IMPERIAL

Constrained Deep Reinforcement Learning for Personalised Cancer Treatments

Author

Savraj Sian

CID: 01847921

Supervised by

Dr Wei Dai

Second Marker

Prof. Pier L. Dragotti

A Thesis submitted in fulfillment of requirements for the degree of
Master of Engineering in Electronic and Information Engineering

Department of Electrical and Electronic Engineering
Imperial College London

2024

Abstract

Recommending the most effective treatment is a key aspect of personalised cancer therapies, with drug ranking being central to this endeavour. This project aims to create the first drug ranking model that integrates constraints relevant to real-world considerations, specifically minimising side effects based on individual patient requirements. To this end, a ”severity score” is devised for each drug, and a previous paper is built upon to develop a deep reinforcement learning (DRL) model that incorporates this constraint. Due to the ranking objective, a novel method is presented to update the model according to this constraint, which is more suitable for this task, since conventional methods for constrained DRL are tailored to different objectives and lack precision in controlling constraint limits. To achieve this, Lagrangian, quadratic penalty, and augmented Lagrangian loss functions are proposed. By adopting this more holistic approach, the model becomes more relevant to the actual treatment strategies used by oncologists and emphasises the importance of incorporating such considerations into recommendation models. On a technical level, this framework aims to be among the first to use DRL for ranking while considering constraints. The results show that the proposed framework is able to successfully enforce the severity constraint and that the Lagrangian formulation is preferred in this situation.

Declaration of Originality

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have submitted, or will submit, an identical electronic copy of my final year project to the provided Blackboard module for Plagiarism checking.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

I have used ChatGPT v4 as an aid in the preparation of my report. I have used it to improve the quality of my English throughout and rephrase text originally written by me, however, all technical content and references comes from my original text.

Copyright Declaration

Acknowledgments

I would like to thank my supervisor, Dr Wei Dai, for his advice and guidance during this project. I would also like to thank Yiming Zhou for his help in this project too.

Contents

List of Acronyms

MDP Markov Decision Process

CMDP constrained Markov decision process

RL Reinforcement learning

DL deep learning

GNN graph neural networks

CNN convolutional neural networks

DRL deep reinforcement learning

PPO Proximal Policy Optimisation

NN neural network

AC actor-critic

PG policy gradient

GAE Generalised advantage estimation

GRU gated recurrent unit

GDSC Genomics of Drug Sensitivity in Cancer

CCLE Cancer Cell Line Encyclopaedia

DDPG Deep deterministic policy gradient

TD3 twin-delayed DDPG

TRPO Trust Region Policy Optimisation

DTR dynamic treatment regimes

SMILES simplified molecular-input line-entry system

KRL Kernelized Rank Learning

CaDRReS Cancer Drug Response prediction using a Recommender System

NDCG normalized discounted cumulative gain

DCG Discounted cumulative gain

IDCG ideal discounted cumulative gain

ADReCS Adverse Drug Reaction Classification System

ADR adverse drug reaction

ADMM alternating direction method of multipliers

CELU Continuously Differentiable Exponential Linear Unit

HPC high performance computing

ReLU Rectified linear unit

ELU Exponential Linear Unit

List of Figures

1 Introduction

Contents

1.1 Motivation

Cancer is a leading cause of death worldwide, with approximately 1 in 2 people in the UK expected to be diagnosed with cancer at some point in their lifetime. Therefore, efficient treatment methods are crucial, including the provision of personalised cancer treatments. This presents considerable challenges, which has fuelled an active area of research to solve this problem. A subset of this area is concentrated on ranking drugs according to their efficacy on specific cancer cell lines, and there are a multitude of methods to attempt this for individualised cancer treatments, with many of these methods relying on machine learning to learn the complex relationships between drugs and cancer cell lines.

The published research on this area naturally focusses on maximising the accuracy of the rankings, however, in reality the treatments administered by medical professionals will take into account numerous factors besides the theoretical most effective drug. The factor chosen for consideration for this project is the severity of drug side effects, as this can significantly affect the patient's quality of life. Depending on the current health status of the patient, recovery from

an especially toxic drug may be unlikely even if it would be the most effective way to eradicate their cancer. Alternatively, a patient may prefer treatments with reduced side effects due to personal preference, prioritising quality of life over maximum efficacy. Therefore, a drug "severity score" is developed to be minimised according to preference.

No existing papers have been found that attempt to take into account other considerations in their drug ranking, therefore, this project builds on the PPORank [1] paper to integrate side effect severity into the ranking through the drug severity score. PPORank utilises DRL to achieve its rankings, and the rationale behind using this methodology will be explained in Section 2.4. Although several papers utilise DRL for ranking, there appear to be none that apply constrained DRL for ranking, as this project intends to do.

1.2 Project Aims

This project aims to deliver a DRL model that is capable of generalising to unseen cancer cell lines and ranking candidate drugs in order of efficacy, whilst incorporating a drug severity constraint which can be adjusted such that it can be given varying importance in the rankings. The code is written in Python, using necessary libraries such as Pytorch, Numpy, Pandas. An unconstrained PPORank benchmark will be used for comparison, as there are no existing direct drug ranking comparisons.

1.3 Report Structure

The report is structured as follows: Chapter 2 introduces Reinforcement learning (RL), both unconstrained and constrained, building up to the main algorithm used in PPORank. Previous work using machine learning in healthcare is discussed with a focus on drug ranking methods. Chapter 3 contains the design of the drug severity scores and the formulation of the constraint function. Details on implementation as well as problems that occurred are explained in Chapter 4. Chapter 5 contains a description and explanation of the testing setup, followed by an analysis of the results. Chapter 6 is an evaluation of the project and the results, and the report ends with conclusions and future work.

2 Background

Contents

2.1 Reinforcement Learning Background

Reinforcement learning consists of interactions between an agent and the environment. It is a continuous learning process that takes place over many episodes (simulations), where the agent learns to maximise a reward signal.

2.1.1 Markov Decision Process

Reinforcement learning is modelled as a Markov Decision Process (MDP) [2]. This is based on the idea of states, actions, transitions, and rewards.

There are a set of states, $s \in S$, that describe the environment, and in an MDP it is assumed that being in a particular state encapsulates any prior information from previous states needed to inform your decisions. The following equation expresses the probability of moving to a new state at time $t+1$, given the current state at time t:

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, ..., s_t] \tag{2.1}$$

Actions, $a \in A$, determine how one moves between states. There are a subset of actions available in each state, and choosing one of these actions will result in the transition from one state to another. Transition probabilities map a state to a new state based on the action taken.

When moving from one state to another, a reward is incurred. This reward can be positive, negative, or neutral, depending on how the transition taken helps towards the overall goal. The expected immediate reward (collected at time $t+1$) after leaving the state s is $\mathbb{E}_\pi[R_{t+1}|S_t = s]$. When the model is learning, it is beneficial to consider all the rewards it will receive along the way which is why the return is used:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \tag{2.2}$$

The discount factor is denoted $\gamma \in [0, 1]$ and is used to balance the prioritisation of short-term and long-term rewards; setting $\gamma \to 0$ prioritises short-term and $\gamma \to 1$ prioritises long-term. This can be likened to the concept of the time value of money in finance, where future cash flows are considered less valuable than receiving that money now.

2.1.2 Useful Functions

In reinforcement learning, decision making is guided by several key functions that enable an agent to learn optimal actions in various states. The policy is a probability distribution that determines the best action the agent should take given the current state: $\pi : S \to A$. The value function is the expected total return, G_t, given the current state and that one follows the policy π:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{2.3}$$

The state-action value function, otherwise known as the Q function, is the expected return

after taking a particular action in a state, and then following the policy from that point forth:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{2.4}$$

The advantage function helps to quantify how much better taking an action, a, is than following the policy and is useful for exploring possible routes for the agent to take:

$$a(s,a) = q_\pi(s,a) - v_\pi(s) \tag{2.5}$$

2.1.3 MDP Solution Methods

There are a variety of ways of having the agent learn and they revolve around maximising the value function, the state-action function, or both. In the 1950s, Bellman proposed the idea of optimising an MDP [3], leading to the famous Bellman's equation.

Model-based learning methods use knowledge of the environment to estimate outcomes, whereas model-free methods do not. Model-free methods can be further subdivided into two categories; value-based and policy-based learning. Value-based methods are prone to instability [4], so policy-based methods are often preferred. There is also the choice of off-policy learning, which uses separate policies for training and selecting actions, and on-policy learning, where the same policy is used for training and evaluation; the target policy is the behaviour policy.

In addition to value-based and policy-based options, policy gradient (PG) and actor-critic (AC) methods have emerged. PG methods optimise the gradient of the performance of a parameterised policy and AC methods learn the value function and policy [5]. The goal of PG is to learn a parameterised policy and maximise a performance objective:

$$J(\theta) = \mathbb{E}[v_{\pi_\theta}(s)] \tag{2.6}$$

The policy can be any learnable function, for example, neural networks. The derivation of the gradient is laid out in [5], and we get the following equation, where τ is the trajectory taken:

$$\nabla_\theta \mathbb{E}[G(\tau)] = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \log\pi(A_t|S_t;\pi_\theta) G(\tau)\right] \tag{2.7}$$

The REINFORCE algorithm [6] is an early and fundamental PG algorithm, however, it suffers from high variance and potential gradient explosion, but the variance issue can be remedied with the addition of a baseline function [7].

In AC methods, the 'actor' learns the policy, the 'critic' learns the values of each state, and this is used to evaluate the current policy. Popular AC methods include Deep deterministic policy gradient (DDPG) [8], twin-delayed DDPG (TD3) [9] and Proximal Policy Optimisation (PPO) [10]. There are various reasons why the researchers behind PPORank chose to use PPO, for example, DDPG was shown to work well in simulated physics tasks but has been criticised as being unstable due to high sensitivity to hyperparameter tuning and tendencies to converge to poor solutions or even diverge [11]. Having this type of behaviour in a medical setting is unacceptable and is understandably not often used in this setting. TD3 attempted to remedy this, but it is significantly more computationally expensive due to its use of six networks.

PPO was developed by OpenAI and remains state-of-the-art, and the company is still using it. Its performance was shown to be superior to other AC methods and the ease of implementation, sample efficiency, and ease of tuning are all key advantages that PPO holds.

2.1.4 Proximal Policy Optimisation

PG methods tend to use the following loss, where $\hat{A}_t$ is an advantage (Equation 2.5) estimator:

$$Loss(\theta) = \mathbb{E}[\log\pi(a_t|s_t)\hat{A}_t] \tag{2.8}$$

It is possible to use other functions in place of the advantage, but this choice reduces variance. PPO takes inspiration from Trust Region Policy Optimisation (TRPO) [12], which updates the policy according to the ratio between the old and new policy to control step sizes:

$$\mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] \tag{2.9}$$

This is used alongside a KL divergence penalty in TRPO but PPO utilises a clipped objective function, instead using a value, ϵ to control the step size:

$$L = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1+\epsilon, 1-\epsilon\right)\hat{A}_t\right) \tag{2.10}$$

The second term clips the probability ratio, meaning that there cannot be a contribution larger than $(1 \pm \epsilon)\hat{A}_t$ to the objective. Taking the minimum of the clipped and unclipped values gives a pessimistic lower bound on the performance of the policy. The paper also includes a KL-penalty version, but it performed worse in testing than the clipped objective and was included as a baseline. To make the advantage estimation, a method called Generalised advantage estimation (GAE) [13] is used.

2.2 PPORank

In the context of PPORank, the state describes the current ranking positions and the drugs left to rank, an action involves selecting a drug to rank and the transition to a new state as a result of choosing that drug removes it from the candidate set. So, a "time step" that is typically associated with RL becomes the selection of a drug.

2.2.1 Objective Functions

The clipped policy objective used by PPORank [1] is the same as PPO, also using GAE:

$$L_t^{\text{clip}}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t^{\pi_{\theta_{\text{old}}}}, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1+\epsilon, 1-\epsilon \right) \hat{A}_t^{\pi_{\theta_{\text{old}}}} \right) \right] \tag{2.11}$$

In practice, the expectation of this function is calculated by taking the empirical average over a number of samples. Since AC also requires the estimation of the value function, the authors constructed this in a similar manner using a supervised loss function:

$$L^{VF}(\theta) = \hat{\mathbb{E}}_t \left[(V^{\pi_\theta}(s_t) - V_t^{targ})^2, \hat{\mathbb{E}} \left(\text{clip}(V_t^{\pi_\theta}, V_t^{\pi_{\theta_{old}}} - \epsilon, V_t^{\pi_{\theta_{old}}} + \epsilon) - V_t^{targ} \right)^2 \right] \tag{2.12}$$

V_t^{targ} is the target value and the easiest implementation of this is $r(s_t, a_t) + \gamma V^{\pi_{\theta_{old}}}(s_{t+1})$, which is the immediate reward due to moving from state s_t to state s_{t+1}, added to the discounted return from then on. $\hat{\mathbb{E}}_t$ is again the empirical average over a sample batch. PPO adds an entropy term to encourage exploration, so PPORank also does in the construction of the surrogate loss:

$$L_t(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S(\pi_\theta, s_t)) \right] \tag{2.13}$$

where $S(\pi_\theta, s_t)$ is the entropy loss, and $\hat{I}_t \in 0, 1$ denotes if the t'th drug is selected:

$$S(\pi_\theta, s_t) = -(1 - \hat{I}_t) \log(1 - \pi_\theta(a_t|s_t)) \tag{2.14}$$

2.2.2 Rewards and Performance Metric

The reward for choosing a drug in PPORank is based off normalized discounted cumulative gain (NDCG) as this is the evaluation metric used to assess the performance of the model and is directly optimised. This metric is widely used for ranking models, and since it uses a discounting process, its use is logical for this DRL use case. Discounted cumulative gain (DCG) is the sum of gains, discounted by their rank. The NDCG is calculated by dividing DCG by the ideal discounted cumulative gain (IDCG), which is the maximum possible DCG with the same scores but best ranking order [14]. It is common to only be interested in looking at the top k items, in which case NDCG becomes NDCG@k. The equations are below, where rel_i is the relevance of each item, k_{rel} signifies the items are ordered by relevance and $\log_2(i+1)$ is the discount factor.

$$\begin{aligned} DCG@k &= \sum_{i=1}^{k} \frac{\text{rel}_i}{\log_2(i+1)} \\ IDCG@k &= \sum_{i=1}^{k_{\text{rel}}} \frac{\text{rel}_i}{\log_2(i+1)} \\ NDCG@k &= \frac{DCG@k}{IDCG@k} \end{aligned} \tag{2.15}$$

In the case of PPORank, the relevance score is $G(f, \pi^{-1}(t)) = 2^{f^{\pi^{-1}(t)}} - 1$, where $G()$ is the function that gives the response score of a drug, and $f^{\pi^{-1}(t)}$ is the response score itself. Therefore, the NDCG metric for all positions in PPORank can be written as follows:

$$NDCG = \frac{1}{Z_M} \sum_{i=1}^{M} \frac{2^{f^{\pi^{-1}(t)}} - 1}{\log_2(i+1)} \tag{2.16}$$

where M is the number of all candidate drugs and $\frac{1}{Z_M}$ is a normalising constant based on the

ground truth response scores.

2.2.3 Architecture

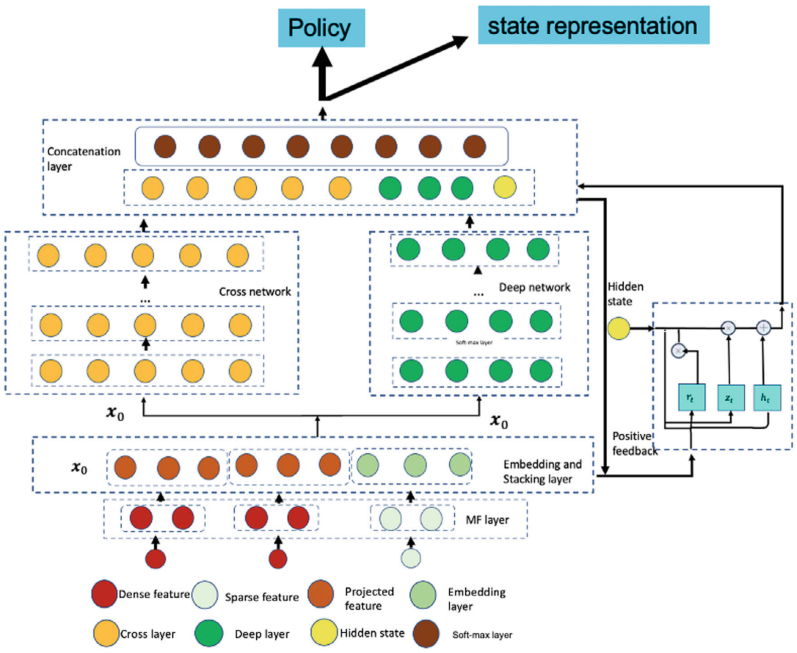


Figure 2.1: PPORank policy architecture. This figure is taken from [1].

Figure 2.1 shows the architecture of the actor network. It begins with a matrix factorisation layer, which is used to project interactions between drugs and cell lines onto the same dimension since this data can come from various sources. The following equations describe the projections:

$$\begin{aligned} Y &= B + UV^T \\ U &= XW_c \\ V &= DW_d \end{aligned} \tag{2.17}$$

where $Y \in \mathbb{R}^{N \times M}$ is the drug response matrix, $B \in \mathbb{R}^{N \times M}$ is the bias matrix, $X \in \mathbb{R}^{N \times P}$ is the cell line features, $D \in \mathbb{R}^{M \times M}$ is the drug features, $W_c \in \mathbb{R}^{P \times f}$ and $W_d \in \mathbb{R}^{M \times f}$ are weight matrices, $U \in \mathbb{R}^{N \times f}$ is the cell line preference matrix, and $V \in \mathbb{R}^{M \times f}$ is the drug preference matrix. These outputs are fed into the embedding and stacking layer, which stacks the dense features of cell lines from the previous layer, with embedding vectors for each drug, and finally the cosine similarity score between cell lines and drug latent vectors is appended:

$$x_0 = [u^T, v_1^T, ..., v_M^T, cos] \tag{2.18}$$

where u is the projected dense features for one cell line and $v_{1...M}$ comes from the drug preference matrix.

This is inspired by the deep ς cross network of [15], as well as the neural network setup. This was originally designed to predict the click-through rates of advertisements and involves the use of a deep, fully connected feed-forward neural network and a cross network to capture high-order relations between the latent vectors u and v. The cross layers are defined as follows:

$$x_{l+1} = f(x_l, w_l, b_l) + x_l = x_0 x_l^T w_l + b_l + x_l \tag{2.19}$$

where x_l and x_{l+1} are outputs from the lth and $l+1$th layers respectively, w_l is the weight vector for layer l and b_l is the bias vector for layer l and x_0 is the original input into the network from the embedding and stacking layer. The deep network is defined as such:

$$h_{l+1} = ReLU(W_l h_l + b_l) \tag{2.20}$$

where h_l and h_{l+1} are outputs from the lth and $l+1$th layers respectively, W_l is the weight matrix and b_l is the bias vector.

The gated recurrent unit (GRU) [16] is not part of the deep ς cross network and was added for PPORank to use evaluation signals to learn dynamic hidden patient preferences or conditions since these may change. It memorises previous evaluation signals and remembers values over multiple iterations so given a positive evaluation signal, it can update the GRU layer; it decides what previous information is needed and acts as the neural network's memory. The following equations define its function:

$$\begin{aligned} \mathbf{z}_t &= \sigma(W_z \mathbf{x}_t + U_z \mathbf{H}_{t-1} + b_z) \\ \mathbf{r}e_t &= \sigma(W_r \mathbf{x}_t + U_r \mathbf{H}_{t-1} + b_r) \\ \mathbf{H}_t = \mathrm{GRU}(\mathbf{H}_{t-1}, \mathbf{x}_t) &= \mathbf{z}_t \circ \mathbf{H}_{t-1} + (1 - \mathbf{z}_t) \circ \tanh(W_h \mathbf{H}_{t-1} + U_h(\mathbf{r}e_t \circ \mathbf{H}_{t-1}) + b_h) \end{aligned} \tag{2.21}$$

where $\mathbf{z}_t$ is the update vector, $\mathbf{r}e_t$ is the reset gate vector, $\mathbf{x}_t$ is the concatenated output from the deep and cross networks, $\mathbf{H}_t$ is the hidden state, W and b are weights and biases, respectively, $\circ$ is the Hadamard product and σ is the ReLU function. The update gate, $\mathbf{z}_t$, determines how much past information (from previous time steps) needs to be passed to future time steps; it controls long-term memory. The reset gate, $\mathbf{r}e_t$, uses the same formula as $\mathbf{z}_t$ but is used differently; it is used to decide how much of the past information to forget when calculating the candidate hidden state. The candidate hidden state is computed in the *tanh* function, and in the calculation of $\mathbf{H}_t$ it is clear to see how $\mathbf{z}_t$ controls how much of the candidate hidden state to incorporate into the current hidden state.

The concatenation layer concatenates the outputs from the cross network, deep network, and GRU using the softmax function to form a probability distribution, i.e. the policy estimate.

As mentioned earlier, the critic network is used to approximate the value function and evaluates the actor network's policy. It also consists of a deep and cross-network, GRU and a feed-forward layer which outputs a single value. The input for the critic network comes from the state representation provided by the actor network, and it is learnt in conjunction with the actor network; the parameters between the policy and value networks are shared since this helped stabilise training in PPORank due to the reduction in parameters.

2.2.4 Medical Data

There are two main datasets used, the Genomics of Drug Sensitivity in Cancer (GDSC) [17] and the Cancer Cell Line Encyclopaedia (CCLE). The GDSC contains data for 1001 cell lines and 265 drugs [18]. [19] has shown that gene expression is the most informative data to use. PPORank adopts the method of [20] to extract these cell line features, resulting in 983 being usable and obtained. CCLE contains 1036 cell lines and 24 drugs, however, only 491 cell lines have both gene expression and drug sensitivity data. Both datasets report sensitivity using $log(IC_{50})$ values, which is the drug concentration needed to inhibit cell growth by 50%.

2.3 Constrained RL

When considering a constrained RL problem, it is necessary to ensure that the policy satisfies certain constraints, and we can denote this set of policies as Π_C. The constrained Markov decision process (CMDP) framework described in [21] has provided a way to illustrate the feasible set:

$$\Pi_C = \{\pi : J_c < d\} \tag{2.22}$$

where J_c is a constraint function based on cost. For this project, the cost is associated with drug severity scores. The optimal policy must now maximise a reward whilst remaining within the feasible set. The new objective can now be viewed as:

$$\max J_r(\pi_\theta) \quad \text{subject to} \quad J_c(\pi_\theta) < d \tag{2.23}$$

where J_r is the objective reward which in the case of PPORank, as mentioned earlier, is based on the NDCG of the rankings. The goal of the CMDP for PPORank is to maximise the rewards (NDCG) while respecting the specified severity constraint.

The paper that introduced OpenAI's Safety Gym environment [22] also put forward the idea of PPO-Lagrangian, although it did not provide any detail beyond the idea. Instead of thinking about maximising a reward, an alternative way of framing the problem is to minimise the PPO loss when it has a penalty applied to it by a constraint function. In this case, the formulation applicable to PPORank is as follows:

$$\min L(\theta, \lambda) = f(\theta) + \lambda(g(\theta) - d) \tag{2.24}$$

where θ is the neural network (NN) parameters, λ is the Lagrange multiplier, $f(\theta)$ is the PPO loss and $g(\theta)$ is the constraint function. This constraint function must penalise ranking drugs with a high severity score at the top. In Equation 2.24, $g(\theta)$ effectively calculates $J_c(\pi_\theta)$. There are other possible formulations for this new loss, described in Section 3.2, which are more intuitive to understand when considering the problem in terms of losses and penalties.

2.4 Machine Learning in Healthcare

As pointed out by [23], applications for RL in healthcare can be split into several categories; dynamic treatment regimes (DTR), automated medical diagnosis, and other general applications such as drug discovery or robot control. This project focusses on DTR which can be seen as sequential decision making and is therefore an ideal application for RL. Through the implementation

of a reward process, the efficacy and side effects of the drug can be optimised. DTR can be further split into critical care, such as treating sepsis and treatment for chronic diseases such as diabetes and cancer.

At this point, it would be prudent to expand on the rationale behind using deep reinforcement learning to build a drug response prediction model. Since it is a continuous learning process, it allows the use of currently available medical data to train the model and then input any future data to update it without having to completely retrain. It is also suited for situations where the immediate effect of your actions may not be apparent, as is the case with medical treatments. The flexibility that this methodology affords, especially the case with PPORank, allows the incorporation of many data sources so if new correlations are discovered, they can be included without needing to re-engineer the model. The scalability of RL is a large part of this too; it can generalise across different types of cancer, treatment methods, and demographics. Using DRL, in particular, is important, as drug interactions tend to be non-linear [24], so the ability of neural networks to capture this behaviour is crucial and also explains the choice of the deep ς cross network in PPORank. The use of RL for this task can help to bridge the gap between general clinical guidelines and individualised patient care.

2.4.1 Methods for Treatment Recommendation

Numerous approaches have been taken for this problem, however, methods involving deep learning tend to perform best as will be expanded upon later. For example, [25] combines several deep learning (DL) architectures to this effect and [26] uses a deep factorisation machine to rank drugs using relative scoring rather than absolute scores. Both of these models outperform models using non-DL methods.

Of the NN architecture options, graph neural networks (GNN)s and convolutional neural networks (CNN)s are the most popular choices [27]. The reasons for this are primarily due to drug representations; in fact, there are several papers just on this topic. The most commonly used representation is simplified molecular-input line-entry system (SMILES), where molecules are described by a string of characters that represent their structure and makeup. This is the format used by GDSC and CCLE. However, graph structures would perhaps better represent these structures, since their structure would be able to be defined by a graph of nodes and edges. Converting drug SMILES to a GNN-friendly format is another area of research with many proposals, so replacing the deep ς cross network with a GNN could be a possible direction to take

for this project. CNNs are frequently used due to their use of fewer learning parameters, however, two papers report that they actually underperform dense layer networks [28], [29], however, the performance is comparable when combined with attention mechanisms.

CDRScan [30] uses an ensemble of five CNN-based models with varied architectures to perform drug ranking, however, PPORank outperformed CDRScan in tests. Dr. VAE [31] employs an auto-encoder technique using pre and post-treatment cell lines. The nature of autoencoders makes them a suitable choice for this task since they become good at extracting information by compressing the input and trying to reconstruct it from the lower dimension. Its performance was better than various benchmarks, but there was no comparison to any of the models that PPORank was compared to. Cancer Drug Response prediction using a Recommender System (CaDRReS) [20] is a matrix factorisation-based approach that uses the GDSC and CCLE datasets. Given the numerous similarities with PPORank, including the methodologies for evaluation and testing, it is logical that CaDRReS was chosen for comparison in the PPORank paper. Kernelized Rank Learning (KRL) is another approach used in comparison; it aims to rank the top drugs k, such as PPORank, again using similar evaluation and testing frameworks.

A paper released after PPORank is a Deep Attention Q-Network [32] that uses a transformer architecture with DRL. This utilises a a different DRL class; optimising for the state-value function comes with different challenges to AC, but they claim that the use of an encoder-decoder architecture aids in taking into account the patient history. The inclusion of the GRU in PPORank helps that model with similar issues; 'hidden states' are a problem in both. This particular paper is focused on ICU patients and is designed to work using patient observation data, so a comparison is not possible, but it is worth mentioning as another approach which could potentially be adapted for cancer drug recommendation.

3

Design of the Constraint and Losses

Contents

3.1 Drug Severity Scores

There is no single simple metric to classify the severity of side effects of a drug. After examining several databases containing patient reactions to drugs, the Adverse Drug Reaction Classification System (ADReCS) [33] database was chosen. This is a comprehensive adverse drug reaction (ADR) database which pulls drug and drug reaction data from multiple sources. There were several reasons behind the decision to use this database; it has the highest overlap of drugs with those used in PPORank, i.e. from GDSC and CCLE, due to the diverse range of sources, it standardises ADR terms and it provides quantitative aspects to its drug reactions, which similar databases tend not to do. Many of these drug reaction databases are for doctors and medical researchers, making much of the information uninterpretable for someone without a medical background, but the aforementioned standardisation for ADReCS and the fact that this quantifies several aspects

also made this database significantly easier to understand. Each drug reaction (side effect) is classified into one of five severity grades. A severity score for each drug can be formalised as follows:

Let w_i represent the weight for each severity grade i and n_i be the number of side effects in the severity grade i. The weight w_i is used to reflect the relative severity and importance of each grade in the overall calculation and can be adjusted accordingly. For this project, the weights assigned to the five severity grades are as follows: 1 for mild, 2 for moderate, 3 for severe, 5 for life-threatening, and 10 for death. The total severity score S for a drug is given by:

$$S = \sum_{i=1}^{5} w_i \cdot n_i \tag{3.1}$$

where i ranges from 1 to 5, corresponding to the severity grades. To ensure comparability of scores between drugs, it should be normalised by dividing by the total number of side effects N:

$$S_{\text{normalized}} = \frac{S}{N} = \frac{\sum_{i=1}^{5} w_i \cdot n_i}{\sum_{i=1}^{5} n_i} \tag{3.2}$$

where $N = \sum_{i=1}^{5} n_i$ is the total number of side effects for the drug.

3.2 Formulating the Constraint

As mentioned in Section 2.3, the PPO surrogate loss is replaced by the Lagrangian formulation $L(\theta, \lambda) = f(\theta) + \lambda(g(\theta) - d)$. Therefore, the construction of the constraint function $g(\theta)$ is necessary.

3.2.1 Creating the Constraint Function

Two separate constraint functions were initially created, one linear and the other logarithmic-based, such that the drugs ranked highest would have more emphasis placed on them since the drugs ranked lower down are unlikely to be considered; the idea is to penalise placing drugs with a high severity score towards the top of the ranking. Denoting s_i = severity score of drug i, $rank_i$ = rank of drug i, n = number of drugs, and penalty_i = penalty for drug i, the linear formulation is as follows:

$$g(\theta) = \frac{1}{n}\sum_{i=1}^{n} \text{penalty}_i = \frac{1}{n}\sum_{i=1}^{n} \frac{s_i}{\text{rank}_i} \tag{3.3}$$

By placing the rank of each drug in the denominator, the drugs ranked lower have less emphasis placed on them in a linear manner. The logarithmic formulation is as follows:

$$g(\theta) = \sum_{i=1}^{n} \text{penalty}_i = \sum_{i=1}^{n} s_i(-\ln(rank_i) + 4) \tag{3.4}$$

The coefficients chosen for this formulation ensure that it always yields a positive penalty for each drug. This is important to provide a clear direction of optimisation for the constraint and to ensure consistency in penalising high severity drugs, and this consistency is necessary to guarantee that the selection of drugs with a high severity score will be reliably punished. Additionally, the backpropagated gradients should push the model towards more feasible solutions due to the more consistent feedback, and this approach enhances the stability of the training process through the consistency and makes the model's behaviour more interpretable in the results. Note that the rank of the drugs is directly used, which is key in the method for updating the model, as will be explained in Section 3.2.3.

The logarithmic term decreases in a manner that applies a considerably greater weight on the severity of the highest ranked drugs compared to the lowest ranked drugs, resulting in significantly higher penalties for drugs at the top and progressively smaller penalties for those below. The logarithmic version was ultimately chosen since it better prioritises the highest-ranked drugs, which would be the ones most likely to be considered in reality. This type of prioritisation aligns better with this project's focus on taking real-world considerations into account while also trying to maximise performance. A graph of this weighting function is visualised below for reference. The calculation for $g(\theta)$ in Equations 3.3 and 3.4 is for a single cell line; in actual use, the mean of this value is taken for all cell lines. Not all rankings are penalised when calculating $g(\theta)$ for the loss, since a ranking truly incurs a penalty only if the value of $g(\theta)$ exceeds d, with the exception of the standard quadratic penalty loss, as will be explained soon.

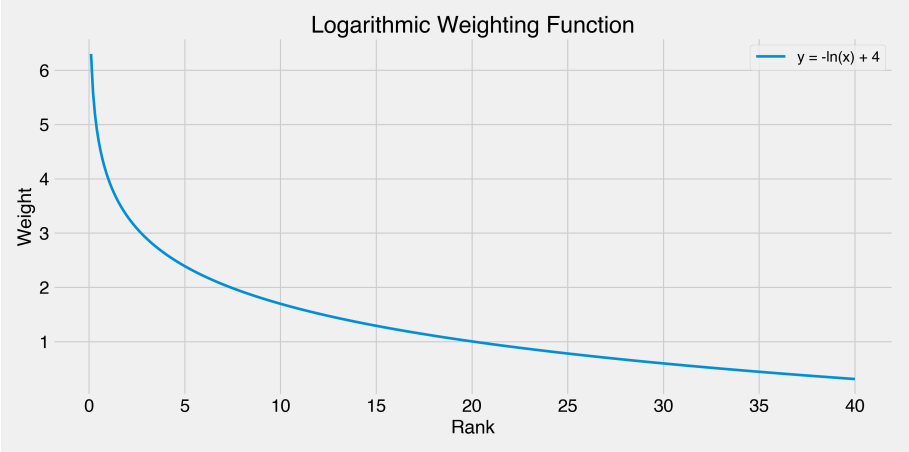


Figure 3.1: Logarithmic weighting function

3.2.2 Total loss formulations

Now that the constraint function, $g(\theta)$, has been established, it must now be included in the total loss function. Whilst the Lagrangian loss formulation is an obvious choice, there are several possible variations of the total loss which are presented below.

Lagrangian

As previously stated, the natural Lagrange formulation for this is as follows:

$$L(\theta, \lambda) = f(\theta) + \lambda(g(\theta) - d) \tag{3.5}$$

where $f(\theta)$ is the PPO loss, $g(\theta)$ represents the constraint function, and $g(\theta) - d$ indicates the degree to which the constraint is violated. During the backward pass, when gradients are backpropagated, PyTorch aims to minimise the total loss. Since this includes the constraint function, the parameters should be adjusted to decrease the PPO loss, thereby increasing performance, while simultaneously reducing the constraint violation. Thus, the policy is pushed to recommend an ordering that maximises NDCG while meeting the constraint. The parameter λ plays a crucial role in balancing the trade-off between the primary objective (maximising NDCG) and satisfying the constraint. Adjusting λ controls the importance given to the constraint in the total loss function.

Although PyTorch uses gradient descent to optimise NNs, performing gradient descent on λ would achieve the opposite effect of what is intended. To decrease the total loss, λ would also be

reduced since $g(\theta)$ is constructed to always be positive. Lowering λ would diminish the weight of the constraint in the total loss function, causing parameter updates to focus more on increasing model performance. Instead of updating the NN parameters to minimise the penalties constructed in 3.4, the model would effectively disregard the constraint by steadily decreasing λ and give it no weight in the total loss. This would technically minimise the penalties, but it would completely undermine their inclusion in the optimisation process. Therefore, the update for λ requires gradient ascent to enforce the constraint, as it would increase when constraint is being violated, resulting in a policy that aims to maximise NDCG while respecting the constraint.

Quadratic penalty

Another possible formulation includes using a standard quadratic penalty:

$$L(\theta, \lambda) = f(\theta) + \frac{1}{\rho}||g(\theta) - d||^2 \tag{3.6}$$

There are several potential benefits to using this total loss instead; since the penalty grows quadratically as the penalty increases, it provides a stronger incentive for the model to reduce large constraint violations. It can also be used in an adaptive manner by increasing or decreasing ρ depending on whether the constraint penalty is increasing or decreasing or using thresholds, although ρ is typically fixed in a quadratic loss formulation. Since the Lagrangian formulation uses an adaptive coefficient, making ρ also adaptive would essentially convert this approach into a mildly modified version of the Lagrangian formulation. For this reason, ρ is set to a fixed value to maintain a clear separation between the two approaches. It is also logical to transform the formulation using a threshold to avoid penalising negative values for $g(\theta)$:

$$L(\theta, \lambda) = f(\theta) + \frac{1}{\rho}||\max\left((g(\theta) - d), 0\right)||^2 \tag{3.7}$$

A negative value for $g(\theta)$ means that the constraint is being over-met; the value for $g(\theta)$ is less than d, yet a penalty is still applied to this in the total loss in Equation 3.6. This thresholded version allows the model to improve the NDCG performance while the constraint is met without being unnecessarily affected by any negative penalties; penalising rankings that exceed the required constraint satisfaction is counterproductive.

Augmented Lagrangian

The final loss considered is an augmented Lagrangian [34] formulation:

$$L(\theta, \lambda) = f(\theta) + \lambda(g(\theta) - d) + \frac{1}{\rho}||g(\theta) - d||^2 \tag{3.8}$$

It is not possible to solve this using a method of multipliers algorithm, such as alternating direction method of multipliers (ADMM) [35], as originally intended when this was first introduced since this is a highly nonlinear and nonconvex problem. However, this loss function still has merits; in this ranking context, it combines the benefits of 3.5 and 3.6, providing a more robust approach to imposing the constraint. By combining the linear penalty term, $\lambda g(\theta)$, from the Lagrangian formulation with the quadratic penalty term, $\frac{1}{\rho}, ||g(\theta)||^2$, the augmented Lagrangian formulation offers immediate penalisation for constraint violations and a quadratically increasing penalty for larger constraint violations. This ensures that smaller violations are sufficiently addressed, while significant violations are strongly punished. Additionally, it is still possible to have adaptive coefficients, and the formulation remains differentiable, which is necessary for this situation, as previously explained. In many methods for solving augmented Lagrangians, the coefficient ρ is often updated based on progress toward meeting the constraint. These methods typically use dual ascent, but in the case of this project, backpropagation of gradients serves as the solver. Thus, following one of the many variations of these methods may not necessarily work.

Therefore, two approaches are explored: using a fixed and adaptive ρ. A fixed ρ means that this formulation essentially becomes a combination of the Lagrangian and quadratic penalty formulations, leveraging the stability of the fixed quadratic penalty term while maintaining the adaptability of λ. This is the simpler approach and could be effective, as long as a suitable ρ is chosen. In the adaptive method, ρ is updated according to the value of $g(\theta) - d$; if it is below a threshold, ρ is increased to reduce the contribution of the quadratic term so that the model can focus more on the primary objective, $f(\theta)$. Conversely, if $g(\theta) - d$ is above a threshold, ρ is decreased to more strictly enforce the constraint. The latter approach affords greater flexibility in handling the constraint and helps to balance the trade-off between performance and constraint satisfaction. The augmented Lagrangian formulation provides a comprehensive framework that integrates the benefits of both the Lagrangian and quadratic penalty methods. However, it requires greater tuning, and it can be more difficult to achieve optimal performance using this method.

Log barrier

The log barrier method [36] appears to be a reasonable approach to enforce the constraint, however, it turns out to be infeasible.

The log-barrier function is defined as:

$$\phi(x) = -\ln(d - x) \tag{3.9}$$

for $x < d$. When $x \geq d$, the function becomes undefined, which poses a significant issue during the initial training stages. The random initialisation of the NNs used to approximate the policy and value functions often results in the constraint not being satisfied at the start. When this is the case, the threshold d is exceeded and the log-barrier function is undefined in this region. Although it is possible to define the function so that it does not become undefined until after d, it would then resemble a much steeper version of the thresholded quadratic loss, with the 'max' term. Clipping the values is not an option either, as this would prevent gradient flow upon a constraint violation, meaning there would be no penalty for this since the backpropagation would carry no information about the violation. Finally, as shown in the results, applying an excessively large penalty for a constraint violation can significantly hinder training and model performance. Given the nature of the log barrier function, this would likely be the case. Therefore, it is clear that the log-barrier method is not as suitable for this project as the previous options.

3.2.3 Method for Updating the Model

At this point, the total loss has been introduced, which includes the constraint function $g(\theta)$ to find the penalty to apply to the PPO loss. The final piece of the framework is the method of actually calculating $g(\theta)$ by finding the rankings while training the model. This section will begin by explaining why conventional constrained DRL methods for updating the model are not able to be used for this drug ranking task, and the new rank-based approach is presented following this.

Why other methods are not suitable

Previous research covering constrained RL, specifically constrained PPO, usually takes the approach of calculating cost/constraint advantages [37], [38], [39]. This approach is due to the nature of the task that they aim to accomplish, which is typically in the OpenAI Safety Gym environment,

where the agent aims to complete tasks such as crossing an arena without hitting any obstacles, with penalties for collisions. Therefore, the Safety Gym environment is very different from the environment in which the PPORank agent learns; penalties are immediately received upon a collision and are more straightforward and based on discrete events. In these scenarios, it is logical to calculate cost advantages since the ordering of actions is not the primary focus; the agent just needs to reach its goal without any collisions. However, in the case of PPORank, the final ranking of the drugs and therefore the order of the actions (drug selection) are paramount, and the penalties should reflect this.

Calculating cost advantages would introduce several complexities. Firstly, this would require the use of an additional cost-value network to estimate the cost of each action. The authors of PPORank have already encountered issues with policy divergence during training when using a separate policy and value network design. Adding another network may further destabilise the learning process, which is already susceptible to failure. Moreover, using advantages makes it more difficult to quantify a value for the threshold d in Equation 2.23 because the advantage function itself does not directly provide a penalty measure; it only indicates a relative cost of actions.

The reason for this is that the aim of the methods that use cost advantages is to eliminate collisions or violations, and as a result they lack fine control over a threshold, such as d. In the Safety Gym environment, where the goal is to prevent collisions, the threshold $d = 0$. This is reflected in the Lagrangian formulation used in the PPO-Lagrangian paper, which uses $\lambda g(\theta)$ in the loss instead of $\lambda(g(\theta) - d)$ as specified in Equation 3.5. This lack of direct penalty control means that, while the policy is encouraged to reduce violations, there is no precise mechanism to ensure the cost remains below a specific threshold. In the ranking context of this project, having clear control over this is crucial to ensure that the constraint is applied correctly in the rankings. The value of d directly influences the degree of importance placed on the severity of a drug compared to its efficacy; a lower d means that drug severity is weighted more heavily than efficacy, whereas a higher value allows for greater emphasis on efficacy over severity. This is why $g(\theta)$ directly uses the rank of a drug in its calculation.

Rank-based approach

Given the unique and specific requirements of ranking using PPO, a new method is developed to update the policy and value networks using a ranking estimation which is only possible due to the ranking objective. By retrospectively examining the ranking choices made and estimating the

policy's preferred ranking of the drugs based on minibatches, one can directly inform updates of the policy and value networks according to the constraint penalty and efficacies of the drugs.

In updates to policy and value networks, the actions taken (drugs selected) are evaluated and the PPO loss is calculated according to Equation 2.13. In practice, there are over 100 actions to consider in each of these updates. This figure is derived from the fact that there are 38 time steps (drugs to pick) per cell line, and there are multiple cell lines in a batch. For example, in a batch of 16 cell lines, there are 608 total actions taken, and each batch is further divided into minibatches. From this, a ranking of what the model considers to be the most effective drugs can be estimated by examining the rankings in the minibatches. There is some stochasticity introduced in this method due to the minibatch splitting, so the actual preferred ordering over all training data may differ slightly, but this still acts as a good estimate. This can be positive, as stochasticity can inhibit overfitting by preventing the model learning specific patterns, aiding generalisation. In the context of RL, this could also encourage further exploration of different combinations of drug rankings. However, slight differences between the estimated and true ranking introduce some inconsistency and potential instability, although this can be managed with sensible hyperparameter choices for parameters such as λ and ρ.

With this estimated ranking, the route to calculate $g(\theta)$ is clear; there is now an ordering of the 38 drugs for the cell lines, and Equation 3.4 is applied to each cell line. For a final single value to use in the loss as $g(\theta)$, the average is calculated. The logarithmic weighting in Equation 3.4, as shown in Figure 3.1, acts as an alternative to the cost advantages; by weighting penalties based on rank, this indirectly accounts for the future impact of early decisions. Selecting a high severity drug early (higher rank) imposes a larger penalty, thus encouraging the policy to consider future consequences of early actions. Therefore, in this method, one of the primary benefits of using cost advantages is retained.

Another option is to calculate the penalty for each drug as they are sampled in episodes. In each step, actions (drugs) are sampled sequentially for each cell line until there are no drugs left. Since not every cell line has ground truth data for all drugs, there are sometimes fewer than 38 steps as drugs with no score for a cell line are excluded from the candidate list. Therefore, this setup also introduces some inaccuracy in the estimate of the rankings and penalty values, and would also require tracking gradients that were not previously tracked, increasing memory usage and training times as the computational graph would become significantly larger. Furthermore, it is more difficult to provide a lower and upper bound for the total penalty this way due to the change in the number of candidate drugs per cell line, making the value of d in Equation 2.23 more

difficult to set and somewhat arbitrary. In contrast, with the method of obtaining the ordering in the policy update, it is easier to determine the minimum and maximum possible values for the penalty by arranging the drugs in ascending and descending order, respectively, and calculating Equation 3.4 as you have a proper order for the 38 drugs.

For these reasons, the minibatch-based option was chosen; the gradients necessary for backpropagation are already being tracked, so computation and memory overheads are not affected. With this approach, all cell lines receive a score from the policy, so all drugs are always ranked, which will provide a better estimate.

4 Implementation

Contents

The authors of the PPORank paper provide their code in a GitHub repository, so there was no need to implement that model completely from scratch. However, some parts of the code were missing and had to be reimplemented to run the model. Also, it became apparent that training the model on a MacBook would not be possible, despite PyTorch's recent addition of the MPS framework to their library for GPU accelerated workflows. Numerous bugs were encountered with this new framework, leading to a significant amount of time wasted debugging code that was otherwise functioning correctly. As a result, it was necessary to switch to using Imperial's high performance computing (HPC) cluster, which cut training times through PyTorch's more mature stable CUDA implementation. Distributed learning across multiple GPUs was then implemented, however, the increased queue times for more than one GPU outweighed any benefit afforded by decreased training times, so a single GPU workflow was used.

4.1 Obtaining a Baseline

The number of drugs that are ranked is now different from the original PPORank paper since not all of those drugs are contained within the ADReCS database, so the drug list is filtered to

include only those common in both. As a result, the tuned hyperparameters and results from the PPORank paper are not able to be used for a baseline, necessitating the creation of a new one. In doing so, several issues were encountered, chief among them being the 'catastrophic forgetting' problem, a term first coined by [40]. This phenomenon, depicted in Figure 4.1, is where the model's performance spontaneously plummets as if it forgets all of the good choices it had learnt. This is a problem commonly encountered within RL, but there is no obvious solution to each case.

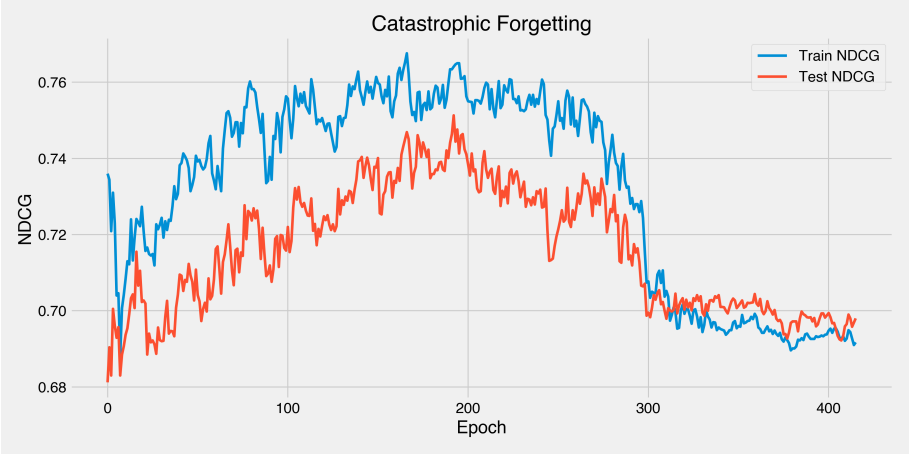


Figure 4.1: An example of catastrophic forgetting

To address this issue, three main fixes were tried: increasing the GAE parameter λ over time, manually checking the KL divergence between old and new policies, and reward scaling. The reason why increasing λ over time could help is that it would cause the advantage estimation to consider longer time horizons and the agent to consider longer-term rewards, theoretically encouraging the policy to better retain information in updates. The KL divergence check is intended to prevent large policy changes by controlling the extent to which the new policy deviates from the old one by finding the approximate KL divergence. TRPO employs a similar tactic in its loss formulation to stabilise training, hence the attempt to push the policy toward more gradual learning by halving the learning rate and PPO clip parameter if a KL divergence beyond an allowed threshold is detected. The final method was reward scaling; since NDCG is used to generate the rewards so that this metric is optimised directly, the rewards were large, resulting in PPO loss values in the millions. To reduce large, abrupt changes in the policy through smoother gradients, rewards are scaled down by a factor of 100.

In reality, only reward scaling had a significant positive effect in mitigating this issue; training is smoother with far fewer sudden drops. The KL divergence checks had little impact and would have taken a considerable level of trial and error to tune properly, so this idea was abandoned.

Increasing the GAE λ parameter did not appear to have much of an effect either, possibly because this value was already initially close to 1. Finally, simply decreasing the learning rate was also an effective partial solution to the 'forgetting' problem. Adjusting other hyperparameters such as the clip parameter ϵ in Equation 2.11 to limit the deviation of new policies from the previous ones, and the entropy coefficient c_2 in Equation 2.13 to encourage exploration was also considered. In practice, the adjustment of ϵ did not positively affect performance and raising c_2 worsened performance. The latter is likely due to the introduction of too much noise into the loss, causing the policy to explore new options too frequently.

4.2 Implementing the Constraint

Since a NN is at the heart of the DRL model, it is trained through backpropagation and therefore the constraint must be implemented in such a way that it is compatible with PyTorch's automatic differentiation engine, *Autograd*. This compatibility is crucial because it ensures that gradients can be correctly calculated and backpropagated through the NNs. In a typical Lagrangian or augmented Lagrangian problem, one differentiates with respect to the input variable(s) and λ, however, in this particular situation, it also seemed reasonable to use a differentiable penalty function to penalise constraint violations and adjust the coefficient based on how the value of this function changes over time. Therefore, both approaches were tried.

There are two possible places to update λ; every time the NN is updated in the PPO update (more than 700 times per epoch) and once per epoch after validating on the entire training dataset. The former did not appear to be a good option, as it could lead to instability in training due to the frequency of updates, and the constraint terms may fluctuate wildly. This noise can cause λ to react to transient violations that do not represent overall model performance, leading to erratic training dynamics and potentially causing the model to oscillate rather than converge.

On the other hand, updating λ once per epoch after validating the entire training addresses the noise issue because it allows for a more reliable inference of the constraint satisfaction by obtaining a ranking of drugs over all cell lines, which reduces the risk of forcing the model to react to unrepresentative constraint violation values. This also gives the model plenty of time to adapt to the new penalty structure. However, this may lead to periods where the constraint is not adequately enforced; since λ is updated less frequently, there may be extended periods of time when the constraint is under or overemphasised, and this lack of responsiveness wastes training

time and hinders progress due to suboptimal λ values. Moreover, the adjustments will need to be larger to have a noticeable effect, resulting in less granular control over the enforcement of the constraint.

To balance the responsiveness of frequent updates and the stability of infrequent updates, a middle-ground approach involves averaging constraint violations within each epoch in batches and periodically updating λ. The update function is called 48 times per epoch, and each time it is called there two loops, resulting in 16 NN updates. By keeping track of the average constraint violations across the internal loops and then updating λ, a more accurate picture is obtained of how well the constraint is being met. This method offers several advantages; it mitigates the noise issue associated with frequent updates, while allowing for more prompt adjustments to λ. This ensures that the model can adapt more smoothly to violations without overreacting to shorter-term fluctuations, and this allows for finer granularity in updates compared to the once-per-epoch method, since λ is adjusted more frequently but in a controlled manner. However, this still requires careful selection of the learning rate for λ so that updates are neither too sluggish nor responsive. Nonetheless, this approach strikes a balance between stability, responsiveness, and computational efficiency.

This did present several difficulties in implementation, as there are multiple areas where tracking variables over time can cause issues with the computational graph. Since the model parameters, θ, are updated every iteration of the loop, the average constraint violation must be tracked separately to avoid affecting the computational graph. Unlike accumulating gradients over batches and updating all parameters at the end, here the model parameters are updated every iteration, while λ is only updated at the end of every 16 iterations.

On the final iteration, the computational graph must be retained between calculating the gradients with respect to the total loss to update θ, and calculating the gradients with respect to $\lambda(g(\theta) - d)$ to update λ. The sequence of these updates is important; if the model parameters are updated first, the subsequent update to λ would use outdated parameter values, leading to errors. Tracking the violation presents an unusual challenge in managing dependencies within the computational graph; the accumulated violation must be detached from the graph during the first 15 iterations of the loop to ensure that accumulating these values does not interfere with the graph. On the 16th iteration, when it is time to update λ, the accumulated violation is reattached to the graph and divided by 16 to calculate the average violation. This approach ensures that the computational graph remains unaffected during the accumulation phase but allows the average violation to be correctly used in backpropagation when needed.

As mentioned at the beginning of this section, updating λ based on the constraint violation is an option; if it is above a threshold or increases over time, then λ must increase, and the inverse holds if the violation is below a threshold or decreases over time. Although, in principle, this could be adjusted enough to weight $g(\theta)$ sufficiently, the thresholds would be another hyperparameter to tune, and the value of λ's gradient allows for more proportional, autonomous updates based on the extent of the violation. Therefore, this manual option is not chosen.

4.3 Obtaining a Drug Ranking

As detailed in Section 3.2.3, it is possible to obtain an order of drugs from the minibatches. Whilst this sounds simple in theory, it is significantly more challenging to implement. Since the policy assigns a score to each drug for each cell line, it would seem logical to just take the mean score for each drug across the cell lines and sort by the drug's average score. However, sorting is a piecewise linear function, and this means that there are kinks where it is nondifferentiable. For this project, a ranking operator, that is, *argsort*, would be more useful to obtain drug indices in order of preference. However, *argsort* is even worse for differentiability since it is piecewise constant, meaning that its derivatives are either undefined or zero. The use of this function breaks the computational graph, meaning any penalty applied using this method would not have the intended effect as there would be no gradient derived from it to backpropagate. In this scenario, it could even act as a random form of regularisation, where the penalty applied to the loss could discourage overfitting by applying some variability into the training process. This would act similarly to the entropy loss that is often added to the PPO loss, which encourages exploration by penalising low-entropy policies. Since this is not of interest to this project, another method of ranking the drugs is needed.

A solution to this problem comes from a paper released by a team at Google, which introduced a fast, differentiable soft sorting and ranking algorithm [41], becoming the first to achieve a $\mathcal{O}(n\log n)$ time complexity and crucially guarantees exact results. Each drug is assigned a score for each cell line by the policy, which is how it decides the rankings of drugs, and a "soft rank" can be applied to rank the drugs for each cell line, which returns drug indices. From this, a severity score for each line is determined by applying Equation 3.4 and the final severity score is calculated by taking the mean of the scores for all cell lines.

Although the workings of this algorithm are complex and deviate far from the focus of this

project, it is important to note that there is a regularisation parameter $\epsilon > 0$. This controls the trade-off between the smoothness of this soft function (ensuring differentiability) and the approximation accuracy of the original *argsort* function. The rankings given are in the correct order regardless of the value of ϵ, however, the exact index values returned can vary significantly with higher values. For example, instead of outputting precise ranks such as $[1, 2, 3, 4, 5]$, higher values of ϵ can produce inconsistent results such as $[1.7, 2.8, 4.1, 5.1, 5.9]$, and this problem is exacerbated by having more items to rank. This poses a problem when applying Equation 3.4 to find the total penalty, since the index (rank) of a drug is used directly in this calculation. Simply rounding these indices is not a viable solution, as it does not guarantee that the correct integer values will be obtained. Therefore, tuning ϵ is critical and setting it to 0.001 provided indices nearly identical to the correct values, without sacrificing differentiability. As $\epsilon \to 0$, the "soft" ranking reverts to the "hard" ranking, that is, the regular *argsort*, which is the reason why the indices became more accurate when ϵ was reduced to 0.01 from the default value of 1.

As mentioned previously, this method of calculating the "costs" to the policy's actions differs from the typical method for constrained DRL and constrained PPO, and is only possible due to the nature of the task at hand. In those situations, costs are collected as time steps are sampled, and cost advantages are calculated alongside and in a similar manner to the regular (reward-based) advantages. This requires the use of an additional cost value network to output a prediction for the expected total cost of the actions taken, while the regular value network still outputs the expected discounted rewards for the reward advantages.

In contrast, the method devised for this project facilitates an arguably simpler and more direct approach to finding a value for $g(\theta)$. By leveraging the differentiable nature of the "soft" ranking algorithm, the overall preferred order of the drugs is found, and drugs that are ranked high up but have a high severity score are penalised. This entire process remains within the computational graph, allowing for backpropagation. Incorporating this ranking into the penalty calculation streamlines the enforcement of the constraint and reduces computational overhead compared to the cost advantage-based method, as a new value network is not necessary. This encourages the network to explore new actions (rankings), while considering that placing drugs with a large severity score high up in the rankings will result in an increased loss. The algorithm for the Lagrange loss is included for reference in Appendix A.2 for reference.

4.4 λ Values

This project uses an inequality constraint, which necessitates $\lambda \geq 0$. It is possible to enforce this through transformations of λ and several methods were considered. Clamping is a straightforward approach, however, this introduces a discontinuity at zero, which can lead to non-smooth gradients. This discontinuity can cause issues, as backpropagation relies on smooth and continuous gradients for effective updates. In addition, clamping results in zero gradients at the clamping point, which prevents gradient flow.

The softplus function [42] was also evaluated. It is smooth and differentiable, which is beneficial for this situation, but since the constraint violations in this project are typically less than |1|, softplus is not suitable because it is designed to push very small numbers to slightly larger positive values. Therefore, it modifies the value of λ too much when it is positive and leads to larger gradients than desired.

Rectified linear unit (ReLU) [43] it is simple and effective in ensuring non-negativity, but it introduces a discontinuity at zero, similar to clamping. In reality, this would probably not be much of an issue, but if λ becomes negative and stays negative, this can cause the ”dying ReLU” problem, where the ReLU function will always output 0, meaning the constraint will never be enforced after this happens. Using Leaky ReLU [44] theoretically addresses this problem by allowing a small gradient when λ is negative, however, it still allows negative values, and has a discontinuity.

Finally, Continuously Differentiable Exponential Linear Unit (CELU) [45] was chosen because it is continuously differentiable, providing smooth and continuous gradients everywhere, and the α parameter in CELU can be adjusted to ensure that negative values are very small. Setting $\alpha = 0.001$ ensures that any negative values are pushed close to zero, effectively enforcing non-negativity while still allowing smooth gradient flow. CELU does not significantly affect values less than 1, making it suitable for this project. Exponential Linear Unit (ELU) [46] was another option, as it also ensures non-negativity, but CELU is preferable because it is continuously differentiable, which avoids any issues in that respect. The choice of $\alpha = 0.001$ essentially eliminates negative values for λ, as shown in Figure 4.2, where CELU(-0.1) = -0.001. This could technically be achieved with leaky ReLU as mentioned earlier, but using CELU avoids any potential issues with discontinuities.

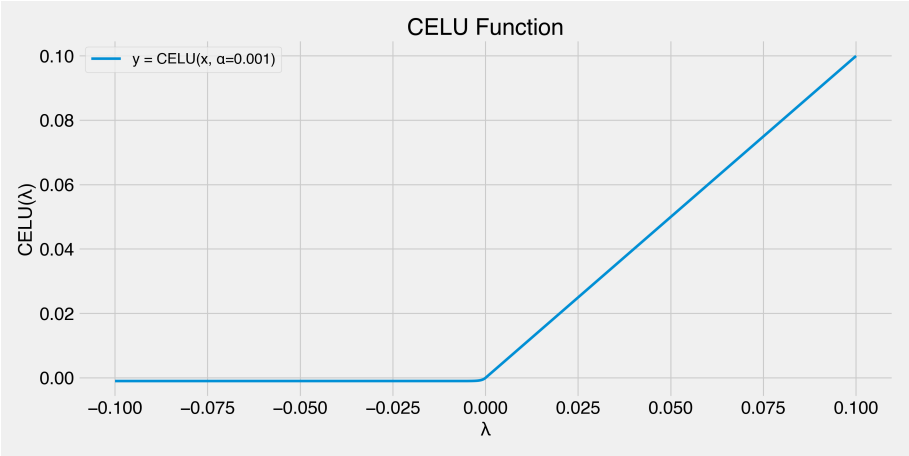


Figure 4.2: CELU with $\alpha = 0.001$

5 Results

Contents

5.1 Testing Methodology

The original PPORank paper appears to have gotten results by training over 1000 epochs or more, but due to queue times on the HPC cluster, a hard limit of 12 hours of training time was set, which generally equated to around 450 epochs. The HPC requires the specification of a walltime limit, and naturally a longer walltime tends to result in longer queue times. In general, 12 hours was a good balance between queue time and training time, since the constrained models generally reached peak performance before 400 epochs.

5.1.1 Model Performance

As with PPORank, NDCG is used to evaluate the performance of the model in ranking drugs in order of their efficacy on cancer cell lines. A graph of NDCG against epoch should clearly show how the model has learnt and what level of performance it has reached. For the constrained models, a valid NDCG value is defined as a score where at the time of a model achieving this score, the constraint violation from the output is nonpositive, i.e, the constraint is met when this score is obtained.

5.1.2 Constraint Satisfaction

To evaluate how well the severity constraint has been taken into account, a slightly different approach can be taken compared to the way $g(\theta)$ is estimated in training. The test dataset consists of 193 cell lines, for which the model predicts the ordering of all 38 drugs. As is the nature of the task, there will be different drugs ranked in each position for different cell lines; since in this instance the severity of the drugs is what we are considering, taking the average penalty for the drugs in each position across all cell lines will show the extent to which the severities are taken into account by the model. This is essentially what is being done in training too, however, given that a differentiable function is not needed here, a regular ranking function can be used to avoid any inaccuracies. For formulations with a changing coefficient over time, a graph showing both the coefficient and the constraint violation over time can provide valuable insights if needed. A target constraint 'score' has to be set, i.e. the parameter d in 2.23 and for testing, this value was set to 3.6 for all tests to make comparisons fair between formulations. The violation is calculated by subtracting the target score from the current constraint score (the value $g(\theta)$ takes). The minimum possible value for $g(\theta)$ is 3.46 and the maximum possible value is 3.97, and this was calculated by ordering the drugs according to their severity scores, as described in Section 3.2.3, meaning there is a total range of 0.51 for $g(\theta)$.

5.1.3 Hyperparameter Tuning

To find the optimal hyperparameters, a modified grid search approach is used. This method begins with large intervals between values to broadly explore the parameter space and identify regions where the best values likely reside. Once a promising region was located, the search intervals are

progressively narrowed, refining the values iteratively until a satisfactory level of performance is achieved.

For the quadratic formulations, the tuning is more straightforward, as there is only one constraint-related parameter ρ. The Lagrangian formulation is slightly more complex as it involves tuning both λ and its learning rate. Using a systematic exploration of the parameter space helps to efficiently find optimal values for these parameters. The augmented formulation requires balancing the influence of the linear and quadratic penalty terms, making the tuning process more complicated. This formulation requires tuning both λ and ρ simultaneously, as well as λ's learning rate. The iterative and refined search strategy is particularly valuable in this case, allowing for careful adjustment of these interdependent parameters to achieve the best performance.

5.2 Benchmark

Following the fixes described in Section 4.1, the following baseline results were obtained. Figure 5.1 shows the NDCG over epochs when the model is evaluated on the train data and the held out test data. As expected, there are fluctuations in performance over time, because the agent is exploring new actions, but overall performance steadily increases over time. The maximum NDCG score reached was 0.763.

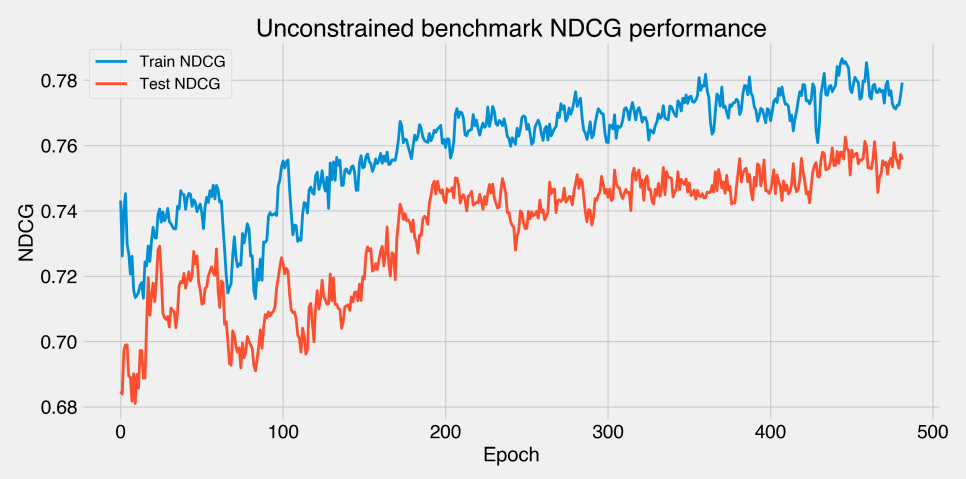


Figure 5.1: Unconstrained benchmark NDCG performance

Figure 5.2 shows the constraint violation over epochs, indicating that it grows to and oscillates around a value of 0.12. Given that the target constraint 'score', d, was set to 3.6, this means

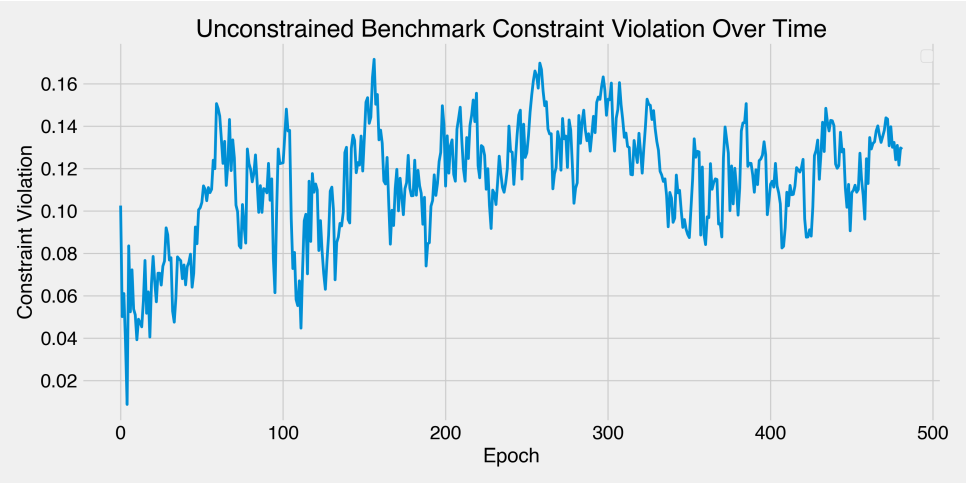


Figure 5.2: Unconstrained benchmark constraint violation

that the value of $g(\theta)$ was approximately 3.72. This choice of 3.6 as the target is strategic; it differs sufficiently from the unconstrained value to impact the rankings, yet close enough to the benchmark constraint value to allow the model to still factor in the efficacy of the drugs in the ranking. The violation of 0.12 is only 3.33% away from the target, which initially appears to be insignificant. However, considering that the total range of values that $g(\theta)$ can take is 0.51, this violation represents approximately 23.53% of the total range, indicating a more substantial deviation in the context of possible values. For the constrained formulations, only the test NDCG is plotted for clarity in the graphs, but the disparity between the train and test performance is similar in all formulations.

Rank	Average Severity Score	Most Common Drug	Severity of Most Common Drug
1	2.767	12	2.546
2	2.941	34	3.042
3	2.965	0	3.220
4	2.876	18	2.901
5	2.847	35	2.595

Table 5.1: Benchmark drug prediction severities

Table 5.1 displays the average severity score for the drugs ranked in positions 1 to 5, the most common drug in each rank, and the severity score for that most common drug. It is not possible to conduct much analysis on this table without a comparison of the constrained runs, so this will take place later. It is worth noting that each cell line will have a different ranking of drugs, so the most common drug is not recommended across all cell lines, hence the discrepancy between the average severity and severity of the most common drug for each rank.

5.3 Lagrange Formulation

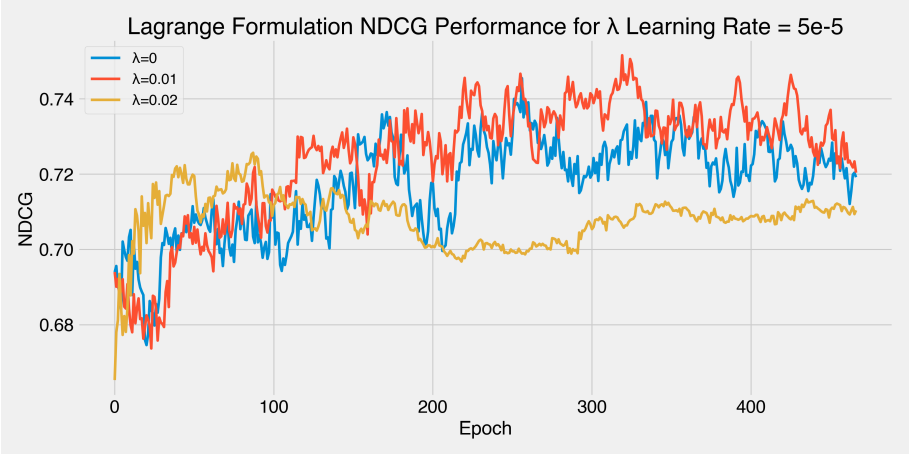


Figure 5.3: Lagrangian formulation NDCG performance for different initial λ values

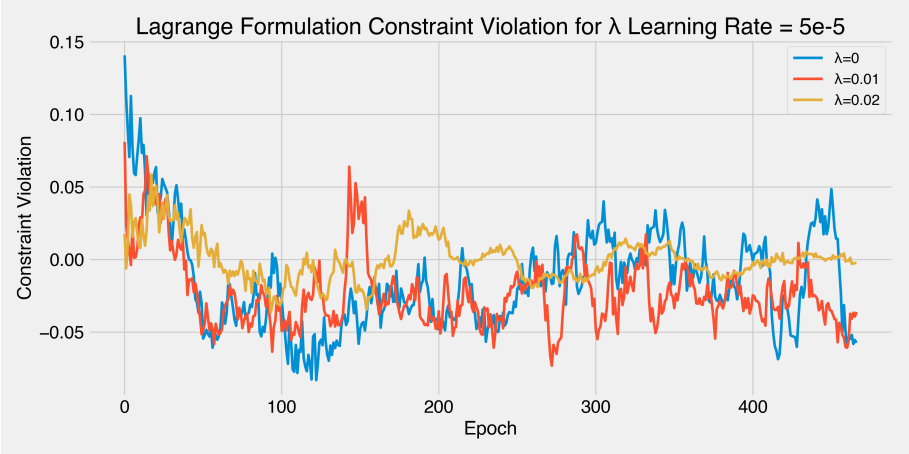


Figure 5.4: Lagrangian formulation constraint violation for different initial λ values

Figure 5.3 displays the NDCG over time for initial λ values of 0, 0.01 and 0.02, and a λ learning rate of 5×10^{-5}. The NDCGs are clearly lower than the benchmark values, but the best performance is achieved with a starting value of 0.01, with the maximum valid NDCG value being 0.751. The starting value of λ evidently has a significant effect on training dynamics. A pattern that emerges over many training runs with different parameter choices is that starting with a higher λ stifles model performance more, but the constraint violation, shown in Figure 5.4, would be less erratic and would remain closer to 0.

In the graphs with a starting λ of 0.02, the NDCG value was markedly below the other two runs, but the constraint violation settled around 0. This is corroborated by another set of tests with a lower λ learning rate to see if the learning rate is partially responsible for this behaviour.

These graphs are included in Appendix A.4, and indeed show that starting with a higher λ results in poor model performance. It appears to be important to be important to let the model learn well without enforcing the constraint too strictly at the beginning; having lower starting λ values can still result in the constraint being satisfied since λ changes over time as necessary.

This learning rate of 5×10^{-5} was chosen as it provided a good balance between granularity in updates while allowing for large enough changes so that λ can adjust adequately. A learning rate that is too aggressive effectively enforces the constraint, but at the expense of the NDCG performance. This is due to the high learning rate causing large and quick adjustments to λ in response to constraint violations. As a result, the loss is more heavily focussed on penalising the violations, potentially overshadowing the primary ranking objective, which leads to the observed plateau in performance. This effect is closely associated with having a large initial λ as the large learning rate results in a larger λ much earlier in training. Additionally, large updates can lead to instability, resulting in a wildly oscillating λ that attempts to go negative to compensate for over-enforcing the constraint, as shown in Figure 5.5 with a learning rate of 1×10^{-4}. The inclusion of the CELU function prohibits this, but the oscillations remain. This limits the model's ability to find more optimal rankings that remain feasible.

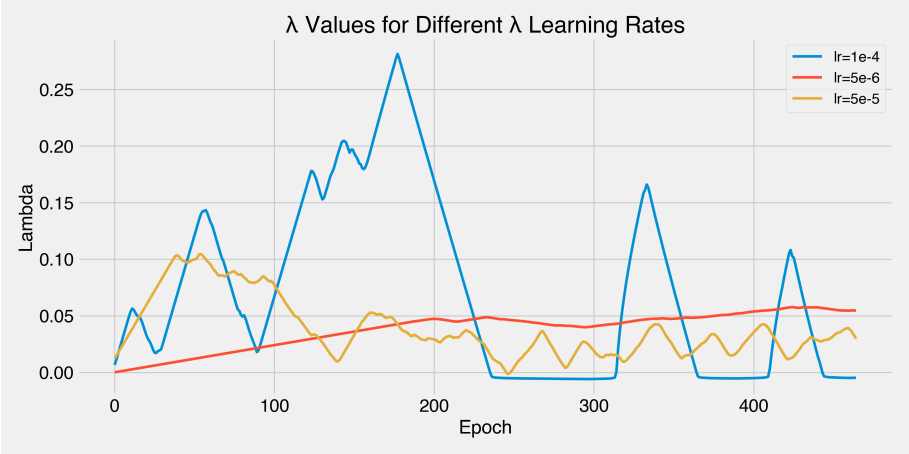


Figure 5.5: λ values over time for different λ learning rates

Conversely, a learning rate that is too passive fails to adjust λ adequately. In this scenario, λ increases slowly, meaning that the constraint is not enforced for a long time, but then the λ values remain high as it cannot fall fast enough, leading to the constraint being consistently over-met, which worsens performance. If the learning rate is far too low, the value for λ never reaches the required level, which means that the constraint is never met or is always overly satisfied, depending on the initial value. The moderate learning rate of 5×10^{-5} allows for smoother adjustments and

better convergence toward the optimal solution, while avoiding the pitfalls of overly aggressive and overly passive updates and facilitates a more effective trade-off between satisfying the constraint and optimising for NDCG.

Rank	Average Severity Score		Most Common Drug		Severity of Most Common Drug	
	Lagrangian	Benchmark	Lagrangian	Benchmark	Lagrangian	Benchmark
1	2.680	2.767	12	12	2.546	2.546
2	2.900	2.941	34	34	3.042	3.042
3	2.723	2.965	4	0	2.814	3.220
4	2.645	2.876	33	18	2.906	2.901
5	2.636	2.847	32	35	2.170	2.595

Table 5.2: Comparison of Lagrangian and benchmark severities

Table 5.2 provides another measure for portraying how the constraint has been met in the Lagrangian formulation compared to the unconstrained benchmark. The particular model chosen for this comparison had the best performance (initial $\lambda = 0.01$ and learning rate 5×10^{-5}), achieving a peak NDCG of 0.751 in an epoch where the constraint violation is -0.005, that is, the constraint is met at this point in time. Notably, the average severity for each rank is consistently lower, although the similar top two average severities and identical top two most common drugs between both approaches suggest that the model prioritises the efficacy of these two drugs to optimise the primary objective of the PPO loss function. However, for ranks below the top two, the average severities and most common drugs distinctly differ between the Lagrangian approach and the benchmark, signifying that the model makes adjustments to the drug recommendations below the top two to better meet the constraint.

Overall, these findings suggest that the Lagrangian approach can effectively integrate constraint satisfaction into the optimisation process without compromising the performance of the primary objective too much.

5.4 Quadratic Formulation

To assess the effectiveness of the quadratic loss formulation, the standard quadratic penalty as shown in Equation 3.6 is tested, as well as the thresholded quadratic penalty, as shown in Equation 3.7.

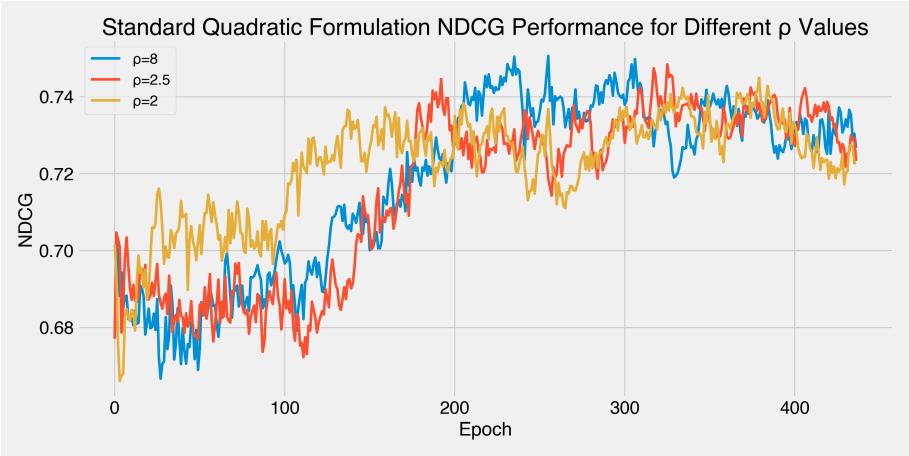


Figure 5.6: Standard quadratic formulation NDCG performance for different ρ values

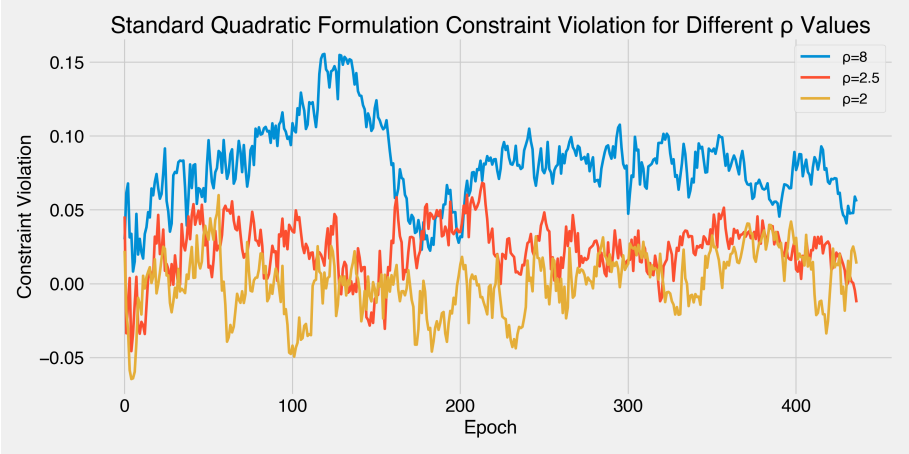


Figure 5.7: Standard quadratic formulation constraint violation for different ρ values

5.4.1 Standard Quadratic Penalty

Figure 5.6 shows the NDCG for the standard quadratic penalty formulation with ρ values of 8, 2.5 and 2. The performance is inferior to the Lagrangian formulation, as the maximum valid NDCG score is 0.746 for $\rho = 2.5$. When $\rho = 8$, the constraint violation never falls to 0, indicating that the constraint is never met and this value is insufficient to enforce it. As expected, $\rho = 2$ enforces the constraint more strongly than $\rho = 2.5$, with smaller deviations of the constraint violation from 0. However, this increased strictness meant that a higher NDCG was achieved with $\rho = 2.5$, and in this case the constraint violation was less than 0 (showing that the constraint was respected). This trend is mirrored with $\rho = 8$, which did not adequately impose the constraint, but resulted in the highest NDCG among these parameters.

The constraint violations shown in Figure 5.7, appear to be centered around 0, especially for

$\rho = 2$, as evidenced by a mean constraint violation of 0.00194 and a standard deviation of 0.0205. This is less the case for $\rho = 2.5$, whose mean constraint violation is 0.0219 and standard deviation is 0.0182. The standard deviation indicates a relatively high degree of variability in the violation values relative to their magnitudes. Therefore, the median can be used to provide a more robust measure of the central tendency. The median constraint violation for $\rho = 2$ is 0.00420, and for $\rho = 2.5$ it is 0.0182, indicating that violations are still in general reduced with a lower ρ and violations for $\rho = 2$ are closer to 0. This behaviour can be explained by the quadratic term in the loss function, which penalises both positive and negative values of the constraint violation, naturally pushing the violation towards 0. For $\rho = 2$, stronger enforcement results in a tighter clustering of constraint violations around 0, also resulting in the constraint being met more consistently. This furthers the case for the thresholded formulation, which modifies the penalty function to only penalise positive violations, allowing the model to avoid unnecessarily enforcing an equality constraint.

The choice of ρ significantly affects the performance of the model and the satisfaction of the constraint. A value of ρ that is too high fails to enforce the constraint effectively, as seen with $\rho = 8$. Conversely, a very low ρ value such as 2 enforces the constraint too strictly, potentially hindering the NDCG. The intermediate value of 2.8 strikes a better balance, enforcing the constraint while allowing the model to achieve a higher peak NDCG.

5.4.2 Thresholded Quadratic Penalty

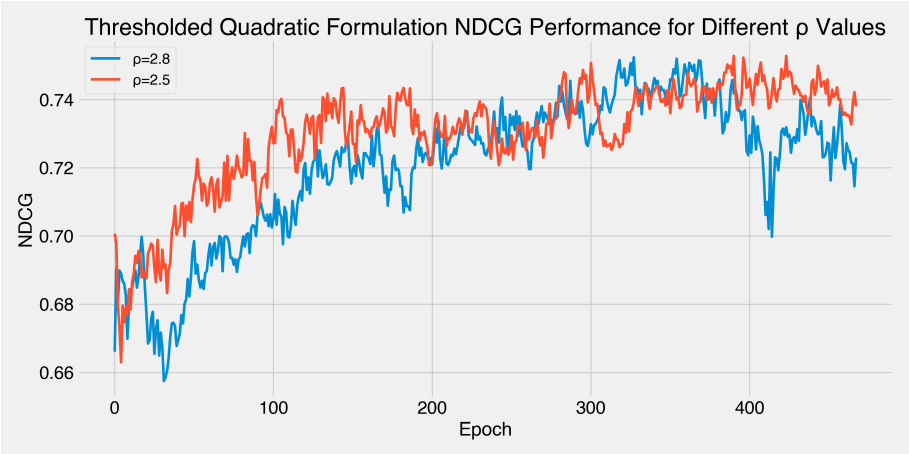


Figure 5.8: Thresholded quadratic formulation NDCG performance for different ρ values

This transformed formulation provides improved performance, with a maximum valid NDCG value of 0.752 for $\rho = 2.8$, as shown in Figure 5.8. This is also the overall maximum value for

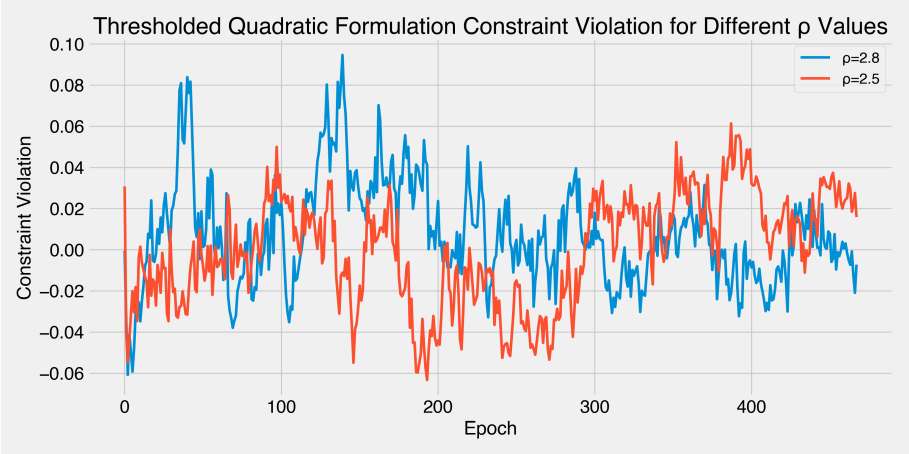


Figure 5.9: Thresholded quadratic formulation constraint violation for different ρ values

this ρ value. As detailed earlier, this formulation focusses on ensuring that the constraint is not violated, without pushing the constraint violation towards zero when it is already negative. This prevents the model from over-penalising and unnecessarily restricting its flexibility, allowing for better optimisation of the NDCG, while still trying to respect the constraint. The ρ values plotted are for 2.8 and 2.5 as these give the best results for the values tested; the reason $\rho = 2.8$ is not plotted for the standard quadratic formulation is that this is not a sufficiently small value to meet the constraint. Running this formulation with $\rho = 2$ enforces the constraint too strongly and significantly worsens the NDCG performance.

The constraint violations in Figure 5.8 are less centralised around 0 compared to Figure 5.7. Although it may appear that the violations for $\rho = 2.8$ are decreasing better than for $\rho = 2.5$, it is important to note that the NDCG for $\rho = 2.5$ is also decreasing, whereas for $\rho = 2.8$, the NDCG remains high towards the end of training, while the violation increases. It is not possible to say with certainty, but there is a possibility that if training continued, then the violation would decrease, along with the NDCG. Another point to note is that training with this formulation is less predictable; one would assume that decreasing ρ would lead to the constraint being more consistently met, as was the case with the standard quadratic formulation. However, the dynamics here are capricious, exemplified by the sudden rise in constraint violation for $\rho = 2.5$ around epoch 290. In general though, decreasing ρ did lead to lower violations overall, with the median violation for $\rho = 2.8$ at 0.00757 and for $\rho = 2.5$, at 0.00155.

Table 5.3 compares the top five rankings for the benchmark and the thresholded quadratic formulation for $\rho = 2.8$, as this gave the best results. The average severity in each rank is lower

Rank	Average Severity Score		Most Common Drug		Severity of Most Common Drug	
	Quadratic	Benchmark	Quadratic	Benchmark	Quadratic	Benchmark
1	2.623	2.767	12	12	2.546	2.546
2	2.829	2.941	34	34	3.042	3.042
3	2.666	2.965	11	0	2.476	3.220
4	2.586	2.876	32	18	2.170	2.901
5	2.742	2.847	19	35	2.902	2.595

Table 5.3: Comparison of quadratic and benchmark severities

for the quadratic formulation, and once again, the top two most common drugs are the same as the benchmark, with the lower-ranked drugs differing. This is consistent with the behaviour of the Lagrangian formulation, which displayed similar characteristics.

Overall, these findings suggest that the thresholded quadratic penalty formulation can effectively integrate the severity constraint into the optimisation process, allowing for good performance while still respecting the constraint.

5.5 Augmented Lagrangian Formulation

Results are presented for the augmented Lagrangian loss formulation with both static and adaptive ρ values, as described in Section 3.2.2.

5.5.1 Static ρ

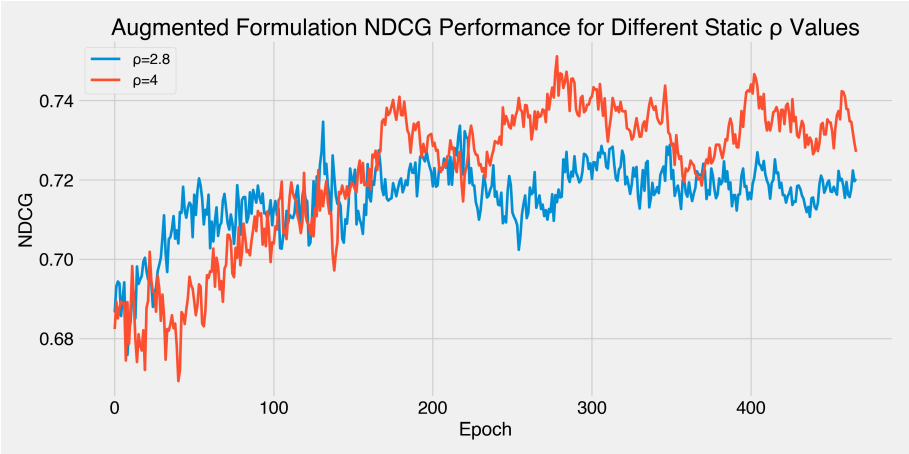


Figure 5.10: Augmented formulation NDCG performance for different static ρ values

The augmented loss formulation is the most challenging for producing good results because of an increase in the number of parameters to tune, meaning that it is the most prone to instability.

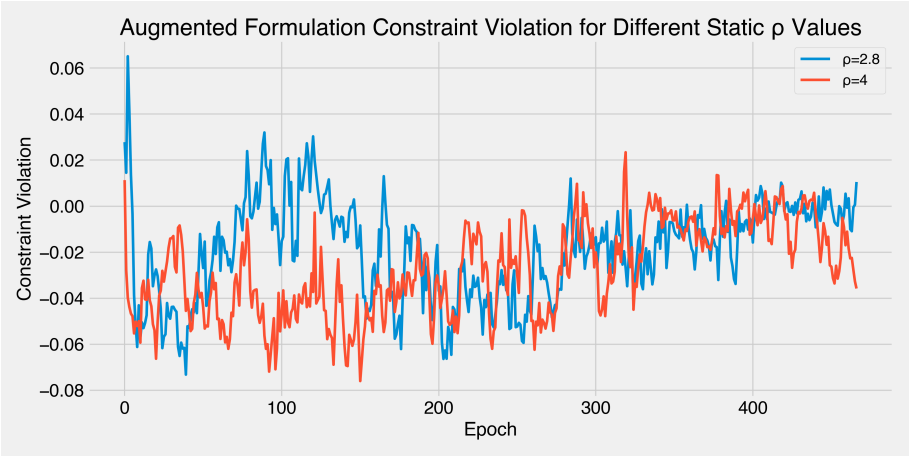


Figure 5.11: Augmented formulation constraint violation for different static ρ values

Once again, a λ learning rate of 5×10^{-5} was used, as this was a good middle ground for the Lagrangian formulation, as well as an initial λ of 0, since the quadratic aspect of this formulation already applies a penalty at the start. Starting λ at 0 reduces the frequency of NDCG crashes, as it lowers the likelihood that the constraint would be enforced too strongly. The choice of $\rho = 4$ results in the best performance, with a maximum valid NDCG of 0.751, as shown in Figure 5.10. Although the optimal ρ value for the quadratic formulation is 2.8, using this value here excessively enforces the constraint due to the presence of the Lagrangian term.

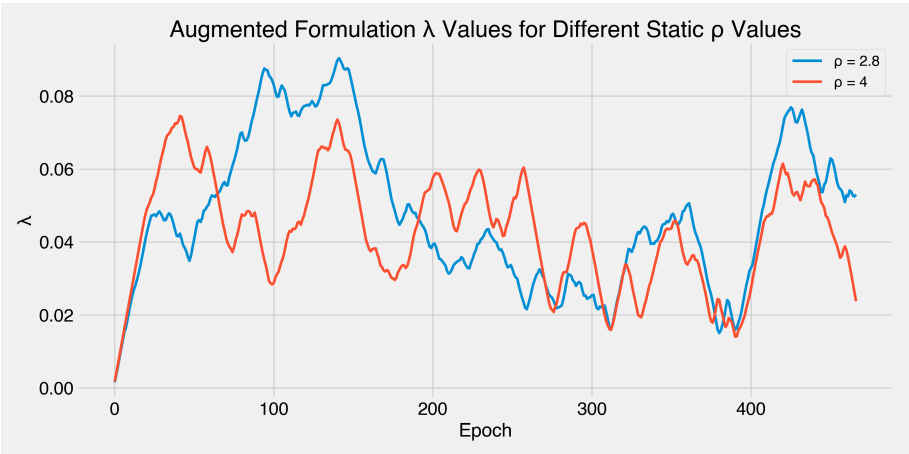


Figure 5.12: Augmented formulation λ values for different static ρ

In epochs 80 to 140, the constraint violation for $\rho = 2.8$, as depicted in Figure 5.11, is not only much higher than for $\rho = 4$, but also positive, which seems to contradict the earlier statement. However, this was early in training and was rectified by a rise in λ, which initially remained lower than for $\rho = 4$, as shown in Figure 5.12. This is likely due to the increased contribution of the quadratic term from the lower ρ value. Beyond this point, the constraint violation for

$\rho = 2.8$ remained mostly lower than $\rho = 4$, except at the end. Here, the violation for $\rho = 2.8$ approaches 0, as the constraint is being respected and λ reduces as a result, so the contribution from the quadratic term becomes more significant. This is consistent with the results of the standard quadratic formulation, as we have seen that this formulation pushes the constraint violation towards 0. Based on these results, it seems logical to adjust the quadratic term of the augmented loss to incorporate the 'max' term from Equation 3.7. However, despite extensive hyperparameter tuning and experimentation, achieving good results with this adjustment proved to be extremely challenging; the constraint was usually met, but the NDCG would be lower than in Figure 5.10.

5.5.2 Adaptive ρ

The added complexity of adjusting ρ makes this the most difficult loss formulation to tune, aside from the aforementioned formulation that failed to produce good results. A learning rate for ρ was necessary and a rate of 5×10^{-5}, the same as for λ, works well. It is also essential to implement violation thresholds for adjustments to take place. The most effective thresholds found are: -0.01 to increase ρ, reducing the contribution of the quadratic term since the constraint is met at this point, and 0.03 to decrease ρ, increasing the quadratic contribution to reduce the violations.

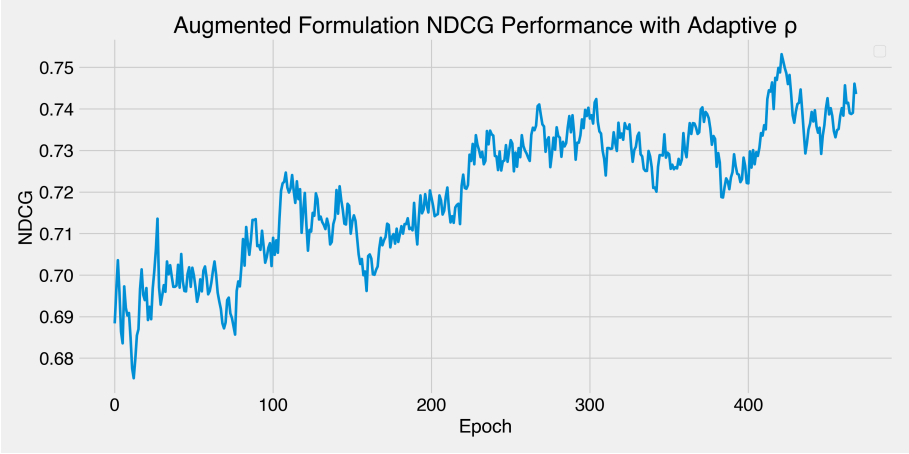


Figure 5.13: Augmented formulation NDCG performance for adaptive ρ

Figure 5.13 presents the performance with the above parameters, and a starting ρ of 2.8. However, this model only achieved a maximum valid NDCG of 0.747. The threshold and starting value of ρ greatly impact training; many other configurations of these parameters result in very poor performance and often lead to catastrophic forgetting. The flip side is that this method likely has the potential to achieve the best performance of all loss formulations tested due to the

increased control over the enforcement of the constraint. This NDCG graph shows relatively linear growth, with no sign of a plateau, which is present in all other NDCG graphs. It is impossible to say for sure, but if training were to continue beyond epoch 468, it is feasible that performance would have continued to improve.

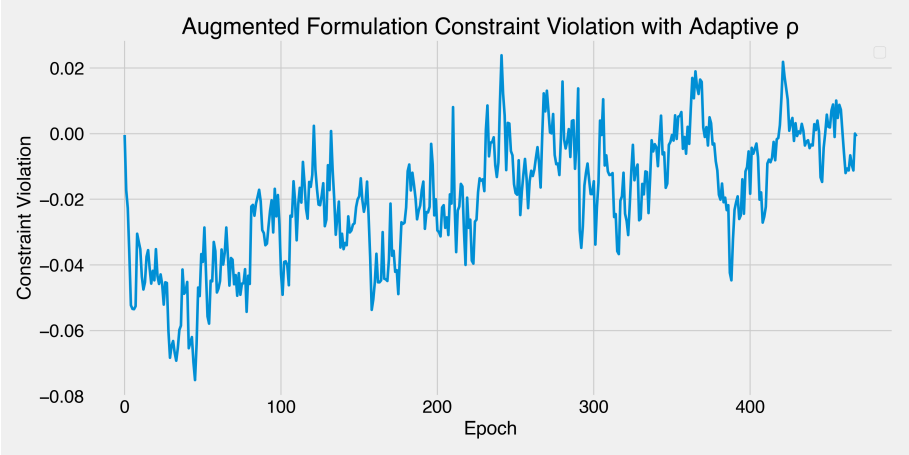
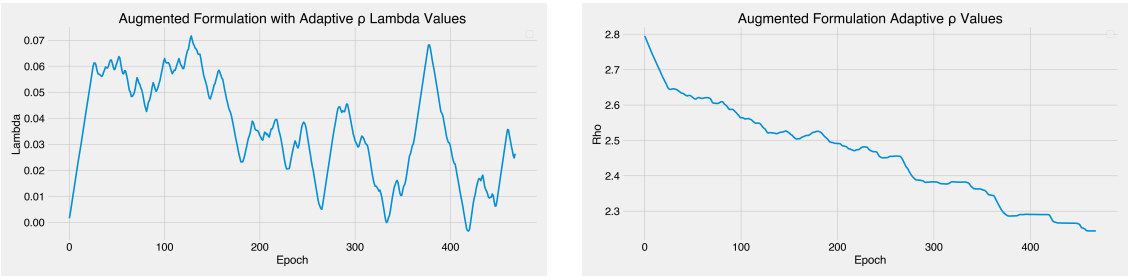


Figure 5.14: Augmented formulation constraint violation for adaptive ρ

The constraint violation displayed in Figure 5.14 shows that it remains largely below 0 for most of the training period, although it increased steadily over time. This trend can be attributed to the generally decreasing values of λ as the constraint is being met. Also, interestingly, the chosen threshold caused ρ to decrease as well, leading to an increased contribution from the quadratic term over time, explaining why the violation was pulled towards 0. The values of λ and ρ are depicted in Figure 5.15. As a result of this, the overall maximum NDCG for these parameters of 0.753 was achieved while the constraint violation was positive, and therefore this overall maximum is invalid. Alternative thresholds that have ρ increasing over time gave worse NDCG performance, since λ has to increase to compensate.



(a) λ values

(b) ρ values

Figure 5.15: λ and adaptive ρ values for augmented formulation

Table 5.4 shows the rankings for the best augmented model, which is a static $\rho = 4$. The top

two ranked most common drugs are once again identical to the benchmark and the most common drugs in ranks three and four are identical to the quadratic rankings in Table 5.3. Drug number 0 is ranked fifth, which is close to its benchmark ranking of third.

Rank	Average Severity Score		Most Common Drug		Severity of Most Common Drug	
	Augmented	Benchmark	Augmented	Benchmark	Augmented	Benchmark
1	2.704	2.767	12	12	2.546	2.546
2	2.811	2.941	34	34	3.042	3.042
3	2.596	2.965	11	0	2.476	3.220
4	2.538	2.876	32	18	2.170	2.901
5	2.885	2.847	0	35	3.220	2.595

Table 5.4: Comparison of augmented and benchmark severities

5.6 Further Comparison of Losses

The constraint violation for the Lagrangian loss is more consistently below 0 than for the quadratic loss (standard and thresholded formulations), whose violations are more erratic and more prone to becoming positive. This is likely as a result of the fixed coefficient for the penalty, whereas the Lagrangian loss has a dynamically adjusting λ to ensure that the constraint is met more often throughout training. The augmented loss also generally has constraint violations below 0 too, although this depends on the choice of parameters. As mentioned earlier, the lowest median constraint violation for the best performing quadratic loss is 0.00757. For the best performing Lagrange loss, using an initial $\lambda = 0.01$, this value is -0.0274 and for the augmented loss it is -0.0285, using a static $\rho = 4$.

The most common drugs in ranks one and two, drugs 12 and 34, are identical between the unconstrained benchmark and constrained models. The severity scores for the drugs range from 2.170 to 3.666, and the full table showing all the severity scores is in Appendix A.3. Notably, drug number 12 has a relatively low severity score of 2.546, meaning that it is illogical for the model to move the drug from this top position. The severity score for drug number 34 is roughly in the middle at 3.042, but the constrained models deem it necessary to keep this drug ranked second to maintain a high NDCG score. The final drug common to all constrained formulation rankings is drug 32, which has the lowest drug severity score of all. This drug was most commonly ranked eighth in the benchmark, so all models astutely determined that moving it higher would help satisfy the constraint while not significantly lowering the NDCG.

6

Evaluation

The framework provided by this project for incorporating constraints into a DRL ranking model, specifically based on PPO, has been shown to work in the results. The three types of loss formulations; Lagrangian, quadratic penalty, and augmented Lagrangian, can all apply the severity constraint with similar degrees of successused by this projectameter choices. A summary of the best performing models for each loss type is given in Table 6.1.

Loss Type	**Peak NDCG**	**Median Constraint Violation**
Benchmark	0.763	0.117
Lagrangian	0.752	-0.0274
Quadratic penalty	0.752	0.00155
Augmented Lagrangian	0.751	-0.0284

Table 6.1: Peak NDCG and Median Penalty for Different Types of Loss

All loss types provide nearly identical peak performance, indicating that they are all able to converge on what appears to be the optimal ranking given the constraint. In comparison to the unconstrained benchmark, the constrained model's NDCG is worse, which is expected given the constraint. However, it should be noted that the benchmark NDCG of 0.763 was achieved over only 480 epochs, and given more training time, it would likely reach a higher score since there were no signs of a plateau. In PPORank [1], the peak performance of 0.761 was reached with 1000 epochs of training, albeit with a much larger drug candidate list of 265, compared to the 38 drugs in this project. The constrained models learnt over roughly 450 epochs, but the peak performance was reached well before the end, indicating that a performance ceiling had been hit. In addition, PPORank produced results using multi-fold cross validation for a more reliable performance figures,

but given the time limitations from the HPC cluster, employing a similar strategy for this project was not reasonable. There is no direct comparison to other papers that can be made for this project, as this is the first to incorporate any constraint in a drug ranking context.

As illustrated in the results section and Table 6.1, the quadratic penalty loss was less effective in consistently adhering to the constraint throughout training, and for this reason, when using this loss, it becomes more likely that the highest NDCG is achieved while the constraint is violated. Also, given that it does not offer a meaningful performance advantage, this use of this loss formulation is not recommended over the other types.

The Lagrangian and augmented Lagrangian formulations have essentially identical peak NDCG and median constraint violations; both are effective at maintaining a ranking over time that respects the constraint. However, given the greatly increased complexity and difficulty in tuning the augmented model, and following Occam's razor, it is logical to choose to use the Lagrangian as the preferred loss type.

This framework for constrained rankings can be generalised for use cases beyond drug ranking. The drug severity score can be replaced with any measure that depends on the ranking, and the logarithmic weighting function in Equation 3.4 can then be applied to this new measure to create a corresponding penalty, with adjustments to the parameters of the weighting function to accommodate the number of items being ranked. Furthermore, the soft ranking algorithm used to obtain rankings is applicable to other contexts, ensuring adaptability in various scenarios. The PPORank model is, of course, specific to drug ranking, but the method used in this project to adapt it to incorporate constraints is applicable to other DRL ranking problems.

Conclusions

The goal of this project is to produce a DRL model that is able to rank drugs according to their efficacy for unseen cancer cell lines, while taking into account a constraint based on the severity of the drug's side effects. The PPORank model developed in [1] is used as the foundation and uses PPO as its backbone. In this model, the policy network (actor) determines the ranking of the drugs, while the value network (critic) estimates the expected return and guides the policy updates by providing a baseline for advantage estimation, specifically using GAE.

In order to incorporate this constraint into the model, a "severity score" is created for each drug using the ADReCS database, which facilitates the quantification of the severity of a drug's side effects. The scores reflect both the frequency and the severity of side effects and are normalised for comparability. The PPO loss used in PPORank must be modified to include a penalty for the model's rankings based on the severity scores of the drugs and their positions. In particular, a drug with a high severity score ranked near the top incurs a more significant penalty, since these top-ranked drugs are under the most consideration. This penalty forms the parameter $g(\theta)$, defined as the constraint function, and a threshold d is set to limit the maximum allowed penalty, thus constraining the placement of high-severity drugs at the top of the rankings. Three main types of loss formulations that include $g(\theta)$ are considered: Lagrangian, quadratic penalty, and augmented Lagrangian.

To calculate $g(\theta)$, conventional methods for determining the cost or penalty of an action, or in this case a ranking, are not suitable. These methods typically compute cost advantages in a similar manner to reward advantages, which requires the creation of an additional cost-value network to estimate the expected cost. However, this introduces complexity and potential instability, and these approaches lack an effective method for controlling the threshold d, since advantages are a relative measure. Therefore, $g(\theta)$ directly uses the rank of a drug to calculate the penalty, necessitating the creation of a new method.

This method estimates drug rankings in the update of the policy, and then $g(\theta)$ can be calculated, with its construction retaining some of the benefits of using cost advantages. However, to obtain a ranking for the drugs, traditional sorting functions cannot be used because they are

nondifferentiable. To address this, a soft ranking algorithm is employed to approximate the ranking function and keep the calculations within the computational graph.

The results demonstrate the effectiveness of this method in accounting for the drug severity constraint. In comparison to the unconstrained benchmark, all loss types were able to successfully meet the requirements of the severity constraint, whilst maintaining a reasonable and very similar level of performance. The Lagrangian loss is considered to be the preferred option due to its ability to dynamically adjust λ to optimise NDCG and meet the constraint requirement. It also met the constraint more consistently throughout training than the quadratic penalty loss type and is significantly easier to tune than the augmented Lagrangian formulation despite having nearly identical performance. Furthermore, this framework used for constrained drug ranking can be generalised to other DRL ranking problems.

Future work includes creating a more comprehensive drug severity scoring system that better encapsulates the side effects of each drug. There is much more information contained within the ADReCS database than was used in this project, however, more in-depth medical knowledge than I have is required to properly leverage it. For the purpose of this project, the severity scores used are sufficient, but the system is far from perfect, and there is considerable scope for improvement. Also, as mentioned in Chapter 6, testing would ideally take place over multiple folds of data, but given the time it would take and the limitations of using the HPC cluster, this was not possible. Therefore, validating the model's performance over the various loss types in this way would be worthwhile. There is still a level of error when using the soft ranking method since the indices returned, although close to the true value, are still sometimes slightly off; looking for ways to improve this accuracy may help improve the performance of the model and the enforcement of the constraint, since the indices (ranks) are directly used in the calculation of $g(\theta)$. Finally, other loss types may be more effective than those tested in this project. For example, instead of using thresholding for the quadratic penalty, one can use $L(\theta, \lambda) = f(\theta) + \frac{1}{\rho}((g(\theta) - d) \cdot \text{abs}(g(\theta) - d))$, which keeps the benefits of squaring but retains the sign so negative violations are not penalised.

A

Appendix

A

A.1 Code

The code for this project is available in the following GitHub repository: `https://github.com/SavrajSian/Constrained_PPORank`

A.2 Algorithm

The algorithm for this project's constrained version of PPORank is presented below for the Lagrange loss. For reference, rollout storage is how the required parameters like rewards and actions are stored when episodes are sampled. PPO epoch represents how the experience buffer (rollout storage) is split up into batches for use in the PPO loss.

Algorithm 1 Constrained PPORank

Input: Labelled training dataset T, input Learning rate η, discount factor γ, PPO epoch K, GAE discount factor λ, clip range ϵ, coefficients c_1, c_2, parallel actors P, minibatch size m, ϵ for soft ranking, initial value of λ, λ learning rate β

Output: model parameters θ

Initialize actor π_θ and critic V_θ with parameters θ
Initialize $\theta_{\text{old}} \leftarrow \theta$
Initialize the rollout storage $\mathcal{B}$
for epoch = 1, 2, … **do**
    for actors = 1 to P **do**
        Run policy $\pi_{\theta_{\text{old}}}$, sample episodes and compute advantage and target value estimates.
    end for
    Update rollout storage with time steps data
    for PPO epoch = 1 to K **do**
        for minibatch = 1 to M **do**
            Obtain ranking for each cell line using policy network π_θ and apply soft ranking
            Calculate $g(\theta)$ using Equation 3.4 and by taking mean of all penalties
            Calculate total loss using surrogate PPO loss in Equation 2.13, $g(\theta) - d$, and λ.
            Optimise total loss with respect to θ
        end for
        Update λ based on average constraint violation: $\lambda \leftarrow \lambda + \beta \cdot$ average violation
    end for
    $\theta_{\text{old}} \leftarrow \theta$
end for
return θ

A.3 Severity Scores

Table A.1 contains the severity score for all drugs in ascending order.

Drug	Severity
32	2.170
23	2.339
6	2.407
11	2.476
8	2.489
12	2.546
27	2.590
35	2.595
20	2.610
30	2.630
7	2.633
36	2.680
2	2.685
3	2.722
21	2.759
31	2.785
4	2.814
29	2.857
18	2.901
19	2.902
33	2.906
28	2.965
37	3.000
25	3.036
9	3.039
34	3.042
10	3.058
24	3.066
17	3.087
26	3.200
22	3.205
14	3.214
0	3.220
5	3.243
1	3.245
16	3.272
15	3.371
13	3.666

Table A.1: Drug severity scores

A.4 Graphs

The following graph shows the NDCG for the Lagrangian formulation, with different initial λ values and a λ learning rate of 1×10^{-5}:

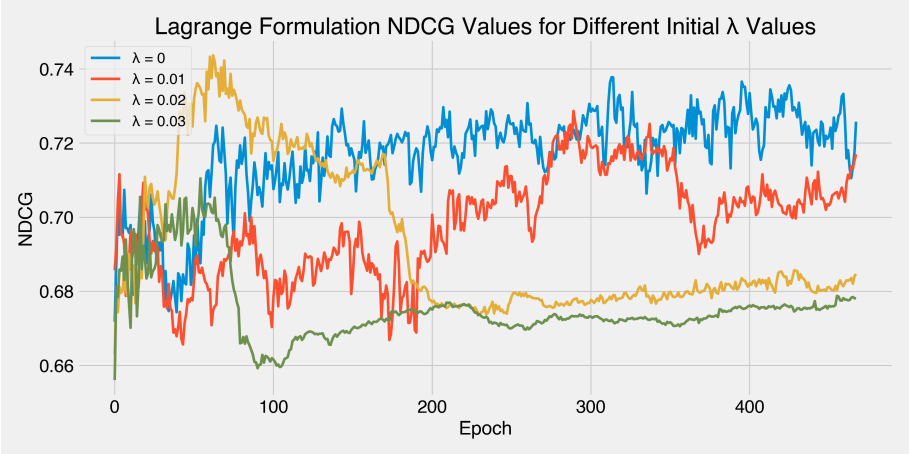


Figure A.1: Lagrangian formulation NDCGS for different initial λ values with λ learning rate 1×10^{-5}

Bibliography

[1] M. Liu, X. Shen, and W. Pan, "Deep reinforcement learning for personalized treatment recommendation," *Statistics in Medicine*, vol. 41, 2022.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction - second edition*. The MIT Press, 2018.

[3] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[4] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," MIT Press, 1999.

[5] M. Morales, *Grokking Deep Reinforcement Learning*. Manning Publications, 2020, ISBN: 9781617295454.

[6] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[7] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning.," *Journal of Machine Learning Research*, vol. 5, no. 9, 2004.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[9] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *CoRR*, vol. abs/1802.09477, 2018.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[11] G. Matheron, N. Perrin, and O. Sigaud, "Understanding failures of deterministic actor-critic with continuous action spaces and sparse rewards," in *Lecture Notes in Computer Science*. Springer International Publishing, 2020, pp. 308–320.

[12] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, *Trust region policy optimization*, 2015.

[13] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[14] webpage on NDCG.

[15] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *Proceedings of the ADKDD'17*, 2017, pp. 1–7.

[16] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[17] W. Yang, J. Soares, P. Greninger, *et al.*, "Genomics of Drug Sensitivity in Cancer (GDSC): a resource for therapeutic biomarker discovery in cancer cells," *Nucleic Acids Research*, vol. 41, no. D1, pp. D955–D961, 2012.

[18] F. Iorio, T. Knijnenburg, D. Vis, *et al.*, "A landscape of pharmacogenomic interactions in cancer," vol. 166, Jul. 2016.

[19] J. C. Costello, L. M. Heiser, E. Georgii, *et al.*, "A community effort to assess and improve drug sensitivity prediction algorithms," *Nature biotechnology*, vol. 32, no. 12, pp. 1202–1212, 2014.

[20] C. Suphavilai, D. Bertrand, and N. Nagarajan, "Predicting cancer drug response using a recommender system," *Bioinformatics*, vol. 34, no. 22, pp. 3907–3914, 2018.

[21] E. Altman, "Constrained markov decision processes," 1999.

[22] J. Achiam and D. Amodei, "Benchmarking safe exploration in deep reinforcement learning," 2019.

[23] C. Yu, J. Liu, S. Nemati, and G. Yin, "Reinforcement learning in healthcare: A survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–36, 2021.

[24] M. E. Andersen, R. S. H. Yang, C. T. French, L. S. Chubb, and J. E. Dennison, "Molecular circuits, biological switches, and nonlinear dose-response," *Environmental Health Perspectives*, pp. 971–978, 2002.

[25] P. Prasse, P. Iversen, M. Lienhard, *et al.*, "Matching anticancer compounds and tumor cell lines by neural networks with ranking loss," *NAR Genomics and Bioinformatics*, vol. 4, Jan. 2022.

[26] R. Su, Y. Huang, D.-g. Zhang, G. Xiao, and L. Wei, "Srdfm: Siamese response deep factorization machine to improve anti-cancer drug recommendation," *Briefings in Bioinformatics*, vol. 23, Jan. 2022.

[27] A. Partin, T. S. Brettin, Y. Zhu, *et al.*, "Deep learning methods for drug response prediction in cancer: Predominant and emerging trends," *Frontiers in Medicine*, vol. 10, p. 1 086 097, 2023.

[28] Z. Zhao, K. Li, C. Toumazou, and M. Kalofonou, "A computational model for anti-cancer drug sensitivity prediction," in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2019, pp. 1–4.

[29] M. Manica, A. Oskooei, J. Born, V. Subramanian, J. Sáez-Rodríguez, and M. Rodriguez Martinez, "Toward explainable anticancer compound sensitivity prediction via multimodal attention-based convolutional encoders," *Molecular pharmaceutics*, pp. 4797–4806, 2019.

[30] Y. Chang, H. Park, H.-J. Yang, *et al.*, "Cancer drug response profile scan (cdrscan): A deep learning model that predicts drug effectiveness from cancer genomic signature," *Scientific Reports*, vol. 8, Jun. 2018.

[31] L. Rampášek, D. Hidru, P. Smirnov, B. Haibe-Kains, and A. Goldenberg, "Dr.vae: Improving drug response prediction via modeling of drug perturbation effects," *Bioinformatics (Oxford, England)*, vol. 35, Mar. 2019.

[32] S. Ma, J. Lee, N. Serban, and S. Yang, "Deep attention q-network for personalized treatment recommendation," *arXiv preprint arXiv:2307.01519*, 2023.

[33] M.-C. Cai, Q. Xu, Y.-J. Pan, *et al.*, *Adrecs: An ontology database for aiding standardization and hierarchical classification of adverse drug reaction terms*, 2015. DOI: 10.1093/nar/gku1066.

[34] M. J. D. Powell, "A method for nonlinear constraints in minimization problems," in *Optimization*, R. Fletcher, Ed., 1969, pp. 283–298.

[35] R. Glowinski and A. Marroco, "Sur l'approximation, par 'elements finis d'ordre un, et la resolution, par penalisation-dualite d'une classe de problemes de Dirichlet non lineaires," pp. 41–76, 1975.

[36] A. V. Fiacco and G. P. McCormick, "Nonlinear programming: Sequential unconstrained minimization techniques," 1968.

[37] F. Yang, W. Zhou, H. Sikchi, and D. Held, "Self-paced policy optimization with safety constraints," 2022.

[38] L. Zhang, L. Shen, L. Yang, *et al.*, *Penalized proximal policy optimization for safe reinforcement learning*, 2022.

[39] N. Wagener, B. Boots, and C.-A. Cheng, *Safe reinforcement learning using advantage-based intervention*, 2021.

[40] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psychology of Learning and Motivation*, vol. 24, pp. 109–165, 1989.

[41] M. Blondel, O. Teboul, Q. Berthet, and J. Djolonga, *Fast differentiable sorting and ranking*, 2020.

[42] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing.," 2000, pp. 472–478.

[43] K. Fukushima, "Visual feature extraction by a multilayered network of analog threshold elements," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 4, pp. 322–333, 1969.

[44] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[45] J. T. Barron, *Continuously differentiable exponential linear units*, 2017.

[46] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*, 2016.