# CS 291K – Advanced Data Mining, Spring 2016
# Machine Problem 1
# (100 points)
# Due *Thursday, April 20 2016, 11:59pm*

---

Notes:

- Make sure to re-read the "Policy on Academic Integrity" on the course syllabus.
- Updates or corrections will be posted on the Announcements page in web page of the course, so check there occasionally.
- You have to work individually for this assignment.
- Each student must turn in their report and code electronically.
- Responsible TA for Machine Problem 1: Honglei Liu honglei@cs.ucsb.edu

---

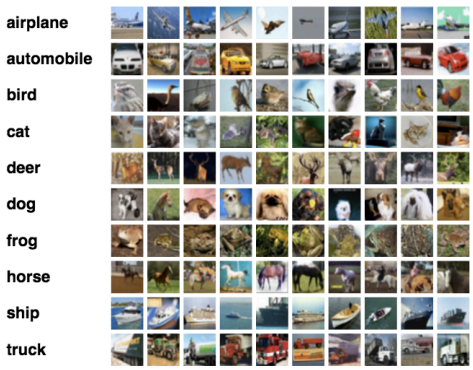
## 1. Problem Definition

You will be given the CIFAR-10 dataset, which consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. You need to implement a classifier, which is a two-layer fully connected neural network with a softmax output layer. You also need to train it properly using the training data and report its accuracy on the test data. The architecture of the neural network is as follows: input - fully connected layer - ReLU - fully connected layer – softmax. In other words, the neural network consists of an input layer, a hidden layer with ReLU activation function and a softmax output layer. The output of this neural network is a 10 dimensional vector representing the predicted probabilities of an image belonging to each class. You can use Python, MATLAB, C/C++ or Java for this assignment.

## 2. What you are given

### 1) The CIFAR-10 dataset

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. Here are the classes in the dataset, as well as 10 random images from each:



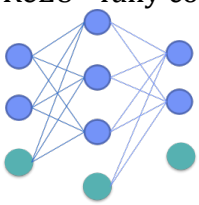
You can check the descriptions of this dataset on https://www.cs.toronto.edu/~kriz/cifar.html. Depending on the programming language you are using, you could also find links to download Python, MATLAB or binary version of the dataset on the website.

**2) A python program to load the dataset**

You will be given a python file named "data_utils.py" within which there is function called "load_CIFAR10" to load all the training and test data from the dataset directory. The input of this function is the dataset directory. The outputs of this function are 4 numpy arrays: Xtr, Ytr, Xte, Yte. Xtr contains the training images, which is a 50000x32x32x3 dimensional array because there are 50000 training images and each image has a size of 32x32 with 3 color channels. Ytr is a 50000x1 array containing the training labels in the range of 0 - 9. Similarly, Xte and Yte contain the test images and test labels. If you are using other programming languages, please follow the instructions on the dataset website to load the data.

**3) The architecture of the neural network**

The architecture of the neural network is as follows: input - fully connected layer - ReLU - fully connected layer – softmax. A sketch of the architecture is as follows.



The input is a 3072x1 vector representing an image. The hidden layer uses ReLU activation function. The output is 10x1 vector from a softmax output layer. During training, you should use the softmax loss function with L2 regularization. The softmax loss function with L2 regularization is defined as follows,

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log p(Y = y_i \mid X = x_i) + \frac{1}{2}\lambda \| W \|_2^2 ,$$

where $N$ is the number of input images in a batch, $y_i$ is the correct label for the $i^{th}$ image, $p(Y = y_i \mid X = x_i)$ is the predicted probability of the $i^{th}$ image belonging to class $y_i$, $W$ is a matrix of all the weight parameters and $\lambda$ represents the regularization strength.

**4) A python file to fill in (for people using python)**

You will be given a python file named "neural_net.py" within which a class named "TwoLayerNet" is already defined for you. The inputs/outputs of the functions to calculate loss, train network and predict labels are also defined. A function called "accuracy" is also given to you to calculate the accuracy of your model on a dataset.

# 3. What you need to do

**1) Implement the neural network from scratch**

- **If you are using Python**:

  You need to implement the neural network from scratch by filling in the "neural_net.py" file. The three functions you need to implement are:

  • **"loss"** to compute the loss and gradients. Please refer to 2.3 for the loss function.
  • **"train"** to train the neural network. It's recommended to implement the training function with mini-batch stochastic gradient decent. You are not required to implement *momentum* or *dropout*, but if you do so, please mention it in the report and there are extra credits!
  • **"predict"** to predict labels for data points. The class label with the largest predicted probability should be the output.

  You may add more functions to the "TwoLayerNet" class, but may not change the names of the functions that are already defined. In some cases, you may add some parameters with default values to the functions that are defined for you. For example, if you want to implement *momentum* or *dropout* for training, probably you may want to add some parameters for the "train" function. You can do so by adding some parameters at the end of the arguments list with default values. **Note that you can only use built-in python libraries and numpy for the implementation.**

- **If you are using other languages**:

  You need to implement the neural network from scratch. Depending on the programming language you are using, you should name the code file as "neural_net.m"/ "neural_net.cpp" / "neural_net.java". There's no requirement on the structure of this file, but you should try to make your implementation easily generalizable to different hyerparameters. You are not allowed to use any third party frameworks or non-native libraries for your implementation. You can use standard and native libraries that are installed on CSIL for every user (the definition of standard or native libraries includes any piece of code that does not require installation or downloading)

**2) Train the neural network and report the accuracies**

You need to train the neural network using the training data and report the accuracies for the training data, validation data and test data in the assignment report. You should report the top-1 accuracy, which only considers the label of an image is correctly predicted if it's the one with the highest probability in the output. **Please do not train your model or tune your hyperparameters on the test data. This will be treated as cheating!** A good practice would be leaving out a small set of training data as validation set or using *k*-fold cross-validation to tune your hyperparameters. We would expect at least an accuracy of **45%** for the validation set.

**3) Make choices about the hyperparameters**

Even though the architecture of the neural network is given, you still need to make decisions about several hyperparameters. For example, you need to decide on the

number of neurons in the hidden layer, the regularization strength, learning rate, number of iterations and batch size. It's ok that you happen to find a set of parameters that works very well, but for the following hyperparameters we would expect some empirical results in the report to support you decisions.

• **number of neurons in the hidden layer**: A figure of number of neurons v.s. accuracy for training and validation set.
• **regularization strength**: A figure of regularization strength v.s. accuracy for training and validation set.
• **learning rate**: A figure of training loss v.s. training steps.

These are just some suggestions. As long as you can justify your decisions, we are good.

**4) Write a program to reproduce the training and testing**

You need to write a program to automatically redo your training process with the best hyperparameters you find and report the accuracies. You should name your source code file as "redo.py" / "redo.m" / "redo.cpp" / "redo.java". This program takes one command-line argument which is the path of the data directory. For example, if the data directory is *"dataset"*, the python version CIFAR-10 data files are located in "*dataset/cifar-10-batches-py"*, the matlab version CIFAR-10 data files are located in "*dataset/cifar-10-batches-mat"* and the binary version CIFAR-10 data files are located in "*dataset/cifar-10-batches-bin".*
Your code should run as

• For Python
```
% python redo.py dataset
```
• For MATLAB
```
% matlab -r "runsort('dataset')" -nodisplay
```
• For C/C++
```
% make
% ./redo dataset
```
• For Java
```
% make
% java –jar redo.jar dataset
```

The program should print the accuracies for the training, validation and testing dataset to the console after some outputs for the training process. For example, the outputs should look like

```
…
Some outputs for the training process
…
Training accuracy: 50%
Validation accuracy: 47%
Testing accuracy: 45%
```

**We would expect that results generated by this program do not differ a lot from the ones in your report.**

**5) Write a report**
As for the report, it should be between 1 and 4 pages in length (no more than 4 pages) with at least 11pt font size and should consist of at least the following sections:
• Implementation: Brief explanations of your code (briefly describe the key points of your implementation)
• Model Building: How you train the neural network and choose the hyperparameters.
• Results: Your results on the provided datasets (training time, accuracy).
• Extra Credits: Things you did to earn extra credits. (optional)
• Challenges: The challenges you faced and how you solved them
• Possible Improvements: Things you could do but haven't done to make your model better.

There should be only one pdf report which includes all the above (no additional readme file).

# 4. Tips

1) Do gradient check with your implementation!
2) Normalize the data by subtracting the mean image. For example, assuming the images are stored in X_train, you could do the following:

```
mean_image = np.mean(X_train, axis=0)
X_train -= mean_image
```

3) When tuning hyperparameters, you don't need to do training for an extremely long time to see the differences. For example, when deciding the learning rate, a few hundred iterations may be already enough.

# 5. Instructions on What and How to Submit

Use the CSIL **turnin** command to submit the necessary files and directories. Note that all directory and file names are case sensitive. For this assignment you need to issue the following command:
**% turnin mp1@cs290k mp1**

For exchange students that do not have a CSIL account:
A. Obtain a *Proof of Registration* to the class, usually with a *Receipt from Extension*.
B. Go to *HFH1140E* with the *Proof of Registration* and someone at the help desk will get the account process started.

Once you have finished developing your code, copy all necessary files (your source code files and the pdf report) into a directory named "mp1". Make sure your code can compile and run on CSIL machines before turning it in.

**If you use C/C++, or Java, please learn how to prepare the necessary Makefile and ensure that your code can compile and execute as follows:**

- For C/C++

```
% make
% ./redo dataset
```

- For Java

```
% make
% java –jar redo.jar dataset
```

Note: Put all the necessary files in a directory named *mp1* and submit the directory as a whole.

# 6. Grading

Grade Breakdown:
- Correctness of the implementation (40 points)
- Completeness of the report (40 points)
- Accuracy on validation set (10 points for greater than 45%)
- Accuracy on test set (10 points for greater than 45%)

# 7. Extra Credits

- Testing accuracy ranked top 1 (20 points)
- Testing accuracy ranked top 5 to top 2 (10 points)
- Implement *momentum* and show its effects on the results (5 points)
- Implement *dropout* and show its effects on the results (5 points)
- Try other initialization method and show its effects on the results (5 points)
- Try other activation functions and show its effects on the results (5 points)

*Plagiarism Warning: We are going to use software to check plagiarism. You are expected to finish the project by yourself without using any code from others.*