

March 23<sup>rd</sup>, 2021

# AI+BD ML Lab. Day 2

## Training Methodologies with Logistic Regression

Jio Gim <jio.gim@postech.edu>



# Contents

## 1. Today's Goals

## 2. Basic Flow of DL

- ★ Flow Diagram & Code Example
- ★ Apply to Logistic Regression

## 3. Basic Training Techniques

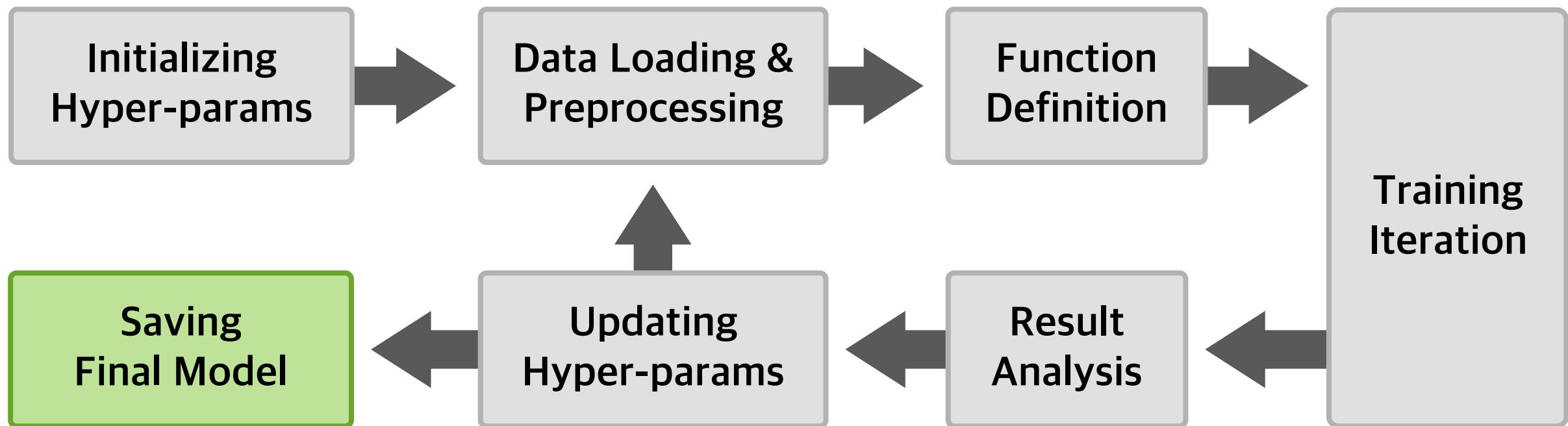
- ★ Weight Initialization
- ★ Hyper-parameter Searching
- ★ Apply to Logistic Regression



1. Understanding **basic flow of deep learning & code**
2. Applying (1) to the **logistic regression** model with mini-batch SGD
3. Understanding the differences from **weight initialization**
4. Finding proper **hyper-parameters** to get fast convergence speed

# Basic Flow of DL

4



# Basic Flow with <code />

5



## Hyper-parameter Initialization

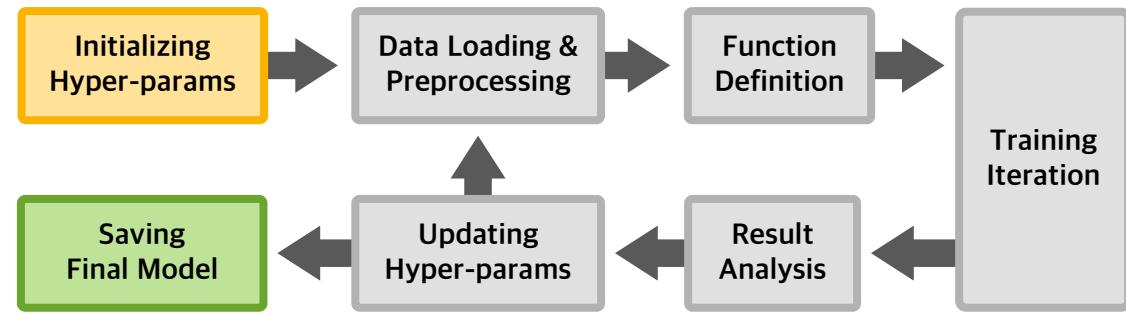
```
### Common variables
iteration_idx = 0
earlystop_cnt = 0
minimum_loss = 1e+20

### Records upon iterations
iteration_idx_records = []
loss_records = []
gradnorm_records = []
alphanorm_records = []
noisenorm_records = []
time_records = []
earlystop_p_records = []

### Data
s_t_np = s_np.transpose()
s_t = torch.Tensor(s_t_np).to(device)
r = torch.Tensor(r_np).to(device)

### Modules
net = torch.nn.Linear(s_shift_len, 1, bias=False).to(device)
opt = torch.optim.LBFGS(
    net.parameters(), history_size=200, max_iter=30, max_eval=45,
)
# * Note that this terms are related to "considering time."
# * That is, as you set terms_weights[0] in large value, the model will search
#   noise-degrading way first, and noise-degrading direction's gradient will
#   pumped in the scale. Then when noise-degrading direction's gradient becomes
#   zero, the second term will be main target of optimum finding.
# * From the experience, I recommend to set first term weight high but not
#   exceeds `s_shift_len`.
terms_weights = [181, 1]

### Weight Initialization
torch.nn.init.zeros_(net.weight.data)
```



```
### Torch Device
# * Available options : cpu, cuda:0, cuda:1, and so on
# * Multiple gpus are not supported for now
# * Note that LBFGS is memory-intensive, so cpu may work more efficiently.
device = 'cpu'

### Iteration Settings
iteration_max = 200
log_interval = 5

### Early Stopping
earlystop_enable = True
earlystop_patience = 5
earlystop_log_from = 4
earlystop_threshold = 0.0001

### Recording Approximation Procedures
growth_album_enable = False
growth_album_ylim = [-0.002, 0.020]

### Loss Terms
# * Note that this terms are related to "considering time."
# That is, as you set terms_weights[0] in large value, the model will search
# noise-degrading way first, and noise-degrading direction's gradient will
# pumped in the scale. Then when noise-degrading direction's gradient becomes
# zero, the second term will be main target of optimum finding.
# * From the experience, I recommend to set first term weight high but not
# exceeds `s_shift_len`.
terms_weights = [181, 1]
```

◀ Initialization code ▶

# Basic Flow with <code />

6



## Data Load & Preprocessing

```
[19]: cir_target = hwk.read_mat('../data/gen2-dev/piai_0918.mat', 'cir_4m')[:, 5].squeeze()

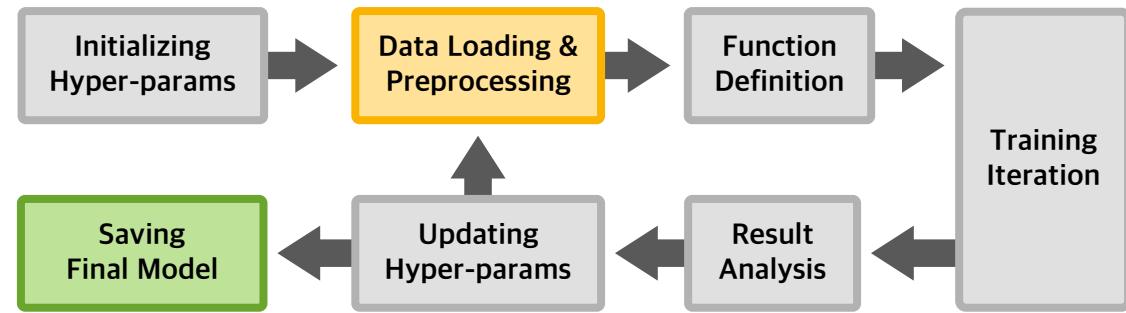
r_np = np.zeros((2 * samp_len, 1))
tmp_complex = cir_target.reshape([samp_len, 1])
r_scale = np.sqrt(np.sum(np.power(np.abs(cir_wire), 2)) / np.sum(np.power(np.abs(tmp_complex), 2)))
tmp_complex = tmp_complex * r_scale / 1000
print(f'r_scale = {r_scale}')
r_np[:, 0] = np.vstack([
    np.real(tmp_complex), np.imag(tmp_complex)
]).squeeze()

print(f'R.shape = {r_np.shape}')

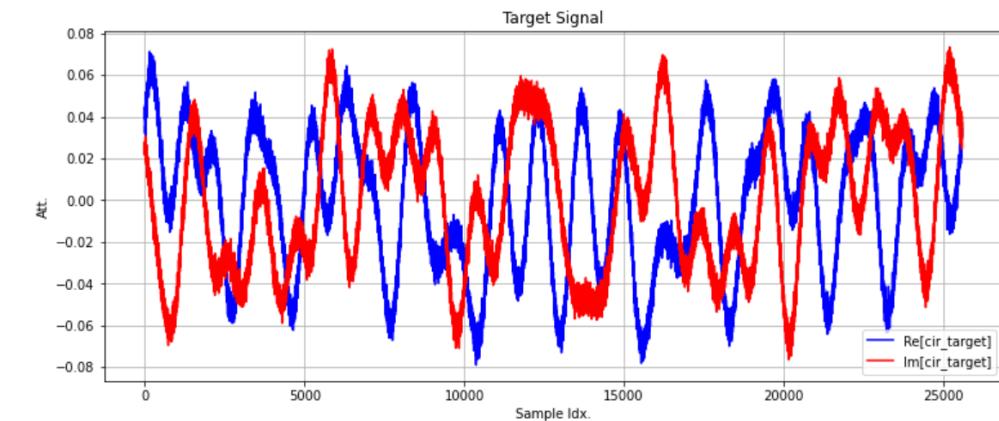
r_scale = 0.6657971810705
R.shape = (51200, 1)
```

▲ Loading MATLAB .mat file & Automatically scaling data

Check your data after preprocessing! ▶



```
[4]: fig = plt.figure(1998, figsize=(12.5, 5))
plt.clf()
plt.plot(np.real(tmp_complex), color='blue')
plt.plot(np.imag(tmp_complex), color='red')
plt.legend(['Re[cir_target]', 'Im[cir_target']'])
plt.xlabel('Sample Idx.')
plt.ylabel('Att.')
plt.title('Target Signal')
plt.grid()
plt.show()
```





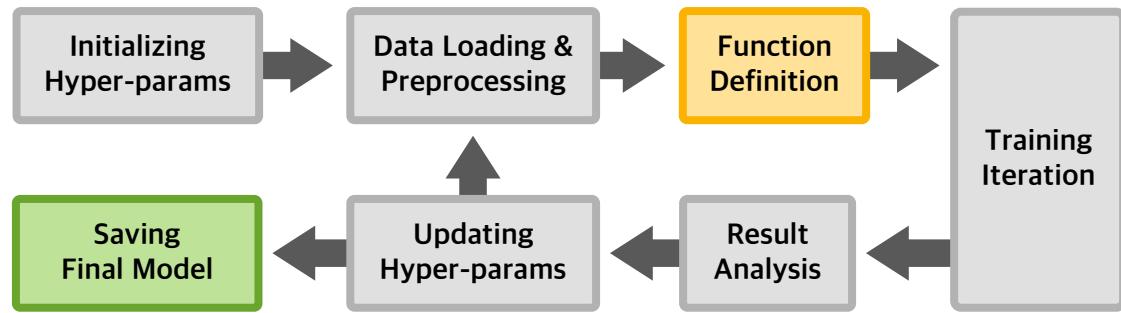
## Function Definition

```
class ResNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 16

        self.conv1 = nn.Conv2d(3, 16, kernel_size=3,
                            stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(16)
        self.layer1 = self._make_layer(16, 2, stride=1)
        self.layer2 = self._make_layer(32, 2, stride=2)
        self.layer3 = self._make_layer(64, 2, stride=2)
        self.linear = nn.Linear(64, num_classes)

    def _make_layer(self, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(BasicBlock(self.in_planes, planes, stride))
            self.in_planes = planes
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = F.avg_pool2d(out, 8)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```



What functions can be defined?

- model class

◀ Defined ResNet model class inheriting torch.nn.Module

# Basic Flow with <code />

8



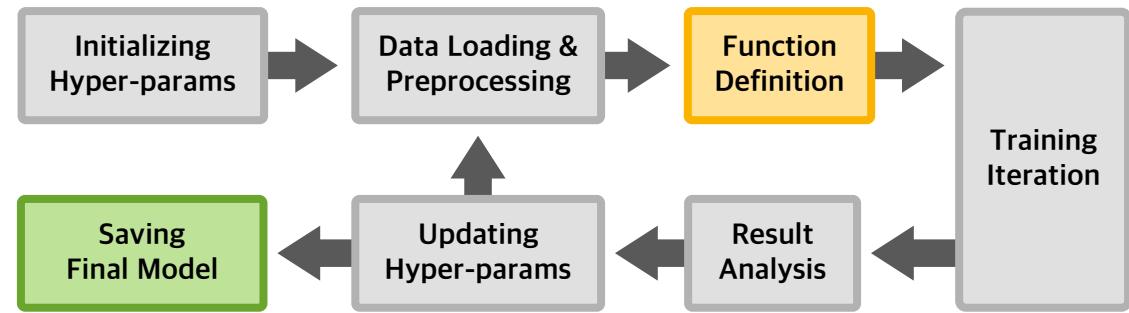
## Function Definition

```
def epoch():
    def closure():
        #####
        ## loss part start #####
        #####
        noise = r - net(s_t)
        noise_norm = torch.norm(noise)
        alpha_norm = torch.norm(net.weight)

        # objective : minimize noise
        objective = terms_weights[0] * torch.pow(noise_norm, 2) / 2

        # constraint : alpha >= 0
        alpha_maxterm = torch.max(
            torch.zeros(s_shift_len).to(device),
            -net.weight.squeeze(),
        )
        alpha_constraint = terms_weights[1] * torch.sum(alpha_maxterm)

        loss = objective + alpha_constraint
        #####
        ##### end #####
        #####
        opt.zero_grad()
        loss.backward()
        return loss
    loss = opt.step(closure)
    last_loss = loss.item()
    last_grad = net.weight.grad
    last_alpha = get_numpy(net.weight).squeeze().reshape([s_shift_len, 1])
    last_noise = r_np - np.matmul(s_t_np, last_alpha)
    return (last_loss, last_alpha, last_noise, last_grad)
```



## What functions can be defined?

- model class
- epoch function

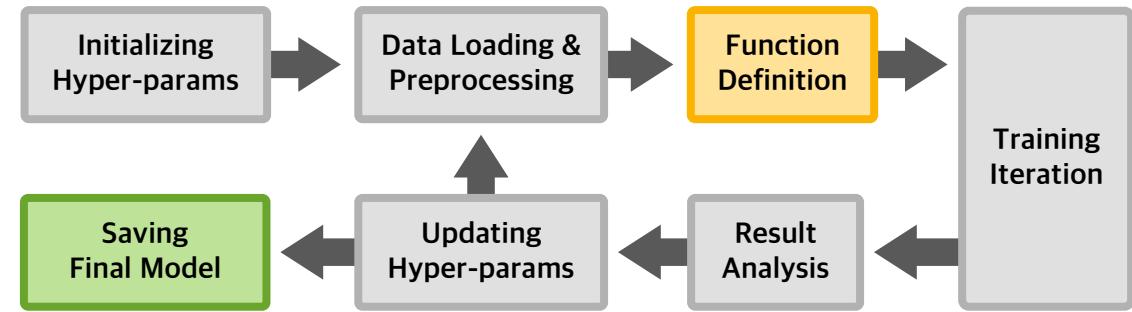
◀ Defined epoch function



## Function Definition

```
def print_log_atomic(_idx, _iter, _loss, _grad, _an, _nn, _time, _p=None):
    tmp_str = f'Iter: {_iter:5d} >> loss {_loss:<17.10f}  G {_grad:<14.7f}' \
              + f' ||a|| {_an:<8.5f}  ||N|| {_nn:<8.5f}  ⏳ {_time:7.3f}s'
    if _p is not None and _p >= earlystop_log_from:
        tmp_str = tmp_str + f' | p={_p}'
    print(tmp_str)

def print_log(figure=True, alpha=None, noise=None, grad=None):
    if figure:
        plt = draw_figure(None, alpha, noise, grad, False)
        clear_output(wait=True)
        plt.show()
        print('')
    for iii in reversed(range(len(loss_records))):
        print_log_atomic(
            iii,
            iteration_idx_records[iii],
            loss_records[iii],
            gradnorm_records[iii],
            alphanorm_records[iii],
            noisenorm_records[iii],
            time_records[iii],
            earlystop_p_records[iii] if earlystop_enable else None,
        )
    print('')
```



## What functions can be defined?

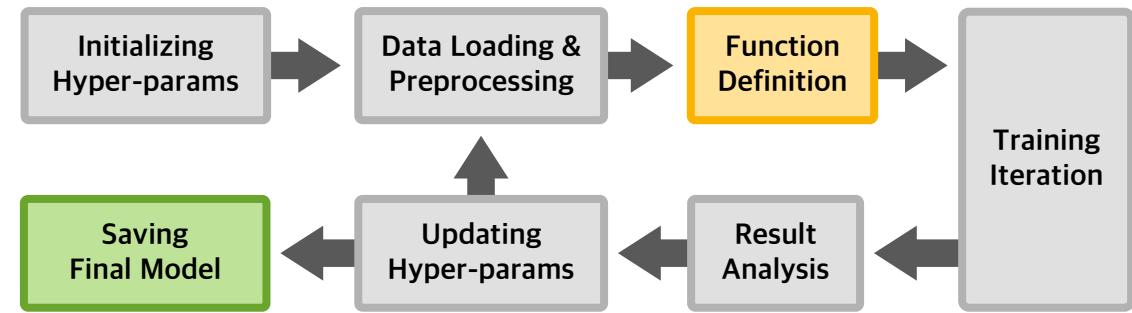
- model class
- epoch function
- logging function

◀ Defined logging function



## Function Definition

```
def draw_figure(title=None, weights=None, noise=None, grad=None, show=True):
    if weights is None: weights = get_numpy(net.weight).squeeze()
    fig = plt.figure(2000, figsize=(12.5, 5), dpi=240)
    plt.clf()
    fig.patch.set_facecolor('#ffffff')
    ax_noise = fig.add_subplot()
    ax_weight_y = ax_noise.twinx()
    ax_weight = ax_weight_y.twiny()
    ax_grad_y = ax_weight.twinx()
    ax_grad = ax_grad_y.twiny()
    ax_grad_y.axis('off')
    ax_grad.axis('off')
    if noise is not None:
        line_noise1, = ax_noise.plot(
            noise[:samp_len], color='blue', alpha=0.35,
        )
        line_noise2, = ax_noise.plot(
            noise[samp_len:], color='red', alpha=0.35,
        )
        ax_noise_color = np.array([164, 107, 255]) / 255
        ax_noise.axhline(0, color=ax_noise_color)
        ax_noise.set_xlabel('Sample Idx.', color=ax_noise_color)
        ax_noise.set_ylabel('Noise', color=ax_noise_color)
        ax_noise.set_ylim([-0.015, 0.015])
        ax_noise.spines['left'].set_color(ax_noise_color)
        ax_noise.spines['bottom'].set_color(ax_noise_color)
        ax_noise.tick_params(axis='both', colors=ax_noise_color)
```



## What functions can be defined?

- model class
- epoch function
- logging function
- plotting function

You can make other things  
into a function, of course~

◀ Defined plotting function

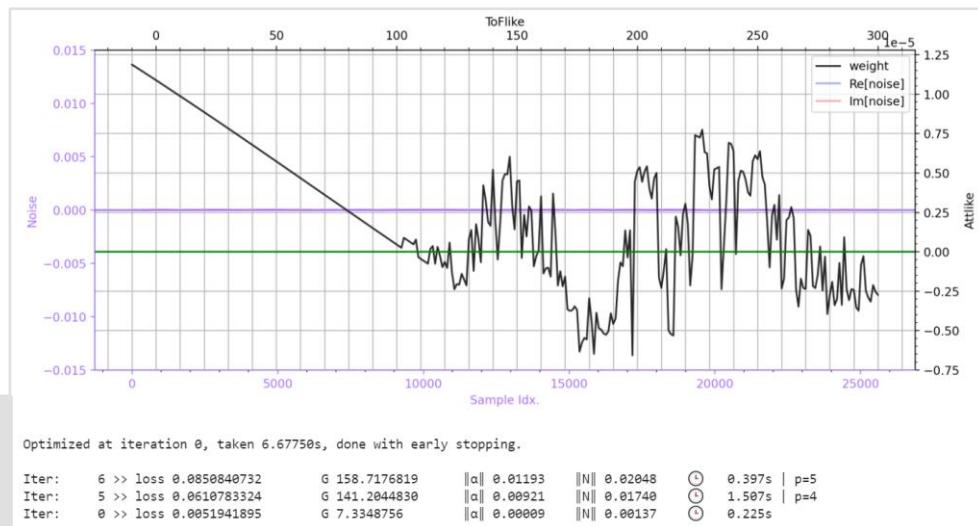
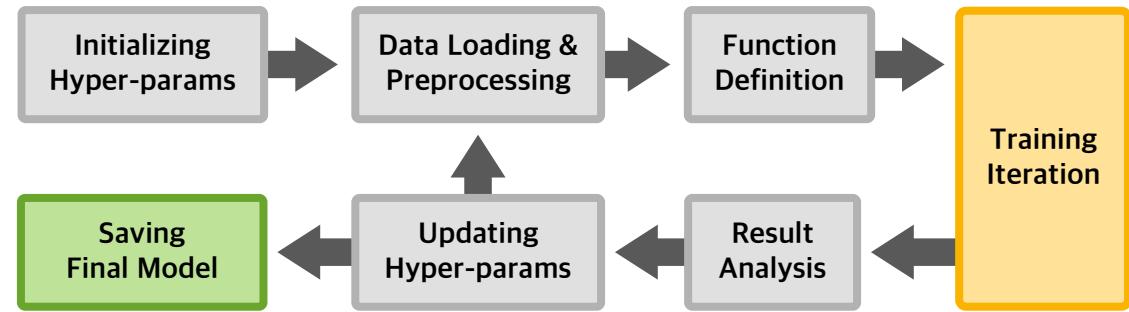
# Basic Flow with <code />

11



## Training Iteration

```
### Iterations
# * Because the loss we get from LBFGS.step is "prev_loss", so using earlystop,
#   we should extract the weight of a previous network from the inflection pt.
#   This is implemented with `last_net2`.
last_loss = float('inf')
last_net, last_net_iter = deepcopy(net), 0
last_net2, last_net2_iter = deepcopy(net), 0
t_all_begin = time.time()
t_begin = time.time()
while True:
    check_exit_result = check_exit()
    if check_exit_result != 'keep going':
        alpha = get_numpy(last_net2.weight).squeeze().reshape([s_shift_len, 1])
        noise = r_np - np.matmul(s_t_np, alpha)
        plot = draw_figure(
            f'Final Model Status [Iteration {last_net2_iter}]\n',
            alpha, noise, None, False,
        )
        clear_output(wait=True)
        t_all_end = time.time()
        plot.show()
        print('')
        print(
            f'Optimized at iteration {last_net2_iter}, '\
            + f'taken {t_all_end - t_all_begin:.5f}s, '\
            + f'done with {check_exit_result}.'
        )
        print('')
        print_log.figure=False
        break
    last_loss, last_alpha, last_noise, last_grad = epoch()
```



Training output ►

◀ Training iteration while

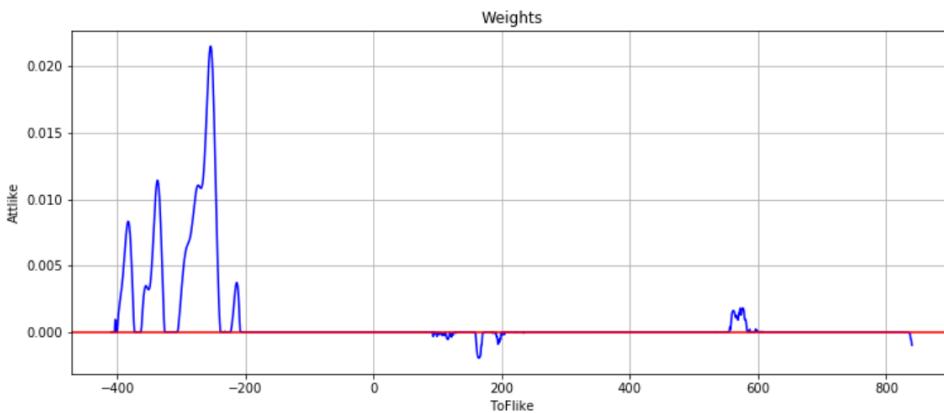
# Basic Flow with <code />

12

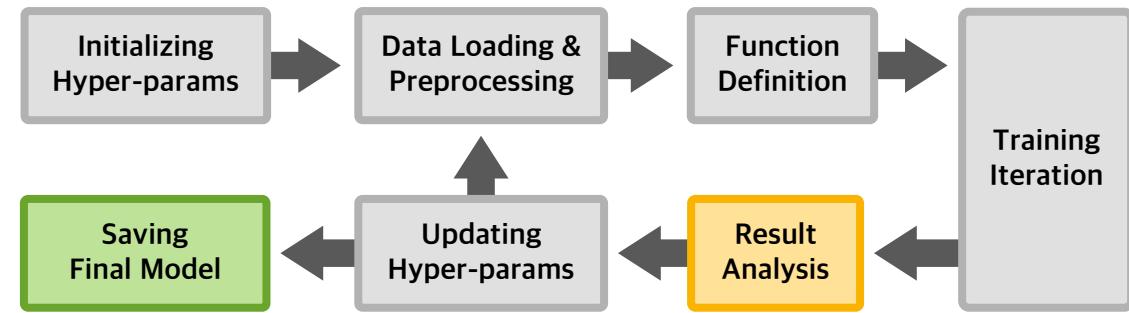


## Result Analysis

```
: fig = plt.figure(2001, figsize=(12.5, 5))
plt.clf()
plt.plot(s_shift_range, result, color='blue')
plt.axline(0, color='red')
plt.xlabel('ToFlike')
plt.ylabel('Attlike')
plt.title(f'Weights')
plt.grid()
plt.show()
```



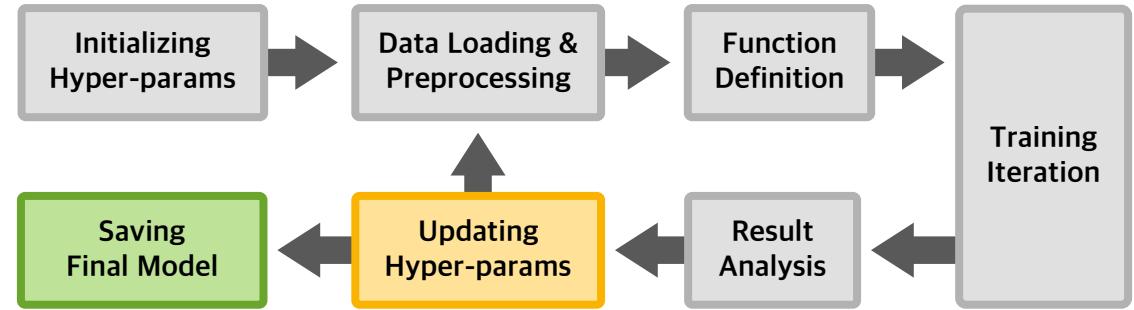
◀ Result analysis ▶



```
target_pow = 0
for idx in range(len(r_np)):
    target_pow += np.power(np.abs(r_np[idx]), 2)
noise_pow = 0
for idx in range(len(noise_complex)):
    noise_pow += np.power(np.abs(noise_complex[idx]), 2)
signal_pow = target_pow - noise_pow
snr_approx = np.log10(signal_pow / noise_pow) * 10
print(f'Target Signal PWR [R] : {target_pow[0]}')
print(f'Approximated Signal PWR [A=S^T·a] : {signal_pow[0]}')
print(f'Result Noise PWR [N=R-A] : {noise_pow[0]}')
print(f'Approximated SNR [A:N] : {snr_approx[0]}')

Target Signal PWR [R] : 44.79031572701082
Approximated Signal PWR [A=S^T·a] : 44.33525883161393
Result Noise PWR [N=R-A] : 0.4550568953968937
Approximated SNR [A:N] : 19.88683548701614
```

## Updating Hyper-parameters



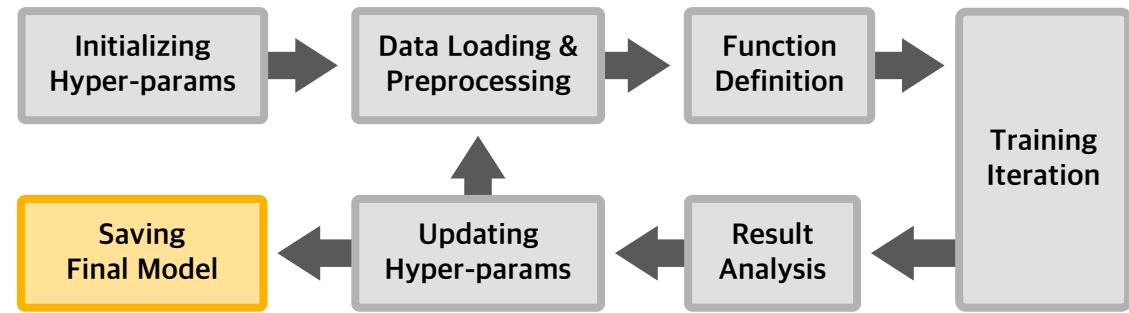
No code, do this by your self.



## Saving Final Model

```
: confdict = {  
    'shift_from': s_shift_from,  
    'shift_to': s_shift_to,  
    'earlystop': earlystop_patience,  
    'term_weights': terms_weights,  
    'samp_len': samp_len,  
}  
  
sav_alpha = result.reshape(s_shift_len, 1)  
sav_noise = noise_complex.reshape(samp_len, 1)  
  
hwk.save_mat('./gen2.0.out.mat', 'hwk_03.ipynb', sav_alpha, sav_noise, confdict)
```

▲ Saving model as .mat for further investigation



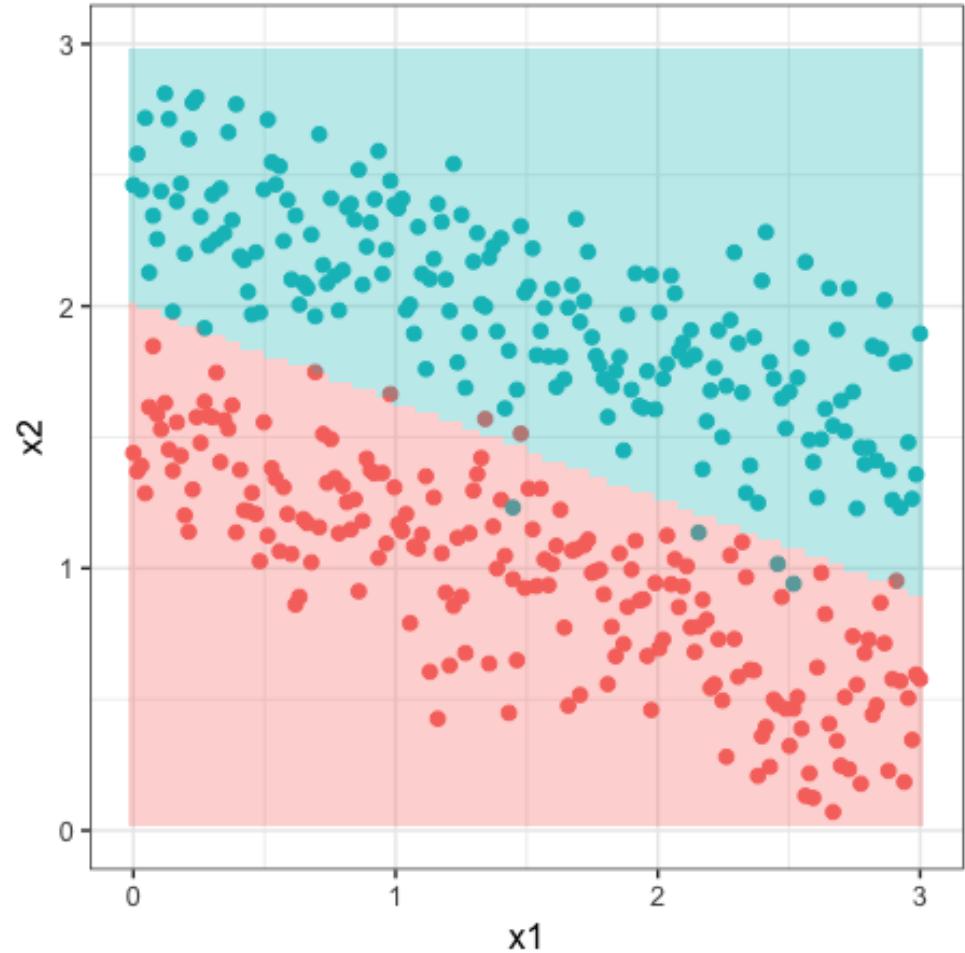
# Backgrounds

For your information & reminding



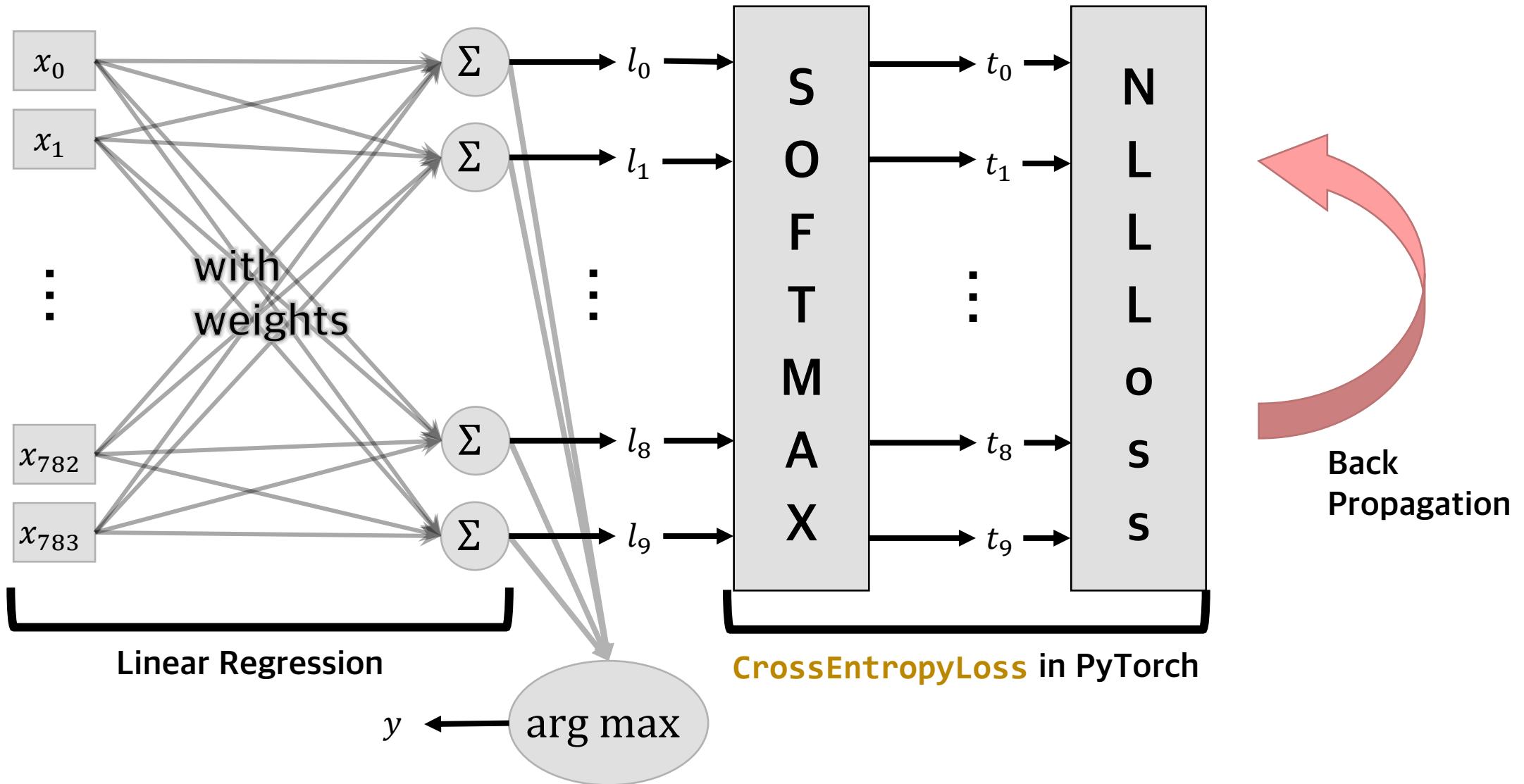
- **Regression** = “Fitting”
- **Logistic** = “Linear Logistic”
  - Decision boundary to be learned is linear
  - But output is not “score”, but logistic probability (softmax)
  - Logistic value is more fit to real world’s probability than linear score
  - Let’s go

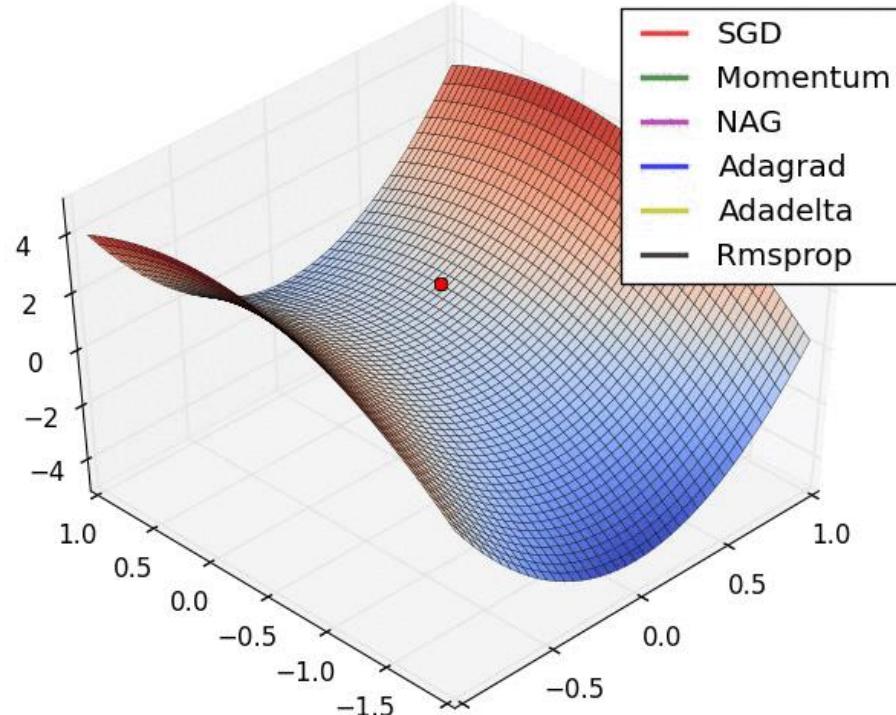
Decision Boundary for Logistic Regression



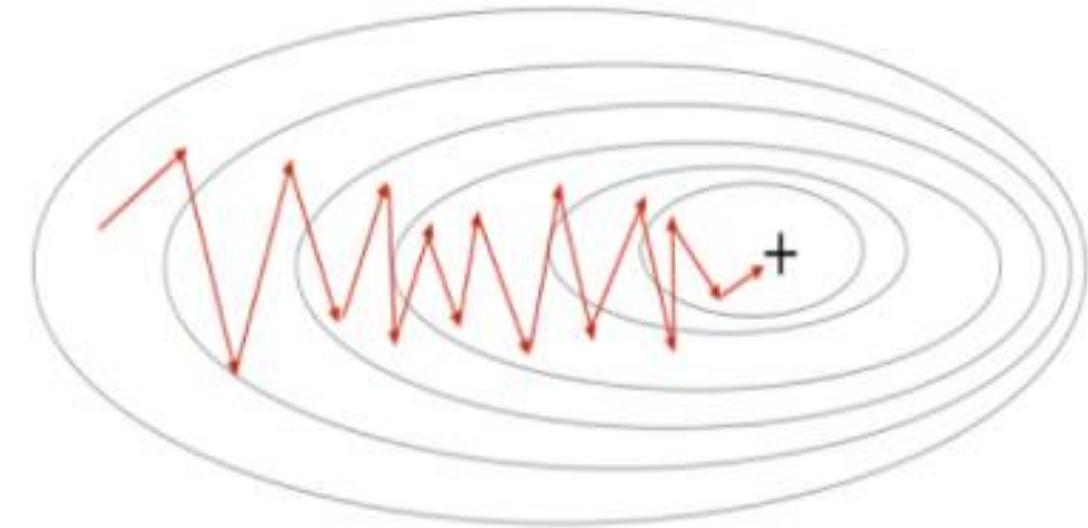
# How is LR expressed in learning model?

17





Gradient Descent



Because we randomly pick dataset to train



Stochastic Gradient Descent

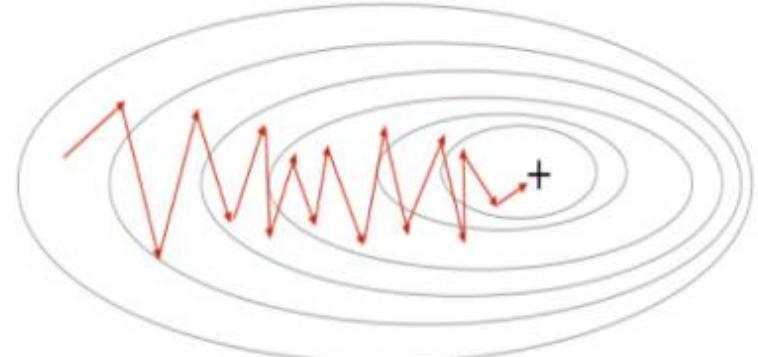


## Why Mini-batch SGD?

SGD samples 1 data from dataset at each iteration

- This one-shot learning makes learning oscillate!
- So put more data pair and accumulate gradients!

Mini-batch SGD samples many data at each iteration

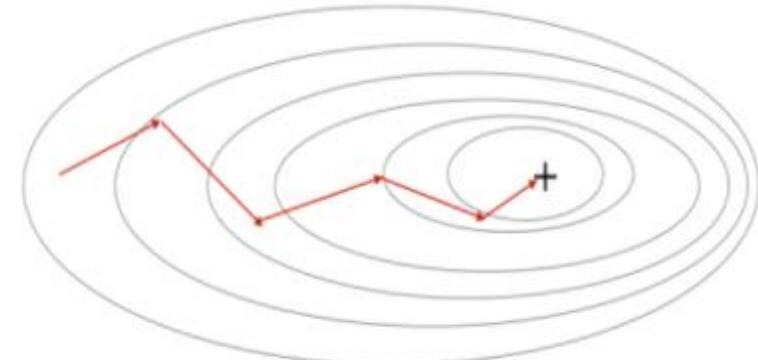


Stochastic Gradient Descent

## How about learn all data pair at an iteration?

Good approach, so-called “Batch SGD”! But…

- It takes a lot of time and memory to learn all data pair when dataset becomes larger
- Hard to overcome local optimal



Mini-batch SGD



- We gonna attach “logistic function” so that the linear regression model can do classification task!
- We gonna give a dataset of specified size (not all !) for better gradient descent performance!

# It's coding time

Let's fill the I.P.Y.N.B



**PDF Version of this PPT :**

<https://laf.gg/aibd-pdf-2>

**Base .ipynb :**

<https://laf.gg/aibd-py-2>

**Required Python Packages :**

- matplotlib numpy scikit-learn
- torch torchvision seaborn



# 1. Library Importation & Device Preparation

Already prepared for you!

```
# You don't need to edit this section today.
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import time
import torch
import torch.nn as nn

from IPython.display import clear_output
from multiprocessing import cpu_count
from sklearn.metrics import confusion_matrix
from torch.optim import SGD
from torch.utils.data import DataLoader, random_split
from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor

MNIST.resources = [
(
    'https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz',
    'f68b3c2dcbeaaa9fbdd348bbdeb94873'
),
(
    'https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz',
    'd53e105ee54ea40749a09fcbcd1e9432'
),
(
    'https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz',
    '9fb629c4189551a2d022fa330f9573f3'
),
(
    'https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz',
    'ec29112dd5afa0611ce80d1b7f02629c'
)
]

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(f'{"CPU" if device == "cpu" else "GPU"} will be used in training/validation.')
```

**Dataset URL in `torchvision` package is outdated**

- So temporarily replaced source URL
- Future `torchvision` package will contain working URLs

```
# Data Loader  
batch_size = 32  
  
# Learning  
logging_dispfig = True  
maximum_epoch = 25  
learning_rate = 0.1
```

## 2. Hyper- Parameters

24



Already prepared for you!

This section is generally filled during coding, but let's fill it in advance today.

- batch\_size : the number of dataset in one mini-batch (generally a multiple of 2)
- logging\_dispfig : switch for displaying learning graph during training
- maximum\_epoch : training epoch count
- learning\_rate : learning rate what you know

### 3. Data Load & Preprocessing

25



But no preprocessing today.

Download training dataset

```
# Load dataset into python variable
train_data = MNIST("./", train=True, transform=ToTensor(), target_transform=None, download=True)
train_data, valid_data = random_split(train_data, [54000, 6000])
test_data = MNIST("./", train=False, transform=ToTensor(), target_transform=None, download=True)
```

#### ToTensor?

Original dataset is in form of “PIL (Python Image Library)” image, so using **torchvision.transforms.ToTensor()** we can convert the image into torch Tensor.

### 3. Data Load & Preprocessing

26



But no preprocessing today.

Split train dataset to training set & validation set (ratio = 9:1)

```
# Load dataset into python variable
train_data = MNIST("./", train=True, transform=ToTensor(), target_transform=None, download=True)
train_data, valid_data = random_split(train_data, [54000, 6000])
test_data = MNIST("./", train=False, transform=ToTensor(), target_transform=None, download=True)
```

Download test dataset

#### Datasets

- `train_data` : dataset used at gradient descent (real training)
- `valid_data` : dataset used to check learning progress
- `test_data` : dataset used to evaluate performance of model

### 3. Data Load & Preprocessing

27



But no preprocessing today.

```
# Check the data
print(f'Train dataset length = {len(train_data)}')
print(f'Valid dataset length = {len(valid_data)}')
print(f'Test dataset length = {len(test_data)}\n')

train_0_x, train_0_y = train_data[0]
print(f'Content of Y (Label, type={type(train_0_y)}) = {train_0_y}')
print(f'Shape of X (Data, type={type(train_0_x)}) = {train_0_x.shape}')
plt.figure(1)
plt.imshow(train_0_x.squeeze())
plt.title(f'train_0_x ({train_0_x.squeeze().shape})')
plt.show()
```

#### Python Template Literal <f> Basic

From python 3.6, template literal <f> is supported

- Write <f> before opening apostrophe
- In string, use {variable} to embed variable

### 3. Data Load & Preprocessing

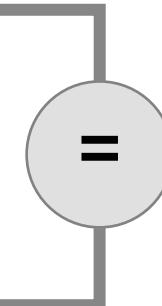
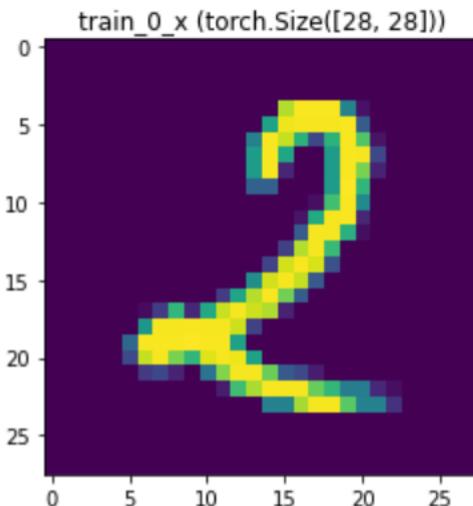
28



But no preprocessing today.

```
Train dataset length = 54000  
Valid dataset length = 6000  
Test dataset length = 10000
```

```
Content of Y (Label, type=<class 'int'>) = 2  
Shape of X (Data, type=<class 'torch.Tensor'>) = torch.Size([1, 28, 28])
```



#### Expected Output

Check “Content of Y” equals with the picture

### 3. Data Load & Preprocessing

29



**shuffle** option

if **True**, DataLoader will shuffle dataset every epoch

But no preprocessing today.

```
# Create data loader
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, pin_memory=True,
                         drop_last=True)
valid_loader = DataLoader(valid_data, batch_size=len(valid_data), pin_memory=True)
test_loader = DataLoader(test_data, batch_size=len(test_data), pin_memory=True)
```

#### DataLoader (**torch.utils.data.DataLoader**)

Useful wrapper for data feeding

- Cut dataset based on **batch\_size** → We are implementing mini-batch using this feature
- Make dataset into “Python **Iterable** object” → We can call **enumerate()**, **iter()** on **DataLoader**



## drop\_last option

when dataset length is not a multiple of batch\_size, if drop\_last=True, it drops last small data.

Example) len(dataset)=38, batch\_size=16  
⇒ 6 data pairs will not be used

## 3. Data Load & Preprocessing

But no preprocessing today.

```
# Create data loader
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, pin_memory=True,
                         drop_last=True)
valid_loader = DataLoader(valid_data, batch_size=len(valid_data), pin_memory=True)
test_loader = DataLoader(test_data, batch_size=len(test_data), pin_memory=True)
```

## pin\_memory option

when use CUDA, if True, dataset will be fixed on CUDA memory



### 3. Data Load & Preprocessing

But no preprocessing today.

#### enumerate function

make an enumerator who iterates **Iterable** with index

Enumerator will return **(index, content)** tuple(pair)

```
# Examine the data loader
train_enumerator = enumerate(train_loader)
ex_batch_idx, (ex_data, ex_label) = next(train_enumerator)
print(f'Idx: {ex_batch_idx} / X.shape = {ex_data.shape} / Y.shape = {ex_label.shape}\n')
print(f'Y[0:{batch_size}] = {ex_label}')

preview_index = 0
plt.figure(2)
plt.imshow(ex_data[preview_index, 0, :, :])
plt.title(f'Batch example data [{preview_index}], label={ex_label[preview_index]}')
plt.show()
```



### 3. Data Load & Preprocessing

But no preprocessing today.

#### next function

iterate enumerator and returns (index, content) tuple

In this case, content is (data,label) pair, so unpacked

```
# Examine the data loader
train_enumerator = enumerate(train_loader)
ex_batch_idx, (ex_data, ex_label) = next(train_enumerator)
print(f'Idx: {ex_batch_idx} / X.shape = {ex_data.shape} / Y.shape = {ex_label.shape}\n')
print(f'Y[0:{batch_size}] = {ex_label}')

preview_index = 0
plt.figure(2)
plt.imshow(ex_data[preview_index, 0, :, :])
plt.title(f'Batch example data [{preview_index}], label={ex_label[preview_index]}'')
plt.show()
```

### 3. Data Load & Preprocessing

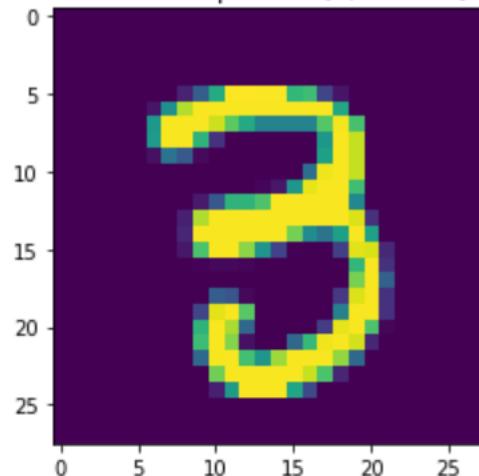
33



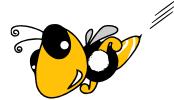
But no preprocessing today.

```
Idx: 0 / X.shape = torch.Size([32, 1, 28, 28]) / Y.shape = torch.Size([32])  
Y[0:32] = tensor([3, 1, 0, 7, 7, 0, 1, 4, 9, 2, 7, 4, 0, 1, 5, 9, 3, 6, 0, 6, 7, 6, 1, 2,  
3, 1, 0, 4, 1, 0, 8, 7])
```

Batch example data [0, label=3]



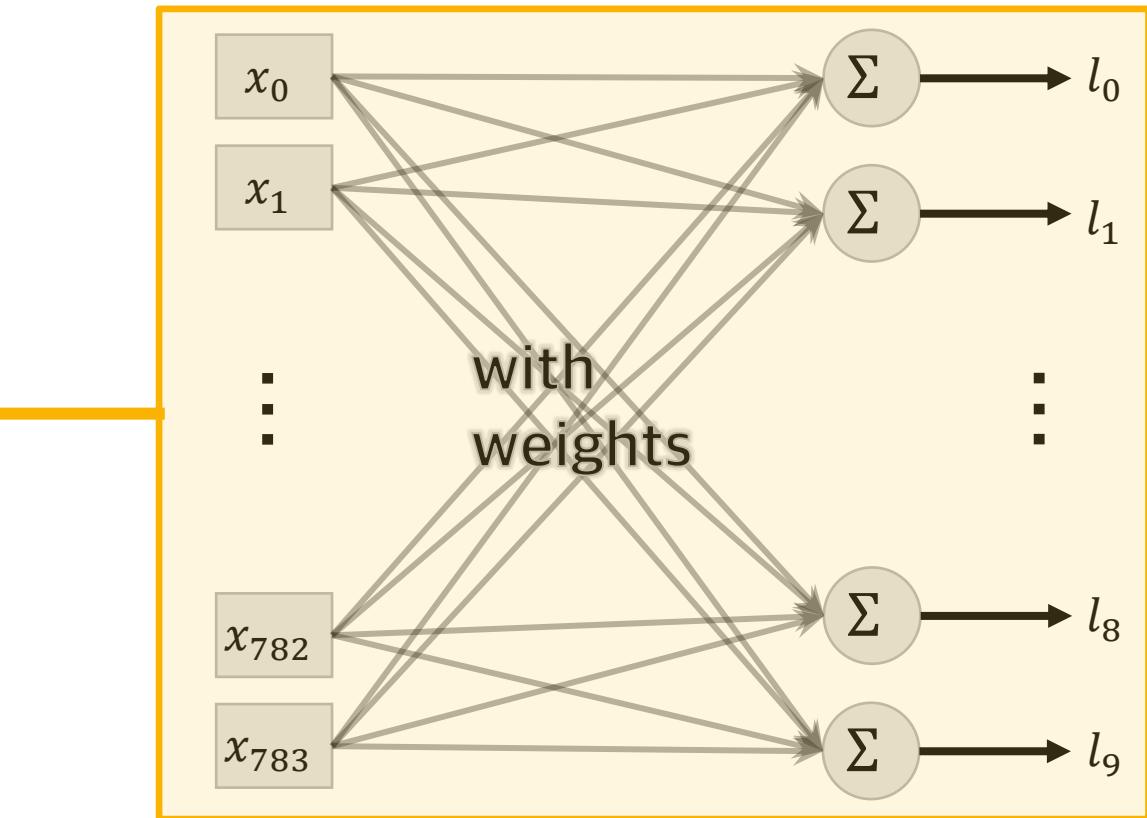
**Expected Output**  
Check label equals with  
the picture



## 4. Function Definitions

### Defining Model

```
# Model
def init_model():
    global net, loss_fn, optim
    net = nn.Linear(len(train_0_x.view([-1])), 10, bias=False).to(device)
    loss_fn = nn.CrossEntropyLoss()
    optim = SGD(net.parameters(), lr=learning_rate)
```



# FYI : <global> keyword in Python

35



```
- □ ×  
  
a = "저 조교 뭔소리하는거냐"  
  
def test():  
    print(a)  
  
test() # output: "저 조교 뭔소리하는거냐"
```

- Without **global**, a function can **read** variables declared outside (“global variables”)
- With **global**, a function now can **read & write** to global variables

```
- □ ×  
  
a = "저 조교 뭔소리하는거냐"  
  
def test():  
    a = "설명이 왜케 어려움?"  
  
test()  
print(a) # output: "저 조교 뭔소리하는거냐"
```

```
- □ ×  
  
a = "저 조교 뭔소리하는거냐"  
  
def test():  
    global a  
    a = "설명이 왜케 어려움?"  
  
test()  
print(a) # output: "설명이 왜케 어려움?"
```



## 4. Function Definitions

### Defining Epoch Functions

```
# Epoch
def init_epoch():
    global epoch_cnt
    epoch_cnt = 0
```

```
def epoch(data_loader):
    # One epoch : gets data_loader as input and returns loss / accuracy, and
    #               last prediction value / its label(truth) value for future use
    global epoch_cnt
    ### Put your script here ###
```

→ Next Page!

```
def epoch_not_finished():
    # For now, let's repeat training fixed times.
    # We will learn how to determine training stop or continue later.
    return epoch_cnt < maximum_epoch
```

## 4. Function Definitions

37



### Defining Epoch Functions

```
def epoch(data_loader):  
    # One epoch : gets data_loader as input and returns loss / accuracy, and  
    #               last prediction value / its label(truth) value for future use  
    global epoch_cnt  
  
    iter_loss, iter_acc = [], []  
  
    last_out, last_label = None, None  
    last_grad_performed = False  
  
    # Mini-batch iterations  
    ### Put your script here ### → Next Page!  
  
    # Up epoch count if backward propagation is done  
    if last_grad_performed:  
        epoch_cnt += 1  
  
    return np.average(iter_loss), np.average(iter_acc), last_out, last_label
```

## 4. Function Definitions

38



### Defining Epoch Functions

```
# Mini-batch iterations
for _data, _label in data_loader:
    data = _data.view([len(_data), -1]).to(device)
    label = _label.to(device)

# 1. Feed-forward

# 2. Calculate accuracy

# 3. Calculate loss

# 4. Backward propagation if not in `torch.no_grad()`

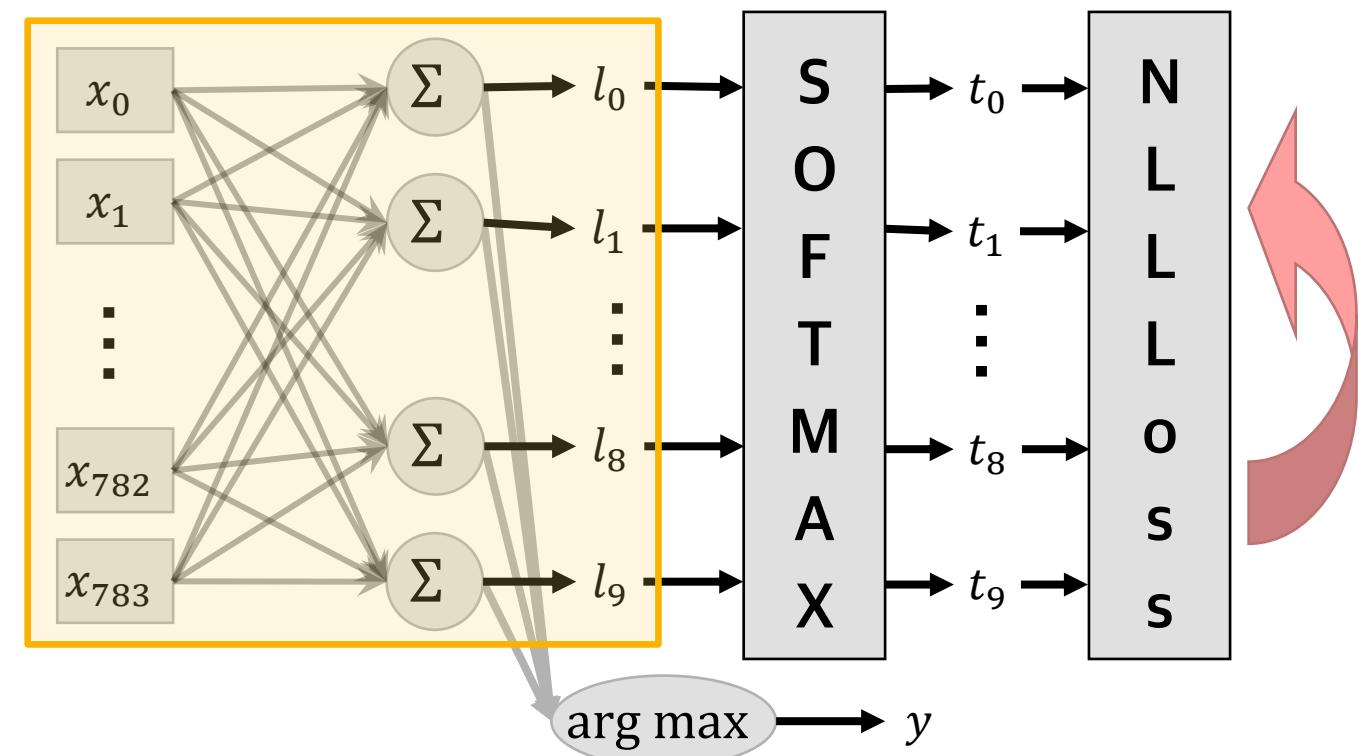
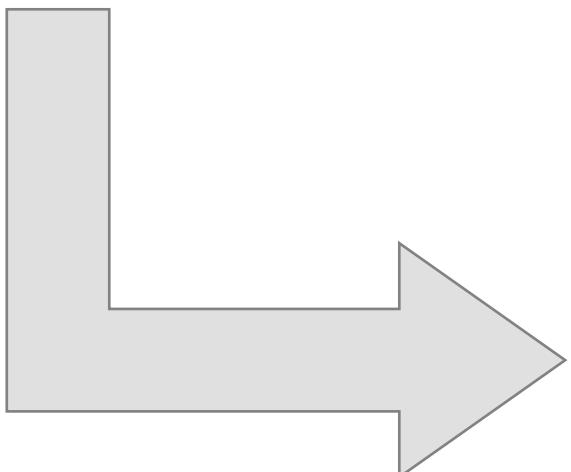
# 5. Save current iteration data for future use
```



## 4. Function Definitions

### Defining Epoch Functions

```
# 1. Feed-forward  
onehot_out = net(data)
```

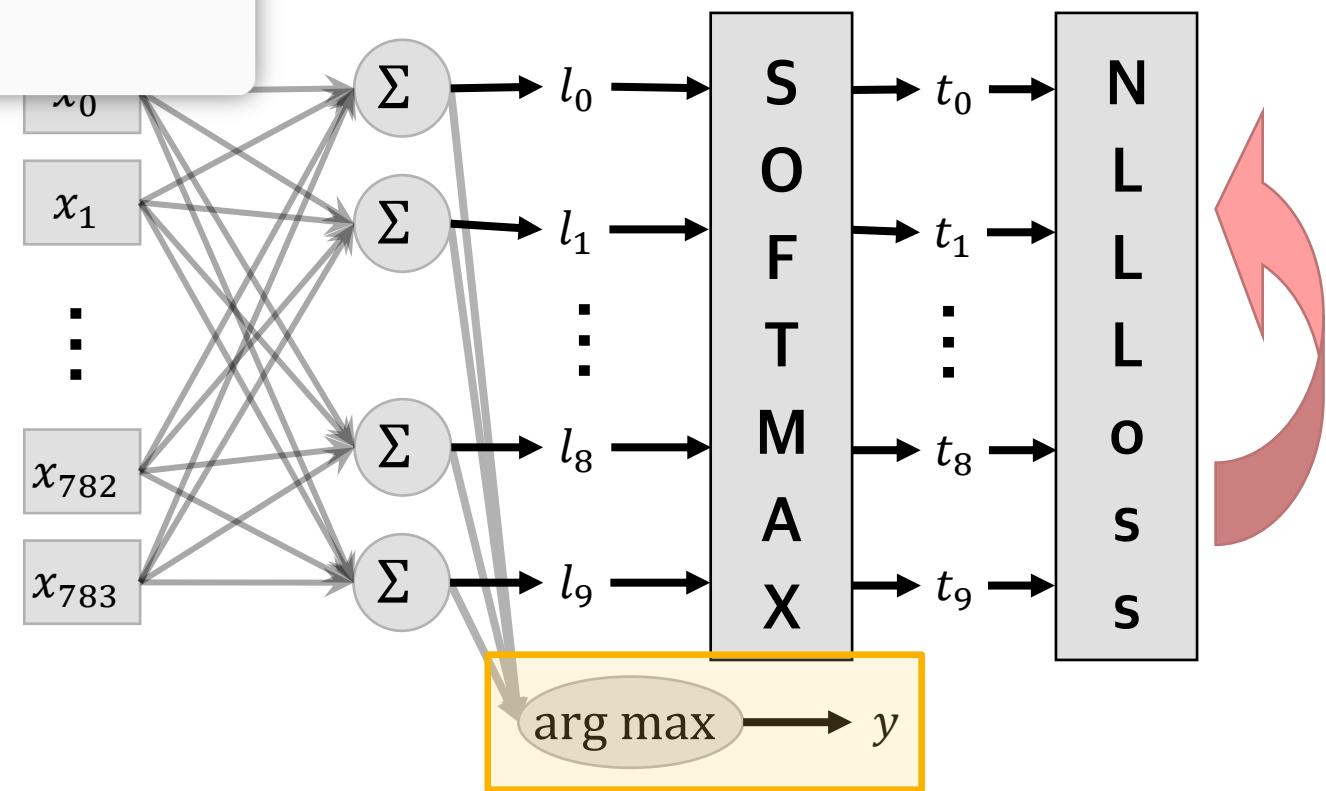




## 4. Function Definitions

### Defining Epoch Functions

```
# 2. Calculate accuracy
_, out = torch.max(onehot_out, 1)
acc_partial = (out == label).float().sum()
acc_partial = acc_partial / len(label)
iter_acc.append(acc_partial.item())
```





# 4. Function Definitions

## Defining Epoch Functions

# 2. Calculate accuracy

```
_ , out = torch.max(onehot_out, 1)
acc_partial = (out == label).float().sum()
acc_partial = acc_partial / len(label)
iter_acc.append(acc_partial.item())
```

```
tensor([[5.9561e-02, 1.4049e-01, -2.4387e-01, -1.4225e-02, -1.4735e-01,
        7.1124e-02, -3.3061e-01, 1.1523e-01, -1.1101e-01, -2.2883e-01,
        1.8862e-02, 2.0651e-01, 1.2741e-03, -1.2919e-01, -1.4365e-01,
        1.4316e-02, 1.0511e-01, 1.1521e-01, -1.1101e-01, -2.2883e-01,
        -8.4315e-02, -7.9726e-02, -1.3875e-02, 4.4802e-02, -2.3686e-01,
        1.1304e-02, -6.1174e-02, 4.3910e-01, 1.1872e-01, 5.7670e-02,
        -4.7760e-02, -9.4298e-02, 1.7634e-02, -4.6932e-02, -2.5942e-01,
        8.7709e-02, 1.0511e-01, 1.1521e-01, -1.1101e-01, -2.2883e-01,
        1.2589e-01, 3.4872e-02, 1.3241e-01, 8.7885e-02, -1.6547e-01,
        1.4645e-01, 1.5634e-01, 1.0699e-01, -9.4469e-02, 1.9469e-01,
        1.4645e-01, -1.6064e-01, 1.0699e-01, -9.4469e-02, 1.9469e-01,
        1.2313e-01, 3.3973e-01, 2.4114e-01, 1.3473e-01, 8.7709e-02,
        -2.5698e-02, -6.8084e-02, 2.8876e-02, -1.0732e-01, -6.5708e-02,
        4.3363e-03, 5.1862e-02, -3.1112e-02, 2.4195e-02, 1.3031e-02,
        -1.1371e-01, 1.6207e-01, 1.1521e-01, -1.1101e-01, -2.2883e-01,
        8.3504e-02, -9.8984e-02, 1.2381e-01, 1.2159e-01, -1.7988e-01,
        -9.9459e-02, 1.2832e-01, 4.0790e-02, -2.4794e-01, -6.4466e-02,
        -9.3873e-02, -1.5129e-01, 1.0394e-01, -3.4828e-02, -4.5244e-02,
        1.4316e-01, 1.0511e-01, 1.1521e-01, -1.1101e-01, -2.2883e-01,
        1.8326e-01, 4.4041e-02, 1.0642e-02, -6.6427e-02, 1.2892e-01,
        -4.3629e-01, 1.6380e-01, -6.2322e-02, -2.0559e-01, 1.6177e-02,
        -4.3629e-01, 1.6380e-01, 1.0642e-02, -6.6427e-02, 1.2892e-01,
        1.3515e-01, 6.1386e-02, -2.9883e-02, -4.8315e-01, -3.1993e-01,
        1.3051e-01, -1.8798e-01, 1.2552e-01, -2.8898e-01, -5.0095e-02,
        -2.1371e-01, 5.7424e-02, -8.5059e-02, -4.7407e-02, -1.2991e-01,
        1.7132e-01, 2.4207e-02, -2.1132e-02, -1.1572e-01, -5.8666e-02,
        -9.6651e-02, 2.6228e-01, -9.0688e-02, -1.1572e-01, -5.8666e-02,
        -2.5152e-01, -1.2891e-01, 4.5177e-02, -2.1318e-01, -1.1742e-01,
        -8.8120e-02, 1.6353e-01, 4.8992e-02, -1.8806e-01, -4.0199e-01,
        2.1023e-01, 1.0511e-01, 1.1521e-01, -1.1101e-01, -2.2883e-01,
        -8.9227e-03, 2.9377e-02, 1.8387e-03, -8.9865e-02, 7.6758e-02,
        -9.4384e-01, -1.9576e-01, 5.2755e-02, -7.1338e-02, -4.2444e-02,
        3.7933e-01, 1.1521e-01, 1.0642e-02, -6.6427e-02, 1.2892e-01,
        1.3323e-01, -1.5124e-01, 1.6146e-01, 6.2269e-02, 2.3689e-01,
        -7.4834e-02, -1.4469e-02, -5.6898e-02, -2.0288e-01, 6.1463e-03,
        -2.5782e-01, 1.6207e-02, -2.7937e-02, -1.9385e-01, 5.0438e-02,
        1.1371e-01, 1.6207e-01, 1.1521e-01, -1.1101e-01, -2.2883e-01,
        -9.3144e-02, -2.3212e-01, 2.1658e-02, -1.7821e-01, -1.1547e-01,
        -8.3014e-02, 1.2362e-01, 3.6289e-01, -2.8786e-01, -1.2853e-01,
        -8.1553e-02, -1.3612e-01, 4.3755e-01, 2.3778e-01, -1.2078e-01,
        -1.8398e-01, 1.6484e-01, 2.9927e-01, 1.3718e-01, 9.7356e-02,
        -1.2013e-01, 2.2959e-01, 2.6733e-01, -4.6533e-01, -4.3897e-02,
        -3.4291e-01, 1.1521e-01, 1.0642e-02, -6.6427e-02, 1.2892e-01,
        -1.2801e-01, 1.2479e-01, 1.9916e-01, 2.0881e-01, -1.5256e-01,
        6.0577e-02, -2.6663e-02, 6.2844e-02, -8.4287e-02, -2.4466e-02,
        -2.0750e-01, 2.1452e-01, 1.5889e-01, -4.8112e-01, -1.1087e-02,
        -1.3171e-01, 1.6207e-01, 1.1521e-01, -1.1101e-01, -2.2883e-01,
        -4.9952e-02, 6.4613e-02, 7.6580e-02, -5.3031e-01, 2.8446e-01,
        1.1840e-01, 1.2143e-01, 1.7759e-01, 6.6608e-02, 4.4754e-02,
        -6.6427e-01, -1.8784e-01, 2.0720e-01, -1.8769e-01, 8.8377e-02,
        1.1808e-01, 1.8784e-01, 2.0720e-01, -1.8769e-01, 8.8377e-02,
        -5.3103e-02, -8.2765e-02, -1.4820e-02, 1.3658e-01, -1.6422e-01,
        2.6071e-01, -1.8729e-01, 7.3351e-02, -6.5008e-02, -4.1201e-01,
        4.4754e-02, -1.8729e-01, 7.3351e-02, -6.5008e-02, -4.1201e-01,
        -1.2921e-01, -4.2055e-01, 2.1417e-01, -8.0963e-01, -3.0731e-02,
        -2.8691e-01, 2.3498e-01, -6.7557e-02, -7.9039e-02, -2.5454e-01,
        -8.2559e-02, 2.4752e-02, 2.2112e-01, -1.0606e-01, -3.0731e-02,
        -1.3171e-01, 1.6207e-01, 1.1521e-01, -1.1101e-01, -2.2883e-01,
        -1.0977e-01, 1.9554e-02, 4.0878e-01, 9.5662e-03, -2.1867e-01,
        -2.2786e-02, -2.0964e-02, 1.6962e-01, 5.2866e-02, 2.2721e-02,
        -1.8398e-01, 1.6484e-01, 2.9927e-01, -1.3718e-01, -9.7356e-02,
        -1.4958e-01, 2.6663e-01, 1.8875e-01, -3.7581e-01, 2.4996e-01,
        1.7620e-01, -4.3163e-01, -9.9159e-02, -2.5398e-01, 9.8225e-02]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

onehot\_out.shape = batch\_size\*10  
(score for each classes, each data)

arg max in “classes” dimension

1

Compare &  
Calculate Correct  
Answer Ratio  
(Accuracy)

Real Label

2

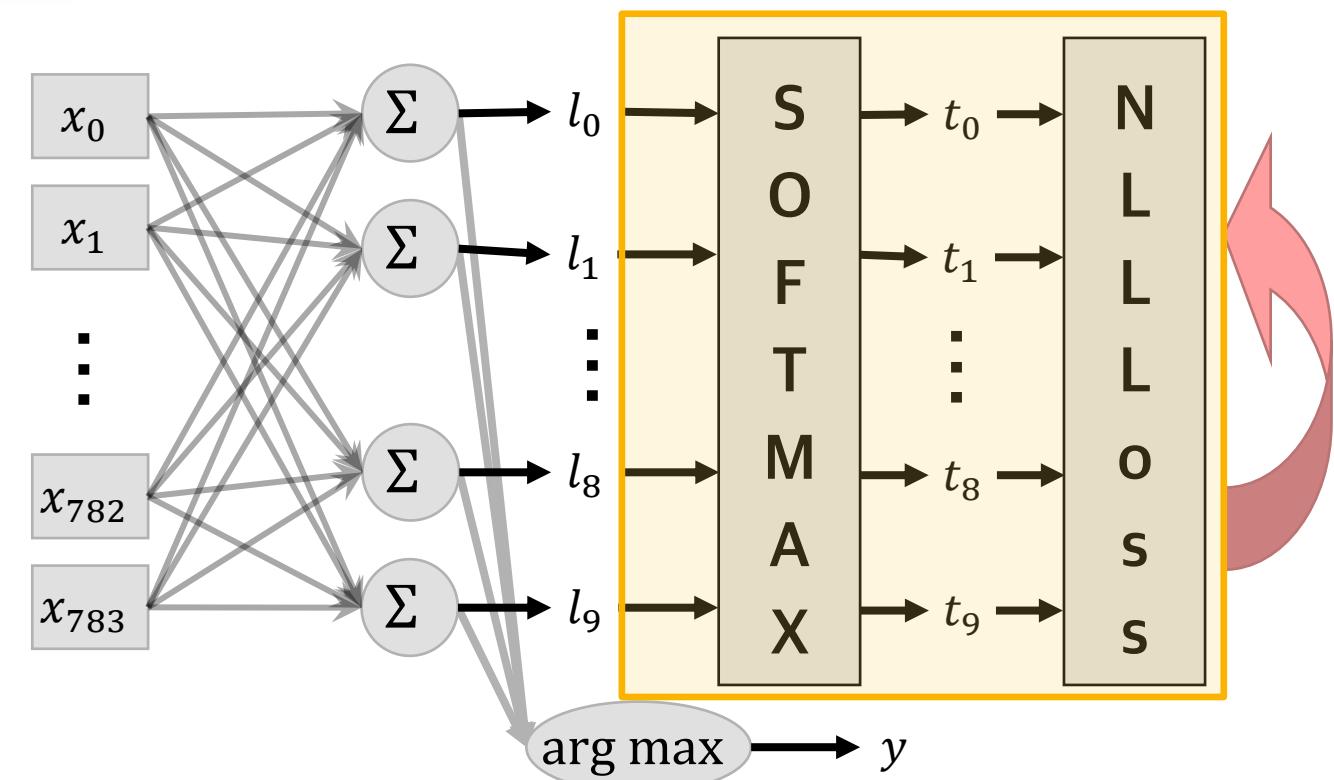
out.shape = batch\_size\*1  
(predicted label for each data)



## 4. Function Definitions

### Defining Epoch Functions

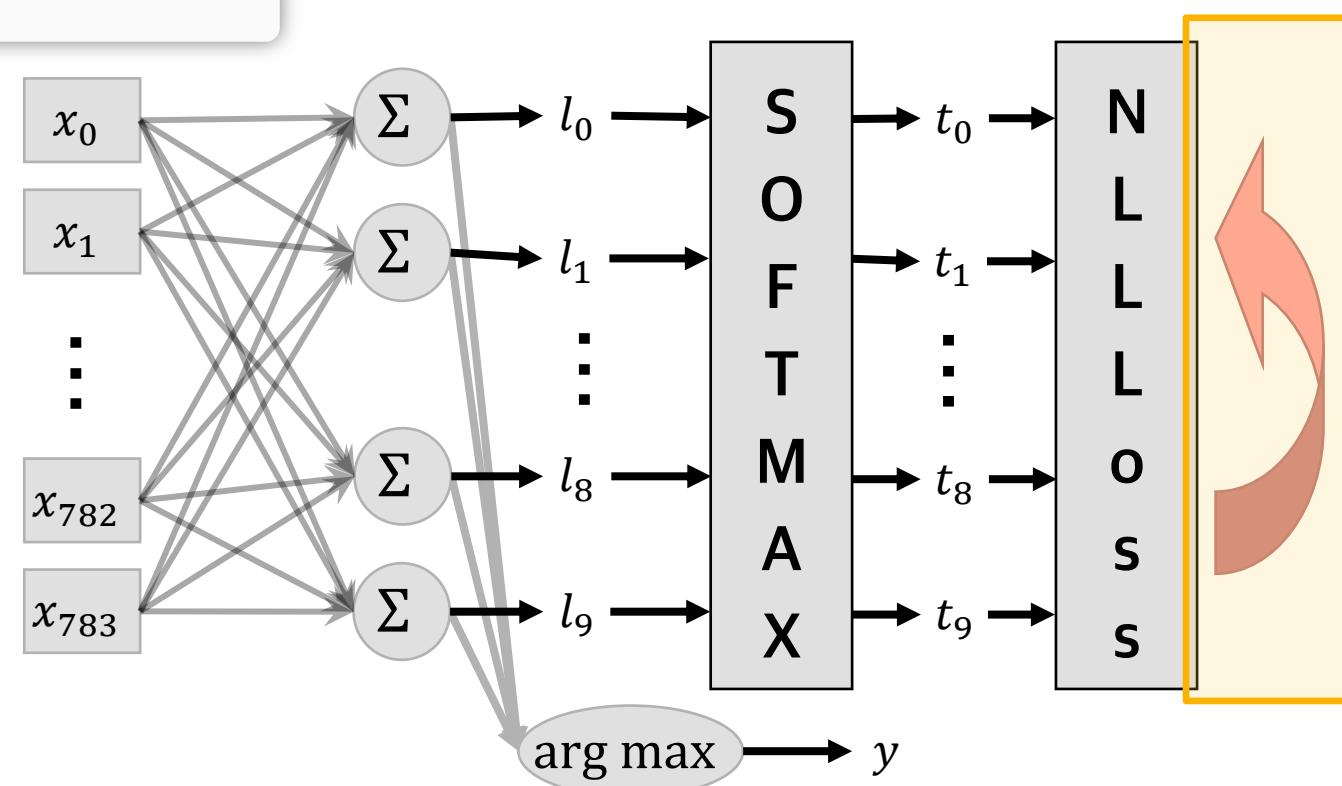
```
# 3. Calculate loss
loss = loss_fn(onehot_out, label)
iter_loss.append(loss.item())
```



## 4. Function Definitions

### Defining Epoch Functions

```
# 4. Backward propagation if not in `torch.no_grad()`
if onehot_out.requires_grad:
    optim.zero_grad()
    loss.backward()
    optim.step()
    last_grad_performed = True
```





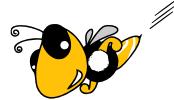
## 4. Function Definitions

### Defining Epoch Functions

```
# 5. Save current iteration data for future use  
last_out = out.cpu().detach()  
last_label = _label
```

<https://laf.gg/aibd-epoch>

If you finished, compare your code with above!



## 4. Function Definitions

### Defining Logging Functions

```
# Logging
def init_log():
    global log_stack, iter_log, tloss_log, tacc_log, vloss_log, vacc_log, time_log
    iter_log, tloss_log, tacc_log, vloss_log, vacc_log = [], [], [], [], []
    time_log, log_stack = [], []

def record_train_log(_iter, _tacc, _time):
    # Push time, training accuracy, and iteration count into lists
    ### Put your script here ###

def record_valid_log(_vloss,
                     _vacc):
    # Push validation loss and validation accuracy into each list
    ### Put your script here ###

def last(log_list):
    # Get the last member of the list
    if len(log_list) > 0: return log_list[-1]
    else: return -1

def print_log():
    # Generate log message and put it into log stack
    ### Put your script here ###

    # Draw figure if want
    if logging_disfig:
        ### Put your script here ###

    # Print log
    ### Put your script here ###
```



## 4. Function Definitions

46



### Defining Logging Functions

```
def init_log():
    global log_stack, iter_log, tloss_log, tacc_log, vloss_log, vacc_log, time_log
    iter_log, tloss_log, tacc_log, vloss_log, vacc_log = [], [], [], [], []
    time_log, log_stack = [], []

def record_train_log(_tloss, _tacc, _time):
    # Push time, training loss, training accuracy, and epoch count into lists
    time_log.append(_time)
    tloss_log.append(_tloss)
    tacc_log.append(_tacc)
    iter_log.append(epoch_cnt)

def record_valid_log(_vloss, _vacc):
    # Push validation loss and validation accuracy into each list
    vloss_log.append(_vloss)
    vacc_log.append(_vacc)
```



## 4. Function Definitions

### Defining Logging Functions

```

def print_log():
    # Generate log string and put it into log stack
    log_str = f'Iter: {last(iter_log):>4d} >> T_loss {last(tloss_log):<8.5f}   '\ \
              + f'T_acc {last(tacc_log):<6.5f}   V_loss {last(vloss_log):<8.5f}   '\ \
              + f'V_acc {last(vacc_log):<6.5f}   ⏺ {last(time_log):5.3f}s'
    log_stack.append(log_str)

    # Draw figure if want
    if logging_dispfig:
        hist_fig, loss_axis = plt.subplots(figsize=(10, 3), dpi=99)
        hist_fig.patch.set_facecolor('white')

        # Draw loss lines
        loss_t_line = plt.plot(iter_log, tloss_log, label='Train Loss', color='#FF9999', marker='o')
        loss_v_line = plt.plot(iter_log, vloss_log, label='Valid Loss', color='#99B0FF', marker='s')
        loss_axis.set_xlabel('epoch')
        loss_axis.set_ylabel('loss')

        # Draw accuracy lines
        acc_axis = loss_axis.twinx()
        acc_t_line = acc_axis.plot(iter_log, tacc_log, label='Train Acc.', color='#FF0000', marker='+')
        acc_v_line = acc_axis.plot(iter_log, vacc_log, label='Valid Acc.', color='#003AFF', marker='x')
        acc_axis.set_ylabel('accuracy')

        # Append annotations
        hist_lines = loss_t_line + loss_v_line + acc_t_line + acc_v_line
        loss_axis.legend(hist_lines, [l.get_label() for l in hist_lines])
        loss_axis.grid()
        plt.title(f'Learning history until epoch {last(iter_log)}')
        plt.draw()

    # Print log
    clear_output(wait=True)
    if logging_dispfig: plt.show()
    for idx in reversed(range(len(log_stack))):
        print(log_stack[idx])

```

## 5. Training Iteration 48



```
# Training Initialization
init_model()
init_epoch()
init_log()

while epoch_not_finished():
    start_time = time.time()
    tloss, tacc, _, _ = epoch(train_loader)
    end_time = time.time()
    time_taken = end_time - start_time
    record_train_log(tloss, tacc, time_taken)
    with torch.no_grad():
        vloss, vacc, _, _ = epoch(valid_loader)
        record_valid_log(vloss, vacc)
    print_log()

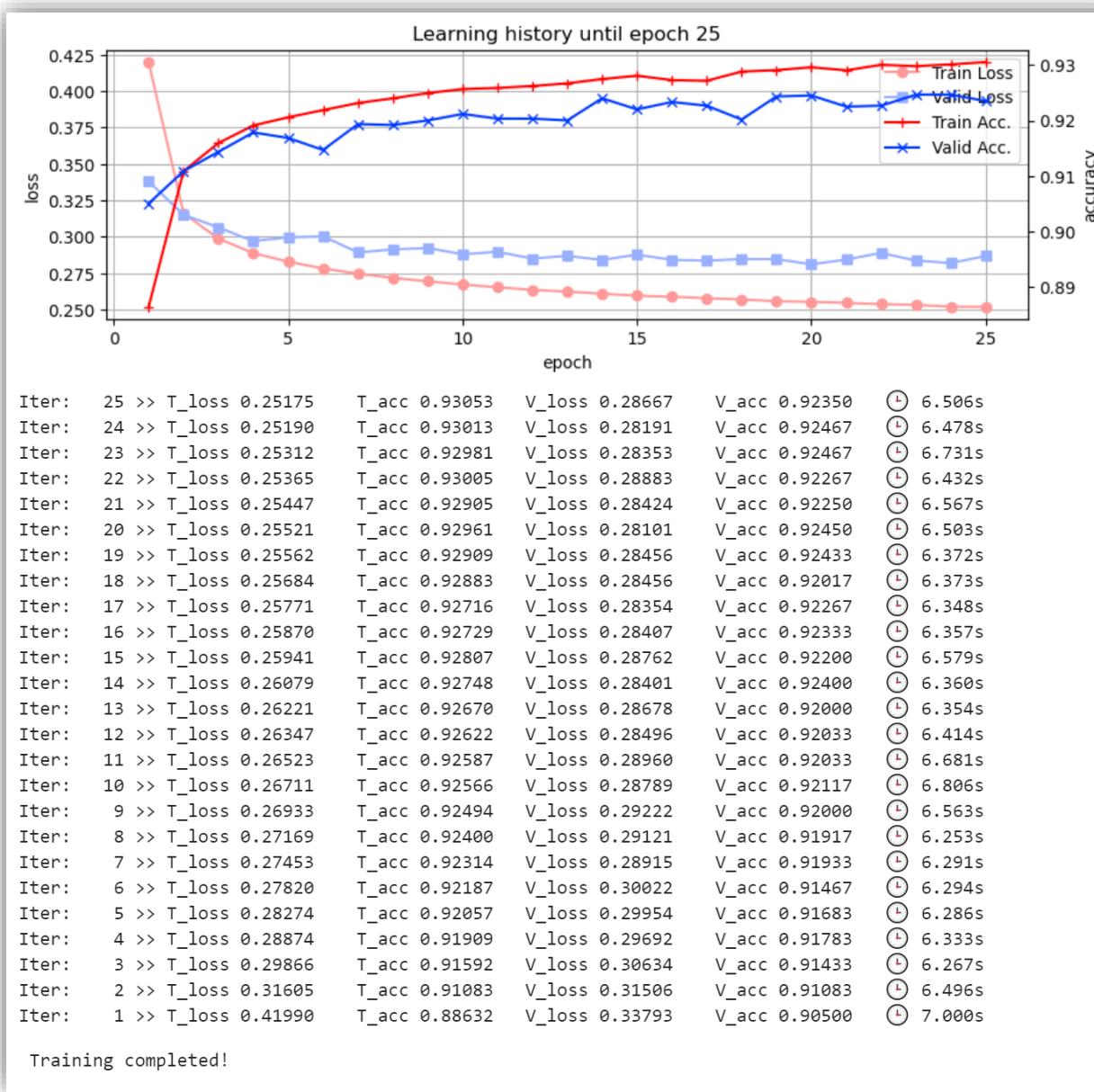
print('\n Training completed!')
```

### **with torch.no\_grad():**

PyTorch code in this section will be operated with autograd to be turned off. So, gradient is not calculated.

- No gradient calculation is needed for validation & test set
- In this section,  
**net(data).requires\_grad** will be **False**

# 5. Training Iteration 49



## Expected Output

# 6. Result Analysis

50



## Accuracy Test

- □ ×

```
# Accuracy for test dataset
with torch.no_grad():
    test_loss, test_acc, test_out, test_label = epoch(test_loader)
    print(f'Test accuracy = {test_acc}\nTest loss = {test_loss}')
```

Test accuracy = 0.92249995470047  
Test loss = 0.2745736539363861

Expected Output

# 6. Result Analysis

51



## Confusion Matrix

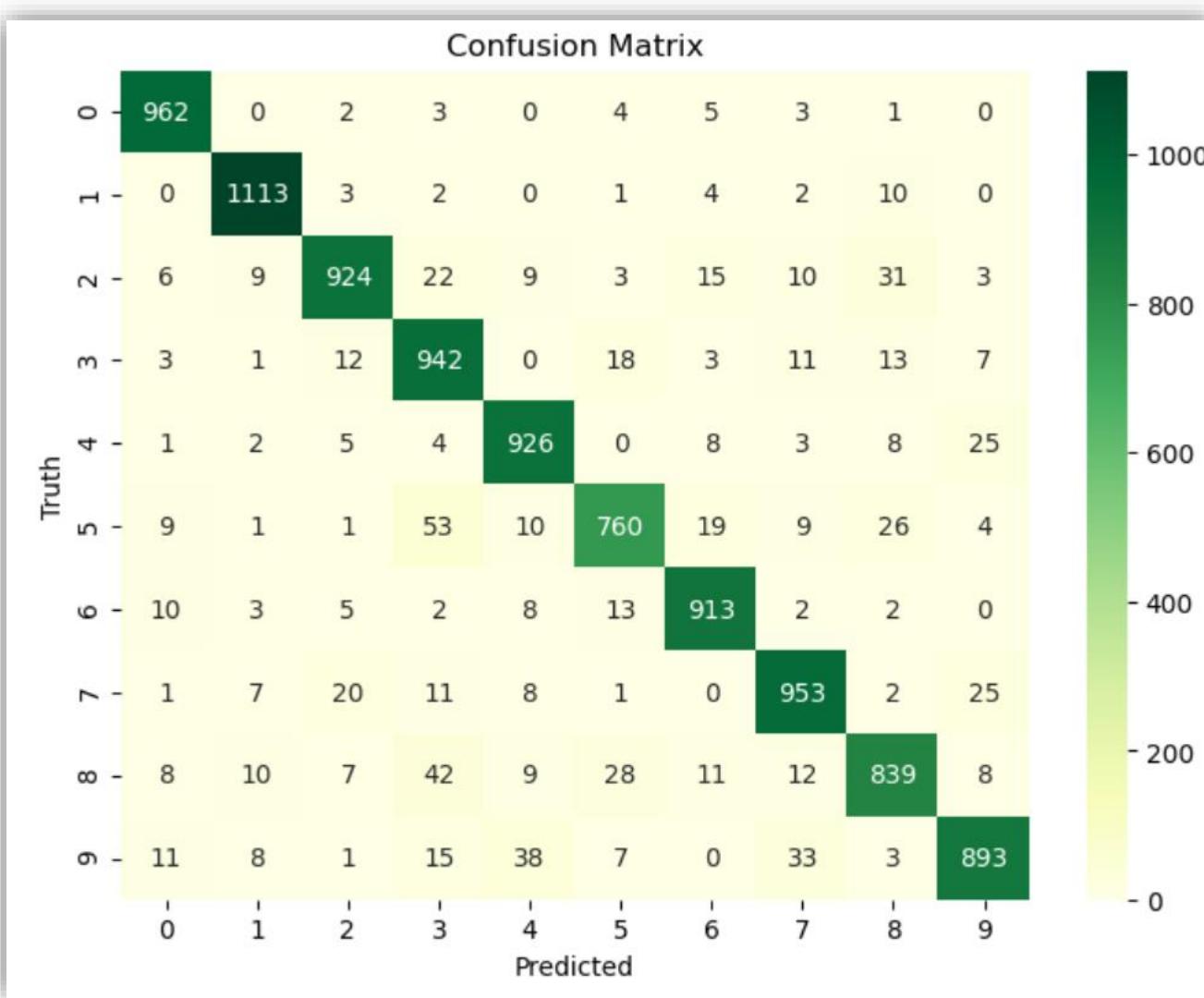
```
# Confusion matrix
our_cmatrix = confusion_matrix(test_label, test_out)
plt.figure(figsize=(8, 6), dpi=99)
sns.heatmap(our_cmatrix, annot=True, fmt='g', cmap='YlGn').set(xlabel='Predicted', ylabel='Truth')
plt.title('Confusion Matrix')
plt.show()
```

# 6. Result Analysis

52



## Confusion Matrix



## Expected Output



— □ ×

*# To save*

```
torch.save(net.state_dict(), './model.pkl')
```

*# To load*

```
net = torch.load('./model.pkl')
```

# Weight Initialization

How weight initialization impact the performance?

# Let's add some layers ...

57

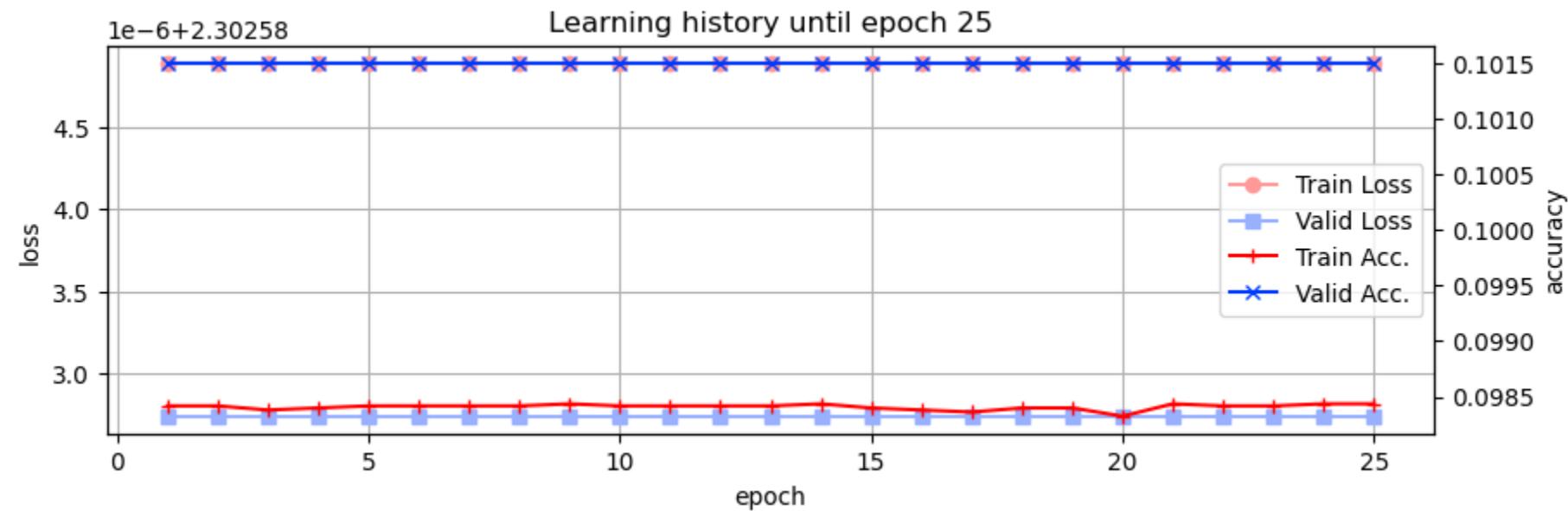


```
- □ ×  
  
# Data Loader  
batch_size = 32  
  
# Model  
hidden_layer = 200  
  
# Learning  
logging_dispfig = True  
maximum_epoch = 25  
learning_rate = 0.1
```

```
- □ ×  
  
# Model  
def init_model():  
    global net, loss_fn, optim  
    net = nn.Sequential(  
        nn.Linear(len(train_0_x.view([-1])), hidden_layer, bias=False),  
        nn.ReLU(),  
        nn.Linear(hidden_layer, 10, bias=False)  
    ).to(device)  
    loss_fn = nn.CrossEntropyLoss()  
    optim = SGD(net.parameters(), lr=learning_rate)  
    nn.init.zeros_(net[0].weight)  
    nn.init.zeros_(net[2].weight)
```



## What's the result?





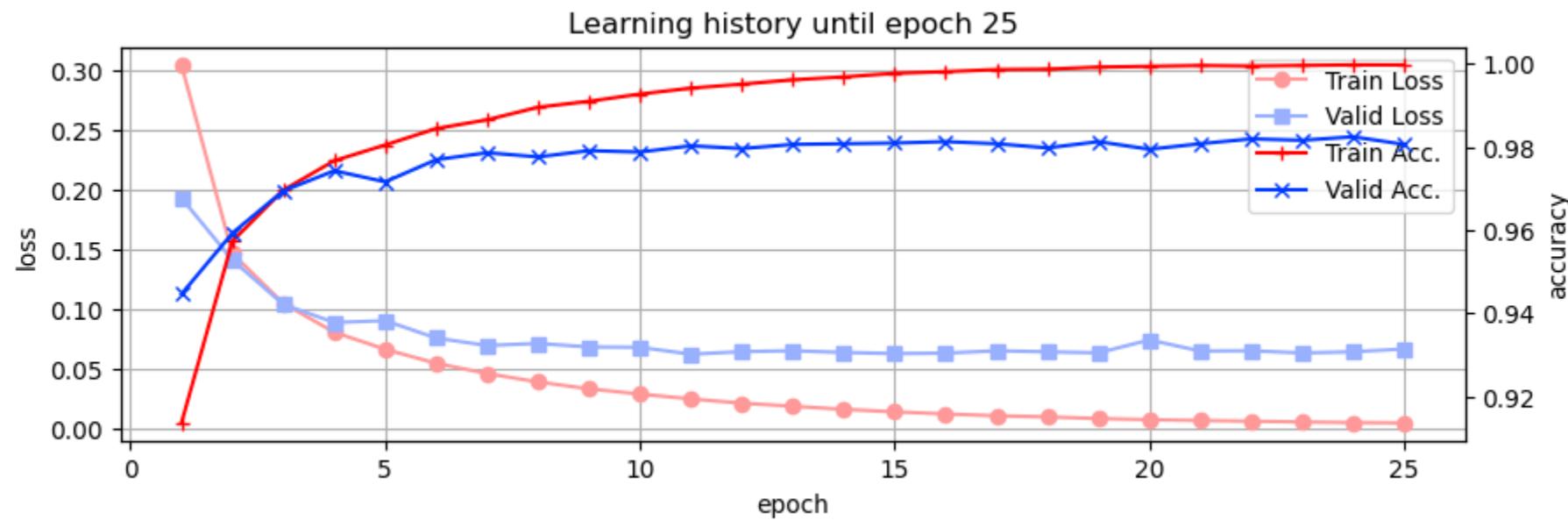
```
# Model
def init_model():
    global net, loss_fn, optim
    net = nn.Sequential(
        nn.Linear(len(train_0_x.view([-1])), hidden_layer, bias=False),
        nn.ReLU(),
        nn.Linear(hidden_layer, 10, bias=False)
    ).to(device)
    loss_fn = nn.CrossEntropyLoss()
    optim = SGD(net.parameters(), lr=learning_rate)
    nn.init.xavier_normal_(net[0].weight)
    nn.init.xavier_normal_(net[2].weight)
```

# And Try Training Again.

60

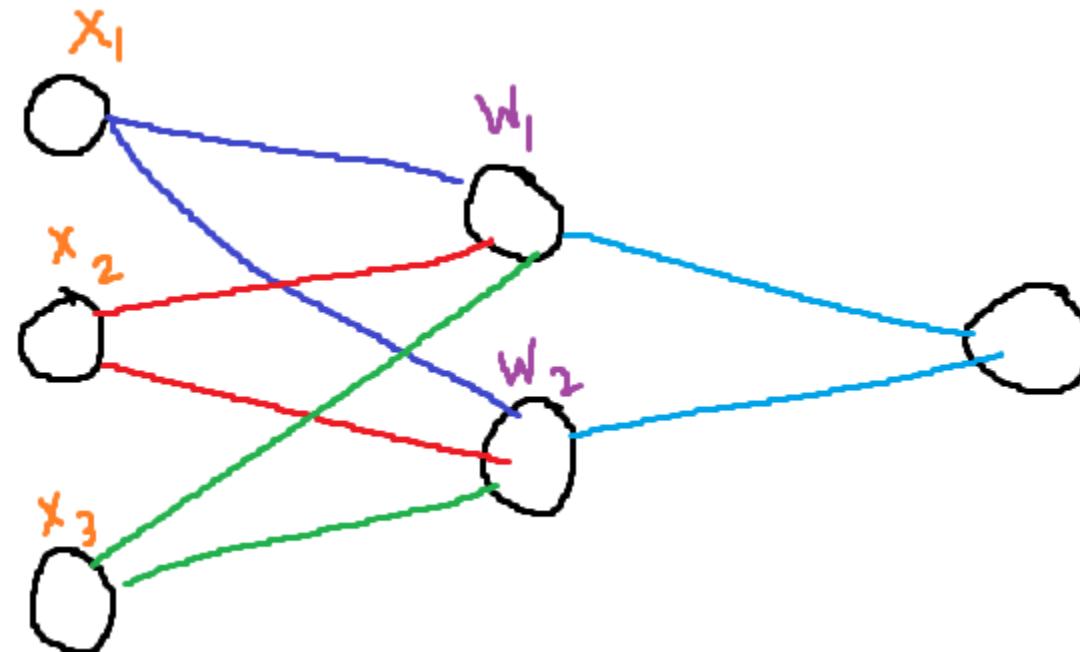


## What's the result?





When a model is initialized with zero, they will have same gradient!



So if you want your model work properly, don't touch to initial value or don't set it zero!

# Finding Proper Hyper-parameters

Is there any way to find best hyper-parameters?