UVA CS 4774

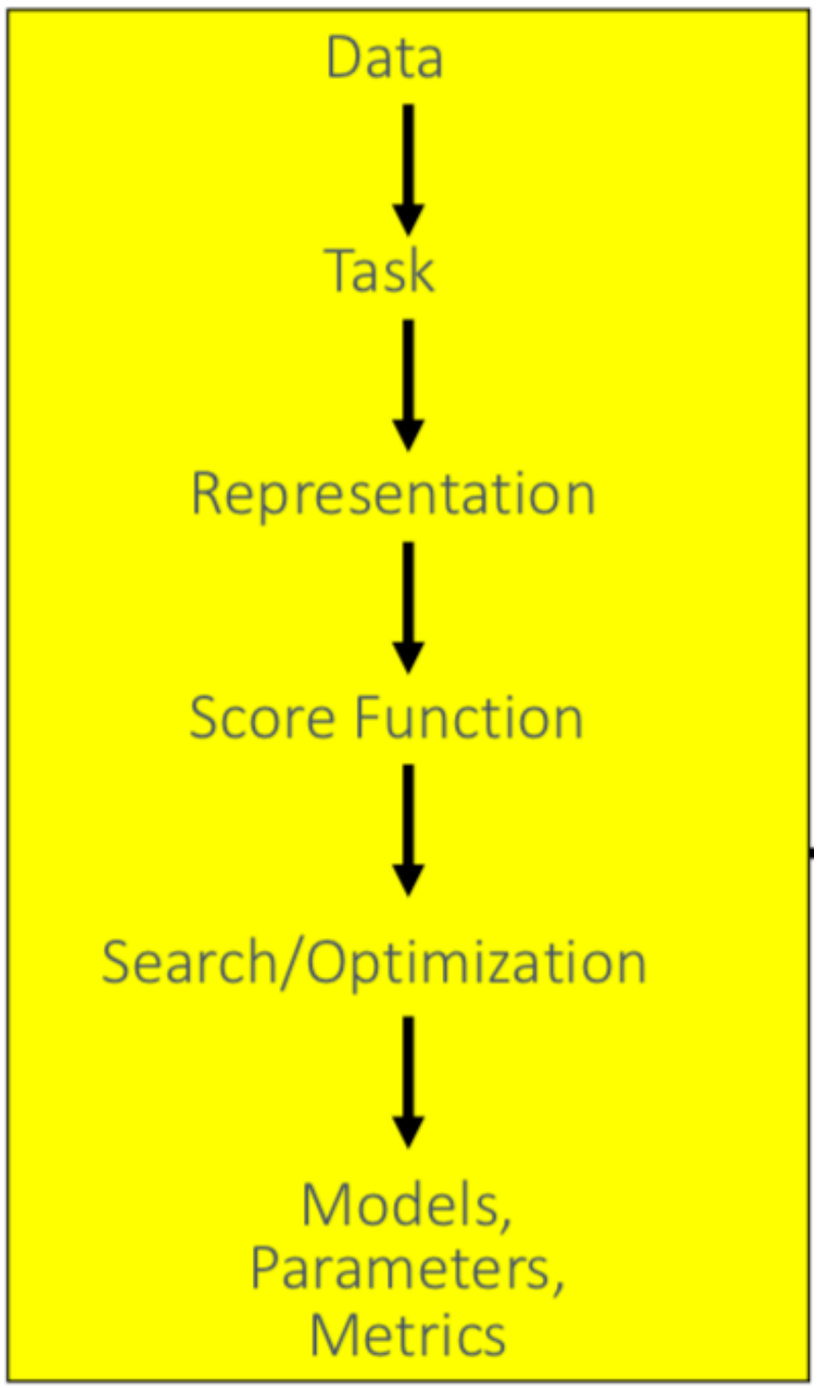
HW1/HW2/HW3 Review

Presented By: Zhe Wang

Professor: Dr. Yanjun Qi

November 12, 2020

First step: be very familiar with your data



Including but not limited to:

- The type of your data
- The shape of your data
- The noise level
- The format of your data

You could:

- print out the shape of your data
- Visualize your data
- Normalize your data to remove noise
- …

Common errors w.r.t data loading

1. Only a fraction of the dataset is loaded

2. Only a fraction of the features is loaded

Example:

In HW1, the dimension of the input X is [200, 2]

In HW2, for ridge regression, there are 200 data, each of them is of 102 dimension.

Common errors in HW1

Common errors in HW1: gradient calculation

1. Normal equations
2. Gradient descent
   - If sum squared loss is used, the gradient should be

$$loss = \sum_{i=1}^{n} ||y_i - x_i w||_w^2. \quad \frac{\partial loss}{\partial w} = 2\sum_{i=1}^{n} (-x_i)(y_i - x_i w)$$

   - If mean squared loss is used

$$loss = \frac{1}{n}\sum_{i=1}^{n} ||y_i - x_i w||_w^2. \quad \frac{\partial loss}{\partial w} = \frac{2}{n}\sum_{i=1}^{n} (-x_i)(y_i - x_i w)$$

3. Stochastic gradient descent

$$idvloss = ||y_i - x_i w||_w^2. \quad \frac{\partial idvloss}{\partial w} = 2(-x_i)(y_i - x_i w)$$

4. Minibatch gradient descent

$$batchloss = \frac{1}{m}\sum_{i=1}^{m} ||y_i - x_i w||_w^2. \quad \frac{\partial batchloss}{\partial w} = \frac{2}{m}\sum_{i=1}^{m} (-x_i)(y_i - x_i w)$$

Common errors in HW1: Data shuffling

1. Shuffle X and Y in the same order

```
x_new = random_shuffle(x)
y_new = random_shuffle(y)
```

→ Wrong!!

One possible solution

```
new_index = random_shuffle(index)
x_new = x[new_index]
y_new = y[new_index]
```

Common errors in HW1: Data shuffling

2. Shuffling should be applied after each epoch.

```
def minibatch_gradient_descent(data, num_epoch):

    new_x, new_y = shuffle(x, y)

    for t in range(num_epoch):

            perform minibatch_gradient_descent on new_x, new_y

    return thetas
```

→ Wrong!!

Correct solution

```
def minibatch_gradient_descent(data, num_epoch):

      for t in range(num_epoch):

            new_x, new_y = shuffle(x, y)
            perform minibatch_gradient_descent on new_x, new_y

    return thetas
```

Common errors in HW1: Minibatch Organization

3. For each epoch, the whole dataset should be used

```
def minibatch_gradient_descent(data, num_epoch):

    for t in range(num_epoch):
        new_x, new_y = shuffle(x, y)
        batch_x = new_x[:batch_size]
        batch_y = new_y[:batch_size]
        batch_grad = grad(thetas, batch_x, batch_y)
        update thetas

    return thetas
```

→ Wrong!!

Correct solution

```
def minibatch_gradient_descent(data, num_epoch):

    for t in range(num_epoch):
        new_x, new_y = shuffle(x, y)

        for j in range(0, len(x), batch_size):
            batch_x, batch_y = new_x[j:j+batch_size], new_y[j:j+batch_size]
            batch_grad = grad(thetas, batch_x, batch_y)
            update thetas

    return thetas
```

Common errors in HW1: Shape Matching

3. broadcasting in python

```
y: vector of shape (3, )
y_pred: vector of shape (3, 1)
diff  = y-y_pred
diff: vector of shape (3, 3)
```

```
y = np.array([1, 2, 3])
y_pred = np.array([[1], [2], [3]])
diff  = y - y_pred
diff = np.array([[0, 1, 2], [-1, 0, 1], [-2, -1, 0]])
```

Check the shape of the predictions and labels

```
def loss(y, y_pred):
    if y.shape==y_pred.shape:
        diff = y - y_pred
        loss = …
   else:
        y_pred = np.squeeze(y_pred)
        diff = y - y_pred
        loss = …
   return loss
```

Common errors in HW2

Common errors in HW2: Data Loading

1. The first thing to do: check the shape of your data.

PolyRegression

```
def load_data_set(filename):
    x = np.loadtxt(filename, usecols=(0))
    y = np.loadtxt(filename, usecols=(1))
    return x, y
    ## x.shape = (200, )     y.shape=(200,)
```

RidgeRegression

```
def load_data_set(filename):
    x = np.loadtxt(filename, usecols=(range(102)))
    y = np.loadtxt(filename, usecols=102)
    return x, y
    ## x.shape = (200, 102)   y.shape = (200, )
```

Warning: do not use your local path within the function

```
def load_data_set(filename):
    filename = 'Zhe/Desktop/HW2/dataPoly.txt'
    x = np.loadtxt(filename, usecols=(0))
    y = np.loadtxt(filename, usecols=(1))
    return x, y
```

Common errors in HW2: Procedures for model selection

```
1. train_set, test_set = data_split(dataset)

2. new_train_set, new_valid_set = data_split(train_set)

3. valid_losses = []

4. for hyper_param in hyper_param_set:
        theta* = model_train(new_train_set, hyper_param)
        valid_loss = model_eval(new_valid_set, theta*, hyper_param)
        valid_losses.append(valid_loss)

5. best_hyper = argmin(valid_losses)

6. theta* = model_train(train_set, best_hyper)

7. reported_performace = model_eval(test_set, theta*, best_hyper)
```

If there is no model selection, only 1, 6, 7 will be remained.

Common errors in HW2: Ridge Regression

1. K-fold CV: Use 1/k of the training set as validation set, not just k data.

```
new_valid_set = train_set[:k]
new_train_set = train_set[k:]
```

→ Wrong!!

2. Normal equations for ridge regression.

$$\theta^*_{ridge} = (X^T X + \lambda I)^{-1} X^T Y \longleftrightarrow \theta^* = (X^T X)^{-1} X^T Y$$

3. The calculation of L2 norm

$$\mathbf{x} = (x_1, x_2, \cdots, x_n) \longrightarrow ||\mathbf{x}||_2 = \sqrt{\sum x_i^2}$$

Common errors in HW2: Gradient Explosion

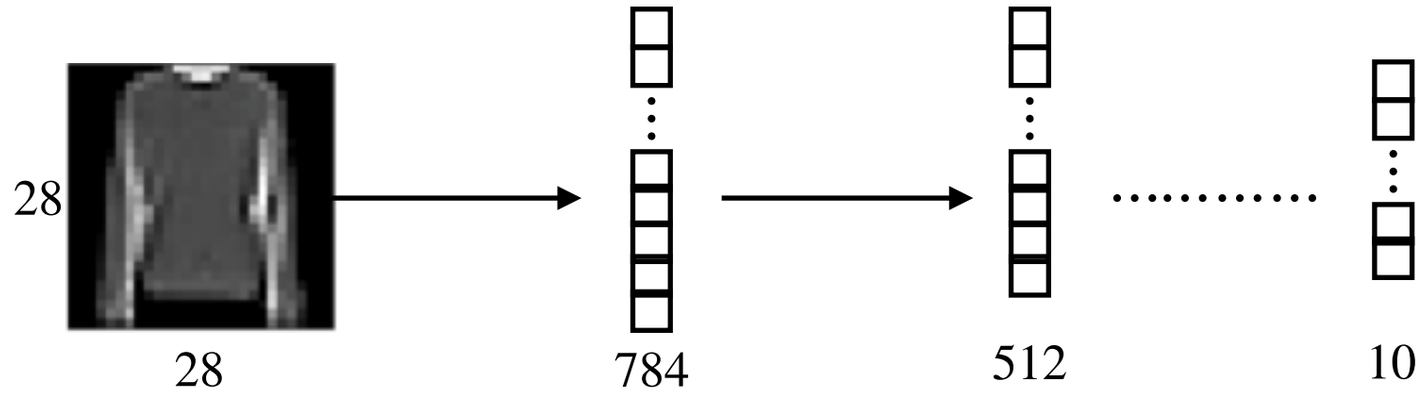
```
# plot for hyper-parameter that is not the best (different from part 2);
    x_train = increase_poly_order(x_train, 11)
    x_test = increase_poly_order(x_test, 11)
    gbest_thetas = gradient_descent(x_train, y_train, 0.005, 1000)

    best_fit_plot()
    plot_epoch_losses()
```

???

Common errors in HW3

Common errors in HW3: Dimension Matching

1. For dense layers, the input dimension of the layer should match the output dimension of the previous layer



2. For convolutional layers, the input channel of the layer should match the output channel of the previous layer.

`torch.nn.Conv2d(in_channels: int, out_channels: int, kernel_size))`

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

1. How many matrices are outputted by your first convolutional layer when receiving a single testing image

   Answer: C_out

2. What are the dimensions of these matrices

   Answer: [H_out, W_out]

3. What are the dimensions of one of these matrices after it passes through your first maxpooling layer

   Answer: [H_out/2, W_out/2] (depending on your pooling size and padding strategy)

FYI:

For MLP models in submissions: number of trainable parameters <1million

For CNN models in submissions: number of trainable parameters <1million

BERT base uncased: number of trainable parameters ~ 110M