Collaborative filtering

Learning steps

- [x] ~~yt video~~
- [x] ~~notebook/own implementation~~
- [x] ~~book chapter~~

Resources

- website 💻
  - lesson 7
- notebooks 🗒️
  - Collaborative Filtering Deep Dive
  - Road to the top: part 3 and part 4
- book 📘
  - chapter 8
  - solutions to exercises

Quick notes

YouTube video (→ link)

- we usually tweak the first or last layers
  - we'll take a look at the rest later on
- road to the top part 2 (RTTT 2)-
  - ConvNext model used to start with + some pre-processing (TTA)
- RTTT 3
  - we'll use larger models ⇒ more parameters
    - they can find more patterns etc
  - the gradients are more numerable and take up more computation space on the GPU

  - potential memory issues when training a model on a GPU (*CUDA out of memory* e.g.)
    - how much memory will a model use? → use func report_gpu() to find out
    - to solve it: use GradientAccumulation
      - we define a batch size to be divided by a *accum* number
      - we then consider a training loop (compute loss, backward on it, subtract, reset)
        - sometimes, we change the training loop: the gradients are accumulated instead of being reset each time

        > 💡 The difference between GPU classes (RTX3070 Ti, 3080, etc) is not the performance but the memory size. On a smaller GPU mem, use smaller batch size and gradient accumulation!

        - 💡 rule of thumb for batch size and learning rate
          - batch size: as big as possible. can use multiples of 8 or powers of 2
          - learning rate: if batch size divided by 2, same for LR (not always perfect)
      - ❗ transformers use a fixed image size, make sure to resize the images to squares of this size
- RTTT 4
  - we'll now work on the last layer of the neural net
  - we'll use a DataBlock (one level deeper ⇒ more flexibility)
    - in Pandas, you can specify an index column for a dataframe → you can then use this df as a dictionary
    - we want a model that predicts 20 things: the 10 diseases and the 2 varieties
  - cross-entropy loss
    - softmax
      - the model must say which one of the available categories it is ⇒ must choose one, no place for uncertainty because it adds up to 1
      - we obtain the prob for each category as an output
    - target value (one-hot encoded)
    - then, we sum up across all categories and multiply the target value by the log of the predicted prob (result of softmax)
      - we compute it for every row and add it up
- collaborative filtering deep dive (pretty much chap 8 of the book)
  - goal: recommandation system for movies based on user preferences (ratings)
  - we don't have info about people preferences, so we'll use SGD to find it out (in excel)
    - we decide (randomly) that we have important five factors (parameters instantiated randomly) for each user <u>and</u> for each move

- we make the .product by matrix multiplication to create random predictions for each user for each movie ⇒ we need to optimize the parameters
- we use root mean square error (RMSE)