

Taking Initiative



Posted on April 3, 2008July 27, 2013 | Artificial Intelligence, Neural Networks

Basic Neural Network Tutorial – Theory

i

57 Votes

Well this tutorial has been a long time coming. Neural Networks (NNs) are something that i'm interested in and also a technique that gets mentioned a lot in movies and by pseudo-geeks when referring to AI in general. They are made out to be these really intense and complicated systems when in fact they are nothing more than a simple input output machine (well at least for the standard Feed Forward Neural Networks (FFNN)). As with any field the more you delve into it the more technical it gets and NNs are the same, the more research you do into them the more complicated architectures, training techniques, activation functions become. For now this is just a simple primer into NNs.

Introduction to Neural Networks

There are many different types of neural networks and techniques for training them but I'm just going to focus on the most basic one of them all – the classic back propagation neural network (BPN). The back propagation refers to the fact that any mistakes made by the network during training get sent backwards through it in an attempt to correct it and so teach the network what's right and wrong.

This BPN uses the gradient descent learning method. Trying to describe this simply at this point is going to be difficult so I'll leave it for a bit later, all you need to know is that it's so called because it follows the steepest gradient down a surface which represents the error function as it tries to find the minimum of the error function and by doing so decrease the error.

I wanted to skip over the basics of neural networks, and all that this is your brain and this is a neuron but I guess it's unavoidable. I'm not going to go into great detail as there is plenty of information already available online. Here are the wiki entries on FFNNs and back propagation:

<http://en.wikipedia.org/wiki/Back-propagation> (<http://en.wikipedia.org/wiki/Back-propagation>) ,
http://en.wikipedia.org/wiki/Feedforward_neural_network (http://en.wikipedia.org/wiki/Feedforward_neural_network) .

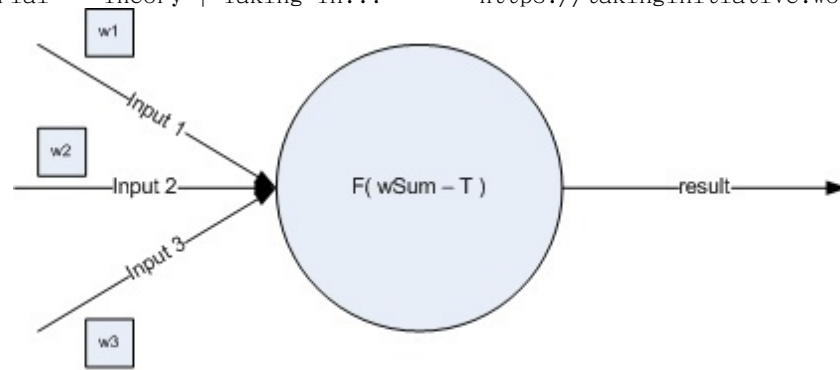
This is the second version of the tutorial since half way through the first one I realized that I needed to actually go over some of the theory properly before I could go over the implementation and so i've decided to split the tutorial into two parts: part 1(this) will go over the basic theory needed and part 2 will discuss some more advanced topics and the implementation.

Now I haven't even told you what a Neural Network is and what it is used for. Silly me! NNs have a variety of uses especially in classification or function-fitting problems, they can also be used to create emerging behaviour in agents reacting to environment sensors. They are one of the most important artificial intelligence tools available today. Just for the record I am by no means an expert on neural networks, I just have a bit of experience implementing and successfully using BPN's in various image classification problems before.

The Neuron

Okay enough blabbering from me; let's get into the thick of it. The basic building block of a NN is the neuron. The basic neuron consists of a black box with weighted inputs and an output.

Note: *perceptron – neuron that classifies its inputs into one of two categories, basically the output of a neuron is clamped to 1 or 0.*



Basic neuron as a black box

The black box section of the neuron consists of an activation function $F(X)$, in our case its $F(wSum - T)$ where $wSum$ is the weighted sum of the inputs and T is a threshold or bias value. We'll come back to the threshold value just now. The weights are initialized to some small random values and during training get updated. The weighted sum ($wSum$) is given below.

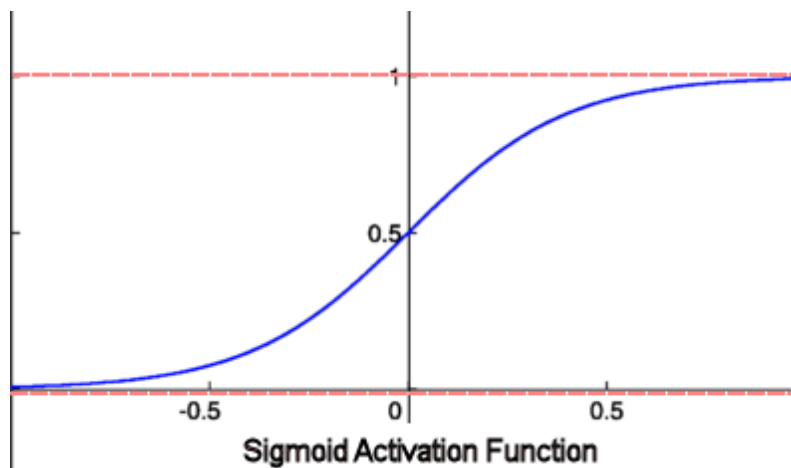
$$wSum = \sum_{i=1}^n weight_i \times input_i$$

Simple, huh? Now for the function F , there are various functions that can be used for F ; the most common ones include the step function and the sigmoid function. We will be using the sigmoid function in our BPN as its again the classical activation function. The sigmoid function and its derivative are defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x) \times (1 - \sigma(x))$$

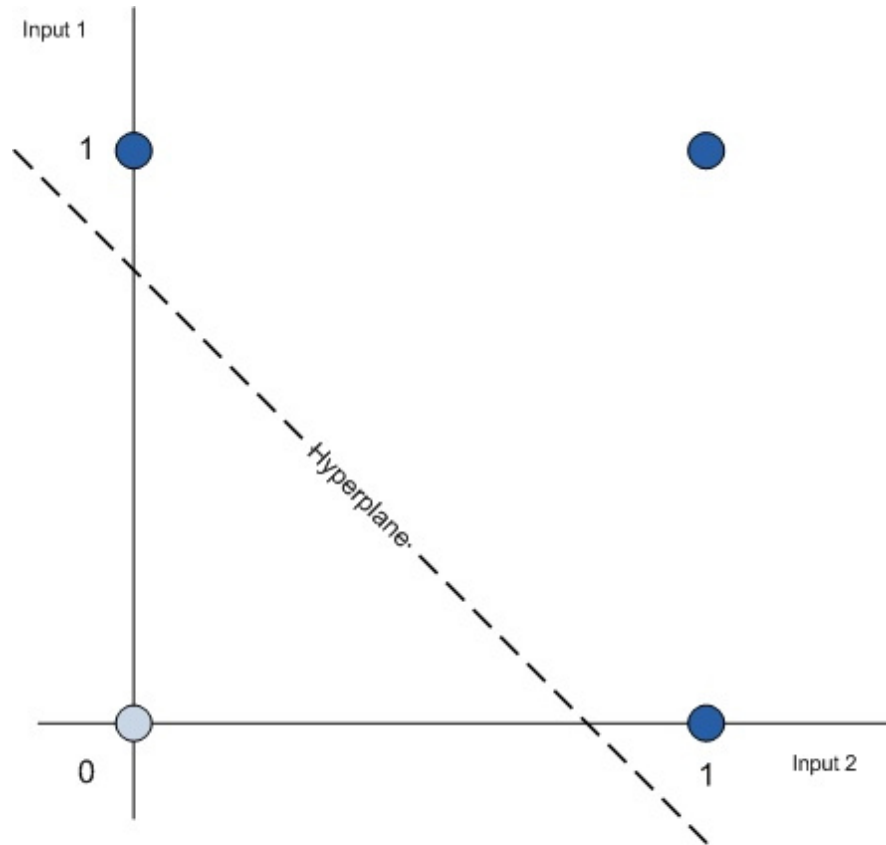
The sigmoid function has the following graph :



Note: Now its very important to realize that the sigmoid function can never return a 0 or a 1 due to its asymptotic nature. So often its a good idea to treat values over 0.9 as 1 and under 0.1 as 0.

Now we need to cover an important point regarding the input data and the desired output. Lets use the binary OR operator as an example to explain the function of the weights and threshold. With OR we

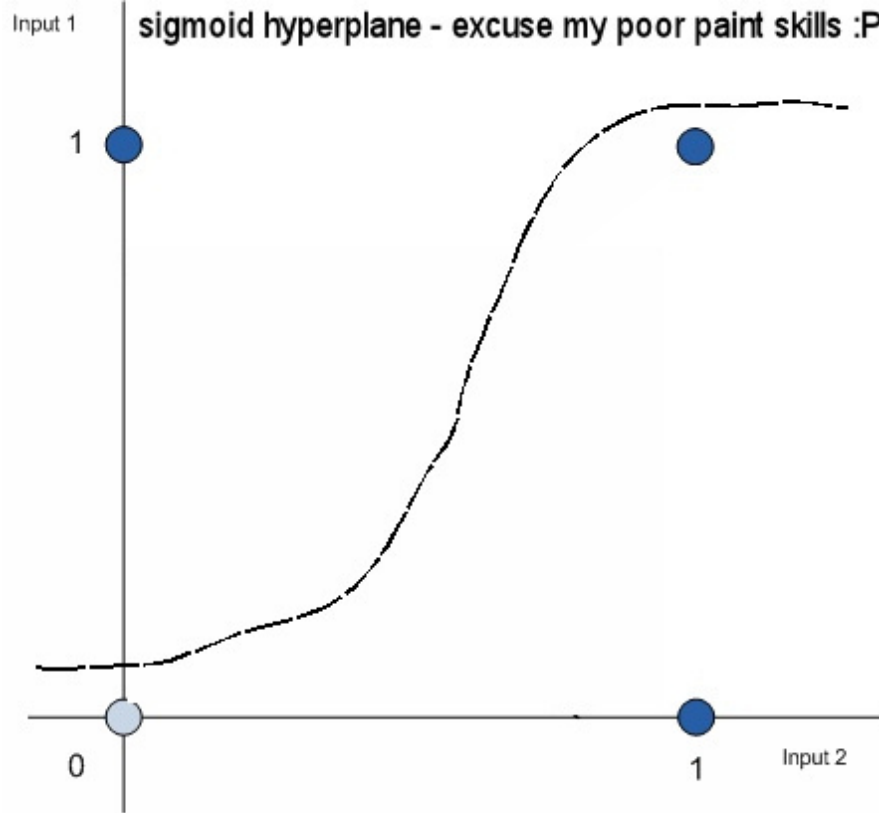
want a binary output telling us whether its true or not, so a single perceptron with two inputs is created. Now the search space for the neural network can be drawn as follows:



The OR operator modeled by a single perceptron

The dark blue dots represents values of true and the light blue dot represents a value of false, you can clearly see how the two classes are separable. We can draw a line separating them as in the above example. This separating line is called a hyperplane. A single neuron can create a single hyperplane and the above function can be solved by a single neuron.

Another important point is that the hyperplane above is a straight line, this means we used a linear activation function (i.e. a step function) for our neuron. If we used a sigmoid function or similar the hyperplane would resemble a sigmoid shape as seen below. (not the best image so please excuse my poor paint skills). The hyperplane generated by the image depends on the activation function used.



The OR operator modeled by a single perceptron

Remember that Threshold (Bias) value we had earlier? What does that do? Simply put it shifts the hyperplane left and right while the weights orientate the hyperplane. In graphical terms the Threshold translates the hyperplane while the weights rotate it. This threshold also need to be updated during the learning process.

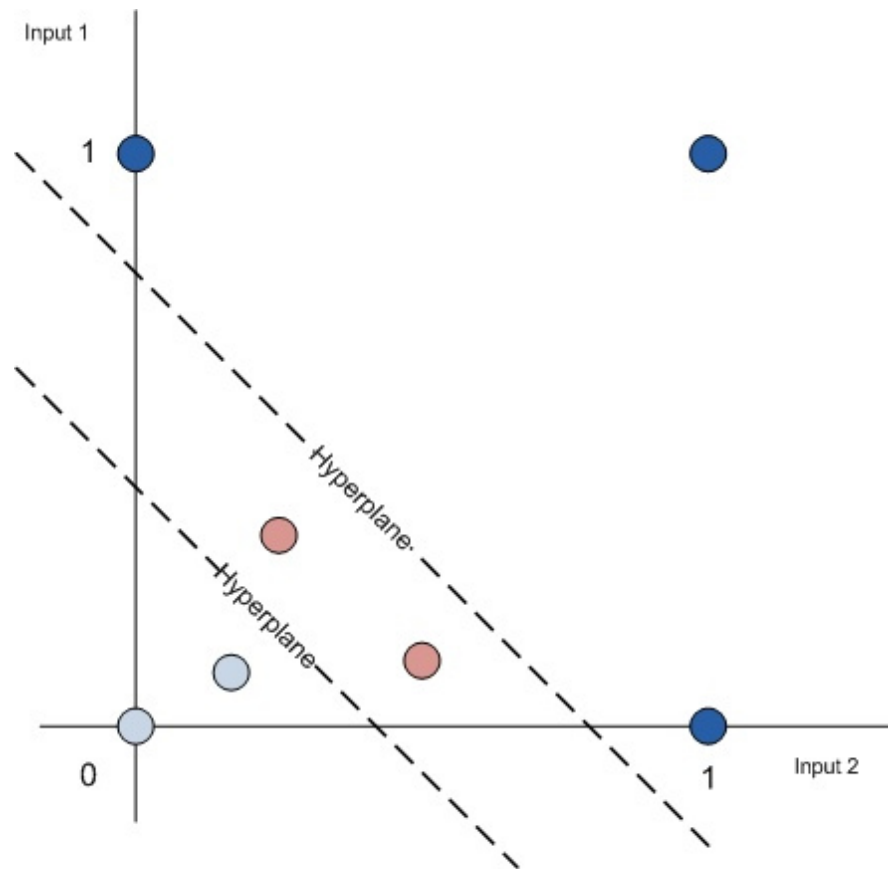
I'm not going to go into the details of how a neuron learns in detail or provide examples; there are many excellent books and online guides to do this. The basic procedure is as follows:

- run an input pattern through the function
- calculate the error (desired value – actual value)
- update the weights according to learning rate and error
- move onto next pattern

The learning rate term is a term that hasn't been mentioned before and is very important, it greatly affects the performance and accuracy of your network. I'll go over this in more detail once we get to the weight updates.

The Multilayer Neural Network

As I mentioned before for linearly separable problems a single neuron is sufficient but what about problems that have more than one class or ones where data isn't so well separated like in the example

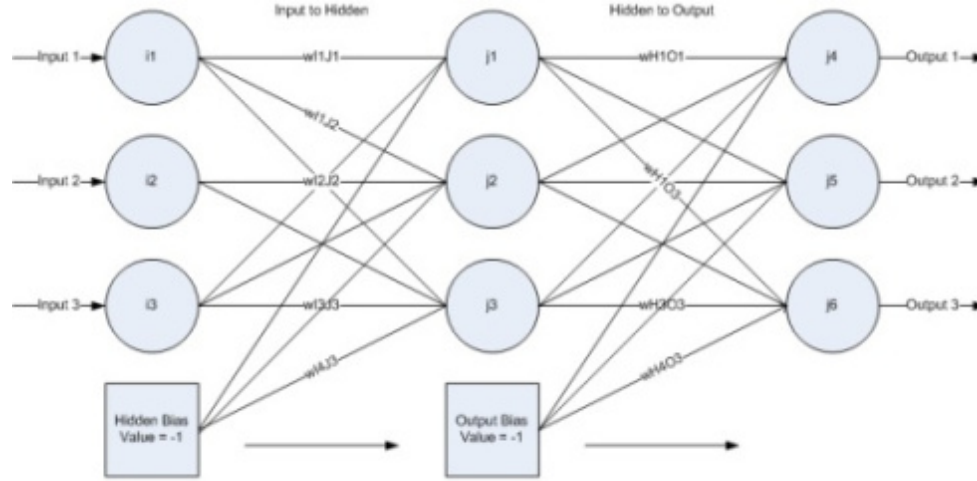


Non-linearly separable dataset

Here we need at least two hyper-planes to solve this problem so we need 2 neurons. This requires us to link up these neurons together, to link them up we'll need shared inputs and outputs – in other words a multilayer neural network. The standard architecture of a NN consists of 3 layers: an input layer, a hidden layer and an output layer. There are several proofs available that you will almost never need more than 3 layers (I'll try get links to the papers soon) and also more importantly we want to keep things simple.

NOTE: you almost never know what your search space looks like, that's why you're using a neural network, often you'll have to experiment with the neural network architecture in regards to how many hidden neurons you need to get a good result.

A Basic Multilayer Neural Network:



Standard Architecture for a Back Propagation Neural Network

Above is a basic multilayer neural network, the inputs are shared and so are the outputs, note that each of these links have separate weights. Now what are those square blocks in the neural network? They are our thresholds (bias) values, instead of having to store and update separate thresholds for each neuron (remember each neuron's activation function took a weighted sum minus a threshold as input), we simply create 2 extra neurons with a constant value of -1. These neurons are then hooked up to the rest of the network and have their own weights (these are technically the threshold values).

This results in the weighted sum + the weight of the threshold multiplied by -1, obviously you can see it's the same as we had earlier. Now when we update the weights for the network during backpropagation we automatically update the thresholds as well, saving us a few calculations and headaches.

Okay so far everything has (hopefully) been pretty simple especially if you have a bit of a background in NNs or have read through an introductory chapter in an AI textbook. There are only 3 things left to discuss – calculating the errors at the output, updating the weights (the back propagation) and the stopping conditions.

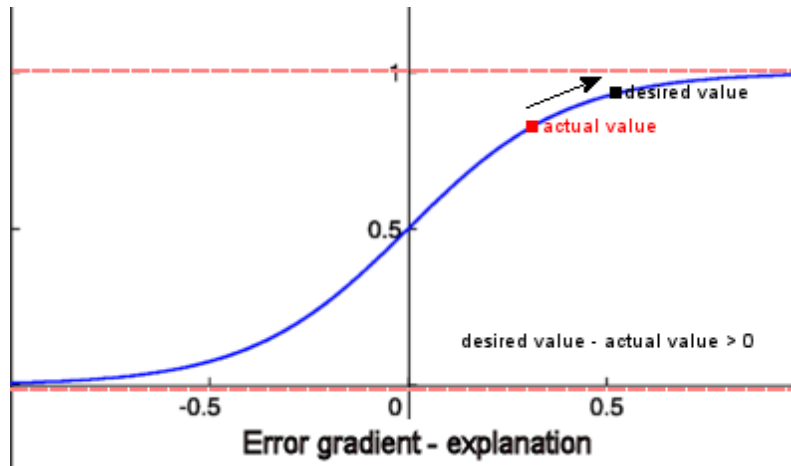
The only control over this architecture you have is over the number of hidden neurons since your inputs and desired outputs are already known, so deciding on how many hidden neurons you need is often a tricky matter, too many is never good, and neither is too little, some careful experimentation will often be required to find out an optimal amount of hidden neurons.

I'm not going to go over feeding the input forward as it's really simple: all you do is calculate the output (the value of the activation function for the weighted sum of inputs) at a neuron and use it as the input for the next layer.

The Neuron Error Gradients

Okay so obviously we need to update the weights in our neural network to give the correct output at the output layer. This forms the basis of training the neural network. We will make use of back-propagation for these weight updates. This just means input is fed in, the errors calculated and filtered back through the network making changes to the weights to try reduce the error.

The weight changes are calculated by using the gradient descent method. This means we follow the steepest path on the error function to try and minimize it. I'm not going to go into the math behind gradient descent, the error function and so on since its not really needed, simply put all we're doing is just taking the error at the output neurons (Desired value – actual value) and multiplying it by the gradient of the sigmoid function. If the difference is positive we need to move up the gradient of the activation function and if its negative we need to move down the gradient of the activation function.



This is the formula to calculate the basic error gradient for each output neuron k :

$$\delta_k = y_k(1 - y_k)(d_k - y_k)$$

where y_k is the value at output neuron k
and d_k is the desired value at output neuron k

There is a difference between the error gradients at the output and hidden layers. The hidden layer's error gradient is based on the output layer's error gradient (back propagation) so for the hidden layer the error gradient for each hidden neuron is the gradient of the activation function multiplied by the weighted sum of the errors at the output layer originating from that neuron (wow, getting a bit crazy here eh?):

$$\delta_j = y_j(1 - y_j) \sum_{k=1}^n w_{jk} \delta_k$$

The Weight Update

The final step in the algorithm is to update the weights, this occurs as follows:

$$w_{ij} = w_{ij} + \Delta w_{ij} \text{ and } w_{jk} = w_{jk} + \Delta w_{jk}$$

$$\text{where } \Delta w_{ij}(t) = \alpha \cdot \text{inputNeuron}_i \cdot \delta_j$$

$$\text{and } \Delta w_{jk}(t) = \alpha \cdot \text{hiddenNeuron}_j \cdot \delta_k$$

α – learning rate
 δ – error gradient

(<https://takinginitiative.files.wordpress.com/2008/04/weightupdates2.png>)

The alpha value you see above is the learning rate, this is usually a value between 0 and 1. It affects how large the weight adjustments are and so also affects the learning speed of the network. This value needs to be carefully selected to provide the best results, too low and it will take ages to learn, too high and the adjustments might be too large and the accuracy will suffer as the network will constantly jump over a better solution and generally get stuck at some sub-optimal accuracy.

The Learning algorithm

The BPN learns during a training epoch, you will probably go through several epochs before the network has sufficiently learnt to handle all the data you've provided it and the end result is satisfactory. A training epoch is described below:

For each input entry in the training data set:

- feed input data in (feed forward)
- check output against desired value and feed back error (back-propagate)

Where back-propagation consists of:

- calculate error gradients
- update weights

Stopping Conditions

These are some commonly used stopping conditions used for neural networks: desired accuracy, desired mean square error and elapsed epochs. I won't go over these in too much detail now as I will be covering them in the next tutorial with some training examples. The main reason I'm not going into detail here is that I haven't described the training of the network in detail, I need to go over the creating of training data sets, what generalization and validation errors are and so on. All this will be covered in greater detail in the next tutorial.

Conclusion

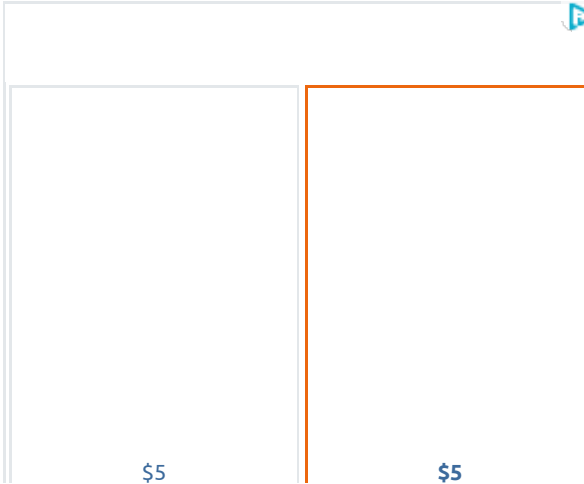
So this is it for my initial tutorial on the basics of neural networks; stay tuned for my next tutorial where I’m going to go over a few more things regarding neural networks including stopping conditions, training techniques, discussion of stochastic and batch learning, some examples of me training the network and I’ll also include my implementation of a classic back-propagation neural network class in c++ that features momentum and batch learning.

Continuation

Tutorial Continues in Part 2 : implementation and c++ source code – [NN Tutorial Part 2](https://takinginitiative.wordpress.com/2008/04/23/basic-neural-network-tutorial-c-implementation-and-source-code/)
(<https://takinginitiative.wordpress.com/2008/04/23/basic-neural-network-tutorial-c-implementation-and-source-code/>)



[Report this ad](#)



[Report this ad](#)

[Artificial Intelligence](#) [back-propagation](#) [Neural Network](#)

99 thoughts on “Basic Neural Network Tutorial – Theory”

kewl tutorial. learning algorithms such as NN have always interested me. Your explanations for the most part, are pretty simple to follow, but at crucial times you go off the rails leaving a disconnect. i guess you gap-out the things you already know and take for granted, but would be cool if you remembered them more explicitly. ☺

Capricorn , April 7, 2008 at 9:17 pm

Reply

thanks for the response, can you point out where i blank out or even anything you think I'm missing or want to know? Maybe i am leaving things out...

Part 2 will have more info and source code so that should help clear things up, i hope ☺

Bobby , April 7, 2008 at 9:25 pm

Reply

rite. now when is part 2 due?

Capricorn , April 10, 2008 at 9:02 pm

Reply

i've edited part 1 a bit, added more info and rewrote some sections. i should be able to put up part 2 tommorrow, hopefully...

depends on how rough work is...

Bobby , April 10, 2008 at 9:33 pm

Reply

great work, how did you do the graphics?

- , April 16, 2008 at 1:11 am

Reply

i used microsoft visio to do the pictures. The formulas were done in microsoft word 2k7 (it has an excellent equation editor)

Bobby , April 16, 2008 at 10:46 am

Reply

thanks! and again: good job.

- , April 16, 2008 at 8:24 pm

Reply

I love this tutorial.

Newbie in Neural Networks and understood everything.

thanks !!

contremaitre , April 18, 2008 at 10:15 am

Reply

thanks! It makes me happy to know I've managed to help someone!!

I'm busy with the second part at the moment, and i realize I'm actually covering a lot of things you wont find in textbooks. Well the textbooks seem take it for granted that you'll magically work these things out on your own not to mention that their explanation often leave a lot to be desired.

Hopefully these tutorials will save you guys some time... ☺

Bobby , April 18, 2008 at 1:27 pm

Reply

Can you please explain me The sigmoid function please, what means "e"? and -x

Please someone post a step by step calculus for the activation function with an example of a neuron

Alex , April 23, 2008 at 2:23 am

Reply

e is simply a standard constant (euler's number) with the value 2.71828...

[http://en.wikipedia.org/wiki/E_\(mathematical_constant\)](http://en.wikipedia.org/wiki/E_(mathematical_constant))

e^x is the exponential function (represented by the exp function in c/c++), so in our case the sigmoid is the inverse of $1 + \text{the exponential function of } -x$.

x is the input parameter you put into the sigmoid function to get a result (the y value if you want to graph it). So for an input of lets say 3 the output of the sigmoid function will be $1/(1+e^{-3})$ or in c++ code

```
double input = 3;
double output = 1 / ( 1 + exp(-input) );
```

I hope this helps. Its a weird question to ask though especially if you're interested in neural networks. I'd expect you to have at least a basic calculus education.

Bobby , April 23, 2008 at 6:32 am

Reply

Hi, I found this website interesting. I am a beginner in neural networks and need some basic info on it.

I'll be pleased if you provide me with the answers.

1. What is the role of a neuron in a NN?
2. How to set the target value for the network?
3. What is the purpose of an activation function?
4. How to calculate the subsequent weighted components in a network?
5. Given n input pairs, how many neurons are needed and how many layers are needed?
6. In what terms do we get the actual output?

Nisha , April 23, 2008 at 9:22 am

Reply

hey nisha,

I'm not sure if you read the tutorial properly since questions 1,3,4,5 and 6 are answered in the above tutorial.

basics of what a neural network is, what a neuron is, why it works etc. this tutorial is a simple explanation of the more complicated topics for a basic BPN.

Bobby , April 23, 2008 at 2:56 pm

Reply

Thanks Bobby, i thought "e" is some part of the neuron and that's why i was confused... yes it helps me

I'm programming in Visual Basic and i think this is the solution:

$1 / (1 + (\text{Exp}(-3))) = 0.952574126822433$

Can you check it for me pls? In C++ the result it's the same?

I just want to check if it's okay because in vb it is a little different than in c++

Alex , April 23, 2008 at 10:04 pm

Reply

that looks perfect! best of luck with your neural network, if you want download my source code in the second tutorial and look through the comments, they will probably be super useful, as you can see how it was implemented.

Bobby , April 24, 2008 at 1:06 pm

Reply

If i make a nn with perceptrons they will adjust to the pattern that i'm giving, i guess i must stop the weights from changing after they are trained? I mean... this can be the only solution for a nn... correct me please if i'm wrong

Alex , April 25, 2008 at 1:46 pm

Reply

yes, remember tho the neurons in the neural network aren't perceptrons since their output isn't just 0 or 1. they are just neurons.

once you've trained the network all you do is feed data forward and check the result. Don't train again unless you have new data that the neural network can't handle, and when you train again you have to train both the old and the data.

Bobby , April 25, 2008 at 5:46 pm

Reply

Man, c++ is dead. Learn java, or c# if you love microsoft so much.

boris , May 7, 2008 at 2:39 pm

Reply

hahaha, i know both and another 10 or so languages... C++, C# and php are my favorite tho. Java IMHO for idiots and c# although great can't come close to the performance or control of c++...

that comment is so stupid, that I'm not gonna bother deleting or arguing ☺

Bobby , May 7, 2008 at 3:27 pm

Reply

poh

jgh , May 8, 2008 at 4:36 pm

Reply

what functions i must use in the first (input, weight) layer while the network is training? I can use

here the function with desired result but, between the hidden layer and output what function will i use?
The same one?

Sonyx , June 13, 2008 at 7:08 pm

Reply

i don't really understand what you're asking, there are no functions between the layers just the weights for the links.

Bobby , June 14, 2008 at 1:59 am

Reply

Thanks Bobby... I made it, i understand it now and made my first multilayer nn and it works great, thx again, keep in touch, write me on my email so i can have your address, bye bye

Sonyx , June 29, 2008 at 4:46 am

Reply

there is a way for the network to get trained with fewer iterations? An example, for learning the xor i need something like 7 000 iterations, it's not a little to much, i just ask you, is there a way 4 the network to learn faster?

Sonyx , September 22, 2008 at 5:56 am

Reply

the performance of the learning is dependent on the learning parameters: the learning rate and momentum. The architecture of the NN is extremely important here too, so for XOR you'd need a architecture of around 2 hidden neurons and then you must play around with the learning rate.

Bobby , September 22, 2008 at 8:46 am

Reply

- with the learning rate 1 it's ready after 5440 iterations
- with higher learning rate than 1 it's less accurate
- and with no learning rate it's ready after 5520 iterations

So, why bother using learning rate? It's pretty much the same
Correct me please if i'm wrong on this one

Sonyx , September 22, 2008 at 2:17 pm

Reply

Thank you so much for this article, it saved me time!

Nelia , May 19, 2017 at 3:42 am

Reply

From the title of this post, I thought you did "The Art of Living" program conducted by Sri Sri Ravi Shankar. Anyway, check it out at artofliving.org, when you get a chance. I'm sure you won't be disappointed!

<http://goanalyze.info/salesforceliveagent.com> , May 31, 2017 at 8:10 pm

Reply

you shouldn't be using a learning rate greater than 1, try something like 0.001 or similar. If the learning rate is too large, the weight changes are too great and it jumps past the correct values.

Bobby , September 22, 2008 at 3:02 pm

Reply

i get a reasonable error when i use 1 as learning rate or when i'm not using it at all, with the LR less

then 1 it takes much longer for the nn to train, and with values above 1 it doesn't train properly i didn't even mentioned or thought of 0.001 because i want it to train more faster, and this really it's slowing it down
it's ok with 5500 iterations for solving xor, i just asked if there is a way to do it more faster...
and i'm doing this with some random weights, not some weights that helps the network to do it more faster
i'm curious if the network is able to solve it quicker, or this is like an average training, with 5000 iterations for getting a good output with a low error

Sonyx , September 22, 2008 at 8:02 pm

Reply

okay listen to me, your architecture should be as follows: 2 inputs, 2 hidden, 1 output

i just tested with my nn, with a learning rate of 0.4 and momentum at 0.9 i get 100% accuracy in around 20-30 iterations on a data set of 100 patterns, increasing the number of hidden neurons may help decrease the iteration time, with a learning rate of 1 and 4 hidden neurons, i can get it down to between 10 and 20 iterations.

its all trial and error to get the optimal parameters, people spend years finding out good techniques to automatically find the optimal parameters, and also this form of training the neural network is the most basic.

Bobby , September 22, 2008 at 9:27 pm

Reply

Hi man I'm working on a company that ask me if it's possible to detect some product in assemble line on a industry (Beer production for example) with a camera filming the assembly line.

Do you know how I can use NN to detect those image patterns..

Ps: I can treat the image with DSP to transform the beer in somethink grayscale and extruded to a very simple form

Leonardo , September 22, 2008 at 10:11 pm

Reply

Hi,

Its a nice explanation about BP algorithm. I have implemented a C- code for this BP algo for character recognition, and I am facing an issue in the training part. Actually I am looking to train the neural network for all 36 characters(26 -> alphabets + 9 -> integers) but it takes a lot of time for that.

Can anyone suggest me some efficient method for training the BP neural network.

Rahul T , October 14, 2008 at 2:00 pm

Reply

BP is the training method, you can try speeding it up with momentum as i explained in the second part. other than that you can look at perhaps using a genetic algorithm to train your neural network.

Bobby , October 14, 2008 at 2:48 pm

Reply

Nice page – although I'm not sure about the accuracy of the statement that the output hyperplane takes on the shape of the output function of your network. Do you have any references for this – I'd be interested to know if it true.

A sigmoid neuron with 2 inputs will produce a straight line division in the input space. The main advantage of the sigmoid over the a digital MCP unit is that sigmoid functions have a gradient over which BP learning can be applied.

Pete ,November 11, 2008 at 1:32 am

Reply

hey pete, you may be correct about the hyperplane, that was something i remembered from a lecture a long time ago, so i may be incorrect, I'll try see if i can dig up some info regarding it. 😊

Bobby ,November 14, 2008 at 2:57 pm

Reply

Wow! I love dogs but i love cats more. I hope you'd be given free cat clip art as well. Well anyways a very big thank you for sharing this free stuff.

Mildred ,May 18, 2017 at 3:26 pm

Reply

Hi,

At begining what I'm going reach. There is neural network recognizing letters and obtain brail's representation of letter.

So I assume input pattern has (5×7)inputs, output pattern has 6 outputs.

Activation function is sigmoid function. The difference is in computing RO, well my implementation is based on http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Of course I encountered problems, and didnt know why net is not learning properly then I have chacked xor – its works and one of N (so one output was responsible for recognision one letter otherwise 0) outputs and there are some problems.

But at most I want to know if in Your opinion usage given equations in this tutorial is crucial to get positive results.

jones ,December 6, 2008 at 11:03 pm

Reply

This tutorial is WRONG. When you say “Another important point is that the hyperplane above is a straight line, this means we used a linear activation function (i.e. a step function) for our neuron. If we used a sigmoid function or similar the hyperplane would resemble a sigmoid shape as seen below. (not the best image so please excuse my poor paint skills). The hyperplane generated by the image depends on the activation function used” It is clear that the form of the activation function can change the shape of the hyper-surface that divides the input space. But a sigmoid does not look like a sigmoid when plotted in the input space. The whole idea of activation function is to apply it to a linear combination of the inputs, and so in your example the sigmoidal structure is seen not in the plane but in the third dimension. In the plane you are watching from above and you would see not a simple hyperplane but a soft surface coming from the plane (that has zero output) towards another plane with the highest (1) activity. If I'm wrong, you could calculate the XOR function with a single neuron with a sigmoidal activation function and I bet my money you cannot do that.

Moreover a step function is non-linear, for linearity is defined as $f(x)$ is linear iff $f(ax+by)=af(x)+bf(y)$, which is clearly not the case for a step function.

Juan C. Valle LISboa ,March 27, 2009 at 1:19 am

Hi Juan,

You are completely correct, I never really proof read the tutorial that well, which is obvious from my “linear” step function reference.

I will correct that. Also you are right about the hyperplane issue as well, furthermore I’ve made another mistake by referring to decision boundaries as hyperplanes, even so the shape is not affected to a great extent by the activation function.

I did some quick research and correct if I’m wrong but the hyperplane actually gets generated by the linear combination of the weights and inputs? the decision boundary is created on this hyperplane / hyper surface by the activation function?

Also this was meant as a beginning tutorial, so I have no idea why I even mentioned the whole topic. Thanks for the comment and I’ll correct the text first chance i get!

Bobby , March 27, 2009 at 7:37 am

Reply

Pingback: [A simple non-linear neuron model « dare2be | eb2erad](#)

Sir, Thank you for your code.. This made us understand how complicated an NN code is.....

KUSUM , November 20, 2009 at 8:39 am

Reply

Sir I have just gone through the code.. And found that NN is really complicated...

Gopus , November 20, 2009 at 8:42 am

Reply

Pingback: [Radical Makers](#)


Looks great and well structured.

How does one change the code ... once the nn has been trained... to allow the nn to be run against another set of inputs with the same derived weights.

Thanks in advance.

Stuart , February 18, 2010 at 3:13 am

Reply

Great tutorial. Just what I was googling for  Thanks!

Dan , April 26, 2010 at 12:35 am

Reply

did you ever hear of probabilistic neural network (PNN)? what is characteristic that distinguishes the PNN with the other neural network methods?

if I read a tutorial about PNN, I did not catch the location of probabilstiknya.

Thank you for your response ...

vv , June 10, 2010 at 11:13 am

Reply

I dont really know much about PNNs, though what i do know might help you in any further

replaced with a RBF (Radial Basis Function) function (this function varies between PNN). These types of NNs are better suited for classification problems than standard FFNNs and are less likely to be affected by local minima.

Thats pretty much as far as my knowledge of PNNs goes.

Bobby , June 10, 2010 at 11:30 am

Reply

thank you...

fifi , June 11, 2010 at 11:20 am

Dear Sir,,

Do you have examples of applications of artificial neural networks for forecasting using matlab software.?

dionisius , August 14, 2010 at 8:07 am

Reply

the part where the weight is updated the formula given is:

$\Delta(W_{jk})(t) = a \cdot \text{hiddenneuron}_j \cdot (\Delta)_k$

i dont understand what hiddenneuronj refers to...pl help

thank you

ritwik , February 1, 2011 at 7:47 pm

Reply

it refers to the j'th hidden neuron... the j value is just an index

Bobby , February 1, 2011 at 9:54 pm

Reply

you mean its the output of the jth hidden neuron?

ritwik , February 2, 2011 at 3:06 pm

Reply

yeh exactly...

Bobby , February 2, 2011 at 3:38 pm

can different neurons in the same layer have different threshold values?

ritwik , February 11, 2011 at 12:21 pm

Reply

yes each neuron can have its own threshold value but i'm not sure if there is any benefit in doing so..

Bobby , February 11, 2011 at 12:37 pm

Reply

but how do i calculate the threshold value for a layer of neurons?

ritwik , February 12, 2011 at 5:32 am

the standard technique is to create a single bias/threshold neuron with a value of -1 for an entire layer. As far as I know there is no single technique for calculating the bias value. I would tweak the value and see what effect it would have on the network for your application.

Bobby ,February 12, 2011 at 9:54 am

Reply

Can different neurons in the same hidden layer have different activation functions??

Amit Choudhary ,February 14, 2011 at 11:13 pm

Reply

yes technically each neuron can have its own activation function, but I'm not sure if there would be any use in doing so. As far as i can recall its recommended that all neurons have the same activation function...

Bobby ,February 14, 2011 at 11:40 pm

Reply

Thanks. Already using this-gets me to understand NN. One simple question as I was modelling some ideas (trying to teach it to learn number "3" by drawing 3 s using bitmaps. Accuracy stays zero (0), not sure yet why. I use same settings as in your example. Every pixel in 8x8 bitmap is input 1 on or 0 off.): If I teach for example to recognize number "3" do I also teach the cases which are not "3"? So is it best to only teach what is 3 or also what is NOT 3 (like giving 2 and putting value for that 0 if 3 is 1)?

Juha ,February 25, 2011 at 6:46 pm

Reply

yes, you need examples of the a 1 (yes) value for your output neuron as well as example for a 0 output at your final neuron.

try reducing the learning rate, the number of hidden neurons as well sa remove momentum (set it to 0) and see what happens. It should learn really quickly and kinda oscilate between 2 or so accuracy values.

Bobby ,February 25, 2011 at 11:33 pm


Reply

Thanks, I just started so I have to learn all of these momentums etc. :). Its very interesting how quickly and efficiently it learns. I created 44 "V" letters and 44 "U" letters (so very similar letters so to see how it sees them). I used again 8x8 bitmap to draw letters by hand. V output value in testing 1 and U=0. I run 2000 epoch s. (83% accuracy). Then I tested running the weights... and it seems to almost always distinct V from U!! This is promising – and also very clearly. V s have output values like 0.98 and U s 0.02-0.03 many times. I am into this! :). Thanks again Bobby for the simple class. I ll be following this site. And hopefully giving some input about my research at some point.

Juha ,February 26, 2011 at 2:35 am

Reply

I'm really glad to hear that! Look forward to hearing about your future research!!

I'm sure if you play around with the number of hidden neurons and the momentum/learning rates you can get that 2000 even lower  With NN's you kinda always have to experiment with the values to find the optimal ones.

Bobby ,February 26, 2011 at 2:44 am

Reply

Dear Bobby,


My research project is on handwritter character recognition. For character feature extraction, 2/28 上午10:45

horizontal and vertical projection techniques are to be implemented. How can we get the projection profile feature vector that is to be applied to the neural network as input. Can you please give me your Email-id as I want to send you a research paper published at <http://www.sciencedirect.com>. Mine is amit.choudhary79@gmail.com

No body is here to guide me. Please help me...
Thanks

Amit Choudhary , March 3, 2011 at 3:37 am

Reply

Amit take a look at my article on the radon transform. The radon transform is used with great success for handwriting recognition. I do not have the time right now to be able to help you in detail, so just do your research and I'm sure you'll be okay 

Bobby , March 3, 2011 at 5:43 am

Reply

This is Awesome !!! thank you for your great work

Jiyung , April 8, 2011 at 8:54 am

Reply

Pingback: [C++ Back Propagation Neural Network Code v2](#) « [Taking Initiative: Bobby Anguelov's Blog](#)

I just like the helpful information you provide for your articles. I will bookmark your blog and take a look at again right here frequently. I am quite sure I'll learn many new stuff proper here! Best of luck for the following!

Разработка, продвижение, установка и создания веб сайтов по доступным ценам! , January 15, 2012 at 8:55 pm

Reply

i love u so nice topic i enjoyed alot

samira , May 8, 2012 at 6:53 pm

Reply

Is it backpropagation neural network can be implemented in PHP to develop a system?

Shahila , December 15, 2012 at 5:26 am

Reply

nice and clear tutorial on a difficult subject

bu , January 17, 2013 at 8:07 pm

Reply

Hello, this is a great tutorial. Can you please let me know which text books I would have to buy to learn all about Neural Networks theory and practicals. Is there any book which can guide me to the use of MATLAB for neural networks? Thank you in advance.

Bindu , January 24, 2013 at 5:49 pm

Reply

hello

I'm wondering where did you get the algorithm for weight initialization?

thanks

Reply

I do not even know how I ended up here, but I thought this post was great. I do not know who you are but definitely you're going to a famous blogger if you are not already ☺ Cheers!

boyfriend jean , February 26, 2013 at 5:16 pm

Reply

I have been browsing online more than three hours today, yet I never found any interesting article like yours. It is pretty worth enough for me. In my view, if all website owners and bloggers made good content as you did, the web will be a lot more useful than ever before.

test site , March 16, 2013 at 12:06 am

Reply

As a comp sci prof, I have to tell you this is very well done. Quite intuitive. With your ability to explain things you might even consider expanding some of those "WTF?" areas.... well done.

Gary Newell , July 17, 2013 at 11:03 pm

Reply

can you help me develop an Artificial Neural network based induction motor speed controller

Ubaka , November 6, 2013 at 12:14 pm

Reply

Was interested until "Java IMHO for idiots". Kind of like saying "who needs a crosshead screwdriver". Just because you are not adept at the tool doesn't make it for idiots. I've had my experience on both sides of the fence and think the term "retraining" may apply ;-). I look forward to reading the rest of your "intellectual" information and may have to apply my mental filter to avoid more replies.

Chris , November 27, 2013 at 7:43 pm

Reply

Hi boby. This tutorial was really very very helpful to understand the concept of neural network.. I understood it more by your given c++ implementation. But I have just one doubt, Could you please tell me, you have considered destructor of class "neuralNetwork" but you have not considered destructor of class "neuralNetworkTrainer" why is it so ?

vivek , January 30, 2014 at 7:15 am

Reply

Pingback: [Artificial Neural Networks in MATLAB | The Control Loop](#)

Thanks so much for writing this up – I had some difficulty understanding neural networks but now it's a lot clearer.

Jonathan Young , May 4, 2014 at 1:08 am

Reply

Neural networks have always fascinated me ever since I became aware of them in the 1990s. Seeing the difficulties many have in understanding how they are programmed, I recently wrote what I believe is a simple and intuitive tutorial. This does not contain any programming hints, but is intended to clear up the mystery surrounding backpropagation, which is the principal method of programming multi-layer neural networks.

See “A Gentle Introduction to Backpropagation” <http://numericinsight.blogspot.com/2014/07/a-gentle-introduction-to-backpropagation.html>

Shashi Sathyanarayana , July 23, 2014 at 10:47 pm

Reply

Very nice and clean code... enjoying to follow the code

Tanmay Mandal , February 1, 2015 at 11:35 pm

Reply

thanks for share,i'm also trying to code in c++, but i don't used to in c++,here is my code can anyone have some time to see my problem. here i'm using gradien decent rule and delta rule, I is given value,and O is target value,i need to find error

```
#include
```

```
#include
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
double n = 0.01; // iteration value
```

```
{
```

```
double I = [-0.5 0 0.5 1 ];
```

```
double O = [0.75 2 3.25 4.5];
```

```
double w0 = 2.2;
```

```
double w1 = 4.0;
```

```
{
```

```
for (int p = 0; p < 50000; ++p)
```

```
dw0 = 0;
```

```
dw1 = 0;
```

```
}
```

```
{
```

```
for i = 1:4
```

```
{
```

```
y(i) = (w0)+((w1)*I(i, 1));
```

```
dw1 = (dw1)+(O(i, 1) - y(i))*I(i, 1);
```

```
dw0 = (dw0)+(O(i, 1) - y(i));
```

```
}
```

```
w0 = (w0)+(dw0)*n;
```

```
w1 = (w1)+(dw1)*n;
```

```
}
```

```
y1 = w1*I + w0
```

```
cout << "the output of y1" << y1 << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

ibrahim , April 15, 2015 at 5:13 pm

Reply

Pingback: [cervical torticollis neck muscles](#)

can anyone tell me what is the difference or differences between the neural network classifier and the computer learning algorithm? sorry if my question turns out obsolete

Montserrat Gerez Malo , July 29, 2015 at 3:09 am

Reply


Neural networks are a type of computer program that mimics the function of the human brain, and neural networks try to find order and logic within the data. The vast amounts of data that the FOREX markets generate are similar to the raw information that the human brain has actually to process.

neural network , December 11, 2015 at 5:29 pm

Reply

Pingback: [Deep Learning-ML Tutorials | Tony Deep Techs](#)

i am not able to compile the program .Friends can you me to get out of this situation by explaining the steps to be followed while executing these programs,as i am a beginner.

Thanks in advance. 

bravensmart , January 28, 2016 at 1:52 pm

Reply

Pingback: [\[Перевод\] Список ресурсов по машинному обучению. Часть 2 - itfm.pro](#)

Tell me how to use this package. I mean tutorial/examples/readme file is needed.

nag , April 23, 2016 at 12:44 pm

Reply

Pingback: [List of Machine Learning tutorials | Love, learn and laugh](#)

Pingback: [A Quick Introduction to Neural Networks – the data science blog](#)

Hi can you help me on how to compile and run the program. i used it on eclipse but its telling me srtcpy_s was not declared in this scope..PLEASE HELP

Abubakr Siddique Jallow , October 15, 2016 at 5:26 pm

Reply

Pingback: [神经网络快速入门: 什么是多层感知器和反向传播? | 守心斋](#)

Pingback: [神经网络快速入门: 什么是多层感知器和反向传播? -人工智能在线](#)

I am doing my mtech by Research in Machine learning techniques..Can you please kindly share me your mail id so that if in case of need i may mail you bro

V.Ellakkia , December 25, 2016 at 7:51 pm

Reply

Pingback: [A Quick Introduction to Neural Networks | Open Data Science](#)

