Research Report

ARTICLE INFO

Keywords:
keyword
keyword
keyword
keyword

ABSTRACT

This file is a general template for writing research summaries. First, write a summary of the introduction and literature overview in your own word. Second, provide a very brief summary of the methodology. For equations, use www.mathpix.com. Each paper is summarized in a single file. In each file, use "subsection" to separate introduction, literature overview and methodology. Make sure there are no errors in red in the box next to *Precompile* above.

1. Dixon (2020) Machine Learning in Finance From Theory to Practice

This review is based on the first chapter of Dixon et al. [2020].

1.1. Abstract

In this chapter, the author provides an overview of main machine learning paradigms and their various applications in finance. He also argues against the common belief that modern machine learning techniques are “black boxes”, by showing how they relate to more traditional techniques such as linear regression.

1.2. Introduction

Since Turing proposed his famous imitation game as a measure of intelligence for machines, many other measures have been suggested (Legg and Hutter [2007]). In recent years, artificial intelligence(AI) has made significant progress, outperforming humans in most tasks. Generally, if the number of variables and the space of possibilities is not small, humans fail to see the big picture at once. These situations, however, are where AI truly shines. With the emergence of big data, analysts have gained access to massive sources of data that were not available before. Social media, financial news outlets and earnings announcement reports are examples of these new “alternative datas”. These new sources of data have important distinctions with traditional datasets. In Lopez de Prado [2019] some properties of these new datasets are discussed.

Classical econometric methods usually fail on these new datasets. These methods are often based on linear algebra, and fail when the number of observations is less than the number of variables. Another pitfall of these methods is that they usually can not recognize the topological relationships that shape the network. Therefore, these methods do not do well when faced with complex patterns in high dimensional spaces.

Another source of alternative data is the cryptocurrency market. By analysing the whole transaction graph(which is publicly available in blockchain based cryptocurrencies), one can learn a lot about the flow of information and use the topological structure of this graph to find patterns. This type of thinking can reveal much more about the network than studying the correlations in historical time series of prices. Sovbetov [2018] is a recent example of such analysis, in which the author considers five of the most common cryptocurrencies and examines factors that influence their prices both in short and long term. Other examples can be found in the work of Dyhrberg [2016] and Gomber et al. [2017].

1.3. Paradigms of Machine Learning

Machine learning(ML) is a broad subject. It covers various types of algorithm for pattern recognition, prediction and decision-making. These algorithms can roughly be categorized into three main types.

1.3.1. Supervised Learning

In supervised learning, a machine is provided a set of input-output pairs(training set). Its task is to infer a function f form these labeled training data that can then be used to map new, unseen inputs. Support vector machines, linear and logistic regression and some neural networks are examples of supervised learning algorithms. When using supervised learning, one should be wary of the bias-variance tradeoff as was noted in Geman et al. [1992].

There are two classes of supervised learning models. A *discriminative* model learns the distribution of the output conditional on the input. Whereas, a *generative* model learns the joint distribution of the input and output. So, if we want to train a model to estimate a function $f : X \to Y$ by finding $\Pr(Y|X)$, a generative model does the following:

1. Assume some functional form for $\Pr(Y)$, $\Pr(X|Y)$.
2. Estimate the parameters of $\Pr(Y)$ and $\Pr(X|Y)$ from training data.
3. Calculate $\Pr(Y|X)$ from Bayes rule.

But a discriminative model works as is written below:

1. Assume some functional form for $\Pr(Y|X)$.
2. Estimate the parameters of $\Pr(Y|X)$ from training data.

While discriminative models are used more frequently, generative models can perform better in certain situations. Ng and Jordan [2002] provide an in depth comparison between the two approaches.

Neural networks are popular supervised learning models that use layers of abstraction to construct a non-linear map $F(X)$ over a high dimensional input space. A deep feedforward network, for example, is a function of the form

$$\hat{Y}(X) := F_{W,b}(X) = \left(f^{(L)}_{W^{(L)},b^{(L)}} \circ \cdots \circ f^{(1)}_{W^{(1)},b^{(1)}} \right).$$

Where $W = \left(W^{(1)}, \cdots, W^{(L)}\right)$ and $b = \left(b^{(1)}, \cdots, b^{(L)}\right)$ are weight matrices and biases respectively and $f^{(l)}_{W^{(l)},b^{(l)}}(X) := \sigma^{(l)}\left(W^{(l)}X + b^{(l)}\right)$ is a semi-affine function where $\sigma^{(l)}$ is a univariate, continuous and non-linear function such as $\tanh(.)$, called the *activation* function. The activation functions are used to approximate non-linear functions. In fact there has been multiple *universal approximation theorems* for different neural network architectures showing that neural nets can approximate a large family of functions(such as continuous functions). Hornik et al. [1989] proved such a theorem for neural networks with just one hidden layer.
When the data is a time series, recurrent neural networks(RNNs) are used to build a memory into the model. RNNs are sequence learners that have been successful in various applications that conceptually, benefit from a short-term memory such as natural language understanding, video processing and other many other tasks(Graves [2008]). However, RNNs have historically had difficulty in learning long-term dynamics(due to their structural design) and are therefore, not well-suited for non-stationary data such as financial time series. In these situations, long short term memory networks can be used. These are variants of RNNs that use integrated memory units.

1.3.2. Unsupervised Learning

In unsupervised learning, a machine looks for patterns in a set of unlabeled data. There is very little human supervision required(in contrast to supervised learning that uses a human-labeled training set). Two main methods used in unsupervised learning are principal component and cluster analysis. Algorithms such as k-means clustering, self-organizing maps and Gaussian mixtures are examples of this type of learning.

1.3.3. Reinforcement Learning

In reinforcement learning(RE), a set of *state*s is defined and relative to each state, a set of possible *action*s is available for an agent to perform. Each action produces an immediate *reward* and the agent learns how to maximize its cumulative reward by interacting with the environment. The way RE works, is by trying to understand which states are more profitable(in long term), and refining the agents *policy* over time as it interacts with the environment. A main dilemma that a learning agent like the one modeled in reinforcement learning faces, is to balance the competing goals of exploration of the unknown versus exploitation of past experiences. In recent years, RL has garnered a lot of attention due to its usage in autonomous driving, natural language processing(Paulus [2019]) and multiple Google's deep mind projects such as Silver et al. [2018].

1.4. Statistical Modeling vs. Machine Learning

As was mentioned earlier, modern datasets are very complicated and it is not easy to infer from them on the basis of a known data generation process, as done in standard statistical models. In fact, it seems quite likely that the exact process of data generation is not known. In contrast, machine learning is data-driven and tries to find structures in large datasets without needing any prior assumption on the data generation process. It uses regularization and dropout for variable or predictor selection.

Machine learning and statistical modeling techniques can also be categorized by whether they are parametric or not. In *parametric modeling*, the response is considered to be a function of input variables and parameters. These methods have inherent limitations in modeling big datasets due the finiteness of their parameter space. Least square

linear regression, polynomial regression, mixture models and hidden Markov models are examples of parametric modeling paradigm. *Non-parametric* models use an infinite dimensional parameter space(equivalent to a hidden function). Examples of this kind of modelling are support vector machines and Gaussian processes. Neural networks are usually considered parametric models. But they can also be used as a non-parametric model as in Philipp and Carbonell [2017].

2. de Prado (2020) Machine Learning for Assset Managers

This review is based on the first chapter of de Prado [2020].

2.1. Abstract

In the first chapter of his book, de Prado enumerates the shortcomings of classical statistics and then suggests ways in which machine learning(ML) can be used to address these deficiencies and help build better theories. He also investigates the particular challenges of employing ML techniques in finance.

2.2. Introduction

Most traditional statistical methods rely on simplistic assumptions and are therefore, not particularly successful in modeling complex systems such as financial markets. In this day and age, there is an abundance of data, in all shapes and forms, and a massive computational power is available to us. This means that we can now construct and test much more complex models than those proposed by classical statistics. ML can not only provide researchers with some invaluable insight about the complex system they are trying to understand, but also be very helpful in testing different theories.

In finance, the importance of developing a theory can not be overemphasized. Only a complete and testable theory can explain the cause-effect mechanisms that might allow us to make profit in the market. As such, ML should be used in helping researchers to devise better theories. Using it to discover and backtest trading rules, although possible, is not the correct way of using the powerful tools of ML.

In fact, a common mistake among financial researchers is using historical simulations(backtests) to develop a strategy. Backtests are not theories. They can rule out a bad strategy, but they can never prove that one is good. A strategy based on a theory, can perform well even in situations that has never occurred before. Easley et al. [2012] is an example of a theory that has performed well when faced with a black swan.

2.3. Machine Learning and Financial Theories

Many researchers view ML as a black box; they use it as a prediction oracle that provides no understanding. This is a misconception caused by the fact that modern ML models are not specified explicitly like the classical "parametric" models in statistics. ML should not be used to just make predictions, rather it should be used to see what is predictable and why. For example, one can use ML to uncover the hidden variables involved in a phenomenon, then use them to formulate a theory. Finally, the theory(and not the ML algorithm) is able to make predictions based on logic and reason. Hence, in the previous example, ML helps us by "decoupling the search for variables and the search for specifications".

As was said, ML does not replace theories. However, it can help us form theories that are based on rich empirical evidence. There are a number of ways in which ML can help scientists. These include, but are not limited to:

1. **Existence:** ML algorithms have been used to evaluate the plausibility of a theory. They can be helpful not just in empirical sciences, but even in mathematics; where they can point to the existence of an uncovered theorem. As is stated in Gryak et al. [2020], "if something can be predicted, there is hope that a mechanism can be uncovered". ML can help find areas where predictions are possible.
2. **Importance:** ML can help discover variables(features) that have significant influence in the environment. Although it might not tell us much about the inner workings of the environmental mechanisms, it can help us get a better understanding of what are the important variables that should be considered in any model that hopes to explain the environment. Mean-decrease accuracy(MDA) is one such algorithm.
3. **Causation:** ML can also help in identifying causal relationships. Consider the following experiment: (1) Use historical data to fit an ML algorithm to predict outcomes, absent of an effect(This is a non-theoretical, oracle-like model that is purely driven by data); (2) Collect observations of outcomes under the presence of that effect; (3) Use the ML model developed in (1) to predict observations collected in (2). Now, the prediction error can be attributed to the effect and a causal relationship(or the absence of one) can be proposed(Athey [2015], Varian [2014]).

4. **Reductionist:** Modern data sets are large, high-dimensional and complex. This means that they are much harder to interpret. For most purposes, we would much rather work with smaller data sets in lower dimensions. ML can help us do this. For example, manifold learning algorithms use the assumption that the data of interest lie on an embedded non-linear manifold. If the manifold is of low dimension, the data can be processed and visualised in a lower dimension. Gashler and Martinez [2011] is an interesting application of such techniques in understanding dynamical systems.
5. **Retriever:** ML can be used to inspect large amounts of data, in search of hidden structures or patterns. One example of such application of ML algorithms occurs in cosmology, where ML algorithms scan through the data collected by the telescopes. Once they find an image that contains a supernova, humans can focus their attention to that part of the space and investigate the data(Lochner et al. [2016]). Another example is outlier detection. Finding outliers is a prediction problem, not an explanation problem. ML algorithms can search for anomalous observations, based on some complex structure it has found in data, even if the structures are not well explained to us(Hodge and Austin [2004]).

2.4. Proper use of Machine Learning

An unwelcome byproduct of the amazing flexibility of ML in modeling complex phenomena, is that in the hands of the untrained, it may result in over-complicated models that completely fit the training data(without learning any actual patterns) or over-simplistic models that can not understand the patterns. This can happen in two major forms: underfitting and overfitting. Underfitting happens when the proposed model is too simplistic to capture the main complexities of the system that its trying to model(e.g. a linear regression with too few parameters). On the other hand, one should also be wary of overfitting. The primary symptom of overfitting is a significant difference between in-sample and out-of-sample performance. This is known as *generalization error*. Overfitting errors fall into one of the two categories:

2.4.1. Train Set Overfitting

Train set overfitting happens when we use such a flexible specification that it learns not only the signal, but also the noise. This is specially common in finance, when the signal to noise ratio in the data is small. Backtests might be able to identify this king of overfitting. Researchers, use three main methods to deal with this kind of overfitting. The first approach, is trying to evaluate the generalization error through resampling techniques(such as cross-validation) and Monte Carlo methods. The second approach is using regularization methods, which try to prevent the model from getting too complicated unless it is justified. Regularization can be done by limiting the number of parameters(e.g., LASSO) or restricting model's structure(e.g., early stoppage and drop-out). The third approach is ensemble methods, which uses a collection of estimators in order to obtain better result and reduce the variance of error. Bootstrap aggregating and boosting are examples of these methods.

2.4.2. Test Set overfitting

When a researcher runs the same statistical tests against the same data, the chances are, he is going to eventually make a false discovery. This mistake, known as selection bias, comes from fitting the model to work well on the tests, not the train set! A common source of test set overfitting is backtesting; when a researcher backtests a strategy and tweaks it until it achieves the appropriate results. When faced with a poor performing backtest, one should change the research process instead of fixing a particular flaw in investment strategy. Such practices are similar to *p-hacking* tricks used by uninformed researchers and will inevitably cause test set overfitting. ML can help in dealing with these kinds of issues. First, we should keep track of the number of tests we have run, so that we can calculate the probability that at least one of our observations is a fasle positve. The deflated sharp ratio(Bailey and de Prado [2014]) utilizes this kind of approach. Second, if numerous test sets are used, the chances of test set overfitting greatly decreases. Additional test sets can be generated by resampling combinatorial splits of train and test sets. Third, by using historical series to estimate the underlying data-generation process, we can sample data sets that have the same statistical properties as those observed in history. Monte Carlo methods are particularly helpful here.

All the methods that were suggested above decrease the likelihood of overfitting. However, they are by no means infallible and it is best to use them all. It is worthwhile to point out, once more, that backtests are not controlled experiments. They cannot simulate black swans and provide no context on why a particular strategy might be profitable. Therefore, they can not replace a theory.

2.5. Classical Statistics and Machine Learning

Financial ML is a rather new area of research and like many other new technologies, people have diverse opinions on it. There are those who regard it as the holy grail and there are those who consider it useless. The truth is, ML can be extremely useful in finance and can help us do things that are not possible if we only use classical statistical methods. Practitioners make many common mistakes in both statistics and ML. In statistics, multicollinearity, missing regressors, nonlinear interaction effects are all violations of assumptions used in classical linear regression models. Any of these violations can lead to an inaccurate model if one is using a linear regression based model. Yet, not many statisticians consider these errors in their modeling. Another common error in classical statistical modeling, is thinking that central limit theorem justifies the use of linear regression models everywhere, and that the only problem is obtaining enough observations. These kind of biases, only prevents investors from recognizing what ML can bring to the table. Lopez de Prado [2019] counts ten recent use cases of ML in finance that may give one a better perspective on this matter.

Financial ML has its own challenges that are unique to the financial world. For example, as we mentioned earlier, signal-to-noise ratio is so low in financial data, that relying on black box predictions of any model is virtually impossible. On the other hand, it is estimated that around 80% of all available data are unstructured. Many of the new data sets are high-dimensional, sparse or non-numeric and often contain vital information regarding networks of agents and behaviour of groups of people. As a result of their complexity, linear regression models or other algebraic or geometric approaches can not learn much from them. Generally, traditional regressions try to fit a fixed functional form to a set of variables. When there is strong conviction regarding that functional form and the interactions between variables, they can be very useful. Starting in 1950s, researchers realized that with the help of computers, we do not have to impose a functional form. We can instead let an algorithm conduct experiments and figure out variable dependencies from the data. This relaxation, in combination with the use of powerful computers opened the door to analyzing these new complex data sets that may include nonlinear, hierarchical, and non-continuous interaction effects.

3. Dixon (2020) Machine Learning in Finance From Theory to Practice

This review is based on the ninth chapter of Dixon et al. [2020]

3.1. Abstract

The ninth chapter of Dixon et al. [2020] is dedicated to the basics of reinforcement learning. Throughout this chapter, the reader gains familiarity with fundamental notations in RL such as Markov decision processes, state-value function and action-value function, etc. Classical approaches of solving the reinforcement problem such as dynamic programming(DP), value iteration, Q-learning, etc., are also discussed.

3.2. Introduction

Reinforcement learning(RL) is a type of learning problem in which an *agent* learns to *act* optimally in an *environment* by interacting with the environment and observing the changes in the environment, following his actions. Acting optimally, given a certain goal, is defined as the problem of maximizing an objective function. Deriving the optimal action given the current *state* of the environment and evaluating the value of doing a particular action when in a state, may seem similar to the problem of supervised learning. However, there are at least three major distinctions between RL and supervised learning. The first one, is the role of a teacher. In both RL and supervised learning a teacher is present but their role is rather different in the two settings. In supervised learning the teacher provides the learner with the "correct" output for a training set of inputs; whereas in RL, the teacher provides *partial feedback* to the actions of the agent in the form of an immediate *reward*. For an agent to maximize its return in the long run, an optimal strategy may involve incurring losses in the short term. Thus, the teachers reward contains no explicit information on what the "right" action to do in each state is, when the goal is maximizing the total cumulative reward. The second major difference is the presence of a *feedback loop*. In RL, actions of an agent have an impact on the environment. Because RL tasks usually involve making a decision in an stochastic environment, the presence of this feedback loop means that RL usually involves *planning*. This is in sharp contrast to both supervised and unsupervised learning in which the impact of the agent on the environment is not considered. Finally, a third differnce between RL and other kinds of learning is an *exploration-exploitation dilemma* that an agent faces. Because the optimal strategy in not known to the agent, even when the agent finds a profitable strategy, there is always a chance that an better strategy exists. Thus, the agent has to simultaneously exploit dynamics of environment with its current knowledge, and exploring the environment for better opportunities.

3.3. Fundamentals

Markov decision process(MDP) provides a mathematical framework for decision making in stochastic environments where outcomes are partially affected by actions of a decision maker. MDP extends Markov models by adding extra degrees of freedom for control. Formally, a MDP is defined by a set of discrete time steps $t_0, \cdots, t_n$ and a quintuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ where:

1. $\mathcal{S}$ is the set of all possible states(i.e., *state space*). each time the agent observes the environment, it observes a state $S_t \in \mathcal{S}$. $\mathcal{S}$ can be finite or infinite, in the latter case the MDP is said to have a continuous state space.
2. $\mathcal{A}$ is the set of possible actions, known as *action space*. Alternatively, $\mathcal{A}(s)$ denotes the set of all possible actions available in state s. Again, $\mathcal{A}(s)$ can be discrete or continuous.
3. $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a function that models the transition between states in the environment. For $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, $p(s'|s, a) = \Pr[S_t = s'|S_{t-1} = s, a_{t-1} = a]$ is the probability of the agent being in state s' if in the previous time step, it was in the state s and performed action a.
4. $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ is the *expected reward function* that specifies the expected value of a random reward R_t received in the state $S_{t-1} = s$ when taking action $a \in \mathcal{A}$:

$$r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, a_{t-1} = a].$$

   $\mathcal{R}$ is defined as the set of possible rewards. Usually, $\mathcal{R} = \mathbb{R}$.
5. $\gamma \in [0, 1]$ is the *discount factor* that indicates how much we value immediate reward over the reward obtained in future.

More generally, it is possible to use the joint probability of a next state s' and reward r:

$$p(s', r|s, a) = \Pr[S_t = s', R_t = r'|S_{t-1} = s, a_{t-1} = a].$$

Here, marginal distributions of p will specify reward and state transition functions.

The rewards can explicitly depend on time. In these cases we write the reward function as $r_t(s, a)$ to emphasize its dependence on time.

The discount factor γ is used in computing the *total discounted reward*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

Where R_t is a random variables determining the reward obtained in time step t and G_t is also known as the *total return* from time step t. For infinite-horizon MDPs, we must use $\gamma < 1$ to have a finite total reward.

Traditionally, MDPs are assumed to have *Markov property*. A stochastic process has the Markov property if the conditional distribution of future states depends only on the current state, not on the sequence of events that precede it. In other word, the environment is assumed to be memoryless. If in an environment, the conditional distribution depends on the K most recent states(for a constant K), one can still have the Markov property by extending the definition of states from S_t to $\hat{S}_t = (S_t, S_{t-1}, \cdots, S_{t-K+1})$. Still, in many cases of interest, dynamics of a system cannot be as simple as a Markov process with sufficiently low number of K.

A better way to model a non-Markov environment is to use hidden variables z_t, and then assume that the environment is jointly Markov in the pair (s_t, z_t) so that path probabilities factorize into single step probabilities:

$$\Pr[s_{0:T-1}, z_{0:T-1}] = \Pr[s_0, z_0] \prod_{t=1}^{T-1} \Pr[z_t|z_{t-1}] \Pr[s_t|z_t]$$

These kind of modeling of environment is known as *hidden Markov models*(HMM). Basically, they describe a Markov environment with an observable part s_t and a hidden part z_t.

Even richer types of non-Markov environments can be modeled with recurrent extensions(such as RNN and LSTM neural nets) where hidden state probabilities depend on a long history of previous hidden states, rather than just on the last hidden state.

Now that the framework of MDP is described, we can present the following definitions:

1. ***Policy function:*** The policy function $\pi_t : \mathcal{S} \to \mathcal{A}$ specifies the strategy of an agent. It shows how the agent would act at time t given the state S_t of the environment. A policy may be deterministic(i.e., π_t is a function of state S_t) or probabilistic. In the latter case, $\pi_t(a|s)$ is a probability distribution over $\mathcal{A}$ that depends on the current state S_t. In other words, a policy $\pi_t(a|s)$ takes a current state $S_t = s$ and outputs a distribution $A_t = a$ over the action space.
   It can be proven that an optimal deterministic policy always exists for an MDP. However, in a partially observable MDP(e.g., when working with HMMs) an optimal deterministic policy may not exist.
2. ***State-value function:*** In order to solve the reinforcement learning problem, it is useful to assign a number to each state that represents its "attractiveness" or "value" for an agent in the long run. The value function $V^\pi(S_t) : \mathcal{S} \to \mathbb{R}$ is therefore defined as the expected cumulative reward that an agent that follows policy π accumulates, given its starting state. Hence, $V^\pi(S_t)$ is a function of the current state S_t and a functional of the policy π.
$$V_t^\pi(s) = \mathbb{E}_t^\pi \left[\sum_{i=0}^{T-t-1} \gamma^i R\left(S_{t+i}, A_{t+i}\right) \mid S_t = s \right]$$
   In the above equation for state-value function, T is the planning horizon. For infinite-horizon tasks, $T = \infty$. Also, $\mathbb{E}_t^\pi[\cdot|S_t = s]$ is the conditional expectation from time t, given that future actions are selected according to policy π.
3. ***Action-value function:*** Similar to state-value function, one can define the action-value function $Q_t^\pi(s, a)$ to be the expected cumulative reward that an agent gathers, given that it starts from state s and performs action a, and then acts according to policy π until termination. Thus,
$$Q_t^\pi(s, a) = \mathbb{E}_t^\pi \left[\sum_{i=0}^{T-t-1} \gamma^i R\left(S_{t+i}, A_{t+i}\right) \mid S_t = s, A_t = a \right].$$

3.4. Solving the Reinforcement Learning Problem

3.4.1. Bellman Equations

Recall that the total (random) reward from time t was defined as

$$G_t = \sum_{T-t-1}^{i=0} \gamma^i R(S_{t+i}, A_{t+i}),$$

where S_t, A_t, $R(S_t, A_t)$ are random variables indicating the agents state in time t, the agents action in time t(which depends on its policy) and the reward obtained by moving from state S_t to S_{t+1} by performing action A_t, respectively. Using this notation, the state-value and action-value functions can be written as

$$V_t^\pi(s) = \mathbb{E}_t^\pi[G_t|S_t = s],$$

$$Q_t^\pi(s, a) = \mathbb{E}_t^\pi[G_t|S_t = s, A_t = a].$$

Notice that if we choose action a in the action-value function from the same distribution as policy π, we obtain a relationship between state-value and action-value functions:

$$V_t^\pi(s) = \mathbb{E}^\pi[Q_t^\pi(s, A)] = \sum_{a \in \mathcal{A}} \pi(a|s) Q_t^\pi(s, a).$$

Consider the state-value function. We can separate the first reward from the sum and write it as

$$V_t^\pi(s) = \mathbb{E}[R_t(s, a)] + \gamma \mathbb{E}_t^\pi \left[\sum_{i=1}^{T-t-1} \gamma^i R(S_{t+i}, A_{t+i}) \mid S_t = s \right].$$

Thus, it can easily be seen that the value of a state s can be written as a sum of immediate expected reward and a discounted expectation of the expected next state value:

$$V_t^\pi(s) = \mathbb{E}[R_t(s, A)] + \gamma \mathbb{E}_t^\pi[V_{t+1}^\pi(S_{t+1})|S_t = s].$$

The above equation, known as ***Bellman expectation equation for state-value function***, provides us with a simple recursive scheme to compute the value function at time t from its values at time $t+1$, by going backwards in time from $t = T - 1$. For an MDP with N states, if the value function at time $t + 1$ (V_{t+1}^{π}) is known, the Bellman expectation equation provides us with a set of N linear equations defining $V_t^{\pi}(s)$ for all states s in terms of expected immediate reward and transition probabilities and the next-step value function, $V_{t+1}^{\pi}(s')$. Thus, if transition probabilities are known, the value function at each time-step can be found by recursively solving systems of linear equations.

We can also use the same logic to derive a recursive formula for solving the action-value function. The below equation is known as ***Bellman expectation equation for action-value function*** and is used similar to the Bellman equation for value function:

$$Q_t^{\pi}(s,a) = \mathbb{E}[R_t(s,a)] + \gamma \mathbb{E}_t^{\pi}[Q_{t+1}^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a].$$

The *optimal value function*, V_t^*, is the value function that has the highest value at **each** state among all possible policies. This value function is attained for some *optimal policy* π_*. It can be proved that for any MDP an optimal value function and policy exist:

$$\forall s \in \mathcal{S} \;\; V_t^*(s) := V_t^{\pi_*}(s) = \max_{\pi} V_t^{\pi}(s).$$

The optimal policy can also be determined in terms of optimal action-value function:

$$\forall s \in \mathcal{S}, a \in \mathcal{A} \;\; Q_t^*(s,a) := Q_t^{\pi_*}(s,a) = \max_{\pi} Q_t^{\pi}(s,a).$$

The following equation relates the optimal value function to the optimal action-value function:

$$V_t^*(s) = \max_{a \in \mathcal{A}} Q_t^*(s,a)$$

From the above equations, one can derive the following two equations which are called ***Bellman optimality equation for action-value function*** and ***Bellman optimality equation for state-value function***, respectively:

$$Q_t^*(s,a) = \mathbb{E}_t^* \left[R_t(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}(S_{t+1}, a') \right],$$

$$V_t^*(s) = \max_{a \in \mathcal{A}} \mathbb{E}_t^* \left[R_t(s,a) + \gamma V_{t+1}^*(S_{t+1}) \right].$$

Notice that due to the presence of max operator, these equations, unlike their expectation counterparts, are not linear and can't be solved using linear algebraic methods. Also, note that, if one knows either state-values or action-values, finding the optimal policy reduces to a simple one step maximization.

3.4.2. Dynamic Programming

Dynamic programming(DP), first introduced by Bellman and Kalaba [1957] is a problem solving technique that can be used to find optimal policies in a finite and fully observable MDP. In practice, DP methods are used when the number of possible states is small and the model of environment is perfectly known. Although in many cases of interest one does not know the environment perfectly or the number of states is not small, it is still useful to examine DP techniques as they can inspire us for finding better methods that do not have these limitations.

Policy Evaluation: The problem of "evaluating" a policy π is to find its corresponding value function, $V_t^{\pi}(s)$. Let us consider time-independent problems in which $V_t^{\pi} = V^{\pi}$. In this setting, evaluating a policy π reduces to solving a system of $|S|$ linear equations given by Bellman expectation equation for value function. As $|S|$ grows, solving a system of $|S|$ linear equations(which involves matrix inversion) becomes computationally infeasible. Bellman equations can be used to solve these equations recursively. Although this produces a multi-step numerical algorithm for finding V^{π}, in practice, they tend to work much better than linear algebraic techniques. In order to evaluate a policy π using Bellman iterations, we follow these steps:

```
Policy_Evaluation(π):
    Initialize V_0^π
    k ←1
    Repeat until convergence:
        For every state s ∈ S:
            V_k^π(s) = E^π [R(s, A = π(s)) + γ V_{k−1}^π(S_{t+1})]
            k ← k + 1
```

In the above algorithm, k is an iteration counter and because transition probabilities in the environment are known, the expected value is just a linear combination of value function in the previous iteration.

Policy Iteration: Policy iteration is another classical algorithm in DP and RL, where the goal is to find the optimal policy. A policy iteration algorithm consists of two components: 1. A method for evaluating a given policy, 2. A method to improve a given policy. In the previous section, we saw that given a policy π, one can evaluate it (i.e., find its value function) using Bellman expectation equation. Here we show how one can improve upon a given policy. To this end, we use one form of Bellman expectation equation for action-value function, which is presented below for convenience:

$$Q(s,a) = \sum_{s'} p(s'|s,a)(R(s,a) + \gamma V^{\pi}(s'))$$

Here, if we choose action a according to policy π, we end up with the state-value function for π: $Q^{\pi}(s,\pi(s)) = V^{\pi}(s)$. This means that by choosing another action(with another policy π'), we can obtain higher values for $Q^{\pi}(s,\pi'(s))$. As a result, by changing the first action we take, we end up with a better policy that has a higher value function(This is known as the policy improvement theorem, for more details, see Sutton and Barto [2018] or Szepesvári [2010]). In order to find a new action to perform, we greedily choose action a such that it maximizes $Q^{\pi}(s,\cdot)$. As such, a complete policy iteration algorithm is obtained:

```
Policy_Iteration():
    π_0 ← initial random policy
    k ←1
    Repeat until convergence:
        V^{(k−1)} ← Policy_Evaluation(π_{k−1})
        For every state s ∈ S:
            π^{(k)}(s) ←arg max_{a∈A} Σ_{s'} p(s'|s, a)[R(s, a) + γ V^{(k−1)}(s')]
        k ←k + 1
```

The above algorithm will converge for finite MDPs with bounded rewards.

Value Iteration: Value iteration is yet another classical algorithm for solving a fully-known, finite MDP and finding its optimal policy and value function. It is similar to policy iteration, but it combines the two steps of policy evaluation and improvement and recursively optimizes the value function without an explicit policy making. It starts by initializing the value function, e.g., for all states $s \in \mathcal{S}$, $V^{(0)}(s) = 0$. It then iteratively updates the value function using Bellman optimality equation. The algorithm is presented below:

```
Value_Iteration():
    Initialize V_0
    k ←1
    Repeat until convergence:
        For every state s ∈ S:
            V^k(s) = max_a E* [R(s, a) + γ V^{(k−1)}(s')] = max_{a∈A} Σ_{s',r} p(s', r|s, a) [r + γ V^{(k−1)}(s')]
        k ← k + 1
```

Notice that the update rule in value iteration is quite similar to that of policy iteration, except that it involves taking a sum over all actions $a \in \mathcal{A}$.

In order to update the value function, we can completely calculate $V^{(k)}$ for all states s and then simultaneously update the value function for all states, $V^{(k-1)}(s) \rightarrow V^{(k)}(s)$. This is called *synchronous* updating. Another option is to update the value function on the fly, as it is computed in the current iteration. This approach is called *asynchronous*

updating. Implementing async updating is very easy, we just omit the iteration index k and use a single function(in implementation, a look-up table) V instead of $V^{(0)}, \cdots, V(T)$. Both of these will converge to the optimal value function.

As can be seen, DP methods can only be used when the number of states is relatively small. For real-world problems, where the set of all possible states is huge, these methods are too slow and require an enormous amount of memory. This is known as the *curse of dimensionality*.

3.4.3. Reinforcement Learning Methods

As was said in the previous section, DP methods become extremely inefficient as the number of states in the problem grows. Recall that DP methods try to find the exact value of state-value function for all states in the state space. In high-dimensional problems, it is more efficient to approximate the value function using some number of parameters that is much less than the number of states in the problem. Like DP, reinforcement learning methods have the same goal of computing the value function. However, they are usually sample based and try to find a "sufficiently good" approximate solution, rather than the exact one. Therefore, they are much more efficient and can actually be used to solve problems of interest in the real world, where dynamics of the environment is not known and the set of states is very large. Three main classes of RL approaches are Monte Carlo methods, policy search methods, and value-based RL.

RL methods do not rely on a perfect model of the environment. *Model-free* RL works directly with samples of data. This, however, does not mean that an imperfect model of the environment is totally useless. In fact, *model-based* RL builds an internal model of the environment as an auxiliary tool that helps in finding the optimal policy. For now, only model-free RL methods are considered.

Monte Carlo Methods: Much like any other RL method, Monte Carlo methods(MC) use samples of states, rewards and actions instead of a perfect model of the environment. They require no prior knowledge of the environment and provide a simulation-based approach for solving MDPs. MC methods are restricted to episodic tasks that eventually terminate, or have a finite planning horizon $T < \infty$. They work directly with the definition of action-value function. Recall that

$$Q_t^\pi(s,a) = \mathbb{E}_t^\pi[G_t | S_t = s, A_t = a].$$

If we have access to N observations of the return $G_t^{(1)}, \cdots, G_t^{(N)}$, we could use the empirical mean to estimate the value of Q function at (s,a):

$$Q_t^\pi(s,a) \approx \frac{1}{N} \sum_{n=1}^{N} \left[G_t^{(n)} | S_t = s, A_t = a \right].$$

Note that each of the N observations is a complete T-step trajectory:

$$G_t^{(i)} = \sum_{k=0}^{T-1} \gamma^k R^{(i)}(S_{t+k}, A_{t+k}).$$

In the above equation, π is the policy using which the data was collected. This means that MC is an *on-policy* algorithm, i.e., It can learn the optimal policy, only if the data it observes comes from the optimal policy. In contrast, an *off-policy* algorithm can learn the optimal policy even if its observations are generated a sub-optimal policy. Off-policy MC methods are discussed in Sutton and Barto [2018]. MC method, requires multiple observations from each state-action pair (s,a), in order to find the value function of the policy π being used. The number of these pairs is $|S||\mathcal{A}|$ which is usually a big number. Also, these estimates are independent for different pairs (s,a). This can be useful as it enables an easy parallelization scheme; however, this also means that MC methods do not *bootstrap*, i.e., they do not use previous or related evaluations to estimate the action-value function at (s,a).

In addition to evaluating a policy, MC methods can also be used to find the optimal policy, provided that the MC agent has access to a real-world or simulated environment. To this end, the agent will use an algorithm similar to policy iteration. For each policy π, A set of N trajectories are sampled using this policy, and the action-value function is estimated using their empirical mean. Then, the agent improves the policy by acting greedily in picking the first action, similar to the policy improvement step in policy iteration algorithm: $\pi'(s) = \arg\max_a Q^\pi(s,a)$. The agent will then repeat the process using the newly upgraded policy π' and does so until convergence(or a fixed number of steps).

It is also possible to update the action-value function after each observation of a trajectory, rather than updating it after all N trajectories are sampled. This is called *batch-mode evaluation* and will convert the algorithm into an *online*

algorithm that updates itself after each observation(of a full trajectory) using the following rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(G_t^{(i)}(s,a) - Q(s,a)).$$

Where $0 < \alpha < 1$ is the learning rate. Notice that this is not a truly online algorithm, as it must wait T time steps so that one sample trajectory finishes, before being able to use the total return G_t.

Policy-Based Learning: Policy-based RL uses a different approach than value-based RL in that it does not try to find the best policy by finding the optimal value function. Policy-based RL considers a family of stochastic policies, $\pi_\theta(a|s)$, which is a probability distribution over the set of possible actions, $\mathcal{A}$. θ is the parameter of this distribution. These method use a relation named "log-likelihood trick" which is obtained by computing the derivative of an expectation $J(\theta) = \mathbb{E}_{\pi_\theta(a)}[G(a)]$. Generally, $G(a)$ can be any function. In the context of RL, $G(a)$ is usually the discounted random return which depends on the action taken. According to the log-likelihood trick,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta(a)}\left[G(a)\nabla_\theta \log \pi_\theta(a)\right].$$

Basically, this means that the gradient of J with respect to θ is the expected value of the function $G(a)\nabla_\theta \log \pi_\theta(a)$. This means that if we can sample from the distribution $\pi_\theta(a)$, we can calculate this function and have an unbiased estimate of $\nabla_\theta J$ with just sampling.

The simplest policy search algorithm, called "REINFORCE", is based on the log-likelihood trick. It starts with an initial value of parameter θ_0 and an iteration counter $k = 0$. At each step, it uses MC sampling to obtain an estimate of the gradient of $J(\theta)$ and uses gradient descent to find the appropriate θ that maximizes the reward. it updated the parameter θ_k using the samples it attains, by the following relationship:

$$\theta_{k+1} = \theta_k + \alpha_k G(a_k)\nabla_\theta \log \pi_{\theta_k}(a_k).$$

Here, α_k is the learning rate at step k, and a_k is obtained by sampling from π_k.

One advantage of these methods is that they can have very flexible parameterization of action policies and can even use neural networks to help model these parameters.

Temporal Difference Learning: One big disadvantage of MC methods is that they must wait until the end of an episode to determine the increment of the action-value function update. Temporal difference(TD) methods use a different mechanism for updating the action-value function that does not need to wait until the end of an episode. Like MC, TD methods can be used for both policy evaluation and policy improvement. Let us first see how TD evaluates a given policy π by computing its state-value function, V_t^π.

When value and policy iteration in DP were discussed, we saw how one could obtain an update rule from Bellman equations. TD uses a similar approach to bootstrap and update the value function using only a single observation. Assume that TD agent was in state$s$ and performed action a. It received a reward r and ended up in state s'. TD uses this single observation to compute an error term δ_t known as *TD error*:

$$\delta_t = R_t(s,a,s') + \gamma V_{t+1}(s') - V_t(s).$$

TD uses δ_t that was observed, to update the value function at state s:

$$V_t(s) \leftarrow V_t(s) + \alpha\delta_t.$$

Where α is the learning rate. Notice the difference between MC and TD update rules. While MC has to wait until the end of an episode(T time-steps) to obtain a sample of G_t that is needed for calculating the updates, TD obtains a sample of δ_t after only one time-step and can immediately update the value function. This means that TD methods,unlike MC, can be used in scenarios where episodes are very long or even infinite.

This method of updating the value function after only one observation is known as TD(0). Instead of one-step updates, one can use multi-step updates. This approach will result in more general TD(λ) and n-step TD methods. Details of these methods can be found in Sutton and Barto [2018].

SARSA and Q-Learning: In applying TD methods for learning the action-value function $Q(s,a)$, one should differentiate between on-policy and off-policy algorithms. Both type of algorithms use the Bellman optimality equation for action-value function but the details of updates vary.

First, consider the case of on-policy learning. If we know that the data was collected under an optimal policy, actions observed correspond to the optimal value function. Thus, the update rule becomes similar to the update rule for value function:

$$\delta_t = R_t(s, a, s') + \gamma Q_{t+1}(s', a') - Q_t(s, a),$$
$$Q_t(s, a) \leftarrow Q_t(s, a) + \alpha \delta_t.$$

This on-policy algorithm is known as SARSA; its name comes from the fact that it uses a quintuple (s, a, r, s', a') of observation to make an update.

Now let us consider the case of off-policy learning. In this case, the update rule must involve a max operator, similar to the Bellman optimality equation itself:

$$Q_t(s, a) \leftarrow Q_t(s, a) + \alpha \left[R_t(s, a, s') + \gamma \max_{a'} Q_{t+1}(s', a') - Q_t(s, a) \right].$$

This is known as Q-learning and is one of the most popular RL algorithms to date. Notice the difference between SARSA and Q-learning. In SARSA, both actions a, a' come from the same policy(by which the data was collected). In Q-learning, however, the next action a' is chosen greedily such that $Q_{t+1}(s', a')$ is maximized. This greedy choice of next action a', is what makes Q-learning an off-policy algorithm that is able to learn form data generated by sub-optimal policies.

Q-learning may look like the perfect algorithm for finding the optimal policy as it works even when the data is generated by some random policy. However, this ability of Q-learning comes with a price which is caused because of the presence of max operator in the update rule. Unlike SARSA, Q-learning can not perform an update just by observing a single transition $(s, a) \xrightarrow{r} (s', a')$, because the chosen next action a' may not maximize $Q_{t+1}(s', a')$. This means that Q-learning should maintain the action-value function for all previously visited (s, a) and use it to estimate $\max Q(s', a')$.

Stochastic Approximations and Batch-Mode Q-Learning: As we have seen, when the model of the environment is not available to us, we try to approximately solve the Bellman equations by replacing the expectations in them by their empirical average. Stochastic approximations such as Robbins-Monroe algorithm (Robbins and Monro [1951]) estimate the mean without directly summing the samples. The idea behind it is simple: Assume we want to calculate the mean of n observations, $x_1, \cdots, x_n$. Instead of waiting to see all observations and then finding the mean using the formula $\frac{1}{n}\sum_{i=1}^{n} x_i$, we can add them one by one, updating the running estimate of the mean,$\hat{x}_k$ iteratively after each observation(k is the number of iterations). The update rule for the current estimate of the mean is $\hat{x}_{k+1} = (1 - \alpha_k)\hat{x}_k + \alpha_k x_k$. Here α_k is the learning rate and should satisfy the following equations:

$$\lim_{n\to\infty} \sum_{k=1}^{n} \alpha_k = \infty, \quad \lim_{n\to\infty} \sum_{k=1}^{n} \alpha_k^2 < \infty$$

Under these constraints, it can be proved that this method of computing the mean converges to the real mean with probability one. Using this method of calculating the mean, we can rewrite Q-learning's update rule as

$$Q_t^{(k+1)}(s, a) = \left(1 - \alpha_k\right) Q_t^{(k)}(s, a) + \alpha_k \left[R_t\left(s, a, s'\right) + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}^{(k)}\left(s', a'\right) \right]$$

Alternatively, stochastic approximation can be used in an offline manner by using chunks of data at a time, to iteratively update models parameter. Batch versions of stochastic approximation is particularly useful in continuous state-action spaces.

Q-Learning in Continuous spaces: So far, we have used a tabular representation of the action-value function that stores $Q(s, a)$ for all state-action pairs. In many cases, this is not possible due to the large memory requirements of this method. In order to address this issue, one can approximate the action value as:

$$Q(s, a) = \sum_{k=0}^{K-1} \theta_k \psi_k(s, a).$$

Here, $\psi_0, \cdots \psi_{K-1}$ are a set of basis functions and coefficients $\theta_0, \cdots, \theta_{K-1}$ are free parameters. We should find values of these parameters such that they best fit Bellman optimality equations. Using the linear combination of a set of K

basis functions to approximate an arbitrary function is known as a *linear architecture*. These linear architectures can be used to represent value function, action-value function, policy function, etc. The main advantages of these architectures are their robustness and computational simplicity. On the other hand, choosing a good set of basis functions is very important and non-trivial. This is a general problem in machine learning known as feature construction(or extraction) problem. One possible alternative to linear architectures is *non-linear architectures* that use generic function approximation tools such as trees or neural networks. In particular, *deep reinforcement learning* is when one uses a deep neural network to approximate value functions or action policy functions.

First, assume that $\psi_0, \cdots, \psi_K : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ is a set of basis functions and that the action-value function is approximated using their linear combination. Now, solving the Bellman optimality equation is reduced to finding parameters $\theta_0, \cdots, \theta_K$. To estimate or update values of these $K > 1$ parameters, using a single observation in a unique and specified way is not possible. Therefore, we use at least K observations. In other words, we operate in the setting of batch-mode or offline RL. Assume that for each time-step t, we have samples from N trajectories. Using a squared loss function, coefficients θ_k can be found by minimizing the following loss function:

$$\mathcal{L}_t(\theta) = \sum_{n=1}^{N} \left(R_t(s_n, a_n) + \gamma \max_{a' \in \mathcal{A}} Q^*_{t+1}(s', a') - \sum_{k=0}^{K} \theta_k \psi_k(s, a) \right)^2.$$

This method is known as the fitted Q iteration (FQI). In the above equation, calculating $\max_{a' \in \mathcal{A}} Q^*_{t+1}(s', a')$ is not trivial. There are several ways of finding it. One way is to use the empirical max that was observed. This will result in an overestimation of Q function due to the Jensen inequality. Another possibility is a method known as double Q-learning proposed by Hasselt [2010]. This method, although useful, will result in an underestimation of Q function. A third possibility is to analytically or numerically, calculate the maximum of $Q(s, a)$ over the next-step action a'. This is possible in settings when $Q(s, a)$ has a specific parametric form, such as linear architectures.

Secondly, in deep reinforcement learning, we represent the Q values as a function $\hat{q}(s, a, \mathbf{w})$ where $\mathbf{w}$ are parameters of the function(e.g., a neural nets weights and biases). In this approximate setting, our update rule becomes

$$\mathbf{w} = \mathbf{w} + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} \hat{q}(s', a', \mathbf{w}) - \hat{q}(s, a, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(s, a, \mathbf{w}).$$

Meaning that we want to minimize the following loss function:

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{s,a,r,s'} \left[r + \gamma \max_{a' \in \mathcal{A}} \hat{q}(s', a', \mathbf{w}) - \hat{q}(s, a, \mathbf{w}) \right]^2.$$

A good example of using deep reinforcement learning is Mnih et al. [2015].

4. Dixon (2020) Machine Learning in Finance From Theory to Practice

This review is based on the tenth chapter of Dixon et al. [2020].

4.1. Abstract

In the tenth chapter of Dixon et al. [2020], the authors use reinforcement learning(RL) to solve three different classical problems in finance; option pricing, portfolio optimization and wealth management. These three problems are first formulated using RL language, and are then solved with the appropriate methods.

4.2. Introduction

In this chapter, several applications of RL in finance are discussed. First, the QLBS model is presented for the problem of option pricing. This model uses batch-mode Q-learning and produces a distribution-free discrete-time approach in option pricing and hedging of options. In the special case when transaction costs and market impact is neglected, this approach is semi-analytical and only requires solving linear matrix equations. After this, another RL method named G-learning is introduced. G-learning is a probabilistic extension of the classical Q-learning algorithm which is then used in solving two problems: 1. portfolio optimization with multiple assets, 2. wealth management. These two problems differ in that wealth management contains extra controls in the form of intermediate cash insertion/withdrawal. When the reward function is quadratic, G-learning results in a semi-analytical solution to these problems.

4.3. Option Pricing

4.3.1. Black-Scholes-Merton (BSM) model

The BSM model originally used for European call and put options. Let us first define what these two options are:
European put option: A contract that gives the holder, the right(but not the obligation) to sell some amount of the underlying asset at an agreed upon price in the future.
European call option: A contract that gives the holder, the right(but not the obligation) to buy some amount of the underlying asset at an agreed upon price in the future.
These options are two of the most popular types of financial derivatives. The BSM model utilizes a *relative value* approach to asset pricing, meaning that it prices assets in terms of other tradable assets. This method for options is known as *dynamic option replication* and is based on the observation that an options payoff depends only on the price of the underlying asset at the expiry time of the option. This means that the options can be simulated using a portfolio made of the underlying asset and cash, called the *hedge portfolio*. Therefore, the goal of dynamic replication is to mimic the option using the hedge portfolio as closely as possible. This is done by dynamically hedging the portfolio between the stock and cash in the right way and, as a consequence, eliminating the risk. Doing this in a continuous time setting will result in the celebrated Black-Scholes partial differential equation, which will gives an exact price for the option. In practice, rebalancing the hedge portfolio happens at finite frequency; furthermore, frequent rebalancing is not possible due to transaction costs. Thus, perfect hedging is usually not possible and a hedged option position contains some mis-hedging risk which should be accounted when pricing the option. In such discrete time settings, option pricing can be thought of as a problem of risk minimization by hedging, in a sequential decision making process. Using RL to solve this problem has two sides: First, if a model for the stock price dynamics is selected, model-based RL methods can be used. Second, we can use model-free RL and not use a stock price dynamics model at all.

To introduce a discrete-time version of BSM model, let us take the view of a seller of a European put option with maturity T and a final payoff of $H_T(S_T)$ at maturity. To hedge the option, the seller uses the revenue from selling the option to set up a replicating (hedge) portfolio Π_t which is made of the stock S_t and a risk-free bank deposit B_t. Let u_t denote the amount of the stock we own at time t. The value of this portfolio at time $t < T$ is therefore,

$$\Pi_t = u_t S_t + B_t.$$

At time T, the option is closed, therefore, we should set $u_T = 0$ and because the portfolio is trying to replicate the option, $\Pi_T = B_T = H(S_T)$. Here, $H(S_T) = (K - S_t)_+$ is the final payoff to the option buyer, when the agreed upon price in the option is K. We use the following relation, which imposes a self-financing condition on the portfolio to compute the amount of cash that needs to be in bank, at time $t < T$:

$$u_t S_{t+1} + e^{r\Delta t} B_t = u_{t+1} S_{t+1} + B_{t+1}.$$

Solving this equation for B_t gives

$$B_t = e^{-r\Delta t}[B_{t+1} + (u_{t+1} - u_t)S_{t+1}], \quad t = T-1, \cdots, 0.$$

Using this and the relation between B_t, Π_t, we derive

$$\Pi_t = e^{-r\Delta t}[\Pi_{t+1} - u_t \Delta S_t], \text{ where } \Delta S_t = S_{t+1} - e^{r\Delta t} S_t.$$

(In all equations, r is the risk-free interest rate)
Because B_t, Π_t depend on their future values, B_0, Π_0 are random variables. For a given hedging strategy $\{u_t\}_{t=0}^T$, they can be estimated using a Monte Carlo simulation and evaluating them backwards in time. The option price can then be set to the expected value of Π_0 plus some premium for risk. Note that the option price will be dependent on the hedging strategy, $\{u_t\}_{t=0}^T$. Hence, the need to compute the optimal hedging strategy arises.

Similar to portfolio evaluation, optimal hedges $\{u_t\}_{t=0}^T$, are computed by going backward in time, from $t = T$. However, Because the future is unknown, all calculations are based on partial information $\mathcal{F}_t$ available at time t. The optimal hedge $u^*(S_t)$ is found by minimizing variance of Π_t when conditioned on $\mathcal{F}_t$:

$$u_t^*(S_t) = \arg\min_u \text{Var}[\Pi_t | \mathcal{F}_t] = \arg\min_u \text{Var}[\Pi_{t+1} - u_t \Delta S_t | \mathcal{F}_t], \quad t = T-1, \cdots, 0$$

The optimal hedge can be found analytically in terms of Π_{t+1}, S_{t+1}. Therefore, if transition probabilities are known, one can derive the optimal hedge strategy.

We can also define the *fair price* $\hat{C}_t$, as the expected value of hedge portfolio at time t; $\hat{C}_t = \mathbb{E}_t[\Pi_t|\mathcal{F}_t]$. By replacing Π_t with $\Pi_{t+1} - u_t \Delta S_t$ and manipulating the expectation, we can obtain a recursive formula for $\hat{C}_t$:

$$\hat{C}_t = \mathbb{E}_t[e^{-r\Delta t}\hat{C}_{t+1}|\mathcal{F}_t] - u_t(S_t)\mathbb{E}_t[\Delta S_t|\mathcal{F}_t].$$

If we find the optimal hedge using $\hat{C}_t$ and substitute the result in the above equation, we obtain a recursive relation for solving $\hat{C}_t$ that is not dependent on the policy:

$$\hat{C}_t = e^{-r\Delta t}\mathbb{E}^{\hat{\mathbb{Q}}}[\hat{C}_{t+1}|\mathcal{F}_t],$$

where $\hat{\mathbb{Q}}$ is a signed measure with transition probabilities

$$\tilde{q}(S_{t+1}|S_t) = p(S_{t+1}|S_t)\left[1 - \frac{(\Delta S_t - \mathbb{E}_t[\Delta S_t])\mathbb{E}_t[\Delta S_t]}{\mathrm{Var}(\Delta S_t|\mathcal{F}_t)}\right],$$

and where $p(S_{t+1}|S_t)$ are transition probabilities under the physical measure $\mathbb{P}$.

Notice that the fair price is not an acceptable price for a seller of the option. This is because there is a risk that at some time in the future, the bank account B_t may be exhausted and the seller might have to add cash to the hedge portfolio. Therefore, the final price of the option is the fair price plus some risk premium. One possible way of defining this risk premium, is by using the cumulative expected discounted variance of the hedge portfolio along all time-steps, with a risk aversion parameter λ:

$$C_0^{(\mathrm{ask})}(S,u) = \mathbb{E}_0\left[\Pi_0 + \lambda\sum_{t=0}^{T} e^{-rt}\mathrm{Var}[\Pi_t|\mathcal{F}_t]\middle| S_0 = s, u_0 = u\right].$$

In order to stay consistent with RL notations, note that the problem of minimizing the appropriate option price $C^{(\mathrm{ask})}$ is equivalent to maximizing $V_t = -C_0^{(\mathrm{ask})}$. Therefore, in the next section we consider the equivalent problem of maximizing V_t:

$$V_t(S_t) = \mathbb{E}_t\left[-\Pi_t - \lambda\sum_{t'=t}^{T} e^{-r(t'-t)}\mathrm{Var}[\Pi_{t'}|\mathcal{F}_{t'}]\middle|\mathcal{F}_t\right].$$

The discrete-time framework that was presented, will lead to the original continuous-time BSM model in the limit $\Delta t \to 0$, if the price dynamics under the physical measure $\mathbb{P}$ is modeled by a geometric Brownian motion with drift μ and volatility σ:

$$\frac{\mathrm{d}S_t}{S_t} = \mu\mathrm{d}t + \sigma\mathrm{d}W_t,$$

where W_t is a standard Brownian motion. Thus, if the stock prices are log-normal we can derive the hedging and pricing formulas of the original BSM model by taking the limit $\Delta t \to 0$, and the celebrated Black-Scholes equation will be obtained:

$$\frac{\partial C_t}{\partial t} + rS_t\frac{\partial C_t}{\partial S_t} + \frac{1}{2}\sigma^2 S_t^2\frac{\partial^2 C_t}{\partial S_t^2} - rC_t = 0.$$

4.3.2. QLBS Model

First, let us reformulate and generalize the discrete-time BSM model, that was explored above, using the MDP framework. Because stock price dynamics usually involve a deterministic drift term, it is a good idea to use a stationary(non-drifting) variable that is obtained by transforming S_t. To this end, for a given stock price S_t, we define a new variable

$$X_t = -(\mu - \frac{\sigma^2}{2})t + \log S_t.$$

This transformation is noticeably useful when S_t is a geometric Brownian motion, as is in the BSM model. In this case, X_t becomes a standard Brownian motion. This also means that X_t will be a martingale(i.e. $\mathbb{E}[\mathrm{d}X_t] = 0$) which can be useful in numerical lattice approximations. Of course, in general if S_t is not log-normal, X_t might not be a martingale, but this transformation is inherently useful for separating the non-stationarity from the price variables. Also, we will

use $a_t = a_t(X_t)$ to denote the actions(policy) given time-homogeneous variable X_t. Actions, $u_t = u_t(S_t)$, given stock prices are then obtained by a simple transformation

$$u_t(S_t) = a_t(X_t(S_t)) = a_t(\log S_t - (\mu - \frac{\sigma^2}{2})t).$$

Individual hedging decisions given a realization of a random state X_t is shown by $a_t(x_t)$, while the hedging strategy that applies for any state X_t, is denoted by $\pi(t, X_t)$, which is a time-dependent policy. Here, we consider deterministic policies which map time t and the current state X_t to an action $a_t \in \mathcal{A}$.

Let us start by rephrasing the problem of maximizing V_t(which was defined as $-C_0^{\text{ask}}$) so that it uses the newly introduced state variables:

$$V_t^{\pi}(X_t) = \mathbb{E}_t \left[-\Pi_t(X_t) - \lambda \sum_{t'=t}^{T} e^{-r(t'-t)} \text{Var}[\Pi_{t'}(X_{t'})|\mathcal{F}_{t'}] \middle| \mathcal{F}_t \right]$$

We can separate the first term in the sigma($t' = t$), and rewrite the sigma in terms of V_{t+1}. Notice that

$$-\lambda \sum_{t'=t+1}^{T} e^{-r(t'-t)} \text{Var}[\Pi_{t'}|\mathcal{F}_{t'}] = \gamma V_{t+1} + \mathbb{E}_{t+1}[\Pi_{t+1}], \gamma := e^{-r\Delta t}.$$

Now, by substituting this into the equation for V_t^{π}, we obtain the *Bellman equation for the QLBS model*:

$$V_t^{\pi}(X_t) = \mathbb{E}_t^{\pi}[R(X_t, a_t, X_{t+1}) + \gamma V_{t+1}^{\pi}(X_{t+1}],$$

where the time-dependent random reward is defined as

$$R(X_t, a_t, X_{t+1}) = \gamma a_t \Delta S_t(X_t, X_{t+1}) - \lambda \text{Var}[\Pi_t|\mathcal{F}_t] = \gamma a_t \Delta S_t(X_t, X_{t+1}) - \lambda \gamma^2 \mathbb{E}_t[\hat{\Pi}_{t+1}^2 - 2a_t \Delta \hat{S}_t \Pi_{t+1} + a_t^2 (\Delta \hat{S}_t)^2],$$

where $\hat{\Pi}_{t+1} := \Pi_{t+1} - \bar{\Pi}_{t+1}$ and where $\bar{\Pi}_{t+1}$ is the sample mean of all values of Π_{t+1}, and similarly for $\Delta \hat{S}_t$. Also notice that $R_T = -\lambda \text{Var}[\Pi_T]$, where Π_t is determined by the terminal condition that was mention earlier. It can be seen that the expected reward R_t is quadratic in the action variable a_t. The first term in the reward formula is the return from the portfolio while the second term penalizes for quadratic risk. This means that the immediate reward includes the risk factor. Hence, our approach is a risk-sensitive RL.

The action value function can also be defined. It is similar to the value function except that it is conditioned on both the current state and the immediate action:

$$Q_t^{\pi}(x, a) = \mathbb{E}_t[-\Pi_t(X_t)|X_t = x, a_t = a] - \lambda \mathbb{E}_t^{\pi} \left[\sum_{t'=t}^{T} e^{-r(t'-t)} \text{Var}\left[\Pi_{t'}\left(X_{t'}\right) \mid \mathcal{F}_{t'}\right] \middle| X_t = x, a_t = a \right]$$

The optimal policy $\pi_t^*(\cdot|X_t)$ is defined as the policy that maximizes the value(or action value) function. In this setting, the Bellman optimality equation for action value function is

$$Q_t^*(x, a) = \mathbb{E}_t \left[R_t\left(X_t, a_t, X_{t+1}\right) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*\left(X_{t+1}, a_{t+1}\right) \mid X_t = x, a_t = a \right],$$

for $t = 0, \cdots, T-1$, and with a terminal condition

$$Q_T^*\left(X_T, a_T = 0\right) = -\Pi_T\left(X_T\right) - \lambda \,\text{Var}\left[\Pi_T\left(X_T\right)\right].$$

Notice that here, Var[] is the variance with respect to all Monte Carlo paths that terminate in a given state.

If we assume that there is no feedback-loop(as is an assumption of BSM model), then after substituting the reward function in the Bellman optimality equation for action value function, it can be seen that the action value function is quadratic in the action variable a_t. Using this, the optimal action(hedge) $a_t^*(S_t)$ that maximizes $Q_t^*(S_t, a_t)$ can be computed analytically:

$$a_t^*\left(X_t\right) = \frac{\mathbb{E}_t\left[\Delta \hat{S}_t \hat{\Pi}_{t+1} + \frac{1}{2\gamma\lambda} \Delta S_t\right]}{\mathbb{E}_t\left[\left(\Delta \hat{S}_t\right)^2\right]}.$$

This can then be plugged into the Bellman optimality equation to produce a recursive formula for calculating the optimal action value function. Basically, we can start from $t = T - 1$ and go backwards in time to $t = 0$. At each time step, finding the optimal action is a convex optimization problem which is solved analytically. This optimal action is then substituted into the Bellman optimality equation to calculate the action value function at the current time step. This means that when the dynamics are known, QLSB method can be used to find both the option value and the optimal hedge.

DP and Monte Carlo Sampling: In practice, solving the recursive formulas that were given above is done in a Monte Carlo setting, where we use N simulated paths for the state variable X_t. We also choose a set of basis functions $\{\Phi_n(x)\}$ and write the optimal action and Q-function as a time-dependent linear combination of these functions:

$$a_t^*(X_t) = \sum_{n=1}^{M} \phi_{nt}\Phi_n(X_t)\ , \quad Q_t^*(X_t, a_t^*) = \sum_{n=1}^{M} \omega_{nt}\Phi_n(X_t).$$

In order to find the coefficients ϕ_{nt}, We use the Bellman optimality equation. We replace the expected values by MC estimates, drop all a_t independent terms, and substitute a_t with its expansion. After all this, we are left with the following function that should be minimized:

$$G_t(\phi) = \sum_{k=1}^{N_{MC}} \left(-\sum_n \phi_{nt}\Phi_n\left(X_t^k\right)\Delta S_t^k + \gamma\lambda\left(\hat{\Pi}_{t+1}^k - \sum_n \phi_{nt}\Phi_n\left(X_t^k\right)\Delta\hat{S}_t^k\right)^2\right)$$

Once we have the coefficients ϕ_{nt}, we find the coefficients ω_{nt} by solving the following least square optimization problem:

$$F_t(\omega) = \sum_{k=1}^{N_{MC}} \left(R_t\left(X_t, a_t^*, X_{t+1}\right) + \gamma \max_{a_{t+1}\in\mathcal{A}} Q_{t+1}^*\left(X_{t+1}, a_{t+1}\right) - \sum_n^M \omega_{nt}\Phi_n\left(X_t^k\right)\right)^2$$

RL solution: Now, we shift our attention to a scenario in which transition probabilities and reward functions are not known. Furthermore, the state-action space is considered to be continuous. In this setting, we use RL methods to solve the QLBS model.

We assume that we only have access to some historically collected data(i.e., we work in the setting of batch-mode). The data is a set of N_{MC} trajectories, for the underlying stock S_t(and from that X_t), hedge position a_t, immediate reward R_t and the next time value X_{t+1}.

$$\mathcal{F}_t^{(n)} = \left\{\left(X_t^{(n)}, a_t^{(n)}, R_t^{(n)}, X_{t+1}^{(n)}\right)\right\}_{t=0}^{T-1}, \quad n = 1, \ldots, N_{MC}$$

We use the fitted Q iteration(FQI) method to solve this RL problem(Ernst et al. [2005], Murphy [2005]). As a first step, we choose linear architectures to represent functions of interest. We use the same set of basis functions $\{\Phi_n(x)\}$ as before. As we mentioned earlier, $Q_t^*(X_t, a_t)$ is a quadratic function of a_t. Hence we can write

$$Q_t^*\left(X_t, a_t\right) = \left(1, a_t, \frac{1}{2}a_t^2\right)\begin{pmatrix} W_{11}(t) & W_{12}(t)\cdots W_{1M}(t) \\ W_{21}(t) & W_{22}(t)\cdots W_{2M}(t) \\ W_{31}(t) & W_{32}(t)\cdots W_{3M}(t)\end{pmatrix}\begin{pmatrix}\Phi_1\left(X_t\right) \\ \vdots \\ \Phi_M\left(X_t\right)\end{pmatrix} := \mathbf{A}_t^T\mathbf{W}_t\mathbf{\Phi}(X_t) := \mathbf{A}_t^T\mathbf{U}_W(t, X_t)$$

We can further rearrange them to convert it into a product of a parameter vector and a vector that is dependent on both the state and the action:

$$Q_t^*(X_t, a_t) = \mathbf{W}_t\mathbf{\Psi}(X_t, a_t).$$

Now, we have to find time-dependent coefficients $\mathbf{W}_t$. These can be computed recursively, backward in time. We again use the Bellman optimality equation and interpret it as a regression. Finally, the coefficients can be found by solving the following least square optimization problem:

$$\mathcal{L}_t\left(\mathbf{W}_t\right) = \sum_{k=1}^{N_{MC}} \left(R_t\left(X_t, a_t, X_{t+1}\right) + \gamma \max_{a_{t+1}\in\mathcal{A}} Q_{t+1}^*\left(X_{t+1}, a_{t+1}\right) - \mathbf{W}_t\Psi\left(X_t, a_t\right)\right)^2.$$

4.4. Dynamic Portfolio Optimization

In this section, a multi asset investment portfolio(specifically, stock portfolio) in a multi-dimensional setting is considered. When using RL in such scenarios with high number of dimensions, one should be wary of two issues. First, the computational cost and the problem of under-sampling due to the curse of dimensionality. Second, the problem of noisy data. When using samples to estimate various parameters in such high-dimensional spaces, the model itself may become noisy. One possible solution to this problem of noise is using probabilistic policies as they can capture this noise better than deterministic policies. This is one of the reasons that in this section, our method of choice is a probabilistic extension of Q-learning called *G-learning*.

4.4.1. Terminology and Foundations

The notation used here is adopted from Boyd et al. [2017]. The dollar values of positions in n assets are denoted as a vector $\mathbf{x}_t$. $(x_t)_i$ denotes the dollar value of asset i in the beginning of period t. An investment portfolio also includes a risk-free cash bank account b_t with risk-free interest rate r_f. $\mathbf{p}_t$ denotes the vector of mean of bid and ask prices of assets at the beginning of period t. Trades $\mathbf{u}_t$ are made at the beginning of period t, so that asset values immediately after trades are

$$\mathbf{x}_t^+ = \mathbf{x}_t + \mathbf{u}_t.$$

Total portfolio value at time t, is denoted by v_t:

$$v_t = \mathbb{1}^T \mathbf{x}_t + b_t$$

Similarly, the post-trade portfolio value is

$$v_t^+ = \mathbb{1}^T \mathbf{x}_t^+ + b_t^+ = v_t + \mathbb{1}^T \mathbf{u}_t + b_t^+ - b_t.$$

We also impose a self-financing constraint, similar to the one in the previous section:

$$\mathbb{1}^T \mathbf{u}_t + b_t^+ - b_t = 0$$

This means that portfolio value remains the same after an immediate rebalancing of wealth between the stock and cash. In other words, $v_t^+ = v_t$.
The return of asset i over period t, is defined as

$$(\mathbf{r}_t)_i = \frac{(\mathbf{p}_{t+1})_i - (\mathbf{p}_t)_i}{(\mathbf{p}_t)_i}.$$

Asset positions and portfolio value at the next time period are

$$\mathbf{x}_{t+1} = \mathbf{x}_t^+ + \mathbf{r}_t \circ \mathbf{x}_t^+,$$

$$v_{t+1} = (1 + \mathbf{r}_t)^T (\mathbf{x}_t + \mathbf{u}_t).$$

Given a vector of returns $\mathbf{r}_t$, the *change of portfolio value in excess of a risk-free growth* is given by

$$\Delta v_t = v_{t+1} - (1 + r_f) v_t = (\mathbf{r}_t - r_f \mathbb{1})^T (\mathbf{x}_t + \mathbf{u}_t).$$

We denote the investment horizon by T. We must give a proper terminal condition at this horizon. One possible choice, is to use a benchmark portfolio and for the terminal condition, require that at time T, all stock positions should be equal to the observed weights of stock in that benchmark. Another possibility for the terminal condition is to set $\mathbf{x}_t = 0$, meaning that all stock positions should be converted to cash at the horizon.

Asset returns model: One-period excess asset returns are assumed to follow a linear specification:

$$\mathbf{r}_t - r_f \mathbb{1} = \mathbf{W} \mathbf{z}_t - \mathbf{M}^T \mathbf{u}_t + \varepsilon_t,$$

where $\mathbf{z}_t$ is a vector of predictors with factor loading matrix $\mathbf{W}$, $\mathbf{M}$ is the matrix of permanent market impacts with a linear impact specification, and ε_t is a vector of residuals with

$$\mathbb{E}[\varepsilon_t] = 0, \mathbb{V}[\varepsilon_t] = \Sigma_r.$$

According to this equation, the next step stock prices are driven by external signals $\mathbf{z}_t$, action variables $\mathbf{u}_t$ and uncontrollable noise ε_t.

Signal dynamics: Generally, a predictor z_t should be considered as a signal in our model only if it has three conditions:1. it correlates with equity returns, 2. is predictable(e.g., is a mean reverting process), 3. its characteristic times τ are larger than the time step Δt. The last condition is mentioned because if $\tau \ll \Delta t$, then the effects of z_t can be considered as a stationary white noise, which is already modeled. As such, for different tasks, different predictors should be considered. For example, for monthly or quarterly trading, macroeconomic variables(e.g., inflation rate) may be used whereas for intra-day trading, variables derived from the limit order book are probably more useful. Similar to what has been done in Gârleanu and Pedersen [2013], signals $\mathbf{z}_t$ are assumed to be a multivariate mean-reverting Ornstein-Uhlenbeck (OU) process:

$$\mathbf{z}_{t+1} = (\mathbf{I} - \Phi) \circ \mathbf{z}_t + \varepsilon_t^z,$$

where $\varepsilon_t^z \sim \mathcal{N}(0, \Sigma_z)$ is the noise term and Φ is a diagonal matrix of mean-reversion rates.
Finally, the state variable is defined as a vector y_t that contains both $\mathbf{x}_i$s and $\mathbf{z}_i$s:

$$y_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{z}_t \end{bmatrix}$$

Reward function: If we consider a friction-free market(no charge for making trades), the instantaneous reward from taking an action(i.e., making a trade $\mathbf{u}_t$) can be computed by considering the change of price of each stock and the amount of each stock that we own. This is simple as we already have an asset return model. In order to make this more realistic, some negative reward due to market impact and transaction fees should be added. Furthermore, another negative reward should be added to compensate for the risk in the newly created portfolio at time $t + 1$. Variance of instantaneous reward, conditioned on the new state, multiplied by a risk aversion parameter is a good candidate. The final risk- and cost-adjusted instantaneous reward function can be written as

$$R_t(y_t, \mathbf{u}_t) = R_t^{(0)}(y_t, \mathbf{u}_t) + R_t^{(\text{risk})}(y_t, \mathbf{u}_t) + R_t^{(\text{impact})}(y_t, \mathbf{u}_t) + R_t^{(\text{fee})}(y_t, \mathbf{u}_t).$$

Notice that the expected value of this reward function is a quadratic form of its inputs, because we can write it as

$$\hat{R}(y_t, \mathbf{u}_t) = y_t^T \mathbf{R}_{yy} y_t + a_t^T \mathbf{R}_{aa} a_t + a_t^T \mathbf{R}_{ay} y_t + a_t^T \mathbf{R}_a.$$

where $\mathbf{R}_{yy}$, $\mathbf{R}_{aa}$, $\mathbf{R}_{ay}$, $\mathbf{R}_a$ are parameter matrices and a_t is a non-negative vector obtained from $\mathbf{u}_t$.

$$a_t = \begin{pmatrix} \mathbf{u}_t^+ \\ \mathbf{u}_t^- \end{pmatrix}$$

Multi-period portfolio optimization: Now that all elements of the MDP model are described, we can formally define the *multi-period risk- and cost-adjusted reward maximization problem* as

$$\begin{aligned} &\text{maximize } \mathbb{E}_t[\sum_{t'=t}^{T-1} \gamma^{t'-t} \hat{R}_{t'}(y_{t'}, a_{t'})] \\ &\text{w.r.t. } a_t = \begin{pmatrix} \mathbf{u}_t^+ \\ \mathbf{u}_t^- \end{pmatrix} \geqslant 0 \\ &\text{subject to } \mathbf{x}_t + \mathbf{u}_t^+ - \mathbf{u}_t^- \geqslant 0. \end{aligned}$$

The last constraint ensures that our portfolio remains long-only, but can be replaced by other conditions. Regardless, this problem is a convex optimization problem with constraints and can be solved numerically in an efficient way(Boyd et al. [2017]).

4.4.2. Choosing the Policy

In Q-learning and most other methods of RL that were discussed, the optimal policy is assumed to be deterministic. But real market data may be sub-optimal or noisy. In this setting, it is preferable to use stochastic policies which are

described as probability distributions $\pi(a_t|y_t)$. Therefore, instead of maximizing with respect to the optimal policy a_t, we reformulate the problem to maximize over probability distributions $\pi(a_t|y_t)$:

$$\text{maximize } \mathbb{E}_{q_\pi}[\sum_{t'=t}^{T-1} \gamma^{t'-t} \hat{R}_{t'}(y_{t'}, a_{t'})]$$

$$\text{w.r.t. } q_\pi(\bar{x}, \bar{a}|y_0) = \pi(a_0|y_0) \prod_{t=1}^{T-1} \pi(a_t|y_t) P(y_{t+1}|y_t, a_t)$$

$$\text{subject to } \int \mathrm{d}a_t \pi(a_t|y_t) = 1.$$

Here, $\mathbb{E}_{q_\pi}$ is expectation with respect to path probabilities defined by the second equation above.
Notice that although by trying to maximize the expected cumulative reward, we are working in the setting of a normal MDP, because of the way we defined the reward function, the risk factor is also incorporated in the model. Such risk adjustment method that use a one-step variance penalties, first appeared in Gosavi [2010] in a non-financial setting and then in Halperin [2020] to solve the option pricing problem.

We also use a probabilistic *reference* (or *prior*) policy $\pi_0(a_t|y_t)$. Here, a simple Gaussian reference policy is used:

$$\pi_0\left(a_t \mid y_t\right) = \frac{1}{\sqrt{(2\pi)^N \left|\Sigma_p\right|}} \exp\left(-\frac{1}{2}\left(a_t - \hat{a}\left(y_t\right)\right)^T \Sigma_p^{-1}\left(a_t - \hat{a}\left(y_t\right)\right)\right),$$

where $\hat{a}(y_t)$ is a deterministic policy that is a linear function of the state variable:

$$\hat{a}(y_t) = \hat{\mathbf{A}}_0 + \hat{\mathbf{A}}_1 y_t.$$

We can further simplify thing by choosing $\hat{\mathbf{A}}_0 = \hat{a}_0 \mathbb{1}_{|A|}$ and $\hat{\mathbf{A}}_1 = \hat{a}_1 \mathbb{1}_{|A|\times|A|}$ where scalars $\hat{a}_0$, $\hat{a}_1$ are thought of as hyperparameters. Similarly, the covariance matrix Σ_p for the prior policy can be a simple matrix with constant correlations and variances, ρ_p, σ_p. It can be proved that an optimal policy has the same form as this prior policy, with updated parameters $\hat{\mathbf{A}}_0$, $\hat{\mathbf{A}}_1$, Σ_p, which can be computed iteratively. Furthermore, we can even set $\hat{\mathbf{A}}_1 = \hat{\mathbf{A}}_1^{(0)} = 0$ in the prior, without losing the linear dependence on y_t! Choosing such state-independent prior $\pi_0(a_t|y_t) = \pi_0(a_t)$, can further reduce the number of hyperparameters in our model to only $\hat{a}_0$, ρ_a, σ_p.

4.4.3. Entropy-based Regularization

For a learned policy $\pi(a_t|y_t)$, the *information cost* relative to the refrence policy $\pi_0(a_t|y_t)$ is defined as

$$g^\pi(y, a) = \log \frac{\pi(a_t|y_t)}{\pi_0(a_t|y_t)}$$

(Fox et al. [2015]). Its expectation with respect to the policy π is the Kullback-Leibler(KL) divergence of $\pi(\cdot|y)$, $\pi_0(\cdot|y)$:

$$\mathbb{E}_\pi[g^\pi(y, a)|y_t] = KL[\pi||\pi_0](y_t) = \sum_{a_t} \pi(a_t|y_t) \log \frac{\pi(a_t|y_t)}{\pi_0(a_t|y_t)}.$$

The *total discounted information cost* for a trajectory is defined as:

$$I^\pi(y) = \sum_{t'=t}^{T-1} \gamma^{t'-t} \mathbb{E}[g^\pi(y_{t'}, a_{t'})|y_t = y].$$

We can now define the *free energy* function $F_t^\pi(y_t)$ which extends the traditional value function in RL, by augmenting the information cost within it:

$$F_t^\pi(y_t) := V_t^\pi(y_t) - \frac{1}{\beta} I^\pi(y_t) = \sum_{t'=t}^{T} \gamma^{t'-t} \mathbb{E}\left[\hat{R}_{t'}(y_{t'}, a_{t'}) - \frac{1}{\beta} g^\pi(y_{t'}, a_{t'})\right].$$

In the above definition, β is known as *inverse temperature* parameter which controls a tradeoff between reward optimization and proximity to reference policy. $F_t^\pi(y_t)$ is an entropy-regularized value function where the amount of regularization can be adjusted to the level of noise in the data. A Bellman equation for this function is

$$F_t^\pi(y_t) = \mathbb{E}_{a|y}\left[\hat{R}_t(y_t, a_t) - \frac{1}{\beta} g^\pi(y_t, a_t) + \gamma \mathbb{E}_{t,a}[F_{t+1}^\pi(y_{t+1})]\right],$$

with a terminal condition

$$F_T^\pi(y_T) = \hat{R}_T(y_T, a_T)|_{a_T = -x_{T-1}}.$$

Similar to regularized value function, the *state-action free energy* function $G^\pi(x, a)$ can be defined. This G function can be thought of as a regularized Q function used frequently in RL:

$$G_t^\pi(y_t, a_t) = \hat{R}_t\left(y_t, a_t\right) + \gamma \mathbb{E}\left[F_{t+1}^\pi\left(y_{t+1}\right) \mid y_t, a_t\right] = \mathbb{E}_{t,a}\left[\sum_{t'=t}^{T} \gamma^{t'-t}\left(\hat{R}(y_{t'}, a_{t'}) - \frac{1}{\beta} g^\pi(y_{t'}, a_{t'})\right)\right].$$

Similar to value and action-value function(and their optimal counterparts) which are related to one another, F and G function are also related to each other:

$$F_t^\pi\left(y_t\right) = \sum_{a_t} \pi\left(a_t \mid y_t\right)\left[G_t^\pi\left(y_t, \mathbf{a}_t\right) - \frac{1}{\beta} \log \frac{\pi\left(a_t \mid y_t\right)}{\pi_0\left(a_t \mid y_t\right)}\right].$$

The following distribution maximizes this functional

$$\pi\left(a_t \mid y_t\right) = \frac{1}{Z_t} \pi_0\left(a_t \mid y_t\right) e^{\beta G_t^\pi(y_t, a_t)}$$
$$Z_t = \sum_{a_t} \pi_0\left(a_t \mid y_t\right) e^{\beta G_t^\pi(y_t, a_t)}$$

The free energy evaluated at this optimal solution is

$$F_t^\pi\left(y_t\right) = \frac{1}{\beta} \log Z_t = \frac{1}{\beta} \log \sum_{a_t} \pi_0\left(a_t \mid y_t\right) e^{\beta G_t^\pi(y_t, a_t)},$$

and the optimal policy can be written as

$$\pi\left(a_t \mid y_t\right) = \pi_0\left(a_t \mid y_t\right) e^{\beta\left(G_t^\pi(y_t, a_t) - F_t^\pi(y_t)\right)}.$$

These two equations along with the equation that was given for the G function in terms of F function and terminal conditions for both these functions, can be used to form a consistent system of equations that can be solved iteratively backwards in time.

4.4.4. G-learning and F-learning

We now try to find the optimal G and F functions using samples of data. If we substitute the optimal solution into the equation for G function, we get

$$G_t^\pi(y, a) = \hat{R}(y_t, a_t) + \mathbb{E}_{t,a}\left[\frac{\gamma}{\beta} \sum_{a_{t+1}} \pi_0(a_{t+1}|y_{t+1}) e^{\beta G_{t+1}^\pi(y_{t+1}, a_{t+1})}\right].$$

This equation is a soft relaxation of Bellman optimality equation for the action-value function. Notice that as $\beta \to \infty$, the last term in the equation approximates the $\max(\cdot)$ operator. For $\beta < \infty$, this equation is used in *G-learning*: an off-policy, TD algorithm that generalizes Q-learning to noisy environments using an entropy-based regularization. The G-learning algorithm of Fox et al. [2015] uses a table to save all values of the G function. In order to make the algorithm compatible to our problem of portfolio optimization where the number of possible states is large, we use a linear architecture to approximate the G function.

Using the equations that were introduced when we modeled the asset returns, a state equation for the portfolio vector $\mathbf{x}_t$ can be derived as:

$$\mathbf{x}_{t+1} = (1 + r_f)(\mathbf{x}_t + \mathbf{u}_t) + \text{diag}(\mathbf{W}\mathbf{z}_t - \mathbf{M}\mathbf{u}_t)(\mathbf{x}_t + \mathbf{u}_t) + \varepsilon(\mathbf{x}_t, \mathbf{u}_t)$$

Recall that $\mathbf{M}$ was defined as a diagonal matrix with elements μ_i of market impacts. The above equation implies that the dynamics are non-linear in controls $\mathbf{u}_t$ due to market impact. We consider two cases:

Neglected Market Impact: First consider the simpler case where market impacts are neglected, i.e., $\mu_i = 0$. This makes the dynamics linear in $\mathbf{u}_t$. Also, recall that rewards were quadratic. In this setting, finding the optimal deterministic policy is a well known linear quadratic regulator(LQR) whose solution is known from control theory. Here, a probabilistic solution to this problem is presented.

In such situation, the dynamics can be written as:

$$\mathbf{x}_{t+1} = \mathbf{A}_t(\mathbf{x}_t + \mathbf{u}_t) + (\mathbf{x}_t + \mathbf{u}_t) \circ \varepsilon_t \text{ where } \mathbf{A}_t = \mathbf{A}(\mathbf{z}_t) = \text{diag}(1 + r_f + \mathbf{W}\mathbf{z}_t).$$

Also, transaction costs are assumed to be convex, $\eta \mathbf{u}_t^T \mathbf{C} \mathbf{u}_t$, where η is a transaction cost parameter and $\mathbf{C}$ is a matrix which can be taken to be a diagonal unit matrix. As such, the expected one-step reward will be given by

$$\hat{R}_t\left(\mathbf{x}_t, \mathbf{u}_t\right) = \left(\mathbf{x}_t + \mathbf{u}_t\right)^T \mathbf{W}\mathbf{z}_t - \lambda \left(\mathbf{x}_t + \mathbf{u}_t\right)^T \Sigma_r \left(\mathbf{x}_t + \mathbf{u}_t\right) - \eta \mathbf{u}_t^T \mathbf{C} \mathbf{u}_t.$$

In this setting, RL with G-learning can be solved semi-analytically by assuming a functional form of value function F_t as a quadratic form of $\mathbf{x}_t$, and finding the appropriate parameters. The algorithm then alternates between policy evaluation steps and policy improvement steps. Basically, policy optimization with the entropy-regularized LQR, is a Bayesian update of a prior distribution.

Non-zero Market Impact: If we do consider market parameters μ_i, the problem is no longer analytically tractable, because dynamics are non-linear. One possible solution, is to iteratively linearize the dynamics. For deterministic policies, an iterative quadratic regulator(IQR) can be used(Todorov and Li [2005]). There are also other methods for working with stochastic policies. One such method is explored in Halperin and Feldshteyn [2018].

4.5. Wealth Management

In the two previous sections, we explored two use cases of RL in finance, both of which relied on a self-financing assumption. In this section, another class of financial problems is considered in which this assumption does not hold. Financial planning and wealth management are two examples of this type of problem. The problem of *optimal consumption with an investment portfolio* is known as the *Merton consumption problem*, named after the work of Merton [1975]. Problems involving cash-injection can be thought of as a negative consumption problem, and are therefore included in this model. Specifically, we consider the problem of optimizing a defined contribution retirement plan.

4.5.1. Terminology and Foundations

For the most part, we use a similar notation as in the previous sections. T is the time horizon for our model. N assets are considered for investment, with $\mathbf{x}_t$ being the vector of dollar values of positions in different assets at time t. The first asset $n = 1$ is considered to be a risk-free bond, while other assets are risky, with random return $\mathbf{r}_t$ whose expected values are $\bar{\mathbf{r}}_t$. Σ_r is the $(N-1)\times(N-1)$ covariance matrix of returns. $\mathbf{u}_t$ is a part of the control variable that denotes the change in each position. c_t is the cash installment in the plan at time t. The pair $(c_t, \mathbf{u}_t)$ can be considered as the action variable. Furthermore, at each time-step t, $\hat{P}_{t+1}$ is a pre-specified target value of a portfolio, at time $t+1$. At each step, a penalty is obtained for under-performance relative to this target. Hence, the following reward function is proposed

$$R_t(\mathbf{x}_t, \mathbf{u}_t, c_t) = -c_t - \lambda \mathbb{E}_t\left[\left(\hat{P}_{t+1} - (1 + \mathbf{r}_t)(\mathbf{x}_t + \mathbf{u}_t)\right)_+\right] - \mathbf{u}_t^T \Omega \mathbf{u}_t,$$

where, the first term is due to cash installment of c_t, the second term is the expected negative reward from under-performance relative to target, and the third term approximates transaction costs by a convex functional with the parameter matrix Ω and plays a similar rule as an L_2 regularization.

Notice the presence of non-linearity in the form of $(\cdot)_+ = \max(0, \cdot)$, which may cause inconvenience. Also, note that decision variables are not independent as the total change in all positions is equal to the cash installment c_t:

$$\sum_{n=1}^{N} (\mathbf{u}_t)_n = c_t.$$

It is for these reasons that we use a modified version of this reward function in which the first term is replaced by $\sum_{n=1}^{N}(\mathbf{u}_t)_n$ and the non-linearity is eliminated by using a quadratic form:

$$R_t(\mathbf{x}_t, \mathbf{u}_t, c_t) = -\sum_{n=1}^{N}(\mathbf{u}_t)_n - \lambda \mathbb{E}_t\left[\left(\hat{P}_{t+1} - (1+\mathbf{r}_t)(\mathbf{x}_t+\mathbf{u}_t)\right)^2\right] - \mathbf{u}_t^T \Omega \mathbf{u}_t.$$

Although this makes the reward function quadratic in $\mathbf{u}_t$, it comes with its own drawback which is that this quadratic form is symmetric, meaning that it also penalizes us when $V_{t+1} = (1+\mathbf{r}_t)(\mathbf{x}_t+\mathbf{u}_t) \gg \hat{P}_{t+1}$. In order to solve this issue, we can choose target values $\hat{P}_{t+1}$ which are significantly higher than time-t expectation of next-period portfolio values. We can also use an alternative representation for this reward function. Let $\mathbf{r}_t = \bar{\mathbf{r}}_t + \tilde{\varepsilon}_t$, where $\tilde{\varepsilon}_t = (0, \varepsilon_t)$, where ε_t is an idiosyncratic noise with covariance Σ_r. Remember that the first asset is risk-free, therefore, $\bar{r}_0(t) = r_f$. Using this notation, the reward function can be written as:

$$R_t\left(\mathbf{x}_t, \mathbf{u}_t\right) = -\lambda \hat{P}_{t+1}^2 - \mathbf{u}_t^T \mathbb{1} + 2\lambda \hat{P}_{t+1}\left(\mathbf{x}_t+\mathbf{u}_t\right)^T\left(\mathbb{1}+\bar{\mathbf{r}}_t\right) - \lambda\left(\mathbf{x}_t+\mathbf{u}_t\right)^T \hat{\Sigma}_t\left(\mathbf{x}_t+\mathbf{u}_t\right) - \mathbf{u}_t^T\Omega\mathbf{u}_t,$$

where $\hat{\Sigma}_t$ is defined as

$$\hat{\Sigma}_t = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Sigma_r \end{bmatrix} + \left(1+\bar{\mathbf{r}}_t\right)\left(1+\bar{\mathbf{r}}_t\right)^T.$$

4.5.2. Optimization

The choice of a square loss reward function is convenient, as it allows one to construct optimal policies semi-analytically as we did in the previous sections. The main difference between the reward function in this problem and dynamic portfolio optimization, which was discussed in the previous section, is the presence of the first term, $-\lambda \hat{P}_{t+1}^2$. Notice that this is independent of states and actions, therefore, it does not impact the optimization. Similar to the previous section, a semi-analytic formulation of G-learning with Gaussian time-varying policies can be used to solve this problem. This is done by first considering the value function as a quadratic form of $\mathbf{x}_t$ and then finding the coefficients of this functional form backwards in time. Then, we can find the optimal action that maximizes the value function F, analytically. The optimal action turns out to be a linear function of state.

In the end, as was mentioned earlier, policy optimization for G-learning with quadratic rewards and Gaussian reference policy is a Bayesian update of parameters of prior distribution. Notice that after using G-learning to find the optimal actions $\mathbf{u}_t$, we can use the relation between $\mathbf{u}_t$ and c_t that was given before, to find the optimal amount of cash contribution at each time-step. In practice, we usually would like to add a constraint on the cash installment c_t. For example, we might want to impose an upper bound c_t such that $0 \leqslant c_t \leqslant c_{\max}$. These constraints can easily be added to the current framework by replacing the unconstrained least squares problem with a constrained one, which is solvable using convex optimization methods.

5. Dixon (2020) Machine Learning in Finance From Theory to Practice

This review is based on the eleventh chapter of Dixon et al. [2020].

5.1. Abstract

In this chapter, the problem of imitation learning is discussed. Several approaches and different methods for solving this problem are presented, and strengths and limitations of each one are examined.

5.2. Introduction

When applying reinforcement learning to different problems, one of the main challenges is to define a reward function. A reward function encodes the agents goals and desires and has a direct impact on the solution that is found. Essentially, by changing the reward function, one changes the whole objective of an agent in the environment. Choosing the right reward function is therefore, a delicate matter. In many cases, it is not easy to find what the reward function should be. In order to eliminate the need for a reward function, some alternative approaches to direct RL are proposed that do not require an explicit reward function. These methods are known as *learning form demonstrations* or *learning from behaviour*. As there is an abundance of behavioural data, these methods can be extremely useful in practice. However, learning the optimal policy by merely observing an agents behaviour in an environment is an

ill-posed problem, as we do not know what the goal toward which we should optimize, exactly is. It is therefore more convenient to focus on the problem of *imitation learning* (IL) in which we assume that the observations belong to an *expert* that follows an optimal (or nearly optimal) policy. In this scenario, we do not need to learn the experts intentions, and it suffices to only imitate his behaviour. This may seem similar to a supervised learning problem, and indeed supervised learning methods can be used to solve this problem, giving birth to an approach called *behavioural cloning*. The major problem with this approach is that it does not generalizes well to new, unseen states. Another problem is that policies learned by behavioural cloning are not portable to other environments. Both of these issues come from the fact that all information on environmental dynamics is lost when using such methods.

On the other hand, a reward function is portable as it does not depend on environmental dynamics while containing all necessary information about properties of the expert. It is for this reason that the idea of learning the reward function from demonstrations looks enticing. Methods that try to solve the imitation learning problem by finding the experts reward function are known as inverse reinforcement learning(IRL).

5.3. Inverse Reinforcement Learning

While reinforcement learning is the problem of learning from experiences, inverse reinforcement learning (IRL) can be thought of as learning from demonstrations. In the setting of an MDP, this is analogues to learning the optimal policy, without observing the rewards. This means that the agent must learn a policy by only observing a series of actions produced by another agent. This is called *imitation learning* and is similar to batch-mode RL except that the agent is not provided with rewards associated with each action. It is reasonable to assume that the policy that generated the observations, is optimal, or nearly optimal. Without access to the rewards, if the observations are not close to optimal(e.g., if they are random), it seems unrealistic to expect an agent to learn anything about the true goal or optimal behaviour. In this setting, the set of observations is

$$\mathcal{F}_t^{(n)} = \left\{ \left(X_t^{(n)}, a_t^{(n)}, X_{t+1}^{(n)} \right) \right\}_{t=0}^{T-1} , \quad n = 1, \cdots, N.$$

Typically, IRL aims to (i) find a reward function $R_t^{(n)}$ that is most consistent with the observed states and actions and (ii) find the optimal policy and action value function(similar to RL). As Russell [1998] noted, a reward function encodes the preferences of an agent while being transferable between environments and agents. Thus, knowing an agents reward function will also reveal its goals and preferences.

In order to differentiate between direct RL and IRL, it is useful to introduce the *occupancy measure*,

$$\rho_\pi(s, a|s_0) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t \Pr[s_t = s|\pi, s_0].$$

Occupancy measure is the discounted sum of probability of being in state s and performing action a, given that the agent starts from state s_0 and follows policy π. Using it, one can define the value function as

$$V(s_0) = \int \rho_\pi(s, a|s_0) r(s, a) \mathrm{d}s\mathrm{d}a.$$

In direct RL, one aims to find the optimal policy π^*, and hence an optimal measure ρ^* satisfying the constraints imposed by model dynamics, that maximizes the value function, given the sampled data in the form of tuples (s_t, a_t, r_t, s_{t+1}). Notice that because numerical rewards are observed, the value function can be directly estimated using Monte Carlo simulations. On the other hand, in IRL, data is attained in the form of (s_t, a_t, s_{t+1}), without rewards r_t. Therefore, solving the IRL problem amounts to finding a policy that maximizes the value function, with respect to some unknown reward function $r(s_t, a_t)$. This is why the assumption that the expert policy is nearly optimal is necessary; otherwise, available observations (s_t, a_t, s_{t+1}) convey very little information on the unknown reward function.

As we mentioned earlier, IRL learns both the reward function r and the state-action occupancy measure ρ_π from the data. Once these two functions are inferred, one can use them to obtain a value function and use that to measure the performance of IRL.

Earlier, it was also mentioned that a reward function is *portable* between environments. Notice that the reward function $r(s_t, a_t)$ is independent of model dynamics. Thus, once the reward function is determined, it can be used to find an optimal policy in environments with different dynamics. Because expected reward is a function of states

and actions, one can try to estimate them by fitting a parameterized action policy π_θ to the observations and claim that the inferred rewards would produce optimal policies in environments with different dynamics than the learning environment. Ng and Russell found that the optimal policy remains unchanged under the following transformation of the reward function:

$$\tilde{r}(s, a, s') = r(s, a, s') + \gamma\Phi(s') - \Phi(s)$$

for arbitrary function $\Phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This reward shape invariance of RL makes the problem of learning a portable reward function quite difficult. Generally, if we use observation from an expert in an environment E_1 to find the reward function, this reward function may not yield an optimal policy in another environment, E_2. For example, if the probability of transition from state s to a desired state s' after performing action a in E_1 is zero($\Pr[s'|s, a] = 0$), using observation from an expert in E_1, one may assume that $r(s, a, s')$ is low. However, the expert did not choose action a, not because it gives a low reward, but because of the way model dynamics work.

5.4. Maximum Entropy Inverse Reinforcement Learning

It is reasonable to assume that the expert follows a stochastic policy $\pi_\theta(a|s)$, under which sub-optimal actions could be observed, though not frequently according to a model for $\pi_\theta(a|s)$. Notice that once a parametric model for $\pi_\theta(a|s)$ is specified, its parameters can be estimated using methods such as maximum likelihood estimation. Our approach in this section is to present a family of stochastic policies $\pi_\theta(a|s)$, obtained by methods that involve maximization of entropy or minimization of KL-divergence, and then find the best fitting policy.

Maximum entropy(MaxEnt) principle Jaynes [1957] is a method for selecting a probability distribution from a family of distributions, given a finite set of constraints on this distribution. The core idea of this approach is that the inferred distribution should as non-informative as possible beyond matching the constraints. Thus, MaxEnt method selects the distribution that maximizes the entropy, while matching all available constraints.

Let $\pi(a|s)$ be a policy and $r(s, a)$ be single step rewards. In a simple single-step RL, an optimal policy should maximize $V^\pi(s) = \int \pi(a|s)r(s, a)\mathrm{d}a$. We define

$$F^\pi(s) := V^\pi(s) + \frac{1}{\beta}H[\pi(a|s)] = \int \pi(a|s)\left[r(s, a) - \frac{1}{\beta}\log\pi(a|s)\right]\mathrm{d}a,$$

where β is a regularization parameter. By taking the derivative with respect to $\pi(a|s)$ and setting it to zero, the optimal policy is found:

$$\pi(a|s) = \frac{1}{Z_\beta(s)}e^{\beta r(s,a)}, \quad Z_\beta(s) := \int e^{\beta r(s,a)}\mathrm{d}a.$$

This optimal policy is the result of maximizing the entropy regularized value function. The same form can be obtained by maximizing the entropy of $\pi(a|s)$ conditional on matching a given average reward $\bar{r}(s)$. The optimal value of β can be calculated by replacing the optimal policy in the equation for F^π and maximizing it with respect to β.

We can also replace the absolute entropy H, with a KL-divergence with some reference policy π_0 to obtain the KL-regularized value function. The same approach for finding the optimal policy can also be used in this setting. In any case, for this simple single-step setting, the MaxEnt optimal policy is exponential in $r(s, a)$.

In general, we would like to consider trajectories that extend over multiple steps, where we have a series of states $\mathbf{S}^T = \mathbf{S}_{0:T}$ and actions $\mathbf{A}^T = \mathbf{A}_{0:T}$ where $T > 1$. Using this notation, the problem of learning a policy is to infer a distribution of actions $\mathbf{A}_{0:T}$ given a distribution of states $\mathbf{S}_{0:T}$. Here, because the sequences are time dependent, we can not consider the simple distribution of actions given states $\Pr[\mathbf{A}_{0:T}|\mathbf{S}_{0:T}]$, as it could violate causality. This is because conditioning on the whole path involves conditioning on the future, while in an MDP, the action taken at time t can depend only on the current state. In order to solve this problem, we define the notion of *causal conditioning*. We say probability of $\mathbf{A}$ causally conditioned on $\mathbf{S}$ and write

$$\Pr[\mathbf{A}^T\|\mathbf{S}^T] = \prod_{t=0}^{T}\Pr[A_t|\mathbf{S}_{0:t}, \mathbf{A}_{0:t-1}].$$

Using this definition, one can factorize the joint distribution $\Pr[\mathbf{A}^T, \mathbf{S}^T]$ as $\Pr[\mathbf{A}^T\|\mathbf{S}^T]\Pr[\mathbf{S}^T\|\mathbf{A}^{T-1}]$. One can also define the causal entropy(Kramer [1998]) as

$$H\left(\mathbf{A}^T\|\mathbf{S}^T\right) = \mathbb{E}_{\mathbf{A},\mathbf{S}}\left[-\log\Pr\left(\mathbf{A}^T\|\mathbf{S}^T\right)\right] = \sum_{t=0}^{T}H\left(\mathbf{A}_t\|\mathbf{S}_{0:t}, \mathbf{A}_{0:t}\right).$$

Furthermore, if the dynamics are Markovian(as is in an MDP), $\Pr[\mathbf{S}^T \| \mathbf{A}^{T-1}] = \prod_t \Pr[s_{t+1} \| s_t, a_t]$. Additionally, because we are considering infinite-horizon MDPs, we use a discounted version of causal entropy:

$$H\left(\mathbf{A}_{0:\infty} \| \mathbf{S}_{0:\infty}\right) = \sum_{t=0}^{\infty} \gamma^t H\left(\mathbf{A}_t \| \mathbf{S}_{0:t}, \mathbf{A}_{0:t}\right).$$

In a first-order MDP, because an action a_t depends only on state s_t, we can further simplify the above equation and write it as

$$H\left(\mathbf{A}_{0:\infty} \| \mathbf{S}_{0:\infty}\right) = \sum_{t=0}^{\infty} \gamma^t H(a_t \| s_t) = -\mathbb{E}_{\mathbf{S}}\left[\int \pi(a_t|s_t) \log \pi(a_t|s_t) \mathrm{d}a_t\right],$$

where the expectation is taken over all future values of s_t.

Causal entropy can be used to extend the MaxEnt method to dynamic processes. Assume that $\mathcal{F}(\mathbf{S}, \mathbf{A})$ is a function of features that was observed in a demonstration with T steps, and that this feature function is additive in time, i.e. $\mathcal{F}(\mathbf{S}, \mathbf{A}) = \sum_t F(s_t, a_t)$. In particular, notice that the total reward is such a function. For such functions we have

$$\mathbb{E}^{\pi}_{\mathbf{A},\mathbf{S}}[\mathcal{F}(\mathbf{S}, \mathbf{A})] = \mathbb{E}^{\pi}_{\mathbf{A},\mathbf{S}}\left[\sum_{t=0}^{\infty} \gamma^t F\left(s_t, a_t\right)\right],$$

where $\mathbb{E}^{\pi}_{\mathbf{A},\mathbf{S}}[\cdot]$ is the expectation over next states and actions, considering environmental dynamics and policy π. If we denote the empirical feature expectation by $\tilde{\mathbb{E}}_{\text{emp}}[\mathcal{F}(\mathbf{S}, \mathbf{A})]$, the objective is to find a policy π that matches $\tilde{\mathbb{E}}_{\text{emp}}[\mathcal{F}(\mathbf{S}, \mathbf{A})]$. Hence, maximum causal entropy optimization can be formulated. We are in particular, interested in its dual problem, attained by swapping the objective and the constraints:

$$\begin{aligned}
&\underset{\pi}{\operatorname{argmax}}\ \mathbb{E}^{\pi}_{\mathbf{A},\mathbf{S}}[\mathcal{F}(\mathbf{S}, \mathbf{A})] - \tilde{\mathbb{E}}_{\mathbf{A},\mathbf{S}}[\mathcal{F}(\mathbf{S}, \mathbf{A})] \\
&\text{Subject to: } H\left(\mathbf{A}^T \| \mathbf{S}^T\right) = \bar{H} \\
&\text{and } \sum_{a_t} \pi\left(a_t \mid s_t\right) = 1, \quad \pi\left(a_t \mid s_t\right) \geq 0, \quad \forall s_t,
\end{aligned}$$

where $\bar{H}$ is some fixed value for entropy. Let us first utilize this dual form in direct RL where rewards are observed and then in IRL.

5.4.1. G-Learning and Soft Q-Learning

When rewards are observed as in the classical RL setting, we can take the expected reward as features and set $F(s_t, a_t) = r(s_t, a_t)$. Also, $\tilde{\mathbb{E}}_{\mathbf{A},\mathbf{S}}$ does not depend on policy π and can be dropped from the objective of maximization problem. We also use KL-divergence with respect to some reference policy π_0 instead of max causal entropy for the regularization. It is defined as

$$\mathcal{KL}[\pi \| \pi_0](\mathbf{s}_t) := \sum_{\mathbf{s}_t} \pi(\mathbf{a}_t|\mathbf{s}_t) \log \frac{\pi(\mathbf{a}_t|\mathbf{s}_t)}{\pi_0(\mathbf{a}_t|\mathbf{s}_t)} = \mathbb{E}_{\pi}[g^{\pi}(\mathbf{s}, \mathbf{a})],$$

where $g^{\pi} = \frac{\pi(\mathbf{a}_t|\mathbf{s}_t)}{\pi_0(\mathbf{a}_t|\mathbf{s}_t)}$ is the one-step information cost of a learned policy π relative to a reference policy. Considering these changes, the problem now transforms to maximizing the following entropy-regularized value function:

$$F^{\pi}_t(\mathbf{s}_t) = \sum_{t'=t}^{T} \gamma^{t'-t} \mathbb{E}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \frac{1}{\beta} g^{\pi}(\mathbf{s}_{t'}, \mathbf{a}_{t'})\right].$$

In the previous chapter, when we talked about G-Learning, we showed that this optimization can be solved by iteratively solving systems of linear equations, backwards in time. It is worth noting that the optimal policy found in G-Learning (or Max-Causal Entropy) has a Boltzmann form.

5.4.2. Maximum Entropy IRL

We now use the G-Learning(maximum causal entropy) framework to solve the IRL problem. Assume that the time-stationary action value function is specified in a parametric form with parameters θ as $G_\theta(s_t, a_t)$. The objective is to learn parameters θ form data. From our discussions on G-Learning in the previous chapter, we know that the policy can be written as

$$\pi_\theta(a_t|s_t) = \frac{1}{Z_\theta(s_t)}\pi_0(a_t|s_t)e^{\beta G_\theta(s_t,a_t)}, \quad Z_\theta(s_t) := \int \pi_0(a_t|s_t)e^{\beta G_\theta(s_t,a_t)}\mathrm{d}a_t.$$

We can now use a conventional maximum likelihood method to fit the parameters θ so that the resulting policy models the observations. More formally, the likelihood of a particular path τ is

$$\Pr[\tau] = p(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)\Pr[s_{t+1}|s_t, a_t].$$

We then take the negative logarithm of this probability and drop the terms that are not dependent on θ to obtain the following loss function:

$$\mathcal{L}(\theta) = \sum_{t=0}^{T-1}(\log Z_\theta(s_t) - \beta G_\theta(s_t, a_t)).$$

Minimizing this loss function will yield an optimal G-function which can be used to find the reward function using the Bellman equations. Gradient of this loss function is

$$\frac{\partial\mathcal{L}(\theta)}{\partial\theta} = \left\langle\frac{\partial G_\theta(\mathbf{s},\mathbf{a})}{\partial\theta}\right\rangle_{\text{model}} - \left\langle\frac{\partial G_\theta(\mathbf{s},\mathbf{a})}{\partial\theta}\right\rangle_{\text{data}}.$$

As can be seen from the above equations, computing loss functions gradient, both in RL and IRL, involves solving an integral over all possible actions(examine Z_θ). This requires a lot of computational power and can become a bottleneck for the algorithm. When the numerical value of integrals over probability distributions are desired, one common method is importance sampling. If we denote the sampling distribution by $\hat{\mu}(\mathbf{a}_t|\mathbf{s}_t)$, as its easier to work with this distribution, we can multiply and divide the integrand by it, and compute the gradients multiplied by a likelihood ratio.

$$\int \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\frac{\partial G_\theta(\mathbf{s}_t,\mathbf{a}_t)}{\partial\theta}\mathrm{d}\mathbf{a}_t = \int \hat{\mu}(\mathbf{a}_t|\mathbf{s}_t)\frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\hat{\mu}(\mathbf{a}_t|\mathbf{s}_t)}\frac{\partial G_\theta(\mathbf{s}_t,\mathbf{a}_t)}{\partial\theta}\mathrm{d}\mathbf{a}_t.$$

This method becomes more accurate if the sampling distribution $\hat{\mu}(\mathbf{a}_t|\mathbf{s}_t)$ is close to the optimal action policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$. There are other variants of this idea, such as "guided cost learning" methods in robotics(Finn et al. [2016]).

Notice that MaxEnt IRL is a behavioral cloning that fits $G_\theta(s_t, a_t)$ to the data. Therefore, it suffers from all the problems of behavioral cloning that were mentioned earlier. One way around some of these problems, is to consider a parametric form for the reward function r_θ (such as a linear architecture or a neural network). For any set of parameters θ, we can then replace the reward function in the Bellman equation, and find the G-function, and use it to optimize parameters θ. Although this makes the estimate of rewards more compatible with the dynamics, this approach requires a massive computational power as it needs to solve a direct RL problem to find G_θ at each of its iterations in optimizing θ.

5.5. Adversarial Imitation Learning and Inverse Reinforcement Learning

In MaxEnt IRL, we assumed a fixed but unknown reward function for the MDP, and solved a direct RL problem with this reward function to obtain an optimal policy. We then calibrated the parameters of this optimal policy such that it matches the observations with respect to some conventional maximum likelihood method. We saw that this approach required solving an RL problem per each iteration. In order to avoid this, the generic feature function $\mathcal{F}(\mathbf{S}, \mathbf{A})$ is replaced by a true unknown reward function $r(s, a)$ and then we simultaneously optimize with respect to both the policy π and the reward $r(s, a)$. First, let us reformulate the problem. Assume that $\rho_\pi(s, a)$, $\rho_E(s, a)$ are the joint state-action distributions induced by policy π and the expert policy π_E(which is available via samples). In order to comply with the convention of RL literature, the reward function is replaced with a (negative) cost function $c(s, a) = -r(s, a)$. Using this notation, the problem can be stated as

$$\max_{c(s,a)}\min_{\pi}\mathbb{E}_{\rho_\pi}[c(s, a)] - \mathbb{E}_{\rho_E}[c(s, a)]$$

Subject to: $H^{\text{causal}}(\pi) = \bar{H}$,

where $H^{\text{causal}}(\pi) = \mathbb{E}_{\rho_\pi}[\pi(a|s)]$ is the causal entropy of policy π, and $\bar{H}$ is some fixed value of the causal entropy. If we use a soft relaxation of the above constraint, we can rephrase the problem as a single minimization problem

$$\text{IRL}(\pi_E) = \max_{c(s,a)} \min_{\pi} -H^{\text{causal}}(\pi) + \mathbb{E}_{\rho_\pi}[c(s,a)] - \mathbb{E}_{\rho_E}[c(s,a)].$$

This IRL problem requires solving the following direct RL problem in its inner loop:

$$\text{RL}(c) = \min_{\pi} -H^{\text{causal}}(\pi) + \mathbb{E}_{\rho_\pi}[c(s,a)].$$

Now, we try to bypass the necessity of solving a direct RL problem when finding an optimal policy from demonstrations that do not include rewards. To this end, we replace the parametric reward function with a non-parametric one and try to solve the optimization analytically. Consider a convex function $\psi(c)$ and add $-\psi(c)$ to the IRL objective function as a regularization. It can be seen that the objective function is convex in π and concave in c. As a result, the optimal solution $(\pi^*, c^*(s,a))$ is a saddle point (max-min solution) of the objective function. Due to strong duality, one can change the order of maximization and minimization in this optimization. Using this, the problem can be stated as

$$\text{RL}\circ\text{IRL}(\pi_E) = \min_{\pi} -H^{\text{causal}}(\pi) + \max_{c(s,a)} -\psi(c) + \mathbb{E}_{\rho_\pi(s,a)}[c(s,a)] + \mathbb{E}_{\rho_{\pi_E}(s,a)}[c(s,a)].$$

Hence,

$$\text{RL}\circ\text{IRL}(\pi_E) = \min_{\pi} -H^{\text{causal}}(\pi) + \max_{c(s,a)} \left(\sum_{s,a} (\rho_\pi - \rho_E)c(s,a) - \psi(c) \right) = \min_{\pi} -H^{\text{causal}}(\pi) + \psi^*(\rho_\pi - \rho_E),$$

where $\psi^* : \mathbb{R}^{S\times\mathcal{A}} \to \bar{\mathcal{R}}$ is the convex conjugate of ψ:

$$\psi^*(\mathbf{x}) = \sup_{\mathbf{y}\in\mathbb{R}^{S\times\mathcal{A}}} \mathbf{x}^T\mathbf{y} - \psi(\mathbf{y}).$$

This means that our objective is now to find a policy π that minimizes the difference between ρ_π and ρ_E as measured by the convex conjugate ψ^*, regularized by the causal entropy(Ho and Ermon [2016]). Notice that for an arbitrary function ψ, this is only a formal improvement, as we have substituted maximization over $c(s,a)$ by another maximization in the definition of convex conjugate. However, if ψ is chosen wisely such that ψ^* is known in a closed form, the maximization step will truly vanish. For example, as was done in Ho and Ermon [2016], Jensen-Shannon divergence $D_{\text{JS}}(\rho_\pi, \rho_E)$ could be considered for the convex conjugate regularization ψ^*, which is a true metric measuring the distance between ρ_π and ρ_E. This definition will lead to the following *generative adversarial imitation learning* (GAIL) algorithm: "Find a policy π whose occupancy measure ρ_π minimizes the JS distance to the expert policy π_E, regularized by its causal entropy:

$$\text{RL}\circ\text{IRL}(\pi_E) = \min_{\pi} D_{\text{JS}}(\rho_\pi, \rho_E) - \lambda H^{\text{causal}}(\pi).\text{"}$$

It is useful to think of JS divergence as the loss function of a binary classifier that, given input (s,a), classifies it as "generated by expert policy" with probability $D(s,a)$ and as "generated by policy π" with probability $1 - D(s,a)$. This classifier, known as a *discriminator*, differentiates between policy π and π_E. The JS divergence can then be written as

$$\psi^*_{\text{GA}}(\rho_\pi - \rho_E) = D_{\text{JS}}(\rho_\pi, \rho_E) = \max_{D\in[0,1]^{S\times\mathcal{A}}} \mathbb{E}_{\pi_E}[\log D(s,a)] + \mathbb{E}_{\pi}[\log(1 - D(s,a))],$$

which gives the result

$$D(s,a) = \frac{\rho_E(s,a)}{\rho_\pi(s,a) + \rho_E(s,a)}.$$

(For trajectories, the log-likelihood of a binary trajectory classifier can be used as an empirical estimate of JS divergence.)

After replacing the optimal D in the equation for D_{JS}, we need to calculate an expected value with respect to policy π, which requires integrating over a high-dimensional space. As was said earlier, importance sampling can be used to evaluate such integrals. This suggests an alternating procedure between using importance sampling with a fixed sampling distribution to evaluate the first term in the loss function, $D_{\text{JS}}(\rho_\pi, \rho_E) - \lambda H^{\text{causal}(\pi)}$, and updating the sampling distribution to bring it closer to optimal policy and expert policy. In practice, this is done by considering a parameterized discriminator $D_w(s,a)$(e.g., implemented by a neural network) and a parameterized sampling distribution $\pi_\theta(a_t|s_t) = G_\theta(s_t, z_t)$, where z_t is a random noise such as Gaussian. Using the latter as a generator to produce samples from policy $\pi \sim G_\theta$, the optimization problem of RL$\circ$IRL(π_E) can be thought of as a min-max game between the generator G_θ and the discriminator D_w:

$$\text{RL}\circ\text{IRL}(\pi_E) = \min_{\pi\sim G_\theta} \max_{D_w\in[0,1]} \mathbb{E}_{\pi_E}[\log D_w(s,a)] + \mathbb{E}_\pi\left[\log\left(1 - D_w(s,a)\right)\right] - \lambda H^{\text{causal}}(\pi)$$

This min-max game employed by GAIL is a special case of *generative adversarial training* introduced in the renowned work of Goodfellow et al. [2014] on GANs.

Gaussian Process IRL:

Thus far, the reward function was modeled as an unknown but deterministic function of states and actions. Here, we wish to consider random (and still unobserved) rewards. let $X = (s,a)$ denote a state-action pair. Reward $r = r(X)$ is a random variable that depends on X. Using a Bayesian terminology, let $p(r|X)$ be a prior probability distribution for the random rewards given features X. For a (fixed) reward function r, the probability of observing data $\mathcal{D}$ is given by a likelihood function $p(\mathcal{D}|r,\theta)$ where θ are parameters of our model. The joint distribution of observing data $\mathcal{D}$ and rewards r given input features is

$$p(\mathcal{D}, r|X) = p(r|X)p(\mathcal{D}|r,\theta).$$

Because the rewards are not observed, we shall use the expected likelihood of data by using the marginal distribution of $\mathcal{D}$ given X:

$$p(\mathcal{D}|X) = \int p(r|X)p(\mathcal{D}|r,\theta)\mathrm{d}r.$$

In order to evaluate this integral, one can discretize r into M possible values. But then one has to solve M direct RL problems of finding $p(\mathcal{D}|r,\theta)$ so that we can solve one IRL problem. Another approach is to use Laplace (saddle-point) approximation to estimate its value. Both of these methods, require a parametric prior distribution $p(r|X)$ of rewards. Instead, Gaussian process (GP) IRL uses a zero mean GP prior:

$$r \sim \mathcal{GP}(0, k_\theta(\mathbf{x}_i, \mathbf{x}_j)),$$

where k_θ is the covariance function, e.g.

$$k_\theta(\mathbf{x}_i, \mathbf{x}_j) = \sigma_k^2 e^{-\frac{\xi}{2}(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}$$

and $\theta = (\sigma_k, \xi)$ is the vector of model parameters. Thus, finding the reward function amounts to finding parameters θ of the kernel function. Notice however, that this is different from a normal GP regression in that here, the values of the reward function are not observed. Instead, GPIRL uses MaxEnt IRL. More formally, if u is the true reward and r is its noisy version, the posterior probability of u and θ given data $\mathcal{D}$ and a set of "inducing points" X_u is

$$\Pr[u,\theta|\mathcal{D}, X_u]\Pr[u,\theta,\mathcal{D}|X_u] = \Pr[u,\theta|X_u]\left[\int \Pr[\mathcal{D}|r]\Pr[r|u,\theta,X_u]\mathrm{d}r\right].$$

Here, $\Pr[\mathcal{D}|r]$ is the probability of observations conditioned on a fixed reward function. GPIRL uses a MaxEnt policy to evaluate this expression. Levine et al. [2011] considered the special case where there is no noise in r, which implies $\Pr[r|u,\theta,X_u] = \delta(r-u)$.

5.6. Surpassing the Teacher

So far, what we have done can be summarized as "trying to imitate a teacher, based on his actions, in novel situations". In doing so, we found a reward function that best "justifies" actions of the teacher, and then used this reward function to predict what the teacher would have done, had he been in a novel situation. Notice that in that

setting, we had no interest in failed demonstrations which can provide highly valuable data on the experts intents and preferences. As was said before, this requires the teacher to be optimal(or close to optimal). It also means that there is very little hope of improving upon the teacher, as the learned reward function does not resemble the *true* reward function of the teacher, except when he is acting optimally(in which case no improvement can be made). In this section, we try to overcome this limitation through different approaches. It is worth noting that the ability to learn the teachers intentions is extremely useful because (i) providing optimal expert data is hard and imitating a sub-optimal strategy is not very interesting, (ii) Ensuring optimality of demonstrations is difficult, (iii) It allows the *explanation* of teacher/expert rather than only *justification* for his actions, and (iv) unsuccessful demonstrations might be easier to produce, and provide as much valuable information on the teachers intentions.

IRL from Failure(IRLF):

The basic idea of IRLF is to extend classical IRL methods(such as MaxEnt IRL) so that they can also learn from failed demonstrations. Conventional IRL tries to find a policy such that feature expectation of the learned policy matches the empirical expectation. IRLF further adds to this that the feature expectation should be maximally dissimilar to the empirical expectation of failed demonstrations. More formally, if $\mathcal{D}$ and $\mathcal{F}$ are the set of successful and failed demonstrations respectively, and that there are K features $\phi_k(s,a)$ such that rewards can be written as $r(s,a) = (w^{\mathcal{D}} + w^{\mathcal{F}})^T \phi(s,a)$, IRLF maximizes the following objective function:

$$J(\pi, \theta, z, w^{\mathcal{D}}, w^{\mathcal{F}}) = H(A\|S) - \frac{\lambda}{2}\|\theta\|^2 + \sum_{k=1}^{K} \theta_k z_k + \sum_{k=1}^{K} w_k^{\mathcal{D}}(\mu_k^{\pi}|_{\mathcal{D}} - \tilde{\mu}_k^{\mathcal{D}}) + \sum_{k=1}^{K} w_k^{\mathcal{F}}(\mu_k^{\pi}|_{\mathcal{F}} - \tilde{\mu}_k^{\mathcal{F}} - z_k)$$

Here, $H(A\|S)$ is the causal entropy, $w^{\mathcal{F}}, w^{\mathcal{D}}$ are two sets of feature expansion coefficients, $\tilde{\mu}_k$ is the empirical feature expectation, μ_k^{π} is the feature expectation under policy π, and $z_k \in \mathbb{R}$ is an auxiliary variable, maximizing over which maximizes the dissimilarity to empirical features in failed demonstrations. The IRLF algorithm then works by iterating between updating the reward function by $r(s,a) = (w^{\mathcal{D}} + w^{\mathcal{F}})^T \phi(s,a)$ and updating the policy π(similar to MaxEnt IRL).

Trajectory-Ranked Reward EXtrapolation(T-REX):

In order to solve the IRL problem, an alternative to learning the reward function is to learn the experts preferences, i.e., rank trajectories in relation to each other without explicitly assigning numerical values to them. The end result of such approach is again represented as a reward function; however, this reward function is based on an inferred intent. Thus, it can be used to improve upon the teachers performance. This is in contrast to previous IRL and IL algorithms that we saw, as they find a reward function that best justifies the observed trajectories.

T-REX proposed by Brown et al. [2019], is an algorithm that utilizes this approach to solve the IRL problem. It uses a number of ranked demonstrations. This can either be in the form of a global ranking, or an ordinal relation on pairs of trajectories. Such a relation is sufficient as T-REX works with pairs of trajectories. More formally, let $\tau_1, \cdots, \tau_N$ be N trajectories, ranked from worst to best(i.e., $\tau_i \prec \tau_j$ if $i < j$). T-REX algorithm has two main steps: (i) reward inference, (ii) policy optimization. The second step is to solve a regular RL problem. Thus, here we focus on the first step, which is the IRL part. T-REX's objective is to find a parameterized reward function $\hat{r}_\theta(s,a)$ that approximates the reward function that the teacher wants to optimize. This is similar to previous IRL methods, however, T-REX adds the additional structural constraint that "cumulative rewards computed with this reward function should match the rank ordering relation".

Let $\hat{J}_\theta(\tau_i) = \sum_t \gamma^t \hat{r}_\theta(s_t, a_t)$ be the discounted cumulative reward of trajectory τ_i. T-REX model trains by minimizing the following loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau_i, \tau_j \sim \Pi}\left[\xi\left(\Pr\left[\hat{J}_\theta(\tau_i) < \hat{J}_\theta(\tau_j)\right], \tau_i \prec \tau_j\right)\right],$$

where $\xi(\cdot,\cdot)$ is a binary loss function and Π is a distribution over pairs of demonstrations. If we model the binary event probability $\Pr[\cdot < \cdot]$ by a soft-max distribution and use a cross-entropy loss function for ξ, the above loss function takes the following form:

$$\mathcal{L}(\theta) = -\sum_{\tau_i \prec \tau_j} \log \frac{\exp \sum_{(s,a)\in\tau_j} \hat{r}_\theta(s,a)}{\exp \sum_{(s,a)\in\tau_i} \hat{r}_\theta(s,a) + \exp \sum_{(s,a)\in\tau_j} \hat{r}_\theta(s,a)}$$

Essentially, minimizing this loss function trains a classifier that given two trajectories, predicts which is preferred to the other. In doing so, it finds a one step reward function $\hat{r}_\theta$.

Disturbance-based reward extrapolation(D-REX) proposed by Brown et al. [2020], is an extension to the above algorithm that automatically ranks the demonstrations, and then follows the T-REX algorithm to learn from them.

6. Dixon (2020) Machine Learning in Finance From Theory to Practice

This review is based on the twelfth chapter of Dixon et al. [2020].

6.1. Abstract

In this chapter, several novel areas of research in financial machine learning are explored. These include multiple ideas that originate from physics and some that try to unify multiple disciplines of machine learning.

6.2. Introduction

Machine learning is experiencing a rapid growth in the recent years. New ideas such as generative adversarial networks, networks with attention, deep reinforcement learning, etc. have all been developed in this time. These new methods specially focus on addressing real-world challenges. Therefore, they can potentially be more successful in financial applications, in comparison to classical methods. In this element, some of the more novel ideas in financial machine learning are introduced. These methods originate from different fields such as dynamical systems, physics, information theory, etc. There will also be a brief discussion on efforts that try to unify different disciplines of machine learning.

6.3. Modeling Market Dynamics

Inverse reinforcement learning IRL, discussed in the previous chapter, can be used to provide a multivariate extension of the geometric mean reversion (GMR) model (Halperin and Feldshteyn [2018]). Quantum equilibrium-disequilibrium(QED) is another model that can describe corporate defaults and market crashes as well as a stable growth phase.

Let X_t denote the total capitalization of a firm at time t, re-scaled to a dimensionless quantity of order one $X_t \sim 1$(e.g. by dividing it by its mean over the observation period). The following equations are used to describe discrete-time market dynamics:

$$X_{t+\Delta t} = (1 + r_t \Delta t)(X_t - c X_t \Delta t + u_t \Delta t),$$
$$r_t = r_f + \mathbf{w}^T \mathbf{z}_t - \mu u_t + \frac{\sigma}{\sqrt{\Delta t}} \varepsilon_t.$$

Here, r_f is the risk free rate, c is a constant dividend rate, $\mathbf{z}_t$ is the vector of predictors with weights $\mathbf{w}$, μ is a market impact parameter with a linear specification, $u_t = u_t(X_t, \mathbf{z}_t)$ is a cash inflow/outflow from outside investors and $\varepsilon_t = \mathcal{N}(0, 1)$ is a white noise.

The amount of capital injected by investors at time t, $u_t \Delta t$, should be a function of current market capitalization X_t and other factors such as signals. Here, a simple functional of X_t is used:

$$u_t = \phi X_t + \lambda X_t^2,$$

with two parameters ϕ and λ and no signals. However, one can also integrate different signals into the above equation, or consider ϕ and λ as slowly varying functions of signals $\mathbf{z}_t$. QED model is obtained by taking the limit $\Delta_t \to 0$ in the market dynamic equations when capital injection has the simple form that was described:

$$\mathrm{d}X_t = \kappa X_t (\frac{\theta}{\kappa} - X_t - \frac{g}{\kappa} X_t^2)\mathrm{d}t + \sigma X_t (\mathrm{d}W_t + \mathbf{w}^T \mathbf{z}_t),$$

where W_t is the standard Brownian motion and parameters g, κ, θ are defined as

$$g = \mu\lambda, \quad \kappa = \mu\phi - \lambda, \quad \theta = r_f - c + \phi.$$

This equation is a special case of the Langevin equation(Langevin [1908]), which extends the model of a free Brownian motion diffusion:

$$\mathrm{d}x_t = -U'(x_t)\mathrm{d}t + \sigma x_t \mathrm{d}W_t.$$

U, is a potential function that codifies the impact of other particles or external fields, and is usually a non-linear function. The GBM model can be thought as a Langevin equation, where the variable x_t is the stock price S_t, the potential function takes the simple linear form of $U_t = \mu(S_t) = \mu S_t$ and $\sigma(S_t) = \sigma S_t$ of a general Itô diffusion:

$$\mathrm{d}S_t = \mu(S_t)\mathrm{d}t + \sigma(S_t)\mathrm{d}W_t.$$

This looks very similar to the equation for a harmonic oscillator, where $U(x) = \frac{m}{2}x^2$, except that the sign of the drift term is different. This means that the equation for GBM model, corresponds to a harmonic oscillator with negative mass, thus, its potential is an inverted parabola with no stable points. As a result, the dynamics described by the GBM model are unstable and the model itself is incomplete.

On the other hand, the QED model corresponds to the following quartic potential function:

$$U(x) = -\frac{1}{2}\theta x^2 + \frac{1}{3}\kappa x^3 + \frac{1}{4}g x^4,$$

which allows the potential to have various forms.

If we set $\theta = \mu$, this potential coincides with the potential $U(x) = \frac{1}{2}\mu x^2$ of the GBM model, in the limit $\kappa \to 0, g \to 0$. Hence, QED model is an extension of the GBM model. However, unlike the GBM model, the QED model allows for dynamics to be *meta-stable*. Such dynamics are different from globally stable dynamics in that they eventually change, although it might take a very long time. In physics, barrier-hopping transitions for the potential function are known as *instantons*, as the transition between a meta-stable state and a regime of instability happens instantaneously, although it might take a long time for this hopping to occur. In finance, hopping over the barrier to reach the zero level, corresponds to corporate bankruptcy(default). Notice that the GBM model can not model these events. It is worth noting that the QED model has only one degree of freedom(the stock price) but is able to model (and train on) corporate defaults, which allows for a better usage of data for calibrating model parameters. It also incorporates capital inflow in the market and their impact on asset prices.

When a traditional model of returns is trained on market data, stocks that have defaulted in the past are removed from the training set. This results in biased data as it conveys no information on these events. In order to make up for this missing data, regularizations are used. In the QED model, "Kramer's regularization" is used to to ensure the presence of a potential barrier between meta-stable and unstable states. This type of regularization, discussed in depth in Halperin and Dixon [2020], encourages using more specialized regularizations that preserve some static or dynamic symmetries, as opposed to general L_1 or L_2 regularizations.

6.4. Ideas from Physics

Monte Carlo methods, Boltzmann machines, maximum entropy inference and energy-based models are all examples of different methods in ML that have origins in physics. Here, some more ideas from physics are presented, which can potentially be used in ML.

Hierarchical Representations: Deep learning constructs a hierarchical representation of input data using a multi-layer composition of non-linear functions. This can be thought of as producing more abstract features as we move higher in the network. *Renormalization group* (RG) is an analogous concept in physics. RG investigates a system when viewed from different scales. It can be used to explain emergence of large-scale structures from small components, as is the case in deep learning. The basic idea of real-space RG (Efrati et al. [2014]) is to coarse-grain a set of degrees of freedom $\mathbf{X}$ so that we can integrate out short-range fluctuations and retain long-range correlations. If $H(\mathbf{X}, \theta)$ is the Hamiltonian in variables $\mathbf{X}$ with parameters θ, the coarse-grained Hamiltonian $H(\mathbf{X}', \theta')$ has the same functional form, but different parameters. *RG flow equations* formalize the relationship between effective theories at different length scales. As a comprehensive theory of deep networks has not been suggested yet, exploring the similarities and differences between RGs and deep learning seems enticing. For example, Ising model, a statistical mechanics model for a magnet, was used to train an unsupervised restricted Boltzmann machine(RBM) in Koch et al. [2020].

Tensor Networks: Tensors are multi-dimensional arrays that extend the notion of a matrix, as a 2 dimensional array, to multiple dimensions. Although in many cases(e.g., example images or financial data) tensors provide a natural format for the data, most of modern machine learning methods(such as neural nets) accept inputs in the form of a vector, rather than a tensor. linearizing a tensor breaks the structural correlations in the data, and these relations will become hidden. There has been studies on analysis of tensor-valued data, known as *tensor decomposition*. Similar to how singular value decomposition(SVD) works with matrices, these methods provide a systematic way of decomposing N dimensional tensors into lower dimensional tensors. Matrix product state (MPS) is an example of such methods that has been used extensively in physics. Kolda and Bader [2009] provides an overview of such methods.

A tensor network is a factorization of an order N tensors into the contracted product of low-order tensors. Similar to deep learning, tensor networks provide construct a hierarchical representation of data, with progressively more abstract features in the higher levels. Similar to PCA, such a decomposition does not allow for total retrieval of the input, it rather stores a de-noised version of it.

Aside from gaining insight on deep learning, tensor networks can be used in many other ML tasks. Stoudenmire [2018] is an example of using tensor networks to construct features in an unsupervised manner.

Bounded Rational Agents in Non-equilibrium Environments: In thermodynamics, a non-equilibrium process is often modeled by a time-dependent parameter $\lambda(t) \in [0, 1]$ that specifies the energy function $E_\lambda(x)$ at time t. When λ changes infinitely slowly, the system probability distribution follows the path of equilibrium distributions $p_\lambda(x) = \frac{1}{Z_\lambda} e^{-\beta E_\lambda(x)}$. However, in general, the non-equilibrium path of probability distributions can be different from the equilibrium path. In finance, the assumption of equilibrium does not hold when the market changes fast and the prices don't have enough time to reach an equilibrium. It is therefore, worthwhile to consider the problem of decision making in a non-equilibrium environment.

A bounded rational agent, first introduced in Simon [1956], is an agent with constrained information processing power. Thus, it may not always act optimally due to a lack of computational resources. Multiple extensions to the classical model using information theory and thermodynamics have been proposed such as Ortega and Braun [2013], Tishby and Polani [2011]. Following these approaches, a bounded rational agent maximizes an augmented reward function that incorporates the information cost of updating some prior policy π_0. In this setting the "rationality" of an agent can be controlled by an inverse temperature parameter β. Thus, "decision making" is considered to be the process of updating a prior policy π_0 to a posterior policy π. For example, G-learning, considered in the previous chapter, is a model for bounded rational agent in an environment where the equilibrium assumption holds.

It is also interesting to consider a bounded rational agent in a non-equilibrium environment, as was done in Grau-Moya et al. [2018]. A bounded rational agent can not fully utilize the difference in free energies of equilibrium states when the environment is out of equilibrium. Grau-Moya et al. [2018] obtained relations for free energy changes of a bounded rational agent upon non-equilibrium changes of environment. However, this was done in a single-step utility optimization. Extending its result to an optimization over multiple steps can be useful for applications in reinforcement learning.

6.5. Grand Unification of ML

Almost all ML techniques fall into one of three categories, supervised learning, unsupervised learning and reinforcement learning. An agent uses the first two to *perceive* and uses the latter to *act*. In most cases, perception and action are closely tied to each other and agents live by repetitively following a *perception-action cycle*. For example, an RL agent uses features that were selected using a supervised or unsupervised learning algorithm. However, perception and action are treated separately and each part of the agent is oblivious to the work of the other parts. It is more natural to think of these different parts as one and try to somehow unify them.

As an example, consider an artificial agent for trading in the market. This agent uses supervised learning to find good "signals" from the data that are both predictive of future asset returns and of their own future value(i.e., have high auto-correlation). This part uses historical data to train(perception). Then, the agent uses a reinforcement learning algorithm to act in the market, given features that were derived(action). As was mentioned, a major problem is that these two tasks are treated separately. This is specially problematic where a feedback loop exists; in this case perception should not be passive. It can be thought of as a bidirectional information flow between the agent and the environment that are codependent. If we treat perception and action separately, the effect of agents actions on the flow of information from environment to agent is ignored. Information theoretic techniques can be used in conjunction with RL to create agents that act more harmoniously. As was seen in the previous chapters, an augmented reward function that is penalized by KL divergent term, can be used to model a bounded rational agent. Therefore, G-learning is able to improve vanilla RL by taking the information flow from the environment to the agent into account. Quite recently, Tiomkin and Tishby [2017] developed an extension that considers the information flow from the agent to the environment. In their model, they considered two asymmetric information channels that describes the interactions between environment and agent. For an infinite horizon MDP, they were able to obtain Bellman-like equations for the causal information between the environment and the agent, which combined with Bellman equation for value function, was used to derive both causal information flow and the value function.

Another approach is being pursued by Google's DeepMind, where the problem of finite-horizon planning based on look-ahead search is the main focus. They mainly use complex, deep neural networks as universal function approximators to represent the value or policy function of an RL agent. For further information on these methods and innovative ideas, one can refer to Silver et al. [2017], Schrittwieser et al. [2019].

7. Socher (2011) Parsing Natural Scenes and Natural Language with Recursive Neural Networks

This review is based on Socher et al. [2011].

7.1. Introduction and Literature Overview

In this paper, recursive neural networks(RNNs) are used to recover the recursive structure present in many areas. Many problems such as parsing natural language sentences or scene understanding exhibit the kind of recursive structure that can be used by this algorithm. This algorithm is therefore a general method that can be used in various settings.

Scene understanding is a fundamental problem in computer vision. It can be broken into three components: understanding what are the objects in the scene(annotation), where the objects are located(segmentation), to what general category does the image belong(classification). These tasks were usually treated separately; for classification, global descriptors(Oliva and Torralba [2001]) that classify images into broad categorize obtained good results but did not have a deep understanding of the objects in a scene. There has also been many approaches for image annotation and segmentation(e.g., Rabinovich et al. [2007] and Gupta and Davis [2008] where connections with NLP are also explored). There has been recent efforts to combine these methods(e.g., Hoiem et al. [2008], Li et al. [2009]).

Syntactic parsing of natural language sentences is also an important task in natural language processing(NLP), as it has a key role in connecting linguistic expressions with meaning. The similarities between NLP ideas and computer vision has been noticed and explored before. Among many others, we can mention Zhu and Mumford [2007] where the idea of creating a grammar for an image was explored and Tighe and Lazebnik [2010] where the authors considered parsing images using super-segmentation and supervised learning techniques. The method that is presented in this paper is similar to these ideas but more general. Deep learning is another idea that has recently been used for both vision tasks and NLP. In most cases, limiting the input size of the neural network is a challenge that has to be overcome. As an example of the recent work in this area, Lee et al. [2009] is an example of the recent work that is being done in this area, where the authors introduced convolutional deep belief networks that can handle more realistic image sizes. In NLP, the inputs are modified using convolutional and max-pooling layers or only a fixed window around a word is considered. The method presented in this paper handles variable length inputs in a more natural way.

7.2. Main Results

We now present an overview of the method that was introduces in the paper. The idea behind the algorithm is to start from small elements in the structure(super pixels in images and words in sentences) and recursively merge adjacent elements in a structurally coherent way to obtain larger elements(this is done using an RNN). After merging all the elements, the final element(which represents the whole image or sentence) can be used to classify the structure.

In order to map elements of images to the space of inputs of the RNN, the algorithm oversegments the image into superpixels. It then computes features such as color, texture features, appearance and shape features, etc. for each segment(Gould et al. [2009]). Using these features, a simple neural network layer is used to map them into the semantic n-dimensional space in which the RNN works. If F_i is the feature vector for a segment, the representation of it is obtained using the following formula:

$$a_i = f(W^{\text{sem}} F_i + b^{\text{sem}}),$$

where W^{sem}, b^{sem} are parameters and f is the sigmoid function. Similarly, for NLP purposes, representations of words as vectors are stored in a word embedding matrix, L. This matrix can be treated as a look-up table to obtain the vector representation of a word:

$$a_i = L\mathbf{e}_k.$$

Aside from the set of activation vectors$\{a_1, \cdots, a_N\}$, a symmetric adjacency matrix A is provided where $A_{ij} = 1$ if segments i and j are neighbours and can be merged.

Each way of merging adjacent elements together until one element is left, can be thought of as a binary parse tree. For a training input x where segment labels are l, $Y(x, l)$ is the set of all "correct" binary parse trees(i.e., all elements with the same label are merged together before merging with other elements). Using this, the structured margin loss(Δ) for a parse tree $\hat{y}$ is defined as the number of its internal nodes d, for which the sub-tree rooted at d does not appear in any of ground truth trees of $Y(x, l)$. The algorithm uses an RNN and a scoring function s to obtain a score for any

parse tree $\hat{y}$, which is higher when the RNN is more certain of the correctness of $\hat{y}$. If we want to ensure the presence of the highest scoring tree in the set of correct trees for all training data, $f_\theta(x_i) \in Y(x_i, l_i)$, while wanting the score of the highest scoring tree y_i to be as large as the structured margin loss, the following regularized loss function is obtained:

$$J(\theta) = \frac{1}{N}\sum_i r_i(\theta) + \frac{\lambda}{2}||\theta||^2, \text{ where}$$

$$r_i(\theta) = \max_{\hat{y}} \left(s(\text{RNN}(\theta, x_i, \hat{y})) + \kappa\Delta(x_i, l_i, \hat{y})\right) - \max_{y_i \in Y(x_i, l_i)} \left(s(\text{RNN}(\theta, x_i, y_i))\right).$$

Now, the RNN structure is described. For each pair of neighbouring segments, their activation's,a_i, a_j, are concatenated and given as input to a neural network. The network computes the potential parent representation for these nodes, $p_{i,j}$. Having this representation of a parent, a score can be obtained by computing an inner product with a row vector $W^{\text{score}} \in \mathbb{R}^{1\times n}$:

$$s_{i,j} = W^{\text{score}} p_{i,j} = W^{\text{score}} f(W[a_i; a_j] + b).$$

After computing the score for all neighbouring segments, the pair with the highest score is selected and its segments are merged together to get a new segment, which is adjacent to all of its components neighbours and its representation is the output of neural net, $p_{i,j}$. This process is repeated until only one segment is remained. The activation for the last segment represents the entire image. The final score that is used for structure prediction is the sum of all local scores:

$$s(\text{RNN}(\theta, x_i, \hat{y})) = \sum_{d \in \text{internal nodes}(\hat{y})} s_d.$$

The label of each super segment that is created along the way(including the last segment which represents the whole structure) can be obtained by applying a softmax layer to its activation:

$$\text{label}_p = \text{softmax}(W^{\text{label}} p).$$

Because in a sentence each word has at most two neighbours and only one correct tree exists(i.e., $|Y(x_i, l_i)| = 1$), some optimizations can be used. See Socher et al. [2010] as an example.

Parameters of this model are $\theta = (W^{\text{sem}}, W, W^{\text{score}}, W^{\text{label}})$. The objective function $J(\theta)$ as defined above is not differentiable, hence, a generalization of gradient descent using sub-gradients is used(Ratliff et al. [2007]), and derivatives are calculated using back-propagation through structure(Goller and Kuchler [1996]).

Hyperparameters of this model are n, the size of the hidden layer, κ the penalization term for incorrect parsing decisions and λ, the regularization parameter. In the experiments the authors chose $n = 100$, $\kappa = 0.05$, $\lambda = 0.001$. For scene understanding, Stanford background data set was used. For natural language parsing, the Wall Street Journal section of Penn Treebank was used. In both vision and NLP tasks, this algorithm beat, or came close to the state-of-art approaches. For a more detailed report on the results, see the original paper.

8. Socher (2013) Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

This review is based on Socher et al. [2013].

8.1. Introduction and Literature Overview

In this article, the authors introduce the *Stanford Sentiment Treebank*; the first database for texts with fully labeled parse trees. This database can be used to train sentiment analysis models. They also present the *Recursive Neural Tensor Network*(RNTN) which is a new model trained on the aforementioned database that can outperform previous methods in sentiment analysis and classification tasks.

In the context of sentiment analysis, the most prominent methods for representing texts are usually based on the *bag of words* model to represent the features using a fixed-size vector(Pang and Lee [2009]). These methods often use co-occurrence statistics of a word and its context to describe each word(Turney and Pantel [2010]). As expected, representations based on the bag of words model are not particularly useful in sentiment analysis as they may even fail to capture the differences in antonyms. Their performance drops considerably when faced with complex, composite sentences as they do not preserve order. One alternative is to use neural word vectors(Bengio et al. [2003]). Their application to specific tasks such as sentiment detection has been explored before(Socher et al. [2011]).

The idea of using compositionality in sentiment vector spaces is another useful idea. Among many others, Mitchell and Lapata [2010] analyzed similarities computed by vector addition, multiplication, etc, for two word phrases. Another useful idea was proposed in Yessenalina and Cardie [2011] where the authors computed matrix representations for larger phrases and defined composition as matrix multiplication. RNTN is a compositional model which uses a tensor to encode the effects of composition. Relating inputs through three way interactions using a tensor has been proposed before, such as in Sutskever et al. [2009] and Yu et al. [2012].

8.2. Main Results

The accuracy of binary sentiment classification of sentences has not passed 80% and for more complicated cases when there are several classes, the accuracy is even lower for short messages(Wang et al. [2012]). This can be attributed to the bag of words representation of sentences(which is used in almost all classifiers) as the order of the words in ignored in this representation. The Stanford sentiment bank, built upon the database in Pang and Lee [2005], uses the Stanford parser(Klein and Manning [2003]) to parse the sentences and then labels the resulting phrases with the help of Amazon Mechanical Turk. It was observed that a 5-class classification of phrases into categorize negative, somewhat negative, neutral, somewhat positive, positive, captures the main variability of the labels. The classification model that is presented uses this database to train and predict the label of unseen sentences and phrases from these five options.

The basic idea behind recursive neural models for sentiment analysis is to compute a compositional vector representation for variable-length phrases and then uses this representation as features that are used to classify each phrase. Thus, given an n-gram, these models first parse it into a binary tree where each leaf corresponds to a single word. Each word is then represented as a vector. the model proceeds by computing parent vectors from its children in the parse tree, using different compositional functions g. In order to train the model, these parent vectors are given to a classifier that predicts their label. The vector representation of words and the classifier is identical in different models. In the beginning, each word is represented as a d-dimensional random vector. By putting all of the word vectors together, the word embedding matrix $L \in \mathbb{R}^{d\times|V|}$ is produced($|V|$ is the size of the vocabulary). This initially random matrix is a parameter that will be learned along the compositionality models. Given a d-dimensional vector a, the label for its corresponding phrase is computed by a $\mathrm{softmax}$ classifier:

$$y^a = \mathrm{softmax}(W_s a),$$

where $W_s \in \mathbb{R}^{5\times d}$ is the sentiment classification matrix that has to be learned.

Knowing the word vector representation and the classifier, different compositional functions g have been proposed:

8.2.1. Recursive Neural Network(RNN)

RNN (Goller and Kuchler [1996], Socher et al. [2011]) computes the parent representation p given its children's vectors u, v(in order) as

$$p = f\left(W\begin{bmatrix} u \\ v \end{bmatrix}\right),$$

where $f = \tanh$ is a non-linear function that is applied to each element of its input $W \in \mathbb{R}^{d\times 2d}$ is the parameter matrix that has to be learned. The vector computed at each step is given to the classifier to train the model and is also used to compute vector representation of its parent.

8.2.2. Matrix-Vector Recursive Neural Network(MV-RNN)

MV-RNN(Socher et al. [2012]) uses both a d-dimensional vector *and* a $d \times d$ matrix to represent every word and phrase. When two elements are combined, the matrix of one is multiplied by the vector of the other and vice versa. More concretely, if u, U and v, V are vectors and matrices corresponding to two children of a parent node, respectively, the vector and matrix of the parent is computed from the following equations:

$$p = f\left(W\begin{bmatrix} Vu \\ Uv \end{bmatrix}\right),\ P = f\left(W_M\begin{bmatrix} U \\ V \end{bmatrix}\right),$$

where $W, W_M \in \mathbb{R}^{d\times 2d}$ are parameters of the model and $f = \tanh$ is the non-linearity. Notice that $P \in \mathbb{R}^{d\times d}$, $p \in \mathbb{R}^d$ are matrix and vector representations that belong to the parent of the two children. The vector representation is used in a similar way to the regular RNN to train a classifier.

8.2.3. Recursive Neural Tensor Network(RNTN)

The main problem with MV-RNNs is that the number of their parameters become very large as the vocabulary grows(recall that the vector and matrix representation of single words are parameters of the model). The RNTN model uses a single, tensor-based composition function for all nodes and therefore, significantly reduces the number of parameters. Therefore, to compute the vector representation of a parent, p, from vectors of its children, a, b, one uses the following relation:

$$p = f\left(\begin{bmatrix} a \\ b \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} a \\ b \end{bmatrix} + W \begin{bmatrix} a \\ b \end{bmatrix}\right),$$

where $V^{[1:d]} \in \mathbb{R}^{2d \times 2d \times d}$ is a tensor and W is defined as in the previous models. The basic RNN is therefore a special case of RNTN in which $V = 0$. The main advantage of RNTN over the simple RNN is that input vectors are directly related via the tensor.

8.2.4. Optimization

Assume that the classifier classifies the input vectors into C classes by outputting a predicted distribution y^i at node i, where the ground truth(target) is t^i. The error(loss) function for the RNTN is defined as the cross-entropy error between the predicted distribution $y^i \in \mathbb{R}^C$, $t^i \in \mathbb{R}^C$. Thus,

$$E(\theta) = \sum_i \sum_j t_j^i \log y_j^i + \lambda ||\theta||^2,$$

where $\theta = (V, W, W_s, L)$ are parameters of the model.

The derivative of $E(\theta)$ with respect to the weights of the classifier, W_s is calculated normally. In order to compute the derivative with respect to other parameters, notice that each node backpropagates its error through the recursively used weights V, W. Let x^i denote the vector at node i. The softmax error vector at node i, is defined as

$$\delta^{i,s} = \left(W_s^T \left(y^i - t^i\right)\right) \otimes f'\left(x^i\right),$$

where $\otimes$ is the Hadamard product between two vectors and f' is the element-wise derivative of f which is generally the function tanh. Now, to compute the full derivative of V and W, one must sum the derivatives at each of the nodes. Let $\delta^{i,\text{com}}$ be the complete incoming error message for a node i. It is clear that the root of the parse tree only receives error from the softmax function that is applied to its vector. Hence, $\delta^{\text{root,com}} = \delta^{\text{root},s}$. Knowing this, the derivative for W(at the root) will be computed by normal backpropagation. The derivative for each "slice" of the tensor $V^{[k]}$ can also be computed in a similar manner:

$$\frac{\partial E^{\text{root}}}{\partial V[k]} = \delta_k^{\text{root,com}} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}^T \quad \forall k = 1, \cdots, d.$$

(a, b are children of root)
Now, the error message from root to its two children is:

$$\delta^{\text{root,down}} = \left(W^T \delta^{\text{root,com}} + S\right) \otimes f'\left(\begin{bmatrix} a \\ b \end{bmatrix}\right),$$

where

$$S = \sum_{k=1}^{d} \delta_k^{\text{root,com}} \left(V^{[k]} + \left(V^{[k]}\right)^T\right) \begin{bmatrix} a \\ b \end{bmatrix}.$$

Each children of root will in turn take half of this vector and add it to their own softmax error and use it to compute their derivative and the error message the should be propagated down to their children. The authors used AdaGrad(Duchi et al. [2011]) in their implementation to optimize the error function.

In the experiments done by the authors, this model when trained on the previously mentioned Stanford sentiment treebank, out-performs the state of art algorithms in binary classification of sentences by a 5.4% margin. After more selective examinations, it became evident that RNTN achieves a higher accuracy in comparison to other methods, in analysing complex sentences, such as contrastive conjunctions or negations of negative sentences.

9. Kearns (2013) Machine Learning for Market Microstructure and High Frequency Trading

This is a review of Kearns and Nevmyvaka [2013].

9.1. Introduction and Literature Overview

In this article, the authors explore the applications of machine learning, in particular reinforcement learning(RL), in discovering microstructures in the market and high frequency trading. High frequency trading(HFT) presents its own unique challenges which require specialized machine learning methods to solve. Granularity of the available data is one such challenge, which makes feature selection even harder. In this article, three important problems in HFT are discussed and for each, a machine learning solution is presented. These problems are

1. optimized trade execution(Nevmyvaka et al. [2006]),
2. predicting price movement and
3. optimized execution in dark pools via censored exploration(Ganchev et al. [2010]).

The above problems are among the most important issues a high frequency trader has to deal with. Each of these problems are discussed separately in the following sections.

Because high frequency trading is a relatively new phenomenon, there are only a few published works where machine learning methods are used in HFT. Two of the case studies presented in this article are based on the previous works of the authors. Despite the lack of empirical work on these types of problems, there have been several theoretical works on similar problems. For instance, the execution problem has been studied extensively before, as in Kharroubi and Pham [2010], Bertsimas and Lo [1998], Guéant et al. [2012]. Similarly, there is a decent amount of literature on dark pools. Ganchev et al. [2010] is the original work of the authors which was extended in Agarwal et al. [2010] to adversarial scenarios. Another extension is used by JP Morgan, which adds another feature and updated the distributions more aggressively. Of course, these are only some of the previous work that was done to solve similar problems to those that will be discussed here.

9.2. High Frequency Data

It is widely acknowledged that the data used in HFT is the most granular data available. They include every order that is placed, executed and cancelled in an exchange. These usually allow for a total reconstruction of the full limit order book. This type of data raised two major challenges: their scale and interpretation. Storing a single days amount of data for a highly liquid stock such as AAPL requires several gigabytes of storage. Thus, storing historic data for a meaningful period of time is a major technical challenge. Interpretation of this data is even more difficult. It is estimated that over 90 per cent of placed orders, are cancelled. Hence, extracting meaningful and informative features from the limit order book in such an environment is far from trivial.

The problem of feature selection is not specific to HFT and comes up in virtually every application of ML. For instance, natural language processing(NLP) and computer vision, two of the most popular applications of ML, also face the same problem. Although in NLP a word is the smallest unit of meaning , there is no clear “smallest unit of meaning” in an image. Therefore, feature selection in computer vision is considered much harder. Computer vision is somewhat similar to HFT in this regard; in both cases, the available data is so granular that finding meaningful features becomes a serious challenge. In each of the following case studies, some proposed features are presented and discussed.

9.3. Optimized Trade Execution

Assume that we want to buy exactly V shares of a particular stock, within T time steps, while minimizing the share prices for doing so. This is a simple form of optimized trade execution problem that is considered in this section. Reinforcement learning is used to solve this problem. RL methods use a state representation to decide which actions they should take. A simple choice for the state is the pair (v, t) where v is the remaining volume that should be bought and t is the number of time steps left. Intuitively, in this state space, we should buy more aggressively if t is low and v is high. Along with the remaining volume and time, we further enrich the state space with the following features that are obtainable from the limit order book:

- **Bid-Ask Spread:** A positive value that indicates the difference between the bid and ask price in the current time.

- **Bid-Ask Volume Imbalance:** The number of shares at the bid minus the number of shares at the ask side in the current time.
- **Signed Transaction Volume:** The number of shares bought in the last 15 seconds minus the number of shares sold in the last 15 seconds.
- **Immediate Market Order Cost:** The cost we would pay if we immediately buy all the remaining shares with a market order.

All of these values are normalized by subtracting the historical mean value and dividing by the historical standard deviation. All of these new features seem informative and useful in the process of decision making nevertheless, the authors ran several experiments to determine the significance of each of these features. They trained and tested the RL model using the basic state (v, t) augmented with different combination of the above features. The percentage reduction in trading costs(implementation shortfall) measures the significance of adding a feature(or a subset of features). With the exception of bid-ask volume imbalance, all features yielded a significantly better strategy. Adding all three features (except the bid-ask volume imbalance) to the state space resulted in an almost 13% reduction in trading costs. They successively added these features to the state representation and trained and tested the RL model with them.

The action space for the RL model consists of placing a limit order for all of the remaining volume at some price. Thus, at each moment, the agent has a single outstanding order, but repositions it according to the state information. The RL agent must also have a reward(or cost) function that given a state and an action, produces a feedback to the agent. In this model, the cost of an action in a particular state is defined as the expenditures from the (partial) excution of the order.

The model uses a tabular representation of the state space and produces a deterministic policy. After the model is trained, the performance of the resulting policy is evaluated by comparing it to the optimal *one-shot* submission strategy, where a single limit order for the entire target volume is placed in the beginning of trading period, and at the end of the trading period, all of the remaining volume is bought using a market order.

The results of training and testing this agent on multiple stocks are presented in the paper. The RL policy significantly outperformed the optimal one-shot submission strategy on all stocks. It was also observed that finer discretizations improved the performance, which indicates that there was enough data to avoid overfitting to a large state space. Further analysis of the resulting policies showed similarities between the policies obtained for different stocks, and all of them seemed in line with the intuitive idea of "buying more aggressively when there is little time and a lot of shares to buy". This indicates the robustness of the learned policies.

9.4. Predicting Price Movement

If the price movements can be predicted from the order book, one can immediately devise a profitable strategy based on it. This problem is therefore of significant interest to high frequency traders. Thus, devising a profitable HFT strategy can be divided into two parts; first, the development of features that allow the prediction of directional price movements and then the development of a learning algorithm for execution that is able to profit from this alpha at a sufficiently low trading cost. Notice that in the previous section, features that may be helpful in predicting the directional movement of stock price were not used, as a trading volume was imposed on the agent. Here, in addition to bid-ask spread, bid-ask volume imbalance and signed transaction volume, that were introduced in the previous section, additional features are considered:

- **Price:** Measures the recent directional movement of executed prices.
- **Smart Price:** The average of bid and ask price, weighted by their inverse volume.
- **Trade Sign:** Measures whether buyers or sellers crossed the spread more frequently in recent executions.

These features are also normalized and discretized using their historic mean and standard deviation, similar to the previous section.

The authors considered a simplistic action space that contains only two action: i) buying a share at the bid-ask midpoint, holding it for t seconds and selling it at again, the bid-ask midpoint and ii) selling a share at the bid-ask midpoint, and buying it back after t seconds at the bid-ask midpoint. Different values of t were considered in the experiments.

To train the RL agent on a particular stock, the order book of 2008 was reconstructed. Using the order book, the current state and the profit or loss of each action can be computed and used for simulation. Similar to the previous section, the learned policy is stored in a tabular fashion and is learned by simulating the 2008 train data and finding the better action at each time. In order to test the policy, the 2009 data was used and the overall profitability of this(deterministic) policy in 2009 was computed.

The two most important findings of running this experiment on different stocks and comparing the results was that the learned policies were consistently profitable on test set and that policies among different stocks are broadly similar. To demonstrate this second finding, the authors plotted the correlation between the value of each feature and the action learned, for all 19 stocks that were considered. It was observed that for nearly all features the sign of the correlation is the same among different stocks, while their numerical value varied significantly across different stocks. This again is an indication of the robustness of the learned policies, as they are all qualitatively similar, while significant quantitative optimization fine tuned them for each stock.

The importance of each feature was also analyzed using the same technique that was used in the previous section(the model was trained using only one additional feature and its profitability was measured). The two important observations from this analysis were:

1. Profitability is usually better when all additional features are used.
2. Smart price appears to be the best single feature and is often slightly more profitable than using all features combined. This is an indication of a slight overfitting. However, it is worth noting that using all features appears to reduce the variance of the policy and prevents major losses.

Furthermore, comparing the learned policies across different holding periods(time t) resulted in some more interesting observations. It turns out that for very short holding periods(t in the order of milliseconds to seconds) the policies exploit a momentum based market and expect the prices to continue their behaviour in the past. This is in sharp contrast to the results for longer holding periods(t in the order of dozens of seconds to minutes). In this case, the policies are mostly *reversion strategies* and prefer buying when the recent price movement has been downward. When the holding time is even longer(t in the order of hours) the learned policies just capture the overall price movement and are not particularly profitable. For example, if the overall movement of price is downwards during the course of training, the learned policy simply sells at every state! This indicates that on longer time scales, the selected features are not informative and lose their explanatory power. In order to reduce the effects of long-term and overall price movements, the authors suggest using the relative profitability of buying versus *buying in every state*. This kind of normalization of rewards, proved fruitful in the experiments and reduced the effects of overall price movements. In the end, the authors point out to the average amount of profit that this agent can gain is much less than transaction costs. Thus, these strategies will not be profitable in the real world. They suggest three approaches to increase the ratio of profit to transaction cost. One can train agents that hold the positions longer. This has the drawback that micro structure features lose their significance in larger time scales, as was mentioned earlier. Another option is to use limit orders instead of market orders to reduce the transaction costs. The third and final option is to use better, more informative features to obtain more significant profits.

9.5. Smart Order Routing in Dark Pools

A dark pool is a private forum for trading different assets where liquidity is more important than price improvement. When a trader is satisfied with paying the midpoint price in exchange for faster liquidation of his assets, a dark pool may be used instead of lit exchanges, where the market impact and transaction costs may be higher and there might not be enough liquidity. In a simple dark pool, orders only specify the desired volume of an asset and the direction of the trade but not the price. Orders are queued according to their time of arrival on the buy and sell side, and if at any moment both queues were non-empty, execution occurs. The price of execution depends on the current prices in the corresponding lit (limit order) market for the desired stock(e.g, it might be the bid-ask midpoint).

Smart order routing(SOR) problem is as follows: Assume that there are n dark pools available, each of which may offer different liquidity profile for a given stock. More precisely, for each $i = 1, \cdots, n$, there is a probability distribution P_i over non-negative integers, such that whenever an order for V_i shares is submitted to the ith pool, a random value S_i is drawn from P_i and $\min(S_i, V_i)$ shares will be executed. Now, assume that we want to buy(or sell) a total of V shares. The SOR problem is to determine how many of the total V shares should we submit to each pool, such that the expected value of the executed shares is maximized.

There are two noticeable differences between dark pools and lit markets. First, there is much less data available from a dark pool as we only observe our own orders execution data and have no idea about market activity, liquidity

imbalances, etc. that are observable from a limit order book. Second, if our order is completely executed, we have no information on how much additional liquidity is still available in the pool. In other words, we can only observe $\min(V, S)$, but not S itself. This is known as *censoring* in statistics.

If the distributions P_i are known, it can be proved that a simple greedy algorithm that assigns each share to the pool with the highest probability of executing a single share(conditioned on the previous submissions), is optimal(Ganchev et al. [2010]). When the distributions are not known a priori, the authors suggest a two step process of optimization and re-estimation. There they assume a parametric model for P_is and uses the *Kaplan-Meier estimator* to estimate its parameters and update our current belief on P_is. Then for each desired volume, the algorithm behaves as if it knows the true distributions and upon observing the results of executions, updates P_is again. They also prove that under some simple assumptions, this algorithm quickly converges to the optimal algorithm. They also provide empirical evidence that this convergence is indeed very fast and that this algorithm outperforms other benchmark algorithms.

10. Author (year) Title

10.1. Introduction and Literature Overview

10.2. Main Theoretical Results

11. Author (year) Title

11.1. Introduction and Literature Overview
11.2. Main Theoretical Results

12. Author (year) Title

12.1. Introduction and Literature Overview
12.2. Main Theoretical Results

13. Author (year) Title

13.1. Introduction and Literature Overview
13.2. Main Theoretical Results

14. Author (year) Title

14.1. Introduction and Literature Overview
14.2. Main Theoretical Results

15. Author (year) Title

15.1. Introduction and Literature Overview
15.2. Main Theoretical Results

16. Bailey (2012) Sharpe Ratio Efficient Frontier

This review is based on Bailey and Lopez de Prado [2012]

16.1. Introduction and Literature Overview

16.1.1. Abstract

This article calculate the probability of a Sharpe ratio exceeds a certain amount when distributions are not normal. This new investor's instrument that is called probabilistic Sharpe ratio have a lot of benefits. It provides us the track record length needed whether to accept or reject a hypothesis that the reported Sharpe ratio is below a certain threshold for a specific confidence interval. It also models the trade off between the track record length and undesirable statistical features. It allows us to compute the Sharpe ratio efficient frontier (SEF) by which we can optimize a portfolio with non-normal distribution.

16.1.2. Introduction

Roy [1952] was the one person to offer the ratio of returns and risk but it was Sharpe [1966] who applied his idea in to Markowitz's mean-variance frame work that is still one of the best performance evaluation metrics. De Prado and Peijan [2004] showed that the assumption of IID normal may hide the drawdown risks of investments which important for hedge funds. Many prominent researchers tried to convince this ratio in spite of its unrealistic assumptions and until today it is one of the best performance metrics. One of its flaws is it is very sensitive to statistical characteristics of returns that make the SR inflated. The goals of this paper are to provide another performance measure to consider these unfavorable traits. This measurement which we call it the probabilistic Sharpe Ratio needs longer track to compensate for those statistical characteristics which cause the SR become inflated. Moreover, we show that the SR is still a good measurement if we have a sufficient length for track record and the concept of minimum track record length (MinTRL) is introduced for rejecting the null hypothesis that the SR is below a certain level. The third purpose of this paper is to introduce the concept of Sharpe Ratio Efficient Frontier (SEF) which enables us to find optimal portfolio under the assumption of non-normal returns. This portfolio maximization approach is different to the approaches which accounts for the higher moments of the distribution because we do not want to put any further assumption on the weightings that higher moments have in utility function. The investors will find it useful because they use SR as their utility function. The SEF can be interpreted as the portfolios which have the highest expected SR. This paper uses the frame work of Mertens [2002] that assumes IID non-normal returns. and also it says that style and skill of the portfolio manager would not change within the observation period. Fortunately this assumptions are confirmed by Opdyke [2007], he says that Merten's equation is valid under more general assumption of stationary and ergodic returns.

In López de Prado and Foreman [2011], readers interested in the estimation of the parameters that characterise a mixture of 2 Gaussians can find an effective algorithm. Goetzmann et al. [2007] demonstrate that the inflationary effect of certain methods and techniques on the Sharpe ratio is more often decreased by sampling returns. Christie [2005] uses a GMM-based method to derive a distribution that only assumes stationary and ergodic returns, enabling conditional variance, serial correlation and non-IID returns. Since hedge fund strategies typically have negative skewedness and fat tails (Brooks and Kat [2002], López de Prado and Rodrigo [2004]), estimated Sharpe ratios are often inflated.

16.1.3. Frame Work

The Sharpe Ratio itself is not a complete measurement for performance because of it unrealistic assumptions like being normally distributed. This article shows that by relaxing the normality assumption the SR may suffer from the large variance that makes us to not trust its accuracy. Now suppose that excess return distribution is normal with mean, mho and standard deviation sigma. the SR would equal to mho over sigma which neither mho, nor sigma are not known for sure. So the true SR can not be estimated certainly and SR is prone to errors and we can determine them by putting some assumptions. If we assume that returns are IID normal then the estimated SR have normal distribution with the mean of SR and variance which is depend on the SR value and number of observations that says investors prefer investment with longer track record the more the number of observation, the smaller the variance of the SR. The SR does not characterize a distribution of returns, in the sense that there are infinite Normal distributions that deliver any given SR. It is rather obvious from the formula of SR that by rescaling the distribution we can obtain the same SR as before. It is proved that any two normal distribution can generate infinite number of skewness and kurtosis with equal SR. It means that no matter how high is SR, this measurement cannot shows severe losses. An important conclusion is drawn from figure one that shows the combination of two normal distribution can generate infinite number of skewness and kurtosis with the same sharp ratio. Financial data has a trade of feature between skewness and kurtosis that the

higher the absolute value of skewness the greater the level of kurtosis. This could turn to serious problems for hedge funds when the distribution has large negative skewness with high kurtosis. So for a risk averse person the SR does not provide the sufficient ranking for preferences unless we include non-normality into account and to find out that skewness and kurtosis is accurate enough we generate 1000 sample observation for each mixture and estimate the skewness and kurtosis for them. The result is that in the logical ranges of moments these estimated moments were pretty accurate in comparison to those actual mixtures. Figure 4 shows that the estimation error of mean is not a function of mean value. The standard deviation estimator is biased toward underestimating risks, but it is not affected by the standard deviation itself. But the estimation error for the third and forth moments are biased and affected by the scaling. All these findings prove that estimating for the higher moments need longer sample length.

16.2. Main Theoretical Results

16.2.1. Incorporating Non-Normality

In the previous section we saw that non-normal distributions with very different parameters and characteristics could have the same SR. This section shows that although skewness and kurtosis do not have impact on the SR itself but they affects the confidence interval and statistical significance immensely. Mertens [2002] showed that by dropping the normality assumption the distribution of SR are still normal but on the other hand the skewness and kurtosis have severe impact on SR and make it become inflated. The estimated standard deviation of estimated SR is like: estimated stand deviation: With the significance level alfa: Eq9:

$$\text{Prob}\left[SR \in \left(\widehat{SR} - Z_{\alpha/2}\hat{\sigma}_{\widehat{SR}}, \widehat{SR} + Z_{\alpha/2}\hat{\sigma}_{\widehat{SR}}\right)\right] = 1 - \alpha$$

16.2.2. Probabilistic Sharpe Ratio (PSR)

Now we can introduce the probabilistic SR with a given benchmark SR: Eq10:

$$\widehat{PSR}\left(SR^*\right) = \text{Prob}\left[\widehat{SR} > SR^*\right] = 1 - \int_{-\infty}^{SR} \text{Prob}(\widehat{SR}) \cdot d\widehat{SR}$$

The probability of SR bigger than a benchmark SR is calculated as: Eq11:

$$\widehat{PSR}\left(SR^*\right) = Z\left[\frac{\left(\widehat{SR} - SR^*\right)\sqrt{n-1}}{\sqrt{1 - \hat{\gamma}_3\widehat{SR} + \frac{\hat{\gamma}_4 - 1}{4}\widehat{SR}^2}}\right]$$

From the above equation we can find out that PSR have positive connection with estimated SR, positive skewness, and longer track record but it has negative connection with kurtosis. Prado shows that hedge funds return distributions have negative skewness and large amount of kurtosis that this measurement take these characteristics in to account and give us the deflated SR. Another important matter is some strategies are weekly or monthly and others are annually so the estimates SR is somewhat undetermined so in these cases the PSR is a better measure since it would not change through calendar and is merely dependent to probabilities so investors prefer PSR over classical SR. If a reacher believes that her estimation of skewness and kurtosis are associated with errors she can assign a lower bound for skewness and an upper bound for kurtosis but if her estimation are pretty accurate she does not need to consider these worst case scenarios. For example the information of a hedge fund is reported in the feature 6, based on these information we cannot reject the null hypothesis of whether this strategy is skill-less or not if assume the distributions are normal, but after accounting for non-normality we can see that the null hypothesis of being skill-less cannot be rejected! In other words, by assuming that distributions are normal we are simply neglecting critical information of potential losses. The SR of 1.59 is not worthless completely and in fact if we raise the length of the track record from 2 years in to 3 years the estimated PSR would have been 0.953 large enough to reject the null hypothesis of being skill-less, so the track length could be a compensation for uncertainty which is originated from non-normal returns. It is important to mention that the PSR does not try to consider the higher moments on the preferences and in fact the investor herself decide to consider them and they are still care for mean and variance of returns distributions. The old important question is that what is the sufficient length of track record in order the estimated sharp ratio is above a given threshold? The minimum

length is computed in the equation 13 and if the track record is smaller than the minimum track record (MinTRL) we cannot be sure that the estimated SR is significant at a given significance level. Eq13:

$$\text{MinTRL} = n^* = 1 + \left[1 - \hat{\gamma}_3 \widehat{SR} + \frac{\hat{\gamma}_4 - 1}{4} \widehat{SR}^2\right] \left(\frac{Z_\alpha}{\widehat{SR} - SR^*}\right)^2$$

Another important matter is that the portfolio manager is penalized with the longer track record if she assume the normality of distributions. All above concept can be understood better by the numerical examples, in the figure 8 the MinTRL is calculated for different estimated SRs and benchmark SR with assumption of daily returns are normal and IID. For the estimated SR of 2 to be greater than 1 in 95 percent confidence level, we need 2.73 years of daily track records. If we want to evaluate the MinTRL for weekly returns we need 2.83 years of track records of normal and IID weekly returns. And if we change the returns from daily to monthly returns the MinTRL would be 3.24 years. This observations show us that the MinTRL increase just with the decrease in the frequency. If we relax the assumption of normality by using the actual data of HFR hedge fund we could easily see that the MinTRL for monthly returns that are not normal increase to 4.99 years which is 54 percent longer than the monthly returns and 82 percent longer than daily returns with normal distributions. Now it is time to test the model with real data of hedge funds. The data which is used contain 134 month observations or 11.167 years from of January 1st 2000 to May 1st 2011. The PSR (0) greater than 0 in confidence level of 95 percent means that SR is greater than zero and it is the same for PSR (0.5) and so on. It needs to be mention that because of our sample data length, the MinTRL for any PSR is smaller than 11.167. The calculations also indicate that the majority of investments are skillful and their SR are greater than zero but there are only 9 hedge funds with PSR greater than 0.5.

References

[1] Alekh Agarwal, Peter Bartlett, and Max Dama. Optimal allocation strategies for the dark pool problem. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 9–16. JMLR Workshop and Conference Proceedings, 2010.

[2] Susan Athey. Machine learning and causal inference for policy evaluation. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 5–6, 2015.

[3] David H Bailey and Marcos López de Prado. The deflated sharpe ratio: correcting for selection bias, backtest overfitting, and non-normality. *The Journal of Portfolio Management*, 40(5):94–107, 2014.

[4] David H Bailey and Marcos Lopez de Prado. The sharpe ratio efficient frontier. *Journal of Risk*, 15(2):13, 2012.

[5] Richard Bellman and Robert Kalaba. Dynamic programming and statistical communication theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(8):749, 1957.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[7] Dimitris Bertsimas and Andrew W Lo. Optimal control of execution costs. *Journal of Financial Markets*, 1(1):1–50, 1998.

[8] Stephen Boyd, Enzo Busseti, Steven Diamond, Ronald N Kahn, Kwangmoo Koh, Peter Nystrup, and Jan Speth. Multi-period trading via convex optimization. *arXiv preprint arXiv:1705.00109*, 2017.

[9] Chris Brooks and Harry M Kat. The statistical properties of hedge fund index returns and their implications for investors. *The Journal of Alternative Investments*, 5(2):26–44, 2002.

[10] Daniel S Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. *arXiv preprint arXiv:1904.06387*, 2019.

[11] Daniel S Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Conference on Robot Learning*, pages 330–359, 2020.

[12] Steve Christie. Is the sharpe ratio useful in asset allocation? *Macquarie Applied Finance Centre Research Paper*, 2005.

[13] Marcos Lopez de Prado. *Machine Learning for Asset Managers*. Cambridge University Press, 2020.

[14] Marcos M Lopez De Prado and Achim Peijan. Measuring loss potential of hedge fund strategies. *The Journal of Alternative Investments*, 7 (1):7–31, 2004.

[15] Matthew F Dixon, Igor Halperin, and Paul Bilokon. *Machine Learning in Finance*. Springer, 2020.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[17] Anne Haubo Dyhrberg. Bitcoin, gold and the dollar–a garch volatility analysis. *Finance Research Letters*, 16:85–92, 2016.

[18] David Easley, Marcos M López de Prado, and Maureen O'Hara. Flow toxicity and liquidity in a high-frequency world. *The Review of Financial Studies*, 25(5):1457–1493, 2012.

[19] Efi Efrati, Zhe Wang, Amy Kolan, and Leo P Kadanoff. Real-space renormalization in statistical mechanics. *Reviews of Modern Physics*, 86 (2):647, 2014.

[20] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[21] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016.

[22] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*, 2015.

[23] Kuzman Ganchev, Yuriy Nevmyvaka, Michael Kearns, and Jennifer Wortman Vaughan. Censored exploration and the dark pool problem. *Communications of the ACM*, 53(5):99–107, 2010.

[24] Nicolae Gârleanu and Lasse Heje Pedersen. Dynamic trading with predictable returns and transaction costs. *The Journal of Finance*, 68(6): 2309–2340, 2013.

[25] Mike Gashler and Tony Martinez. Temporal nonlinear dimensionality reduction. In *The 2011 International Joint Conference on Neural Networks*, pages 1959–1966. IEEE, 2011.

[26] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[27] William Goetzmann, Jonathan Ingersoll, Matthew Spiegel, and Ivo Welch. Portfolio performance manipulation and manipulation-proof performance measures. *The Review of Financial Studies*, 20(5):1503–1546, 2007.

[28] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352. IEEE, 1996.

[29] Peter Gomber, Jascha-Alexander Koch, and Michael Siering. Digital finance and fintech: current research and future research directions. *Journal of Business Economics*, 87(5):537–580, 2017.

[30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[31] Abhijit Gosavi. Finite horizon markov control with one-step variance penalties. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1355–1359. IEEE, 2010.

[32] Stephen Gould, Richard Fulton, and Daphne Koller. Decomposing a scene into geometric and semantically consistent regions. In *2009 IEEE 12th international conference on computer vision*, pages 1–8. IEEE, 2009.

[33] Jordi Grau-Moya, Matthias Krüger, and Daniel A Braun. Non-equilibrium relations for bounded rational decision-making in changing environments. *Entropy*, 20(1):1, 2018.

[34] A Graves. Supervised sequence labelling with recurrent neural networks [ph. d. dissertation]. *Technical University of Munich, Germany*, 2008.

[35] Jonathan Gryak, Robert M Haralick, and Delaram Kahrobaei. Solving the conjugacy decision problem via machine learning. *Experimental Mathematics*, 29(1):66–78, 2020.

[36] Olivier Guéant, Charles-Albert Lehalle, and Joaquin Fernandez-Tapia. Optimal portfolio liquidation with limit orders. *SIAM Journal on Financial Mathematics*, 3(1):740–764, 2012.

[37] Abhinav Gupta and Larry S Davis. Beyond nouns: Exploiting prepositions and comparative adjectives for learning visual classifiers. In *European conference on computer vision*, pages 16–29. Springer, 2008.

[38] Igor Halperin. Qlbs: Q-learner in the black-scholes (-merton) worlds. *The Journal of Derivatives*, 2020.

[39] Igor Halperin and Matthew Dixon. "quantum equilibrium-disequilibrium": Asset price dynamics, symmetry breaking, and defaults as dissipative instantons. *Physica A: Statistical Mechanics and its Applications*, 537:122187, 2020.

[40] Igor Halperin and Ilya Feldshteyn. Market self-learning of signals, impact and optimal trading: Invisible hand inference with free energy (or, how we learned to stop worrying and love bounded rationality). *Impact and Optimal Trading: Invisible Hand Inference with Free Energy (Or, How We Learned to Stop Worrying and Love Bounded Rationality)(May 7, 2018)*, 2018.

[41] Hado V Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[42] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[43] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

[44] Derek Hoiem, Alexei A Efros, and Martial Hebert. Putting objects in perspective. *International Journal of Computer Vision*, 80(1):3–15, 2008.

[45] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[46] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[47] Michael Kearns and Yuriy Nevmyvaka. Machine learning for market microstructure and high frequency trading. *High Frequency Trading: New Realities for Traders, Markets, and Regulators*, 2013.

[48] Idris Kharroubi and Huyen Pham. Optimal portfolio liquidation with execution cost and risk. *SIAM Journal on Financial Mathematics*, 1 (1):897–931, 2010.

[49] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*, pages 423–430, 2003.

[50] Ellen De Mello Koch, Robert De Mello Koch, and Ling Cheng. Is deep learning a renormalization group flow? *IEEE Access*, 8:106487–106505, 2020.

[51] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[52] Gerhard Kramer. *Directed information for channels with feedback*. Citeseer, 1998.

[53] Paul Langevin. Sur la théorie du mouvement brownien. *Compt. Rendus*, 146:530–533, 1908.

[54] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616, 2009.

[55] Shane Legg and Marcus Hutter. Tests of machine intelligence. In *50 years of artificial intelligence*, pages 232–242. Springer, 2007.

[56] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.

[57] Li-Jia Li, Richard Socher, and Li Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic

framework. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2036–2043. IEEE, 2009.

[58] Michelle Lochner, Jason D McEwen, Hiranya V Peiris, Ofer Lahav, and Max K Winter. Photometric supernova classification with machine learning. *The Astrophysical Journal Supplement Series*, 225(2):31, 2016.

[59] M López de Prado and M Foreman. Exact fit for a mixture of two gaussians: The ef3m algorithm. *RCC at Harvard University, Working paper. http://ssrn. com/abstract*, 1898783, 2011.

[60] M López de Prado and C Rodrigo. Invertir en hedge funds. *Madrid: Díaz de Santos*, 2004.

[61] Marcos Lopez de Prado. Beyond econometrics: A roadmap towards financial machine learning. *Available at SSRN 3365282*, 2019.

[62] Marcos Lopez de Prado. Ten applications of financial machine learning. *Available at SSRN 3365271*, 2019.

[63] Elmar Mertens. Comments on variance of the iid estimator in lo (2002). In *University of Basel Working Paper*. 2002.

[64] Robert C Merton. Optimum consumption and portfolio rules in a continuous-time model. In *Stochastic Optimization Models in Finance*, pages 621–661. Elsevier, 1975.

[65] Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429, 2010.

[66] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[67] Susan A Murphy. A generalization error for q-learning. *Journal of Machine Learning Research*, 6(Jul):1073–1097, 2005.

[68] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680, 2006.

[69] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

[70] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.

[71] John Douglas JD Opdyke. Comparing sharpe ratios: so where are the p-values? *Journal of Asset Management*, 8(5):308–336, 2007.

[72] Pedro A Ortega and Daniel A Braun. Thermodynamics as a theory of decision-making with information-processing costs. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120683, 2013.

[73] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *arXiv preprint cs/0506075*, 2005.

[74] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Comput. Linguist*, 35(2):311–312, 2009.

[75] Romain Paulus. Deep reinforced model for abstractive summarization, November 12 2019. US Patent 10,474,709.

[76] George Philipp and Jaime G Carbonell. Nonparametric neural networks. *arXiv preprint arXiv:1712.05440*, 2017.

[77] Andrew Rabinovich, Andrea Vedaldi, Carolina Galleguillos, Eric Wiewiora, and Serge Belongie. Objects in context. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[78] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. (approximate) subgradient methods for structured prediction. In *Artificial Intelligence and Statistics*, pages 380–387, 2007.

[79] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[80] Andrew Donald Roy. Safety first and the holding of assets. *Econometrica: Journal of the econometric society*, pages 431–449, 1952.

[81] Stuart Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

[82] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[83] William F Sharpe. Mutual fund performance. *The Journal of business*, 39(1):119–138, 1966.

[84] David Silver, Hado Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, pages 3191–3199. PMLR, 2017.

[85] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362 (6419):1140–1144, 2018.

[86] Herbert A Simon. Rational choice and the structure of the environment. *Psychological review*, 63(2):129, 1956.

[87] Richard Socher, Christopher D Manning, and Andrew Y Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 deep learning and unsupervised feature learning workshop*, volume 2010, pages 1–9, 2010.

[88] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y Ng, and Christopher D Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011.

[89] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 151–161, 2011.

[90] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1201–1211, 2012.

[91] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[92] Yhlas Sovbetov. Factors influencing cryptocurrency prices: Evidence from bitcoin, ethereum, dash, litcoin, and monero. *Journal of Eco-*

nomics and Financial Analysis, 2(2):1–27, 2018.

[93] E Miles Stoudenmire. Learning relevant features of data with multi-scale tensor networks. *Quantum Science and Technology*, 3(3):034003, 2018.

[94] Ilya Sutskever, Joshua Tenenbaum, and Russ R Salakhutdinov. Modelling relational data using bayesian clustered tensor factorization. *Advances in neural information processing systems*, 22:1821–1828, 2009.

[95] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[96] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[97] Joseph Tighe and Svetlana Lazebnik. Superparsing: scalable nonparametric image parsing with superpixels. In *European conference on computer vision*, pages 352–365. Springer, 2010.

[98] Stas Tiomkin and Naftali Tishby. A unified bellman equation for causal information and value in markov decision processes. *arXiv preprint arXiv:1703.01585*, 2017.

[99] Naftali Tishby and Daniel Polani. Information theory of decisions and actions. In *Perception-action cycle*, pages 601–636. Springer, 2011.

[100] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.

[101] Peter D Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188, 2010.

[102] Hal R Varian. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28, 2014.

[103] Hao Wang, Doğan Can, Abe Kazemzadeh, François Bar, and Shrikanth Narayanan. A system for real-time twitter sentiment analysis of 2012 us presidential election cycle. In *Proceedings of the ACL 2012 system demonstrations*, pages 115–120, 2012.

[104] Ainur Yessenalina and Claire Cardie. Compositional matrix-space models for sentiment analysis. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 172–182, 2011.

[105] Dong Yu, Li Deng, and Frank Seide. Large vocabulary speech recognition using deep tensor neural networks. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[106] Song-Chun Zhu and David Mumford. *A stochastic grammar of images*. Now Publishers Inc, 2007.