Exercise II

AMTH/CPSC 663b - Spring Semester 2022

Published: Monday, February 13, 2023
Due: Friday, March 10, 2023 - 11:59 PM

The current problem set requires a working installation of PyTorch (v1.4), torchvision, numpy, matplotlib, sklearn, and scipy.

Please include all of your written answers and figures in a single PDF, titled `452_<lastname and initals>_assignment2.pdf`, e.g. for a student named Tom Marvolo Riddle, `452_riddletm_assignment2.pdf`. Upload all relevant files for your solutions to Gradescope, including the PDF report and any Python scripts specified. (For this assignment, the Python scripts include `prob4_CNN.py`, `prob5_AE.py`, and `prob6_CAE.py`.) Your homework should be submitted before Wednesday, March 1, 2023 at 11:59 PM.

We've provided a skeleton code for Problem 4 in a file called `prob4_CNN.py`. Any supplemental code you write (e.g. calling these functions to generate plots, or trying out different parameters) can be handled however you choose. A well-structured Jupyter notebook with neatly produced and labelled figures is an excellent way to compile assignment reports; just be sure to submit a PDF *and* the separate `prob4_CNN.py`, `prob5_AE.py`, and `prob6_CAE.py` file alongside the notebook. However you produce your report, ensure all your figures are clearly labelled.

Programming assignments should use built-in functions in Python and PyTorch. In general, you may use the `scipy` stack [1]; however, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

Important submission instruction

Please name your files exactly as follows:

- `452_<lastname and initials>_assignment2.pdf`
- `prob4_CNN.py`
- `prob5_AE.py`
- `prob6_CAE.py`
- `<lastname and initials>_assignment2.ipynb` (optional)

and submit them on Gradescope (NOT inside a folder; the files should be at the top level on Gradescope after they finish uploading). If there are extra files and scripts you want to submit, you can name and organize them however you see fit.

Problem 1

1. Provide a geometric interpretation of gradient descent in the one-dimensional case. (Adapted from the Nielsen book, chapter 1)
2. An extreme version of gradient descent is to use a mini-batch size of just 1. This procedure is known as online or incremental learning. In online learning, a neural network learns from just one training input at a time (just as human beings do). Name one advantage and one disadvantage of online learning compared to stochastic gradient descent with a mini-batch size of, say, 20. (Adapted from the Nielsen book, chapter 1)

Problem 2

1. Backpropagation with a single modified neuron (Nielsen book, chapter 2)
   Suppose we modify a single neuron in a feedforward network so that the output from the neuron is given by $f(\sum_j w_j x_j + b)$, where f is some function other than the sigmoid. How should we modify the backpropagation algorithm in this case?
2. Backpropagation with softmax and the log-likelihood cost (Nielsen book, chapter 3)
   To apply the backpropagation algorithm for a network containing sigmoid layers to a network with a softmax layer, we need to figure out an expression for the error $\delta_j^L = \partial C / \partial z_j^L$ in the final layer. Show that a suitable expression is: $\delta_j^L = a_j^L - y_j$
3. Backpropagation with linear neurons (Nielsen book, chapter 2)
   Suppose we replace the usual non-linear σ function (*sigmoid*) with $\sigma(z) = z$ throughout the network. Rewrite the backpropagation algorithm for this case.

Problem 3

1. It can be difficult at first to remember the respective roles of the ys and the as for cross-entropy. It's easy to get confused about whether the right form is $-[y \ln a + (1-y) \ln (1-a)]$ or $-[a \ln y + (1-a) \ln (1-y)]$. What happens to the second of these expressions when $y = 0$ or 1? Does this problem afflict the first expression? Why or why not? (Nielsen book, chapter 3)

2. Show that the cross-entropy is still minimized when $\sigma(z) = y$ for all training inputs (i.e. even when $y \in (0, 1)$). When this is the case the cross-entropy has the value: $C = -\frac{1}{n} \sum_x [y \ln y + (1-y) \ln (1-y)]$ (Nielsen book, chapter 3)

3. Given the network in Figure 1, calculate the derivatives of the cost with respect to the weights and the biases and the backpropagation error equations (i.e. δ^l for each layer l) for the first iteration using the cross-entropy cost function. Please use sigmoid activation function on h1, h2, o1, and o2. Initial weights are colored in red, initial biases are colored in orange, the training inputs and desired outputs are in blue. This problem aims to optimize the weights and biases through backpropagation to make the network output the desired results.

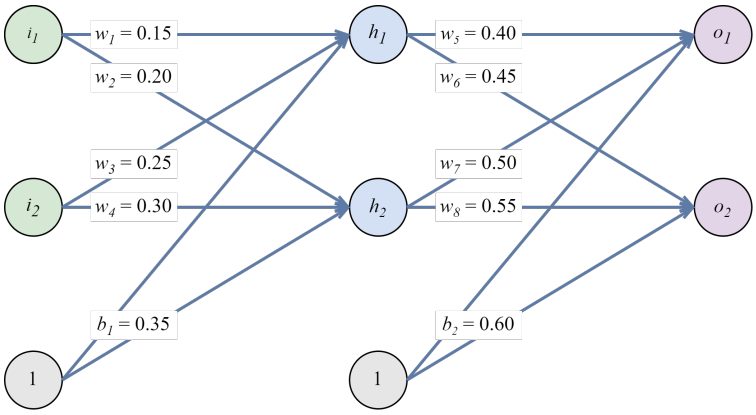

Figure 1: Simple neural network with initial weights and biases.

Problem 4

In this problem, you will review some of the unique parameters involved in the construction of convolutional neural networks (CNNs). Then you will train a CNN on MNIST using the code provided in `p1.py` and visualize a sample of the learned filters of the network.

1. Tracking the dimensionality of representations as they pass through the CNN is slightly different than for fully-connected networks. For the following, give the final output dimensions in terms of height h and width w if the size of the input image is 50x50 (hxw). Here k, s, and p refer to kernel size, stride and padding, respectively. For a) and b), assume the image is passing through a convolutional

layer while for c) and d), assume a max-pooling layer. Assume a square kernel. (This is a helpful visualization.)

(a) k=4, s=1, p=1

(b) k=8, s=5, p=0

(c) k=10, s=2, p=2

(d) k=2, s=1, p=0

2. The following questions can be answered by looking at the provided code in `prob4_CNN.py`

   (a) Train the model for 5 epochs. Visualize filters from the first convolutional layer (40 filters total). Include this image in your report.

   (b) Produce a confusion matrix using the test split and comment on any noticeable class confusion (one class is commonly mislabeled as the other). You may use the sklearn for this. Include the plot in your report.

Problem 5

1. Autoencoders learn to recreate their input after passing it through a sequence of layers in a neural network. This goal can be satisfied by simply learning the identity function. What design choices can be made (e.g. about the network architecture) to prevent them from learning the identity function? Pick at least two and discuss why an autoencoder with your design choice would be more useful than one without it.

2. In class we discussed denoising autoencoders that learn to correct a corruption process that we model with $C(\hat{x} \mid x)$. Describe an example of such a corruption process, express it in a formula, and provide a plot of an original two-dimensional dataset (e.g. samples from $y = x^2$ or anything of your choosing) and the same dataset after you've corrupted it with your function C.

3. Describe the similarity between PCA and a linear autoencoder.

4. Build an autoencoder that embeds the MNIST dataset into two dimensions (tip: start by modifying your CNN MNIST classifier in `prob4_CNN.py`). Save your implementation in `prob5_AE.py` and save your trained model as `mnist_ae.pt`. You can experiment with the details of the architecture, but a good baseline is

   - An encoder that uses four linear layers and ReLU nonlinearities to compress the images into two dimensions, followed by a four-layer decoder that undoes this compression. Do not apply a nonlinearity when entering or exiting the encoder's embedding.
   - Use an Adam optimizer with learning rate = 0.001, and train for at least 10 epochs.
   - Compare the results of your autoencoder with the original images to see if the reconstructions are reasonable.

After you've trained your model and have obtained reasonable reconstruction accuracy, obtain the 2-dimensional embeddings of a batch of images, and plot with colors indicating the label of each image. Describe what you can learn about this dataset from looking at the points embedded into the latent space of the autoencoder.

Problem 6

1. Convert your autoencoder in `prob5_AE.py` into a convolutional autoencoder. Save your implementation in `prob6_CAE.py` and save your trained model as `mnist_cae.pt`. The new autoencoder should include the two convolutional layers and two pooling layers in the encoder, and two upsampling and two convolutional layers in decoder. You may reuse parts of `prob4_CNN.py` and `prob5_AE.py`.

   The model architecture should look something like this:

   For the encoder,

   - Convolution (projecting from 1 to 4 channels, with a kernel size of 3, stride 1, and padding 1)
   - ReLU
   - Max Pooling (with a kernel size of 2 and stride of 2)
   - Convolution (projecting from 4 to 8 channels, with a kernel size of 3, stride 1, and padding 1)
   - ReLU
   - Max Pooling (with a kernel size of 2 and stride of 2)
   - Linear (from the flattened output of the CNN to 20 dimensions)
   - ReLU
   - Linear (from 20 to 2 dimensions)

   And for the decoder, reverse the number of channels and dimensions

   - Linear (from 2 to 20 dimensions)
   - ReLU
   - Linear (from 20 to 392 dimensions)
   - ReLU (and then reshape input to 7×7 square)
   - Upsamling (with the scale factor of 2, and using nearest neighbor algorithm)
   - Convolution (projecting from 8 to 4 channels)
   - ReLU
   - Upsamling (with the scale factor of 2, and using nearest neighbor algorithm)
   - Convolution (projecting from 4 to 1 channel)
   - Sigmoid (to cast values to 0 to 1 range)

Compare the results of your autoencoder with the original images. Obtain the 2-dimensional embeddings of a batch of images, and plot with colors indicating the label of each image. Describe what you can learn about this dataset from looking at the points embedded into the latent space of the autoencoder.

2. Now add skip connections between each pooling layer, and its corresponding upsampling layer. Save your trained model as `mnist_cae_skip.pt`. Again, compare the results of your autoencoder with the original images and plot the 2-dimensional embeddings. Compare the reconstructed images, embeddings, and training losses that you got with and without the skip connections.

3. Repeat parts 6.1 and 6.2, but this time corrupt the images before passing them to the model during training by adding random Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.3)$. When you compare the reconstructed images with the original image, also remember to add random Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.3)$ before passing the image to the model, and compare the reconstructed images with the corrupted images. Do you see any difference in the reconstruction with and without skip connections? How about their 2-dimensional embeddings?

Bonus

1. Where does the softmax name come from? (Nielsen book, chapter 3)

References

[1] "The scipy stack specification." [Online]. Available: https://www.scipy.org/stackspec.html