March 24th, 2021

AI+BD ML Lab. Day 3

MLP & Regularization (Batch Norm. / Dropout)

YoungIn Kim
<youngkim21@postech.edu>

MoNet Laboratory

Contents

Preparation

Base .ipynb :

```
https://git.io/aibd-mlp-3
```

Reminding

1. Library Importation
   & Device Preparation

```
# Library Importation
import matplotlib.pyplot as plt
import numpy as np
import time
import torch
import torch.nn as nn
import torch.nn.functional as F

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

# Device Preparation
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.')
```

Reminding

2. Hyper-Parameters

```
# MNIST dataset
data_path = "../data"

# Data Loader
batch_size = 32

# Model
hidden_layer = 200

# Learning
logging_dispfig = True
maximum_epoch = 25
learning_rate = 0.1
```

change it to
your own path

Reminding

3. Data Load & Preprocessing

```python
# 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)

# Check the data
print('==================== Check the data ========================₩n')
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()

# 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)

# Examine the data loader
print('================== Check the data loader =====================₩n')
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()
```

Same as before

Reminding

4. Function Definitions

```
# Model
def init_model(_net):
    global net, loss_fn, optim
    net = _net.to(device)
    loss_fn = nn.CrossEntropyLoss()
    optim = SGD(net.parameters(), lr=learning_rate)
```

Reminding

4. Function Definitions (continue)

```python
# 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

    ...

def epoch_not_finished():
    # For now, let's repeat training fixed times, e.g. 25 times.
    # We will learn how to determine training stop or continue later.
    return epoch_cnt < maximum_epoch
```

Same as before

Reminding

4. Function Definitions (continue)

IMPORTANT!

```
# 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

    ...

def epoch_not_finished():
    # For now, let's repeat training fixed times, e.g. 25 times.
    # We will learn how to determine training stop or continue later.
    return epoch_cnt < maximum_epoch
```

Reminding

4. Function Definitions
   (continue)

Same as before
BUT try it by yourself.
Put your Script on the blank

```
# Mini-batch iterations
for _data, _label in data_loader:
    data, label = ### Put Your Script Here ###

    # 1. Feed-forward
    onehot_out = ### Put Your Script Here ###

    # 2. Calculate accuracy
    _, out = ### Put Your Script Here ###
    acc_partial = ### Put Your Script Here ###
    acc_partial = acc_partial / len(label)
    iter_acc.append(acc_partial.item())

    # 3. Calculate loss
    loss = ### Put Your Script Here ###
    iter_loss.append(loss.item())

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

        ### Put Your Script Here ###

        last_grad_performed = True

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

Reminding

4. Function Definitions (continue)

```python
# Mini-batch iterations
for _data, _label in data_loader:
    data, label = ### Put Your Script Here ###

    # 1. Feed-forward
    onehot_out = ### Put Your Script Here ###

    # 2. Calculate accuracy
    _, out = ### Put Your Script Here ###
    acc_partial = ### Put Your Script Here ###
    acc_partial = acc_partial / len(label)
    iter_acc.append(acc_partial.item())
```

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

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

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

Reminding

4. Function Definitions (continue)

```python
# Mini-batch iterations
for _data, _label in data_loader:
    data, label = ### Put Your Script Here ###

    # 1. Feed-forward
    onehot_out = ### Put Your Script Here ###

    # 2. Calculate accuracy
    _, out = ### Put Your Script Here ###
```

```python
# 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())

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

Reminding

4. Function Definitions (continue)

```
# Mini-batch iterations
for _data, _label in data_loader:
    data, label = ### Put Your Script Here ###

    # 1. Feed-forward
    onehot_out = ### Put Your Script Here ###

    # 2. Calculate accuracy
    _, out = ### Put Your Script Here ###
    acc_partial = ### Put Your Script Here ###
    acc_partial = acc_partial / len(label)
    iter_acc.append(acc_partial.item())
```

```
    last_label = _label
```

```
# 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
```

Reminding

4. Function Definitions (continue)

```
# 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(_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)


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

Same as before

MLP

Multi Layer Perceptron with nn.Module

Make model with nn.Module class

5. Model Architectures
   (before)

```
# before

net = nn.Linear(784, 10)

net = nn.Sequential(
    nn.Linear(len(train_0_x.view([-1])), hidden_layer, bias=False),
    nn.ReLU(),
    nn.Linear(hidden_layer, 10, bias=False)
)
```

How we implement model more fancy way?

Make model with nn.Module class

5. Model Architectures (before)

```
# before

net = nn.Linear(784, 10)

net = nn.Sequential(
    nn.Linear(len(train_0_x.view([-1])), hidden_layer, bias=False),
    nn.ReLU(),
    nn.Linear(hidden_layer, 10, bias=False)
)
```

How we implement model more fancy way?

`use Python ‘Class’ with nn.Module!`

Make model with nn.Module class

5. Model Architectures
   (after)

```
net = nn.Linear(784, 10)
```

```
class LinearModel(nn.Module): # inherit nn.Module
    def __init__(self): # initailize instance
        super(LinearModel, self).__init__()
        self.linear = nn.Linear(784, 10)

    def forward(self, x): # you must implement "forward" method
        return self.linear(x)

model = LinearModel()
```

Make model with nn.Module class

5. Model Architectures (after)

```
net = nn.Linear(784, 10)
```

```
class LinearModel(nn.Module): # inherit nn.Module
    def __init__(self): # initailize instance
        super(LinearModel, self).__init__()
        self.linear = nn.Linear(784, 10)

    def forward(self, x): # you must implement "forward" method
        return self.linear(x)

model = LinearModel()
```

IMPORTANT!

Make model with nn.Module class

5. Model Architectures (after)

```
# before
net = nn.Sequential(
    nn.Linear(len(train_0_x.view([-1])), hidden_layer, bias=False),
    nn.ReLU(),
    nn.Linear(hidden_layer, 10, bias=False)
)
```

```
# after
class MLP1():

    ### Put Your Script Here ###
```

Make model with nn.Module class

5. Model Architectures (after)

```python
# before
net = nn.Sequential(
    nn.Linear(len(train_0_x.view([-1])), hidden_layer, bias=False),
    nn.ReLU(),
    nn.Linear(hidden_layer, 10, bias=False)
)
```

```python
#after
class MLP1(nn.Module):
    def __init__(self):
        super(MLP1, self).__init__()

        self.fc1 = nn.Linear(len(train_0_x.view([-1])), hidden_layer, bias=False)
        self.act = nn.ReLU()
        self.fc2 = nn.Linear(hidden_layer, 10, bias=False)

    def forward(self, x):
        out = self.fc1(x)
        hidden = self.act(out)
        onehot_out = self.fc2(hidden)

        return onehot_out
```

Make model with nn.Module class

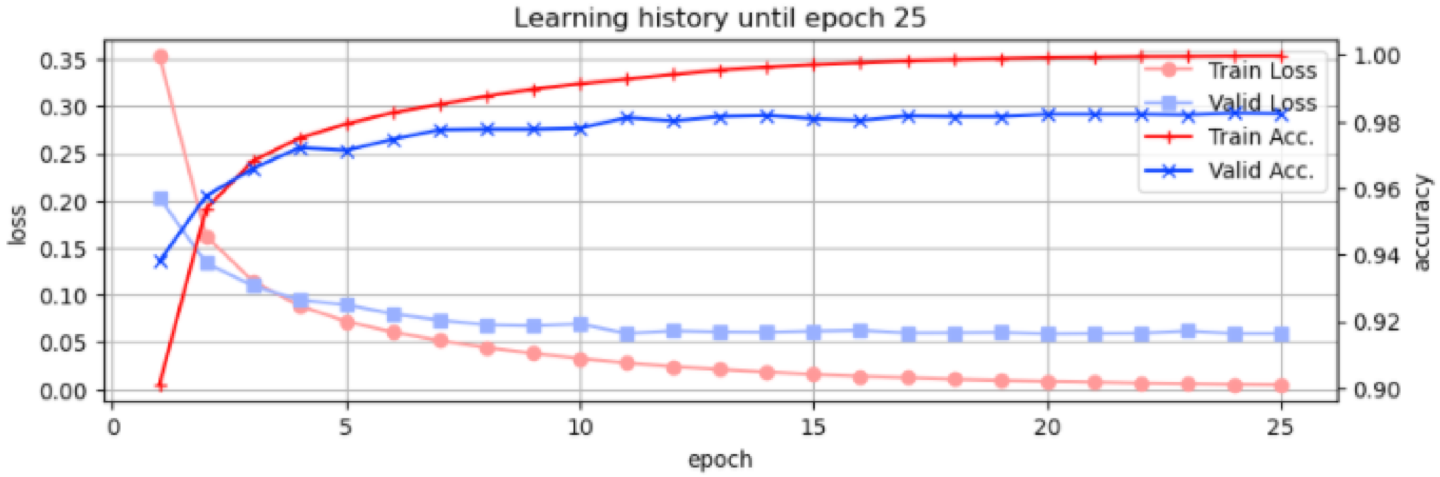
6. Training Iteration
   & Test Result

```python
# Training Initialization
init_model(MLP1())
init_epoch()
init_log()

# Training Iteration
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!')

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

```python
# Model
def init_model(net):
    global net, loss_fn, optim
    net = net.to(device)
    loss_fn = nn.CrossEntropyLoss()
    optim = SGD(net.parameters(), lr=learning_rate)
```

Make model with nn.Module class

6. Training Iteration
   & Test Result



Iter: 25 >> T_loss 0.00487   T_acc 0.99972   V_loss 0.05918   V_acc 0.98233   3.036s

```
==================== Test Result ====================

Test accuracy = 0.9801999926567078
Test loss = 0.06666535884141922
```

Test Acc. : 98.02 %

Make more Deep!

6-1. Training Iteration & Test Result (MLP2)

```
class MLP2(nn.Module):
    def __init__(self, in_features, out_features):
        super(MLP2, self).__init__()

        self.hidden_layer1 = 165
        self.hidden_layer2 = 165

        ### Put Your Script Here ###

    def forward(self, x):

        ### Put Your Script Here ###

        return onehot_out
```

Make more Deep!

6-1. Training Iteration & Test Result (MLP2)

```python
import torch.nn as nn
import torch.nn.functional as F
```

```python
class MLP2(nn.Module):
    def __init__(self, in_features, out_features):
        super(MLP2, self).__init__()

        self.hidden_layer1 = 165
        self.hidden_layer2 = 165

        self.fc1 = nn.Linear(in_features, self.hidden_layer1)
        self.act1 = nn.ReLU()
        self.fc2 = nn.Linear(self.hidden_layer1, self.hidden_layer2)
        self.act2 = nn.ReLU()
        self.fc3 = nn.Linear(self.hidden_layer2, out_features)

    def forward(self, x):
        hidden1 = self.act1(self.fc1(x))  ## self.act nn.ReLU() -> instance of class
        hidden2 = F.relu(self.fc2(hidden1))  ## F.relu -> function

        onehot_out = self.fc3(hidden2)

        return onehot_out
```

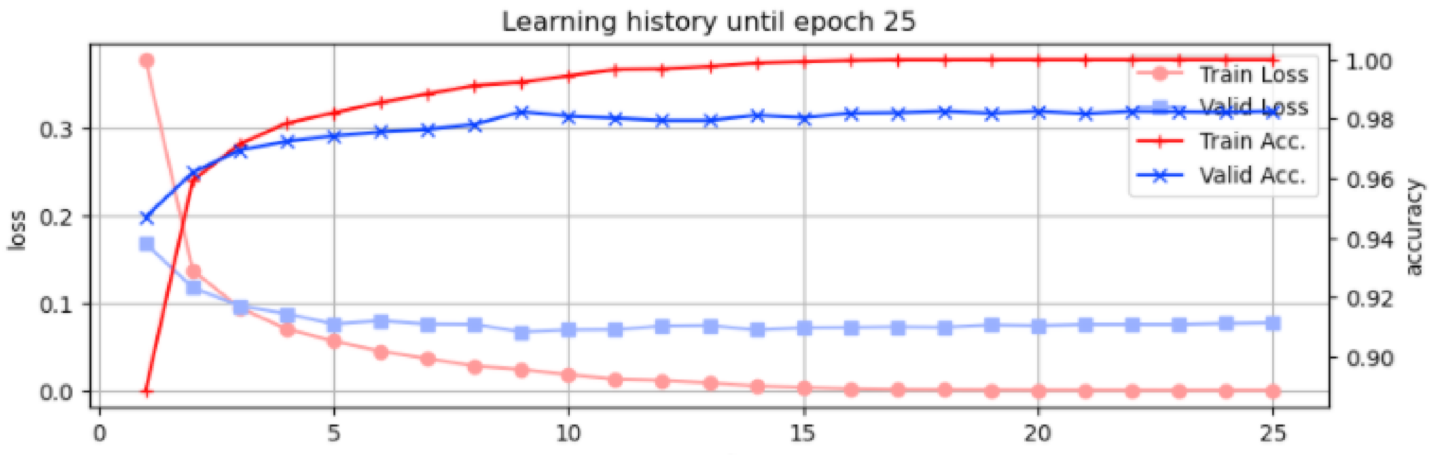
```python
# Training Initialization
init_model(MLP2(len(train_0_x.view([-1])), 10))
init_epoch()
init_log()
```

Make more Deep!

6-1. Training Iteration & Test Result (MLP2)

```
Test Acc. : 98.02 %
```

```
Test Acc. : 98.26 %
```




```
Iter:   25 >> T_loss 0.00047   T_acc 1.00000  V_loss 0.07750   V_acc 0.98267   3.298s

Test accuracy = 0.9826999902725220
Test loss = 0.07591626048088074
```

Reguralization

Batch Normalization & Drop out

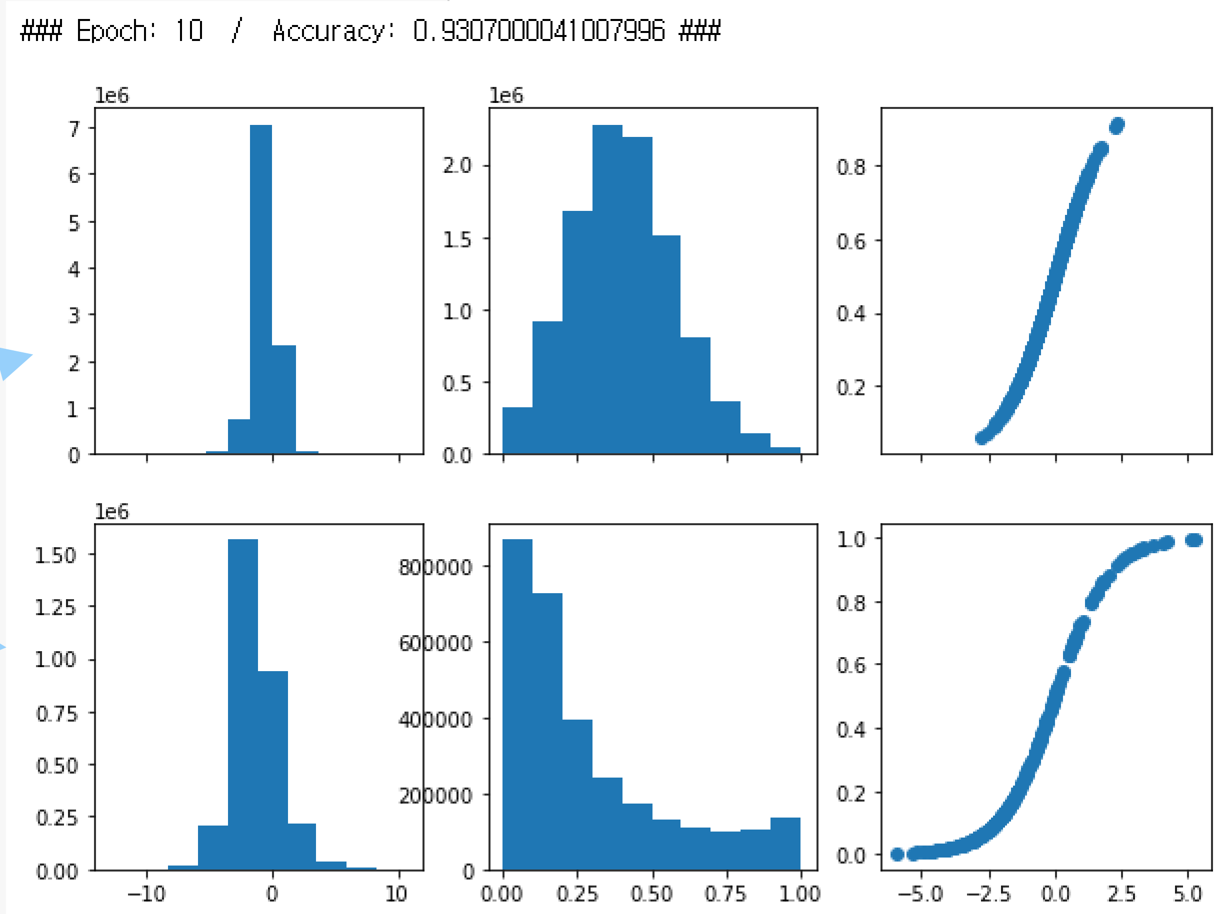
Batch Normalization - Look inside of network

Already prepared for you

```python
def plot_inner_dist():
    for epoch in range(10):
        net.train()
        for _data, _label in train_loader:
            data, label = _data.view([len(_data), -1]).to(device), _label.to(device)

            # Feed-forward
            _, _, _, _, onehot_out = net(data)
            loss = loss_fn(onehot_out, label)

            # Backward propagation
            optim.zero_grad()
            loss.backward()
            optim.step()

        net.eval()
        with torch.no_grad():
            for _data, _label in test_loader:
                data, label = _data.view([len(_data), -1]).to(device), _label.to(device)

                # Feed-forward
                o1, h1, o2, h2, onehot_out = net(data)

                _, out = torch.max(onehot_out, 1)
                acc_test = (out == label).float().sum()
                acc_test = acc_test / len(label)

        # plot inner distribution
        o1, h1, o2, h2 = o1.cpu().detach().numpy(), h1.cpu().detach().numpy(), o2.cpu().detach().numpy(),
        fig, axs = plt.subplots(2, 3, figsize=(10, 7), sharex='col')
        axs[0, 0].hist(o1.reshape(-1))
        axs[0, 1].hist(h1.reshape(-1))
        axs[0, 2].scatter(o1[0], h1[0])
        axs[1, 0].hist(o2.reshape(-1))
        axs[1, 1].hist(h2.reshape(-1))
        axs[1, 2].scatter(o2[0], h2[0])

        clear_output(wait=True)
        print(f'\n### Epoch: {epoch+1}  /  Accuracy: {acc_test} ###')
        plt.show()
```

Batch Normalization - Look inside of network


```python
class MLPSigmoid(nn.Module):
    def __init__(self, in_features, out_features):
        super(MLPSigmoid, self).__init__()

        self.hidden_layer1 = 1024
        self.hidden_layer2 = 300

        self.fc1 = nn.Linear(in_features, self.hidden_layer1)
        self.act1 = nn.Sigmoid()
        self.fc2 = nn.Linear(self.hidden_layer1, self.hidden_layer2)
        self.act2 = nn.Sigmoid()
        self.fc3 = nn.Linear(self.hidden_layer2, out_features)


    def forward(self, x):
        output1 = self.fc1(x)
        hidden1 = self.act1(output1)

        output2 = self.fc2(hidden1)
        hidden2 = self.act2(output2)

        onehot_out = self.fc3(hidden2)

        return output1, hidden1, output2, hidden2, onehot_out


init_model(MLPSigmoid(len(train_0_x.view([-1])), 10).to(device))
plot_inner_dist()
```

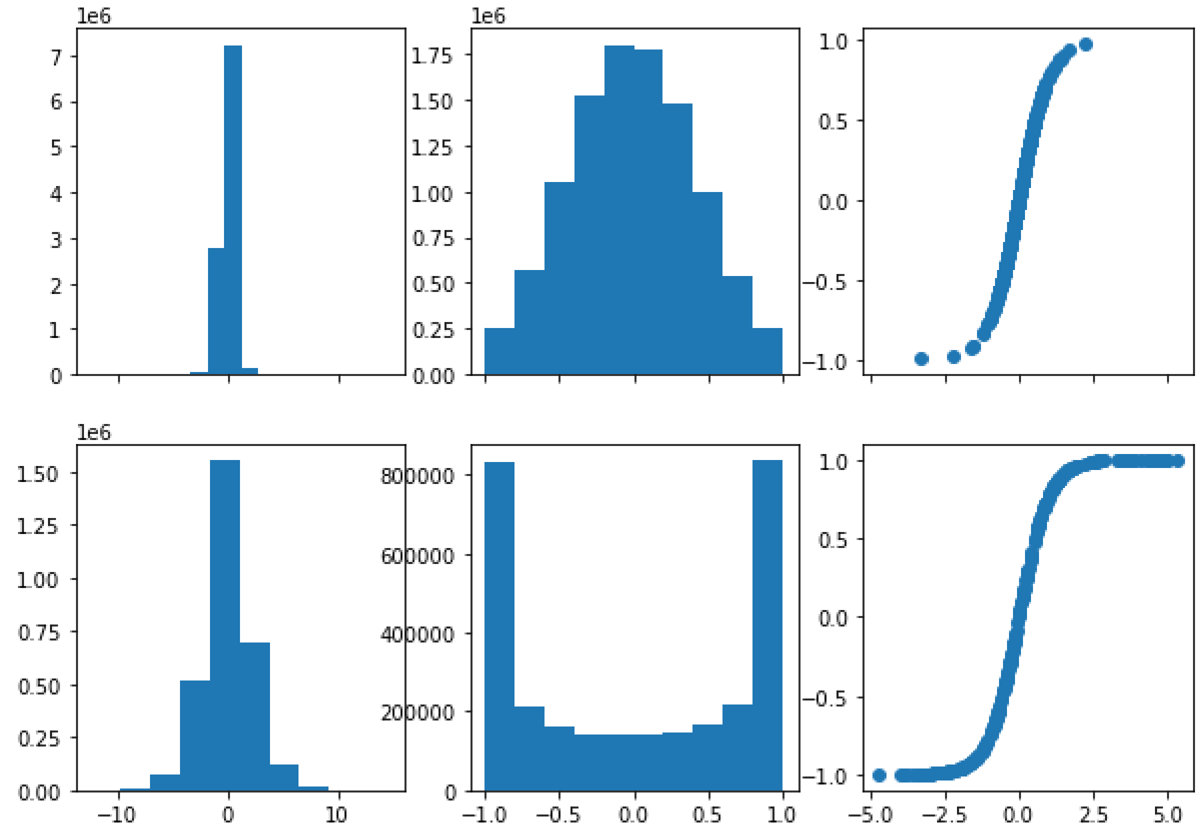


Axis Y : count / Axis X : value

Batch Normalization - Look inside of network


```python
class MLPTanh(nn.Module):
    def __init__(self, in_features, out_features):
        super(MLPTanh, self).__init__()

        self.hidden_layer1 = 1024
        self.hidden_layer2 = 300

        self.fc1 = nn.Linear(in_features, self.hidden_layer1)
        self.act1 = nn.Tanh()
        self.fc2 = nn.Linear(self.hidden_layer1, self.hidden_layer2)
        self.act2 = nn.Tanh()
        self.fc3 = nn.Linear(self.hidden_layer2, out_features)

    def forward(self, x):
        output1 = self.fc1(x)
        hidden1 = self.act1(output1)

        output2 = self.fc2(hidden1)
        hidden2 = self.act2(output2)

        onehot_out = self.fc3(hidden2)

        return output1, hidden1, output2, hidden2, onehot_out

init_model(MLPTanh(len(train_0_x.view([-1])), 10).to(device))
plot_inner_dist()
```

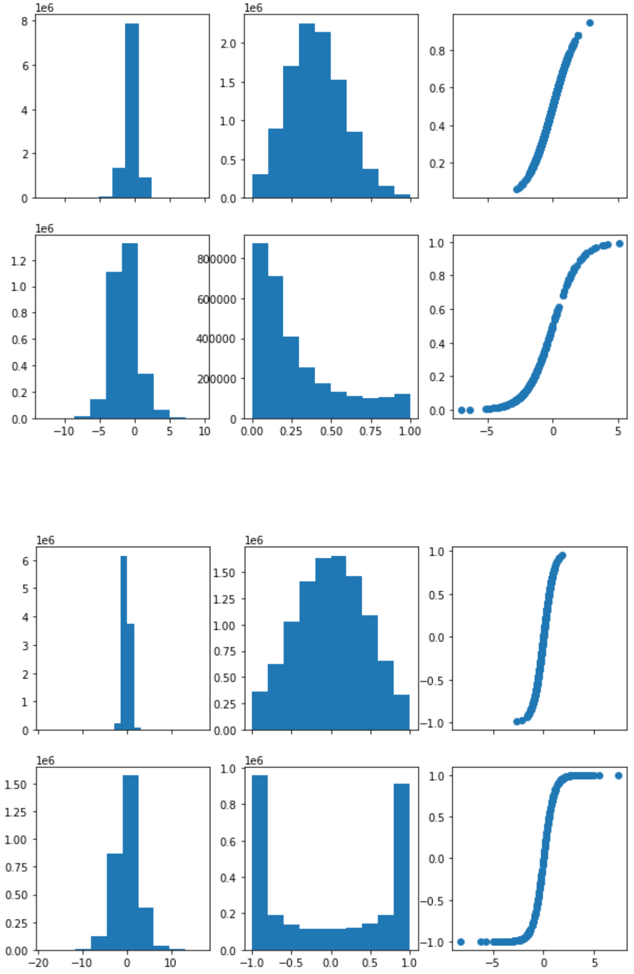
Epoch: 10 / Accuracy: 0.9781000018119812

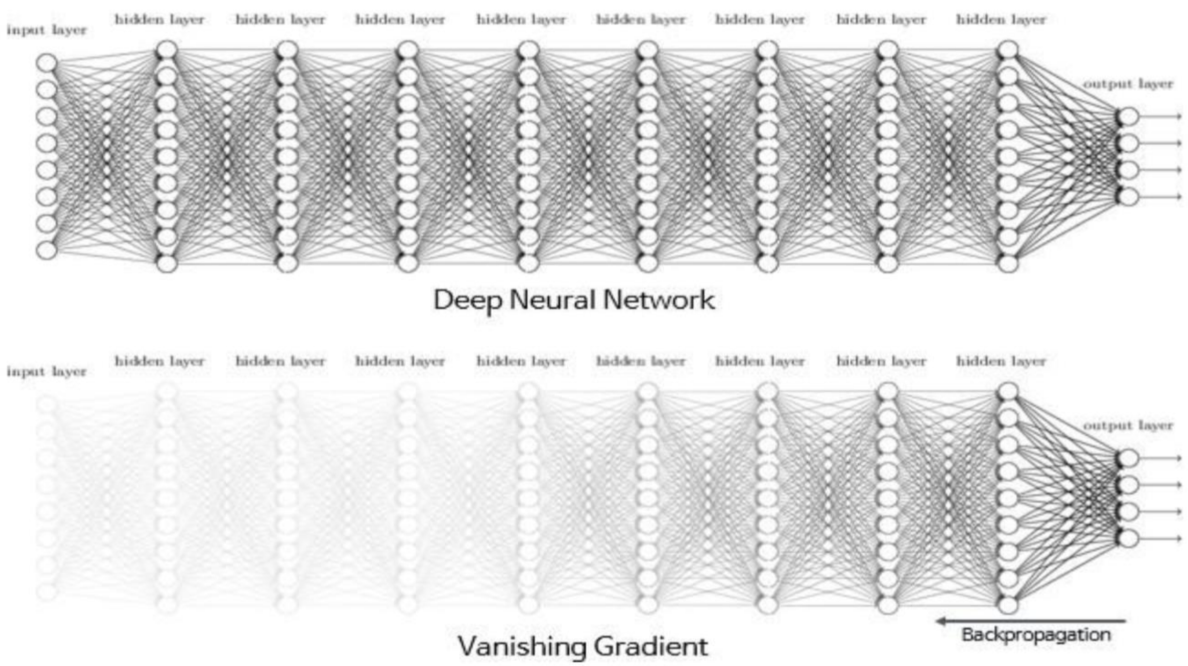


Batch Normalization - Look inside of network

Sigmoid

tanh

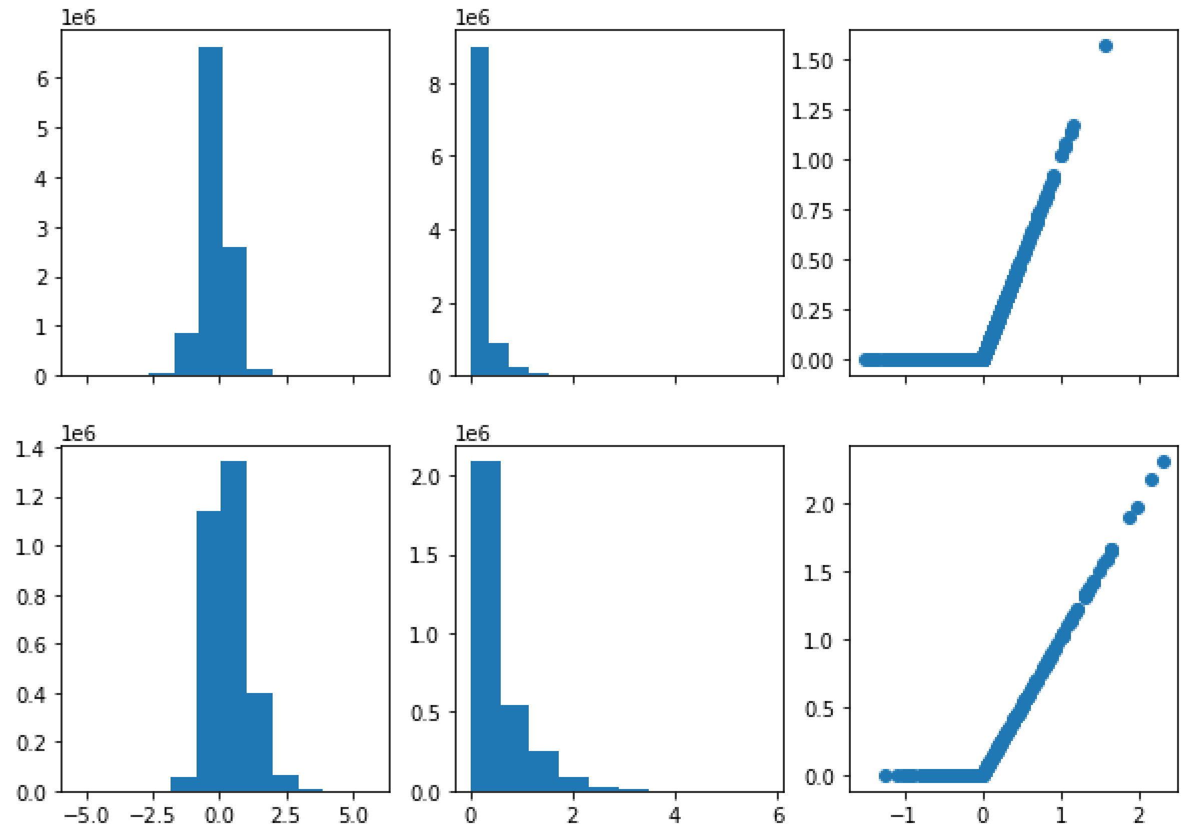




Batch Normalization - Look inside of network


```python
class MLPReLU(nn.Module):
    def __init__(self, in_features, out_features):
        super(MLPReLU, self).__init__()

        self.hidden_layer1 = 1024
        self.hidden_layer2 = 300

        self.fc1 = nn.Linear(in_features, self.hidden_layer1)
        self.act1 = nn.ReLU()
        self.fc2 = nn.Linear(self.hidden_layer1, self.hidden_layer2)
        self.act2 = nn.ReLU()
        self.fc3 = nn.Linear(self.hidden_layer2, out_features)

    def forward(self, x):
        output1 = self.fc1(x)
        hidden1 = self.act1(output1)

        output2 = self.fc2(hidden1)
        hidden2 = self.act2(output2)

        onehot_out = self.fc3(hidden2)

        return output1, hidden1, output2, hidden2, onehot_out

init_model(MLPReLU(len(train_0_x.view([-1])), 10).to(device))
plot_inner_dist()
```

Epoch: 10 / Accuracy: 0.9822999835014343

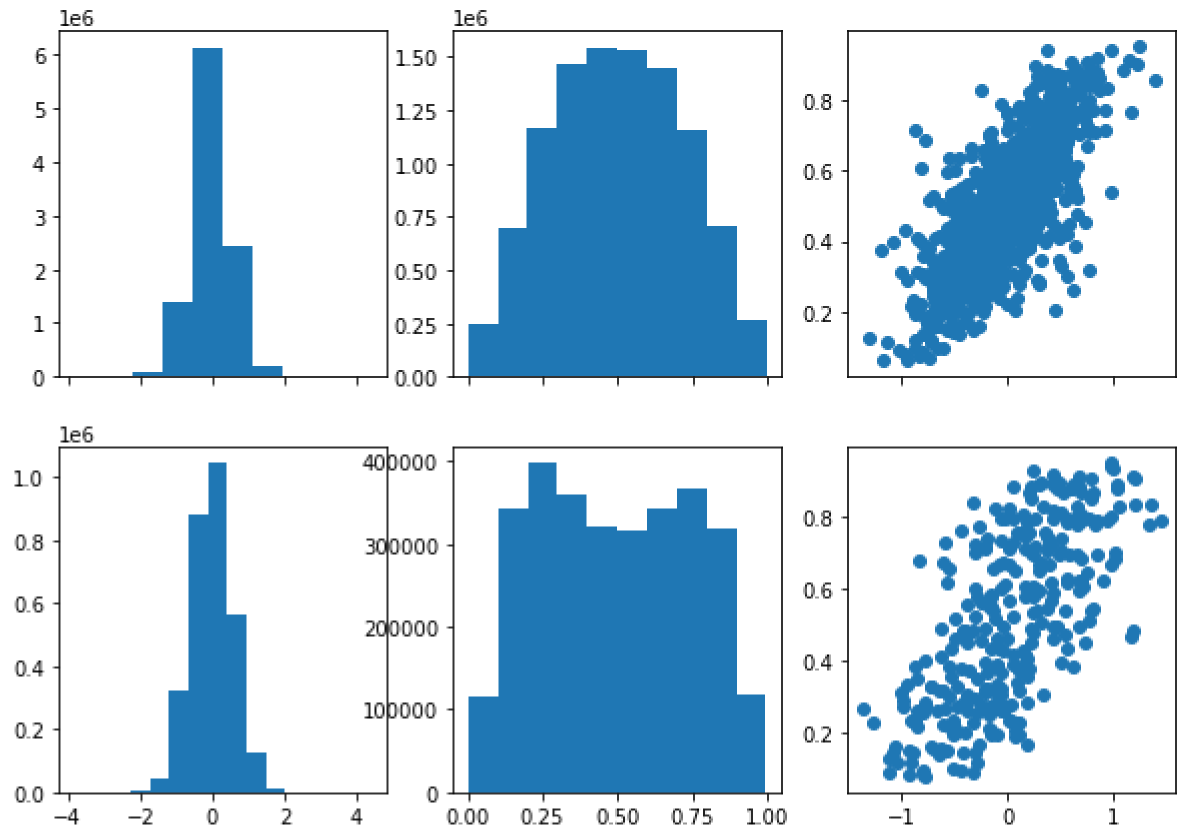


Batch Normalization

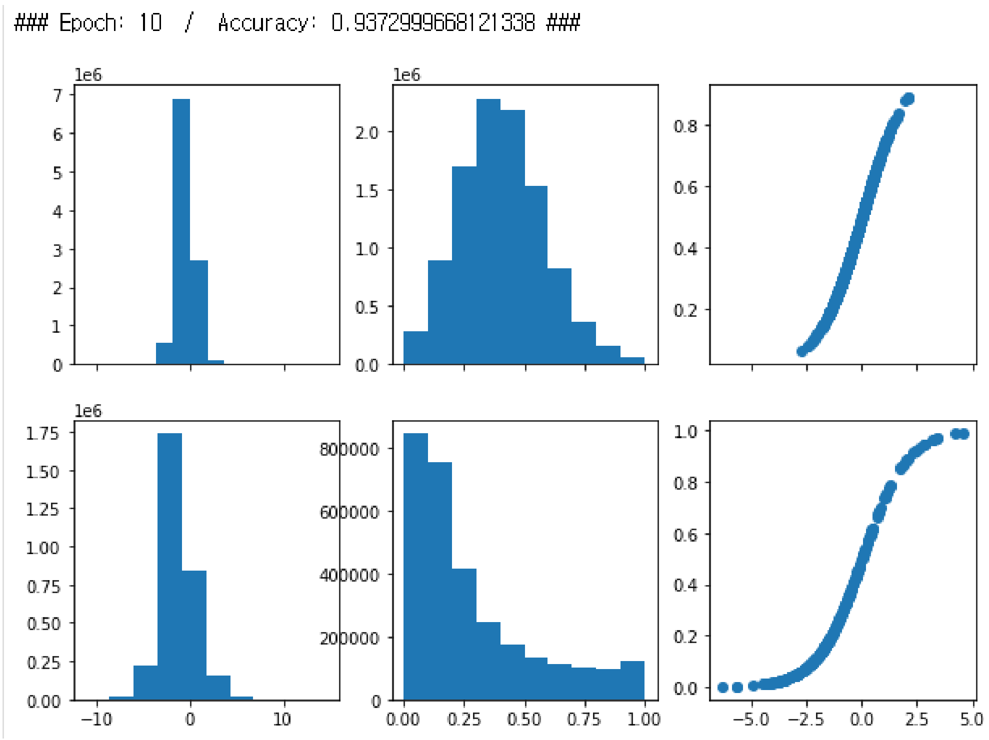

```python
class MLPSigmoidBatchNorm(nn.Module):
    def __init__(self, in_features, out_features):
        super(MLPSigmoidBatchNorm, self).__init__()

        self.hidden_layer1 = 1024
        self.hidden_layer2 = 300

        self.fc1 = nn.Linear(in_features, self.hidden_layer1)
        self.bn1 = nn.BatchNorm1d(self.hidden_layer1)
        self.act1 = nn.Sigmoid()
        self.fc2 = nn.Linear(self.hidden_layer1, self.hidden_layer2)
        self.bn2 = nn.BatchNorm1d(self.hidden_layer2)
        self.act2 = nn.Sigmoid()
        self.fc3 = nn.Linear(self.hidden_layer2, out_features)


    def forward(self, x):
        output1 = self.fc1(x)
        bn1 = self.bn1(output1)
        hidden1 = self.act1(bn1)

        output2 = self.fc2(hidden1)
        bn2 = self.bn2(output2)
        hidden2 = self.act2(bn2)

        onehot_out = self.fc3(hidden2)

        return output1, hidden1, output2, hidden2, onehot_out

init_model(MLPSigmoidBatchNorm(len(train_0_x.view([-1])), 10).to(device))
plot_inner_dist()
```

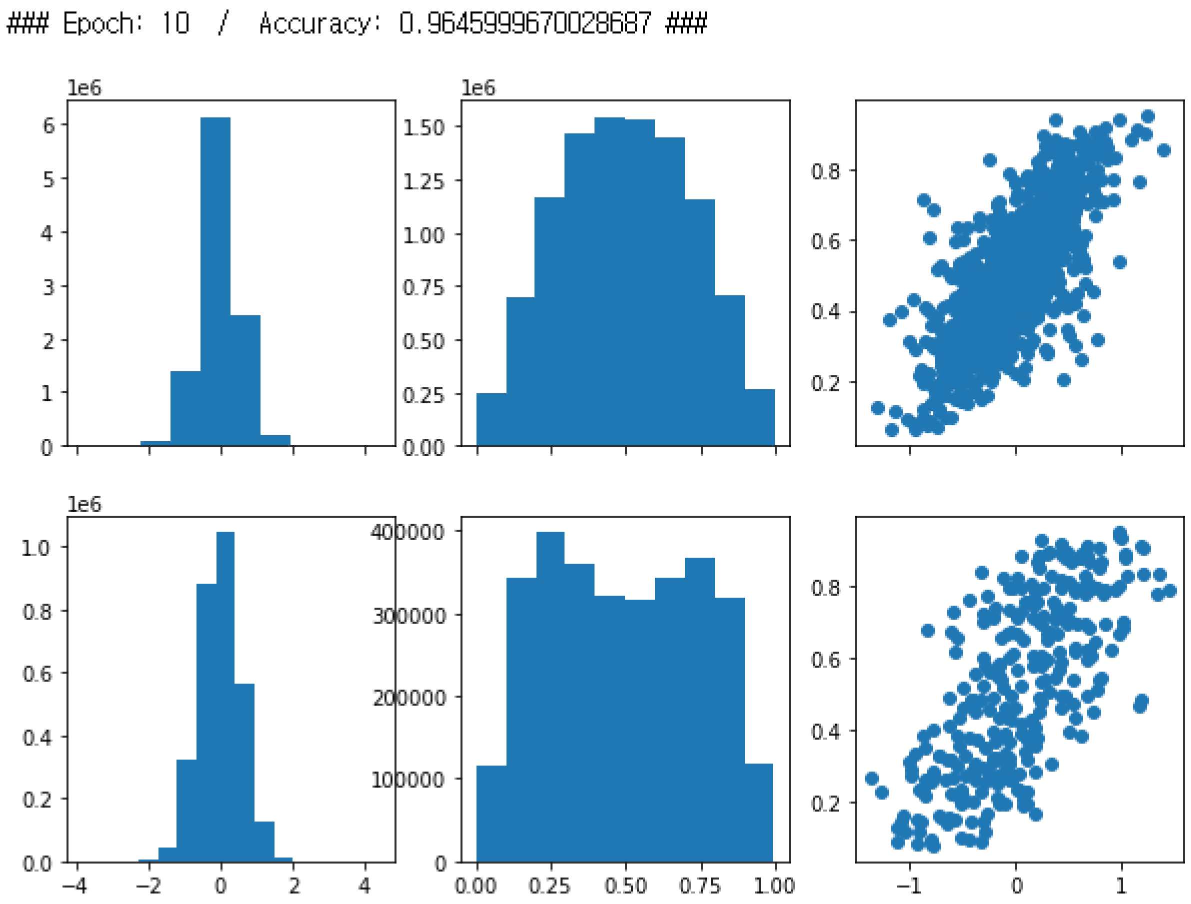
before activation

Epoch: 10 / Accuracy: 0.9645999670028687



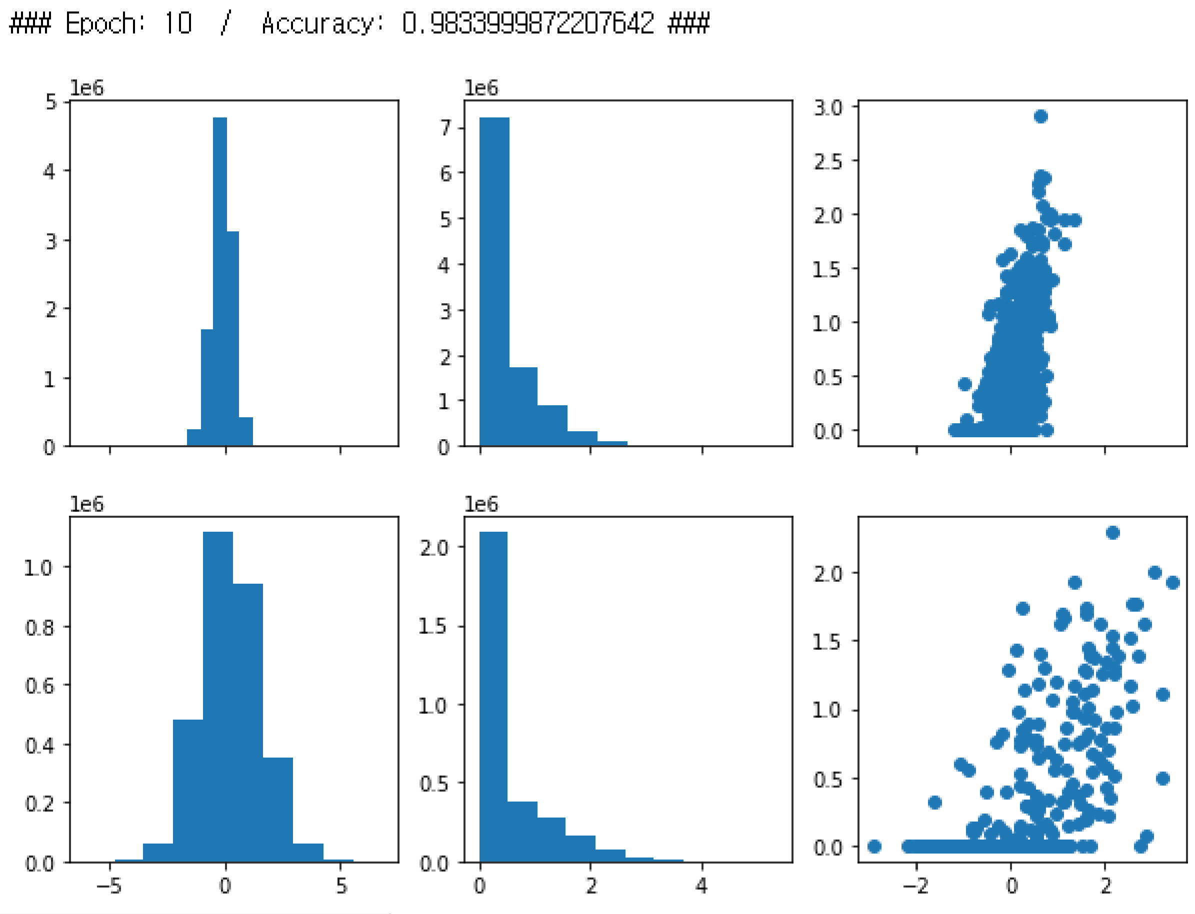
Batch Normalization





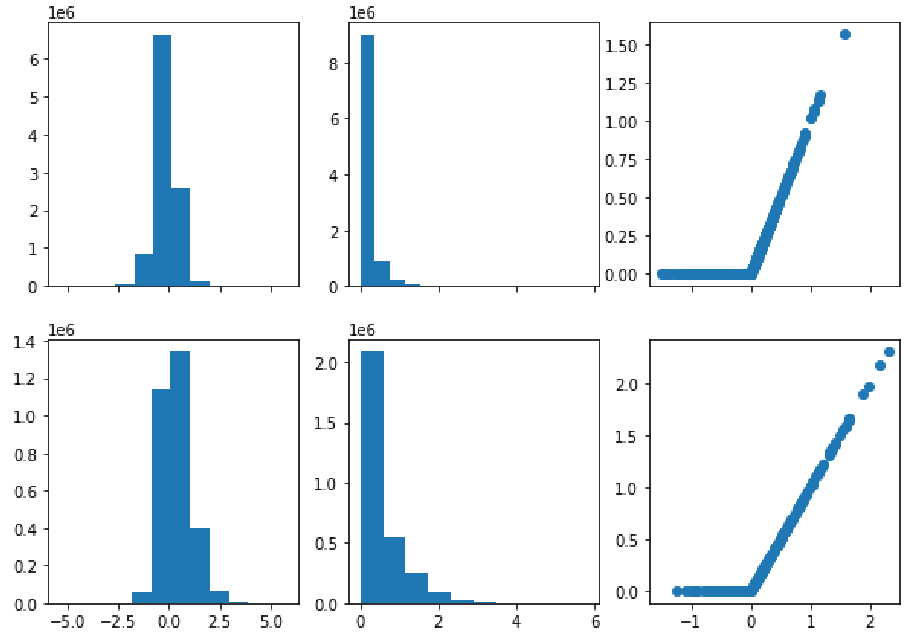
Batch Normalization


```python
class MLPReLUBatchNorm(nn.Module):
    def __init__(self, in_features, out_features):
        super(MLPReLUBatchNorm, self).__init__()

        self.hidden_layer1 = 1024
        self.hidden_layer2 = 300

        self.fc1 = nn.Linear(in_features, self.hidden_layer1)
        self.bn1 = nn.BatchNorm1d(self.hidden_layer1)
        self.act1 = nn.ReLU()
        self.fc2 = nn.Linear(self.hidden_layer1, self.hidden_layer2)
        self.bn2 = nn.BatchNorm1d(self.hidden_layer2)
        self.act2 = nn.ReLU()
        self.fc3 = nn.Linear(self.hidden_layer2, out_features)

    def forward(self, x):
        output1 = self.fc1(x)
        bn1 = self.bn1(output1)
        hidden1 = self.act1(bn1)

        output2 = self.fc2(hidden1)
        bn2 = self.bn2(output2)
        hidden2 = self.act2(bn2)

        onehot_out = self.fc3(hidden2)

        return output1, hidden1, output2, hidden2, onehot_out

init_model(MLPReLUBatchNorm(len(train_0_x.view([-1])), 10).to(device))
plot_inner_dist()
```

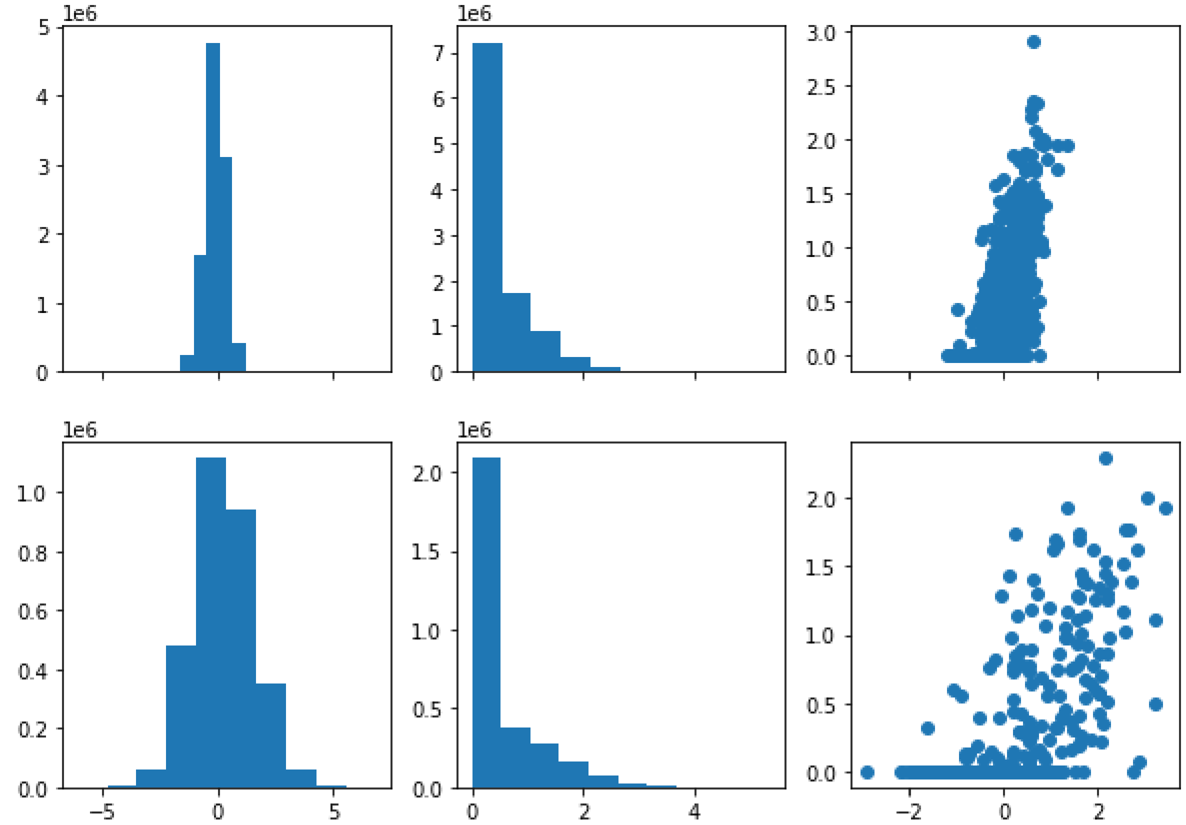


Batch Normalization

Epoch: 10 / Accuracy: 0.9822999835014343



Epoch: 10 / Accuracy: 0.9833999872207642

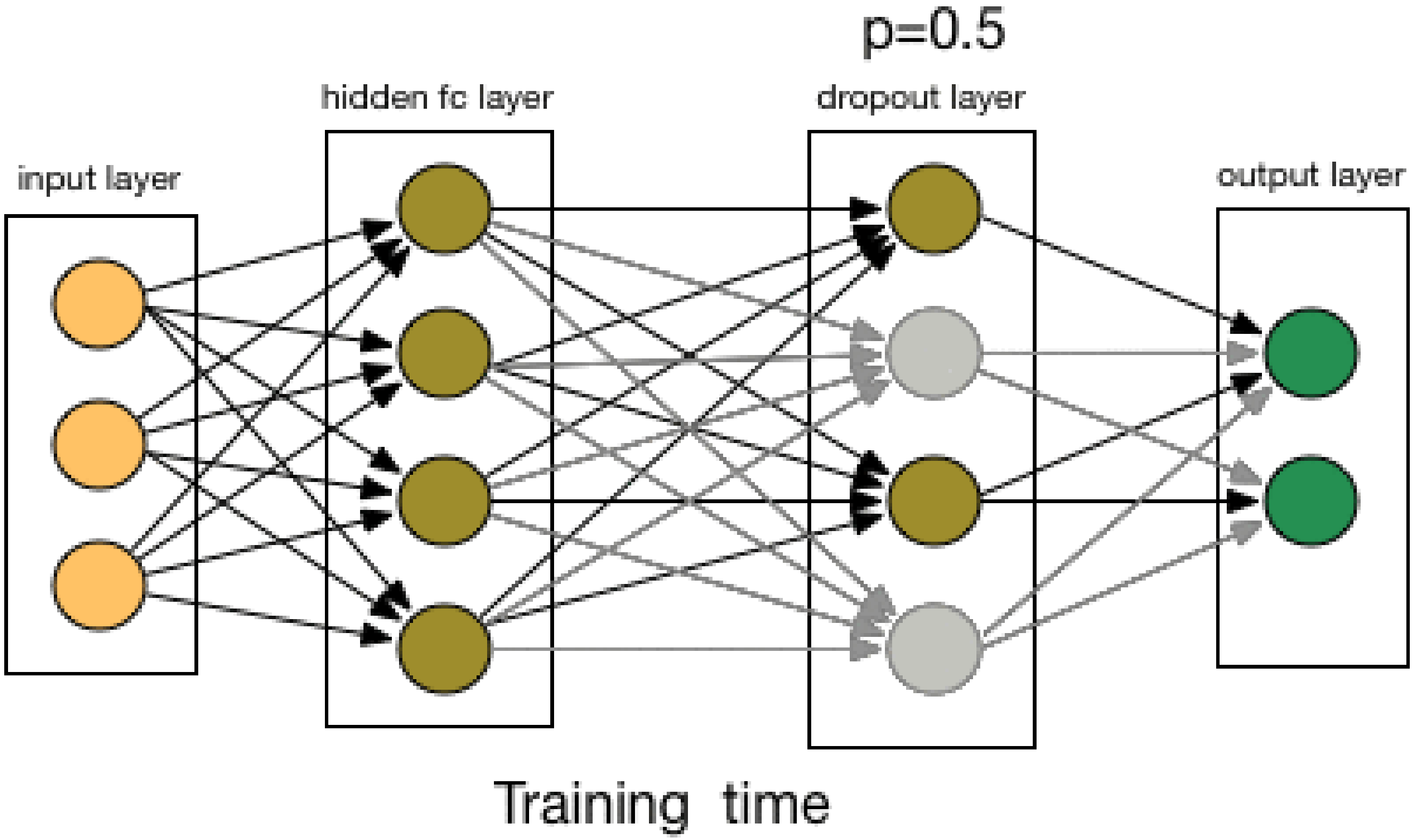


Batch Normalization

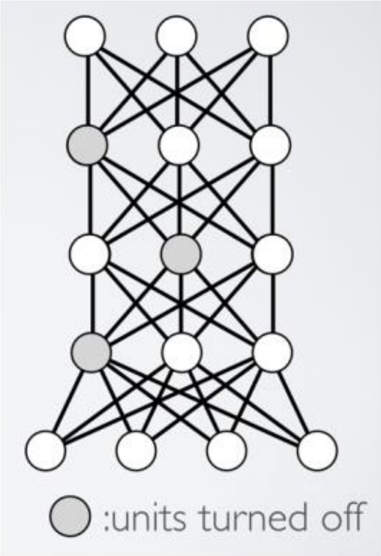
Accuracy Comparison

- `MLP Sigmoid` : 93.07 % → 96.80 %
- `MLP Tanh` : 97.81 %
- `MLP ReLU` : 98.23 % → 98.34 %

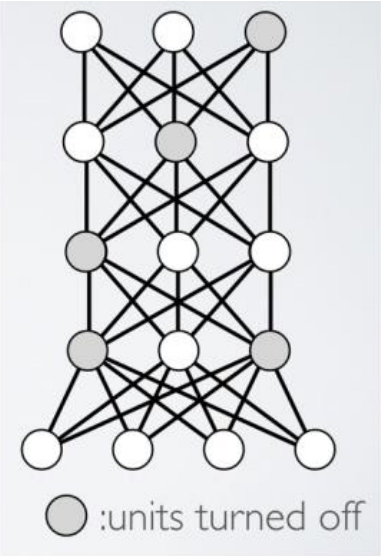
Dropout


p=0.5
hidden fc layer
dropout layer
input layer
output layer
Training time

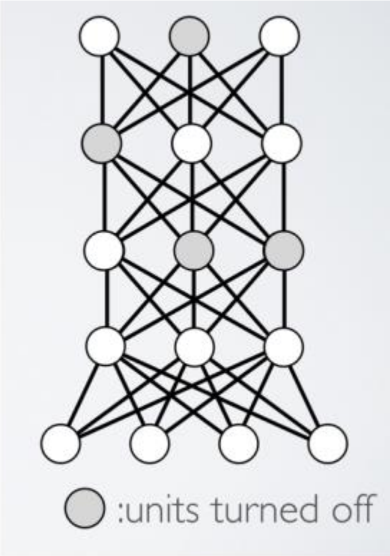
Dropout



얼굴위주



색지우고



귀 빼고

Dropout


```
class Dropout(nn.Module):
    def __init__(self, in_features, out_features):
        super(Dropout, self).__init__()

        self.hidden_layer = 32
        self.dropout_rate = .2  # probability

        self.fc1 = nn.Linear(in_features, self.hidden_layer)
        self.act1 = nn.ReLU()
        self.drop1 = nn.Dropout(self.dropout_rate)
        self.fc2 = nn.Linear(self.hidden_layer, out_features)

    def forward(self, x):
        hidden1 = self.act1(self.fc1(x))
        drop1 = self.drop1(hidden1)

        onehot_out = self.fc2(drop1)

        return onehot_out
```

after activation

Dropout


```python
# Training Initialization
init_model(Dropout(len(train_0_x.view([-1])), 10))
init_epoch()
init_log()

# Training Iteration

### Put Your Script Here ###

print('\n Training completed!')

# Accuracy for test dataset

### Put Your Script Here ###

    print('\n===================== Test Result =====================\n')
    print(f'Test accuracy = {test_acc}\nTest loss = {test_loss}')
```

Can you make this code by yourself?

Dropout

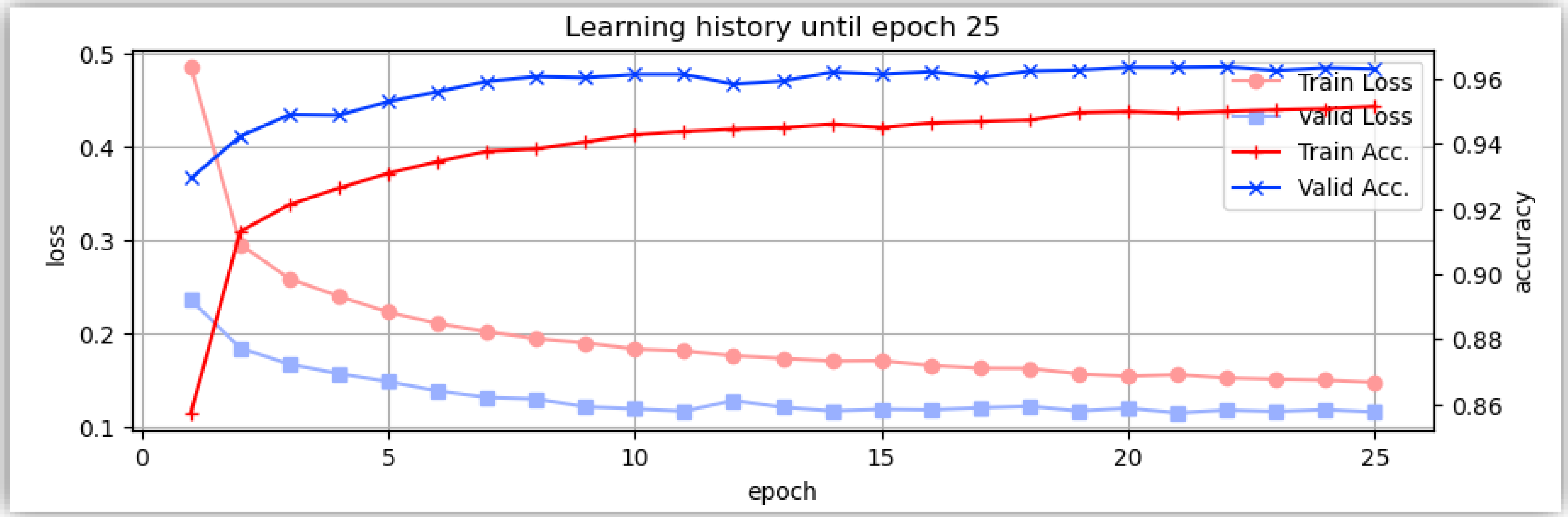

```python
# Training Iteration
while epoch_not_finished():
    start_time = time.time()
    net.train()
    tloss, tacc, _, _ = epoch(train_loader)
    end_time = time.time()
    time_taken = end_time - start_time
    record_train_log(tloss, tacc, time_taken)
    with torch.no_grad():
        net.eval()
        vloss, vacc, _, _ = epoch(valid_loader)
        record_valid_log(vloss, vacc)
    print_log()

print('\n Training completed!')

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

POP-UP QUIZ!

What’s the difference with before? and why?

Dropout

Anything Strange?



Batch Normalization + Dropout

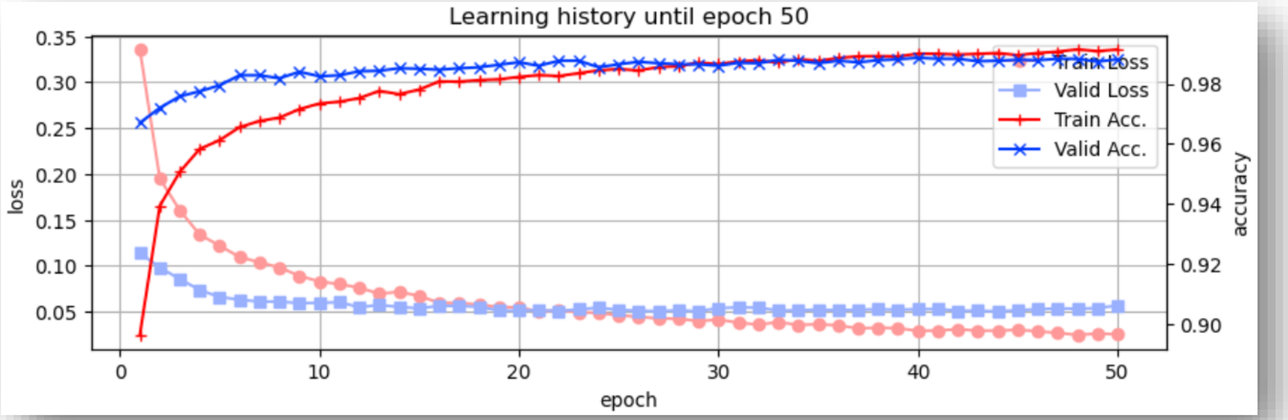
Final Round !

```
class MyModel(nn.Module):

    ### Put Your Script Here ###
```

Make your own model

- 2 hidden layers
- add batch normalization
- add dropout
- you can add additional techniques if you want

Batch Normalization + Dropout




```
Test accuracy = 0.9861999750137329
Test loss = 0.050298091024160385
```

"Beat the TA" competition