COMS W4701: Artificial Intelligence

Lecture 12: Intro to Machine Learning

Tony Dear, Ph.D.

Department of Computer Science

School of Engineering and Applied Sciences

Today

- Decision trees
- Entropy and information gain

- Linear classifiers and perceptrons

- Neural networks

Review: Supervised Learning

- Many ML tasks involve **classification** or **regression** problems
- Classification: Given data input, predict an output label or class (discrete)
- Regression: Given data input, predict an output *value* (continuous)

- **Supervised learning**: Given *training data* with input-output pairs $(x_1, y_1), \dots, (x_N, y_N)$ generated by an unknown function f, find a *hypothesis* to best approximate f

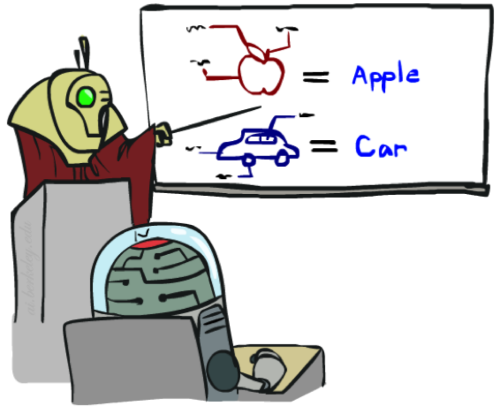
- Naïve Bayes solved this using a specific probabilistic model

- We can use other (non-probabilistic) models as well…

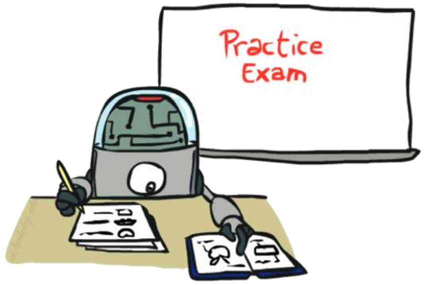
Learning a General Model

- Acquire labeled (classes and features) data and split into three categories
- Build a model that minimizes an error rate or maximizes a likelihood on **training data**
- Regularize model (tune *hyperparameters*) by maximizing performance on **validation data**
- Evaluate on **test data**: use different accuracy metrics to qualify performance

Training Data



Validation Data



Test Data

Decision Tree Representation

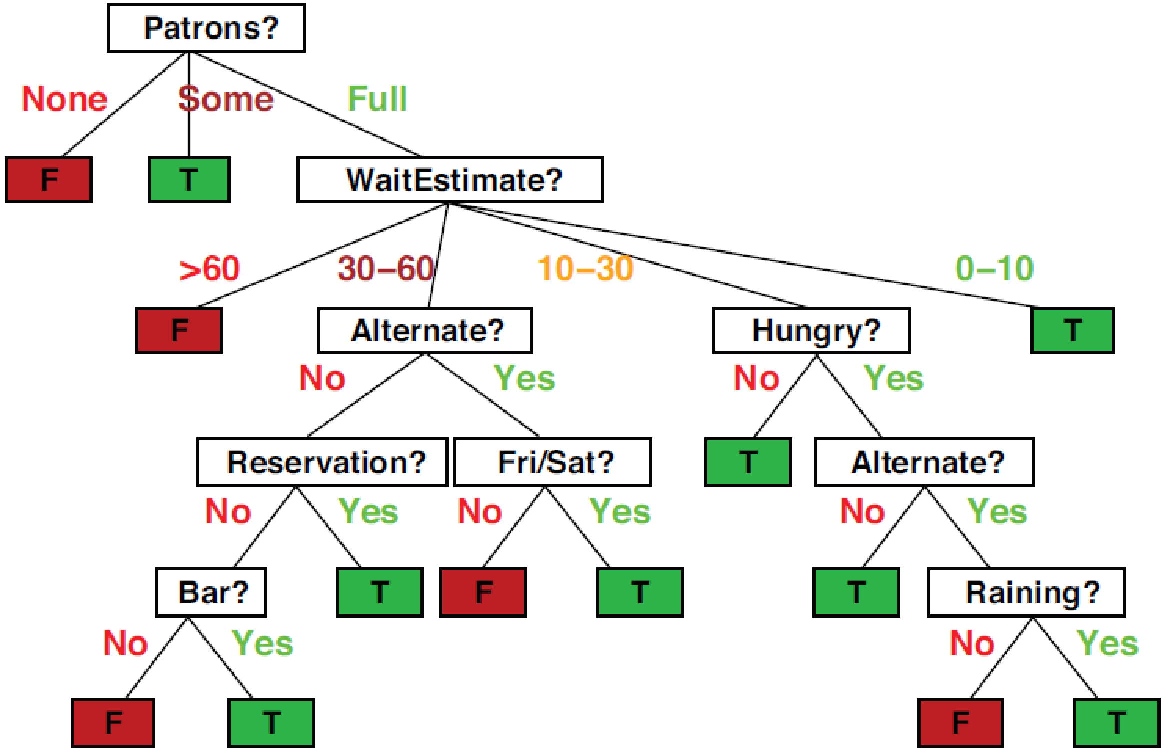
- Classification is an *episodic decision-making* problem
- Make decisions given fixed information, instead of over time/space

- Suppose information is broken down by features
- Different feature values or combinations of values lead to a decision

- The decision function has a tree structure over the features
- "If f1=a then check if f2=x, elif f1=b then check if f3=y, elif…"

Example: Restaurant Wait Decision

- Input features: Attributes of restaurant, time/day, personal emotions
- Output decision: To wait or not to wait?
- Nodes: Test of feature values
- Branches: Possible feature values
- Leaves: Decision value of function



Learning Decision Trees

- Decision tree classification is linear in number of features
- How do we learn one in the first place?

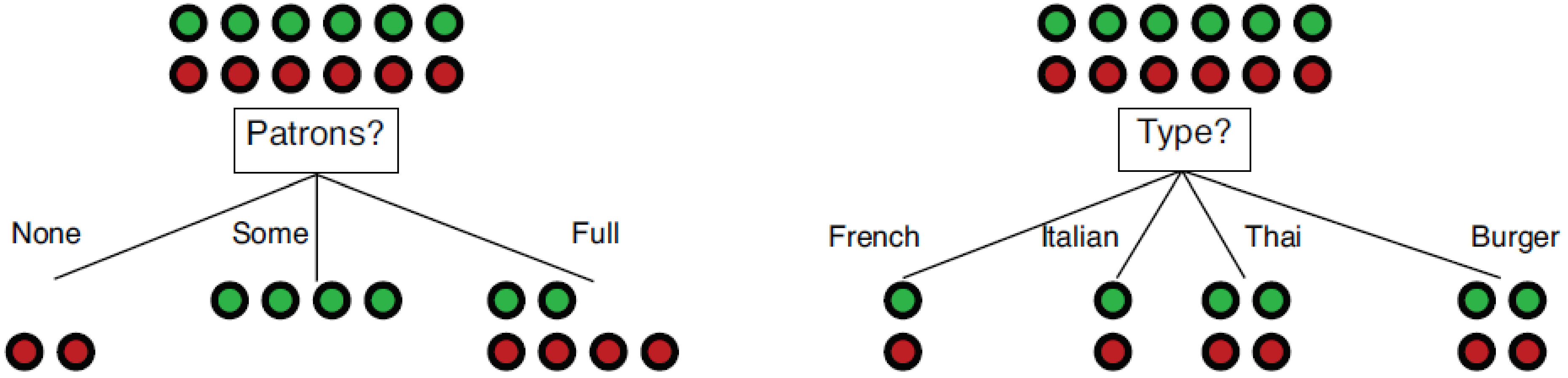
- We generally want a shallower tree with fewer tests and shorter paths
- Make decisions by checking as few features as possible

- Idea: Recursively identify "most significant" attribute in training data and build tree from the root
- Each feature splits to a separate path down the tree

Choosing Attributes

- A shallow decision tree should drive toward a decision as quickly as possible
- The "purer" the remaining classes of training examples in a tree leaf, the more informative the parent attribute

- Which attribute below gives us more information?



Entropy

- **Entropy** measures uncertainty: the "closeness" of a dataset to a uniform split
- Let d_c be proportion of dataset D that belong to class c

$$H(D) = -\sum_{c \in C} d_c \log_2(d_c)$$

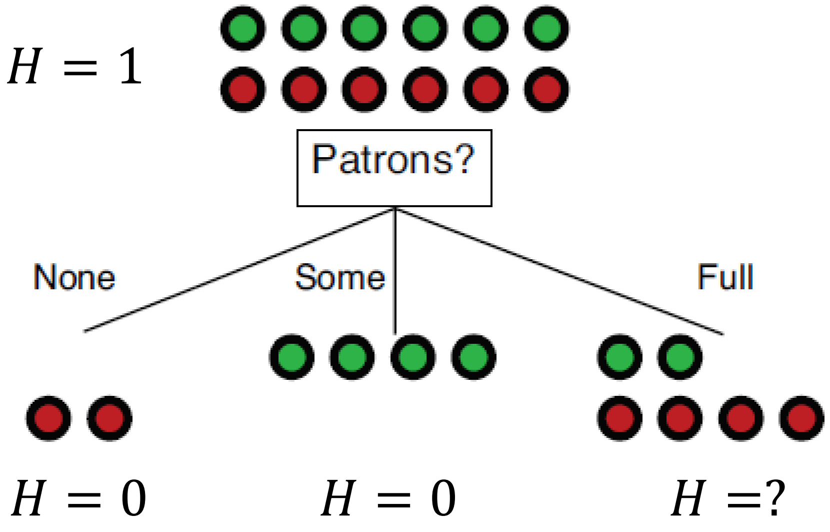
- If D contains only one class c, then $d_c = 1$ and $H(D) = -\log_2(1) = 0$
- If D contains n data, k classes, and $\frac{n}{k}$ data per class, then $d_c = \frac{1}{k}$ and

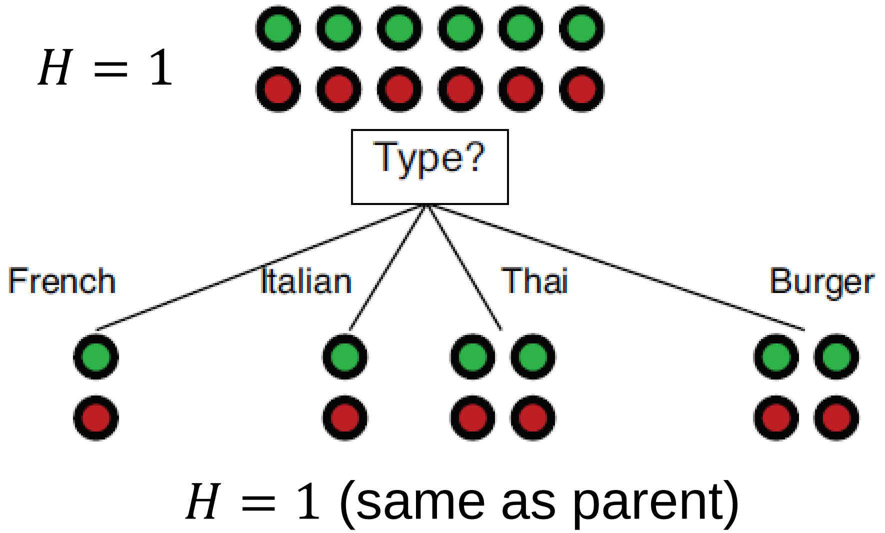
$$H(D) = -k\left(\frac{1}{k}\right)\log_2\left(\frac{1}{k}\right) = \log_2(k)$$

- More classes -> higher entropy

Entropy

$$H(D) = -\sum_{c \in C} d_c \log_2(d_c)$$

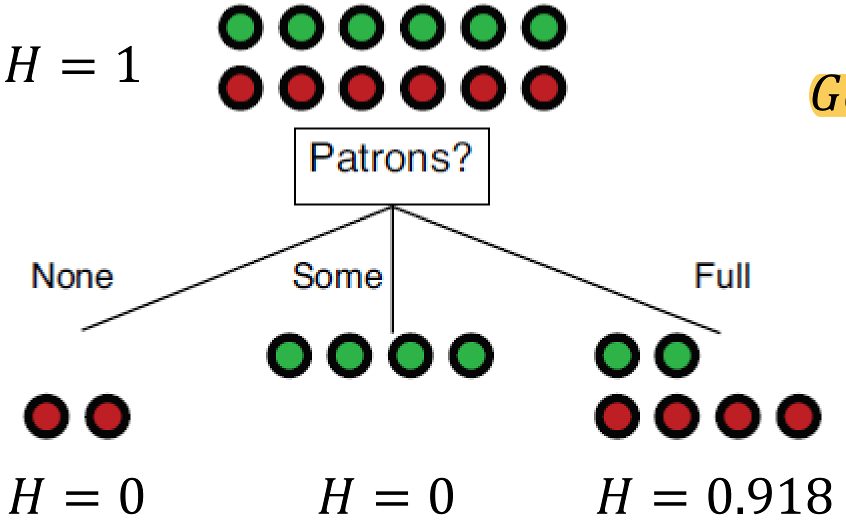




Information Gain

- We want to build a tree that *lowers* entropy with each attribute split
- Choose the attribute A at the current level to be the one that maximizes **expected information gain**, or expected entropy reduction

$$Gain(A) = H(D \text{ before splitting}) - E(H(D \text{ after splitting on } A))$$



$$Gain(Patrons) = 1 - \left(\frac{2}{12}(0) + \frac{4}{12}(0) + \frac{6}{12}(0.918)\right) = 0.541$$

Can also compute $Gain(Type?) = 0$, hence making $Patrons$ a more informative attribute at this level of the tree!

Example: Hiring CS Students

Degree	Experience	Fav. Language	Needs Visa?	Hire?
Bachelors	Mobile Dev	Objective C	yes	yes
Masters	Web Dev	Java	no	yes
Masters	Web Dev	Java	yes	yes
PhD	Mobile Dev	Objective C	yes	yes
PhD	Web Dev	Objective C	yes	no
Bachelors	UX Design	Objective C	yes	no
Bachelors	Mobile Dev	Java	no	yes
PhD	Web Dev	Objective C	no	no
Bachelors	UX Design	Java	no	yes
Masters	UX Design	Objective C	yes	no
Masters	UX Design	Java	no	yes
PhD	Mobile Dev	Java	no	no
Masters	Mobile Dev	Java	yes	yes
Bachelors	Web Dev	Objective C	no	no

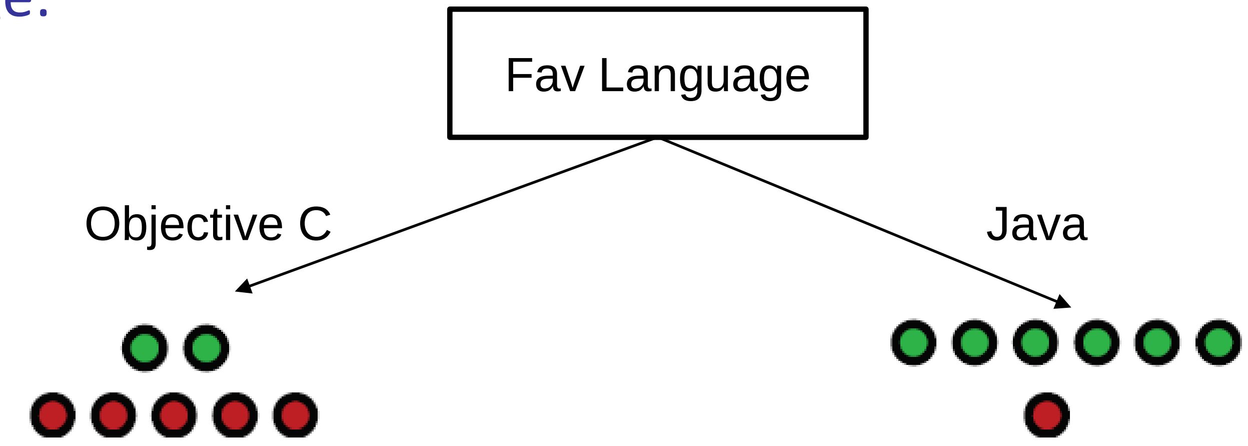
Entropy of data: $H = -\left(\frac{8}{14}\log_2\frac{8}{14}\right) - \left(\frac{6}{14}\log_2\frac{6}{14}\right) = 0.985$

Hire Don’t hire

Root Attribute: Favorite Language

- Need to choose an attribute for the root of the decision tree
- If we choose Favorite Language:

Fav. Language	Hire?
Objective C	yes
Java	yes
Java	yes
Objective C	yes
Objective C	no
Objective C	no
Java	yes
Objective C	no
Java	yes
Objective C	no
Java	yes
Java	no
Java	yes
Objective C	no



$$H = -\left(\frac{2}{7}\log_2\frac{2}{7} + \frac{5}{7}\log_2\frac{5}{7}\right) = 0.863$$

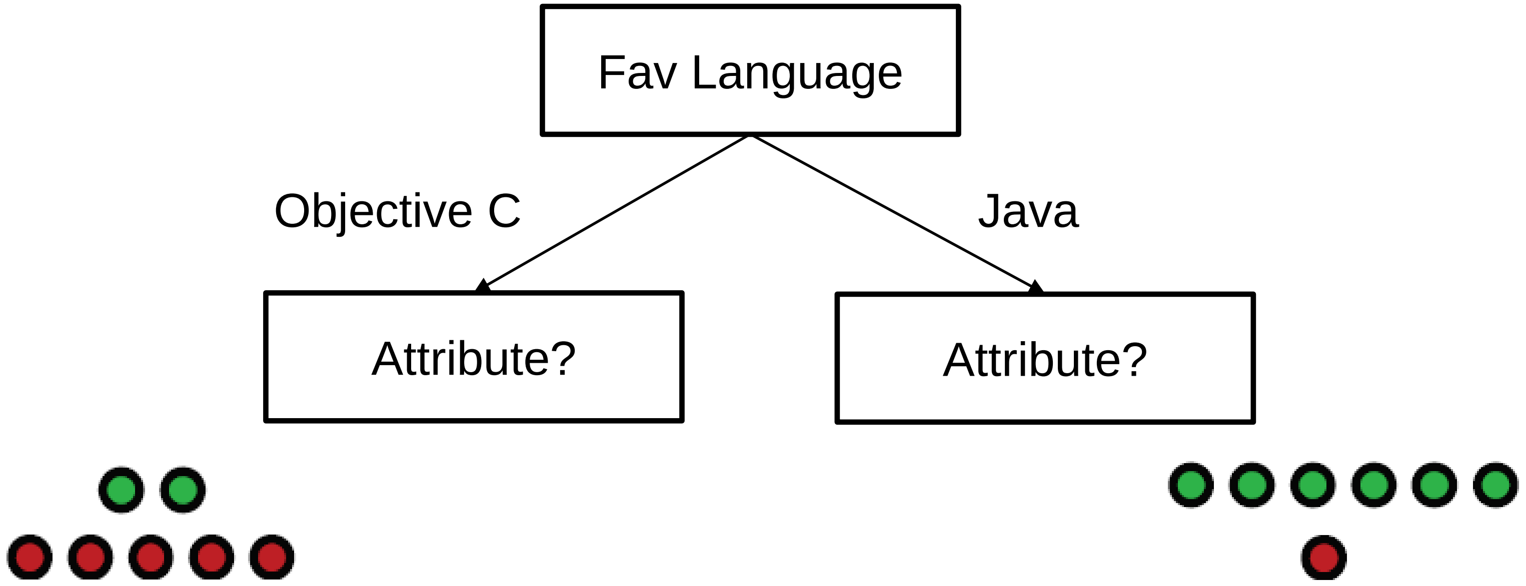
$$H = -\left(\frac{6}{7}\log_2\frac{6}{7} + \frac{1}{7}\log_2\frac{1}{7}\right) = 0.592$$

$$\boldsymbol{Gain} = 0.985 - \left(\frac{7}{14}(0.863) + \frac{7}{14}(0.592)\right) = 0.258$$

Determining Attributes of Subtrees

- Favorite Language provides greatest information gain, so it becomes the tree root
- Self-exercise: Compute information gain for other attributes

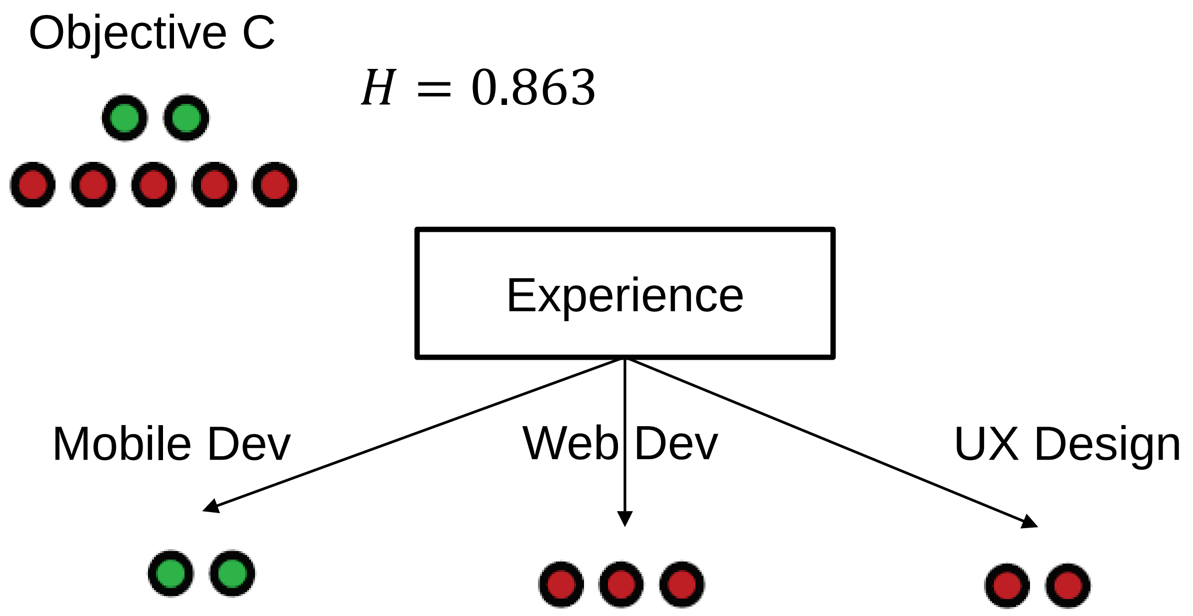
Fav. Language	Hire?
Objective C	yes
Java	yes
Java	yes
Objective C	yes
Objective C	no
Objective C	no
Java	yes
Objective C	no
Java	yes
Objective C	no
Java	yes
Java	no
Java	yes
Objective C	no



Experience Attribute Following "Objective C"

- Let's try Experience after Objective C value of Favorite Language
- Pure split—max entropy gain!

Experience	Fav. Language	Hire?
Mobile Dev	Objective C	yes
Web Dev	Java	yes
Web Dev	Java	yes
Mobile Dev	Objective C	yes
Web Dev	Objective C	no
UX Design	Objective C	no
Mobile Dev	Java	yes
Web Dev	Objective C	no
UX Design	Java	yes
UX Design	Objective C	no
UX Design	Java	yes
Mobile Dev	Java	no
Mobile Dev	Java	yes
Web Dev	Objective C	no



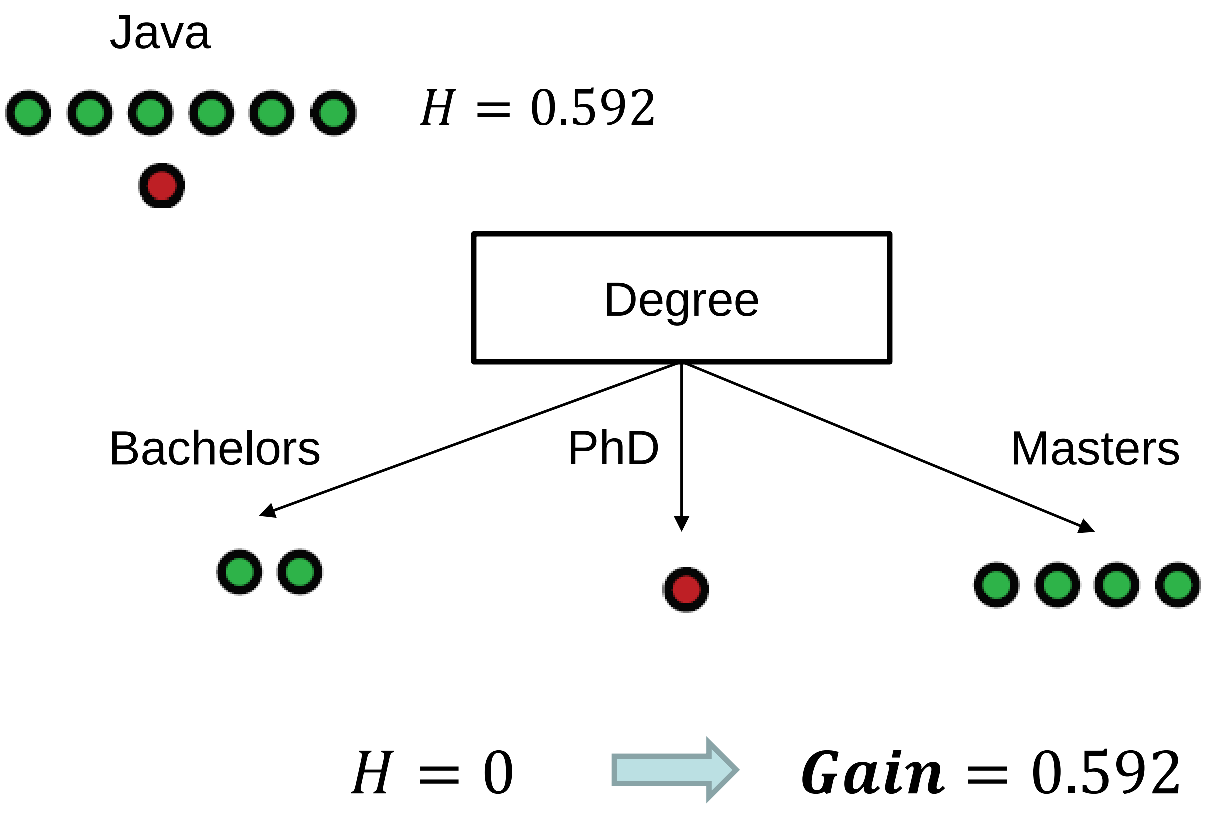
$H = 0.863$

$H = 0$ $\Rightarrow$ $\boldsymbol{Gain} = 0.863$

Degree Attribute Following "Java"

- Let's try Degree after Java value of Favorite Language
- Pure split—max entropy gain!

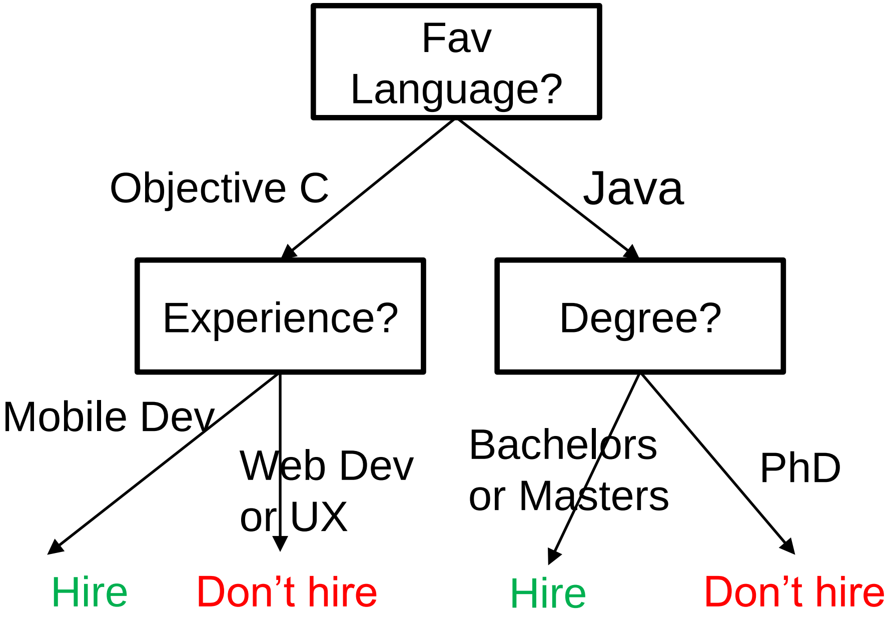
Degree	Fav. Language	Hire?
Bachelors	Objective C	yes
Masters	Java	yes
Masters	Java	yes
PhD	Objective C	yes
PhD	Objective C	no
Bachelors	Objective C	no
Bachelors	Java	yes
PhD	Objective C	no
Bachelors	Java	yes
Masters	Objective C	no
Masters	Java	yes
PhD	Java	no
Masters	Java	yes
Bachelors	Objective C	no



Learned Decision Tree

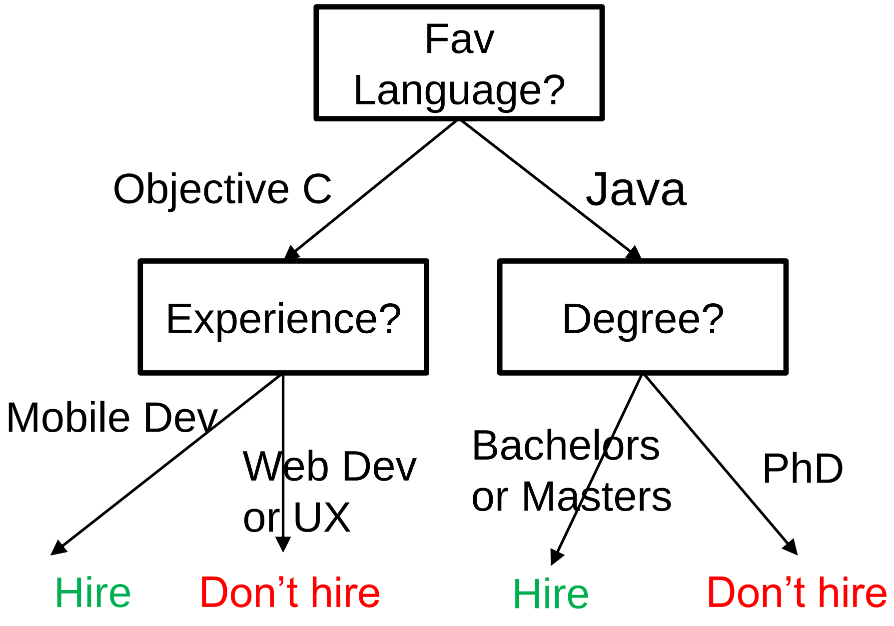
- Our learned decision tree correctly classifies all of our training data
- Smallest size possible—note that some attributes are unused!

Degree	Experience	Fav. Language	Needs Visa?	Hire?
Bachelors	Mobile Dev	Objective C	yes	yes
Masters	Web Dev	Java	no	yes
Masters	Web Dev	Java	yes	yes
PhD	Mobile Dev	Objective C	yes	yes
PhD	Web Dev	Objective C	yes	no
Bachelors	UX Design	Objective C	yes	no
Bachelors	Mobile Dev	Java	no	yes
PhD	Web Dev	Objective C	no	no
Bachelors	UX Design	Java	no	yes
Masters	UX Design	Objective C	yes	no
Masters	UX Design	Java	no	yes
PhD	Mobile Dev	Java	no	no
Masters	Mobile Dev	Java	yes	yes
Bachelors	Web Dev	Objective C	no	no



CS Hiring Decision Tree

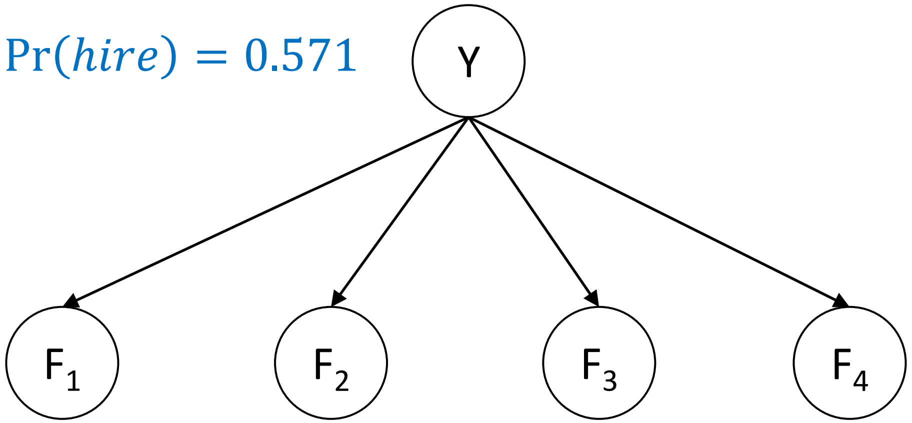
Degree	Experience	Fav. Language	Needs Visa?	Hire?
Bachelors	Mobile Dev	Objective C	yes	yes
Masters	Web Dev	Java	no	yes
Masters	Web Dev	Java	yes	yes
PhD	Mobile Dev	Objective C	yes	yes
PhD	Web Dev	Objective C	yes	no
Bachelors	UX Design	Objective C	yes	no
Bachelors	Mobile Dev	Java	no	yes
PhD	Web Dev	Objective C	no	no
Bachelors	UX Design	Java	no	yes
Masters	UX Design	Objective C	yes	no
Masters	UX Design	Java	no	yes
PhD	Mobile Dev	Java	no	no
Masters	Mobile Dev	Java	yes	yes
Bachelors	Web Dev	Objective C	no	no



Hire (PhD, mobile, Java)?

CS Hiring Naïve Bayes

$\Pr(hire) = 0.571$



$\Pr(visa \mid hire) = 0.5$
$\Pr(no\ visa \mid hire) = 0.5$
$\Pr(visa \mid no\ hire) = 0.5$
$\Pr(no\ visa \mid no\ hire) = 0.5$

$\Pr(bachelors \mid +h) = 0.375$
$\Pr(masters \mid +h) = 0.5$
$\Pr(PhD \mid +h) = 0.125$
$\Pr(bachelors \mid -h) = 0.333$
$\Pr(masters \mid -h) = 0.167$
$\Pr(PhD \mid -h) = 0.5$

$\Pr(mobile \mid +h) = 0.5$
$\Pr(web \mid +h) = 0.25$
$\Pr(UX \mid +h) = 0.25$
$\Pr(mobile \mid -h) = 0.167$
$\Pr(web \mid -h) = 0.5$
$\Pr(UX \mid -h) = 0.333$

$\Pr(Java \mid hire) = 0.75$
$\Pr(ObjC \mid hire) = 0.25$
$\Pr(Java \mid no\ hire) = 0.167$
$\Pr(ObjC \mid no\ hire) = 0.833$

Hire (PhD, mobile, Java)?
Yes: $0.571 \times 0.125 \times 0.5 \times 0.75 = 0.027$
No: $0.429 \times 0.5 \times 0.167 \times 0.167 = 0.006$

Decision Tree Considerations

- Many ways of choosing attributes (e.g., gain ratio, Gini index)
- Learning can be *unstable*: small differences in data can lead to very different trees

- **Overfitting** can occur when we have too many attributes
- Can split on rare or irrelevant attributes to obtain purer splittings
- Allow us to better classify training data, but vulnerable to noise in data

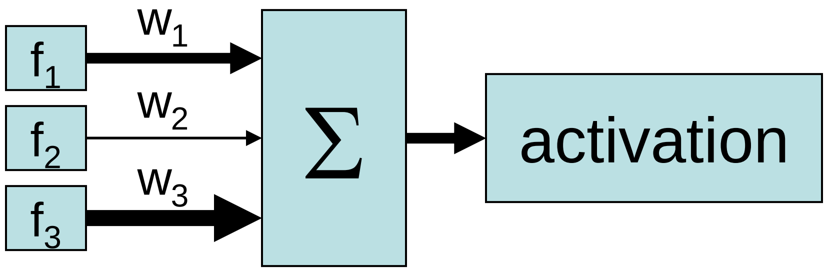
- *Early stopping* or *pruning* can help—simply pretend non-leaves are leaves
- Allow for non-pure splits and classification errors in training data
- In such cases, classify based on class plurality of remaining data

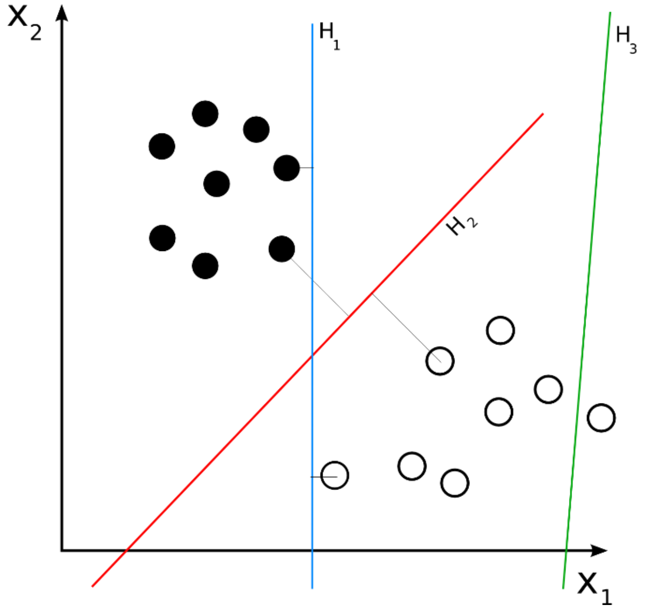
Linear Classifiers

- Idea: Feature inputs live in a high-dimensional vector space
- Distinct areas of the space correspond to distinct classes
- A **weight vector** w determines an **activation score** for a set of features

$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- Output label depends on score

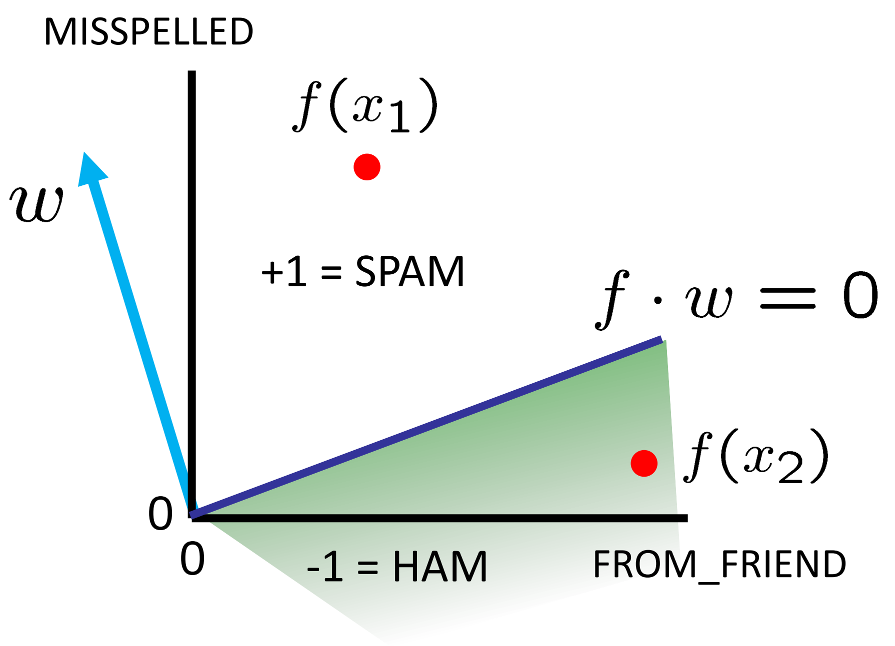




Binary Decision Boundary

- For a **binary-class** problem, score is compared to a *threshold* (e.g., 0) to classify an input
- The two classes correspond to two *linearly separated* regions of the feature space!
- Weight vector generates a **linear separator**
- Feature scores on boundary equal to 0
- Weight vector itself is *orthogonal* to the decision boundary

$Score > 0$: Predict class +1
$Score < 0$: Predict class -1
$Score = 0$: Tiebreaker choice



Example: Linear Classifier

Degree	Experience	Fav. Language	Needs Visa?	Hire?
Bachelors	Mobile Dev	Objective C	yes	yes
Masters	Web Dev	Java	no	yes
Masters	Web Dev	Java	yes	yes
PhD	Mobile Dev	Objective C	yes	yes
PhD	Web Dev	Objective C	yes	no
Bachelors	UX Design	Objective C	yes	no
Bachelors	Mobile Dev	Java	no	yes
PhD	Web Dev	Objective C	no	no
Bachelors	UX Design	Java	no	yes
Masters	UX Design	Objective C	yes	no
Masters	UX Design	Java	no	yes
PhD	Mobile Dev	Java	no	no
Masters	Mobile Dev	Java	yes	yes
Bachelors	Web Dev	Objective C	no	no

$\xrightarrow{f}$

Deg	Exp	Lang	Visa?	**Hire?**
0	0	-1	1	**1**
1	1	1	-1	**1**
1	1	1	1	**1**
-1	0	-1	1	**1**
-1	1	-1	1	**-1**
0	-1	-1	1	**-1**
0	0	1	-1	**1**
-1	1	-1	-1	**-1**
0	-1	1	-1	**1**
1	-1	-1	1	**-1**
1	-1	1	-1	**1**
-1	0	1	-1	**-1**
1	0	1	1	**1**
0	1	-1	-1	**-1**

Example: Linear Classifier

- Activation score is

$$w \cdot f(x) = (w_{deg}, w_{exp}, w_{lang}, w_{visa}) \cdot f(x)$$

- Suppose we have weights $w = (3,2,2,0)$
- Equation of a hyperplane in 4D space:

$$y = 3f_{deg}(x) + 2f_{exp}(x) + 2f_{lang}(x) + 0f_{visa}(x)$$

- $w \cdot f(x_1) = 3(0) + 2(0) + 2(-1) = -2$ -> **No hire**
- $w \cdot f(x_2) = 3(1) + 2(1) + 2(1) = 7$ -> **Hire**
- $w \cdot f(x_{14}) = 3(0) + 2(1) + 2(-1) = 0$ -> Tie

$f(x)$	Deg	Exp	Lang	Visa?
x_1	0	0	-1	1
x_2	1	1	1	-1
x_3	1	1	1	1
x_4	-1	0	-1	1
x_5	-1	1	-1	1
x_6	0	-1	-1	1
x_7	0	0	1	-1
x_8	-1	1	-1	-1
x_9	0	-1	1	-1
x_{10}	1	-1	-1	1
x_{11}	1	-1	1	-1
x_{12}	-1	0	1	-1
x_{13}	1	0	1	1
x_{14}	0	1	-1	-1

Binary Perceptron Learning Rule

- We've assumed a linear model. How to learn the weights?
- **Error-driven learning**: Use current model (weights) to classify training data; update model if results are incorrect

Initialize weights w (e.g., all 0) and learning rate α

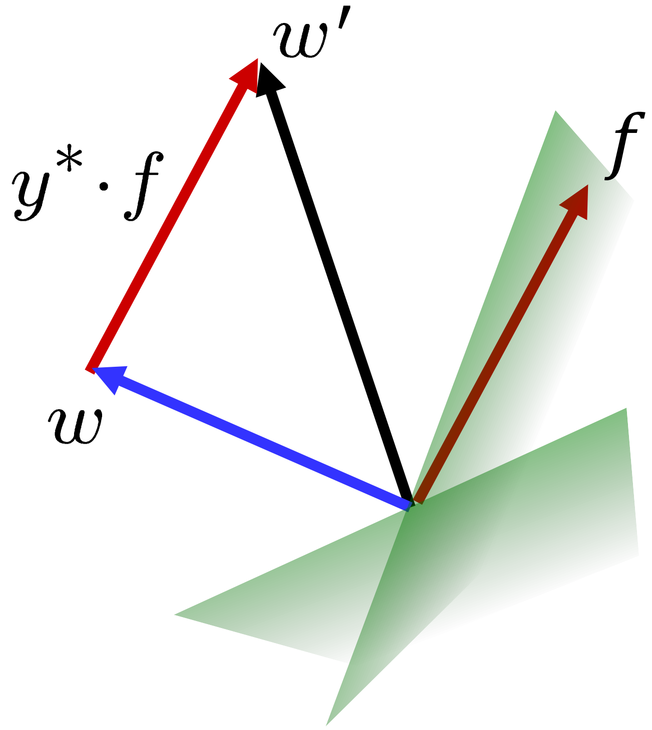
While not converged:

  For each training instance $f(x)$:

    Predict class y using current weights w

    If incorrect ($y \neq y^*$):

      Update weights: $w \leftarrow w + \alpha(y^* \times f(x))$



Example: CS Hiring

- Initialize $w = (0,0,0,0)$, $\alpha = 1$; ties predict hire (1)

- $f(x_1)$ to $f(x_4)$: Predict **hire (1)**, no update to w

- $w \cdot f(x_5) = 0$: incorrectly predict **hire (1)**
  - $w \leftarrow w - f(x_5) = (1, -1, 1, -1)$

- $w \cdot f(x_6) = -1$; correctly predict **no hire (-1)**
- $w \cdot f(x_7) = 2$; correctly predict **hire (1)**
- $w \cdot f(x_8) = -2$; correctly predict **no hire (-1)**
- …
- What's the next update to w?

Sample	Deg	Exp	Lang	Visa?	**Hire?**
1	0	0	-1	1	**1**
2	1	1	1	-1	**1**
3	1	1	1	1	**1**
4	-1	0	-1	1	**1**
5	-1	1	-1	1	**-1**
6	0	-1	-1	1	**-1**
7	0	0	1	-1	**1**
8	-1	1	-1	-1	**-1**
9	0	-1	1	-1	**1**
10	1	-1	-1	1	**-1**
11	1	-1	1	-1	**1**
12	-1	0	1	-1	**-1**
13	1	0	1	1	**1**
14	0	1	-1	-1	**-1**

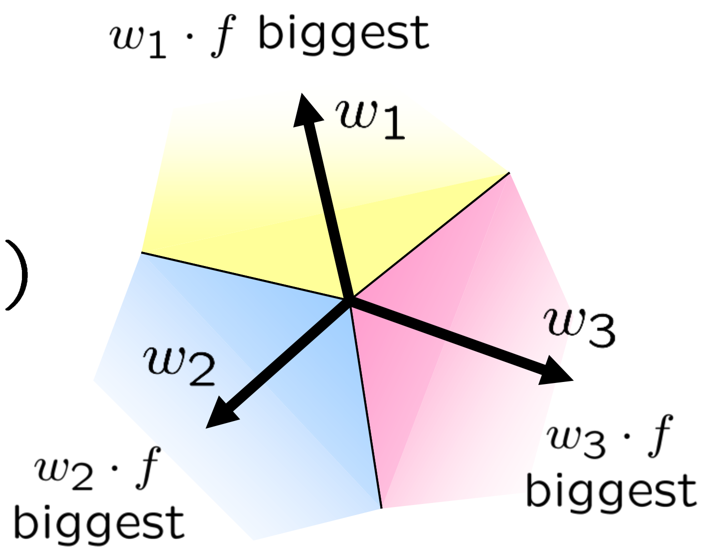
Multiclass Perceptron

- What if we have more than two classes?
- Feature space is split into multiple regions!

- Multiple weight vectors, one for each class
- Compute activation score for each class: $w_y \cdot f(x)$

- Predicted class is the one with largest score:

$$y = \arg\max_y \; w_y \cdot f(x)$$



Multiclass Perceptron Learning Rule

- Error-driven learning can also learn a multiclass perceptron
- We perform weight updates for *both* the correct and incorrect classes!

Initialize weights w_c for each class c and learning rate α

While not converged:

For each training instance $f(x)$:

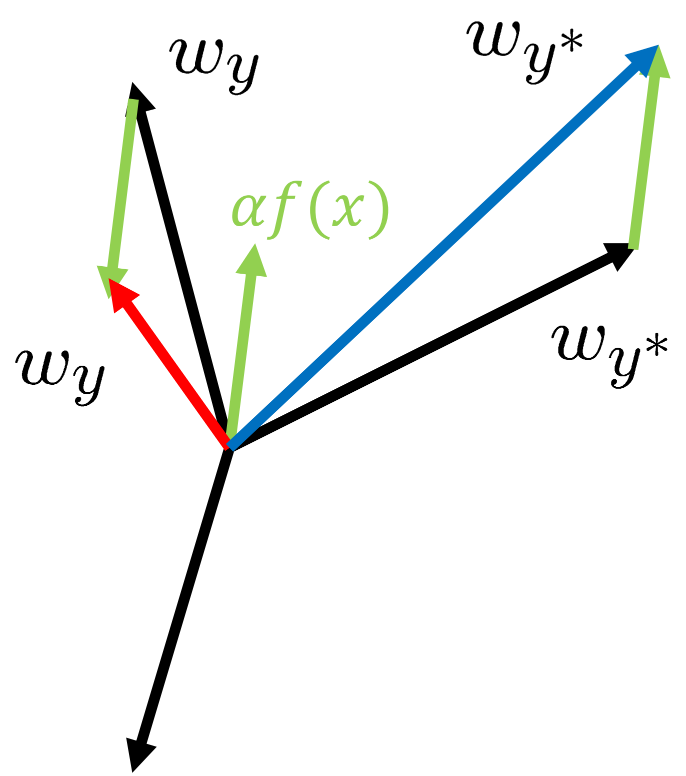
Obtain predicted class y using argmax rule

$$y = \arg\max_y \; w_y \cdot f(x)$$

If incorrect ($y \neq y^*$): Update weights for classes y and y^*

$$w_y \leftarrow w_y - \alpha f(x) \quad \text{Wrongly predicted class}$$

$$w_y^* \leftarrow w_y^* + \alpha f(x) \quad \text{Correct class}$$



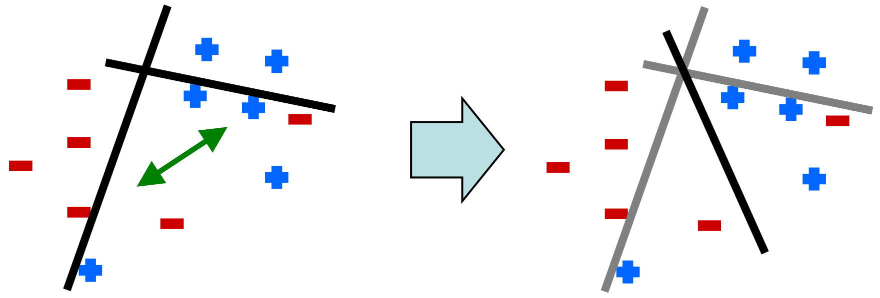
Example: Multiclass CS Hiring

- Weights so far: $w_{-1} = (0,0,0,0), w_0 = (1,1,1,-1)$, $w_1 = (0,0,-1,1), \alpha = 1$; ties predict -1
- $f(x_1)$, ..., $f(x_4)$ all correct
- $\text{argmax}\, w_i \cdot f(x_5) = 1$; **incorrect**
  - $w_1 \leftarrow w_1 - f(x_5) = (1,-1,0,0)$
  - $w_{-1} \leftarrow w_{-1} + f(x_5) = (-1,1,-1,1)$
- $\text{argmax}\, w_i \cdot f(x_6) = -1$; **incorrect**
  - $w_{-1} \leftarrow w_{-1} - f(x_6) = (-1,2,0,0)$
  - $w_0 \leftarrow w_0 + f(x_6) = (1,0,0,0)$
- $\text{argmax}\, w_i \cdot f(x_7) = -1$; **incorrect**…

Sample	Deg	Exp	Lang	Visa?	**Hire?**
1	0	0	-1	1	**1**
2	1	1	1	-1	**0**
3	1	1	1	1	**0**
4	-1	0	-1	1	**1**
5	-1	1	-1	1	**-1**
6	0	-1	-1	1	**0**
7	0	0	1	-1	**1**
8	-1	1	-1	-1	**-1**
9	0	-1	1	-1	**0**
10	1	-1	-1	1	**-1**
11	1	-1	1	-1	**1**
12	-1	0	1	-1	**-1**
13	1	0	1	1	**1**
14	0	1	-1	-1	**-1**

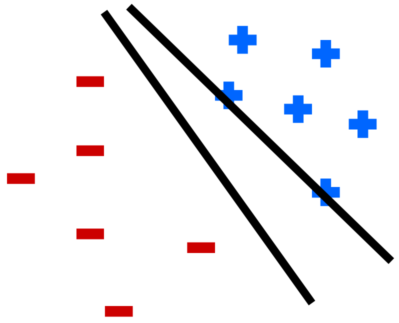
Perceptron Properties and Issues

- Weights will thrash if data is not linearly separable!



- Use adaptive learning rate that decreases over time
- Forego perfect accuracy on training data

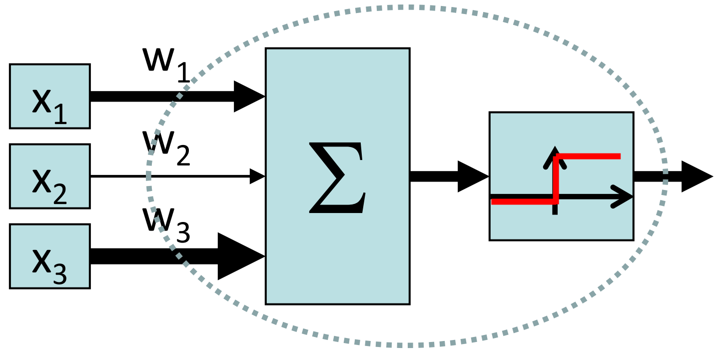
- Otherwise, perceptron will eventually **converge** with finite number of updates
- (See perceptron convergence theorem)



- But may overfit and not generalize very well
- Also mediocre solutions, such as barely separating margins

Perceptrons as Neurons

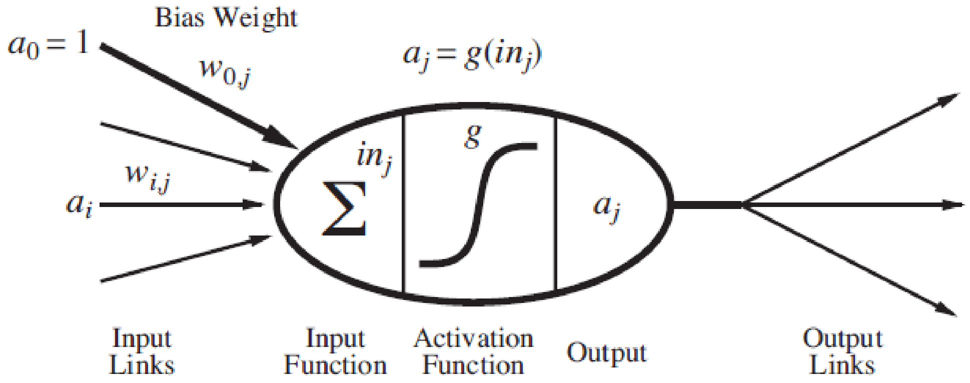
- Perceptron uses a *hard threshold* to find output from linear combination of inputs
  - Not limited to linear functions (e.g. kernels)



- We can use other **activation functions** and combine multiple outputs together!

$$a_j = g(in_j) = g\left(\sum_i w_{i,j} a_i\right)$$

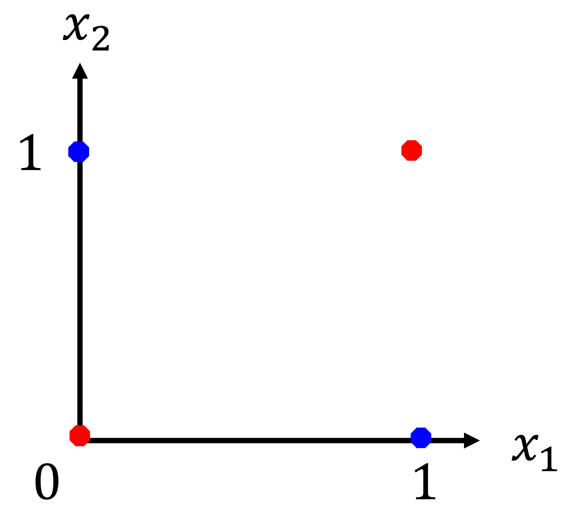
- **Neural network**: Set of *nodes* N, links and *weights* $w_{i,j}$, *activation function* g

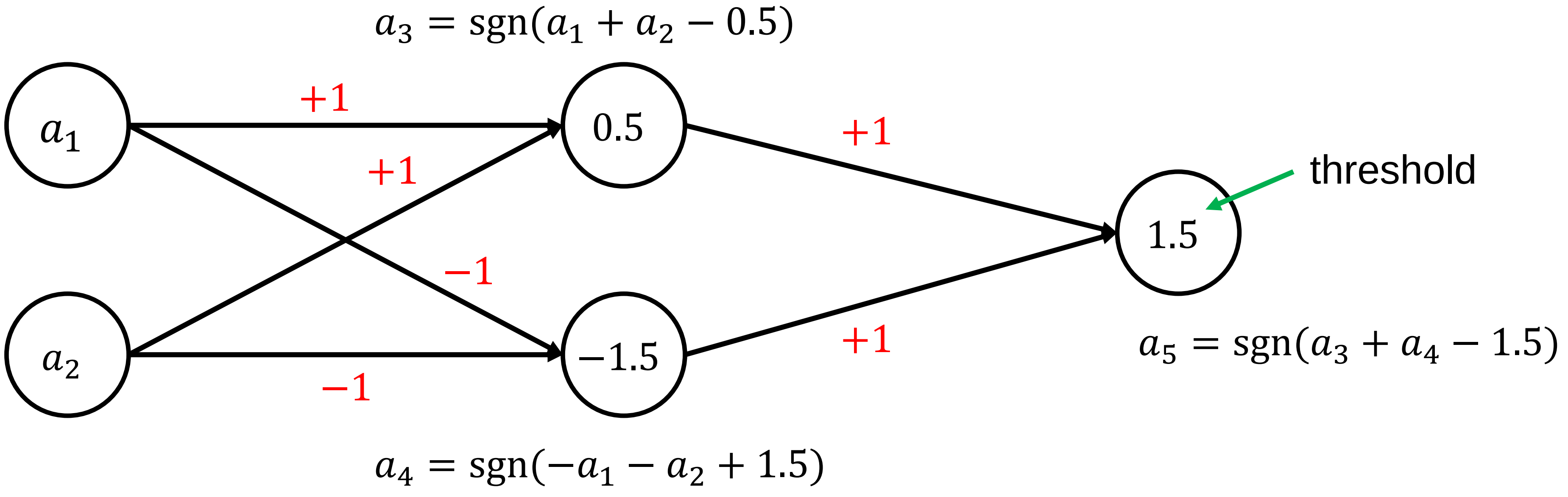


Example: XOR Function

- No linear separator exists for these 4 points
- Need some kind of nonlinear decision boundary

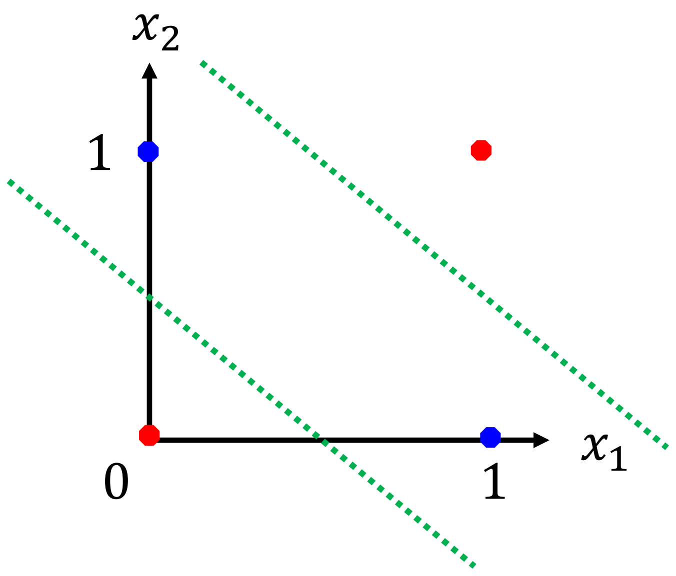
- How about multiple perceptrons?

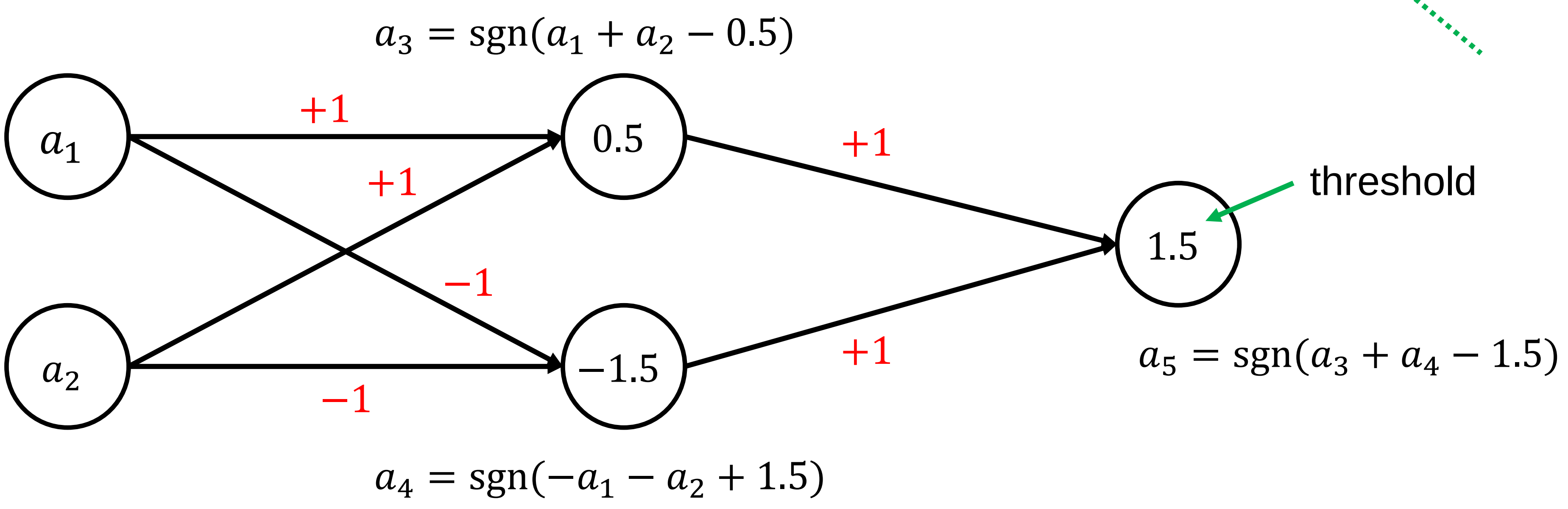




Example: XOR Function

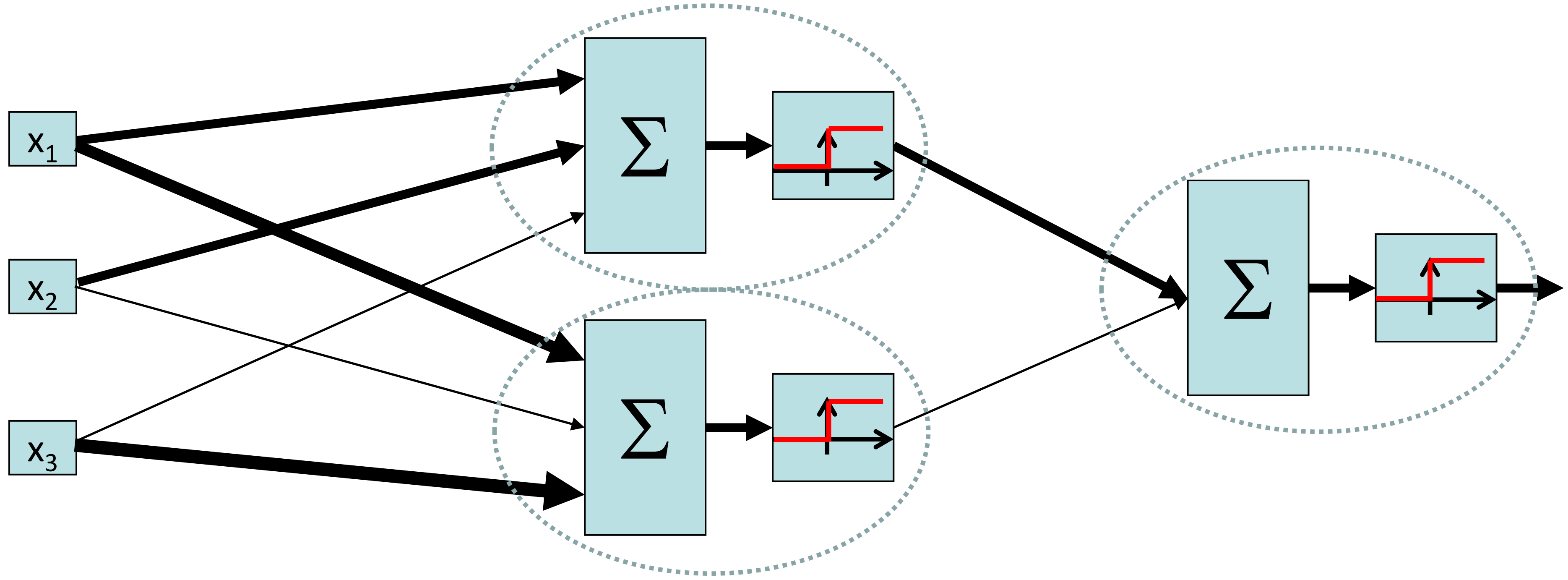
- $x = (a_1, a_2) = (0,0) \rightarrow a_3 = 0, a_4 = 1 \rightarrow a_5 = 0$
- $x = (a_1, a_2) = (1,1) \rightarrow a_3 = 1, a_4 = 0 \rightarrow a_5 = 0$
- $x = (a_1, a_2) = (0,1) \rightarrow a_3 = 1, a_4 = 1 \rightarrow a_5 = 1$
- $x = (a_1, a_2) = (1,0) \rightarrow a_3 = 1, a_4 = 1 \rightarrow a_5 = 1$





Multilayer Perceptrons

- **Universal approximation theorem:** Any continuous (including nonlinear!) function can be approximated with a two-layer network*
  - *With certain requirements on activation function

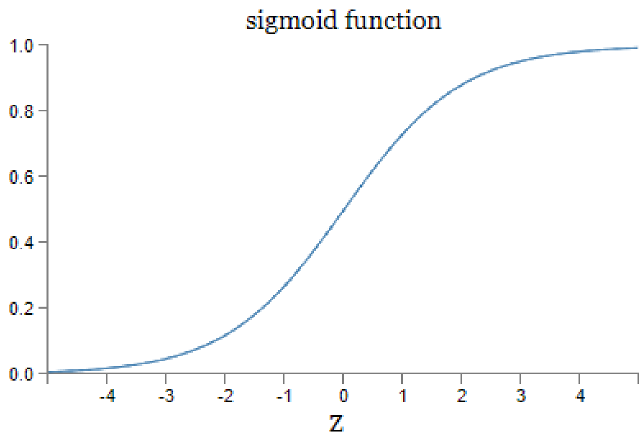


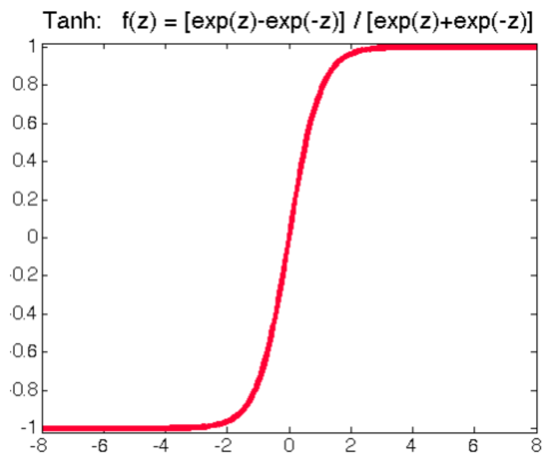
Input layer **Hidden layer** **Output layer**

Activation Functions

- We aren't limited to the step threshold as our activation function

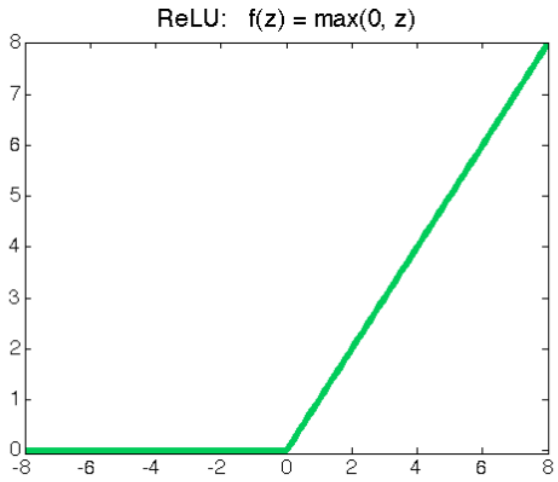
- Sigmoid: $S(z) = \frac{1}{1+\exp(-z)}$



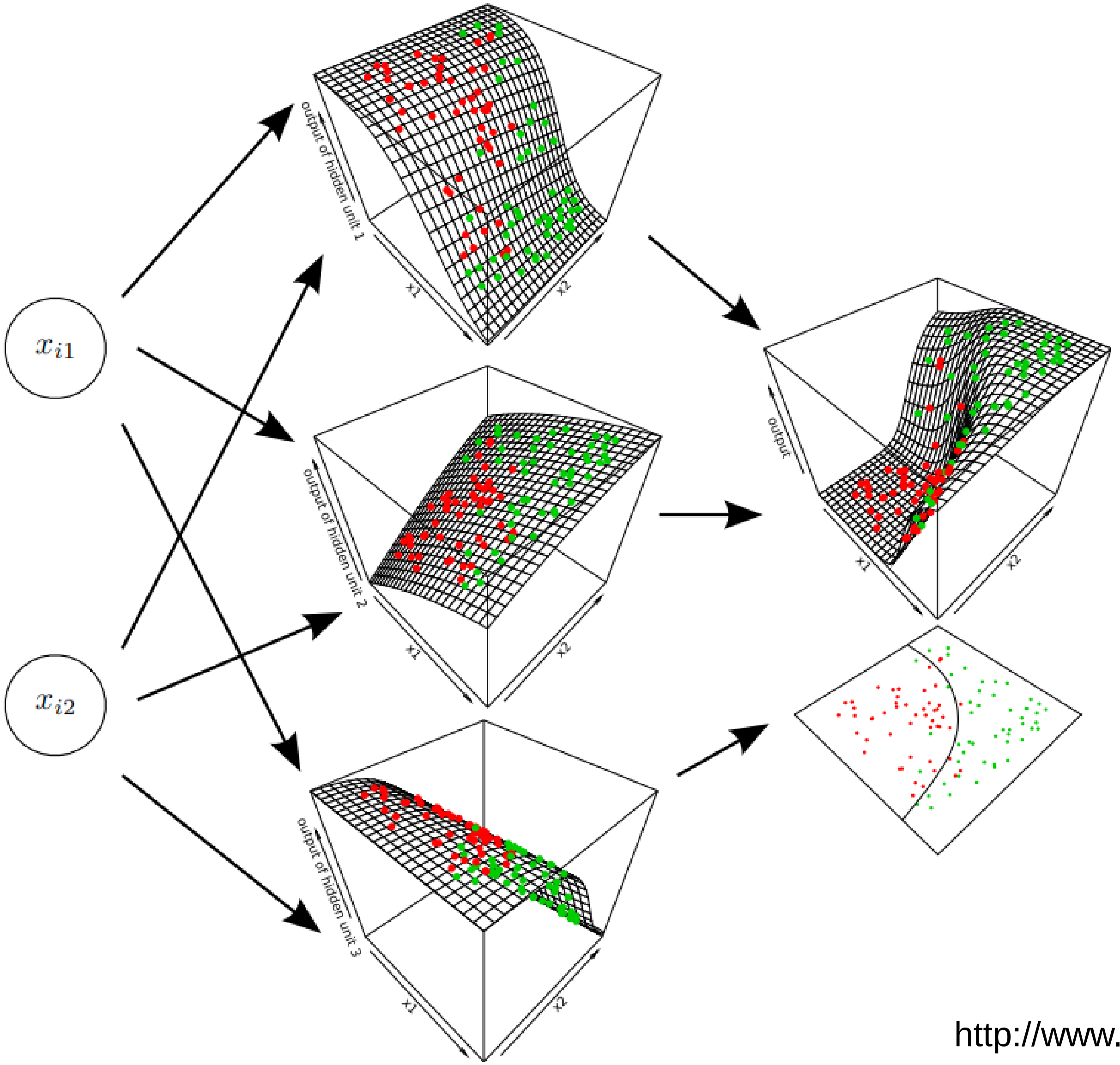


- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$

- Rectified linear unit: $\mathrm{ReLU}(z) = \max(0, z)$



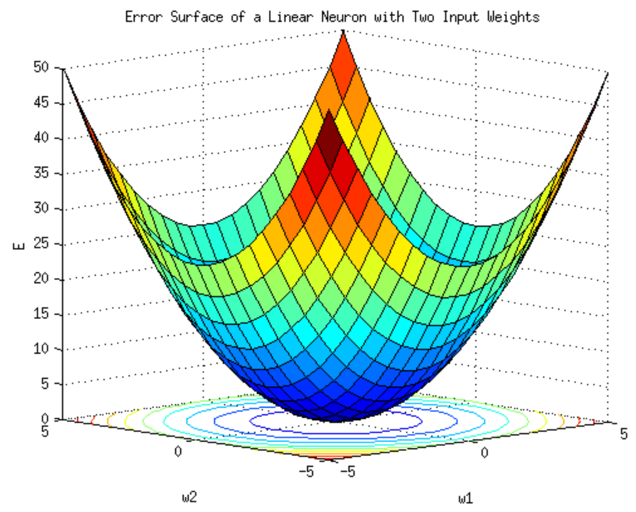
Example: Combining Sigmoids



http://www.stats.ox.ac.uk/~sejdinov/talks/

Training Network Weights

- How to find the network weights? We generally want to minimize some **loss function**
- E.g., quadratic loss: $E(x) = \sum_i \frac{1}{2}(y_i - \hat{y}_i)^2$
- Or any other functions that are "easy" to optimize over



- Idea: Make a prediction on a training data instance
- If incorrect, find error contribution from each network unit

- How? Compute derivatives of error wrt network units!
- Weights can then be updated according to **gradient descent**

Many Other ML Tasks…

- Regression (linear and otherwise) for continuous data

- Unsupervised learning and clustering
- How to detect patterns in unlabeled data?

- Reinforcement learning: Using neural networks to represent **Q-functions** instead of explicit Q-values

- Learning probabilistic model structure and parameters