Learning-Theoretic Foundations of Neural Architecture Selection
Meeting Notes – William H. Guss – 12-19-2017

Existing Approach for Algorithm Selection Theory

Algorithm selection theory essentially attempts to given sample complexity and computational efficiency guarantees to algorithms which learn different algorithmic configurations given a specific application domain. For example, one might ask what sample complexity yields an ϵ-optimal architecture in neural architecture search (Zoph and Le, 2017)? The traditional tools of statistical learning theory aren't equipped to answer these questions because algorithms which are seemingly "close" in algorithm-space can have drastically different behavior.

The basis of algorithm selection theory aims to solve the following problem formally. Recall a basic definition of Gupta and Roughgarden. Let Π be a set of "problem instances."

Definition 1. *A learning algorithm L (ϵ, δ)-learns the algorithm class $\mathcal{A}$ with respect to the cost function* `cost` *if, for every distribution $\mathcal{D}$ over Π, with probability at least $1 - \delta$ over the choice of a sample $\mathcal{S} \sim \mathcal{D}^m$, L outputs an algorithm $\hat{h} \in \mathcal{A}$ such that*

$$\mathbb{E}_{x \in \mathcal{D}} \left[cost\left(\hat{h}, x\right) \right] - \min_{h \in \mathcal{A}} \{ \mathbb{E}_{x \in \mathcal{D}} [cost(h, x)] \} < \epsilon.$$

We require that the number of samples be polynomial in n, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, where n is an upper bound on the size of the problem instances in the support of $\mathcal{D}$.

Additionally, we want to find an algorithm that (ϵ, δ)-learns an algorithm class with computational efficiency (polynomial time in $n, 1/\delta, 1/\epsilon$.) For infinite algorithm classes, this is a difficult theoretical problem, as naive approaches yield algorithms with exponential or potentially infinite running time.

In (Balcan et al. 2017), a central approach to finding such algorithms is in observing that although parameterizations of algorithm-space are generally not regular (continuous, or smooth) with respect to the performance of those algorithms, a deeper analysis of algorithmic structure often results in those algorithms being piece-wise regular. In this scenario, the traditional tools of learning theory can be used on those individual pieces and integrated in a finite (although potentially combinatorial way). To understand the methodology, we turn to a concrete example.

Specifically the selection of *rounding* functions in the Random Projection, Randomized Rounding algorithms for solving the graph max-cut problem is studied by an analysis of a particular single-parameter family of rounding functions:

1. A family of these functions $\Phi = \{\phi_s \mid s \in \mathbb{R}_{\geq 0}\}$ is shown to be piecewise quadratic with respect to `cost`, for a fixed input. Note that $\phi_s \in \Phi$ induces an algorithm $A_s \in \mathcal{A}$.

2. The pseudo-dimension (think VC-dimension) of the induced hypothesis class $\mathcal{H}_A$ is shown to be $\Theta(\log n)$–This enables us to apply the standard learning theory bounds of (Anthony and Bartlett (2009)).

3. Show computational efficiency of some algorithm selection meta-algorithm as a result of the piecewise regularity of $\mathcal{A}$. Furthermore, it is shown that the algorithm achieves optimality on the sample.

4. Given sample optimality and finite pseudo-dimension, we can then bound the expected performance on new "problem instances" drawn from $\mathcal{D}$ using normal learning theory bounds.

A similar analysis is given in the case of agglomerative clustering, again with a real valued parameterization of an algorithm space.

Application to Neural Architecture Selection

In neural architecture selection, given some training data $\mathcal{D}$ we would like to chose an architecture which is suitably expressive and regularized enough to generalize "optimally" to the training data. A framing in the the context of algorithm configuration theory is as follows.

- **The algorithm class.** A neural architecture G is a *modular* configuration of a learning algorithm $\texttt{NNlearn}_G(\mathcal{D})$ which executes stochastic gradient descent on the data set D initializing a neural network instantiated from G.

  This algorithm class potentially has several variants:

  - *Single step of gradient descent.* We could take $\texttt{NNlearn}_G$ to only make one gradient descent step with respect to the whole dataset $\mathcal{D}$ or a batch, depending on our notion of "problem" instance.
  - *Assume algorithm learns.* For the sake of simplicity, we may as well remove any particularity of learning; that is, assume that $\texttt{NNlearn}_G(\mathcal{D})$ achieves the a "good-enough" local optima in cost with respect to its parameters.
  - *Gradient descent until stopping condition.*

- **The goal.** We would then find a computationally efficient meta-algorithm which (ϵ, δ)-learns the algorithm class $\mathcal{A} = \{\texttt{NNlearn}_G \mid G \in \mathcal{G}\}$ where $\mathcal{G}$ is some candidate architecture space. Note the parameterization of G is critical!

- **The problem instances.** In neural architecture selection/search methods, we consider the simple setting where we have access only to one dataset. Furthermore there is an extremely large cost to executing $\texttt{NNlearn}_G$ on large datasets with G sufficiently powerful. Therefore problem instances must be carefully selected.

  - *Problem instances as datasets in a particular application domain.*
  - *Problem instances as batches.* If batches are seen as the problem instance, the meta-algorithm would find an architecture G and thereby $\texttt{NNlearn}_G$ which performs most optimally on each batch in expectation. This would simplify computation of the relationship between G and `cost`.

    This might also be interesting in the context of Nina's work on Private/Online Optimization of Piecewise Lipschitz Functions; specifically, online architecture selection.

- **The parameterization of $\mathcal{G}$.** Central to the computational efficiency of the algorithms given in Nina's analysis is the parameterization of certain algorithm configurations by a real-valued family which is piecewise regular with respect to `cost`.

  In neural architecture selection, subspaces of $\mathcal{G}$ are largely discrete. For example, the number of neurons on a layer, the depth of an architecture, or more generally the graph topology of

a particular architecture. To retain the effectiveness of current techniques, such a parameterization is essential.

- *Continuous parameterization of expressivity* (see (Guss, 2016)).
- *Continuous interpolation between networks with different numbers of neurons.* (Dropout-esque, fuzzy sets, etc).
- *Grouping of architectures according to a more course measure of performance.* This could be topological phase transitions. We could also choose an exponential parameterization, ie. # neurons $= 2^g, g \in \mathbb{R}$.

Regardless of choice of parameterization, it is important to find one that requires that we need not re-run the optimization too many times.

Other Notes.

- **Compositionality.** In Algorithm 2 of (Balcan et al. 2017), the multiple steps of max-cut provide a particular advantage. The computation of the SPD embedding for each of the problem instances need only be done once in a first step, then the selection of s mereley effects a second step.

  This compositional configurability presents both a challenge and a basis for the transposition of these ideas to neural architecture search: on one hand, $\texttt{NNlearn}_G$, is merely a single-step process with respect to the chosen G, as we change G we need re-run $\texttt{NNlearn}_G$; this could be a costly process. On the other hand, neural architecture search is in essence compositional; usually we determine the number of neurons on each layer one layer at a time. Therefore the right meta-learning algorithm might select the optimal architecture with respect to a sample compositionally: first compute the number of neurons on the first layer, then compute the number of neurons on the second layer, then ..., stop.