An Online Latent Perceptron Algorithm for Weakly Supervised Object Detection [*]

Chirag Gupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
chiragvg@iitk.ac.in

Advisors
Professor Purushottam Kar
Professor Vinay Namboodiri

1 Overview

Object Detection refers to the problem of learning a model that given an image, detects the objects present in the image. In the full-blown setting, we would like to obtain each and every object present in the image, potentially along with the spatial relationships between them. Instead, the most common technique involves training a classifier that detects the presence of a single object, and using a bunch of such classifiers, possibly in a hierarchy. In addition, the classifier may output a bounding box that gives the position of the image. What could be the kind of labeling that would be required by such a classifier? There are two possibilities -

- **Fully supervised object detection** refers to the scenario wherein along with the objects present in the image, it is also tagged with the exact position of the object, in the form of a rectangular bounding box.
- **Weakly supervised Object Detection** refers to the scenario wherein only a 0-1 label is present that says whether a certain object is present or not, instead of the complete bounding box.

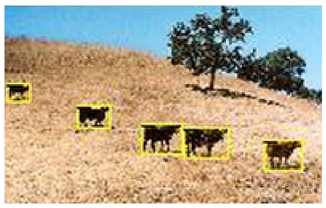
Considering the amount of time and resources that would be required to have complete supervision, it is an important problem in Computer Vision to perform detection in the weakly supervised setting. Bilen et al [1] treated the bounding box as a latent variable, and applied a max-margin formulation based on the latent SVM developed by Yu and Joachims [17]. Although the latent SVMs makes it possible to optimize the non-convex objective over the latent variable through DC programming [18], it utilizes an underlying QP solver that incrementally incorporates the most violated constraint (cutting-plane methods [9]). This solver becomes prohibitively time-expensive on a large data-set such as Pascal VOC [3] with large amount of features, which is typical in Computer Vision. Although, the current state of the art for object detection utilizes the latent SVM, and [1] reports results on Pascal VOC, implementing the same on HPC2013, takes many days for training for a single object.

See Figure 3 for examples of images in Pascal VOC 2007, and the corresponding bounding boxes. Pascal VOC is known to be a difficult dataset with images that are non-trivial for even tagging by humans. See, for example, Figure 2

A simple algorithm that optimized the latent SVM surrogate in an online fashion was proposed in previous work by us. In this work, we incorporate this algorithm and demonstrate its utility in giving

[*]Parts of this report have been inherited from a previous report by the same author titled "A Perceptron Algorithm for Latent Variables". This work applies the algorithm identified previously to solve an important problem in a particularly suitable setting.

(a) Image containing cat and other objects with intersecting bounding boxes.


(b) Image containing many cows. Note that it is difficult even for the human eye to classify these as cows.

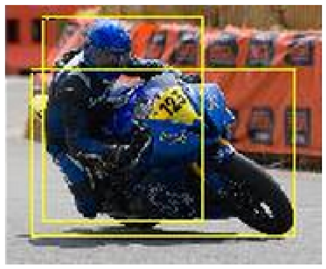
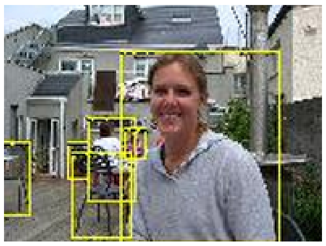
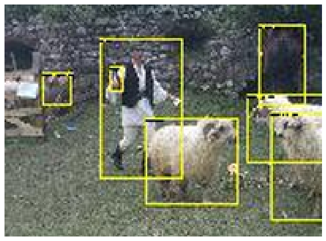

(c) Image containing two objects - a person, and a motorbike.

Figure 1: Sample images from Pascal VOC 2007, along with their bounding boxes. We consider the weakly supervised setting in which these bounding boxes are *not* provided while training.


(a) Object: Potted Plant


(b) Object: Bottle

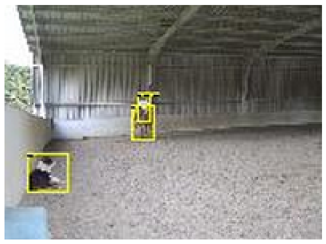

(c) Object: Horse

Figure 2: Some difficult images in Pascal VOC 2007

orders of magnitude speedups over the cutting-plane method. Further, we show that the algorithm can be seamlessly parallelized, so that size of the mini-batch could be the number of cores available. If this is ensured, the algorithm scales almost linearly with the amount of processing power available.

2 Latent Variables for Images

A latent variable incorporates information that is not available to the model while training. A latent variables lies in-between the input data, and the output tag. It allows the algorithm designer to encode in the algorithm, certain real-world information that it would leverage to solve a problem. The classical example is clustering using Gaussian Mixture Models, in which the latent variable is the mean of the Gaussians (learnt using Expectation-Maximization). The modern example is Deep Learning, in which the 'deep' part essentially refers to internal states that can be considered latent.

In the case of images, potential real world information could be the location of various parts of an object and the spatial relationship between them. Felzenswalb [4] used this idea in an influential paper, to obtain benchmark accuracies for Object Detection on Pascal VOC. Due to computational constraints, we however consider the most simple latent information possible - and that is the bounding box within which the object can be *localized*. What is the kind of latent information that is getting stored in such a formulation? We're essentially saying that real-world images are continuous blocks. It could be interesting to consider scenarios in which this is not the case, eg. when an object is partially hidden by another object, where the human brain is still able to identify the object owing to the Gestalt effect.

In any case, given whether an object is present in an image, its location, or equivalently its bounding box, becomes a latent variable. Let us formalize this notion. For a given image, identified by the index i, we identify a domain $\mathbb{D}$ over the joint space of the tag and the latent variable over which we are optimizing.

$$\mathbb{D} = (-1, \perp) \cup \{(1, h) \mid h \in \mathbb{H}\} \tag{1}$$

Here, $\mathbb{H}$ is the set of all bounding boxes under consideration. Given a $(y, h) \in \mathbb{D}$, y essentially refers to the tag we assign to the image, and h is the latent variable that identifies the bounding box that we are associating with the particular detection. Clearly if we detect -1, or the absence of the object, then we would not be identifying a particular bounding box that it goes with. The $\perp$ thus refers to considering the complete image instead of any particular bounding box.

How do we identify possible bounding boxes and *featurize* them for feeding into a classification model? Classically, we would consider a linear sweep of equally sized bounding boxes across the image, with or without overlap. Clearly however, this formulation has problems of scale (not considering the right-sized bounding boxes), and problems of considering insignificant bounding boxes (eg. containing a background object). Vision literature was mature enough to identify *significant* portions of the image, that could be potential regions with the object. Leveraging the semantic depth of CNN-features, Girschick et al [6] developed a framework that created region proposals based on available techniques in Computer Vision, and extracted 4096-dimensional feature vectors using Caffe [8] based on the CNN architecture described by Krizhevsky et al. [10]. This method is called RCNN (Region with CNN features). Thematically, their pipeline is described in Figure **??**.

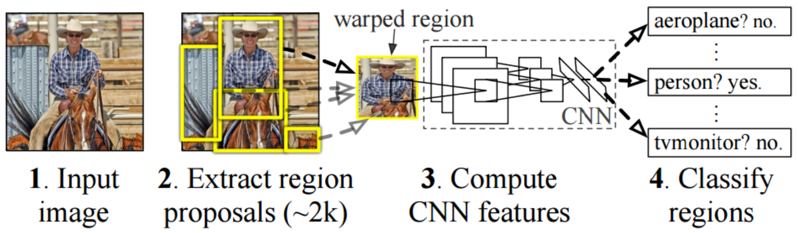


Figure 3: RCNN pipeline, as presented in [6]

What is the classification problem now? Minimize $\sum_{i=1}^{n} \mathbb{1}_{y_i}(y_i^*)$ where y_i refers to the true label, and y_i^* is the predicted label. The prediction could happen by *maximizing* over the latent variable,

$$(y^*, h^*) = \arg\max_{d \in \mathbb{D}} \mathbb{M}(d) \tag{2}$$

or *marginalizing* over the latent variable.

$$y^* = \begin{cases} 1 & \text{if } \int_{\mathbb{H}} \mathbb{P}(h)dh > \mathbb{P}(\perp) \\ -1 & oth. \end{cases} \tag{3}$$

The model $\mathbb{M}$ could in general be anything that gives a score. The probability distribution $\mathbb{P}$ could in general be anything that gives a normalized probability distribution. What we choose depends on the problem at hand. Hidden Conditional Random Fields (CRFs) [14] can be used to model the latter.

For our problem, it makes more sense to take the bounding box that best explains the presence of the object, rather than whether bounding boxes across the image indicate the presence of the object. We hence consider the maximum formulation in a latent SVM framework. In this setting, our model $\mathbb{M}$ will be a linear weight function w on the features Ψ along with a bias. As described above, we extract interesting regions as bounding boxes. For these bounding box, and also the complete image bounding box, we extract 4096-dimensional CNN features, and call this Ψ (for complete image, we will multiply it by -1).

3 Latent SVMs

Yu and Joachims [17] developed the latent SVM formulation to learn Structural SVMs [9] with latent variables. Like the Structural SVM formulation, this involves a joint feature Ψ that depends on both the input features x and the structured output y. Additionally, it depends on a latent variable h. The model w is linear in Ψ. The prediction is thus given by,

$$(y^*, h^*) = \arg\max_{(y,h)} w^T \Psi(x, y, h) \tag{4}$$

The loss incurred by a prediction (y^*, h^*) is given by $\Delta(y, y^*, h^*)$. The inclusion of the latent variable h^* in the loss function, is mainly because the latent SVM *allows* us to optimize over it; in most natural situations it simply decomposes as $\Delta(y, y^*)$. Given structured output, the definition of the loss function is not obvious de-facto. The objective to be minimized, over a group of n samples $S = \{(x_i, y_i) | i = 1...n\}$ is,

$$\sum_{i=1}^{n} \Delta(y_i, y_i^*, h_i^*) \tag{5}$$

Clearly, the objective is not a convex function of w. The optimization problem is infeasible. A general purpose surrogate was developed in [17] that upper bounds the above loss. It can be shown that

$$\sum_{i=1}^{n} \Delta(y_i, y_i^*, h_i^*) \leq \sum_{i=1}^{n} \left([\max_{y',h'} (w^T \Psi(x_i, y', h') + \Delta(y_i, y', h'))] - [\max_{h'} (w^T \Psi(x_i, y_i, h'))] \right) \tag{6}$$

The regularized max-margin formulation is now given as

$$\min_{w} \left[\frac{1}{2} ||w^2|| + \sum_{i=1}^{n} \left([\max_{y',h'} (w^T \Psi(x_i, y', h') + \Delta(y_i, y', h'))] \right) - \sum_{i=1}^{n} \left([\max_{h'} (w^T \Psi(x_i, y_i, h'))] \right) \right] \tag{7}$$

The first two quantities inside the objective are both convex in w, and the last quantity is negative convex. Hence, the overall objective is a difference of convex functions. The CCCP algorithm developed by Yuille & Rangarajan [18] finds a local minima for such functions, commonly known as DC functions.

The CCCP algorithm could be seen as a coordinate descent over the latent variables and the output variables. Iteratively, it convexifies the objective, by completing the latent variables, and then solving the (now-convex) optimization problem using Quadratic Programming. Without going into the details, the new optimization problem looks like -

$$\min_{w} \left[\frac{1}{2} ||w^2|| + \sum_{i=1}^{n} \left([\max_{y',h'} (w^T \Psi(x_i, y', h') + \Delta(y_i, y', h'))] \right) - \sum_{i=1}^{n} \left(w^T \Psi(x_i, y_i, \bar{h}) \right) \right] \tag{8}$$

where, $\bar{h} = \arg\max_{h'} (w^T \Psi(x_i, y_i, h'))$.

Thus, to convexify the problem, we need to find $\bar{h}$. Once this is done, we're left with a Quadratic Optimization problem, with $|\mathbb{D} = \mathbb{Y} \times \mathbb{H}|$ many constraints. This could potentially be an exponential number of constraints. Cutting-plane methods [9] go around the problem by considering only an increasing set of constraints per iteration, which is augmented with the *most-violated constraint* after the iteration. The hope is that the set of important, or *informative* constraints is small.

For our problem, the most-violated constraint is essentially the argmax over y and h. Hence, an efficient algorithm is required for 1) convexification (finding $\bar{h}$) and for 2) evaluating the most-violated constraint. Given these, the cutting plane algorithm solves the optimization problem.

However, cutting-plane methods are known to be extremely slow. Complexity of solving the QP is heavily dependent on how informative the constraints are. Cutting plane methods can only go around this if many constraints can grossly be ignored. Further, the optimization algorithm is completely batch, i.e. it requires all data points to be loaded in memory in one go. For large data-sets this imposes infeasible conditions over RAM availability, which is one major issue when training with large Vision datasets. We look at a perceptron-like framework that is completely online, and does not solve a QP internally, thus giving us both computational speedup, and lower memory usage.

4 A Perceptron update rule

It was suggested in the Pegasos paper [16], that the Perceptron can be seen as implementing a stochastic gradient step. We augment this understanding. A perceptron update is a stochastic gradient step, taken only when we make a *mistake*, or we misclassify. Let us have a look at the latent

SVM objective given in Equation 8 again, and see what could be a gradient step. Look at all terms whose inner products are taken with w, and perform a descent step in the opposite direction. Optionally, at every mistake, we can decrease the norm of w, by decrementing by a constant multiple of w. The proposed latent perceptron update rule then looks like the following -

$$w_{t+1} \leftarrow w_t + \eta_1(\Psi(x, y, \bar{h}) - \Psi(x, y^*, h^*)) - \eta_2 w_t \tag{9}$$

Here, $\bar{h}$ is obtained by taking the argmax by fixing y to the true output (last term in equation 8). y^*, h^* are obtained by taking the complete argmax. The parameters η can be chosen by cross validation. Improved performance is also seen if η falls with the number of iterations, as we approach the minima. Rate of fall could be another parameter that can be tuned by observing whether the objective is converging or diverging. The following algorithm summarizes our approach.

```
Data: Images in a stream, with features $x$ and tags $y \in \{+1, -1\}$
$t \leftarrow 0$;
$w_t \leftarrow random$;
while there are images left in the stream do
    Get next image $x$;
    Compute features for possible bounding boxes, to form $\Psi(1, \mathbb{H})$;
    Compute features for complete bounding box and multiply by $-1$, to form $\Psi(-1, \perp)$
     $(y^*, h^*) \leftarrow \arg\max w^T \Psi(D)$;

    read $y_{true}$;
    if $y^* != y_{true}$ then
        $\bar{h} \leftarrow \arg\max_h (w^T \Psi(x, y_{true}, h))$;
        Update according to equation 9;
        $t \leftarrow t + 1$;
    end
end
return $w_t$
```

Our perceptron update does something very intuitive. The Ψ obtained by taking the max over the latent variable, given the output variable ($\hat{h}$) enjoys more information by knowing the output variable. The Ψ obtained by taking the max over the joint domain of the both the latent variable and the output variable, on the other hand, is hazardous and wrong if we are misclassifying. It has a different (and incorrect) value for y, which is why we are misclassifying. We penalize the feature of the latent variable that best explains the incorrect y, and move towards the feature of the latent variable that best explains the correct y. A mistake bound for the perceptron update rule, with respect to the surrogate designed by Joachims was derived in the previous work. For sake of completeness, it is presented in the Appendix.

Parallelization

Although Joachims' latent SVM cannot be parallelized, the perceptron update can seamlessly be. Pascal VOC consists of around 2501 images for training. We divide all images into 40 mini-batches of around 60 images each. Then, the above algorithm is modified, so that we look at a mini-batch of 60 images in one go, calculating the update value for every image in parallel, and making a single update after the mini-batch is over. Clearly, each of the individual update steps are independent from one another. In this way, we allow the perceptron algorithm to make multiple runs through the data.

5 Experiments

For running experiments, we used the HPC2013 supercomputer at IIT Kanpur. Most runs reported were over the *hyperthread* thread provided by HPC2013. This is a highly parallel machine. Both classification scores and meanAveragePrecision (mAP) scores were quite bad, and we are working on understanding bugs that could be existing in our implementation. In terms of time, Joachims' formulation took more than 100 hours to perform 7 outer loop iterations (latent variable completion + cutting-plane solver), whereas our approach took on an average 4-5 minutes to run through one mini-batch (60 images). This amounts to around 3-4 hours for one run through the entire data-set.

Acknowledgments

Apart from the advisors, the work was done in collaboration with Ayush Mittal, a Dual-degree student. We also recognize using code written by Subhrata Debnath, passed out Master's student. Invaluable assistance regarding implementation of Joachims' code was provided by Soumya Roy. Most experiments were run on HPC2013, the supercomputer at IIT Kanpur. We are grateful to the support staff for allowing us to exceed the data storage limits.

References

[1] Hakan Bilen, Vinay P Namboodiri, and Luc Van Gool. Object and action classification with latent variables. In *The British Machine Vision Conference*, 2011.

[2] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of on-line learning algorithms. *Information Theory, IEEE Transactions on*, 50(9):2050–2057, 2004.

[3] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[4] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[5] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[7] Tamir Hazan, Joseph Keshet, and David A McAllester. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, pages 1594–1602, 2010.

[8] Yangqing Jia. Caffe: An open source convolutional architecture for fast feature embedding, 2013.

[9] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*, pages 3–10, 2007.

[12] Mehryar Mohri and Afshin Rostamizadeh. Perceptron mistake bounds. *arXiv preprint arXiv:1305.0208*, 2013.

[13] Albert BJ Novikoff. On convergence proofs for perceptrons. Technical report, DTIC Document, 1963.

[14] Ariadna Quattoni, Sybor Wang, Louis-Philippe Morency, Michael Collins, and Trevor Darrell. Hidden conditional random fields. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (10):1848–1852, 2007.

[15] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[16] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[17] Chun-Nam John Yu and Thorsten Joachims. Learning structural svms with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1169–1176. ACM, 2009.

[18] Alan L Yuille and Anand Rangarajan. The concave-convex procedure. *Neural computation*, 15(4):915–936, 2003.

Appendix

We present a mistake bound for the perceptron presented in 9. The given bound is for the case, $\eta_1 = 1$ and $\eta_2 = 0$.
We assume that for all (y, y', h'), either $\Delta(y, y', h') = 0$, or $\Delta(y, y', h') \geq \beta$ for some $\beta \in \mathbb{R}^+$.
Also assume that for all (y, y', h'), $||\Psi(y, y', h')||_2 \leq R$. Suppose the data stream runs for T iterations.
Define $\Delta_T = \sum_{t=1}^{T} \Delta(y_t, y_t^*, h_t^*)$, where (y_t^*, h_t^*) is the predicted output.
The surrogate loss $\hat{\Delta}$, as defined previously, is,
$\hat{\Delta}_t = \left([\max_{y',h'} (w_{t-1}^T \Psi(x_t, y', h') + \Delta(y_t, y', h'))] - [\max_{h'} (w_{t-1}^T \Psi(x_t, y_t, h'))] \right)$
We define $\hat{\Delta}_T(w) = \sum_{t=1}^{T} \hat{\Delta}_t(w)$, for any w.
We also set the notation $P_t(w) = w^T w_t$. Henceforth, any norm, unless otherwise specified, shall refer to the L_2 norm.

Theorem 5.1.

$$\Delta_T \leq \inf_w \left(\frac{2R||w||}{\beta} + \sqrt{\hat{\Delta}_T(w) + 2RT||w||} \right)^2 \tag{10}$$

Proof.

Lemma 5.2.

$$||w_t||^2 \leq ||w_{t-1}||^2 + 4R^2 \leq ||w_{t-1}||^2 + \frac{4R^2 \Delta_t}{\beta} \tag{11}$$

Proof. We show the first inequality for the non-trivial case when $w_t \neq w_{t-1}$. The second inequality follows trivially.
Let $v_t = \Psi(x, y, \hat{h}) - \Psi(x, y^*, h^*)$. Then, $w_t = w_{t-1} + v_t$.
$||w_t||^2 = ||w_{t-1}||^2 + ||v_t||^2 + 2w_{t-1}^T v_t$
$||v_t|| \leq 4R^2$, and $w_{t-1}^T v_t \leq 0$ (since the second term in v_t is a global max), and the result follows. □

Lemma 5.3.

$$P_t(w) \geq P_{t-1}(w) + \Delta_t - \hat{\Delta}_t(w) - 2R||w|| \tag{12}$$

Proof. For the case $w_t = w_{t-1}$, $P_t(w) = P_{t-1}(w)$.
Since, $\hat{\Delta}_t \geq \Delta_t$, the statement holds.
In the other case,
$P_t = P_{t-1} + w^T v_t$
$= P_{t-1} + w^T \Psi(x_t, y_t, \hat{h_t}) - w^T \Psi(x_t, y_t^*, h_t^*)$
$= P_{t-1} + w^T \Psi(x_t, y_t, \hat{h_t}) - w^T \Psi(x_t, y_t, \hat{h_t}^w) + Q_t$
where, $Q_t = w^T \Psi(x_t, y_t, \hat{h_t}^w - w^T \Psi(x_t, y_t^*, h_t^*)$, and $\hat{h_t}^w$ is the same argmax when taken with w as the model variable instead of w_t.
It can be shown in a straightforward manner that $Q_t \geq \Delta_t - \hat{\Delta}(w, x_t, y_t)$.
We would then have,
$P_t \geq P_{t-1} + \Delta_t - \hat{\Delta}(w, x_t, y_t) + w^T(\Psi(x_t, y_t, \hat{h_t}) - \Psi(x_t, y_t, \hat{h_t}^w))$
$\geq P_{t-1} + \Delta_t - \hat{\Delta}(w, x_t, y_t) - 2R||w||$ □

It follows from Lemma 5.2, by taking a telescopic sum -

$$||w_T||^2 \leq \frac{4R^2}{\beta} \Delta_T \tag{13}$$

Similarly, from Lemma 5.3 it follows that -

$$w_T^T w = P_t \geq \Delta_T - \hat{\Delta}_T(w) - 2RT||w|| \tag{14}$$

Since, $||w_T|| \cdot ||w|| \geq w_T^T w$, we have that,

$$\Delta_T - \hat{\Delta}_T(w) - 2RT||w|| \leq ||w|| \cdot 2R\sqrt{\frac{\Delta_t}{\beta}} \tag{15}$$

On completing the square, the theorem follows. □