# 18-661 Introduction to Machine Learning

## Reinforcement Learning

Spring 2025

ECE – Carnegie Mellon University

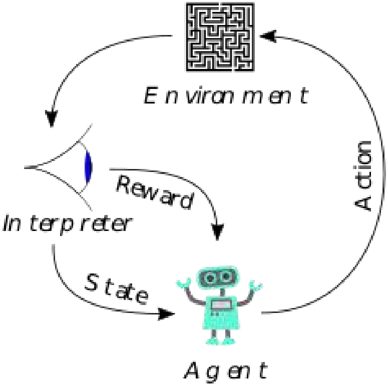
# Outline

1. MDP Recap
2. Bellman Equations
3. Finding the Optimal Policy
4. Q-learning
5. Course Summary

# MDP Recap

# Overview of Reinforcement Learning

In RL, an agent learns through interaction with the (unknown) environment.



# Objectives of RL

We choose the actions that maximize the expected total reward:

$$R(T) = \sum_{t=0}^{T} \mathbb{E}\left[r(a(t), s(t))\right], \quad R(\infty) = \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t r(a(t), s(t))\right].$$

**Infinite-horizon discounted reward.** We *discount* the reward at future times by $\gamma < 1$ to ensure convergence when $T = \infty$. The expectation is taken over the probabilistic evolution of the state, and possibly the probabilistic reward function.

A policy tells us which action to take, given the current state.

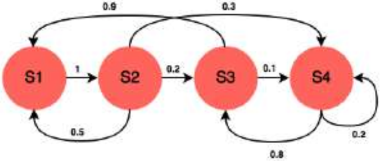
- Deterministic policy: $\pi : S \to A$ maps each state $s$ to an action $a$.
- Stochastic policy: $\pi(a|s)$ specifies a probability of taking each action $a \in A$ given state $s$. We draw an action from this probability distribution whenever we encounter state $s$.

# Transition Matrices

A Markov chain can be represented with a transition matrix:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{bmatrix}$$

Each entry $(i, j)$ is the probability of transitioning from state $s_i$ to state $s_j$.




$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.2 & 0.3 \\ 0.9 & 0 & 0 & 0.1 \\ 0 & 0 & 0.8 & 0.2 \end{bmatrix}$$

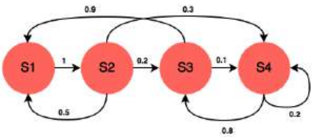
# Markov Decision Process

**Markov decision processes make state transitions dependent on user actions**

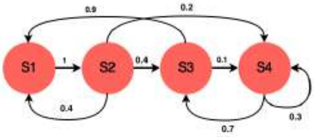
Markov processes represent transitions from state $s(t)$ to $s(t+1)$, but they do not include actions taken by the user. We now need to introduce a set of transition matrices:

$$p_{i,j} \rightarrow p^a_{i,j} = P(s(t+1) = s_j | s(t) = s_i, a(t) = a),$$

where $a$ is a given action.




Action 1




Action 2

# Evaluating a Policy: The State-Value Function

The state-value function of a given policy $\pi : S \to A$ gives its expected future reward when starting at state $s$:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(s),P,r}\left[\sum_{t=0}^{\infty}\gamma^t r\left(a(t), s(t)\right)|s(0) = s\right].$$

The expectation may be taken over a stochastic policy and reward as well as the Markov decision process (MDP) of the state transitions.

- The action $a(t)$ at any time $t$ is determined by the policy, $\pi(s(t))$.
- Due to the Markov property of the underlying MDP, the optimal policy at any time is only a function of the last observed state.

# The Action-Value Function

Consider a variant of $V_\pi(s)$, the $Q$ function

$$Q_\pi(a, s) = \mathbb{E}\left[ r(a, s) + \underbrace{\sum_{t=1}^{\infty} \gamma^t r\left(\pi(s(t)), s(t)\right)}_{\text{use policy } \pi} | s(0) = s, a(0) = a \right]$$

The $Q$ function gives the expected value of taking an action $a$ at time 0, and then following a given policy $\pi$.

# Bellman Equations

# Evaluating the State-Value Function

Let the sequence of received rewards be $r_0, r_1, \dots$ following policy $\pi$:

$$\begin{aligned}
V(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right] \\
&= \mathbb{E}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \dots | s_0 = s\right] \\
&= \mathbb{E}\left[r_0 | s_0 = s\right] + \gamma \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots | s_0 = s\right] \\
&= \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots | s_0 = s, s_1 = s'\right] \\
&= \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots | s_1 = s'\right] \\
&= \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} V(s')
\end{aligned}$$

The value function can be decomposed into two parts

- immediate reward $\mathbb{E}\left[r(s)\right]$
- discounted value of at the successor state $\gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} V(s')$

this defines a recursive relationship between the state values!

# Bellman Equation

$$\forall s: \; V(s) = \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} V(s')$$

Put this in a matrix form using the probability transition matrix $\boldsymbol{P}$ induced by the policy, and

$$\boldsymbol{V} = \begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_n) \end{bmatrix}, \qquad \boldsymbol{r} = \begin{bmatrix} \mathbb{E}\left[r(s_1)\right] \\ \mathbb{E}\left[r(s_2)\right] \\ \vdots \\ \mathbb{E}\left[r(s_n)\right] \end{bmatrix}$$

we have the Bellman equation:

$$\boldsymbol{V} = \boldsymbol{r} + \gamma \boldsymbol{P} \boldsymbol{V}$$

# Bellman Equation

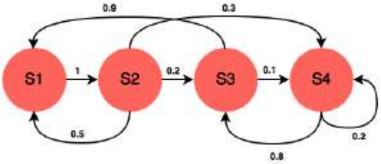
$$\boldsymbol{V} = \boldsymbol{r} + \gamma \boldsymbol{P}\boldsymbol{V}$$

- We can solve the Bellman equation directly by matrix inversion:

$$\begin{aligned}(\boldsymbol{I} - \gamma \boldsymbol{P})\boldsymbol{V} &= \boldsymbol{r} \\ \boldsymbol{V} &= (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1}\boldsymbol{r}\end{aligned}$$

- Computational complexity is $O(n^3)$ for $n$ states
- Try iterative methods for large MRPs, e.g. dynamic programming, monte-Carlo evaluation, temporal-difference learning.

# Example




$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.2 & 0.3 \\ 0.9 & 0 & 0 & 0.1 \\ 0 & 0 & 0.8 & 0.2 \end{bmatrix}$$

Assume $r(s_1) = 1$, $r(s_2) = 1$, $r(s_3) = 1$, $r(s_4) = 10$, and $\gamma = 0.9$, then the value function is

$$\boldsymbol{V} = (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1} \boldsymbol{r} = \begin{bmatrix} 21.5896 \\ 22.8773 \\ 21.2656 \\ 30.8674 \end{bmatrix}$$

Verify: $V(s_3) = r(s_3) + 0.9[0.9V(s_1) + 0.1V(s_4)]$

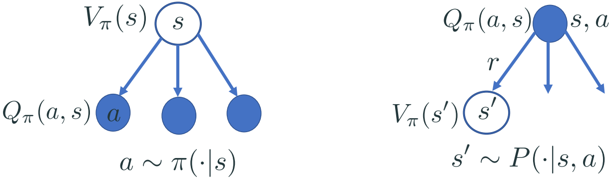
# Bellman Equations for MDP

Bellman equation for state-value function:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(a, s),$$

Bellman equation for action-value function:

$$Q_\pi(a, s) = \mathbb{E}[r(a, s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_\pi(s'),$$



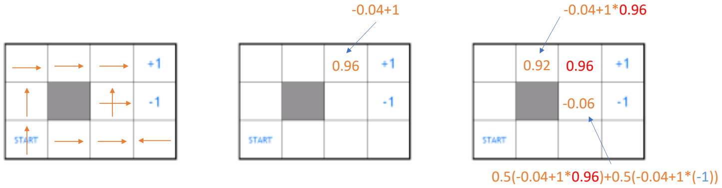

Bellman equation for state-value function:

$$V_\pi(s) = \mathbb{E}[r(a, s)] + \gamma \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s, a) V_\pi(s')$$

# Example

Consider the same maze example as earlier, where $\gamma = 1$ and

- we receive a reward of $+$ 1 if we reach [4,3] and -1 if we reach [4,2], which terminate the epoch; -0.04 for taking an action;
- The policy is plotted in arrows; where if there are two arrows in a cell, both directions take equal probability.
- One can certainly use the Bellman equation to solve the value function, but for this example, we can also find it by "dynamic programming" from the end state.



# Finding the Optimal Policy

# Optimizing the Action-Value Function

- Suppose we know the "optimal" Q-function, $Q^*(a, s)$, for each action $a$ and state $s$.
- Then, we maximize the state-value function by choosing the action $a^*(s)$ at state $s$ so as to maximize $Q^*(a, s)$.

$$V^*(s) = \max_a Q^*(a, s)$$

But this gives us $\pi^*$!

$$\pi^*(s) = \arg\max_a Q^*(a, s)$$

$$V^*(s) = V_{\pi^*}(s)$$

greedification of $Q^*$ yields the optimal policy!

- For tabular MDPs, there is always a *deterministic* optimal policy (not necessarily unique).

# Bellman's Optimality Equation

The optimal value functions are recursively related by the Bellman optimality equations:

$$V^*(s) = \max_a Q^*(a, s)$$

$$Q^*(a, s) = \mathbb{E}[r(a, s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^*(s')$$

Putting them together:

$$V^*(s) = \max_a \left[ \mathbb{E}[r(a, s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^*(s') \right]$$

$$Q^*(a, s) = \mathbb{E}[r(a, s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) max_a Q^*(a, s')$$

These equations define recursive relationships that allow us to find the optimal value functions.

# Solving the Bellman Optimality Equation

Bellman operator: one-step look-ahead

$$\mathcal{T}(Q)(a,s) = \mathbb{E}[r(a,s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) max_a Q(a,s')$$

then the Bellman's optimality equation says $Q^*$ is the (unique) fixed point of $\mathcal{T}$:

$$Q^* = \mathcal{T}(Q^*)$$

- Bellman Optimality Equation is non-linear
- There is in general no closed form solution
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-Learning
  - more...

# Value Iteration

Initialize $V(s)$ for each state $s$. Suppose we know the reward function $r(a, s)$ and transition probabilities $P(s'|s, a)$.

- Update $Q(a, s)$:

$$Q(a, s) \leftarrow \mathbb{E}[r(a, s)] + \gamma \underbrace{\sum_{s' \in \mathcal{S}} P(s'|s, a) V(s')}_{\text{Estimate of expected future reward}}$$

- Update $V(s) \leftarrow \max_a Q(a, s)$.
- Repeat until convergence.

Value iteration is guaranteed to converge to the optimal value function $V^*(s)$, and optimal action-value function $Q^*(a, s)$.

# Deterministic Value Iteration

- If we know the solution to subproblems $V^*(s')$, then the solution $V^*(s)$ can be found by one-step lookahead

$$V^*(s) = \max_a \left[ \mathbb{E}[r(a,s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right]$$

- The idea of value iteration is to apply these updates iteratively
- Start with final rewards and work backwards

# Policy Iteration

Initialize $V(s)$ and a deterministic policy $\pi(s)$ for each state $s$. Suppose we know the reward function $r(a, s)$ and transition probabilities $P(s'|s, a)$.

- Policy evaluation: update the estimate of $V(s)$:

$$V^{\pi}(s) \leftarrow \mathbb{E}\left[r(\pi(s), s)\right] + \gamma \underbrace{\sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^{\pi}(s')}_{\text{Estimate of expected future reward}}$$

- Policy improvement: improve the policy at each state $s$:

$$\pi(s) \leftarrow \arg\max_a \left[\mathbb{E}\left[r(a, s)\right] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi}(s')\right]$$

- Repeat until $\pi$ converges.
- Enjoys the same convergence rate as value iteration.

# Q-learning

# $Q$-Learning

What if we don't know $r(a, s)$ and $P(s'|s, a)$? Evolve the action-value function to find its optimal value $Q_{\pi^*}(a, s)$.

- Initialize our estimate of $Q(a, s)$ to some arbitrary value. We have dropped the dependence on $\pi^*$, as $\pi^*$ is determined by $Q_{\pi^*}(a, s)$.
- After playing action $a$ in state $s(t)$ and observing the next state $s(t+1)$, update $Q$ as follows:

$$Q(a, s(t)) \leftarrow (1-\alpha)\underbrace{Q(a, s(t))}_{\text{old value}}+\alpha\underbrace{\left(r(a, s(t)) + \gamma \max_a Q(a, s(t+1))\right)}_{\text{learned value}}.$$

  Here $\alpha$ is the learning rate and $r(a, s(t))$ is our observed reward. The term $\max_a Q(a, s(t+1))$ is our estimate of the expected future reward for the optimal policy $\pi^*$.

$Q$-learning has many variants: for instance, deep $Q$-learning uses a neural network to approximate $Q$.

# Comparison with the Bellman Operator

$$\mathcal{T}(Q)(a,s) = \mathbb{E}[r(a,s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) max_a Q(a,s')$$

Since we no longer have access to the model $P(s'|s,a)$ and $r(a,s)$, we are replacing them by the observations, specifically

- replace $\mathbb{E}[r(a,s)]$ by the observed reward;
- replace $\sum_{s' \in \mathcal{S}} P(s'|s,a)$ by the observed state transition $s'$

In addition, make a conservative update instead of a full Bellman mapping to account for stochasticity:

$$Q(a,s) \leftarrow (1-\alpha)\underbrace{Q(a,s)}_{\text{old value}} + \alpha \underbrace{\left(r(a,s(t)) + \gamma \max_a Q(a,s')\right)}_{\text{learned value}}.$$

# Exploration vs. Exploitation

Given $Q(a, s)$, how do we choose our action $a$?

- Exploitation: Take action $a^* = \arg\max_a Q(a, s)$. Given our current estimate of $Q$, we want to take what we think is the optimal action.
- Exploration: But we might not have a good estimate of $Q$, and we don't want to bias our estimate towards an action that turns out not to be optimal.
- $\epsilon$-Greedy: With probability $1 - \epsilon$, choose $a^* = \arg\max_a Q(a, s)$, and otherwise choose $a$ randomly. Usually, we decrease $\epsilon$ over time as additional exploration becomes less important.

# $Q$-Learning Algorithm

Initialize our estimate of $Q(a, s)$ to some arbitrary value.

- Choose an action $a$ using the $\epsilon$-greedy policy.
- Observe reward $r(a, s(t))$ and state $s(t)$.
- Update $Q$ as follows:

$$Q(a, s(t)) \leftarrow (1-\alpha)\underbrace{Q(a, s(t))}_{\text{old value}} + \alpha \underbrace{\left(r(a, s(t)) + \gamma \max_a Q(a, s(t+1))\right)}_{\text{learned value}}.$$

- Repeat for $T$ iterations.

$Q$-learning always learns an optimal policy, regardless of how $a$ is chosen (you don't need to use $\epsilon$-greedy)! Thus, it is an off-policy method.

# Types of Machine Learning

**Supervised Learning**

- Training data: $(x, y)$ (features, label) samples. We want to predict $y$ to minimize a loss function.
- Regression, classification

**Unsupervised Learning**

- Training data: $x$ (features) samples only. We want to find "similar" points in the $x$ space.
- Clustering, PCA

**Online/Sequential Learning**

- Training data: $(s, a, r)$ (state,action,reward) samples. We want to find the best sequence of decisions so as to maximize long-term reward.
- multi-armed bandits, reinforcement learning

# Summary

You should know:

- Bellman optimality equation
- Value iteration, policy iteration, Q-learning

# Course Summary

# Definition

**Machine learning is**: the study of methods that

*improve their performance*

*on some task*

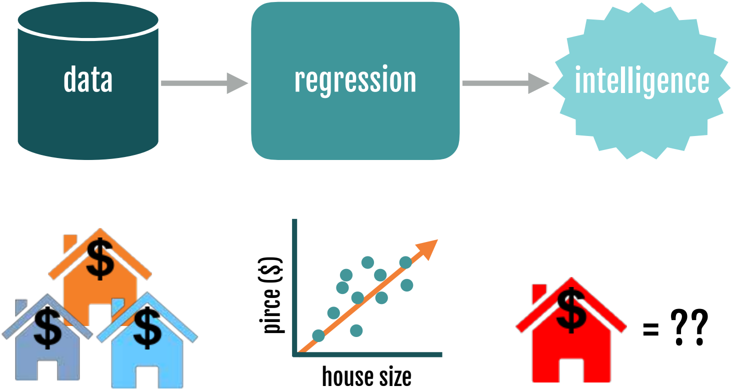
*with experience*

# Machine Learning Pipeline

# Examples

# Example 1: Regression

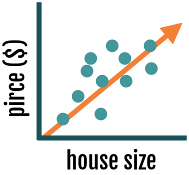
*How much should you sell your house for?*




**input**: houses & features **learn**: $x \to y$ relationship **predict**: $y$ (*continuous*)

# Example 1: Regression

*How much should you sell your house for?*





**input**: houses & features **learn**: $x \to y$ relationship **predict**: $y$ (*continuous*)

**Models**

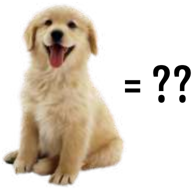
- Linear regression
- Nonlinear models: neural networks/deep learning, decision trees, nearest neighbors

**Methods**

- MLE/MAP
- Gradient descent
- Ensamble methods (bagging, boosting)

# Example 2: Classification

*Cat or dog?*


**input**: cats and dogs

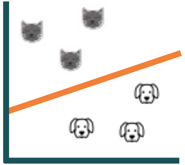
**learn**: $x \to y$ relationship

**predict**: $y$ (*categorical*)

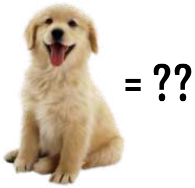
# Example 2: Classification

*Cat or dog?*

**input**: cats and dogs



**learn**: $x \to y$ relationship




**predict**: $y$ (*categorical*)

**Models**

- Linear classification: Naïve Bayes, logistic regression, SVM
- Nonlinear models: kernel SVM, neural networks, decision trees, nearest neighbors

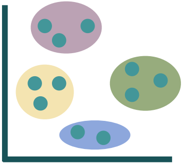
**Methods**

- MLE/MAP
- Gradient descent
- Ensamble methods (bagging, boosting)

# Example 3: Clustering

*How to segment an image?*


**input**: raw pixels $\{x\}$



**separate**: $\{x\}$ into sets

**output**: cluster labels $\{z\}$

# Example 3: Clustering

*How to segment an image?*


**Models**

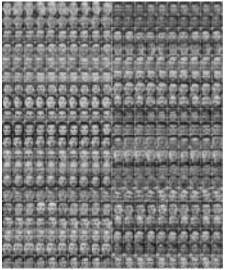
- Clustering
- Gaussian mixture models

**Methods**

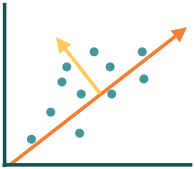
- $k$-means
- EM

# Example 4: Embedding

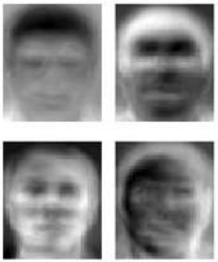
*How to reduce size of dataset?*

**input**: large dataset $\{x\}$



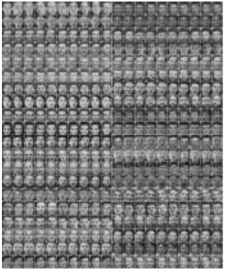
**find**: sources of variation

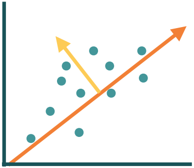


**return**: representation $\{z\}$

# Example 4: Embedding

*How to reduce size of dataset?*





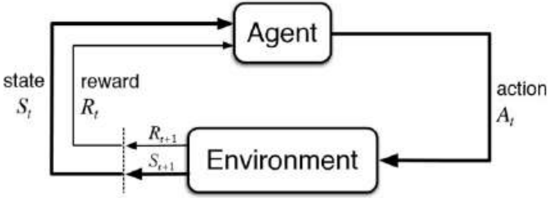
**input**: large dataset $\{x\}$ **find**: sources of variation **return**: representation $\{z\}$

## Methods and Concepts

- PCA

# Example 5: Sequential and Online Learning

*How to learn sequentially in unknown environments?*




**Methods and Concepts**

- Markov decision processes for reinforcement learning

## Key Topics

**Models**

- Linear regression
- Linear classification: Naïve Bayes, logistic regression, SVM
- Nonlinear models: kernels, neural networks & deep learning, decision trees
- Nearest neighbors
- Clustering, GMM

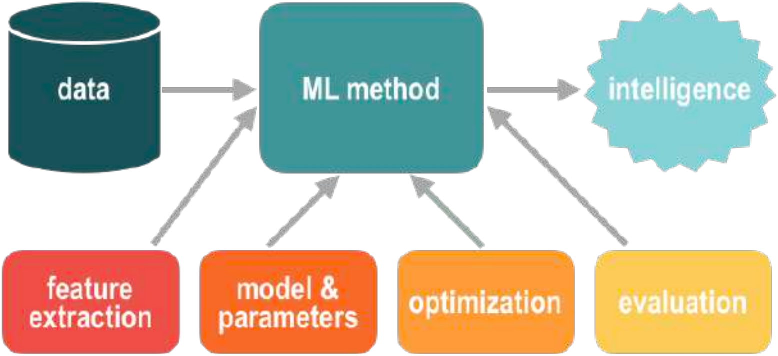
**Methods**

- (Stochastic) gradient descent
- Boosting
- $k$-means
- EM
- PCA
- Distributed learning

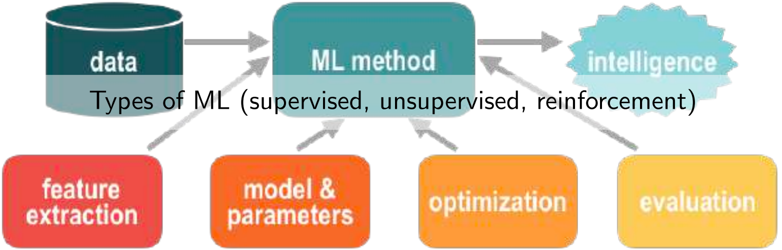
**Concepts**

- MLE, MAP
- Loss functions, bias-variance tradeoff, cross-validation
- Bandits
- MDP, Bellman equation
- Types of ML (supervised, unsupervised, reinforcement)

# Goal: Learn about ML Pipeline



# Fitting the Course into the Pipeline




Types of ML (supervised, unsupervised, reinforcement)

- PCA

- MLE, MAP
- Linear regression
- Linear classification
- Nonlinear models
- Nearest neighbors, clustering, GMM

- Gradient descent
- Boosting
- EM

- Loss functions, bias-variance tradeoff, cross-validation
- Overfitting

# Please Complete Course Evaluations!

We will appreciate your constructive feedback on:

- Course Content/Structure
- Level of Math
- Homework and Pytorch assignment
- Our teaching
- and other suggestions/feedbacks

Completing course evaluations will earn you 1 bonus point towards your final grade. Please upload the completion screenshot or email on Gradescope. Your answers will remain anonymous.

**Thank you for taking the class!!**

**We hope you've enjoyed it as much as we have.**

# Final Exam Logistics

- Final Exam Friday of next week, May 2, 1 pm-4 pm ET (10am-1pm PT, 7pm-10pm CAT) in HOA 160 (Pitt), CMR F205 (Kigali), B23 118 (SV). This is not the usual Pittsburgh lecture room.
- All topics are included, with a focus on post-midterm topics
- Mix of true/false, multiple choice, and descriptive questions. No coding questions.
- 2 double-sided, US-letter or A4 sized handwritten cheat sheets allowed
- Calculators not permitted and not useful.
- Take the practice final exam, which will be posted on Piazza soon.

Office hours will be held as normal with exceptions noted on Piazza. Recitation will also give you a chance to practice for the exam, and ask questions about the material.

**Good luck!**