Section 10: Autoencoder and CNN

1. Autoencoder

Here is a two dimensional feature space $y = ax^2 + c$ where x and y are two features with non-linear relation and c is some noise. We use an autoencoder to reconstruct this feature space. What does encoder do, and what does decoder do in this autoencoder?

2. Convolutional Neural Networks

(a) Discuss the advantages of a convolutional layer compared to a fully connected one.

(b) Discuss the advantages of maxpooling in CNN.

3. Shapes in Convolutional Neural Networks

When designing a convolutional neural network, it's important to think about the shape of the data flowing through the data. In this problem you will get gain experience with thinking about the shapes in a neural network and a better intuition for why convolutional neural networks require so few parameters compared to fully connected layers.

Shape of a convolutional layer / maxpooling output: For a $n \times n$ input, $f \times f$ filter, padding p and stride s, the output size is $o \times o$ where:

$$o = \frac{n - f + 2p}{s} + 1$$

We will use Pytorch Conv2d to represent a 2D convolution, and Pytorch MaxPool2d to represent a 2D max pooling. For more information about Conv2d and MaxPool2d, click on the following link: Conv2d; MaxPool2d

(a) Assume your input is a batch of N 64x64 RGB images. The input tensor your neural network reveives will have shape $(N, 3, 64, 64)$. For each of the following operations, determine the new shape of the tensor as it flows through the network. Note that activations are omitted since they don't change the shape of the data.

1. `Conv2D(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)`

2. `MaxPool2d(kernel_size=2, stride=2, padding=0)`

3. `Conv2D(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=0)`

4. `MaxPool2d(kernel_size=2, stride=2, padding=1)`

5. `Conv2D(in_channels=32, out_channels=8, kernel_size=1, stride=1, padding=0)`

6. `Conv2D(in_channels=8, out_channels=4, kernel_size=5, stride=1, padding=0)`

7. `Flatten`

8. `Linear(in_features=576, out_features=10)`

(b) Again assume your input is a batch of N 64x64 RGB images. Now compute the number of parameters that each layers has. For the convolutional layers, also compute the number of parameters a fully connected layer mapping from the flattened input channels to the flattened output channels would have. It is okay to leave the number of parameters as products and additions such as $64 \cdot 32 + 16$.

1. `Conv2D(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)`

2. `MaxPool2d(kernel_size=2, stride=2, padding=0)`

3. `Conv2D(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=0)`

4. `MaxPool2d(kernel_size=2, stride=2, padding=1)`

5. `Conv2D(in_channels=32, out_channels=8, kernel_size=1, stride=1, padding=0)`

6. `Conv2D(in_channels=8, out_channels=4, kernel_size=5, stride=1, padding=0)`

7. `Flatten`

8. `Linear(in_features=576, out_features=10)`