CS 475 Machine Learning: Homework 4
EM and Clustering
Due: Thursday Nov 3, 2016, 11:59pm

Jin Yong Shin
Jshin44

1. Semi-supervised EM algorithm (10 points) We consider a d-dimensional mixture model with the following probability density function

$$f(x;\theta_1,...,\theta_k)=\sum_{i=1}^{K}\pi_i f_i(x;\theta_i),$$

where $f_i(x;\theta_i)$ is a probability density function. Suppose that we observe $n+m$ samples independently generated from the above mixture model, and m of the observed samples are labelled. Specifically, we have $x_1,...,x_n\in\mathbb{R}^d$ and $x_{n+1},...,x_{n+m}\in\mathbb{R}^d$. Meanwhile, we know the labels corresponding to $x_{n+1},...,x_{n+m}$, i.e., $y_{n+1},...,y_{n+m}\in\{1,...,K\}$. Design an EM algorithm to cluster the data.

[Hint: For $x_{n+1},...,x_{n+m}$, the corresponding labels are no longer missing values]

(a) Write the likelihood objective for this model.

(b) Write the update rules in each iteration.

ANSWER: For the semi-supervised learning algorithm, we can use labeled algorithm to learn unlabeled algorithm. So in here, when we use semi-supervised EM algorithm, we can compute initial parameters from probabilistic model and then use them to run EM algorithm to compute expected labels and get maximum likelihood.
Since we can use an EM algorithm, I will use Multivariate Gaussian Distribution to implement this new algorithm

(a) Write the new likelihood objective for this new algorithm.
Now let's recall our multivatriate gaussiance distribution

$$N(x|\mu,\sigma)=\frac{1}{(2\pi|\sigma|)^{\frac{1}{2}}}exp\{-\frac{1}{2}(x-\mu)^T\sigma^{-1}(x-\mu)\}$$

Then loglikelihood for above multivariate gaussian distribution is

$$log(N(x|\mu,\sigma))=L(\mu,\sigma)=\sigma_{i=1}^{n}\ln N(x_i|\mu,\sigma)$$

$$=\sigma_{i=1}^{n}(-\frac{1}{2}(x_i-\mu)^T\sigma^{-1}(x_i-\mu)-\pi\ln|\sigma|)$$

I will apply this to basic GMM model $p(x)$,

$$p(x)=\sum_{k=1}^{K}\pi_k N(x|\mu_k,\sigma_k)$$

where $\sum_{k=1}^{K} \pi_k = 1$. Also we can estimate the following parameters where $0 \leqq \pi_k \leqq 1$. π_k, μ_k, σ_k. Then we can find the loglikelihood of the unlabeled (unsupervised) values as following:

$$\ln p(x|\pi, \mu, \sigma) = \sum_{i=1}^{n} \ln\{\sum_{k=1}^{K} \pi_k N(x_i|\mu_k, \sigma_k)\}$$

When we combine the labeled and unlabeled loglikelihood to build our model, we will have

$$\ln p(x|\pi, \mu, \sigma) = \sum_{i=1}^{n} \ln(\sum_{k=1}^{K} \pi_k N(x_i|\mu_k, \sigma_k)) + \sum_{i=n+1}^{n+m} \ln(N(x_i|\mu_{y_i}, \sigma_{y_i})$$

where $y_{n+1}, ..., y_{n+m} \in \{1, ..., K\}$.
As we can see, the first part of the equation is from unlabeled data and second term of the equation is from labeled.

(b) Write the new update rules in each iteration.
For this part of the question, we will have three variables need to be updated. Therefore, we will have three update rules overall.
We will first initialize the following three variables that we are going to use in GMM model μ_k, σ_k, π_k
Then we will begin our EM algorithm with E-Step first:

$$r_{ik} = p(z_{ik} = 1|x_i, \pi, \mu, \sigma)$$

$$= \frac{p(z_{ik} = 1)p(x_i|z_{ik} = 1, \pi, \mu, \sigma)}{\sum_{k=1}^{K} p(z_{ik} = 1)p(x_i|z_{ik} = 1, \pi, \mu, \sigma)}$$

$$= \frac{\pi_k N(x_i|u_k, \sigma_k)}{\sum_{k=1}^{K} \pi_k N(x_i|u_k, \sigma_k)}$$

Where $\sum_{k=1}^{k} r_{ik} = 1$ for all i.
For now, we will update following parameters in M-Step : μ_k, σ_k, π_k using r_{ik} that we have defined (or found) in above.

(1) π_k

$$E(\ln p(x, z|\pi, \mu, \sigma)) = \sum_{i=1}^{n} \sum_{k=1}^{K} r_{ik}\{\ln \pi_k + \ln N(x_i|\mu_k, \sigma_k)\}$$

π_k, π_k will remain unchanged. Therefore:

$$\pi_k = \frac{\sum_i r_{ik}}{n}$$

(2) σ_k
We will find derivative first in for σ.

$$\frac{\partial L}{\partial \sigma} = -\frac{1}{2}(\frac{\partial \ln(|\sigma|)}{\partial \sigma}) - \frac{1}{2}(\frac{\partial (x-\mu)^T \sigma^{-1}(x-\mu)}{\partial \sigma})$$

$$= -\frac{1}{2}(\sigma^{-1} - \sigma^{-1}(x-\mu)(x-\mu)^T \sigma^{-1})$$

To find the value, we will recall the value in (1) r_{ik} from above.

$$\sigma_k = \frac{\sum_{i=1}^{n} unlabeled_{ik}(x_i - \mu_k)(x_i - \mu_k)^T + \sum_{i=n+1}^{n+m} labeled_i(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{n} unlabeled_{ik} + \sum_{i=n+1}^{n+m} labeled_i}$$

notation for the following parameters are : $labeled_i$ is for labeled data in kth cluster and $unlabeled_i$ is for unlabeled data.

(3) μ_k
Here, we will first compute loglikelihood for μ.

$$\frac{\partial L}{\partial \mu} = -\frac{1}{2}(\frac{\partial (x-\mu)^T \sigma^{-1}(x-\mu)}{\partial \mu})$$

$$= \sigma^{-1}(x - \mu)$$

Then use (1) r_{ik} and substitute

$$\mu_k = \frac{\sum_{i=1}^{n} u_{ik}x_i + \sum_{i=n+1}^{n+m} labeled_i x_i}{\sum_{i=1}^{n} unlabeled_{ik} + \sum_{i=n+1}^{n+m} labeled_i}$$

Theses three are the update rules fall in EM algorithm with MGD

2) Deep Neural Networks (15 points)

(a) Consider a 2-layer neural network, with M input nodes, Z nodes in the hidden layer and K nodes in the output layer. The network is fully connected, i.e. every node in the $n-1$th layer is connected to every node in the nth layer. However, for your application of interest, you suspect that only some of the nodes in the input are relevant. How would you modify the objective function to reflect this belief?

(b) Consider a N layer neural network. We could (a) train the entire network at once using back-propagation or (b) pre-train each layer individually, and then tune the final network with back-propagation. Will (a) and (b) converge to the same solution? Why would we favor strategy (a) vs. strategy (b)?

(c) Consider a $N \geq 2$ layer neural network with a single node in the output layer. We wish to train this network for binary classification. Rather than use a cross entropy objective, we want to take a max-margin approach and ensure a margin of $\gamma = 1$. Describe the structure of the last layer of the network, including the final activation function, and the training objective function that implements a max-margin neural network. What are the benefits of this network compared to one trained with cross entropy? Will a max-margin trained neural network learn the same decision boundary as an SVM?

ANSWER:

(a) We are given the neural networks with a fully connected layers. And we are suspecting that some of the nodes are not relevant and only some of the nodes in the input are relevant. This means that we need to reduce the objective function to disregard the unnecessary node in making our prediction. We could simply introduce regularization by making weight of the not relevant node's weight to 0. Because we know that the network is fully connected, every node has weight from one to another. So I would like to introduce techniques that we could reduce and minimize our neural network by making certain weights to 0. (Objective function in neural networks are either regression or cross entropy error function). We could provide large regularization term to our objective function. Or we could create a objective function that find a Neural network solution that reduces the sum of squared errors as well as the number of non-zero weights. Additionally, by adding this regularization term, we can now have minimized version of our objective function where we can disregard irrelevant input nodes by giving them 0 weights. Or we could start with the weights near zero at the beginning and use gradient descent. The more gradient we have done, more function we can reach. So we can modify our objective function in a way that we can depend on the number of steps of gradient descent not weights. $J_i(W) = \frac{1}{2}(\hat{y}_i - y_i)^2$. Then we can take as many chain rule as possible to find necessary gradient. By adding regularization

(b) Let's say method (i) is training whole network with back-propagation and (ii) is training with pre-train and back-propagation.
If we use method (i), we are using Stochastic Gradient Descent and chain-rule multiplication at each layer to find the most appropriate movement. There are two majors concern with this method. First is that we might have to compute N many updates or even worse at each iteration because we are using small steps and thus gradient step are decreasing. Second problem is that this also can be from first problem but since we are having lots of layers and don't know which one have maximum. Meaning that the layers are solely

depended on previous layer thus can be stopped at local minimum, which cause Vanishing Gradient problem.
(ii) method can be used to avoid Vanishing Gradient problem. By using pre-train method, we can train unlabeled data by using labeled data. By working bottom-up sense in the layers, we can also use labeled data to perform supervised fine tuning so that we don't have to worry about Vanishing Gradient Problem and slowing down.
Thus, we can now see that both (i) and (ii) method do not necessary converge since (i) method has some problem called Vanishing Gradient and cannot exit local optimum if it misunderstood it as global optimum. For these reasons, method (ii) is more favorable since it covers the odd of first method.

(c) If we use SVM and binary classification to build neural network, there are several pros that we can obtain from them. As the problem mentioned, the biggest advantage is that SVM tend to perform well based on maximum margin. If we use max margin principle, we can derive objective function that are similar to logically SVM which use hinge loss for the objective function.

$$L(w) = C\sum_{i=1}^{n} max(1 - w^T * \theta_n * y_n, 0) + \lambda\frac{1}{n}||w||^2$$

where z_n indicates the activation value at layer n which locates at the end. This model will certainly ensure our max margin = 1.
Now take derivative of L(w) respect to θ_n to find the loss function of last layer and then find rest by using back-propagation.

$$\frac{dL(w)}{d\theta_n} = -C * y_n * w(cond(w^T * \theta_n * y_n < 1))$$

where

$$cond(x) = \begin{cases} 1 & \text{if } x - true \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

.
Our main object is to classify the neural network like binary classification. Therefore, we will bring our linear regression or sigmoid function to return label 0 or 1.
Benefits that we can get from using SVM and implement max-margin principle at the last layer will be 1) reducing overfitting issues 2) better behavior with outliers. Therefore, rather than using original entropy based objective function, neural network can achieve better performance by using max-margin since max margin is originally focuses on loss minimization. Using the back-propagation, we could also find better lower layer. This is why we are using max margin neural nets and we can find discriminative representation for each nodes by using max margin.
Finally, we can conclude that max margin principle oriented neural network training and SVM will not necessarily learn decision boundary. This is because they have different set of input variables passed through layers and functions. Therefore, at the last stage, it can be differed.

3) Neural Networks (15 points) Suppose we have two inputs: x_1 and x_2. Both x_1 and x_2 are real numbers and their values are restricted such that $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. You may use activation functions of the form:

$$\theta_r(z) = \begin{cases} 1 & \text{if } z \leq r \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where r determines the activation function. You may select a different r for each node in an artificial neural network. Create artificial neural networks for the following functions.

(a) Create a multi-layer network to recognize when $x_2 \geq \max(x_1, 1 - x_1)$.

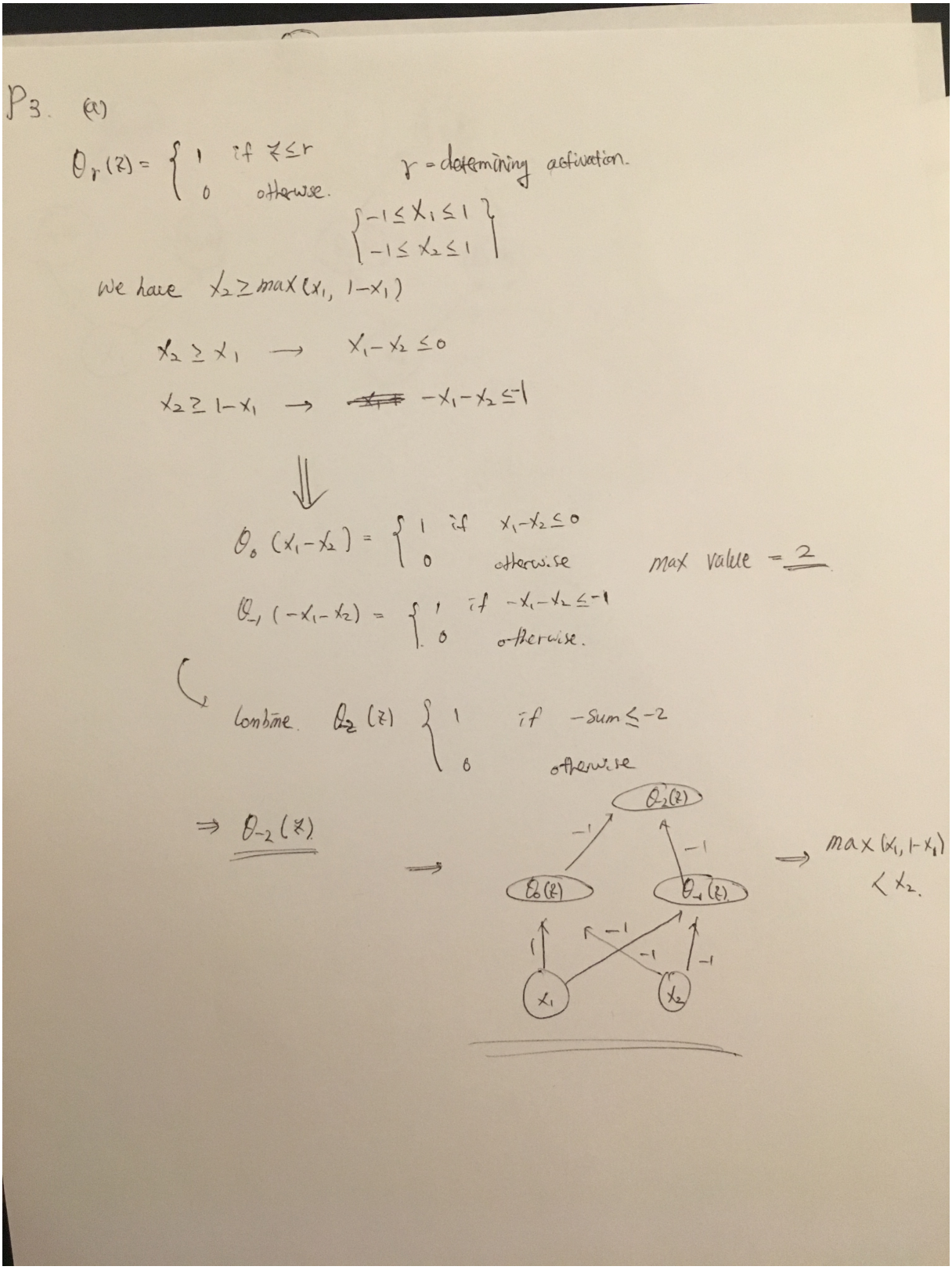
(b) Create a multi-layer network to recognize when $|x_2| + |x_1| \leq 1$.

(c) Suppose we wanted to build a multi-layer network that approximates (with some error) the decision $x_1^2 + x_2^2 \leq 1$. Explain such an approximation.

For each network, describe the network structure, the value of each weight, and the activation function for each node. You may do this in words, or by drawing a picture. Please keep your answers clear and concise, i.e. we do not need a long explanation of how the network works; we only need the network definition.

ANSWER :

(a) Constraint in this model is that r value cannot go over 2. Since both max indicates the limits and if we combine the limit, maximum value will be -2



(b)

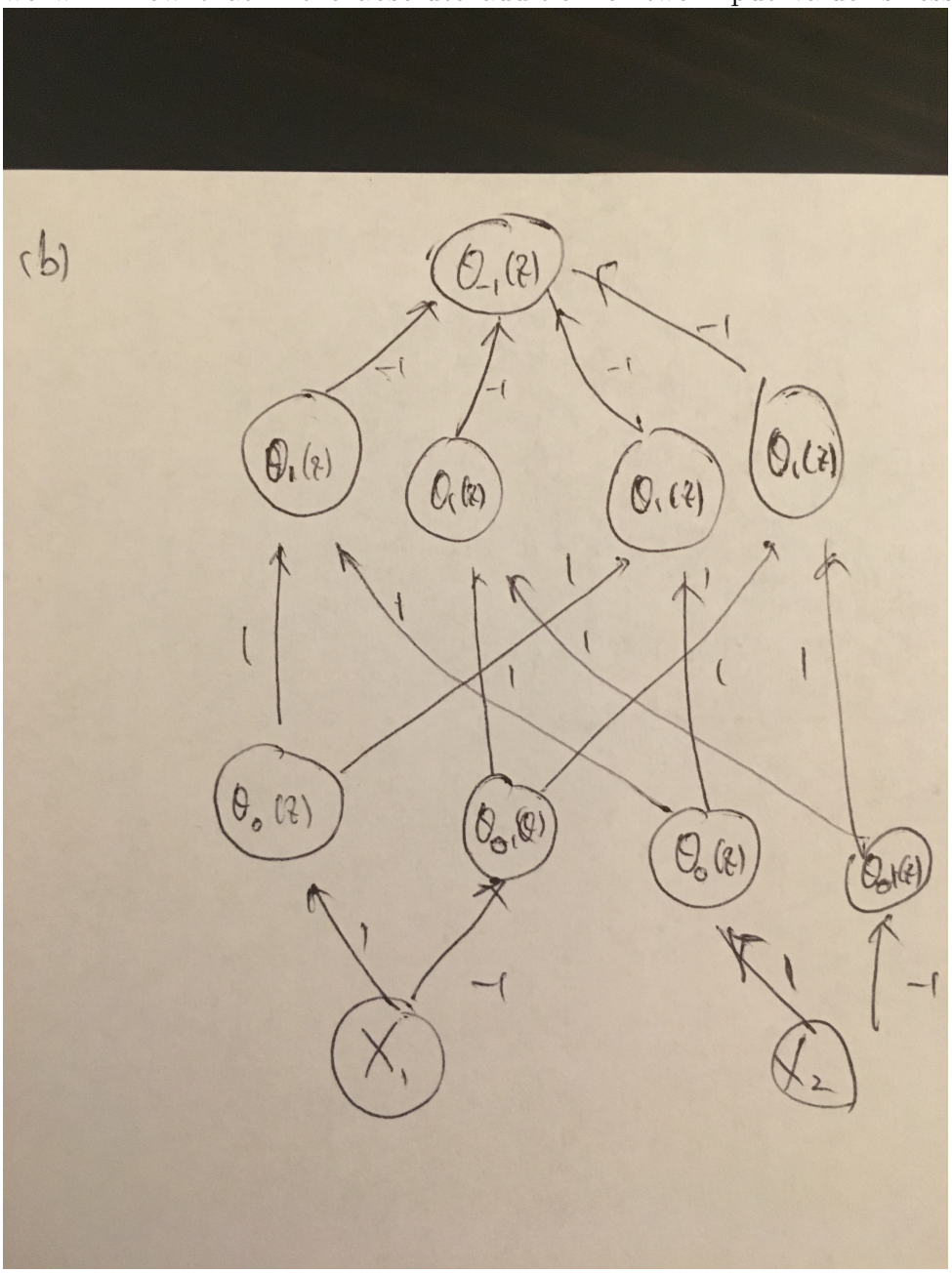
There is four cases where we have to consider:

i) For $x_2 \leqslant 0$ and $x_1 \leqslant 0$, check if $-x_1 - x_2 \leqslant 1$

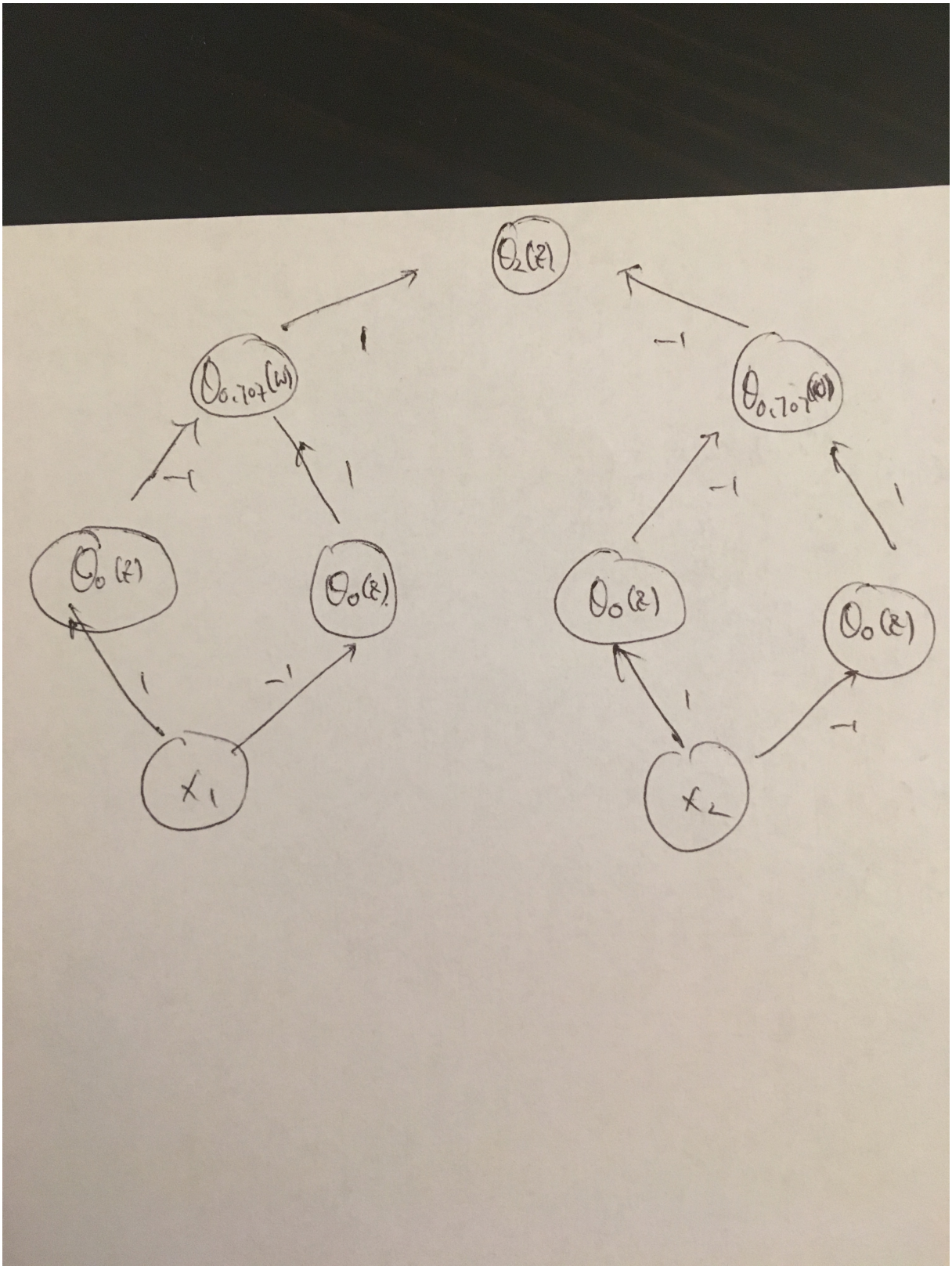
i) For $x_2 \leqslant 0$ and $x_1 \geqslant 0$, check if $x_1 - x_2 \leqslant 1$

i) For $x_2 \geqslant 0$ and $x_1 \leqslant 0$, check if $-x_1 + x_2 \leqslant 1$

i) For $x_2 \geqslant 0$ and $x_1 \geqslant 0$, check if $x_1 + x_2 \leqslant 1$
By giving weight 1, -1, 1, -1, and setting 0, we know that if input passes the layer (aka satisfy θ condition), then we can know that this inputs have negative or positive. Then additionally, if we use second layer to check aforementioned conditions, we will know that if the absolute addition of two input value is less thank 1 or not.



(c)
For this question, we actually need to compute the equation first.
we have $x_1^2 + x_2^2 \leqslant 1$. Since $-1 \leqslant x_1, x_2 \leqslant 1$ and it's squared, we only need to care about the magnitude. We can put the restriction on each x_1 and x_2 such that $|x_1| \leqslant \sqrt{0.5}$ as well as $|x_2| \leqslant \sqrt{0.5}$ Thus the equality at the problem is satisfied under our assumption. We know that approximately $x_1, x_2 = 0.707$. Therefore, we can build artificial network with this number to find or recognize the condition. Also we need to make the value positive, always negate so that $x_1 \leqslant 0.0707$ to determine the sign of this input. as


$\theta_2(z)$
$\theta_{0.707}(w)$
$\theta_{0.707}(w)$
$\theta_0(z)$
$\theta_0(z)$
$\theta_0(z)$
$\theta_0(z)$
1
-1
1
-1
1
-1
1
-1
x_1
x_2