- Comment the code on slide 3 class 16 slides
- On slide 4 comment only on how slide 4 generalizes the code in slide 3.
- Look at NOAAnewA, this is missing the last two years of NOAAnew
- Make NOAAnewB to match NOAAnewA but with those added two years and run the greedy neural net optimizer on it to fit the regression.

CODE WITH COMMENTS:

```
# function to train a neural network with a single hidden layer
my.neuralnet<-function(X1,Y,hidden=3,output="linear"){
  #adds a column of 1s to the input X1
  X<-cbind(1,X1)
  #initializes length of the input
  input.layer.length<-length(X[1,])
  #initializes weights with random normal values
  w01<-rnorm(input.layer.length*hidden)
  w02<-rnorm(hidden)
  w0<-c(w01,w02)
  print(w0)

  #the optimization function to optimize the weights using conjugate gradient method
  myoptfunc<-
    function(w0){my.eval2.nnet(w0,X,Y,hidden,output)$llik}
  dum<-optim(w0,myoptfunc,method="CG")
  wfinal<-dum$par
  ss<-dum$val
  #returns predicted values by evaluating inputs using the optimized weights
  pred<-my.eval2.nnet(wfinal,X,Y,hidden,output)$pred
  #plots predicted vs orginal Y values
  plot(pred,Y)
  #returns list of optim values (ss) and final weights
  list(ss=ss,wfinal=wfinal)
}
#function to evaluate neural net mod for a single input and given weight
my.eval1.nnet<-function(Xrow,w0,hidden,output){
    input.layer.length<-length(Xrow)
    # seperate weights for connections between layers (input, hidden, output)
    w01<-w0[c(1:(length(Xrow)*hidden))]
    w02<-w0[-c(1:(length(Xrow)*hidden))]
    w1<-matrix(w01,input.layer.length,hidden)
    print(w1)
    print(Xrow)
    print(w02)
```

```
    #calculates the weigted sum of the input
    xhidden<-t(Xrow)%*%w1
    #apply the activation function (logistic sigmoid)
    zhidden<-my.logistic(xhidden)
    #calculates the weigted sum of the hidden units
    out<-sum(w02*zhidden)
    #apply the activation func if output is binary
    if(output=="binary"){
      out<-my.logistic(out)
    }
    out
  }
#function to eval neural net mod for a set of inputs and a given weight vec
my.eval2.nnet<-function(w0,X,Y,hidden,output){
    zfunc<-
      function(V){my.eval1.nnet(V,w0,hidden,output)}
    #apply the model to inputs
    pred<-apply(X,1,zfunc)
    #calculate the loss values (we are trying to minimize these)
    if(output=="binary"){
      llik<-(-1)*sum(log(pred)*Y+log(1-pred)*(1-Y))
    }else{
      llik<-sum((pred-Y)^2)
    }
    #return predicted values and loss values
    list(llik=llik,pred=pred,y=Y)
}
#logistic sigmoid activation function
#to allow the network to learn complex nonlinear relationships between input and output var
my.logistic<-function(z){exp(z)/(1+exp(z))}
```

EXPLANATION on how slide 4 generalizes the code in slide 3:

```
#multilayer with regularization
#the following code generalizes the code above by adding two new parameters
#1. num.layers which allows the user to specify the number of hidden layers
#this means that all of the functions that we used previously have been modified
#to handle multiple hidden layers.
#2. Lambda which encourages the optimization function to choose smaller values for the
#weights to reduce the risk of overfitting.
my.neuralnet.multilayer<-
  function(X1,Y,hidden=3,output="linear",num.layers=1,lambda=0)
  {
    X<-cbind(1,X1)
    input.layer.length<-length(X[1,])
```

```
  w01<-rnorm(input.layer.length*hidden+(num.layers-
                                    1)*hidden*hidden)
  w02<-rnorm(hidden)
  w0<-c(w01,w02)
  #print(w0)
  myoptfunc<-
    function(w0){my.eval2.nnet.ml(w0,X,Y,hidden,output,num.layers,lambda)$llik}
  dum<-optim(w0,myoptfunc,method="CG")
  wfinal<-dum$par
  ss<-dum$val
  duh<-
    my.eval2.nnet.ml(wfinal,X,Y,hidden,output,num.layers,lambda)
  plot(duh$pred,Y,
       main=paste("llik=",duh$llik,"\nSS=",ss),xlab="Prediction")
  list(ss=ss,wfinal=wfinal,ll=duh$llik)
 }
my.eval1.nnet.ml<-
 function(Xrow,w0,hidden,output,num.layers=1){
  input.layer.length<-length(Xrow)
  w01<-w0[c(1:(length(Xrow)*hidden))]
  w0A<-w0[-c(1:(length(Xrow)*hidden))]
  w1<-matrix(w01,input.layer.length,hidden)
  xhidden<-t(Xrow)%*%w1
  zhidden<-my.logistic(xhidden)
  if(num.layers==1){
    w02<-w0A
  }
  else{
    nlayers<-num.layers
    while(nlayers>1){
      w01<-w0A[c(1:(hidden*hidden))]
      w0A<-w0A[-c(1:(hidden*hidden))]
      w01<-matrix(w01,hidden,hidden)
      xhidden<-(zhidden)%*%w01
      zhidden<-my.logistic(xhidden)
      nlayers<-nlayers-1
    }
  }
  w02<-w0A
  out<-sum(w02*zhidden)
  if(output=="binary"){
    out<-my.logistic(out)
  }
  out
```

```
 }
my.eval2.nnet.ml<-
 function(w0,X,Y,hidden,output,num.layers,lambda)
 {
  zfunc<-
   function(V){my.eval1.nnet.ml(V,w0,hidden,output,
                                num.layers)}
  pred<-apply(X,1,zfunc)
  if(output=="binary"){
   llik<-(-1)*sum(log(pred)*Y+log(1-pred)*(1-Y))
  }else{
   llik<-sum((pred-Y)^2)
  }
  loglik<-llik
  llik<-llik+lambda*sum(w0^2)
  list(llik=llik,pred=pred,y=Y,loglik=loglik)
 }
```

RUNNING GREEDY NEURAL NET OPTIMIZER:

```
my.greedy.neuralnet<-
 function(X0,Y,hidden=3,output="linear",num.layers=1,lambda=0,trials=15)
{
 par(mfrow=c(4,4))
 dum1<-my.neuralnet.multilayer(X0,Y,hidden,output,num.layers,lambda)
 print(c(dum1$ss,dum1$ll))
 for(i in 1:(trials-1)){
  dum2<-my.neuralnet.multilayer(X0,Y,hidden,output,num.layers,lambda)
  print(c(dum2$ss,dum2$ll))
  if(dum2$ss<dum1$ss){
   dum1<-dum2
  }
 }
 dum0<-
  my.eval2.nnet.ml(dum1$wfinal,cbind(1,X0),Y,hidden,output,num.layers,lambda)
 plot(dum0$pred,Y,
     main=paste("llik=",dum0$llik,"\nSS=",dum1$ss),xlab="Prediction")
 dum1
}
```

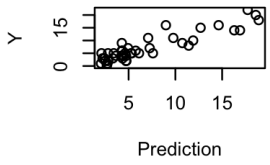
#Look at NOAAnewA, this is missing the last two years of NOAAnew
#Make NOAAnewB to match NOAAnewA but with those added two years and run the greedy neural net optimizer on it to fit the regression.

```
>new_row <- c(4.1, 20, 0.85, (4.1*4.1), (0.85*0.85),(4.1*0.85))
```

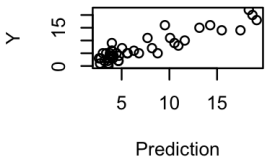
```
>NOAAnewB <- rbind(NOAAnewB, new_row)
>new_row <- c(4.2, 18, 0.89, (4.2*4.2), (0.89*0.89),(4.2*0.89))
>NOAAnewB <- rbind(NOAAnewB, new_row)
>NOAAnewB<-as.matrix(NOAAnewB)
>my.greedy.neuralnet(NOAAnewB[,c(-2)],NOAAnewB[,2],hidden=4,num.layers=2,lambda=0.003
,63)

[1] 214.2761 214.6699
[1] 214.2091 214.2541
[1] 224.7212 224.9631
[1] 241.9927 242.0952
[1] 468.7262 471.2786
[1] 200.9483 200.9916
[1] 247.8898 247.9384
[1] 258.9508 260.4954
[1] 284.5578 284.5617
[1] 283.5078 283.8125
[1] 225.9936 226.0048
[1] 251.2856 251.3699
[1] 207.2809 207.3271
[1] 205.6656 206.9910
[1] 235.4188 235.4417
$ss
[1] 200.9483

$wfinal
 [1]  1.49546408  2.67089523 -1.50697240
 [4] -1.52524379 -0.52380866  2.32275002
 [7] -0.39140251 -0.08299143 -0.34213275
[10]  2.66285487  0.73559492 -0.37185815
[13]  3.77872511  3.11366274  0.73653505
[16] -0.77425670 -0.13689247 -1.22997466
[19]  0.32655230 -1.12367493 -0.81676885
[22]  1.00229518 -0.03583663  2.17453139
[25] -1.46286059  2.72956794 -2.96226969
[28]  0.03155177 -2.35339643  0.76835467
[31] -1.35083652  1.20371639 -0.92462835
[34]  1.11403106 -1.37063985  1.41015459
[37] -1.74957819  1.77007460 -4.87370398
[40]  2.16330476  5.73113342  4.72725768
[43]  3.97830286  7.32867332

$ll
[1] 200.9916
```

REGRESSION PLOTS:

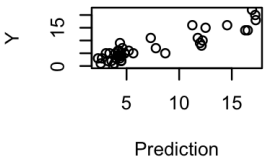
llik= 214.669885463909
SS= 214.276050298633



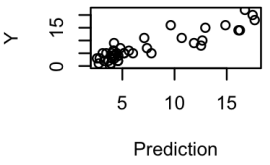
llik= 214.254050616808
SS= 214.209084439873



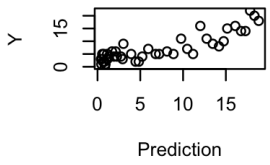
llik= 224.963090676082
SS= 224.721172854664



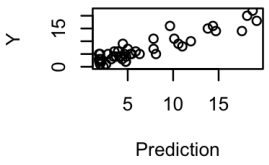
llik= 242.095179764413
SS= 241.992690639926



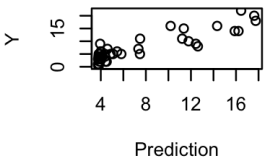
llik= 471.278630833397
SS= 468.726170153911



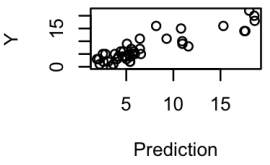
llik= 200.99157942221
SS= 200.948258203206



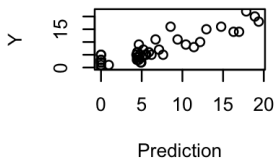
llik= 247.938422406128
SS= 247.889848527629



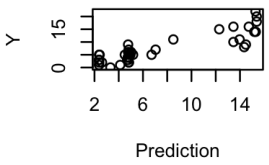
llik= 260.495397441882
SS= 258.950826414081



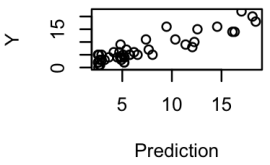
llik= 284.561656823608
SS= 284.557821100945



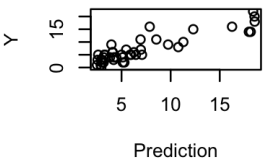
llik= 283.812470234974
SS= 283.507842446065



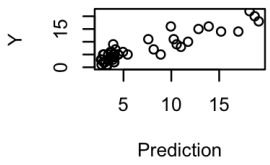
llik= 226.004807950833
SS= 225.993642893583



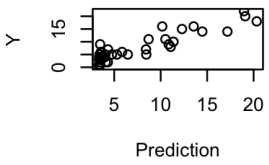
llik= 251.369869740022
SS= 251.285625671203



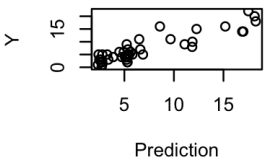
llik= 207.327070875059
SS= 207.280911404732



llik= 206.991045182167
SS= 205.665550883398



llik= 235.441669134977
SS= 235.418751769499



llik= 200.99157942221
SS= 200.948258203206