ECE/CS/ME 539 – Fall 2024 — Homework 8

1.

(2 points) In this exercise, we will develop an MLP classifier for the MNIST digit recognition dataset. Please download and familiarize yourself with the starter code provided with this activity (`MLP_torch_starter.ipynb`). The starter code downloads and visualizes the MNIST data. The code also defines, trains and evaluates a simple linear classifier by minimizing the cross-entropy loss using stochastic gradient descent.

(a) Run the code and report the test accuracy and training loss obtained after 10 epochs. Use a learning rate of 0.01 and batch size of 16.

(b) Change the starter code to implement the cross-entropy loss. Train your model again and report the test accuracy and training loss. Discuss your findings.

(c) Change the code from part (b) to implement an MLP with 3 layers, with 350, 50 and 10 neurons at layers 1, 2 and 3, respectively. Train your model again and report the test accuracy and training loss. Discuss your findings.

(d) Change the code from part (c) to implement stochastic gradient descent with momentum. Use a momentum coefficient of 0.9. Train your model again and report the test accuracy and training loss. Discuss your findings.

(e) Train the model from part (d) for 50 epochs. Discuss your findings.

2.

(2 points) In this exercise, we will develop a deep multi-layer perceptron model for the iris dataset. The starter code provided with this exercise (`DNN_torch_starter.ipynb`) defines a 2-layer MLP classifier and trains it by minimizing the cross-entropy loss using stochastic gradient descent for 200 epochs. The MLP is defined with 30 neurons at all hidden layers and a ReLU activation function.

1. Start by running the code and report the final test accuracy. How many epochs was required to achieve convergence?

2. Observe what happens as you increase the number of layers. Train the model with 3 layers, 4 layers, 5 layers, 6 layers. Why did the model stop learning?

3. Implement a new model called `Residual_DNN`. This model should be the same as the DNN model, but instead of computing the output of each layer by `x = self.relu(layer(x))` we will add a residual connection `x = x + self.relu(layer(x))`. Can all layers have a residual connection?

4. Train the new `Residual_DNN` model with increasing number of layers (2, 3, 4, 5, 10, ..., 100). Report and discuss your observations. Why do residual connections help train deep neural network?

3.

(3 points) Download the dataset `winequality-white.csv`. It has 7 class labels (last column): 3, 4, 5, 6, 7, 8, 9 which should be converted into one-hot encoding. Perform feature normalization before applying them to the MLP. Partition the data (Holdout, stratified) into 60/20/20 partitions where 60% are for training, 20% are for validation and the remaining 20% are for testing (and should never be used during training).

Develop a 5-layer MLP with 11 inputs, 8 output and 3 hidden layers with H hidden neurons in each.

Compare the performance (# iterations of training to convergence, final loss value and accuracy (on testing dataset)).

4.

(3 pts) Consider a two-layer neural network with 2 input features and 1 output. The hidden layer processes the 2 input features into 2 hidden neurons with a sigmoidal activation function, σ. The output layer contains 1 neuron with a sigmoidal activation function, σ. Mathematically, given an input matrix $X \in \mathbb{R}^{N\times 2}$ containing N samples, the model output is computed as follows:

$$\begin{aligned} U_1 &= [\mathbf{1} \quad X]W_1 \\ Z_1 &= \sigma(U_1) \\ U_2 &= [\mathbf{1} \quad Z^1]W_2 \\ Z_2 &= \sigma(U_2) \end{aligned}$$

where $W_1 \in \mathbb{R}^{3\times 2}$ and $W_2 \in \mathbb{R}^{3\times 1}$. Our goal is to learn a model that minimizes the square loss function between the predictions $Z_2 \in \mathbb{R}^N$ and the ground-truth labels $Y \in \mathbb{R}^N$. The loss is defined as:

$$\begin{aligned} L_{sq} &= \frac{1}{2N}(Z_2 - Y)^T(Z_2 - Y) \\ &= \frac{1}{2N}\sum_{i=1}^{N}[Z_2(i) - Y(i)]^2 \end{aligned}$$

1. Derive the equations for the deltas: $\delta_2^i = \frac{\partial L_{sq}}{\partial U_2(i)}$ and $\delta_1^i = \frac{\partial L_{sq}}{\partial U_1(i)}$ (for each sample i)

2. Derive the equations for the gradients wrt to the parameters: $\frac{\partial L_{sq}}{\partial W_1}$ and $\frac{\partial L_{sq}}{\partial W_2}$. These gradients should be a function of δ_1^i and δ_2^i computed in part a).

3. Write down the gradient descent update rule for the parameters of your model.