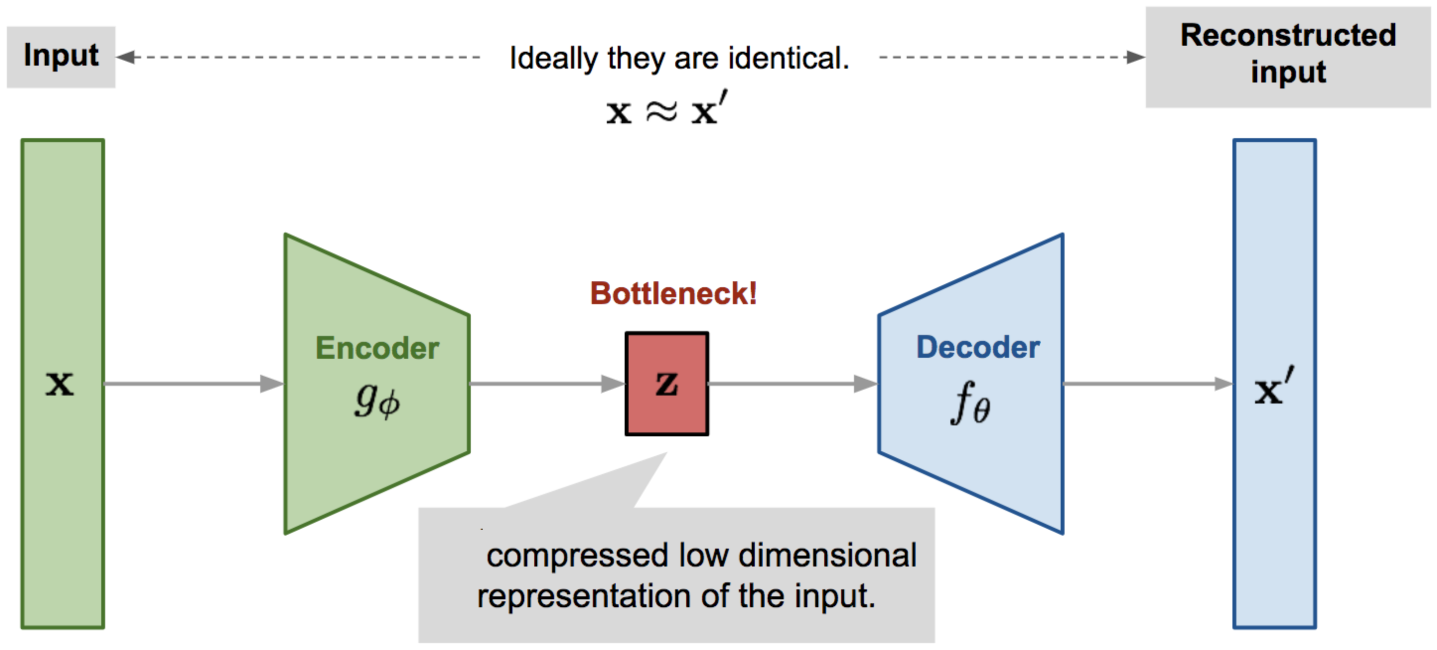
CMPUT 328 Assignment 9

Semi-supervised learning with Autoencoder

Goal: Implement a semi-supervised learning algorithm using compressed features from an autoencoder by adding a classification branch to it.

Introduction: An autoencoder is a special type of neural network architecture that comprises of two parts: an encoder and a decoder.
The encoder compresses the original input into a latent-space representation that has lower dimensionality than the input. This is usually implemented using multiple consecutive fully connected or convolutional layers whose final output dimensionality is smaller than that of the input.
The decoder does the opposite of the encoder, i.e. reconstruct the original input from this latent-space representation via multiple consecutive layers that have larger output dimensionality than their input and same as the original input of the encoder.
The autoencoder is then trained to minimize the difference between the original input and its reconstruction via L2/L1… loss.


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
compressed low dimensional representation of the input.

The picture above shows the general structure of an autoencoder. More details about autoencoders can be found here:
https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f
https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726
https://www.deeplearningbook.org/contents/autoencoders.html

In this assignment, you need to design an autoencoder with a classification branch that will consist of one or more fully connected layers like the last 3 layers of LeNet and will start from the features $\mathbf{z}$ of the encoder.

The new loss will be a combination of the L2 reconstruction loss for the autoencoder and the cross entropy classification loss for the classification branch. There is one catch, though. In the semi-supervised learning problem, only a part of the data has classification labels on it. For example, in this assignment, **only 20% of images have classification labels** for training. In this case, the loss function will have no classification loss for unlabeled images. The loss function for each image looks like this:

$$L = ||\hat{x} - x||^2 + C * CrossEntropyLoss(F(z), y)$$

The first term is the reconstruction loss where x is the input image while $\hat{x}$ is the reconstructed image produced by the autoencoder. The second term is the classification loss where z is the encoded feature vector, $F(z)$ is a shallow neural network that classifies the input image x and y is the classification label. C is a scalar that depends on whether the image is labeled or not: $C = 0$ when image is unlabeled and $C = C_0$ otherwise. C_0 is a positive value that controls the relative weightage of the two losses. Finding an optimal value for C_0 is a part of this assignment.

Task: You are provided with two files: **A9_main.py** and **A9_modules.py**. Your task is to create the autoencoder, including the classification branch, by completing three classes - **`Encoder`**, **`Decoder`**, **`Classifier`** - that define its different components. Both **`Decoder`** and **`Classifier`** receive input from **`Encoder`** so their input sizes must match the output size of **`Encoder`**.
You will be training and testing your model on two datasets: **MNIST** and a more difficult version called **Fashion MNIST** that contains images showing items of clothing instead of handwritten digits.
In both cases, the training set of 60K images is split into 48K for training and 12K for validation. **20%** of the training images (or 9600 images) have labels for the classification loss.

The training, validation and testing code along with the splitting of labeled/unlabeled data is provided in **A9_main.py**. However, you can change hyperparameters for the training inside the classes **`TrainParams`**, **`MNISTParams`** and **`FMNISTParams`**.
`TrainParams` should contain parameter values common to both datasets while the other two can contain dataset-specific values if needed.
You also need to find the optimal C_0 by either changing the parameter **`c0`** in which case it is a simple multiplicative factor (as in the above equation) or by setting it to 0 and adding custom code for combining the two losses in the class **`CompositeLoss`**.
The latter technique can also be used to make C_0 (or parameters for anything equivalent implemented in **`CompositeLoss`**) learnable and so optimized along with the network parameters during training.

There are **four constraints** on the architecture of the autoencoder:

1. All layers must be **fully connected**
2. Encoder and decoder must be **symmetric**, i.e. they must have the same number of layers and their input-output dimensionalities must be exactly inverted.
3. Encoder and decoder layers must **share weights**. There is no built-in mechanism in pytorch to share weights between modules so this must be done by manually defining weight tensors to implement fully connected layers (instead of using `nn.Linear`).
4. Encoder output size must not exceed **150**

All layers in the classifier must also be fully connected.
Each of the classes has an **`init_weights`** function that also needs to be implemented to initialize weights. In addition, **`Encoder`** has a **`get_weights`** function whose output is passed to the **`init_weights`** of **`Decoder`** to facilitate weight sharing between the two.

Marking:

- **Code (75%)**: A minimum reconstruction quality as measured by PSNR must be achieved to qualify for marking whereupon marks will be scaled linearly with classification accuracy as follows:
  - MNIST: **93%** - **97%** accuracy with PSNR threshold of **25**
  - Fashion MNIST: **83%** - **87%** accuracy with PSNR threshold of **17**

  Also, the testing time should not exceed **25 seconds** on Colab **CPU**.
  Performance on each dataset is worth half the marks for this part.

- **Lab Quiz (25%)** in the week of Nov 18 – 22

What to Submit: You need to submit a single **zip** file containing the modified **A9_modules.py** along with the **checkpoints** folder. The latter should include two subfolders named **mnist** and **fmnist** containing the respective saved weights. Note that the code can save multiple checkpoints in each run but only the one to be loaded for testing should be included. Also make sure to add all import statements in **A9_modules.py** needed to make it work as an imported module from **A9_main.py**. Submission deadline is **November 22, 11:55 PM**.