Question 1.

1) At the green point $p(C_1|x) = p(C_2|x) = p(C_3|x)$ $= \frac{1}{3}$

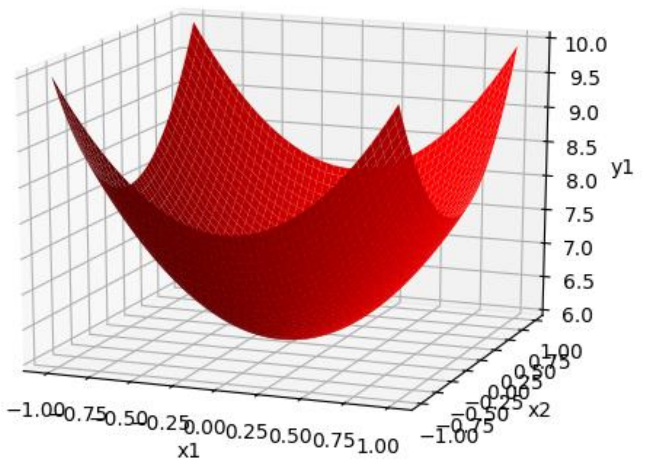
2) On the red line the two activiation functions have the same value.

When we move along the red line the two equaled activation functions are getting larger and larger, and are close to $\frac{1}{2}$, another activation function is getting smaller and close to 0.

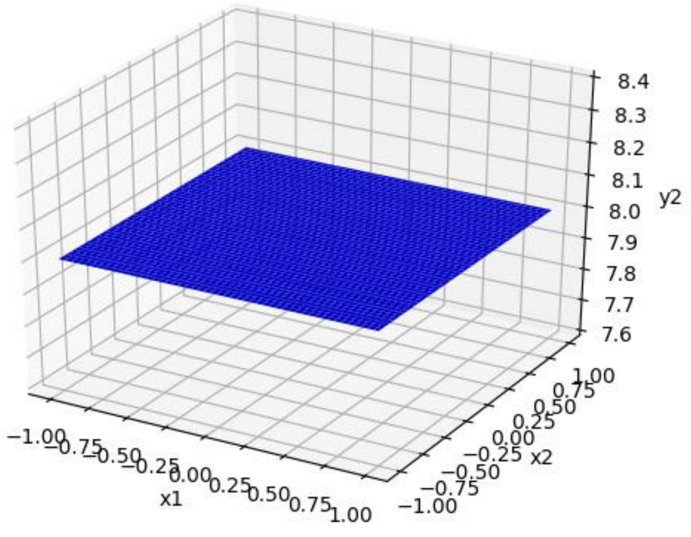
3) As we move away from the intersection point to one regin, the activation function of that regin is getting bigger and bigger until "1", at the same time the other two activation functions are very close to "0".

Question 2

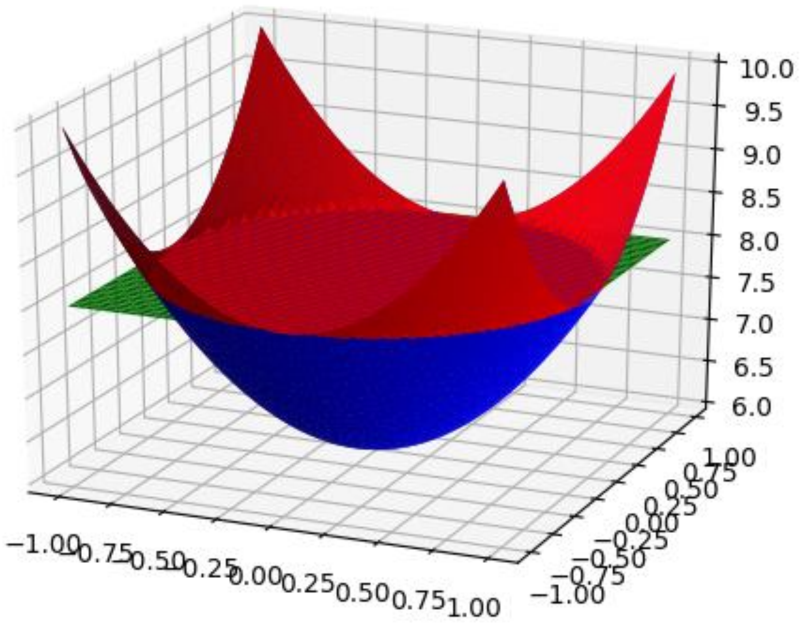
1)



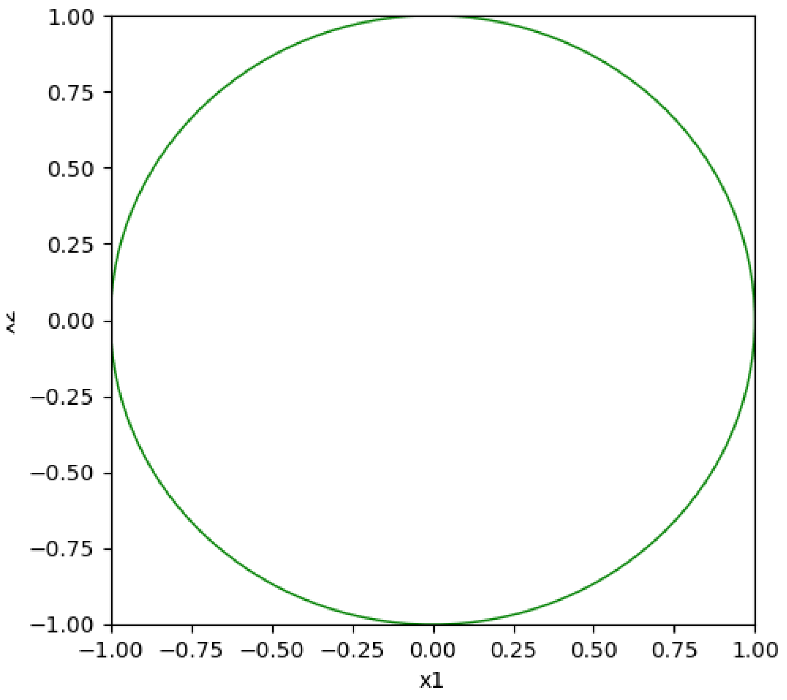
2)



3)
The decision region in 3D:



The decision boundary $x_1^2 + x_2^2 = 1$ in 2D is a circle with radius=1:



Question 3.

1) For final output node $h(a) = a$.

So $y_1^{(4)} = a_1^{(4)}$

$$\delta_1^{(4)} = \frac{\partial E_n(w)}{\partial a_1^{(4)}} = \frac{\partial}{\partial a_1^{(4)}} \frac{1}{2}(a_1^{(4)} - t_n)^2 = a_1^{(4)} - t_n.$$

[ error function: $E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2$ ]

$$\frac{\partial E_n(w)}{\partial w_{12}^{(3)}} = \frac{\partial E_n(w)}{\partial a_1^{(4)}} \frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}} \longrightarrow$$

$$a_1^{(4)} = \sum_i w_{1i}^{(3)} z_i^{(3)} = w_{11}^{(3)} z_1^{(3)} + w_{12}^{(3)} z_2^{(3)} + w_{13}^{(3)} z_3^{(3)}$$

$$= \delta_1^{(4)} z_2^{(3)}$$

$$= z_2^{(3)}(a_1^{(4)} - t_n).$$

2) $$\frac{\partial E_n(w)}{\partial a_1^{(3)}} = \frac{\partial E_n(w)}{\partial a_1^{(4)}} \frac{\partial a_1^{(4)}}{\partial a_1^{(3)}}$$

$\nearrow$ $\delta_1^{(3)}$

$$= \delta_1^{(4)} \frac{\partial}{\partial a_1^{(3)}} (w_{11}^{(3)} z_1^{(3)} + w_{12}^{(3)} z_2^{(3)} + w_{13}^{(3)} z_2^{(3)})$$

$$= \delta_1^{(4)} w_{11}^{(3)} h'(a_1^{(3)})$$

$$\frac{\partial E_n(w)}{\partial w_{11}^{(2)}} = \frac{\partial E_n(w)}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial w_{11}^{(2)}}.$$

$$= \delta_1^{(3)} \frac{\partial}{\partial w_{11}^{(2)}} (z_1^{(2)} w_{11}^{(2)} + z_2^{(2)} w_{12}^{(2)} + z_3^{(2)} w_{13}^{(2)})$$

$$= \delta_1^{(3)} z_1^{(2)}$$

$$= z_1^{(2)} h'(a_1^{(3)}) \delta_1^{(4)} w_{11}^{(3)}.$$

3) $\frac{\partial E_n(w)}{\partial a_1^{(2)}} = \delta_1^{(2)} = \sum_{k=1}^{3} \frac{\partial E_n(w)}{\partial a_k^{(3)}} \frac{\partial a_k^{(3)}}{\partial a_1^{(2)}}$

$$= \sum_{k=1}^{3} \delta_k^{(3)} \frac{\partial a_k^{(3)}}{\partial a_1^{(2)}}$$

$$= h'(a_1^{(2)}) \sum_{k=1}^{3} W_{k1}^{(2)} \delta_k^{(3)}.$$

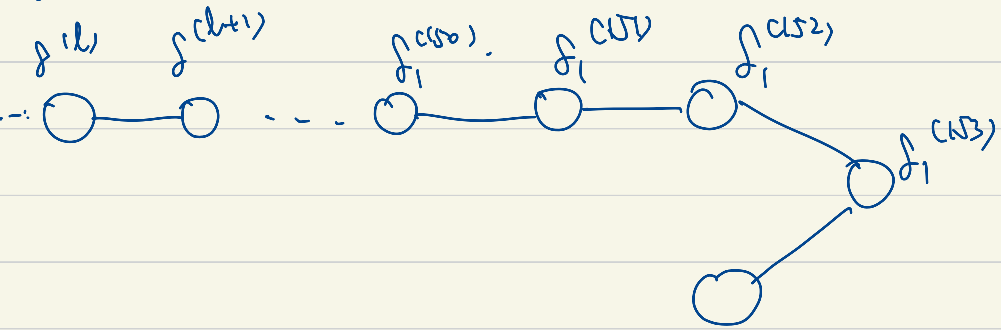
$$\frac{\partial E_n(w)}{\partial w_{11}^{(1)}} = \frac{\partial E_n(w)}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial w_{11}^{(1)}}$$

$$= \delta_1^{(2)} \frac{\partial}{\partial w_{11}^{(1)}} \left( x_1 w_{11}^{(1)} + x_2 w_{12}^{(1)} + x_3 w_{13}^{(1)} \right)$$

$$= \delta_1^{(2)} \cdot x_1$$

$$= x_1 \cdot h'(a_1^{(2)}) \sum_{k=1}^{3} W_{k1}^{(2)} \delta_k^{(3)}$$

Question 4.



$$\delta_1^{(152)} = h'(a_1^{(152)}) w_{11}^{(152)} \delta_1^{(153)}.$$

$$\delta_1^{(151)} = \frac{\partial E_n(w)}{\partial a_1^{(151)}} = \frac{\partial E_n(w)}{\partial a_1^{(152)}} \frac{\partial a_1^{(152)}}{\partial a_1^{(151)}}$$

$$= h'(a_1^{(N1)})\, w_{11}^{(N1)}\ \delta_1^{(N2)}$$

$$= h'(a_1^{(N1)})\, w_{11}^{(N1)}$$

$$h'(a_1^{(N2)})\, w_{11}^{(N2)}\ \delta_1^{(N3)}$$

$$\delta_1^{(N0)} = \frac{\partial E_n(w)}{\partial a_1^{(N0)}} = \frac{\partial E_n(w)}{\partial a_1^{(N1)}} \frac{\partial a_1^{(N1)}}{\partial a_1^{(N0)}}$$

$$= h'(a_1^{(N0)})\, w_{11}^{(N0)}\ \delta_1^{(N1)}$$

$$= h'(a_1^{(N0)})\, w_{11}^{(N0)}$$

$$h'(a_1^{(N1)})\, w_{11}^{(N1)}$$

$$h'(a_1^{(N2)})\, w_{11}^{(N2)}\ \delta_1^{(N3)}$$

$$\vdots$$

$$\delta_1^{(l+1)} = h'(a_1^{(l+1)})\, w_{11}^{(l+1)}$$

$$h'(a_1^{(l+2)})\, w_{11}^{(l+2)}$$

$$\vdots$$

$$h'(a_1^{(N3)})\, w_{11}^{(N3)}\ \delta_1^{(N3)}.$$

$$= \delta_1^{(N3)} \prod_{k=l+1}^{N2} h'(a_1^{(k)})\, w_{11}^{(k)}$$

$$\frac{\partial E_n(w)}{\partial w_{11}^{(l)}} = \frac{\partial E_n(w)}{\partial a_1^{(l+1)}} \frac{\partial a_1^{(l+1)}}{\partial w_1^{(l)}}$$

$$= \delta_1^{(l+1)} z_1^{(l)}$$

$$= z_1^{(l)} \delta_1^{(153)} \prod_{k=l+1}^{152} h'(a_1^{(k)}) w_{11}^{(k)}.$$

2). $h(a) = \frac{1}{1+\exp(-a)} = (1+e^{-a})$

$$h'(a) = -(1+e^{-a})^{-2} \cdot (-1) \cdot (e^{-a}) = \frac{e^{-a}}{(1+e^{-a})^2}.$$

$$= \left(\frac{1}{1+e^{-a}}\right)\left(1-\frac{1}{1+e^{-a}}\right)$$

$$= h(a)(1-h(a)) \qquad \boxed{eq. 1}$$

If the sigmoid activation $h(a)$ are very close to the boundary of their operating range i.e. 0 and 1 in the network for small $l$. Then according to eq. 1 the gradient of activation $h'(a)$s are very small so that the gradients of weights $\frac{\partial E_n(w)}{\partial w}$ could be very small since it's the product of $h'(a)$

Therefore the learning process in early stage are very slow.

Other reasons which the gradient of weights could be small.
According to:

$$\frac{\partial E_n}{\partial w_{11}^{(l)}} = z_1^{(l)} \delta_1^{(153)} \prod_{k=l+1}^{152} h'(a_1^{(k)}) w_{11}^{(k)}.$$

1). If $z_1^{(l)}$ is very small.

2) If many of $w_{11}^{(l+1)}, w_{11}^{(l+2)}$ $w_{11}^{(l+3)}$ ... are very small.

It should be noticed that the gradient of sigmoid activation function has operation range $[0, 0.25]$. which means for deep neuronetwork the product of $h'(a_1^{k}) h'(a_1^{k+1})$... will be small which will cause vanishing gradient.

3). $h(a) = \max(0, a)$.

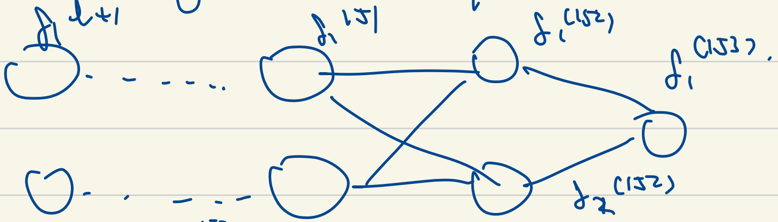
$$h'(a) = \begin{cases} 0 & \text{if } a^{(k)} < 0 \\ 1 & \text{o.w.} \end{cases}$$

$$\frac{\partial E_n(w)}{\partial w_{11}^{(l)}} = z_1^{(l)} \delta_1^{(153)} \prod_{k=l+1}^{152} h'(a_1^{(k)}) w_{11}^{(k)}.$$

So, when we have $a^{(k)} < 0$, i.e $h'(a) = 0$ we will have "0" gradient.

Another reason to cause gradient equal to zero is the weights $w$ equal to zero.

4).



$$\delta_1^{(152)} = \frac{\partial E_n(w)}{\partial a_1^{(153)}} \frac{\partial a_1^{(153)}}{\partial a_1^{(152)}}$$

$$= h'(a_1^{(152)}) w_{11}^{(152)} \cdot \delta_1^{(153)}$$

$$\delta_2^{(152)} = \frac{\partial E_n(w)}{\partial a_1^{(153)}} \frac{\partial a_1^{(153)}}{\partial a_2^{(152)}}$$

$$= h'(a_2^{(152)}) w_{12}^{(152)} \delta_1^{(153)}.$$

$$\delta_1^{(l+1)} = \frac{\partial E_n(w)}{\partial a_1^{(l+2)}} \frac{\partial a_1^{(l+2)}}{\partial a_1^{(l+1)}} + \frac{\partial E_n(w)}{\partial a_2^{(l+2)}} \frac{\partial a_2^{(l+2)}}{\partial a_1^{(l+1)}}$$

since it's bipartite.

$$= \delta_1^{(l+2)} w_{11}^{(l+1)} h'(a_1^{(l+1)}) + \delta_2^{(l+2)} w_{21}^{(l+1)} h'(a_1^{(l+1)})$$

$$a_2^{(l+2)} = w_{21}^{(l+1)} z_1^{(l+1)} + w_{22}^{(l+1)} z_2^{(l+1)}.$$

$$a_1^{(l+2)} = w_{11}^{(l+1)} z_1^{(l+1)} + w_{12}^{(l+1)} z_2^{(l+2)}$$

$$= h'(a_1^{(l+2)}) w_{11}^{(l+2)} \delta_1^{(l+3)} w_{11}^{(l+1)} h'(a_1^{(l+1)}) + .$$

$$h'(a_2^{(l+2)}) w_{12}^{(l+2)} \delta_1^{(l+3)} w_{21}^{(l+1)} h'(a_1^{(l+1)}).$$

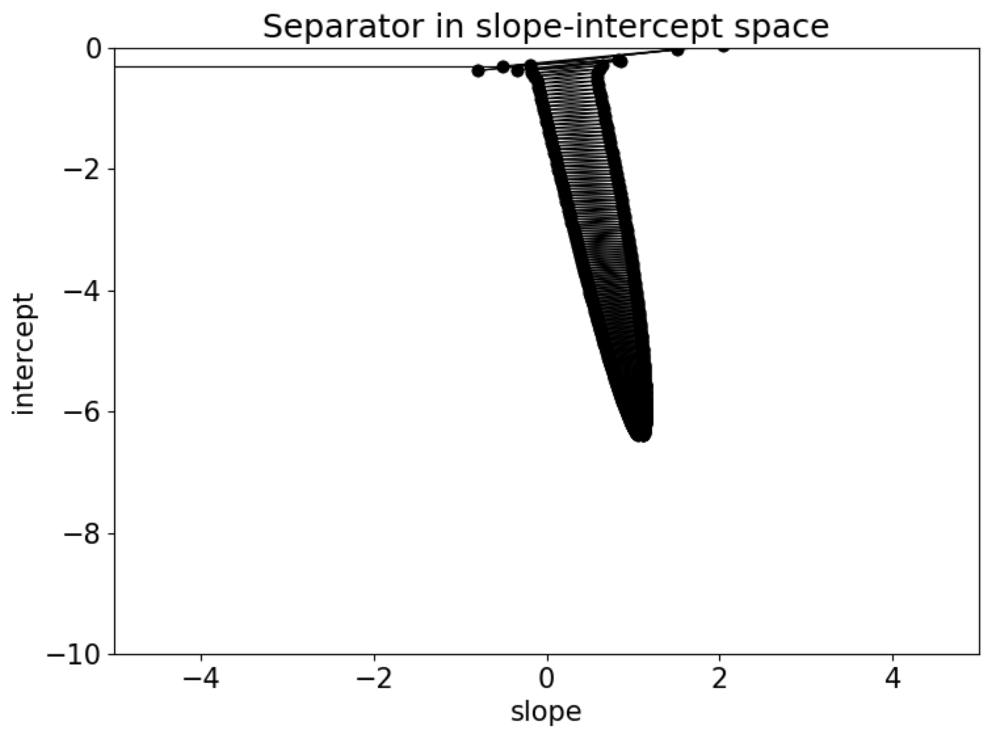
So the bipartite network makes the $\delta^{(l+1)}$ to the form of " sum of chains. "

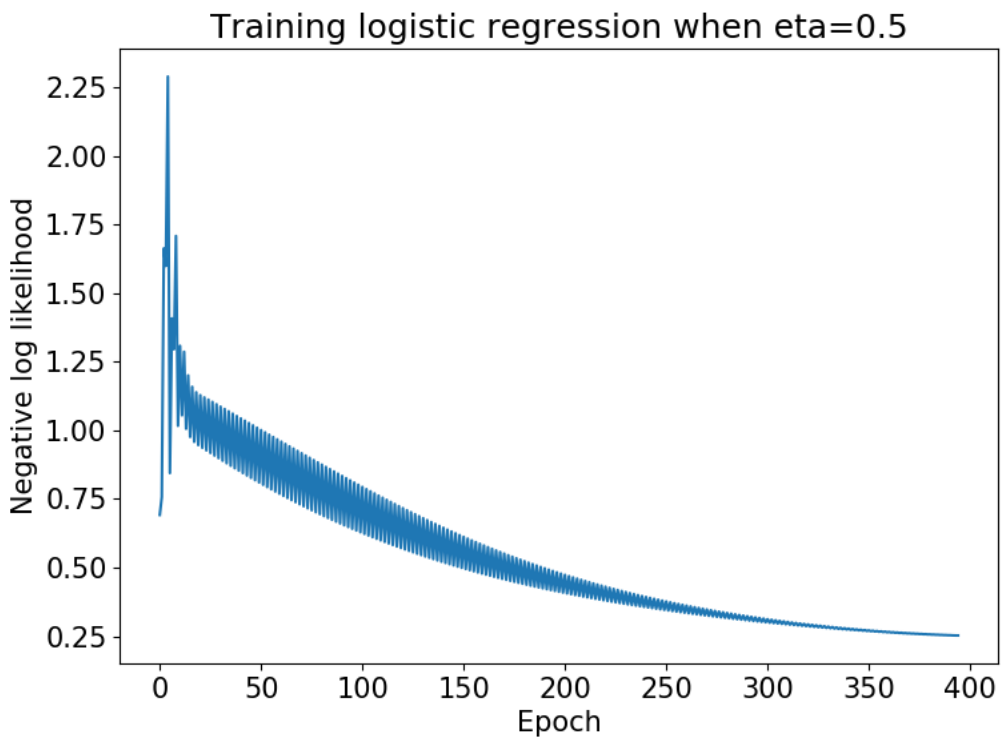
In this case to make the gradients of weights equal to zero. we need all the edges coming out of the $(l+1)$ node has $h'() = 0$ i.e negative activation or weights $= 0$.

the node $w^{(l)}$ connect to .

Or say the sum of two chains is zero
since there may be a special case first part of
sum is negative and second part is positive.
which yields a "zero" sum.

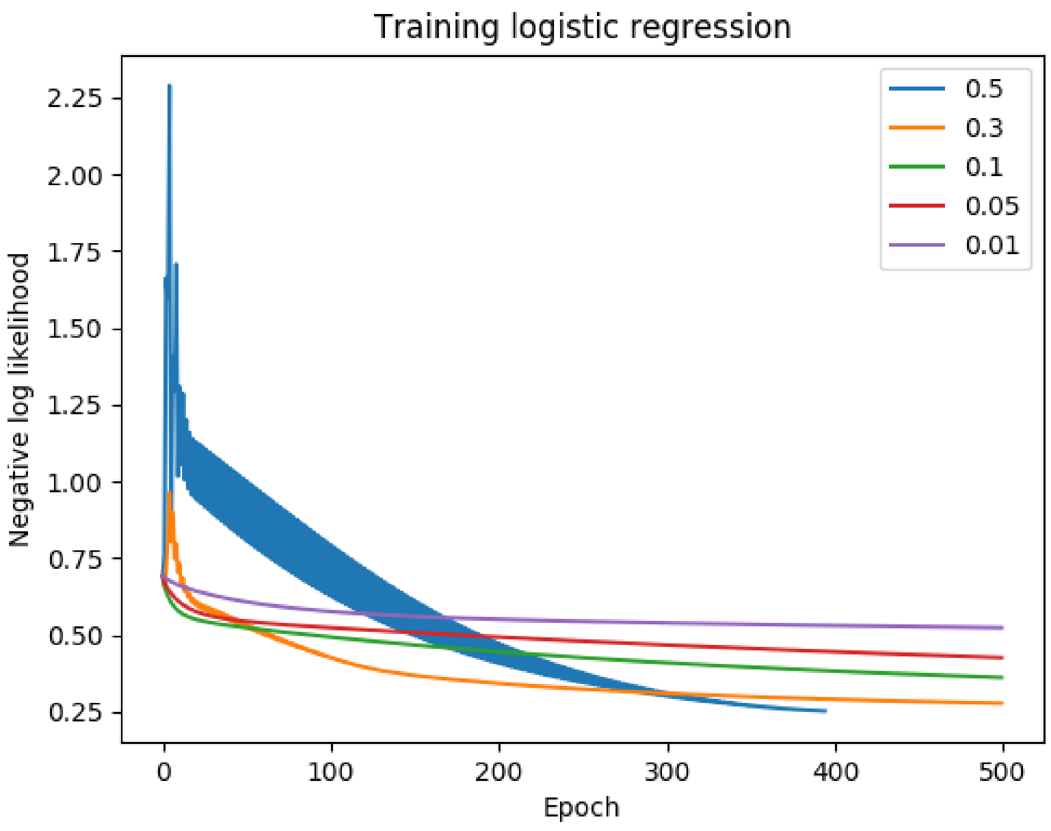
Question 5
1)





The oscillating of the error curve could be caused by the relatively large learning rate eta=0.5. The error decrease in the direction of the gradient but if the learning rate is too large we could step over the local minimum and jump to some instances with even higher error.

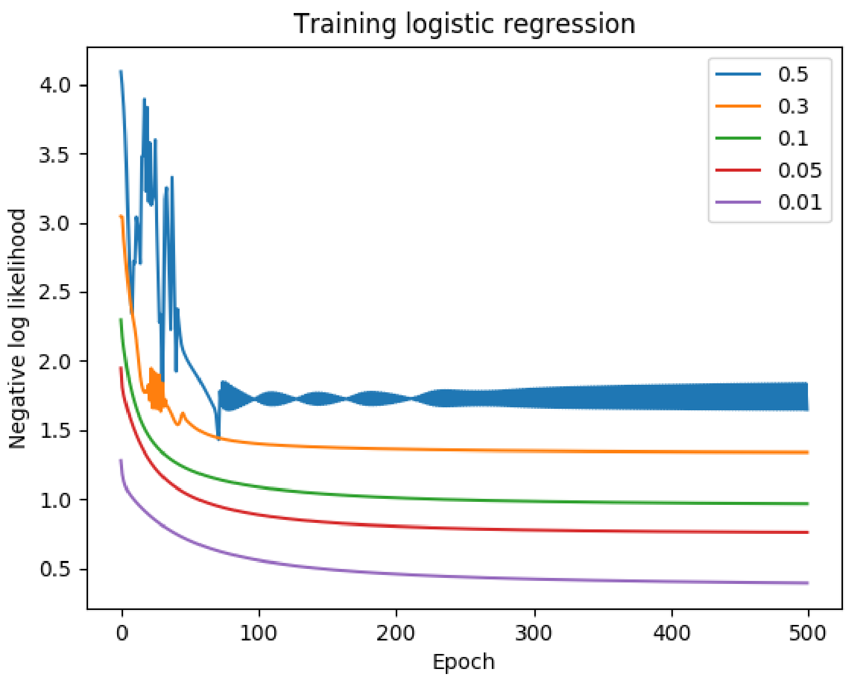
2)



The relatively small learning rate 0.3, 0.1, 0.05, 0.01 gives less oscillating curve since the step we move each time in gradient decent now is smaller.
The relatively large 0.5 learning rate gave us the lowest error overall and it achieves the tolerance level most quickly, in other words before 500 epochs.
The 0.1 learning rate gave us the low error most quickly as you can see on the graph that green line is the steepest before 100 epochs.
So I think there's no fixed answer for the learning rate choice in this case. It all depends on the experiment setting.

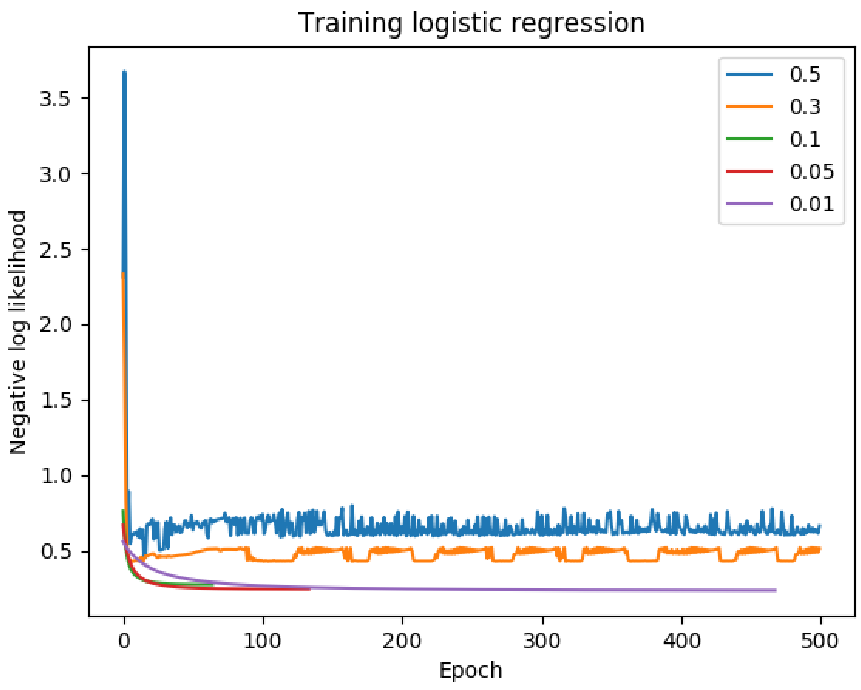
3)
I have talked with TA about this question since I noticed that shuffle made a big difference for the result at different place (shuffle for each epoch/eta or shuffle at the beginning). The TA explained to me that usually shuffle doesn't affect the model a lot and the most reasonable place to shuffle is for each epoch since it increases the possibility to find the right w the most. But for our case, the dataset given this time is special so that without shuffle the SGD doesn't make the optimization faster and also with shuffle at different place we will see different effects. The following graphs will illustrate this clearer.

**Without Shuffle:**



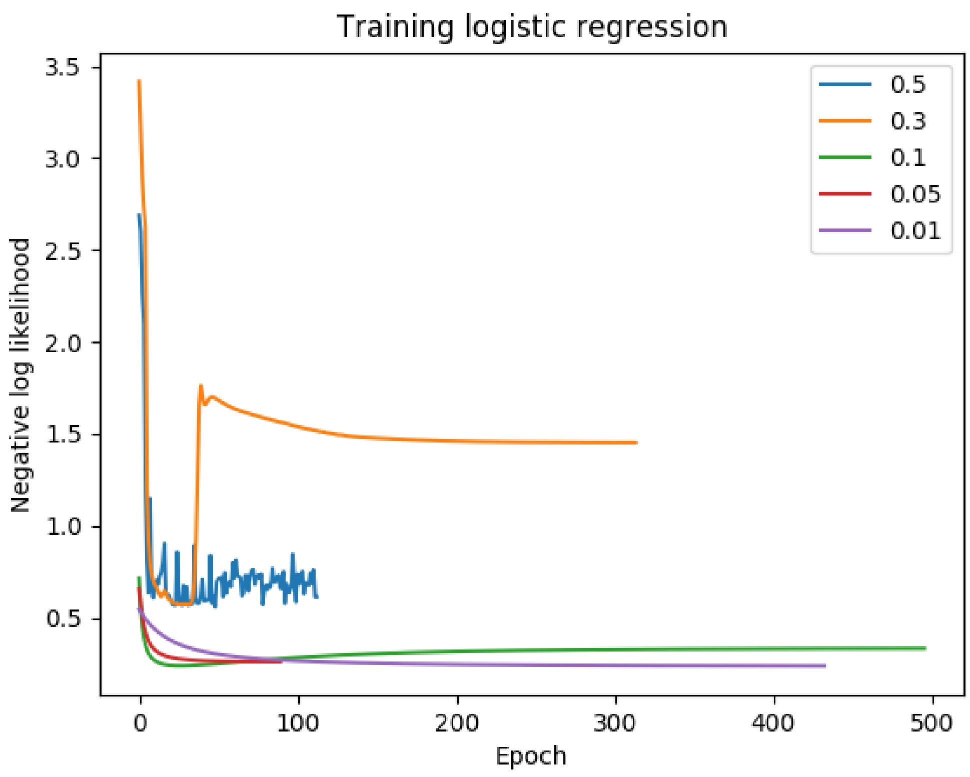
The SGD isn’t faster than gradient decent since all of the learning rate didn’t achieve the tolerance level before 500 epochs.

**One instance (Each time the graph is different due to shuffle) with shuffle at the beginning (before the iteration):**



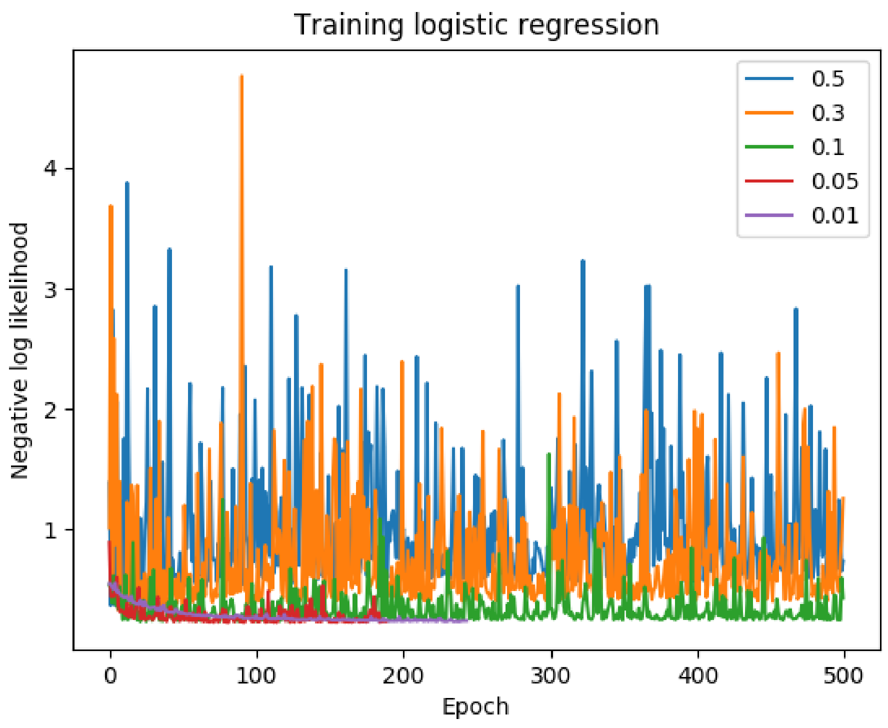
Large learning rate resulted in oscillation and kind-of increasing error curve.
With shuffle at the beginning, the SGD is faster than gradient decent since it generates a low error within 500 epochs.

**One instance with shuffle for each eta:**



Large learning rate resulted in oscillation and increasing error curve.
With shuffle for each eta, the SGD is faster than gradient decent since it generates a low error within 500 epochs.

**One instance with shuffle for each epoch:**



The SDG is also faster but the oscillation is huge now.

Question 6(Option3):
To run this code on GPU I used the cuda library:

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

and send the corresponding tasks to the gpu memory so that it could be run parallelly in gpu.

```
model = cifar_resnet20().to(device)
inputs, labels = data[0].to(device),data[1].to(device)
```

To apply L2 regularization to the coefficients we can add a weight_decay argument in optimizer so the code is:

```
optimizer = optim.SGD(list(model.fc.parameters()), lr=0.001, momentum=0.9, weight_decay=0.2)
```

The weight_decay arugment here decided the value of regularizer.

And in the code, I also added a tester for testing data and print a corresponding accuracy on 10000 test images.
Without Regularization:
Accuracy of the network on the 10000 test images: 67%
With Regularization = 0.1:
Accuracy of the network on the 10000 test images: 65%
With Regularization = 0.2:
Accuracy of the network on the 10000 test images: 64%

We can see that for our model as we increase the regularizer($\lambda$), the model accuracy is decreasing. The regularization is actually decreasing the model's prediction power. In assignment 1 question 3.3 we have discussed a similar scenario. In most cases the testing error for regularized regression is lower than unregularized regression if we have a high degree/deep network which is highly likely to cause overfitting.
But the lower error is not guaranteed, if we have a weak/underfitting/luckily proper model then the regularization will slash the predictive power even more and make the testing error larger.

**Question2**

```python
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d


def y1(x1, x2):
    return 6 + 2 * x1 ** 2 + 2 * x2 ** 2


def y2(x1, x2):
    return 8 + 0 * x1 ** 2 + 0 * x2 ** 2

# Draw y1(x)
fig = plt.figure(1)
ax = plt.axes(projection='3d')
x1 = x2 = np.arange(-1.0, 1.0, 0.01)
(x1,x2)=np.meshgrid(x1,x2)
y1 = y1(x1,x2)
ax.plot_surface(x1, x2, y1, color='red')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y1')
plt.show()


# Draw y2(x)
fig = plt.figure(2)
ax = plt.axes(projection='3d')
x1 = x2 = np.arange(-1.0, 1.0, 0.01)
(x1,x2)=np.meshgrid(x1,x2)
y2 = y2(x1,x2)
ax.plot_surface(x1, x2, y2, color='blue')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y2')
plt.show()

# Draw decision boundary
fig=plt.figure(3)
plt.axis([-1,1,-1,1])
ax=fig.add_subplot(1,1,1)
circ=plt.Circle((0,0), radius=1, color='g', fill=False)
ax.add_patch(circ)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
plt.show()

# Draw decision boundary
fig = plt.figure(4)
ax = plt.axes(projection='3d')
x1 = x2 = np.arange(-1.0, 1.0, 0.01)
(x1,x2)=np.meshgrid(x1,x2)
y2 = y2(x1,x2)
y1 = y1(x1,x2)
y3 = np.maximum(y1,y2)
ax.plot_surface(x1, x2, y1, color='blue')
ax.plot_surface(x1, x2, y2, color='green')
ax.plot_surface(x1, x2, y3, color='red')
plt.show()
```

**Logistic_regression_mod.py**

```python
#!/usr/bin/env python

import numpy as np
import scipy.special as sps
import matplotlib.pyplot as plt
import assignment2 as a2

# Maximum number of iterations.  Continue until this limit, or when error change is below tol.
max_iter = 500
tol = 0.00001

# Step size for gradient descent.
etas = [0.5, 0.3, 0.1, 0.05, 0.01]

# Load data.
data = np.genfromtxt('data.txt')

# Data matrix, with column of ones at end.
X = data[:, 0:3]

# Target values, 0 for class 1, 1 for class 2.
t = data[:, 3]

# For plotting data
class1 = np.where(t == 0)
X1 = X[class1]
class2 = np.where(t == 1)
X2 = X[class2]


# Initialize legend
legend = []

for eta in etas:
    # We need to put initialization of w in the for loop o.w the w is getting smaller and smaller
    w = np.array([0.1, 0, 0])
    e_all = []
    legend.append(str(eta))
    for iter in range(0, max_iter):

        # Compute output using current w on all data X.
        y = sps.expit(np.dot(X, w))

        # e is the error, negative log-likelihood (Eqn 4.90)
        e = -np.mean(np.multiply(t, np.log(y)) + np.multiply((1 - t), np.log(1 - y)))

        # Add this error to the end of error vector.
        e_all.append(e)

        # Gradient of the error, using Eqn 4.91
        grad_e = np.mean(np.multiply((y - t), X.T), axis=1)

        # Update w, *subtracting* a step in the error derivative since we're minimizing
        w_old = w
        w = w - eta * grad_e

        # Print some information.
        print('epoch {0:d}, negative log-likelihood {1:.4f}, w={2}'.format(iter, e, w.T))

        # Stop iterating if error doesn't change more than tol.
        if iter > 0:
            if np.absolute(e - e_all[iter - 1]) < tol:
                break

    plt.plot(e_all)
```

```
# Plot error over iterations
plt.ylabel('Negative log likelihood')
plt.title('Training logistic regression')
plt.xlabel('Epoch')
plt.legend(legend)
plt.show()
```

**Logistic_regression_sgd.py**

```python
#!/usr/bin/env python

import numpy as np
import scipy.special as sps
import matplotlib.pyplot as plt
import assignment2 as a2

tol = 0.00001
max_iter = 500
etas = [0.5, 0.3, 0.1, 0.05, 0.01]
data = np.genfromtxt('data.txt')
legend=[]
np.random.shuffle(data)
X = data[:, 0:3]
t = data[:, 3]

for eta in etas:
    w = np.array([0.1, 0, 0])
    e_all = []
    legend.append(str(eta))
    for itr in range(0, max_iter):
        for i in range(0, len(X)):
            y = sps.expit(np.dot(X[i], w))
            grad_e = np.multiply((y - t[i]), X[i,:].T)
            w = w - eta * grad_e

        y = sps.expit(np.dot(X, w))
        e = -np.mean(np.multiply(t, np.log(y+1e-5)) + np.multiply((1 - t), np.log(1 - y+1e-5)))
        e_all.append(e)
        if itr > 0:
            if np.absolute(e_all[itr] - e_all[itr - 1]) < tol:
                break
    plt.plot(e_all)

plt.ylabel('Negative log likelihood')
plt.title('Training logistic regression')
plt.xlabel('Epoch')
plt.legend(legend)
plt.show()
```

**Question 6**

```python
class cifar_resnet20(nn.Module):
    def __init__(self):
        super(cifar_resnet20, self).__init__()
        ResNet20 = CifarResNet(BasicBlock, [3, 3, 3])
        url = 'https://github.com/chenyaofo/CIFAR-pretrained-models/releases/download/resnet/cifar100-resnet20-8412cc70.pth'
        ResNet20.load_state_dict(model_zoo.load_url(url))
        modules = list(ResNet20.children())[:-1]
        backbone = nn.Sequential(*modules)
        self.backbone = nn.Sequential(*modules)
        self.fc = nn.Linear(64, 10)

    def forward(self, x):
        out = self.backbone(x)
        out = out.view(out.shape[0], -1)
        return self.fc(out)



if __name__ == '__main__':
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    model = cifar_resnet20().to(device)
    transform = transforms.Compose([transforms.ToTensor(),
                                    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465),
                                                         std=(0.2023, 0.1994, 0.2010))])
    trainset = datasets.CIFAR10('./data', download=True, transform=transform)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=32,
                                              shuffle=True, num_workers=2)

    testset = datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=32,
                                             shuffle=False, num_workers=2)
    classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    criterion = nn.CrossEntropyLoss().to(device)
    optimizer = optim.SGD(list(model.fc.parameters()), lr=0.001, momentum=0.9, weight_decay=0.1 )
    ## Do the training
    for epoch in range(NUM_EPOCH):  # loop over the dataset multiple times
        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # get the inputs
            inputs, labels = data[0].to(device),data[1].to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if i % 20 == 19:    # print every 20 mini-batches
                print('[%d, %5d] loss: %.3f' %
                    (epoch + 1, i + 1, running_loss / 20))
                running_loss = 0.0
    print('Finished Training')

    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data[0].to(device),data[1].to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print('Accuracy of the network on the 10000 test images: %d %%' % (
        100 * correct / total))
```