10/17/2016

Principles of Intelligent Systems.

Assignment 2

Aidan Mcmillan – S2946192
&
Luke Hands – s2921970

Table of Contents

The Neural Network Structure

The structure of the neural network is based off of the data that we are working with. As there are 12 images, each having 144 possible values the neural network structure reflects this.

Within the neural network we have three layers in total, as well as connections between layers:

Input Layer
This layer is used for the data to be input. Therefore, it contains 144 neurons for data input. This layer also contains a bias neuron with a constant output of 1.

Hidden Layer
This layer can contain an arbitrary number of neurons. The number of neurons we have chosen to use is seven. This allows for the training algorithm to reach a result faster. This layer also contains a bias neuron with a constant output of 1.

Output Layer
As there are 12 possible pictures for the neural network to recognize there are 12 outputs as well. This is necessary in order for training and identification.

Connections
Connections were created between neurons of adjacent layers (E.g Input->Hidden->Output). These connections are initialized with random, small weights (0.5 to 0.5) and are the only parts of the neural network that are adjusted when training.

The Neural Network Data Structures:

The Neuron object has the following elements:

Float input.
Float output.
Function - Sigmoid Function.
Array lists of incoming and outgoing Connections.
The InputNeuron Function is to set output as inputs value.

The Connection Object has the following elements:

- Float weight
- Neuron from
- Neuron to

Pre & Post Data Processing

The Data in its raw form comes as a 12 * 12 pixel text file (*.txt) using the '*' to represent a dot.

This data needs to be shown in this form to the end user, so the data is first converted into 2d char array, where a blank pixel is represented as " " and "*" is represented as"*" keeping it very easy for the human to read.

This 2d char array is then converted into a 2d float array, where a " " is represented as a "0.00" and a "*" is represented as "1.00", the 2d array is then flattened into a 1d array to make feeding the data to the network easiest.

The Activation Function

We used the sigmoid function as shown in the lectures, we used this function because the Sigmoid is a squashing function. The sigmoid function allows the network to handle problems which are not just linear combinations for the inputs. This allows the network to handle problems that can't easily be sorted into two groups linearly. For example, the difference in complexity between and "or" gate and the "xor" gate. Without this function our network would not have been able to work.

The Learning Algorithm

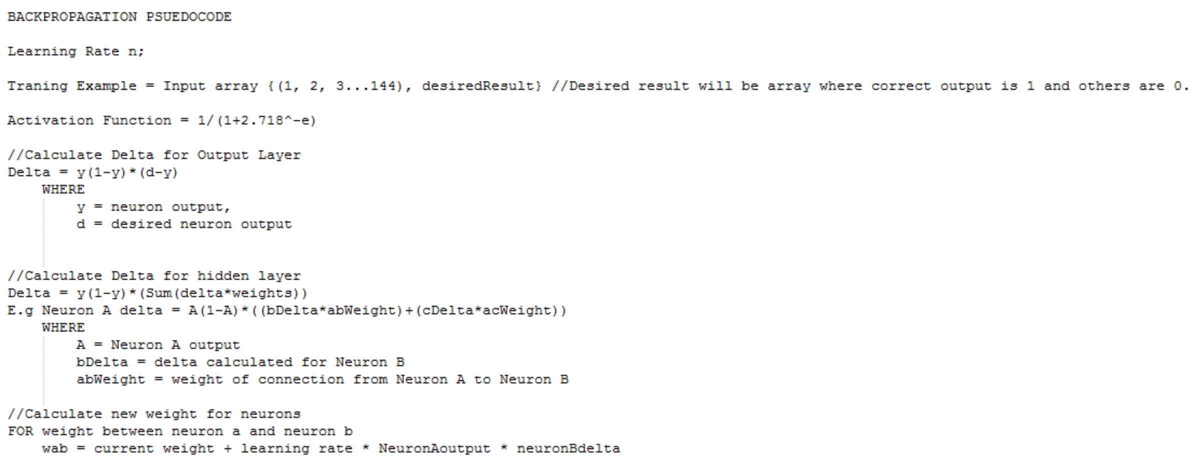
When training data the method of back propagation was used. This process consists of working backwards through a neural network and calculating the difference between the desired output and the actual output and adjusting the weights of the connections to mitigate the error value.

For each neuron a delta is calculated which is based off the difference between desired and actual output. The process of delta calculation varies between the output and hidden layers; the output layer bases its delta simply off of the difference between desired and actual output while the method used by the hidden layer is more complex.
As the hidden layer has outgoing connection to multiple neurons it must base its delta off of the products of deltas calculated for all the connected output neurons and the weight of the corresponding connections. This is why it is necessary to work backwards when calculating weights.

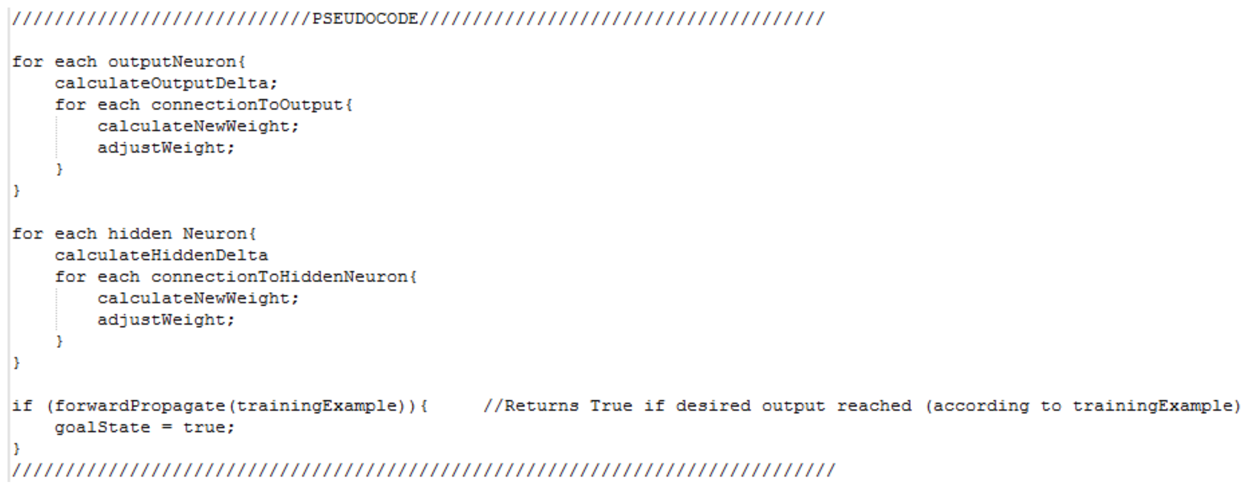
After delta is calculated for a neuron the weight of the connection going to that neuron is adjusted using four values:

- the current weight of a connection
- the output of the output neuron
- the delta of the input neuron
- a constant "learning rate" between 0 and 1 used to set the severity of change.

Pseudocode illustrating the complete method used as well as the equations used to calculate the deltas and weights can be found below.

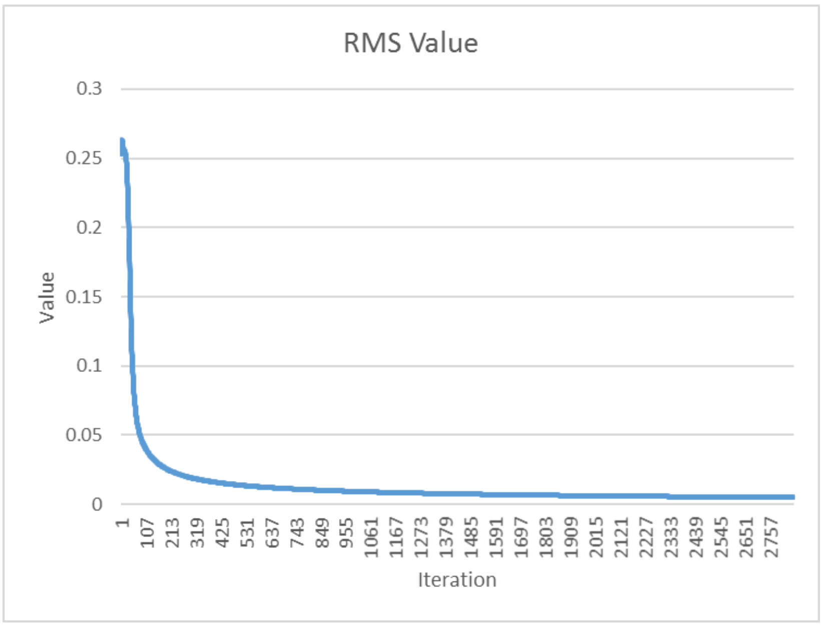


```
BACKPROPAGATION PSUEDOCODE

Learning Rate n;

Traning Example = Input array {(1, 2, 3...144), desiredResult} //Desired result will be array where correct output is 1 and others are 0.

Activation Function = 1/(1+2.718^-e)

//Calculate Delta for Output Layer
Delta = y(1-y)*(d-y)
    WHERE
        y = neuron output,
        d = desired neuron output


//Calculate Delta for hidden layer
Delta = y(1-y)*(Sum(delta*weights))
E.g Neuron A delta = A(1-A)*((bDelta*abWeight)+(cDelta*acWeight))
    WHERE
        A = Neuron A output
        bDelta = delta calculated for Neuron B
        abWeight = weight of connection from Neuron A to Neuron B

//Calculate new weight for neurons
FOR weight between neuron a and neuron b
    wab = current weight + learning rate * NeuronAoutput * neuronBdelta
```



```
///////////////////////////PSEUDOCODE/////////////////////////////////////

for each outputNeuron{
    calculateOutputDelta;
    for each connectionToOutput{
        calculateNewWeight;
        adjustWeight;
    }
}

for each hidden Neuron{
    calculateHiddenDelta
    for each connectionToHiddenNeuron{
        calculateNewWeight;
        adjustWeight;
    }
}

if (forwardPropagate(trainingExample)){      //Returns True if desired output reached (according to trainingExample)
    goalState = true;
}
////////////////////////////////////////////////////////////////////////////
```

RMS Error Graph



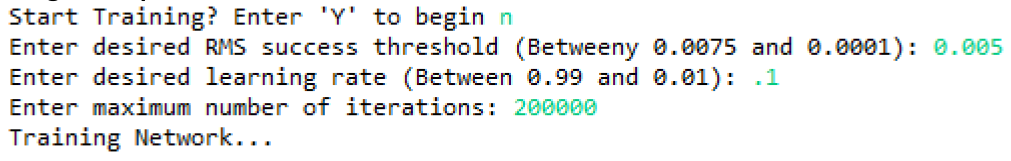
This graph represents how the change occurring in the RMS error calculation decreases exponentially per iteration.

This particular graph maps the values:

- RMS Threshold – 0.005
- Learning rate - 0.1

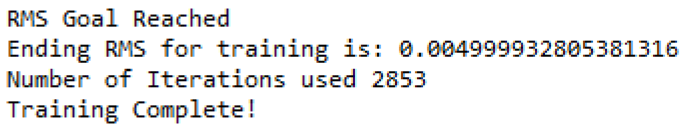
The neural network accurately recognizes patterns after the threshold of ~0.01 is reached; after that any more iterations just increase accuracy. The graph above has a much lower RMS threshold (0.005) which is why so many iterations have occurred. This is to display how little the effect of change is once the error is insignificantly small. A good way to increase accuracy is to decrease the value of the learning rate as the RMS decreases.

Program Input



```
Start Training? Enter 'Y' to begin n
Enter desired RMS success threshold (Betweeny 0.0075 and 0.0001): 0.005
Enter desired learning rate (Between 0.99 and 0.01): .1
Enter maximum number of iterations: 200000
Training Network...
```

Program Output

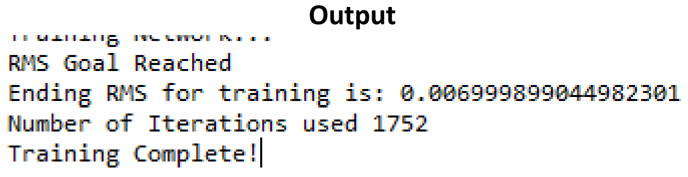


```
RMS Goal Reached
Ending RMS for training is: 0.004999932805381316
Number of Iterations used 2853
Training Complete!
```

Performance

In every iteration there is a clear indication of what the neural network thinks the image is and it was correct 100% of the time. The lowest value output was .83 for an image with 10% noise. Compared to the other outputs all registering <0.05 this clearly indicated a functioning neural network.

Output



```
RMS Goal Reached
Ending RMS for training is: 0.006999899044982301
Number of Iterations used 1752
Training Complete!
```

Image	Noise 5 Correct Output	Noise 10 Correct Output	Correct Output
1	0.98327416	0.9063856	YES
2	0.9469291	0.98000014	YES
3	0.98021424	0.8313372	YES
4	0.9646021	0.96587557	YES
5	0.9822124	0.9812608	YES
6	0.98224545	0.98079544	YES
7	0.9885118	0.87299097	YES
8	0.93459376	0.930498	YES
9	0.97774017	0.94341505	YES
10	0.986105	0.93457687	YES
11	0.98072696	0.97686714	YES
12	0.9883382	0.98684794	YES

(See appendix 1 for full values)

Limitations

There aren't any significant limitations the neural network suffers from as far as we have found (as per the assignment requirements). However, there has been some difficulty training it especially in reference to images 8 and 12 which cause slight conflict. This isn't a massive deal but is interesting to observe.

Overall the network works very well, the only improvement I would like to make is the introduction of better error-handling in the UI and possibly an alteration of the learning rate according to the current RMS value. I believe this would increase accuracy further as well as reducing the time taken to get an accurate result.

How To Use The Program

- Load an appropriate Java IDE
- Import the files from the zip into a new project.
- Copy and custom files for testing into the "custom" folder
- Run the assignment2 class.
- The assignment2 class is the main method
- The program will prompt a start. Enter "Y" or "N"

N:

- the program will ask for a desired RMS success Threshold.
- The program will ask for a desired learning rate between 0.99 and 0.01
- The program will ask for desired iterations.
- The program will continue on through the Y: sequence.

Y:

- The program will run the neural net for the training data.
- The user will be asked if they would like to load a custom file (Y/N)

Y:

- The user will then be asked for the name of the custom file including the extension, ie custom.txt (NOTE: these files MUST be in the custom folder of the project)
- The program will show the custom loaded picture to the user.
- The custom picture will then be forward propagated through the network.
- The results will be shown.
- The program will then show the picture it believes is closest to the custom image.

N:

- The program will ask the user if they would like to use the 5 degrees of noise or the 10 degrees of noise test sets.
- The program will then ask for a picture number to test.
- The picture will then be shown to the user.
- The pictures dataset will be forward propagated through the network.
- The results will be shown.
- The program will then show the image it believes is closest to the test image.

Summary of Contributions:

Luke:

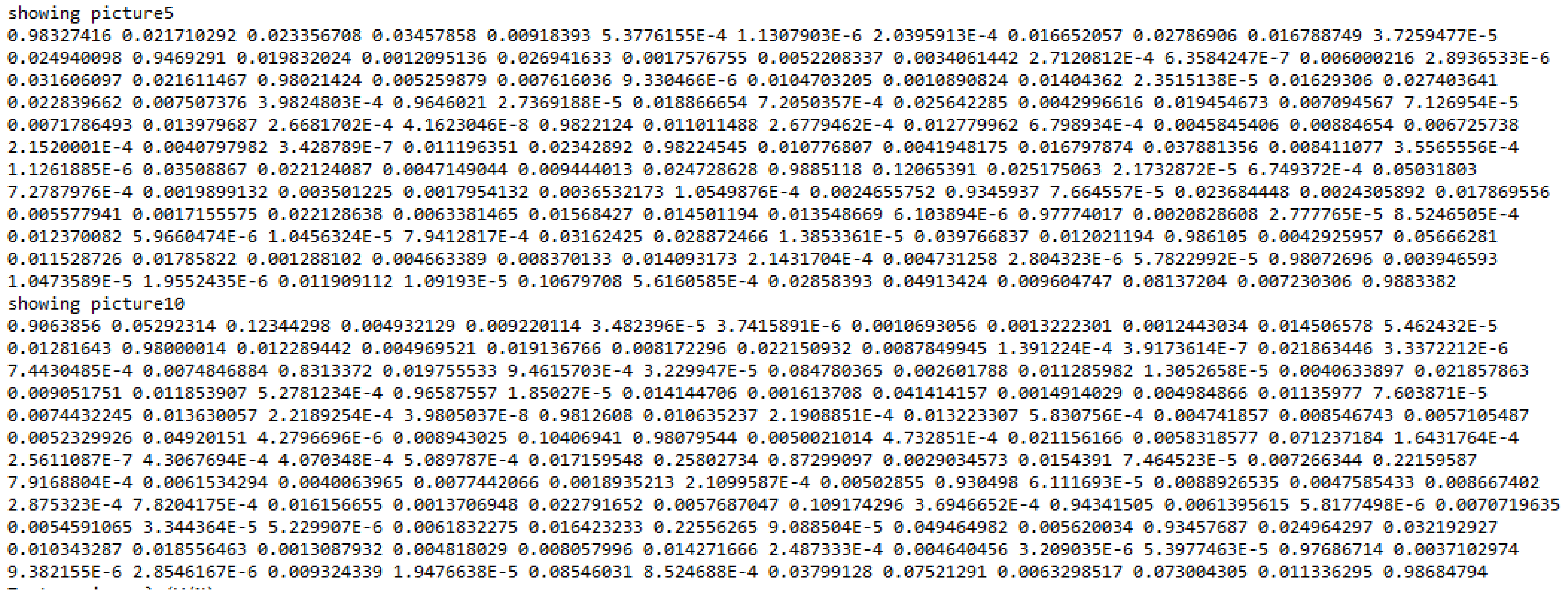
- Created the neuron class, including the child classes, input hidden & outputs.
- Created the Connection class.
- Started the network layer.
- Worked on document.
- Changed implementation of backpropagation method to handle more than one file.
- Implement assignment2 class for user interface of assignment criteria.
- Implemented custom file ability.
- Testing the neural network.

Aidan:

- Changed implementation of network layer for forward propagation.
- Implemented back propagation
- Implemented error handling of hidden and input connection weights.
- Worked on document
- Tested Neural network.
- Implemented RMS & Graphing
- Implemented Customizable network capability
- Debugged network layer.

Both team members worked really well together and shared the workload equally. Different Sections were completed by different members but everything was understood equally by everyone.

Appendix 1



```
showing picture5
0.98327416 0.021710292 0.023356708 0.03457858 0.00918393 5.3776155E-4 1.1307903E-6 2.0395913E-4 0.016652057 0.02786906 0.016788749 3.7259477E-5
0.024940098 0.9469291 0.019832024 0.0012095136 0.026941633 0.0017576755 0.0052208337 0.0034061442 2.7120812E-4 6.3584247E-7 0.006000216 2.8936533E-6
0.031606097 0.021611467 0.98021424 0.005259879 0.007616036 9.330466E-6 0.0104703205 0.0010890824 0.01404362 2.3515138E-5 0.01629306 0.027403641
0.022839662 0.007507376 3.9824803E-4 0.9646021 2.7369188E-5 0.018866654 7.2050357E-4 0.025642285 0.0042996616 0.019454673 0.007094567 7.126954E-5
0.0071786493 0.013979687 2.6681702E-4 4.1623046E-8 0.9822124 0.011011488 2.6779462E-4 0.012779962 6.798934E-4 0.0045845406 0.00884654 0.006725738
2.1520001E-4 0.0040797982 3.428789E-7 0.011196351 0.02342892 0.98224545 0.010776807 0.0041948175 0.016797874 0.037881356 0.008411077 3.5565556E-4
1.1261885E-6 0.03508867 0.022124087 0.0047149044 0.009444013 0.024728628 0.9885118 0.12065391 0.025175063 2.1732872E-5 6.749372E-4 0.05031803
7.2787976E-4 0.0019899132 0.003501225 0.0017954132 0.0036532173 1.0549876E-4 0.0024655752 0.9345937 7.664557E-5 0.023684448 0.0024305892 0.017869556
0.005577941 0.0017155575 0.022128638 0.0063381465 0.01568427 0.014501194 0.013548669 6.103894E-6 0.97774017 0.0020828608 2.777765E-5 8.5246505E-4
0.012370082 5.9660474E-6 1.0456324E-5 7.9412817E-4 0.03162425 0.028872466 1.3853361E-5 0.039766837 0.012021194 0.986105 0.0042925957 0.05666281
0.011528726 0.01785822 0.001288102 0.004663389 0.008370133 0.014093173 2.1431704E-4 0.004731258 2.804323E-6 5.7822992E-5 0.98072696 0.003946593
1.0473589E-5 1.9552435E-6 0.011909112 1.09193E-5 0.10679708 5.6160585E-4 0.02858393 0.04913424 0.009604747 0.08137204 0.007230306 0.9883382
showing picture10
0.9063856 0.05292314 0.12344298 0.004932129 0.009220114 3.482396E-5 3.7415891E-6 0.0010693056 0.0013222301 0.0012443034 0.014506578 5.462432E-5
0.01281643 0.98000014 0.012289442 0.004969521 0.019136766 0.008172296 0.022150932 0.0087849945 1.391224E-4 3.9173614E-7 0.021863446 3.3372212E-6
7.4430485E-4 0.0074846884 0.8313372 0.019755533 9.4615703E-4 3.229947E-5 0.084780365 0.002601788 0.011285982 1.3052658E-5 0.0040633897 0.021857863
0.009051751 0.011853907 5.2781234E-4 0.96587557 1.85027E-5 0.014144706 0.001613708 0.041414157 0.0014914029 0.004984866 0.01135977 7.603871E-5
0.0074432245 0.013630057 2.2189254E-4 3.9805037E-8 0.9812608 0.010635237 2.1908851E-4 0.013223307 5.830756E-4 0.004741857 0.008546743 0.0057105487
0.0052329926 0.04920151 4.2796696E-6 0.008943025 0.10406941 0.98079544 0.0050021014 4.732851E-4 0.021156166 0.0058318577 0.071237184 1.6431764E-4
2.5611087E-7 4.3067694E-4 4.070348E-4 5.089787E-4 0.017159548 0.25802734 0.87299097 0.0029034573 0.0154391 7.464523E-5 0.007266344 0.22159587
7.9168804E-4 0.0061534294 0.0040063965 0.0077442066 0.0018935213 2.1099587E-4 0.00502855 0.930498 6.111693E-5 0.0088926535 0.0047585433 0.008667402
2.875323E-4 7.8204175E-4 0.016156655 0.0013706948 0.022791652 0.0057687047 0.109174296 3.6946652E-4 0.94341505 0.0061395615 5.8177498E-6 0.0070719635
0.0054591065 3.344364E-5 5.229907E-6 0.0061832275 0.016423233 0.22556265 9.088504E-5 0.049464982 0.005620034 0.93457687 0.024964297 0.032192927
0.010343287 0.018556463 0.0013087932 0.004818029 0.008057996 0.014271666 2.487333E-4 0.004640456 3.209035E-6 5.3977463E-5 0.97686714 0.0037102974
9.382155E-6 2.8546167E-6 0.009324339 1.9476638E-5 0.08546031 8.524688E-4 0.03799128 0.07521291 0.0063298517 0.073004305 0.011336295 0.98684794
```