

[2018년 여름]

# Tensorflow를 이용한 Deep Learning의 기초

제2강: 이미지 분류를 위한 첫 번째 신경망



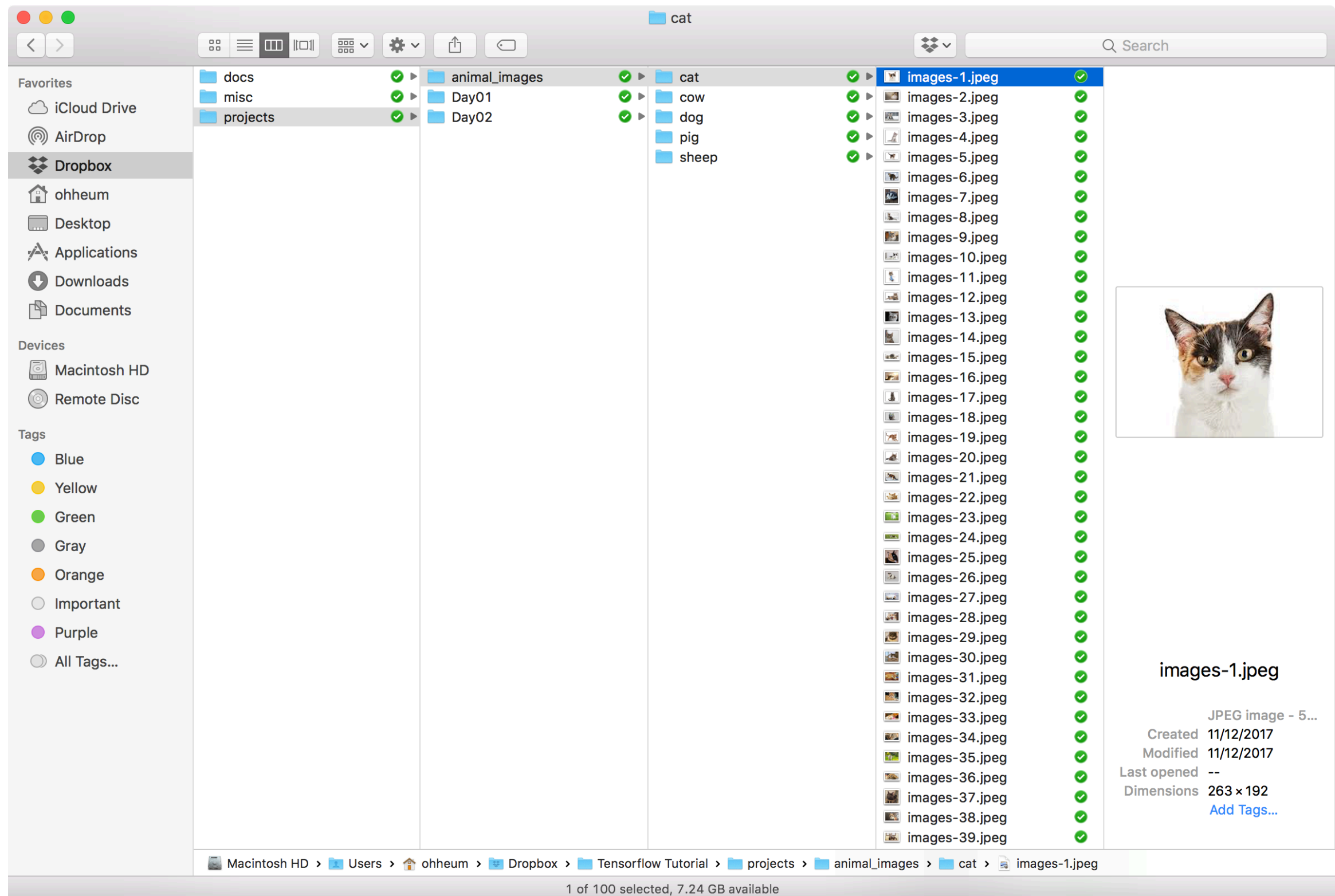
- 간단한 **image classification** 문제를 다룬다.
- Python에서 **이미지 파일을 읽고 다루는 방법들을 살펴본다.**
- Tensorflow를 이용하여 **기본적인 구조의 신경망을 구성해본다.**
  - 학습 데이터 전체를 메모리에 로드할 수 있을 정도의 규모에서 사용하는 방법
  - Fully (densely) connected layer로만 구성된 신경망

# Reading Image Files

## 이미지 파일 읽기

- Google 이미지 검색에서 cat, dog, sheep, cow, pig 각각 100장씩 이미지를 확보하였다. [여기서](#) 다운로드 한다.
- 압축을 풀면 동물 종류별로 5개의 서브 디렉토리를 가진 animal\_images 디렉토리가 나온다. (mac에서 만든 압축 파일이라서 압축을 해제하면 \_MACOSX라는 폴더가 생성되는데 그냥 삭제한다.)
- 이 디렉토리를 적절한 위치에 옮겨둔다. 이 튜토리얼에서 앞으로 만들 프로젝트들을 저장할 폴더를 적절한 위치에 만들고 그 안에 animal\_images 디렉토리를 옮겨둔다.

# 사용할 이미지 데이터의 준비



★ **Note:** The preferred way to feed data into a tensorflow program is using the `tf.data` API.

There are four methods of getting data into a TensorFlow program:

- `tf.data` API: Easily construct a complex input pipeline. (preferred method)
- Feeding: Python code provides the data when running each step.
- `QueueRunner` : a queue-based input pipeline reads the data from files at the beginning of a TensorFlow graph.
- Preloaded data: a constant or variable in the TensorFlow graph holds all the data (for small data sets).

강좌의 후반부에 다룰 예정이지만  
여기까지 진도를 나갈 수 있을지 잘 모르겠음

강좌의 전반부에는 이 방법을 사용



## • Handling Images

- **PIL (Python Image Library):** discontinued
- **Pillow:** PIL fork, actively maintained
- **OpenCV:** primarily for vision
- **Tensorflow** 자체도 image handling library를 제공

## • Matplotlib

- for various **plotting**

```
import numpy as np
import os
import cv2
```

```
paths = ['../animal_images/cat/images-1.jpeg', '../animal_images/cat/images-2.jpeg']
```

```
imgs = [cv2.imread(path) for path in paths]
```

← imgs는 이미지들의 list이다.

```
img = imgs[0]
```

← img는 numpy array이다.  
PyCharm Debugger로 확인해보자.

```
print(img.ndim)
print(img.shape)
print(img.dtype)
```

← numpy array가 가진 기본 attribute들을 확인해 본다.

```
print(img)
print(img.tolist())
```

- ▶ opencv를 이용하여 이미지를 프로그램 내로 load하고 간단한 조작을 하는 것을 연습한다.
- ▶ 프로그램 내에서 이미지는 numpy array로 저장된다. numpy array의 기본적인 사용법을 python list와 비교해서 어느정도 습득해야 한다.

- ▶ cv2.imread() 함수의 2번째 매개변수로 다음 중 하나를 지정할 수 있다.
  - cv2.IMREAD\_COLOR: default flag
  - cv2.IMREAD\_GRAYSCALE: grayscale mode
  - cv2.IMREAD\_UNCHANGED: include alpha channel
- ▶ 모드를 바꿔가면서 테스트해본다.



- ▶ 이미지를 화면에 display하고, resize하고, 이미지 파일로 저장하고, color space를 변환하는 일을 해본다.

```
cv2.imshow('Test Image', img)
cv2.waitKey(0) & 0xFF
cv2.destroyAllWindows()
```

← 이미지를 화면에 display한다. 아무 키나 누르면 사라진다.

```
img = cv2.resize(img, (200, 200), interpolation=cv2.INTER_CUBIC) # resize
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # cv2 Load images as BGR, convert it to RGB
```

```
cv2.imwrite('converted.jpg', img)
```

← 이미지를 파일로 저장한다.

1. <https://www.youtube.com/watch?v=rN0TREj8G7U>
2. <https://www.youtube.com/watch?v=a8aDcLk4vRc&list=PLeo1K3hjS3uset9zIVzJWqplaWBiacTEU&index=2>
3. [https://www.youtube.com/watch?v=d\\_Ka-ks2a0&list=PLeo1K3hjS3uset9zIVzJWqplaWBiacTEU&index=3](https://www.youtube.com/watch?v=d_Ka-ks2a0&list=PLeo1K3hjS3uset9zIVzJWqplaWBiacTEU&index=3)
4. <https://www.youtube.com/watch?v=XawR6CjAYV4&list=PLeo1K3hjS3uset9zIVzJWqplaWBiacTEU&index=4>

data visualization package

data plotting module

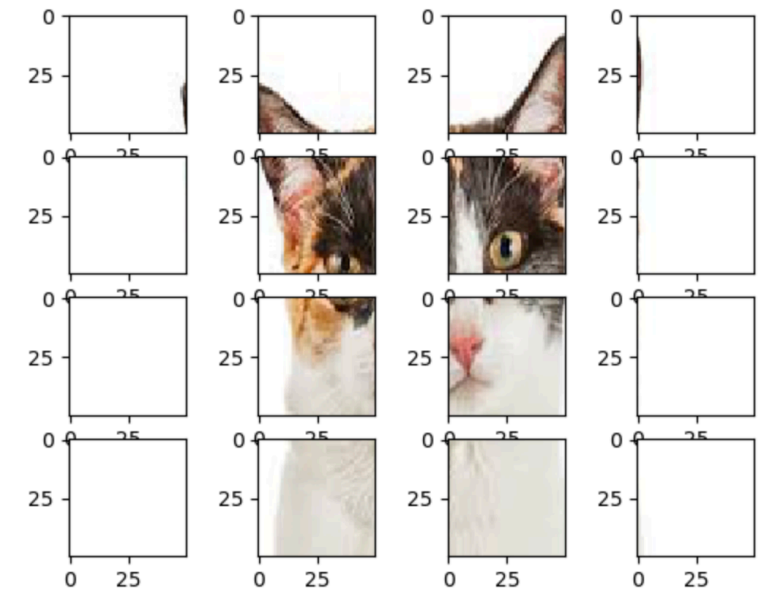
from matplotlib import pyplot as plt

plt.imshow(img)  
plt.show()

matplotlib 패키지가 설치되어 있지 않으면 이 부분에서 오류가 난다.  
커서를 이 부분에 위치시키고 Alt-Enter를 누르면 “install package matplotlib”라는 메뉴가 생기는데 이것을 실행한다.  
혹은 Anaconda Prompt를 실행하고 다음과 같이 설치해도 된다.  
\$ activate tfenv3.5  
\$ conda install matplotlib

```
def plot_images(image):
    fig, axes = plt.subplots(4, 4)
    for i, ax in enumerate(axes.flat):
        row = i // 4
        col = i % 4
        image_frag = image[row*50:(row+1)*50, col*50:(col+1)*50, :]
        ax.imshow(image_frag)
    plt.show()
```

plot\_images(img) ← 이 함수를 호출해 본다.



↑  
이미지를 가로 세로 4조각으로  
분할(slicing)해본다.

- ▶ Python의 enumerate 함수를 연습해보자.
- ▶ Numpy array를 어떻게 slicing하는지 살펴본다.

```
img_reshaped = img.reshape(100, 400, 3)
```

```
plt.imshow(img_reshaped)  
plt.show()
```

```
flattened_image = img.ravel()
```

```
img_f32 = np.float32(img)
```

```
normalized_img32 = img_f32/255.
```

```
zero_centered_img = (img_f32 - np.mean(img_f32))/np.std(img_f32)
```

```
print(zero_centered_img)
```

- `cv2.imread()`함수에서 이미지를 grayscale로 읽어들이어서 프로그램 내부에서 어떻게 표현되는지 살펴본다.
- `code01.py`에서 읽은 이미지의 일부를 crop하여 display하거나 이미지 파일로 저장해 본다.
- 이미지를 채널별로 따로 분리하여 display하거나 저장해본다.

# Preparing Training Data

## 학습 데이터 준비하기



code02.py에서는 준비한 모든 이미지들을 프로그램으로 읽어와서 저장하고, 배치 단위로 분할하는 일을 해본다.

```
import numpy as np
import os
import cv2
```

```
IMG_HEIGHT = 60
IMG_WIDTH = 60
NUM_CHANNEL = 3
NUM_CLASS = 5
```

← 이미지 크기는 60\*60으로 resize해서 사용할 계획이다.  
NUM\_CLASS는 분류할 동물 종류의 가지 수이다.

```
def load_image(addr):
    img = cv2.imread(addr)
    img = cv2.resize(img, (IMG_HEIGHT, IMG_WIDTH), interpolation=cv2.INTER_CUBIC)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
    img = img.astype(np.float32)
```

← unit8 타입의 데이터를 float32 타입으로 변환하는 다른 방법  
(preferred method)

```
    img = (img - np.mean(img))/np.std(img)
    return img
```

```
IMAGE_DIR_BASE = '../animal_images'
```

```
image_dir_list = os.listdir(IMAGE_DIR_BASE)
```

```
print(image_dir_list)
```

← os.listdir 함수를 이용하여 디렉토리에 속한  
파일과 서브 디렉토리의 목록을 읽는다.

glob 패키지를 이용하면 와일드 카드를 사용하여 좀 더 편리하게 파일 목록을 얻을 수 있다.

- ▶ import glob
- ▶ file\_names = glob.glob('../animal\_images/cat/\*.jpg');

- ▶ 보통 데이터(우리의 경우 동물 이미지)를 feature, 정답(분류)을 label이라고 부른다.
- ▶ 모든 이미지들을 하나의 Python list로 저장하고, 각 이미지의 라벨은 0(cat), 1(cow), 2(dog), 3(pig), 4(sheep)으로 표시하여, feature에 이미지가 저장된 순서대로 labels 리스트에 저장한다.

```
features = []  
labels = []
```

← features와 labels는 empty Python list이다.  
Python list가 numpy array보다 조작이 쉬운 면이 있다.

```
for cls_index, dir_name in enumerate(image_dir_list):  
    image_file_list = os.listdir(IMAGE_DIR_BASE + os.sep + dir_name)  
    for file_name in image_file_list:  
        image = load_image(IMAGE_DIR_BASE + os.sep + dir_name + os.sep + file_name)
```

```
        features.append(image.ravel())  
        labels.append(cls_index)
```

← ravel()은 다차원 배열을 1차원 배열로 변환

← append는 Python 리스트에 새로운 값을 추가한다.

```
print(len(features))    # just for test
```

```
from random import shuffle
```

```
c = list(zip(features, labels))
shuffle(c)
features, labels = zip(*c)
```

```
features = np.array(features)
labels = np.array(labels)
```



features와 labels는 Python tuple이다. np array로 변환해준다.

```
print(len(labels))
print(labels)
print(labels.shape)
```

```
image = features[0]
image = image.reshape((IMG_HEIGHT, IMG_WIDTH, NUM_CHANNEL))
image = image.astype(np.uint8)
cv2.imshow('Restored Image', image)
cv2.waitKey(0) & 0xFF
cv2.destroyAllWindows()
```

#### Python built-in list와 zip 함수

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> print(zipped)
<zip object at 0x10d9d9f88>
>>> zipped_list = list(zipped)
>>> print(zipped_list)
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zipped_list)
>>> print(x == list(x2) and y == list(y2))
True
```

← 제대로 되고 있는지  
확인하기 위해서  
이런 저런 테스트를 해본다.

- ▶ 데이터를 training set과 test set으로 분할하였다. 8:2의 비율이 일반적으로 사용된다.
- ▶ Validation data set은 언제까지 training을 지속할지 결정하기 위해서 사용되며, validation data를 사용할 경우 보통 6:2:2의 비율로 분할한다.
- ▶ 우리는 당분간 validation data를 사용하지 않는다.

```
train_features = features[0:int(0.8 * len(features))]  
train_labels = labels[0:int(0.8 * len(labels))]
```

```
# val_features = features[int(0.6 * len(features)):int(0.8 * len(features))]  
# val_labels = labels[int(0.6 * len(features)):int(0.8 * len(features))]
```

```
test_features = features[int(0.8 * len(features)):]  
test_labels = labels[int(0.8 * len(labels)):]
```

```
import math

def is_prime(number):
    if number <= 1:
        return False
    if number == 2:
        return True
    if number % 2 == 0:
        return False
    for div in range(3, int(math.sqrt(number) + 1), 2):
        if number % div == 0:
            return False
    return True
```

```
def get_primes(number):
    while True:
        if is_prime(number):
            yield number
        number += 1
```

```
prime_iterator = get_primes(1)
```

```
for _ in range(100):
```

```
    next_prime_number = next(prime_iterator)
```

```
    print(next_prime_number)
```

- ▶ Python generator는 generator iterator를 반환하는 함수이다.
- ▶ Generator iterator에 대해서 next()가 함수가 호출될 때 함수의 시작부터가 아니라 이전 호출에서 yield를 실행한 다음부터 실행이 재개되어 다음 yield가 실행될 때 까지 진행된다.
- ▶ 반복된 호출에서 함수의 상태(지역변수의 값)는 보존된다.

매개변수 number보다 크거나 같은 모든 소수를  
순서대로 생성해주는 generator

▶ Rearranging numpy array based on index

```
>> arr = np.array([10, 20, 30, 40, 50])
```

```
>> idx = [1, 0, 3, 4, 2]
```

```
>> arr[idx]
```

```
array([20, 10, 40, 50, 30])
```

```
BATCH_SIZE = 50
```

```
def train_data_iterator():
```

```
    while True:
```

매 epoch마다 새로 shuffling한다.



```
        idxs = np.arange(0, len(train_features))
        np.random.shuffle(idxs)
        shuf_features = train_features[idxs]
        shuf_labels = train_labels[idxs]
```

```
        batch_size = BATCH_SIZE
```

```
        for batch_idx in range(0, len(train_features), batch_size):
            images_batch = shuf_features[batch_idx:batch_idx+batch_size]
            labels_batch = shuf_labels[batch_idx:batch_idx+batch_size]
            yield images_batch, labels_batch
```



```
iter_ = train_data_iterator() ← generator를 호출하면 iterator를 반환한다.  
for step in range(100):
```

```
    # get a batch of data
```

```
    images_batch_val, labels_batch_val = next(iter_)
```

```
    print(images_batch_val)
```

```
    print(labels_batch_val)
```

↑  
iterator에 대해서 next함수를 실행할 때마다  
하나의 배치가 반환된다.

# **Implementing Image Classification Network**

## **이미지 분류 네트워크 구현**

$x : 1 \times 10800$  vector

$y : 1 \times 5$  vector

$W_1 : 10800 \times 1024$  matrix

$b_1 : 1 \times 1024$  vector

$fc_1 = S(xW_1 + b_1)$

$W_2 : 1025 \times 5$  matrix

$b_2 : 1 \times 5$  vector

$y_- = fc_1 \cdot W_2 + b_2$

Let  $x = [123, 8, 67, 12, \dots, 98]$

Then  $y_- = ?$

- ▶ 고등학교때의 기억을 되살려보면 이런 식의 symbolic representation이 procedural한 코드보다 우리에게 훨씬 익숙한 방식이다.
- ▶ 먼저 변수들 간의 관계를 수식 혹은 함수로 기술하고, 계산은 나중에 한다.
- ▶ Tensorflow는 이런 방식의 symbolic representation을 사용한다.

# Symbolic Representation of Computation

$x : 1 \times 10800$  vector

$y : 1 \times 5$  vector

$W_1 : 10800 \times 1024$  matrix

$b_1 : 1 \times 1024$  vector

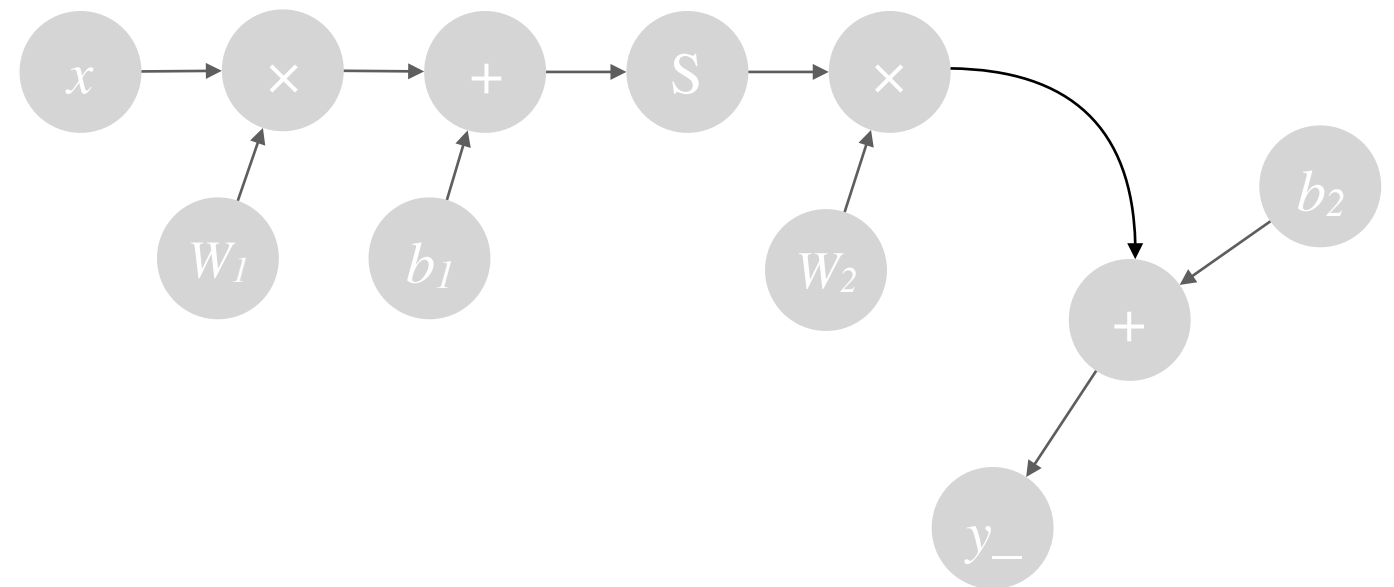
$fc_1 = S(xW_1 + b_1)$

$W_2 : 1025 \times 5$  matrix

$b_2 : 1 \times 5$  vector

$y_- = fc_1 \cdot W_2 + b_2$

이 부분에서는 실제로 어떤 계산도 일어나지 않는다.  
아래 그림과 같은 그래프를 구성할 뿐이다.



Let  $x = [123, 8, 67, 12, \dots, 98]$  ← 여기서  $x$ 의 값이 제공된다.

Then  $y_- = ?$  ←  $y_-$ 의 값을 물어볼때 드디어 위의 그래프가 순서대로 계산된다.

# Symbolic Representation of Computation

$x : 1 \times 10800$  vector  
 $y : 1 \times 5$  vector

나중에 실제로 제공될 입력 데이터가  
들어갈 자리를 placeholder라고 부른다.

$W_1 : 10800 \times 1024$  matrix  
 $b_1 : 1 \times 1024$  vector  
 $W_2 : 1025 \times 5$  matrix  
 $b_2 : 1 \times 5$  vector

Weight와 bias가 저장될 변수들을 그냥 변수(variable) 혹은  
model variable이라고 부른다.

$fc_1 = S(xW_1 + b_1)$   
 $y_- = fc_1 \cdot W_2 + b_2$

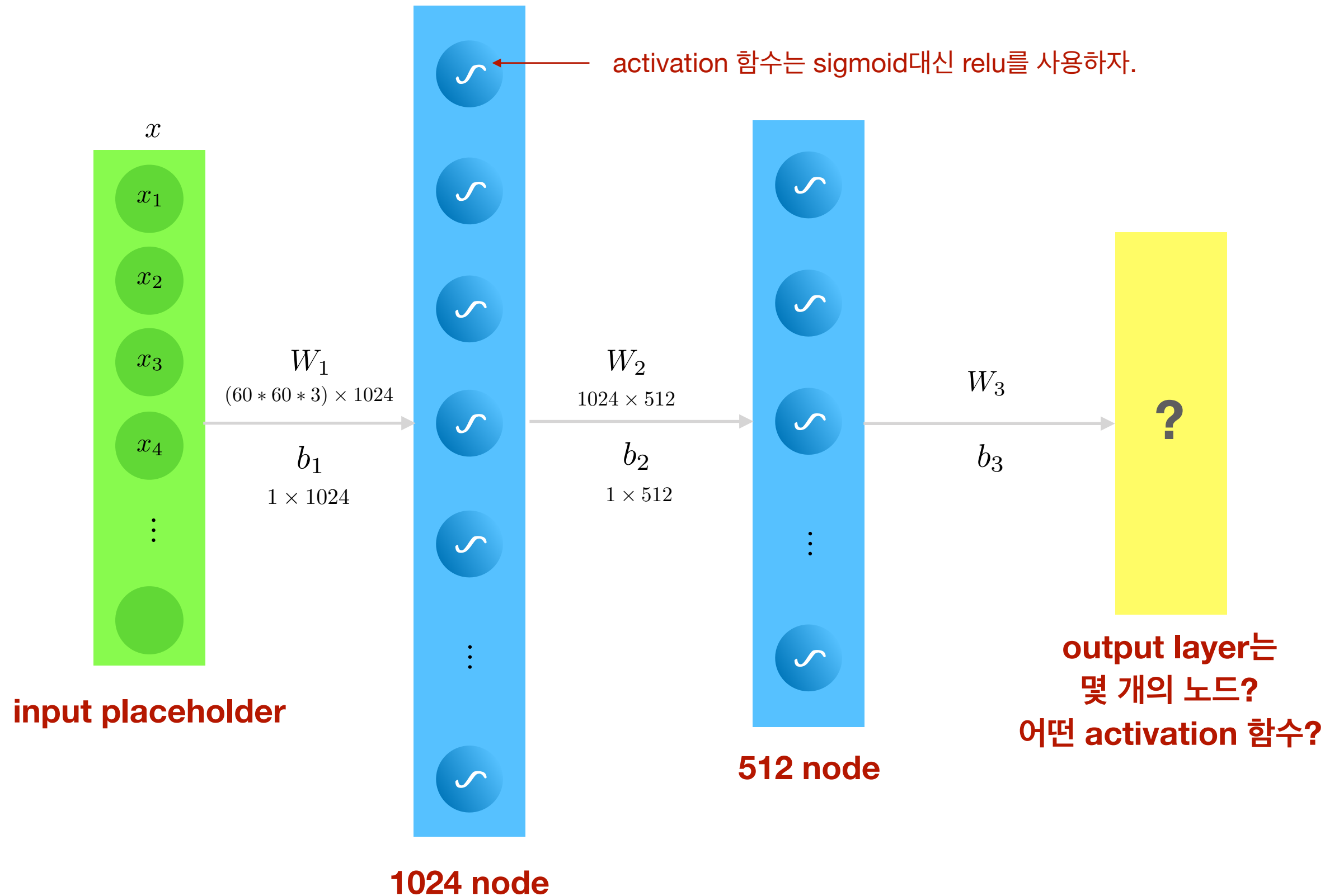
일어나야할 계산들을 표현한다. Placeholder와 Variable을 노드로 하는  
하나의 그래프로 표현된다.

실제 계산을 하기전에 이 부분에서 Session을 생성해야 한다.  
Session은 실제 계산을 수행하는 프로세스 정도로 이해하고 넘어가자.

Let  $x = [123, 8, 67, 12, \dots, 98]$   
Then  $y_- = ?$

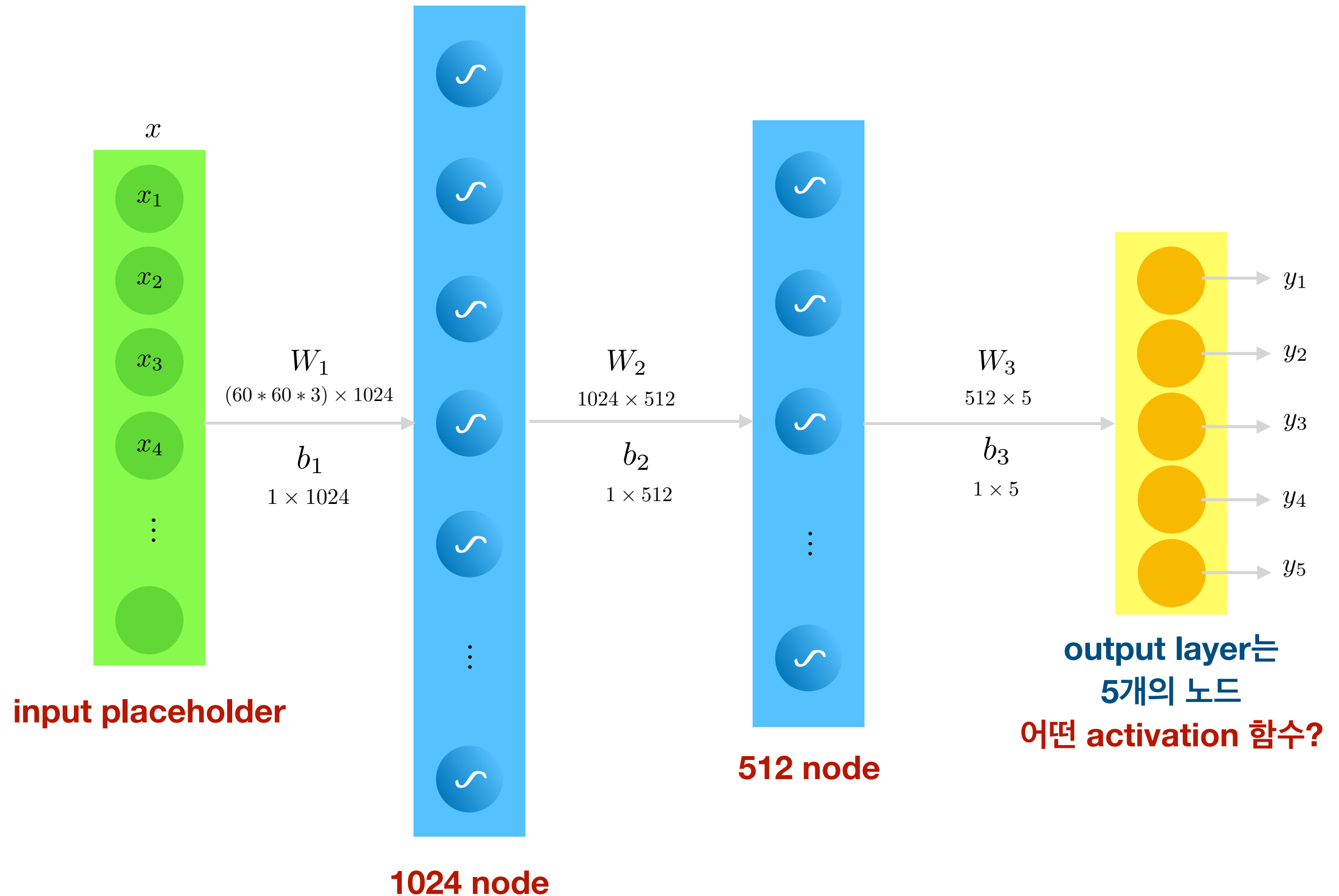
Placeholder에 입력 데이터를 제공하고,  
내가 알고싶은 값을 지정하면서  
session을 run하면 계산 결과를 fetch다.

- **Placeholder**: 그래프에 입력(features, labels 등)을 공급하는 장소로 사용됨
- **Model variable**: 가중치(weight)와 바이어스(bias) 값을 저장
- Placeholder의 입력과 변수로 부터 계산하는 연산(**operations**)
- **Cost (loss) 함수**: 정답(label)과 실제 출력의 차이를 표현하는 오류 함수
- 최적화 알고리즘(**optimization method**): Gradient descent 알고리즘으로 변수의 값을 갱신





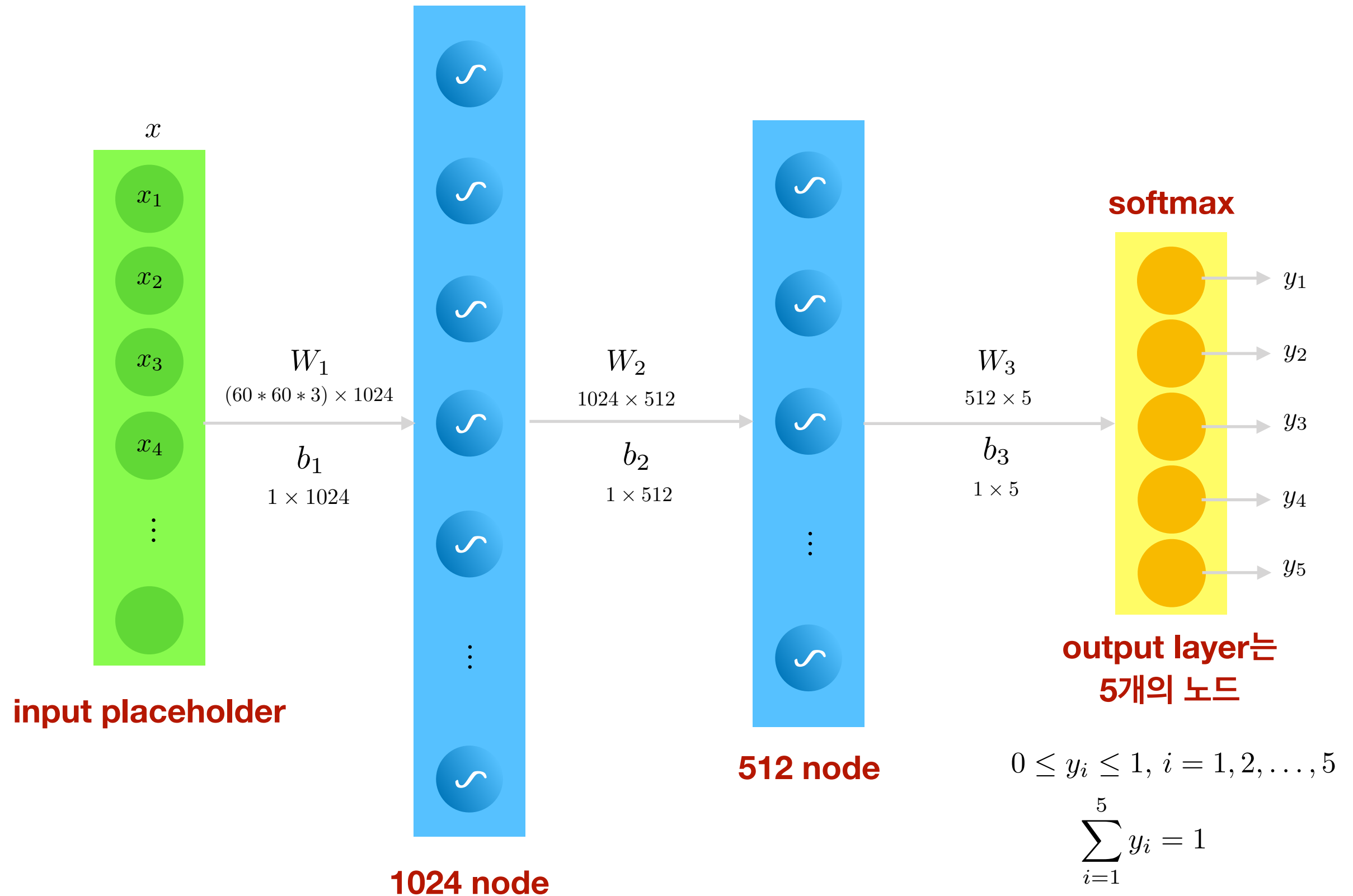
- 이미지 classification 문제에서는 label과 네트워크의 출력을 **one-hot encoding**으로 표현하는 것이 일반적이다.
- 클래스의 개수가 k개일 때 각각의 이미지에 대해서 라벨과 출력을 k-차원 벡터로 표현한다. 해당 이미지가 소속된 클래스에 해당하는 하나의 값만 1이고 나머지는 모두 0이다.
  - 우리의 예에서 클래스의 개수는 5개이고, 순서는 cat, cow, dog, pig, sheep라고 하자.
  - cat은 [1, 0, 0, 0, 0]으로, cow는 [0, 1, 0, 0, 0], dog는 [0, 0, 1, 0, 0], pig는 [0, 0, 0, 1, 0], 그리고 sheep는 [0, 0, 0, 0, 1]로 표현한다.
- **그냥 0, 1, 2, 3, 4와 같이 하나의 스칼라 값으로 표현하는 것이 적절하지 않은 이유는?**
- 우리의 경우 label은 one-hot-encoding으로 표현되어 있지 않다.



- classification 문제에서 출력 벡터 (우리의 경우 5차원 벡터)는 해당 이미지가 **각 클래스에 속할 확률**을 표현하면 자연스럽다.
- 임의의 K차원 벡터  $z = [z_1, z_2, \dots, z_K]$ 를 크기 순서를 유지하면서 각 원소가 0 ~ 1범위에 들어가고 합이 1이 되도록 변환해주는 함수가 softmax 함수이다.

$$\sigma : \mathbb{R}^K \rightarrow [0, 1]^K$$
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

- softmax함수는 classification 문제의 출력층에서 가장 흔히 사용되는 activation 함수이다.



# Constructing Graph: Placeholder

- ▶ 이미지는 `images_batch`라는 이름의 placeholder를 통해서 네트워크에 공급한다.
- ▶ 라벨은 `labels_batch`라는 이름의 placeholder를 통해서 네트워크에 공급한다.
- ▶ 각각의 이미지는 `IMG_HEIGHT*IMG_WIDTH*NUM_CHANNEL` 크기의 1차원 벡터로 표현된다.

```
import tensorflow as tf
```

입력 이미지가 제공될 placeholder의 크기는 `batch_size * image_size`이다.  
`None`은 나중에 실제로 제공되는 데이터의 크기에 맞춘다는 의미이다.

```
images_batch = tf.placeholder(dtype=tf.float32,  
                             shape=[None, IMG_HEIGHT*IMG_WIDTH*NUM_CHANNEL])
```

```
labels_batch = tf.placeholder(dtype=tf.int32, shape=[None, ])
```

크기가 미정인 1차원 벡터의 shape은 이렇게 지정한다.  
(이렇게 하면 label이 one-hot-encoding이 아님을 기억하자.)

## tf.placeholder

```
placeholder(  
    dtype,  
    shape=None,  
    name=None  
)
```

Defined in `tensorflow/python/ops/array_ops.py`.

See the guides: [Inputs and Readers > Placeholders](#), [Reading data > Feeding](#)

Inserts a placeholder for a tensor that will be always fed.

**Important:** This tensor will produce an error if evaluated. Its value must be fed using the `feed_dict` optional argument to `Session.run()`, `Tensor.eval()`, or `Operation.run()`.

# Constructing Graph: First Hidden Layer

이렇게 변수의 이름을 지정해 둔다.

weight 행렬의 크기는 “이전 layer의 노드 개수 \* 현재 layer의 노드 개수”이다.

```
w1 = tf.get_variable("w1", [IMG_HEIGHT*IMG_WIDTH*NUM_CHANNEL, 1024])
b1 = tf.get_variable("b1", [1024])
```

```
fc1 = tf.nn.relu( tf.matmul(images_batch, w1) + b1 )
```

activation 함수로는 sigmoid대신 relu를 적용한다.

images\_batch\*w1은 [batch\_size, 1024] 크기의 행렬이고,  
b1은 [1024] 크기의 1차원 벡터이다. 행렬의 각 행에 b1이 더해진다.  
이것을 numpy에서는 broadcasting이라고 부른다.

Layer1의 출력 벡터이다.

▶ Relu (Rectified Linear Unit)은 대표적인 activation 함수이며 다음과 같이 정의된다:

$$relu(x) = \max(0, x)$$

▶ Relu는 sigmoid보다 vanishing gradient 문제에 대해서 효과적이다.



## tf.get\_variable

```
get_variable(  
    name,  
    shape=None,  
    dtype=None,  
    initializer=None,  
    regularizer=None,  
    trainable=True,  
    collections=None,  
    caching_device=None,  
    partitioner=None,  
    validate_shape=True,  
    use_resource=None,  
    custom_getter=None,  
    constraint=None  
)
```

## Constructing Graph: Second and Output Layer

```
w2 = tf.get_variable("w2", [1024, 512])  
b2 = tf.get_variable("b2", [512])
```

이전 layer의 출력이 이 layer의 입력이 된다.

↓

```
fc2 = tf.nn.relu(tf.matmul(fc1, w2) + b2)
```

```
w3 = tf.get_variable("w3", [512, NUM_CLASS])  
b3 = tf.get_variable("b3", [NUM_CLASS])  
y_pred = tf.matmul(fc2, w3) + b3
```



출력 layer에는 아직 activation 함수를 적용하지 않았다.

y\_pred는 임의의 실수를 원소로 가지는 batch\_size \* NUM\_CLASS 크기의 행렬이다.

# Constructing Graph: Loss Function

y\_pred는 아직 softmax 함수를 적용하기 전이고,  
labels\_batch는 one-hot-encoding이 아니다. 이런 상황에서 필요한 모든 일을 대신 처리해주는  
함수가 tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits이다.

loss = **tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits**(logits=y\_pred,  
labels=labels\_batch)

만약 labels\_batch가 one-hot-encoding되어 있다면  
softmax\_cross\_entropy\_with\_logits 함수를 대신 사용하면 된다.

Loss 함수로 지난 주 이론 수업에서 설명한 mean-squared error  
대신 cross entropy함수를 사용하였다. 본질적인 의미는  
동일하다.

loss\_mean = tf.reduce\_mean(loss)

loss는 각 이미지별로 따로 계산되므로  
1×batch\_size 크기의  
벡터이다. 여기에서 평균 loss를 계산한다.

- ▶ Cross entropy 함수는 다음과 같이 정의된다. (  $\bar{y}$  를 one-hot- encoded label,  $y$ 를 출력이라고 하자.  $y$ 는 각 클래스에 속할 확률을 나타내는 벡터의 형태를 가진다. 즉, softmax의 출력이다.)

$$\text{cross\_entropy}(y, \bar{y}) = - \sum \bar{y} \log y$$

- ▶ Cross entropy 함수는 MSE보다 learning 속도 측면에서 효율적이다.
- ▶ 그냥 squared error sum을 사용하려면 다음과 같이 하면 된다. 단, label은 먼저 one-hot-encoding되어야 한다.

```
prediction = tf.nn.softmax(y_pred)
loss_mean = tf.losses.mean_squared_error(label, prediction)
```

## tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits

```
sparse_softmax_cross_entropy_with_logits(  
    _sentinel=None,  
    labels=None,  
    logits=None,  
    name=None  
)
```

Computes sparse softmax cross entropy between `logits` and `labels`.

Measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class). For example, each CIFAR-10 image is labeled with one and only one label: an image can be a dog or a truck, but not both.

**NOTE:** For this operation, the probability of a given label is considered exclusive. That is, soft classes are not allowed, and the `labels` vector must provide a single specific index for the true class for each row of `logits` (each minibatch entry). For soft softmax classification with a probability distribution for each entry, see `softmax_cross_entropy_with_logits`.

**WARNING:** This op expects unscaled logits, since it performs a `softmax` on `logits` internally for efficiency. Do not call this op with the output of `softmax`, as it will produce incorrect results.

A common use case is to have logits of shape `[batch_size, num_classes]` and labels of shape `[batch_size]`. But higher dimensions are supported.

**Note that to avoid confusion, it is required to pass only named arguments to this function.**

# Constructing Graph: Backpropagation Algorithm

- ▶ 지금까지 정의한 모든 노드(텐서)들이 결국 어떤 “값”을 표현 한다면 train\_op는 “알고리즘(operation)”을 표현한다.
- ▶ train\_op 타겟을 run하면 backpropagation이 이루어진다.

```
train_op = tf.train.AdamOptimizer().minimize(loss_mean)
```

↑  
loss값을 최소화하는 backpropagation을 실행할 optimizer이다.  
역시 지난 주 설명한 gradient descent method 대신 AdamOptimizer를 사용했다.  
AdamOptimizer는 gradient descent method에 learning rate ( $\rho$ )를 adaptive하게 자동 조절하는 기능이 추가된 버전이라고 생각하면 된다.

- ▶ 기본적인 gradient descent method를 쓰려면 다음과 같이 한다. 이 경우 learning rate ( $\rho$ )를 지정해야 한다.

```
train_op = tf.train.GradientDescentOptimizer(0.0001).minimize(cost)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

← 지금까지 구성한 그래프를 실행할 session을 생성하고  
변수들을 초기화한다.

```
iter_ = train_data_iterator()
for step in range(500):
```

```
    images_batch_val, labels_batch_val = next(iter_)
```

```
    _, loss_val = sess.run([train_op, loss_mean], ← 이렇게 실행할 타겟을 지정한다.
```

```
        feed_dict={
            images_batch:images_batch_val,
            labels_batch:labels_batch_val
        })
```

← Python dictionary의 형태로  
placeholder에 제공할  
데이터를 공급한다.

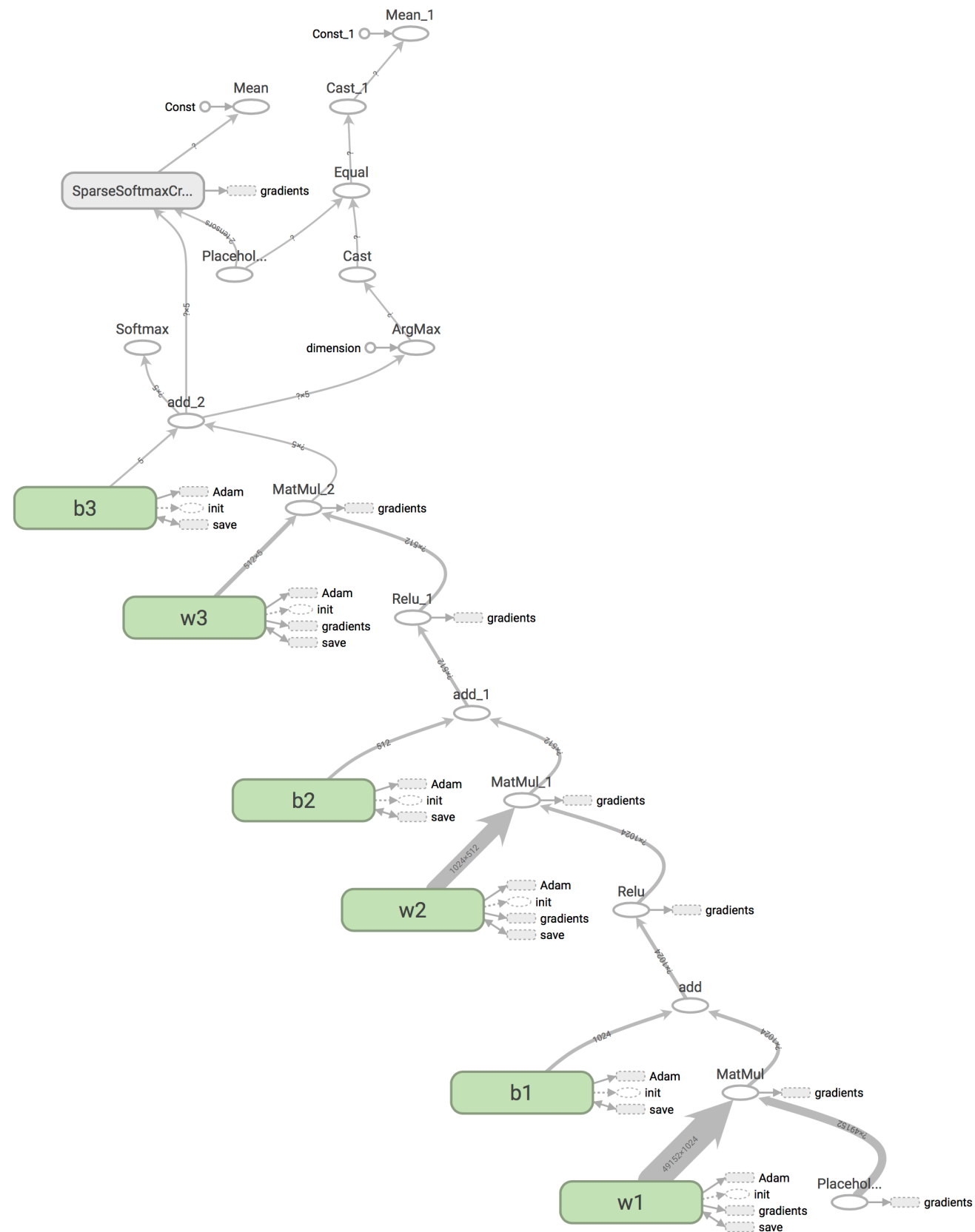
```
    print(loss_val)
```

```
print('Training Finished....')
```

```
print('Test begins...')
TEST_BSIZE = 50
for i in range(int(len(test_features)/TEST_BSIZE)):

    images_batch_val = test_features[i*TEST_BSIZE:(i+1)*TEST_BSIZE]
    labels_batch_val = test_labels[i*TEST_BSIZE:(i+1)*TEST_BSIZE]

    loss_val = sess.run([loss_mean], feed_dict={
        images_batch:images_batch_val,
        labels_batch:labels_batch_val
    })
    print(loss_val)
```





- 현재의 코드는 loss값만을 출력한다. 뭔가 되고 있기는 하지만 얼마나 잘 되고 있는지는 알 수 없다. Test에서 정확도(적중율, accuracy)를 출력하려면 어떻게 해야 할까?
- 노드 개수, 이미지 크기 등을 바꿔가면서 정확도를 개선해본다.