Trial Exam IN3050/4050 Spring 2021 – With Solutions

Hi IN3050/4050-students! Below, we have made a trial exam consisting of questions representative for what you will see in this year's exam. Many of them have been used before (e.g. in last year's exam/trial exam), but these are all questions that we believe would also be fitting for this year's exam, and you can expect to see a similar style of problems on this year's final exam.

A difference between this and the final exam, however, is that the final exam will be given and delivered in Inspera. So, please take some time to familiarize yourself with how to do exams in Inspera if you have not done this already. Some links to useful information:

- Examination guidelines for Spring 2021 at the Faculty of Mathematics and Natural Sciences
- UiO's page for preparing for exams in Inspera
- Each question in our exam will be answered in the "long form assignment" style demonstrated here.
- We encourage you to log in to Inspera and test their demo exams, especially the "long form assignment" demo. Accessible here after logging in. In particular, have a look at the equation editor, with which it may be useful to have a bit of practice well before the exam.

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 like? (3p)

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

Suggested solutions

a) We would have a method doing almost only exploitation. This would be similar to hill climbing / local search, but with a bit more randomness, depending on how low we set the temperature.

b) We would have a lot of randomness and never settle on the areas in the fitness landscape with the best solution. This would be similar to random search/exhaustive search. *We would essentially move through random neighbors in the search landscape.*

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 solution

You could for example:

-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.

Suggested solution

With a constant population size, we need to select 5 individuals for the next generation. 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

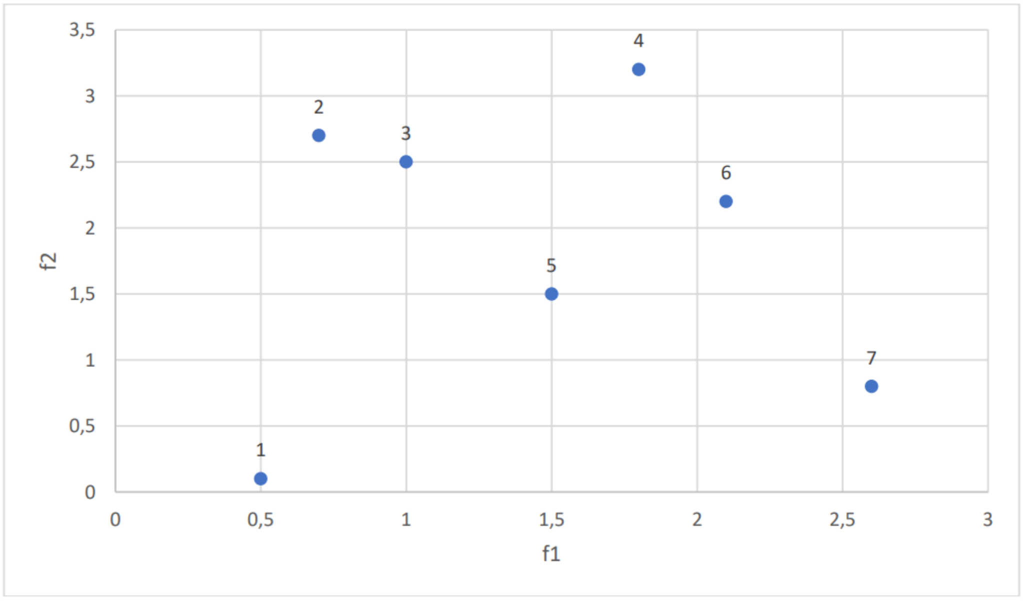
Expected number of copies:

- 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)
- 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)

We expect the final mating pool to contain 2 copies of individual 5, and 1 of each of individuals 2,3 and 4.

Pareto Optimality (9p)

For an optimization problem we wish to optimize solutions according to two different objectives, f1 and f2. The fitness values according to the two objectives for 7 different solutions are plotted in the figure below.



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

Find the Pareto optimal set of solutions when

b) Maximizing f1 and f2 (3p)
c) Maximizing f1 but minimizing f2 (3p)

Suggested solutions

a) Solutions on the Pareto optimal set have to be non-dominated. That is, no other solutions should be better according to both objectives (or even better on some but equally good on others).
b) 4, 6 and 7
c) 1, 7

Perceptron and linear regression classifier (12p)

Given the following data

Item	x1	x2	Class
A	1	2	yes =1
B	2	1	yes =1
C	1	1	no =0
D	1	0	no =0

a) Are the data linearly separable? State reasons for your answer. (4p)

b) We will train a perceptron on the data. We add a bias $x_0 = -1$ to each of the data points. Suppose the current weigths to be $\boldsymbol{w} = (0, -1, 1)$. Assume a learning rate of 0.1. How should the weights be updated if point A is considered? How would the weights have been updated if the algorithm instead had considered point B? (4p)

c) Say, we instead had applied a linear regression classifier. How should the weights have been updated when considering datapoint A, again assuming a learning rate of 0.1. And how would they have been updated if we instead considered point B? (4p)

Suggested solutions

a) The datapoints can be separated e.g. by the line $x_1 + x_2 = 2.5$

b) For A, we get $z = w_0x_0 + w_1x_1 + w_2x_2 = 0(-1) - 1(1) + 1(2) = 1$.
Since $z > 0$, we get the prediction $y = 1$.
Since $y = t$, there is no change to $\boldsymbol{w}$.

For B, we get $z = w_0x_0 + w_1x_1 + w_2x_2 = 0(-1) - 1(2) + 1(1) = -1$.
Since $z < 0$, we get the prediction $y = 0$. The update
$\boldsymbol{w} = \boldsymbol{w} - \eta(y - t)\boldsymbol{x} = (0, -1,1) - 0.1(0 - 1)(-1,2,1) = (-0.1, -0.8,1.1)$

c) Point A. Since $y = z = 1 = t$, there is no update.

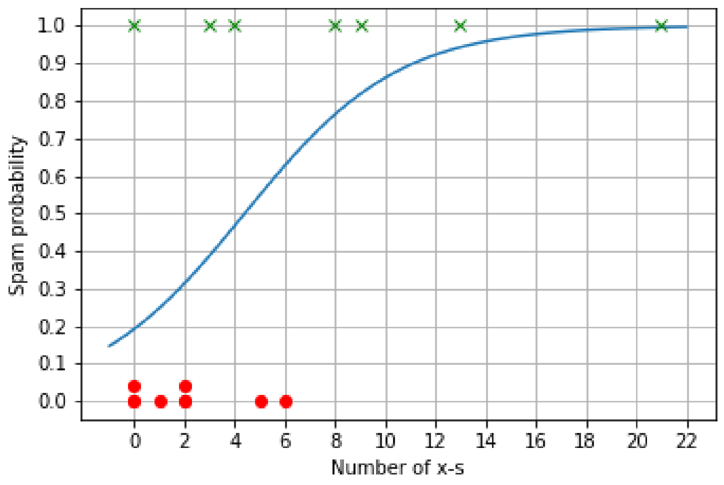
Datapoint B. Since $y = z = -1$, the update is
$\boldsymbol{w} = \boldsymbol{w} - \eta(y - t)\boldsymbol{x} = (0, -1,1) - 0.1(-1 - 1)(-1,2,1) = (-0.2, -0.6,1.2)$

Logistic Regression (12 p)

Kim is building a spam filter. She has the hypothesis that counting the occurrences of the letter ‘x’ in the e-mails will be a good indicator of spam or no-spam. She collects 7 spam messages and 7 no-spam messages and counts the number of x-s in each. Here is what she finds.

- Number of ‘x’-s in each spam: [0, 3, 4, 8, 9, 13, 21]
- Number of ‘x’-s in each no-spam: [0, 0, 1, 2, 2, 5, 6]

She trains a logistic regression classifier on the data and plots the classifier against the data.



Assume the logistic regression model and answer the following questions:

a) Consider an e-mail with no ‘x’-s. According to the model, what is roughly the probability of this message being a spam message and what is the probability of it not being a spam. (3p)

b) How many x-s must an e-mail contain to guarantee it is a spam mail? (3p)

c) How is a logistic regression model normally turned into a binary classifier? If you turn the model into a classifier in this way, what is the accuracy of the classifier on the training data? (3p)

d) It is most important that no no-spams are classified as spams. How can this goal be described in terms of precision and recall? How can the logistic regression classifier be modified to try to achieve this goal? (3p)

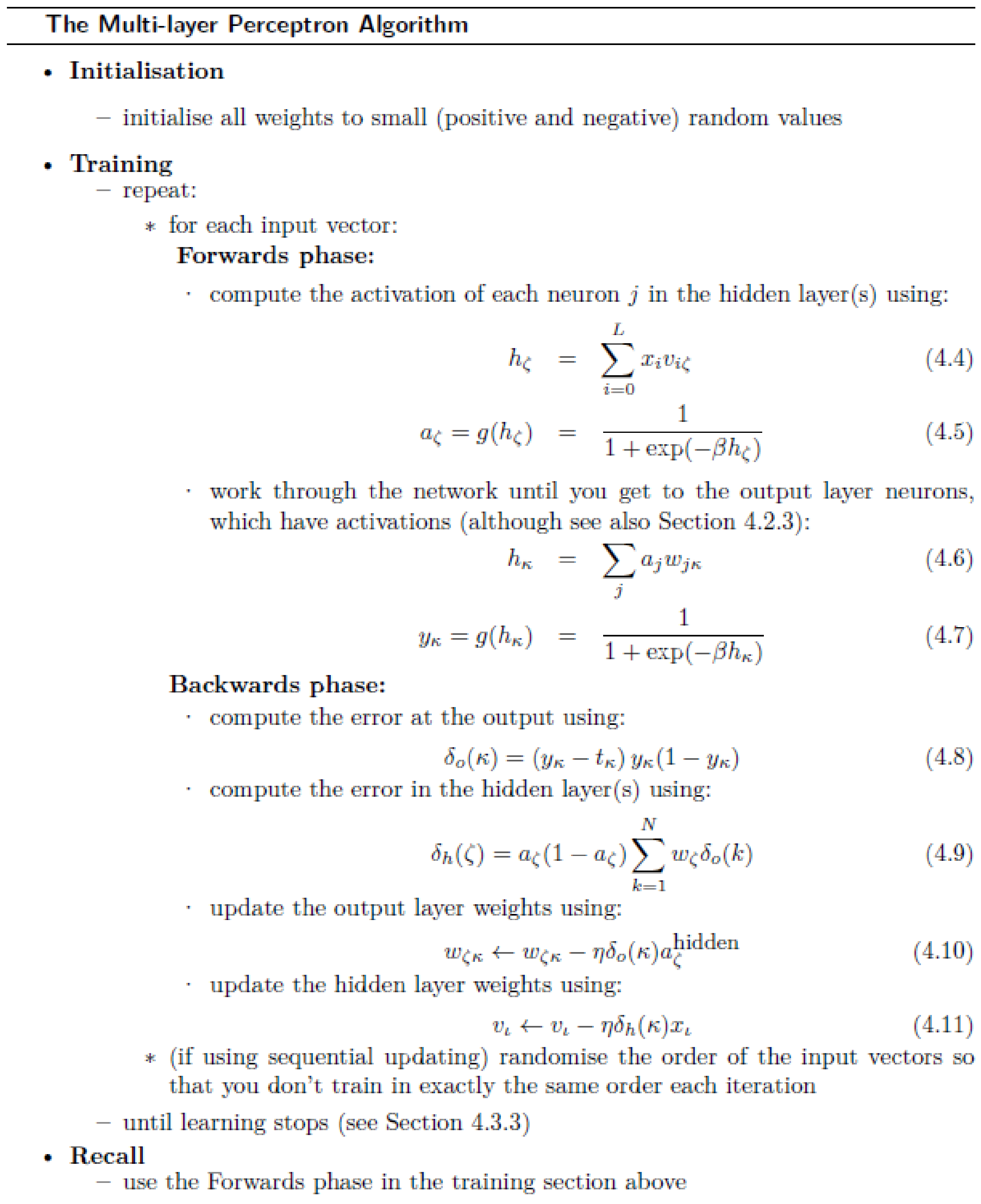
Suggested Solution

a) We see from the graph that P(1 | x=0) = 0.2 and P(0 | x=0) = 0.8.

b) You can never be 100% sure with a logistic regression model,

c) This is normally done by choosing the class 1 if P(1 | x) > 0.5. We see from the graph that this classify 4 spams correctly and 3 spams incorrectly and 5 no-spams correctly and 2 incorrectly. Altogether 9 out of 14 are classified correctly, yielding and accuracy of 9/14.

d) We could either say that the goal is to get a good precision for spam, or a good recall for no-spam. We achieve this by raising the threshold from 0.5. For the training data, a threshold of 0.7 would suffice. If we want to be prepared for more variation in the test data, we could set the threshold even higher.



The Multi-layer Perceptron Algorithm

```
• Initialisation
    – initialise all weights to small (positive and negative) random values
• Training
    – repeat:
        ∗ for each input vector:
          Forwards phase:
            · compute the activation of each neuron $j$ in the hidden layer(s) using:
                  $h_\zeta = \sum_{i=0}^{L} x_i v_{i\zeta}$                                  (4.4)
                  $a_\zeta = g(h_\zeta) = \frac{1}{1+\exp(-\beta h_\zeta)}$                 (4.5)
            · work through the network until you get to the output layer neurons,
              which have activations (although see also Section 4.2.3):
                  $h_\kappa = \sum_{j} a_j w_{j\kappa}$                                      (4.6)
                  $y_\kappa = g(h_\kappa) = \frac{1}{1+\exp(-\beta h_\kappa)}$              (4.7)
          Backwards phase:
            · compute the error at the output using:
                  $\delta_o(\kappa) = (y_\kappa - t_\kappa)\, y_\kappa(1 - y_\kappa)$        (4.8)
            · compute the error in the hidden layer(s) using:
                  $\delta_h(\zeta) = a_\zeta(1 - a_\zeta) \sum_{k=1}^{N} w_\zeta \delta_o(k)$    (4.9)
            · update the output layer weights using:
                  $w_{\zeta\kappa} \leftarrow w_{\zeta\kappa} - \eta \delta_o(\kappa) a_\zeta^{\text{hidden}}$    (4.10)
            · update the hidden layer weights using:
                  $v_\iota \leftarrow v_\iota - \eta \delta_h(\kappa) x_\iota$               (4.11)
        ∗ (if using sequential updating) randomise the order of the input vectors so
          that you don't train in exactly the same order each iteration
    – until learning stops (see Section 4.3.3)
• Recall
    – use the Forwards phase in the training section above
```

Figure 1

MLP and Back-propagation (10p)

a) Figure 1 shows the Multi-layer Perceptron Algorithm as presented by Marsland in the course book. As presented, this is an algorithm for classification. Suppose you instead will use an MLP for regression. Which lines in the algorithm do you have to change? What are their new forms? (5p)

b) Which activation function does Marsland's algorithm apply in the hidden layer? Suppose you instead will use the RELU activation function in the hidden layer. Which lines do you have to change and what will they look like with RELU? (5p)

Suggested solutions

a) Equations (4.7) and (4.8)

New forms
4.7) $y_k = g(h_k) = h_k$

4.8) $\delta_o(k) = (y_k - t_k)$

b) Marsland uses the logistic (sigmoid) activation function.

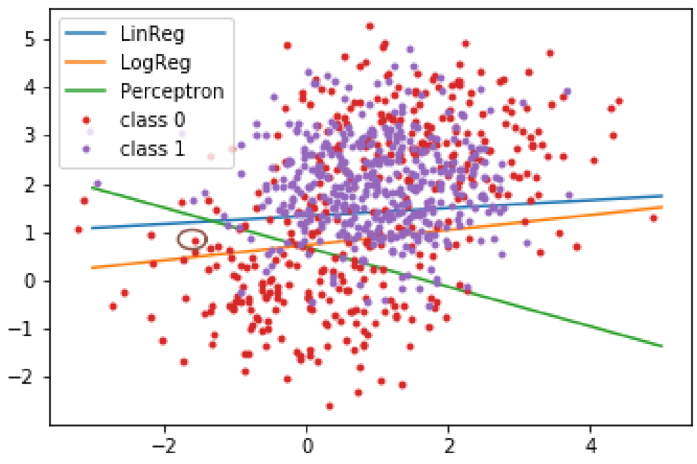
Equations (4.5) and (4.9)

New forms
4.5) $a_\zeta = g(h_\zeta) = RELU(h_\zeta) = \max(h_\zeta, 0)$

4.9) $\delta_h(\zeta) = integer(a_\zeta \geq 0) \sum_{k=1}^{N} w_\zeta \delta_0(k)$

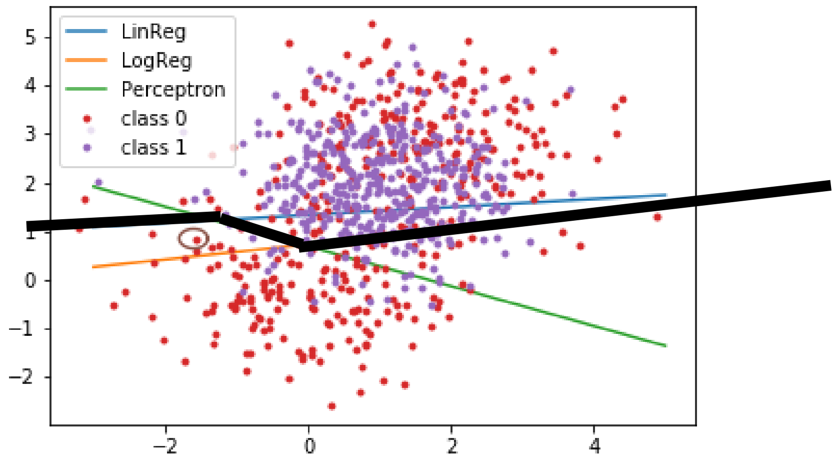
Majority Voting Classifier (8 p)

We have trained three different classifiers on the same training data (from mandatory assignment 2); a linear regression classifier, a logistic regression classifier, and a perceptron classifier. We have plotted the decision boundaries for all three classifiers on the training data in the figure. They all classify the points above their boundaries as class 1 (purple) and the points below the boundary as class 0 (red). By referring to the figure and the circled point, explain how a majority voting classifier works



Suggested Solution

A voting classifier consists of several different classifiers trained on the same problem and (possibly subsets of) the same data. To make a prediction, the voting classifier applies all the classifiers on the datapoint and let them each predict a class; collect the predictions and choose the majority class. In the figure, the circled red point will be classified as red, since it is classified as red by two of the three classifiers (the perceptron and the linear regression) even though it is classified as purple by the logistic regression classifier. In this case, the decision boundary will look as the thick, black lines.



Master's students only: Regularization (8p)

In several forms of supervised machine learning we find a model with a set of weights. The goal of training is to find the weights which best fit the training data $(\boldsymbol{X}, \boldsymbol{t})$. To determin what we mean by best fit, we introduce a loss function, L, and the goal is to determine the weights which minimize the loss, in symbols $\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmax}}(-L(\boldsymbol{X}, \boldsymbol{t}, \boldsymbol{w}))$

In reguarization, one replaces the objective $-L(\boldsymbol{X}, \boldsymbol{t}, \boldsymbol{w})$ with another objective, e.g., for L2-regularization one replaces it with $-L(\boldsymbol{X}, \boldsymbol{t}, \boldsymbol{w}) - \alpha||\boldsymbol{w}||^2$.

a) Why does one apply regularization, and what is achieved by regularization?

b) How is α determined?

Suggested Solutions

a) When training towards the original learning objective, there is a danger for overfitting, which means that the trained model fits the training set very well, but it does not generalize equally well to other data. In particular, some features might be particular important and by repeated applications of gradient descent, the model ascribes more weight to these features. The goal of regularization is to avoid putting too much weight on some features, and instead try to get more even weights between the features.

b) Alpha determines how much larger weights get punished. There is no fixed α which fits all cases. The optimal α depends on the task in question. α is tuned experimentally. One separates the development data into a training set and a development test set. For various values of α, one trains models on the training set and evaluates on the development test set. The value of α which gives the best score on the development test set is chosen for the final model.

Bachelor students only: Training and test sets (8p)

Describe what is meant by a training set, test set and development test set in supervised machine learning, and how they are used.

Suggested Solution

In machine learning, one applies algorithms that improve automatically through experience. The algorithms are trained by the help of data, a training set. To see that the algorithms have learned, one needs a precise description of what the learning objective is, and ways to measure improvement with respect to this objective. To measure the improvement and how well the system performs, one uses a test set of relevant data. The test set must be disjoint from the training set.

Often, one would compare different ML systems trained on the same data. For this, one needs a development test set different from both the training set and the test set. One may train repeatedly different systems on the same training set and test them on the development test set. One then decides on the system which performs best on the development test set. After deciding on the best system, this can be tested on the (final) test set, which hasn't been considered so far.

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)

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)

c) (**All students)** 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)

Suggested solutions

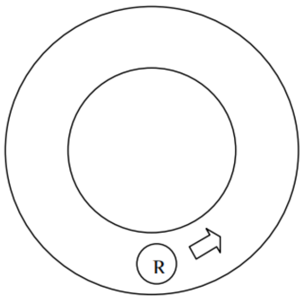
a) PCA can not be learn high-dimensional representations: intuitively, we can not find more orthogonal dimensions that the ones provided; formally, the number of eigenvectors is limited by the dimensionality. Autoencoders can learn higher-dimensional representations, although the loss function may have to change.

b) 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.

c) 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.



a) 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. 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

(6 points, 2 per sub-task)

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 textbook and lectures. (7 points)

Suggested solutions

a1) Genetic representation (genotype): Since we are representing the weights of a neural network, the genotype needs to encode several numbers, that can be mapped to the neural network connections. The most straightforward way is to define each genotype as a list of floating-point values, where each value represents the weight of a single specific network connection.

a2) Variation operators: 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. Other operators applicable to the representation are also accepted.

a3) 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.

b) Since this robot control problem is continuous, rather than discrete, there is a potentially infinite number of different states and actions. We therefore need to **discretize states and actions** before modelling this problem in the traditional RL way. For instance, we could model the problem this way:

States: States need to include information about **distance to walls**. To guide the movements of the robot, we should also know on which side of the robot the wall is. There are many ways to represent this information. One example is to represent each state as two variables, one of which represents the **direction** towards the wall (dir), and the other the **distance** to it (dist). To guide actions, we need to discretize these states, for instance into the sets dir (left, front, behind, right) and dist (close, medium, far).

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

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)

b) 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)

c) 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)

Suggested solutions

a) This question is very subjective and there are many possible valid solutions. You can for example set the initial velocity of every particle to a very high number. You can also add a repulsion function that makes particles repel one another. Thereby particles are gathered around a 'center of mass' but cannot squeeze on the same position. Related to how 'boids' flock which was mentioned in the lecture.

b) h := jjh := hhjjh := jjhjjhhhjjh

c) '+' 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.