COMP4680/COMP8650: Advanced Topics in SML

Assignment #6: Deep Learning Programming Assignment

Due: 11:55pm on Sunday 28 October, 2018.
Submit as a single ZIP file containing Python code and PDF report (or as asingle Juypter Notebook) via Wattle. Make sure that your name and student ID appears both on your report and in your code.

In this assignment you will build a simple image de-noising auto-encoder neural network model using a dataset of Pokemon characters. We will provide you with starter code using the PyTorch deep learning library, which can be downloaded from `https://pytorch.org/`. Follow the installation instructions (for the stable release), being sure to install both `pytorch` and `torchvision`. Browse through some of the user documentation and tutorials.

Code and data for this assignment can be downloaded from Wattle.

1. `Conv2d` **and** `ConvTranspose2d` **(10 marks).** The model that you will be building uses 2D convolution and transposed convolution layers. These layers take a batch of multi-channel input images and convolve them with learnable filters to produce a batch of multi-channel output image. It is important that you understand how the output size is determined from the the input size and meta-parameters (filter size, stride and padding) of the layer. Let $(N, C_{\text{in}}, H_{\text{in}}, W_{\text{in}})$ and $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ be the sizes of the input and output tensors (i.e., images batches), respectively, where N is the batch size, $C_\bullet$ is the number of channels, $H_\bullet$ is the image height and $W_\bullet$ is the image width. Provide written answers to the following.

   (a) Given a $(32, 3, 64, 64)$-input, what is the size of the output for a `Conv2d` layer with 3×3 filters, 16 output channels, stride of 1, and padding of 1?

   (b) Given a $(32, 128, 1, 1)$-input, what is the size of the output for a `ConvTranspose2d` layer with 3×3 filters, 128 output channels, stride of 1, and padding of 0?

   (c) Given a $(32, 64, 31, 31)$-input, what is the size of the output for a `ConvTranspose2d` layer with 5×5 filters, 32 output channels, stride of 2, and padding of 0?

   (d) How many learnable parameters are there in a `Conv2d` layer with 3×3 filters, that takes a $(32, 3, 64, 64)$-input and produces a 16-channel output?

2. **Implementation.** Your task is to implement encoder and decoder models. We have provided model templates in the file `models.py`. Write your code in the `__init__` methods for the `Encoder` and `Decoder` classes following instructions in the `TODO` comments in the code and specification below. **Do not modify any code outside of the** `models.py` **file.**

   (a) **Encoder (30 marks).** We will use a multi-layer perceptron to implement the encoder. In PyTorch each layer of a multi-layer perceptron is implemented using a `Linear` layer followed by an activation function layer (e.g., `ReLU` or `Tanh`). Often a `BatchNorm1d` layer is inserted between the `Linear` layer and activation function to improve training. Implement a two-layer perceptron of the following architecture:

$$\underbrace{\texttt{Linear} \rightarrow \texttt{BatchNorm1d} \rightarrow \texttt{ReLU}}_{\text{hidden layer}} \rightarrow \underbrace{\texttt{Linear} \rightarrow \texttt{Tanh}}_{\text{output layer}}$$

   The hidden layer and output layer should both have dimensionality `z_dim` $\times 1 \times 1$.

(b) **Decoder (30 marks).** The decoder is implemented as a sequence of transposed and regular convolutions. The transposed convolutions have the effect of incrementally upsampling from the `z_dim` $\times 1 \times 1$ latent representation (input to layer #1) to the $3 \times 64 \times 64$ image (output of layer #5). The regular convolutions smooth the upsampled output. Implement the following architecture:

$$\underbrace{\texttt{ConvTransposed2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU} \rightarrow \texttt{Conv2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU}}_{\text{layer \#1}}$$
$$\rightarrow$$
$$\underbrace{\texttt{ConvTransposed2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU} \rightarrow \texttt{Conv2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU}}_{\text{layer \#2}}$$
$$\rightarrow$$
$$\underbrace{\texttt{ConvTransposed2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU} \rightarrow \texttt{Conv2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU}}_{\text{layer \#3}}$$
$$\rightarrow$$
$$\underbrace{\texttt{ConvTransposed2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU} \rightarrow \texttt{Conv2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU}}_{\text{layer \#4}}$$
$$\rightarrow$$
$$\underbrace{\texttt{ConvTransposed2d} \rightarrow \texttt{BatchNorm2d} \rightarrow \texttt{ReLU} \rightarrow \texttt{Conv2d} \rightarrow \texttt{Tanh}}_{\text{layer \#5}}$$

The specification for each layer is as follows:

	`ConvTransposed2d`			`Conv2d`			
	Filters	Kernel Size	Stride	Filters	Kernel Size	Stride	Padding
Layer #1	128	7	1	128	3	1	Y
Layer #2	64	3	2	64	3	1	Y
Layer #3	64	3	2	64	3	1	Y
Layer #4	32	3	2	32	3	1	Y
Layer #5	32	2	1	3	1	1	N

Use the provided `check_models.py` to check that your models are outputting the correct size tensors. Note that this does not ensure that you have correctly implemented the models, only that their output size is correct. If all the tests pass you will get a message like:

```
Ran 4 tests in 0.627s

OK
```

If one (or more) of the tests fails you will get a message with something like:

```
======================================================================
FAIL: test_layers (__main__.TestEncoder)
Test that the Encoder produces the correct size output for each layer.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_models.py", line 46, in test_layers
    self.assertEqual(tensor.size(), torch.Size([batch_size, latent_size]))
AssertionError: torch.Size([32, 256]) != torch.Size([32, 128])

----------------------------------------------------------------------
```

In this example the message is telling you that one of the layers in your Encoder is producing the wrong size output. In particular, the text is expecting a tensor of size 32×128 but is getting a tensor of size 32×256.

Your code may assume that images are always 64-by-64 pixels.

3. **Experiments (30 marks).** The code will run on a GPU if one is available and the appropriate cuda libraries are installed. Otherwise it will run on the CPU (and take much longer for training).

   (a) Run the `check_models.py` script to ensure that `pytorch` is installed and your model layers are correctly sized.

   (b) Run the asgn6.py script. By default the script will run for 500 epochs through the training data. The loss function, evaluated at each iteration, is saved to `loss.txt` and sample images saved to the directory `samples` every `save_interval` iterations. **Warning:** This may take several hours on a CPU.

   (c) Plot the loss function and comment on whether you think training has converged.

   (d) Look at the `sample-*.png` images. These show the ground-truth images, noisy input images, and denoised images for the mini-batch. Comment on the quality of the denoised images as training progresses.

   (e) Look at the `novel-*.png` images. These show images generated from random noise injected into the decoder model. Comment on the quality of the generated images as training progresses.

   (f) Play around with your encoder and decoder models (they no longer need to pass the tests in `check_models.py`). Report any interesting findings.

A Convolutions and Transposed Convolutions

The assignment makes use of convolutions and transposed convolutions (also, incorrectly, known as deconvolutions).

Convolutions are standard operations in signal processing. As discussed in lectures they provide a mechanism for parameter sharing in neural networks (and have other advantages such as spatial invariance). Given a filter $h \in \mathbb{R}^k$ and input signal $x \in \mathbb{R}^n$ the 1D convolution $y = h * x$ has elements

$$y_i = \sum_{j=1}^{k} h_{k-j+1} x_{i+j-1}, \qquad i = 1, \ldots, n-k+1$$

Consider the example of convolving $h = (-1, 0, 1)$ with $x = (1, 2, 0, -1, 2)$. We have

$$\underbrace{\begin{bmatrix} 1 \\ 3 \\ -2 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} 1 & 0 & -1 & & \\ & 1 & 0 & -1 & \\ & & 1 & 0 & -1 \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} 1 \\ 2 \\ 0 \\ -1 \\ 2 \end{bmatrix}}_{x}$$

Sometimes x is padded with zeros to make the output the same size as the input, e.g, $\tilde{x} = (0, 1, 2, 0, -1, 2, 0)$ would result in output $y = h * \tilde{x} = (-1, 1, 3, -2, 2)$.

Convolutions can also have different strides which shifts the filter by more than one input index for each output index. Alternatively, you can think of stride as subsampling the output (i.e., rows of H).

A transposed convolution sums weighted combinations of the filter coefficients at the output. So the 1D transposed convolution $y = h *^T x$ has elements

$$y_i = \sum_{j=1}^{k} x_{i-j+1} h_j, \qquad i = 1, \ldots, n$$

where we define $x_i \triangleq 0$ for $i \leq 0$.

Consider the example of applying a transposed convolution to $h = (-1, 0, 1)$ and $x = (1, 3, -2)$. We have

$$\underbrace{\begin{bmatrix} 1 \\ 3 \\ -3 \\ -3 \\ 2 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} 1 & & \\ 0 & 1 & \\ -1 & 0 & 1 \\ & -1 & 0 \\ & & -1 \end{bmatrix}}_{H^T} \underbrace{\begin{bmatrix} 1 \\ 3 \\ -2 \end{bmatrix}}_{x}$$

Note that the transposed convolution is *not* the inverse operation of convolution.

As with regular convolutions, transposed convolutions can be padded and have stride other than one.