KU KONKUK UNIVERSITY

Convolutional Neural Network

자연어처리연구실

PART-I

ANN Programming with Class

```
import os
import numpy as np
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
from torch.utils.data import (DataLoader, RandomSampler, TensorDataset)
```

Configure in main

```
config = {"mode": "test",
          "model_name":"epoch_{0:d}.pt".format(1000),
          "output_dir":output_dir,
          "input_data":input_data,
          "input_node":2,
          "hidden_node":10,
          "output_node":1,
          "learn_rate":1,
          "batch_size":4,
          "epoch":1000,
          }
```

실습 코드 다운로드:
https://github.com
/KUNLP/Lecture

XOR Class

```
class XOR(nn.Module):

  def __init__(self, config):
    super(XOR, self).__init__()

    # 입력층 노드 수
    self.inode = config["input_node"]
    # 은닉층 데이터 크기
    self.hnode = config["hidden_node"]
    # 출력층 노드 수: 분류해야 하는 레이블 수
    self.onode = config["output_node"]

    # 활성화 함수로 Sigmoid 사용
    self.activation = nn.Sigmoid()

    # 신경망 설계
    self.linear1 = nn.Linear(self.inode, self.hnode, bias=True)
    self.linear2 = nn.Linear(self.hnode, self.onode, bias=True)
```

Hypothesis 만들기

```
  def forward(self, input_features):

    output1 = self.linear1(input_features)
    hypothesis1 = self.activation(output1)

    output2 = self.linear2(hypothesis1)
    hypothesis2 = self.activation(output2)

    return hypothesis2
```


ANN Programming with Class

Training 함수

```
# 모델 학습 함수
def train(config):

  # 모델 생성
  model = XOR(config).cuda()

  # 데이터 읽기
  (input_features, labels) = load_dataset(config["input_data"])

  # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
  train_features = TensorDataset(input_features, labels)
  train_dataloader = DataLoader(train_features, shuffle=True, batch_size=config["batch_size"])


  # 이진분류 크로스엔트로피 비용 함수
  loss_func = nn.BCELoss()
  # 옵티마이저 함수 (역전파 알고리즘을 수행할 함수)
  optimizer = torch.optim.SGD(model.parameters(), lr=config["learn_rate"])
```

```
# 데이터 읽기 함수
def load_dataset(file):
  data = np.loadtxt(file)
  print("DATA=",data)

  input_features = data[:,0:-1]
  print("INPUT_FEATURES=",input_features)

  labels = np.reshape(data[:,-1],(4,1))
  print("LABELS=",labels)

  input_features = torch.tensor(input_features, dtype=torch.float)
  labels = torch.tensor(labels, dtype=torch.float)

  return (input_features, labels)
```

ANN Programming with Class

```
for epoch in range(config["epoch"]+1):

  # 학습 모드 셋팅
  model.train()

  # epoch 마다 평균 비용을 저장하기 위한 리스트
  costs = []

  for (step, batch) in enumerate(train_dataloader):

    # batch = (input_features[step], labels[step])+batch_size
    # .cuda()를 통해 메모리에 업로드
    batch = tuple(t.cuda() for t in batch)

    # 각 feature 저장
    input_features, labels = batch

    # 역전파 변화도 초기화
    # .backward() 호출 시, 변화도 버퍼에 데이터가 계속 누적한 것을 초기화
    optimizer.zero_grad()

    # H(X) 계산: forward 연산
    hypothesis = model(input_features)
    # 비용 계산
    cost = loss_func(hypothesis, labels)
    # 역전파 수행
    cost.backward()
    optimizer.step()

    # 현재 batch의 스텝 별 loss 저장
    costs.append(cost.data.item())

  # 100 에폭마다 평균 loss 출력하고 모델을 저장
  if epoch%100 == 0:
    print("Average Loss= {0:f}".format(np.mean(costs)))
    torch.save(model.state_dict(), os.path.join(config["output_dir"], "epoch_{0:d}.pt".format(epoch)))
    do_test(model, train_dataloader)
```


Edited by Harksoo Kim

ANN Programming with Class

Test 함수

```
# 모델 평가 함수
def test(config):

  model = XOR(config).cuda()

  # 저장된 모델 가중치 로드
  model.load_state_dict(torch.load(os.path.join(config["output_dir"], config["model_name"])))

  # 데이터 load
  (features, labels) = load_dataset(config["input_data"])

  test_features = TensorDataset(features, labels)
  test_dataloader = DataLoader(test_features, shuffle=True, batch_size=config["batch_size"])

  do_test(model, test_dataloader)
```

```
# 모델 평가 결과 계산을 위해 텐서를 리스트로 변환하는 함수
def tensor2list(input_tensor):
    return input_tensor.cpu().detach().numpy().tolist()

# 평가 수행 함수
def do_test(model, test_dataloader):

  # 평가 모드 셋팅
  model.eval()

  # Batch 별로 예측값과 정답을 저장할 리스트 초기화
  predicts, golds = [], []

  with torch.no_grad():

    for step, batch in enumerate(test_dataloader):

      # .cuda()를 통해 메모리에 업로드
      batch = tuple(t.cuda() for t in batch)

      input_features, labels = batch
      hypothesis = model(input_features)
      logits = (hypothesis > 0.5).float()
      x = tensor2list(logits)
      y = tensor2list(labels)

      # 예측값과 정답을 리스트에 추가
      predicts.extend(x)
      golds.extend(y)

  print("PRED=",predicts)
  print("GOLD=",golds)
  print("Accuracy= {0:f}\n".format(accuracy_score(golds, predicts)))
```

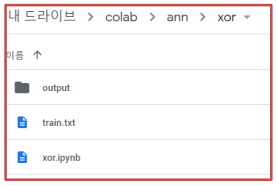

Edited by Harksoo Kim

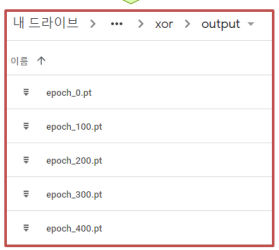
ANN Programming with Class

Training in Main

```
if(__name__=="__main__"):

    root_dir = "/gdrive/My Drive/colab/ann/xor"
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    input_data = "{0:s}/{1:s}".format(root_dir,"train.txt")

    config = {"mode": "train"
              "model_name":"epoch_{0:d}.pt".format(1000),
              "output_dir":output_dir,
              "input_data":input_data,
              "input_node":2,
              "hidden_node":10,
              "output_node":1,
              "learn_rate":1,
              "batch_size":4,
              "epoch":1000,
              }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

```
DATA= [[0. 0. 0.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 0.]]
INPUT_FEATURES= [[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]]
LABELS= [[0.]
 [1.]
 [1.]
 [0.]]
Average Loss= 0.693318
PRED= [[0.0], [0.0], [1.0], [1.0]]
GOLD= [[1.0], [0.0], [0.0], [1.0]]
Accuracy= 0.500000

Average Loss= 0.690562
PRED= [[1.0], [1.0], [0.0], [0.0]]
GOLD= [[1.0], [0.0], [1.0], [0.0]]
Accuracy= 0.500000

Average Loss= 0.667773
PRED= [[0.0], [1.0], [0.0], [1.0]]
GOLD= [[1.0], [1.0], [0.0], [0.0]]
Accuracy= 0.500000

Average Loss= 0.433278
PRED= [[1.0], [0.0], [0.0], [1.0]]
GOLD= [[1.0], [0.0], [0.0], [1.0]]
Accuracy= 1.000000

Average Loss= 0.133441
PRED= [[0.0], [1.0], [0.0], [1.0]]
GOLD= [[0.0], [1.0], [0.0], [1.0]]
Accuracy= 1.000000
```





Edited by Harksoo Kim

ANN Programming with Class

Test in Main

```
if(__name__=="__main__"):

    root_dir = "/gdrive/My Drive/colab/ann/xor"
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    input_data = "{0:s}/{1:s}".format(root_dir,"train.txt")

    config = {"mode": "test",
              "model_name":"epoch_{0:d}.pt".format(1000)
              "output_dir":output_dir,
              "input_data":input_data,
              "input_node":2,
              "hidden_node":10,
              "output_node":1,
              "learn_rate":1,
              "batch_size":4,
              "epoch":1000,
              }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

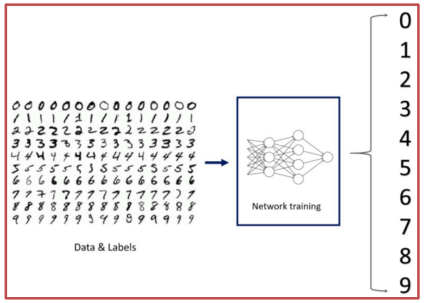
1,000 epoch 모델 결과를 읽어 들여 테스트 수행

```
DATA= [[0. 0. 0.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 0.]]
INPUT_FEATURES= [[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]]
LABELS= [[0.]
 [1.]
 [1.]
 [0.]]
PRED= [[0.0], [1.0], [1.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy= 1.000000
```


Edited by Harksoo Kim

실습

- XOR 문제에 사용한 "ANN Class in PyTorch" 프로그램을 수정하여 손 글씨 숫자를 판독하는 프로그램을 작성하시오.
  - 입력 데이터셋
    - MNIST dataset
    - 0~9 손 글씨 이미지에 대한 픽셀 값 데이터
  - 문제
    - 이미지의 픽셀 값을 입력으로 하여 해당 이미지가 0~9 중에 어떤 숫자인지 분류



실습

MNIST Class

```
class MNIST(nn.Module):

  def __init__(self, config):
    ?
    # 입력층 노드 수
    self.inode = config["input_node"]
    # 은닉층 데이터 크기
    self.hnode = config["hidden_node"]
    # 출력층 노드 수: 분류해야 하는 레이블 수
    self.onode = config["output_node"]

    # 활성화 함수로 Sigmoid 사용
    self.activation = nn.Sigmoid()

    # 신경망 설계
    self.linear1 = nn.Linear(self.inode, self.hnode, bias=True)
    self.linear2 = nn.Linear(self.hnode, self.onode, bias=True)

  def forward(self, input_features):

    output1 = self.linear1(input_features)
    hypothesis1 = self.activation(output1)

    output2 = self.linear2(hypothesis1)
    hypothesis2 = self.activation(output2)

    return hypothesis2
```

데이터 읽기

```
import os
import numpy as np
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
from torch.utils.data import (DataLoader, RandomSampler, TensorDataset)
from keras.datasets import mnist

# 데이터 읽기 함수
def load_dataset():
  ? (keras.datasets)

  train_X = train_X.reshape(-1, 28*28)
  print(train_X.shape)
  test_X  = test_X.reshape(-1, 28*28)

  train_X = torch.tensor(train_X, dtype=torch.float)
  train_y = torch.tensor(train_y, dtype=torch.long)
  test_X = torch.tensor(test_X, dtype=torch.float)
  test_y = torch.tensor(test_y, dtype=torch.long)

  return (train_X, train_y), (test_X, test_y)
```


실습

Test 함수

```
# 모델 평가 함수
def test(config):
  ?

  # 저장된 모델 가중치 로드
  model.load_state_dict(torch.load(os.path.join(config["output_dir"], config["model_name"])))

  ?

  test_features = TensorDataset(features, labels)
  test_dataloader = DataLoader(test_features, shuffle=True, batch_size=config["batch_size"])

  do_test(model, test_dataloader)
```


실습

Training 함수

```
# 모델 학습 함수
def train(config):
  ?

  # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
  train_features = TensorDataset(input_features, labels)
  train_dataloader = DataLoader(train_features, shuffle=True, batch_size=config["batch_size"])
  ?

    # 현재 batch의 스텝 별 loss 저장
    costs.append(cost.data.item())

  # 에폭마다 평균 비용을 출력하고 모델을 저장
  print("Average Loss= {0:f}".format(np.mean(costs)))
  torch.save(model.state_dict(), os.path.join(config["output_dir"], "epoch_{0:d}.pt".format(epoch)))
  do_test(model, train_dataloader)
```

실습

Main

```
if(__name__=="__main__"):

    root_dir = "/gdrive/My Drive/colab/ann/mnist"
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    config = {"mode": "train",
              "model_name":"epoch_{0:d}.pt".format(10),
              "output_dir":output_dir,
              "input_node":784,
              "hidden_node":512,
              "output_node":10,
              "learn_rate":0.001,
              "batch_size":32,
              "epoch":10,
              }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

```
Downloading data from https://storage.googleapis.com/tensorflow
11493376/11490434 [==============================] - 0s 0us/ste
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
(60000, 784)
Average Loss= 1.620250
PRED= [3, 2, 1, 0, 4, 1, 4, 1, 9, 6, 1, 4, 5, 7, 5, 4, 7, 8, 4,
GOLD= [3, 2, 1, 0, 4, 1, 4, 1, 9, 6, 1, 4, 5, 7, 5, 4, 7, 2, 4,
Accuracy= 0.915850

Average Loss= 1.563620
PRED= [0, 8, 7, 9, 1, 7, 7, 6, 3, 6, 9, 3, 1, 6, 8, 7, 2, 1, 9,
GOLD= [0, 8, 7, 3, 1, 7, 7, 6, 3, 6, 9, 3, 1, 6, 8, 7, 2, 1, 9,
Accuracy= 0.918350

Average Loss= 1.551876
PRED= [9, 8, 8, 4, 3, 0, 5, 9, 0, 2, 1, 8, 7, 1, 1, 1, 8, 2, 9,
GOLD= [9, 8, 8, 4, 3, 6, 5, 9, 0, 2, 1, 8, 7, 1, 1, 1, 5, 2, 9,
Accuracy= 0.922050
```


PART-II에 계속됩니다!