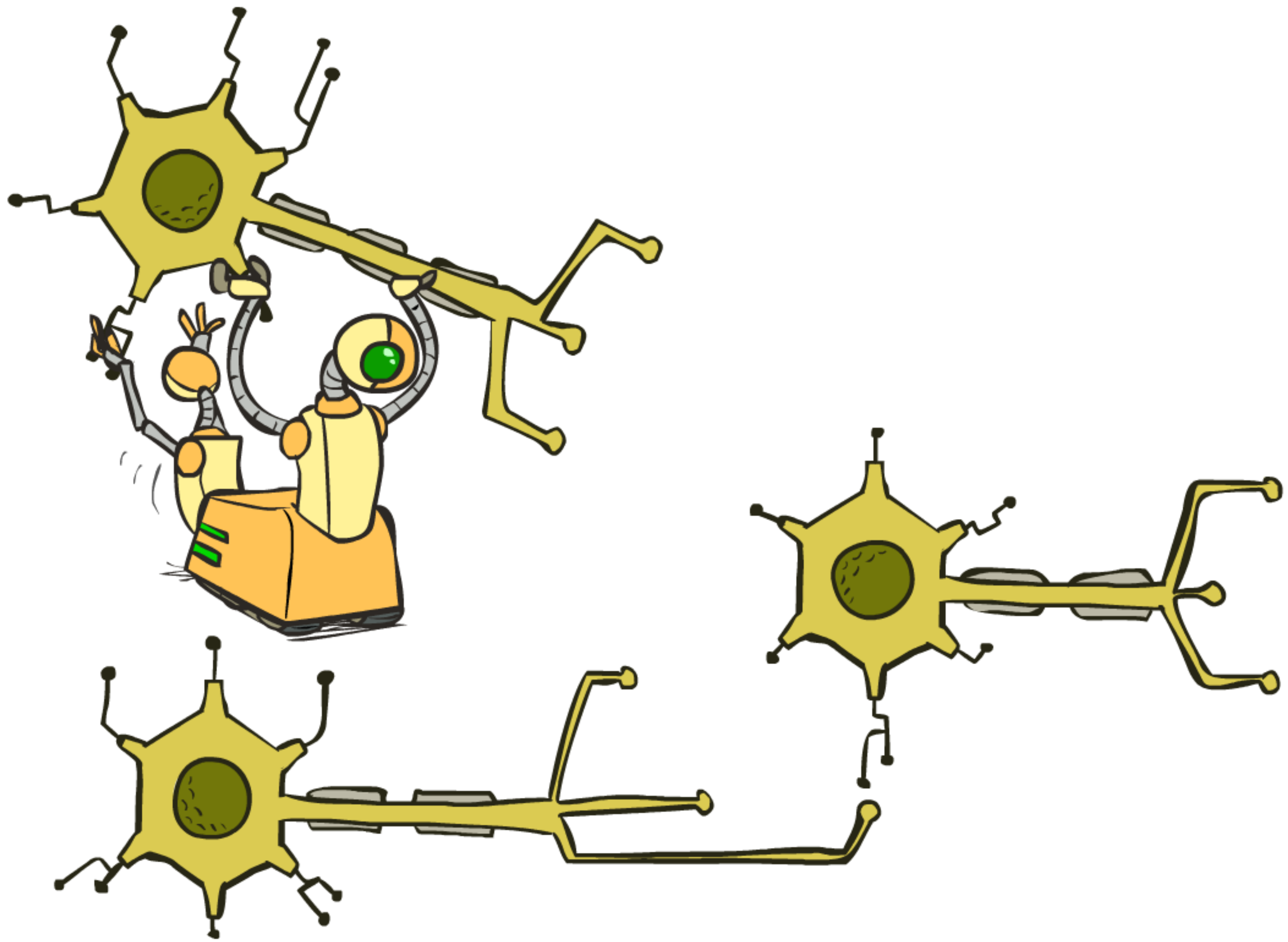
CS 188: Artificial Intelligence

Neural Nets and Decision Trees

Instructors: Sergey Levine and Stuart Russell --- University of California, Berkeley

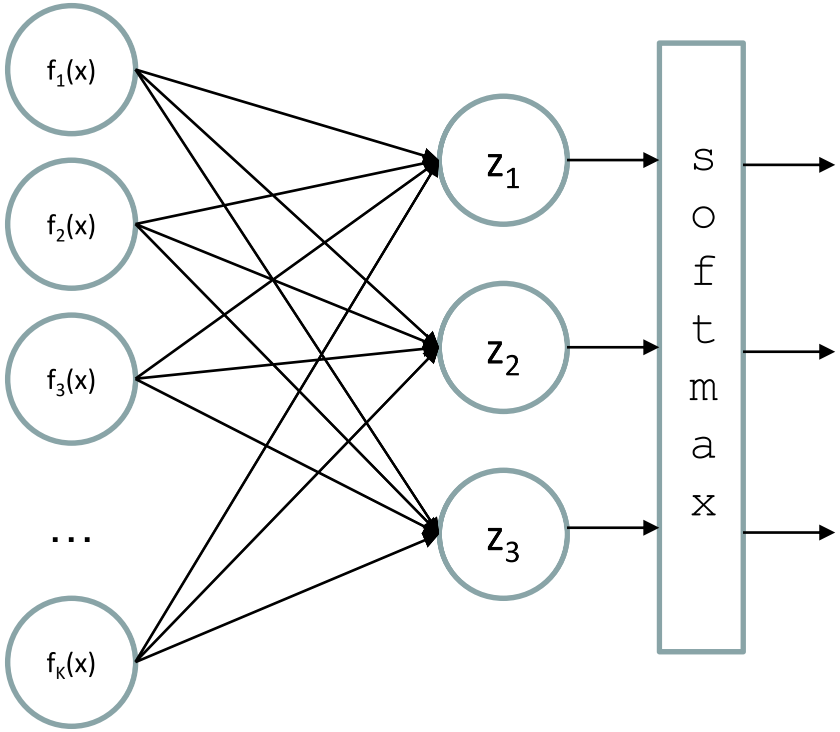
[These slides were created by Dan Klein, Pieter Abbeel, Sergey Levine. All CS188 materials are at http://ai.berkeley.edu.]

Neural Networks



Multi-class Logistic Regression

- = special case of neural network

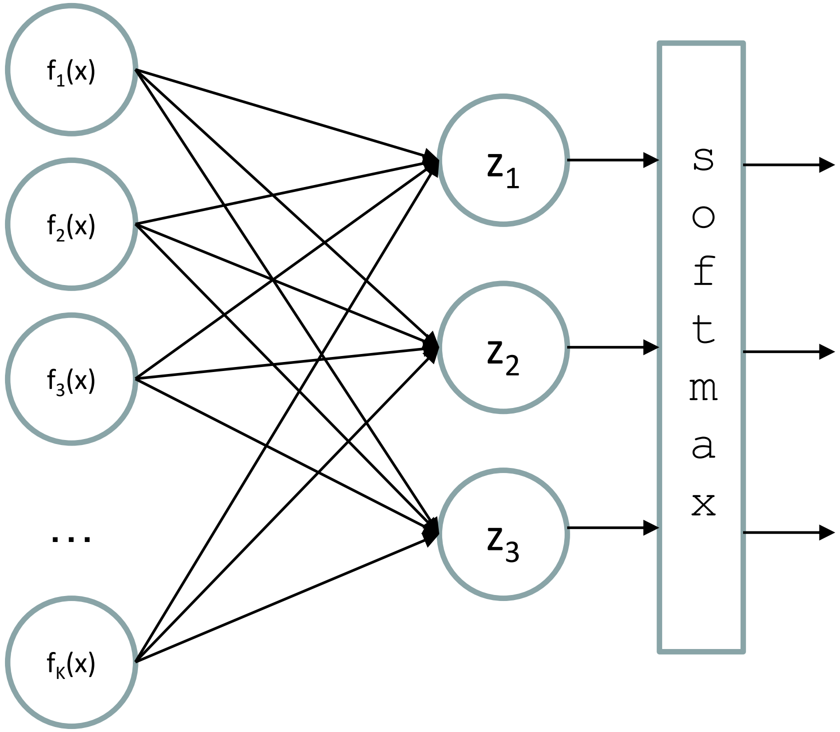


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Deep Neural Network = Also learn the features!

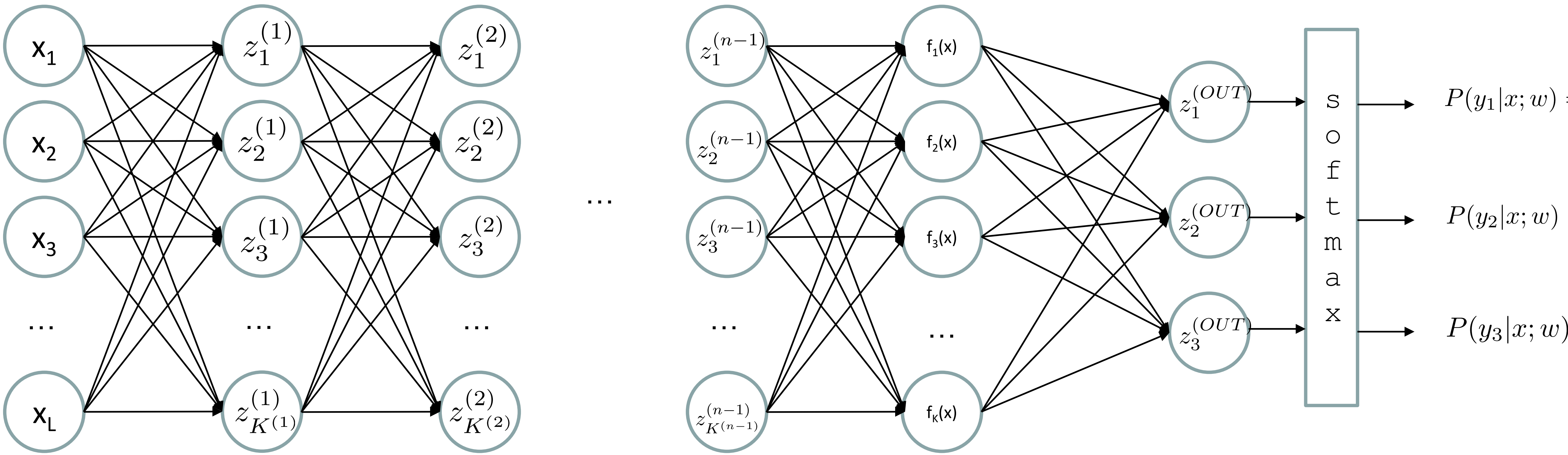


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

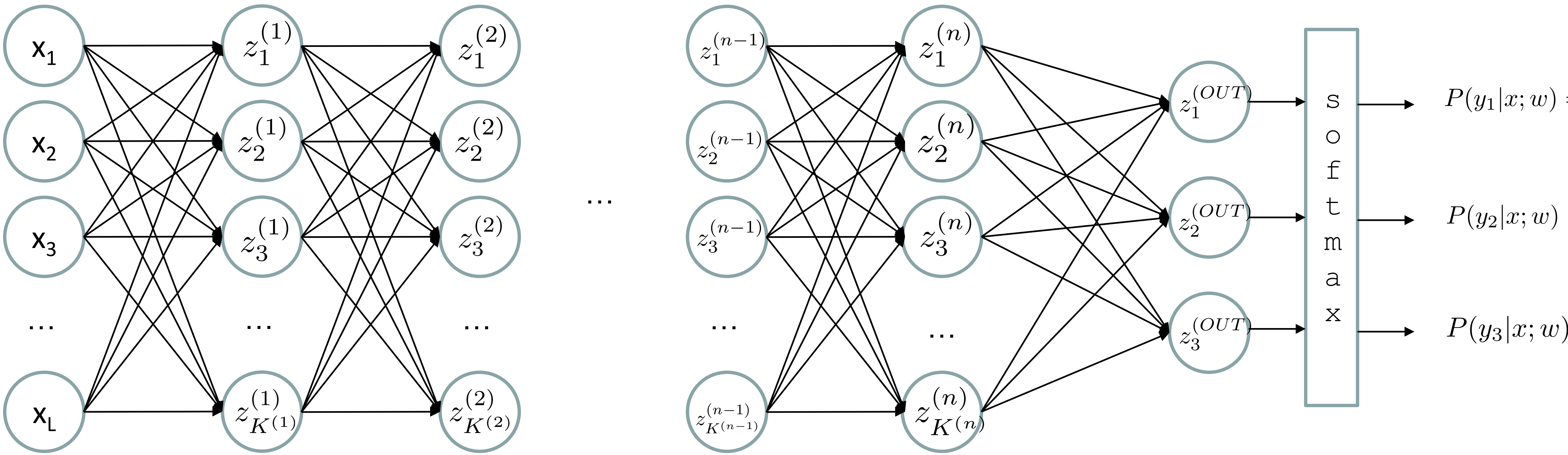
Deep Neural Network = Also learn the features!



$$z_i^{(k)} = g\left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}\right)$$

g = nonlinear activation function

Deep Neural Network = Also learn the features!

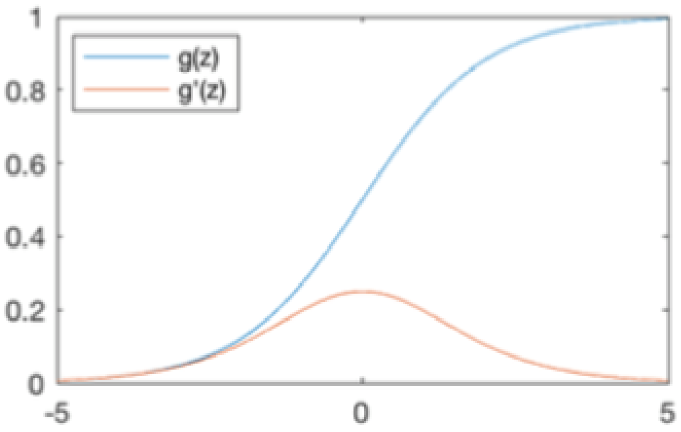


$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Common Activation Functions

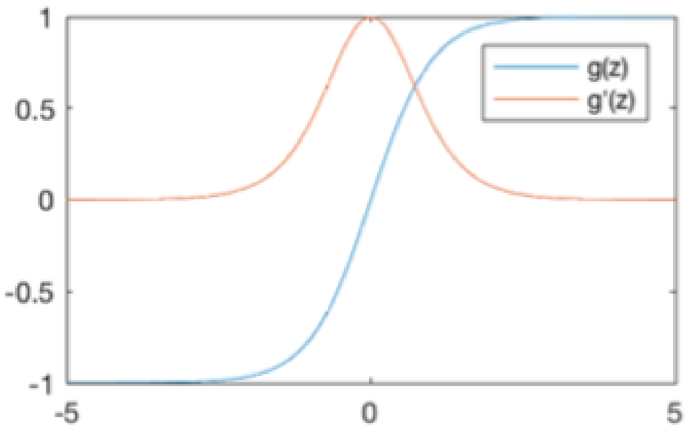
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

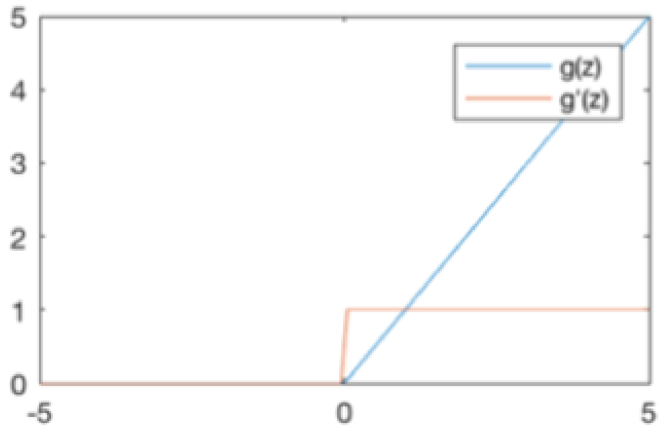
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 introtodeeplearning.com]

Deep Neural Network: Also Learn the Features!

- Training the deep neural network is just like logistic regression:

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

just w tends to be a much, much larger vector ☺

→just run gradient ascent

+ stop when log likelihood of hold-out data starts to decrease

Neural Networks Properties

- Theorem (Universal Function Approximators). A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

- Practical considerations
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
    - (hence early stopping!)

Neural Net Demo!

https://playground.tensorflow.org/

How about computing all the derivatives?

- Derivatives tables:

$$\frac{d}{dx}(a)=0$$

$$\frac{d}{dx}(x)=1$$

$$\frac{d}{dx}(au)=a\frac{du}{dx}$$

$$\frac{d}{dx}(u+v-w)=\frac{du}{dx}+\frac{dv}{dx}-\frac{dw}{dx}$$

$$\frac{d}{dx}(uv)=u\frac{dv}{dx}+v\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{u}{v}\right)=\frac{1}{v}\frac{du}{dx}-\frac{u}{v^2}\frac{dv}{dx}$$

$$\frac{d}{dx}(u^n)=nu^{n-1}\frac{du}{dx}$$

$$\frac{d}{dx}(\sqrt{u})=\frac{1}{2\sqrt{u}}\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u}\right)=-\frac{1}{u^2}\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u^n}\right)=-\frac{n}{u^{n+1}}\frac{du}{dx}$$

$$\frac{d}{dx}[f(u)]=\frac{d}{du}[f(u)]\frac{du}{dx}$$

$$\frac{d}{dx}[\ln u]=\frac{d}{dx}[\log_e u]=\frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}\left[\log_a u\right]=\log_a e\frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}e^u=e^u\frac{du}{dx}$$

$$\frac{d}{dx}a^u=a^u\ln a\frac{du}{dx}$$

$$\frac{d}{dx}\left(u^v\right)=vu^{v-1}\frac{du}{dx}+\ln u\;\; u^v\frac{dv}{dx}$$

$$\frac{d}{dx}\sin u=\cos u\frac{du}{dx}$$

$$\frac{d}{dx}\cos u=-\sin u\frac{du}{dx}$$

$$\frac{d}{dx}\tan u=\sec^2 u\frac{du}{dx}$$

$$\frac{d}{dx}\cot u=-\csc^2 u\frac{du}{dx}$$

$$\frac{d}{dx}\sec u=\sec u\tan u\frac{du}{dx}$$

$$\frac{d}{dx}\csc u=-\csc u\cot u\frac{du}{dx}$$

[source: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/derfunc.html

How about computing all the derivatives?

- But neural net f is never one of those?
  - No problem: CHAIN RULE:

If $$f(x) = g(h(x))$$

Then $$f'(x) = g'(h(x))h'(x)$$

→ **Derivatives can be computed by following well-defined procedures**

Automatic Differentiation

- Automatic differentiation software
  - e.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function g(x,y,w)
  - Can automatically compute all derivatives w.r.t. all entries in w
  - This is typically done by caching info during forward computation pass of f, and then doing a backward pass = "backpropagation"
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass
- Need to know this exists
- How this is done? -- outside of scope of CS188

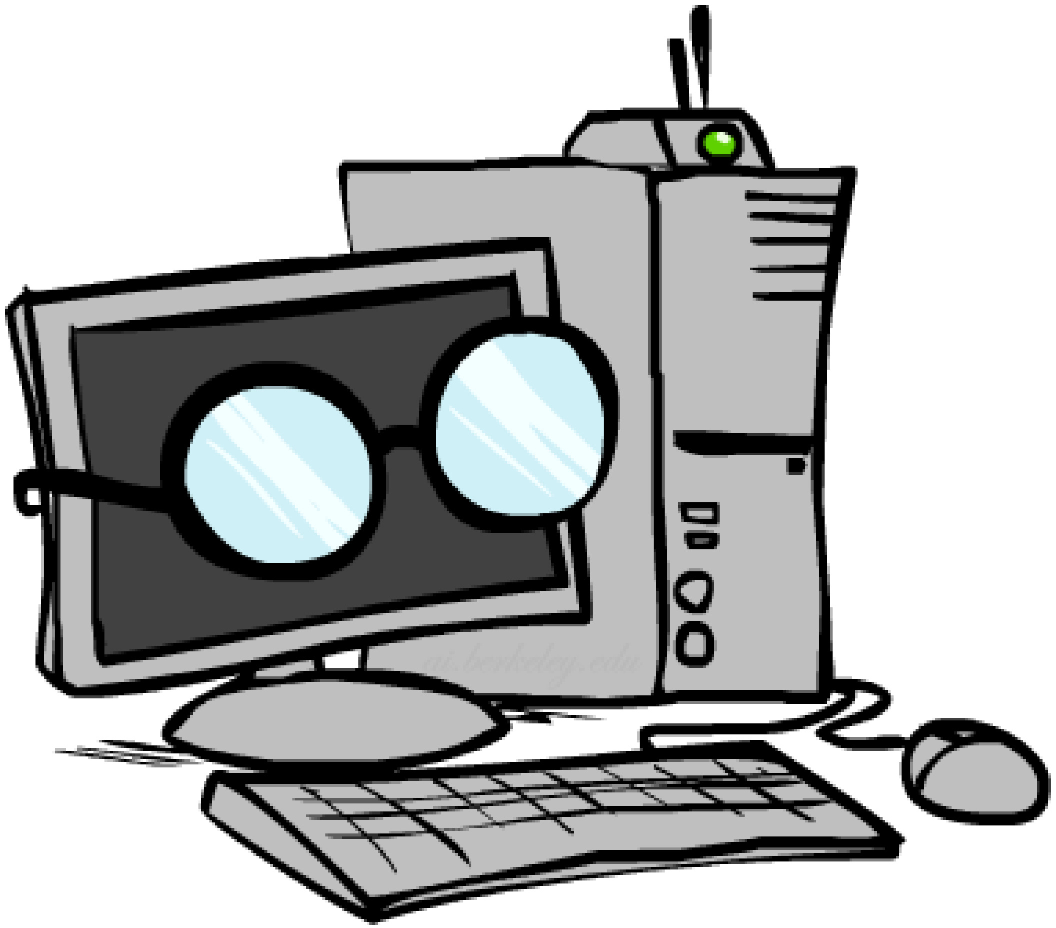
Summary of Key Ideas

- Optimize probability of label given input $\max_w \quad ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$

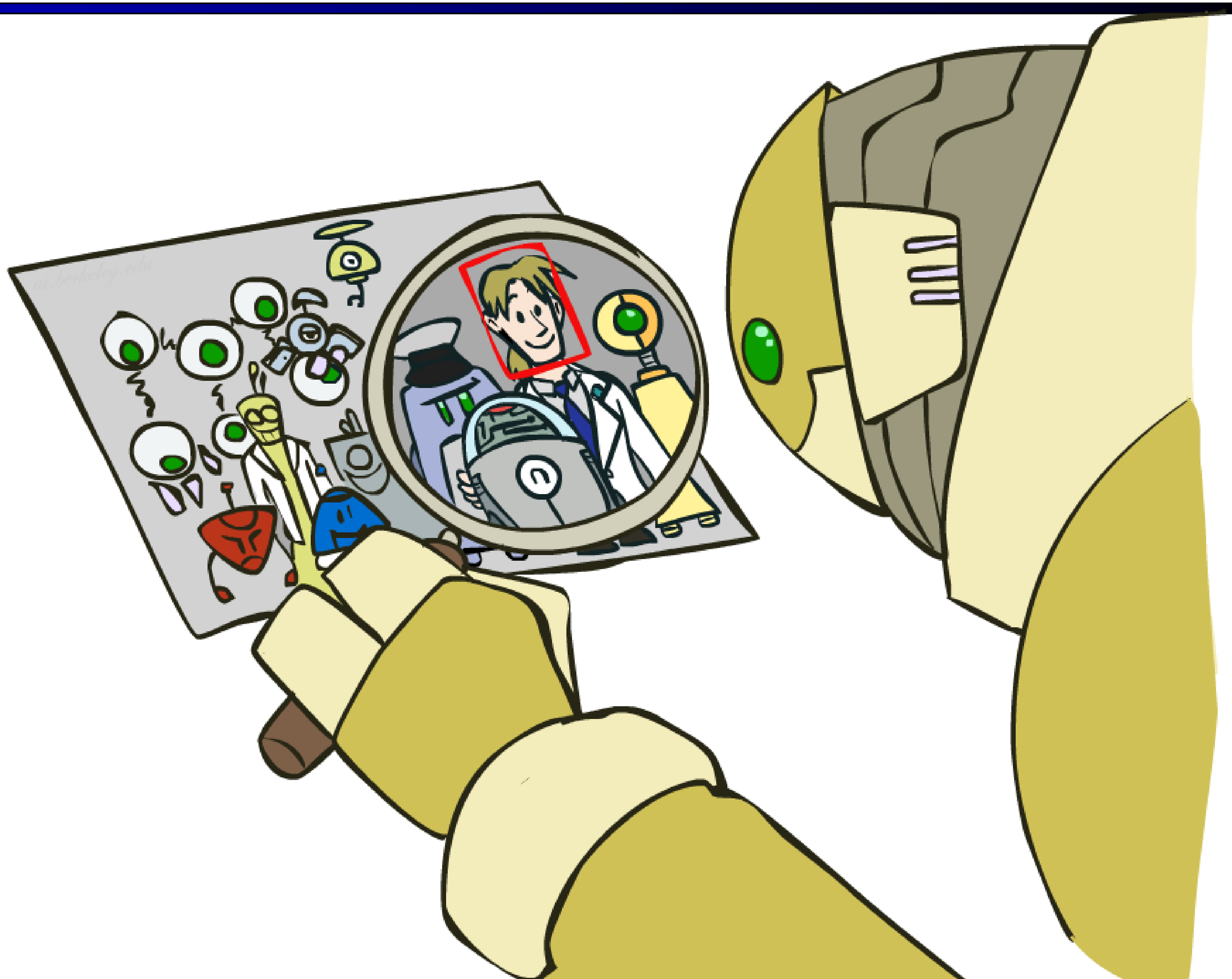
- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
    - Repeat (until held-out data accuracy starts to drop = "early stopping")

- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
    - = computing the features
    - → the features are learned rather than hand-designed
  - Universal function approximation theorem
    - `If` neural net is large enough
    - `Then` neural net can represent any continuous mapping from input to output with arbitrary accuracy
    - But remember: need to avoid overfitting / memorizing the training data → early stopping!
  - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of 188)

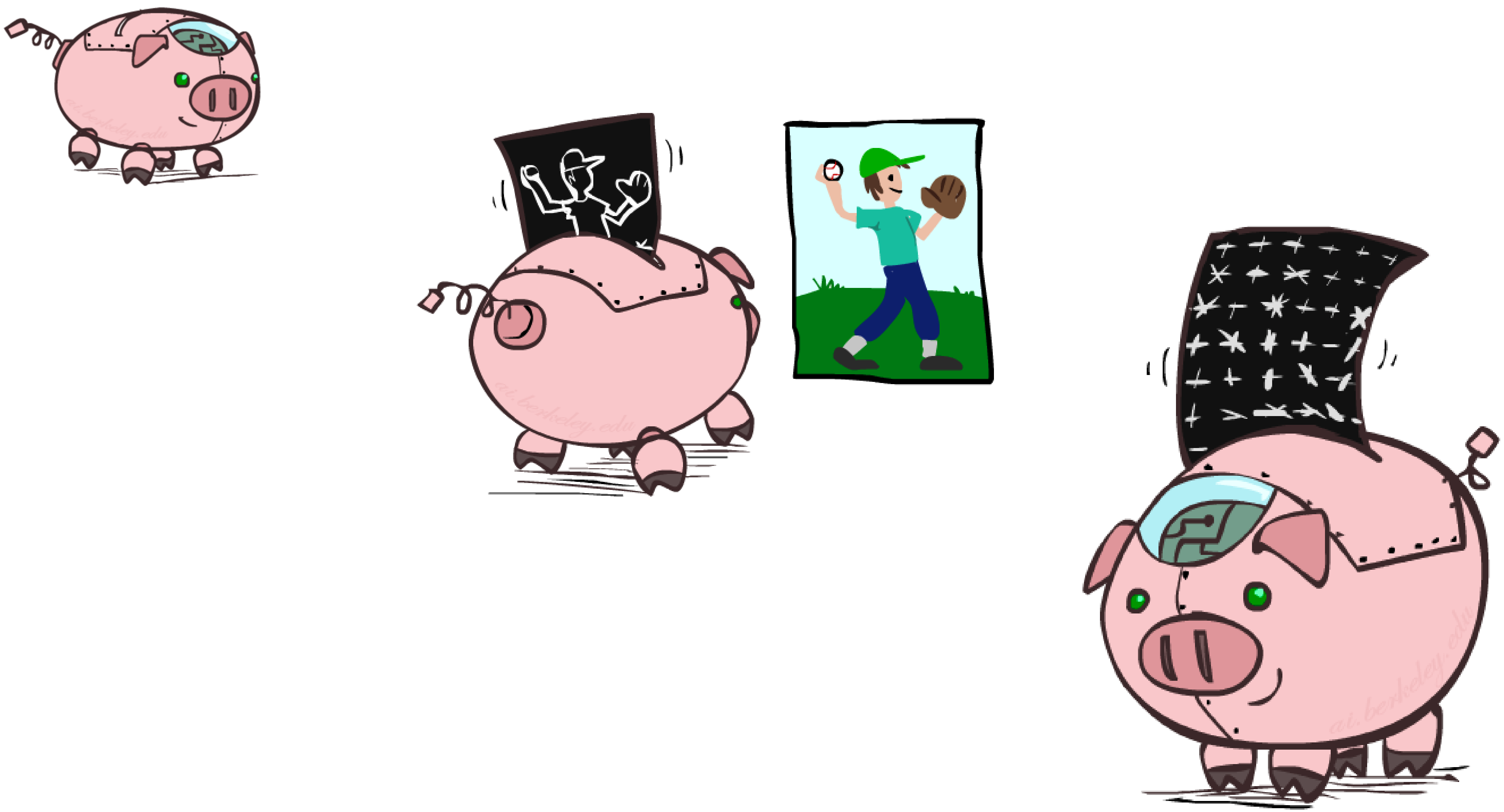
Computer Vision



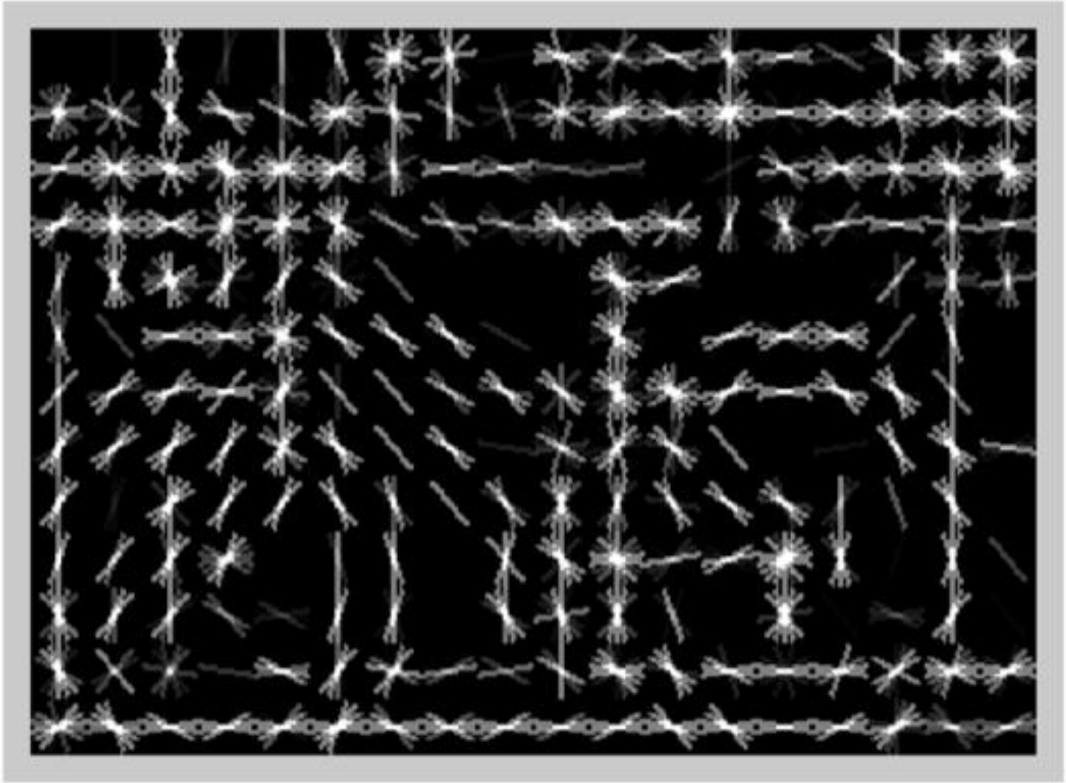
Object Detection



Manual Feature Design



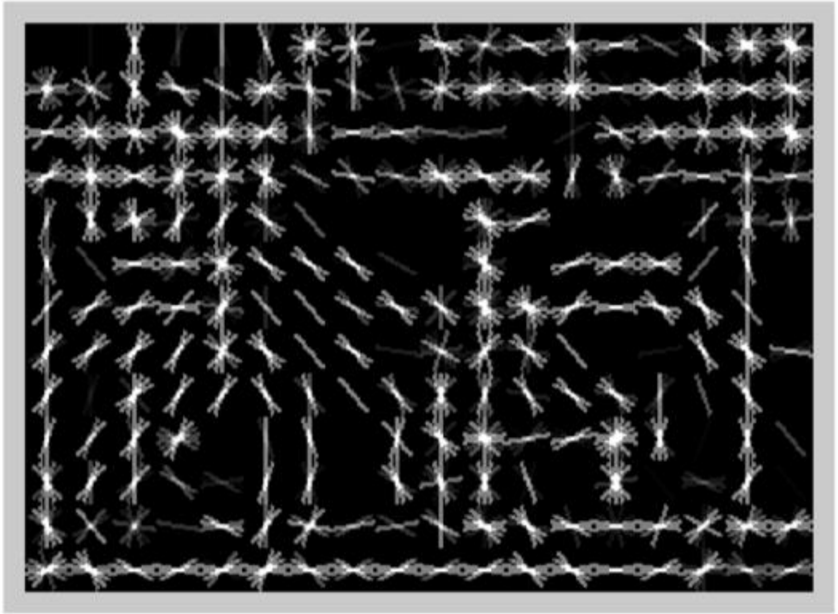
Features and Generalization



[HoG: Dalal and Triggs, 2005]

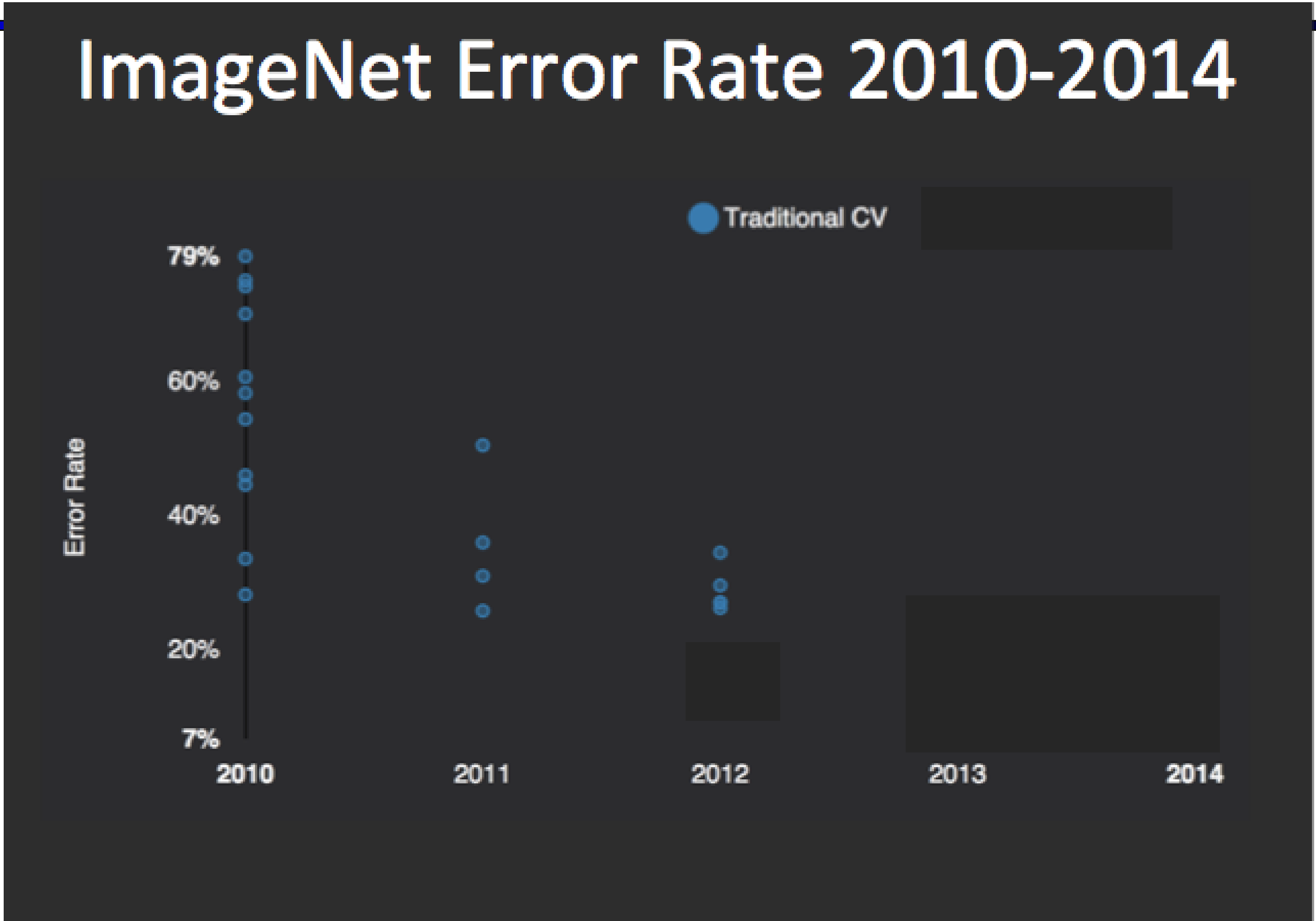
Features and Generalization

Image



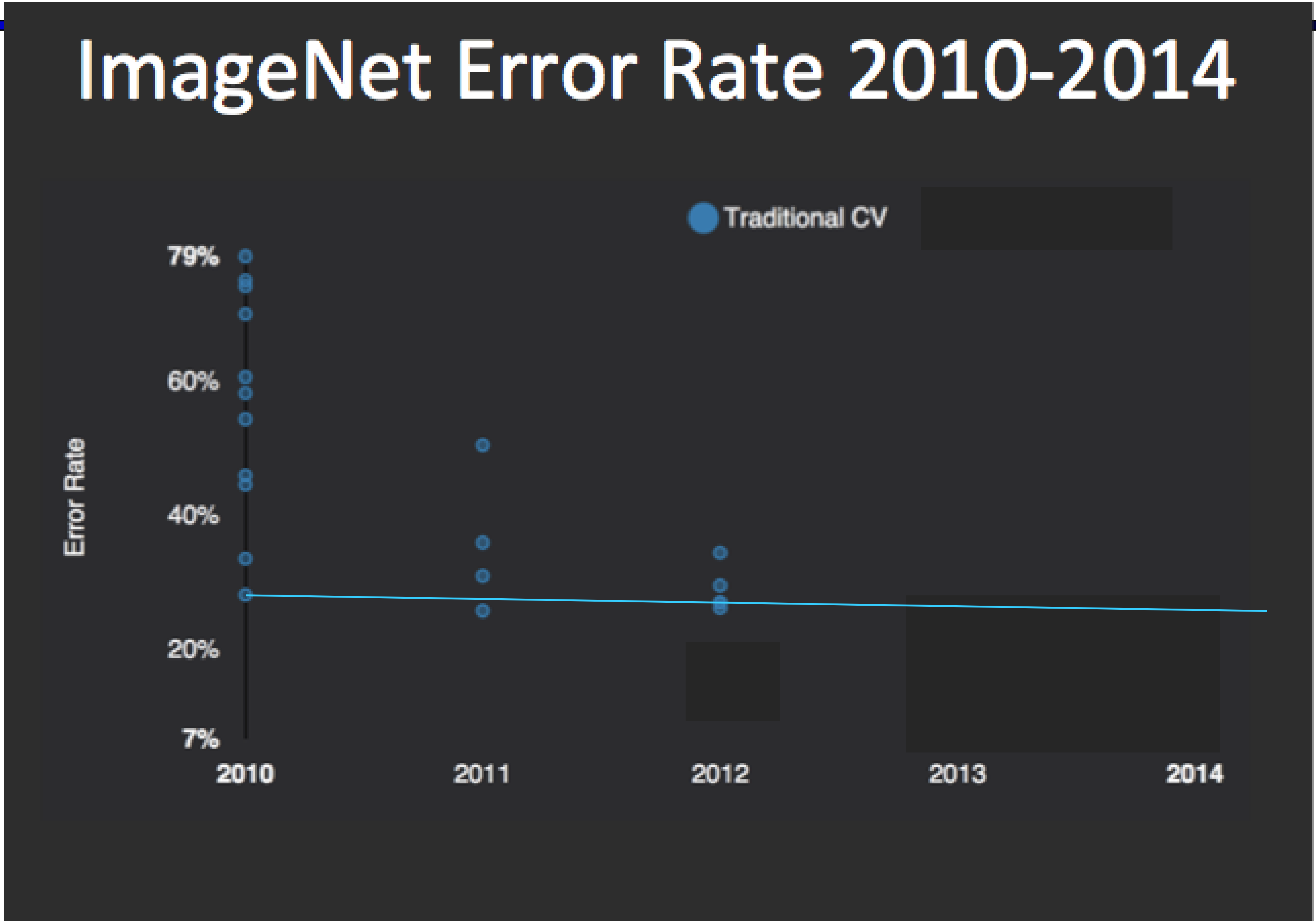
HoG

Performance



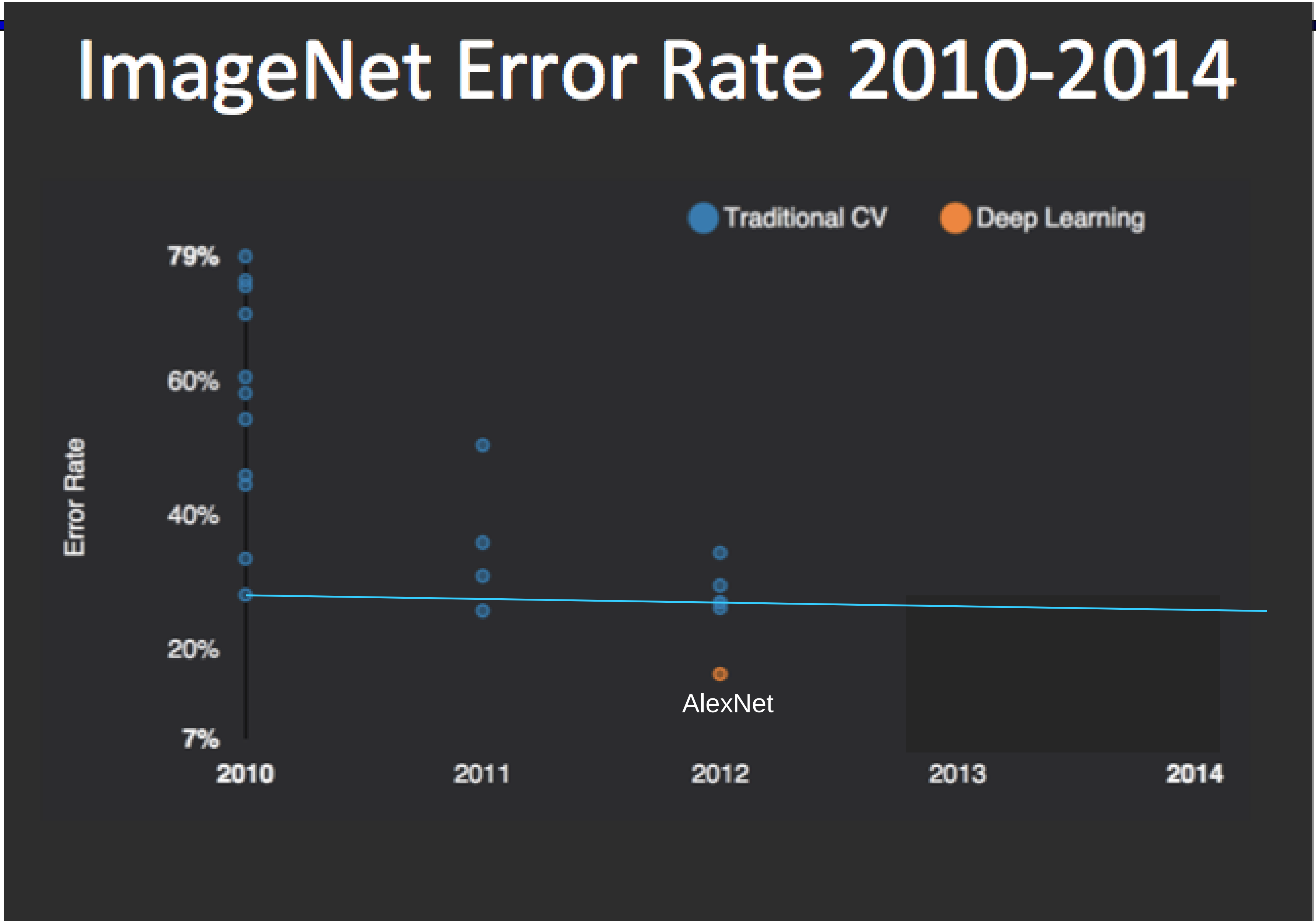
graph credit Matt Zeiler, Clarifai

Performance



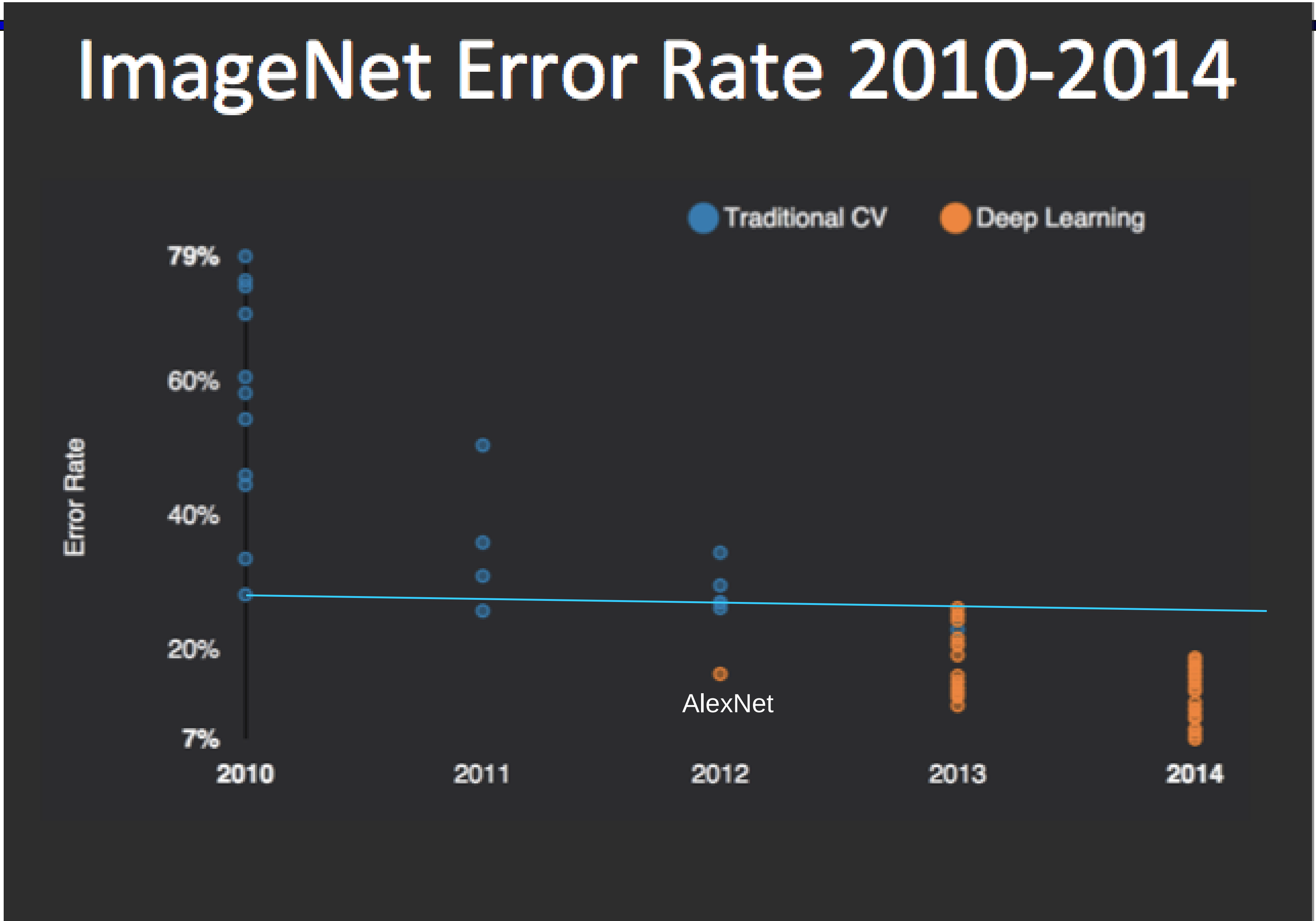
graph credit Matt Zeiler, Clarifai

Performance



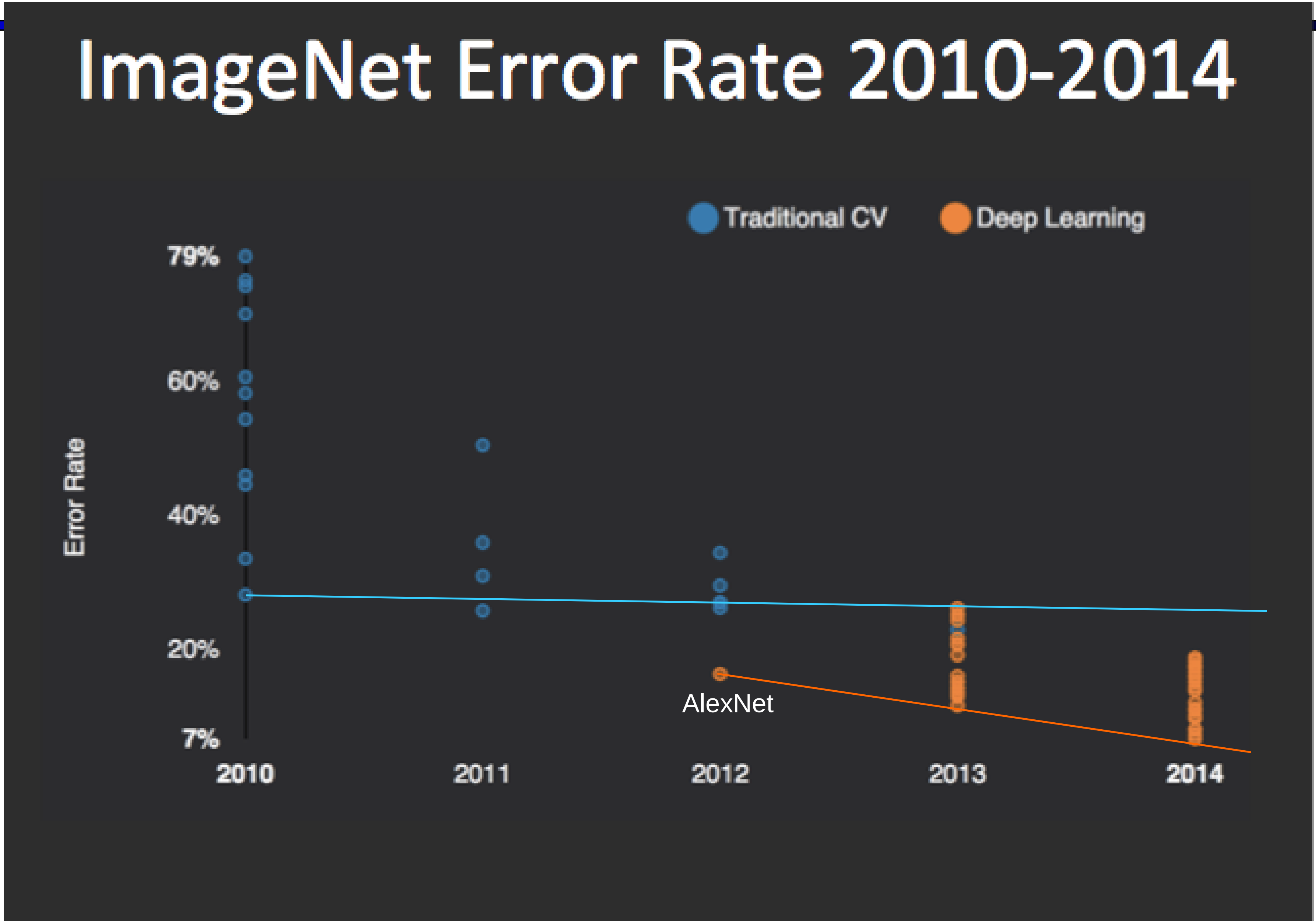
graph credit Matt Zeiler, Clarifai

Performance



graph credit Matt Zeiler, Clarifai

Performance



graph credit Matt Zeiler, Clarifai

MS COCO Image Captioning Challenge

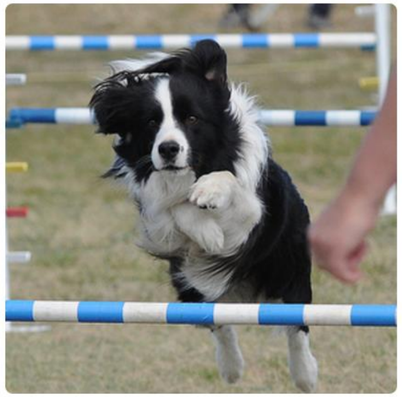
"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."



"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

Karpathy & Fei-Fei, 2015; Donahue et al., 2015; Xu et al, 2015; many more

Visual QA Challenge

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh

What vegetable is on the plate?
Neural Net: broccoli
Ground Truth: broccoli

What color are the shoes on the person's feet ?
Neural Net: brown
Ground Truth: brown

How many school busses are there?
Neural Net: 2
Ground Truth: 2

What sport is this?
Neural Net: baseball
Ground Truth: baseball

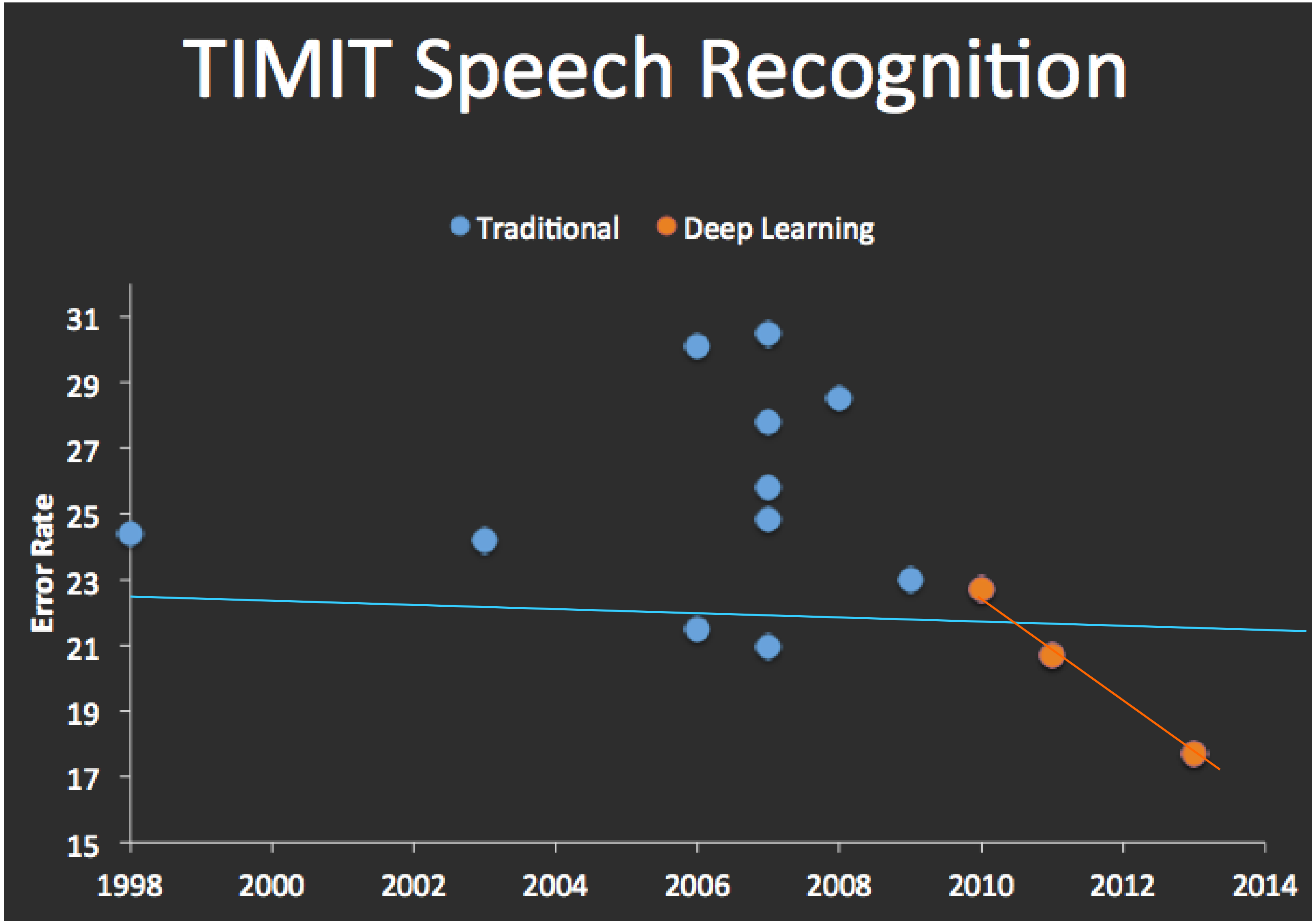
What is on top of the refrigerator?
Neural Net: magnets
Ground Truth: cereal

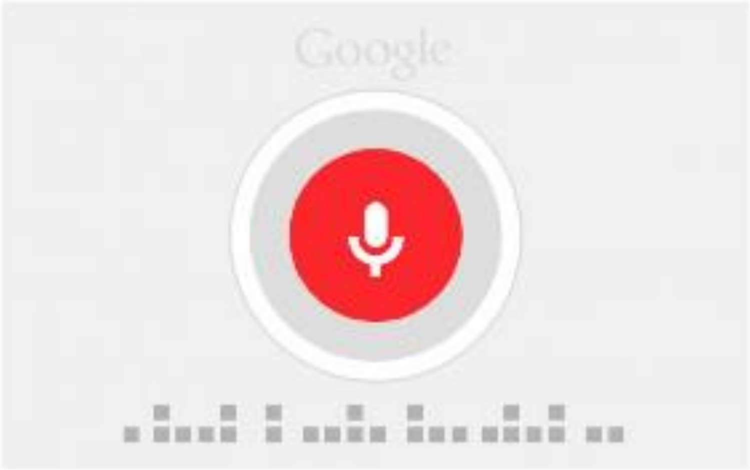
What uniform is she wearing?
Neural Net: shorts
Ground Truth: girl scout

What is the table number?
Neural Net: 4
Ground Truth:40

What are people sitting under in the back?
Neural Net: bench
Ground Truth: tent

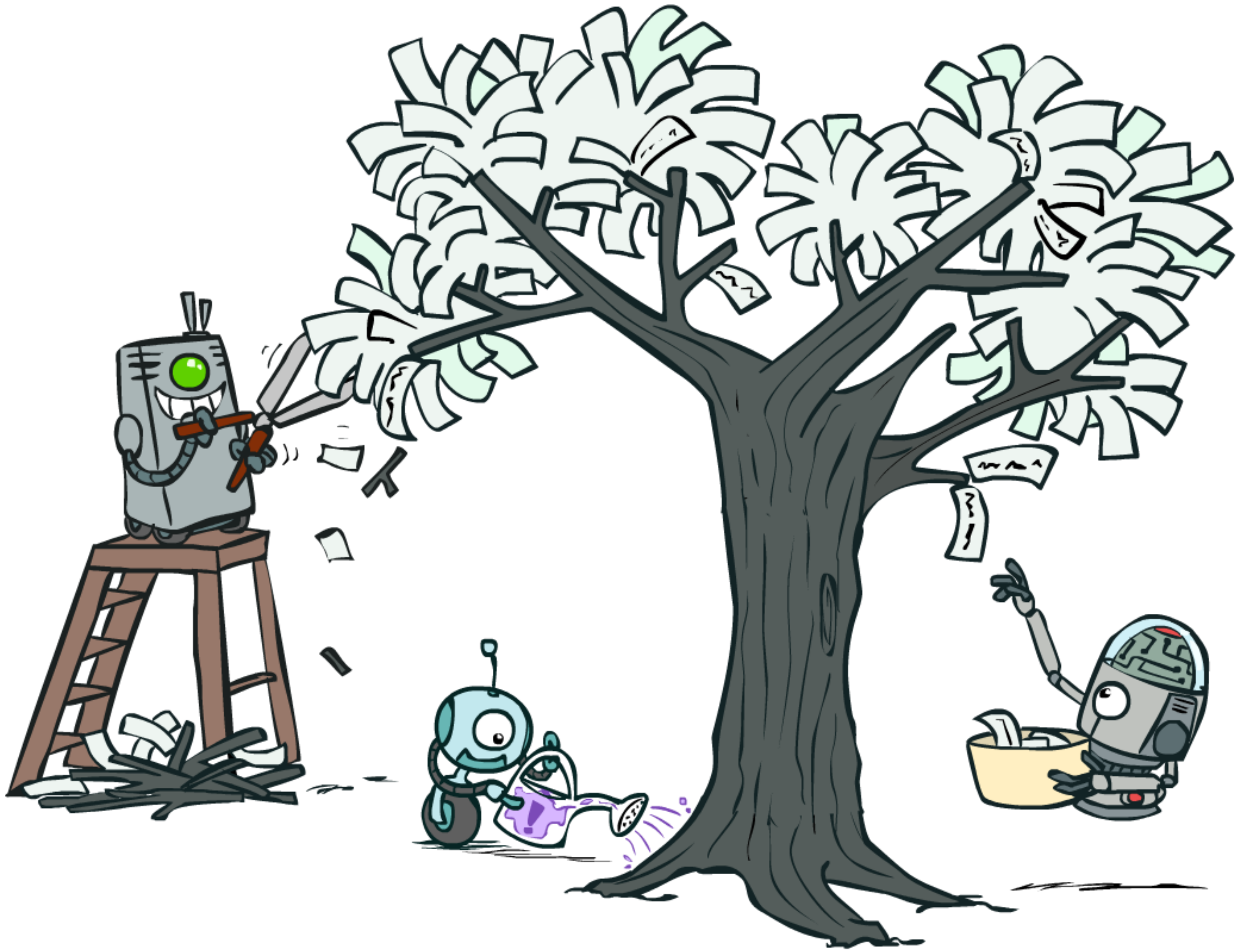
Speech Recognition





graph credit Matt Zeiler, Clarifai

Decision Trees



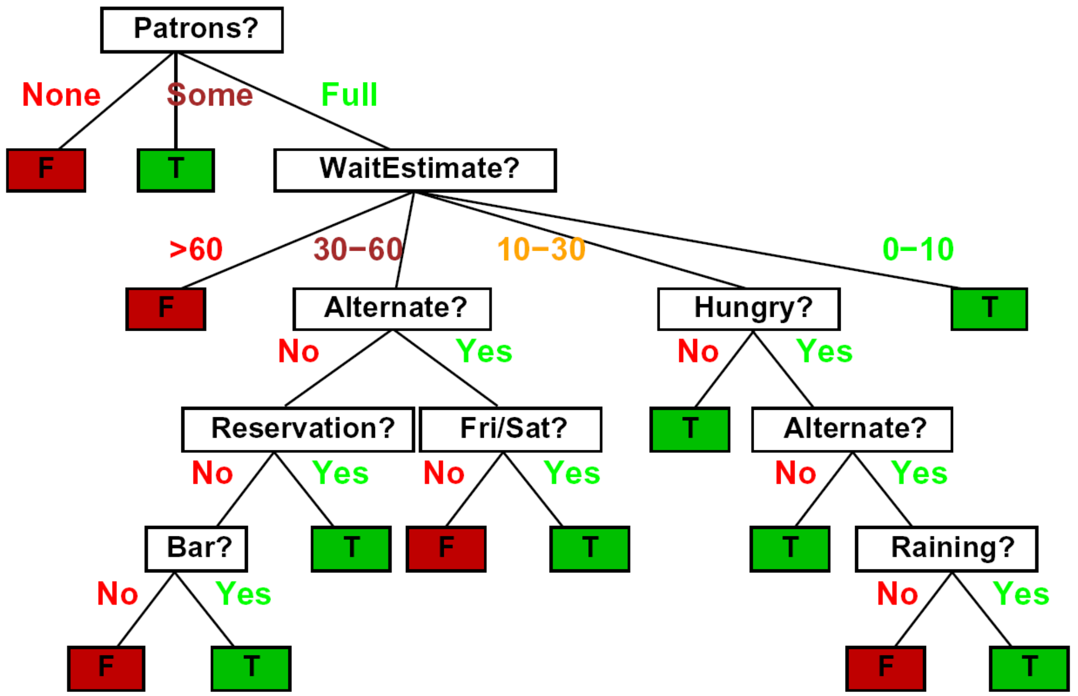
Reminder: Features

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

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

Decision Trees

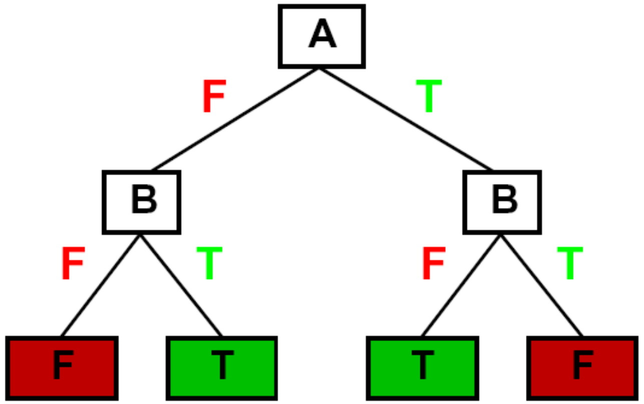
- Compact representation of a function:
  - Truth table
  - Conditional probability table
  - Regression values
- True function
  - Realizable: in H



Expressiveness of DTs

- Can express any function of the features

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



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

- However, we hope for compact trees

Comparison: Perceptrons

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

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*

- 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

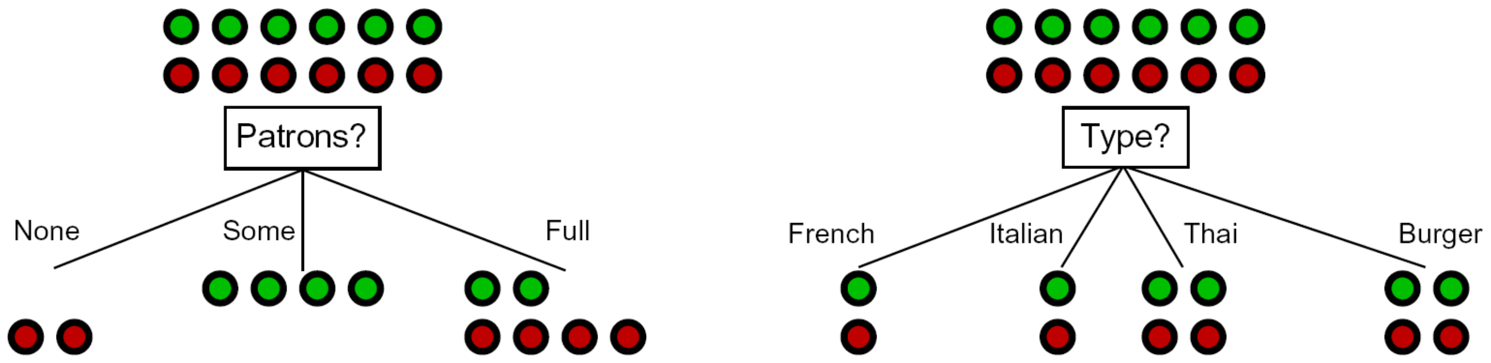
Decision Tree Learning

- 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 <1/2, 1/2>?
    - Answer to 4-way question with prior <1/4, 1/4, 1/4, 1/4>?
    - Answer to 4-way question with prior <0, 0, 0, 1>?
    - Answer to 3-way question with prior <1/2, 1/4, 1/4>?
- A probability p is typical of:
  - A uniform distribution of size 1/p
  - A code of length log 1/p

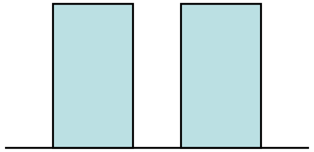
Entropy

- General answer: if prior is <$p_1, \ldots, p_n$>:
  - Information is the expected code length

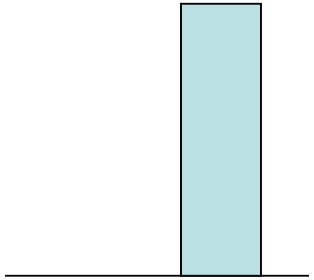
$$H(\langle p_1, \ldots, p_n \rangle) = E_p \log_2 1/p_i$$

$$= \sum_{i=1}^{n} -p_i \log_2 p_i$$

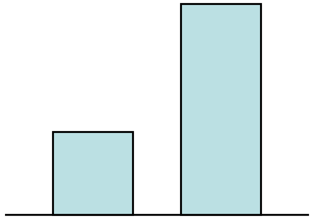
- Also called the entropy of the distribution
  - More uniform = higher entropy
  - More values = higher entropy
  - More peaked = lower entropy



1 bit



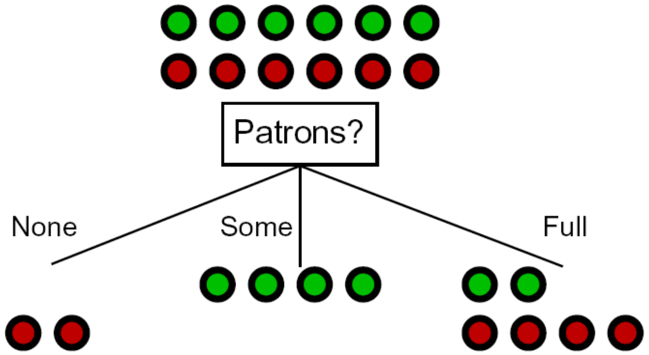
0 bits

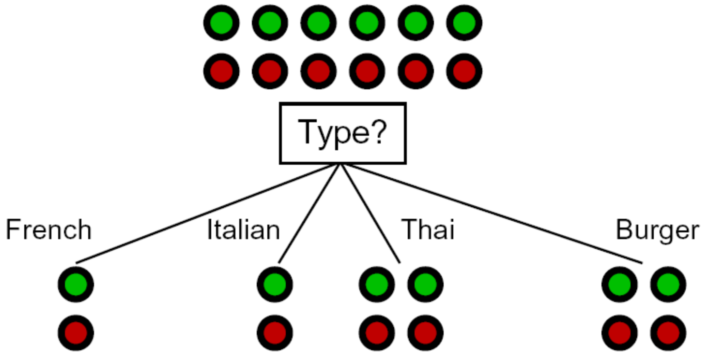


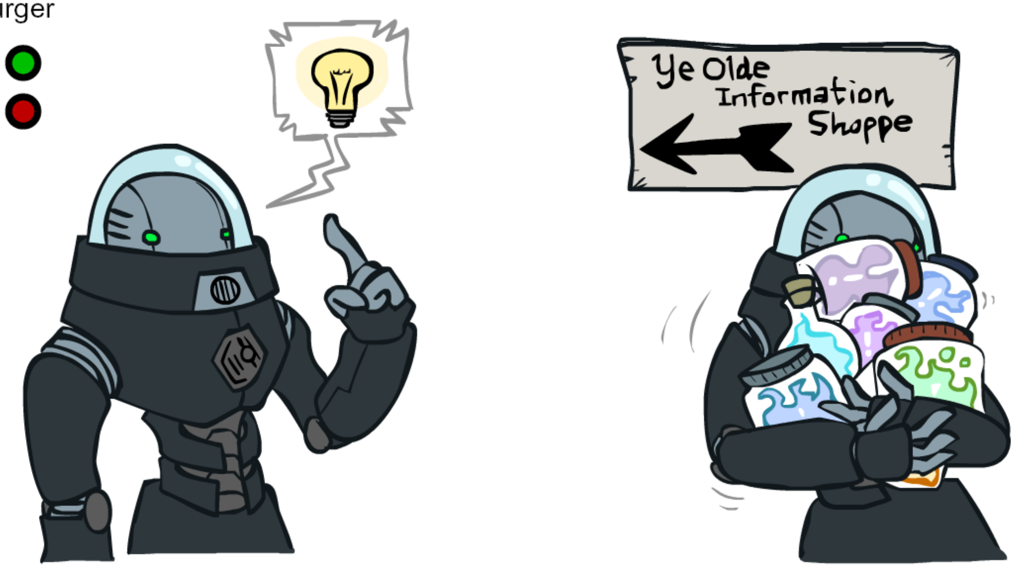
0.5 bit

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!



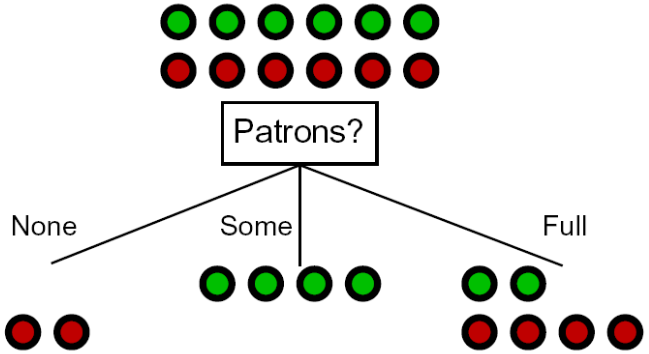




  - Solution: use expected entropy, weighted by the number of examples

Next Step: Recurse

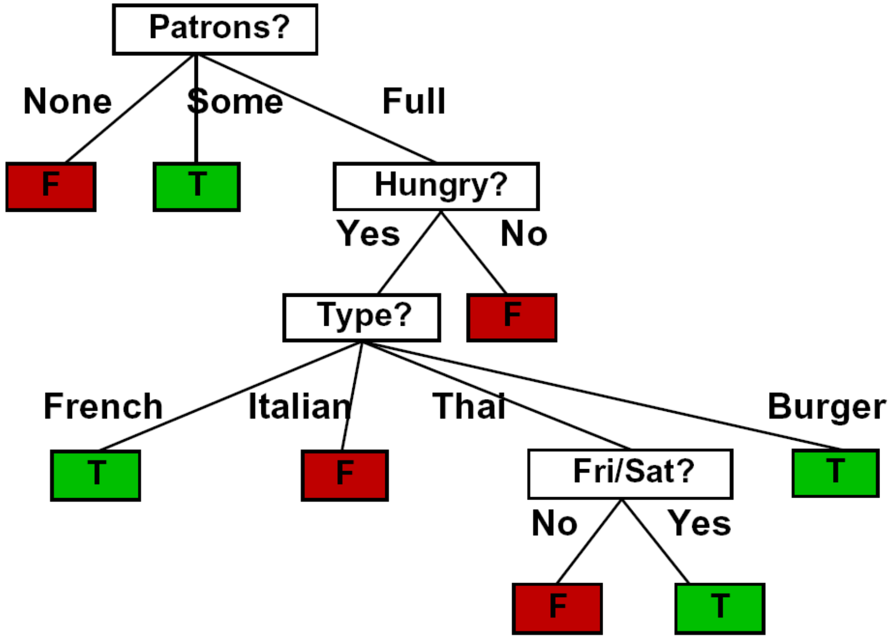
- Now we need to keep growing the tree!
- Two branches are done (why?)
- What to do under "full"?
  - See what examples are there…



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

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

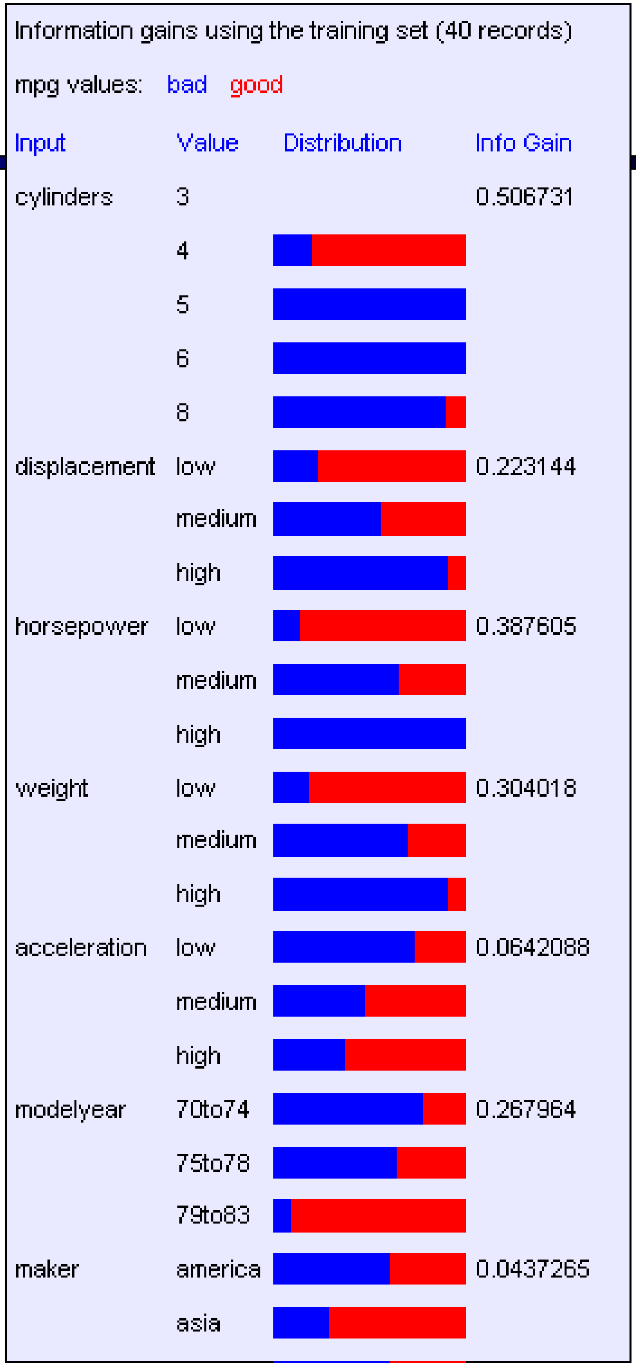
Example: Miles Per Gallon

40 Examples

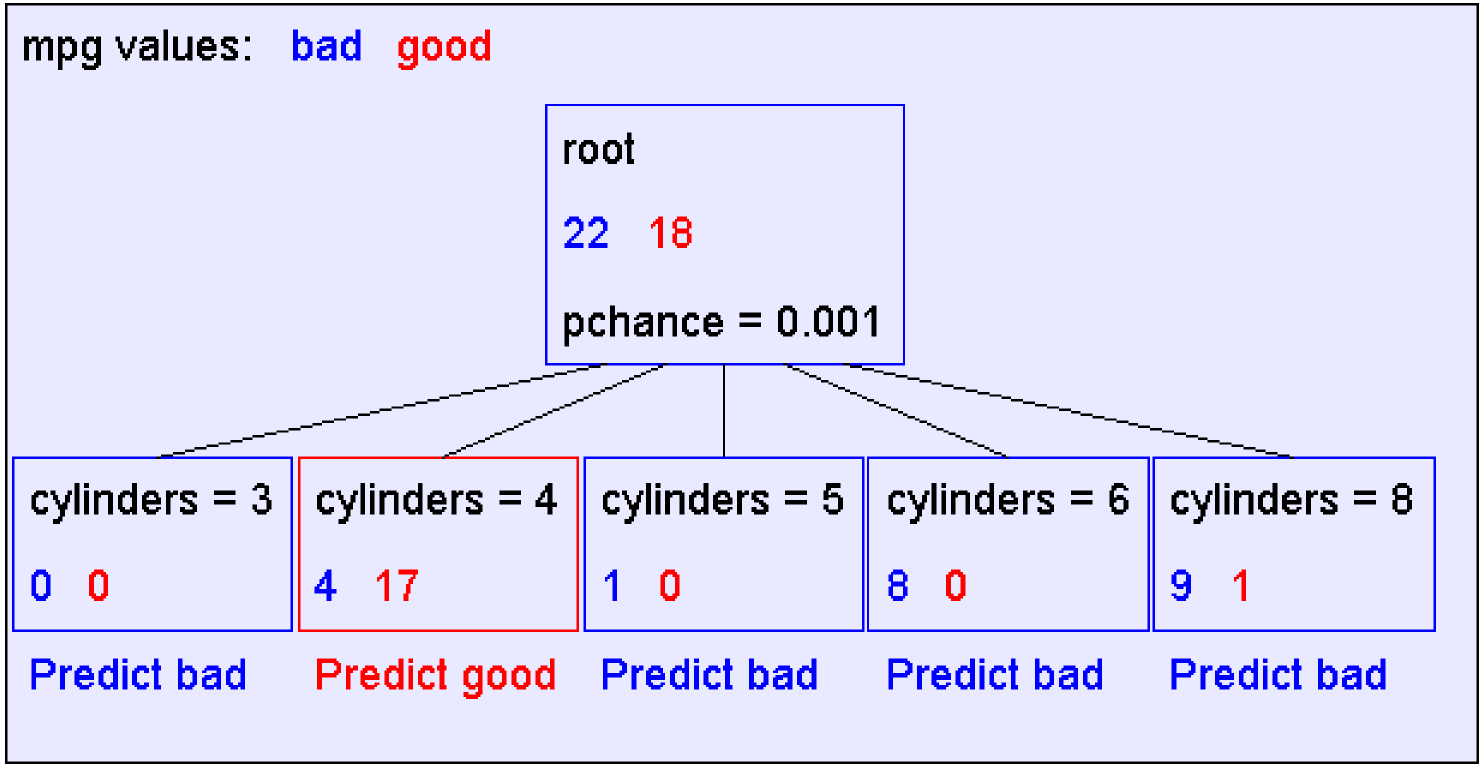
mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europe
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low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

Find the First Split

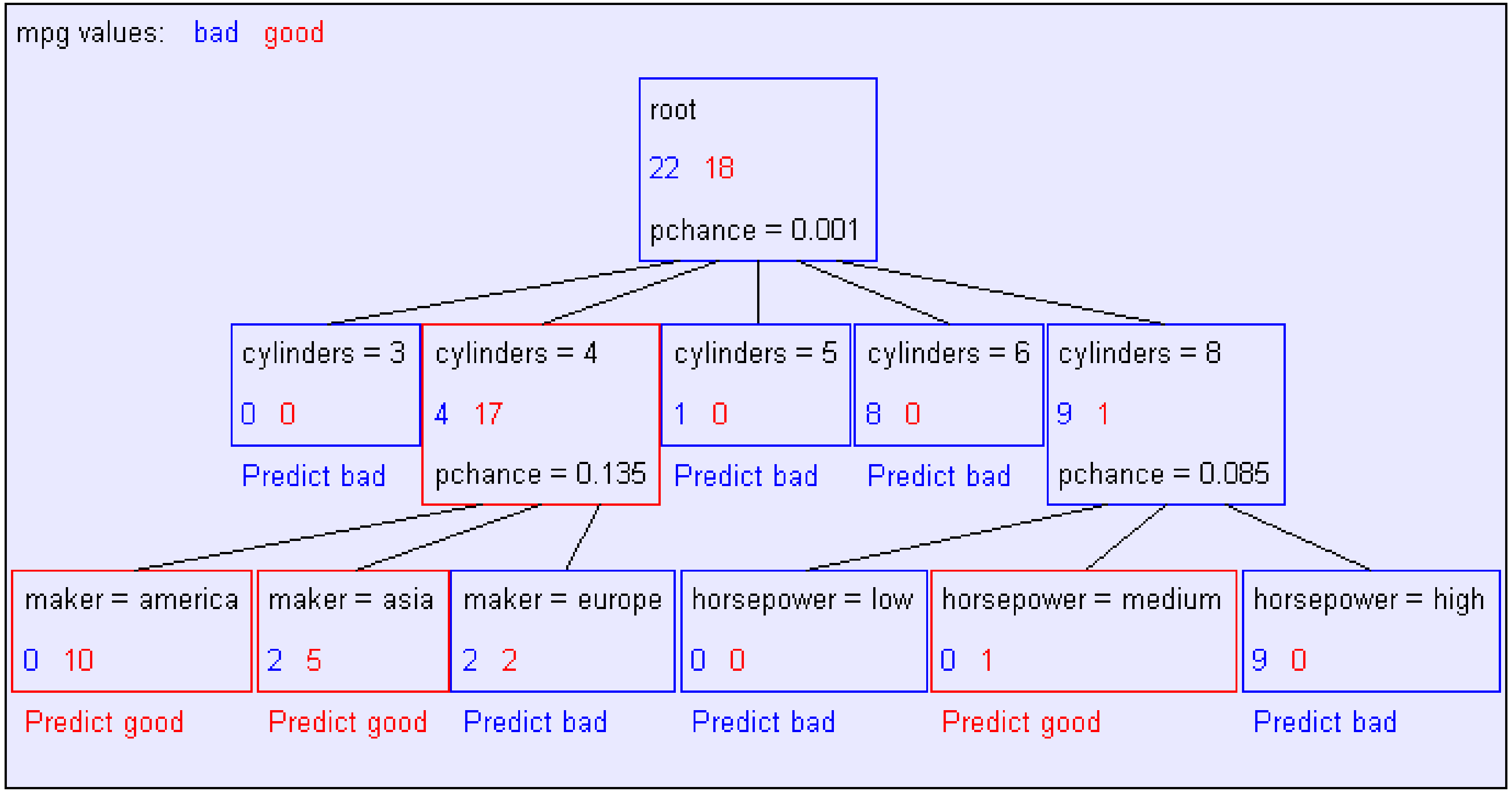
- Look at information gain for each attribute
- Note that each attribute is correlated with the target!
- What do we split on?



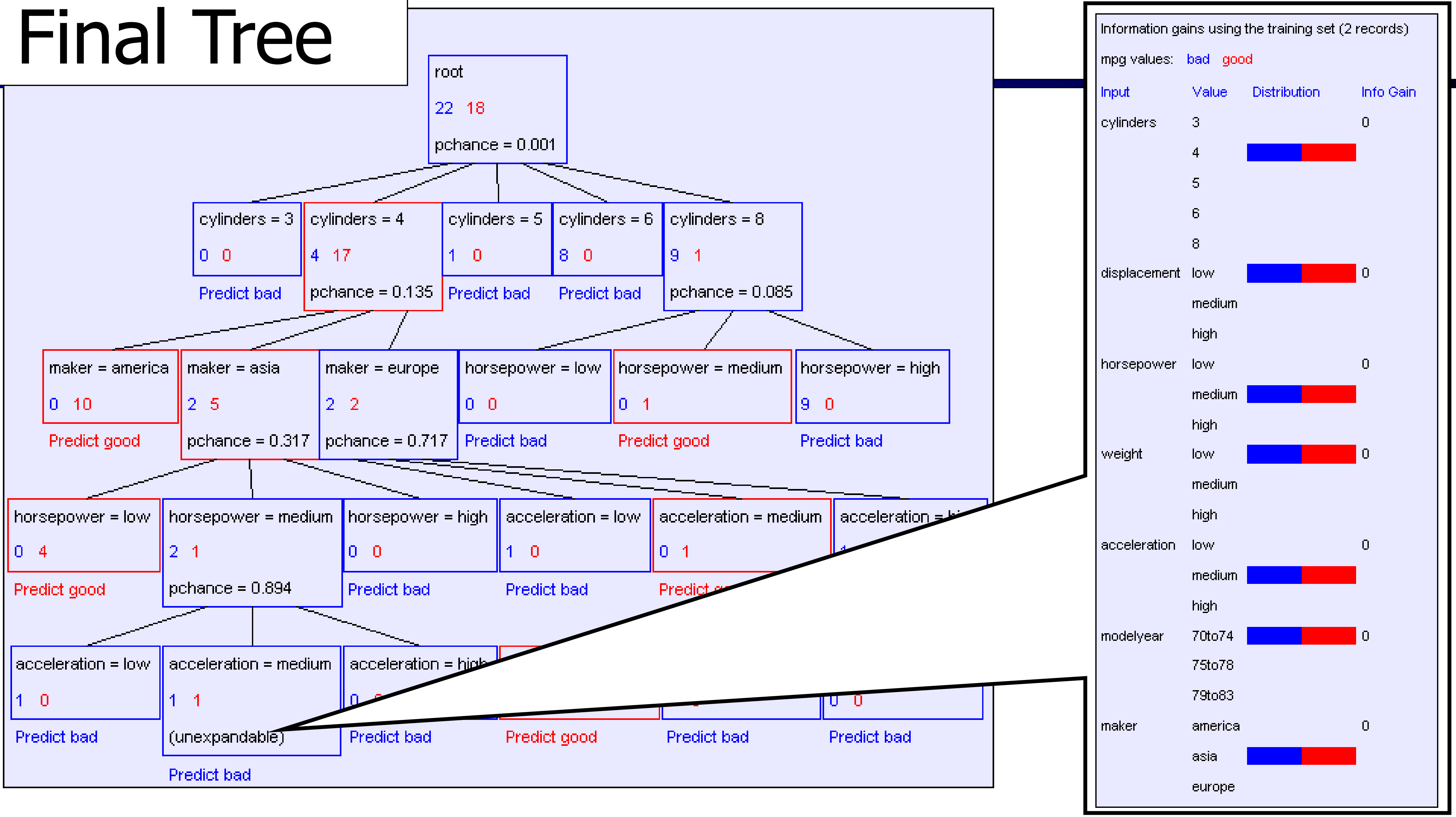
Result: Decision Stump



Second Level



Final Tree



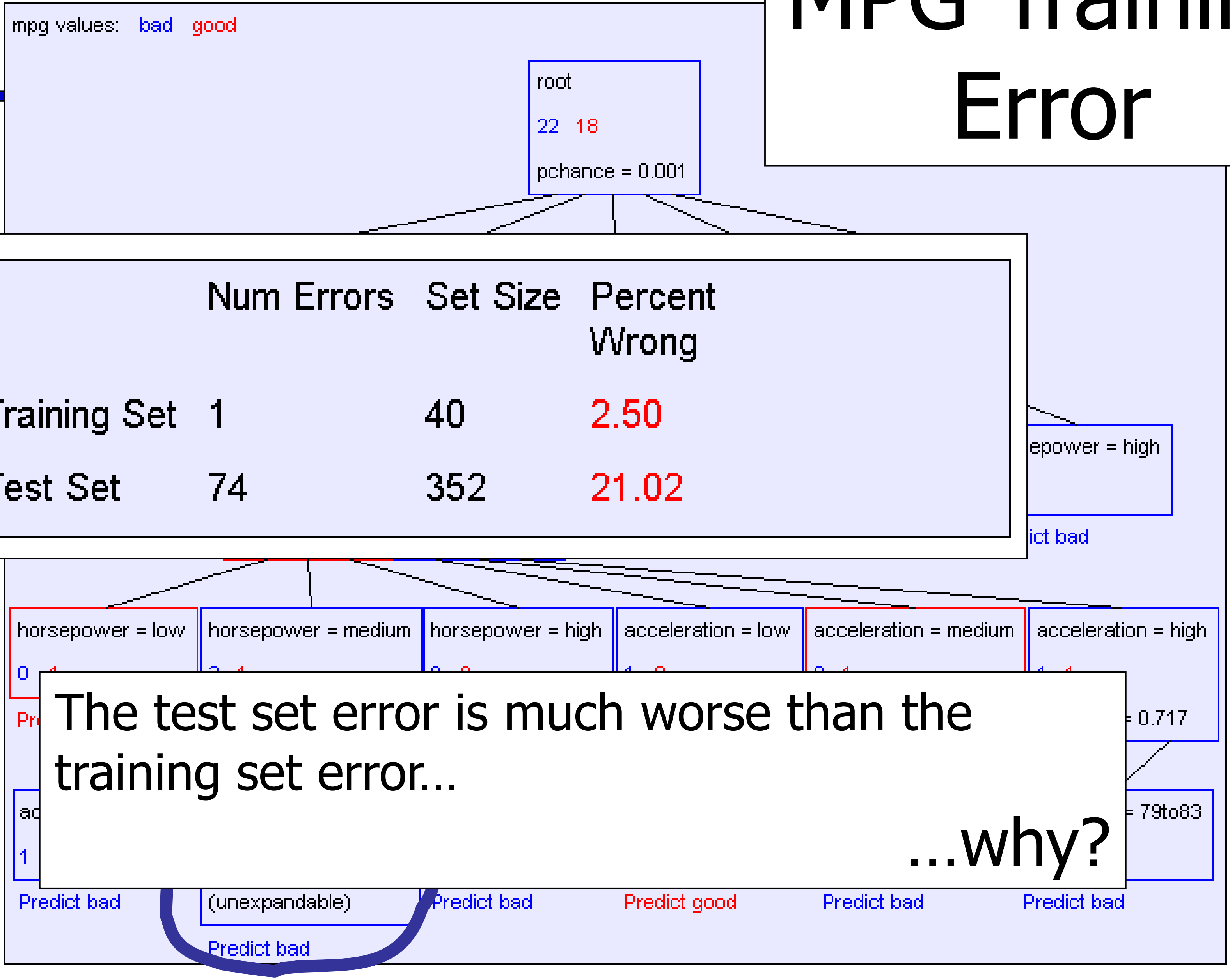
root
22 18
pchance = 0.001
cylinders = 3
0 0
Predict bad
cylinders = 4
4 17
pchance = 0.135
cylinders = 5
1 0
Predict bad
cylinders = 6
8 0
Predict bad
cylinders = 8
9 1
pchance = 0.085
maker = america
0 10
Predict good
maker = asia
2 5
pchance = 0.317
maker = europe
2 2
pchance = 0.717
horsepower = low
0 0
Predict bad
horsepower = medium
0 1
Predict good
horsepower = high
9 0
Predict bad
horsepower = low
0 4
Predict good
horsepower = medium
2 1
pchance = 0.894
horsepower = high
0 0
Predict bad
acceleration = low
1 0
Predict bad
acceleration = medium
0 1
acceleration = low
1 0
Predict bad
acceleration = medium
1 1
(unexpandable)
Predict bad
Predict bad
Predict good
Predict bad
Predict bad
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



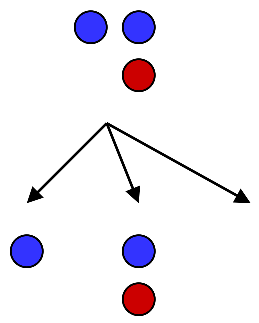
	Num Errors	Set Size	Percent Wrong
Training Set	1	40	2.50
Test Set	74	352	21.02

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_{CHANCE}

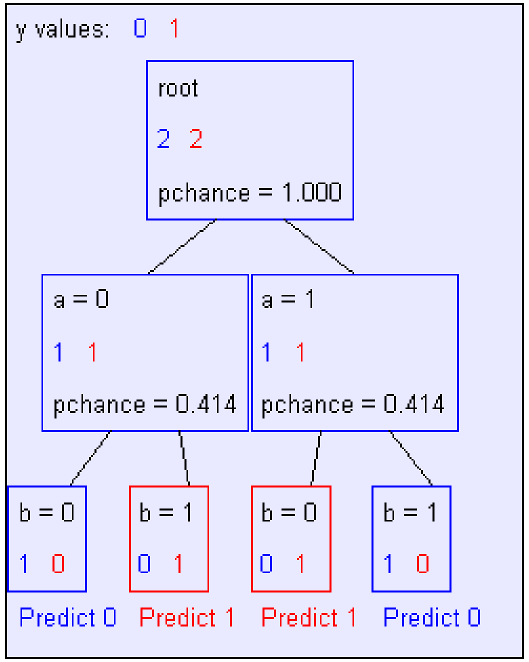
Keeping it General

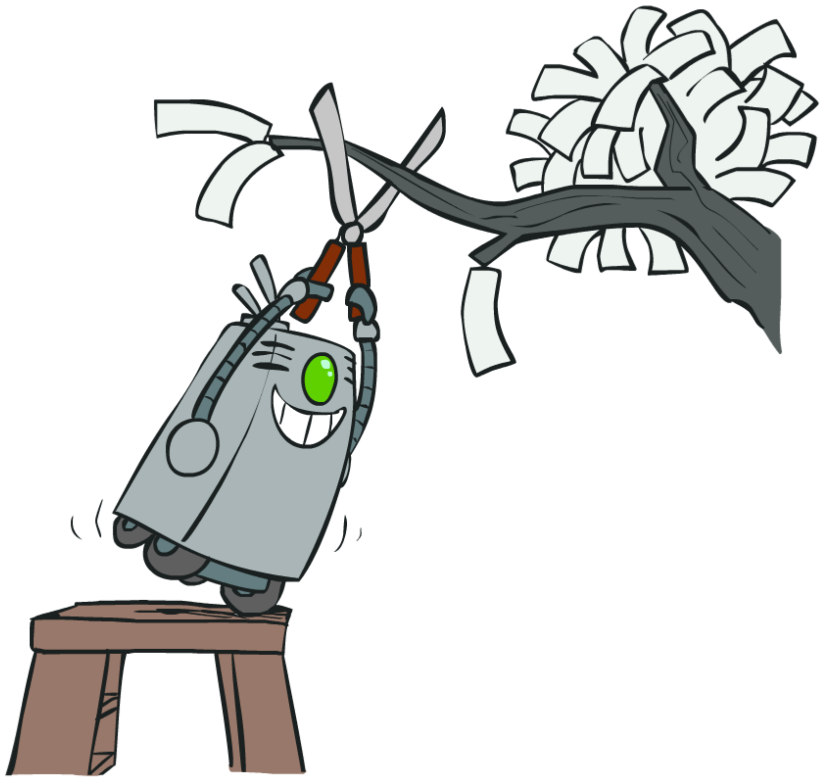
- Pruning:
  - Build the full decision tree
  - Begin at the bottom of the tree
  - Delete splits in which

    $p_{CHANCE} > MaxP_{CHANCE}$
  - Continue working upward until there are no more prunable nodes

y = a XOR b

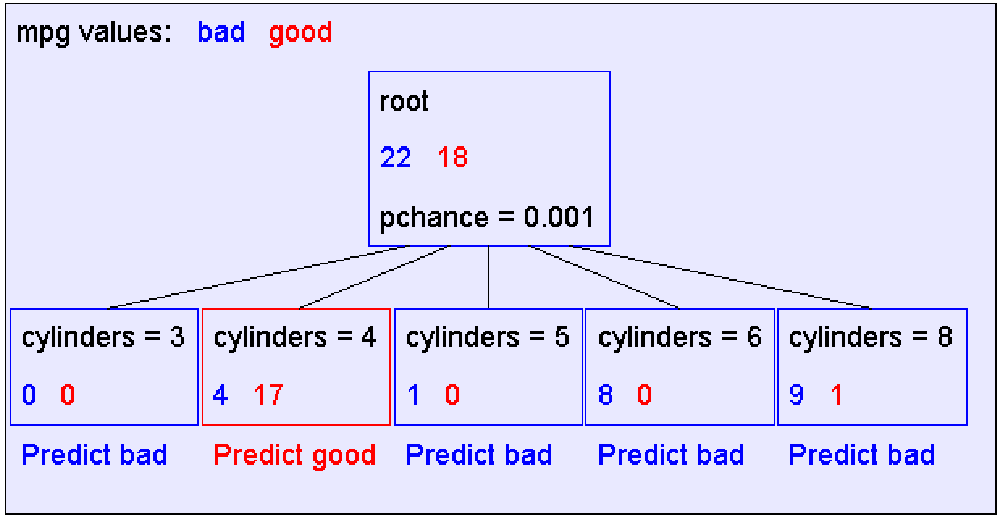
a	b	y
0	0	0
0	1	1
1	0	1
1	1	0





Pruning example

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

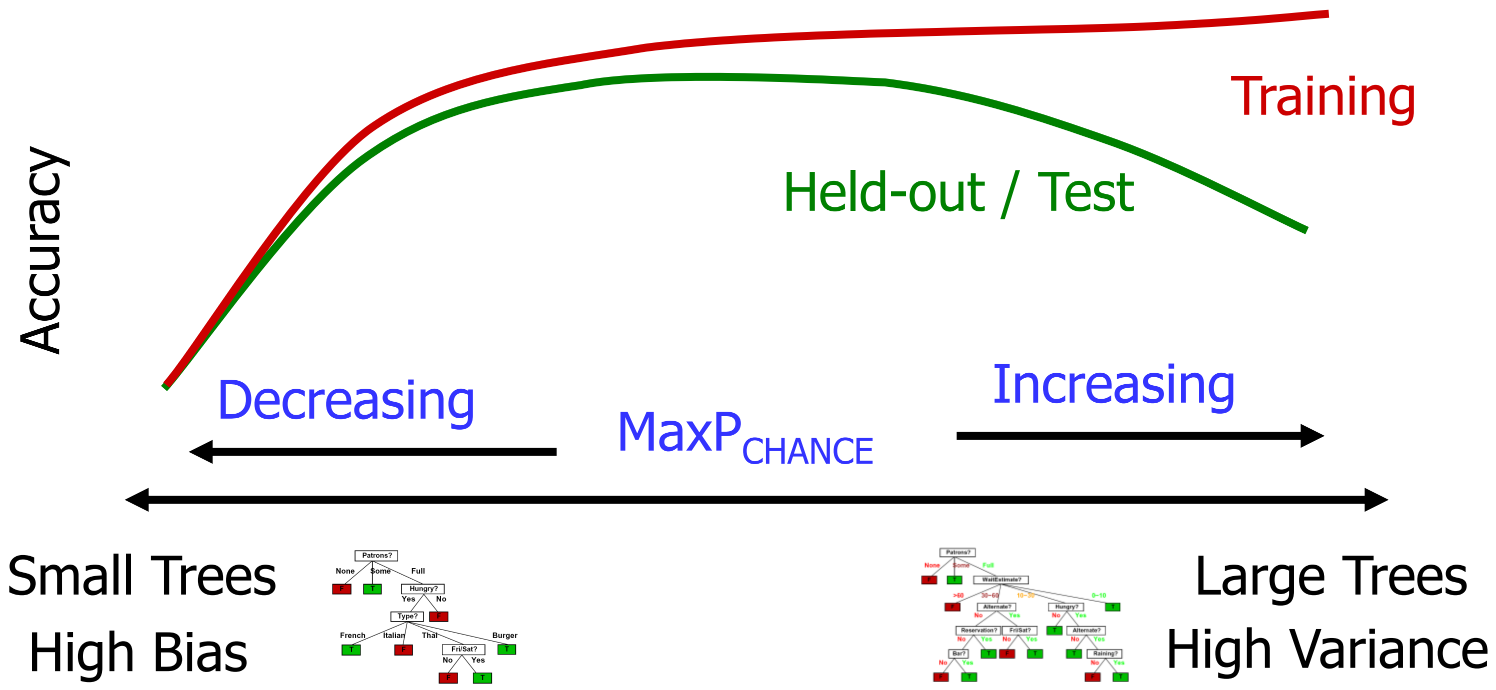


Note the improved test set accuracy compared with the unpruned tree

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

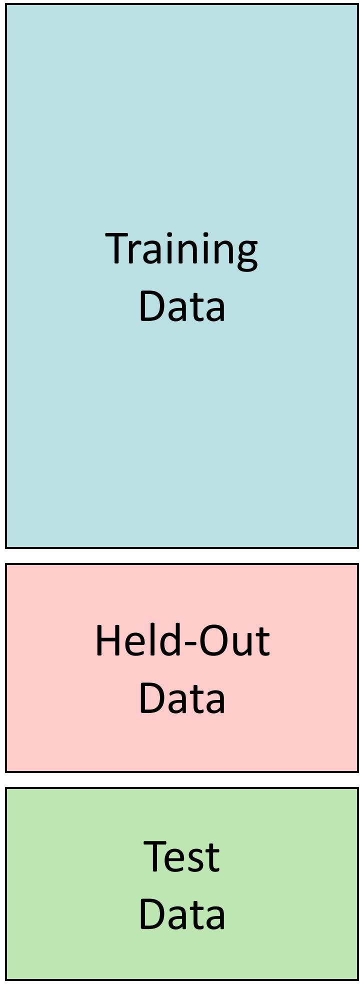
Regularization

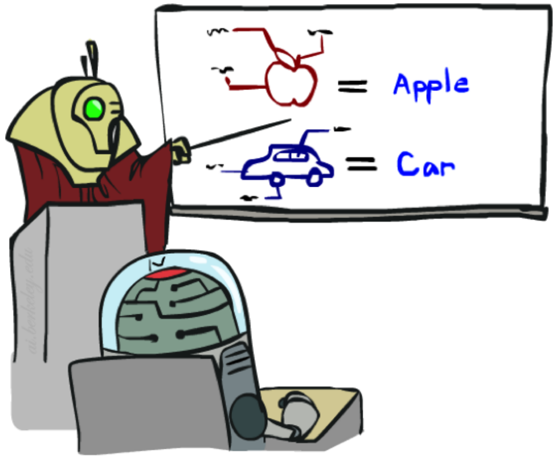
- $\text{MaxP}_{\text{CHANCE}}$ is a regularization parameter
- Generally, set it using held-out data (as usual)

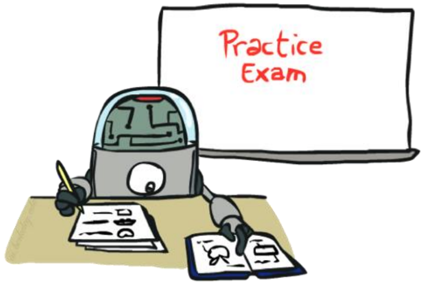


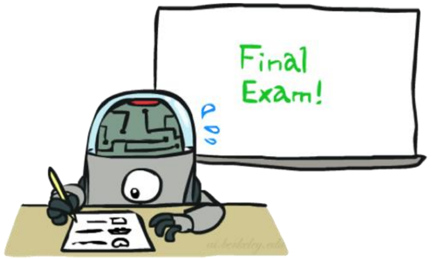
A few important points about learning

- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Features: attribute-value pairs which characterize each x
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - (Tune hyperparameters on held-out set)
  - Compute accuracy of test set
  - Very important: never “peek” at the test set!
- Evaluation
  - Accuracy: fraction of instances predicted correctly
- Overfitting and generalization
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well
  - Underfitting: fits the training set poorly





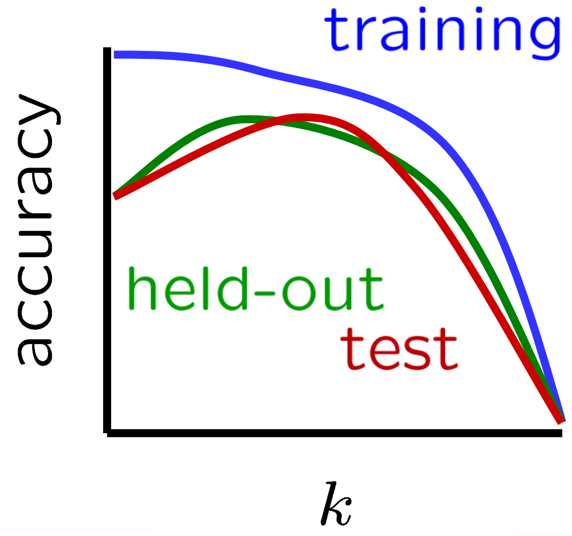


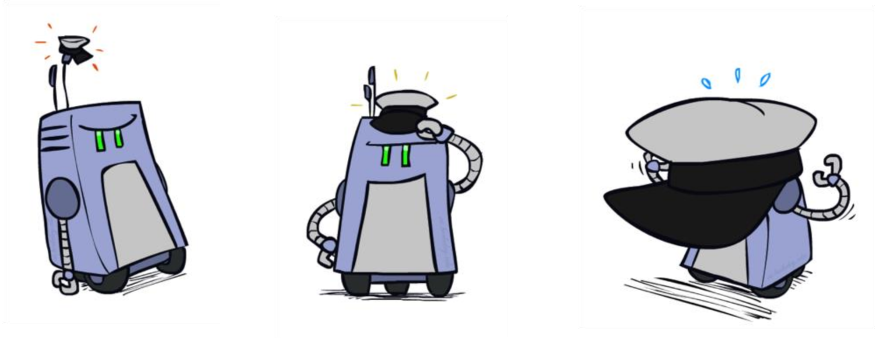


A few important points about learning

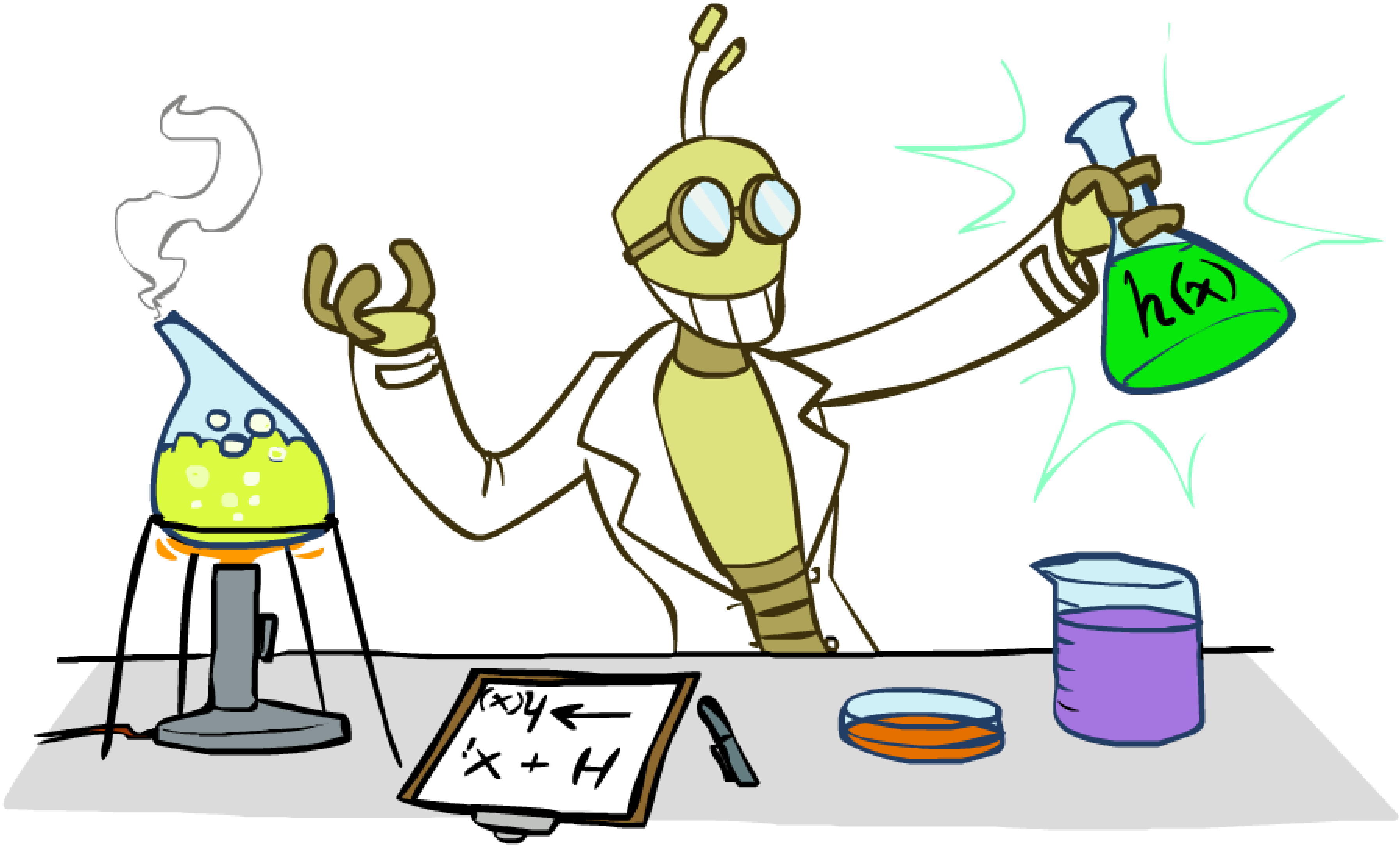
- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data
    - Why?
  - For each value of the hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data

- What are examples of hyperparameters?



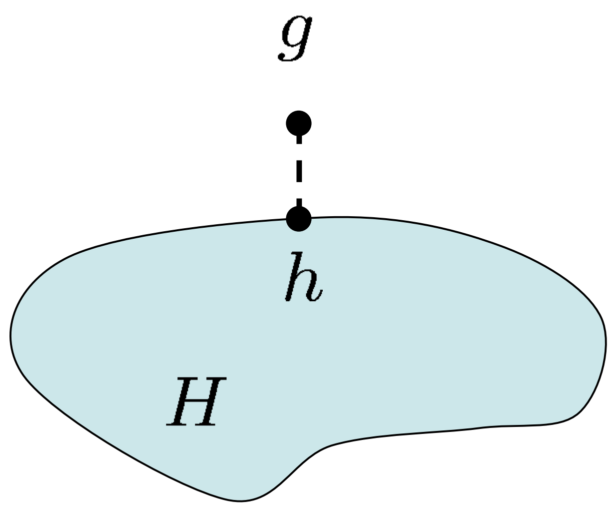


Inductive Learning



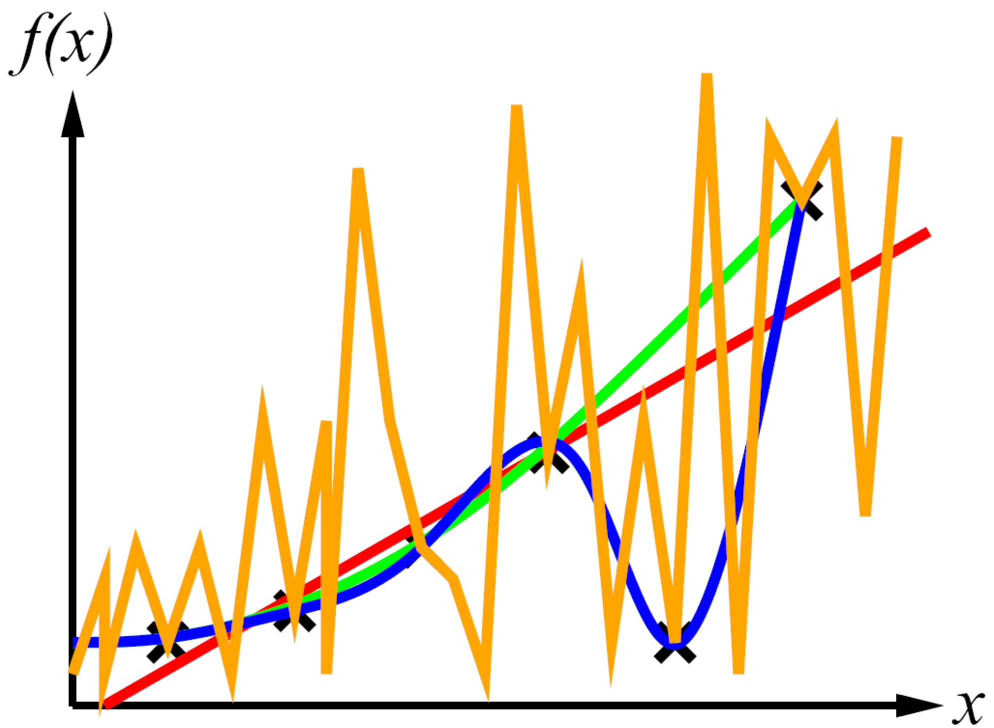
Inductive Learning (Science)

- 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