Module Interface Specification for ANN (Artificial Neural Network)

Tanya Djavehrpour

March 19, 2024

1 Revision History

Date	Version	Notes
March 19	1.0	Initial Draft

2 Symbols, Abbreviations and Acronyms

See SRS Documentation Djavaherpour (2024b) at HERE.

Contents

3 Introduction

The following document details the Module Interface Specifications for ANN (Artificial Neural Network). This document specifies how every module is interfacing with every other parts.

Complementary documents include the System Requirement Specifications (SRS) Djavaherpour (2024b) and Module Guide (MG) Djavaherpour (2024a). The full documentation and implementation can be found at Github repository for ANN.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by ANN.

Data Type	Notation	Description
1D array	$\mathbf{A}_i$	A linear sequence of elements
2D matrix	$\mathbf{M}_{ij}$	A collection of elements arranged in rows and columns
3D matrix	$\mathbf{M}_{ijk}$	A structure composed of elements arranged in a grid with three dimensions
boolean	*bool*	True or False
string	*str*	A sequence of characters
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
positive Integer	$\mathbf{Z}_+$	a number without a fractional component in (0, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of ANN uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, ANN uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide Djavaherpour (2024a) document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	ANN Control Module Saved ANN Model Module Output Module Input Classifier Module Input Image Module Training Model Module
Software Decision	Input Preparing and Preprocessing Module Data Preparing and Preprocessing Module Training and Testing Module

Table 1: Module Hierarchy

6 MIS of ANN Control Module

6.1 Module

`main`

6.2 Uses

- Hardware-Hiding Module
- Saved ANN Model Module (7)
- Output Module (8)

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
`main`	-	-	-

6.4 Semantics

6.4.1 State Variables

None.

6.4.2 Environment Variables

None.

6.4.3 Assumptions

- The ANN Control Module assumes that the Hardware-Hiding Module, Saved ANN Model Module, and Output Module are implemented according to their specifications. However, it does include error handling to manage unexpected behaviors or failures in these modules.
- The system environment (operating system, hardware) is assumed to be stable. Also, essential libraries and dependencies are presumed to be correctly installed and configured.

6.4.4 Access Routine Semantics

main():

- transition: Initializes the program.

Note: As the ANN Control Module mainly serves as a coordinator between different modules without maintaining its own state or producing output, its primary function is to ensure the correct sequence of operations and interactions between these modules. It relies on the robustness of the called modules' error handling.

6.4.5 Local Functions

None.

7 MIS of Saved ANN Model Module

7.1 Module

`model`

7.2 Uses

- Hardware-Hiding Module
- Training and Testing Module (14)

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
`save_model`	-	*bool*	`PermissionError`
`load_model`	-	$[\mathbf{M}_{ijk}, \mathbf{M}_{ij}]$	`FileNotFoundError`

7.4 Semantics

7.4.1 State Variables

- `modelData`: Data structure holding the current ANN model's data. This is an array including weights and biases.

7.4.2 Environment Variables

`modelFile`: A file on the file system where the ANN model data is saved and from where it is loaded.

7.4.3 Assumptions

None.

7.4.4 Access Routine Semantics

`save_model`():

- transition: Writes the current state of the ANN model (weights and biases) to the `modelFile`.

- output: Returns `True` if model is saved successfully.
- exception: Raises `PermissionError` if the module lacks the necessary permissions to write to `modelFile`. It may also raise an IOError if there are issues with the file system, such as insufficient storage space.

`load_model`():

- transition: Reads the model data from `modelFile`.
- output: Returns the trained model based on weights and biases from `modelData`. The return value is a list. The first argument is a list of weights (each weight matrix is $\mathbf{M}_{ij}$), so that this argument is a $\mathbf{M}_{ijk}$. The second argument is a list of biases (each bias vector is $\mathbf{A}_i$). This argument is a $\mathbf{M}_{ij}$.
- exception: Raises `FileNotFoundError` if `modelFile` does not exist or cannot be accessed. Additionally, an exception may be raised for data corruption or format mismatch, indicating issues with the integrity or compatibility of the stored model data.

7.4.5 Local Functions

None.

8 MIS of Output Module

8.1 Module

`output`

8.2 Uses

- Hardware-Hiding Module
- Input Classifier Module (9)

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
`set_class`	-	-	-
`set_feedback`	*str*	*bool*	`PermissionError`

8.4 Semantics

8.4.1 State Variables

None.

8.4.2 Environment Variables

None.

8.4.3 Assumptions

None.

8.4.4 Access Routine Semantics

`get_class`():

- transition: Receives the classification result from the Input Classifier Module (9).
- output: None.
- exception: None.

`set_feedback`():

- transition: Sets user feedback on classification result.
- output: Returns a confirmation message or status after recording the feedback.
- exception: Raises `PermissionError` if the module lacks the necessary permissions to record feedback.

8.4.5 Local Functions

None.

9 MIS of Input Classifier Module

9.1 Module

`classifier`

9.2 Uses

- Hardware-Hiding Module
- Saved ANN Module (7)
- Input Preparing and Preprocessing Module (12)

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
`load_model`	-	-	-
`get_image_pixels`	-	-	-
`get_class`	-	*str*	-

9.4 Semantics

9.4.1 State Variables

- `modelData`: Data structure holding the current ANN model's data. This is an array including weights and biases.
- `imagePixels`: An array of input image's pixels after preprocessing.

9.4.2 Environment Variables

None.

9.4.3 Assumptions

None.

9.4.4 Access Routine Semantics

`load_model`():

- transition: Receives trained model from the Saved ANN Model Module (7) and saves it in `modelData`.
- output: None.
- exception: None.

`set_image_pixels`():

- transition: Receives the array of input image from Input Preparing and Preprocessing Module (12) and saves in `imagePixels`.
- output: None.
- exception: None.

`get_class`():

- transition: Classifies the input image.
- output: The class of the input image.
- exception: None.

10 MIS of Input Image Module

10.1 Module

`input`

10.2 Uses

- Hardware-Hiding Module

10.3 Syntax

10.3.1 Exported Constants

- `HEIGHT`: A value ($\mathbf{Z}_+$) describing acceptable height of input image (currently 32).
- `WIDTH`: A value ($\mathbf{Z}_+$) describing acceptable width of input image (currently 32).
- `IMAGE_FORMAT`: A list of strings (*str*) of acceptable types of input image (currently PNG and JPEG).

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
`set_input`	`File path`	-	`FileNotFoundError`, `InvalidSize`, `InvalidFormat`
`get_image_pixels`	-	$\mathbf{M}_{ijk}$	-

10.4 Semantics

10.4.1 State Variables

- `inputImage`: A $\mathbf{M}_{ijk}$ of the RGB input image.

10.4.2 Environment Variables

None.

10.4.3 Assumptions

None.

10.4.4 Access Routine Semantics

`set_input`(inputImagePath):

- transition: Receives the input image from end user and saves its matrix as `inputImage`.
- output: None.
- exception: Raises `FileNotFoundError` if `inputImagePath` does not exist or cannot be accessed. Also, `InvalidSize` is raised when the size of input image in not compatible with `HEIGHT` or `WIDTH`. Additionally, `InvalidFormat` is thrown if the input image's format is not compatible with `IMAGE_FORMAT`.

`get_image_pixels`():

- transition: Getter of the input image's matrix.
- output: $\mathbf{M}_{ijk}$ of `inputImage`.
- exception: None.

10.4.5 Local Functions

None.

11 MIS of Training Model Module

11.1 Module

`training_model`

11.2 Uses

- Data Preparing and Preprocessing Module (13)

11.3 Syntax

11.3.1 Exported Constants

- `LAYERS_NUMBER`: A value ($\mathbf{Z}_+$) describing the number of neural network's layers.
- `LAYERS_NEURONS`: An array including each layer's number of neurons.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
`create_gradients`	-	$\mathbf{A}_i$	-

11.4 Semantics

11.4.1 State Variables

None.

11.4.2 Environment Variables

None.

11.4.3 Assumptions

None.

11.4.4 Access Routine Semantics

`create_gradients`():

- transition: Creates zero arrays for all needed gradients based on the `LAYERS_NUMBER` and `LAYERS_NEURONS`.
- output: All gradients' zero vector ($\mathbf{A}_i$)
- exception: None.

11.4.5 Local Functions

None.

12 Input Preparing and Preprocessing Module

12.1 Module

`input_prep`

12.2 Uses

None.

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
`input_prep`	-	$\mathbf{A}_i$	-

12.4 Semantics

12.4.1 State Variables

- `input`: A $\mathbf{M}_{ijk}$ incling input image pixels.

12.4.2 Environment Variables

None.

12.4.3 Assumptions

None.

12.4.4 Access Routine Semantics

`input_prep`():

- transition: Prepares and preprocess the input image to change it the way model can use it to predicts the class.
- output: an $\mathbf{A}_i$ including prepared and preprocessed input image.
- exception: None.

12.4.5 Local Functions

- `set_image_pixels`():
  - transition: Receives the input image pixels and saves in `input`.
  - output: None.
  - exception: None.
- `rgb2gray`():
  - transition: Converts RGB data (`input`) into grayscale in order to reduce complexity.
  - output: Grayscaled input ($\mathbf{M}_{ijk}$).
  - exception: None.
- `prep_pixels`(grayInput):
  - transition: Normalizes grayscaled input to change the range of data between 0 and 1.
  - output: Normalized input.
  - exception: None.
- `flat_data`(normalInput):
  - transition: Data is flatten since input image should be vectorized with the size of 1024. After grayscaling input image is a $\mathbf{M}_{ij}$. this should be an $\mathbf{A}_i$ to be used by implemented model.
  - output: flatten input.
  - exception: None.

13 Data Preparing and Preprocessing Module

13.1 Module

`data`

13.2 Uses

None.

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
`get_train_dataset`	-	$[\mathbf{M}_{ij}, \mathbf{A}_i]$	`UnableToLoad`
`get_test_dataset`	-	$[\mathbf{M}_{ij}, \mathbf{A}_i]$	`UnableToLoad`

13.4 Semantics

13.4.1 State Variables

- `train_data`: Data structure holding train data images and their labels. Since train images after processing are vectors ($\mathbf{A}_i$), a list of these images is a matix ($\mathbf{M}_{ij}$). Alos, labels are saving in a vector ($\mathbf{A}_i$). Consequently, this data structure is a list in [$\mathbf{M}_{ij}$, $\mathbf{A}_i$] format.
- `test_data`: Data structure holding test data images and their labels. Since test images after processing are vectors ($\mathbf{A}_i$), a list of these images is a matix ($\mathbf{M}_{ij}$). Alos, labels are saving in a vector ($\mathbf{A}_i$). Consequently, this data structure is a list in [$\mathbf{M}_{ij}$, $\mathbf{A}_i$] format.
- `train_label`: An $\mathbf{A}_i$ incling traing data labels.
- `test_label`: An $\mathbf{A}_i$ incling test data labels.

13.4.2 Environment Variables

None.

13.4.3 Assumptions

None.

13.4.4 Access Routine Semantics

`get_train_dataset`():

- transition: Returns the prepared and processed train data.
- output: `train_data`.
- exception: None.

`get_test_dataset`():

- transition: Returns the prepared and processed test data.
- output: `test_data`.
- exception: None.

13.4.5 Local Functions

- `load_data`():
  - transition: Downloads and extracts dataset. After extracting data, this function saves `train_label` and `test_label`.
  - output: Unprocessed train and test data, and their labels. These images are in RGB.
  - exception: This function raises `UnableToDownload`, when there is a problem with downloading or extracting data.
- `rgb2gray`(unprocessedTrainData, unprocessedTestData):
  - transition: Converts RGB data into grayscale in order to reduce complexity.
  - output: Grayscaled train and test data ($\mathbf{M}_{ijk}$).
  - exception: None.
- `prep_pixels`(grayTrainData, grayTestData):
  - transition: Normalizes grayscaled images to change the range of data between 0 and 1.
  - output: Normalized train and test data.
  - exception: None.

- `flat_data`(normalTrainData, normalTestData):
  - transition: Data is flatten since imagws should be vectorized with the size of 1024. After grayscaling each image is a $\mathbf{M}_{ij}$. These images should be an $\mathbf{A}_i$ to be used in training and testing process by implemented model.
  - output: flatten train and test data.
  - exception: None.
- `shuffle_data`(flatTrainData, flatTestData):
  - transition: Attaches traind data and test data to their labels (`train_label` and `test_label`). Also, shuffle the attached data to improve the performance.
  - output: `train_data` and `test_data`.
  - exception: None.

14 MIS of Training and Testing Module

The details of fucntions using here are described in SRS document Djavaherpour (2024b).

14.1 Module

`train_and_test`

14.2 Uses

- Training Model Module (11)
- Data Preparing and Preprocessing Module (13)

14.3 Syntax

14.3.1 Exported Constants

- `BATCH_SIZE`: The partition size of the dataset for each step of learning, typically a power of two.
- `LEARNING_RATE`: Speed at which the model learns; controls adjustments to the weights.
- `EPOCHS`: Total number of training cycles through the entire dataset.

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
`train_and_test`	$\mathbf{A}_i$, $[\mathbf{M}_{ij}, \mathbf{A}_i]$, $[\mathbf{M}_{ij}, \mathbf{A}_i]$	$[\mathbf{M}_{ijk}, \mathbf{M}_{ij}]$	-
`set_layers`	-	-	-

14.4 Semantics

14.4.1 State Variables

- `layers`: Layers dimensions are defined based on the model architecture and are saved in an $\mathbf{A}_i$ named layers.

14.4.2 Environment Variables

None.

14.4.3 Assumptions

None.

14.4.4 Access Routine Semantics

`train_and_test`(gradientsArrays, train_data, test_data):

- transition: Train the model based on constant variables defined in 11. Gradient arrays and train data are needed for training. Input parameters are achieved from 11 and 13. At the same time, apply changes on `test_data` to make sure it is working and being trained correctly.
- output: Returns a data structure ($[\mathbf{M}_{ijk}, \mathbf{M}_{ij}]$) including weights and biases of trained model.
- exception: None.

`set_layers`():

- transition: `layers` is set based on Train Model Module (11).
- output: None.
- exception: None.

14.4.5 Local Functions

- `sigmoid`(x):
  - transition: Claculates sigmoid function for x, as the activation function.
  - output: Sigmoid value of x.
  - exception: None.
- `initialize_parameters`(layers):
  - transition: Allocates random normal weights ($\mathbf{M}_{ij}$) and zero biases ($\mathbf{A}_i$) for each layer.
  - output: Returns a dictionary (named `parameters`) that the keys define weights or biases, and the values are allocated random numbers to each of them.
  - exception: None.
- `compute_cost`(predicted, actual):
  - transition: Claculates the sum of the squared errors based on the predicted and actual values.
  - output: Returns the sum of the squared errors
  - exception: None.
- `feed_forward`(predicted, parameters, layersNumb):

- transition: Claculates feedforwarding process as described in SRS Djavaherpour (2024b). This is done by using the predicted value of previous step, parameters and the number of layers, `parameters` dictionary and the number of layers.
- output: Returns the new predicted value and a `cache` including new and old parameters.
- exception: None.

- `extract_parameters`(cache):
  - transition: Extracts parameters saved during forwardfeeding from the cache, based on `layers`, `parameters` dictionary and the number of layers.
  - output: Returns extracted parameters.
  - exception: None.

- `backpropagation`(cache, predicted, actual, layers):
  - transition: Claculates backpropagation process as described in SRS Djavaherpour (2024b) to calculate gradients of wights and biases.
  - output: A dictionary as gradients that keys are labels of gradients to define weights or biases, and values are gradients.
  - exception: None.

References

Tanya Djavaherpour. Module guide. https://github.com/tanya-jp/ANN-CAS741/blob/main/docs/Design/SoftArchitecture/MG.pdf, 2024a.

Tanya Djavaherpour. System requirements specification. https://github.com/tanya-jp/ANN-CAS741/blob/main/docs/SRS/SRS.pdf, 2024b.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.