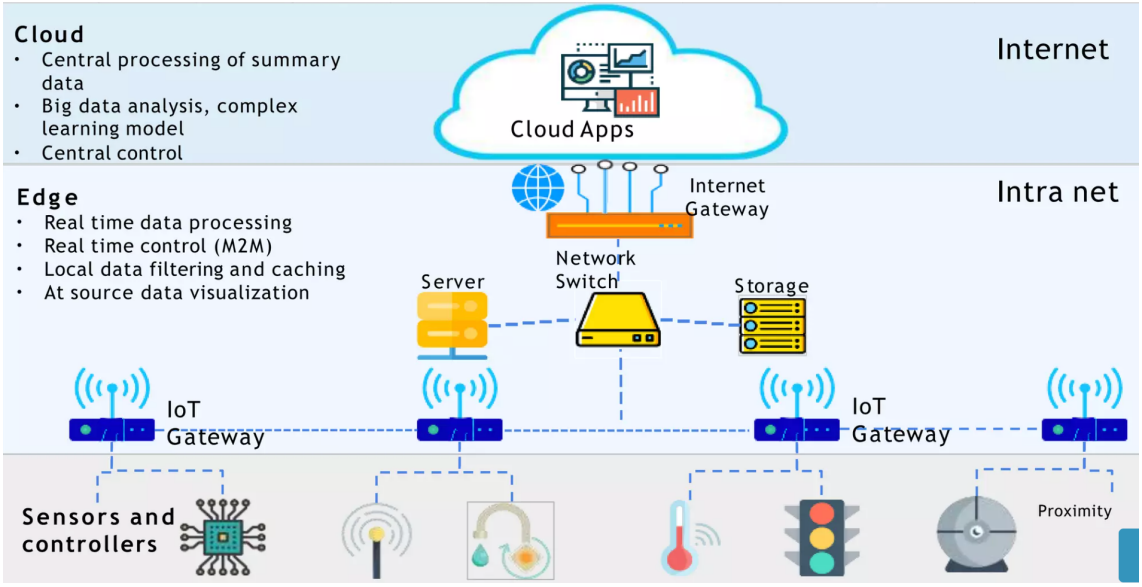
Project 1 - Fuzzy Systems and Neural Networks

1 – Introduction

The explosive growth and increasing computing power of Internet of Things (IoT) devices has resulted in unprecedented volumes of data. From connected vehicles to intelligent bots on the factory floor, the amount of data from devices being generated in our world is higher than ever before. Yet, most of this IoT data is not exploited or used at all, since sending all that device-generated data to a centralized data center or to the cloud causes bandwidth and latency issues.



In the IoT, **edge devices** are the computing devices that process data on a local level and transmit data to the local network and the cloud. These devices can thus translate between the protocols or languages that local systems use, into the protocols where data is further processed. Additionally, such devices can potentially use their **spare computing capabilities** to enable faster and more comprehensive data analysis, creating the opportunity for deeper insights, faster response times, better bandwidth availability and improved customer experience. In **Edge computing**, edge devices offer this more efficient alternative, where data is processed and analyzed closer to the point where it's created. Because data does not traverse over a network to a cloud or data center to be processed, latency is significantly reduced. Edge computing — and mobile edge computing on 5G networks — enables faster and more comprehensive data analysis, creating the opportunity for deeper insights, faster response times and improved customer experiences.

Essentially, Edge computing is a distributed information technology architecture in which client data is processed at the periphery of the network, as close to the originating source as possible. It encompasses any computing and network resources along the path between data sources and cloud data centers. An important strategic question in Edge computing is, how to manage the deployment of workloads that perform these types of

actions without compromising aspects such as network security risks, management complexities, and the limitations of latency and bandwidth.

Edge computing must therefore address three principal network limitations: bandwidth, latency and congestion.

- **Bandwidth.** Bandwidth is the amount of data which a network can carry over time, usually expressed in bits per second. All networks have a limited bandwidth, and the limits are more severe for wireless communication. This means that there is a finite limit to the amount of data – or the number of devices – that can communicate data across the network. Although it's possible to increase network bandwidth to accommodate more devices and data, the cost can be significant, and there are several other problems that won't be solved this way.
- **Latency.** Latency is the time needed to send data between two points on a network. Although communication ideally takes place at the speed of light, large physical distances coupled with network congestion or outages can delay data movement across the network. This delays any analytics and decision-making processes, and reduces the ability for a system to respond in real time. It can even cost lives in scenarios involving, for example, autonomous vehicles.
- **Congestion.** The internet is basically a global "network of networks." Although it has evolved to offer good general-purpose data exchanges for most everyday computing tasks – such as file exchanges or basic streaming – the volume of data involved with tens of billions of devices can overwhelm the internet, causing high levels of congestion and forcing time-consuming data retransmissions. In other cases, network outages can exacerbate congestion and even sever communication to some internet users entirely – making the internet of things useless during outages.

2 – Fuzzy System

2.1 Objectives and Considerations

The goal of this project is to build an intelligent system that manages the computing load assigned for Edge computing tasks on an Edge device (CLP), without compromising the main role of the device. It is assumed that each device only processes data coming from lower levels (e.g., sensors). When the device is operating as a pure Edge device (i.e., CLP = 0), only networking duties (including routing data to cloud) are performed.

Unfortunately, there are several hurdles to overcome: you have no data available to train a system; the edge devices can have very different hardware capabilities (e.g: memory, processor); there is too much uncertainty; etc. Given this, it was decided that the best alternative is to build a Fuzzy Inference System (FIS).

To help you to develop such system, I gathered some experts and let them talk freely about how they view the problem. The following paragraphs, generated using ChatoGPT, resume their impressions and opinions:

- The FIS should be device agnostic (independent from the processor/memory/OS characteristics). Assume that all devices can run the code you will generate.
- The output of the FIS, **CLPVariation**, should indicate if the computing load percentage (CLP) assigned for Edge computing tasks should increase, maintain, or decrease. The output should obviously be "gradual", in a Fuzzy sense. Use the interval [-1,1] to represent CLPVariation.
- Useful available features for such system are:
    - Memory usage (%)
    - Processor load (%) – Note: Processor load is not the same as CLP
    - Input network throughput
    - Output network throughput
    - Available output bandwidth
    - Latency (mS)
- The features of this type of system should be one-minute averages, and a decision should be made every minute.
- Assume that you can also have access to the variation rate of each feature, i.e., you know if a feature has increased or decreased (and by how much) in the last minute.
- Probably it's overkill to use all the 12 features (average value and variation of each feature).
- Network throughput refers to the rate of message delivery over a communication channel, such as Ethernet or packet radio.
- High latency means that it's better to process data locally.
- Note that throughput and latency aren't the same thing and can often not be correlated.
- You don't want to waste the available processing capability, so try to maintain the processor load and available memory above average.
- A device with very high memory usage or processor load (e.g. above 85%) won't likely be able to perform its basic Edge duties efficiently (simple data processing, routing, network security, etc.)
- Decreasing CLP implies the necessity to move more of the input data to the cloud (and this might be limited by the available output bandwidth.)
- Abrupt changes in CLP should be avoided unless the situation is critical.

The experts are also trying to collect, analyze and annotate some data that might be helpful to see how your system is performing. They will make it available as soon as possible (tentative date: September 30th).

Note: The available data is merely indicative. Your system is not expected to give the exact same output. If the values are similar, you shouldn't be concerned. The problem is if your system is giving outputs that contradict the examples (or if the scale of the variation is too different).

2.2 – Implementation Notes

You must use Simpful to implement this project. Solutions using ScikitFuzzy will not be accepted.

To implement a manageable Fuzzy System that doesn't suffer from combinatorial rule explosion (and lack of interpretability), you will need to reduce the number of inputs in the used Rulebases. You can achieve this by using a hierarchical system that allows you to combine features in small groups (see the slides about fuzzy system complexity).

As an "expert", it is up to you to analyze the problem, see which features might or might not be helpful, check how features might can be combined, etc.

Be **creative**, use **common sense**, and show your Fuzziness.

3 – Neural Networks

Once the experts have collected and annotated enough data, it might be possible to try to implement a supervised approach to address this problem. With the available data, implement a model based on a Multilayer Neural Network (NN). Note that this is a "regression" problem, not a "classification" problem.

Remember that the more complex the NN, the more data is needed to train it properly. Therefore, you should experiment with the available data to decide on the architecture of the NN (number of inputs, number of hidden layers, etc.). I advise you to start with a "standard" MLP, using the most usual activation function (logistic), the most usual solver (stochastic gradient descent – sgd), not using regularization, etc. Note that these ARE NOT the default parameters for NN in Scikit, so you must choose them explicitly.

Use all the knowledge you have acquired so far regarding data preparation, experimental setup and Neural Networks, to see how well you can solve this problem and build a good model (please re-check the theory slides regarding data and experimental setup before starting the project).

Try to avoid OVERFITTING, since later I will check how your system performs under previously unseen data. The best way to know if there is overfitting or not is to use a Train/Validation/Test split, and only use the Test set once your model is completely defined (input features, number of layers, number of neurons per layer, activation functions, etc.). I.e, use a Holdout set! If after checking the results on the Test Set, you change any parameter or hyperparameter (even if it's a simple change in the number of neurons in a hidden layer), then you are accidentally fitting the model to the Test set (which might result in overfit).

3.1 – Using the NN as a multiclass classifier

Now let's assume that you want to use the NN as a multiclass classifier with the following 3 classes: "Decrease", "Maintain", "Increase". Define which intervals of the NN output correspond to each class and calculate the performance of your system on the used Training, Validation and Test sets.

4 – Submission details and Deadline

This Project accounts for 50% of the Lab final grade. The final code and a comprehensive report must be submitted via Fenix until **Monday, October 7th, at 23:59**.

The students must submit:

a) All the developed code.
b) A piece of code that will allow to test each of your models. This code, called TestMe, accepts as a parameter a "Proj1_TestS.csv" file that contains 12 columns of normalized data (between 0 and 1), 6 columns containing the features indicated in the previous sections, plus 6 columns with the respective variation rates. The code must give the output (**CLPVariation**) of your FIS and of your NN for each of the lines in the file stored in a .csv file named "TestResult.csv". The CLPVariation must be a value in the interval [-1,1]. Note that the NN model cannot be trained on this data. You need to load a previously saved model trained on the previously available data.
c) A report with two parts:
    i. Part I: Indicate the "architecture" of your FIS and the rationale you used to build it. Include all the membership functions, linguistic terms, and rules. Illustrative graphics are encouraged. Also describe how you tested the system to see if it performed as you envisioned it, and any tunning and modifications you made until you reached the final version. Present the results you obtained and analyze them.
    ii. Part II: Present the NN model, the rationale you used to build, train, validate and test it. Don't forget to include and analyze the results you obtained when using the model for both the regression problem and the classification problem.