Learning and Inference for Structured Prediction: A Unifying Perspective

Aryan Deshwal[1] , **Janardhan Rao Doppa**[1] , **Dan Roth**[2]
[1]School of EECS, Washington State University
[2]Department of Computer and Information Science, University of Pennsylvania
{aryan.deshwal,jana.doppa}@wsu.edu, danroth@seas.upenn.edu

Abstract

In a structured prediction problem, one needs to learn a predictor that, given a structured input, produces a structured object, such as a sequence, tree, or clustering output. Prototypical structured prediction tasks include part-of-speech tagging (predicting POS tag sequence for an input sentence) and semantic segmentation of images (predicting semantic labels for pixels of an input image). Unlike simple classification problems, here there is a need to assign values to multiple output variables accounting for the dependencies between them. Consequently, the prediction step itself (aka "inference" or "decoding") is computationally-expensive, and so is the learning process, that typically requires making predictions as part of it. The key learning and inference challenge is due to the exponential size of the structured output space and depend on its complexity. In this paper, we present a unifying perspective of the different frameworks that address structured prediction problems and compare them in terms of their strengths and weaknesses. We also discuss important research directions including integration of deep learning advances into structured prediction methods, and learning from weakly supervised signals and active querying to overcome the challenges of building structured predictors from small amount of labeled data.

1 Introduction

Structured prediction (SP) tasks arise in several domains including natural language processing, computer vision, computational biology, and graph analysis. In a SP problem, the goal is to learn a mapping from structured inputs to structured outputs. For example, in semantic labeling of images, the structured input is an image and the structured output is a labeling of the image regions. In structured prediction tasks, we need to predict multiple output variables by exploiting the dependencies between them. Viewed as a traditional classification problem, the set of candidate classes in structured prediction is exponential in the size of the output. The large number of candidate structured outputs pose significant inference and learning challenges (inference task is NP-Hard and learning for probabilistic SP is #P-hard due to the computation of partition function. Specifically, the time complexity of exact inference depends on the complexity of model that tries to capture the dependency structure between input and output variables. Efficient solutions exist only when this dependency structure forms a tree with small width. Therefore, the core research challenge in structured prediction has been to achieve a balance between two conflicting goals: 1) It must be flexible to allow for complex and accurate predictors to be learned, and 2) It must support computationally-efficient inference of outputs.

There are different structured prediction frameworks that make varying trade-offs between the above two goals. In this paper, we present a unifying perspective of the different frameworks to solve structured prediction problems and compare them in terms of their strengths and weaknesses. The unifying perspective relies on two key abstractions: 1) Inference is formulated as implicit or explicit search process; and 2) Learning is performed with a fixed inference scheme or learning also supports to optimize the efficiency and accuracy of inference. We also discuss some important future research directions in this area, namely, integrating advances in deep learning for structured prediction and learning from small amount of labeled data. Since the literature on structured prediction is vast, we refer the reader to our recent IJCAI and AAAI tutorials for a complete set of references.

2 Problem Setup

A structured prediction problem specifies a space of structured inputs $\mathcal{X}$, a space of structured outputs $\mathcal{Y}$, and a non-negative *loss function* $L : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re^+$ such that $L(x, y', y^*)$ is the loss associated with labeling a particular input x by output y' when the true output is y^*. Without loss of generality, we assume that each structured output $y \in \mathcal{Y}$ be represented using d discrete and/or continuous variables $v_1, v_2, \cdots, v_d$, and each variable v_i can take candidate values from a set $C(v_i)$. Since all algorithms will be learning functions or objectives over input-output pairs, they assume the availability of a *joint feature function* $\Phi : \mathcal{X} \times \mathcal{Y} \mapsto \Re^m$ that computes an m dimensional feature vector for any pair.

Example 1: Part-of-speech (POS) tagging task. Each structured input is a sequence of words. Each output variable v_i stands for POS tag of a word and $C(v_i)$ is the list

of all candidate POS tags. Hamming loss (number of incorrect POS tags) is typically employed as loss function. Joint features include *unary features* (representing words and their POS tags as in a multi-class classifier) and structural features (e.g., label pairs to capture the strength of label transitions).

Example 2: Image labeling task. Each structured input is an image. Each output variable v_i corresponds to a semantic label of one pixel in the image and $C(v_i)$ is the list of all candidate labels. Intersection-over-Union (IoU) loss – similarity between the predicted region and the ground-truth region for a given semantic labeling – is employed as loss function. Unlike Hamming loss, IoU loss doesn't decompose over the loss of individual output variables. Joint features include unary features and structural features (e.g., context features that count the co-occurrences of different labels in different spatial relationships such as *left*, *right*, *above*, and *below*. Intuitively, we are capturing, for example, whether a pixel labeled "sky" is below another pixel labeled "grass").

We are provided with a training set of structured input-output pairs $\{(x, y^*)\}$ drawn from an unknown target distribution $\mathcal{D}$. The goal is to return a function/predictor from structured inputs to outputs whose predicted outputs have low expected loss with respect to the distribution $\mathcal{D}$. The manner in which this goal is achieved varies among SP algorithms.

3 Cost Function Learning Approaches

Cost function learning approaches correspond to generalizations of traditional classification algorithms: Conditional Random Fields (CRFs) [Lafferty *et al.*, 2001], Structured Perceptron [Collins, 2002], and Structured Support Vector Machines (SSVM) [Taskar *et al.*, 2003; Tsochantaridis *et al.*, 2004]. These methods typically learn a linear cost function $C(x, y) = w \cdot \Phi(x, y)$ to score a candidate structured output y given a structured input x, where $w \in \Re^m$ stands for *weights/parameters*. Given such a cost function and a new input x, the output computation then involves finding the minimum cost output (aka *Argmin inference* problem):

$$\hat{y} = \arg \min_{y \in Y(x)} C(x, y).$$

These methods typically *do not have any explicit search formulation* for solving the inference problem.

Learning objective. The goal is to learn weights w of cost function $C(x, y)$ such that for each training example (x, y^*), the cost of the correct output is lower than score of any other candidate output y, i.e., $C(x, y^*) < C(x, y), \forall y \in Y(x) \setminus y^*$.

General learning framework. The general template for cost function learning is shown in Algorithm 1. There are two key design choices: 1) *Inference method* to solve the argmin inference problem. Efficient methods exist only when joint features $\Phi(x, y)$ are simple. Some examples include Viterbi algorithm for sequences and CKY algorithm for parsing. To optimize a given loss function L, we need to solve loss-augmented inference problem (i.e., adding loss in addition to cost to score candidate outputs). 2) *Optimization method* to tune the weights of cost function based on negative examples. Some examples include stochastic gradient descent, cutting plane, and dual coordinate descent algorithms. The general learning approach is iterative by nature. In each iteration, we perform three main steps: select a subset of inputs D' from the training set and compute predictions D'_y by running the inference algorithm with current weights; generate ranking examples if predictions don't match with ground-truth outputs; and update weights based on new or aggregate ranking examples using appropriate optimization method. By varying $|D'|$, we can get online ($|D'| = 1$), mini-batch ($|D'| < |D|$), and full batch ($|D'| = |D|$) training methods. Cost function learning approaches treat inference solver as a "black-box" during training (aka *learning with inference*).

Algorithm 1 Cost Function Learning Framework

Input: $D = \{x, y^*\}$, structured training examples
1: Initialize the weights of cost function C: $w \leftarrow 0$
2: **repeat**
3: Select a batch $D' \subseteq D$ for (loss-augmented) inference
4: // Call inference algorithm
5: $D'_y \leftarrow$ `Inference-Solver`$(D', C(x, y))$
6: **for** each $(x, y^*) \in D'$ and $\hat{y} \in D'_y$ **do**
7: If $y^* \neq \hat{y}$, create a ranking example $C(x, y^*) < C(x, \hat{y})$
8: **end for**
9: // Optimization to update weights
10: Update weights w using new or aggregate ranking examples
11: **until** convergence
12: **return** w, weights of the learned cost function C

Learning with approximate inference. It is conceivable that the above learning mechanism may not be reliable with approximate inference solvers. Researchers have investigated the impact of approximation on learning by considering two categories of approximate inference solvers [Finley and Joachims, 2008]: 1) *Undergenerating*, which consider a subset of the structured output space (e.g., greedy search); and 2) *Overgenerating*, which consider a superset of the output space (e.g., LP relaxations). Overgenerating methods are found to better than undergenerating ones both theoretically and empirically [Finley and Joachims, 2008; Kulesza and Pereira, 2007]. Learning methods that explicitly account for approximation in inference is an active area of research [Stoyanov *et al.*, 2011; Hazan *et al.*, 2016].

Key challenges. The three main challenges for cost function learning framework are as follows: 1) Time complexity of solving inference problem is very high for complex (i.e., higher-order) features; 2) Since weight learning approach repeatedly calls the inference solver, training is computationally expensive; and 3) It is hard to optimize non-decomposable loss functions (e.g., IoU loss) due to the difficulty in solving loss-augmented inference problems.

3.1 Recent Advances

In this section, we discuss some recent advances to address the above challenges.

Constrained conditional models and ILP inference. [Roth and Yih, 2004] developed a generic formulation of inference problem as a Integer Linear Program (ILP). The *constrained conditional modeling (CCM)* framework [Chang *et al.*, 2012] allows to combine the learned cost function $C(x, y)$ and background knowledge in the form of declarative constraints using ILP based inference. The CCM approach has

shown great success in a large number of NLP applications. ILP formulations were found to be helpful in modeling a large number of problems; the inference problems can be solved exactly or via various relaxation methods and are shown to work very well in practice. A recent work [Meshi *et al.*, 2016] provided a theoretical insight that explains this success: a large number of relaxed solutions are integral (relaxations are tight), and this tightness on training instances generalizes to testing instances (PAC theory for ILP based inference).

Decomposed learning. The decomposed learning framework [Samdani and Roth, 2012; Sontag *et al.*, 2010] improves the speed of training by performing inference over a subset of the structured output space $Y'(x) \subset Y(x)$. The size of $Y'(x)$ is determined by a parameter k and grows exponentially as a function of k. The general construction considers all candidate structured outputs whose Hamming loss with respect to correct output y^* is at most k (referred as *neighborhood* of y^*): $Y'(x) = \{y \in Y(x) : \texttt{Hamm-Loss}(y, y^*) \leq k\}$. Samdani and Roth [Samdani and Roth, 2012] provide theoretical conditions under which decomposed learning is equivalent to standard learning that considers entire output space $Y(x)$. In practice, decomposed learning is shown to achieve same accuracy as standard learning with a small value of k (e.g., 1,2,3) [Samdani and Roth, 2012; Sontag *et al.*, 2010]: significantly improves the training time.

Amortized inference and structured learning. We need to solve inference problems for multiple structured inputs during both training and testing. The naive approach is to run an inference solver *independently* on each input example. It is conceivable that we can learn useful knowledge while solving inference problems on past examples to improve the speed of inference on future examples. This is referred to as *amortizing* the cost of inference [Srikumar *et al.*, 2012], which is highly related to the speedup learning literature [Fern, 2010]. Recent work has exploited the ILP inference formulation as an abstraction to provably achieve amortized inference [Srikumar *et al.*, 2012] and significantly improve the speed of inference and of training cost functions [Chang *et al.*, 2015b]. The key idea is to store a set of cached solutions for ILP problems and reuse them for new inference problems without calling the inference solver when theoretical conditions are met. For NLP applications, many sentences have identical structured outputs such as POS tag sequences, parse trees, semantic parses etc. Therefore, the amortization theorems "fire" often and result in significant savings. By viewing inference procedures as computational search processes will allow us to study generic approaches to address speedup learning problems arising in structured prediction. Some examples include treating ILP inference as a white box (i.e., branch and bound search) to learn heuristic functions to achieve amortized inference.

Theoretical results. Early theoretical results for generalization were based on covering number bounds for decomposable loss functions and linear scoring functions [Taskar *et al.*, 2003]; and PAC Bayesian theory. Recent results based on factor graph complexity provide improved bounds, and served as a motivation to derive the voted risk minimization principle to achieve better generalization by learning a ensemble of simpler scoring functions [Cortes *et al.*, 2016].

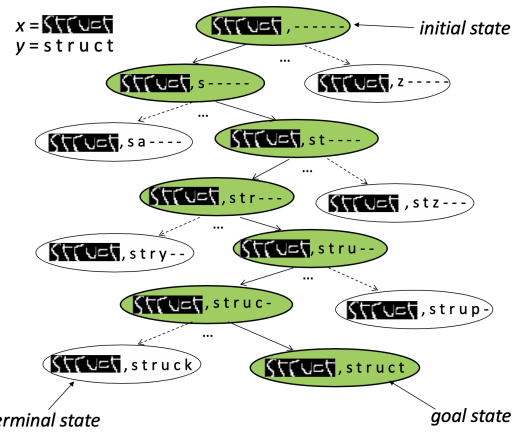

Figure 1: An example search space for handwriting recognition.

Structured prediction cascades. This approach addresses inference complexity via cascade training [Felzenszwalb and McAllester, 2007; Weiss and Taskar, 2010], where efficiency is achieved by performing multiple runs of inference from a coarse level to a fine level of abstraction using learned cost functions of varying complexity. We can view this as a form of progressive filtering of candidate structured outputs by trading-off the accuracy (number of errors) and efficiency (number of filtered outputs) of filtering at each level. [Weiss and Taskar, 2010] developed a forward training approach to learn the weights of different cost functions employed in the cascade. These methods have shown good success in practice, but they need to place some restrictions on the form of the cost functions to facilitate "cascading."

4 Search-based Learning Approaches

In this section, we discuss different search-based learning approaches and their distinction with traditional cost function learning methods. Search-based approaches formulates inference as an *explicit search problem*.

Overview. Search-based methods formulate the problem of structured prediction as an *explicit state-space search process* using a search architecture (search space, search procedure, and termination criteria). They learn appropriate search control knowledge (e.g., heuristics, cost functions) using training data to optimize the accuracy of this search architecture in making predictions. Unlike traditional approaches, there is no need to solve a global optimization problem at prediction time. In effect, the system learns how to do inference (aka *learning for inference*) when compared to *learning with inference*) style of cost function learning approaches.

Potential advantages. Some advantages of search-based methods include: 1) Scale gracefully with the representation complexity. We can employ higher-order features for functions that guide the search without increasing the inference complexity. 2) Since inference is modeled as "white-box", learning process can observe search errors and perform robust training. It can help with debugging from a practi-

tioner's perspective. 3) Can potentially train to optimize non-decomposable loss functions.

4.1 Heuristic Learning Methods

Heuristic learning approaches vary depending on the search space and search procedure employed in the overall search architecture. There are three key elements in the heuristic learning framework: 1) Search space over partial structured outputs; 2) Search procedure that is used to make predictions (e.g., beam search, greedy search); and 3) Search control knowledge to guide the search process.

Search space. A search space over partial structured outputs S_p is a 4-tuple $\langle I, A, \Phi, T \rangle$, where I is a function that maps an input x to an initial search node, A is a finite set of actions (or operators), Φ is a feature function from search nodes to real-valued feature vectors, and T is the terminal state predicate that maps search nodes to $\{0, 1\}$ indicating whether the node is a terminal or not. Each terminal node in the search space corresponds to a *complete structured output*, while non-terminal nodes correspond to *partial structured outputs*. Fig. 1 provides an illustration of search space for a simple handwriting recognition problem. Each search state is a pair (x, y') where x is the structured input (binary image of the handwritten word) and y' is a partial labeling of the word. The arcs in this space correspond to actions that label the characters in the input image in a left-to-right order by extending y' in all possible ways by one element. The terminal states or leaves of this space correspond to complete labelings of input x. The terminal state corresponding to the correct output y^* is labeled as *goal state*.

Making predictions. Given a structured input x, a search procedure P, and a learned heuristic function $\mathcal{H}$, the decision process for predicting a structured output corresponds to exploring the search tree rooted at initial node until reaching a terminal state. The complete structured output $\hat{y}$ from that terminal state is returned as the predicted output.

Learning objective. The goal is to learn parameters of heuristic function H such that for each training example (x, y^*), the search procedure P guided by H will return y^* as the predicted output.

Learning for Beam Search

We discuss the general framework for learning beam search heuristics and different instantiations below.

General framework. The heuristic function H is represented as a linear function: $H(n) = w \cdot \Phi(n)$, where w stands for weights and $\Phi(n)$ correspond to features of search node n. The general template for learning weights of heuristic H is shown in Algorithm 2. It is an iterative online learning approach. In each iteration, we perform learning on each training example $(x, y^*) \in D$ as follows. Beam B is initialized with the initial state x. Beam search is performed using the current weights until reaching a terminal state. If a search error is identified, weights are updated to move towards correctness and search is continued by updating the beam. There are three key design choices in this framework: 1) How is *search error* defined?; 2) What is the form of *weight update*; and 3) How to *update beam* after weight update?

Algorithm 2 Heuristic Learning for Beam Search

Input: $D = \{x, y^*\}$, structured input-output training examples; $S_p = \langle I, A, \Phi, T \rangle$, search space; b: beam width

```
Initialize the weights of heuristic function H: w ← 0
repeat
    for each training example (x, y*) ∈ D do
        Initialize beam B with I(x)
        repeat
            Perform beam search with current weights w
            if Search Error then
                Update weights w
                Reset beam B
            end if
            (Dis)continue search
        until reaching terminal state
    end for
until convergence or maximum iterations
return w, weights of the learned heuristic function H
```

Instantiations. We describe three main instantiations of the general framework that make different design choices. 1) *Early update* [Collins and Roark, 2004]. A search error is declared when a target node — node with all output variables are correct (green colored nodes in Fig. 1) — is not present in the beam. Intuitively, the search cannot reach the goal node anymore. Weight update is similar to structured perceptron over the partially labeled outputs. The positive and negative labelings differ in the last output variable. After weight update, beam is reset with the initial node $I(x)$ and search is (dis)continued for the current training example. One drawback on early-update is that learning can be very slow. 2) *Learning as Search Optimization (LaSO) update* [Daumé III and Marcu, 2005; Xu *et al.*, 2009]. Search error is defined similar to early-update. Intuitively, the weight update tries to increase the score of target nodes and decrease the score of non-target nodes. Specifically, weights are updated by adding the average feature vectors of target nodes and subtracting the average feature vectors of non-target nodes in the candidate set. After weight update, the beam is reset with target nodes in the candidate set (ties are broken using the updated heuristic scores). [Xu *et al.*, 2009] study different notions of margin and the conditions under which the weights will converge. Their work formalizes the intuition that learning becomes easier as we increase the beam width b (i.e., amount of search for reasoning). 3) *Max-violation update* [Huang *et al.*, 2012]. The key insight in this approach is that we only need violations to ensure convergence of weights (and not exact search). This framework is referred as *violation-fixing perceptron*. Early-update is a special case of this framework as it guarantees violation, but learning is very slow. To improve the speed of learning, max-violation update considers the worst-mistake (i.e., prefix where violation is maximum) by performing beam search until reaching a terminal state. A single weight update is performed per training example (x, y^*).

Learning for Greedy Search

Greedy search corresponds to constructing structured outputs based on a sequence of decisions: one for each output vari-

able $v_1, v_2, \cdots, v_d$. This is formalized for a given structured-prediction problem by defining a search space by employing a fixed ordering over output variables (e.g., left-to-right in sequence labeling). The search control knowledge corresponds to a recurrent classifier (or policy) that predicts the label for the next output variable. In Fig. 1, highlighted search nodes correspond to the trajectory of the optimal recurrent classifier (i.e., a classifier that chooses correct action at every state leading to the goal state).

Imitation learning formulation. The problem of learning greedy policy can be formulated in the framework of Imitation Learning (IL) . IL is considered to be an efficient framework for learning sequential decision-making policies when a good expert policy is available to drive the learning process. For structured prediction, ground-truth structured outputs can be considered as demonstrations of expert policy.

Exact imitation and error propagation. The most basic approach to learning a recurrent classifier is via *exact imitation* of the trajectory followed by the optimal classifier. For example, the sequence of highlighted states in Fig. 1 correspond to the solution path for producing y^* from x. The exact imitation approach learns a classifier by creating one classification training example for each search node n on the solution path with feature vector $\Phi(n)$ and label equal to the action followed by the path at node n. The aggregate set of classification examples collected over all the training examples in D are given to a classifier learning algorithm (e.g., SVM) to induce a recurrent classifier. We can view this as a reduction of structured prediction to classifier learning for simple outputs [Beygelzimer *et al.*, 2016]. This allows us to leverage off-the-shelf classifier learning algorithms to solve structured prediction problems. Unlike supervised learning problems that assume IID input examples, our problem is non-IID because the next state depends on the decision of the classifier at the previous state. Therefore, recurrent classifiers learned via exact imitation can be prone to error propagation.

Advanced imitation learning algorithms. To address the error propagation issue, we can employ advanced imitation learning algorithms including SEARN [Daumé III *et al.*, 2009], DAgger [Ross *et al.*, 2011], and LOLS [Chang *et al.*, 2015a]. The general idea is to iteratively observe the behavior of the current recurrent classifier and augment training data to better represent important states. This additional data allows us to learn robust classifiers that can recover from their own mistakes. Recent work has extended IL methods to handle continuous structured outputs including time-series [Venkatraman *et al.*, 2015; Le *et al.*, 2017].

Easy-First Learning

The main drawback of classifier-based structured prediction approaches is that they require a fixed ordering over the output variables. The easy-first framework [Goldberg and Elhadad, 2010; Stoyanov and Eisner, 2012; Xie *et al.*, 2015] overcomes this drawback by allowing the learner to dynamically select the ordering. Specifically, we modify the search space to consider labeling all unassigned output variables at each greedy search step. The structured output is constructed incrementally by making the easiest (most confident) decision at each decision step to gather more evidence for making hard decisions later. This principle is very similar to that of constraint satisfaction algorithms.

Good and bad actions. Given the actions $A(s)$ from a specific state s, we consider any action $a \in A(s)$ that results in a state s' with $L(s') < L(s)$ as a *good action*; otherwise, it is a *bad action*. Within the Easy-first framework, it is typical to encounter states that have more than one good action. We denote the set of all good actions in state s as $G(s)$ and the set of all bad actions as $B(s)$.

Action scoring function. The action scoring function $\mathcal{H}$, evaluates all actions in $A(s)$ and guides the greedy search to incrementally produce the structured output. Existing approaches consider linear action-scoring functions $\mathcal{H}(s, a) = w \cdot \Phi(s, a)$, where w is the weight vector and $\Phi(s, a)$ is a predefined feature function.

Learning approach. The *learning goal* within the Easy-first framework is to learn a weight vector w such that the highest scoring action in each step is a *good* action. The general online training procedure is described in Algorithm 3. In any given state s, if the current highest scoring action a_p is a bad action, then weights are updated. Existing strategies for weight update include: 1) perceptron style update involving highest-scoring good action and highest-scoring bad action [Goldberg and Elhadad, 2010]; 2) highest-scoring good action and all violated bad actions [Xie *et al.*, 2015]; and 3) the regularized versions of (1) and (2) to minimize the change of weights that were shown to perform better in practice [Xie *et al.*, 2015]. After the weight update, a `ChooseAction` procedure is called to select the next action, and we transit to the next search state. The choice of `ChooseAction` will determine the training trajectories. Two types of approaches have been pursued in the literature for this purpose: on-trajectory training, which always chooses an action in $G(s)$ (e.g., the highest-scoring action in $G(s)$), and off-trajectory training, which always chooses the highest-scoring action based on the current scoring function even when it is a bad action. This is repeated until reaching a terminal state.

4.2 Heuristic and Cost Learning Methods

In this section, we discuss approaches that learn both heuristic and cost functions for structured prediction.

HC-Search Framework

Overview. The HC-Search framework [Doppa *et al.*, 2014a; Lam *et al.*, 2015; Doppa *et al.*, 2014c] decomposes the structured prediction problem into three steps: 1) Find an initial complete output; 2) Explore a search tree of alternative candidate outputs rooted at the initial solution; and 3) Score each of these candidates to select the best one. Step 1 can be done by any method. Step 2 is guided by a learned heuristic H and Step 3 is performed by a learned cost function C.

Advantages. First, in the standard approaches a global cost function must be trained to "defend against" the exponentially-large set of all wrong candidate outputs to the problem. This is expensive both computationally and in terms of sample complexity. It can require highly expressive representations (e.g., higher-order potentials). In contrast, the

Algorithm 3 Easy-First Policy Learning

Input: $D = \{x, y^*\}$, structured input-output training examples; $S_p = \langle I, A, \Phi, T \rangle$; L: loss function

Initialize the weights of policy H: $w \leftarrow 0$
repeat
  for each training example $(x, y^*) \in D$ **do**
    $s \leftarrow \mathrm{I}(x_i)$ and TERMINATE $\leftarrow False$
    while not TERMINATE **do**
      $a_p \leftarrow_{a \in A(s)} w \cdot \Phi(s, a)$
      // update weights if a bad action is selected
      if $a_p \in B(s)$ **then**
        UPDATE$(w, G(s), B(s))$
      end if
      $a_c \leftarrow$ `ChooseAction`$(A(s))$
      $s \leftarrow$ Apply a_c on s
      if Terminal(s) **then**
        TERMINATE $= True$
      end if
    end while
  end for
until convergence or maximum iterations

heuristic function H only needs to correctly rank the successors of each state that is expanded during the heuristic search in Step 2, and the cost function C only needs to correctly find the best of these in Step 3. These are much easier learning problems, and hence, simpler potential functions can be applied. Second, for making predictions there is no need to solve a global optimization problem at prediction time. In effect, the system learns not only how to score candidate solutions but also how to *find* good candidates—it learns to do inference more efficiently. Third, HC-Search can be applied to non-decomposable loss functions. This is another consequence of using a search space formulation. Finally, HC-Search provides a clean engineering methodology for determining which components of the system would most benefit from additional engineering effort.

Key learning challenges. There are three key learning challenges in this framework: How can we automatically *design* an efficient search space over outputs? (Search space design); How can we learn a heuristic function H for effectively *guiding* the search? (Heuristic learning); and How can we learn a cost function C that can accurately *select* the best output among the candidate outputs? (Cost function learning)

Search space design. The effectiveness of this framework critically depends on the quality of search space, where quality is defined as the expected search depth at which target outputs y^* can be located. If target search depth is small, it will make the heuristic and cost function learning problems easier. Some generic search spaces over structured outputs include *Flipbit space*, *Limited Discrepancy Search (LDS) space* [Doppa *et al.*, 2014b] that leverages greedy recurrent classifers, and *Randomized Segmentation space* [Lam *et al.*, 2015] for computer vision tasks. We note that search space design is a very under-studied problem and lot more work needs to be done in this direction.

Heuristic function learning. The heuristic learning approach is based on the observation that for many SP problems, we can quickly generate very high-quality outputs by guiding the search procedure using the true loss function (which is only available during training). Motivated by this observation, the heuristic learning problem is formulated in the framework of *imitation learning* to learn a heuristic that mimics the search decisions made by the true loss function on training examples. This is a generic approach that is applicable to all ranking-based search procedures (e.g., greedy search and beam search). The aggregate set of ranking constraints collected over all of the training examples is given to a rank learning algorithm to learn the heuristic function H.

Cost function learning. Given a learned heuristic H, we want to learn a cost function that correctly ranks the candidate outputs generated by the search procedure guided by H. This is formulated as another rank-learning problem such that the cost function C scores the best loss output generated during search higher than the other outputs.

Other Frameworks

Re-Ranking methods generate a k-best set of candidate outputs and score them using a learned cost function. Candidate outputs can be produced using a generative model [Collins, 2002] or a MAP inference solver [Yadollahpour *et al.*, 2013]. blueAdditionally, if we model ILP inference solver as a computational search process using branch-and-bound procedure, we can learn heuristics and policies to adaptively decide the node searching order to improve the speed of inference — another way to amortize the cost of inference. Recent work on randomized greedy search based amortized inference and learning approach can be seen as an instantiation of HC-Search. Evaluation function to select good starting solutions for greedy search and scoring function to select the final structured output correspond to H and C respectively, but they are learned using different methods.

5 Deep Learning ∩ Structured Prediction

An interesting question to ask is "Do we need structured prediction in the era of deep learning?" The answer is *yes* because deep models are sample in-efficient and structure is a compact way of representing large amounts of data [Liang *et al.*, 2008]. Therefore, it is beneficial to integrate the advances in deep learning with structured prediction frameworks.

Decoupled integration. Classical deep networks allow us to learn complex unary features over structured input variables. In decoupled integration, we can employ the unary features learned from deep networks into existing structured prediction methods [Huang *et al.*, 2015].

Search-based learning with deep models. To leverage structural dependencies, auto-regressive models including recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and attention networks [Sutskever *et al.*, 2014; Kim *et al.*, 2017] order the output variables and predict one variable at a time conditioned on the previous variables. This ordered sequence of output variables is typically generated using a decoding procedure based on greedy search or beam search. During the training phase, prediction of each variable is conditioned on the ground truth values of the previous output variables. As a result, the model is never ex-

posed to its own error during training. This phenomenon is referred as *exposure bias* problem. To overcome this challenge, prior work has leveraged the general ideas from search-based methods presented in Section 4.1. For example, LaSO approach is leveraged for beam search decoding [Wiseman and Rush, 2016] and advanced imitation learning methods are leveraged for greedy decoding [Bengio *et al.*, 2015]. Another challenge with auto-regressive models is the mismatch between training loss (e.g., token-level cross entropy) and task loss defined over complete structure (e.g., BLEU score in translation). Reinforcement learning (RL) methods are employed with greedy decoding [Ranzato *et al.*, 2015] to alleviate this challenge by training over entire structure using the task loss as a reward function. A recent work [Tan *et al.*, 2018] presented a unified framework of training via both token-level maximum likelihood principle and RL.

Cost function learning with deep models. The two above-mentioned approaches employ a very simple form of structure among the output variables to learn representations: *no dependency structure between output variables* in decoupled integration and *linear ordering among output variables* in search-based learning. Therefore, these methods impose excessively strict inductive bias. To overcome these drawbacks, recent work explored tight integration of deep models with cost function learning approaches that were discussed in section 3. Due to lack of space, we only provide some representative examples. Deep SP framework replaces clique potentials with a deep network and approximates the partition function with loopy Belief Propagation [Chen *et al.*, 2015]. Structured prediction energy networks (SPENs) framework allows us to learn a non-linear cost function over structured input-output pairs in the structured SVM training regime [Belanger *et al.*, 2017]. SPENs fall within the general framework of energy-based learning [LeCun *et al.*, 2006]. Deep value networks (DVNs) learn a non-linear regressor to approximate the negative loss value of a candidate structured output [Gygli *et al.*, 2017]. Both SPENs and DVNs employs gradient-based inference in the relaxed continuous space. Approximate inference networks (InfNet) method [Tu and Gimpel, 2018] employs an additional trained deep neural network to directly produce the output of inference problem with a given cost function. A recent work [Graber *et al.*, 2018] developed a generalization of SPENs with improved results.

Outstanding challenges. Some important challenges for methods that integrate deep models for structured prediction are as follows. *a) Incorporating constraints:* Many SP tasks require the structured outputs to satisfy some constraints (e.g., valid trees in parsing). Incorporating these constraints is a major challenge for methods that perform gradient-based inference and learning [Lee *et al.*, 2017]. These constraints can be classified into three main categories: relational, logical, and scientific. Relational constraints enforce simple relations among entities which can be specified manually or mined from large amounts of unstructured text available on web. Logical constraints occur in domains where structured output variables are related by logical propositions [Xu *et al.*, 2017]. Little attention has been paid to scientific constraints which require the predicted outputs to satisfy the true dynamics of our world based on physics [Stewart and Ermon, 2017]. These constraints can also be thought of as prior knowledge for DL models and can improve their sample-efficiency. *b) Stability and Robustness:* The training of DL models in SP is prone to unstability as discussed in earlier version of SPENs. Training of SPENs was improved [Belanger *et al.*, 2017], but this approach is prone to over-fitting. In our own experience, performance of DVNs is very sensitive to the parameters of gradient-based inference. Deep SP with non-linear output transformations approach [Graber *et al.*, 2018] doesn't support variable length structured outputs yet.

6 Future Directions

In this section, we briefly discuss three important future directions in structured prediction research.

Structured learning with weak supervision. Since acquiring large amount of labeled data is resource-expensive for many real-world SP tasks, there is a great need to study SP algorithms with indirect and weak supervision [Chang *et al.*, 2010; Roth, 2017]. Prior work has studied algorithms for leveraging some forms of weak supervision. Posterior Regularization (PR) [Ganchev *et al.*, 2010] is a general framework to leverage side-information (e.g., constraints) and has shown to be successful in multiple domains. The key idea behind this approach is to restrict the posterior distribution within a space of distributions which satisfies the constraints in expectation. The PR framework also provides a unifying perspective and shows the connections to closely related approaches like constraint-driven learning, generalized expectation criteria, and bayesian measurements. However, the most general paradigm to deal with the difficulty of obtaining labeled structures for training is to consider a simple derivative of the structured representation, one that is easy to supervise, and that depends on the quality of the structured representation. For example, a derivative for the aforementioned POS tagging task could be the question of whether a given sequence of words has a corresponding sequence of POS tags that is "legitimate", which is easy to supervise. A derivative for a semantic parsing problem could be the correctness of response from a database where the semantic parse is used to convert a natural language query into a database query [Clarke *et al.*, 2010]. Many open problem exist within this formulation, and addressing these is likely to facilitate more scalable and realistic structured prediction.

Active learning for structured prediction. Active learning allows the learner to ask labeling queries to improve the sample-efficiency of learning. Prior work has studied active learning algorithms for structured prediction [Culotta and McCallum, 2005; Roth and Small, 2006; Tosh and Dasgupta, 2018]. They differ in the SP framework (e.g., CRFs, Structured SVM); query form (e.g., labeling complete structured outputs, single output variable); and query selection strategies (e.g., uncertainty reduction, margin-based). Recent work [Ning *et al.*, 2019] has shown that querying labels for partial structures and learning from them allows us to use the available query budget better. Co-active learning [Shivaswamy and Joachims, 2012] allows the human to observe the predicted output and provide a slightly corrected output (better

than prediction but not optimal) as feedback to improve the model. Future work should explore alternate query forms that are easy for humans to answer and still provide useful training signal for SP models; and good human-computer interfaces to improve the usability of interactive learning systems.

Multi-Task structured prediction. Prior work on structured prediction mostly considers single tasks. However, interpreting unstructured data (e.g., text, images, videos, speech, and sensor data) requires solving multiple SP tasks jointly. For example, entity analysis in natural language processing involves solving multiple structured prediction problems such as mention detection, coreference resolution, and entity linking. The most common approach is to employ a "pipeline" architecture, where the different tasks are solved one after another in sequence. While it has the advantages of simplicity and scalability, the pipeline architecture is too sensitive to the task order and is prone to error propagation. There is some recent work on exploring scalable joint modeling approaches to solve multi-task SP problems [Durrett and Klein, 2014; Ma *et al.*, 2017; Sanh *et al.*, 2018], but this is a very rich and fertile problem space with many challenges.

Acknowledgements

This work was supported in part by NSF grants #1910213 and #1845922, and in part by Contracts W911NF15-1-0461 and HR0011-17-S-0048 with the US Defense Advanced Research Projects Agency (DARPA). The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[Belanger *et al.*, 2017] David Belanger, Bishan Yang, and Andrew McCallum. End-to-End Learning for Structured Prediction Energy Networks. In *ICML*, pages 429–439, 2017.

[Bengio *et al.*, 2015] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *NIPS*, 2015.

[Beygelzimer *et al.*, 2016] Alina Beygelzimer, Hal Daumé III, John Langford, and Paul Mineiro. Learning Reductions That Really Work. *Proceedings of the IEEE*, 104(1), 2016.

[Chang *et al.*, 2010] M. Chang, V. Srikumar, D. Goldwasser, and D. Roth. Structured Output Learning with Indirect Supervision. In *ICML*, pages 199–206, 2010.

[Chang *et al.*, 2012] Ming-Wei Chang, Lev-Arie Ratinov, and Dan Roth. Structured Learning with Constrained Conditional Models. *MLJ*, 88(3), 2012.

[Chang *et al.*, 2015a] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to Search Better than Your Teacher. In *ICML*, 2015.

[Chang *et al.*, 2015b] Kai-Wei Chang, Shyam Upadhyay, Gourab Kundu, and Dan Roth. Structural Learning with Amortized Inference. In *AAAI*, 2015.

[Chen *et al.*, 2015] Liang-Chieh Chen, Alexander G. Schwing, Alan L. Yuille, and Raquel Urtasun. Learning Deep Structured Models. In *ICML*, pages 1785–1794, 2015.

[Clarke *et al.*, 2010] J. Clarke, D. Goldwasser, M. Chang, and D. Roth. Driving Semantic Parsing from the World's Response. In *CoNLL*, pages 18–27, 2010.

[Collins and Roark, 2004] Michael Collins and Brian Roark. Incremental Parsing with the Perceptron Algorithm. In *ACL*, 2004.

[Collins, 2002] Michael Collins. Ranking Algorithms for Named Entity Extraction: Boosting and the Voted Perceptron. In *ACL*, 2002.

[Cortes *et al.*, 2016] Corinna Cortes, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Structured Prediction Theory Based on Factor Graph Complexity. In *NIPS*, pages 2514–2522, 2016.

[Culotta and McCallum, 2005] Aron Culotta and Andrew McCallum. Reducing Labeling Effort for Structured Prediction Tasks. In *AAAI*, 2005.

[Daumé III and Marcu, 2005] Hal Daumé III and Daniel Marcu. Learning as Search Optimization: Approximate Large Margin methods for Structured Prediction. In *ICML*, 2005.

[Daumé III *et al.*, 2009] Hal Daumé III, John Langford, and Daniel Marcu. Search-based Structured Prediction. *MLJ*, 75(3), 2009.

[Doppa *et al.*, 2014a] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. HC-Search: A Learning Framework for Search-based Structured Prediction. *JAIR*, 50:369–407, 2014.

[Doppa *et al.*, 2014b] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured Prediction via Output Space Search. *JMLR*, 15(1):1317–1350, 2014.

[Doppa *et al.*, 2014c] Janardhan Rao Doppa, Jun Yu, Chao Ma, Alan Fern, and Prasad Tadepalli. HC-Search for Multi-Label Prediction: An Empirical Study. In *AAAI*, 2014.

[Durrett and Klein, 2014] Greg Durrett and Dan Klein. A Joint Model for Entity Analysis: Coreference, Typing, and Linking. In *TACL*, volume 2, pages 477–490, 2014.

[Felzenszwalb and McAllester, 2007] Pedro F. Felzenszwalb and David A. McAllester. The Generalized A* Architecture. *JAIR*, 29:153–190, 2007.

[Fern, 2010] Alan Fern. Speedup learning. In *Encyclopedia of Machine Learning*, pages 907–911. 2010.

[Finley and Joachims, 2008] Thomas Finley and Thorsten Joachims. Training Structural SVMs when Exact Inference is Intractable. In *ICML*, pages 304–311, 2008.

[Ganchev *et al.*, 2010] Kuzman Ganchev, Jennifer Gillenwater, Ben Taskar, et al. Posterior Regularization for Structured Latent Variable Models. *JMLR*, 11(Jul):2001–2049, 2010.

[Goldberg and Elhadad, 2010] Yoav Goldberg and Michael Elhadad. An Efficient Algorithm for Easy-first Non-directional Dependency Parsing. In *NAACL*, pages 742–750, 2010.

[Graber *et al.*, 2018] Colin Graber, Ofer Meshi, and Alexander Schwing. Deep Structured Prediction with Nonlinear output Transformations. In *NIPS*, pages 6320–6331, 2018.

[Gygli *et al.*, 2017] Michael Gygli, Mohammad Norouzi, and Anelia Angelova. Deep Value Networks Learn to Evaluate and Iteratively Refine Structured outputs. In *ICML*, 2017.

[Hazan *et al.*, 2016] Tamir Hazan, Alexander G. Schwing, and Raquel Urtasun. Blending Learning and Inference in Conditional Random Fields. *JMLR*, 17(1):8305–8329, 2016.

[Huang *et al.*, 2012] Liang Huang, Suphan Fayong, and Yang Guo. Structured Perceptron with Inexact Search. In *NAACL*, 2012.

[Huang *et al.*, 2015] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[Kim *et al.*, 2017] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured Attention Networks. In *ICLR*, 2017.

[Kulesza and Pereira, 2007] Alex Kulesza and Fernando Pereira. Structured Learning with Approximate Inference. In *NIPS*, 2007.

[Lafferty *et al.*, 2001] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *ICML*, 2001.

[Lam *et al.*, 2015] Michael Lam, Janardhan Rao Doppa, Sinisa Todorovic, and Thomas G. Dietterich. HC-Search for Structured Prediction in Computer Vision. In *CVPR*, 2015.

[Le *et al.*, 2017] Hoang Minh Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated Multi-agent Imitation Learning. In *ICML*, pages 1995–2003, 2017.

[LeCun *et al.*, 2006] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A Tutorial on Energy-based Learning. *Predicting structured data*, 2006.

[Lee *et al.*, 2017] Jay Yoon Lee, Michael Wick, Sanket Vaibhav Mehta, Jean-Baptiste Tristan, and Jaime Carbonell. Gradient-based inference for networks with output constraints. *arXiv preprint arXiv:1707.08608*, 2017.

[Liang *et al.*, 2008] Percy Liang, Hal Daumé III, and Dan Klein. Structure Compilation: Trading Structure for Features. In *ICML*, pages 592–599, 2008.

[Ma *et al.*, 2017] Chao Ma, Janardhan Rao Doppa, Prasad Tadepalli, Hamed Shahbazi, and Xiaoli Z. Fern. Multi-Task Structured Prediction for Entity Analysis: Search-Based Learning Algorithms. In *ACML*, 2017.

[Meshi *et al.*, 2016] Ofer Meshi, Mehrdad Mahdavi, Adrian Weller, and David Sontag. Train and Test Tightness of LP Relaxations in Structured Prediction. 2016.

[Ning *et al.*, 2019] Qiang Ning, Hangfeng He, Chuchu Fan, and Dan Roth. Partial or Complete, That's The Question. In *NAACL*, 2019.

[Ranzato *et al.*, 2015] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence Level Training with Recurrent Neural Networks. *arXiv preprint arXiv:1511.06732*, 2015.

[Ross *et al.*, 2011] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *AISTATS*, 2011.

[Roth and Small, 2006] Dan Roth and Kevin Small. Margin-Based Active Learning for Structured Output Spaces. In *ECML*, 2006.

[Roth and Yih, 2004] D. Roth and W. Yih. A Linear Programming Formulation for Global Inference in Natural Language Tasks. In *CoNLL*, 2004.

[Roth, 2017] Dan Roth. Incidental Supervision: Moving beyond Supervised Learning. In *AAAI*, 2017.

[Samdani and Roth, 2012] Rajhans Samdani and Dan Roth. Efficient Decomposed Learning for Structured Prediction. In *ICML*, 2012.

[Sanh *et al.*, 2018] Victor Sanh, Thomas Wolf, and Sebastian Ruder. A Hierarchical Multi-task Approach for Learning Embeddings from Semantic Tasks. *CoRR*, abs/1811.06031, 2018.

[Shivaswamy and Joachims, 2012] Pannaga Shivaswamy and Thorsten Joachims. Online Structured Prediction via Coactive Learning. In *ICML*, 2012.

[Sontag *et al.*, 2010] David Sontag, Ofer Meshi, Tommi S. Jaakkola, and Amir Globerson. More Data means Less Inference: A Pseudo-max approach to Structured Learning. In *NIPS*, pages 2181–2189, 2010.

[Srikumar *et al.*, 2012] Vivek Srikumar, Gourab Kundu, and Dan Roth. On Amortizing Inference Cost for Structured Prediction. In *EMNLP*, pages 1114–1124, 2012.

[Stewart and Ermon, 2017] Russell Stewart and Stefano Ermon. Label-free Supervision of Neural Networks with Physics and Domain Knowledge. In *AAAI*, 2017.

[Stoyanov and Eisner, 2012] Veselin Stoyanov and Jason Eisner. Easy-first Coreference Resolution. In *COLING*, 2012.

[Stoyanov *et al.*, 2011] Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical Risk Minimization of Graphical Model Parameters Given Approximate Inference, Decoding, and Model Structure. In *AISTATS*, 2011.

[Sutskever *et al.*, 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In *NIPS*, pages 3104–3112, 2014.

[Tan *et al.*, 2018] Bowen Tan, Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric Xing. Connecting the Dots Between MLE and RL for Sequence Generation. *arXiv preprint arXiv:1811.09740*, 2018.

[Taskar *et al.*, 2003] Benjamin Taskar, Carlos Guestrin, and Daphne Koller. Max-Margin Markov Networks. In *NIPS*, 2003.

[Tosh and Dasgupta, 2018] Christopher Tosh and Sanjoy Dasgupta. Interactive Structure Learning with Structural Query-by-Committee. In *NeurIPS*, pages 1121–1131, 2018.

[Tsochantaridis *et al.*, 2004] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *ICML*, page 104, 2004.

[Tu and Gimpel, 2018] Lifu Tu and Kevin Gimpel. Learning Approximate Inference Networks for Structured Prediction. In *ICLR*, 2018.

[Venkatraman *et al.*, 2015] Arun Venkatraman, Martial Hebert, and J. Andrew Bagnell. Improving Multi-Step Prediction of Learned Time Series Models. In *AAAI*, 2015.

[Weiss and Taskar, 2010] David Weiss and Benjamin Taskar. Structured Prediction Cascades. In *AISTATS*, 2010.

[Wiseman and Rush, 2016] Sam Wiseman and Alexander M. Rush. Sequence-to-Sequence Learning as Beam-Search Optimization. In *EMNLP*, pages 1296–1306, 2016.

[Xie *et al.*, 2015] Jun Xie, Chao Ma, Janardhan Rao Doppa, Prashanth Mannem, Xiaoli Z. Fern, Thomas G. Dietterich, and Prasad Tadepalli. Learning Greedy Policies for the Easy-First Framework. In *AAAI*, 2015.

[Xu *et al.*, 2009] Yuehua Xu, Alan Fern, and Sung Wook Yoon. Learning Linear Ranking Functions for Beam Search with Application to planning. *JMLR*, 10(Jul):1571–1610, 2009.

[Xu *et al.*, 2017] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. *arXiv preprint arXiv:1711.11157*, 2017.

[Yadollahpour *et al.*, 2013] Payman Yadollahpour, Dhruv Batra, and Gregory Shakhnarovich. Discriminative Re-ranking of Diverse Segmentations. In *CVPR*, pages 1923–1930, 2013.