📝

# Deliverable 2

## Evolving Neural Networks

The purpose of this project is to build the first stepping stones for a “DENSER-like” framework.

Based on the work developed in the practical classes, this project will consist of:

1. Defining a string-based representation for the networks (the genotype).
2. Defining a network class that is able to parse those instructions and build a functional pytorch network structure (the phenotype).
3. Defining a string-based representation for the optimizer.
4. Defining a way to parse those instructions and build and functional pytorch optimizer.
5. Sample all parameter values from a grammar (this allows for a restricted search space and removes the need to deal with invalid combinations).
6. Define 4 simple genetic operators:
    a. Network crossover
    b. Add layer mutation
    c. Remove layer mutation
    d. Change optimizer mutation

## Details

- The grammar should contain a reasonable number of parameters for each of the layers/optimizers. Provided the values of the parameters are standard and within reason, there is no need to justify their choice.

- The networks need to be able to use the following layers and their respective parameters:
    - `Linear` : number of features, bias
    - `BatchNorm1d` : eps, momentum
    - `LayerNorm` : eps
    - `Dropout` : dropout probability (p)
    - `AlphaDropout` : dropout probability
    - [`Sigmoid`, `ReLU`, `PReLU`, `ELU`, `SELU`, `GELU`, `CELU`, `SiLU`]
- The networks need to be able to be trained using the following optimizers and their respective parameters:
    - `Adam` : lr, betas
    - `AdamW` : lr, betas, weight decay
    - `Adadelta` : lr, rho
    - `NAdam` : lr, betas, momentum decay
    - `SGD` : lr, momentum, nesterov
- Each network needs to have at least 1 layer and a maximum 50 layers. This choice needs to be made at random when generating the genotype.
- The genetic operators that need to be implemented are the following:
    - `Crossover` : Takes a subset of layers from 2 different networks and swaps them. This subset cannot include the first and final layers.
      how big, consecutive layers?
    - `Add layer mutation` : Add any new layer to any part of the genotype, and rebuild the network. It cannot be added before the first or after the last layer.
      one or many?
    - `Remove layer mutation` : Remove any layer that is neither the first nor the last from the genotype and rebuild the network.
    - `Change optimizer mutation` : Change any parameter from the optimizer genotype and rebuild it. If the type of optimizer is changed, the parameters must also be changed to ensure a valid optimizer is generated.

## Experimental setup

Generate 5 random networks and train them. Then, apply crossover to two of them and one of each mutation to the remaining 3 networks. Then, retrain the newly generated networks.

The networks should be trained on the mnist dataset, for 50 epochs, following the same procedure as used in the practical classes (showing the training and validation loss and accuracy per epoch).

## Evaluation

- Defining the grammar: `2pt`
- Generating a network genotype: `2pt`
- Generating an optimizer genotype: `2pt`
- Generating a network phenotype: `3pt`
- Generating an optimizer phenotype: `3pt`
- Ensuring the networks have between 1 and 50 layers: `1pt`
- Implementing the genetic operators: `5pt`
- Following the described experimental setup: `2pt`

## Rules

Each group must submit a jupyter notebook that can be run on google colab, and a 3-page **maximum** report **in pdf** format.

The notebook should be organised into easy-to-follow sections and may include some text if deemed necessary (this text should only be used to highlight certain details, as the rest should be included in the report).

The report should contain clear and easy-to-follow descriptions of the classes and functions created in the project and brief mentions of the theoretical motivations used for uncommon parameter choices.

The projects need to be submitted on moodle by the end of the 11th of June (23:59).