Unit-2

Learning Paradigms

By Dr. Gopal Chandra Jana, Assistant Professor, DCSE, SSCSE

Course Page: https://www.gcjana.in/courses/shardauniversity/2502/CSA203/

Topic to be Covered: Unit 2 Learning Paradigms

A.1. Learning Paradigms and their real Applications,
A.2. Supervised learning,
A.3. Unsupervised learning,
A.4. Reinforcement learning,
A.5. Offline and online learning and
A.6. Their applications based on real life problems.

B.1. Training patterns and teaching inputs,
B.2. Use of training samples,
B.3. Data set split into training,
B.4. Validation and testing data,
B.5. Implication of splitting of data set,
B.6. Learning curves and their importance in diagnostics

C.1. Gradient optimization procedures,
C.2. Hebbian learning rule

What is there to learn

- What are the different **ways** in which computers can learn?
- Are there learning tasks that humans can do much **better** than computers?

Where from WE (Humans) learn ?

Humans learn from **experience** and **adapt their actions for future tasks**.

Can machines adapt their behavior using experience?

Since the 1950s, researchers have been trying to develop techniques that enable machines to learn. There have been much success in areas such as automatic control, recognition systems and natural language processing. Other successes are emerging.

A.1. Learning Paradigms and their real Applications

A learning paradigm defines: How a machine improves its performance from experience (data).

Tom Mitchell's (1997)formal definition:

A program learns from experience E with respect to task T and performance measure P, if its performance on T, measured by P, improves with E.

Nutshell: A system learns if **P** improves on **T** with **E**.

A.1. Learning Paradigms and their real Applications

Can we think like this?

An algorithm is said to ***learn*** from experience ***E*** with respect some class of tasks ***T*** and performance measure ***P*** …

… if its performance at tasks in ***T***, as measured by ***P***, improves as it does task in ***T*** (experience ***E***).

Applications of Learning

Example 1 **Learning to recognize faces**

- T: recognize faces
- P: % of correct recognitions
- E: opportunity to makes guesses and being told what the truth is

Example 2 **Learning to find clusters in data**

- T: finding clusters
- P: compactness of groups detected
- E: analyses of a growing set of data

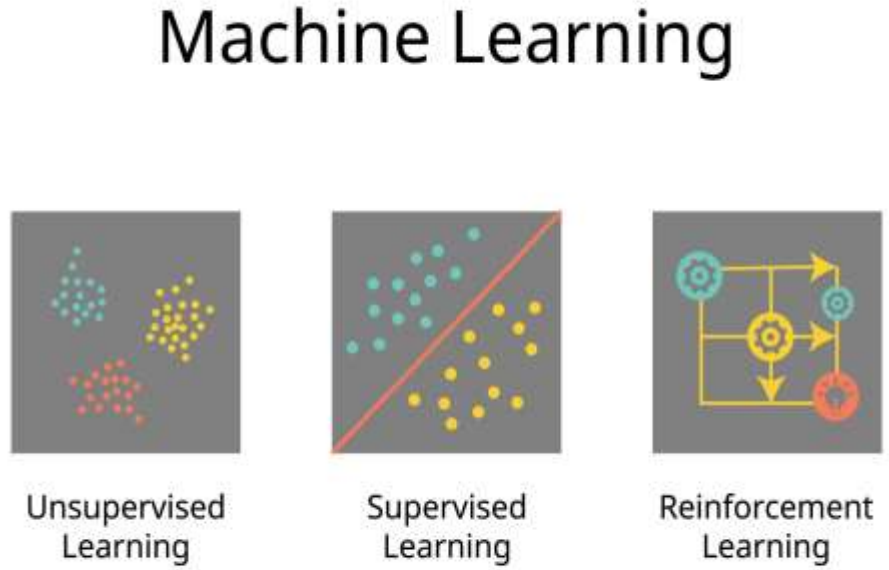
Applications of Learning

Application	Input	Output
Email spam detection	Email text	Spam / Not spam
Disease prediction	Patient data	Disease class
EEG seizure detection	EEG features	Seizure / Non-seizure
Credit scoring	Financial profile	Risk score
Face recognition	Image	Person ID

A.1. Learning Paradigms and their real Applications

Machine learning is commonly separated into three main learning paradigms:

- **Supervised Learning**,
- **Unsupervised Learning**, and
- **Reinforcement Learning**



A.1. Learning Paradigms and their real Applications

Points to be noted:

- These paradigms differ in the tasks they can solve and in how the data is presented to the computer.
- Usually, the task and the data directly determine which paradigm should be used (and in most cases, it is supervised learning).
- In some cases, though, there is a choice to make.
- Often, these paradigms can be used together in order to obtain better results.

A.2. Supervised Learning - Core Mathematical Model

You are given a dataset:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

- $x_i \in \mathbb{R}^d$ → feature vector
- y_i → label (real value for regression, class for classification)

We want to learn a function:

$$f(x; \theta)$$

where

θ = model parameters (weights)

A.2. Supervised Learning - Core Mathematical Model

You are given a dataset:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

- $x_i \in \mathbb{R}^d$ → feature vector
- y_i → label (real value for regression, class for classification)

We want to learn a function:

$$f(x; \theta)$$

where

θ = model parameters (weights)

Supervised learning is basically:

Find a function that maps inputs to outputs by minimizing error.

A.2. Supervised Learning - Core Mathematical Model

You are given a dataset:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

- $x_i \in \mathbb{R}^d$ → feature vector
- y_i → label (real value for regression, class for classification)

We want to learn a function:

$$f(x; \theta)$$

where

θ = model parameters (weights)

Supervised learning is basically:

Find a function that maps inputs to outputs by minimizing error.

Minimize a Loss Function

A.2. Supervised Learning - Matrix Form (Compact & Powerful)

Let

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Prediction:

$$\hat{y} = Xw$$

Loss:

$$L(w) = \|y - Xw\|^2$$

A.2. Supervised Learning - Prediction Model (Hypothesis Function)

Example: **Linear model**

$$\hat{y} = f(x;\theta) = w^T x + b$$

or

$$\hat{y} = \sum_{j=1}^{d} w_j x_j + b$$

This is the hypothesis.

❒ **Hypothesis** — The hypothesis is noted h_θ and is the model that we choose. For a given input data $x^{(i)}$ the model prediction output is $h_\theta(x^{(i)})$.

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning/

A.2. Supervised Learning - Loss Function (Error Measure)

This measures how wrong the prediction is.

❒ **Loss function** — A loss function is a function $L : (z, y) \in \mathbb{R} \times Y \longmapsto L(z, y) \in \mathbb{R}$ that takes as inputs the predicted value z corresponding to the real data value y and outputs how different they are. The common loss functions are summed up in the table below:

We measure **error** between prediction and true value:

$$\text{error}_i = \mathcal{L}(y_i, \hat{y}_i)$$

This $\mathcal{L}$ is called the **loss for one example.**

Regression → Mean Squared Error (MSE)

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning/

Classification → Cross Entropy Loss

For binary classification:

$$\hat{y} = \sigma(w^T x + b)$$

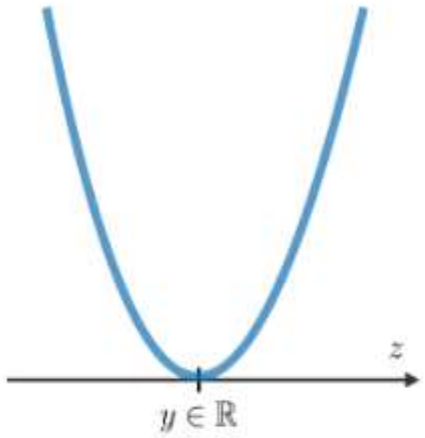
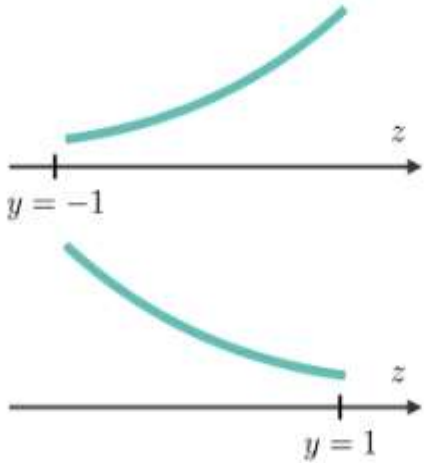
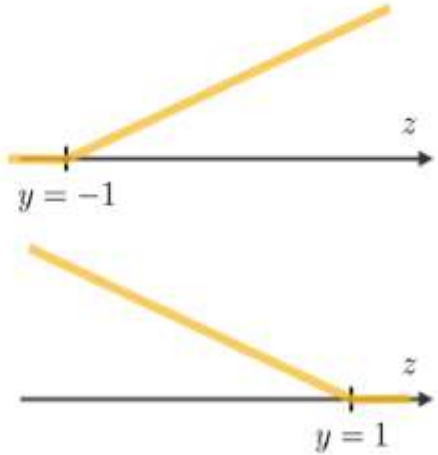
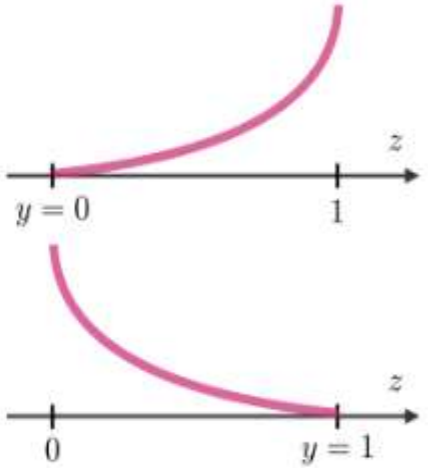
where sigmoid:

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

Loss:

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

A.2. Supervised Learning - Loss Function (Error Measure)

Least squared error	Logistic loss	Hinge loss	Cross-entropy
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0, 1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Linear regression	Logistic regression	SVM	Neural Network

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning/

A.2. Supervised Learning - Cost Function (Error Measure)

❒ **Cost function** — The cost function J is commonly used to assess the performance of a model, and is defined with the loss function L as follows:

$$J(\theta) = \sum_{i=1}^{m} L(h_\theta(x^{(i)}), y^{(i)})$$

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning/

A.2. Supervised Learning - Cost Function (Error Measure)

The **cost function** (also called empirical risk) is just the **sum / average of all individual losses:**

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\big(y_i,\ f(x_i;\theta)\big)$$

This $J(\theta)$ **is the cost function.**

> It depends on θ because predictions depend on θ.

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning/

A.2. Supervised Learning - Example: Linear Regression Cost

Model:

$$\hat{y}_i = w^T x_i + b$$

Loss per point:

$$(y_i - \hat{y}_i)^2$$

Cost:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - (w^T x_i + b)\right)^2$$

Here:

$$\theta = \{w, b\}$$

So **cost is a function of weights.**

A.2. Supervised Learning - Example: Logistic Regression Cost

Model:

$$\hat{y}_i = \sigma(w^T x_i + b)$$

Loss per point (cross entropy):

$$-\left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right]$$

Cost:

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right]$$

A.2. Supervised Learning - Objective Function (Optimization Target)

This is the heart of supervised learning. $$\theta^* = \arg\min_{\theta} L(\theta)$$

❒ **Likelihood** — The likelihood of a model $L(\theta)$ given parameters θ is used to find the optimal parameters θ through likelihood maximization. We have:

$$\boxed{\theta^{\text{opt}} = \arg\max_{\theta} L(\theta)}$$

Remark: in practice, we use the log-likelihood $\ell(\theta) = \log(L(\theta))$ *which is easier to optimize.*

Minimizing Cost $\iff$ Maximizing Likelihood

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning/

A.2. Supervised Learning - How Parameters Are Learned → Gradient Descent

We update parameters using derivatives.

$$\theta := \theta - \eta \nabla_\theta L(\theta)$$

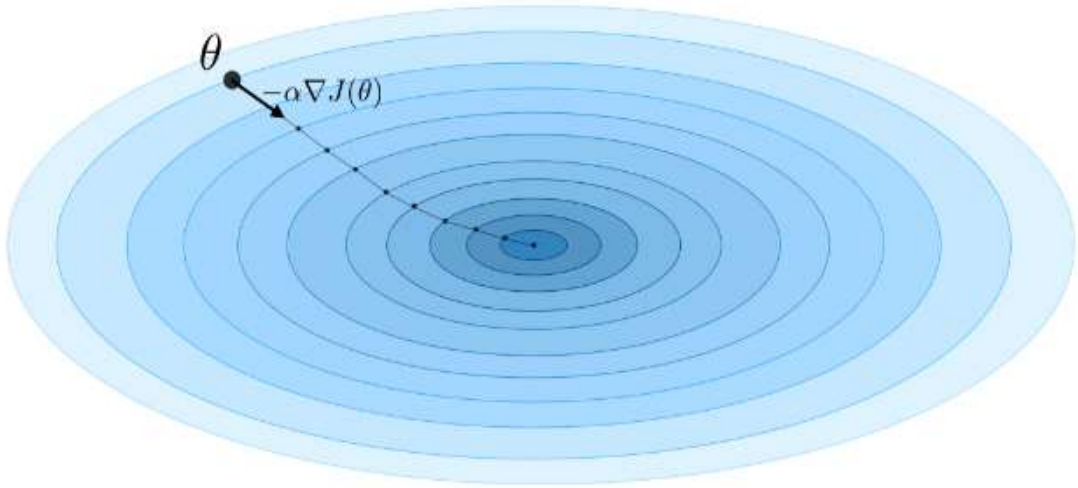
where

η = learning rate

A.2. Supervised Learning - How Parameters Are Learned → Gradient Descent

❒ **Gradient descent** — By noting $\alpha \in \mathbb{R}$ the learning rate, the update rule for gradient descent is expressed with the learning rate and the cost function J as follows:

$$\boxed{\theta \longleftarrow \theta - \alpha \nabla J(\theta)}$$



Remark: Stochastic gradient descent (SGD) is updating the parameter based on each training example, and batch gradient descent is on a batch of training examples.

A.2. Supervised Learning - Example derivative for Linear Regression

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - \hat{y}_i)$$

Update:

$$w := w + \eta\frac{2}{n}\sum x_i(y_i - \hat{y}_i)$$

A.2. Supervised Learning - Regularization (Avoid Overfitting)

Ridge (L2):

$$L(w) = \|y - Xw\|^2 + \lambda\|w\|^2$$

$$w = (X^T X + \lambda I)^{-1} X^T y$$

Lasso (L1):

$$L(w) = \|y - Xw\|^2 + \lambda \sum |w_j|$$

A.2. Supervised Learning - General Supervised Learning Template

Almost every algorithm (SVM, NN, Logistic, etc.) follows:

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i; \theta)) + \lambda \Omega(\theta)$$

where

- $\mathcal{L}$ = loss
- f = model
- Ω = regularization

A.2. Unsupervised Learning - Baseline

- ✓ **In supervised learning, we have seen $x \rightarrow y$.**

A.2. Unsupervised Learning - Baseline

- ✓ **In supervised learning, we have seen $x \rightarrow y$.**

"If I give you only data points and no labels… what can you still do?"

A.2. Unsupervised Learning - Baseline

✓ **In supervised learning, we have seen $x \rightarrow y$.**

"If I give you only data points and no labels… what can you still do?"

We can do:

- Group them
- Find patterns
- Compress them

A.2. Unsupervised Learning - Baseline

✓ **In supervised learning, we have seen $x \rightarrow y$.**

"If I give you only data points and no labels… what can you still do?"

We can do:

- Group them
- Find patterns
- Compress them

That is Unsupervised Learning.

A.2. Unsupervised Learning - Baseline

✓ **In supervised learning, we have seen $x \rightarrow y$.**

"If I give you only data points and no labels… what can you still do?"

We can do:

- Group them
- Find patterns
- Compress them

That is Unsupervised Learning.

? In unsupervised learning, the shock is:

There is no y. Only x.

A.2. Unsupervised Learning - Baseline

Motivation: The goal of unsupervised learning is to find hidden patterns in unlabeled data $\{x(1),...,x(n)\}$.

$$D = \{x_1, x_2, \ldots, x_n\}, \quad x_i \in \mathbb{R}^d$$

Goal: discover **structure** in data.

We learn parameters θ from only x.

A.2. Unsupervised Learning - General Objective

Motivation: The goal of unsupervised learning is to find hidden patterns in unlabeled data $\{x(1),...,x(n)\}$.

$$\theta^* = \arg\min_{\theta} J(\theta; X)$$

$$D = \{x_1, x_2, \ldots, x_n\}, \quad x_i \in \mathbb{R}^d$$

Goal: discover **structure** in data.

We learn parameters θ from only x.

But now J is **not error vs label**.

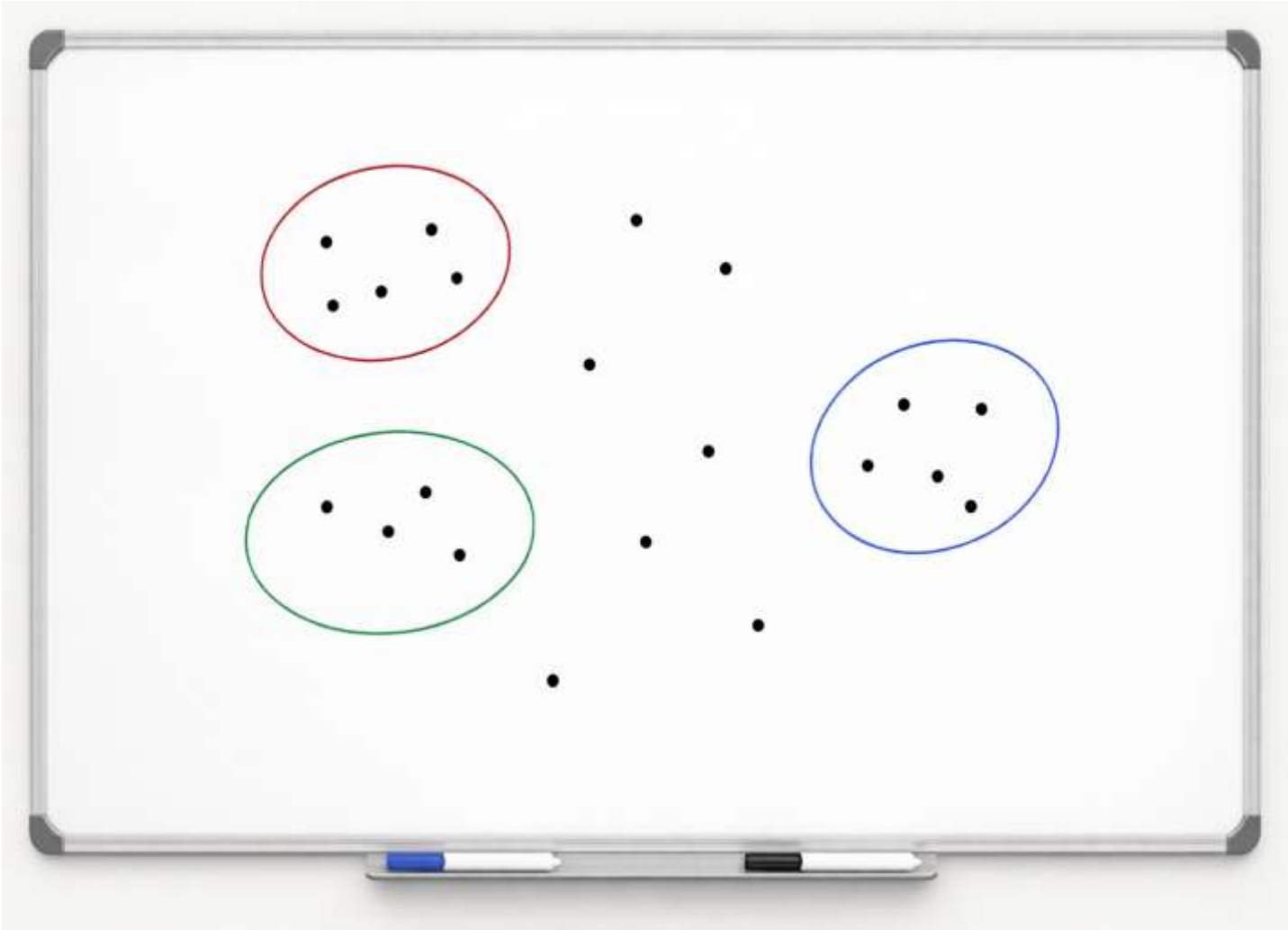
It measures: Similarity, compactness, reconstruction, probability of data, and low-dimensional structure.

A.2. Unsupervised Learning - Baseline

Motivation: The goal of unsupervised learning is to find hidden patterns in unlabeled data $\{x(1),...,x(n)\}$.

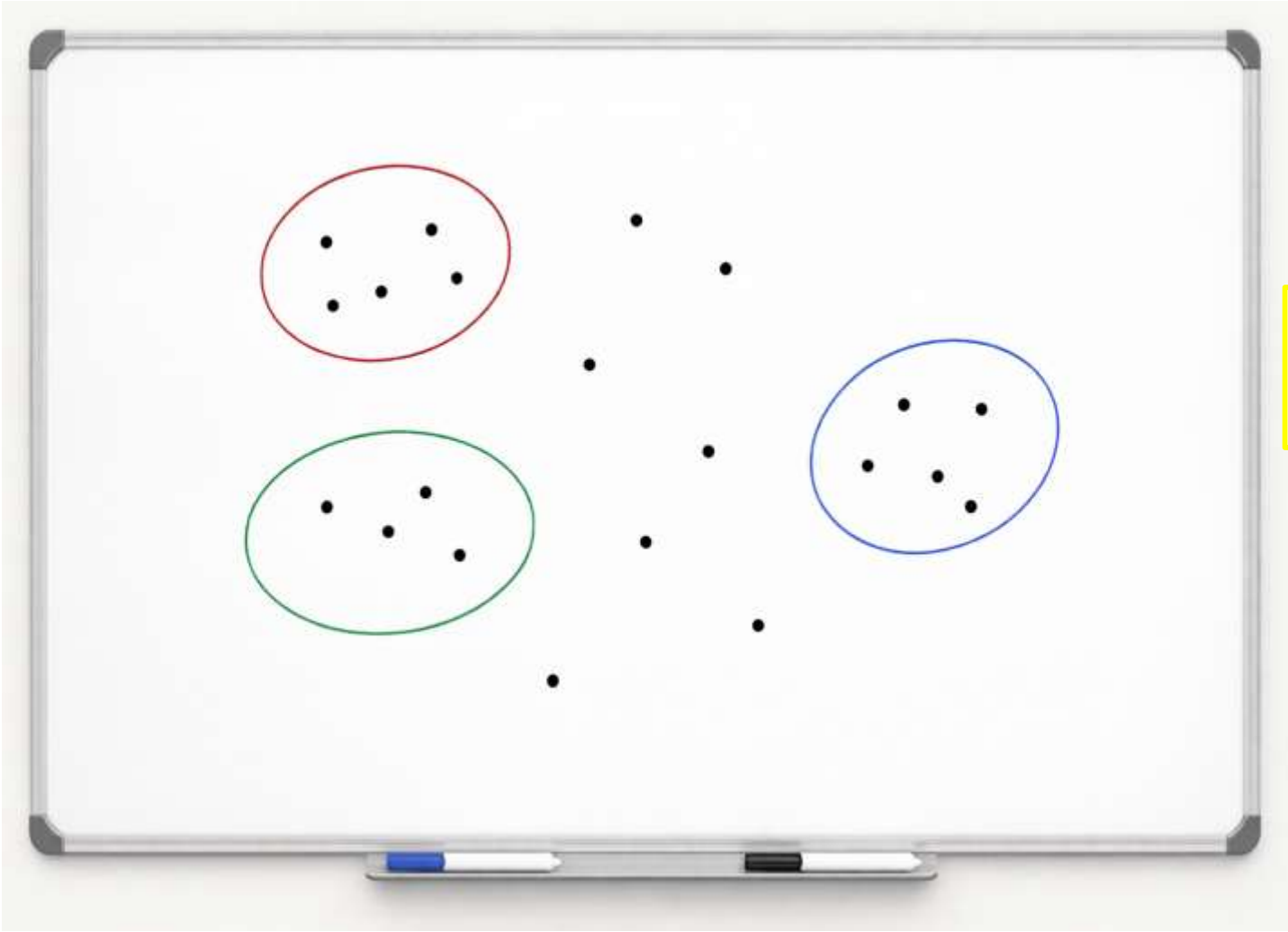
"If I give you only points on paper (no labels)… what quantity can you minimize so that a pattern appears automatically?"

A.2. Unsupervised Learning - Baseline



- Draw 10 to 12 random dots on your class notebook page.
- Circle points that look close.

A.2. Unsupervised Learning - Baseline



- Draw 10 to 12 random dots on your class notebook page.
- Circle points that look close.

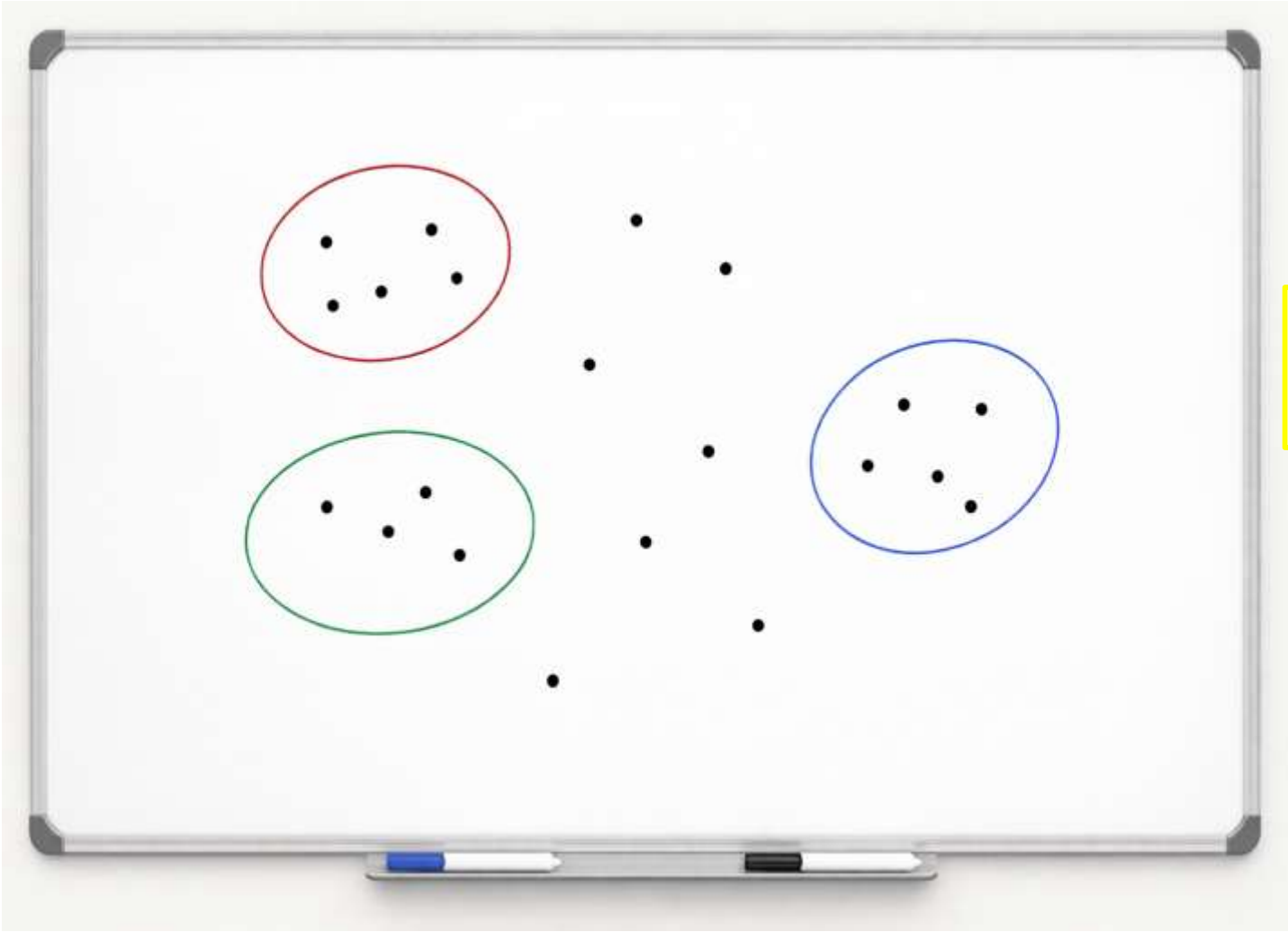
$$J = \sum \mathcal{L}(y_i, f(x_i))$$

Earlier in supervised Learning

Now in Unsupervised Learning

$$J = \sum \text{distance}(x_i, \text{nearby points})$$

A.2. Unsupervised Learning - Baseline



- Draw 10 to 12 random dots on your class notebook page.
- Circle points that look close.

$$J = \sum \mathcal{L}(y_i, f(x_i))$$

Earlier in supervised Learning

Now in Unsupervised Learning

$$J = \sum \text{distance}(x_i, \text{nearby points})$$

A.2. Unsupervised Learning - Clustering

Let's formalize distance.

$$J = \sum_{i=1}^{n} \|x_i - \mu_{c_i}\|^2$$

$$\theta^* = \arg\min_{\theta} J(\theta; X)$$

We invented our own 'labels' c_i — called cluster ids.

- In supervised → **labels given**
- In unsupervised → **labels created**

A.2. Unsupervised Learning - k-means clustering

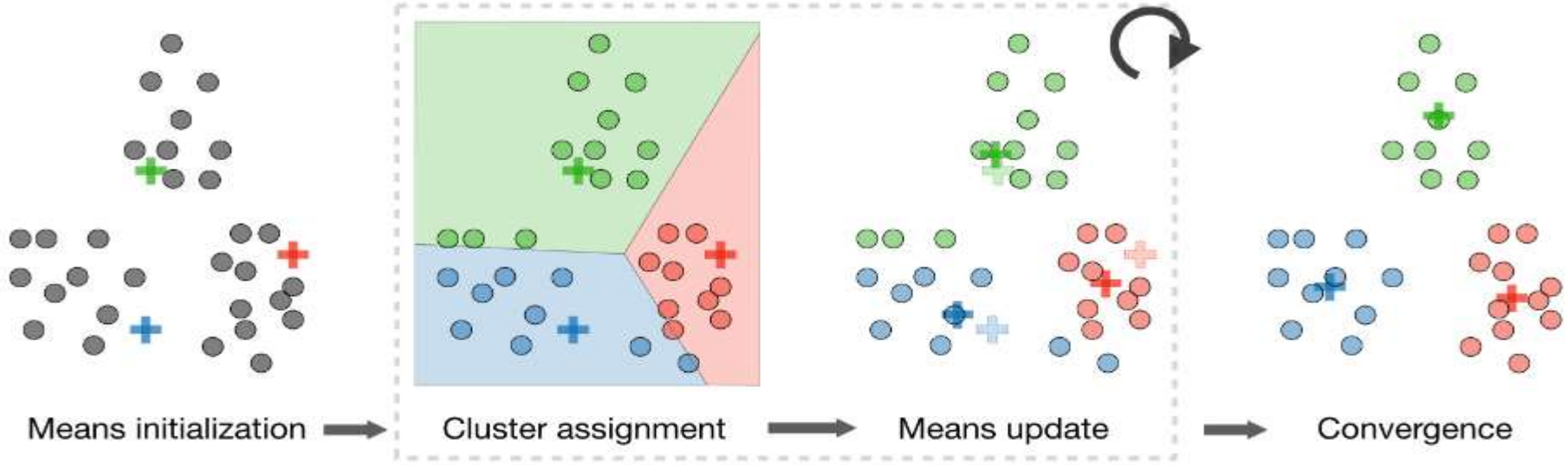
We note $c^{(i)}$ the cluster of data point i and μ_j the center of cluster j.

❒ **Algorithm** — After randomly initializing the cluster centroids $\mu_1, \mu_2, ..., \mu_k \in \mathbb{R}^n$, the k-means algorithm repeats the following step until convergence:

$$c^{(i)} = \arg\min_j ||x^{(i)} - \mu_j||^2 \quad \text{and} \quad \mu_j = \frac{\sum_{i=1}^{m} 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{m} 1_{\{c^{(i)}=j\}}}$$

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-unsupervised-learning/

A.2. Unsupervised Learning - k-means clustering



❒ **Distortion function** — In order to see if the algorithm converges, we look at the distortion function defined as follows:

$$J(c,\mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-unsupervised-learning/

So far… Unsupervised Learning

https://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf

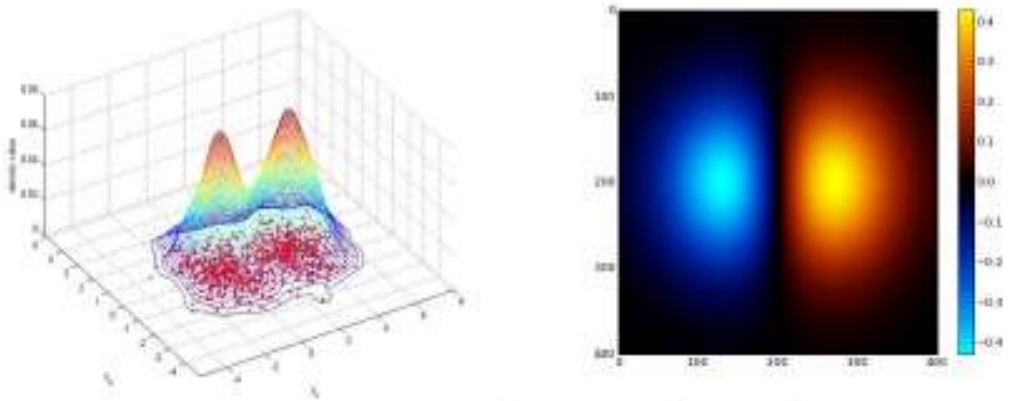
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation

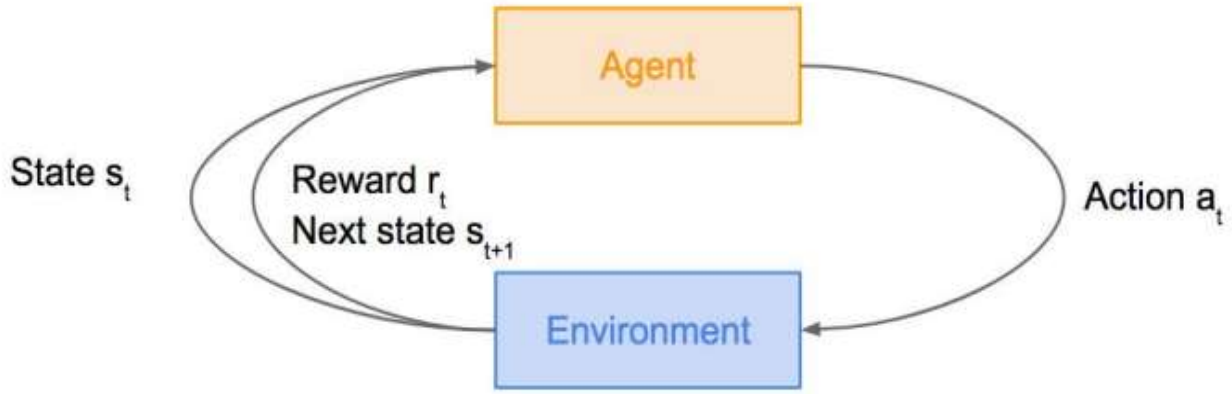


2-d density estimation

A.3. Reinforcement Learning

An **agent** learns by **interacting** with an **environment** and receiving **rewards/penalties**.

Goal: Learn a **policy** that **maximizes** long-term **reward**.



Elements:

State (s) – current situation

Action (a) – decision taken

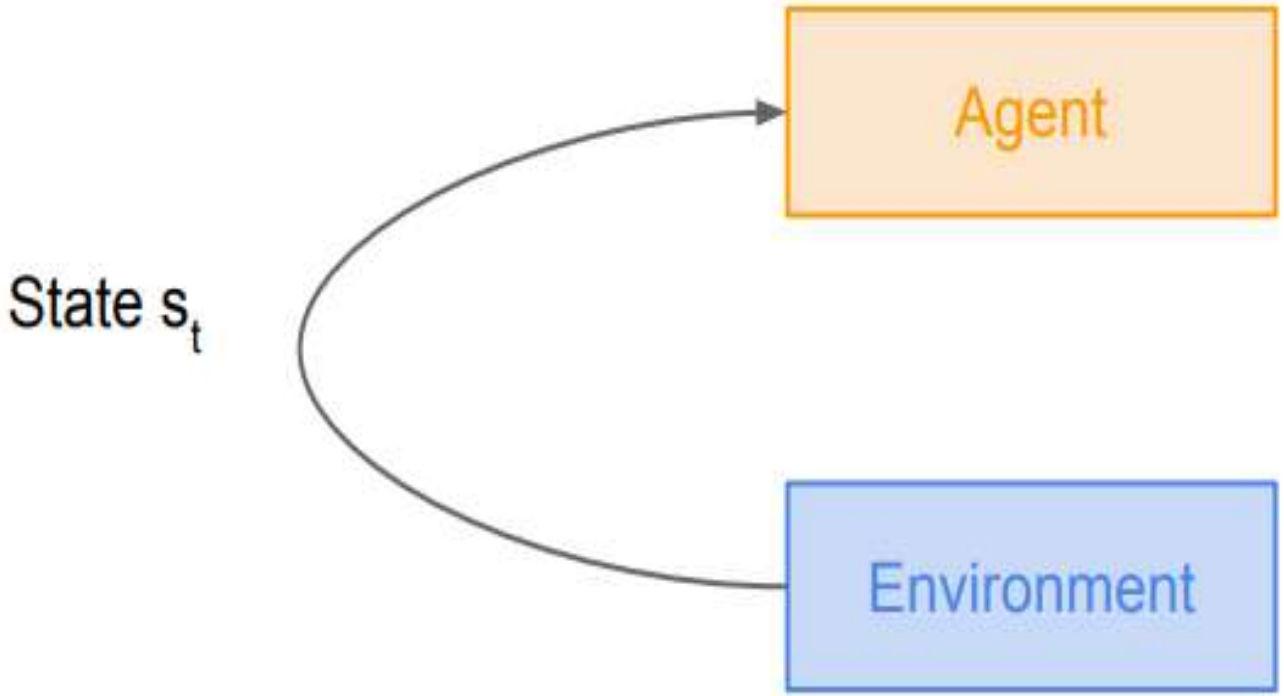
Reward (r) – feedback

Policy (π) – mapping $\pi(s) \rightarrow a$

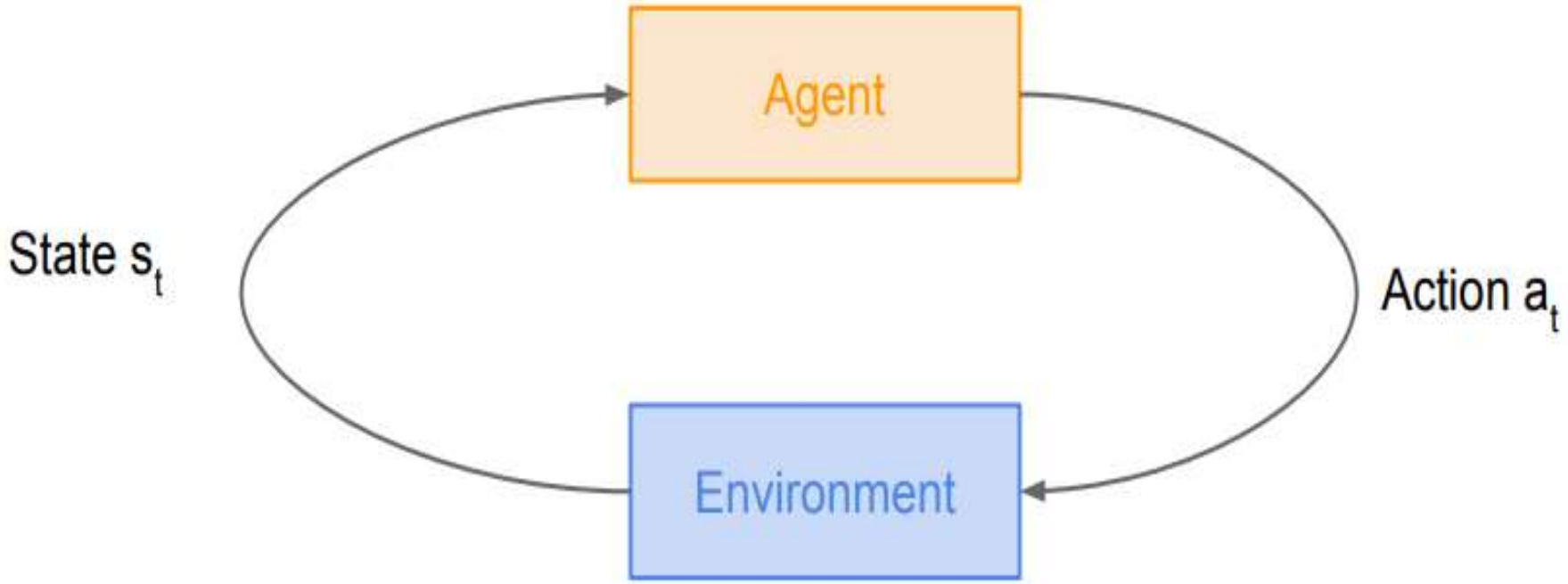
Value/Q-value – expected future reward

A.3. Reinforcement Learning

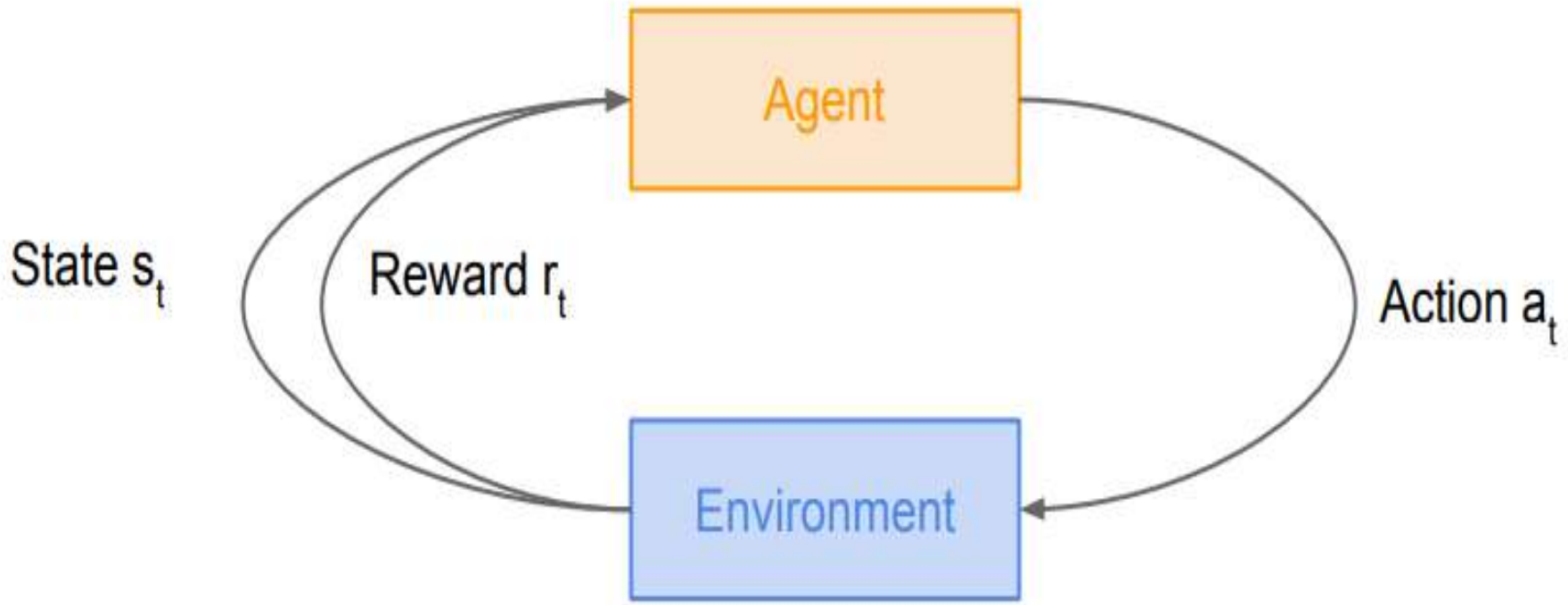
A.3. Reinforcement Learning



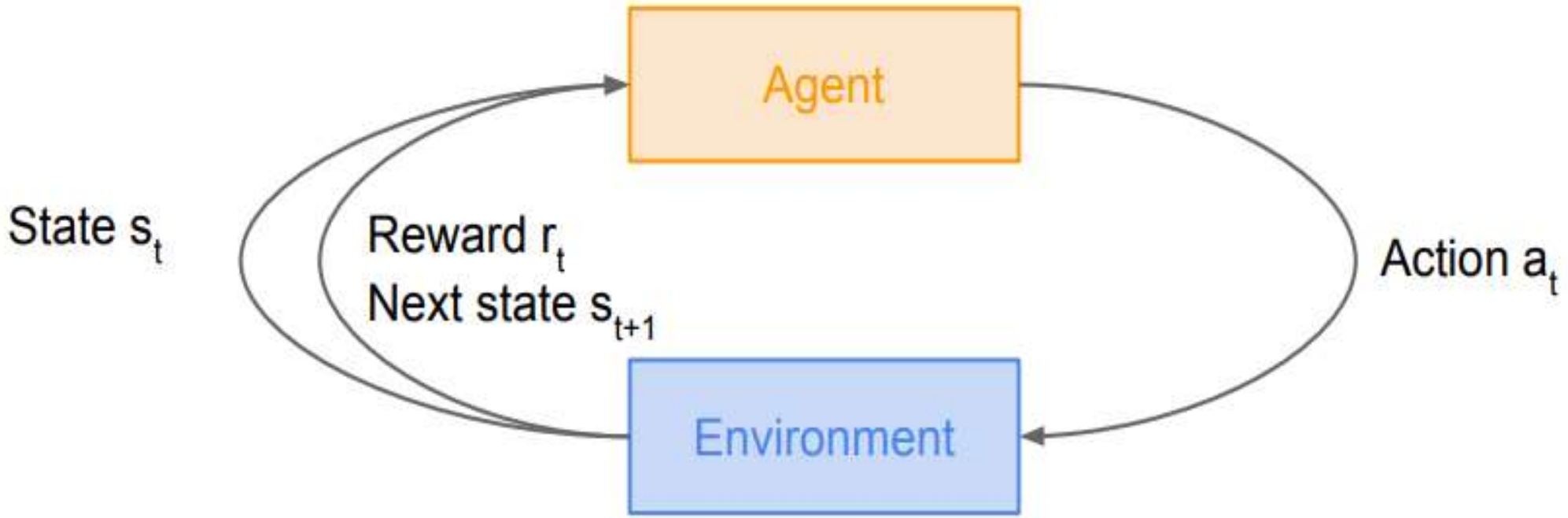
A.3. Reinforcement Learning



A.3. Reinforcement Learning

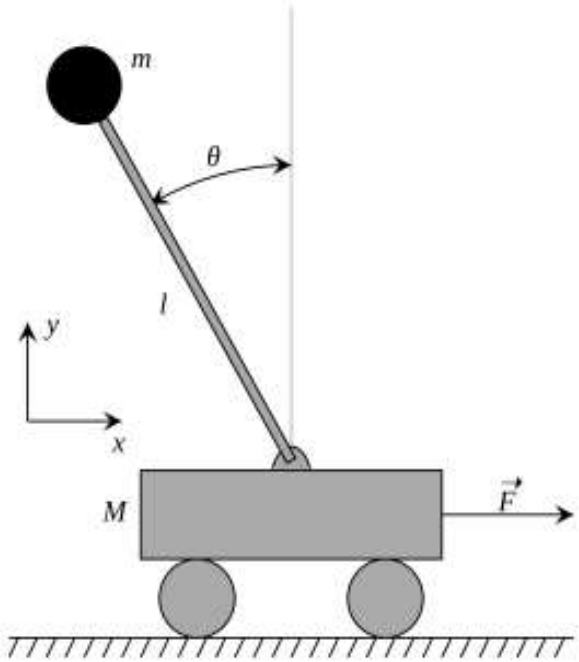


A.3. Reinforcement Learning



A.3. Reinforcement Learning

Cart-Pole Problem

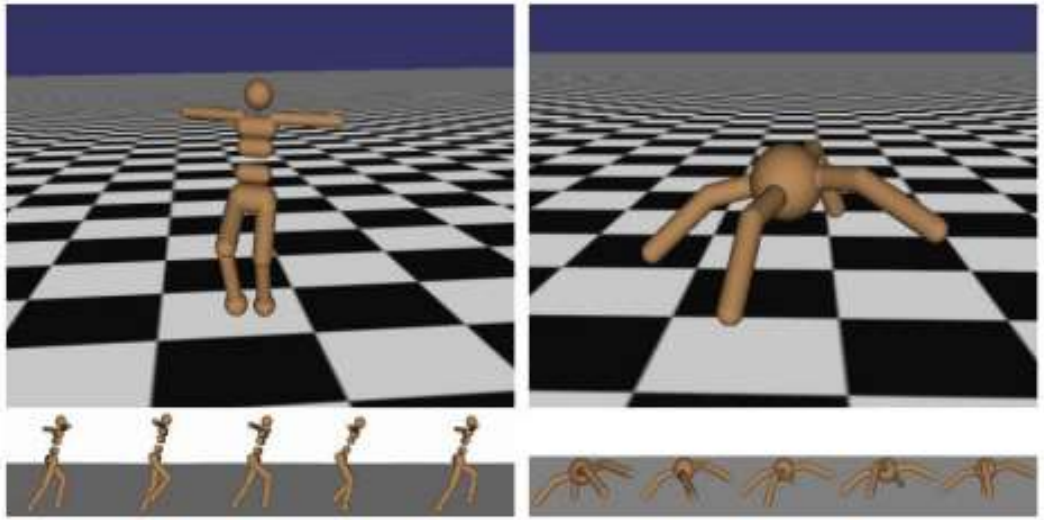


Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

A.3. Reinforcement Learning

Robot Locomotion

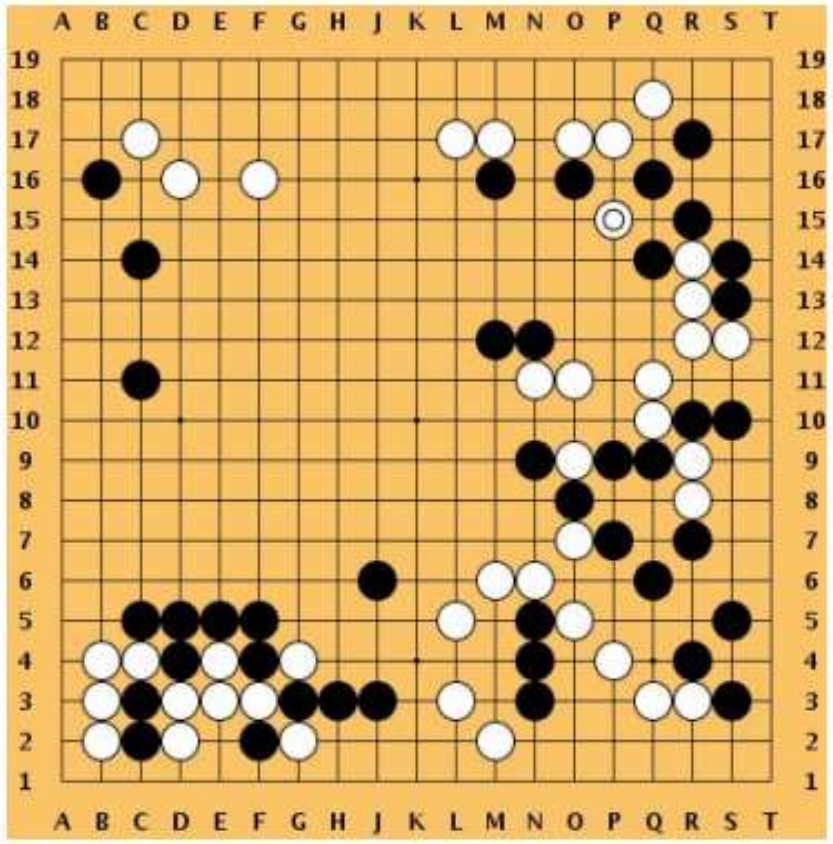


Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

A.3. Reinforcement Learning

Go

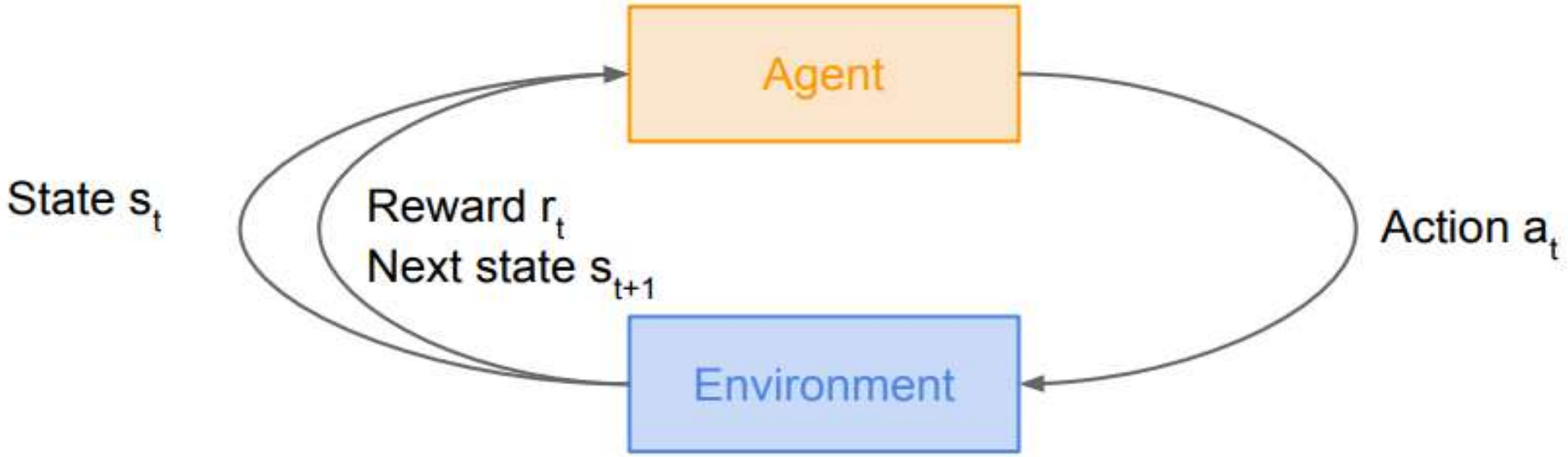


Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

A.3. Reinforcement Learning

How can we mathematically formalize the RL problem?



A.3. Reinforcement Learning

Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

A.3. Reinforcement Learning

Markov Decision Process

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(. | s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. | s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

- A policy π is a function from S to A that specifies what action to take in each state
- **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

A.3. Reinforcement Learning

The optimal policy π^*

We want to find optimal policy π^* that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?

A.3. Reinforcement Learning

The optimal policy π^*

We want to find optimal policy π^* that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?
Maximize the **expected sum of rewards!**

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

A.3. Reinforcement Learning

$$\max R = \sum_{t=0}^{\infty} \gamma^t r_t$$

γ: discount factor (importance of future reward)

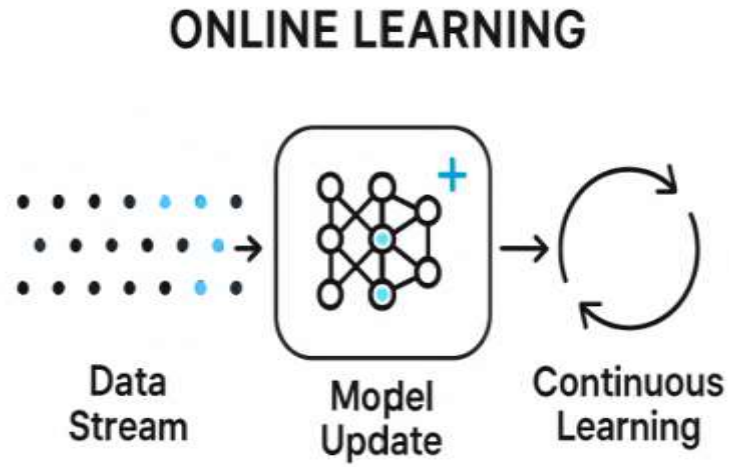
Learning (Q-Learning)

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

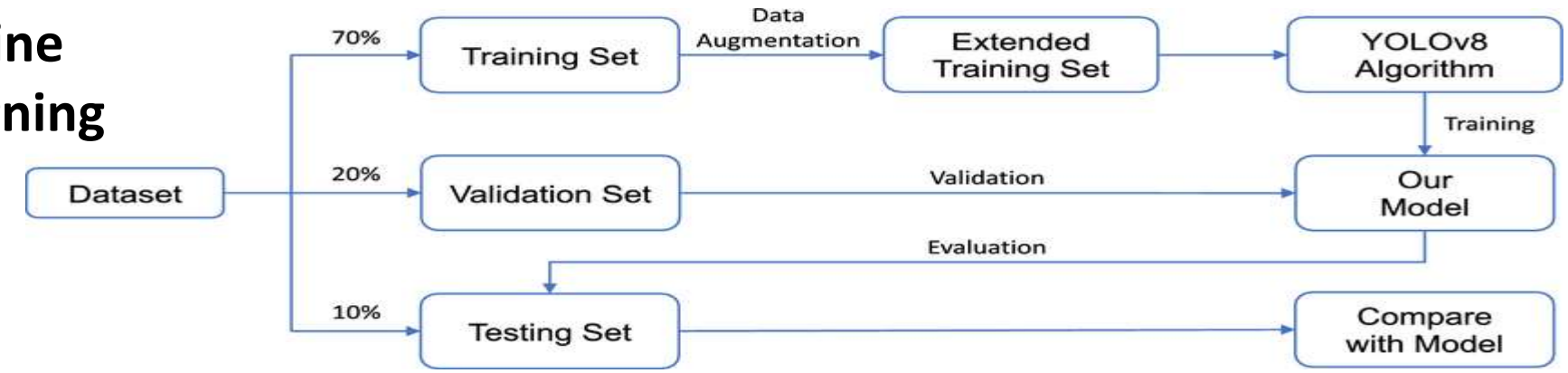
A.1. Learning Paradigms and their real Applications

Offline vs Online

Aspect	Offline (Batch)	Online (Incremental)
Data	Static	Streaming
Update	Periodic	Continuous
Algorithm	Batch GD	SGD
Adaptation	Slow	Fast
Example	Medical diagnosis	Fraud detection



Offline Learning



Training Patterns and Teaching Inputs:

What are Training Patterns?

- A **training pattern** = one example used to train a model
- Represented as:

$$(x, t)$$

Where:

- x: input features
- t: target (label/output)

Teaching Inputs (For Supervised Learning)

- Inputs are provided along with **correct output labels**
- Model learns mapping:

$$x \rightarrow t$$

Key Idea

- Model improves by minimizing **error between predicted and actual output**

Types of Training Inputs

- **Labeled data** → Supervised learning
- **Unlabeled data** → Unsupervised learning
- **Partially labeled** → Semi-supervised

Why Training Data is Important

•Determines how well model learns patterns

•Poor data → poor model

Key Factors

✓ **Quantity**

•More data → better generalization

✓ **Quality**

•Clean, noise-free data improves performance

✓ **Diversity**

•Covers all possible scenarios

Common Issues

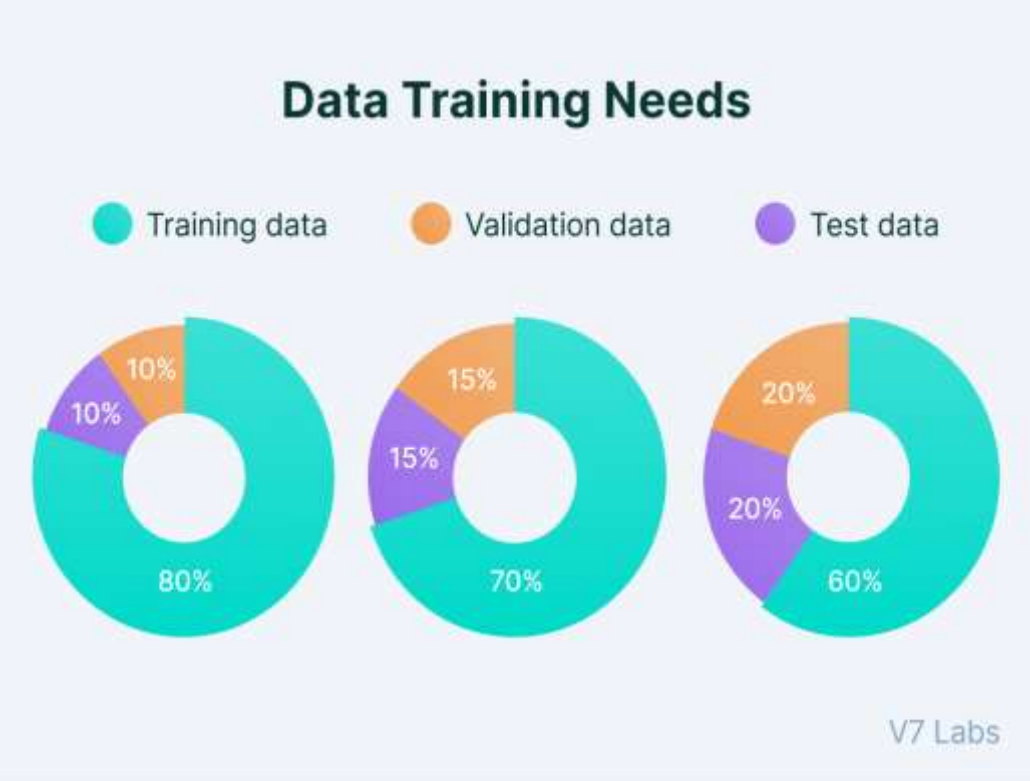
•Overfitting (memorizing data)

•Underfitting (not learning enough)

•Bias in data

Why Split Data?

To evaluate model on **unseen data**.



Three Types of Data

1. Training Set

- ❖ Used to **train model**
- ❖ Typically 60–80%

2. Validation Set

- •Used for **tuning parameters**
- •Helps avoid overfitting

3. Test Set

- •Used for **final evaluation**
- •Never seen during training

Typical Splits

- •70% Training / 15% Validation / 15% Testing
- •80% Training / 10% Validation / 10% Testing

Unit 2: Learning Paradigms

Supervised Learning

Labeled Data

Learn: $f(x) \rightarrow y$

Examples:

- Spam Detection
- Medical Diagnosis
- Face Recognition

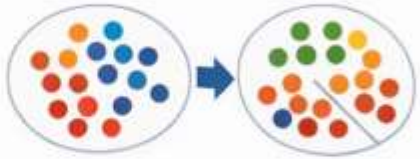
Unsupervised Learning

Unlabeled Data

Find Patterns & Clusters

Examples:

- *Customer Segmentation*
- *Anomaly Detection*
- *Feature Extraction*



Reinforcement Learning

Learn via Reward

Agent & Environment ● *Maximize Reward*

Online vs. Offline Learning

Offline Learning (Batch)

Train on Full Dataset

Online Learning

Real-Time Updates

EXAMPLES:

- *Stock Prediction*
- *Fraud Detection*