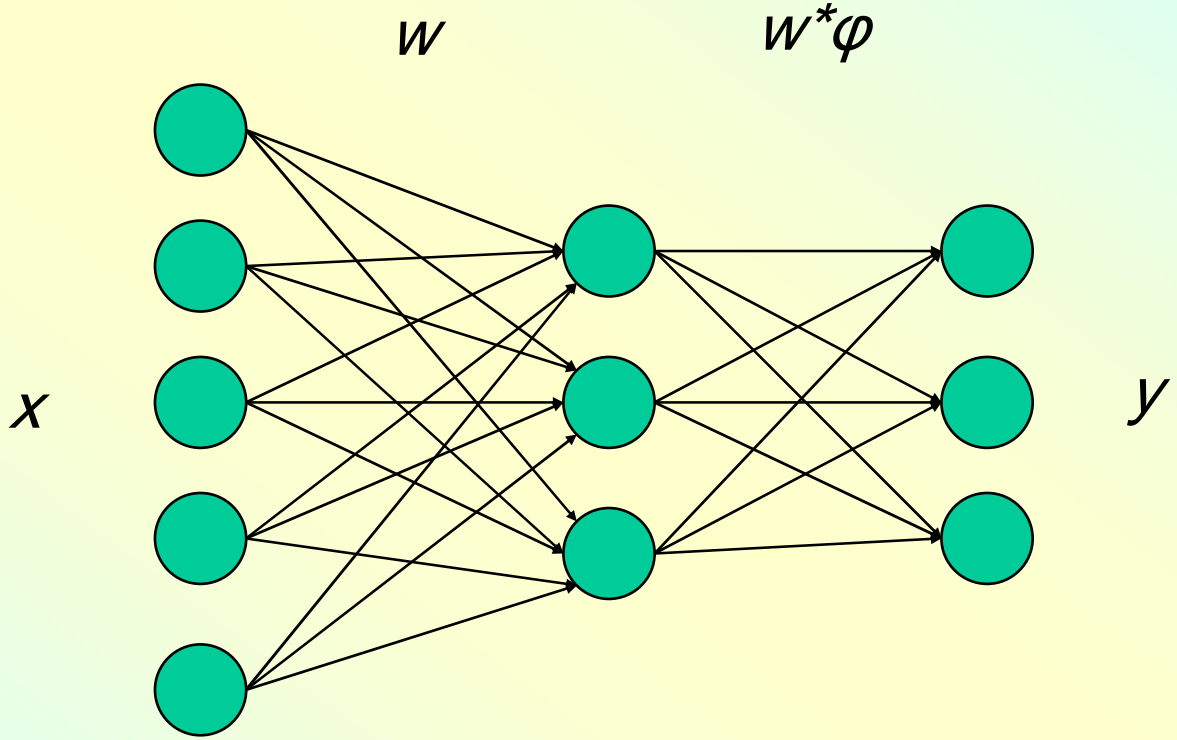
Session 14

Demystifying Neural Networks

Overview

- The model:
  - An input node for every determining variable
  - A specified number of internal nodes
  - An output node for each component of the result
- Form linear functions of the input variables for each internal layer
- Transform and form linear functions for each node of the output layer
- Add in ‘skip layer’ linear functions and transform again

$$y_k = \phi_0\left(\alpha_k + \sum_{i\to k} w_{ik}x_i + \sum_{j\to k} w_{jk}\phi_j\left(\alpha_j + \sum_{i\to j} w_{ij}x_i\right)\right)$$



5-3-3 Neural net, no skip layer

Regression function

- We think of these as regression equations:

$$y_k = f_k\left(\mathbf{w};\mathbf{x}\right)$$

- Internal layer transformations: logistic

$$\phi_h(z) = \frac{\exp(z)}{1+\exp(z)}$$

- Output transformations: Linear, logistic or threshold
- Estimation criteria: OLS, Logistic or 'softmax' (p. 245)

Penalized estimation

- General idea: minimize

$$E + \lambda C(f)$$

- Scale the x-variables to be in [0,1] and measure 'roughenss' as

$$C(f) = \sum_{l,m} w_{lm}^2$$

- Entropy fit: $\lambda \approx 10^{-2} - 10^{-1}$
- Least squares fit: $\lambda \approx 10^{-4} - 10^{-2}$

Special cases

- No internal nodes, no penalties, linear output function
  - (Multivariate) linear regression (the hard way)
- No internal nodes, no penalties, logistic output function, one output node
  - Logistic regression (the hard way)
- No skip layer, one internal node, Kullback-Liebler objective function with no penalties
  - Logistic regression again
- No internal nodes, 'softmax' objective function, no penalties:
  - Multiple logistic, multinomial regression

Linear Regression the Hard Way: a check

```
tst <- nnet(1000/MPG.city ~ Weight+Origin+Type,
  Cars93, linout = T, size = 0, decay = 0, rang =
  0, skip = T, trace = T)
# weights:  8
initial  value 213926.872885
iter  10 value 1474.564153
final  value 1461.849294
converged
coef(tst)
      b->o      i1->o      i2->o    i3->o      i4->o
 6.079536 0.01337819 -0.6014517 2.225679 -0.2456949

      i5->o      i6->o     i7->o
 0.07327528 -0.2431229 0.3443452
```

Check

```
tst1 <- lm(1000/MPG.city ~ Weight + Origin+Type,
  Cars93)
coef(tst1)
 (Intercept)    Weight     Origin    Type1      Type2
    6.079531 0.0133782 -0.6014518 2.225679 -0.2456951

      Type3      Type4     Type5
 0.07327431 -0.2431231 0.3443448

range(coef(tst) - coef(tst1))
[1] -7.084505e-009  4.709652e-006
```

Logistic regression is a special case

- Birth weight data again

```
tst <- nnet(low ~ ptd + ftv/age, data = bwt,
   entropy = T, skip = T, size = 0, decay = 0,
   rang = 0, trace = T)
# weights:  7
initial  value 131.004817
iter  10 value 101.713567
final  value 101.676271
converged
tst1 <- glm(low ~ ptd + ftv/age, binomial, bwt)
range(coef(tst) - coef(tst1))
[1] -0.0001691513  0.0003218216
```

Birth weight example, continued.

- Set up train/test and start with a parametric model:

```
sb1 <- sample(1:nrow(bwt), 100)
bwt.train <- bwt[sb1,  ]
bwt.test <- bwt[ - sb1,  ]
bm.train <- update(tst1, data = bwt.train)
bm.tst <- predict(bm.train, bwt.test, type =
  "resp")
bm.class <- round(bm.tst)
table(bm.class, bwt.test$low)
bm.class  0  1
       0 57 15
       1  6 11
```

Now consider a tree model

```
require(tree)
tm.train <- tree(factor(low) ~ race + smoke + age
  + ptd + ht + ui + ftv, bwt.train)
plot(cv.tree(tm.train, FUN = prune.misclass))
tm.train <- prune.tree(tm.train, best = 4)
tm.class <- predict(tm.train, bwt.test, type =
  "class")
table(tm.class, bwt.test$low)
tm.class  0  1
       0 40 16
       1 23 10
```

Some initial explorations of a NN

- Not clear what degree of non-linearity is warranted, but try some:

```
X0 <- max(abs(model.matrix( ~ race + smoke + age + ptd
   + ht + ui + ftv, bwt.train)))

nm.train <- nnet(low ~ race + smoke + age + ptd + ht +
   ui + ftv, data = bwt.train, size = 3, entropy = T,
   rang = 1/X0, skip = T, decay = 0.01, trace = T, maxit
   = 1000)
# weights:  43
initial  value 68.527783
iter  10 value 52.906990
iter  20 value 48.319329
…
iter 290 value 25.683421
iter 300 value 25.683393
final  value 25.683390
converged
```

Test data

- Normalisations are somewhat tedious:

```
nm.tst <- predict(nm.train, newdata = bwt.test,
   type = "raw")
nm.class <- round(nm.tst)
table(nm.class, bwt.test$low)

nm.class  0  1
       0 49 14
       1 14 12

testPred2(nm.train, bwt.test)
[1] 35.95506
```

A more challenging test: credit cards

```
CC.nnet <- nnet(credit.card.owner ~ ., CCTrain,
  size = 7, rang = 2e-7, skip = T, decay = 0.05,
  trace = T, maxit = 1000)

testPred2(CC.nnet)

[1] 15.92593
```

- Not as good as random forests, better than trees and parametric models.