G+1 0 আরো পরবর্তী ব্লগ» ব্লগ তৈরি করুন সাইন ইন

Arduino Basics

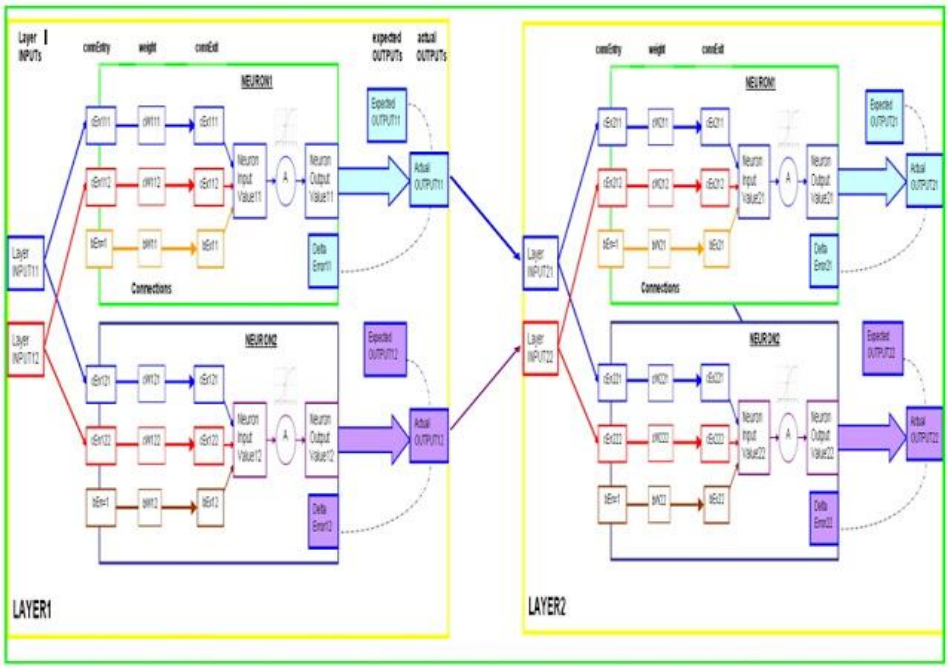
An Arduino tutorial blog. Free Arduino tutorials for everyone !

12 August 2011

Neural Network (Part 4): The Neural Network class

The Neural Network

Finally, we made it to the Neural Network class. This is the top level class that manages the communication between the layers. This class provides the necessary glue to hold all the layers together. It also controls the flow of information forwards and backwards from layer to layer. Lets have a look at a simple 2 layer neural network.



The above neural network starts with 2 inputs in Layer1 and finishes with 2 outputs in Layer 2.

This Neural Network structure is very modular in design. Layers can be added quite easily without affecting the functionality of the neural network code. The only layer that is treated a little bit differently from other layers in the neural network, is the last layer. However, any/all preceding layers are treated exactly the same. And the differences in the last layer, only really come into effect in Neural Network training.

When building my network, I just need to make sure that the number of outputs (or neurons) in the current layer match the number of connections in the next. If I wanted to create a neural network, that accepted 3 input signals from the outside world, to process them with 10 neurons, but only wanted to output one result... I could build the neural network in the following way.

addLayer(3,10) : This first layer would have 3 layerINPUTs and 10 neurons
addLayer(10,1) : This second (and last) layer would have 10 layerINPUTs and 1 neuron

The neural network class would manage these two layers, to ensure that information was passed from one to the other seemlessly. Here is the code for the Neural Network class that would make it possible:

Search This Blog
Search

Translate
Select Language
Powered by Google Translate

Pages
Arduino Basics Projects Page
Forum
Arduino Basics YouTube Videos
Arduino Webserver Data Viewer
Money Jar

Connect
Follow @ArduinoBasics
YouTube 534
Follow me on
Instagram
Follow me on Periscope
ArduinoBasics

If you like this site, feel free to put a TIP into my money jar. It will be used to buy a Digital Storage Oscilloscope.

DONATE
to ArduinoBasics

processing code Neural Network Class

```
01
02  /* ------------------------------------------------------------------------------
03     The Neural Network class is a container to hold and manage all the layers
04     ------------------------------------------------------------------------------
05  */
06
07  class NeuralNetwork{
08    Layer[] layers = {};
09    float[] arrayOfInputs={};
10    float[] arrayOfOutputs={};
11    float learningRate;
12    float networkError;
13    float trainingError;
14    int retrainChances=0;
15
16    NeuralNetwork(){
17      /* the default learning rate of a neural network is set to 0.1, which can changed by
18  the setLearningRate(lR) function. */
19      learningRate=0.1;
20    }
21
22
23
24    /* Function to add a Layer to the Neural Network */
25    void addLayer(int numConnections, int numNeurons){
26      layers = (Layer[]) append(layers, new Layer(numConnections,numNeurons));
27    }
28
29
30
31    /* Function to return the number of layers in the neural network */
32    int getLayerCount(){
33        return layers.length;
34    }
35
36
37
38    /* Function to set the learningRate of the Neural Network */
39    void setLearningRate(float tempLearningRate){
40      learningRate=tempLearningRate;
41    }
42
43
44
45    /* Function to set the inputs of the neural network */
46    void setInputs(float[] tempInputs){
47      arrayOfInputs=tempInputs;
48    }
49
50
51
52    /* Function to set the inputs of a specified layer */
53    void setLayerInputs(float[] tempInputs, int layerIndex){
54      if(layerIndex>getLayerCount()-1){
55        println("NN Error: setLayerInputs: layerIndex=" + layerIndex + " exceeded limits= "
56   + (getLayerCount()-1));
57      } else {
58        layers[layerIndex].setInputs(tempInputs);
59      }
60    }
61
62
63
64    /* Function to set the outputs of the neural network */
65    void setOutputs(float[] tempOutputs){
66      arrayOfOutputs=tempOutputs;
67    }
68
69
70
71    /* Function to return the outputs of the Neural Network */
72    float[] getOutputs(){
73      return arrayOfOutputs;
74    }
75
76
77
78    /* Function to process the Neural Network's input values and convert them to an output
```

Total Pageviews

2,066,740

Subscribe

Posts

Comments

```
pattern using ALL layers in the network */
  void processInputsToOutputs(float[] tempInputs){
    setInputs(tempInputs);

    /* Check to make sure that the number of NeuralNetwork inputs matches the Neuron Conn
ection Count in the first layer. */
    if(getLayerCount()>0){
      if(arrayOfInputs.length!=layers[0].neurons[0].getConnectionCount()){
        println("NN Error: processInputsToOutputs: The number of inputs do NOT match the
NN");
        exit();
      } else {
        /* The number of inputs are fine : continue */
        for(int i=0; i<getLayerCount(); i++){

          /*Set the INPUTs for each layer: The first layer gets it's input data from the
NN whereas the 2nd and subsequent layers get their input data from the previous layer's a
ctual output. */
          if(i==0){
            setLayerInputs(arrayOfInputs,i);
          } else {
            setLayerInputs(layers[i-1].actualOUTPUTs, i);
          }

          /* Now that the layer has had it's input values set, it can now process this da
ta, and convert them into an output using the layer's neurons. The outputs will be used a
s inputs in the next layer (if available). */
          layers[i].processInputsToOutputs();
        }
        /* Once all the data has filtered through to the end of network, we can grab the
actualOUTPUTs of the LAST layer
           These values become or will be set to the NN output values (arrayOfOutputs), t
hrough the setOutputs function call. */
        setOutputs(layers[getLayerCount()-1].actualOUTPUTs);
      }
    }else{
      println("Error: There are no layers in this Neural Network");
      exit();
    }
  }




  /* Function to train the entire network using an array. */
  void trainNetwork(float[] inputData, float[] expectedOutputData){
    /* Populate the ENTIRE network by processing the inputData. */
    processInputsToOutputs(inputData);

    /* train each layer - from back to front (back propagation) */
    for(int i=getLayerCount()-1; i>-1; i--){
      if(i==getLayerCount()-1){
        layers[i].setDeltaError(expectedOutputData);
        layers[i].trainLayer(learningRate);
        networkError=layers[i].getLayerError();
      } else {
    /* Calculate the expected value for each neuron in this layer (eg. HIDDEN LAYER) */
        for(int j=0; j<layers[i].getNeuronCount(); j++){
          /* Reset the delta error of this neuron to zero. */
          layers[i].neurons[j].deltaError=0;
   /* The delta error of a hidden layer neuron is equal to the SUM of [the PRODUCT of the
 connection.weight and error of the neurons in the next layer(eg OUTPUT Layer)].      Con
nection#1 of each neuron in the output layer connect with Neuron#1 in the hidden layer */
          for(int k=0; k<layers[i+1].getNeuronCount(); k++){
            layers[i].neurons[j].deltaError += (layers[i+1].neurons[k].connections[j].wei
ght * layers[i+1].neurons[k].deltaError);
          }
          /* Now that we have the sum of Errors x weights attached to this neuron.
             We must multiply it by the derivative of the activation function. */
          layers[i].neurons[j].deltaError *= (layers[i].neurons[j].neuronOutputValue * (1
-layers[i].neurons[j].neuronOutputValue));
        }
        /* Now that you have all the necessary fields populated, you can now
           Train this hidden layer and then clear the Expected outputs, ready for the nex
t round */
        layers[i].trainLayer(learningRate);
        layers[i].clearExpectedOUTPUT();
      }
    }
  }
```

```
161
162
163
164   /* Function to train the entire network, using an array of input and expected data with
165 in an ArrayList */
166   void trainingCycle(ArrayList trainingInputData, ArrayList trainingExpectedData, Boolean
167  trainRandomly){
168       int dataIndex;
169
170       /* re-initialise the training Error with every cycle */
171       trainingError=0;
172
173       /* Cycle through the training data either randomly or sequentially */
174       for(int i=0; i<trainingInputData.size(); i++){
175         if(trainRandomly){
176           dataIndex=(int) (random(trainingInputData.size()));
177         } else {
178           dataIndex=i;
179         }
180
181         trainNetwork((float[]) trainingInputData.get(dataIndex),(float[]) trainingExpecte
182 dData.get(dataIndex));
183
184         /* Use the networkError variable which is calculated at the end of each individua
185 l training session to calculate the entire trainingError. */
186         trainingError+=abs(networkError);
187       }
188   }
189
190
191
192
193
194   /* Function to train the network until the Error is below a specific threshold */
195   void autoTrainNetwork(ArrayList trainingInputData, ArrayList trainingExpectedData, floa
196 t trainingErrorTarget, int cycleLimit){
197     trainingError=9999;
198     int trainingCounter=0;
199
200
201     /* cycle through the training data until the trainingError gets below trainingErrorTa
202 rget (eg. 0.0005) or the training cycles have exceeded the cycleLimit variable (eg. 10000
203 ). */
204     while(trainingError>trainingErrorTarget && trainingCounter<cycleLimit){
205
206       /* re-initialise the training Error with every cycle */
207       trainingError=0;
208
209       /* Cycle through the training data randomly */
210       trainingCycle(trainingInputData, trainingExpectedData, true);
211
212       /* increment the training counter to prevent endless loop */
213       trainingCounter++;
214     }
215
216     /* Due to the random nature in which this neural network is trained. There may be occ
217 asions when the training error may drop below the threshold. To check if this is the case
218 , we will go through one more cycle (but sequentially this time), and check the trainingE
219 rror for that cycle. If the training error is still below the trainingErrorTarget, then w
220 e will end the training session. If the training error is above the trainingErrorTarget,
221 we will continue to train. It will do this check a  Maximum of 9 times. */
222     if(trainingCounter<cycleLimit){
223        trainingCycle(trainingInputData, trainingExpectedData, false);
224        trainingCounter++;
225
226        if(trainingError>trainingErrorTarget){
227          if (retrainChances<10){
228            retrainChances++;
229            autoTrainNetwork(trainingInputData, trainingExpectedData,trainingErrorTarget,
230 cycleLimit);
231           }
232        }
233
234     } else {
235       println("CycleLimit has been reached. Has been retrained " + retrainChances + " tim
236 es.  Error is = " + trainingError);
237     }
238   }
239 }
240
```

code formatter

The neural network constructor: NeuralNetwork() - automatically sets the learning rate to 0.1. Other than that, the Neural Network object is an empty shell waiting to be filled. The main setup function of the Neural Network class is the addLayer() function, which adds a layer with a specified number of connections and neurons.

Here are the functions of the Neural Network (NN) class:

addLayer() : adds a layer to the NN (from left to right)

getLayerCount() : returns the number of layers in the NN

setLearningRate(): sets the learning Rate to a specified value.

setInputs(): sets the inputs of the NN, and become the layerINPUTs of the 1st layer

setLayerInputs(): set the inputs of a specified layer

setOutputs(): set the outputs of the NN= same as the actualOUTPUTs of the last layer.

getOutputs(): returns the outputs of the NN

processInputsToOutputs(): Converts the NN inputs into outputs by feeding through layers.

trainNetwork(): uses a training set to train the neural network (using an Array).

trainingCycle(): uses a training set to train the neural network (using an ArrayList).

autoTrainNetwork(): cycles through the training data until a specific condition is met

Ok - so we now have all of the structural components required to build a feed forward neural network. And here is how you would create it. Let us build a neural network that accepts data from 4 sensors and has 3 layers . The layers will have 6, 8 and 2 neurons respectively...

```
NeuralNetwork NN = new NeuralNetwork();

NN.addLayer(4,6);

NN.addLayer(6,8);

NN.addLayer(8,2);
```

Perfect, we have just created an entire neural network, with randomised weights and biases etc etc.
Unfortunately, the neural network is not that useful at the moment. Before we can even start to use it, we need to put it through school. We need to teach it.

Try to imagine a colour that you have never seen before. Hard isn't it ? That is how the Neural Network feels. So before you can get it to make any sort of classifications, you have to show it some examples of what you are looking for. Once the neural network can make sense of your examples, you use a "validation set" to put it through it's paces. If it performs well, then you are good to go, otherwise it is back to school until it can pass the test. My neural network doesn't have a validation set at the moment, but it seems to work quite well without it. This statement is not entirely true. I tend to validate it using the training data, but maybe in future I will fix it up to use a proper validation set.

So how do you train the neural network ? Back-propagation !

Up Next: Neural Network (Part 5): Back Propagation

To go back to the table of contents click here

Posted by Scott C at 22:47 G+1 Recommend this on Google

Labels: ArduinoBasics, best arduino blog, code, Connection, Layer, Neural Network, Neuron, Processing.org, Programming, project, tutorial

6 comments:

Anonymous 26 February 2012 at 09:36

This is so great, thank you for sharing your code!

Reply

Replies

ScottC 28 February 2012 at 18:35

No problem, I am glad I could help.

Reply

Anonymous 4 May 2012 at 01:16

Great bit of code and well documented.

In line 197 above:

if(trainingError>trainingErrorTarget)

Should this not be:

if(trainingError<trainingErrorTarget)

Many thanks
Andrew

Reply

Replies

ScottC 4 May 2012 at 21:07

Hi Andrew,
The WHILE loop (before the line in question) will cycle through randomly until the cycle count exceeds 10000 cycles or until the training error drops below 0.005 (which ever comes first). The IF statement after the WHILE LOOP checks to see the cycle count is responsible for breaking the WHILE LOOP conditions. If the cycle count has exceeded the maximum cycle count (or cycleLimit), then it will display a message to that effect. On the other hand, if there are still some cycles remaining (trainingCounter<cycleLimit), it means that the reason the WHILE loop has exited, was due to a trainingError that has dropped below 0.005. Therefore, it will cycle through one more time, however on this occasion, will cycle through sequentially (rather than randomly). At the end of this cycle, a new trainingError will be set, and will be tested against the trainingTarget of 0.005. If on this occasion it is now GREATER than 0.005, then it will need to go back and try again (by training randomly once again). Hence the call to autoTrainNetwork. It will only have 9 chances to get it right, before it gives up.

I hope this explains it a bit better. The main aim of this part of code was to randomise the training. There are probably better ways of doing this, but the way I did it made sense to me at the time.

Reply

Selçuk KARAOSMANOĞLU 5 June 2014 at 15:07

In my system there are two outputs but at the time of training sometimes one of the outputs will not be able to provided. However i want the NN produce output for that output too. Is this possible? In this case how can we apply the single data?

thanks for your help...

Reply

Replies

Scott C 11 June 2014 at 00:48

You will need to post your query in a forum. I have no idea.

Reply

Enter your comment...

Comment as: Unknown (Goo ▾ Sign out

Publish Preview Notify me

Feel free to leave a comment about this tutorial below.
Any questions about your particular project should be asked in the ArduinoBasics forum.

Comments are moderated due to large amount of spam.

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

G+1 6.7k

This Week's Most Popular Posts

433 MHz RF module with Arduino Tutorial 1

HC-SR04 Ultrasonic Sensor

Simple Arduino Serial Communication

Relay Module

433 MHz RF module with Arduino Tutorial 2

Most Recent Posts

Get Arduino Data over the internet using jQuery and AJAX

Two Million Views

Generosity Campaign Update - Day 3

Generosity Campaign Update - Day 2

Generosity campaign - Day 1

Recent Posts Widget by Helplogger

Awesome Inc. template. Powered by Blogger.

G+1 0 আরো পরবর্তী ব্লগ» mdshs31@gmail.com ড্যাশবোর্ড সাইন আউট

Arduino Basics

An Arduino tutorial blog. Free Arduino tutorials for everyone !

12 August 2011

Neural Network (Part 5): The Back Propagation process

Back-propagation

Back propagation is the process by which you move backwards through the neural network to adjust the weights and biases so as to reduce the total error of the network. The total error of the network is essentially the difference between the end results (*actual* Outputs) and the *expected* results. If you expected to get a result of 1, but instead got a 0: you would go back through the network and tweak each of the weights (and bias) values so that your end result was a little bit closer to 1 than before.

The process of back-propagation is such that larger errors and larger weights and biases that create those errors are penalised more than their smaller counterparts. Bigger weights have a bigger influence on the final outcome than smaller weights, and are therefore penalised more for incorrect answers.

After many training cycles, the neural network reaches a stage of equilibrium (not quite, but close enough), whereby the tweaking is insignificant to the final outcome.

If you under-train, then you will get the wrong result more often than desired.
If you over-train, then the neural network will not be able to "think outside the sqaure", so to speak.

So how do you propagate backwards ??

Step 1: Feed-forward pass through:
Send some data through the network to populate all the variables. This feed-forward pass allows you calculate your actualOUTPUTs, which you will use to compare against your expectedOUTPUTs.

Step 2: Calculate delta-error for the neurons in the last layer (output layer).
The delta-error calculation for the neuron(s) in the last layer of the neural network is a liitle bit different than the other layers. You can work this out once you calculate the actualOUTPUTs from the feedforward pass.

Let Last Layer Neuron1.deltaError = LLN1.dE
Last Layer.actualOutput1 = aO1 <--- This is the same as the Neuron1 Output Value
Last Layer.expectedOutput1 = exO1

LLN1.dE = (aO1) x (1-aO1) x (exO1 - aO1);

Once you have calculated the deltaError for every neuron in the last layer (output layer), you can move onto the next step.

Step 3: Calculate the delta-error for the hidden layer neurons

The hidden layers for this neural network, is any layer in the neural network, that is not the last layer. However, each layer should sit like ducks in a row. And we are now going to calculate the delta error for the second last layer in the neural network. This could in theory be the first layer in the network (if this network only had 2 layers).

HLN = Hidden Layer Neuron,

Search This Blog
Search

Translate
Select Language
Powered by Google Translate

Pages
Arduino Basics Projects Page
Forum
Arduino Basics YouTube Videos
Arduino Webserver Data Viewer
Money Jar

Connect
Follow @ArduinoBasics
YouTube 534
Follow me on
Instagram
Follow me on Periscope
ArduinoBasics

If you like this site, feel free to put a TIP into my money jar. It will be used to buy a Digital Storage Oscilloscope.

DONATE
to ArduinoBasics

LLN = Last Layer Neuron,
aO=actualOUTPUT,
dE=deltaError

HLN.dE = (HLN.aO) x (1-HLN.aO) x (Sum of [LLN.dE x LLN to HLN connection weight])

Keep moving back through the network layers until you reach the 1st layer (ie, you run out of layers).

Step 4: Update the weights of the connections and Bias of neuron.
a) Multiply the neuron's deltaError which was calculated in either step 2 or 3, by the learning rate (0.1), and by the connection's connEntry value.
b) Then add this calculated value (in Step (4a)) to the current weight of the connection.

```
neuron.connections[i].weight += (learningRate * neuron.connections[i].connEntry * neuron.deltaError);
```

The bias is like a connection with a constant connEntry of 1, therefore the calculation is

```
neuron.bias +=  (learningRate * 1 * neuron.deltaError);
```

Up Next: Neural Network (Part 6):
Back Propagation - a fully worked example.

To go back to the table of contents click here

Posted by Scott C at 23:27 G+1 Recommend this on Google

Labels: ArduinoBasics, Back propagation, best arduino blog, Learning, Neural Network, Neuron, Processing.org, Programming, project, tutorial

2 comments:

Anonymous 17 March 2012 at 18:10

Hi, I wonder that how do you denormalize the output values from 0 to 1? Thanks

Reply

Replies

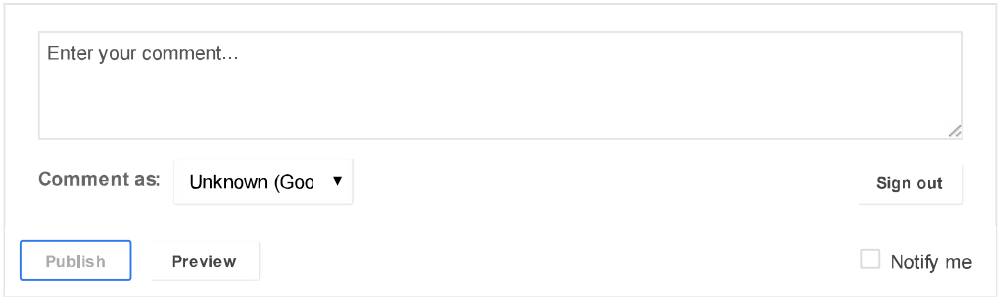
ScottC 26 March 2012 at 00:07

I am not sure what you mean. Please explain in more detail.

Reply

Total Pageviews
2,066,760

Subscribe
Posts
Comments



Feel free to leave a comment about this tutorial below.
Any questions about your particular project should be asked in the ArduinoBasics forum.

Comments are moderated due to large amount of spam.

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

G+1 6.7k

This Week's Most Popular Posts

433 MHz RF module with Arduino Tutorial 1

HC-SR04 Ultrasonic Sensor

Simple Arduino Serial Communication

Relay Module

433 MHz RF module with Arduino Tutorial 2

Most Recent Posts

Get Arduino Data over the internet using jQuery and AJAX

Two Million Views

Generosity Campaign Update - Day 3

Generosity Campaign Update - Day 2

Generosity campaign - Day 1

Recent Posts Widget by Helplogger

Awesome Inc. template. Powered by Blogger.

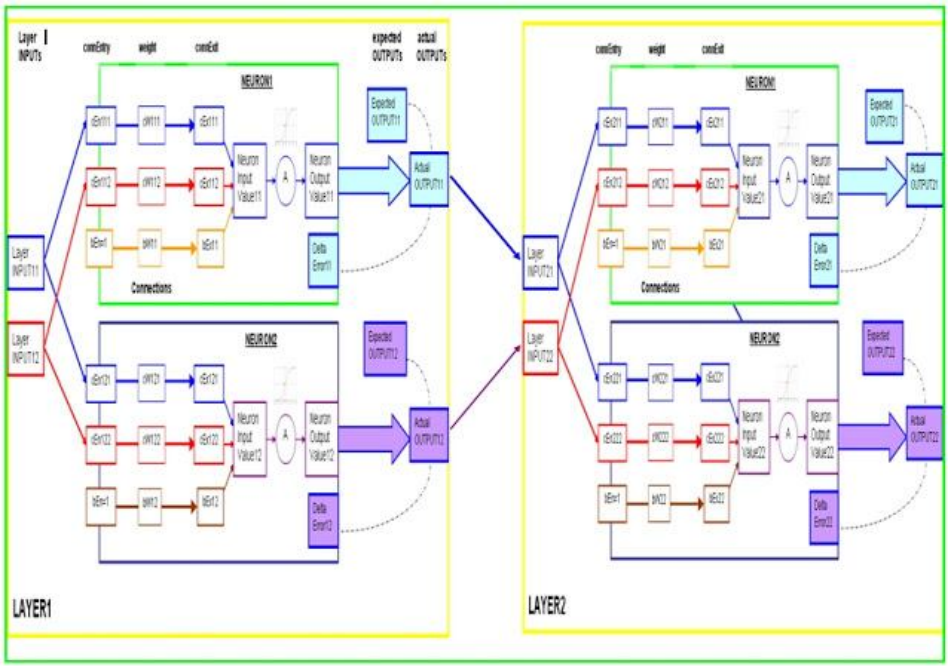
Arduino Basics

An Arduino tutorial blog. Free Arduino tutorials for everyone !

14 August 2011

Neural Network (Part 6) : Back Propagation, a worked example

A worked example of a Back-propagation training cycle.



In this example we will create a 2 layer network (as seen above), to accept 2 readings, and produce 2 outputs. The readings are (0,1) and the expectedOutputs in this example are (1,0).

Step 1: Create the network

```
NeuralNetwork NN = new NeuralNetwork();
NN.addLayer(2,2);
NN.addLayer(2,2);
float[] readings = {0,1};
float[] expectedOutputs = {1,0};
NN.trainNetwork(readings,expectedOutputs);
```

This neural network will have **randomised** weights and biases when created.
Let us assume that the network generates the following random variables:

LAYER1.Neuron1
Layer1.Neuron1.Connection1.weight = cW111 = 0.3
Layer1.Neuron1.Connection2.weight = cW112 = 0.8
Layer1.Neuron1.Bias = bW11 = 0.5

LAYER1.Neuron2

Search This Blog
Search

Translate
Select Language
Powered by Google Translate

Pages
Arduino Basics Projects Page
Forum
Arduino Basics YouTube Videos
Arduino Webserver Data Viewer
Money Jar

Connect
Follow @ArduinoBasics
YouTube 534
Follow me on
Instagram
Follow me on Periscope
ArduinoBasics

If you like this site, feel free to put a TIP into my money jar. It will be used to buy a Digital Storage Oscilloscope.

DONATE
to ArduinoBasics

Layer1.Neuron2.Connection1.weight = cW121 = 0.1
Layer1.Neuron2.Connection2.weight = cW122 = 0.1
Layer1.Neuron2.Bias = bW12 = 0.2

LAYER2.Neuron1
Layer2.Neuron1.Connection1.weight = cW211 = 0.6
Layer2.Neuron1.Connection2.weight = cW212 = 0.4
Layer2.Neuron1.Bias = bW21 = 0.4

LAYER2.Neuron2
Layer2.Neuron2.Connection1.weight = cW221 = 0.9
Layer2.Neuron2.Connection2.weight = cW222 = 0.9
Layer2.Neuron2.Bias = bW22 = 0.5

Step 2: Process the Readings through the Neural Network

a) Provide the Readings to the first layer, and calculate the neuron outputs

The readings provided to the neural network is (0,1), which go straight through to the first layer (layer1).
Starting with Layer 1:
Layer1.INPUT1 = 0
Layer1.INPUT2 =1

Calculate Layer1.Neuron1.NeuronOutput
ConnExit (cEx111) = ConnEntry (cEn111) x Weight (cW111) = 0 x 0.3 = 0;
ConnExit (cEx112) = ConnEntry (cEn112) x Weight (cW112) = 1 x 0.8 = 0.8;
Bias (bEx11) = ConnEntry (1) x Weight (bW11) = 1 x 0.4 = 0.4
NeuronInputValue11 = 0 + 0.8 + 0.4 = 1.2
NeuronOutputValue11 = 1/(1+EXP(-1 x 1.2)) = **0.768525**

Calculate Layer1.Neuron2.NeuronOutput
ConnExit (cEx121) = ConnEntry (cEn121) x Weight (cW121) = 0 x 0.1 = 0;
ConnExit (cEx122) = ConnEntry (cEn122) x Weight (cW122) = 1 x 0.1 = 0.1;
Bias (bEx12) = ConnEntry (1) x Weight (bW12) = 1 x 0.2 = 0.2
NeuronInputValue12 = 0 + 0.1 + 0.2 = 0.3
NeuronOutputValue12 = 1/(1+EXP(-1 x 0.3)) = **0.574443**

b) Provide LAYER2 with Layer 1 Outputs.

Now lets move to Layer 2:
Layer2.INPUT1 = NeuronOutputValue11 = **0.768525**
Layer2.INPUT2 = NeuronOutputValue12 = **0.574443**

Calculate Layer2.Neuron1.NeuronOutput
ConnExit (cEx211) = (cEn211) x Weight (cW211) = 0.768525 x 0.6 = 0.461115;
ConnExit (cEx212) = (cEn212) x Weight (cW212) = 0.574443 x 0.4 = 0.229777;
Bias (bEx21) = ConnEntry (1) x Weight (bW21) = 1 x 0.4 = 0.4
NeuronInputValue21 = 0.461115 + 0.229777 + 0.4 = 1.090892
NeuronOutputValue21 = 1/(1+EXP(-1 x 1.090892)) = **0.74855**

Calculate Layer2.Neuron2.NeuronOutput
ConnExit (cEx221) = (cEn221) x Weight (cW221) = 0.768525 x 0.1 = 0.076853;
ConnExit (cEx222) = (cEn222) x Weight (cW222) = 0.574443 x 0.1 = 0.057444;
Bias(bEx22) = ConnEntry (1) x Weight (bW22) = 1 x 0.5 = 0.5
NeuronInputValue22 = 0.076853 + 0.057444 + 0.5 = 0.634297
NeuronOutputValue22 = 1/(1+EXP(-1 x 0.634297)) = **0.653463**

Step 3) Calculate the delta error for neurons in layer 2

-Because layer 2 is the last layer in this neural network -
we will use the expected output data (1,0) to calculate the delta error.

LAYER2.Neuron1:
Let Layer2.ExpectedOutput1 = eO21 = 1

Total Pageviews
2,066,766

Subscribe
Posts
Comments

Layer2.ActualOutput1= aO21 = NeuronOutputValue21= **0.74855**
Layer2.Neuron1.deltaError1 = dE21

dE21 = aO21 x (1 - aO21) x (eO21 - aO21)
= (0.74855) x (1 - 0.74855) x (1 - 0.74855)
= (0.74855) x (0.25145) x (0.25145)
= 0.047329

LAYER2.Neuron2:
Let Layer2.ExpectedOutput2 = eO22 = 0
Layer2.ActualOutput2 = aO22 = NeuronOutputValue22 = **0.653463**
Layer2.Neuron2.deltaError = dE22

dE22 = aO22 x (1 - aO22) x (eO22 - aO22)
= (0.653463) x (1 - 0.653463) x (0 - 0.653463)
= (0.653463) x (0.346537) x (-0.653463)
= -0.14797

Step 4) Calculate the delta error for neurons in layer 1

LAYER1.Neuron1 delta Error calculation

Let Layer1.Neuron1.deltaError = dE11
Layer1.actualOutput1 = aO11 = NeuronOutputValue11 = **0.768525**
Layer2.Neuron1.Connection1.weight = cW211 = 0.6
Layer2.Neuron1.deltaError = dE21 = 0.047329
Layer2.Neuron2.Connection1.weight = cW221 = 0.9
Layer2.Neuron2.deltaError = dE22 = -0.14797

dE11 = (aO11) x (1 - aO11) x ([cW211 x dE21] + [cW221 x dE22])
= (**0.768525**) **x (1 - 0.768525) x ([0.6** x 0.047329] + [0.9 x -0.14797])
= **-0.01864**

LAYER1.Neuron2 delta Error calculation

Let Layer1.Neuron2.deltaError = dE12
Layer1.actualOutput2 = aO12 = NeuronOutputValue12 = **0.574443**
Layer2.Neuron1.Connection2.weight = cW212 = 0.4
Layer2.Neuron1.deltaError = dE21 = 0.047329
Layer2.Neuron2.Connection2.weight = cW222 = 0.9
Layer2.Neuron2.deltaError = dE22 = -0.14797

dE12 = (aO12) x (1 - aO12) x ([cW212 x dE21] + [cW222 x dE22])
= (**0.574443**) **x (1 - 0.574443) x ([**0.4 x 0.047329] + [0.9 x -0.14797])
= -0.02793

Step 5) Update Layer_2 neuron connection weights and bias (with a learning rate (LR) = 0.1)

Layer 2, Neuron 1 calculations:

Let
Layer2.Neuron1.Connection1.New_weight = New_cW211
Layer2.Neuron1.Connection1.Old_weight = Old_cW211 = 0.6
Layer2.Neuron1.Connection1.connEntry = cEn211 = 0.768525
Layer2.Neuron1.deltaError = dE21 = 0.047329

New_cW211 = Old_cW211 + (LR x cEn211 x dE21)
= 0.6 + (0.1 x 0.768525 x 0.047329)
= 0.6 + (0.003627)

= **0.603627**

Layer2.Neuron1.Connection2.New_weight = New_cW212
Layer2.Neuron1.Connection2.Old_weight = Old_cW212 = 0.4
Layer2.Neuron1.Connection2.connEntry = cEn212 = 0.574443
Layer2.Neuron1.deltaError = dE21 = 0.047329

New_cW212 = Old_cW212 + (LR x cEn212 x dE21)
= 0.4 + (0.1 x 0.574443 x 0.047329)
= 0.4 + (0.002719)
= **0.402719**

Layer2.Neuron1.New_Bias = New_Bias21
Layer2.Neuron1.Old_Bias = Old_Bias21 = 0.4
Layer2.Neuron1.deltaError = dE21 = 0.047329

New_Bias21 = Old_Bias21 + (LR x 1 x de21)
= 0.4 + (0.1 x 1 x 0.047329)
= 0.4 + (0.0047329)
= **0.4047329**

Layer 2, Neuron 2 calculations:

Layer2.Neuron2.Connection1.New_weight = New_cW221
Layer2.Neuron2.Connection1.Old_weight = Old_cW221 = 0.9
Layer2.Neuron2.Connection1.connEntry = cEn221 = 0.768525
Layer2.Neuron2.deltaError = dE22 = -0.14797

New_cW221 = Old_cW221 + (LR x cEn221 x dE22)
= 0.9 + (0.1 x 0.768525 x -0.14797)
= 0.9 + (-0.01137)
= **0.88863**

Layer2.Neuron2.Connection2.New_weight = New_cW222
Layer2.Neuron2.Connection2.Old_weight = Old_cW222 = 0.9
Layer2.Neuron2.Connection2.connEntry = cEn222 = 0.574443
Layer2.Neuron2.deltaError = dE22 = -0.14797

New_cW222 = Old_cW222 + (LR x cEn222 x dE22)
= 0.9 + (0.1 x 0.574443 x -0.14797)
= 0.9 + (-0.0085)
= **0.8915**

Layer2.Neuron2.New_Bias = New_Bias22
Layer2.Neuron2.Old_Bias = Old_Bias22 = 0.5
Layer2.Neuron2.deltaError = dE22 = -0.14797

New_Bias22 = Old_Bias22 + (LR x 1 x de22)
= 0.5 + (0.1 x 1 x -0.14797)
= 0.5 + (-0.014797)
= **0.485203**

Step 6) Update Layer_1 neuron connection weights and bias.

Layer 1, Neuron 1 calculations:

Let
Layer1.Neuron1.Connection1.New_weight = New_cW111
Layer1.Neuron1.Connection1.Old_weight = Old_cW111 = 0.3
Layer1.Neuron1.Connection1.connEntry = cEn111 = 0
Layer1.Neuron1.deltaError = dE11 = **-0.01864**

New_cW111 = Old_cW111 + (LR x cEn111 x dE11)
= 0.3 + (0.1 x 0 x **-0.01864**)
= 0.3 + (0)
= 0.3

Layer1.Neuron1.Connection2.New_weight = New_cW112
Layer1.Neuron1.Connection2.Old_weight = Old_cW112 = 0.8
Layer1.Neuron1.Connection2.connEntry = cEn112 = 1
Layer1.Neuron1.deltaError = dE11 = **-0.01864**

New_cW112 = Old_cW112 + (LR x cEn112 x dE11)
= 0.8 + (0.1 x 1 x **-0.01864**)
= 0.8 + (-0.001864)
= 0.798136

Layer1.Neuron1.New_Bias = New_Bias11
Layer1.Neuron1.Old_Bias = Old_Bias11 = 0.5
Layer1.Neuron1.deltaError = dE11 = **-0.01864**

New_Bias11 = Old_Bias11 + (LR x 1 x dE11)
= 0.5 + (0.1 x 1 x **-0.01864**)
= 0.5 + (-0.001864)
= 0.498136

Layer 1, Neuron 2 calculations:

Layer1.Neuron2.Connection1.New_weight = New_cW121
Layer1.Neuron2.Connection1.Old_weight = Old_cW121 = 0.1
Layer1.Neuron2.Connection1.connEntry = cEn121 = 0
Layer1.Neuron2.deltaError = dE12 = -0.02793

New_cW121 = Old_cW121 + (LR x cEn121 x dE12)
= 0.1 + (0.1 x 0 x -0.02793)
= 0.1 + (0)
= 0.1

Layer1.Neuron2.Connection2.New_weight = New_cW122
Layer1.Neuron2.Connection2.Old_weight = Old_cW122 = 0.1
Layer1.Neuron2.Connection2.connEntry = cEn122 = 1
Layer1.Neuron2.deltaError = dE12 = -0.02793

New_cW122 = Old_cW122 + (LR x cEn122 x dE12)
= 0.1 + (0.1 x 1 x -0.02793)
= 0.1 + (-0.002793)
= 0.097207

Layer1.Neuron2.New_Bias = New_Bias12
Layer1.Neuron2.Old_Bias = Old_Bias12 = 0.2
Layer1.Neuron2.deltaError = dE12 = -0.02793

New_Bias12 = Old_Bias12 + (LR x 1 x de12)
= 0.2 + (0.1 x 1 x -0.02793)
= 0.2 + (-0.002793)

= 0.197207

All done. That was just one training cycle. Thank goodness we have computers !
A computer can process these calculations really quickly, and depending on how complicated your neural network is (ie. number of layers, and number of neurons per layer), you may find that the training procedure may take some time. But believe me, if you have designed it right, it is well worth the wait.
Because once you have the desired weights and bias values set up, you are good to go, and as you receive data, the computer can do a single forward pass in a fraction of a second, and you will get your desired output, hopefully :)

Here is a complete Processing.org script that demonstrates the use of my neural network.
Neural Network (Part 7): Cut and Paste Code (click here).

If you liked my tutorial - please let me know in the comments. It is sometimes hard to know if anyone is actually reading this stuff. If you use my code in your own project, I am also happy for you to leave a link to a YouTube video etc in the comments also.

To go back to the table of contents click here

Posted by Scott C at 17:11 G+1 +1 Recommend this on Google

Labels: ArduinoBasics, Back propagation, best arduino blog, Feed Forward, Neural Network, Processing.org, project, Training, tutorial

19 comments:

Anonymous 8 September 2011 at 08:20

It's been nearly a month since this was posted, but this is exactly what I've been searching for. I'm a Cognitive Science student, and I'm trying to study neural networks a bit before I actually attend a formal lecture on them.

I made a duplicate of your code for python 3.1.2 just to see what I could do with this. This explanation is clear and concise, a step by step guide to building the foundation for a basic neural net.

Thank you! :)

Reply

ScottC 12 September 2011 at 22:43

It took me a month to get my head around this stuff, and as you can see, my neural net structure deviates slightly from the traditional feed-forward neural nets, however, the underlying equations are still there, and the flow of signals is pretty much the same.
Thankyou for the feedback !

Reply

Nathan Lindorff 12 November 2011 at 19:39

Thanks for posting this Scott! I'm doing Stanford's online machine learning course at the moment (ml-class.org) and was getting to the point where I was struggling to see what was happening and wanted a worked example. This was brilliant, and exactly what I needed. Thanks!

Reply

ScottC 13 November 2011 at 23:52

Hi Nathan,

I am glad you could make sense of my coding system. This stuff is not that easy to understand, and even harder to put into a "readable" format. Good luck with your course !

Reply

Siva Ganesh 19 December 2011 at 01:26

Great .Finally I got it.Thanks to publisher-Siva ganesh.GVP PG College,vizag.

Reply

Schteeve 18 April 2012 at 04:57

I've just come accross this and it's great! I studied ANN's a few years ago as part of my degree, and the lecturer took a damn long and confusing route to explain exactly what you have.

Thanks very much.

Reply

Replies

ScottC 18 April 2012 at 19:35

Thanks Schteeve

Miguel Navas 27 May 2012 at 23:25

thanks a lot

Reply

giovanni cosi 10 July 2012 at 20:05

very good thanks for this Guide. Sorry for my english.

I haven't red all this guide yet (I'm reading but I'm too curious) but I have a question: how many kByte a ANN could be. For example if I have 10 input and 1 output and only 1 hidden layer with up to 10 neurons, how can I calculate the memory wight in kByte?

Maybe it is a stupid question, but I have to buy a microcontroller, but I don't know ho many memory I will need.

Reply

Replies

ScottC 10 July 2012 at 21:49

Hi Giovanni,

In my examples, the ANN is not run on the Arduino, it is run on the computer using the processing language. I have not tried running this off of a microcontroller directly.
The Arduino sends the sensor data to the Computer (layer input), and the computer does all the hard work.
I'm sorry, I don't know how much memory it uses either.

Anonymous 12 November 2012 at 19:19

Great project and very useful, thank you.
I have a question,Here the readings are (0,1), so as written above
Starting with Layer 1:
Layer1.INPUT1 = 0
Layer1.INPUT2 =1

but Layer1.INPUT1 and Layer1.INPUT2 can be from A/D Converter from the microcontroller Arduino ? , for example :
Layer1.INPUT1 = 510
Layer1.INPUT2 =1000

thank you

ScottC 12 November 2012 at 21:18

Hi Anonymous,

The input can be as big or as small as you want it to be. The weights will adjust. And depending on what activation function you use, these values are generally transformed into numbers between 0 and 1. But one really good way of seeing this, is to substitute your values and see what happens. Time to get the calculator out :)
Also - have a look at Part 2 of this series - it may help explain it a bit more.

Reply

Anonymous 12 November 2012 at 19:15

Great project and very useful, thank you.
I have a question,Here the readings are (0,1), but it can be from ADC from the microcontroller Arduino ? , for example suppose two readings, the readings are (100,512). as you said : The Arduino sends the sensor data to the Computer (layer input), the sensor data is a float or integer ?

thank you

Reply

Replies

ScottC 12 November 2012 at 21:24

Hi Anonymous,
You can send a float or an integer (it depend on how you program it), but it is going to output a float - see Part 2 of this series, and take a look on how the activation function works.

In fact - I would recommend starting from Part 1 and working your way through to the end.. it might make more sense if you do it that way.

Reply

Anonymous 6 December 2012 at 23:00

hello, thank you very much for your tutorial, I read all your post about neural network, it helps me a lot, i made a program in vb6, just before implement it in a 32 bit microcontroller, but i have a question, I DONT UNDERSTAND very well the layers configuration, you have 2 inputs and 2 neurons to process it ?, or there are 2 inputs and 4 neurons(2 neurons in layer 1 and 2 neurons in layer 2) ?
respect the image at the top, you have 2 inputs for layer 1, can i have 6 inputs and only two neurons each layer (layer 1 and layer 2). Sorry I really dont understand the layers function, because if I increase the inputs I must increase the hidden layers. so please can you answer the questions. Thanks in advance.

Reply

Replies

Scott C 7 December 2012 at 00:24

Hi Anonymous,
You have asked a very good question. In the image at the top of this post, you will notice that there are 2 layers (LAYER1 and LAYER2).
In this specific example, there are 2 Layer inputs. These inputs receive data from the previous layer. However, there is no previous layer for LAYER1, so it by default becomes the neural network input.

Each Neural Network, will have layers, each layer will have neurons, and each neuron will have connections. The layer is mainly a container to group neurons, and also to hold values that will be transmitted to each of the neurons within. It also holds values that will be used to transmit information to the next layer in the neural network.

In this example, there are 2 layers, each layer has 2 neurons. So there are a total of 4 neurons in this Neural Network. But this was only done to simplify the example.

You can have 6 inputs into the neural network, and only have 2 neurons in Layer1. Please

take notice of the colours used in the image. If Layer1 had 6 inputs, then each neuron in layer 1 would need 6 connections in order to receive the signal from the layer inputs. Because Layer1 would only have 2 neurons, it will only have 2 outputs. There is one layer output per neuron.
Layer 2 would need to have 2 Inputs in order to receive the signal from Layer 1, but Layer2 could have 1000 neurons. Each neuron in Layer2 would have 2 connections to receive the signal from the layer2 inputs, and if Layer2 had 1000 neurons, then Layer2 would have 1000 outputs.

Did this help explain?
Try to follow the equations on paper... it is a whole lot easier.

Scott C 7 December 2012 at 00:28

http://arduinobasics.blogspot.com.au/2011/08/neural-network-part-3-layer.html

Reply

Anonymous 4 January 2015 at 08:04

You have an inconsistency in your work. In the lines
Calculate Layer2.Neuron2.NeuronOutput
ConnExit (cEx221) = (cEn221) x Weight (cW221) = 0.768525 x 0.1 = 0.076853;
ConnExit (cEx222) = (cEn222) x Weight (cW222) = 0.574443 x 0.1 = 0.057444;
You have cW221 and cW222 as 0.1, but you initialize them earlier to 0.9.

Reply

Replies

Scott C 4 January 2015 at 23:03

Well picked up... you get 10 points :)
It has taken over 3 years for anyone to notice that (including myself).
Despite the inconsistency, you should still be able to follow along.
I hope that was the only mistake :)

Reply

Enter your comment...

Comment as: Unknown (Goo ▾ Sign out

Publish Preview Notify me

Feel free to leave a comment about this tutorial below.
Any questions about your particular project should be asked in the ArduinoBasics forum.

Comments are moderated due to large amount of spam.

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

G+1 6.7k

This Week's Most Popular Posts

433 MHz RF module with Arduino Tutorial 1

HC-SR04 Ultrasonic Sensor

Simple Arduino Serial Communication

Relay Module

433 MHz RF module with Arduino Tutorial 2

Most Recent Posts

Get Arduino Data over the internet using jQuery and AJAX

Two Million Views

Generosity Campaign Update - Day 3

Generosity Campaign Update - Day 2

Generosity campaign - Day 1

Recent Posts Widget by Helplogger

Awesome Inc. template. Powered by Blogger.

G+1 2 আরো পরবর্তী ব্লগ» mdshs31@gmail.com ড্যাশবোর্ড সাইন আউট

Arduino Basics

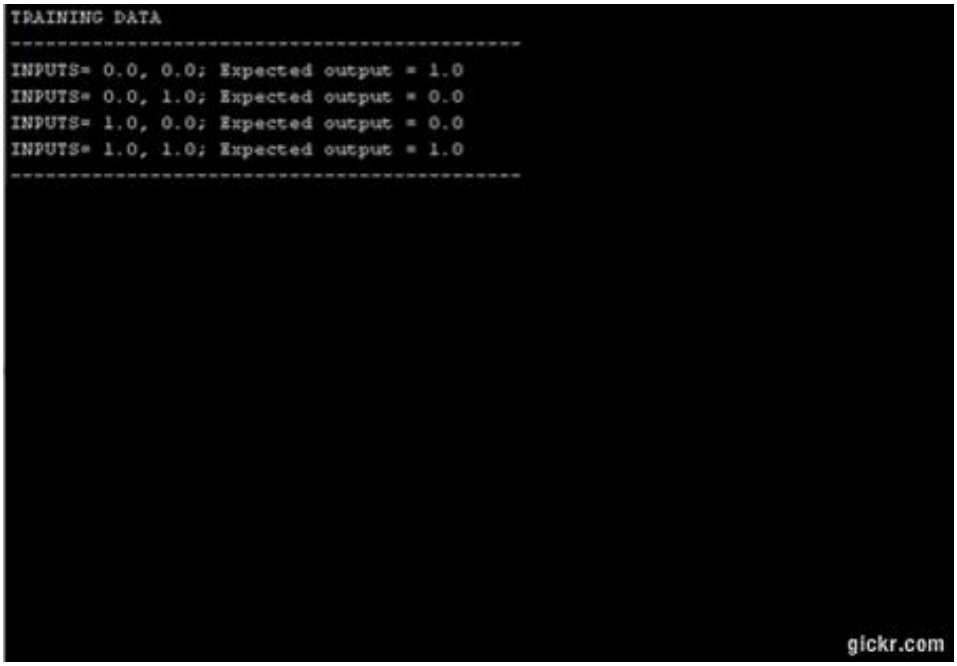
An Arduino tutorial blog. Free Arduino tutorials for everyone !

15 August 2011

Neural Network (Part 7) : Cut and Paste Code

Ok - so you don't like tutorials, and would rather just cut and paste some code.
This is for you.
http://www.openprocessing.org/visuals/?visualID=33991

Make sure to select "Source code" when you get there, otherwise it will be quite boring.
Here is an animated gif which shows the program in action.



See BELOW for the WHOLE screenshot so that you don't have to speed read.

Search This Blog
Search

Translate
Select Language
Powered by Google Translate

Pages
Arduino Basics Projects Page
Forum
Arduino Basics YouTube Videos
Arduino Webserver Data Viewer
Money Jar

Connect
Follow @ArduinoBasics
YouTube 534
Follow me on
Instagram
Follow me on Periscope
ArduinoBasics

If you like this site, feel free to put a TIP into my money jar. It will be used to buy a Digital Storage Oscilloscope.

DONATE
to ArduinoBasics

```
TRAINING DATA
-----------------------------------------
INPUTS= 0.0, 0.0; Expected output = 1.0
INPUTS= 0.0, 1.0; Expected output = 0.0
INPUTS= 1.0, 0.0; Expected output = 0.0
INPUTS= 1.0, 1.0; Expected output = 1.0
-----------------------------------------
Before Training
Feed Forward:  INPUT = 0,0; OUTPUT=0.3948359
Feed Forward:  INPUT = 0,1; OUTPUT=0.44980294
Feed Forward:  INPUT = 1,0; OUTPUT=0.36210358
Feed Forward:  INPUT = 1,1; OUTPUT=0.41312146

-----------------------------------------
Begin Training
CycleLimit has been reached. Has been retrained 0 times.  Error is = 0.026643272

End Training

-----------------------------------------
Test the neural network
Feed Forward:  INPUT = 0,0; OUTPUT=0.99297017
Feed Forward:  INPUT = 0,1; OUTPUT=0.0061992505
Feed Forward:  INPUT = 1,0; OUTPUT=0.0061998377
Feed Forward:  INPUT = 1,1; OUTPUT=0.9927852
```

If you want to know how it works, then you will have to go back and read part 1 to 6.

Neural Network

Part 1 : The Connection class

Part 2 : The Neuron class

Part 3 : The Layer class

Part 4 : The Neural Network class

Part 5: Back Propagation - the process

Part 6 : Back Propagation (A complete walk through)

But as you can see from the example above:
Before the neural network is trained, the outputs are not even close to the expected outputs. After training, the neural network produces the desired results (or very close to it).

Please note, this neural network also allows more than one output neuron, so you are not limited to single yes no decisions. You can use this neural network to make classifications. You will soon see this with my LED colour sensor.

Feel free to use this Neural Network in your own projects, but please let me know if you do, just for curiosity sake.

Update: See below for the Processing Sketch (much easier than going to the open processing site). It is a bit long - but saves you from having to navigate to another site.

Processing sketch

```
1 /*Neural Network created by ScottC on 15th Aug 2011
2
3 Please visit my blog for a detailed explanation of my Neural Network
4 http://arduinobasics.blogspot.com/p/arduinoprojects.html
```

Total Pageviews

2,066,773

Subscribe

Posts

Comments

```
 5
 6 */
 7
 8
 9
10 void setup(){
11   ArrayList myTrainingInputs = new ArrayList();
12   ArrayList myTrainingOutputs = new ArrayList();
13
14   float[] myInputsA={0,0};
15   float[] myInputsB={0,1};
16   float[] myInputsC={1,0};
17   float[] myInputsD={1,1};
18   float[] myOutputsA={1};
19   float[] myOutputsB={0};
20
21
22   println("TRAINING DATA");
23   println("-----------------------------------------");
24   myTrainingInputs.add(myInputsA);
25   myTrainingOutputs.add(myOutputsA);
26   println("INPUTS= " + myInputsA[0] + ", " + myInputsA[1] + "; Expected output =
27   myTrainingInputs.add(myInputsB);
28   myTrainingOutputs.add(myOutputsB);
29   println("INPUTS= " + myInputsB[0] + ", " + myInputsB[1] + "; Expected output =
30   myTrainingInputs.add(myInputsC);
31   myTrainingOutputs.add(myOutputsB);
32   println("INPUTS= " + myInputsC[0] + ", " + myInputsC[1] + "; Expected output =
33   myTrainingInputs.add(myInputsD);
34   myTrainingOutputs.add(myOutputsA);
35   println("INPUTS= " + myInputsD[0] + ", " + myInputsD[1] + "; Expected output =
36   println("-----------------------------------------");
37
38   NeuralNetwork NN = new NeuralNetwork();
39   NN.addLayer(2,2);
40   NN.addLayer(2,1);
41
42   println("Before Training");
43   float[] myInputDataA1={0,0};
44   NN.processInputsToOutputs(myInputDataA1);
45   float[] myOutputDataA1={};
46   myOutputDataA1=NN.getOutputs();
47   println("Feed Forward:  INPUT = 0,0; OUTPUT=" + myOutputDataA1[0]);
48
49   float[] myInputDataB1={0,1};
50   NN.processInputsToOutputs(myInputDataB1);
51   float[] myOutputDataB1={};
52   myOutputDataB1=NN.getOutputs();
53   println("Feed Forward:  INPUT = 0,1; OUTPUT=" + myOutputDataB1[0]);
54
55   float[] myInputDataC1={1,0};
56   NN.processInputsToOutputs(myInputDataC1);
57   float[] myOutputDataC1={};
58   myOutputDataC1=NN.getOutputs();
59   println("Feed Forward:  INPUT = 1,0; OUTPUT=" + myOutputDataC1[0]);
60
61   float[] myInputDataD1={1,1};
62   NN.processInputsToOutputs(myInputDataD1);
63   float[] myOutputDataD1={};
64   myOutputDataD1=NN.getOutputs();
65   println("Feed Forward:  INPUT = 1,1; OUTPUT=" + myOutputDataD1[0]);
66
67   println("");
68   println("-----------------------------------------");
69
70   println("Begin Training");
71   NN.autoTrainNetwork(myTrainingInputs,myTrainingOutputs,0.0001,500000);
72   println("");
73   println("End Training");
74   println("");
75   println("-----------------------------------------");
76   println("Test the neural network");
77   float[] myInputDataA2={0,0};
78   NN.processInputsToOutputs(myInputDataA2);
79   float[] myOutputDataA2={};
80   myOutputDataA2=NN.getOutputs();
81   println("Feed Forward:  INPUT = 0,0; OUTPUT=" + myOutputDataA2[0]);
```

```
 82
 83   float[] myInputDataB2={0,1};
 84   NN.processInputsToOutputs(myInputDataB2);
 85   float[] myOutputDataB2={};
 86   myOutputDataB2=NN.getOutputs();
 87   println("Feed Forward:  INPUT = 0,1; OUTPUT=" + myOutputDataB2[0]);
 88
 89   float[] myInputDataC2={1,0};
 90   NN.processInputsToOutputs(myInputDataC2);
 91   float[] myOutputDataC2={};
 92   myOutputDataC2=NN.getOutputs();
 93   println("Feed Forward:  INPUT = 1,0; OUTPUT=" + myOutputDataC2[0]);
 94
 95   float[] myInputDataD2={1,1};
 96   NN.processInputsToOutputs(myInputDataD2);
 97   float[] myOutputDataD2={};
 98   myOutputDataD2=NN.getOutputs();
 99   println("Feed Forward:  INPUT = 1,1; OUTPUT=" + myOutputDataD2[0]);
100
101
102 }
103
104
105 /* ---------------------------------------------------------------------
106 A connection determines how much of a signal is passed through to the neuron.
107 ---------------------------------------------------------------------  */
108
109 class Connection{
110   float connEntry;
111   float weight;
112   float connExit;
113
114   //This is the default constructor for an Connection
115   Connection(){
116     randomiseWeight();
117   }
118
119   //A custom weight for this Connection constructor
120   Connection(float tempWeight){
121     setWeight(tempWeight);
122   }
123
124   //Function to set the weight of this connection
125   void setWeight(float tempWeight){
126     weight=tempWeight;
127   }
128
129   //Function to randomise the weight of this connection
130   void randomiseWeight(){
131     setWeight(random(2)-1);
132   }
133
134   //Function to calculate and store the output of this Connection
135   float calcConnExit(float tempInput){
136     connEntry = tempInput;
137     connExit = connEntry * weight;
138     return connExit;
139   }
140 }
141
142
143 /* ---------------------------------------------------------------------
144    A neuron does all the processing and calculation to convert an input into an
145 ---------------------------------------------------------------------- */
146
147 class Neuron{
148   Connection[] connections={};
149   float bias;
150   float neuronInputValue;
151   float neuronOutputValue;
152   float deltaError;
153
154   //The default constructor for a Neuron
155   Neuron(){
156   }
157
158   /*The typical constructor of a Neuron - with random Bias and Connection weight
```

```
159   Neuron(int numOfConnections){
160     randomiseBias();
161     for(int i=0; i<numOfConnections; i++){
162       Connection conn = new Connection();
163       addConnection(conn);
164     }
165   }
166
167   //Function to add a Connection to this neuron
168   void addConnection(Connection conn){
169       connections = (Connection[]) append(connections, conn);
170   }
171
172   /* Function to return the number of connections associated with this neuron.*/
173   int getConnectionCount(){
174       return connections.length;
175   }
176
177   //Function to set the bias of this Neron
178   void setBias(float tempBias){
179     bias = tempBias;
180   }
181
182   //Function to randomise the bias of this Neuron
183   void randomiseBias(){
184     setBias(random(1));
185   }
186
187   /*Function to convert the inputValue to an outputValue
188   Make sure that the number of connEntryValues matches the number of connections
189
190   float getNeuronOutput(float[] connEntryValues){
191     if(connEntryValues.length!=getConnectionCount()){
192       println("Neuron Error: getNeuronOutput() : Wrong number of connEntryValues
193       exit();
194     }
195
196     neuronInputValue=0;
197
198     /* First SUM all of the weighted connection values (connExit) attached to th
199     for(int i=0; i<getConnectionCount(); i++){
200       neuronInputValue+=connections[i].calcConnExit(connEntryValues[i]);
201     }
202
203     //Add the bias to the Neuron's inputValue
204     neuronInputValue+=bias;
205
206     /* Send the inputValue through the activation function to produce the Neuron
207     neuronOutputValue=Activation(neuronInputValue);
208
209     //Return the outputValue
210     return neuronOutputValue;
211   }
212
213   //Activation function
214   float Activation(float x){
215     float activatedValue = 1 / (1 + exp(-1 * x));
216     return activatedValue;
217   }
218
219 }
220
221 class Layer{
222   Neuron[] neurons = {};
223   float[] layerINPUTs={};
224   float[] actualOUTPUTs={};
225   float[] expectedOUTPUTs={};
226   float layerError;
227   float learningRate;
228
229
230   /* This is the default constructor for the Layer */
231   Layer(int numberConnections, int numberNeurons){
232     /* Add all the neurons and actualOUTPUTs to the layer */
233     for(int i=0; i<numberNeurons; i++){
234       Neuron tempNeuron = new Neuron(numberConnections);
235       addNeuron(tempNeuron);
```

```
236       addActualOUTPUT();
237     }
238   }
239
240
241   /* Function to add an input or output Neuron to this Layer */
242   void addNeuron(Neuron xNeuron){
243         neurons = (Neuron[]) append(neurons, xNeuron);
244   }
245
246
247   /* Function to get the number of neurons in this layer */
248   int getNeuronCount(){
249     return neurons.length;
250   }
251
252
253   /* Function to increment the size of the actualOUTPUTs array by one. */
254   void addActualOUTPUT(){
255       actualOUTPUTs = (float[]) expand(actualOUTPUTs,(actualOUTPUTs.length+1));
256   }
257
258
259   /* Function to set the ENTIRE expectedOUTPUTs array in one go. */
260   void setExpectedOUTPUTs(float[] tempExpectedOUTPUTs){
261     expectedOUTPUTs=tempExpectedOUTPUTs;
262   }
263
264
265   /* Function to clear ALL values from the expectedOUTPUTs array */
266   void clearExpectedOUTPUT(){
267        expectedOUTPUTs = (float[]) expand(expectedOUTPUTs, 0);
268   }
269
270
271   /* Function to set the learning rate of the layer */
272   void setLearningRate(float tempLearningRate){
273     learningRate=tempLearningRate;
274   }
275
276
277   /* Function to set the inputs of this layer */
278   void setInputs(float[] tempInputs){
279     layerINPUTs=tempInputs;
280   }
281
282
283
284   /* Function to convert ALL the Neuron input values into Neuron output values i
285   void processInputsToOutputs(){
286
287     /* neuronCount is used a couple of times in this function. */
288     int neuronCount = getNeuronCount();
289
290     /* Check to make sure that there are neurons in this layer to process the in
291     if(neuronCount>0) {
292       /* Check to make sure that the number of inputs matches the number of Neur
293       if(layerINPUTs.length!=neurons[0].getConnectionCount()){
294         println("Error in Layer: processInputsToOutputs: The number of inputs do
295         exit();
296       } else {
297         /* The number of inputs are fine : continue
298            Calculate the actualOUTPUT of each neuron in this layer,
299            based on their layerINPUTs (which were previously calculated).
300            Add the value to the layer's actualOUTPUTs array. */
301         for(int i=0; i<neuronCount;i++){
302           actualOUTPUTs[i]=neurons[i].getNeuronOutput(layerINPUTs);
303         }
304       }
305     }else{
306       println("Error in Layer: processInputsToOutputs: There are no Neurons in t
307       exit();
308     }
309   }
310
311
312   /* Function to get the error of this layer */
```

```
313   float getLayerError(){
314     return layerError;
315   }
316
317
318   /* Function to set the error of this layer */
319   void setLayerError(float tempLayerError){
320     layerError=tempLayerError;
321   }
322
323
324   /* Function to increase the layerError by a certain amount */
325   void increaseLayerErrorBy(float tempLayerError){
326     layerError+=tempLayerError;
327   }
328
329
330   /* Function to calculate and set the deltaError of each neuron in the layer */
331   void setDeltaError(float[] expectedOutputData){
332     setExpectedOUTPUTs(expectedOutputData);
333     int neuronCount = getNeuronCount();
334     /* Reset the layer error to 0 before cycling through each neuron */
335     setLayerError(0);
336      for(int i=0; i<neuronCount;i++){
337       neurons[i].deltaError = actualOUTPUTs[i]*(1-actualOUTPUTs[i])*(expectedOU
338
339       /* Increase the layer Error by the absolute difference between the calcul
340       increaseLayerErrorBy(abs(expectedOUTPUTs[i]-actualOUTPUTs[i]));
341      }
342   }
343
344
345   /* Function to train the layer : which uses a training set to adjust the conne
346   void trainLayer(float tempLearningRate){
347     setLearningRate(tempLearningRate);
348
349     int neuronCount = getNeuronCount();
350
351     for(int i=0; i<neuronCount;i++){
352       /* update the bias for neuron[i] */
353       neurons[i].bias += (learningRate * 1 * neurons[i].deltaError);
354
355       /* update the weight of each connection for this neuron[i] */
356       for(int j=0; j<neurons[i].getConnectionCount(); j++){
357           neurons[i].connections[j].weight += (learningRate * neurons[i].connect
358       }
359     }
360   }
361 }
362
363 /* ------------------------------------------------------------
364    The Neural Network class is a container to hold and manage all the layers
365    ------------------------------------------------------------------ */
366
367 class NeuralNetwork{
368   Layer[] layers = {};
369   float[] arrayOfInputs={};
370   float[] arrayOfOutputs={};
371   float learningRate;
372   float networkError;
373   float trainingError;
374   int retrainChances=0;
375
376   NeuralNetwork(){
377     /* the default learning rate of a neural network is set to 0.1, which can ch
378     learningRate=0.1;
379   }
380
381
382
383   /* Function to add a Layer to the Neural Network */
384   void addLayer(int numConnections, int numNeurons){
385     layers = (Layer[]) append(layers, new Layer(numConnections,numNeurons));
386   }
387
388
389
```

```
390  /* Function to return the number of layers in the neural network */
391  int getLayerCount(){
392      return layers.length;
393  }
394
395
396
397  /* Function to set the learningRate of the Neural Network */
398  void setLearningRate(float tempLearningRate){
399    learningRate=tempLearningRate;
400  }
401
402
403
404  /* Function to set the inputs of the neural network */
405  void setInputs(float[] tempInputs){
406    arrayOfInputs=tempInputs;
407  }
408
409
410
411  /* Function to set the inputs of a specified layer */
412  void setLayerInputs(float[] tempInputs, int layerIndex){
413    if(layerIndex>getLayerCount()-1){
414      println("NN Error: setLayerInputs: layerIndex=" + layerIndex + " exceeded
415    } else {
416      layers[layerIndex].setInputs(tempInputs);
417    }
418  }
419
420
421
422  /* Function to set the outputs of the neural network */
423  void setOutputs(float[] tempOutputs){
424    arrayOfOutputs=tempOutputs;
425  }
426
427
428
429  /* Function to return the outputs of the Neural Network */
430  float[] getOutputs(){
431    return arrayOfOutputs;
432  }
433
434
435
436  /* Function to process the Neural Network's input values and convert them to a
437  void processInputsToOutputs(float[] tempInputs){
438    setInputs(tempInputs);
439
440    /* Check to make sure that the number of NeuralNetwork inputs matches the Ne
441    if(getLayerCount()>0){
442      if(arrayOfInputs.length!=layers[0].neurons[0].getConnectionCount()){
443        println("NN Error: processInputsToOutputs: The number of inputs do NOT m
444        exit();
445      } else {
446        /* The number of inputs are fine : continue */
447        for(int i=0; i<getLayerCount(); i++){
448
449          /*Set the INPUTs for each layer: The first layer gets it's input data
450          if(i==0){
451            setLayerInputs(arrayOfInputs,i);
452          } else {
453            setLayerInputs(layers[i-1].actualOUTPUTs, i);
454          }
455
456          /* Now that the layer has had it's input values set, it can now proces
457          layers[i].processInputsToOutputs();
458        }
459        /* Once all the data has filtered through to the end of network, we can
460           These values become or will be set to the NN output values (arrayOfOu
461        setOutputs(layers[getLayerCount()-1].actualOUTPUTs);
462      }
463    }else{
464      println("Error: There are no layers in this Neural Network");
465      exit();
466    }
```

```
467   }
468
469
470
471
472   /* Function to train the entire network using an array. */
473   void trainNetwork(float[] inputData, float[] expectedOutputData){
474     /* Populate the ENTIRE network by processing the inputData. */
475     processInputsToOutputs(inputData);
476
477     /* train each layer - from back to front (back propagation) */
478     for(int i=getLayerCount()-1; i>-1; i--){
479       if(i==getLayerCount()-1){
480         layers[i].setDeltaError(expectedOutputData);
481         layers[i].trainLayer(learningRate);
482         networkError=layers[i].getLayerError();
483       } else {
484         /* Calculate the expected value for each neuron in this layer (eg. HIDDE
485         for(int j=0; j<layers[i].getNeuronCount(); j++){
486           /* Reset the delta error of this neuron to zero. */
487           layers[i].neurons[j].deltaError=0;
488           /* The delta error of a hidden layer neuron is equal to the SUM of [th
489           /* Connection#1 of each neuron in the output layer connect with Neuron
490           for(int k=0; k<layers[i+1].getNeuronCount(); k++){
491             layers[i].neurons[j].deltaError += (layers[i+1].neurons[k].connectio
492           }
493           /* Now that we have the sum of Errors x weights attached to this neuro
494           layers[i].neurons[j].deltaError *= (layers[i].neurons[j].neuronOutputV
495         }
496         /* Now that you have all the necessary fields populated, you can now Tra
497         layers[i].trainLayer(learningRate);
498         layers[i].clearExpectedOUTPUT();
499       }
500     }
501   }
502
503
504
505
506
507   /* Function to train the entire network, using an array of input and expected
508   void trainingCycle(ArrayList trainingInputData, ArrayList trainingExpectedData
509       int dataIndex;
510
511       /* re-initialise the training Error with every cycle */
512       trainingError=0;
513
514       /* Cycle through the training data either randomly or sequentially */
515       for(int i=0; i<trainingInputData.size(); i++){
516         if(trainRandomly){
517           dataIndex=(int) (random(trainingInputData.size()));
518         } else {
519           dataIndex=i;
520         }
521
522         trainNetwork((float[]) trainingInputData.get(dataIndex),(float[]) traini
523
524         /* Use the networkError variable which is calculated at the end of each
525         trainingError+=abs(networkError);
526       }
527   }
528
529
530
531
532
533   /* Function to train the network until the Error is below a specific threshold
534   void autoTrainNetwork(ArrayList trainingInputData, ArrayList trainingExpectedD
535     trainingError=9999;
536     int trainingCounter=0;
537
538
539     /* cycle through the training data until the trainingError gets below traini
540
541 variable (eg. 10000). */
542     while(trainingError>trainingErrorTarget && trainingCounter<cycleLimit){
543
```

```
544       /* re-initialise the training Error with every cycle */
545       trainingError=0;
546
547       /* Cycle through the training data randomly */
548       trainingCycle(trainingInputData, trainingExpectedData, true);
549
550       /* increment the training counter to prevent endless loop */
551       trainingCounter++;
552     }
553
554     /* Due to the random nature in which this neural network is trained. There m
555        To check if this is the case, we will go through one more cycle (but sequ
556        If the training error is still below the trainingErrorTarget, then we wil
557        If the training error is above the trainingErrorTarget, we will continue
558     if(trainingCounter<cycleLimit){
559        trainingCycle(trainingInputData, trainingExpectedData, false);
560        trainingCounter++;
561
562        if(trainingError>trainingErrorTarget){
563          if (retrainChances<10){
564            retrainChances++;
565            autoTrainNetwork(trainingInputData, trainingExpectedData,trainingErro
566          }
567        }
568
569     } else {
570       println("CycleLimit has been reached. Has been retrained " + retrainChance
571     }
572   }
573 }
```

The above code was formatted using hilite.me

Posted by Scott C at 21:06

G+1 +2 Recommend this on Google

Labels: ArduinoBasics, Back propagation, best arduino blog, code, Example, Neural Network, Processing, Processing.org, project, tutorial

16 comments:

Anonymous 25 August 2011 at 14:55

Very cool set of tutorials. Really liked it.

Reply

Anonymous 23 October 2011 at 13:46

This is an amazing tutorial for explaining how to build a neural network! Just beginning to learn how to work with these, and though a bit too complex sometimes for a novice like myself, there's plenty of depth to keep me busy for quite a while. Thank you.

Reply

ScottC 30 October 2011 at 01:30

Build a neural network, how hard can it be ? It is only until you try to build one yourself that you appreciate how complex it can get. But don't give up, it is worth the trouble.

Reply

cr0sh 15 November 2011 at 11:27

This is a great code base; thanks for sharing it. I too am taking the Stanford ML class (finished the backprop homework assignment last night - and going to be watching the first video for this week's stuff tonight); this code seems to follow the general stuff (I looked at the cut-n-paste code, and I don't think there was anything like regularization or cost functions included). I would say this makes a great base anyhow; I would encourage anyone who wants to know more to check out Stanford's online class...

Reply

Anonymous 10 January 2012 at 19:03

This is a very interesting project!
However,the command "ArrayList" is undefined in my Arduino program. Would you please explain to me? Thank you very much!

Reply

ScottC 11 January 2012 at 07:49

You need to paste the code into a "Processing" project, and not into the Arduino IDE. The processing program looks very similar to the Arduino program.
To download processing, point your browser to http://processing.org/download/

Reply

Anonymous 13 January 2012 at 08:30

Thanks for replying!
I thought this is Arduino project, so can "Processing" program works on the Arduino board.
And is there a way that I can change the "ArrayList", so I can use it in the Arduino program?
Thank you very much!

Reply

Replies

Anonymous 13 January 2012 at 17:04

Just got it, thanks!

ScottC 17 January 2012 at 23:51

The Neural Network tutorial is a Processing program that is run on your computer. This particular program has absolutely nothing to do with an arduino. However, if you have a look at

The poor man's colour detector project:

http://arduinobasics.blogspot.com/2011/08/poor-mans-colour-detector-part-2.html

You will see an example of how this neural network can be used in combination with your arduino to "detect colour" using a few LEDs.

Reply

Anonymous 31 July 2012 at 09:18

Hi,
I think your blog is great.
Currently I'm trying to write my own NN...
I was looking for your code in the 'open processing' website, but it seems like it's not working anymore (it's impossible to switch between tabs).
Is there any other source I can take the code from?
Thanks

Reply

Replies

ScottC 1 August 2012 at 00:27

Thanks Anonymous,
I went to the open processing site - and did not have a problem, but have decided to include the code in this post because I am now using a more advanced code highlighter (which was not available when I first wrote it). Hope this helps.

Reply

Anonymous 1 August 2012 at 19:41

Thanks a lot!!!, that's great!

Reply

Guille 7 August 2013 at 22:17

excelent tutorial, there are a lot of book, paper, technical report of ANN but dont have the way to implemented!! most of we do implementation seek this type of information, thank you very much for sharing! regards

Reply

Replies

Scott C 8 August 2013 at 06:35

No problem Guille,
I am glad it helped.

Regards Scott

Reply

Anonymous 10 November 2015 at 16:26

Thanks very much for sharing! I'm just learning NN for my final year project, really helpful.

Reply

Replies

Scott C 10 November 2015 at 18:37

It has been a while since I wrote it... but glad that it is still relevant.
Thank you for the feedback.

Regards
Scott

Reply

Enter your comment...

Comment as: Unknown (Goc ▼ Sign out

Publish Preview Notify me

Feel free to leave a comment about this tutorial below.
Any questions about your particular project should be asked in the ArduinoBasics forum.

Comments are moderated due to large amount of spam.

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

G+1 6.7k

This Week's Most Popular Posts

433 MHz RF module with Arduino Tutorial 1

HC-SR04 Ultrasonic Sensor

Simple Arduino Serial Communication

Relay Module

433 MHz RF module with Arduino Tutorial 2

Most Recent Posts

Get Arduino Data over the internet using jQuery and AJAX

Two Million Views

Generosity Campaign Update - Day 3

Generosity Campaign Update - Day 2

Generosity campaign - Day 1

Recent Posts Widget by Helplogger

Awesome Inc. template. Powered by Blogger.