Lab 6: Feed-forward Variational Auto-Encoder

In this lab, we will be discussing how to implement a variational auto-encoder in pyTorch. In Part 1 of this lab, we will start with a tutorial packaged in Torch (on how to use VAEs on MNIST digit dataset). Then, in Part 2, we will modify the model for a melody generation task.

Datasets

1. MNIST
   Available at https://pytorch.org/vision/0.8/datasets.html#mnist

2. A dataset of chord/melodies provided at http://marg.snu.ac.kr/chord_generation/

   Lim, Hyungui, Seungyeon Rhyu, and Kyogu Lee. "Chord generation from symbolic melody using BLSTM networks." arXiv preprint arXiv:1712.01011 (2017).

Preparing Environments

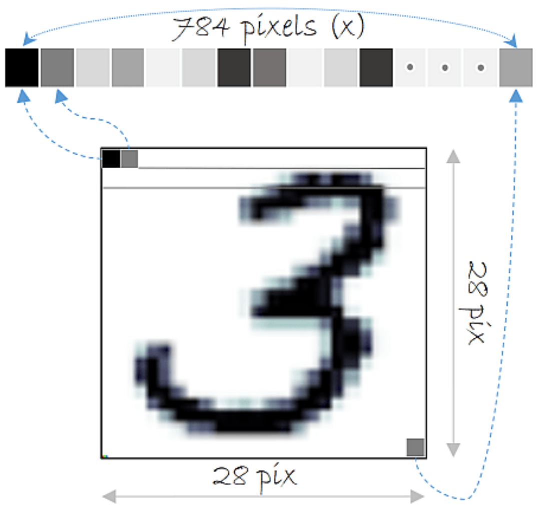
For this lab, you will be needing the following Python libraries:
1. PyTorch and TorchVision: https://pytorch.org/get-started/locally/
2. Pandas: https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html

Note: You may need other dependencies that are not mentioned here. A Docker image will also be provided sometime in the next week, however, please ensure that you have these dependencies installed in case the preparation of the Docker image gets delayed.

Exercise 1: VAE Implementation (for MNIST data)

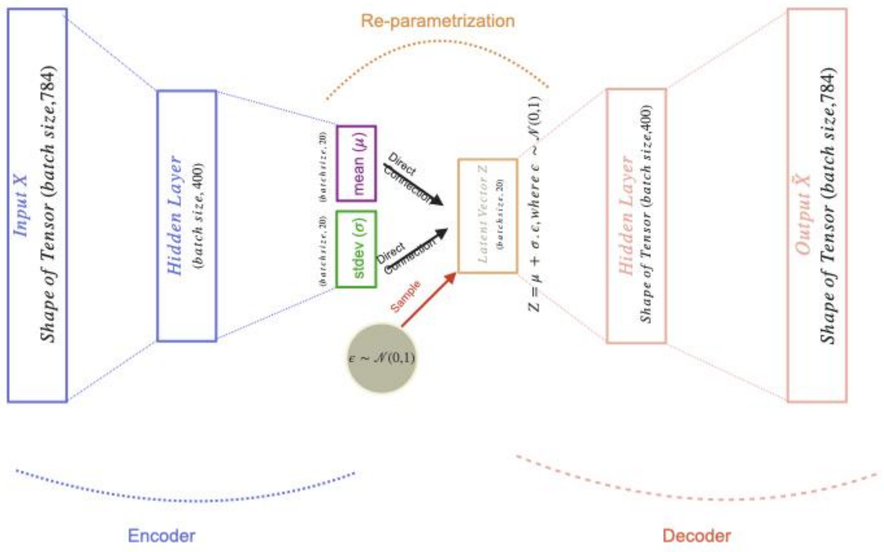
As mentioned, this is just a demo of Torch's tutorial on a basic VAE implementation using feedforward neural networks. The scheme of the implementation is as follows:

1. Flatten images as follows:



https://towardsdatascience.com/using-object-detection-for-complex-image-classification-scenarios-part-3-770d3fc5e3f7

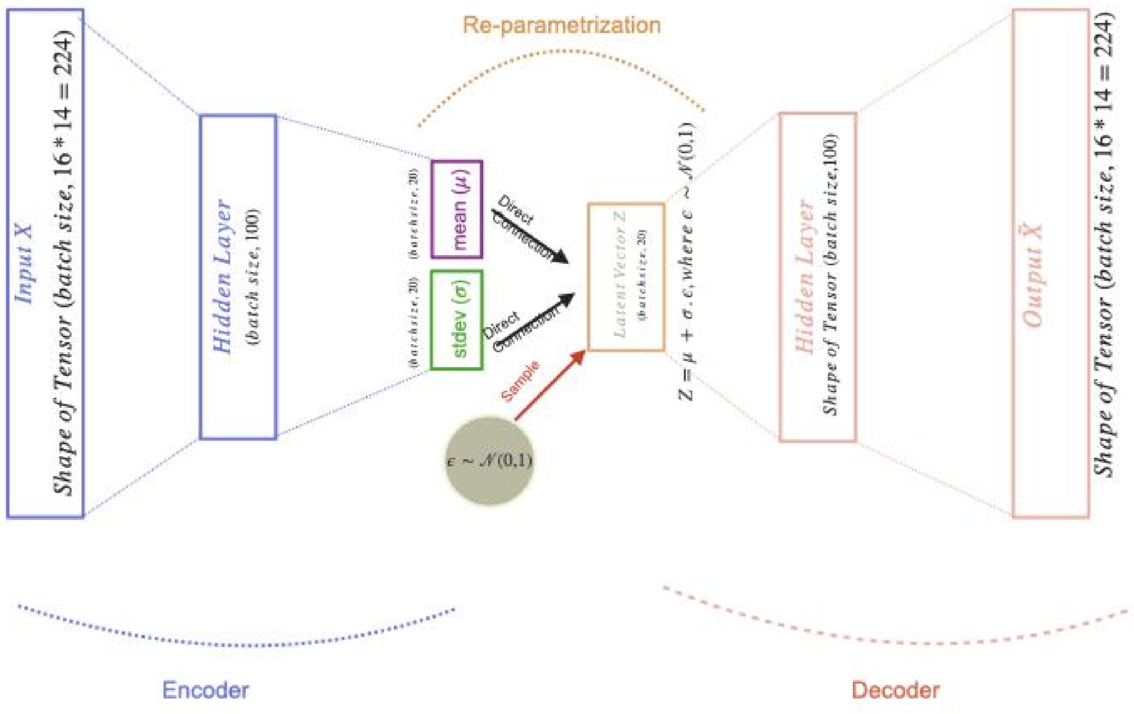
2. Train the following VAE on the flattened vectors/matrices (called tensors in torch)



Exercise 2: VAE Implementation (for Melodies)

A slight modification of the model in Exercise 1 will be used for generating melodies. Same as the images in the previous case, the melodies (initially a 3D tensor of batch size x time step x pitch classes per step) are flattened into 2D tensors of batch size x (time step x pitch classes per step). In this exercise, we extract 1 bar melodies from the dataset and use them as our input/output tensors in the VAE architecture. A more detailed overview of the melody representations can be found on the following page.

The architecture in this exercise is similar to the previous exercise, with the exception of the tensor dimensions:



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Rest	🟩	🟩	🟩	🟩					🟦	🟦	🟦	🟦				
A																
Bb/A#																
Cb/B																
B#/C					🟥	🟥	🟥									
C#/Db																
D								⬜								
D#/Eb																
E/Fb													🟨			
E#/F														🟪		
F#/Gb																
G															▨	▨
Ab/G#																

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Hold		1	1	1		1	1			1	1	1				1
1	Rest	1								1							
2	A																
3	Bb/A#																
4	Cb/B																
5	B#/C																
6	C#/Db					1											
7	D								1								
8	D#/Eb																
9	E/Fb													1			
10	E#/F														1		
11	F#/Gb															1	
12	G																
13	Ab/G#																

Measure	Note	Duration (multiple of 16th Note in 4/4)	Cumulative Sum	Start Position
2	Rest	4	4	0
2	C	3	7	4
2	D	1	8	7
2	Rest	4	12	8
2	E	1	13	12
2	F	1	14	13
2	G	2	16	14

Tasks for Submission:

Using the code provided in Exercise 2 and the Jupyter Notebook:

1. Install all dependencies and make sure that the provided exercises work on your local computers (No need to test the MNIST exercise, just focus on the Melody VAE exercise) [2 Marks]
2. Using this tutorial, implement saving and loading of the torch models (https://pytorch.org/tutorials/beginner/saving_loading_models.html) [2 Marks]

3. Grab two 1-bar melodies that are in a major scale (either write the melodies yourself or use the test set). Using the encoder in exercise 2, encode each of the melodies. [3 Marks]

4. Find 5 equally distanced z-vectors between the embeddings of the two melodies above. Pass each of the intermediary z-embeddings to the decoder and store the decoded melodies associated with the embeddings [2 Marks]

5. Create an image of the interpolation from one pattern to the other. [1 Mark]

As always, please don't forget to submit a video (if no audio commentary, make sure you submit a supporting document or add proper comments to the Jupyter Notebook) showing your approach and explaining your code.

For Parts 2B to 5, use the provided Lab6_Exercises.ipynb notebook

BONUS: Write a function that exports the generated samples to a midi file!