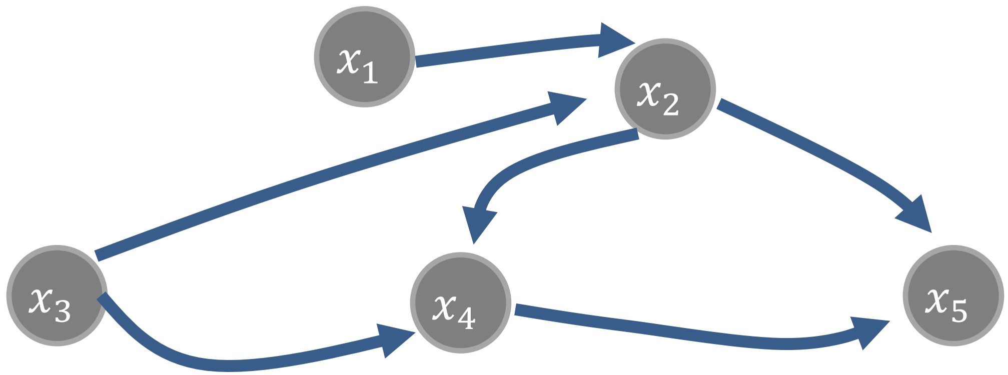
CMSC 471: Reasoning with Bayesian Belief Network

Chapters 12 & 13

KMA Solaiman – ksolaima@umbc.edu

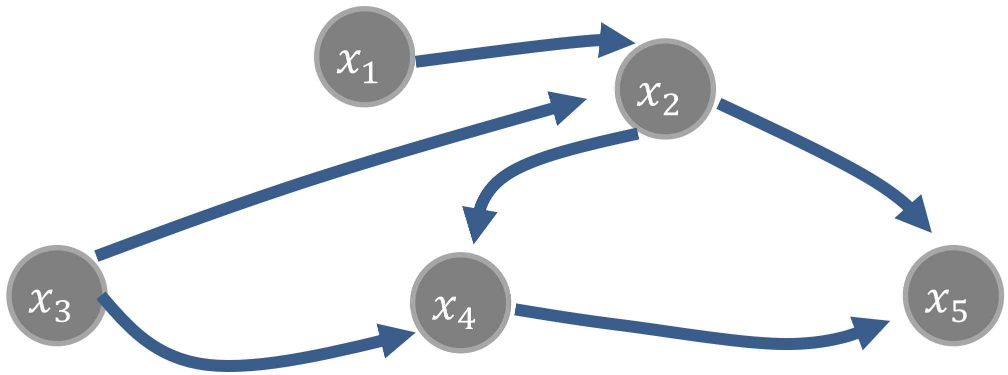
Some slides courtesy Tim Finin and Frank Ferraro

Bayesian Networks: Directed Acyclic Graphs



$$p(x_1, x_2, x_3, x_4, x_5) =$$
$$p(x_1)p(x_3)p(x_2|x_1, x_3)p(x_4|x_2, x_3)p(x_5|x_2, x_4)$$

Bayesian Networks: Directed Acyclic Graphs

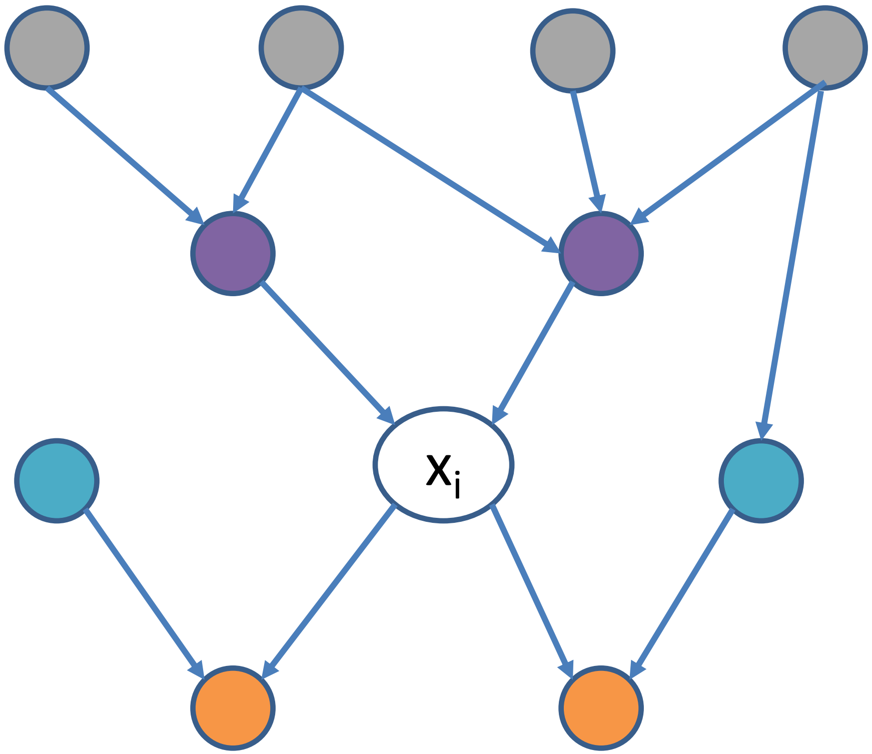


$$p(x_1, x_2, x_3, \dots, x_N) = \prod_i p(x_i \mid \pi(x_i))$$

exact inference in general DAGs is NP-hard

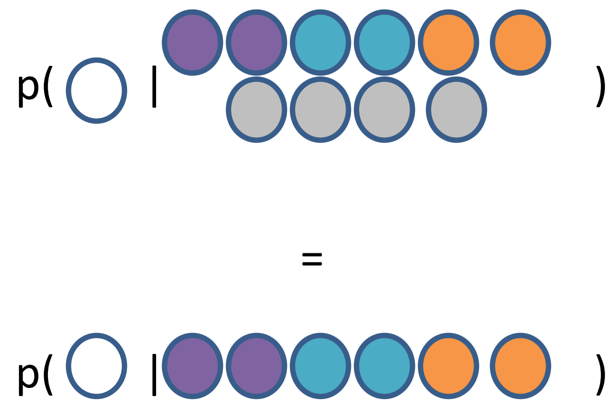
inference in trees can be exact

Markov Blanket



Markov blanket of a node x is its parents, children, and children's parents

The **Markov Blanket** of a node x_i the set of nodes needed to form the complete conditional for a variable x_i



Given its Markov blanket, a node is conditionally independent of all other nodes in the BN

(in this example, shading does not show observed/latent)

Variable Elimination

- Inference: Compute posterior probability of a node given some other nodes

$$p(Q|x_1, \dots, x_j)$$

- Variable elimination: An algorithm for exact inference
  - Uses dynamic programming
  - Not necessarily polynomial time!

Variable Elimination (High-level)

Goal: $p(Q|x_1, \ldots, x_j)$

(The word "factor" is used for each CPT.)

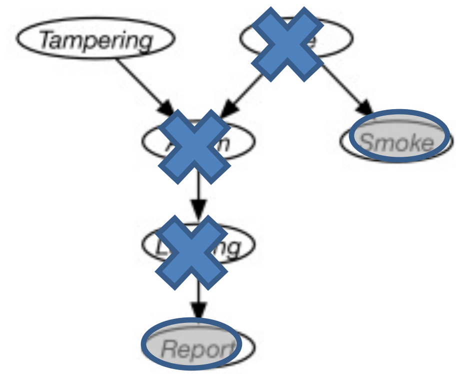
1. Pick one of the non-conditioned, MB variables
2. Eliminate this variable by marginalizing (summing) it out from all factors (CPTs) that contain it
3. Go back to 1 until no (MB) variables remain
4. Multiply the remaining factors and normalize.

Variable Elimination: Example

(The word "factor" is used for each CPT.)

1. Pick one of the non-conditioned, MB variables
2. Eliminate this variable by marginalizing (summing) it out from all factors (CPTs) that contain it
3. Go back to 1 until no (MB) variables remain
4. Multiply the remaining factors and normalize.

Conditional Probability	*Factor*
$P(Tampering)$	$f_0(Tampering)$
$P(Fire)$	$f_1(Fire)$
$P(Alarm \mid Tampering, Fire)$	$f_2(Tampering, Fire, Alarm)$
$P(Smoke = yes \mid Fire)$	$f_3(Fire)$
$P(Leaving \mid Alarm)$	$f_4(Alarm, Leaving)$
$P(Report = yes \mid Leaving)$	$f_5(Leaving)$



Goal: P(Tampering | Smoke=true ∧ Report=true)

Task: Normalize in order to compute ***p(Tampering)***

We'll have a single factor f8(Tampering):

$$p(T = yes) = \frac{f_8(T = yes)}{f_8(T = yes) + f_8(T = no)}$$

Variable Elimination: Example

- The posterior distribution over *Tampering* is given by

$$\frac{P(Tampering = u)\, f_8(Tampering = u)}{\sum_v P(Tampering = v)\, f_8(Tampering = v)}$$

Fundamental Inference Question

- Compute posterior probability of a node given some other nodes
$$p(Q|x_1, \dots, x_j)$$
- Some techniques
  - MLE (maximum likelihood estimation)/MAP (maximum a posteriori) [covered 2nd]
  - Variable Elimination [covered 1st]
  - (Loopy) Belief Propagation ((Loopy) BP)
  - Monte Carlo
  - Variational methods
  - …

Advanced topics

Parameter estimation

- Assume known structure
- Goal: estimate BN parameters θ
  - entries in local probability models, $P(X \mid \text{Parents}(X))$
- A parameterization θ is good if it is likely to generate the observed data:

$$L(\theta : D) = P(D \mid \theta) = \prod_m P(x[m] \mid \theta)$$

i.i.d. samples

- Maximum Likelihood Estimation (MLE) Principle: Choose θ^* so as to maximize *L*

Parameter estimation II

- The likelihood **decomposes** according to the structure of the network

  → we get a separate estimation task for each parameter
- The MLE (maximum likelihood estimate) solution for **discrete** data & RV values:
  - for each value *x* of a node *X*
  - and each instantiation ***u*** of *Parents(X)*

$$\theta^*_{\boldsymbol{x}|\boldsymbol{u}} = \frac{N(\boldsymbol{x},\boldsymbol{u})}{N(\boldsymbol{u})}$$

sufficient statistics

  - Just need to collect the counts for every combination of parents and children observed in the data
  - MLE is equivalent to an assumption of a uniform prior over parameter values

Machine Learning: Decision Trees

Chapter 19.3

Some material adopted from notes by Chuck Dyer

Choosing best attribute

- **Key problem:** choose attribute to split given set of examples
- Possibilities for choosing attribute:
  - **Random:** Select one at random
  - **Least-values:** one with smallest # of possible values
  - **Most-values:** one with largest # of possible values
  - **Max-gain:** one with largest expected ***information gain***
  - **Gini impurity:** one with smallest gini impurity value
- The last two measure the **homogeneity** of the target variable within the subsets
- The ID3 and C4.5 algorithms uses **max-gain**

A Simple Example

For this data, is it better to start the tree by asking about the restaurant **type** or its current **number of patrons**?

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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

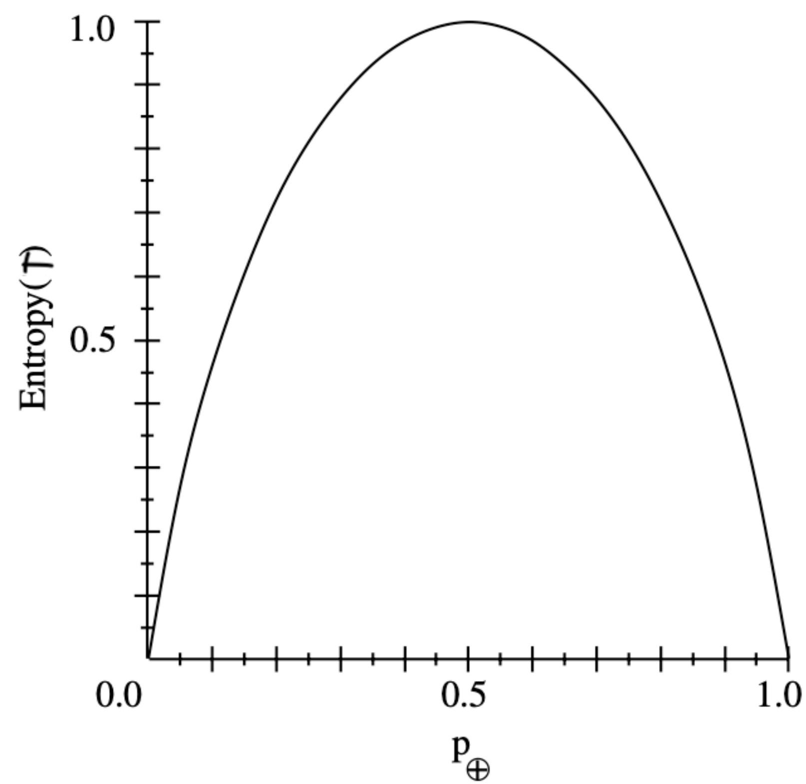
Information gain in knowing an attribute

- **Gain(X,T) = Info(T) - Info(X,T)** is difference of
  - **Info(T):** info needed to identify T's class
  - **Info(X,T):** info needed to identify T's class after attribute X's value known
- This is gain in information due to knowing value of attribute X
- Used to rank attributes and build DT where each node uses attribute with greatest gain of those not yet considered in path from root
- goal: **create small DTs** to minimize questions

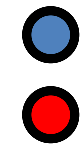
Information Gain

Info(T) = Entropy (T)
$= -\sum_c \widehat{p_c} \log_2 \widehat{p_c}$

Info(X, T)
= expected reduction in entropy due to sorting on X
$= \sum_i \frac{|T_i|}{|T|} \text{Info}(T_i)$



Information Gain



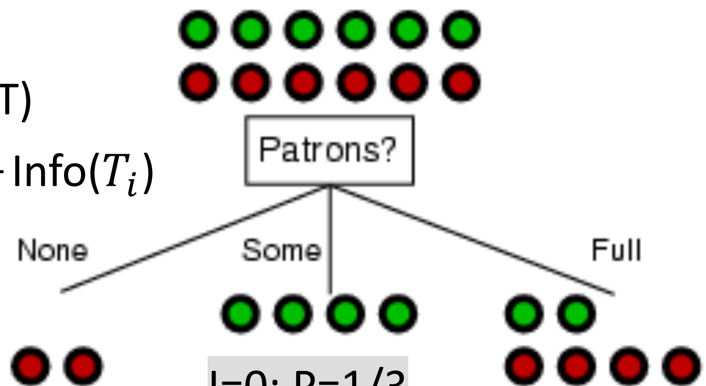
I = Info(T)

$= -\sum_c \widehat{p_c} \log_2 \widehat{p_c}$

I = -(.5*$\log_2$(.5) + .5*$\log_2$(.5)) = 0.5+0.5 => **1.0**

Info(X, T)

$= \sum_i \frac{|T_i|}{|T|} \text{Info}(T_i)$

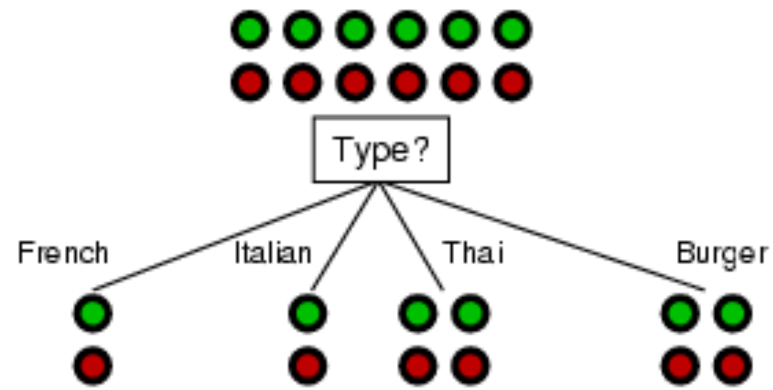


I=0; P=1/6

I=0; P=1/3

I=-(1/3*$\log_2$(1/3)+2/3*$\log_2$(2/3)),
P=6/12=1/2 => 0.91/2 = 0.46

Information gain = 1 - 0.46 => **0.54**



I=1;P=1/6 | I=1; P=1/6 | I=1; P=2/6 | I=1; P=2/6

I = 6/6*1 => **1.0**

Information gain = 1 - 1 => **0.0**

- **Information gain** for asking **Patrons = 0.54**, for asking **Type = 0**
- Note: If only one of the N categories has any instances, the information entropy is always 0

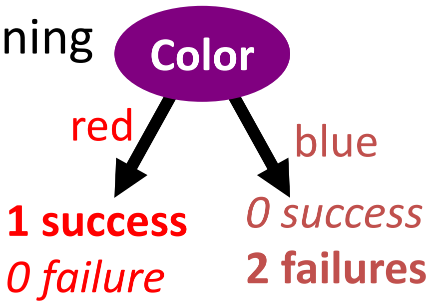
Avoiding Overfitting

- Remove obviously **irrelevant features**
  - E.g., remove 'year observed', 'month observed', 'day observed', 'observer name' from the attributes used
- Get **more training data**
- **Pruning** lower nodes in a decision tree
  - E.g., if info. gain of best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes

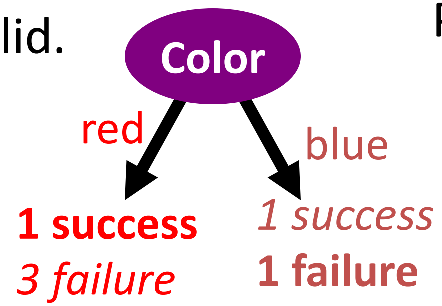
Pruning decision trees

- Pruning a decision tree is done by replacing a whole subtree by a leaf node
- Replacement takes place if the expected error rate in the subtree is greater than in the single leaf, e.g.,
  - **Training data:** 1 training red success and 2 training blue failures
  - **Validation data**: 3 red failures and one blue success
  - Consider replacing subtree by a single node indicating failure
- After replacement, only 2 errors instead of 4

Training



Valid.

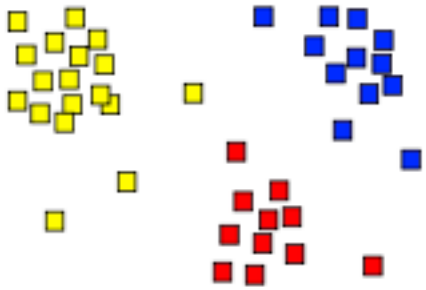


Pruned

FAILURE

2 success

4 failure



Unsupervised Learning: Clustering

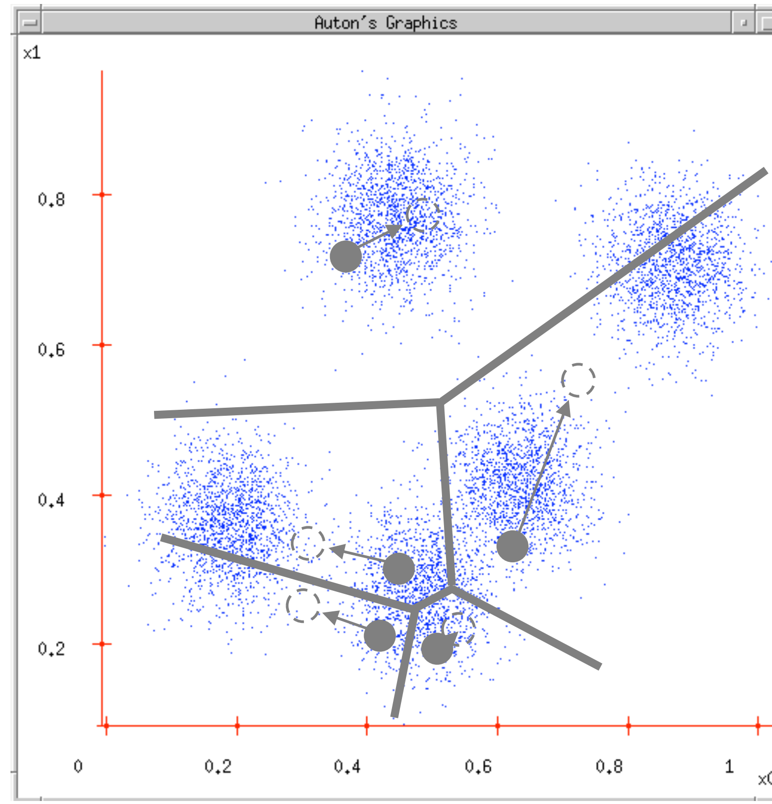
Introduction and Simple K-means

Some material adapted from slides by Andrew Moore, CMU

K-Means Clustering

K-Means (k, data)

- Randomly choose k cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of closest centroid
  - Re-estimate cluster centroids based on data assigned to each
- **Convergence: no point is assigned to a different cluster**



Problems with K-Means

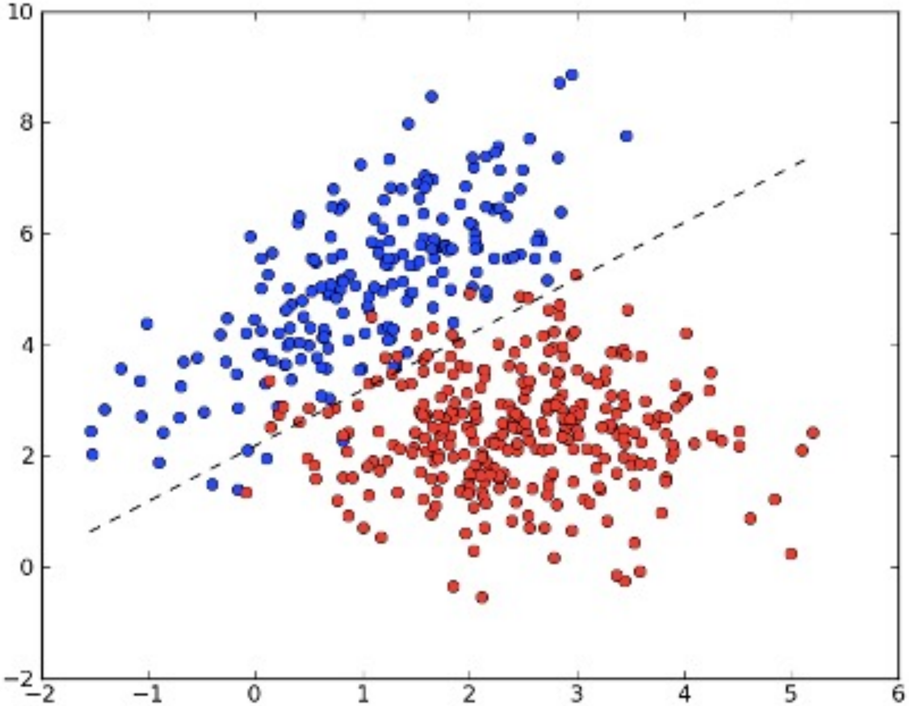
- Only works for numeric data (typically reals)
- ***Very*** sensitive to the initial points
  - **fix:** Do many runs, each with different initial centroids
  - **fix:** Seed centroids with non-random method, e.g., **farthest-first** sampling
- Sensitive to outliers
  - **fix:** identify and remove outliers
- **Must manually choose k**
  - **E.g.: find three**
  - Learn optimal k using some performance measure

CMSC 471: Machine Learning

KMA Solaiman – ksolaima@umbc.edu

Some slides courtesy Tim Finin and Frank Ferraro

Linear Models: Core Idea



Model the relationship between the input data X and corresponding labels Y via a linear relationship (non-zero intercepts b are okay)

$$Y = W^T X + b$$

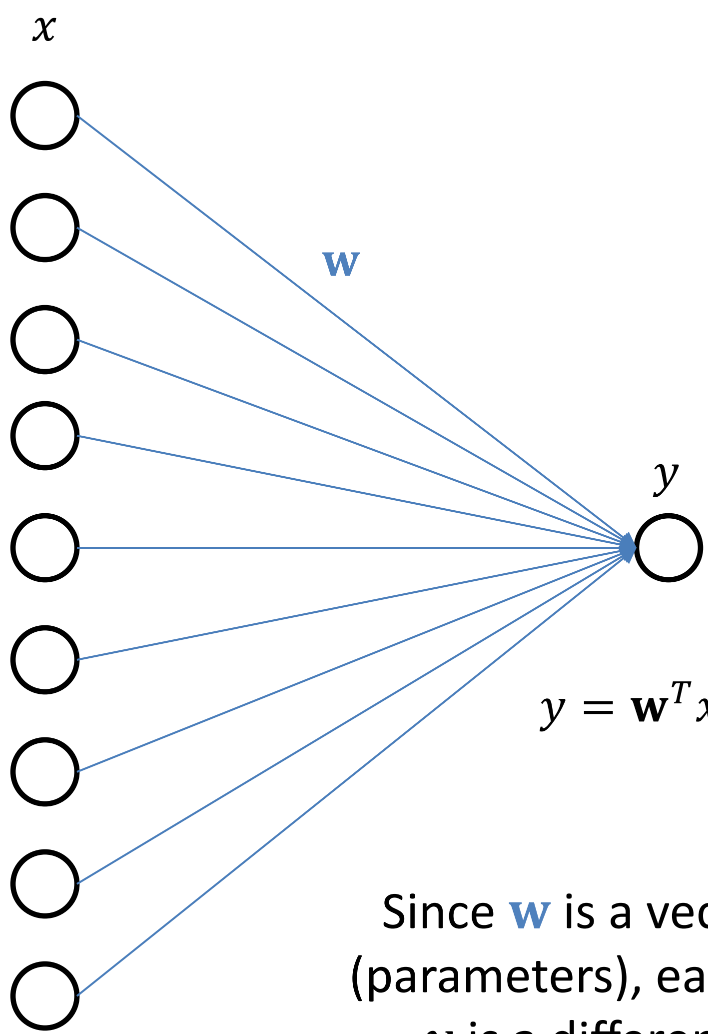
Items to learn: W, b

For regression: the output of this equation *is* the predicted value

For classification: one class is on one side of this line, the other class is on the other

Image courtesy Hamed Pirsiavash

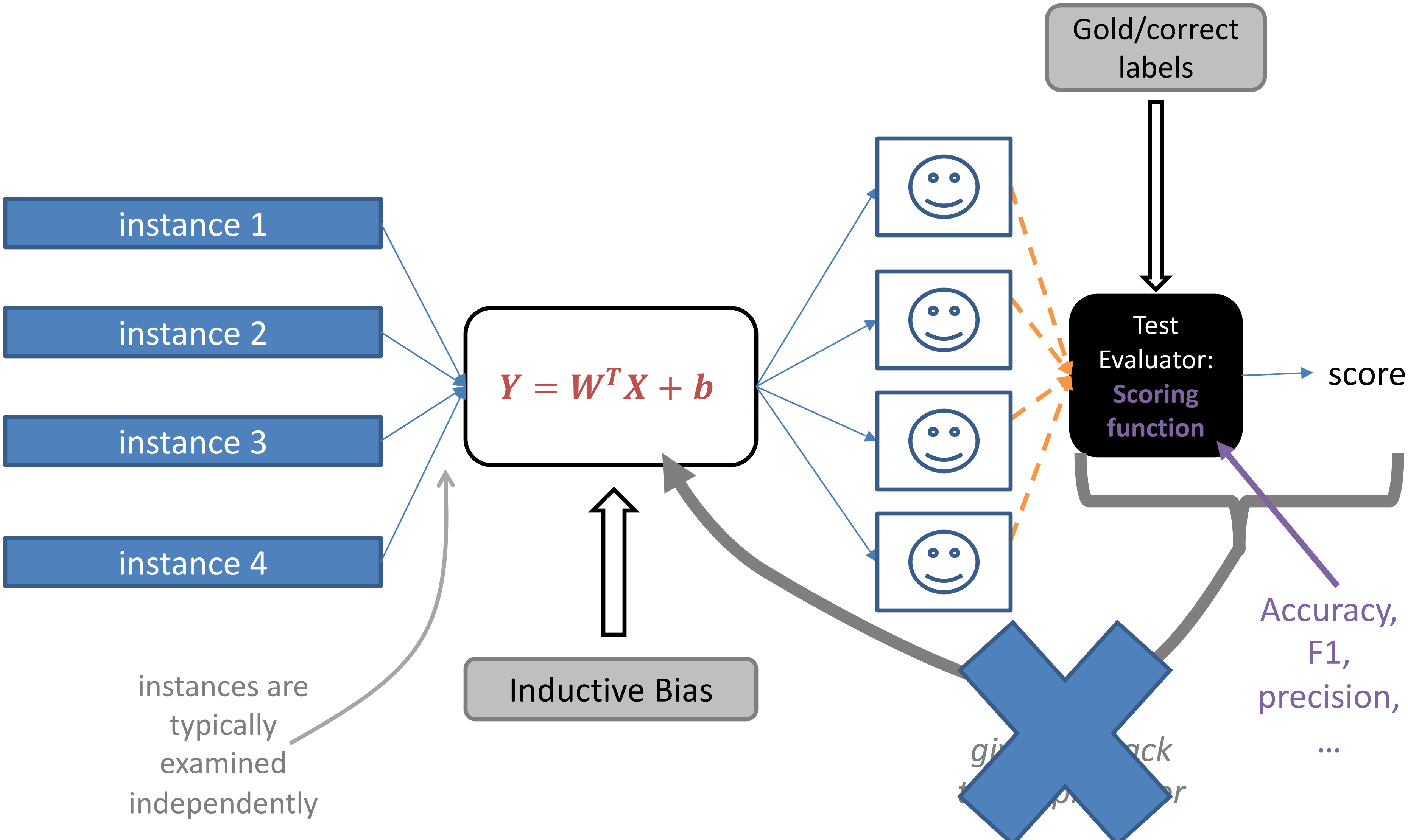
A Graphical View of Linear Models



$$y = \mathbf{w}^T x + b$$

Since $\mathbf{w}$ is a vector of weights (parameters), each arc from x to y is a different parameter

How do we evaluate these linear classification methods? Change the eval function.

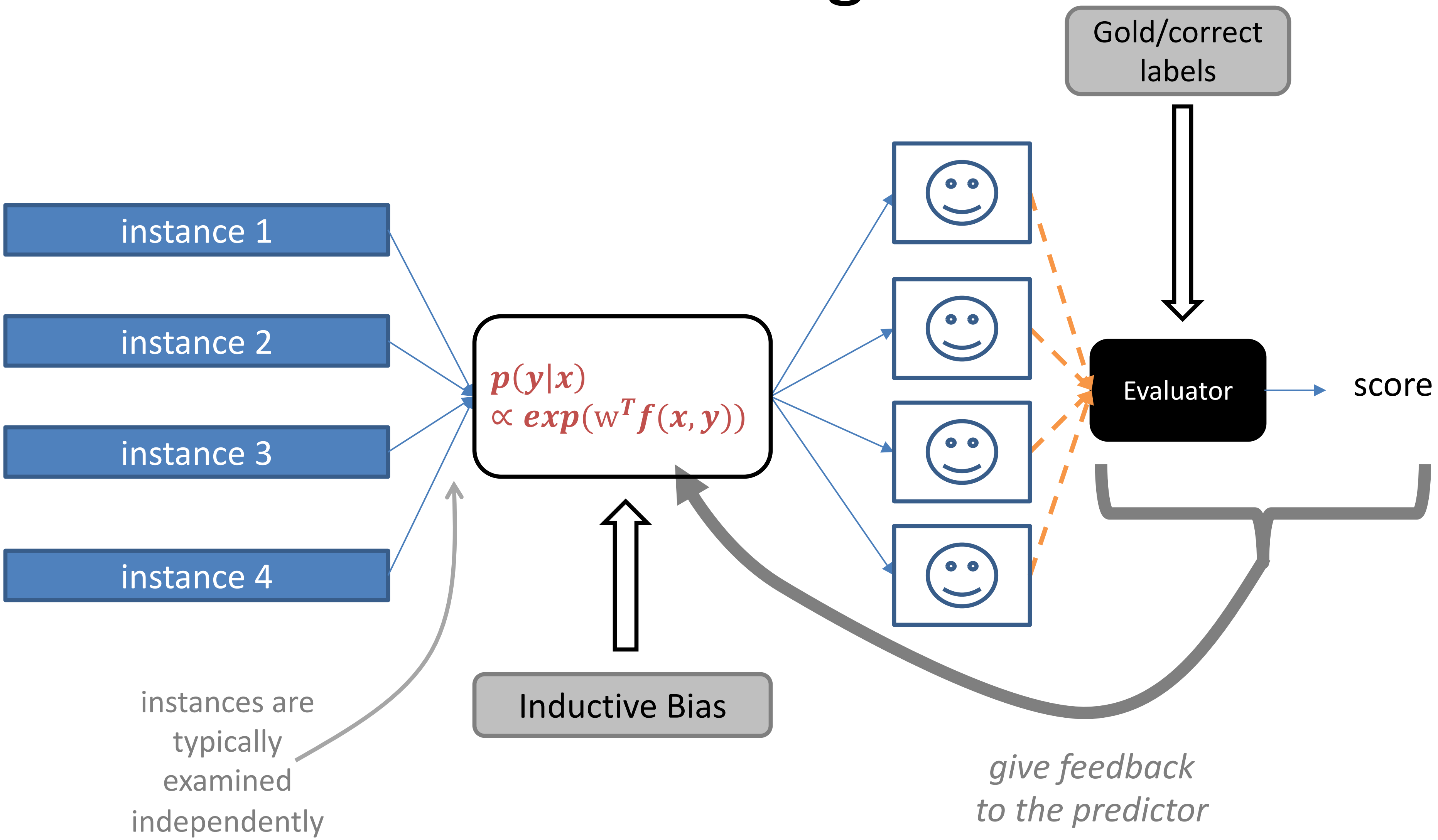


Core Aspects to Maxent Classifier p(y|x)

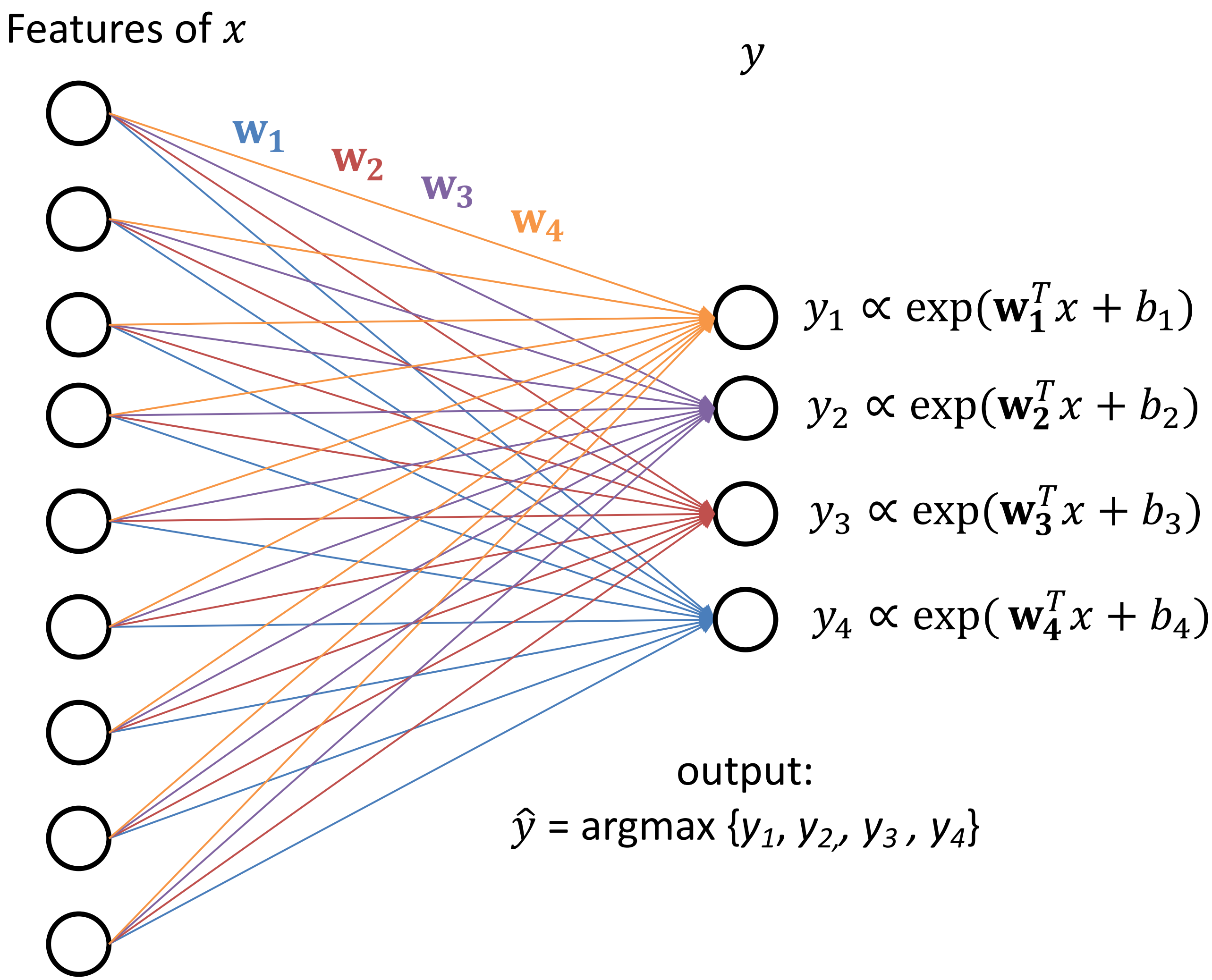
- **features** $f(x, y)$ between x and y that are meaningful;
- **weights** w (one per feature) to say how important each feature is; and
- a way to **form probabilities** from f and w

$$p(y \mid x) = \frac{\exp(\mathrm{w}^T f(x, y))}{\sum_{y'} \exp(\mathrm{w}^T f(x, y'))}$$

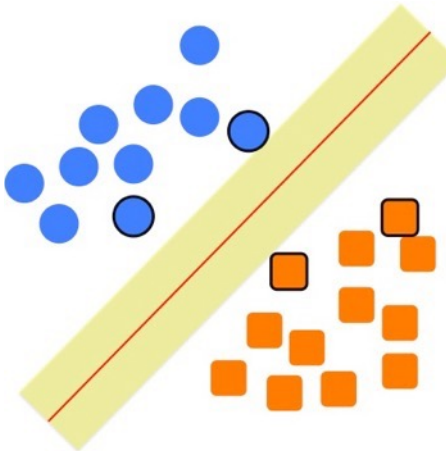
Machine Learning Framework: Learning



A Graphical View of Logistic Regression/Classification (4 classes)



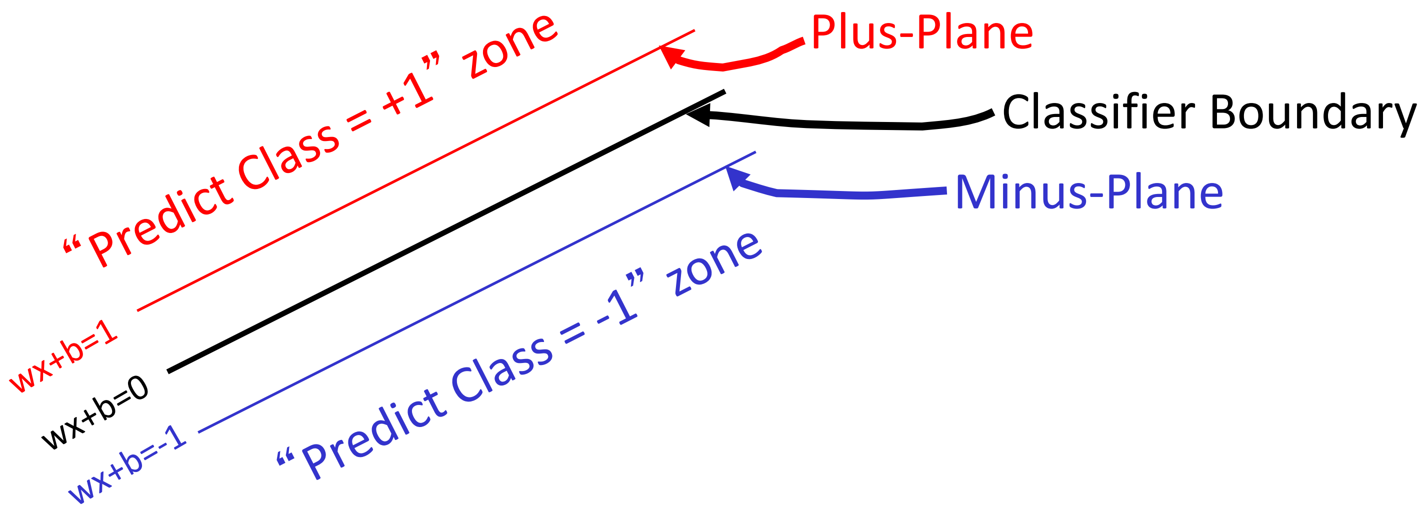
output:
$\hat{y}$ = argmax $\{y_1, y_2, y_3, y_4\}$



Support Vector Machines

Some slides borrowed from Andrew Moore's [slides on SVM]s.

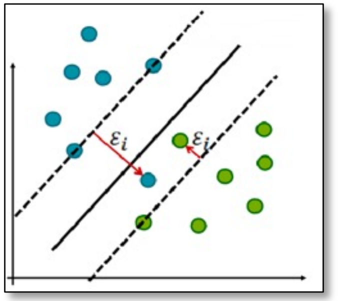
Specifying a line and margin



- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$

Classify as..	+1	if	$\mathbf{w} \cdot \mathbf{x} + b >= 1$
	-1	if	$\mathbf{w} \cdot \mathbf{x} + b <= -1$
	Universe explodes	if	$-1 < \mathbf{w} \cdot \mathbf{x} + b < 1$

Soft margin classification



- What if data from two classes not linearly separable?
- Allow a fat decision margin to make a few mistakes
- Some points, **outliers** or noisy examples, are inside or on wrong side of the margin
- Each outlier incurs a cost based on distance to hyperplane

Kernel Trick example

Can't separate the blue & red points with a line

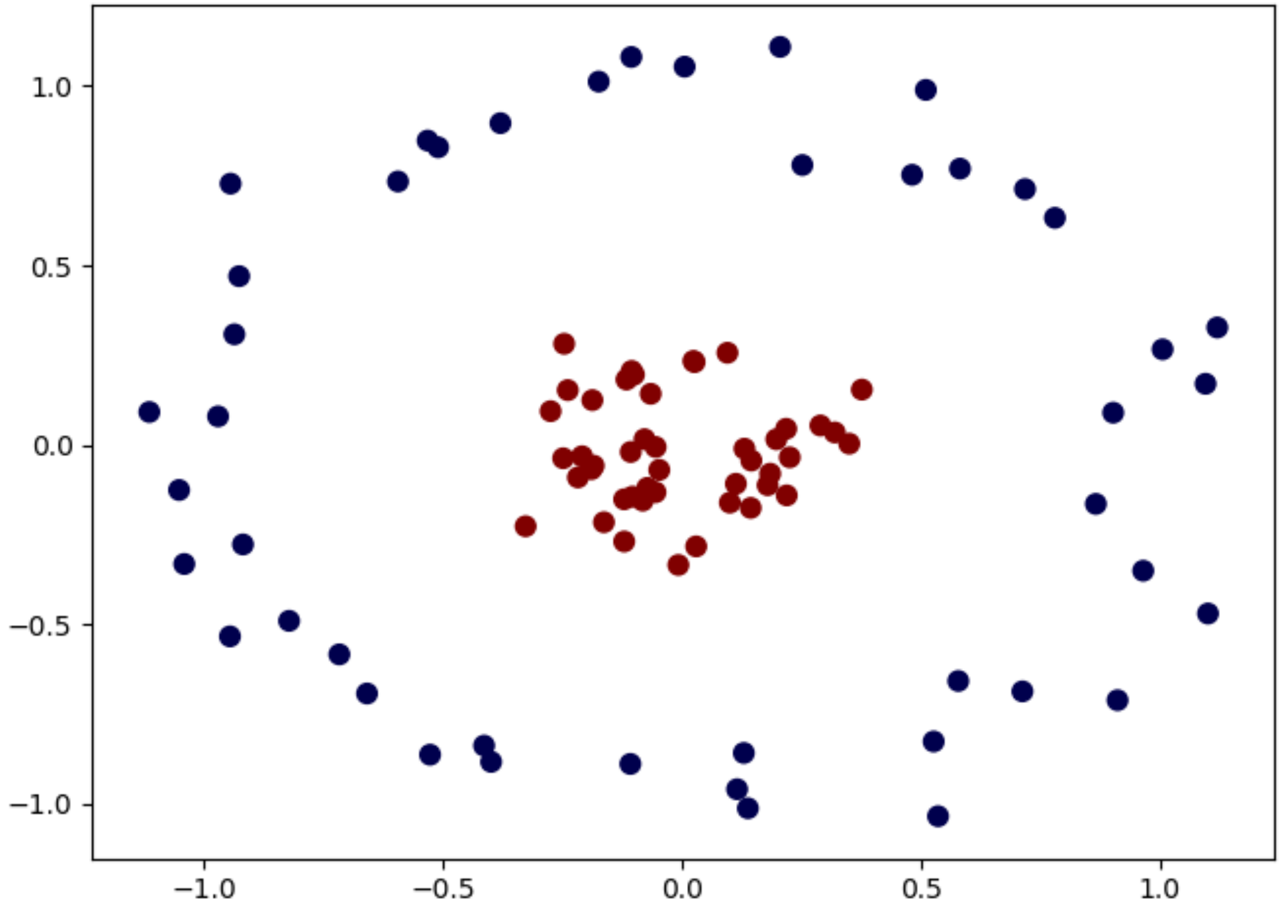
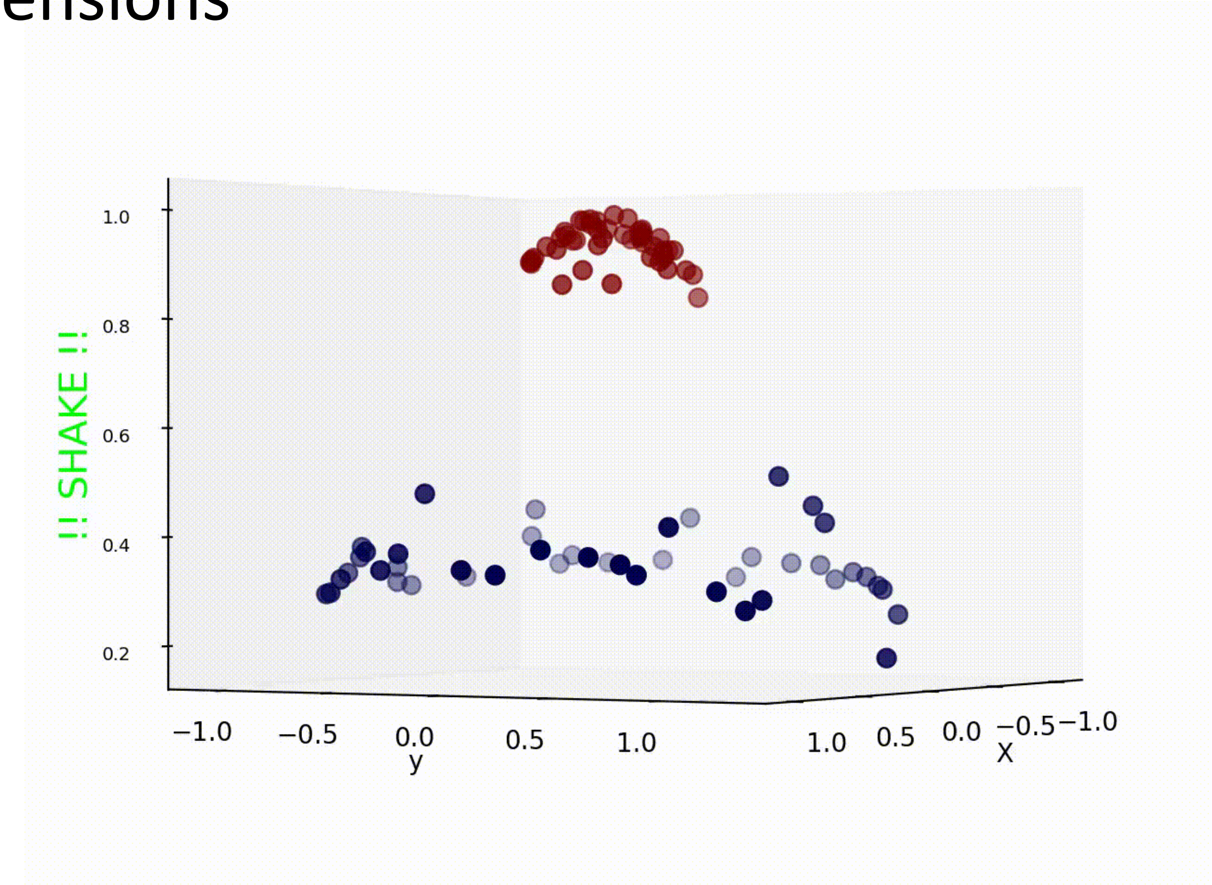


Figure from [S. Bhattacharyya]

Use a different kernel

- Applying a kernel can transform data to make it more nearly linearly separable
- E.g., use polar coordinates or map to three dimensions



Binary vs. multi classification

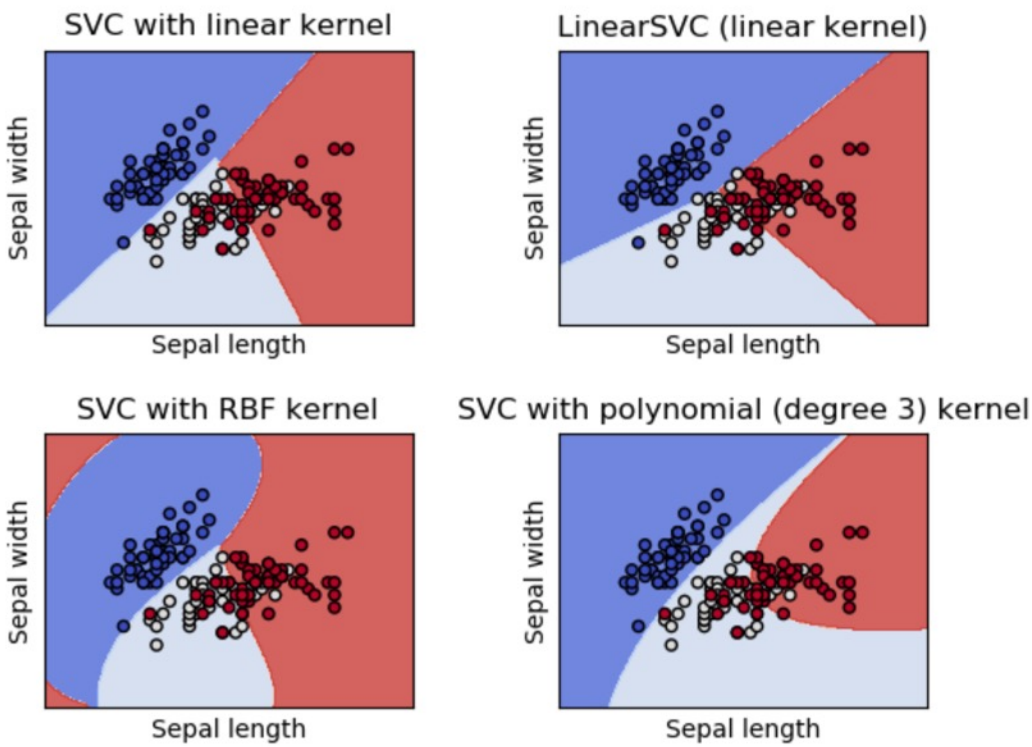
- SVMs only do **binary** classification ☹
  - E.g.: can't classify an iris into one of three species
- A common constraint for many ML classifiers
- Two approaches to [multiclass classification](): OVA and OVO
- Consider Zoo dataset, which classifies animals into one of 7 classes based on 17 attributes
  - **Classes:** mammal, bird, reptile, fish, amphibian, insect, invertebrate
  - **Attributes:** hair, feathers, eggs, milk, aquatic, toothed, fins, …

OVA or one-vs-all classification

- **OVA** or one-vs-all: turn n-way classification into n binary classification tasks
  - Also know as one-vs-rest
- For zoo problem with 7 categories, train and run 7 binary classifiers:
  - mammal vs. not-mammal
  - fish vs. not-fish
  - bird vs. not-bird, …
- Pick the one that gives the highest score
  - For an SVM this could be measured the one with the **widest margin**

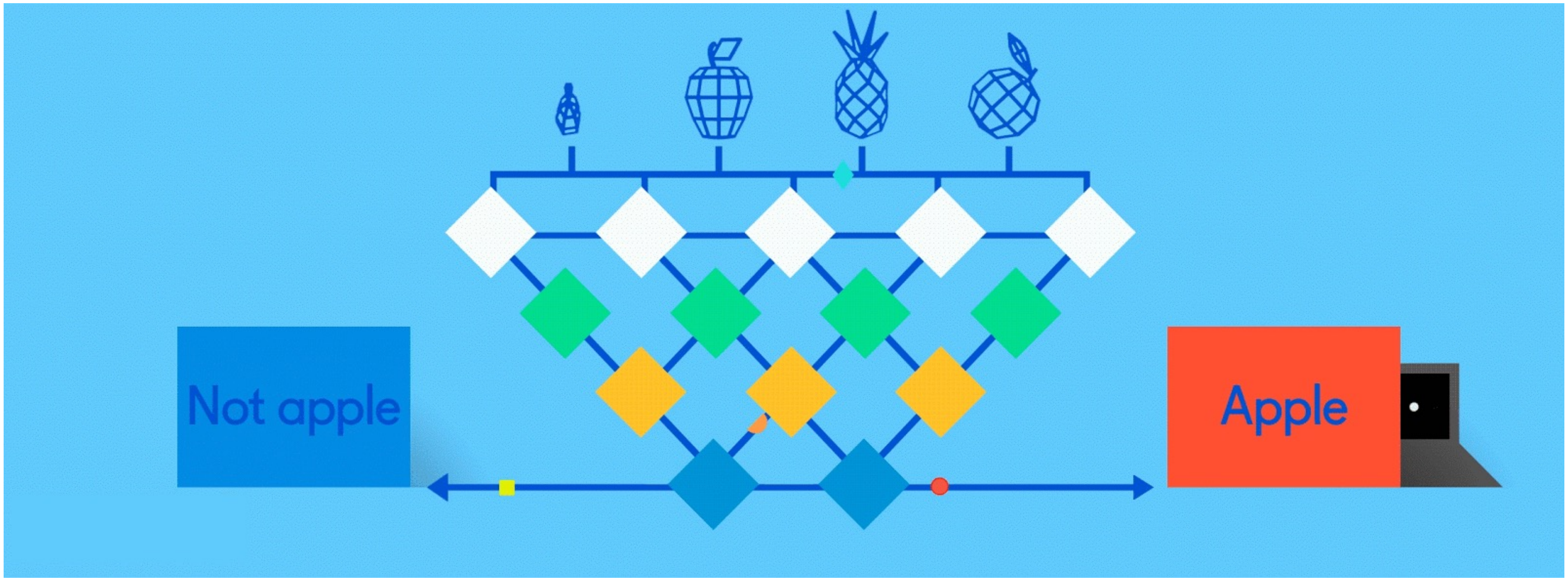
SVMs in scikit-learn

- Scikit-learn has three SVM classifiers: SVC, NuSVC, and LinearSVC
- Data can be either in **dense numpy** arrays or **sparse scipy** arrays
- All directly support multi-way classification, SVC and NuSCV using OvO and LinearSVC using OvA

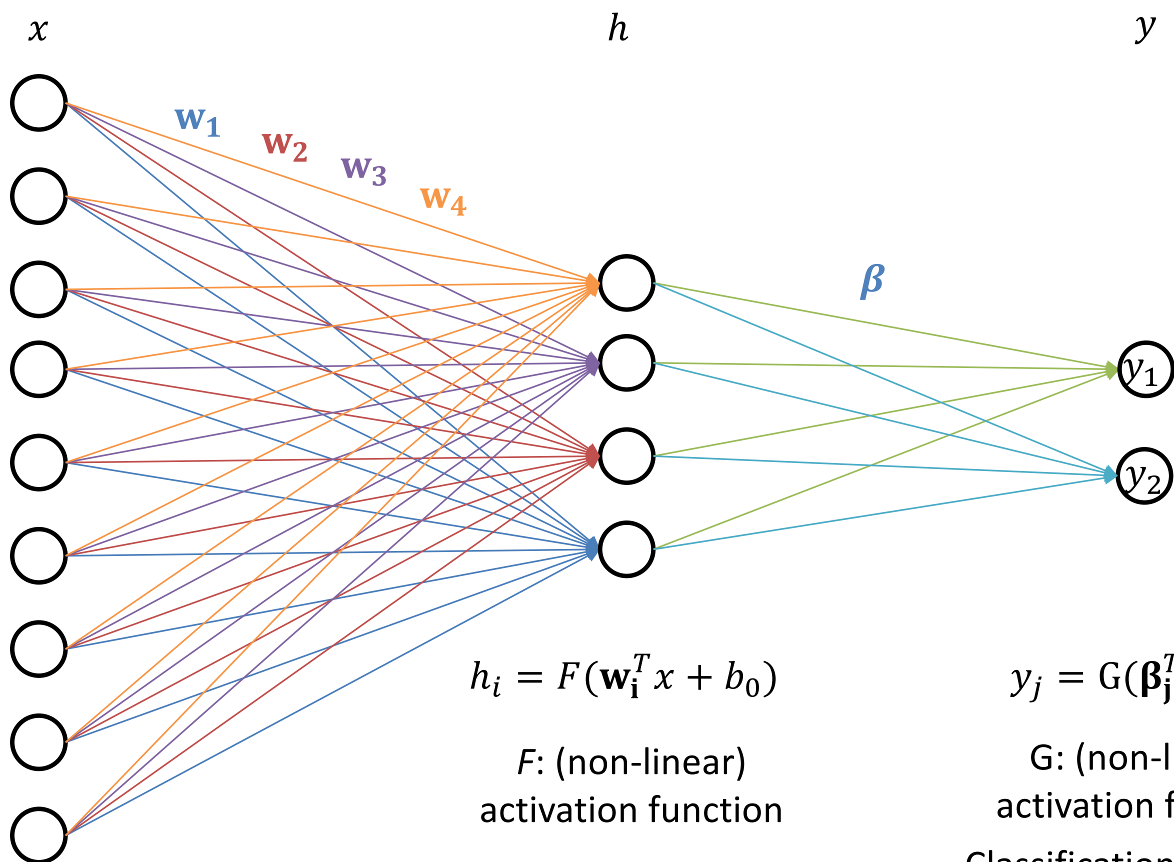


This colab jupyter notebook gets an accuracy of 97% using OvO with scikit.svm.SVC

Neural Networks for Machine Learning



Multilayer Perceptron, a.k.a. Feed-Forward Neural Network



$$h_i = F(\mathbf{w}_\mathbf{i}^T x + b_0)$$

F: (non-linear) activation function

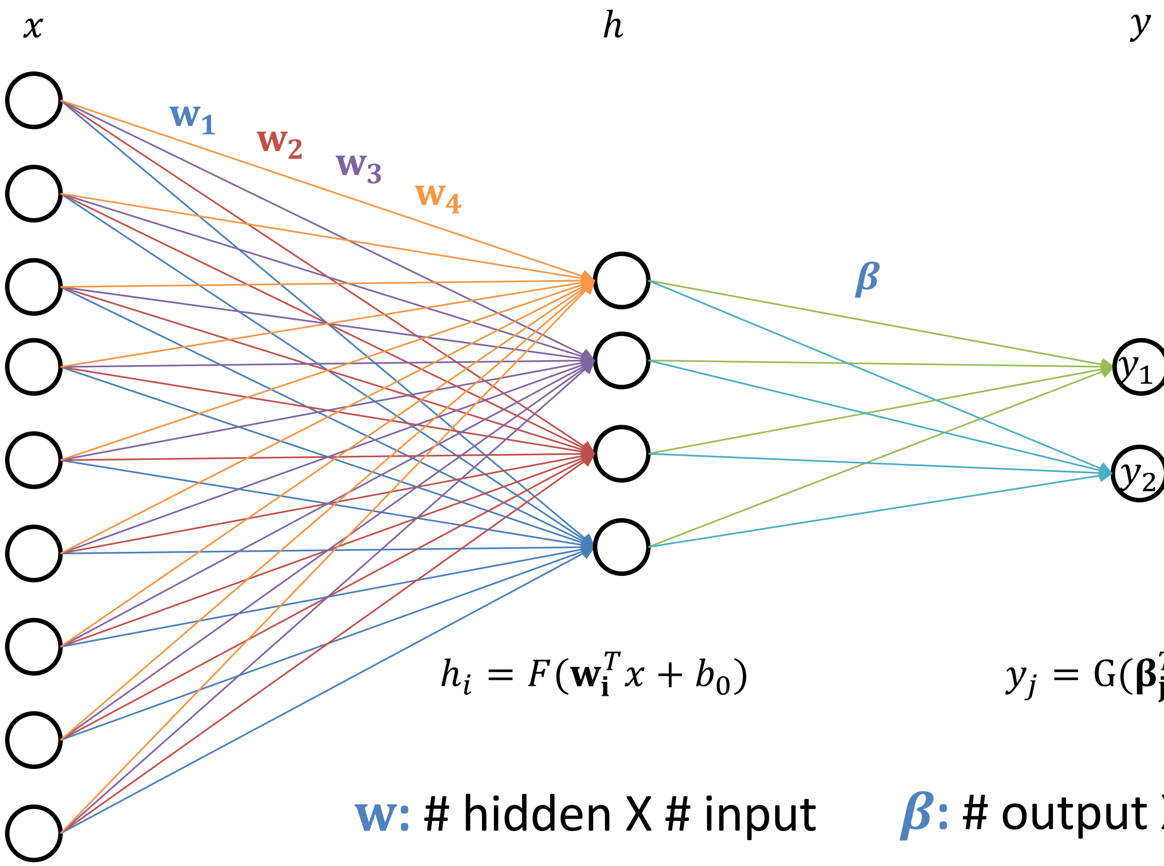
$$y_j = \mathrm{G}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$$

G: (non-linear) activation function

Classification: softmax

Regression: identity

Feed-Forward Neural Network

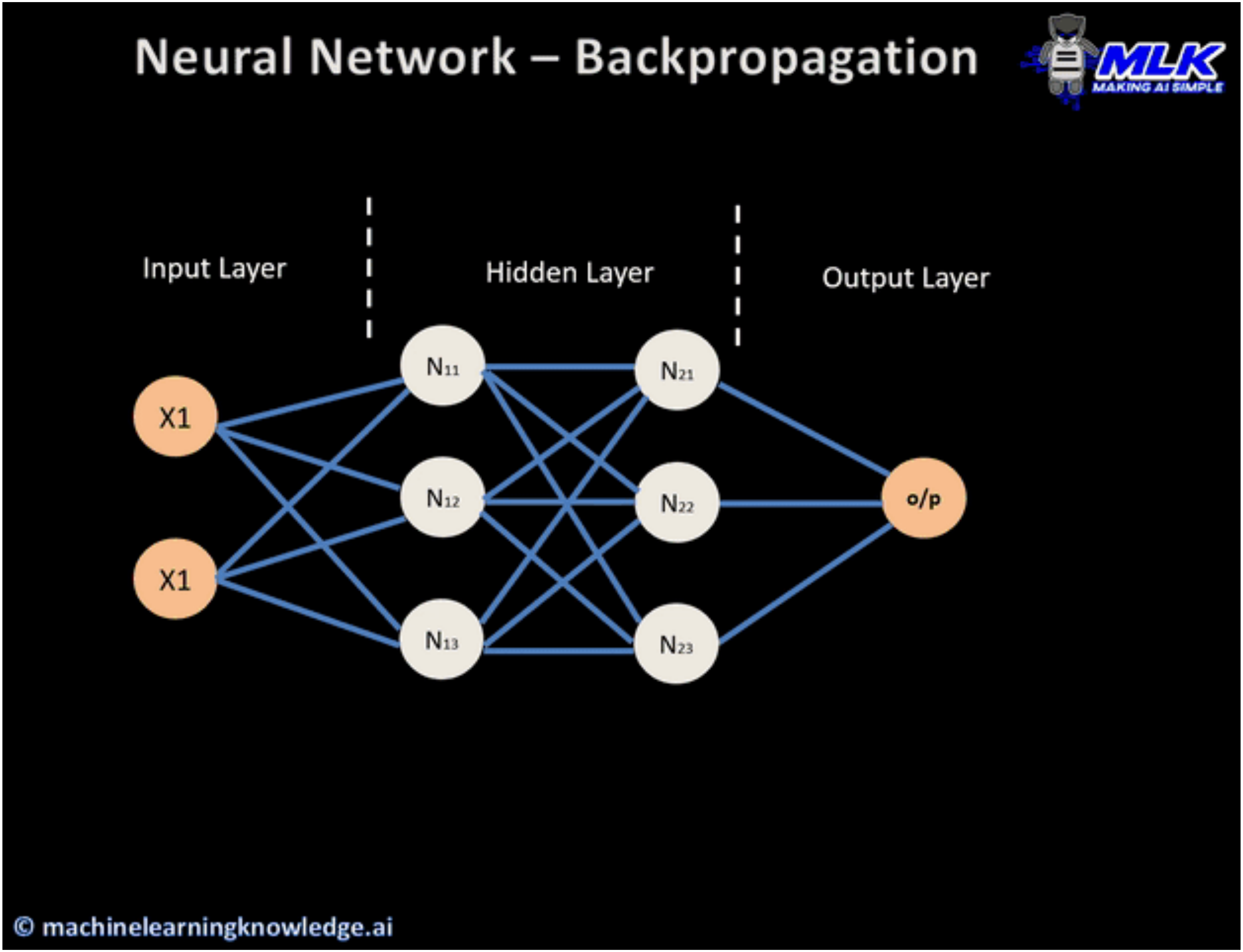


$h_i = F(\mathbf{w_i}^T x + b_0)$

$y_j = \mathrm{G}(\boldsymbol{\beta}_\mathbf{j}^T h + b_1)$

$\mathbf{w}$: # hidden X # input

$\boldsymbol{\beta}$: # output X # hidden


Neural Network – Backpropagation
MLK
MAKING AI SIMPLE
Input Layer
Hidden Layer
Output Layer
X1
X1
N11
N12
N13
N21
N22
N23
o/p
© machinelearningknowledge.ai

Universal Function Approximator

Theorem [Kurt Hornik et al., 1989]: Let F be a continuous function on a bounded subset of D-dimensional space. Then there exists a two-layer network G with finite number of hidden units that approximates F arbitrarily well. For all x in the domain of F, $|F(x) – G(x)| < \varepsilon$

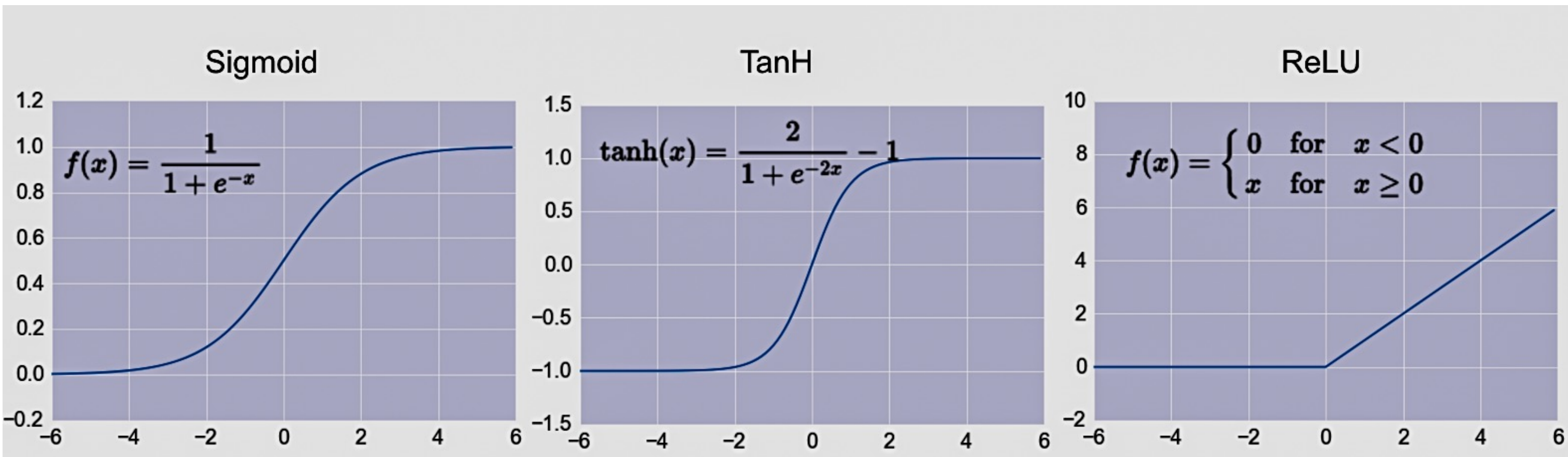
“a two-layer network can approximate any function”

Going from one to two layers dramatically improves the representation power of the network

Slide courtesy Hamed Pirsiavash

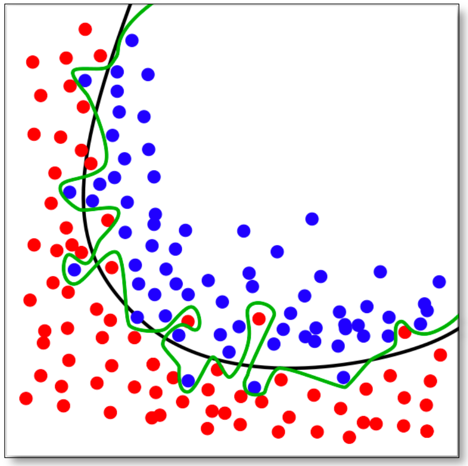
Common Activation Functions

- Define the output of a node given an input
- Very simple functions!



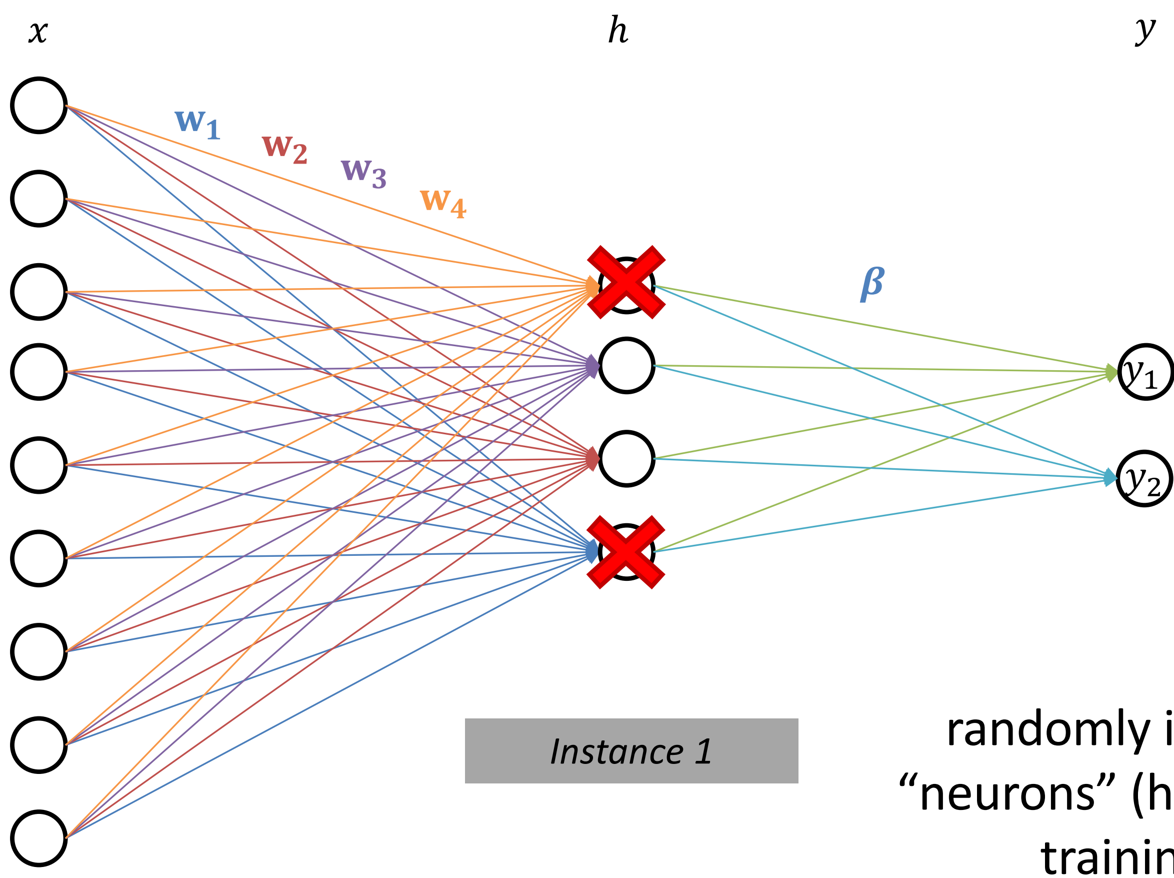
- Choice of activation function depends on problem and available computational power
- Comprehensive list of activation functions
- In practice

Regularization



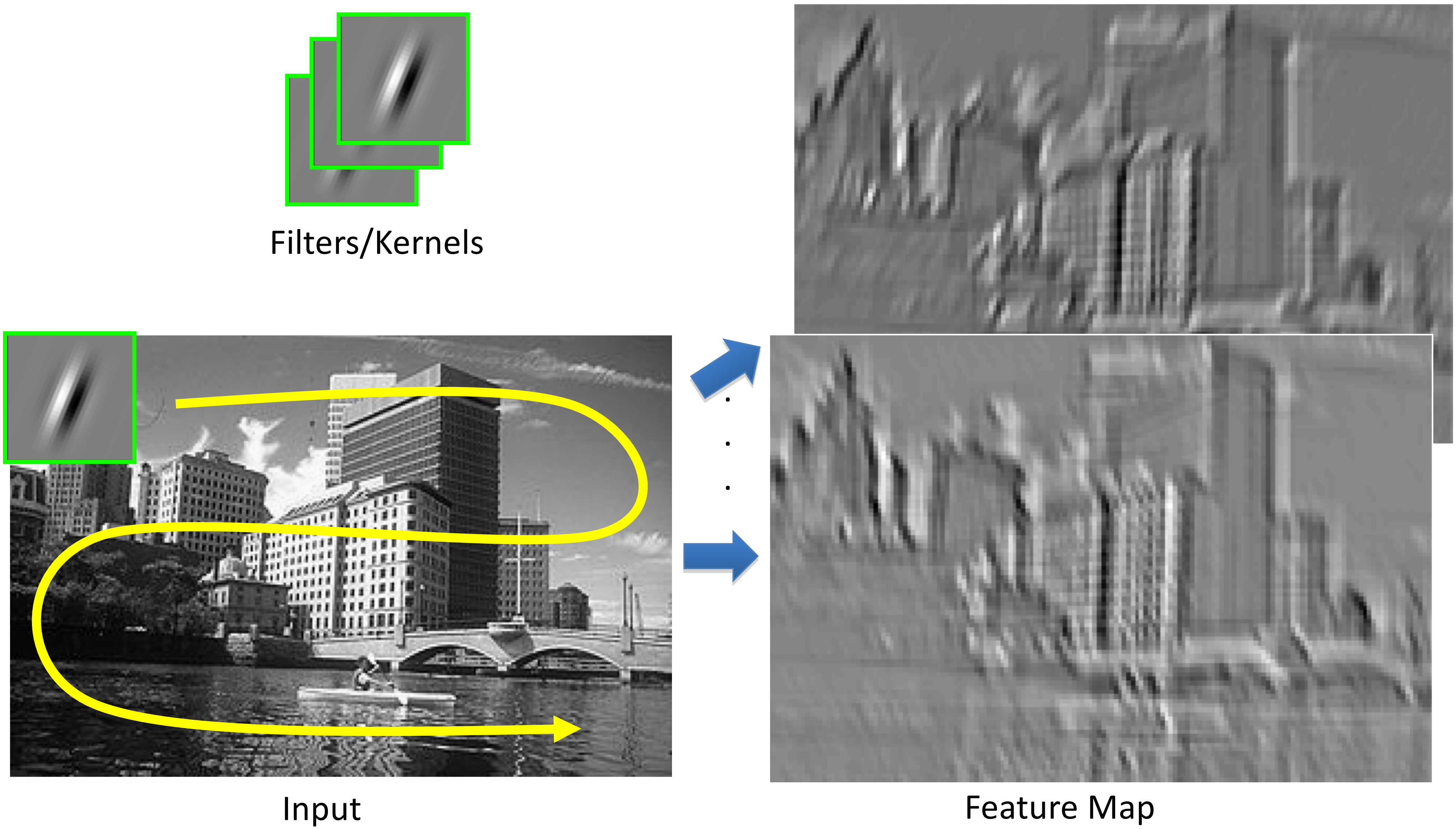
- Parameter to control overfitting, i.e. when the model does well on training data but poorly on new, unseen data
- L2 regularization is the most common
- Using dropout is another common way of reducing overfitting in neural networks
  - At each training stage, some nodes in hidden layer temporarily removed (dropped out)

Dropout: Regularization in Neural Networks



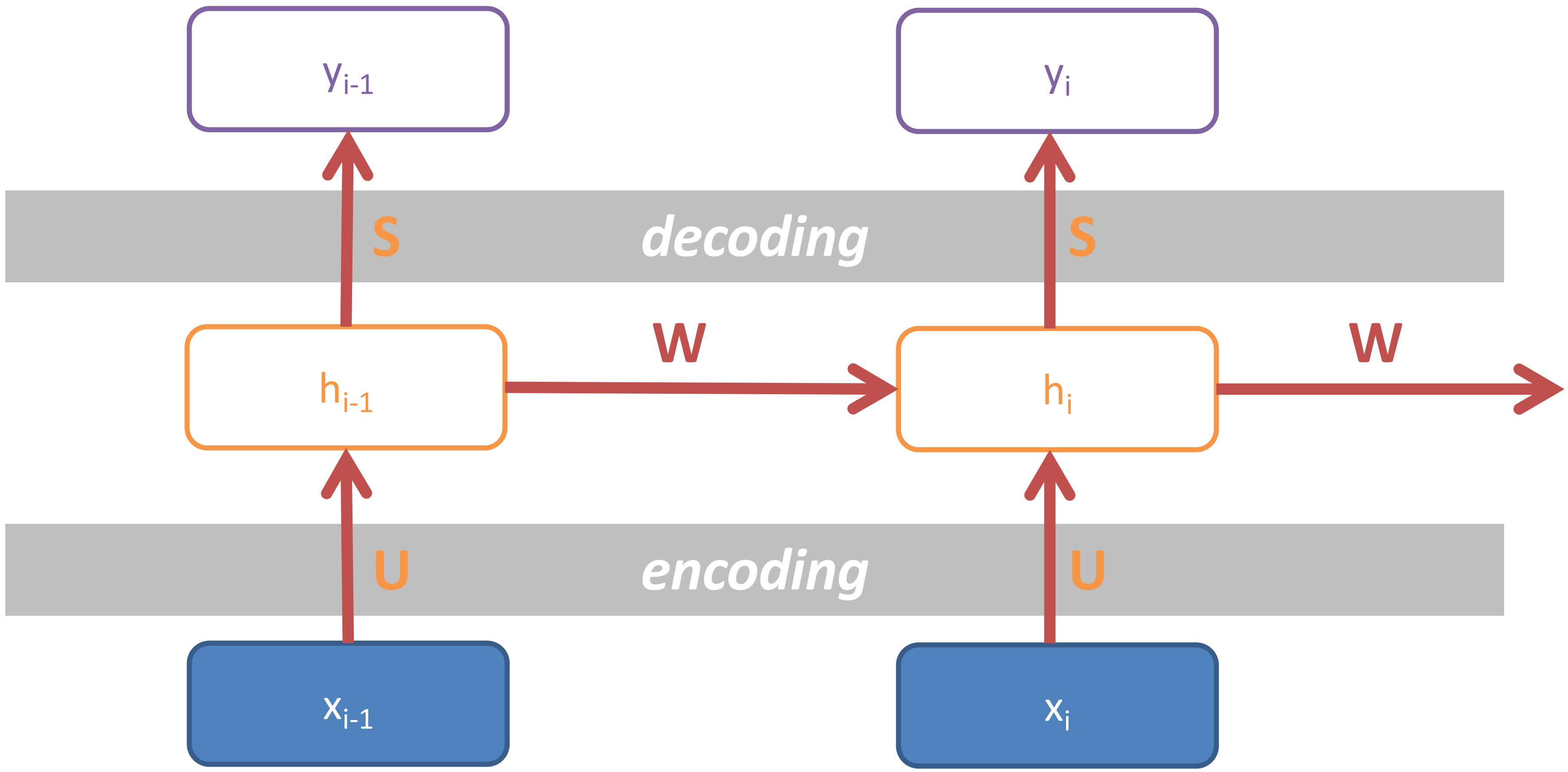
randomly ignore "neurons" (h_i) during training

Convolution as feature extraction



Slide credit: Svetlana Lazebnik

A *Simple* Recurrent Neural Network Cell



$$h_i = \tanh(Wh_{i-1} + Ux_i)$$

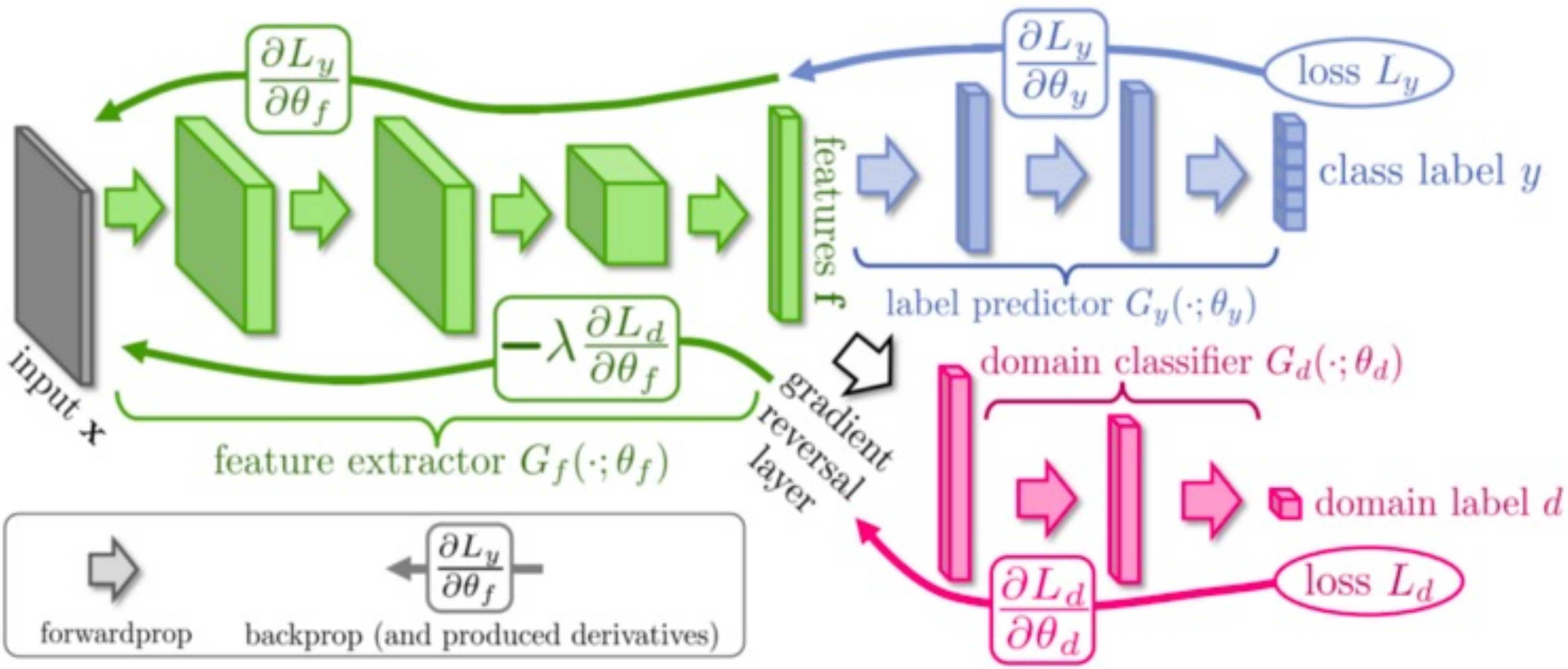
$$y_i = \mathrm{softmax}(Sh_i)$$

Weights are shared over time

unrolling/unfolding: copy the RNN cell across time (inputs)

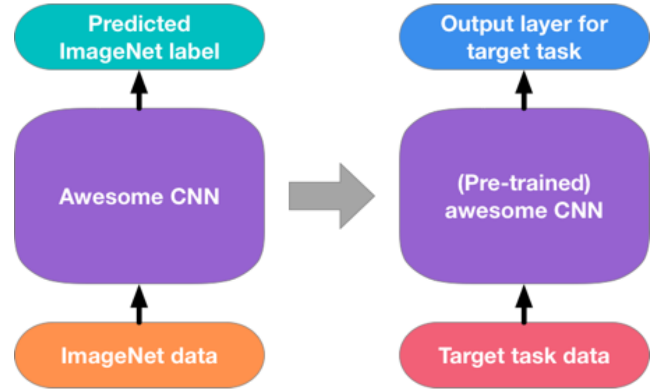
Good at Transfer Learning

- For images, the initial stages of a model learn high-level visual features (lines, edges) from pixels
- Final stages predict task-specific labels



source:http://ruder.io/transfer-learning/

Fine Tuning a NN Model



- Special kind of transfer learning
  - Start with a pre-trained model
  - Replace last output layer(s) with a new one(s)
  - One option: Fix all but last layer by marking as trainable:false
- Retraining on new task and data very fast
  - Only the weights for the last layer(s) are adjusted
- Example
  - Start: NN to classify animal pix with 100s of categories
  - Finetune on new task: classify pix of 10 common pets

Evaluation Metrics

Classification

- Precision, Recall, F1
- Accuracy
- Log-loss
- ROC-AUC
- …

Regression

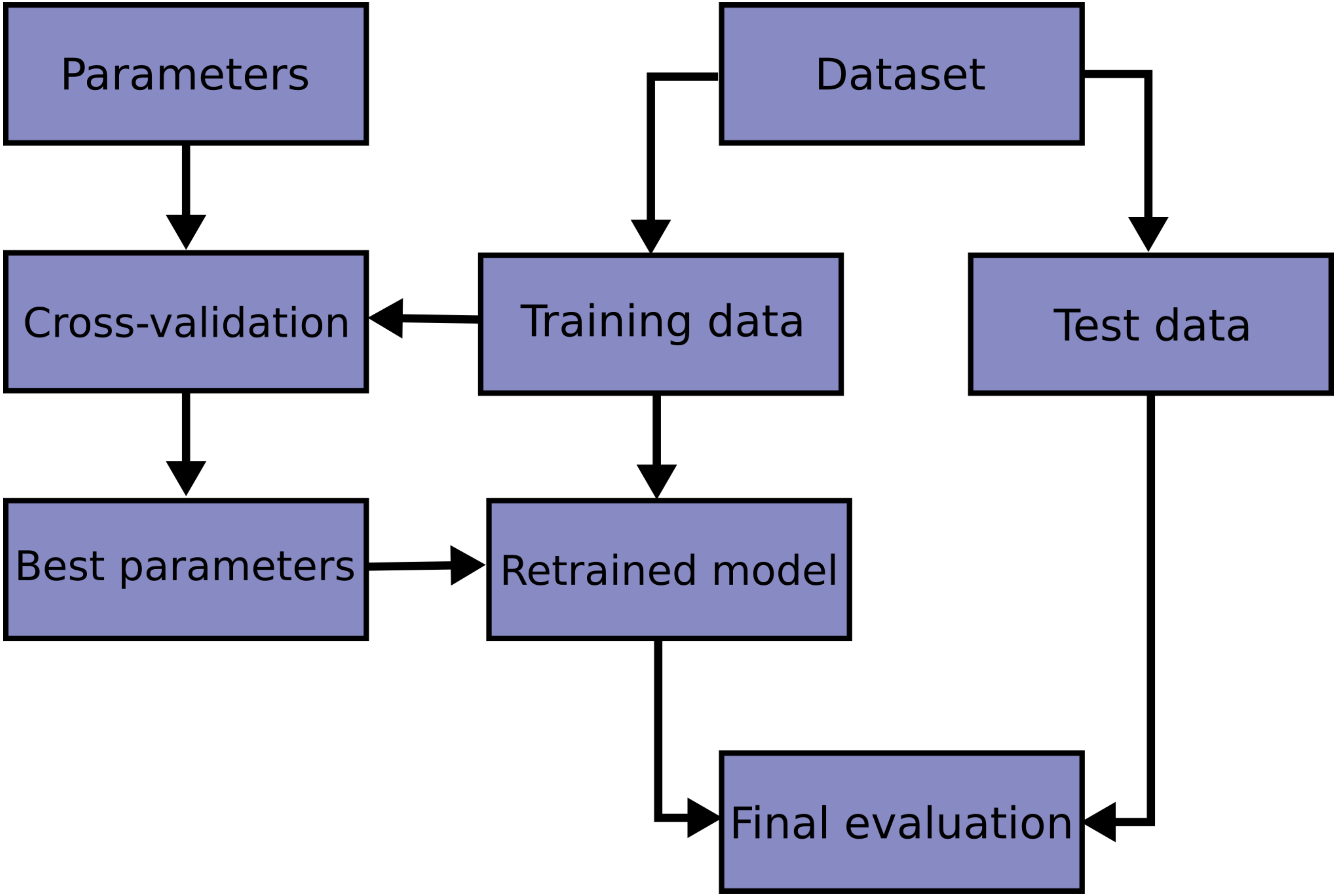
- (Root) Mean Square Error
- Mean Absolute Error
- …

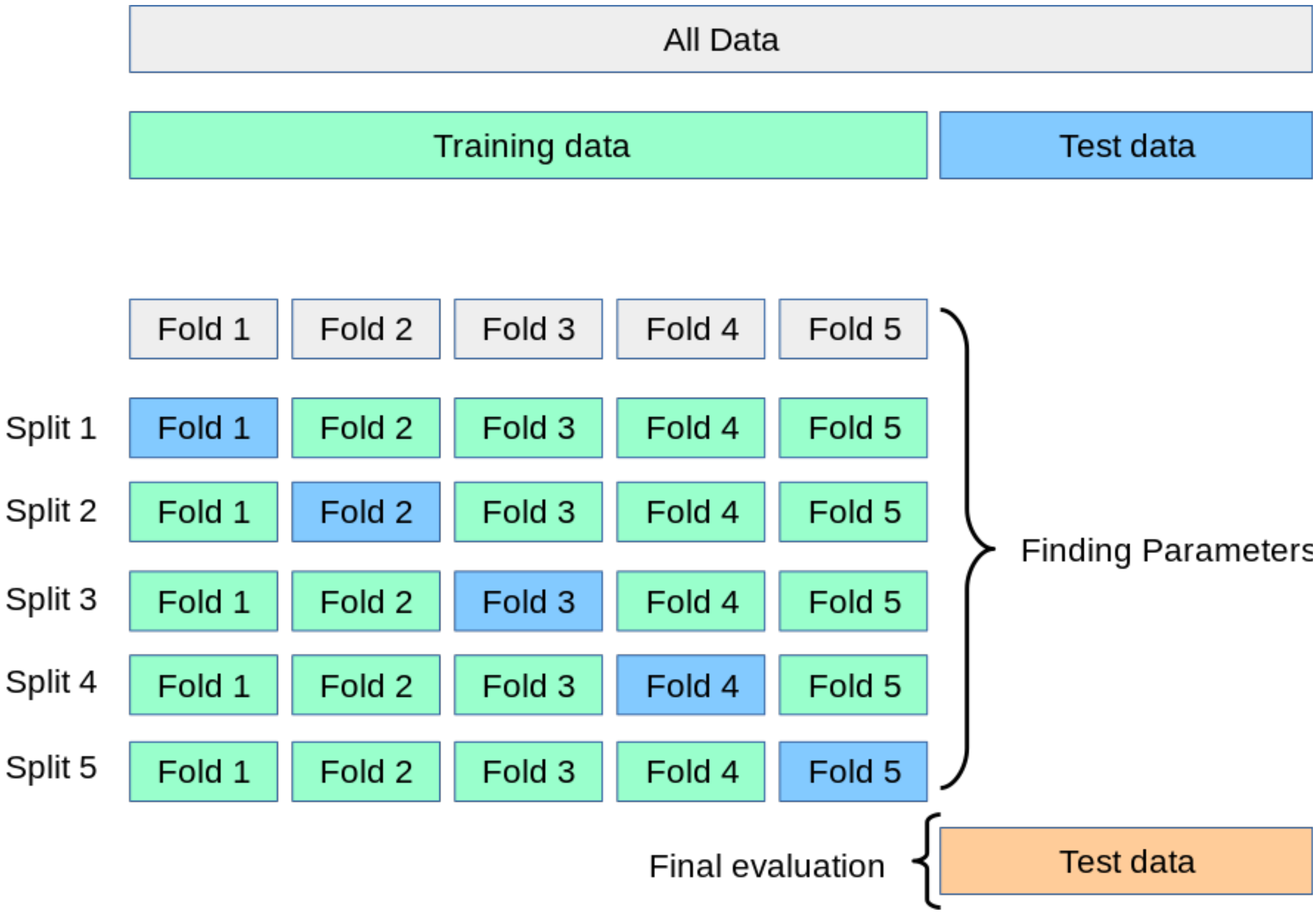
Clustering

- Mutual Information
- V-score
- …

the ***task****: what* kind *of problem are you solving?*

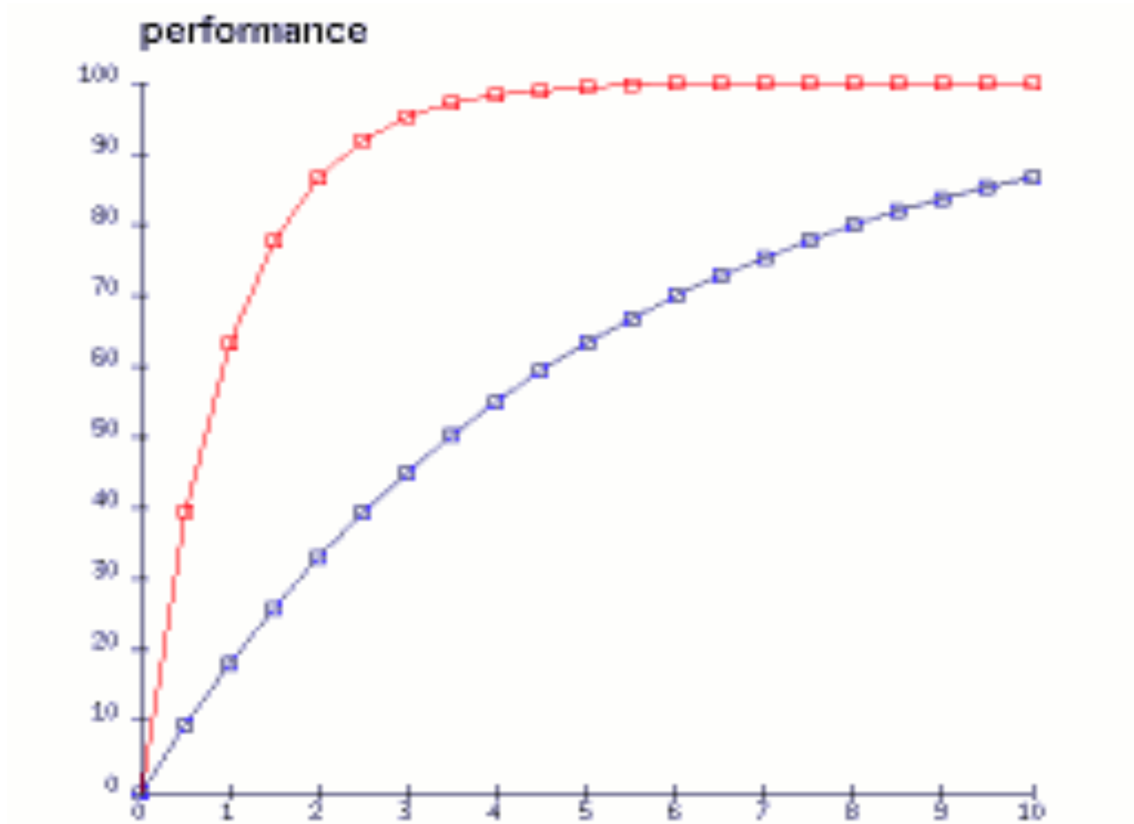
This does not have to be the same thing as the loss function you optimize


Parameters
Dataset
Cross-validation
Training data
Test data
Best parameters
Retrained model
Final evaluation


All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Split 2
Split 3
Split 4
Split 5
Finding Parameters
Final evaluation
Test data

Learning curve

- When evaluating ML algorithms, steeper learning curves are better
- They represents faster learning with less data



Training set size

Here the system with the red curve is better since it requires less data to achieve desired accuracy

A combined measure: F

Weighted (harmonic) average of **P**recision & **R**ecall

$$F = \frac{(1+\beta^2) \ * P \ * R}{(\beta^2 * P) + R}$$

Balanced F1 measure: β=1

$$F_1 = \frac{2 \ * P \ * R}{P + R}$$

P/R/F in a Multi-class Setting: Micro- vs. Macro-Averaging

If we have more than one class, how do we combine multiple performance measures into one quantity?

Macroaveraging: Compute performance for each class, then average.

Microaveraging: Collect decisions for all classes, compute contingency table, evaluate.

Micro- vs. Macro-Averaging: Example

Class 1

	Truth : yes	Truth : no
Classifier: yes	10	10
Classifier: no	10	970

Class 2

	Truth : yes	Truth : no
Classifier: yes	90	10
Classifier: no	10	890

Micro Ave. Table

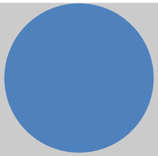
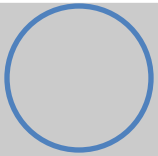
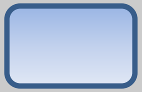
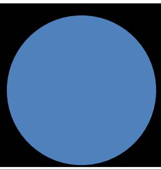
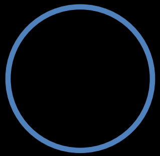
	Truth : yes	Truth : no
Classifier: yes	100 (90+10)	20 (10+10)
Classifier: no	20	1860

Macroaveraged precision: (10/10+10) + (90/90+10)/2 = (0.5 + 0.9)/2 = 0.7

Microaveraged precision: 100/100+20 = .83

Microaveraged score is dominated by score on frequent classes

Confusion Matrix: Generalizing the 2-by-2 contingency table

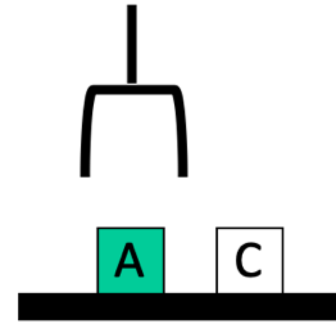
		Correct Value		
				
Guessed Value		#	#	#
		#	#	#
		#	#	#

Planning

Chapter 11.1-11.3

Some material adopted from notes by
Andreas Geyer-Schulz and Chuck Dyer

Blocks world



Typical representation uses a logic notation to represent the state of the world:

ontable(a) ontable(c)
clear(a) clear(c)
handempty

And possible **actions/ operators** with their preconditions and effects:

Pickup Putdown
Stack Unstack

Planning vs. problem solving

- Problem solving methods solve similar problems
- Planning is more powerful and efficient because of the representations and methods used
- States, goals, and actions are decomposed into sets of sentences (usually in first-order logic)
- Search often proceeds through *plan space* rather than *state space* (though there are also state-space planners)
- Sub-goals can be planned independently, reducing the complexity of the planning problem

Blocks World Operators

- Classic basic **operations** for the Blocks World
  - **stack(X,Y):** put block X on block Y
  - **unstack(X,Y):** remove block X from block Y
  - **pickup(X):** pickup block X
  - **putdown(X):** put block X on the table
- Each represented by
  - list of **preconditions**
  - list of new facts to be added (**add-effects**)
  - list of facts to be removed (**delete-effects**)
  - optionally, set of (simple) variable **constraints**

Blocks World Stack Action

stack(X,Y):

- **preconditions**(stack(X,Y), [holding(X), clear(Y)])
- **adds**(stack(X,Y), [handempty, on(X,Y), clear(X)])
- **deletes**(stack(X,Y), [holding(X), clear(Y)]).
- **constraints**(stack(X,Y), [X≠Y, Y≠table, X≠table])

STRIPS planning

- STRIPS maintains two additional data structures:
  - State List - all currently true predicates.
  - Goal Stack - push down stack of goals to be solved, with current goal on top
- If current goal not satisfied by present state, find action that adds it and push action and its preconditions (subgoals) on stack
- When a current goal is satisfied, POP from stack
- When an action is on top stack, record its application on plan sequence and use its add and delete lists to update current state

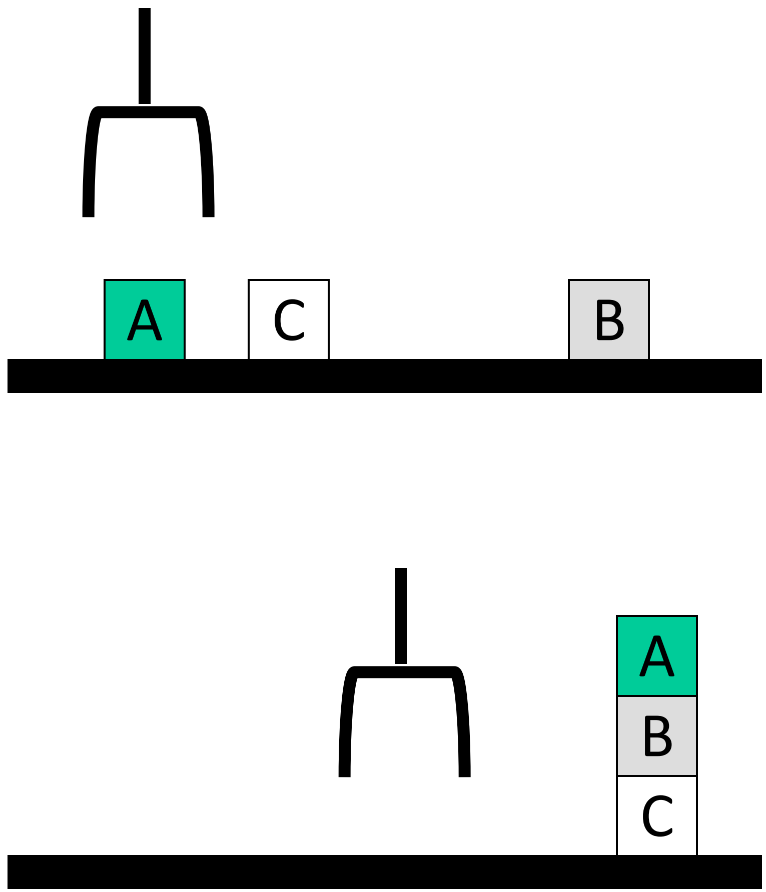
Another BW planning problem

Initial state:

clear(a)
clear(b)
clear(c)
ontable(a)
ontable(b)
ontable(c)
handempty

Goal:

on(a,b)
on(b,c)
ontable(c)

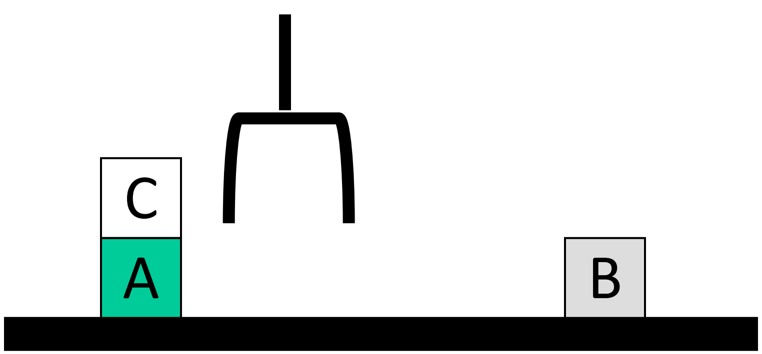


A plan:

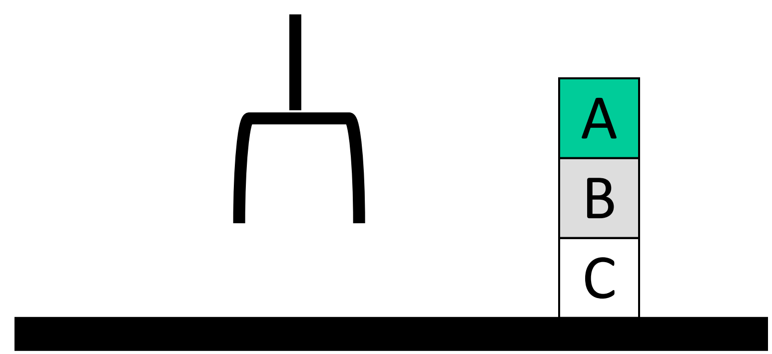
pickup(a)
stack(a,b)
unstack(a,b)
putdown(a)
pickup(b)
stack(b,c)
pickup(a)
stack(a,b)

Goal interaction

- Simple planning algorithms assume independent sub-goals
  - Solve each separately and concatenate the solutions
- Sussman Anomaly: an example of goal interaction problem:
  - Solving on(A,B) first (via unstack(C,A),stack(A,B)) is undone when solving 2nd goal on(B,C) (via unstack(A,B), stack(B,C))
  - Solving on(B,C) first will be undone when solving on(A,B)
- Classic STRIPS couldn't handle this, although minor modifications can get it to do simple cases



Initial state



Goal state

State-Space Planning

- STRIPS searches thru a space of situations (where you are, what you have, etc.)
- Find plan by searching **situations** to reach goal
- **Progression planner**: searches forward
  - From initial state to goal state
  - Prone to exploring irrelevant actions
- **Regression planner**: searches backward from goal
  - Works **iff** operators have enough information to go both ways
  - Ideally leads to reduced branching: planner is only considering things that are relevant to the goal
  - but it's harder to define good heuristics – so most current systems favor forward search

CMSC 471

Propositional and First-Order Logic

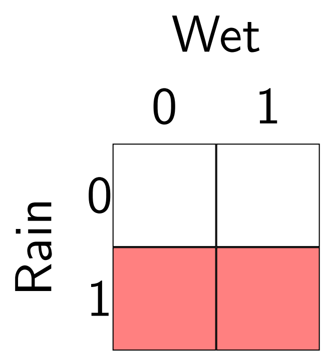
KMA Solaiman

ksolaima@umbc.edu

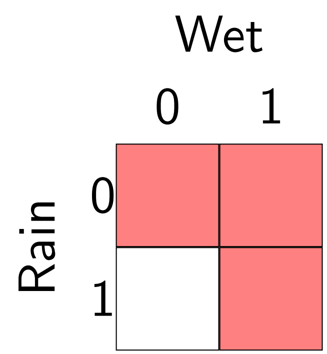
Some slides courtesy Tim Finin

Knowledge base: example

$\mathcal{M}(\text{Rain})$

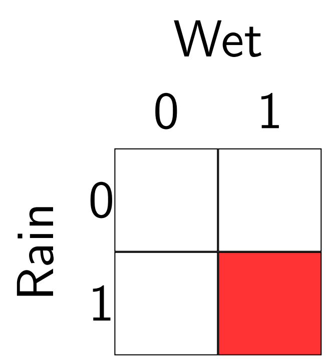


$\mathcal{M}(\text{Rain} \to \text{Wet})$



Intersection:

$\mathcal{M}(\{\text{Rain}, \text{Rain} \to \text{Wet}\})$



- As a concrete example, consider the two formulas Rain and Rain $\rightarrow$ Wet. If you know both of these facts, then the set of models is constrained to those where it is raining and wet.

Models for a KB

- KB: [P∨Q, P→R, Q→R]
- What are the formulas?

  f1: P∨Q

  f2: P→R

  f3: Q→R
- What are the propositional variables?

  P, Q, R
- What are the candidate models?

1) Consider all **eight** possible assignments of T|F to P, Q, R
2) Check if each sentence is consistent with the model

P	Q	R	s1	s2	s3
F	F	F	x	✓	✓
F	F	T	x	✓	✓
F	T	F	✓	✓	x
F	T	T	✓	✓	✓
T	F	F	✓	x	✓
T	F	T	✓	✓	✓
T	T	F	✓	x	x
T	T	T	✓	✓	✓

Here **x** means the model makes the sentence False and ✓ means it doesn't make it False

Models for a KB

- KB: [P∨Q, P→R, Q→R]
- What are the formulas?
  - f1: P∨Q
  - f2: P→R
  - f3: Q→R
- What are the propositional variables?
  - P, Q, R
- What are the candidate models?
  1) Consider all **eight** possible assignments of T|F to P, Q, R
  2) Check truth tables for consistency, eliminating any row that does not make every KB sentence true

P	Q	R	s1	s2	s3
~~F~~	~~F~~	~~F~~	~~x~~	~~✓~~	~~✓~~
~~F~~	~~F~~	~~T~~	~~x~~	~~✓~~	~~✓~~
~~F~~	~~T~~	~~F~~	~~✓~~	~~✓~~	~~x~~
F	**T**	**T**	✓	✓	✓
~~T~~	~~F~~	~~F~~	~~✓~~	~~x~~	~~✓~~
T	**F**	**T**	✓	✓	✓
~~T~~	~~T~~	~~F~~	~~✓~~	~~x~~	~~x~~
T	**T**	**T**	✓	✓	✓

- Only 3 models are consistent with KB
- R true in all of them
- Therefore, R is true and can be added to the KB

A simple example

The KB

P

Q ∨ ¬ R

The KB has 2 formulas.

The KB has 3 variables.

The KB has 3 models for which $I(f, w) = 1$.

Another way to look at this is:
$\mathcal{M}\ (\boldsymbol{P})$ is true in first 3
$\mathcal{M}$ (**Q ∨ ¬ R**)is true in first 3
So $\mathcal{M}\ (\boldsymbol{KB})$ is first 3

Models for the KB, $\mathcal{M}\ (\boldsymbol{KB})$

P	Q	R	KB
T	T	F	T
T	T	T	T
T	F	F	T
T	F	T	F
F	T	F	F
F	T	T	F
F	F	T	F
F	F	F	F

Another simple example

The KB

$P \wedge Q$

$R \wedge \neg P$

The KB has 2 formulas.

The KB has 3 variables.

Models for the KB

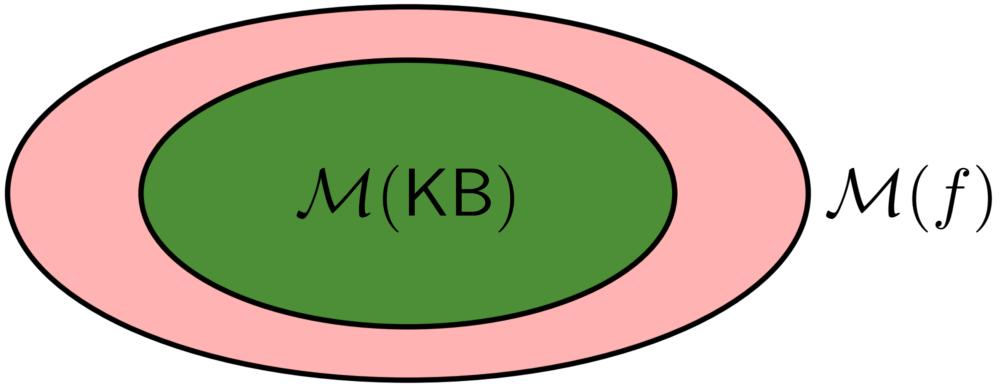
P	Q	R

The KB has no models. There is no assignment of True or False to every variable that makes every sentence in the KB true

Desiderata for inference rules

Semantics

Interpretation defines **entailed/true** formulas: KB $\models f$:



Syntax:

Inference rules **derive** formulas: KB $\vdash f$

How does $\{f : \text{KB} \models f\}$ relate to $\{f : \text{KB} \vdash f\}$?

- We can apply inference rules all day long, but now we desperately need some guidance on whether a set of inference rules is doing anything remotely sensible.
- For this, we turn to semantics, which gives an objective notion of truth. Recall that the semantics provides us with $\mathcal{M}$, the set of satisfiable models for each formula f or knowledge base. This defines a set of formulas $\{f : \text{KB} \models f\}$ which are defined to be true.
- On the other hand, inference rules also gives us a mechanism for generating a set of formulas, just by repeated application. This defines another set of formulas $\{f : \text{KB} \vdash f\}$.

Truth

$$\{f : \mathsf{KB} \models f\}$$

- Imagine a glass that represents the set of possible formulas entailed by the KB (these are necessarily true).
- By applying inference rules, we are filling up the glass with water.

Soundness

Definition: soundness

A set of inference rules Rules is sound if:

$$\{f : \mathsf{KB} \vdash f\} \subseteq \{f : \mathsf{KB} \models f\}$$

An inference rule is sound if every formula f it produces from a KB logically follows from the KB i.e., inference rule creates no contradictions

- We say that a set of inference rules is **sound** if using those inference rules, we never overflow the glass: the set of derived formulas is a subset of the set of true/entailed formulas.

Completeness

Definition: completeness

A set of inference rules Rules is complete if:

$$\{f : \text{KB} \vdash f\} \supseteq \{f : \text{KB} \models f\}$$

it can produce every formula that logically follows from (is entailed by) the KB
- Similar to complete search algorithms

CMSC 471: Reinforcement Learning

Some slides courtesy Cynthia Matuszek and Frank Farrero, with some material from Marie desJardin, Lise Getoor, Jean-Claude Latombe, and Daphne Koller

Review: Formalizing Agents

- Given:
  - A state space S
  - A set of actions $a_1, \ldots, a_k$ including their results
  - Reward value at the end of each trial (series of action) (may be positive or negative)
- Output:
  - A **mapping from states to actions**
  - Which is a **policy**, π

Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- P: state-action transition distribution
- γ: discount factor

Start in initial state s_0
for t = 1 to …:
    choose action a_t
    "move" to next state $s_t \sim \pi(\cdot|s_{t-1}, a_t)$
    get reward $r_t = \mathcal{R}(s_t, a_t)$

objective: maximize discounted reward

$$\max_{\pi} \sum_{t>0} \gamma^t r_t$$

"solution" $$\pi^* = \operatorname*{argmax}_{\pi} \mathbb{E}\left[\sum_{t>0} \gamma^t r_t \,; \pi\right]$$