Learning Policies for Neural Network Architecture Optimization Using Reinforcement Learning

Raghav Vadhera, Manfred Huber
Department of Computer Science & Engineering
University of Texas at Arlington, Arlington, TX 76019-0015, USA
bvadhera@post.harvard.edu, huber@cse.uta.edu

Abstract

Deep learning systems tend to be very sensitive to the specific network architecture both in terms of learning ability and performance of the learned solution. This, together with the difficulty of tuning neural network architectures leads to a need for automatic network optimization. Previous work largely optimizes a network for one specific problem using architecture search, requiring significant amounts of time training different architectures during optimization. To address this and to open up the potential for transfer across tasks, this paper presents a novel approach that uses Reinforcement Learning to learn a policy for network optimization in a derived architecture embedding space that incrementally optimizes the network for the given problem. By utilizing policy learning and an abstract problem embedding, this approach brings the promise of transfer of the policy across problems and thus the potential optimization of networks for new problems without the need for excessive additional training. For an initial evaluation of the base capabilities, experiments for a standard classification problem are performed in this paper, showing the ability of the approach to optimize the architecture for a specific problem within a given rang of fully connected networks, and indicating its potential for learning effective policies to automatically improve network architectures.

Introduction

Deep Neural Networks have become a powerful and versatile tool but their performance tends to be highly sensitive to the specific network architecture and data set. Automatic Neural Architecture design (ANA) (Kapanova, Dimov, and Sellier 2018) has shown its potential in discovering powerful neural network architectures in a range of problems. However, despite the fact it has been a research topic for years, there still are certain issues regarding the development of an optimized ANA model, including the high complexity of the search utilized. Deriving a good architecture generally depends on the balance of model performance and computational expenses required to train the model, and is heavily influenced by the properties of the data set. To begin, one needs a starting point and initial set of hyper-parameters. In sequential models involving Multilayer Perceptrons (MLP), one of the key hyper-parameters is the number of hidden layers and the number of nodes required for these layers (Gaurav 2019). In the absence of other data finding a good choice is usually up to the experience of the designer, further making the point for better automatic network architecture design tools to make these models more widely usable.

Different approaches for this problem have investigated various aspects of the ANA modelling procedure, from training data collection and pre/post-processing to elaborate training schemes and algorithms. Increased attention is directed to a systematic way to establish an appropriate architecture in contrast to the current practice that calls for a repetitive, time consuming trial-and-error process (Baker, Gupta, Naik, et al. 2016). Reinforcement Learning (RL) has been used to learn a policy to incrementally build neural network classifiers for a specific problem (Baker, Gupta, Naik, et al. 2016)(Luo et al. 2018). However, this approach is limited to very simple problems using a very simple action space. (Bose and Huber 2016) take a similar approach but learn not only one network but an ensemble of networks, allowing them to faster adjust to problem changes.

The goal in this paper is to take a first step in developing a more general framework that uses RL with a more flexible architecture modification action space, that allows to represent a broader task domain through architecture embedding, and that facilitates planning through a performance prediction component. While this paper focuses on a simple initial case, it includes structures that can adapt to larger spaces.

Related Work

A number of approaches for automatic network architecture optimization using RL or evolutionary algorithms (EA) (Zoph and Le 2016) have been proposed. Most of these conduct architecture search in a discrete space, which is highly inefficient. To address this, (Luo et al. 2018) proposed the use of an autoencoder structure to learn a network architecture embedding space, allowing them to perform network optimization using a combination of local gradient ascent and significant heuristic search. This limits optimization to a single learning task and incurs significant computational expenses. This is in strong contrast to (Bose and Huber 2016), where network modification actions are discrete and predetermined but the resulting policy derived from RL is applicable to new problems without the need for large amounts of additional network training. Their method, however, is limited to ensembles of simple fully connected binary and one-vs-all classifiers (Rifkin and Klautau 2004).

This paper proposes an approach that combines the concepts of a network embedding space to obtain a generic action space with a general problem representation and RL framework to learn complete optimization policies using RL that might be applicable to new problems. Taking inspiration from previous work using network embeddings (Luo et al. 2018), Ensembles (Bose and Huber 2016) (Zhou, Wu, and Tang 2002) (Granitto, Verdes, and Ceccatto 2005) and various approaches to network architecture search (Baker, Gupta, Naik, et al. 2016), (Baker, Gupta, Raskar, et al. 2017), (Brock et al. 2017), (Cai et al. 2017), (Kapanova, Dimov, and Sellier 2018) we propose an architecture that provides the promise to extend to various network types as well as to transfer optimization strategies across problems.

Background

We are presenting an approach that uses Reinforcement Learning on a network embedding space to learn a policy that can yield an optimized network architecture. The embedding space here converts the space of network architectures into a lower-dimensional continuous space and also defines a usable, continuous action space for the RL agent.

Embedding Space

Network architecture optimization needs to represent the space of applicable networks and, if using RL, an action space of network modifications. To efficiently represent the set of networks for a deep learning system, much like (Luo et al. 2018) we map a network description language into an embedding space using an encoder-decoder component. This lower-dimensional latent embedding for networks also defines a continuous action space as displacement vectors in the embedding space. The current network embedding vector, augmented with a feature vector encoding the problem for which the network architecture is to be optimized form the state for the RL problem while displacements in the embedding space provide the RL learner's action space.

Reinforcement Learning (RL)

RL (Sutton and Barto 2018) is used here as it allows to learn from quantitative feedback without the need for training data that contains the correct solution. This addresses the problem in network architecture optimization that optimal network configurations for the target problems generally are not known and thus traditional labeled data is hard to obtain.

Utilizing the network embedding space, RL operates here by executing actions in embedding space to modify the network architecture and observing resulting reward in the form of changes in actual or predicted network performance. Based on this, a value function is estimated and a policy is learned that optimizes the obtained rewards. Figure 1 shows this basic operation of the RL learning agent.

Value Function The Q-Value measures performance as the expected utility of the learning agent in state **s** when performing action **a**, and otherwise following policy(π). This value function not only considers immediate reward feedback, $r(s, a)$, but also takes into account future payoffs.

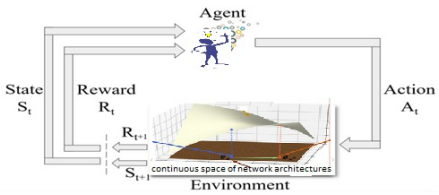

Figure 1: Overview of Reinforcement Learning Agent

$$Q_\pi(s,a) = \sum_{s'} P^a_{ss'}(r(s,a) + \gamma \sum_{a'} \pi(a'|s').Q_\pi(s',a')$$

The Bellman equation (TORRES 2020) provides the basis for performance estimation and policy learning.

In the network optimization problem considered here, reward represents increase in performance achieved by a network modification action and the Q-value tries to predict expected improvements achievable by policy π. The state is provided by concatenating the embedding vector (representing the current network architecture) and a problem specific feature vector (capturing problem complexity related information) while possible actions are continuous displacement vectors in the embedding space. Figure 2 shows the connection of the embedding space and the value function.

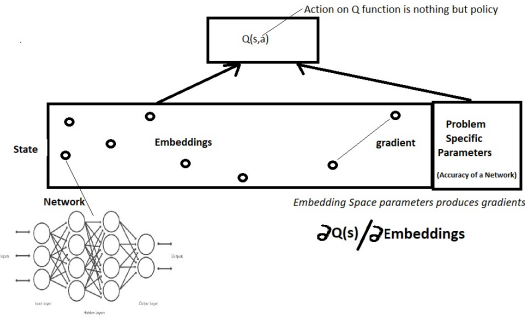

Figure 2: Network Embedding Space (left) and Problem Features (right) as State and Action Space for RL (top)

Policy A policy in the RL problem is a mapping from states to actions, representing a strategy for incremental network modification. The goal of RL here is to learn a policy that maximizes the performance improvements it achieves by modifying the neural network architecture for learning problem. As we use a continuous embedding space as well as a continuous action space in the form of embedding space displacement vectors, we utilize an actor-critic RL approach to perform policy and value function learning.

Actor-Critic Methods and Policy Gradient

Actor-critic (Yoon 2019) and policy gradient (Yoon 2018) are RL methods that can learn value functions and policies in continuous state and action spaces and are amenable to deep learning implementations. In actor-critic systems, the 'critic network' estimates the value function while the 'actor network' updates the policy using policy gradients.

To update the actor-critic system, reward in the form of actual measured (if training online) or predicted network performance changes (if using planning) is used to compute update the critic network using the Bellman error. To update the actor network, the value function prediction is back-propagated through the critic network to the actor network,

effectively providing policy changes that increase predicted Q-values and thus improved performance networks.

Twin Delayed Deep Deterministic Policy Gradient (TD3) A range of deterministic and probabilistic actor-critic and policy gradient algorithms have been proposed. Here we utilize TD3 (Byrne 2019) as a method that has proven robust for continuous action spaces by reducing overestimation bias using a pair of critic networks along delayed actor updates while managing action noise regularization.

Approach

To address the network optimization, the approach in this paper utilizes Reinforcement Learning on top of a network embedding space. The embedding space is learned using an encoder-decoder network structure and provides a compact representation of network architectures for the problem task as well as a corresponding fixed-dimensional, continuous network modification action space. Using this and an additional problem-specific feature vector to capture information regarding the complexity of the target problem, TD3 actor-critic networks are used to learn and execute a network modification policy to optimize the performance on the target problem. To reduce the need for target network retraining and facilitate planning, an accuracy prediction network is added that attempts to predict the performance of the modified network and thus to generate reward predictions. Moreover, the accuracy prediction network is also trained to predict illegal regions of the embedding space which do not correspond to decodable network architectures. The proposed network optimization architecture is shown in Figure 3.

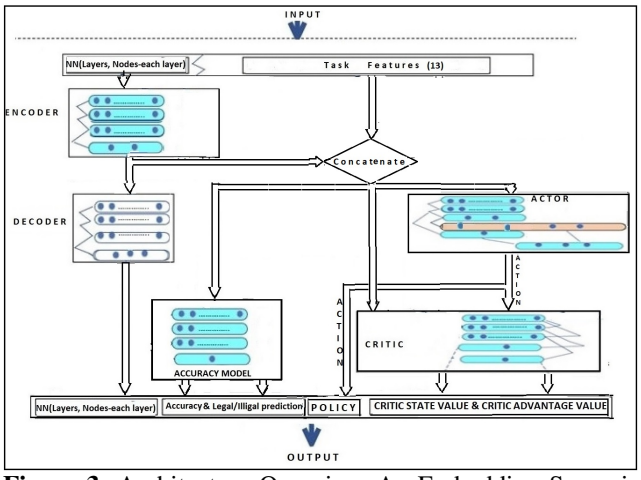

Figure 3: Architecture Overview. An Embedding Space is Derived Using an Encoder-Decoder (left) and Augmented with Problem Features to Form the State. On this, an Accuracy and Legal Embedding Prediction Network (center) Predicts Reward, and an Actor-Critic Pair (right) Learns the Value Function and Network Modification Policy.

As shown, the system uses a 'network architecture' and a 'problem' description as its input for training and prediction. The encoder model transforms the network description into an embedding space which, with the problem description, serves as input to the 'accuracy prediction' and 'actor network' and the 'critic network'. The 'decoder model' is used to translate the embedding vector back to a corresponding network architecture for use on the problem task.

The encoder-decoder model is trained together with the accuracy prediction network as an autoencoder with the accuracy prediction serving as a fine-tuning criterion for the embedding space construction. The accuracy prediction network is trained to predict the expected accuracy of the embedded network architecture on the target problem.

The TD3-based actor-critic networks are trained using the change in expected network accuracy on the target problem as a reward function. Therefore the critic will predict achievable accuracy gain given the policy while the actor will attempt to learn a policy to navigate in the embedding space towards optimal network architectures. Use of the accuracy prediction network here allows planning-based updates which use predicted rewards rather than true rewards which would require training the modified network in the embedding space on the target task after each step of the policy. This reduces computational overhead and permits the actor-critic to be trained more effectively using (partially) simulated embedding space trajectories. This is particularly important given that the TD3 actor-critic is largely an on-policy learner and thus requires discounting of old trajectories and extensive generation of new ones during training.

Implementation and Experiments

To simplify the proof of concept implementation and evaluation, the initial experiments are focused on fully connected networks of limited size as well as classification problems from the UCI data set (Dua and Graff 2017). This permits the input network specification to be supplied in terms of layers and unit numbers per layer and provides an easy means of generating initial training data by training 300 random networks, each on, one of the 10 target problems. Since the network architecture input formulation allows for illegal inputs, an addition 300 illegal network descriptions were generated and assigned accuracy values of 0 to allow the system to learn the demarcations of the embedding space. For this initial proof of concept, target problems are trained independently, yielding problem-specific policies.

To study the performance and operation of the different architecture components, training occurred as a sequence of pre-training steps which focus on different components and results were studied before training the next component.

Embedding Space Training To validate the encoding space, we first pre-trained the encoder-decoder network together with the accuracy prediction network on the chosen UCI classification data set to form a 2-dimensional embedding space. The loss function, $\mathcal{L}_{eda}$, used for training used three components: decoder reconstruction loss, $\mathcal{L}_{rec}$, accuracy prediction loss, $\mathcal{L}_{acc}$, and illegal embedding prediction loss, $\mathcal{L}_{leg}$. The first two are measured in terms of squared error while the last uses binary cross-entropy.

$$\mathcal{L}_{eda} = \alpha_{dec}\mathcal{L}_{rec} + \alpha_{acc}\mathcal{L}_{add} + \alpha_{leg}\mathcal{L}_{leg}$$

The resulting representation of the embedding space is shown in Figure 5 where blue points indicate valid networks

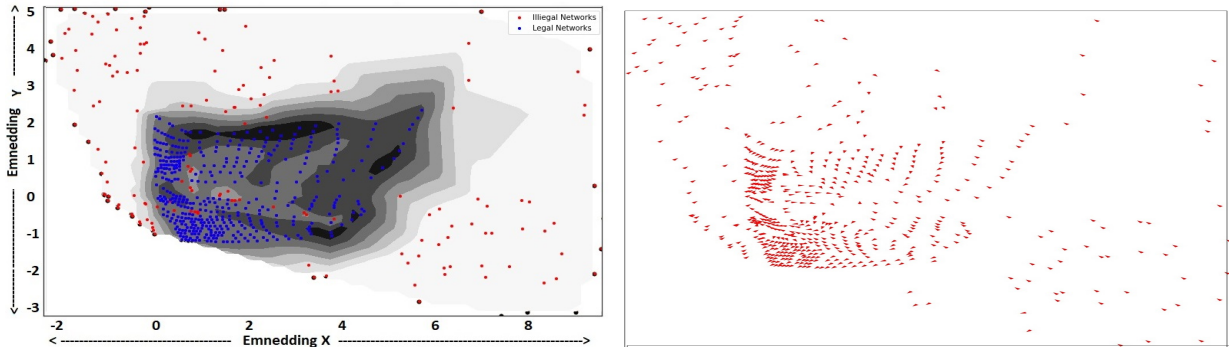

Figure 4: Critic Value (left) and Learned Policy Actions (right) over Embeddings of 2-Layer Networks

while red dots indicate parts of the embedding space that do not correspond to legal network configurations.

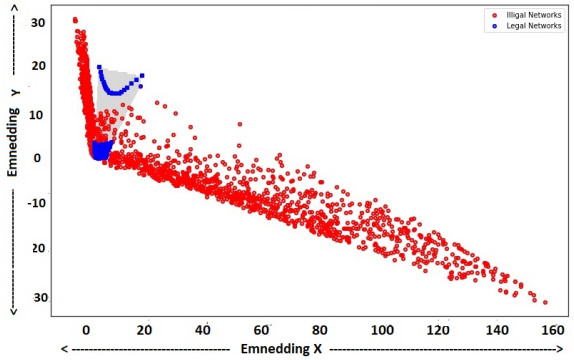

Figure 5: Embedding Space Representation for Random Legal (blue) and Illegal (red) Network Configurations

This figure shows that the encoder-decoder structure is able to form separate regions for legal networks, separating them from illegal network configurations. However, it also shows that it can yield disconnected regions which, while bridgeable by learned policies, make the task of policy learning more difficult and should be addressed in future work.

Figure 6 shows the learned accuracy for 2-Layer Networks, reflecting the left bottom blue embedding area.

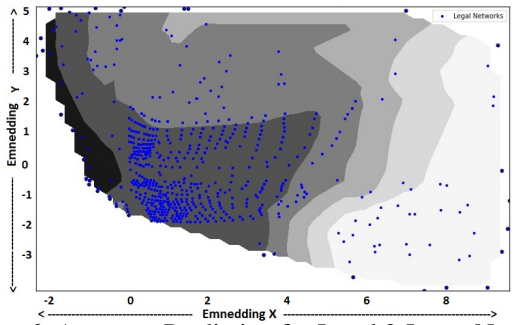

Figure 6: Accuracy Prediction for Legal 2-Layer Networks

The encoder-decoder and accuracy networks achieved reconstruction and legal prediction accuracies above 95% predicted target accuracy with a squared error below 10^{-5}.

Actor-Critic Training were used to evaluate whether RL can actually produce a policy that will optimize the network architecture for the UCI dataset used. For our initial proof of concept we are only using one problem to train our actor-critic network to help us to learn a policy for which we have already built lots of networks. This experiment will verify that RL can actually learn a policy that will reconstruct good networks that we have built before. Our ultimate goal is to take a problem for which we haven't built a lot of networks and have our policy find a new optimize Network.

Actor-Critic Pre-Training: To simplify initial actor-critic training we first pre-trained our critic with a uniform policy and $\gamma = 0$, initializing it with the reward. Then we pre-trained the actor network with this critic. In a final pre-training step we trained the critic with $\gamma = 0.9$ to achieve consistency between actor and critic.

Actor-Critic Training: After pre-training we trained actor and critic using TD3 with a large, temporally discounted replay buffer where old sample trajectories are continuously removed and replaced with new ones. For exploration, we perturb action's Gaussian noise.

Figure 4 shows on the left resulting value function for the 2-layer network region while the right shows the learned policy as a vector field. Examining the policy reveals a strategy that converts random starting networks over time to a small number of final networks which correspond closely to local maxima in the accuracy space. Thus the architecture is able to learn an embedding, a performance predictor, a critic, and an actor for the presented network optimization problem.

Conclusion

Finding good network architectures is essential for the success of deep learning systems. This paper proposes a novel deep learning framework that uses Reinforcement Learning to learn a network architecture modification policy that has the promise to allow network optimization for different problems without the need for significant costly re-training.

The proposed approach optimizes network architectures in a learned network embedding space constructed using an autoencoder network. Using this and a performance predictor, an actor-critic RL component is trained which uses the embedding vector as a network representation as well as problem specific features to learn a policy for network architecture modification over the continuous action space of embedding space moves. Initial experiments using a UCI classification dataset show that the system is able to learn an embedding as well as a policy that derives close to optimal network architectures from random starting networks.

References

Baker, Bowen, Otkrist Gupta, Nikhil Naik, et al. (2016). "Designing neural network architectures using reinforcement learning". In: *arXiv preprint arXiv:1611.02167*.

Baker, Bowen, Otkrist Gupta, Ramesh Raskar, et al. (2017). "Accelerating neural architecture search using performance prediction". In: *arXiv preprint arXiv:1705.10823*.

Bose, Sourabh and Manfred Huber (2016). "Incremental learning of neural network classifiers using reinforcement learning". In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, pp. 002097–002103.

Brock, Andrew et al. (2017). "Smash: one-shot model architecture search through hypernetworks". In: *arXiv preprint arXiv:1708.05344*.

Byrne, Donal (2019). *TD3: Learning To Run With AI*. URL: https://towardsdatascience.com/td3-learning-to-run-with-ai-40dfc512f93 (visited on 06/15/2019).

Cai, Han et al. (2017). "Reinforcement learning for architecture search by network transformation". In: *arXiv preprint arXiv:1707.04873* 4, p. 3.

Dua, Dheeru and Casey Graff (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Gaurav, Mausam (2019). *How to find the optimum number of hidden layers and nodes in a neural network model?* URL: https://datagraphi.com/blog/post/2019/12/17/how-to-find-the-optimum-number-of-hidden-layers-and-nodes-in-a-neural-network-model (visited on 12/17/2019).

Granitto, Pablo M, Pablo F Verdes, and H Alejandro Ceccatto (2005). "Neural network ensembles: evaluation of aggregation algorithms". In: *Artificial Intelligence* 163.2, pp. 139–162.

Kapanova, KG, I Dimov, and JM Sellier (2018). "A genetic approach to automatic neural network architecture optimization". In: *Neural Computing and Applications* 29.5, pp. 1481–1492.

Luo, Renqian et al. (2018). "Neural architecture optimization". In: *Advances in neural information processing systems* 31.

Rifkin, Ryan and Aldebaro Klautau (2004). "In defense of one-vs-all classification". In: *The Journal of Machine Learning Research* 5, pp. 101–141.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

TORRES, Jordi (2020). *The Bellman Equation, V-function and Q-function Explained*. URL: https://towardsdatascience.com/the-bellman-equation-59258a0d3fa7 (visited on 07/11/2020).

Yoon, Chris (2018). *Deriving Policy Gradients and Implementing REINFORCE*. URL: https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63 (visited on 12/29/2018).

– (2019). *Deep Deterministic Policy Gradients Explained*. URL: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f (visited on 02/06/2019).

Zhou, Zhi-Hua, Jianxin Wu, and Wei Tang (2002). "Ensembling neural networks: many could be better than all". In: *Artificial intelligence* 137.1-2, pp. 239–263.

Zoph, Barret and Quoc V Le (2016). "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578*.