The purpose of this library is to implement the matrix multiplication technique for calculating derivatives, to determine if there is a speed benefit associated with this technique over back propagation. A copy of the “Expressions of Derivatives” document has been pulled from the Philosophy folder (and updated to v2 then v3); this is the electronic record of the hand-work deriving the matrix publication technique for calculating derivatives.

The purpose of this document is to define the requirements for this library.

Using {} instead of [] for my own notes, as using [] for requirement numbers.

ConOps: A user shall be able to build, train and use a PINN.

{Decide on the appropriate level of systems engineering rigor; should we move this to an Excel workbook to enable additional notes, two-way traceability of parent/child relationships, validation methods and status, etc.?}

Requirements: {Continue decomposition}

[01-01] The library shall enable a user to define an NN.

[01-02] The library shall enable a user to define multiple NNs.

[01-03] The library shall enable the calculation of derivatives of an NN with respect to the model inputs using the matrix multiplication technique.

[01-04] The library shall enable a user to define multiple loss functions.

[01-05] The library shall enable the calculation of derivatives of the loss functions with respect to the model inputs and model parameters using the matrix multiplication technique.

[01-06] The library shall enable training of the model parameters.

[01-07] The library shall enable the saving of the NN.

[01-08] The library shall enable the loading of an NN.

[01-09] The library shall enable inference using an NN.

[02-01] The library shall define a Model Class.

[02-02] The Model Class shall be initialized by a call from user code.

[02-03] The Model Class shall be provide a method to add a fully dense layer of neurons to the end of the model.

[02-04] The Model Class shall have a data structure, W, to store model weights.

[02-05] The Model Class shall have data structures x, y, z, Q, P, V, A, B, C, D, E and F as defined by {the equations from “Expressions of Derivatives v3.docx”}.

[02-06] The Model Class shall provide a method for inference.

[02-07] The Model Class shall provide a method to calculate first derivatives of model outputs with respect to model inputs.

[02-08] The Model Class shall provide a method to calculate second derivatives of model outputs with respect to model inputs.

[02-09] The Model Class shall provide a method to calculate the gradient of the inferred values with respect to model weights.

[02-10] The Model Class shall provide a method to calculate the gradient of the first derivatives of model outputs with respect to model inputs, with respect to model weights, for each model input by which the model output first derivative is called.

[02-11] The Model Class shall provide a method to calculate the gradient of the second derivatives of model outputs with respect to model inputs, with respect to model weights, for each model input by which the model output second derivative is called.

[02-12] The Model Class initialization method shall enable the user to specify an optimization method to be used for model training.

[02-13] The Model Class shall provide a train method to adjust the model weights according to the optimization method defined for the model.

[02-14] The Model Class train method shall enable at least one optimization method that utilizes the first-order gradient with respect to the model weights.

Note: There is no requirement to enable optimization methods that utilize second-order gradients (Hessians).

[02-15] All methods shall operate on lists of inputs.

[02-16] All data structures shall assume model inputs are provided as lists.

[02-17] The Model Class shall provide a method to save a model and any optimizer state information.

[02-18] The Model Class shall provide a method to load a model and any optimizer state information.

[03-01] The Model Class initialization method shell enable the user to specify the number of input variables to the model.

[03-02] The Model Class add\_layer method shall enable the user to specify the number of nodes in the added layer.

{Continue}

[03-06] The Model Class inference method shall calculate x and y for all nodes in all layers.

[03-07] The Model Class first derivative method shall ensure that z is calculated.

[03-08] The Model Class first derivative method shall ensure that Q and P are calculated for each model input for which the first derivative method is called.

[03-09] The Model Class second derivative method shall ensure that z is calculated.

[03-10] The Model Class second derivative method shall ensure that Q, P are calculated

[04-01] The 0th index of all layer-indexed model data structures shall correspond to the model input layer.

Rewrite each of these as requirements!

Things to have or calculate at each layer: W, x, y, z, Q, P, V, A, B, C, D, E, F

Will probably put all of those data at the model level. The issue with putting it at the model level is that the size of each layer may be different, and the activation functions applied (and thus the equations for q+) at each layer may be different. These things are also meant to be calculated sequentially through the layers, so let’s build them into the layers instead. That said, the first construct will require the size of each layer to be the same width. Actually, if we use lists rather than numpy arrays, we won’t each of the layers to be the same size because the data structure will be very flexible. Then we would be better off putting the data structures at the model level, and the calculation of those structures at the layer level, sending in layer and the previous layer (from which it is built). Actually that doesn’t make a whole lot of sense anymore, as the data is not in the previous layer, and because the data address depends on the layer number. If the data is in the model rather than the layers, the calculations belong in the model as well. Then the only thing the layer holds is the number of inputs, number of nodes and activation function, but those can all be stored at the model level and there would be no need for a Layer class. This might also be supported because we want to do matrix operations rather than iterable operations wherever we can.

For the MVP, we don’t need derivatives, just make networks able to train to data or with governing equations that are functions of the model inputs.

The model inputs shall be represented as the 0th layer in data structures.