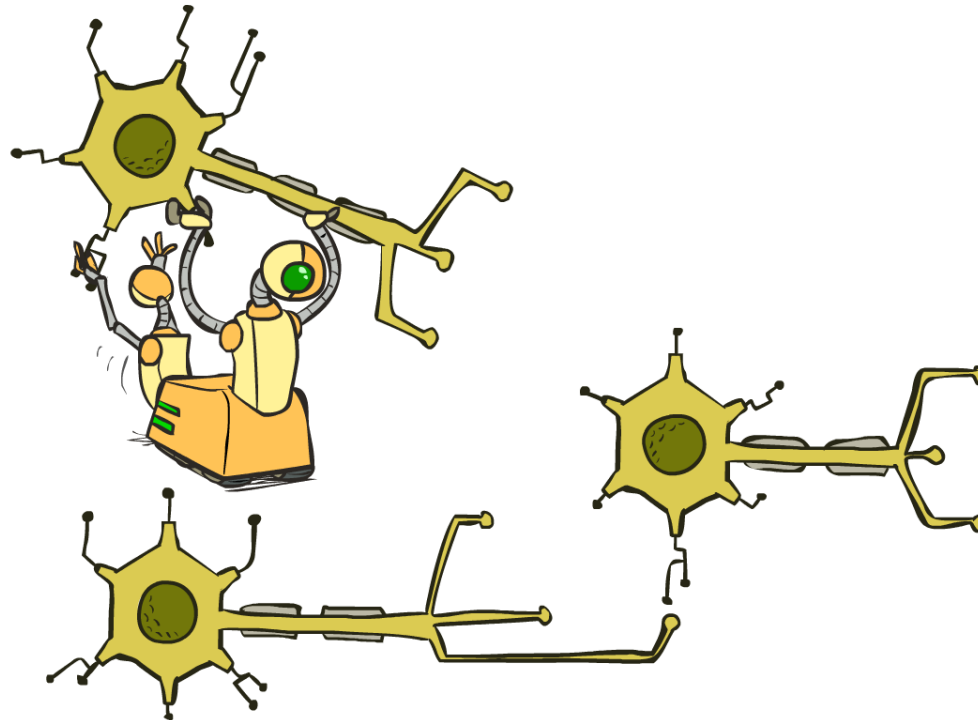


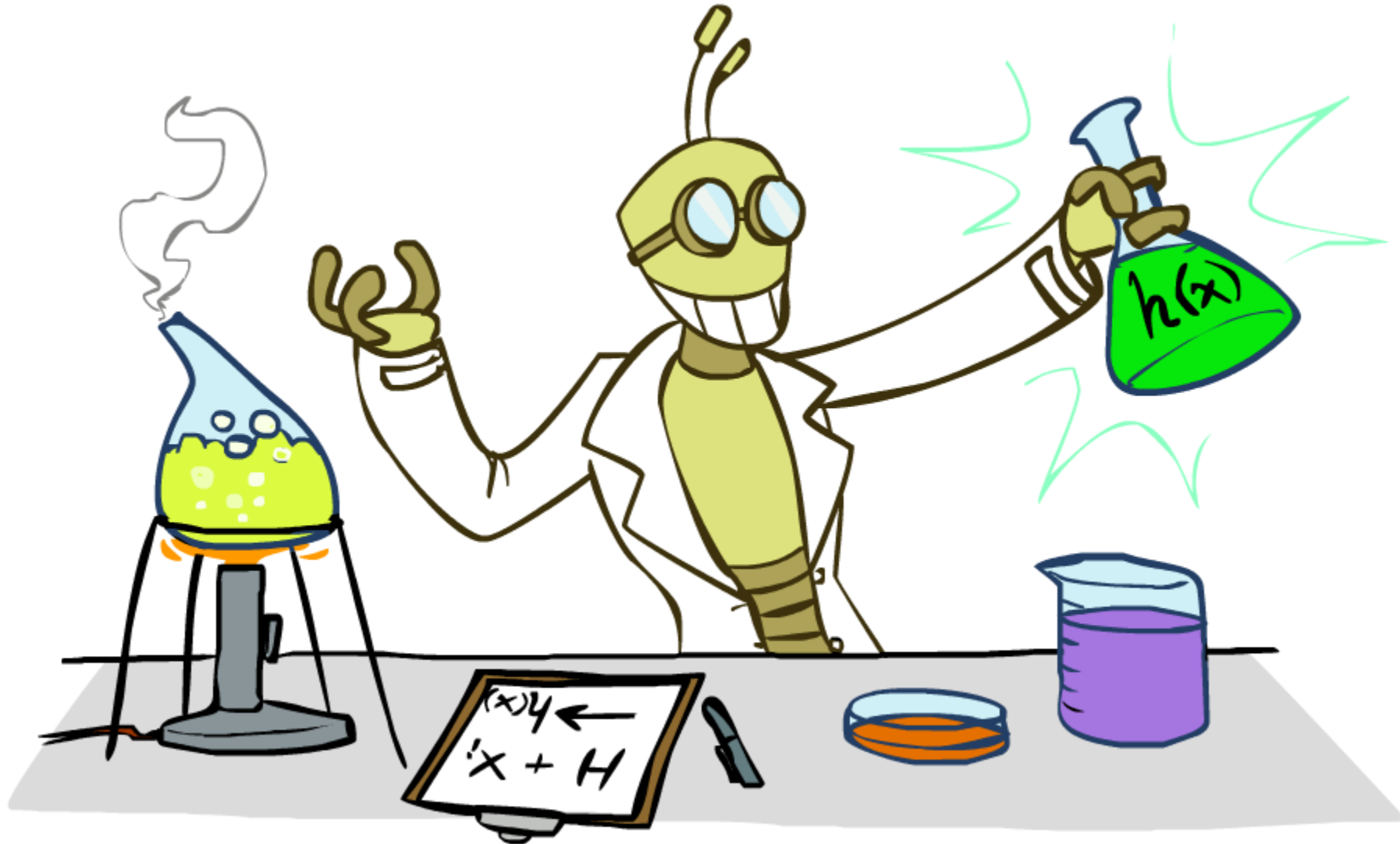
# CS 188: Artificial Intelligence

## Optimization and Neural Nets



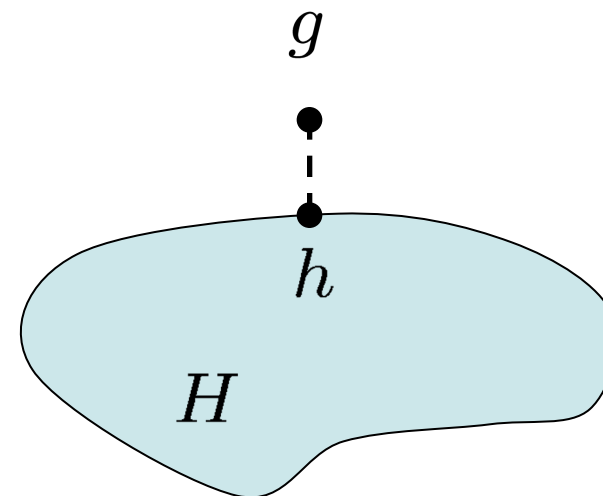
Instructor: Nicholas Tomlin

# Inductive Learning



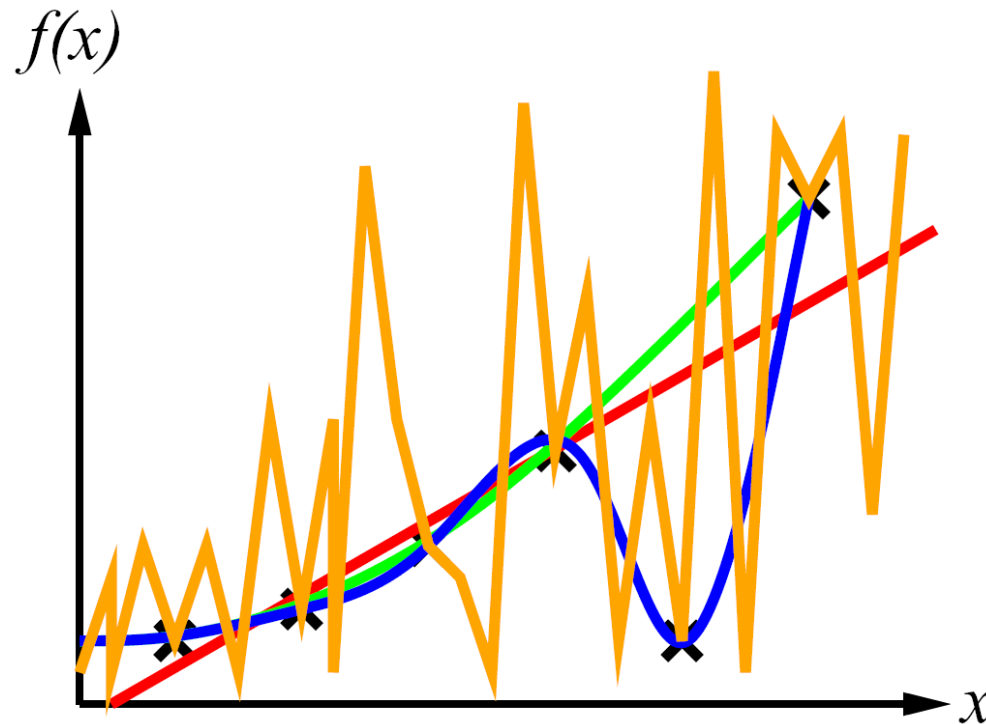
# Inductive Learning (Science)

- Simplest form: learn a function from examples
  - A target function:  $g$
  - Examples: input-output pairs  $(x, g(x))$
  - E.g.  $x$  is an email and  $g(x)$  is spam / ham
  - E.g.  $x$  is a house and  $g(x)$  is its selling price
- Problem:
  - Given a hypothesis space  $H$
  - Given a training set of examples  $x_i$
  - Find a hypothesis  $h(x)$  such that  $h \sim g$
- Includes:
  - Classification (outputs = class labels)
  - Regression (outputs = real numbers)
- How do perceptron and naïve Bayes fit in? ( $H, h, g$ , etc.)



# Inductive Learning

- Curve fitting (regression, function approximation):



- Consistency vs. simplicity
- Ockham's razor

# Consistency vs. Simplicity

---

- Fundamental tradeoff: bias vs. variance
- Usually algorithms prefer consistency by default (why?)
- Several ways to operationalize “simplicity”
  - Reduce the **hypothesis space**
    - Assume more: e.g. independence assumptions, as in naïve Bayes
    - Have fewer, better features / attributes: feature selection
    - Other structural limitations (decision lists vs trees)
  - **Regularization**
    - Smoothing: cautious use of small counts
    - Many other generalization parameters (pruning cutoffs today)
    - Hypothesis space stays big, but harder to get to the outskirts

# Decision Trees



# Reminder: Features

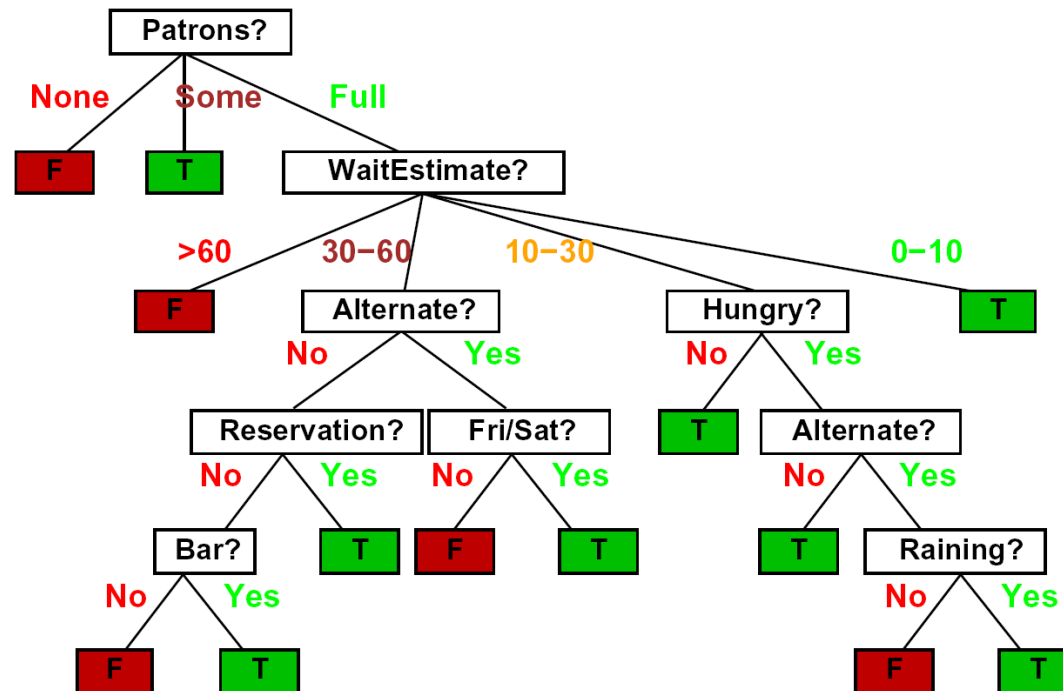
- Features, aka attributes
  - Sometimes: TYPE=French
  - Sometimes:  $f_{\text{TYPE=French}}(x) = 1$

Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
$X_1$	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>0-10</i>	<i>T</i>
$X_2$	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30-60</i>	<i>F</i>
$X_3$	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>Some</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>T</i>
$X_4$	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>10-30</i>	<i>T</i>
$X_5$	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>&gt;60</i>	<i>F</i>
$X_6$	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Italian</i>	<i>0-10</i>	<i>T</i>
$X_7$	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>F</i>
$X_8$	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Thai</i>	<i>0-10</i>	<i>T</i>
$X_9$	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>&gt;60</i>	<i>F</i>
$X_{10}$	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>Italian</i>	<i>10-30</i>	<i>F</i>
$X_{11}$	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>0-10</i>	<i>F</i>
$X_{12}$	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>30-60</i>	<i>T</i>

# Decision Trees

- Compact representation of a function:
  - Truth table
  - Conditional probability table
  - Regression values

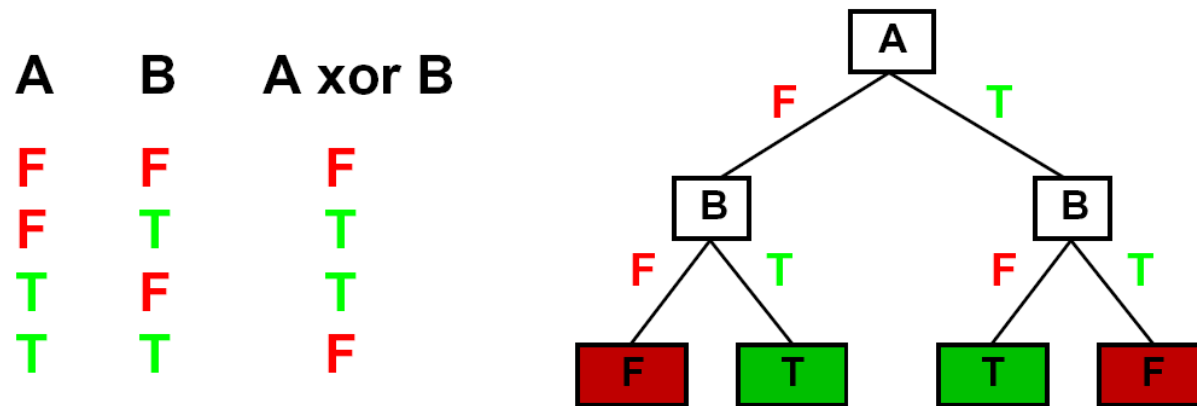
- True function
  - Realizable: in  $H$





# Expressiveness of DTs

- Can express any function of the features



$$P(C|A, B)$$

- However, we hope for compact trees

# Comparison: Perceptrons

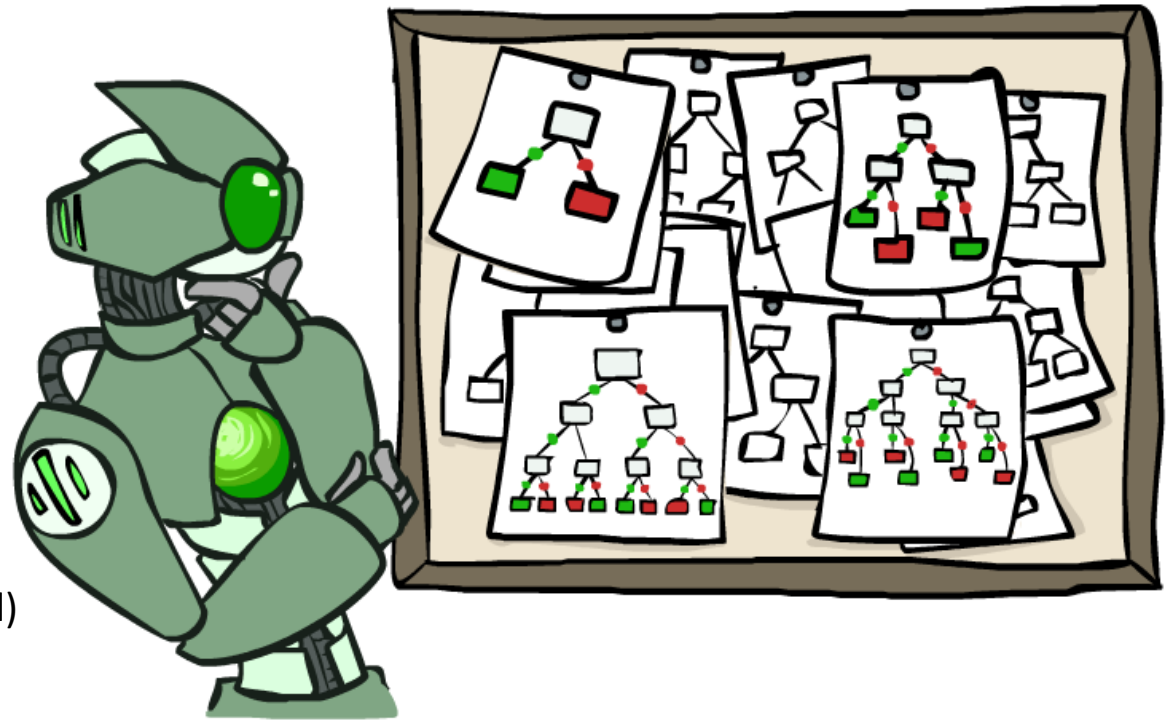
- What is the expressiveness of a perceptron over these features?

Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
$X_1$	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>0-10</i>	<i>T</i>
$X_2$	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30-60</i>	<i>F</i>

- For a perceptron, a feature's contribution is either positive or negative
  - If you want one feature's effect to depend on another, you have to add a new conjunction feature
  - E.g. adding "PATRONS=full  $\wedge$  WAIT = 60" allows a perceptron to model the interaction between the two atomic features
- DTs automatically conjoin features / attributes
  - Features can have different effects in different branches of the tree!
- Difference between modeling relative evidence weighting (NB) and complex evidence interaction (DTs)
  - Though if the interactions are too complex, may not find the DT greedily

# Hypothesis Spaces

- How many distinct decision trees with  $n$  Boolean attributes?
  - = number of Boolean functions over  $n$  attributes
  - = number of distinct truth tables with  $2^n$  rows
  - =  $2^{(2^n)}$ 
    - E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
- How many trees of depth 1 (decision stumps)?
  - = number of Boolean functions over 1 attribute
  - = number of truth tables with 2 rows, times  $n$
  - =  $4n$ 
    - E.g. with 6 Boolean attributes, there are 24 decision stumps
- More expressive hypothesis space:
  - Increases chance that target function can be expressed (good)
  - Increases number of hypotheses consistent with training set (bad, why?)
  - Means we can get better predictions (lower **bias**)
  - But we may get worse predictions (higher **variance**)



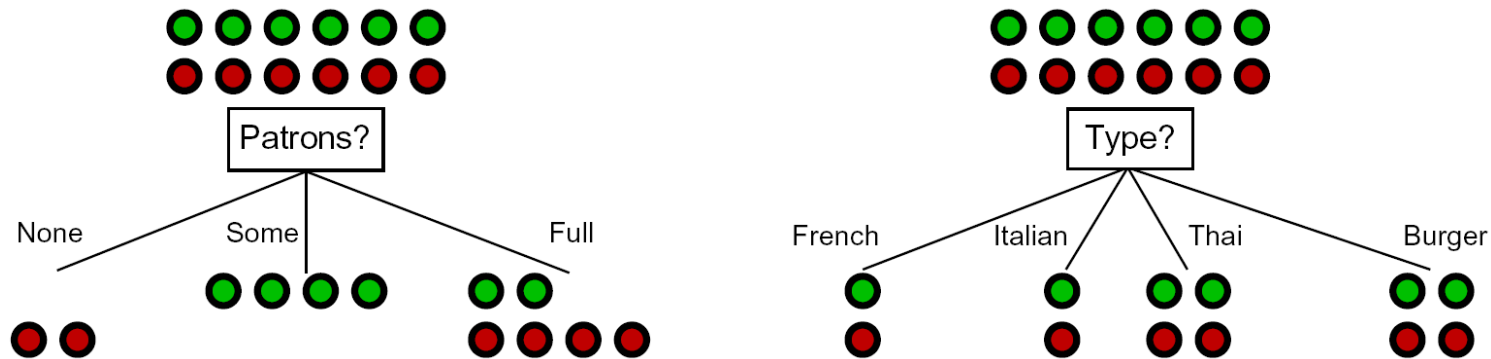
# Decision Tree Learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose “most significant” attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
    best ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test best
    for each value  $v_i$  of best do
       $examples_i$  ← {elements of examples with  $best = v_i$ }
      subtree ← DTL( $examples_i$ , attributes – best, MODE(examples))
      add a branch to tree with label  $v_i$  and subtree subtree
  return tree
```

# Choosing an Attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”



- So: we need a measure of how “good” a split is, even if the results aren’t perfectly separated out

# Entropy and Information

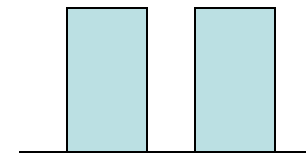
- **Information** answers questions
  - The more uncertain about the answer initially, the more information in the answer
  - Scale: bits
    - Answer to Boolean question with prior  $\langle 1/2, 1/2 \rangle$ ?
    - Answer to 4-way question with prior  $\langle 1/4, 1/4, 1/4, 1/4 \rangle$ ?
    - Answer to 4-way question with prior  $\langle 0, 0, 0, 1 \rangle$ ?
    - Answer to 3-way question with prior  $\langle 1/2, 1/4, 1/4 \rangle$ ?
- A probability  $p$  is typical of:
  - A uniform distribution of size  $1/p$
  - A code of length  $\log 1/p$

# Entropy

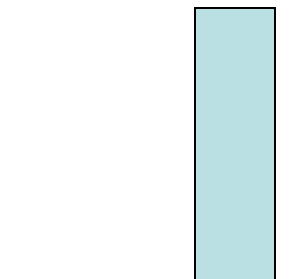
- General answer: if prior is  $\langle p_1, \dots, p_n \rangle$ :
  - Information is the expected code length

$$\begin{aligned} H(\langle p_1, \dots, p_n \rangle) &= E_p \log_2 1/p_i \\ &= \sum_{i=1}^n -p_i \log_2 p_i \end{aligned}$$

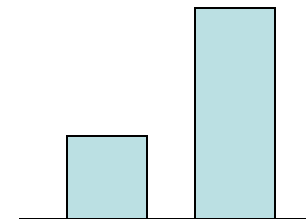
- Also called the **entropy** of the distribution
  - More uniform = higher entropy
  - More values = higher entropy
  - More peaked = lower entropy
  - Rare values almost “don’t count”



1 bit



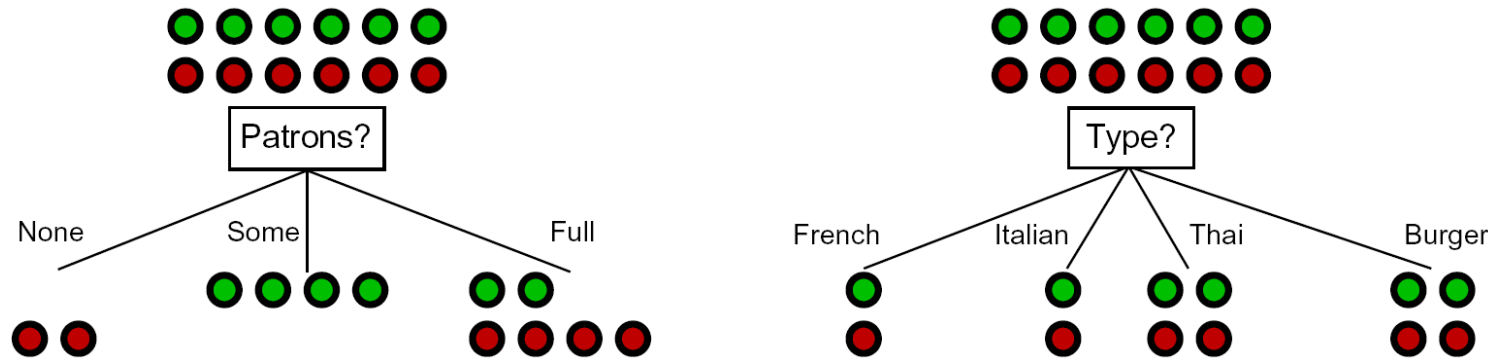
0 bits



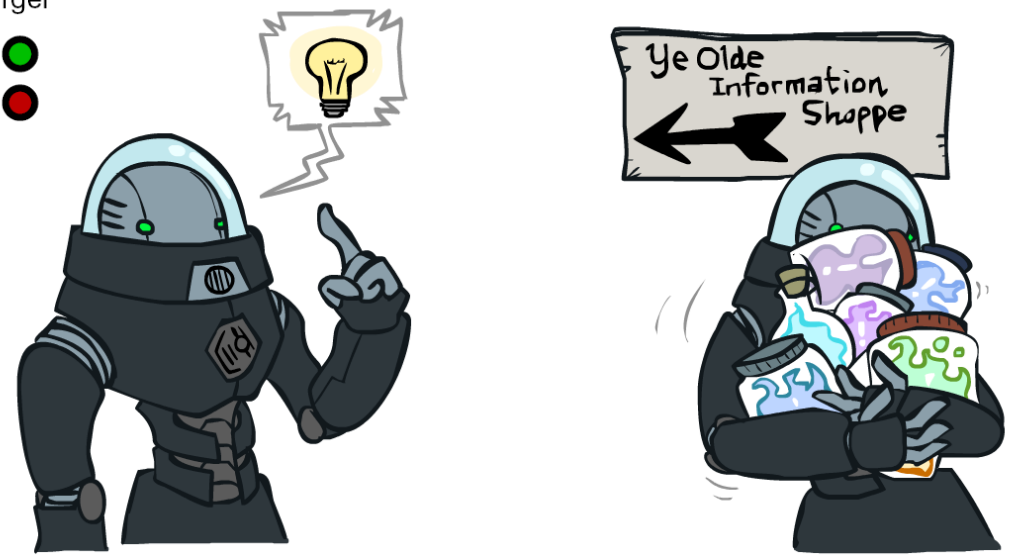
0.5 bit

# Information Gain

- Back to decision trees!
- For each split, compare entropy before and after
  - Difference is the **information gain**
  - Problem: there's more than one distribution after split!



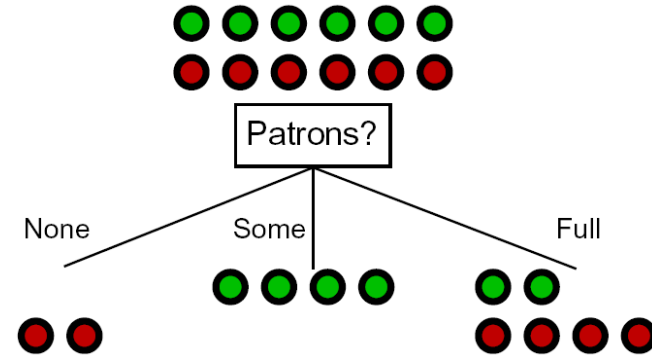
- Solution: use **expected entropy**, weighted by the number of examples





# Next Step: Recurse

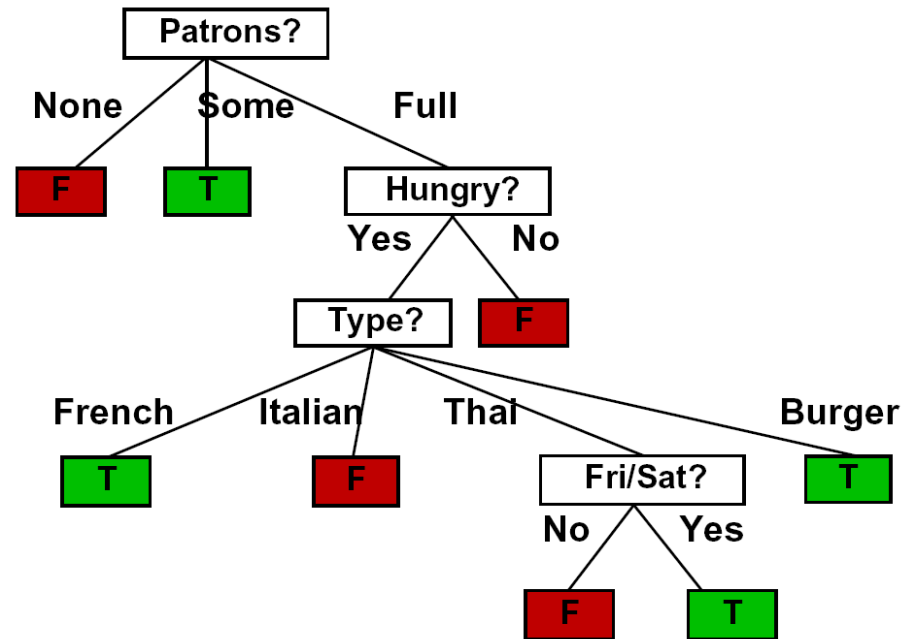
- Now we need to keep growing the tree!
- Two branches are done (why?)
- What to do under “full”?
  - See what examples are there...



Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
$X_1$	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>0-10</i>	<i>T</i>
$X_2$	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30-60</i>	<i>F</i>
$X_3$	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>Some</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>T</i>
$X_4$	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>10-30</i>	<i>T</i>
$X_5$	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>&gt;60</i>	<i>F</i>
$X_6$	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Italian</i>	<i>0-10</i>	<i>T</i>
$X_7$	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>F</i>
$X_8$	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Thai</i>	<i>0-10</i>	<i>T</i>
$X_9$	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>&gt;60</i>	<i>F</i>
$X_{10}$	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>Italian</i>	<i>10-30</i>	<i>F</i>
$X_{11}$	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>0-10</i>	<i>F</i>
$X_{12}$	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>30-60</i>	<i>T</i>

# Example: Learned Tree

- Decision tree learned from these 12 examples:



- Substantially simpler than “true” tree
  - A more complex hypothesis isn't justified by data
- Also: it's reasonable, but wrong

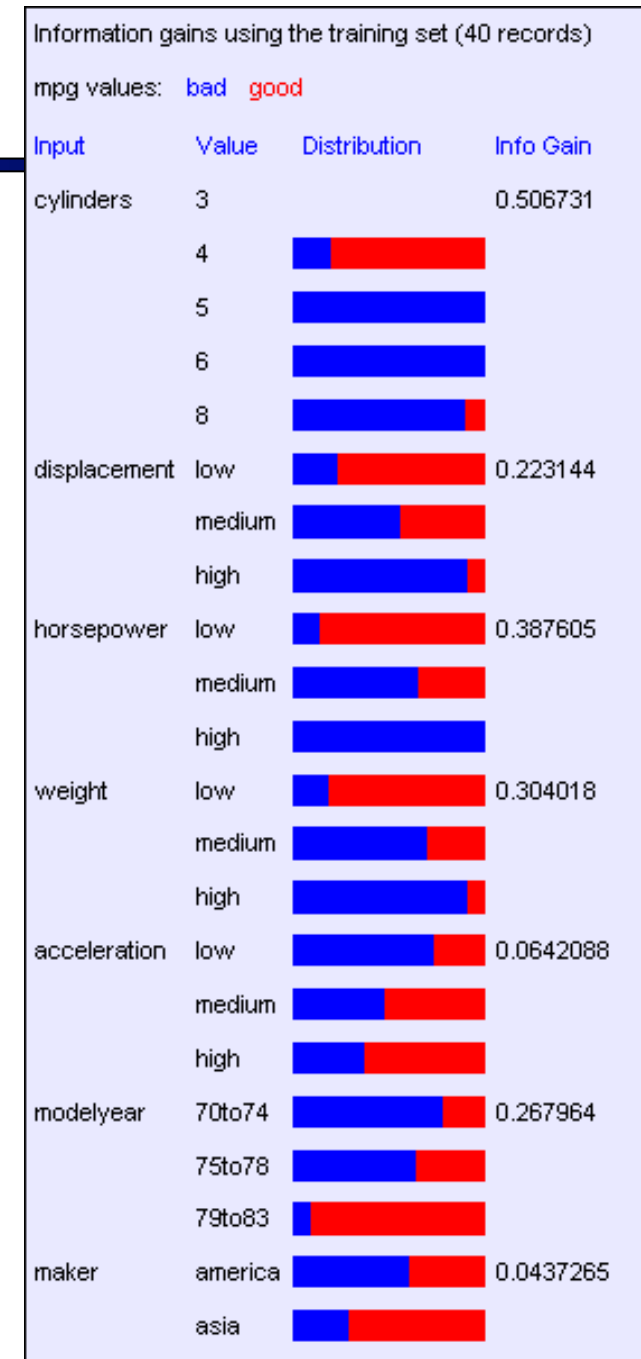
# Example: Miles Per Gallon

40 Examples

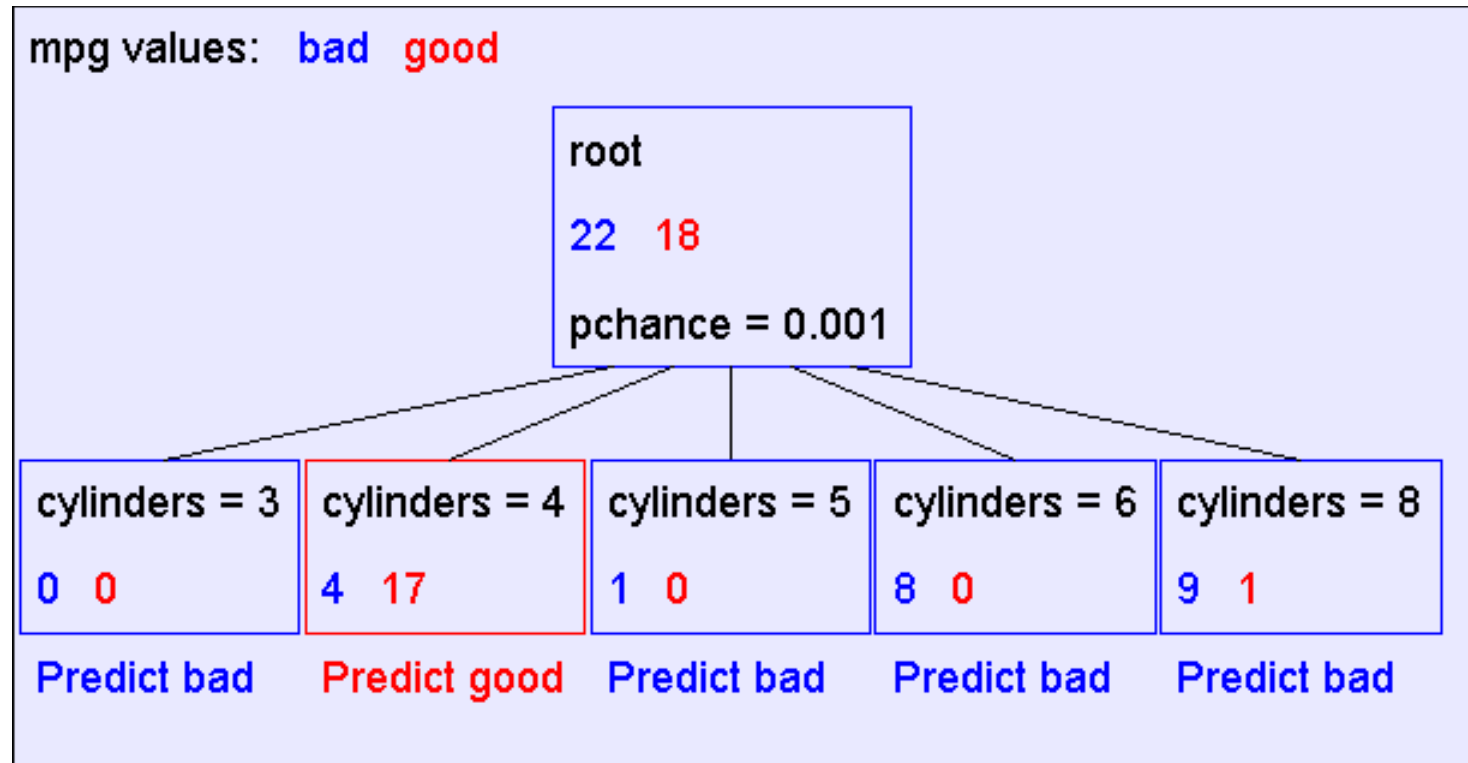
mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europa
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america
bad	8	high	high	high	low	75to78	america
good	4	low	low	low	low	79to83	america
bad	6	medium	medium	medium	high	75to78	america
good	4	medium	low	low	low	79to83	america
good	4	low	low	medium	high	79to83	america
bad	8	high	high	high	low	70to74	america
good	4	low	medium	low	medium	75to78	europa
bad	5	medium	medium	medium	medium	75to78	europa

# Find the First Split

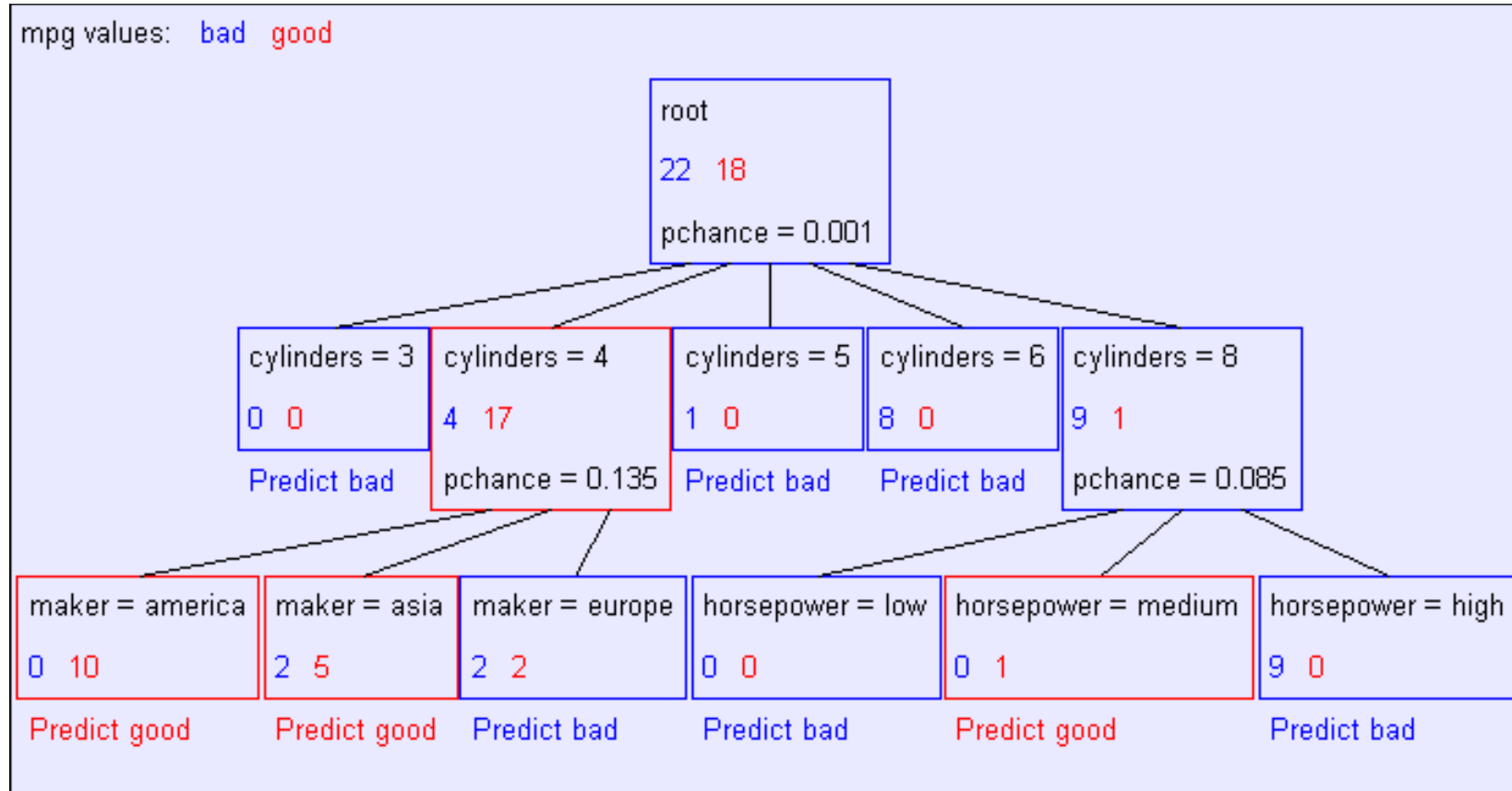
- Look at information gain for each attribute
- Note that each attribute is correlated with the target!
- What do we split on?



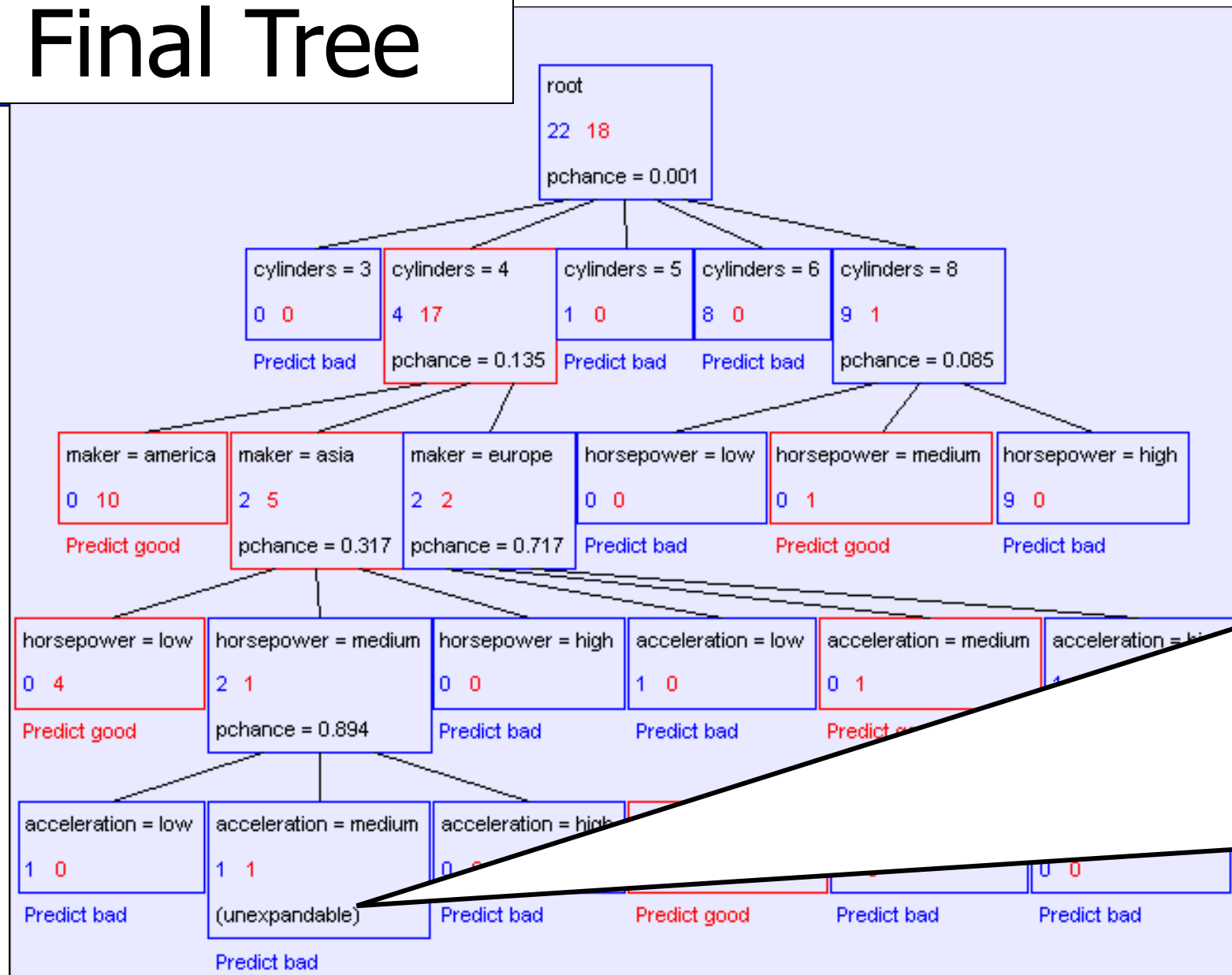
# Result: Decision Stump



# Second Level



# Final Tree



Information gains using the training set (2 records)

mpg values: bad good

Input	Value	Distribution	Info Gain
cylinders	3		0
	4		
	5		
	6		
	8		
displacement	low		0
	medium		
	high		
horsepower	low		0
	medium		
	high		
weight	low		0
	medium		
	high		
acceleration	low		0
	medium		
	high		
modelyear	70to74		0
	75to78		
	79to83		
maker	america		0
	asia		
	europe		

# Reminder: Overfitting

---

- **Overfitting:**
  - When you stop modeling the patterns in the training data (which generalize)
  - And start modeling the noise (which doesn't)
- **We had this before:**
  - Naïve Bayes: needed to smooth
  - Perceptron: early stopping



# MPG Training Error

mpg values: bad good

root  
22 18  
pchance = 0.001

	Num Errors	Set Size	Percent Wrong
Training Set	1	40	2.50
Test Set	74	352	21.02

power = high

ict bad

horsepower = low

horsepower = medium

horsepower = high

acceleration = low

acceleration = medium

acceleration = high

0

Pr

ad

1

Predict bad

(unexpandable)

Predict bad

Predict good

Predict bad

Predict bad

= 0.717

= 79to83

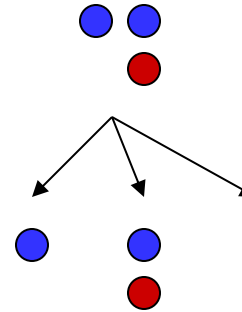
The test set error is much worse than the training set error...

...why?



# Significance of a Split

- Starting with:
  - Three cars with 4 cylinders, from Asia, with medium HP
  - 2 bad MPG
  - 1 good MPG
- What do we expect from a three-way split?
  - Maybe each example in its own subset?
  - Maybe just what we saw in the last slide?
- Probably shouldn't split if the counts are so small they could be due to chance
- A chi-squared test can tell us how likely it is that deviations from a perfect split are due to chance\*
- Each split will have a **significance value**,  $p_{\text{CHANCE}}$



# Keeping it General

## ■ Pruning:

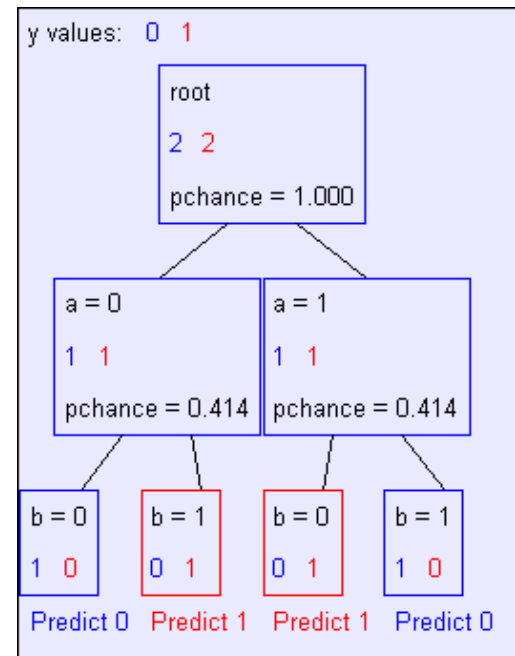
- Build the full decision tree
- Begin at the bottom of the tree
- Delete splits in which

$$P_{\text{CHANCE}} > \text{Max}P_{\text{CHANCE}}$$

- Continue working upward until there are no more prunable nodes
- Note: some chance nodes may not get pruned because they were “redeemed” later

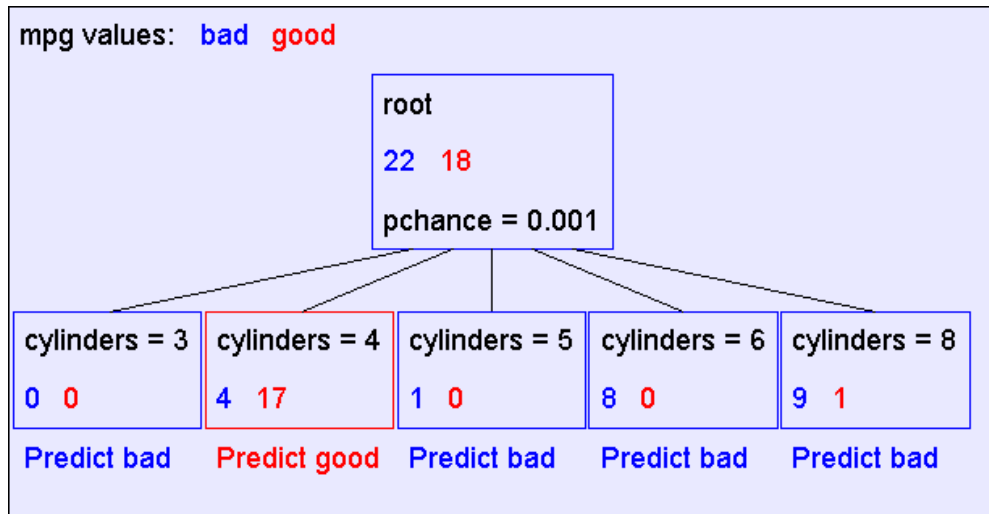
$$y = a \text{ XOR } b$$

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0



# Pruning example

- With  $\text{MaxP}_{\text{CHANCE}} = 0.1$ :

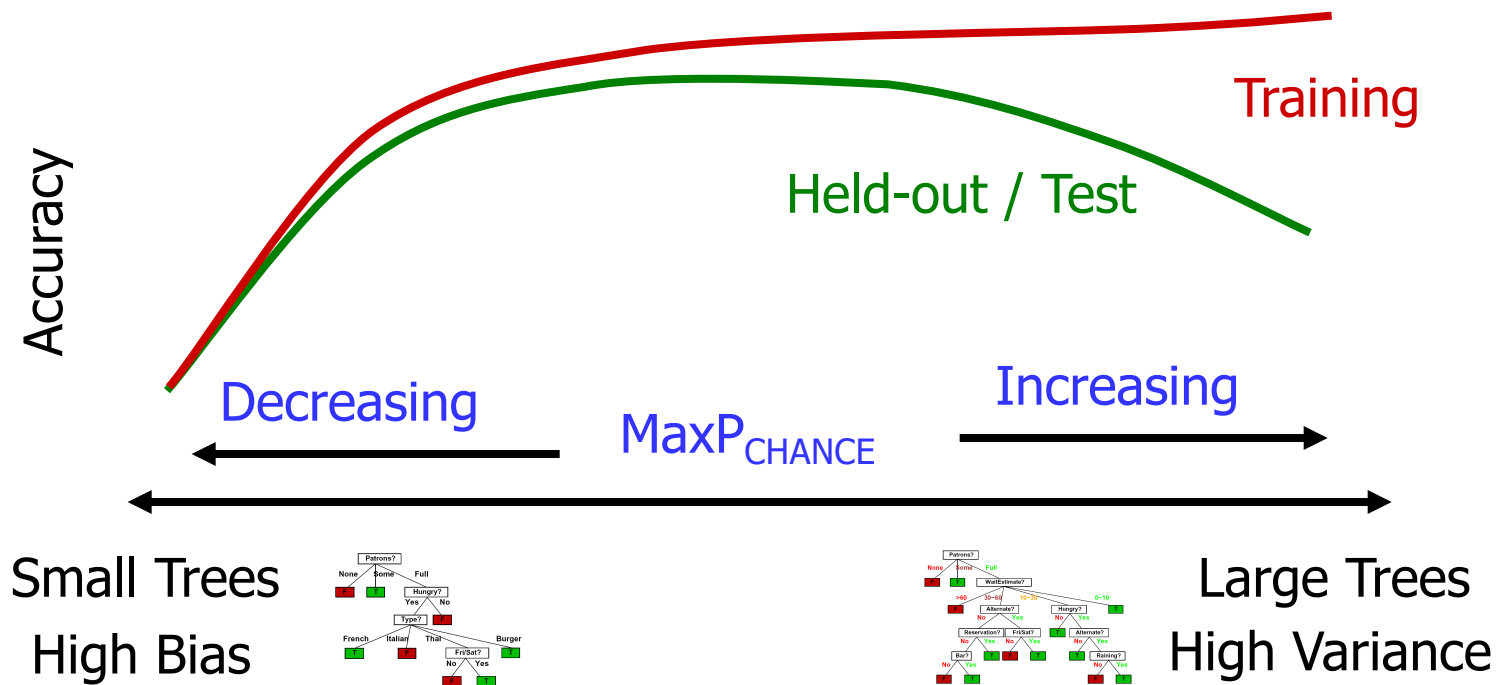


Note the improved test set accuracy compared with the unpruned tree

	Num Errors	Set Size	Percent Wrong
Training Set	5	40	12.50
Test Set	56	352	15.91

# Regularization

- $\text{MaxP}_{\text{CHANCE}}$  is a regularization parameter
- Generally, set it using held-out data (as usual)



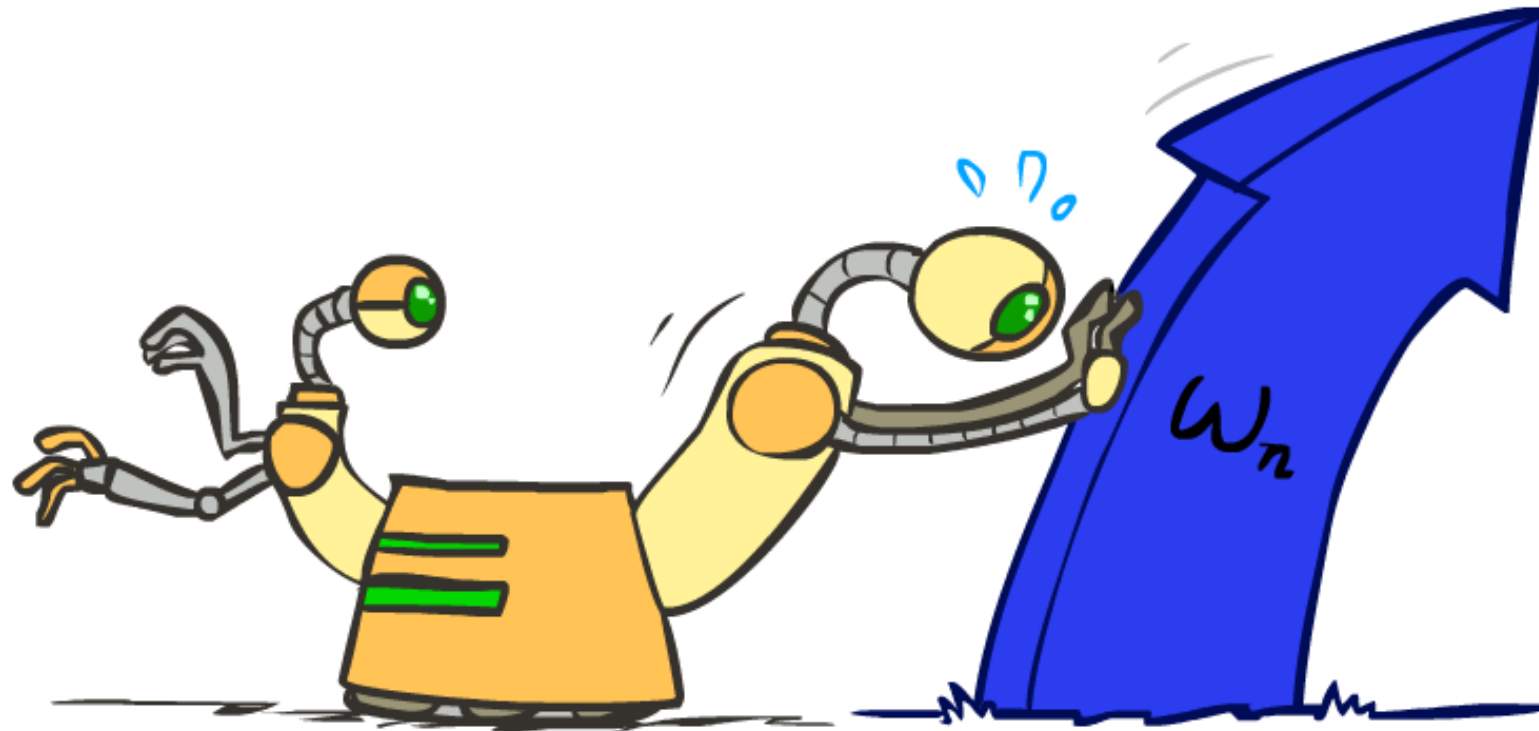
# Two Ways of Controlling Overfitting

---

- Limit the hypothesis space
  - E.g. limit the max depth of trees
  - Easier to analyze
- Regularize the hypothesis selection
  - E.g. chance cutoff
  - Disprefer most of the hypotheses unless data is clear
  - Usually done in practice

# Other Optimizers

---





# Recall: Batch Gradient Ascent

$$\max_w ll(w) = \max_w \underbrace{\sum_i \log P(y^{(i)} | x^{(i)}; w)}_{g(w)}$$

- `init  $w$`
- `for iter = 1, 2, ...`

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)} | x^{(i)}; w)$$

# Recall: Stochastic Gradient Ascent

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

**Observation:** once gradient on one training example has been computed, might as well incorporate before computing next one

- `init  $w$`
- `for iter = 1, 2, ...`
  - `pick random  $j$`

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)} | x^{(j)}; w)$$

# Recall: Mini-batch Gradient Ascent

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

**Observation:** gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- `init w`
- `for iter = 1, 2, ...`
  - pick random subset of training examples  $J$

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)} | x^{(j)}; w)$$

# Concept #1: Computing Second-Order Derivatives

## Newton's Method (in 1D):

- Want to optimize:  $\max_{\theta} f(\theta)$
- Apply Taylor expansion:

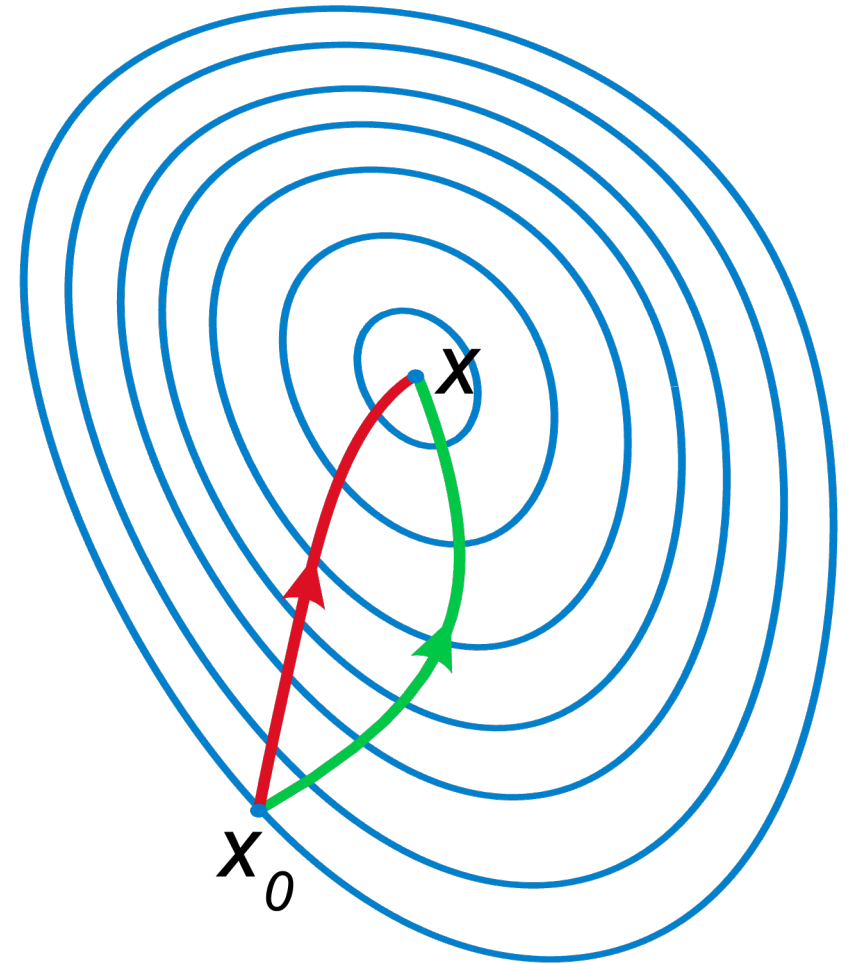
$$f(\theta + h) = f(\theta) + f'(\theta)h + \frac{1}{2}f''(\theta)h^2$$

- Find value of  $t$  that maximizes this:

$$\begin{aligned} 0 &= \frac{\partial}{\partial h} \left[ f(\theta) + f'(\theta)h + \frac{1}{2}f''(\theta)h^2 \right] \\ &= f'(\theta) + f''(\theta)h \end{aligned}$$

- Rearrange terms to get update:

$$h = -\frac{f'(\theta)}{f''(\theta)} \quad \theta_{t+1} = \theta_t + h = \theta_t - \frac{f'(\theta)}{f''(\theta)}$$



Caveat: can be computationally expensive or difficult to compute in higher dimensions

# Concept #2: Momentum

- Potential issues with vanilla SGD:
  - Can take a long time to converge if the learning rate is too low
  - Can bounce around in “ravines” without making much progress toward a local optimum

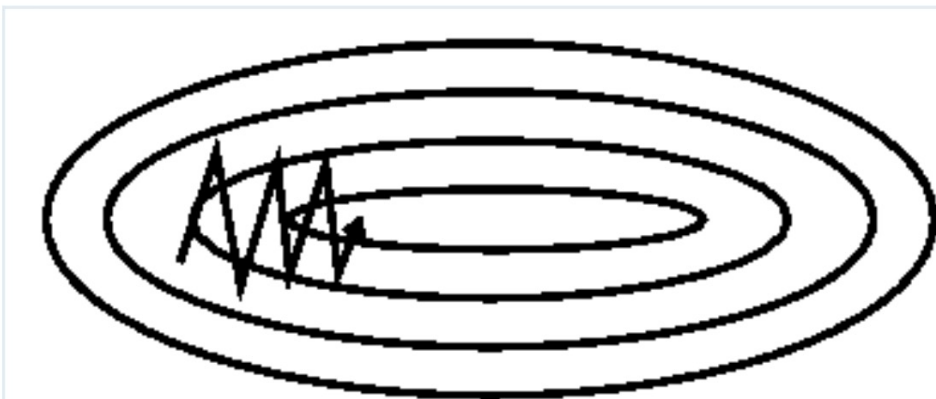


Image 2: SGD without momentum

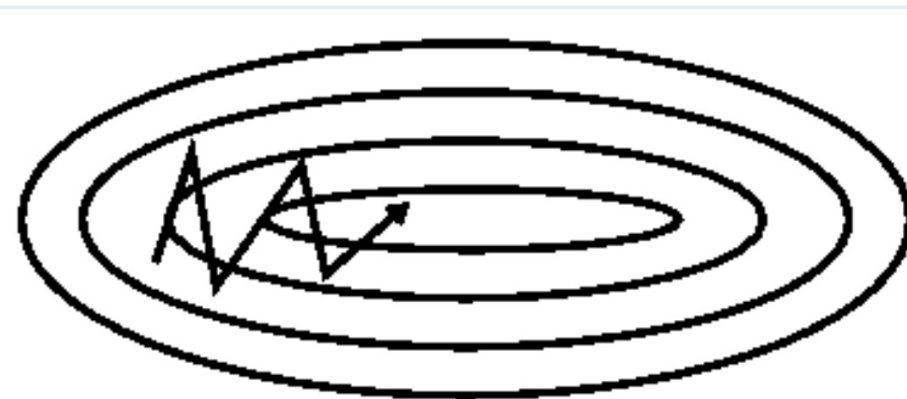


Image 3: SGD with momentum

# Concept #2: Momentum

## Stochastic Gradient Descent with Momentum

- Keep a running sum of old updates:  $v_t = \gamma v_{t-1} + \eta \nabla_{\theta} f(\theta_t)$
- Perform a standard gradient descent step:  $\theta_{t+1} = \theta_t - v_t$

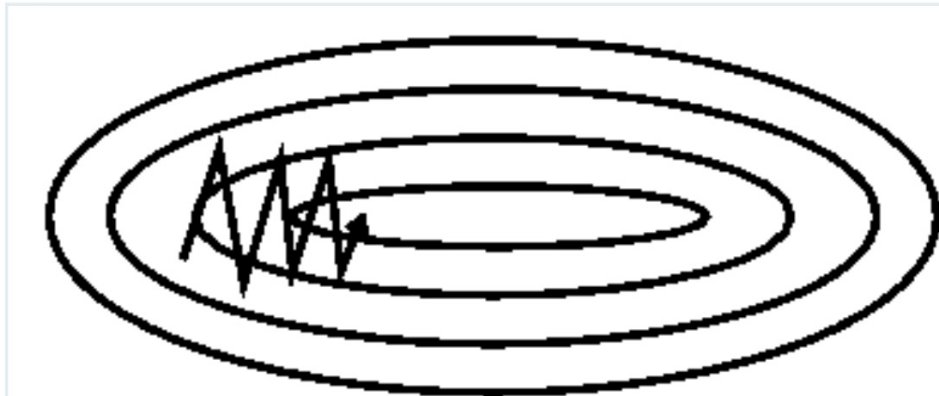


Image 2: SGD without momentum

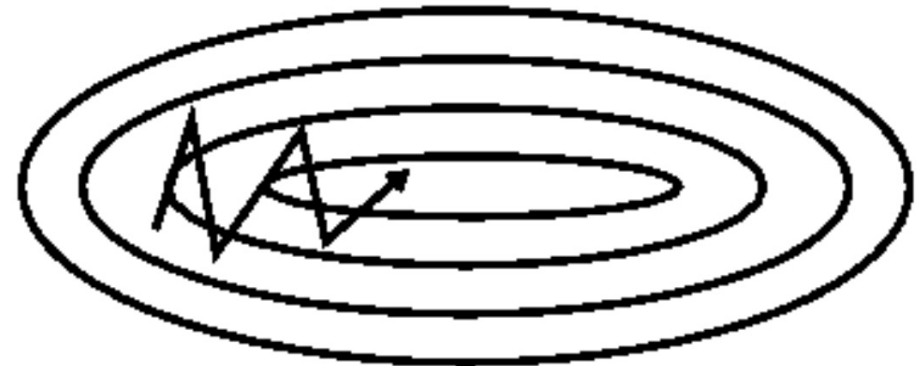


Image 3: SGD with momentum

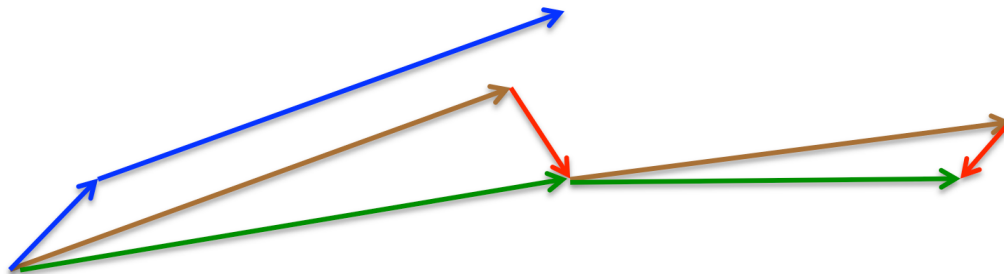
# Concept #2: Momentum

## Nesterov Accelerated Gradient (NAG)

- Key idea: “anticipate” where momentum will take you and compute the gradient at that point instead. Can think of this as momentum with planning

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} f(\theta_t - \gamma v_{t-1})$$

$$\theta_{t+1} = \theta_t - v_t$$



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

# Concept #3: Adaptive Learning Rates

- Recall: learning rates

- Determines how much we update weights in the direction of the gradient
- Often: want to set this in terms of how much it updates the weights
- Often: want to lower learning rate over time (*learning rate scheduling*)

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} f(\theta_t)$$

- Key idea: different learning rates for each parameter

- We can make larger or smaller updates depending on how important a feature is
- Small updates for frequent features; big updates for rare features
- This idea underlies: Adagrad, RMSProp, Adam, etc.



# Concept #3: Adaptive Learning Rates

## Adagrad

- Compute the gradient at  $i$ -th parameter:

$$g_{t,i} = \nabla_{\theta}(\theta_{t,i})$$

- Make an update based on adaptive learning rate:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} - \epsilon}} \cdot g_{t,i}$$

- Where each  $G_t$  is a diagonal matrix where entry  $i, i$  is the sum of squares of the gradients up to timestep  $t$ 
  - Biggest issue with this method: sum of squares of gradients continues to grow over time

# Concept #3: Adaptive Learning Rates

## RMSProp

- Based on decaying running average of gradients:

$$E[g^2]_t = 0.9 \cdot E[g^2]_{t-1} + 0.1 \cdot g_t^2$$

- Adaptive learning rate based on running average instead of accumulated sum:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_{t,i} + \epsilon}} \cdot g_{t,i}$$

# Concept #3: Adaptive Learning Rates

## Adaptive Moment Estimation (Adam)

- Combining momentum with adaptive learning rates

$$\begin{aligned}m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2\end{aligned}$$

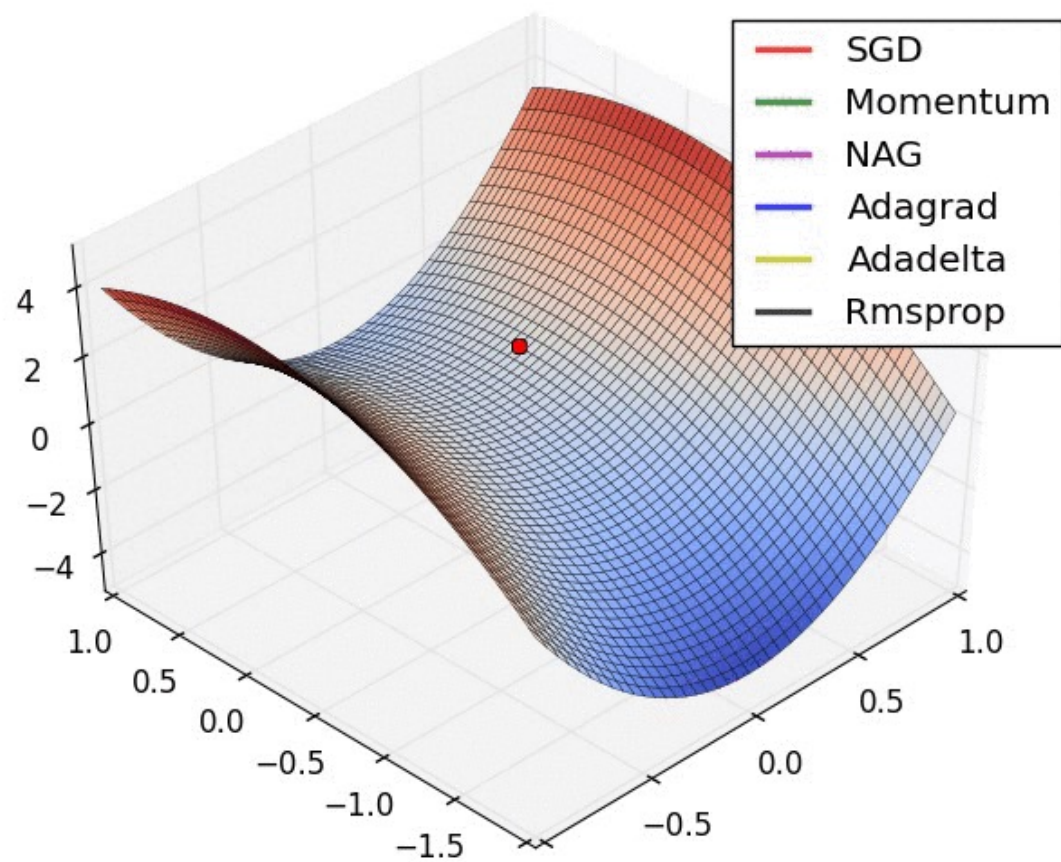
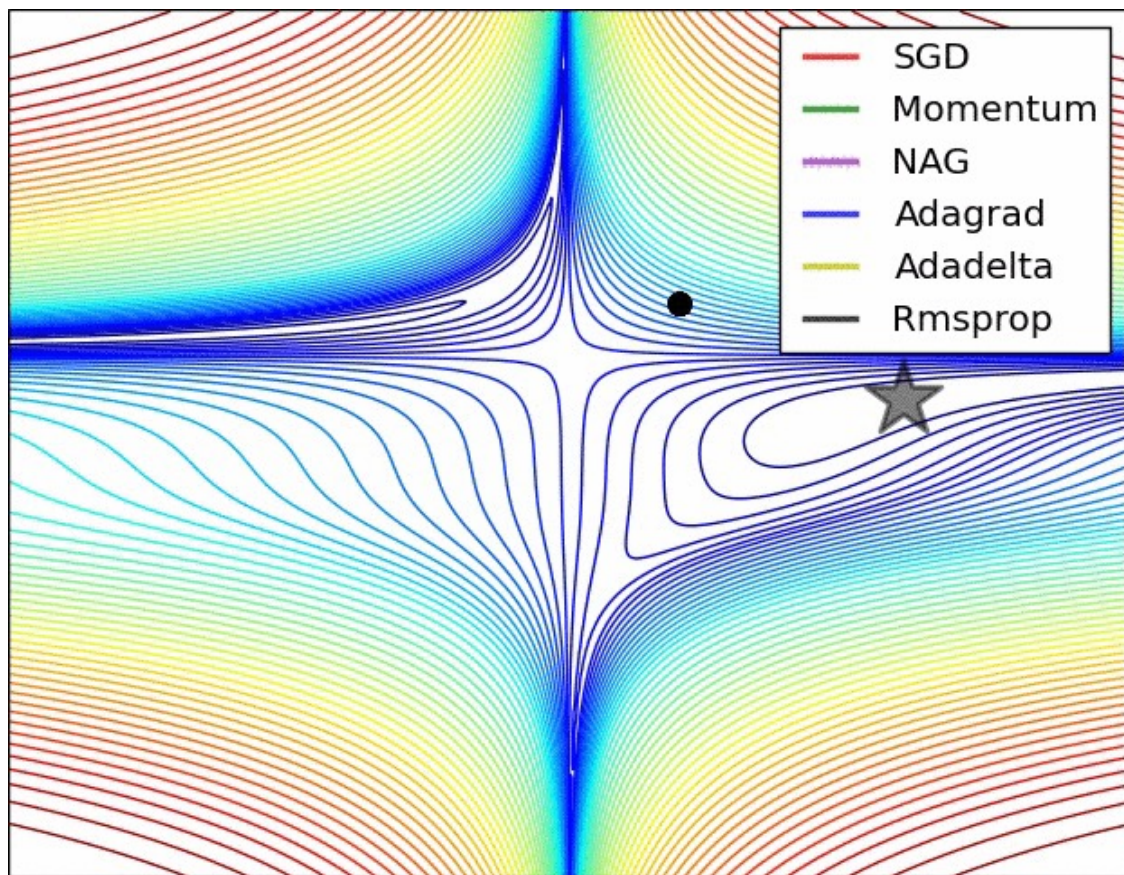
- Bias correction to prevent running averages from tending toward zero:

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1} \qquad \widehat{v}_t = \frac{v_t}{1 - \beta_2}$$

- Then update parameters:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v}_t} + \epsilon} \widehat{m}_t$$

# Visualizations



# Summary: Key Ideas in Optimization

---

- Gradient descent
  - Batch: update based on the whole dataset
  - SGD: update based on a single randomly chosen training example
  - Minibatch: update based on  $k$  randomly chosen training examples
- Beyond gradient descent:
  - Second order optimization (e.g., Newton's method)
  - Momentum (Nesterov's accelerated gradient, Adam)
  - Adaptive learning rates (Adagrad, RMSProp, Adam, etc.)

Learn more: <https://www.ruder.io/optimizing-gradient-descent/>