

Homework 1: Learning

Instructor: Sid Nadendla

Due: Feb 20, 2023 (*Extended to Feb 27, 2023*)

Goals and Directions:

- The main goal of this assignment is to implement perceptrons and neural networks from scratch, and train them on any 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, move, or remove any files, 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.

Problem 1 Neural Network Components

5 points

- **BASIS FEATURES:** Implement a linear function in `hw1/mlcvlab/nn/basis.py` (1 points)

You may test your implementation by running `hw1/test_basis.py`.

Linear Basis:

- X is a $K \times 1$ vector
- W is a $M \times K$ vector – Note that M is a hyperparameter.
- *Linear function:* $Y = W \cdot X$ is a $M \times 1$ vector.
- *Gradient of Linear function:* $\nabla_W Y = X$

- **ACTIVATION FUNCTIONS:** Implement four activation functions, namely step, ReLU, Sigmoid, Softmax and Tanh function in `hw1/mlcvlab/nn/activations.py`. (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 $i = 1, \dots, N$

Also, you may test your implementation by running `hw1/test_activations.py`.

ReLU Activation:

- *ReLU function:* $y = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$

- Gradient of ReLU function: $\text{relu_grad}(y) = \nabla_x y = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$
- Note that the above definition includes the subgradient of ReLU at $x = 0$.

Sigmoid Function:

- Sigmoid function: $y = \frac{1}{1 + e^{-x}}$
- Gradient of Sigmoid Function: $\nabla_x y = y(1 - y)$

Softmax Function:

- Softmax function: $y_i = e^{x_i} \cdot \left(\sum_{k=1}^N e^{x_k} \right)^{-1}$, for all $i = 1, \dots, N$
- Gradient of Softmax Function: $\frac{\partial y_i}{\partial x_j} = \begin{cases} y_i(1 - y_i), & \text{if } i = j, \\ -y_i y_j, & \text{otherwise.} \end{cases}$

Hyperbolic Tangent Function:

- Hyperbolic Tangent function: $y = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Gradient of Hyperbolic Tangent function: $\nabla_x y = 1 - y^2$

- **LOSS FUNCTIONS:** Implement two loss functions, namely mean squared error (MSE) and binary cross entropy in `hw1/mlcvlab/nn/losses.py`. (2 points)

You may test your implementation by running `hw1/test_losses.py`.

 ℓ_2 norm:

- ℓ_2 norm function: $z = l(y, \hat{y}) = \|y - \hat{y}\|_2 = \left[\sum_{i=1}^N (y_i - \hat{y}_i)^2 \right]^{\frac{1}{2}}$
- Gradient of ℓ_2 norm¹: $\nabla_{\hat{y}} z = \frac{\partial z}{\partial \hat{y}_i} = \frac{1}{z}(y - \hat{y})$

Binary Cross Entropy:

- Binary Cross Entropy: $z = l(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$
- Gradient of Binary Cross Entropy: $\nabla_{\hat{y}} z = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}}$

¹Note that gradient formulae can throw an error for edge cases when the denominator is zero, or it is in $\frac{0}{0}$ form. In practice, such problems are addressed via distorting both numerator and denominator by a small amount $\epsilon = 10^{-6}$. A better approach is to perform gradient clipping, or normalization to control exploding gradients.

Problem 2 Models

8 points

Using library functions defined in `hw1/mlcvlab/nn/*`, do the following:

- **2-layer Neural Network:** Implement a two-layer NN in `hw1/mlcvlab/models/nn2.py`

NN2 model: Implement in `nn2` definition. (2 points)

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

Gradient of NN2 model (Backpropagation): Implement in `grad` definition. (4 points)

- Let $\mathbf{z}_1 = W_1 \cdot \mathbf{x}$, $\tilde{\mathbf{z}}_1 = \sigma_1(\mathbf{z}_1)$, and $z_2 = \mathbf{w}_2^T \cdot \tilde{\mathbf{z}}_1$. Then, $\hat{y} = \sigma_2(z_2)$.
- Gradient Computation (Backpropagation): $\nabla_{\mathbf{w}} \ell(y, \hat{y}) = \begin{bmatrix} \nabla_{W_1} \ell(y, \hat{y}) \\ \nabla_{\mathbf{w}_2} \ell(y, \hat{y}) \end{bmatrix}$, where

$$* \quad \nabla_{W_1} \ell = (\nabla_{\mathbf{z}_1} \ell)^T \cdot \nabla_{W_1} \mathbf{z}_1 = \left[\frac{\partial \ell}{\partial W_1(i, j)} \right] \in \mathbb{R}^{M \times K}$$

$$\nabla_{\mathbf{z}_1} \ell = (\nabla_{\tilde{\mathbf{z}}_1} \ell)^T \cdot \nabla_{\mathbf{z}_1} \tilde{\mathbf{z}}_1 = \left[\frac{\partial \ell}{\partial z_m} \right] \in \mathbb{R}^{M \times 1}$$

$$\nabla_{\tilde{\mathbf{z}}_1} \ell = (\nabla_{z_2} \ell)^T \cdot \nabla_{\tilde{\mathbf{z}}_1} z_2 = \left[\frac{\partial \ell}{\partial \tilde{\mathbf{z}}_1(m)} \right] \in \mathbb{R}^{M \times 1}$$

$$\nabla_{z_2} \ell = (\nabla_{\hat{y}} \ell)^T \cdot \nabla_{z_2} \hat{y} = \left[\frac{\partial \ell}{\partial z_2} \right] \in \mathbb{R}^{1 \times 1}$$

$$* \quad \nabla_{\mathbf{w}_2} \ell = (\nabla_{z_2} \ell)^T \cdot \nabla_{\mathbf{w}_2} z_2 = \left[\frac{\partial \ell}{\partial \mathbf{w}_2(m)} \right] \in \mathbb{R}^{M \times 1}$$

$$\nabla_{z_2} \ell = (\nabla_{\hat{y}} \ell)^T \cdot \nabla_{z_2} \hat{y} = \left[\frac{\partial \ell}{\partial z_2} \right] \in \mathbb{R}^{1 \times 1}$$

- $\nabla_{\hat{y}} \ell$ is the gradient of loss function, implemented in `hw1/mlcvlab/nn/losses.py`.

Gradient of Empirical Risk of NN2 model: Implement in `emp_loss_grad` definition.

... (2 points)

- Given a training data $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$, 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

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

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

Problem 3 Optimization Algorithms

6 points

- **SGD:** Implement SGD in hw1/mlcvlab/optim/sgd.py (3 points)
 - Hyperparameter: δ
 - Identify one random parameter in $\mathbb{W} = \{W_1, \dots, W_L\}$, say the j^{th} parameter amongst all scalar parameters in \mathbb{W} .
 - Zero-out all the other parameters in \mathbf{W}^{r-1} , except the j^{th} parameter. Let this new matrix be $[\mathbf{W}^{r-1}]_j$.
 - Compute the gradient of empirical loss with respect to $[\mathbf{W}^{r-1}]_j$ using *emp_loss_grad* function in the model class.
 - Compute the update step for any model: $\mathbb{W}^{(r)} = \mathbb{W}^{(r-1)} - \delta [\nabla L_N(\mathbb{W}^{(r-1)})]_j$
 - **Note:** There is no momentum term here. We are interested in the basic SGD.
- **AdaM:** Implement AdaM in hw1/mlcvlab/optim/adam.py (3 points)
 - Assume the gradient of empirical loss with respect to $\mathbb{W} = \{W_1, \dots, W_L\}$ is computed elsewhere and given.
 - Hyperparameter: $\delta, \alpha, \beta_1, \beta_2$
 - Momentum: $\mathbf{m}^{(r+1)} = \beta_1 \cdot \mathbf{m}^{(r)} + (1 - \beta_1) \cdot \nabla [L_N(\mathbb{W}^{(r)} + \beta_1 \cdot \mathbf{m}^{(r)})]_j$
 - RMSProp: $s^{(r)} = \beta_2 \cdot s^{(r-1)} + (1 - \beta_2) \cdot [\nabla L_N(\mathbb{W}^{(r)})]^2$
 . (the square operation is computed element-wise here.)
 - Normalization: $\tilde{s}^{(r)} = \frac{s^{(r)}}{(1 - \beta_2^r)}$, and $\tilde{\mathbf{m}}^{(r+1)} = \frac{\mathbf{m}^{(r+1)}}{(1 - \beta_1^{r+1})}$
 - Compute the update step for any model: $\mathbb{W}^{(r+1)} = \mathbb{W}^{(r)} - \frac{\alpha}{\sqrt{\tilde{s}^{(r)}} + \epsilon} \tilde{\mathbf{m}}^{(r+1)}$
 . (the square-root operation is computed element-wise here.)

Problem 4 Classification on MNIST² Data

6 points

For this question, write your code in the Jupyter notebook, labeled as hw1/HW1_MNIST_NN2.ipynb

- **Data Preprocessing on MNIST:** (2 points)

²Original 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).
 - Convert these 10-ary labels into a binary label, where the outcome is ‘1’ if the original image label is an **odd** number, and ‘0’ otherwise.
 - Partition the entire dataset into $T = 10,000$ test samples and the remaining as training samples.
- **Training on MNIST:** Train NN-2 model on the training portion of the pre-processed MNIST dataset in hw1/HW1_MNIST_NN2.ipynb. (2 points)
- Note:** Your model performance depends on how well you choose your hyperparameters.
- **Testing on MNIST:** Validate the performance of the trained NN-2 model using the testing portion of the pre-processed MNIST dataset in HW1_MNIST_NN2.ipynb file. Report your performance in terms of accuracy, which is defined as

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

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

..

(2 points)