REINFORCEMENT LEARNING ALGORITHMS FOR CONTROLLING QUANTUM SPIN-1/2 NETWORK

FINAL YEAR PROJECT

Anastasia Ugaste
Department of Computer Science
Cardiff University
Cardiff, CF24 3AA
`Ugaste.A@cardiff.ac.uk`

Supervisor: Dr F.C.Langbein

May 26, 2021

1 Introduction

Quantum control offers methods to steer the dynamics of quantum systems; this is particularly useful for building devices for quantum computing, simulation and networking.

This is done using optimisation algorithms such as L-BFGS (which approximates the second derivative in the Newton algorithm that working with a vector rather than a matrix) and standard optimisation algorithms for control like GRAPE or KROTOV [3].

While a large number of algorithms for optimizing quantum dynamics for different objectives have been developed, a common limitation is a reliance on good initial guesses, being either random or based on heuristics and intuitions [1]. Therefore, we are having a situation where L-BFGS is used in an ideal setting to find the best controller. This makes L-BFGS perform extraordinary in the conditions given, however in reality two factors are affecting the performance.

First of all, when observing the system from the outside we have limited knowledge of the structure inside the system and its interactions within it – making it is not possible to calculate Hamiltonians 100% correct. The second issue is there is always decoherence in this system – therefore the values received from controllers are never realistic. All of this leads to the conclusion that we need to find a more robust way of exploring the quantum spin-½ network landscape and finding the control parameters.

Reinforcement learning algorithms can be a new method to ensure the robustness of the controllers found without them being largely affected by distortion. The philosophy behind using unsupervised model-free machine learning algorithms (such as reinforce and deep-q learning) is that after initial exploration of the landscape and learning of the unknown system model by probing it for data, the algorithm will exploit that information to control it [2]. Those machine learning algorithms then will be tested in both noise and noise-free environments and compared with the L-BFGS approach's results that are commonly used. This will help me to conclude how robust the controls found by reinforcement learning algorithms tested are, if reinforcement learning algorithms tested during this project can give us more robust controls under uncertainties compared to the gradient-based methods and bring a benefit to the currently used methods in quantum optimization problems. This will be demonstrated by showing how much the fidelity of the controls varies under uncertain parameters in the Hamiltonian and dephasing in simulation.

In this report, the problem and its background will be introduced first. Then the methodology of the experiments is going to be explained. In the evaluation section, we will be talking about the results obtained using the methods and parameters discussed in the methodology before. Next, the conclusion will be drawn based on the results, and future work ideas will be suggested. To conclude the report, there will be a personal reflection on this project work conducted in the semester.

2 Background

2.1 State of Art

The general goal of quantum control is to actively manipulate dynamical processes at the atomic or molecular scale, typically using external electromagnetic fields or forces [3].

$$\imath\hbar\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = H_o\left|\psi(t)\right\rangle, \qquad \left|\psi(t=0)\right\rangle = \left|\psi_o\right\rangle \tag{1}$$

Eq. (1) shows a generic control model where H_o is a system Hamiltonian, $|\Psi(t)\rangle$ is a quantum wave function, t is time; $\hbar$ is the reduced Planck constant.

The total Hamiltonian (Eq. (2)), where H_k - control Hamiltonian, u_k - vector of control variables

$$H(t) = H_o + \Sigma_k u_k(t) H_k \tag{2}$$

then determines the controlled evolution

$$\imath\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = [H_o + \sum_k u_k(t)H_k]\left|\psi(t)\right\rangle \tag{3}$$

The goal in a typical quantum control problem defined on the system and described by the equation above is to find a final time t > 0 and a set of admissible controls

$$u_k(t) \in R \tag{4}$$

which drives the system from the initial state into a predefined target state [4].

2.2 Control Techniques Available

2.2.1 Numerical controls for quantum spin networks

The numerical control algorithms include gradient ascent algorithms and Krotov-type methods – which both allow to extend to second-order quasi-Newton and Newton methods if needed. The main difference between these two approaches is that the control is updated (or replaced) for all times simultaneously in the case of gradient ascent algorithms and sequentially in the case of Krotov algorithms. The algorithms are comparatively easy to use and several program packages include optimal control modules with necessary modifications to account for experimental imperfections and limitations and to ensure the robustness of the solution [2].

2.2.2 Controls via adiabatic dynamics

The well-known method among adiabatic dynamics controls is Stimulated Raman Adiabatic Passage (STIRAP). Adiabatic techniques usually employ a sequence of very intense pulses over a comparatively long-timescale which enforces adiabatic following of the system dynamics. The pulses can be frequently chirped according to the structure of the energy levels. Such processes are inherently robust to small variations of laser or system parameters and thus well-suited to open-loop control. However, these methods have a significant drawback in terms of the total time and energy required for the performance - which cannot always be met in an experimental setting [2].

2.2.3 Alternative approaches that are being investigated.

Dynamic control involves dynamically altering certain couplings. However, it typically requires the ability to rapidly modulate or switch fields. Furthermore, for networks with a high degree of symmetry such as rings with XX uniform coupling controllability is generally limited by dynamic symmetries. An alternative to this dynamic control is to shape the potential landscape to facilitate the flow of the information from an initial state (input node) to the target state (output node) [5]. However, this comes at a cost of fixing the time for the particular problem, as well as fully defining a quantum landscape problem given – which makes a dynamic problem to be described by a static set of equations. Fixing the time makes it possible to reduce the control parameters and make optimization simpler, however makes the process of finding the precise controls harder (as in reality the system is still dynamic) and makes controls not as robust as desired in practice.

2.3 Reinforcement learning algorithms

Reinforcement learning (RL) is the task of learning how agents ought to take sequences of actions in an environment to maximize cumulative rewards [6]. The reasoning behind using the reinforcement learning approach in spin-½ problems is the possibility of model-free unsupervised learning ability of machine learning – thus making the solutions valid for any number of qubits as it finds the patterns within the landscapes given and tries to define the formulas used for optimization itself.

2.3.1 REINFORCE

Reinforce is an example of a Policy-Gradient method that estimates an optimal policy's weights through gradient ascent [7]. The name is an acronym for "Reward Increment = Non -negative Factor times Offset Reinforcement times Characteristic Eligibility" that described the algorithm. The first thing we need to define is a trajectory, just a state-action-rewards sequence (but we ignore the reward). A trajectory is a little bit more flexible than an episode because there are no restrictions on its length; it can correspond to a full episode or just a part of an episode. When maximizing expected return over trajectories (instead of episodes), it lets the method search for optimal policies for both episodic and continuing tasks. The sum of returns of a trajectory is the cumulative reward. The return of the trajectory is used to estimate the gradients and update the weights of the policy found [8].

2.3.2 Deep Q learning

In deep Q learning, we approximate the optimal action-value function by deriving the policy from a Q function used to evaluate state-action pairs and carry out policy evaluation via TD methods to obtain the next Q function [9]. Learning the optimal Q-function is accomplished by minimizing the squared Bellman loss (error). DQN training typically involves repeated interactions with a simulator, or the use of historical data to constantly evaluate and adjust its Q functions and thus find the correct weights.

2.4 The problem investigated

For this project, we will assume to be working with a 5 qubit spin-½ ring network and compare an L-BFGS approach that uses the gradient ascend method to find the controls with several machine learning algorithms such as reinforce and deep-Q learning. The qubits in the network are Hamiltonians following the Heisenberg model which is described by Eq. (5) where J_{mn} - the coupling between spin m, n; X_m, Y_m, Z_m - the Pauli operators lifted to the full Hilbert space; Δ_n is the local energy landscape bias;

$$H_{full} = \sum_{n=1}^{N} \Delta_n Z_n + \sum_{m \neq n} J_{mn}[X_m X_n + Y_m Y_n + \kappa Z_m Z_n] \tag{5}$$

More precisely, we will be working with the reduced to single excitation subspace Hamiltonians (see the equation below), where |m> <n| can be thought of as a matrix which is zero except for a 1 in the (m.n) position.

$$H_{\Delta} = \sum_{n=1}^{N} \Delta_n \left|n\right\rangle \left\langle n\right| + \sum_{m \neq n} J_{mn} \left|m\right\rangle \left\langle n\right| \tag{6}$$

The probability of the signal transmission from the given input node |in> to the output node |out> in time t is given by Eq. (7)

$$p(t) = |\left\langle out\right| e^{-itH_{\Delta}} \left|in\right\rangle|^2 \tag{7}$$

Assuming the energies are controllable, we can find the control parameters so they satisfy Eq. (8)

$$p(t) = \max_{\Delta} |\left\langle out\right| e^{-itH_{\Delta}} \left|in\right\rangle|^2 \tag{8}$$

at given time t. By finding the correct diagonal of the matrix, we therefore can accomplish the information transfer with maximum fidelity in minimum time [5].

3 Methodology & Implementation

In this section the methods used in order to conduct the experiments and compare the results will be discussed. Whilst not repeating the information that can be found in the background, we will go in greater depth on the implementation part of the algorithms so they can be repeated by anyone to compare with the results later on in this paper

3.1 L-BFGS basic approach

I was provided with L-BFGS algorithm code that uses quasi-newton method to solve the problem by using second derivative approximations which was developed for XX spin-½ rings. The Hamiltonians in the program are generated using Sobel sequencing input therefore insuring randomization and symmetry of the parameters. The code asks for inputs in terms of range of bias and time we want the solution to be found and uses the Newton method to calculate the propagator and predict infidelity and gradient for all i/o maps. The code is a python version of Matspinnet that can be found online [10]. Even though I have not modified this code at all, this code was used for comparison with the reinforcement learning method as well as proven useful when verifying the results returned by neural networks.

3.2 Machine learning general approach

Machine learning helps us to solve the Markov reward and decision processes (i.e., finding the optimal police and value functions) by using bellman equation. The Bellman equation (where v_s is a value of a state, R - a reward, γ - discount factor, P_v - transitional probability) can be expressed concisely using matrices (Eq. (9,10))

$$v_s = R + \gamma P v \tag{9}$$

where v is a column vector with one entry per state.

$$\begin{bmatrix} v(1) \\ ... \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ ... \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & ... & P_{1n} \\ ... & & \\ P_{11} & ... & P_{nm} \end{bmatrix} \begin{bmatrix} v(1) \\ ... \\ v(n) \end{bmatrix} \tag{10}$$

Every machine learning algorithm requires the environment in which it operates, the possible list of actions and the reward policy (often discounted reward is used) for getting closer to the goal. Markov's Reward Process (MRP) states that in order to get to the next state, you only need to know the current state. From this, optimal actions and reward functions can be derived. However, direct solution only possible for small MRPs. There are many iterative methods for large MRPs, e.g. Dynamic programming, Monte-Carlo evaluation, Temporal-Difference learning [11] For any machine learning scenario described below, our environment is the Heisenberg Hamiltonian spin-½ ring which the reinforcement learning should change the actions (biases and time) to get closer to the target state (output qubit) using the reward policy given (change in fidelity works the best as discovered).

3.2.1 REINFORCE pseudocode

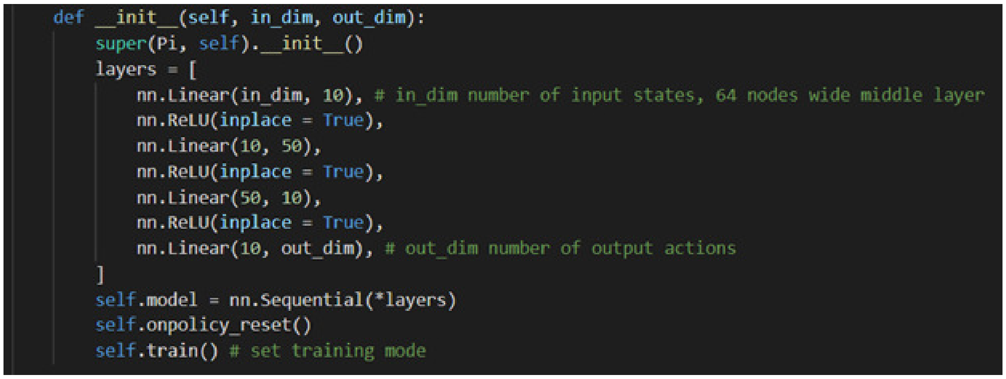
Reinforce is one of the basic machine learning algorithms. Whereas we touched on it above, I want to go into a deeper detail about its implementation. Here is the pseudo code for REINFORCE:

```
Function REINFORCE():
    Initialise $\theta$ arbitrarily;
    for each episode $\{s_1, \alpha_1, r_2, ..., s_{T-1}, \alpha_{T-1}, r_T\} \sim \pi_\theta$ do
        for t=1 to T-1 do
            $\theta \leftarrow \theta + \alpha \bigtriangledown_\theta log\pi_\theta(s_t, a_t) v_t$ ;
        end
    end
    return $\theta$ ;
```

Algorithm 1: Reinforce with baseline (episodic), adapted from [14], [15]

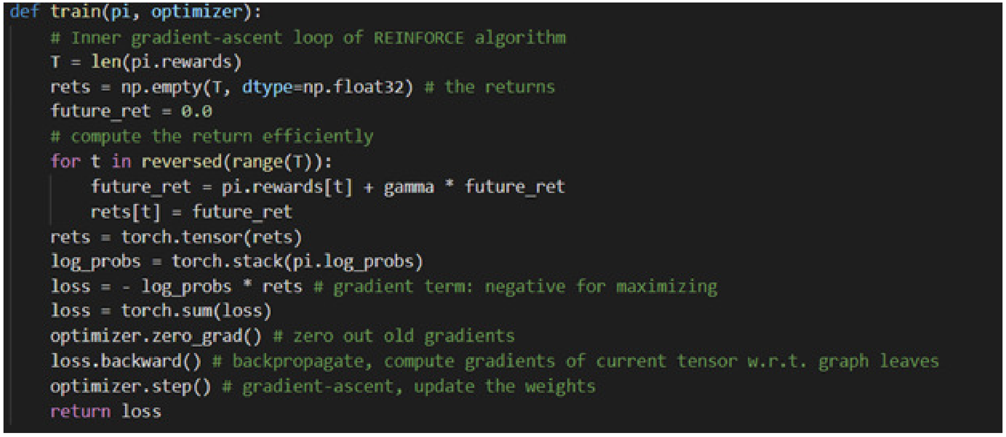
The algorithm first explores the trajectory using the current policy, storing log probabilities (of policy) and reward values at each step. Then the reward is calculated for each of the trajectories explored. After this, the policy gradient is computed, and policy parameters are updated. The algorithm is then repeated again.

In my code we can see it being done by using Pytorch library (which is one of the best machine learning libraries). We are using the following layer structure for the reinforce to work upon:



```
    def __init__(self, in_dim, out_dim):
        super(Pi, self).__init__()
        layers = [
            nn.Linear(in_dim, 10), # in_dim number of input states, 64 nodes wide middle layer
            nn.ReLU(inplace = True),
            nn.Linear(10, 50),
            nn.ReLU(inplace = True),
            nn.Linear(50, 10),
            nn.ReLU(inplace = True),
            nn.Linear(10, out_dim), # out_dim number of output actions
        ]
        self.model = nn.Sequential(*layers)
        self.onpolicy_reset()
        self.train() # set training mode
```

And this is the code implementation of a reinforce pseudocode described in Algorithm 1:



```
def train(pi, optimizer):
    # Inner gradient-ascent loop of REINFORCE algorithm
    T = len(pi.rewards)
    rets = np.empty(T, dtype=np.float32) # the returns
    future_ret = 0.0
    # compute the return efficiently
    for t in reversed(range(T)):
        future_ret = pi.rewards[t] + gamma * future_ret
        rets[t] = future_ret
    rets = torch.tensor(rets)
    log_probs = torch.stack(pi.log_probs)
    loss = - log_probs * rets # gradient term: negative for maximizing
    loss = torch.sum(loss)
    optimizer.zero_grad() # zero out old gradients
    loss.backward() # backpropagate, compute gradients of current tensor w.r.t. graph leaves
    optimizer.step() # gradient-ascent, update the weights
    return loss
```

3.2.2 Deep Q learning pseudocode

Deep Q-learning uses the best characteristics of the reinforce however is deemed to be more advanced one. As a reminder of what is different from the reinforce algorithm - we approximate the optimal action-value function by deriving the policy from a Q function used to evaluate state-action pairs and carry out policy evaluation via TD methods to obtain the next Q function. DQN uses a neural network ability to learn to estimate the Q-value function. The input for the network is the current Q-value, while the output is the corresponding Q-value for each of the action [12].

```
Initialise replay memory D to a capacity N;
Initialise action -value function Q with random weights $\theta$;
Initialise target action-value function $\hat{\theta}$ with weights $\theta^{-} = \theta$;
for episode = 1, M do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$ ;
    for t=1, T do
        With probability $\varepsilon$ select random action $a_t$;
        otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$;
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$;
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$;
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D;
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D ;
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{\grave{a}} \hat{Q}(\phi_{j+1}, \grave{a}; \theta^{-}) & \text{otherwise} \end{cases}$;
        Perform a gradient descend step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to network parameters $\theta$ ;
        Every C steps reset $\hat{Q} = Q$ ;
    end
end
```

Algorithm 2: Deep Q learning with experience replay (episodic), adapted from [14], [15]

Algorithm 2 shows the importance of the experience replay – the network stores values in its memory (therefore the local machine should have enough memory to store the parameters of the runs) and retrieves them accordingly to find the best parameters based on the current run and the knowledge it has so far, therefore constantly improving. The layer structure I am using is almost the same as the reinforce one, however there are more nodes (20 instead of 10) in my DQN implementation – as the algorithm is more advanced, more complex network structure is needed.

3.3 Producing the graphs

In order to create the resulting plots, matplotlib library as well as writing to excel file techniques were used. With matplotlib, it was convenient to see the results after each run separately whilst developing and testing my neural networks. However, as I wanted to also plot the medians of the values found and the comparisons of several methods during the course of several runs on a single graph, the most convenient method I discovered was to record all these values needed for the analysis in the excel file and create a graph using the excel functionalities.

To analyse the graphs, two metrics were used – median and standard deviation. Both of them were chosen as the most reliable for this experiment. Compared to mean, median is more resilient to extreme values found which gives an advantage if one or two values are significantly off compared to the rest as it represents more accurate value. Standard deviation allows to make conclusions on how robust and reliable the controller is – as higher standard deviation, less reliable the algorithm is in finding the robust controls.

4 Evaluation & Results

In this section the results obtained will be discussed and the methods explored during the project will be compared against each other. This will allow to draw a reliable conclusion if machine learning methods can be a new way of finding the robust controls.

4.1 The experiment

For comparison with other algorithms used, the bias range given is 2-60 (with a step of 0.02) and the time is fixed at 7.13 – thus the solutions found will be in one of the shortest time possible.

First attempts to create neural networks had the time action as well (increase/decrease the time) and therefore could operate within the min-max time range given, however the experiments were not deemed successful – time has affected the performance significantly confusing the neural network. Therefore, I left the time at 7.13 (for both min max time given). In order to validate the output solutions (control vector) of my neural network, I checked them against L-BFGS result returned – thus I know if neural network is on the right track and learning.

It is important to understand that the overall the dynamics of the system can be represented as an oscillation – therefore longer the time, more peaks are available – however more local minimas the neural network has to fall into as well. That is why there is a Boost parameter in the action space in both neural networks. Boost helps the neural network to get out of the minimas discovered if it gets stuck and the increase in fidelity is not significant – by doing “boost” action and increasing the default step it has a chance to climb back up to the peaks.

All experiments were conducted at 5% noise level. The noise is added into the system using the following algorithm:

1. The Muller/Marsaglia sampling of d-sphere is performed first. Then it is normalized so that it gives a uniform distribution of the points on a sphere. This is the noise matrix (u) which will be needed at the next step.
2. Using the u noise matrix, S matrix is produced – this matrix has entries from 0 to N-1 on the diagonal and N-1 and N-1 in the off diagonal. It is vital that S matrix is symmetric and has a unit direction.
3. The newly obtained S matrix is added to the current Hamiltonian with scaling factor n (which can be drawn from uniform distribution). This Hamiltonian is the distorted Hamiltonian with noise (n=0.05) that is used in this experiment.

Without the noise, there is no doubt that L-BFGS algorithm will perform the best because of its robust mathematical calculations. However, under the noise L-BFGS tend to perform worse than any of the neural networks. This happens due to the fact that the noise affects the function for the slope and therefore L-BFGS cannot find the correct peaks despite using the correct formula. As machine learning algorithms have to discover the formulas themselves, they are less affected by the noise parameter and demonstrate stable results even with the noise applied.

The machine learning solution is valid for noise levels up to 25% - after this, the disturbance in the system is too high for the network to cope with.

4.2 REINFORCE

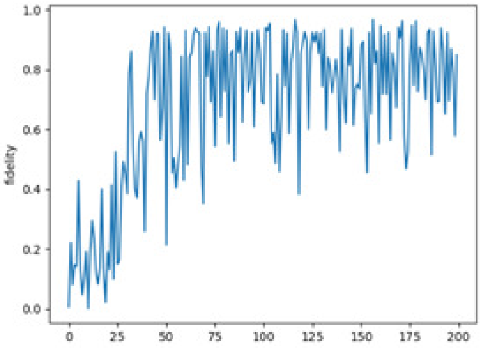


Figure 1: REINFORCE learning curve, 200 iterations, no noise

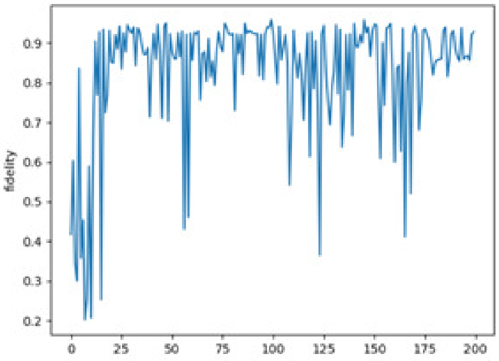


Figure 2: REINFORCE learning curve, 200 iterations, 5% noise

Based on Figure 1, Figure 2, the undisputed fact that the REINFORCE is correctly learning the landscape can be seen.

Figure 3 shows that REINFORCE stably reaches the 0.935 fidelity target with a median value of 0.95-0.96 for the controls found – showing equally efficient results both with 1% and 5% noise (at 5% noise level the median is even higher than at 1% noise level). Standard deviation in all of the cases is no more than 1% and therefore we can trust the results obtained and make a conclusion that the controls are robust.

4.3 Deep Q learning

In contrast with REINFORCE, we do not see such a smooth learning curve on Figure 4 and 5 in comparison with Figure 1 and 2. The target fidelity (in this case, 0.9) is still reached (exactly as in REINFORCE), however sudden drops in fidelity are more frequent.

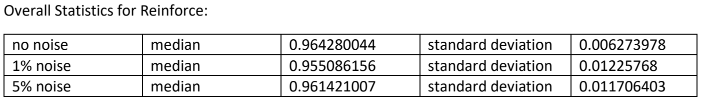
According to Figure 6, we can see that the results obtained are slightly worse than the ones obtained using a REINFORCE algorithm. The controls found are still robust to the relatively same extent (standard deviation is no more than 1%), however the median under noise is much lower:

	REINFORCE	DQN
1% noise	0.955086156	0.931184599
5% noise	0.961421007	0.92166763

There are few reasons why the more complex DQN neural network performed worse compared to a simpler REINFORCE approach:

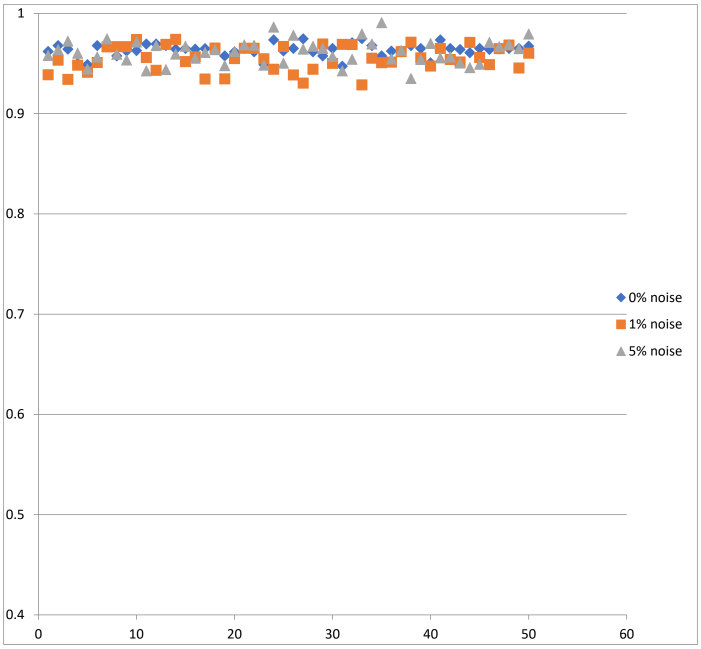
1. The algorithm may simply need more time to learn the Q function. The limit on number of iterations was purely to the local machine constraints (one iteration takes approximately 45 minutes to be completed) and when this number of iterations can be enough for the REINFORCE approach, it is still not enough for the Deep Q-learning one. Some experiments were conducted with more than 700 trajectories which allowed Deep Q-network to explore more actions and improve its learning, however due to the long runtime I had to sacrifice the number of iterations for the better parameters to be implemented.
2. The Boost parameter in Deep Q-learning solution needs better adjustments. If a parameter is too big, the peak is missed, however if it is too little the peak is never reached. I found out that the parameter currently implemented works relatively good and therefore decided to keep it – however, more experiments with different parameters can be done to define the optimal one.
3. The target fidelity reached at which we consider that the network has learnt its Q function is lower than the target fidelity in the REINFORCE algorithm. Different target fidelity was tested:
    - If the target fidelity is more than 0.92, the target is never reached within 200 iterations – therefore the neural network is not encouraged to try to reach higher fidelities values and drops to 0.3-0.4 fidelity obtained.
    - If the target is lower than 0.8, 200 trajectories is enough to train the neural network, however it does not seem to be encouraged to look for fidelity that is more than 0.84 and just plateaus there.

Figure 3: REINFORCE statistics at 0%,1%,5% noise levels



Overall Statistics for Reinforce:

no noise	median	0.964280044	standard deviation	0.006273978
1% noise	median	0.955086156	standard deviation	0.01225768
5% noise	median	0.961421007	standard deviation	0.011706403



- Based on the experiments above, the fidelity boundary that is currently used is 0.9 (compared to the 0.92 in REINFORCE)

4.4 L-BFGS

In order to make a proper comparison and to see at which level L-BFGS algorithm starts performing poorly, we can look at this table and the graph (Figure 7). Both maximum and minimum time are still fixed at 7.13.

As predicted, at just 1% noise level L-BFGS cannot find any useful controllers anymore (standard deviation is about 40% at this noise level) and therefore the results given by both neural network solution with 5% noise that are in the range of 0.9-1.0 are clearly better. The reason for this is simple – as L-BFGS is a gradient-based model assuming the target function it optimises is not noisy, incorrect input (slightly distorted Hamiltonian) leads to incorrect controls found.

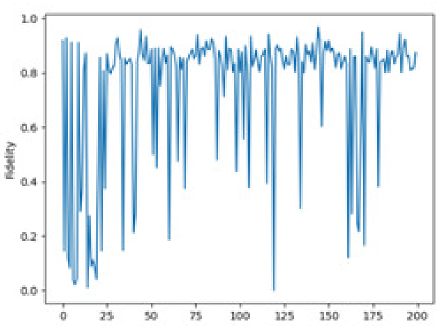


Figure 4: Deep Q learning curve, 200 iterations, no noise

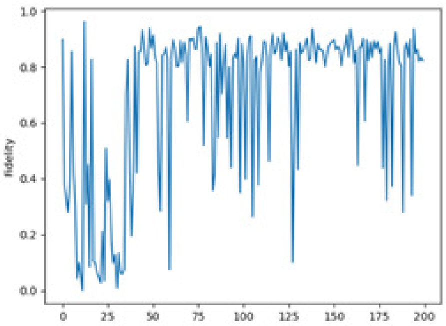


Figure 5: Deep Q learning curve, 200 iterations, 5% noise

5 Future work

Another approach that could demonstrate better results can be the use of temporal difference reinforcement learning. Temporal difference algorithm applies the knowledge of the agent on every timestep (action) rather than on every trajectory (reaching the goal or end state). Therefore, with this experiment where we had 24 actions, we could have explored in detail it would be a better choice. However, despite the possibility of better results the method was proven harder to implement in practice due to the amount of computer memory required and the runtime for each epoch. Thus, the decision to use Deep Q-learning approach was made.

The other approach worth looking at is Trust Region Policy Optimization (TRPO). This algorithm is effective for optimizing large nonlinear policies such as neural networks. The current problem is that Policy Gradient methods use the first-order derivative and approximate the surface to be flat. In this problem's case, the surface has curvature involved therefore sometime the neural network can make horrible moves[13].

In the trust region, we specify the initial maximum step size (will be modified with the iterations if required) that needs to be explored and then search for the optimal point within this trust region (a circle with a radius of maximum step size). The process is repeated iteratively until reaching the peak. Other ideas on how to improve the results include:

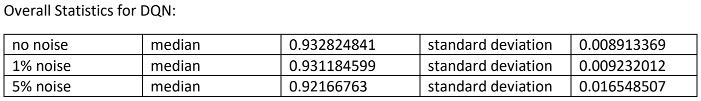
- Making the time a dynamic parameter but only after the high fidelity (e.g. >0.93) is achieved. This way, the program will no longer be concentrated on finding the right parameters (as they are given) however will try to do this in a shortest time possible. This can be imagined as two neural networks following each other – when one can pass its best control's information to the other one which will find the best time parameters for the particular controls.
- Test the reinforcement solutions discussed in this work on larger variety of problems – including chain dynamic systems and rings of the larger size. With more data and the variety in systems more reliable conclusions can be made
- Better and more complex reward function can be developed to address the issues found in the Deep Q-learning solution.

6 Conclusion

There is no doubt that under ideal circumstances L-BFGS will perform the best amongst the tested in this paper algorithms. However, as soon as input Hamiltonian is distorted (and in real experiments there will be inevitable noise in the system), the controls are no longer robust and therefore target fidelity (higher than 0.99) can be no longer achieved.

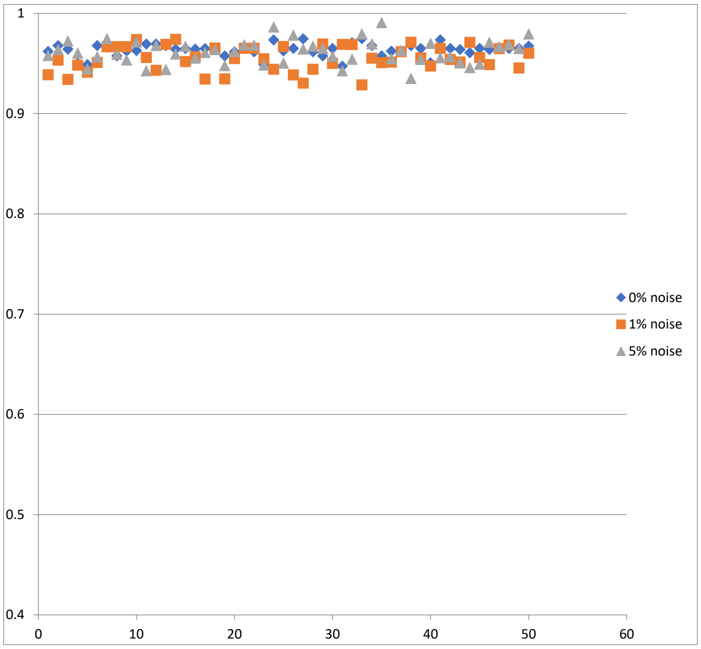
Reinforcement learning methods allow the controllers to be robust (standard deviation is less than 1% for all the methods explored) under noise simulation as well. Even the results are lower than controls found by L-BFGS algorithm with no noise, the target fidelity of 0.9 is still achieved in both cases. We can draw a conclusion that the quality of the controls is sacrificed for its robustness. Future work is required to see if alternations to the explored algorithms, particularly Deep Q-learning, can lead to a higher fidelity without a robustness loss.

Figure 6: DQN statistics at 0%,1%,5% noise levels



Overall Statistics for DQN:

no noise	median	0.932824841	standard deviation	0.008913369
1% noise	median	0.931184599	standard deviation	0.009232012
5% noise	median	0.92166763	standard deviation	0.016548507



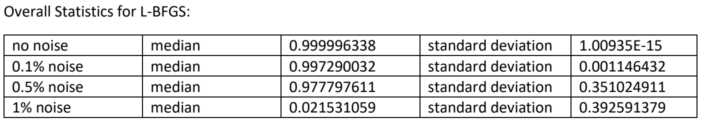
7 Personal Reflection

1. Time management challenges

   Time management of this project was one of my biggest concerns throughout the semester. Due to its experimental nature, the progress could have occurred at different rates throughout its length – for example, there were weeks when I needed a time to just read on some background information and therefore no new code was developed. However, next week was dedicated to coding what I have learnt the previous week – and therefore the entire code for one of the algorithms (reinforce) was completed in two days. The biggest change for me was to figure out my actions for the environment – it took me a while to understand a simple concept that changing biases are my Action parameters. This caused a 2-week delay in code development which I was afraid could affect the final result. However, once I solved this problem the rest became straight forward and therefore, I have managed to catch up in a few weeks. Next time, I will ensure I have at least 2 weeks before any "progress checkpoints" rather than just a single week (as I have originally planned and what can be seen in the initial plan) – this way if a slight delay occurs on the way, I will still be on track and within the deadline.

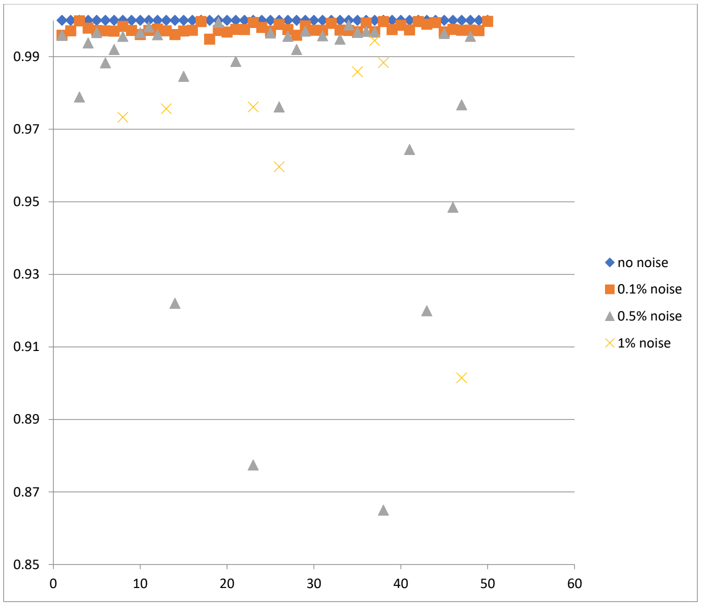
2. Background knowledge

Figure 7: L-BFGS statistics at 0%,0.5%,1% noise levels



Overall Statistics for L-BFGS:

no noise	median	0.999996338	standard deviation	1.00935E-15
0.1% noise	median	0.997290032	standard deviation	0.001146432
0.5% noise	median	0.977797611	standard deviation	0.351024911
1% noise	median	0.021531059	standard deviation	0.392591379



The problem that I was solving required a certain background knowledge of quantum computing. Despite me starting to read the information available prior to Christmas break, I quickly realized that the problem is more advanced that seems on the first sight and therefore there is not only the general concept that needs to be understood but also the way certain algorithms perform (for example, L-BFGS and its structure was very new to me, as well as any machine learning algorithms that I had to read two books about) and where those come from. I have discovered for myself a new area of computer science whilst working on this project as well as learnt how to quickly adapt to the new environment. This skill will help me in my future life as I cannot imagine myself working in one sector all my life and therefore being able to quickly adapt to different situation and environment is a very useful skill.

3. Adaptability to failures

   Due to experimental nature of the problem, there was a significant doubt in final results being produced, or, if successfully obtained, that they will be proved useful. The initial experiments which involved time parameters as actions did not show the progress of neural network and learning curve looked poorly which made me doubt my own abilities. I had to remind myself that I am working with an experimental problem and every result is useful – does not matter whether the solution with the method used shows a significant progress or nothing at all. This realization made me calmer about producing my final results as well as will (hopefully) make me more resilient to failures in the future life.

4. Plot analysis decisions

   From early on I had start to think how exactly I need to plot my findings. There were several risks regarding it:

   - The large number of graphs produced for a various modification of reinforcement learning algorithms. The challenge in this scenario is that in order to demonstrate trends, all those graphs need to be united together. However, running hundreds of experiments for each little modification is very time consuming and requires extra time for the deep analysis. Therefore, I had to make a choice to investigate only one particular example (5 spin-½ ring) of the problem given. This has taught me that sometimes I have to choose between options and can't do everything – and this is fine and it is not a disadvantage. Better to produce one good quality work than to spread thin on lots of things.
   - Metric to be used in the analysis. It was not enough to just plot the graphs, the right metrics need to be chosen to summarise the results so that findings are clear for the readers. I had to research which metrics are there and chose two that I think are most suitable for this particular problem. This little analysis problem has taught me that even smallest things need attention and careful consideration – as the wrong decision can nullify all the previous things you have done, no matter how impressive and huge they are.

References

[1] Dalgaard, M. et al. (2020) Global optimization of quantum dynamics with AlphaZero deep exploration [Online] Available at: https://www.nature.com/articles/s41534-019-0241-0. NPJ Quantum Information, 6:6

[2] Khalid, C. et al. (2021) Reinforcement Learning vs. Gradient-Based Optimisation for RobustEnergy Landscape Control of Spin-1/2 Quantum Networks. Preprint

[3] Glaser, S. et al. (2015) Training Schrodinger's cat: quantum optimal control.The European Physical Journal D, 69:279

[4] Dong D. et al. (2010) Quantum control theory and applications: A survey. IET Control Theory Applications, 4:12

[5] Langbein, F. et al. (2015) Time optimal information transfer in spintronics networks. Proc. IEEE 54th Annual Conference on Decision and Control (CDC), pp. 6454-6459.

[6] François-Lavet, V. wt al. (2018) "An Introduction to Deep Reinforcement Learning", Foundations and Trends in Machine Learning: Vol. 11, No. 3-4. DOI: 10.1561/2200000071

[7] Williams, R. et al. (1992) Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning volume 8, pp. 229–256.

[8] TORRES, J. (2021) Policy-Gradient Methods [Online] Available at: https://towardsdatascience.com/policy-gradient-methods-104c783251e0

[9] Ramaswamy, A et al. (2021) Deep Q-Learning: Theoretical Insights from an Asymptotic Analysis. arXiv:20008.10870v2

[10] SpinNet - Quantum control schemes for spin networks with a focus on robust quantum control and energy landscape control. Available at: https://qyber.black/spinnet/code-matspinnet

[11] Silver, D. Reinforcement Learning: Markov Decision Processes. [Online] Available at: https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf

[12] Mnih, V et al. (2015) Human-level control through deep reinforcement learning. Nature Vol.518, pp.529–533.

[13] John Schulman Joschu, J et al. (2015) Trust Region Policy Optimization. ICML.

[14] Sutton, R. S. and Barto, A. G. (2018) Reinforcement Learning: An Introduction (2nd Edition, in preparation). MIT Press.

[15] UToronto lecture slides, (2018) Machine Learning and Data Mining , [Online] Available at: http://www.cs.toronto.edu/ rgrosse/courses/csc411f18