Advanced Topics in Machine Learning Assignment # 2
Universität Bern

Mehdi Noroozi(noroozi@inf.unibe.ch)

Due date: 24/04/2018

In this assignment you need to upload a zip file to ILIAS which includes: 1) assignment02.py and AutoencoderNet.py files and 2) a pdf of your answers to the questions (see next page). The zip file name must be FirstName_LastName.zip. If your implementation requires auxiliary functions, you must implement that function inside the corresponding .py file. We prefer pdf format (over docx) because it is universally supported on all platforms. Please also indicate your name on top of the first page in your pdf submission. Please do not print or display anything in the code (comment it out before submission).

Maximum grade: 4 Points

The goal is to train a convolutional autoencoder and use its intermediate features to train a linear classifier on the MNIST dataset. You need to implement your network architecture in the `AutoencoderNet` class by completing the definition of the`self.encoder, self.decoder` variables. You are not allowed to change other existing functions of this class, but might need to add your own functions. The encoder should produce a 32 dimensional tensor, e.g. $1 \times 1 \times 32$ or $2 \times 2 \times 8$. Otherwise, the linear classification will generate an error. The output of the decoder should be larger than 28×28. The `forward` function crops the proper central square. You need to also fill up the sections marked as `#TODO` in `assignment02.py`. You can find a smaple log file that a sucessful implementation generates in `sample_log.txt`.

You are allowed to use as many layers as you want with arbitrary specifications (e.g., padding, kernel sizes etc.) according to the limitations imposed in each step in the task descriptions below. Your whole script should not take more than 10 **minutes to run on a `corei7` cpu (this includes training and evaluation).**

- **[1.5 points]** Design and implement your autoencoder using only `torch.nn.Conv2d` and `nn.ConvTranspose2d` modules and pooling layers. You need to set the proper stride and padding to get a 32-dimensional tensor as the output of the encoder. Complete the `TrainAutoencoder` function in `assignment02.py`. Which optimizer works for your setting? Report the best reconstruction PSNR.
- **[1.0 points]** Complete the `TrainLinearclassifier` function to train a linear classifier on top of your encoded features. Report the best `top-1` and `top-5` accuracies that you obtain by training the network within 10 minutes.
- **[1.5 point]** Redesign the autoencoder by adding nonlinearity, e.g. `ReLU`, layers. Report the best reconstruction PSNR and classification `top-1` and `top-5` accuracies. Do you get a higher performance? Justify your observation.

Extra points:

- **[1 point]** Methods performing at more than or equal to 88% top-1 accuracy on MNIST classification will receive 1 extra bonus point.

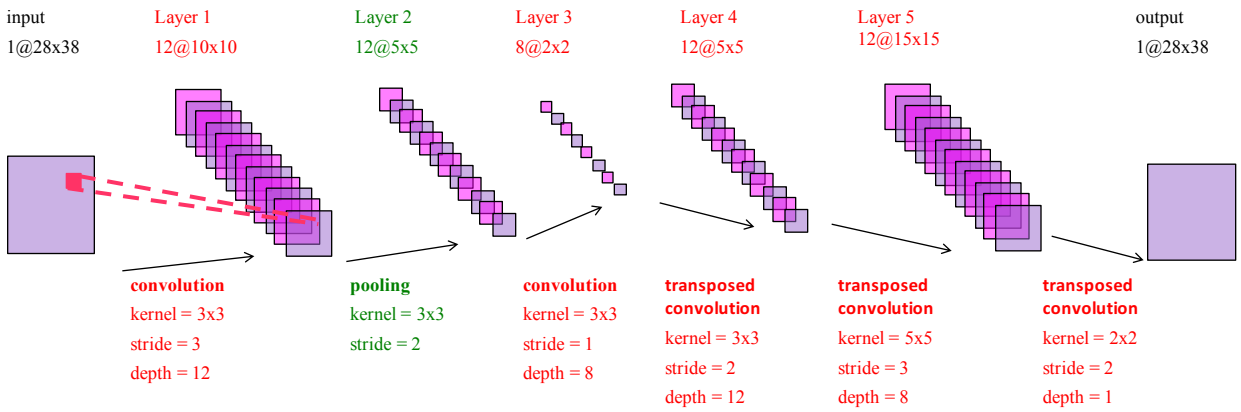

Figure 1: A sample autoencoder architecure

Appendix

This appendix provides a brief overview over CNNs and a sample solution to the autoencoder architecture asked in this assignment. For further references about CNNs see here.

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network). Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations.

Convolutional layer. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

Spatial arrangement. The **depth** of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We must specify the **stride** with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.

Pooling Layer.. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged.

Figure 1 shows a possible architecture of the autoencoder. The text at the bottom shows the layer specifications and the text at the top shows the dimension of each layer output. Note that a transposed convolutional layer carries out a regular convolution but reverts its spatial transformation (having stride > 1 will increase the output dimensions). This architecture does not necessarily lead to a high performance on MNIST classification.