CNNs and Autoencoders

Question 1: Gradient of a Convolution

[4 marks] The network on the right shows a convolutional layer in a larger network. The input is $\mathbf{x} \in \mathbb{R}^{50}$. Consider the kernel (or "filter") $\mathbf{w}$, with 9 elements and a bias, so that the input current for the next layer is

$$z_j \;=\; b \;+\; \left(\mathbf{w} \circledast \mathbf{x}\right)_j \qquad \text{for} \quad j = 0, \ldots, 49 \tag{1}$$

where b is the kernel's bias, and

$$\left(\mathbf{w} \circledast \mathbf{x}\right)_j \;\equiv\; \sum_{i=0}^{8} w_i \, x_{j-4+i} \; .$$

Note that $x_j = 0$ and $z_j = 0$ for $j < 0$ or $j > 49$.

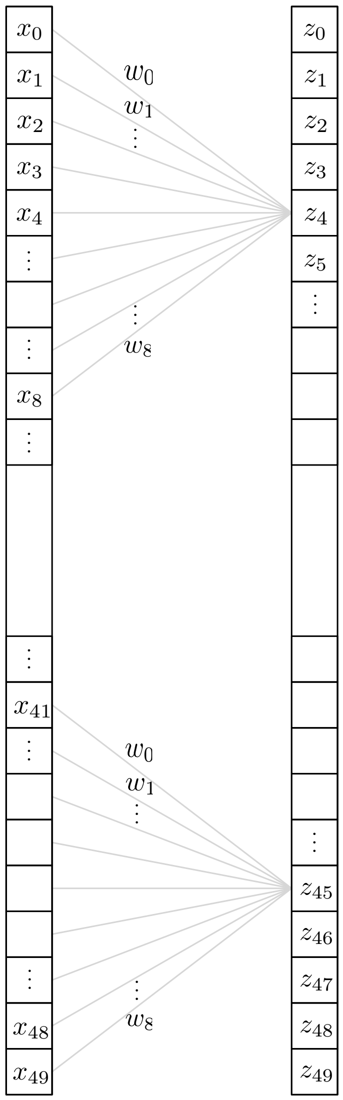
Suppose you are given the gradient of the loss function with respect to these input currents,

$$\nabla_{\mathbf{z}} E \;=\; \begin{bmatrix} \dfrac{\partial E}{\partial z_0} & \cdots & \dfrac{\partial E}{\partial z_{49}} \end{bmatrix}$$

Write a formula that computes $\nabla_{\mathbf{x}} E$,

$$\nabla_{\mathbf{x}} E \;=\; \begin{bmatrix} \dfrac{\partial E}{\partial x_0} & \cdots & \dfrac{\partial E}{\partial x_{49}} \end{bmatrix} \; .$$

Express your answer using vectors and the $\circledast$ operator, as in (1).



Question 2: Autoencoders

Background

Download the jupyter notebook `a05_YOU.ipynb`. Note that this notebook requires PyTorch (`torch`). It also uses `tqdm`, which is a simple text-based progress bar (find out more here). You can remove the dependence on `tqdm` if you like.

This notebook also requires Torchvision and will automatically download the MNIST dataset into a subdirectory called `files`. If you have difficulty getting the dataset, you can also grab it as a zip file here. Unpack it in your working directory, and it should create a subfolder called `files`.

This assignment is more computationally intense than previous assignments. It might be wise to do preliminary testing on a smaller version of the dataset, and with fewer epochs; the notebook has instructions on how to alter the size of the input images. Once you are confident with your code, you can run the full test. And remember that you can use Google's colab,

https://colab.research.google.com

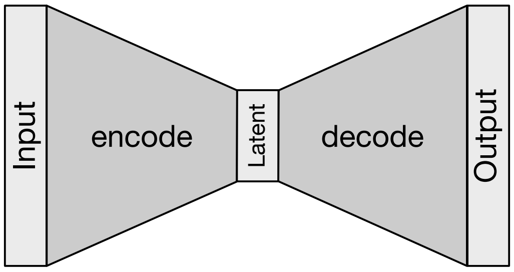
The notebook contains a class called `MyAE`; it will implement an autoencoder, but does not do anything interesting yet.

A. [3 marks] **Complete the** `__init__` **and** `forward` **methods** in the class. You should set up your implementation so that

```
h = net.encode(x)
y = net.decode(h)
```

yields `h`, the 2D latent representation of the input `x`, and then generates `y`, a reconstruction of the input `x`. These methods should work on `torch.tensor` input, including a batch with one sample in each row. There is more than one way to set up the class' encode/decode functionality.

The encoder must have at least two hidden layers (interleaved by nonlinear activation functions), followed by an embedding (latent) layer with only 2 nodes. Likewise, the decoder must have at least 2 hidden layers between the embedding layer and the output layer. You should hardcode the number of nodes in each hidden layer (choose what seems sensible), but the input and output layers should each have `img_size`^{2} nodes, and the embedding layer should have `embedding_dim` nodes (those are both inputs to the class constructor).



B. [2 marks] **Add code that trains your network** on the dataset of 1024 full-sized MNIST digits. You should use one of PyTorch's optimizers, like `SGD` or `Adam`. Feel free to use other optimization bells-and-whistles, like momentum (but it's not required). Use mini-batches of between 8 and 256 samples, whatever size seems like a good compromise between speed and quality. You will probably have to train it for a few hundred epochs.

C. [2 marks] **Plot the latent representations** of all the samples as a scatter plot. The supplied function `plot_latent` does most of the work for you, and produces output similar to (but not exactly like) that shown in Fig. 1.

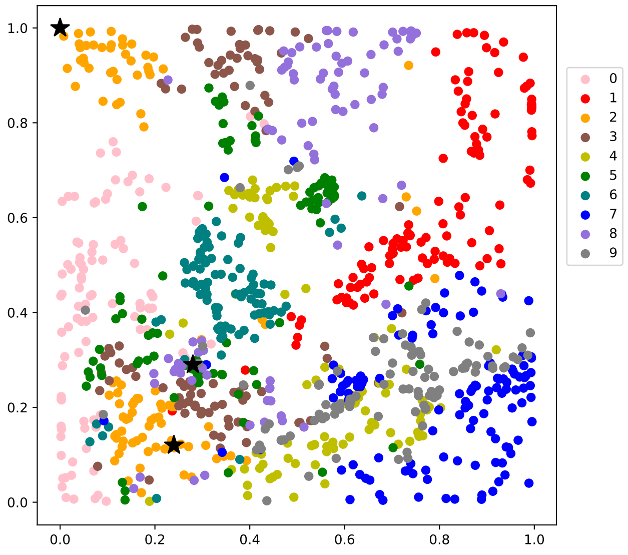
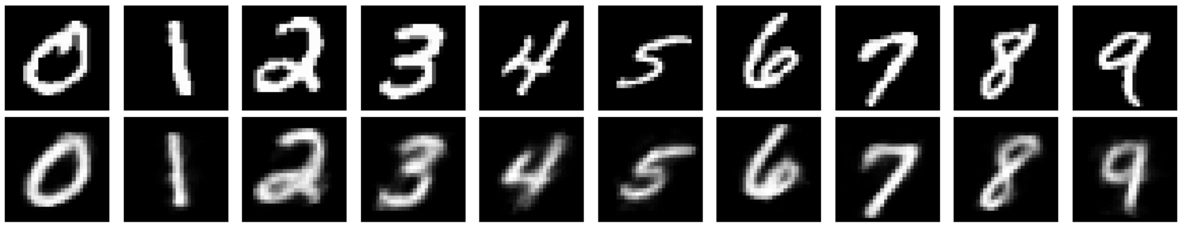


Figure 1: Latent-space scatter plot, including 3 stars for question E.

D. [2 marks] **Plot a reconstructed sample for each of the 10 classes**, like shown below. The top row shows the input, and the bottom row shows the corresponding output from the network.



E. [3 marks] Investigate the latent space by reconstructing the following images.

(a) Choose a class that has points that are widely spread out, but still reasonably separated from the other classes. Select a location in the 2D embedding space that is within this cloud of points. Call this point A. Select another point within the same class, but far away from A. Call this point B. For each of A and B, **decode and display** the corresponding output.

(b) There are places where the point clouds of different classes overlap. Choose a location from where classes overlap, and **reconstruct and display** the decoded output.

(c) **Display a star symbol** in the latent space scatter plot for each of your 3 reconstructions for parts (a) and (b), as shown in Fig. 1. You can either create a new scatter plot, or add the stars to your scatter plot for question C.

What to submit

Question 1: You can:

- typeset your solutions using a word-processing application, such as Microsoft Word, LaTeX, Google docs, etc., or
- write your solutions using a tablet computer, or
- write your solutions on paper and take photographs.

In any case, it is **your responsibility** to make sure that your solutions are sufficiently legible.

Submit your solution to Crowdmark only. Crowdwark will accept PDFs or image files (JPEG or PNG). You may submit multiple files for each question, if needed.

Question 2: You must submit your jupyter notebook in two places: **Crowdmark**, and **D2L**.

Crowdmark: Export your jupyter notebook as a PDF, and submit the PDF to Crowdmark.

D2L: Submit your `ipynb` file to Desire2Learn in the designated dropbox. You do not need to zip the file.

Make sure you submit your solutions, and not just a copy of the supplied skeleton code. We suggest you rename the `ipynb` file by replacing "`YOU`" with your WatIAM ID (eg. `a05_jorchard.ipynb`).

Important note about submitting code
We re-run your notebook in its entirety. So make sure that all your code cells run, from top to bottom. Any code cell that crashes will stop us from re-running your notebook (and you don't want that). Also, we extract the class and function definitions and autotest them. To facilitate this extraction process, please ensure that:

- all `import` commands are in a single code cell (at the top of the notebook), and
- code cells that contain function definitions and/or class definitions should not include any lines of code that are not part of the definition.