Missouri University of Science & Technology — Department of Computer Science

Spring 2025 — **CS 5480: Deep Learning**

Homework 1: Multi-Layer Perceptrons and Training

Instructor: *Sid Nadendla* — **Due:** *Mar 9, 2025*

Goals and Directions:

- The main goal of this assignment is to implement a multi-layer perceptron from scratch, and train it on a given dataset
- Comprehend the impact of hyperparameters and learn to tune them effectively.
- You are **not** allowed to use neural network libraries like PyTorch, Tensorflow and Keras.
- You are also **not** allowed to add, or remove any functions or classes, or even modify their names.
- You are also **not** allowed to change the signature (list of input attributes) of each function.
- Please note that this code may take several hours to run on one CPU. You will not find any gain by requesting GPUs on Mill.
- Finally, make sure that the gradients are structured according to their respective dimensions. Perform error handling if a dimensionality mismatch is observed.

Problem 1 Primitives *5 points*

1. **LINEAR BASIS:** Implement a linear function in LINEAR() and its gradient in LINEAR_GRAD() (1 point)
   - X is a $K \times 1$ vector, and W is a $M \times K$ marix (Note that M is a hyperparameter).
   - *Linear function:* $Y = W \cdot X$ is a $M \times 1$ vector.
   - *Gradient of Linear function:* Return one of the following two gradients as needed:

$$\nabla_X Y = W (\text{dimension} = M \times K), \qquad \text{and} \qquad \nabla_W Y = \begin{bmatrix} X & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & X \end{bmatrix}_{M \times M \times K}$$

2. **ACTIVATION FUNCTIONS:** Implement four activation functions, namely ReLU in RELU(), its gradient in RELU_GRAD(), logistic function in LOGISTIC() and its gradient in LOGISTIC_GRAD() (2 points)

   Note: Let x_i be one of the entries in X. Then, activation functions are typically defined on each entry in X, i.e. $y_i = \sigma(x_i)$ for all $i = 1, \cdots, M$

   ReLU Activation:

- *ReLU function:* $y_i = \begin{cases} x_i, & \text{if } x_i \geq 0, \\ 0, & \text{otherwise.} \end{cases}$
- *Gradient of ReLU function:* $\nabla_{x_j} y_i = \begin{cases} 1, & \text{if } i = j \text{ and } y_i \geq 0, \\ 0, & \text{otherwise.} \end{cases}$

Note: The above definition includes the subgradient of ReLU at $x = 0$. Also, $\nabla_x y$ is $M \times M$ dimensional, and does not equate to identity matrix for every y.

<u>Logistic Function</u>:

- *Logistic function:* $y_i = \dfrac{1}{1 + \exp(x_i)}$ for all $i = 1, \cdots, M$.
- *Gradient of Logistic Function:* $\nabla_x y = \begin{bmatrix} -y_1(1-y_1) & 0 & \cdots & 0 \\ 0 & -y_2(1-y_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -y_M(1-y_M) \end{bmatrix}_{M \times M}$

3. **LOSS FUNCTIONS:** In this primitive alone, assume that the true label Y, and therefore, the neural network output $\hat{Y}$ are binary variables (i.e. $M = 1$). Return an error message if this condition does not hold true. Implement the binary cross-entropy loss in CROSSENTROPY(), and its gradient in CROSSENTROPY_GRAD(). (2 points)

   - *Binary Cross Entropy:* $z = l(y, \hat{y}) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$
   - *Gradient of Binary Cross Entropy:* $\nabla_{\hat{y}} z = \dfrac{1-y}{1-\hat{y}} - \dfrac{y}{\hat{y}}$ (a scalar quantity)

   Note: Be aware of the $\frac{0}{0}$ case. A typical solution is to add a very small number $\delta = 10^{-6}$ to both numerator and denominator.

Problem 2 NN-2 Model *5 points*

Using primitive functions defined in Problem 1, implement a two-layer NN in NN2 class, with the following definitions:

1. **<u>Forward pass</u>:** Implement in NN2.FORWARD() definition. (1 point)

   - Function: $\hat{y} = \sigma_2\Big(\boldsymbol{w}_2^T \cdot \sigma_1(W_1 \cdot \boldsymbol{x})\Big)$
   - Assume $\sigma_2(\cdot)$ is a logistic function, and $\sigma_1(\cdot)$ a ReLU function.
   - Assume W_1 is a $M \times K$ matrix, and $\boldsymbol{w}_2$ is a $M \times 1$ vector.

   <u>Backward Pass</u>: Implement in NN2.BACKWARD() definition. (3 points)

   - Let $\boldsymbol{z}_1 = W_1 \cdot \boldsymbol{x}$, $\tilde{\boldsymbol{z}}_1 = \sigma_1(\boldsymbol{z}_1)$, and $z_2 = \boldsymbol{w}_2^T \cdot \tilde{\boldsymbol{z}}_1$. Then, $\hat{y} = \sigma_2(z_2)$.

- Gradient Computation (Backpropagation):

  $$* \quad \nabla_{W_1}\ell = \nabla_{\hat{y}}\ell \cdot \nabla_{\boldsymbol{z}_2}\hat{y} \cdot \nabla_{\tilde{\boldsymbol{z}}_1}\boldsymbol{z}_2 \cdot \nabla_{\boldsymbol{z}_1}\tilde{\boldsymbol{z}}_1 \cdot \nabla_{W_1}\boldsymbol{z}_1 = \left[\frac{\partial \ell}{\partial W_1(i,j)}\right] \in \mathbb{R}^{1\times M\times K}$$

  $$* \quad \nabla_{\boldsymbol{w}_2}\ell = \nabla_{\hat{y}}\ell \cdot \nabla_{\boldsymbol{z}_2}\hat{y} \cdot \nabla_{\boldsymbol{w}_2}\boldsymbol{z}_2 = \left[\frac{\partial \ell}{\partial \boldsymbol{w}_2(m)}\right] \in \mathbb{R}^{1\times M}$$

- Hint: You may use NUMPY.DOT() to compute the product of gradients.

Empirical loss gradient: Implement in NN2.EMP_LOSS_GRAD() definition. (1 point)

- Given a training data $(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)$, the empirical risk is given by

  $$L_N = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, \hat{y}_i).$$

- The gradient of empirical risk is given by

  $$\nabla_{\boldsymbol{w}} L_N = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{w}}\ell(y_i, \hat{y}_i).$$

- **Note:** Everytime the optimization algorithm updates $\boldsymbol{w}$, the gradient of loss function needs to be computed since $\hat{y}$ changes accordingly.

Problem 3 Optimization Algorithms *5 points*

- **GD:** Implement in GRADIENT_DESCENT() function.
  - Hyperparameter: Learning rate ϵ, Number of iterations R
  - Initialize $\boldsymbol{W}^0 \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 I)$.
  - In the r^{th} iteration, compute the gradient of empirical loss with respect to $\boldsymbol{W}^{r-1}$ using *emp_loss_grad* function in the model class.
  - Compute the update step: $\mathbb{W}^{(r)} = \mathbb{W}^{(r-1)} - \epsilon \cdot \nabla L_N(\mathbb{W}^{(r-1)})$
  - **Bonus Points:** You will receive one extra point for each one of the following optimization algorithms: SGD, AdaM

Problem 4 Classification on MNIST[1] Data *5 points*

1. **Data Preprocessing on MNIST:** (2 points)

[1]Original Source: http://yann.lecun.com/exdb/mnist/

- MNIST data comprises of 70,000 images of handwritten digits from 0 to 9 (10 label classes), where each image has 28×28 pixels of gray-scale values ranging from 0 (black) to 1 (white). Flatten each image (28×28 matrix) into one 784×1 vector. Append a '-1' to the end of this array to get the 785×1 input, i.e. $K = 785$.
- Convert these 10-ary labels into a binary label, where the outcome is '1' if the original image label is an **even** number, and '0' otherwise.
- Partition the entire dataset into $T = 10,000$ test samples and the remaining as training samples.

2. **Training on MNIST:** Train NN-2 model on the training portion of the pre-processed MNIST dataset by choosing appropriate hyperparameters M, ϵ, R and the initial weights for W_1 and $\boldsymbol{w}_2$. (1 point)

   Note: Your model performance depends on how well you choose your hyperparameters.

3. **Testing on MNIST:** Validate the performance of the trained NN-2 model using the testing portion of the pre-processed MNIST dataset. Report your performance in terms of accuracy, which is defined as

$$Acc = \frac{1}{|\text{Test Samples}|} \sum_{i \in \text{Test Samples}} \mathbb{1}\left(y_i = \hat{y}_i\right),$$

   where $\mathbb{1}(A)$ is a indicator function that returns a value '1', when A is true. (2 points)

   Note: An acceptable model produces at least 75% accuracy.