Introduction to Machine Learning
Individual Laboration Report –6–

Erik Sven Vasconcelos Jansson
erija578@student.liu.se
Linköping University, Sweden

December 15, 2016

Finally, the last machine learning topic covered are *artificial neural networks*. These estimators are very flexible, such that even a *single layer feed-forward neural network* complies with the *universal approximation theorem*, presented by *Csáji* [Csá01]:

Theorem 1 (Universal Approximation Theorem). *Any artificial feed-forward neural network with a single hidden layer, containing a finite amount of neurons, can approximate any continuous functions on the compact subset $\mathcal{R}^n$ (with restrictions on σ).*

Proof. Csáji's [Csá01] derivation of Theorem 1. □

Basically, the theorem states that even simple neural networks can represent interesting functions, given some suitable subset of *activation functions*. For this assignment, we want to approximate $\sin x$, where we are given *25 observation* for *training set*. Also, we are given a *validation set* of length *25* for checking if our neural network is under/overfitting. We are using the *R package* `neuralnet` for our fit with *10 hidden units* in a *single hidden layer*, also initialized with *random weights* in $[-1, 1]$ interval. See Listing 1 for the entire assignment source code.

For the curious, Equations 1, 2, and 3 are given:

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

$$\boldsymbol{w}_{(i)} = \boldsymbol{w}_{(i-1)} - \eta_k \nabla E(\boldsymbol{w}_{(i-1)}) \tag{2}$$

$$\hat{y}_j(\boldsymbol{x}) = \sigma(w_0 + \sum_{h=1}^{H} \sigma(w_{0h} + \boldsymbol{w}_h^\intercal \boldsymbol{x})) \tag{3}$$

1. **Sigmoid Activation Function:** "S"-shaped function which converges $\sigma(u) = 1$ as $u \to \infty$ and $\sigma(u) = 0$ as $u \to -\infty$. Used in Equation 3.
2. **Batch Gradient Descent:** finds the "step" in the right direction for *minimizing error E*. This is achieved with the *gradient* of E given in respect to the weights $\boldsymbol{w}$; giving a *hyperplane*.
3. **Single-Layer Neural Network Estimator:** uses Equations 1 and 2 to find $\hat{y}_j$ by finding the parameters $\boldsymbol{w}$ in each layer (a linear equation) by means of *gradient descent* and producing a *non-linear result in subsequent layers* by the *activation function*. This is the primary reason why neural networks are so flexible & general.

By using a *threshold* for the *gradient descent* we can stop the *neural network* from either *overfitting* or *underfitting*. This simply done by increasing the threshold iteratively and taking the validation set's:

Threshold	**S.S.E.**
0.001	0.01367691527
0.002	0.01262419958
0.003	0.00988418900
0.004	0.00850089424
0.005	0.00955545744
0.006	0.00974372099
0.007	0.01583926857
0.008	0.01649252416
0.009	0.02112490377
0.010	0.02735909554

Table 1: Neural Network Values

After finding the "optimal threshold" of *0.004* by picking the 4^{th} iteration (where $i = 4$ that is) which gives the least amount of error for a validation set, we plot the best neural network in Figure 1, and also the predictions in Figure 2 for a *sine function*. Notice that the fit is pretty good, and the estimator gives a pretty "spot on" prediction for the function. It seems *neural networks* are incredibly powerful, but take time to train and are harder to reason about (for example, how do we choose the number of hidden layers and units? How long will it take?)

References

[Csá01] Balázs Csanád Csáji. Approximation with Artificial Neural Networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48, 2001.

[FHT09] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*. Springer series in statistics, Berlin, second (11th) edition, 2009.

[GF10] Frauke Günther and Stefan Fritsch. neuralnet: Training of Neural Networks. *The R Journal*, 2(1):30–38, 2010.

Figure 1: Neural Network

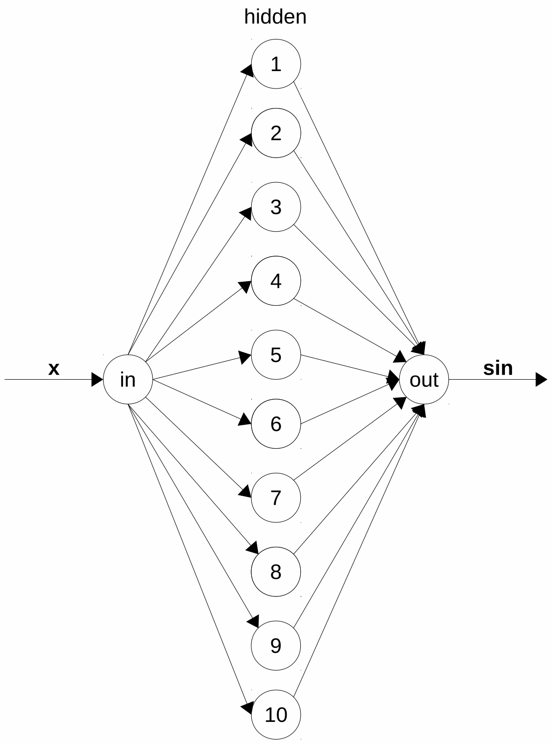
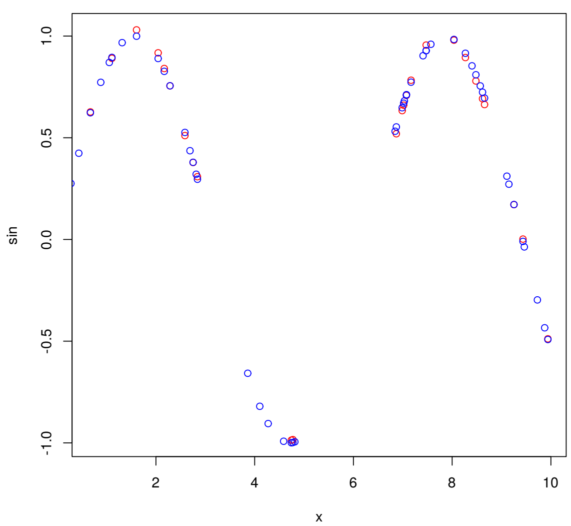


Figure 2: Neural Network's Produced Predictions (in the graph are raw values and predicted values).



Appendix

Listing 1: Feed-Forward Backpropagating Neural Network Sine Estimator Script

```
library("ggplot2")
library("reshape2")
library("neuralnet")
library("grDevices")
set.seed(1234567890)

variable <- runif(50, 0, 10)
sine <- data.frame(x=variable, sin=sin(variable))
training <- sine[1:25,] ; testing <- sine[26:50,]

candidate_error <- Inf
units <- 10 # Hidden baby!
candidate_threshold <- Inf
weights <- runif(50, -1, +1)

for (threshold_attempt in 1:10) {
    thresholdi <- threshold_attempt / 1000
    nn <- neuralnet(sin~x, training, units,
                    startweights = weights,
                    threshold = thresholdi)

    predicted <- compute(nn, testing$x)
    error <- sum((testing$sin - predicted$net.result)^2)
    cat("NN Threshold", thresholdi, "->", error, "SSE \n")
    if (error < candidate_error) {
        candidate_error = error
        candidate_threshold = thresholdi
    }
}

nn <- neuralnet(sin~x, training, units,
                   candidate_threshold,
                startweights = weights)
predicted <- compute(nn, testing$x)

plot(nn)
setEPS()
cairo_ps("predictions.eps")
plot(testing$x, predicted$net.result, col = "red",
     xlab = "x", ylab = "sin")
points(sine, col = "blue")
dev.off()
```

Listing 2: Output About the Produced Neural Network in the Assignment

```
$response
              sin
1   0.31115890803
2  -0.65787112371
3   0.85356988285
4   0.92820698816
5   0.71194544538
6   0.95969186755
7   0.27531467859
8  -0.03662256168
```

```
9  -0.29718457265
10 -0.43427724087
11  0.27176755816
12  0.96762993527
13  0.87023024548
14  0.90319426880
15  0.53211225475
16 -0.90515370065
17 -0.99209419164
18  0.75493516282
19  0.43639270658
20  0.42400734122
21  0.77254200174
22  0.68138797265
23  0.32070401674
24 -0.99484612705
25 -0.82027249428

$covariate
              [,1]
 [1,] 9.1083657276
 [2,] 3.8595812093
 [3,] 8.4019783861
 [4,] 7.4727497180
 [5,] 7.0754500036
 [6,] 7.5690891198
 [7,] 0.2789170155
 [8,] 9.4614087138
 [9,] 9.7265206091
[10,] 9.8740137112
[11,] 9.1495487187
[12,] 1.3156641065
[13,] 1.0556694935
[14,] 7.4103393406
[15,] 6.8442786904
[16,] 4.2733338126
[17,] 4.5865617390
[18,] 8.5692227143
[19,] 2.6900070859
[20,] 0.4378655413
[21,] 0.8828348410
[22,] 7.0328426105
[23,] 2.8151199827
[24,] 4.8139597056
[25,] 4.1034799209

$err.fct
function (x, y)
{
    1/2 * (y - x)^2
}
<environment: 0x339a758>
attr(,"type")
[1] "sse"

$act.fct
function (x)
{
    1/(1 + exp(-x))
}
<environment: 0x339a758>
attr(,"type")
```

```
[1] "logistic"

$linear.output
[1] TRUE

$data
               x          sin
1  9.1083657276  0.31115890803
2  3.8595812093 -0.65787112371
3  8.4019783861  0.85356988285
4  7.4727497180  0.92820698816
5  7.0754500036  0.71194544538
6  7.5690891198  0.95969186755
7  0.2789170155  0.27531467859
8  9.4614087138 -0.03662256168
9  9.7265206091 -0.29718457265
10 9.8740137112 -0.43427724087
11 9.1495487187  0.27176755816
12 1.3156641065  0.96762993527
13 1.0556694935  0.87023024548
14 7.4103393406  0.90319426880
15 6.8442786904  0.53211225475
16 4.2733338126 -0.90515370065
17 4.5865617390 -0.99209419164
18 8.5692227143  0.75493516282
19 2.6900070859  0.43639270658
20 0.4378655413  0.42400734122
21 0.8828348410  0.77254200174
22 7.0328426105  0.68138797265
23 2.8151199827  0.32070401674
24 4.8139597056 -0.99484612705
25 4.1034799209 -0.82027249428

$net.result
$net.result[[1]]
              [,1]
1   0.30018554412
2  -0.64466078637
3   0.82577426828
4   0.95531707389
5   0.71041413678
6   0.98604140527
7   0.27220011825
8  -0.02393016651
9  -0.27977135370
10 -0.42452441939
11  0.26376340300
12  0.97423750701
13  0.86489547875
14  0.92926095080
15  0.49438343511
16 -0.91926885456
17 -0.98908629285
18  0.72304892189
19  0.42816650637
20  0.43013044493
21  0.76891173652
22  0.67393025023
23  0.32904062832
24 -0.97934291894
25 -0.83193245662
```

```

$weights
$weights[[1]]
$weights[[1]][[1]]
             [,1]          [,2]         [,3]         [,4]         [,5]
[1,] 0.3718763846 -10.931812117  8.275060257 7.8216380497  1.551228805
[2,] 0.5081317757   1.628735531 -2.289303563 0.1166254588 -0.594052483
              [,6]          [,7]          [,8]          [,9]        [,10]
[1,]  4.7453831203 -0.6070429987 9.38110372186 -0.1377222063  5.958245683
[2,] -0.4968713811  0.1924524647 0.09270470267  3.1685937697 -2.602280174

$weights[[1]][[2]]
                [,1]
 [1,] -0.06529714022
 [2,] -0.70033511720
 [3,]  3.84669442716
 [4,]  2.74586539382
 [5,] -0.75978348453
 [6,] -9.09090264177
 [7,]  6.65416477397
 [8,] -8.24869453812
 [9,] -0.20335986240
[10,]  1.00884789102
[11,]  2.01492912933



$startweights
$startweights[[1]]
$startweights[[1]][[1]]
             [,1]          [,2]          [,3]         [,4]          [,5]
[1,] 0.4591262657 -0.5031667114  0.9014065554 0.9589186665  0.3389983536
[2,] 0.5853618421  0.4307389595 -0.8788680169 0.4978831881 -0.5358421607
              [,6]          [,7]         [,8]           [,9]         [,10]
[1,]  0.6293357364 -0.4028865024 0.5968314419 -0.07648990629  0.7421438890
[2,] -0.4464168334 -0.3094406570 0.9041418806 -0.17025629710 -0.8588890089

$startweights[[1]][[2]]
               [,1]
 [1,] -0.2698646844
 [2,] -0.9254528601
 [3,]  0.3498687637
 [4,] -0.7230960354
 [5,] -0.9836924160
 [6,] -0.6452815034
 [7,]  0.8491520174
 [8,]  0.6410233583
 [9,] -0.4020887311
[10,]  0.9406797229
[11,]  0.2142777084



$generalized.weights
$generalized.weights[[1]]
             [,1]
1  -4.18437409317
2   0.84720544205
3  -3.90202008113
4   8.87306916010
5   4.06386829427
6  18.83336456829
```

```
7   5.28857232424
8  38.75962317972
9   2.72921588482
10  1.62852618574
11 -4.58128912031
12 13.07325336492
13  4.31497701909
14  6.93919826366
15  4.07243773606
16  0.23010347764
17  0.02853630559
18 -3.31477743585
19 -3.26159533671
20  3.80902182260
21  3.41516125086
22  3.98677345674
23 -3.57280260613
24 -0.06878886818
25  0.40865817424


$result.matrix
                                            1
error                          0.003576080337
reached.threshold              0.003929680826
steps                      23174.000000000000
Intercept.to.1layhid1          0.371876384634
x.to.1layhid1                  0.508131775660
Intercept.to.1layhid2        -10.931812117300
x.to.1layhid2                  1.628735530657
Intercept.to.1layhid3          8.275060257474
x.to.1layhid3                 -2.289303563425
Intercept.to.1layhid4          7.821638049688
x.to.1layhid4                  0.116625458773
Intercept.to.1layhid5          1.551228805249
x.to.1layhid5                 -0.594052483023
Intercept.to.1layhid6          4.745383120333
x.to.1layhid6                 -0.496871381057
Intercept.to.1layhid7         -0.607042998673
x.to.1layhid7                  0.192452464714
Intercept.to.1layhid8          9.381103721859
x.to.1layhid8                  0.092704702673
Intercept.to.1layhid9         -0.137722206303
x.to.1layhid9                  3.168593769723
Intercept.to.1layhid10         5.958245682591
x.to.1layhid10                -2.602280174422
Intercept.to.sin              -0.065297140222
1layhid.1.to.sin              -0.700335117198
1layhid.2.to.sin               3.846694427157
1layhid.3.to.sin               2.745865393824
1layhid.4.to.sin              -0.759783484527
1layhid.5.to.sin              -9.090902641770
1layhid.6.to.sin               6.654164773966
1layhid.7.to.sin              -8.248694538124
1layhid.8.to.sin              -0.203359862396
1layhid.9.to.sin               1.008847891021
1layhid.10.to.sin              2.014929129330

attr(,"class")
[1] "nn"
```