sbd291

(1)

1. **(10 points)** In class, we discussed how to represent XOR-like functions using quadratic features, since standard linear classifiers (such as perceptrons) are insufficient for this task. However, here we show that XOR-like functions can indeed be simulated using *multi-layer* networks of perceptrons. This example shows a glimpse of the expressive power of "deep neural networks": merely increasing the depth from 1 to 2 layers can help reproduce nonlinear decision boundaries.

   a. Consider a standard two-variable XOR function, where we have 2-dimensional inputs $x_1, x_2 = \pm 1$, and output $y = x_1(XOR)x_2 = \begin{cases} -1 & \text{if } x_1 = x_2 \\ 1 & \text{otherwise} \end{cases}$.

   Geometrically argue why a single perceptron cannot be used to simulate the above function.

   b. Graphically depict, and write down the equation for, the optimal decision region for the following logical functions:
      (i) $x_1(AND)(NOT(x_2))$
      (ii) $(NOT(x_1))(AND)x_2$
      (iii) $x_1(OR)x_2$ Make note of the weights learned corresponding to the optimal decision boundary for each function.

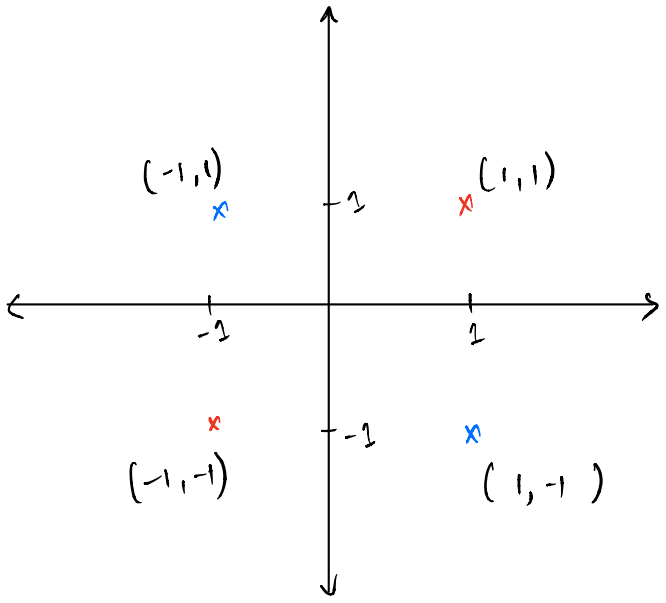
   c. Using the above information, simulate a multi-layer perceptron *network* for the XOR operation with the learned weights from Part (b).

(a)

x_1	x_2	XOR	Perceptron equation		
-1	-1	-1	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$	$\rightarrow$	$w_0 - w_1 - w_2 < 0$
-1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$	$\rightarrow$	$w_0 - w_1 + w_2 \geq 0$
1	-1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$	$\rightarrow$	$w_0 + w_1 - w_2 \geq 0$
1	1	-1	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$	$\rightarrow$	$w_0 + w_1 + w_2 < 0$

Obtained by substituting the input values, x_1 & x_2

By looking at last column that contains equations that can be solved to obtain the weights, we can see some clear contradiction in the values of weights. This means that there are no definite weights that can be learnt for solving the XOR function.



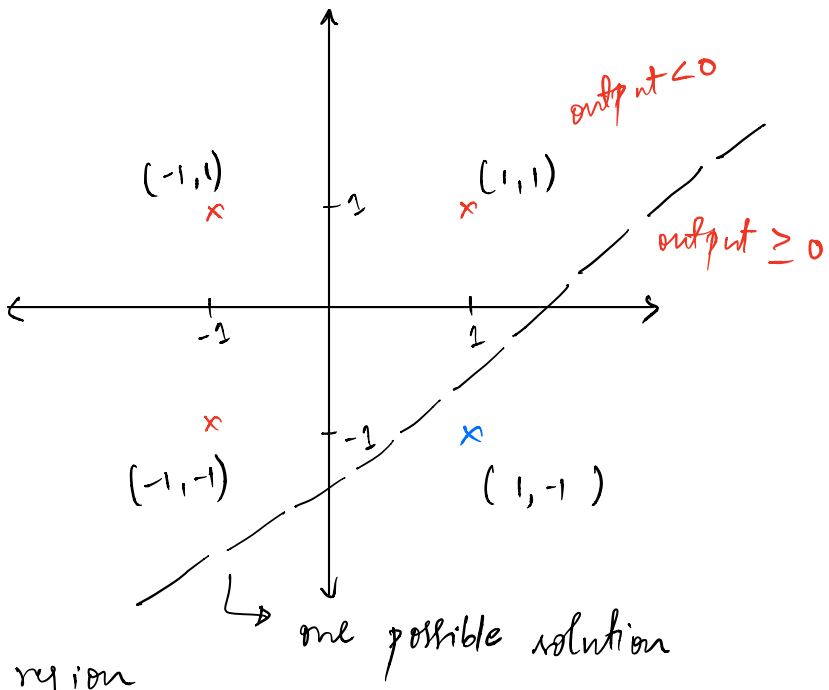
Geometrically, it is evident that no solution exists that can classify these 4 points according to their labels → {-1, 1}

X → XOR output 1

X → XOR output -1

(b) (i) $y = x_1$ (AND) NOT (x_2)

x_1	x_2	y	
-1	-1	-1	X
-1	1	-1	X
1	-1	1	X
1	1	-1	X



For label=1: Equation for decision region

↳ $w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

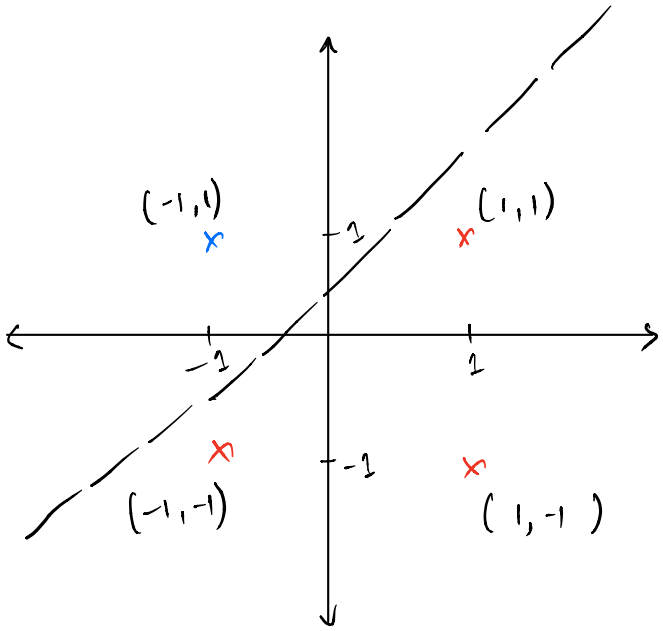
For label=-1: $w_0 + \sum_{i=1}^{2} w_i x_i < 0$

$w_0 = -1.5$, $w_1 = 1$, $w_2 = -1$

Eq of perceptron → $-1.5 + 1 - 1 = 0$

(ii) $y = \text{NOT}(x_1)$ AND x_2

x_1	x_2	y	
-1	-1	-1	x
-1	1	1	x
1	-1	-1	x
1	1	-1	x



For label = 1 : $w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

For label = -1 : $w_0 + \sum_{i=1}^{2} w_i x_i < 0$

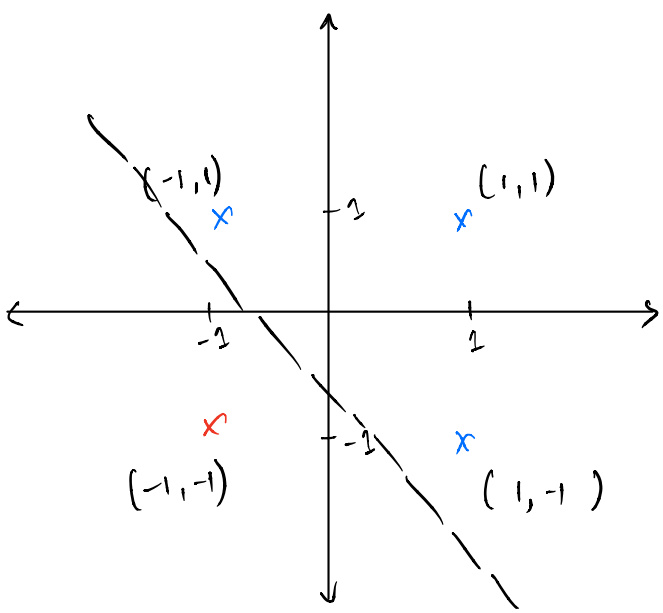
$w_0 = -0.5$, $w_1 = -1$, $w_2 = 1$

Eq of perceptron is :

$-0.5 - x + y = 0$

(iii) $y = x_1$ OR x_2

x_1	x_2	y	
-1	-1	-1	x
-1	1	1	x
1	-1	1	x
1	1	1	x

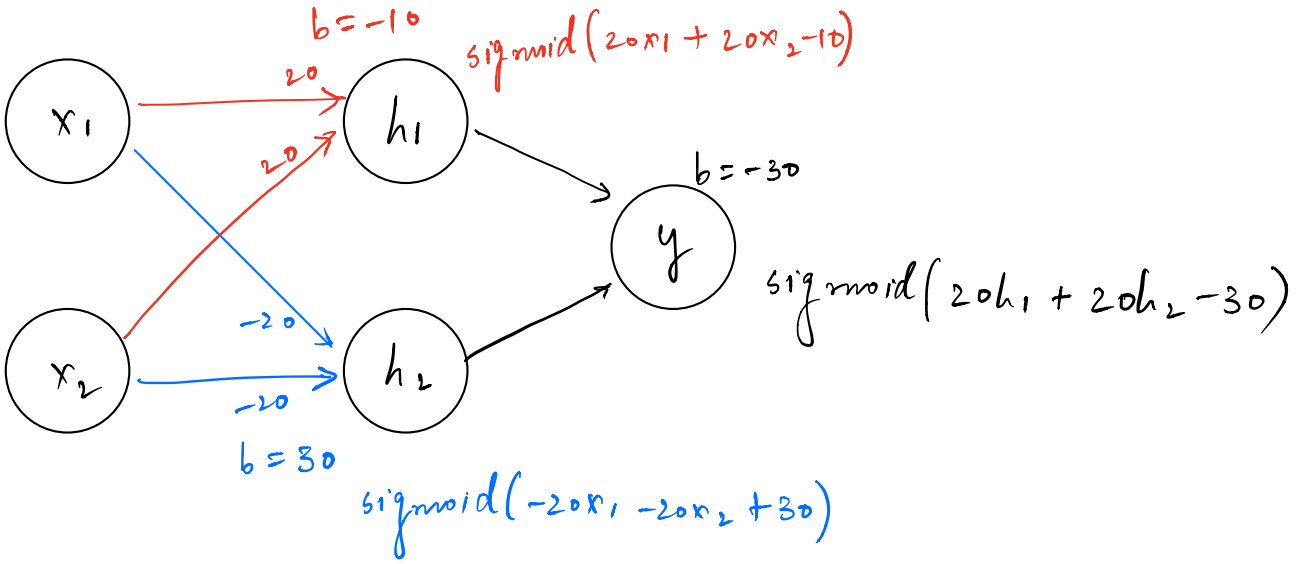


For label = 1 : $w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

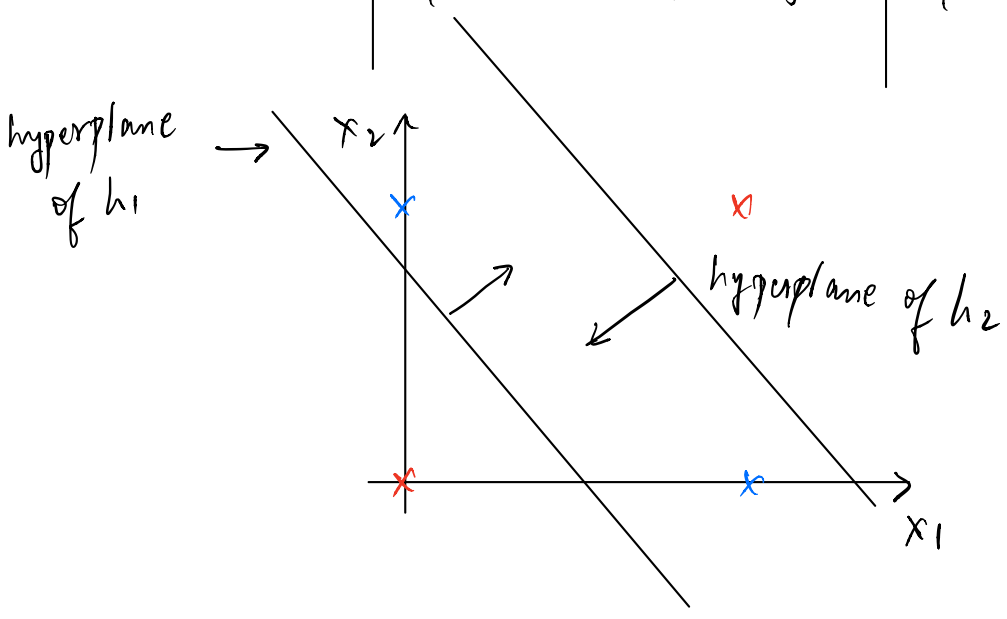
For label = -1 : $w_0 + \sum_{i=1}^{2} w_i x_i < 0$

$w_0 = 0.5$, $w_1 = 1$, $w_2 = 1$, Eq of perceptron $= 0.5 + x_1 + x_2 = 0$

c) To solve the XOR function consider the following multi layer NN, initialized with some weights and biases.



Outputs of h_1	Outputs of h_2	output at y
$\sigma[(20)(0) + (20)(0) - 10] = 0$	$\sigma[(-20)(0) - (20)(0) + 30] = 1$	$\sigma[(20)(0) + (20)(1) - 30] = 0$
$\sigma[(20)(1) + (20)(1) - 10] = 1$	$\sigma[(-20)(1) - (20)(1) + 30] = 0$	$\sigma[(20)(1) + (20)(0) - 30] = 0$
$\sigma[(20)(0) + (20)(1) - 10] = 1$	$\sigma[(-20)(0) - (20)(1) + 30] = 1$	$\sigma[(20)(1) + (20)(1) - 30] = 1$
$\sigma[(20)(1) + (20)(0) - 10] = 1$	$\sigma[(-20)(1) - (20)(0) + 30] = 1$	$\sigma[(20)(1) + (20)(1) - 30] = 1$



(3)

3. **(10 points)** In Lectures 2 and 5, we derived a closed form expression for solving linear and ridge regression problems. This is great for finding linear behavior in data; however, if the data is nonlinear, just as in the classification case, we have to resort to the *kernel trick*, i.e., replace all dot products in the data space with kernel inner products. In this problem, we theoretically derive kernel regression. Suppose we are given training data $\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$ where each response y_i is a scalar, and each data point x_i is a vector in d dimensions.

   a. Assume that all data have been mapped into a higher dimensional space using the feature mapping $x \mapsto \phi(x)$, write down an expression for the squared error function using a linear predictor w in the high-dimensional space.

   b. Let Φ be the matrix with n rows, where row i consists of the feature mapping $\phi(x_i)$. Write down a closed form expression for the optimal linear predictor w as a function of Φ and y.

   c. For a new query data point z, the predicted value is given by $f(z) = \langle w, \phi(z) \rangle$. Plug in the closed form expression for w from the previous sub-problem to get an expression for $f(z)$.

   d. Suppose you are given black-box access to a kernel dot product function K where $K(x, x') = \langle \phi(x), \phi(x') \rangle$. Mathematically show that all the calculations in (b) and (c) can be performed by invoking the kernel dot product alone *without explicitly writing down* $\phi(x)$ *ever*. You may want to use the Sherman-Morrison-Woodbury identity for matrices:
   $$(A^{-1} + B^T C^{-1} B)^{-1} B^T C^{-1} = A B^T (B A B^T + C)^{-1}.$$

3a) $L(w) = \min \sum_{i=1}^{n} (w^T x_i - y_i)^2$ - Loss function without Kernelization

x is mapped to $\phi(x)$ $\therefore$ $L(w) = \min \sum_{i=1}^{n} (w^T \phi(x_i) - y_i)^2$

The dimensions of this are very high. So, its hard to compute.

3b) closed form expression for linear regression

$$w = (X X^T)^{-1} X y$$

When expanded to higher dimensions using, $x \to \phi(x)$

$$w = (\phi \phi^T)^{-1} \phi y$$

By expressing 'w' as a linear combination of inputs of the training space, we get

$$w = \sum_{i=1}^{n} \alpha_i x_i = X\vec{\alpha}$$

$x \to \phi(x)$ $\therefore$ X is replaced by ϕ

$$\phi\vec{\alpha} = w = (\phi\phi^T)^{-1}\phi y$$

Multiplying on both sides by $\phi^T\phi\phi^T$

$$(\phi^T\phi)(\phi^T\phi)\vec{\alpha} = \phi^T(\phi\phi^T(\phi\phi^T)^{-1})\phi y$$

Replace $\phi^T\phi$ by K

$$K^2\vec{\alpha} = Ky$$

$$\boxed{\vec{\alpha} = K^{-1}y}$$

Kernel regression can be extended to Kernelized Ridge regression where, $\boxed{\vec{\alpha} = (K + \lambda I)^{-1}y}$

$\therefore$ The closed form expression for Kernelized Ridge regression is given by,

$w = \phi\vec{\alpha}$, where $\vec{\alpha} = (K + \lambda I)^{-1}y$ & $K = \phi\phi^T$

$$\therefore \boxed{w = (\phi\phi^T + \lambda I)^{-1}\phi y}$$

3c) For a test point z, $f(z) = \langle w, \phi(z) \rangle$

$$f(z) = \left\langle \left(\phi\phi^T + \lambda I\right)^{-1} \phi y, \phi(z) \right\rangle$$

$$\boxed{f(z) = \phi(z)\phi\left(\phi\phi^T + \lambda I\right)^{-1} y}$$

3d) The answer obtained in '3c' is not an efficient way of computing the prediction for the label of the test point. Computing ϕ, $\phi(z)$ and the product of ϕ & $\phi(z)$ is hard to compute. But we actually don't need to compute any of these things including $\phi(x)$.

The inner product $\phi(x)\,\phi(z)$ can be efficiently computed by,

$$\phi(x)\,\phi(z) = 1.1 + x_1 z_1 + x_2 z_2 + \ldots\ldots + x_1 x_2 z_1 z_2 + \ldots\ldots z_d$$

$$= \prod_{K=1}^{d} (1 + x_K z_K)$$

The sum of 2^d terms is converted to a product of d terms. Running time is reduced from $O(2^d)$ to $O(d)$.

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

↳ Kernel function

For n training samples, all possible inner products can be precomputed and stored in a matrix K. With this Kernel dot product function 'K', we can now perform a look-up to get any K_{ij} value.

Here, $K_{ij} = \phi(x_i)^T \phi(x_j)$

While training in the $\phi(x)$ space, to perform G.D we need to compute $\langle w^T . \phi(x) \rangle$. Because 'w' can be expressed as a linear combination of the training samples, $w = \sum_{j=1}^{n} \alpha_j \phi(x_j)$.

G.D $\rightarrow$ $w_{t+1} = w_t - (lr)\left(\frac{\partial l}{\partial w}\right)$

where, $\frac{\partial l}{\partial w} = 2\left(w^T \phi(x_i) - y_i\right) x_i$

$\frac{\partial l}{\partial w}$ becomes $2\left(\sum_{j=1}^{n} \alpha_j K_{ij} - y_i\right)$, so the gradient update becomes

$$\alpha_i^{t+1} = \alpha_i^t - 2(lr)\left[\sum_{i=1}^{n} \alpha_j^t K_{ij} - y_i\right]$$

So, in conclusion we never calculate $\phi(x_i)$ we just calculate the dot product of $\phi(x_i)$ with $\phi(x_j)$ in $O(d)$ time.