UiO : University of Oslo

institutt for informatikk

IN3050/IN4050 - Introduction to Artificial Intelligence and Machine Learning

Trial Exam – Suggested Solutions

Jan Tore Lønning

Kai Olav Ellefsen

Simulated Annealing (6p)

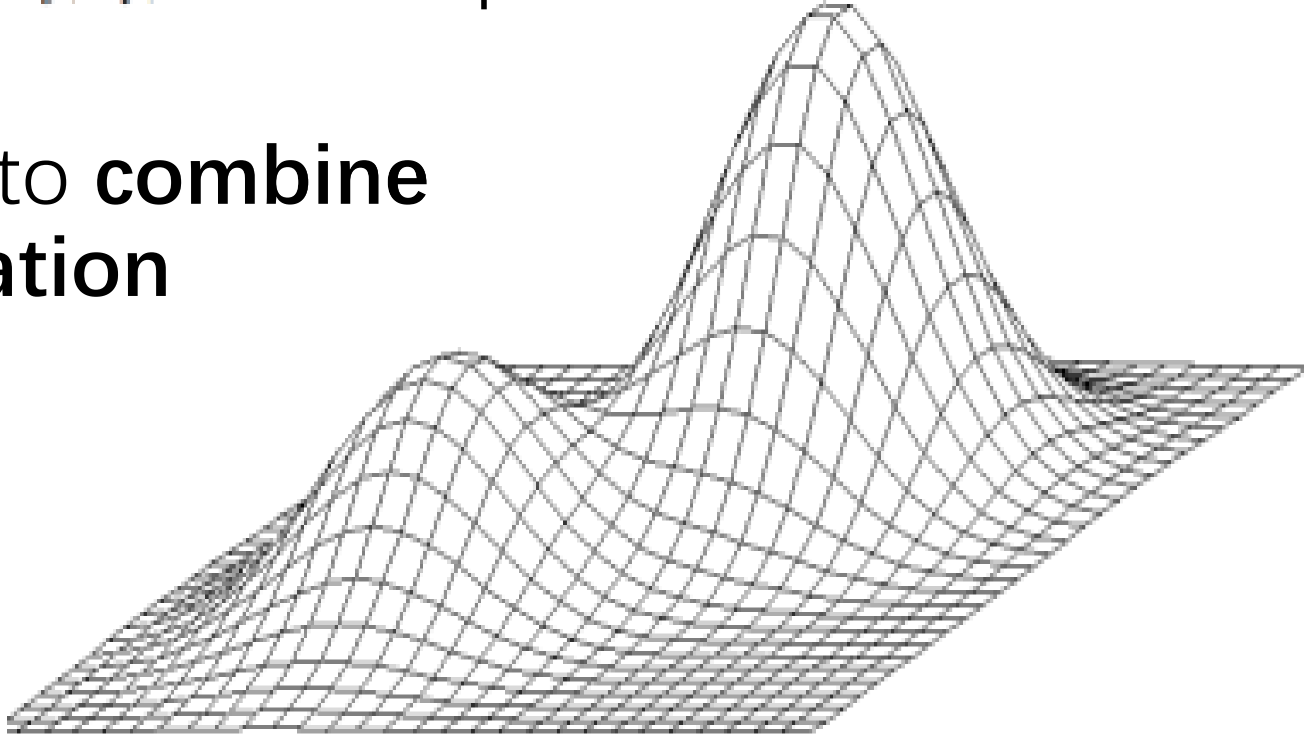
In simulated annealing,

a) What would happen if we start with a very low temperature (keeping low through search)? Which search algorithm would this be similar to? (3p)

b) What would happen if we start with a very high temperature and never decrease? (3p)

Global optimization

- Most of the time, we must expect the problem to have **many local optima**
- Ideally, we want to find the best local optimum: **the global optimum**
- The best strategy is often to **combine exploration and exploitation**



Simulated annealing

- Set an initial temperature T
- Pick an initial solution
- Repeat:
  - Pick a solution neighboring the current solution
  - If the new one is better, keep it
  - Otherwise, keep the new one with probability *p*
    - *p* depends on the **difference in quality** and the **temperature.** high temp -> high *p* *(more randomness)*
  - Reduce T

Simulated Annealing (6p)

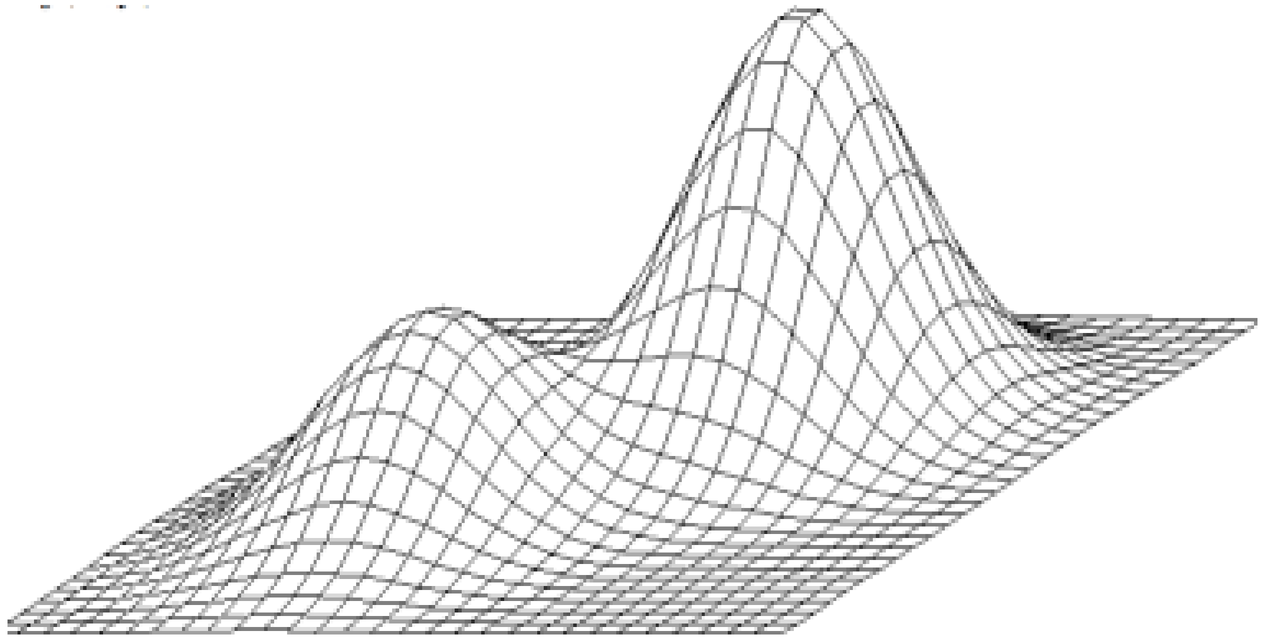
In simulated annealing,

a) What would happen if we start with a very low temperature (keeping low through search)? Which search algorithm would this be similar to? (3p)

b) What would happen if we start with a very high temperature and never decrease? (3p)

Hill climbing

- Pick a solution as the current best (e.g. a random solution)
- Compare to neighbor solution(s)
  - If the neighbor is better, replace the current best
  - Repeat until we reach a certain number of evaluations



Simulated Annealing (6p)

In simulated annealing,

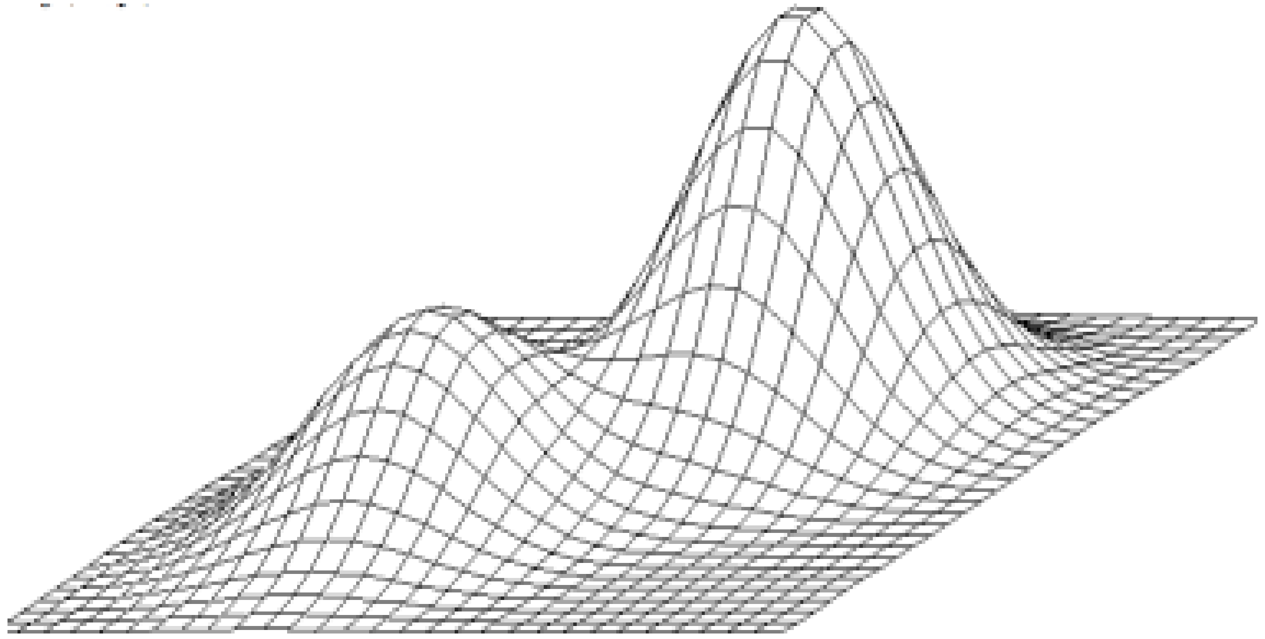
a) What would happen if we start with a very low temperature (keeping low through search)? Which search algorithm would this be similar to? (3p)

b) What would happen if we start with a very high temperature and never decrease? (3p)

Exhaustive search

Only works for simple discrete problems, but can be approximated in continuous problems

- Sample the space at regular intervals (grid search)
- Sample the space randomly N times

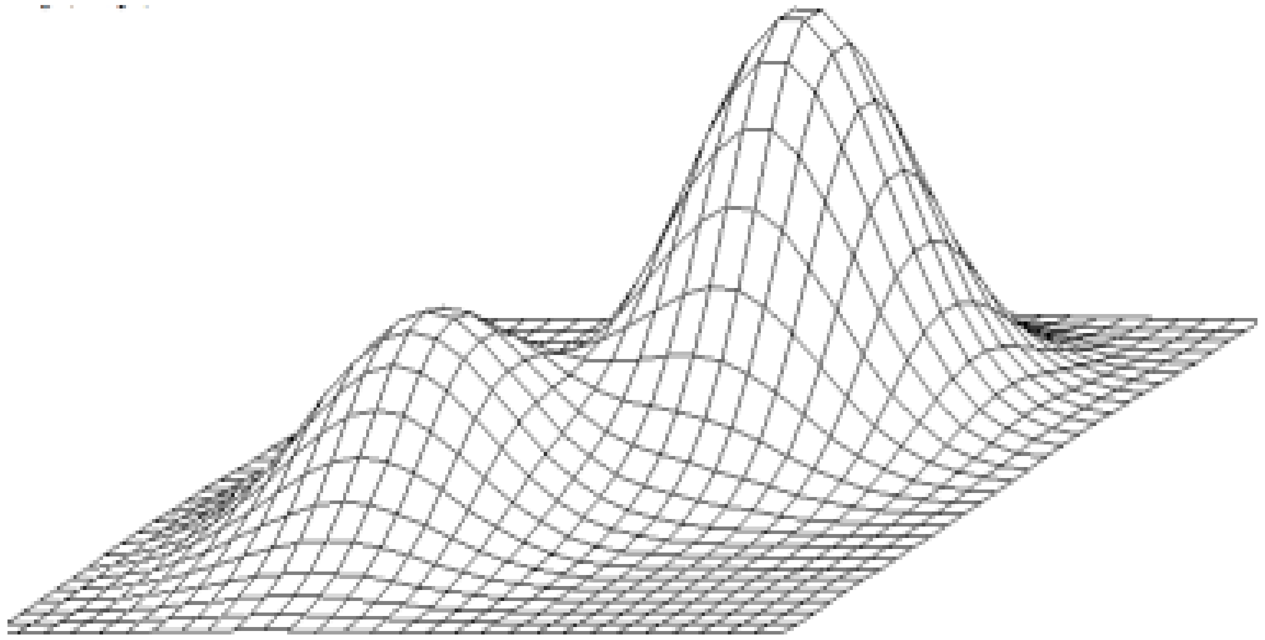


Master Students Only: Search (5p)

In a few sentences, sketch how you could modify a hill climbing algorithm in order to improve chances of finding the global optimum.

Hill climbing

- Pick a solution as the current best (e.g. a random solution)
- Compare to neighbor solution(s)
  - If the neighbor is better, replace the current best
  - Repeat until we reach a certain number of evaluations



Master Students Only: Search (5p)

In a few sentences, sketch how you could modify a hill climbing algorithm in order to improve chances of finding the global optimum.

Suggested Solutions:

- Run it multiple times with random starting positions
- Sometimes randomly choose worse solutions to not get stuck in local optima (like Simulated Annealing does)

Bachelor students only: EA Selection (5p)

Five strings have the following fitness values: 3, 6, 9, 12, 15. Under Fitness Proportionate Selection, compute the expected number of copies of each string in the mating pool if a constant population size, n = 5, is maintained.

Bachelor students only: EA Selection (5p)

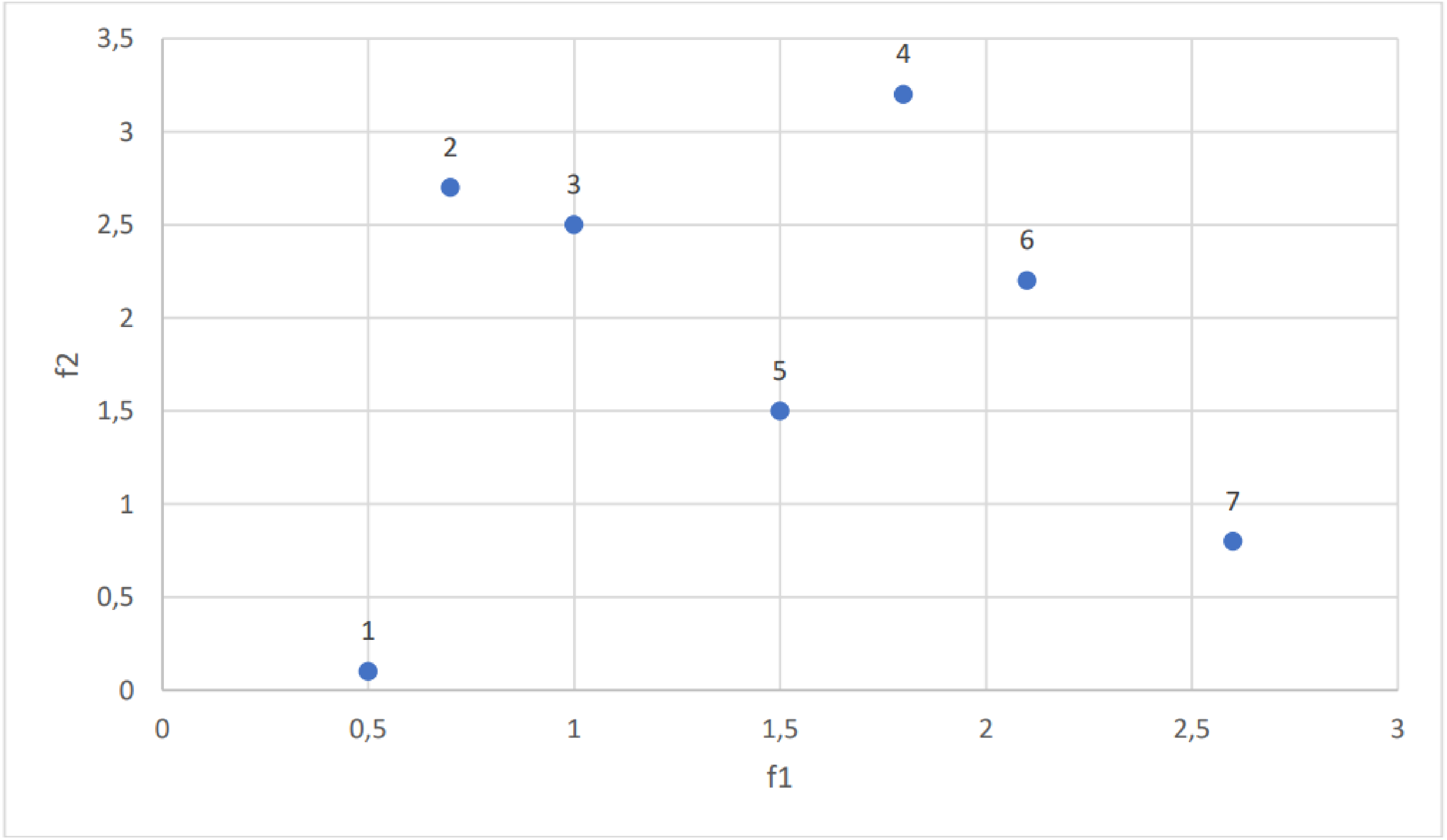
- Fitness values: 3, 6, 9, 12, 15
- Fitness Proportionate Selection implies that each of these 5 solutions should have **a chance of being picked equal to its proportion of the total fitness** in the population.
- Total population fitness: 3+6+9+12+15 = 45
- Expected number of copies:
  - Individual 1 (fitness = 3): Probability of getting chosen once: 3/45. Multiplying probability with 5 draws: (3/45)*5 = 0.33 copies are expected (0 if rounding off)

Bachelor students only: EA Selection (5p)

- Individual 1 (fitness = 3): Probability of getting chosen once: 3/45. Multiplying probability with 5 draws: (3/45)*5 = 0.33 copies are expected (0 if rounding off)
- Individual 2 (fitness = 6): Probability of getting chosen once: 6/45. Multiplying probability with 5 draws: (6/45)*5 = 0.66 copies are expected (1 if rounding off)
- Individual 3 (fitness = 9): Probability of getting chosen once: 9/45. Multiplying probability with 5 draws: (9/45)*5 = 1 copy is expected.
- Individual 4 (fitness = 12): Probability of getting chosen once: 12/45 Multiplying probability with 5 draws: (12/45)*5 = 1.33 copies are expected (1 if rounding off)
- Individual 5 (fitness = 15): Probability of getting chosen once: 15/45 Multiplying probability with 5 draws: (15/45)*5 = 1.66 copies are expected (2 if rounding off)

Pareto Optimality (9p)

- For an optimization problem we wish to optimize solutions according to two different objectives, f1 and f2.

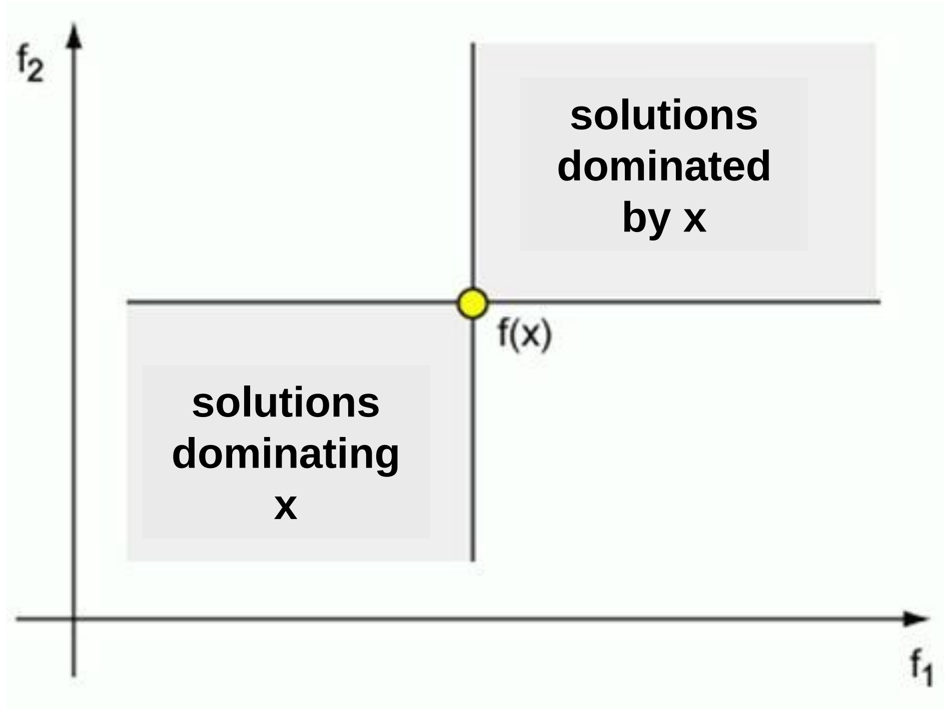


Pareto Optimality (9p)

What requirements do the solutions in a Pareto optimal set need to fulfill? (3p)

Dominance relation

- Solution x dominates solution y, (x $\preceq$ y), if:
    - x is better than y in at least one objective,
    - x is not worse than y in all other objectives



Pareto optimality

- Solution x is **non-dominated** among a set of solutions Q if no solution from Q dominates x

- A set of non-dominated solutions from the entire feasible solution space is the **Pareto set**, or **Pareto front**, its members Pareto-optimal solutions

Pareto Optimality (9p)

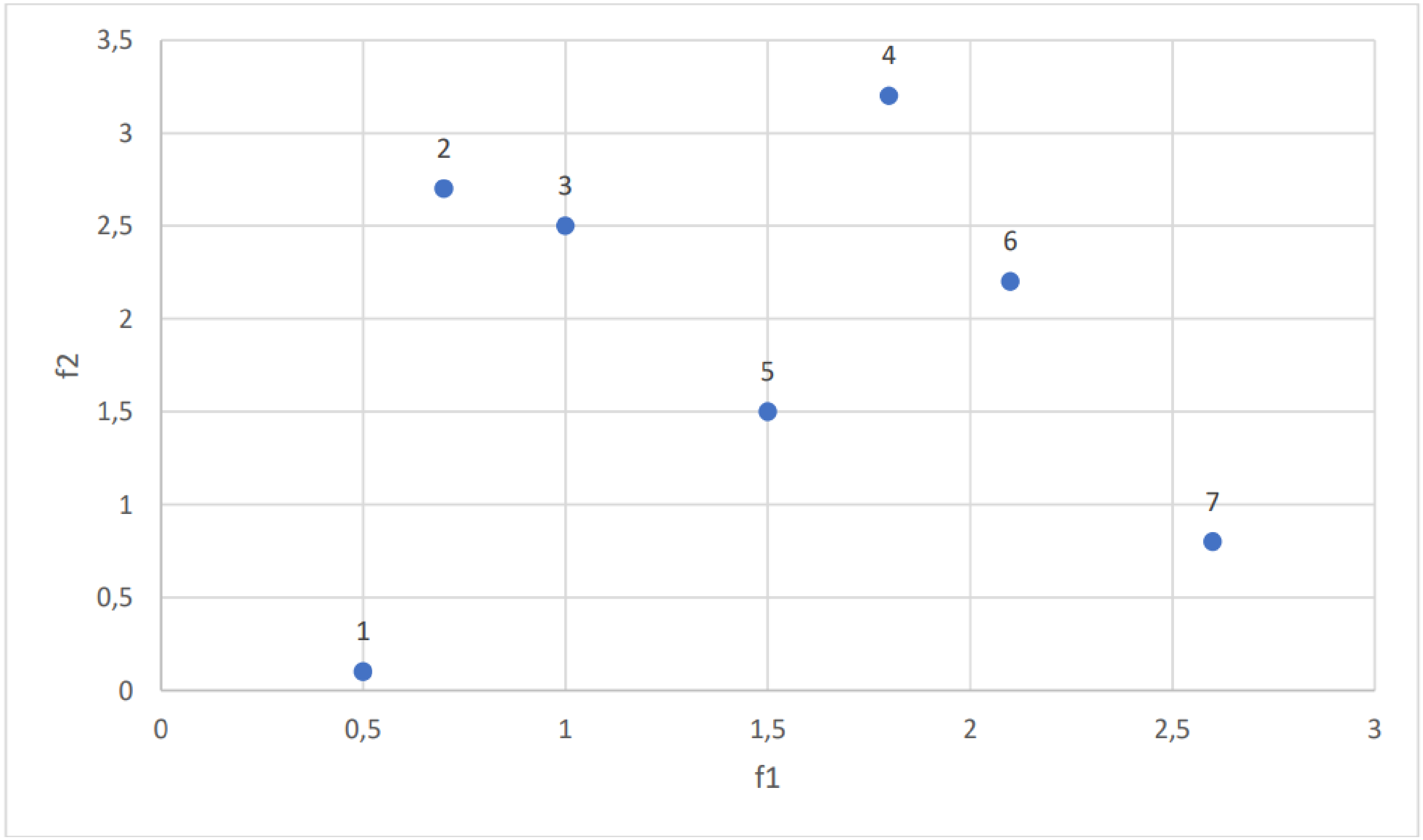
Find the Pareto optimal set of solutions when

b) Maximizing f1 and f2 (3p)

4, 6 and 7

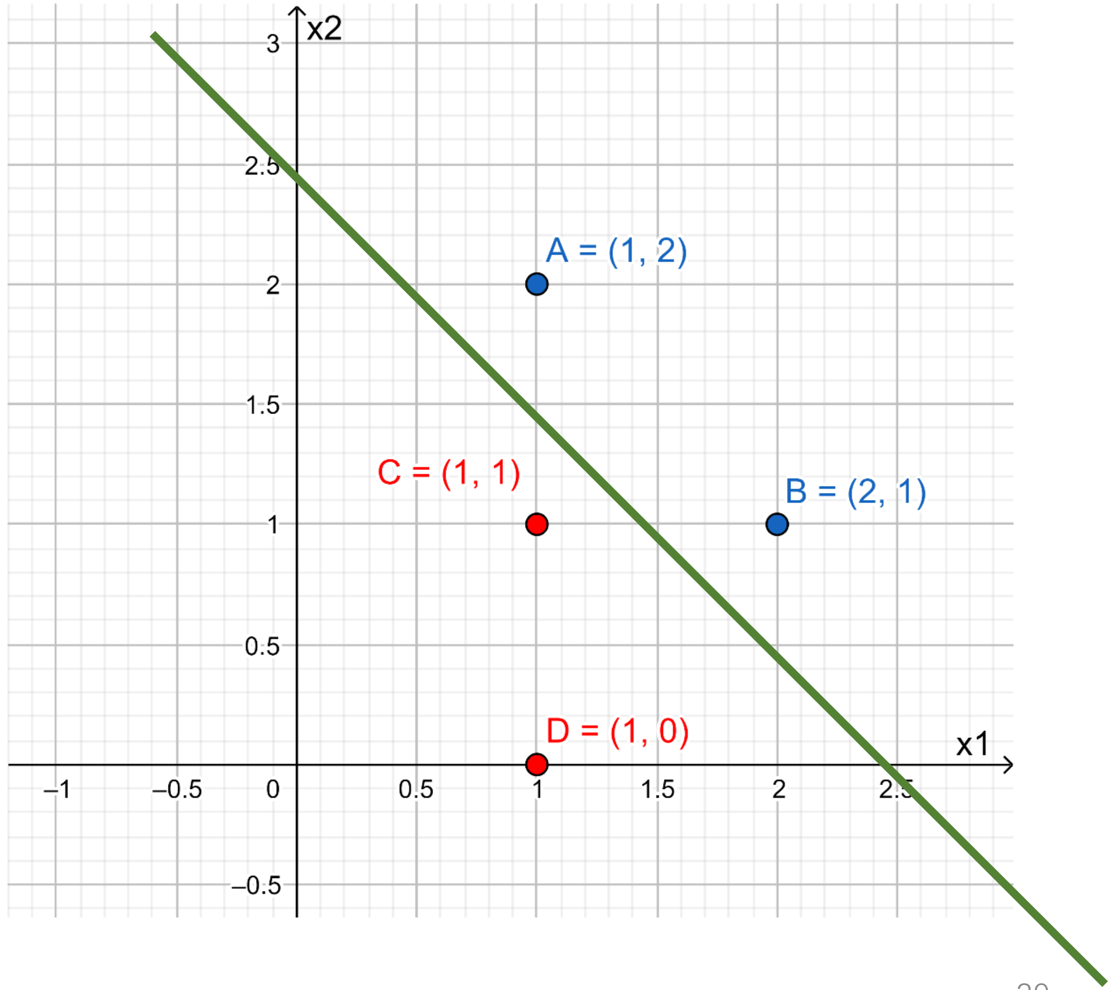
c) Maximizing f1 but minimizing f2 (3p)

1, 7



a) Linearly separable?

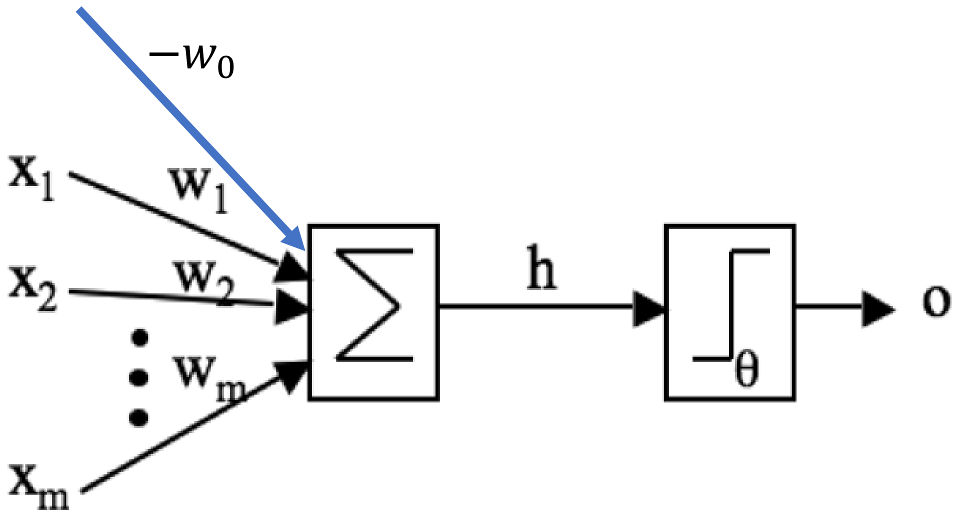
- YES
- Either show figure, or
- Provide an equation
  - e.g., $x_1 + x_2 = 2.5$



Training one perceptron

- Inputs: $x_0, x_1, \dots x_m$
- Label: t, which is 1 or 0
- Calculate the output of the perceptron
  - $h = \sum_{i=1}^{m} w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$

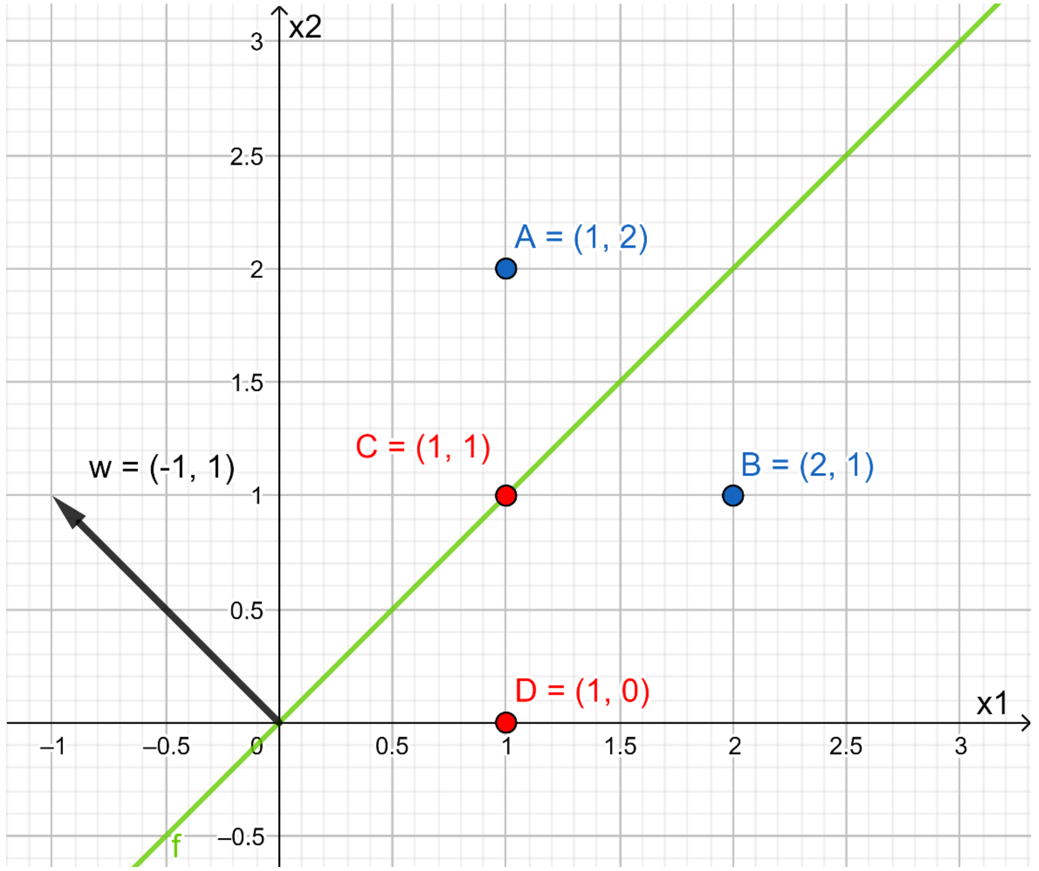
$$y = o = g(h) = \begin{cases} 1 \; if \; h > \theta \\ 0 \; if \; h \le \theta \end{cases}$$



- (With bias $\theta = 0$)
- If $y = t$, do nothing, if $y \neq t$, update weights

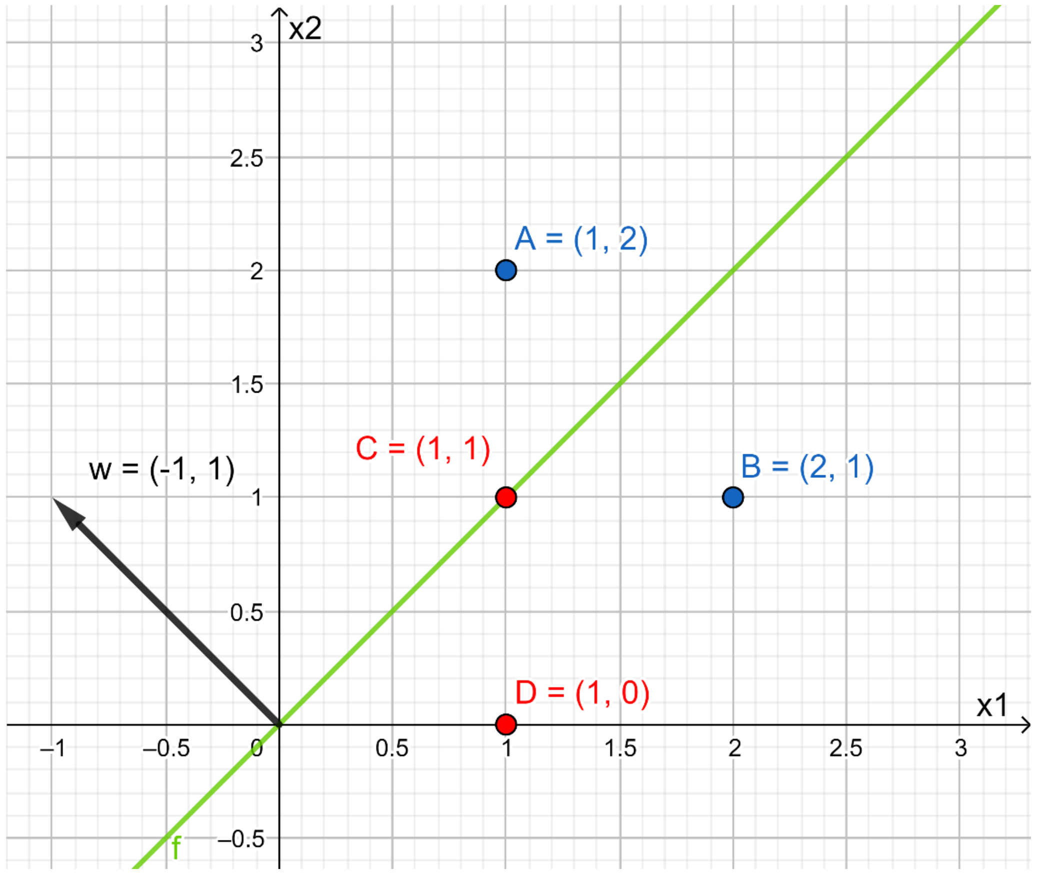
b) Update perceptron point A

- $t = 1$
- $z = \boldsymbol{w} \cdot \boldsymbol{x} =$
  $(0, -1,1) \cdot (-1,1,2) = 1$
- $y = 1$, because $z > 0$
- Since $y = t$, do nothing!

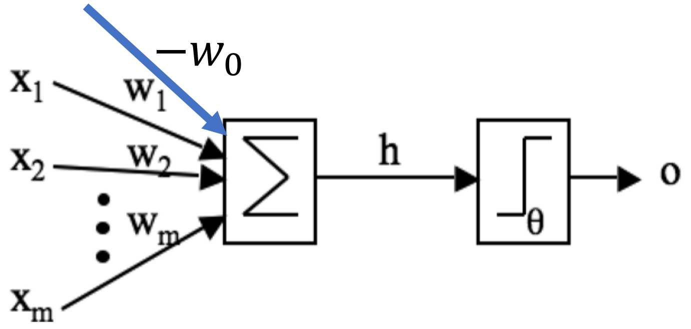


b) Update perceptron point B

- $t = 1$
- $z = \boldsymbol{w} \cdot \boldsymbol{x} = (0, -1, 1) \cdot (-1, 2, 1) = -1$
- $y = 0$, because $z \leq 0$



Update weights



$\eta > 0$ is the fixed learning rate

if $t = 1, y = 0$

- increase $\sum_{i=0}^{m} w_i x_i$
  - by increasing each $w_i x_i$:
    - if $x_i > 0$: increase w_i
    - if $x_i < 0$: decrease w_i
  - $w_i = w_i + \eta x_i$
    - cover both cases

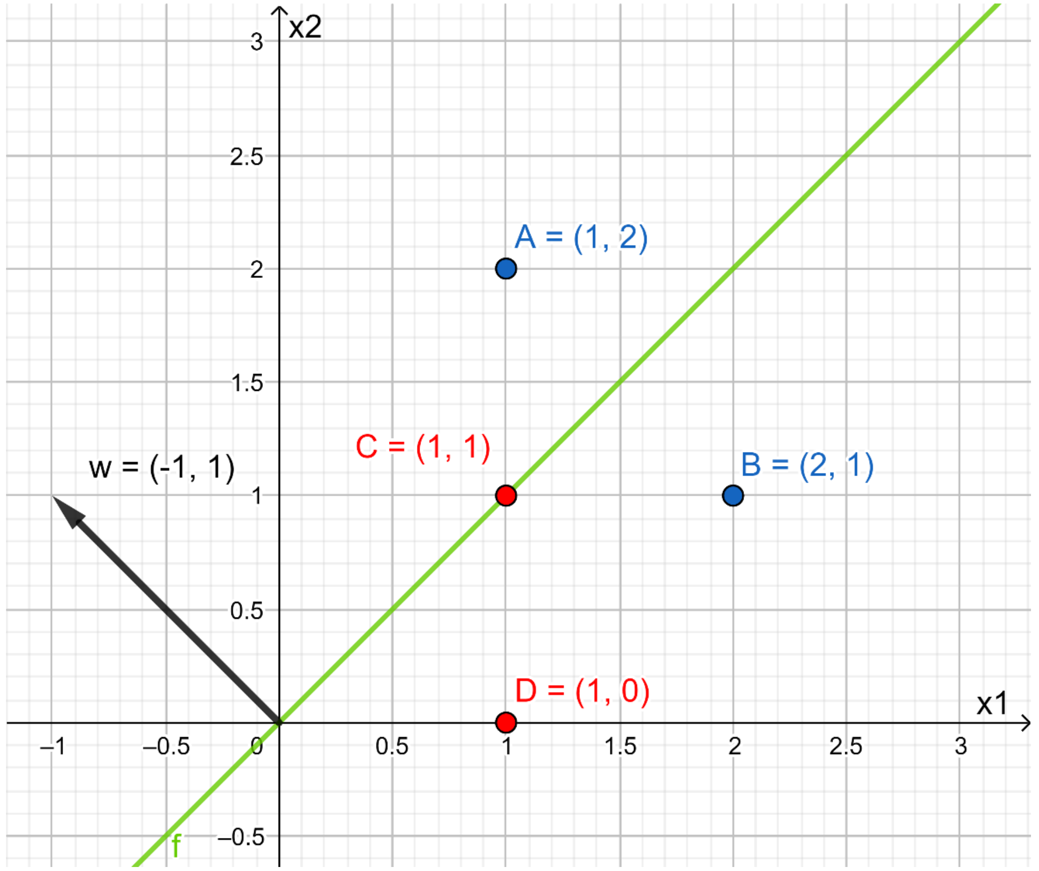
if $t = 0, y = 1$

- decrease $\sum_{i=0}^{m} w_i x_i$
  - by decreasing each $w_i x_i$:
    - if $x_i > 0$: decrease w_i
    - if $x_i < 0$: increase w_i
  - $w_i = w_i - \eta x_i$

$w_i = w_i + \eta(t - y)x_i \;=\; w_i - \eta(y - t)x_i$
(covers all cases)

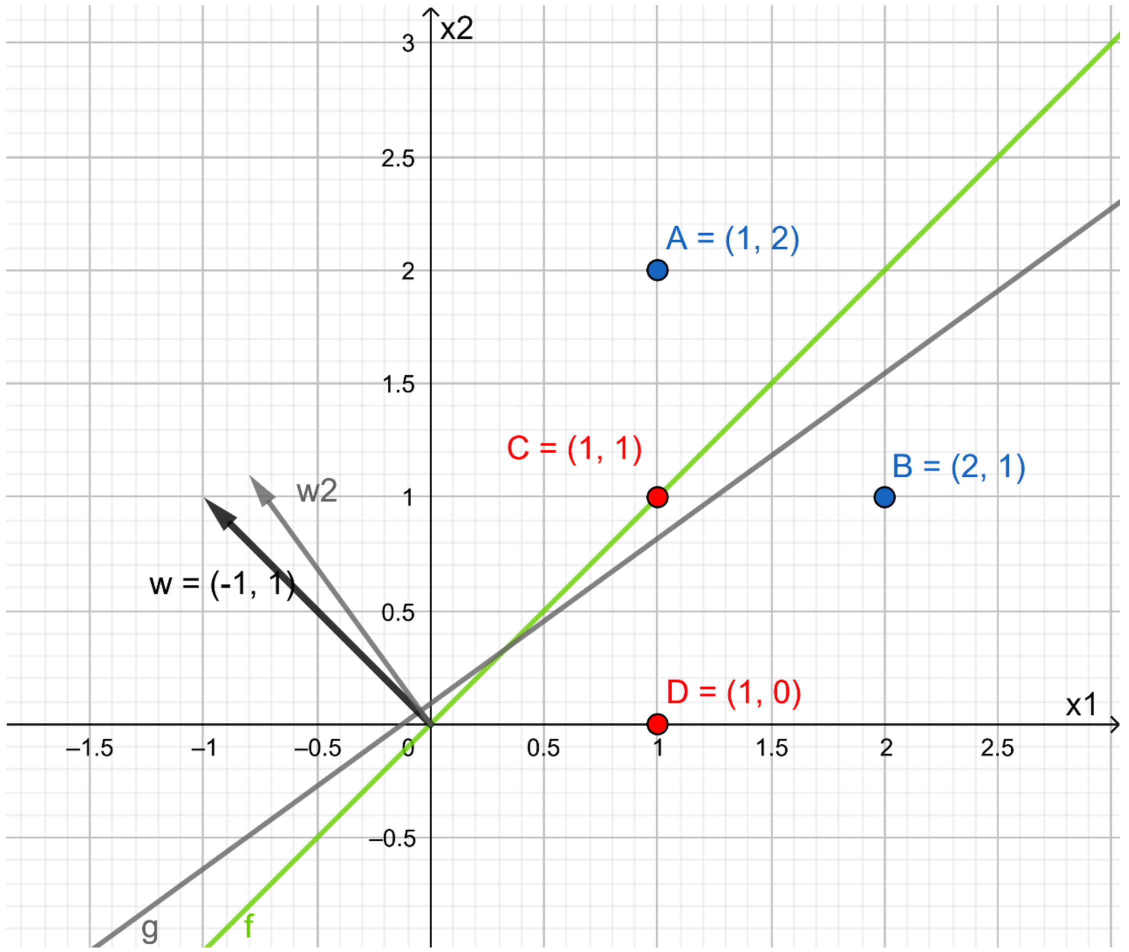
b) Update perceptron point B

- $t = 1$
- $z = \boldsymbol{w} \cdot \boldsymbol{x} =$
  $$(0, -1,1) \cdot (-1,2,1) = -1$$
- $y = 0$, because $z \leq 0$
- $w_i = w_i - \eta(y - t)x_i$
- Vector form: $\boldsymbol{w} = \boldsymbol{w} - \eta(y - t)\boldsymbol{x}$
- $\boldsymbol{w} = (0, -1,1) - 0.1(0 - 1)(-1,2,1)$
  $$= (-0.1, -0.8, 1.1)$$



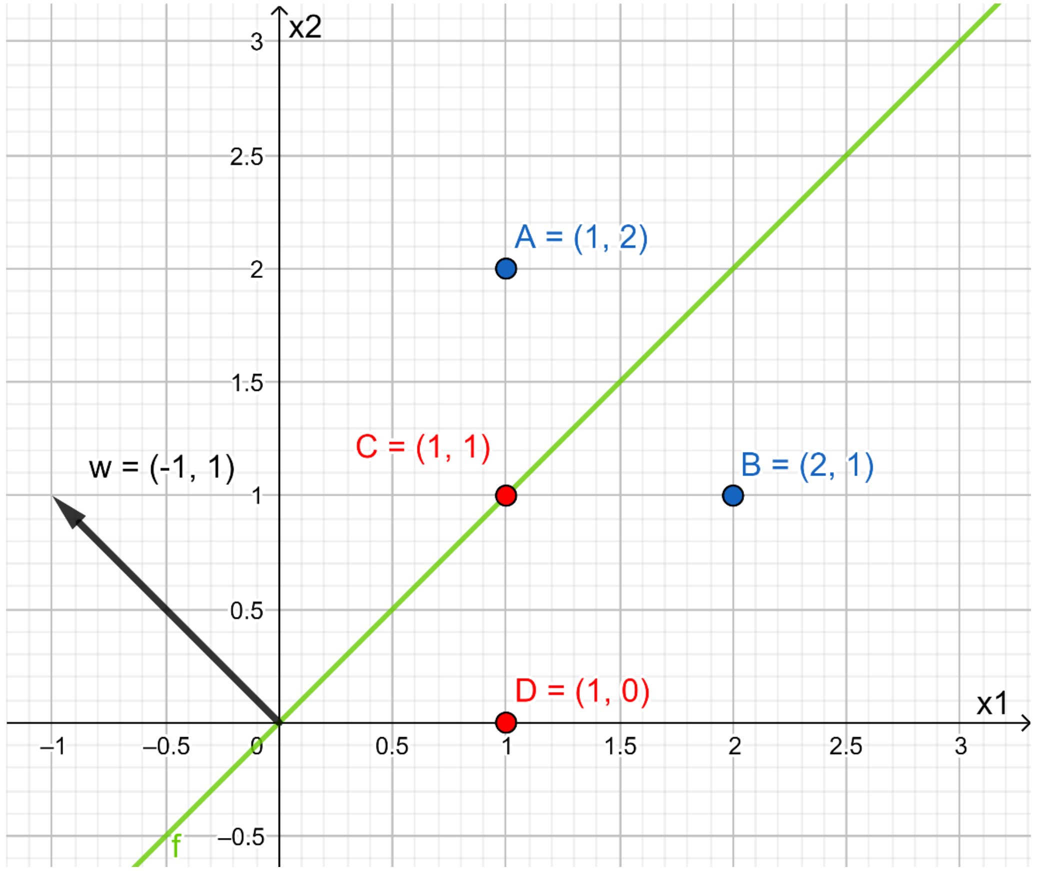
b) Update perceptron point B

- $t = 1$
- $z = \boldsymbol{w} \cdot \boldsymbol{x} =$
  $(0, -1, 1) \cdot (-1, 2, 1) = -1$
- $y = 0$, because $z \leq 0$
- $w_i = w_i - \eta(y - t)x_i$
- Vector form: $\boldsymbol{w} = \boldsymbol{w} - \eta(y - t)\boldsymbol{x}$
- $\boldsymbol{w} = (0, -1, 1) - 0.1(0 - 1)(-1, 2, 1)$
  $= (-0.1, -0.8, 1.1)$



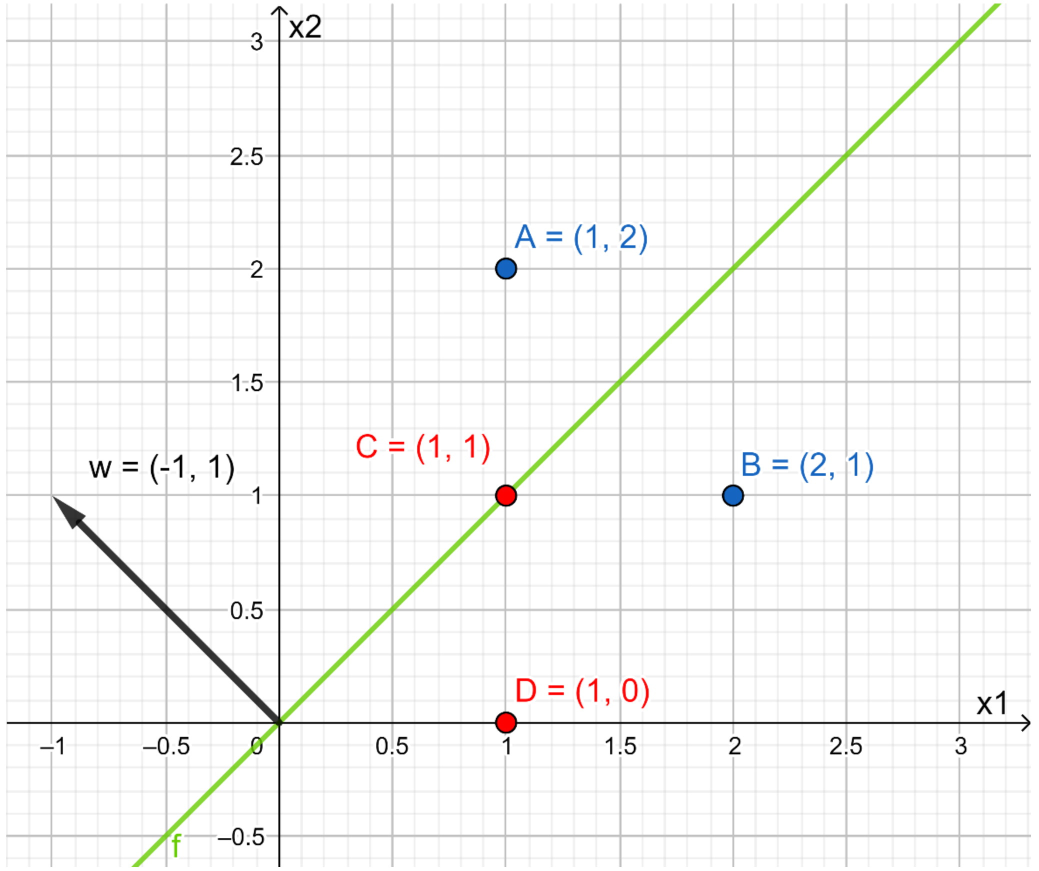
c) Update linear regression point A

- $t = 1$
- $y = z = \boldsymbol{w} \cdot \boldsymbol{x} =$
  $(0, -1, 1) \cdot (-1, 1, 2) = 1$
- Since $y = t$, do nothing!
- But beware:
  - The weights could be updated even for correctly classified points, e.g., E=(1,10)



c) Update linear regression point B

- $t = 1$
- $y = z = \boldsymbol{w} \cdot \boldsymbol{x} =$
  $$(0, -1,1) \cdot (-1,2,1) = -1$$
- $\boldsymbol{w} = \boldsymbol{w} - \eta(y - t)\ \boldsymbol{x}$
- $\boldsymbol{w} = (0, -1,1) - 0.1(-1 - 1)(-1,2,1) = (-0.2, -0.6, 1.2)$
- Provided we use 0.5 squared error as loss:
  - $\frac{1}{2}\sum_{j=1}^{N}(y_j - t_j)^2$
- With mean squared error, we must modify accordingly



Footnote: variants

- Root Mean Square Error(RMSE):
  - $\sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2}$
- MSE: $\frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$
- SE: $\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$
- $\frac{1}{2}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$

- They all have reach minimum for the same values of x_i-s and w_i-s.
- RMSE has the ''right scale'', but not suited for finding min.
- Mean (MSE or RMSE) is needed for comparison across different training/test sets

MLP (a)

Forwards (4.7):

- Classification:
  - $y_k = g(h_k) = \frac{1}{1+e^{-\beta h_k}}$
- Regression:
  - $y_k = g_r(h_k) = h_k$

Backwards (4.8):

- Classification
  - $\delta_o(k) = \underbrace{(y_k - t_k)}_{\text{From loss}}\underbrace{y_k(1 - y_k)}_{g'(h_k)}$
- Regression
  - $\delta_o(k) = \underbrace{(y_k - t_k)}_{\text{From loss}}$ $\boxed{g_r'(h_k) = 1}$

MLP (b)

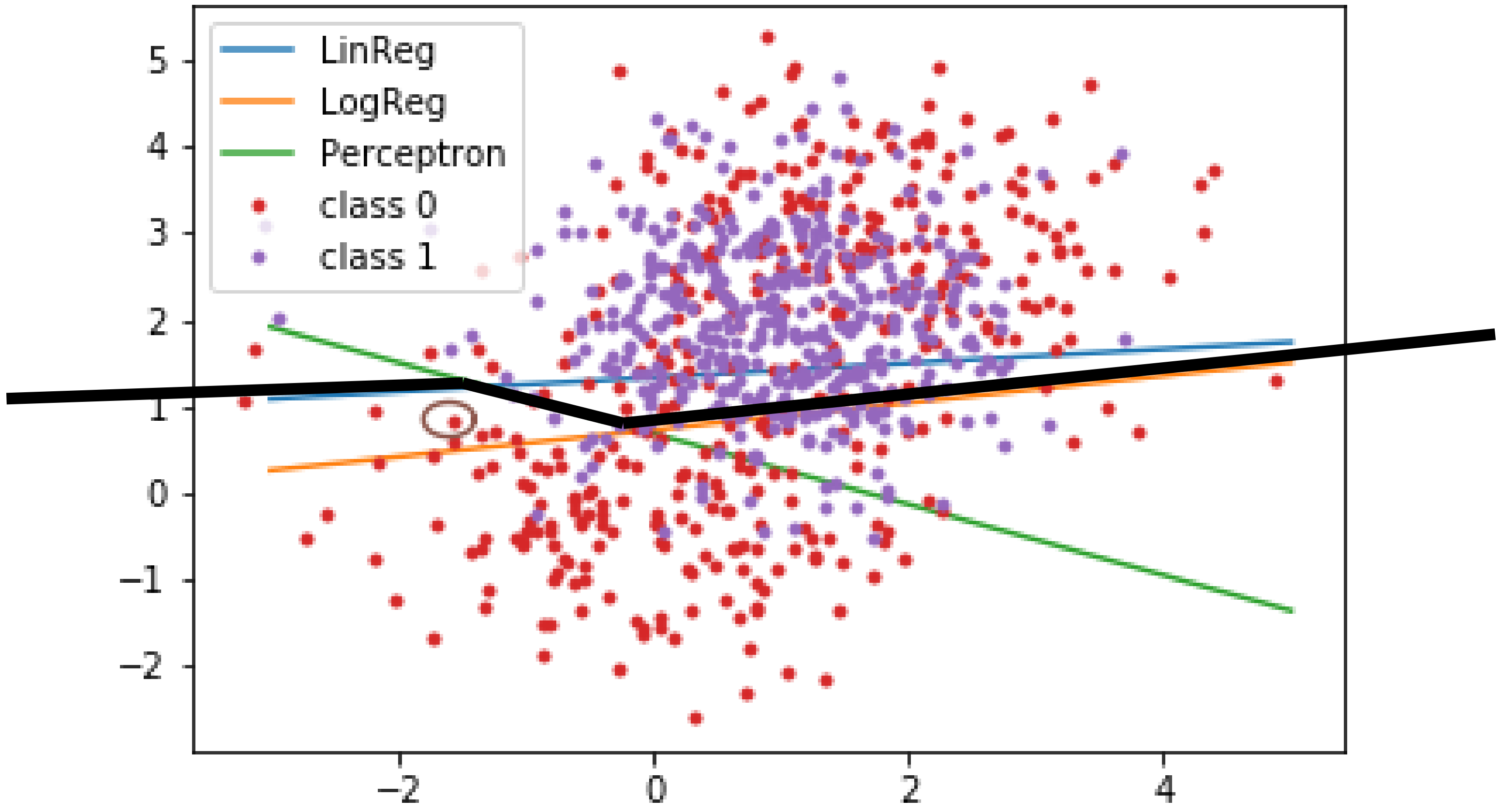
Forwards (4.5):

- Logistic:
  - $a_\zeta = g(h_\zeta) = \frac{1}{1+e^{-\beta h_\zeta}}$
- RELU:
  - $a_\zeta = g_R(h_\zeta) = RELU(h_\zeta) = \max(h_\zeta, 0)$

Backwards (4.9):

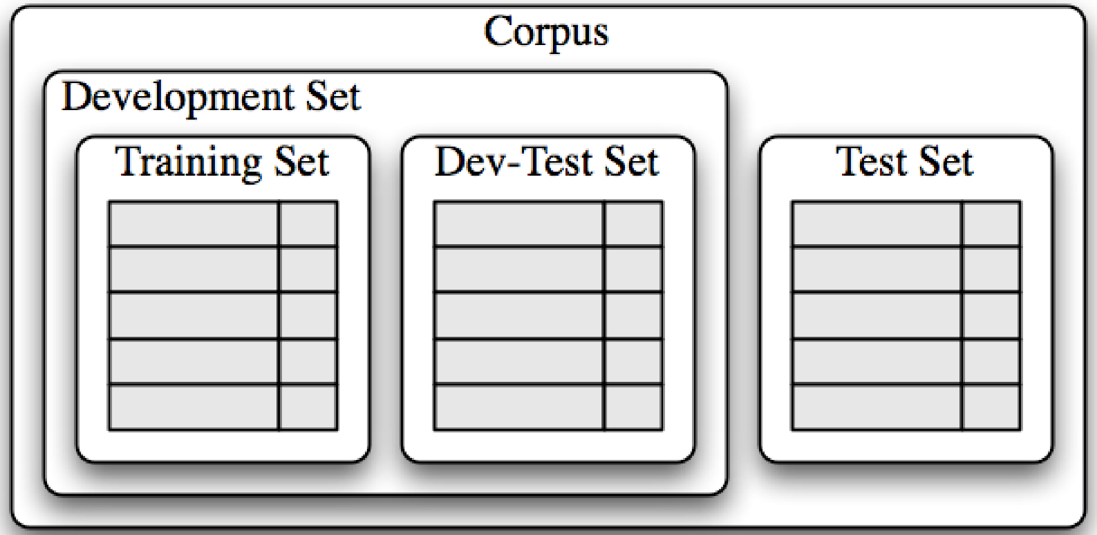
- Logistic
  - $\delta_h(\zeta) = \underbrace{a_\zeta(1 - a_\zeta)}_{\boxed{g'(h_k)}} \underbrace{\sum_{k=1}^{N} w_\zeta \delta_0(k)}_{\boxed{\text{From next layer}}}$
- RELU
  - $\delta_h(\zeta) = \underbrace{int(h_\zeta \geq 0)}_{\boxed{g_R'(h_k)}} \underbrace{\sum_{k=1}^{N} w_\zeta \delta_0(k)}_{\boxed{\text{From next layer}}}$

Majority voting classifier



Training and test sets

- To measure improvement, we need (at least) two disjoint labeled sets:
  - Training set
  - Test set
- Train on the training set.
- Predict labels on the test set (after removing the labels)
- Compare the prediction to the given labels



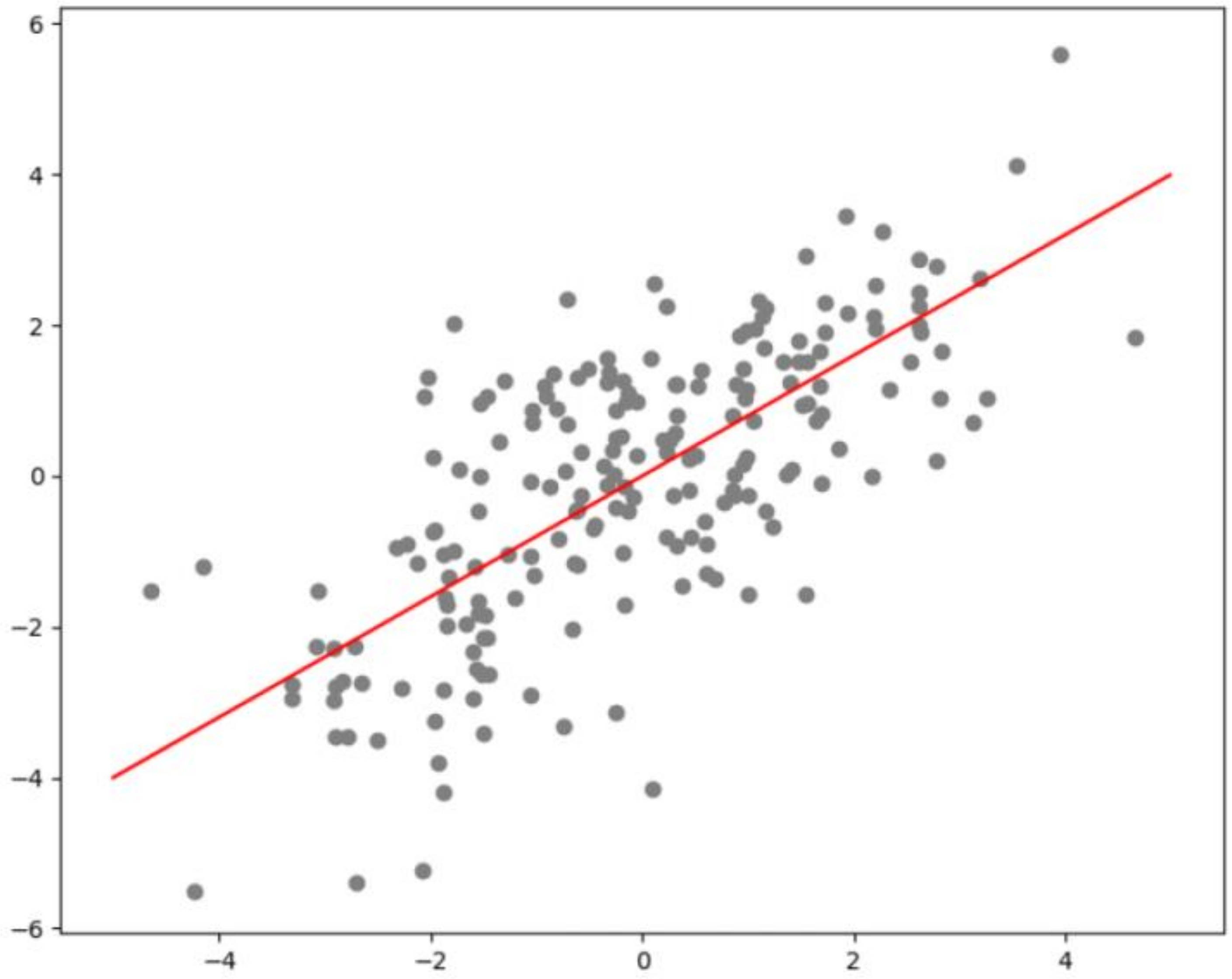
https://www.nltk.org/book/ch06.html

- For repeated development we need (at least) two test sets.
  - One for repeated testing during development
  - One for final testing

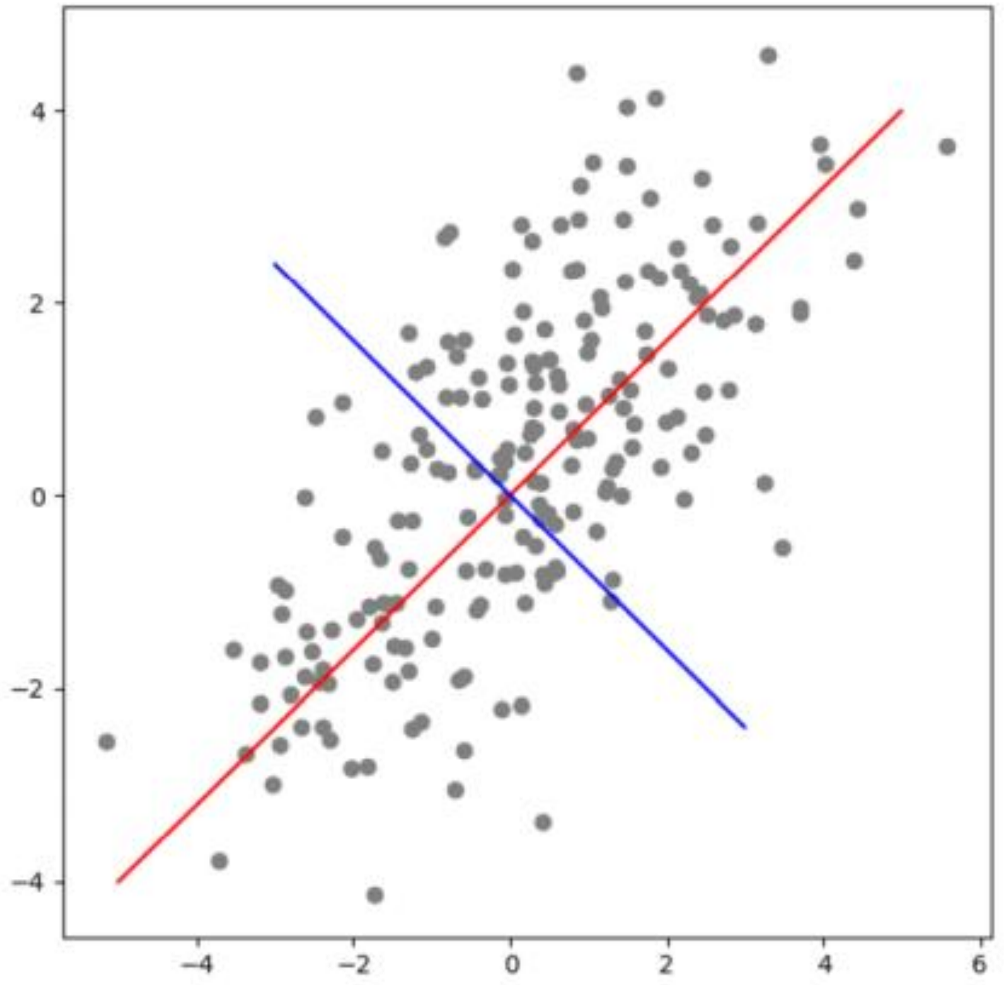
Unsupervised Learning (8p bachelor / 8p master)

a) (Master Students Only) An application of unsupervised learning discussed in class and in the exercises is dimensionality reduction. Sometimes, however, it may be necessary to project data from a lower-dimensional space to a higher-dimensional space. Consider the two algorithms that you have used for dimensionality reduction: **PCA and autoencoders**; **can they be used to generate higher dimensional representations?** Briefly explain why/why not. (4p)

PCA selects that dimension along which the data spread the most.



Further dimensions are chosen to be perpendicular to the one already selected.



Given data matrix $\mathbf{X}$ with dimension $N \times D$ (N samples, D dimensions), we want to compute the lower-dimensional representation $\mathbf{Z}$ with dimension $N \times M$:

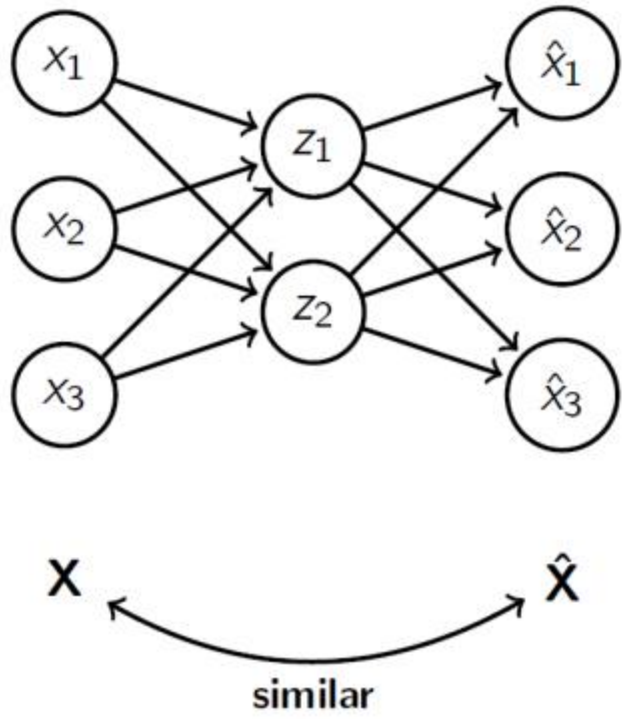
1. (Center the data $\mathbf{X}$)
2. Compute the *coviariance matrix* of the data:

$$\mathbf{C} = \frac{1}{N}\mathbf{X}^T\mathbf{X}$$

3. Compute the *eigenvalues* $\lambda_1, \lambda_2, ..., \lambda_D$ and the associated *eigenvectors* $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_D$;
4. Sort eigenvalues from big to small and select top-M eigenvalues and their associated eigenvectors;
5. Assemble the chosen eigenvectors into a matrix:

$$\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_M]$$

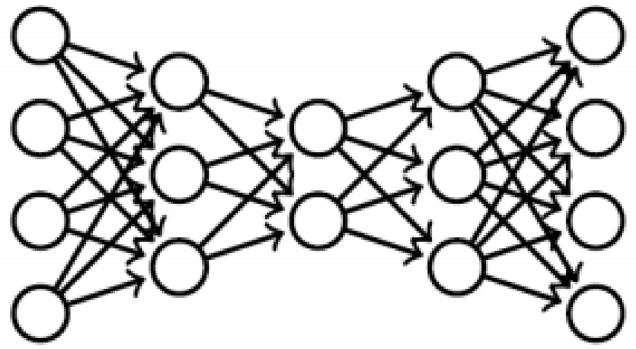
An **autoencoder** uses the same original data $\mathbf{X}$ as a target for training.



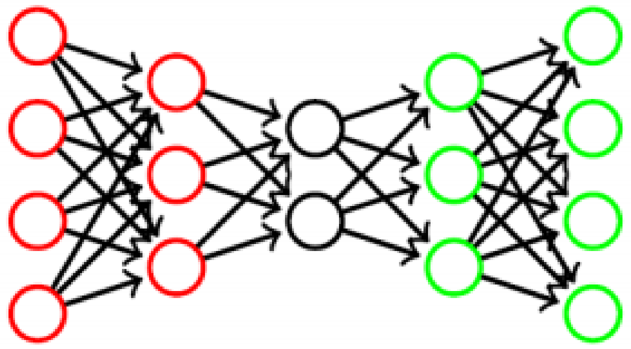
The original data $\mathbf{X}$ and the reconstruction $\hat{\mathbf{X}}$ are forced to be as similar as possible.

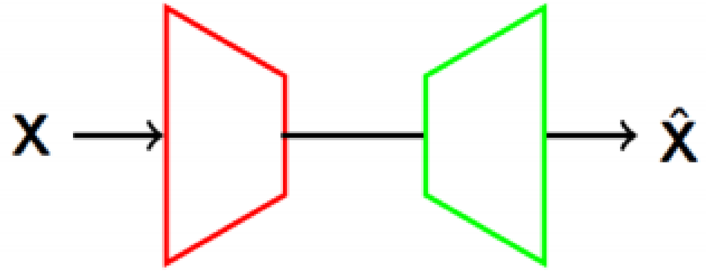
An **autoencoder** can be viewed as:

Bottleneck single neural network



An encoder and decoder network



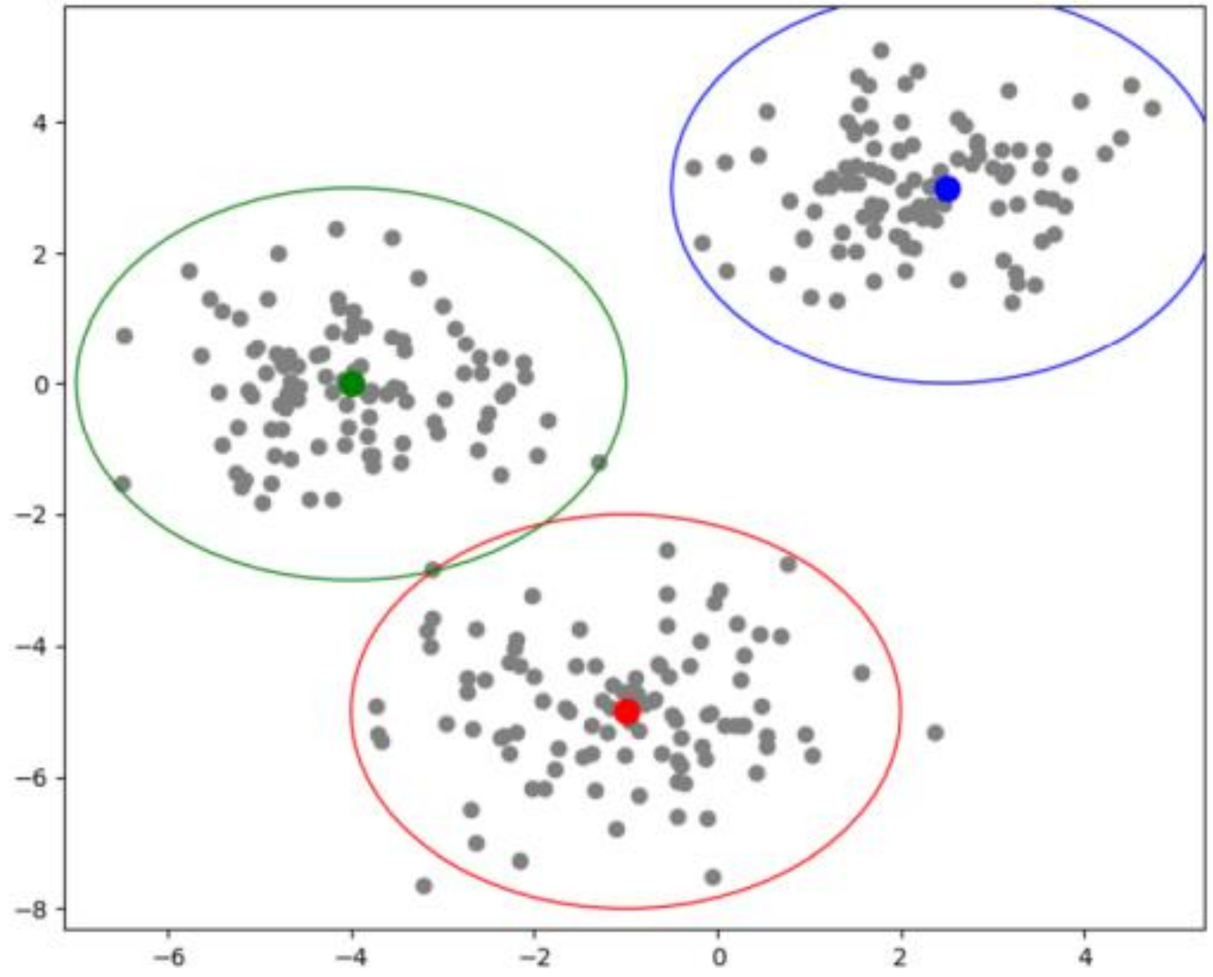


(Strictly speaking, an autoencoder must not necessarily have a bottleneck shape)

Unsupervised Learning

b) (Bachelor Students Only) Overfitting is a central problem in learning. Suppose you have been given an **unlabeled data set containing 1000 samples in 20 dimensions**. You ran the **K-means algorithms on it using 900 centroids**, and you achieved a satisfying matching. Now, **someone examines your work, and states that your algorithm is overfitting** the data. What does she mean? How is this related to overfitting? (4p)

K-Means finds iteratively the centers of clusters.



Overfitting?

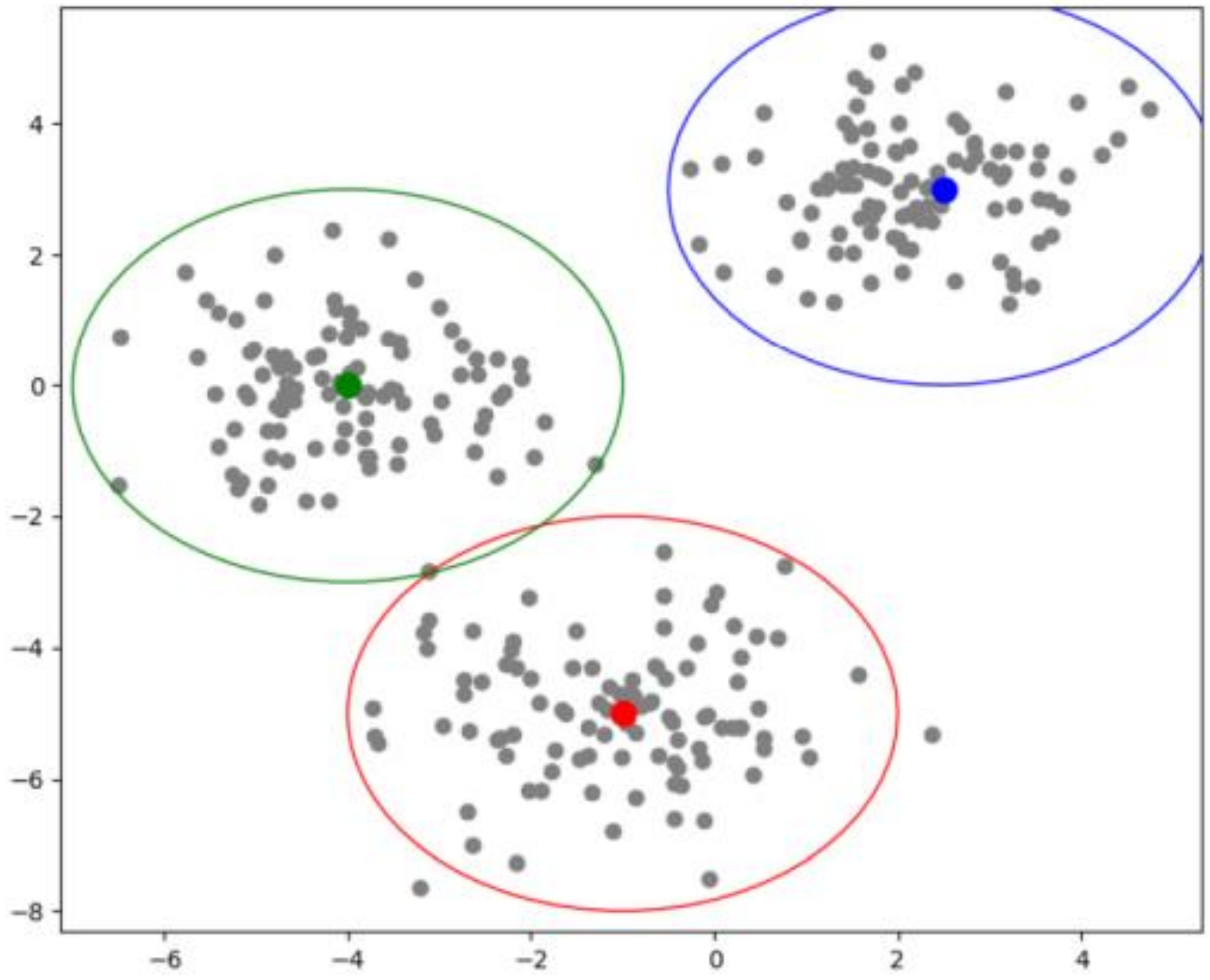
- The large number of centroids means that **most centroids are likely identifying individual datapoints**; there is no real learning, as **the whole data is memorized with no generalization**; **processing of new data will likely be unreliable**.

Unsupervised Learning

- **Big data** is undoubtedly an important driver for machine learning. However, in certain situations there may be limits on the memory space or the computational power available (think of embedded systems, for instance). In these cases, **we may prefer running our algorithms on few selected datapoints**. Suppose **you have been given an unlabeled data set containing 1000 samples in 20 dimensions**, and **someone has told you that they want only 10 representative samples (*prototypes*)** to run their analysis. **Out of the unsupervised algorithms you have studied (PCA, K-means, autoencoders), which one would you use and how?** (4p)

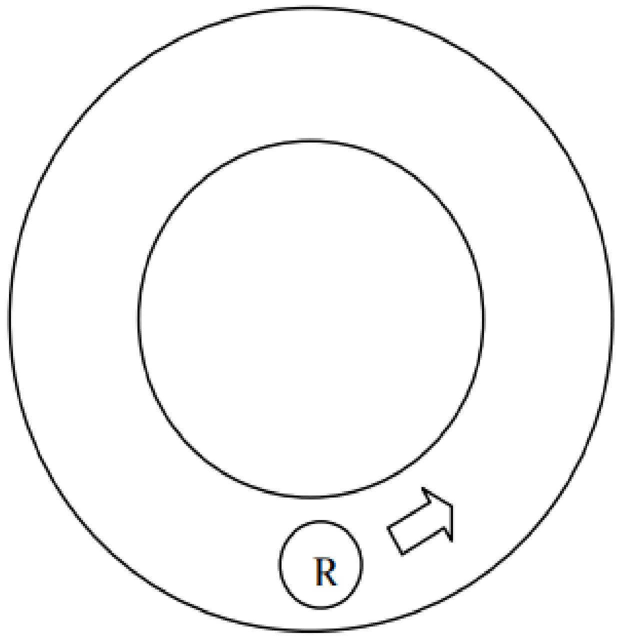
Data Prototypes

- The most natural solution would be to **use k-means to find 10 centroids, and use those as prototypes.**



Evolutionary Algorithms and Reinforcement Learning (13p)

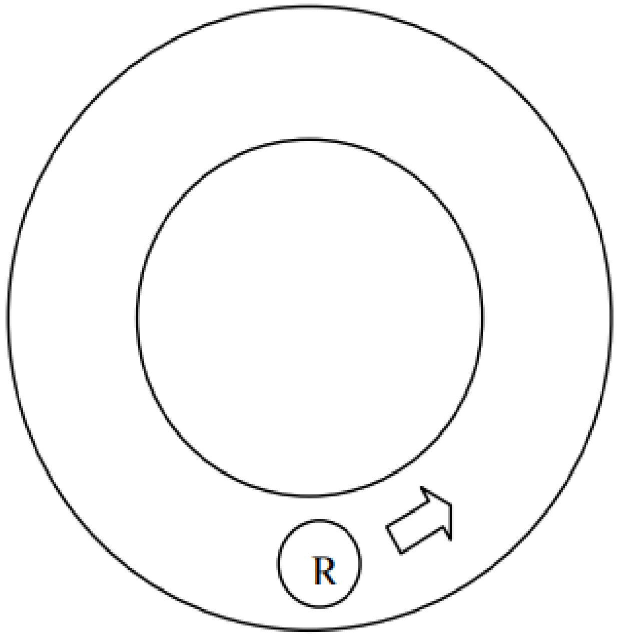
We would like to set up a **neural network (multilayer perceptron) for robot control**. The **inputs** to the neural network are measurements from **range sensors**, and the **output** is a **direction** of movement. The robot is inserted into the circular maze shown to the right, and the **goal** is to enable it to **drive in the direction of the arrow**, getting **as far as possible** within a given time limit, while **colliding** with the walls as **few times as possible**.



Evolutionary Algorithms and Reinforcement Learning (13p)

One way to design this neural network is by use of an **evolutionary algorithm (EA)**. The individuals in the **population** will be **possible robot controlling networks** that get their **fitness computed in simulation**.

The “lifetime” of one EA individual (a neural network) thus consists of many timesteps, where in **each timestep the individual receives inputs** from sensors, **and calculates outputs** (by feeding inputs forward through its connections) that are given as speed settings to the robot’s wheels. The wheels then follow this setting until a new output is available. **Each individual gets N timesteps to try to drive as far as possible.**



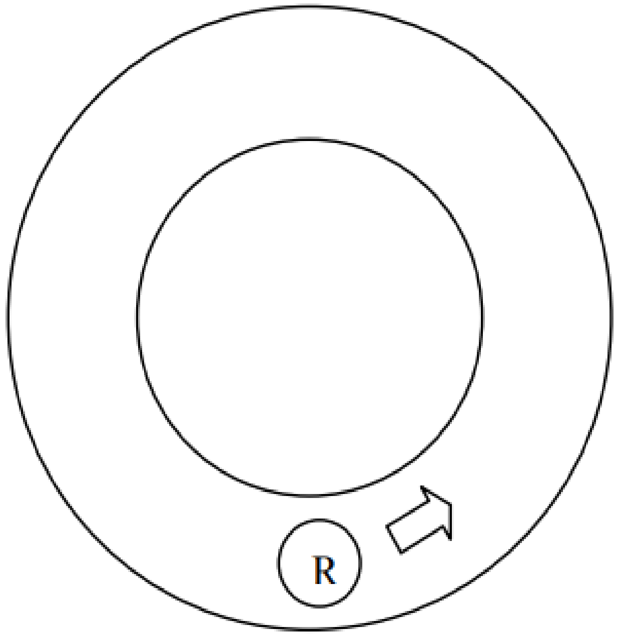
Evolutionary Algorithms and Reinforcement Learning (13p)

Assuming that the **structure of the network is already specified**, briefly describe how you could allow an EA to **find the proper weights** for this neural network. Include in your description a possible choice for:

a1) the genetic representation (**genotype**)

a2) **variation operators**. Include both their names and a brief description of how they work

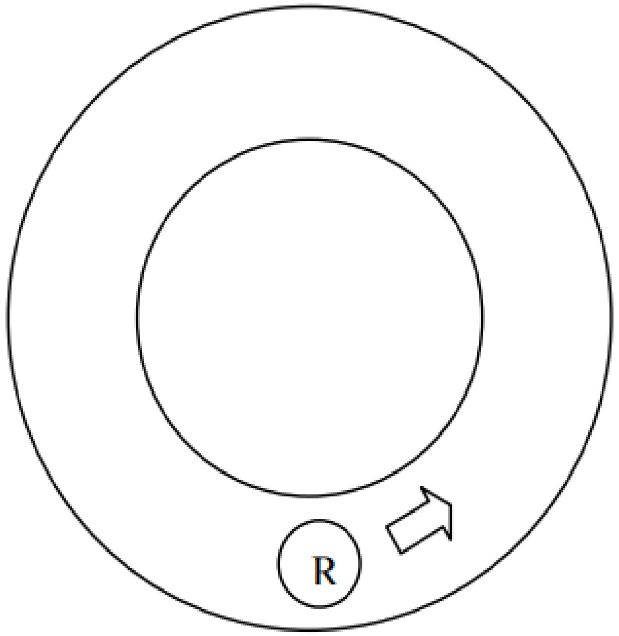
a3) **which measurements to include in the fitness function**. You can assume the robot or the simulator can gather any physical measurements of relevance to fitness calculation



Evolutionary Algorithms and Reinforcement Learning (13p)

a1) the genetic representation (**genotype**)

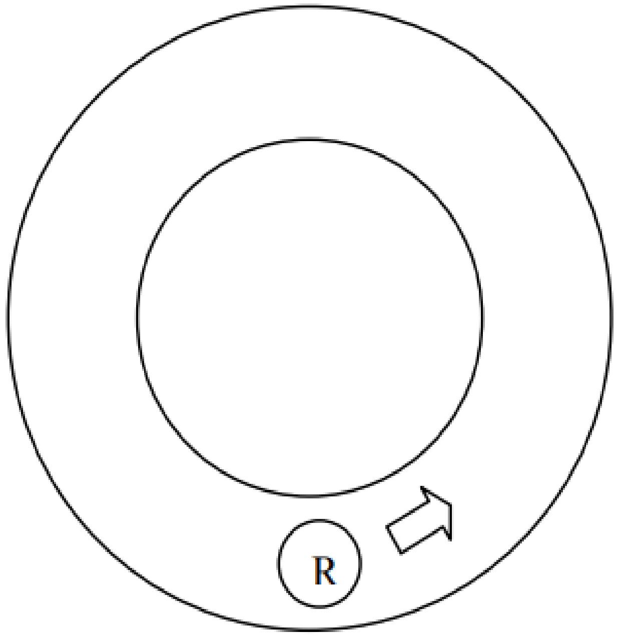
Easiest solution: A **list of floating-point values,** where **each value represents the weight** of a single specific network connection



Evolutionary Algorithms and Reinforcement Learning (13p)

a2) **variation operators**. Include both their names and a brief description of how they work

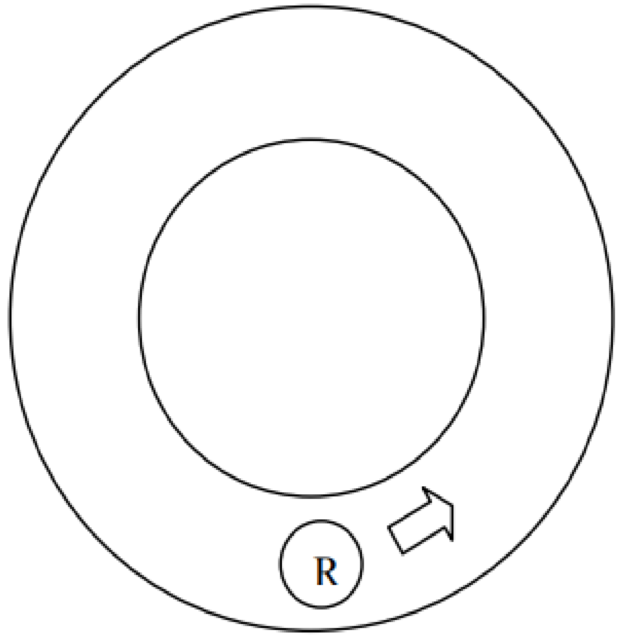
Here, one should **choose variation operators suitable for the representation defined in a1**. Since we defined the genome as a list of floating-point values, we could for instance select **uniform mutation** and **simple arithmetic crossover** here.



Evolutionary Algorithms and Reinforcement Learning (13p)

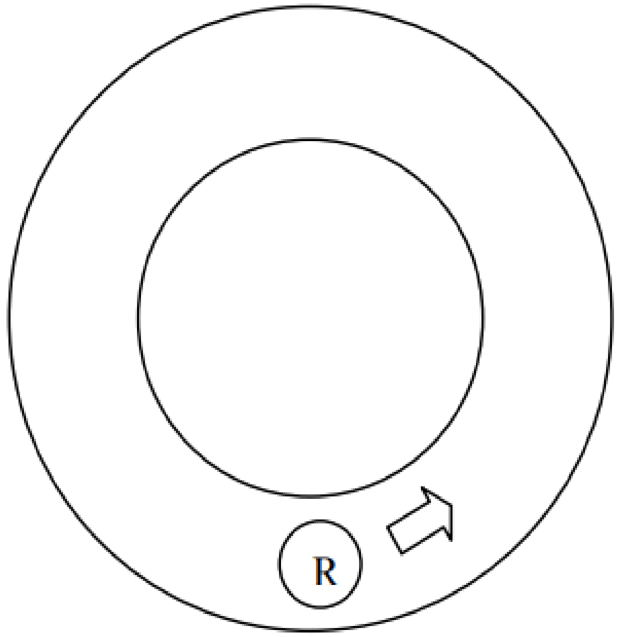
a3) **which measurements to include in the fitness function**.

Since the goal of evolved controllers is to drive as far as possible within a time limit without crashing into walls, we should include measurements of the **distance travelled** and the **total number of wall collisions** in the fitness function.



Evolutionary Algorithms and Reinforcement Learning (13p)

b) A different way to solve this problem is to apply **reinforcement learning (RL).** Describe how you would model this problem as a reinforcement learning problem, including **how you would define rewards, states and actions**. The RL *algorithm* is not to be described. Note that you should not describe this as a deep reinforcement learning problem, or with other function approximation techniques – rather, formulate it as a discrete RL problem like the examples from the text book and lectures. (7 points)



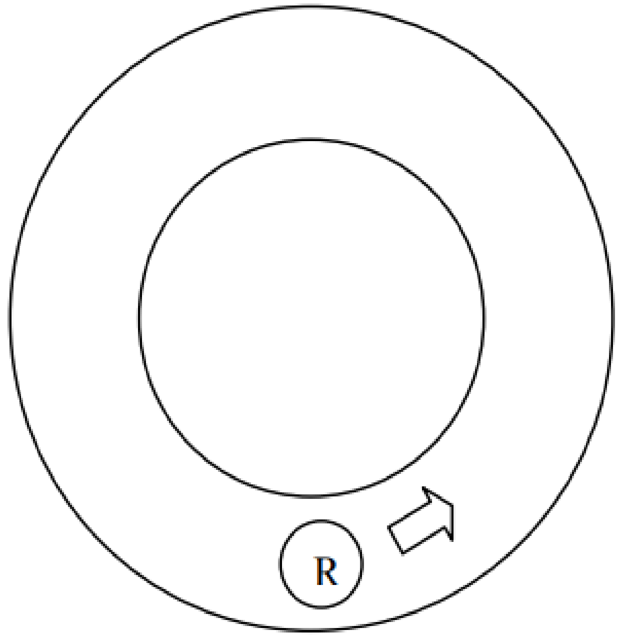
Evolutionary Algorithms and Reinforcement Learning (13p)

b) States:

-should include information about **distance to walls** and the **direction** towards walls

-Example: **State = (dir_wall, dist_wall)**

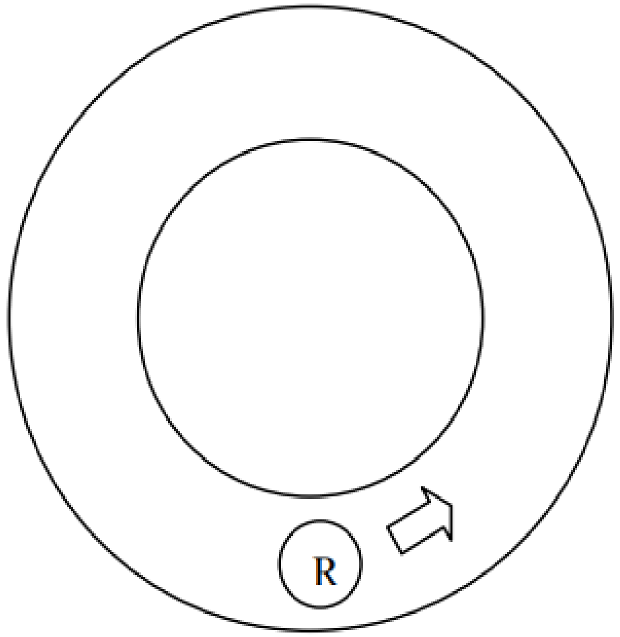
-We should define a **discrete set of state values**, e.g. dir_wall = [left, front, behind, right], dist_wall = [short, medium, far]



Evolutionary Algorithms and Reinforcement Learning (13p)

b) **Actions:**

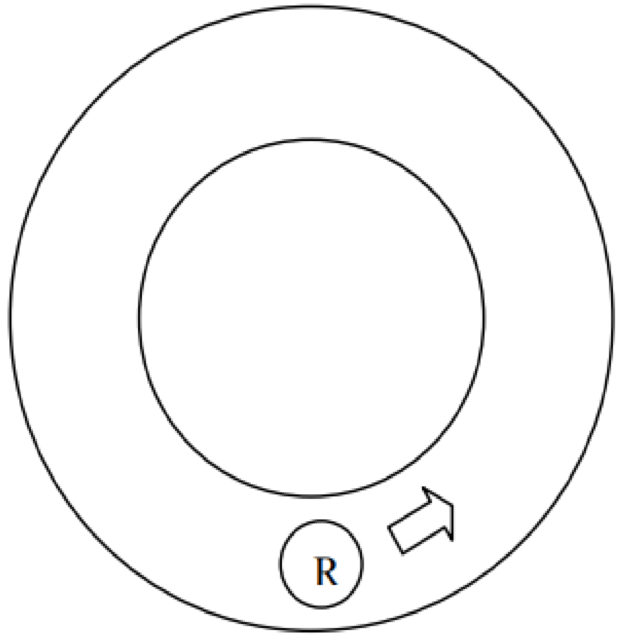
These need to be the **operations the robot can carry out** in order to complete its task. We could **discretize** the robot's (continuous) control into a few different **actions** such as (**go forward, go backward, turn left, turn right**).



Evolutionary Algorithms and Reinforcement Learning (13p)

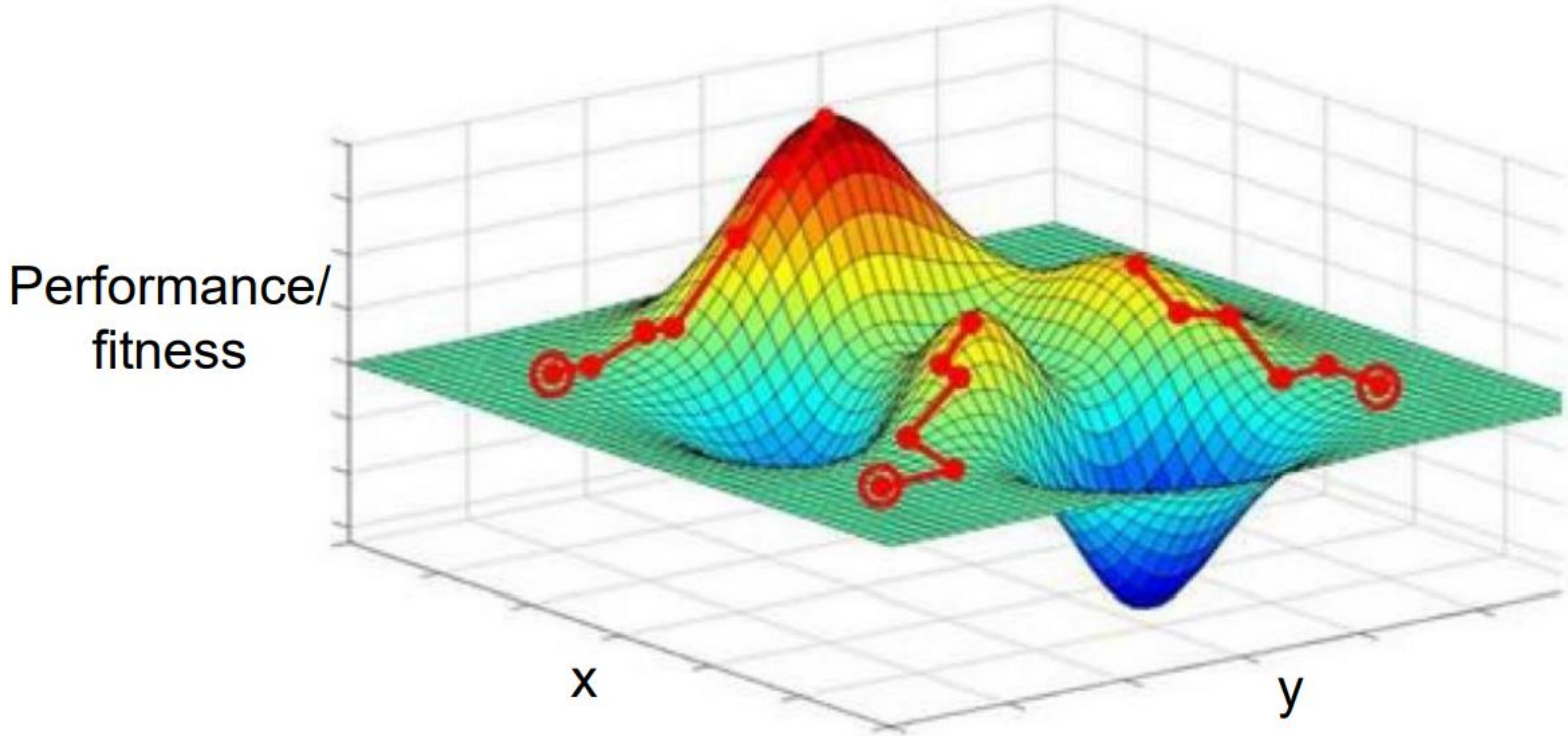
b) **Rewards:**

These need to be adapted to the robot's goal, which is to drive far without collisions. For instance, one could give a **positive reward for every N cm driven**, and a **negative reward for every collision**.

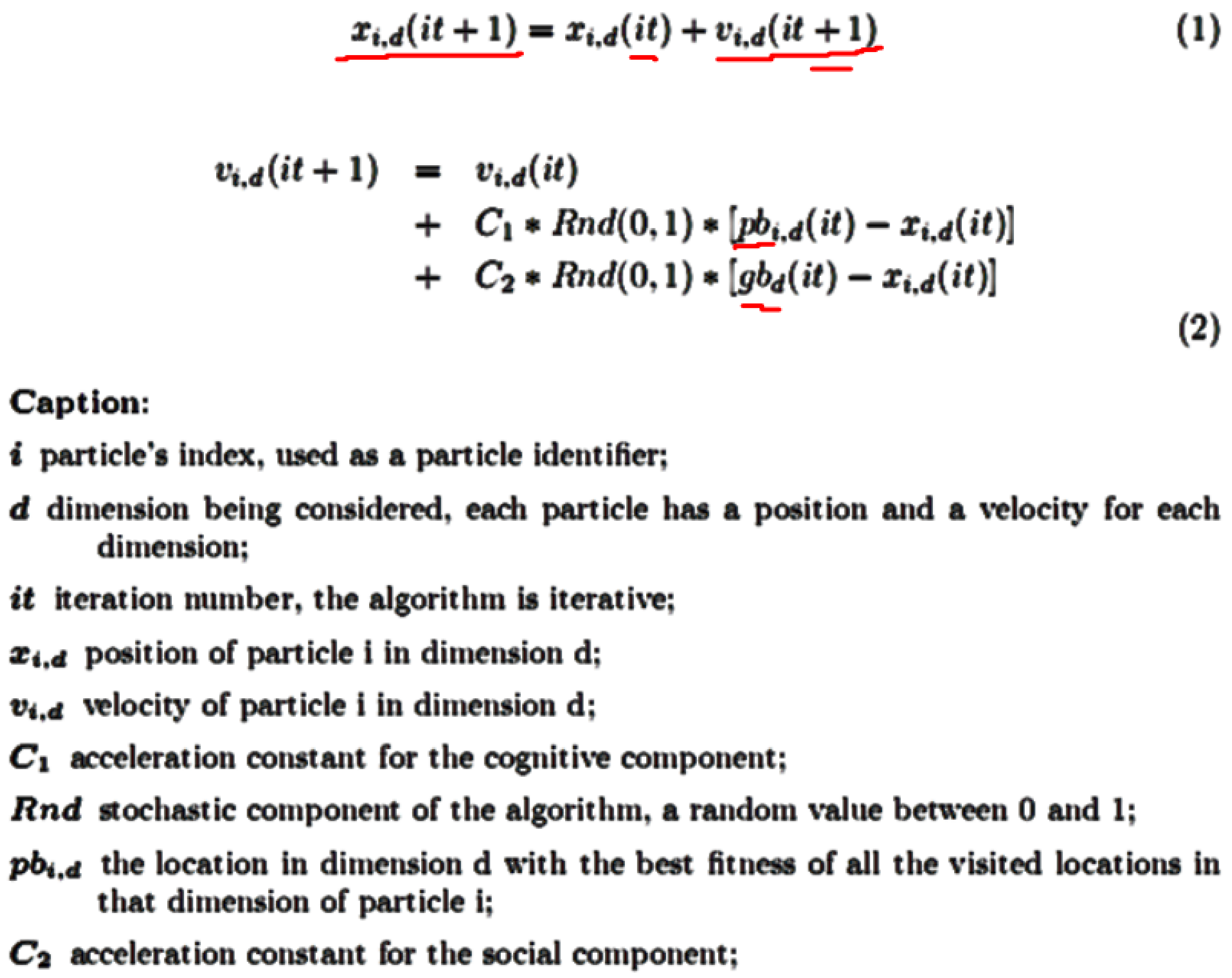


Particle Swarm Optimization (PSO) and Developmental Systems (9p)

a) Describe what happens when the **position of all particles in PSO are set to the same value of a local optimum** (Think about fitness landscapes). How could you **adjust PSO to get out of the local optimum to potentially find the global optimum without resetting the particles** to random positions initially? Describe your approach in up to 100 words. (3p)


Performance/
fitness
x
y

- (eq 1) updates positional parameters *x*
- (eq 2) updates velocity parameters *v*
  - *C1* and *C2* are *acceleration constants*
  - *pb* and *gb* are the *particle's best* and *global best* positions found



$$\underline{x_{i,d}(it+1)} = x_{i,d}(\underline{it}) + \underline{v_{i,d}(it+1)} \quad (1)$$

$$\begin{aligned} v_{i,d}(it+1) &= v_{i,d}(it) \\ &+ C_1 * Rnd(0,1) * [\underline{pb}_{i,d}(it) - x_{i,d}(it)] \\ &+ C_2 * Rnd(0,1) * [\underline{gb}_d(it) - x_{i,d}(it)] \end{aligned} \quad (2)$$

Caption:

i particle's index, used as a particle identifier;

d dimension being considered, each particle has a position and a velocity for each dimension;

it iteration number, the algorithm is iterative;

$x_{i,d}$ position of particle i in dimension d;

$v_{i,d}$ velocity of particle i in dimension d;

C_1 acceleration constant for the cognitive component;

Rnd stochastic component of the algorithm, a random value between 0 and 1;

$pb_{i,d}$ the location in dimension d with the best fitness of all the visited locations in that dimension of particle i;

C_2 acceleration constant for the social component;

Particle Swarm Optimization (PSO): Get out of the local optimum

You can for example:

-set the initial velocity of every particle to a very high number

-add a repulsion function that makes particles repel one another

$$x_{i,d}(it+1) = x_{i,d}(it) + v_{i,d}(it+1) \tag{1}$$

$$\begin{aligned} v_{i,d}(it+1) &= v_{i,d}(it) \\ &+ C_1 * Rnd(0,1) * [pb_{i,d}(it) - x_{i,d}(it)] \\ &+ C_2 * Rnd(0,1) * [gb_d(it) - x_{i,d}(it)] \end{aligned} \tag{2}$$

Particle Swarm Optimization (PSO) and Developmental Systems (9p)

- An L-System is a parallel rewrite method. Describe how from an alphabet {h,j}, the axiom h will be rewritten when using the rewrite rules: h->jjh and j->h. Write down three iterations/recursions. (3p)

Rewriting systems

- Rewriting Systems
  - Sequential (Formal Grammar)
  - Parallel (Lindenmayer-System)

$$A \rightarrow AB$$

$$B \rightarrow A$$

Simple rewrite	**L-system**
AB	AB
AA	ABA
ABAB	ABAAB
AAAA	ABAABABA
ABABABAB	ABAABABAABAAB

Particle Swarm Optimization (PSO) and Developmental Systems (9p)

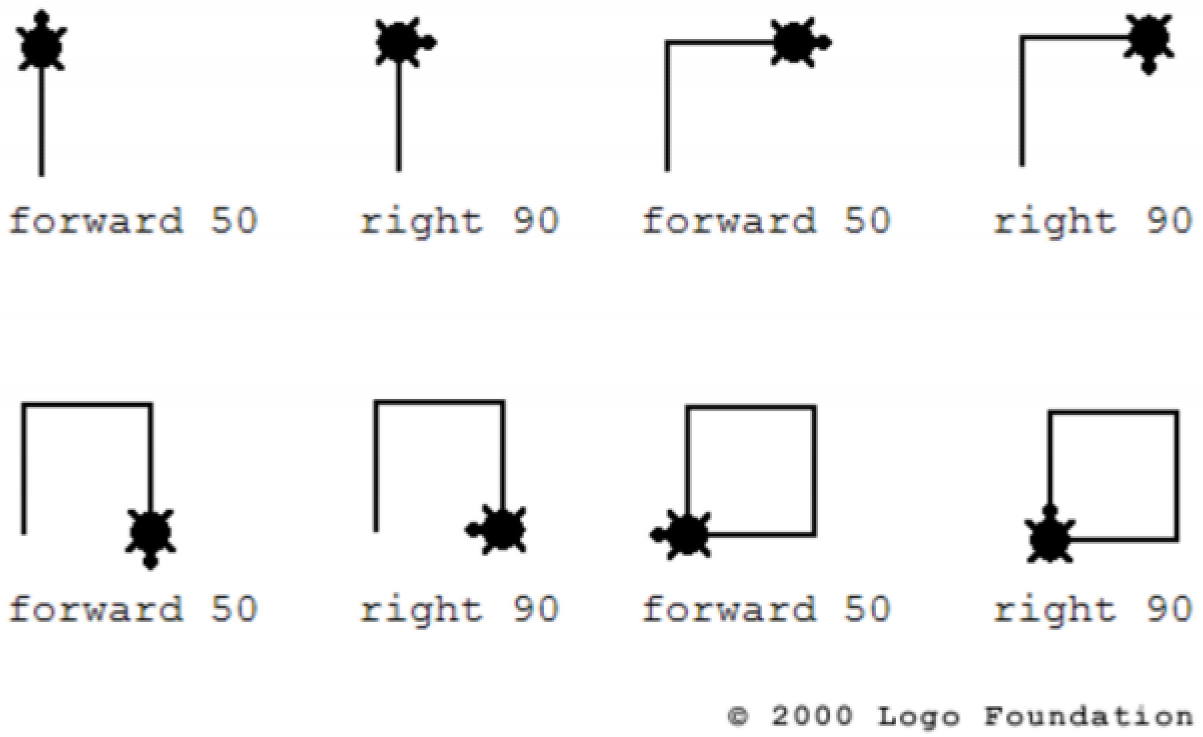
- An L-System is a parallel rewrite method. Describe how from an alphabet {h,j}, the axiom h will be rewritten when using the rewrite rules: h->jjh and j->h. Write down three iterations/recursions. (3p)
- h := jjh := hhjjh := jjhjjhhhjjh

Particle Swarm Optimization (PSO) and Developmental Systems (9p)

When visualizing a string of an L-System, it is useful to implement a bracketed L-System. Describe what ‘+’, ‘-’, ‘[‘ and ‘]’ in such L-Systems are used for. (3p)

L-Systems for generating graphics: turtle graphics

- Read the string produced by the L-System from left to right changing the state of the turtle



Example L-system for Drawing

- Alphabet: {F, f, +, -}
- F: move the turtle forward (drawing a line)
- f: move the turtle forward (don’t draw)
- +/-: turn right/left (by some angle)

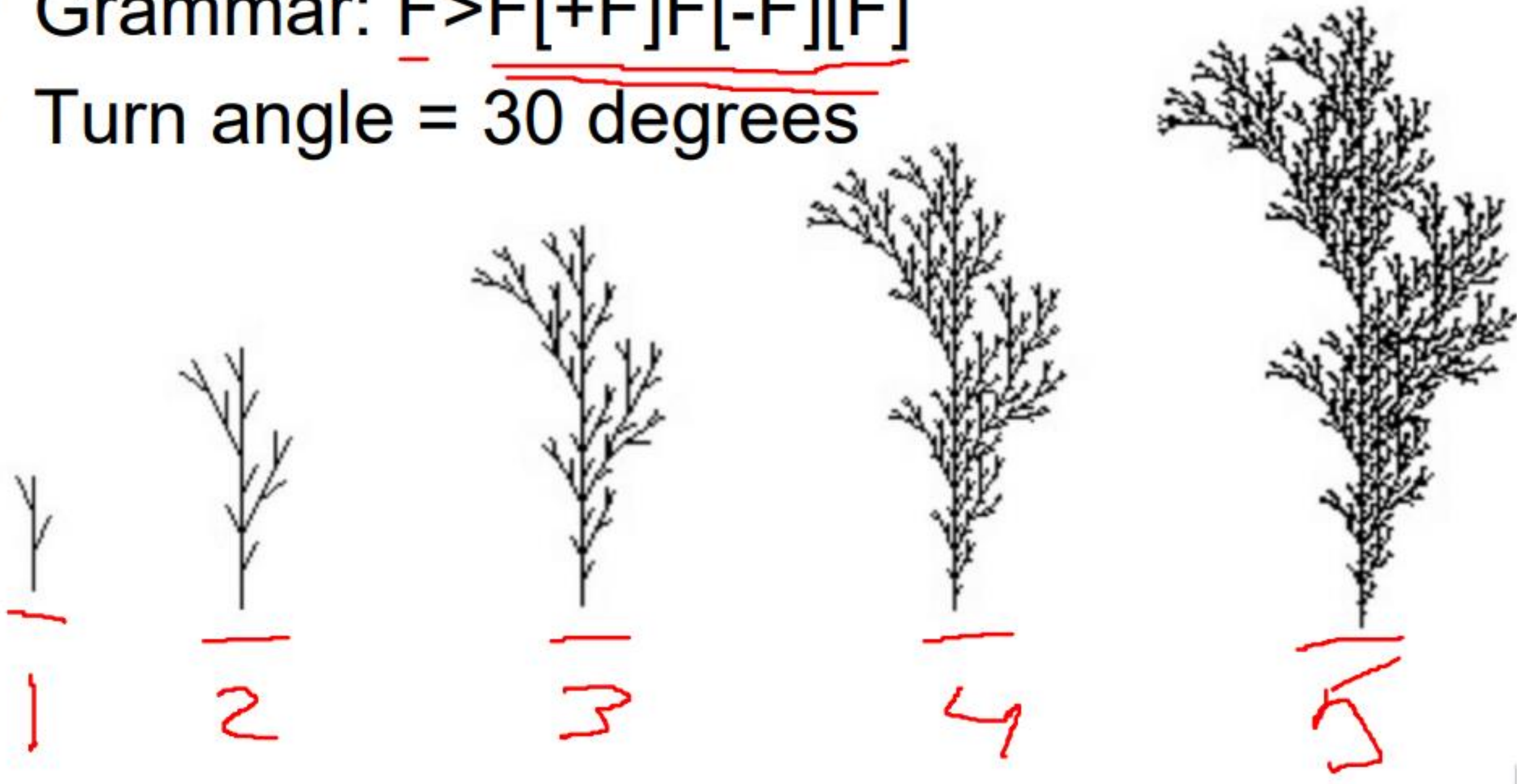
- What would FfFfFfFf do?
- What would F+F+F+F+ do?

Bracketed L-Systems

- Alphabet: {F, f, +, -, [,]}
- [: push the current state (x, y, heading of the turtle) onto a pushdown stack
-]: pop the current state of the turtle and move the turtle there without drawing
- Enables branching structures!

L-System

- Axiom: F
- Grammar: F>F[+F]F[-F][F]
- Turn angle = 30 degrees



Particle Swarm Optimization (PSO) and Developmental Systems (9p)

When visualizing a string of an L-System, it is useful to implement a bracketed L-System. Describe what ‘+’, ‘-’, ‘[‘ and ‘]’ in such L-Systems are used for. (3p)

Particle Swarm Optimization (PSO) and Developmental Systems (9p)

-) ‘+’ and ‘-’ refer to **turning left and right with a predefined angle**. The **brackets ‘[‘ and ‘]’ push and pop the state when drawing an L-System**. Pushing meaning that when the bracket is being read, the **position and angle are being stored**. The pop **returns the previously stored position and angle**. Bracketed L-Systems therefore enable branching. The string of an expanded L-System can be read as a set of instructions. Classical example is by having F draw a straight line. F+F[FF]+F would read as: (1)draw (2) turn (3) draw (4) push (5) draw (6) draw (7) pop (8) turn (9) draw.