Mariia Semenenko
Student ID: 22100679

Task 2. In the given code, the parameters a, b and c control the amplitude, frequency and damping factor of the signal respectively and are initialized as PyTorch parameters, which means they will be automatically optimized during training to fit the input data. The initial values for the parameters are 1.0 and they will be modified using stochastic gradient descent to minimize the mean squared error between the predicted function output and the actual voltage measurements. The adjustments made to the parameters can provide knowledge about the frequency response and damping properties of the circuit.

A parametric function a*cos(2*pi*x/b)*exp(-c*x) in the code is a common form of a damped sinusoidal signal used to model a wide range of physical systems, including oscillating electrical circuits. This function is applied to the input data using the model method, and then the resulting output is compared to the target values using the mean squared error loss function. The training loop runs for 10,000 epochs with a learning rate of 0.1. The function applies a cosine function to the input signal with a frequency of 2*pi/b, which corresponds to the circuit's natural frequency. The function also multiplies the cosine function by an exponential function with a negative exponent of c times the input signal, which corresponds to the damping factor of the circuit.

The fact a damped sinusoidal signal provides a small loss and a good fit to the data suggests that the underlying electrical circuit may be a second-order low-pass filter, which is composed of an integrator and an inverter with a feedback loop and a multiplier to scale the output. The damping signal indicates that there is some energy dissipation in the circuit, possibly resulting from resistance or capacitance factors. The noise in the voltage measurements could be due to the presence of high-frequency components in the input signal that are not filtered out by the circuit or due to the limitations of the measuring equipment.

The results can provide valuable insights into the behaviour of the circuit. By comparing the actual data with the predicted data, one can determine how well the model fits the measurements and assess the accuracy of the model. The loss value provides a quantitative measure of the error between the predicted and actual data. The given parametric function captures the oscillatory behaviour of the signal and the exponential decay, which are characteristic of a second-order system.

Task 3. Assuming the program needs to be very efficient on a CPU, the specific part of the code that could benefit from implementation in assembly is the loop over the epochs. The training loop comprises several computations, such as resetting gradients, predicting output, computing gradients, calculating loss, and updating parameters, each of which requires multiple instructions that can be optimized for performance. Implementing a loop in assembly can minimize the overhead of high-level language interpreters and compilers, resulting in faster and more efficient code. This would involve using low-level operations such as register manipulation, memory access and arithmetic operations.

However, the advantages of implementing a loop in assembly rely on the specific hardware being used. Different CPUs have distinct architectures and instruction sets, necessitating a customized implementation. If the hardware being utilized supports the x86 architecture, specialized instructions such as SSE (Streaming SIMD Extensions) or AVX (Advanced Vector Extensions) can be used. They are designed especially for mathematical operations on big sets of data, like those used in machine learning and signal processing. By using SIMD operations, these instructions can perform multiple operations at once, resulting in a faster processing speed than using regular instructions.

The parameterized function forward() is applied to the input data using PyTorch operations, which are optimized for performance on CPU architectures that support SIMD instructions. However, in some cases, it may be advantageous to manually implement some parts of the code using assembly language to optimize performance further. For example, if the size of the input data is very large, using SSE or AVX instructions to perform computationally expensive element-wise multiplication and exponential operations could significantly boost performance.

However, it is important to note that manually implementing parts of the code using assembly language and SIMD instructions can also introduce additional complexity and reduce code maintainability. As such, this approach should only be considered if the performance gains outweigh the additional development time and complexity.