

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

### 1 Submission

- Assignment due: Apr 8 (11:55pm)
- Individual assignment
- Up to 2 page summary write-up with resulting visualization (more than 2 page assignment will be automatically returned.).
- Submission through Canvas.
- Following skeletal functions are already included in the `cnn.py` file ([https://www-users.cs.umn.edu/~hspark/csci5561\\_S2021/HW4.zip](https://www-users.cs.umn.edu/~hspark/csci5561_S2021/HW4.zip))
  - `main_slp_linear`
  - `main_slp`
  - `main_mlp`
  - `main_cnn`
- List of function to submit:
  - `get_mini_batch`
  - `fc`
  - `fc_backward`
  - `loss_euclidean`
  - `train_slp_linear`
  - `loss_cross_entropy_softmax`
  - `train_slp`
  - `relu`
  - `relu_backward`
  - `train_mlp`
  - `conv`
  - `conv_backward`
  - `pool2x2`
  - `pool2x2_backward`
  - `flattening`
  - `flattening_backward`
  - `trainCNN`

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

- A list of MAT files to submit that contain the following trained weights:
  - `slp_linear.mat`: `w`, `b`
  - `slp.mat`: `w`, `b`
  - `mlp.mat`: `w1`, `b1`, `w2`, `b2`
  - `cnn.mat`: `w_conv`, `b_conv`, `w_fc`, `b_fc`
- DO NOT SUBMIT THE PROVIDED IMAGE DATA
- The function that does not comply with its specification will not be graded.
- You are not allowed to use computer vision related package functions unless explicitly mentioned here. Please consult with TA if you are not sure about the list of allowed functions.

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

### 2 Overview



Figure 1: You will implement (1) a multi-layer perceptron (neural network) and (2) convolutional neural network to recognize hand-written digit using the MNIST dataset.

The goal of this assignment is to implement neural network to recognize hand-written digits in the MNIST data.

**MNIST Data** You will use the MNIST hand written digit dataset to perform the first task (neural network). We reduce the image size ( $28 \times 28 \rightarrow 14 \times 14$ ) and subsample the data. You can download the training and testing data from here:

[http://www.cs.umn.edu/~hspark/csci5561\\_F2020/ReducedMNIST.zip](http://www.cs.umn.edu/~hspark/csci5561_F2020/ReducedMNIST.zip)

*Description:* The zip file includes two MAT files (`mnist_train.mat` and `mnist_test.mat`). Each file includes `im_*` and `label_*` variables:

- `im_*` is a matrix ( $196 \times n$ ) storing vectorized image data ( $196 = 14 \times 14$ )
- `label_*` is  $1 \times n$  vector storing the label for each image data.

$n$  is the number of images. You can visualize the  $i^{\text{th}}$  image, e.g.,

```
plt.imshow(mnist_train['im_train'][:, 0].reshape((14, 14), order='F'), cmap='gray').
```

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

### 3 Single-layer Linear Perceptron

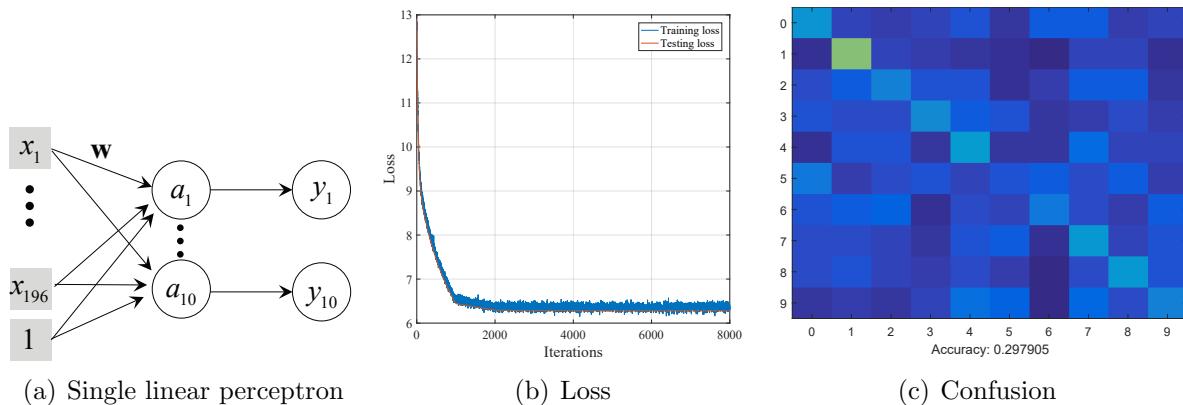


Figure 2: You will implement a single linear perceptron that produces **accuracy near 30%**. **Random chance is 10% on testing data.**

You will implement a single-layer *linear* perceptron (Figure 2(a)) with **stochastic gradient descent** method. We provide `main_slp_linear` where you will implement `get_mini_batch` and `train_slp_linear`.

```
def get_mini_batch(im_train, label_train, batch_size)
    ...
    return mini_batch_x, mini_batch_y
```

**Input:** `im_train` and `label_train` are a set of images and labels, and `batch_size` is the size of the mini-batch for stochastic gradient descent.

**Output:** `mini_batch_x` and `mini_batch_y` are cells that **contain a set of batches** (images and labels, respectively). **Each batch** of images is a matrix with size **`196×batch_size`**, and each batch of labels is a matrix with size **`10×batch_size`** (one-hot encoding). Note that the number of images in the last batch may be smaller than `batch_size`.

**Description:** You should **randomly** permute the the order of images when building the batch, and whole sets of `mini_batch_*` must span all training data.

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

```
def fc(x, w, b)
...
    return y
```

**Input:**  $x \in \mathbb{R}^{m \times 1}$  is the input to the fully connected layer, and  $w \in \mathbb{R}^{n \times m}$  and  $b \in \mathbb{R}^{n \times 1}$  are the weights and bias.

**Output:**  $y \in \mathbb{R}^{n \times 1}$  is the output of the linear transform (fully connected layer).

**Description:** FC is a linear transform of  $x$ , i.e.,  $y = wx + b$ .

```
def fc_backward(dl_dy, x, w, b, y)
...
    return dl_dx, dl_dw, dl_db
```

**Input:**  $dl_dy \in \mathbb{R}^{1 \times n}$  is the loss derivative with respect to the output  $y$ .

**Output:**  $dl_dx \in \mathbb{R}^{1 \times m}$  is the loss derivative with respect to the input  $x$ ,  $dl_dw \in \mathbb{R}^{1 \times (n \times m)}$  is the loss derivative with respect to the weights, and  $dl_db \in \mathbb{R}^{1 \times n}$  is the loss derivative with respect to the bias.

**Description:** The partial derivatives w.r.t. input, weights, and bias will be computed.  $dl_dx$  will be back-propagated, and  $dl_dw$  and  $dl_db$  will be used to update the weights and bias.

```
def loss_euclidean(y_tilde, y)
...
    return l, dl_dy
```

**Input:**  $y_tilde \in \mathbb{R}^m$  is the prediction, and  $y \in \{0, 1\}^m$  is the ground truth label.

**Output:**  $l \in \mathbb{R}$  is the loss, and  $dl_dy$  is the loss derivative with respect to the prediction.

**Description:** `loss_euclidean` measure Euclidean distance  $L = \|y - \tilde{y}\|^2$ .

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

```
def train_slp_linear(mini_batch_x, mini_batch_y)
...
    return w, b
```

**Input:** `mini_batch_x` and `mini_batch_y` are cells where each cell is a batch of images and labels.

**Output:**  $w \in \mathbb{R}^{10 \times 196}$  and  $b \in \mathbb{R}^{10 \times 1}$  are the trained weights and bias of a single-layer perceptron.

**Description:** You will use `fc`, `fc_backward`, and `loss_euclidean` to train a single-layer perceptron using a stochastic gradient descent method where a pseudo-code can be found below. Through training, you are expected to see reduction of loss as shown in Figure 2(b). As a result of training, the network should produce more than 25% of accuracy on the testing data (Figure 2(c)).

---

### Algorithm 1 Stochastic Gradient Descent based Training

---

- 1: Set the learning rate  $\gamma$
- 2: Set the decay rate  $\lambda \in (0, 1]$
- 3: Initialize the weights with a Gaussian noise  $w \in \mathcal{N}(0, 1)$
- 4:  $k = 1$
- 5: **for** `iIter = 1 : nIters` **do**
- 6:     At every 1000<sup>th</sup> iteration,  $\gamma \leftarrow \lambda\gamma$
- 7:      $\frac{\partial L}{\partial w} \leftarrow 0$  and  $\frac{\partial L}{\partial b} \leftarrow 0$
- 8:     **for** Each image  $x_i$  in  $k^{\text{th}}$  mini-batch **do**
- 9:         Label prediction of  $x_i$
- 10:         Loss computation  $l$
- 11:         Gradient back-propagation of  $x_i$ ,  $\frac{\partial l}{\partial w}$  using back-propagation.
- 12:          $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial w} + \frac{\partial l}{\partial w}$  and  $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial b} + \frac{\partial l}{\partial b}$
- 13:     **end for**
- 14:      $k++$  (Set  $k = 1$  if  $k$  is greater than the number of mini-batches.)
- 15:     Update the weights,  $w \leftarrow w - \frac{\gamma}{R} \frac{\partial L}{\partial w}$ , and bias  $b \leftarrow b - \frac{\gamma}{R} \frac{\partial L}{\partial b}$
- 16: **end for**

---

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

### 4 Single-layer Perceptron

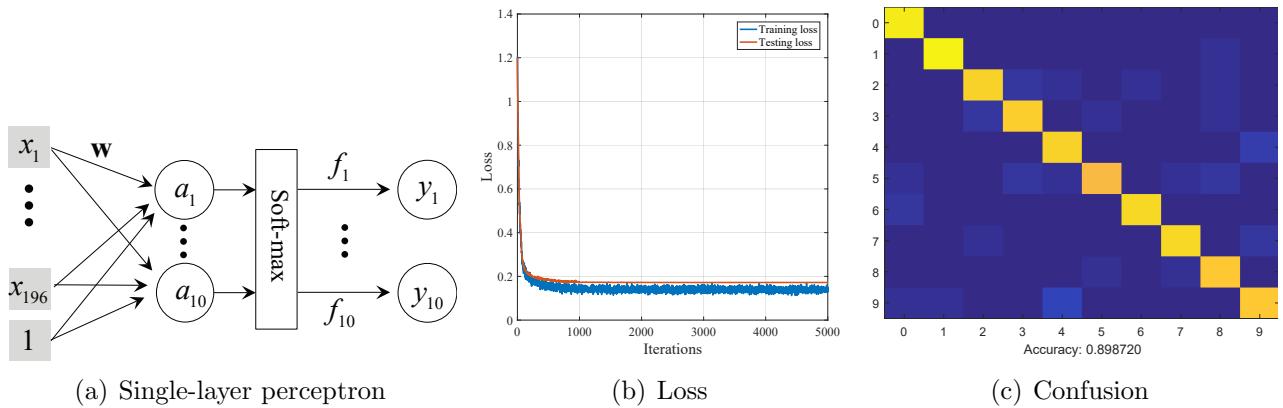


Figure 3: You will implement a single perceptron that produces **accuracy near 90%** on testing data.

You will implement a single-layer perceptron with ***soft-max cross-entropy*** using **stochastic gradient descent method**. We provide `main_slp` where you will implement `train_slp`. Unlike the single-layer linear perceptron, it has a soft-max layer that **approximates a max function by clamping the output to [0, 1] range** as shown in Figure 3(a).

```
def loss_cross_entropy_softmax(x, y)
    ...
    return l, dl_dy
```

**Input:**  $x \in \mathbb{R}^{m \times 1}$  is the input to the soft-max, and  $y \in \{0, 1\}^m$  is the ground truth label.

**Output:**  $L \in \mathbb{R}$  is the loss, and  $dl_dy$  is the loss derivative with respect to  $x$ .

**Description:** `Loss_cross_entropy_softmax` measure cross-entropy between two distributions  $L = \sum_i^m y_i \log \tilde{y}_i$  where  $\tilde{y}_i$  is the soft-max output that approximates the max operation by clamping  $x$  to  $[0, 1]$  range:

$$\tilde{y}_i = \frac{e^{x_i}}{\sum_i e^{x_i}},$$

where  $x_i$  is the  $i^{\text{th}}$  element of  $x$ .

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

```
def train_slp(mini_batch_x, mini_batch_y)
    ...
    return w, b
```

**Output:**  $w \in \mathbb{R}^{10 \times 196}$  and  $b \in \mathbb{R}^{10 \times 1}$  are the trained weights and bias of a single-layer perceptron.

**Description:** You will use the following functions to train a single-layer perceptron using a stochastic gradient descent method: `fc`, `fc_backward`, `loss_cross_entropy_softmax`

Through training, you are expected to see reduction of loss as shown in Figure 3(b). As a result of training, the network should produce more than 85% of accuracy on the testing data (Figure 3(c)).

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

### 5 Multi-layer Perceptron

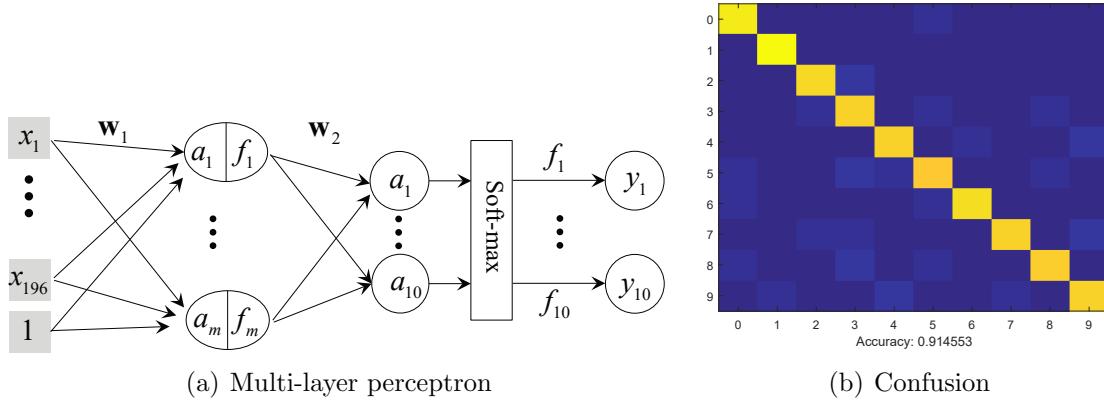


Figure 4: You will implement a multi-layer perceptron that produces **accuracy more than 90%** on testing data.

You will implement a multi-layer perceptron with **a single hidden layer** using a **stochastic gradient descent method**. We provide `main_mlp`. The **hidden layer is composed of 30 units** as shown in Figure 4(a).

```
def relu(x)
    ...
    return y
```

**Input:**  $x$  is a general tensor, matrix, and vector.

**Output:**  $y$  is the output of the Rectified Linear Unit (ReLU) with the same input size.

**Description:** ReLU is an activation unit ( $y_i = \max(0, x_i)$ ). In some case, it is possible to use a Leaky ReLU ( $y_i = \max(\epsilon x_i, x_i)$  where  $\epsilon = 0.01$ ).

```
def relu_backward(dl_dy, x, y)
    ...
    return dl_dx
```

**Input:**  $dl_dy \in \mathbb{R}^{1 \times z}$  is the loss derivative with respect to the output  $y \in \mathbb{R}^z$  where  $z$  is the size of input (it can be tensor, matrix, and vector).

**Output:**  $dl_dx \in \mathbb{R}^{1 \times z}$  is the loss derivative with respect to the input  $x$ .

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

```
def train_mlp(mini_batch_x, mini_batch_y)
    ...
    return w1, b1, w2, b2
```

**Output:**  $w_1 \in \mathbb{R}^{30 \times 196}$ ,  $b_1 \in \mathbb{R}^{30 \times 1}$ ,  $w_2 \in \mathbb{R}^{10 \times 30}$ ,  $b_2 \in \mathbb{R}^{10 \times 1}$  are the trained weights and biases of a multi-layer perceptron.

**Description:** You will use the following functions to train a multi-layer perceptron using a stochastic gradient descent method: `fc`, `fc_backward`, `relu`, `relu_backward`, `loss_cross_entropy_softmax`. As a result of training, the network should produce more than 90% of accuracy on the testing data (Figure 4(b)).

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

### 6 Convolutional Neural Network

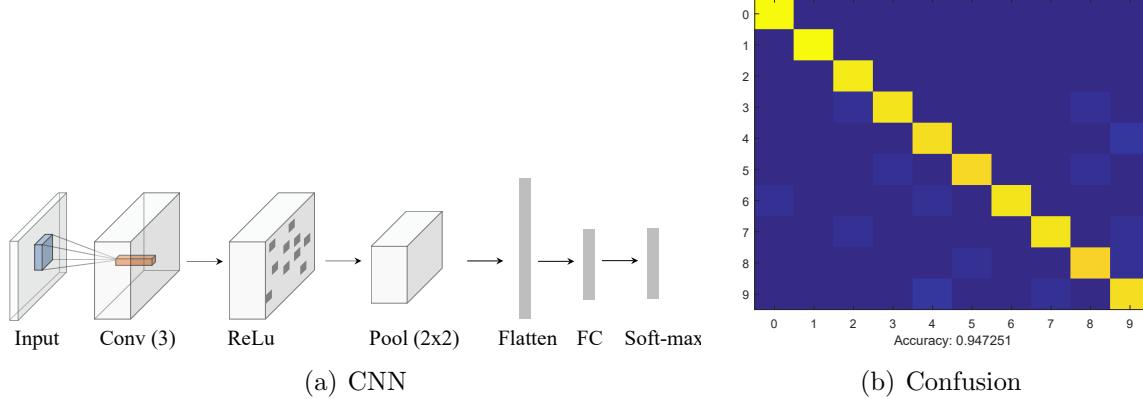


Figure 5: You will implement a convolutional neural network that produces accuracy more than **92%** on testing data.

You will implement a convolutional neural network (CNN) using a stochastic gradient descent method. We provide `main_cnn`. As shown in Figure 4(a), the network is composed of: a **single channel input ( $14 \times 14 \times 1$ )** → Conv layer ( **$3 \times 3$  convolution with  $3$  channel output** and **stride 1**) → ReLu layer → Max-pooling layer ( **$2 \times 2$  with stride 2**) → Flattening layer (**147 units**) → FC layer (**10 units**) → Soft-max.

```
def conv(x, w_conv, b_conv)
    ...
    return y
```

**Input:**  $x \in \mathbb{R}^{H \times W \times C_1}$  is an input to the convolutional operation,  $w_{conv} \in \mathbb{R}^{h \times w \times C_1 \times C_2}$  and  $b_{conv} \in \mathbb{R}^{C_2 \times 1}$  are weights and bias of the convolutional operation.

**Output:**  $y \in \mathbb{R}^{H \times W \times C_2}$  is the output of the convolutional operation. Note that to get the same size with the input, you may pad zero at the boundary of the input image.

**Description:** You can use `np.pad` for padding 0s at boundary. Optionally, you may use `im2col`<sup>1</sup> to simplify convolutional operation.

---

<sup>1</sup>[https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/making\\_faster.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/making_faster.html)

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

```
def conv_backward(dl_dy, x, w_conv, b_conv, y)
    ...
    return dl_dw, dl_db
```

**Input:**  $dl_dy$  is the loss derivative with respect to  $y$ .

**Output:**  $dl_dw$  and  $dl_db$  are the loss derivatives with respect to convolutional weights and bias  $w$  and  $b$ , respectively.

**Description:** Note that for the single convolutional layer,  $\frac{\partial L}{\partial x}$  is not needed. Optionally, you may use `im2col` to simplify convolutional operation.

```
def pool2x2(x)
    ...
    return y
```

**Input:**  $x \in \mathbb{R}^{H \times W \times C}$  is a general tensor and matrix.

**Output:**  $y \in \mathbb{R}^{\frac{H}{2} \times \frac{W}{2} \times C}$  is the output of the  $2 \times 2$  max-pooling operation with stride 2.

```
def pool2x2_backward(dl_dy, x, y)
    ...
    return dl_dx
```

**Input:**  $dl_dy$  is the loss derivative with respect to the output  $y$ .

**Output:**  $dl_dx$  is the loss derivative with respect to the input  $x$ .

# CSCI 5561: Assignment #4

## Convolutional Neural Network

---

```
def flattening(x)
```

```
    ...
```

```
    return y
```

**Input:**  $x \in \mathbb{R}^{H \times W \times C}$  is a tensor.

**Output:**  $y \in \mathbb{R}^{HWC}$  is the vectorized tensor (column major).

```
def flattening_backward(dl_dy, x, y)
```

```
    ...
```

```
    return dl_dx
```

**Input:**  $dl_dy$  is the loss derivative with respect to the output  $y$ .

**Output:**  $dl_dx$  is the loss derivative with respect to the input  $x$ .

```
function train_cnn(mini_batch_x, mini_batch_y)
```

```
    ...
```

```
    return w_conv, b_conv, w_fc, b_fc
```

**Output:**  $w_{conv} \in \mathbb{R}^{3 \times 3 \times 1 \times 3}$ ,  $b_{conv} \in \mathbb{R}^3$ ,  $w_{fc} \in \mathbb{R}^{10 \times 147}$ ,  $b_{fc} \in \mathbb{R}^{10 \times 1}$  are the trained weights and biases of the CNN.

**Description:** You will use the following functions to train a convolutional neural network using a stochastic gradient descent method: `conv`, `conv_backward`, `pool2x2`, `pool2x2_backward`, `Flattening`, `flattening_backward`, `fc`, `fc_backward`, `relu`, `relu_backward`, `loss_cross_entropy_softmax`. As a result of training, the network should produce more than 92% of accuracy on the testing data (Figure 5(b)).