UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Neural architecture search: a probabilistic approach

Author:
Volodymyr LUT

Supervisor:
Yuriy KHOMA

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

Declaration of Authorship

"It's inspiring to see how AI is starting to bear fruit that people can actually taste. There is still a long way to go before we are truly an AI-first world, but the more we can work to democratize access to the technology—both in terms of the tools people can use and the way we apply it—the sooner everyone will benefit."

Sundar Pichai, CEO Alphabet Inc., May 17, 2017

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Neural architecture search: a probabilistic approach

by Volodymyr LUT

Abstract

In this paper we review different approaches to use probabilistic methods in existing AutoML solutions using Reinforcement Learning. We focus on providing additional knowledge about probability distribution provided to Reinforcement Learning agents solving Neural Architecture Search tasks. Based on the results of the research we come with an agent designed to model Neural Architectures for image classification tasks.

Acknowledgements

First of all, I would like to thank my supervisor Mr. Yuriy Khoma and Mr. Vasily Ganishev for all their support, patience and knowledge shared with me during this project.

I would also like to thank all my colleagues in UCU - those who already graduated, who were working on their masters during this year, and those who left UCU - for being a bright lighthouse, which navigated me towards excellence, for showing a good example and for being a strong and supportive community.

Many thanks to the UCU team - to all those people, who build and design the educational system I am proud being a part of, to all those who inspired me, shared their vision and made my time being a masters program student an unforgettable experience. I thank everyone who works every day in the UCU - no matter whether they are responsible for brilliant coffee or brilliant courses and workshops.

Special thanks goes to my friend and business partner Serhii Chepkyi for providing a big support during my studies and also for providing a big help with visual materials for this project.

Last but definitely not least - I would like to thank my closest people and my family - without your eternal love this would not be possible. This was the hardest years in my life and I'm glad and thankful that every second I've felt that I'm not alone.

Contents

List of Figures

List of Tables

List of Abbreviations

ML	**M**achine **L**earning
AutoML	**Auto**mated **M**achine **L**earning
NAS	**N**eural **A**rchitecture **S**earch
RL	**R**einforcement **L**earning
MDP	**M**arkov **D**ecission **P**rocess
CNN	**C**onvolutional **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
UCB	**U**pper **C**onfidence **B**ound

Physical Constants

Speed of Light	$c_0 = 2.997\,924\,58 \times 10^{8}\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

For all the brave people who make it possible for millions of young Ukrainians to hold books in their hands instead of rifles and grenades.

Chapter 1

Introduction

As machine learning provides a huge variety of automation possibilities for different industries the problem of automation of ML industry itself seems natural. For decades ML engineers were pioneers in the new era of computer science research. As a result, the new industry was shaped and this industry requires automation.

AutoML is a general name of automation in routine work of ML engineers including but not limited to data preparation, feature engineering, feature extraction, neural architecture search, hyperparameters selection, etc.

ML is reshaping businesses and other aspects of everyday life worldwide. We believe that everyone would benefit from the democratization of these new tools. Having the ability to run models on portable devices, IoT chips, and other mass-market hardware we treat AutoML as a big move towards in terms of a variety of different applications created.

In recent years AutoML becomes a natural product for almost all big technological companies. Google, Amazon, Salesforce, and others are offering AutoML products that allow non-experts to create their ML solutions.

Still, existing AutoML techniques require lots of computational resources and most of the research in the field is covered by tech giants nowadays.

We are focusing on neural architecture search problems, especially on hyperparameter optimization tasks because historically this problem is solved mainly using exhaustive search techniques, such as grid search. Engineers often follow their empirical knowledge and try to guess optimal parameters to tune models.

We are using the reinforcement learning paradigm since it is performing well in solving NAS problems. RL agents can design better architectures than related hand-designed models in terms of error-rate and efficiency - see Zoph and Le, 2016.

Moreover, we believe that RL could benefit from probabilistic approaches. We are deeply inspired by DeepAR Salinas, Flunkert, and Gasthaus, 2017 used by Amazon to build forecasting models. We show that Gaussian probability distribution could be used to effectively balance the exploration and exploitation of RL agents solving NAS tasks.

Chapter 2

Background overview

2.1 History

The idea of using RL agents to build neural networks is not new, however, there are not so many research projects nowadays. Mostly the reason for this is that most of the research is held by business, and business usually is not optimistic about RL in production.

However, some good progress was made in recent years. In 2015, ResNet becomes a winner of ILSVRC 2015 in image classification, detection, and localization and winner of MS COCO 2015 detection and segmentation. This enormous network contained 152 layers optimized by a lot of professional engineers manually. This process is expensive in terms of time and resources. Image classification contests are constantly showing a growing amount of layers for best-performing networks (AlexNet, 2012 - 8 layers, GoogleNet, 2014 - 22 layers). Resnet has 1.7 million parameters. Each competition is turning researchers more and more towards automation of this work - and this is a place where NAS becomes a new trend.

Barret Zoph and Quoc Le. in Zoph and Le, 2016 used a recurrent network to generate the model descriptions of neural networks and train this RNN via RL agent to maximize the expected accuracy of the generated architectures on a validation set. This paper is one the most cited in this field and our research is heavily based on it.

In 2019, Google researchers developed a family of models, called EfficientNets, which surpass state-of-the-art accuracy with up to 10x better efficiency (smaller and faster) using AutoML - see Tan and Le, 2019.

Amazon has two AutoML products to offer - Amazon SageMaker Autopilot for the creation of the classification and regression machine learning models and Amazon DeepAR for forecasting scalar (one-dimensional) time series using RNN. This paper is also heavily based on the probabilistic approach used in DeepAR because of it's spectacular results.

This section would not be full without the paper which shares a lot in common with this project. RL agent (Q-Learning, epsilon-greedy exploration rate control, finite state space) described in the paper outperformed meta-modeling approaches for network design on image classification tasks Baker et al., 2016.

The interest to the topic becomes even hotter when the NAS benchmark and dataset was introduced by Google Research team Ying et al., 2019

A variety of methods have been proposed to perform NAS, including reinforcement learning, Bayesian optimization with a Gaussian process model, sequential model-based optimization (SMAC), evolutionary search, and gradient descent over the past few years. We see a lot of research potential in this field and we share a big passion for RL paradigm - and that's why this project exists.

2.2 Reinforcement Learning

RL is an machine learning paradigm often used when the exact mathematical model is unknown and data is unlabeled. In this project we would mainly concentrate on the Q Learning approach. We require having observable environment where software agent would be able to take actions which would lead to reward. Agent is designed in the way it should maximize reward by utilizing existing knowledge (exploitation) or exploring random actions (exploration). Algorithm needs to learn a policy which determines which action should be taken next given current state (or set of recent states). The environment agent is operating with is typically a Markov Decision Process.

2.2.1 Markov Decision Process

MDB is a generalization of mathematical framework which allows performing a sequence of actions in an environment where future states would depend on current states and yield a partly random outcomes.

In other words, Markov Decision Process is a description of set of actions agents should take to receive some reward. It is described by:

- State (or states) $\mathcal{S}$ that are fully observable by agent
- Set of actions $\mathcal{A}$ which agent could take in given state
- Transition model $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longmapsto [0, 1]$ - an effect in state accused by action taken by agent
- A reward function R which defines a reward received by agent for taking this action
- A discount factor $\gamma \in [0, 1)$ which is used to value a reward that could be received in future
- A policy $\pi : \mathcal{S} \times \mathcal{A} \longmapsto [0, 1]$ agent should learn

A policy which allows agent to maximize it's reward is called a solution of given MDP. Notation of MDP used above is taken from Thomas, 2015.

2.2.2 Exploration and exploitation dilemma

To demonstrate exploration over exploitation problem we usually often refer to Multiarmed Bandit Problem.

Multiarmed bandit problem is MDP with stochastic reward. Imagine having N machines with reward probabilities $Q_1..Q_n$. Playing k-th bandit could result into reward of 1 with a possibility of Q_k or into reward 0 with possibility of $1 - Q_k$. Real reward probabilities are unknown to the agent. Also, resources are limited, so, exploring them using law of large numbers is inefficient and impossible in terms of this problem.

The goal of agent playing multiarmed bandits is maximization of cummulative reward.

Simply saying, by playing known action (performing exploitation) agent may miss better option. This becomes a loss function of such algorithms - a regret player might by not selecting the optimal action. If the agent always plays random actions (exploration) it will be completely useless and would yield no good results. In other words, to balance exploration and exploitation problem we would need to ensure that the agent is not overfitting to known actions but still uses gained knowledge. It often happens, that RL agents observes some **good enough** action and starts to use it constantly. This is obviously not the way we generally want an RL algorithm to behave.

2.2.3 Epsilon-greedy approach

Epsilon-greedy approach to solve exploration over exploitation dilemma is based on the idea, where agent would generally take best actions, but at some time iterations t it would explore a random actions. It often assumes that at the beginning of the training number of random actions would be bigger (to ensure that agent would be able to explore good actions at the very beginning) and later it would decay (to ensure that agent is actually playing best actions). Expected reward is often represented as a mean of previously received rewards when playing this action.

2.2.4 Upper-confidence bound

UCB is another instrument to solve exploration over exploitation dilemma. General approach is similar to epsilon-greedy algorithm, however, UCB allows to use unexplored actions in favor of actions which algorithm is very certain about being bad ones.

UCB measures potential of action using upper confidence bound of the reward value in such a way that larger number of trials of certain action should give smaller bound. This also prevents algorithm from overfitting.

In this work we would refer to Theorem 1 from Auer, Cesa-Bianchi, and Fischer, 2002 using a simple UCB approach which allows to receive logarithmic regret uniformly and without any preliminary knowledge about the reward distributions.

$$U(a) = Q(a) + \sqrt{\frac{2logn_a}{N}}$$

Where Q(a) is expected reward from action a (generally a mean of rewards received by playing that action), n_a is number of times this action was played and N is total number of plays.

2.2.5 Deep Q Learning

While Q learning is a algorithm which allows agent to learn an effective policy for maximizing cumulative reward in general MDP, deep Q learning is a same, model-free approach that uses Deep Neural Network to approximate the values of expected reward.

In recent years a lot of research is done in this field, even though Deep Q learning is treated as unstable solution because a slight change in input can change outputs significantly.

However, this approach has a proven success story. This work is inspired by Mnih et al., 2013 which was one of the first successful aprroaches to use deep neural network as a backend for reinforcement learning agent.

In general Deep Q learning algorithms stores experienced rewards in replay memory, from where they are later batched and used as input to the underlying neural network. That's usually the reason why a lot of research uses RNNs with LSTM (Long-Short Term Memory) cells as a backend.

This work uses a convolutional neural network as a backend for RL algorithm.

2.3 Image classification problem

Image classification problem is a machine learning problem aiming to recognize a visual concept of given image and assign some class (label) to it as an output.

Data-driven approach for this problem required to train algorithm on loads of images in order to understand which features images from one class have in common to other images of this class and different from images of other classes.

Naturally, at the beginning KNN (K Nearest Neighbors) and other simple clusterization algorithms were used.

However, the real move forward started when backpropagation techiques were applied to neural network architectures, which opened a way for a deep learning techiques. As Yann LeCun released LeNet-5 (see LeCun et al., 1989) modern convolutional architectures started to develop rapidly.

There are several popular datasets such as MNIST, Flowers, ImageNet etc. which are constantly used in image classification contests. Modern deep neural networks outperform human-level accuracy on those datasets.

2.4 Convolutional Neural Networks

2.5 Gaussian Distribution

Chapter 3

Proposed approach

3.1 Pipeline

3.2 RL agent

3.2.1 Reward function

3.2.2 Exploration and exploitation

3.2.3 Gaussian layer

When it comes to probabilistic approach to RL algorithms, scoring becomes very important and may have a big impact on action taken by the controller - see Gneiting, Balabdaoui, and Raftery, 2007

If master CNN in RL agent is solving a regression problem (which is exactly our case) this CNN would use backpropagation to update it's weights (as noted above) in a way that error metrics of a test set would be minimized. Originally outputs of the last layer would be simple values - in our case, values which determines the actions that would be taken by controller.

Those values obviously depends on input and weights. In order to receive a Gaussian distribution we would need to modify last layer so that it would return mean and variance of output variable, which is enough to describe a Gaussian distribution of this variable. This allows us to bring a prediction uncertainity into the outputs of CNN. As described in Salinas, Flunkert, and Gasthaus, 2017 this requires also another approach to computation of loss function.

Generally saying we are using Gaussian distribution instead of single values because we need to have a measure of uncertainity of prediction of output. Then, if we have quite big uncertainity, it would be smarter to explore a random action. Otherwise, agent should use predicted action.

3.3 Master CNN

3.4 Slave CNN

Chapter 4

Experiments

4.1 Datasets

Our RL agent is generating CNN architectures which are performing an image classification task on the CIFAR-10 and CIFAR-100 datasets.

CIFAR datasets are described in very deep details in Chapter 3 of Krizhevsky, 2012, especially details about it's collection.

I would not go into details, just will note that CIFAR is a set of 32×32 colour images depicting real-world objects.

After training RL algorithm on CIFAR10 it is heavily switched to use it's knowledge on CIFAR100.

Key differences between CIFAR10 and CIFAR100 is denoted in table 4.1.

Dataset	Size	Number of classes	Images in class
CIFAR10	60000	10	6000
CIFAR100	60000	100 grouped into 20 superclasses	600

TABLE 4.1: Comparison of CIFAR10 and CIFAR100 datasets

Both in CIFAR10 classes are exclusive and do not assume instances overlapping - see 4.1.

Figure 4.1 shows a boat.

We use CIFAR datasets as is, meaning that we also share same train/test dataset split for evaluating generated CNN architectures. There are 50000 training images and 10000 test images in the dataset.

4.2 Metrics

4.3 Environment and training

4.4 Results

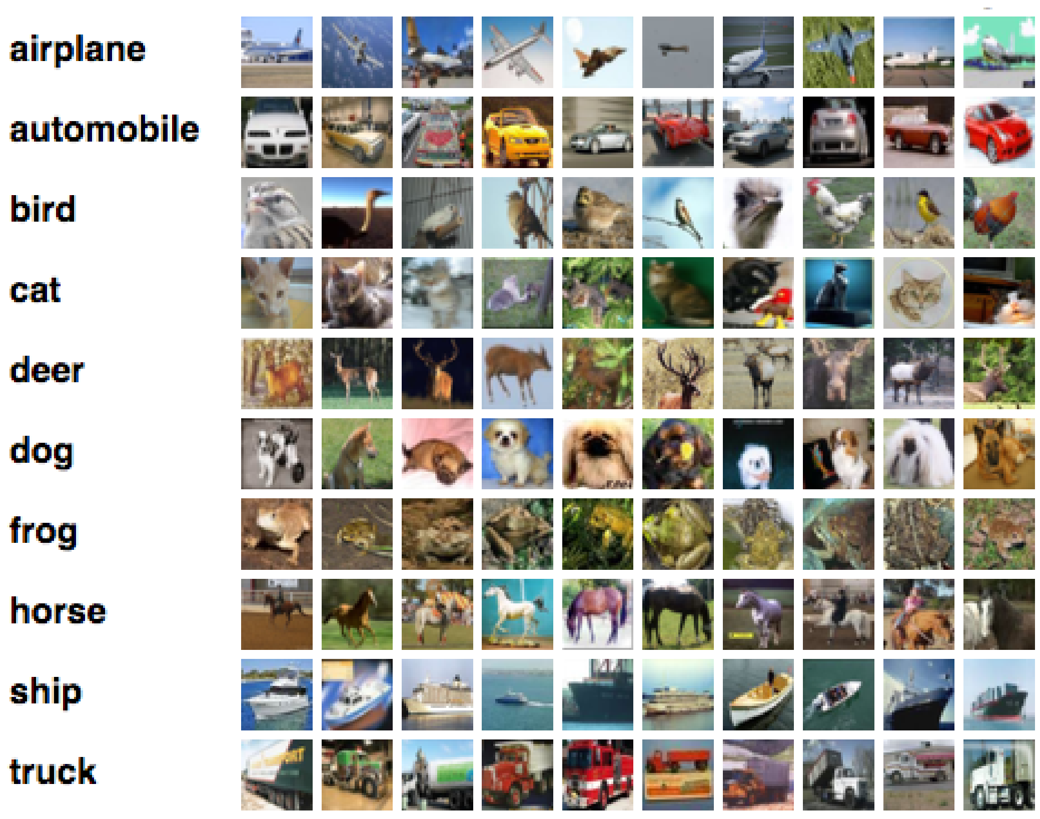

FIGURE 4.1: Classes of CIFAR10 including 10 random images from each Krizhevsky, 2012

Chapter 5

Conclusion

Bibliography

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer (2002). "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2, pp. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352. URL: https://doi.org/10.1023/A:1013689704352.

Baker, Bowen et al. (2016). "Designing Neural Network Architectures using Reinforcement Learning". In: *ArXiv* abs/1611.02167.

Gneiting, Tilmann, Fadoua Balabdaoui, and Adrian E. Raftery (2007). "Probabilistic forecasts, calibration and sharpness". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 69.2, pp. 243–268. DOI: 10.1111/j.1467-9868.2007.00587.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2007.00587.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2007.00587.x.

Krizhevsky, Alex (2012). "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto*.

LeCun, Y. et al. (1989). "Handwritten Digit Recognition: Applications of Neural Net Chips and Automatic Learning". In: *IEEE Communication*. invited paper, pp. 41–46.

Mnih, Volodymyr et al. (2013). "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

Salinas, David, Valentin Flunkert, and Jan Gasthaus (2017). "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks". In: *arXiv e-prints*, arXiv:1704.04110, arXiv:1704.04110. arXiv: 1704.04110 [cs.AI].

Tan, Mingxing and Quoc V. Le (2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *arXiv e-prints*, arXiv:1905.11946, arXiv:1905.11946. arXiv: 1905.11946 [cs.LG].

Thomas, Philip S. (2015). "A Notation for Markov Decision Processes". In: *CoRR* abs/1512.09075. arXiv: 1512.09075. URL: http://arxiv.org/abs/1512.09075.

Ying, Chris et al. (2019). "NAS-Bench-101: Towards Reproducible Neural Architecture Search". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, pp. 7105–7114. URL: http://proceedings.mlr.press/v97/ying19a.html.

Zoph, Barret and Quoc V. Le (2016). "Neural Architecture Search with Reinforcement Learning". In: *CoRR* abs/1611.01578. arXiv: 1611.01578. URL: http://arxiv.org/abs/1611.01578.