Missouri University of Science & Technology — Department of Computer Science

Spring 2022 — **CS 6406: Machine Learning for Computer Vision (Sec: 101/102)**

Homework 1: Learning

Instructor: *Sid Nadendla* — **Due:** *March 4, 2022 (extended and final)*

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.

Problem 1 Neural Network Components *5 points*

- **BASIS FEATURES (1 points):** Implement a linear function in hw1/mlcvlab/nn/basis.py
  You may test your implementation by running HW1_v2/test_basis.py.

  Linear Basis:

  for NN1 w will be same as X
  for NN2 W will be hyper parameter based on how many features we want to learn (100-120)

  – 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:* $\nabla_W Y = X$

  Radial Basis:

  K = 784 or 785 when bias added (-1 or 1)

  – X is a $K \times 1$ vector
  – W is a $K \times 1$ vector
  – *Radial Basis function:* $Y = ||X - W||_2^2$, which is a scalar value
  – *Gradient of Linear function:* $\nabla_W Y = 2\sqrt{Y}$

  Test cases to be fixed

- **ACTIVATION FUNCTIONS (2 points):** Implement four activation functions, namely step, ReLU, Sigmoid, Softmax and Tanh function in hw1/mlcvlab/nn/activations.py.

  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 $i = 1, \cdots, N$
  Also, you may test your implementation by running HW1_v2/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:* $y = \dfrac{1}{1 + e^{-x}}$
- *Gradient of Sigmoid Function:* $\nabla_x y = y(1 - y)$

Softmax Function:

SIGNS

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

Hyperbolic Tangent Function:

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

- **LOSS FUNCTIONS (2 points):** Implement two loss functions, namely mean squared error (MSE) and binary cross entropy in hw1/mlcvlab/nn/losses.py.
  You may test your implementation by running HW1_v2/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 norm:* $\nabla_{\hat{y}} z = \dfrac{\partial z}{\partial \hat{y}_i} = \dfrac{1}{z}(y - \hat{y})$

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:* $\nabla_{\hat{y}} z = \dfrac{1 - y}{1 - \hat{y}} - \dfrac{y}{\hat{y}}$

Problem 2 Optimization Algorithms *6 points*

loop here

- **SGD (3 points):** Implement SGD in hw1/mlcvlab/optim/sgd.py

  $r \to$ rotation (iteration)

  - Hyperparameter: δ
  - Identify one random parameter in $\mathbb{W} = \{W_1, \cdots, W_L\}$, say the j^{th} parameter amongst all scalar parameters in $\mathbb{W}$.

    picked random W
  - Zero-out all the other parameters in $\boldsymbol{W}^{r-1}$, expect the j^{th} parameter. Let this new matrix be $\left[\boldsymbol{W}^{r-1}\right]_j$. $\to$ 785 x 1
  - Compute the gradient of empirical loss with respect to $\left[\boldsymbol{W}^{r-1}\right]_j$ using *emp_loss_grad* function in the model class.
  - Compute the update step for any model: $\mathbb{W}^{(r)} = \mathbb{W}^{(r-1)} - \delta\left[\nabla L_N(\mathbb{W}^{(r-1)})\right]_j$ same shape as $\mathbb{W}^r$ & $\mathbb{W}^{r-1}$
  - **Note:** There is no momentum term here. We are interested in the basic SGD.

    list of weights for all layers

- **AdaM (3 points):** Implement AdaM in hw1/mlcvlab/optim/adam.py
  - Assume the gradient of empirical loss with respect to $\mathbb{W} = \{W_1, \cdots, W_L\}$ is computed elsewhere and given.
  - Hyperparameter: $\delta, \alpha, \beta_1, \beta_2$
  - Momentum: $\boldsymbol{m}^{(r+1)} = \beta_1 \cdot \boldsymbol{m}^{(r)} + (1-\beta_1) \cdot \nabla\left[L_N(\mathbb{W}^{(r)} + \beta_1 \cdot \boldsymbol{m}^{(r)})\right]_j$
  - RMSProp: $s^{(r)} = \beta_2 \cdot s^{(r-1)} + (1-\beta_2) \cdot \left[\nabla L_N(\mathbb{W}^{(r)})\right]^T \cdot \nabla L_N(\mathbb{W}^{(r)})$
  - Compute the update step for any model: $\mathbb{W}^{(r+1)} = \mathbb{W}^{(r)} - \dfrac{\alpha}{\sqrt{s^{(r)}} + \epsilon}\boldsymbol{m}^{(r+1)}$

Problem 3 Models *8 points*

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

- **1-layer Neural Network (4 points):** Implement a one-layer NN in hw1/mlcvlab/models/nn1.py

  NN1 model: Implement in *nn1* definition.
  - Function: $\hat{y} = \sigma(\boldsymbol{w}^T\boldsymbol{x})$
  - Assume $\sigma(\cdot)$ is a sigmoid function
  - Assume $\boldsymbol{w}$ and $\boldsymbol{x}$ have same shape, i.e. both are $K \times 1$ vectors.

  Gradient of NN1 model (Backpropagation): Implement in *grad* definition.
  - Let $z = \boldsymbol{w}^T\boldsymbol{x}$. Then, $\hat{y} = \sigma(z)$

X input shape 785 x 1 after we add -1 or 1 on top or bottom of the x vector to replace bias term in wx+b. The same is done for training data train_x.

- Gradient Computation (Backpropagation): $\nabla_{\boldsymbol{w}} \ell(y,\ \hat{y}) = \nabla_{\boldsymbol{w}} \ell(y,\ \sigma(\boldsymbol{w}^T \cdot \boldsymbol{x}))$

if working with all images at once, this will be a peacewise multiplication * & we will have 60 000 x 1 vector.

$$\nabla_{\boldsymbol{w}} \ell = \overbrace{(\nabla_z \ell)}^{\mathbb{R}}{}^T \cdot \overbrace{\nabla_{\boldsymbol{w}} z}^{x} \rightarrow 785 \times 1 = \left[\frac{\partial \ell}{\partial w_k}\right] \in \mathbb{R}^{K \times 1} \rightarrow 785 \times 1$$

when calcul everything at once this will end up being 785 x 60000 matrix.

$$\nabla_z \ell = \underbrace{(\nabla_{\hat{y}} \ell)^T}_{\mathbb{R}} \cdot \underbrace{\nabla_z \hat{y}}_{\text{sigmoid\_grad}} \rightarrow \mathbb{R} \quad = \left[\frac{\partial \ell}{\partial z}\right] \in \mathbb{R}$$

- $\nabla_{\hat{y}} \ell$ 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.

- 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.

- **2-layer Neural Network (4 points):** Implement a two-layer NN in hw1/mlcvlab/models/nn2.py

NN2 model: Implement in *nn2* definition.

- 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 sigmoid 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.

Gradient of NN2 model (Backpropagation): Implement in *grad* definition.

- 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): $\nabla_{\mathbb{W}} \ell(y,\ \hat{y}) = \begin{bmatrix} \nabla_{W_1} \ell(y,\ \hat{y}) \\ \nabla_{\boldsymbol{w}_2} \ell(y,\ \hat{y}) \end{bmatrix}$, where

1×M · M×M×K ⇒ 1×M×K

WRONG ← M×1   M×M×K

$$
\begin{array}{llll}
* & \nabla_{W_1}\ell = (\nabla_{\boldsymbol{z}_1}\ell) \cdot \nabla_{W_1}\boldsymbol{z}_1 & = \left[\dfrac{\partial \ell}{\partial W_1(i,j)}\right] & \in \mathbb{R}^{M\times K} \\
 & \nabla_{\boldsymbol{z}_1}\ell = (\nabla_{\tilde{\boldsymbol{z}}_1}\ell)^T \cdot \nabla_{\boldsymbol{z}_1}\tilde{\boldsymbol{z}}_1 \Rightarrow M\times M & = \left[\dfrac{\partial \ell}{\partial z_m}\right] & \in \mathbb{R}^{\cancel{M\times 1}} \; 1\times M \\
 & \nabla_{\tilde{\boldsymbol{z}}_1}\ell = (\nabla_{z_2}\ell)^T \cdot \nabla_{\tilde{\boldsymbol{z}}_1} z_2 & = \left[\dfrac{\partial \ell}{\partial \tilde{\boldsymbol{z}}_1(m)}\right] & \in \mathbb{R}^{M\times 1} \\
 & \nabla_{z_2}\ell = (\nabla_{\hat{y}}\ell)^T \cdot \nabla_{z_2}\hat{y} & = \left[\dfrac{\partial \ell}{\partial z_2}\right] & \in \mathbb{R}^{1\times 1} \\
 & & & \\
* & \nabla_{\boldsymbol{w}_2}\ell = (\nabla_{z_2}\ell)^T \cdot \nabla_{\boldsymbol{w}_2} z_2 & = \left[\dfrac{\partial \ell}{\partial \boldsymbol{w}_2(m)}\right] & \in \mathbb{R}^{M\times 1} \\
 & \nabla_{z_2}\ell = (\nabla_{\hat{y}}\ell)^T \cdot \nabla_{z_2}\hat{y} & = \left[\dfrac{\partial \ell}{\partial z_2}\right] & \in \mathbb{R}^{1\times 1}
\end{array}
$$

R

- $\nabla_{\hat{y}}\ell$ 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.

- 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 4 Classification on MNIST Data *6 points*

For this question, write your code in the Jupyter notebooks, labeled as hw1/HW1_MNIST_NN1.ipynb and hw1/HW1_MNIST_NN2.ipynb

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

- Convert these 10-ary labels into a binary label, where the outcome is '1' if the original image label is an **<u>even</u>** number, and '0' otherwise.
- Partition the entire dataset into $T = 10,000$ test samples and the remaining as training samples.

- **Training on MNIST (2 points):**

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

  - Train NN-1 model on the training portion of the pre-processed MNIST dataset in hw1/HW1_MNIST_NN1.ipynb.
  - Train NN-2 model on the training portion of the pre-processed MNIST dataset in hw1/HW1_MNIST_NN2.ipynb.

- **Testing on MNIST (2 points):**

  - Validate the performance of the trained NN-1 model using the testing portion of the pre-processed MNIST dataset in hw1/HW1_MNIST_NN1.ipynb. Report your performance in terms of accuracy:

$$Acc = \frac{1}{T} \sum_{i \in \text{Test Samples}} \mathbb{1}\left(|y_i - \hat{y}_i| > 0\right),$$

    where $\mathbb{1}(A)$ is a indicator function that returns a value '1', when A is true.
  - Validate the performance (in terms of accuracy) of the trained NN-2 model using the testing portion of the pre-processed MNIST dataset in hw1/HW1_MNIST_NN2.ipynb.