Progress Report
1-20 May 2013

Amin Karbasi
E-mail: amin,karbasi@epfl.ch
Amir Hesam Salavati
E-mail: hesam.salavati@epfl.ch

Supervisor: Prof. Amin Shokrollahi
E-mail: amin.shokrollahi@epfl.ch

Algorithmics Laboratory (ALGO)
Ecole Polytechnique Federale de Lausanne (EPFL)

1 Summary

In the past 20 days, we worked on extending the idea of learning polynomial non-linear constraints to the case of classifiers. More specifically, instead of learning the non-linear curve that is "orthogonal" to the data in one class, we learn the non-linear curve that separates two classes in the n-dimensional space. This is hardly a new topic as there are numerous non-linear kernel SVMs. However, the proposed method here seems to be easy to implement and rather fast. More extensive investigations are necessary though.

We was also made some progress with the master semester projects. In one of the projects, Maximization of Mutual Information (MMI) idea has been successfully applied to the CIFAR-10 dataset. Preliminary results look very fascinating.

I the other project, a GUI is being built to apply different feature extraction techniques for compressing the images while maintaining the reconstruction quality to an acceptable level.

2 Classifiers Based on Polynomial Non-Linear Constraints

In the last report, we explained the idea of learning polynomial non-linear equations from the input data. Now the same technique can be used to implement a classifier. More specifically, we can now learn the non-linear line that separates the two classes. To this end, we are going to use the same two-layer neural network we used last time, as shown in Figure 1.

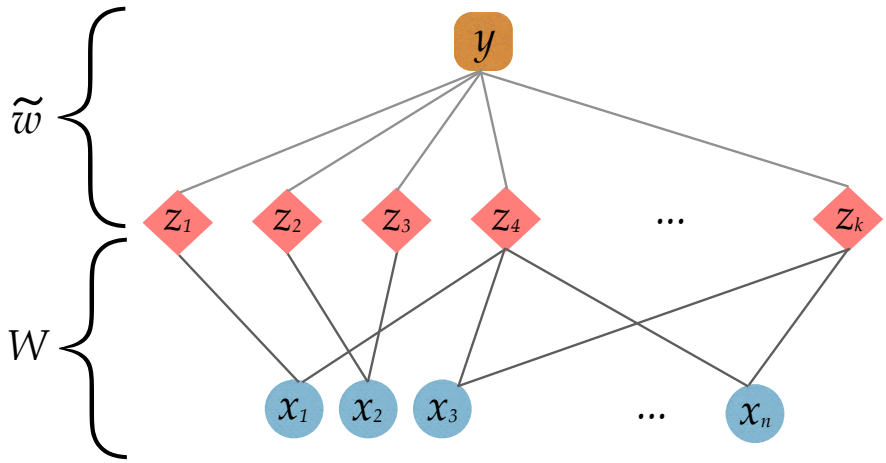

Figure 1: The neural network for classifying based on a polynomial kernel.

Let $x_1, \ldots, x_n$ represent the elements of an n-dimensional input pattern x that belongs to class

c. For simplicity, we assume there are only two classes $c = 1$ and $c = -1$.[1] We first apply a non-linear polynomial mapping to x, which yields a new data vector z. As before, the trick is to give $\ln(x) = [\ln(x_1), \ldots, \ln(x_n)]$ to the input part of the first layer. Therefore, the sate of the output neuron i in the first layer is determined by

$$z_i = f([Wx]_i) = \exp(\sum_{j=1}^{n} W_{ij} \ln(x_j)) = \prod_{j=1}^{n} x_j^{W_{ij}}$$

Now, we are interested in finding a weight vector $\tilde{w}$ such that $\tilde{w}^\top z > 0$ if $c = 1$ and $\tilde{w}^\top z < 0$ if $c = -1$. In other words, for every pattern x (z) we are looking to find $\tilde{w}$ and W such that $c\tilde{w}^\top f(Wx) > \xi$, for some $\xi > 0$. For ease of notation, let $y = \tilde{w}^\top z$.

This problem can be formulated in an optimization problem, much like the way SVMs are formulated. To this end, let $g(.)$ be the function that penalizes the misclassification. If $c^{(\mu)}$ is the actual class of pattern μ (either $+1$ or -1), and $y^{(\mu)}$ is the classifier's decision, then $g(c^{(\mu)}y^{(\mu)})$ should be large when $c^{(\mu)}y^{(\mu)} < 0$ and approach zero for $c^{(\mu)}y^{(\mu)} > \xi$, as shown in Figure 2. Several functions can be used for this purpose. However, we will work with g for the moment to derive the optimization problem and hen examine how different choices affect the overall algorithm.

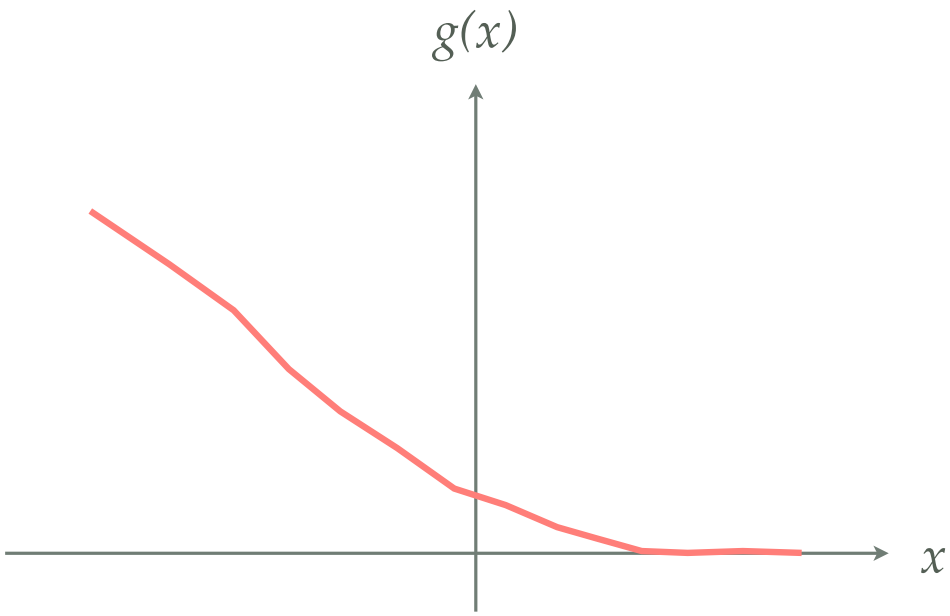

Figure 2: The misclassification penalty function.

With the misclassification penalty function, we can formulate the classification problem as follows

$$\min_{W,\tilde{w}} E = \sum_{\mu} g(c^{(\mu)}y^{(\mu)}) = \sum_{\mu} g(c^{(\mu)}\tilde{w}^\top f(Wx^{(\mu)}) \tag{1}$$

As usual, we take the derivative of the objective function with respect to W_{ij} and $\tilde{w}_i$ to find

[1]The multi-class scenario is a straightforward extension of the two-class case, as is in the linear SVMs.

Algorithm 1 A New Polynomial-Kernel SVM

Input: Maximum iteration number $t_{\max}$
Output: Correct classification of most patterns in the dataset
1: Initialize $\tilde{w}(0)$ and $W(0)$ randomly.
2: **for** $t = 1 \rightarrow t_{\max}$ **do**
3: Find the best $\tilde{w}(t)$ using a standard SVM solver.
4: Update $W(t)$ according to equation (3) with $\tilde{w}(t)$ from the last step and α_t chosen based on the Wolfe Condition.
5: **end for**

the update rule for the weights.

$$\tilde{w}_i(t+1) = \tilde{w}_i(t) - tilde\alpha_t c(t) g'(u(t)) z_i(t), \tag{2}$$

where $u(t) = Wz(t)$, $c(t)$ is the class of the current pattern and $g'(.)$ is the derivative of the function $g(.)$. Likewise, we find

$$W_{ij}(t+1) = W_{ij}(t) - \alpha_t c(t) g'(u(t)) W_{ij}(t) \tilde{w})_i x_j(t)^{W_{ij}-1}, \tag{3}$$

We have tested multiple options for the misclassification penalty function g. A short list is given below

1. Hinge loss: Here, $g(x) = -x + \xi$ for $x < \xi$ and $g(x) = 0$ for $x > \xi$.
2. Exponential: $g(x) = e^{-(\beta x - \theta)}$. We also used $\tanh(e^{-(\beta x - \theta)})$ to stop rapid divergence.
3. Sign: $g(x) = -\max \text{sign}(x - \theta), 0$.
4. **Sigmoid/Logistic**: Here, $g(x) = \frac{1}{1+exp(\beta x - \theta)}$, for some properly chosen values of β and θ.

Each of the above choices has its pros and cons. Some of them converge slowly, some of them may be faster but diverge with a small perturbation. In the end, we ended up using the Logistic function in most cases.

Results: So so: This approach was really sensitive to the choice of $g(.)$. In some cases, it converged but it was difficult to get the desired result.

2.1 the second variant

To overcome the sensitivity issue, we tried a different scheme: in each iteration, first fix W and use an off-the-shelf SVM solver (e.g. the one built in MATLAB) to find the best $\tilde{w}$ given current W. We play with different "box-coefficients" to obtain an SVM that classifies the majority of the training patterns correctly. Then, fix $\tilde{w}$ and update W according to equation (3). Furthermore, we use the Wolfe Condition [1] to find a proper step size α_t in each iteration. The whole process is summarized in Algorithm 1.

Results: Success: Surprisingly, this approach worked like a charm and we got good classification rates on the CIFAR-10 dataset. But even more interesting part of all was that an update to W was not necessary at all in most of the cases. In other words, the SVM training over $\tilde{w}$ with W initialized randomly was enough to achieve acceptable classification rates in the training phase.

This gave us the idea to cascade several such classifiers to get a better success rate. The details are given in the next section.

3 Convolutional Classifiers

So far, we have a classifier that works based on applying a random non-linear mapping to the patterns and then learn the SVM that separates the classes in the mapped space. For CIFAR-10 dataset, preliminary results gives us $60-70\%$ success rate,[2] which is far from the 90% state of the art result.

However, we can cascade several such classifiers, each with a different random mapping matrix W. Since W is chosen independently each time, and since the sub-pattern that lies within the domain of W will be different for each classifier, there is a chance that the decision made by the classifiers are uncorrelated. Thus, if each classifier classifies a given pattern successfully with probability p, then the majority of L such classifiers successfully classifies a given pattern with probability

$$P = \sum_{l=\lfloor L/2 \rfloor + 1}^{L} \binom{L}{l} p^l (1-p)^{L-l}, \tag{4}$$

which can be significantly larger than p. Note that if the classifiers weren't uncorrelated, their decision would be all the same. In practice, the classifiers might not be totally uncorrelated. So the overall success probability should lie somewhere between p and P.

Using this approach, we could easily reach around 90% classification rates with $50-100$ classifiers.

4 Future Work

We should try to apply the proposed approach to other distastes, such as STL-10 or CIFAR-100, to see if we observe the same trend there as well.

Furthermore, we should try to repeat the results by applying the proposed method to the original images (or some light transformations), since in its current form, where we apply the classifier to the output of the sparse filtering, the whole process is rather computationally expensive.

References

[1] P. Wolfe, *Convergence conditions for ascent methods*, SIAM Review, Vol. 11, No. 2, 1969, pp. 226-000.

[2] J. Ngiam, P. W. Koh, Z. Chen, S. Bhaskar,A. Y. Ng, *Sparse filtering*, Proc. Advances in Neural Information Processing Systems (NIPS), Vol. 24, 2011, pp. 1125–1133.

[2] This success rate is achieved by first applying sparse filtering [2] to the images and them using the proposed approach to perform classification.