# DL - LAB 4

## Analysis of the obtained results

Task 2:

Running task 2

h = 1e-6

Analytical vs Autograd RMSE: 2.5136e-16

Analytical vs Central RMSE: 1.9891e-10

h = 1e-5

Analytical vs Autograd RMSE: 2.5136e-16

Analytical vs Central RMSE: 8.0686e-11

h = 1e-4

Analytical vs Autograd RMSE: 2.5136e-16

Analytical vs Central RMSE: 9.1850e-09

> Observation

- Autograd method is more accurate than central difference method due to the fact that autograd method uses the exact gradient of the function while central difference method uses an approximation of the gradient of the function.
- Also, we can observe that central difference method is sensitive to the value of h, if h is too small, the approximation of the gradient will be inaccurate, if h is too large, the approximation of the gradient will be inaccurate as well

Task 3:

Running task 2

Complexity 1 : f (x) = ∑xi^2

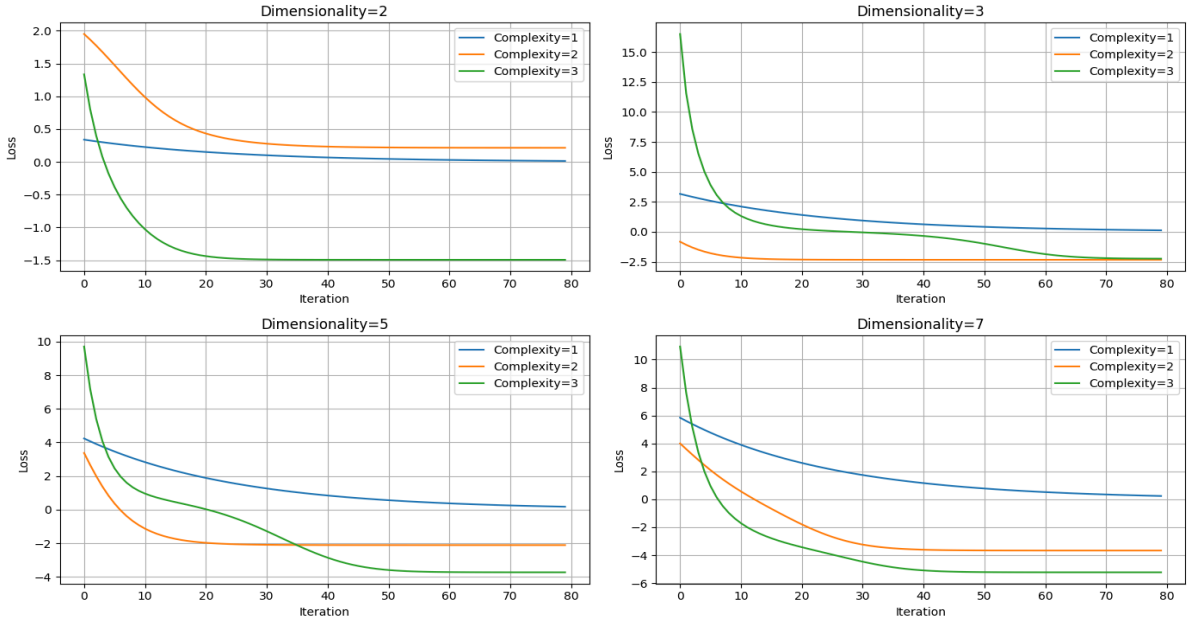
Complexity 2 : f (x) = ∑xi^2+∑sin(3xi)

Complexity 3 : f (x) = ∑xi^2+∑sin(3xi) + ∑xi^4

Final values>

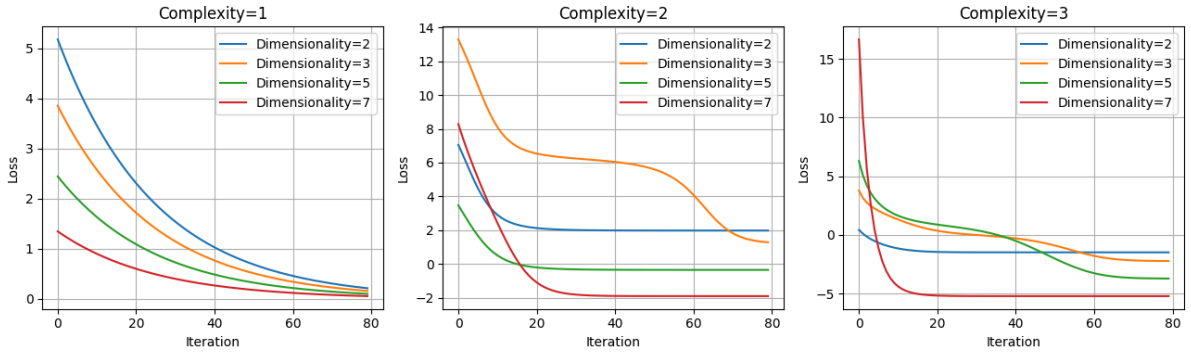
| FINAL LOSS TABLE | Dimension = 2 | Dimension = 3 | Dimension = 5 | Dimension = 7 |
|---|---|---|---|---|
| Complexity = 1 | 0.055883 | 0.095640 | 0.309576 | 0.215865 |
| Complexity = 2 | 0.214863 | -0.018648 | -3.879860 | -5.431815 |
| Complexity = 3 | -1.489983 | -2.239426 | -3.727675 | -5.225347 |

**Dimensionality>**



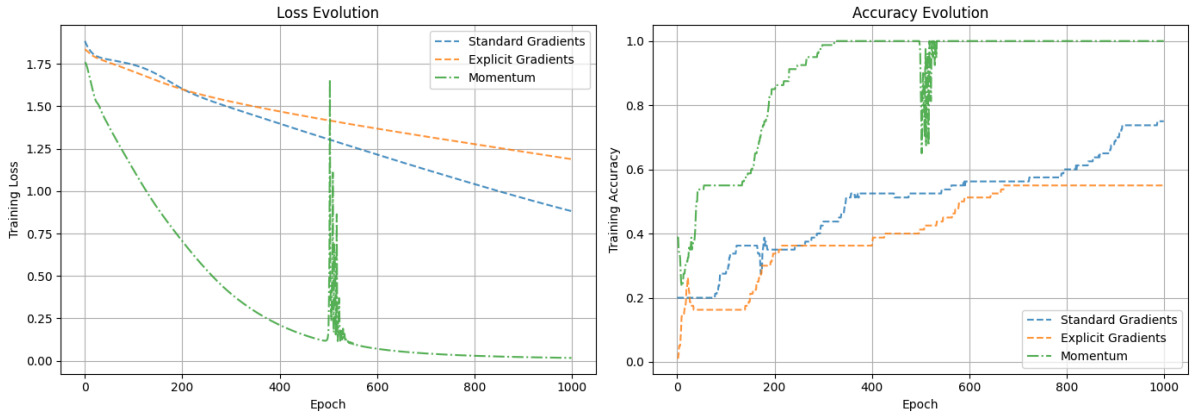

- For complexity level 1, as dimensionality increases from 2 to 7, the final loss values rise from 0.055883 to 0.215865, indicating that higher dimensionality may slow down convergence and lead to a suboptimal minimum.
- At the same time, for complexity levels 2 and 3, higher dimensionality leads to more negative final loss values, suggesting that gradient descent effectively minimizes these more complex functions in higher dimensions.

**Complexity>**




- As function complexity increases, gradient descent may face challenges such as navigating non-convex landscapes with multiple minima and adapting to varying curvature, which can affect the stability and speed of convergence.

Task 4:

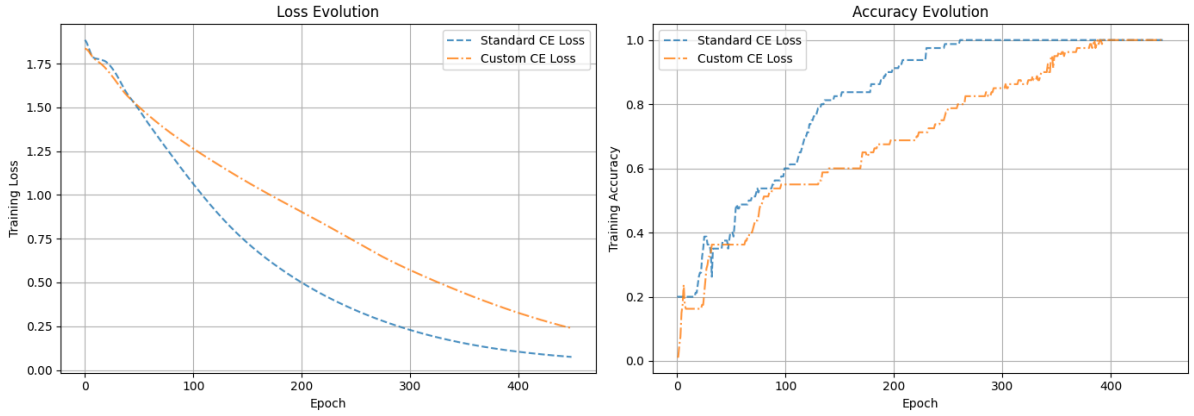



- The model trained with momentum gradients leads to faster, more stable convergence and a much higher final accuracy than simple gradient descent with explicit gradient calculations (this for both implementation, pytorch standard and scratch function written).

- For our implementation of simple gradient descent will be necessary around 2000 epochs to reach the same accuracy as the momentum gradients lead in 400 epochs, which shows an optimisation in time and computation of 5 times faster. That value evidentiate actually de importance of optimisation in training also.

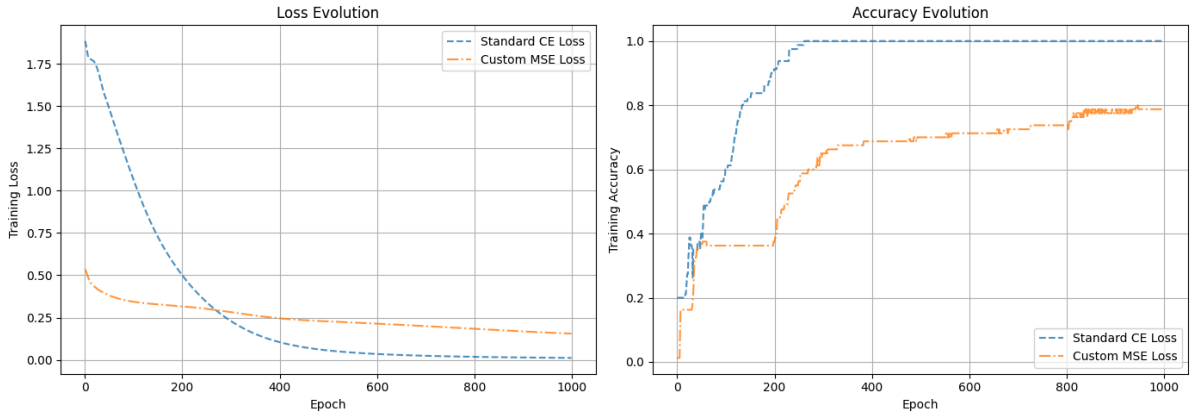
- Implantation from pytorch for simple gradient descent is only slightly better than our function with explicit gradient calculations, maybe due to internal optimisation from inside the framework

Task 5:




- From a first perspective, it seems that our custom cross-entropy function performs worse than the already implemented function inside the framework (`nn.CrossEntropyLoss()`), this one may also be caused due to framework optimisations from the inside it. But besides that, the result can be still compared, meaning that our implementation is still a reliable method.

Task 6:




- From a convergence perspective, the current implementation of the MSE loss function is clearly outperformed by the Standard Cross-Entropy Loss. MSE loss convergence decades much more slowly than CE loss does, also the accuracy goes if using CE loss, the model achieves an accuracy of 1.0000, highest it gets using MSE loss is 0.7000 on the test set, which is not great, but acceptable in comparison with Standard Cross-Entropy Loss.