Foundations of Data Science - Homework 3 Solution.

Mohammad Afzal Shadab (ms82697)

mashadab@utexas.edu

Gradient Descent and Stochastic Gradient Descent.

Due Wednesday, March 24, 2021

Lecture 7.

1. **Weighted least squares** The *weighted* least squares problem is a generalization of ordinary least squares:

$$\min_{\mathbf{x}\in\mathbb{R}^d} \|\mathbf{W}\cdot\mathbf{A}\mathbf{x} - \mathbf{W}\cdot\mathbf{b}\|_2^2,$$

where $\mathbf{W}$ is a diagonal matrix of positive weights

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & w_n \end{pmatrix}$$

Suppose $\mathbf{A}$ is an $n \times d$ overdetermined full-rank matrix. Suppose $\mathbf{b} = \mathbf{A}\mathbf{x} + \epsilon$, and the noise vector $\epsilon = (\epsilon_i)_{i=1}^n$ is assumed to be a random vector whose components $\epsilon_1, \dots, \epsilon_n$ are independent, mean-zero, and have variances $\sigma_1^2, \dots, \sigma_n^2$.

(a) Under this model, derive the natural choice of weights in the weighted least squares problem, and justify your answer.

(b) Derive the closed-form expression for the solution to the weighted least squares problem.

Answer: **(a)** This is a problem involving *heteroscedasticity* as variance in the errors is a not constant. Here,

$$\mathbf{b} = \mathbf{A}\mathbf{x} + \boldsymbol{\epsilon} \tag{0.1}$$

$$\text{Cov}(\boldsymbol{\epsilon}) = \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_n^2 \end{bmatrix}_{n\times n} \tag{0.2}$$

Let's first find the expectation of the error,

$$\begin{aligned}
\mathbb{E}[||\mathbf{W}\cdot\mathbf{A}\mathbf{x}-\mathbf{W}\cdot\mathbf{b}||_2^2] &= \mathbb{E}[||\cancel{\mathbf{W}\cdot\mathbf{A}\mathbf{x}}-\mathbf{W}\cdot(\cancel{\mathbf{A}\mathbf{x}}+\boldsymbol{\epsilon})||_2^2] && (0.3)\\
&= \mathbb{E}[||\mathbf{W}\cdot\boldsymbol{\epsilon}||_2^2] && (0.4)\\
&= \mathbb{E}[\sum_{i=1}^{n} w_i^2\sigma_i^2] && (0.5)\\
&= \sum_{i=1}^{n} w_i^2\mathbb{E}[\epsilon_i^2] && (0.6)\\
&= \sum_{i=1}^{n} w_i^2\sigma_i^2 && (0.7)
\end{aligned}$$

It is clear from the above relation that we can safely define the reciprocal of square root of each variance σ_i^2, as the weight $w_i = \frac{1}{\sigma_i}$, then let matrix $\mathbf{W}$ be a diagonal matrix containing these weights:

$$\mathbf{W} = \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_n \end{bmatrix}_{n\times n} \Rightarrow \mathbf{W} = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}_{n\times n} \quad (0.8)$$

The justification of this choice can be summarized with the following observations:

- Since each weight is inversely proportional to the error variance, it reflects the information in that observation. So, an observation with small error variance has a large weight since it contains relatively more information than an observation with large error variance (small weight).
- The weights have to be known (or more usually estimated) up to a proportionality constant.

(b) The minimization problem can be written as follows

$$G(\mathbf{x}) \;=\; (\mathbf{W}\cdot\mathbf{A}\mathbf{x}-\mathbf{W}\cdot\mathbf{b})^T(\mathbf{W}\cdot\mathbf{A}\mathbf{x}-\mathbf{W}\cdot\mathbf{b}) \quad (0.9)$$

Taking the gradient with respect to $\mathbf{x}$, we get

$$\mathbf{G}'(\mathbf{x}) = (\mathbf{W}\cdot\mathbf{A})^T(\mathbf{W}\cdot\mathbf{A}\mathbf{x}-\mathbf{W}\cdot\mathbf{b}) \quad (0.10)$$

At the extremum, $\mathbf{G}'(\mathbf{x}) = \mathbf{0}$. Therefore,

$$\begin{aligned}
\mathbf{0} &= (\mathbf{W}\cdot\mathbf{A})^T(\mathbf{W}\cdot\mathbf{A}\mathbf{x}-\mathbf{W}\cdot\mathbf{b}) && (0.11)\\
\Rightarrow \mathbf{x} &= (\mathbf{A}^T\mathbf{W}^*\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}^*\mathbf{b} && (0.12)
\end{aligned}$$

where $\mathbf{W}^*$ is $\mathbf{W}^T\mathbf{W}$.

2. **Gradient descent for Least Squares.** In lecture, we proved that gradient descent with constant step-size converges for the least squares problem, provided $\eta \leq 1/\lambda_{\max}(\mathbf{A}^T\mathbf{A})$.

(a) Extend the proof to show that gradient descent with constant step-size converges for the least squares problem when $\eta < \frac{2}{\lambda_{\max}}(\mathbf{A}^T\mathbf{A})$.

(b) Argue that the condition $\eta < \frac{2}{\lambda_{\max}(\mathbf{A}^T\mathbf{A})}$ is also necessary for gradient descent to converge by constructing an example of a least squares problem for which gradient descent does not converge when $\eta = \frac{2}{\lambda_{\max}(\mathbf{A}^T\mathbf{A})}$.

Answer: (a) Continuing the derivation from,

$$||\mathbf{I} - \eta\mathbf{A}^T\mathbf{A}||_2 < 1 \tag{0.13}$$

Now, the eigendecomposition of $\mathbf{A}^T\mathbf{A}$ is $\mathbf{U}^T\mathbf{\Lambda}\mathbf{U}$ with real and positive eigen values. Then eigen decomposition of $\mathbf{I} - \eta\mathbf{A}^T\mathbf{A} = \mathbf{U}^T(\mathbf{I} - \eta\mathbf{\Lambda})\mathbf{U}^T$. So, Equation (0.13) can be re-written as:

$$\begin{aligned}
|1 - \eta\lambda(\mathbf{A}^T\mathbf{A})) &< 1 \quad (0.14)\\
-1 < (1 - \eta\lambda(\mathbf{A}^T\mathbf{A})) &< 1 \quad (0.15)\\
\Rightarrow \eta &< 2/\lambda_{max}(\mathbf{A}^T\mathbf{A}) \quad (0.16)
\end{aligned}$$

as $\lambda_{max}(\mathbf{A}^T\mathbf{A})) > 0$s. For this case, $||\mathbf{U}(\mathbf{I} - \eta\mathbf{\Lambda})\mathbf{U}^T)|| < 1$ and the gradient update is a contraction mapping which must converge to a fixed point. □

(b) Let's try to find the best fit for line $y = mx$ through data points $(5,5), (-5,5)$. The corresponding least square problem can be then be written as,

$$\underset{m}{\text{argmin}}\, ||mx - y||^2 \tag{0.17}$$

$$\text{argmin}\left|\left| m\begin{pmatrix}5\\-5\end{pmatrix} - \begin{pmatrix}5\\5\end{pmatrix}\right|\right|^2 \tag{0.18}$$

The exact solution of this problem is $m = 0$ since the line is parallel to x-axis. In this problem, matrix $\mathbf{A} = \begin{pmatrix}5\\-5\end{pmatrix}$ and vector $\mathbf{b} = \begin{pmatrix}5\\5\end{pmatrix}$. So, $\mathbf{A}^T\mathbf{A} = 25$. Therefore, $\lambda_{max}(\mathbf{A}^T\mathbf{A}) = 25$. So, $\eta = 2/\lambda_{max}(\mathbf{A}^T\mathbf{A}) = 2/25 = 0.08$. So, the gradient descent iteration then becomes

$$\begin{aligned}
m_{t+1} &= m_t - \eta\mathbf{A}^T(\mathbf{A}m - \mathbf{b}) \quad (0.19)\\
&= m_t - 0.08 * (5,-5)\left(\begin{pmatrix}5\\-5\end{pmatrix} m - \begin{pmatrix}5\\5\end{pmatrix}\right) \quad (0.20)\\
&= m_t - 2m_t = -m_t \quad (0.21)
\end{aligned}$$

So, m_t oscillates in between -1 and $+1$ instead of converging to its true value 0 □

Lecture 8.

3. **Neural networks and Machine Learning.** Consider a two-layer neural network model of the form

$$f(\mathbf{W}, \mathbf{a}, \mathbf{x}) = \sum_{r=1}^{m} a_r \sigma(\langle \mathbf{w}_r, \mathbf{x} \rangle) \tag{0.22}$$

with ReLU activation function $\sigma(x) = \max(0, x)$.

Fixing the second layer weights $\mathbf{a}$ and only training the first layer weights $\mathbf{W} = (\mathbf{w}_r)_{r=1}^{m} \in \mathbb{R}^{m \times d}$ using least squares loss, the optimization problem is

$$\min_{\mathbf{W}} L(\mathbf{W}) = \min_{\mathbf{W}} \frac{1}{2} \cdot \frac{1}{n} \sum_{j=1}^{n} (f(\mathbf{W}, \mathbf{a}, \mathbf{x}_j) - y_j)^2 \tag{0.23}$$

(a) Derive an expression for the partial gradient $\frac{\partial L(\mathbf{W})}{\partial \mathbf{w}_r} \in \mathbb{R}^d$, where among subgradients of the ReLU activation $\sigma(x)$ at $x = 0$, we define $\left.\frac{d\sigma}{dx}\right|_{x=0} = 1$

(b) In neural network training, the weights $\mathbf{w}_r$ are initialized in gradient descent as independent spherical Gaussian vectors. Show that

$$\mathbb{E}_{\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d \times d})} \left[\mathbf{x}_i^T \mathbf{x}_j \mathbb{I}\{\mathbf{w}^T \mathbf{x}_i \geq 0, \mathbf{w}^T \mathbf{x}_j \geq 0\} \right] = \mathbf{x}_i^T \mathbf{x}_j \frac{\pi - \arccos(\mathbf{x}_i^T \mathbf{x}_j)/(||\mathbf{x}_i||\ ||\mathbf{x}_j||)}{2\pi}.$$

(c) Let $\mathbf{H} \in \mathbb{R}^{n \times n}$ be the matrix with entries

$$H_{i,j} = \mathbb{E}_{\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d \times d})} \left[\mathbf{x}_i^T \mathbf{x}_j \mathbb{I}\{\mathbf{w}^T \mathbf{x}_i \geq 0, \mathbf{w}^T \mathbf{x}_j \geq 0\} \right]$$

where for events E_1 and E_2, $\mathbb{I}(E_1, E_2) = 1$ if E_1 and E_2 are true, and $\mathbb{I}(E_1, E_2) = 0$ else.

- When the training data points $\mathbf{x}_j \in \mathbb{R}^d$ are orthonormal, what does the matrix $\mathbf{H}$ reduce to?
- Argue that the condition number of $\mathbf{H}$ is good if the data points are sufficiently almost orthonormal, $\max_{j,k} \frac{|\langle \mathbf{x}_j, \mathbf{x}_k \rangle|}{\|\mathbf{x}_j\|_2 \|\mathbf{x}_k\|_2} \leq \epsilon$.
- What do these computations suggest about the benefit of randomly initialized weights and well distributed training data points at initialization of gradient descent in neural network training?

Answer: **(a)** Taking the partial derivative of Equation (0.23) with respect to $\mathbf{w}_r$ using chain rule, we get

$$\frac{\partial L(\mathbf{W})}{\partial \mathbf{w}_r} = \frac{1}{n} \sum_{j=1}^{n} (f(\mathbf{W}, \mathbf{a}, \mathbf{x}_j) - y_j) \cdot \nabla_{\mathbf{w}_r} f(\mathbf{W}, \mathbf{a}, \mathbf{x}_j) \tag{0.24}$$

where

$$\nabla_{\mathbf{w}_r} f(\mathbf{W}, \mathbf{a}, \mathbf{x}_j) = \begin{cases} \mathbf{0}, & \langle \mathbf{w}_r, \mathbf{x} \rangle < 0 \\ \mathbf{a}_r \mathbf{x}, & \langle \mathbf{w}_r, \mathbf{x} \rangle \geq 0 \end{cases} \tag{0.25}$$

by using $\sigma(x) = \max(0, x)$ and $\left.\frac{d\sigma}{dx}\right|_{x=0} = 1$.

(b)

$$\mathbb{E}_{\mathbf{w}\sim\mathcal{N}(\mathbf{0},\mathbf{I}_{d\times d})}\left[\mathbf{x}_i^T\mathbf{x}_j\mathbb{I}\{\mathbf{w}^T\mathbf{x}_i \geq 0, \mathbf{w}^T\mathbf{x}_j \geq 0\}\right] = \tag{0.26}$$

$$\begin{aligned}
&= \mathbf{x}_i^T\mathbf{x}_j\mathbb{E}_{\mathbf{w}\sim\mathcal{N}(\mathbf{0},\mathbf{I}_{d\times d})}\left[\mathbb{I}\{\mathbf{w}^T\mathbf{x}_i \geq 0, \mathbf{w}^T\mathbf{x}_j \geq 0\}\right] && (0.27)\\
&= \mathbf{x}_i^T\mathbf{x}_j(1 \times P(\mathbf{w}^T\mathbf{x}_i \geq 0, \mathbf{w}^T\mathbf{x}_j \geq 0)) && (0.28)\\
&= \mathbf{x}_i^T\mathbf{x}_jP(\mathbf{w}^T\mathbf{x}_i \geq 0, \mathbf{w}^T\mathbf{x}_j \geq 0) && (0.29)
\end{aligned}$$

The approach we will take is to *(i) project the vectors on a 2D plane* $\mathbf{w}_{||}$ containing $\mathbf{x}_i$ and $\mathbf{x}_j$, *(ii) prove that* $\mathbf{w}_{||}$ *has a normal distribution* and is *(iii) spherically symmetric in the orthonormal basis to use angles*, and finally *(iv) using the spherically symmetry to express the probability as a ratio of intersecting angle and* 2π.

(i.) To calculate the probability $P(\mathbf{w}^T\mathbf{x}_i \geq 0, \mathbf{w}^T\mathbf{x}_j \geq 0)$, we can consider $\mathbf{w}$ to be consisted of 2 planes, one plane $\mathbf{w}_{||}$ contains both the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ with normal in the direction of $\hat{\mathbf{e}}_1 = \frac{\mathbf{x}_i\times\mathbf{x}_j}{||\mathbf{x}_i\times\mathbf{x}_j||}$ and the other plane $\mathbf{w}_\perp$. The projection of $\mathbf{w}$ on $\mathbf{w}_{||}$ is

$$\mathbf{w}_{||} = \mathbf{w} - (\mathbf{w}\cdot\hat{\mathbf{e}}_1)\hat{\mathbf{e}}_1 \tag{0.30}$$

(ii.) From the previous homework problem, we can conclude that the components of $\mathbf{w}_{||}$ are zero-mean Gaussians. However, the variance is not 1 and covariance is not a diagonal matrix.

$$\begin{aligned}
\mathbb{E}[\mathbf{w}_{||}] &= \mathbb{E}[\mathbf{w} - (\mathbf{w}\cdot\hat{\mathbf{e}}_1)\hat{\mathbf{e}}_1] && (0.31)\\
&= \mathbb{E}[\mathbf{w}] - (\mathbb{E}[\mathbf{w}]\cdot\hat{\mathbf{e}}_1)\hat{\mathbf{e}}_1) = 0 && (0.32)\\
\mathrm{Cov}(\mathbf{w}_{||}) &= \mathbb{E}[(\mathbf{w} - (\mathbf{w}\cdot\hat{\mathbf{e}}_1)\hat{\mathbf{e}}_1)(\mathbf{w} - (\mathbf{w}\cdot\hat{\mathbf{e}}_1)\hat{\mathbf{e}}_1)^T] && (0.33)\\
&= \mathbf{I}_{d\times d} - 2\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T && (0.34)\\
&= \mathbf{I}_{d\times d} - \hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T && (0.35)
\end{aligned}$$

Therefore, $\mathbf{w}_{||} = \mathcal{N}(0, \mathrm{Cov}(\mathbf{w}_{||}))$.

(iii.) So, we need to express $\mathbf{w}_{||}$ in orthonormal basis as $\mathbf{x}_i$ and $\mathbf{x}_j$ are not orthogonal. So, let $\hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3$ be orthonormal bases in this plane such that $\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3$ are mutually perpendicular.

$$\mathbf{w}_{||} = c_2\hat{\mathbf{e}}_2 + c_3\hat{\mathbf{e}}_3 = [\hat{\mathbf{e}}_2 \quad \hat{\mathbf{e}}_3]\mathbf{c} \tag{0.36}$$

where c_2 and c_3 are the coefficients and $\mathbf{c} = [c_2 \quad c_3]^T$. We can easily see that $\mathbf{c} = [\hat{\mathbf{e}}_2 \quad \hat{\mathbf{e}}_3]^T\mathbf{w}_{||}$. Therefore,

$$\mathbb{E}(\mathbf{c}) = \mathbb{E}([\hat{\mathbf{e}}_2 \quad \hat{\mathbf{e}}_3]^T\mathbf{w}_{||}) = \mathbf{0} \tag{0.37}$$

Since $\mathbf{c}$ a combination of Gaussian variables, it is therefore a Gaussian.

$$\begin{aligned}
\mathrm{Cov}(\mathbf{c}) &= \mathbb{E}[\mathbf{c}\mathbf{c}^T] && (0.38)\\
&= [\hat{\mathbf{e}}_2 \quad \hat{\mathbf{e}}_3]^T\mathbb{E}[\mathbf{w}_{||}\mathbf{w}_{||}^T][\hat{\mathbf{e}}_2 \quad \hat{\mathbf{e}}_3] && (0.39)\\
&= [\hat{\mathbf{e}}_2 \quad \hat{\mathbf{e}}_3]^T(\mathbf{I} - \hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T)[\hat{\mathbf{e}}_2 \quad \hat{\mathbf{e}}_3] && (0.40)\\
&= (\mathbf{I}_{2\times 2} - 0) && (0.41)\\
&= \mathbf{I}_{2\times 2} && (0.42)
\end{aligned}$$

Therefore, $\mathbf{c} = \mathcal{N}(0, \mathbf{I}_{2\times 2})$ and therefore, $\mathbf{w}_{||}$ has spherically symmetric distribution in 2D plane as $\mathbf{c}$ is just $\mathbf{w}_{||}$ expressed in a different basis.

(iv.) Now, we can now work in terms of angles. $\mathbf{w}_{||}$ can lie in between 0 to 2π. If $\mathbf{x}_i$ and $\mathbf{x}_j$ include an angle $\arccos(\mathbf{x}_i^T\mathbf{x}_j/(||\mathbf{x}_i||\;||\mathbf{x}_j||))$, the zone where $(\mathbf{w}^T\mathbf{x}_i \geq 0, \mathbf{w}^T\mathbf{x}_j \geq 0)$ is the true includes and angle $\pi - \arccos(\mathbf{x}_i^T\mathbf{x}_j)$.

Therefore,

$$P(\mathbf{w}^T\mathbf{x}_i \geq 0, \mathbf{w}^T\mathbf{x}_j \geq 0) = P(\mathbf{w}_{||}^T\mathbf{x}_i \geq 0, \mathbf{w}_{||}^T\mathbf{x}_j \geq 0) = \frac{\pi - \arccos(\mathbf{x}_i^T\mathbf{x}_j)/(||\mathbf{x}_i||\;||\mathbf{x}_j||)}{2\pi} \tag{0.43}$$

Plugging this result in Equation (0.29), we get

$$\mathbb{E}_{\mathbf{w}\sim\mathcal{N}(\mathbf{0},\mathbf{I}_{d\times d})}\left[\mathbf{x}_i^T\mathbf{x}_j\mathbb{I}\{\mathbf{w}^T\mathbf{x}_i \geq 0, \mathbf{w}^T\mathbf{x}_j \geq 0\}\right] = \mathbf{x}_i^T\mathbf{x}_j\frac{\pi - \arccos(\mathbf{x}_i^T\mathbf{x}_j/(||\mathbf{x}_i||\;||\mathbf{x}_j||))}{2\pi} \tag{0.44}$$ □

(c) *(i.)* When data is orthonormal,

$$\mathbf{x}_i^T\mathbf{x}_j = \begin{cases} 1, & i \neq j \\ 0, & i = j \end{cases} \tag{0.45}$$

and $\arccos(1) = 0$. So, plugging it in relation given for $\mathbf{H}_{i,j}$, we get a diagonal matrix as a result with entries $H_{j,j} = \frac{1}{2}$ for $1 \leq j \leq n$.

(ii.) For this case, the entries of $\mathbf{H}$ satisfy

$$\mathbf{H}_{ij}\begin{cases} = \frac{1}{2}, & i = j \\ \leq \epsilon \cdot \frac{\pi - \arccos(\epsilon)}{2\pi}, & i \neq j \end{cases} \tag{0.46}$$

We can observe $\mathbf{H}$ is a diagnonally-dominant symmetric matrix. More specifically, the sum of the absolute values of the off-diagonal entries of the k^{th} row R_i satisfies

$$R_i \leq \frac{\epsilon}{2(n-1)} \tag{0.47}$$

Using the Gershgorin circle theorem, the eigenvalues of matrix $\mathbf{H}$ must lie in the disk $D\big(\frac{1}{2}, \frac{\epsilon}{2(n-1)}\big)$. Thus for sufficiently small ϵ, the conditioning number of the normal matrix $\mathbf{H}$ is small and $\mathbf{H}$ is well conditioned.

(iii.) We observe from part (a) that $\mathbf{H}$ is related to the Hessian of the objective function through a linear transformation. These computations therefore suggest that randomly initialized weights and well distributed training data points lead to a well conditioned Hessian, known to lead the the gradient descent algorithm to a better convergence.

Lecture 9.

4. Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is a differentiable function satisfying the PL-inequality:

$$\frac{1}{2}\|\nabla f(w)\|^2 \geq \mu(f(w) - f^*), \quad \forall w.$$

A function is called *invex* if it is differentiable and there exists a vector valued function η such that for any x and y in $\mathbb{R}^d$, the following inequality holds:

$$f(y) \geq f(x) + \nabla f(x)^T\eta(x,y). \tag{0.48}$$

(a) Show that f as above is necessarily an invex function.

(b) Argue that any stationary point of an invex function is a global minimizer. (Hence, any stationary point of a differentiable PL function is a global minimizer.)

Answer (a) To prove that *functions satisfying PL inequality implies invexity.* We first need to prove that *PL inequality implies all every stationary points are minimizers.* And then *Everg stationary point of f is a global minimizer and then also an invex function.*

Theorem 1: Functions f satisfying PL inequality implies that all stationary points are global minimizer.
Proof by contradiction: PL inequality

$$\frac{1}{2}\|\nabla f(w)\|^2 \geq \mu(f(w) - f^*), \quad \forall w \tag{0.49}$$

For any extremum, $\overline{x}, \nabla f = 0$. Therefore, Equation (0.49) becomes,

$$0 \quad \geq \quad \mu(f(\overline{x}) - f^*) \tag{0.50}$$
$$\tag{0.51}$$

As $\mu \geq 0$ and $(f(\overline{x}) - f^*) \geq 0$,

$$f(\overline{x}) = f^* \tag{0.52}$$
$$\tag{0.53}$$

Therefore, all stationary points of function satisfying PL inequality are global minimizers.

Theorem 2: If every stationary point is a global minimizer, then the function is invex.
Proof by construction: Assuming that every stationary point of f is a global minimizer. If $\nabla f(y) = 0$, letting $\eta(x, y) = 0$. But if $\nabla(x, y) \neq 0$, we can choose,

$$\eta(x, y) = \frac{(f(x) - f(y))}{||f(y)||^2}\nabla f(y) \tag{0.54}$$

With above construction, f satisfies the definition of invex function $\forall x, y$.

Final proof for Part (a): Theorem 1 implies that stationary points of functions f satisfying the PL inequality are global minimizers. Theorem 2 implies that if every stationary point is a global minimizer, then the function is invex. Combining both, functions satisfying PL inquality is necessarily an invex function. □

(b) At the stationary points $\overline{x}$, $\nabla f(\overline{x}) = 0$, therefore Equation (0.48) becomes

$$f(y) \geq f(\overline{x}) + 0 \tag{0.55}$$
$$\Rightarrow f(y) \geq f(\overline{x}), \forall y \in \operatorname{dom}(f) \tag{0.56}$$

Therefore, $\overline{x}$ is a global minimizer. □

Lecture 10.

5. Suppose f is a C^2 function, convex, and suppose ∇f is L-Lipschitz. Let x be a minimizer of f. Prove that for any w,

$$\|\nabla f(w) - \nabla f(x)\|^2 \leq 2L(f(w) - f(x))$$

Proof: Using Taylor expansion of $f(x)$ around w,

$$f(x) = f(w) + \langle \nabla f(w), w - x\rangle + \frac{1}{2}(w-x)^T \nabla^2 f(w)(w-x) \tag{0.57}$$

Using the L-smoothness of f, i.e., Lipschitz continuity of ∇f, the Hessian $\nabla^2 f(w) \leq L\mathbf{I}$. Therefore, we get

$$f(x) \leq f(w) + \langle \nabla f(w), w - x\rangle + \frac{L}{2}||w-x||^2 \tag{0.58}$$

Now, let's minimize $f(x)$ w.r.t. x. The LHS will become $f(x)$ as it given as a minimizer of f. To find the minimizer of RHS, let's take its gradient with respect to x and equate it to 0. So, we get

$$\nabla_x \left[f(w) + \langle \nabla f(w), w - x\rangle + \frac{L}{2}||w-x||^2 \right] = 0 \tag{0.59}$$

As the first term is independent of x, it simplifies to

$$0 + \nabla_x \left[\langle \nabla f(w), w - x\rangle + \frac{L}{2}||w-x||^2 \right] \;=\; 0 \tag{0.60}$$

$$\Rightarrow \nabla f(w) + L(w-x) \;=\; 0 \tag{0.61}$$

$$\Rightarrow w - x \;=\; -\frac{1}{L}\nabla f(w) \tag{0.62}$$

Plugging Equation (0.62) into Equation (0.58), we get

$$f(x) \;\leq\; f(w) + \langle \nabla f(w), -\frac{1}{L}\nabla f(w)\rangle + \frac{L}{2}\left|\left| -\frac{1}{L}\nabla f(w)\right|\right|^2 \tag{0.63}$$

$$\Rightarrow f(x) \;\leq\; f(w) - \frac{1}{L}||\nabla f(w)||^2 + \frac{1}{2L}||\nabla f(w)||^2 \tag{0.64}$$

$$\Rightarrow f(x) \;\leq\; f(w) - \frac{1}{2L}||\nabla f(w)||^2 \tag{0.65}$$

$$\Rightarrow ||\nabla f(w)||^2 \;\leq\; 2L(f(w) - f(x)) \tag{0.66}$$

$$\Rightarrow ||\nabla f(w) - \nabla f(x)||^2 \;\leq\; 2L(f(w) - f(x)), \quad \text{as } \nabla f(x) = 0 \quad \square \tag{0.67}$$

6. Suppose that f is C^2 and μ-strongly convex. Suppose w_* is the unique minimizer of f. Prove that for any w,

$$\langle w - w_*, \nabla f(w)\rangle = \langle w - w_*, \nabla f(w) - \nabla f(w_*)\rangle \geq \mu\|w - w_*\|_2^2 \tag{0.68}$$

Proof: Since w_* is unique minimizer of $f \Rightarrow \nabla f(w_*) = 0$. Therefore, the first equality of equation (0.68) satisfies trivially. Now, using Taylor expansion for $\mu-$strongly convex function $f(w_*)$ around w:

$$f(w_*) \leq f(w) + \langle \nabla f(w), w_* - w\rangle + \frac{\mu}{2}||w - w_*||^2 \tag{0.69}$$

$$\langle \nabla f(w), w_* - w\rangle \geq -(f(w) - f(w_*)) + \frac{\mu}{2}||w - w_*||^2 \tag{0.70}$$

Since $(f(w) - f(w_*)) \geq 0$ and $\frac{\mu}{2}||w - w_*||^2 \geq 0$, we can write

$$f(w_*) \leq f(w) + \langle \nabla f(w), w_* - w \rangle + \frac{\mu}{2}||w - w_*||^2 \tag{0.71}$$

$$\langle \nabla f(w), w - w_* \rangle \geq (f(w) - f(w_*)) + \frac{\mu}{2}||w - w_*||^2 \tag{0.72}$$

Now, Taylor expanding $f(w)$ around the unique minimizer w_*,

$$f(w) \leq f(w_*) + \langle \cancelto{0}{\nabla f(w_*)}, w_* - w \rangle + \frac{\mu}{2}||w_* - w||^2 \tag{0.73}$$

$$-(f(w) - f(w_*)) \geq -\frac{\mu}{2}||w_* - w||^2 \tag{0.74}$$

Plugging result (0.74) in Equation (0.72), we get

$$\langle \nabla f(w), w - w_* \rangle \geq \mu||w - w_*||^2 \tag{0.75}$$

As $\nabla f(w_*) = 0$,

$$\Rightarrow \langle \nabla f(w) - \nabla f(w_*), w - w_* \rangle \geq \mu||w - w_*||^2 \quad \square \tag{0.76}$$

7. **Stochastic Gradient Descent.** Implement stochastic gradient descent for the least squares loss

$$f(w) = \frac{1}{2}\|Aw - b\|_2^2.$$

Provide convergence plots to validate the convergence guarantee for SGD provided in class. In particular, compare empirical and theoretical convergence rates for the following example: $A \in \mathbb{R}^{10,000 \times 1,000}$ has i.i.d. $\mathcal{N}(0, 1/\sqrt{1,000})$ Gaussian entries, $b = Ax + \epsilon$ for $x = (1, 1, \ldots, 1)$, and ϵ has i.i.d. Gaussian entries with variances $0, .01, .1$, and 1. Repeat the comparison, but now consider $A \in \mathbb{R}^{10,000 \times 1,000}$ whose jth row a_j has i.i.d. $\mathcal{N}(0, 1/\sqrt{j \cdot 1,000})$ Gaussian entries.

Answer: For implementation, we first used a random zero-mean and unit variance Gaussian vector, $x_0 \sim \mathcal{N}(0, 1)$. We also used a fixed permutation of the row indices to iterate through the component functions $f_j(w)$ using Stochastic indexing.

The plots and the corresponding codes are provided for both cases in the next few pages. For the first case, shown in Figure 1, it is clear that the computed mean square errors (MSE) are lower than the theoretical MSE. In this case as well, the computed MSE is significantly lower than the theoretical MSE.

For the second case, shown in Figure 2, the convergence rate is much lower for all four choices of the random noise σ. This is likely due to the fact that the magnitude of the gradient updates vary widely from row to row due to the dependence on j of the variance of the entries of A.

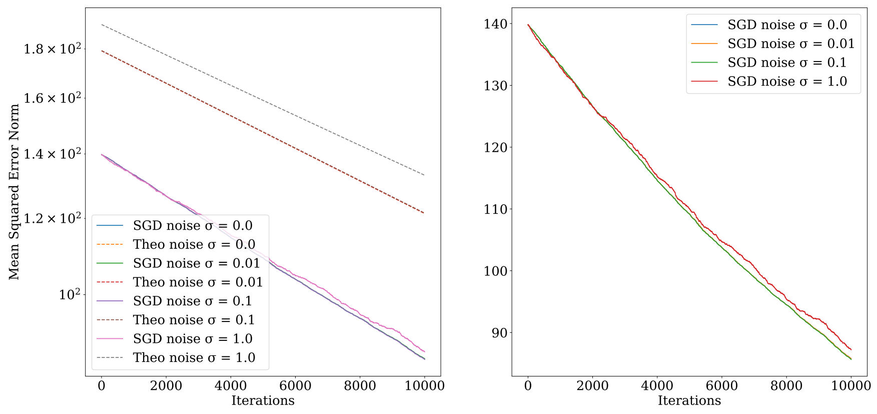

Figure 1: Evolution of MSE with number of iterations $\mathcal{A} \in \mathbb{R}^{10000\times 1000}$ i.i.d. $\mathcal{N}(0, 1/\sqrt{1000})$ Gaussian entries. Left subplot includes the theoretical estimates of the noise while the right subplot focuses mainly on SGD noise.

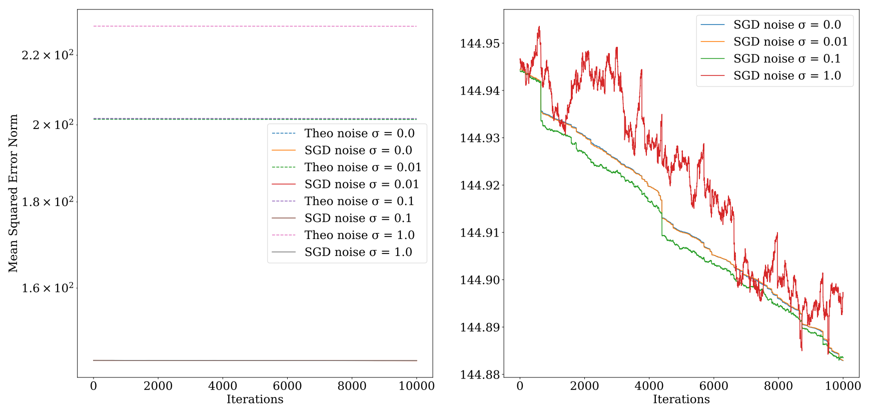

Figure 2: Evolution of MSE with number of iterations $\mathcal{A} \in \mathbb{R}^{10000\times 1000}$ i.i.d. $\mathcal{N}(0, 1/\sqrt{j \cdot 1000})$ Gaussian entries. Left subplot includes the theoretical estimates of the noise while the right subplot focuses mainly on SGD noise.

Code: The complete program consists of a main file and a functions file as given below.
Main program:

```
#class    : Foundations of ML and Data Science
#homework: 3
#question: 7
#author   : Mohammad Afzal Shadab
#email    : mashadab@utexas.edu

#Stochastic Gradient Descent
import numpy as np
import matplotlib.pyplot as plt
from hw3q7SGD_functions import generate_random_matrix, theoretical_conv
                                    , stochastic_index_compute
plt.rcParams["font.family"] = "Serif"

# Part (a)
n   = 10000        #number of data points
d   = 100          #dimensions
max_iters = 10000 #maximum iteration
mu_data = np.zeros((n,1)) #vector of means
sigma_data =  np.ones((n,1)) / np.sqrt(1000) #vector of std dev
A = generate_random_matrix(mu_data,sigma_data,d) #Random matrix n X d
x_opt = np.ones((d,1)) #optimal value of x
x_init_guess = np.random.normal(0,1,x_opt.shape) #initial guess
t = np.arange(max_iters)

x_iters = np.zeros((max_iters,d))
x_iters[0,:] = np.reshape(np.random.normal(0,1,x_opt.shape),(-1))

stochastic_index = stochastic_index_compute(n,max_iters) #### sigma^2
                                    from notes stochastic_index

L = n * np.max(np.diagonal(np.transpose(A) @ A))
alpha = 1/(2*L)
epsilon_sigma = np.array([0.0,0.01,0.1,1.0])

fig, ax = plt.subplots(1,2, figsize=(10,7.5),dpi=80)
plt.subplots_adjust(hspace = 0.001)
ax[0].set_xlabel('Iterations')
ax[1].set_xlabel('Iterations')
ax[0].set_ylabel('Mean Squared Error Norm')
ax[0].set_yscale('log')
for i in range(len(epsilon_sigma)):
    epsilon = np.random.normal(0,epsilon_sigma[i], (n,1))
    b = A @ x_opt + epsilon

    theoretical_err_norm_sq = theoretical_conv(A,b,x_opt,x_init_guess,
                                        max_iters,L)
    err_norm_sq = np.zeros((max_iters,1))
    err_norm_sq[0] = np.linalg.norm(x_iters[0,:]-x_opt)**2.0

    for j in range(1,max_iters):
        x_iters[j,:] = x_iters[j-1,:] - alpha*n*(np.dot(x_iters[j-1,:],
                                            A[stochastic_index[j],:])-b
                                            [stochastic_index[j]])*A[
                                            stochastic_index[j],:]

        err_norm_sq[j] = np.linalg.norm(x_iters[j,:]-x_opt)

    ax[0].plot(t[1:], err_norm_sq[1:, 0], label = f'SGD noise \u03C3 =
                                        {epsilon_sigma[i]}')
    ax[1].plot(t[1:], err_norm_sq[1:, 0], label = f'SGD noise \u03C3 =
                                        {epsilon_sigma[i]}')
    ax[0].plot(t[1:], theoretical_err_norm_sq[1:], label = f'Theo noise
                                         \u03C3 = {epsilon_sigma[i]}',
                                        linestyle='--')

ax[0].legend(loc='best', shadow=False, fontsize='medium')
ax[1].legend(loc='best', shadow=False, fontsize='medium')
manager = plt.get_current_fig_manager()
manager.window.showMaximized()
plt.savefig('Q7parta.pdf')


#Part (b)
sigma_data = np.zeros((n))
for j in range(n):
    sigma_data[j] = 1/np.sqrt((j+1)*1000)

A = generate_random_matrix(mu_data,sigma_data,d)
x_iters = np.zeros((max_iters,d))
x_iters[0,:] = np.reshape(np.random.normal(0,1,x_opt.shape), (-1))

stochastic_index = stochastic_index_compute(n,max_iters)

fig, ax = plt.subplots(1,2, figsize=(10,10))
plt.subplots_adjust(hspace = 0.001)
ax[0].set_xlabel('Iterations')
ax[1].set_xlabel('Iterations')
ax[0].set_ylabel('Mean Squared Error Norm')
ax[0].set_yscale('log')

for i in range(len(epsilon_sigma)): # Looping over each epsilon
    epsilon = np.random.normal(0, epsilon_sigma[i], (n,1))
    b = A @ x_opt + epsilon
    theoretical_err_norm_sq = theoretical_conv(A, b, x_opt,
                                        x_init_guess, max_iters,L)
                                        # Theoretical rate for each
                                        epsilon
    err_norm_sq = np.zeros((max_iters,1))
    err_norm_sq[0] = np.linalg.norm(x_iters[0,:]-x_opt)**2
    ### SGD for each epsilon
    for j in range(1, max_iters):
        x_iters[j,:] = x_iters[j-1,:] - alpha*n*(np.dot(x_iters[j-1,:],
                                            A[stochastic_index[j],:])-b
                                            [stochastic_index[j]])*A[
                                            stochastic_index[j],:]
        err_norm_sq[j] = np.linalg.norm(x_iters[j,:]-x_opt)

    ax[0].plot(t[1:], err_norm_sq[1:, 0], label = f'SGD noise \u03C3 =
                                        {epsilon_sigma[i]}')
    ax[1].plot(t[1:], err_norm_sq[1:, 0], label = f'SGD noise \u03C3 =
                                        {epsilon_sigma[i]}')
    ax[0].plot(t[1:], theoretical_err_norm_sq[1:], label = f'Theo noise
                                         \u03C3 = {epsilon_sigma[i]}',
                                        linestyle='--')

ax[0].legend(loc='best', shadow=False, fontsize='medium')
ax[1].legend(loc='best', shadow=False, fontsize='medium')

manager = plt.get_current_fig_manager()
manager.window.showMaximized()
```

```
plt.savefig('Q7partb.pdf')
```

Supporting function file:

```
#class    : Foundations of ML and Data Science
#homework: 3
#question: 7, function file
#author   : Mohammad Afzal Shadab
#email    : mashadab@utexas.edu

import numpy as np


def generate_random_matrix(mu,sigma,d):
    # n - Number of data points
    # d - Number of dimensions
    # mu- Mean of the Gaussian variables vector in each row vector N X
                                        1
    #sigma - Standard deviation of Gaussian variables in each row
                                        vector N X 1
    #Making n X d random matrix
    return np.array([np.random.normal(m, s, d) for m,s in zip(mu, sigma
                                        )])

def theoretical_conv(A,b,x_opt,x_initial_guess,max_iters,L):
    # A - n X d normal matrix
    # b - n X 1 RHS matrix
    # x_opt - optimal value of d X 1
    # x_initial_guess - initial guess of size d X 1
    ATA_eig, _ = np.linalg.eig(np.transpose(A) @ A)

    mu = np.min(ATA_eig) #lowest EVal

    sigma_sq = 0 #initialize
    n = np.shape(A)[0]

    for i in range(n):
        sigma_sq = sigma_sq + np.linalg.norm(A[i,:])**2.0 * ( A[i,:] @
                                            x_opt - b[i] )**2.0

    sigma_sq = sigma_sq * n

    expct_err_norm_sq = np.zeros((max_iters))
    expct_err_norm_sq[0] = np.linalg.norm(x_initial_guess-x_opt)**2.0

    for t in range(1,max_iters):
        expct_err_norm_sq[t] = (1-mu/(2*L))**t * expct_err_norm_sq[0] +
                                             sigma_sq/(mu*L)

    return expct_err_norm_sq

def stochastic_index_compute(n,max_iters):
    stochastic_index = np.random.permutation(n)
    if(max_iters%n != 0 and int(max_iters/n) != 0):
        stochastic_index = np.concatenate(np.tile(stochastic_index, int
                                            (max_iters/n)),
                                            stochastic_index[:(
                                            max_iters%n)])
    elif(max_iters%n == 0):
        stochastic_index = np.tile(stochastic_index, int(max_iters/n))
    elif(int(max_iters/n) == 0):
        stochastic_index = stochastic_index[:(max_iters%n)]
    else:
        print("Error!")
    return stochastic_index
```