Hybrid-Quantum Neural Architecture Search for The Proximal Policy Optimization Algorithm

Moustafa Zada[1]

[1]The University of Aleppo, Aleppo, Syria.

Contributing authors: moustafa7zada@proton.me;

Abstract

Recent studies in quantum machine learning advocated the use of hybrid models to assist with the limitations of the currently existing Noisy Intermediate Scale Quantum (NISQ) devices, but what was missing from most of them was the explanations and interpretations of the choices that were made to pick those exact architectures and the differentiation between good and bad hybrid architectures, this research attempts to tackle that gap in the literature by using the Regularized Evolution algorithm to search for the optimal hybrid classical-quantum architecture for the Proximal Policy Optimization (PPO) algorithm, a well-known reinforcement learning algorithm, ultimately the classical models dominated the leaderboard with the best hybrid model coming in eleventh place among all **unique** models, while we also try to explain the factors that contributed to such results, and for some models to behave better than others in hope to grasp a better intuition about what we should consider good practices for designing an efficient hybrid architecture.

Keywords: Quantum Neural Architecture Search, Quantum Reinforcement Learning

1 Introduction

Quantum machine learning (QML) has emerged in recent years as a promising approach to advance classical machine learning, considering that they can evaluate classically intractable functions [1] since they have properties that the classical computational models don't have access to like the exponential Hilbert space, entanglement, and the parallelistic nature of quantum computation in the presence of superposition.

But, Since we are still in the NISQ era, with limited quantum computers, many studies have discussed the use of hybrid architectures in the hope that they can get an advantage from the currently existing quantum computers or if they could find a way to distribute the work harmonically and efficiently between the classical and the quantum part, each part with what they are good at. An important Distinguish that had to be made here is that when we are not considering variational quantum circuits as hybrid classical-quantum models, since they use the classical computer only to train and optimize the quantum circuit but they don't interfere with the actual learning and the inference of the model, instead, we treat the VQC itself as a layer, a quantum layer, in the neural network of the hybrid model along with classical layers with artificial neurons.

In this paper, we apply the proximal policy optimization (PPO) algorithm to the classical CartPole environment problem to ascertain whether the quantum-enhanced PPO algorithm gives any advantage over the classical version. In particular, we use a genetic algorithm-esque version of quantum PPO to train the system.

This paper is divided as follows:

In Sec.2, we examine analogous studies to the experiments we have performed in order to gauge what has been done, and how we can improve upon it.

In Sec.3, we provide a brief synopsis of the operation of the PPO algorithm.

In Sec.4, we provide the details of the experimental constraints considered in the training and the results from the training.

In Sec.5 and 6, we reflect on the results obtained and allude to future considerations we hope to investigate.

2 Related Work

As demonstrated in this comprehensive survey on Quantum Reinforcement Learning (QRL), most of the literature focuses on developing the VQC themselves, their ansatz (or architecture often called in the literature , contrary to what we mean by architecture in this paper), and using them as function approximation instead of the classical neural networks, but some extensions are made to hybrid quantum classical models like those we are working on in this paper, for instance, Chen(2023) [2] constructed a hybrid model for the A3C actor-critic algorithm that has a linear layer before the VQC, which was a well known and widely used ansatz for VQC, and it shown comparable or even superior results to other classical models with comparable sizes, in another work by Chen et al.(2023)[3] they used also a hybrid model with two convolutional layers, a linear layer, a quantum layer, then a linear layer to solve a maze navigation problem using Deep Q-Learning algorithm, the model demonstrated comparable although slightly worse performance than the classical counterpart but with less parameters, a well-known benefit from using VQC. Another interesting paper by Lockwood and Si(2021)[4] that tried to use QRL for a more challenging task, playing atari games, the authors experimented with 12 different hybrid configurations while all of them had a common feature, having a classical layer before the quantum layer to reduce the high dimensions of the data coming from the atari game to the VQC, which as the authors suspect, the extreme reduction of size lead to all models failing to learn and the classical model with sutabele number of parameters performing much better, they came to this conclusion considering that *a.* the models don't have anything fundamentally wrong with them or bugged code since they can learn other simpler tasks, *b.* a classical model with $O(1o^4)$ doesn't learn either, indicating that the problem is too hard for small models quantum or classical to learn the underlying function approximations. a notable work by Dragan et al.(2022)([5] have they conducted experiments with 19 different hand-picked and well-known ansatz for VQC as a part of a hybrid model for a quantum variation of PPO that has a VQC then a 4-neuron linear layer as the architecture for both networks(actor and critic), in this study the authors to analyze the results that they got in hope to understand why some VQC ansatzes are performing better than others while having the same 4-neuron pre-processing layer with same hyperparameters, three metrics were considered, expressibility, entanglement capability, and the effective dimension, eventually, No direct and clear correlations were shown to the performance, keeping this question unanswered.

we build on that work to search without **Human Bias** for more than 1000 iterations for PPO since most of the work done above either hand-picked famous ansatz architectures or just stated that the model that they have picked proved to be the most promising after conducting experiments. Thus, this work investigates this gap in the literature, using an evolutionary algorithm for Neural Architecture Search (NAS) called Regularized Evolution [6] which proved to be quite effective in searching for an optimal or sub-optimal architecture surpassing currently used reinforcement learning approaches for NAS, we used this algorithm to search for an efficient architecture-finding the optimal one is not guaranteed- for the Proximal Policy Optimization Algorithm [7]. This work also tries to explain the experimental results obtained by running the search algorithm for **1000+ iterations**.

3 Background

3.1 Proximal Policy Optimization

Briefly, **PPO** is an actor-critic algorithm, meaning there are two models involved, an Actor which *Acts* in the environment and takes actions, and a Critic which evaluates the Actor's actions contributing to the objective function(the loss function) of the Actor by determining the goodness/badness of the actions at a specific observation state. generally speaking, the only interesting thing about PPO is the brilliantly crafted loss function, which is clipped to ensure that the changes of the network parameters in the weights space are small and do not diverge widely if you encounter an action state pair that the model thinks is too good or too bad, taking moderate steps in such extreme cases makes the training much more stable. the objective function for the Actor without clipping:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]. \tag{1}$$

After clipping:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right]. \tag{2}$$

the complete loss function for the whole model, consisting of the loss for the Actor, the loss for the Critic, and the entropy of the Actor's actions probabilities which introduces some randomness to our model to encourage exploration

rather than consistent exploitation:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right]. \tag{3}$$

Note that our experiment used the same architecture for both networks, inspired by famous, well-documented implementations of the PPO algorithm [8].

3.2 Regularized Evolution

The way that Regularized Evolution[6] works is quite straightforward: we start with a **P** number of randomly generated architectures, and we keep only **P** number of architecture at our hand at any time, we call it the **population**, and we keep another list of arch. we call the **history** were we store every architecture we encounter, then we iteratively and randomly pick a sample of size **S** from the population and mutate the highest scoring one of them, then, train it, add the newly trained model to the population, and delete the oldest arch. in the population even if it has the highest score, statistically its guaranteed to get back to that arch. or to another even better one so it's safe to do this, and that's why the authors called it aging evolution throughout the paper.

4 Experiments

When designing the mutation, you essentially create the search space for the search algorithm, meaning that the more that you add freedom to the mutations that could be taken, the less human bias that you introduce into the search, and the higher the chance to find interesting and unusual results, the mutations act on a DNA sequence that has a detailed explanation in the code repository [9].

4.1 The Mutations Used

Note that some tuning to the neighboring layers is required after applying most of these mutations. The details of such changes are present in the code [9].

1. **Add a Classical Layer:** Add a classical layer at a random position.
2. **Remove a Classical Layer:** Remove a Classical Layer picked randomly.
3. **Add a Quantum Layer:** Add a quantum layer in a location chosen at random.
4. **Remove a Quantum Layer:** If a quantum layer exists, remove it.
5. **Alter a Layer [Add]:** Add a random (constrained) number of neurons or qubits to a randomly chosen layer.
6. **Alter a Layer [Remove]:** Remove a random (constrained) number of neurons or qubits to a randomly chosen layer.
7. **Change the repetitions of the ansatz:** Pick a quantum layer at random and change its number of layers in the ansatz.
8. **Change the Entanglement type of the Ansatz:** Pick a quantum layer at random and change its entanglement type.
9. **Change the Activation Function of a Layer:** Pick a classical layer randomly and change its activation function. The supported functions are either Tanh or ReLU.
10. **Identity:** Do not change anything.

4.2 Experimental Constraints

1. Maximum number of layers = 10.
2. Maximum number of neurons in a classical layer = 64.
3. Maximum number of qubits in a quantum Layer = 10.
4. Minimum number of neurons in a classical Layer = 2.
5. Minimum number of qubits in a quantum layer = 2.

An Important Detail of this implementation is that there are two kinds of quantum layers used, layers that output only the expectation value of at most n qubits, where n being the number of qubits of the layer under consideration, and layers that outputs 2^n measurement results for all the possible bit strings using the default shots number of 1024, the former was used if there are two quantum layers adjacent since the input of any quantum layer should be equal to its qubit count n thus using the second option was not plausible, or the network would get very shallow if two quantum layers come after each other because of a random mutation, cause in that case we should use logarithmic

values of the second quantum layer's qubits n_2 as the qubit number of the first quantum layer n_1.

$$\log_2(n_2) = n_1$$

The latter approach was used if there is a classical linear layer after that quantum layer, which takes any number of inputs.

This idea is not very usual. It was taken to limit the extreme shallowness of the network in case two quantum layers come together and to limit the human bias to allow the search algorithm to be as free as possible. However, it might have negatively affected the results, which is unclear.

The Experiment was conducted using PyTorch for the Classical Part of the models, and the training, using the ADAM optimizer for the Quantum Layers Pennylane [10], was used. The experiment took approximately 20 Hours. Having the Classical Layers with the parameters of the quantum layers on the GPU – an RTX 3070 with 8GB of VRAM – and the quantum layer execution on the CPU, this configuration might seem un-optimized, but this was the best configuration possible considering missing features in Pennylane and the small width of our circuits makes the execution on the GPU not so intriguing. The used environment is **CartPole-V1**, a famous OpenAI gym environment [11].

4.3 Results

Out of all the 1000+ iterations, **666** unique models were found. **390** models were purely classical without any quantum layers, **236** models had one quantum layer, **34** models had two quantum layers, **4** had three layers, and **2** had four layers, without having any higher number of quantum layers in any model. The Average score of the model over 10 different episodes from 0 to 500 will be used as the sorting factor between all the 1000+ models, the highest scoring model being a classical model with a score of 442.6/500.
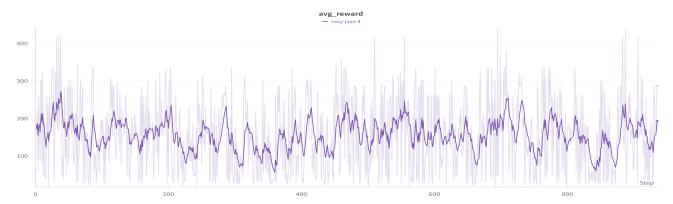

Fig. 1: The Average Reward on The Span of approx. 950 Iterations

If we discard the duplicated models(since when we checked if the model is already trained to use existing values from a previously trained instance of that model) then we would have a leaderboard of the top 10 models shown below in Tab.1, you can see that the first 8 models are Classical models and the 11th unique model is a hybrid model with a small quantum layer in it with two Qubits only. Note that all 11 models achieved at least one perfect score of 500 in one of the evaluating episodes, with the 9th one having three perfect episodes and one episode score equal to only 17 points! , which made it lose to other consistent classical models. This is rather an interesting and unusual behavior that the hybrid model has shown.

The Used encoding method is angle encoding, and in the shown DNA sequences, **C** stands for a classical layer with the number of its neurons next to it, **R** and **T** are activation functions and **Q** stands for a quantum layer with its number of Qubits next to it, along with the entanglement type of the layer, either **L** for *BasicEntanglingLayer* or **F** for *StrongEntanglingLayer* with the number of repetitions next to it.

DNA Sequence	Average Reward
C 37, T, C 41, T, C 2, R, C 24, T, C 5, T, C, 1	442.6
C 50, T, C 13, R, C 2, T, C 9, R, C 64, T, C 42, T, C 38, R, C 1	437.7
C 50, T, C 13, R, C 2, T, C 9, R, C 64, T, C 42, R, C 38, R, C 1	418.6
C 50, T, C 16, R, C 2, T, C 50, R, C 64, T, C 42, R, C 38, R, C 1	389.7
C 57, R, C 41, T, C 2, R, C 24, T, C 5, T, C, 1	381.2
C 50, T, C 13, R, C 2, T, C 59, R, C 55, T, C 42, R, C 38, R, C 1	370.6
C 50, T, C 16, R, C 3, T, C 9, R, C 64, T, C 42, R, C 38, R, C 1	366.9
C 50, T, C 13, T, C 2, T, C 9, R, C 64, T, C 42, R, C 38, R, C 1	355.1
C 51, R, C 24, T, C 5, T, C, 1	341.5
C 37, R, C 38, T, C 6, T, C 5, T, C 1	340.8
C 50, T, C 13, R, C 2, T,**Q 2 F 3**, C 9, R, C 55, T, C 29, R, C 42, R, C 38, R, C 1	**339.7**

Table 1: Results from the 11 best-performing experiments.

5 Conclusions

Empirically, we can conclude some observations and patterns in the data:

- One can see in Table 1 that the training is not stable, while most of the instability comes from tweaking the quantum layers, the smallness of the networks which makes them so sensitive to any change in the network is also a big contributor to the instability, which means that the used mutations are too free and random for such an experiment.
- The Idea of using a big quantum layer that we do not measure all of its qubits, and relaying on the entanglement to propagate the information between the qubits(throughout the width of the circuit) is not a great idea, as it is showing in the data[9] most of the lowest performing models share that feature, which hints that having a healthy backpropagation throughout your model, where all parameters do make a difference to the loss value,is far better than this case where the optimizer tunes the parameters without a real difference to the loss because the learning is dependent on the propagation of information via entanglement rather than directly.
- Quantum Layers with 4 or 3 Layers were only present in the random generation of the population phase of the evolution algorithm and the algorithm didn't favor them again due to their bad performance.
- Quantum layers with many qubits proved to be harder to train using the used hyper-parameters, which made them not favorable by the search algorithm.
- By mutating the best hybrid-model 1 by hand, we could see that changing the entanglement from *Strong* to *Basic* results in the average reward being halved(while CartPole is not the most accurate environment which may result in the huge variation that happens) this suggests that having a quantum layer with a small number of qubits and high entanglement is favorable.
- Finally, this experiment showed that quantum variational circuits as quantum layers in hybrid models in their current form do not hint at any advantage over using well-designed classical models.

6 Future Work

In coming versions of this exact paper or continuations of this paper, I hope that I can test more environments to get more coherent data and provide stronger evidence for the conclusions, an effort that I have not made yet due to the computational demand that other environments require since the used one is considered quite simple, and it took 20 hours of training despite that. In addition, more neural architecture search algorithms should be explored. Ideally, **new custom** hybrid Quantum-Classical NAS Algorithms should be developed to aid the existing efforts to adopt hybrid models in real-world applications.

Acknowledgments

Disclaimer: This research was neither funded nor supervised by any organization or institution, including the home university of the author. It represents an independent effort by the author.

Code and Data Availability

The code and data for this project may be accessed from the author's GitHub account [9].

References

[1] Killoran, N.: Hybrid Quantum-Classical Machine Learning. Xanadu. https://www.youtube.com/watch?v=t9ytqPTij7k Accessed 2020-11-17

[2] Chen, S.Y.-C.: Asynchronous training of quantum reinforcement learning. arXiv. arXiv:2301.05096 [quant-ph] (2023). https://doi.org/10.48550/arXiv.2301.05096 . http://arxiv.org/abs/2301.05096 Accessed 2024-11-26

[3] Chen, H.-Y., Chang, Y.-J., Liao, S.-W., Chang, C.-R.: Deep-Q Learning with Hybrid Quantum Neural Network on Solving Maze Problems. arXiv. arXiv:2304.10159 [quant-ph] (2023). https://doi.org/10.48550/arXiv.2304.10159 . http://arxiv.org/abs/2304.10159 Accessed 2024-11-22

[4] Lockwood, O., Si, M.: Playing atari with hybrid quantum-classical reinforcement learning. In: Bertinetto, L., Henriques, J.F., Albanie, S., Paganini, M., Varol, G. (eds.) NeurIPS 2020 Workshop on Pre-registration in Machine Learning. Proceedings of Machine Learning Research, vol. 148, pp. 285–301. PMLR, ??? (2021). https://proceedings.mlr.press/v148/lockwood21a.html

[5] Drăgan, T.-A., Monnet, M., Mendl, C.B., Lorenz, J.M.: Quantum Reinforcement Learning for Solving a Stochastic Frozen Lake Environment and the Impact of Quantum Architecture Choices. arXiv. arXiv:2212.07932 [quant-ph] (2022). https://doi.org/10.48550/arXiv.2212.07932 . http://arxiv.org/abs/2212.07932 Accessed 2024-11-26

[6] Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized Evolution for Image Classifier Architecture Search. arXiv. arXiv:1802.01548 [cs] (2019). https://doi.org/10.48550/arXiv.1802.01548 . http://arxiv.org/abs/1802.01548 Accessed 2024-11-22

[7] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arXiv. arXiv:1707.06347 [cs] (2017). https://doi.org/10.48550/arXiv.1707.06347 . http://arxiv.org/abs/1707.06347 Accessed 2024-11-22

[8] Huang, S., Dossa, R.F.J., Raffin, A., Kanervisto, A., Wang, W.: The 37 implementation details of proximal policy optimization. In: ICLR Blog Track (2022). https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

[9] Zada, M. https://github.com/moustafa7zada/Quantum-Hybrid-NAS-via-Regularized-Evolution/

[10] Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Ahmed, S., Ajith, V., Alam, M.S., Alonso-Linaje, G., Akash-Narayanan, B., Asadi, A., Arrazola, J.M., Azad, U., Banning, S., Blank, C., Bromley, T.R., Cordier, B.A., Ceroni, J., Delgado, A., Di Matteo, O., Dusko, A., Garg, T., Guala, D., Hayes, A., Hill, R., Ijaz, A., Isacsson, T., Ittah, D., Jahangiri, S., Jain, P., Jiang, E., Khandelwal, A., Kottmann, K., Lang, R.A., Lee, C., Loke, T., Lowe, A., McKiernan, K., Meyer, J.J., Montañez-Barrera, J.A., Moyard, R., Niu, Z., O'Riordan, L.J., Oud, S., Panigrahi, A., Park, C.-Y., Polatajko, D., Quesada, N., Roberts, C., Sá, N., Schoch, I., Shi, B., Shu, S., Sim, S., Singh, A., Strandberg, I., Soni, J., Száva, A., Thabet, S., Vargas-Hernández, R.A., Vincent, T., Vitucci, N., Weber, M., Wierichs, D., Wiersema, R., Willmann, M., Wong, V., Zhang, S., Killoran, N.: PennyLane: Automatic differentiation of hybrid quantum-classical computations. arXiv. arXiv:1811.04968 [physics, physics:quant-ph] (2022). http://arxiv.org/abs/1811.04968 Accessed 2024-11-23

[11] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)

[12] White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., Hutter, F.: Neural Architecture Search: Insights from 1000 Papers. arXiv. arXiv:2301.08727 [cs, stat] (2023). https://doi.org/10.48550/arXiv.2301.08727 . http://arxiv.org/abs/2301.08727 Accessed 2024-11-22