Requirements Document

Stakeholders

Software developers using the machine learning library to build and train models
DevOps engineers responsible for deploying trained models
End-users of the models built and trained using the library
The software developer (me) developing the library
Regulatory bodies concerned with the fair and ethical use of AI (e.g., the EU Artificial Intelligence Board)

Functional Requirements

Dense layer must randomly initialize weights and biases upon initialization, according to user-specified dimensions *[unit]*
Dense layer must compute the function $Wx + b$ during forward propagation *[unit]*
Dense layer must update weights according to the function $W^{(t+1)} = W^{(t)} - \alpha \cdot \nabla L\left(W^{(t)}\right)$ during backpropagation *[unit]*

Dropout layer must disable some neurons with probability $p(1.0 - \text{dropout\_rate})$ during training *[unit]*
Dropout layer must multiply the derivatives with the dropout mask during backpropagation
Dropout layer must be bypassed during inference *[unit]*

ReLU layer must compute the function $f(x) = \max(0, x)$ during forward propagation *[unit]*
ReLU layer must compute the function $f'(\max(0, x)) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$ *[unit]*

Softmax layer must compute the function $\frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ during forward propagation *[unit]*
Softmax layer must compute the function $\hat{y}_i - y_i$ (where $\hat{y}_i$ are the predicted probabilities and y_i are the true labels) during backpropagation *[unit]*
Softmax outputs should sum to 1 *[unit]*

Sigmoid layer must compute the function $\frac{1}{1+e^{-x}}$ during forward propagation *[unit]*
Sigmoid layer must compute the function $\sigma(x)(1 - \sigma(x))$ during backward propagation *[unit]*

Linear layer must compute the identity function during forward and backward propagation *[unit]*

Given predictions and ground truth values, the accuracy functions should compute the mean accuracy *[integration]*
Regression accuracy should compute the function $|\hat{y}_i - y_i| < p$ where p is a user-specified precision threshold *[unit]*
Categorical accuracy should compute the function $\begin{cases} 1 & \text{if } \hat{y}_i = y_i \\ 0 & \text{if } \hat{y}_i \neq y_i \end{cases}$ *[unit]*

Categorical Cross Entropy loss should compute $-\log\left(\frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}\right)$ *[unit]*

Binary Cross Entropy loss should compute $-\sum_{i=1}^{C'=2} t_i \log\left(f(x_i)\right)$ *[unit]*

Mean Squared Error loss should compute $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ *[unit]*
Mean Absolute Error loss should compute $\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$ *[unit]*

Stochastic gradient descent should perform basic gradient updates during backpropagation *[unit + integration] (these optimiser requirements are at both the unit and integration level because unit tests are required to check that the mathematical implementation is correct, but they must also be supported by integration tests to ensure that data is correctly being handed between layers during gradient descent)*
Adagrad optimizer should perform gradient updates scaled by an adaptive learning rate based on a historical sum of gradients *[unit + integration]*
RMSprop optimizer should perform as Adagrad, BUT prevent step size over time from decaying to zero *[unit + integration]*
Adam should perform as RMSprop but with the addition of bias correction *[unit + integration]*

Users should be able to add layers to instances of the model class *[integration]*
Users should be able to configure an optimizer, accuracy metric, and loss function for the model *[integration]*
Calling the model's train method should perform forward, then backwards propagation for a user specified number of epochs, split over a user specified number of batches *[integration]*
At the end of each timestep, the model should output its accuracy, loss, regularization loss, learning rate, and the time it took to execute the timestep *[unit]*
The model's predict method must take as input a feature vector and output the model's prediction as a probability distribution vector *[unit]*

Every layer must be templated so that it can be instantiated with any decimal type (i.e., float, double, float16, etc.) *[system]*

The library code must not be vulnerable to buffer overflow vulnerabilities *[system]*

Measurable Attributes

Performance: the library must match or exceed the performance of equivalent CPU-only machine learning libraries *[system]*
Memory usage: the library must be able to train with large datasets and large amounts of parameters on a consumer-grade system without memory overflow *[system]*
Accuracy: the library must achieve high precision for common classification tasks such as MNIST or CIFAR-10 *[system]*
Convergence speed: the library's optimizers must converge to optimal solutions within a reasonable number of epochs *[system]*
Cross-platform support: the library must compile with popular C++ compilers (GCC, MSVC, Clang) on common operating systems (Windows, Linux, macOS) *[system]*
Reproducibility: given the same random seed, trained models should have the same weights and produce the same outputs on successive training runs *[system]*

Qualitative Attributes

The API must be easy and intuitive to use *[system]*
The library should have good quality documentation *[system]*

The library should have a modular and extensible design so that users can add their own custom extensions and functionalities *[system]*
The library code must be maintainable so that new developers can easily be brought on board *[system]*