Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Structural Learning with Amortized Inference

Kai-Wei Chang, Shyam Upadhyay, Gourab Kundu, and Dan Roth
Department of Computer Science
University of Illinois at Urbana-Champaign
{kchang10,upadhya3,kundu2,danr}@illinois.edu

Abstract

Training a structured prediction model involves performing several loss-augmented inference steps. Over the lifetime of the training, many of these inference problems, although different, share the same solution. We propose AI-DCD, an **A**mortized **I**nference framework for **D**ual **C**oordinate **D**escent method, an approximate learning algorithm, that accelerates the training process by exploiting this redundancy of solutions, *without* compromising the performance of the model. We show the efficacy of our method by training a structured SVM using dual coordinate descent for an entity-relation extraction task. Our method learns the same model as an exact training algorithm would, but call the inference engine only in 10% – 24% of the inference problems encountered during training. We observe similar gains on a multi-label classification task and with a Structured Perceptron model for the entity-relation task.

Introduction

The desired output in many machine learning tasks is a structured object such as a tree, an alignment of nodes or a sequence. Learning prediction models for such problems is considerably difficult, because of the inter-dependence of the output variables comprising the structure. In the past decade, several *structured learning models* have been proposed (Collins 2002; Taskar, Guestrin, and Koller 2004; Tsochantaridis et al. 2005). Training these models usually involves performing an *inference step* repeatedly, where the algorithm finds the best structure according to the current model for a training example.

Existing learning algorithms (Tsochantaridis et al. 2005; Joachims, Finley, and Yu 2009; Shevade et al. 2011; Chang and Yih 2013; Lacoste-Julien et al. 2013) often treat the inference procedure as a black-box and solve the inference problems encountered during training *independently*. However, a learning algorithm makes multiple passes over the training examples, and updates its parameters for each example several times. Consequently, the inference problems it encounters may often have identical or similar solutions. To verify this intuition, we trained a structured SVM model on a joint entity-relation recognition task (Roth and Yih 2007) and plotted the numbers of inference engine calls (black solid line) made in Figure 1. As shown in the figure, a large fraction of these calls returns identical solutions, as indicated by the blue dotted line, even when the objective functions of the inference problems are different. This suggests the possibility of amortizing these inference steps (see details in caption of Figure 1).

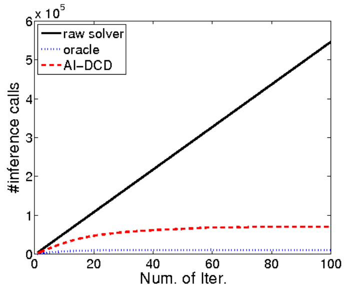

Figure 1: The number of inference engine calls by a raw solver, an oracle solver and our method when training a structured SVM model on an entity-relation recognition corpus. The raw solver calls an inference engine each time an inference is needed, while the oracle solver only calls the engine when the solution is different from all solutions observed previously. Our method significantly reduces the number of inference calls required to train a model.

In this paper, we propose to *amortize the inferences during learning*, accelerating the training process of a structured model. That is, we measure the overall cost of inference over the life time of the algorithm, and show that this "amortized" cost can be significantly reduced. Our framework only depends on the fact that the inference is formulated as an Integer Linear Program (ILP), a very general condition, as discussed below, that does not restrict its applicability. Given a tolerance parameter ϵ, we derive a condition which, when satisfied, guarantees that two *different* inference problems posed as ILPs will yield similar solutions, such that the optimal solution of one problem is an $(1 + M\epsilon)$-approximation of the other (here M is a problem specific constant). We show that this approximation enjoys sound theoretical guarantees. We further consider adjusting ϵ during training, and provide guarantees that a dual coordinate descent solver (DCD) with amortized inference converges to the same optimal solution as in the case where exact inference is used. We provide sound theoretical properties of the proposed methods and show empirically (Figure 1), that our framework can significantly reduce the number of inference calls required for reaching a certain level of performance (red dashed lines).

Amortized inference techniques for structured prediction have been studied in (Srikumar, Kundu, and Roth 2012; Kundu, Srikumar, and Roth 2013), where the authors amortized the inference calls at *test time*. However, using amortization during training is more challenging because 1) in the test phase, a fixed model is used, while in the training phase, the model changes gradually and therefore the inference problems show more variability in terms of solutions; 2) in the test phase, a cache can be precomputed using a large corpus, while in the training phase, the cache has to be accumulated in an online fashion.

Our learning framework is related to approximate training paradigms (Finley and Joachims 2008; Meshi et al. 2010; Hazan and Urtasun 2010; Sutton and McCallum 2007; Samdani and Roth 2012), but with some crucial differences. These methods use approximate inference to find suboptimal solutions to otherwise intractable inferences problems. In contrast, we focus on scenarios where exact solutions can be computed, developing an approximate scheme to accelerate training without affecting the solution quality. Caching inference solutions has been discussed in a different context for 1-slack cutting plane method (Joachims, Finley, and Yu 2009), where cached solutions can be used to generate a new cutting-plane. We will discuss this in later sections.

The proposed framework is general for several reasons. First, the formulation of the inference step as an ILP problem is extremely general. Many inference problems in natural language processing, vision and other fields can be cast as ILP problems (Roth and Yih 2004; Clarke and Lapata 2006; Riedel and Clarke 2006). In fact, all discrete MPE problems can be cast as ILPs (Roth and Yih 2004; Sontag 2010). Second, our framework is independent of the inference engine used to solve the inference problems but it can maintain the exactness (or approximation) guarantees of the solver chosen by the user. Third, amortized inference can be plugged into any learning framework which requires solving inference repeatedly. (e.g., (Joachims, Finley, and Yu 2009; Lacoste-Julien et al. 2013)).

Amortized Inference

Let $\boldsymbol{y} = \{y_1, y_2, \cdots y_N\}$ be a structure where $y_j \in \mathcal{Y}_j$, a discrete set of candidates for the output variable y_j, and $\boldsymbol{y} \in \mathcal{Y}$, a set of feasible structures. The inference problem in structured prediction attempts to find the best structure $\boldsymbol{y}^* \in \mathcal{Y}$ for $\boldsymbol{x}$, according to a given model $\boldsymbol{w}$

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \quad \boldsymbol{w}^T\phi(\boldsymbol{x}, \boldsymbol{y}). \tag{1}$$

$\phi(\boldsymbol{x}, \boldsymbol{y})$ is a feature vector extracted from both input $\boldsymbol{x}$ and output structure $\boldsymbol{y}$. One can use a binary vector $\mathbf{z} = \{z_{j,a} \mid j = 1 \cdots N, a \in \mathcal{Y}_j\}, \mathbf{z} \in \mathcal{Z} \subset \{0,1\}^{\sum|\mathcal{Y}_i|}$ to represent the output variable $\boldsymbol{y}$, where $z_{j,a}$ denotes whether y_j assumes the value a or not. This way (1) can be reformulated as a binary integer linear program (ILP) as noted in (Roth and Yih 2004; 2007), where linear constraints are added to enforce problem-specific constraints on legitimate $\mathbf{z}$:

$$\mathbf{z}^* = \arg\max_{\mathbf{z}\in\mathcal{Z}} \quad \mathbf{c}^T\mathbf{z}, \tag{2}$$

where $\mathbf{c}^T\mathbf{z} = \boldsymbol{w}^T\phi(\boldsymbol{x}_i, \boldsymbol{y})$, $\sum_a z_{j,a} = 1$ and $c_{j,a}$ is the coefficient corresponding to $z_{j,a}$.

Some specific output structures (e.g., linear chain structure) allow the inference to be cast as a search problem, for which efficient exact algorithms are readily available (e.g., Viterbi algorithm). However, the inference in general structured prediction problem is usually computationally expensive. Srikumar, Kundu, and Roth (2012) observed that (1) different objective functions often yield the same solution, and (2) only a small fraction of the exponentially many possible structures is actually seen in practice. They showed that significant speed-up can be obtained for inference problems posed as ILPs (Eq. (2)), by characterizing the set of ILP objectives which share the same solution. The following theorem generalizes Theorem 1 from (Srikumar, Kundu, and Roth 2012).

Theorem 1 *Let* $\mathbf{p}$ *and* $\mathbf{q}$ *be inference problems that share the same feasible set and have the same number of variables. Let* $\boldsymbol{y}^{\mathbf{p}} = \{y_1^{\mathbf{p}}, y_2^{\mathbf{p}}, ...y_N^{\mathbf{p}}\}$ *and* $\boldsymbol{y}^{\mathbf{q}} = \{y_1^{\mathbf{q}}, y_2^{\mathbf{q}}, ...y_N^{\mathbf{q}}\}$ *be the optimal solutions to the inference problems* $\mathbf{p}$ *and* $\mathbf{q}$ *respectively, with corresponding binary representations* $\mathbf{z}^{\mathbf{p}}$ *and* $\mathbf{z}^{\mathbf{q}}$*. Denote the objective function of* $\mathbf{p}$ *by* $f^{\mathbf{p}}(\mathbf{z}) = \mathbf{c}^{\mathbf{p}} \cdot \mathbf{z}$*, and of* $\mathbf{q}$ *by* $f^{\mathbf{q}}(\mathbf{z}) = \mathbf{c}^{\mathbf{q}} \cdot \mathbf{z}$*. W.l.o.g, assume* $f^{\mathbf{q}}(\mathbf{z}^{\mathbf{p}}) \geq 0$ *and* $f^{\mathbf{q}}(\mathbf{z}^{\mathbf{q}}) \geq 0$*. If there exists a constant* $M \in \mathcal{R}^+$ *such that*

$$M \geq \sum\nolimits_j (|c^{\mathbf{q}}_{j,y_j^{\mathbf{p}}}| + |c^{\mathbf{q}}_{j,y_j^{\mathbf{q}}}|)/f^{\mathbf{q}}(\mathbf{z}^{\mathbf{p}}) \tag{3}$$

and $\mathbf{c}^{\mathbf{p}}$ *and* $\mathbf{c}^{\mathbf{q}}$ *satisfy the following condition:*

$$(2z^{\mathbf{p}}_{j,a} - 1)(c^{\mathbf{p}}_{j,a} - c^{\mathbf{q}}_{j,a}) \leq \epsilon|c^{\mathbf{q}}_{j,a}|, \quad \forall j, a \tag{4}$$

then $\mathbf{z}^{\mathbf{p}}$ *is an* $(1/(1+M\epsilon))$*-approximation to the inference problem* $\mathbf{q}$*. That is,* $f^{\mathbf{q}}(\mathbf{z}^{\mathbf{q}}) \leq (1+M\epsilon)f^{\mathbf{q}}(\mathbf{z}^{\mathbf{p}})$*.*

To use this theorem, one would store a large cache of $(\mathbf{p}, \mathbf{c}^{\mathbf{p}})$ pairs with their solutions. Upon encountering a new inference problem $\mathbf{q}$, we look in the cache for $\mathbf{p}$ that satisfies (4). We thus obtain a $(1/(1+M\epsilon))$-approximate solution to $\mathbf{q}$, saving a call to the inference engine. We later show that such approximate solutions suffice for training purposes. Note that this scheme is oblivious to the inference engine used. When the tolerance parameter $\epsilon = 0$, the solution obtained from the above process is exact. Otherwise, a large ϵ increases the number of inference problems that can be amortized, but also amplify the risk of obtaining suboptimal solutions.[1] The theorem assumes the optimal objective function value of $\mathbf{q}$ is positive. If the assumption is not satisfied, one can add a dummy variable with a large coefficient, and the theorem holds. Note that constructing an ILP problem for (2) and checking condition (4) are usually cheaper than solving Eq. (1).

[1] Srikumar, Kundu, and Roth (2012) proposed to use $(2z^{\mathbf{p}}_{j,a} - 1)(c^{\mathbf{p}}_{j,a} - c^{\mathbf{q}}_{j,a}) \leq \epsilon, \quad \forall j, a$ for approximate amortized inference without providing a guarantee. Applying a similar derivation, we can prove that $f^{\mathbf{q}}(\mathbf{z}^q) - f^{\mathbf{p}}(\mathbf{z}^p) \leq 2\epsilon N$ under their condition, where N is the number of output variables.

Learning with Amortized Inference

When training a structured prediction model, an inference engine repeatedly solves (7) for every training instance. This step is, in general, computationally expensive. Although ways to reduce the number of inference calls during training have been studied, they often treat this inference step as a black box, and overlook the possibility of amortizing the inference step. In the following, we present **AI-DCD**, an **A**mortized **I**nference framework for **D**ual **C**oordinate **D**escent method that accelerates the training process by amortizing the inference problems. We also apply the same framework to Structured Perceptron and refer to it as **AI-SP**(**A**mortized **I**nference for **S**tructured **P**erceptron).

Structured SVM with Amortized Inference

We start by incorporating amortized inference technique in training a structured SVM model with dual coordinate descent (DCD) (Chang and Yih 2013). We are given a set of annotated training instances $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{l}, \boldsymbol{x}_i \in \mathcal{X}, \boldsymbol{y}_i \in \mathcal{Y}$. Training a structured SVM model $\boldsymbol{w}$ involves solving the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\xi}} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum\nolimits_i \ell(\xi_i) \\ &\quad \text{s.t. } \ \boldsymbol{w}^T\phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i) \ge \Delta(\boldsymbol{y}_i, \boldsymbol{y}) - \xi_i, \ \forall i, \boldsymbol{y} \in \mathcal{Y}, \end{aligned} \tag{5}$$

where $\phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i) = \phi(\boldsymbol{y}_i, \boldsymbol{x}_i) - \phi(\boldsymbol{y}, \boldsymbol{x}_i)$. We use $\ell(\xi) = \xi^2$ (L2-loss) here. $\Delta(\boldsymbol{y}_i, \boldsymbol{y})$ is a loss function that measures the difference between the gold structure $\boldsymbol{y}_i$ and the predicted structure $\boldsymbol{y}$. DCD solves the dual form of (5):

$$\begin{aligned} &\min_{\boldsymbol{\alpha} \ge 0} \frac{1}{2} \left\| \sum\nolimits_{\alpha_{i,\boldsymbol{y}}} \alpha_{i,\boldsymbol{y}} \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i) \right\|^2 \\ &\qquad + \frac{1}{4C} \sum\nolimits_i \Big(\sum\nolimits_{\boldsymbol{y}} \alpha_{i,\boldsymbol{y}} \Big)^2 - \sum\nolimits_{i,\boldsymbol{y}} \Delta(\boldsymbol{y}, \boldsymbol{y}_i) \alpha_{i,\boldsymbol{y}}. \end{aligned} \tag{6}$$

We define the objective function in Eq. (6) to be $D(\boldsymbol{\alpha})$. Each dual variable $\alpha_{i,\boldsymbol{y}}$ corresponds to a feasible output structure $\boldsymbol{y}$ and an example $\boldsymbol{x}_i$. The dual optimum $\boldsymbol{\alpha}^*$ and the primal optimum $\boldsymbol{w}^*$ have the relationship $\boldsymbol{w}^* = \sum_{i,\boldsymbol{y}} \alpha^*_{i,\boldsymbol{y}} \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i)$. In a linear model, we can maintain a temporary vector $\boldsymbol{w} \equiv \sum_{i,\boldsymbol{y}} \alpha_{i,\boldsymbol{y}} \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i)$ while solving (6) to assist the computations (Hsieh et al. 2008).

In general, the number of variables in (6) is exponentially large. Therefore, the dual method maintains an active set $\mathcal{A}$ and only updates $\alpha_{i,\boldsymbol{y}} \in \mathcal{A}$ while fixing the rest $\alpha_{i,\boldsymbol{y}} \notin \mathcal{A}$ to 0. To select a dual variable for update, DCD solves the following optimization problem for each example $\boldsymbol{x}_i$:

$$\begin{aligned} \bar{\boldsymbol{y}} &= \arg\max\nolimits_{\boldsymbol{y} \in \mathcal{Y}} \quad (-\nabla D(\boldsymbol{\alpha}))_{i,\boldsymbol{y}} \\ &= \arg\max\nolimits_{\boldsymbol{y} \in \mathcal{Y}} \quad \boldsymbol{w}^T \phi(\boldsymbol{x}_i, \boldsymbol{y}) + \Delta(\boldsymbol{y}_i, \boldsymbol{y}) \end{aligned} \tag{7}$$

and adds $\alpha_{i,\bar{\boldsymbol{y}}}$ into $\mathcal{A}$ if

$$-\nabla D(\boldsymbol{\alpha})_{i,\bar{\boldsymbol{y}}} > \delta, \tag{8}$$

where δ is a user specified tolerance parameter. Eq. (7) is often referred to as a loss-augmented inference problem. In practice, one may choose a loss $\Delta(\boldsymbol{y}_i, \boldsymbol{y})$ such that Eq. (7) can be formulated as an ILP problem (2).

Algorithm 1 A Dual Coordinate Descent Method with Amortized Inference

```
$\boldsymbol{w} \leftarrow \mathbf{0}, \boldsymbol{\alpha} \leftarrow \mathbf{0}, \mathcal{A} = \emptyset$, and an initial cache $\Omega = \emptyset$
while stopping conditions are not satisfied do
    for all $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ (loop over each instance) do
        $\bar{\boldsymbol{y}} \leftarrow$ InferenceSolver($\mathbf{q}, \Omega, \bar{\epsilon}$), where $\mathbf{q}$ be the ILP of Eq. (7) with $\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i$.
        Add $(\bar{\boldsymbol{y}}, \mathbf{q})$ to $\Omega$.
        if $\alpha_{i,\bar{\boldsymbol{y}}} \notin \mathcal{A}$ and $-\nabla D(\boldsymbol{\alpha})_{i,\bar{\boldsymbol{y}}} > \delta$, holds then
            $\mathcal{A} \leftarrow \mathcal{A} \cup \{\alpha_{i,\bar{\boldsymbol{y}}}\}$.
    while inner stopping conditions are not satisfied do
        update $\alpha_{i,\boldsymbol{y}}$ and $\boldsymbol{w}$ by Eq. (10) for $\alpha_{i,\boldsymbol{y}} \in \mathcal{A}$.
return $\boldsymbol{w}$

Function: InferenceSolver($\mathbf{q}, \Omega, \bar{\epsilon}$):
for all $\mathbf{p} \in \Omega$ and $\mathbf{p} \in [Q]$, where $[\cdot]$ is the set of all inference problems with same feasible set. do
    if condition (4) holds (with $\epsilon$ set to be $\bar{\epsilon}$) then
        return $\boldsymbol{y}_{\mathbf{p}}$
return $\boldsymbol{y}_{\mathbf{q}}$ by solving $\mathbf{q}$ using an ILP solver.
```

Starting from an initial cache $\Omega = \Omega_0$, AI-DCD maintains a cache that stores the augmented-loss inference problems involved in the training process and their corresponding solution. The training process generates a sequence of models $\boldsymbol{w}^0, \boldsymbol{w}^1, \cdots, \boldsymbol{w}^*$ and alternates between two phases 1) the inference phase and 2) the model update phase. The procedure is described in Algorithm 1.

Inference Phase. In the inference phase, AI-DCD loops over each training instance and chooses active variables based on Eq. (7). We pose Eq. (7) as an ILP problem $\mathbf{q}$. Then for each entry $\mathbf{p} \in [Q]$ in the cache, if condition (4) holds, we assign $\boldsymbol{y}_{\mathbf{p}}$ as the solution to $\mathbf{q}$. Here $[Q]$ is the set of all inference problems with the same feasible set as problem $\mathbf{q}$. Otherwise, we solve $\mathbf{q}$ by calling the inference engine. Once we obtain the solution to $\mathbf{q}$, we add $\boldsymbol{y}_{\mathbf{q}}$ to the cache for future iterations. This way, we populate the solution cache in an online fashion during training.

To reduce the overhead of verifying Eq. (4), we partition the problems in $[Q]$ into groups based on their optimal solution. When solving $\mathbf{q}$, we first find

$$\boldsymbol{y}' = \arg\max\nolimits_{\boldsymbol{y} \in \{\boldsymbol{y}_{\mathbf{p}} | \mathbf{p} \in [Q]\}} \quad \text{objective of } \mathbf{q}. \tag{9}$$

and then we only check cached problems whose optimal solution is $\boldsymbol{y}'$. This approach is related to the caching heuristic in (Joachims, Finley, and Yu 2009) (see later section).

Model Update Phase. In the model update phase, we sequentially visit $\alpha_{i,\boldsymbol{y}} \in \mathcal{A}$ and update the model based on solving a one-dimensional sub-problem:

$$\begin{aligned} &\bar{d}_{i,\boldsymbol{y}} = \arg\min\nolimits_{d \in \mathcal{R}} D(\boldsymbol{\alpha} + d\mathbf{e}_{i,\boldsymbol{y}}) \ \text{ s.t. } \ \alpha_{i,\boldsymbol{y}} + d \ge 0 \\ &\alpha_{i,\boldsymbol{y}} \leftarrow \alpha_{i,\boldsymbol{y}} + \bar{d}_{i,\boldsymbol{y}}, \ \text{ and } \boldsymbol{w} \leftarrow \boldsymbol{w} + \bar{d}_{i,\boldsymbol{y}} \phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i). \end{aligned} \tag{10}$$

Note that we maintain $\mathcal{A}$ throughout the optimization process. Therefore, once an $\alpha_{i,\boldsymbol{y}}$ is added into $\mathcal{A}$, it will be updated at every iteration. To maintain a small size of $\mathcal{A}$, we remove bounded dual variables using a shrinking strategy described in (Chang, Srikumar, and Roth 2013).

Stopping Condition. We follow (Tsochantaridis et al. 2005; Hsieh et al. 2008) to stop the training process if no candidate structure is added during current iteration and if the $\|\cdot\|_\infty$ norm of the projected gradient associated with the α variables in the working set is bounded by δ.

Analysis for AI-DCD. We use the analyses presented in (Finley and Joachims 2008). They consider two types of approximations for a combinatorial optimization problem. An under-generating method (e.g., greedy search) finds a suboptimal feasible solution, while an over-generating method (e.g, LP relaxation) obtains an optimal solution to a superset of the feasible set. Our approximate inference scheme is in the first category. We prove the following theorem.

Theorem 2 *AI-DCD stops after adding $O(\frac{l\Delta^2}{\delta^2}(R^2C+1))$ candidate structures to the working set, where $R = \max_{i,y}\|\phi(\boldsymbol{y}, \boldsymbol{y}_i, \boldsymbol{x}_i)\|$, l is the number of instances, δ is defined in* (8)*, and Δ is the upper bound of $\Delta(y_i, y)$. When the optimization process stops, the empirical risk is bounded.*

Proof Sketch. The polynomial bound of the working set size is independent of the quality of the inference solution. Furthermore, the risk bound can be derived by plugging-in $\rho = 1/(1 + M\epsilon)$ in Theorems 2 and Theorems 3 in (Finley and Joachims 2008).[2]

Getting Exact Learning Model. According to Theorem 2, if we verify condition (4) with $\epsilon = 0$, the solution to Eq. (5) is exact. Then, Algorithm 1 converges to the optimal solution of Eq. (5) as a standard DCD method (Chang and Yih 2013). When $\epsilon > 0$, we only obtain an approximate solution. However, by adapting ϵ along iterations, AI-DCD can provably converge to the optimum of (5). The following theorem provides the conditions:

Theorem 3 *For $T, \tau \in \mathbb{N}$, the model generated by Algorithm 1 converges to the optimal solution of Eq.* (5) *if exact inference is applied at least once every τ successive inference calls after iteration T.*

It is trivial to see that the model is optimal if exact inference is called every time after iteration T, because the objective function of Eq. (6) is convex. However, we relax the condition by only requiring an exact inference call every τ successive inference calls. This can be derived from the fact that the inference procedure is only involved in selecting αs into the working set. A non-zero α in the optimal solution will be selected into the working set eventually via the exact inference call if its projected gradient is larger than 0. Note that, the dual objective function value (6) is monotonically non-increasing whether the inference is exact or not.

Structured Perceptron with Amortized Inference

The same amortized inference technique described above can be applied to Structured Perceptron (Collins 2002). The Structured Perceptron algorithm sequentially visits training examples and solves an inference problem (1) on an example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ with the current model $\boldsymbol{w}$. The updates in the model are based on the best structured output $\bar{\boldsymbol{y}}$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(\phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \bar{\boldsymbol{y}})),$$

where η is a learning rate. Similar to AI-DCD, we can maintain a cache during the training; hence the previous inference problems can be used to amortize later ones.

Kulesza and Pereira (2008) prove that Structured Perceptron with an over-generating inference still enjoys the mistake bound guarantees. However, for an under-generating method, there is no guarantee of convergence even if the data is separable. Note that using a violation-fixing technique, we can still maintain the mistake bound guarantees (Huang, Fayong, and Guo 2012). We leave it for future work.

Experiments and Results

In this section, we demonstrate the results on an entity-relation extraction and a multi-label classification task. Our experiments aim to show the following properties: (1) the amortization techniques significantly reduce the number of inference solver calls, and hence the training time is reduced; (2) with an adaptation on ϵ, the model converges to the same objective function value; (3) with the help of pre-cached examples, we achieve higher amortization ratio and further speed up the training process.

Entity-Relation Extraction The entity-relation extraction task involves the simultaneous identification of entities in a given text and the relations between them. For example, in the sentence *Lee Oswald assassinated JFK in Dallas in 1963*, we should label *Lee Oswald* and *JFK* as PERSON, *Dallas* as a LOCATION and the relation KILL between *Lee Oswald* and *JFK*. The relation should be compatible with the types of entities, i.e. the relation KILL cannot hold between two LOCATION entity. We modeled the prediction as a 0-1 integer linear program, where the binary variables represent the possible assignments. We use the annotated corpus from Roth and Yih (2007), which consists of 5,925 sentences.

Multi-label classification Multi-label classifier predicts a set of proper labels for each instance. When label correlation is modeled, the learning and inference is often intractable. We use a dataset, scene[3], with 6 labels and 1,211 instances to demonstrate the performance of our method.

We implemented our algorithms based on a publicly available Structured SVM package[4] in JAVA and conducted experiments on a machine with Xeon E5-2440 processors. Unless otherwise stated, we show the performance of training Structured SVM model with C = 0.1 with stopping condition $\delta = 0.1$. For Perceptron, we used an averaged Perceptron implementation in JAVA and set the max number of iterations to 100. We use a commercial package, Gurobi, as a base solver to solve the ILP problems.

[2] Finley and Joachims (2008) analyze a cutting plane method with approximate inference by considering only one example and show that this result can be generalized to multiple examples. See Finley and Joachims (2008) for details.

[3] Available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets

[4] http://cogcomp.cs.illinois.edu/page/software_view/JLIS

Entity-Relation Extraction Task

Method	ϵ	Model Type	% Solver Calls	Inf. in Training		Dual Obj.	Performance	
				Time	Speedup		Ent F1	Rel F1
Structured SVM								
Baseline	-	Exact	100%	293	1	895.31	87.69	47.55
AI-DCD	0	Exact	77%	274	1.1x	895.31	87.69	47.55
AI-DCD	**adapt**	**Exact**	**24%**	**162**	**1.8x**	**895.19**	**87.66**	**47.7**
AI-DCD	0.1	Approx.	54%	213	1.4x	895.31	87.69	47.55
AI-DCD	1	Approx.	29%	115	2.5x	894.73	87.66	48.01
AI-DCD	**10**	**Approx.**	**10%**	**60**	**4.8x**	**874.08**	**87.34**	**47.83**
Structured SVM with strict stopping condition								
Baseline		Exact	100%	1,663	1	897.51	87.76	47.40
AI-SP	1	Approx.	5%	246	6.8x	897.50	87.76	47.45
Averaged Structured Perceptron								
Baseline		Exact	100%	705	1	-	89.09	37.44
AI-DCD	1	Approx.	36%	319	2.2x	-	89.02	39.05

Multi-label Classification

Method	ϵ	Model Type	% Solver Calls	Inf. in Training		Dual Obj.	Hamming Loss
				Time	Speedup		
Structured SVM							
Baseline		Exact	100%	1,300	1	219.20	0.609
AI-DCD	adapt.	Exact	1.6%	18	72x	219.18	0.609
AI-DCD	1	Approx.	0.6%	12	108x	219.15	0.606

Table 1: Ratio of inference engine calls, inference time, inference speedup, final negative dual objective function value ($-D(\alpha)$ in Eq. (6)) and the test performance of each method. Time is in seconds. The results show that AI-DCD significantly speeds up the inference during training, while maintaining the quality of the solution. Remarkably, it requires 24% of the inference calls and 55% of the inference time to obtain an exact learning model. The reduction of time is implementation specific.

Learning with Amortized Inference

We compare the following approaches for training a structured SVM model:

- Baseline: the baseline method that calls an inference engine every time.
- AI-DCD ($\epsilon = \rho$): Algorithm 1 using the approximate version of the function InferenceSolver with ϵ set to be ρ. Note that when $\rho = 0$, the InferenceSolver is exact.
- AI-DCD (adaptive ϵ): We consider a simple adaptation strategy. We begin with running Algorithm 1 with $\epsilon = 10$. When the stopping condition is satisfied, we switch to use $\epsilon = 0.1$ and continue the training process. When the stopping condition is satisfied again, we change to use exact inference ($\epsilon = 0$) toward the end.

Similarly, we compare Structured Perceptron to the version with amortized inference (AI-SP).

Table 1 summarizes the results. Checking condition Eq. (4) takes only 0.04 ms, while executing an ILP engine takes 2ms to solve an inference sample in average. Therefore, when a method achieves a higher amortization ratio, it takes less time. When $\delta = 0.1$, using amortized inference with $\epsilon = 0$ reduces about 22% of calls to an inference engine. The improvement is more significant when checking validity using an approximate condition. With $\epsilon = 0.1$, more than 46% of inference calls are amortized, while the model converges to the same model as exact inferences are made. AI-DCD ($\epsilon = 1$) saves 71% of inference calls and this leads about 60% reduction of inference time during the training. Because inference is faster, AI-DCD ($\epsilon = 1$) reduces the total running time from 534 seconds to 297 seconds compared to the baseline model. Amortized inference is especially helpful in the final stage of optimization, because the model converges. For example, in the last iteration of AI-DCD ($\epsilon = 0.1$), only 12% of inferences require to call an inference engine. As a result, the amortized inference technique is particular useful when an accurate model is required. We verify this by showing the case when running DCD solver with a strict stopping condition ($\delta = 0.01$, maximal iteration=300), AI-DCD ($\epsilon = 1$) speed up the inference by 6.8 times. When ϵ is bigger, the model starts deviating from the optimum. However, this can be remedied by using an adaptive ϵ. Using an adaptive ϵ saves 45% of inference time, while the convergence is guaranteed. Our method is not sensitive to the choice of ϵ. The first 5 rows in Table 1 show that using $\epsilon \leq 1$ speedups the baseline, while obtaining a solution close to the optimum.

In Theorem 1, we show that the quality of the inference is controlled by ϵ and M as defined in Eq. (3). Empirically, when $\epsilon = 0.1$, M is 9.4. Therefore, a 0.5-approximate solution is guaranteed. However, Theorem 1 is not tight. We observe that the worst approximation ratio is actually 0.9996 and our inference method almost always returns an optimal solution. Even when ϵ is large ($\epsilon = 10$), we can obtain an 0.97-approximate solution on average during the learning process, after 5 outer iterations of DCD.

As we mentioned in the previous section, 1-slack SVM (Joachims, Finley, and Yu 2009) uses $\boldsymbol{y}'$ in Eq. (9) to generate a new cutting-plane. A similar heuristic can be applied in DCD. If $\boldsymbol{y}'$ is satisfied with Eq. (8), we can add $\boldsymbol{y}'$ into $\mathcal{A}$ and update $\boldsymbol{w}$ based on $\alpha_{i,\boldsymbol{y}'}$ without solving inference. This approach seems plausible, but does not work well in our setting. It still requires 96.3% of the inference engine calls when $\delta = 0.1$. The main reason is that after a few iterations, most αs associated with the optimal solution of instances have been added into $\mathcal{A}$. However, DCD updates all $\alpha \in \mathcal{A}$ at each loop; therefore $\alpha \in \mathcal{A}$ are unlikely to be satisfied with Eq. (8). As a result, this caching method does not save inference engine calls. In contrast, the amortized inference method verifies the optimality of a cached solution, and can reduce inference calls in this case.

Table 1 also demonstrates the effectiveness of applying amortized inference in the Structured Perceptron model. As shown in the table, most inference problems involved in training can be amortized. As a result, AI-SP with $\epsilon = 1$ significantly reduces the running time, while achieves almost the same test performance.

Multi-label classification. We further ask a question: what other structured problems can use amortized learning? We believe that problems which involve complex output structures but less output variables will benefit most from our framework. To empirically prove this hypothesis, we apply our method on a multi-label classification problem. Following Finley and Joachims (2008) and Chang et al. (2013), we model the multi-label classification using a fully connected pairwise Markov random field. This creates a complex structured prediction problem, where the inference is generally intractable. The last block in Table 1 shows the results. Both the exact (adaptive ϵ) and approximate ($\epsilon = 1$) versions of AI-DCD perform extremely well. That is because the number of variables involved in the ILP formulation of the pairwise Markov random field is small[5], and the condition 4 is more likely to be satisfied. Moreover, the intractable nature of the problem makes the inference engine slow. Therefore, our model speeds up the inference involved in training by an order of 2. The amortized inference removes the bottleneck in the training. When a raw solver is used, 71% of the training time is spent on solving the inference problems. When an amortized inference solver is used, the ratio of the inference time to the total running time reduces to 2%.

Note that the above experiment uses an exact inference solver. The raw solver can be replaced by any approximate inference solver, because checking condition (4) is independent of solver type. The amortized inference technique can be applied in other multi-label classification frameworks.

Cross Validation using Amortization

Our framework is also applicable in situations such as cross-validation, model selection, and feature engineering, where we have to repeatedly run a learning algorithm with different parameters. In the following, we demonstrate our methods when performing a 5-fold cross validation for picking the best regularization parameter C for structured SVM on the entity-relation recognition task. We choose C from $\{0.01$, 0.5 0.1, 0.5, $1\}$, running 5 folds for each, for a total of 25 runs. δ is set to 0.1 for all folds.

We consider two caching policies. In the first, we reset the cache every 5 runs, so the cross validation for each C starts with an empty cache (we call this *cache resetting*). In the second setting, we do not reset the cache during any of the runs, thereby caching the inference problems in all the 25 runs (we call this *cache forwarding*). We compare both these policies with the baseline of using no cache.

The total number of solver calls for the baseline is 18.9M. With cache-resetting, this number drops to 4.4M for AI-DCD ($\epsilon = 1$). Furthermore, with cache forwarding, the number of total solver calls for AI-DCD ($\epsilon = 1$) reduces to 3.3M. When we use adaptive ϵ, the number of solver calls reduces from 12.5M for cache-resetting to 7M for cache forwarding. The reduction in solver calls is smaller when using adaptive ϵ than when $\epsilon = 1$, because the adaptive scheme does exact inference periodically in an attempt to converge to the optimal solution, thereby taking more iterations.

We noticed that the amortization suffers due to resetting the cache, and therefore the amortization ratio drops in the initial folds. We remedy this by using cache forwarding to accumulate the cache. Indeed, using an accumulated cache increases the average amortization ratio from 0.46 to 0.64 in AI-DCD ($\epsilon = 1$). The fact that problems cached for a different C aid in amortizing future problems also suggests that if we need to run the learning process multiple times (for parameter tuning, feature selection etc.), we can benefit by populating the cache in advance. Although we performed this experiment in a serial fashion, this is not a restriction imposed by our amortization framework. Parameter tuning can be done in parallel. In principle, the cache can be shared among several threads or cluster nodes.

Discussion

This paper presents a novel approach to training structured predictors, by amortizing the inference step. It builds on the observation that a large number of calls to an inference engine are needed in the course of training a structured predictor, but even when the inference objectives are different, it is often the case that they produce identical solutions. Exploiting a theory that efficiently recognizes whether two inference objectives will produce the same solution, we show that training structured predictors can be accelerated significantly without any performance degradation.

Acknowledgments This research sponsored by the Army Research Laboratory (ARL) under agreement W911NF-09-2-0053 and by DARPA under agreement number FA8750-13-2-0008. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The opinions, findings, views, recommendations, and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, of ARL, or the U.S. Government. This work is also supported by an ONR Award on "Guiding Learning and Decision Making in the Presence of Multiple Forms of Information."

[5] For a multi-label problem with k labels, the number of variables of the ILP formulation is $k + C(k, 2)$. In our case, k = 6.

References

Chang, M., and Yih, W. 2013. Dual coordinate descent algorithms for efficient large margin structural learning. *Transactions of the Association for Computational Linguistics*.

Chang, K.-W.; Sundararajan, S.; Keerthi, S. S.; and Sellamanickam, S. 2013. Tractable semi-supervised learning of complex structured prediction models. In *ECML*.

Chang, K.-W.; Srikumar, V.; and Roth, D. 2013. Multi-core structural svm training. In *ECML*.

Clarke, J., and Lapata, M. 2006. Constraint-based sentence compression: An integer programming approach. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*.

Collins, M. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*.

Finley, T., and Joachims, T. 2008. Training structural SVMs when exact inference is intractable. In *ICML*.

Hazan, T., and Urtasun, R. 2010. A primal-dual message-passing algorithm for approximated large scale structured prediction. In *NIPS*.

Hsieh, C.-J.; Chang, K.-W.; Lin, C.-J.; Keerthi, S. S.; and Sundararajan, S. 2008. A dual coordinate descent method for large-scale linear svm. In *ICML*.

Huang, L.; Fayong, S.; and Guo, Y. 2012. Structured perceptron with inexact search. In *NAACL*.

Joachims, T.; Finley, T.; and Yu, C.-N. 2009. Cutting-plane training of structural svms. *Machine Learning*.

Kulesza, A., and Pereira, F. 2008. Structured learning with approximate inference. In *NIPS*.

Kundu, G.; Srikumar, V.; and Roth, D. 2013. Margin-based decomposed amortized inference. In *ACL*.

Lacoste-Julien, S.; Jaggi, M.; Schmidt, M.; and Pletscher, P. 2013. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *ICML*.

Meshi, O.; Sontag, D.; Jaakkola, T.; and Globerson, A. 2010. Learning efficiently with approximate inference via dual losses. In *ICML*.

Riedel, S., and Clarke, J. 2006. Incremental integer linear programming for non-projective dependency parsing. In *EMNLP*.

Roth, D., and Yih, W. 2004. A linear programming formulation for global inference in natural language tasks. In Ng, H. T., and Riloff, E., eds., *CoNLL*.

Roth, D., and Yih, W. 2007. Global inference for entity and relation identification via a linear programming formulation. In Getoor, L., and Taskar, B., eds., *Introduction to Statistical Relational Learning*.

Samdani, R., and Roth, D. 2012. Efficient decomposed learning for structured prediction. In *ICML*.

Shevade, S. K.; P., B.; Sundararajan, S.; and Keerthi, S. S. 2011. A sequential dual method for structural svms. In *SDM*.

Sontag, D. 2010. *Approximate Inference in Graphical Models using LP Relaxations*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Srikumar, V.; Kundu, G.; and Roth, D. 2012. On amortizing inference cost for structured prediction. In *EMNLP*.

Sutton, C., and McCallum, A. 2007. Piecewise pseudolikelihood for efficient training of conditional random fields. In Ghahramani, Z., ed., *ICML*.

Taskar, B.; Guestrin, C.; and Koller, D. 2004. Max-margin markov networks. In *NIPS*.

Tsochantaridis, I.; Joachims, T.; Hofmann, T.; and Altun, Y. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*.