딥러닝 연습 - Tensorflow 기초 3

Lecture Notes on 인공지능입문, Spring 2018

Made by JeHwan Ryu

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National Univertisy

http://bi.snu.ac.kr

Troubleshooting

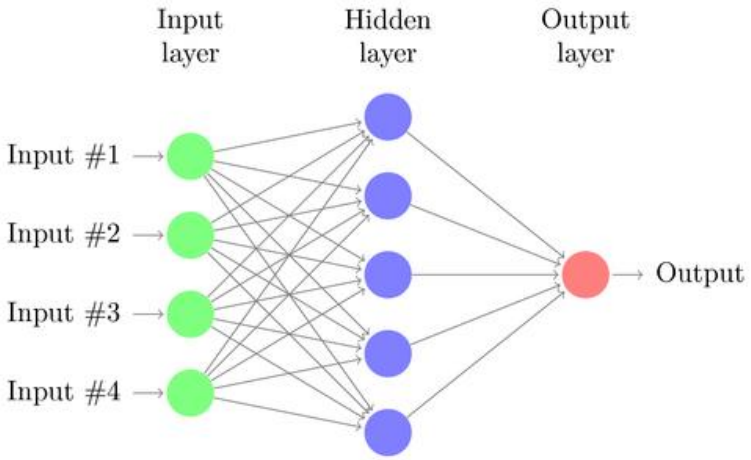
- 공통 시도 사항
  - python –m pip install --upgrade pip
- Tensorboard가 작동을 안하는 경우
  - 1. python –m pip install tensorboar
  - 2. python –m pip show tensorboard (or python –m pip show tensorflow)
  - 3. 그 location으로 이동
  - 4. cd tensorboard
  - 5. python main.py --logdir=/path/to/log_file/
- Imshow에서 blank canvas만 뜨는 경우
  - plt.show() 를 추가
  - 다른 이미지로 시도
  - 이미지를 어레이로 바꿔서 시도(img = np.asarray(img))
- Jupyter 환경 외에서 Matplotlib이 잘 안되는 경우
  - python –m pip install matplotlib (or python –m pip install --upgrade matplotlib)
  - Plt.show() 를 추가
- Pillow 에러가 나는 경우
  - python –m pip install pillow (or python –m pip install --upgrade pillow)

Multi-Layer Perceptron 구현

Multi-Layer Perceptron

- 정의
  - 입력층과 출력층 사이에 하나 이상의 중간층(은닉층, hidden layer)이 존재하는 신경망
  - 각 층 내의 연결, 출력층에서 입력층으로의 직접적 연결은 존재하지 않는 전방향(feedforward) 네트워크

- 특성
  - 퍼셉트론(Perceptron)의 다층화
  - 인간의 뉴런 구조 모방
  - 딥러닝 모델의 근간



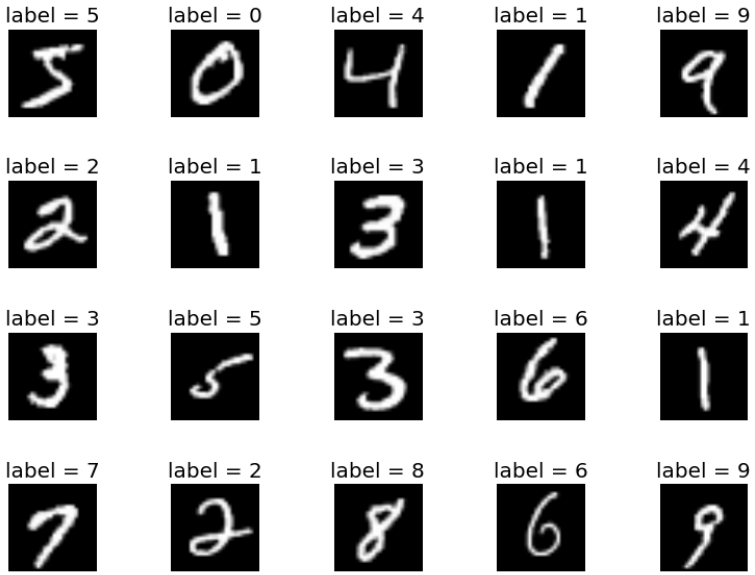
데이터 넣기

- MNIST 데이터 다운받기
  - https://www.dropbox.com/s/457cvbwbhd82uab/mnist.hdf5?dl=0
  - .ipynb(노트북 파일)이 있는 폴더/data 에 저장 권장
  - 혹은 원하는 다른 폴더

- 노트북 파일 위치 찾기
  - 터미널(cmd)에서 jupyter notebook 커맨드를 실행시키면 해당 폴더가 jupyter gui가 처음 실행될 때 해당 폴더를 기준으로 열림
  - 위치를 찾지 못하겠으면 cmd에서 ctrl+c(정지 커맨드)를 눌러 현재 폴더를 확인
  - “./data” 라고 경로를 표현할 때 “./”은 현재 폴더를 의미

MNIST

- 1990년대 말 얀 리쿤의 연구실에서 수집하고 배포한 데이터
  - Label : 0~9
  - Image shape : 28x28
  - # of data : Training 60,000 (6,000 / class) / Test 10,000 (1,000 / class)



Functions

- 해당 notebook에 포함되어있는 함수들 (Tensorflow 자체 제공 함수가 아님)

- load_data(dataset)
  - cifar-10 이나 Mnist 데이터를 로드해주는 함수
  - dataset = “cifar”일 경우 cifar-10을 로드,
    그 외의 경우 Mnist를 로드
  - 해당 노트북(혹은 python 메인 파일)이 있는 폴더 밑의 data에 hdf5 파일이 존재해야 함

- load_cifar(file_name), load_mnist(file_name)
  - load_data(dataset)의 하위 함수로, 실제적으로 각 데이터셋을 불러오는 일을 함

```python
# 데이터를 로드해주는 함수입니다.
def load_data(dataset):
    if (dataset=="cifar"):
        return load_cifar()
    else:
        return load_mnist()

def load_cifar(file_name = './data/cifar-10.hdf5'):
    hf = h5py.File(file_name,'r')
    print('datasets: {}'.format(hf.keys()))

    shapes = hf['train_set_x'].shape
    trainingSize = hf['train_set_x'].shape[0]
    validationSize = hf['valid_set_x'].shape[0]
    testSize = hf['test_set_x'].shape[0]
    w = shapes[2]
    h = shapes[3]
    c = shapes[1]
    print ("size of each set (train: %d, valid: %d, test: %d): " %(trainingSize, validationSize, testSize))

    return {'w': w, 'h': h, 'c': c, 'tr_size':trainingSize, 'v_size':validationSize,
            'te_size':testSize, 'data': hf}

def load_mnist(file_name = './data/mnist.hdf5'):
    hf = h5py.File(file_name,'r')
    print('datasets: {}'.format(hf.keys()))

    shapes = hf['x_train'].shape
    trainingSize = hf['x_train'].shape[0]
    validationSize = hf['x_valid'].shape[0]
    testSize = hf['x_test'].shape[0]

    w = np.sqrt(shapes[1]).astype(int)
    h = np.sqrt(shapes[1]).astype(int)
    c = 1
    print ("size of each set (train: %d, valid: %d, test: %d): " %(trainingSize, validationSize, testSize))

    return {'w': w, 'h': h, 'c': c, 'tr_size':trainingSize, 'v_size':validationSize,
            'te_size':testSize, 'data': hf}
```

Functions

- dense_to_one_hot(labels_dense, num_classes=10)
  - labels_dense 텐서를 받아서 one hot 텐서를 반환해주는 함수
  - num_classes는 클래스 개수이며, 별도의 입력이 없으면 기본적으로 10이 배정
  - tf.one_hot(indices, depth)
    - 주어진 indices의 one hot tensor를 return 해주는 함수
    - Indices : one hot으로 변환할 값 (현재는 class label)
    - Depth : one hot의 깊이 (현재는 class의 개수)

- load_batch(dataset, hf, phase, startIdx, endIdx)
  - startIdx부터 endIdx까지의 data를 하나의 batch로 반환해주는 함수
  - dataset = “cifar”일 경우 cifar-10을 로드, 그 외의 경우 Mnist를 로드
  - hf : 로딩된 데이터의 hf 객체
  - phase = “train”, “valid”, “test”가 가능하며 각각 해당하는 유형의 데이터를 반환
  - startIdx : 읽어오길 원하는 데이터 범위의 시작지점 index
  - endIdx : 읽어오길 원하는 데이터 범위의 끝 지점 index

```
'''Convert class labels from scalars to one-hot vectors.'''
def dense_to_one_hot(labels_dense, num_classes=10):
    labels_one_hot = tf.one_hot(labels_dense, num_classes, name="labels_one_hot")
    return labels_one_hot

def load_batch(dataset, hf, phase, startIdx, endIdx):
    if(dataset == "cifar"):
        if(phase == "train"):
            x_ = np.array(hf['train_set_x'][startIdx:endIdx])
            x_ = np.rollaxis(x_,1,4)
            y_ = np.array(hf['train_set_y'][startIdx:endIdx])
            return x_, y_
        elif(phase == "valid"):
            x_ = np.array(hf['valid_set_x'][startIdx:endIdx])
            x_ = np.rollaxis(x_,1,4)
            y_ = np.array(hf['valid_set_y'][startIdx:endIdx])
            return x_, y_
        else:
            x_ = np.array(hf['test_set_x'][startIdx:endIdx])
            x_ = np.rollaxis(x_,1,4)
            y_ = np.array(hf['test_set_y'][startIdx:endIdx])
            return x_, y_
    else:
        if(phase == "train"):
            x_ = np.array(hf['x_train'][startIdx:endIdx]).reshape(batch_size,28,28,1)
            y_ = np.array(hf['t_train'][startIdx:endIdx])
            return x_, y_
        elif(phase == "valid"):
            x_ = np.array(hf['x_valid'][startIdx:endIdx]).reshape(batch_size,28,28,1)
            y_ = np.array(hf['t_valid'][startIdx:endIdx])
            return x_, y_
        else:
            x_ = np.array(hf['x_test'][startIdx:endIdx]).reshape(batch_size,28,28,1)
            y_ = np.array(hf['t_test'][startIdx:endIdx])
            return x_, y_
```

Functions

- linear(input_, output_size, scope=None, stddev=0.1, with_w=False)
  - (Nonlinear 부분은 포함되지 않은) MLP의 은닉층을 구성하는 연산
  - input_ : 해당 layer의 input
    output_size : 해당 layer에서 output으로 나가는 size
    scope : 해당 layer에 붙이고 싶은 scope name,
    없으면 scope name은 “Linear”가 됨
    stddev : 해당 layer의 weight 및 bias의 초기값 standard deviation
    with_w : weight 및 bias 텐서를 return 할지의 여부. 기본값은 False

  - tf.truncated_normal(shape, dtype, stddev) 및
    tf.random_normal(shape, dtype, stddev) 함수
    - 특정 분포에서 random으로 값을 선택해주는 함수
    - Random_normal : 정규분포,
      Truncated_normal : 정규분포에서 일부 구간을 잘라낸 분포
    - shape : 원하는 tensor의 행렬수 정보 list (ex. [20] or [10, 10])
    - stddev : 해당 분포의 standard deviation

```python
def linear(input_, output_size, scope=None, stddev=0.1, with_w=False):
    shape = input_.get_shape().as_list()
    with tf.variable_scope(scope or "Linear"):
        matrix = tf.Variable(tf.truncated_normal([shape[1], output_size],
                                                 dtype=tf.float32,
                                                 stddev=stddev), name='Matrix')

        bias = tf.Variable(tf.random_normal([output_size], dtype=tf.float32, stddev=stddev), name='bias')
        if with_w:
            return tf.matmul(input_, matrix) + bias, matrix, bias
        else:
            return tf.matmul(input_, matrix) + bias
```

Functions

- calc_Acc(hf, phase)
  - 정확도(accuracy)를 계산해주는 함수
  - hf : 로딩된 데이터의 hf 객체
    phase = “train”, “valid”, “test”가 가능하며 각각 해당하는 유형의 데이터를 사용하여 accuracy 계산
  - temp_accuracy에 각 batch의 accuracy를 적립한 후에 전체 batch 수로 나누어 정확도를 계산

```
def calc_Acc(hf, phase):
    startIdx = 0
    endIdx = batch_size
    temp_accuracy = 0
    if (phase == 'train'):
        n_batch = n_train_batches
    elif (phase == 'valid'):
        n_batch = n_valid_batches
    else:
        n_batch = n_test_batches

    for minibatch in range(int(n_batch)):
        x_data, y_data = load_batch(dataset, hf, phase, startIdx, endIdx)
        startIdx = endIdx
        endIdx = endIdx + batch_size
        temp_accuracy = temp_accuracy + accuracy.eval(feed_dict={x: x_data, y_: y_data})

    temp_accuracy = temp_accuracy / n_batch
    return temp_accuracy
```

코드 설명

```
# sess = tf.Session()
# sess = tf.Session(config=tf.ConfigProto(gpu_options=tf.GPUOptions(allow_growth=True)))
sess = tf.InteractiveSession()
```

- 세션 생성
  - InteractiveSession()을 사용할 경우 default session이 현재 세션으로 자동 설정 (eval 함수를 사용할때 따로 session을 줄 필요가 없음)
  - (optional) GPU를 사용할 경우 위와 같이 옵션을 주지 않으면 GPU 메모리를 추가로 사용 불가능!

Preparing

- 각종 parameter 및 변수 지정
  - Batch_size : 하나의 batch에 들어가는 데이터의 수
  - *Size : 각 데이터 유형(train, valid, test)의 수
  - n_*_batches : 각 데이터 유형별 batch의 수(batch size가 아님)

Preparing

```
## 학습 셋팅입니다.
dataset = "mnist" # mnist 또는 cifar를 입력합니다.
loaded_data = load_data(dataset)
n_class     = 10
width_size  = loaded_data['w']
height_size = loaded_data['h']
channel = int(loaded_data['c'])
batch_size = 200

trainingSize = loaded_data['tr_size']
validationSize = loaded_data['v_size']
testSize = loaded_data['te_size']

n_train_batches = trainingSize / batch_size
n_valid_batches = validationSize / batch_size
n_test_batches = testSize / batch_size
```

```
datasets: ['t_test', 't_train', 't_valid', 'x_test', 'x_train', 'x_valid']
size of each set (train: 50000, valid: 10000, test: 10000):
```

Preparing

```
x = tf.placeholder("float", shape=[batch_size, width_size, height_size, channel], name = 'X-input')
y_ = tf.placeholder("int32", shape=[batch_size], name = 'Y-input')
y_hot = dense_to_one_hot(y_, n_class)
```

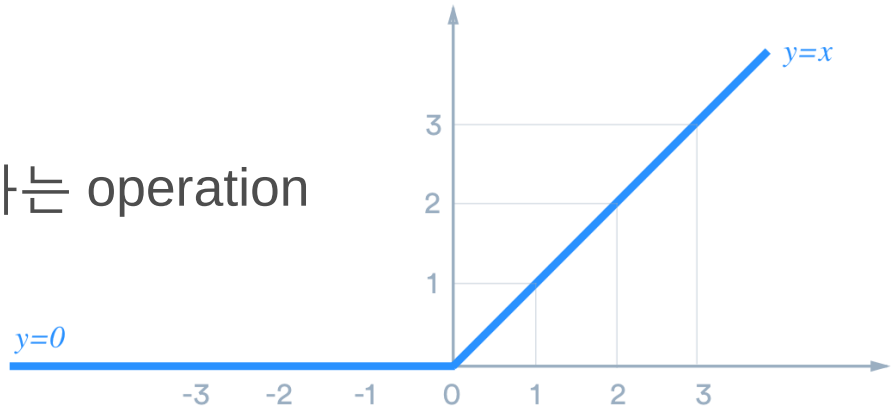
- Placeholder 선언
  - 이미지는 float형태, 레이블은 int32형태인 것에 주의
  - dtype은 string으로 인자를 줄 수도, tf.float32등의 형태로 인자를 줄 수도 있음

Modeling

```
# 모델을 정의합니다.
img_flat = tf.reshape(x, [batch_size, -1])
fc1, W_fc1, b_fc1 = linear(img_flat, 64, 'fc_1', with_w=True)
fc1 = tf.nn.relu(fc1)
fc2, W_fc2, b_fc2 = linear(fc1, 128, 'fc_2', with_w=True)
fc2 = tf.nn.relu(fc2)
fc3, W_fc3, b_fc3 = linear(fc2, 10, 'fc_3', with_w=True)
```

- tf.reshape(tensor, shape)
  - 주어진 tensor를 주어진 shape 대로 바꿔주는 operation
  - Shape는 차원정보를 담고 있는 list

- tf.nn.relu(features)
  - 주어진 features에 ReLU를 적용하는 operation

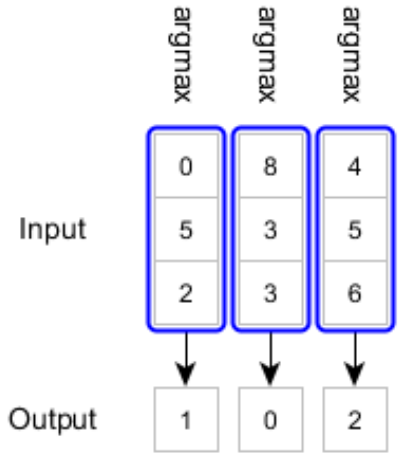


Modeling

- tf.nn.softmax(logits)
  - 주어진 logits의 softmax를 return하는 operation
  - Logits : 입력 데이터를 네트워크를 통해 전방향 진행하여 나온 결과물

- tf.clip_by_value(t, clip_value_min, clip_value_max)
  - 주어진 tensor t에서 clip_value_min 보다 작은 값을 clip_value_min으로, clip_value_max 보다 큰 값을 clip_value_max로 바꿔주는 operation

- tf.argmax(input, axis=None)
  - 주어진 axis를 따라 Input 중 가장 큰 value의 index를 return하는 operation
  - Ex.

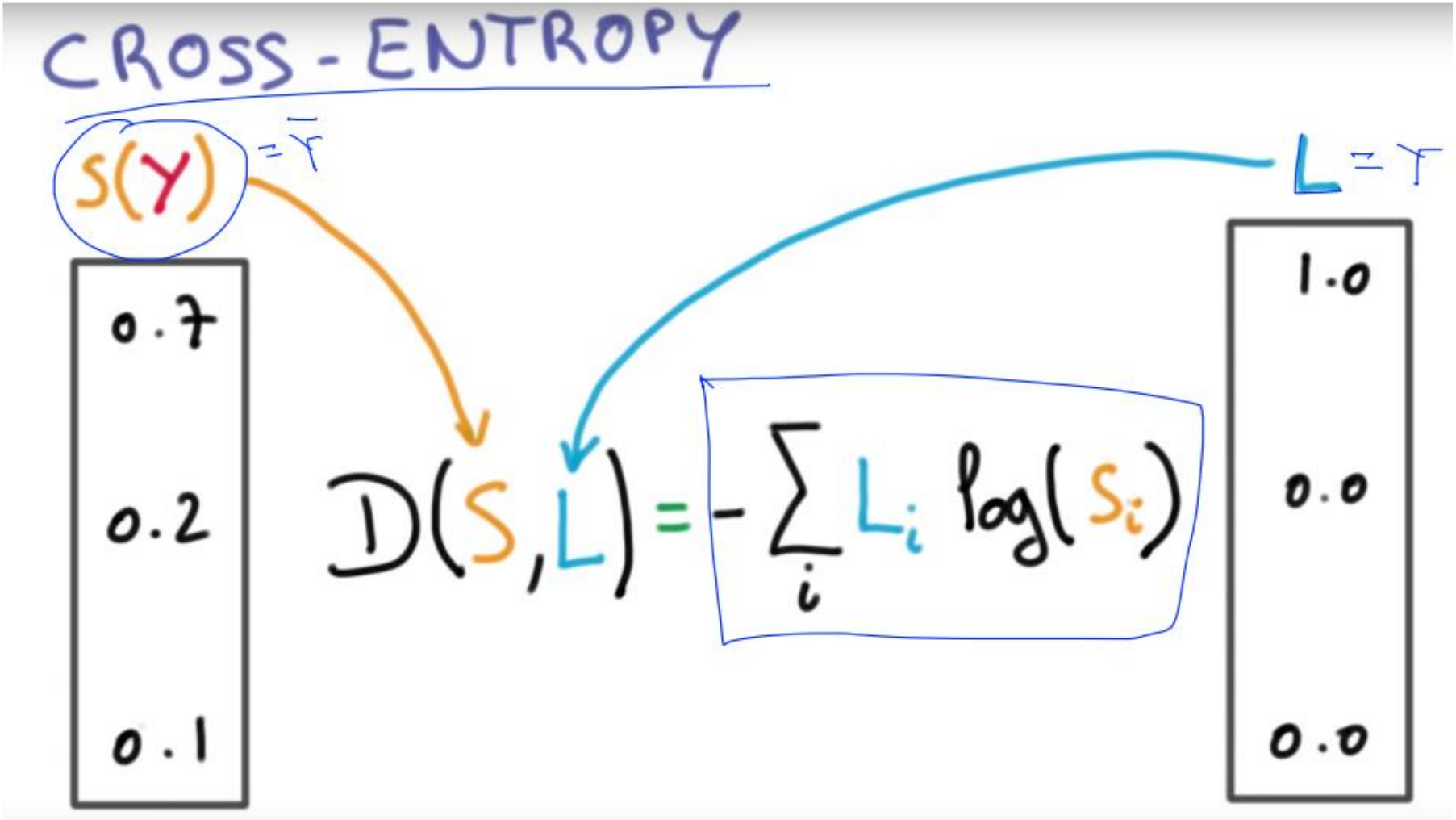


Modeling

- tf.equal(x, y)
  - x와 y를 element-wise로 비교하여 해당 위치에 같으면 1, 다르면 0이 할당된 tensor를 반환해주는 operation

- tf.cast(x, dtype)
  - 주어진 tensor를 주어진 dtype으로 변환시켜 reture해주는 operation

Cross-entropy



```python
output = tf.nn.softmax(fc3)

# name_scope는 텐서 보드에서 하나의 박스로 묶을 변수들을 정의합니다.
# 이는 복잡한 모델의 경우 수천개의 노드를 가지는데, 이를 다 표시하기 보다는
# 비슷한 유형끼리 하나의 덩어리로 묶고 박스 하나만 표시하는 것이 더욱 가독성이 좋기 때문입니다.
# 물론 박스를 펼쳐서 속에있는 내용을 자세하게 볼 수 있습니다.

with tf.name_scope("cost") as scope:
    cross_entropy = -tf.reduce_sum(y_hot*tf.log(tf.clip_by_value(output, 1e-10, 1.0)))
    loss = cross_entropy
    # 텐서 보드에 출력할 값으로 추가합니다.
    cost_summ = tf.summary.scalar("cost", loss)

with tf.name_scope("train") as scope:
    train_step = tf.train.AdamOptimizer(1e-3).minimize(loss)

    # 정답인지 아닌지에 대한 판단은 네트워크의 출력 값중
    # 가장 확률이 큰 뉴런의 인덱스와 타겟 값의 인덱스를 비교하는 방식입니다.
    correct_prediction = tf.equal(tf.argmax(output,1), tf.argmax(y_hot,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

    # 텐서 보드에 출력할 값으로 추가합니다.
    acc_summ = tf.summary.scalar("acc", accuracy)

# tensorboard --logdir=알맞은/폴더/경로 => 텐서보드 활성화
merged = tf.summary.merge_all()

saver = tf.train.Saver()
sess.run(tf.global_variables_initializer())
summary_writer = tf.summary.FileWriter("./logs/", sess.graph)
```

Learning

- 몇가지 선언된 값들
  - epoch_size : 전체 데이터를 몇번 학습시킬지 나타내는 숫자
  - hf : 학습에 사용될 데이터
  - complete_start_time : 학습이 시작되었을 때의 시각
    epoch_start_time : epoch이 시작되었을 때의 시각
  - avg_second : 각 epoch당 걸린 시간이 저장될 변수

  - accuracy_* : 각 데이터 유형(train, valid, test)별 정확도가 저장될 변수

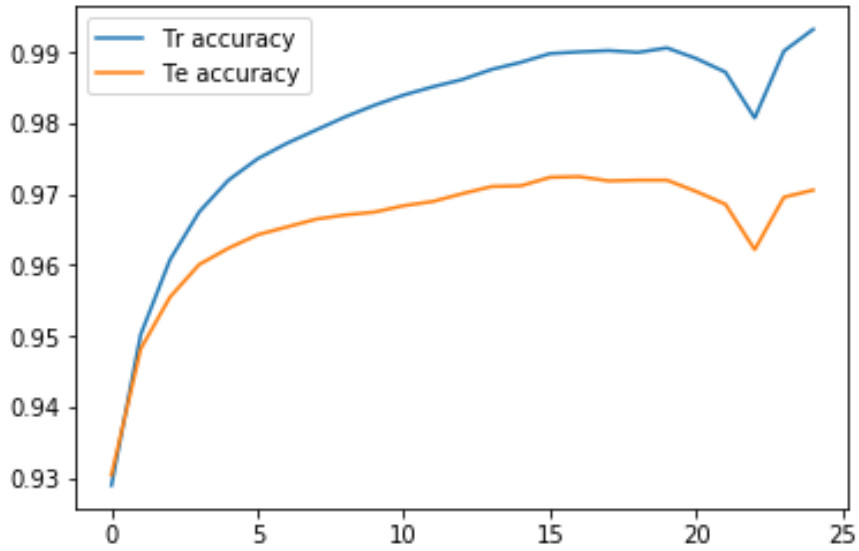
- time library
  - 시각에 관련된 함수들을 제공해주는 라이브러리
  - time.time()으로 현재 시각을 return 받을 수 있음(단위 : second)

Learning

- 학습 순서
  - 한 epoch 당 :
    - Accuracy 등의 변수 초기화
    - 한 batch 당 :
      - 해당 data batch 불러오기(load_batch 함수)
      - Train과 loss operation을 run 하고 train 과정의 cost를 저장
      - 해당 batch로 train accuracy를 평가
      - Batch idx 변경
      - 만약 일정 step에 도달했으면(여기는 10000 step 마다)
        - Train accuracy와 cost를 display
      - 만약 한 epoch이 끝났으면
        - 모델 저장
        - Tensorboard summary 저장
        - 각 데이터 유형에 대한 accuracy display
    - 학습 epoch에 걸리고 있는 시간 display
  - 전체 학습에 걸린 시간 display

```
## 모델 학습 루틴
epoch_size = 25
hf = loaded_data['data']
checkpoint_file = "./model/model.ckpt" #모델 저장 폴더

complete_start_time = time.time()
avg_second = 0.0

tr_accs = []
te_accs = []
for epoch in range(epoch_size):
    epoch_start_time = time.time()
    accuracy_train = 0
    accuracy_valid = 0
    accuracy_test = 0
    startIdx = 0
    endIdx = batch_size

    for minibatch_index in range(int(n_train_batches)):
        x_train, y_train = load_batch(dataset, hf, "train", startIdx, endIdx)
        _, cost= sess.run([train_step, loss], feed_dict={x: x_train, y_: y_train})
        accuracy_train = accuracy_train + accuracy.eval(feed_dict={x: x_train, y_: y_train})
        startIdx = endIdx
        endIdx = endIdx + batch_size
        if endIdx % 10000 == 0:
            print('%d/%d epoch, @iter = %d/%d, training accuracy : %.5f, cost : %.5f' %(epoch, epoch_size, endIdx, trainingSize,
                                                                                        accuracy_train/(minibatch_index + 1), cost))

        if minibatch_index == (n_train_batches - 1):
            summary_str = sess.run(merged, feed_dict={x: x_train, y_: y_train})
            summary_writer.add_summary(summary_str, epoch) # tensorboard 요약 추가
            summary_writer.flush()
            saver.save(sess, checkpoint_file) # 모델 저장

            train_accuracy = calc_Acc(hf, 'train')
            print('epoch %i, train accuracy %.5f' %(epoch, train_accuracy))
            tr_accs.append(train_accuracy)

            accuracy_valid = calc_Acc(hf, 'valid')
            print('epoch %i, valid accuracy %.5f' %(epoch, accuracy_valid))

            accuracy_test = calc_Acc(hf, 'test')
            print('epoch %i, test accuracy %.5f' %(epoch, accuracy_test))
            te_accs.append(accuracy_test)

    print("--- %s seconds per epoch ---" % (time.time() - epoch_start_time))
    avg_second = avg_second + (time.time() - epoch_start_time)
    print ("--- %s AVGseconds per epoch ---" % (avg_second/(epoch+1)))

print("--- %s seconds for complete ---" % (time.time() - complete_start_time))
```

```
plt.plot(tr_accs, label='Tr accuracy')
plt.plot(te_accs, label='Te accuracy')
plt.legend()
plt.show()

print("final train acc : ", tr_accs[-1])
print("final test acc : ", te_accs[-1])
```




```
final train acc :  0.9932800061702728
final test acc :  0.9706000018119813
```

Problem1

- 해당 learning model의 성능을 개선시킬 수 있는 방안을 한가지 이상 말하시오
- Code를 제출할 필요 없이 서술문만 제출