Simon Fraser University
Assignment 2 - CMPT 419/726: Machine Learning, Fall 2018

Date**: October 25th, 2018**

Name: **Anurag Bejju**
Student ID: **301369375**
Professor: **Dr. Greg Mori**

---

1. Softmax for Multi-Class Classification (10 marks)

1.1. Since the green point lies at the intersection of all the three activation functions $a_1, a_2 \; and \; a_3$, the probability for the input x to be in classes $C_1, C_2 \; and \; C_3$ is equal.

That implies:

$$P(C_1|\, x) = P(C_2|\, x) = \; P(C_3|\, x) = \frac{1}{3}$$

1.2. If the input point x lies along any of the three red lines, then the probability of x belonging to the two classes the line divides would be equal. Let us assume that the point lies on the line that cuts Class 2 and Class 3, then

$$P(C_2|\, x) = \; P(C_3|\, x)$$

As the point moves along this red line, the probability of x belonging to the other class (i.e in the above case Class 1) would be decreasing. Which implies, $P(C_2|\, x) \; and \; P(C_3|\, x)$ would be equal where as $P(C_1|\, x)$ would decrease as the point moves along the red line and away from the green point.

1.3. If the input x moves far away from the green point (or intersection point) and stays in the middle of any one region, then the probability for that point belonging to this region would increase. It also means that this probability would be greater that the probabilities of it belonging to other two regions. For instance, if it stays in the Region 2 (Class 2) then:

$$P(C_2|\, x) > \; P(C_1|\, x) \; and \; P(C_3|\, x)$$

2. Error Back Propagation (40 marks)

Given: The activation functions $h(\cdot)$ for the hidden layers are logistics.
For the final output node, the activation function is an identity function $h(a) = a$

Error Function: $E_n(w) = \frac{1}{2}(y(x_n, w) - t_n)^2$

2.1. For the output layers:

2.1.1. Calculate $\frac{\partial E_n(w)}{\partial a_1^{(4)}}$:

Given: $a_1^{(4)}$ is the activation for the output node and $\frac{\partial E_n(w)}{\partial a_1} \equiv \delta_1^{(4)}$

Since, the activation function for the output node is an identity function
$i.e: h(a) = a$

So the output function would be:

$$E_n(w) = \frac{1}{2}\left(h(a_1^{(4)})\right) - t_n)^2 = \frac{1}{2}(a_1^{(4)} - t_n)^2$$

When we partially differentiate $E_n(w)$ w.r.t. $a_1^{(4)}$, we get:

$$\boldsymbol{\delta_1^{(4)} \equiv \frac{\partial E_n(w)}{\partial a_1} = a_1^{(4)} - t_n}$$

2.1.2. Calculate $\frac{\partial E_n(w)}{\partial w_{12}^{(3)}}$:

$\frac{\partial E_n(w)}{\partial w_{12}^{(3)}}$ can be written as $\frac{\partial E_n(w)}{\partial a_1^{(4)}} \frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}}$

Substituting the answer from 2.1.1 we get

$$\frac{\partial E_n(w)}{\partial w_{12}^{(3)}} = \left(a_1^{(4)} - t_n\right)\frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}}$$

Since $a_1^{(4)} = \sum_{k=1}^{3} w_{1k}^{(3)} z_k^{(3)}$, partially differentiating it w.r.t. $w_{12}^{(3)}$, we get

$$\frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}} = \frac{\partial}{\partial w_{12}^{(3)}} \sum_{k=1}^{3} w_{1k}^{(3)} z_k^{(3)} = z_2^{(3)}$$

Therefore:

$$\boldsymbol{\frac{\partial E_n(w)}{\partial w_{12}^{(3)}} = \left(a_1^{(4)} - t_n\right) z_2^{(3)} = \delta_1^{(4)} z_2^{(3)}}$$

2.2. For penultimate layer of nodes:

2.2.1. Write an expression for $\frac{\partial E_n(w)}{\partial a_1^{(3)}}$ using $\delta_1^{(4)}$ in it:

$\frac{\partial E_n(w)}{\partial a_1^{(3)}}$ can be written as $\frac{\partial E_n(w)}{\partial a_1^{(4)}} \frac{\partial a_1^{(4)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}}$ and

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

$$\frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} = \frac{\partial}{\partial a_1^{(3)}} \left(h\left(a_1^{(3)} \right) \right) = h'\left(a_1^{(3)} \right)$$

$$\frac{\partial E_n(w)}{\partial a_1^{(4)}} \equiv \delta_1^{(4)}$$

Therefore:

$$\boxed{\frac{\partial E_n(w)}{\partial a_1^{(3)}} = \frac{\partial E_n(w)}{\partial a_1^{(4)}} \frac{\partial a_1^{(4)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} = h'\left(a_1^{(3)} \right) \delta_1^{(4)} w_{11}^{(3)}}$$

2.2.2. Use 2.2.1 to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(2)}}$:

$\frac{\partial E_n(w)}{\partial w_{11}^{(2)}}$ can be written as $\frac{\partial E_n(w)}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial w_{11}^{(2)}}$

$$\frac{\partial a_1^{(3)}}{\partial w_{11}^{(2)}} = \frac{\partial}{\partial w_{11}^{(2)}} \left(w_{11}^{(2)} z_1^{(2)} + w_{12}^{(2)} z_1^{(2)} + w_{13}^{(2)} z_1^{(2)} \right) = z_1^{(2)}$$

$$\frac{\partial E_n(w)}{\partial a_1^{(3)}} = h'\left(a_1^{(3)} \right) \delta_1^{(4)} w_{11}^{(3)}$$

Therefore:

$$\boxed{\frac{\partial E_n(w)}{\partial w_{11}^{(2)}} = \frac{\partial E_n(w)}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial w_{11}^{(2)}} = h'\left(a_1^{(3)} \right) \delta_1^{(4)} w_{11}^{(3)} z_1^{(2)}}$$

2.3. Use weights connecting from the input:

2.3.1. Write an expression for $\frac{\partial E_n(w)}{\partial a_1^{(2)}}$ using $\delta_k^{(3)}$ in it:

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

We know

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

and

$$a_k^{(3)} = \sum_{k=1}^{3}\sum_{q=1}^{3} w_{kq}^{(2)} z_q^{(2)} = \sum_{k=1}^{3}\sum_{q=1}^{3} w_{kq}^{(2)} h(a_q^{(2)})$$

Calculating

$$\frac{\partial a_k^{(3)}}{\partial a_1^{(2)}} = \frac{\partial}{\partial a_1^{(2)}} \sum_{k=1}^{3}\sum_{q=1}^{3} w_{kq}^{(2)} h(a_q^{(2)}) = h'\left(a_1^{(2)}\right) \sum_{k=1}^{3} w_{k1}^{(2)}$$

Therefore:

$$\boldsymbol{\delta_1^{(2)} \equiv \frac{\partial E_n(w)}{\partial a_1^{(2)}} = \sum_{k=1}^{3} \frac{\partial E_n(w)}{\partial a_k^{(3)}} \frac{\partial a_k^{(3)}}{\partial a_1^{(2)}} = h'\left(a_1^{(2)}\right) \sum_{k=1}^{3} w_{k1}^{(2)} \delta_k^{(3)}}$$

2.3.2. Use 2.3.1. result to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(1)}}$:

$\frac{\partial E_n(w)}{\partial w_{11}^{(1)}}$ can be written as $\frac{\partial E_n(w)}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial w_{11}^{(1)}}$

$$\frac{\partial a_1^{(2)}}{\partial w_{11}^{(1)}} = \frac{\partial}{\partial w_{11}^{(1)}} \left(w_{11}^{(1)} z_1^{(1)} + w_{12}^{(1)} z_1^{(1)} + w_{13}^{(1)} z_1^{(1)}\right) = z_1^{(1)}$$

$$\boldsymbol{\frac{\partial E_n(w)}{\partial w_{11}^{(1)}} = \frac{\partial E_n(w)}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial w_{11}^{(1)}} = \delta_1^{(2)} z_1^{(1)} = z_1^{(1)} h'\left(a_1^{(2)}\right) \sum_{k=1}^{3} w_{k1}^{(2)} \delta_k^{(3)}}$$

3. Vanishing Gradients (40 marks)

3.1. Write an expression for $\frac{\partial E_n(w)}{\partial w_{11}^{(l)}}$ for all layers l in the network.

For $l = 152$:

$$\frac{\partial E_n(w)}{\partial w_{11}^{(152)}} = \frac{\partial E_n(w)}{\partial a_{11}^{(153)}} \frac{\partial a_{11}^{(153)}}{\partial w_{11}^{(l)}}$$

As we have seen before:

$\frac{\partial E_n(w)}{\partial a_{11}^{(153)}} = a_1^{(153)} - t_n$ and $\frac{\partial a_{11}^{(153)}}{\partial w_{11}^{(l)}} = z_1^{(152)}$

$$\frac{\partial E_n(w)}{\partial w_{11}^{(152)}} = (a_1^{(153)} - t_n)\, z_1^{(152)}$$

Therefore:

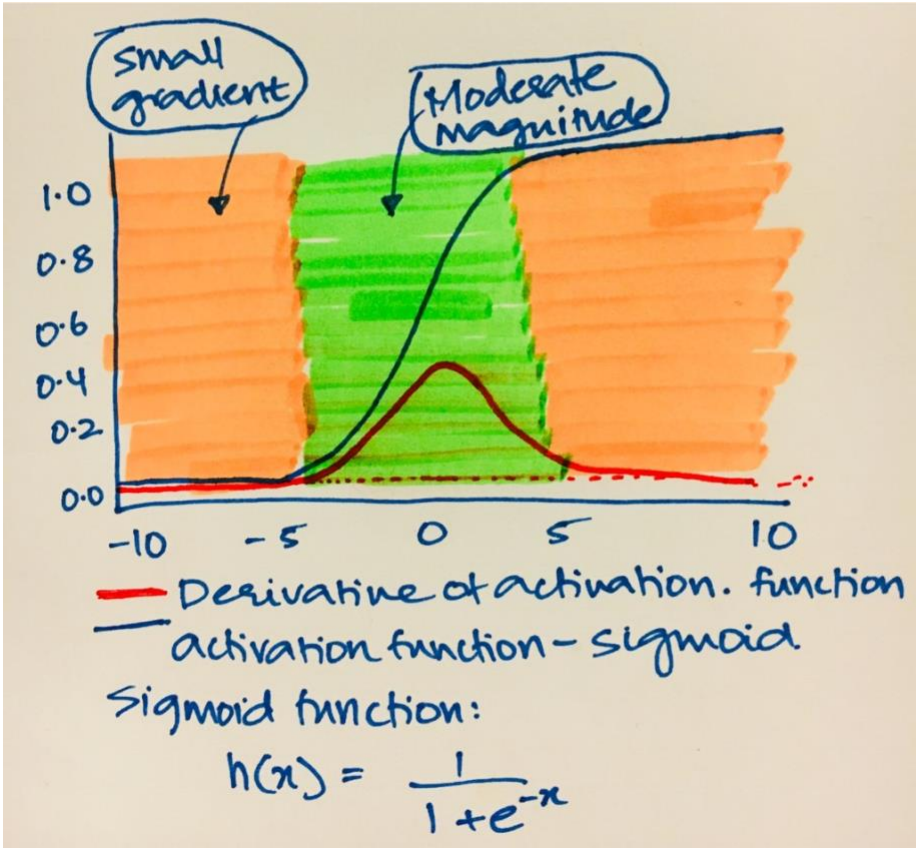
If $n \geq 153$ and $l \geq 152$:

$$\frac{\partial E_n(w)}{\partial w_{11}^{(152)}} = (a_1^{(153)} - t_n)$$

else if $n \leq 152$, $l \leq 151$ and $l = n - 1$:

$$\frac{\partial E_n(w)}{\partial w_{11}^{(l)}} = (a_1^{(153)} - t_n) \prod_{n=l+1}^{153} h'\left(a_1^{(n)}\right) w_{11}^{(n)}\, z_1^{(l)}$$

3.2. When we perform Back-propagation, gradients tend to get smaller and smaller as we keep moving backward in the network. Therefore, the neurons in the earlier layers learn very slowly as compared to the neurons in the later layers in the hierarchy resulting to an increase in training time and decrease the prediction accuracy of the model. A logistic activation function such as sigmoid, 'squash' their input into a very small output range in a very non-linear fashion.



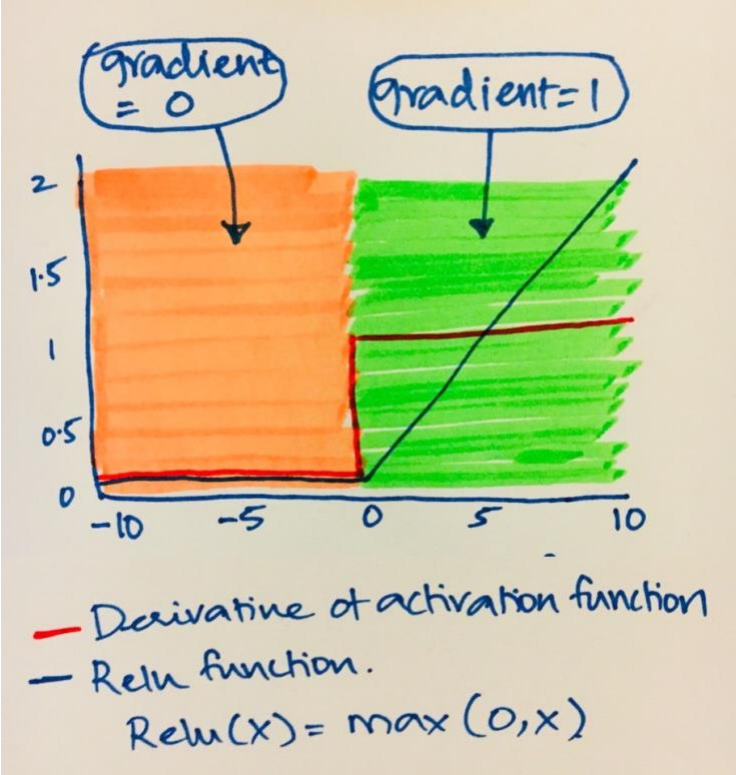
$$\boldsymbol{Sigmoid}\ (\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{(x)}}$$

Sigmoid maps the real number line onto a "small" range of [0, 1] resulting to a large region of the input space getting mapped to an extremely small range. This becomes much worse when we stack multiple layers of such non-linearity's on top of each other. Even a large change in the parameters of the first layer doesn't affect the output much.

This implies that for the "earlier" layers the gradient for weights in the network tends to be very small, creating a "Vanishing Gradient" problem. On the other hand, in the "back" layers of the network, if the sigmoid function value is nether too high nor too low, then gradients will be reasonable in magnitude.

3.3. *ReLU* or *Rectified Linear Unit* function outputs zero when the input is smaller than zero. This speeds-up the time to compute it. Since the derivative remains zero when the output is zero making the rate of change of the function zero.

On the other hand, the output is simply the input when the input is greater or equal to zero, making its derivative equal to one. When you graph this derivative, it would look exactly like a typical step function.



$$\boldsymbol{Relu}\,(\boldsymbol{x}) = \boldsymbol{max}(\,\mathbf{0}, \boldsymbol{x}\,)$$

This solves "vanishing gradient problem" to an extent as the activation function's derivative isn't bounded by the range (0, 1). On the flip side, they "die" (output zero) when the input to it is negative.

This can completely block backpropagation because the gradients will just be zero after one negative value has been inputted and will output the same value for almost all of other activities—zero. Since ReLUs cannot "recover" from this, it results to no modification in its weights.

3.4. When we modify the graph to have bipartite connections, $\frac{\partial E_n(w)}{\partial w_{11}^{(l)}}$ will become:

$$\frac{\partial E_n(w)}{\partial w_{11}^{(l)}} = h'\left(a_1^{(l+1)}\right) \sum_{k=1}^{2} w_{k1}^{(l+1)} \delta_k^{(l+2)} z_1^{(l)}$$

In this case the graph is modified to have bipartite connections, to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(l)}}$ for the layer earlier than the $l+1$ layer, we need to consider both the nodes. So their gradient will become zero only when $h'\left(a_1^{(l+1)}\right)$ and $h'\left(a_2^{(l+1)}\right)$ goes to zero (i.e when you add them up, the derivative of the cost function w.r.t $w_{11}^{(l)}$ goes to zero

4. Logistic Regression (40 Marks)

4.1. The plot of separator path in slope-intercept space is showed in Figure 2 and the plot of neg. log likelihood over iterations is presented in Figure 3.

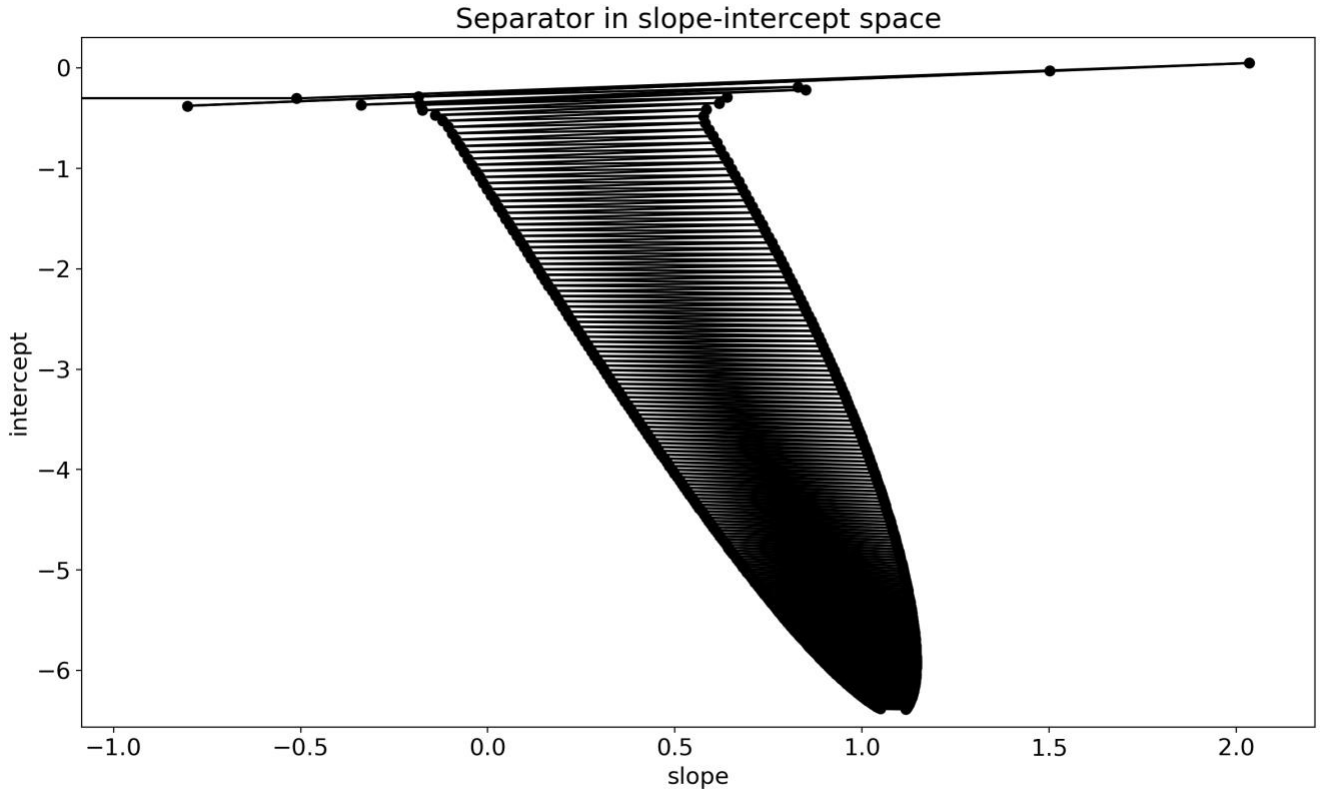


Figure 2: Separator path in slope-intercept space

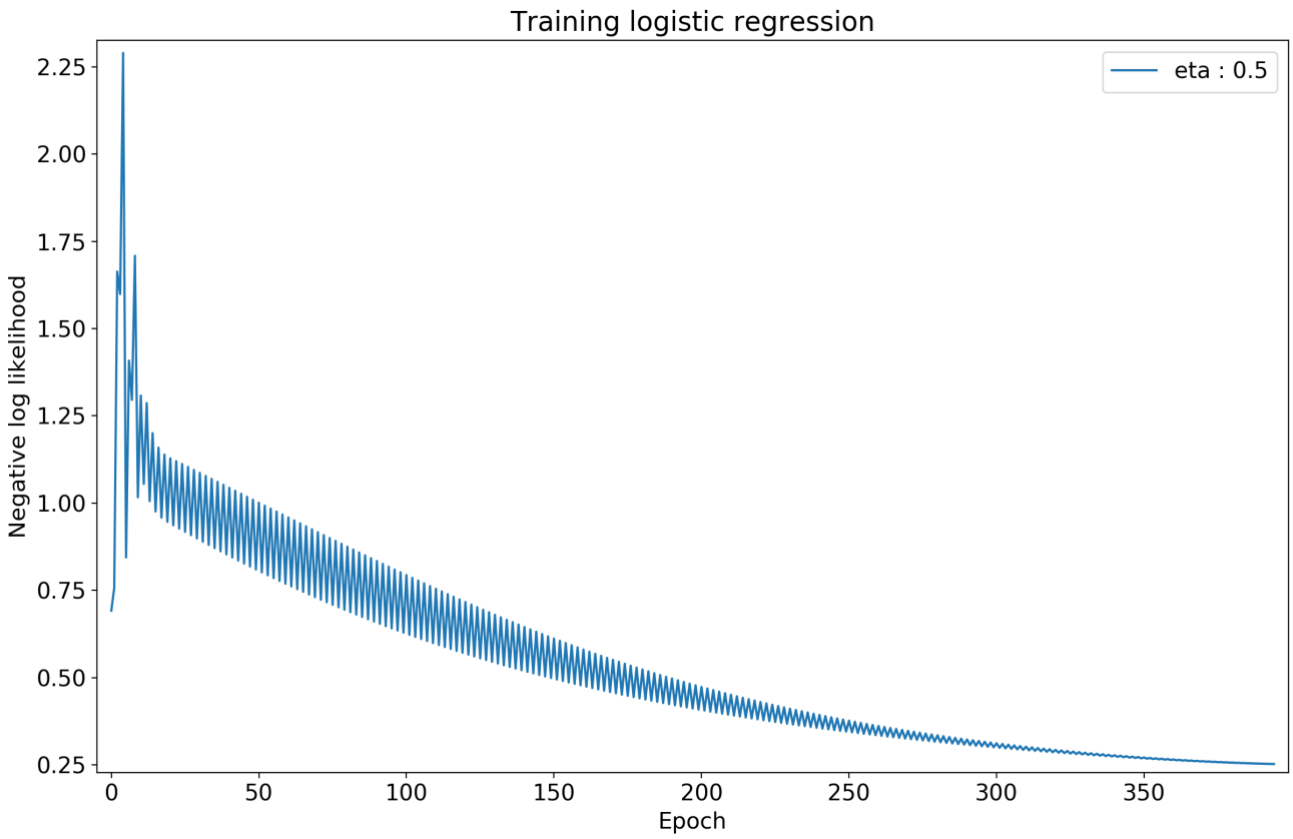


Figure 3: Plot of neg. log likelihood over iterations

Why are these plots oscillating?

The oscillation witnessed in the above plots is caused by the learning rate and the direction of gradient descent. Since the gradient descent method evaluates the gradient and the current value and then steps a finite amount in that direction, the learning rate (0.5) would be too large resulting to gradient descent jumping a large distance in each iteration.

Due to this the line oscillates around the minimum value, constantly stepping past it then turning around and stepping past the minimum again. Also the direction of the gradient is not absolutely pointing to the bottom, therefore inducing a lot of oscillation.

4.2. Logistic Regression for multiple eta values:

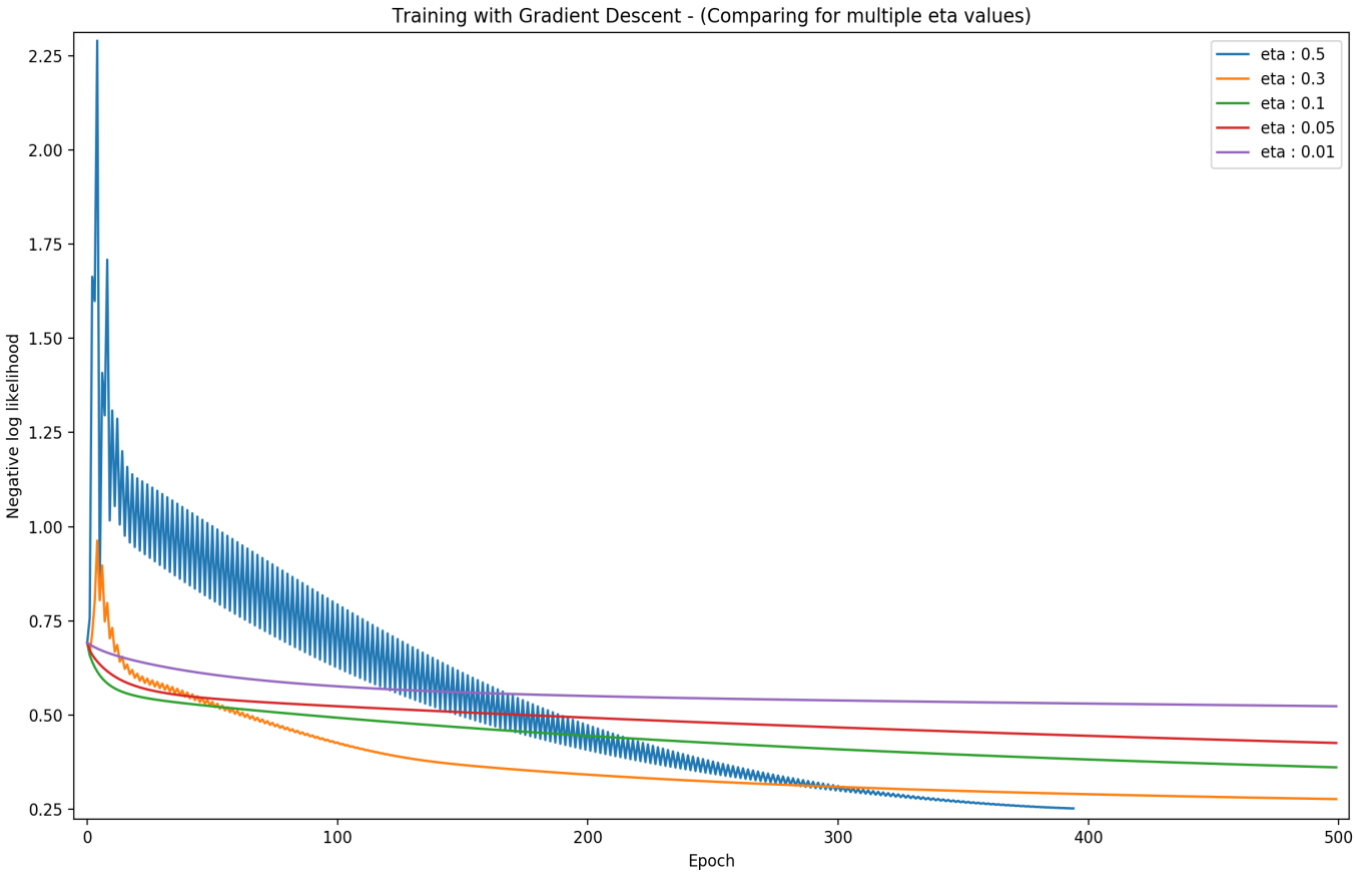


Figure 4: Plot of neg. log likelihood over iterations for multiple eta values

Compare these results. What are the relative advantages of the different rates?

As you can see in Figure 4, It appears as though a moderate step size would produce the best results. A step size that is too large descends quickly but can oscillate around the minimum and does not produce the smallest error where as if it is too small, it converges slowly. Even though the gradient pushes the system towards the minimum and might produce more accurate results, the small step size takes more iterations to reach the minimum.

Therefore, we could conclude that a moderate step size produces a balance between speed and accuracy. Also as the learning rate η grows, the speed of convergence is improved but the oscillation will increase as well. Whereas if η become smaller, the regression will become more smooth but the speed for convergence will be reduced

4.3. Stochastic gradient descent:

The logistic regression script was modified to use stochastic gradient descent. This method uses the same concepts as gradient descent however, instead of taking the gradient of the entire function, stochastic gradient descent calculates the gradient at a single training example and updates the function. As the sample size is so less, I have considered all 200 points in this model.

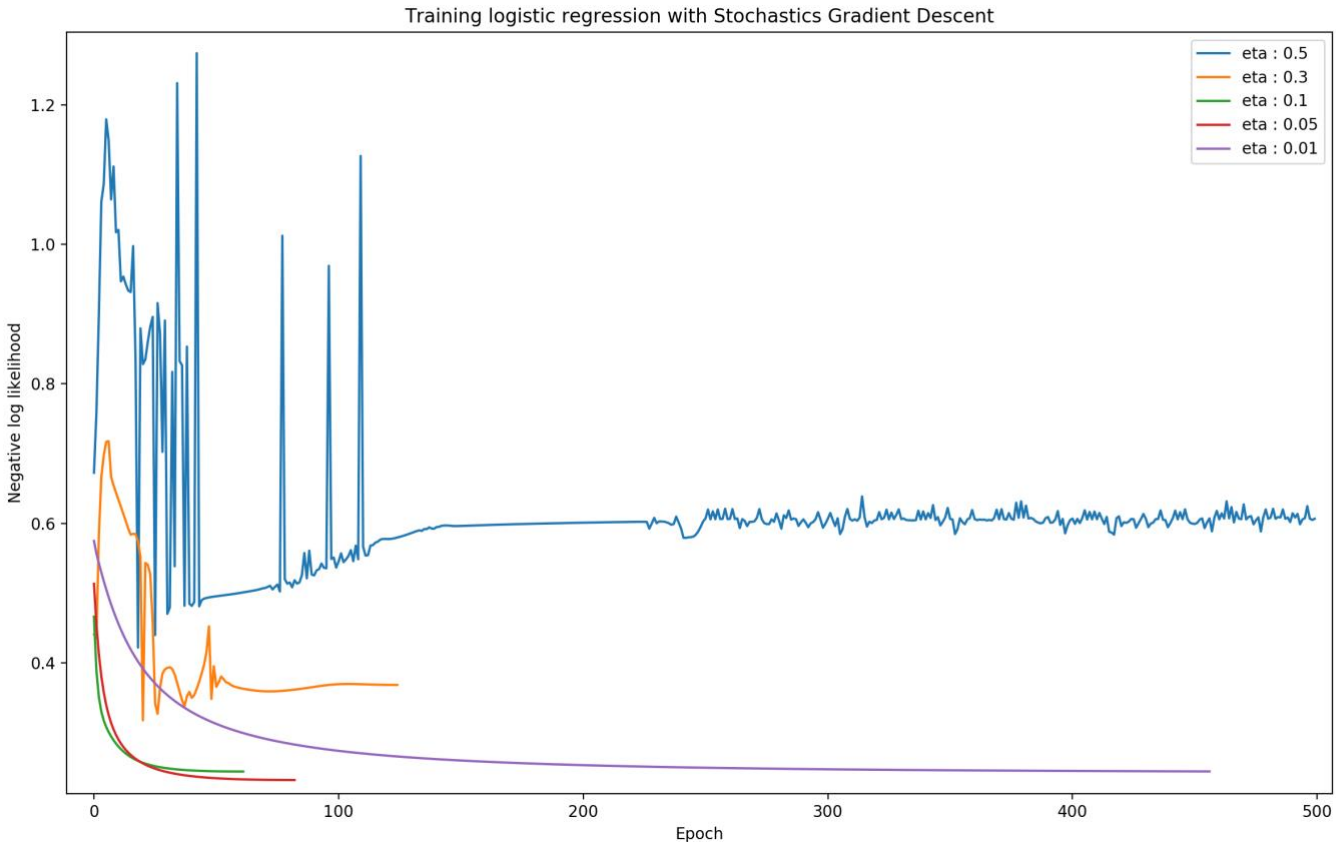


Figure 5: Plot of neg. log likelihood over iterations for multiple eta values - Stochastic Gradient Descent

The stochastic gradient descent method worked faster than the gradient descent method. This can be seen in the graphs. For all step sizes, the error function for the stochastic gradient descent reached a lower error level then the error level on the gradient descent graph in less iterations.

5. Fine-Tuning a Pre-Trained Network (30 marks)

Objective: Write a Python function to be used at the end of training that generates HTML output showing each test image and its classification scores. You could produce an HTML table output for example.

Implementation:

Training the model:

A deep neural network is constructed starting from ResNet 50 up to its average pooling layer. Then, a small network with 32 hidden nodes then 10 output nodes (dense connections) is added on top. Then the weights of the ResNet 50 portion with the parameters from training on ImageNet are initialized.

The code performs training on only the new layers using CIFAR10 dataset and all other weights are fixed to their values learned on ImageNet.

Training Parameters:

A sample size of 50000 images spanning over 10 classes are taken to train the model. Also, the entire network was trained for 3 epochs. The 10 classes are as follows:

[plane , car , bird , cat , deer , dog , frog , horse , ship , truck]

Testing the model:

In this part of the code,

#Step 1: I have first created a test set containing 10000 images spanning over 10 classes. This test set was loaded into *testloader* and a batch size of 1 with shuffle enabled is passed here.

#Step 2: Later *the ResNet50_CIFAR()* model is initialized and the saved dictionary values of our training model are loaded onto it.

#Step 3: html_header(): This function get the header part of the html code you want to write the output to.

#Step 4: Then we iterate through each test sample and get the tensor image as well as its corresponding label index.

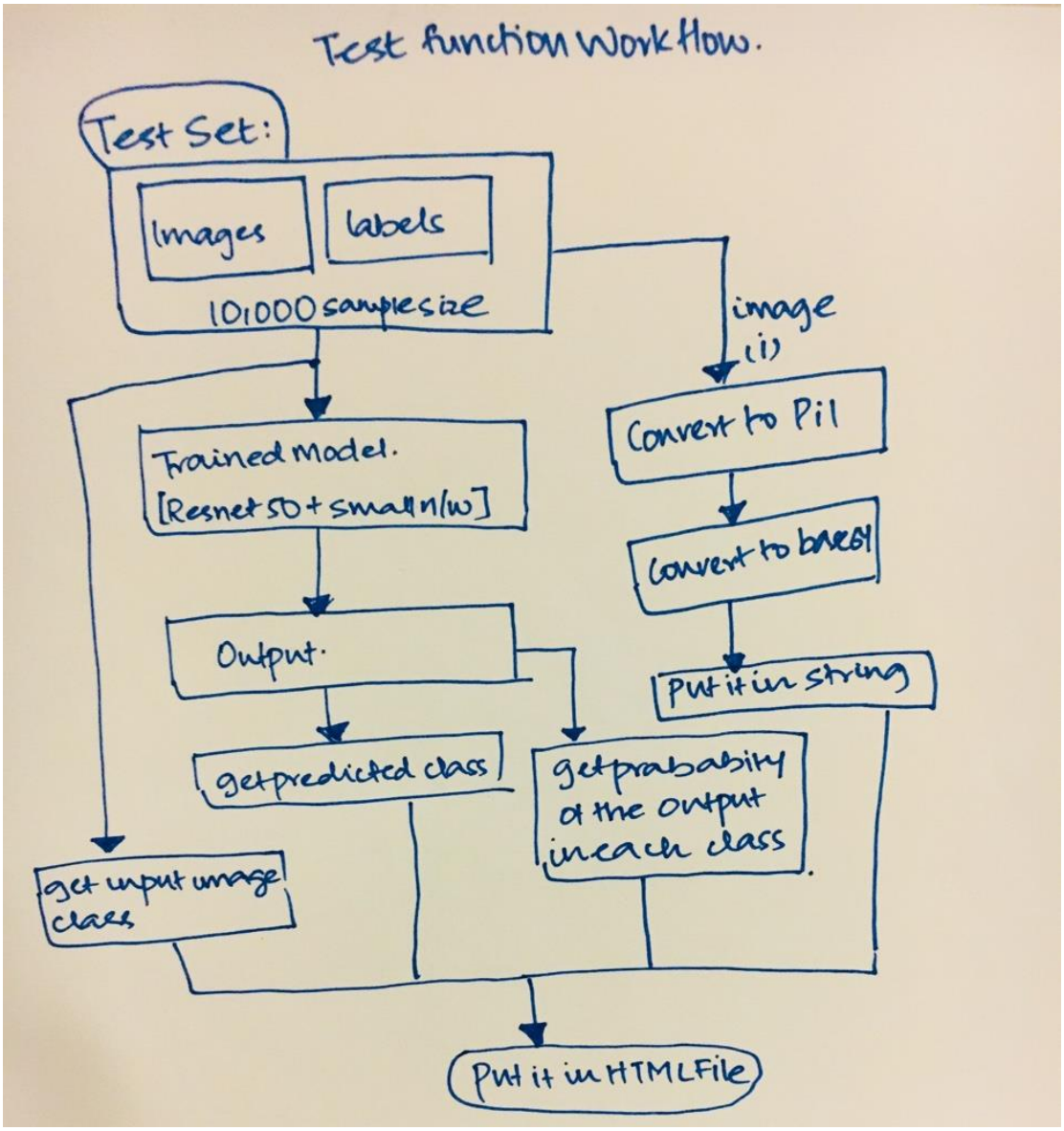


Figure 6: Test Function workflow

#Step 5:

convert_to_pil(inputs):
This function takes in the tensor image and converts it into a PIL image for further processing in the code. Here the input is not cuda enabled as that feature doesn't exists.
def convert_to_pil(inputs): *images_p = inputs / 2 + 0.5* *return(transforms.ToPILImage()(images_p.squeeze()))*

#Step 6:

image_to_base64(image_t):
This function converts image from PIL format to base 64 in order to embed it into a html without saving the image locally.
def image_to_base64(image_t): *in_mem_file = BytesIO()* *image_t.save(in_mem_file, format = "PNG")* *in_mem_file.seek(0)* *img_bytes = in_mem_file.read()* *base64_encoded_result_bytes = base64.b64encode(img_bytes)* *base64_encoded_result_str =* *base64_encoded_result_bytes.decode('ascii')* *return (""+base64_encoded_result_str)*

#Step 7: Then the input values is then passed to the model and predicted class for the output is found

#Step 8:

get_probability(outputs):
This function takes in the outputs and returns the probabilities for the outputs to be classified into any of the 10 classes.
def get_probability(outputs): *sm = nn.Softmax(dim=-1)* *probabilities = sm(outputs)* *return(probabilities.detach().numpy())*

#Step 9: Lastly we put in the Test Image, Test Image class, the predicted class of that image as well as all individual classification scores for that image into a html file.

Libraries Importing: torchvision, torch, numpy, matplotlib, OS, Image (PIL), base64, BytesIO

Output: A HTML file named **output.html** containing Test Image, Test Image class, the predicted class of that image as well as all individual classification scores. (Next page contains a see a screenshot of it)

Simon Fraser University

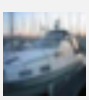
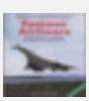
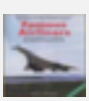
Assignment 2 - CMPT 419/726: Machine Learning, Fall 2018

Name: **Anurag Bejju**
Student ID: 301369375
Date: October 25th, 2018
Professor: Dr. Greg Mori

Question 5: Fine Tuning a Pre-Trained Network:

Test Sample Number	Image	Class	Predicted Class	P(plane)	P(car)	P(bird)	P(cat)	P(deer)	P(dog)	P(frog)	P(horse)	P(ship)	P(truck)
Sample #1		frog	plane	0.340	0.338	0.013	0.046	0.004	0.002	0.182	0.004	0.067	0.004
Sample #2		frog	car	0.014	0.667	0.001	0.003	0.001	0.001	0.171	0.001	0.141	0.001
Sample #3		cat	frog	0.015	0.213	0.011	0.007	0.003	0.009	0.520	0.007	0.205	0.010
Sample #4		cat	frog	0.007	0.036	0.012	0.002	0.002	0.015	0.830	0.005	0.086	0.006
Sample #5		frog	car	0.048	0.434	0.009	0.041	0.004	0.006	0.399	0.006	0.038	0.015
Sample #6		plane	car	0.135	0.689	0.006	0.004	0.007	0.009	0.080	0.002	0.063	0.006
Sample #7		ship	frog	0.012	0.365	0.038	0.012	0.009	0.023	0.444	0.008	0.069	0.020
Sample #8		cat	car	0.044	0.407	0.134	0.081	0.042	0.052	0.154	0.029	0.031	0.025
Sample #9		frog	car	0.079	0.372	0.003	0.068	0.002	0.003	0.325	0.001	0.144	0.004
Sample #10		car	car	0.133	0.292	0.006	0.211	0.004	0.009	0.287	0.002	0.046	0.010
Sample #11		cat	car	0.023	0.713	0.006	0.019	0.004	0.005	0.192	0.004	0.032	0.002
Sample #12		car	car	0.057	0.508	0.003	0.007	0.001	0.003	0.389	0.001	0.027	0.004
Sample #13		plane	car	0.066	0.584	0.020	0.020	0.003	0.004	0.230	0.003	0.064	0.006
Sample #14		ship	car	0.049	0.706	0.028	0.044	0.014	0.019	0.094	0.010	0.014	0.022

Sample #15		cat	frog	0.027	0.387	0.001	0.018	0.000	0.001	0.397	0.001	0.166	0.002
Sample #16		plane	ship	0.147	0.200	0.011	0.104	0.004	0.008	0.191	0.007	0.307	0.021
Sample #17		car	car	0.110	0.385	0.022	0.012	0.008	0.022	0.159	0.007	0.251	0.024
Sample #18		cat	car	0.014	0.392	0.005	0.015	0.001	0.001	0.242	0.003	0.327	0.002
Sample #19		frog	frog	0.103	0.094	0.006	0.018	0.003	0.007	0.601	0.003	0.156	0.009
Sample #20		ship	frog	0.008	0.360	0.005	0.026	0.002	0.009	0.386	0.007	0.186	0.011
Sample #21		cat	car	0.036	0.453	0.011	0.007	0.012	0.030	0.329	0.009	0.085	0.027
Sample #22		cat	car	0.046	0.413	0.024	0.101	0.007	0.003	0.136	0.007	0.253	0.010
Sample #23		frog	car	0.013	0.602	0.002	0.021	0.000	0.001	0.241	0.001	0.118	0.001
Sample #24		frog	car	0.046	0.866	0.014	0.020	0.004	0.002	0.014	0.002	0.022	0.008
Sample #25		ship	car	0.010	0.496	0.004	0.005	0.002	0.010	0.429	0.004	0.037	0.003
Sample #26		frog	frog	0.018	0.084	0.011	0.022	0.001	0.002	0.830	0.001	0.028	0.003
Sample #27		ship	car	0.004	0.641	0.034	0.002	0.007	0.025	0.062	0.007	0.202	0.017
Sample #28		ship	car	0.271	0.422	0.010	0.009	0.007	0.014	0.210	0.003	0.042	0.013
Sample #29		frog	frog	0.010	0.233	0.019	0.045	0.004	0.029	0.512	0.011	0.099	0.039
Sample #30		frog	frog	0.063	0.357	0.001	0.002	0.000	0.001	0.551	0.001	0.025	0.001
Sample #31		car	frog	0.148	0.161	0.008	0.003	0.003	0.008	0.496	0.006	0.156	0.011
Sample #32		cat	car	0.006	0.706	0.008	0.051	0.001	0.001	0.134	0.002	0.085	0.006
Sample #33		car	frog	0.005	0.464	0.002	0.026	0.000	0.001	0.474	0.001	0.024	0.003

Sample #34		ship	car	0.005	0.613	0.004	0.003	0.001	0.004	0.104	0.002	0.262	0.003
Sample #35		frog	frog	0.009	0.340	0.008	0.030	0.003	0.010	0.553	0.007	0.031	0.010
Sample #36		frog	car	0.062	0.562	0.019	0.008	0.004	0.014	0.229	0.006	0.081	0.014
Sample #37		plane	frog	0.117	0.027	0.005	0.138	0.001	0.002	0.672	0.001	0.035	0.004
Sample #38		car	car	0.036	0.784	0.005	0.002	0.003	0.004	0.062	0.008	0.091	0.007
Sample #39		frog	frog	0.040	0.266	0.024	0.020	0.005	0.007	0.490	0.004	0.128	0.016
Sample #40		car	frog	0.005	0.153	0.003	0.028	0.001	0.008	0.769	0.002	0.026	0.005
Sample #41		frog	car	0.026	0.688	0.004	0.024	0.003	0.002	0.188	0.002	0.060	0.002
Sample #42		frog	car	0.004	0.494	0.002	0.000	0.000	0.001	0.469	0.001	0.028	0.001
Sample #43		cat	frog	0.043	0.092	0.014	0.338	0.001	0.002	0.347	0.001	0.157	0.005
Sample #44		car	frog	0.141	0.257	0.005	0.004	0.003	0.007	0.570	0.004	0.005	0.004
Sample #45		ship	car	0.132	0.213	0.127	0.097	0.066	0.097	0.102	0.038	0.059	0.070
Sample #46		car	car	0.014	0.377	0.051	0.110	0.017	0.026	0.258	0.012	0.112	0.021
Sample #47		frog	car	0.048	0.460	0.024	0.028	0.005	0.012	0.298	0.013	0.074	0.039
Sample #48		ship	frog	0.010	0.131	0.019	0.040	0.006	0.012	0.746	0.014	0.010	0.012
Sample #49		plane	car	0.014	0.921	0.002	0.018	0.001	0.000	0.009	0.001	0.033	0.001
Sample #50		plane	car	0.013	0.694	0.029	0.005	0.009	0.022	0.115	0.019	0.059	0.035
Sample #51		ship	car	0.004	0.518	0.002	0.005	0.001	0.008	0.424	0.005	0.030	0.003
Sample #52		car	car	0.011	0.847	0.011	0.011	0.004	0.006	0.087	0.004	0.010	0.008