what you have learned in this class.

Dealing with ambiguity
Random variable
histograms
Comparative boxplots
quantiles
distributions
probability (e.g. from Poisson)
sample mean and variance
distr mean and variance
qqplots

scatterplots
correlation
regression (multiple, polynomial, ...)
ANOVA (R^2, $s_e \sim$ RMSE)
overfitting, collinearity, interaction

sampling distribution
1-sample Confidence Interval for ...
2-sample CI for ...
t-distribution

Hypothesis testing with p-values
1-sample, 2-sample, paired, ... tests
tests for means and proportions
chi-squared test of multiple proportions in 1 pop
chi-squared test of indepedence of two categorical variables

1-way ANOVA F-test for the equality of multiple pop means.

t-test of regression coefficients
Confidence and Prediction Intervals
F-test of model utility

Model selection via bootstrapping (and cross-validation)
Neural networks (as a regression model).

Lecture 27 (Regression, Neural Networks, model selection)

I must deliver on one more promise by answering the question of how many predictors should we include in a regression model, and with what powers?

This problem is referred to as **Model Selection**.

You already have some tools at your disposal
E.g. Regression F-test (followed by a bunch of t-tests).

↳ H_0: All β's are zero $H_0: \beta_i = 0$, one i at a time.

$df = n-(k+1)$

Model selection arises not just in regression, but in every branch of science. Machine learning is one place where it shows-up. And given the recent popularity of machine learning, I will take one example (Neural Networks) for demonstration.

Recall that the real problem in model building is overfitting, ie. the situation where the model performs really good on the data that was used to estimate the model params (e.g. $\hat{\alpha}, \hat{\beta}, \ldots$), but it performs poorly on new/independent data.

Generally, overfitting occurs when the model has too many params.

In some problems, with some assumptions, you can come-up with measures that penalize for number of parameters.

- $R^2_{adj} = 1 - \dfrac{SSE/[n-(k+1)]}{SST/[n-1]}$ high = good.
- Akaiki's Information Criterion (AIC)
- Bayes " " (BIC)

} low = good.

But, none of these truly fix the problem in general, in the sense that a truly bad model can still give good values for these measures.

Part of the problem is philosophical in that there is no such thing as "the best" model. It depends on

- the purpose of the model
- measure of performance (e.g. mean vs variance? or MSE vs. MAE, or ...)

Because of all this, "proving" that model A is "better" than model B is extremely difficult.

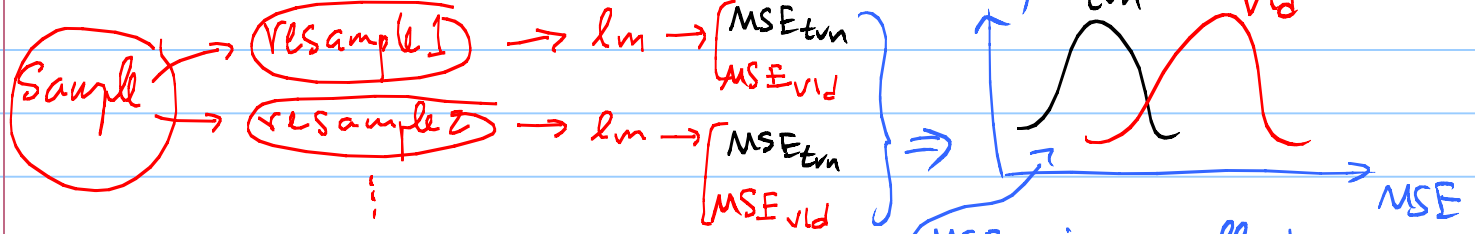
One save thing is that "best" refers to predictions on new, independent / future / ... data, not used in model development. But it turns out that if you pick a model that does best on a fixed independent data, then you will be overfitting that data!

Here are two common Model selection methods:

1) Bootstrap:

training Set

1) Take a random (re-) sample from your sample
2) Develop a model on that resample., ie. Train.
3) Test the model on the unused portion of sample Validation Set
4) Repeat 1-3 many times. Say 100.



MSE_{trn} is generally lower than MSE_{vld}, because MSE_{trn} is minimized in the process of developing the model.

5) Repeat 1-4 for different models
6) Compare MSE_{vld} across models:

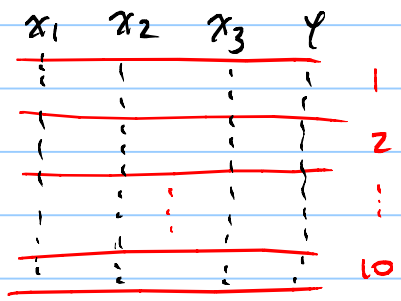
$\Rightarrow$ model A is better in terms of mean of MSE.
But maybe you care more about the width of the MSE.

E.g. 10-fold

2) Cross-Validation

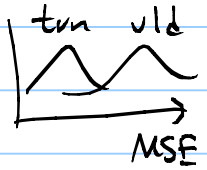
FYI

data: x_1 x_2 x_3 y

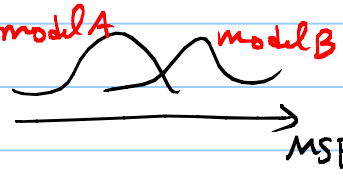


1) use sets 1...9 for trn
2) use sets for validation
3) Repeat 1,2 for a different 9/10 and 1/10 for training and validating.

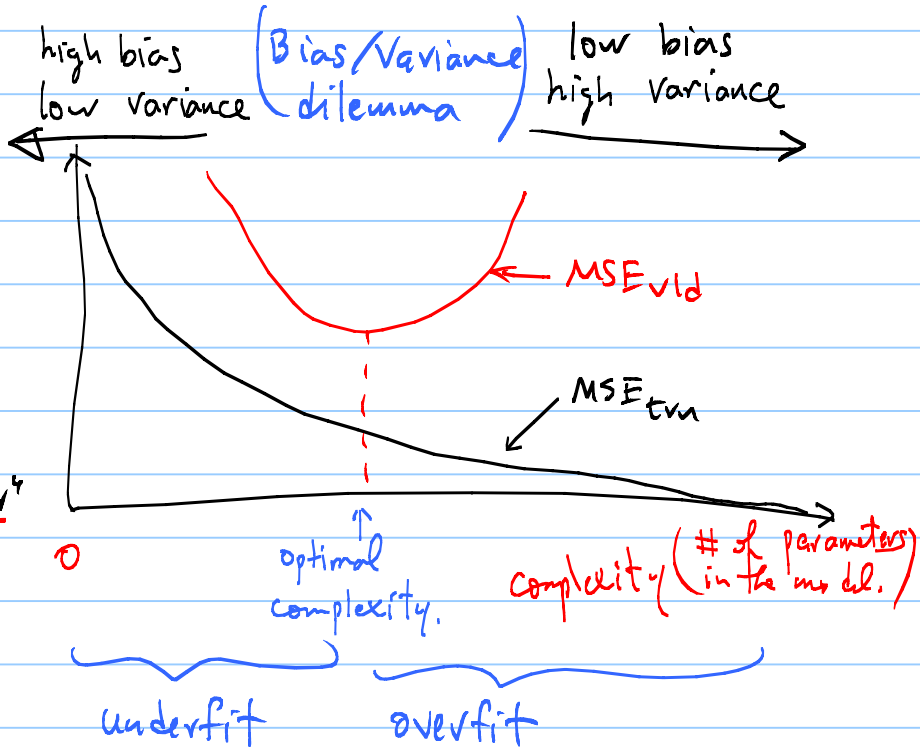
$\Rightarrow$ 10 values of MSE_{trn} and 10 values of MSE_{vld} $\Rightarrow$



Last step (same as bootstrap):



Either way, the general pattern you will see is like this:



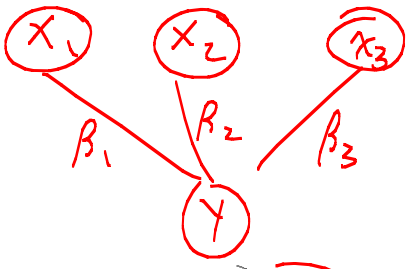
All models have some "parameter(s)" that control its complexity. And the question is what's the optimal value of that parameter. Here, "parameter" is not $\alpha, \beta, \ldots$, but rather, for example, the order of the polynomial in regression. In other models (e.g. NN, below), it'll be something else.

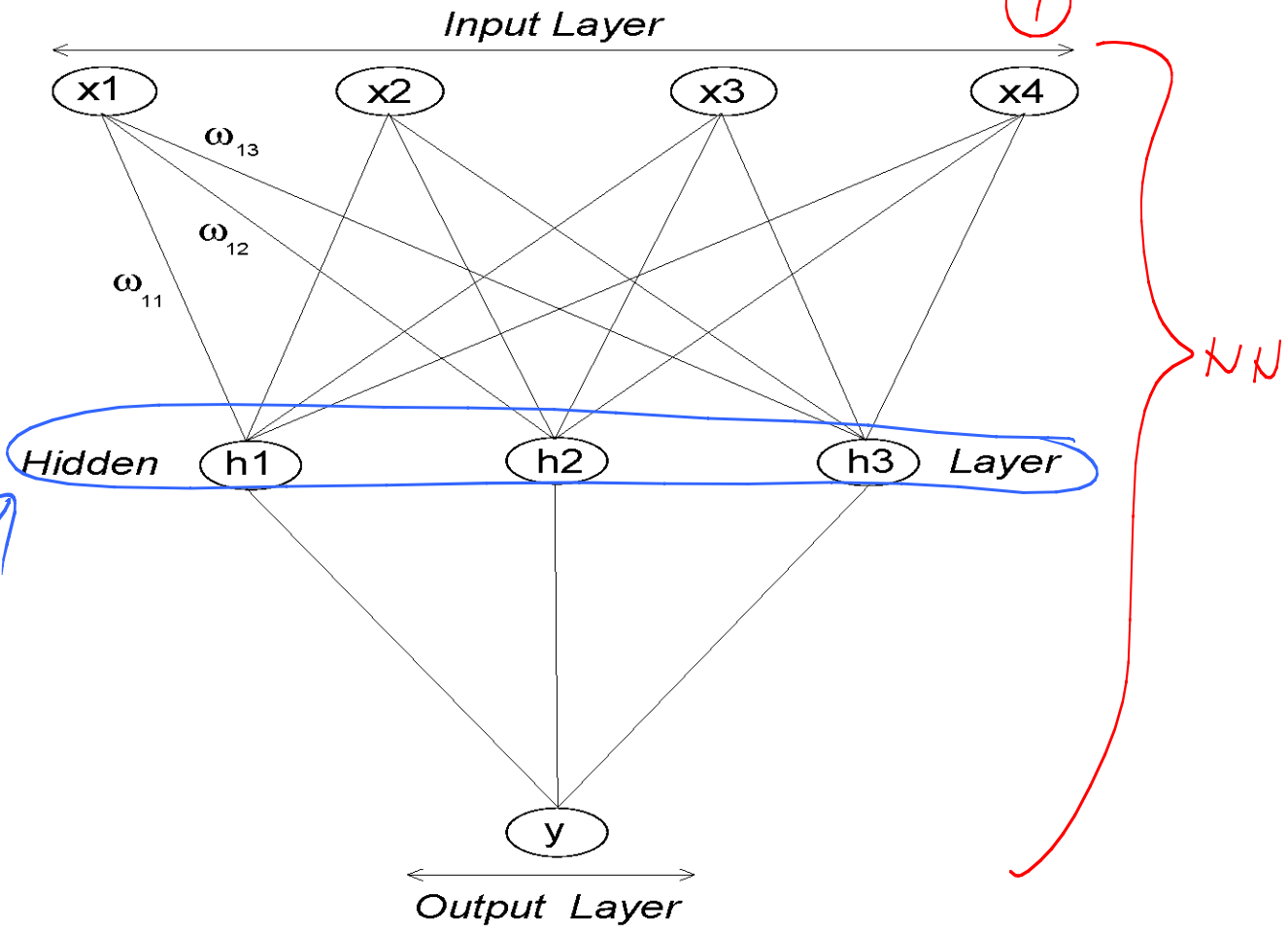
And don't forget that there is always the possibility that there exists a range of optimum values.

Neural Nets (NN):

A neural net is simply a kind of regression!
Look at the similarity:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \iff$$

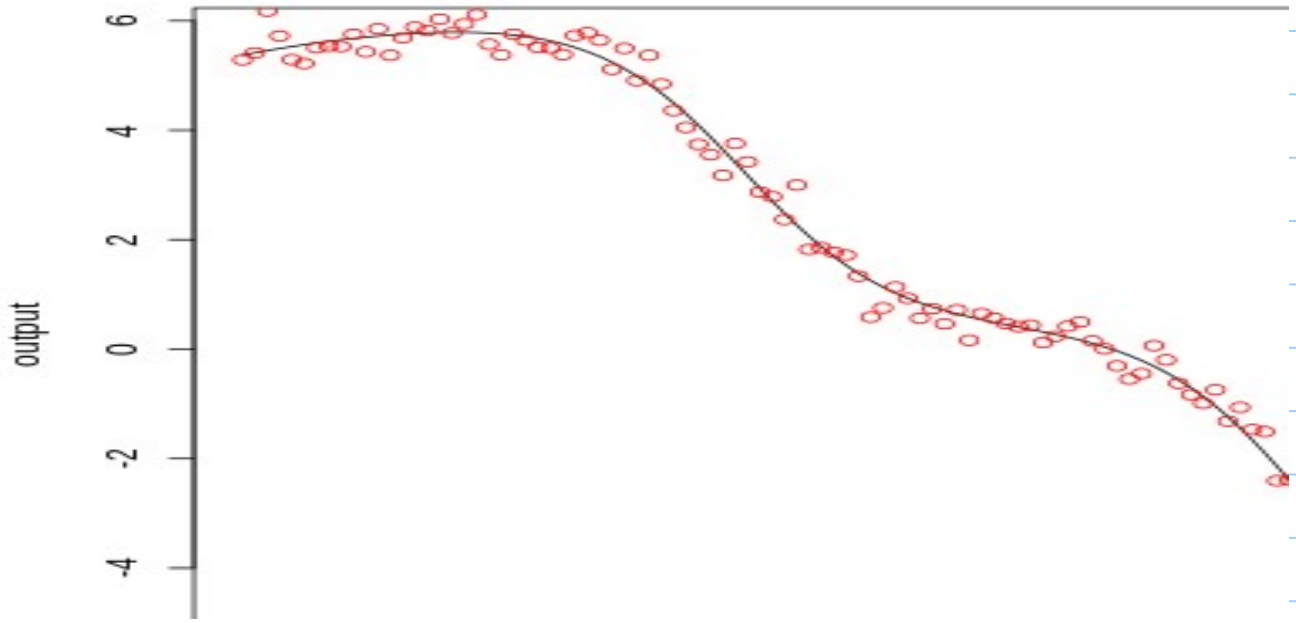




The so-called "hidden nodes" simply allow for more complex functions (non-linear terms, interaction terms, ...)
In fact, the number of such nodes, H, is a measure of the complexity of an NN. So, to avoid overfitting, we need to find the optimal H (or the range of optimal H values).

Most recently, it has become clear that some problems require many hidden layers, each with many hidden nodes. These NN's are called Deep Neural Networks. But the method described here can be used to find the "optimal" number of hidden layers, too.

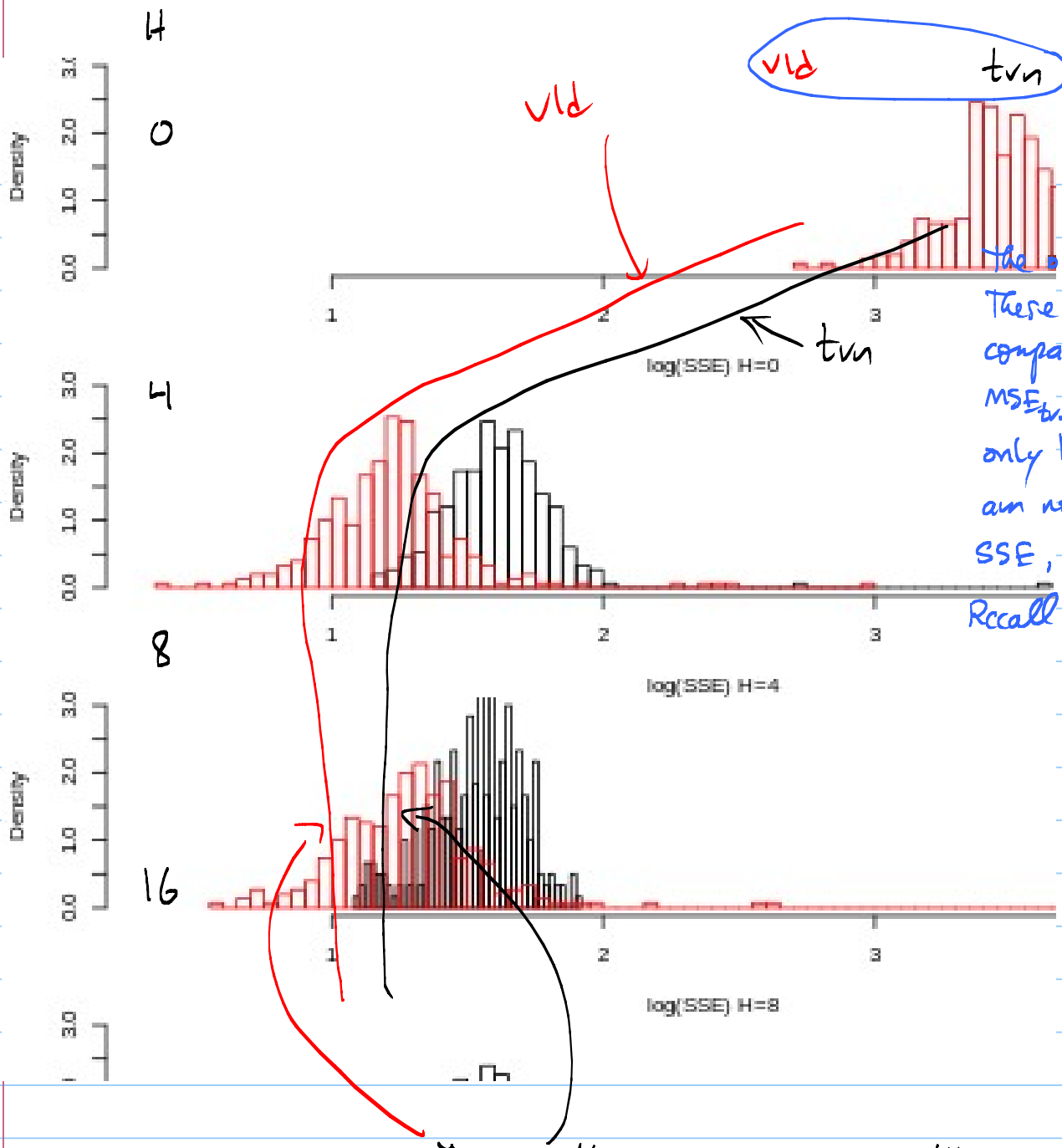
Let's do an example of how to train a NN.
Here is some (fake) data on 1 x and 1 y. The black curve is the true fit (look at the hw, below), and the red dots are our data.



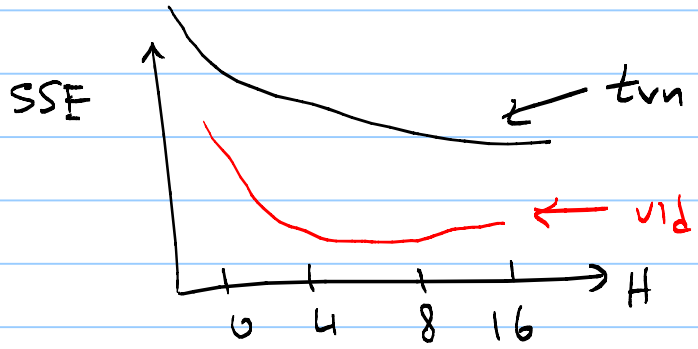
The question is what's the "optimal" value (or range) of H?

Here, let's use bootstrap

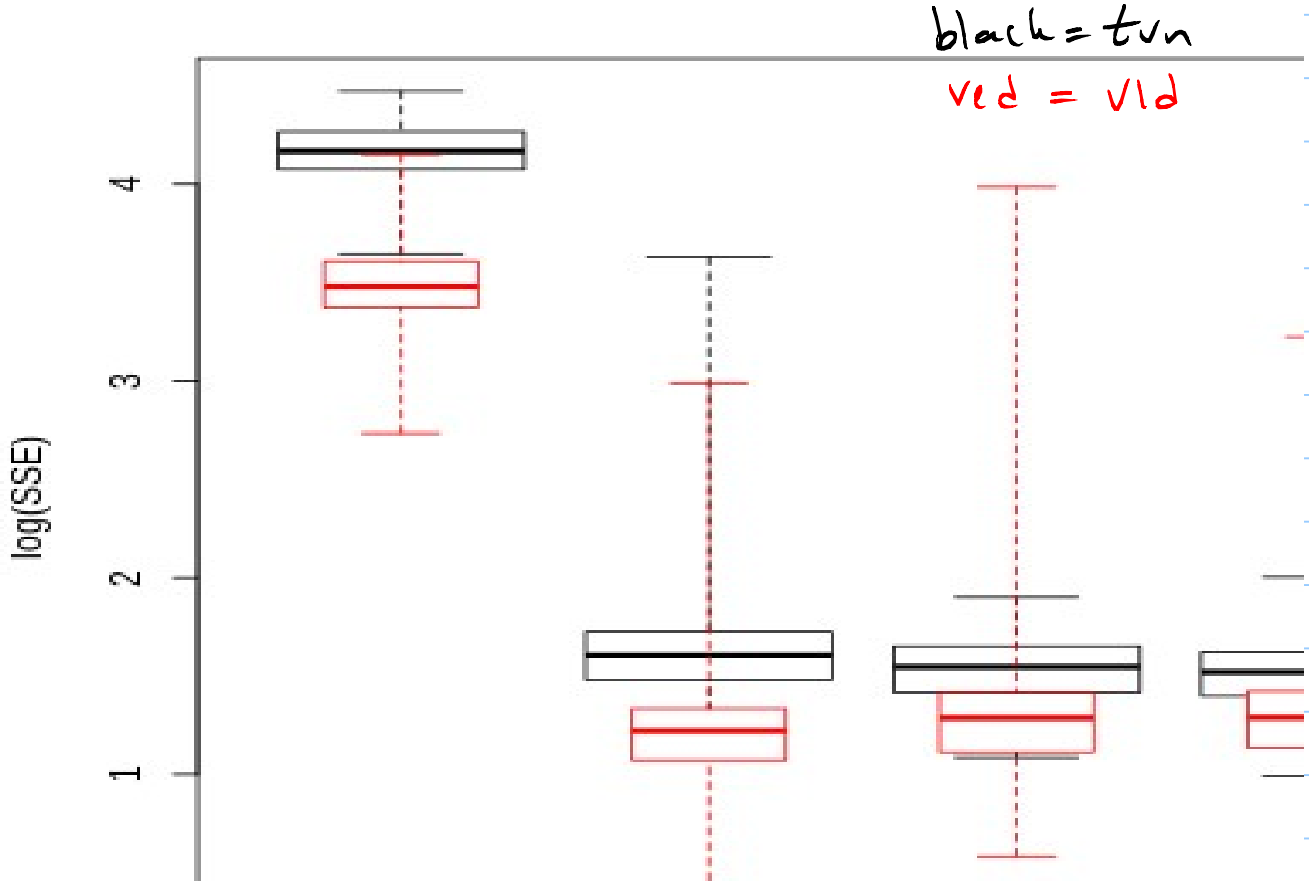
So, expect a hist of a 100 MSE_{trn} and MSE_{vld} values for several values of H (here 0, 4, 8, 16)



If you look at these "curves" sideways, you'll see the generic pattern I mentioned above:



Except, in reality, each "curve" is really a box plot:



The lowest vld errors seem to come from the H=4 model. So, one could conclude that the optimal H is 4.

Truth be told: I made the data with a NN with H=4!

So, this bootstrap model selection method got it right.

But, H=8, and even H=16 are also possible choices, because their is a great deal of overlap in their boxplots. For simplicity, we would still go for H=4.

The hw, below will re-generate all these figs., and it will even ask you to apply some of the tests we have studied to make a more objective comparison of the models. You will also see that the belief that neural nets are "better" than regression is wrong.

copy/paste from here into R.
If you cannot, let me know.

```
# hw_optional (By R).
# Install the R package called nnet. Then,

   library(nnet)

# Simply run the following block, which makes the object "data." The 1st (2nd) column is x (y).
You don't need to understand what this block does.

   n = 100
   n.hd = 4                        # This assures that the true H is 4.
   n.in = n.out =1
   input = seq(-1,1,length=n)
   target = rep(1,n)               # Not used, but necessary for nnet()
   n.wts = (n.in+1)*n.hd + (n.hd+1)*n.out
   set.seed(1)
   nn.1 = nnet(input, target, size = n.hd, rang=10, maxit = 0, linout=T)
   output = predict(nn.1)
   y = output + rnorm(n, 0, 0.3)
   data = data.frame(input,y)
     plot(input, output, type="l")
     lines(input, y , type="p",col=2)

# Now, do go through the rest of this code and understand what it does. The idea behind it is
explained in the "bootstrap" part of the lecture.

   set.seed(1)
   ntrial = 300                      # number of (re) samples that will be taken.
   SSE_vld0 = SSE_vld4 = SSE_vld8 = SSE_vld16 = numeric(ntrial)
   SSE_trn0 = SSE_trn4 = SSE_trn8 = SSE_trn16 = numeric(ntrial)

   for( trial in 1:ntrial){

   trn = sample(1:n, 90, rep=T)                    # Take a (re) sample of size 90.
   nn = nnet( data[trn,1], data[trn,2], skip=T, linout=T, size = 0)  # The only argument you need
to know about is "size" which is what I called H in the lecture, i.e., the number of hidden
nodes.
   SSE_trn0[trial] = nn$value                      # This returns/selects SSE.
   pred_nn = predict(nn, newdata=data.frame(data[-trn,1]))  # -trn means everything other than
trn .
   SSE_vld0[trial] = sum(( pred_nn - data[-trn,2] )^2 )

     nn = nnet( data[trn,1], data[trn,2], linout=T, size = 4)      # Repeat for H = 4
     SSE_trn4[trial] = nn$value
     pred_nn = predict(nn, newdata=data.frame(data[-trn,1]))
     SSE_vld4[trial] = sum(( pred_nn - data[-trn,2] )^2 )
       nn = nnet( data[trn,1], data[trn,2], linout=T, size = 8)    # H = 8
       SSE_trn8[trial] = nn$value
       pred_nn = predict(nn, newdata=data.frame(data[-trn,1]))
       SSE_vld8[trial] = sum(( pred_nn - data[-trn,2] )^2 )
         nn = nnet( data[trn,1], data[trn,2], linout=T, size = 16) # H = 16
         SSE_trn16[trial] = nn$value
         pred_nn = predict(nn, newdata=data.frame(data[-trn,1]))
         SSE_vld16[trial] = sum(( pred_nn - data[-trn,2] )^2 )

   } # end of loop over trial.
```

```
# Here is one of the figs shown in the lecture note. It's more convenient to work with the
log of SSE; otherwise, the hists are all highly skewed. You can try without log, and see
for yourself.

    lim = log(range(SSE_trn0, SSE_trn4, SSE_trn8, SSE_trn16, SSE_vld0, SSE_vld4, SSE_vld8,
SSE_vld16))
    par(mfrow=c(4,1), mar=c(4,4,0,0))
    hist(log(SSE_trn0), breaks=40, xlim=lim, ylim=c(0,3), main="", xlab="log(SSE) H=0",
freq=F)
    hist(log(SSE_vld0), breaks=50, add=T, border = 2, freq=F)
    hist(log(SSE_trn4), breaks=50, xlim=lim, ylim=c(0,3), main="", xlab="log(SSE) H=4",
freq=F)
    hist(log(SSE_vld4), breaks=50, add=T, border = 2, freq=F)
    hist(log(SSE_trn8), breaks=30, xlim=lim, ylim=c(0,3), main="", xlab="log(SSE) H=
8",freq=F)
    hist(log(SSE_vld8), breaks=50, add=T, border = 2, freq=F)
    hist(log(SSE_trn16), breaks=30, xlim=lim, ylim=c(0,3), main="", xlab="log(SSE) H=16",
freq=F)
    hist(log(SSE_vld16), breaks=50, add=T, border = 2, freq=F)

# Here is a comparative boxplot, summarizing the above histograms:

   boxplot(log(SSE_trn0), log(SSE_trn4), log(SSE_trn8), log(SSE_trn16), range = 0, axes=F,
ylab="log(SSE)", xlab="H", ylim=lim)
   boxplot(log(SSE_vld0), log(SSE_vld4), log(SSE_vld8), log(SSE_vld16), range = 0, axes=F,
add=T, border=2, boxwex=0.5)
   axis(1,labels=c(0,4,8,16),at=c(1:4),cex.axis=1)
   axis(2, labels=T) ; box()

# a) Based on the boxplots, one cannot tell if SE_vld4, SSE_vld8, and SSE_vld16 have truly
different means. To test that more objectively, perform an appropriate test, report the p-
value, and state your conclusion, at alpha = 0.05.

# b) Perform the appropriate test if we want to see whether SSE_vld4 is less than
SSE_vld8.  Report the p-value, and state your conclusion, at alphs = 0.05 .

# It is commonly believed that neural nets are "better" than regression. Let's prove that
wrong. To that end,

# c) Revise the code above to ALSO compute SSE_trn_lm and SSE_vld_lm, corresponding to a 9
^th order polynomial regression model.  Hint: Here is how to do one instance of
regression:

   x = data[trn,1]
   y = data[trn,2]
   lm.1 = lm( y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^
9))
   summary.aov(lm.1)[[1]][10,2]
   new.data = data.frame(data[-trn,1])
   colnames(new.data) = c("x")
   pred_lm = predict(lm.1, newdata=new.data)
   sum(( pred_lm - data[-trn,2] )^2 )

# d) Finally, make a comparative boxplot, like the one above, that includes SSE_trn_lm and
SSE_vld_lm. Do not worry about the x-axis labels.
```