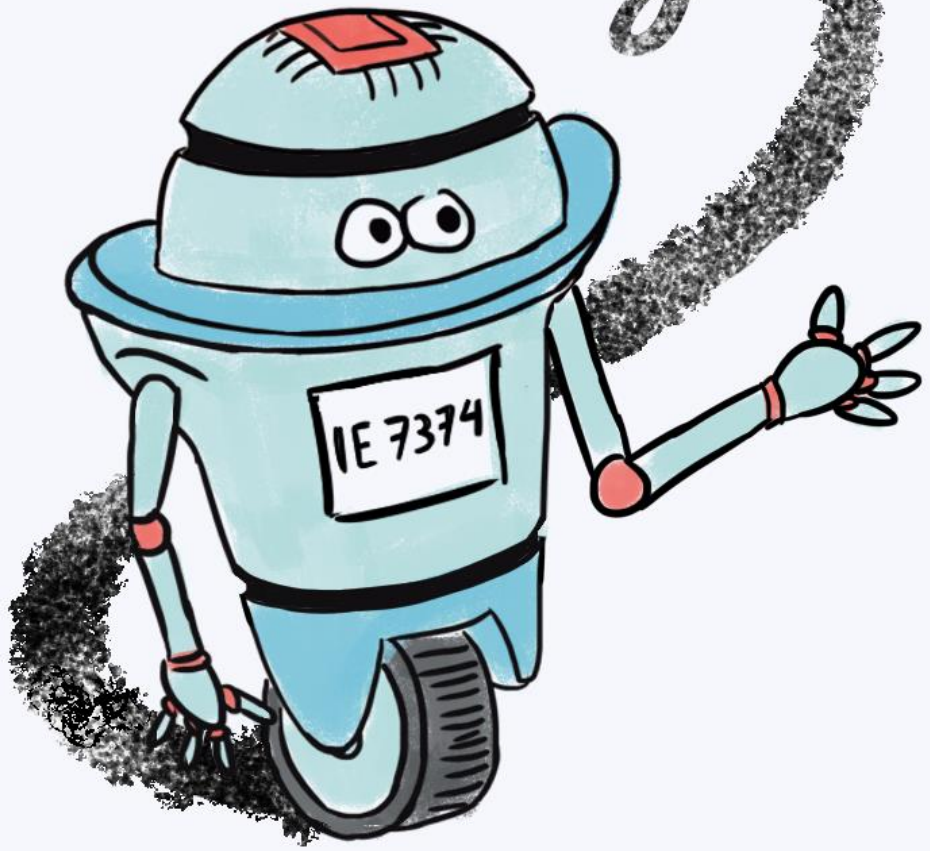
Reinforcement Learning



On-policy Prediction with Approximation (Part b)

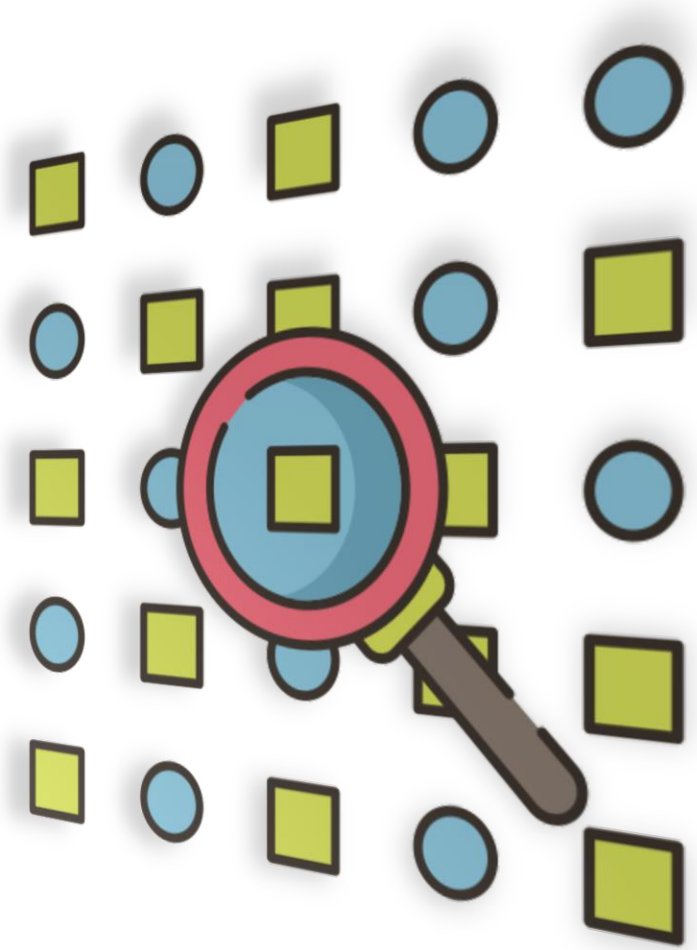
Mohammad **Dehghani**

Mechanical and Industrial Engineering Department

Northeastern University

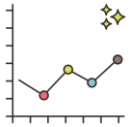
Apr-20

Feature Construction for Linear Methods



Feature Construction

Introduction



Linear methods are interesting and favorable because:

- their convergence guarantees
- their efficiency in terms of both data and computation.

The quality of approximation in linear methods critically deepens on feature selection

- adding prior domain knowledge to reinforcement learning systems.
- features should correspond to the aspects of the state space along which generalization may be appropriate.

Expert knowledge could benefit to design good features.

- appropriate feature selection helps linear methods to learn quickly and estimate values with high accuracy.

Feature Construction

Example: Car price prediction

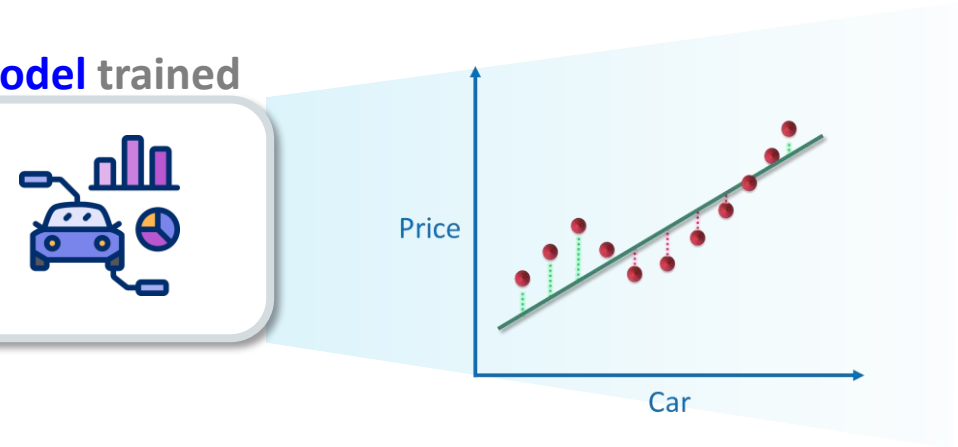
Input

Cylinder
4
6
...
4

Target

Price
$ 30,000
$ 12,000
...
$ 3,000

Model trained



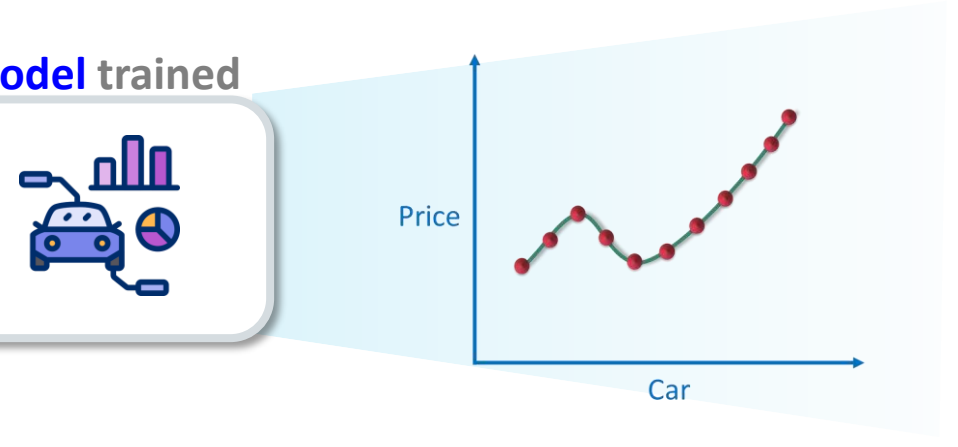
Input

Odometer	Cylinder	MPG
10,000	4	28
50,000	6	22
...	...	...
73,000	4	19

Target

Price
$ 30,000
$ 12,000
...
$ 3,000

Model trained



Feature Construction

Limitation of linear methods

$$\hat{v}(s) = w_1 X + w_2 Y$$

Y \ X	1	2	3	4	5
1	2	3	4	5	6
2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9
5	6	7	8	9	10

$w_1 = 1, w_2 = 1$

$$\hat{v}(s) = w_1 X + w_2 Y$$

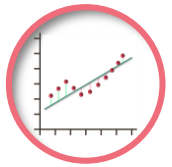
Y \ X	1	2	3	4	5
1	0	0	0	0	0
2	0	4	5	6	0
3	0	5	6	7	0
4	0	6	7	8	0
5	0	0	0	0	0

Blue states $w_1 = 1, w_2 = 1$

Red states $w_1 = 0, w_2 = 0$

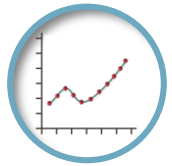
Feature Construction

Polynomials



Linear models are simple, but might be inadequate to perform FVA.

- The effect of the predictors on the output variable may not be linear in nature, therefore, ***under-fitting*** may occur.



One of the possible ways to improve FVA, is using Polynomials

- Polynomials Make up one of the simplest families of features used for interpolation and regression.
- Polynomial models can perfectly interpolate any data

Polynomials take into account any interactions between states dimensions

- i.e : a single representative state s, with two numbers (s_1, s_2)

$$\mathbf{x}(\mathbf{s}) = (s_1, s_2)^{\mathrm{T}}$$

$$\mathbf{x}(\mathbf{s}) = (1, s_1, s_2, s_1 s_2)^{\mathrm{T}}$$

affine functions

$$\mathbf{x}(\mathbf{s}) = \left(1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2\right)^{\mathrm{T}}$$

Feature Construction

k-dimensional Polynomials

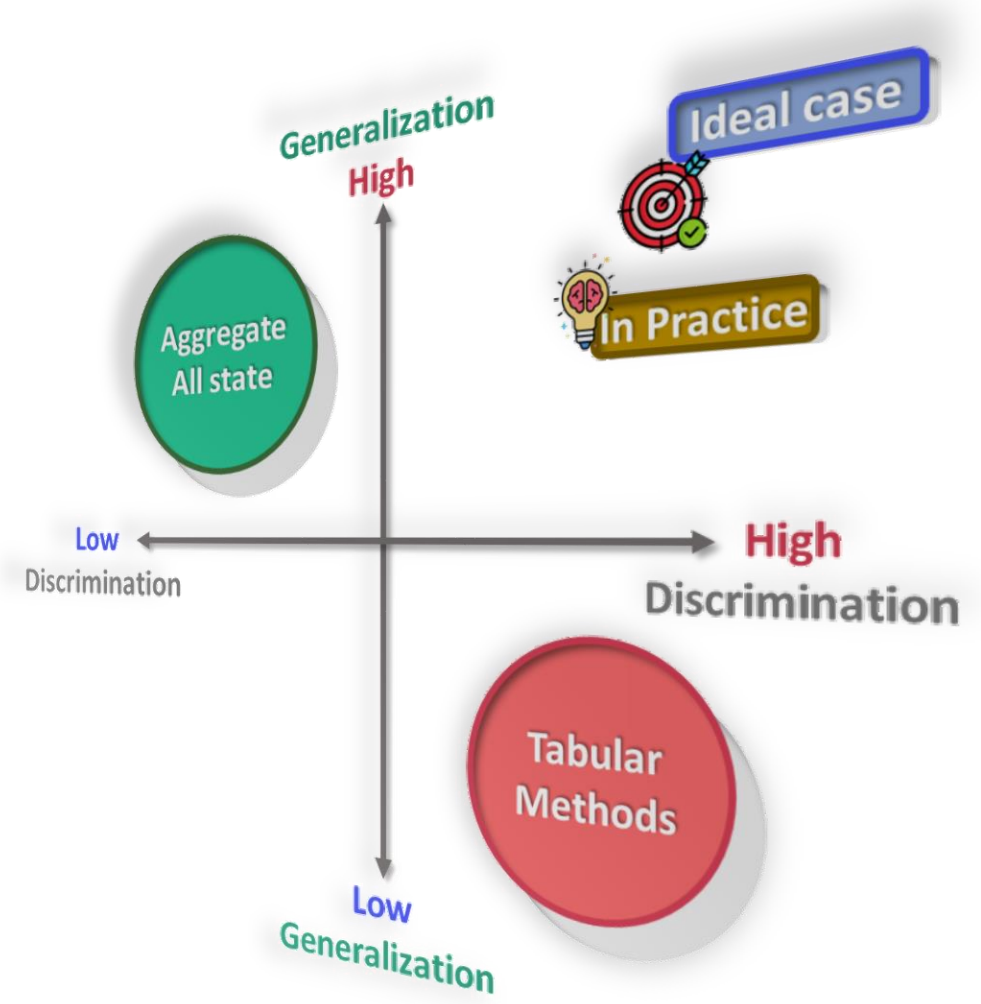
Suppose each state s corresponds to k numbers, s_1, s_2, ..., s_k, with each $s_i \in \mathbb{R}$. For this k-dimensional state space, each order-n polynomial-basis feature x_i can be written as

$$x_i(s) = \Pi_{j=1}^{k} s_j^{c_{i,j}}, \tag{9.17}$$

where each $c_{i,j}$ is an integer in the set $\{0, 1, \ldots, n\}$ for an integer $n \geq 0$. These features make up the order-n polynomial basis for dimension k, which contains $(n+1)^k$ different features.

- **Higher-order polynomial bases allow for more accurate approximations of more complicated functions**
  - The number of features in an order-n polynomial basis grows exponentially
  - So, it is generally necessary to select a subset of them for function approximation.

Generalization and Discrimination

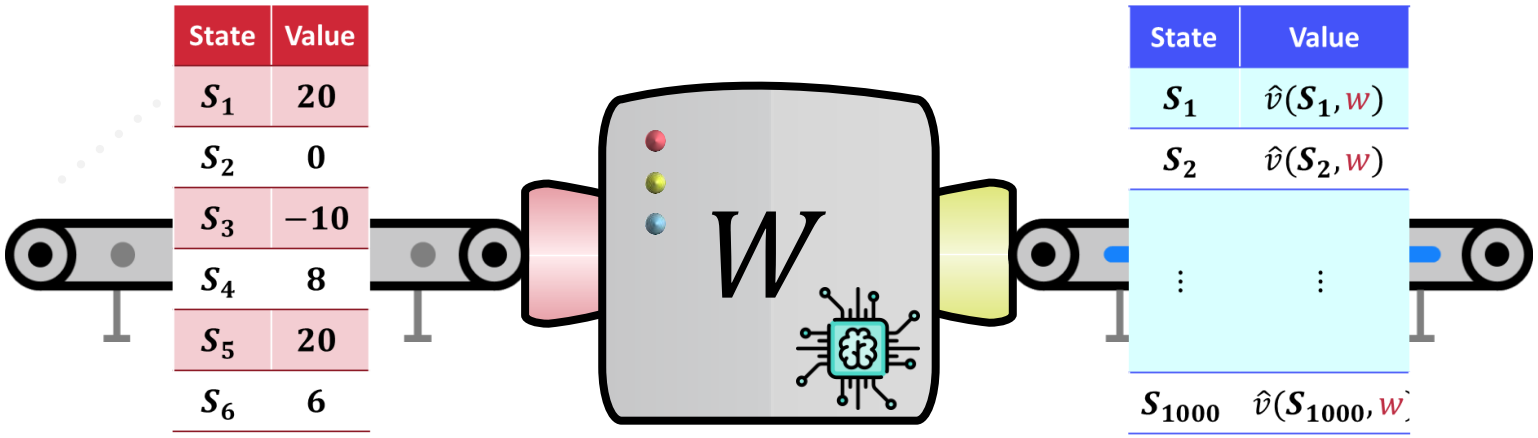


Generalization and Discrimination

Generalization

- **Generalization is a form of abstraction**
  - Intuitively generalization means that you predict well on unseen samples
    - As humans, we generalize with incredible ease. (ride bicycle, drive cars, etc.)
    - What about machines?

- **In RL, intelligent agents would be able to generalize between tasks, using prior experience to pick up new skills more quickly.**



$$\hat{v}(s, w) \approx v_\pi(s)$$

Generalization and Discrimination

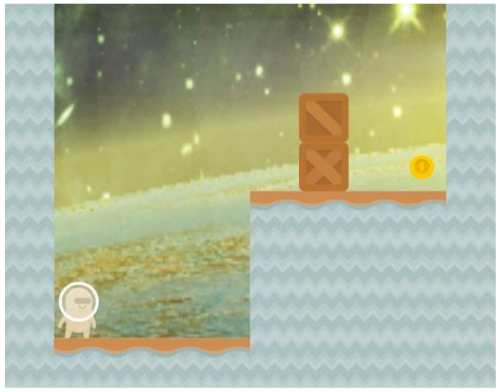
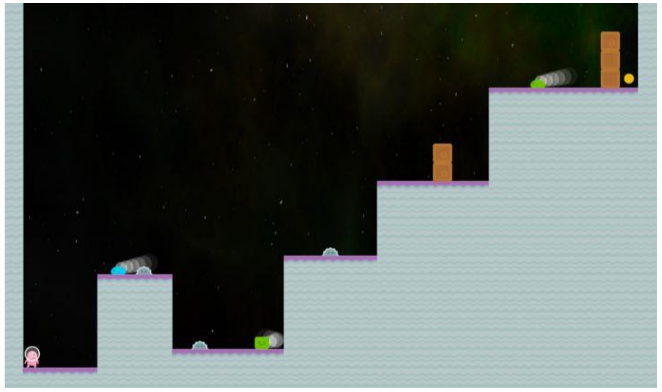
Example: Generalization in Coin Run

Generalization in different environments



Generalization in different levels

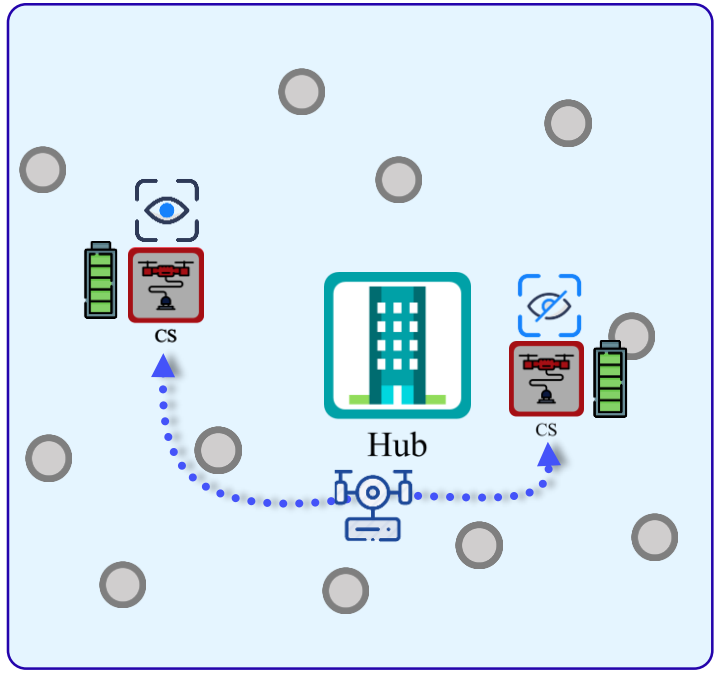
Level 1 to 3

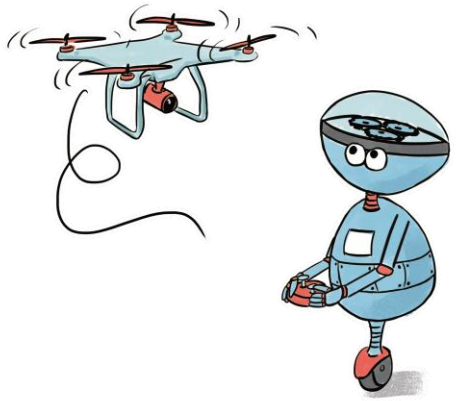



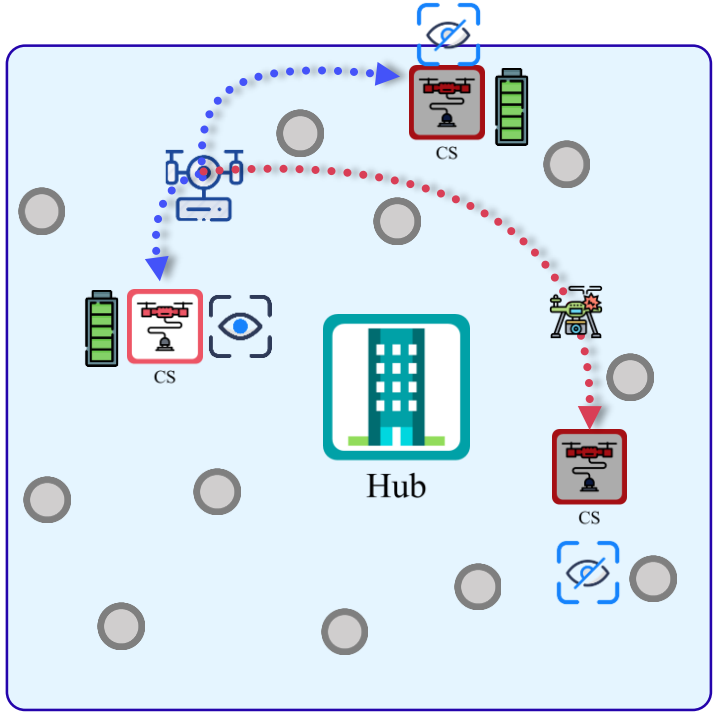
OpenAI

Generalization and Discrimination

Example: Generalization in Drone routing







Generalization and Discrimination

Discrimination

Discrimination learning is defined in psychology as the ability to respond differently to different stimuli.

- **Animal:** A dog might be trained to use discrimination learning to detect differences in complex odor compounds so that they are able to sniff out different drugs to assist police.
- **Human:** baby who reacts differently to their mother's voice than to a stranger's voice.

In RL, discrimination means the ability to distinguish between states to make the values for two states differently.

- In the drone example, it is useful to distinguish between charging station based on their distance from the drone or some other factors.

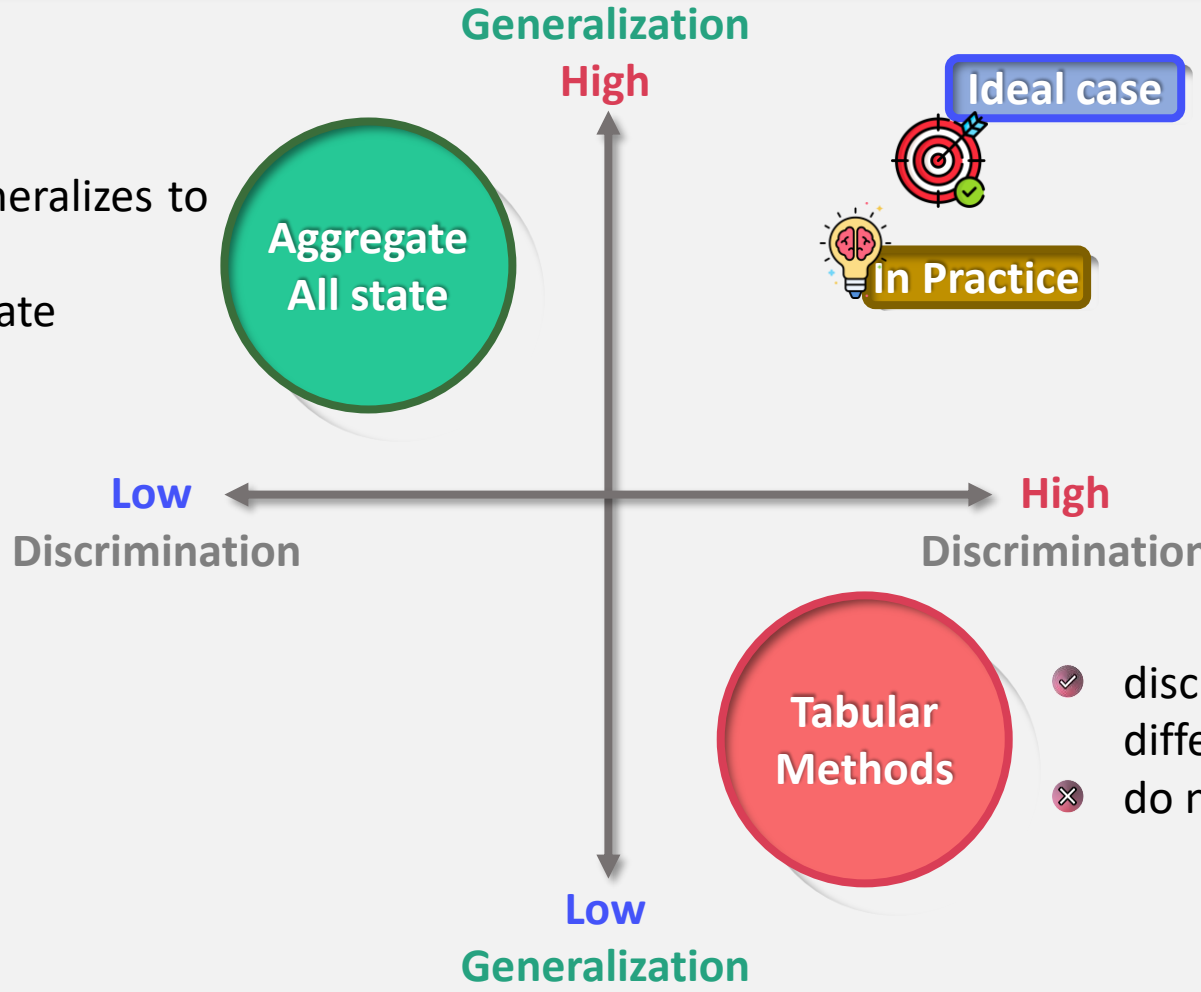
So, there is a trade-off between generalization and discrimination

- It is important to use generalization to enable agnet to learn state values quickly.
- At the same time, it is important to consider critical difference between states that likely would impact their value.

Generalization and Discrimination

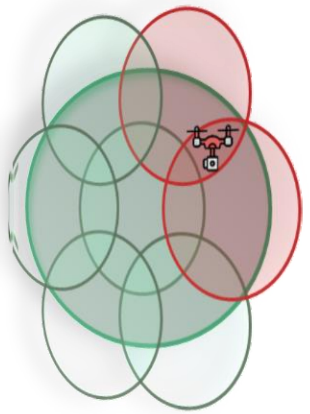
Generalization and Discrimination Trade-off

- Each update generalizes to all states
- do not discriminate



- discriminate between different states perfectly
- do not generalize

Coarse Coding

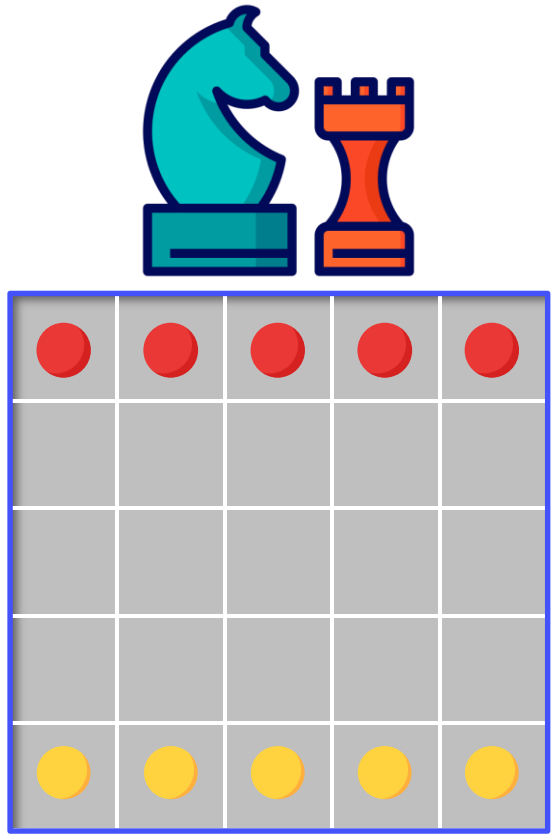


$$\mathbf{x}(s) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Coarse Coding

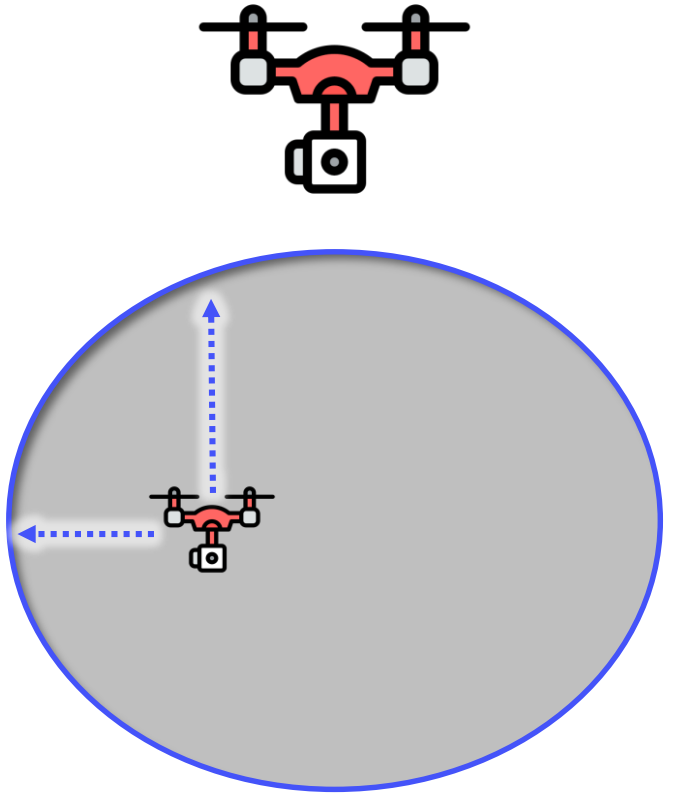
Discrete vs. Continuous Space

- **In general, there two possible environment spaces in RL:**
  - **Discrete**, and **Continuous**



$s \in \{s_0, s_1, \cdots, s_n\}$

$a \in \{a_0, a_1, \cdots, a_m\}$



$s \in \mathbb{R}^n$

$a \in \mathbb{R}^n$

Coarse Coding

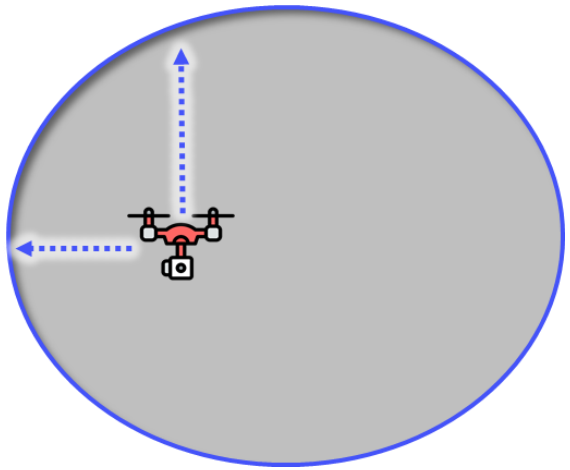
Tackling the Continuity problem

- **Obviously for many situations, the environment may not be discrete but continuous.**
  - Discretize spaces, then use regular RL methods
    - e.g. Coarse coding, Tile coding:
    - But: How fine-grained? Where to put focus? Bad generalization.
  - Use parameter vector $\vec{\theta}_t$ of a function approximator for updates
    - often neural networks are used and the weights as parameters

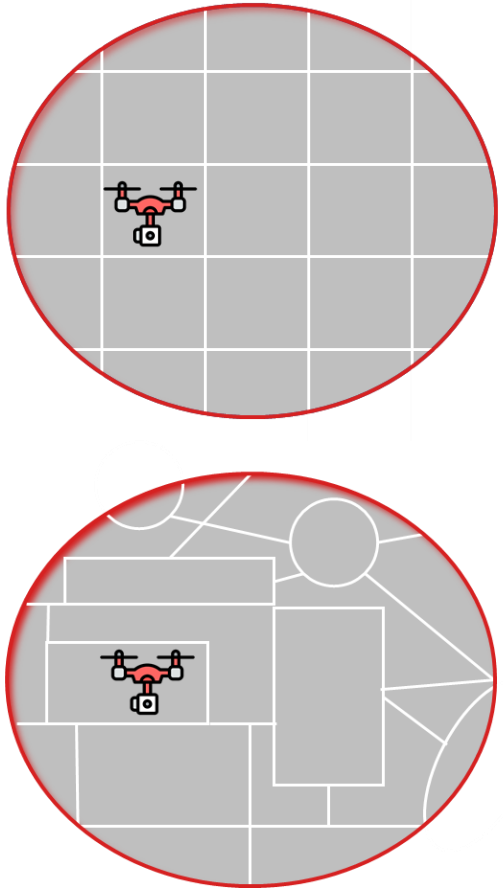
Coarse Coding

Discretize spaces by Coarse Coding

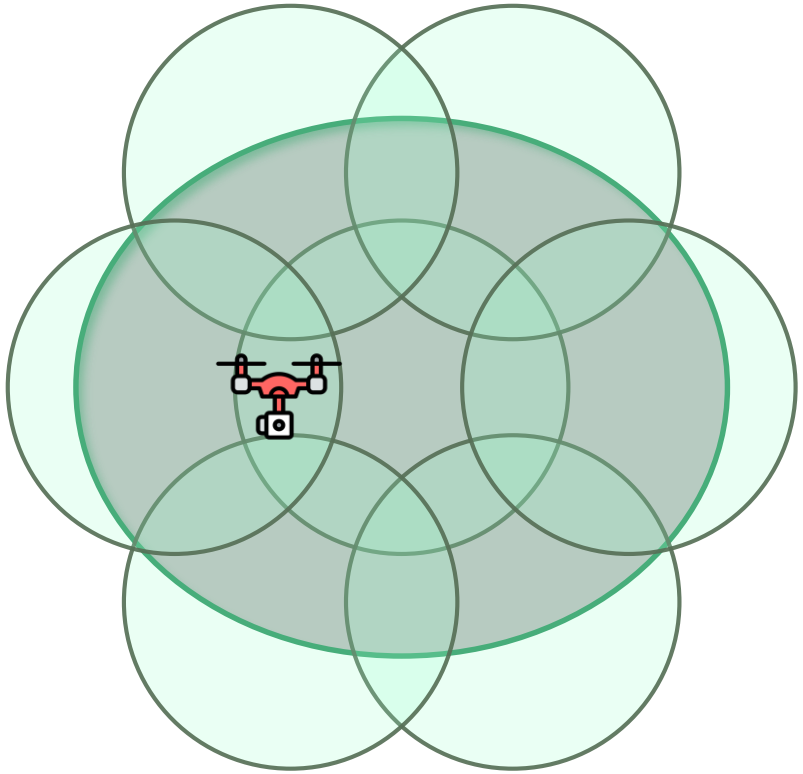
Continuous Env. space



State aggregation



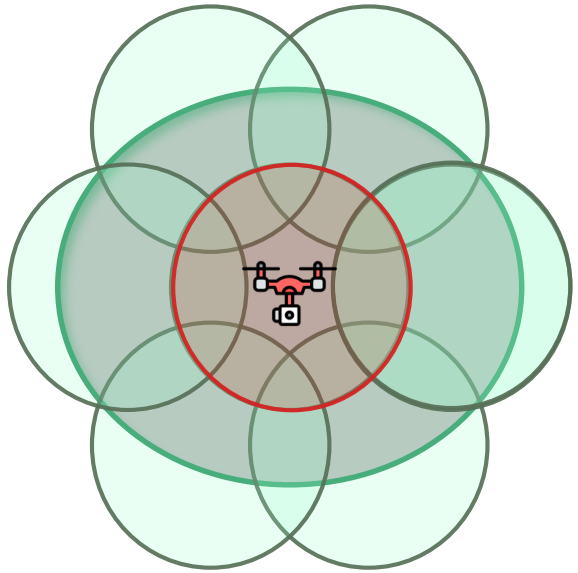
Coarse coding



Coarse Coding

Flexible feature representations with Coarse Coding

- **Coarse coding enables us to have a flexible class of feature representations**



$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{x}(\mathbf{s}) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

1 **Present** feature

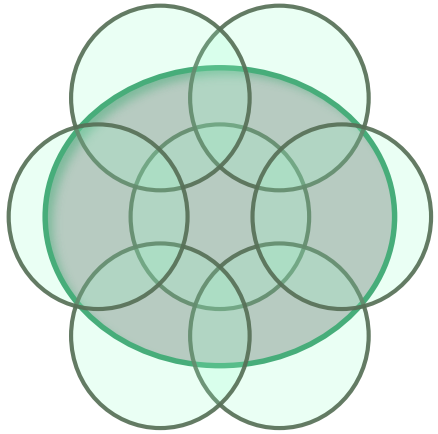
0 **Absent** feature

$$\hat{v}(s, \boldsymbol{w}) \doteq \mathbf{w}^{\mathbf{T}}\mathbf{x}(s)$$

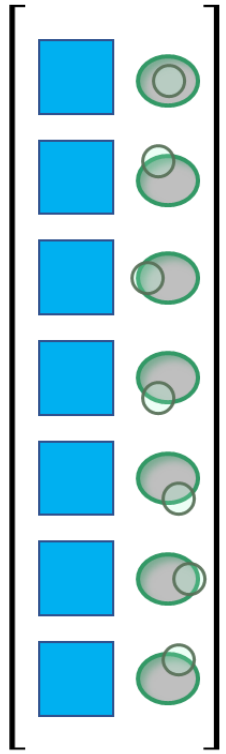
Coarse Coding

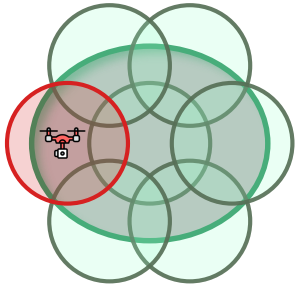
Flexible feature representations with Coarse Coding

- **Coarse coding enables us to have a flexible class of feature representations**

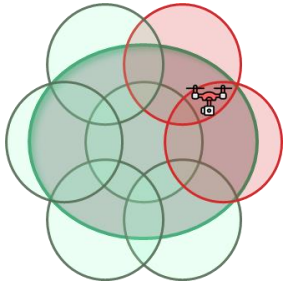


$$x(s) =$$

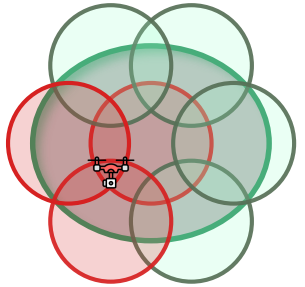




$$\mathbf{x}(\mathbf{s}) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



$$\mathbf{x}(\mathbf{s}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



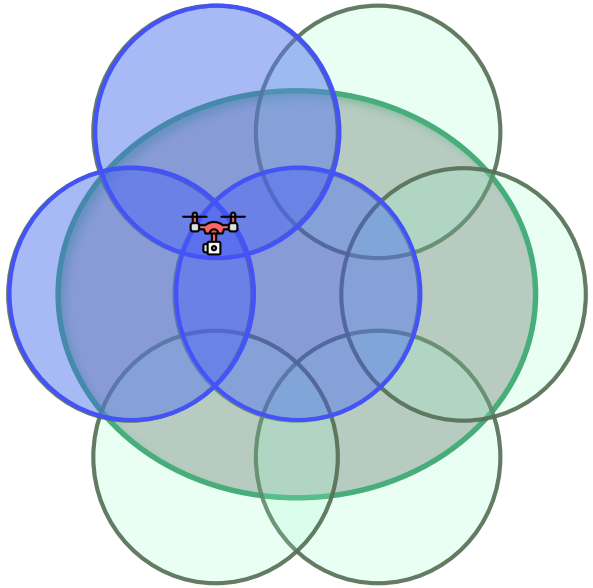
$$\mathbf{x}(\mathbf{s}) = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Coarse Coding

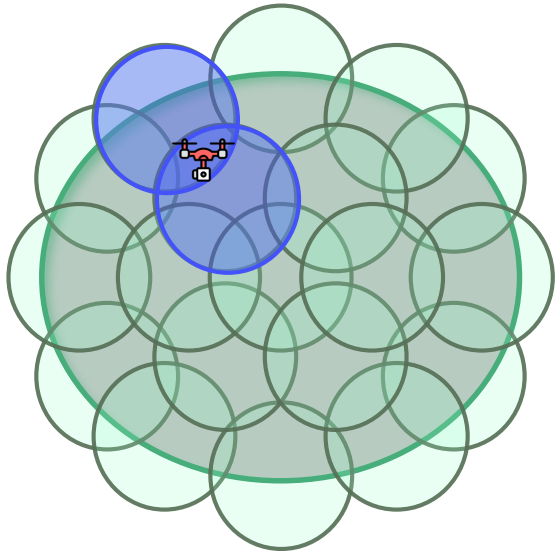
Generalization in coarse coding

- **Changing the properties of coarse coding affects generalization and discrimination.**
  - Properties such as ***sizes*** and ***shapes*** of the features' receptive fields

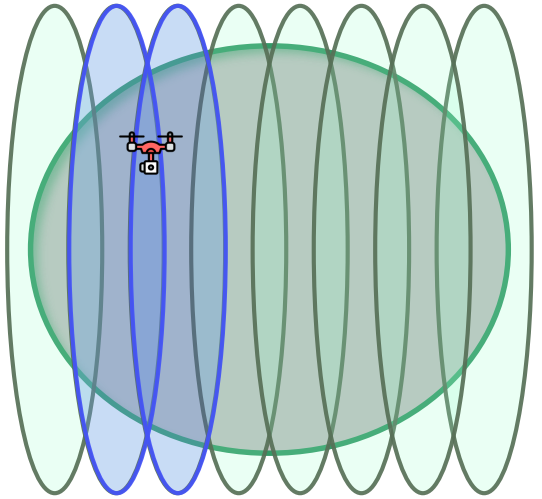
Broad generalization



Narrow generalization



Asymmetric generalization

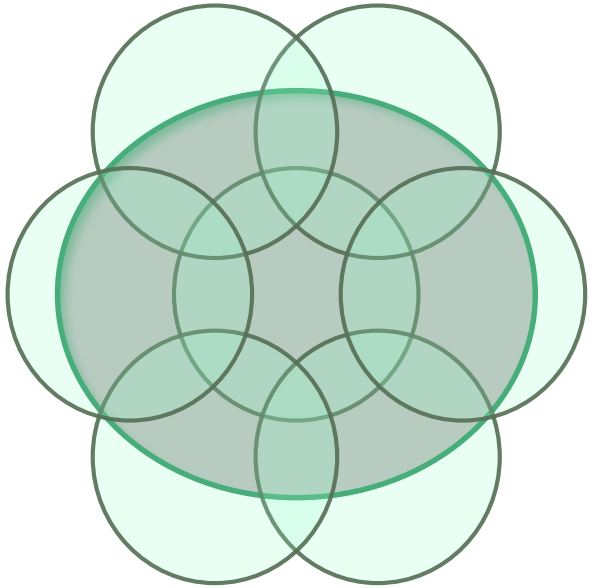


Coarse Coding

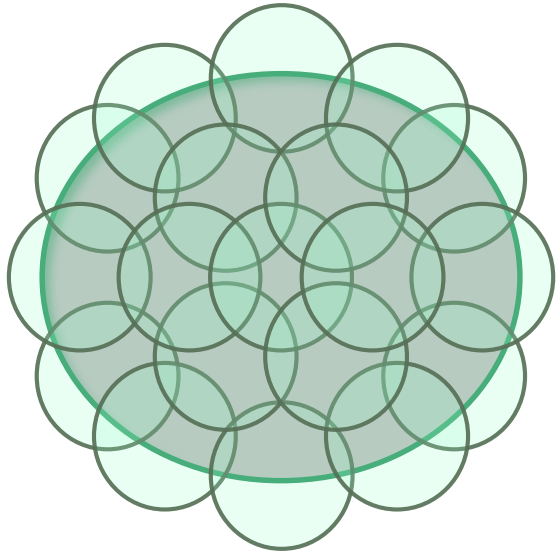
Discrimination in coarse coding

- **In coarse coding, the overlap between receptive fields indicates the level of discrimination**

Low Discrimination

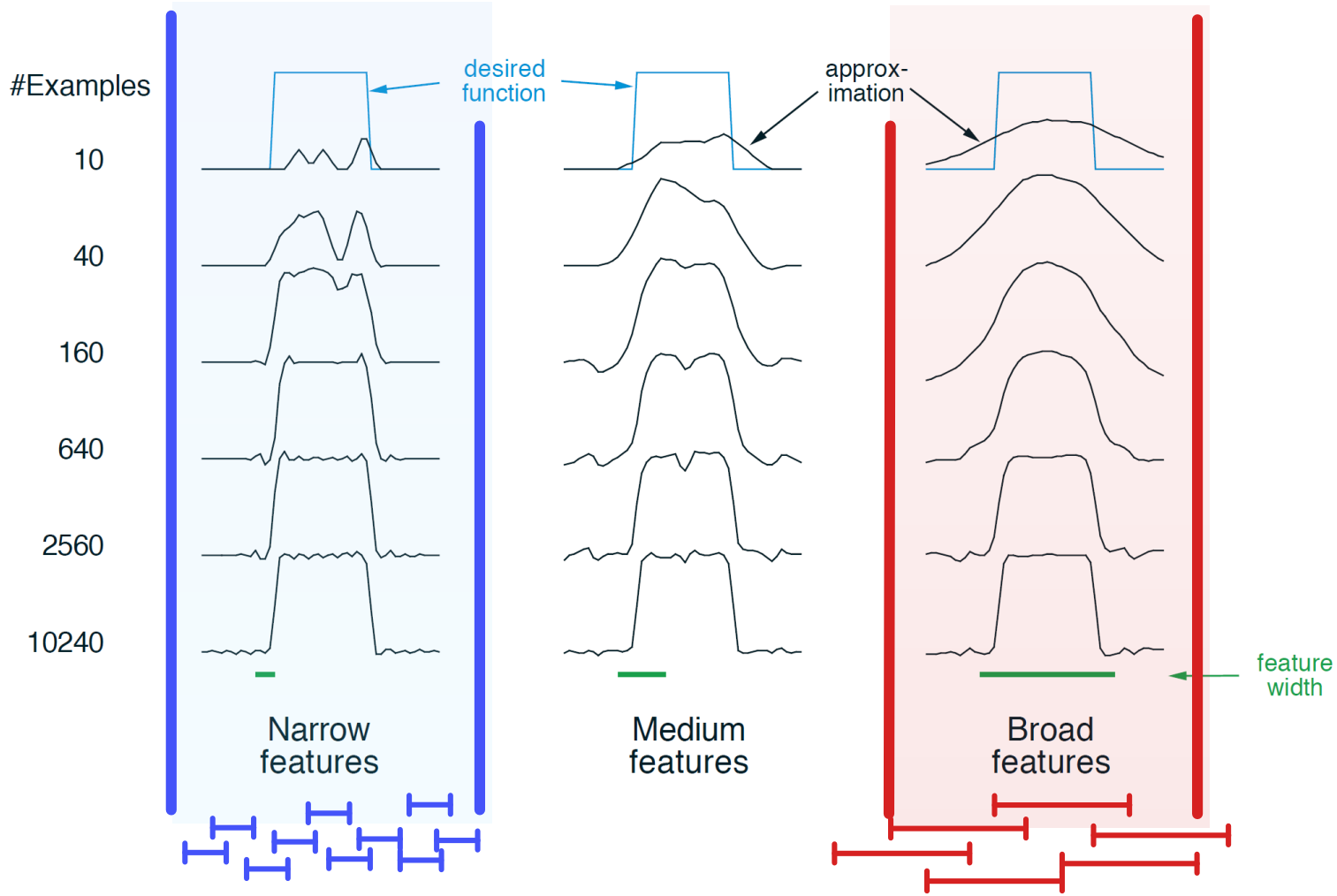


High Discrimination

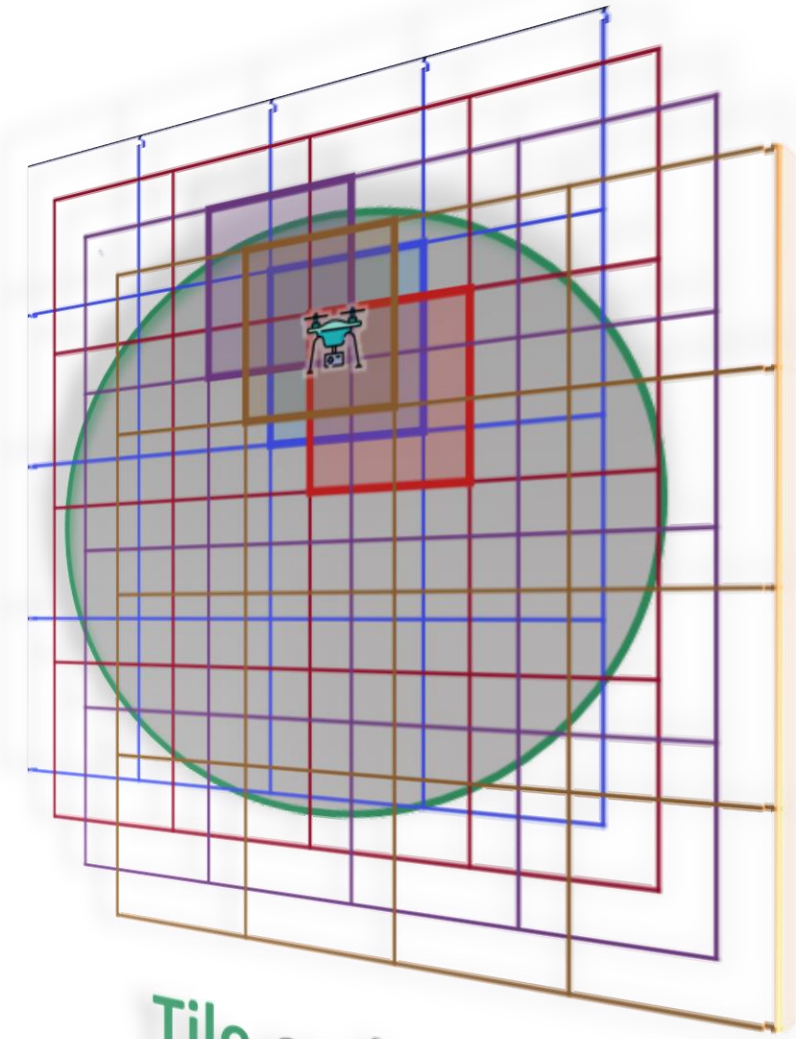


Coarse Coding

Coarseness of Coarse Coding: one-dimensional square-wave function



Tile Coding

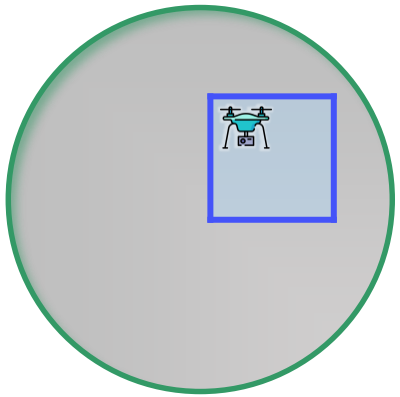


Tile Coding

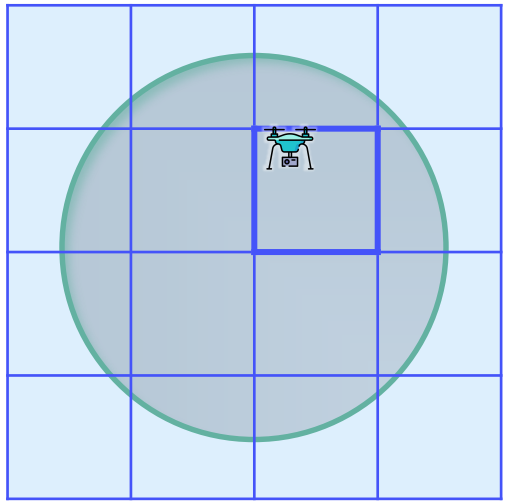
An exhaustive version of coarse coding

In tile coding the receptive fields of the features are grouped into partitions of the state space.

- Each such partition is called a ***tiling***, and each element of the partition is called a ***tile***.

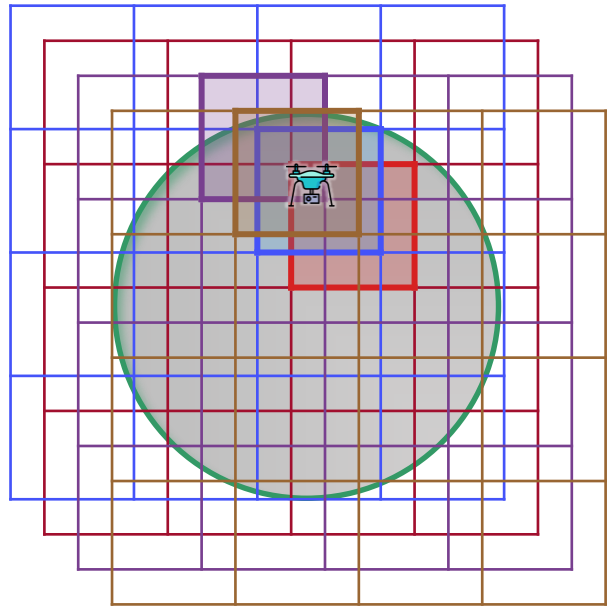


Tile



Tiling

State aggregation



Tile coding

Tile Coding

Benefits

In tile coding the receptive fields of the features are grouped into partitions of the state space.

- Each such partition is called a ***tiling***, and each element of the partition is called a ***tile***.

Tile coding uses several overlapping tilings and for each tiling, maintains the weights of its tiles.

- The approximate value of a given point is found by summing the weights of the tiles, one per tiling, in which it is contained

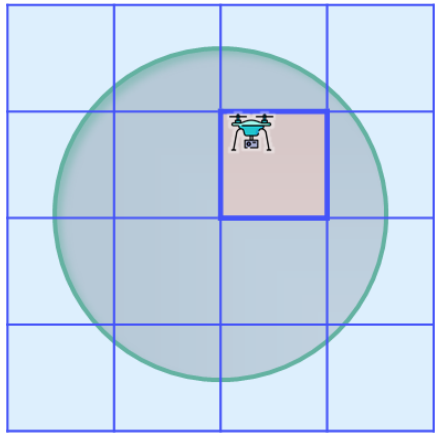
Tile coding strikes an empirically successful balance along:

- Representational power
- Computational cost
- Ease of use

Tile Coding

Feature vector size

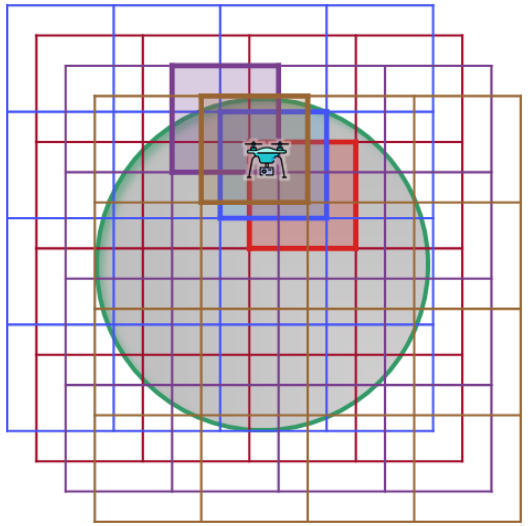
- **Every state, falls in exactly one tile in each of tilings.**
  - Each of tiles correspond to a feature that become active when the state occurs.
  - The feature vector $\mathbf{x}(\mathbf{s})$ has one component for each tile in each tiling.



$$\mathbf{x}(\mathbf{s}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Tiling

$4 \times 4 = 16$



$$\mathbf{x}(\mathbf{s}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

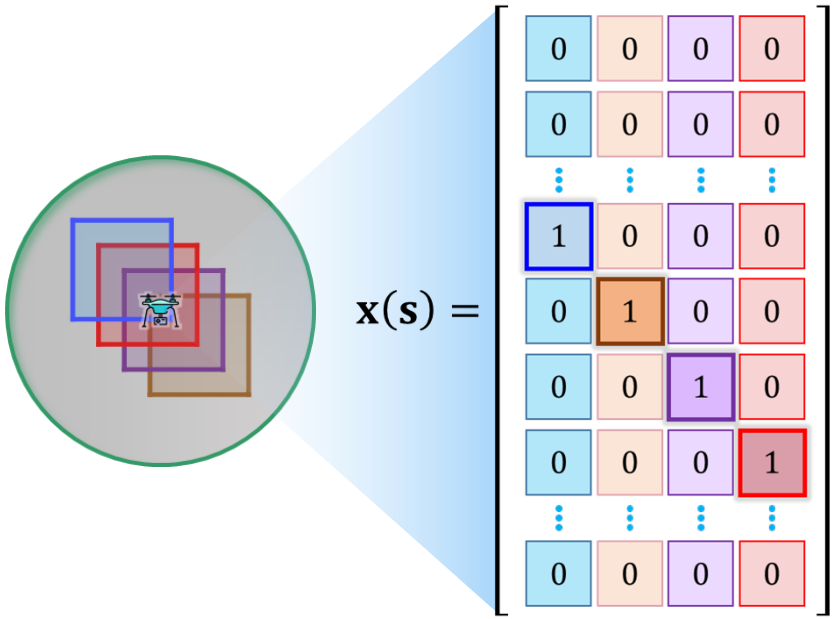
Tile coding

$16 \times 4 = 64$

Tile Coding

Parametrization

- **The state value updates is depend on the weights from all tilings**
  - Exactly one feature is present in each tiling, so the total number of features present is always the same as the number of tilings.
    - This allows the step-size parameter, alpha, to be set in an easy, intuitive way.

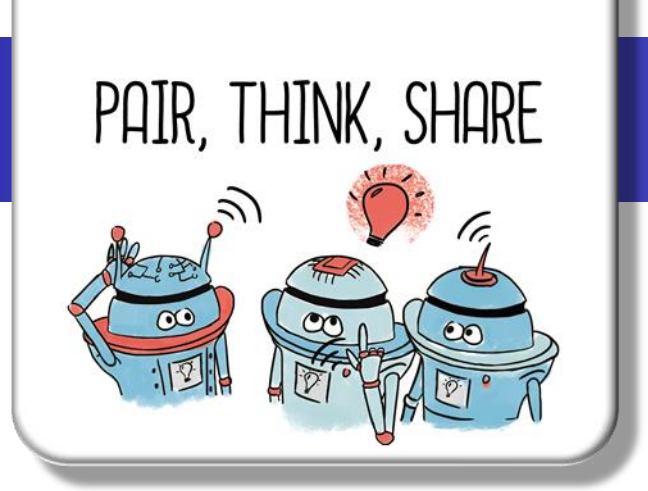


α
Update size

$\alpha = \frac{1}{n} \Rightarrow \hat{v}(s, \mathbf{w}_{t+1}) = v$
Tiling results in exact one-trial learning

$\alpha = \frac{1}{10n} \Rightarrow \hat{v}(s, \mathbf{w}_{t+1}) = \frac{1}{10}$ Target Value
trained state moves 1/10 of the way to the target

$$\hat{v}(\mathbf{w}, \mathbf{x}(s)) = \frac{1}{4}\sum \mathbf{w}.\mathbf{x}(s) = \frac{1}{4}\,(w_{1,i} + w_{2,j} + w_{3,k} + w_{4,l})$$

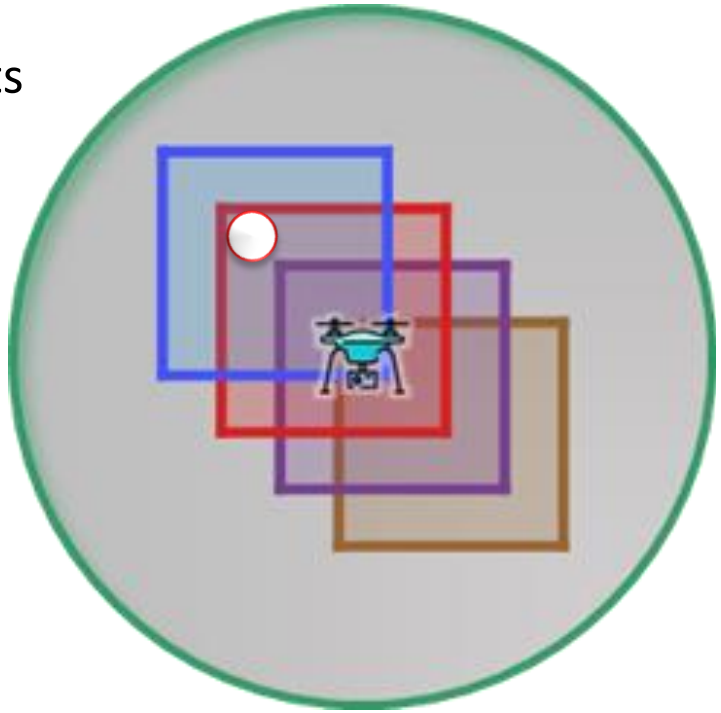


Tile coding Generalization

- The approximate value of a given point is found by summing the weights of the tiles, one per tiling, in which it is contained

1. **How Generalization can occurs to states other than the one trained**

(Assuming those states fall within any of the same tiles, i.e the **white state** highlighted in the figure)



Tile Coding computational power

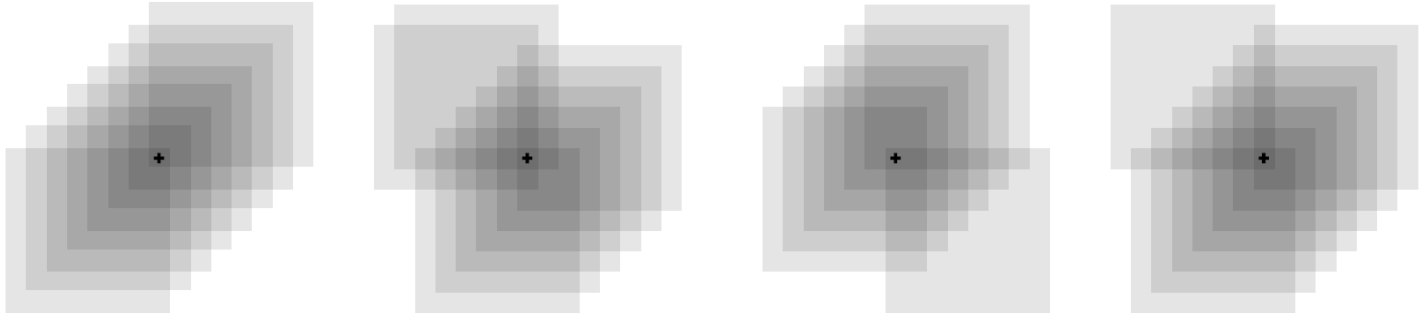
- Tile coding uses the binary feature vectors, because each component is either 0 or 1

2. **Why this relates with computational advantages in Tile coding?**

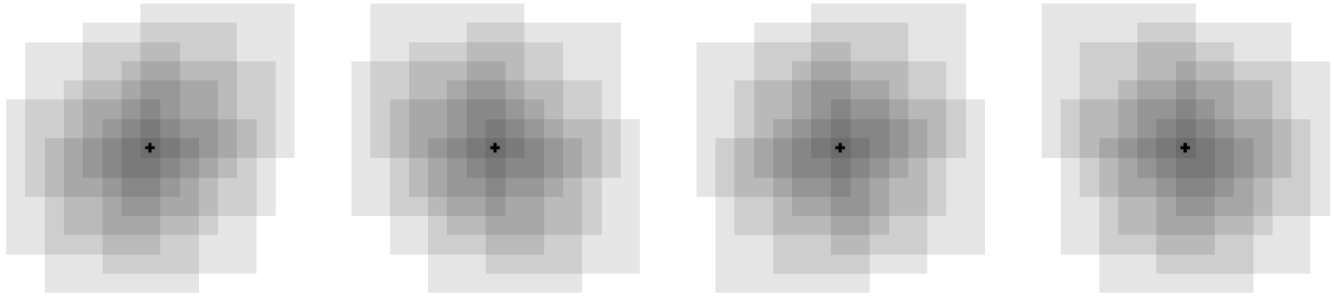
Tile Coding

Spherical and homogeneous Generalization

Uniformly offset tilings



- Diagonal artifacts
- Substantial variations



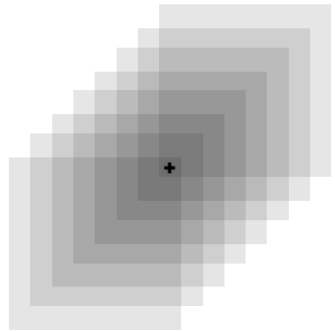
Asymmetrically offset tilings

- Better and Well centered
- More spherical and homogeneous Generalization

Tile Coding

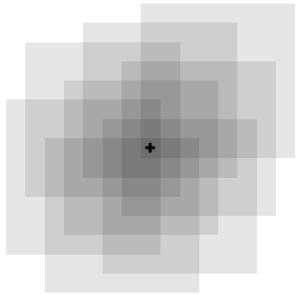
Tiling offset

- **Tilings in all cases are offset from each other by a *fraction* of a tile *width* in each dimension.**
- **Therefore, the *Tiling offset* is calculated based on two factor:**
  - **1: Displacement Vector**
  - **2: Fundamental unit :** $\frac{w}{n}$ ($\boldsymbol{w}$: tile width , $\boldsymbol{n}$: the number of tilings
    - i.e. in the 2D *uniformly offset tilings* the Displacement Vector is (1,1)
    - This means that the offset from the previous tiling by w/n times this vector.



  - For **asymmetrically offset tilings**, displacement vector is recommended to be consisting of the first odd integers

$$(1, 3, 5, 7, \ldots, 2\mathrm{k} - 1)$$



    - i.e. in the 2D *asymmetrically offset tilings* the Displacement Vector is (1,3)

Tile Coding

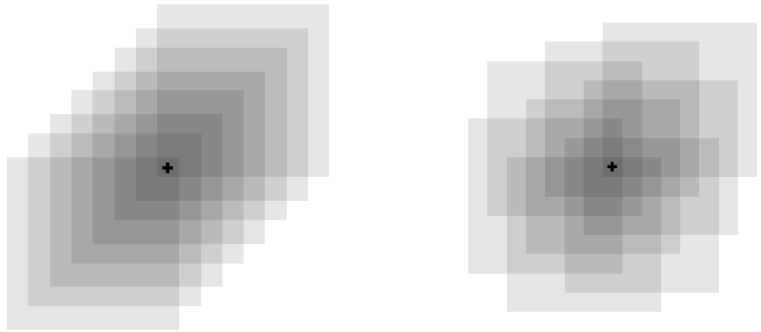
Number of tiling

- **The number of tilings, along with the size of the tiles, determines the resolution or fineness of the asymptotic approximation**
- ***In tile* coding, the number of tiling set to an integer power of 2 greater than or equal to 4k**

$$2^n \geq 4k$$

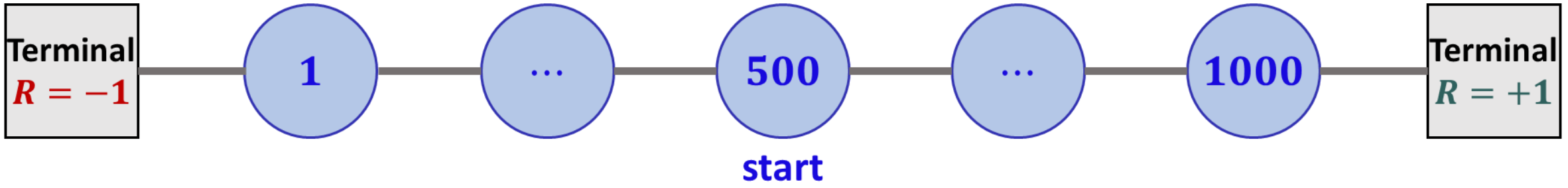
  - *K: # of dimensions*

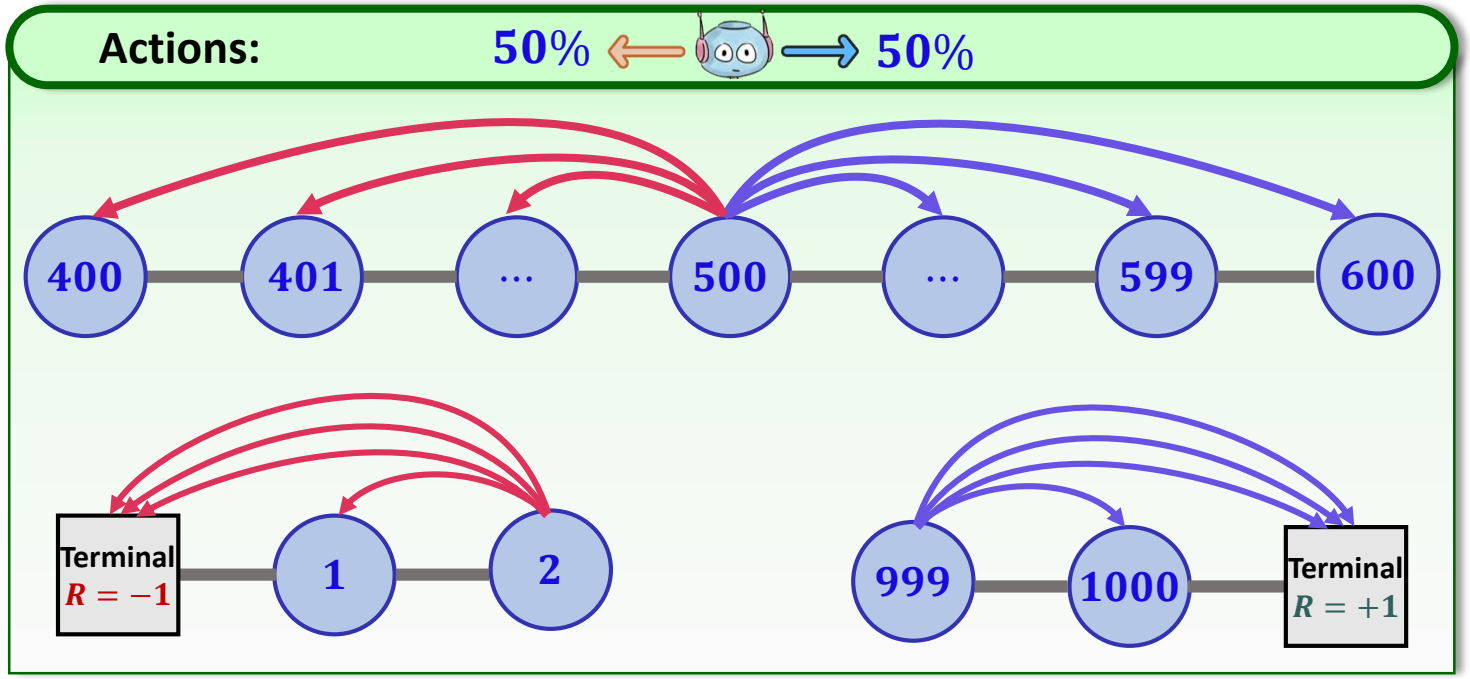
- *In 2D dimensional case: K=2, therefore, number of tiling is 8*
  - $n = 2^3 = 8 \geq 4 \times 2$



Tile Coding

Example: 1000-state random walk





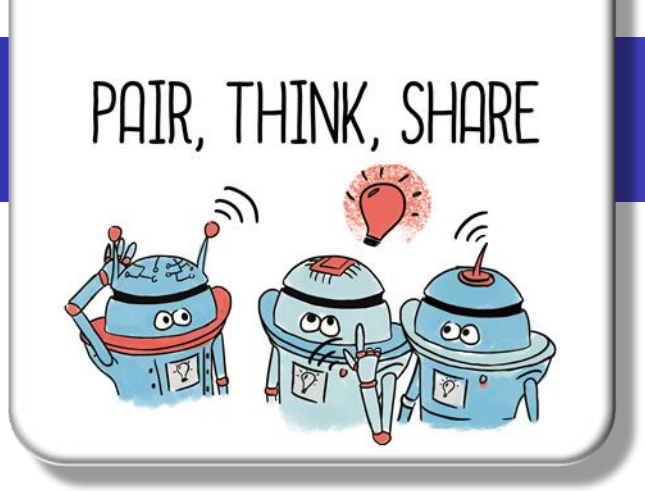
Settings:

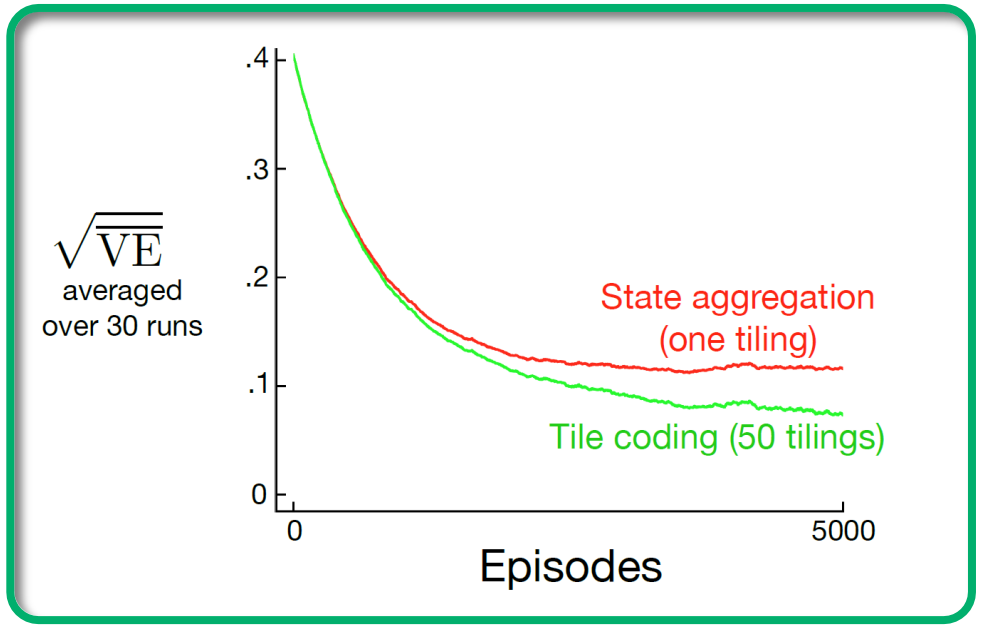
$\gamma = 1$

State Aggregation: 200 states

Tiling: 50 tiles

$$\alpha = \frac{0.0001}{50}$$



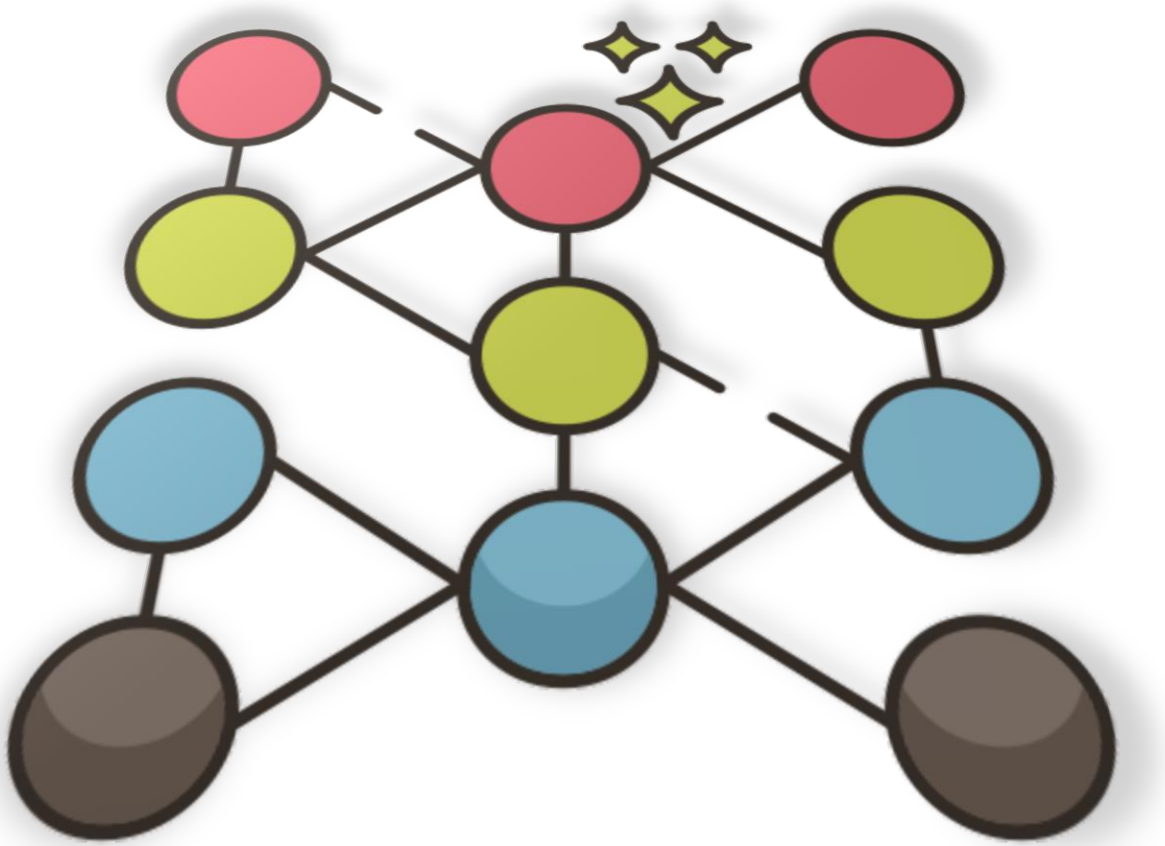


1000-state Random Walk

- Shown are learning curves on the 1000-state random walk example for the gradient Monte Carlo algorithm with a single tiling and with multiple tilings.

1. **How many tiles do we need to cover the full space of 1,000 states** (each tile corresponds to 200 states)
2. **Which method learns quicker? Why?**
3. **Which method has discriminates between states better?**

Artificial Neural Networks



Artificial Neural Networks

Introduction

ANN is a network of interconnected units that have some of the properties of neurons, the main components of nervous systems.

- The latest advances in training deeply-layered ANNs (deep learning) being responsible for some of the most impressive abilities of machine learning systems

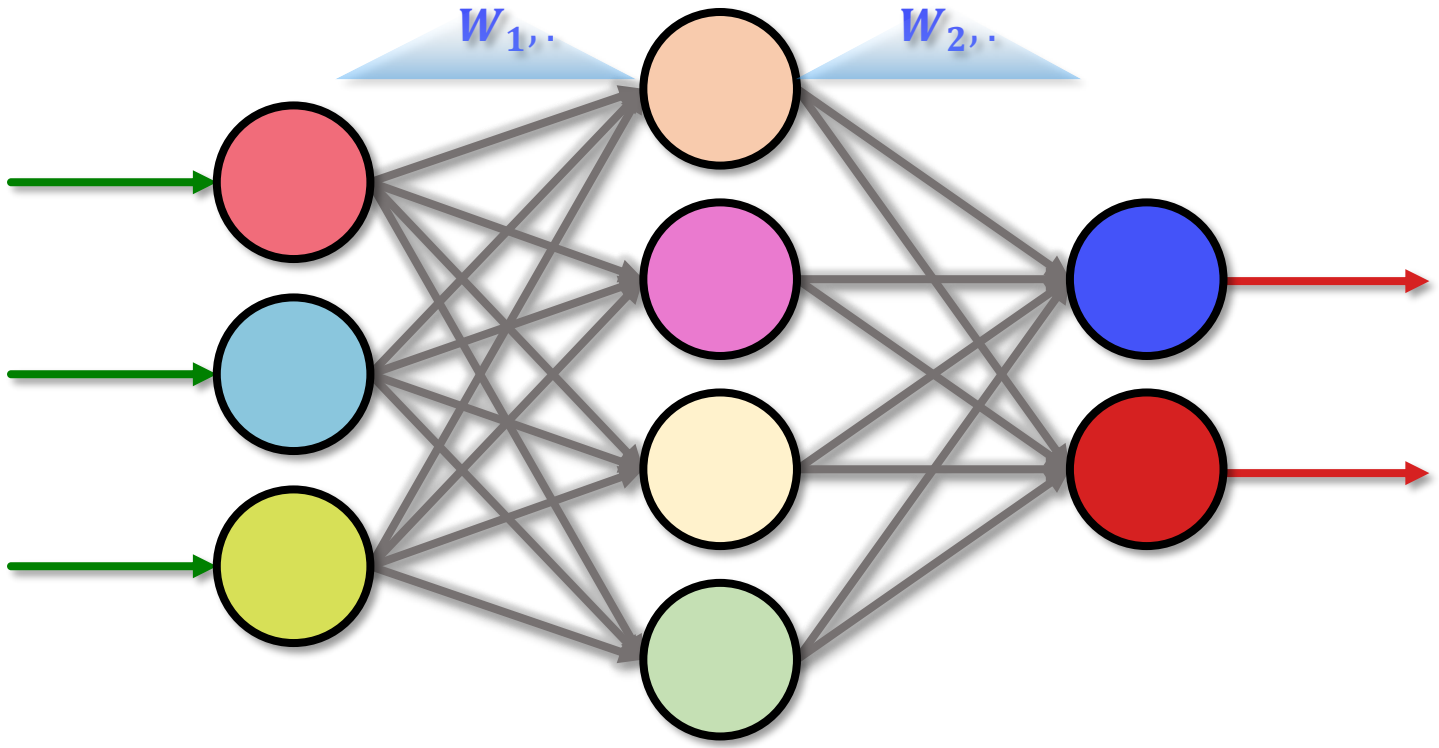
Deep learning techniques are one the best solutions to deal with high dimension data and extract discriminative information from the data.

- Deep learning algorithms have the capability of automating feature extraction (the extraction of representations) from the data.

In recent years, deep reinforcement learning has gained huge attraction from both academic communities and industrial sectors

Artificial Neural Networks

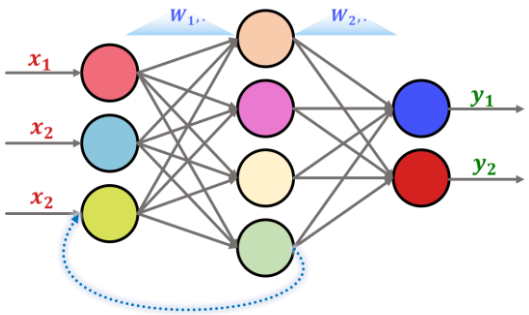
Simple Neural Network structure



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

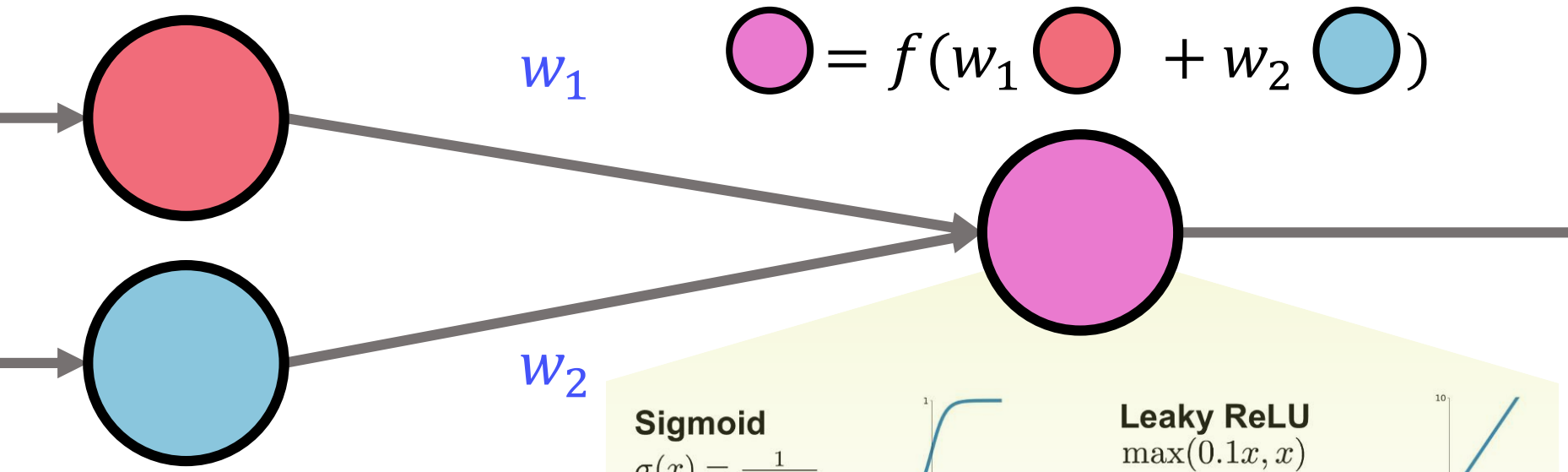
Feed forward neural network

Recurrent NN



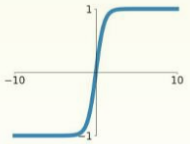
Artificial Neural Networks

Simple Neural Network structure

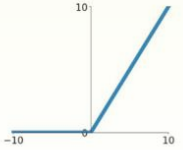


Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

tanh
$\tanh(x)$



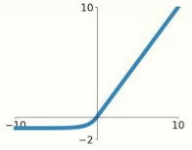
ReLU
$\max(0, x)$



Leaky ReLU
$\max(0.1x, x)$

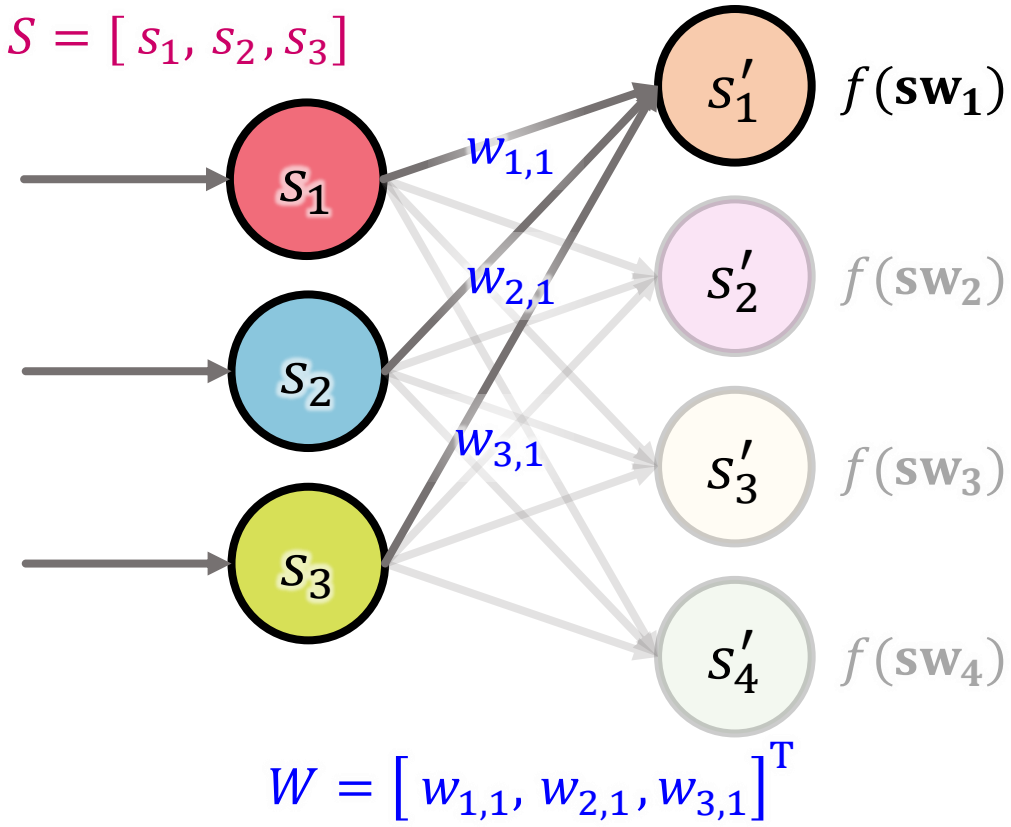
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

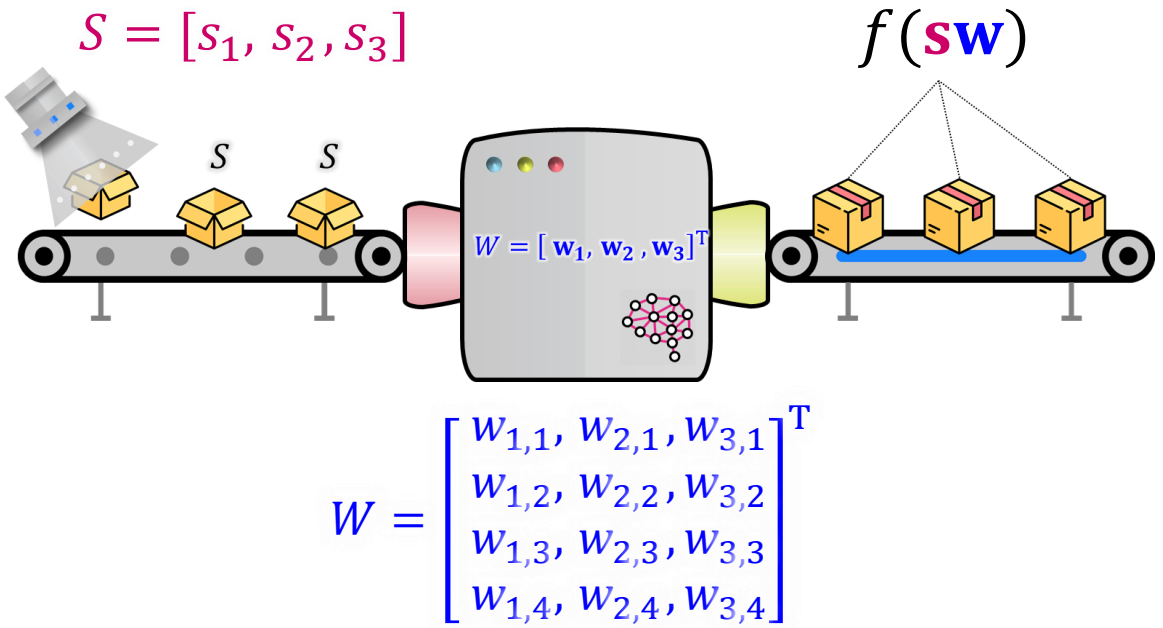
ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Artificial Neural Networks

Implementation

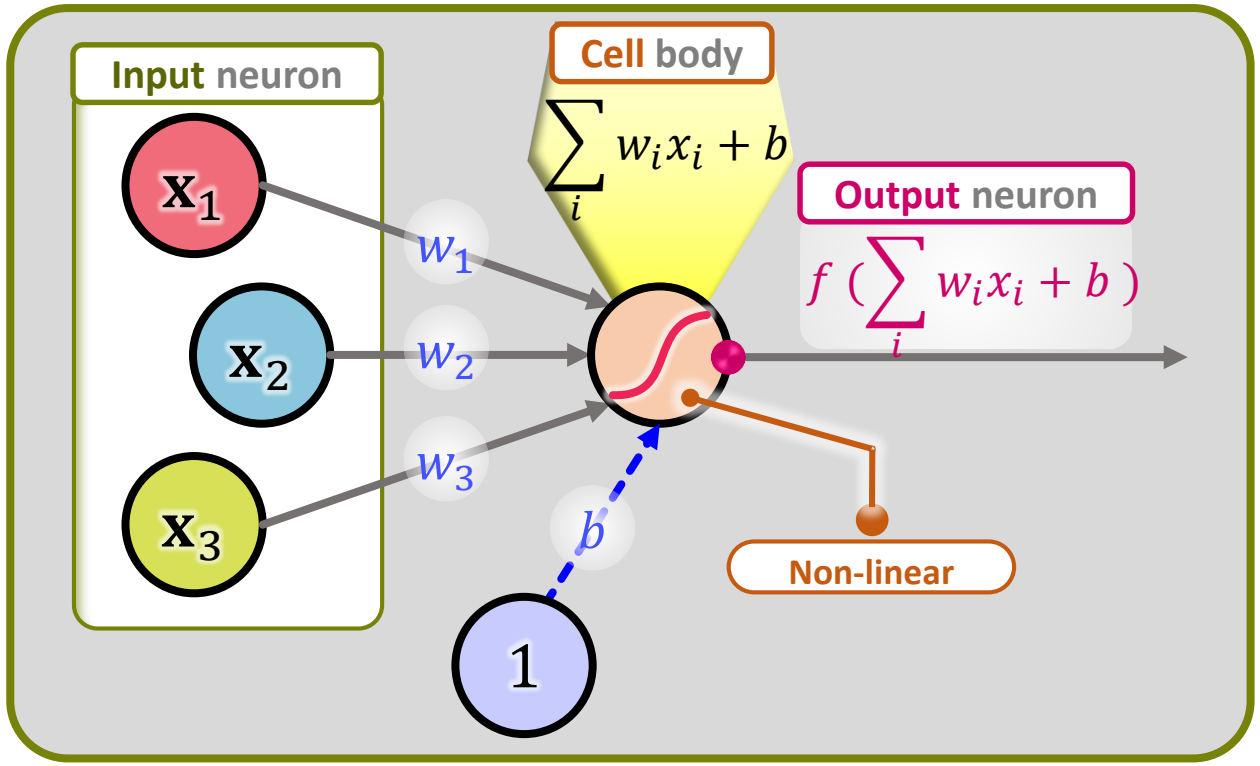




Artificial Neural Networks

Non-linear representation

- **The activation of each output unit of a feedforward ANN is a *nonlinear* function of the activation patterns over the network's input units.**
  - The functions are parameterized by the network's connection weights.

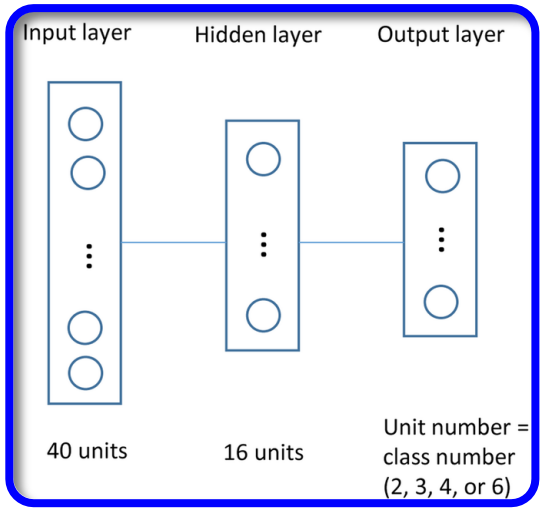


- **Why Non-Linear activation function**
  1. **Restrict values to a certain limit**
  2. **Add non-linearity into a neural network**

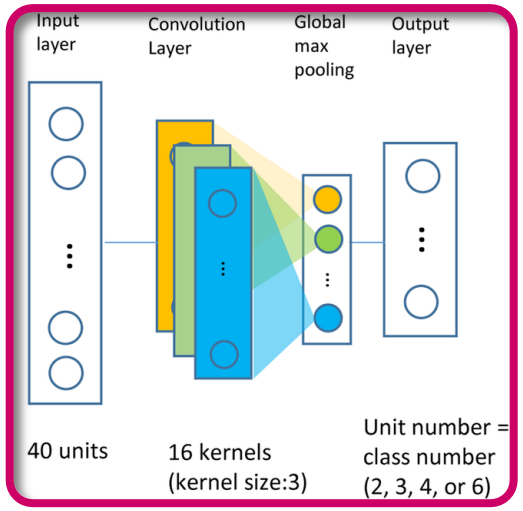
Artificial Neural Networks

The architecture of an ANN

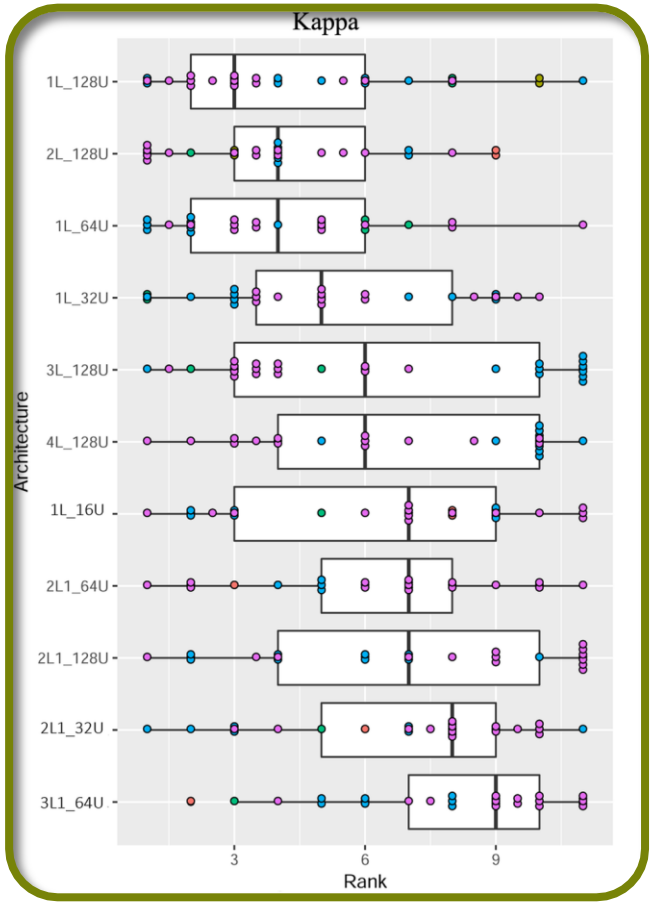
- **Successful implementation of deep learning lays in the careful design of the neural network architecture.**
- **A neural network's architecture can simply be defined as:**
  - **the number of layers (especially the hidden ones)**
  - **the number of hidden neurons within these layers.**



Multi-Layer Perceptrons (MLPs)



Convolutional Neural Network (CNN)

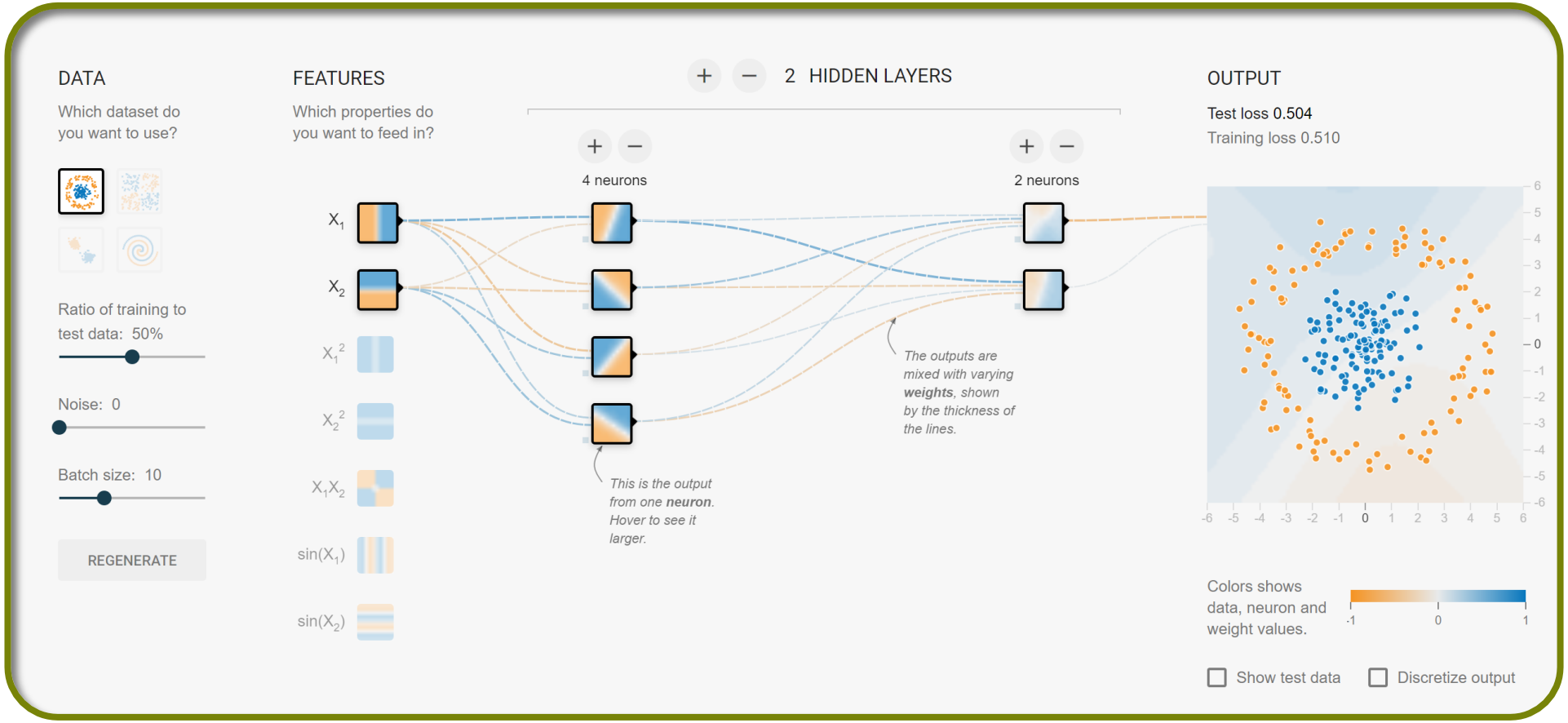


L: Layers
U: Neurons

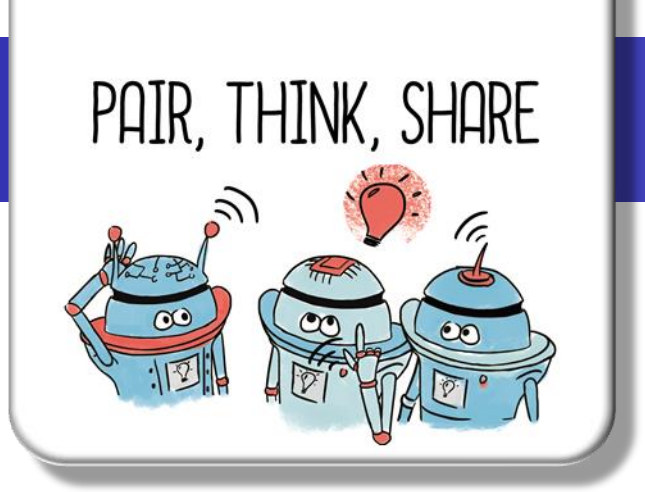
Yu, H., et. al. (2019). Architectures and accuracy of artificial neural network for disease classification from omics data. BMC genomics, 20(1), 167.

Artificial Neural Networks

TensorFlow playground



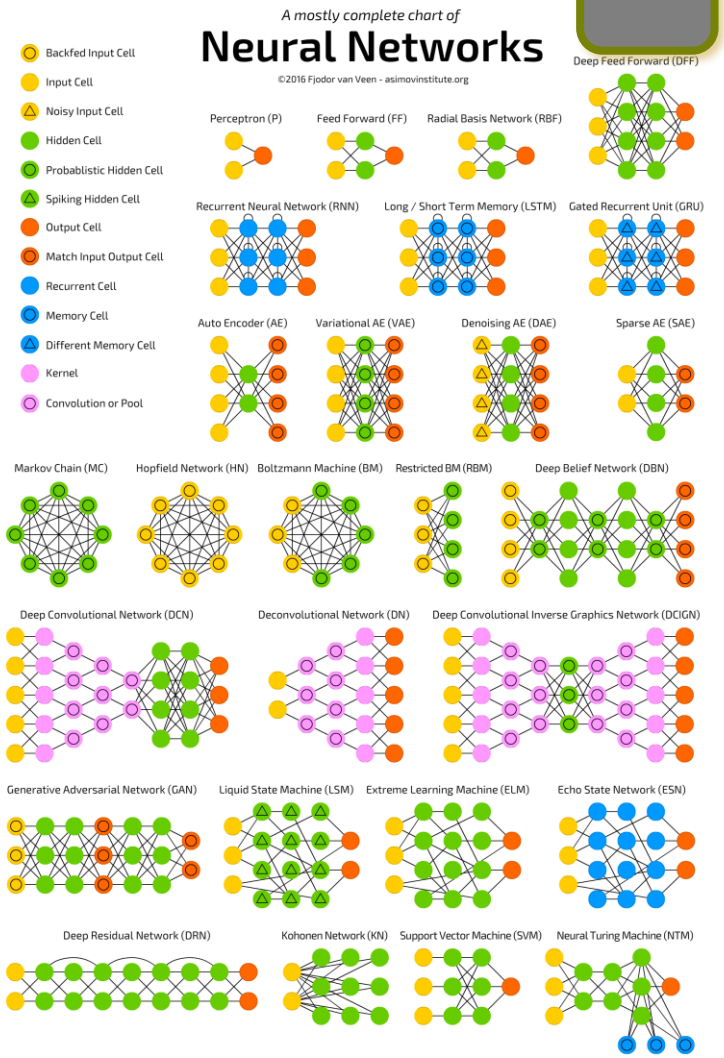
https://playground.tensorflow.org/



Neural Networks Architecture

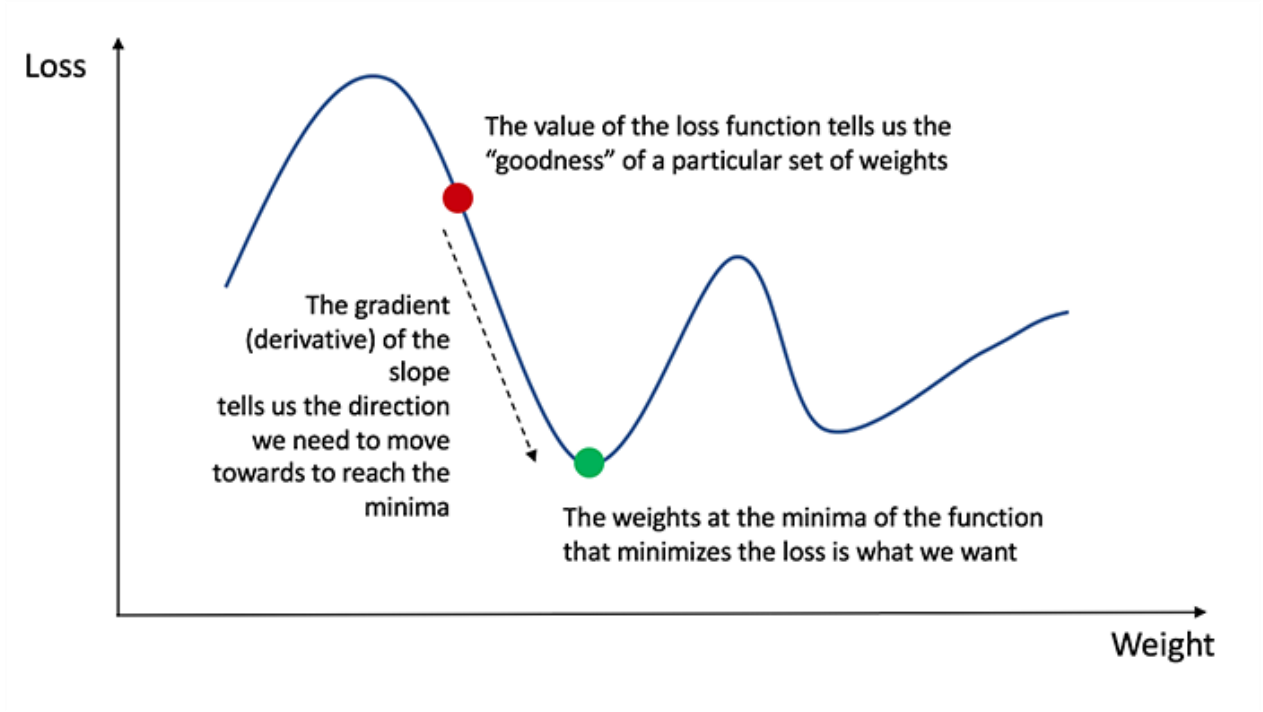
- The information processing capability of artificial neural networks (ANNs) is closely related to its ***architecture*** and ***weights***.

1. **What happens if we remove the hidden layer from the model?**
2. **Can an ANN with a single hidden layer approximate any continuous function**
3. **What happens if we select linear activation furcation rather than non-liner functions**



Loss function

- **In the most common supervised learning case, the objective function is the expected error, or loss, over a set of labeled training examples**

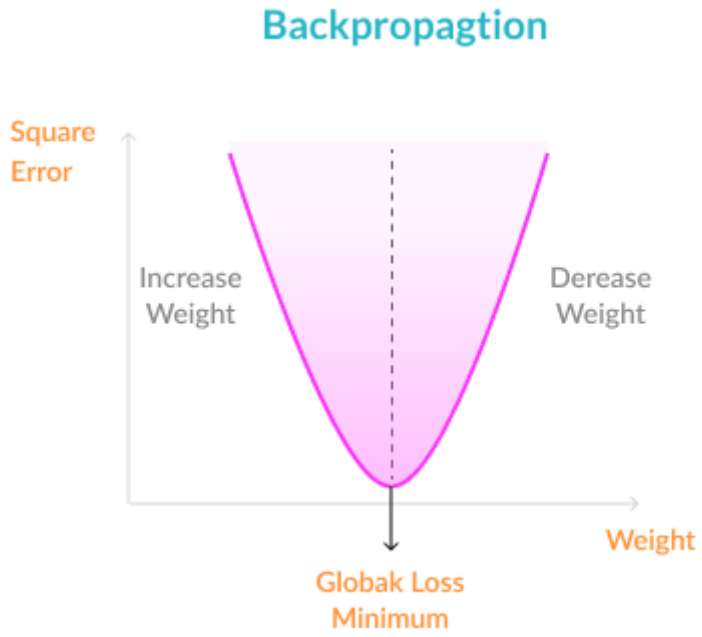


- In reinforcement learning, ANNs can use TD errors to learn value functions, or they can aim to maximize expected reward as in a gradient bandit

Artificial Neural Networks

Gradient Descent for Training

- **The most successful way to minimize loss for ANNs with hidden layers is by using gradient of differentiable activation functions**
- **This is called the *backpropagation* algorithm which consists of alternating forward and backward passes through the network.**

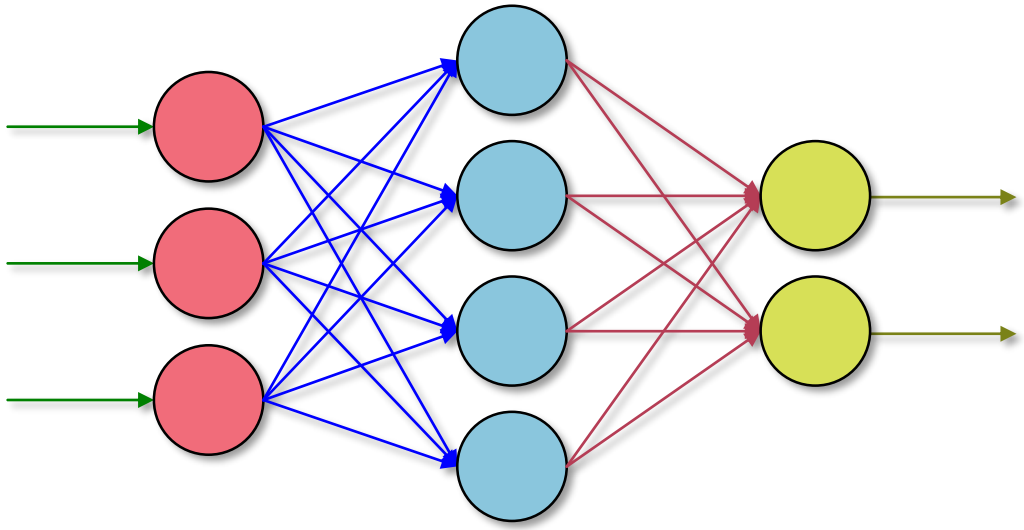


Update rule

$$w \leftarrow w - \alpha \frac{\partial L}{\partial w}$$

Artificial Neural Networks

Backpropagation



s A x B $\hat{y}$

Loss function

$$L(\hat{y}_k, y_k) = (\hat{y}_k - y_k)^2$$

$$A = A - \alpha\delta^A s$$

$$B = B - \alpha\delta^B s$$

Artificial Neural Networks

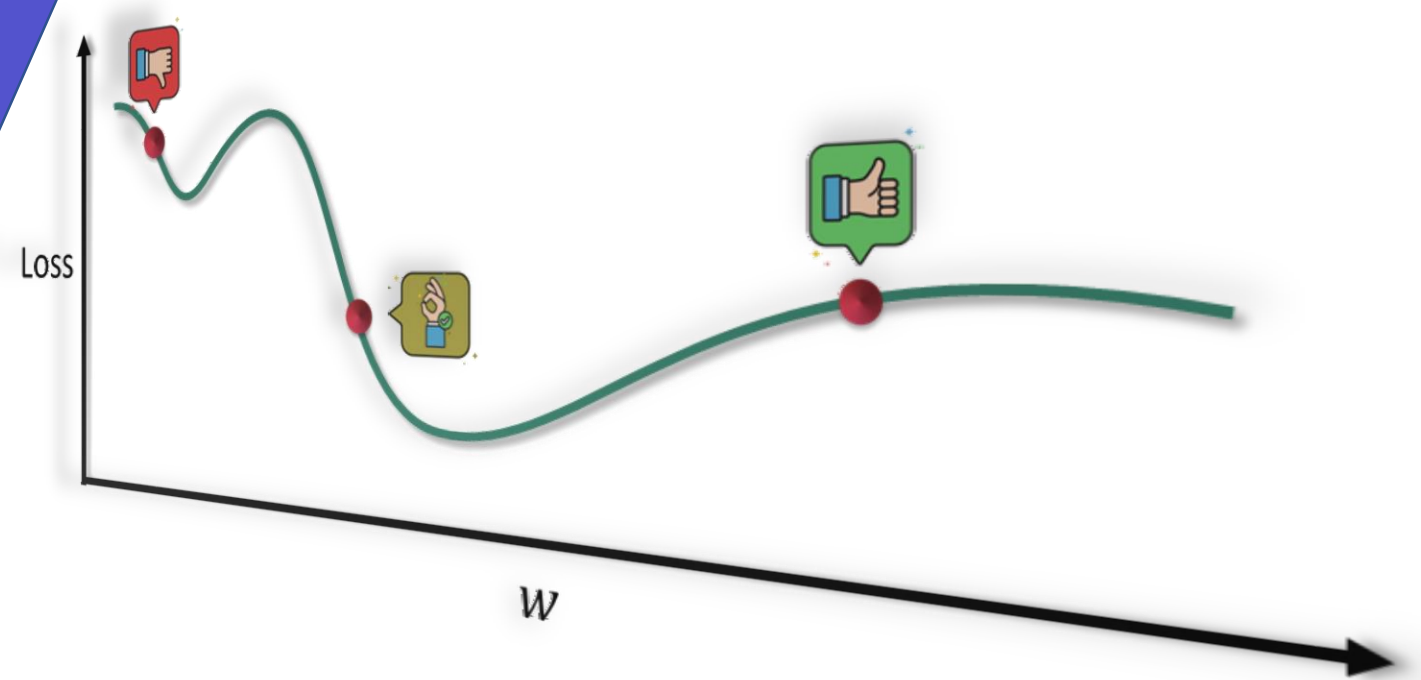
Extra learning resources

- **Introduction to Neural Networks (link)**
  - A great graphical representation on Neural Network with a detailed explanation of structure of layer and their connections together

- **TensorFlow Playground (link)**
  - This is an interactive visualization of neural networks. It contains a tiny neural network library that meets the demands of this educational visualization. You can simulate, in real time, in your browser, small neural networks and see the results.

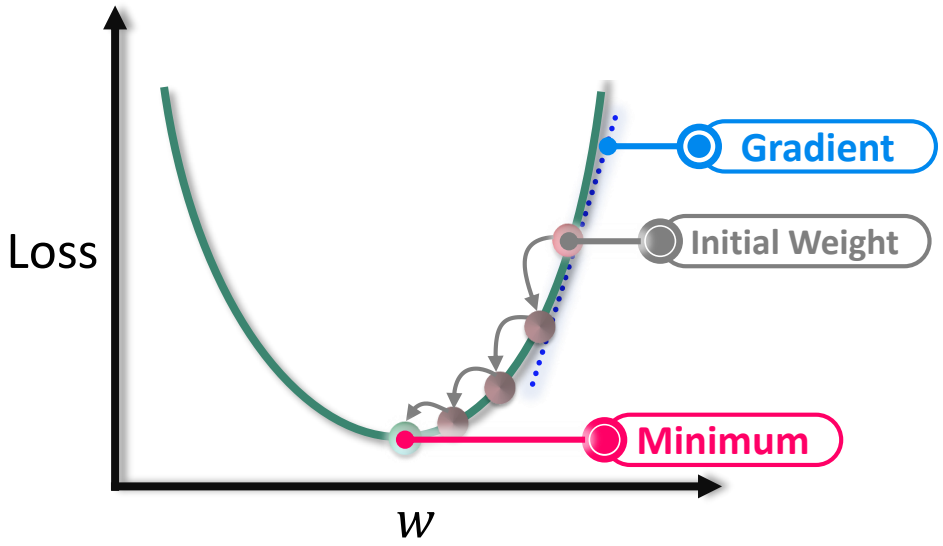
- **Gradient Decent for Neural Networks (link)**
  - This video shows a proper walk-through backpropagation for Neural Networks with accurate explanations and cool animations.

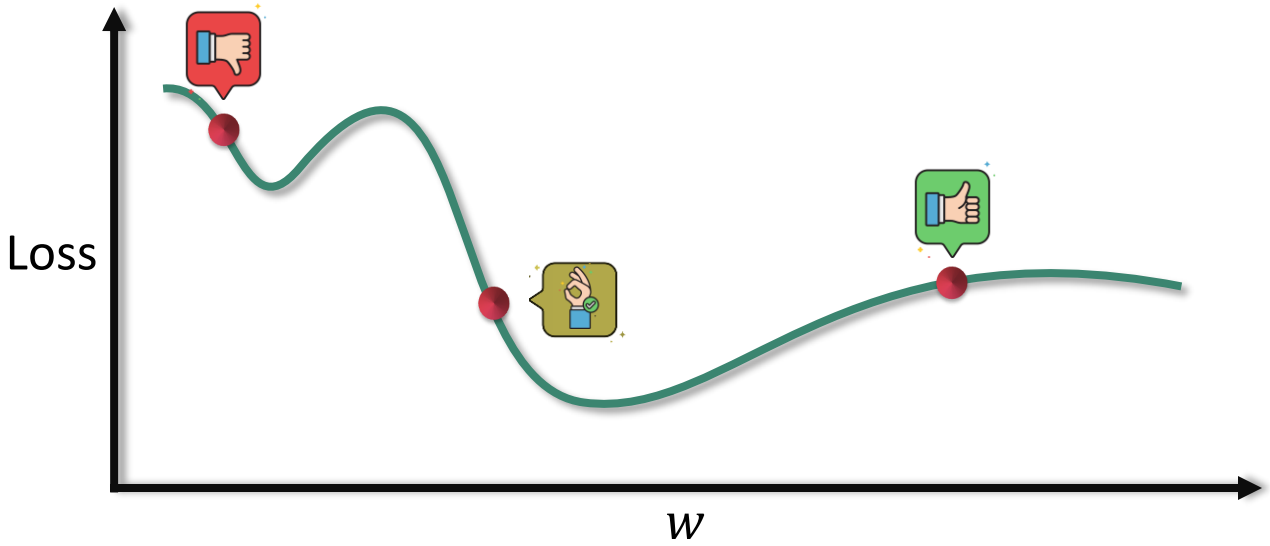
Optimization Strategies for NNs

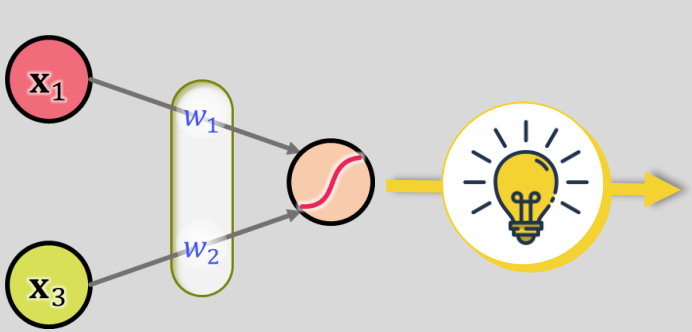


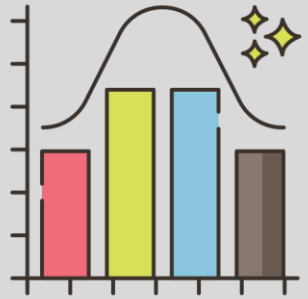
Optimization Strategies for NNs

Effective initialization strategy

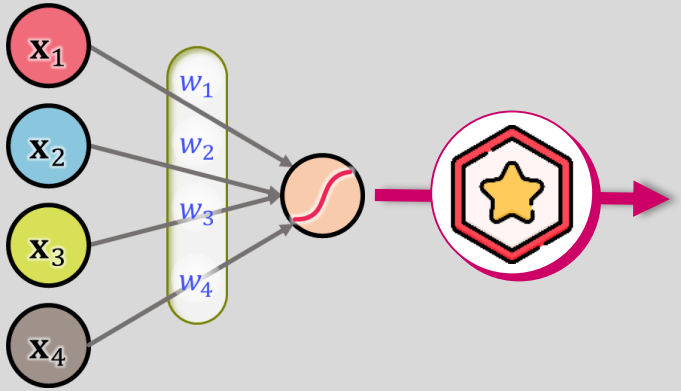






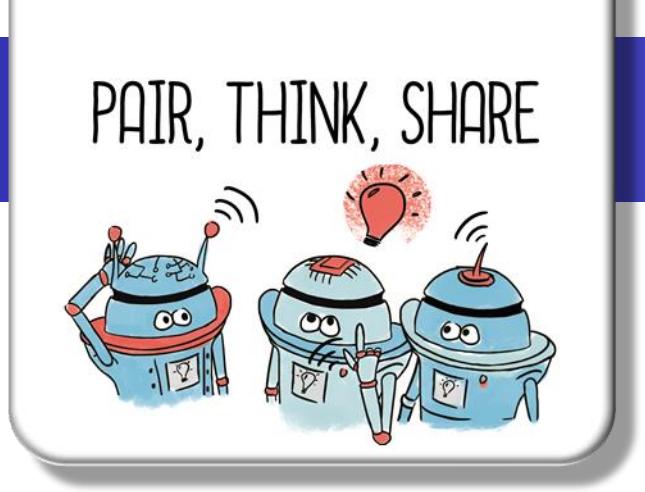


$w \sim \mathcal{N}(0,1)$



Weight initialization

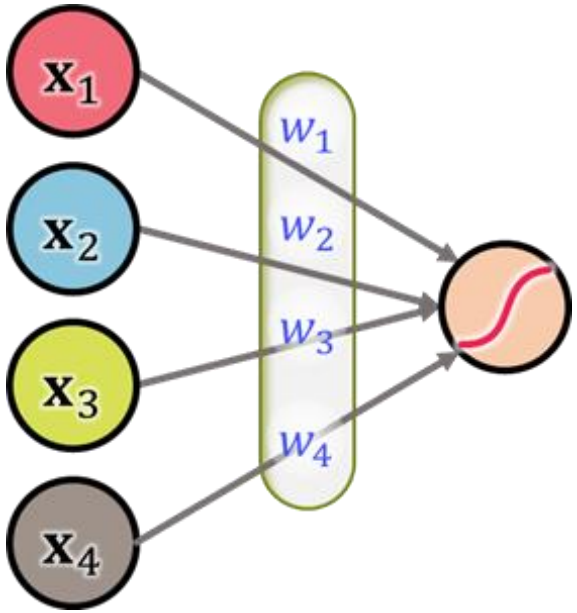
$$W_{init} \sim \frac{\mathcal{N}(0,1)}{\sqrt{n_{inputs}}}$$



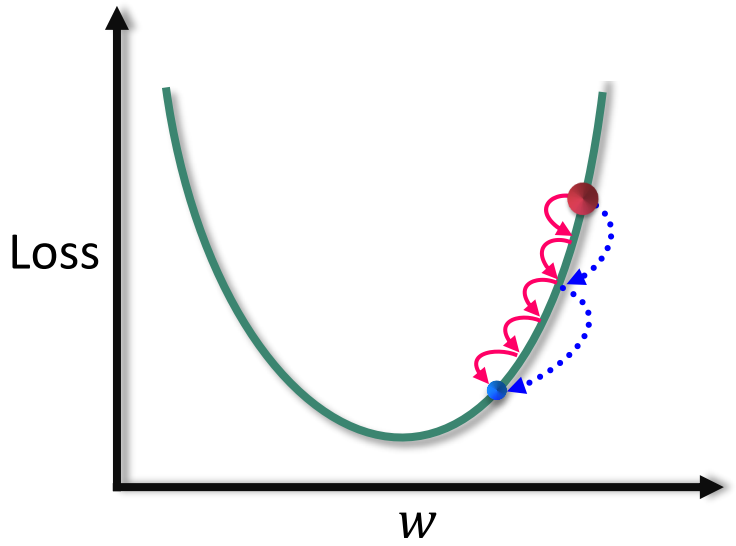
Weight Initialization

- One of the starting points to take care of while building your network is to initialize your weight matrix correctly..

What happens if initialize all weights to zero?



- **SGD with momentum is method which helps accelerate gradients vectors in the right directions, thus leading to faster converging.**
  - **Momentum helps accelerate gradients in the right direction.**



SGD update rule

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} \boldsymbol{L}(\mathbf{w}_t)$$

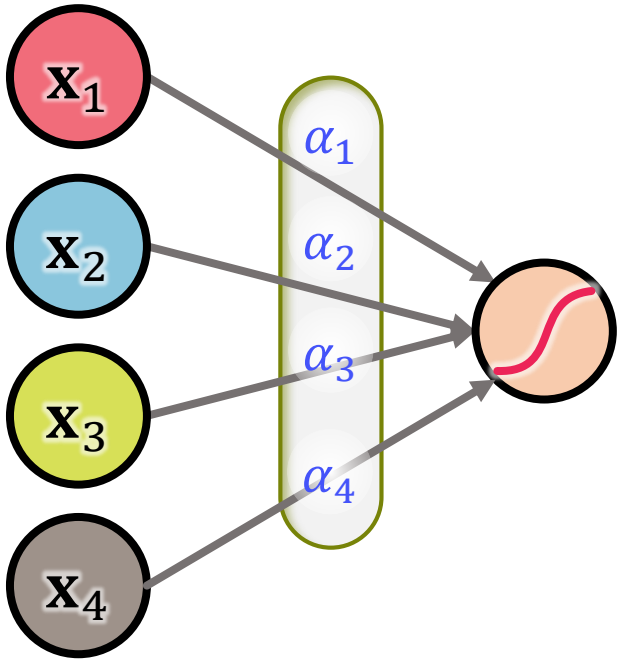
Momentum update rule

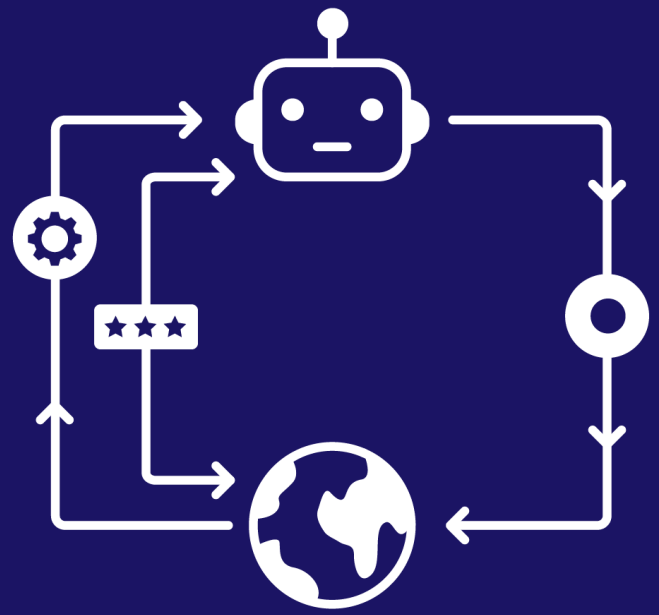
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} \boldsymbol{L}(\mathbf{w}_t) + \lambda \mathbf{M}_t$$

$$\mathbf{M}_{t+1} \leftarrow \lambda \mathbf{M}_t - \alpha \nabla_{\mathbf{w}} \boldsymbol{L}$$

Vector step size adaptation

- **A feature with large or small step sizes will have a large or small impact on generalization from training examples.**
  - **A feature with large or small step sizes will have a large or small impact on generalization from training examples**





Summary