Exercise 1

i)

$$l_{PR}(w) := -\sum_{i=1}^{n} y_i \log \psi(\phi(x_i)^T w) + (1-y_i)\log \psi(-\phi(x_i)^T w)$$

$$= -\sum_{i=1} y_i \log \psi(\Sigma_{ji} w_j) + (1-y_i)\log \psi(\Sigma_{ji} w_j)$$

$$\frac{\partial w_j}{\partial w_a} = -\sum_{i=1} \frac{y_i}{\psi(\Sigma_{ji} w_j)} \frac{\partial(\psi(\Sigma_{ji} w_j))}{\partial w_a} + \frac{1-y_i}{\psi(-\Sigma_{ji} w_j)} \cdot \frac{\partial \psi(-\Sigma_{ji} w_j)}{\partial w_a}$$

$$\psi(x) = \int_{-\infty}^{x} \psi(t)\, dt$$

$$\psi'(x) = \psi(x)$$

chain rule since $\frac{\partial}{\partial w}$

$$\psi'(\Sigma_{ji} w_j) = \psi(\Sigma_{ji} w_j) \cdot \Sigma_{ji}$$

$$\psi'(-\Sigma_{ji} w_j) = \psi(-\Sigma_{ji} w_j) \cdot \Sigma_{ij}$$

$$= -\sum_{i=1} \frac{y_i \psi(\Sigma_{ji} w_j) \cdot \Sigma_{ji}\, \delta_{ja}}{\psi(\Sigma_{ji} w_j)} + \frac{(1-y_i)(\psi(-\Sigma_{ji} w_j)(-\Sigma_{ji})\, \delta_{ja}}{\psi(-\Sigma_{ji} w_j)}$$

$$= \sum_{i=1} -\frac{y_i \psi(\Sigma_{ji} w_j) \cdot \Sigma_{ji}\, \delta_{ja}}{\psi(\Sigma_{ji} w_j)} - \frac{(1-y_i)(\psi(-\Sigma_{ji} w_j)(-\Sigma_{ji})\, \delta_{ja}}{\psi(-\Sigma_{ji} w_j)}$$

$$= \sum_{i=1} -\frac{y_i \psi(\Sigma_{ji} w_j) \cdot \Sigma_{ji}\, \delta_{ja}}{\psi(\Sigma_{ji} w_j)} + \frac{(1-y_i)(\psi(-\Sigma_{ji} w_j)(\Sigma_{ji})\, \delta_{ja}}{\psi(-\Sigma_{ji} w_j)}$$

$$= \sum_{i=1} -\frac{y_i \psi(\Sigma_{ai} w_a) \cdot \Sigma_{ai}}{\psi(\Sigma_{ai} w_a)} + \frac{(1-y_i)(\psi(-\Sigma_{ai} w_a)(\Sigma_{ai})}{\psi(-\Sigma_{ai} w_a)}$$

$$= \left(\sum_{i=1} -\frac{y_i \psi(\phi(x_i)^T w) \cdot \phi(x_i)}{\psi(\phi(x_i)^T w)} + \frac{(1-y_i)\psi(\phi(x_i)^T w) \cdot \phi(x_i)}{\psi(-\phi(x_i)^T w)} \right)_a$$

$$= \left(\sum_{i=1} \psi(\phi(x_i)^T w) \left[-\frac{y_i}{\psi(\phi(x_i)^T w)} + \frac{(1-y_i)}{\psi(-\phi(x_i)^T w)} \right] \phi(x_i) \right)_a$$

and thus we conclude that

$$\nabla_w l_{PR}(w) = \sum_{i=1}^{n} \psi(\phi(x_i)^T w) \left[\frac{(1-y_i)}{\psi(-\phi(x_i)^T w)} - \frac{y_i}{\psi(\phi(x_i)^T w)} \right] \phi(x_i)$$

Furthermore we can expand and see that:

$$\left[\frac{(1-y_1)\psi(\phi(x_1)^T w)}{\psi(-\phi(x_1)^T w)} - \frac{y_1\psi(\phi(x_1)^T w)}{\psi(\phi(x_1)^T w)}\right]\phi(x_1) +$$

$$\left[\frac{(1-y_2)\psi(\phi(x_2)^T w)}{\psi(-\phi(x_2)^T w)} - \frac{y_2\psi(\phi(x_2)^T w)}{\psi(\phi(x_2)^T w)}\right]\phi(x_2) +$$

$$+ \cdots\cdots\cdots + \left[\frac{(1-y_n)\psi(\phi(x_n)^T w)}{\psi(-\phi(x_n)^T w)} - \frac{y_n\psi(\phi(x_n)^T w)}{\psi(\phi(x_n)^T w)}\right]\phi(x_n)$$

can be written in matrix form:

$$\underbrace{\begin{bmatrix} | & | & & | \\ \phi(x_1) & \phi(x_2) & \cdots & \phi(x_N) \\ | & | & & | \end{bmatrix}}_{\Phi \;\leftarrow\; p \times N \;\rightarrow} \underbrace{\begin{bmatrix} \frac{(1-y_1)\psi(\phi(x_1)^T w)}{\psi(-\phi(x_1)^T w)} \\ \frac{(1-y_2)\psi(\phi(x_2)^T w)}{\psi(-\phi(x_2)^T w)} \\ \vdots \\ \frac{(1-y_n)\psi(\phi(x_1)^T w)}{\psi(-\phi(x_n)^T w)} \end{bmatrix}}_{Q_1 \;\leftarrow\; N \times 1 \;\rightarrow} - \underbrace{\begin{bmatrix} \frac{y_1\psi(\phi(x_1)^T w)}{\psi(\phi(x_1)^T w)} \\ \frac{y_2\psi(\phi(x_2)^T w)}{\psi(\phi(x_2)^T w)} \\ \vdots \\ \frac{y_n\psi(\phi(x_n)^T w)}{\psi(\phi(x_n)^T w)} \end{bmatrix}}_{Q_2}$$

which then gives us the desired form of

$$\nabla_w \ell_{PR}(w) = \Phi(Q_1 - Q_2)$$

ii) $\psi(x) = (2\pi)^{-1/2} \exp(-x^2/2)$ (I)

Now plugging in $\psi(-x) = (2\pi)^{-1/2} \exp(-(-x)^2/2)$

$= (2\pi)^{-1/2} \exp(-x^2/2)$

(by (I)) $= \psi(x)$

We know $\Psi(x) = \text{cdf of } N(0,1)$

$= \mathbb{P}(Z \le x)$ (I)

$\Psi(-x) = \mathbb{P}(Z \le -x)$

$= \mathbb{P}(Z \ge x)$

$= 1 - \mathbb{P}(Z \le x)$

$= 1 - \Psi(x)$ ← (by (I))

$\psi(x) = (2\pi)^{-1/2} \exp(-x^2/2)$ (I)

$\psi'(x) = (2\pi)^{-1/2} \exp(-x^2/2) \cdot (-\frac{2x}{2})$

$= -x(2\pi)^{-1/2} \exp(-x^2/2)$

$= -x\,\psi(x)$ ← (by (I))

$$\frac{\partial}{\partial x} = \frac{\psi'(x)}{\Psi(x)} = \frac{\Psi(x)\psi'(x) - \psi(x)\psi'(x)}{\Psi(x)^2}$$

(since $\psi'(x) = -x\,\psi(x)$ and $\Psi'(x) = \psi(x)$)

$$= \frac{\Psi(x)(-x\,\psi(x)) - \psi(x)\psi(x)}{\Psi(x)^2}$$

$$= \frac{\psi(x)[-x\Psi(x) - \psi(x)]}{\Psi(x)^2}$$ ■

$$\frac{\partial}{\partial x}\frac{\psi(x)}{\Psi(-x)} = \frac{\partial}{\partial x}\frac{\psi(x)}{1-\Psi(x)} = \frac{\Psi(-x)\psi'(x) - \psi(x)\Psi'(-x)}{\Psi(-x)^2}$$

$$= \frac{\Psi(-x)(-x\,\psi(x)) - \psi(x)(1-\Psi(x))}{\Psi(-x)^2}$$

$$= \frac{\Psi(-x)(-x\,\psi(x)) - \psi(x)(-\psi(x))}{\Psi(-x)^2}$$

$$= \frac{\psi(x)[-x\Psi(-x) + \psi(x)]}{\Psi(-x)^2}$$

iii)

$$\begin{bmatrix} \frac{(1-y_1)\psi(\phi(x_1)^T w)}{\psi(-\phi(x_1)^T w)} - \frac{y_1 \psi(\phi(x_1)^T w)}{\psi(\phi(x_1)^T w)} \\ \frac{(1-y_2)\psi(\phi(x_2)^T w)}{\psi(-\phi(x_2)^T w)} - \frac{y_2 \psi(\phi(x_2)^T w)}{\psi(\phi(x_2)^T w)} \\ \vdots \\ \underbrace{\frac{(1-y_n)\psi(\phi(x_1)^T w)}{\psi(-\phi(x_n)^T w)}}_{Q_1} - \underbrace{\frac{y_n \psi(\phi(x_n)^T w)}{\psi(\phi(x_n)^T w)}}_{Q_2} \end{bmatrix}$$

Now let $z_i = \phi(x_i)^T w$ which lets us rewrite as

$$Q_1^T = \left[(1-y_1)\frac{\psi(z_1)}{\psi(-z_1)} \cdots\cdots (1-y_N)\frac{\psi(z_N)}{\psi(-z_N)} \right]$$

$$Q_2^T = \left[y_1 \frac{\psi(z_1)}{\psi(z_1)} \cdots\cdots\cdots y_N \frac{\psi(z_N)}{\psi(z_N)} \right]$$

$$(Q_1^T - Q_2^T) =$$

$$\left[(1-y_1)\frac{\psi(z_1)}{\psi(-z_1)} \cdots\cdots (1-y_N)\frac{\psi(z_N)}{\psi(-z_N)} \right] - \left[y_1 \frac{\psi(z_1)}{\psi(z_1)} \cdots\cdots\cdots y_N \frac{\psi(z_N)}{\psi(z_N)} \right]$$

$$\frac{\partial (Q_1^T - Q_2^T)}{\partial w_a} =$$

$$\left[(1-y_1)\frac{\psi(z_1)}{\psi(-z_1)}\frac{\partial}{\partial w_a} \cdots\cdots (1-y_N)\frac{\psi(z_N)}{\psi(-z_N)}\frac{\partial}{\partial w_a} \right] - \left[y_1 \frac{\psi(z_1)}{\psi(z_1)}\frac{\partial}{\partial w_a} \cdots\cdots\cdots y_N \frac{\psi(z_N)}{\psi(z_N)}\frac{\partial}{\partial w_a} \right]$$

(plug in from (ii) on each term)

$$\left[\left((1-y_1)\frac{\psi(z_1)[-z_1\psi(z_1) + \psi(z_1)]}{\psi(z_1)^2} \cdot \phi(x_1) \right)_a \cdots\cdots \left((1-y_N)\frac{\psi(z_N)[-z_N\psi(z_N) + \psi(z_N)]}{\psi(z_N)^2} \cdot \phi(x_N) \right)_a \right] -$$

$$\left[\left(y_1 \frac{\psi(z_1)[-z_1\psi(z_1) - \psi(z_1)]}{\psi(z_1)^2} \cdot \phi(x_1) \right)_a \cdots\cdots \left(y_N \frac{\psi(z_N)[-z_N\psi(z_N) - \psi(z_N)]}{\psi(z_N)^2} \cdot \phi(x_N) \right)_a \right]$$

(applying chain rule)

Rearranging into one array:

$$\left[\left(1-y_1\right)\frac{\psi(z_1)\left[-z_1\psi(z_1)+\psi(z_1)\right]}{\psi(-z_1)^2}\cdot\phi(x_1)\right)_a - \left(y_1\frac{\psi(z_1)\left[-z_1\psi(z_1)-\psi(z_1)\right]}{\psi(z_1)^2}\cdot\phi(x_1)\right)_a \cdots\cdots\cdots$$

$$\left(1-y_N\right)\frac{\psi(-z_N)\left[-z_N\psi(z_N)+\psi(z_N)\right]}{\psi(-z_N)^2}\cdot\phi(x_1)\right)_a - \left(y_N\frac{\psi(z_N)\left[-z_N\psi(z_N)-\psi(z_N)\right]}{\psi(z_N)^2}\cdot\phi(x_N)\right)_a \Bigg]$$

Now lets factor out the $\phi(x_i)$ and redistribute negative sign:

$$\left[\left(1-y_1\right)\frac{\psi(z_1)\left[-z_1\psi(z_1)+\psi(z_1)\right]}{\psi(-z_1)^2}\cdot + y_1\frac{\psi(z_1)\left[z_1\psi(z_1)+\psi(z_1)\right]}{\psi(z_1)^2}\right)\phi(x_1)\cdots\cdots\cdots$$

$$\left(1-y_N\right)\frac{\psi(-z_N)\left[-z_1\psi(z_N)+\psi(z_N)\right]}{\psi(-z_N)^2} + y_N\frac{\psi(z_N)\left[z_N\psi(z_N)+\psi(z_N)\right]}{\psi(z_N)^2}\right)\phi(x_N)\Bigg]_a$$

Now let $G_i = (1-y_i)\dfrac{\psi(z_i)\left[-z_i\psi(-z_i)+\psi(z_i)\right]}{\psi(-z_i)^2} + \dfrac{y_i\left[z_i\psi(z_i)+\psi(z_i)\right]}{\psi(z_i)^2}$

and rewritting our matrix gives us:

$$\begin{bmatrix} | & & | \\ G_1\phi(x_1) & \cdots & G_N\phi(x_N) \\ | & & | \end{bmatrix}$$

which we can rewrite as:

$$\begin{bmatrix} | & & | \\ G_1\phi(x_1) & \cdots & G_N\phi(x_N) \\ | & & | \end{bmatrix} = \underbrace{\begin{bmatrix} | & & | \\ \phi(x_1) & \cdots & \phi(x_N) \\ | & & | \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} G_1 & 0 & \cdots & 0 \\ 0 & G_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & & G_N \end{bmatrix}}_{D_i}$$

iv) Hessian of $L = (\nabla^2 L)_{ab} = \frac{\partial}{\partial w_b}\left(\frac{\partial L}{\partial w_a}\right)$

from part i) we have:

$$\frac{\partial L}{\partial w_a} = \frac{-y_i \pi(\phi_i^T w)\, \phi_{ai}}{\psi(\phi_i^T w)} + \frac{(1-y_i)\pi(\phi_i^T w)\,\phi_{ai}}{\psi(-\phi_i^T w)}$$

so now:

$$\frac{\partial}{\partial w_b}\left(\frac{\partial L}{\partial w_a}\right) = \frac{\partial}{\partial w_b}\left(\frac{-y_i \pi(\phi_i^T w)\,\phi_{ai}}{\psi(\phi_i^T w)} + \frac{(1-y_i)\pi(\phi_i^T w)\,\phi_{ai}}{\psi(-\phi_i^T w)}\right)$$

(f = $-y_i \pi(\phi_i^T w)\phi_{ai}$, $g = \psi(\phi_i^T w)$; f = $(1-y_i)\pi(\phi_i^T w)\phi_{ai}$, $g = \psi(-\phi_i^T w)$)

($\pi'(x) = x\,\pi(x)$ part i); $\psi'(x) = \pi(x)$)

$$= -y_i \frac{\phi_{ai}\left((\phi_i^T w\, \pi(\phi_i^T w)\phi_{bi})(\psi(\phi_i^T w)) - (\pi(\phi_i^T w))\,\pi(\phi_i^T w)\phi_{bi}\right)}{\psi(\phi_i^T w)^2}$$

(f', g, f, g'; g^2)

$$+ \frac{(1-y_i)\phi_{ai}\left(\phi_i^T w\, \pi(\phi_i^T w)\phi_{ai})(\psi(-\phi_i^T w)) - (\pi(\phi_i^T w)(-\phi_{bi}\,\pi(\phi_i^T w))\right)}{\psi(-\phi_i^T w)^2}$$

(f', g, f, g' ← chain rule gives (−); g^2)

Now let $z_i = \phi(x_i)^T w$ and let

$$G_i = (1-y_i)\frac{\psi(z_i)[-z_i\psi(-z_i) + \psi(z_i)]}{\psi(-z_i)^2} + \frac{y_i[z_i\psi(z_i) + \pi(z_i)]}{\psi(z_i)^2}$$

which lets us rewrite as:

$$\phi_{ai} I_{bi} \begin{bmatrix} G_1 & 0 & \cdots & 0 \\ 0 & G_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & G_N \end{bmatrix}$$

and finally $H = \nabla^2 L = \phi_{ai}\phi_{bi} D_{ii}$ ∎

Exercise 2

i)

```
def fit_PR_GD(Y, H, W0=None, sub_iter=100, stopping_diff=0.01):
        '''
        Convex optimization algorithm for Probit Regression using Gradient Descent
        Y = (n x 1), H = (p x n) (\Phi in lecture note), W = (p x 1)
        Logistic Regression: Y ~ Bernoulli(Q), Q = Probit(H.T @ W)
        '''
        def loss(w):
            result = 0
            for i in range(H.shape[1]):
                result += Y[i]*np.log(norm.cdf(H[:,i].T @ w)) + (1-Y[i])*np.log(norm.cdf(-H[:,i].T @ w))
            return (-result)


        if W0 is None:
            W0 = np.random.rand(H.shape[0],1) #If initial coefficients W0 is None, randomly initialize

        W1 = W0.copy()
        i = 0
        grad = np.ones(W0.shape)
        while (i < sub_iter) and (np.linalg.norm(grad) > stopping_diff):
            Q = norm.pdf(H.T @ W1) * ( (1-Y)/norm.cdf(-H.T @ W1) - Y/norm.cdf(H.T @ W1) )

            # grad = H @ (Q - Y).T + alpha * np.ones(W0.shape[1])
            grad = H @ Q
            W1 = W1 - (np.log(i+1) / (((i + 1) ** (0.5)))) * grad
            i = i + 1
            print(f"loss of iter {i} = {loss(W1)}")
            # print('iter %i, grad_norm %f' %(i, np.linalg.norm(grad)))
        return W1
```

```
loss of iter 1 = [8353.44488145]
loss of iter 2 = [7760.17745326]
loss of iter 3 = [7105.56364897]
loss of iter 4 = [6505.93216464]
loss of iter 5 = [5979.68659287]
loss of iter 6 = [5521.6733652]
loss of iter 7 = [5122.30426431]
loss of iter 8 = [4772.37404871]
loss of iter 9 = [4464.05832262]
loss of iter 10 = [4190.9175486]
loss of iter 11 = [3947.68268518]
loss of iter 12 = [3730.02922008]
loss of iter 13 = [3534.3867775]
loss of iter 14 = [3357.78812404]
loss of iter 15 = [3197.75127268]
loss of iter 16 = [3052.18741152]
loss of iter 17 = [2919.32857541]
loss of iter 18 = [2797.67039074]
                ⋮
loss of iter 82 = [857.90045256]
loss of iter 83 = [850.28817461]
loss of iter 84 = [842.84780586]
loss of iter 85 = [835.57345973]
loss of iter 86 = [828.45951566]
loss of iter 87 = [821.50060437]
loss of iter 88 = [814.69159397]
loss of iter 89 = [808.02757706]
loss of iter 90 = [801.50385861]
loss of iter 91 = [795.11594464]
loss of iter 92 = [788.85953159]
loss of iter 93 = [782.7304964]
loss of iter 94 = [776.72488713]
loss of iter 95 = [770.83891422]
loss of iter 96 = [765.06894222]
loss of iter 97 = [759.41148212]
loss of iter 98 = [753.86318398]
loss of iter 99 = [748.42083014]
loss of iter 100 = [743.08132874]
```

We can see that the loss value in each iteration decreases monotonically after every iteration. The code runs until it reaches the condition of 100 iterations, where the loss value ends up being 743. I did notice although that this method takes much more computational power than LR method as it takes our program much longer to run.

ii)

$\eta = 10\log(t+1)/\sqrt{t+1}$

```
loss of iter 1 = [8350.80251607]
loss of iter 2 = [4186.09534378]
loss of iter 3 = [2427.25907745]
loss of iter 4 = [1670.97268953]
loss of iter 5 = [1272.9310954]
loss of iter 6 = [1033.19469666]
loss of iter 7 = [874.5846982]
loss of iter 8 = [762.35384974]
loss of iter 9 = [678.89517479]
loss of iter 10 = [614.43338422]
loss of iter 11 = [563.14015246]
loss of iter 12 = [521.33767311]
loss of iter 13 = [486.59596375]
loss of iter 14 = [457.24742633]
loss of iter 15 = [432.11070855]
loss of iter 16 = [410.3260407]
loss of iter 17 = [391.25303119]
loss of iter 18 = [374.40500638]
                  ⋮
loss of iter 82 = [148.78306181]
loss of iter 83 = [147.91517615]
loss of iter 84 = [147.06544808]
loss of iter 85 = [146.23326774]
loss of iter 86 = [145.41805305]
loss of iter 87 = [144.61924813]
loss of iter 88 = [143.83632183]
loss of iter 89 = [143.06876634]
loss of iter 90 = [142.31609591]
loss of iter 91 = [141.57784566]
loss of iter 92 = [140.85357043]
loss of iter 93 = [140.14284374]
loss of iter 94 = [139.44525682]
loss of iter 95 = [138.76041766]
loss of iter 96 = [138.08795019]
loss of iter 97 = [137.42749341]
loss of iter 98 = [136.77870069]
loss of iter 99 = [136.14123899]
loss of iter 100 = [135.51478824]
```

We can see that it monotonically dereases. This step size performs much better than the original step size in part i). It initially dereases very fast and gets slower as our loss function reached it's minimum. Iteration 100 stopped at 135.

$\eta = 1$

```
loss of iter 1 = [7175.85784388]
loss of iter 2 = [6306.66479441]
loss of iter 3 = [5624.24541304]
loss of iter 4 = [5068.41476007]
loss of iter 5 = [4604.80748913]
loss of iter 6 = [4211.74944584]
loss of iter 7 = [3874.46170053]
loss of iter 8 = [3582.27104004]
loss of iter 9 = [3327.14705552]
loss of iter 10 = [3102.86937274]
loss of iter 11 = [2904.51715264]
loss of iter 12 = [2728.13579559]
loss of iter 13 = [2570.50797283]
loss of iter 14 = [2428.98996148]
loss of iter 15 = [2301.39105077]
loss of iter 16 = [2185.88259821]
loss of iter 17 = [2080.92820392]
loss of iter 18 = [1985.22933355]
                  ⋮
loss of iter 88 = [497.81481619]
loss of iter 89 = [493.16221032]
loss of iter 90 = [488.61154143]
loss of iter 91 = [484.15946642]
loss of iter 92 = [479.80278636]
loss of iter 93 = [475.53843884]
loss of iter 94 = [471.36349078]
loss of iter 95 = [467.27513177]
loss of iter 96 = [463.27066766]
loss of iter 97 = [459.34751475]
loss of iter 98 = [455.50319411]
loss of iter 99 = [451.73532642]
loss of iter 100 = [448.041627011]
```

Monotonically dereases as well. This one at first performed as well as our original step size, but by the 100 iteration our loss function ended at 448, which is better than the original but not as well as

←

$\eta = 1/t$

```
loss of iter 1 = [7204.50932662]
loss of iter 2 = [6751.94506443]
loss of iter 3 = [6482.69357597]
loss of iter 4 = [6294.70420513]
loss of iter 5 = [6151.91008696]
loss of iter 6 = [6037.63199085]
loss of iter 7 = [5942.86453301]
loss of iter 8 = [5862.22372248]
loss of iter 9 = [5792.24840345]
loss of iter 10 = [5730.58908979]
loss of iter 11 = [5675.58186313]
loss of iter 12 = [5626.00760828]
loss of iter 13 = [5580.94788035]
loss of iter 14 = [5539.6944702]
loss of iter 15 = [5501.69040688]
loss of iter 16 = [5466.4901817]
loss of iter 17 = [5433.73216616]
loss of iter 18 = [5403.11901386]
                  ⋮
loss of iter 82 = [4681.71662047]
loss of iter 83 = [4676.60098558]
loss of iter 84 = [4671.55588231]
loss of iter 85 = [4666.57951201]
loss of iter 86 = [4661.67014245]
loss of iter 87 = [4656.82610453]
loss of iter 88 = [4652.04578936]
loss of iter 89 = [4647.32764541]
loss of iter 90 = [4642.67017584]
loss of iter 91 = [4638.07193599]
loss of iter 92 = [4633.53153102]
loss of iter 93 = [4629.04761368]
loss of iter 94 = [4624.61888216]
loss of iter 95 = [4620.24407813]
loss of iter 96 = [4615.92198485]
loss of iter 97 = [4611.65142535]
loss of iter 98 = [4607.43126073]
loss of iter 99 = [4603.26038862]
loss of iter 100 = [4599.13774157]
```

Monotonically dereases as well. This one performed WORSE than the original. It also did worse than the two on the left

←

iii) After trying out a few options, it seemed that our loss function would decrease faster by choosing a smaller δ and a larger r. The optimal paramaters I choose were $\delta = 0.00001$ and $r = 95$. Choosing a bigger r value would cause issues, (I'm assuming we had to big of a step size causing us to miss our min and making our loss function nan). The following are the results of step size $\eta_t = 95 + 0.00001$. This was better performance than any of the above step sizes.

```
loss of iter 1 = [1279.16250167]
loss of iter 2 = [528.89379941]
loss of iter 3 = [379.40763624]
loss of iter 4 = [311.93796957]
loss of iter 5 = [275.40179095]
loss of iter 6 = [253.06272691]
loss of iter 7 = [237.86897582]
loss of iter 8 = [226.51721999]
loss of iter 9 = [217.3697411]
loss of iter 10 = [209.58071304]
loss of iter 11 = [202.69624818]
loss of iter 12 = [196.46228046]
loss of iter 13 = [190.72933857]
loss of iter 14 = [185.40420073]
loss of iter 15 = [180.4248363]
loss of iter 16 = [175.74715574]
loss of iter 17 = [171.33785601]
loss of iter 18 = [167.1704645]
```

.....

```
loss of iter 82 = [69.19669762]
loss of iter 83 = [68.60472815]
loss of iter 84 = [68.02349648]
loss of iter 85 = [67.45270236]
loss of iter 86 = [66.89205674]
loss of iter 87 = [66.34128131]
loss of iter 88 = [65.80010796]
loss of iter 89 = [65.26827836]
loss of iter 90 = [64.74554348]
loss of iter 91 = [64.23166322]
loss of iter 92 = [63.726406]
loss of iter 93 = [63.22954838]
loss of iter 94 = [62.74087472]
loss of iter 95 = [62.26017682]
loss of iter 96 = [61.78725366]
loss of iter 97 = [61.32191102]
loss of iter 98 = [60.86396128]
loss of iter 99 = [60.41322306]
loss of iter 100 = [59.96952103]
```

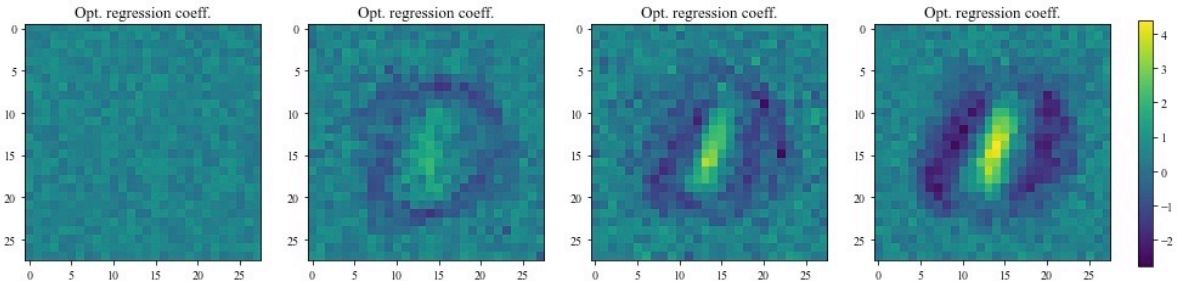
Exercise 3

i)

```
def sample_binary_MNIST(list_digits=['0','1'], full_MNIST=None, noise_ratio=0):
    # get train and test set from MNIST of given two digits
    # e.g., list_digits = ['0', '1']
    if full_MNIST is not None:
        X, y = full_MNIST
    else:
        X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
        X = X / 255.

    idx = [i for i in np.arange(len(y)) if y[i] in list_digits] # list of indices where the label y is in list_digits

    X01 = X[idx,:]
    y01 = y[idx]
    y01= y01.values
    X_train = []
    X_test = []
    y_test = [] # list of integers 0 and 1s
    y_train = [] # list of integers 0 and 1s

    for i in np.arange(X01.shape[0]):
        # for each example i, make it into train set with probabiliy 0.8 and into test set otherwise
        U = np.random.rand() # Uniform([0,1]) variable
        label = 0
        if y01[i] == str(list_digits[1]):
            label = 1

        if U<0.8:
            if noise_ratio != 0:
                for j in np.arange(X01.shape[1]):
                    R = np.random.rand() # ----Uniform([0,1]) variable which decides to add noise or not
                    V = np.random.rand() # ----uniform([0,1]) variable that will be added to pixels
                    if R < noise_ratio:
                        X01[i,j] += V
            X_train.append(X01[i,:])
            y_train.append(label)
        else:
            X_test.append(X01[i,:])
            y_test.append(label)
```

ii) noise ratio: 0.1

MNIST Binary Classification by LR for 0 vs. 1



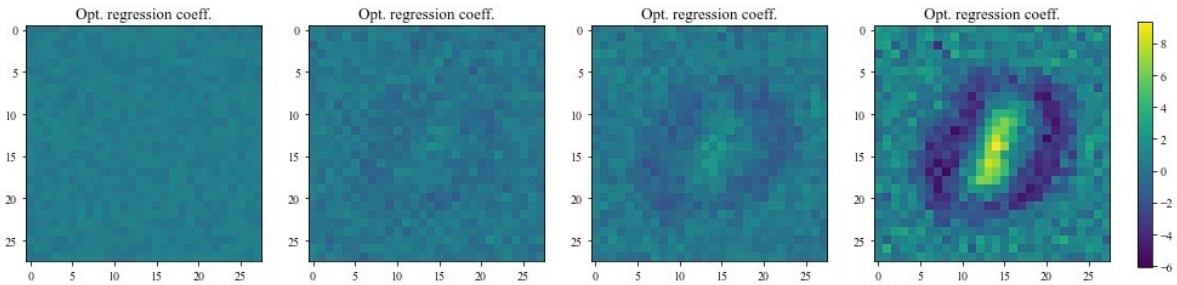
Accuracy = 0.523
Sensitivity = 0.0
Specificity = 1.0
Precision = 0.523
Fall_out = 1.0
Miss_rate = 0.0
train size = 1

Accuracy = 0.967
Sensitivity = 0.93
Specificity = 1.0
Precision = 0.94
Fall_out = 0.07
Miss_rate = 0.0
train size = 10

Accuracy = 0.997
Sensitivity = 0.998
Specificity = 0.996
Precision = 0.998
Fall_out = 0.002
Miss_rate = 0.004
train size = 30

Accuracy = 0.996
Sensitivity = 0.994
Specificity = 0.998
Precision = 0.995
Fall_out = 0.006
Miss_rate = 0.002
train size = 100

noise ratio: 0.5



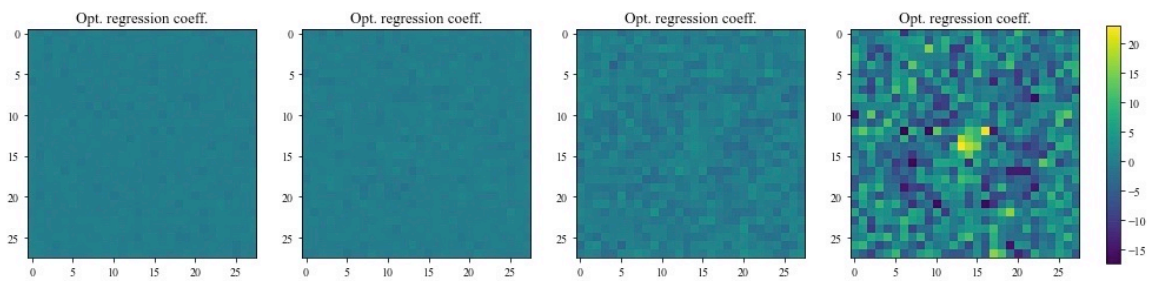
Accuracy = 0.536
Sensitivity = 0.0
Specificity = 1.0
Precision = 0.536
Fall_out = 1.0
Miss_rate = 0.0
train size = 1

Accuracy = 0.995
Sensitivity = 0.999
Specificity = 0.991
Precision = 0.999
Fall_out = 0.001
Miss_rate = 0.009
train size = 10

Accuracy = 0.994
Sensitivity = 1.0
Specificity = 0.988
Precision = 1.0
Fall_out = 0.0
Miss_rate = 0.012
train size = 30

Accuracy = 0.997
Sensitivity = 0.998
Specificity = 0.996
Precision = 0.998
Fall_out = 0.002
Miss_rate = 0.004
train size = 100

noise ratio: 0.9



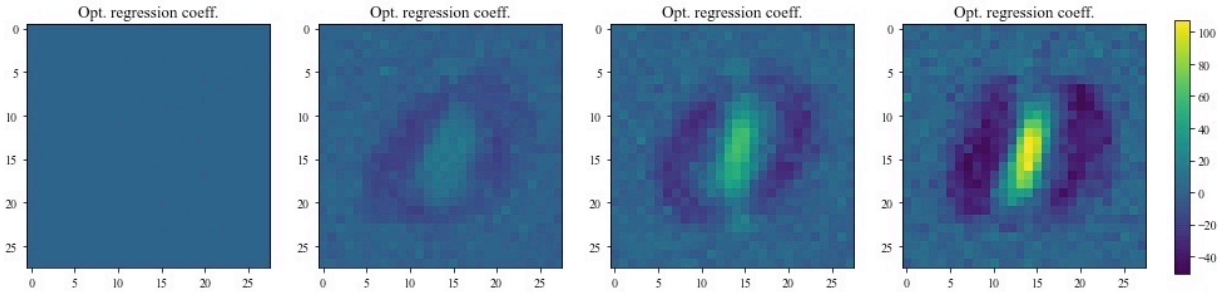
Accuracy = 0.377
Sensitivity = 0.181
Specificity = 0.546
Precision = 0.435
Fall_out = 0.819
Miss_rate = 0.454
train size = 1

Accuracy = 0.515
Sensitivity = 0.046
Specificity = 0.921
Precision = 0.527
Fall_out = 0.954
Miss_rate = 0.079
train size = 10

Accuracy = 0.994
Sensitivity = 0.997
Specificity = 0.99
Precision = 0.997
Fall_out = 0.003
Miss_rate = 0.01
train size = 30

Accuracy = 0.996
Sensitivity = 0.997
Specificity = 0.995
Precision = 0.997
Fall_out = 0.003
Miss_rate = 0.005
train size = 100

iii) noise ratio : 0.1



Accuracy = 0.538
Sensitivity = 0.0
Specificity = 1.0
Precision = 0.538
Fall_out = 1.0
Miss_rate = 0.0
train size = 1

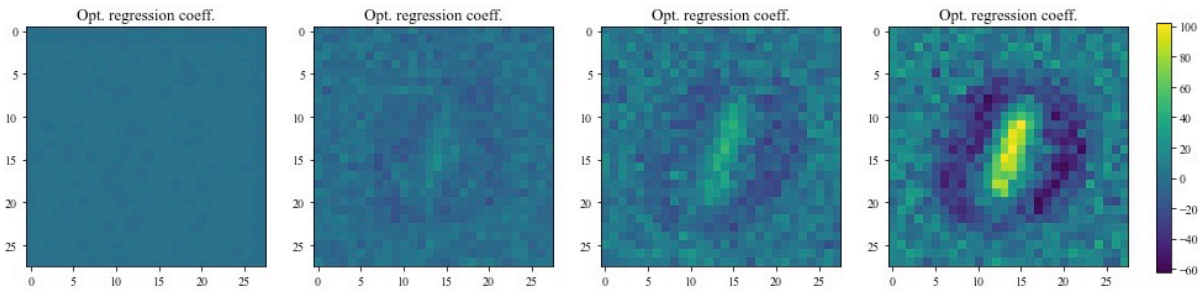
Accuracy = 0.857
Sensitivity = 1.0
Specificity = 0.734
Precision = 1.0
Fall_out = 0.0
Miss_rate = 0.266
train size = 10

Accuracy = 0.991
Sensitivity = 0.982
Specificity = 0.999
Precision = 0.984
Fall_out = 0.018
Miss_rate = 0.001
train size = 30

Accuracy = 0.998
Sensitivity = 0.999
Specificity = 0.997
Precision = 0.999
Fall_out = 0.001
Miss_rate = 0.003
train size = 100

noise ratio : 0.5

MNIST Binary Classification by PR for 0 vs. 1



Accuracy = 0.538
Sensitivity = 0.006
Specificity = 0.986
Precision = 0.54
Fall_out = 0.994
Miss_rate = 0.014
train size = 1

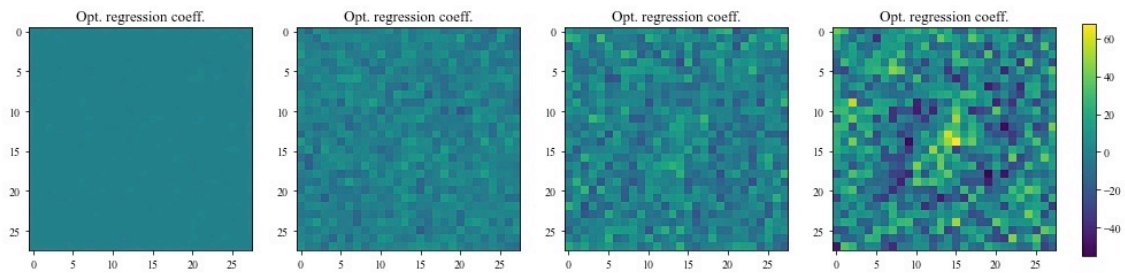
Accuracy = 0.694
Sensitivity = 1.0
Specificity = 0.435
Precision = 1.0
Fall_out = 0.0
Miss_rate = 0.565
train size = 10

Accuracy = 0.993
Sensitivity = 0.99
Specificity = 0.996
Precision = 0.992
Fall_out = 0.01
Miss_rate = 0.004
train size = 30

Accuracy = 0.995
Sensitivity = 0.993
Specificity = 0.997
Precision = 0.994
Fall_out = 0.007
Miss_rate = 0.003
train size = 100

noise ratio: 0.9

MNIST Binary Classification by PR for 0 vs. 1



Accuracy = 0.46
Sensitivity = 1.0
Specificity = 0.0
Precision = nan
Fall_out = 0.0
Miss_rate = 1.0
train size = 1

Accuracy = 0.897
Sensitivity = 0.945
Specificity = 0.857
Precision = 0.948
Fall_out = 0.055
Miss_rate = 0.143
train size = 10

Accuracy = 0.989
Sensitivity = 0.991
Specificity = 0.988
Precision = 0.992
Fall_out = 0.009
Miss_rate = 0.012
train size = 30

Accuracy = 0.994
Sensitivity = 0.996
Specificity = 0.992
Precision = 0.996
Fall_out = 0.004
Miss_rate = 0.008
train size = 100

iii) For the most part, both ways of the regression models seemed similar in robustness. For the most part, both preformed well with 0.9 noise ratio as long as the trainsize was atleast 30 it most case. I was supprised to see a 0.4 accuracy rate with both models when we had 0.9 noise, which shows how efficient both of these methods. Even though it is difficult to see the '0' and '1' in the 0.9 noise ratio graphs, our program still does a great job!