Fachprojekt Machine Learning
Sebastian Konietzny, Marc Höftmann

Wednesday, 22. May 2023
Group 1 & 2 **DUE Wednesday, 12. June 2024**
Group 3 & 4 **DUE Wednesday, 19. June 2024**

Task #3

Feel free to ask questions via our chat program Mattermost. **Keep the deadline in mind!**

Your task is to implement the first paper (MADE, VAE, GAN or NICE). The Fachprojekt is different from other modules, so you have to coordinate mostly by yourself. We recommend to use at least a chat program like Discord or Mattermost. Also, feel free to program the code together, as it makes it usually easier. But stick to the contribution guidelines below. Your up-to-date code should be in the `main`-branch on GitHub.

Very important: Log your contribution on GitHub!

1. Journal: "20.04. - Marc: Implemented X","02.05. - Sebastian: Add Y"
2. Commits: If you do pair-programming then split code in multiple commits. Each commit with a **different** GitHub account!

1. Datasets

Create the file `datasets.py` and set up your data pipeline correctly.

(a) Add all the functions that you need to satisfy the requirements below.

(b) Load your dataset (MNIST or CIFAR-10) using PyTorch. Load the train and test dataset.

(c) Implement the pre-processing of the images. Are the images normalized? Are they ready for training? Use PyTorch transforms if necessary.

(d) Create the dataloaders for training and test dataset and set their parameters as good as possible.

(e) Create a function `'def visualize_dataset(dataloader):'` to visualize images from MNIST and CIFAR-10 dataset. You can use `matplotlib` for visualization.

(f) Implement the function `'def create_dataset(dataset_name):'` and return the dataloader.

2. Neural networks

Create the file `networks.py` and build your neural network(s).

(a) Add a class for each network or component. It needs to inherit from `torch.nn.Module`. E.g.:

- Masked-Linear-Layer, MADE, etc. (Autoregressive)
- Encoder, Latent, Decoder, etc. (VAE)
- Generator, Discriminator, etc. (GANS)
- Inverse, etc. (Flow)

(b) `(Group-specific)` Add own classes for sub-components of your network.

- **Hint**: A class could also inherit from `torch.nn.Linear` or `torch.nn.Conv2d` which then inherits from `torch.nn.Module` automatically. Don't forget to call the `super()` constructor or respectively `super(...).__init__(...)`.

(c) Implement the class function `'def forward(self, x):'`.

- `(Group-specific)` Add meaningful custom functions for your network class to modularize the `forward(self, x)` function.

3. **Sampling**

Create the file `sampling.py` to generate new images.

(a) Add the function `'def sample(...)'`.

- **Hint:** For testing, you can use this function also with an untrained neural network, but the results will look like random noise. For good results, plug in your trained model later.

(b) Use your neural network(s) to generate up to 100 images.

(c) Show or save all generated images by constructing e.g. a 10x10 grid using `matplotlib`.

(d) Save the grid as a big image (`.png or .jpg`).

Example

4. **Training**

Create a jupyter notebook `train.ipynb` to train your network.

(a) Load all necessary components from `datasets.py, networks.py, sampling.py`.

(b) Show some images from the training dataset.

(c) Create a cell at the top which includes all relevant hyperparameters:
e.g. `lr, batch_size, num_epochs, lr_strategy, network_size, etc.`

(d) Get the dataloader, create your optimizer and setup everything for training.

(e) Implement the main training loop. Do **not** use the test dataset for training! In general, implement the following steps:

i. Load a batch of data.

ii. Forward pass.

iii. Compute loss.

iv. Backward pass.

v. Optimization.

vi. Repeat.

(f) Evaluation

- Create a file `evaluation.py` and write a function `'def evaluate(...):'` to calculate the mean loss for the entire test dataset. Do **not** optimize the model with the test dataset.
  - **Hint:** Use a context manager: "`with torch.no_grad():`".

(g) Logging

- Create a nice-looking console printing to track the training progress.
- After each training epoch, log the current test error. Measure and print the time that has elapsed in order to work on performance improvements later.
- After each training epoch, generate 5 images using the `sample` function und log them.
- Frequently (e.g. every 150th batch), print the training loss and the current learning rate.
- After training: Create and show a 2D plot. Log the train (line) and test loss (scatter) on the y-axis in a single graph. For plotting, only use the printed training losses and use the test loss after each episode (calculating test loss is a bit computational expensive).
- After training: Create another 2D plot that shows the learning rate over time.
- `(Group-specific)` After training: Create a 2D plot for each loss component, where the loss value is on the y-axis. Use the same amount of loss values as for the full loss (every 150th batch).

(h) Advanced techniques and code optimization. When your neural network is successfully learning, consider these improvements to increase training speed and quality.

- Consider the improvements suggested in `PyTorch_optimization.ipynb` (can be found in our GitHub repository).
- Implement at least two different learning rate decay strategies to choose from e.g. `Cosine decay`, `Step decay`. Also, keep the no decay strategy.
- Implement the ability to choose different optimizers e.g. `SGD`, `RMSProp`, `Adam` or `AdamW`.

Appendix

Useful methods / classes using PyTorch. Look through the documentation[1] for a full picture.

- `torch.nn.ReLU`, `torch.nn.LeakyReLU` or `torch.nn.BatchNorm2d`
- `torch.optim.Adam, torch.optim.SGD`
- `torch.nn.MSELoss, torch.nn.CrossEntropyLoss, torch.nn.NLLLos, torch.nn.BCELoss, torch.nn.BCEWithLogitsLoss, torch.nn.KLDivLoss`
- `torch.randn(...)`
- `torch.nn.utils.masked_fill`

[1] `https://pytorch.org/docs/stable/index.html`

- `self.register_buffer(...)` in a `nn.Module` subclass
- `super(<subclass_name>, self).__init__(...)` in a `nn.Linear` subclass
- `torch.permute`, `torch.reshape`, `torch.multinomial`
- `tensor.detach()`, `tensor.cpu()`, `tensor.numpy()`
- Also, look into `PyTorch_optimization.ipynb` (GitHub repository)