CSCI 566: Deep Learning and its Applications - Spring 2022
Take-Home Coding Entrance Exam
Due Date: Jan 14, Friday 11:59am PST
Instructor: Prof. Xiang Ren

This take-home coding exam consists of 7 pages (including this cover page) and 2 problems. These problems are designed to: (1) assess your ability to preprocess data and (2) train a simple machine learning model on your preprocessed data. **Before you start the exam, please carefully read the following instructions.**

- **This is an individual assignment**. Do not collaborate with *anyone*. A plagiarism detection program will be used to check all submissions.
- For this exam, you should use the **Python3** programming language. The "Implementation" section of this document provides instructions for setting up a virtual environment, completing the coding tasks, and submitting the exam.
- Submit your Python code via the following Google Form: https://forms.gle/6drj25mWMMGB5WM99.
- To submit your exam, **upload only one Python file** with the filename `MyUSCID.py`, where `MyUSCID` is your 10-digit USC ID number (*e.g.*, `4916525888.py`).
- You are allowed to make up to **five** submissions prior to the deadline, by clicking "Edit your response" then uploading a new file. **We will only grade your last submission**.
- The only Python libraries you are permitted to use are: `sys`, `pickle`, and `numpy`.

Image Classification with Multiclass Perceptrons

1. **Preprocessing the CIFAR-10 Dataset**

   For this exam, we will use CIFAR-10, a very popular dataset for benchmarking image classification models. CIFAR-10 consists of RGB color images, each represented as a 3 $\times$ 32 $\times$ 32 (*i.e.*, `[num RGB channels]` $\times$ `[image height]` $\times$ `[image length]`) tensor.

   Every image is labeled as one of ten classes: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, *truck*. CIFAR-10 contains 60000 total images, with 6000 images per class. There are 50000 training images and 10000 test images.

   To preprocess the data, please complete the following steps:

   - **Download the data.** Using the following link, download only the Python version of CIFAR-10: https://www.cs.toronto.edu/~kriz/cifar.html.

- **Load the data.** The CIFAR-10 data files are pickled, so you can use the `pickle` library to read these files. You might find the function `pickle.load` helpful to unpickle the file.
- **Preprocess the data.** Each image consists of 8-bit pixel values ranging from 0 to 255. Normalize all pixel values so that they range from 0 to 1. This will help your classification model converge faster. Feel free to experiment with other data preprocessing techniques.

2. **Training a Multiclass Perceptron**

The multiclass perceptron is an algorithm for online multiclass classification. Let $\mathcal{D} = \{(\vec{x}_i, y_i)\}_{i=1}^{N}$ be a classification dataset, where $\vec{x}_i \in \mathbb{R}^d$ is the i-th data point, $y_i \in \{\ell_j\}_{j=0}^{M-1}$ is $\vec{x}_i$'s class label, and N is the number of data points in $\mathcal{D}$. The objective of the multiclass perceptron is to learn weight vectors $W = \{\vec{w}_{\ell_j}\}_{j=0}^{M-1}$ and bias scalars $b = \{b_{\ell_j}\}_{j=0}^{M-1}$, such that $\arg\max_{\ell_j}(\vec{w}_{\ell_j}\vec{x}_i + b_{\ell_j}) = y_i$ for all $(\vec{x}_i, y_i) \in \mathcal{D}$. Together, W and b constitute our *classification model*.

Define the training and test sets as $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, respectively, where $\mathcal{D} = \mathcal{D}_{\text{train}} + \mathcal{D}_{\text{test}}$. Then, we have $N = N_{\text{train}} + N_{\text{test}}$, where $N_{\text{train}} = |\mathcal{D}_{\text{train}}|$ and $N_{\text{test}} = |\mathcal{D}_{\text{test}}|$. For CIFAR-10, we have $N = 60000$, $N_{\text{train}} = 50000$, $N_{\text{test}} = 10000$, and $M = 10$. For time and memory limitations of using physical computers, we will separate $\mathcal{D}_{\text{train}}$ into B distinct batches, with $N_{\text{batch}} = \lceil \frac{|\mathcal{D}_{\text{train}}|}{B} \rceil$. Performing updates with these smaller batches may also help the model avoid some local minima in practice.

In this problem, you need to implement and train a multiclass perceptron to perform image classification on the CIFAR-10 dataset. Here, we obtain $\vec{x}_i$ by flattening the i-th image into a d-dimensional feature vector, where $d = 3 \times 32 \times 32 = 3072$.

The multiclass perceptron algorithm is explained in the pseudocode on the following page.

Algorithm 1 Multiclass Perceptron

Require: training set $\mathcal{D}_{\text{train}} = \{(\vec{x}_i, y_i)\}_{i=1}^{N_{\text{train}}}$, test set $\mathcal{D}_{\text{test}} = \{(\vec{x}_{i'}, y_{i'})\}_{i'=1}^{N_{\text{test}}}$, number of classes M, number of batches B, learning rate η

Initialize weights $W = \{\vec{w}_{\ell_j} = 0\}_{j=0}^{M-1}$ and biases $b = \{b_{\ell_j} = 0\}_{j=0}^{M-1}$

for $batch = 1, ..., B$ **do**

create the batch $\mathcal{D}_{\text{batch}}$ from $\mathcal{D}_{\text{train}}$

for each example $(\vec{x}_i, y_i) \in \mathcal{D}_{\text{batch}}$ **do**

Compute the predicted class probabilities as $\vec{p}_i = \texttt{softmax}(\vec{w}_{\ell_j}\vec{x}_i + b_{\ell_j})$.

end for

Compute the loss (prediction error) as $\mathcal{L} = \frac{1}{N_{\text{batch}}} \sum_{i=1}^{N_{\text{batch}}} \texttt{cross\_entropy}(\vec{y}_i, \vec{p}_i)$.

Compute the gradients of the loss w.r.t. the weights and biases: $\frac{\partial \mathcal{L}}{\partial W}$; $\frac{\partial \mathcal{L}}{\partial b}$.

Minimize the prediction error by updating the weights and biases via gradient descent: $W = W - \eta * \frac{\partial \mathcal{L}}{\partial W}$; $b = b - \eta * \frac{\partial \mathcal{L}}{\partial b}$.

end for

for each example $(\vec{x}_{i'}, y_{i'}) \in \mathcal{D}_{\text{test}}$ **do**

Compute the predicted label as $\hat{y}_{i'} = \arg\max_{\ell_j}(\texttt{softmax}(\vec{w}_{\ell_j}\vec{x}_{i'} + b_{\ell_j}))$.

end for

Compute the test accuracy: $A_{\text{test}} = \frac{1}{N_{\text{test}}} \sum_{i'=1}^{N_{\text{test}}} \texttt{accuracy}(y_{i'}, \hat{y}_{i'})$

As outlined in the pseudocode, you should train your model using only the training data, then evaluate your model's accuracy on the testing data. Your script should print the model's test accuracy at the end.

Note: In this problem, your multiclass perceptron is expected to yield a test accuracy of around 30%. Meanwhile, recent advances in deep learning have pushed the state-of-the-art CIFAR-10 test accuracy to 99.5%! Hopefully, this serves as compelling motivation for taking this deep learning course.

Implementation

Setup

You can set up a virtual environment by using `virtualenv`.

```
$ sudo pip install virtualenv # If you didn't install it
$ virtualenv -p python3 .env
$ source .env/bin/activate
# [Work on your coding exam...]
$ deactivate
```

You can install required Python libraries by running the following command:

```
$ pip install pickle numpy
```

Code

- We provide you with a skeleton code file called `exam.py`, which contains a model

class and various functions. Please do **NOT** modify the `main()` function to accept arguments or any function definitions. You should implement your solution only by filling in the `TODO` sections of the skeleton code. We describe some key parts of the skeleton code below, but more detailed instructions for `TODO` are included in the skeleton code. Please make sure you fill out all the `TODO` sections in the skeleton code.

- The `forward()` function returns the softmax probabilities for each image in the batch. The softmax of a M-dimensional vector $\vec{q} = [q_j]_{j=0}^{M-1}$ is defined as:

$$\texttt{softmax}(\vec{q})_j = \frac{e^{q_j}}{\sum_{j=0}^{M-1} e^{q_j}}$$

  This transforms $\vec{q}$ into a distribution $\vec{p}$, where the elements of $\vec{p}$ sum to 1.

- The `loss()` function implements the cross-entropy loss, which is defined as:

$$\texttt{cross\_entropy}(\vec{y}, \vec{p}) = -\sum_{j=0}^{M-1} \vec{y}_j \log(\vec{p}_j)$$

  Here, $\vec{y}$ is the true label distribution and $\vec{p}$ is the model's predicted label distribution. For classification with CIFAR-10, we often represent $\vec{y}$ as a one-hot vector, which is a binary vector containing all zeros except for a one at the index of the true class label. For example, if an image in the CIFAR-10 dataset has class label $y = 3$ (*bird*), then the one-hot encoding of y would be $\vec{y} = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]^T$.

- The `compute_gradients()` function returns the gradients. The `gradient_descent()` function uses the gradients and learning rate to update the model. Do not negate the weights and biases or update the model in the `compute_gradients()` function. For each batch, you should compute the average gradient across the batch, then update your model parameters using this batch-average gradient.

- The `accuracy()` function uses the accuracy metric to evaluate the model's predicted labels with respect to the true labels.

Training Details

- As a starting point, we recommend using a learning rate of $\eta = 0.005$ and a batch size of 100.

- The model contains a total of 30730 trainable parameters: 30720 weight terms and 10 bias terms. All parameter values should be initialized as 0.

Hyperparameter Tuning

- You can try to improve your model's accuracy by tuning the model's hyperparameters (*e.g.*, learning rate, batch size). However, it is not appropriate to access the test set until the final model evaluation. Instead, the common practice is to hold out part of the training set to use as a validation set for hyperparameter tuning. By tuning only on the validation set, we can estimate the model's generalization ability to the testing set (w.r.t. different hyperparameter configurations) without accessing the test data. The pseudocode below shows one possible high-level procedure for tuning the hyperparameters:

Algorithm 2 Hyperparameter Tuning

Require: training set $\mathcal{D}_{\text{train}} = \{(\vec{x}_i, y_i)\}_{i=1}^{N_{\text{train}}}$, number of classes M,
  for *several iterations* **do**
    Partition $\mathcal{D}_{\text{train}}$ into distinct sets $\mathcal{D}_{\text{train'}} + \mathcal{D}_{\text{val}}$
    Initialize hyperparameters: batch size (B) and learning rate (η).
    Perform Algorithm 1 ($\mathcal{D}_{\text{train'}}$, $\mathcal{D}_{\text{val}}$, M, B, η) to find validation set accuracy for (B, η).
  end for
  return hyperparameters B, η that provided the highest validation set accuracy.

- You may update your models to use the hyperparameters that you found in your hyperparameter search. Remember that your submission should use the original training and testing sets in its `main()` function. Therefore, you are encouraged to make a separate function to perform a hyperparameter search.

Output

- After training for one epoch (*i.e.*, pass over all training examples)
- Please output the test accuracy as a percentage (*i.e.*, between 0 and 1), in the `.4f` format (*e.g.*, `.3032`). If the accuracy value is stored in a variable called `acc`, then you can print the accuracy by doing `print(f"{acc:.4f}")`.
- Before you submit your script, remember to change your filename from `exam.py` to `YourUSCID.py`.
- Make sure your script does not download the dataset during execution.

Evaluation

We will grade your coding exam using automated tests, based on the following criteria:

- Your script's **functionality** (*e.g.*, whether the script is runnable, functions working as expected).

- Your script's **adherence to the exam instructions** (*e.g.*, correctly formatted outputs, correct filename).
- Your script's **runtime**. If your script your takes more than five minutes to iterate over the training set, then it will not be graded.
- Your model's **test accuracy**. We recommend that you monitor how the accuracy changes across batches, since it should increase over time.

THE EXAM ENDS HERE.

Frequently Asked Questions

- **Can we use `scikit-learn` or other non-standard Python libraries?**

  You are **NOT** allowed to use `scikit-learn` or any other non-standard Python library, except `numpy`. Also, you are allowed to use standard libraries (*e.g.*, `sys`, `pickle`). We want you to implement your own multiclass perceptron from scratch.

- **What Python version should we use?**

  We will evaluate your code using a Python 3.8 virtual environment, so you need to make sure your code works in such an environment.

- **What files should `inputs_file_path` contain?**

  You should assume that `inputs_file_path` contains unzipped files in `pickle` format (*not* `.gz` or `.zip` format). Our testing folder will have the following six files, which you can assume you are directly accessible by name:

  - `data_batch_1`
  - `data_batch_2`
  - `data_batch_3`
  - `data_batch_4`
  - `data_batch_5`
  - `test_batch`

- **Are we allowed to use any external libraries for loading CIFAR-10 files?**

  No, loading data from the downloaded files should be easy enough without using any external libraries. The assignment instructions and https://www.cs.toronto.edu/~kriz/cifar.html provide everything you need.

- **My loss sometimes increases. Is this normal?**

  Yes, your loss may sometimes increase, but you should expect it to generally decrease. If your loss gets very large (*i.e.*, above an average of 10 per example, you may need to check your implementation). One debugging strategy is to check for convergence by repeatedly training on a small dataset of 10 examples.

- **My script takes a very long time to run! What is happening?**

  In general, you should vectorize your code as much as possible. That is, try to convert any for loops into tensor multiplications. On a standard laptop, your implementation should take less than 30 seconds to iterate over the training set once.

- **The filename of my Google Form submission was automatically changed. Is this expected?**

  Yes, if you submit your code with the filename `YourUSCID.py`, then your name will be automatically appended to your filename.