Collaborative Learning and Optimization

Presenter: Ke Tang* and Kai Qin†

*USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications (UBRI)
School of Computer Science and Technology
University of Science and Technology of China, Hefei, China

†School of Computer Science and Information Technology
RMIT University, Melbourne, Australia

November 11, 2014
Outline

- Learning and optimization: fundamentals and relationships
- Collaborative learning and optimization (COLO)
- Part I: Optimization for learning
  - Why? What? How?
  - Examples in detail
- Part II: Learning for optimization
  - Why? What? How?
  - Examples in detail
- Emerging challenges in COLO
- Promising research initiatives
Learning and optimization: fundamentals

- **Learning (machine learning)**
  - Machine learning deals with the construction and study of algorithms that can learn from data. Such algorithms operate by building a model from inputs and using that to make predictions or decisions.
  - **Key aspects:** generalization and adaptation
  - Supervised, unsupervised, and reinforcement learning

- **Optimization (search-based optimization)**
  - Search-based optimization iteratively searches for the best decision variables in regard to some criteria (i.e., minimization or maximization of certain objective functions) given a defined domain and/or a set of constraints.
  - **Key aspects:** accuracy and efficiency
  - Our focus: black-box metaheuristic optimization
Learning and optimization are distinct but closely related.

Learning involves optimization of

- structure and parameters of a specific learning model
- desirable data properties in regard to a specific learning task

Optimization involves learning of

- desirable search direction, range and strength
- desirable search operation and parameters
Collaborative learning and optimization

- Traditional hybridization of learning and optimization performs like simple plug-ins.

- Collaborative learning and optimization demands the synergy of learning and optimization.
  - To identify and formulate various crucial optimization tasks in learning, and leverage optimization approaches to address these tasks.
    - Example: metaheuristic optimization of non-convex generalization-driven criteria in learning
  - To identify and formulate various crucial learning tasks in optimization, and leverage learning methods to address these tasks.
    - Example: reinforcement learning of the most effective search operation in the course of search
  - To identify and formulate real-world problems that require the seamless integration of learning and optimization, and develop novel techniques to address these problems.
Part I: Optimization for learning

- Why bother optimization in learning?
- What to optimize in learning?
- How to optimize in learning (two examples)?
  - Evolving artificial neural networks
  - Learning with uncertain misclassification costs
Why bother optimization in learning?

- Many optimization tasks in learning can be challenging because they are:
  - Non-convex
    - Example: receiver operating characteristics (ROC) based measures are usually complicated non-convex functions.
  - Large-scale
    - Example: an artificial neural network may involve many hidden layers with a large number of hidden neurons per layer.
  - Dynamic
    - Example: data stream mining needs the update of models dynamically.
What to optimize in learning?

- Generalization performance, which
  - might have slight different definitions in different scenarios.
  - could only be estimated, but not measured precisely.
  - is related to model complexity.

- Any optimization task in learning should be formulated and solved to improve the generalization performance in some contexts.
Evolving artificial neural networks

- EPNet (Yao and Liu, 1997): Optimization of architecture and weights of artificial neural networks

Evolving artificial neural networks

- **A straightforward approach:**
  Encode both architecture and weights information in individuals and optimize/evolve them simultaneously.

- However, encode weights and architecture together will make the evaluation of NN architecture even more noisy, as the true quality of an architecture depends on the optimal weights for it.

- More intuitively, architectures should undergo less changes than weights so as to evaluate the former more thoroughly/accurately.
Evolving artificial neural networks

- Training with gradient-based methods (e.g., back propagation)
- Training with a less greedy method (e.g., simulated annealing)
- Modifying the architecture (optimization of architecture)
Learning with uncertain misclassification costs

- Many real-world classification problems are cost-sensitive.
- A classifier with small total misclassification cost is required in such scenarios.
- In practice, the misclassification cost might be uncertain or even unknown. That is, we need a classifier that could generalize well (or easily adapt) to potentially different misclassification costs.
- Optimizing the ROC performance plays an important role for such scenarios.
Learning with uncertain misclassification costs

- The Convex Hull of the ROC curve represents the “generalizability” of a classifier in this case.

- Under any target cost, the best classifier must be a vertex or on the edge of the convex hull of all classifiers.
Learning with uncertain misclassification costs

- When seeking a classifier with maximum ROCCH, we seek a set of classifiers.

- ROCCH maximization
  - A set-oriented optimization problem
  - Can only be tackled with heuristic approaches
  - Metaheuristic optimization seems to provide a natural way to search for the desired classifier set.
Learning with uncertain misclassification costs

- A straightforward approach: An off-the-shelf multi-objective evolutionary algorithm?
- Maximize true positive rate (TP)
- Minimize false positive rate (FP)
Learning with uncertain misclassification costs

- How about trying some most popular MOEAs, e.g., NSGA-II?
- Direct application of NSGA-II (or any other MOEA) might not perform well because:
  - A Pareto optimal solution is not necessarily on the convex hull.
  - The objective space of the problem is essentially discrete (may cause redundant solutions).
A modified approach: Convex Hull-based MOEA (CH-MOEA)

New features of CH-MOEA:

- A new sorting scheme dedicated to ROCCH maximization.
- Redundancy elimination

Learning with uncertain misclassification costs

- New sorting scheme

**Algorithm 2** \(\text{DeltaArea} (T)\)

Require: \( T \neq \emptyset \)

1: \( T \) is a solution set

Ensure: \( \text{DeltaArea} \)

2: \( Q = T \)

3: \( m = \text{sizeof}(Q) \)

4: \( E \) is performance of \( Q \)

5: \( \text{DeltaH}_1, \ldots, \text{DeltaH}_m \leftarrow 0 \)

6: if \( m < 3 \) then

7: Set \( \text{DeltaH}_1, \ldots, \text{DeltaH}_m \leftarrow \infty \)

8: else

9: Set \( \text{DeltaH}_1, \text{DeltaH}_m \leftarrow \infty \)

10: for \( 2 \leq i \leq \text{sizeof}(Q) - 1 \) do

11: \( \text{DeltaH}_i = 0.5 \times \det((E_i-E_{i-1}) \circ (E_{i+1}-E_{i-1})) \)

12: end for

13: while \( \text{sizeof}(Q) > 2 \) do

14: \( r \leftarrow \text{argmin}\{\text{DeltaH}\} \)

15: \( Q \leftarrow Q \setminus \{Q_r\} \)

16: Update(\(\text{DeltaH}_{r-1}, \text{DeltaH}_{r+1}\))

17: end while

18: end if

19: Return (\(\text{DeltaH}\))

Point \( x \) has vector \( X \)
Point \( l \) has vector \( L \)
Point \( u \) has vector \( U \)

The contribution of \( x \) to its current ROCCH front A

\[
0.5 \times \det((X-L) \cdot (U-L)) = \Delta H(X, A)
\]
Learning with uncertain misclassification costs

- Redundancy Elimination
Part II: Learning for optimization

- Why bother learning in optimization?
- What to learn in optimization?
- How to learn in optimization?

- Two examples in detail
  - Self-adaptive differential evolution (SaDE)
  - Classification and regression assisted differential evolution (CRADE)
Why bother learning in optimization?

- **Issues in optimization**
  - Too many available algorithms and too many operators and parameters
  - Highly expensive objective functions in complex engineering problems
  - Analysis and categorization of optimizers and problems

- **Optimization is intrinsically a data-driven process.**
  - Generate-and-test procedure produces a lot of data, containing valuable information which, if learnt appropriately, can much facilitate optimization.

- **Benefits from learning to optimization**
  - Learning may help adapt algorithms, operators and parameters for varying search stages and various search problems.
  - Learning from a set of truly evaluated data may provide a low-cost model (surrogate) in lieu of expensive function evaluations.
  - Learning from the performance data of different optimizers on different optimization problems may help categorize optimizers and problems.
What to learn in optimization?

- Learning can be performed from different aspects:
  - To approximate the landscape of the search space
  - To reveal certain properties of the landscape of the search space
    - Topological properties (e.g., attractive basins)
    - Ranking properties
  - To predict the efficacy of an algorithm, an operator or a parameter setting as search proceeds
  - To categorize optimizers as per their optimization capabilities or to categorize problems as per their optimization difficulties
How to learn in optimization?

- **Based on data generated in the course of search**
  - **Supervised learning**
    - To infer a prediction function based on a set of labeled data instances with performance measure defined as per prediction error
    - Example: surrogate modelling (regression, ranking and classification)

- **Unsupervised learning**
  - To infer a grouping function based on a set of unlabeled data instances with performance measure defined as per cluster homogeneity
  - Example: speciation, algorithm (or problem) categorization

- **Reinforcement learning**
  - To infer a decision-making function from the course of interaction with the environment, taking as inputs intermediate observations of the environment with performance measure defined as per reward related to each decision
  - Example: self-adaption of algorithms, operators and parameters
Example 1: SaDE

- Proposed by Qin et al. in 2009
  

- Basic idea: using the concepts of reinforcement learning to adapt search strategy and its associated parameters on the fly.

  ![Diagram of the reinforcement learning process](Courtesy of Richard S. Sutton)

  **Actions:** search strategies (and their parameters)
  **Rewards:** success rate in generating promising solutions

- SaDE can relieve the practical difficulty faced by DE in selecting among many candidates the most effective search strategy and its associated parameters.

- SaDE has demonstrated outstanding performance compared to many state-of-the-art DE variants and evolutionary algorithms.
Differential Evolution (DE)

- Highly effective for continuous black-box optimization.

**Major characteristics:** population-based stochastic search, floating-point encoding, differential mutation, simple, powerful, and robust.

Secretes behind DE: differential mutation scheme and the capability of exploiting contour matching (i.e., population members tend to distribute along functional level sets).

**Feats:** amongst top performers in various optimization competitions held at the IEEE Congress on Evolutionary Computation, e.g., CEC-2005 single-objective, CEC-2007 and CEC-2009 multi-objective, CEC-2008 large-scale continuous optimization, etc.
Illustration of “DE/rand/1/bin” in the 2D search space
Illustration of “DE/rand/1/bin” in the 2D search space
Illustration of “DE/rand/1/bin”

Four operating vectors in the 2D search space
Illustration of “DE/rand/1/bin”

Mutation operation:

\[ v_{i,g} = x_{r0,g} + F \cdot (x_{r1,g} - x_{r2,g}) \]

\( F \): mutation scale factor
Illustration of “DE/rand/1/bin”

Crossover operation:

\[ u_{i,g} = \{ u^1_{i,g}, u^2_{i,g} \} \]

\[ u_{i,g}^j = \begin{cases} 
  v_{i,g}^j & \text{if } (\text{rand} \{0, 1\} \leq CR) \text{ or } j = j_{\text{rand}} \\
  x_{i,g}^j & \text{otherwise} \end{cases} \quad j = 1, 2 \]

**CR:** crossover rate
Illustration of “DE/rand/1/bin”

Replacement operation:

\[
x_{i,g+1} = \begin{cases} 
    u_{i,g} & \text{if } f(u_{i,g}) \geq f(x_{i,g}) \\
    x_{i,g} & \text{otherwise}
\end{cases}
\]
Illustration of “DE/rand/1/bin”

Illustration of classic DE in the 2D search space
Issues in DE

- DE’s performance closely depends on its parameter settings: population size ($NP$), mutation scale factor ($F$) and crossover rate ($CR$).

- Trial-and-error scheme may identify suitable parameter settings at the expense of demanding computational costs.

- It is hard to find out one unique configuration to solve all problems.

- DE’s performance also depends on its search strategies: DE/rand/1/bin, DE/best/1/bin, etc.

- An effective search strategy needs to be armed with suitable parameter settings.

- One single search strategy equipped with the best calibrated parameter setting may not guarantee consistent effectiveness at different search stages.
Self-adaptive Differential Evolution (SaDE)

- Gradually adjusting the employed trial vector generation strategy and its associated parameter setting to adapt to different sub-regions of the search space explored at varying search stages.

- Online learning of preceding behaviors of already applied search strategies and their associated parameter settings.

- **Characteristics:** a pool of elite yet complementary strategies, two control parameters, i.e., population size ($NP$) and learning period ($LP$)

- **Feats:** high citation counts (Google Scholar citations: 849; SCI citations: 499), ranked the 5th among all TEVC papers published within the past 10 years; winner of the 2012 IEEE TEVC outstanding paper awards.
Mechanism of SaDE

**Strategy adaptation:** for each target vector, we select one strategy from the pool, with selection probabilities as per the success history of each strategy in the pool for generating promising trial vectors within a certain number of preceding generations (i.e., learning period), to generate its trial vector.

**Four strategies in the pool:**

- **DE/rand/1/bin:**
  \[ V_{i,G} = X_{r_1,G} + F \cdot (X_{r_2,G} - X_{r_3,G}) \]

- **DE/rand/2/bin:**
  \[ V_{i,G} = X_{r_1,G} + F \cdot (X_{r_2,G} - X_{r_3,G} + X_{r_4,G} - X_{r_5,G}) \]

- **DE/current-to-best/2/bin:**
  \[ V_{i,G} = X_{i,G} + F \cdot (X_{\text{best},G} - X_{i,G}) + F \cdot (X_{r_1,G} - X_{r_2,G} + X_{r_3,G} - X_{r_4,G}) \]

- **DE/current-to-rand/1:**
  \[ V_{i,G} = X_{i,G} + \text{rand}_u(0,1) \cdot (X_{r_1,G} - X_{i,G}) + F \cdot (X_{r_2,G} - X_{r_3,G}) \]

**Selection probabilities** \( p_{k,G} (k=1,..,4) \) are updated by:

\[
p_{k,G} = \begin{cases} 
  1/4, & \text{if } G \leq LP \\
  \frac{S_{k,G}}{\sum_{k=1}^{K} S_{k,G}} & \text{otherwise}
\end{cases}
\]

with \( S_{k,G} = \frac{\sum_{g=\text{LP}}^{G-1} n_{sf_{k,g}}}{\sum_{g=\text{LP}}^{G-1} n_{sf_{k,g}} + \sum_{g=\text{LP}}^{G-1} n_{f_{k,g}}} \) + \( \varepsilon \).
Mechanism of SaDE

**Parameter adaptation:** we manually specify $NP$, and adaptively adjust $CR$ while randomizing $F$ for any selected strategy.

- $CR$: archiving $CR$ values of all "succeeding" strategies within the preceding $LP$ generations. The mean of those achieved $CR$ values with respect to each strategy is used as the mean of a normal distribution with the standard deviation of 0.1 to generate $CR$ values for the corresponding strategy in the current generation.

- $F$: randomizing $F$ using a normal distribution with the mean of 0.5 and the standard deviation of 0.3, resulting in $F$ almost falling into $[-0.4, 1.4]$.

- $NP$: manually specifying $NP$ according to the available knowledge about the problem (e.g., the problem dimension) and the computational budget.

We let initial $LP$ generations to accumulate sufficient searching information: strategy selection probabilities are set to be equal, and the mean value of the normal distribution for generating $CR$ is set to 0.5.
Example 2: CRADE

- Proposed by Lu and Tang in 2012


- Basic idea: use classification and regression models as surrogates to reduce the time-consuming function evaluations in DE.

- CRADE has demonstrated higher computation efficacy and accuracy over existing surrogate-assisted DE variants, such as ranking-based surrogate DE and regression-based surrogate DE.
Classification Assisted DE (CADE)

- Replacement operation in DE:
  \[
  x_{i,g+1} = \begin{cases} 
  u_{i,g} & \text{if } f(u_{i,g}) \geq f(x_{i,g}) \\
  x_{i,g} & \text{otherwise}
  \end{cases}
  \]
  Trial vector: \( u_{i,g} \)
  Target vector: \( x_{i,g} \)

- Trial vector competes with target vector with the worse discarded.

- This is a **binary classification** problem: +1 (not worse) and -1 (worse).

- A surrogate model can be built in the neighborhood of a trial vector to solve this classification problem.

- A proportion of function evaluations can be saved per generation.

- Classification-based surrogates are simpler than regression-based and ranking-based ones.
Two stages in CADE

- Database building stage (for the first few generations)
  - Original DE
    - All trial vectors will be truly evaluated.

- Surrogate modelling and database updating stage
  - Surrogate-assisted replacement operation in DE
    - Neighbourhood determination
    - Support vector machine (SVM) based surrogate modelling
    - Trial vector classification using this surrogate
    - Trial vector evaluation on those with positive classification outputs
    - Replacement and database updating
Classification and Regression Assisted DE (CARDE)

- In CADE, winning trail vectors will be truly evaluated (time-consuming).

- A regression-based surrogate can approximate the objective function to further reduce computational cost.

- Classification and regression assisted DE (CRADE) was proposed, which synergizes classification-based and regression-based surrogates for the replacement operation in DE.
  - The classification-based surrogate first checks whether a trial vector is not worse than its corresponding target vector.
  - If so, it will be further evaluated by the regression-based surrogate.
  - If this surrogate’s output also confirms trial vector is no worse than target vector, this trial vector will take on this approximate value.
  - Otherwise, this trial vector is truly evaluated.
Emerging challenges in COLO

Seamless integration of learning and optimization

From Sim Lim Square to Singapore Sports Hub
COLO was co-founded by Ke Tang and Kai Qin in 2011.

It aims at providing a forum for academic and industrial researchers from both learning and optimization communities to collaboratively explore promising directions on the synergy of techniques from these two fields.
Thank You

Q & A