CS 156a Learning Systems - Homework #5*

Ling Li, ling@cs.caltech.edu

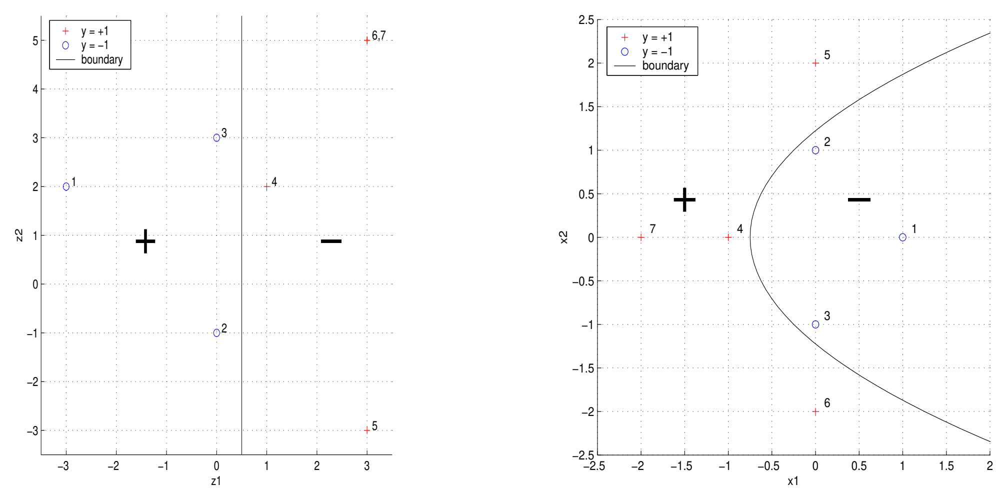
November 30, 2000

5.1 Support Vectors - Geometrically

(i) The training set has been transformed as

$\mathbf{x}$	$(1,0)$	$(0,1)$	$(0,-1)$	$(-1,0)$	$(0,2)$	$(0,-2)$	$(-2,0)$
$\mathbf{z}$	$(-3,2)$	$(0,-1)$	$(0,3)$	$(1,2)$	$(3,-3)$	$(3,5)$	$(3,5)$
y	-1	-1	-1	1	1	1	1

(ii) The boundary in Z-space is $z_1 = 0.5$, shown in Fig. 1(a), with support vectors $\mathbf{z}_2$, $\mathbf{z}_3$, and $\mathbf{z}_4$.



(a) Boundary in Z-space.

(b) Boundary in X-space.

Figure 1: The decision boundaries in different spaces. Points with label 2, 3, 4 are support vectors.

(iii) Transforming the boundary $z_1 = 0.5$ back to the X-space, we get the boundary in X-space is

$$x_2^2 - 2x_1 - 1 = 0.5,$$

which is a parabola (Fig. 1(b)). The support vectors are $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$.

*All the Matlab codes can be found at http://www.cs.caltech.edu/~ling/course/cs156a/hw5/.

5.2 Support Vectors - Analytically

(i) Let $\underline{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_7)$. We want to maximize

$$\mathcal{L}(\underline{\lambda}) = \sum_{n=1}^{7} \lambda_n - \frac{1}{2} \sum_{i=1}^{7} \sum_{j=1}^{7} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to $\sum_{n=1}^{7} \lambda_n y_n = 0$ and $\lambda_n \geq 0$, for $n = 1..7$. Use `quadprog()` in Matlab, we got the optimal $\underline{\lambda}$ is $(0, 0.185185, 1.222222, 0.888889, 0, 0.518519, 0)$.[1]

(ii) Since $\lambda_2, \lambda_3, \lambda_4, \lambda_6 > 0$, $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$ and $\mathbf{x}_6$ must be support vectors. (However, as we will see below, they are not all the support vectors.) Using $\mathbf{x}_2$ as $\mathbf{x}^-$ and $\mathbf{x}_4$ as $\mathbf{x}^+$, we get

$$t = \frac{1}{2} \sum_{\text{SV}} \lambda_n y_n \left(K(\mathbf{x}_n, \mathbf{x}^+) + K(\mathbf{x}_n, \mathbf{x}^-) \right) = 1.666667.$$

Thus the learned function is

$$g(\mathbf{x}) = \text{sgn} \left(\sum_{\text{SV}} \lambda_n y_n K(\mathbf{x}_n, \mathbf{x}) - t \right).$$

We can verify that for all $n = 1..7$, $y_n = g(\mathbf{x}_n)$. Thus those points are linearly separable in the transformed space.

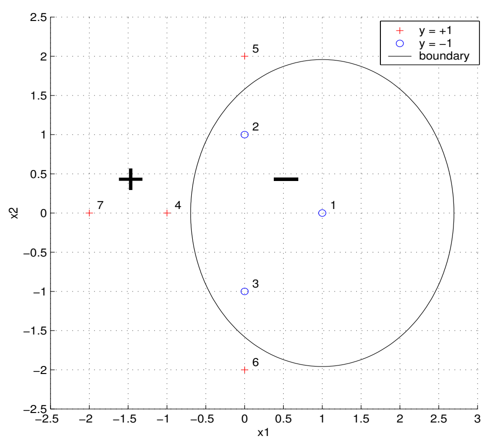


Figure 2: Boundary in the X-space. $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^2$. Points with label 2–6 are support vectors.

We can also calculate the distances of the training points to the boundary in the transformed space, by

$$\left| \sum_{\text{SV}} \lambda_n y_n K(\mathbf{x}_n, \mathbf{x}) - t \right|.$$

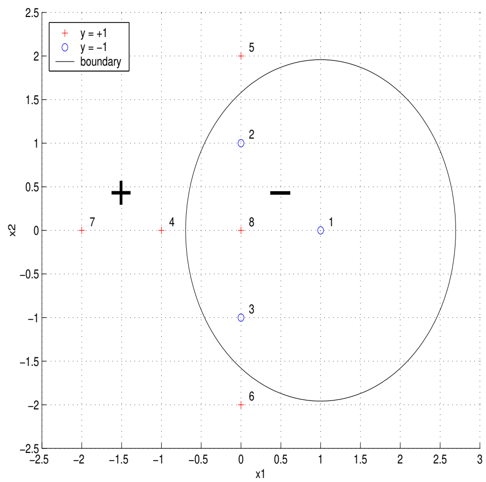
The distances of those 7 points to the boundary are 2.555556, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000, and 5.444444, which means that $\mathbf{x}_5$ is also a support vector, though $\lambda_5 = 0$.

The boundary is shown in Fig. 2. $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$, $\mathbf{x}_5$ and $\mathbf{x}_6$ are support vectors.

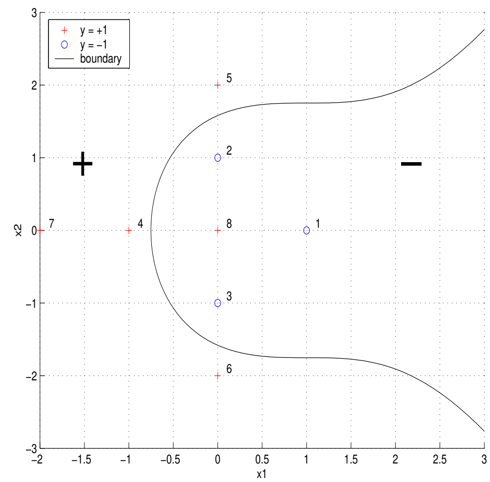
[1] We can infer from these numerical results that $\underline{\lambda} = (0, \frac{5}{27}, \frac{11}{9}, \frac{8}{9}, 0, \frac{14}{27}, 0)$; Also, we can get $t = \frac{5}{3}$.

5.3 More Support Vectors

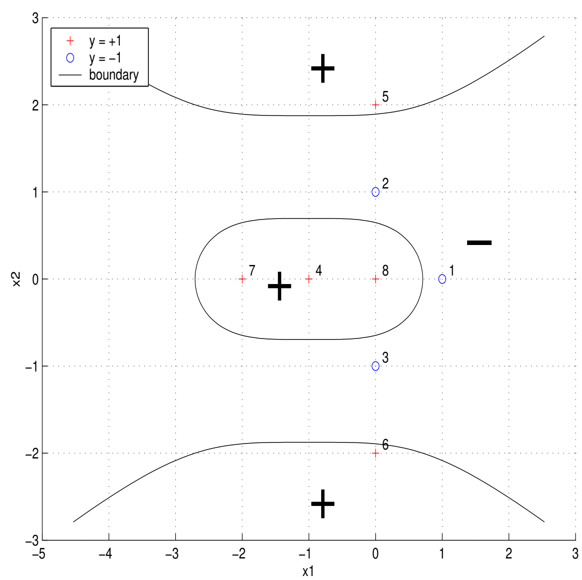
The smallest M that with zero training error is $M = 4$. Here I listed the 'boundaries' for $M = 2, 3, 4$



(a) $M = 2$



(b) $M = 3$



(c) $M = 4$, with zero error

Figure 3: Boundaries in x-space with $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^M$, $M = 2, 3, 4$.

in Fig. 3. However, we may find out that $M = 2, 3$ can not result in a SVM with zero training error. For $M = 4$, similar to problem 5.2(ii), we have $t = -1$ and the support vectors are $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_5$, $\mathbf{x}_6$, $\mathbf{x}_7$, and $\mathbf{x}_8$.

5.4 Jeopardy!

The topic I want to discuss here is the variation of weight decay given in the lecture on Oct. 17, i.e., instead of w^2, the penalty of weight w being large is

$$\frac{w^2}{w^2 + \beta^2}.$$

For simplicity, the same β^l is used for the weights in the same layer l. Thus

$$\hat{E} = E + \lambda \sum_{i,j,l} \frac{(w_{ij}^l)^2}{(w_{ij}^l)^2 + (\beta^l)^2},$$

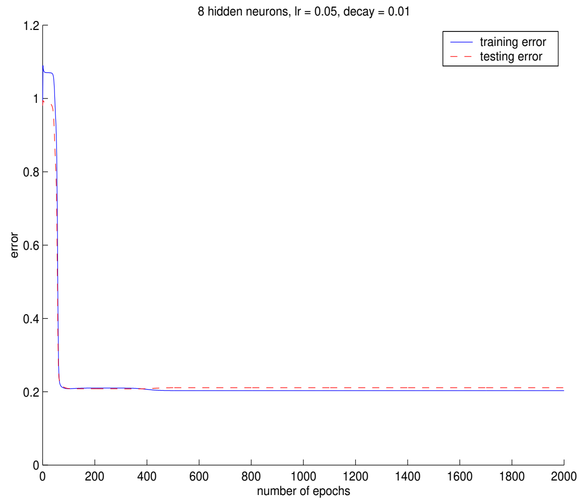
$$\frac{\partial \hat{E}}{\partial w_{ij}^l} = \frac{\partial E}{\partial w_{ij}^l} + 2\lambda \frac{(\beta^l)^2}{\left((w_{ij}^l)^2 + (\beta^l)^2\right)^2} w_{ij}^l,$$

and

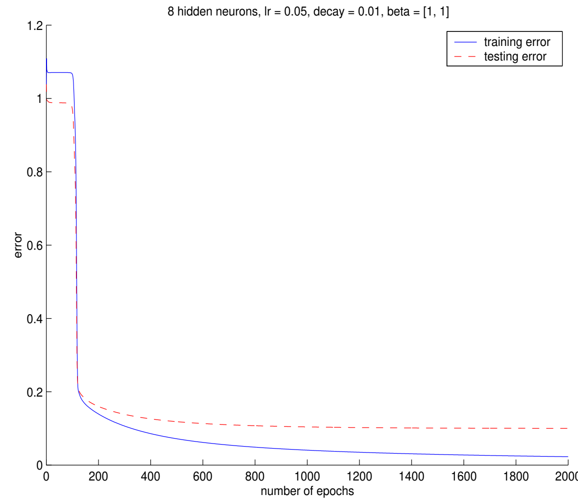
$$w_{ij}^l + \Delta w_{ij}^l = w_{ij}^l - \eta \frac{\partial \hat{E}}{\partial w_{ij}^l} = \left[1 - \frac{2\eta\lambda(\beta^l)^2}{\left((w_{ij}^l)^2 + (\beta^l)^2\right)^2}\right] w_{ij}^l - \eta \frac{\partial E}{\partial w_{ij}^l}.$$

Some simulations were carried out to find the performance of the networks with different combinations of β^l and λ. Specifically, 48 combinations of $\beta^1, \beta^2 \in \{0.1, 0.5, 1, 2\}$ and $\lambda \in \{0.001, 0.01, 0.1\}$ were tried, 3 trials each. The data sets of homework 2.3 were used. Some typical plots of training and testing errors are in Figure 4 on page 5. I listed some discussion in the description of the plots.

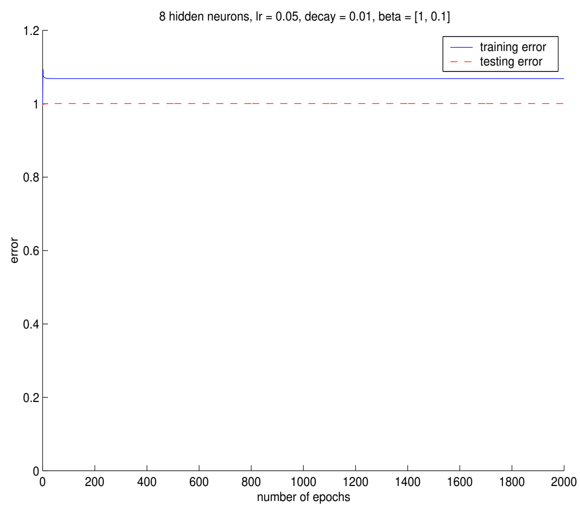
The conclusion is: with the variation of weight decay, we have more control on the trade-off of generalization and the model complexity. However, we should carefully design λ and β to get a better result.



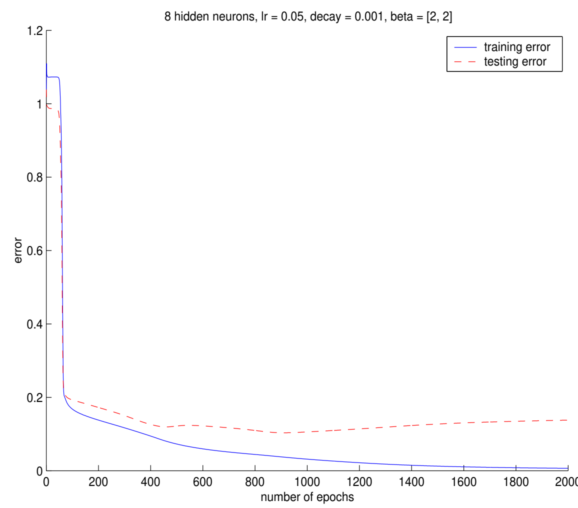
(a) weight decay $\lambda = 0.01$



(b) weight $\lambda = 0.01$, $\beta = \{1, 1\}$



(c) weight decay $\lambda = 0.01$, $\beta = \{1, 0.1\}$



(d) weight decay $\lambda = 0.001$, $\beta = \{2, 2\}$

Figure 4: Some typical plots of training/testing errors. (**a**) Normal weight decay. The generalization is good. However, the weights are confined too much such that some better hypothesis are lost. (**b**) Compared to (a), the network has more freedom to explore the weights space due to reduced penalty, and has smaller training/testing errors. Meanwhile, the generalization is still good. (**c**) With relatively small β (compared to λ), the range of weights with small penalty is shrunk. Thus the model is not capable to learn the training data quite well, resulting in large training and testing errors. This happens with $\beta^1 = 0.1$ or $\beta^2 = 0.1$ when $\lambda = 0.01$, and almost all plots with $\lambda = 0.1$. (**d**) With too small penalty, that is, with small λ and large β, the network performances like the one with no weight decay — over-fitting to the training data.