Using Neural Network to solve Y = X * X + b type formula

Update 1/6/2014: I've updated the question so that I'm trying to solve a non-linear equation. As many of you pointed out I didn't need the extra complexity (hidden-layer, sigmoid function, etc) in order to solve a non-linear problem.

Also, I realize I could probably solve even non-linear problems like this using other means besides neural networks. I'm not trying to write the most efficient code or the least amount of code. This is purely for me to better learn neural networks.

I've created my own implementation of back propagated neural network.

It is working fine when trained to solve simple XOR operations.

However now I want to adapt it & train it to solve Y = X * X + B type formulas, but I'm not getting expected results. After training the network does not calculate the correct answers. Are neural networks well-suited for solving algebra equations like this? I realize my example is trivial I'm just trying to learn more about neural networks and their capabilities.

My hidden layer is using a sigmoid activation function and my output layer is using an identity function.

If you could analyze my code and point out any errors I'd be grateful.

Here is my full code (C# .NET):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NeuralNetwork
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Training Network...");

            Random r = new Random();
            var network = new NeuralNetwork(1, 5, 1);
            for (int i = 0; i < 100000; i++)
            {
                int x = i % 15;
                int y = x * x + 10;
                network.Train(x);
                network.BackPropagate(y);
            }

            //Below should output 20, but instead outputs garbage
            Console.WriteLine("0 * 0 + 10 = " + network.Compute(0)[0]);

            //Below should output 110, but instead outputs garbage
            Console.WriteLine("10 * 10 + 10 = " + network.Compute(10)[0]);

            //Below should output 410, but instead outputs garbage
            Console.WriteLine("20 * 20 + 10 = " + network.Compute(20)[0]);
        }
    }

    public class NeuralNetwork
```

c# neural-network backpropagation

edited Jan 7 at 1:09

asked Jan 6 at 6:18

craigrs84

339 2 12

add comment

3 Answers

you use sigmoid as output funnction which in in the range [0-1] but you target value is double the range is [0 - MAX_INT], i think it is the basic resaon why you are getting NAN□ i update you code , and try to normalize the value in the range in [0-1], and I can get ouptut like this which is what I expect

I think I am getting close to the truth□I am not sure why this answer is getting vote down

```
Training Network...
2.62124390739838
2.41111276795694
2.28130586319731
2.19045320372317
2.11913904170007
2.05593511090139
1.99246337039587
1.92169063406158
1.83806955883833
1.73897602924367
1.62620328940286
1.50571444049788
1.38525181203309
1.27152563305166
1.16873390567891
1.0785860186953 4
1.00104833549352
0.935134189211126
0.879468887608428
0.832617094189722
0.793242393605558
0.76016847960944
0.732389660583377
0.709058546212349
0.689465600220947
0.673017612218772
0.659218076065573
0.647650440401349
0.637964269863785
0.629863984791551
0.623099743944259
0.617460049899339
0.612765716366498
0.608864905880896
0.60562901023837
0.602949199394821
0.600733506672607
0.598904350202068
0.597396414391054
0.596154832690811
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NeuralNetwork
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Training Network...");

            Random r = new Random();
            var network = new NeuralNetwork(1, 3, 1);
            for (int k = 0; k < 60; k++)
            {
                for (int i = 0; i < 1000; i++)
                {
                    double x = i / 1000.0;// r.Next();
                    double y = 3 * x;
                    network.Train(x);
                    network.BackPropagate(y);
                }
                double output = network.Compute(0.2)[0];
                Console.WriteLine(output);
            }
            //Below should output 10, but instead outputs either a very large nu
           /* double output = network.Compute(3)[0];
            Console.WriteLine(output);*/
        }
    }

    public class NeuralNetwork
    {
```

edited Jan 6 at 12:38 answered Jan 6 at 9:53


michaeltang
1,636 3 10

Thanks. Normalizing the inputs did indeed help. I'll have to do more research on why this helps. However, now that I've revised the question to be a non-linear problem the normalization seems to no longer work correctly. – craigrs84 Jan 7 at 2:56

I think it is the sigmoid function choose, when the input value is large the, the output value of the hidden layer will be 0 , so is the gradient of the hidden layer, I normalize the input to [0 - 1],the result is pretty much explainable – michaeltang Jan 7 at 10:22

add comment

You really don't need to use a multi-layer network to solve problems of ax + b = y. A single layer perceptron would do the trick.

In fact, for a problem this simple you don't even need to break out the complexity of a real neural network. Check out this blog post:

http://dynamicnotions.blogspot.co.uk/2009/05/linear-regression-in-c.html

answered Jan 6 at 23:04


John Wakefield
11 1

add comment

I did not analyze your code, it is far too long. But I can give you the answer for the basic question:

Yes, neural networks are well suited for such problems.

In fact, for a `f : R -> R` in the form of `ax+b=y` you should use **one neuron** with **linear** activation function. No three-layered structure is required, just one neuron is enough. If your code fails in such case, then you have an implementation error, as it is a simple linear regression task solved using gradient descent.

answered Jan 6 at 9:30


lejlot
11.7k 2 8 30

Please leave a comment for "-1" vote exaplining what is wrong in the provided answer – lejlot Jan 6 at 11:30

Thanks for the information. Sorry it wasn't me that downvoted. It makes sense that this type of problem solving doesn't require multi-layer and sigmoid activation. However if I switched the algebra formula from a linear function to a something like y = x*x + 1 then I would require sigmoid plus multi-layer correct? – craigrs84 Jan 6 at 15:39

Yes, once you go to the class of non-linear functions you need one hidden layer of non-linear activation functions (as the universal approximation theory states) . – lejlot Jan 6 at 15:43

For the record, I'm the one who downvoted, I must have clicked it accidentally and now the system won't let me remove it unless the answer is edited.. Sorry. – Daniel Jan 8 at 0:33

add comment

Not the answer you're looking for? Browse other questions tagged c# neural-network backpropagation **or ask your own question.**