optimization problem by **maximizing the expected total reward** from future actions over a time period.

2. Stock trading is a **continuous process** of testing new ideas, getting feedback from the market, and trying to optimize the trading strategies over time. We can model stock trading process as **Markov decision process** which is the very foundation of Reinforcement Learning.

1.4 The advantages of deep reinforcement learning

1. Deep reinforcement learning algorithms can **outperform human players in many challenging games**. For example, on March 2016, DeepMind's AlphaGo program, a deep reinforcement learning algorithm, beat the world champion Lee Sedol at the game of Go.

2. **Return maximization as trading goal**: by defining the reward function as the change of the portfolio value, Deep Reinforcement Learning maximizes the portfolio value over time.

3. The stock market provides **sequential feedback**. **DRL** can **sequentially** increase the model performance during the training process.

4. **The exploration-exploitation technique** balances trying out different new things and taking advantage of what's figured out. This is difference from other learning algorithms. Also, there is no requirement for a skilled human to provide training examples or labeled samples. Furthermore, during the exploration process, the agent is encouraged to explore the uncharted by human experts.

5. **Experience replay**: is able to overcome the correlated samples issue, since learning from a batch of consecutive samples may experience high variances,

hence is inefficient. Experience replay efficiently addresses this issue by randomly sampling mini-batches of transitions from a pre-saved replay memory.

6. **Multi-dimensional data**: by using a continuous action space, DRL can handle large dimensional data.

7. **Computational power**: Q-learning is a very important RL algorithm, however, it fails to handle large space. DRL, empowered by neural networks as efficient function approximator, is powerful to handle extremely large state space and action space.

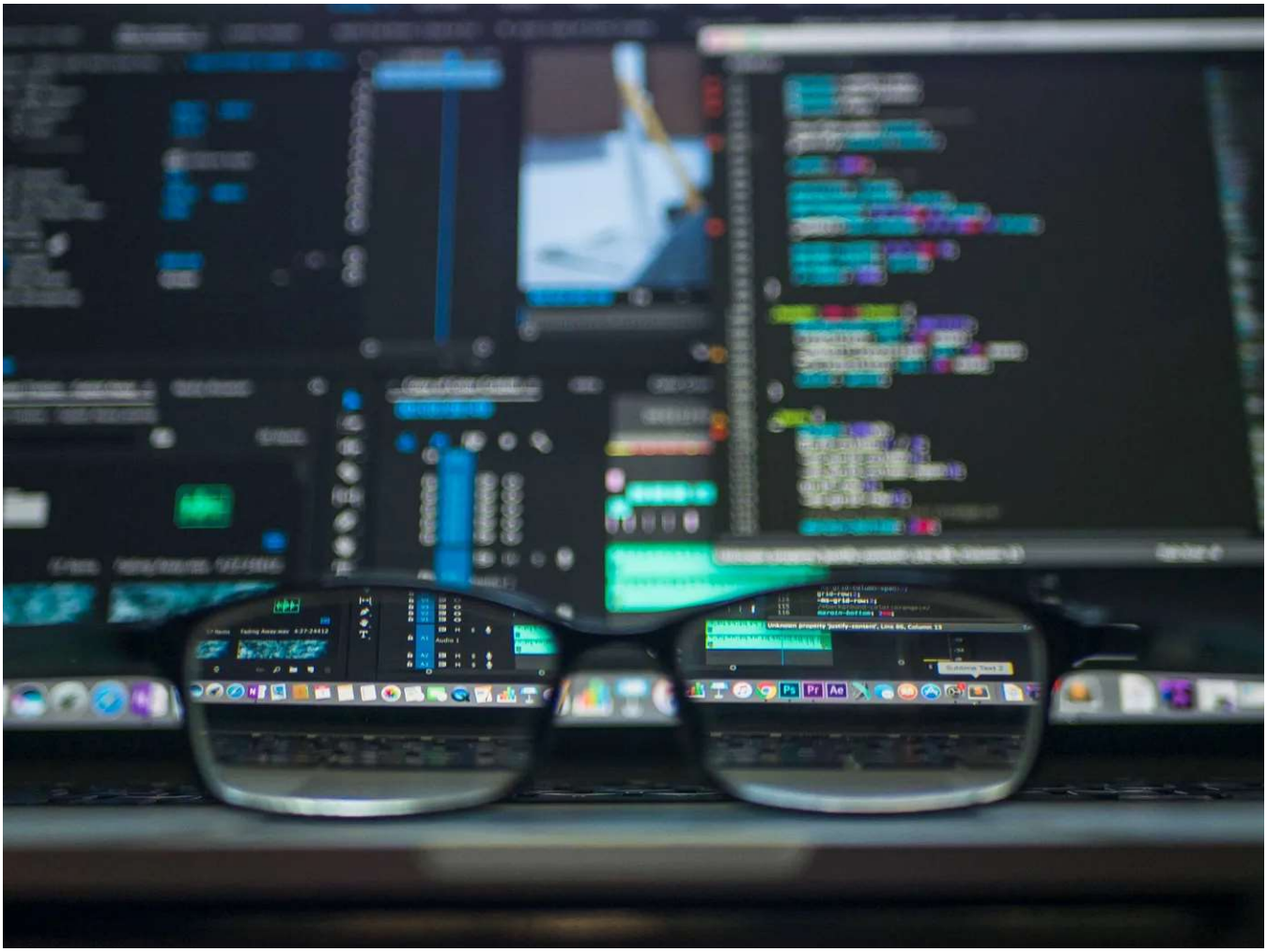


Image by Kevin on Unsplash

Part 2: What is Reinforcement Learning? What is Deep Reinforcement Learning? What are some of the related works to use Reinforcement Learning for stock trading?

2.1 Concepts

Reinforcement Learning is one of three approaches of machine learning techniques, and it trains an agent to interact with the environment by sequentially receiving states and rewards from the environment and taking actions to reach better rewards.

Deep Reinforcement Learning approximates the Q value with a neural network. Using a neural network as a function approximator would allow reinforcement learning to be applied to large data.

Bellman Equation is the guiding principle to design reinforcement learning algorithms.

Markov Decision Process (MDP) is used to model the environment.

2.2 Related works

Recent applications of deep reinforcement learning in financial markets consider discrete or continuous state and action spaces, and employ one of these learning approaches: **critic-only approach, actor-only approach, or and actor-critic approach.**

1. Critic-only approach: the critic-only learning approach, which is the most common, solves a discrete action space problem using, for example, Q-learning, Deep Q-learning (DQN) and its improvements, and trains an agent on a single stock or asset. **The idea of the critic-only approach** is to use a **Q-value function** to learn the optimal action-selection policy that maximizes the expected future reward given the current state. Instead of calculating a state-action value table, **DQN minimizes** the mean squared error between the target Q-values, and uses a neural network to perform function approximation. The major limitation of the critic-only approach is that it only works with discrete and finite state and action spaces, which is not practical for a large portfolio of stocks, since the prices are of course continuous.

- **Q-learning:** is a value-based Reinforcement Learning algorithm that is used to find the optimal action-selection policy using a Q function.
- **DQN:** In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of allowed actions is the predicted output.

2. Actor-only approach: The idea here is that the agent directly learns the optimal policy itself. Instead of having a neural network to learn the Q-value, the neural network learns the policy. The policy is a probability distribution that is essentially a strategy for a given state, namely the likelihood to take an allowed action. The actor-only approach can handle the continuous action space environments.

- **Policy Gradient:** aims to maximize the expected total rewards by directly learns the optimal policy itself.

3. Actor-Critic approach: The actor-critic approach has been recently applied in finance. The idea is to **simultaneously update the actor network that represents the policy,** and **the critic network that represents the value function**. The critic estimates the value function, while the actor updates the policy probability distribution guided by the critic with policy gradients. Over time, the actor learns to take better actions and the critic gets better at evaluating those actions. The actor-critic approach has proven to be able to learn and adapt to large and complex environments, and has been used to play popular video games, such as Doom. Thus, the actor-critic approach fits well in trading with a large stock portfolio.

- **A2C:** A2C is a typical actor-critic algorithm. A2C uses copies of the same agent working in parallel to update gradients with different data samples. Each agent works independently to interact with the same environment.
- **PPO:** PPO is introduced to control the policy gradient update and ensure that the new policy will not be too different from the previous one.
- **DDPG:** DDPG combines the frameworks of both Q-learning and policy gradient, and uses neural networks as function approximators.

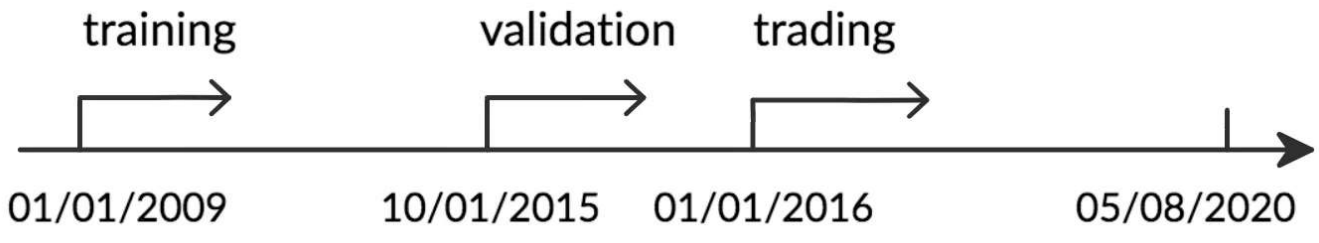
Part 3: How to use DRL to trade stocks?

Image by Markus on Unsplash

3.1 Data

We track and select the **Dow Jones 30 stocks (at 2016/01/01)** and use historical daily data from **01/01/2009 to 05/08/2020** to train the agent and test the performance. The dataset is downloaded from Compustat database accessed through **Wharton Research Data Services (WRDS)**.

The whole dataset is split in the following figure. Data from 01/01/2009 to 12/31/2014 is used for **training**, and the data from 10/01/2015 to 12/31/2015 is used for **validation** and tuning of parameters. Finally, we test our agent's performance on **trading** data, which is the unseen out-of-sample data from 01/01/2016 to 05/08/2020. To better exploit the trading data, we continue training our agent while in the trading stage, since this will help the agent to better adapt to the market dynamics.



3.2 MDP model for stock trading:

• **State** $\boldsymbol{s} = [\boldsymbol{p}, \boldsymbol{h}, b]$: a vector that includes stock prices $\boldsymbol{p} \in$ R+^D, the stock shares $\boldsymbol{h} \in$ Z+^D, and the remaining balance $b \in$ R+, where D denotes the number of stocks and Z+ denotes non-negative integers.

• **Action** $\boldsymbol{a}$: a vector of actions over D stocks. The allowed actions on each stock include selling, buying, or holding, which result in decreasing, increasing, and no change of the stock shares $\boldsymbol{h}$, respectively.

• **Reward** $r(s,a,s')$:the direct reward of taking action a at state s and arriving at the new state s'.

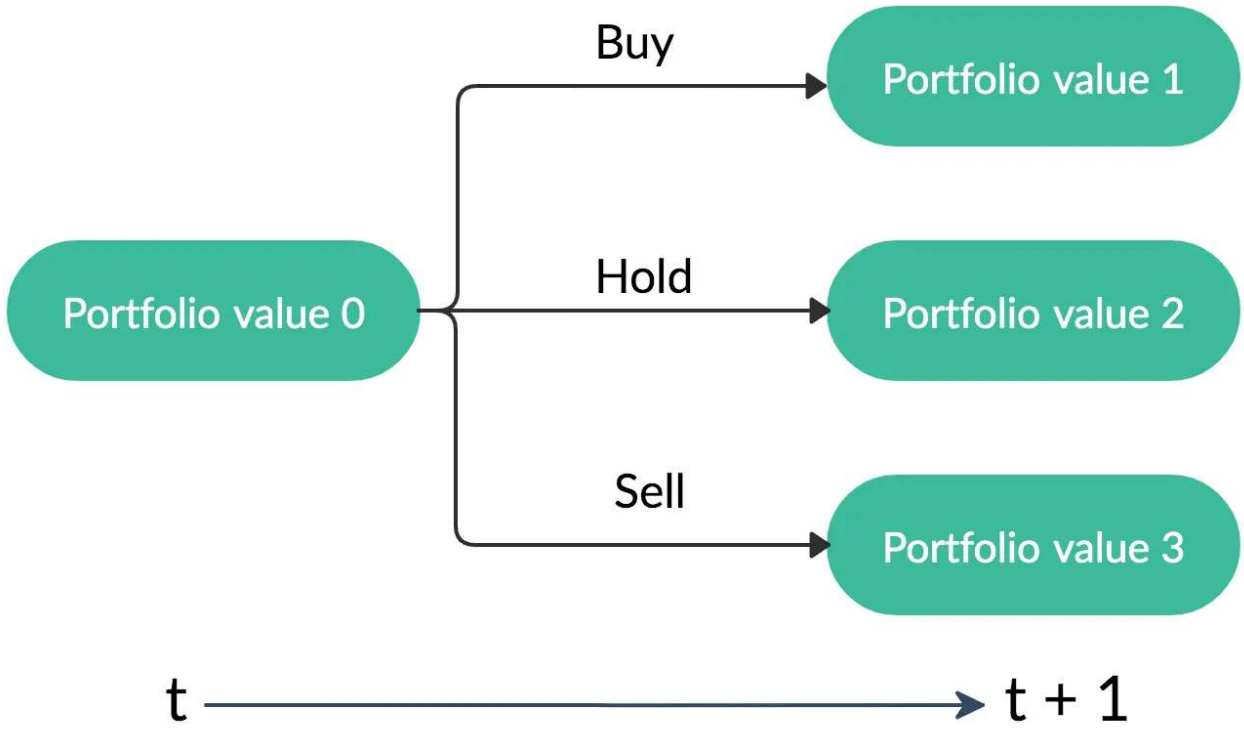
• **Policy** $\pi(s)$: the trading strategy at state s, which is the probability distribution of actions at state s.

• **Q-value** $Q\pi(s, a)$: the expected reward of taking action a at state s following policy π .

The state transition of our stock trading process is shown in the following figure. At each state, one of three possible actions is taken on stock d ($d = 1, \ldots, D$) in the portfolio.

- **Selling** $\boldsymbol{k}[d] \in [1,\boldsymbol{h}[d]]$ shares results in $\boldsymbol{ht}+1[d] = \boldsymbol{ht}\,[d] - \boldsymbol{k}[d]$,where$\boldsymbol{k}[d] \in$Z+ and$d$ =1,…,D.
- **Holding**, $\boldsymbol{ht}+1[d]=\boldsymbol{ht}[d]$.
- **Buying** $\boldsymbol{k}[d]$ shares results in $\boldsymbol{ht}+1[d] = \boldsymbol{ht}\,[d] + \boldsymbol{k}[d]$.

At time t an action is taken and the stock prices update at t+1, accordingly the portfolio values may change from “portfolio value 0” to “portfolio value 1”, “portfolio value 2”, or “portfolio value 3”, respectively, as illustrated in Figure 2. Note that the portfolio value is $\boldsymbol{pT}\,\boldsymbol{h} + b$.



3.3 Constraints:

- **Market liquidity**: The orders can be rapidly executed at the close price. We assume that stock market will not be affected by our reinforcement trading agent.
- **Nonnegative balance**: the allowed actions should not result in a negative balance.
- **Transaction cost**: transaction costs are incurred for each trade. There are many types of transaction costs such as exchange fees, execution fees, and SEC fees. Different brokers have different commission fees. Despite these variations in fees, we assume that our transaction costs to be 1/1000 of the value of each trade (either buy or sell).
- **Risk-aversion for market crash**: there are sudden events that may cause stock market crash, such as wars, collapse of stock market bubbles, sovereign debt default, and financial crisis. To control the risk in a worst-case scenario like 2008 global financial crisis, we employ the financial **turbulence index** that measures extreme asset price movements.

3.4 Return maximization as trading goal

We define our reward function as **the change of the portfolio value** when action a is taken at state s and arriving at new state $s + 1$.

The goal is to design a trading strategy that maximizes the **change of the portfolio value** $r(st,at,st+1)$ in the dynamic environment, and we employ the deep reinforcement learning method to solve this problem.

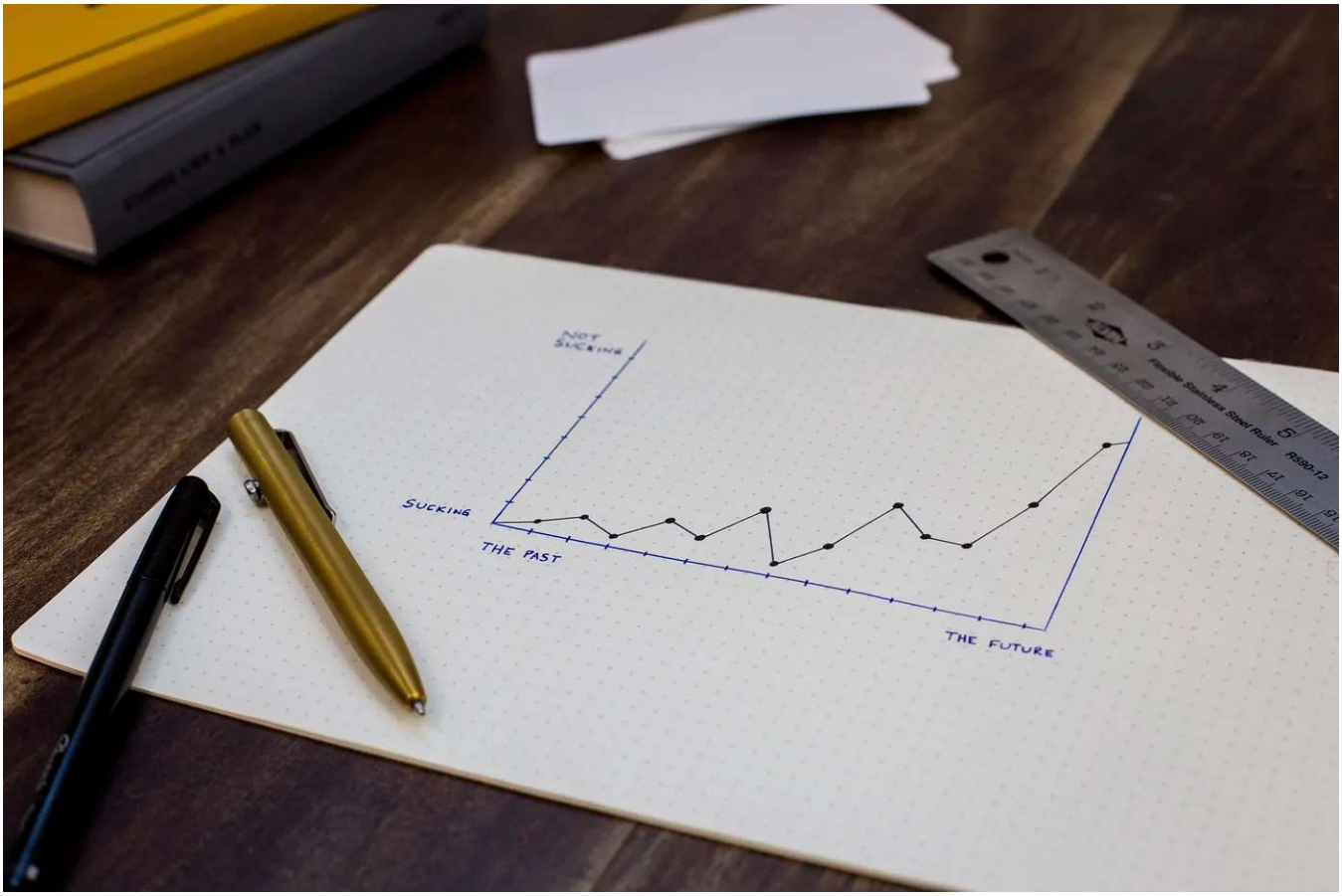


Image by Isaac on Unsplash

3.5 Environment for multiple stocks:

State Space: We use a 181-dimensional vector (30 stocks * 6 + 1) consists of seven parts of information to represent the state space of multiple stocks trading environment

1. **Balance**: available amount of money left in the account at current time step
2. **Price**: current adjusted close price of each stock.
3. **Shares**: shares owned of each stock.
4. **MACD**: Moving Average Convergence Divergence (MACD) is calculated using close price.

5. **RSI**: Relative Strength Index (RSI) is calculated using close price.
6. **CCI**: Commodity Channel Index (CCI) is calculated using high, low and close price.
7. **ADX**: Average Directional Index (ADX) is calculated using high, low and close price.

Action Space:

1. For a single stock, the action space is defined as **{-k,...,-1, 0, 1, ..., k}**, where k and -k presents the number of shares we can buy and sell, and **k ≤h_max** while **h_max** is a predefined parameter that sets as the maximum amount of shares for each buying action.
2. For multiple stocks, therefore the size of the entire action space is **(2k+1)^30.**
3. The action space is then normalized to **[-1, 1]**, since the RL algorithms A2C and PPO define the policy directly on a Gaussian distribution, which needs to be normalized and symmetric.

```
class StockEnvTrain(gym.Env):

"""A stock trading environment for OpenAI gym"""
 metadata = {'render.modes': ['human']}

def __init__(self, df, day = 0):
 self.day = day
 self.df = df

 # Action Space
 # action_space normalization and shape is STOCK_DIM
 self.action_space = spaces.Box(low = -1, high = 1,shape =
(STOCK_DIM,))

 # State Space
 # Shape = 181: [Current Balance]+[prices 1-30]+[owned shares 1-30]
 # +[macd 1-30]+ [rsi 1-30] + [cci 1-30] + [adx 1-30]
 self.observation_space = spaces.Box(low=0, high=np.inf, shape =
(181,))

 # load data from a pandas dataframe
 self.data = self.df.loc[self.day,:]
 self.terminal = False

 # initalize state
 self.state = [INITIAL_ACCOUNT_BALANCE] + \
 self.data.adjcp.values.tolist() + \
```

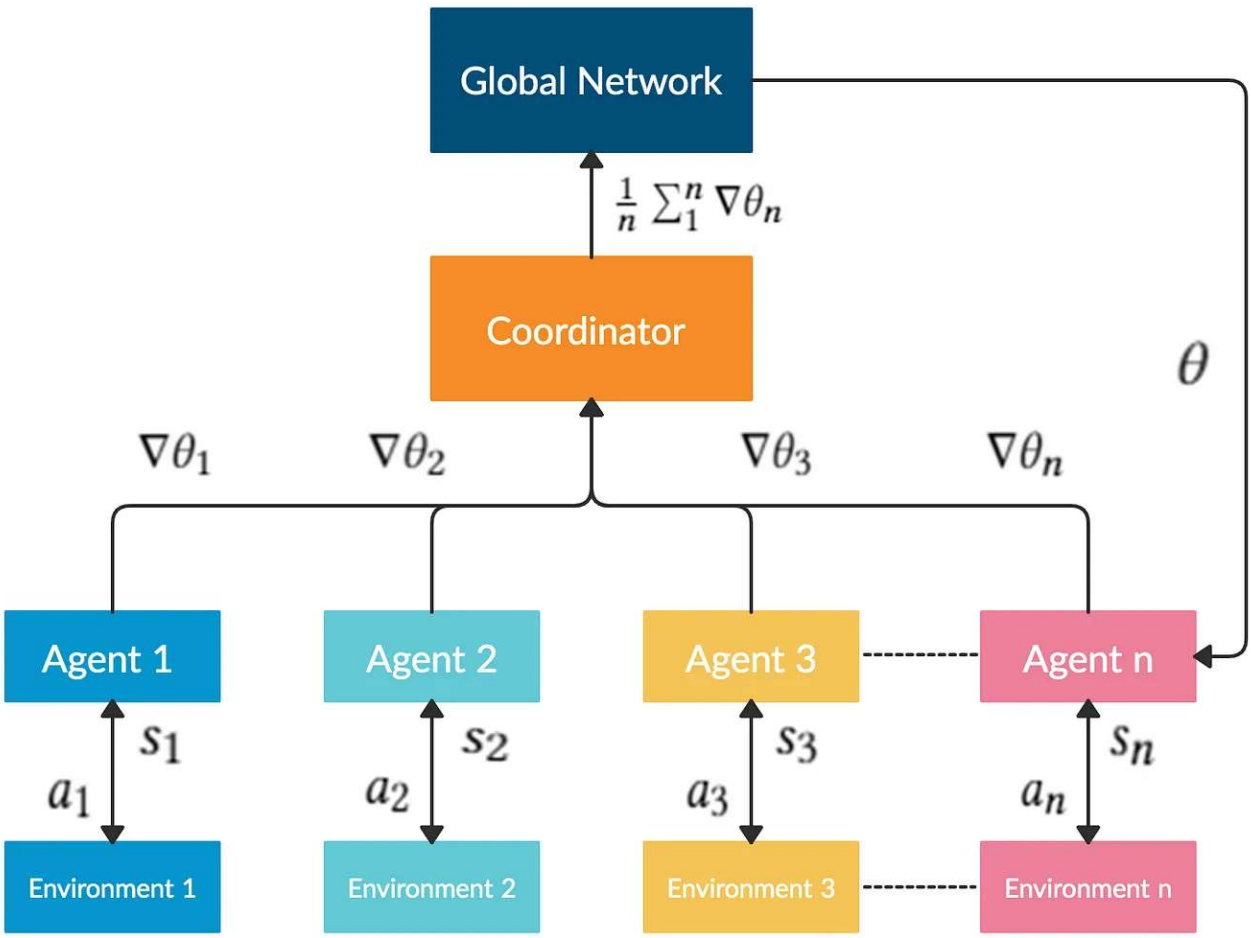
```
[0]*STOCK_DIM + \
self.data.macd.values.tolist() + \
self.data.rsi.values.tolist() + \
self.data.cci.values.tolist() + \
self.data.adx.values.tolist()

# initialize reward
self.reward = 0
self.cost = 0

# memorize all the total balance change
self.asset_memory = [INITIAL_ACCOUNT_BALANCE]
self.rewards_memory = []
self.trades = 0
#self.reset()
self._seed()
```

3.6 Trading agent based on deep reinforcement learning

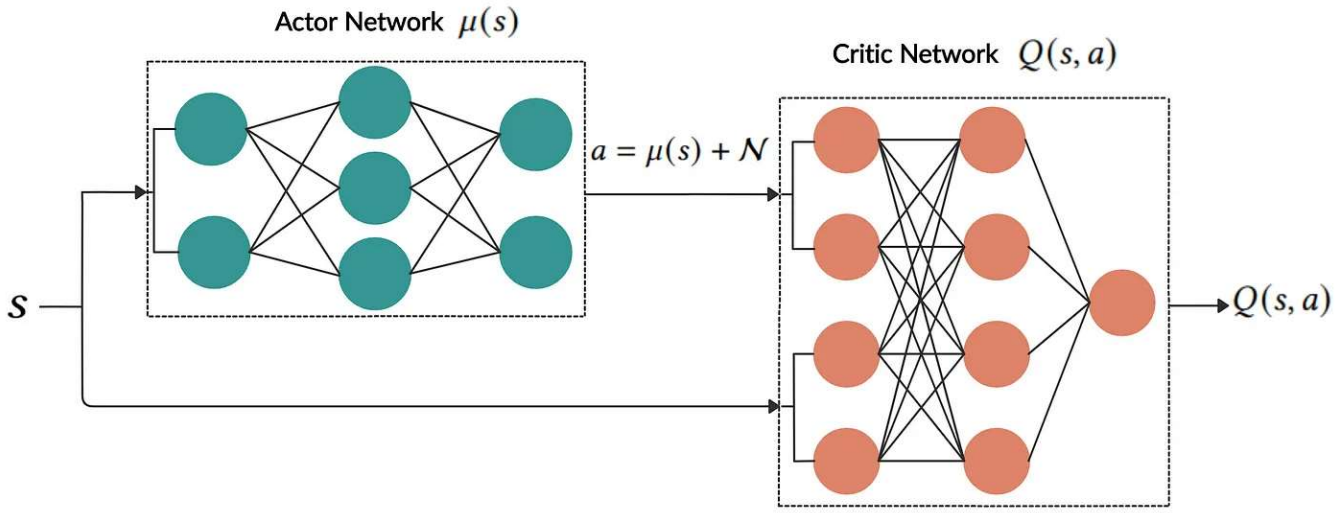
A2C



A2C is a typical **actor-critic algorithm** which we use as a component in the ensemble method. A2C is introduced to improve the policy gradient updates. A2C utilizes an **advantage function** to reduce the variance of the policy gradient. Instead of only estimates the value function, the **critic network estimates the advantage function**. Thus, the evaluation of an action not only depends on how good the action is, but also considers how much better it can be. So that it reduces the high variance of the policy networks and makes the model more robust.

A2C uses **copies of the same agent** working in parallel to update gradients with different data samples. Each agent works independently to interact with the same environment. After all of the parallel agents finish calculating their gradients, A2C uses **a coordinator** to pass the average gradients over all the agents to a **global network**. So that the global network can update the actor and the critic network. The presence of a global network increases the diversity of training data. The synchronized gradient update is more cost-effective, faster and works better with large batch sizes. A2C is a great model for stock trading because of its stability.

DDPG



DDPG is an actor-critic based algorithm which we use as a component in the ensemble strategy to maximize the investment return. DDPG **combines** the frameworks of both **Q-learning and policy gradien**t, and uses neural networks as function approximators. In contrast with DQN that learns indirectly through Q-values tables and suffers the curse of dimensionality problem, DDPG learns directly from the observations through policy gradient. It is proposed to deterministically map states to actions to better fit the continuous action space environment.

PPO

We explore and use PPO as a component in the ensemble method. PPO is introduced to control the policy gradient update and ensure that the new policy will not be too different from the older one. PPO tries to simplify the objective of **Trust Region Policy Optimization (TRPO)** by introducing a clipping term to the objective function.

The objective function of PPO takes the **minimum of the clipped and normal objective**. PPO discourages large policy change move outside of the clipped interval. Therefore, PPO improves the stability of the policy networks training by restricting the policy update at each training step. We select PPO for stock trading because it is stable, fast, and simpler to implement and tune.

Ensemble strategy

Our purpose is to create a highly robust trading strategy. So we use **an ensemble method** to automatically select the best performing agent among **PPO**, **A2C**, and **DDPG** to trade based on the **Sharpe ratio**. The ensemble process is described as follows:

Step 1. We use a growing window of n **months** to retrain our three agents concurrently. In this paper, we retrain our three agents at every **three months**.

Step 2. We validate all three agents by using a **3-month validation** rolling window followed by training to pick the best performing agent which has the **highest Sharpe ratio**. We also adjust risk-aversion by using **turbulence index** in our validation stage.

Step 3. After validation, we only **use the best model with the highest Sharpe** ratio to predict and trade for the next quarter.

```
from stable_baselines import SAC
from stable_baselines import PPO2
from stable_baselines import A2C
from stable_baselines import DDPG
from stable_baselines import TD3
from stable_baselines.ddpg.policies import DDPGPolicy
from stable_baselines.common.policies import MlpPolicy
from stable_baselines.common.vec_env import DummyVecEnv

def train_A2C(env_train, model_name, timesteps=10000):
 """A2C model"""
 start = time.time()
```

```
 model = A2C('MlpPolicy', env_train, verbose=0)
 model.learn(total_timesteps=timesteps)
 end = time.time()

 model.save(f"{config.TRAINED_MODEL_DIR}/{model_name}")
 print('Training time (A2C): ', (end-start)/60,' minutes')
 return model

def train_DDPG(env_train, model_name, timesteps=10000):
 """DDPG model"""
 start = time.time()
 model = DDPG('MlpPolicy', env_train)
 model.learn(total_timesteps=timesteps)
 end = time.time()

 model.save(f"{config.TRAINED_MODEL_DIR}/{model_name}")
 print('Training time (DDPG): ', (end-start)/60,' minutes')
 return model

def train_PPO(env_train, model_name, timesteps=50000):
 """PPO model"""
 start = time.time()
 model = PPO2('MlpPolicy', env_train)
 model.learn(total_timesteps=timesteps)
 end = time.time()

 model.save(f"{config.TRAINED_MODEL_DIR}/{model_name}")
 print('Training time (PPO): ', (end-start)/60,' minutes')
 return model

def DRL_prediction(model, test_data, test_env, test_obs):
 """make a prediction"""
 start = time.time()
 for i in range(len(test_data.index.unique())):
   action, _states = model.predict(test_obs)
   test_obs, rewards, dones, info = test_env.step(action)
   # env_test.render()
 end = time.time()
```

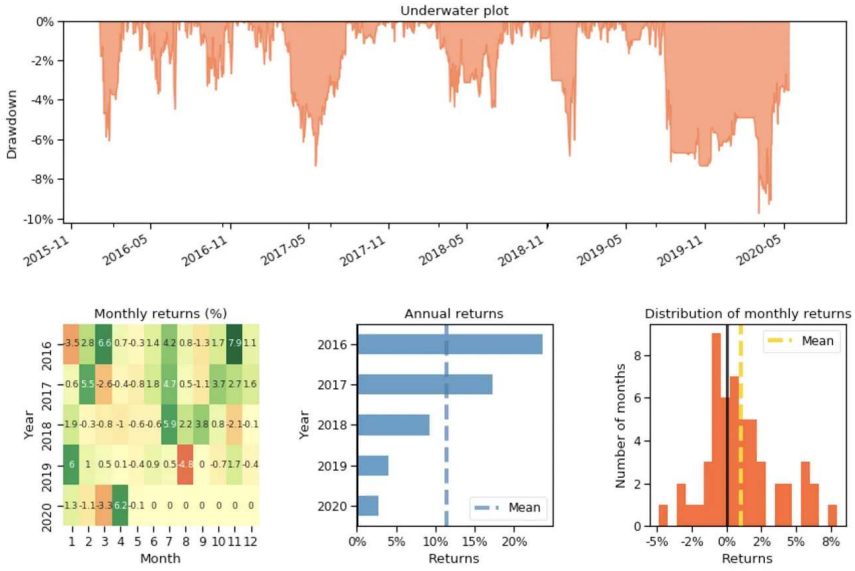
3.7 Performance evaluations

We use Quantopian's pyfolio to do the backtesting. The charts look pretty good, and it takes literally one line of code to implement it. You just need to convert everything into daily returns.

```
import pyfolio

with pyfolio.plotting.plotting_context(font_scale=1.1):
 pyfolio.create_full_tear_sheet(returns = ensemble_strat,
 benchmark_rets=dow_strat, set_context=False)
```

Start date	2016-01-04
End date	2020-05-12
Total months	52
	Backtest
Annual return	12.882%
Cumulative returns	69.466%
Annual volatility	9.725%
Sharpe ratio	1.30
Calmar ratio	1.32
Stability	0.91
Max drawdown	-9.731%



(2016/01/04-2020/05/08)	Ensemble (Ours)	PPO	A2C	DDPG	Min-Variance	DJIA
Cumulative Return	70.4%	83.0%	60.0%	54.8%	31.7%	38.6%
Annual Return	13.0%	15.0%	11.4%	10.5%	6.5%	7.8%
Annual Volatility	9.7%	13.6%	10.4%	12.3%	17.8%	20.1%
Sharpe Ratio	1.30	1.10	1.12	0.87	0.45	0.47
Max Drawdown	-9.7%	-23.7%	-10.2%	-14.8%	-34.3%	-37.1%

References:

A2C:

Volodymyr Mnih, Adrià Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep

reinforcement learning. The 33rd International Conference on Machine Learning (02 2016). https://arxiv.org/abs/1602.01783

DDPG:
Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. International Conference on Learning Representations (ICLR) 2016 (09 2015). https://arxiv.org/abs/1509.02971

PPO:
John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. 2015. Trust region policy optimization. In The 31st International Conference on Machine Learning. https://arxiv.org/abs/1502.05477

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv:1707.06347 (07 2017). https://arxiv.org/abs/1707.06347

Deep Reinforcement | Reinforcement Learning | Machine Learning | Stock Trading | Markov Decision Process