**CS446: Machine Learning** **Fall 2016**

# Problem Set 1

Shengliang Dai (sdai7) *Handed In: October 21, 2016*

1. **[VC Dimension - 30 points]**

   (a) The VC dimension of the set of triangles is 7.

   There are sets of 7 points on a circle that can be fully shattered using triangles. We use a regular heptagon as example.

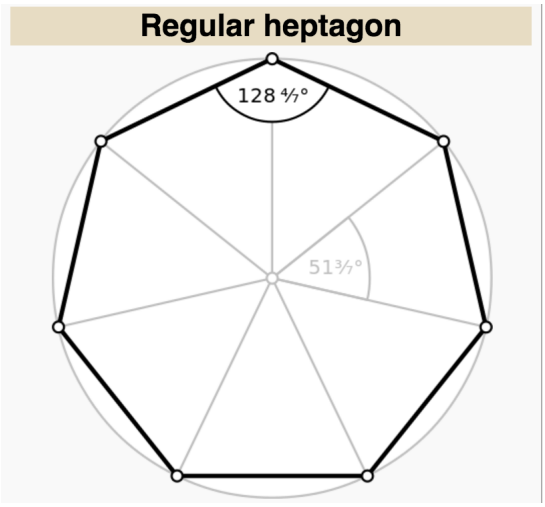


Figure 1: regular heptagon

   Suppose try to label point i as positive and the rest as negative in 8 points 0, 1, ..., 7. If one of them is in the convex hull of the others, it is not possible to label all of them with the sample label. By convexity, a triangle containing the 7 points will also contain the point in the convex hull. If we label the points with -, + -, +, -, + -, + respectively, it is not possible to shatter them with a triangle (if you label it, it is obvious). Therefore, VC <8. VC=7 is feasible.

   (b) We first show the VC-dimension is at least 4 when d = 2. There exists a 4-point set shattered by the concept set, there is no 5-point set that can be shattered.

   **Proof**

   1. An example 4-point set is shown in Fig 2. So we have VC-dim $\geq 4$.

   2. For any 5-point set, we can construct a data assignment in this way: pick the topmost, bottom-most, leftmost and rightmost points and give them the label +. Because there are 5 points, there must be at least one point left to which we assign -. Any rectangle that contains all the + points must contains the - point, which is a case where shattering is not possible. This proves that VC-dim $< 5$. Overall, VC = 4.

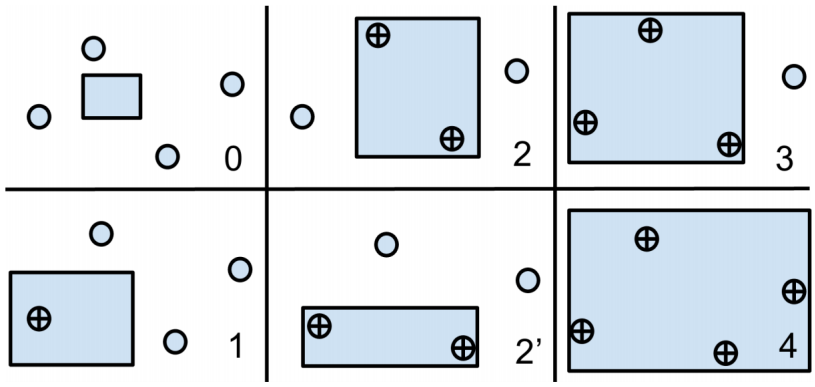


Figure 2: Proving that VC-dim is at least 4 points

Consider a sequence of 2d + 1 distinct points, we label the point in the similar way as d = 2, we cannot shatter it. Therefore the VC dimension is at most 2d. We can shatter it as before, so VC dimension is 2d.

2. **[Kernels - 30 points]**

   (a) **[5 points] Write down the dual representation of the Perceptron algorithm.**

   We predict the label for a new example $\vec{x}$ in M examples using the following equation:

   DualPerceptron(M, $\vec{x}$):

   $$Th_\theta \sum_{(\vec{x_m}, y_m) \in M} y_m \vec{x_m}^T \vec{x_m}$$

   Training:

   $M = \emptyset$

   for $(\vec{x}, y \in S)$:

   — if y ! = DualPerceptron(M, $\vec{x}$):

   — — M = M $\cup(\vec{x}, y)$

   — End if

   End for

   (b) **[5 points] Prove that this is a valid kernel function.** To prove, we need to prove any of the components is valid.

   To show that $(\vec{x}^T \vec{z})^3$ is a valid kernel, we can show that it is a dot product

   $$(\vec{x}^T \vec{z})^3 = x_1^3 z_1^3 + 3x_1^2 z_1^2 x_2 z_2 + 3x_1 z_1 x_2^2 z_2^2 + x_2^3 z_2^3$$

   where $\Phi(\vec{x})$ is defined as $\begin{bmatrix} x_1^3 \\ \sqrt{3}x_1^2 x_2 \\ \sqrt{3}x_1 x_2^2 \\ x_2^3 \end{bmatrix}$

$$(\vec{x}^T\vec{z})^2 = x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2$$

where $\Phi(\vec{x})$ is defined as $\begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$

$\vec{x}^T\vec{z}$ is valid kernel function. Thus it is a valid kernel function.

(c) **[5 points]** when both c(x) and c(z) are true, c(x)c(z) can be 1. This means that c only involves variables that are true for both x and z. Then count the number of satisfying c in monotone conjunction C up to k variables. This takes linear time in n.

(d) **[15 points]** In class, we proved a theorem by Novikoff under the assumption that perceptron function $f(x) = sgn(w * x)$.

Now every example in the original space translates to an example containing $\sum_{j=0}^{k}\binom{n}{j}$ active features in the blown up space.

$$R = \sqrt{\sum_{j=0}^{k}\binom{n}{j} + 1}$$

Since $|u| = 1, y_iux_i \geq \gamma$ for all i, we would like to maximize $\gamma$ to make the bound as tight as possible. In monotone k-DNF, we want the smallest maximum $\gamma$, suppose the blown-up space is $n'$, we are learning the sample with size $r \leq n'$. We assume u maximize $\gamma$ in the form of:

$$u = \frac{u_1 = 1, u_2 = 1, \cdots, u_r = 1, u_{r+1} = 0, \cdots, u_{n'-1} = 0, u_{n'} = \theta}{\sqrt{r + \theta^2}}$$

where $-1 < \theta < 0$.

The last feature $u_{n'}$ must be active, the distance between the positive example $x^+$ and hyperplane is $\frac{1+\theta}{\sqrt{r+\theta^2}}$, for negative example is $\frac{\theta}{\sqrt{r+\theta^2}}$.

$$\gamma = min(\frac{1+\theta}{\sqrt{r+\theta^2}}, \frac{\theta}{\sqrt{r+\theta^2}})$$

$$\theta = -\frac{1}{2}$$

$$\gamma = \frac{1}{\sqrt{4r+1}}$$

The worst case for r is $\sum_{j=0}^{k}\binom{n}{j}$ Thus, $M \leq \frac{R^2}{\gamma^2} = \frac{\sum_{j=0}^{k}\binom{n}{j}+1}{\frac{1}{4\sum_{j=0}^{k}\binom{n}{j}+1}}$

3. **[Neural Networks - 40 points]**

(a) Function1:

This is used to derive the (global) learning rule which performs gradient descent in the weight space in an attempt to minimize the error function.

$$\Delta w_{ij} = -R\frac{\partial E}{\partial w_{ij}}$$

$$Err(x_i, w) = \frac{1}{2}\sum_{k\in K}(t_k - o_k)^2$$

$$\frac{\partial Err(x_i, w)}{\partial o} = -(t_k - o_k)$$

Where $K$ is the set of nodes in the output layer, $t_k$ is the correct value for output node $k$, and $o_k$ is the output of node $k$. Function 2:

Net input to a unit is defined as

$$net_j = \sum w_{ij}x_i$$

Function 3:

The activation function:

$$f(net_j) = max(0, net_j)$$

$$\frac{\partial f(net_j)}{\partial net_j} = \begin{cases} 1 & net_j > 0 \\ 0 & \text{otherwise;} \end{cases}$$

Weight updates of output units:

$w_{ij}$ influences the output only through $net_j$

Therefore,

$$\frac{\partial E_d}{\partial w_{ij}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \begin{cases} -(t_j - o_j)x_{ij} & net_j > 0 \\ 0 & \text{otherwise;} \end{cases}$$

Therefore,

$$\Delta w_{ij} = \begin{cases} R(t_j - o_j)x_{ij} & net_j > 0 \\ 0 & \text{otherwise;} \end{cases}$$

Weights of hidden units:

$w_{ij}$ influences the output only through all the units whose direct input include j.

$$\frac{\partial E_d}{\partial w_{ij}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \sum_{k\in downstream(j)}\frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j}x_{ij} =$$

$$\begin{cases} \sum_{k\in downstream(j)} -(t_k - o_k)w_{jk}x_{ij} & net_j > 0 \\ 0 & \text{otherwise;} \end{cases}$$

Therefore,

$$\Delta w_{ij} = \begin{cases} R\sum_{k\in downstream(j)}(t_k - o_k)w_{jk}x_{ij} & net_j > 0 \\ 0 & \text{otherwise;} \end{cases}$$

(b) code and compile results are in mnist.ipynb, circle.ipynb.

Parameter tuning:

mnist: Choose batch size = 10, learning rate = 0.1, activation function = 'tanh', units in each hidden layer = 10.

```
b: 10, l: 0.1, a: tanh, u: 10
average acc on train: 98.0033382015
average acc on valid: 97.3455759599
b: 10, l: 0.1, a: tanh, u: 50
average acc on train: 97.6486542875
average acc on valid: 97.2120200334
b: 10, l: 0.1, a: relu, u: 10
average acc on train: 97.7800959733
average acc on valid: 96.5692821369
b: 10, l: 0.1, a: relu, u: 50
average acc on train: 96.8892134363
average acc on valid: 95.7011686144
b: 10, l: 0.01, a: tanh, u: 10
average acc on train: 97.3962027957
average acc on valid: 97.1118530885
b: 10, l: 0.01, a: tanh, u: 50
average acc on train: 96.8975589401
average acc on valid: 96.6861435726
b: 10, l: 0.01, a: relu, u: 10
average acc on train: 97.6006676403
average acc on valid: 96.6694490818
b: 10, l: 0.01, a: relu, u: 50
average acc on train: 97.6736907991
average acc on valid: 96.7863105175
b: 50, l: 0.1, a: tanh, u: 10
average acc on train: 97.6152722721
average acc on valid: 97.203672788
b: 50, l: 0.1, a: tanh, u: 50
average acc on train: 97.1646150636
average acc on valid: 97.0116861436
b: 50, l: 0.1, a: relu, u: 10
average acc on train: 97.6006676403
average acc on valid: 96.6444073456
b: 50, l: 0.1, a: relu, u: 50
average acc on train: 97.6507406635
average acc on valid: 96.7028380634
b: 50, l: 0.01, a: tanh, u: 10
average acc on train: 96.9100771959
average acc on valid: 96.7362270451
b: 50, l: 0.01, a: tanh, u: 50
average acc on train: 96.5637387857
average acc on valid: 96.32721202
b: 50, l: 0.01, a: relu, u: 10
average acc on train: 97.5213853536
average acc on valid: 96.8447412354
b: 50, l: 0.01, a: relu, u: 50
average acc on train: 97.558940121
average acc on valid: 96.8697829716
b: 100, l: 0.1, a: tanh, u: 10
average acc on train: 97.4149801794
average acc on valid: 97.0868113523
b: 100, l: 0.1, a: tanh, u: 50
average acc on train: 96.8975589401
average acc on valid: 96.7445742905
b: 100, l: 0.1, a: relu, u: 10
average acc on train: 97.6131858961
average acc on valid: 96.83639399
b: 100, l: 0.1, a: relu, u: 50
average acc on train: 97.6423951596
average acc on valid: 96.8614357262
b: 100, l: 0.01, a: tanh, u: 10
average acc on train: 96.6868349677
average acc on valid: 96.5442404007
b: 100, l: 0.01, a: tanh, u: 50
average acc on train: 96.549134154
average acc on valid: 96.3522537563
b: 100, l: 0.01, a: relu, u: 10
average acc on train: 97.4149801794
average acc on valid: 96.878130217
b: 100, l: 0.01, a: relu, u: 50
average acc on train: 97.4108074275
average acc on valid: 96.878130217
```

Figure 3: Average Accuracy for mnist

circle: Choose batch size = 10, learning rate = 0.1, activation function = 'tanh', units in each hidden layer = 10.

(c) The accuracy of both models on the test data for each of the two data sets.

The plot for MNIST is in Fig. 5.

The plot for Circle is in Fig. 6.

As we can see, neural network performs better in both dataset. NN can achieve a 100 percent accuracy on circle, while perceptron has a very low accuracy (fluctuates around 55 %) on it because circle is not linear separable. The accuracy of circle is low consistently during the 100 iterations, while NN hit 100 percent at 40 iterations.

The performance of perceptron on MNIST data is not bad, but it fluctuates around 90 percent from the learning curve plot. NN performs very well on MNIST from the beginning of the iterations.

```
b: 10, l: 0.1, a: tanh, u: 10
average acc on train: 100.0
average acc on valid: 100.0
b: 10, l: 0.1, a: tanh, u: 50
average acc on train: 100.0
average acc on valid: 100.0
b: 10, l: 0.1, a: relu, u: 10
average acc on train: 100.0
average acc on valid: 100.0
b: 10, l: 0.1, a: relu, u: 50
average acc on train: 100.0
average acc on valid: 100.0
b: 10, l: 0.01, a: tanh, u: 10
average acc on train: 73.84375
average acc on valid: 71.75
b: 10, l: 0.01, a: tanh, u: 50
average acc on train: 83.0
average acc on valid: 82.125
b: 10, l: 0.01, a: relu, u: 10
average acc on train: 100.0
average acc on valid: 100.0
b: 10, l: 0.01, a: relu, u: 50
average acc on train: 100.0
average acc on valid: 100.0
b: 50, l: 0.1, a: tanh, u: 10
average acc on train: 88.625
average acc on valid: 88.0
b: 50, l: 0.1, a: tanh, u: 50
average acc on train: 93.125
average acc on valid: 92.625
b: 50, l: 0.1, a: relu, u: 10
average acc on train: 100.0
average acc on valid: 100.0
b: 50, l: 0.1, a: relu, u: 50
average acc on train: 100.0
average acc on valid: 100.0
b: 50, l: 0.01, a: tanh, u: 10
average acc on train: 54.375
average acc on valid: 50.375
b: 50, l: 0.01, a: tanh, u: 50
average acc on train: 54.28125
average acc on valid: 50.75
b: 50, l: 0.01, a: relu, u: 10
average acc on train: 93.875
average acc on valid: 93.875
b: 50, l: 0.01, a: relu, u: 50
average acc on train: 100.0
average acc on valid: 100.0
b: 100, l: 0.1, a: tanh, u: 10
average acc on train: 77.25
average acc on valid: 75.375
b: 100, l: 0.1, a: tanh, u: 50
average acc on train: 81.375
average acc on valid: 81.375
b: 100, l: 0.1, a: relu, u: 10
average acc on train: 100.0
average acc on valid: 100.0
b: 100, l: 0.1, a: relu, u: 50
average acc on train: 100.0
average acc on valid: 100.0
b: 100, l: 0.01, a: tanh, u: 10
average acc on train: 52.96875
average acc on valid: 49.25
b: 100, l: 0.01, a: tanh, u: 50
average acc on train: 54.4375
average acc on valid: 50.5
b: 100, l: 0.01, a: relu, u: 10
average acc on train: 92.96875
average acc on valid: 92.5
b: 100, l: 0.01, a: relu, u: 50
average acc on train: 95.84375
average acc on valid: 95.5
```

Figure 4: Average Accuracy for circle

| domain | NN | NN learning curve | Perceptron | Perceptron learning curve |
|---|---|---|---|---|
| mnist | 96.72 % | 96.77% | 88.86% | 88.86% |
| circles | 100% | 100% | 49.5% | 49.5% |

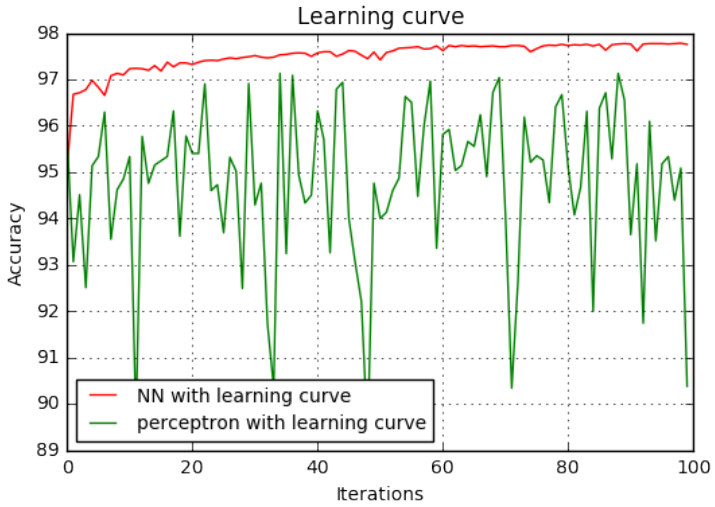


Figure 5: Learning Curve for MNIST

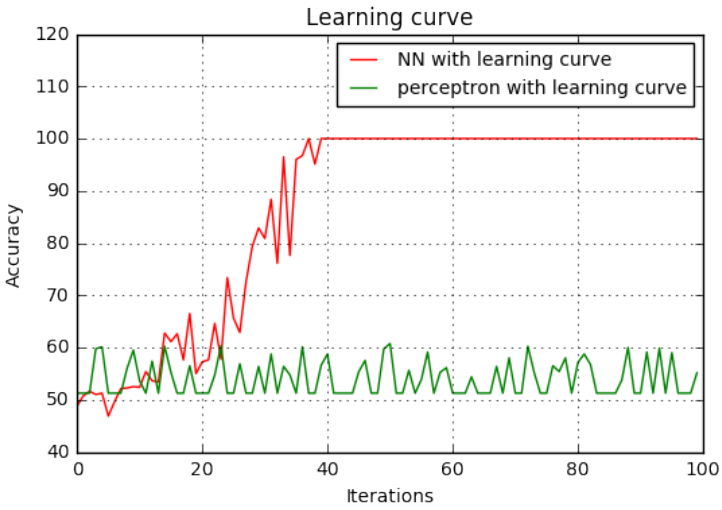


Figure 6: Learning Curve for circle