

Homework 3

*Handed Out: November 8th, 2018**Due: November 19th, 2018, 11:59 PM*

Submitted by: Shruti Sinha

Although the solutions are my own, I consulted with the following people while working on this homework: Hemanth Kothapali, Andres

Please look through pages ahead to find my report corresponding to the sections they are associated with.

1 [Neural Networks - 50 points]

In this problem, you will implement both Feed-forward Neural Network and Convolutional Neural Network (CNN) on the CIFAR-10 image dataset. The goal of this problem is to help you understand how machine learning algorithms could apply to image classification task. In doing it you will better understand neural learning architectures (in particular, how the hyperparameters could affect model performance) and gain some hands-on experience with the deep learning framework Pytorch.

Dataset:

In this homework, you will only use a subset of the CIFAR-10 dataset to train a model and evaluate it. Overall, you will use 10,000 training images, and 2,000 testing images. Each image has a size of 32x32x3 in the RGB scale. The detailed dataset description can be found here: <https://www.cs.toronto.edu/~kriz/cifar.html>

Introduction to Pytorch:

We will use Pytorch, one of the most popular deep learning frameworks nowadays. In this part, you will need to read and understand our Pytorch tutorial before starting to use it. After fully understanding the tutorial code, you should be able to implement the simple feed-forward networks and convolutional neural networks using Pytorch. A useful suggestion is to plot the loss versus training epoch to check if the loss decreases during training. You can use the pytorch tutorial as your reference, and create a new python file to implement the following tasks.

Default Parameters:

As your default parameters, you should use:

1. SGD as the optimizer
2. 0.001 as the learning rate
3. 0.9 as the momentum
4. 64 as the batch size

5. 100 as the maximum number of epochs
6. Cross-entropy loss as the loss function

Experiment 1: Baseline Feed-Forward Neural Network [10 points]

In this part, you will implement a one-hidden layer neural network, and the architecture is shown in the table below. The original image size is 32x32x3, so you need to reshape the input image as a 3072x1 vector and feed into the network. The hidden layer has 1500 neurons followed by sigmoid activation function. The last layer outputs 10 probabilities for each class. Plot the training accuracy over epoch and report the final test accuracy.

Layer	Hyperparameters
Fully Connected1	Out channels = 1500. Then use Sigmoid
Fully Connected2	Out channels = 10. Then use Sigmoid

Note you should complete experiment 1 by programming the class FeedForwardNN in the skeleton python file provided.

Experiment 2: Baseline Convolutional Neural Network [15 points]

The CNN architecture we would like you to use is shown in the table below. Plot the training accuracy over epoch and report the final test accuracy.

Layer	Hyperparameters
Convolution1	Kernel=(3x3), Out channels=7, stride=1, padding=0. Then use ReLU
Pool1	MaxPool operation. Kernel size=(2x2)
Convolution2	Kernel=(3x3), Out channels=16, stride=1, padding=0. Then use ReLU
Fully Connected1	Out channels=130. Then use ReLU
Fully Connected2	Out channels=72. Then use ReLU
Fully Connected3	Out channels=10. Then use Sigmoid

Note you should complete experiment 2 by programming the class ConvolutionalNN in the skeleton python file provided.

Experiment 3: Image Preprocessing [5 points]

In this part, you will explore how image pre-processing and data augmentation can play an important role in the performance of a convolutional neural network. First, instead of using the raw images, you should normalize images before training. Specifically, do the following:

Take each image and normalize pixel values in each of the RGB channel by subtracting its mean and dividing by the standard deviation. For example, if you are normalizing the red channel for an image, then for each of the red pixel values RP_i , you should compute:

$$\frac{RP_i - \text{mean}(RP_i)}{\text{std}(RP_i)}$$

Similarly, follow this guideline to normalize the blue and green channel of each image.

Note you should complete experiment 3 by programming the function `normalize_image` in the skeleton python file provided. You will then have to train your baseline convolutional neural network from Experiment 2 using these normalized images.

Additional Optional Experiments [A. Extra Credit 5 points]: Data augmentation is also a useful technique used to improve the classification accuracy during training. Common data augmentation includes: randomly flipping the image horizontally and/or vertically, randomly cropping the images, and so on. You are encouraged to check and experiment with the Pytorch tutorial link below to explore the different ways you can preprocess the data. (<http://pytorch.org/docs/master/torchvision/transforms.html>). If you choose to do this extra credit, please include all of your work in the final pdf report and clearly label it as **A. Extra Credit**. You should include a mixture of discussions as well as graph plots of your model performance over test accuracy, training accuracy and loss following the data augmentation choices that you have made to the images in the report.

Experiment 4: Hyper-parameterization [10 points]

Hyper-parameter tuning is a very important procedure when training a neural network. In this part, you will change different hyper-parameters for both your baseline feed forward neural network from Experiment 1 and baseline convolutional neural network from Experiment 2 with some suggested values, such as

1. batch size [64, 128, 256],
2. learning rate [0.005, 0.003, 0.001],
3. convolutional kernel size [3x3, 5x5, 7x7],
4. number of neurons in the fully connected layers [your choices], and
5. number of fully connected layers [your choices].

You do not need to try out all possible parameter combinations. The only requirement is that you alter the number of neurons in the fully connected layers or the number of fully connected layers to achieve a better test accuracy than the default settings. Implement the new hyper-parameterized neural network architecture in the `HyperParamsFeedForwardNN` and `HyperParamsConvNN` class as seen in the skeleton python file provided. Feel free to also change the max training iterations to be as large as you want in order to improve the model accuracy.

Additional Optional Experiments [B. Extra Credit 5 points]: You can also try different activation functions and different optimizations, such as Adagrad, SGD and Adam. You are also encouraged to try more advanced CNN architectures, such as AlexNet, VGG, and ResNet. Again, if you choose to do this extra credit, please include all of your work in the final pdf report and clearly label it as **B. Extra Credit**. You should include a mixture of discussions as well as graph plots of your model performance over test accuracy, training

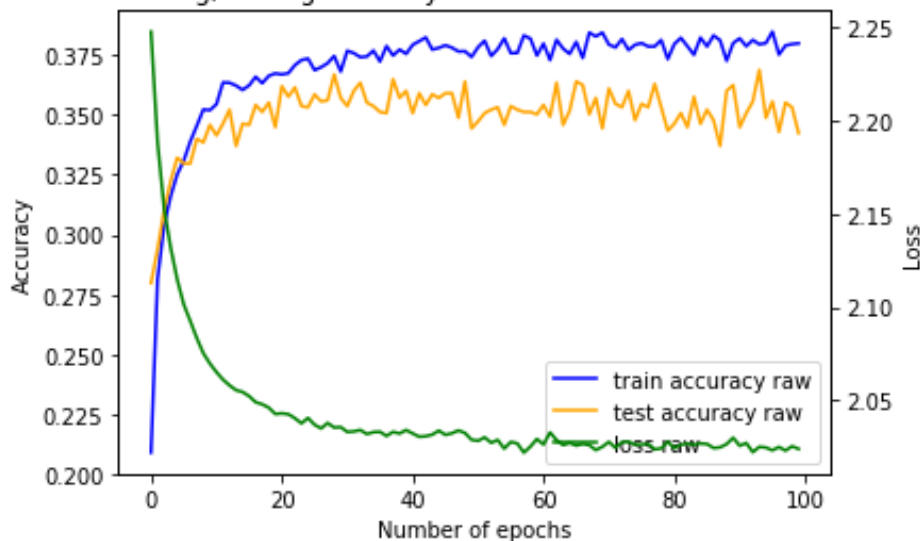
accuracy and loss following your additional experiments in the report.

What to Report: [10 points]

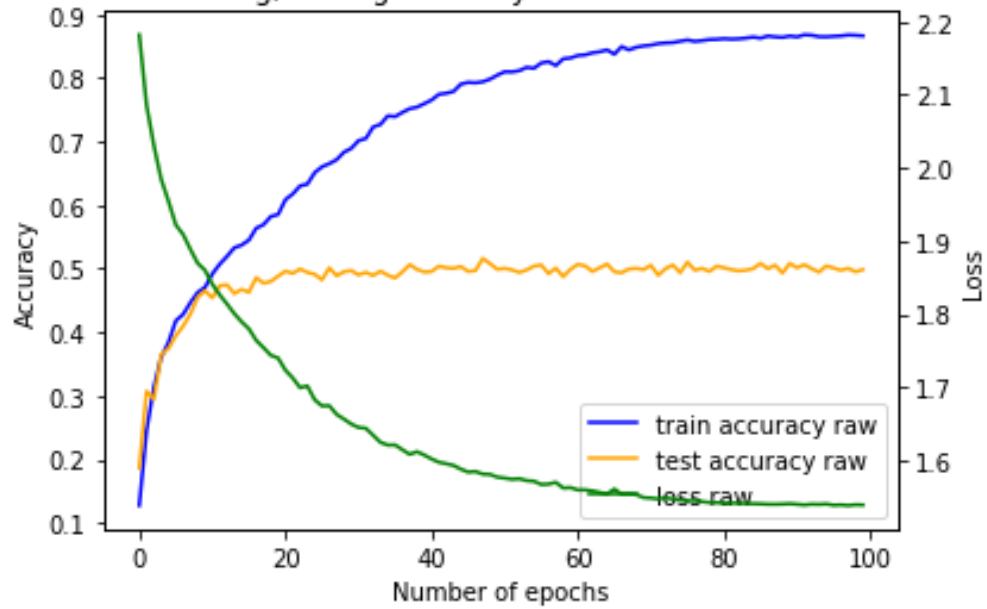
1. **Experiment 1:** Plot the test accuracy, training accuracy and loss of your feed forward neural network over 100 epochs in your report.
2. **Experiment 2:** Plot the test accuracy, training accuracy and loss of your baseline convolutional neural network over 100 epochs in your report.
3. **Experiment 3:** Plot the test accuracy, training accuracy and loss of your baseline convolutional neural network over 100 epochs with the raw images and normalized images on the same graph in your report. Describe the graph by comparing and contrasting their performance over test accuracy, training accuracy and loss over 100 epochs.
4. **Experiment 4:** Plot the test accuracy, training accuracy and loss of your hyperparameterized feed forward neural network and hyperparameterized convolutional neural network over 100 epochs in your report. Provide a table that summarizes the different hyperparameters that you have chosen for both the feed forward neural network and convolutional neural network.
5. **Discussion:** Discuss how the model performance changes from experiment 1 to experiment 4. Do you see any improvements? If so, briefly discuss why? If not, briefly discuss why not? Discuss any other interesting patterns that you observe from the model performance.

1. All plots

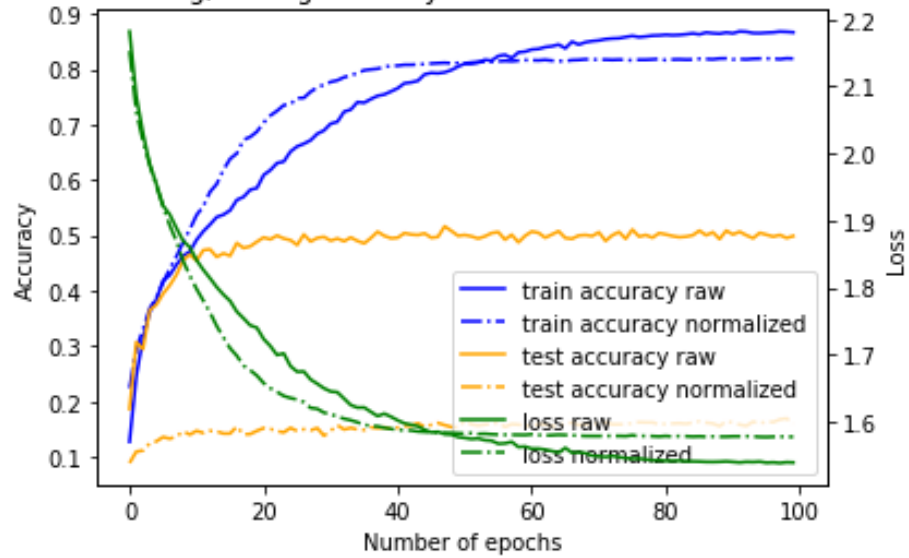
Experiment 2: training, testing accuracy & loss for Baseline Feed Forward raw images



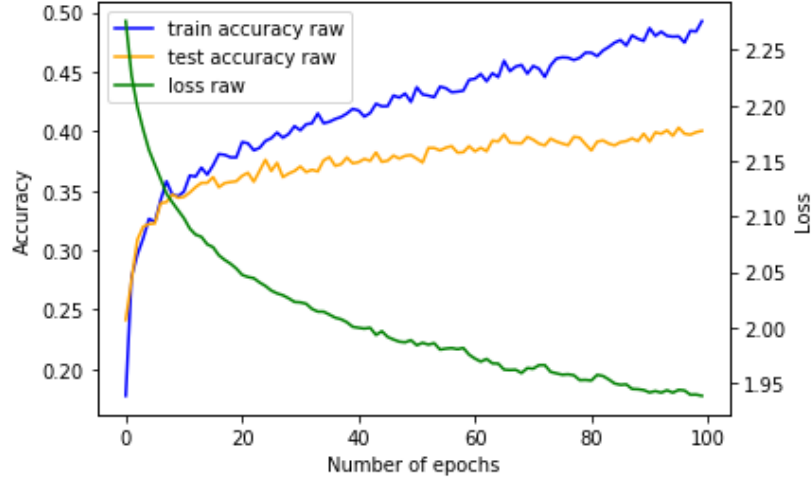
Experiment 2: training, testing accuracy & loss for Baseline CNN raw images



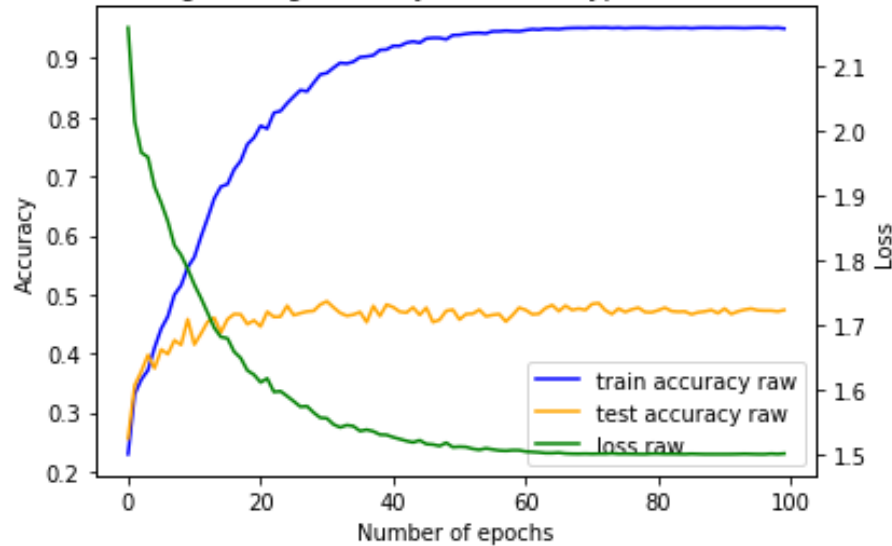
Experiment 3: training, testing accuracy & loss for CNN raw and normalized images



Experiment 2: training, testing accuracy & loss for HyperParameter Feed forward for raw images



Experiment 2: training, testing accuracy & loss for HyperParameter CNN raw images



NOTE:Experiment numbers in the plots are mislabelled, please read the headings.

DISCUSSION:

On moving from experiment 1 to 2, the test and train accuracy improved because we moved from a simpler ff to more complex CNN network,from experiment 2 to 3, the accuracy decreased. This was due to normalization which might not be suitable for our dataset. In experiment 4, we checked various parameters to improve performance of CNN and Feed forward network on raw images. The results are discussed as below:

Experiment1: The first plot shows the training, testing and loss observed over 100 epochs for Feed Forward Neural Networks with raw images .It is observed that with every next epoch the loss decreases and training and test accuracies increases initially and then after some epochs, start varying around the same range.This is because model has fit the data as best as it can.We change different parameters to alter it.

The final training accuracy is 37.96 and test accuracy is 34.22. The loss decreases from 2.25 to 2.02 over 100 epochs.

Experiment2: The plot shows the training, testing and loss observed over 100 epochs for Baseline Convolution Neural Networks with raw images. It is observed that training accuracy and test accuracy increases with epochs but the loss decreases.

The final training accuracy is 86.6 and test accuracy is 49.85. The loss decreases from 2.2 to 1.4 over 100 epochs.

Experiment3: The plot shows the training, testing accuracy and loss observed over 100 epochs for Baseline Convolution Neural Networks with raw and normalised images respectively. It is observed that after normalisation the training and testing accuracy both drop. There is only a slight drop in training accuracy, however test accuracy decreases considerably. This might be due to the dataset given to us. It is possible that the data given, has inputs such that some are of more importance than others. By normalising we give each input equal weight and thus it might not be suitable in this case.

The loss curves are almost the same.

For CNN raw: training accuracy is 86.6 and test is 49.85.

For CNN normalised: training accuracy is 81.8 and testing is 15.9.

Experiment4: HyperParameter

HYPERPARAMETERS FOR CONVOLUTIONAL NN

Layer	Hyperparameters(lr=0.005, batch size=64)
Convolution1	Kernel=(3x3), Out channels=7, stride=1, padding=0. Then use ReLU
Pool1	MaxPool operation. Kernel size=(2x2)
Convolution2	Kernel=(3x3), Out channels=16, stride=1, padding=0. Then use ReLU
Fully Connected1	Out channels=250. Then use ReLU
Fully Connected2	Out channels=100. Then use ReLU
Fully Connected3	Out channels=10. Then use Sigmoid

i	learning rate	kernel size	batch size	neurons	train accuracy	test accuracy
2	0.001	3	64	250,100	49.5	89.1
3	0.005	3	256	250,100	39.2	84.6
4	0.005	3	128	250,100	47.8	93.2
5	0.005	3	64	250,100	51.1	95.5
6	0.005	7	64	250,100	11.1	20.5
7	0.005	3	64	90,30	0.09	0.10

HYPER PARAMETERS FOR FEED FORWARD NN

Layer	Hyperparameters(lr =0.001, Batch=256)
Fully Connected1	Out channels = 2000. Then use Sigmoid
Fully Connected2	Out channels = 10. Then use Sigmoid

i	learning rate	batch size	neurons	fc layers	test accuracy	train accuracy
1	0.001	64	2000,10	2	36.2	38.6
2	0.005	64	2000,10	2	33.8	34.1
3	0.001	128	2000,10	2	38.58	45.17
4	0.005	128	2000,10	2	29.10	29.16
5	0.001	256	2000,10	2	40.5	48.5
6	0.001	64	2000,500,10	3	35.5	37.4
7	0.001	128	1000,10	2	37.53	41.2
8	0.001	128	1500,10	2	37.5	43.5

HyperParamConvNN : We can see from the plot and the table that i(5)(51.1,95.5) gave slightly better performance than our Baseline CNN(49.6,89.4). However I believe their performance is comparable.

- 1)It was observed that decreasing neurons in fc layer resulted in decrease in accuracies. Thus, we increased the no. of neurons to 250 and 100.
- 2)As the batch size increased, the accuracies decreased.
- 3)A higher learning rate gave better training accuracy and slightly better test accuracy.
- 4)On changing the kernel size, the accuracies dropped considerably.

Thus, we chose the hyperparameters as shown in the table.

HyperFeedForwardNN: By observing the plots for HyperParamFeedForward Neural Network, we see that the maximum training accuracy is 48.5 and testing accuracy is 40.5, which is higher than the one observed for Baseline Feed Forward neural network which had a training accuracy of 37 and testing accuracy of 34.

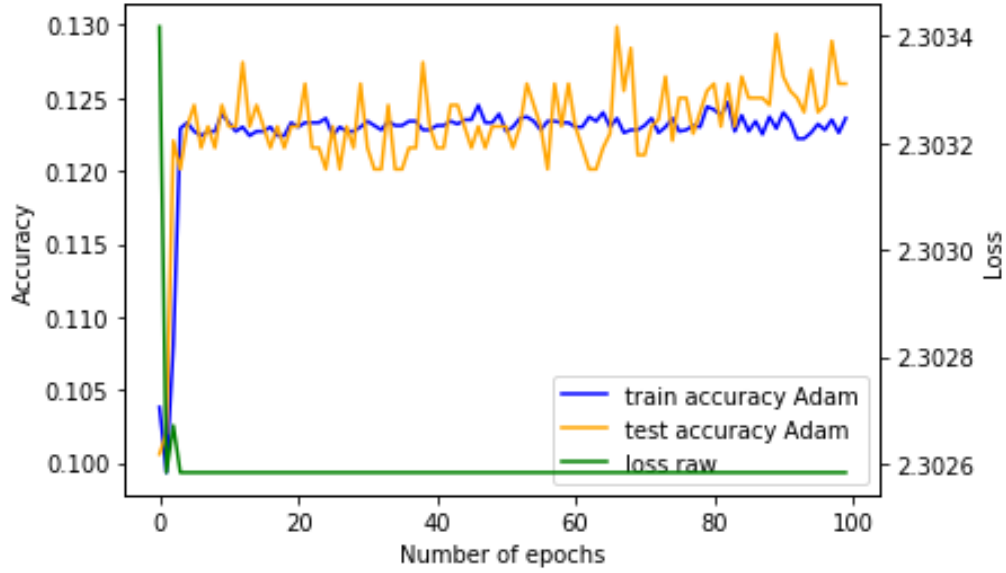
- 1)It is also observed that by increasing the number of neurons alongwith the increase in batch size gave better accuracies, as more examples are now available for the model to train in every batch.
- 2)The increase in number of neurons also increases accuracy as can be seen from the table. As we increase neurons from 1000 to 2000, the testing accuracy increases alongwith train accuracy.
- 3)It was also observed that with increase in learning rate the accuracies dropped. This might be because the larger steps taken skip through local minima. A slower learning rate 0.001 takes computationally longer than 0.005 and is able to traverse through local minimas.
- 4)The increase in fc layers also resulted in a drop in accuracies.

Better hyperparameters might be found by using different combinations , but the among combinations we tried, i(5) gave the best testing accuracy. Therefore we choose the hyperparameters as listed above.

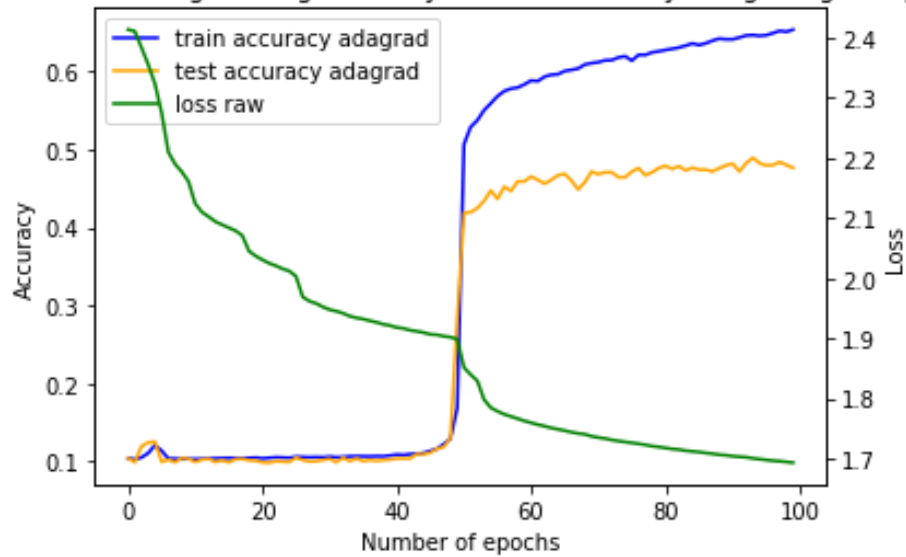
B.EXTRA CREDIT We tried different optimizers, Adam and Adagrad on BaseLine CNN. The following plots were obtained.

1. All plots

Experiment 2: training, testing accuracy & loss for CNN by using Adam optimiser



Experiment 2: training, testing accuracy & loss for CNN by using Adagrad optimiser



For Adagrad: training accuracy was 47.3 and test accuracy was 65.7

For Adam: training accuracy was 12.4 and test accuracy was 12.9

Comparing it to baseline CNN with SGD, these perform poorly in terms of training, however Adam gets almost the same test accuracy as for SGD. It is possible that we could change other parameters of Adagrad and Adam to get better accuracies. But with learning rate=0.001, Adam was computationally slower than the other two. Moreover, the train accuracy for Adagrad suddenly increases after going through half the epochs. In contrast, for Adam the train accuracy gets almost linear after only a few epochs in the start.

i	Label	Hypothesis 1				Hypothesis 2			
		D_0	$f_1 \equiv [x > 1]$	$f_2 \equiv [y > 4]$	$h_1 \equiv [x > 1]$	D_1	$f_1 \equiv [x > 9]$	$f_2 \equiv [y > 11]$	$h_2 \equiv [y > 11]$
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
1	−	0.1	−	+	−	0.0625	−	−	−
2	−	0.1	−	−	−	0.0625	−	−	−
3	+	0.1	+	+	+	0.0625	−	−	−
4	−	0.1	−	−	−	0.0625	−	−	−
5	−	0.1	−	+	−	0.0625	−	+	+
6	−	0.1	+	+	+	0.25	−	−	−
7	+	0.1	+	+	+	0.0625	+	−	−
8	−	0.1	−	−	−	0.0625	−	−	−
9	+	0.1	−	+	−	0.25	−	+	+
10	+	0.1	+	+	+	0.0625	−	−	−

Table 1: Table for Boosting results

2 [Boosting - 25 points]

Consider the following examples $(x, y) \in \mathbb{R}^2$ (i is the example index):

i	x	y	Label
1	0	8	−
2	1	4	−
3	3	6	+
4	-2	1	−
5	-1	13	−
6	9	11	−
7	12	7	+
8	-7	-1	−
9	-3	12	+
10	5	9	+

In this problem, you will use Boosting to learn a hidden Boolean function from this set of examples. We will use two rounds of AdaBoost to learn a hypothesis for this data set. In each round, AdaBoost chooses a weak learner that minimizes the error ϵ . As weak learners, use hypotheses from the following classes of functions: either functions of the form (a) $f_1 \equiv [x > \theta_x]$, or functions of the form (b) $f_2 \equiv [y > \theta_y]$, for some integers θ_x, θ_y . There is no need to try many values of θ_x, θ_y ; appropriate values should be clear from the data.

1. **[7 points]** Start the first round with a uniform distribution D_0 . Place the value for D_0 for each example in the third column of Table 1. Find the best weak learners, f_1 and f_2 , by selecting an integer θ_x for f_1 and an integer θ_y for f_2 that minimizes their errors. Then, fill up the table based on the prediction of $f_1 \equiv [x > \theta_x]$ in column 4 and $f_2 \equiv [y > \theta_y]$ in column 5 for each of the example in the Table 1 above.

2. [6 points] Find the hypothesis given by the weak learner that minimizes the error ϵ for that distribution. Place this hypothesis as the heading to the sixth column of Table 1, and give its prediction for each example in that column.
3. [7 points] Now compute D_1 for each example, find the new best weak learners f_1 and f_2 , and select hypothesis that minimizes error on this distribution, placing these values and predictions in the seventh to tenth columns of Table 1.
4. [5 points] Write down the final hypothesis produced by AdaBoost.

What to submit: Fill out Table 1 as explained, show computation of α and $D_1(i)$, and give the final hypothesis, H_{final} . This should all be included in the final pdf report along with your neural network experiments.

Calculation:

Hypothesis 1: Since we have uniform distribution, we have all $D_0(i) = 0.1$, column 3 as 0.1, where $0 < i \leq 10$

We get $f_1 \equiv [x > 1]$ and $f_2 \equiv [y > 4]$. We choose the final hypothesis as the one among which makes less mistakes.

$x > 1$ makes 2 mistakes for i(6,9) giving $\epsilon_t = 0.2$, while $y > 4$ makes 3 mistakes for i(1,5,6) giving $\epsilon_t = 0.3$.

Thus, we choose $x > 1$ as our hypothesis1 $h_1 \equiv [x > 1]$ and makes predictions based on this hypothesis to fill column 5.

For calculation of D_1 , We first need to find Z_t .

$$Z_t = \sum_i 0.1 \exp(-\alpha_t y_i h_t(x_i))$$

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \alpha_t = \frac{1}{2} \ln\left(\frac{1 - 0.2}{0.2}\right) = 0.693$$

$$Z_t = 0.1e^{-0.693} * 8 + 0.1e^{0.693} * 2 = 0.8$$

For correct classification: $D_1(i) = D_0(i)/Z_t e^{-\alpha_t}$

$$D_1(i) = \frac{0.1e^{-0.693}}{Z_t} = 0.0625$$

For incorrect classification: $D_1(i) = D_0(i)/Z_t e^{\alpha_t}$

$$D_1(i) = \frac{0.1e^{0.693}}{Z_t} = 0.25$$

We again find the hypothesis. To find hypothesis 2 h_2 , we get $f_1 \equiv [x > 9]$ and $f_2 \equiv [y > 11]$. $x > 9$ makes 6 mistakes giving $\epsilon_t = 4 * 0.25 + 2 * 0.0625 = 1.125$, while $y > 11$ makes 4 mistakes giving $\epsilon_t = 0.0625 * 4 = 0.25$.

Thus, we choose $y > 11$ as our hypothesis2 $h_2 \equiv [y > 11]$ and makes predictions based on this hypothesis to fill column 10. We find,

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - 0.25}{0.25}\right) = 0.549$$

$$H_{final} = \text{sign} \sum \alpha_t h_t(x) = \text{sign}(0.693 * h_1 + 0.549 * h_2)$$

3 [SVM - 25 points]

We have a set of six labeled examples D in the two-dimensional space, $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(6)}, y^{(6)})\}$, $\mathbf{x}^{(i)} \in \mathbb{R}^2$ and $y^{(i)} \in \{1, -1\}$, $i = 1, 2, \dots, 6$ listed as follows:

i	$\mathbf{x}_1^{(i)}$	$\mathbf{x}_2^{(i)}$	$y^{(i)}$
1	0	-2	1
2	1.6	-2.4	1
3	-2.6	1.3	-1
4	2.5	-0.3	1
5	-0.2	3	-1
6	-2	0	-1

(a) [4 points] We want to find a linear classifier where examples \mathbf{x} are positive if and only if $\mathbf{w} \cdot \mathbf{x} + \theta \geq 0$.

1. [1 points] Find an easy solution (\mathbf{w}, θ) that can separate the positive and negative examples given.

Define $\mathbf{w} = [1, 0]$

Define $\theta = [0]$

2. [4 points] Recall the Hard SVM formulation:

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \tag{1}$$

$$\text{s.t } y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + \theta) \geq 1, \forall (\mathbf{x}^{(i)}, y^{(i)}) \in D \tag{2}$$

What would the solution be if you solve this optimization problem? (Note: you don't actually need to solve the optimization problem; we expect you to use a simple geometric argument to derive the same solution SVM optimization would result in).

Define $\mathbf{w} = [0.5, -0.5]$

Define $\theta = [0]$

The solution to this problem was to find the maximum margin hyperplane that separates the xi points which have yi = -1 (negative examples) and those which have yi= 1(positive examples).

By observing the examples geometrically, we can see that points [0,-2] and [-2,0] are the examples that would become support vectors(closest points to separator)

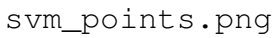


Figure 1: Training examples for SVM in question 1.(a)

and would give the maximum margin. We need to find the hyperplane that is perpendicular bisector to the line connecting the 2 points. The hyperplane would thus go through the origin and hence $\theta = 0$.

Now, to get the weight vector, we see that the weight vector points from $(-2,0)$ to $(0,-2)$ (negative to positive example). Thus we would want weight vector to point from the middle of the line connecting those two, that is $(-1,-1)$ to $(0,-2)$. Thus we get weight vector after scaling as $w = [1/2, -1/2]$. Hence, we find the perpendicular bisector of the line connecting the two points to get the solution.

3. [5 points] Given your understanding of SVM optimization, how did you derive the SVM solution for the points in Figure 1?

We want the hyperplane that separates positive and negative examples with the maximum margin. Thus, we can derive the solution from the support vectors which are the points $[0, -2]$ and $[-2, 0]$ (closest points to the separator).

We need to find the hyperplane that is perpendicular bisector to the line connecting the 2 points. The hyperplane would thus go through the origin and hence $\theta = 0$. Now, to get the minimum weight vector, we look at the equations using $y_i(w^T x_i + \theta) = 1$:

$$-1 * (w_1 * -2 + w_2 * 0 + \theta) = 1$$

$$1 * (w_1 * 0 + w_2 * -2 + \theta) = 1$$

Solving these mathematically, we get, $w_1 = 0.5, w_2 = -0.5$ which is the same as that obtained geometrically.

Thus $w = [0.5, -0.5], \theta = 0$

- (b) [15 points] Recall the dual representation of SVM. There exists coefficients $\alpha_i > 0$ such that:

$$\mathbf{w}^* = \sum_{i \in I} \alpha_i y^{(i)} \mathbf{x}^{(i)} \quad (3)$$

where I is the set of indices of the support vectors.

1. [5 points] Identify support vectors from the six examples given.

Define $I = [0, -2]$ and $[-2, 0]$ (points 1 and 6)

2. [5 points] For the support vectors you have identified, find α_i such that the dual representation of \mathbf{w}^* is equal to the primal one you found in (a)-2.

Define $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_{|I|}\} = 0.25, 0.25$

$\alpha_1 = 0.25, \alpha_2 = 0.25$

We find this using the equation given,

$$\alpha_1 = -1/4 = -0.25$$

$$1 * \alpha_1 [0, -2] + \alpha_2 [-2, 0] * -1 = [0.5, -0.5]$$

Solving we get,

$$\alpha_1 = 1/4 = 0.25, \alpha_2 = 0.25$$

Thus, we get $\alpha = [0.25, 0.25]$

3. [5 points] Compute the value of the hard SVM objective function for the optimal solution you found.

Objective function value = 0.25

We calculate this using, $1/2 \|\mathbf{w}\|^2 = \frac{1}{2} (\sqrt{0.5^2 + 0.5^2})^2 = 0.25$

- (c) [10 points] Recall the objective function for soft representation of SVM.

$$\min \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{j=1}^m \xi_j \quad (4)$$

$$\text{s.t } y^{(i)} (\mathbf{w} \cdot \mathbf{x}^{(i)} + \theta) \geq 1 - \xi_i, \xi_i \geq 0, \forall (\mathbf{x}^{(i)}, y^{(i)}) \in D \quad (5)$$

where m is the number of examples. Here C is an important parameter. For which **trivial** value of C , the solution to this optimization problem gives the hyperplane that

you have found in (a)-2? Comment on the impact on the margin and support vectors when we use $C = \infty$, $C = 1$, and $C = 0$. Interpret what C controls.

Sol:

C parameter tells SVM optimization how much misclassification is allowed. C is a parameter that determines the constraint placed on empirical error. It determines how much regularization do we want. A higher value of C thus implies that we want to minimize empirical error and thus the examples allowed inside the margin are only a few. However a lower value of C implies that we tolerate empirical error and thus more than a few examples are allowed inside the separator. Thus a low C , implies more support vectors.

For $C = \infty$, implies no examples are allowed inside the support vectors, thus the soft SVM turns into hard SVM. The optimization would choose a smaller margin hyperplane if all the training points are classified correctly and outside margin. Thus, a lot of importance is given to minimizing empirical errors. No examples are allowed inside the support vectors and there are higher constraints placed. It thus gives big weight vector and small margin.

For $C = 0$, implies that very little/no importance is given to minimizing empirical error, thus many examples are allowed inside the margin. This means, that the objective function is now given the freedom to increase —w— a lot, leading to high error and margin. Thus, no weight on empirical error. The function tolerates error and tries to find small weight vector (large margin).

For $C = 1$, implies that we give some importance to minimizing empirical error. The support vectors are more than $C = \infty$ but less than $C = 0$. It allows some errors while minimizing the weight vector to find a large margin. Smaller weight vector than $C = \infty$ case

Therefore for (a)-2, we have hard SVM, we achieve that by having $C = \infty$.

Checklist:

1. Submit your report in a pdf format to Canvas. Make sure to include your experiments, answers to Boosting and SVM as well as any extra credits in your report.
2. Submit your report in a pdf format to Gradescope. Make sure to include your experiments, answers to Boosting and SVM as well as any extra credits in your report.
3. Submit your code to Canvas.
4. Comment out any code which is not an import statement or contained in a function definition, then submit it to Gradescope and pass the autograder.

Note on Grading for Homework 3

For the programming portion of the assignment, we will only be conducting simple test cases on Gradescope. If you pass all the test cases on Gradescope, you should automatically receive 20 points. The remaining 30 points including the write up of your report will be graded manually for correctness.