Homework 4 – Deep Neural Networks (CSDS/541, Whitehill, Spring 2024)

You may complete this homework assignment either individually or in teams up to 2 people. In contrast to previous assignments, you may use ChatGPT, Gemini, and other AI tools without restriction. However, you should evaluate their responses critically and not just use them blindly. For training and visualizing results, I recommend using Google Colab (which has a handy integrated AI tool).

1. **Visualizing SGD Trajectories of Fully-Connected Neural Networks (FCNNs) [25 pts]**: First, read all the "Introduction to PyTorch" tutorials (see `https://pytorch.org/tutorials/beginner/basics/intro.html`), including "Training a classifier", "Learn the Basics", "Quickstart", "Tensors", "Datasets & Dataloaders", "Transforms", "Build Model", "Autograd", "Optimization", and "Save & Load Model". Then complete the following tasks:

   (a) **[10 pts]**: Use PyTorch to build and train a simple FCNN with at least 2 hidden layers to classify the Fashion MNIST dataset (similar to Homework 3). You can use label-preserving transformations such as rotation (make sure not to rotate the images by more than, say, $\pm 10°$) to improve generalization. Your network **must** be fully-connected – it cannot use convolution. Report the test accuracy you get in the PDF.

   (b) **[10 pts]**: For any fixed FCNN architecture with at least 2 hidden layers, visualize the gradient descent trajectory in 3-D from **two different random parameter initializations**. In your plot, you should use the first two axes to represent different values for the NN's parameters (which we will denote here collectively simply as $\mathbf{p}$) and the third (vertical) axis to represent the cross-entropy $f_{\mathrm{CE}}(\mathbf{p})$. Of course, there are far (!) more than just 2 parameters in the NN, and thus it will be necessary to perform dimensionality reduction. You should use principal component analysis (PCA) to reduce the parameter space down to just 2 dimensions. The two PCs will represent different "directions" along which the NN parameters can vary, where these directions are chosen to minimize the reconstruction error of the data. In general, it will *not* be the case that a PC corresponds to just a single weight/bias; rather, moving along each PC axis will correspond to changing *all* of the NN's parameters at once.

   Concrete steps:

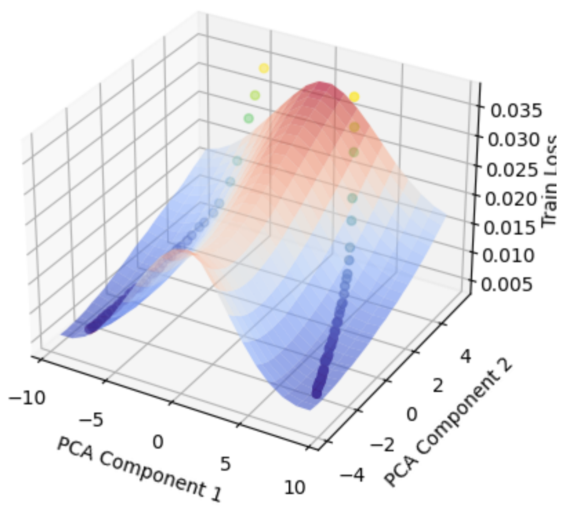
      i. Run SGD at least two times to collect multiple trajectories of $\mathbf{p}$. (Ask ChatGPT for help on how to extract all the parameters from the entire NN as a single vector.) For each value, save the corresponding training cost $f_{\mathrm{CE}}(\mathbf{p})$ – you will need these later. To keep things tractable, train the network on just 1000 examples of Fashion MNIST.

      ii. Use the collected $\mathbf{p}$ vectors to estimate the first 2 principal components that map from the full parameter space down to just 2 dimensions (see `sklearn.decomposition.PCA`).

      iii. For each $\mathbf{p}$ that was encountered during training, project it into the 2-d space, and then plot it as part of a 3-d scatter plot with its associated $f_{\mathrm{CE}}(\mathbf{p})$ value that was computed during SGD.

      iv. To give a sense of the "landscape" through which SGD is "hiking", compute a dense grid of at least 25x25 points in the 2-d PC space. For each point $\tilde{\mathbf{p}}$ in this 2-d space, project it back into the full parameter space (see `PCA.inverse_transform`) to obtain a value for $\mathbf{p}$. Load these parameters $\mathbf{p}$ into the NN (ask ChatGPT for help on how to do this). Finally, make a surface plot (use the `plot_surface` function in `matplotlib`) showing the corresponding cost values over all points in this grid. Render this surface plot in the same figure as the 3-d scatter plot. Make sure to compute the f_{CE} values (for both the 3-d scatter plot and the surface plot) on the set of 1000 *training* data, since that is what SGD directly optimizes. Include your figure in the PDF you submit.

   An example figure is shown below:



As you can see, the two SGD trajectories started on different sides of a ridge and ended up descending into different valleys.

(c) **[3 pts]**: In the figure you created in part (b), the f_{CE} values of the points in the 3-d scatter plot (computed during SGD) do not always exactly equal the corresponding values from the surface plot generated on the dense grid of points. Why is that, and why would it be impractical to create a surface plot over the grid that exactly matches the "real" f_{CE} values obtained using SGD? (Think about how PCA works and how it is used here.) Answer in a few sentences in the PDF.

(d) **[2 pts]**: Assume that the set of all the $\mathbf{p}$ vectors you collected has zero mean (i.e., the sum of all the $\mathbf{p}$ vectors equals the zero vector). Let $\mathbf{p}, \mathbf{p}'$ represent two different configurations of the NNs parameters, and let $\tilde{\mathbf{p}}, \tilde{\mathbf{p}}' \in \mathbb{R}^2$ represent their respective projections in the 2-d PC space. Furthermore, let $\hat{\mathbf{p}}, \hat{\mathbf{p}}'$ represent the reconstructions (using `PCA.inverse_transform`) of the NN's parameters from $\tilde{\mathbf{p}}, \tilde{\mathbf{p}}'$.

Which of the following statements are always true? Report the correct statements in your PDF.

i. If $\mathbf{p} = 2\mathbf{p}'$ (i.e., the NN parameters in the first configuration are all twice the magnitude of the corresponding parameters in the second configuration), then $\tilde{\mathbf{p}} = 2\tilde{\mathbf{p}}'$.

ii. If $\tilde{\mathbf{p}} = 2\tilde{\mathbf{p}}'$, then $\mathbf{p} = 2\mathbf{p}'$.

iii. If $\tilde{\mathbf{p}} = 2\tilde{\mathbf{p}}'$, then $\hat{\mathbf{p}} = 2\hat{\mathbf{p}}'$.

iv. $f_{\text{CE}}(\hat{\mathbf{p}}) \leq f_{\text{CE}}(\mathbf{p})$.

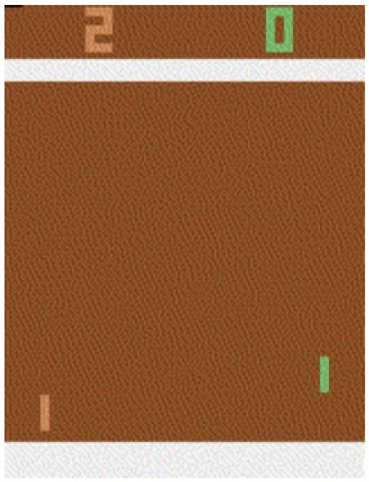
2. **Simple CNN for Fashion MNIST [15 pts]**: Read the following PyTorch tutorial: `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`.

   Then, apply the same methodology to the Fashion MNIST dataset (see the `torchvision.datasets.FashionMNIST` class). Note that, since Fashion MNIST images are grayscale, they have just a single color channel rather than 3. Hence, you will need to adapt the CNN slightly so that the number of input channels in the first convolutional layer is 1 instead of 3. Also, the normalization step `transforms.Normalize` will use just 1 channel instead of 3. Finally, as the image size is different compared to CIFAR10, the size of the feature maps will also be different. You will thus need to update the number of incoming neurons to the first fully-connected (`nn.Linear`) layer accordingly. After making these modifications, train and evaluate a classifier on this dataset. Report the test accuracy in the PDF you submit.

3. **Supervised Pre-Training and Fine-Tuning of CNNs [15 pts]**: Read the following PyTorch tutorial: `https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html`.

Then, apply the same methodology to the Fashion MNIST dataset. Report the test accuracies in the PDF you submit, using either (a) fine-tuning of the whole model or (b) training just the final (classification) layer.

4. **CNNs for Behavioral Cloning in Pong [20 pts]**: Train an AI agent to play Pong (see `https://www.gymlibrary.dev/environments/atari/pong/`). In this game, each player can execute one of 6 possible actions at each timestep (NOOP, FIRE, RIGHT, LEFT, RIGHTFIRE, and LEFTFIRE). The goal is to execute the best action at each timestep based on the current state of the game.



To get started, first download the following files:

- `https://s3.amazonaws.com/jrwprojects/pong_actions.pt`
- `https://s3.amazonaws.com/jrwprojects/pong_observations.pt`

Together, these files define (image, action) pairs generated by an expert player from the Atari Pong video game. Using PyTorch, implement and train any NN architecture you choose (I recommend a simple CNN) to map the images to their corresponding expert actions. Your NN will implement the **control policy** that dictates how the agent behaves in differnet situations, and the approach of training this NN in a supervised manner from expert trajectories is called **behavioral cloning**. After training your NN, save it to a file (use `torch.save`). Then, load the model in `play_atari.py` and see how well your AI "player" does against the computer. To receive full credit, your trained agent should be able to beat the computer (i.e., reach 21 points first). (Note: for this part of the homework, you need to use a standard Python interpreter – Google Colab cannot render the frames of the video game.)

In addition to your Python code (`homework4_WPIUSERNAME1.py` or `homework4_WPIUSERNAME1_WPIUSERNAME2.py` for teams), create a PDF file (`homework4_WPIUSERNAME1.pdf` or `homework4_WPIUSERNAME1_WPIUSERNAME2.pdf` for teams) containing the screenshots described above.