Neural Architecture Search with Bayesian Optimisation and Optimal Transport (NIPS-2019)

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, Eric Xing (CMU)

Jungtaek Kim (jtkim@postech.ac.kr)

Machine Learning Group,
Department of Computer Science and Engineering, POSTECH,
77 Cheongam-ro, Nam-gu, Pohang 37673,
Gyeongsangbuk-do, Republic of Korea

January 8, 2019

Table of Contents

Neural Architecture Search with Bayesian Optimisation and Optimal Transport

Motivation

- Neural architecture search (NAS) is a method to find a neural architecture which shows the better performance than the previous ones.
- In NAS literature, none of the methods have been designed with a focus on the expense of evaluating a neural network.
- Therefore, Bayesian optimization (BO) can be a potential way to solve NAS problem efficiently.
- But, the majority of the BO literature has focused on settings where the domain is either Euclidean or categorical.

Contributions

- A pseudo-distance for neural network architectures called `OTMANN` (optimal transport metrics for architectures of neural networks)
- A BO framework for optimizing functions on neural network architectures called `NASBOT` (neural architecture search with Bayesian optimization and optimal transport)
- `https://github.com/kirthevasank/nasbot`

Distance of Neural Networks

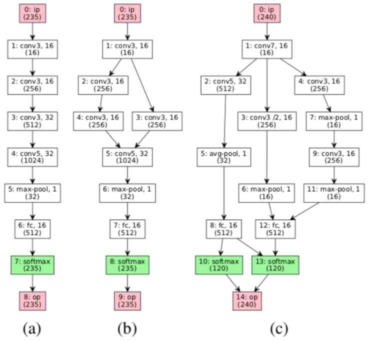


Figure 1: An illustration of some CNN architectures. In each layer, i: indexes the layer, followed by the label (e.g conv3), and then the number of units (e.g. number of filters). The input and output layers are pink while the decision (softmax) layers are green.

From Section 3: The layer mass is denoted in parentheses. The following are the normalised and unnormalised distances $d, \bar{d}$. All self distances are 0, i.e. $d(\mathcal{G},\mathcal{G}) = \bar{d}(\mathcal{G},\mathcal{G}) = 0$. Unnormalised: $d(\mathrm{a},\mathrm{b}) = 175.1$, $d(\mathrm{a},\mathrm{c}) = 1479.3$, $d(\mathrm{b},\mathrm{c}) = 1621.4$. Normalised: $\bar{d}(\mathrm{a},\mathrm{b}) = 0.0286$, $\bar{d}(\mathrm{a},\mathrm{c}) = 0.2395$, $\bar{d}(\mathrm{b},\mathrm{c}) = 0.2625$.

A Mathematical Formalism for Neural Networks

- A neural network $\mathcal{G} = (\mathcal{L}, \mathcal{E})$ is defined by a set of layers $\mathcal{L}$ and directed edges $\mathcal{E}$.
- An edge $(u, v) \in \mathcal{E}$ is an ordered pair of layers.
- A layer $u \in \mathcal{L}$ is equipped with a layer label $ll(u)$ which denotes the type of operations performed at the layer.
  - $ll(1) = \texttt{conv3}$ and $ll(5) = \texttt{max-pool}$.
- The attribute lu denotes # of computational units in a layer.
  - $lu(5) = 32$ and $lu(7) = 16$.
- Each network has decision layers which are used to obtain the predictions of the network.
  - For classification task, `softmax` and for regression task, `linear`.
- Every network has unique input and output layers u_{ip} and u_{op}.
- All layers that are not input, output, or decision layers are processing layers.

The OTMANN Distance

- The distance for NAS should satisfy to measure the amount of computation at each layer, the types of these operations, and how the layers are connected.
- OTMANN is defined as the minimum of a matching scheme which attempts to match the computation at the layers of one network to the layers of the other.
- Moreover, it needs to penalize for matching layers with different types of operations or those at structurally different positions.

The OTMANN Distance

- Layer masses: The layer masses $lm : \mathcal{L} \to \mathbb{R}_+$ will be the quantify that we match between the layers of two networks when comparing them.
    - For example, in Fig. 1b, $lm(5) = 32 \times (16 + 16)$.
    - In this paper, use $lm(u_{\mathrm{ip}}) = lm(u_{\mathrm{op}}) = \zeta \sum_{u \in \mathcal{PL}} lm(u)$ where $\mathcal{PL}$ denotes the processing layers and $\zeta \in (0, 1)$ is a hyperparameter.
    - For decision layers, $\forall u \in \mathcal{DL}, lm(u) = \frac{\zeta}{|\mathcal{DL}|} \sum_{u \in \mathcal{PL}} lm(u)$.

The OTMANN Distance

- Path lengths from/to $u_{\mathrm{ip}}/u_{\mathrm{op}}$: In a neural network $\mathcal{G}$, a path from u to v is a sequence of layers $u_1, \ldots, u_s$ where $u_1 = u$, $u_s = v$ and $(u_i, u_{i+1}) \in \mathcal{E}$ for all $i \leq s - 1$.
- There are the shortest, longest, and random walk path lengths $\delta^{\mathrm{sp}}(u)$, $\delta^{\mathrm{lp}}(u)$, and $\delta^{\mathrm{rw}}(u)$.
- For any $s \in \{\mathrm{sp}, \mathrm{lp}, \mathrm{rw}\}$ and $t \in \{\mathrm{ip}, \mathrm{op}\}$, $\delta_t^s(u)$ can be computed for all $u \in \mathcal{L}$, in $\mathcal{O}(|\mathcal{E}|)$ time.

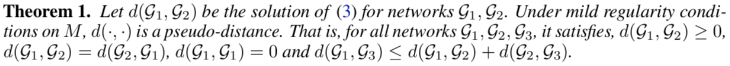
The OTMANN Distance

- Given two networks $\mathcal{G}_1 = (\mathcal{L}_1, \mathcal{E}_1)$, $\mathcal{G}_2 = (\mathcal{L}_2, \mathcal{E}_2)$ with n_1, n_2 layers respectively, we will attempt to match the layer masses in both networks.
- We let $Z \in \mathbb{R}_+^{n_1 \times n_2}$ be such that $Z(i, j)$ denotes the amount of mass matched between layer $i \in \mathcal{G}_1$ and $j \in \mathcal{G}_2$.
- The `OTMANN` distance is computed by solving the following optimization problem:

$$\arg\min_Z \phi_{\mathrm{lmm}}(Z) + \phi_{\mathrm{nas}}(Z) + \nu_{\mathrm{str}}\phi_{\mathrm{str}}(Z)$$
$$\text{subject to} \sum_{j \in \mathcal{L}_2} Z_{ij} \leq lm(i), \sum_{i \in \mathcal{L}_1} Z_{ij} \leq lm(j), \forall i, j$$

where ϕ_{lmm} is label mismatch penalty, ϕ_{nas} is non-assignment penalty, and ϕ_{str} is structural penalty.

The OTMANN Distance


Theorem 1. *Let* $d(\mathcal{G}_1, \mathcal{G}_2)$ *be the solution of* (3) *for networks* $\mathcal{G}_1, \mathcal{G}_2$. *Under mild regularity conditions on* M, $d(\cdot, \cdot)$ *is a pseudo-distance. That is, for all networks* $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$, *it satisfies,* $d(\mathcal{G}_1, \mathcal{G}_2) \geq 0$, $d(\mathcal{G}_1, \mathcal{G}_2) = d(\mathcal{G}_2, \mathcal{G}_1)$, $d(\mathcal{G}_1, \mathcal{G}_1) = 0$ *and* $d(\mathcal{G}_1, \mathcal{G}_3) \leq d(\mathcal{G}_1, \mathcal{G}_2) + d(\mathcal{G}_2, \mathcal{G}_3)$.

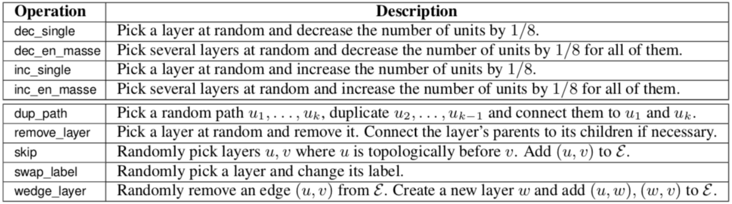
NASBOT

- In this paper, a negative exponentiated distance for κ is used.
- Precisely, $\kappa = \alpha e^{-\beta d} + \bar{\alpha} d^{-\beta \bar{d}}$ where $d, \bar{d}$ are the OTMANN distance and its normalized version.
- In this paper, "We mention that while this has the form of popular kernels, we do not know yet if it is in fact a kernel. In our several experiments, we did not encounter an instance where the eigenvalues of the kernel matrix were negative."
- We use an evolutionary algorithm (EA) approach to optimize the acquisition function.

NASBOT

1. Start from an initial pool of networks and evaluate the acquisition φ_t on those networks.
2. Generate a set of N_{mut} mutations of this pool:
    - Stochastically select N_{mut} candidates from the set of networks already evaluated such that those with higher φ_t values are more likely to be selected than those with lower values.
    - Modify each candidate with the several fixed rules, to produce a new architecture.
3. Evaluate the acquisition on this N_{mut} mutations.
4. Add them to the initial pool.

NASBOT



Operation	Description
dec_single	Pick a layer at random and decrease the number of units by $1/8$.
dec_en_masse	Pick several layers at random and decrease the number of units by $1/8$ for all of them.
inc_single	Pick a layer at random and increase the number of units by $1/8$.
inc_en_masse	Pick several layers at random and increase the number of units by $1/8$ for all of them.
dup_path	Pick a random path $u_1, \dots, u_k$, duplicate $u_2, \dots, u_{k-1}$ and connect them to u_1 and u_k.
remove_layer	Pick a layer at random and remove it. Connect the layer's parents to its children if necessary.
skip	Randomly pick layers u, v where u is topologically before v. Add (u, v) to $\mathcal{E}$.
swap_label	Randomly pick a layer and change its label.
wedge_layer	Randomly remove an edge (u, v) from $\mathcal{E}$. Create a new layer w and add $(u, w), (w, v)$ to $\mathcal{E}$.

Table 2: Descriptions of modifiers to transform one network to another. The first four change the number of units in the layers but do not change the architecture, while the last five change the architecture.

Experiments

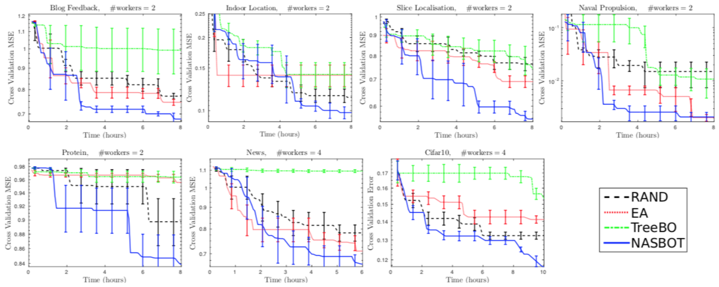


Figure 2: *Cross validation results:* In all figures, the x axis is time. The y axis is the mean squared error (MSE) in the first 6 figures and the classification error in the last. Lower is better in all cases. The title of each figure states the dataset and the number of parallel workers (GPUs). All figures were averaged over at least 5 independent runs of each method. Error bars indicate one standard error.