Task

1. Train a neural network

* Train a neural network in Pytorch on the IRIS dataset
* The dataset is, for instance, available as part of the [scikit-learn package](https://scikit-learn.org/stable/) or on this [site](https://archive.ics.uci.edu/dataset/53/iris)
* The network should have 4 Input- and 3 Output-Neurons without activation function. Furthermore, it should have two hidden layers with 10 neurons each. All “hidden” neurons should have ReLU as their activation function.
* The network should be trained for 200 epochs using a learning rate of 1e-3 and cross-entropy loss.
* Export the network to the [ONNX](https://onnx.ai/) file format. Pytorch provides a function for the export after installing the onnx and onnxruntime packages (via pip install).

2. Implement a neural network verifier

* Get familiar with adversarial robustness and the encoding of a neural network in LRA. See lecture slides or this [textbook](https://verifieddeeplearning.com/).
* Implement a program that takes a neural network (as an onnx file), a data point, and a real number epsilon as input. The output of your program should be “verified” if the network is robust for the given data point and epsilon. It should output “counterexample” and a counterexample if the network is not robust for the given data point and epsilon.
* Implement the verifier yourself, do **not** build an API for an existing tool
* Use [Microsoft’s Z3](https://github.com/Z3Prover/z3) as a constraint solver
* As distance function for adversarial robustness use the L∞-norm (elementwise distance)
* It may be helpful to
  + First convert the onnx file back to a pytorch model (e.g., with the [onnx2torch](https://pypi.org/project/onnx2pytorch/) package)
  + Write a function that takes a Z3 solver, a data point and an epsilon as input and adds the precondition for adversarial robustness to the solver
  + Write a function that takes a Z3 solver, and a neural network as input and adds the encoding of the network to the solver
  + Write a function that takes a Z3 solver, and a data point as input and adds the postcondition for adversarial robustness to the solver

3. Verify a neural network

* Verify the neural network you trained in Step 1. Use the first data point from the IRIS dataset and
  1. epsilon = 0.1
  2. epsilon = 2
* Find the maximal epsilon for which your network is still robust (you may use a modified version of your code from Step 2)

4. Documentation

* Provide a README for your program from Step 2. Thie README should contain how to “install” your program (contain information on the required packages) and how to run the code.
* Provide good documentation for all your code (Steps 1-3).