Arduino Basics

An Arduino tutorial blog. Free Arduino tutorials for everyone !

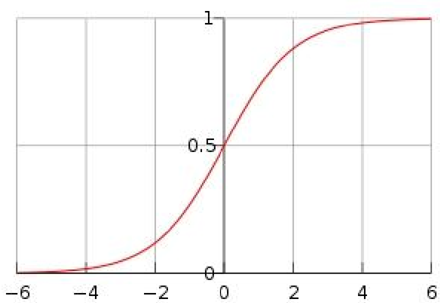
12 August 2011

Neural Network (Part 2) : The Neuron

The Neuron

The main purpose of the Neuron is to store the values that flow through the neural network. They also do a bit of processing through the use of an Activation function. The activation function is quite an important feature of the neural network. It essentially enables the neuron to make decisions (yes/no and greyzone) based on the values provided to them. The other feature of activation functions is the ability to map values of infinite size to a range of 0 to 1. We will be using the Sigmoid function.

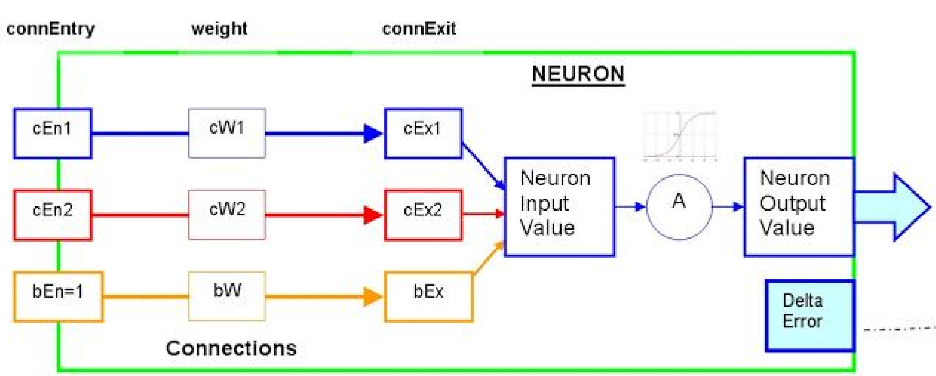
$$P(t) = \frac{1}{1 + e^{-t}}.$$



The above pictures can be found at Wikipedia

The neuron accumulates signals from one or more connections, adds a bias value, and then sends this value through the activation function to produce a neuron output. Each neuron output will be within the range of 0 to 1 (due to the activation function).

So lets take a look at a single neuron with 2 connections (+ bias) feeding into it.
As seen below.



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

connEntry
Neuron.Connection1.connEntry = **cEn1**
Neuron.Connection2.connEntry = **cEn2**
Neuron.Bias.connEntry = **bEn = 1** (always)

weight
Neuron.Connection1.weight = **cW1**
Neuron.Connection2.weight = **cW2**
Neuron.Bias.weight = **bW**

connExit
Neuron.Connection1.connExit = **cEx1**
Neuron.Connection2.connExit = **cEx2**
Neuron.Bias.connExit = **bEx**

You need to multiply the connEntry value by the weight to get the connExit value.

Connection #1:
cEx1 = cW1 x cEn1

Connection # 2:
cEx2 = cW2 x cEn2

Bias:
bEx = bW x 1

As you can see from the Bias calculation, the Bias.connEntry (bEn) value is always 1, which means that the Bias.connExit (bEx) value is always equal to the Bias.weight (bW) value.

The Neuron Input Value is equal to the sum of all the connExit values (cEx1 and cEx2 in this case) plus the bias (bEx), and can be represented by the following formula.

Neuron Input Value:

Neuron Input Value = cEx1 + cEx2 + bEx
or
Neuron Input Value = [cW1 x cEn1] + [cW2 x cEn2] + [bW x 1]

Now that we have the Neuron Input Value, we can put this through the Activation Function to get the Neuron Output Value. This can be represented by the following formula.

Neuron Output Value:

Neuron Output Value = 1 / (1 + EXP(-1 x Neuron Input Value))

We will use this Neuron Output value as an input to the next layer, therefore it automatically becomes a connEntry for one of the neuron connections in the next layer.

Now that we know what a neuron does, let us create a neuron class. See the code below.

processing code Neuron Class

```
/* ------------------------------------------------------------------------
   A neuron does all the calculations to convert an input to an output
------------------------------------------------------------------------  */

class Neuron{
  Connection[] connections={};
  float bias;
  float neuronInputValue;
  float neuronOutputValue;
```

Total Pageviews

2,066,848

Subscribe

Posts
Comments

```
  float deltaError;

  //The default constructor for a Neuron
  Neuron(){
  }

  //The typical constructor of a Neuron - with random Bias and Connection weights
  Neuron(int numOfConnections){
    randomiseBias();
    for(int i=0; i<numOfConnections; i++){
      Connection conn = new Connection();
      addConnection(conn);
    }
  }

  //Function to add a Connection to this neuron
  void addConnection(Connection conn){
      connections = (Connection[]) append(connections, conn);
  }

  //Function to return the number of connections associated with this neuron.
  int getConnectionCount(){
      return connections.length;
  }

  //Function to set the bias of this Neuron
  void setBias(float tempBias){
    bias = tempBias;
  }

  //Function to randomise the bias of this Neuron
  void randomiseBias(){
    setBias(random(1));
  }

  //Function to convert the neuronInputValue to an neuronOutputValue
  //Make sure that the number of connEntryValues matches the number of connections

  float getNeuronOutput(float[] connEntryValues){
    if(connEntryValues.length!=getConnectionCount()){
      println("Neuron Error: getNeuronOutput() : Wrong number of connEntryValues");
      exit();
    }

    neuronInputValue=0;

    //First SUM all of the weighted connection values (connExit) attached to this neuron.
    for(int i=0; i<getConnectionCount(); i++){
      neuronInputValue+=connections[i].calcConnExit(connEntryValues[i]);
    }

    //Add the bias to the Neuron's inputValue
    neuronInputValue+=bias;

    //Send the inputValue through the activation function to produce the Neuron's outputValue
    neuronOutputValue=Activation(neuronInputValue);

    //Return the outputValue
    return neuronOutputValue;
  }

  //Sigmoid Activation function
  float Activation(float x){
    float activatedValue = 1 / (1 + exp(-1 * x));
    return activatedValue;
  }

}
```

code formatter

When you create a new Neuron, you have the option to create the basic neuron shell with the Neuron() constructor, however, you are more likely to call the 2nd constructor, which allows you to set the number of input connections attached to this neuron.
For example, if I create a neuron using New Neuron(4), then this will automatically

attach and associate four connections with this neuron randomise the weight of each connection

randomise the bias of this neuron

These are the functions within the Neuron Class:

addConnection(): adds a new connection to this neuron

getConnectionCount(): returns the number of connections associated with this neuron

setBias(): sets the Bias of the neuron to a specific value.

randomiseBias() : randomises the Bias of this neuron

getNeuronOutput() : values are fed through the neuron's connections to create the neuronInputValue, which is then sent through the Sigmoid Activation function to create the neuronOutputValue.

Activation() : is the function used by the getNeuronOutput() function to generate the neuronOutputValue.

Up next: Neural Network (Part 3) : The Layer

To go back to the table of contents click here

Posted by Scott C at 00:58 G+1 Recommend this on Google

Labels: Activation function, ArduinoBasics, best arduino blog, Bias, code, Neural Network, Neuron, Processing.org, project, Sigmoid, tutorial, Weight

No comments:

Post a Comment

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.

Enter your comment...

Comment as: Unknown (Goc Sign out

Publish Preview Notify me

Links to this post

Create a Link

Arduino Basics

An Arduino tutorial blog. Free Arduino tutorials for everyone !

Home | Arduino Basics Projects Page | Forum | Contact Author | Money Jar

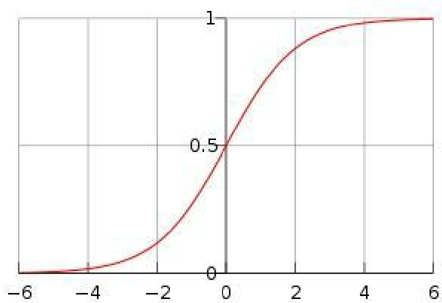
12 August 2011

Neural Network (Part 2) : The Neuron

The Neuron

The main purpose of the Neuron is to store the values that flow through the neural network. They also do a bit of processing through the use of an Activation function. The activation function is quite an important feature of the neural network. It essentially enables the neuron to make decisions (yes/no and greyzone) based on the values provided to them. The other feature of activation functions is the ability to map values of infinite size to a range of 0 to 1. We will be using the Sigmoid function.

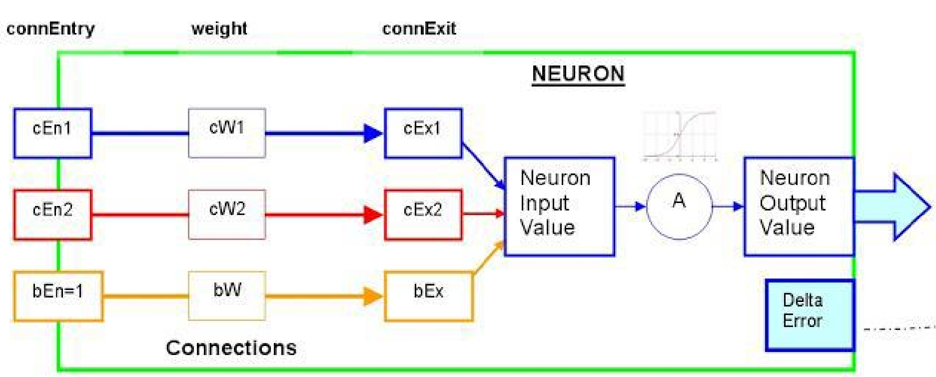
$$P(t) = \frac{1}{1+e^{-t}}.$$



The above pictures can be found at Wikipedia

The neuron accumulates signals from one or more connections, adds a bias value, and then sends this value through the activation function to produce a neuron output. Each neuron output will be within the range of 0 to 1 (due to the activation function).

So lets take a look at a single neuron with 2 connections (+ bias) feeding into it.
As seen below.



connEntry
Neuron.Connection1.connEntry = **cEn1**
Neuron.Connection2.connEntry = **cEn2**
Neuron.Bias.connEntry = **bEn = 1** (always)

weight
Neuron.Connection1.weight = **cW1**
Neuron.Connection2.weight = **cW2**
Neuron.Bias.weight = **bW**

connExit
Neuron.Connection1.connExit = **cEx1**
Neuron.Connection2.connExit = **cEx2**
Neuron.Bias.connExit = **bEx**

You need to multiply the connEntry value by the weight to get the connExit value.

Connection #1:
cEx1 = cW1 x cEn1

Connection # 2:
cEx2 = cW2 x cEn2

Bias:
bEx = bW x 1

As you can see from the Bias calculation, the Bias.connEntry (bEn) value is always 1, which means that the Bias.connExit (bEx) value is always equal to the Bias.weight (bW) value.

The Neuron Input Value is equal to the sum of all the connExit values (cEx1 and cEx2 in this case) plus the bias (bEx), and can be represented by the following formula.

Neuron Input Value:

Neuron Input Value = cEx1 + cEx2 + bEx
or
Neuron Input Value = [cW1 x cEn1] + [cW2 x cEn2] + [bW x 1]

Now that we have the Neuron Input Value, we can put this through the Activation Function to get the Neuron Output Value. This can be represented by the following formula.

Neuron Output Value:

Neuron Output Value = 1 / (1 + EXP(-1 x Neuron Input Value))

We will use this Neuron Output value as an input to the next layer, therefore it automatically becomes a connEntry for one of the neuron connections in the next layer.

Now that we know what a neuron does, let us create a neuron class. See the code below.

processing code Neuron Class

```

/* -----------------------------------------------------------------------
   A neuron does all the calculations to convert an input to an output
-----------------------------------------------------------------------  */

class Neuron{
  Connection[] connections={};
  float bias;
  float neuronInputValue;
  float neuronOutputValue;
```

Total Pageviews

2,066,848

Subscribe

Posts

Comments

```
  float deltaError;

  //The default constructor for a Neuron
  Neuron(){
  }

  //The typical constructor of a Neuron - with random Bias and Connection weights
  Neuron(int numOfConnections){
    randomiseBias();
    for(int i=0; i<numOfConnections; i++){
      Connection conn = new Connection();
      addConnection(conn);
    }
  }

  //Function to add a Connection to this neuron
  void addConnection(Connection conn){
      connections = (Connection[]) append(connections, conn);
  }

  //Function to return the number of connections associated with this neuron.
  int getConnectionCount(){
      return connections.length;
  }

  //Function to set the bias of this Neuron
  void setBias(float tempBias){
    bias = tempBias;
  }

  //Function to randomise the bias of this Neuron
  void randomiseBias(){
    setBias(random(1));
  }

  //Function to convert the neuronInputValue to an neuronOutputValue
  //Make sure that the number of connEntryValues matches the number of connections

  float getNeuronOutput(float[] connEntryValues){
    if(connEntryValues.length!=getConnectionCount()){
      println("Neuron Error: getNeuronOutput() : Wrong number of connEntryValues");
      exit();
    }

    neuronInputValue=0;

    //First SUM all of the weighted connection values (connExit) attached to this neuron.
    for(int i=0; i<getConnectionCount(); i++){
      neuronInputValue+=connections[i].calcConnExit(connEntryValues[i]);
    }

    //Add the bias to the Neuron's inputValue
    neuronInputValue+=bias;

    //Send the inputValue through the activation function to produce the Neuron's outputVa
lue
    neuronOutputValue=Activation(neuronInputValue);

    //Return the outputValue
    return neuronOutputValue;
  }

  //Sigmoid Activation function
  float Activation(float x){
    float activatedValue = 1 / (1 + exp(-1 * x));
    return activatedValue;
  }

}
```

code formatter

When you create a new Neuron, you have the option to create the basic neuron shell with the Neuron() constructor, however, you are more likely to call the 2nd constructor, which allows you to set the number of input connections attached to this neuron.
For example, if I create a neuron using New Neuron(4), then this will automatically

attach and associate four connections with this neuron randomise the weight of each connection

randomise the bias of this neuron

These are the functions within the Neuron Class:

addConnection(): adds a new connection to this neuron

getConnectionCount(): returns the number of connections associated with this neuron

setBias(): sets the Bias of the neuron to a specific value.

randomiseBias() : randomises the Bias of this neuron

getNeuronOutput() : values are fed through the neuron's connections to create the neuronInputValue, which is then sent through the Sigmoid Activation function to create the neuronOutputValue.

Activation() : is the function used by the getNeuronOutput() function to generate the neuronOutputValue.

Up next: Neural Network (Part 3) : The Layer

To go back to the table of contents click here

Posted by Scott C at 00:58 G+1 Recommend this on Google

Labels: Activation function, ArduinoBasics, best arduino blog, Bias, code, Neural Network, Neuron, Processing.org, project, Sigmoid, tutorial, Weight

No comments:

Post a Comment

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.

Enter your comment...

Comment as: Unknown (Goc ▼ Sign out

Publish Preview Notify me

Links to this post

Create a Link

Arduino Basics

An Arduino tutorial blog. Free Arduino tutorials for everyone !

12 August 2011

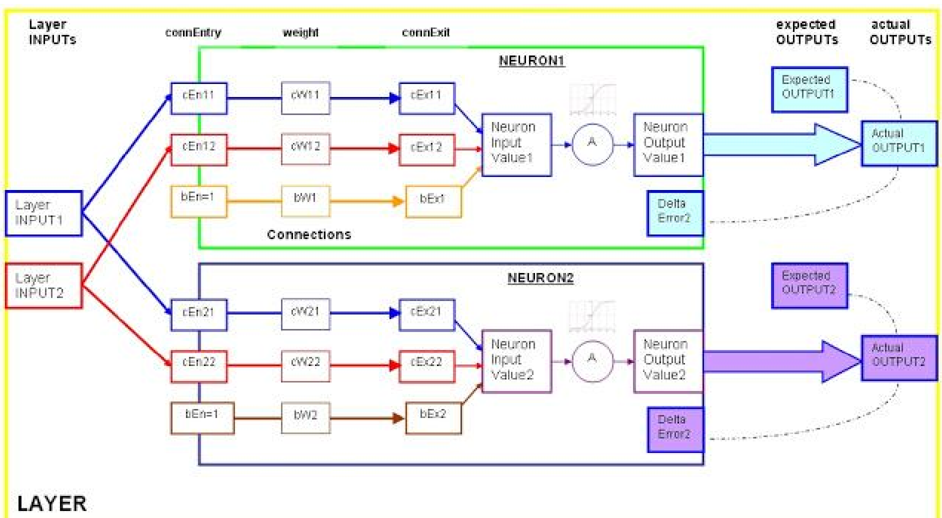
Neural Network (Part 3): The Layer

The Layer

The purpose of a layer is mainly to group and manage neurons that are functionally similar, and provide a means to effectively control the flow of information through the network (forwards and backwards). The forward pass is when you convert input values into output values, the backwards pass is only done during network training.

In my neural network, the Layer is an essential building block of the neural network. They can work individually or as a team. Let us just focus on one layer for now

This is what a layer with two neurons looks like in my neural network:



As you can see from the picture above, I tend to treat each neuron as an individual.
Both neurons have the same connEntry values (cEn11 = cEn21) and (cEn12 = cEn22), which stem from the layerINPUTs. Other than that, the rest is different. Which means that Neuron1 could produce an output value close to 1, and at the same time Neuron2 could produce an output close to 0, even though they have the same connEntry values at the start.

The following section will describe how the layerINPUTs get converted to actualOUTPUTs.

Step1: Populate the layerINPUTs:
layerINPUTs are just a placeholder for values that have been fed into the neural network, or come from a previous layer within the same neural network. If this is the first layer in the network, then the layerINPUTs would be the Neural Network's input values (eg Sensor data). If this is the 2nd layer in the network, then the layerINPUTs would equal the actualOUTPUTs of Layer One

Step2: Send each layerINPUT to the neuron's connection in the layer.

You will notice that every neuron in the same layer will have the same number of connections. This is because each neuron will connect only once to each of the layerINPUTs. The relationship between layerINPUTs and the connEntry values of a neuron are as follows.

layerINPUTs1:
cEn11 = layerINPUTs1 Neuron1.Connection1.connEntry
cEn21 = layerINPUTs1 Neuron2.Connection1.connEntry

layerINPUTs2:
cEn12 = layerINPUTs2 Neuron1.Connection2.connEntry
cEn22 = layerINPUTs2 Neuron2.Connection2.connEntry

Therefore, the connEntry of the *first* connection of every neuron in this layer will equal the *first* layerINPUT value in this layer.

Step3: Calculate the connExit values of each connection (including bias)

Neuron 1:
cEx11 = cEn11 x cW11
cEx12 = cEn12 x cW12
bEx1 = 1 x bW1

Neuron 2:
cEx21 = cEn21 x cW21
cEx22 = cEn22 x cW22
bEx2 = 1 x bW2

Step4: Calculate the NeuronInputValue for each neuron

Neuron1:
Neuron1.NeuronInputValue = cEx11 + cEx12 + bEx1

Neuron2:
Neuron2.NeuronInputValue = cEx21 + cEx22 + bEx2

Step5: Send the NeuronInputValues through an Activation function to produce a NeuronOutputValue

Neuron1:
Neuron1.NeuronOutputValue= 1/(1+EXP(-1 x Neuron1.NeuronInputValue))

Neuron2:
Neuron2.NeuronOutputValue= 1/(1+EXP(-1 x Neuron2.NeuronInputValue))

Please note that the NeuronInputValues for each neuron are different !

Step6: Send the NeuronOutputValues to the layer's actualOUTPUTs

actualOUTPUT1 = Neuron1.NeuronOutputValue

actualOUTPUT2 = Neuron2.NeuronOutputValue

The NeuronOutputValues become the actualOUTPUTs of this layer, which then become the layerINPUTs of the next layer. And the process is repeated over and over until you reach the final layer, where the actualOUTPUTs become the outputs of the entire neural network.

Here is the code for the Layer class:

processing code Layer Class

```

class Layer{
  Neuron[] neurons = {};
  float[] layerINPUTs={};
  float[] actualOUTPUTs={};
  float[] expectedOUTPUTs={};
  float layerError;
  float learningRate;


  /* This is the default constructor for the Layer */
  Layer(int numberConnections, int numberNeurons){
    /* Add all the neurons and actualOUTPUTs to the layer */
    for(int i=0; i<numberNeurons; i++){
      Neuron tempNeuron = new Neuron(numberConnections);
      addNeuron(tempNeuron);
      addActualOUTPUT();
    }
  }


  /* Function to add an input or output Neuron to this Layer */
  void addNeuron(Neuron xNeuron){
        neurons = (Neuron[]) append(neurons, xNeuron);
  }


  /* Function to get the number of neurons in this layer */
  int getNeuronCount(){
    return neurons.length;
  }


  /* Function to increment the size of the actualOUTPUTs array by one. */
  void addActualOUTPUT(){
      actualOUTPUTs = (float[]) expand(actualOUTPUTs,(actualOUTPUTs.length+1));
  }


  /* Function to set the ENTIRE expectedOUTPUTs array in one go. */
  void setExpectedOUTPUTs(float[] tempExpectedOUTPUTs){
    expectedOUTPUTs=tempExpectedOUTPUTs;
  }


  /* Function to clear ALL values from the expectedOUTPUTs array */
  void clearExpectedOUTPUT(){
       expectedOUTPUTs = (float[]) expand(expectedOUTPUTs, 0);
  }


  /* Function to set the learning rate of the layer */
  void setLearningRate(float tempLearningRate){
    learningRate=tempLearningRate;
  }


  /* Function to set the inputs of this layer */
  void setInputs(float[] tempInputs){
    layerINPUTs=tempInputs;
  }



  /* Function to convert ALL the Neuron input values into Neuron output values in this layer,
      through a special activation function. */
  void processInputsToOutputs(){

    /* neuronCount is used a couple of times in this function. */
```

```
    int neuronCount = getNeuronCount();

    /* Check to make sure that there are neurons in this layer to process the inputs */
    if(neuronCount>0) {
      /* Check to make sure that the number of inputs matches the number of Neuron Connections. */
      if(layerINPUTs.length!=neurons[0].getConnectionCount()){
        println("Error in Layer: processInputsToOutputs: The number of inputs do NOT match the number of Neuron connections in this layer");
        exit();
      } else {
        /* The number of inputs are fine : continue
           Calculate the actualOUTPUT of each neuron in this layer,
           based on their layerINPUTs (which were previously calculated).
           Add the value to the layer's actualOUTPUTs array. */
        for(int i=0; i<neuronCount;i++){
          actualOUTPUTs[i]=neurons[i].getNeuronOutput(layerINPUTs);
        }
      }
    }else{
      println("Error in Layer: processInputsToOutputs: There are no Neurons in this layer");
      exit();
    }
  }


  /* Function to get the error of this layer */
  float getLayerError(){
    return layerError;
  }


  /* Function to set the error of this layer */
  void setLayerError(float tempLayerError){
    layerError=tempLayerError;
  }


  /* Function to increase the layerError by a certain amount */
  void increaseLayerErrorBy(float tempLayerError){
    layerError+=tempLayerError;
  }


  /* Function to calculate and set the deltaError of each neuron in the layer */
  void setDeltaError(float[] expectedOutputData){
    setExpectedOUTPUTs(expectedOutputData);
    int neuronCount = getNeuronCount();
    /* Reset the layer error to 0 before cycling through each neuron */
    setLayerError(0);
     for(int i=0; i<neuronCount;i++){
       neurons[i].deltaError = actualOUTPUTs[i]*(1-actualOUTPUTs[i])*(expectedOUTPUTs[i]-actualOUTPUTs[i]);

       /* Increase the layer Error by the absolute difference between the calculated value (actualOUTPUT) and the expected value (expectedOUTPUT). */
       increaseLayerErrorBy(abs(expectedOUTPUTs[i]-actualOUTPUTs[i]));
     }
  }


  /* Function to train the layer : which uses a training set to adjust the connection weights and biases of the neurons in this layer */
  void trainLayer(float tempLearningRate){
    setLearningRate(tempLearningRate);

    int neuronCount = getNeuronCount();

    for(int i=0; i<neuronCount;i++){
      /* update the bias for neuron[i] */
      neurons[i].bias += (learningRate * 1 * neurons[i].deltaError);

      /* update the weight of each connection for this neuron[i] */
      for(int j=0; j<neurons[i].getConnectionCount(); j++){
          neurons[i].connections[j].weight += (learningRate * neurons[i].connections[j].connEntry * neurons[i].deltaError);
      }
    }
  }
}
```

code formatter

Within each layer, you have neuron(s) and their associated connection(s). Therefore it makes sense to have a constructor that automatically sets these up.

If you create a new Layer(2,3), this would automatically

- add 3 neurons to the layer
- create 2 connections for each neuron in this layer (to connect to the previous layer neurons).
- randomise each neuron bias and connection weights.
- add 3 actualOUTPUT slots to hold the neuron output values.

Here are the functions of the Layer class:

- **addNeuron()** : adds a neuron to the layer
- **getNeuronCount()** : returns the number of neurons in this layer
- **addActualOUTPUT()** : adds an actualOUTPUT slot to the layer.
- **setExpectedOUTPUTs()** : sets the entire expectedOUTPUTs array in one go.
- **clearExpectedOUTPUT()** : clear all values within the expectedOUTPUTs array.
- **setLearningRate()** : sets the learning rate of the layer.
- **setInputs()** : sets the inputs of the layer.
- **processInputsToOutputs()** : convert all layer input values into output values
- **getLayerError()** : returns the error of this layer
- **setLayerError()** : sets the error of this layer
- **increaseLayerErrorBy()** : increases the layer error by a specified amount.
- **setDeltaError()** : calculate and set the deltaError for each neuron in this layer
- **trainLayer()** : uses a training set to adjust the connection weights and biases of the neurons in this layer

There are a few functions mentioned above, which I have not yet discussed, and are used for neural network training (back-propagation). Don't worry, we'll go through them in the "back-propagation" section of the tutorial.

Next up: Neural Network (Part 4) : The Neural Network class

To go back to the table of contents click here

Posted by Scott C at 21:58 G+1 Recommend this on Google

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

2 comments:

Mohammad Hefny 29 September 2013 at 20:25

Thank you very much for your great articles and effort and the way you describe in details .... I really appreciate your great effort ...
Many Thanks again

Reply

Replies

Scott C 16 October 2013 at 21:36

no problem.. glad it helped you

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.

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

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