Lab 3

CS 3793/5233 – Fall 2015
Tom Bylander, Instructor

In Lab 3, you will implement a program to learn to classify images. Your grade on the lab will depend on the behavior and accuracy of the program that is implemented. The initial program is cifar.zip, which you can download from the course web site.

CIFAR Environment

The CIFAR-10 dataset at http://www.cs.toronto.edu/~kriz/cifar.html contains 60,000 images each labeled with one of ten classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). Each image is $32 \times 32$ in RGB. This lab will consider the first 10,000 of these images.

The Cifar program repeatedly does the following steps. It reads an image and fills three arrays with the red, green, and blue values of each pixel. It outputs “ready”. It then inputs a prediction, a single digit on a line by itself. After the prediction, it outputs the correct label and a summary of the error rate so far.

Running the main method in Cifar.java will display a picture of the image for you to classify.

CIFAR Agent

The CifarLearner code currently repeats the following steps:

1. Wait for a “ready” message.
2. Convert the color image into a grayscale image.
3. Convert the grayscale image into 769 features.
4. Call LinearLearner.makePrediction and output the prediction. Currently, this method always returns 0.
5. Read the target value.
6. Call LinearLearner.updateWeights. This method currently does nothing.

Your tasks are to code:

1. LinearLearner.makePrediction so it makes a prediction based on the weights $w$ and the example $x$.
2. LinearLearner.updateWeights so the weights are updated if the prediction differs from the target.

You will end up with a program that performs online learning, that is, its accuracy will improve as additional examples are presented to it. The suggested algorithm is a version of perceptrons. The initial code download reads in all the examples when Interact.java is run, but it always predicts 0. The methods you need to modify are in LinearLearner.java.
LinearLearner.makePrediction

The array $\mathbf{w}$ contains 7690 weights, a set of 769 weights for each category. For category $i$, you need to compute the dot product of $w_i = \mathbf{w}[i]$ with the example $x$:

$$y_i = w_i \cdot x = \sum_j w_{ij} x_j$$

The prediction corresponds to the $y_i$ with the highest value. In case of ties, choose the lower number.

LinearLearner.updateWeights

If the prediction $p$ is not equal to the target $t$, then add the example $x$ to the weights for the target $w_t = \mathbf{w}[t]$ and subtract $x$ from the weights for the prediction $w_p = \mathbf{w}[p]$. For each $j$ from 0 to 768:

$$w'_{tj} = w_{tj} + x_j$$

$$w'_{pj} = w_{pj} - x_j$$

Expected Results and a Contest

Do not include the data file `data_batch_1.bin` in your submission. Doing so will be a 20-point deduction. Grading will be done by copying this file to the appropriate location in your java code: either the same directory as your source code or one level up.

A correct implementation should have an error rate around 66% over all 10,000 examples. The output of my program is at the end of this assignment. 66% is not very good, but better results require much more sophisticated features and learning algorithms, plus much more computing time. However, 66% error shows that some learning is happening because random guessing would have a 90% error.

There are several ways to improve the error rate. Two are discussed below. The five best error rates in this lab (must be under 65%) will receive an additional 100, 80, 60, 40, or 20 points.

An alternative weight update that can achieve about 60% error is to compare the target’s dot product $y_t$ and the highest dot product $y_k$ of the other categories. If $y_k + 1 > y_t$, then update the weights by:

$$w'_{tj} = w_{tj} + \alpha x_j$$

$$w'_{kj} = w_{kj} - \alpha x_j$$

where $\alpha$ is the learning rate. To find the best value for $\alpha$, start looking at $\alpha = 0.1$.

Another way to improve this program is better features. One set of features you could add to do a few percent better is a color histogram. You could divide up the red, green, and
blue values into four intervals, giving you $4 \times 4 \times 4 = 64$ bins. There are a few places in the code that you need to modify if you add 64 features, plus you need to pass the red, green, and blue values to `CifarFeatures.allFeatures`. Be sure to normalize the 64 values, else you will make things worse.

**Testing Your Program**

A correct implementation should run in less than a minute (it takes about 10-20 seconds on my 6-year-old laptop), and produce something like the following output when the main method in `Interact.java` is run.

```java
class Cifar ready
class CifarLearner 0
class Cifar 6 error rate = 1/1
class Cifar ready
class CifarLearner 6
class Cifar 9 error rate = 2/2
class Cifar ready
class CifarLearner 9
class Cifar 9 error rate = 2/3
class Cifar ready
class CifarLearner 9
class Cifar 4 error rate = 3/4
class Cifar ready
class CifarLearner 4
class Cifar 1 error rate = 4/5
class Cifar ready
class CifarLearner 1
class Cifar 1 error rate = 4/6
class Cifar ready
class CifarLearner 1
class Cifar 2 error rate = 5/7
class Cifar ready
class CifarLearner 2
class Cifar 7 error rate = 6/8
class Cifar ready
class CifarLearner 7
class Cifar 8 error rate = 7/9
class Cifar ready
class CifarLearner 8
class Cifar 3 error rate = 8/10
class Cifar ready
class CifarLearner 2
class Cifar 6 error rate = 15/20
class Cifar ready
```
class CifarLearner 4
class Cifar 0 error rate = 24/30
class Cifar ready
class CifarLearner 3
class Cifar 3 error rate = 33/40
class Cifar ready
class CifarLearner 2
class Cifar 0 error rate = 41/50
class Cifar ready
class CifarLearner 7
class Cifar 3 error rate = 50/60
class Cifar ready
class CifarLearner 0
class Cifar 8 error rate = 58/70
class Cifar ready
class CifarLearner 3
class Cifar 1 error rate = 68/80
class Cifar ready
class CifarLearner 9
class Cifar 4 error rate = 77/90
class Cifar ready
class CifarLearner 4
class Cifar 1 error rate = 85/100
class Cifar ready
class CifarLearner 8
class Cifar 0 error rate = 171/200
class Cifar ready
class CifarLearner 4
class Cifar 4 error rate = 247/300
class Cifar ready
class CifarLearner 4
class Cifar 4 error rate = 326/400
class Cifar ready
class CifarLearner 7
class Cifar 7 error rate = 404/500
class Cifar ready
class CifarLearner 1
class Cifar 1 error rate = 476/600
class Cifar ready
class CifarLearner 0
class Cifar 1 error rate = 551/700
class Cifar ready
class CifarLearner 9
class Cifar 0 error rate = 629/800
class Cifar ready
class CifarLearner 7
class Cifar 6 error rate = 704/900
class Cifar ready
class CifarLearner 3
class Cifar 5 error rate = 782/1000
class Cifar ready
class CifarLearner 9
class Cifar 0 error rate = 1466/2000
class Cifar ready
class CifarLearner 6
class Cifar 4 error rate = 2128/3000
class Cifar ready
class CifarLearner 4
class Cifar 4 error rate = 2801/4000
class Cifar ready
class CifarLearner 6
class Cifar 6 error rate = 3450/5000
class Cifar ready
class CifarLearner 0
class Cifar 8 error rate = 4111/6000
class Cifar ready
class CifarLearner 7
class Cifar 7 error rate = 4746/7000
class Cifar ready
class CifarLearner 9
class Cifar 3 error rate = 5369/8000
class Cifar ready
class CifarLearner 7
class Cifar 9 error rate = 5996/9000
class Cifar ready
class CifarLearner 2
class Cifar 5 error rate = 6616/10000