University of Michigan
EECS 442: Computer Vision
Fall 2021. Instructor: Andrew Owens.

Problem Set 3: Introduction to Machine Learning

Posted: Wednesday, September 22, 2021 **Due:** Wednesday, September 29, 2021
Submission instructions:

- Canvas:
  - Colab notebook (`.ipynb`) containing solutions and visualizations for 3.1 and 3.2. Before submitting, please make sure to rename the file to `<uniqname>_<umid>.ipynb`.
- Gradescope:
  - `.pdf` version of Colab notebook (conversion instructions here). For your convenience, we have included the PDF conversion script at the end of the notebook.

The starter code can be found at:
`https://colab.research.google.com/drive/1rk3pf4HpTzRE1tAY4HMjFOpyxaOQ4_p9?usp=sharing`

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 3.1 *Nearest Neighbor Classification*

In this problem, we will implement the K-Nearest Neighbor algorithm to recognize objects in tiny images, using the CIFAR-10 dataset (Figure 1). The code for loading and pre-processing the dataset has been provided for you. Your task is to fill in the missing code.

Note: there is a `DEBUG` flag in the starter code that you can set to `True` while you are debugging your code. When the flag is set, only 20% of the training set will be loaded, so the rest of the code should take less time to run. However, before reporting the answers to questions, please remember to set the flag back to `False`, and to rerun the cells!

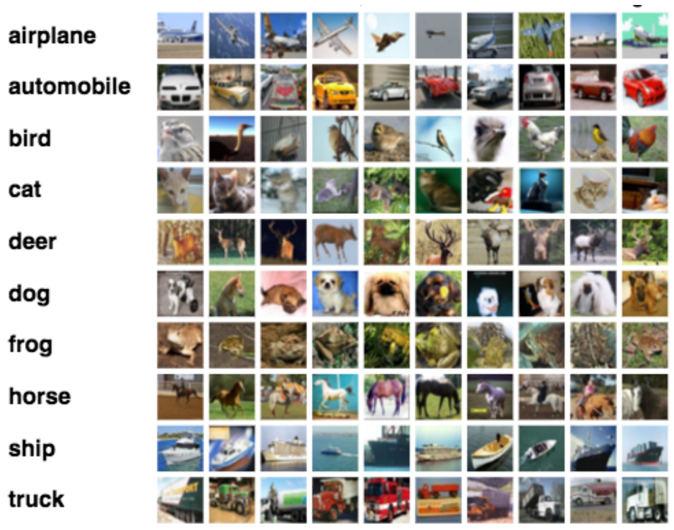

Figure 1: The CIFAR-10 dataset, which we'll be classifying in this problem. [1]

(a) For the class `KNearestNeighbor` defined in the notebook, please finish implementing the following methods:

i. **(1 point)** Please read the header for the method ***compute_distance_two_loops*** and understand its inputs and outputs. Fill the remainder of the method as indicated in the notebook, to compute the l_2 distance between the images in the test set and the images in the training set. The l_2 distance is computed as the square root of the sum of the squared differences between the corresponding pixels of the two images.
Hint: You may use `np.linalg.norm` to compute the l_2 distance.

ii. **(1 point)** It will be important in subsequent problem sets to write fast *vectorized* code: that is, code that has as few `for` loops as possible. As practice, please complete the methods ***compute_distance_one_loops*** which computes the l_2 distance only using a single for loop (and is thus partially vectorized) and ***compute_distance_no_loops*** which computes the L2 distance without using any loops and is thus fully vectorized.
Hint: $||x - y||^2 = ||x||^2 + ||y||^2 - 2x^T y$.

iii. **(1 point)** Similarly, complete the implementation of the method ***predict_labels*** to find the k nearest neighbors for each test image.
Hint: It might be helpful to use the function `np.argsort`.

(b) **(0 points)** Run the subsequent cells, so that we can check your implementation above. You will use `KNearestNeighbor` to predict the labels of test images and calculate the accuracy of these predictions. We have implemented the code for $k = 1$ and $k = 3$. For $k = 1$, you should expect to see approximately 27% test accuracy.

(c) **(1 point)** Find the best value for k using *grid search* on the validation set: for each value of k, calculate the accuracy on the validation set, then choose the highest one. Report the highest accuracy and the associated k in the provided cell below in the notebook. Also, please run the code that we've provided which uses the best k to calculate accuracy on the test set, and to see some visualizations of the nearest neighbors.

(Optional, 0 points) Please run the provided cells below to see the effects of normalization on the accuracy.

(d) **(2 points)** Instead of finding the most similar images based on raw pixels, we obtain better performance using hand-crafted image features. We'll use a simplified version of the Histogram of Oriented Gradients (HOG) features [1] that we discussed in class (Lecture 6). To compute these features, you will:

i. compute the orientations of the gradients by filling in the ***compute_angles*** function ***Hint:*** Please use `np.gradient` to compute the gradients.

ii. create a histogram of edge orientation, with votes weighted by the gradient magnitudes by filling in the ***compute_hog_linear_interp*** function. You will need to linearly interpolate based on angle to determine the weight of magnitude to the two nearby bins

iii. **(0 points)** perform block normalization across the histogram (provided in starter code)

Please read the descriptions in the starter code and fill in the code blocks. Please also run the cells below to test your code. You should expect slightly higher accuracy with HOG than that with raw pixels. Our implementation obtains 1.6% higher accuracy.

(e) ***(Optional, 0 points)*** For reference, we have provided code that computes full HOG features, using a library function. These features should obtain significantly higher accuracy (50.2% in our implementation).

Problem 3.2 *Linear classifier with Multinomial Logistic (Softmax) Loss*

In this problem, we will train a linear classifier using the softmax (multinomial logistic) loss (Equation 2) for image classification (Figure 1), using stochastic gradient descent.

(a) **(3 points) Estimating the loss and gradients**. Complete the implementation of the ***softmax_loss_naive*** function and its gradients using the formulae we have provided, following its specification. Please note that we are calculating the loss on a *minibatch* of N images. The inputs are $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...(\mathbf{x}_N, y_N)$ where $\mathbf{x}_i$ represents the i-th image in the batch, and y_i is its corresponding label.

We first calculate the scores for each object class, i.e. the unnormalized probability that the image is of a particular class. We'll denote the scores ***for a single image*** as $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_C$ where C is the total number of classes, and compute them as, $\mathbf{s} = Wx_i$. The softmax loss ***for a single image***, L_i can be defined as,

$$L_i(\mathbf{s}, y) = -\log \frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}} \tag{1}$$

The total loss $\mathcal{L}$ for all images in the *minibatch* can then be calculated by averaging the losses of all images in the *minibatch*:

$$\mathcal{L}(W) = \frac{1}{N} \sum_{i=1}^{N} L_i. \tag{2}$$

Caution: When you exponentiate large numbers in your softmax layer, the result could be quite large, resulting in values of `inf`. To avoid these numerical issues, you can first subtract the maximum score from each scores as shown below:

$$L_i = -\log \frac{e^{s_{y_i}}/e^{\max(s_j)}}{\sum_{j=1}^{C} e^{s_j}/e^{\max(s_j)}} = -\log \frac{e^{s_{y_i}-\max(s_j)}}{\sum_{j=1}^{C} e^{s_j-\max(s_j)}} \tag{3}$$

Gradients We provide the formulae for the gradients, $\frac{\partial L}{\partial W}$, which will also be returned by ***softmax_loss_naive***:

$$\frac{\partial L_i}{\partial W_{y_i}} = (\frac{e^{s_{y_i}-\max(s_j)}}{\sum_{j=1}^{C} e^{s_j-\max(s_j)}} - 1)x_i \tag{4}$$

$$\frac{\partial L_i}{\partial W_j} = (\frac{e^{s_j-\max(s_j)}}{\sum_{j=1}^{C} e^{s_j-\max(s_j)}})x_i, \quad j \neq y_i \tag{5}$$

As described in the notebook, after implementing this, please run the indicated cells for loss check and gradient check and make sure you get the expected values.

(b) **(3 points)** For the `LinearClassifier` class defined in the notebook, please complete the implementation of the following:

i. **Stochastic gradient descent.** Read the header for the method ***train*** and fill in the portions of the code as indicated, to sample random elements from the training data to form batched inputs and perform parameter update using gradient descent. (Loss and gradient calculation has already been taken care of by us) .

ii. **Running the classifier.** Similarly, write the code to implement ***predict*** method which returns the predicted classes by the linear classifier.

(c) **(0 points)** Please run the rest of the code that we have provided, which uses `LinearClassifier` to train on the training split of the dataset and obtain the accuracies on the training and validation sets. Observe the accuracy on the test set, which should be around 38%. Finally, please refer to the visualizations of the learned classifiers. In these visualizations, we treat the classifier weights as though they were an image, and plot them. You may observe some interesting patterns in the way that each classifier distributes its weight.

Acknowledgement

Part of the homework and the starter code are taken from previous CS231n course at Stanford University by Fei-Fei Li, Justin Johnson and Serena Yeung. Please feel free to similarly re-use our problems while similarly crediting us.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.