Introduction to machine learning, WS 2021/22

Problem 1 – part A

Description: An industrial company uses a production line to transform raw material into final product material. Since quality of the final material is critical for the company customers, the company relies on a manual procedure for quality checking before the final material is released. Though very precise, it is also very slow and expensive (requires the work of a human expert).

The quality of the final material clearly depends on the quality of the raw material (the company gets the quality indicators from their supplier) and the quality of processing on the production line. The company installed sensors onto the production line to monitor the process automatically and wants you to use these data to predict the final material quality.

In the `ProductionQuality.zip` file you will find two csv files: `data_X.csv, data_Y.csv`.

`data_X.csv` contains the sensor measurements from the production line and the raw material quality indicators. The production line consists of 5 chambers each equipped with three temperature-measuring sensors. The raw material quality indicators are its height and moisture content. Data are collected every minute and the measurements are synchronized so that in each line in the dataset we have the respective sensor data from all 5 chambers for the material rolling out of the production line at the given Datetime index.

`data_Y.csv` contains results of the final material manual quality measurements for selected Datetime indexes (every 2hrs).

Your task: Use the available data to train a linear model to predict the quality.

Details of your task: You shall create a jupyter notebook in which you describe your steps (in plain English in markdown cells) and put your code (in the code cells) so that that your results can be reproduced when I (or anybody else) re-rerun the complete jupyter notebook.

Implementation rules: All work shall be done in Python (no Excel preprocessing) using PyTorch as the main library of choice. This means ***you should create your data objects as torch tensors*** (not numpy arrays) and use standard torch operations for preferably all your work (except the data alignment step below where you may want to use pandas). ***For this problem, you are not allowed to use any of the following packages `torch.nn, torch.optim, torch.utils`.*** (We will explore them later.) You can develop your code completely inside jupyter notebook or with parts outside as a standalone code (e.g. a module to be imported). In the latter case, make sure the notebook really can load the module and executes as expected.

Teams: Create teams of two. You shall be working on this and all the following problems included in the portfolio together. (If you cannot find a team member to work with you, please let me know as soon as possible.)

Submission: Submit the complete jupyter notebook via e-learning (only once for the team of two). ***The deadline is Sunday 14.11. 23:59***.

Questions: If you have any, please ask me right away (preferably via mattermost or alternatively email). Don't wait till we see each other in the course, it may be too late for you then.

Detailed list of steps to follow:

1. Align the data: You will see that you can use only part of the complete data. Remember that for training a supervised model, you need to have for every input $x^{(i)}$ the corresponding output $y^{(i)}$. You may want to use the python in-built `csv` library or the dedicated data-handling package `pandas` to load the csv files and align the input and output data. How exactly, you should figure out on your own (googling/ reading docs).
2. Write a generator that yields a batch of examples on every iteration. The generator shall use as inputs three arguments: input data, output data and the size of the batch. Each outputted batch shall be a tuple of the inputs and outputs. Remember that the assignments of the data to the batches shall not be always the same across the epochs (so you need to shuffle your data before you start yielding the batches) and that you should cater for the case that the last batch may be incomplete (smaller).
3. Define the linear model. This shall be a function which takes as arguments the input data and the model parameters (weights and bias) and returns the output predictions.
4. Define the loss function as the average squared error of the predictions – a function taking in the predictions and true labels and returning the value of the loss.
5. Define the optimization algorithm - minibatch stochastic gradient descent. Here you need to define a function for a single step of the update, that is function which takes in three arguments: the parameter you wish to update (assume that this contains the gradient), the learning rate and the batch size. The function shall return the updated value of the parameter after the following update step
$$\theta \leftarrow \theta - \frac{\alpha}{|\mathcal{B}|}\nabla_\theta$$
where θ is the parameter, α is the learning rate, $|\mathcal{B}|$ is the batch size, and ∇_θ is the parameter gradient. A few more tricks you need here (and we have not discussed yet):
    - Remember that PyTorch automatically keeps track of all operations performed on parameters for which you have set `requires_grad=True`. It records these operations into the computational graph so that if you later ask for calculating the gradients using the `backward()` command, it can follow the computation graph in the backward direction through all the functions and calculate the gradients - it can ***backpropagate***. The computational graph shall track the operations you do in your model, in your neural network. The update of the parameters through the SGD is an operation in the optimization procedure and as such shall not be part of the computational graph - it has nothing to do with the model we defined higher up. Hence, you need to disable the operation tracking temporarily. You do this by using the `with torch.no_grad():` block. Operations within this block will be executed but will not be recorded into the computational graph and hence will have no effect on the parameter gradients. All operations in your sgd step function shall be included within the no_grad block.
    - Once you have updated your parameters using the gradient, you will no longer need the gradient values and it is a good habit to set them to zero. Remember that the gradient is contained within the parameters itself and you can get it by calling `param.grad` (where param is the `torch.tensor` with the parameter). To set the gradient to zero, you can use `param.grad.zero_()` at the end of your sgd step update.
6. Write the model training procedure. This is what you will need to do:
    a. Fix manually the hyper-parameters of your training: learning rate (this should be something rather small, e.g. 0.01), number of training epochs – each epoch is one run through all your

training data (e.g. 100), batch_size (e.g. 256). You can set these anyway you like, these are just suggestions.

b. Initiate randomly the model parameters (weight and bias). Remember to require the gradient calculation for these.
c. You will iterate through the complete dataset for the selected number of epochs.
d. In each epoch you will iterate through all the data batches produced by your batch generator (defined in point 2 above). For each batch, you need to:
    i. Pass the batch inputs and outputs to you lin reg model (defined in point 3 above) and get predictions
    ii. Use your loss function (defined in point 4 above) to get the loss value
    iii. Use pytorch autograd to get the gradients by calling `loss.backward()`
    iv. Use your sgd step function (defined in point 5 above) to update the parameter values.

   At the end of the epoch (after iterating through all the batches) it is a good habit to print out the value of your loss for monitoring. You may want to print out the loss of the very last batch, or the average across all the batches within the epoch. It does not matter too much, it is just for monitoring the loss evolution – the loss should be gradually decreasing (that's what we want, have the loss ~ prediction errors as small as possible) though not necessarily at every step, every epoch. But it should have a generally decreasing tendency. If it does not, or if you are getting inf, nan or similar results, it's an indicator that either you have a bug in your code (-> find it and correct it) or that your leaning rate is too high (-> make it smaller).

7. The very last step after the training is to use the trained parameters for predictions on the whole dataset. Iterate once through all the data examples (through all the batches) and calculate the total average loss (as the avg of the batch losses). This is the final indication of your training error – print it out. (Note: this step is not part of the model training, you will never need the gradients over these operations so you shall turn of again the gradient monitoring via the `torch.no_grad()`).
8. Play with the values of the hyper-parameters, try different combinations and see how good/bad your model training is. Write a short text (directly in the jupyter notebook) describing what you discovered, what is a reasonable learning rate and how many epochs you need to train to a good value of the loss (low loss).