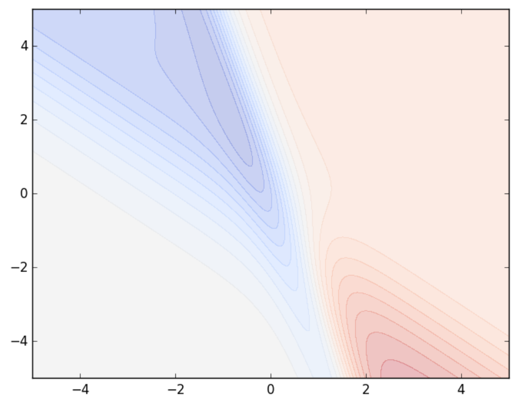
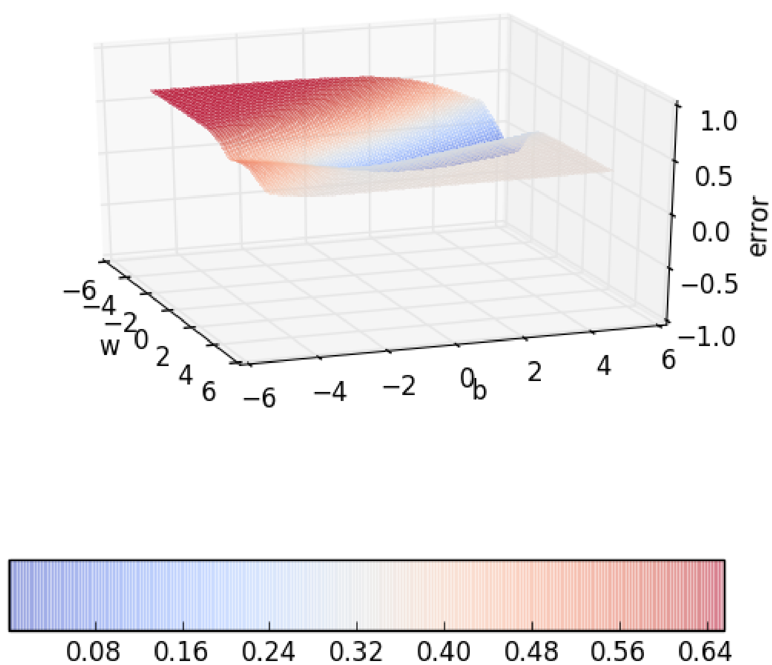
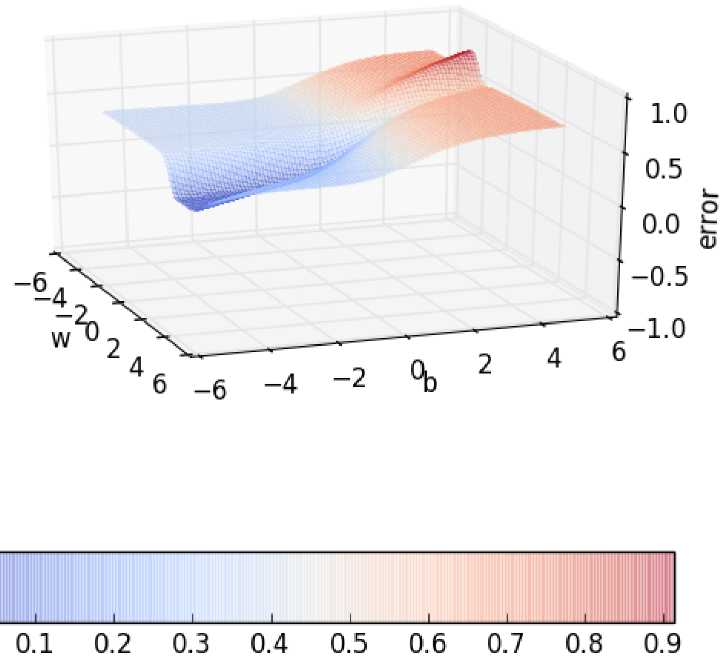
1. Consider the following contour map plotted in 2d.



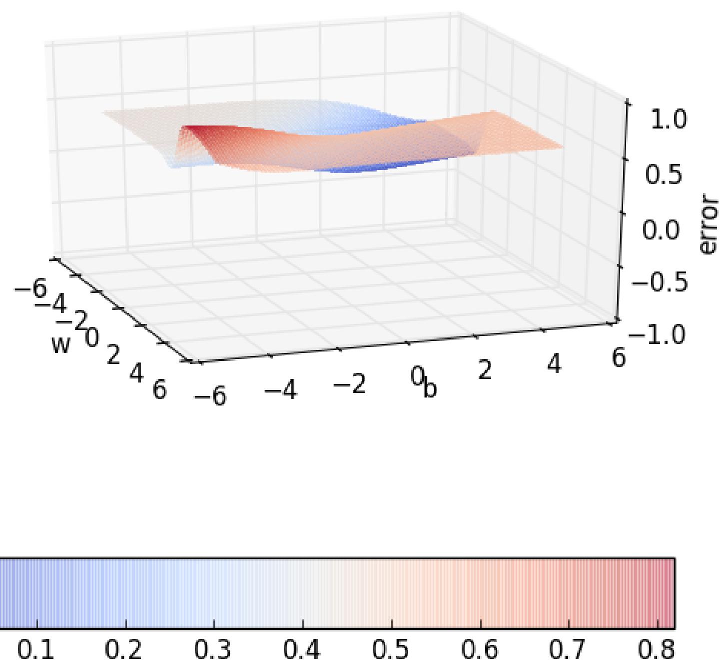
Note that in the above 2d plot the horizontal axis corresponds to the parameter w and the vertical axis corresponds to the parameter b. Which of the 3d plots below corresponds to the 2d plot shown in the figure above (please see carefully which axis corresponds to w and which to b in the 3d plot).



A.



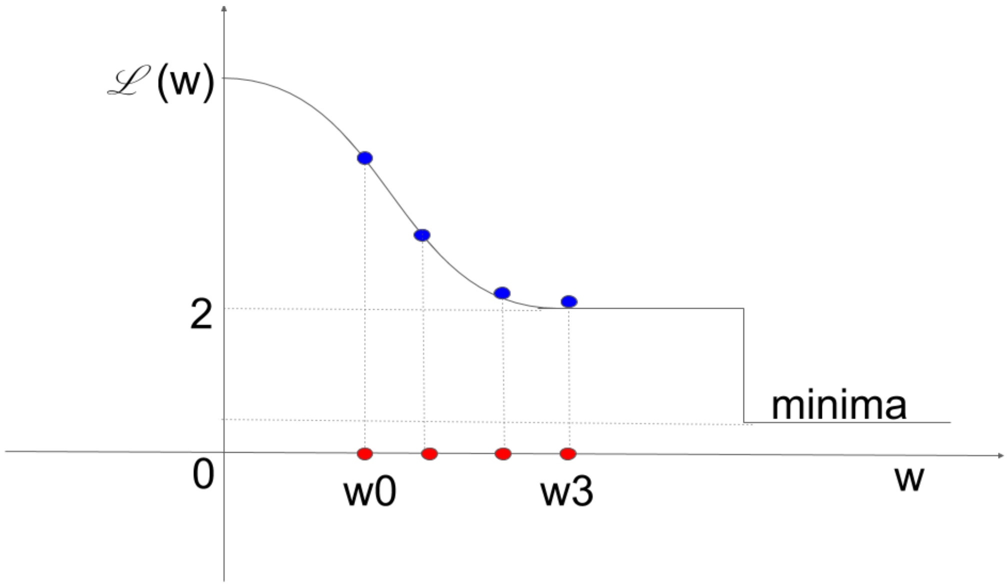
B.



C.

Solution: Option C is the correct answer.

2. Consider the loss function $\mathscr{L}(w)$ as shown in the figure below. You are interested in finding the minima of this function *i.e.*, the value(s) of w for which the function will take its lowest value. To do so you run gradient descent starting with a random value w_0 (the leftmost red dot in the figure). After running, three steps of gradient descent you have the updated value of w as w_3. The red dots in the figure show the value of w at each step and the blue dots show the corresponding value of the loss function $\mathscr{L}(w)$. Now, what will happen if you run the 4th step of gradient descent, *i.e.*, if you try to update the value of w using the gradient descent update rule. Assume that the learning rate is 1.



A. the value of w will increase ($i.e.$, $w_4 > w_3$)

B. the value of w will remain the same ($i.e.$, $w_4 = w_3$)

C. the value of w will decrease ($i.e.$, $w_4 < w_3$)

Solution: Option B is the correct answer.
Note that the update rule for gradient descent is given by:

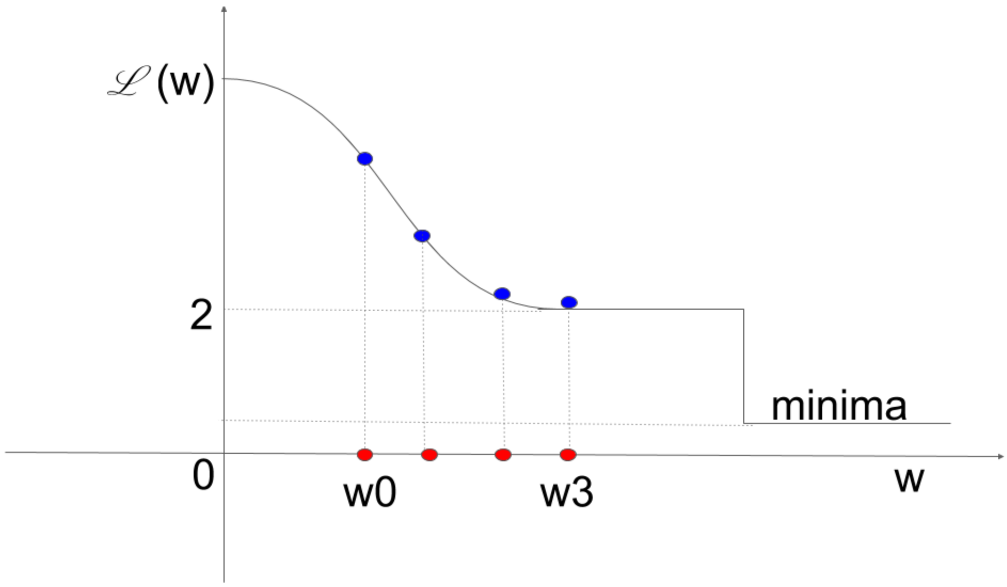
$$w_{t+1} = w_t - \eta \nabla w_t$$

After running 3 steps of gradient descent we have reached a region where the function is absolutely flat, *i.e.*, there is no change in the value of the function in the small neighborhood around w_3. Hence, the derivative ∇w_t at this point will be zero and the value of w will not get updated. Hence, **Option B** is the correct answer.

3. Continuing the previous question and referring to the same figure again, suppose instead of gradient descent you ran 3 iterations of momentum based gradient descent resulting in the value w_3 as shown in the figure. Note that the update rule of momentum based gradient descent is:

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

Assume that the learning rate is 1 and the momentum parameter $\gamma > 0$. Now, what will happen if you run the 4th step of gradient descent, *i.e.*, if you try to update the value of w using the update rule of momentum based gradient descent.



A. the value of w will increase ($i.e.$, $w_4 > w_3$)

B. the value of w will remain the same ($i.e.$, $w_4 = w_3$)

C. the value of w will decrease ($i.e.$, $w_4 < w_3$)

Solution: Option A is the correct answer.
Once again, after running 3 steps of momentum based gradient descent we have reached a region where the function is absolutely flat, *i.e.*, there is no change in the value of the function in the small neighborhood around w_3. Hence, the derivative ∇w_t at this point will be zero. However, the momentum will still be non-zero and hence w will continue to move in the same direction as it did during the last 3 updates, *i.e.*, the value of w will increase. Hence, **Option A** is the correct answer.

4. Suppose we choose a model $f(x) = \sigma(wx + b)$ which has two parameters w, b. Further, assume that we are trying to learn the parameters of this model using 200 training points. If we use mini-batch gradient descent with a batch size of 10 then how many times will each parameter get updated in one epoch.

   A. 10

   B. 20

   C. 100

   D. 200

Solution: **Option B** is the correct answer. The parameters will get updated once for every mini-batch and the data is divided into $\frac{200}{10} = 20$ mini-batches. Hence, the parameters will get updated 20 times in one epoch.

5. Note that the update rule for momentum based gradient descent is given by

$$\begin{aligned} update_t &= \gamma \cdot update_{t-1} + \eta \nabla w_t \\ w_{t+1} &= w_t - update_t \end{aligned}$$

Let $\eta = 1$ and $\gamma = 0.9$ and ∇w_1 be the derivative computed at the first time step. If you run momentum based gradient descent for 10 iterations then what fraction of ∇w_1 will be a part of $update_{10}$

A. $0.9 \nabla w_1$

B. $\frac{1}{0.9} \nabla w_1$

C. $\frac{0.9}{10-1} \nabla w_1$

D. $(0.9)^{(10-1)} \nabla w_1$

Solution: If we expand the formula for $update_t$ we get

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t = \gamma^{t-1} \cdot \eta \nabla w_1 + \gamma^{t-2} \cdot \eta \nabla w_2 + ... + \eta \nabla w_t$$

Hence, the fraction of ∇w_1 that will be a part of $update_t$ is $\gamma^{t-1} \cdot \eta \nabla w_1$. Substituting, $\gamma = 0.9, \eta = 1$ and $t = 10$ we get $0.9^{(10-1)} \nabla w_1$. Hence, **Option D** is the correct answer.

6. We saw the following update rule for Adam :

$$
\begin{aligned}
m_t &= \beta_1 * m_{t-1} + (1-\beta_1) * \nabla w_t \\
v_t &= \beta_2 * v_{t-1} + (1-\beta_2) * (\nabla w_t)^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\
w_{t+1} &= w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t
\end{aligned}
$$

$\hat{m}_t, \hat{v}_t$ are the bias corrected values of m_t, v_t. Suppose, instead of using the above equation for m_t we use the following equation where $0 \leq \alpha_1 \leq 1$ and $0 \leq \beta_1 \leq 1$

$$
m_t = \frac{\alpha_1}{\beta_1} * m_{t-1} + \frac{(\beta_1 - \alpha_1)}{\beta_1} * \nabla w_t
$$

then what would the bias corrected value of m_t be ?

A. $\hat{m}_t = \frac{m_t}{\alpha_1^t - \beta_1^t}$

B. $\hat{m}_t = \frac{\alpha_1^t m_t}{1-\beta_1^t}$

C. $\hat{m}_t = \frac{\alpha_1^t m_t}{\alpha_1^t - \beta_1^t}$

D. $\hat{m}_t = \frac{\beta_1^t m_t}{\beta_1^t - \alpha_1^t}$

Solution: Option D is the correct answer

Note that when,

$$
m_t = \beta_1 * m_{t-1} + (1-\beta_1) * \nabla w_t
$$

the bias corrected value was,

$$
\hat{m}_t = \frac{m_t}{1-\beta_1^t}
$$

Now the new equation for m_t is

$$
\begin{aligned}
m_t &= \frac{\alpha_1}{\beta_1} * m_{t-1} + \frac{(\beta_1 - \alpha_1)}{\beta_1} * \nabla w_t \\
&= \frac{\alpha_1}{\beta_1} * m_{t-1} + (1 - \frac{\alpha_1}{\beta_1}) * \nabla w_t
\end{aligned}
$$

in other words β_1 has been replaced by $\frac{\alpha_1}{\beta_1}$. Hence the bias corrected value can also be obtained by replacing β_1 by $\frac{\alpha_1}{\beta_1}$, *i.e.*,

$$\begin{aligned}\hat{m}_t &= \frac{m_t}{1-(\frac{\alpha_1}{\beta_1})^t} \\ &= \frac{\beta_1^t m_t}{\beta_1^t - \alpha_1^t}\end{aligned}$$

7. In this question you will implement the Adam algorithm on toy 2-D dataset which consists of 40 data points, *i.e.*, 40 (x,y) pairs. You can download the dataset using the URL : https://drive.google.com/file/d/1w6aXg7K7nIv4DBUyWAM-abXvKh2frYRM/view?usp=sharing
For this question you have to use the squared error loss function which is given as,

$$loss = \frac{1}{2}(\hat{y} - y)^2$$

where $\hat{y}$ is the output of your model given by:

$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

Now given the following hyperparameter settings,

- learning rate = 0.01
- initial weight, w = 1
- initial bias, b = 1
- number of iterations = 100
- $\beta_1 = 0.9$
- $\beta_2 = 0.99$

What is the value of the loss at the end of 100 iterations?

A. loss = 0.058

B. loss = 0.0

C. loss = 1.58

D. loss = 0.58

Solution: Option A is correct. You can find the value of the loss at the end of 100 iterations by using the following code(the one which is given in the lecture video).

```
# Importing libraries
import pandas as pd
import numpy as np
import math

def f(w,b,x):
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error(w, b): #Calculate loss/error
    err = 0.0
```

```python
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def run_adam(X,Y,init_w,init_b,eta, max_epochs): # ADAM algorithm
    # Inititializations
    w,b = init_w, init_b
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0,0,0,0,1e-8,0.9,0.999

    for i in range(max_epochs):
        dw,db = 0,0
        for x,y in zip(X,Y):
            dw += grad_w(w,b,x,y)
            db += grad_b(w,b,x,y)
        # Compute history
        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w +(1-beta2)*dw**2
        v_b = beta2 * v_b +(1-beta2)*db**2

        # Apply bias correction
        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        # Apply ADAM's update rule
        w = w - (eta/np.sqrt(v_w + eps)) * m_w
        b = b - (eta/np.sqrt(v_b + eps)) * m_b

    return w,b

if __name__ == "__main__":
    filename = 'A4_Q7_data.csv'
    df = pd.read_csv(filename) #Loading data
    X = df['X']
    Y = df['Y']
    initial_w = 1
    initial_b = 1
    eta = 0.01
```

```
    max_epochs = 100
    w,b = run_adam(X, Y, initial_w , initial_b , eta , max_epochs)
    error = error(w,b)
    print("error = {}".format(error))
```

The code below is as per the corrected ADAM update equations.(We have updated the lecture slides, please refer to them.)

Solution:

```
import pandas as pd
import numpy as np
#import matplotlib.pyplot as plt
import math

def f(w,b,x):
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error(w, b): #Calculate loss/error
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_adam(X,Y,init_w ,init_b ,eta , max_epochs):
    w,b = init_w , init_b
    m_w, m_b, v_w, v_b, m_w_hat, m_b_hat, v_w_hat, v_b_hat, eps, beta1, beta2 = 0,0,0,0,0,0,0,0,1e-8,0.9,0.99
    for i in range(max_epochs):
        dw,db = 0,0
        for x,y in zip(X,Y):
            dw += grad_w(w,b,x,y)
            db += grad_b(w,b,x,y)
        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

        v_w = beta2 * v_w +(1-beta2)*dw**2
        v_b = beta2 * v_b +(1-beta2)*db**2

```

```python
        m_w_hat = m_w/(1-math.pow(beta1,i+1))
        m_b_hat = m_b/(1-math.pow(beta1,i+1))

        v_w_hat = v_w/(1-math.pow(beta2,i+1))
        v_b_hat = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta/np.sqrt(v_w_hat + eps)) * m_w_hat
        b = b - (eta/np.sqrt(v_b_hat + eps)) * m_b_hat
    return w,b

if __name__ == "__main__":
    filename = 'A4_Q8_data.csv'
    df = pd.read_csv(filename) #Loading data
    X = df['X']
    Y = df['Y']
    initial_w = 1
    initial_b = 1
    eta = 0.01
    max_epochs = 100
    w,b = do_adam(X, Y, initial_w, initial_b, eta, max_epochs)
    error = error(w,b)
    print("error = {}".format(error))
```

On executing this code, the error value which you should get is 0.0036