CS 189: Introduction to Machine Learning

Fall 2017

•

Homework 9

•

Solutions by

Jinhong Du

3033483677

Question 1

(a)

Jinhong Du
jaydu@berkeley.edu

(b)

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up. Jinhong Du

Question 2

(a)

For any sample x_i, suppose that x_i is in class 1, then

$$R(f(x_i)|x_i) = \sum_{j=1}^{c} L(f(x_i), j)\mathbb{P}(y_i = j|x_i)$$

If we make the first decision (a),

$$\begin{aligned} R(f(x_i)|x_i) &= \sum_{j=2}^{c} L(f(x_i), j)\mathbb{P}(y_i = j|x_i) \\ &= \lambda_s[1 - \mathbb{P}(y_i = 1|x_i)] \\ &\leqslant \lambda_s \frac{\lambda_r}{\lambda_s} \\ &= \lambda_r \end{aligned}$$

In such condition, $\lambda_r < \lambda_s$, so $R(f(x_i)|x_i) \leqslant \lambda_r < \lambda_s$. For other policies, the risk must bigger than $\lambda_s[1 - \mathbb{P}(y_i = j|x_i)] \geqslant \lambda_s[1 - \mathbb{P}(y_i = 1|x_i)]$, $\forall\, j \in \{1, 2, \cdots, c\}$. So the policy in such condition minimize the risk.
If we make the secend decision (b),

$$\begin{aligned} R(f(x_i)|x_i) &= \sum_{j=1}^{c} L(f(x_i), j)\mathbb{P}(y_i = j|x_i) \\ &= \lambda_s \end{aligned}$$

It is the same for all policies.
Therefore, the policy obtains the minimum risk.

(b)

If $\lambda_r = 0$, in both decisions,

$$R(f(x_i)|x_i) \equiv \lambda_s$$

Since $\lambda_r = 0$ means that the doubt sample won't give contribution to risk function. So the risk function only depends on the loss incurred for making a misclassication.
If $\lambda_r > \lambda_s$, then

$$\mathbb{P}(Y = i|x) < 1 - \frac{\lambda_r}{\lambda_s}$$

therefore

$$R(f(x_i)|x_i) \equiv \lambda_s$$

Since intuitively, the loss incurred for choosing doubt should be less than or equals the loss for making a misclassification. Otherwise, $\lambda_r > \lambda_s$ means that choosing wrong is better than choosing doubt.

Question 3

(a)

$$\mathbb{P}(X|L=i) = \frac{1}{\sqrt{2\pi}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu_i)^T\Sigma^{-1}(X-\mu_i)}$$

(1) MLE

The decision boundary is

$$\mathbb{P}(X|L=1) = \mathbb{P}(X|L=2)$$

i.e.

$$-\frac{1}{2}(X-\mu_1)^T\Sigma^{-1}(X-\mu_1) = -\frac{1}{2}(X-\mu_2)^T\Sigma^{-1}(X-\mu_2)$$

i.e.

$$f_{MLE}(X) = (\mu_1-\mu_2)^T\Sigma^{-1}X - \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2) = 0$$

The decision rule is

$$\hat{L}_{MLE}(X) = \begin{cases} 1 & ,[\Sigma^{-1}(\mu_1-\mu_2)]^TX > \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2) \\ 2 & ,[\Sigma^{-1}(\mu_1-\mu_2)]^TX \leqslant \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2) \end{cases}$$

(2) MAP

$\because$

$$\begin{aligned}
\mathbb{P}(L=1|X) &= \frac{\mathbb{P}(X|L=1)\mathbb{P}(L=1)}{\mathbb{P}(X|L=1)\mathbb{P}(L=1)+\mathbb{P}(X|L=2)\mathbb{P}(L=2)} \\
&= \frac{\mathbb{P}(X|L=1)\pi_1}{\mathbb{P}(X|L=1)\pi_1+\mathbb{P}(X|L=2)\pi_2} \\
&= \frac{1}{1+\frac{\mathbb{P}(X|L=2)\pi_2}{\mathbb{P}(X|L=1)\pi_1}} \\
&= \frac{1}{1+e^{\ln\mathbb{P}(X|L=2)-\ln\mathbb{P}(X|L=1)+\ln\pi_2-\ln\pi_1}} \\
&= \frac{1}{1+e^{(\mu_2-\mu_1)^T\Sigma^{-1}X-\frac{1}{2}(\mu_2^T\Sigma^{-1}\mu_2-\mu_1^T\Sigma^{-1}\mu_1)+\ln\pi_2-\ln\pi_1}}
\end{aligned}$$

$\therefore$ the MAP decision boundary is

$$f_{MAP}(X) = (\mu_2-\mu_1)^T\Sigma^{-1}X - \frac{1}{2}(\mu_2^T\Sigma^{-1}\mu_2 - \mu_1^T\Sigma^{-1}\mu_1) + \ln\pi_2 - \ln\pi_1 = 0$$

The MAP decision rule is

$$\hat{L}_{MAP}(X) = \begin{cases} 1 & ,[\Sigma^{-1}(\mu_1-\mu_2)]^TX > \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2) + \ln\pi_2 - \ln\pi_1 \\ 2 & ,[\Sigma^{-1}(\mu_1-\mu_2)]^TX \leqslant \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2) + \ln\pi_2 - \ln\pi_1 \end{cases}$$

When $\pi_1 = \pi_2 = \frac{1}{2}$, then these two decision rules are the same.

(b)

$$\begin{aligned}
\mathbb{E}X &= \pi_1\mu_1 + \pi_2\mu_2 \\
&= \frac{\mu_1+\mu_2}{2} \\
\mathbb{E}[XX^T|\text{class } 1] &= Var[X|\text{class } 1] + (\mathbb{E}[X|\text{class } 1])(\mathbb{E}[X|\text{class } 1])^T \\
&= \Sigma + \mu_1\mu_1^T \\
\mathbb{E}[XX^T|\text{class } 2] &= Var[X|\text{class } 2] + (\mathbb{E}[X|\text{class } 2])(\mathbb{E}[X|\text{class } 2])^T \\
&= \Sigma + \mu_2\mu_2^T \\
\Sigma_{XX} &= \mathbb{E}[(X-\mathbb{E}X)(X-\mathbb{E}X)^T] \\
&= \mathbb{E}[XX^T] - (\mathbb{E}X)(\mathbb{E}X)^T \\
&= \pi_1\mathbb{E}[XX^T|\text{class } 1] + \pi_2\mathbb{E}[XX^T|\text{class } 2] - \frac{1}{4}(\mu_1+\mu_2)(\mu_1+\mu_2)^T \\
&= \Sigma + \frac{1}{2}\mu_1\mu_1^T + \frac{1}{2}\mu_2\mu_2^T - \frac{1}{4}(\mu_1+\mu_2)(\mu_1+\mu_2)^T \\
&= \Sigma - \frac{1}{4}(\mu_1-\mu_2)(\mu_1-\mu_2)^T \\
\mathbb{E}Y &= \pi_1\begin{pmatrix}1\\0\end{pmatrix} + \pi_2\begin{pmatrix}0\\1\end{pmatrix} \\
&= \begin{pmatrix}\frac{1}{2}\\\frac{1}{2}\end{pmatrix} \\
\Sigma_{XY} &= \mathbb{E}[(X-\mathbb{E}X)(Y-\mathbb{E}Y)^T] \\
&= \pi_1\mathbb{E}[(X-\mathbb{E}X)(Y-\mathbb{E}Y)^T|\text{class } 1] + \pi_2\mathbb{E}[(X-\mathbb{E}X)(Y-\mathbb{E}Y)^T|\text{class } 2] \\
&= \pi_1\begin{pmatrix}\frac{\mu_1-\mu_2}{2}\end{pmatrix}\begin{pmatrix}\frac{1}{2} & -\frac{1}{2}\end{pmatrix} + \pi_2\begin{pmatrix}\frac{\mu_2-\mu_1}{2}\end{pmatrix}\begin{pmatrix}-\frac{1}{2} & \frac{1}{2}\end{pmatrix} \\
&= \frac{1}{4}\begin{pmatrix}\mu_1-\mu_2 & \mu_2-\mu_1\end{pmatrix} \\
\Sigma_{YY} &= \mathbb{E}[(Y-\mathbb{E}Y)(Y-\mathbb{E}Y)^T] \\
&= \pi_1\begin{pmatrix}\frac{1}{2}\\-\frac{1}{2}\end{pmatrix}\begin{pmatrix}\frac{1}{2} & -\frac{1}{2}\end{pmatrix} + \pi_2\begin{pmatrix}-\frac{1}{2}\\\frac{1}{2}\end{pmatrix}\begin{pmatrix}-\frac{1}{2} & \frac{1}{2}\end{pmatrix} \\
&= \begin{pmatrix}\frac{1}{4} & -\frac{1}{4}\\-\frac{1}{4} & \frac{1}{4}\end{pmatrix}
\end{aligned}$$

(c)

$$
\begin{aligned}
\rho &= \max_{\substack{u\in\mathbb{R}^d \\ v\in\mathbb{R}^2}} \rho(u^T X, v^T Y) \\
&= \max_{\substack{u\in\mathbb{R}^d \\ v\in\mathbb{R}^2}} \frac{Cov(u^T X Y^T v)}{\sqrt{Var(u^T X) Var(v^T Y)}} \\
&= \max_{\substack{u\in\mathbb{R}^d \\ v\in\mathbb{R}^2}} \frac{u^T \Sigma_{XY} v}{\sqrt{u^T \Sigma_{XX} u}\sqrt{v^T \Sigma_{YY} v}} \\
&= \max_{\substack{u\in\mathbb{R}^d \\ v\in\mathbb{R}^2}} \frac{u^T(\mu_1-\mu_2)(v_1-v_2)}{2\sqrt{u^T\Sigma_{XX}u}|v_1-v_2|} \\
&= \max_{u\in\mathbb{R}^d} \frac{u^T(\mu_1-\mu_2)}{2\sqrt{u^T\Sigma_{XX}u}}
\end{aligned}
$$

$$
\begin{aligned}
&\xlongequal{u'=\Sigma_{XX}^{\frac{1}{2}}u} \max_{u'\in\mathbb{R}^d} \frac{u'^T\Sigma_{XX}^{-\frac{1}{2}}(\mu_1-\mu_2)}{2\sqrt{u'^T u'}} \\
&= \max_{\substack{u_2\in\mathbb{R}^d \\ \|u_2\|=1}} \frac{1}{2} u_2^T \Sigma_{XX}^{-\frac{1}{2}}(\mu_1-\mu_2)
\end{aligned}
$$

Let $v' = \Sigma_{XX}^{-\frac{1}{2}}(\mu_1-\mu_2)$, then by Cauchy Schwarz iequality,

$$
\begin{aligned}
(u_2^T v')^2 &\leqslant \|u_2\|_2^2 \|v'\|_2^2 \\
&= (\mu_1-\mu_2)^T \Sigma_{XX}^{-1}(\mu_1-\mu_2)
\end{aligned}
$$

the equation holds only when $u_2 = \pm\frac{v'}{\|v'\|} = \pm\frac{\Sigma_{XX}^{-\frac{1}{2}}(\mu_1-\mu_2)}{\|v'\|}$, where c is a constant. i.e. $\rho(u^T X, v^T Y)$ is maximized when

$$
\begin{aligned}
u &= \Sigma_{XX}^{-\frac{1}{2}} u' \\
&= c\Sigma_{XX}^{-1}(\mu_1-\mu_2) \\
&= c(\Sigma - \frac{1}{4}(\mu_1-\mu_2)(\mu_1-\mu_2)^T)^{-1}(\mu_1-\mu_2) \\
&= c(\Sigma^{-1} - \frac{\frac{1}{4}\Sigma^{-1}(\mu_1-\mu_2)(\mu_1-\mu_2)^T\Sigma^{-1}}{1+\frac{1}{4}(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)})(\mu_1-\mu_2) \\
&= c\Sigma^{-1}(\mu_1-\mu_2) - c\frac{\frac{1}{4}\Sigma^{-1}(\mu_1-\mu_2)\left[(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)\right]}{1+\frac{1}{4}(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)} \\
&= \frac{c}{1+\frac{1}{4}(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)}\Sigma^{-1}(\mu_1-\mu_2)
\end{aligned}
$$

$\therefore$ u^* is proportional to $\Sigma^{-1}(\mu_1-\mu_2)$

u^* is proportional to the coefficient of X in $f(X)$, i.e. $f(X) = du^{*T}X$ where d is a constant

(d)

$$
\begin{aligned}
\mathbb{E}X &= \pi_1\mu_1 + \pi_2\mu_2 \\
\mathbb{E}[XX^T|\text{class 1}] &= Var[X|\text{class 1}] + (\mathbb{E}[X|\text{class 1}])(\mathbb{E}[X|\text{class 1}])^T \\
&= \Sigma + \mu_1\mu_1^T \\
\mathbb{E}[XX^T|\text{class 2}] &= Var[X|\text{class 2}] + (\mathbb{E}[X|\text{class 2}])(\mathbb{E}[X|\text{class 2}])^T \\
&= \Sigma + \mu_2\mu_2^T \\
\Sigma_{XX} &= \mathbb{E}[(X-\mathbb{E}X)(X-\mathbb{E}X)^T] \\
&= \mathbb{E}[XX^T] - (\mathbb{E}X)(\mathbb{E}X)^T \\
&= \pi_1\mathbb{E}[XX^T|\text{class 1}] + \pi_2\mathbb{E}[XX^T|\text{class 2}] - (\pi_1\mu_1+\pi_2\mu_2)(\pi_1\mu_1+\pi_2\mu_2)^T \\
&= \Sigma + \pi_1\mu_1\mu_1^T + (1-\pi_1)\mu_2\mu_2^T - \pi_1^2\mu_1\mu_1^T - 2\pi_1\pi_2\mu_1\mu_2^T - \pi_2^2\mu_2\mu_2^T \\
&= \Sigma + \pi_1\pi_2(\mu_1-\mu_2)(\mu_1-\mu_2)^T \\
\mathbb{E}Y &= \pi_1\begin{pmatrix}1\\0\end{pmatrix} + \pi_2\begin{pmatrix}0\\1\end{pmatrix} \\
&= \begin{pmatrix}\pi_1\\\pi_2\end{pmatrix} \\
\Sigma_{XY} &= \mathbb{E}[(X-\mathbb{E}X)(Y-\mathbb{E}Y)^T] \\
&= \pi_1\mathbb{E}[(X-\mathbb{E}X)(Y-\mathbb{E}Y)^T|\text{class 1}] + \pi_2\mathbb{E}[(X-\mathbb{E}X)(Y-\mathbb{E}Y)^T|\text{class 2}] \\
&= \pi_1\pi_2(\mu_1-\mu_2)\begin{pmatrix}\pi_2 & -\pi_2\end{pmatrix} + \pi_1\pi_1(\mu_2-\mu_1)\begin{pmatrix}-\pi_1 & \pi_1\end{pmatrix} \\
&= \pi_1\pi_2\begin{pmatrix}\mu_1-\mu_2 & \mu_2-\mu_1\end{pmatrix} \\
\Sigma_{YY} &= \mathbb{E}[(Y-\mathbb{E}Y)(Y-\mathbb{E}Y)^T] \\
&= \pi_1\begin{pmatrix}\pi_2\\-\pi_2\end{pmatrix}\begin{pmatrix}\pi_2 & -\pi_2\end{pmatrix} + \pi_2\begin{pmatrix}-\pi_1\\\pi_1\end{pmatrix}\begin{pmatrix}-\pi_1 & \pi_1\end{pmatrix} \\
&= \begin{pmatrix}\pi_1\pi_2 & -\pi_1\pi_2\\-\pi_1\pi_2 & \pi_1\pi_2\end{pmatrix}
\end{aligned}
$$

Solution (cont.)

$$
\begin{aligned}
\rho &= \max_{\substack{u\in\mathbb{R}^d\\ v\in\mathbb{R}^2}} \rho(u^T X, v^T Y)\\
&= \max_{\substack{u\in\mathbb{R}^d\\ v\in\mathbb{R}^2}} \frac{Cov(u^T XY^T v)}{\sqrt{Var(u^T X)Var(v^T Y)}}\\
&= \max_{\substack{u\in\mathbb{R}^d\\ v\in\mathbb{R}^2}} \frac{u^T\Sigma_{XY}v}{\sqrt{u^T\Sigma_{XX}uv^T\Sigma_{YY}v}}\\
&= \max_{\substack{u\in\mathbb{R}^d\\ v\in\mathbb{R}^2}} \frac{\sqrt{\pi_1\pi_2}u^T(\mu_1-\mu_2)(v_1-v_2)}{\sqrt{u^T\Sigma^{-\frac{1}{2}}u}|v_1-v_2|}\\
&= \max_{u\in\mathbb{R}^d} \frac{\sqrt{\pi_1\pi_2}u^T(\mu_1-\mu_2)}{\sqrt{u^T\Sigma_{XX}u}}\\
&\overset{u'=\Sigma^{\frac{1}{2}}u}{=\!=\!=} \max_{u\in\mathbb{R}^d} \frac{\sqrt{\pi_1\pi_2}u'^T\Sigma_{XX}^{-\frac{1}{2}}(\mu_1-\mu_2)}{\|u'\|}\\
&= \max_{\substack{u_2\in\mathbb{R}^d\\ \|u_2\|=1}} \sqrt{\pi_1\pi_2}u_2^T\Sigma_{XX}^{-\frac{1}{2}}(\mu_1-\mu_2)
\end{aligned}
$$

Let $v' = \Sigma_{XX}^{-\frac{1}{2}}(\mu_1-\mu_2)$, then by Cauchy Schwarz iequality,

$$
\begin{aligned}
(u_2^T v')^2 &\leqslant \|u_2\|_2^2\|v'\|_2^2\\
&= (\mu_1-\mu_2)^T\Sigma_{XX}^{-1}(\mu_1-\mu_2)
\end{aligned}
$$

the equation holds only when $u_2 = \pm\frac{v'}{\|v'\|} = \pm\frac{\Sigma_{XX}^{-\frac{1}{2}}(\mu_1-\mu_2)}{\|v'\|}$, where c is a constant. i.e. $\rho(u^T X, v^T Y)$ is maximized when

$$
\begin{aligned}
u &= \Sigma_{XX}^{-\frac{1}{2}}u'\\
&= c\Sigma_{XX}^{-1}(\mu_1-\mu_2)\\
&= c(\Sigma-\pi_1\pi_2(\mu_1-\mu_2)(\mu_1-\mu_2)^T)^{-1}(\mu_1-\mu_2)\\
&= c(\Sigma^{-1}-\frac{\pi_1\pi_2\Sigma^{-1}(\mu_1-\mu_2)(\mu_1-\mu_2)^T\Sigma^{-1}}{1+\pi_1\pi_2(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)})(\mu_1-\mu_2)\\
&= c\Sigma^{-1}(\mu_1-\mu_2) - c\frac{\pi_1\pi_2\Sigma^{-1}(\mu_1-\mu_2)\left[(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)\right]}{1+\pi_1\pi_2(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)}\\
&= \frac{c}{1+\pi_1\pi_2(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)}\Sigma^{-1}(\mu_1-\mu_2)
\end{aligned}
$$

$\therefore$ u^* is proportional to $\Sigma^{-1}(\mu_1-\mu_2)$

u^* is proportional to the coefficient of X in $f(X)$, i.e. $f(X) = du^{*T}X$ where d is a constant

(e)

(1) Compute the sample covariance matrix

$$\hat{\mu}_l = \frac{1}{n} \sum_{x_i \in \text{class } l} x_i$$

$$\hat{\Sigma} = \sum_{l=1}^{2} \frac{\pi_l}{|\text{class } l|} \sum_{x_i \in \text{class } l} (x_i - \hat{\mu}_l)(x_i - \hat{\mu}_l)^T$$

Suppose that $\hat{\mu}_1 < \hat{\mu}_2$ and $\mu_1 + \mu_2 \geqslant 0$.

(2) Subtract the mean

$$X'_{train} = X_{train} - \frac{\hat{\mu}_2 - \hat{\mu}_1}{2}$$

(3) Estimate u^* by

$$\hat{\Sigma}^{-1}(\frac{\hat{\mu}_2 + \hat{\mu}_1}{2})$$

(4) Project X_{test} into the subspace of the range of X that contributes most to predicting Y

$$u^{*T} X_{test}$$

(5) Predict

If $u^{*T} X_{test} < 0$ then predict Y to be class 1; otherwise predict Y to be class 2.

Question 4

(a)

```python
class Generative_Model(object):
    def __init__(self,
                 num_gd_replicates = 20,
                 total_step_count = 1000,
                 step_size = lambda i: 1/(1+i),
                 err = 1e-5):
        self.num_gd_replicates = num_gd_replicates
        self.total_step_count = total_step_count
        self.step_size = step_size
        self.err = err


    def gradient1(self, obj_loc, d, sen_loc):
        g = np.zeros_like(sen_loc)
        for i in range(len(g)):
            g[i] = -np.dot(((np.linalg.norm(obj_loc-sen_loc[i],
              axis=1)-d[:,i])/np.linalg.norm(obj_loc-sen_loc[i],
              axis=1)).T,(obj_loc-sen_loc[i]))
        if np.any(np.isnan(g)):
            return 0
        else:
            return g

    def gradient_descend_step1(self,obj_loc, d, sen_loc, step_count):
        sen_loc = sen_loc - self.step_size(step_count) *
          self.gradient1(obj_loc, d, sen_loc)
        return sen_loc

    def gradient_descend1(self,obj_loc, d, sen_loc):
        positions = [np.array(sen_loc)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step1(obj_loc, d,
              positions[-1], k)
            if np.linalg.norm(positions[-1]-new)<self.err:
                break
            else:
                positions.append(new)
        return np.array(positions)

    def gradient2(self,obj_loc, d, sen_loc):
```

Solution (cont.)

```
        g = np.dot((((np.linalg.norm(obj_loc-sen_loc,axis=1)-d)
          /np.linalg.norm(obj_loc-sen_loc,axis=1))).T,
          (obj_loc-sen_loc))
        if np.any(np.isnan(g)):
            return 0
        else:
            return g/len(g)

    def gradient_descend_step2(self,obj_loc, d, sen_loc, step_count):
        """Computes the new point after the update at x."""
        obj_loc = obj_loc - self.step_size(step_count) *
          self.gradient2(obj_loc, d, sen_loc)
        return obj_loc

    def gradient_descend2(self,obj_loc, d, sen_loc):
        positions = [np.array(obj_loc)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step2(
              positions[-1], d, sen_loc, k)
            if np.max(np.linalg.norm(positions[-1]-new,axis=1))
              <self.err:
                break
            else:
                positions.append(new)
        return positions

    def evaluate(self,obj_loc,d):
        est_obj_loc = np.zeros_like(obj_loc)
        for j in range(len(est_obj_loc)):
            p = []
            for i in range(self.num_gd_replicates):
                initial_position = np.random.randn(1,2)*100
                optimal_solutions = self.gradient_descend2(
                  initial_position, d[j], self.est_sen_loc)
                p += [optimal_solutions[-1]]
            p = np.array(p)
            est_obj_loc[j] = np.unique(np.round(p,2),axis=0)[
              np.argmax(np.unique(np.round(p,2),axis=0,
                return_counts=True)[1])]
        error = np.mean(np.linalg.norm(est_obj_loc - obj_loc,axis=1))
        return {'MSE':error,'est_obj_loc':est_obj_loc}

    def train(self,obj_loc,sen_loc,d):
```

Solution (cont.)

```
        m,n = np.shape(sen_loc)
        p = []
        for i in range(self.num_gd_replicates):
            initial_position = np.random.randn(m,n)*100
            optimal_solutions = self.gradient_descend1(obj_loc,
                d, initial_position)
            p += [optimal_solutions[-1]]
        p = np.array(p)
        self.est_sen_loc = np.unique(np.round(p,2),axis=0)[
          np.argmax(np.unique(np.round(p,2),axis=0,
            return_counts=True)[1])]
        self.train_error = self.evaluate(obj_loc,d)['MSE']

    def test(self,obj_loc,d):
        self.test_error = self.evaluate(obj_loc,d)['MSE']
        return self.test_error

class Linear_Model(object):
    def __init__(self):
        pass
    def train(self,X,Y,lamb=0):
        m,n = np.shape(X)
        self.A = np.linalg.solve(X.T.dot(X)+lamb*
          np.identity(n),X.T.dot(Y))
        yhat = X.dot(self.A)
        self.train_error = np.mean(np.linalg.norm(yhat-Y,axis=1))

    def test(self,X,Y):
        m,n = np.shape(X)
        yhat = X.dot(self.A)
        self.test_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
        return {'A':self.A,'error':self.test_error}

import itertools
class Second_Model(object):
    def __init__(self):
        pass
    def train(self,X,Y,lamb=0):
        m,n = np.shape(X)
        X2 = np.ones((m,1))
        X2 = np.hstack((X2,X))
        for i in range(2,3):
            for j in itertools.combinations_with_replacement(
```

Solution (cont.)

```
                np.arange(n), i):
                xn = np.ones((1,m))
                for k in j:
                    xn = xn*X[:,k]
                X2 = np.hstack((X2,xn.reshape(m,1)))
        self.A = np.linalg.solve(X2.T.dot(X2)+lamb*
          np.identity(len(X2.T)),X2.T.dot(Y))
        yhat = X2.dot(self.A)
        self.train_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
    def test(self,X,Y):
        m,n = np.shape(X)
        X2 = np.ones((m,1))
        X2 = np.hstack((X2,X))
        for i in range(2,3):
            for j in itertools.combinations_with_replacement(
                np.arange(n), i):
                xn = np.ones((1,m))
                for k in j:
                    xn = xn*X[:,k]
                X2 = np.hstack((X2,xn.reshape(m,1)))
        yhat = X2.dot(self.A)
        self.test_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
        return {'A':self.A,'error':self.test_error}

class Third_Model(object):
    def __init__(self):
        pass
    def train(self,X,Y,lamb=0):
        m,n = np.shape(X)
        X2 = np.ones((m,1))
        X2 = np.hstack((X2,X))
        for i in range(2,4):
            for j in itertools.combinations_with_replacement(
                np.arange(n), i):
                xn = np.ones((1,m))
                for k in j:
                    xn = xn*X[:,k]
                X2 = np.hstack((X2,xn.reshape(m,1)))
        self.A = np.linalg.solve(X2.T.dot(X2)+lamb*
          np.identity(len(X2.T)),X2.T.dot(Y))
        yhat = X2.dot(self.A)
        self.train_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
    def test(self,X,Y):
```

Solution (cont.)

```python
        m,n = np.shape(X)
        X2 = np.ones((m,1))
        X2 = np.hstack((X2,X))
        for i in range(2,4):
            for j in itertools.combinations_with_replacement(
              np.arange(n), i):
                xn = np.ones((1,m))
                for k in j:
                    xn = xn*X[:,k]
                X2 = np.hstack((X2,xn.reshape(m,1)))
        yhat = X2.dot(self.A)
        self.test_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
        return {'A':self.A,'error':self.test_error}

# Gradient descent optimization
# The learning rate is specified by eta
class GDOptimizer(object):
    def __init__(self, eta):
        self.eta = eta

    def initialize(self, layers):
        pass

    def update(self, layers, g, a):
        m = a[0].shape[1]
        for layer, curGrad, curA in zip(layers, g, a):
            update = np.dot(curGrad,curA.T)
            updateB = np.sum(curGrad,1).reshape(layer.b.shape)
            layer.updateWeights(-self.eta/m *
              np.dot(curGrad,curA.T))
            layer.updateBias(-self.eta/m *
              np.sum(curGrad,1).reshape(layer.b.shape))

# Cost function used to compute prediction errors
class QuadraticCost(object):
    @staticmethod
    def fx(y,yp):
        return 0.5 * np.square(yp-y)

    # Derivative of the cost function with respect to yp
    @staticmethod
    def dx(y,yp):
        return y - yp
```

Solution (cont.)

```

    # Sigmoid function fully implemented as an example
    class SigmoidActivation(object):
        @staticmethod
        def fx(z):
            return 1 / (1 + np.exp(-z))

        @staticmethod
        def dx(z):
            return SigmoidActivation.fx(z) * (1 -
              SigmoidActivation.fx(z))

    # Hyperbolic tangent function
    class TanhActivation(object):

        # Compute tanh for each element in the input z
        @staticmethod
        def fx(z):
            return np.tanh(z)

        # Compute the derivative of the tanh function with respect to z
        @staticmethod
        def dx(z):
            return 1 - np.square(np.tanh(z))

    # Rectified linear unit
    class ReLUActivation(object):
        @staticmethod
        def fx(z):
            return np.maximum(0,z)

        @staticmethod
        def dx(z):
            return (z>0).astype('float')

    # Linear activation
    class LinearActivation(object):
        @staticmethod
        def fx(z):
            return z

        @staticmethod
        def dx(z):
```

Solution (cont.)

```
            return np.ones(z.shape)

class DenseLayer(object):

    def __init__(self, numNodes, activation):
        self.numNodes = numNodes
        self.activation = activation

    def getNumNodes(self):
        return self.numNodes

    def initialize(self, fanIn, scale=1.0):
        s = scale * np.sqrt(6.0 / (self.numNodes + fanIn))
        self.W = np.random.normal(0, s,
                                  (self.numNodes, fanIn))
        #self.b = np.zeros((self.numNodes,1))
        self.b = np.random.uniform(-1,1,(self.numNodes,1))

    # Apply the activation function of the layer on the input z
    def a(self, z):
        return self.activation.fx(z)

    def z(self, a):
        #print('a:\n'+str(a))
        #print('Wa:\n'+str(self.W.dot(a)))
        return self.W.dot(a) + self.b

    def dx(self, z):
        return self.activation.dx(z)

    # Update the weights of the layer by adding dW to the weights
    def updateWeights(self, dW):
        self.W = self.W + dW

    # Update the bias of the layer by adding db to the bias
    def updateBias(self, db):
        self.b = self.b + db

class Model(object):

    def __init__(self, inputSize):
        self.layers = []
        self.inputSize = inputSize
```

Solution (cont.)

```

    # Add a layer to the end of the network
    def addLayer(self, layer):
        self.layers.append(layer)

    # Get the output size of the layer at the given index
    def getLayerSize(self, index):
        if index >= len(self.layers):
            return self.layers[-1].getNumNodes()
        elif index < 0:
            return self.inputSize
        else:
            return self.layers[index].getNumNodes()

    def initialize(self, cost, initializeLayers=True):
        self.cost = cost
        if initializeLayers:
            for i in range(0,len(self.layers)):
                if i == len(self.layers) - 1:
                    self.layers[i].initialize(
                      self.getLayerSize(i-1))
                else:
                    self.layers[i].initialize(
                      self.getLayerSize(i-1))

    def evaluate(self, x):
        curA = x.T
        a = [curA]
        z = []
        for layer in self.layers:
            z.append(layer.z(curA))
            curA = layer.a(z[-1])
            a.append(curA)
        yp = a.pop()
        return yp, a, z

    def predict(self, a):
        a,_,_ = self.evaluate(a)
        return a.T

    def train(self, x, y, numEpochs, optimizer):

        # Initialize some stuff
```

Solution (cont.)

```
        n = x.shape[0]
        hist = []
        optimizer.initialize(self.layers)

        # Run for the specified number of epochs
        for epoch in range(0,numEpochs):

            yp, a, z = self.evaluate(x)

            # Compute the error
            C = self.cost.fx(yp,y.T)
            d = self.cost.dx(yp,y.T)
            grad = []

            # Backpropogate the error
            idx = len(self.layers)
            for layer, curZ in zip(reversed(self.layers),
              reversed(z)):
                idx = idx - 1
                grad.insert(0,np.multiply(d,layer.dx(curZ)))
                d = np.dot(layer.W.T,grad[0])

            # Update the errors
            optimizer.update(self.layers, grad, a)

            # Compute the error at the end of the epoch
            yh = self.predict(x)
            C = self.cost.fx(yh,y)
            C = np.mean(C)
            hist.append(C)
        return hist

    def trainBatch(self, x, y, batchSize, numEpochs, optimizer):

        x = x.copy()
        y = y.copy()
        hist = []
        n = x.shape[0]

        for epoch in np.arange(0,numEpochs):

            # Shuffle the data
            r = np.arange(0,x.shape[0])
```

Solution (cont.)

```
            x = x[r,:]
            y = y[r,:]
            e = []

            # Split the data in chunks and run SGD
            for i in range(0,n,batchSize):
                end = min(i+batchSize,n)
                batchX = x[i:end,:]
                batchY = y[i:end,:]
                e += self.train(batchX, batchY, 1, optimizer)
            hist.append(np.mean(e))

        return hist
```

(b)

(1) Generating model:

Strengths: Good to generalize to shifted data.

Weaknesses: Need lots of calculation.

(2) Linear model:

Strengths: Simple to model and compute.

Weaknesses: Underfitting for the original data.

(3) Second-Order Polynomial Regression Model

Strengths: Better than Linear Model. Have small testing error in shifted data.

Weaknesses: Bad to be generlized to shifted data.

(4) Third-Order Polynomial Regression Model

Strengths: More complicated to represent the data.

Weaknesses: Overfitting. Causing large errors when testing. Bad to be generlized to shifted data.

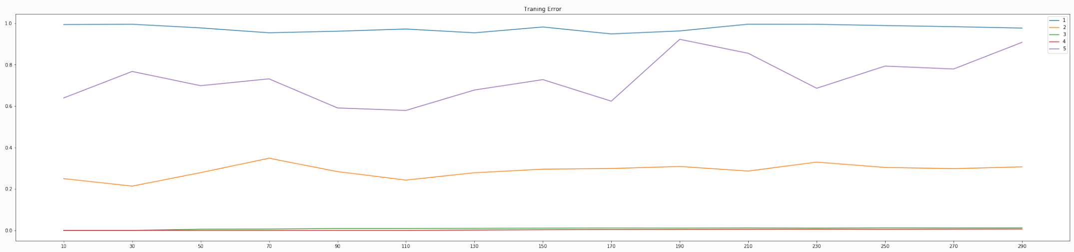
(5) Neural Network Model

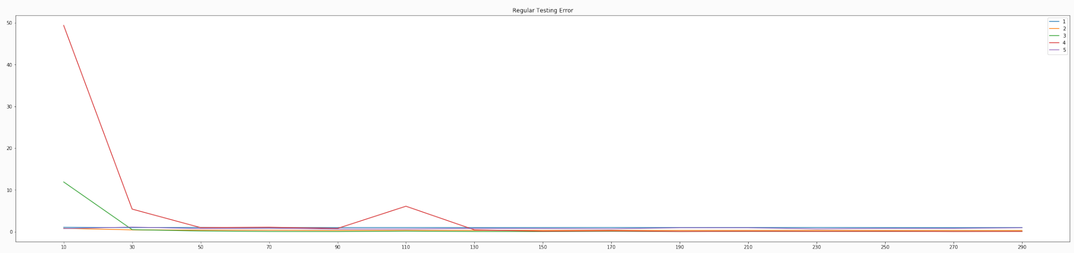
Strengths: Easy to chain. Good to generalize to shifted data.

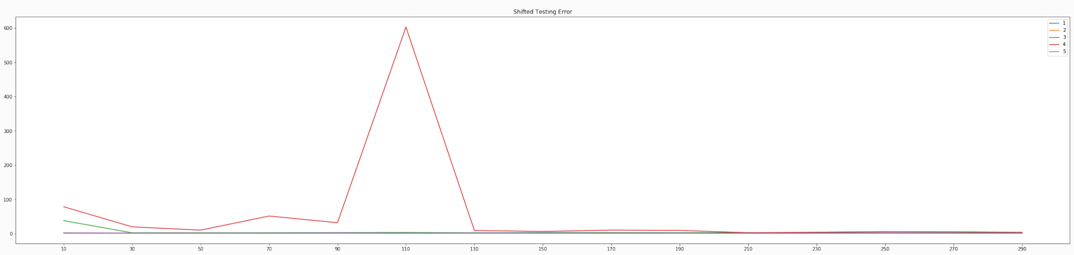
Weaknesses: Not stable.

Error Plot

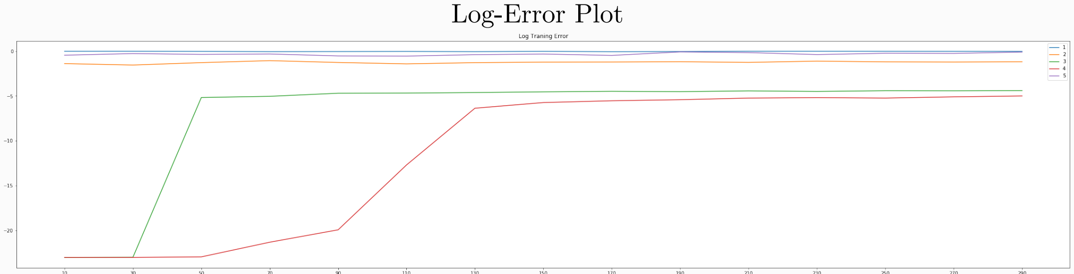
Solution (cont.)

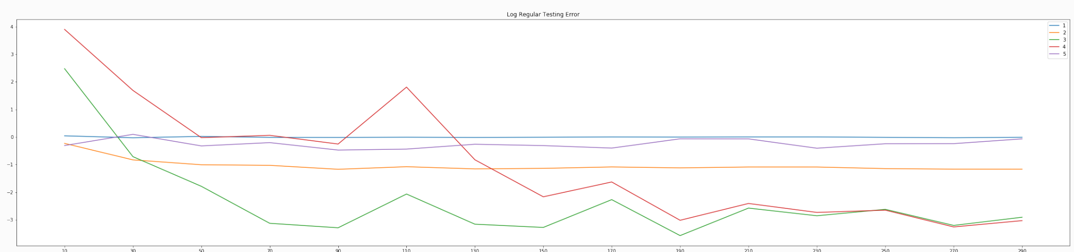


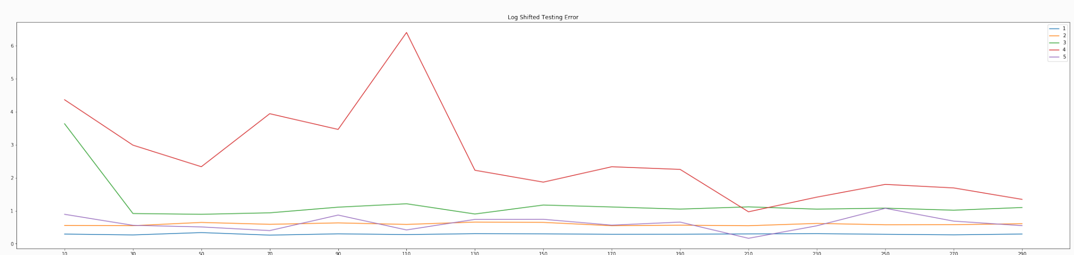




Log-Error Plot

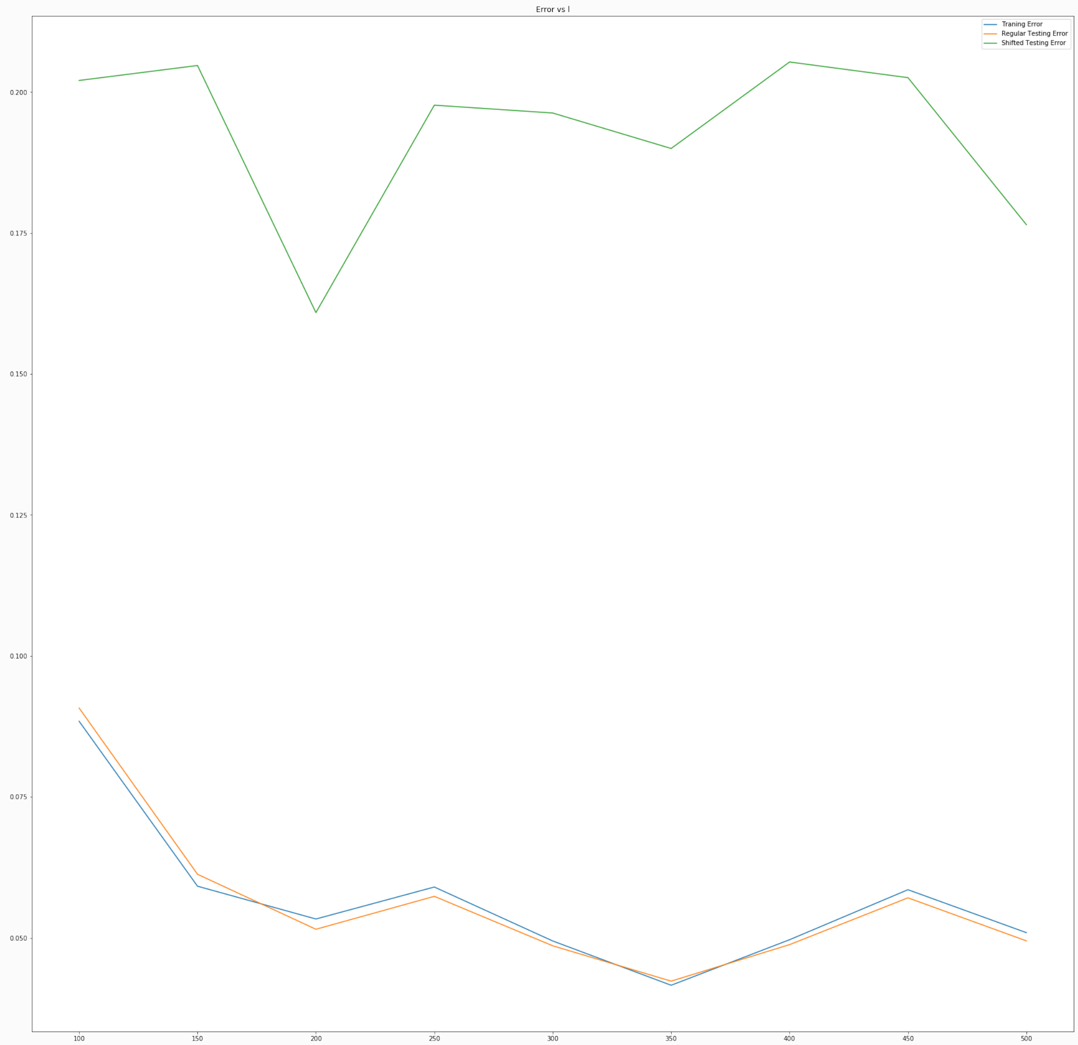






(c)

We can see that shifted testing error is much bigger than others. Shifted testing data is the true testing data since the model never sees such data. So we should choose $l = 150$ such that the shifted testing error is minimized.




```python
l = np.arange(100,550,50)
n = len(l)
resultc = np.zeros((3,n))

for i in range(n):
    modelc1 = Model(xtrain.shape[1])
    modelc1.addLayer(DenseLayer(l[i],ReLUActivation()))
    modelc1.addLayer(DenseLayer(l[i],ReLUActivation()))
    modelc1.addLayer(DenseLayer(2,LinearActivation()))
    modelc1.initialize(QuadraticCost())
    hist = modelc1.train(xtrain,ytrain,500,GDOptimizer(eta=0.0001))
    yHat = modelc1.predict(xtrain)
```

Solution (cont.)

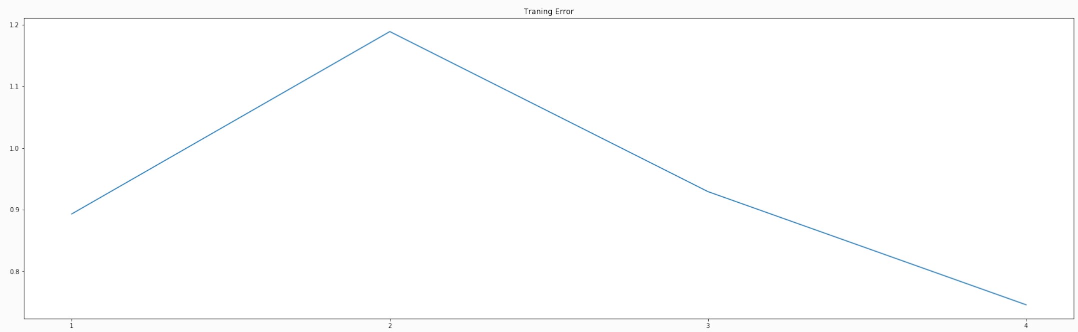
```
        resultc[0,i] = np.mean(np.linalg.norm(yHat - ytrain,axis=1))
        resultc[1,i] = np.mean(np.linalg.norm(modelc1.predict(xtest1)
         - ytest1,axis=1))
        resultc[2,i] = np.mean(np.linalg.norm(modelc1.predict(xtest2)
         - ytest2,axis=1))

plt.figure(figsize=(30,30))
for i in range(3):
        plt.plot(np.arange(9),resultc[i,:],label=s[i])
plt.title('Error vs l')
plt.legend()
plt.xticks(np.arange(9),l)
plt.show()
```

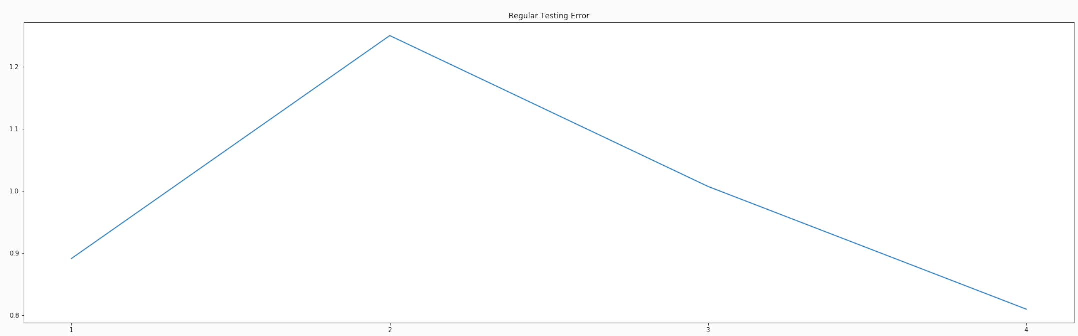
(d)

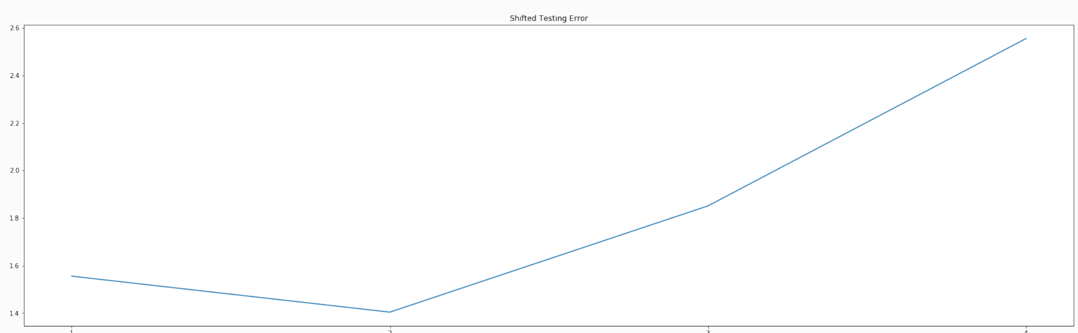
$$n = 7l + (k-1)l^2 + 2l = 9l + (k-1)l^2$$

To get $n = 10000$, we have $l = \begin{cases} \dfrac{-9+\sqrt{81+10000(k-1)}}{2(k-1)} & ,k \geqslant 1 \\ \frac{10000}{9} & ,k = 1 \end{cases}$

The best choice for k is 2. It is because that when $k \leqslant 2$ the traning error and regular testing error increase while the shifted testing error decreases as k increases. When $k > 2$ the traning error and regular testing error decrease while the shifted testing error increases as k increases. So when $k < 2$, the model may be underfitting and when $k > 2$, the model may be overfitting. Therefore the best choice should be $k = 2$.








```
k2 = [i for i in range(1,5)]
n = len(l2)
resultd = np.zeros((3,n))

for i in range(n):
    modeld1 = Model(xtrain.shape[1])
```

Solution (cont.)

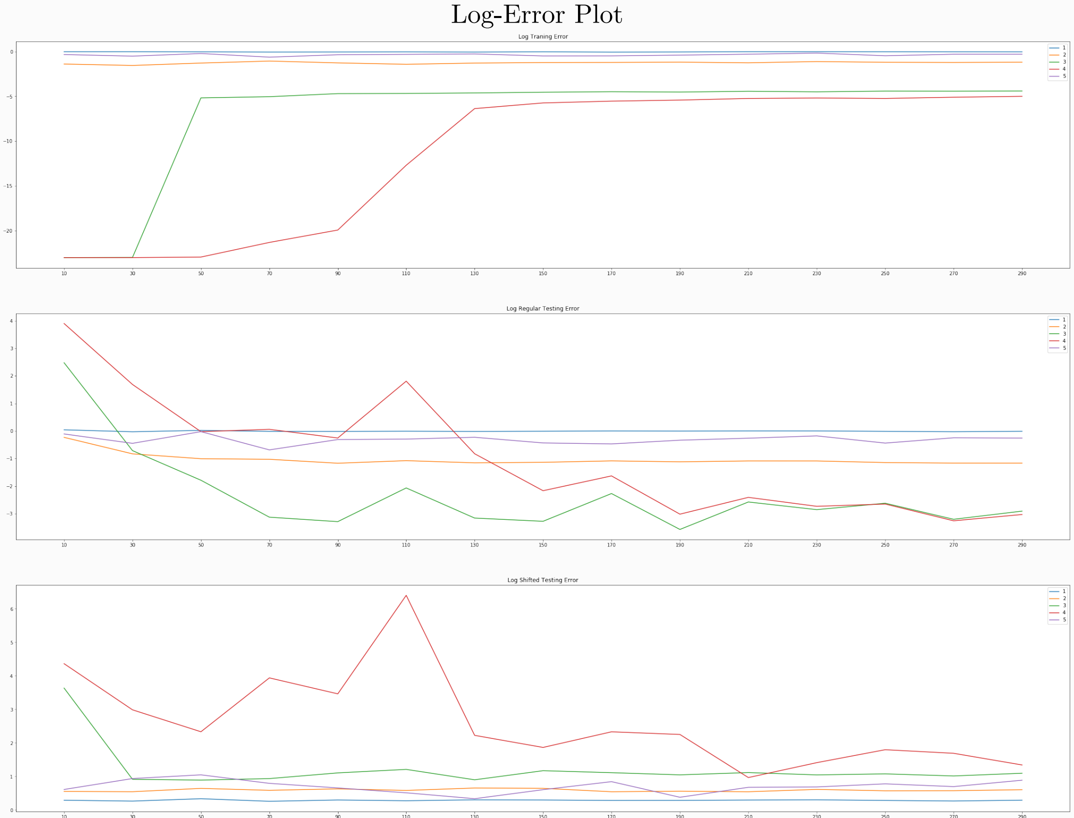
```
    for _ in range(k2[i]):
        modeld1.addLayer(DenseLayer(l2[i],ReLUActivation()))
    modeld1.addLayer(DenseLayer(2,LinearActivation()))
    modeld1.initialize(QuadraticCost())
    hist = modeld1.train(xtrain,ytrain,500,GDOptimizer(eta=0.0001))
    yHat = modeld1.predict(xtrain)
    resultd[0,i] = np.mean(np.linalg.norm(yHat - ytrain,axis=1))
    resultd[1,i] = np.mean(np.linalg.norm(modeld1.predict(xtest1)
     - ytest1,axis=1))
    resultd[2,i] = np.mean(np.linalg.norm(modeld1.predict(xtest2)
     - ytest2,axis=1))

plt.figure(figsize=(30,30))
s = ['Traning Error', 'Regular Testing Error', 'Shifted Testing Error']
for i in range(3):
    plt.subplot(3,1,i+1)
    plt.plot(np.arange(4),resultd[i,:])
    plt.xticks(np.arange(4),k2)
    plt.title(s[i])
plt.show()
```

(e)

The nerual network is still not very stable but it has been improved.

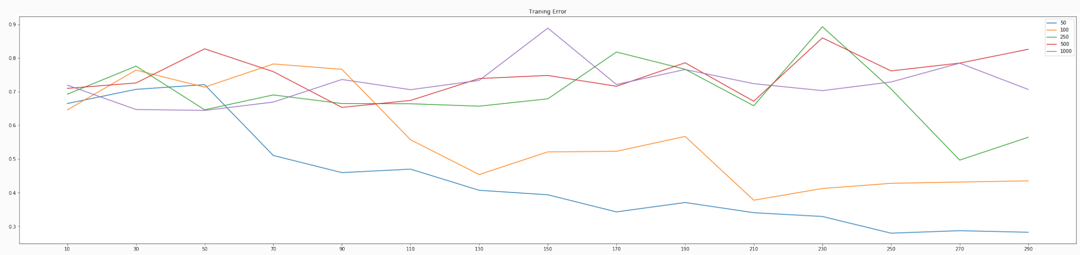
Error Plot

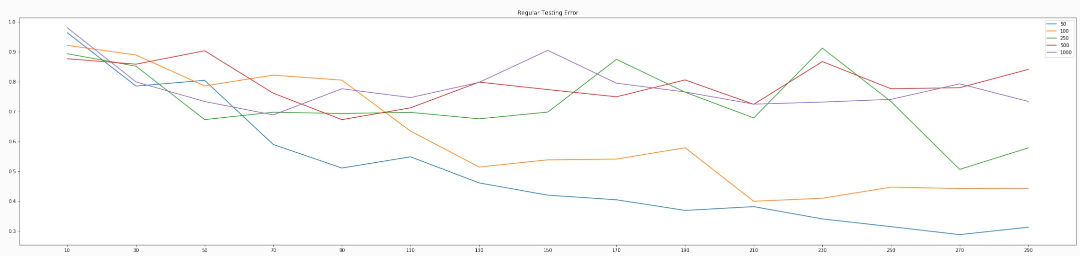
Log-Error Plot

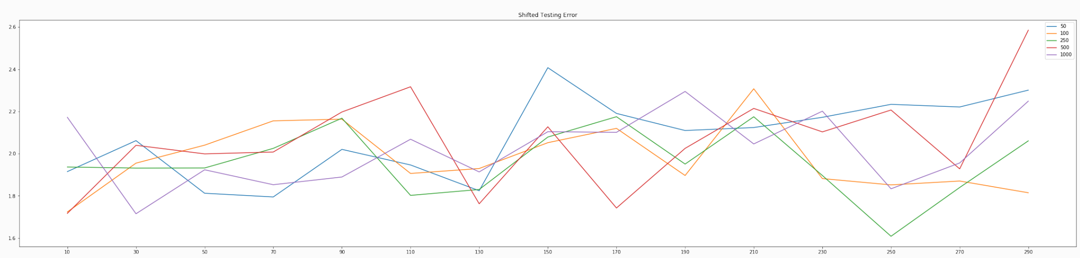


(f)

In defferent sizes of data set, SGD with defferent batch sizes have different impact on the model training. Sometimes it will be better to just use gradient descend (batch size = 1000). But it is not stable and sometimes need more iterations to train the network.








```
for i in range(15):
    for _ in range(5):
        model5 = Model(ztrain[i][0].shape[1])
        model5.addLayer(DenseLayer(100,ReLUActivation()))
        model5.addLayer(DenseLayer(100,ReLUActivation()))
        model5.addLayer(DenseLayer(2,LinearActivation()))
        model5.initialize(QuadraticCost())
        hist = model5.train(ztrain[i][0],ztrain[i][1],500,
          GDOptimizer(eta=0.0001))
        yHat = model5.predict(ztrain[i][0])
        resultf[0,4,i] += np.mean(np.linalg.norm(yHat - ztrain[i][1],
          axis=1))
        resultf[1,4,i] += np.mean(np.linalg.norm(
          model5.predict(xtest1) - ytest1,axis=1))
        resultf[2,4,i] += np.mean(np.linalg.norm(
          model5.predict(xtest2) - ytest2,axis=1))

batchsize = [50,100,250,500]
for j in range(len(batchsize)):
```

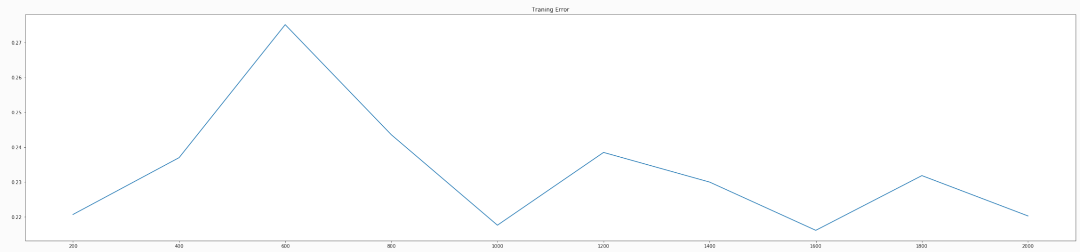
Solution (cont.)

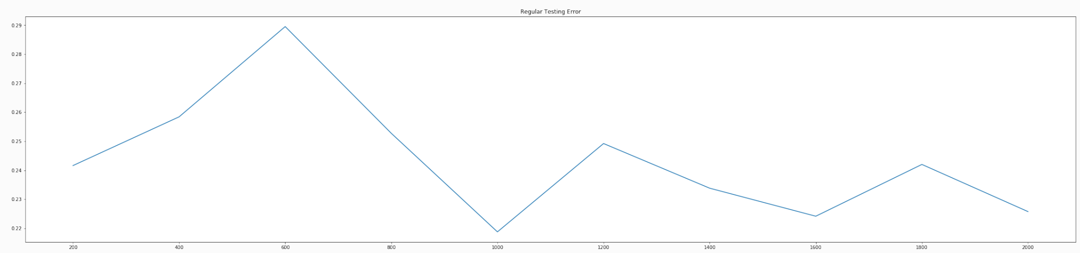
```
        for i in range(15):
            for _ in range(5):
                model5 = Model(ztrain[i][0].shape[1])
                model5.addLayer(DenseLayer(100,ReLUActivation()))
                model5.addLayer(DenseLayer(100,ReLUActivation()))
                model5.addLayer(DenseLayer(2,LinearActivation()))
                model5.initialize(QuadraticCost())
                hist = model5.trainBatch(ztrain[i][0],ztrain[i][1],
                  batchsize[j],500,GDOptimizer(eta=0.0001))
                yHat = model5.predict(ztrain[i][0])
                resultf[0,j,i] += np.mean(np.linalg.norm(yHat
                  - ztrain[i][1],axis=1))
                resultf[1,j,i] += np.mean(np.linalg.norm(
                  model5.predict(xtest1) - ytest1,axis=1))
                resultf[2,j,i] += np.mean(np.linalg.norm(
                  model5.predict(xtest2) - ytest2,axis=1))

resultf /= 5
```

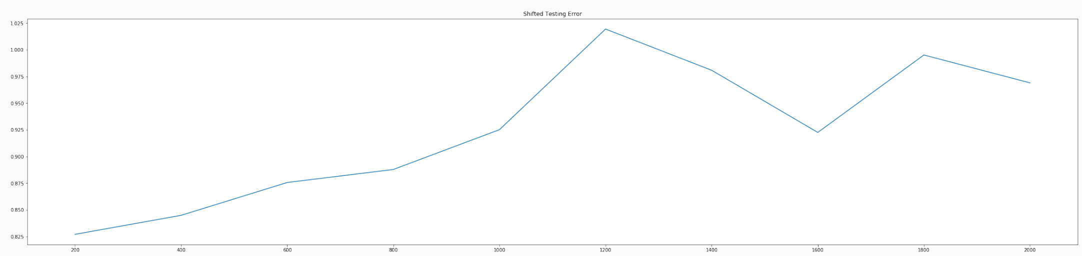
(g)

Use $k = 2$, $l = 46$, $learning\ rate = 0.0001$, $epoch = 1000$, $n_{train} = 1000$

We can see that $n_{train} = 1000$ is best for regular data set; $n_{train} = 200$ is best for shifted data set and therefore it's good to generlize to shifted data.








```
for i in range(15):
    for _ in range(10):
        model5 = Model(ztrain[i][0].shape[1])
        model5.addLayer(DenseLayer(150,ReLUActivation()))
        model5.addLayer(DenseLayer(150,ReLUActivation()))
        model5.addLayer(DenseLayer(2,LinearActivation()))
        model5.initialize(QuadraticCost())
        hist = model5.train(ztrain[i][0],ztrain[i][1],500,
          GDOptimizer(eta=0.0001))
        yHat = model5.predict(ztrain[i][0])
        resultg[0,i] += np.mean(np.linalg.norm(yHat - ztrain[i][1],
          axis=1))
        resultg[1,i] += np.mean(np.linalg.norm(model5.predict(xtest1)
          - ytest1,axis=1))
        resultg[2,i] += np.mean(np.linalg.norm(model5.predict(xtest2)
          - ytest2,axis=1))
resultg /= 10
```

Question 5

The training error on MNIST : 0.9138

```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

FLAGS = None


def main(_):
    # Import data
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])
    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    y = tf.matmul(x, W) + b

    # Define loss and optimizer
    y_ = tf.placeholder(tf.float32, [None, 10])

    # The raw formulation of cross-entropy,
    #
    #   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
    #                                 reduction_indices=[1]))
    #
    # can be numerically unstable.
    #
    # So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
    # outputs of 'y', and then average across the batch.
    cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
    train_step =
     tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Solution (cont.)

```python

    sess = tf.InteractiveSession()
    tf.global_variables_initializer().run()
    # Train
    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

        # Test trained model
        correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        print(str(i)+': ',sess.run(accuracy,
            feed_dict={x: mnist.test.images,
            y_: mnist.test.labels}))

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str,
                        default='/tmp/tensorflow/mnist/input_data',
                        help='Directory for storing input data')
    FLAGS, unparsed = parser.parse_known_args()
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

Question 6

Question What's the difference among various deep learning frameworks?

Solution

Framwork	Institution	Programming Language	Stars	Forks	Contributors
Thensorflow	Google	Python/C++/Go/$\cdots$	41628	19339	568
Caffe	BVLC	C++/Python	14956	9282	221
Keras	fchollet	Python	10727	3575	322
CNTK	Microsoft	C++	9063	2144	100
MXNet	DMLC	Python/C++/R/$\cdots$	7393	2745	241
Torch7	Facebook	Lua	6111	1784	113
Theano	U.Montreal	Python	5352	1868	271
Deeplearning4J	DeepLearning4J	Java/Scala	5053	1927	101
Leaf	AutumnAI	Rust	4562	216	14

Tensorflow

(1) Low-level core (C++/CUDA)

(2) Simple Python API to define the computational graph

(3) High-level API (TF-Learn, TF-Slim, soon Keras)

Theano

(1) Pioneered the use of a computational graph.

(2) General machine learning tool -¿ Use of Lasagne and Keras.

(3) Very popular in the research community, but not elsewhere. Falling behind. The development has been stopped.

Keras

(1) Easy-to-use Python library

(2) It wraps Theano and TensorFlow (it benefits from the advantages of both)

(3) Guiding principles: modularity, minimalism, extensibility, and Python-nativeness

(4) Less flexible

(5) Less projects available online than caffe

(6) Multi-GPU not 100% working

Caffe

(1) Applications in machine learning, vision, speech and multimedia.

(2) Good Python and MATLAB interfaces.

(3) No auto-differentiation.

(4) Need of examples to template own code.

HW9

October 28, 2017

1 Question 4

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import scipy.spatial
        import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt
```

```
In [89]: ########################################################################
         #########  Data Generating Functions ###################################
         ########################################################################

         def generate_sensors(num_sensors = 7, spatial_dim = 2):
             """
             Generate sensor locations.
             Input:
             num_sensors: The number of sensors.
             spatial_dim: The spatial dimension.
             Output:
             sensor_loc: num_sensors * spatial_dim numpy array.
             """
             sensor_loc = 100*np.random.randn(num_sensors,spatial_dim)
             return sensor_loc

         def generate_dataset(sensor_loc, num_sensors = 7, spatial_dim = 2,
                          num_data = 1, original_dist = True, noise = 1):
             """
             Generate the locations of n points.

             Input:
             sensor_loc: num_sensors * spatial_dim numpy array. Location of sensor.
             num_sensors: The number of sensors.
             spatial_dim: The spatial dimension.
             num_data: The number of points.
             original_dist: Whether the data are generated from the original
             distribution.
```

```
Output:
obj_loc: num_data * spatial_dim numpy array. The location of the num_data objects.
distance: num_data * num_sensors numpy array. The distance between object and
the num_sensors sensors.
"""
assert num_sensors, spatial_dim == sensor_loc.shape

obj_loc = 100*np.random.randn(num_data, spatial_dim)
if not original_dist:
    obj_loc += 1000

distance = scipy.spatial.distance.cdist(obj_loc,
                                       sensor_loc,
                                       metric='euclidean')
distance += np.random.randn(num_data, num_sensors) * noise
return distance, obj_loc
```

1.1 Generative Model

```
In [172]: class Generative_Model(object):
              def __init__(self,
                           num_gd_replicates = 20,
                           total_step_count = 1000,
                           step_size = lambda i: 1/(1+i),
                           err = 1e-5):
                  self.num_gd_replicates = num_gd_replicates
                  self.total_step_count = total_step_count
                  self.step_size = step_size
                  self.err = err


              def gradient1(self, obj_loc, d, sen_loc):
                  g = np.zeros_like(sen_loc)
                  for i in range(len(g)):
                      g[i] = -np.dot(((np.linalg.norm(obj_loc-sen_loc[i],axis=1)-d[:,i])/np.lina
                  if np.any(np.isnan(g)):
                      return 0
                  else:
                      return g

              def gradient_descend_step1(self,obj_loc, d, sen_loc, step_count):
                  """Computes the new point after the update at x."""
                  sen_loc = sen_loc - self.step_size(step_count) * self.gradient1(obj_loc, d, se
                  return sen_loc


              def gradient_descend1(self,obj_loc, d, sen_loc):
```

```
        """Computes several updates towards the minimum of ||Ax-b|| from p.

        Params:
            obj_loc: object locations
            d: measurements of the distance from each sensor to objects
            sen_loc: initialization point of sensors
            total_step_count: number of iterations to calculate
            step_size: function for determining the step size at step i
        """
        positions = [np.array(sen_loc)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step1(obj_loc, d, positions[-1], k)
            if np.linalg.norm(positions[-1]-new)<self.err:
                break
            else:
                positions.append(new)
        return np.array(positions)

    def gradient2(self,obj_loc, d, sen_loc):
        g = np.dot((((np.linalg.norm(obj_loc-sen_loc,axis=1)-d)/np.linalg.norm(obj_loc
        if np.any(np.isnan(g)):
            return 0
        else:
            return g/len(g)

    def gradient_descend_step2(self,obj_loc, d, sen_loc, step_count):
        """Computes the new point after the update at x."""
        obj_loc = obj_loc - self.step_size(step_count) * self.gradient2(obj_loc, d, se
        return obj_loc

    def gradient_descend2(self,obj_loc, d, sen_loc):
        positions = [np.array(obj_loc)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step2(positions[-1], d, sen_loc, k)
            if np.max(np.linalg.norm(positions[-1]-new,axis=1))<self.err:
                break
            else:
                positions.append(new)
        return positions

    def evaluate(self,obj_loc,d):
        est_obj_loc = np.zeros_like(obj_loc)
        for j in range(len(est_obj_loc)):
            p = []
            for i in range(self.num_gd_replicates):
                initial_position = np.random.randn(1,2)*100
                optimal_solutions = self.gradient_descend2(initial_position, d[j], sel
                p += [optimal_solutions[-1]]
```

```
            p = np.array(p)
            est_obj_loc[j] = np.unique(np.round(p,2),axis=0)[np.argmax(np.unique(np.ro
        error = np.mean(np.linalg.norm(est_obj_loc - obj_loc,axis=1))
        return {'MSE':error,'est_obj_loc':est_obj_loc}

    def train(self,obj_loc,sen_loc,d):
        m,n = np.shape(sen_loc)
        p = []
        for i in range(self.num_gd_replicates):
            initial_position = np.random.randn(m,n)*100
            optimal_solutions = self.gradient_descend1(obj_loc, d, initial_position)
            p += [optimal_solutions[-1]]
        p = np.array(p)
        self.est_sen_loc = np.unique(np.round(p,2),axis=0)[np.argmax(np.unique(np.roun
        self.train_error = self.evaluate(obj_loc,d)['MSE']

    def test(self,obj_loc,d):
        self.test_error = self.evaluate(obj_loc,d)['MSE']
        return self.test_error
```

1.2 Linear Model

```
In [91]: class Linear_Model(object):
             def __init__(self):
                 pass
             def train(self,X,Y,lamb=0):
                 m,n = np.shape(X)
                 self.A = np.linalg.solve(X.T.dot(X)+lamb*np.identity(n),X.T.dot(Y))
                 yhat = X.dot(self.A)
                 self.train_error = np.mean(np.linalg.norm(yhat-Y,axis=1))

             def test(self,X,Y):
                 m,n = np.shape(X)
                 yhat = X.dot(self.A)
                 self.test_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
                 return {'A':self.A,'error':self.test_error}
```

1.3 Second-Order Polynomial Regression Model

```
In [92]: import itertools
         class Second_Model(object):
             def __init__(self):
                 pass
             def train(self,X,Y,lamb=0):
                 m,n = np.shape(X)
                 X2 = np.ones((m,1))
```

```
            X2 = np.hstack((X2,X))
            for i in range(2,3):
                for j in itertools.combinations_with_replacement(np.arange(n), i):
                    xn = np.ones((1,m))
                    for k in j:
                        xn = xn*X[:,k]
                    X2 = np.hstack((X2,xn.reshape(m,1)))
            self.A = np.linalg.solve(X2.T.dot(X2)+lamb*np.identity(len(X2.T)),X2.T.dot(Y))
            yhat = X2.dot(self.A)
            self.train_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
        def test(self,X,Y):
            m,n = np.shape(X)
            X2 = np.ones((m,1))
            X2 = np.hstack((X2,X))
            for i in range(2,3):
                for j in itertools.combinations_with_replacement(np.arange(n), i):
                    xn = np.ones((1,m))
                    for k in j:
                        xn = xn*X[:,k]
                    X2 = np.hstack((X2,xn.reshape(m,1)))
            yhat = X2.dot(self.A)
            self.test_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
            return {'A':self.A,'error':self.test_error}
```

1.4 Third-Order Polynomial Regression Model

```
In [93]: class Third_Model(object):
             def __init__(self):
                 pass
             def train(self,X,Y,lamb=0):
                 m,n = np.shape(X)
                 X2 = np.ones((m,1))
                 X2 = np.hstack((X2,X))
                 for i in range(2,4):
                     for j in itertools.combinations_with_replacement(np.arange(n), i):
                         xn = np.ones((1,m))
                         for k in j:
                             xn = xn*X[:,k]
                         X2 = np.hstack((X2,xn.reshape(m,1)))
                 self.A = np.linalg.solve(X2.T.dot(X2)+lamb*np.identity(len(X2.T)),X2.T.dot(Y))
                 yhat = X2.dot(self.A)
                 self.train_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
             def test(self,X,Y):
                 m,n = np.shape(X)
                 X2 = np.ones((m,1))
                 X2 = np.hstack((X2,X))
                 for i in range(2,4):
                     for j in itertools.combinations_with_replacement(np.arange(n), i):
```

```
                xn = np.ones((1,m))
                for k in j:
                    xn = xn*X[:,k]
                X2 = np.hstack((X2,xn.reshape(m,1)))
        yhat = X2.dot(self.A)
        self.test_error = np.mean(np.linalg.norm(yhat-Y,axis=1))
        return {'A':self.A,'error':self.test_error}
```

1.5 Neural Network Model

```
In [94]: # Gradient descent optimization
         # The learning rate is specified by eta
         class GDOptimizer(object):
             def __init__(self, eta):
                 self.eta = eta

             def initialize(self, layers):
                 pass

             # This function performs one gradient descent step
             # layers is a list of dense layers in the network
             # g is a list of gradients going into each layer before the nonlinear activation
             # a is a list of of the activations of each node in the previous layer going
             def update(self, layers, g, a):
                 m = a[0].shape[1]
                 for layer, curGrad, curA in zip(layers, g, a):
                     update = np.dot(curGrad,curA.T)
                     updateB = np.sum(curGrad,1).reshape(layer.b.shape)
                     layer.updateWeights(-self.eta/m * np.dot(curGrad,curA.T))
                     layer.updateBias(-self.eta/m * np.sum(curGrad,1).reshape(layer.b.shape))

         # Cost function used to compute prediction errors
         class QuadraticCost(object):

             # Compute the squared error between the prediction yp and the observation y
             # This method should compute the cost per element such that the output is the
             # same shape as y and yp
             @staticmethod
             def fx(y,yp):
                 return 0.5 * np.square(yp-y)

             # Derivative of the cost function with respect to yp
             @staticmethod
             def dx(y,yp):
                 return y - yp

         # Sigmoid function fully implemented as an example
         class SigmoidActivation(object):
```

```
    @staticmethod
    def fx(z):
        return 1 / (1 + np.exp(-z))

    @staticmethod
    def dx(z):
        return SigmoidActivation.fx(z) * (1 - SigmoidActivation.fx(z))

# Hyperbolic tangent function
class TanhActivation(object):

    # Compute tanh for each element in the input z
    @staticmethod
    def fx(z):
        return np.tanh(z)

    # Compute the derivative of the tanh function with respect to z
    @staticmethod
    def dx(z):
        return 1 - np.square(np.tanh(z))

# Rectified linear unit
class ReLUActivation(object):
    @staticmethod
    def fx(z):
        return np.maximum(0,z)

    @staticmethod
    def dx(z):
        return (z>0).astype('float')

# Linear activation
class LinearActivation(object):
    @staticmethod
    def fx(z):
        return z

    @staticmethod
    def dx(z):
        return np.ones(z.shape)

# This class represents a single hidden or output layer in the neural network
class DenseLayer(object):

    # numNodes: number of hidden units in the layer
    # activation: the activation function to use in this layer
    def __init__(self, numNodes, activation):
        self.numNodes = numNodes
```

```
        self.activation = activation

    def getNumNodes(self):
        return self.numNodes

    # Initialize the weight matrix of this layer based on the size of the matrix W
    def initialize(self, fanIn, scale=1.0):
        s = scale * np.sqrt(6.0 / (self.numNodes + fanIn))
        self.W = np.random.normal(0, s,
                                   (self.numNodes,fanIn))
        #self.b = np.zeros((self.numNodes,1))
        self.b = np.random.uniform(-1,1,(self.numNodes,1))

    # Apply the activation function of the layer on the input z
    def a(self, z):
        return self.activation.fx(z)

    # Compute the linear part of the layer
    # The input a is an n x k matrix where n is the number of samples
    # and k is the dimension of the previous layer (or the input to the network)
    def z(self, a):
        #print('a:\n'+str(a))
        #print('Wa:\n'+str(self.W.dot(a)))
        return self.W.dot(a) + self.b # Note, this is implemented where we assume a is

    # Compute the derivative of the layer's activation function with respect to z
    # where z is the output of the above function.
    # This derivative does not contain the derivative of the matrix multiplication
    # in the layer.  That part is computed below in the model class.
    def dx(self, z):
        return self.activation.dx(z)

    # Update the weights of the layer by adding dW to the weights
    def updateWeights(self, dW):
        self.W = self.W + dW

    # Update the bias of the layer by adding db to the bias
    def updateBias(self, db):
        self.b = self.b + db

# This class handles stacking layers together to form the completed neural network
class Model(object):

    # inputSize: the dimension of the inputs that go into the network
    def __init__(self, inputSize):
        self.layers = []
        self.inputSize = inputSize
```

```python
    # Add a layer to the end of the network
    def addLayer(self, layer):
        self.layers.append(layer)

    # Get the output size of the layer at the given index
    def getLayerSize(self, index):
        if index >= len(self.layers):
            return self.layers[-1].getNumNodes()
        elif index < 0:
            return self.inputSize
        else:
            return self.layers[index].getNumNodes()

    # Initialize the weights of all of the layers in the network and set the cost
    # function to use for optimization
    def initialize(self, cost, initializeLayers=True):
        self.cost = cost
        if initializeLayers:
            for i in range(0,len(self.layers)):
                if i == len(self.layers) - 1:
                    self.layers[i].initialize(self.getLayerSize(i-1))
                else:
                    self.layers[i].initialize(self.getLayerSize(i-1))

    # Compute the output of the network given some input a
    # The matrix a has shape n x k where n is the number of samples and
    # k is the dimension
    # This function returns
    # yp - the output of the network
    # a - a list of inputs for each layer of the newtork where
    #     a[i] is the input to layer i
    # z - a list of values for each layer after evaluating layer.z(a) but
    #     before evaluating the nonlinear function for the layer
    def evaluate(self, x):
        curA = x.T
        a = [curA]
        z = []
        for layer in self.layers:
            z.append(layer.z(curA))
            curA = layer.a(z[-1])
            a.append(curA)
        yp = a.pop()
        return yp, a, z

    # Compute the output of the network given some input a
    # The matrix a has shape n x k where n is the number of samples and
    # k is the dimension
    def predict(self, a):
```

```
        a,_,_ = self.evaluate(a)
        return a.T

    # Train the network given the inputs x and the corresponding observations y
    # The network should be trained for numEpochs iterations using the supplied
    # optimizer
    def train(self, x, y, numEpochs, optimizer):

        # Initialize some stuff
        n = x.shape[0]
        hist = []
        optimizer.initialize(self.layers)

        # Run for the specified number of epochs
        for epoch in range(0,numEpochs):

            # Feed forward
            # Save the output of each layer in the list a
            # After the network has been evaluated, a should contain the
            # input x and the output of each layer except for the last layer
            yp, a, z = self.evaluate(x)

            # Compute the error
            C = self.cost.fx(yp,y.T)
            d = self.cost.dx(yp,y.T)
            grad = []

            # Backpropogate the error
            idx = len(self.layers)
            for layer, curZ in zip(reversed(self.layers),reversed(z)):
                idx = idx - 1
                # Here, we compute dMSE/dz_i because in the update
                # function for the optimizer, we do not give it
                # the z values we compute from evaluating the network
                grad.insert(0,np.multiply(d,layer.dx(curZ)))
                d = np.dot(layer.W.T,grad[0])

            # Update the errors
            optimizer.update(self.layers, grad, a)

            # Compute the error at the end of the epoch
            yh = self.predict(x)
            C = self.cost.fx(yh,y)
            C = np.mean(C)
            hist.append(C)
        return hist

    def trainBatch(self, x, y, batchSize, numEpochs, optimizer):
```

```
        # Copy the data so that we don't affect the original one when shuffling
        x = x.copy()
        y = y.copy()
        hist = []
        n = x.shape[0]

        for epoch in np.arange(0,numEpochs):

            # Shuffle the data
            r = np.arange(0,x.shape[0])
            x = x[r,:]
            y = y[r,:]
            e = []

            # Split the data in chunks and run SGD
            for i in range(0,n,batchSize):
                end = min(i+batchSize,n)
                batchX = x[i:end,:]
                batchY = y[i:end,:]
                e += self.train(batchX, batchY, 1, optimizer)
            hist.append(np.mean(e))

        return hist
```

1.6 (b)

1.7 Generate Data

```
In [330]: np.random.seed(0)
          sensor_loc = generate_sensors()
          train = {i:generate_dataset(sensor_loc=sensor_loc,num_data=10+20*i) for i in range(15)
          test1 = generate_dataset(sensor_loc=sensor_loc,num_data=1000)
          test2 = generate_dataset(sensor_loc=sensor_loc,num_data=1000,original_dist=False)

          zsensor_loc = (sensor_loc - np.average(sensor_loc))/np.std(sensor_loc)

          xtest1 = (test1[0] - np.average(test1[0])) / np.std(test1[0]+1e-6);
          xtest2 = (test2[0] - np.average(test2[0])) / np.std(test2[0]+1e-6);
          ytest1 = (test1[1] - np.average(test1[1])) / np.std(test1[1]+1e-6);
          ytest2 = (test2[1] - np.average(test2[1])) / np.std(test2[1]+1e-6);
          ztrain = {i:[(train[0][0] - np.average(train[0][0])) / np.std(train[0][0]+1e-6),
                       (train[0][1] - np.average(train[0][1])) / np.std(train[0][1]+1e-6)] for i
          result = np.zeros((3,5,15))

In [331]: zsensor_loc = (sensor_loc - np.average(sensor_loc))/np.std(sensor_loc)

In [345]: xtest1 = (test1[0] - np.average(test1[0])) / np.std(test1[0]+1e-6);
          xtest2 = (test2[0] - np.average(test2[0])) / np.std(test2[0]+1e-6);
```

```
        ytest1 = (test1[1] - np.average(test1[1])) / np.std(test1[1]+1e-6);
        ytest2 = (test2[1] - np.average(test2[1])) / np.std(test2[1]+1e-6);
        ztrain = {i:[(train[i][0] - np.average(train[i][0])) / np.std(train[i][0]+1e-6),
                     (train[i][1] - np.average(train[i][1])) / np.std(train[i][1]+1e-6)] for i
        result = np.zeros((3,5,15))
```

```
In [74]: np.shape(train[0][0])
```

```
Out[74]: (10, 7)
```

```
In [351]: #model1 = [Generative_Model() for _ in range(15)]
          for i in range(15):
              model1[i].train(ztrain[i][1],zsensor_loc,ztrain[i][0])
              result[0,0,i] = model1[i].train_error
              print(i)
              model1[i].test(ytest1, xtest1)
              result[1,0,i] = model1[i].test_error
              print(i)
              model1[i].test(ytest2, xtest2)
              result[2,0,i] = model1[i].test_error
              print(i)
```

```
0
0
0
1
1
1
2
2
2
3
3
3
4
4
4
5
5
5
6
6
6
7
7
7
8
8
8
```

```
9
9
9
10
10
10
11
11
11
12
12
12
13
13
13
14
14
14
```

In [352]:
```python
import pickle
with open('result','wb') as f:
    pickle.dump({'result':result},f,True)
```

In [333]:
```python
with open('result','rb') as f:
    result = pickle.load(f)['result']
```

In [367]:
```python
result[:,3,:]
```

Out[367]:
```
array([[  1.70209589e-13,   2.53009930e-13,   6.40847141e-12,
          4.46966993e-10,   2.08082862e-09,   3.04794575e-06,
          1.71872998e-03,   3.23326160e-03,   3.95691864e-03,
          4.45329000e-03,   5.34026340e-03,   5.61710652e-03,
          5.34789962e-03,   6.11200923e-03,   6.75728029e-03],
       [  4.94112804e+01,   5.40902723e+00,   9.76510054e-01,
          1.06257919e+00,   7.75477816e-01,   6.09656234e+00,
          4.37831537e-01,   1.14482557e-01,   1.95893033e-01,
          4.89012267e-02,   8.99328073e-02,   6.50131274e-02,
          7.05950582e-02,   3.84164237e-02,   4.82885821e-02],
       [  7.85630463e+01,   1.98350641e+01,   1.03112067e+01,
          5.14169197e+01,   3.19479968e+01,   6.02993199e+02,
          9.26404141e+00,   6.46171482e+00,   1.03004258e+01,
          9.51386164e+00,   2.63143591e+00,   4.10362085e+00,
          6.03794058e+00,   5.42340147e+00,   3.83103954e+00]])
```

In [355]:
```python
model2 = [Linear_Model() for _ in range(15)]
for i in range(15):
    model2[i].train(ztrain[i][0],ztrain[i][1])
    result[0,1,i] = model2[i].train_error
```

```
        model2[i].test(xtest1,ytest1)
        result[1,1,i] = model2[i].test_error
        model2[i].test(xtest2,ytest2)
        result[2,1,i] = model2[i].test_error
```

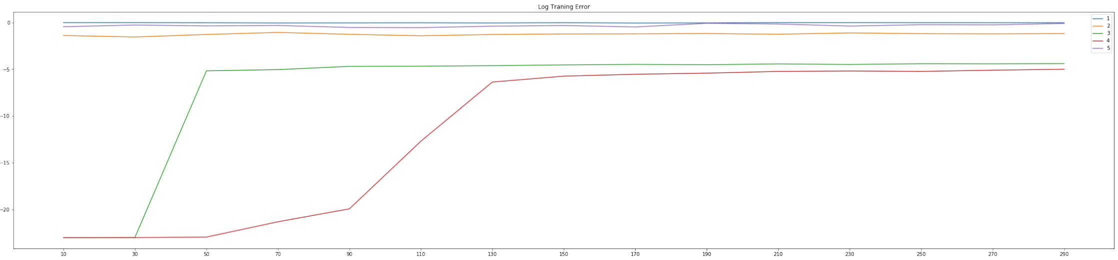
In [356]:
```
model3 =[Second_Model() for _ in range(15)]
for i in range(15):
    model3[i].train(ztrain[i][0],ztrain[i][1])
    result[0,2,i] = model3[i].train_error
    model3[i].test(xtest1,ytest1)
    result[1,2,i] = model3[i].test_error
    model3[i].test(xtest2,ytest2)
    result[2,2,i] = model3[i].test_error
```

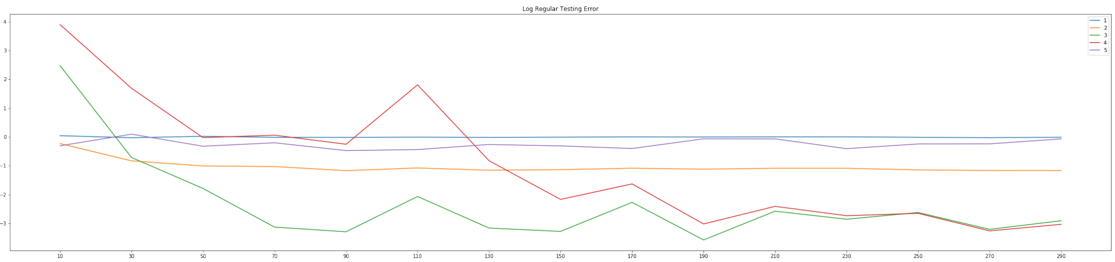
In [357]:
```
model4 =[Third_Model() for _ in range(15)]
for i in range(15):
    model4[i].train(ztrain[i][0],ztrain[i][1])
    result[0,3,i] = model4[i].train_error
    model4[i].test(xtest1,ytest1)
    result[1,3,i] = model4[i].test_error
    model4[i].test(xtest2,ytest2)
    result[2,3,i] = model4[i].test_error
```

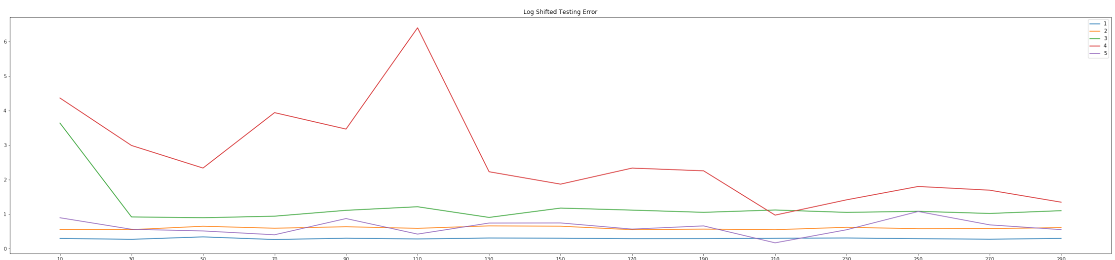
In [358]:
```
for i in range(15):
    model5 = Model(ztrain[i][0].shape[1])
    model5.addLayer(DenseLayer(100,ReLUActivation()))
    model5.addLayer(DenseLayer(100,ReLUActivation()))
    model5.addLayer(DenseLayer(2,LinearActivation()))
    model5.initialize(QuadraticCost())
    hist = model5.train(ztrain[i][0],ztrain[i][1],500,GDOptimizer(eta=0.0001))
    yHat = model5.predict(ztrain[i][0])
    result[0,4,i] = np.mean(np.linalg.norm(yHat - ztrain[i][1],axis=1))
    result[1,4,i] = np.mean(np.linalg.norm(model5.predict(xtest1) - ytest1,axis=1))
    result[2,4,i] = np.mean(np.linalg.norm(model5.predict(xtest2) - ytest2,axis=1))
```

In [359]:
```
plt.figure(figsize=(40,30))
s = ['Traning Error', 'Regular Testing Error', 'Shifted Testing Error']
for i in range(3):
    plt.subplot(3,1,i+1)
    for j in range(5):
        plt.plot(np.arange(15),result[i,j,:],label=str(j+1))
    plt.title(s[i])
    plt.legend()
    plt.xticks(np.arange(15),np.arange(10,310,20))
plt.show()
```


```
In [360]: plt.figure(figsize=(40,30))
          for i in range(3):
              plt.subplot(3,1,i+1)
              for j in range(5):
                  plt.plot(np.arange(15),np.log(result[i,j,:]+1e-10),label=str(j+1))
              plt.legend()
              plt.title('Log '+s[i])
              plt.xticks(np.arange(15),np.arange(10,310,20))
          plt.show()
```







1.8 (c)

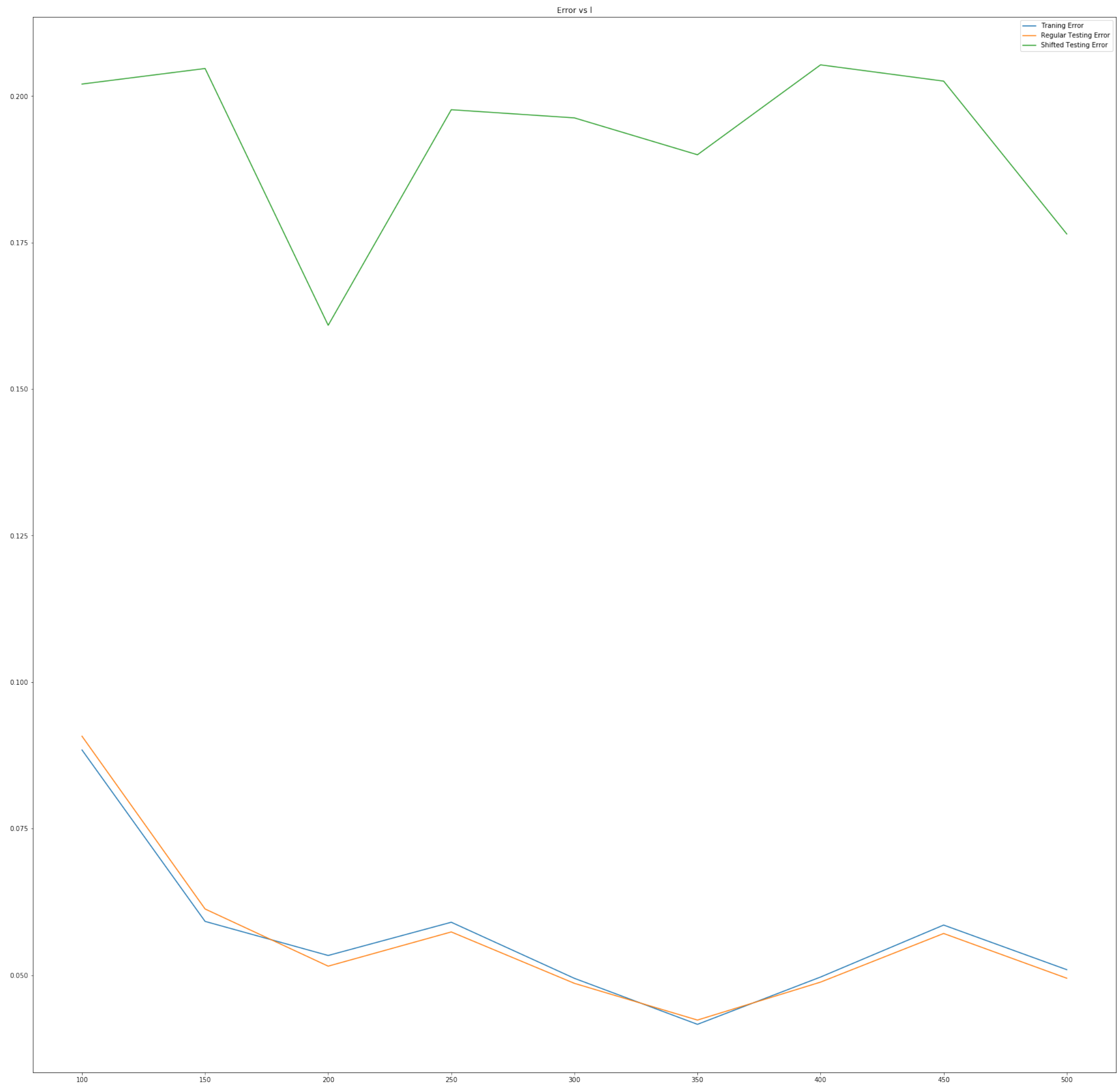
```
In [379]: np.random.seed(0)
          sensor_loc = generate_sensors()
          train = generate_dataset(sensor_loc=sensor_loc,num_data=200)
          test1 = generate_dataset(sensor_loc=sensor_loc,num_data=1000)
          test2 = generate_dataset(sensor_loc=sensor_loc,num_data=1000,original_dist=False)
```

```
In [380]: xtest1 = (test1[0] - np.average(test1[0])) / np.std(test1[0]+1e-6);
          xtest2 = (test2[0] - np.average(test2[0])) / np.std(test2[0]+1e-6);
          ytest1 = (test1[1] - np.average(test1[1])) / np.std(test1[1]+1e-6);
          ytest2 = (test2[1] - np.average(test2[1])) / np.std(test2[1]+1e-6);
          xtrain = (train[0] - np.average(train[0])) / np.std(train[0]+1e-6);
          ytrain = (train[1] - np.average(train[1])) / np.std(train[1]+1e-6);
```

```
In [382]: l = np.arange(100,550,50)
          n = len(l)
          resultc = np.zeros((3,n))

          for i in range(n):
              for _ in range(10):
```

```
        modelc1 = Model(xtrain.shape[1])
        modelc1.addLayer(DenseLayer(l[i],ReLUActivation()))
        modelc1.addLayer(DenseLayer(l[i],ReLUActivation()))
        modelc1.addLayer(DenseLayer(2,LinearActivation()))
        modelc1.initialize(QuadraticCost())
        hist = modelc1.train(xtrain,ytrain,500,GDOptimizer(eta=0.0001))
        yHat = modelc1.predict(xtrain)
        resultc[0,i] = np.mean(np.linalg.norm(yHat - ytrain,axis=1))
        resultc[1,i] = np.mean(np.linalg.norm(modelc1.predict(xtest1) - ytest1,axis=1)
        resultc[2,i] = np.mean(np.linalg.norm(modelc1.predict(xtest2) - ytest2,axis=1)
resultc /= 10
```

In [383]:
```
plt.figure(figsize=(30,30))
for i in range(3):
    plt.plot(np.arange(9),resultc[i,:],label=s[i])
plt.title('Error vs l')
plt.legend()
plt.xticks(np.arange(9),l)
plt.show()
```




```
In [384]: import pickle
          with open('resultc','wb') as f:
              pickle.dump({'resultc':resultc},f,True)
```

1.9 (d)

```
In [232]: l2 = [round(10000/9)]
          for k in range(2,5):
              l2.append(round((-9+np.sqrt(81+10000*(k-1)))/2/(k-1)))
          l2 = np.array(l2,dtype=int)
          strk = ''
          for i in range(len(l2)):
              strk += str(i+1)+'\t'
          print('k: '+strk)
```

```
strl = ''
for i in range(len(l2)):
    strl += str(l2[i])+'\t'
print('l: '+strl)
```

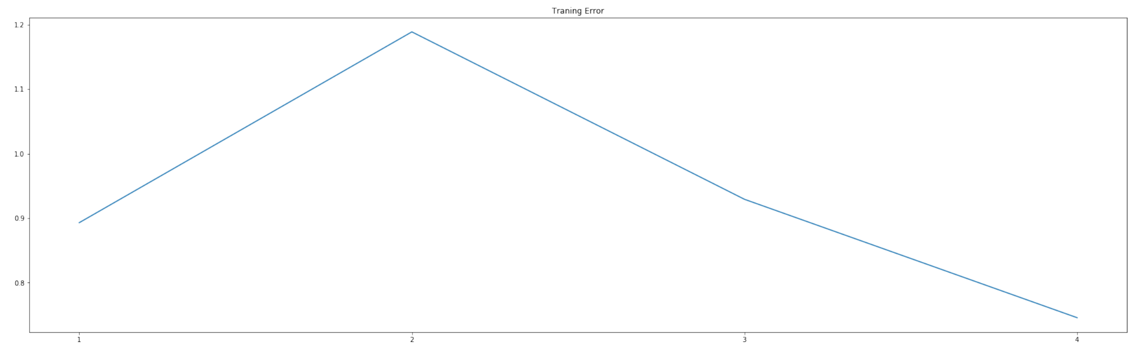
```
k: 1        2        3        4
l: 1111        46        33        27
```

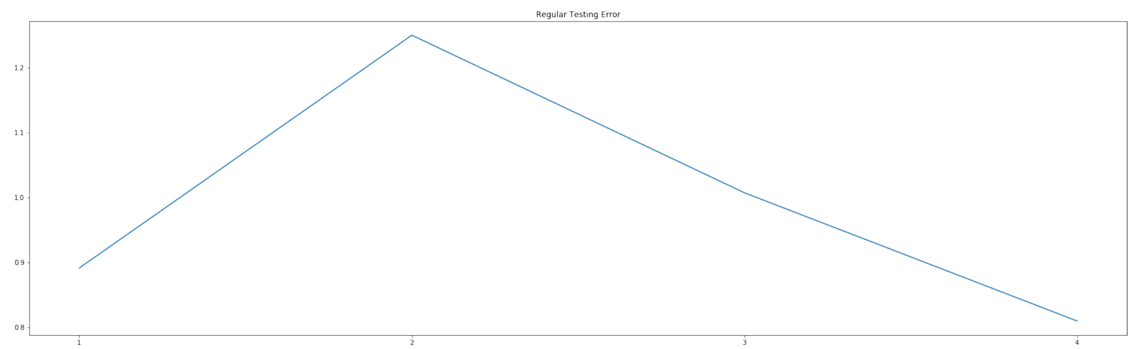
```
In [213]: np.random.seed(0)
          sensor_loc = generate_sensors()
          train = generate_dataset(sensor_loc=sensor_loc,num_data=200)
          test1 = generate_dataset(sensor_loc=sensor_loc,num_data=200)
          test2 = generate_dataset(sensor_loc=sensor_loc,num_data=200,original_dist=False)
```

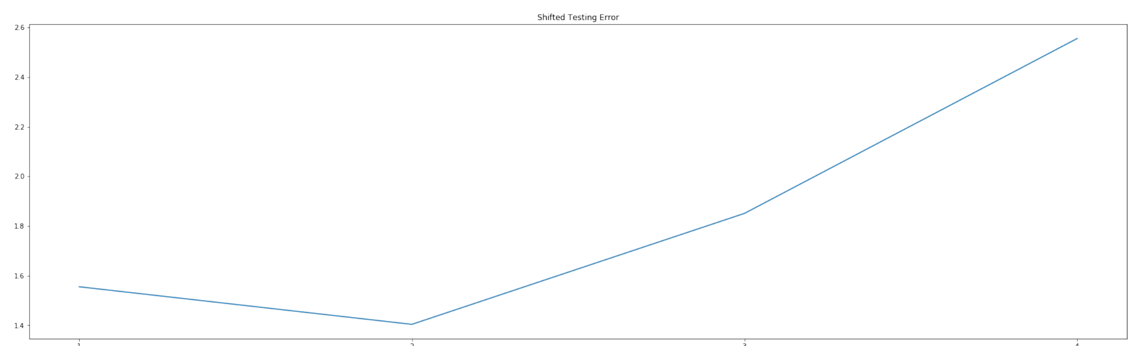
```
In [214]: xtest1 = (test1[0] - np.average(test1[0])) / np.std(test1[0]+1e-6);
          xtest2 = (test2[0] - np.average(test2[0])) / np.std(test2[0]+1e-6);
          ytest1 = (test1[1] - np.average(test1[1])) / np.std(test1[1]+1e-6);
          ytest2 = (test2[1] - np.average(test2[1])) / np.std(test2[1]+1e-6);
          xtrain = (train[0] - np.average(train[0])) / np.std(train[0]+1e-6);
          ytrain = (train[1] - np.average(train[1])) / np.std(train[1]+1e-6);
```

```
In [234]: k2 = [i for i in range(1,5)]
          n = len(l2)
          resultd = np.zeros((3,n))

          for i in range(n):
              modeld1 = Model(xtrain.shape[1])
              for _ in range(k2[i]):
                  modeld1.addLayer(DenseLayer(l2[i],ReLUActivation()))
              modeld1.addLayer(DenseLayer(2,LinearActivation()))
              modeld1.initialize(QuadraticCost())
              hist = modeld1.train(xtrain,ytrain,500,GDOptimizer(eta=0.0001))
              yHat = modeld1.predict(xtrain)
              resultd[0,i] = np.mean(np.linalg.norm(yHat - ytrain,axis=1))
              resultd[1,i] = np.mean(np.linalg.norm(modeld1.predict(xtest1) - ytest1,axis=1))
              resultd[2,i] = np.mean(np.linalg.norm(modeld1.predict(xtest2) - ytest2,axis=1))
```

```
In [240]: plt.figure(figsize=(30,30))
          s = ['Traning Error', 'Regular Testing Error', 'Shifted Testing Error']
          for i in range(3):
              plt.subplot(3,1,i+1)
              plt.plot(np.arange(4),resultd[i,:])
              plt.xticks(np.arange(4),k2)
              plt.title(s[i])
          plt.show()
```








```
In [244]: import pickle
          with open('resultd','wb') as f:
              pickle.dump({'resultd':resultd},f,True)
```

1.10 (e)

```
In [241]: np.random.seed(0)
          sensor_loc = generate_sensors()
          train = {i:generate_dataset(sensor_loc=sensor_loc,num_data=10+20*i) for i in range(15)
          test1 = generate_dataset(sensor_loc=sensor_loc,num_data=1000)
          test2 = generate_dataset(sensor_loc=sensor_loc,num_data=1000,original_dist=False)
```

```
In [252]: xtest1 = (test1[0] - np.average(test1[0])) / np.std(test1[0]+1e-6);
          xtest2 = (test2[0] - np.average(test2[0])) / np.std(test2[0]+1e-6);
```

```
ytest1 = (test1[1] - np.average(test1[1])) / np.std(test1[1]+1e-6);
ytest2 = (test2[1] - np.average(test2[1])) / np.std(test2[1]+1e-6);
ztrain = {i:[(train[i][0] - np.average(train[i][0])) / np.std(train[i][0]+1e-6),
             (train[i][1] - np.average(train[i][1])) / np.std(train[i][1]+1e-6)] for i
```

In [361]:
```
resulte = result
```

In [362]:
```
for i in range(15):
    model5 = Model(ztrain[i][0].shape[1])
    model5.addLayer(DenseLayer(150,ReLUActivation()))
    model5.addLayer(DenseLayer(150,ReLUActivation()))
    model5.addLayer(DenseLayer(2,LinearActivation()))
    model5.initialize(QuadraticCost())
    hist = model5.train(ztrain[i][0],ztrain[i][1],500,GDOptimizer(eta=0.0001))
    yHat = model5.predict(ztrain[i][0])
    resulte[0,4,i] = np.mean(np.linalg.norm(yHat - ztrain[i][1],axis=1))
    resulte[1,4,i] = np.mean(np.linalg.norm(model5.predict(xtest1) - ytest1,axis=1))
    resulte[2,4,i] = np.mean(np.linalg.norm(model5.predict(xtest2) - ytest2,axis=1))
```

In [363]:
```
plt.figure(figsize=(40,30))
s = ['Traning Error', 'Regular Testing Error', 'Shifted Testing Error']
for i in range(3):
    plt.subplot(3,1,i+1)
    for j in range(5):
        plt.plot(np.arange(15),resulte[i,j,:],label=str(j+1))
    plt.title(s[i])
    plt.legend()
    plt.xticks(np.arange(15),np.arange(10,310,20))
plt.show()
```

In [364]:
```
plt.figure(figsize=(40,30))
for i in range(3):
    plt.subplot(3,1,i+1)
    for j in range(5):
        plt.plot(np.arange(15),np.log(resulte[i,j,:]+1e-10),label=str(j+1))
    plt.legend()
    plt.title('Log '+s[i])
    plt.xticks(np.arange(15),np.arange(10,310,20))
plt.show()
```


```
In [365]: import pickle
          with open('resulte','wb') as f:
              pickle.dump({'resulte':resulte},f,True)
```

1.11 (f)

```
In [317]: np.random.seed(0)
          sensor_loc = generate_sensors()
          train = {i:generate_dataset(sensor_loc=sensor_loc,num_data=10+20*i) for i in range(15)
          test1 = generate_dataset(sensor_loc=sensor_loc,num_data=1000)
          test2 = generate_dataset(sensor_loc=sensor_loc,num_data=1000,original_dist=False)

In [318]: zsensor_loc = (sensor_loc - np.average(sensor_loc))/np.std(sensor_loc)

In [319]: xtest1 = (test1[0] - np.average(test1[0])) / np.std(test1[0]+1e-6);
          xtest2 = (test2[0] - np.average(test2[0])) / np.std(test2[0]+1e-6);
          ytest1 = (test1[1] - np.average(test1[1])) / np.std(test1[1]+1e-6);
          ytest2 = (test2[1] - np.average(test2[1])) / np.std(test2[1]+1e-6);
          ztrain = {i:[(train[i][0] - np.average(train[i][0])) / np.std(train[i][0]+1e-6),
                       (train[i][1] - np.average(train[i][1])) / np.std(train[i][1]+1e-6)] for i

In [368]: resultf = np.zeros((3,5,15))
```

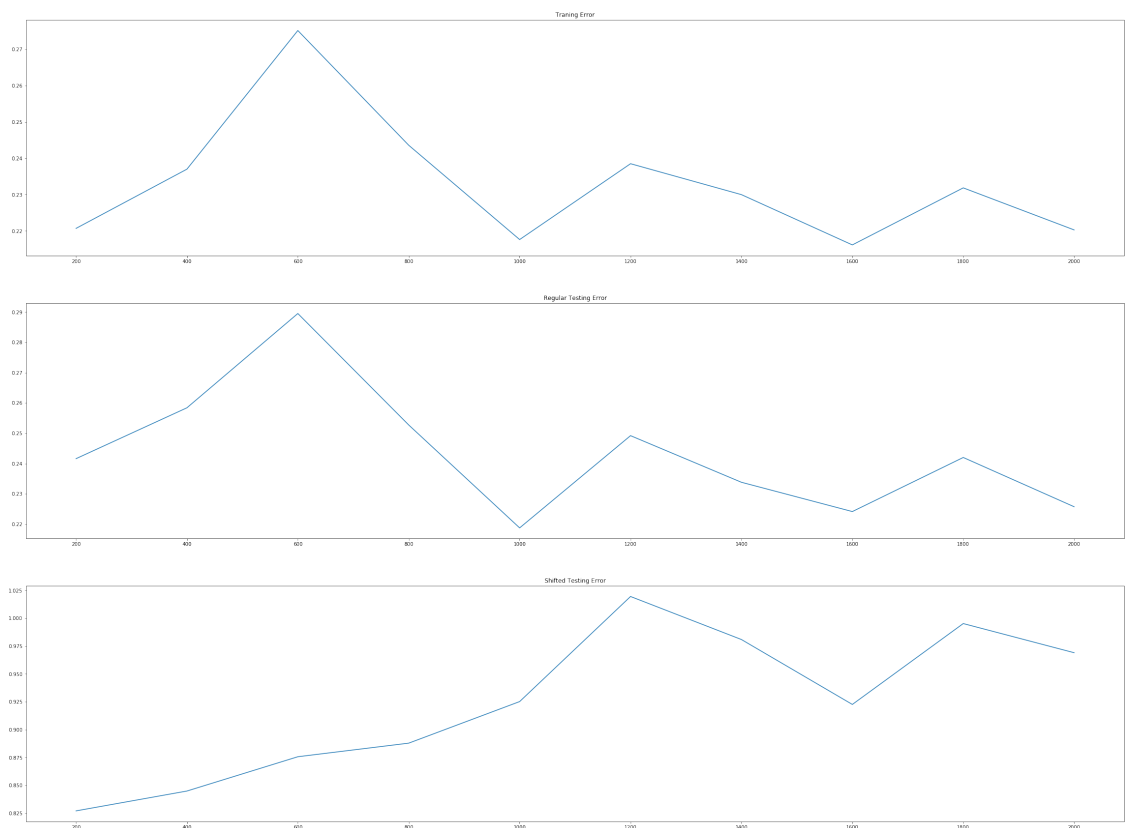
```
In [369]: for i in range(15):
              for _ in range(5):
                  model5 = Model(ztrain[i][0].shape[1])
                  model5.addLayer(DenseLayer(100,ReLUActivation()))
                  model5.addLayer(DenseLayer(100,ReLUActivation()))
                  model5.addLayer(DenseLayer(2,LinearActivation()))
                  model5.initialize(QuadraticCost())
                  hist = model5.train(ztrain[i][0],ztrain[i][1],500,GDOptimizer(eta=0.0001))
                  yHat = model5.predict(ztrain[i][0])
                  resultf[0,4,i] += np.mean(np.linalg.norm(yHat - ztrain[i][1],axis=1))
                  resultf[1,4,i] += np.mean(np.linalg.norm(model5.predict(xtest1) - ytest1,axis=
                  resultf[2,4,i] += np.mean(np.linalg.norm(model5.predict(xtest2) - ytest2,axis=

In [370]: batchsize = [50,100,250,500]
          for j in range(len(batchsize)):
              for i in range(15):
                  for _ in range(5):
                      model5 = Model(ztrain[i][0].shape[1])
                      model5.addLayer(DenseLayer(100,ReLUActivation()))
                      model5.addLayer(DenseLayer(100,ReLUActivation()))
                      model5.addLayer(DenseLayer(2,LinearActivation()))
                      model5.initialize(QuadraticCost())
                      hist = model5.trainBatch(ztrain[i][0],ztrain[i][1],batchsize[j],500,GDOpti
                      yHat = model5.predict(ztrain[i][0])
                      resultf[0,j,i] += np.mean(np.linalg.norm(yHat - ztrain[i][1],axis=1))
                      resultf[1,j,i] += np.mean(np.linalg.norm(model5.predict(xtest1) - ytest1,a
                      resultf[2,j,i] += np.mean(np.linalg.norm(model5.predict(xtest2) - ytest2,a

In [371]: resultf /= 5

In [372]: plt.figure(figsize=(40,30))
          s = ['Traning Error', 'Regular Testing Error', 'Shifted Testing Error']
          b = batchsize + [1000]
          for i in range(3):
              plt.subplot(3,1,i+1)
              for j in range(5):
                  plt.plot(np.arange(15),resultf[i,j,:],label=str(b[j]))
              plt.title(s[i])
              plt.legend()
              plt.xticks(np.arange(15),np.arange(10,310,20))
          plt.show()
```

1.12 (g)

```
In [388]: np.random.seed(0)
          sensor_loc = generate_sensors()
          train = {i:generate_dataset(sensor_loc=sensor_loc,num_data=200*(i+1)) for i in range(1
          test1 = generate_dataset(sensor_loc=sensor_loc,num_data=1000)
          test2 = generate_dataset(sensor_loc=sensor_loc,num_data=1000,original_dist=False)

In [389]: xtest1 = (test1[0] - np.average(test1[0])) / np.std(test1[0]+1e-6);
          xtest2 = (test2[0] - np.average(test2[0])) / np.std(test2[0]+1e-6);
          ytest1 = (test1[1] - np.average(test1[1])) / np.std(test1[1]+1e-6);
          ytest2 = (test2[1] - np.average(test2[1])) / np.std(test2[1]+1e-6);
          ztrain = {i:[(train[i][0] - np.average(train[i][0])) / np.std(train[i][0]+1e-6),
                      (train[i][1] - np.average(train[i][1])) / np.std(train[i][1]+1e-6)] for i
          resultg = np.zeros((3,10))

In [390]: zsensor_loc = (sensor_loc - np.average(sensor_loc))/np.std(sensor_loc)

In [391]: for i in range(10):
              for _ in range(5):
                  model5 = Model(ztrain[i][0].shape[1])
                  model5.addLayer(DenseLayer(150,ReLUActivation()))
```

```
                model5.addLayer(DenseLayer(150,ReLUActivation()))
                model5.addLayer(DenseLayer(2,LinearActivation()))
                model5.initialize(QuadraticCost())
                hist = model5.train(ztrain[i][0],ztrain[i][1],1000,GDOptimizer(eta=0.0001))
                yHat = model5.predict(ztrain[i][0])
                resultg[0,i] += np.mean(np.linalg.norm(yHat - ztrain[i][1],axis=1))
                resultg[1,i] += np.mean(np.linalg.norm(model5.predict(xtest1) - ytest1,axis=1)
                resultg[2,i] += np.mean(np.linalg.norm(model5.predict(xtest2) - ytest2,axis=1)
        resultg /= 10
```

In [395]:
```
plt.figure(figsize=(40,30))
s = ['Traning Error', 'Regular Testing Error', 'Shifted Testing Error']
for i in range(3):
    plt.subplot(3,1,i+1)
    plt.plot(np.arange(10),resultg[i,:])
    plt.title(s[i])
    plt.xticks(np.arange(10),np.arange(200,2200,200))
plt.show()
```



In [ ]:
```
import pickle
with open('resultg','wb') as f:
    pickle.dump({'resultg':resultg},f,True)
```

2 Question 5

In [ ]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

FLAGS = None


def main(_):
    # Import data
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])
    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    y = tf.matmul(x, W) + b

    # Define loss and optimizer
    y_ = tf.placeholder(tf.float32, [None, 10])

    # The raw formulation of cross-entropy,
    #
    #   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
    #                                 reduction_indices=[1]))
    #
    # can be numerically unstable.
    #
    # So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
    # outputs of 'y', and then average across the batch.
    cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
    train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

    sess = tf.InteractiveSession()
    tf.global_variables_initializer().run()
    # Train
    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

```python
            # Test trained model
            correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
            print(str(i)+': ',sess.run(accuracy, feed_dict={x: mnist.test.images,
                                                  y_: mnist.test.labels}))

    if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/mnist/input_dat
                          help='Directory for storing input data')
        FLAGS, unparsed = parser.parse_known_args()
        tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

```
Extracting /tmp/tensorflow/mnist/input_data/train-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/train-labels-idx1-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-labels-idx1-ubyte.gz
0:  0.4075
1:  0.4029
2:  0.4927
3:  0.5023
4:  0.6827
5:  0.4972
6:  0.7257
7:  0.7076
8:  0.6808
9:  0.7821
10:  0.7311
11:  0.7984
12:  0.7478
13:  0.7959
14:  0.816
15:  0.8162
16:  0.8256
17:  0.8379
18:  0.8455
19:  0.7874
20:  0.8062
21:  0.8478
22:  0.8206
23:  0.8508
24:  0.8498
25:  0.8481
26:  0.8395
27:  0.83
28:  0.8551
29:  0.8696
```

```
30:  0.8515
31:  0.8429
32:  0.8581
33:  0.8546
34:  0.8602
35:  0.8588
36:  0.8337
37:  0.8494
38:  0.7981
39:  0.8557
40:  0.869
41:  0.8695
42:  0.8684
43:  0.8542
44:  0.8672
45:  0.8553
46:  0.8643
47:  0.8606
48:  0.8672
49:  0.8609
50:  0.8725
51:  0.8744
52:  0.8247
53:  0.8613
54:  0.8773
55:  0.878
56:  0.875
57:  0.8757
58:  0.8839
59:  0.8809
60:  0.8772
61:  0.876
62:  0.869
63:  0.8794
64:  0.8745
65:  0.8747
66:  0.8716
67:  0.8671
68:  0.8655
69:  0.8604
70:  0.8629
71:  0.8495
72:  0.8724
73:  0.8746
74:  0.8797
75:  0.8577
76:  0.8707
77:  0.884
```

```
78:  0.8814
79:  0.8832
80:  0.8829
81:  0.8853
82:  0.8755
83:  0.8857
84:  0.8859
85:  0.8867
86:  0.8852
87:  0.8861
88:  0.8859
89:  0.8828
90:  0.882
91:  0.8854
92:  0.8876
93:  0.8814
94:  0.891
95:  0.887
96:  0.8672
97:  0.8766
98:  0.889
99:  0.8943
100:  0.8948
101:  0.8874
102:  0.8916
103:  0.8873
104:  0.8921
105:  0.8949
106:  0.8937
107:  0.8975
108:  0.8949
109:  0.8974
110:  0.896
111:  0.8949
112:  0.898
113:  0.8961
114:  0.8835
115:  0.8928
116:  0.8958
117:  0.8973
118:  0.8966
119:  0.8978
120:  0.8975
121:  0.9004
122:  0.9016
123:  0.8948
124:  0.888
125:  0.8931
```

126: 0.8975
127: 0.8892
128: 0.9007
129: 0.8963
130: 0.8966
131: 0.9003
132: 0.8919
133: 0.896
134: 0.8806
135: 0.8977
136: 0.8994
137: 0.899
138: 0.9019
139: 0.8988
140: 0.9014
141: 0.9003
142: 0.8959
143: 0.9031
144: 0.9001
145: 0.8816
146: 0.9021
147: 0.9036
148: 0.8938
149: 0.8854
150: 0.8926
151: 0.8967
152: 0.8943
153: 0.9026
154: 0.9005
155: 0.902
156: 0.8955
157: 0.8826
158: 0.8878
159: 0.8923
160: 0.8992
161: 0.8976
162: 0.9045
163: 0.8981
164: 0.9031
165: 0.9021
166: 0.902
167: 0.9059
168: 0.9048
169: 0.9019
170: 0.902
171: 0.8981
172: 0.9022
173: 0.9028

```
174:  0.8983
175:  0.8973
176:  0.8862
177:  0.8949
178:  0.8975
179:  0.8921
180:  0.9029
181:  0.904
182:  0.9073
183:  0.905
184:  0.8976
185:  0.9066
186:  0.8946
187:  0.9046
188:  0.9035
189:  0.9019
190:  0.9044
191:  0.9007
192:  0.9001
193:  0.9053
194:  0.9067
195:  0.9001
196:  0.9002
197:  0.8928
198:  0.9074
199:  0.9037
200:  0.9031
201:  0.8988
202:  0.9057
203:  0.9036
204:  0.9037
205:  0.9051
206:  0.8996
207:  0.9061
208:  0.9059
209:  0.9065
210:  0.9048
211:  0.907
212:  0.9073
213:  0.9041
214:  0.9042
215:  0.9032
216:  0.9054
217:  0.8933
218:  0.9029
219:  0.906
220:  0.9026
221:  0.9009
```

222: 0.8942
223: 0.8999
224: 0.9028
225: 0.9028
226: 0.9027
227: 0.9055
228: 0.9032
229: 0.9045
230: 0.9063
231: 0.902
232: 0.8933
233: 0.9053
234: 0.9022
235: 0.902
236: 0.9037
237: 0.9011
238: 0.9083
239: 0.9069
240: 0.9061
241: 0.9036
242: 0.9057
243: 0.9006
244: 0.9065
245: 0.9016
246: 0.9057
247: 0.9022
248: 0.9024
249: 0.8998
250: 0.8989
251: 0.9005
252: 0.9048
253: 0.908
254: 0.9082
255: 0.9074
256: 0.9079
257: 0.9056
258: 0.8981
259: 0.8992
260: 0.9101
261: 0.9076
262: 0.9068
263: 0.9095
264: 0.9104
265: 0.9036
266: 0.9093
267: 0.9042
268: 0.9059
269: 0.9064

```
270:  0.9093
271:  0.9021
272:  0.9051
273:  0.8835
274:  0.9038
275:  0.9043
276:  0.9087
277:  0.9081
278:  0.9091
279:  0.9105
280:  0.9045
281:  0.9013
282:  0.9076
283:  0.9099
284:  0.907
285:  0.9053
286:  0.908
287:  0.9027
288:  0.9073
289:  0.9107
290:  0.9098
291:  0.9087
292:  0.911
293:  0.9098
294:  0.9049
295:  0.8869
296:  0.9073
297:  0.9048
298:  0.9069
299:  0.9083
300:  0.9074
301:  0.9042
302:  0.9095
303:  0.9085
304:  0.909
305:  0.9014
306:  0.8978
307:  0.9089
308:  0.909
309:  0.9087
310:  0.9034
311:  0.9085
312:  0.9088
313:  0.9109
314:  0.9055
315:  0.9049
316:  0.9065
317:  0.9066
```

318: 0.9048
319: 0.9051
320: 0.9051
321: 0.9047
322: 0.9025
323: 0.8969
324: 0.8985
325: 0.889
326: 0.906
327: 0.9074
328: 0.9082
329: 0.9059
330: 0.9064
331: 0.9059
332: 0.9076
333: 0.9031
334: 0.9081
335: 0.9055
336: 0.906
337: 0.911
338: 0.9098
339: 0.9071
340: 0.9109
341: 0.9098
342: 0.9076
343: 0.8946
344: 0.8856
345: 0.9065
346: 0.9017
347: 0.9083
348: 0.8991
349: 0.9069
350: 0.9063
351: 0.9086
352: 0.9099
353: 0.9077
354: 0.9077
355: 0.9106
356: 0.9117
357: 0.9098
358: 0.908
359: 0.9107
360: 0.9093
361: 0.9088
362: 0.9061
363: 0.9005
364: 0.9077
365: 0.9066

366: 0.9098
367: 0.906
368: 0.9048
369: 0.9073
370: 0.9062
371: 0.8861
372: 0.8831
373: 0.9064
374: 0.9095
375: 0.9054
376: 0.9088
377: 0.9093
378: 0.9065
379: 0.9087
380: 0.9061
381: 0.9072
382: 0.9086
383: 0.9092
384: 0.9088
385: 0.9095
386: 0.9095
387: 0.9015
388: 0.9048
389: 0.9093
390: 0.911
391: 0.9121
392: 0.9121
393: 0.9101
394: 0.9087
395: 0.9091
396: 0.9115
397: 0.9112
398: 0.9107
399: 0.9037
400: 0.9037
401: 0.9005
402: 0.9042
403: 0.9089
404: 0.9102
405: 0.91
406: 0.9079
407: 0.9089
408: 0.9116
409: 0.9098
410: 0.9112
411: 0.9042
412: 0.9083
413: 0.9116

414: 0.9094
415: 0.9105
416: 0.9015
417: 0.9091
418: 0.9023
419: 0.9047
420: 0.9099
421: 0.9078
422: 0.9094
423: 0.9087
424: 0.9079
425: 0.9051
426: 0.9059
427: 0.9135
428: 0.9112
429: 0.9057
430: 0.9093
431: 0.9113
432: 0.912
433: 0.9098
434: 0.9123
435: 0.9104
436: 0.9081
437: 0.9101
438: 0.912
439: 0.9012
440: 0.9051
441: 0.9095
442: 0.9085
443: 0.914
444: 0.9142
445: 0.9102
446: 0.9146
447: 0.9142
448: 0.9105
449: 0.9078
450: 0.9102
451: 0.9107
452: 0.908
453: 0.9112
454: 0.9055
455: 0.9158
456: 0.9143
457: 0.9127
458: 0.9079
459: 0.9108
460: 0.9114
461: 0.909

462: 0.9023
463: 0.9052
464: 0.9097
465: 0.9112
466: 0.911
467: 0.91
468: 0.9116
469: 0.9088
470: 0.9122
471: 0.9127
472: 0.91
473: 0.9128
474: 0.9131
475: 0.9166
476: 0.9143
477: 0.9083
478: 0.9125
479: 0.9054
480: 0.9123
481: 0.9082
482: 0.9135
483: 0.9175
484: 0.9158
485: 0.9122
486: 0.9096
487: 0.9112
488: 0.9086
489: 0.9073
490: 0.9134
491: 0.912
492: 0.9149
493: 0.9134
494: 0.9071
495: 0.9131
496: 0.9132
497: 0.9116
498: 0.9059
499: 0.9094
500: 0.9125
501: 0.9094
502: 0.9107
503: 0.9142
504: 0.9135
505: 0.9103
506: 0.915
507: 0.9124
508: 0.9097
509: 0.9111

510: 0.9151
511: 0.9148
512: 0.9145
513: 0.9117
514: 0.9141
515: 0.9147
516: 0.913
517: 0.911
518: 0.912
519: 0.9141
520: 0.9128
521: 0.9148
522: 0.9131
523: 0.9115
524: 0.9157
525: 0.9088
526: 0.9111
527: 0.909
528: 0.9159
529: 0.9138
530: 0.9144
531: 0.914
532: 0.9126
533: 0.9097
534: 0.8998
535: 0.9155
536: 0.9148
537: 0.9134
538: 0.9129
539: 0.9141
540: 0.9131
541: 0.9137
542: 0.9094
543: 0.9075
544: 0.8954
545: 0.8949
546: 0.9001
547: 0.8937
548: 0.8986
549: 0.9041
550: 0.9108
551: 0.9103
552: 0.9116
553: 0.9114
554: 0.9125
555: 0.9123
556: 0.9144
557: 0.9112

558: 0.9095
559: 0.9066
560: 0.903
561: 0.9102
562: 0.9137
563: 0.9096
564: 0.9119
565: 0.9136
566: 0.9143
567: 0.9062
568: 0.9083
569: 0.914
570: 0.9151
571: 0.9169
572: 0.9125
573: 0.9181
574: 0.9169
575: 0.9152
576: 0.9136
577: 0.9157
578: 0.9177
579: 0.9149
580: 0.9142
581: 0.917
582: 0.9146
583: 0.9135
584: 0.9106
585: 0.9107
586: 0.9068
587: 0.9115
588: 0.9139
589: 0.916
590: 0.9168
591: 0.9158
592: 0.9121
593: 0.9103
594: 0.9155
595: 0.9155
596: 0.9148
597: 0.9145
598: 0.9141
599: 0.9127
600: 0.9166
601: 0.9149
602: 0.9106
603: 0.9145
604: 0.9175
605: 0.9174

606: 0.9174
607: 0.9134
608: 0.9161
609: 0.9152
610: 0.9144
611: 0.9158
612: 0.9134
613: 0.9177
614: 0.9136
615: 0.9173
616: 0.9148
617: 0.9116
618: 0.9138
619: 0.9152
620: 0.9135
621: 0.9121
622: 0.9152
623: 0.9135
624: 0.9165
625: 0.9093
626: 0.911
627: 0.9117
628: 0.9109
629: 0.9112
630: 0.9133
631: 0.9128
632: 0.9169
633: 0.9171
634: 0.9153
635: 0.9147
636: 0.916
637: 0.9165
638: 0.9147
639: 0.9156
640: 0.9165
641: 0.9115
642: 0.9152
643: 0.9154
644: 0.9167
645: 0.9158
646: 0.915
647: 0.9148
648: 0.9159
649: 0.9148
650: 0.9153
651: 0.9154
652: 0.9171
653: 0.9173

```
654:  0.9162
655:  0.9173
656:  0.9177
657:  0.9145
658:  0.9134
659:  0.9118
660:  0.9119
661:  0.9058
662:  0.9129
663:  0.9146
664:  0.9178
665:  0.9182
666:  0.9152
667:  0.9139
668:  0.9149
669:  0.9126
670:  0.9144
671:  0.9135
672:  0.9145
673:  0.9157
674:  0.9178
675:  0.9149
676:  0.9135
677:  0.9141
678:  0.9139
679:  0.9185
680:  0.9148
681:  0.9144
682:  0.9156
683:  0.9176
684:  0.9147
685:  0.9159
686:  0.9132
687:  0.9085
688:  0.9161
689:  0.9141
690:  0.9138
691:  0.9139
692:  0.9118
693:  0.9123
694:  0.9134
695:  0.9142
696:  0.9145
697:  0.9155
698:  0.9173
699:  0.9175
700:  0.9169
701:  0.9156
```

702: 0.9158
703: 0.911
704: 0.903
705: 0.9131
706: 0.9178
707: 0.9156
708: 0.916
709: 0.9155
710: 0.9186
711: 0.9142
712: 0.915
713: 0.9183
714: 0.9188
715: 0.9173
716: 0.9179
717: 0.9167
718: 0.9168
719: 0.9149
720: 0.9187
721: 0.9151
722: 0.9183
723: 0.9159
724: 0.9185
725: 0.9175
726: 0.9178
727: 0.9167
728: 0.9173
729: 0.9186
730: 0.9176
731: 0.9173
732: 0.9177
733: 0.9172
734: 0.9151
735: 0.9187
736: 0.9169
737: 0.9176
738: 0.9033
739: 0.9134
740: 0.9163
741: 0.9186
742: 0.9158
743: 0.917
744: 0.9182
745: 0.9188
746: 0.918
747: 0.9136
748: 0.9151
749: 0.9184

750: 0.9175
751: 0.9161
752: 0.9136
753: 0.9132
754: 0.9153
755: 0.9168
756: 0.9158
757: 0.9142
758: 0.9155
759: 0.9169
760: 0.9162
761: 0.9184
762: 0.9199
763: 0.9181
764: 0.9172
765: 0.916
766: 0.917
767: 0.9141
768: 0.9174
769: 0.9167
770: 0.9167
771: 0.9144
772: 0.9162
773: 0.9162
774: 0.9165
775: 0.9191
776: 0.9153
777: 0.9175
778: 0.9167
779: 0.9168
780: 0.9177
781: 0.9171
782: 0.9178
783: 0.918
784: 0.9143
785: 0.9183
786: 0.9149
787: 0.916
788: 0.9135
789: 0.9161
790: 0.9177
791: 0.9137
792: 0.916
793: 0.9108
794: 0.9149
795: 0.9148
796: 0.9142
797: 0.9141

798: 0.9126
799: 0.918
800: 0.9186
801: 0.9151
802: 0.9145
803: 0.9152
804: 0.9185
805: 0.9189
806: 0.915
807: 0.909
808: 0.9145
809: 0.9136
810: 0.9109
811: 0.9169
812: 0.9171
813: 0.9107
814: 0.9159
815: 0.9163
816: 0.9166
817: 0.9183
818: 0.9144
819: 0.9172
820: 0.917
821: 0.9165
822: 0.9165
823: 0.9163
824: 0.9124
825: 0.915
826: 0.917
827: 0.9154
828: 0.9163
829: 0.9138
830: 0.9135
831: 0.9151
832: 0.9155
833: 0.9176
834: 0.9174
835: 0.9186
836: 0.9176
837: 0.9168
838: 0.919
839: 0.9169
840: 0.9178
841: 0.9169
842: 0.9139
843: 0.9099
844: 0.9095
845: 0.9142

846: 0.9134
847: 0.9161
848: 0.916
849: 0.9185
850: 0.9153
851: 0.9128
852: 0.9141
853: 0.9133
854: 0.9159
855: 0.9179
856: 0.9146
857: 0.9134
858: 0.916
859: 0.9161
860: 0.9174
861: 0.9167
862: 0.9183
863: 0.9166
864: 0.9171
865: 0.9169
866: 0.9165
867: 0.9143
868: 0.9185
869: 0.9163
870: 0.9191
871: 0.918
872: 0.92
873: 0.9222
874: 0.9212
875: 0.9179
876: 0.9191
877: 0.9182
878: 0.9174
879: 0.9208
880: 0.9189
881: 0.917
882: 0.9159
883: 0.9122
884: 0.9149
885: 0.9142
886: 0.9168
887: 0.9132
888: 0.9157
889: 0.9155
890: 0.9141
891: 0.9179
892: 0.9197
893: 0.9152

894: 0.9184
895: 0.9183
896: 0.9173
897: 0.9172
898: 0.9157
899: 0.9194
900: 0.9175
901: 0.9171
902: 0.9206
903: 0.917
904: 0.9165
905: 0.9175
906: 0.9167
907: 0.9174
908: 0.9168
909: 0.9169
910: 0.9171
911: 0.9187
912: 0.9161
913: 0.9205
914: 0.918
915: 0.919
916: 0.918
917: 0.9177
918: 0.9191
919: 0.9155
920: 0.9143
921: 0.9168
922: 0.9142
923: 0.9154
924: 0.9156
925: 0.9168
926: 0.9172
927: 0.917
928: 0.9156
929: 0.9151
930: 0.9185
931: 0.9159
932: 0.9142
933: 0.9144
934: 0.9194
935: 0.9138