DOCUMENTATION

This is a documentation about a fully connected neural network built in Ptolemy II. The purpose of this document is to describe the actors that have been used, specify the input data types and define its functionality.

We used Python to extract the weights, biases and validation data of a trained model for a digit recognition problem. The used dataset is the MNIST dataset, which contains:

- Training set: 50.000 images
- Test set: 10.000 images
- Validation set: 10.000 images

We will use this data (weights, validation data, biases) as inputs in our neural network in Ptolemy II to prove that it predicts the digits correctly.

The following network has an *input layer* of 784 neurons (28x28 pixels images of our dataset), a *hidden layer* with 30 neurons and an *output layer* of 10 neurons (numbers from 0 to 9).

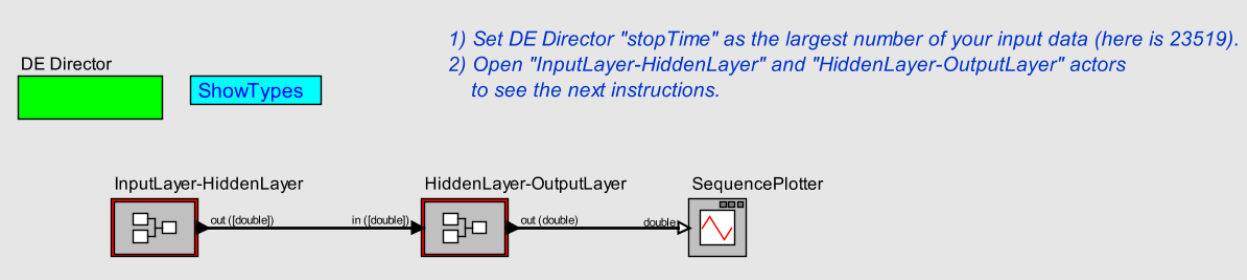


Image-1: Complete fully connected neural network in Ptolemy II.

Image-1 shows the fully connected neural network built in Ptolemy II platform. We have 3 actors here: *InputLayer-HiddenLayer*, *HiddenLayer-OutputLayer* (composite actors) and a *SequencePlotter* actor to display the results.

InputLayer-HiddenLayer

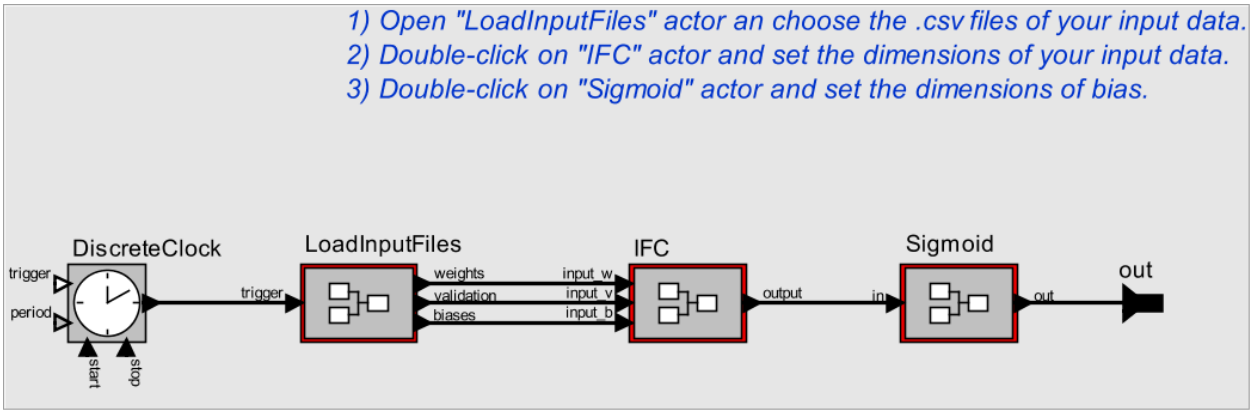


Image-2: InputLayer-HiddenLayer composite actor.

This composite actor reads the input data, do the mathematics between matrices and gives the output between the input layer and the hidden layer.

1. ***DiscreteClock***
   We use this actor to trigger *LineReader* actors in order to read the input data.

2. ***LoadInputFiles***

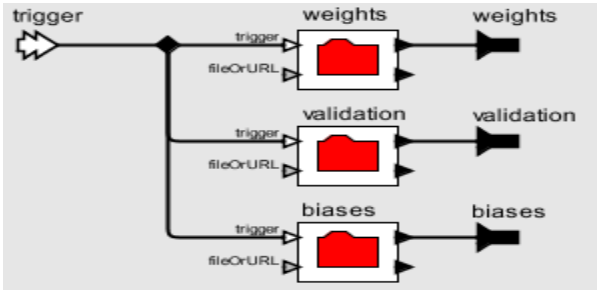


Image-3: LoadInputFiles composite actor.

This is a composite actor that is used to read the input data files. **Double-click** on weights, validation, biases *LineReader* actors to choose the .csv files of the input data. Input data files (weights, validation, biases) must be **.csv files** and the data in these files must be written as **vectors**. In our

example, input layer has 784 neurons and hidden layer 30 neuros. So, we have to convert our 784x30 matrix into a **(784x30)x1** vector. **Every row is turned into a column.**
The actor has one input port (*trigger*) and 3 output ports (*weights, validation, biases).*

3. IFC (InputFullyConnected)

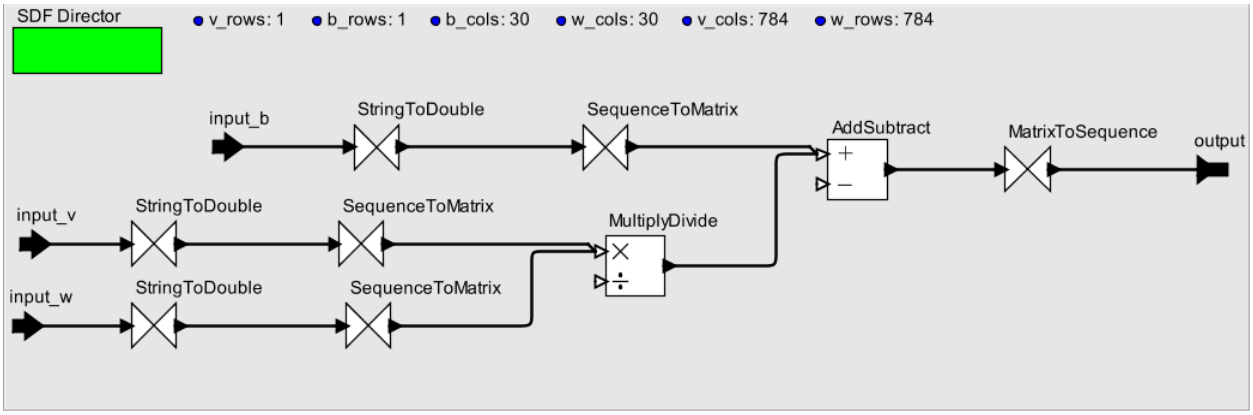


Image-4: IFC composite actor.

Here, we do the mathematical part of the network. **Double-click** on the *IFC* actor to write the dimensions of your input data you selected previously. Notice that validation data and biases (*v_rows, v_cols, b_rows, b_cols)* are transposed in order to satisfy the multiplication rules.

This actor has 3 input ports (input data) and one output port (the validation data for the next layer).

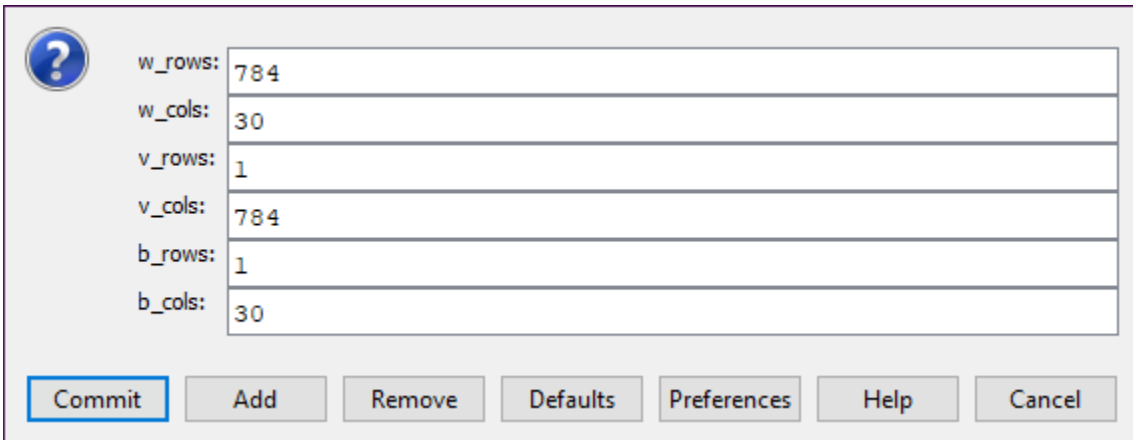


Image-5: IFC composite actor (double-click).

4. Sigmoid

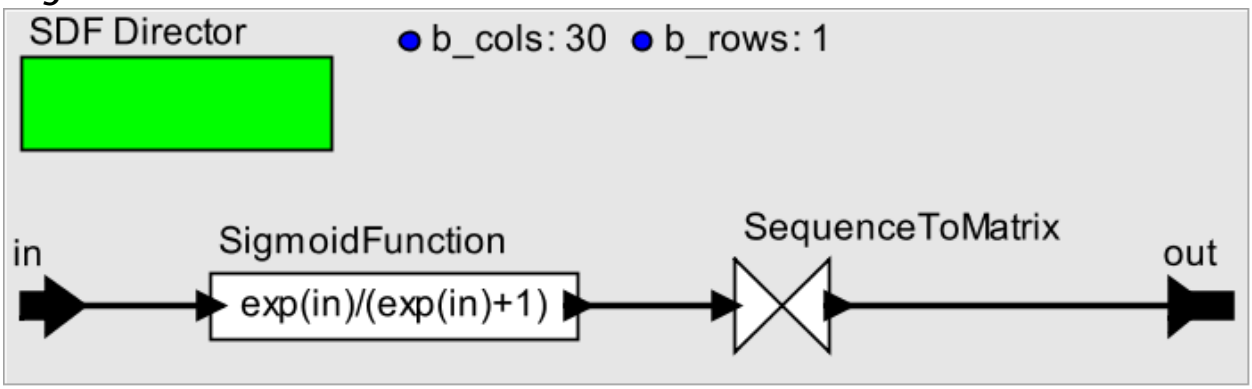


Image-6: Sigmoid composite actor.

This actor is used to apply the **activation function** on the output data. Here, we have *sigmoid* activation function. **Double-Click** on the on the *Sigmoid* actor to write the dimensions of the output data (same dimensions as biases, that's why we have *b_cols*, *b_rows*).

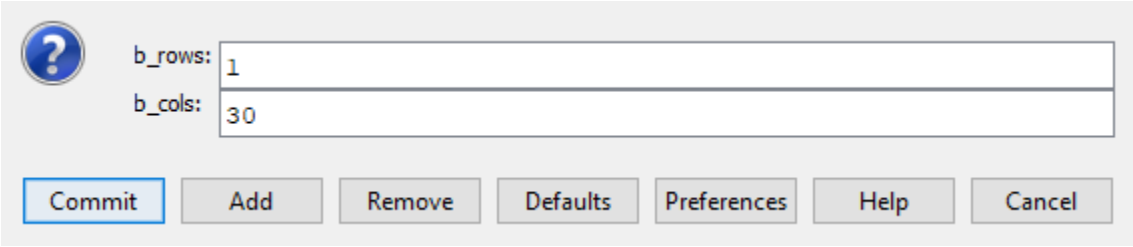


Image-7: Sigmoid composite actor (double-click).

HiddenLayer-OutputLayer

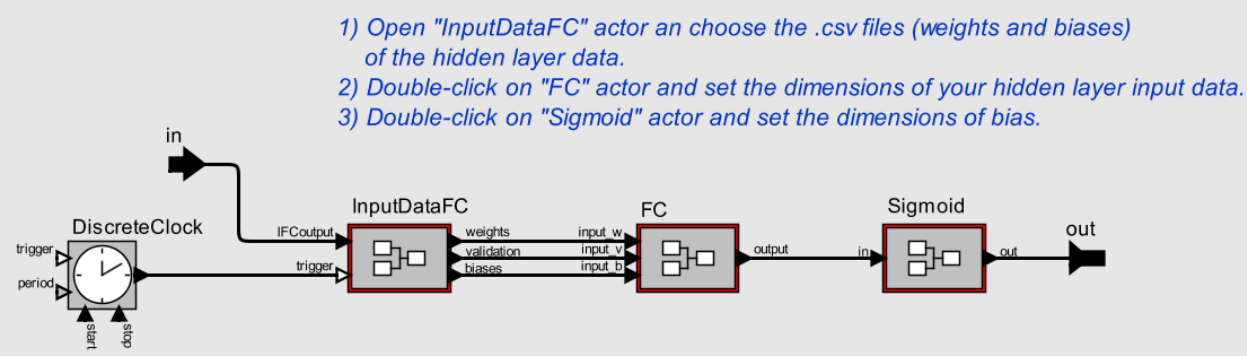


Image-8: HiddenLayer-OutputLayer composite actor.

Same as the *InputLayer-OutputLayer* composite actor, but this time for the connection between hidden layer and the output layer.

1. ***DiscreteClock***
   Same as before.

2. ***InputDataFC***

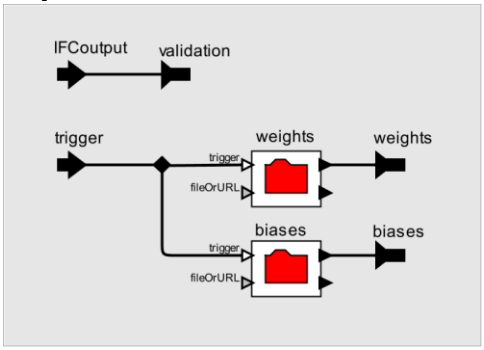


Image-9: InputDataFC composite actor.

For *weights* and *biases,* we do the same thing as before.
NOTE: .csv files **must** be in the form we discussed later!
This time, as *validation* data we use the output data of the output of the *InputLayer-HiddenLayer* composite actor.
Again, we have 2 input ports (*trigger, IFCOutput)* and 3 output ports (weights, validation, biases).

3. ***FC (Fully Connected)***

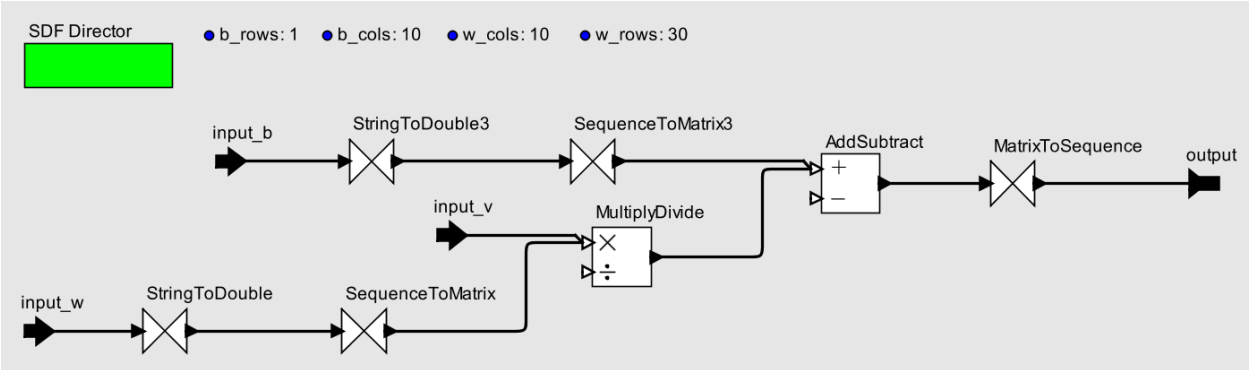


Image-10: FC composite actor.

Same as before, **double-click** and write the dimensions of the input data (this time only for weights and biases). Again, biases' dimensions are transposed.

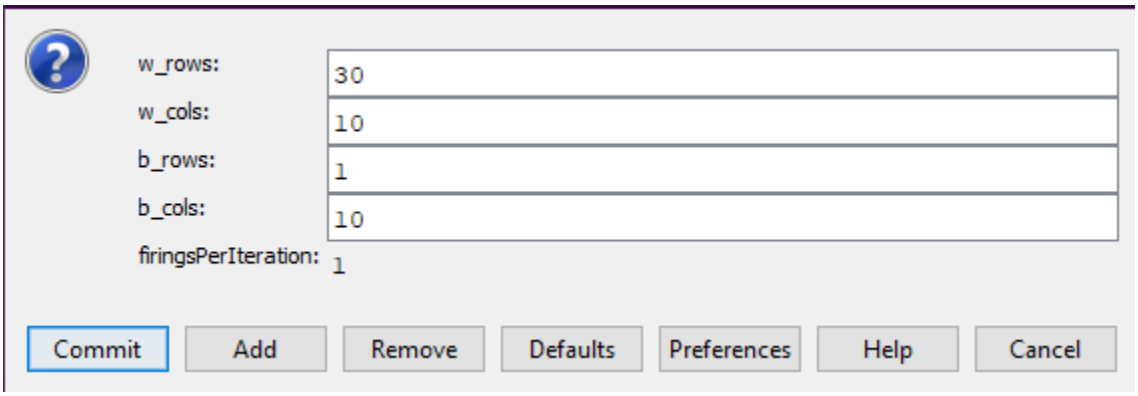


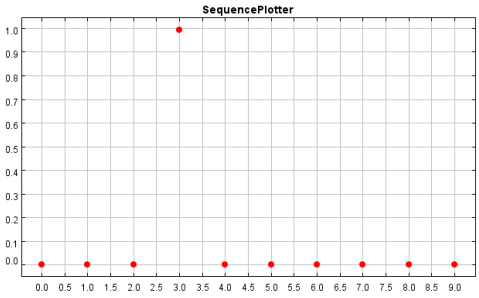
Image-11: FC composite actor (double click).

4. *Sigmoid*

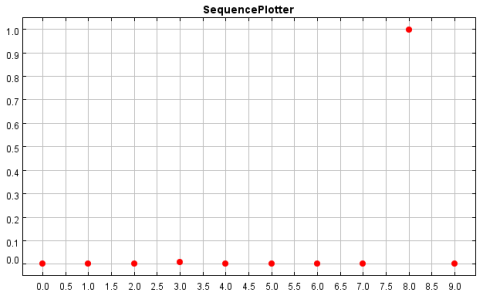
Same as before, but with the dimensions of the new biases (connection between hidden layer and output layer).

NOTE: we deleted the *SequenceToMatrix* actor, in order to display the results as sequences in a *SequencePlotter* actor.

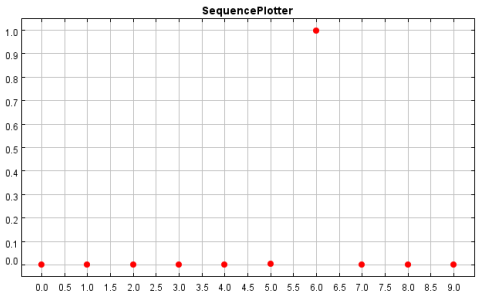
Here are some of the predictied results for the first 5 validation data in Ptolemy:



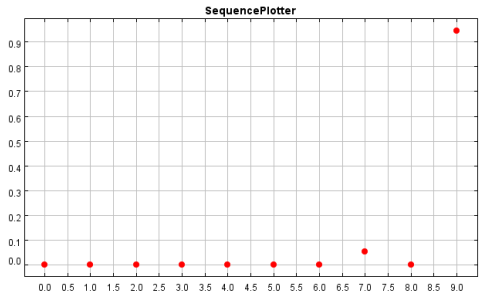
val1



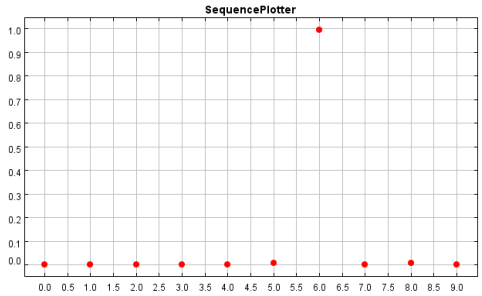
val2



val3



val4



Val5

In python, we used the command:

```
results = [(numpy.argmax(net.feedforward(x)), y) for (x, y) in validation_data]
```

in order to get the evaluation results on the validation data. The following image shows some pairs of predicted digits and actual digits:

predicted_digit	actual_digit
3	3
8	8
6	6
9	9
6	6
9	4
5	5
3	3
8	8
4	4
5	5

Image-12 evaluation on validation data – some predicted results

As we can see for the first 5 validation data, our network predicts the digits correctly!

NOTES

1. This Ptolemy II network works only in DE Domain. With some modifications it can also work in SDF Domain.
2. You can use more *HiddenLayer-OutputLayer* composite actors, if you want more hidden layers in your network.
3. You can also use another activation function. Just modify the mathimatical equation in *Expression* actor, inside the *Sigmoid* composite actor.
4. The .csv files format must be CSV (Comma Delimited).