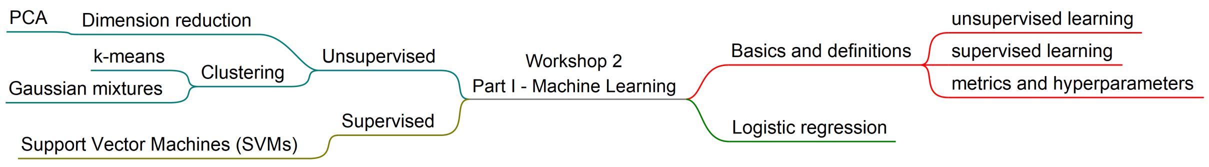
Table of Contents

Workshop 2 – Machine and Deep Learning [3 weeks]

1 Part I – Machine Learning (ML)

1.1 Topics Covered

- Principal Component Analysis (PCA)
- Support Vector Machines (SVMs)
- Logistic regression
- Performance metrics and hyperparameter selection



1.2 Workflow and Assessment

This subject follows a problem- and project-oriented approach. In this learning workflow, the focus is on solving practical (engineering) problems, which motivate acquiring theoretical (background) knowledge at the same time.

1.2.1 Objectives

- Use these tasks as a motivation to learn the fundamentals of machine and deep learning covered in lectures.
- Gain hands-on experience with machine and deep learning algorithms.
- Familiarise yourself with Python and Keras for *Deep Neural Networks (DNNs)* as widely-used practical software tools.
- Basics of time series analysis relevant to engineering.
- Solve basic machine learning problems using DNNs and the Keras library.
- Solve basic reinforcement learning problems using multi-armed bandit models.
- Connect theoretical knowledge and practical usage by doing it yourself.

1.2.1.1 Common objectives of all workshops

Gain hands-on experience and learn by doing! Understand how theoretical knowledge discussed in lectures relates to practice. Develop motivation for gaining further theoretical and practical knowledge beyond the subject material.

> **Self-learning** is one of the most important skills that you should acquire as a student. Today, self-learning is much easier than it used to be thanks to a plethora of online resources.

1.2.2 Assessment Process

1. Follow the procedures described below, perform the given tasks, and answer the workshop questions **in this Python notebook itself! The resulting notebook will be your Workshop Report!**
2. Submit the workshop report at the announced deadline
3. Demonstrators will conduct a brief (5min) oral quiz on your submitted report in the subsequent weeks.
4. Your workshop marks will be a combination of the report you submitted and oral quiz results.

> **The goal is to learn**, NOT blindly follow the procedures in the fastest possible way! **Do not simply copy-paste answers (from Internet, friends, etc.). You can and should use all available resources but only to develop your own understanding. If you copy-paste, you will pay the price in the oral quiz!**

1.3 PCA and Clustering

It is time to move beyond the toy data set. The next data set is still small and clean but is more interesting!

1.3.1 Example: Wireless Indoor Localization

The next data set shows the recorded signal strength from 7 different base stations at a smart phone. The phone is in one of the four rooms {1, 2, 3, 4}. The goal is to cluster the phones based on their signal strengths. Ideally, this should match the location of the phones to one of the four rooms.

This dataset (http://archive.ics.uci.edu/ml/datasets/Wireless+Indoor+Localization) was used in the following publications:

> - Jayant G Rohra, Boominathan Perumal, Swathi Jamjala Narayanan, Priya Thakur, and Rajen B Bhatt, 'User Localization in an Indoor Environment Using Fuzzy Hybrid of Particle Swarm Optimization & Gravitational Search Algorithm with Neural Networks', in Proceedings of Sixth International Conference on Soft Computing for Problem Solving,2017, pp. 286-295.
> - Rajen Bhatt, 'Fuzzy-Rough Approaches for Pattern Classification: Hybrid measures, Mathematical analysis, Feature selection algorithms, Decision tree algorithms, Neural learning, and Applications', Amazon Books

In [1]:

```
%matplotlib notebook
import pandas as pd
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
import matplotlib
from sklearn import cluster, datasets, mixture
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from sklearn import decomposition
from sklearn.mixture import GaussianMixture
from sklearn import metrics
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
```

In [ ]:

```
dataw = pd.read_csv('files/wifi_localization.csv', names=[f"s{i}" for i in range(1, 8)
dataw.head() # comment one to see the other
dataw.tail()
```

In [ ]:

```
print(dataw.size, dataw.shape)
```

In [ ]:

```
SRI = dataw.iloc[:,:7]
# a.shape
loc = dataw.iloc[:,7]-1
# loc.shape

# split into training and test sets
SRItrain, SRItest, loctrain, loctest = train_test_split(SRI, loc)
```

1.3.2 Question: Combined PCA and Clustering

1. Use k-means clustering on the training data to find the 4 clusters corresponding to the 4 rooms. Then, test the quality of the clusters using a mutual information based score. (https://scikit-learn.org/stable/modules/clustering.html#mutual-information-based-scores)
2. Conduct a PCA analysis (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) on the data with 2, 3, 4 features: find singular values, variance ratios, and plot in 2D or 3D (for 2 or 3 features). What is the number of features that balances information and complexity? Is there redundancy in data? Discuss.
3. Now, repeat k-means clustering with the PCA-transformed data. Do you see an improvement in scores? Why or why not? Discuss, based on your knowledge of PCA and k-means clustering.
4. We have the ground truth (location of phones) for this clustering task. Do we have this "luxury" in all clustering tasks? Discuss.
5. What is the difference between clustering and classification? Why is it not a good idea to use k-means clustering for classification? Discuss.

In [ ]:

```
''' Answer as code here '''
```

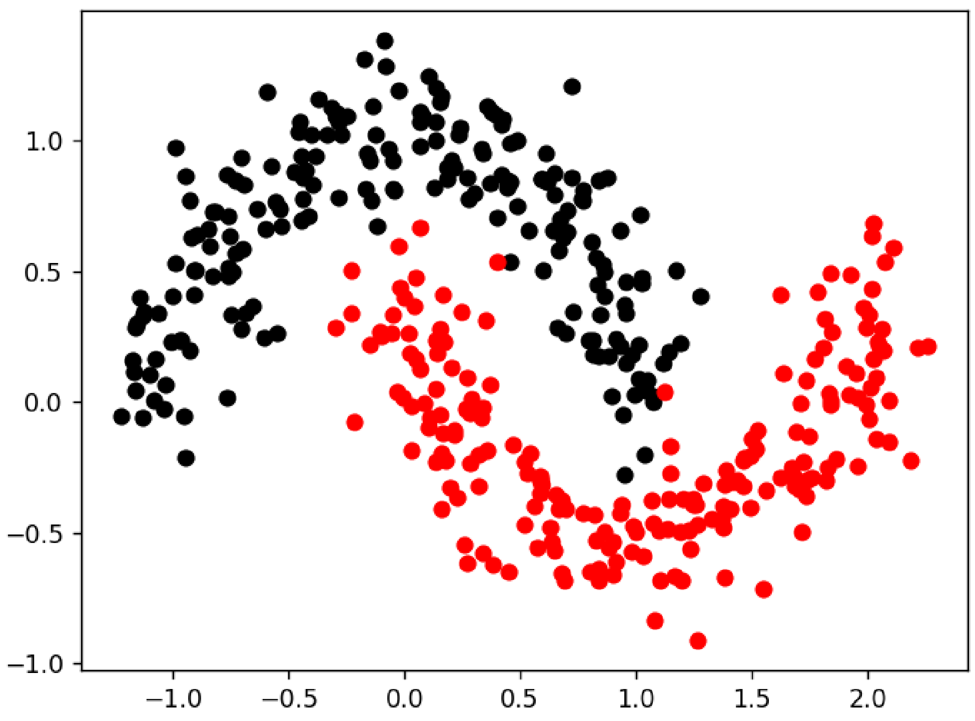
Answer as text here

1.4 Support Vector Machines (SVMs)

We have covered the theory of SVMs (https://en.wikipedia.org/wiki/Support-vector_machine) during the lectures. It is now time to see SVMs in action! It is appropriate to start with a noisy moon data set. We will use sklearn.svm (https://scikit-learn.org/stable/modules/svm.html) library.

In [2]:

```python
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
from sklearn import metrics

# helper function to visualise decision boundary, uses the svm model as input
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1
                                  ], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

# this is not necessary if you run it in the cell earlier...
#np.random.seed(Put the same or different group specific number!)

# Create a new moons data set
new_moons = datasets.make_moons(n_samples=400, noise=0.15)
Xm = new_moons[0] # data points
ym = new_moons[1] # 0, 1 labels of class, 200 each - giving us the ground truth

# Visualise the data set
order_ind = np.argsort(ym) # order labels, 200 each class
Xm1 = Xm[order_ind[0:200]]   # class 1 - only for visualisation
Xm2 = Xm[order_ind[201:400]] # class 2 - only for visualisation
plt.figure()
plt.scatter(Xm1[:,0], Xm1[:,1], color='black')
plt.scatter(Xm2[:,0], Xm2[:,1], color='red')
plt.show()
```



ıata values is

Ↄ.html) with a linear kernel and different C parameters. Plot the test output and boundary. Discuss your observations and comment on linear separability of this data. Provide the precision, recall, and F-score metrics. *Hint: see sklearn.metrics (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics), especially classification_report (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html#sklearn.metrics.classificatic*

2. Next, use an RBF kernel (https://en.wikipedia.org/wiki/Radial_basis_function_kernel) and repeat the first part for different C and gamma parameters. Do you observe an improvement compared to the linear version (both visually and in terms of scores)? Discuss your results.
3. Perform a cross-validated grid-search over a parameter grid to find good C and gamma hyper-parameters. Plot the AUC (`mean_train_AUC`) vs gamma for the best C. *Hint: check GridSearchCV (https://scikit-learn.org/stable/auto_examples/model_selection/plot_multi_metric_evaluation.html#sphx-glr-auto-examples-model-selection-plot-multi-metric-evaluation-py) and scoring parameter (https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter). Note that you should use semi-log plot!*

In [ ]:

```
''' Answer as code here '''
```

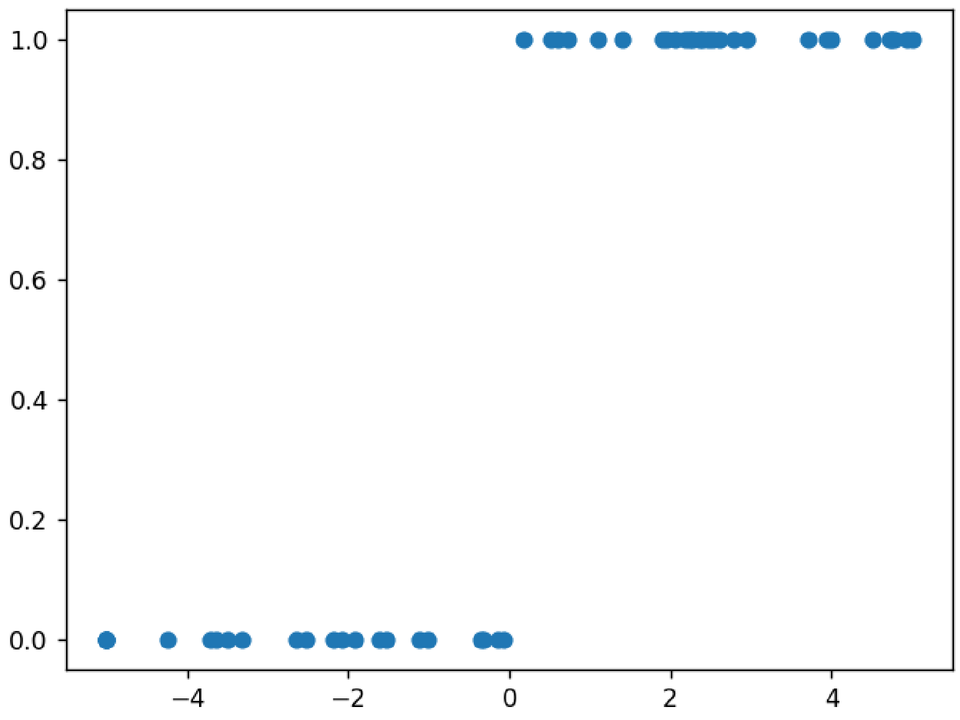
Answer as text here

1.5 Logistic Regression

Logistic regression uses a linear statistical model. It measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function. It is also used for classification. Let's start with a very simple example to **visualise the logistic function**.

In [3]:

```
# Generate a toy dataset

toy_samples = 50
X_toy = np.linspace(-5, 5, toy_samples)
Xtoy_test = np.linspace(-5, 5, 200)
# gaussian noise added
X_toy = X_toy + 2*np.random.normal(size=toy_samples)
# upper, lower bound
X_toy = np.clip(X_toy,-5, 5).reshape(-1,1)
# create labels
y_toy = ((np.sign(X_toy)+1)/2.0).ravel()

# visualise
plt.figure()
plt.scatter(X_toy, y_toy)
plt.show()
```



1.5.1 Question: Simple Logistic Regression (LR)

1. Fit a logistic and linear regression model to the data *(X_toy, y_toy)*. Find the logistic and linear model coefficients and bias (`trained_model.coef_` , `trained_model.intercept_`). Hint: check sklearn

   [logistic regression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html).
2. Test your models on a simple test set (`Xtoy_test`) given above. Plot your results and discuss.

Answer as text here

1.6 Example: Electrical Grid Stability with Simulated Data

This simulated [dataset (http://archive.ics.uci.edu/ml/datasets/Electrical+Grid+Stability+Simulated+Data+)](http://archive.ics.uci.edu/ml/datasets/Electrical+Grid+Stability+Simulated+Data+) is related to the local stability of a 4-node star system (where the electricity producer is in the center), which implements a decentralised Smart Grid Control concept.

> Arzamasov, Vadim, Klemens Boehm, and Patrick Jochem. ['Towards Concise Models of Grid Stability.' (https://dbis.ipd.kit.edu/download/DSGC_simulations.pdf)](https://dbis.ipd.kit.edu/download/DSGC_simulations.pdf) Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), 2018 IEEE International Conference on. IEEE, 2018

Note: In many engineering applications, such datasets can be generated through simulations (easy) or experimentation (harder). Different from classical ML applications, engineers often have a very good understanding of the underlying physical models, which gives a unique advantage. We will, however, keep it simple in this workshop and use the dataset as is.

Let's load and process the dataset.

In [5]:

```
griddata = pd.read_csv('files/Data_for_UCI_named.csv')
griddata.head()
```

Out[5]:

	tau1	tau2	tau3	tau4	p1	p2	p3	p4	g
0	2.959060	3.079885	8.381025	9.780754	3.763085	-0.782604	-1.257395	-1.723086	0.65045
1	9.304097	4.902524	3.047541	1.369357	5.067812	-1.940058	-1.872742	-1.255012	0.41344
2	8.971707	8.848428	3.046479	1.214518	3.405158	-1.207456	-1.277210	-0.920492	0.16304
3	0.716415	7.669600	4.486641	2.340563	3.963791	-1.027473	-1.938944	-0.997374	0.44620
4	3.134112	7.608772	4.943759	9.857573	3.525811	-1.125531	-1.845975	-0.554305	0.79711

In [7]:

```
Xgrid = griddata.iloc[:, 0:12]  # last two columns are the labels!
Xgrid.head()
```

Out[7]:

	tau1	tau2	tau3	tau4	p1	p2	p3	p4	g
0	2.959060	3.079885	8.381025	9.780754	3.763085	-0.782604	-1.257395	-1.723086	0.65045
1	9.304097	4.902524	3.047541	1.369357	5.067812	-1.940058	-1.872742	-1.255012	0.41344
2	8.971707	8.848428	3.046479	1.214518	3.405158	-1.207456	-1.277210	-0.920492	0.16304
3	0.716415	7.669600	4.486641	2.340563	3.963791	-1.027473	-1.938944	-0.997374	0.44620
4	3.134112	7.608772	4.943759	9.857573	3.525811	-1.125531	-1.845975	-0.554305	0.79711

In [ ]:

```
ygrid = griddata.iloc[:, 13]
# 0 if unstable and 1 if stable
ygrid = [ 0 if x=='unstable' else 1 for x in ygrid]

print("prepared data: ",Xgrid[0:5], ygrid[0:5])
```

1.6.1 Question: Checking Grid Stability using Logistic Regression (LR)

Now, we can use the simulated dataset (`Xgrid` , `ygrid`) to check grid stability. We will use first logistic regression for this purpose. Unfortunately, it is not possible to directly visualise this dataset, so we have to use performance metrics.

1. Preprocess and normalise `Xgrid` using a *sklearn.preprocessing* (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing). You can use, for example, `StandardScaler` or `MinMaxScaler` .
2. Train a logistic regression model that classifies the grid as stable or not, based on input `Xgrid` . Don't forget to divide `Xgrid` into training and test sets. Quantify the performance of LR using standard metrics such as accuracy, precision, recall, and f1 score on the test set. Plot the ROC curve (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics). How do these change w.r.t. parameter C (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)? Discuss your observations.
3. What are the coefficients of the LR that you trained? What do they tell you about the impact of independent input variables on the dependent output variable `ygrid` ? Discuss. *Hint: you can use statsmodels package (https://www.statsmodels.org) logistic regression to calculate p-values. Here is a nice tutorial (https://pythonguides.com/scikit-learn-logistic-regression/) on this. For further info, see e.g. this (http://www.r-tutor.com/elementary-statistics/logistic-regression/significance-test-logistic-regression) or this (http://blog.minitab.com/blog/adventures-in-statistics-2/how-to-interpret-regression-analysis-results-p-values-and-coefficients).*
4. Use a nonlinear SVM, e.g. with rbf kernel, to solve the problem. Quantify the performance of SVM classifier and compare with LR one. Discuss your findings.

Answer as text here

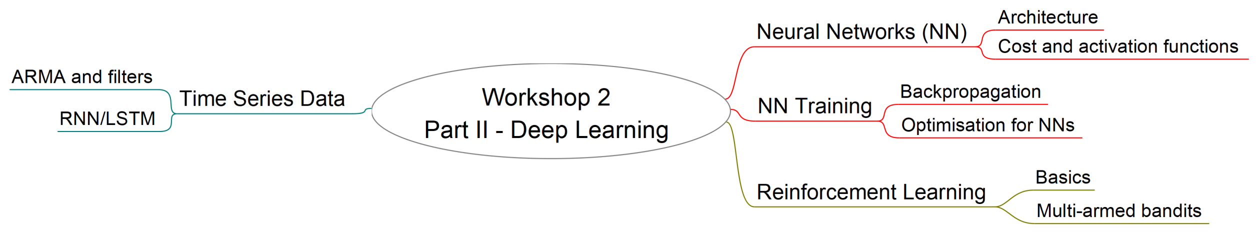
In [ ]:

```
''' Answer as code here '''
```

2 Part II – Deep Learning

2.1 Topics Covered

- Neural Networks (NNs) and Deep Learning
- Training NNs and optimisation
- Time series data and estimation
- Long Short-Term Memory (LSTM) – an example Recurrent Neural Network (RNN)
- Reinforcement learning basics
- Multi-armed bandits



2.2 Topic Notes

The [history of artificial neural networks (https://en.wikipedia.org/wiki/Artificial_neural_network#History)](https://en.wikipedia.org/wiki/Artificial_neural_network#History) is full of ups and downs. People got excited about and ignored them multiple times since mid 20th century. Since the start of the 21st century, artificial neural networks have enjoyed a big comeback in the form of [deep learning and DNNs (https://en.wikipedia.org/wiki/Deep_learning)](https://en.wikipedia.org/wiki/Deep_learning). This last wave rides on important and un-ignorable trends including rapid advances in computing (CPUs, GPUs, and specialised hardware), availability of sensors/data, and abundance of storage. While modern DNNs have already been applied to traditional problems in computer science such as image recognition and information retrieval with great success, their influence on engineering applications is only starting to be felt.

In this workshop, you will learn about basics of time-series analysis and how to solve various machine learning problems using DNNs. Doing this yourself will give you a chance to connect theoretical knowledge and practical usage. We will start with some of the data sets we have used in the previous workshop, which will make it easier to compare and contrast different approaches. More interesting problems will be posed as open-ended (and optional) tasks.

You will also familiarise yourself with [Keras, Python Deep Learning Library (https://keras.io/)](https://keras.io/), which is chosen for its popularity but most importantly, ease-of-use. Keras often uses the underlying and more flexible [TensorFlow (https://www.tensorflow.org/)](https://www.tensorflow.org/) framework. As usual, the tools and data in this workshop are chosen completely for educational reasons (simplicity, accessibility, cost). There are and will be better Deep Learning frameworks and more complex data sets but it is not realistic to cover all in a limited time.

> In the future, you should consider learning additional Deep Learning software packages and libraries. Finding the right tool for the right job is an important skill obtained through knowledge and experience.

2.3 Additional packages to install

In this workshop, we will use Tensorflow and Keras. If you are using a lab computer, *these packages should already be there in your Anaconda environment (please check!).* If not (or if you are using your own device), the best way to go forward is by creating a new environment (e.g you can name it `tfenv`), which you can do inside of *Anaconda Navigator* and after that, installing Tensorflow in your new environment.

Alternatively, you can create the environment using these instructions (https://www.pugetsystems.com/labs/hpc/How-to-Install-TensorFlow-with-GPU-Support-on-Windows-10-Without-Installing-CUDA-UPDATED-1419/#Step3%29CreateaPython%5C) followed by the command `conda install tensorflow-gpu` or `conda install tensorflow-gpu` if you wish to make use of your computer's NVIDIA graphics card (https://www.tensorflow.org/install/gpu). Note that installing tensorflow in either case pulls all the necessary packages including `scipy` , `scikit` etc. *In case this does not happen, be sure to install (* `scikit-learn` *,* `matplotlib` *,* `pandas` *, and any others that cause* `import error` *s) one by one.*

Another package we will need is **statsmodels** (https://www.statsmodels.org). You can also install it directly using *Anaconda Navigator.* or following instructions on their website.

Ask for help from your demonstrator in case you need it.

Don't forget to launch your notebook from the right environment!

2.4 DNNs for Classification

We will use first the now-familiar two-moon data set as an exercise for classifying with DNNs. This will help you to learn basics of Keras and deep learning on a problem which you have already solved with classical ML methods.

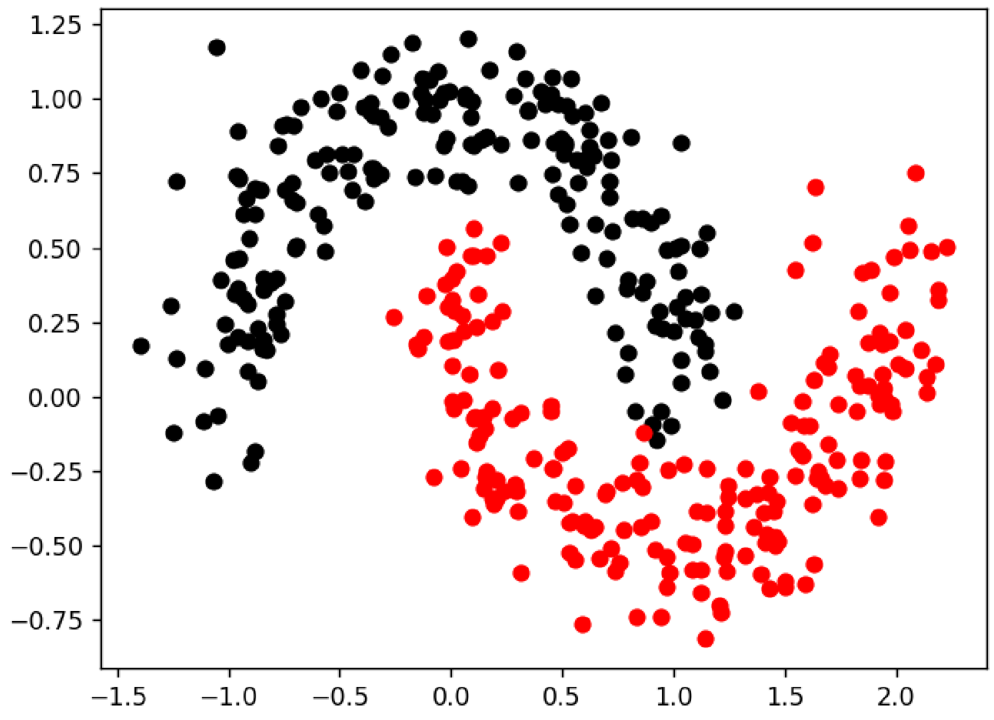
Note remember that Scikit Learn uses the numpy random state (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.random.seed.html#numpy.random.seed). See the code below and uncomment as instructed for repeatable results.

2.4.1 Important Note on Random Number/Vector Generation

Each group has to use a different number seed (which is an arbitrary number as illustrated above) and groups cannot share seeds. The pseudo-randomness is used here to create diversity. Otherwise, if groups use the same seed, the results will be the same (opening the door to plagiarism) and significant number of points will be taken off! As a practical hint, you can use a modified-combination of your student numbers.

In [8]:

```
%matplotlib notebook
import pandas as pd
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import train_test_split
from sklearn import cluster, datasets, mixture
from sklearn.cluster import KMeans
from sklearn.utils import shuffle

# Set a random seed as you did in optimisation workshop by uncommenting the line below!
#np.random.seed(Put here a group-specific number!)

# Create a new moons data set
new_moons = datasets.make_moons(n_samples=400, noise=0.15)
Xm = new_moons[0] # data points
ym = new_moons[1] # 0, 1 labels of class, 200 each - giving us the ground truth

# Visualise the data set
order_ind = np.argsort(ym) # order labels, 200 each class
Xm1 = Xm[order_ind[0:200]]   # class 1 - only for visualisation
Xm2 = Xm[order_ind[201:400]] # class 2 - only for visualisation
plt.figure()
plt.scatter(Xm1[:,0], Xm1[:,1], color='black')
plt.scatter(Xm2[:,0], Xm2[:,1], color='red')
plt.show()
```



In [ ]:

```
# split into training and test sets
Xmtrain, Xmtest, ymtrain, ymtest = train_test_split(Xm, ym)
```

2.5 Example: DNN with Keras

We first define our neural network (model) and compile it with an optimisation method, loss function, and metrics relevant to our problem. See the following documents as a starting point:

- Keras documentation, guide to sequential model (https://www.tensorflow.org/guide/keras/sequential_model)
- Tensorflow 2 Keras API (https://www.tensorflow.org/api_docs/python/tf/keras)

We are now using Tensorflow 2 (TF2) but a lot of the online material is on TF1. Therefore, you cannot use those scripts directly anymore but it is easy to modify them to TF2!

Additional information you might find helpful is available all over the web, e.g. evaluating performance (https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/) and Reduce Overfitting (https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-neural-networks-with-weight-constraints-in-keras/)

2.5.1 Reproducibility and Pseudo-randomness

It is possible to get reproducible results with Keras (https://machinelearningmastery.com/reproducible-results-neural-networks-keras/). However, this standard method (as implemented in the code below) works only with the CPU implementation of tensorflow as far as I understand.

Please don't forget to change the random seed in the code below and choose a group-specific arbitrary number as in previous workshops for full credit!

If your computer uses CUDA/GPU, don't worry about reproducibility for now. If you really wish to learn more about reproducibility with CUDA/GPU *optionally* you can have a look at this project (https://github.com/NVIDIA/tensorflow-determinism).

In [ ]:

```
%load_ext tensorboard
import datetime
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras import regularizers
#from tensorflow import keras

print(tf.__version__)
print("GPU is", "available" if tf.config.list_physical_devices('GPU') else "NOT AVAILAB
```

In [ ]:

```
# CHANGE THE RANDOM SEED FOR YOUR GROUP!
np.random.seed(216565)
tf.random.set_seed(216565)

# Define the DNN sequential model

model = Sequential()
model.add(tf.keras.layers.InputLayer(input_shape=(2,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.summary()

model.compile(
    optimizer=tf.keras.optimizers.SGD(),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy()]
)

# log results
log_dir = "logs/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=
```

Next, we train the DNN we have created using the training data.

In [ ]:

```
# The command below continues training from where you left it!
# If you wish to restart training from beginning rerun the cell above to reinitialise
 
# Train the model, iterating on the data in batches, record history
train_hist = model.fit(Xmtrain, ymtrain, epochs=200, batch_size=16, verbose=0, callbac
# use verbose = 1 to print individual epochs!
```

Note that the accuracy and loss start from different values whenever you restart the model and you end up with a different final accuracy and loss values whenever you train it. This is due to random initialisation and local minimum solutions in training optimisation. However, since the *fit* command is stateful and continues training from where it left, the results improve. How many epochs are needed to get over 90% accuracy?

If for some reason you wish to restart training from the beginning, rerun the previous cell to reinitialise the model!

Below, we look closer at how the network is structured and which parameters are trained.

In [ ]:

```
print(model.summary())
weights = model.get_weights() # Getting params
print(weights)
```

In [ ]:

```
# Plot model graph
# you need to install pydot and graphviz via anaconda for this to work! restart noteboo
# A better alternative is to use the tensorboard below!

#tf.keras.utils.plot_model(model, to_file='model.png', show_shapes=True, show_layer_nam
```

We can print the the actual score we have chosen and visualise the evolution of loss and accuracy over training epochs.

In [ ]:

```
score = model.evaluate(Xmtest, ymtest, batch_size=16, verbose=2)
print(score)

#train_hist.history

plt.figure()
plt.plot(train_hist.history['loss'])
plt.plot(train_hist.history['binary_accuracy'])
plt.xlabel('Epoch number')
plt.title('Training Loss and Accuracy')
plt.legend(['Loss', 'Accuracy'], loc='center right')
plt.show()
```

Finally, we compute and display the confusion matrix (https://en.wikipedia.org/wiki/Confusion_matrix), using sklearn's confusion_matrix (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html) function.

In [ ]:

```
from sklearn.metrics import confusion_matrix

ympred = model.predict(Xmtest)
ympredbinary = (ympred > 0.5)

cm = confusion_matrix(ymtest, ympredbinary)

pd.DataFrame(cm, columns=["Pred 0", "Pred 1"], index=["True 0", "True 1"])
```

We can use the tool `tensorboard` to analyse our results! Note the *log* directory in the folder where you have run your script.

In [ ]:

```
%tensorboard --logdir logs --host localhost
```

2.6 Question: Classifying Two-Moon Data with DNN

Use the same two-moon data (Xm, ym) given above for deriving the training and test sets. You can use the default ratio as done before or change it a bit, e.g. 0.3. The range of data values is OK so you can skip data normalisation.

1. Try different DNN structures instead of (8, 4, 1). For example, you can use only one hidden layer or many more layers. You can also use different activation functions as long as you end up with a single node binary classifier. Try also different optimisers and loss functions. Which one works best? Try, observe, and discuss!
2. For the best combination you managed to find, investigate the impact of training epochs and batch sizes on DNN performance. Measure performance in different ways using the metrics from Keras (https://www.tensorflow.org/api_docs/python/tf/keras/metrics) or classical Machine Learning as discussed during ML lectures. You can use the same sklearn library functions as in WS2 to document performance (see e.g. above). Observe the difference between training and test set loss and accuracy. Interpret your results. What does a big difference between training and test set performance mean?
3. Try different regularizers from Keras (https://www.tensorflow.org/api_docs/python/tf/keras/regularizers) to prevent over-fitting. Document your results and observations.

Some resources from the web, which may or may not be relevant:

- Measuring performance and basics (https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/)
- Weight constraints (different from regularisation) (https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-neural-networks-with-weight-constraints-in-keras/)
- A nice example (https://heartbeat.fritz.ai/introduction-to-deep-learning-with-keras-c7c3d14e1527)

Note: We are now using Tensorflow 2 which integrates Keras. Therefore, you probably cannot copy paste old scripts from web!

Answer as text here

In [ ]:

```
''' Answer as code here '''
```

2.7 Question: Wireless Indoor Localization *revisited*

We now revisit the wireless indoor localisation dataset (http://archive.ics.uci.edu/ml/datasets/Wireless+Indoor+Localization) from WS2. Remember that the data shows the recorded signal strength from 7 different base stations at a smart phone. The phone is in one of the four rooms $\{1, 2, 3, 4\}$. The goal is to classify the location of the phone to one of the four rooms.

1. Solve this classification problem with a DNN. Determine appropriate input and output layers and experiment with different numbers and sizes of hidden layers. *Hint: you can use, e.g., two sigmoid outputs to binary encode four classes.*
2. Measure performance in different ways using the metrics from Keras or classical Machine Learning as discussed during ML lectures. You can use the same sklearn library functions as in WS2 to document performance. Discuss your findings.

In [ ]:

```
dataw = pd.read_csv('files/wifi_localization.csv', names=[f"s{i}" for i in range(1, 8)]
dataw.head() # comment one to see the other
dataw.tail()
```

In [ ]:

```
print(dataw.size, dataw.shape)
```

In [ ]:

```
SRI = dataw.iloc[:,:7]
# a.shape
loc = dataw.iloc[:,7]-1
# loc.shape

# split into training and test sets
SRItrain, SRItest, loctrain, loctest = train_test_split(SRI, loc)
```

Answer as text here

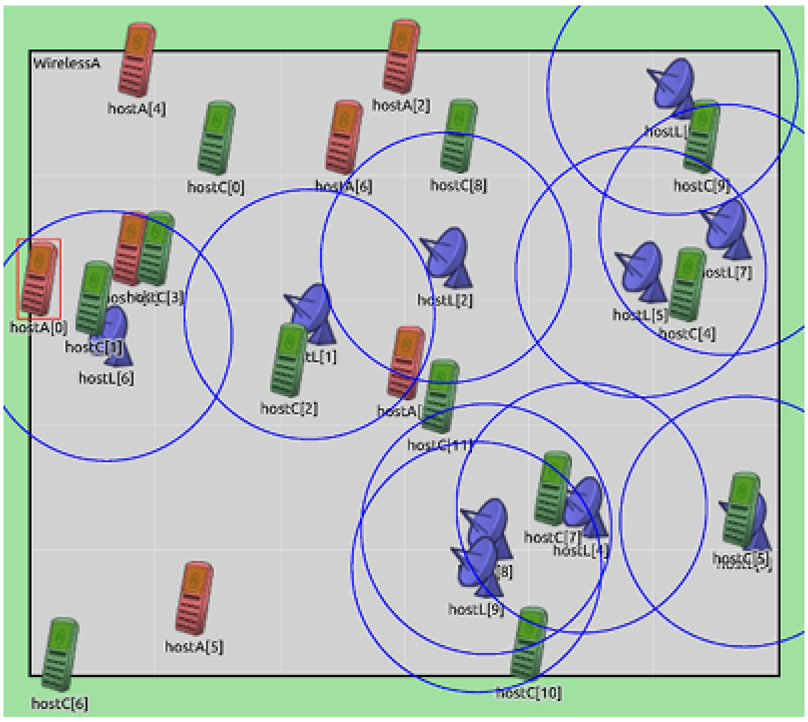
In [ ]:

```
''' Answer as code here '''
```

2.8 Communications Detective Mini-Project

Your job as a detective is to distinguish malicious people's communications from background civilian communication traffic. As a 21st century detective, you have access to a cognitive radio network and you have ML knowledge!

The [dataset (files/crn_data.csv)](files/crn_data.csv) is collected from a simulation where there are multiple malicious people and civilians communicating in a region with multiple passive cognitive radio nodes. Data about each transmission source is collected from the listener nearest to it. **The objective is to classify if a transmission source is a rogue agent or a civilian based on the data.**



The data file contains data from 2 classes:

- civilians - 129 instances (labeled as +1)

- rogue agents - 129 instances (labeled as -1)

Features/attributes are **not** normalised.

1. label 2. carrier_frequency 2. bandwidth 3. bitrate 4. session duration 5. message_length 6. inter-arrival time (iat)

This is an open-ended mini-project. However, for full points, you should consider:

1. Run a PCA to visualize the data.
2. Try multiple classifiers, e.g. SVM and DNN.
3. Do cross validation, give performance results using metrics, compare/contrast methods.
4. Normalise the data and repeat parts 1, 2 and 3. Discuss your findings.

In [ ]:

```
commdata = pd.read_csv('files/crn_data.csv')
commdata.head()
```

Answer as text here

In [ ]:

```
''' Answer as code here '''
```

2.9 Time Series Estimation

We will next use household electrical power demand as an interesting time-series, which is relevant to power systems and electrical engineering.

2.9.1 Electrical Power Household Demand Estimation

Estimating household power consumption is an important problem in power systems. The demand estimation is easy at the state or regional level due to low-pass filtering (or the law of large numbers) effect from aggregating thousands or even millions of customers' demands. The problem is much more challenging when the demand of individual houses is studied. It is almost impossible to predict when an individual household is going to boil water in the kettle or put on a load of washing. However, it is still possible to make good estimates.

We are given the yearly power consumption of two houses.

In [9]:

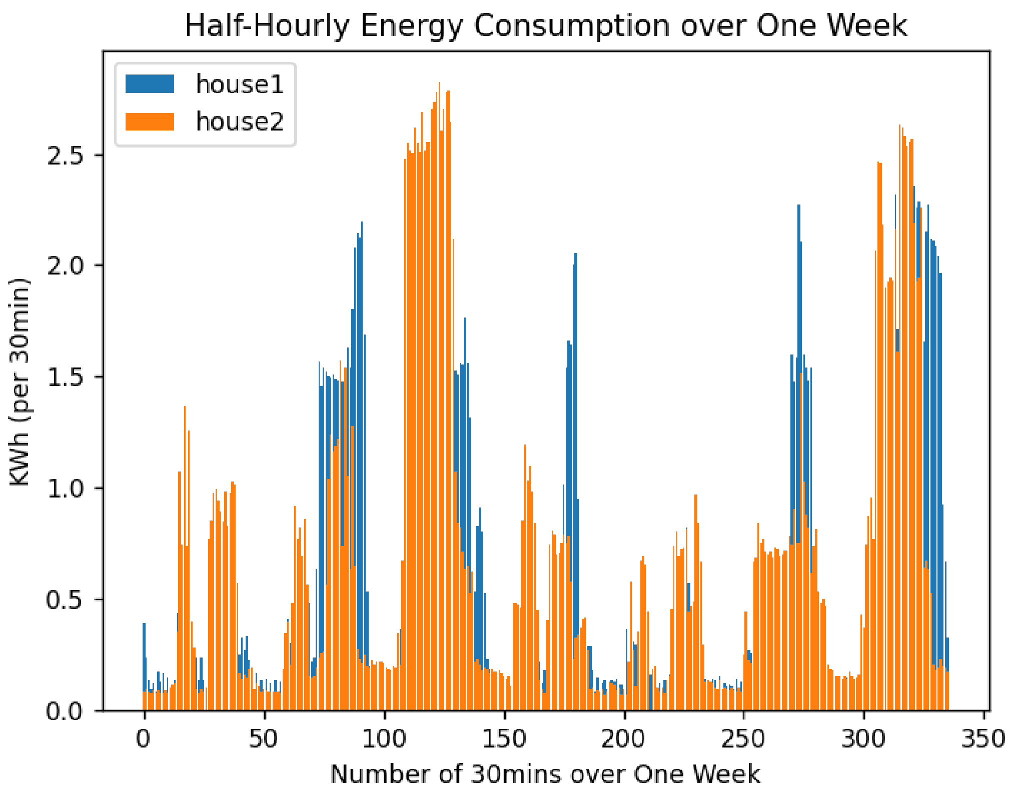
```
raw_data = pd.read_csv('files/two_houses.csv')
raw_data.head()
```

Out[9]:

	day	time	house1	house2
0	0	SMAPV3001	0.288	0.150
1	0	SMAPV3002	0.394	0.081
2	0	SMAPV3003	0.238	0.094
3	0	SMAPV3004	0.138	0.081
4	0	SMAPV3005	0.094	0.075

In [10]:

```
house1 = raw_data.iloc[1:,2]
house2 = raw_data.iloc[1:,3]

plt.figure()
plt.bar(np.arange(48*7),house1[0:48*7])
plt.bar(np.arange(48*7),house2[0:48*7])
plt.title('Half-Hourly Energy Consumption over One Week')
plt.xlabel('Number of 30mins over One Week')
plt.ylabel('KWh (per 30min)')
plt.legend(['house1','house2'])
plt.show()

house1.shape, house2.shape
```



```
C:\Users\tansu\AppData\Local\Temp\ipykernel_24060\1074511241.py:5: FutureW
arning: The behavior of `series[i:j]` with an integer-dtype index is depre
cated. In a future version, this will be treated as *label-based* indexin
g, consistent with e.g. `series[i]` lookups. To retain the old behavior, u
se `series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  plt.bar(np.arange(48*7),house1[0:48*7])
C:\Users\tansu\AppData\Local\Temp\ipykernel_24060\1074511241.py:6: FutureW
arning: The behavior of `series[i:j]` with an integer-dtype index is depre
cated. In a future version, this will be treated as *label-based* indexin
g, consistent with e.g. `series[i]` lookups. To retain the old behavior, u
se `series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  plt.bar(np.arange(48*7),house2[0:48*7])
```

Out[10]:

```
((17567,), (17567,))
```

2.9.2 Question: Time Series Estimation using ARMA Models

Use the ARMA linear estimation method to estimate the power consumption of house 1 and house 2. You can use statsmodel time series analysis tools (https://www.statsmodels.org/stable/tsa.html) for this.

1. Define and fit an ARMA model for the first 960 data points. Next, forecast the next 48 points. Measure your performance, e.g. in terms of Mean-squared Error (MSE) using statsmodels tools (https://www.statsmodels.org/stable/tools.html#measure-for-fit-performance-eval-measures), and plot results.
2. Try different AR and MA degrees and different data/time windows. Document and discuss your observations, including the performance vs fitting time trade-off (being careful not to let your model fit for hours!)

Hints: see ARIMA model (https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html), ARIMA results (https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMAResults.html), and ARIMA example (https://www.statsmodels.org/stable/examples/notebooks/generated/tsa_arma_1.html)

This is an alternative example implementation. (https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/)

Pointers for solution

- Use ARIMA model with order (p, 0, q) for implementing a pure ARMA model. ARIMA (https://otexts.com/fpp2/arima.html) differs from ARMA.
- Specific commands to use are ARIMA (https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMA.html) for creating the model and ARIMA.fit (https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.fit.html) with appropriate arguments as documented.
- After training, ARIMAResults (https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMAResults.html) functions such as `summary()` , `fittedvalues` , `params` , and `forecast(steps=nbrsteps)` will be very useful.

In [ ]:

```
from statsmodels.tsa.arima_model import ARIMA
```

Answer as text here

In [ ]:

```
''' Answer as code here '''
```

2.9.3 Question: Time Series Estimation using DNN/LSTM

Now, we will use DNNs, specifically LSTM to estimate the power consumption of house 1 and house 2. Specifically, we prepare our data to estimate the next 24 hour period based on the past 24 hours. Note that 24 hours mean 48 data points due to smart meters reporting half-hourly energy usage.

1. Define and train a Keras model that consists of LSTM and Dense layers with a 48 feature input and 48 feature output to forecast demand over the next 24 hour period based on past 24 hours. What type of activation function would you use at the output layer? Why? Try different (appropriate) loss functions and optimisers. You can use "mse" and "adam" as default choices. Choose `batch_size=128` and `epoch=20` as parameters to begin with. You can change these to your liking and are encouraged to experiment.
2. Provide model summary and keep track of training history to provide a plot of loss over epochs. Make predictions for different days and plot your predictions along with actual data. You can evaluate performance by calculating mean-squared error per day or over multiple days in the test set.
3. *[optional, no points]* you can try using 1-D CNN layer(s) before LSTM ones as a non-linear filter. Do you observe any improvements?

Useful documents and functions

- Keras model api documentation (https://www.tensorflow.org/api_docs/python/tf/keras), visualisation (https://www.tensorflow.org/guide/keras/train_and_evaluate#visualizing_loss_and_metrics_during_training sequential model (https://www.tensorflow.org/guide/keras/sequential_model).
- `fit`, `summary`, `evaluate`, `predict`
- a few links to resources that may be useful:

> https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#attributes-and-underlying-data (https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#attributes-and-underlying-data)
> https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/ (https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/)
>
> https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/ (https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/)
>
> https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/ (https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/)
>
> https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf (https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf)
> https://github.com/ni79ls/har-keras-cnn (https://github.com/ni79ls/har-keras-cnn)

In [ ]:

```
# sliding window function for next 24 hourly estimate
# see, e.g. https://towardsdatascience.com/using-lstms-to-forecast-time-series-4ab6883
# or https://machinelearningmastery.com/reframe-time-series-forecasting-problem/
def house_data(inseries):
    window_size = 48+48
    series = inseries
    series_s = inseries.copy()
    for i in range(window_size):
        series = pd.concat([series, series_s.shift(-(i+1))], axis = 1)
    series.dropna(axis=0, inplace=True)
    X = series.iloc[:,0:48]
    yday = series.iloc[:,48:48+48] # next day
    return X, yday


# get the estimate data for house1 and house2
X1, yday1 = house_data(house1[0:8736])
X2, yday2 = house_data(house2[0:8736])

X1.shape, yday1.shape
```

In [ ]:

```
# split into training and test sets for house 1
X1train, X1test, y1train, y1test = train_test_split(X1, yday1)
X1train = np.array(X1train).reshape(X1train.shape[0],X1train.shape[1],1)
X1test = np.array(X1test).reshape(X1test.shape[0],X1test.shape[1],1)

X1train.shape, X1test.shape, y1train.shape, y1test.shape
```

In [ ]:

```
from tensorflow.keras.layers import LSTM
import time
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

In [ ]:

```
''' Answer as code here '''
```

Answer as text here

2.10 Reinforcement Learning Overview

Reinforcement Learning (RL) has been making headlines the last few years and there are good reasons for it! Extensions of the methods you will see in this workshop have been used to make computers learn how to play Atari games by themselves (https://openai.com/blog/openai-baselines-dqn/) (see also this) (https://deepmind.com/research/publications/playing-atari-deep-reinforcement-learning/). The recent advances in solving most challenging board games (https://deepmind.com/research/alphago/) have been very impressive. Until even ten years ago, many people believed that computers would never learn how to play the game "Go" due to its combinatorial complexity. Today, AlphaGo variants are the first computer program to defeat a professional human Go player, the first program to defeat a Go world champion, and arguably the strongest Go player in history. It is a testament to the power of RL that AlphaGo Zero (https://deepmind.com/blog/alphago-zero-learning-scratch/) learns to play simply by playing games against itself, starting from completely random play.

The theoretical foundations of RL have been known for a long while as presented in lectures. Today's successes basically come from well-engineered or designed software that runs on powerful computing systems. Multiple heuristic algorithms and designs verified through extensive experimentation seem to be the key methodology. Despite introducing state-of-the-art concepts, tools, and implementations, this workshop provides only an initial starting point to the world of modern RL.

> Learning more on RL requires good coding skills and a powerful computer (often with a good CUDA-supporting graphics card) or a cloud computing account with one of the major providers. Computer and board games have been the natural playground of modern RL. However, application of RL to engineering disciplines (https://blog.insightdatascience.com/using-reinforcement-learning-to-design-a-better-rocket-engine-4dfd1770497a) remains an under-explored and very exciting domain!

2.11 RL with Multi-armed Bandits

In a **k-armed (multi-armed) bandit** problem, a decision-making agent repeatedly chooses one of k different actions. Each action can be interpreted as pulling one of the k different levers. After each choice, the agent receives a reward obtained from a probability distribution that depends on the selected action. The objective is to maximise the expected total reward over a time horizon, for example, over $1,000$ action selections, or time steps. Multi-armed bandits have a variety of important applications (https://medium.com/@CornellResearch/whats-behind-your-navigation-app-79d2754e6878) ranging from clinical trials and routing (including navigation) to recommender systems.

As a special case of *reinforcement learning*, the multi-armed bandit (https://en.wikipedia.org/wiki/Multi-armed_bandit) problem has actually only a single state. The agent still has to learn the environment represented by the underlying probability distributions and rewards. The problem provides a nice introduction to *reinforcement learning* and an opportunity to explore the fundamental **exploration versus exploitation** trade-offs involved.

Hint: Example implementations online (randomly selected, not guaranteed to be correct):

- https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/ (https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/)
- https://peterroelants.github.io/posts/multi-armed-bandit-implementation/ (https://peterroelants.github.io/posts/multi-armed-bandit-implementation/)
- https://lilianweng.github.io/lil-log/2018/01/23/the-multi-armed-bandit-problem-and-its-solutions.html (https://lilianweng.github.io/lil-log/2018/01/23/the-multi-armed-bandit-problem-and-its-solutions.html)
- https://towardsdatascience.com/comparing-multi-armed-bandit-algorithms-on-marketing-use-cases-8de62a851831 (https://towardsdatascience.com/comparing-multi-armed-bandit-algorithms-on-marketing-use-cases-8de62a851831)

In [ ]:

```
%matplotlib notebook
import pandas as pd
import numpy as np
import time
import random
import matplotlib.pyplot as plt
import matplotlib
from collections import deque
from tensorflow.keras.optimizers import Adam
```

2.11.1 10-armed bandit data set

Let's first (create or) *load* a random 10-armed data set that approximately matches the description in Section 2.3 of Sutton and Barto RL book. (http://incompleteideas.net/book/the-book-2nd.html)

In [ ]:

```
def gen_data(num_bandits=10, T=2000, filename='10armdata'):
    ## function generates a synthetic data set with given parameters
    ## and saves the result to files folder under the given name

    # init data array
    tenarm_data = np.zeros((T,num_bandits))

    # random mean awards
    mean_rewards = np.random.normal(size=num_bandits)
    #print(mean_rewards)
    #print(np.random.normal(0,1,num_bandits))
    for t in range(T):
        tenarm_data[t,:]=np.random.normal(mean_rewards,1,num_bandits)

    np.save('./files/'+filename, tenarm_data)


# No need to set the random seed again if you did it in above cells.

#gen_data()
#tenarm_data.shape
#tenarm_data[0:10,:]

# use generated data
tenarm_datal = np.load('./files/10armdata.npy')
tenarm_datal.shape
```

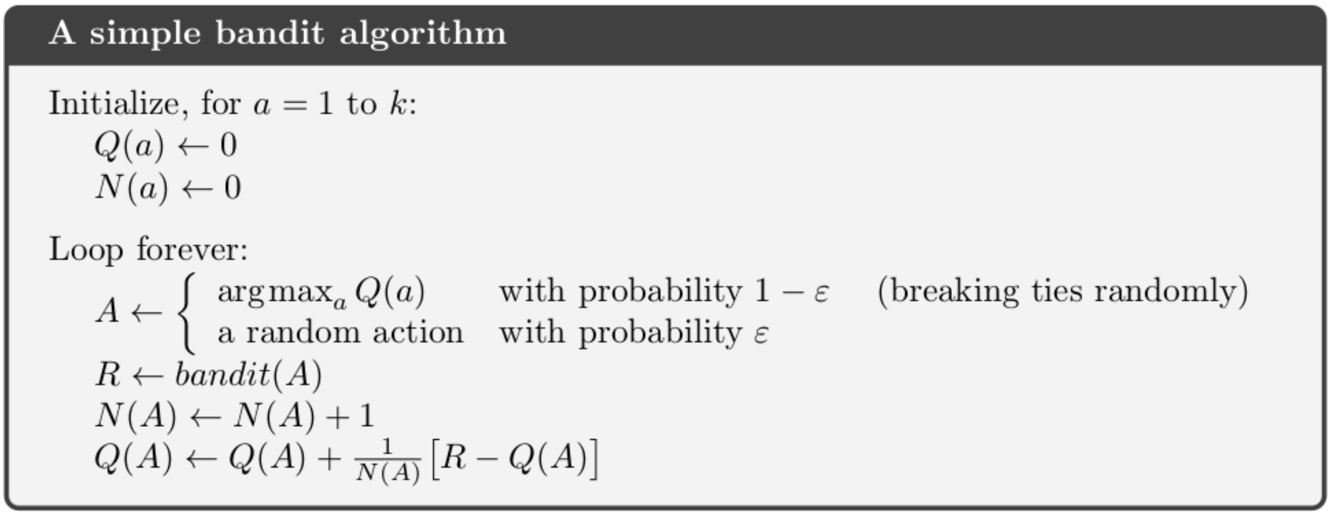
2.11.2 Multi-armed Bandit Algorithms

We now implement a simple random strategy for selecting actions. The results are also random as expected. This can be considered as pure **exploration** since the algorithm keeps randomly choosing actions. However, note that we do not make proper use of the randomly collected observations yet.

In [ ]:

```python
def bandit_random(data=tenarm_datal):
    # random selection bandit algorithm

    num_bandits = tenarm_datal.shape[1]
    T = tenarm_datal.shape[0]
    # init storage arrays
    selections = np.zeros(T) # sequence of lever selections
    step_rewards = np.zeros(T) # sequence of step selections
    cum_rewards = np.zeros(T) # sequence of cumulative rewards
    # main loop
    for t in range(T):
        sel = random.randrange(num_bandits)
        selections[t] = sel
        step_rewards[t] = data[t,sel]
        if t>0:
            cum_rewards[t] = step_rewards[t]+cum_rewards[t-1]
        else:
            cum_rewards[t] = step_rewards[t]

    total_reward = cum_rewards[-1] # the last one is total reward!

    return (selections, step_rewards, cum_rewards, total_reward)



(selections, step_rewards, cum_rewards, total_reward) = bandit_random()

print(total_reward)
plt.figure()
plt.title('Cumulative Reward vs Time')
plt.xlabel('Time')
plt.ylabel('Cumulative Reward')
plt.plot(cum_rewards)
plt.show()
```

Let us consider next a more meaningful strategy, known as ε**-greedy algorithm**. The idea is to explore with a pre-determined, fixed probability $\varepsilon < 1$ and exploit, i.e. get the maximum reward given the current knowledge, with probability $1 - \varepsilon$. The observations are now used to estimate the values of actions by averaging. This well-known algorithm is discussed in Section 2.7 of Sutton and Barto book (http://incompleteideas.net/book/the-book-2nd.html) and described below:



```
A simple bandit algorithm

Initialize, for $a = 1$ to $k$:
    $Q(a) \leftarrow 0$
    $N(a) \leftarrow 0$

Loop forever:
    $A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
    $R \leftarrow bandit(A)$
    $N(A) \leftarrow N(A) + 1$
    $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$
```

We provide a rudimentary implementation below as a single run.

In [ ]:

```python
def bandit_epsgreedy(data=tenarm_datal, eps=0.1):
    # epsilon-greedy bandit algorithm

    # parameters
    num_bandits = data.shape[1]
    T = data.shape[0]

    # init storage arrays
    Q = np.zeros(num_bandits)
    N = np.zeros(num_bandits)
    selections = np.zeros(T) # sequence of lever selections
    step_rewards = np.zeros(T) # sequence of step selections
    cum_rewards = np.zeros(T) # sequence of cumulative rewards
    # main loop
    for t in range(T):

        # pull lever
        if np.random.rand() < eps:
            # make a random selection
            sel = random.randrange(num_bandits)
        else:
            # choose the best expected reward
            sel = np.argmax(Q)

        # update nbr of selections made
        N[sel] = N[sel] + 1
        # update mean reward estimate
        Q[sel] = Q[sel] + (1/N[sel])*(data[t,sel]- Q[sel])

        # store values
        selections[t] = sel
        step_rewards[t] = data[t,sel]
        if t>0:
            cum_rewards[t] = step_rewards[t]+cum_rewards[t-1]
        else:
            cum_rewards[t] = step_rewards[t]

    total_reward = cum_rewards[-1] # the last one is total reward!

    return (selections, step_rewards, cum_rewards, total_reward)



(selections, step_rewards, cum_rewards, total_reward) = bandit_epsgreedy(eps=0.15)

print(total_reward)
plt.figure()
plt.title('Cumulative Reward vs Time')
plt.xlabel('Time')
plt.ylabel('Cumulative Reward')
plt.plot(cum_rewards)
plt.show()
```

Next, we run the algorithm over multiple simulations, which we generate by permutating the input data. The obtained average results are naturally less "noisy". **It may take many simulations to get low-variance, averaged results.**

In [ ]:

```
def bandit_epsgreedy_sims(datasim=tenarm_datal, epsilon=0.1, nbr_sims=10):
    # parameters
    num_bandits = datasim.shape[1]
    T = datasim.shape[0]

    # store values
    sim_cum_rewards = np.zeros((nbr_sims,T))
    sim_total_rewards = np.zeros(nbr_sims)

    for s in range(nbr_sims):
        (dummy,dummy, cum_rewards, total_reward) = bandit_epsgreedy(data=np.random.perr
                                                                    eps=epsilon)
        sim_cum_rewards[s,:] = cum_rewards
        sim_total_rewards[s] = total_reward

    return (sim_cum_rewards, sim_total_rewards)


(sim_cum_rewards, sim_total_rewards) = bandit_epsgreedy_sims(epsilon=0.15, nbr_sims=10(
print('Average total reward = ', np.average(sim_total_rewards))

sim_avg_rewards = np.average(sim_cum_rewards, axis=0)
sim_avg_rewards.shape
plt.figure()
plt.title('Average Cumulative Reward vs Time')
plt.xlabel('Time')
plt.ylabel('Average Cumulative Reward')
plt.plot(sim_avg_rewards)
plt.show()
```

2.11.3 Exploration vs Exploitation Trade-off

It is important to investigate the relationship between the outcome (average cumulative reward over time) and ε parameter. For small ε, the algorithm is more greedy and chooses the best action (given knowledge level, here Q estimate) most of the time. This is called **exploitation** in reinforcement learning (RL) (https://en.wikipedia.org/wiki/Reinforcement_learning). For large ε, the algorithm spends more time in **exploration** mode and obtains better Q estimates. This **exploration vs exploitation** trade-off is fundamental to all RL approaches (https://www.coursera.org/learn/fundamentals-of-reinforcement-learning), not just multi-armed bandits. The same concepts are also relevant to dual control (https://en.wikipedia.org/wiki/Dual_control_theory) as well as adaptive control (https://en.wikipedia.org/wiki/Adaptive_control).

2.11.4 Question: A Multi-armed bandit for CDN Optimisation

In this question, the problem of real-world data retrieval from multiple redundant sources is investigated. This communication network problem is commonly known as the Content Distribution Network (CDN) problem (see a relevant paper, right click to download) (./files/performance_of_CDN.pdf). An agent must retrieve data through a network with several redundant sources available. For each retrieval, the agent selects one source and waits until the data is retrieved. The objective of the agent is to minimize the sum of the delays for the successive retrievals. This problem is investigated in Section 4.2 of this paper, (right click to download) (./files/bandit.pdf) and this related project (http://bandit.sourceforge.net/) as well as discussed in this practical book (http://shop.oreilly.com/product/0636920027393.do).

We will use a subset of the publicly available (./files/license.txt) university web latency data set from the bandit project (http://bandit.sourceforge.net/), which contains retrieval delay/latency measurements from over 700 universities' homepages in milliseconds. Let's decrease the number of options (columns) randomly to 20 to make it computationally less time consuming (but you can change this later if you wish). The rewards are the negatives of the delays.

In [ ]:

```
univ_data = pd.read_csv('./files/univ-latencies.csv')
univ_data = -univ_data.sample(n=20, axis=1) #choose 20 columns randomly for computatior
univ_data.head()
```

Answer the following by implementing and simulating well-known multi-armed bandit algorithms.

1. Apply the ε-greedy algorithm to the CDN problem. Sample randomly with replacement from the *univ-latencies* dataset in order to simulate latencies. You should use negative of latencies as rewards here since high latency is not desirable. Try different ε values to investigate the exploration vs exploitation trade-off and the best total average reward.
2. Implement and apply the *upper confidence bound* (UCB) action selection algorithm to the same data set. Compare your results and briefly discuss your findings.

Answer as text here

In [ ]:

```
''' Answer as code here '''
```

In [ ]:

```
```

3 Workshop Assessment Instructions

You should complete the workshop tasks and answer the questions within the allocated session! **Submission deadline is usually end of the last week of the workshop. Please check Canvas for the exact deadline!**

It is **mandatory to follow all of the submissions guidelines** given below. *Don't forget the Report submission information on top of this notebook!*

1. The completed Jupyter notebook and its Pdf version (you can simply print-preview and then print as pdf from within your browser) should be uploaded to the right place in Canvas. *It is your responsibility to follow the announcements!* **Late submissions will be penalised (up to 100% of the total mark depending on delay amount)!**
2. Filename should be “ELEN90088 Workshop **W: StudentID1-StudentID2** of session **Day-Time**", where **W** refers to the workshop number, **StudentID1-StudentID2** are your student numbers, **Day-Time** is your session day and time, e.g. *Tue-14*.

3. Answers to questions, simulation results and diagrams should be included in the Jupyter notebook as text, code, plots. *If you don't know latex, you can write formulas/text to a paper by hand, scan it and then include as image within Markdown cells.*
4. Please submit your report as a group.

3.1 Workshop Marking

- **Each workshop has 15 points corresponding to 15% of the total subject mark inclusive individual oral examination.** You can find the detailed rubrics on Canvas.
- Individual oral quizzes will be scheduled within the next two weeks following the report submission. They will be during workshop hours. Therefore, it is important that you attend the workshops!
- The individual oral examination will assess your answers to workshop questions, what you have done in that workshop, and your knowledge of the subject material in association with the workshop.

3.1.1 Additional guidelines for your programs:

- Write modular code using functions.
- Properly indent your code. But Python forces you do that anyway ;)
- Heavily comment the code to describe your implementation and to show your understanding. No comments, no credit!
- Make the code your own! It is encouraged to find and get inspired by online examples but you should exactly understand, modify as needed, and explain your code via comments. If you resort to blind copy/paste, you will certainly not do well in the individual oral quizzes.

Report Submission Information (must be completed before submitting report!)

- Student 1 Full Name and Number:
- Student 2 Full Name and Number:
- Workshop day: e.g., Wednesday
- Workshop time: e.g., 12pm