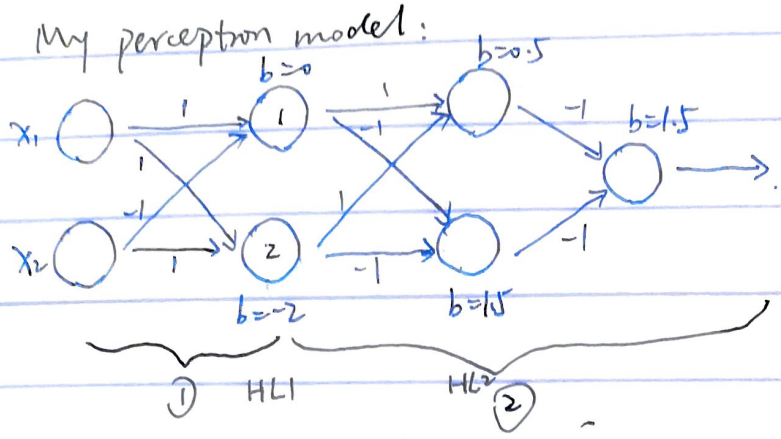
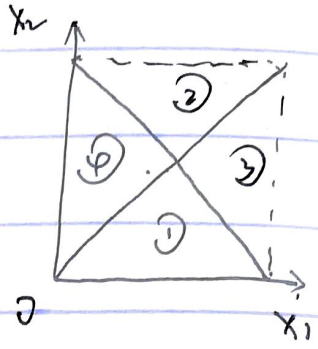
1. My perceptron model:





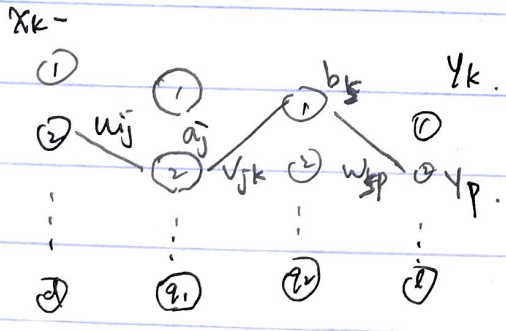
My model can be separated into 2 parts.

For the first part, I use the principles of linear programming to map the points in the first quadrant into two integers from $\{0, 1\}$. For the points in area ① and ②, $(x_1 - x_2) \cdot (x_1 + x_2 - 2) > 0$, so the outputs of the first hidden layer are both 1. For the points in ③ and ④, $(x_1 - x_2)(x_1 + x_2 - 2) < 0$. so the output of the first hidden layer are 0 and 1 for each node.

The second part can be described as an XNOR (exclusive NOR) gate. I just modified the weights and bias of a ~~nand~~ standard XOR gate such that the output will be 1 if the two inputs are the same and vice versa.

I just plunged some specific points to verify the model. But the procedure is difficult to present in this report, so I omit it.

2.



$x_k = (x_1^k, x_2^k, \cdots, x_d^k)$

$y_k = (y_1^k, y_2^k, \cdots, y_e^k)$.

(i) $x = (x_1, x_2, \cdots, x_d) \xrightarrow{x_i} a_j = f\left(\sum_{i=1}^{d} u_{ij} x_i\right) \xrightarrow{\alpha_j} a_j \xrightarrow{b_k = f\left(\sum_{j=1}^{q_1} v_{js} a_j\right)} b_s \xrightarrow{\beta_s} y_p = f\left(\sum_{k=1}^{q_2} w_{sp} b_s\right) \xrightarrow{\gamma_p} y_p$

$E_k = \frac{1}{2} \sum_{t=1}^{l} (\hat{y}_t^k - y_t^k)^2$.

② $\Delta w_{kp} = -\eta \frac{\partial E_k}{\partial w_{kp}}$

$\frac{\partial E_k}{\partial w_{kp}} = \frac{\partial E_k}{\partial \hat{y}_p^k} \cdot \frac{\partial \hat{y}_p^k}{\partial \gamma_p} \cdot \frac{\partial \gamma_p}{\partial w_{sp}}$

$= (\hat{y}_p^k - y_p^k) \cdot \hat{y}_p^k (1 - \hat{y}_p^k) \cdot b_s$

$\Delta w_{kp} = -\eta b_s (\hat{y}_p^k - y_p^k) \hat{y}_p^k (1 - \hat{y}_p^k) = \eta b_s g_p \quad (g_p = -\frac{\partial E_k}{\partial \gamma_p})$

② $\Delta v_{js} = -\eta \frac{\partial E_k}{\partial v_{js}}$

$\frac{\partial E_k}{\partial v_{js}} = \sum_{p=1}^{l} \frac{\partial E_k}{\partial \gamma_p} \cdot \frac{\partial \gamma_p}{\partial b_s} \cdot \frac{\partial b_s}{\partial \beta_s} \cdot \frac{\partial \beta_s}{\partial v_{js}}$

$= \sum_{p=1}^{l} -g_p \cdot w_{sp} \cdot b_s (1 - b_s) \cdot a_j$

$= -\sum_{p=1}^{l} g_p \cdot w_{sp} \cdot b_s (1 - b_s) \cdot a_j$

$= -a_j b_s (1 - b_s) \sum_{p=1}^{l} g_p w_{sp}$

$\Delta v_{jk} = \eta a_j b_s (1 - b_s) \sum_{p=1}^{l} g_p w_{sp} = \eta a_j e_s \quad (e_s = b_s (1 - b_s) \sum_{p=1}^{l} g_p w_{sp} = -\frac{\partial E_k}{\partial \beta_s})$

③ $\Delta u_{ij} = -\eta \frac{\partial E_k}{\partial u_{ij}}$

$\frac{\partial E_k}{\partial u_{ij}} = \sum_{s=1}^{q_2} \frac{\partial E_k}{\partial \beta_s} \cdot \frac{\partial \beta_s}{\partial a_j} \cdot \frac{\partial a_j}{\partial \alpha_j} \cdot \frac{\partial \alpha_j}{\partial u_{ij}}$

$= \sum_{s=1}^{q_2} \cdot -e_s \cdot v_{js} \cdot a_j (1 - a_j) \cdot x_i$

$\therefore \Delta u_{ij} = \eta a_j (1 - a_j) x_i \sum_{s=1}^{q_2} e_s v_{js} = \eta x_i h_j \quad (h_j = a_j (1 - a_j) \sum_{s=1}^{q_2} e_s v_{js} = -\frac{\partial E_k}{\partial \alpha_j})$

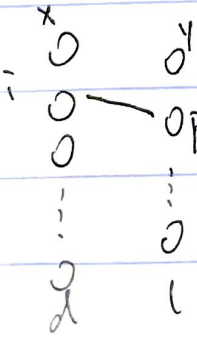
(ii) For the original model with 2 hidden layers:

$y_p = c \sum_{s=1}^{q_2} w_{sp} b_s = c \sum_{k=1}^{q_2} w_{sp} c\left(\sum_{j=1}^{q_1} v_{js} c\left(\sum_{i=1}^{d} u_{ij} x_i\right)\right)$

$= c \left\{ \sum_{s=1}^{q_2} \sum_{j=1}^{q_1} \sum_{i=1}^{d} w_{sp} v_{js} u_{ij} x_i \right\}$

let $\sum_{s=1}^{q_2} \sum_{j=1}^{q_1} w_{sp} v_{js} u_{ij} = u'_{ip}$, then $y_p = c \left\{ \sum_{i=1}^{d} u'_{ip} x_i \right\}$

For a simple neural network:



$y_p = c \sum_{i=1}^{d} u_{ip} x_i$

Therefore, the two networks are equivalent to each other.

Machine Learning HW4

PART 2

(i). I just ignore the warnings relating to GPU.

Epoch 1/10
1875/1875 - 2s - loss: 1.4529 - accuracy: 0.7020 - val_loss: 0.8840 - val_accuracy: 0.8346
Epoch 2/10
1875/1875 - 1s - loss: 0.7178 - accuracy: 0.8446 - val_loss: 0.5740 - val_accuracy: 0.8698
Epoch 3/10
1875/1875 - 1s - loss: 0.5334 - accuracy: 0.8706 - val_loss: 0.4657 - val_accuracy: 0.8862
Epoch 4/10
1875/1875 - 1s - loss: 0.4552 - accuracy: 0.8826 - val_loss: 0.4108 - val_accuracy: 0.8940
Epoch 5/10
1875/1875 - 1s - loss: 0.4115 - accuracy: 0.8899 - val_loss: 0.3779 - val_accuracy: 0.8999
Epoch 6/10
1875/1875 - 1s - loss: 0.3830 - accuracy: 0.8953 - val_loss: 0.3554 - val_accuracy: 0.9032
Epoch 7/10
1875/1875 - 1s - loss: 0.3626 - accuracy: 0.8987 - val_loss: 0.3398 - val_accuracy: 0.9057
Epoch 8/10
1875/1875 - 1s - loss: 0.3472 - accuracy: 0.9029 - val_loss: 0.3258 - val_accuracy: 0.9090
Epoch 9/10
1875/1875 - 1s - loss: 0.3348 - accuracy: 0.9057 - val_loss: 0.3155 - val_accuracy: 0.9105
Epoch 10/10
1875/1875 - 1s - loss: 0.3245 - accuracy: 0.9079 - val_loss: 0.3069 - val_accuracy: 0.9129
Test Accuracy on the test set: 0.913

(ii). **The results are shown below. The accuracy is improved after changing the activate function.**

Epoch 1/10
1875/1875 - 1s - loss: 0.6675 - accuracy: 0.8320 - val_loss: 0.3602 - val_accuracy: 0.9026
Epoch 2/10
1875/1875 - 1s - loss: 0.3406 - accuracy: 0.9050 - val_loss: 0.2963 - val_accuracy: 0.9194
Epoch 3/10
1875/1875 - 1s - loss: 0.2917 - accuracy: 0.9181 - val_loss: 0.2648 - val_accuracy: 0.9260
Epoch 4/10
1875/1875 - 1s - loss: 0.2617 - accuracy: 0.9271 - val_loss: 0.2429 - val_accuracy: 0.9324
Epoch 5/10
1875/1875 - 1s - loss: 0.2391 - accuracy: 0.9327 - val_loss: 0.2224 - val_accuracy: 0.9371
Epoch 6/10
1875/1875 - 1s - loss: 0.2210 - accuracy: 0.9386 - val_loss: 0.2061 - val_accuracy: 0.9430
Epoch 7/10
1875/1875 - 1s - loss: 0.2054 - accuracy: 0.9430 - val_loss: 0.1958 - val_accuracy: 0.9456
Epoch 8/10
1875/1875 - 1s - loss: 0.1924 - accuracy: 0.9464 - val_loss: 0.1844 - val_accuracy: 0.9486

Epoch 9/10
1875/1875 - 1s - loss: 0.1809 - accuracy: 0.9497 - val_loss: 0.1740 - val_accuracy: 0.9517
Epoch 10/10
1875/1875 - 1s - loss: 0.1711 - accuracy: 0.9523 - val_loss: 0.1652 - val_accuracy: 0.9540
Test Accuracy on the test set: 0.954

(iii). **The accuracy does change.**
Epoch 1/10
1875/1875 - 2s - loss: 0.2704 - accuracy: 0.9227 - val_loss: 0.1469 - val_accuracy: 0.9571
Epoch 2/10
1875/1875 - 1s - loss: 0.1217 - accuracy: 0.9636 - val_loss: 0.1046 - val_accuracy: 0.9681
Epoch 3/10
1875/1875 - 1s - loss: 0.0853 - accuracy: 0.9742 - val_loss: 0.0880 - val_accuracy: 0.9738
Epoch 4/10
1875/1875 - 1s - loss: 0.0658 - accuracy: 0.9801 - val_loss: 0.0764 - val_accuracy: 0.9768
Epoch 5/10
1875/1875 - 1s - loss: 0.0508 - accuracy: 0.9844 - val_loss: 0.0823 - val_accuracy: 0.9752
Epoch 6/10
1875/1875 - 1s - loss: 0.0421 - accuracy: 0.9868 - val_loss: 0.0751 - val_accuracy: 0.9780
Epoch 7/10
1875/1875 - 1s - loss: 0.0338 - accuracy: 0.9890 - val_loss: 0.0738 - val_accuracy: 0.9784
Epoch 8/10
1875/1875 - 1s - loss: 0.0285 - accuracy: 0.9913 - val_loss: 0.0745 - val_accuracy: 0.9786
Epoch 9/10
1875/1875 - 1s - loss: 0.0241 - accuracy: 0.9926 - val_loss: 0.0834 - val_accuracy: 0.9759
Epoch 10/10
1875/1875 - 1s - loss: 0.0202 - accuracy: 0.9941 - val_loss: 0.0768 - val_accuracy: 0.9794
Test Accuracy on the test set: 0.979
The accuracy becomes even better, so 'adam' is the better optimizer.

(iv). **The accuracy slightly changes, but makes no difference.**
Epoch 1/10
1875/1875 - 2s - loss: 0.2586 - accuracy: 0.9264 - val_loss: 0.1442 - val_accuracy: 0.9575
Epoch 2/10
1875/1875 - 1s - loss: 0.1141 - accuracy: 0.9666 - val_loss: 0.0961 - val_accuracy: 0.9699
Epoch 3/10
1875/1875 - 1s - loss: 0.0790 - accuracy: 0.9758 - val_loss: 0.0945 - val_accuracy: 0.9700
Epoch 4/10
1875/1875 - 1s - loss: 0.0593 - accuracy: 0.9822 - val_loss: 0.0835 - val_accuracy: 0.9746
Epoch 5/10
1875/1875 - 1s - loss: 0.0450 - accuracy: 0.9862 - val_loss: 0.0755 - val_accuracy: 0.9762
Epoch 6/10
1875/1875 - 1s - loss: 0.0366 - accuracy: 0.9886 - val_loss: 0.0734 - val_accuracy: 0.9770
Epoch 7/10

1875/1875 - 1s - loss: 0.0288 - accuracy: 0.9911 - val_loss: 0.0827 - val_accuracy: 0.9757
Epoch 8/10
1875/1875 - 1s - loss: 0.0245 - accuracy: 0.9923 - val_loss: 0.0698 - val_accuracy: 0.9803
Epoch 9/10
1875/1875 - 1s - loss: 0.0196 - accuracy: 0.9939 - val_loss: 0.0685 - val_accuracy: 0.9795
Epoch 10/10
1875/1875 - 1s - loss: 0.0163 - accuracy: 0.9948 - val_loss: 0.0777 - val_accuracy: 0.9795
Test Accuracy on the test set: 0.979

(v). **The accuracy even decreased a little bit.**
Epoch 1/10
1875/1875 - 1s - loss: 0.2392 - accuracy: 0.9301 - val_loss: 0.1349 - val_accuracy: 0.9578
Epoch 2/10
1875/1875 - 2s - loss: 0.1003 - accuracy: 0.9701 - val_loss: 0.0912 - val_accuracy: 0.9714
Epoch 3/10
1875/1875 - 1s - loss: 0.0722 - accuracy: 0.9776 - val_loss: 0.1076 - val_accuracy: 0.9676
Epoch 4/10
1875/1875 - 1s - loss: 0.0562 - accuracy: 0.9815 - val_loss: 0.0766 - val_accuracy: 0.9755
Epoch 5/10
1875/1875 - 1s - loss: 0.0442 - accuracy: 0.9856 - val_loss: 0.0883 - val_accuracy: 0.9739
Epoch 6/10
1875/1875 - 1s - loss: 0.0373 - accuracy: 0.9879 - val_loss: 0.0902 - val_accuracy: 0.9741
Epoch 7/10
1875/1875 - 2s - loss: 0.0294 - accuracy: 0.9902 - val_loss: 0.0857 - val_accuracy: 0.9755
Epoch 8/10
1875/1875 - 1s - loss: 0.0250 - accuracy: 0.9918 - val_loss: 0.0862 - val_accuracy: 0.9765
Epoch 9/10
1875/1875 - 2s - loss: 0.0225 - accuracy: 0.9923 - val_loss: 0.0875 - val_accuracy: 0.9759
Epoch 10/10
1875/1875 - 1s - loss: 0.0207 - accuracy: 0.9928 - val_loss: 0.0889 - val_accuracy: 0.9771
Test Accuracy on the test set: 0.977

(vi). **The accuracy becomes much higher.**
Epoch 1/10
1875/1875 - 6s - loss: 0.1768 - accuracy: 0.9476 - val_loss: 0.0703 - val_accuracy: 0.9773
Epoch 2/10
1875/1875 - 6s - loss: 0.0618 - accuracy: 0.9818 - val_loss: 0.0572 - val_accuracy: 0.9818
Epoch 3/10
1875/1875 - 6s - loss: 0.0423 - accuracy: 0.9866 - val_loss: 0.0555 - val_accuracy: 0.9813
Epoch 4/10
1875/1875 - 6s - loss: 0.0305 - accuracy: 0.9907 - val_loss: 0.0543 - val_accuracy: 0.9834
Epoch 5/10
1875/1875 - 6s - loss: 0.0208 - accuracy: 0.9935 - val_loss: 0.0709 - val_accuracy: 0.9770
Epoch 6/10

1875/1875 - 6s - loss: 0.0164 - accuracy: 0.9951 - val_loss: 0.0463 - val_accuracy: 0.9860
Epoch 7/10
1875/1875 - 6s - loss: 0.0116 - accuracy: 0.9963 - val_loss: 0.0468 - val_accuracy: 0.9865
Epoch 8/10
1875/1875 - 6s - loss: 0.0100 - accuracy: 0.9966 - val_loss: 0.0575 - val_accuracy: 0.9852
Epoch 9/10
1875/1875 - 6s - loss: 0.0080 - accuracy: 0.9977 - val_loss: 0.0495 - val_accuracy: 0.9861
Epoch 10/10
1875/1875 - 6s - loss: 0.0060 - accuracy: 0.9980 - val_loss: 0.0524 - val_accuracy: 0.9870
Test Accuracy on the test set: 0.987

(vii). **The accuracy and the description of the model are as follows. I use the model.summary() function to help report the structure of this neural network.**

Epoch 1/10
469/469 - 8s - loss: 0.1713 - accuracy: 0.9474 - val_loss: 0.0517 - val_accuracy: 0.9839
Epoch 2/10
469/469 - 7s - loss: 0.0500 - accuracy: 0.9844 - val_loss: 0.0409 - val_accuracy: 0.9874
Epoch 3/10
469/469 - 7s - loss: 0.0338 - accuracy: 0.9893 - val_loss: 0.0317 - val_accuracy: 0.9902
Epoch 4/10
469/469 - 7s - loss: 0.0257 - accuracy: 0.9920 - val_loss: 0.0315 - val_accuracy: 0.9898
Epoch 5/10
469/469 - 7s - loss: 0.0200 - accuracy: 0.9934 - val_loss: 0.0269 - val_accuracy: 0.9921
Epoch 6/10
469/469 - 7s - loss: 0.0159 - accuracy: 0.9948 - val_loss: 0.0283 - val_accuracy: 0.9906
Epoch 7/10
469/469 - 7s - loss: 0.0139 - accuracy: 0.9953 - val_loss: 0.0303 - val_accuracy: 0.9906
Epoch 8/10
469/469 - 8s - loss: 0.0129 - accuracy: 0.9957 - val_loss: 0.0283 - val_accuracy: 0.9911
Epoch 9/10
469/469 - 8s - loss: 0.0101 - accuracy: 0.9966 - val_loss: 0.0254 - val_accuracy: 0.9920
Epoch 10/10
469/469 - 8s - loss: 0.0085 - accuracy: 0.9972 - val_loss: 0.0280 - val_accuracy: 0.9923
Test Accuracy on the test set: 0.992

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 25, 25, 32)	544
max_pooling2d (MaxPooling2D)	(None, 12, 12, 32)	0
conv2d_1 (Conv2D)	(None, 9, 9, 64)	32832

max_pooling2d_1 (MaxPooling2	(None, 4, 4, 64)	0
dropout (Dropout)	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 800)	820000
dense_1 (Dense)	(None, 10)	8010

Total params: 861,386
Trainable params: 861,386
Non-trainable params: 0

Some details:
conv2d: Convolution layer with 32 4 by 4 filters, the activation is relu.
max_pooling2d: Max pooling layer with 2 by 2 pooling window.
conv2d_1: Convolution layer with 64 4 by 4 filters, the activation is relu.
max_pooling2d_1: Max pooling layer with 2 by 2 pooling window.
dropout: I drop out 10% of the data.
flatten
dense: First hidden layer with 800 hidden nodes with the relu activation function.
dense_1: The output layer with 10 classes output with the sigmoid activation function.