Deep reinforcement learning for complex evaluation of one-loop diagrams in quantum field theory

Andreas Windisch, Thomas Gallien, and Christopher Schwarzlmüller

Phys. Rev. E **101**, 033305 — Published 18 March 2020

DOI: 10.1103/PhysRevE.101.033305

Deep reinforcement learning for complex evaluation of one-loop diagrams in quantum field theory

Andreas Windisch,[1, 2, *] Thomas Gallien,[2, †] and Christopher Schwarzlmüller[2, ‡]

[1]*Department of Physics, Washington University in St. Louis, MO 63130, USA*

[2]*Silicon Austria Labs GmbH, Inffeldgasse 25F, 8010 Graz, Austria*

(Dated: February 28, 2020)

In this paper we present a novel technique based on deep reinforcement learning that allows for numerical analytic continuation of integrals that are often encountered in one-loop diagrams in quantum field theory. In order to extract certain quantities of two-point functions, such as spectral densities, mass poles or multi-particle thresholds, it is necessary to perform an analytic continuation of the correlator in question. At one-loop level in Euclidean space, this results in the necessity to deform the integration contour of the loop integral in the complex plane of the square of the loop momentum, in order to avoid non-analyticities in the integration plane. Using a toy model for which an exact solution is known, we train a reinforcement learning agent to perform the required contour deformations. Our study shows great promise for an agent to be deployed in iterative numerical approaches used to compute non-perturbative 2-point functions, such as the quark propagator Dyson-Schwinger equation, or more generally, Fredholm equations of the second kind, in the complex domain.

PACS numbers: 02.70.-c, 07.05.Mh, 11.55.Bq, 02.30.Cj

I. INTRODUCTION

Studying two-point functions in the complex domain is a worthwhile endeavor, as valuable insights about the properties of the propagating degree of freedom become accessible. Information about masses, multi-particle thresholds, possible composite nature, and even a sufficient criterion for removing the degree of freedom from the physical state space, can be derived once the analytic structure of the correlator is known. In the realm of non-perturbative Quantum Chromo Dynamics (QCD), the continuum approach of Dyson-Schwinger Equations (DESs) and Bethe-Salpeter Equations (BSEs) can be used to study hadron phenomenology, see e. g. [1–8], which also requires knowledge of the analytic properties of some of the quantities involved. Solving two-point functions with cubic vertices at one-loop level for complex momentum squares, however, is already a technically challenging task, as the integration contour of the radial variable associated with the loop momentum, expressed in hyperspherical coordinates in Euclidean space, cannot be maintained along the positive real half-axis once the external momentum square is allowed to assume complex values. Since there are two momenta involved, an external momentum, and a loop momentum, and because we consider cubic interactions, at least one of the propagators in the loop has to carry a mixed momentum. With an adequate choice of coordinates, the four-fold integral over the Euclidean loop momentum can thus be reduced to a two-fold integral over the radial component of the loop momentum, as well as over the angle between the external momentum and the loop momentum, see Appendix A. As discussed on the basis of a very simple example in [9], the angular integral then produces branch cuts in the complex plane of the radial integration variable, and one has to deform the contour in order to avoid the cut, as well as poles that might be present as well. Such strategies have been applied in various situations, see e. g. [10–18]. While this task becomes very tedious already in a situation where the required deformations can be deduced from analyzing the analytic properties in the integration plane, in non-perturbative approaches, such as Dyson-Schwinger equations of two-point functions in the complex domain, it becomes extremely challenging, see e. g. [11, 19–22] and references therein. These equations are such, that the function that is to be computed on the left hand side of the equation also appears in the integrand on its right hand side, which is the general form of a Fredholm integral equation of the second kind. The analytic properties of the integrand can thus only be determined for a given starting guess for the unknown function to be computed, but will change throughout the next iteration steps. A contour that was valid for the initial guess of the function would then have to be re-adjusted. The goal of this study is to provide a proof of principle that a deep reinforcement learning (DRL) agent (see e. g. [23] for a recent review and [24] for a standard text book on the subject) can be trained to conduct the contour deformations as needed. Such an agent could then be used in an iterative setting by deducing the contour deformation from observing the integration plane before each iteration step is conducted. With this approach, an analytic continuation of Dyson-Schwinger Equations could become feasible.

The paper is organized as follows. In Section II, we re-

[*]Electronic address: windisch@physics.wustl.edu

[†]Electronic address: thomas.gallien@silicon-austria.com

[‡]Electronic address: christopher.schwarzlmueller@silicon-austria.com

view the toy model that we used to train the agent. In Section III we provide a short introduction to deep reinforcement learning, and Section IV addresses some of the prerequisites needed for training the agent on the toy model. The numerical results are presented in Section V, and we conclude in Section VI, where we specifically focus on how to further improve upon the agent's performance and also on the question of how to modify our approach such that it directly applicable to Dyson-Schwinger equations. Finally, Appendix A gives an overview of our convention.

II. THE TOY MODEL

A. Setting the stage

The toy model we chose for this study features propagators of the Gribov type, the so-called *i*-particles [25]. Apart from being physically interesting, there is another aspect that renders this model particularly useful for our cause. The correlator that we are interested in has an exact solution, and has been used in a study on numerical contour deformations [12] that reproduced the analytic result perfectly. This allows us to compare the deformations provided by the trained agent with the ones that have been obtained analytically in [12], where the criterion of similarity is qualitative, as we discuss in this paper. With that, we can deduce that we could solve the integral successfully without the necessity of actually evaluating it.

The starting point for our study is given by equation (32) in [25], which is an expression for a correlator that has also been the main focus of [12]. In D dimensions, the correlator can be expressed as

$$\mathscr{G}(p^2) = \int \frac{d^D q}{(2\pi)^4} \frac{1}{(p-q)^2 - i\sqrt{2}\theta^2} \frac{1}{q^2 + i\sqrt{2}\theta^2}, \tag{1}$$

with an external 4-momentum p and loop 4-momentum q. The mass parameter θ is given through the relation

$$2\sqrt{2}\theta^2 \equiv 1, \tag{2}$$

which matches the choice of [25], where this particular definition has been selected for convenience. In our study we use the same definition, as we want to work on the exact same problem as the one treated in [25]. This allows for direct comparison, if desired.
The problem at hand is motivated by constructing physically meaningful, composite operators in the Gribov-Zwanziger framework, see [25] for details. $\mathscr{G}(p^2)$ is the D-dimensional, Euclidean correlation function of such composite operators $\mathcal{O}$ (comprised of pairs of *i*-particles) at one-loop order,

$$\langle \mathcal{O}(x)\mathcal{O}(0)\rangle = \int \frac{d^D p}{(2\pi)^D} \exp\{ip\cdot x\}\mathscr{G}(p^2). \tag{3}$$

Equation (1) is the expression the we would like to solve in $D = 4$ dimensions. In [25], an exact solution to this integral is presented (after it has been regularized), which reads

$$\mathscr{G}_{sub}(x) = \frac{1}{16\pi^2}\left(1 - \frac{\pi}{2x} + \frac{\sqrt{1-x^2}}{x}\arccos(x)\right), \tag{4}$$

where the subscript *sub* indicates, that the original integral has been regularized in the spirit of Bogoliubov, Parasiuk, Hepp and Zimmermann (BPHZ) [26–29]. The detailed calculation is described in [25]. Note, that the x in equation (4) is not a spatial coordinate, but corresponds to the complex square of the external loop momentum, that is, $x \equiv p^2 \in \mathbb{C}$. After switching to hyperspherical coordinates (see Appendix A), regularizing the integral using the BPHZ scheme, as well as integrating the two trivial angles of the 4-dimensional hypersphere, the loop integral, equation (3), becomes

$$\mathscr{G}_{sub,rescaled}(x) = \frac{2}{\pi}\int_0^\infty dy \frac{y}{y^2+\frac{1}{4}} \tag{5}$$
$$\times \int_{-1}^{+1} dz \sqrt{1-z^2}\frac{-x+2\sqrt{x}\sqrt{y}z}{\left(x+y-2\sqrt{x}\sqrt{y}z-\frac{i}{2}\right)},$$

where the additional subscript *rescaled* indicates that the loop integral (3) has been multiplied by a factor of $16\pi^2$ to eliminate the same factor in the denominator of the solution (4). As before, x is the complex external momentum square, and y is the complex square of the loop momentum q, that is, $y \equiv q^2 \in \mathbb{C}$. z is the cosine of the angle between the external and the loop momentum, $z \equiv \cos\theta$. For details as of how to arrive at equation (5) when starting with equation (3), see [12].

B. The goal

Now that we have revisited all the required expressions, let us state the goal of this study. In [12] it was shown, that the results of the integral equation for the F^2 correlator presented (and analytically solved) in [25] can be reproduced numerically by applying suitable contour deformations. The requirements for successful contours are:

- They avoid the branch cut in the integration plane.
- They satisfy continuous deformability with respect to the original contour along the positive real half-axis

The second requirement ensures, that no residue of any pole is picked up by the contour. The reason for this is the following. We want the integral along the deformed contour to acquire the same value as the integral along the positive real half-axis would have produced, if the branch cut had not interfered with this path. This can only be achieved, if the original contour is continuously deformable into the new one, as the closed contour of the composition of the original one, and the deformed one, would pick up the residue of any enclosed pole, which would in turn render the results of the two integration paths unequal. We thus consider the agent to be successful, if it is able to produce contours that satisfy these conditions.

III. METHODOLOGY

The goal is to find a valid integration contour in the complex plane of the radial integration variable y for any given external momentum square x. In our approach we achieve this goal by reformulating the problem such that it becomes applicable to deep reinforcement learning. More specifically, we define the task of contour deformation as an episodic game, where a reinforcement learning agent is trained to escape a maze in the presence of traps, trying to reach a dedicated target location. The analogy to the original problem is as follows: The branch cut essentially defines a maze in a continuous space and cannot be crossed by the agent, while the target location is given by an ϵ-area around the cut-off on the positive real half-axis. Since the agent starts each episode at the origin of the complex plane, the path taken to the target location corresponds then to the desired contour that solves the problem. The traps are basically represented by the poles, since, if the agents picks up a residue along its way to the target location, the game is considered to be lost, regardless of whether the agent reaches the target location.

A. Deep reinforcement learning

Deep reinforcement learning describes a class of goal-orientated machine learning algorithms taking advantage of powerful function approximators in the context of deep learning [23, 24]. Unlike supervised or unsupervised machine learning, these algorithms do not require a dedicated set of training data, since they are designed to learn from experience by interacting with their environment. The key ingredient thereby is a scalar reward signal, which essentially reinforces the learning entity to pick the desired actions. It is important to note that the reward signal should not be mistaken as an error signal in a supervised setting, since it can, but doesn't need to, represent an error signal, nor is it required to be differentiable. One can encounter situations where the reward signal is sparse and rarely available, like e. g. in case of a game where the information about failure or success is provided only at the end of an episode, which poses additional challenges.

B. Fundamentals and Definitions

The fundamental working principle of reinforcement learning algorithms is based upon the concept of Markov decision processes (MDPs) and involves two entities, an *agent*, and an *environment* [24]. Given a state $s_t \in \mathcal{S}$ at time step t, the agent interacts with the environment by picking an action $a_t \in \mathcal{A}(s_t)$ according to a policy $\pi(a_t|s_t)$, where $\mathcal{S}$ and $\mathcal{A}(s_t)$ denote the state- and action space, respectively. The policy is considered to be stochastic in general, hence the agent samples the action from a conditional distribution. In that sense, we treat a deterministic policy $\mu(s_t)$ as a special case $\pi(a_t|s_t) = \delta(a_t - \mu(s_t))$, where $\delta(.)$ denotes the Dirac distribution. The environment responds to the action a_t by setting the consecutive next state s_{t+1} with probability $\Pr\{S_{t+1} = s_{t+1}|s_t, a_t\}$ [40] and gives rise to a reward $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$, where $\mathcal{R}$ defines the reward space. Since the reward is considered to be stochastic as well, the dynamics of the MDP is entirely determined by the probability $\Pr\{S_{t+1} = s_{t+1}, R_{t+1} = r_{t+1}|s_t, a_t\}$.

The goal in reinforcement learning is to find a policy $\pi(.)$ such that the cumulative expected reward is maximized. The definition of the return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ provides a performance measure, where the parameter γ, $0 < \gamma \leq 1$, denotes the discount rate and essentially controls the impact of future states on the agent's decisions. Furthermore, we introduce the state-value function $v_\pi(s_t) = \mathbb{E}_\pi\{G_t|s_t\}$ and the action-value function $q_\pi(s_t, a_t) = \mathbb{E}_\pi\{G_t|s_t, a_t\}$, where the expectation is taken with respect to the policy π. Both value functions are maximized by an optimal policy $\pi_*(.)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, hence $v_*(s_t) = \max_\pi v_\pi(s_t)$ and $q_*(s_t, a_t) = \max_\pi q_\pi(s_t, a_t)$. In case of a finite MDP, $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ are finite sets and an optimal policy $\pi_*(.)$ can be derived by solving the Bellman optimality equation either for the state-value function

$$v_*(s_t) = \max_{a_t} \mathbb{E}\{R_{t+1} + \gamma v_*(S_{t+1})|s_t, a_t\}, \tag{6}$$

or the action-value function

$$q_*(s_t, a_t) = \mathbb{E}\left\{R_{t+1} + \max_{a'} \gamma q_*(S_{t+1}, a')|s_t, a_t\right\}, \tag{7}$$

where we used the relationship $v_*(s_t) = \max_{a_t} q_*(s_t, a_t)$.

C. Approximate Methods

Finding optimal policies by solving Bellman's optimality equations requires a model of the environment in form of a discrete state transition distribution $p(s_{t+1}|s_t, a_t)$ and is thus not feasible for the vast majority of applications. Hence, reinforcement learning algorithms deal with approximate solutions and rely on a random component in order to explore the environment. Always picking the action that maximizes a value function estimate (greedy action) prevents the agent from exploring the state space. Consequently, approximate methods pick the greedy action only with a certain probability. This probability is controlled by a parameter and is increased with increasing confidence in the value function estimate.

Provided that $\mathcal{S}$ defines a discrete state space, Monte Carlo methods produce unbiased estimates of value functions by observing entire sequences of state-action-reward tuples, in order to calculate the return G_t. Although unbiasedness is a striking argument, Monte Carlo methods suffer from the usual drawbacks like e. g. high variance estimates and tremendous computational costs, since updates are solely performed after an entire episode has been observed. In contrast, temporal difference (TD) methods update the value function by bootstrapping, which leads in general to much faster convergence, but biased estimates.

So far we assumed a discrete state space $\mathcal{S}$, which essentially limits the applicability of Monte Carlo and TD methods to a specific set of applications. Moreover, we assumed that the state space is entirely observable by the learning entity. However, this is often not the case, since environmental observations are possibly continuous and only functionally related to the underlying states. Hence, the learning entity must be capable to generalize and to infer the relevant information from the observations. This is where function approximators, and thus deep neural networks, come into play.

Value-based approaches utilize neural networks to approximate either state-value or action-value functions in order to derive a well-performing policy (see e. g. [30]). However, this approach is impractical for problems dealing with a continuous action space, since picking the greedy action involves a max operation with respect to all $a \in \mathcal{A}(s)$. Consequently, this would require to solve an optimization problem at every iteration. In contrast, policy-based methods directly target the policy and are thus applicable to continuous control problems.

D. Policy Gradient Methods

Policy gradient methods aim to maximize the expected return (or a related value-based measure) by updating a parameterized policy π_θ by means of gradient ascent [31]. In case the advantage function $A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$ is used for assessing the policy's performance, the gradient of the loss for updating the policy network is given by

$$g = \mathbb{E}_{\tau \sim \pi_\theta} \left\{ \sum_{t=0}^{\infty} \nabla\theta \log \pi_\theta(a_t|s_t) A_{\pi_\theta}(s_t, a_t) \right\}, \tag{8}$$

where $\tau = (s_0, a_0, \dots, s_H, a_H, s_{H+1})$ denotes a trajectory of state and action values up to horizon H generated by sampling actions from the policy network π_θ. Since deep learning frameworks rely on stochastic gradient like optimization algorithms, we can define the policy gradient loss per training iteration as

$$L^{\mathrm{PG}}(\theta) = \hat{\mathbb{E}} \left\{ \log \pi_\theta(a_t|s_t) \hat{A}_t \right\}, \tag{9}$$

where $\hat{\mathbb{E}}$ denotes the empirical mean over a batch of samples and $\hat{A}_t$ is an estimate of the advantage function at time step t.

However, vanilla policy gradient methods use only first-order derivatives for updating the policy network, and thus, the step size plays a crucial role. If chosen too small, the agent learns too slowly in order to produce a well-performing policy, and if chosen too big, the agent might pick an action which is very far from the greedy one. Since the loss function is non-convex, there is a high risk that the update step overshoots. Hence, the agent proverbially falls off a reward cliff and the performance drops, possibly without any prospect of recovery. Another issue is the poor sample efficiency of vanilla policy gradient methods, since an entire trajectory is used to perform one single update. Consequently, a lot of interaction with the environment is required in order to train the policy network.

E. Proximal Policy Optimization

Trust region policy optimization (TRPO) [32] mitigates the shortcomings of vanilla policy gradient methods. On the one hand, trust region optimization algorithms are much more robust when dealing with non-convex problems. These methods first set an 'area of interest' before determining the direction of the consecutive optimization step. On the other hand, TRPO also increases the sample efficiency per trajectory, since samples originating from a previous policy can be used to update the policy network by means of an importance sampling scheme. Hence, the optimization (surrogate) objective for the TRPO agent is

$$\max_\theta \hat{\mathbb{E}} \left\{ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)} \hat{A}_t \right\} \tag{10}$$

subject to

$$\hat{\mathbb{E}} \left\{ D_{\mathrm{KL}} \left(\pi_{\theta_{\mathrm{old}}}(.|s_t) || \pi_\theta(.|s_t) \right) \right\} \leq \delta, \tag{11}$$

where $D_{\mathrm{KL}}(.)$ is the Kullback-Leibler divergence and δ is a hyper-parameter that defines the size of the trust region. We can convert this objective in an unconstrained optimization problem by using a penalty term instead of the constraint. Nevertheless, the need for calculating the Kullback-Leibler divergence still persists for every policy, which is computationally expensive.

Proximial Policy Optimization [33] (PPO) utilizes the key concepts of TRPO, but avoids to explicitly calculate the Kullback-Leibler divergence. The idea is surprisingly simple: Let η_t define the policy ratio used in the surrogate objective before,

$$\eta_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}. \tag{12}$$

Then the loss for each policy update is given by

$$L^{\text{PPO}}(\theta) = \hat{\mathbb{E}}\left\{\min\left(\eta_t \hat{A}_t, \text{clip}\left(\eta_t, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right\}, \tag{13}$$

where

$$\text{clip}\left(\eta_t, 1-\epsilon, 1+\epsilon\right) = \begin{cases} 1-\epsilon & \text{for } \eta_t < 1-\epsilon \\ \eta_t & \text{for } 1-\epsilon \le \eta_t \le 1+\epsilon \\ 1+\epsilon & \text{for } \eta_t > 1+\epsilon \end{cases} \tag{14}$$

and ϵ is a hyper-parameter and usually chosen to be $0.1 \le \epsilon \le 0.3$. Hence, clipping the policy ratio to values near 1 has a similar effect as using the Kullback-Leibler divergence in constraint optimization.

IV. CONSTRUCTING THE ENVIRONMENT

Our PPO RL agent is trained by immersing it to a virtual environment in which the agent can act. The environment, in this case, is the complex y-plane, that is, the complex plane of the radial integration variable. The training goal of the agent is to find a path that connects the origin with the ultraviolet cutoff on the positive real axis for any given complex value of x, while avoiding the branch cut and the poles in the plane. Before the environment can be implemented, a thorough analysis of the integrand and the resulting cut structure is in order, as the information obtained from this analysis will be used directly to guide the agent throughout its learning phase. As a first step, we will assume that the information about the branch cut, as well as about the poles, is available in an analytic form. In Section VI we furthermore discuss strategies as of how to pre-process data on the analytic properties of the integrand that is solely available numerically, in order to make the strategy developed in this paper applicable to those cases as well. For now, we will direct our focus on extracting the relevant information about the analytic properties of the regularized integrand in a scenario where all information is accessible.

A. Branch cut structure

In this Section we will analyze the integrand thoroughly, such that we can set up a training environment for the RL agent based on this information. The task of the agent will be to find valid integral contours for the y-integral in (5). The first term produces two poles, located at $y_{p_{1,2}} = \pm\frac{i}{2}$. Furthermore, a branch cut appears in the complex y-plane, induced by the angular integral over the variable z. As outlined in [9, 12], we have to solve for the zeros of the denominator while keeping x fixed, and while varying z from -1 to +1. This yields two parametrizations,

$$\xi_\pm(x,z) = \left(\sqrt{x}z \pm \sqrt{-x(1-z^2)+\frac{i}{2}}\right)^2, \tag{15}$$

and it suffices to consider just one of these two congruent parametrizations. There is, however, a problem that we have to address at this point. It turns out, that for some values of x, the parametrization of the cut becomes discontinuous in z, as explained in Figure 1.

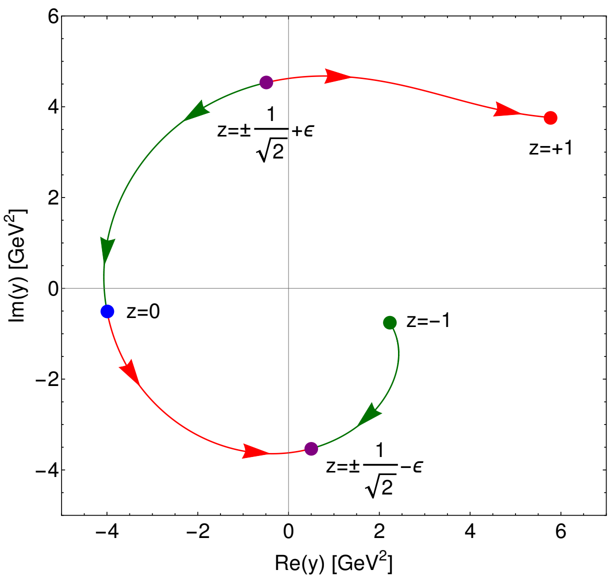


Figure 1: The parametrization of the branch cut through the angular variable z turns out to be discontinuous for some values of x. The image shows an example of such a discontinuity, here for $x = 4 + i$. For this value of x, the branch cut (15) acquires the shape shown in the image. Starting at $z = -1$, with increasing value for z, the point $-\frac{1}{\sqrt{2}} - \epsilon$ is approached. At $z = -\frac{1}{\sqrt{2}}$, the parametrization jumps to the point $-\frac{1}{\sqrt{2}} + \epsilon$, and then approaches the first point from the other side, until it finally jumps back to the second point and runs towards $z = +1$, now tracing out the entire branch cut.

In order to being able to use the information about the non-analyticities in the complex y-plane to construct the environment for the reinforcement learning agent in a convenient way, this discontinuity must be addressed. The occurrence of the discontinuity in the parametrization comes from the fact that the imaginary part of the radicand of the second square root in the branch cut parametrization (15) changes its sign (i. e. the result lies on the other Riemann sheet of the square root as the one we want it to be on). The parametrization can be rendered continuous by insertion of the proper sign in the appropriate region, that can be determined, and expressed, analytically. However, in our implementation we chose a numerical approach to resolve this issue as follows.

property	domain	description
$\|x\|$	$\mathbb{R}$	modulus of ext. momentum square
$\arg x$	$[-\pi, \pi]$	argument of ext. momentum square
Λ	$\mathbb{R}$	contour end point for agent
y_{p_1}	$\mathbb{C}$	location of first pole
y_{p_2}	$\mathbb{C}$	location of second pole
ξ_i	$\mathbb{C}$	starting point of branch cut
ξ_f	$\mathbb{C}$	endpoint of branch cut
y_A	$\mathbb{C}$	position of agent in integration plane
ω	$\{T, F\}$	agent left cut structure

Table I: Observables available for the RL agent.

One key ingredient for this environment is to determine whether a given contour, comprised of a sequence of connected line segments, intersects the branch cut at any point. We thus sample points from the (possibly for a given x, discontinuous) branch cut parametrization. Then we start at the point $z = -1$ and choose the point among the sampled values with the shortest distance to the starting point. Since the points on the branch cut are not distributed uniformly as z is varied in fixed increments, we introduced a maximally allowed distance between any two neighboring points. This is important to have control over the accuracy of the collision detection mechanism. In case we find this maximally allowed distance to be violated, we simply increase the number of points until the desired resolution is achieved. Repeating this procedure produces an ordered sequence of sampled points that we can then use in the branch cut collision detection.

B. Vectorized observations

We provide the agent with relevant information about its environment as described in Table I.

The modulus $|x|$ of the external momentum is related to the extent of the branch cut structure, and the argument $\arg x$ determines the direction of the opening of the cut structure. In addition, we provide the starting- and endpoint (ξ_i, ξ_f) of the branch cut structure, which always lie on different sides of the line defined by $\arg x$. Λ denotes the point on the positive real axis that the agent has to reach, and y_{p_i} denotes the location of two complex conjugate poles. The position of the agent in the plane is given by y_A, and we also provide a boolean value ω that indicates whether the agent has left the cut structure or not, which is determined by looking for an intersection between the agent's path and the line connecting ξ_i with ξ_f. Based on this information, the agent is able to learn a policy that produces suitable contours. Figure 2 shows a depiction of the observed quantities.

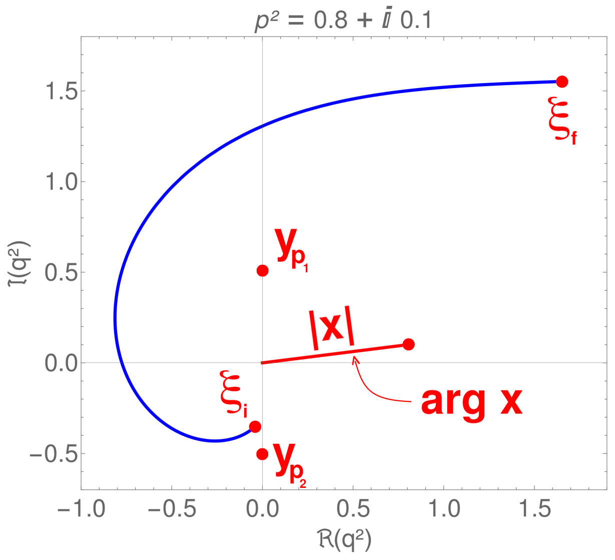


Figure 2: Properties observed by the reinforcement learning agent.

C. Actions

Standing at any point in the complex plane, the agent chooses a direction and a distance for its next step. Its action space is thus 2-dimensional and continuous. The neural network responsible for the agent's actions has two output neurons, each of which is activated by a hyperbolic tangent. Since the hyperbolic tangent assumes values within the range of $[-1, 1]$, the actions of the agent are determined as follows. The length of the next step is computed by adding 1 to the output of the associated output neuron, which maps the output to the interval $[0, 2]$. This quantity is then multiplied with $\frac{\Lambda}{2}$, where Λ has been chosen as $\Lambda = 10$ GeV^2. The agent can thus choose any value between 0 and 10 Gev^2 for its stride in the complex plane. Apart from the distance, the agent also has to choose a direction for its next step. This can be achieved by simply multiplying the output of the associated output neuron with π, which can then be interpreted as the complex argument that determines the direction of the next step.

D. Pole- and collision detection

With every step taken, we then have to check whether a collision with the branch cut occurred or not. The branch cut is approximated by line segments spanned by the ordered list of points described above. Detecting a collision thus boils down to just checking for an intersection between the line segment from the previous point to the new point the agent stepped to, with all the line segments that comprise the approximated branch cut. If a collision is detected, the environment throws the agent back to where it was before it chose to take that step, and it can come up with a new suggestion.

The ultraviolet cutoff of the radial integral that has been used for the calculations in [12] was $\Lambda_{UV} = 10^4$ GeV^2. A successful candidate for a contour thus starts at the origin $y = 0$ and ends at $y = 10^4$. However, in the study at hand we are only interested in evaluating the correlator in a complex region around the origin that is at least three orders of magnitudes smaller than this cutoff. As the typical scale of the non-analyticities arising in the integration plane is set by $|x|$, and because this value is at most of the order 1, the integrand is analytic along the real axis for values of $x \approx 10$ GeV^2 or greater. We thus set the point the agent has to reach to $y = 10$ GeV^2. The rest of the contour can then be added as the line connecting this point with the actual ultraviolet cutoff of the integral.

There is also another reason why this particular setup is preferable. If this approach is applied to non-perturbative settings – such as Dyson-Schwinger equations – in the future, one will have to restrict the area in the complex plane for which the equation is to be evaluated to a small region around the origin. Since it lies within the nature of DSEs that the analytic properties of the integrand are not fully accessible, one will have to introduce a boundary, along which one closes the contour to a point on the positive real half axis not too far off from the origin. For this to be justifiable, one would have to look for evidence that suggests that the errors induced by this restriction have little impact on the result. For example, it is imaginable that the discontinuities of the branch cuts become smaller with increasing $|x|$, which would justify this procedure and allow for a direct mapping from a DSE setting to the environment at hand.

In our model, we constrain the agent to the square $x \in [-1.5\Lambda, 1.5\Lambda]$ GeV^2 $\times\, i[-1.5\Lambda, 1.5\Lambda]$ GeV^2, and we treat any step that takes the agent out of this region similar to it trying to cross a branch cut. Since the analytic properties of the toy model are accessible exactly, we would not have to impose this restriction. However, it is beneficial in at least two ways. First, we will have to impose such a restriction in a future setting for DSEs, so we will be able to directly map the DSE setup on the one developed here, as discussed in Section VI below. Second, it also helps the agent to stick around the origin and thus close to the endpoint, which helps it to find its target more quickly.

Once the agent successfully enters a small region around the endpoint $y = 10$ GeV^2, we have to check whether the path suggested by the agent is continuously deformable into the original path along the positive real half-axis if we neglect the presence of the branch cut, but maintain the poles. In order to determine whether this is the case, we (numerically) integrate a known function that has poles with residue 1 at the same position as the integrand, but no branch cut, along the suggested contour. If we then add the value of the integral along the real axis from $y = 10$ GeV^2 to $y = 0$ GeV^2, evaluated on the same known function, we can deduce whether the path is continuously deformable in the sense discussed above by exploiting Cauchy's integral formula.

E. Reward function

The reward function is a crucial quantity, as it allows us to directly encourage or discourage the agent to take certain actions in certain situations. We punish the agent for failing to produce a valid contour and for bumping into the branch cut (or the boundary) by providing it with a negative reward. We furthermore associate a small punishment for every step taken, which encourages the agent to take as few steps as possible to reach the goal. The agent receives a positive reward if it manages to reach the goal along a valid contour. We furthermore introduce a small reward or punishment depending on whether the last step took the agent closer to, or farther from, the desired endpoint. This helps the agent to figure out where it should go. There is an inherent asymmetry between the two constraints the agent has to consider as it finds its path towards the endpoint. While a collision of the path with the branch cut can be detected, and punished, immediately, the information of whether the contour is continuously deformable into the original contour in the absence of the branch cut can only be provided once the agent has reached its goal. It thus receives the respective reward or punishment only at the very end of an episode, which makes it much harder for the agent to figure out which of the steps that it took was responsible for the reward or punishment it received in the end. This is the aforementioned problem of sparse rewards and poses a challenge that has to be addressed.

V. NUMERICAL EXPERIMENTS AND RESULTS

In Section IV E above we discussed the issue of the pole detection being particularly difficult to learn for an agent, as the reward or punishment is only provided at the end of the episode. We thus decided to study two separate cases. First, we investigated scenarios that featured branch cuts only, that is, we neglected the presence of the poles. This scenario is easier for an agent, as it is provided with rewards and punishments immediately, that is, after every single step. Its actions thus have very immediate consequences, and it is easy for the agent to derive optimal actions. In a second scenario, we consider the full problem, that is, besides the branch cut we also introduce the poles. This has a severe effect on the agents performance, as the reward signal is now delayed. As a possible simplification of the active pole detection scenario we also introduced a mechanism that allows the environment to automatically close the contour when the agent has successfully exited the branch cut structure. We refer to this mechanism as path auto-completion. The training of the agents has been con-

property	values	description
State space dim.	14	number of parameters in state space
Action space dim.	2	dimension for step length and angle
First pole	$0 + 0.5i$	position first pole
Second pole	$0 - 0.5i$	position second pole
$\|x\|$	$\mathbb{R}$	modulus of ext. momentum square
$\arg x$	$[-\pi, \pi]$	argument of ext. momentum square
Λ	10	contour end point for agent
ϵ-region	0.5	radius around goal
γ	0.99	discount factor
eps_clip	0.2	clip parameter for PPO
batch_size	80	batch size
update_steps	6000	time steps between policy updates
learning_rate	5×10^{-5}	learning rate for optimizer
act_std	0.4	standard deviation for action distribution (multivariate normal)
rew_fail	-50000	reward for failure
rew_coll	-10	reward for collision with contour
rew_iter	-5	reward for each step
rew_attr	$\{.01,\ .0055,\ .001\}$	reward for goal attraction
rew_won	1000	reward for winning
max episodes	5000	max number of episodes for a single policy training
max steps	1500	max number of steps per episode

Table II: Parameters of the three numerical experiments without pole detection as described in Section V A. The same parameter sets have also been used in the pole detection runs with auto-completion, see Section V C. The only difference between the three runs is in the parameter `rew_attr`, which controls the reward or punishment received if the agent steps closer to, or away from, the desired endpoint.

ducted as follows. For each episode, we pick a value for x at random, by uniformly sampling from the plane $x \in [-5, 5]$ GeV^2 $\times$ $i[-5, 5]$ GeV^2. The position of the poles, if present, is fixed at $\pm\frac{i}{2}$ GeV^2.

The policy network (actor) and value estimator network (critic) were chosen identically. They consist of two hidden layers, the first with 64 neurons and the second with 32 units. The activation function for each layer is the hyperbolic tangent, and as optimizer we used Adam [34]. An overview of the other parameters is summarized in Table II.

All numerical experiments were conducted on Nvidia Titan RTX Graphics Processing Units (GPUs). On three GPUs, we could conduct 48 experiments in parallel. The program code has been written in Python, using PyTorch [35] with GPU support. The CPU on which large parts of the calculations for the rewards were calculated is an Intel® Xeon® W-2145 processor. For a training with pole detection turned on (we refere here to our best experiment) we needed an average of 8.94212 seconds to calculate one episode and 0.01066 seconds to calculate one action.

A. Experiments without pole detection

We conducted our first numerical experiments with the simplified environment, that is, we ignored the presence of the poles. We considered three configurations that differ in the choice for the parameter `rew_attr`. This parameter is responsible for controlling the reward (or punishment) the agent receives as it steps closer to, or farther from, the desired endpoint. The three parameter values were $\{.01,\ .0055,\ .001\}$. All other parameters were identical to the values shown in Table II.

The outcome of all three experiments lie above a 90 percent success rate. For the experiment with an attractor reward of 0.01 we get a success rate of 92.94%, with an attractor reward of 0.0055 we get 94.56%, and with an attractor reward of 0.001 we get a success rate of 95.58%. Figure 3 illustrates two non-trivial scenarios where the agent (in this case the agent with a final success rate of 95.58%) was able to reach a point within a disc with radius ϵ-`region`, centered around the end-point. Figure 4 shows the learning curve of the best agent trained in this environment.

B. Experiments with pole detection

To investigate the performance of a reinforcement learning agent on the full problem, that is, with poles included, we conducted a total of 720 numerical experiments with different parameters. The parameters that we varied were the following. We chose different values for the standard deviation of the noise distribution used for the actions (`act_std`), for the number of steps in between the updates of the policy (`update_steps`), for the learning rate used by the optimizer (`learning_rate`), the batch size (`batch_size`), and for the parameter `rew_attr` that was also varied throughout the experiments without the poles being present. The overall 720 numerical experiments are all combinations of:

- `act_std` $\in \{.1,\ .2,\ .3,\ .4\}$
- `update_steps` $\in \{250,\ 562,\ 875,\ 1187,\ 1500\}$
- `learning_rate` $\in \{10^{-5},\ 10^{-4},\ 10^{-3}\}$
- `batch_size` $\in \{60, 70, 80, 90\}$
- `rew_attr` $\in \{.01,\ .0055,\ .001\}$

The best experiment with activated pole detection achieved a success rate of 27.82%. The parameters associated with the best performing agent were:

- `act_std`: .4
- `update_steps`: 1187
- `learning_rate`: 10^{-5}
- `batch_size`: 90
- `rew_attr`: .0055

This is significantly smaller than the success rates achieved by the agents trained in the environment without pole detection. Figure 5 shows two situations, one with positive and one with negative outcome. In contrast to the simplified experiments discussed in the previous section, the agent cannot always count reaching the goal within a given tolerance as success in this case. Positive experiences are much less frequent, which slows down the learning process. The learning curve of the best agent trained in this scenario is shown in Figure 6

C. Experiments with pole detection and auto-completion

In order to make the reward, or punishment, for enclosing a pole more immediate to the agent, we decided to alter the environment as follows. Rather than allowing the agent to move freely within the plane $x \in [-1.5\Lambda, 1.5\Lambda]$ GeV^2 $\times$ $i[-1.5\Lambda, 1.5\Lambda]$ GeV^2, we first drew a bounding box around the branch cut. The size of the box was chosen such that it confines the cut as close as possible, without actually intersecting it. One can then make the box slightly larger such that its edges can be used as part of the integration contour. This slight modification is necessary, as the numerical integration becomes increasingly expensive with decreasing distance to the branch cut. The agent was then trained to produce a contour that connects the origin with the box that encloses the branch cut. Once such a contour exists, where the opening of the branch cut has to be respected, that is, no intersections with the cut are allowed, the auto-completion algorithm takes over. The contour is then closed automatically by following along the edges of the box and then parallel to the real axis, until the real part of the contour is equal to Λ, after which the contour is closed by a vertical line segment. Note that the auto-completion algorithm also checks for the location of the poles. Since the potential pickup of a pole's residue depends upon whether the auto-completed contour is closed by following the edges to the left, or by following the edges to the right of the intersection point, or cannot be avoided regardless of how the contour is closed, the algorithm proceeds as follows. If the agent reaches the edge of the bounding box in the second or third quadrant, we test both ways of closing the contour and accept the one that doesn't pick up the residue, if it exists. This setup improves upon the delayed reward problem, as the auto-completion is – apart from the direction of the closure – beyond the agent's control, whose responsibility ends with reaching one of the edges of the bounding box. This makes the impact of its choices much more immediate. Consequently, we found a much better performance of the agent working on the full problem. We conducted three experiments, using the same parameters as the ones used for the simplified environment shown in Table II. The best success rate was around 47%, produced by the agent using a `rew_attr` of 0.001. Note furthermore, that this approach can also be adopted for the Dyson-Schwinger equation setting. Figure 7 shows two contours suggested throughout training by an agent learning in this environment.

VI. SUMMARY AND OUTLOOK

In this paper we introduced a novel technique based on deep reinforcement learning that allows for numerical analytic continuation of loop integrals in Euclidean space in quantum field theory. The main challenge to solve such integrals in the complex domain is to find suitable contours that avoid branch cuts and poles in the integration plane. By referring to an example where contour deformation has been successfully applied to solve the problem in the past, we trained reinforcement learning agents to produce valid contours for the same example, providing a proof of principle that machine learning can indeed be

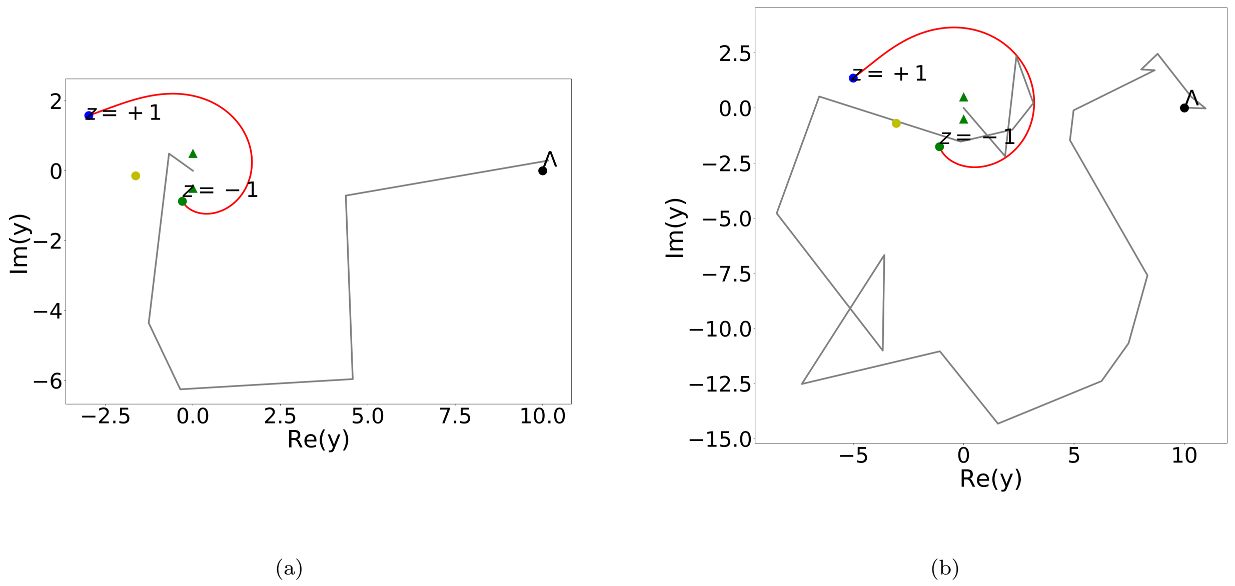


Figure 3: Two solutions provided by the same agent throughout training. Note that we do not apply any smoothing mechanisms, nor do we discourage or remove loops in the paths produced by the agent. Each contour starts at the origin and traces out a path that leads to the cutoff labeled Λ. The (green) triangles mark the position of the poles, which were ignored in this particular run. The (yellow) circles between the cut's endpoints indicate the value of x used for this particular run. The solutions presented here were produced by the best performing agent among the three trained in this setting. It has a success rate of almost 96%. Note that both contours solve the problem. Figure (a) shows a valid solution produced by the agent after training for 1783 episodes. Figure (b) shows a valid solution produced by the agent after training for 4726 episodes. This path could be post-processed to remove the loops, which we haven't implemented.

of assistance in solving such tedious problems.

While the best agent trained in the absence of poles reaches success rates that justify deployment in such scenarios, agents trained on settings with poles being present achieved success rates that produce valid contours only for every other value of x. Since we did not directly address the issue of delayed rewards in this first study, we are confident that this success rate can be increased sufficiently to allow for deployment of this approach in the presence of poles as well.

In a follow-up study we will thus focus on two main tasks. First, we will improve upon the agent's performance in the presence of poles, which is a scenario in which we have to deal with sparse rewards. For studies addressing this issue see e. g. [36–38]. We will explore the applicability of the strategies presented in these studies to our problem (note that, e. g. [37] requires an off-policy approach). We will also investigate how an ensemble of trained agents performs at solving the problem of producing valid contours in the presence of poles. Since we can validate each contour by means of the auxiliary function used throughout training, we could then just ask every agent in the ensemble to produce a contour and accept one that is valid. If, on the other hand, the ensemble is incapable of producing a valid contour for a given set of parameters, this will give us insights as to why the agents fail to solve that particular situation, such that we can use domain knowledge to provide guidance. In addition to looking into approaches that try to improve upon the sparse reward problem, as well as investigating ensembles of agents, we furthermore will also explore the potential of runtime enforcement (shielding) [39] for our application.

Another important question is, how the environment can be generalized in order to being able to deal with integral equations of the Fredholm 2 type, such as Dyson-Schwinger equations. In order to address this, we plan to take numerical input of the complex integration plane before an iteration step is conducted. Here are two possibilities as of how the perception of the non-analyticities is approached. One could use conventional image detection algorithms in combination with the theory of complex functions to detect poles and branch cuts (see e. g. [22] for such an approach to detect poles) in the integration plane. Instead of conventional image processing one could also work with convolutional neural networks on a 'pixel' basis, where the pixels are in fact the moduli of the numerically computed complex values of the integrand.

Once the information about the obstructions in the integration plane is available, it can be fed into the vectorized environment presented here. We can then sample the integration plane for various values of x and train the

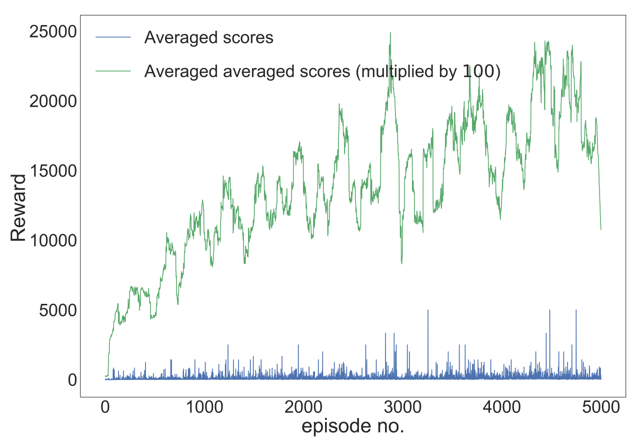


Figure 4: This plot illustrates the development of averaged rewards throughout training of the best reference experiment, where the pole detection was deactivated. The blue curve represents the averaged return for each episode, while the green curve is the average over 100 consecutive averaged returns, rescaled for readability. The trend of improvement of the performance over training time is clearly visible.

agent in a similar fashion. Thus, every iteration step corresponds to the exact same problem as the one treated in this paper. Consequently, the agent has to be re-trained for every iteration step. One can, however, start with the policy the agent used for the previous iteration step, and if the shift in the analytic properties in the integration plane is not too severe, the agent should be able to adapt easily. Another aspect that should help in this regard is, that we formulated the environment in a very general way. For example, it is not expected that branch cuts arising in such integrals open in directions other than $\arg x$, but they will most likely just differ by its shape. The difference in shape will force the agent to focus more on the direction of the opening until it leaves the structure.

But even assuming that both strategies, the one directly addressing the sparse reward problem and the ensemble approach, fail to produce a valid answer for a certain point of x in a Dyson-Schwinger equation setting, the consequences are not expected to be severe, since the system is solved iteratively, so even if an iteration step fails, the system could still converge, given that the overall success rate is not too low.

With a success rate of 96% without poles, and close to 50% with poles being present (and without applying any of the delayed reward signal strategies mentioned above), our study shows that machine learning in general, and reinforcement learning in particular, can help in tackling such demanding numerical problems as contour deformations to compute one-loop integrals in quantum field theory.

VII. ACKNOWLEDGMENTS

This work has been supported by the Silicon Austria Labs (SAL) funded by the Austrian federal government and the federal states of Carinthia, Styria and Upper Austria and Austrian Association for the Electric and Electronics Industry. Furthermore, AW acknowledges that parts of this work have been supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under Award Number #DE-FG-02-05ER41375, as well as by the Austrian Science Fund (FWF), Schrödinger Fellowship J3800-N27.

Appendix A: Conventions

All calculations are performed in Euclidean space.

1. Hyperspherical coordinates

Using hyperspherical coordinates, the 4-momentum q can be expressed as

$$\begin{aligned}
&\int_{\mathbb{R}^4} d^4q \rightarrow \qquad\qquad\qquad\qquad\qquad\qquad \text{(A1)}\\
&\int_0^{2\pi} d\phi \int_0^{\infty} dq\; q^3 \int_0^{\pi} d\theta_1 \sin^2\theta_1 \int_0^{\pi} d\theta_2 \sin\theta_2\\
&= \left| \begin{array}{c} y \equiv q^2 \rightarrow dy = 2qdq \\ \theta_1 \equiv \arccos z \rightarrow d\theta_1 = -\frac{dz}{\sqrt{1-z^2}} \\ \theta_2 \equiv \arccos w \rightarrow d\theta_2 = -\frac{dw}{\sqrt{1-w^2}} \end{array} \right|\\
&= \frac{1}{2}\int_0^{2\pi} d\phi \int_0^{\infty} dy\; y \int_{-1}^{1} dz \sqrt{1-z^2} \int_{-1}^{1} dw.
\end{aligned}$$

With two momenta involved, only the radial and one angular integral remains. With an IR cutoff ε and an UV cutoff Λ_{UV}, we get

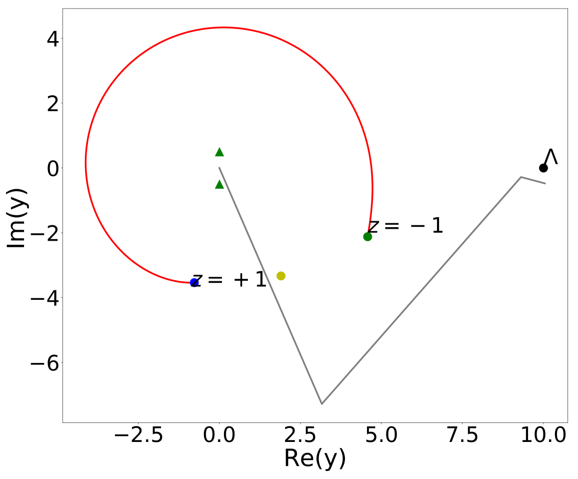
$$\begin{aligned}
&\int_{\mathbb{R}^4} \frac{d^4q}{(2\pi)^4} \rightarrow \qquad\qquad\qquad \text{(A2)}\\
&\frac{1}{(2\pi)^3}\int_{\varepsilon}^{\Lambda_{UV}} dy\; y \int_{-1}^{1} dz \sqrt{1-z^2}.
\end{aligned}$$

Defining x to be the square of the external momentum p^2,

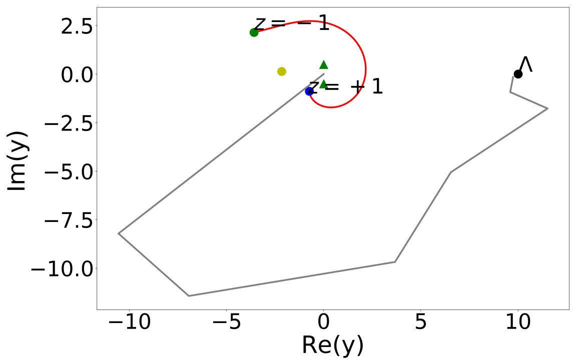
$$x := p^2, \tag{A3}$$

the scalar product between p and q becomes.

$$p.q = \sqrt{x}\sqrt{y}z. \tag{A4}$$



(a)



(b)

Figure 5: Two solutions suggested by the same agent throughout training on the full problem (poles included). The (green) triangles mark the position of the poles. The solutions presented here were produced at a late time throughout training by the best performing agent among the 720 trained in this setting. It has a success rate of almost 28%, which is considerably lower than the one achieved without pole detection. Figure (a) shows a valid solution produced by the agent after training for 4996 episodes. Figure (b) shows an invalid solution produced by the agent after training for 4674 episodes. This contour must be rejected, because the path is not continuously deformable into the path along the positive real half-axis in absence of the branch cut.

[1] R. Alkofer and L. von Smekal, *Phys. Rept.* **353** (2001) 281, `arXiv:hep-ph/0007355 [hep-ph]`.

[2] C. S. Fischer, *J. Phys.* **G32** (2006) R253–R291, `arXiv:hep-ph/0605173 [hep-ph]`.

[3] C. D. Roberts, M. S. Bhagwat, A. Holl, and S. V. Wright, *Eur. Phys. J. ST* **140** (2007) 53–116, `arXiv:0802.0217 [nucl-th]`.

[4] H. Sanchis-Alepuz and R. Williams, *J. Phys. Conf. Ser.* **631** (2015) no. 1, 012064, `arXiv:1503.05896 [hep-ph]`.

[5] I. C. Cloet and C. D. Roberts, *Prog. Part. Nucl. Phys.* **77** (2014) 1–69, `arXiv:1310.2651 [nucl-th]`.

[6] G. Eichmann, H. Sanchis-Alepuz, R. Williams, R. Alkofer, and C. S. Fischer, *Prog. Part. Nucl. Phys.* **91** (2016) 1–100, `arXiv:1606.09602 [hep-ph]`.

[7] H. Sanchis-Alepuz and R. Williams, *Comput. Phys. Commun.* **232** (2018) 1–21, `arXiv:1710.04903 [hep-ph]`.

[8] G. Eichmann, "Towards resonance properties in the Dyson-Schwinger approach," 2019. `arXiv:1912.08873 [hep-ph]`.

[9] A. Windisch, M. Q. Huber, and R. Alkofer, *Acta Phys. Polon. Supp.* **6** (2013) no. 3, 887–892, `arXiv:1304.3642 [hep-ph]`.

[10] P. Maris, *Phys. Rev.* **D52** (1995) 6087–6097, `arXiv:hep-ph/9508323 [hep-ph]`.

[11] S. Strauss, C. S. Fischer, and C. Kellermann, *Phys. Rev. Lett.* **109** (2012) 252001, `arXiv:1208.6239 [hep-ph]`.

[12] A. Windisch, R. Alkofer, G. Haase, and M. Liebmann, *Comput. Phys. Commun.* **184** (2013) 109–116, `arXiv:1205.0752 [hep-ph]`.

[13] A. Windisch, M. Q. Huber, and R. Alkofer, *Phys. Rev.* **D87** (2013) no. 6, 065005, `arXiv:1212.2175 [hep-ph]`.

[14] E. Weil, G. Eichmann, C. S. Fischer, and R. Williams, *Phys. Rev.* **D96** (2017) no. 1, 014021, `arXiv:1704.06046 [hep-ph]`.

[15] J. M. Pawlowski, N. Strodthoff, and N. Wink, *Phys. Rev.* **D98** (2018) no. 7, 074008, `arXiv:1711.07444 [hep-th]`.

[16] R. Williams, *Phys. Lett.* **B798** (2019) 134943, `arXiv:1804.11161 [hep-ph]`.

[17] G. Eichmann, P. Duarte, M. T. Peña, and A. Stadler, *Phys. Rev.* **D100** (2019) no. 9, 094001, `arXiv:1907.05402 [hep-ph]`.

[18] A. S. Miramontes and H. Sanchis-Alepuz, *Eur. Phys. J.* **A55** (2019) no. 10, 170, `arXiv:1906.06227 [hep-ph]`.

[19] R. Alkofer, W. Detmold, C. S. Fischer, and P. Maris, *Phys. Rev.* **D70** (2004) 014014, `arXiv:hep-ph/0309077 [hep-ph]`.

[20] S. M. Dorkin, L. P. Kaptari, T. Hilger, and B. Kampfer, *Phys. Rev.* **C89** (2014) 034005, `arXiv:1312.2721 [hep-ph]`.

[21] S. M. Dorkin, L. P. Kaptari, and B. Kämpfer, *Phys. Rev.* **C91** (2015) no. 5, 055201, `arXiv:1412.3345 [hep-ph]`.

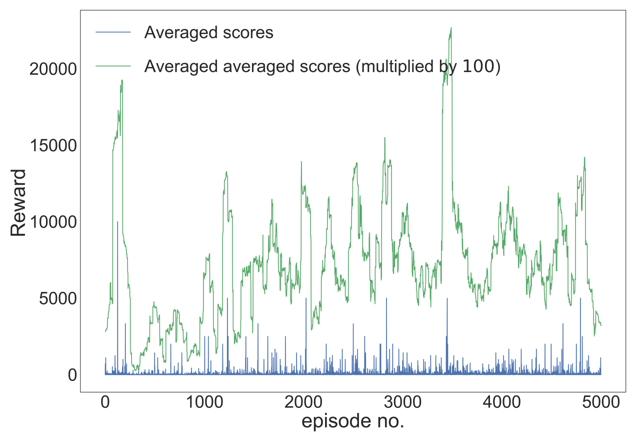


Figure 6: This plot illustrates the development of averaged rewards throughout training of the best experiment with pole detection. The blue curve represents the averaged return for each episode, while the green curve is the average over 100 consecutive averaged returns, rescaled for readability.

[22] A. Windisch, *Phys. Rev.* **C95** (2017) no. 4, 045204, `arXiv:1612.06002 [hep-ph]`.

[23] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, *CoRR* **abs/1811.12560** (2018) , `arXiv:1811.12560`. `http://arxiv.org/abs/1811.12560`.

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[25] L. Baulieu, D. Dudal, M. S. Guimaraes, M. Q. Huber, S. P. Sorella, N. Vandersickel, and D. Zwanziger, *Phys. Rev.* **D82** (2010) 025021, `arXiv:0912.5153 [hep-th]`.

[26] N. N. Bogoliubov and O. S. Parasiuk, *Acta Math.* **97** (1957) 227–266.

[27] N. N. Bogolyubov and D. V. Shirkov, *Intersci. Monogr. Phys. Astron.* **3** (1959) 1–720.

[28] K. Hepp, *Commun. Math. Phys.* **2** (1966) 301–326.

[29] W. Zimmermann, *Commun. Math. Phys.* **15** (1969) 208–234. [Lect. Notes Phys.558,217(2000)].

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, *Nature* **518** (2015) 529–33.

[31] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pp. 1057–1063. MIT Press, Cambridge, MA, USA, 1999. `http://dl.acm.org/citation.cfm?id=3009657.3009806`.

[32] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, eds., vol. 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897. PMLR, Lille, France, 07–09 Jul, 2015. `1502.05477`. `http://proceedings.mlr.press/v37/schulman15.html`.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *CoRR* **abs/1707.06347** (2017) , `arXiv:1707.06347`. `http://arxiv.org/abs/1707.06347`.

[34] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2014. `arXiv:1412.6980 [cs]`.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

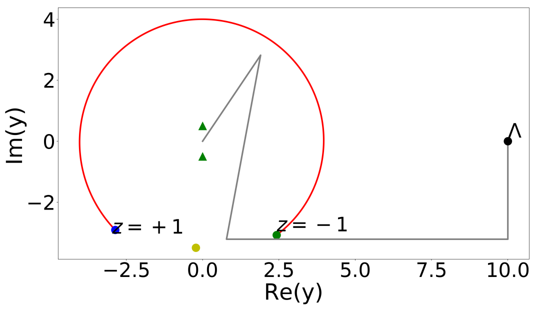
[36] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, *CoRR* **abs/1611.05397** (2016) , `arXiv:1611.05397`. `http://arxiv.org/abs/1611.05397`.

[37] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, *CoRR* **abs/1707.01495** (2017) , `arXiv:1707.01495`. `http://arxiv.org/abs/1707.01495`.

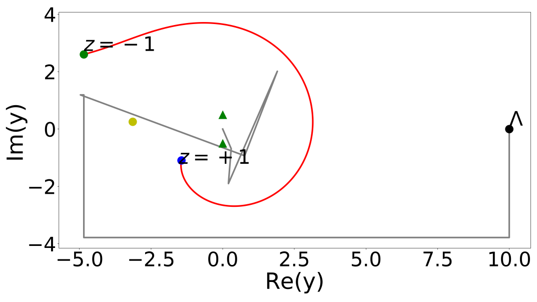
[38] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, eds., vol. 70 of *Proceedings of Machine Learning Research*, pp. 2778–2787. PMLR, International Convention Centre, Sydney, Australia, 06–11 Aug, 2017. `http://proceedings.mlr.press/v70/pathak17a.html`.

[39] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 2669–2678. 2018.

[40] In our notation we use lowercase characters for the realization of the the corresponding random variable, which we denote in upper case



(a)



(b)

Figure 7: Two solutions suggested by the same agent throughout training on the full problem (poles included) with auto-completion. The (green) triangles mark the position of the poles. The solutions presented here were produced by the best agent throughout training. It has a success rate of almost 48%, which is a dramatic increase in performance as compared to the best agent trained with naive pole detection. This indicates that this issue is due to a delayed reward signal. Figure (a) shows a valid solution produced by the agent after training for 126 episodes. Figure (b) shows a valid solution produced by the agent after training for 1820 episodes.