



# Introduction to Machine Learning

**WSS ML Workshop**

Hosein Hasani



WSS 2024

# Outline: Session 1

- Introduction and Motivation
- ML Problems
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning

# Machine Learning: Motivation

**Extensive influence** of Machine Learning across multitude of applications and everyday life:

- Computer Vision
- Signal Processing
- Audio and Speech Recognition
- Natural Language Processing
- Computational Social Science
- Control



# Machine Learning: Motivation

**Extensive influence** of Machine Learning across multitude of applications and everyday life:

- Computational Biology and Bioinformatics
- Medicine, Diagnosis and Health Care
- Computational Neuroscience
- Brain-Computer Interface
- Financial Forecasting
- Recommender Systems

# Machine Learning: Motivation

Why ML applications are growing?

- Improved machine learning algorithms
- Availability of data  
(Increased data capture, networking, ...)
- Algorithms too complex to write by hand
  - Demand for complex systems  
(high-dimensional, multi-modal, ...)
  - Demand for self-customization to user or environment



# Machine Learning: Concept

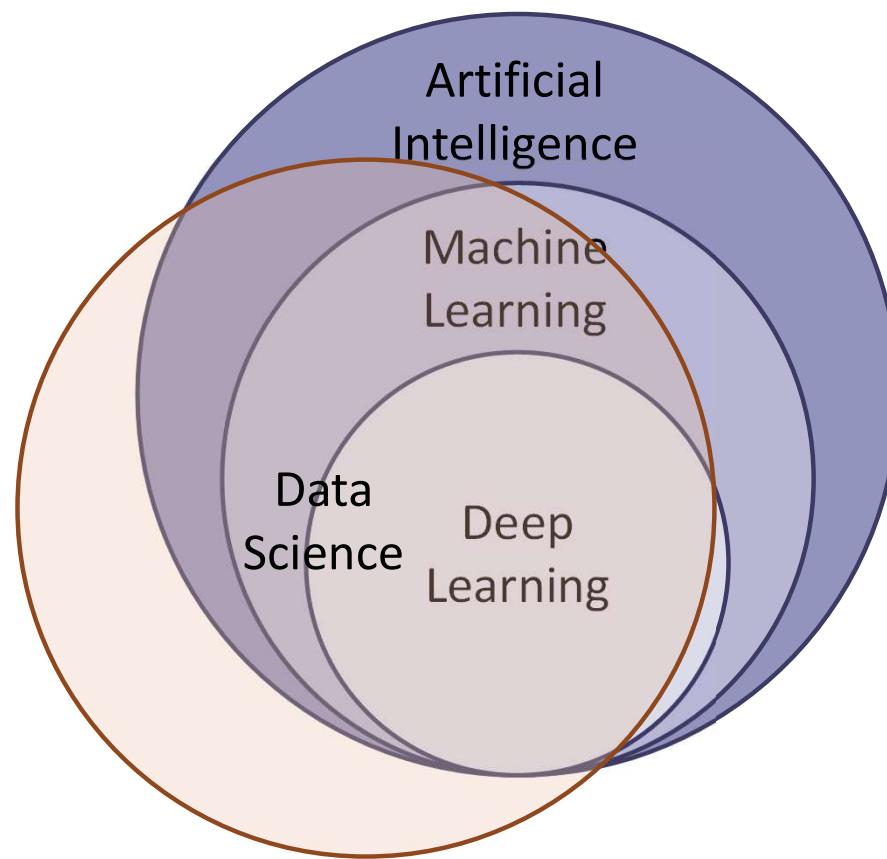
- Making machines learn!
- Using statistical models and algorithms to perform a specific task by **learning** data **patterns**, without being explicitly **programmed**
- **Generalization** to new unseen examples.

# Machine Learning: Main Recipe

- A pattern exist ...
- We do not know it mathematically!
- We have data on it :)

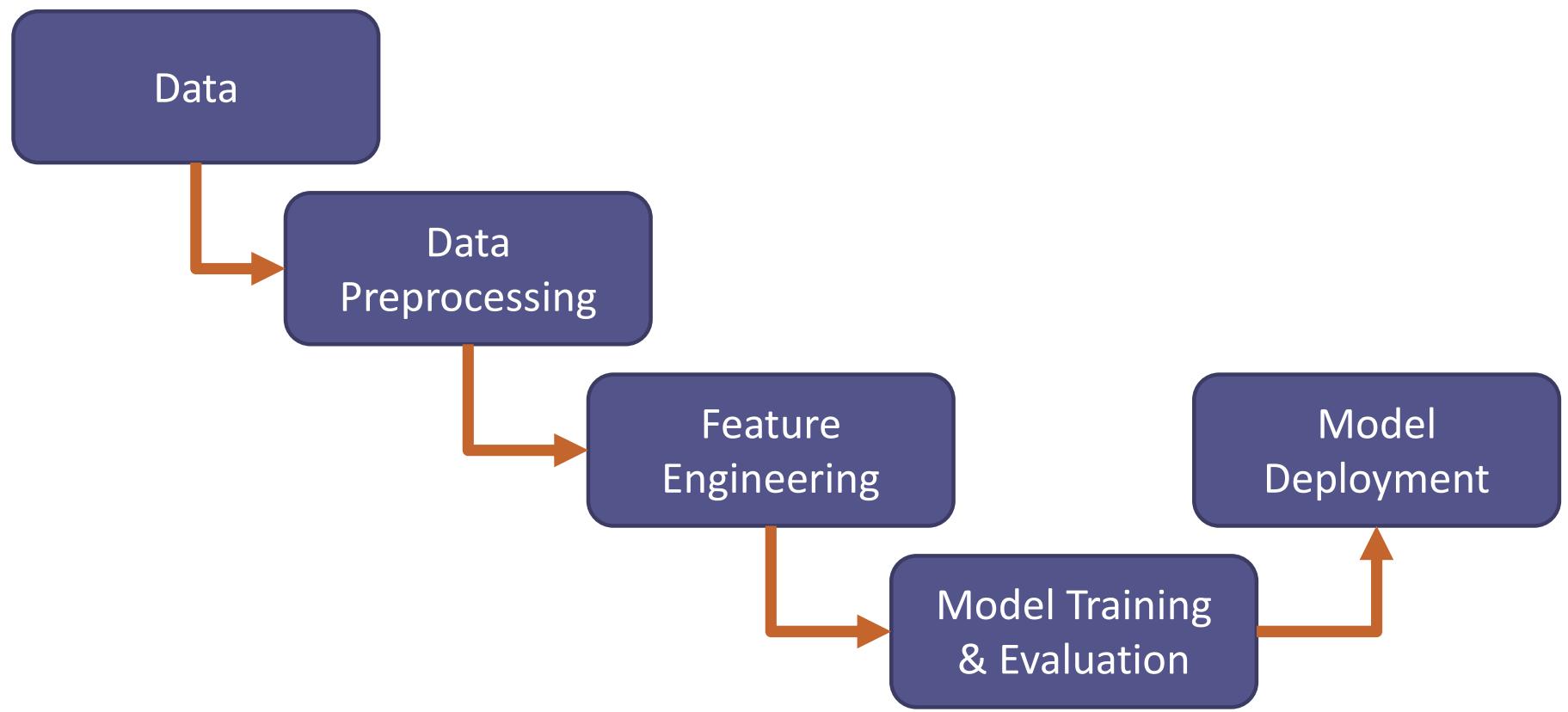
# Machine Learning: Role of Data

What's the scope?



# Machine Learning: Main Steps

- Typical steps:



# Main ML Problems

- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
  - Density Estimation
  - Generative Modeling
  - Clustering
  - Dimensionality Reduction
- Reinforcement Learning
  - Multi-armed Bandit



# Supervised Learning vs. Unsupervised Learning

- **Supervised learning**

Given: Training set

Labeled set of N input-output pairs  $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$

Goal: Learning a mapping from  $x$  to  $y$



# Supervised Learning vs. Unsupervised Learning

- **Supervised learning**

Given: Training set

Labeled set of N input-output pairs  $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$

Goal: Learning a mapping from  $x$  to  $y$

- **Unsupervised learning**

Given: Training set

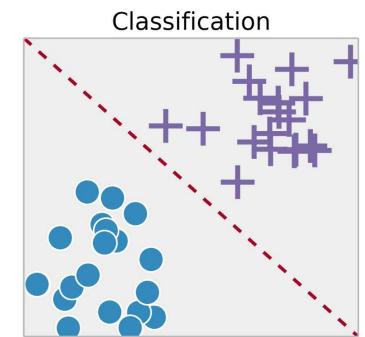
$$D = \{(x^{(i)})\}_{i=1}^N$$

Goal: Revealing structure in the observed data and finding groups or intrinsic structures in the data



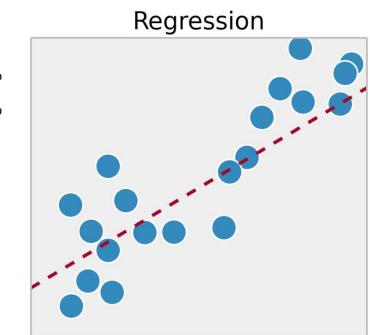
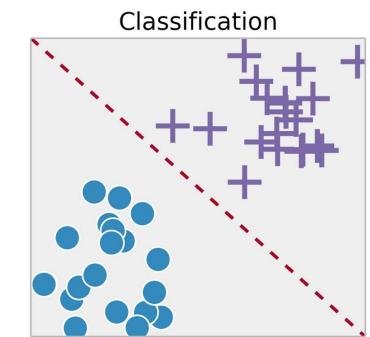
# Supervised Learning: Classification vs. Regression

- **Classification:** predict a **discrete** target variable e.g.  $y \in \{1, 2, \dots, C\}$



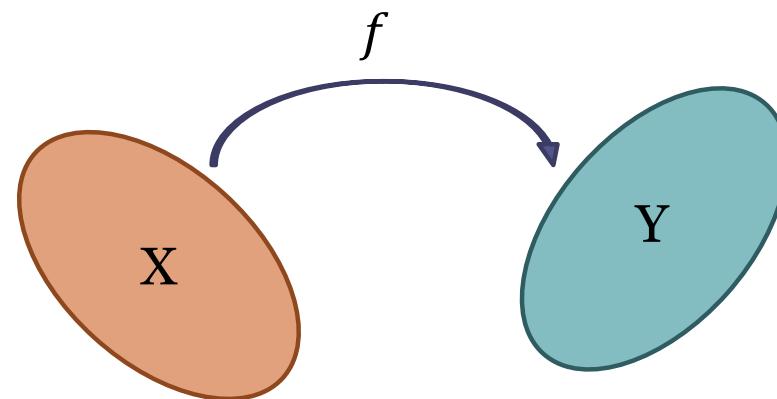
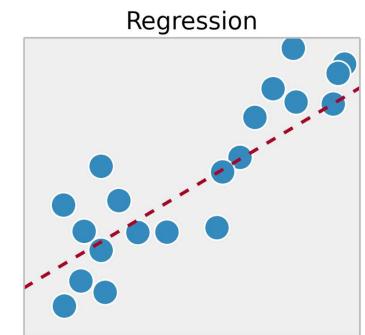
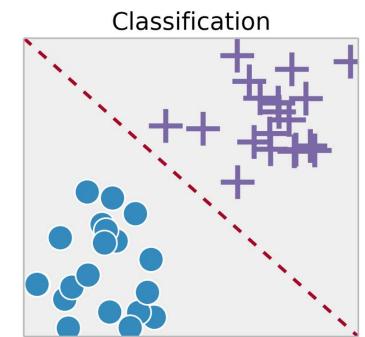
# Supervised Learning: Classification vs. Regression

- **Classification:** predict a **discrete** target variable e.g.  $y \in \{1, 2, \dots, C\}$
- **Regression:** predict a **continuous** target variable e.g.  $y \in [0, 1]$



# Supervised Learning: Classification vs. Regression

- **Classification:** predict a **discrete** target variable e.g.  $y \in \{1, 2, \dots, C\}$
- **Regression:** predict a **continuous** target variable e.g.  $y \in [0, 1]$

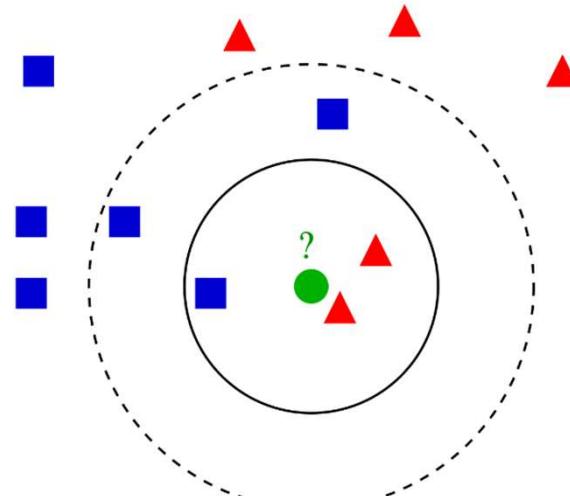


# Classification

- A function  $f : R^n \rightarrow \{1, \dots, k\}$  specifies which of  $k$  categories an input vector  $x$  belongs to.

# Classification

- A function  $f : R^n \rightarrow \{1, \dots, k\}$  specifies which of  $k$  categories an input vector  $x$  belongs to.
- Case Study: KNN (K Nearest Neighbors)
  - Stores all training cases and classify new cases based on similarity measure (like Euclidean distance)



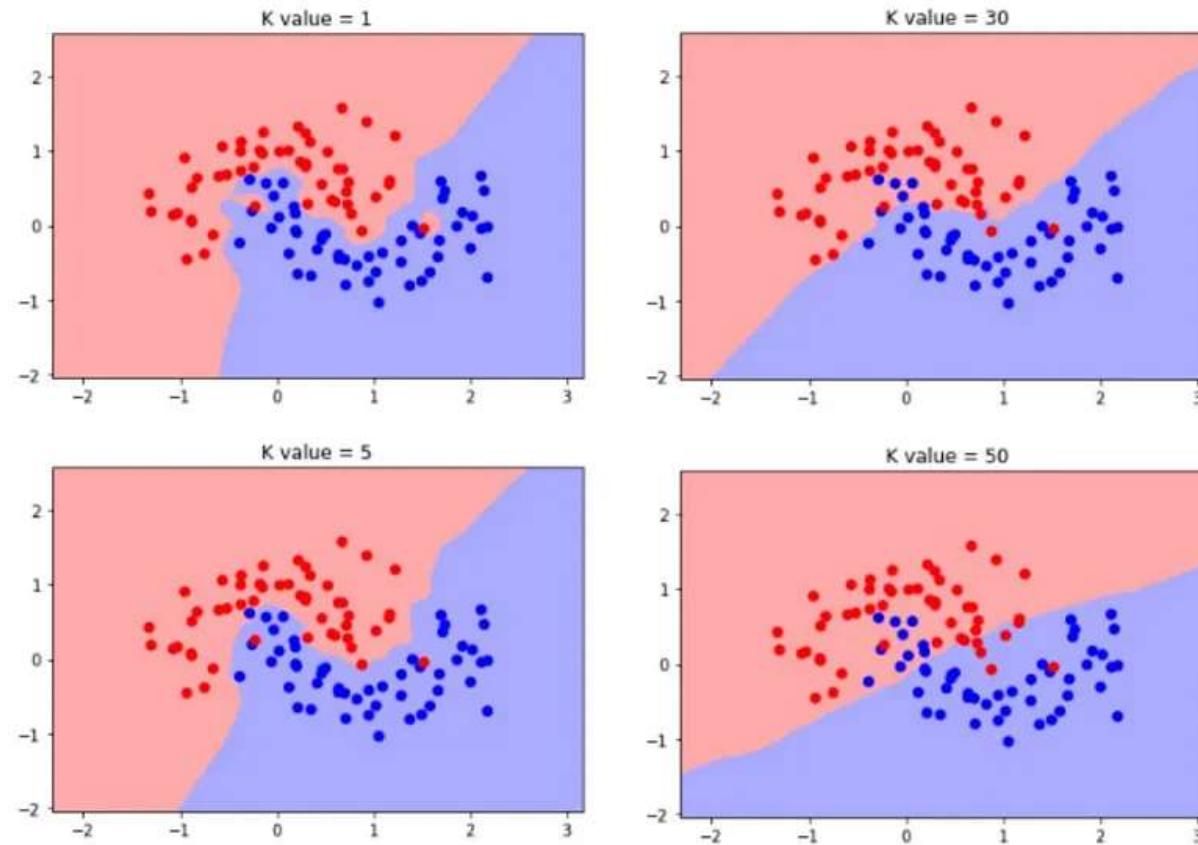
# Classification: KNN

- Given
  - Training data  $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}$  are simply stored.
  - Test sample:  $\mathbf{x}$
- To classify  $\mathbf{x}$ :
  - Find  $k$  nearest training samples to  $\mathbf{x}$
  - Identify the number of samples  $k_j$  belonging to class  $\mathcal{C}_j$
  - Assign  $\mathbf{x}$  to the class  $\mathcal{C}_{j^*}$  where  $j^* = \operatorname{argmax}_{j=1,\dots,c} k_j$



# Classification: KNN

- Effect of  $K$  on decision boundaries



# Classification

More advanced applications:



Object Recognition

# Regression

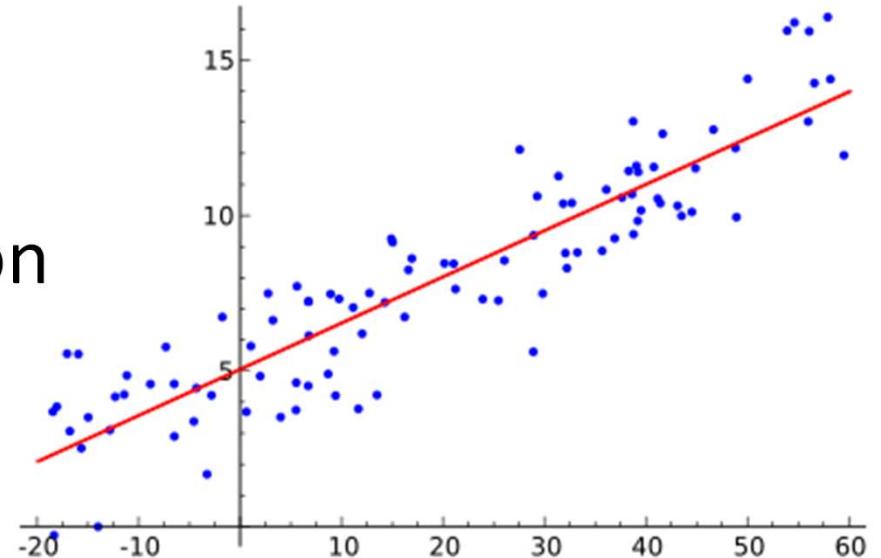
- A function  $f : R^n \rightarrow R$  that maps an input vector  $x$  to a continuous value  $y$ .

# Regression

- A function  $f : R^n \rightarrow R$  that maps an input vector  $x$  to a continuous value  $y$ .

- Case study: Linear Regression

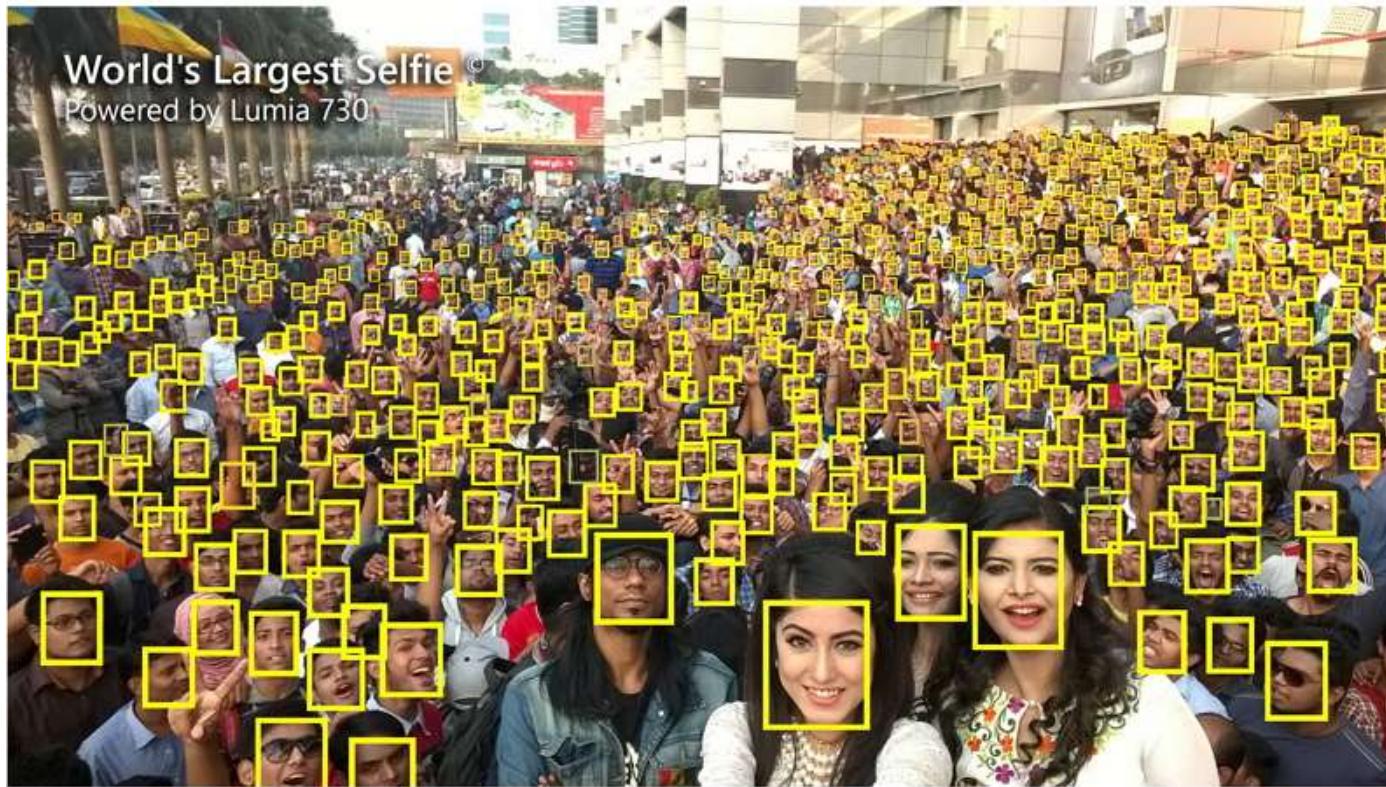
$$f(x; w) = w_0 + w_1 x$$



$w = [w_0, w_1]$  : Parameters that are estimated during optimization

# Regression

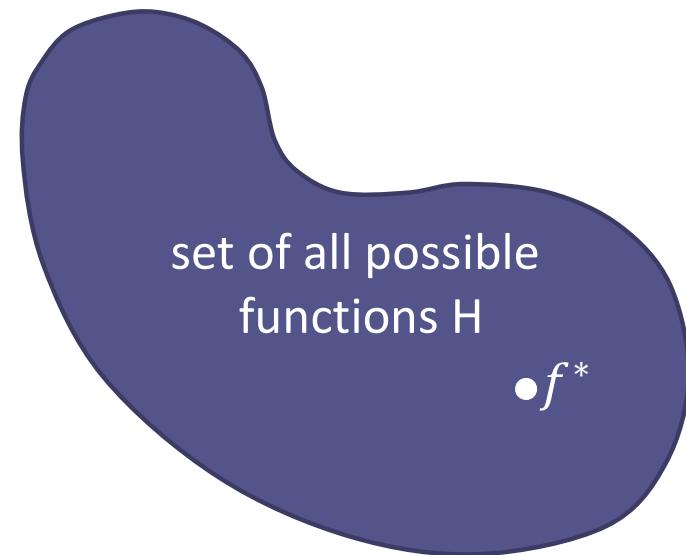
- More advanced applications:



Object Detection

# Hypothesis Class and Inductive Bias

- The aim of supervised learning is to find  $f^*$  (best solution) from **hypothesis space** (e. g. the set of all possible functions)

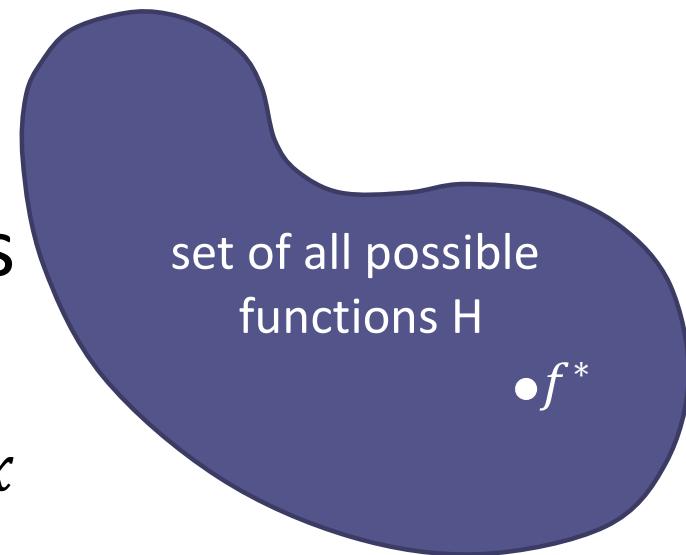


# Hypothesis Class and Inductive Bias

- The aim of supervised learning is to find  $f^*$  (best solution) from **hypothesis space** (e. g. the set of all possible functions)
- Example:

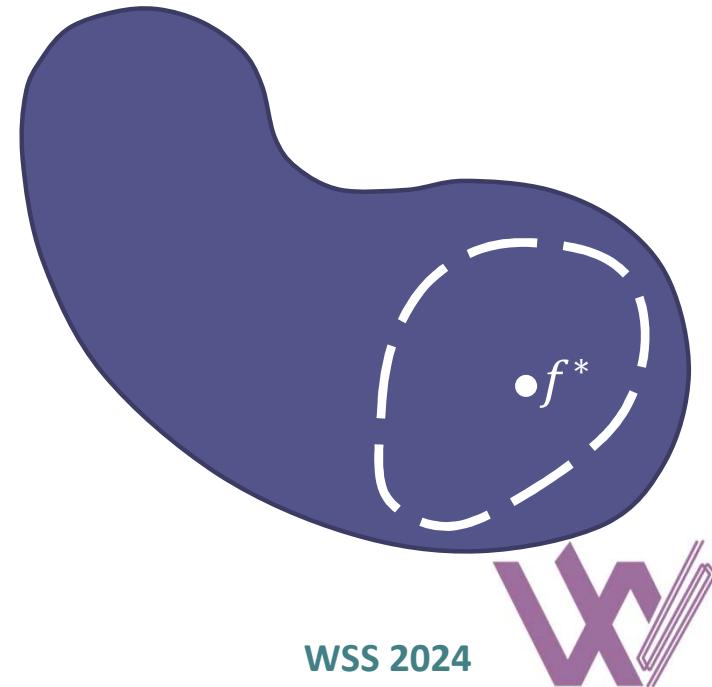
In linear regression the hypothesis space include all possible functions with the form of

$$f(x; w_0, w_1) = w_0 + w_1 x$$



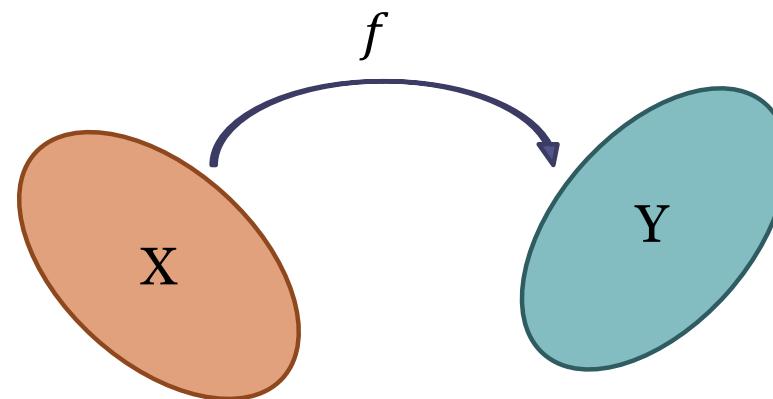
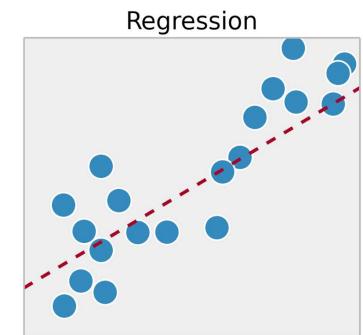
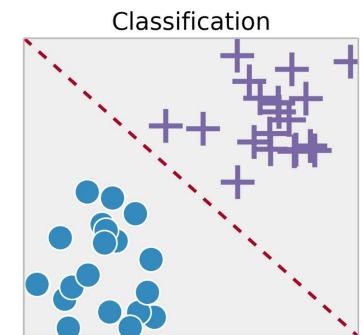
# Hypothesis Class and Inductive Bias

- **Inductive bias** is the set of assumptions that a learner uses to predict outputs of given inputs.
- Some times we use our knowledge about the nature of data to **restrict** the hypothesis space.



# Supervised Learning (Recap)

- **Classification:** predict a **discrete** target variable e.g.  $y \in \{1, 2, \dots, C\}$
- **Regression:** predict a **continuous** target variable e.g.  $y \in [0, 1]$



# Unsupervised Learning

## Unsupervised learning

Given: Training set

$$D = \{(x^{(i)})\}_{i=1}^N$$

Goal: Revealing structure in the observed data and finding groups or intrinsic structures in the data

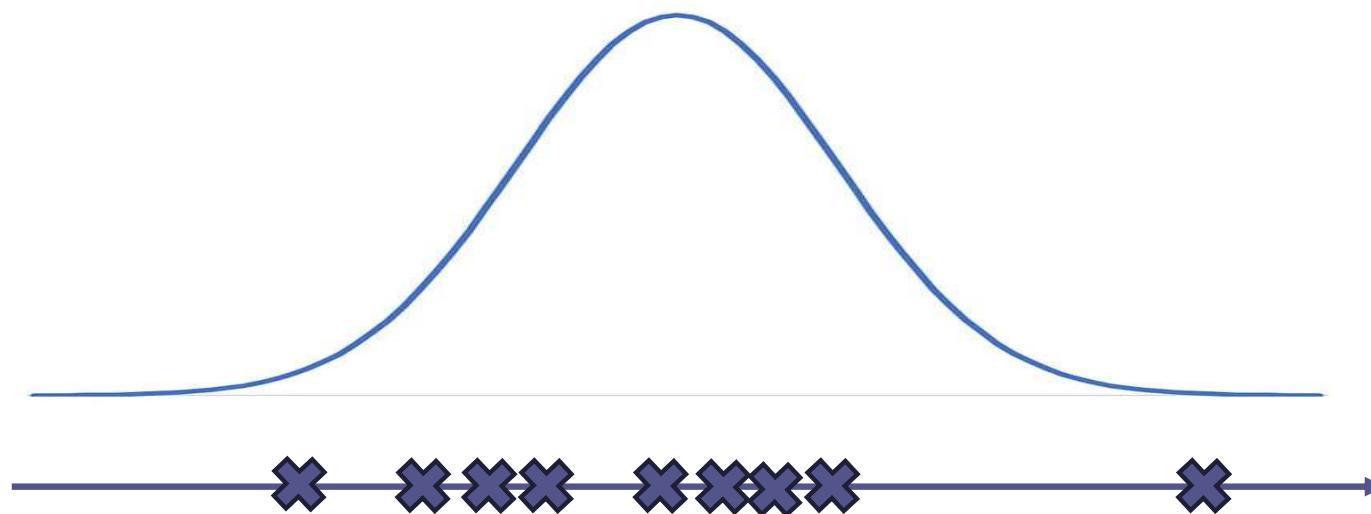
## Main Approaches:

- Density Estimation
- Generative Modelling
- Clustering
- Dimensionality Reduction



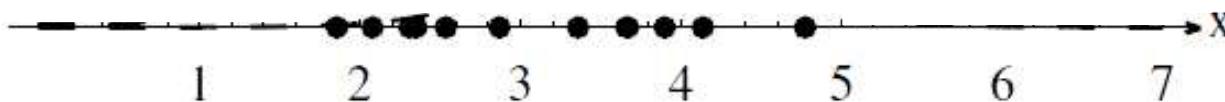
# Density Estimation

- Estimating the probability density function  $p(x)$ , given a set of data points  $\{x^{(i)}\}_{i=1}^N$  drawn from it.



# Density Estimation

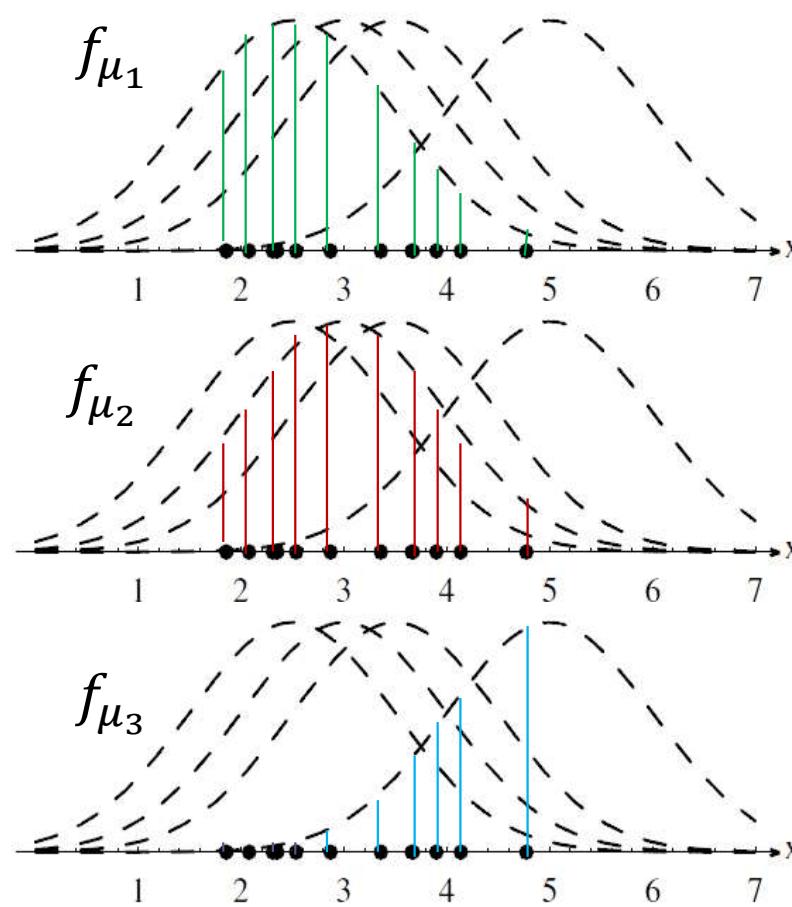
Example: Normal distribution



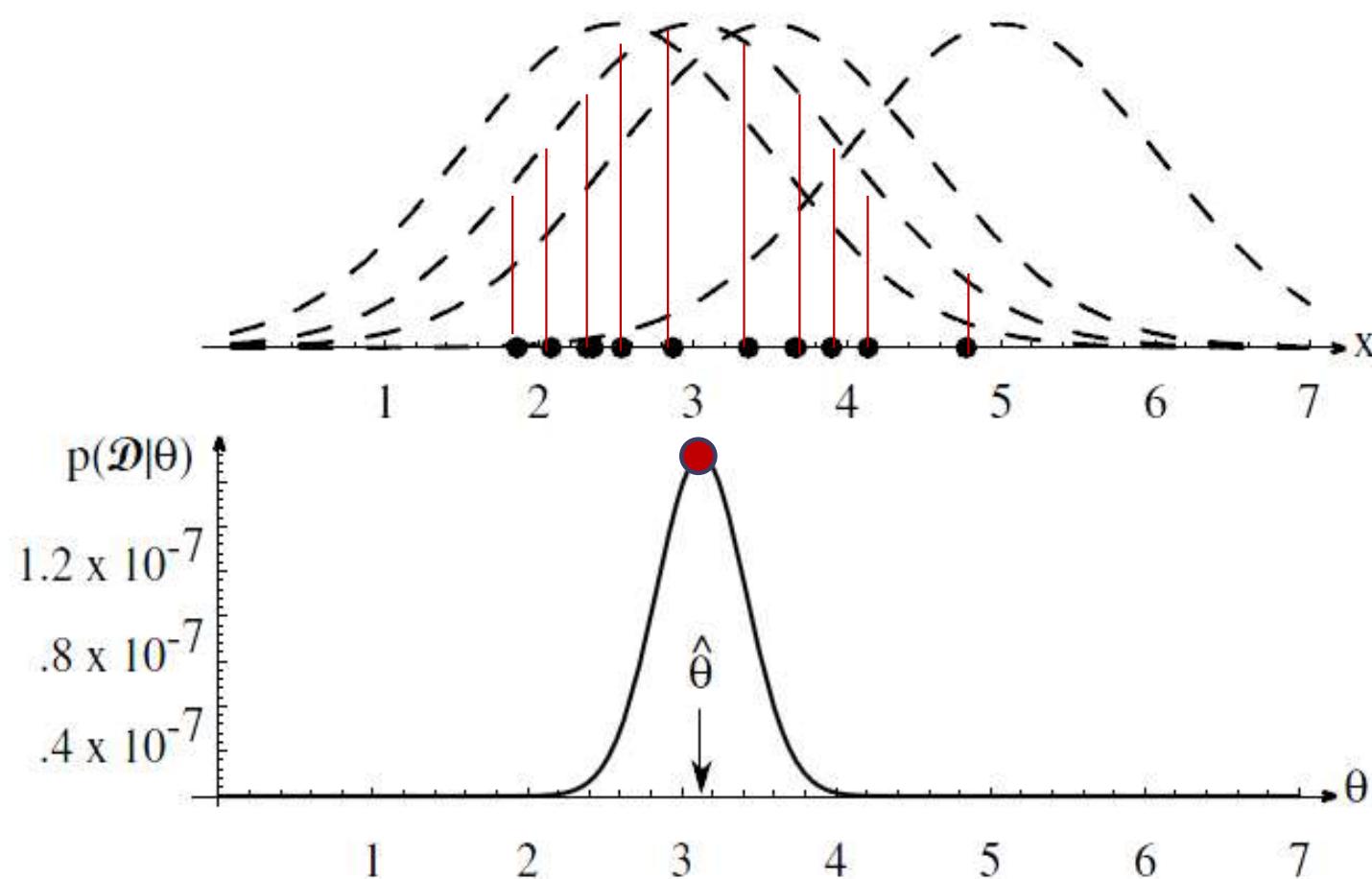
$$p_\mu(x) = N(x; \mu, 1)$$

$$p_\mu(x^{(1)}, x^{(2)}, \dots, x^{(N)}) = \prod_{i=1}^N p_\mu(x^{(i)})$$

# Density Estimation: Normal distribution

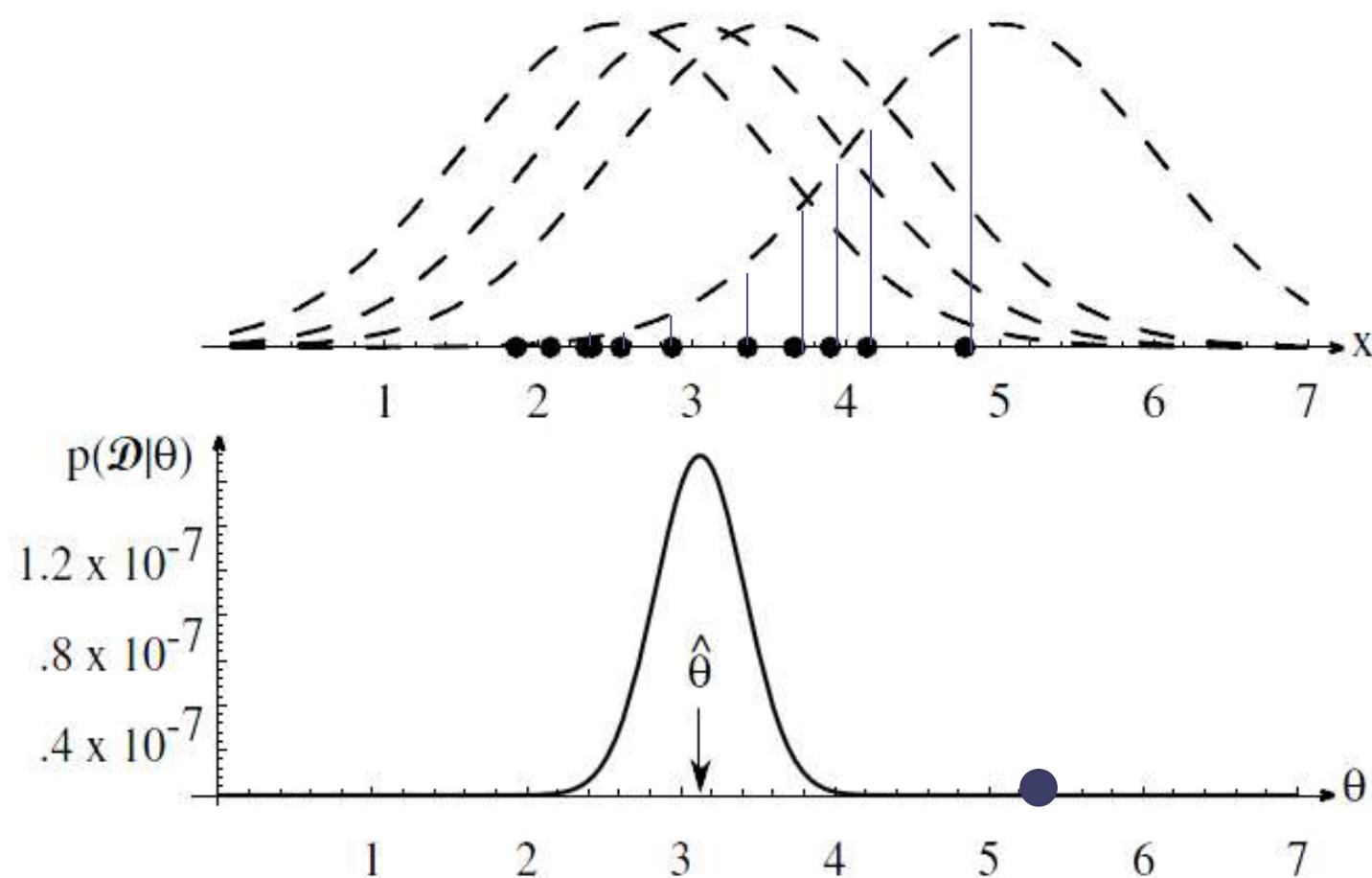


# Density Estimation: Normal distribution



$\hat{\theta}$  best agrees with the observed samples

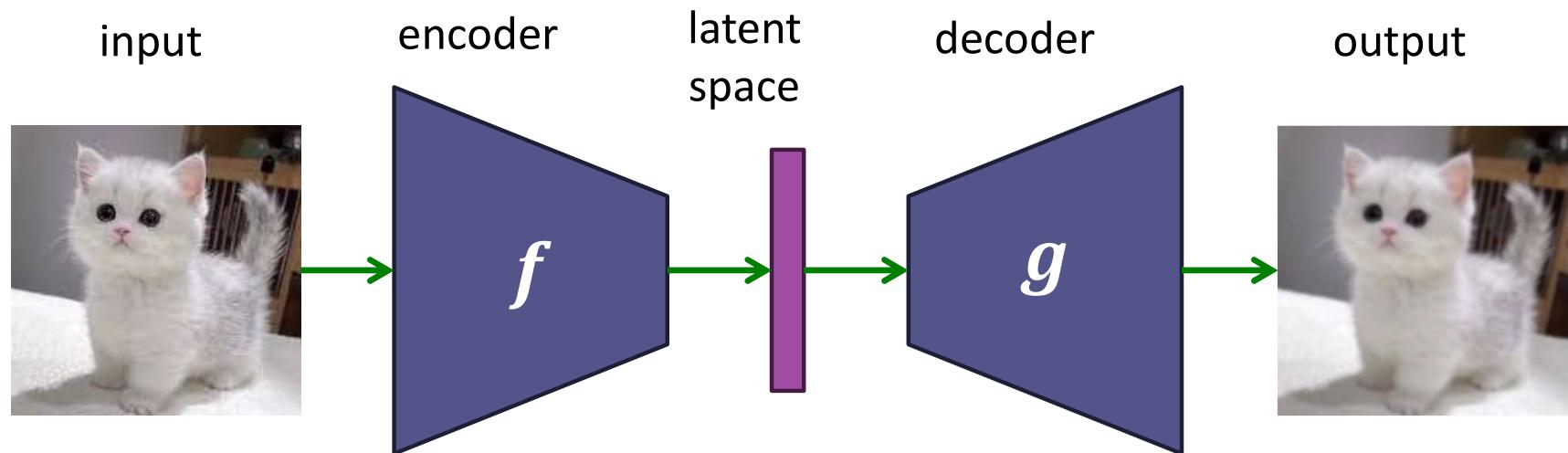
# Density Estimation: Normal distribution



$\hat{\theta}$  best agrees with the observed samples

# Density Estimation

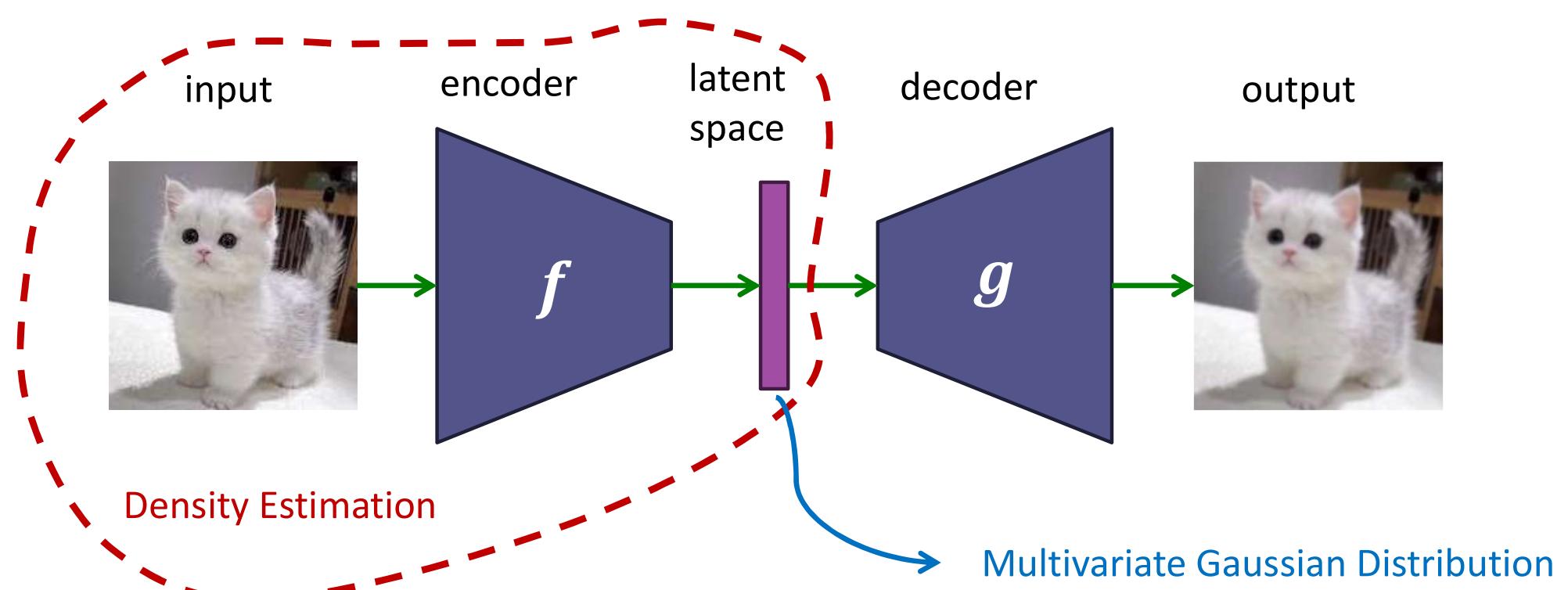
More sophisticated applications:



Variational Autoencoder (VAE)

# Density Estimation

More sophisticated applications:

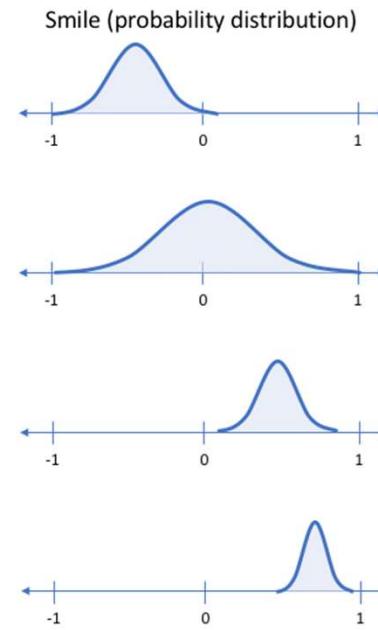
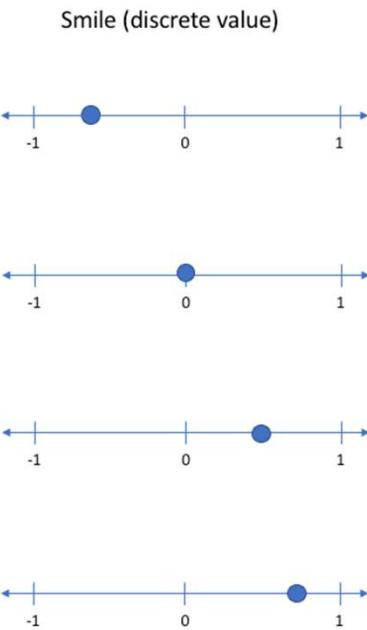


Variational Autoencoder (VAE)

# Density Estimation

More sophisticated applications:

latent variable value



Variational Autoencoder (VAE)

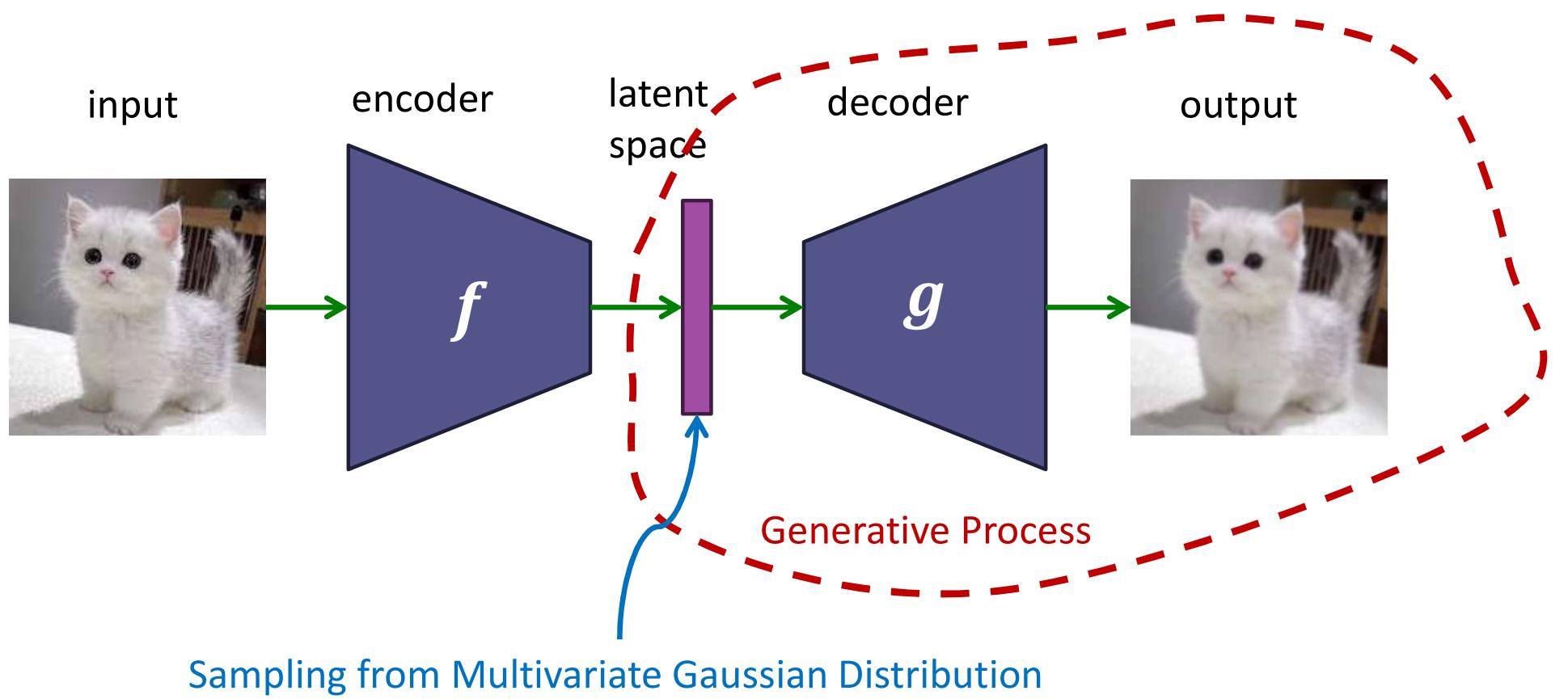
# Generative Modeling

- Sampling new data points based on observed data distribution.
- This can be achieved with / without explicitly learning density function  $p(x)$ .



# Generative Modeling:

More sophisticated applications (explicit density estimation)



Variational Autoencoder (VAE)

# Generative Modeling:

More sophisticated applications

- Generative Adversarial Networks (implicit density estimation)
- Diffusion Models



# Generative Modeling:

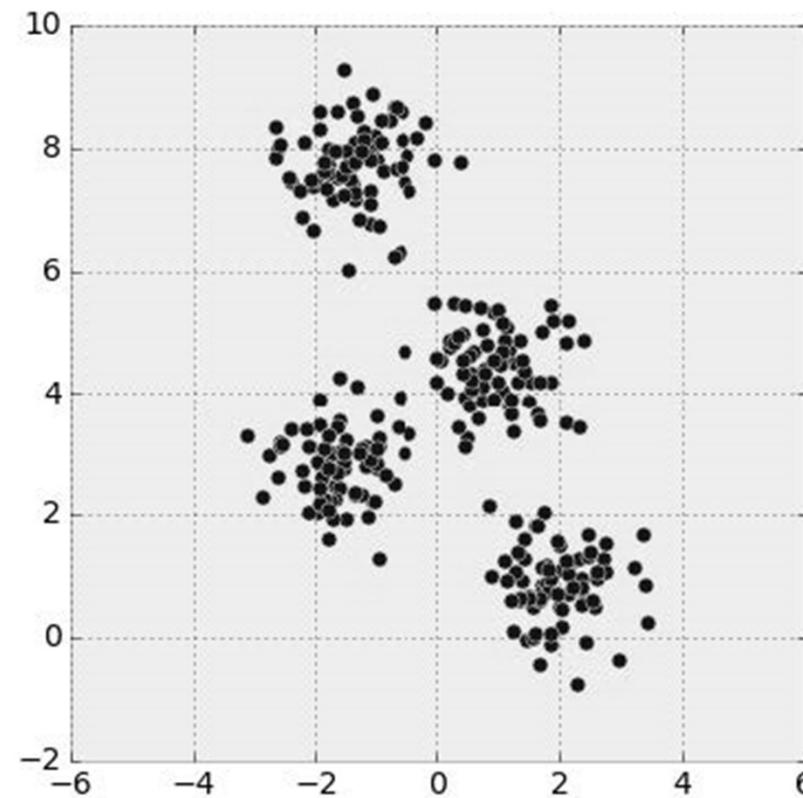
## More sophisticated applications

- Large Language Models (LLMs)
- Generative pre-trained transformers (GPT)

The screenshot shows a Microsoft Copilot interface. At the top, there's a user profile icon labeled 'You', a name 'Hosein' with a person icon, a trophy icon, and a three-line menu icon. In the main area, a user prompt asks: 'Write a poem about a PhD student named Hosein, who stayed awake all night to prepare slides for the WSS workshop!'. Below this, the 'Copilot' AI responds: 'Certainly! Here's a succinct tribute to Hosein, the tireless PhD student:'. It then generates a poem:  
**Hosein's Vigil**  
In the quiet corridors of academia, Where knowledge blooms like distant stars,  
Lived a diligent soul named Hosein, A seeker of wisdom, fueled by late-night  
hours.  
His days blurred into nights, Each keystroke etching equations and prose, For the  
WSS workshop, he labored, Crafting slides that would illuminate minds.  
The moon witnessed his dedication, As he wove pixels into cosmic constellations,  
Syntax danced, charts whispered secrets, And the coffee machine hummed its  
caffeinated hymn.  
When dawn tinted in casting shadows, Hosein stood tall, weary yet triumphant.

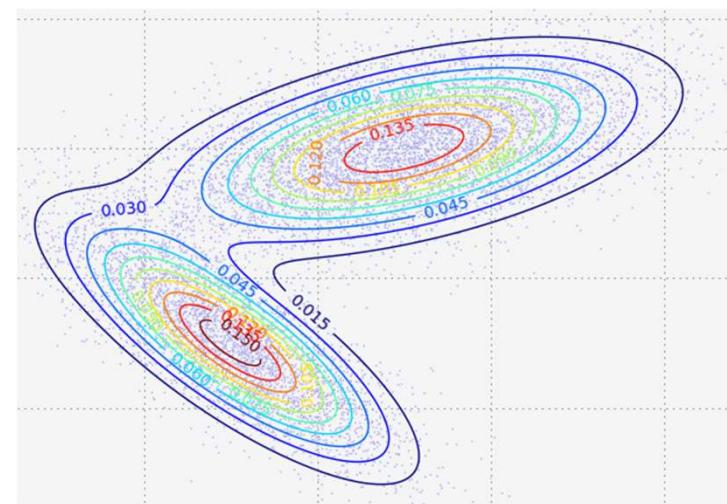
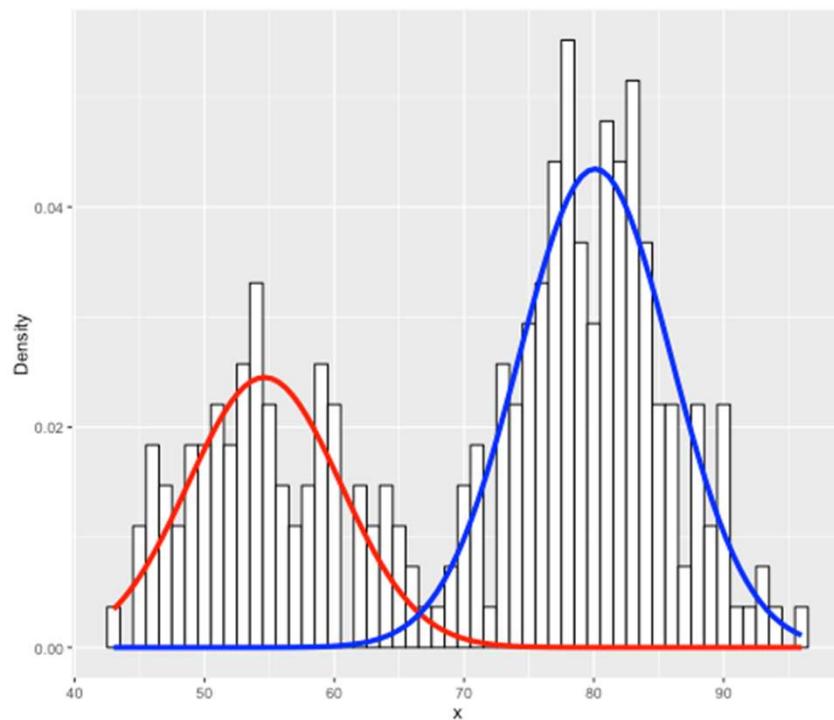
# Clustering

- A technique to assign each point into a specific group.



# Clustering

- A technique to assign each point into a specific group.



Gaussian Mixture Model

# Clustering: K-means Algorithm

## K-means Algorithm:

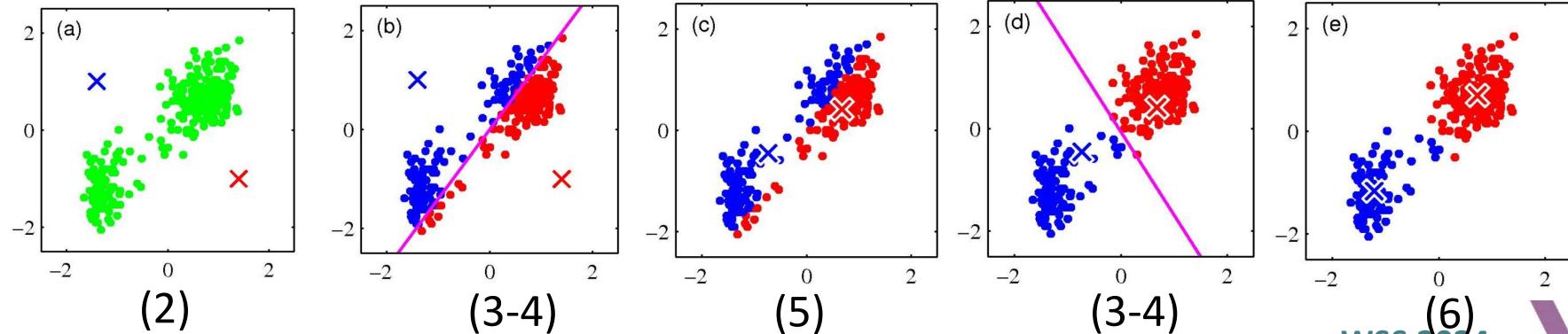
1. Choose number of clusters K.
2. Pick K random points as cluster centers (centroid)
3. Compute the distance between data points and all centroids
4. Assign each data point to the closest centroid
5. Compute the centroids for the clusters (by averaging)
6. Iterate steps 3-5 until convergence (no centroid change)



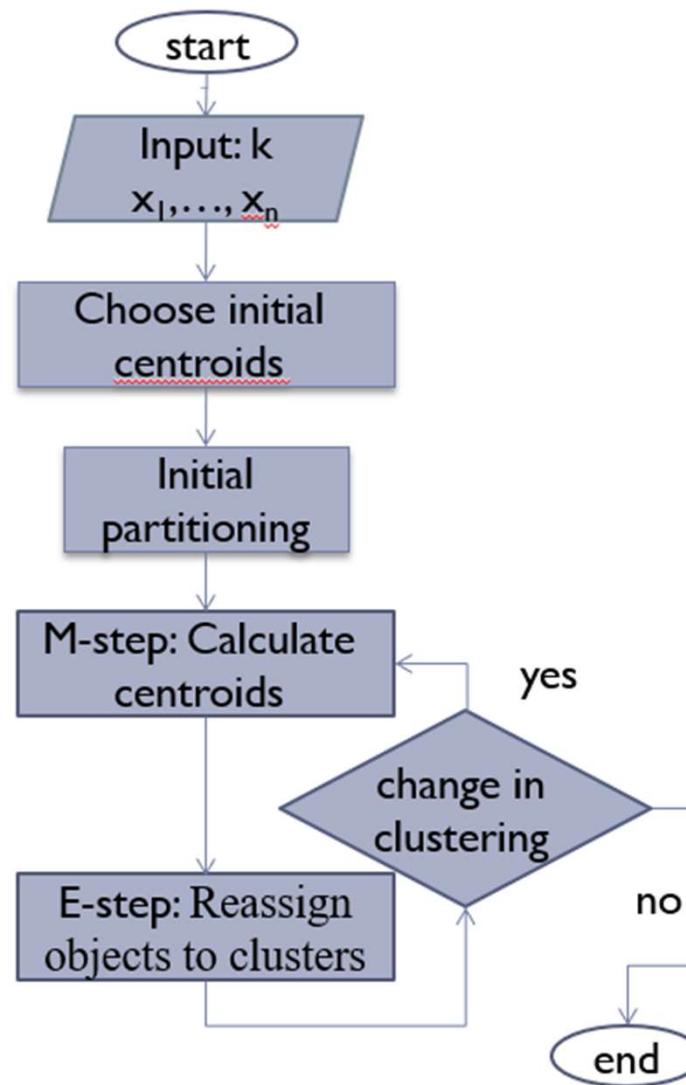
# Clustering: K-means Algorithm

## K-means Algorithm:

1. Choose number of clusters K.
2. Pick K random points as cluster centers (centroid)
3. Compute the distance between data points and all centroids
4. Assign each data point to the closest centroid
5. Compute the centroids for the clusters (by averaging)
6. Iterate steps 3-5 until convergence (no centroid change)

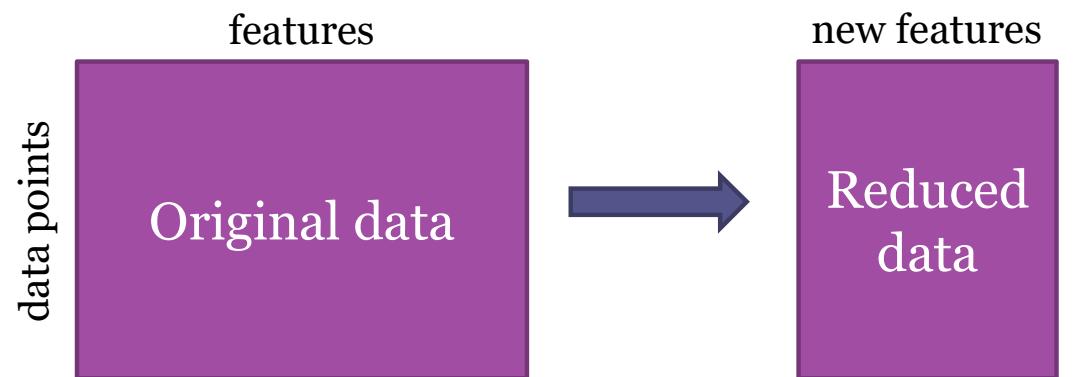


# Clustering: K-means Algorithm



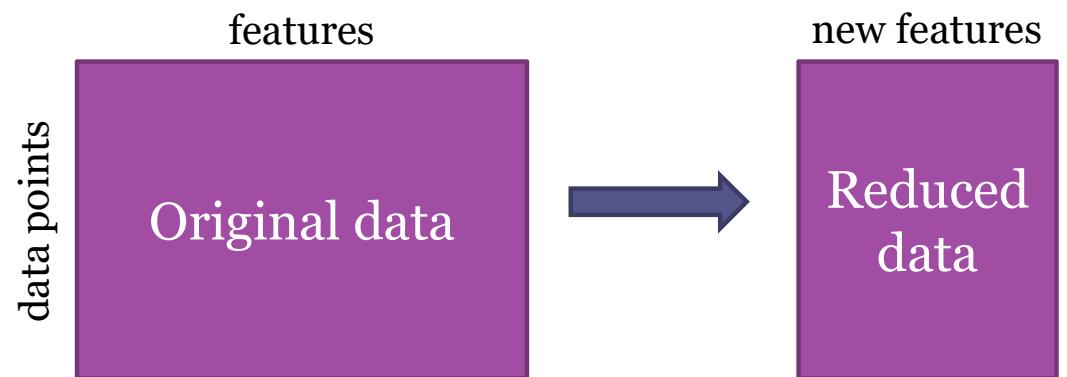
# Dimensionality Reduction

- A technique to find a lower-dimensional representation of data features that preserves some of its properties.



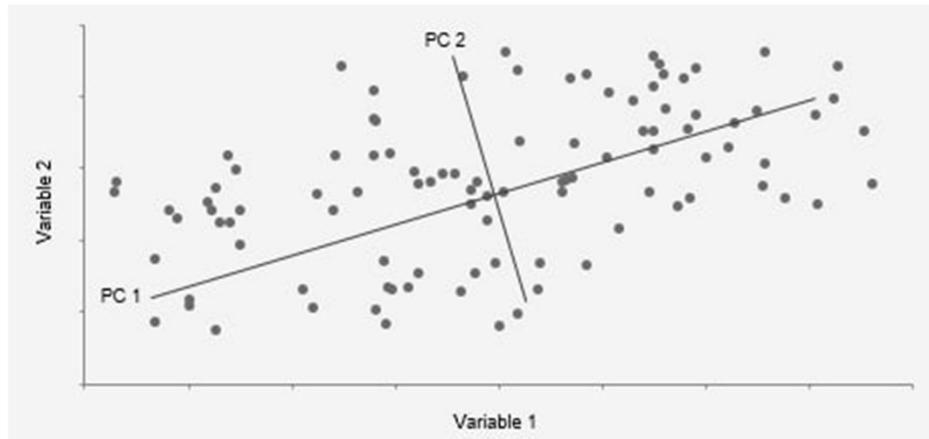
# Dimensionality Reduction

- A technique to find a lower-dimensional representation of data features that preserves some of its properties.
- Motivations:
  - Computation
  - Visualization
  - Feature extraction



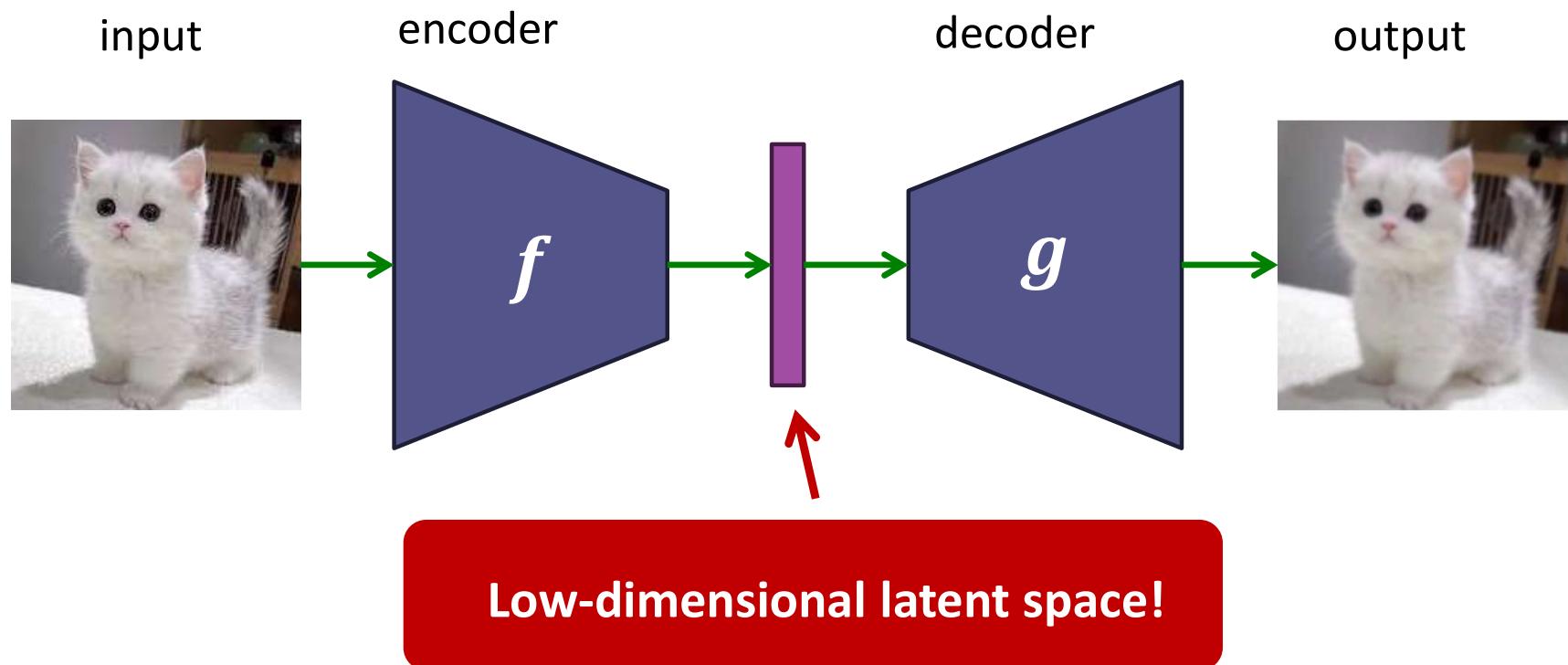
# Dimensionality Reduction

- A technique to find a lower-dimensional representation of data features that preserves some of its properties.
- Case Study:  
Principal Component Analysis (PCA)



# Dimensionality Reduction

More sophisticated methods:



Variational Autoencoder (VAE)

# Unsupervised Learning (Recap)

## Unsupervised learning

Given: Training set

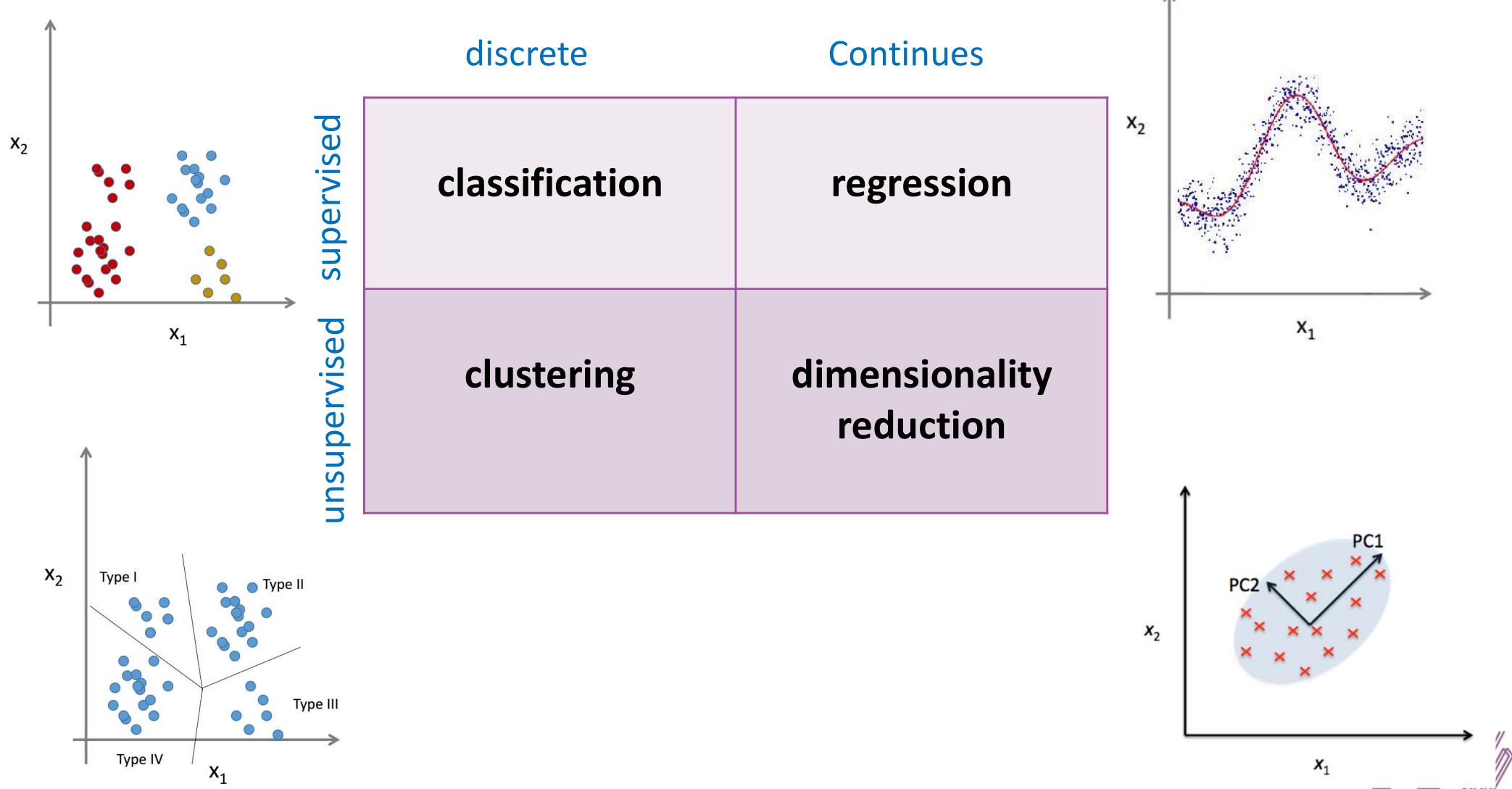
$$D = \{(x^{(i)})\}_{i=1}^N$$

Goal: Revealing structure in the observed data and finding groups or intrinsic structures in the data

- Main Approaches:
  - Density Estimation
  - Generative Modelling
  - Clustering
  - Dimensionality Reduction



# Supervised Learning vs. Unsupervised Learning



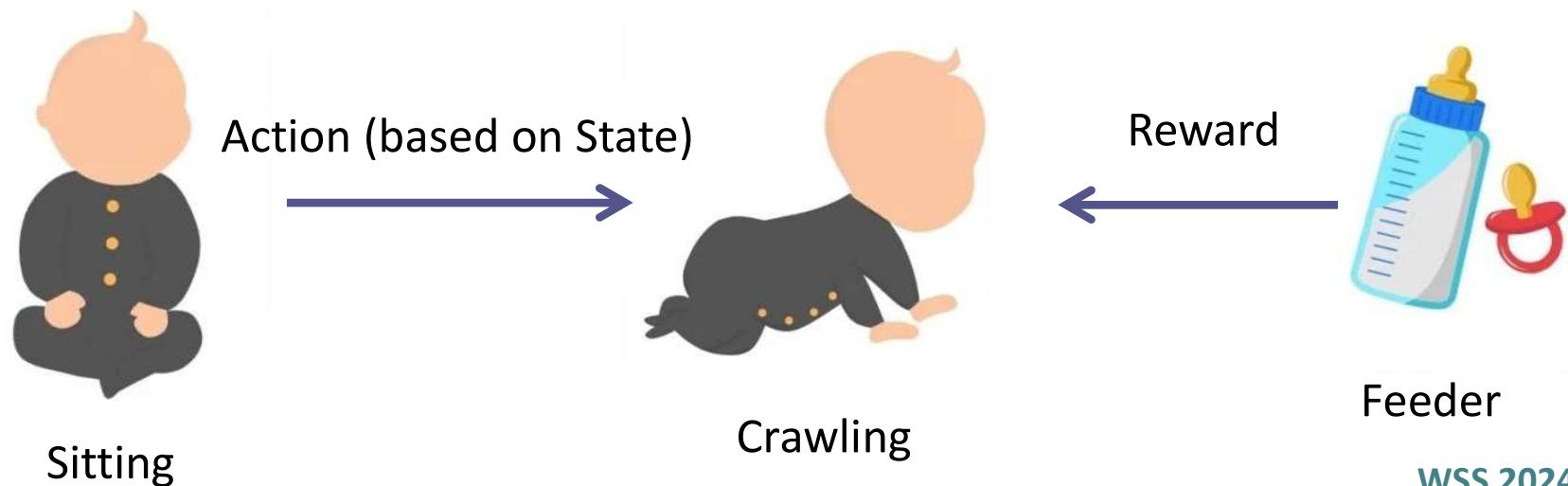
# Outline: Session 2

- Introduction and Motivation
- ML Problems
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning
- Loss Function and Optimization

# Reinforcement Learning

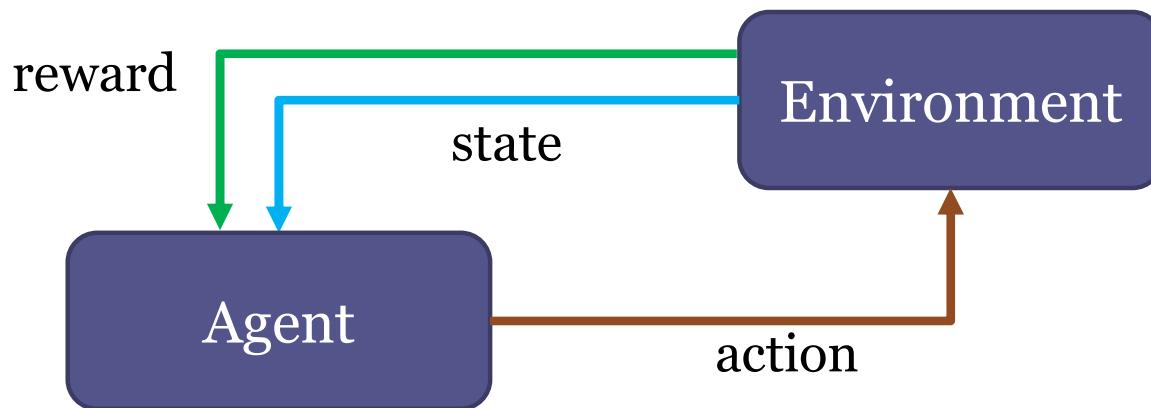
- Most natural way of learning
- Examples:
  - Baby movement
  - Investment

Agent (Baby)



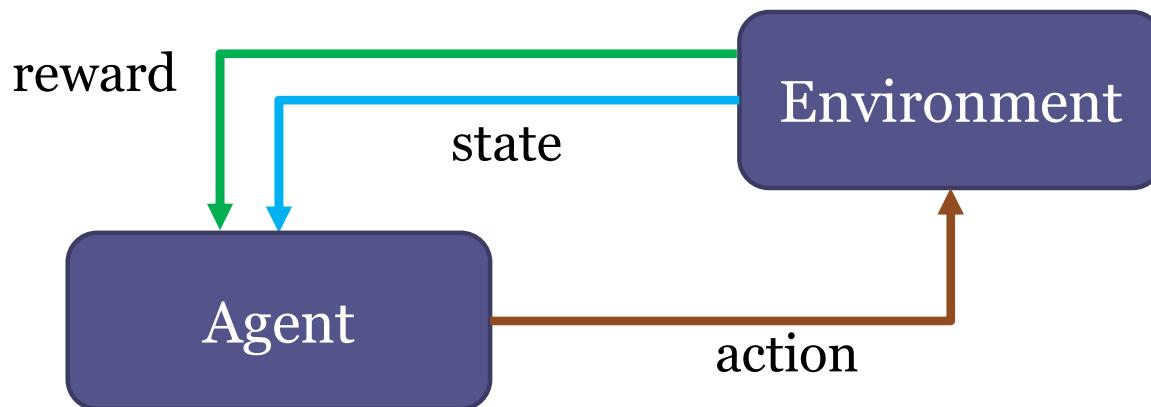
# Reinforcement Learning

- Sequential decision making with (possibly delayed) rewards
- An agent learns appropriate actions (policy) based on environment feedbacks to maximize reward.



# Reinforcement Learning

- Sequential decision making with (possibly delayed) rewards
- Data in supervised learning:  
 $(\text{input}, \text{label})$
- Data in reinforcement learning:  
 $(\text{input}, \text{some output}, \text{a grade of reward for this output})$



# Reinforcement Learning

- State: Agent's observation from the world
- Environment model:
  - Transition probability  $p(s_{t+1}|s_t, a_t)$
  - Reward function  $R(s_t, a_t, s_{t+1})$
- Policy: Mapping from states to actions

$$\pi_\theta: S \rightarrow A$$

- Goal: Learning an optimal policy in order to maximize its long-term reward

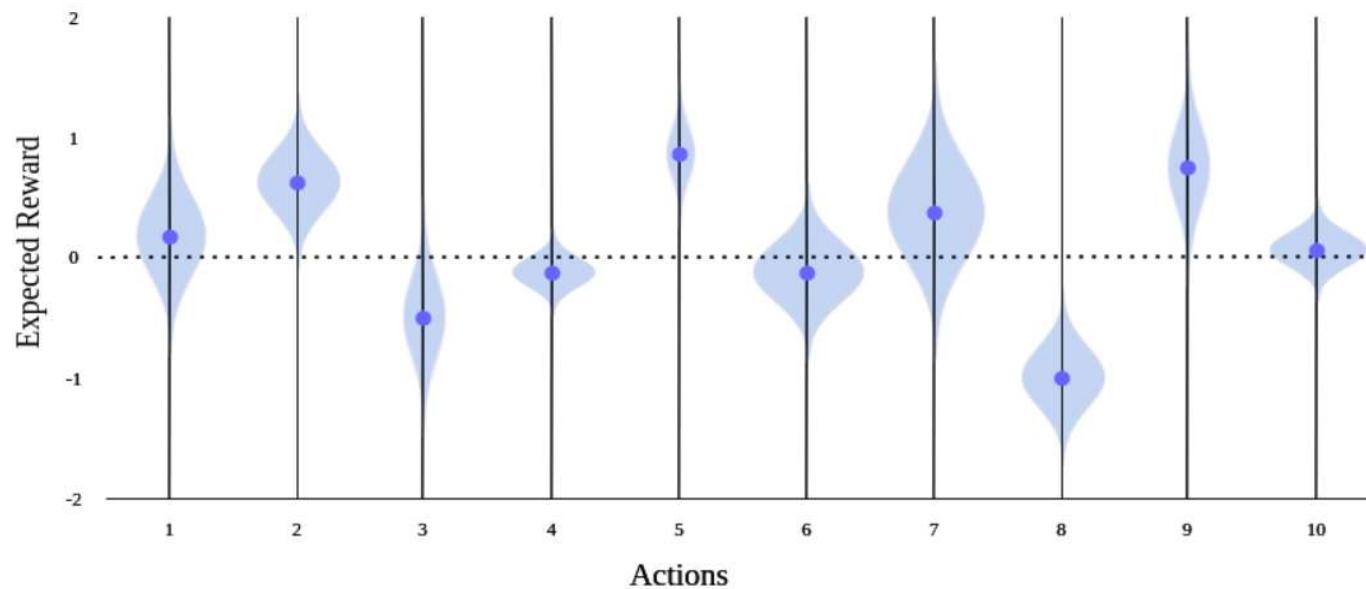
# Multi-Armed Bandit



Multiple bandits with unknown average rewards

# Multi-Armed Bandit

- Finding the **best arm** (in the sense of expected reward) with minimum trial and error.
- Minimizing cumulative **regret**.
- **Exploration-exploitation** trade-off!

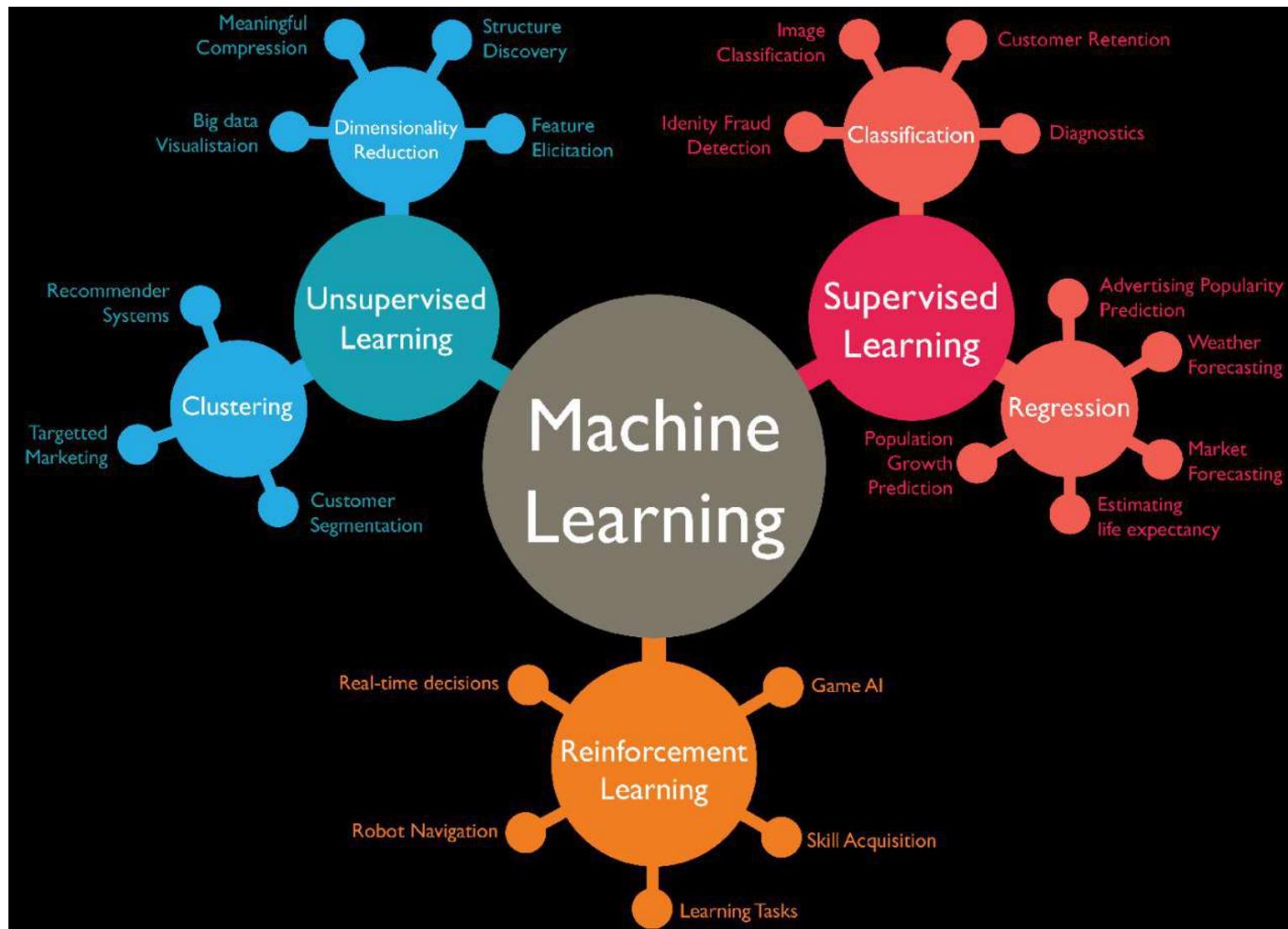


# Multi-Armed Bandit

Applications:

- Online advertisement
- Recommender systems
- Clinical trials
- Mining
- Network (packet routing)

# Primary ML Problems (Review)



# Outline: Session 3

- Introduction and Motivation
- ML Problems
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning
- Loss Function and Optimization
- Generalization and Overfitting
- Neural Networks

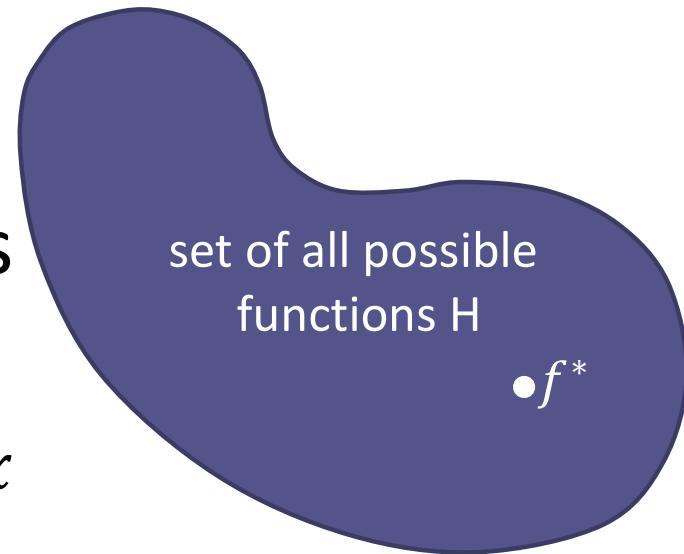


# Hypothesis Class (Recap)

- The aim of supervised learning is to find  $f^*$  (best solution) from **hypothesis space** (e. g. the set of all possible functions)
- Example:

In linear regression the hypothesis space include all possible functions with the form of

$$f(x; w_0, w_1) = w_0 + w_1 x$$



# Loss Function and Optimization

- **Loss Function:**

Quantifies how much undesirable is each parameter vector across the training data.

- **Optimization:**

Apply an **optimization** algorithm that finds the parameters that minimize the loss function.

# Steps of Learning Procedure

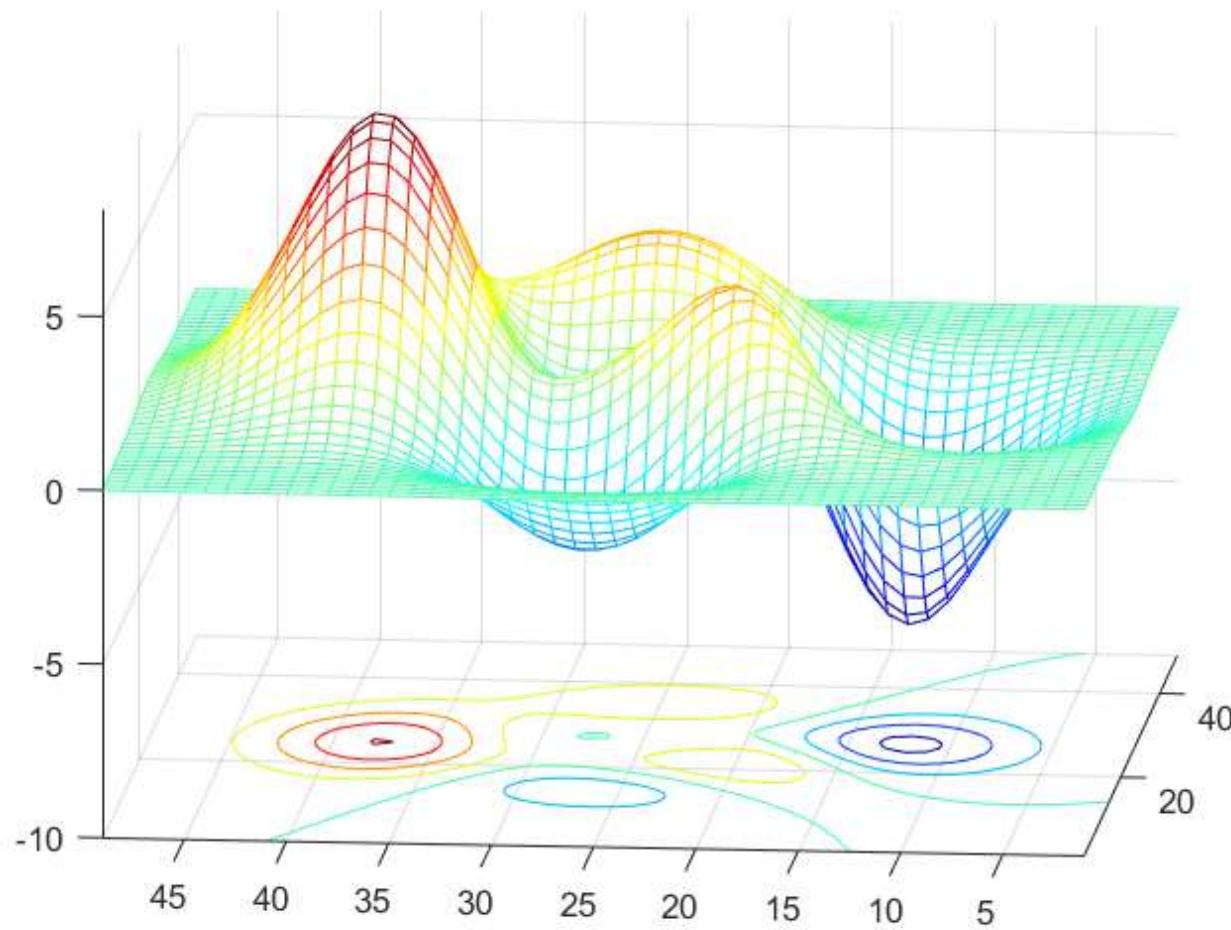
Typical steps of solving (supervised) learning problems:

- Select the **hypothesis space**:
- Define a **loss function** that quantifies how much undesirable is each parameter vector across the training data.
- Apply an **optimization** algorithm that efficiently finds the parameters that minimize the loss function.
- **Evaluate** the obtained model.

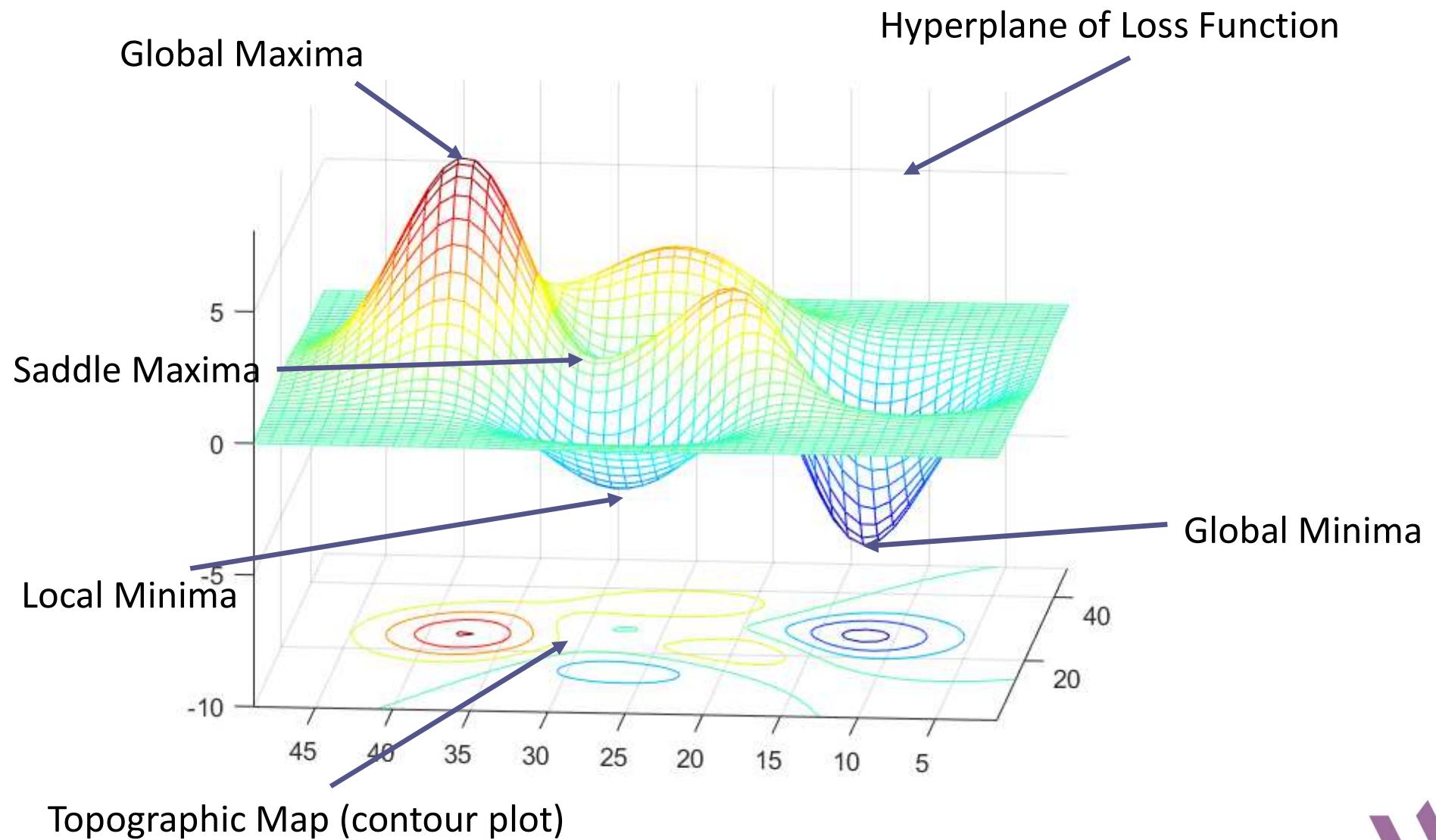
# Loss Function

- Error: The **difference** between the actual outputs (e.g. **ground truth**) and the predicted outputs.
- The function that is used to compute this error is known as Loss Function  $L(\cdot)$  or  $J(\cdot)$ .
- Generally, consists of **empirical risk term** and **regularization term**.

# Loss Function: Loss Landscape

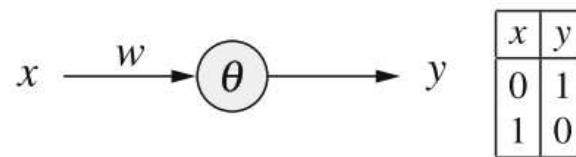


# Loss Function: Loss Landscape

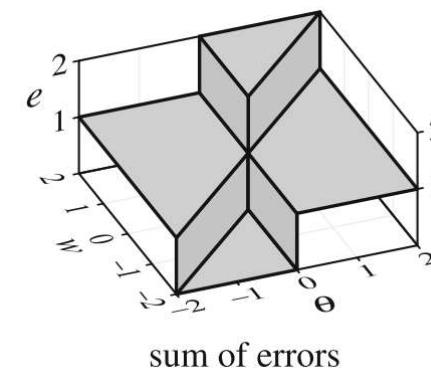
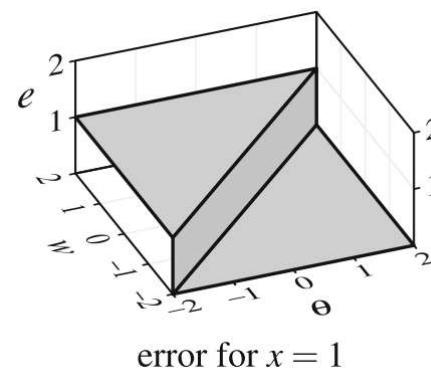
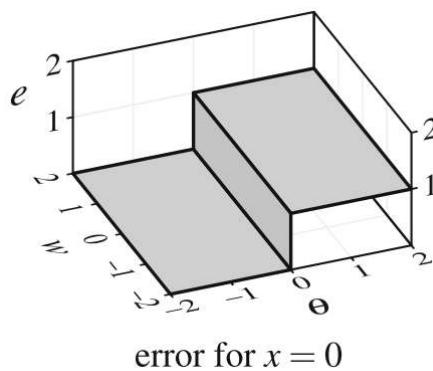


# Loss Functions: Negation Problem

- Consider a threshold logic unit with a single input and training examples for the negation:



- Error of computing the negation w.r.t. the  $\theta$  and  $w$ :



$$L = \sum_i L_i$$

# Loss Functions in Supervised Learning

- Regression Losses
  - Mean Square Error (L2 Loss)
- Classification Losses
  - Hinge Loss (Multi class SVM)
  - Cross Entropy Loss

# Mean Squared Error

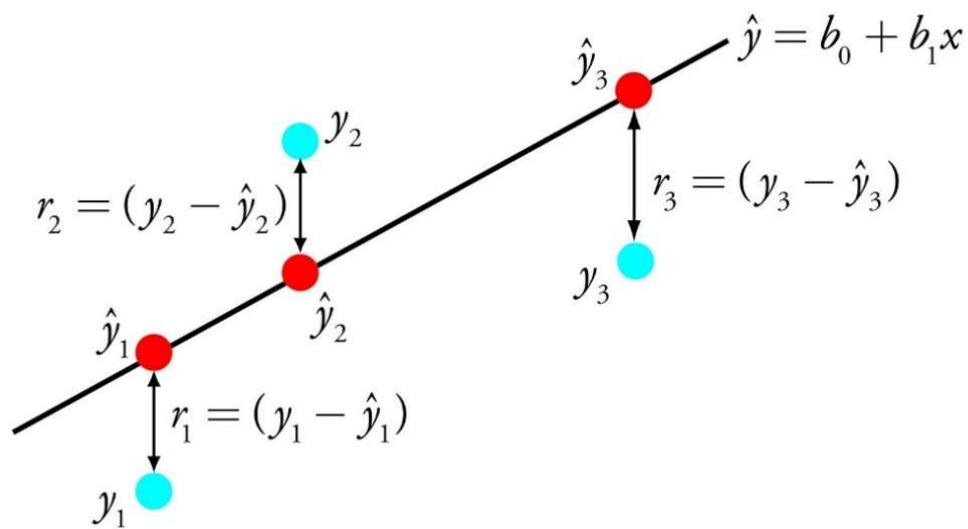
**Mean Squared Error (MSE)** loss function is widely used in **regression** problems.

$$J(w) = \sum_{i=1}^N (y^{(i)} - \underbrace{w^T x^{(i)}}_{\widehat{y}^{(i)}})^2$$

# Mean Squared Error

**Mean Squared Error (MSE) loss function is widely used in regression problems.**

$$J(w) = \sum_{i=1}^N (y^{(i)} - \underbrace{w^T x^{(i)}}_{\widehat{y}^{(i)}})^2$$



# Mean Squared Error

**Mean Squared Error (MSE)** loss function is widely used in **regression** problems.

$$J(w) = \sum_{i=1}^N (y^{(i)} - \underbrace{w^T x^{(i)}}_{\widehat{y}^{(i)}})^2$$

**Goal:** Find  $w^*$  which minimizes  $J(w)$ :

$$w^* = \operatorname{argmin}_w J(w)$$

# Mean Squared Error

**Mean Squared Error (MSE)** loss function is widely used in **regression** problems.

$$J(w) = \sum_{i=1}^N (y^{(i)} - \underbrace{w^T x^{(i)}}_{\widehat{y}^{(i)}})^2$$

Goal: Find  $w^*$  which minimizes  $J(w)$ :

$$w^* = \operatorname{argmin}_w J(w)$$

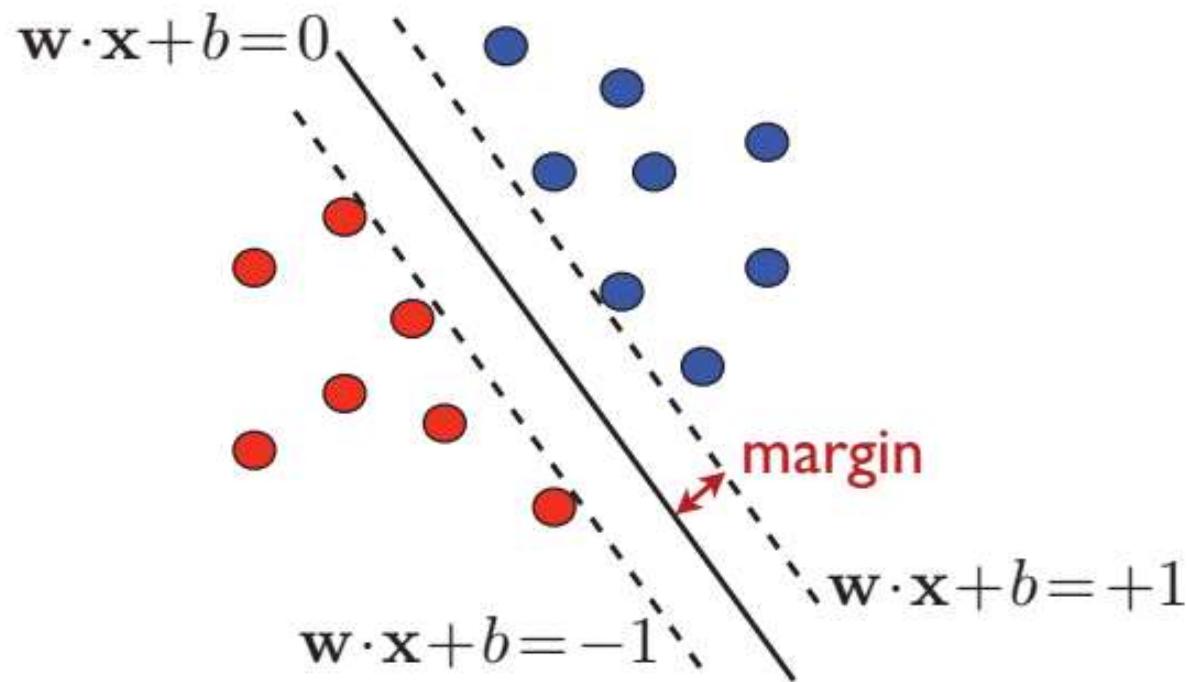
Optimization

# Loss Functions in Supervised Learning

- Regression Losses
  - Mean Square Error (L2 Loss)
- Classification Losses
  - Hinge Loss (Multi class SVM)
  - Cross Entropy Loss

# Hinge Loss: SVM Classifier

## Support Vector Machine (SVM)



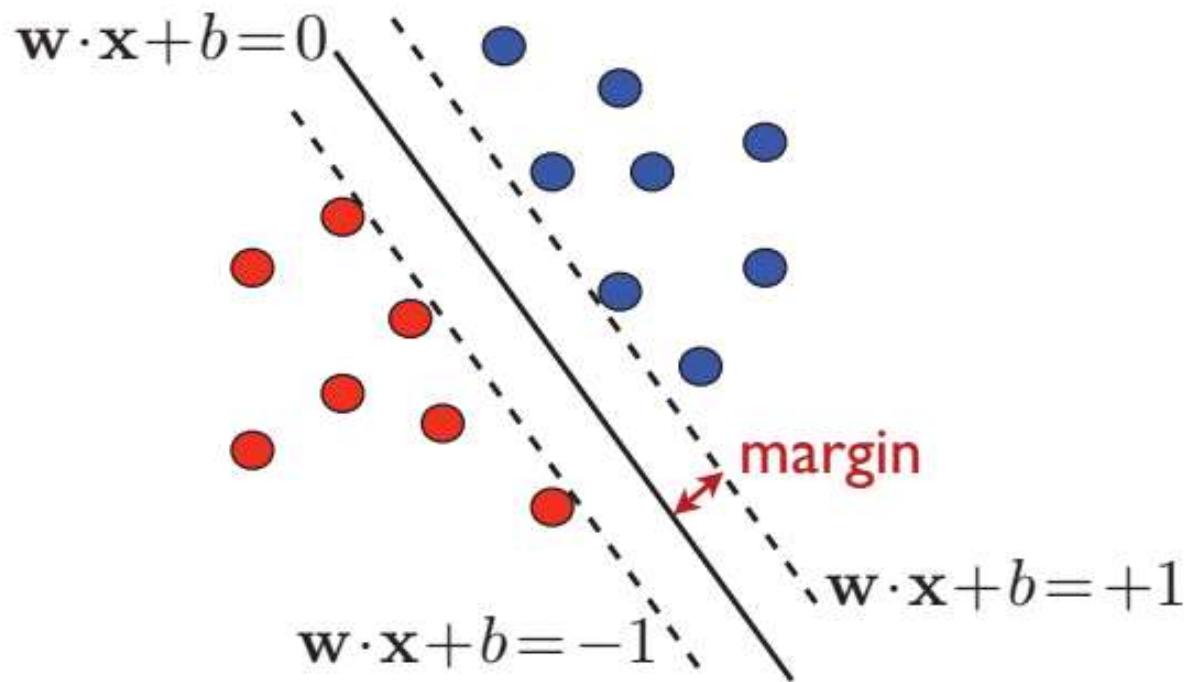
[Mohri]

WSS 2024



# Hinge Loss: SVM Classifier

## Support Vector Machine (SVM)



Misclassification Error:  $-sign(y^{(i)}[w \cdot x^{(i)} + b])$

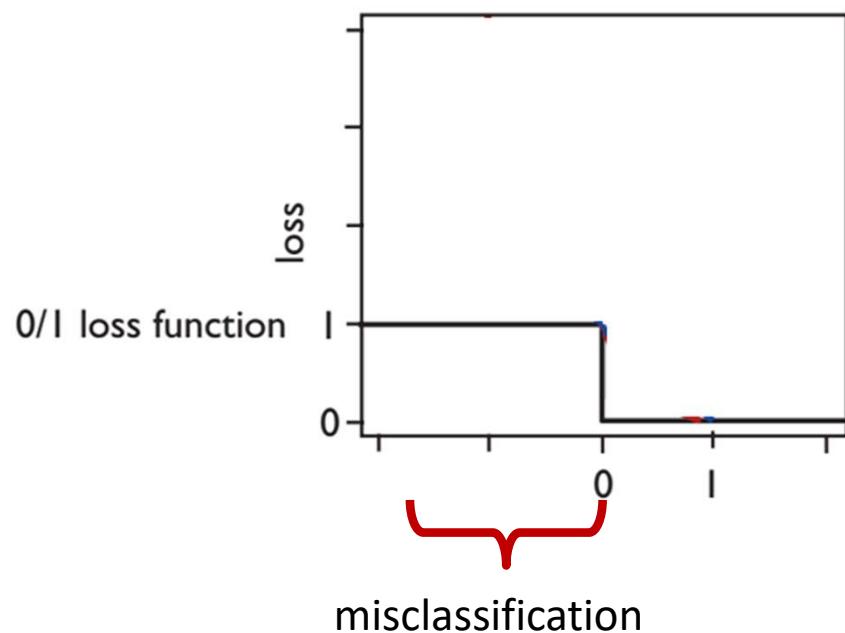
[Mohri]

WSS 2024



# Hinge Loss: SVM Classifier

## Support Vector Machine (SVM)



Misclassification Error:  $- \text{sign}(y^{(i)}[w \cdot x^{(i)} + b])$

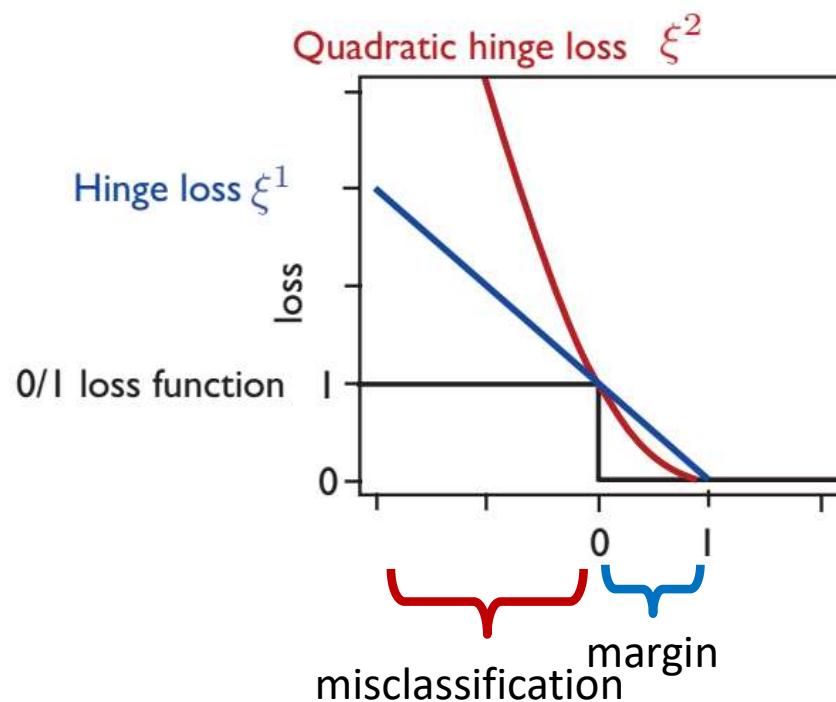
[Mohri]

WSS 2024



# Hinge Loss: SVM Classifier

## Support Vector Machine (SVM)



Misclassification Error:  $-sign(y^{(i)}[w \cdot x^{(i)} + b])$

[Mohri]

WSS 2024



# Cross Entropy Loss

- Cross-entropy:

$$H(q, p) = - \sum_x q(x) \log p(x)$$

# Cross Entropy Loss

- Cross-entropy:

$$H(q, p) = - \sum_x q(x) \log p(x)$$

- Multi-class cross-entropy Loss:

$$L(y, \hat{y}) = - \sum_j y_j \log \hat{y}_j$$

# Cross Entropy Loss

- Multi-class cross-entropy Loss:

$$L(y, \hat{y}) = - \sum_j y_j \log \hat{y}_j$$

**predicted probability of each class!**



# Cross Entropy Loss: Softmax Classifier

- Multi-class cross-entropy Loss:

$$L(y, \hat{y}) = - \sum_j y_j \log \hat{y}_j$$

**predicted probability of each class!**



- Softmax classifier:

$$s = f(x)$$

$$\hat{y}_j = \frac{e^{s_j}}{\sum_{k=1}^C e^{s_k}}$$

# Cross Entropy Loss: Softmax Classifier

- Multi-class cross-entropy Loss:

$$L(y, \hat{y}) = - \sum_j y_j \log \hat{y}_j$$

**predicted probability of each class!**

The true distribution (one-hot vector:  $q = [0,0,0, \dots, 1, \dots, 0]$ )

- Softmax classifier:

$$s = f(x)$$

$$\hat{y}_j = \frac{e^{s_j}}{\sum_{k=1}^C e^{s_k}}$$

# Multi-Class Classification: Case Study



**[32x32x3]**  
array of numbers 0...1

$$f(x, W) = Wx \quad \begin{matrix} 3072 \times 1 \\ 10 \times 3072 \end{matrix}$$

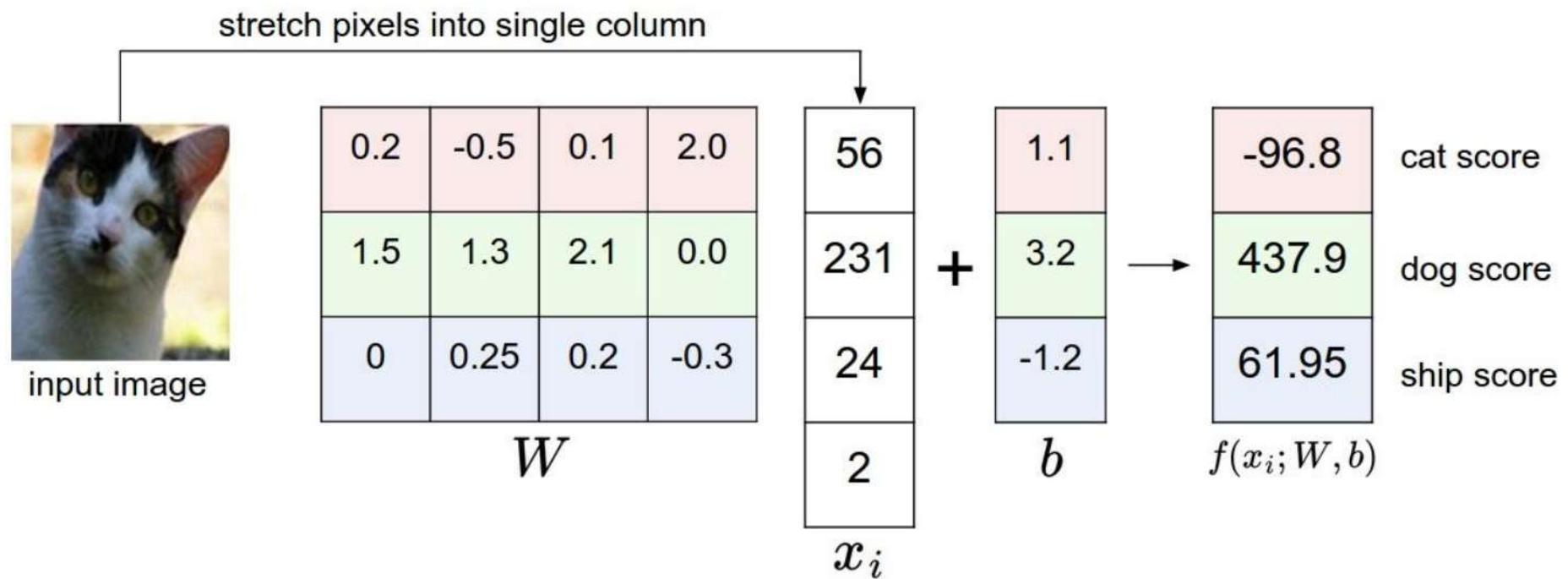
$$(+b) \quad \begin{matrix} 10 \times 1 \end{matrix}$$

10 numbers,  
indicating class  
scores

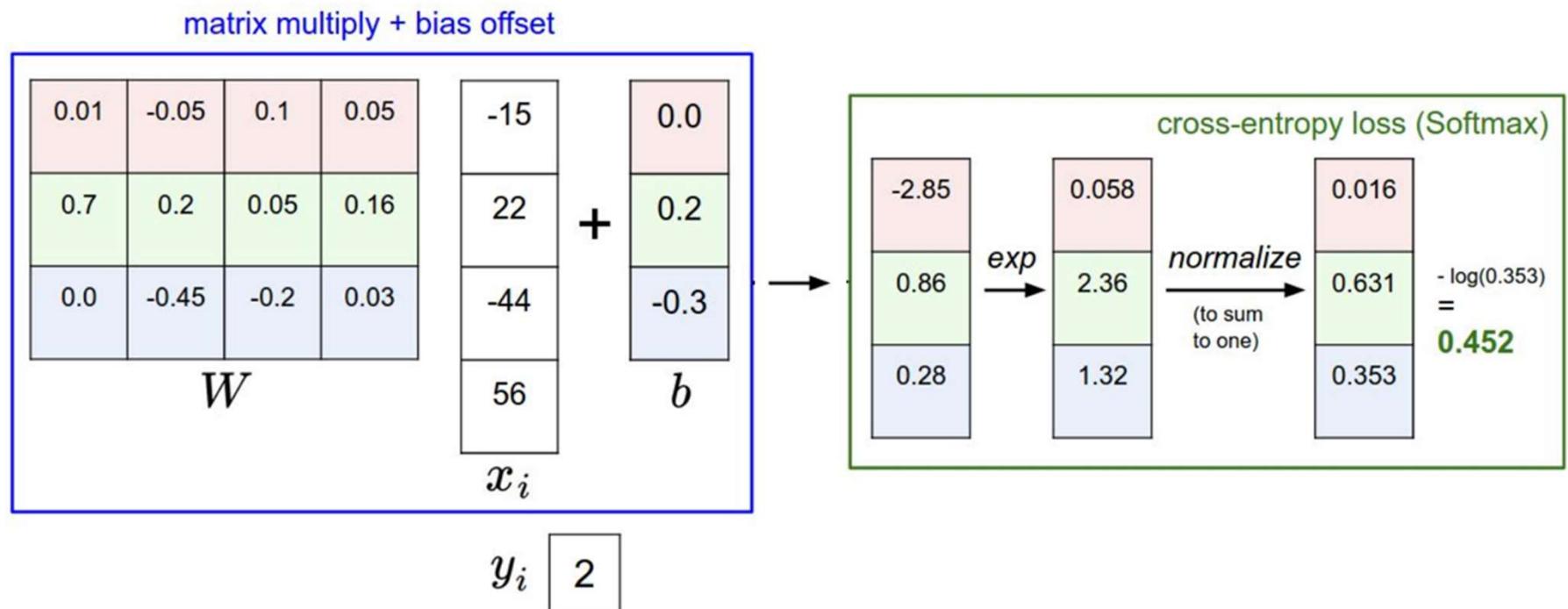
parameters, or “weights”

# Multi-Class Classification: Case Study

Example with an image with 4 pixels, and 3 classes (**cat/dog/ship**)



# Multi-Class Classification: Case Study



# Cross Entropy Loss

- Binary classification:

$$J = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$



# Cross Entropy Loss

- Binary classification:

$$J = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$



Scalar value!  
probability of being 1

# Cross Entropy Loss

- Binary classification:

$$J = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$



- Sigmoid function:

$$s = f(x)$$

$$\hat{y} = \frac{1}{1 + e^{-s}}$$

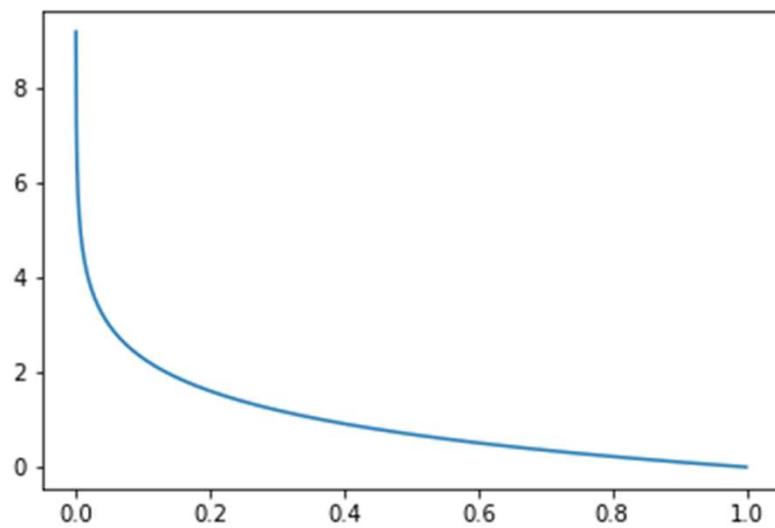
Scalar value!  
probability of being 1

# Cross Entropy Loss

- Binary classification:

$$J = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

(if  $y = 1$ ):  
 $J = -\log \hat{y}$

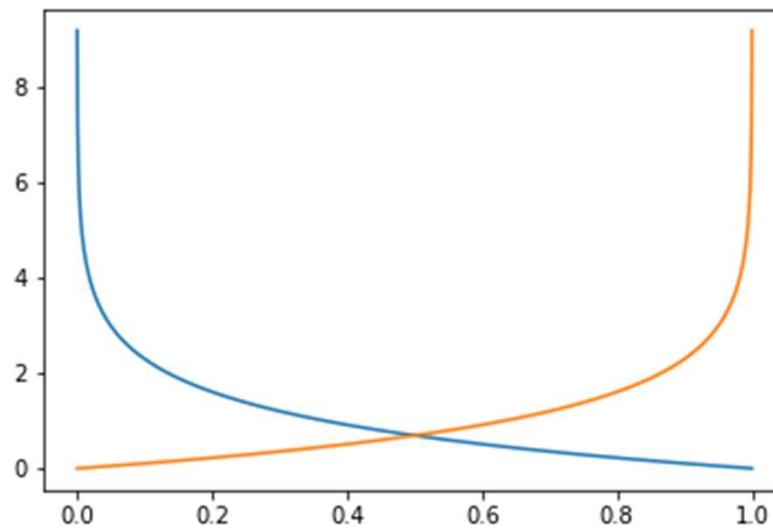


# Cross Entropy Loss

- Binary classification:

$$J = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

(if  $y = 1$ ):  
 $J = -\log \hat{y}$



(if  $y = 0$ )  
 $J = -\log(1 - \hat{y})$

# Loss Functions in Supervised Learning

- Regression Losses
  - Mean Square Error (L2 Loss)
- Classification Losses
  - Hinge Loss (Multi class SVM)
  - Cross Entropy Loss

# Learning and Optimization (Agenda)

- Loss Function
- Generalization and Overfitting
  - Regularization
  - Model Selection and Validation
- Optimization

# Generalization

- We don't intend to **memorize** data but need to figure out the **pattern**.
- A core objective of learning is to **generalize** from experiences.



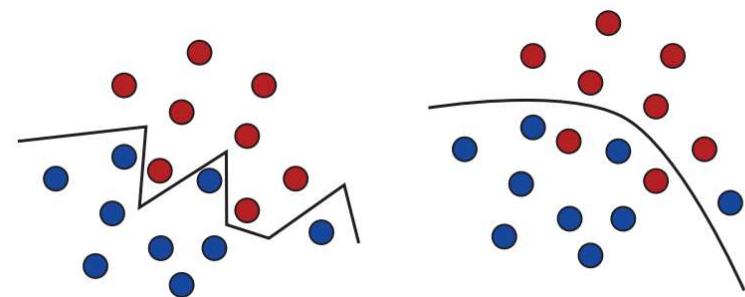
# Generalization

- We don't intend to **memorize** data but need to figure out the **pattern**.
- A core objective of learning is to **generalize** from experiences.
- **Generalization**: ability of a learning algorithm to perform accurately on new unseen examples after having trained on training data.



# Overfitting

- **Overfitting** happens when a model learns the detail and noise in the training data but it loses the performance on new (test) data.
- **Bad Generalization:** Model fails to generalize unseen samples.

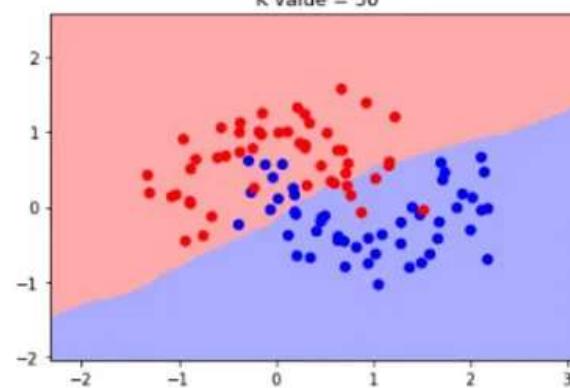
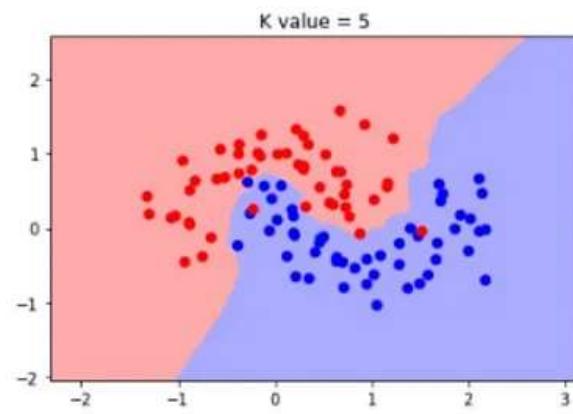
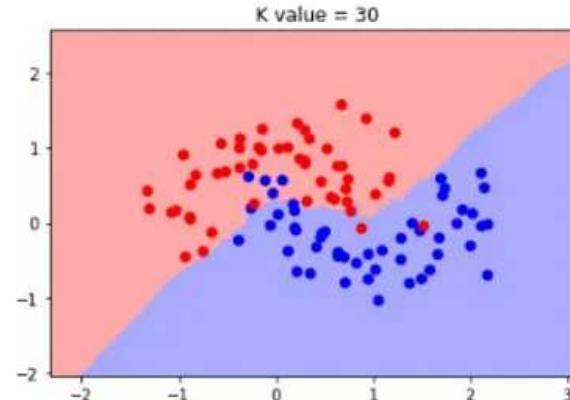
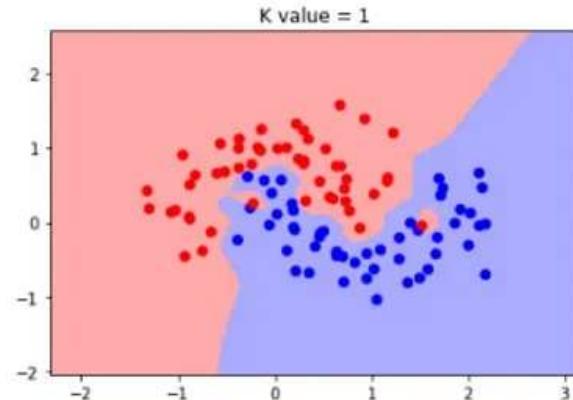


[Mohri]

# Overfitting: KNN

- Effect of  $K$  on decision boundaries

Overfitting!



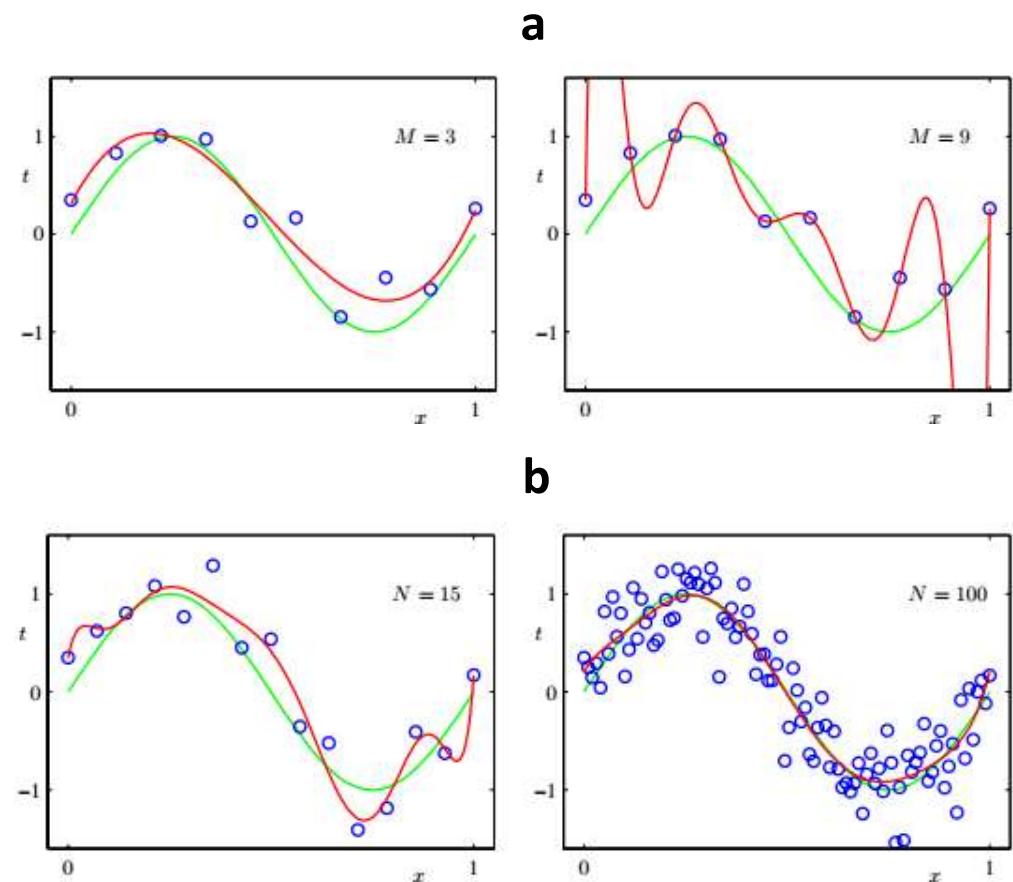
Underfitting!

# Overfitting

- **Overfitting** happens when a model learns the detail and noise in the training data but it loses the performance on new (test) data.
- **Bad Generalization:** Model fails to generalize unseen samples.
- Main reasons:
  - a) **Model complexity**
  - b) **Small data size**

# Overfitting

- **Bad Generalization:** Model fails to generalize unseen samples.
- Main reasons:
  - Model complexity**
  - Small data size**



[Bishop]

# Overcome Overfitting

Regularization Terms:

Prevent weight growing (weight decay)



# Overcome Overfitting: Regularization Terms

Prevent weight growing (weight decay)

L2 (ridge) regularization:  $R(w) = \sum_{k=1}^K \sum_{l=1}^d w_{lk}^2$

L1 (lasso) regularization:  $R(w) = \sum_{k=1}^K \sum_{l=1}^d |w_{lk}|$

Full loss:

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R(w)$$

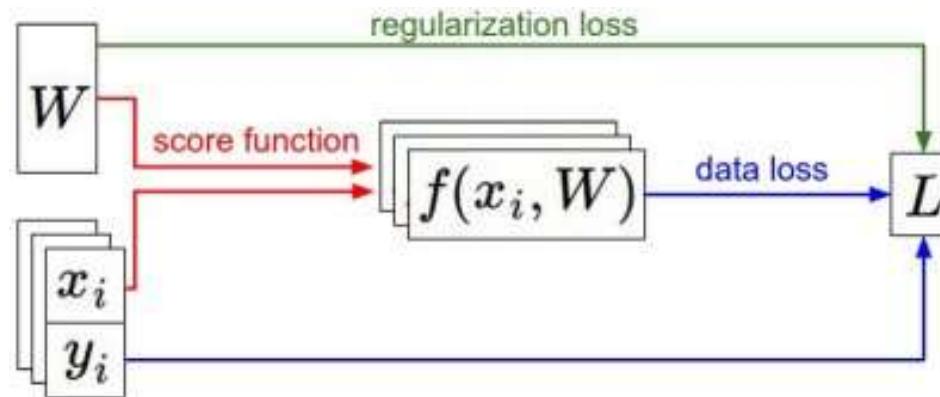


# Overcome Overfitting: Regularization Terms

Prevent weight growing (weight decay)

Full loss:

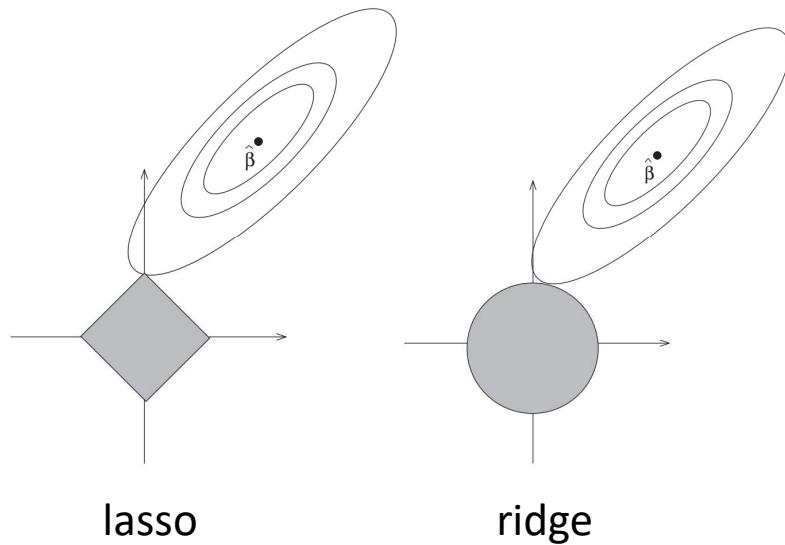
$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R(w)$$



# Regularization Terms: Comparison

L2 (ridge) regularization:  $R(w) = \sum_{k=1}^K \sum_{l=1}^d w_{lk}^2$

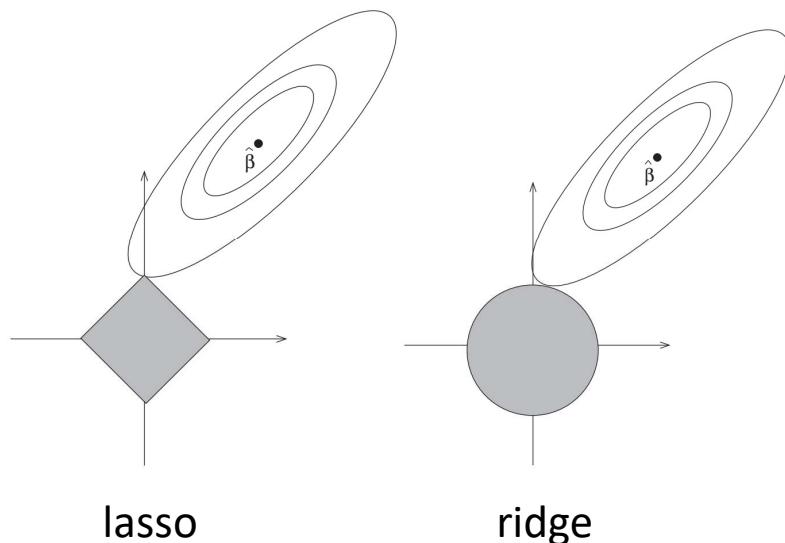
L1 (lasso) regularization:  $R(w) = \sum_{k=1}^K \sum_{l=1}^d |w_{lk}|$



# Regularization Terms: Comparison

L2 (ridge) regularization:  $R(w) = \sum_{k=1}^K \sum_{l=1}^d w_{lk}^2$

L1 (lasso) regularization:  $R(w) = \sum_{k=1}^K \sum_{l=1}^d |w_{lk}|$



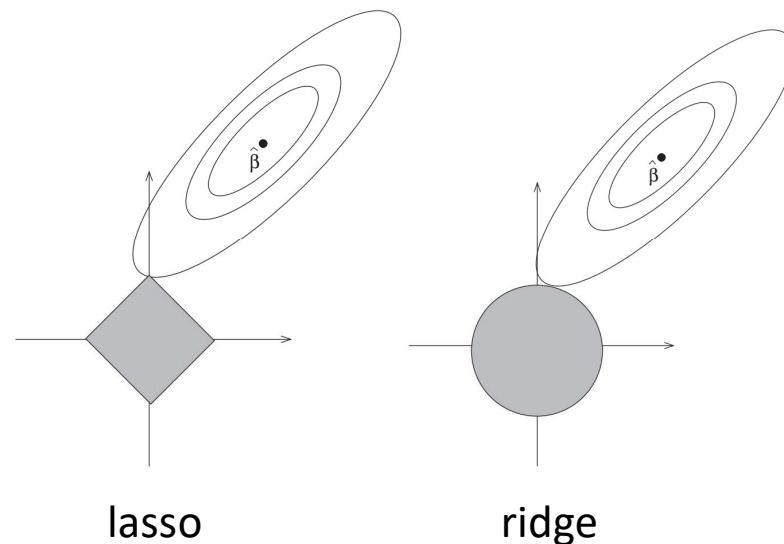
lasso

ridge



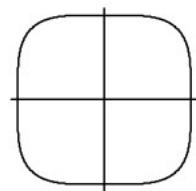
regularization + feature selection

# Regularization Terms: Comparison

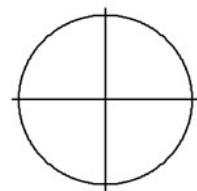


Other norms:

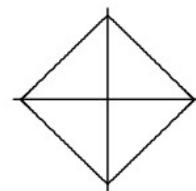
$q=4$



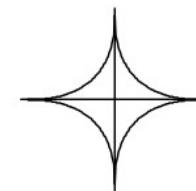
$q=2$



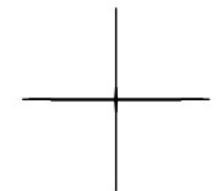
$q=1$



$q=0.5$



$q=0.1$



$$\sum_j |\beta_j|^q$$

[Tibshirani]

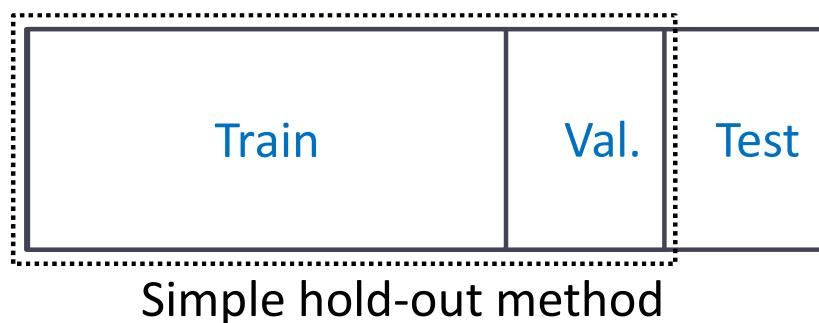
# Overcome Overfitting

- Model Selection and Validation:  
Determine model complexity based on  
**validation error.**



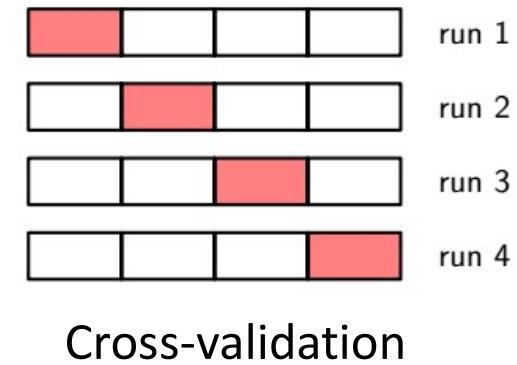
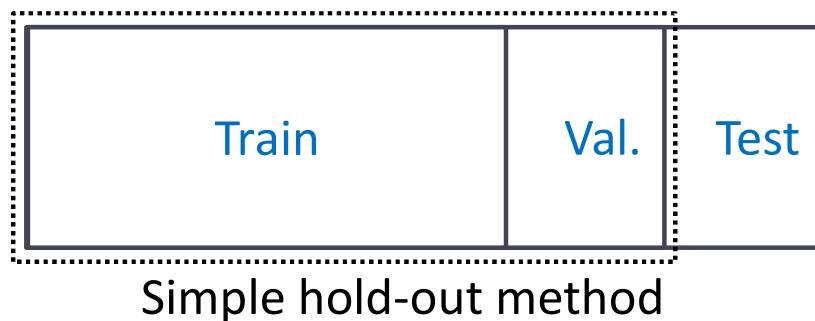
# Overcome Overfitting

- Model Selection and Validation:  
Determine model complexity based on  
**validation error.**



# Overcome Overfitting

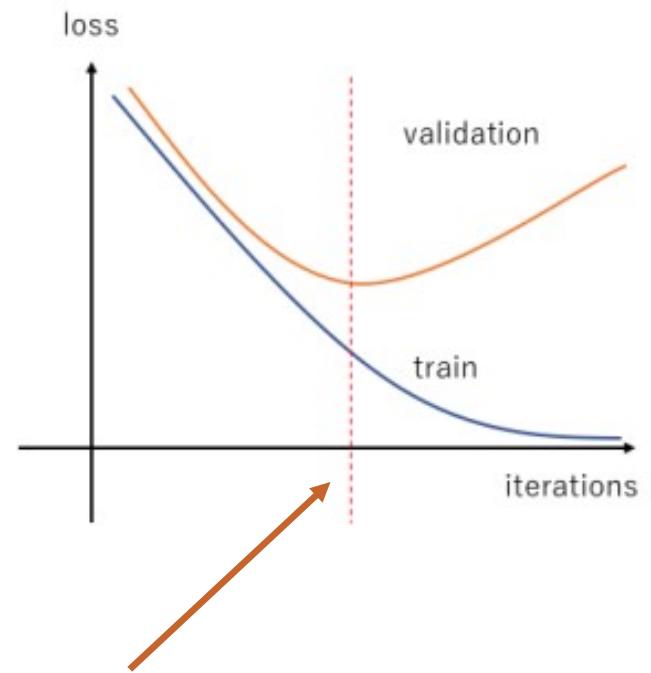
- Model Selection and Validation:  
Determine model complexity based on  
**validation error.**



# Overcome Overfitting

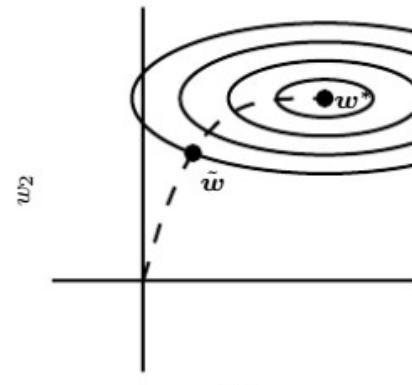
- **Early Stopping:**

Stop training procedure when validation loss begins to increase.

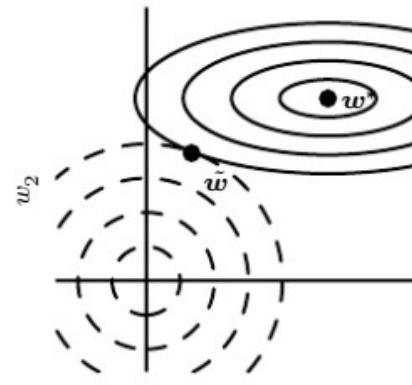


# Overcome Overfitting: Early Stopping

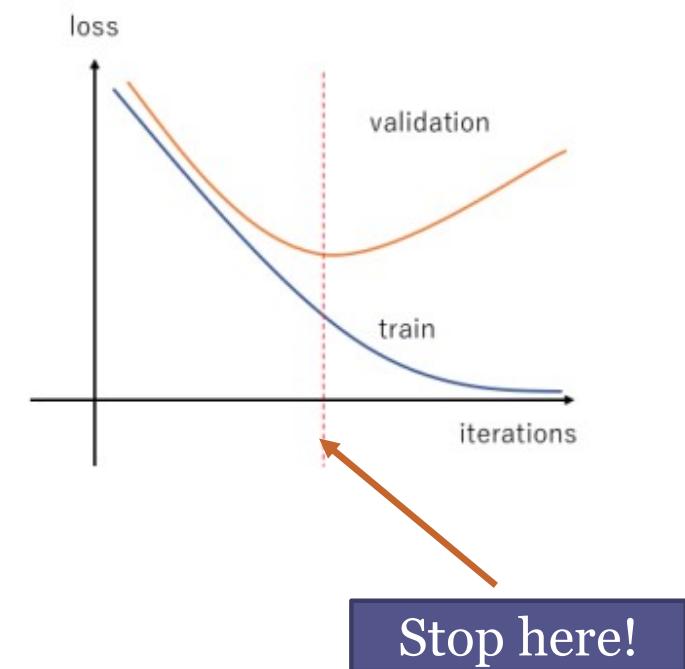
- Stop training procedure when validation loss begins to increase.
- Similar to L2 regularization



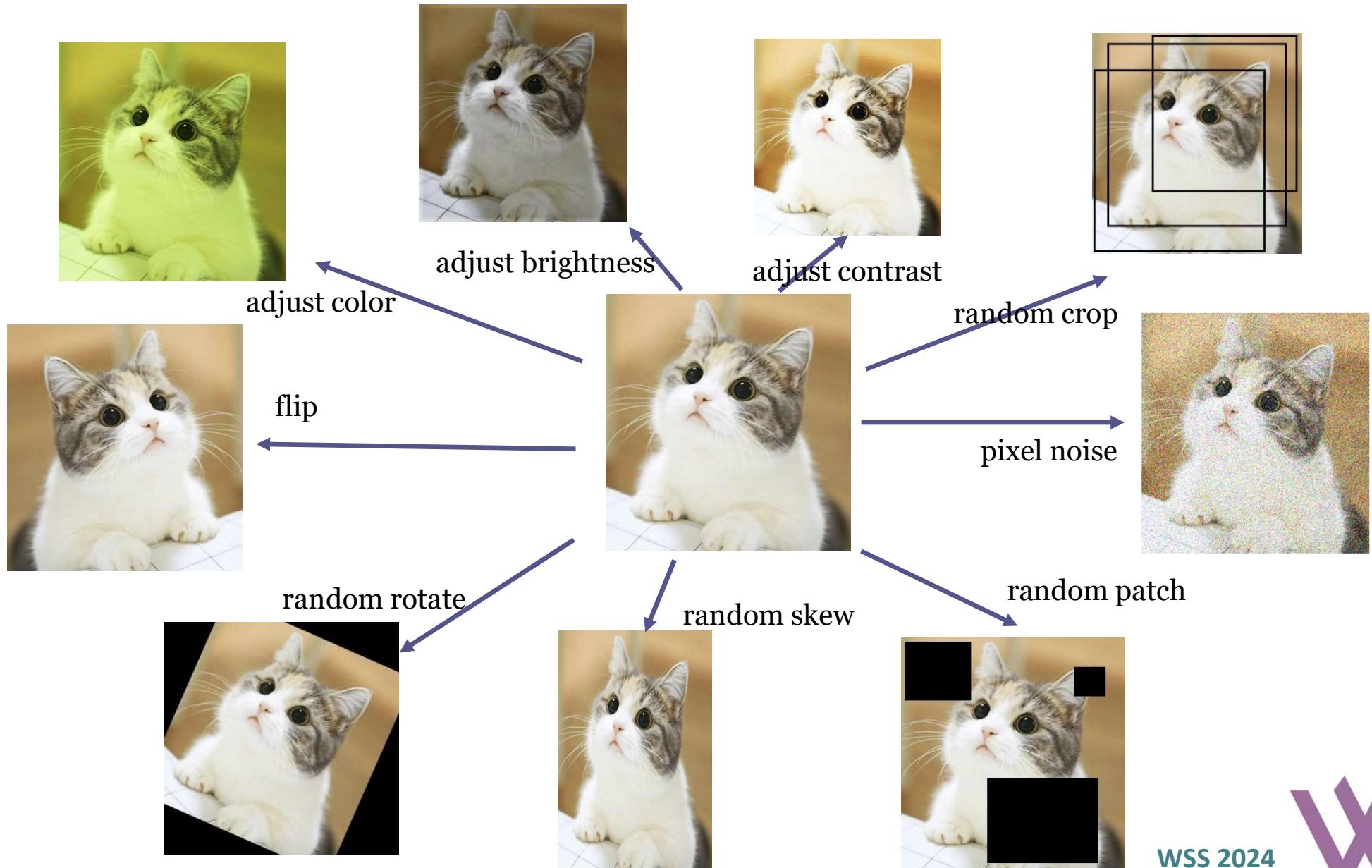
Early stopping



L2 regularization



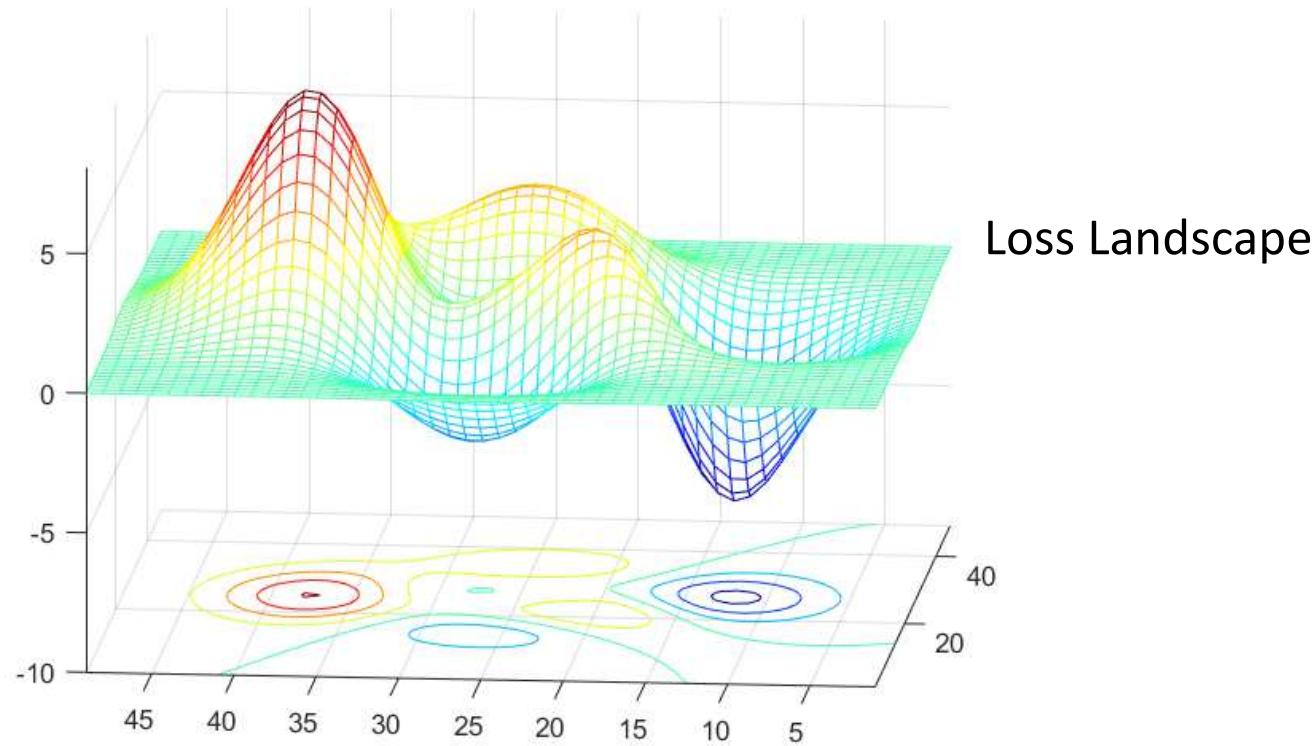
# Overcome Overfitting: Data Augmentation



# Overcome Overfitting (Summary)

- Add regularization terms to loss functions
- Early Stopping
- Dataset Augmentation
- Feature drop-out

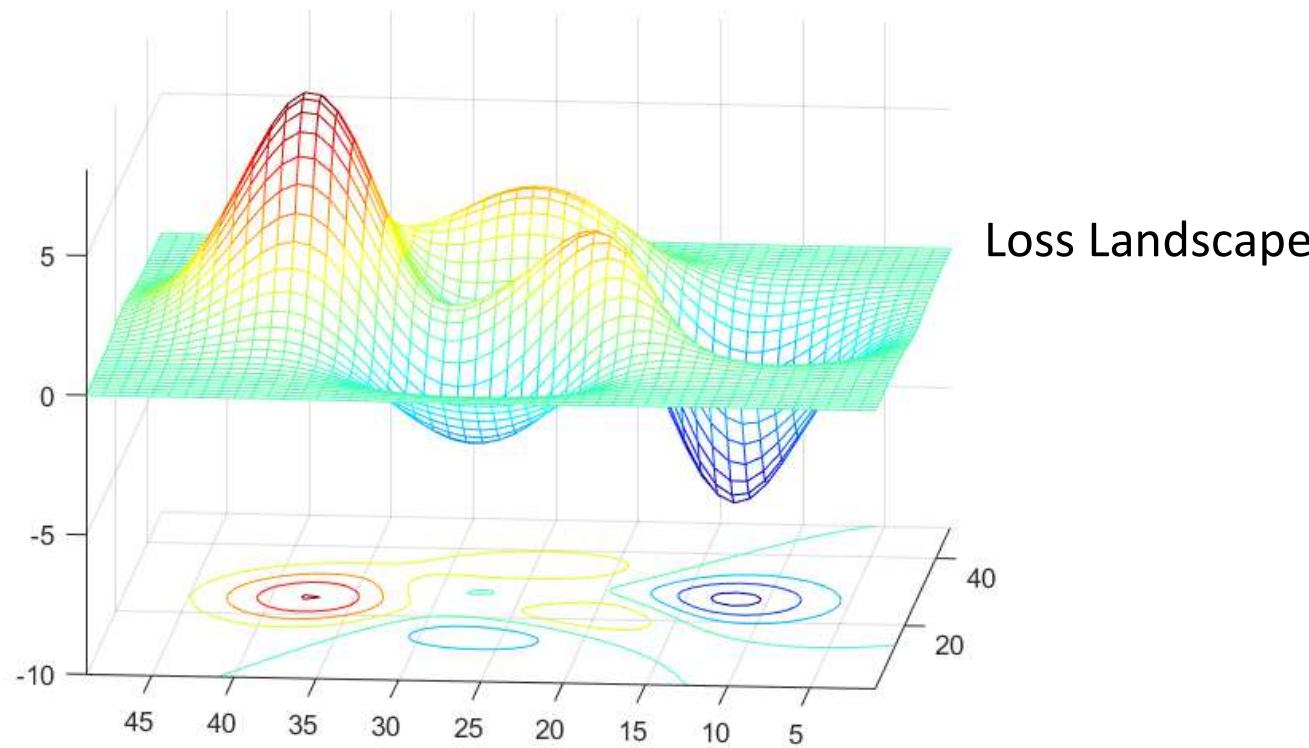
# Loss Function (Recap)



**Goal:**

Finding Local Minima!

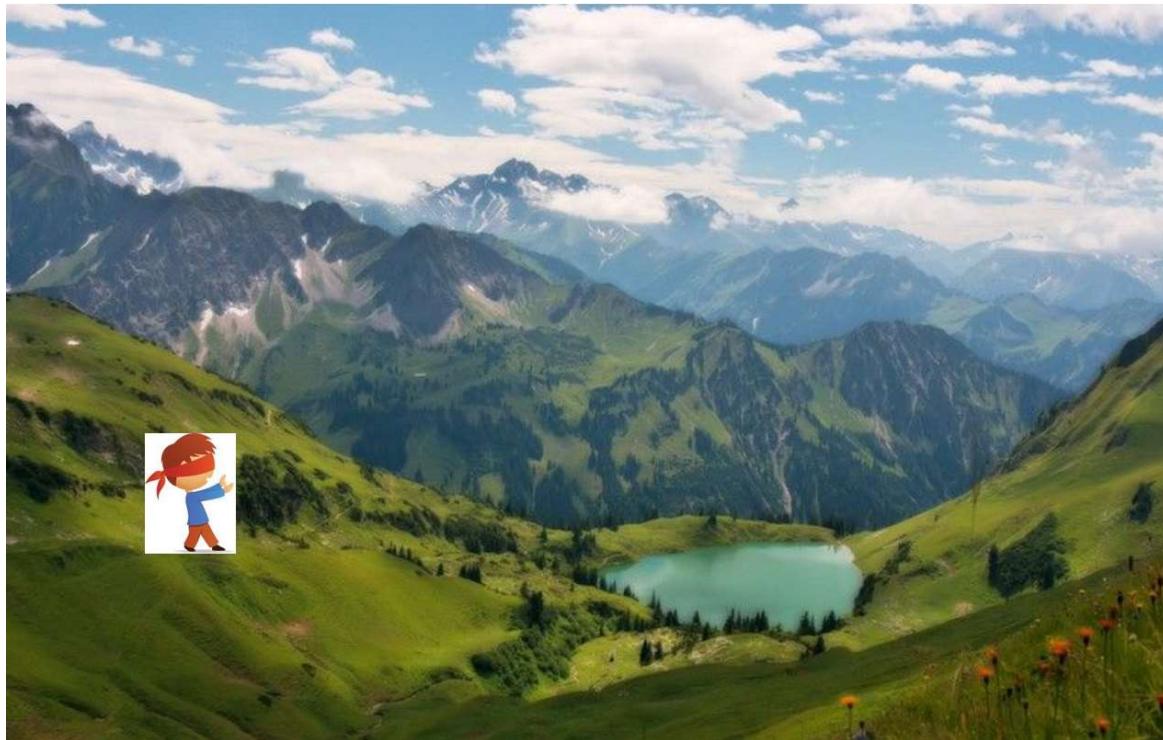
# Loss Surface



## Problem!

We can just evaluate loss value for a fixed set of parameters!

# Loss Surface

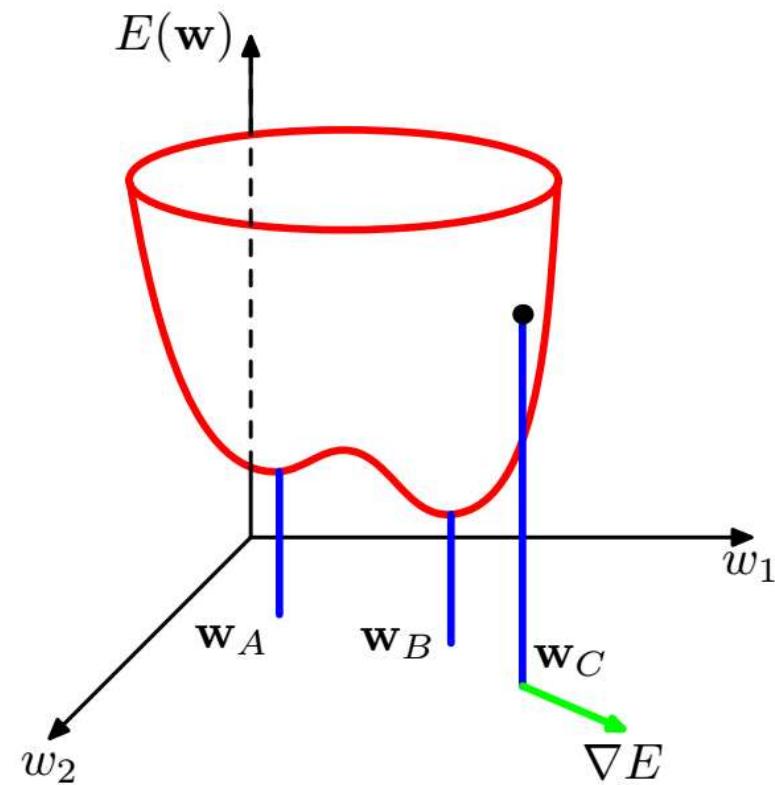


## Problem!

We can just evaluate loss value for a fixed set of parameters!

# Optimization

## Gradient Based Optimizations



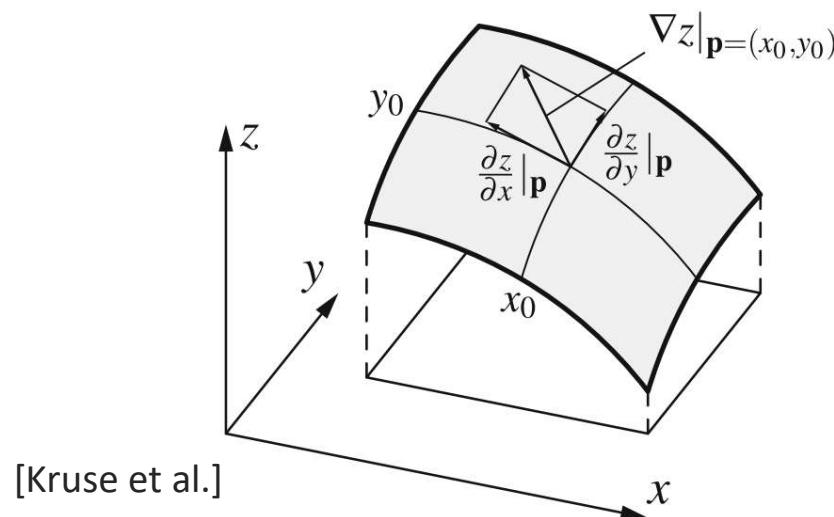
[Bishop]

# Optimization: Gradient Descend

Minimize  $J(w)$ :

$$w^{t+1} = w^t - \eta \nabla_w J(w^t)$$

$$\nabla_w J(w) = \left[ \frac{\partial J(w)}{\partial w_1}, \frac{\partial J(w)}{\partial w_2}, \dots, \frac{\partial J(w)}{\partial w_d} \right]$$



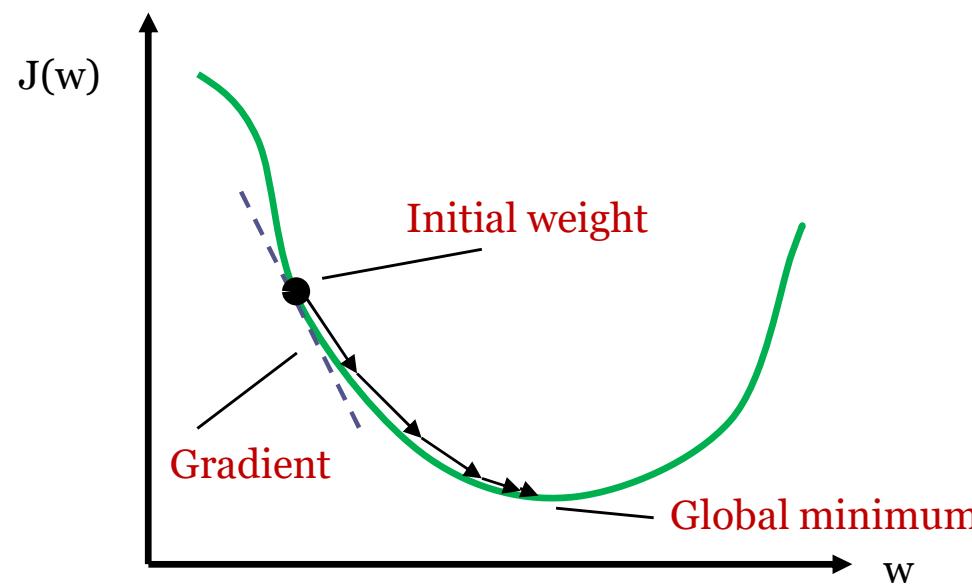
# Optimization: Gradient Descend

Minimize  $J(w)$ :

learning rate

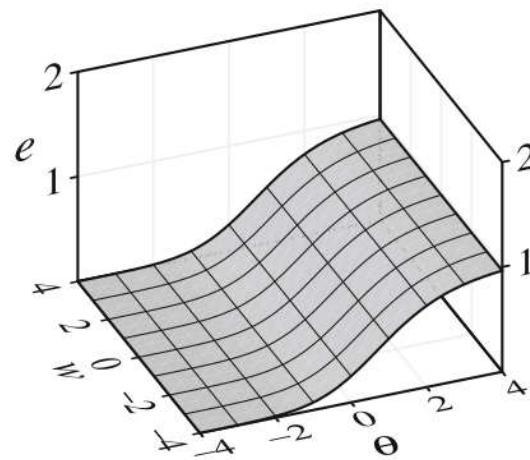
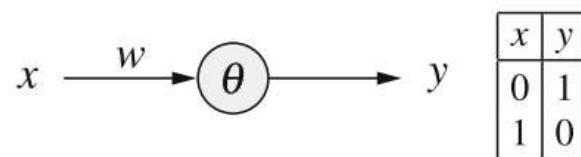
$$w^{t+1} = w^t - \eta \nabla_w J(w^t)$$

$$\nabla_w J(w) = \left[ \frac{\partial J(w)}{\partial w_1}, \frac{\partial J(w)}{\partial w_2}, \dots, \frac{\partial J(w)}{\partial w_d} \right]$$

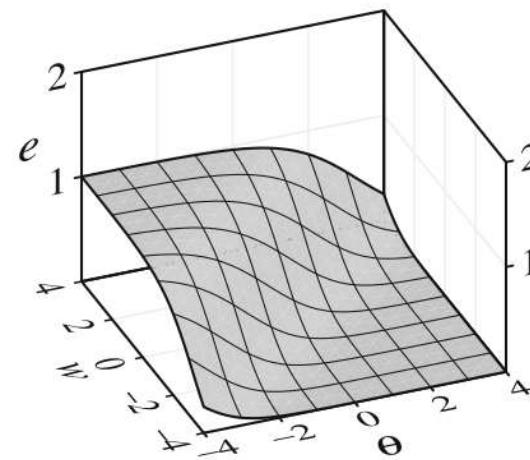


# Gradient Descend: Case Study

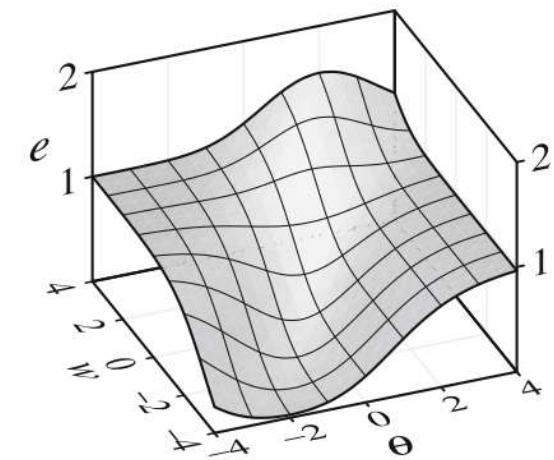
## Negation Problem



error for  $x = 0$



error for  $x = 1$

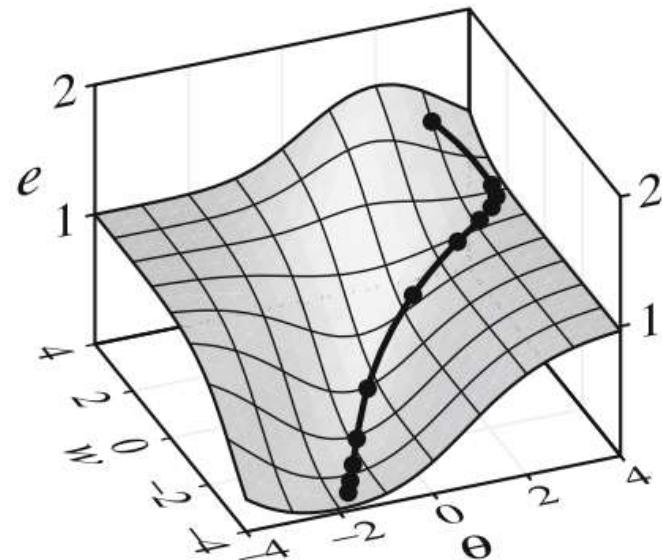


sum of errors

[Kruse et al.]

# Gradient Descend: Case Study

## Negation Problem



Training processes with initial values  $\theta = 3$ ,  $w = \frac{7}{2}$  and learning rate 1

[Kruse et al.]

WSS 2024



# Stochastic Gradient Descend

- Batch techniques process the entire training set in one go.
  - computationally costly for large data sets.

$$J(w) = \sum_{i=1}^N J^{(i)}(w)$$

- SGD: Update after presentation of a **mini-batch** of data:

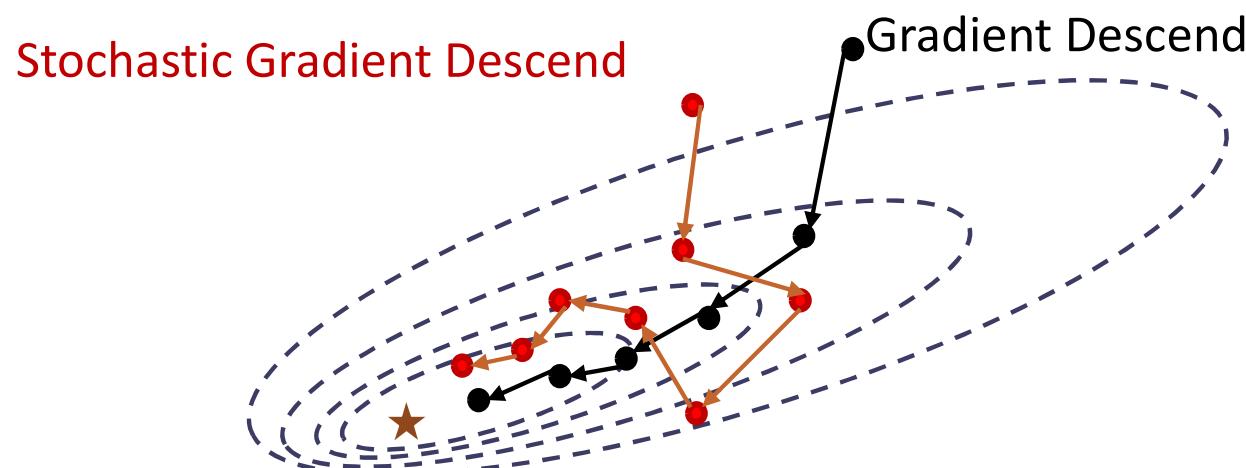
$$w^{t+1} = w^t - \eta \sum_{j \in S} \nabla_w J^{(j)}(w)$$



# Stochastic Gradient Descend

- SGD: Update after presentation of a **mini-batch** of data:

$$w^{t+1} = w^t - \eta \sum_{j \in S} \nabla_w J^{(j)}(w)$$



# Learning and Optimization (Summary)

- Loss Function
- Generalization and Overfitting
  - Regularization
  - Model Selection and Validation
- Optimization

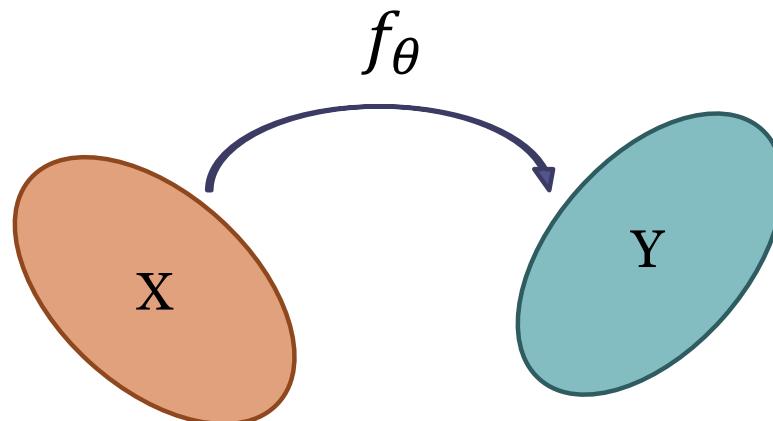
# Supervised Learning (Recap)

- Supervised learning

Given: Training set

Labeled set of N input-output pairs  $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$

**Goal: Learning a mapping from  $x$  to  $y$**



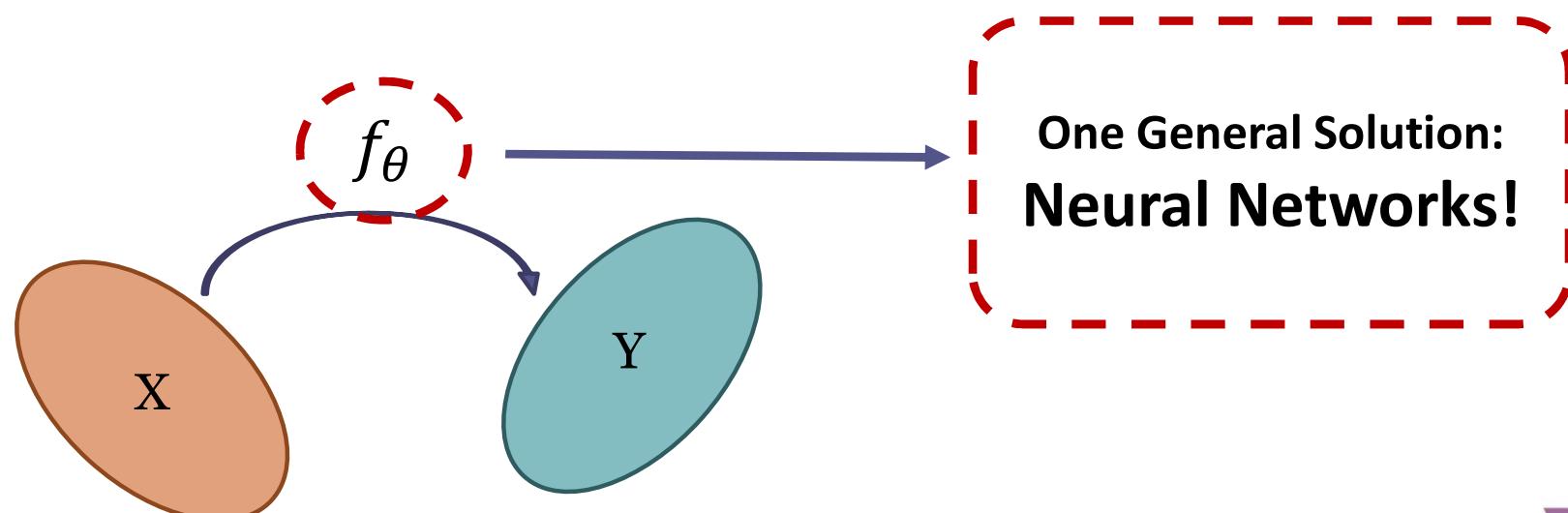
# Supervised Learning (Recap)

- Supervised learning

Given: Training set

Labeled set of N input-output pairs  $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$

**Goal: Learning a mapping from  $x$  to  $y$**

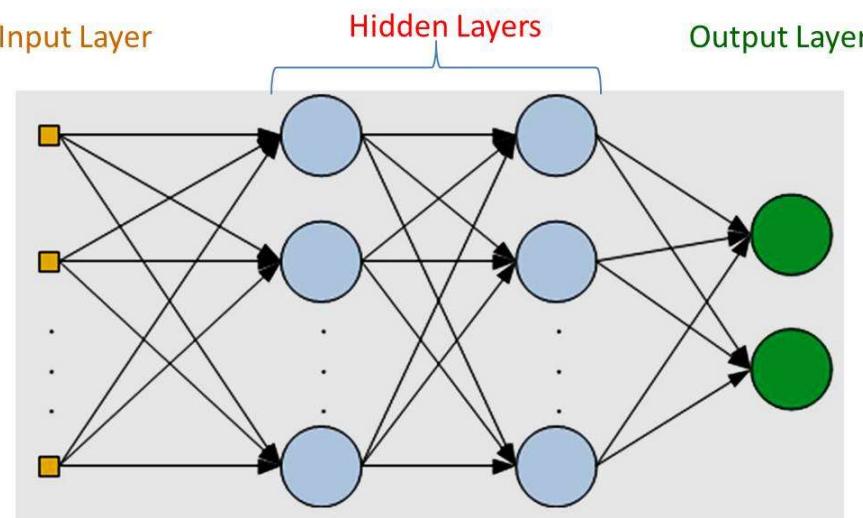


# Neural Networks

- Multilayer Perceptrons
- Convolutional Neural Networks
- Recurrent Neural Networks
- Transformers

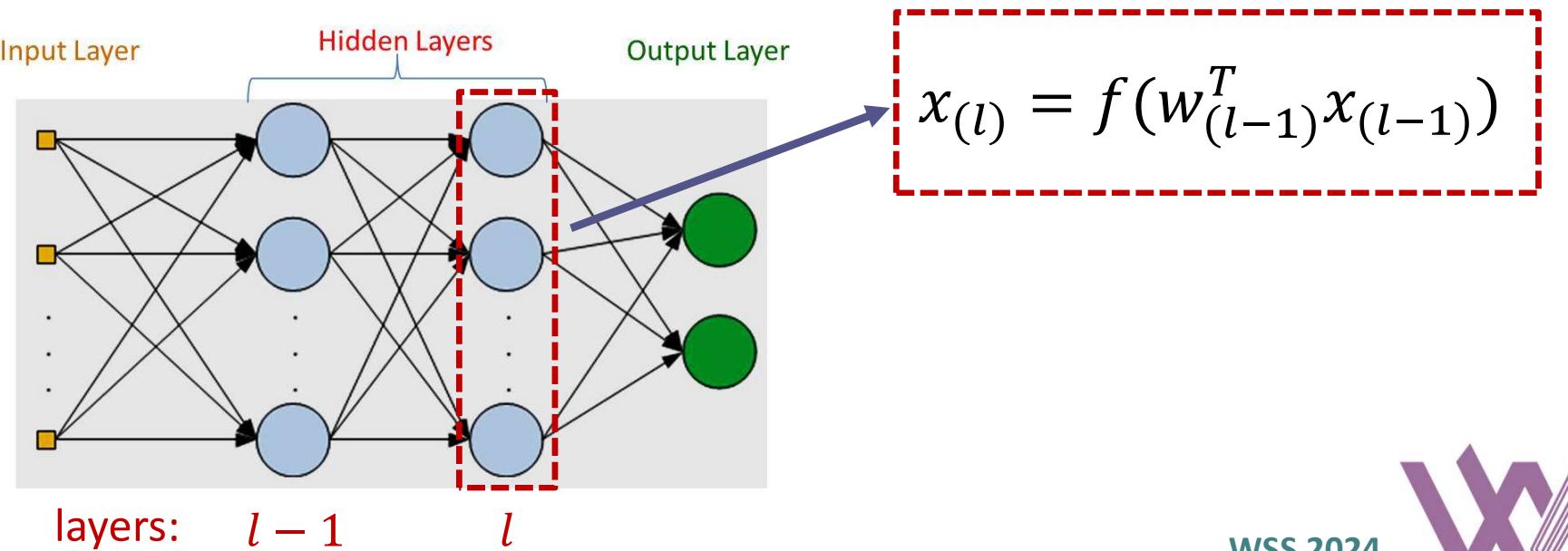
# Multilayer Perceptrons (MLP)

- An MLP consists of **multiple layers** of computation nodes.
- Each layer is **fully connected** to the next one.
- Modeling complex non-linear functions



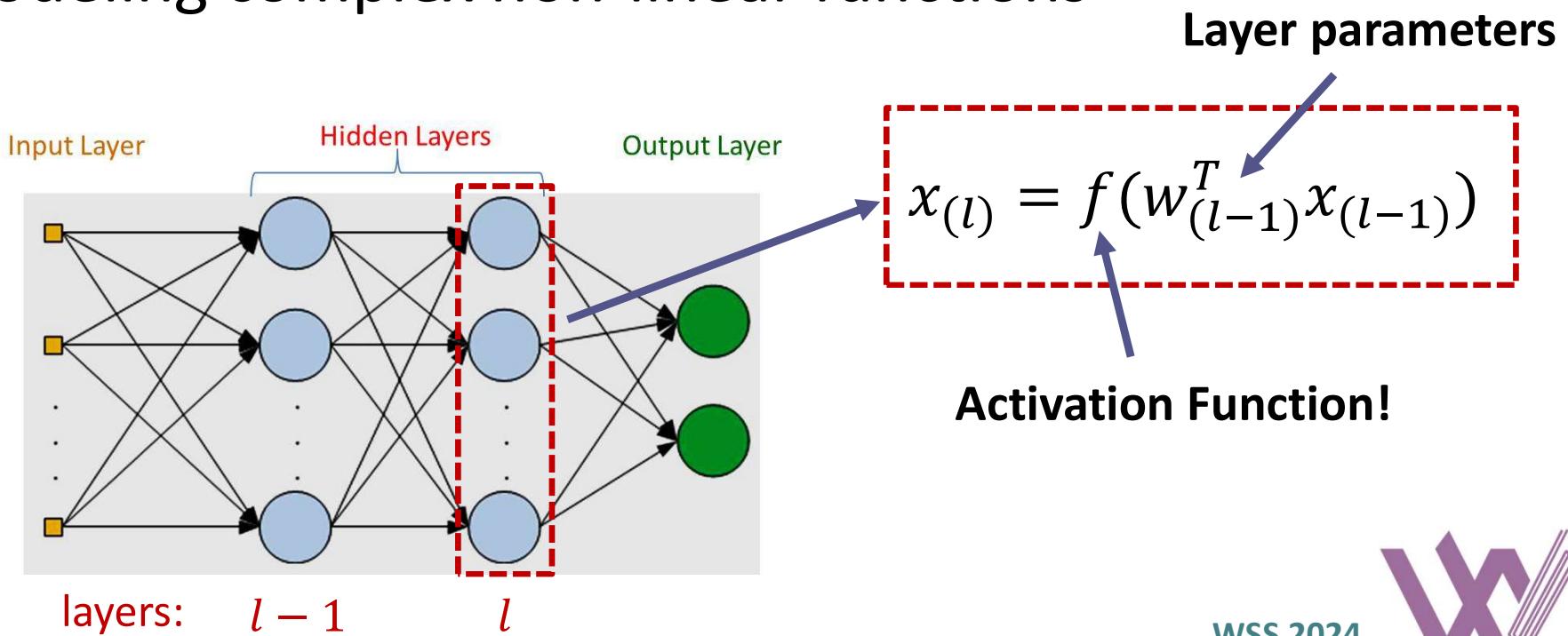
# Multilayer Perceptrons (MLP)

- An MLP consists of **multiple layers** of computation nodes.
- Each layer is **fully connected** to the next one.
- Modeling complex non-linear functions



# Multilayer Perceptrons (MLP)

- An MLP consists of **multiple layers** of computation nodes.
- Each layer is **fully connected** to the next one.
- Modeling complex non-linear functions



# Activation Function

- Networks without hidden layers are very limited in the input-output mappings they can learn to model
- We need multiple layers of adaptive, non-linear hidden units
- What if we use linear activation functions?

# Activation Function

Name	Plot	Equation	Derivative (with respect to $x$ )	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}^{[1]}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$
Softsign <sup>[7][8]</sup>		$f(x) = \frac{x}{1 +  x }$	$f'(x) = \frac{1}{(1 +  x )^2}$	$(-1, 1)$
Inverse square root unit (ISRU) <sup>[9]</sup>		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1 + \alpha x^2}}\right)^3$	$\left(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}}\right)$
Rectified linear unit (ReLU) <sup>[10]</sup>		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$

# Convolutional Neural Networks

Problem with MLP:

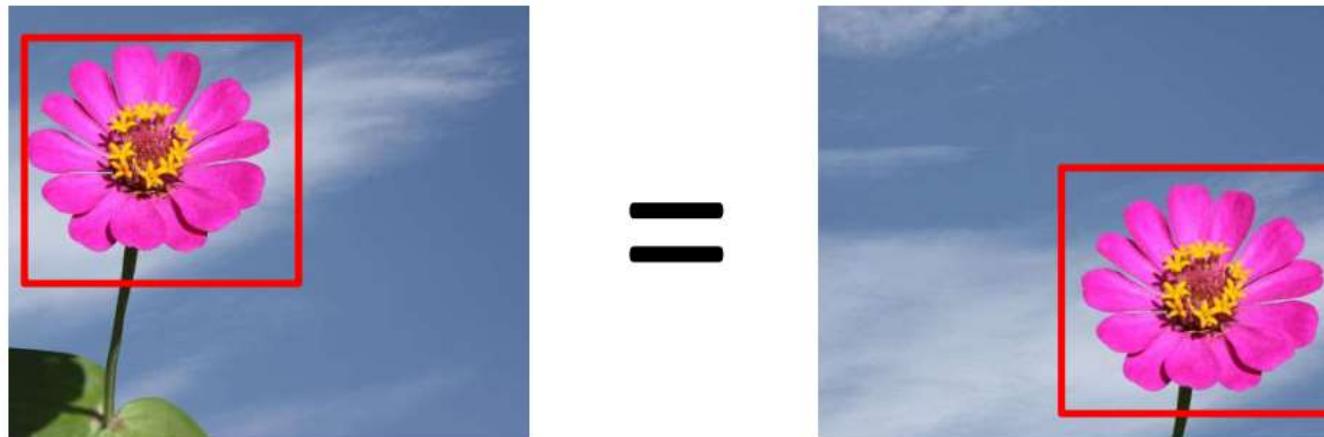


=



# Convolutional Neural Networks

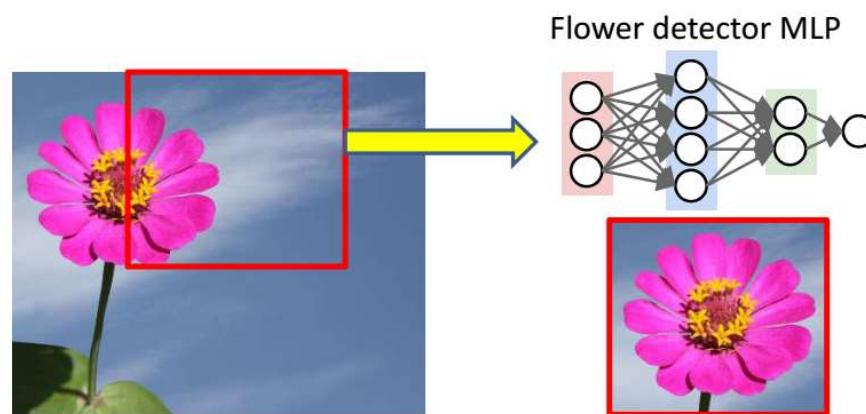
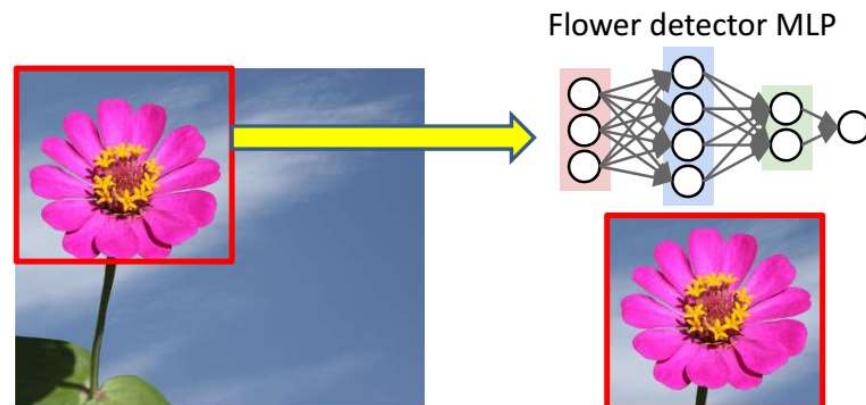
Problem with MLP:



- Need a simple network that will fire regardless of the precise location of the target object
- The need for **shift invariance**

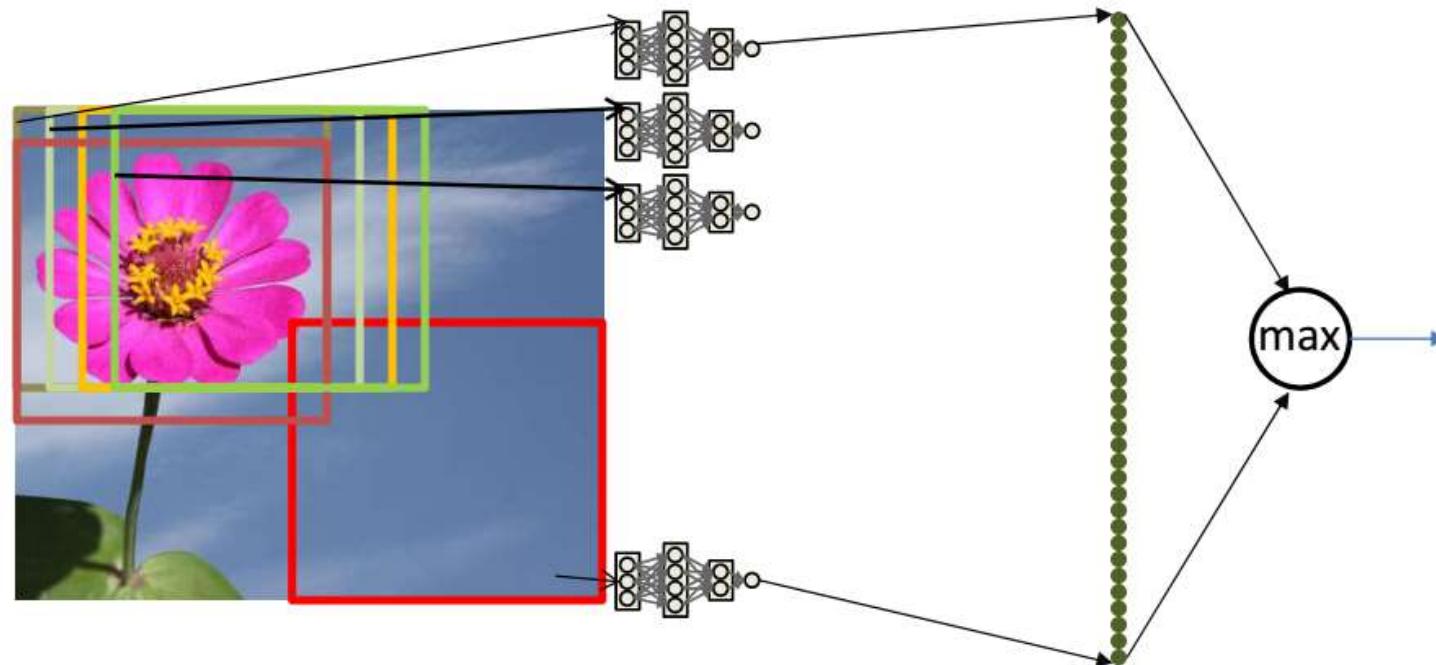
# Convolutional Neural Networks

Solution: Scan for desired pattern



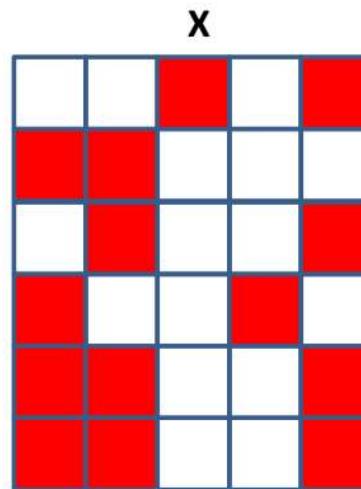
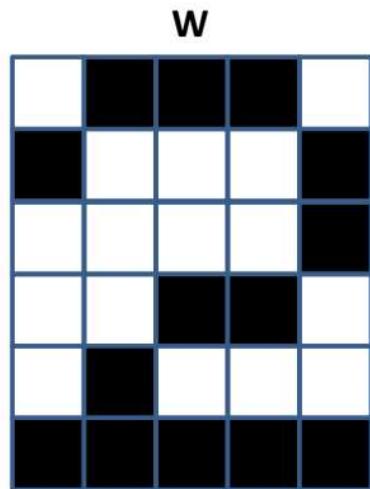
# Convolutional Neural Networks

Solution: Scan for desired pattern

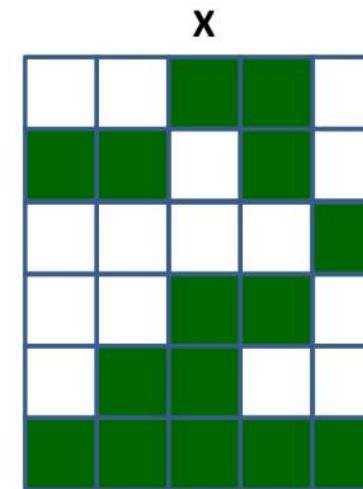


# Convolution Kernels

- Filter (kernel) as a “template”



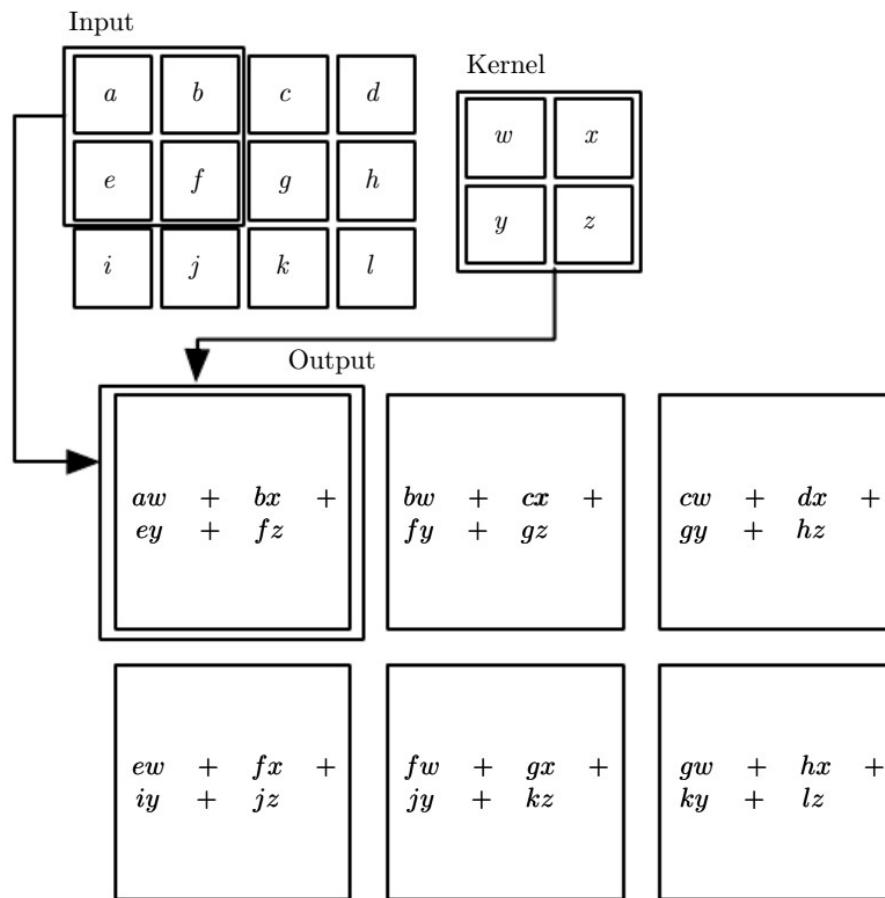
Correlation = 0.57



Correlation = 0.82

- The green pattern looks more like the weights pattern (black) than the red pattern
- The green pattern is more *correlated* with the weights

# Convolution Kernels



[Goodfellow et al]

# Convolutional Neural Network

- Three main types of layers
  - **Convolutional Layer**
    - Extract useful features
    - Output of neurons are connected to local regions in the input
    - Applying the same filter on the whole image (weight sharing)

# Convolutional Neural Network

- Three main types of layers
  - **Convolutional Layer**
    - Extract useful features
    - Output of neurons are connected to local regions in the input
    - Applying the same filter on the whole image (weight sharing)
  - **Pooling Layer**
    - Perform a down-sampling operation along the spatial dimensions
    - Increase feature invariance



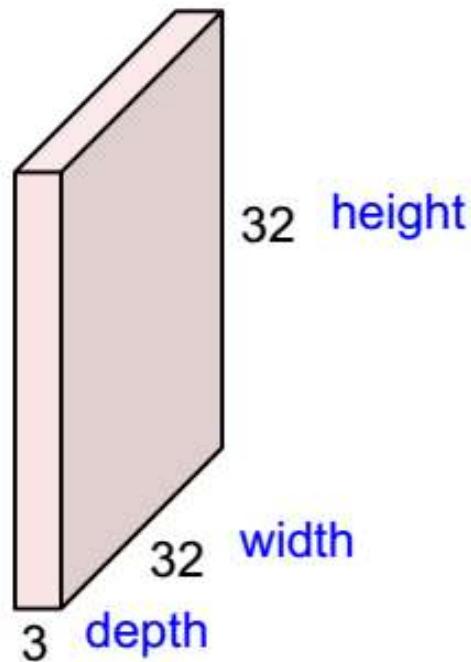
# Convolutional Neural Network

- Three main types of layers
  - **Convolutional Layer**
    - Extract useful features
    - Output of neurons are connected to local regions in the input
    - Applying the same filter on the whole image (weight sharing)
  - **Pooling Layer**
    - Perform a down-sampling operation along the spatial dimensions
    - Increase feature invariance
  - **Fully-Connected Layer (typical layer in MLPs)**
    - Classification or regression based on extracted features



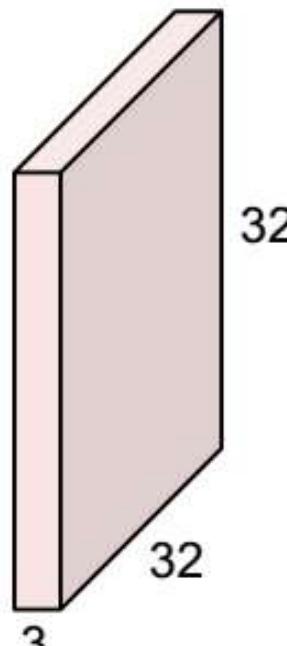
# Convolution Layer

32x32x3 image

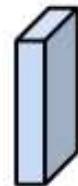


# Convolution Layer

32x32x3 image

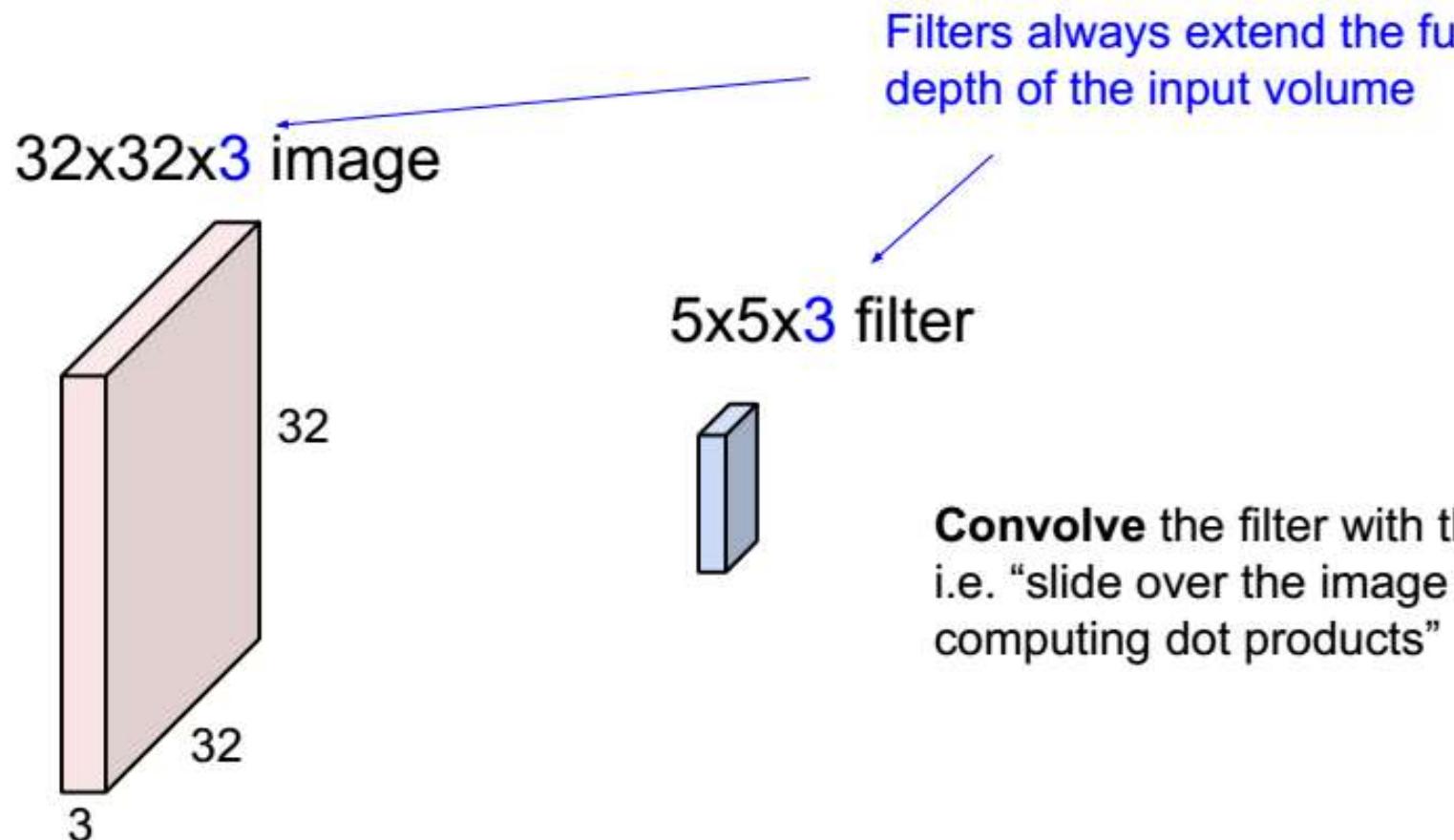


5x5x3 filter

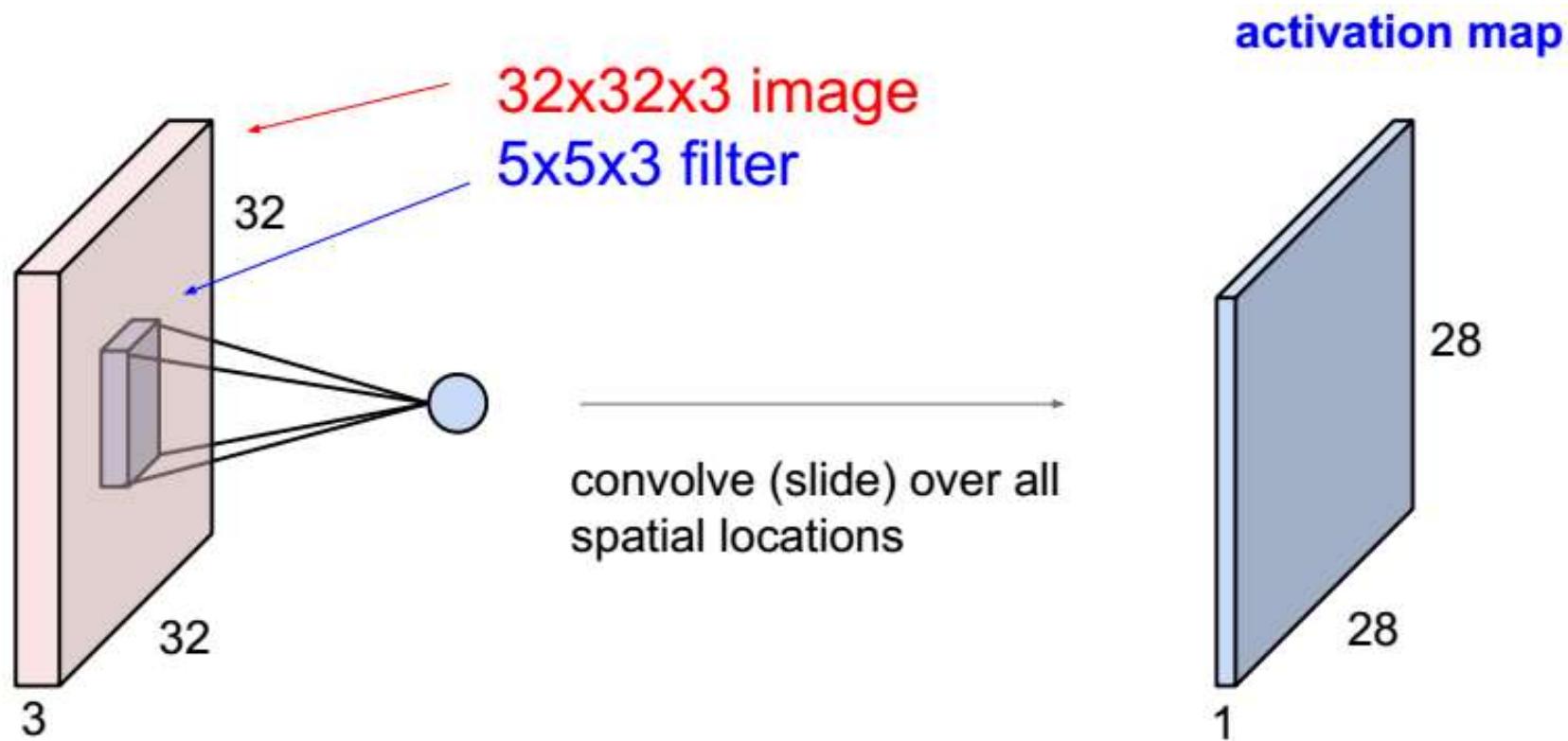


**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

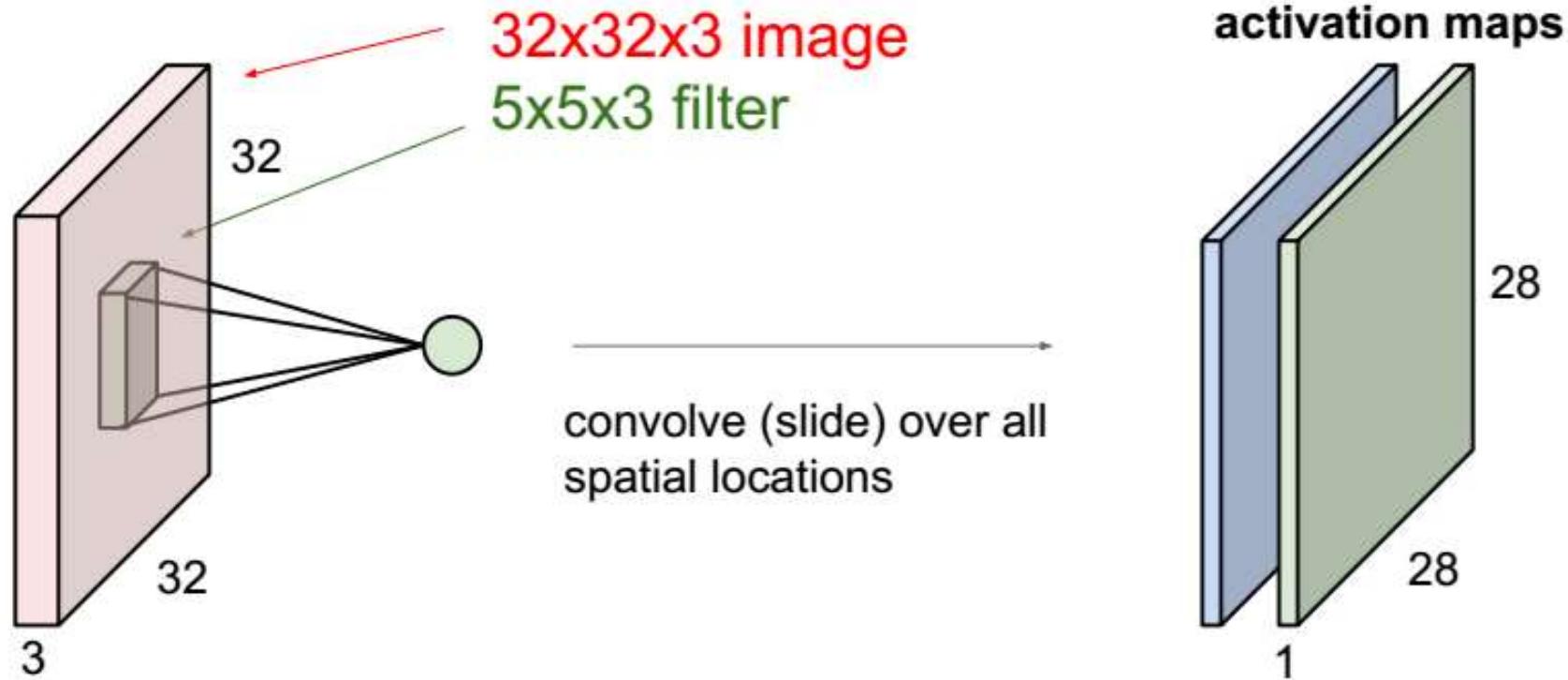


# Convolution Layer



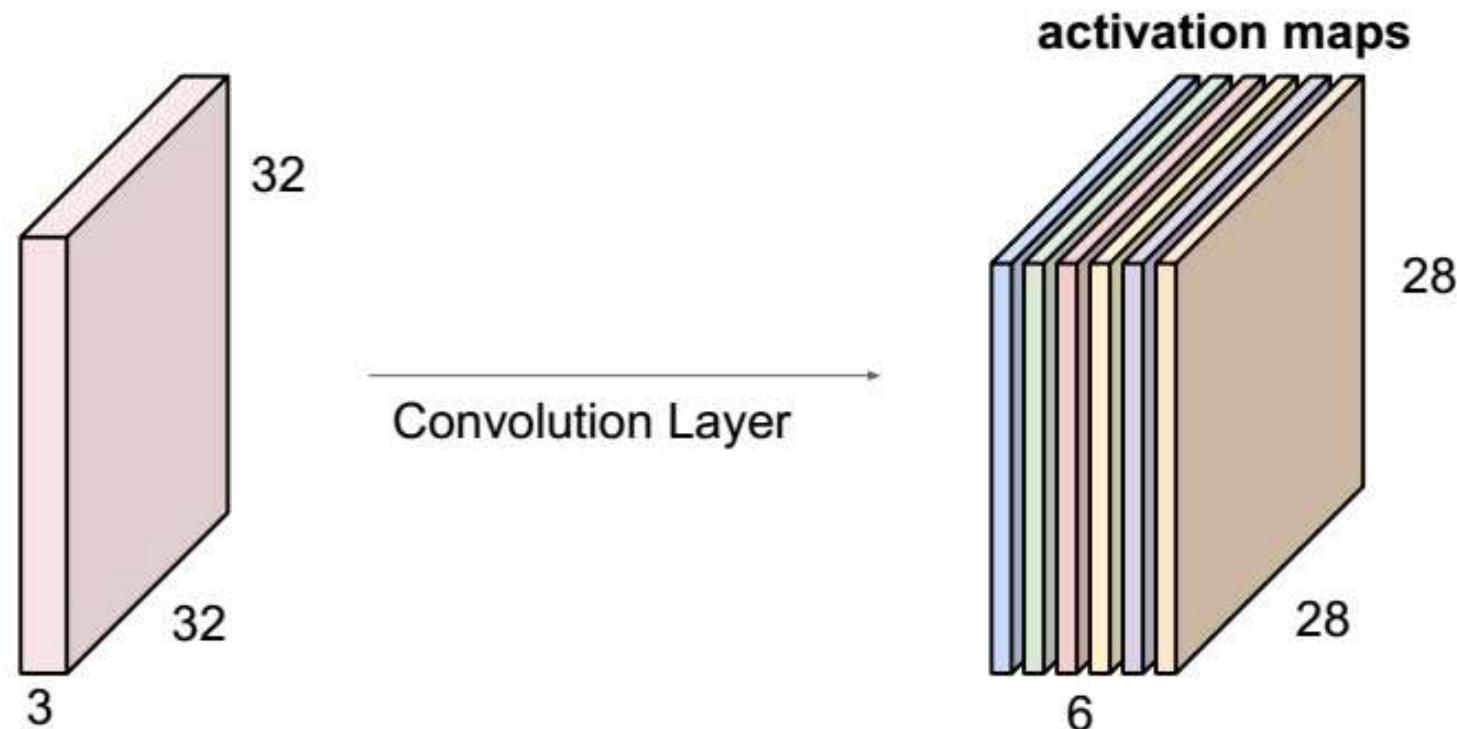
# Convolution Layer

consider a second, green filter



# Convolution Layer

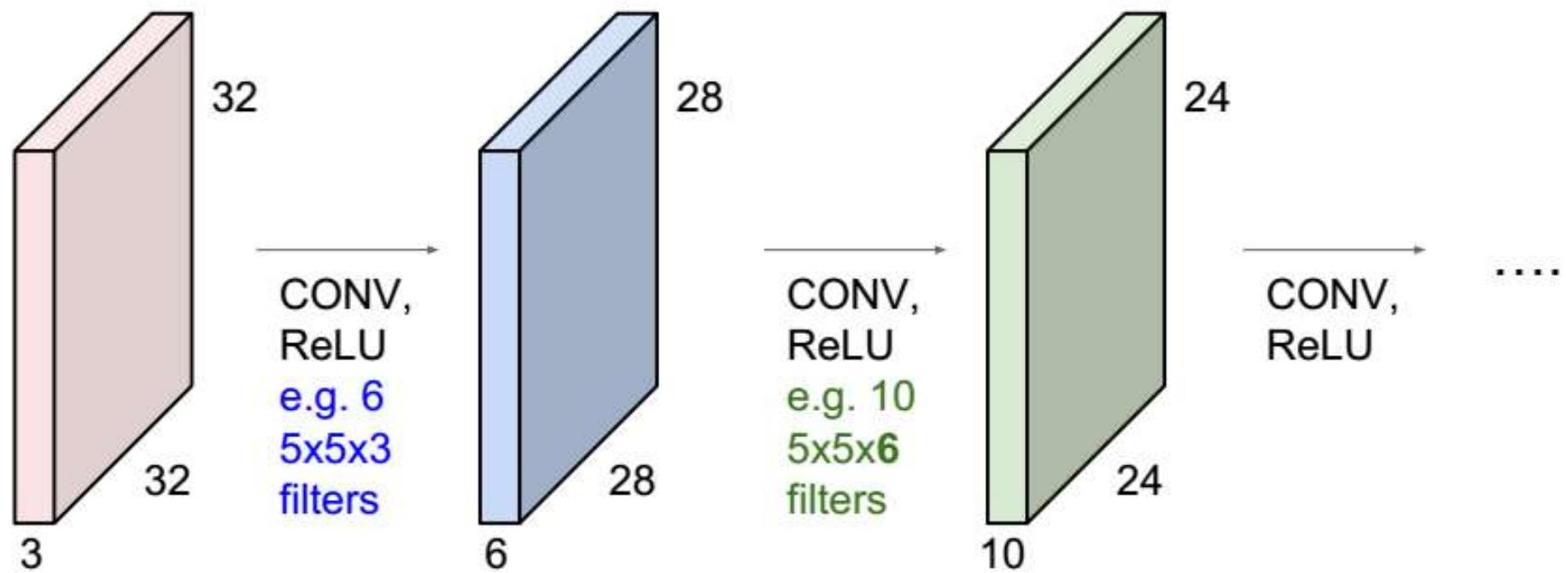
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size  $28 \times 28 \times 6$ !

# Convolution Layer

**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



# Recurrent Neural Networks (RNN)

Processing Sequential Data:

- In many situations one must consider a series of inputs to produce an output.
- RNNs have a memory which captures information about what has been seen so far.

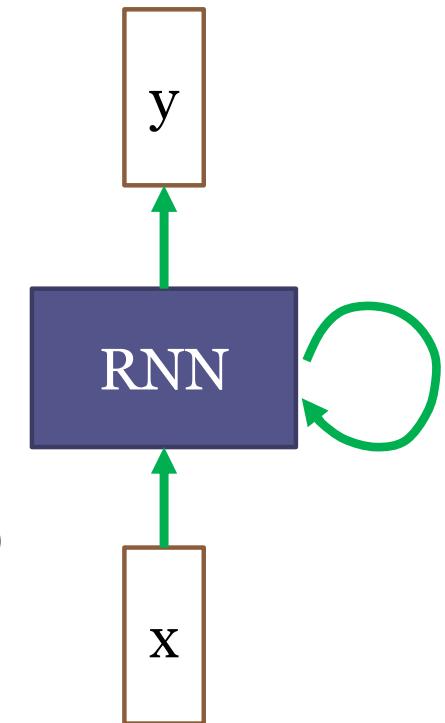


# Recurrent Neural Networks: Vanilla RNN

- We can process a sequence of vectors  $x$  by applying a recurrence formula at every time step:

hidden vector

$$h_t = f_w(h_{t-1}, x_t)$$



- Note that the same function and the **same** set of parameters are used at every time step

# Recurrent Neural Networks: Vanilla RNN

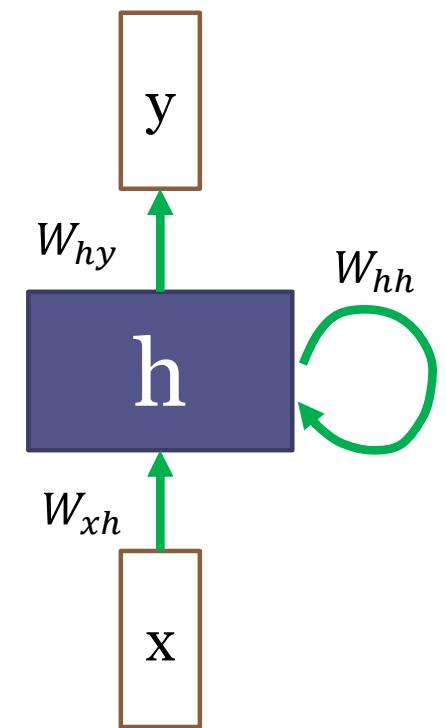
- Every time step we calculate  $h_t$ :

$$h_t = f_w(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

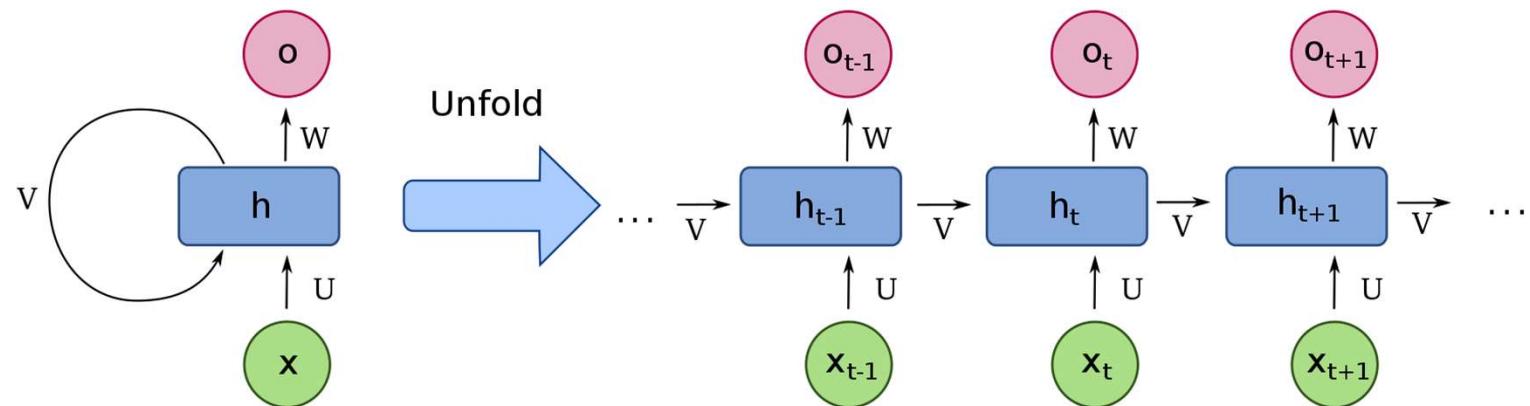
- When we need  $y$ , it's obtained by:

$$y_t = W_{hy}h_t$$



# RNN: Computational Graph

- It's useful to **unfold** a recurrent computation into a computational graph that has a repetitive structure, typically corresponding to a chain of events.



# References

- Dr. Soleymani courses: ML & Deep
- Pattern Recognition and Machine Learning (Christopher Bishop)
- Foundations of Machine Learning (Mehryar Mohri)
- Computational intelligence: a methodological introduction (Kruse, et al.)
- Deep Learning (Goodfellow, et al.)