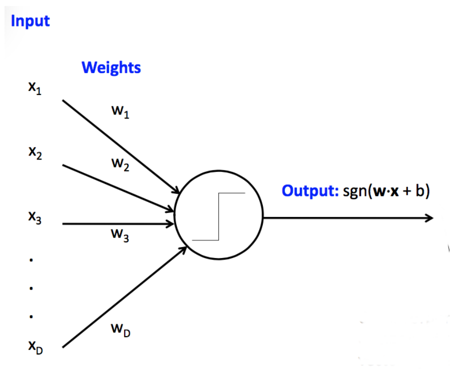

# CS440/ECE448 Spring 2021

# Assignment 5: Perceptron and kNN

**Due date: Wednesday November 3rd, 11:59pm**

In this assignment, we are going to see if we can teach a computer to distinguish living things from non-living things. More precisely, you will implement the perceptron and K-nearest neighbors algorithms to detect whether or not an image contains an animal or not.

## General guidelines and submission

Basic instructions are the same as for previous MPs. To summarize:

- For general instructions, see the main MP page and the course policies.

You should submit on Gradescope:

- A copy of **classify.py** containing all your new code.

## Problem Statement

You are given a dataset consisting of images. Each image contains a picture of an animal or something else. Your task is to implement two algorithms to classify which images have animals in them. Using the training set, you will train a perceptron classifier and K-nearest neighbor classifier that will predict the right class label for an unseen image. Use the development set to test the accuracy of your learned models. We will have a separate (unseen) test set that we will use to run your code after you turn it in. You may use NumPy in this MP to program your solution. Aside from that library, no other outside non-standard libraries can be used.

## The template package

The template package contains everything you need to get started on your MP. Specifically, it contains

- **reader.py** - This file is responsible for reading in the data set. It returns the training images, labels for these images, development data, and labels for the development data.
- **mp5.py** - This is the main file that starts the program, and computes the accuracy, precision, recall, and F1-score using your implementation of the classifers.
- **classify.py** This is the file where you will be doing all of your work.
- **mp5_data.zip** The dataset. When you uncompress this file, you'll find a binary object that our reader code will unpack for you.

To understand more about how to run the MP, run **python3 mp5.py -h** in your terminal.

Add your code to classify.py. **Do not modify the code provided in the other files.**

## The dataset

The dataset consists of 10000 32x32 colored images total. We have split this data set for you into 2500 development examples and 7500 training examples. The images have their RGB values scaled to range

0-1. This is a subset of the CIFAR-10 dataset, provided by Alex Krizhevsky.

The reader will supply the set of test images as one giant numpy array. The development data is in the same format. So your code must read the dimensions from the input numpy array.

In the dataset, each image is 32x32 and has three (RGB) color channels, yielding 32*32*3 = 3072 features. However, be aware that synthetic tests in the autograder may have different dimensions. So do not hardcode the dimensions

## Perceptron Model

The perceptron model is a linear function that tries to separate data into two or more classes. It does this by learning a set of weight coefficients $w_i$ and then adding a bias $b$. Suppose you have features $x_1, \ldots, x_n$ then this can be expressed in the following fashion:

$$f_{w,b}(x) = \sum_{i=1}^{n} w_i x_i + b$$

You will use the perceptron learning algorithm to find good weight parameters $w_i$ and $b$ such that $\text{sign}(f_{w,b}(x)) > 0$ when there is an animal in the image and $\text{sign}(f_{w,b}(x)) \leq 0$ when there is a no animal in the image.

Your function classifyPerceptron() will take as input the training data, training labels, development data, learning rate, and maximum number of iterations. It should return a list of labels for the development data. Do not hardcode values for tuning parameters inside your classifier functions, because the autograder tests pass in specific values for these parameters.

### Training and Development

Please see the textbook and lecture notes for the perceptron algorithm. You will be using a single classical perceptron whose output is either positive or negative (i.e. sign/step activation function).

- **Training:** To train the perceptron you are going to need to implement the perceptron learning algorithm on the training set. Each pixel of the image is a feature in this case. **Be sure to initialize weights and biases to zero.**
  **Note:** To get full points on the autograder, use a constant learning rate (no decay) and do not shuffle the training data.

- **Development:** After you have trained your perceptron classifier, you will have your model decide whether or not each image in the development set contains animals. In order to do this take the sign of the function $f_{w,b}(x)$. If it is negative or zero then classify as $0$. If it is positive then classify as $1$.

Use only the training set to learn the weights.

## Using numpy

For this MP (unlike some previous MPs), it is much easier to write fast code by using numpy operations. Your data is provided as a numpy array. Use numpy operations as much as possible, until right at the end when you choose a label for each image and produce the output list of labels.

NumPy Tips:

- Running computations on arrays tend to be faster when run using NumPy arrays. If you are having issues with timing out, then consider using a NumPy implementation

- Linear algebra operators (dot product, vector norm, matrix multiplication) are immensely simplified when using NumPy. Consider looking up methods to perform some of these operations if they are needed in your solution.
- NumPy Broadcasting may make your life easier for this assignment.

## K-nearest neighbors Model

Perceptron is a simple linear model, and although this is sufficient in a lot of cases, this method has its limits. To see the performance of a different simple classifier, implement the K-Nearest Neighbor algorithm. See the textbook, lecture notes, and/or Wikipedia page for details. Your implementation should use Euclidean distance. **To break ties, use the negative label (no animal class).** You must implement this algorithm on your own with only standard libraries and NumPy.

**Note:** To prevent memory errors due to the autograder's limitations, your algorithm should iterate through the list of test images rather than (say) creating a vector containing all the test images.

Your function classifyKNN() will take as input the training data, training labels, development set, and the number of neighbors used (k). It should return a list of labels for the development data.

Our autograder tests will pass in various values for the parameter k. You may not reset k inside your classifier function. For your own understanding, you should experiment with different values of k to see how this affects the accuracy. Also try to understand how this algorithm compares to a perceptron, in accuracy but also efficiency.