Divide and Conquer with Neural Networks

Alex Nowak, Joan Bruna

NYU

alexnowakvila@gmail.com

February 15, 2017

Overview

Outline for section 1

Motivation

- Successful DL models capture inductive biases that are aligned with the structure of the data.
- Two famous examples:
  - **Convolutional Neural Networks**: Exploit spatial stationarity and geometrical stability of computer vision tasks:
    If $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ is a CV task and $\mathcal{T}$ a small deformation, then $\mathcal{F}\mathcal{T}x \approx \mathcal{T}\mathcal{F}x$ for all x.
    - CNNs leverage this prior by breaking the representation at different scales and by learning shared weights.
  - **Recurrent Neural Networks**: many time series are stationary across time and have fast decaying memory.
    - Joint data distribution is well approximated with autoregressive models.

Inductive bias for algorithmic discrete tasks

- Consider discrete algorithmic tasks as discrete combinatorics, discrete geometry and graph problems.
- Some problems have some kind of scale stationarity, i.e, solving a problem for size m is 'similar' as solving the problem for size $n > m$.
- This **self-similarity across scales** is made explicit by the concept of **recursion** which leads to the **principle of divide and conquer**.
- Recursion:
  - Relates solutions of different scales in a 'simple' way.
  - Generally leads to optimal complexity algorithms by exploiting the self-similarity of the problem across scales.
- How can this inductive-bias be implemeneted/exploited to learn efficiently on the class of tasks that are near scale-invariant?

Outline for section 2

Classical formulation of dynamic programming

Suppose we want to solve the task $T(X)$, ($|X| = n$), and we are able to write this expression in the following way:

$$T(X) = M(T(S(X)))$$

where:

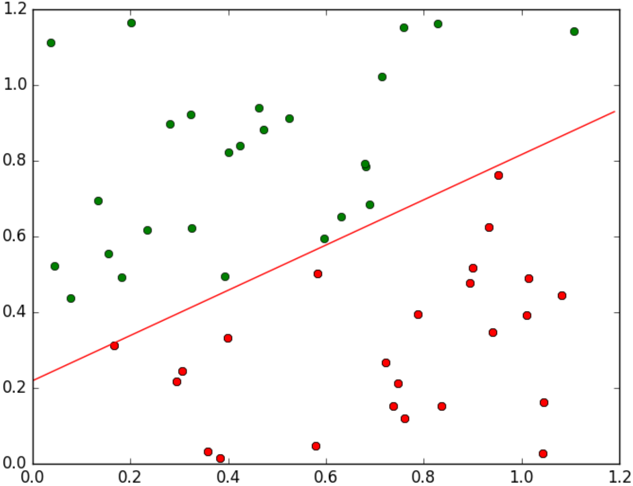
- S (split):
  - Commonly split the set X into k subsets $\{X_k\}_k$, $X_k \subsetneq X$
  - Can also be $X_k \not\subseteq X$ as long as $|X_k| < |X|$
- M (merge): merge the solutions $\{T(X_k)\}_k$

- Sorting (*quicksort*):
  - $S(X) = (X_{\leq m}, X_{>m})$ (split w.r.t the median)
  - $M(T(S(X))) = (T(X_{\leq m}), T(X_{>m}))$ (concatenate)

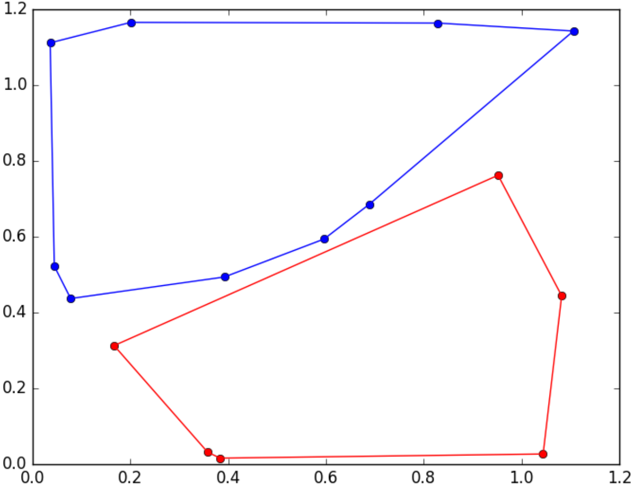
- Sorting (*mergesort*):
  - $S(X) = (X_{(1:\lceil \frac{n}{2} \rceil)}, X_{(\lceil \frac{n}{2} \rceil + 1:n)})$ (split w.r.t middle coordinate)
  - $M(T(S(X))) = M(T(X_{(1:\lceil \frac{n}{2} \rceil)}), T(X_{(\lceil \frac{n}{2} \rceil + 1:n)}))$ (merge two sorted vectors)

- Finding planar convex hull: Given a set of points, find the extremal points of the convex set generated by this points.
    - $S(X) = (X_{h(X)\leq 0}, X_{h(X)>0})$ (split w.r.t an affine plane h)
    - $M(T(S(X))) = M(T(X_{h(X)\leq 0}), T(X_{h(X)>0}))$ (merge disjoint convex hulls)

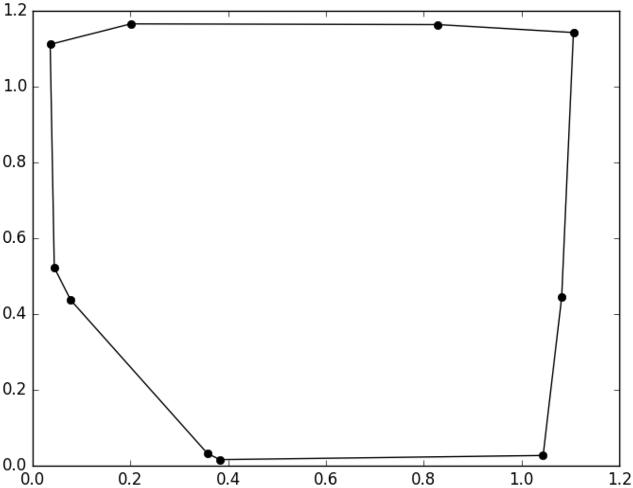
Split the input $S(X) = (X_{h(X) \leq 0}, X_{h(X)>0})$



Solve the subproblems $T(X_{h(X)\leq 0})$, $T(X_{h(X)>0})$



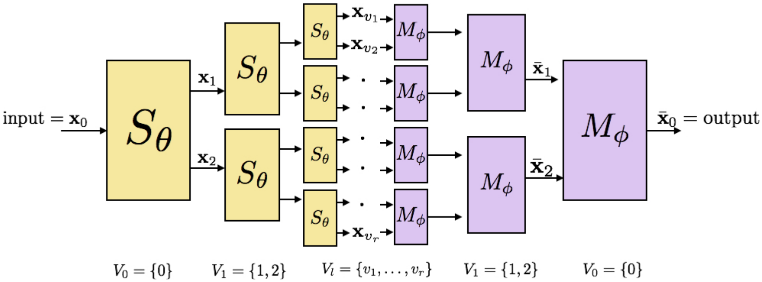
Merge the solutions $M(T(X_{h(X)\leq 0}), T(X_{h(X)>0}))$



Outline for section 3

Model of the dynamic arquitecture

- We will impose the inductive bias of stationarity across scales by dynamically solving the given task where $M_\phi(\cdot)$ and/or $S_\theta(\cdot)$ will be modeled by neural networks.
- The set of parameters ϕ and θ will be then shared across scales.
- The balancing is *dynamic*, data-dependent.



REMARK: The architecture is **not** end-to-end differentiable.

Important Simplification

- In this work we will only consider split and merge tasks **separately**.
- Consider tasks for which the dynamic solution can be computed (or mostly), only with $M_\phi(\cdot)$ or $S_\theta(\cdot)$.

Forward Pass

- **Split**:
  1. $S_\theta(\mathbf{x}) = \hat{y}(\theta)$ usually vector of bernouilli probabilities for every coordinate of $\mathbf{x}$.
  2. Build partition from it: $\hat{y}(\theta) \to \{\mathbf{x}_1, \mathbf{x}_2\}$
- **Merge**
  1. $M_\phi(\mathbf{x}_1, \mathbf{x}_2) = \hat{y}(\phi)$ usually also vector of probabilities for every coordinate of $\mathbf{x}_1, \mathbf{x}_2$.
  2. Build output $\mathbf{x}$ from it: $\hat{y}(\phi) \to \{\mathbf{x}\}$

Outline for section 4

Strong vs Weak supervision

We can differentiate between two kinds of supervision depending on the amount of information we have for training.

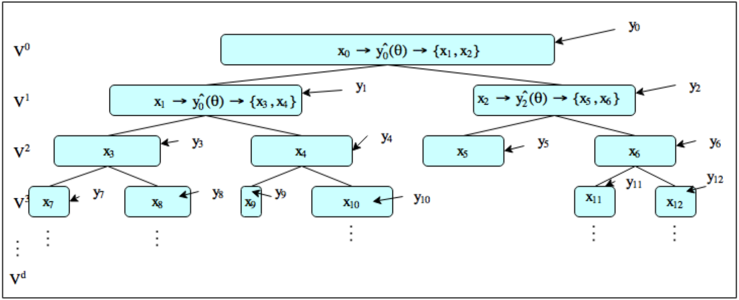
- **Strong Supervision**: We know the dynamic task by hand and we have access to the targets at every node of the tree.
- **Weak Supervision**: We only observe input-output pairs at the largest scale, we do not have intermediate labels for each split/merge decision.

Strong Supervision

Suppose we know the dynamic task by hand and we have access to the targets y_v at every node of the generated tree.

- We can supervise all nodes of the dynamic tree.

$$\mathcal{L}(\theta) = \sum_{k=1}^{d} \sum_{v \in V^k} l^s(y_v^s, \hat{y}_v(\theta))$$



It is a form of **currculum learning**.

Weak Supervision

- Since we only observe input-output pairs at the largest scale, we do not have intermediate labels for each split/merge decision.
- The training by **weak supervision** balances two quantities:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{acc}}(\theta) + \mathcal{L}_{\text{compl}}(\theta)$$

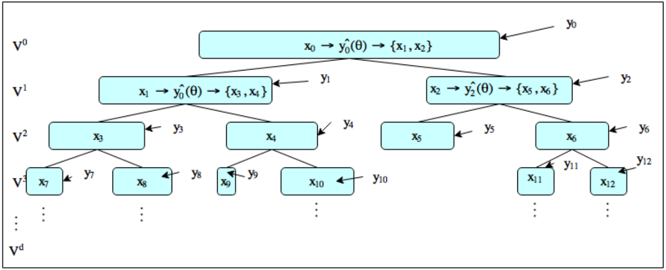
  where
  - $\mathcal{L}_{\text{acc}}(\theta)$ enforces solving the task with **high accuracy**.
  - $\mathcal{L}_{\text{compl}}(\theta)$ enforces solving the task with **small complexity** by exploiting a task-specific prior.

Weak Supervision

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{acc}}(\theta) + \mathcal{L}_{\text{compl}}(\theta) = \sum_{k=1}^{d} \sum_{v \in V^k} [l^w(y_v^w, \hat{y}_v(\theta)) + \alpha_k R_\theta(\hat{y}_v(\theta))]$$

where:

- $l^w(y_v^w, \hat{y}_v(\theta))$ only supervises elements which are present at the final target (y_v^w).
- $R_\theta(\hat{y}_v(\theta))$ acts as a regularizer enforcing some prior knowledge for the task.



Outline for section 5

1. Motivation
2. Introduction
   - Classical formulation of dynamic programming
   - Some examples
3. Model
4. Training
5. Experiments
   - Sorting
   - Planar ConvexHull
6. Current and Future work
7. Summary

Split model for sorting

- The split model will receive a sequence of real numbers and will output a binary mask on them.
- We will use a model for sets, i.e, the function will be covariant to permutations $S_\theta(\mathbf{x}_\sigma) = S_\theta(\mathbf{x})_\sigma$.

$$\mathbf{x} = (x_1, \dots, x_n)$$

$$h_i^{k+1} = \mathrm{sigmoid}(W_1^k(h_i^k, x_i) + W_2^k \bar{h}^k) \qquad k = 0, \dots, l-1$$

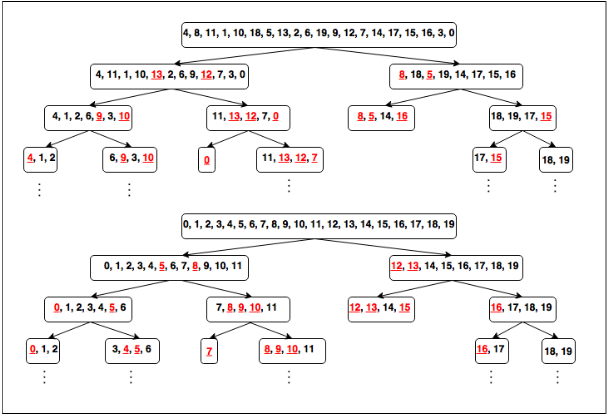
$$\hat{y} = p_i = \mathrm{sigmoid}(Wh_i^l + b)$$

where $\bar{h}^k = \frac{1}{n}\sum_i h_i^k$.

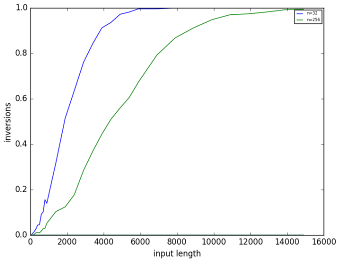
- Sample or take the mode from $\mathbf{p}$ to decide the partition.

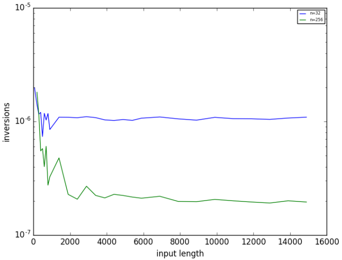
Training sorting

- Only can supervise elements which are in the correct branch of the tree according to the target.
- $R(\hat{y}(\theta)) = \hat{\sigma}(\hat{y}_v) = \frac{1}{n_v - 1}\sum_{i=1}^{n_v}(p_i^2 - \hat{\mu})$ maximize variance of $\mathbf{p} = (p_1, \ldots, p_{n_v})$ to enforce even splits.



	Test n = 32	Test n =[64,128]
HAM trained n = 32	0.04%	0.24%
DivConq trained n = 32	0.02%	0.16%
DivConq trained n = 256	0%	0.1%





Planar Convex Hull

We will suppose an oracle split, i.e, a tree-structured partition of the input set of points into disjoint sets.

The dataset consists of input-output pairs where:

- inputs are tree-structured disjoint convex hulls.
- targets are the final convex hull (WS), or the target convex hulls at each dynamic step (SS).

At each scale the model sees much less points:

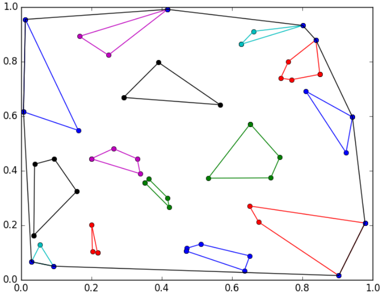
- if $X = \{x_1, \dots, x_n\} \in K$ taken uniformly from a planar polytope, then

$$\mathbb{E}|CH(X)| \lesssim \log n$$

- This property can be exploited by a dynamic model! The number of points is considerably reduced across scales.

Planar Convex Hull: Model and Weak supervision

- We use a RNN with attention over the points and output a vector of probabilities.
- Only can supervise points that belong to the final convex hull.
- $R(\hat{y}_v(\theta)) = |\sum_{i=1}^{n_v} p_i - \alpha_k n_v|$ acts as a regularizer forcing the model to pick a certain amount of elements.



Convex Hull: Experiments

- All models trained for 60 points, 5 scales.
- Results:

Points,scales	60, 5	120, 6	240, 7
No-Disc	52%	26%	4%
Disc	51%	29%	9%
Baseline	46%	16%	0.2%

Outline for section 6

Current work

- Explore deeper the convex hull model. Make it end-to-end differentiable by not ruling out points at each scale.
- Dynamic shortest path.
- Prove **consistency** for weak supervision.

Future work

- Joint training **Split** + **Merge**.
- Applications to Hierarchical Reinforcement Learning.

Outline for section 7

- We present a dynamic model for algorithmic discrete tasks that exploits self-similarity across scales.
- **Pros**
  - Ability to train consistently with only input-output pairs.
  - Proved much better generalization for $n \to \infty$ than models that try to solve directly the task.
  - Flexible running complexity and learnable in a differentiable manner.
- **Cons**
  - For now, very tailored for the task.
  - Most of the times Merge/Split model can't be implemented with binary masks and order matters a lot.

Thanks!