ECE49595CV: Undergraduate Introduction to Computer Vision
ECE59500CV: Graduate Introduction to Computer Vision

Fall 2025

Homework 1

Due 5:00pm ET Friday 26 September 2025

A fully-connected (FC) linear layer is:

$$\texttt{linear}(\mathbf{x}; W, \mathbf{b}) = W\mathbf{x} + \mathbf{b}$$

where W is a weight matrix and $\mathbf{b}$ is a bias vector.

The sigmoid function is:

$$\frac{1}{e^{-x}+1}$$

A `sigmoid` layer applies this elementwise to a vector.

A single layer perceptron (SLP) is:

$$\texttt{slp}(\mathbf{x}; W, \mathbf{b}) = \texttt{sigmoid}(\texttt{linear}(\mathbf{x}; W, \mathbf{b}))$$

A multilayer perceptron (MLP) is:

$$\texttt{mlp}(\mathbf{x}; W_1, \ldots, W_n, \mathbf{b}_1, \ldots, \mathbf{b}_n) = \texttt{slp}(\ldots\texttt{slp}(\mathbf{x}; W_1, \mathbf{b}_1)\ldots; W_n, \mathbf{b}_n)$$

We say that n is the number of layers and $n-1$ is the number of hidden layers. The number of columns of W_1 is the input dimension, the number of rows of W_n is the output dimension, and the number of rows of each of $W_1, \ldots, W_{n-1}$ is the number of hidden units in each hidden layer. The number of layers and the number of hidden units in each hidden layer are called the hyperparameters.

The L2 (Euclidean) distance (loss) between two vectors $\mathbf{u}$ and $\mathbf{v}$ is:

$$\texttt{loss}(\mathbf{u}, \mathbf{v}) = (\mathbf{u} - \mathbf{v}) \cdot (\mathbf{u} - \mathbf{v})$$

Naive gradient descent finds a minimum of a function $f(\mathbf{x})$ by starting with an initial guess $\mathbf{x}_0$ and iterating:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla f(\mathbf{x}_i)$$

where η is called the learning rate.

Given a training set of input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$, each paired with output vectors $\mathbf{y}_1, \ldots, \mathbf{y}_m$, one can train a multilayer perceptron with naive gradient descent to minimize

$$\sum_{i=1}^{m} \texttt{loss}(\mathbf{y}_i, \texttt{mlp}(\mathbf{x}_i, W_1, \ldots, W_n, \mathbf{b}_1, \ldots, \mathbf{b}_n))$$

For this homework, I would like you to implement a mechanism to train multilayer perceptrons. You can implement this in any programming language you wish, and develop this on any computer you wish, so long as the course staff can run it on RCAC Scholar as described in the syllabus.

This homework assignment should be submitted as a single ZIP file whose name is your Purdue Career account userid and whose extension is `.zip` that the course staff can run on RCAC Scholar. You should create a directory whose name is your Purdue Career account name, then put everything you wish to submit in that directory, and then package that as a ZIP file. That directory should have at least a single file named `run` that is an executable `bash` script that runs your homework. In my case, my Purdue Career account name is `qobi`. So I would do something like this:

```
qobi@sapiencia>cd /tmp/
qobi@sapiencia>mkdir qobi
qobi@sapiencia>cd qobi
qobi@sapiencia>chmod a+x run
qobi@sapiencia>pwd
/tmp/qobi
qobi@sapiencia>ls -l
total 4
-rwxr-xr-x 1 qobi qobi 31 Aug 19 12:55 run*
qobi@sapiencia>cat run
#!/bin/bash
echo my submission
qobi@sapiencia>./run
my submission
qobi@sapiencia>cd ..
qobi@sapiencia>zip -r qobi qobi
  adding: qobi/ (stored 0%)
  adding: qobi/run (stored 0%)
qobi@sapiencia>
```

Obviously, there will be more files and the `run` script will be different for a real homework submission. Homeworks will be submitted through Brightspace. What the course staff will do to grade your homework is something like this:

```
qobi@sapiencia>ls -l qobi.zip
-rw------- 1 qobi qobi 335 Aug 19 12:56 qobi.zip
qobi@sapiencia>unzip qobi.zip
Archive:  qobi.zip
   creating: qobi/
 extracting: qobi/run
qobi@sapiencia>cd qobi
qobi@sapiencia>./run
my submission
qobi@sapiencia>
```

one at a time for each submission. This should run in less than five minutes on RCAC Scholar.

You are free to prepare your homework on any machine that you wish. But the course staff must be able to run it in less that five minutes RCAC Scholar.

I have requested that the university provide accounts for all enrolled students on RCAC Scholar. However, I do not know anything about RCAC Scholar and am unable to provide any support.

I don't want you to use GPUs for this homework. Your code should run on just CPUs. You don't have to parallelize it. Speed is not important.

I don't want you to use any preexisting libraries or tools. No deep learning frameworks like PyTorch or TensorFlow. No automatic differentiation (AD) tools like jax. Your code should be written in straight c, c++, Java, Python, or whatever, with nothing other than the most basic standard library included in each particular language.

Your code should allow you to control the number of layers n, the input and output dimensions, the number of hidden units in each layer, and the learning rate. You can use whatever method you wish to randomly initialize naive gradient descent and select how many iterations to run.

I wish you to create two datasets. One will be for the xor function that has 2 inputs and 1 output. The other will be for a two-bit binary adder that has five inputs (a_0, b_0, c_0, a_1, and b_1) and three output (s_0, s_1, and c_2). For each function, I wish you to partition the dataset into a variety of training and test sets (splits), train a multilayer perceptron of a variety of hyperparameters and learning rates on each, and evaluate the loss on the test set for each. You get to choose and explore the splits, the hyperparameters, the learning rates, and the numbers of iterations. Your `run` script should demonstrate the range of splits, hyperparameters, learning rates, and numbers of iterations you tried. Your `run` script can output whatever you wish. The only stipulation is that the output should be easily human readable and should quickly convince both you and the course staff that your code works.

You can implement this with either forward mode or reverse mode automatic differentiation (AD) as taught in class. But you do not need to make an AD package that can apply to arbitrary code. You only need to implement a sufficient portion to support training multilayer perceptrons. That can be hard-coded for that purpose.

Good luck and have fun!