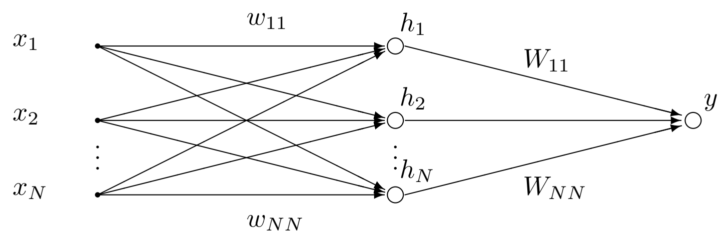
Neural Networks and Learning Systems
TBMI26 / 732A55

Computer Assigments

2020



Neural Networks and Learning Systems
Computer Assignment Collection

Contents

General Information

This compendium contains instructions for Assignment 1, 2 and 4. Instructions for assignment 3 is published separately on the LISAM course page. You have to pass all assignments to get the lab credits.
The assignments will be graded based on your reports. Templates for the reports are found in the .zip file together will all data needed for the assignments (LISAM). No more than two persons may cooperate on a single report. Reports are sent in to LISAM (*lisam.liu.se*). 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 will not look at it until the next re-examination period. *So, send in your reports in time*!

Assignments

1. Supervised learning: kNN and Backpropagation

We start of the course by exploring different supervised learning methods. And to get some practical experience with two of the most common methods you will implement the kNN algorithm and backpropagation neural networks.
Your task is to implement the algorithms and evaluate them on different datasets. And, whilst doing so you will learn how the parameters of the methods influences the results.

The Data

All data is divided into three parts. The matrix X with all data samples, the vector L with all class labels and the matrix D with the desired output of the neural networks.
For this assignment you will work with 4 different datasets. The first three consists of 2D data with 2 to 3 classes. These are good for trying out your classifiers while writing the code. You can also use these to test linear and non-linear properties. The last dataset consists of OCR digits so there are 10 classes with 64 features. Include images of your best result for each dataset.
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 together with the data can be downloaded from the course LISAM page. Once you have downloaded and unpacked the code, start the assignment by running the following script in Matlab:

```
>> run setupSupervisedLab.m
```

By running this script you will setup the paths Matlab need in order to find the other files and functions. In the folder you will also find folders for the data and some help functions as well as a folder for each classifier you will implement.

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

setupSupervisedLab	Script for setting up the lab environment (run this first)
main_kNN	Main script for the kNN task
main_SingleLayer	Main script for the single-layer neural network task
main_MultiLayer	Main script for the multi-layer neural network task
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
calcConfusionMatrix	Your function for calculating the confusion matrix of predicted results
calcAccuracy	Your function for calculating the accuracy of predicted results
combineBins	Used to combine cell arrays to a single matrix, useful for cross validation
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. Some of them 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.m*. Once you have familiarized yourselves with the evaluation code you can start to implement the algorithm by filling in the *kNN.m* function found in the *kNN* folder. Furthermore, to evaluate the performance and to identify any weaknesses of our classifiers you will calculate the confusion matrix and the accuracy after each training.

kNN Implementation Tasks:

- Read and understand *main_kNN.m*
- Implement the kNN algorithm in *kNN/kNN.m*
- Implement *calcConfusionMatrix.m*
- Implement *calcAccuracy.m*

kNN Classification Task:

Once you have a stable kNN classifier implementation its time to find the optimal k for each dataset. Do this using cross validation. You have to create your own script for this, but you can borrow most of the code form the previous script.

Accuracy limits for dataset Nr.:

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 a single 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, i.e. maximal, output wins and sets the class.

Backprop Implementation Tasks:

The code is implemented in a similar way as kNN evaluation. Start of with the single layer network as 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.m*
- 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.

Accuracy limits for dataset Nr.:
1: 98 %
2: 99 %
3: 99 %
4: 96 % (OCR)

Create a non-genaraslisable solution Last but not least we want you to use the third dataset and create an example that illustrate a non-generalisable solution by reducing the number of training samples and adjusting the parameters accordingly.

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 lab part of the course reported in LADOK together with the exam. If not, you'll get the lab part 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. 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 Haar-features can create very powerful detection algorithms for computer vision, for example to detect faces in images.

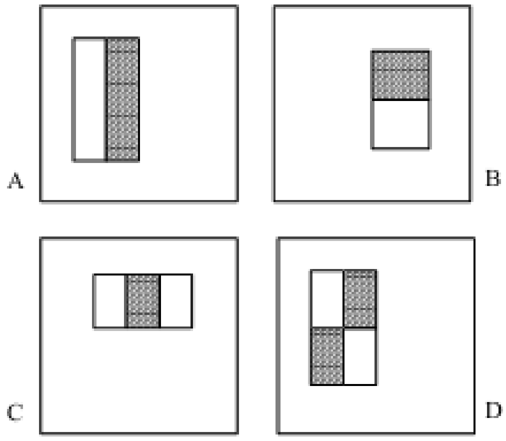


Figure 1: Four types of Haar-feature masks.

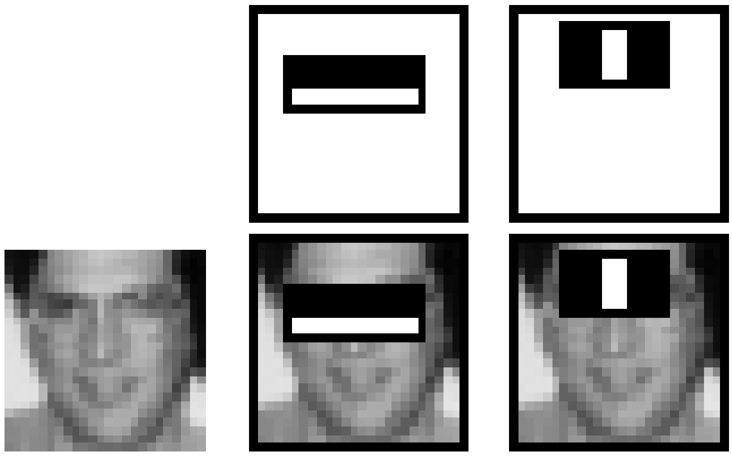


Figure 2: Simple Haar features measuring contrasts at different positions in an image box have proven to be efficient for learning to detect faces and other objects in images. Images taken from Robust Real-time Object Detection by P. Viola and M. Jones

2.1. Your Task

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 scripts and function in this assignment:

AdaBoost	Main script where you should implement your algorithm
WeakClassifier	Your implementation of a weak classifier using a decision stump
WeakClassifierError	Your implementation to get the error of a weak classifier
GenerateHaarFeatureMasks	Function that generates random Haar feature masks
ExtractHaarFeatures	Function that extracts the Haar feature values from images

The two 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. You should start with the script called *AdaBoost* and read the comments. You will also have to implement the functions *WeakClassifier* and *WeakClassifierError*. The weak classifiers in the AdaBoost algorithm 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 0.93 on the test data.

2.2. 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 lab part of the course reported in LADOK together with the exam. If not, you'll get the lab part 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.

3. Deep Learning

The instructions for the Deep Learning Assignment will be provided separately on LISAM. Please note that questions about Assignment 1, 2 and 4 will not be answered during this session. Plan accordingly.

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.

4.1. 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.

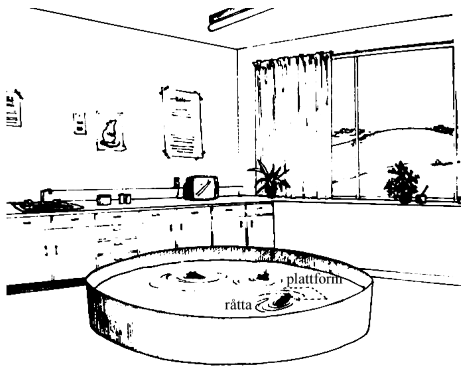


Figure 3: Diagram of the experiment (Adapted from: Fundamentals of Human Neuropsychology by B. Kolb and I. Q. Whishaw)

4.2. Your Task

Write a Matlab program using the provided commands (and the hints below) to implement a 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.

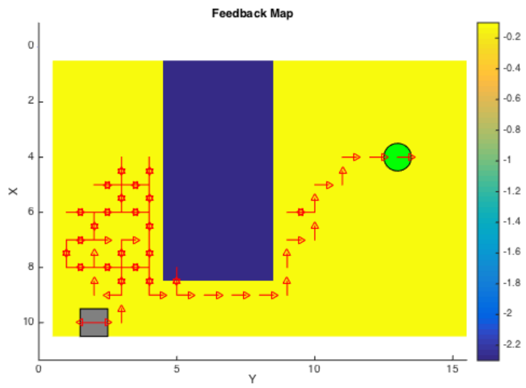


Figure 4: The figure shows a world from the assignment. The robot is symbolized by the grey square, the goal by the green circle. A bright background marks a small negative feedback while a dark background marks a large negative feedback. The arrows show one possible path to be explored by the robot.

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.

4.3. 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 lab part of the course reported in LADOK together with the exam. If not, you'll get the lab part 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 influences the policy and V-function. Use figures to make your point.

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

Q10. Explain how the exploration rate influences the policy and V-function. Use figures to make your point. Did you use any strategy for changing 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? A hint is to use the Internet.

Q13. (Optional) Try your implementation in the other available worlds 5-12. Does it work in all of them, or did you encounter any problems, and in that case how would you solve them?

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

4.4. Matlab code

Set up the required Matlab paths by running the *setupQlearning* script. There are several files required to run the lab, located in the *gwfunctions* and *helperfunctions* folders. It might be helpful to start the lab by reading the documentation for these functions. Use the *help* command in Matlab to show information about a function, for example *help gwinit*. The *gwfunctions* are used to initialize, draw, and interact with the lab environment, and the *helperfunctions* will be useful when implementing your training algorithm. There are also some skeleton files in the main folder which you must implement by yourself. The intended use of these is also documented in the function help.

Here is a quick description of all files in this lab:

Qlearning	Main file where you should implement your training algorithm
getpolicy	Your implementation to get the movement policy from your Q-table
getvalue	Your implementation to get the V-function from your Q-table
getepsilon	Your implementation to implement an exploration strategy (optional)
gwinit	Initializes the gridworld
gwstate	Get the state of the world
gwaction	Perform an action with the robot
gwdraw	Draw the world
gwdrawpolicy	Draw a movement policy on the world map as a set of arrows
gwplotarrow	Draw a single arrow at a state (used by gwdrawpolicy)
setupQlearning	Set up the lab environment (run this first)
chooseaction	Select an action based on optimal choice and exploration rate
sample	Select an item with given probabilities (used by chooseaction)

A good way to get started with the assignment is to test by manually driving the robot around.

```
>> gwinit(2);
>> gwdraw();
>> gwaction(2)

ans =
         ysize: 10
         xsize: 15
           pos: [2x1 double]
    isterminal: 0
       isvalid: 1
      feedback: -0.1000

>> gwdraw();
>> gwaction(2)

ans =
         ysize: 10
         xsize: 15
          pos: [2x1 double]
    isterminal: 0
       isvalid: 1
      feedback: -0.1000

>> gwdraw();
```

4.5. 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 Matlab commands $min(A, [], dim)$ and $max(A, [], dim)$. These commands will work along, and therefore collapse, the dimension dim.
- Remember that the Q-function has to be 0 for all actions at the goal. Remember also that several positions in the table representing the Q-function will not be valid actions, standing at the top of the world and taking a step "up" for instance. 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.

- In order to plot the Q-function you can use the command $imagesc(Q(:,:,k))$, where k is a specific action. In general, $imagesc$ can be used to plot a 2-D array. You can also plot things with the command $surf$ that works in the same way but will return a 3D-plot instead. Try both.
- A way to represent the Q-function in the initial world is $Q(yposition, xposition, action)$, where Q is a multidimensional array in matlab. You can initialize such an array using commands like $ones$, $zeros$ och $rand$.
- 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.