Deep Learning Homework 3

Spring 2025

Deep Learning / Spring 2025

Homework 3

Please upload your assignments on or before May 4, 2025.

- You are encouraged to discuss ideas with each other, or to consult online tools (such as LLMs). But you **must acknowledge** who you collaborated with or which tools you used, and you **must compose your own** writeup and code independently. Answers should be self-contained and must not appear to be generated by an LLM. We will not grade generic responses.

- We **require** answers to theory questions typeset. Handwritten homework submissions will not be graded.
- We **require** answers to coding questions in the form of a Jupyter notebook. Within each notebook, it is **necessary** to include brief, coherent explanations of both your code and your results to show us your understanding. Use the text block feature of Jupyter notebooks to include explanations.
- Upload both your theory and coding answers in the form of a **single PDF** on **Gradescope**.

---

1. **(5 points)** *Minimax optimization*. In this problem we will see how training GANs is somewhat fundamentally different from regular training. Consider a simple problem where we are trying to minimax a function of two scalars:
$$\min_x \max_y f(x, y) = 4x^2 - 4y^2.$$

   You can try graphing this function in Python if you like (no need to include it in your answer.

   a. Determine the saddle point of this function. A saddle point is a point (x, y) for which f attains a local minimum along one direction and a local maximum in an orthogonal direction.

   b. Write down the gradient descent/ascent equations for solving this problem starting at some arbitrary initialization (x_0, y_0).

   c. Determine the range of allowable step sizes to ensure that gradient descent/ascent converges.

   d. (2 points). What if you just did regular gradient descent over both variables instead? Comment on the dynamics of the updates and whether there are special cases where one might converge to the saddle point anyway.

2. **(5 points)** *Vision-Language models*. As we discussed in class, contrastive vision-language pre-training (CLIP) is one of the major breakthrough ideas in AI over the last few years, and opened the door to a lot of exciting progress. In this problem, we will train and test a few basic CLIP models on the Flickr8K dataset. Your CLIP model should be able to perform zero-shot classification and image retrieval . Good example code is given on the class repository here or in the attached example notebook. You can follow exactly the same recipe as in this notebook, but please explain each part of your code carefully using comments and text blocks.

   a. Train three distinct CLIP models using any combination of vision encoders (e.g. ResNet-18, ResNet-32, or ResNet-50) and text encoder (e.g. DistilBERT, BERT, or RoBERTa).

b. Test each of these models on two different text prompts of your choice. An example test is given at the end of the notebook above. Experiment with various prompt choices to obtain best (qualitative) results.

c. Comment on any trends that you observe. (e.g. Does increasing scale also increase performance?) You don't have to do rigorous tests; qualitative observations will suffice.

3. **(5 points)** *GANS*. In this problem, the goal is to train and visualize the outputs of a simple Deep Convolutional GAN (DCGAN) to generate realistic-looking (but synthetic) images of clothing items.

   a. Use the FashionMNIST training dataset (which we used in previous assignments) to train the DCGAN. Images are grayscale and size 28×28.

   b. Use the following discriminator architecture (kernel size = 5×5 with stride = 2 in both directions):

      - 2D convolutions ($1 \times 28 \times 28 \to 64 \times 14 \times 14 \to 128 \times 7 \times 7$)
      - each convolutional layer is equipped with a Leaky ReLU with slope 0.3, followed by Dropout with parameter 0.3.
      - a dense layer that takes the flattened output of the last convolution and maps it to a scalar.

   Here is a link that discusses how to appropriately choose padding and stride values in order to desired sizes.

   c. Use the following generator architecture (which is essentially the reverse of a standard discriminative architecture). You can use the same kernel size. Construct:

      - a dense layer that takes a unit Gaussian noise vector of length 100 and maps it to a vector of size $7 * 7 * 256$. No bias terms.
      - several transpose 2D convolutions ($256 \times 7 \times 7 \to 128 \times 7 \times 7 \to 64 \times 14 \times 14 \to 1 \times 28 \times 28$). No bias terms.
      - each convolutional layer (except the last one) is equipped with Batch Normalization (batch norm), followed by Leaky ReLU with slope 0.3. The last (output) layer is equipped with tanh activation (no batch norm).

   d. Use the binary cross-entropy loss for training both the generator and the discriminator. Use the Adam optimizer with learning rate 10^{-4}.

   e. Train it for 50 epochs. You can use minibatch sizes of 16, 32, or 64. Training may take several minutes (or even up to an hour), so be patient! Display intermediate images generated after $T = 10$, $T = 30$, and $T = 50$ epochs. If the random seeds are fixed throughout then you should get results of the following quality: