Courant Institute of Mathematical Sciences

Deep Cascades with Perceptron

Ankit Vani
Srivas Venkatesh
December 21, 2015

1 Introduction

The Deep Cascades [1] algorithm generates a cascade, using rich families of functions, structured as a decision tree with the node question and leaf predictors being picked from these rich families. The algorithm exhibits strong data dependent guarantees that allow for use of such rich families of functions. The cited paper talks about using kernelized SVMs for training cascades with complex hypotheses at the leaf nodes and elaborates on how the deep cascades algorithm is adapted for use with SVMs.

We look at adapting the Deep cascade algorithm to work with the Perceptron algorithm [2] and it's kernelized variant [3] to provide a rich set of functions. The perceptron algorithm learns a linear separator in an online fashion, examining one point at a time. We have, however, used it in a batch setting making use of polynomial kernels.

2 Background on Deep Cascades, Perceptron

2.1 Deep Cascades

The work on Deep cascades presents data dependent generalization guarantees for cascades with leaf predictors chosen from hypothesis sets H_k and node questions selected from hypothesis sets Q_j. In the simplest form they present the following guarantee on the

generalization error:

$$R(f) \leq \widehat{R}_S(f) + \sum_{k=1}^{l} \min \left(4 \left[\sum_{j=1}^{d_k} \widehat{\mathfrak{R}}_S(Q_j) + \widehat{\mathfrak{R}}_S(H_k), \frac{m_k^+}{m} \right] \right) + \widetilde{O}\left(l\sqrt{\frac{\log pl}{m}} \right) \tag{2.1}$$

where l is the number of levels in the cascade. We define $\mathcal{H}$ to be the set of hypothesis sets from which the hypothesis sets H_k are selected, and $\mathcal{Q}$ to be the set of hypothesis sets from which the sets Q_k are selected. p is the number of distinct hypothesis sets we can choose. Additionally, m_k^+ is the number of points reaching the leaf k and are classified correctly, with m being the total number of sample points. Since we consider cascades with small depths, the last term is practically insignificant and hence any cascade minimizing the first 2 terms will provide good generalization guarantees.

With these guarantees in mind the paper provides an adaptation of the algorithm for use with SVMs at each level. The topology of the cascade is as shown in Figure 2.1

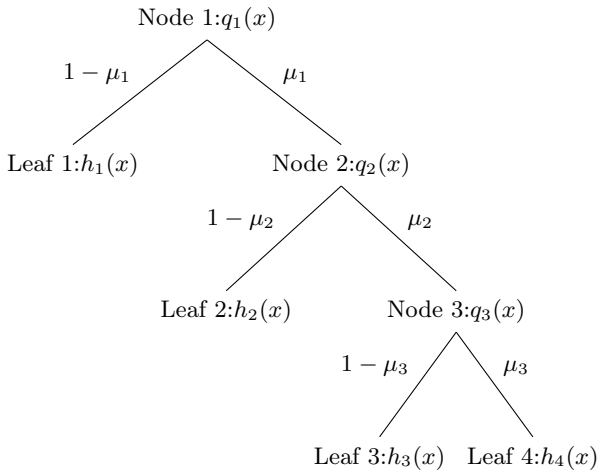

Figure 2.1: Topology of Deepcascade SVM

That is, at each internal node, an SVM is trained on the incoming points. Then the closest fraction μ_k of points to the separating hyperplane are removed from the current set of points and sent to the next level for it to be trained with a new SVM with a possibly different kernel. The remainder of $1 - \mu_k$ fraction of points stay classified by the same hyperplane which is treated as that particular leaf's hypothesis.

The pseudocode for the DeepCascadeSVM is as follows:

Algorithm 1 Deep Cascade SVM

1: **function** DEEPCASCADESVM(L,$\mathcal{M}$,γ)
2: **for** $l \in [1 \cdots L], (\mu_k)_{1 \leq k \leq l} \subseteq \mathcal{M}, (\delta_k)_{1 \leq k \leq l} \subseteq \mathcal{G}$ **do**
3: $S_1 \leftarrow S$

4: **for** $k = 1$ to l **do**
5: $h_k \leftarrow \text{SVM}(\delta_k, S_k)$
6: $q_k \leftarrow \arg_{q \in Q_k}\{|q^{-1}(1) \cap S_k| = \mu_k |S_k|\}$
7: $S_{k+1} \leftarrow q_k^{-1}(1) \cap S_k$
8: **end for**
9: $f(\cdot) \leftarrow \sum_{k=1}^{l-1} \left(\prod_{j=1}^{k-1} q_j\right) \bar{q_k} h_k + \left(\prod_{j=1}^{l} q_j\right) h_l$
10: $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$
11: **end for**
12: $f^* \leftarrow \text{argmin}_{f \in \mathcal{F}} \widehat{R}_s(f) + \sum_{k=1}^{l} \min\left(4\gamma \left[\sum_{j=1}^{d_k} \sqrt{\frac{d_{f,j} \log \frac{em}{d_{f,j}}}{m}} + \sqrt{\frac{d_{f,k} \log \frac{em}{d_{f,k}}}{m}}\right], \frac{m_k^+}{m}\right)$
13: **end function**

The deep cascade functions generated by the algorithm are based on repeatedly using SVMs combined with polynomial kernels of different degree. The leaf hypothesis sets H_k are decision surfaces defined by polynomial kernels (hyperplanes in the kernel hilbert space). The hypothesis $h_k \in H_k$ is learned via the SVM algorithm with a polynomial kernel of degree δ_k on subsample S_k . The node question hypothesis set Q_k is the set of indicator functions which indicate if a point is within a distance c from the hyperplane such that $\mu_k |S_k|$ points lie within that distance c from the hyperplane.

A search is done over all depths $[1 \cdots L]$, fraction of points $\mu_k \in \mathcal{M}$ and degree of kernels $\delta_k \in \mathcal{G}$ to find a cascade that minimizes the generalization error using the aforementioned bounds. We also see that the Rademacher complexities from (2.1) have been bounded by the VC dimension of the respective SVM $(d_{f,j}, d_{f,k})$ and that a factor γ has been introduced to scale the hinge loss. Thus the DeepCascadeSVM only tries to get the minimal upper bound in terms of the VC dimension rather than trying to minimize the Rademacher complexities.

2.2 Perceptron Algorithm

The perceptron algorithm is one of a broad family of online learning algorithms and has quite a few variants. We are looking at the kernelized perceptron variant of the same. The original perceptron algorithm is an "error driven" algorithm that trains on the dataset to learn a separating linear homogeneous threshold function given as $w \cdot x = 0$. Each time a mistake is made on point x_i, w is adjusted by $x_i y_i$.

In the case of the kernelized perceptron we actually need to learn a hyperplane in the kernel hilbert space, for which we first need to identify the dual formulation. This is done by observing that w can be expressed as linear combination of the training samples. The equation being $w = \sum_{i=1}^{m} \alpha_i y_i x_i$. Then we can see that adjusting w with $y_i x_i$ in the primal form is the same as adjusting the α_i by 1. Now we see that the predicted $\widehat{y}_i$ is given using

$sgn(w \cdot x_i)$. Writing this with the dual we see that:

$$\begin{aligned} \widehat{y}_t &= sgn(w^T x_t) \\ &= sgn\left(\sum_{i=1}^{m} \alpha_i y_i x_i\right)^T x_t \\ &= sgn \sum_{i=1}^{m} \alpha_i y_i (x_i \cdot x_t) \end{aligned} \tag{2.2}$$

Since it has come to a dot product form we can now use kernels to evaluate the same. This is the principle behind kernelized perceptron. Also this can be adapted to a batch setting by simply running the algorithm over all the points and repeating for a few passes if needed. The psuedocode is as follows:

Algorithm 2 Batch Kernelized Perceptron

function KernelizedPerceptron(S, K) ▷ Where S is the sample set and K is the kernel
  $\alpha \leftarrow \alpha^0$ ▷ Where α^0 is the α corresponding to the initial w,typically 0
  for $p = 1$ to $passes$ **do**
    for $t = 1$ to T **do**
      Receive($x_t \in S$)
      $\widehat{y}_t \leftarrow sgn \sum_{i=1}^{m} \alpha_i y_i K(x_i, x_t)$
      Receive($y_t \in S$)
      if $\widehat{y}_t \neq y_t$ **then**
        $\alpha_t \leftarrow \alpha_t + 1$
      end if
    end for
  end for
  return α
end function

The point to be noted with regards to perceptrons is the fact that training time for this is linear in the number of samples.

3 Algorithm

We have adapted the deep cascades to work with kernelized perceptrons serving as the leaf hypothesis and node questions. The adaptation is similar to the DeepCascadeSVM with a few subtle differences. The psuedocode is as follows:

Algorithm 3 Deep Cascade Kernelized Perceptron

function DeepCascadePerceptron($S, L, \mathcal{M}, \mathcal{G}, \gamma$)
  for $l \in [1 \cdots L], (\mu_k)_{1 \leq k \leq l} \subseteq \mathcal{M}, (\delta_k)_{1 \leq k \leq l} \subseteq \mathcal{G}$ **do**
    $S_1 \leftarrow S$
    for $k = 1$ to l **do**
      $h_k \leftarrow$ KernelizedPerceptron($S_k, poly\_kernel(\delta_k)$)
      $threshold \leftarrow c_k : |\{x_t : |\sum_{i=1}^{|S_k|} \alpha_i y_i K(x_i, x_t)| \leq c_k\}| = \mu_k |S_k|$
      $q_k \leftarrow 1_{|\sum_{i=1}^{|S_k|} \alpha_i y_i K(x_i, x)| \leq threshold}$

8: $S_{k+1} \leftarrow q_k^{-1}(1) \cap S_k$
9: **end for**
10: $f(\cdot) \leftarrow \sum_{k=1}^{l-1} \left(\prod_{j=1}^{k-1} q_j\right) \bar{q_k} h_k + \left(\prod_{j=1}^{l} q_j\right) h_l$
11: $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$
12: **end for**
13: $f^* \leftarrow \text{argmin}_{f \in \mathcal{F}} \widehat{R}_s(f) + \sum_{k=1}^{l} \min\left(4\gamma \left[\sum_{j=1}^{d_k} \sqrt{\frac{d_{f,j} \log \frac{em}{d_{f,j}}}{m}} + \sqrt{\frac{d_{f,k} \log \frac{em}{d_{f,k}}}{m}}\right], \frac{m_k^+}{m}\right)$
14: **end function**

We standardize the features by centering around the mean and scaling to unit variance for the samples before being sent to the algorithm. This is done to center the data and normalize it. The same scaling is applied to the test data as well (using the parameters used to scale the training data). This centered scaling becomes quite important in the case of perceptrons as it learns only a linear homogeneous threshold function.

The algorithm essentially consists of training perceptrons h_k at each level (total of l levels) with the incoming points using polynomial kernels of dimension δ_k. Once the perceptron at level k is trained on S_k, the distances to all points are measured as $\Phi(w_k) \cdot \Phi(x) = K(w_k, x) = \sum_{i=1}^{|S_k|} \alpha_i y_i K(x_i, x)$ (ignoring the factor of $\|\Phi(w_k)\|$ as we just want to get the μ_k fraction of the closest points). With the distances calculated we decide on a threshold as the distance from the hyperplane such that $\mu_k |S_k|$ points lie within that distance. The node question is then: whether or not a given point lies within this threshold from the trained hyperplane. If yes, the point is sent to the next level. This process is then repeated for each level.

The algorithm generates multiple cascades for each of the depths $l \in [1 \cdots L]$, any sequence of fraction values $(\mu_k)_{1 \le k \le l} \subseteq \mathcal{M}$ and sequence of polynomial kernel degrees $(\delta_k)_{1 \le k \le l} \subseteq \mathcal{G}$. The best cascade f^* is chosen as the one that minimizes an upper bound on the generalization error bound of (2.1) similar to what is done for the DeepCascadeSVM [1]. That is, f^* is chosen to minimize the bound:

$$R(f) \le \widehat{R}_s(f) + \sum_{k=1}^{l} \min\left(4\gamma \left[\sum_{j=1}^{d_k} \sqrt{\frac{d_{f,j} \log \frac{em}{d_{f,j}}}{m}} + \sqrt{\frac{d_{f,k} \log \frac{em}{d_{f,k}}}{m}}\right], \frac{m_k^+}{m}\right) \tag{3.1}$$

where d_k is the depth of node k and $d_{f,k}$ is the VC dimension of the perceptron trained at level k. The parameter γ is introduced to scale down VC dimension upper bound since the original bounds are in terms of Rademacher complexities, which are harder to compute. We perform cross-validation to find the best γ from a set of values.

The VC dimension of a perceptron accepting samples of dimension n and working with a polynomial kernel of degree δ is $\binom{n+\delta}{\delta}$. This can be seen by the definition of polynomial kernel: $(K(x, y) = (1 + x \cdot y)^\delta)$ and using multinomial theorem.

Also since we are generating multiple cascades and since all cascades are independent of one another, we can parallelize the the process. We do this in our algorithm by training for trees with different max levels as separate processes. Similarly the cross validation for different values of γ is also done in parallel.

4 Experiments

We performed some preliminary experiments on the *breastcancer, ionosphere* datasets from UCI machine learning repository [4].

In our experiments we used the following parameters: max cascade depth $L = 4$, the set of fraction values was chosen as $\mathcal{M} = \{0.2, 0.4, 0.6, 0.8\}$ and the degree of the kernels were chosen from $\mathcal{G} = \{1 \cdots 4\}$.

We split our dataset into a 80% training set and 20% test set. The training set was used to generate 3 folds to perform 3-fold cross-validation with. We used this cross validation to figure out the best $\gamma \in \{10^{-i} : i \in \{0, 1, 2, 3\}\}$.

The results of our algorithm on these datasets is compared with the results of SVM and kernelized perceptron trained with polynomial kernels. The degree of the kernel to be used and the SVM hyper-parameters are determined by doing a 3-fold cross-validation as described for the DeepCascadePerceptron. The results are shown in Table 4.1 and Table 4.2.

Dataset	Number of examples	Number of features	DeepCascadePerceptron Validation error ± Deviation	SVM Validation error ± Deviation
breastcancer	699	9	0.036 ± 0.008	0.027 ± 0.004
ionosphere	351	34	0.071 ± 0.032	0.093 ± 0.033
german	1000	24	0.321 ± 0.012	0.236 ± 0.006

Table 4.1: Cross Validation error for DeepCascadePerceptron

Dataset	Cascade Depth	DeepCascadePerceptron	SVM	Perceptron
breastcancer	4	0.02	0.057	0.036
ionosphere	2	0.126	0.126	0.112
german	2	0.26	0.215	0.235

Table 4.2: Test error for DeepCascadePerceptron, SVM, Kernelized Perceptron run on the same set

5 Conclusions

Deep cascades learn a decision function that takes advantage of strong learning guarantees while still being able to use complex hypothesis sets at the leaves of the cascade. Previous work elaborated on using deep cascades with SVMs to learn hypotheses at the leaf nodes. We present an algorithm for training deep cascades with perceptrons as leaf predictors. Using perceptron results in a faster training time than an SVM-based cascade, while still benefiting from nature of the cascade and the corresponding learning guarantees.

We show that the results obtained by using Deep Cascades with perceptron is comparable to that of kernelized SVM and kernelized perceptron. We can further improve our results by choosing a larger set of μ's to choose from, and considering deeper cascades.

The code for our algorithm is available on Github [6]

References

[1] Giulia DeSalvo, Mehryar Mohri, and Umar Syed. "Learning with Deep Cascades". In: *Proceedings of the Twenty-Sixth International Conference on Algorithmic Learning Theory (ALT 2015)*. 2015.

[2] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[3] M. A. Aizerman, E. M. Braverman, and Lev I. Rozonoer. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and Remote Control* 25 (1964), p. 821.

[4] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[5] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. Adaptive computation and machine learning series. Cambridge (Mass.), London: MIT Press, 2012. ISBN: 978-0-262-01825-8. URL: `http://opac.inria.fr/record=b1134085`.

[6] Srivas Venkatesh Ankit Vani. *perceptron-deepcascades*. 2015. URL: `https://github.com/ankitkv/perceptron-deepcascades`.