Neural Network planning document

Tensor operations

We will be using Eigen for all operations in tensors such as multiplication, reshape and more.

Project structure

Layer class

Superclass containing all methods and attributes for a neural network layer.

Methods:

- updateWeights
- forward
- backward

Inheriting layers:

Input

Layer containing all logic for the first layer of a model

DenseLayer

Operations done in a dense neural network layer

Sigmoid

Sigmoid activation funcions

Linear

Linear activation function

Model class

Methods:

- forward
- backward
- add_layer
- predict
- train (might be moved out to a separate trainer class later)

Data structure

The code and efficiency

Object-oriented, with the use of the eigen3 library for matrix-math operations. Eigen3 is a template library consisting purely of headers. Performance-wise eigen3 claim to be quite fast eigen3 own benchmarks against other c++ matrix libraries.

Other libraries:

- **OpenMP**: seems included both in GNU and Clang OpenMP.

- **OpenMPI**: Might have to download OpenMPI depending if we find performance gains OpenMPI

> **Note**: * Not likely since it seems both pyTorch and Tensorflow have gone for OpenMP for distributing work on the CPU)*

Memory will be handled safely as we will reuse the parameters underways for training.

- Initialization should be sufficient
- Rewrite auxiliary structure and weights underways

Inefficient memory

- training data will be loaded and constant right before training
- otherwise, it will be referenced

Data structures

We are just gonna use our own defined classes **(mentioned above)**, matrices, and arrays. They fit our use case.

API example

The API is inspired by the functional API found in Keras. We also considered mimicking the API found in PyTorch, but we believe the Keras API is easier to implement and we are more familiar with it.

```
DenseNeuralNetwork model = Neural Network();
model.add_layer(Input(256));
model.add_layer(DenseLayer(128));
model.add_layer(Sigmoid());
model.add_layer(DenseLayer(64));
model.add_layer(Sigmoid());

model.train(...);

predictions = model.predict(...);
```

Concurrency

We plan to implement concurrency by splitting the forward and backward passes in the train loop across processes. For example, with a batch size of 4 and 4 processes, the network will process 4*4=16 datapoints at the same time. After the forward pass, the different processes will sync up and reduce to a loss shared by all processes. The loss will then be backpropagated with the same 4*4 data points simultaneously.

We are not sure how this will affect the runtime of our software due to the overhead from the concurrency operations, but implementing it will still be a good learning experience. There are also other ways of parallelizing neural network training, but we believe this is the most realistic way to implement it due to time and complexity constraints.

Some other methods include:

- Splitting batches into tasks by sample and distributing them evenly among the workers, sync at the end of a forward pass, and the same for the backward pass.
- A hybrid between the first method and the method above.

There is also a possibility to make eigen3 operations multi-threaded
https://eigen.tuxfamily.org/dox/TopicMultiThreading.html

Time complexities

Time complexities are hard to determine as they will depend on the sizes of input, number of hidden layers, hidden layer sizes, and number of outputs.

In simple case scenario, should be around `O(N*M*L)`

N = number of nodes in a layer

M = Sigmoid/relu/.. complexity of activation function

L = number of layers

Complications arise as we change/add a variety of activation functions, layer sizes, shape transformations, filter functions, neuron dropping, etc.

However, we can estimate a worst-case scenario where the forward pass has a worst-case O(n^3) where n = max(sizes of input, number of hidden layers, hidden layer sizes, and number of outputs). This is due to the time complexity of the standard matrix multiplication algorithm being O(n^3) (There might be a slightly more efficient algorithm being used in eigen3, not sure about this)