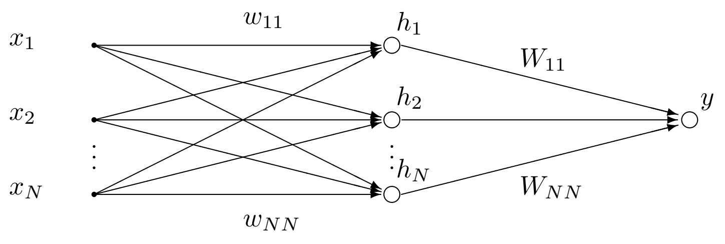
Neural Networks and Learning Systems TBMI26 / 732A55

Computer Assignments

2022



Neural Networks and Learning Systems
Computer Assignment Collection

Contents

General Information

This compendium contains instructions for the 4 computer assignments. Additional instructions and resources for assignment 2 is published separately on the LISAM course page (*lisam.liu.se*).

You have to pass all assignments to get the lab credits. The assignments will be graded based on your reports and submitted code. Templates for the reports and all data needed for the assignments are found in the zip file available on LISAM. No more than two persons may cooperate on a single report. Reports are submitted in LISAM and will be checked for plagiarism using Urkund. Don't forget to include both group members in the report *and* the LISAM submission page!

The deadlines for the reports can be found in LISAM. If a report is sent in late we might not look at it until the next re-examination period. *So, submit your reports in time*!

Important: This year, all four assignments are done in Python for the first time. In case you discover any technical problems with the provided code, we would be grateful if you notified us as soon as possible so the issue can be fixed.

Common setup: All four assignments are written in python notebooks. This is a file type used to do interactive coding in python. We recommend that you use Jupyter Lab to run the notebooks, and that you install the Variable Inspector extension. This will give you a list of all current variables and their values, which will make debugging much more convenient. Once you are up and running with your first notebook, you can open the Variable Inspector by right-clicking anywhere in the notebook window and selecting "Open Variable Inspector". In the same way, you can also open a python console for the notebook, which can be useful when testing your code.

Assignments

1. Supervised learning: kNN and Backpropagation

We start by exploring different supervised learning methods. To get some practical experience with two of the most common methods, you will implement the kNN algorithm and two different backpropagation neural networks.

Your task is to implement the algorithms and evaluate them on different datasets. While doing so you will learn how the hyperparameters of the methods influence the results.

The data

Each dataset is divided into three parts. The matrix X with data samples, the vector L with class labels and the matrix D with the desired output of the neural networks.

For this assignment you will work with four different datasets. The first three consists of 2D data with 2 or 3 classes. These are good for trying out the classifiers while implementing your code. You can also use these to test linear and non-linear properties. The last dataset is for optical character recognition (OCR) and contains 8×8 pixel images of handwritten digits, i.e. 10 classes and 64 features. See http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits.

The code

We have provided you with some code to help you get started. The code and data can be downloaded from the course LISAM page. You will write your code in the provided python notebooks (*.ipynb*), and in the files *classifiers.py* and *utils.py*.

The following is a list of all files and functions in this assignment:

Python notebooks	
main_kNN	Main notebook for the kNN task
main_SingleLayer	Main notebook for the single-layer neural network task
main_MultiLayer	Main notebook for the multi-layer neural network task
Functions in classifiers.py	
kNN	Your implementation of the kNN algorithm
runSingleLayer	Your implementation of the forward pass on the single-layer network
trainSingleLayer	Your implementation of the training of the single-layer network
runMultiLayer	Your implementation of the forward pass on the multi-layer network
trainMultiLayer	Your implementation of the training of the multi-layer network
Functions in utils.py	
calcConfusionMatrix	Your function for calculating the confusion matrix of predicted results
calcAccuracy	Your function for calculating the accuracy of predicted results
Helper functions in utils.py	
loadDataSet	Load the datasets
plotCase	Plots a simple view of the first 3 datasets
plotData	Plots data and predictions for the first 3 datasets
plotResultsDots	Plots data and prediction fields for the first 3 datasets
plotResultsOCR	Plots data and predictions for the 4th dataset
selectTrainingSamples	Used to split data into separate bins
tanhprim	Derivative of tanh

Note that you don't have to directly use all of these functions. The helper functions are used by the parts of the skeleton code which you don't have to modify.

1.1. kNN

We will start of with the rather straightforward kNN classifier. To help you load your data and evaluate the performance of your implementation you will use *main_kNN.ipynb*. Once you have familiarized yourselves with the evaluation code you can start to implement the algorithm by filling in the *kNN* function found in the file *classifiers.py*. Furthermore, to evaluate the performance and to identify any weaknesses of your classifier, you will calculate the confusion matrix and the accuracy after each training.

kNN implementation tasks

- Read and understand *main_kNN.ipynb*
- Implement the kNN algorithm in *classifiers.py*
- Implement *calcConfusionMatrix* in *utils.py*
- Implement *calcAccuracy* in *utils.py*

kNN classification task
Once you have a stable kNN classifier implementation it is time to find the optimal k for each dataset. Do this using cross validation.

Required accuracy for each dataset using kNN

1: 98%
2: 99%
3: 99%
4: 96% (OCR)

1.2. Backpropagation Networks

Now, lets move on to neural networks. You will implement two types. First a linear classifier based on a single layer which will take an input X and directly calculate an output. Second, a multi-layer network with one hidden layer.

There are many ways of coding the output of neural networks. In this assignment you will use a voting scheme. The single layer, which is actually the last layer of the multi layer network, will have one output for each class, i.e. one neuron for each class. The output-neuron that gives the strongest output wins and sets the class.

Backprop Implementation Tasks
The code is implemented in a similar way as kNN evaluation. Start with the single layer network since this is the easiest to implement. Do not forget to handle the bias weights in a proper way, especially for the two layers of the multi-layer network.

Single Layer Implementation Tasks

- Read and understand *main_SingleLayer.ipynb*
- We have supplied you with the D matrices, these are used to train a certain behaviour that can be translated to a class. Look at this matrix in combination with the labels in L. Figure out how D enforces a network were the class is given by the neuron with the strongest output.
- Fill in the code for adding the bias.
- Fill in the code that initialize the parameters in W. If you get stuck, think about how many neurons you need for the dataset and how many features there is.
- Implement *runSingleLayer.m*
- Implement *trainSingleLayer.m*

Multi Layer Implementation Tasks

- Read and understand *main_MultiLayer.m*
- Fill in the code that initialize the parameters in W and V
- Implement *runMultiLayer.m*
- Implement *trainMultiLayer.m*

Backprop Classification Tasks:
Train networks that can consistently produce the test-data accuracies specified below, when using all the data available to you. You are expected to use different parameter settings for each dataset, i.e. different

combinations of number of hidden, learning rate and number of iterations. Note that some of the more advanced dataset may require a long training time.

Required accuracy for each dataset using multi-layer neural network

1: 98%
2: 99%
3: 99%
4: 96% (OCR)

Create a non-generalizable solution
Last but not least we want you to use the third dataset and create an example that illustrate a non-generalizable solution by reducing the number of training samples and adjusting the parameters accordingly. In other words, create a scenario in which you get high training accuracy, but low test accuracy, and explain why the solution fails to generalize.

1.3. Report

In order to pass the assignment you will need to answer the following questions and upload the document to LISAM. Please upload the document in PDF format. There is a template for the report together with the lab files. **You will also need to upload all code in .m-file format.** We will correct the reports continuously so feel free to send them as soon as possible. If you meet the deadline you will have the assignments reported in LADOK together with the exam. If not, they will be reported during the re-exam period.

Q1. Give an overview of the data from a machine learning perspective. Consider if you need linear or non-linear classifiers etc.
Q2. Explain why the down sampling of the OCR data (done as pre-processing) result in a more robust feature representation. See http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits
Q3. Give a short summary of how you implemented the kNN algorithm.
Q4. Explain how you handle draws in kNN, e.g. with two classes (k = 2)?
Q5. Explain how you selected the best k for each dataset using cross validation. Include the accuracy and images of your results for each dataset.
Q6. Give a short summary of your backprop network implementations (single + multi). You do not need to derive the update rules.
Q7. Present the results from the backprop training and how you reached the accuracy criteria for each dataset. Motivate your choice of network for each dataset. Explain how you selected good values for the learning rate, iterations and number of hidden neurons. Include images of your best result for each dataset, including parameters etc.
Q8. Present the results, including images, of your example of a non-generalizable backprop solution. Explain why this example is non-generalizable.
Q9. Give a final discussion and conclusion where you explain the differences between the performances of the different classifiers. Pros and cons etc.
Q10. Do you think there is something that can improve the results? Pre-processing, algorithm-wise etc.

Good luck, have fun and do not forget to save images and numbers that you may need for the report.

2. Deep learning

In this assignment you will implement a convolutional neural network (CNN) for image classification using Keras, a network development API. To gain familiarity with the tools and methods you will use during this assignment, you have at your disposal an introductory jupyter notebook. Here, you will find information about how to navigate the notebook and a description of Keras and its functionality. Make sure to go through the introductory notebook beforehand!

Given the computationally demanding nature of training CNNs, each group has been allocated a virtual compute unit in Microsoft Azure. Instructions on how to access this resource and use it for the assignment can be found in **Instructions_Azure.pdf** in the LISAM course room.

The data

The introductory jupyter notebook makes use of a higher resolution version of the MNIST dataset which you are already familiar with. The actual assignment, instead, uses the CIFAR10 dataset (more information in the assignment notebook).

2.1. Report

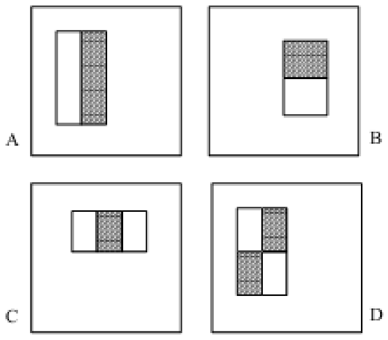
In order to pass the assignment you will have to fill in the notebook with your code and answer all the related questions in the report. *Save the jupyter notebook with outputs as an HTML-file and submit it along with the report in LISAM*. We will correct the reports continuously so feel free to send them as soon as possible. If you meet the deadline you will have the assignments reported in LADOK together with the exam. If not, they will be reported during the re-exam period.

- Q1. The shape of **X_train** and **X_test** has 4 values. What do each of these represent?
- Q2. Train a Fully Connected model that achieves above 45% accuracy on the test data. Provide a short description of your model and show the evaluation image.
- Q3. Compare the model from Q2 to the one you used for the MNIST dataset in the first assignment, in terms of size and test accuracy. Why do you think this dataset is much harder to classify than the MNIST handwritten digits?
- Q4. Train a CNN model that achieves at least 62% test accuracy. Provide a short description of your model and show the evaluation image.
- Q5. Compare the CNN model with the previous Fully Connected model. You should find that the CNN is much more efficient, i.e. achieves higher accuracy with fewer parameters. Explain in your own words how this is possible.
- Q6. Train the CNN-model with added Dropout layers. Describe your changes and show the evaluation image.
- Q7. Compare the models from Q4 and Q6 in terms of the training accuracy, validation accuracy, and test accuracy. Explain the similarities and differences (remember that the only difference between the models should be the addition of Dropout layers). Hint: what does the dropout layer do at test time?
- Q8. Train the CNN model with added BatchNorm layers and show the evaluation image.
- Q9. When using BatchNorm one must take care to select a good minibatch size. Describe what problems might arise if the wrong minibatch size is used. You can reason about this given the description of BatchNorm in the Notebook, or you can search for the information in other sources. Do not forget to provide links to the sources if you do!
- Q10. Design and train a model that achieves at least 75% test accuracy in at most 25 epochs. Explain your model and motivate the design choices you have made and show the evaluation image.

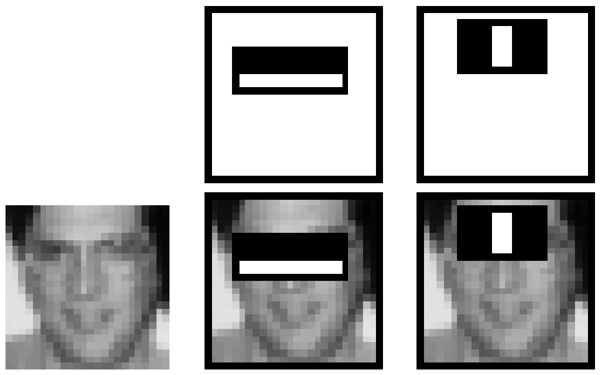
Good luck and have fun!

3. Boosting

In this assignment you will implement the AdaBoost algorithm for face detection in images. The AdaBoost algorithm sequentially trains a number of so-called weak classifiers that are combined into one strong classifier. In a seminal article by Viola and Jones, *Robust Real-time Object Detection*, it is shown how boosting based on simple so called Haar-features can create very powerful detection algorithms for computer vision, for example to detect faces in images.



(a) Four types of Haar-feature masks.



(b) Simple Haar features measuring contrasts at different positions in an image box have proven to be efficient for learning to detect faces and other objects in images. Images taken from "Robust Real-time Object Detection" by P. Viola and M. Jones

Your task is to implement the standard AdaBoost algorithm to recognize face images. A large number of face images and non-face images are available in the supplementary material. All images are 24x24 pixels large and the non-face images are random image patches from photos found on the internet. Your classifier should be trained to distinguish images of a face from other images using Haar features. To kick-start your work, we have provided some skeleton code and a few functions.

This is a list of the files and functions in this assignment:

Python notebook	
AdaBoost	Main notebook where you should implement your algorithm
Functions in classifier.py	
WeakClassifier	Your implementation of a weak classifier using a decision stump
WeakClassifierError	Your implementation to get the error of a weak classifier
Helper functions in utils.py	
GenerateHaarFeatureMasks	Function that generates random Haar feature masks
ExtractHaarFeatures	Function that extracts the Haar feature values from images

The two helper functions generate the Haar filter masks, as shown in the figure above, and apply them to the images to calculate the features. A set of randomized Haar features will be used in this work. It is recommended to start with a relatively low number of features to keep the computational time low while developing your algorithm. Start by opening the notebook and read the comments. You will also have to implement the functions *WeakClassifier* and *WeakClassifierError*. Each weak classifier in the final trained AdaBoost classifier should be a thresholding of a well-selected Haar-feature x_i at a well selected threshold t, i.e., $x_i > t$ or $x_i < t$. This can be written $px_i > pt$ for polarity $p = 1$ or $p = -1$ respectively.

The implementation should include the following parts:

- Training the AdaBoost algorithm with a "smart" choice of number of training data and Haar-features.
- Determine the best number of weak classifiers based on the number of training data and number of Haar-features used.
- Apply the final strong classifier on the test data, i.e. the rest of the data not used for training. At least half of the available data must be used for testing the AdaBoost performance. (You should not train the system again, only apply the strong classifier on the test data.)

- Once done with the implementation, optimize the parameters such that the accuracy is stable and greater than 93% on the test data.

3.1. Report

In order to pass the assignment you will need to answer the following questions and upload the document to LISAM. Please upload the document in PDF format. There is a template for the report together with the lab files. **You will also need to upload all code in .m-file format.** We will correct the reports continuously so feel free to send them as soon as possible. If you meet the deadline you will have the assignments reported in LADOK together with the exam. If not, they will be reported during the re-exam period.

Q1. Plot how the classification accuracy on training data and test data depend on the number of weak classifiers (in the same plot). Be sure to include the number of training data (non-faces + faces), test-data (non-faces + faces), and the number of Haar-Features.

Q2. How many weak classifiers did you use when training? How many of them did you use for the final strong classifier? Motivate your choices.

Q3. What is the accuracy on the training data and test data after applying the optimized strong classifier? Discuss your choice of hyperparameters and how they influence the accuracies.

Q4. Plot the Haar-features selected by your classifier (one for each weak classifier). If you have many weak classifiers, select some representative subset. Can you think of why they would be useful for classifying faces?

Q5. Plot some of the misclassified faces and non-faces that seem hard to classify correctly. Why do you think they are difficult to classify?

Q6. Are your results reasonable? Can you think of any way to improve the results?

Q7. Can we expect perfect results? Motivate your answer.

Good luck, have fun and do not forget to save images and numbers that you may need for the report.

4. Reinforcement Learning

This assignment deals with reinforcement learning, i.e. a system that learns from a scalar feedback value. The goal of the system is to maximize the reward over time. The result can be interpreted as a map describing expected feedback (a sum or a weighted sum of future feedback) given that the system is in a certain state and performs a certain action. This may be more similar than anything else in the course to what we usually mean with learning, i.e. learning a behaviour in order to solve a task as good as possible.

The aim of this assignment is to give you a better understanding of how reinforcement learning works in practice and how it can be used to solve easy tasks. The method that you will implement is called Q-learning, a robust method that is also relatively easy to implement.

Gridworld – The fastest way

In a world consisting of a grid, the task is to teach a small robot different behaviours using reinforcement learning. The different squares represent states that naturally correspond to the position of the robot. Given a certain state, the robot can choose to move left, right, up or down. The robot's objective is to make its way as efficiently as possible from a arbitrary starting point to a given goal, i.e. find the goal and try to maximize the sum of the feedback under-way. It can choose to explore new parts of the world, or pursue old accustomed patterns.

You can control the robot and draw its position with the help of a set of Matlab functions. Your task is to write a program for Q-learning that given output data from the robot, plans the next action and updates the Q-function.

Example experiment

A rat is dropped into a water tank, while somewhere in the tank there is a small platform hidden precisely under the water surface. Rats are small animals that lose body heat fast since their body surface is relatively large compared to their body volume. Therefore the rats are motivated to find the platform and get out of the cold water. This experiment is subsequently repeated, several times, in order to see if the rat learns to immediately find the platform. The rat is assumed to approximately know its position in the tank by looking at the surrounding environment. The number of successful trials a rat needs before it swims the shortest way to the platform is very low, under 10. Our robot is not as bright, as you will notice.

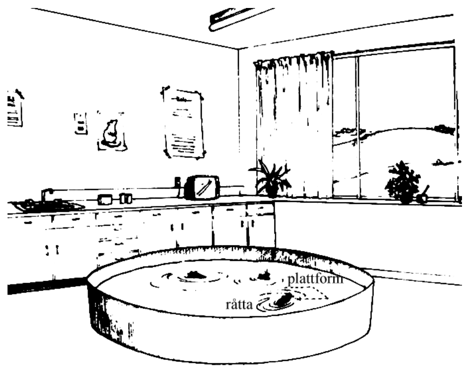


Figure 2: Diagram of the experiment (Adapted from: Fundamentals of Human Neuropsychology by B. Kolb and I. Q. Whishaw)

Your task

Implement the Q-learning algorithm that learns the optimal policy for moving the robot to the goal in the different worlds. The following worlds should be explored:

W1. "Irritating blob", a static world that is good for exploration and to test your algorithm. Consider how different choices of exploration strategy affect the convergence rate of your solution. Also, try different initializations of the Q-function. Is it better, for instance, to use noise instead of a constant when initializing it?

W2. ”Suddenly irritating blob”, a variation of the previous world where some randomness has been introduced. How does the robot behave now?

W3. ”The road to HG”, experiment giving different values to the parameters. Notice that the start position is fixed. How can you get a good policy from every state in this world?

W4. ”The road home from HG”. This is the most important of all the worlds, it illustrates a central property of Q-learning, which? Try playing around with γ (discount factor) and η (learning rate). You will need a large number (maybe several thousands) of iteration to converge.

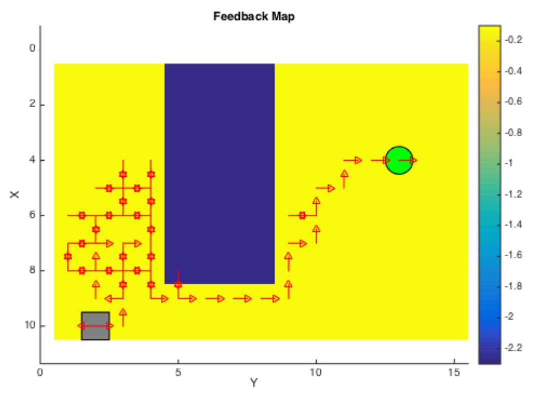


Figure 3: The figure shows a world from the assignment. The robot is symbolized by the grey square, the goal by the green circle. A bright background marks a small negative feedback while a dark background marks a large negative feedback. The arrows show one possible path to be explored by the robot.

As a suggestion you should start with a ϵ-strategy in order to deliberate between performing an optimal action given the current Q-function, or taking a step in another direction in order to explore more of the state space. This means you should with a probability of $1 - \epsilon$ choose an optimal action according to the current Q-function, and with the probability ϵ choose an arbitrary (other) action. Use the function $gwinit(k)$, where k is the number of the world, each time you have reached the goal in order to re-initiate the robot to a new random position. Other available functions are documented in the end of this instruction.

4.1. Report

In order to pass the assignment you will need to answer the following questions and upload the document to LISAM. Please upload the document in PDF format. There is a template for the report together with the lab files. **You will also need to upload all code in .m-file format.** We will correct the reports continuously so feel free to send them as soon as possible. If you meet the deadline you will have the assignments reported in LADOK together with the exam. If not, they will be reported during the re-exam period.

Q1. Define the V- and Q-function given an optimal policy. Use equations and describe what they represent. (See lectures/classes)

Q2. Define a learning rule (equation) for the Q-function and describe how it works. (Theory, see lectures/classes)

Q3. Briefly describe your implementation, especially how you hinder the robot from exiting through the borders of a world.

Q4. Describe World 1. What is the goal of the reinforcement learning in this world? What parameters did you use to solve this world? Plot the policy and the V-function.

Q5. Describe World 2. What is the goal of the reinforcement learning in this world? This world has a hidden trick. Describe the trick and why this can be solved with reinforcement learning. What parameters did you use to solve this world? Plot the policy and the V-function.

Q6. Describe World 3. What is the goal of the reinforcement learning in this world? Is it possible to get a good policy from every state in this world, and if so how? What parameters did you use to solve this world? Plot the policy and the V-function.

Q7. Describe World 4. What is the goal of the reinforcement learning in this world? This world has a hidden trick. How is it different from world 3, and why can this be solved using reinforcement learning? What parameters did you use to solve this world? Plot the policy and the V-function.

Q8. Explain how the learning rate α influences the policy and V-function. Use figures to make your point.

Q9. Explain how the discount factor γ influences the policy and V-function. Use figures to make your point.

Q10. Explain how the exploration rate ε influences the policy and V-function. Use figures to make your point. Did you use any strategy for changing the exploration rate during training?

Q11. What would happen if we instead of reinforcement learning were to use Dijkstra's cheapest path finding algorithm in the "Suddenly irritating blob" world? What about in the static "Irritating blob" world?

Q12. Can you think of any application where reinforcement learning could be of practical use?

Good luck, have fun and do not forget to save images and numbers that you may need for the report.

4.2. The code

Your code will interact with the gridworld using the *World* class implemented in *world.py*. This class provides a set of methods that you can use to initialize the gridworld and agent position, to execute actions, and to draw the current state. In addition to the main algorithm, which should be implmented in the notebook, you will also implement some additional function in the file *utils.py*.

Here is a quick description of all files and functions in this assignment:

Python notebook	
Qlearning	Main file where you should implement your training algorithm
Methods in the World class	
constructor (init)	Initializes the gridworld
action	Perform an action with the agent, and return if the action was valid as well as the reward
draw	Draw the world state
Functions in utils.py	
getpolicy	Your implementation to get the movement policy from your Q-table
getvalue	Your implementation to get the V-function from your Q-table
Helper function in utils.py	
plotarrows	Draw a movement policy on the world map as a set of arrows (used by the draw method in the World class)

A good way to get started with the assignment is to test by manually driving the robot around. For example, try the following commands:

```
W = World(1)
W.draw()
W.action(1)
W.draw()
W.action(4)
W.draw()
```

4.3. Hints

- For some worlds, the convergence of the Q-function will take a long time; you may have to wait many minutes before it is finished. If you don't wait the result may be hard to interpret, so don't be afraid to run your algorithm for a long time if you get strange results.

- In order to get max or min along some dimension in a multidimensional array, use the `axis` input parameter of the numpy functions `np.min(...)` and `np.max(...)`. This will work along, and therefore collapse, the specified dimension (axis) of the data array.
- Initializing the Q-matrix to only negative values, then manually setting the Q-values for all actions in the goal state to 0 can speed up the training process, since this means that the goal is the highest value in the grid world by default.
- Several positions in the Q-matrix will not be valid actions, for example standing at the top of the world and taking a step ”up”. It can therefore be appropriate to set the Q-value for these positions/actions to minus infinity (-Inf), so that they don’t affect the result of some max-operation. Note however that depending on your implementation this trick might not work well in all worlds.
- A way to represent the Q-function in the initial world is $Q(yposition, xposition, action)$, where Q is a multidimensional array. You can initialize such an array using numpy commands like `np.ones(...)`, `np.zeros(...)`, and `np.random.rand(...)`.
- An exploration strategy that simply chooses a random direction for robot motion is not as dumb as it may seem. It could be an easy way to get started before implementing the ϵ-strategy.
- When you think that you have learned an optimal policy, you should implement a test-loop where you draw each step of the robots path towards the goal (i.e. watch the robot walk according the policy). In this loop you should not explore, so ϵ should be 0 and the optimal action will always be chosen. Using this test will be very valuable to understand some of the worlds.