# Machine Learning for Signal Processing

*[5LSL0]*

Ruud van Sloun
Rik Vullings
Nishith Chennakeshava
Hans van Gorp

**Assignments Optimization and Regularization**

April, 2021

# Optimization and regularization

In this assignment you will investigate several optimization and regularization strategies for training nonlinear models.

## Error backpropagation

Consider the following model:

$$f(\mathbf{x}; \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{w}^{(2)}, b^{(2)}) = \left(\mathbf{w}^{(2)}\right)^T \max(0, \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + b^{(2)}. \tag{.1}$$

In the previous assignment (Nonlinear models), you learned to derive the gradients of a cross-entropy cost function with respect to the model parameters by back-propagation using the chain rule. We will exploit these gradients to update the parameters through various optimization strategies:

- Stochastic gradient descent, SGD (default settings: $\mu = 0.0001$)
- SGD with momentum (default settings: $\mu = 0.0001, \rho = 0.9$)
- AdaGrad (default settings: $\mu = 0.0001$)
- RMSprop (default settings: $\mu = 0.0001, \rho = 0.999$)
- Adam (default settings: $\mu = 0.0001, \rho_1 = 0.9, \rho_2 = 0.999$)

We will start however with implementing error backpropagation. For this part of the assignment, you need to download a few files from Canvas:

a) `backprop_template.py`

b) `backprop_test_script.py`

c) `data.h5`

The script `backprop_test_script.py` can be used to test your implementation of error backpropagation (from now on referred to as the more ”popular” `backprop`). You will see that this test script already shows a basic implementation of the SGD optimizer. To run the test script, replace in `functions = __import__("backprop_example")` the text `"backprop_example"` and make sure you import your own backprop implementation. Use as a filename your first name, last name, and ID as indicated in the header of the `backprop_template.py` file. The name of one of the team members only is enough. To keep things clear, use this name for all Exercises in this assignment. The data file `data.h5` is a data file in hdf5 format, with keys "X" and "y", $X$ being the input data and $y$ being the classes.

**Q1: Implement your backpropagation algorithm in Python** Complete all empty fields/functions in the `backprop_template.py` file. To guide you through the process, read the header of the file carefully. For the mathematical description of the various gradients, use your answers to the last Exercise of the Assignment on Nonlinear models (in case you think that your answer was wrong, here is a chance to correct it).

Test your solution with the test script: if the gradients propagate correctly through the network, the loss should decrease. Make a plot of your training loss and use this plot in your report. Also, upload the completed and renamed `backprop_template.py` file to Canvas.

Note: While the choice of where to place the sigmoid is arbitrary in most cases, please do include it as a part of your network, rather than your loss function for this assignment.

**Q2: For a correctly implemented backprop, do you expect the loss need to monotoniously decrease? Why (or why not)?**

## Optimization

**Q3: Explain how and why "SGD with momentum", "AdaGrad", "RMSprop", and "Adam" improve convergence w.r.t. plain SGD. What are their advantages and possible disadvantages?**

In the next part of this Assignment, you will implement several optimizers. These optimizers will use the output of your backprop function. Naturally, we will assess your implementation of the optimizers in the coming Exercises and not the implementation of backprop (we did that already in the previous Exercises). Still, in case your backprop is wrong, your optimizers – even when you implemented them correctly – might show unexpected behavior. In case of doubt about your backprop, use the file `backprop_example.pyc` that is available on Canvas, instead of your own file. This should allow you to verify for yourself whether your implemented the optimizers correctly.

For the following Exercises, you need to download a new set of files from Canvas:

a) `optimization_template.py`

b) `optimization_test_script.py`

As data, you can use the same data as for Exercise 1. Similar as for Exercise 1, implement your optimizers and regularizers by completing the `optimization_template.py` file and **upload the completed file to Canvas**.

**Q4: Implement all of these solvers for the above model (default parameters), and evaluate the convergence of all parameters, as well as the cost as a function**

**of the number of iterations. Plot these convergence graphs, include them in your report, and discuss the results. Also include a scatter plot of the input data points ($x_1$ vs $x_2$) and their classification. Note: use at least 50000 iterations.**

**Q5: Show what happens if you significantly increase or decrease the learning rates, and discuss your findings.**

## Regularization

**Q6: Implement both $\ell_2$ and $\ell_1$ regularization strategies, and evaluate their impact on the convergence of the parameters using the Adam solver with $\lambda = 0.001, 0.01$ and $0.1$. Plot the convergence graphs, include them in your report.**

**Q7: Explain your findings (compare to the convergence graphs of Q4), and explain in which cases either of these regularization strategies would be appropriate.**

**Q8: Explain "dropout" regularization, and explain why it would or would not have been an appropriate strategy for the model described above.**

## Appendix: Useful Functions and Short-hands

Below are some potentially useful functions, and short-hands for the completion of the assignment.

| Description | Math | Python Code |
|---|---|---|
| Matrix multiplication | $y = Ax$ | y = np.matmul(A, x) |
| Alternative matrix multiplication | $y = Ax$ | y = A @ x |
| Exponent | $c = \sqrt{a^2 + b^2}$ | c = (a**2 + b**2)**0.5 |
| Return the current shape of an array | n.a. | dimensions = A.shape |
| Transpose of a matrix | $A^T$ | A.T |
| Inverse of matrix | $A^{-1}$ | np.linalg.inv(A) |