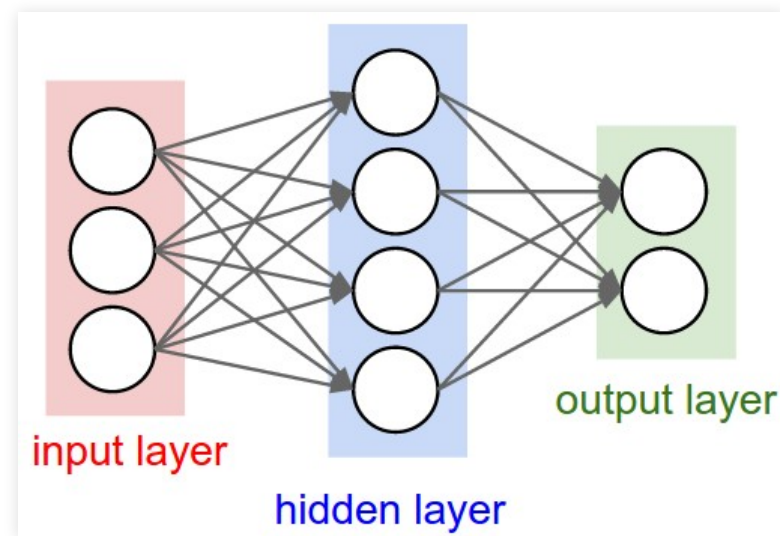


Error: 0.093071 Steps: 28642

Part I: Introduction to Neural Networks

What is a neural network?

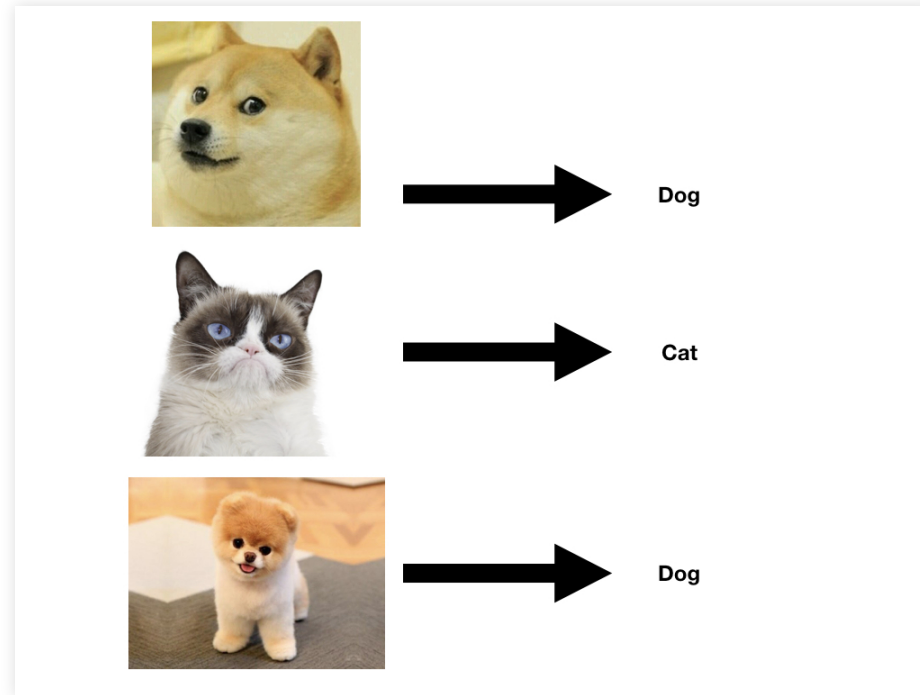
- Predictive machine learning algorithm
- Takes in inputs and outputs a prediction
 - Classification: Predicts a category (color, political party, type of cell)
 - Regression: Predicts a number (temperature, stock price, cell count)



Supervised machine learning

- You 'train' the algorithm with a set of inputs with known outputs, or labels
 - neural networks, linear regression, support vector machines, random forests

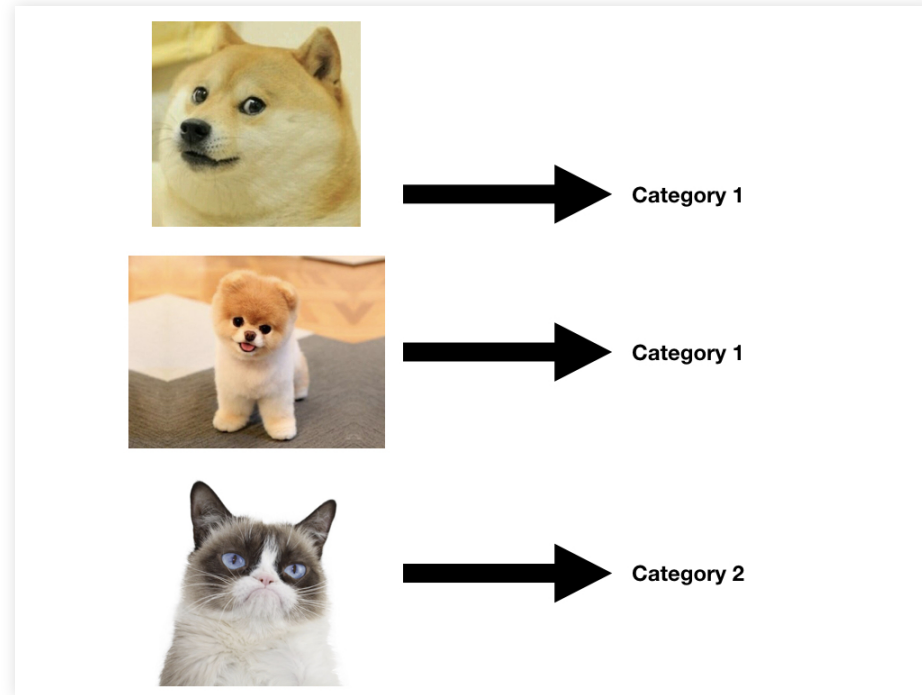
Supervised Machine Learning



Unsupervised machine learning

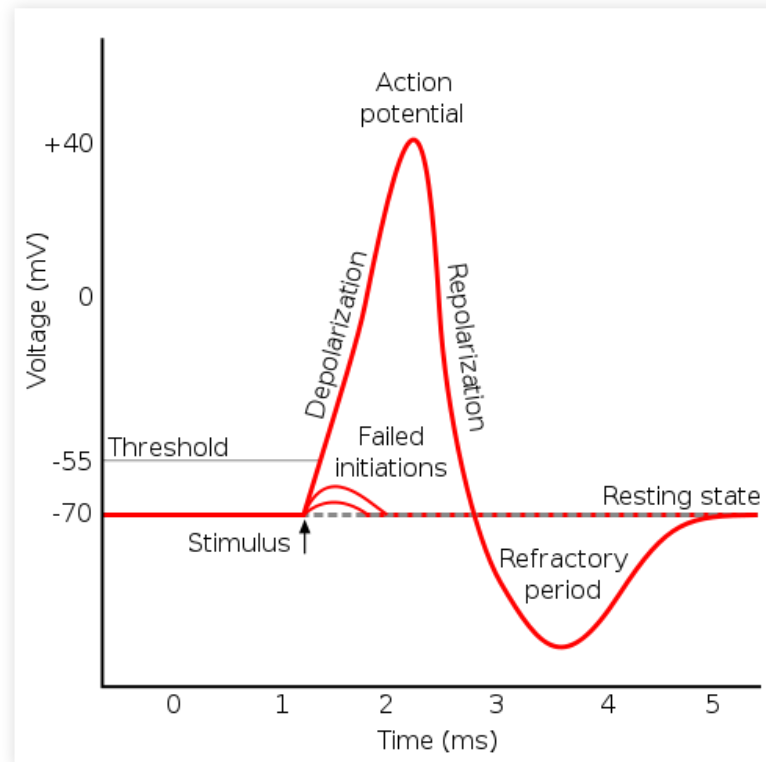
- The algorithm does not get known outputs/labels
 - clustering, principal components analysis

Unsupervised Machine Learning



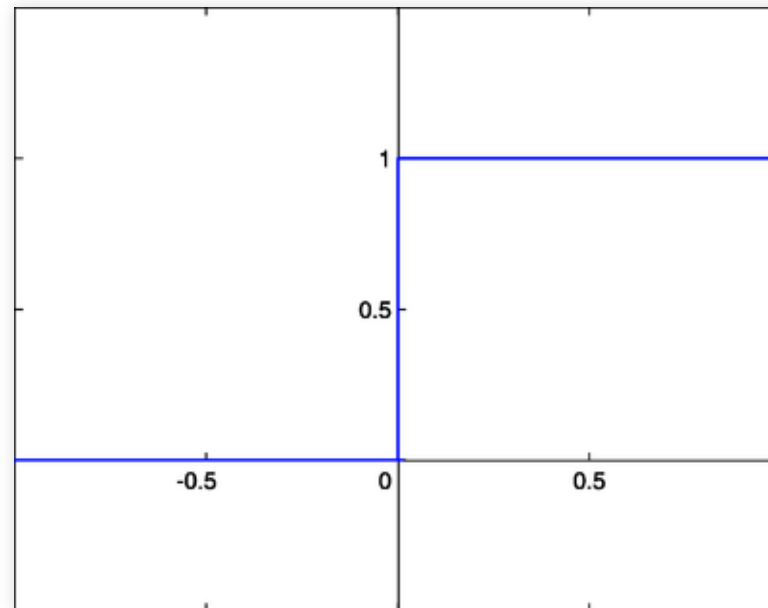
Neurons in the Brain

- Neurons in the brain fire when a stimulus causes their electrical current to exceed a certain threshold

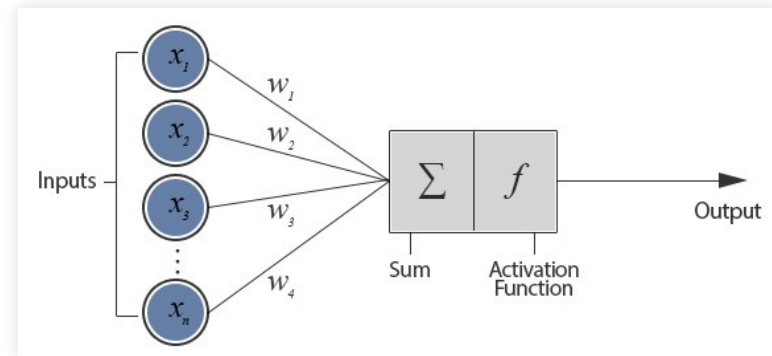


Modeling Neurons

- We can model the firing of a neuron using a step function
- Activation function: is this neuron active, i.e. will it pass along its signal?

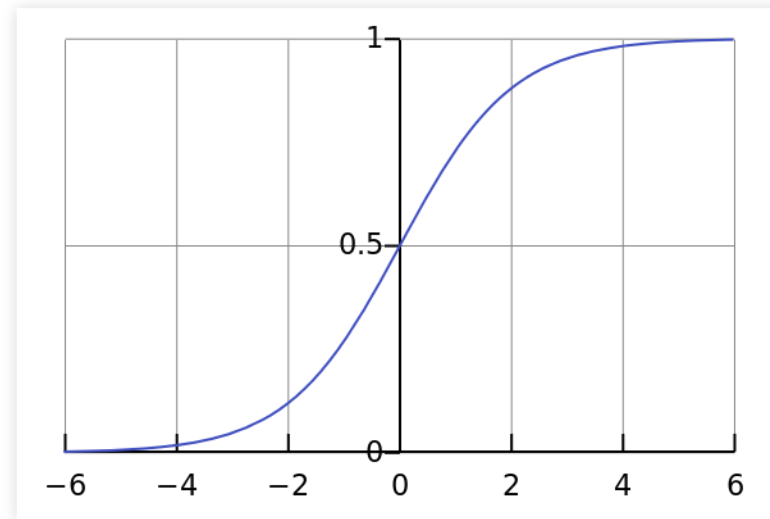


Perceptron



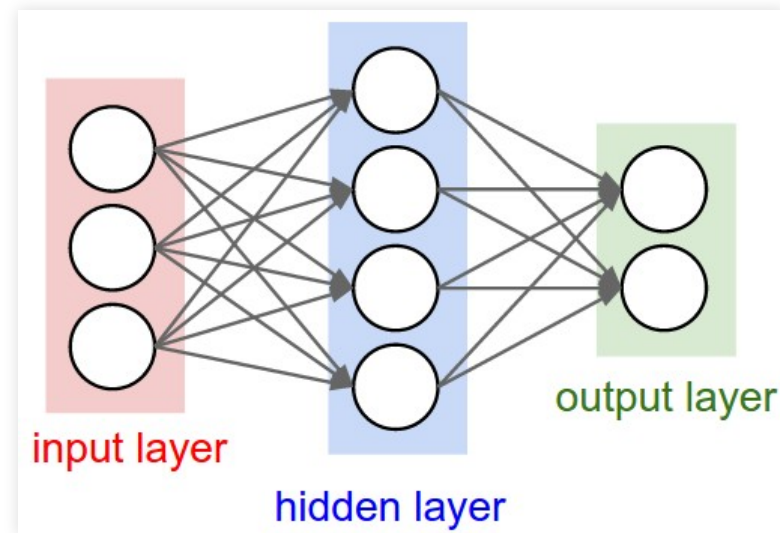
- Node in a neural network (like neuron in the brain)
- Takes in inputs (x_1, x_2) and weights (w_1, w_2) as a weighted sum
- Processes inputs and weights using activation function f
- If activation functions reaches particular threshold, it fires

Modeling Neurons



- Sometimes we prefer a 'fuzzier' signal for our activation function
- Rather than a yes/no for whether a perceptron has fired, we give a value between 0 and 1
- Determines the strength of the signal
- Examples: logistic function, other sigmoid functions

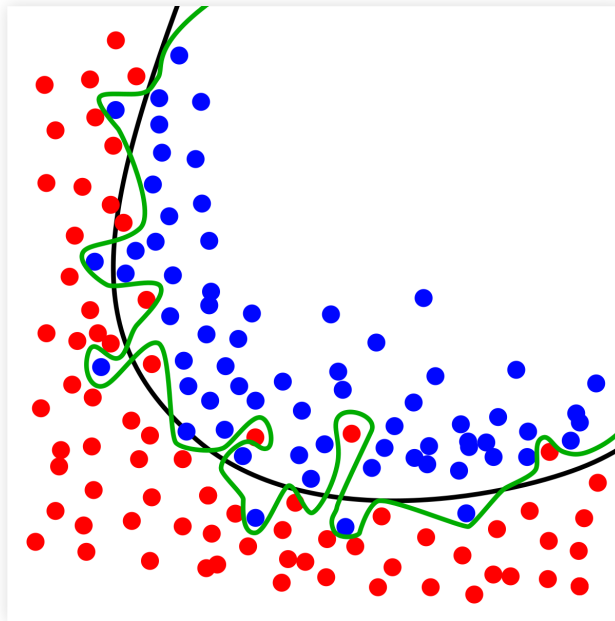
Neural Network Structure



- Input layer: Features (also known as covariates or independent variables)
- Hidden layer: Perceptrons
- Output layer: Prediction

Training a Neural Network

- Data is split into two sets: training set, and test set
- Training stage determines optimal weights for the supplied data
- Testing stage evaluates model using predictive performance
- Splitting data reduces over-fitting



Should you use a neural network?

- Neural networks work for both classification and regression
- Very good predictive performance, even with complex problems like image recognition
- Can approximate almost any function, including non-linear functions

- Prone to over-fitting
- Tuning parameters are required (number of nodes, layers)
- Provides little insight into the data (black box model)
- Can be computationally intensive



History of Neural Networks

- 1950s: Perceptron algorithm invented
- 1960s-1970s: Single perceptrons proven to be not useful for modeling even simple nonlinear functions; research stalls
- 1980s: Multi-layer perceptron networks introduced that solve problems of single perceptrons; neural networks become popular

History of Neural Networks

- 1990s: Neural networks did not scale well, support vector machines replace them in popularity
- 2000s: Increased computational power brings back neural network popularity, deep learning introduced
- 2010s: Improvements in available data, research, and computational power make deep learning very successful and popular
- Deep Learning: A complex neural network with many hidden layers

THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG
PILE OF LINEAR ALGEBRA, THEN COLLECT
THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.



Data Example: Predicting penguin species

```
library(palmerpenguins)
head(penguins)
```

```
# A tibble: 6 × 8
  species island bill_length_mm
bill_depth_mm flipper_length_...
body_mass_g sex
  <dbl>      <fct>      <fct>      <dbl>
<dbl>      <int>      <int> <fct>
1 Adelie   Torge...      39.1
18.7      181      3750 male
2 Adelie   Torge...      39.5
17.4      186      3800 fema...
3 Adelie   Torge...      40.3
18      195      3250 fema...
4 Adelie   Torge...      NA
NA      NA      NA <NA>
5 Adelie   Torge...      36.7
19.3      193      3450 fema...
6 Adelie   Torge...      39.3
```

```
20.6          190          3650 male  
# ... with 1 more variable: year <int>
```

```
set.seed(28072022)
```

Preprocessing

```
library(tidyverse)

penguins2 = penguins %>%
  # create an ID column
  mutate(id =
as.character(1:nrow(penguins)))

# extract just the feature names for
future reference
feature.names = c("bill_length_mm",
"bill_depth_mm",
                  "flipper_length_mm",
"body_mass_g", "sex")
# excluding island because it is too
predictive!

# for future modeling
penguins3 = penguins2 %>%
  # add dummy variable columns in
addition to categorical column
  mutate(name = paste0("species_",
species), value = TRUE) %>%
```



```
    pivot_wider(names_from = name,  
values_from = value, values_fill = FALSE)
```

Normalization

```
penguins4 = penguins3 %>%  
  mutate_if(is.numeric, ~./sum(., na.rm  
= TRUE))
```

```
head(penguins4)
```

```
# A tibble: 6 × 12  
  species island bill_length_mm  
bill_depth_mm flipper_length_...  
body_mass_g sex  
  <dbl>    <dbl>    <dbl>    <dbl> <fct>  
1 Adelie  Torge... 0.00260  
0.00319      0.00263 0.00261 male  
2 Adelie  Torge... 0.00263  
0.00297      0.00271 0.00264  
fema...  
3 Adelie  Torge... 0.00268  
0.00307      0.00284 0.00226  
fema...  
4 Adelie  Torge... NA      NA  
NA      NA      <NA>
```

```
5 Adelie Torge... 0.00244
0.00329 0.00281 0.00240
fema...
6 Adelie Torge... 0.00262
0.00351 0.00277 0.00254 male
# ... with 5 more variables: year <dbl>, id
<chr>, species_Adelie <lgl>,
# species_Gentoo <lgl>,
species_Chinstrap <lgl>
```

Training and Testing

```
# split into training and testing
train.sample      = sample(penguins4$id,
250)
test.sample       = penguins4$id[!
(penguins4$id %in% train.sample)]
penguins4.train   = penguins4[penguins4$id
%in% train.sample,]
penguins4.test    = penguins4[penguins4$id
%in% test.sample,]
```

Fitting a Neural Network Using nnet

```
library(nnet)

nnet.formula = as.formula(
  paste('species ~', paste(feature.names,
    collapse = ' + '))
)
print(nnet.formula)
```

```
species ~ bill_length_mm + bill_depth_mm
+ flipper_length_mm +
  body_mass_g + sex
```

```
nnet.model = nnet(nnet.formula, data =
  penguins4.train, size = 5)
```

```
# weights:  48
initial  value 270.077760
iter   10 value 255.923144
iter   20 value 255.801707
iter   30 value 254.838718
iter   40 value 248.835836
```

```
iter 50 value 188.304991
iter 60 value 162.687206
iter 70 value 132.532282
iter 80 value 81.549817
iter 90 value 52.427195
iter 100 value 28.677359
final value 28.677359
stopped after 100 iterations
```

Fitting a Neural Network Using nnet

```
head(round(predict(nnet.model), 3))
```

	Adelie	Chinstrap	Gentoo
1	0.572	0.428	0
2	0.983	0.017	0
3	1.000	0.000	0
4	1.000	0.000	0
5	0.983	0.017	0
6	1.000	0.000	0



Fitting a Neural Network Using neuralnet

```
library(neuralnet)

# this package requires all numeric
variables and no missing values
penguins5 = penguins4 %>%
  # make sex into a dummy variable
  mutate_at("sex", function(x){ ifelse(x
== "female", 1, 0)}) )

# remove missing values
penguins5 =
penguins5[complete.cases(penguins5),]

# names of output dummy cols
dummy.cols = c("species_Adelie",
"species_Gentoo", "species_Chinstrap")

# split into train/test
train.sample      = sample(penguins5$id,
200)
test.sample       = penguins5$id[!]
```

```
(penguins5$id %in% train.sample)]  
penguins5.train = penguins5[penguins5$id  
%in% train.sample,]  
penguins5.test  = penguins5[penguins5$id  
%in% test.sample,]
```

Fitting a Neural Network Using neuralnet

```
# fit neural network
neuralnet.formula = paste("species_Adelie
+ species_Gentoo + species_Chinstrap ~",
                          paste(feature.names,
                                collapse = " + "))
print(neuralnet.formula)
```

```
[1] "species_Adelie + species_Gentoo +
species_Chinstrap ~ bill_length_mm +
bill_depth_mm + flipper_length_mm +
body_mass_g + sex"
```

```
neuralnet.model = neuralnet(
  neuralnet.formula, data =
penguins5.train, hidden = c(5),
linear.output = FALSE
)
print(head(round(neuralnet.model$net.result[[1]],
3)))
```

	[,1]	[,2]	[,3]
[1,]	1.000	0	0.011
[2,]	1.000	0	0.000
[3,]	1.000	0	0.000
[4,]	1.000	0	0.000
[5,]	0.996	0	0.373
[6,]	1.000	0	0.000

Network Visualization

Fitting a Neural Network using tidymodels and parsnip

```
library(tidymodels)
library(parsnip)

model.spec =
  parsnip::mlp() %>%
  parsnip::set_engine("nnet") %>%
  parsnip::set_mode("classification")

parsnip.model =
  model.spec %>%
  parsnip::fit_xy(
    x = penguins4.train[,feature.names],
    y = penguins4.train$species %>%
    factor()
  )
```

Model Evaluation

```
# calculate accuracy on training set
train.probabilities = predict(
  parsnip.model, new_data =
penguins4.train,
  type = "prob"
)

# ensure columns are in the right order!
train.predictions = max.col(
  train.probabilities[,
    sort(colnames(train.probabilities))]
)
train.true.values = max.col(
  penguins4.train[,dummy.cols]
  [,sort(dummy.cols)]
)
print(paste(
  "Train Accuracy:",
  round(100*mean(train.predictions ==
train.true.values,
    na.rm = TRUE)), "%"))
```

```
[1] "Train Accuracy: 96 %"
```

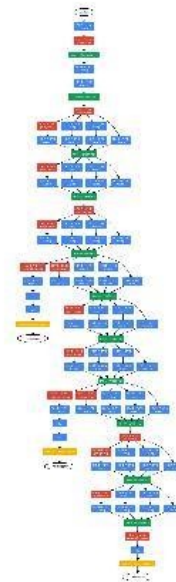
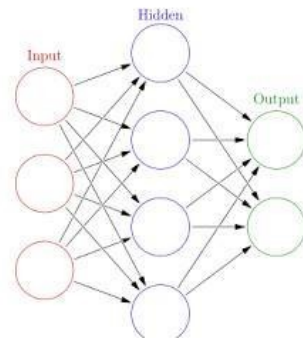
```
[1] "Test Accuracy: 98 %"
```




Cross Validation

- Cross validation can be used to pick the optimal number of hidden nodes
- Possible rule of thumb: between 1 and the number of input variables (other rules of thumb also exist)
- k-fold cross validation: split data into k folds. For each fold, train on k-1 folds, test on the kth fold
- Choose hidden layer with best average predictive performance across folds (such as Mean Squared Error)
- Other types of cross validation: leave-one-out, Monte Carlo cross validation
- Many packages automate this process!

you vs. the guy she tell you not to worry
about



Cross Validation

```
cv = function(data, hidden, n.folds = 2,  
formula, feature.names, output.names)  
{  
  # split data into 'folds' for cross  
validation  
  fold.size = floor(nrow(data)/n.folds)  
  category = NULL  
  for(k in 1:n.folds){  
    category = c(category, rep(k,  
fold.size))  
  }  
}
```

Cross Validation

```
# we often have rounding errors (i.e.
if we have 10 folds and
# the number of rows is not divisible
by 10)
# so randomly assign the leftover rows
to different folds
if(length(category) != nrow(data))
{
  extra.length = nrow(data) -
length(category)
  extra = sample(seq(1, n.folds),
extra.length)
  category = c(category, extra)
}
data$cv.fold = sample(category)
```

Cross Validation

```
mse = rep(NA, n.folds)
for(k in 1:n.folds)
{
  # extra training/test data for this
fold
  train.cv = data[data$cv.fold != k,]
  test.cv = data[data$cv.fold == k,]

  # the algorithm starts out with a
random set of weights
  # occasionally it starts with
suboptimal weights and does not converge
  # within the allowed time (maximum
steps). Rather than increasing
  # the max number of steps, which
would slow down the script,
  # I retry the model until it
converges
  attempts = 0
  model.cv = NULL
  while( is.null(model.cv) & attempts
<= 10)
```

```
    {  
        model.cv = try(neuralnet(formula,  
train.cv, hidden = hidden, linear.output  
= FALSE))  
        attempts = attempts + 1  
    }
```

Cross Validation

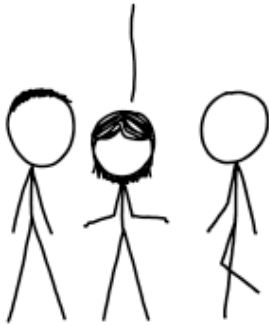
```
    # calculate Mean Squared Error
    test.predictions =
max.col(predict(model.cv,
test.cv[,feature.names]))
    test.true.values =
max.col(test.cv[,output.names])
    mse[k] = sum((test.predictions -
test.true.values)^2)/nrow(test.cv)
  }
  return(mean(mse))
}
```


Cross Validation

```
test.size = function(data, n.folds,
                      formula, feature.names, output.names)
{
  hidden.values = seq(1,
length(feature.names))
  n.values = length(hidden.values)
  mse.all = rep(NA, n.values)
  for(i in 1:n.values)
  {
    mse.all[i] = cv(
      data, hidden.values[i], n.folds,
      formula, feature.names,
output.names)
  }
  best = which.min(mse.all)
  return(hidden.values[best])
}
```

Cross Validation

OUR FIELD HAS BEEN
STRUGGLING WITH THIS
PROBLEM FOR YEARS.



STRUGGLE NO MORE!
I'M HERE TO SOLVE
IT WITH *ALGORITHMS!*



SIX MONTHS LATER:

WOW, THIS PROBLEM
IS REALLY HARD.

YOU DON'T SAY.



Part II: Using Debugging and Profiling Tools

BRACE YOURSELVES

**A LIVE DEMO IS
COMING**

memegenerator.net

Debug on Error

- Tools > Global Options > Advanced > uncheck “Use debug error handler only when my code contains errors”
- Debug > On Error > Break in Code
- Optional: turn warnings into errors

```
options(warn = 2)
```






Debug Example

```
foo = function(x){  
  if(x > 0)  
  {  
    salutation = "Hello"  
  } else  
  {  
    salutation = "Goodbye"  
  }  
  print(salutation)  
}  
foo(x = log(-1))
```

Debug Tool

```
> source('~/Desktop/R Ladies/error.R')
Error in if (x > 0) { : missing value where TRUE/FALSE needed
In addition: Warning message:
In log(-1) : NaNs produced
Called from: foo(x = log(-1))
Browse[1]> print(x)
[1] NaN
Browse[1]> |
```

Console ~/Desktop/R Ladies/ ↗

 Next |  |  |  Continue |  Stop

Browse[1]> n
> foo = function(x){
+ if(x > 0)

Traceback

The image shows a screenshot of an R IDE. At the top, a 'Traceback' window is open, displaying the following error sequence:

```
→ eval(expr, envir, enclos)
  foo(x = log(-1)) at error.R:2
  [Debug source] at error.R:11
```

Below the traceback, the source editor shows the code for 'error.R'. The code defines a function 'foo' and calls it with a negative value, which causes the error.

```
1 foo = function(x){
2   if(x > 0)
3   {
4     salutation = "Hello"
5   } else
6   {
7     salutation = "Goodbye"
8   }
9   print(salutation)
10 }
→ 11 foo(x = log(-1))
```

The source editor window has tabs for 'nn.Rpres *', 'error.R *', and 'adherenc'. The toolbar includes navigation arrows, a search icon, and a 'Source on Save' checkbox.



99 little bugs in the code.
99 little bugs in the code.
Take one down, patch it around.

127 little bugs in the code...

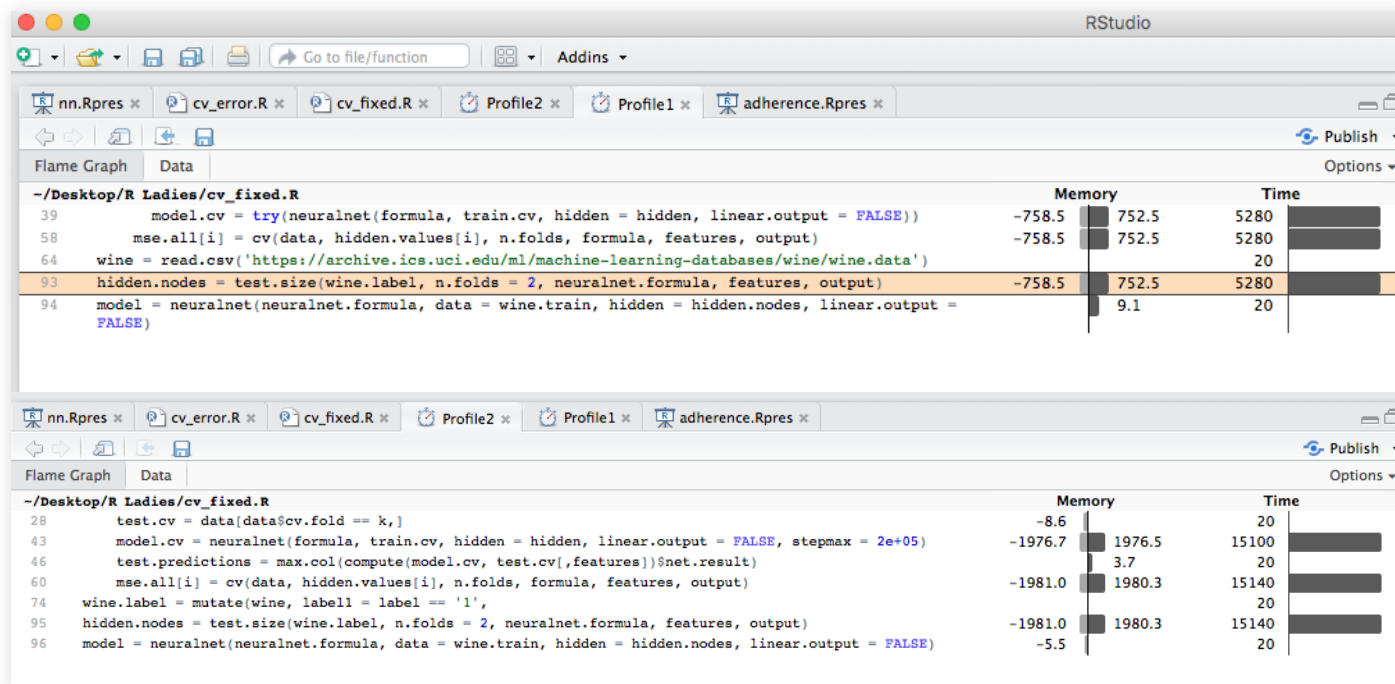
R Debug

- Built-in tools in Rstudio to help you debug your code
- Enter debug mode at a certain line, on error, or when a particular function executes
- s: step into function, n: execute next statement
- f: finish function/loop, c: continue running
- Q: quit
- *Most important function:* `recover()`: steps out to allow you to enter a different function/level
- Note: in base R, same functionality by using functions like `browser()`, `trace()`, `recover()`

R Profiler

- Built-in tools in Rstudio to help you identify slow or memory intensive parts of your code
- Allows you to easily compare speed when you modify code
- Note: in base R, same functionality by using functions like `Rprof()`, `profr` library

Comparing Scripts



References

- **Classification with Neural Networks**
- **Customizing R Presentations**
- **Dummies' Introduction to Neural Networks**
- **Cross Validation for selecting hidden node number**
- **Deep Learning 101**
- **Debugging with Rstudio**

About me

- B.A. in Statistics from Harvard
- 3 years in industry (data scientist and software engineer at an education analytics company)
- Ph.D. in Statistics from Harvard
- Starting soon as faculty at University of New South Wales
- CRAN package for calculating power for experiments: <https://github.com/MDRCNY/PUMP>
- Twitter: https://twitter.com/stats_hunter
- LinkedIn: <https://www.linkedin.com/in/kristenbhunter/>

Questions?

