2a. It abstracts away some details that may be cumbersome to deal with. Two simple examples are that it randomly shuffles the data for each epoch and allows the use of batch sizes that do not evenly divide data(by setting `drop_last` to True). Another advantage over a simple list of Tensors is that it is an iterable object. A list of Tensor objects would consume an excessive amount of memory; an iterable object only loads the batch of data that would be processed and then throws it away, i.e., does not store it in runtime memory. All of this wouldn't be too bad to implement on our own; however, combining all the small advantages that it allows creates a valuable abstraction and allows us to concentrate on stuff that matters more.

2b. The most significant observation is that the model seems to converge faster with smaller batch sizes and tends to train faster with larger batch sizes. However, the difference in training time was not as significant as I had expected. It's likely due to the fact that the model is running on a CPU on my PC, and it does not have the parallelization benefits of a GPU. In terms of the number of examples alone, I realized that as long as there is a sufficient number of examples, the model manages to fully learn the dataset with a reasonable batch size. So, more data does not seem to overfit the model, though I did not do an experiment with more than 50,000 examples.

2c. The two values that I used as the batch size are: 20 and 200 with 2,000 number of examples. The latter one runs in 1.8 seconds, the initial one in 4.3 seconds. While I'm not sure exactly if pytorch does optimizations related to parallelizations on CPUs, generally a very small number that does not likely align well with the computer architecture could lead to more cache misses etc.. However, I know that the reason why larger batch sizes lead to significant improvements on GPU is due to parallel computing, i.e., making more use of hardware resources concurrently.

2d. The two batch_size values I used are 500 and 5,000, with a number of examples being 5,000. In my CPU case, I think it is because 500 probably aligns better with the computer architecture. In the GPU case, after a certain number, the batch size becomes too large for the GPU to compute all the examples concurrently.