Deep Learning

Exercise 9: Convolutional Auto-Encoder

Room: **BIN-1-B.01**

Instructor: Manuel Günther

Email: guenther@ifi.uzh.ch

Office: AND 2.54

Friday, May 6, 2022

Outline

Outline

1. PyTorch
   - Fractionally-Strided Convolution
   - Network Implementation

Fractionally-Strided Convolution

Convolution by Multiplication

$$\underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}}_{\textbf{Output}} = \underbrace{\begin{bmatrix} w_1 & w_2 & w_3 & & & & \\ & w_1 & w_2 & w_3 & & & \\ & & w_1 & w_2 & w_3 & & \\ & & & w_1 & w_2 & w_3 & \\ & & & & w_1 & w_2 & w_3 \end{bmatrix}}_{\textbf{Kernel}} \otimes \underbrace{\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{bmatrix}}_{\textbf{Input}}$$

Implemnentation

- Implemented: `torch.nn.Conv2d`

Transposed Convolution

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix}}_{\textbf{Output}} = \underbrace{\begin{bmatrix} w_1 & & & & \\ w_2 & w_1 & & & \\ w_3 & w_2 & w_1 & & \\ & w_3 & w_2 & w_1 & \\ & & w_3 & w_2 & w_1 \\ & & & w_3 & w_2 \\ & & & & w_3 \end{bmatrix}}_{\textbf{Transposed Kernel}} \otimes \underbrace{\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \end{bmatrix}}_{\textbf{Deep Feature}}$$

Fractionally-Strided Convolution

Implementation

- **Inverts** `Conv2d`
- `torch.nn.ConvTranspose2d`
- Parameters
  - → `stride`: fractional stride
  - → `padding`: same as `Conv2d`
  - → `output_padding`

Output Padding

- For `stride=2`, output dimensions always odd
- Add output padding of 1

Transposed Convolution

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} w_1 & & & & \\ w_2 & w_1 & & & \\ w_3 & w_2 & w_1 & & \\ & w_3 & w_2 & w_1 & \\ & & w_3 & w_2 & w_1 \\ & & & w_3 & w_2 \\ & & & & w_3 \end{bmatrix} \otimes \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \end{bmatrix}$$

Output

Transposed Kernel

Deep Feature

Fractionally-Strided Convolution

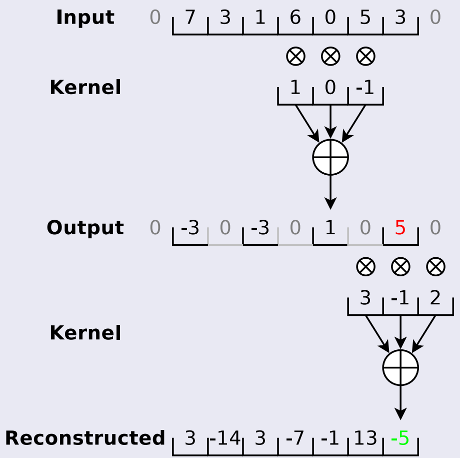
Implementation

- **Inverts** `Conv2d`
- `torch.nn.ConvTranspose2d`
- Parameters
  - → `stride`: fractional stride
  - → `padding`: same as `Conv2d`
  - → `output_padding`

Output Padding

- For `stride=2`, output dimensions always odd
- Add output padding of 1

Fraction. Strided Conv.



Network Implementation

Reminder: Modules in PyTorch

- A module can be
  - $\rightarrow$ A separate layer (e.g. Linear, ReLU, ...)
  - $\rightarrow$ A block of layers (e.g. ResNet Block)
  - $\rightarrow$ A complete network (e.g. LeNet, ResNet)

$\Rightarrow$ A network is a module

Building Cascaded Modules

- Submodules can be networks
- Auto-encoder: two networks
  - $\rightarrow$ Encoder network $\vec{\varphi} = \mathcal{E}(\mathcal{X})$
  - $\rightarrow$ Decoder network $\mathcal{Y} = \mathcal{D}(\vec{\varphi})$
- Creating auto-encoder:
  - $\rightarrow$ Implement encoder module
  - $\rightarrow$ Implement decoder module
  - $\rightarrow$ Instantiate both in `__init__`
  - $\rightarrow$ Cascade both in `forward`
  - $\rightarrow$ Access through variable

Network Implementation

Encoder Network

```
class Encoder (torch.nn.Module):
  def __init__(self, ...):
    self.conv1 = torch.nn.Conv2d(...)
```

Decoder Network

```
class Decoder (torch.nn.Module):
  def __init__(self, ...):
    self.deconv1 = torch.nn.ConvTranspose2d(...)
```

Auto-Encoder Network

```
class AutoEncoder (torch.nn.Module):
  def __init__(self, ...):
    self.encoder = Encoder(...)
    self.decoder = Decoder(...)
```

Network Implementation

Encoder Network

1. 2D convolution: Q_1 channels, 5×5 kernel, stride 2, padding 2
2. activation function
3. 2D convolution: Q_2 channels, 5×5 kernel, stride 2, padding 2
4. flatten layer
5. activation function
6. fully-connected layer: K outputs

Decoder Network

1. fully-connected layer: K inputs
2. activation function
3. reshaping
4. 2D fractionally strided-convolution: Q_2 channels, 5×5 kernel, stride 2, padding 2
5. activation function
6. 2D fractionally strided-convolution: Q_1 channels, 5×5 kernel, stride 2, padding 2

Decoder goes backward through encoder layers

Outline

2. Convolutional Auto-Encoder
   - Dataset
   - Auto-Encoder
   - Training and Evaluation

Dataset

Task 1: Dataset

- Rely on MNIST dataset, ignore labels
  - $\rightarrow$ Except for evaluation Task 7 where we need labels
- Create training and validation/test sets
  - $\rightarrow$ Simple `ToTensor` transform
- `DataLoader`s: $B = 32$ for training, $B = 100$ for validation
- Nothing special.

Auto-Encoder

Task 2: Encoder Network

- Implement convolutional `Encoder` network
- Provide constructor `__init__(self, ...)`
- Provide `forward(self, x)` implementation

Task 3: Decoder Network

- Implement fractionally strided-convolutional `Decoder` network
- Provide constructor `__init__(self, ...)`
- Provide `forward(self, x)` implementation
- Output values should be in range $[0, 1]$

Auto-Encoder

Task 4: Joint Auto-Encoder Network

- Implement `AutoEncoder`
- Combine `Encoder` and `Decoder`
- Implement `__init__(self, ...)` and `forward(self, x)`

Test 1: Output Sizes

- Instantiate auto-encoder network: $Q_1 = 32, Q_2 = 32, K = 10$
- Define input $\mathbf{X}$ for network
- Compute output of network and check its size and value range
- Check `network.encoder` and `network.decoder` separately

Training and Evaluation

Task 5: Training and Validation Loop

- Use $\mathcal{J}^{L_2}$ loss function `torch.nn.MSELoss`
  - $\rightarrow$ Compare output $\mathbf{Y}$ with input $\mathbf{X}$
- Use Adam optimizer with learning rate $\eta = 0.001$ (or lower)
- Train for 10 epochs
  - $\rightarrow$ Compute training set loss during training
  - $\rightarrow$ Compute and report validation set loss after epoch

When losses stagnate after epoch 2, reduce η!

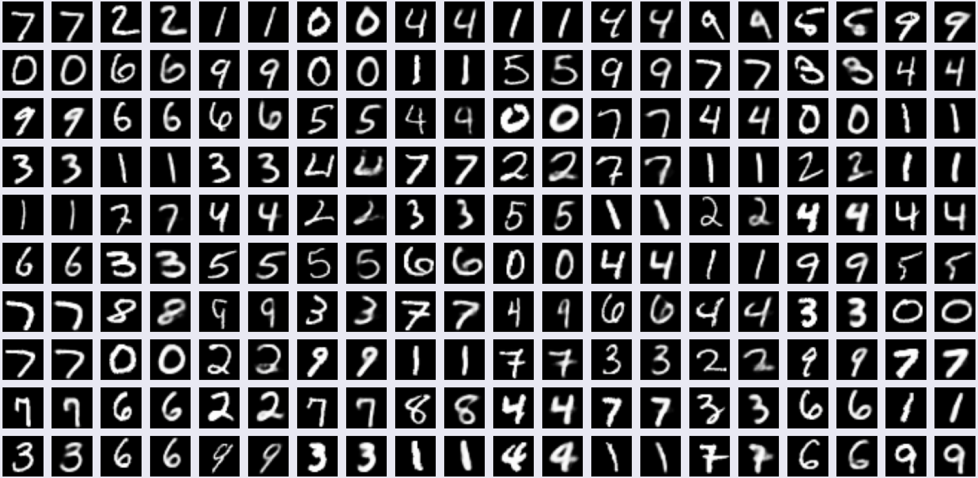
Training and Evaluation

Task 6: Reconstruction Result

- Take batch of images from test set
- Run batch through auto-encoder
- Plot input and output next to each other
  $\rightarrow$ 10 rows with 10 pairs of images

Training and Evaluation

Pairs of Inputs and Outputs of Auto-Encoder



Training and Evaluation

Task 7: Mean Vector per Class

- Use encoder to extract deep feature for validation samples
- Split them by class label
- Compute average feature $\vec{\mu}_o$ for each class o

Task 8: Decode Mixtures of Classes

- Iterate over all possible pairs (o_1, o_2) of classes
- Compute mixed deep feature $\frac{\vec{\mu}_{o_1}+\vec{\mu}_{o_2}}{2}$ of the two classes
  $\rightarrow$ If $o_1 = o_2 = o$, we get the class average $\vec{\mu}_o$
- Use decoder to generate output for deep feature
- Plot in a grid with 10 rows and 10 columns

Training and Evaluation

Reconstructed Mix of Classes