Codelet 7: Sparse Autoencoder using PyTorch

The due date for this codelet is **Wednesday, Mar 26 at 11:59PM**.

Introduction

The aim of this codelet is to build on your PyTorch skills by implementing a sparse autoencoder on MNIST data. You will also gain hands-on practice building a custom loss function that mixes multiple factors. You should draw on your prior codelets and the class lecture on autoencoders.

Important: Only use PyTorch, numpy, pandas, matplotlib, PIL, and in-built Python for this codelet. The use of any other libraries, including the books d2l library results in an automatic unsatisfactory grade for this assignment. A core goal of this class is to build your competencies as a machine learning engineer. I want to minimize abstractions from other libraries so that you build these skills.

Outline

- ☐ Grading
- ☐ Sparse Autoencoder
    - ☐ Model
    - ☐ Visualizations
    - ☐ Training

Your assignment

Your task is to:

1. Download `codelet7.zip` from the course website and open it. You will find these instructions, `utils.py`, which includes a basic plotting function, a folder `figures`, which includes sample visualizations prior to training, and `codelet7.py`, which has scaffolding for you.
2. Complete each of the 3 broad tasks below in `codelet7.py` and include a file called `codelet7.pdf` with your answers to the written questions.

Grading

When assessing your work, satisfactory achievement is demonstrated, in part, by:

- Simple, minimal, and clear code
- Code utilizes PyTorch methods in a way that simplifies your code
- Code passes relevant tests
- Explanation extends beyond repeating the textbook/material online
- Response are concrete and clearly demonstrate an understanding of the concepts

Sparse Autoencoder

You will implement a sparse autoencoder using Pytorch. By the end, you will visualize your model's reconstructed images and conduct some initial analysis of what the sparse features represent. You have three tasks, of varying difficulty:

1. Implement a model class
2. Complete visualizations for model representations and output
3. Train a model on MNIST

Help

To orient you to your problem, and to scaffold the later visualization task, a plotting function, `show` has been provided with in `utils.py` and imported into `codelet7.py`. When first running `codelet7.py`, the MNIST will be loaded and put in a PyTorch dataloader (`train_dataloader`). A transformation is applied to the image to normalize the pixel values around 0 and to make them PyTorch tensors. Below, I've provided a small bit of code that will plot 64 images, in 4 rows of 16 images, so that you can see the data. It uses `make_grid` from `torchvision` which takes a list of images and a desired number of images per row, and returns a grid which we can pass to `show`. You should add this code to `codelet7.py` and confirm you can generate plots.

```
batch = next(iter(train_dataloader))
images, _ = batch
g = [image for image in images]
grid = make_grid(g, 16)
show(grid)
```

Model

In this codelet, the aim is to build a simple sparse autoencoder and apply it to MNIST data. Some scaffolding is provided in `codelet7.py`. Your task is to complete the `__init__`, `forward`, `batched_sparsity_penalty`, and `loss_function` methods of the `SparseAutoencoder` class. The sparse autoencoder we are building is a simple encoder-decoder structure. The encoder should be a torch `Linear` layer mapping `inputDims` to `hDims` which a `Sigmoid` activation function is applied to. The decoder should be a torch `Linear` layer mapping `hDims` to `inputDims` and applying a `Tanh` activation function. After creating the linear layers, you should use `init_weights`, which takes a linear layer, to initialize the weights. We are using Xavier Initialization, which helps our model converge faster and more consistently. A brief overview of this initialization strategy, if you are curious, can be found here.

The sparseness of the sparse autoencoder is enforced by the use of regularization, discussed in class. In `batched_sparsity_penalty`, you should calculate the following penalty (which uses KL divergence):

$$\sum_{j=0}^{|h|} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \tag{1}$$

where ρ is 0.05, j ranges over the number of hidden nodes, and $\hat{\rho}_j$ is the average activation of hidden node j for input. That is, we want the average activation of a node to approach 0.05 (i.e., it should selectively "fire" for a subset of the input). Note that we have two edge cases that pose a problem for this penalty, when $\hat{\rho}_j$ is 0 and when it is 1 (because log is not defined for 0). To avoid this, you should use `torch.clamp` to clamp the values between $1e^{-8}$ and $1 - 1e^{-8}$.

For example, given the following hidden representations of 5 inputs (where the hidden representation has 3 nodes), we should get an output of `1.1490`.

$$\begin{pmatrix} 0.3 & 0. & 0.1 \\ 0.2 & 0. & 0.2 \\ 0.5 & 0. & 0.1 \\ 0.5 & 0. & 0.3 \\ 0.5 & 0. & 0.3 \end{pmatrix} \tag{2}$$

Finally, the loss should combine mean squared error and the sparsity penalty, in `loss_function`. In particular you should add to the MSE between the true and predicted pixel values to the sparsity penalty, weighted by $1e^{-4}$.

You evidence completion of this task with successfully training a model.

Visualizations

You should write code to generate two visualizations. One plots 16 rows of input images and their reconstructed output using your model. Another visualizes the relationship between a given hidden node and input. In particular, you should write code that will plot the 100 input images that have the highest hidden activation for a given node. Each row of your plot should have 10 images. Examples of both plots for a model prior to training with 10 hidden units is provided in `figures`.

You evidence completion of this task with successfully training a model.

Training

Finally, you should train a model for 20 epochs using the Adam optimizer with a learning rate of `0.0001`. Your model should have 10 hidden dimensions. For comparison purposes, below is my initial and final loss after 20 epochs.

```
Epoch:  1/20 - Train Loss: 0.8029
Epoch: 20/20 - Train Loss: 0.5329
```

To evidence completion of this task, you should add to the pdf accompanying your code your initial and final loss values and the two visualization figures for your trained model. Additionally, you should look at a few nodes and comment on, at a high level, what you think is being learned as the features in the hidden representation.