# A Perceptron Algorithm for Latent Variables

**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

The project began with a goal to understand the latent SVM surrogate and suggest alternate formulations or devise bounds on the existing one. Various structured learning formulations, and the behavior of their surrogates was explored. Later, a perceptron-like algorithm was devised, which stochastically optimized the latent SVM surrogate. It had a mistake bound, and gave comparable post-learning performance to the original batch optimizer which solved a QP optimization problem and hence was prohibitively slow in some cases. Further, it enhanced understanding of the latent SVM. Future directions could be to compare the batch and perceptron optimization approaches in their raw form in an attempt to understand how each of them work. The work could lead to a general perceptron learning rule as an alternative to a lot of optimization problems.

## 2 Perceptron-like algorithms

Perceptron-like algorithms have recently become popular due to their immense scalability, and provable bounds. The basic perceptron algorithm was proposed by Rosenblatt [7] in 1958 for classification in the separable case.

```
Data: Data-points in a stream, with features $x$ and tags $y \in \{+1, -1\}$
$t \leftarrow 0$;
$w_t \leftarrow random$;
$w_{MODEL} \leftarrow w_t$;
while there are data-points left in the stream do
    read $x$;
    $y' \leftarrow sgn(w_t^T x)$;
    read $y$;
    if $y' != y$ then
        $w_{t+1} \leftarrow w_t + y * x$;
        $t \leftarrow t + 1$;
    end
    $update(w_{MODEL}, w_t)$;
end
return $w_{MODEL}$
```

### Mistake bounds

Online learning algorithms predict, possibly incur a loss, and modify the model according to the result. We define a mistake bound as the number of times the online algorithm makes a *mistake*, or incurs a non-zero loss. Equivalently, since by definition a perceptron updates exactly when it incurs a non-zero loss, the mistake bound counts the number of times the perceptron updates.

Novikoff [6] gave the first mistake bound for perceptrons in 1963. Freund and Schapire [2] extended the algorithm and gave a mistake bound for the non-separable case in terms of the $L_1$ norm of the losses incurred by the model for general $\rho$-margin losses. Mohri [5] gave a finer bound in terms of the $L_2$ norm. Cesa-Bianchi [1] gave a generalization bound that allows conversion of the online setting to the batch setting. We require the batch setting if we wish to output a final model, as given by $w_{MODEL}$ above.

Let the $\rho$ margin hinge loss for a model $u$, cumulatively over all data points be $L_\rho(u)$. Further assume that all input points lie within an $L_2$ ball around the origin of radius $r$. Then the following mistake bound $M$ is presented in [2].

$$M \leq \inf_{\rho > 0, ||u|| \leq 1} \left( \frac{r}{\rho} + \sqrt{L_{rho}(u)} \right)^2 \tag{1}$$

For a nice concise proof, see [5]. The bound essentially says that the number of mistakes is not much worse than the loss incurred by the best possible model.

### Perceptron for structured and latent settings

Structured losses are different from standard 0-1 losses. They could be non-decomposable over samples in the sense that they are not simply the sum of losses incurred on individual samples, but depend on the output on the entire dataset as a whole. More generally, they are losses defined for multi-class output which generically covers outputs such as parse trees, part models, etc. Examples of structured losses are Precision@k and area under the ROC curve.

Hazan et al [3] gave a perceptron update rule for directly optimizing structured losses. Previous techniques like structural Conditional Random Fields and Support Vector Machines optimized surrogates on these losses. The update rule was fast, online, and outperformed these standard algorithms on the TIMIT corpus. Their work inspires one of our update rules.

## 3 Latent Variables

Often, the final output variables (the tag) of a data-point is all that is available in a machine learning task, and critical real-world hidden information is unavailable for training. A standard technique is to treat this information as a latent variable, and optimize over the joint domain of the latent variables and the output variables. The likelihood in the case of maximum likelihood estimates, or the objective function in the case of optimization settings, are then constructed so as to include both a term for the latent variable, and a term for the output given the latent variable.

In maximum likelihood settings, this problem is best solved by the EM algorithm that alternatively takes the expectation of the latent variable and maximizes the likelihood of the model variable. An example is Gaussian Mixture Models (GMMs) where the latent variable is used to encode the identity of the Gaussian from which a particular point is generated. The parameters of the Gaussian are learned through EM.

Some examples of discriminative learning tasks that benefit from the use of latent variable are -

### Document retrieval

Precision@k is a good performance measure for document retrieval task, since often the few documents on the top are all that are important. Once we're optimizing precision@k, the ranking of documents becomes important. The latent variable in this case would thus be an overall ranking on all the documents. This is latent, since we are only given whether a document is relevant or not, and not how relevant it is.

**Object detection**

Objects in real world images are continuous blocks, often modeled in vision literature as bounding boxes. Thus, given whether an object is present in an image, its location, or equivalently its bounding box, becomes a latent variable.

## 4 Latent SVMs

Yu and Joachims [9] developed the latent SVM formulation to learn Structural SVMs with latent variables. Like the Structural SVM formulation, this involves a joint feature $\Psi$ 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 $\Psi$. The prediction is thus given by,

$$(y^*, h^*) = \arg\max_{(y,h)} w^T \Psi(x, y, h) \tag{2}$$

The loss incurred by a prediction $(y^*, h^*)$ is given by $\Delta(y, y^*, h^*)$. The loss is typically not defined de-facto, and defining a good loss function is itself a challenge. The objective to be minimized, over a group of $n$ samples $S = \{(x_i, y_i) | i = 1...n\}$ is,

$$\sum_{i=1}^{n} \Delta(y_i, y_i^*, h_i^*) \tag{3}$$

Clearly, the objective is not a convex function of $w$. The optimization problem is infeasible. A general purpose surrogate was developed in [9] that upper bounds the above loss. It can be shown that

$$\sum_{i=1}^{n} \Delta(y_i, y_i^*, h_i^*) \leq \sum_{i=1}^{n} \left( [\max_{y',h'} (w^T \Psi(x_i, y', h') + \Delta(y_i, y', h'))] - [\max_{h'} (w^T \Psi(x_i, y_i, h'))] \right) \tag{4}$$

The regularized max-margin formulation is now given as

$$\min_{w} \left[ \frac{1}{2} ||w^2|| + \sum_{i=1}^{n} \left( [\max_{y',h'} (w^T \Psi(x_i, y', h') + \Delta(y_i, y', h'))] \right) - \sum_{i=1}^{n} \left( [\max_{h'} (w^T \Psi(x_i, y_i, h'))] \right) \right] \tag{5}$$

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 [10] finds a local minima for such functions, commonly known as ACDC functions.

The range of the output variable $y$ could potentially be exponential. The optimization routines used to minimize the above quantity typically need to evaluate the most violated constraint, which boils down to evaluating the above argmax'es over $y$ and $h$. Thus, the problem is only feasible when there is an efficient method to evaluate them. The surrogate has to be engineered appropriately and an internal efficient algorithm has to be given to facilitate the evaluation of the most violated constraint efficiently.

Given an efficient algorithm to compute the most violated constraints, a QP optimization problem is solved, which is prohibitively slow and is batch, i.e. it requires all data points to be loaded in memory. We attempt to convert this optimization problem into a perceptron-like framework. The next section deals with our formulation, and bounds for the same.

## 5 Perceptron learning rule for latent SVM surrogate

Let us have a look at the latent SVM objective given in Equation 5 again. In the seminal Pegasos paper [8], the authors suggested that the Perceptron can be seen as implementing a stochastic gradient step. Taking inspiration from this observation, we look at the terms whose inner products are taken with $w$, and perform a descent step in the opposite direction (since the objective is to be minimized). 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 -

$$w_{t+1} \leftarrow w_t + \eta_1(\Psi(x, y, \hat{h}) - \Psi(x, y^*, h^*)) - \eta_2 w_t \tag{6}$$

Here, $\hat{h}$ is the $h$ obtained when the argmax is taken by fixing $y$ to the true output (last term in equation 5). $y^*, h^*$ are obtained by taking the complete argmax. The parameters $\eta$ can be chosen by cross validation. Improved performance is also seen if the parameters fall with the number of iterations.

Before we report results with this update rule, and give a mistake bound, let us compare this update rule to the classical perceptron update rule. The perceptron learning rule given in Algorithm 1 does something very intuitive. Whenever it classifies a postive data-point as negative, it moves towards that data-point; whenever it classifies a postive data-point as negative, it moves away from that data-point.

We propose that our update rule can be inspired in a similar manner. The $\Psi$ obtained by taking the max over the latent variable, given the output variable ($\hat{h}$) enjoys much more information by knowing the output variable. The $\Psi$ 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. It has a different (and incorrect) value for $y$, since otherwise the max'es would both be the same. We penalize the feature of the latent variable that best explains that particular $y$, since it shouldn't ideally do so.

## 6 A Mistake Bound

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.

**Lemma 6.1.**

$$||w_t||^2 \leq ||w_{t-1}||^2 + 4R^2 \leq ||w_{t-1}||^2 + \frac{4R^2 \Delta_t}{\beta} \tag{7}$$

*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 6.2.**

$$P_t(w) \geq P_{t-1}(w) + \Delta_t - \hat{\Delta}_t(w) - 2R||w|| \tag{8}$$

*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 6.1, by taking a telescopic sum -

$$||w_T||^2 \leq \frac{4R^2}{\beta}\Delta_T \tag{9}$$

Similarly, from Lemma 6.2 it follows that -

$$w_T^T w = P_t \geq \Delta_T - \hat{\Delta}_T(w) - 2RT||w|| \tag{10}$$

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

$$\Delta_T - \hat{\Delta}_T(w) - 2RT||w|| \leq ||w|| \cdot 2R\sqrt{\frac{\Delta_t}{\beta}} \tag{11}$$

Consequently, we obtain our mistake bound given in the following theorem.

**Theorem 6.3.**

$$\Delta_T \leq \inf_w \left(\frac{2Rw}{\beta} + \sqrt{\hat{\Delta}_t(w) + 2RT||w||}\right)^2 \tag{12}$$

## 7 Results

We compared our algorithm to the benchmark set by the original latent SVM formulation [9] on the OHSUMED tasks of the LETOR 3.0 dataset [4] for optimizing Precision@k. The model is exactly the same as the one used in the latent SVM paper. Only the optimization is through a perceptron stochastically.

We demonstrate catastrophic improvements in runtime, for only a very small loss in error. We in fact improve upon an earlier, but popular Ranking SVM formulation which also runs considerably slowly. Our results are contained in the table below. The paramters of the algorithm are chosen by cross validation. Runtime results are yet to be normalized and tabulated.

| Algorithm | P@1 | P@3 | P@5 | P@10 |
|---|---|---|---|---|
| Ranking SVM | 0.597 | 0.543 | 0.532 | 0.486 |
| **Latent Perceptron** | **0.5575** | **0.5617** | **0.5432** | **0.487** |
| Latent SVM | 0.680 | 0.573 | 0.567 | 0.494 |

Table 1: Precision@k on OHSUMED (with 5-fold CV)

We are still working on comparing our algorithm to other approaches, and demonstrating generic performance across problems. Only preliminary results have been presented here.

## References

[1] 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.

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

[3] 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.

[4] 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.

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

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

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

[8] 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.

[9] 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.

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