Two aspects of learning algorithms: generalization under shifts and loss functions

Stefanie Jegelka
MIT

based on joint work with
Keyulu Xu, Ching-Yao Chuang, Nikolaos Karalias, Joshua Robinson, Andreas Loukas, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi
Learning and Algorithms

- **Predicting good configurations / solvers**  

- **Predicting problem parameters**  

- **Machine learning oracles within a fixed algorithm**  
  (online algorithms, branch-and-bound,..)  

- **Learning a full algorithm, Algorithm implemented as a neural network**  

Figures: Bengio, Lodi, Provoust 2020
Today:
Some thoughts on learning (for) algorithms

Disclaimer

• Not the “latest & greatest” here in terms of experiments…
• … but some ideas to better understand ML models’ behavior
How to do learning for algorithms?

\[
\min_x c^T x \\
Ax \leq b \\
l \leq x \leq u \\
x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}
\]

Focus here:
Input is a graph with attributes

(Gasse et al 2019)
(Selsam et al 2018)
Setup: learning task

“graph regression”

- **Input** $x$: graph (or set) with attributes (optimization instance)
- **Desired output** $g(x)$: optimal value
- **Training data**: $\{(x^{(i)}, g(x^{(i)}))\}_{i=1}^N$ with $x^{(i)} \sim P$ $\Rightarrow$ $P$ unknown
- **Goal**: find a function $f \in \mathcal{F}$ that generalizes (= low “risk”)

\[
\mathbb{E}_{x \sim P} [ \ell(f(x), g(x))] 
\]

Other aspects:
- learning setting
- loss
- constraints
Learning

Training data:

\[
\min_{f \in F} \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i), g(x_i))
\]

\[\hat{f}(\cdot) = \text{predicted label}\]
Learning

Training data:

Test data:

Evaluation: generalization

\[ \mathbb{E}_{x \sim P} [ \ell(f(x), g(x)) ] \]
Outline

- Graph Neural Networks

- To what kinds of instances will my model generalize?
  Prediction under distribution shifts
  - **stability**: measuring data shifts appropriately
  - **large shifts**: understanding model behavior by decomposition

- Beyond regression: extending set functions as loss functions for neural networks
(Message passing) Graph neural networks

Input: graph $G$ with node attributes $x_v \in \mathbb{R}^{d_0}$, edge attributes $w(u,v) \in \mathbb{R}^{d'}$

$f_\theta(G) = f_{\text{Read}}(\{h_v \mid v \in V\}) 
\in \mathbb{R}^{d_{\text{out}}}$

**Idea:**
1. Encode each node (node’s neighborhood): *node embedding*
2. Aggregate set of node embeddings into a *graph embedding*
Node embedding: message passing

In each round $k$:

**Aggregate** over neighbors

$$m^{(k)}_{\mathcal{N}(v)} = \text{AGGREGATE}^{(k)}\left(\{h^{(k-1)}_u : u \in \mathcal{N}(v)\}\right)$$

**Update:** Combine with current node

$$h^{(k)}_v = \text{COMBINE}^{(k)}\left(h^{(k-1)}_v, m^{(k)}_{\mathcal{N}(v)}\right)$$
Node embedding: message passing
Message passing unrolled

In each round $k$:

**Aggregate** over neighbors and update representation

“Unrolled”: computation tree

**Structured arrangement of learnable “modules”**
Outline

• Graph Neural Networks

• To what kinds of instances will my model generalize?
  Prediction under distribution shifts
  • stability: measuring data shifts appropriately
  • large shifts: understanding model behavior by decomposition

• Beyond regression: extending set functions as loss functions for neural networks
Graph predictions and distribution shifts

\[ \mathbb{E}_{x \sim Q} \left[ \ell(f_\theta(x), g(x)) \right] \]

\text{support}(Q) \supset \text{support} (\text{training dist})

\text{different graph size,}
\text{graph structure, edge weights, ...}


Physical reasoning
\text{different position, mass, number of objects}
Big picture: when may extrapolation “work”?

1) Data distributions in training and test are sufficiently similar

   same distribution of computation trees (message passing GNNs) \cite{Yehudai-Fetaya-Meiron-Chechik-Maron 21}
   shared underlying structure (spectral GNNs) \cite{Levie et al 2019, Ruiz et al 2020}

   --- or … ---

2) Understand what the model “learns”, and work around that:
   restrict the model via prior knowledge

   \cite{Xu-Zhang-Li-Du-Kawarabayashi-Jegelka 21}

   Neural network structure, optimization algorithm, data geometry
Graph predictions and distribution shifts

- **Worst-case scenario**: arbitrary predictions on unseen computation trees

\[
\mathbb{E}_{x \sim Q} [\ell(f_\theta(x), g(x))] \\
\text{support}(Q) \supset \text{support}(\text{training dist})
\]

**Theorem (Yehudai et al 2021):** Let \( \mathcal{P} \) and \( \mathcal{Q} \) be finitely supported distributions on graphs, and \( \mathcal{P}^t \), \( \mathcal{Q}^t \) the distribution of computation trees at depth \( t \). If any graph in \( \mathcal{Q} \) contains a tree in \( \mathcal{Q}^t \setminus \mathcal{P}^t \), then there is a GNN with depth at most \( t + 3 \) that perfectly solves the task on \( \mathcal{P} \) but has arbitrarily large error on all graphs from \( \mathcal{Q} \).

Smother degrading performance with appropriate metric on graphs?
Stability and measuring perturbations: 
Tree mover’s distance

C. Chuang, S. Jegelka. Tree Mover’s Distance: Bridging Graph Metrics and Stability of Graph Neural Networks. NeurIPS, 2022
An appropriate metric?

- Metric should capture Lipschitz/stability properties of GNNs, including invariances

- Message passing GNN compares sets of computation trees (subtree patterns)

- Idea: Optimal transport distance
Tree mover’s distance

- Earth mover’s distance between sets in Euclidean space:
  \[
  \min_{\gamma \in \Gamma(X,Y)} \sum_{i,j=1}^{n,m} d(x_i, y_j) \cdot \gamma_{ij}
  \]
  \[
  \Gamma(X, Y) = \{ \gamma \in \mathbb{R}^{n \times m}_+ \mid \gamma 1_m = 1_n; \gamma^\top 1_n = 1_m \}
  \]

- Tree mover’s distance: \( x_i, x_j \) are trees.

Distance between vectors

Distance between trees?
Tree Distance via Hierarchical OT

Tree Distance

\[ \text{Tree Distance} \left( \right) = \| x_{T_a} - x_{T_b} \| + \text{OT} \left( \{ \text{Subtree } T_a \} , \{ \text{Subtree } T_b \} \right) \]

Root Difference

\[ = \| x_{T_a} - x_{T_b} \| + \text{TD} \left( \{ \text{Subtree } T_a \} , \{ \text{Subtree } T_b \} \right) \]

Subtree Difference

Recursive!

Tree Distance via Hierarchical OT

\[ \text{TD}_w(T_a, T_b) := \begin{cases} 
\| x_{T_a} - x_{T_b} \| + w(L) \cdot \text{OT}_{\text{TD}_w}(\rho(T_{T_a}, T_{T_b})) & \text{if } L > 1 \\
\| x_{T_a} - x_{T_b} \| & \text{otherwise},
\end{cases} \]

where \( L = \max(\text{Depth}(T_a), \text{Depth}(T_b)) \) and \( w : \mathbb{N} \to \mathbb{R}^+ \) is a depth-dependent weighting function.
Properties & implications of Tree mover’s distance

- **pseudo-metric**: distinguishes the same graphs as the color refinement / Weisfeiler-Leman algorithm and GNNs, but graded: same “invariances”!

- **relation to stability of GNNs**: Lipschitz constant of GNN (GIN)

\[ \| h(G_a) - h(G_b) \| \leq \prod_{l=1}^{L+1} K^{(l)}_{\phi} \cdot \text{TMD}_{w+1}^{L+1}(G_a; G_b) \]

- **use TMD in cross-domain generalization bound** (Shen et al, 2018):

\[ R_T(h) \leq R_S(h) + 2\text{Lip}(h) \cdot \mathcal{W}_1(p_S, p_T) + \text{small value} \]
Empirically

• comparison with Wasserstein Weisfeiler-Leman metric \((Togninalli \ et \ al \ 2019)\)

1. **stability:**

   3 Layers
   \[ r = 0.76 \]
   
   2 Layers
   \[ r = 0.81 \]
   
   1 Layers
   \[ r = 0.85 \]

   MUTAG data, randomly sampled pairs

2. correlation with **accuracy drops under domain shifts** (PTC data):
   \[
   \begin{align*}
   \text{WWL: } & 0.489 \\
   \text{TMD: } & 0.712
   \end{align*}
   \]
Large perturbations: inductive biases

Generalization to very different data

Physical reasoning

different position, mass, number of objects

Different graph size, graph structure, edge weights, ...

Big picture: when may extrapolation “work”?

1) Data distributions in training and test are sufficiently similar
   
   same distribution of computation trees (message passing GNNs) \((Yehudai-Fetaya-Meiron-Chechik-Maron 21)\)
   
   shared underlying structure (spectral GNNs) \((Levie et al 2019, Ruiz et al 2020)\)

   --- or … ---

2) Understand what the model “learns”, and work around that:
   restrict the model via prior knowledge
   \((Xu-Zhang-Li-Du-Kawarabayashi-Jegelka 21)\)

   Neural network structure, optimization algorithm, data geometry
Neural network architecture and algorithms

- Algorithm  =  structured arrangement of subroutines
- (Graph) Neural network  =  structured arrangement of learnable “modules”

Bellman-Ford

\[
\text{for } k = 1 \ldots |S| - 1: \\
\text{for } u \text{ in } S: \\
\quad d[k][u] = \min_v d[k-1][v] + \text{cost}(v, u)
\]

GNN

\[
\text{for } k = 1 \ldots \text{GNN iter}: \\
\text{for } u \text{ in } S: \\
\quad h_u^{(k)} = \Sigma_v \text{MLP}(h_v^{(k-1)}, h_u^{(k-1)})
\]

Algorithmic Alignment:
Neural Network can mimic algorithm via few, easy-to-learn “modules”
Empirically: MLP with ReLU activations

\[ h^{(k)}_{v_i} = \sum_{\nu_j \in \mathcal{N}(i)} \text{MLP}^{(k)}(h^{(k-1)}_{v_i}, h^{(k-1)}_{\nu_j}) \]
Extrapolation in fully connected ReLU networks

**Theorem** (Xu-Zhang-Li-Du-Kawarabayashi-J 21)

Let $f$ be a 2-layer ReLU MLP trained with Gradient Descent. Along any direction $v \in \mathbb{R}^d$, $f$ approaches a linear function: let $x = tv$. As $t \to \infty$: $f(x + hv) - f(x) \to \beta_v h$ with rate $O(1/t)$.

(Linear regions: Montufar et al 2014, Arora et al 2018, Hanin & Rolnick, 2019; Hein et al., 2019, XZDKJ20)
Implications

1. Can only extrapolate linear functions

2. Training Data geometry
Implications for the full GNN

Shortest Path: \( \text{dist}[k][v] = \min_{u \in \mathcal{N}(v)} \text{dist}[k - 1][u] + w(u, v) \)

GNN:
\[
  h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w(u, v))
\]

GNN II:
\[
  h_v^{(k)} = \max_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w)
\]

(Veličkovic et al 2020)

Need MLP to be nonlinear!
Implications for the full GNN

Shortest Path:  
\[
\text{dist}[k][v] = \min_{u \in \mathcal{N}(v)} \text{dist}[k-1][u] + w(u, v)
\]

GNN:  
\[
h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w(u, v))
\]

GNN II:  
\[
h_v^{(k)} = \max_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w)
\]

Task-specific nonlinearities help extrapolation. Empirically reflected in many works

Need MLP to be nonlinear!
Encode nonlinearities in the ...

... architecture

\[
\text{NALU}: \quad y = g \odot a + (1 - g) \odot m
\]

\[
m = \exp W(\log(|x| + \epsilon)), \quad g = \sigma(Gx)
\]

Exp log for learning multiplication
(Trask et al 2018)

Library of programs
(Johnson et al 2017, Yi et al 2018, Mao et al 2019, …)

Prior knowledge or representation learning

(figure of n-body system: Battaglia et al 2018)

Learning physics laws
(Cranmer et al 2019, 2020)

... input representation
Neural Network Losses from Set Function Extensions

Setup

- use NN as “solver”, and objective function of the optimization problem as a loss
- What if the objective is a set function? \( F(S), S \subseteq [n] \)

- Continuous extension:
  \[
  F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \rightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}
  \]

- Want that:
  - \( f \) is continuous
  - \( f(1_S) = F(S) \)

- Strategy:
  \[
  f(x) = \sum_{S \subseteq [n]} p_x(S) F(S)
  \]
  \( \text{marginals: } \)
  \[
  p_x(i \in S') = x_i
  \]
Example: Lovász extension of submodular set function
Extensions: higher-dimensional

NN outputs…

\[ S \in \{0, 1\}^n \]

\[ F(S) \]

\[ x \in [0, 1]^n \]

\[ f(x) = \sum_{S \subseteq [n]} p_x(S)F(S) \]

\[ f(\mathbf{1}_S) = F(S) \]

\[ X \in S_+ \]

\[ f(X) = \sum_{S, T \subseteq [n]} p_X(S, T)F(S \cap T) \]

\[ f(\mathbf{1}_S \mathbf{1}_S^\top) = F(S) \]

We want:
Derivation in a nutshell

- vector extension for $x \in [0, 1]^n$: $p_x(S)$ solution to dual of "$f$ is a convex envelope of $F$":

$$\min_{\{y_S \geq 0\} S \subseteq [n]} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S 1_S = x, \quad \sum_{S \subseteq [n]} y_S = 1$$

$=> marginals$

$$p_x(i \in S) = x_i$$
Derivation in a nutshell

- vector extension for $x \in [0, 1]^n$: $p_x(S)$ solution to dual of “$f$ is a convex envelope of $F$”:

$$\min_{\{y_S \geq 0\}_{S \subseteq [n]}} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S 1_S = x, \sum_{S \subseteq [n]} y_S = 1$$

- use SDP version of this

- any valid vector extension leads to a valid matrix extension: Let $X = \sum_i \lambda_i v_i v_i^\top$

$$\sum_{S, T \subseteq [n]} p_X(S, T) F(S \cap T) \quad \text{with} \quad p_X(S, T) = \sum_i \lambda_i p_{v_i}(S)p_{v_i}(T)$$
Empirical results

max clique

Approximation ratio

k-clique

REINFORCE
LOVASZ EXT
MATRIX LOVASZ

ENZYMES PROTEINS IMDB-Bin MUTAG COLLAB
GNNs for Learning for Combinatorial Optimization

- Active, recent area, but need to understand learned functions

- To what examples will my models generalize? - understanding the data space. metric from the GNN perspective

- What are important architectural choices? - understanding the model. nonlinearities and alignment with algorithms

- How to train the model? - choice of loss function. higher-dimensional extensions of set functions tend to work better

chat more? stefje@mit.edu