THE UNIVERSITY OF TEXAS AT AUSTIN

McCOMBS BUSINESS ANALYTICS

A Super Quick Intro to Neural Nets

Jared S. Murray
The University of Texas McCombs School of Business

Neural Nets

For basic neural nets, the inputs (x_j's) need to be numeric (like linear regression)

Most common solution: "One-hot" encodings (k dummies for a categorical variable with k levels)

Specific neural network architectures for e.g. mapping images to numbers ("embedding") exist but are beyond our scope.

Zagat data

Here is the zagat data:

```
zag = read.table("zagat.txt",header=T)
summary(zag)
      food            decor          service           price      
 Min.   :14.00   Min.   : 2.00   Min.   :10.00   Min.   :11.00  
 1st Qu.:18.00   1st Qu.:14.00   1st Qu.:16.00   1st Qu.:25.00  
 Median :20.00   Median :16.50   Median :18.00   Median :32.50  
 Mean   :19.61   Mean   :16.58   Mean   :17.77   Mean   :33.32  
 3rd Qu.:21.00   3rd Qu.:20.00   3rd Qu.:20.00   3rd Qu.:41.00  
 Max.   :27.00   Max.   :28.00   Max.   :26.00   Max.   :65.00  
```

Let's rescale so that each x is in (0,1).

nnet package... One layer example

First you have to load the neural net library, nnet:

```
> library(nnet)
```

Here is the command:

```
> znn = nnet(price~food,zagsc,size=3,decay=.1,linout=T)
```

As usual, a data structure is returned containing
(in some possibly obscure way!!)
the results.

The first two arguments are familiar.

linout=T is appropriate for a numeric y.

size, decay, ...

size and decay, are the two key parameters for controlling the flexibility of the neural net fit.

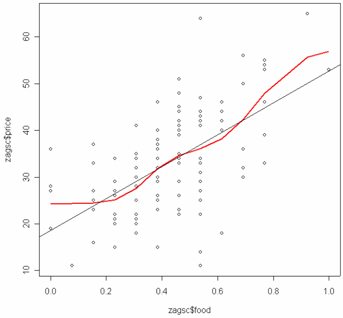
After we understand the basic structure of the model we will discuss these.

These will be the parameters that control the complexity of the model like k in KNN and λ in the LASSO.

Let's have a look at the fits.
Just as with trees and regression,
we use the predict command:

```
> fznn = predict(znn,zagsc)
> plot(zagsc$food,zagsc$price)
> oo = order(zagsc$food)
> lines(zagsc$food[oo],fznn[oo],col="red",lwd=2)
> abline(lm(price~food,zagsc)$coef)
```

zzn: nnet fit
zagsc: data frame with scaled x's.



The red is the nn fit and the straight line is linear regression.

What is the structure of the model ?

```
> summary(znn)
a 1-3-1 network with 10 weights
options were - linear output units  decay=0.1
 b->h1 i1->h1
  4.35  -0.24
 b->h2 i1->h2
 -7.42  21.41
 b->h3 i1->h3
 -9.93  13.28
 b->o h1->o h2->o h3->o
12.33 12.09 10.70 22.74
```

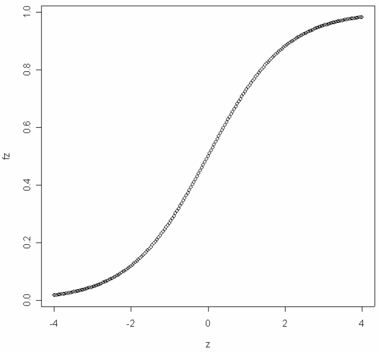
What does this mean ?

First note:

Let, $F(z) = \dfrac{e^z}{1+e^z}$

```
> z =  (-100:100)/25
> fz  = exp(z)/(1+exp(z))
> plot(z,fz)
```

This F is often called the logistic function.



Let, $F(z) = \dfrac{e^z}{1+e^z}$

food

z_1=4.35-.24food z_2=-7.42-21.41food z_3=-9.93-13.28food

$$y = 12.33 + 12.09F(z_1) + 10.70F(z_2) + 22.74F(z_3)$$

1. Form several different linear functions of the x's.
2. Apply the logistic function to each.
3. Take a linear combination of the results of 2.

The z's are called the *hidden layer*.

Each of the z's (linear function) is called a unit.

In the call to nnet the parameter "size" is the number of units in the hidden layer.

Here is the fit of a neural net with 5 units in the hidden layer.

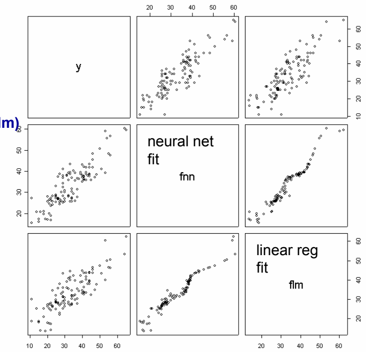
```
> znn = nnet(price~food,zagsc,size=5,decay=.1,linout=T)
> summary(znn)
a 1-5-1 network with 16 weights
options were - linear output units  decay=0.1
 b->h1 i1->h1
  2.70   0.40
 b->h2 i1->h2
 -9.69  13.02
 b->h3 i1->h3
  0.71  -5.99
 b->h4 i1->h4
 -6.63  19.76
 b->h5 i1->h5
  2.70   0.39
 b->o h1->o h2->o h3->o h4->o h5->o
 7.41  7.00 23.30  7.62 13.56  6.99
```

the coefficients for the 5 linear functions of x, the 5 z's.

the coefficients for the five f(z)

All three x's

```
> znn = nnet(price~.,zagsc,size=5,decay=.1,linout=T)
> fznn = predict(znn,zagsc)
>
> zlm = lm(price~.,zagsc)
> fzlm = predict(zlm,zagsc)
>
> temp = data.frame(y=zagsc$price,fnn=fznn,flm=fzlm)
> pairs(temp)
>
> print(cor(temp$y,temp$fnn))
[1] 0.867858
> print(cor(temp$y,temp$flm))
[1] 0.829138
> print(cor(temp$fnn,temp$flm))
[1] 0.9705388
```



The fitted model with 3 x’s and 5 units in the hidden layer.

a 3-5-1 network. 3 x’s, 5 units, 1 y.

```
> summary(znn)
a 3-5-1 network with 26 weights
options were - linear output units  decay=0.1
 b->h1 i1->h1 i2->h1 i3->h1
 -5.64  -2.64  11.48   8.31
 b->h2 i1->h2 i2->h2 i3->h2
-18.09  20.98  19.53  -0.64
 b->h3 i1->h3 i2->h3 i3->h3
  1.45  -4.79   1.95   1.00
 b->h4 i1->h4 i2->h4 i3->h4
  1.44  -0.94  -7.64   3.35
 b->h5 i1->h5 i2->h5 i3->h5
-20.40   9.93  14.09   4.48
  b->o  h1->o  h2->o  h3->o  h4->o  h5->o
  5.15  13.15  13.33   6.75  12.51  24.42
```

# of weights = 4*5 + 6.

The General Model

A k-m-1 network.

X

$$z_1 = \beta_{10} + \beta_{11}x_1 + \cdots + \beta_{1k}x_k \qquad z_2 = \beta_{20} + \beta_{21}x_1 + \cdots + \beta_{2k}x_k \qquad \ldots\ldots \qquad z_m = \beta_{m0} + \beta_{m1}x_1 + \cdots + \beta_{mk}x_k$$

$$\hat{y} = \beta_0 + \beta_1 F(z_1) + \cdots + \beta_m F(z_m)$$

k x’s
m hidden units.

Why on earth, would this work ?

Size and Decay

The size of the neural net is the number of units in the hidden layer.

Clearly, the more units the richer the model.

The more we are able to fit the data.

The more we are able to overfit the data.

Size and Decay

The decay parameter is the L2 regularlization parameter.

Fit minimizes:

$$\text{error} + \text{decay} \, * \, \sum \text{coefficient}^2$$

where, for example,

$$\text{error} = \sum (y_i - \hat{y}_i)^2.$$

Size and Decay

Whether a coefficient is large or small depends on the units of the x's.

This is the fundamental reason we rescale the x's.

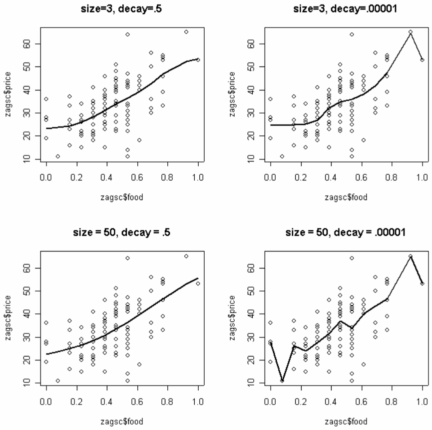
Only if the x's are on the same scale does the decay parameter work properly.

People have found that in practice the decay parameter is useful for walking the fit/overfit line.

Size and Decay

Left to right we can see that lower decay means a more flexible fit, the coefficients are freer.

With low decay (right two plots) increasing the size really frees up the fit.



With high decay adding more units does not seem to hurt !!

Size and Decay

We also see that even with 50 hidden units,
a large decay parameter can restrain the fit.

In practice, this has lead to the following strategy
for fitting neural nets.

1. Fix a large number of hidden units.

2. Use the three set approach or cross validation
   to choose the decay parameter.

Of course, you could use cv or three sets to
choose both size and decay.

How does it work?

How could this possibly work???!!!

Let's fit a few simple examples and see how the pieces add up to the overall fit.

How does it work?

Let's calculate the pieces for a simple example and see how they can add up to a decent fit.

```
x = zagsc$food
y = zagsc$price

z1 = 4.35  -0.24 *x
z2 = -7.42 +21.41*x
z3 = -9.93 +13.28*x
```

The three linear functions of x, the z's.

```
f1 = 12.09*exp(z1)/(1+exp(z1))
f2 = 10.7*exp(z2)/(1+exp(z2))
f3 = 22.74*exp(z3)/(1+exp(z3))
```

*coefficent * f(z)*

```
plot(x,y-12.33)
lines(x[oo],f1[oo],col=2)
lines(x[oo],f2[oo],col=3)
lines(x[oo],f3[oo],col=4)
lines(x[oo],(f1+f2+f3)[oo],col=5)
```

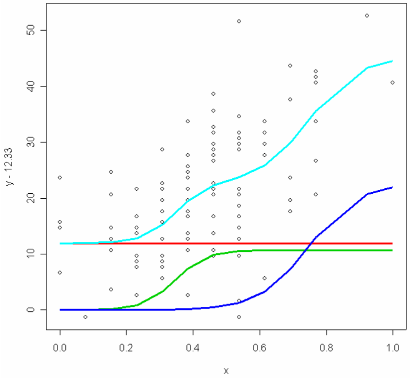
```
b->h1 i1->h1
 4.35  -0.24
b->h2 i1->h2
-7.42  21.41
b->h3 i1->h3
-9.93  13.28
b->o h1->o h2->o h3->o
12.33 12.09 10.70 22.74
```

How does it work?

cyan:fit

red:first component
blue:second
green:third

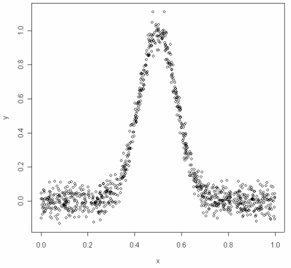
Wow,
scary and
cool !



How does it work?

How would you fit a bump?

```
set.seed(23)
x = runif(1000)
x = sort(x)
y = exp(-80*(x-.5)*(x-.5)) + .05*rnorm(1000)
plot(x,y)
df = data.frame(y=y,x=x)
```



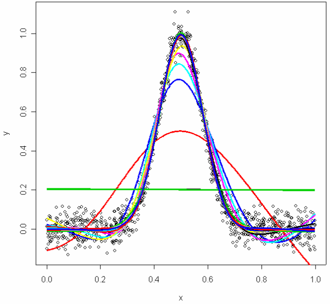
How does it work?

```
plot(x,y)
sz =  3
for(i in 1:20) {
nnsim = nnet(y~x,df,size=sz,decay = 1/2^i,linout=T,maxit=1000)
simfit = predict(nnsim,df)
lines(x,simfit,col=i,lwd=3)
print(i)
readline()
}
```

Try various decay values.

With 3 units
it takes a
small decay.

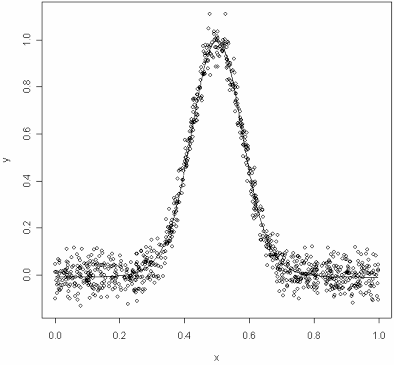
decay = 1/2^12
works.



How does it work?

```
nnsim = nnet(y~x,df,size=3,decay=1/2^12,linout=T,maxit=1000)
thefit = predict(nnsim,df)
plot(x,y)
lines(x,thefit)
```

Plot with
nn fits.
Pretty good.



Here is the fitted model:

```
> summary(nnsim)
a 1-3-1 network with 10 weights
options were - linear output units  decay=0.0002441406
 b->h1 i1->h1
  5.26 -13.74
 b->h2 i1->h2
 -6.58  13.98
 b->h3 i1->h3
 -9.67  17.87
  b->o  h1->o  h2->o  h3->o
 -2.20   2.21   7.61  -5.40
```

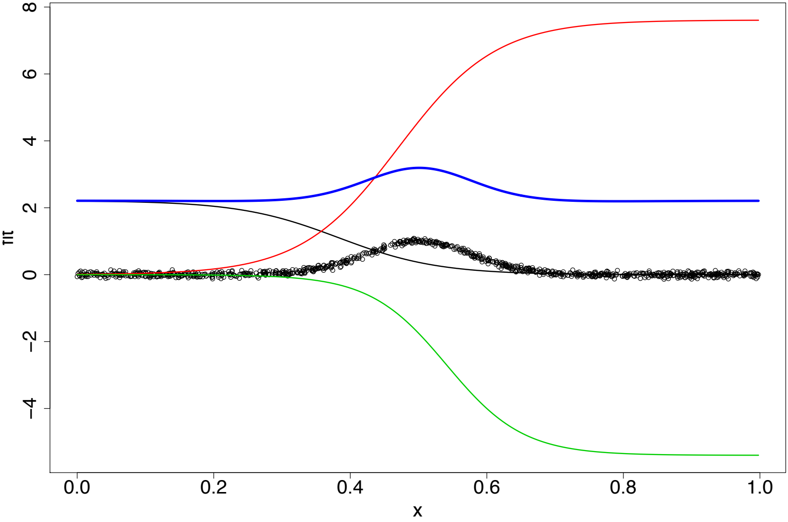
Add up the pieces:

```
F = function(x) {return(exp(x)/(1+exp(x)))}

z1 =  5.26 - 13.74*x
z2 = -6.58 + 13.98*x
z3 = -9.67 + 17.87

f1 = 2.21*F(z1)
f2 = 7.61*F(z2)
f3 = -5.40*F(z3)
```

```
rx=range(x)
ry = range(c(f1,f2,f3,y))
plot(rx,ry,type="n",xlab="x",ylab="fit",cex.axis=2,cex.lab=2)
points(x,y)
lines(x,f1,col=1,lwd=2)
lines(x,f2,col=2,lwd=2)
lines(x,f3,col=3,lwd=2)
lines(x,f1+f2+f3,col=4,lwd=4)
```



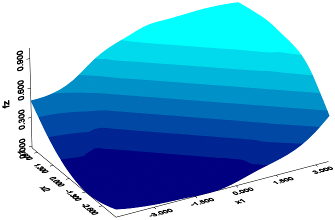
8
6
4
2
0
−2
−4
π
0.0
0.2
0.4
0.6
0.8
1.0
x

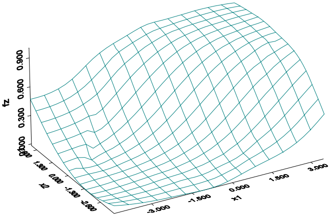
More than one x?

With more than one x it is a little harder to see how this works.

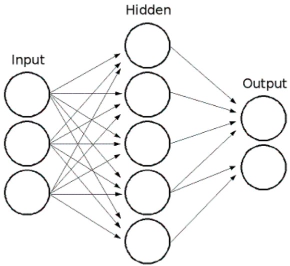
For each z, we get “ridge functions”.

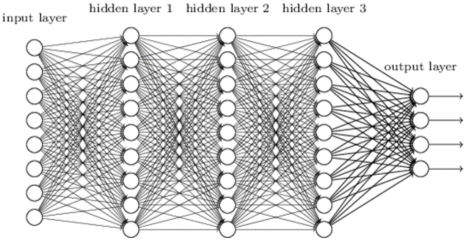
F(x1+x2)





Single Layer vs. Deep Neural Nets





Some comments

- Fitting neural nets is no trivial task! Stability of output is an issue...
- Approximating functions with "deep" nets can be easier that wide single nets... somehow a better navigation of the bias-variance trade-off
- DNN are very popular these days... they seem to work best in highly non-linear but low-noise problems (think images)... it is unclear how successful they are in high-noise social science/economics type of applications.