Minimizing Expected Losses in Perturbation Models with Multidimensional Parametric Min-cuts

Adrian Kim, Kyomin Jung
Seoul National University
Seoul, South Korea
{adkim955, kjung}@snu.ac.kr

Yongsub Lim
KAIST
Daejeon, South Korea
ddiyong@kaist.ac.kr

Daniel Tarlow, Pushmeet Kohli
Microsoft Research
Cambridge, UK
{dtarlow, pkohli}@microsoft.com

Abstract

We consider the problem of learning perturbation-based probabilistic models by computing and differentiating expected losses. This is a challenging computational problem that has traditionally been tackled using Monte Carlo-based methods. In this work, we show how a generalization of parametric min-cuts can be used to address the same problem, achieving higher accuracy and faster performance than a sampling-based baseline. Utilizing our proposed *Skeleton Method*, we show that we can learn the perturbation model so as to directly minimize expected losses. Experimental results show that this approach offers promise as a new way of training structured prediction models under complex loss functions.

1 INTRODUCTION

Many problems in machine learning can be formulated as structured-output prediction, such as pixel labelling problems in computer vision and protein side-chain prediction in bio-informatics. A key challenge in the solution of these problems is to build structured prediction models that capture key correlations within the outputs and to learn these models from data. There are a range of approaches to this problem, including training a deterministic predictor to minimize (regularized) empirical risk (e.g., structural SVMs (Taskar et al., 2003; Tsochantaridis et al., 2005)), PAC Bayesian-based approaches where the goal is to train a randomized predictor to minimize a regularized empirical risk (Keshet et al., 2011), and probabilistic modelling paired with Bayesian decision theory (Schmidt et al., 2010).

Perturbation models (Papandreou & Yuille, 2011; Tarlow et al., 2012; Hazan & Jaakkola, 2012) are an approach that have been a focus of interest in recent years, and are closely related to both PAC-Bayesian approaches and probabilistic modelling. The idea is to build a probabilistic model over structured outputs by drawing a random energy function and then returning the argmin of the random energy function as a sample from the model. These models can then be trained under maximum likelihood-like objectives (Papandreou & Yuille, 2011; Tarlow et al., 2012; Hazan & Jaakkola, 2012) or to minimize expected loss (Keshet et al., 2011; Hazan et al., 2013). Typically the distribution over energy functions is restricted so that the optimization step is tractable (e.g., it is a min-cut problem). When this is the case, perturbation models have the desirable property that exact samples can be drawn efficiently with a single call to an efficient optimization procedure.

Our aim in this work is to revisit the problem of training perturbation models to minimize expected losses. Previous works (Keshet et al., 2011; Hazan et al., 2013) have used Monte Carlo-based methods to estimate the needed gradients. A concern with these approaches is that the gradient estimates can have high variance, as is the case with the well-known REINFORCE algorithm (Williams, 1992). Instead, our approach here is to explore combinatorial methods that take advantage of the structure of the optimization problem in order to more efficiently make use of optimizer runs. As a first foray into this approach, we restrict attention to the case where the perturbation model takes the form of a uniform distribution over model parameters followed by a call to a min-cut/maxflow routine.

Our method is based on a generalization of the parametric min-cut algorithm (Gallo et al., 1989) which in the 1-dimensional case is able to efficiently compute all parameter values (breakpoints) where the minimum energy (MAP) solution changes. To demonstrate the efficacy of our method, we compare estimated expected losses and their gradients computed by our method with those obtained from a sampling-based scheme. Experimental results show that we get more accurate solutions with fewer calls to the optimization procedure and less overall wall time.

As a full application, we also show that our method is use-

ful towards training structured prediction models to minimize expected losses. The method is indifferent to the loss function used, so there is potential to use the same method for loss functions that are typically difficult to work with. Experimentally, we compare our method to learning using Perturb-and-MAP (a.k.a. P&M) (Papandreou & Yuille, 2011) to learn a probabilistic model, then making loss-aware predictions using Bayesian Decision theory. We also show that the Skeleton method can be used in place of sampling within the training procedure from (Papandreou & Yuille, 2011) to give gradients with lower variance.

2 BACKGROUND: PERTURBATIONS, EXPECTED LOSSES

We will focus on the case where perturbation models are used to define a conditional probability model $P(y \mid x; \theta)$, where x is an input (e.g., an image), $y \in \{0,1\}^n$ is a structured output (e.g., a foreground-background image segmentation), and $\theta \in \mathbb{R}^m$ is a real-valued vector of parameter values. We additionally assume access to a feature vector $\phi(x, y) \in \mathbb{R}^m$ which contains unary and pairwise potentials. Perturbation models begin by defining an energy function $E(y \mid x; \theta) = \langle \theta, \phi(x, y) \rangle$. The second component to a perturbation model is the noise distribution $P(\gamma)$ which is a distribution over noise vectors $\gamma \in \mathbb{R}^m$. The probability model $P(y \mid x; \theta)$ can then be defined as follows:

$$\gamma \sim P(\gamma) \tag{1}$$

$$y = \operatorname*{argmin}_{y'} E(y' \mid x; \theta + \gamma). \tag{2}$$

It will be useful to define *minimizer* $f(\theta) = \operatorname{argmin}_y E(y \mid x; \theta)$, *dual function* $g(\theta) = \min_y E(y \mid x; \theta)$, and *inverse set* $f^{-1}(y) = \{\theta : f(\theta) = y\}$. Under this definition, the probability of a configuration y can be expressed as $P(y \mid x; \theta) = \int \mathbf{1}_{\{\theta+\gamma \in f^{-1}(y)\}} P(\gamma) d\gamma$. We are interested in expected losses under perturbation models. The expected loss (or *risk*) is a function of a given y^* (in our case, the ground truth configuration) and parameters θ. It is defined as

$$R(y^*, \theta) = \sum_{y \in \{0,1\}^n} P(y \mid x; \theta) L(y^*, y) \tag{3}$$

$$= \sum_{y \in \{0,1\}^n} \int \mathbf{1}_{\{\theta+\gamma \in f^{-1}(y)\}} P(\gamma) L(y^*, y) d\gamma, \tag{4}$$

where $L(y^*, y)$ assigns a loss value for predicting y when the ground truth is y^*. The ultimate goal we are working towards is to learn parameters θ so as to minimize $R(y^*, \theta)$. First, we focus on the prerequisite tasks of computing and differentiating $R(y^*, \theta)$. We will use deterministic update rules to calculate gradients with the Skeleton method to learn the model.

3 ALGORITHM: SKELETON METHOD

We begin by making some assumptions. First, let $P(\gamma)$ be a uniform distribution such that $\theta + \gamma$ is distributed uniformly over a m-dimensional hyperrectangular region $S_\theta = \prod_{i=1}^m [\theta_i, \theta_i + w_i]$, where $\gamma_i \in [0, w_i]$ and $w_i \in \mathbb{R}^{>0}$. Also assume that the minimizer $f(\theta)$ is unique for all θ except for a set with measure zero, so $f(\theta)$ can be treated as having a unique value. Finally, assume that for all $\theta \in S$, $E(y \mid x; \theta)$ is submodular and can be optimized efficiently.

In the following, it will be convenient for us to redefine the inverse set $f^{-1}(y)$ so that only regions in S are included. That is, $f^{-1}(y) = \{\theta : f(\theta) = y \wedge \theta \in S\}$. Then from above, we have that $R(y^*, \theta) = \sum_{y \in \{0,1\}^n} \int \mathbf{1}_{\{\theta+\gamma \in f^{-1}(y)\}} P(\gamma) L(y^*, y) d\gamma$. Noting that $L(y^*, y)$ is not a function of γ and that $\int \mathbf{1}_{\{\theta+\gamma \in f^{-1}(y)\}} P(\gamma) d\gamma = \operatorname{Volume}(f^{-1}(y)) / \operatorname{Volume}(S)$, we can rewrite $R(\cdot)$ as $\sum_{y \in Y_S} L(y^*, y) \operatorname{Volume}(f^{-1}(y)) / \operatorname{Volume}(S)$, where $Y_S = \{y : \exists \theta, \theta \in S \wedge f(\theta) = y\}$ is the set of configurations that are minimizers for some $\theta \in S$.

In this paper, we introduce a novel method to find the minimizers $y \in Y_S$ and their inverse sets by iteratively updating a graph structure that we call a *skeleton*. Note that for a fixed y, $E(y \mid x; \theta)$ is a linear function of θ, which implies that the dual function $g(\theta) = \min_y E(y \mid x; \theta)$ is a piecewise concave function, where pieces are hyperplanes corresponding to minimizers y. Let h_y be the corresponding hyperplane for some fixed minimizer y. Intuitively, the skeleton $G_Y = (V_Y, E_Y)$ is a graphical representation of the dual $g(\theta)$ over S. The skeleton will be constructed on the given parameter space S by finding new minimizers, or hyperplanes, at each iteration until there are no more minimizers. At each iteration, the growing skeleton represents an upper bound on the dual g, which we call the *subset dual*.

Definition 1 (Subset dual g_Y) *For some given minimizer set $Y \subseteq Y_S$, let $g_Y(\theta) = \min_{y \in Y} E(y \mid x; \theta)$ be the subset dual, which is a piecewise concave function.*

For some given subset dual $g_Y(\cdot)$, each hyperplane h_y has a corresponding graph which we refer as a *facet* G_y. A facet $G_y = (V_y, E_y)$ is defined as the smallest convex hull made by the intersections of h_y and other hyperplanes, where V_y, E_y are boundary vertices and edges of the convex hull. Let θ_v be the parameter value and $z_v = g_Y(\theta_v)$ be the subset dual value corresponding to the vertex v. Note that a facet can be cut because of the boundaries the given parameter space makes. A skeleton is defined using the union of these facets as follows.

Definition 2 (Skeleton of g_Y over S) *For some given subset dual g_Y, the skeleton of g_Y on S can be represented by the following structure $G_Y = (V_Y, E_Y)$. Let (u, v) be*

an edge between u and v, where $u, v \in V_Y$.

- $V_Y = \bigcup_{y \in Y} \{v : \textit{Boundary vertices of } G_y, \textit{ where } \theta_v \in S, z_v = g_Y(\theta_v)\}$
- $E_Y = \bigcup_{y \in Y} \{(u,v) : \textit{Boundary edges of } G_y, \textit{ where } u, v \in V_Y\} \cup \{(u,v) : u \in V_Y, \theta_u \in \Pi_{i=0}^{m} \{w_i^-, w_i^+\}, v = (\theta_u, -\infty)\}$

For example, Figure 1 is a skeleton over some parameter space $S \in \mathbb{R}^2$ *given a subset dual* g_Y*, where* $Y = \{y_1, ..., y_5\}$*. There are five facets on the skeleton, where four are cut by the boundaries of* S*.*

From the given definitions, it is clear that an inverse set $f_Y^{-1}(y) = \{\theta : y = \text{argmin}_{y' \in Y} f_\theta(y') \wedge \theta \in S\}$ of y defined on a subset dual g_Y directly corresponds to the projection of the facet G_y. Thus, in order to calculate the volume of $\theta_Y(y)$, we can use the projected vertices of G_y on S. One of the main points of our method is that we are able to track every facet with every iteration, so that we can calculate the approximate expected loss every time we update the skeleton.

We now describe our *Skeleton method* and how it works. Figure 1 describes a visual example on how a skeleton is constructed and updated by a single iteration of our algorithm. Algorithm 1 is the pseudo code of the algorithm.

3.1 INITIALIZATION

The initial skeleton $G_Y = (V_Y, E_Y)$ is given by the following.

- $Y = \phi$
- $V_Y = \{(\theta_{v_n}, z_{v_n}) : \theta_{v_n} = \Pi_{i=0}^{m} \{w_i^-, w_i^+\}, z_{v_n} = \infty\}$
- $E_Y = \{(u,v) : u \in V_Y, v = (\theta_u, -\infty)\}$

3.2 FINDING A NEW FACET

In order to find a new facet, the algorithm first picks some vertex $u = (\theta_u, z_u) \in V_Y$. Using graph cut, a new solution $y_u = f(\theta_u)$ can be found. The first step is to determine whether the new solution improves the current dual in any region. This can be checked by $h_{y_u}(\theta_u) < z_u$. If this is the case, we say that a *cut* is made, and y_u is added to Y.

Next, we must find new intersection points where h_{y_u} intersects other hyperplanes defining the subset dual. The key property of the new intersection points is that they will either appear at existing vertices $v \in V_Y$, or they appear on an edge $(p_h, p_t) \in E_Y$ that "crosses" the new hyperplane; that is $h_{y_{p_h}}(\theta_{p_h}) < z_{p_h}$ and $h_{y_{p_t}}(\theta_{p_t}) > z_{p_t}$. The set of vertices where $h_{y_v}(\theta_v) < z_v$, form a connected component

Algorithm 1 Skeleton Method

Input: Oracle f, Loss function $L(y^*, y)$
$(Y, G_Y) \leftarrow InitSkeleton()$
for all $u = (\theta_u, z_u) \in V_i$ **do**
  $y_u = f(\theta_u)$
  if $h_{y_u}(\theta_u) < z_u$ **then**
    Add y_u to Y
    $(I, H) \leftarrow FindIntersection(G_Y, h_{y_u})$
    Add $f_{y_u} = (y_u, I)$ to F_Y
    $V_Y = (V_Y \cup I) - H$
    for all Intersection vertices $p \in I$ **do**
      if p is a new vertex **then**
        Add p to all $G_y \in$ {Facets sharing (p_t, p_h), where p_h is above and p_t is below h_{y_u}}
        Append new edge (p_t, p) to E_Y
      end if
    end for
    Remove vertices $r \in H$ above h_{y_u} from all facets
    Remove $E_- = \{(u,v) : u \text{ or } v \in H\}$ from E_Y
    Append $E_+ =$ {Boundary edges of G_{y_u}} to E_Y
    $R = \sum_{y \in Y} Volume_{f^{-1}(y)} L(y^*, y)$
  end if
end for

$H \subset G_Y$, and the crossing edges are the boundary edges of this connected component. Thus, the intersection points can be found by exploring a search tree outwards from u. When a vertex v is encountered such that $h_{y_v}(\theta_v) < z_v$, the intersection point between v and its parent is computed by finding some point p where $h_{y_p}(\theta_p) = z_p$, and the search tree is not searched further down that path. Upon termination, vertices of the connected component H are removed from V_Y, and the new intersection points, notated as I, are added. A step of this procedure is illustrated in Figure 1(b), where there is a cut after selecting vertex u_i, colored in red.

3.3 UPDATING THE SKELETON G_Y

When a cut is done in the skeleton, it should be updated with the new upper bound made by h_{y_u}. The nontrivial case is when some intersection point $p \in I$ is a new point made on some edge $(p_h, p_t) \in E_Y$, which is (u_1, v_1) for p_1 in Figure 1b. The new vertex p is added to all facets which share the edge (p_h, p_t). Also, a new edge (p_t, p) should be added to the skeleton. Boundary edges made from the convex hull of the new polytope G_{y_u} are also added to E_Y. Finally, the skeleton update is done when all vertices $r \in H$ are deleted from every facet and all edges including r are removed from E_Y.

3.4 CALCULATING EXPECTED LOSS R

At this point, the skeleton is fully updated. To compute expected loss $R(y^*, \theta)$, we use an off-the-shelf subroutine

for computing the volume of each inverse set $f_Y^{-1}(y)$ for $y \in Y$. The volumes are multiplied by the loss value for each y, and the products are summed to get the full expected loss. For normalization, the value is divided by the volume of S.

3.5 EXAMPLE : TWO PARAMETERS

Figure 1 describes a single iteration of the Skeleton Method on a perturbation model having two parameters, θ_1, θ_2. Note that the leftside represents the subset dual g_Y and that the right image isthe projection of facets on the given parameter space S. The iteration starts from a skeleton which has already done five iterations by the algorithm ($Y = \{y_1 ... y_5\}$). There are five facets on the parameter space so that the expected loss is $R(y^*, \theta) = \sum_{n=1}^{5} \text{Volume}(f_Y^{-1}(y_n)) L(y^*, y_n) / \text{Volume}(S)$. Suppose we take some unused vertex u_1. In this case, we can see that the hyperplane h_{y_6} makes a cut in the skeleton ($Y' = Y \cup \{y_6\}$). By updating the skeleton, a new facet f_6 is found. Since there is a unique loss value for each facet, we can calculate the expected loss as $R(y^*, \theta) = \sum_{n=1}^{6} \text{Volume}(f_{Y'}^{-1}(y_n)) L(y^*, y_n) / \text{Volume}(S)$.

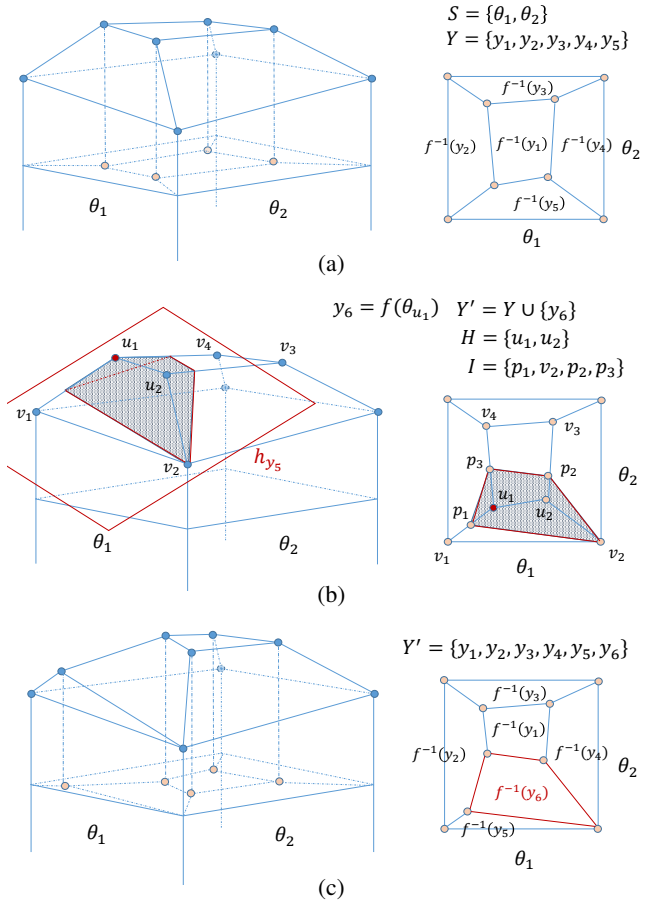


Figure 1: Visual Example of the Skeleton Method with Two Parameters.

4 LEARNING

4.1 COMPUTING GRADIENTS: SLICING

Our main focus is not only computing expected losses; the ultimate aim is to learn parameters that yield a perturbation model that achieves low expected loss. In order to update the perturbation model to minimize the expected loss, we calculate the gradients by applying a simple finite-differencing-based technique named **slicing**.

Before going through details, we add more assumptions from the previous section. To be more flexible, let the parameter space where $\theta + \gamma$ is sampled from notated as $S_\theta = \theta + S = \theta + \Pi_i^m \gamma$, where $\gamma_i \in [0, w_i]$. The expected loss of the region which S creates on parameter θ will be notated as $R_S(y^*, \theta)$. The approximation we use is

$$\frac{\partial R(y^*, \theta)}{\partial \theta_i} \approx \frac{R_S(y^*, \theta + \delta e_i) - R_S(y^*, \theta)}{\delta}, \quad (5)$$

where δ is a small value and $e_i \in \mathbb{R}^m$ is a unit vector with 1 in the ith coordinate and 0 elsewhere. Intuitively, this is identical to the difference between expected losses of regions shifted to the + direction of θ_i by a small distance of δ. Therefore, we can use a Monte Carlo-based method or the Skeleton method on these two regions and compute the differences to find the gradients.

When shifting a region by δ, we see that the contribution to the gradient comes just from the δ-width end-regions illustrated in Figure 3 (b). We call these end regions *slices*. Motivated by this, instead of computing expected losses of the shifted and unshifted full regions, we compute expected losses only on the slices. Intuitively, we expect the slices to have fewer minimizers defining the Skeleton structure than the full regions that include them, and we expect that focusing only on the regions of difference will lead to faster and more accurate gradient estimates.

Let $s^i = \Pi_j^m \gamma_j$ be the size of the thin slice where $\gamma_j = [0, w_j]$ except the ith range $\gamma_i \in [0, \delta]$. From this setting, $R_{s^i}(y^*, \theta)$ stands for the expected loss of the sliced region of size s^i. Using this we can apply gradient descent updates.

$$\frac{\partial R(y^*, \theta)}{\partial \theta_i} \approx R_{s^i}(y^*, \theta + w_i) - R_{s^i}(y^*, \theta) \quad (6)$$

$$\theta_i(t+1) = \theta_i(t) - \alpha_i \frac{\partial R(y^*, \theta)}{\partial \theta_i} \quad (7)$$

Each parameter θ_i in iteration $t+1$ is updated with the gradient value with a constant step size of α_i, which is proportional by the feature size of θ_i. One thing to be cautious about when selecting a learning rate is, that if the

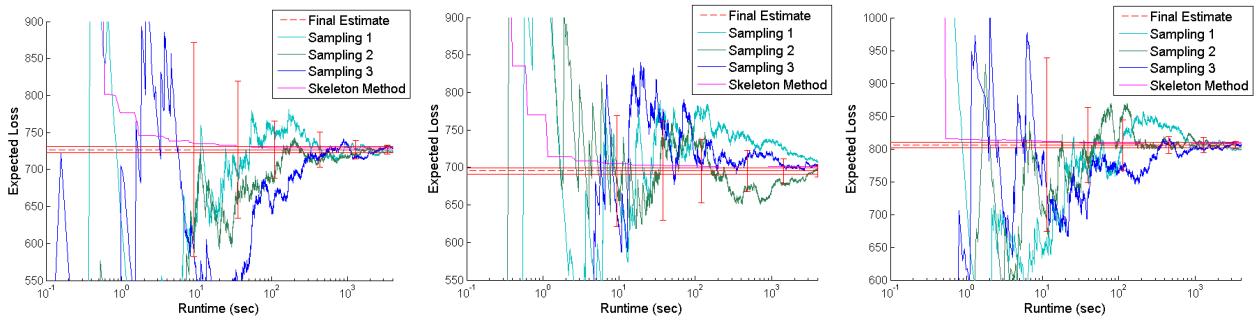
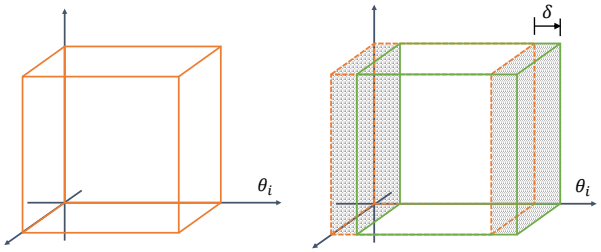


Figure 2: Comparisons between the Monte Carlo Estimator and the Skeleton Method. Using both methods we computed expected Hamming losses to identical settings and compared by runtime. Images of size [90x120] taken from (Rother et al., 2004) are used.



(a) S_θ (b) $S_\theta \rightarrow S_{\theta+\delta e_i}$

Figure 3: Visual Discription of Slices. The Slicing method computes the differences in gray regions in (b) to estimate the gradient with respect to parameter θ_i.

learning rate is too large, then the parameters may make the model jump to an **unlearnable** state (plateau in the objective), which is a state where S_θ holds only one inverse set.

4.2 TRAINING

In order to learn the parameters for our perturbation model, we exploit the Slicing method so that the model is trained directly from minimizing expected loss. One main advantage for our method is that we can use an arbitrary loss function very easily in this process. Suppose we have a training set with a size of N and have m parameters. For each iteration, we make $2m$ slices from the model. Parameters are updated by using the mean value of gradients from all training images like the following equation.

$$\theta_i(t+1) = \theta_i(t) + \alpha_i \frac{1}{N} \sum_n^N \frac{\partial R(y_n^*, \theta)}{\partial \theta_i} \tag{8}$$

Note that it is not necessary to evaluate the expected loss objective at every step of the optimization.

4.3 EXPLOITING THE SKELETON METHOD

Previous approaches focus on how to use sampling methods to learn their models, which although many have well understood theoretical convergence properties as the sample size goes to infinity, suffer from problems with high variance in practice. In fact, the Skeleton method can be used in place of sampling more generally; for example, the P&M model in (Papandreou & Yuille, 2011) is trained using a moment-matching objective described in Eq. 9-11.

$$\theta_i(t+1) = \theta_i(t) - \alpha_i \Delta\theta_i \tag{9}$$

$$\Delta\theta_i = E_{S_\theta}[\phi_i(y)] - E[\phi_i(y^*)] \tag{10}$$

$$E_{S_\theta}[\phi_i(y)] = \frac{1}{M} \sum_j^M \phi_i(y_j) \tag{11}$$

To compute the expectations $E_{S_\theta}[\phi_i(y)]$, where $\phi_i(y)$ is a feature function, the standard approach is to use sampling. However, we can replace the sampling-based approach with a skeleton-based approach. Specifically, we replace the term to be $E_{S_\theta}[\phi_i(y)] = \sum_{y \in \{0,1\}^n} P(y \mid x, \theta) L_H(y^*, y_j)$ and then use the method described above to compute the quantities needed in Eq. 9-11. This gives an alternative method for optimizing the original P&M objective; we call this approach Skeleton Perturb-and-MAP (Skeleton P&M).

5 EXPERIMENTS AND DISCUSSION

5.1 DATA AND SETUP

In this section, we apply the Skeleton Method to a foreground-background image segmentation task, comparing against Monte Carlo baselines which estimate expected losses by drawing samples from the prior and reporting the average incurred loss. All images used in experiments are

originally from the Berkeley image segmentation set by (Rother et al., 2004). The energy function used is of the following form:

$$\begin{aligned} E(y \mid x; \theta) &= \langle \theta, \phi(x, y) \rangle \\ &= E(x) + \theta_1 \sum_i^n x_i \\ &+ \theta_2 \sum_{(y_i, y_j) \in E_v} (y_i \neq y_j) + \theta_3 \sum_{(y_i, y_j) \in E_h} (y_i \neq y_j) \end{aligned} \tag{12}$$

where E_v, E_h are each sets of neighboring vertical and horizontal pairs of pixels respectively. x_i is the ith pixel's label of the noised input, which is made by switching values of ground truth labels with a uniform probability of 5%.

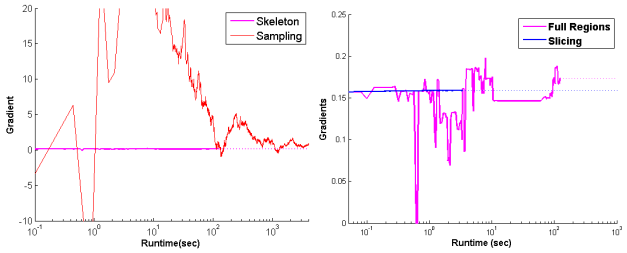
Expected losses were computed over a parameter space $S_\theta = \theta + \gamma \subseteq \mathbb{R}^m$ defined from a uniform distribution $\gamma \sim P(\gamma)$ where $\gamma \in [0, 1]^3$. Intuitively, S_θ is a cube shaped region positioned by θ on the parameter space where parameters are sampled from. Expected loss over region S_θ will be notated as $R_S(y^*, \theta)$.

In default, the loss function for the following experiments will be defined as the Hamming distance, $L_H(y^*, y) = \sum_{i=1}^n (y_i \neq y_i^*)$. Note that this formulation supports arbitrary loss functions other than the Hamming distance.

5.2 CALCULATING EXPECTED LOSSES

To evaluate the methods, it would be ideal to have a ground truth value of expected losses for a given parameter setting. Unfortunately this is hard to calculate accurately, because the Skeleton method does not always run to termination within practical time, and there is necessarily some variance in the estimates returned by the sampling estimate. Thus, we report the estimates from each method along with 95% confidence intervals derived from the sampling method. For the sampling method, in each trial, parameters were independently sampled 100k times, and this was repeated 10 times.

Figure 2 shows plots of expected losses calculated by the two methods versus runtime. The average sampling estimate (across all trials) appear as red dashed lines in Figure 2(a) -2(c). Also shown are the cumulative averages for three representative trials of the sampling (green to blue curves), and the Skeleton method (magenta). The main take-away is that the expected loss values of both methods converge to similar values, but particularly with few samples, there is high variance in sampling. While the Monte Carlo estimator has significant variance even after 1000 seconds, the Skeleton Method has essentially converged to its final, accurate estimate after approximately 10 seconds. This suggests that we can even stop running the method in the middle of the algorithm to estimate the expected loss with high accuracy. The reason such behavior appears is related to the high concentration of vertices in the later iterations of the algorithm. Many calculations made in later iterations induce inverse sets which have very small volumes, implying the low contribution to the expected loss.



(a) (b)

Figure 4: Gradients of Expected Hamming Loss for θ_1. (a) Sampling and Skeleton method on two full regions (b) Skeleton method on full regions and Slicing method.

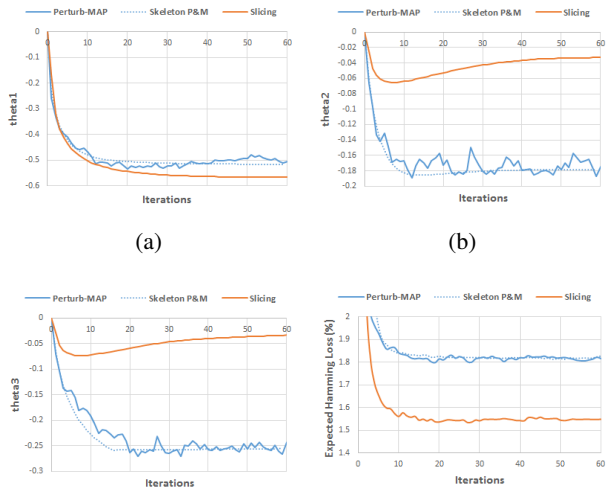
5.3 CALCULATING GRADIENTS

We now turn attention to evaluating the Skeleton method and Monte Carlo method for computing gradients of expected losses. For the Skeleton method, we evaluate our recommended Slicing method, and also a variant that computes expected losses over full regions that are shifted by δ, which would be the more standard finite-difference approach. We use the thickness $\delta = 0.001$ and parameter θ_1, which is for the unary term, for the experiments. A comparison of the Monte Carlo approach (red) and the full-region Skeleton method (magenta) appear in Figure 4 (a). The red curve shows the cumulative average Monte Carlo estimate averaged across 10 repetitions. Even with this averaging, we see a great deal of variance in the estimates. The Skeleton method, by contrast, quickly converges to a value near where the Monte Carlo estimator appears to be converging to.

We then zoom in (note the y-axis scales) and consider the recommended Slicing variant of the Skeleton method and compare it to the full-region version shown in Figure 4 (a). The result appears in Figure 4 (b). Here we see that the Slicing variant is faster and much more stable than the full-region variant. As mentioned above, we believe the reason for the disparity is that number of unique inverse sets in the Slicing variant is smaller, and there is no variance that arises from the two runs computing slightly different estimates of the expected loss in the middle region that is contained by both the original and shifted full region.

5.4 MODEL LEARNING

Learning was done an image set including 30 images each having approximately 2500 pixels. The data set was randomly split into $N = 24$ training images and $N' = 6$ test



(a) (b) (c) (d)

Figure 5: Parameter Learning with P&M and the Slicing Method. (a)-(c): Parameter updates (d): Expected Hamming Loss updates

images.

5.4.1 Learning

We performed learning with the Slicing method, where gradients are computed with slices having a thickness of $\delta = 0.001$. The starting parameter is $\theta = (0, 0, 0)$, with a uniform perturbation $\gamma \in \Pi_i^3[0, 1]$ defining a cube-shaped region on the parameter space. Gradients are computed for each parameter, which makes 6 slices to use. All slices can be computed independently, where in most cases 1-3 seconds are enough to get significant accuracy. By every iteration, the region will shift to a certain direction, and the process is repeated. The orange plots of Figure 5 show how the Slicing method learns the model for 60 iterations.

As a baseline for our method, we trained the P&M model with the same settings. Note that the Slicing method and P&M model have different behaviors, which are due to the difference in objectives; our Slicing method directly tries to minimize expected Hamming loss while the P&M model uses a moment-matching rule to estimate the posterior. The behavior of the learning P&M model is illustrated as the solid blue line of Figure 5, with the Skeleton P&M model being the dotted blue line. Take note that the Skeleton P&M strongly resembles the original P&M trace, but its trajectory is smoother, presumably due to lower variance in the gradient estimates.

At test time, instead of computing expected losses accurately, there may be a desire to sacrifice accuracy over runtime in estimating the value. One easy example is to use a finite number of samples such as 20 and compute the average of losses. Another approach is to sample a single output from a moderate position such as the center of the parameter space. Table 1 shows the expected losses computed from the mentioned methods. Each column represents the method we choose to compute expected loss. Each row represents the selected model trained for 60 iterations. Both from Figure 5(d) and Table 1, it is clear that our method is superior to the P&M model in optimizing the expected Hamming loss.

Table 1: Expected Hamming Losses. Expected losses are computed with three ways 1) Average loss of 20 samples 2) Skeleton method 3) Single sampled loss from center. The performance for each model is described in each row, where values were computed separately on the training set and test set.

METHOD	SAMPLED	EXPECTED	CENTER
P&M (Train)	1.694±.0011	1.812	**.2369**
Skel. P&M (Train)	1.764±.0017	1.816	**.2369**
Slicing (Train)	**1.480±.0012**	**1.535**	.2932
P&M (Test)	2.186±.0011	2.257	**1.172**
Skel. P&M (Test)	2.197±.0016	2.268	1.178
Slicing (Test)	**2.048±.0043**	**2.134**	1.391

5.4.2 Other Loss Functions

With our method, it is possible to minimize an arbitrary loss function's expected value. In the following experiments we try to minimize the following loss function.

- **Boundary-only Pixel Loss** L_P: Hamming loss on only pixels which have at least one neighbor with a different label in the ground truth

$$L_P(y^*, y) = \sum_{y_i^* \neq y_j^*} (y_i \neq y_i^*)$$

The solid lines of Figure 6 shows the expected losses changing by the Slicing method in 60 iterations. The dashed lines are loss values from a baseline where we use the learned Skeleton P&M parameters to make loss-directed predictions using an approximation of Bayesian decision theory, similar to that used by (Premachandran et al., 2014). approximation Bayesian Decision Theory prediction framework. Specifically, we sample $M = 100$ segmentations $Y = \{y^{(1)}, \ldots, y^{(M)}\}$ from the learned model, then we make predictions by restricting possible predictions to be one of the M sampled segmentations, and we approximate expected losses by taking averages over the M segmentations; specifically, we predict as $\arg\min_{y' \in Y} \sum_{y \in Y} \Delta(y', y)$. Figure 6 shows the expected boundary-only pixel loss being learned from the Slicing method as solid lines and the approximate Minimum Bayes Risk (MBR) prediction loss as dashed lines. This experiment shows that our method gives better results than the classical approach in minimizing expected losses.

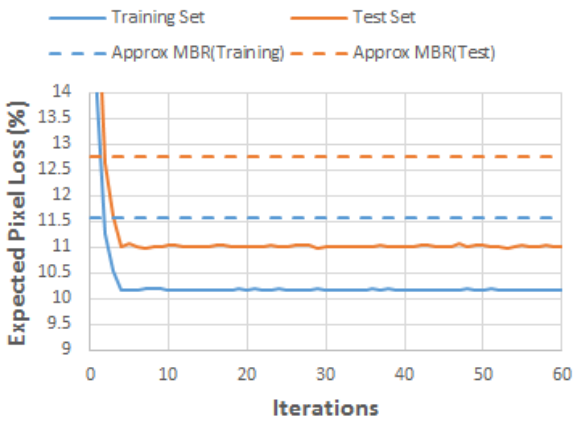


Figure 6: Learning Other Expected Losses. Orange and blue lines represent the values computed from the test set and training set respectively. Dashed lines are Approximate MBR prediction loss values, while the solid lines are learned from the Slicing method.

5.5 EXPECTED SEGMENTATIONS

To visualize how different parameters, or regions, effect the expected loss, we can use a probabilistic image constructed from every solution captured by our algorithm. This image or *expected segmentation* is made by weighting each configuration by the volume of its inverse set and summing up to a gray scale image. Note that this implies that the values of Table 1 are identical to the l_1 distance between the ground truth image and the expected segmentation. Examples are shown in Figure 7. The example images were selected from the test set. From the figure, you can see that expected segmentations made from our perturbation model have higher quality, smoother segmentations than those from the P&M models.

6 CONCLUSION

Our results show that the Skeleton method is a promising alternative to Monte Carlo methods. The Skeleton method converges in a nice deterministic behavior, which shows higher accuracy than using samples. Another benefit of the Skeleton method is that it is applicable for any loss function. We have shown it applied to a boundary-only pixel loss; in future work it would be interesting to apply it to even more complicated loss functions. The Skeleton method also appears to be a general drop-in replacement for sampling-based computation of expectations in perturbation models. We showed this by adapting the method to the moment-matching objective that the original P&M paper proposed, showing that the Skeleton method leads to similar-but-smoother learning trajectories.

The idea of iteratively building piecewise linear approximations arises in many cases, such as when computing the value function in POMDPs (Porta et al., 2006; Isom et al., 2008; Brechtel et al., 2013). While the high level ideas are similar to these and other methods, the details are quite different; for example, in the above works, no volume computations are required, whereas they are core to our method.

The primary challenge going forward is to broaden the applicability of the method, extending to higher dimensions and enlarging the space of supported perturbation distributions. It is likely in these cases that exactness of the method will need to be abandoned due to the fact that the number of solutions will likely grow, and the computations of necessary volumes will become computationally hard. Despite this, we believe the algorithm presented here will be useful going forward. There are two possibilities we are interested in exploring: first, using a hybrid of the Skeleton and sampling methods where some dimensions are sampled and some are integrated analytically using the Skeleton method (producing a Rao-Blackwellized sampler); second, we believe there to be opportunities for computing and differentiating upper bounds based on the Skeleton structure, which could lead to interesting new learning methods.

Acknowledgements

K.Jung is with ASRI, Seoul National University,and he is funded by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2012R1A1A1014965).

References

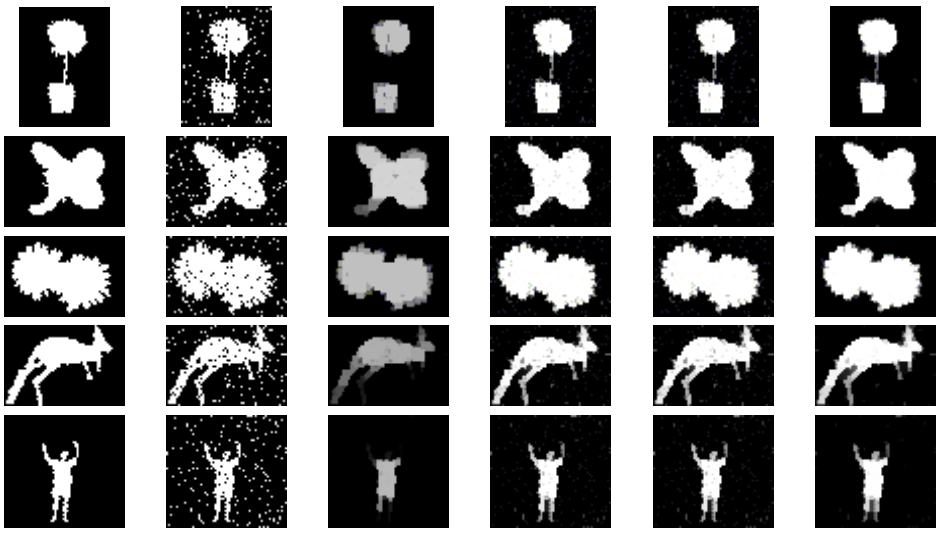
Brechtel, Sebastian, Gindele, Tobias, et al. Solving continuous pomdps: Value iteration with incremental learning of an efficient space representation. In *Proceedings of the 30th International conference on machine learning*, pp. 370–378, 2013.

Gallo, Giorgio, Grigoriadis, Michael D, and Tarjan, Robert E. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1): 30–55, 1989.

Hazan, Tamir and Jaakkola, Tommi S. On the Partition Function and Random Maximum A-Posteriori Perturbations. In *ICML*, pp. 991–998, 2012.

Hazan, Tamir, Maji, Subhransu, Keshet, Joseph, and Jaakkola, Tommi. Learning efficient random maximum a-posteriori predictors with non-decomposable loss functions. In *Advances in Neural Information Processing Systems*, pp. 1887–1895, 2013.

Isom, Joshua D, Meyn, Sean P, and Braatz, Richard D. Piecewise linear dynamic programming for constrained pomdps. In *AAAI*, pp. 291–296, 2008.



(a) (b) (c) (d) (e) (f)

Figure 7: Expected Segmentations. (a) Ground Truth (b) Noised Input (c) Default (0,0,0) (d) P&M (e) Skeleton P&M (f) Slicing Method

Keshet, Joseph, McAllester, David, and Hazan, Tamir. Pac-bayesian approach for minimization of phoneme error rate. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 2224–2227. IEEE, 2011.

Papandreou, G. and Yuille, A. Perturb-and-MAP Random Fields: Using Discrete Optimization to Learn and Sample from Energy Models. In *ICCV*, pp. 193–200, Barcelona, Spain, November 2011. doi: 10.1109/ICCV.2011.6126242.

Porta, Josep M, Vlassis, Nikos, Spaan, Matthijs TJ, and Poupart, Pascal. Point-based value iteration for continuous pomdps. *The Journal of Machine Learning Research*, 7:2329–2367, 2006.

Premachandran, Vittal, Tarlow, Daniel, and Batra, Dhruv. Empirical minimum bayes risk prediction: How to extract an extra few% performance from vision models with just three more parameters. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1043–1050. IEEE, 2014.

Rother, Carsten, Kolmogorov, Vladimir, and Blake, Andrew. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (TOG)*, volume 23, pp. 309–314. ACM, 2004.

Schmidt, Uwe, Gao, Qi, and Roth, Stefan. A generative perspective on mrfs in low-level vision. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 1751–1758. IEEE, 2010.

Tarlow, Daniel, Adams, Ryan Prescott, and Zemel, Richard S. Randomized Optimum Models for Structured Prediction. In *AISTATS*, pp. 21–23, 2012.

Taskar, Ben, Guestrin, Carlos, and Koller, Daphne. Max-margin markov networks. In *Advances in Neural Information Processing Systems*, pp. None, 2003.

Tsochantaridis, Ioannis, Joachims, Thorsten, Hofmann, Thomas, and Altun, Yasemin. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pp. 1453–1484, 2005.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.