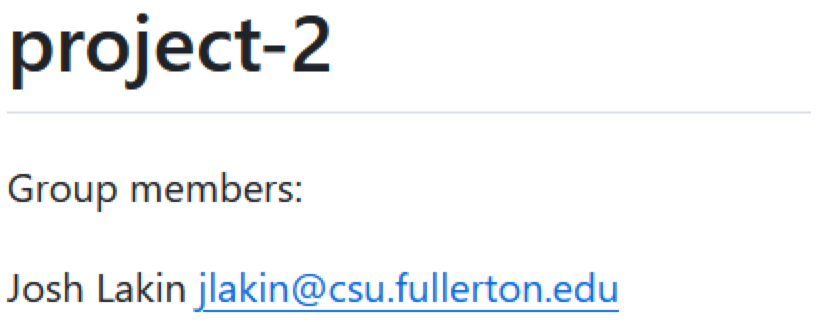
The goal of this project was to learn how to do backpropagation and inference on a multilayer perceptron using CUDA kernels. I chose to do regression on images, which didn't work out that well since regression is a much harder task than classification for multilayer perceptrons. I made a CPU version of stochastic gradient descent and a GPU version. The CPU version is commented out in main.cu. The difference between the two is the CPU computes the gradient over the whole image, while the GPU version computes the gradient over blocks of pixels in the image (batches). To calculate a gradient, you need to sum over all the gradients calculated from each input separately. Doing a parallel sum requires synchronization, and in CUDA you can only have synchronization between threads in the same block, so you are limited to summing up arrays of the length of the max block size (1024). You can split the array into chunks of length 1024 and assign each chunk to a block, but memory becomes an issue. Calculating gradients over blocks of pixels already consumes a lot of GPU memory due to the need for scratch memory to store intermediate calculations. This is probably why in most deep learning libraries you typically do backpropagation with fixed batch sizes.

The code outputs a pgm file called out.pgm, it is very small (main.cu loads feep.pgm) so you have to magnify it if you want to see it. Even so, it's a pretty bad reconstruction since regression is very difficult for small multilayer perceptrons. There is an architecture that allows for better regression for multilayer perceptrons, but this is out of the scope for this class.

One thing to note is that if the batch size is greater than the size of the image, learning is a lot easier, but if the batch size is smaller then learning becomes really difficult.

1. Project 2 Submission, Josh Lakin

2.



project-2

Group members:

Josh Lakin jlakin@csu.fullerton.edu

3.

A multilayer perceptron (MLP) is a simple example of a neural network consisting of a chain of matrix multiplications followed by a nonlinear activation function. Each layer computes f(Wx + b), where W is a learnable weight matrix, b is a learnable bias vector, x is an input vector from the previous layer, and f is a function that is independently applied to each element of a vector. Each neuron represents a scalar output, and a group of neurons forms a single layer represented by a vector output.

```
To reconstruct an image:
For all pixel i,j in width height:
        pixel(i , j) <- invoke the MLP at x = i, y = j
```

Gradient descent is an iterative algorithm for finding the minimum of a function. In this case this function is a loss function that computes the squared difference between pixel approximations from the MLP and actual pixel values. This is a function defined over the weights: E(w1,w2,...,wn) where wi are the weights of the MLP. To update we do Wi+1 = Wi - gradient(E, Wi), the gradient of E with respect to Wi.

I calculated the gradients analytically:

$$\frac{\partial E}{\partial \overrightarrow{a^l}} = W^{{l+1}^\intercal} \frac{\partial E}{\partial \overrightarrow{b^{l+1}}}$$

$$\frac{\partial E}{\partial \overrightarrow{b^l}} = \frac{\partial E}{\partial \overrightarrow{a^l}} \odot f'^l\left(\overrightarrow{z^l}\right)$$

$$\frac{\partial E}{\partial W^l} = \frac{\partial E}{\partial \overrightarrow{b^l}} \otimes \overrightarrow{a^{l-1}}$$

z = Wx, a = f(z), W is the weight matrix, b is the bias vector

4. Use “make build” and “./a.out” or run in Visual Studio on release mode

5.

feep.pgm (this is a very small image, so the batch size is bigger than the image):

```
Epoch 0, Batch number 0, Loss: 61.666092
Epoch 1, Batch number 0, Loss: 61.389107
Epoch 2, Batch number 0, Loss: 60.906616
Epoch 3, Batch number 0, Loss: 59.911720
Epoch 4, Batch number 0, Loss: 57.194901
Epoch 5, Batch number 0, Loss: 45.180653
Epoch 6, Batch number 0, Loss: 3.552812
Epoch 7, Batch number 0, Loss: 1.146199
Epoch 8, Batch number 0, Loss: 0.740070
Epoch 9, Batch number 0, Loss: 0.549335
Epoch 10, Batch number 0, Loss: 0.437395
Epoch 11, Batch number 0, Loss: 0.363615
Epoch 12, Batch number 0, Loss: 0.311328
Epoch 13, Batch number 0, Loss: 0.272357
Epoch 14, Batch number 0, Loss: 0.242221
Epoch 15, Batch number 0, Loss: 0.218234
Epoch 16, Batch number 0, Loss: 0.198709
Epoch 17, Batch number 0, Loss: 0.182520
Epoch 18, Batch number 0, Loss: 0.168888
Epoch 19, Batch number 0, Loss: 0.157256
Epoch 20, Batch number 0, Loss: 0.147228
Epoch 21, Batch number 0, Loss: 0.138492
Epoch 22, Batch number 0, Loss: 0.130820
Epoch 23, Batch number 0, Loss: 0.124032
Epoch 24, Batch number 0, Loss: 0.117986
Epoch 25, Batch number 0, Loss: 0.112570
Epoch 26, Batch number 0, Loss: 0.107690
Epoch 27, Batch number 0, Loss: 0.103274
Epoch 28, Batch number 0, Loss: 0.099260
Epoch 29, Batch number 0, Loss: 0.095596
Epoch 30, Batch number 0, Loss: 0.092240
…
```

pepper.pgm (this is an image that’s larger than the batch size, the output is truncated for one epoch):

```
Epoch 0, Batch number 0, Loss: 6.792533
Epoch 0, Batch number 1, Loss: 26.193291
Epoch 0, Batch number 2, Loss: 21.834482
Epoch 0, Batch number 3, Loss: 21.064716
Epoch 0, Batch number 4, Loss: 21.700581
Epoch 0, Batch number 5, Loss: 21.673124
Epoch 0, Batch number 6, Loss: 20.850222
Epoch 0, Batch number 7, Loss: 20.264820
Epoch 0, Batch number 8, Loss: 18.639317
Epoch 0, Batch number 9, Loss: 15.351122
Epoch 0, Batch number 10, Loss: 12.759095
Epoch 0, Batch number 11, Loss: 10.984519
Epoch 0, Batch number 12, Loss: 10.559717
Epoch 0, Batch number 13, Loss: 10.829639
Epoch 0, Batch number 14, Loss: 11.966697
Epoch 0, Batch number 15, Loss: 13.668625
Epoch 0, Batch number 16, Loss: 14.752352
Epoch 0, Batch number 17, Loss: 15.605523
Epoch 0, Batch number 18, Loss: 16.088758
Epoch 0, Batch number 19, Loss: 16.242731
Epoch 0, Batch number 20, Loss: 16.593660
Epoch 0, Batch number 21, Loss: 17.110924
Epoch 0, Batch number 22, Loss: 17.442589
Epoch 0, Batch number 23, Loss: 17.934662
Epoch 0, Batch number 24, Loss: 18.047537
Epoch 0, Batch number 25, Loss: 18.610237
Epoch 0, Batch number 26, Loss: 18.696615
Epoch 0, Batch number 27, Loss: 18.275129
Epoch 0, Batch number 28, Loss: 17.620693
Epoch 0, Batch number 29, Loss: 16.661360
Epoch 0, Batch number 30, Loss: 15.649189
Epoch 0, Batch number 31, Loss: 14.683064
Epoch 0, Batch number 32, Loss: 14.352322
Epoch 0, Batch number 33, Loss: 14.330112
Epoch 0, Batch number 34, Loss: 14.152554
Epoch 0, Batch number 35, Loss: 13.630630
Epoch 0, Batch number 36, Loss: 13.626005
Epoch 0, Batch number 37, Loss: 14.247101
Epoch 0, Batch number 38, Loss: 14.292183
Epoch 0, Batch number 39, Loss: 13.855833
Epoch 0, Batch number 40, Loss: 13.751808
Epoch 0, Batch number 41, Loss: 13.514337
Epoch 0, Batch number 42, Loss: 13.349724
Epoch 0, Batch number 43, Loss: 13.146861
…
```