실습코드

11-1

```
# Lab 11 MNIST and Convolutional Neural Network
import tensorflow as tf
import random
# import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# hyper parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 10])

# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv     -> (?, 28, 28, 32)
#    Pool     -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
'''
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
```

Conv2d와 max_pool로 conv layer와 pooling을 구현함

```
    Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
    '''

    # L2 ImgIn shape=(?, 14, 14, 32)
    W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
    #    Conv      ->(?, 14, 14, 64)
    #    Pool      ->(?, 7, 7, 64)
    L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
    L2 = tf.nn.relu(L2)
    L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
    L2_flat = tf.reshape(L2, [-1, 7 * 7 * 64])
    '''
    Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
    Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
    Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
    Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)
    '''

    # Final FC 7x7x64 inputs -> 10 outputs
    W3 = tf.get_variable("W3", shape=[7 * 7 * 64, 10],
                         initializer=tf.contrib.layers.xavier_initializer())
    b = tf.Variable(tf.random_normal([10]))
    logits = tf.matmul(L2_flat, W3) + b

    # define cost/loss & optimizer
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
        logits=logits, labels=Y))
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

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

층 구현하고 cost function 설정

Sess는 현재 tensorflow버전에서는 삭제된 기능. 디폴트로 그 기능이 구현된다.

```
    # train my model
    print('Learning started. It takes sometime.')
    for epoch in range(training_epochs):
        avg_cost = 0
        total_batch = int(mnist.train.num_examples / batch_size)

        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            feed_dict = {X: batch_xs, Y: batch_ys}
            c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
            avg_cost += c / total_batch

        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

    print('Learning Finished!')

    # Test model and check accuracy
    correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print('Accuracy:', sess.run(accuracy, feed_dict={
          X: mnist.test.images, Y: mnist.test.labels}))

    # Get one and predict
    r = random.randint(0, mnist.test.num_examples - 1)
    print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
    print("Prediction: ", sess.run(
        tf.argmax(logits, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

    # plt.imshow(mnist.test.images[r:r + 1].
    #           reshape(28, 28), cmap='Greys', interpolation='nearest')
    # plt.show()

```

11-2

```python
# Lab 11 MNIST and Deep learning CNN
import tensorflow as tf
import random
# import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# hyper parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# dropout (keep_prob) rate  0.7~0.5 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 10])

# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv     -> (?, 28, 28, 32)
#    Pool     -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
```

```
    L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
    L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
    '''
    Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
    Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
    Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
    Tensor("dropout/mul:0", shape=(?, 14, 14, 32), dtype=float32)
    '''

    # L2 ImgIn shape=(?, 14, 14, 32)
    W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
    #    Conv      ->(?, 14, 14, 64)
    #    Pool      ->(?, 7, 7, 64)
    L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
    L2 = tf.nn.relu(L2)
    L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
    L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
    '''
    Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
    Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
    Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
    Tensor("dropout_1/mul:0", shape=(?, 7, 7, 64), dtype=float32)
    '''

    # L3 ImgIn shape=(?, 7, 7, 64)
    W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
    #    Conv      ->(?, 7, 7, 128)
    #    Pool      ->(?, 4, 4, 128)
    #    Reshape   ->(?, 4 * 4 * 128) # Flatten them for FC
    L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
    L3 = tf.nn.relu(L3)
```

```
    L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[
                        1, 2, 2, 1], padding='SAME')
    L3 = tf.nn.dropout(L3, keep_prob=keep_prob)
    L3_flat = tf.reshape(L3, [-1, 128 * 4 * 4])
    '''
    Tensor("Conv2D_2:0", shape=(?, 7, 7, 128), dtype=float32)
    Tensor("Relu_2:0", shape=(?, 7, 7, 128), dtype=float32)
    Tensor("MaxPool_2:0", shape=(?, 4, 4, 128), dtype=float32)
    Tensor("dropout_2/mul:0", shape=(?, 4, 4, 128), dtype=float32)
    Tensor("Reshape_1:0", shape=(?, 2048), dtype=float32)
    '''

    # L4 FC 4x4x128 inputs -> 625 outputs
    W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
                         initializer=tf.contrib.layers.xavier_initializer())
    b4 = tf.Variable(tf.random_normal([625]))
    L4 = tf.nn.relu(tf.matmul(L3_flat, W4) + b4)
    L4 = tf.nn.dropout(L4, keep_prob=keep_prob)
    '''
    Tensor("Relu_3:0", shape=(?, 625), dtype=float32)
    Tensor("dropout_3/mul:0", shape=(?, 625), dtype=float32)
    '''

    # L5 Final FC 625 inputs -> 10 outputs
    W5 = tf.get_variable("W5", shape=[625, 10],
                         initializer=tf.contrib.layers.xavier_initializer())
    b5 = tf.Variable(tf.random_normal([10]))
    logits = tf.matmul(L4, W5) + b5
    '''
    Tensor("add_1:0", shape=(?, 10), dtype=float32)
    '''

    # define cost/loss & optimizer
```

```
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
        logits=logits, labels=Y))
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

    # initialize
    sess = tf.Session()
    sess.run(tf.global_variables_initializer())

    # train my model
    print('Learning started. It takes sometime.')
    for epoch in range(training_epochs):
        avg_cost = 0
        total_batch = int(mnist.train.num_examples / batch_size)

        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
            c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
            avg_cost += c / total_batch

        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

    print('Learning Finished!')

    # Test model and check accuracy

    # if you have a OOM error, please refer to lab-11-X-mnist_deep_cnn_low_memory.py

    correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print('Accuracy:', sess.run(accuracy, feed_dict={
          X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))
```

이전 예제와 같은 CNN 구조이다. 다만, 사이즈 형식에는 변화가 있다.

```
    # Get one and predict
    r = random.randint(0, mnist.test.num_examples - 1)
    print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
    print("Prediction: ", sess.run(
        tf.argmax(logits, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob: 1}))

    # plt.imshow(mnist.test.images[r:r + 1].
    #           reshape(28, 28), cmap='Greys', interpolation='nearest')
    # plt.show()

    '''
    Learning stared. It takes sometime.
    Epoch: 0001 cost = 0.385748474
    Epoch: 0002 cost = 0.092017397
    Epoch: 0003 cost = 0.065854684
    Epoch: 0004 cost = 0.055604566
    Epoch: 0005 cost = 0.045996377
    Epoch: 0006 cost = 0.040913645
    Epoch: 0007 cost = 0.036924479
    Epoch: 0008 cost = 0.032808939
    Epoch: 0009 cost = 0.031791007
    Epoch: 0010 cost = 0.030224456
    Epoch: 0011 cost = 0.026849916
    Epoch: 0012 cost = 0.026826763
    Epoch: 0013 cost = 0.027188021
    Epoch: 0014 cost = 0.023604777
    Epoch: 0015 cost = 0.024607201
    Learning Finished!
    Accuracy: 0.9938
    '''
```

11-3

```
class Model:

    def __init__(self, sess, name):
        self.sess = sess
        self.name = name
        self._build_net()

    def _build_net(self): ...
        self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    def predict(self, x_test, keep_prop=1.0):
        return self.sess.run(self.logits, feed_dict={self.X: x_test, self.keep_prob: keep_prop})

    def get_accuracy(self, x_test, y_test, keep_prop=1.0):
        return self.sess.run(self.accuracy, feed_dict={self.X: x_test, self.Y: y_test, self.keep_prob: keep_prop})

    def train(self, x_data, y_data, keep_prop=0.7):
        return self.sess.run([self.cost, self.optimizer], feed_dict={
            self.X: x_data, self.Y: y_data, self.keep_prob: keep_prop})

# initialize
```

CNN 모델을 클래스로 구성한 것

11-4

```
19    class Model:
26        def _build_net(self):
27            with tf.variable_scope(self.name):
28                # dropout (keep_prob) rate  0.7~0.5 on training, but should be 1
29                # for testing
30                self.training = tf.placeholder(tf.bool)
31
32                # input place holders
33                self.X = tf.placeholder(tf.float32, [None, 784])
34
35                # img 28x28x1 (black/white), Input Layer
36                X_img = tf.reshape(self.X, [-1, 28, 28, 1])
37                self.Y = tf.placeholder(tf.float32, [None, 10])
38
39                # Convolutional Layer #1
40                conv1 = tf.layers.conv2d(inputs=X_img, filters=32, kernel_size=[3, 3],
41                                         padding="SAME", activation=tf.nn.relu)
42                # Pooling Layer #1
43                pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
44                                                padding="SAME", strides=2)
45                dropout1 = tf.layers.dropout(inputs=pool1,
46                                             rate=0.3, training=self.training)
47
48                # Convolutional Layer #2 and Pooling Layer #2
49                conv2 = tf.layers.conv2d(inputs=dropout1, filters=64, kernel_size=[3, 3],
50                                         padding="SAME", activation=tf.nn.relu)
51                pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
52                                                padding="SAME", strides=2)
53                dropout2 = tf.layers.dropout(inputs=pool2,
54                                             rate=0.3, training=self.training)
55
56                # Convolutional Layer #3 and Pooling Layer #3
```

```
19      class Model:
26          def _build_net(self):
53              dropout2 = tf.layers.dropout(inputs=pool2,
54                                           rate=0.3, training=self.training)
55
56              # Convolutional Layer #3 and Pooling Layer #3
57              conv3 = tf.layers.conv2d(inputs=dropout2, filters=128, kernel_size=[3, 3],
58                                       padding="same", activation=tf.nn.relu)
59              pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2],
60                                              padding="same", strides=2)
61              dropout3 = tf.layers.dropout(inputs=pool3,
62                                           rate=0.3, training=self.training)
63
64              # Dense Layer with Relu
65              flat = tf.reshape(dropout3, [-1, 128 * 4 * 4])
66              dense4 = tf.layers.dense(inputs=flat,
67                                       units=625, activation=tf.nn.relu)
68              dropout4 = tf.layers.dropout(inputs=dense4,
69                                           rate=0.5, training=self.training)
70
71              # Logits (no activation) Layer: L5 Final FC 625 inputs -> 10 outputs
72              self.logits = tf.layers.dense(inputs=dropout4, units=10)
73
74          # define cost/loss & optimizer
75          self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
76              logits=self.logits, labels=self.Y))
77          self.optimizer = tf.train.AdamOptimizer(
78              learning_rate=learning_rate).minimize(self.cost)
79
80          correct_prediction = tf.equal(
81              tf.argmax(self.logits, 1), tf.argmax(self.Y, 1))
82          self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
83
```

CNN Layer 구현 예제

```
    # Convolutional Layer #2 and Pooling Layer #2
    conv2 = tf.layers.conv2d(inputs=dropout1, filters=64, kernel_size=[3, 3],
                             padding="SAME", activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
                                    padding="SAME", strides=2)
    dropout2 = tf.layers.dropout(inputs=pool2,
                                 rate=0.3, training=self.training)

    # Convolutional Layer #3 and Pooling Layer #3
    conv3 = tf.layers.conv2d(inputs=dropout2, filters=128, kernel_size=[3, 3],
                             padding="SAME", activation=tf.nn.relu)
    pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2],
                                    padding="SAME", strides=2)
    dropout3 = tf.layers.dropout(inputs=pool3,
                                 rate=0.3, training=self.training)

    # Dense Layer with Relu
    flat = tf.reshape(dropout3, [-1, 128 * 4 * 4])
    dense4 = tf.layers.dense(inputs=flat,
                             units=625, activation=tf.nn.relu)
    dropout4 = tf.layers.dropout(inputs=dense4,
                                 rate=0.5, training=self.training)

    # Logits (no activation) Layer: L5 Final FC 625 inputs -> 10 outputs
    self.logits = tf.layers.dense(inputs=dropout4, units=10)

# define cost/loss & optimizer
self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=self.logits, labels=self.Y))
self.optimizer = tf.train.AdamOptimizer(
    learning_rate=learning_rate).minimize(self.cost)
```

- Ensemble 적용

12-1

```python
# Lab 12 RNN
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # reproducibility

idx2char = ['h', 'i', 'e', 'l', 'o']
# Teach hello: hihell -> ihello
x_data = [[0, 1, 0, 2, 3, 3]]   # hihell
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 0],   # l 3
              [0, 0, 0, 1, 0]]]  # l 3

y_data = [[1, 0, 2, 3, 3, 4]]    # ihello

num_classes = 5
input_dim = 5  # one-hot size
hidden_size = 5  # output from the LSTM. 5 to directly predict one-hot
batch_size = 1   # one sentence
sequence_length = 6  # |ihello| == 6
learning_rate = 0.1

X = tf.placeholder(
    tf.float32, [None, sequence_length, input_dim])  # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label

cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size, state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
    cell, X, initial_state=initial_state, dtype=tf.float32)

# FC layer
```

```
    X_for_fc = tf.reshape(outputs, [-1, hidden_size])
    # fc_w = tf.get_variable("fc_w", [hidden_size, num_classes])
    # fc_b = tf.get_variable("fc_b", [num_classes])
    # outputs = tf.matmul(X_for_fc, fc_w) + fc_b
    outputs = tf.contrib.layers.fully_connected(
        inputs=X_for_fc, num_outputs=num_classes, activation_fn=None)

    # reshape out for sequence_loss
    outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])

    weights = tf.ones([batch_size, sequence_length])
    sequence_loss = tf.contrib.seq2seq.sequence_loss(
        logits=outputs, targets=Y, weights=weights)
    loss = tf.reduce_mean(sequence_loss)
    train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

    prediction = tf.argmax(outputs, axis=2)

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for i in range(50):
            l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
            result = sess.run(prediction, feed_dict={X: x_one_hot})
            print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

            # print char using dic
            result_str = [idx2char[c] for c in np.squeeze(result)]
            print("\tPrediction str: ", ''.join(result_str))
```

DNN 코드

다음의 코드로 char 처리를 하는 코드를 만들 수 있다.

12-2

hyper parameters

dic_size = len(char2idx) # RNN input size (one hot size)

hidden_size = len(char2idx) # RNN output size

```
num_classes = len(char2idx)  # final output size (RNN or softmax, etc.)
batch_size = 1  # one sample data, one batch
sequence_length = len(sample) - 1  # number of lstm rollings (unit #)
learning_rate = 0.1

sample_idx = [char2idx[c] for c in sample]  # char to index
x_data = [sample_idx[:-1]]  # X data sample (0 ~ n-1) hello: hell
y_data = [sample_idx[1:]]  # Y label sample (1 ~ n) hello: ello

X = tf.placeholder(tf.int32, [None, sequence_length])  # X data
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label

x_one_hot = tf.one_hot(X, num_classes)  # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0
cell = tf.contrib.rnn.BasicLSTMCell(
    num_units=hidden_size, state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
    cell, x_one_hot, initial_state=initial_state, dtype=tf.float32)

# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
outputs = tf.contrib.layers.fully_connected(X_for_fc, num_classes, activation_fn=None)
```

12-3

다음의 코드를 넣어 softmax를 할 수 있다

```
# flatten the data (ignore batches for now). No effect if the batch size is 1
X_one_hot = tf.one_hot(X, num_classes)  # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0
X_for_softmax = tf.reshape(X_one_hot, [-1, rnn_hidden_size])

# softmax layer (rnn_hidden_size -> num_classes)
softmax_w = tf.get_variable("softmax_w", [rnn_hidden_size, num_classes])
softmax_b = tf.get_variable("softmax_b", [num_classes])
outputs = tf.matmul(X_for_softmax, softmax_w) + softmax_b

# expend the data (revive the batches)
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])
weights = tf.ones([batch_size, sequence_length])
```

Additional codes:
스코프 예제

```
# Lab 13 Using Scope

import tensorflow as tf

import random


from tensorflow.examples.tutorials.mnist import input_data


tf.set_random_seed(777)  # reproducibility


mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# Check out https://www.tensorflow.org/get_started/mnist/beginners for

# more information about the mnist dataset


# parameters

learning_rate = 0.001

training_epochs = 15

batch_size = 100


# input place holders

X = tf.placeholder(tf.float32, [None, 784])

Y = tf.placeholder(tf.float32, [None, 10])


# dropout (keep_prob) rate  0.7~0.5 on training, but should be 1 for testing

keep_prob = tf.placeholder(tf.float32)
```

```
# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow
with tf.variable_scope('layer1') as scope:
    W1 = tf.get_variable("W", shape=[784, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
    b1 = tf.Variable(tf.random_normal([512]))
    L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
    L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

with tf.variable_scope('layer2') as scope:
    W2 = tf.get_variable("W", shape=[512, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
    b2 = tf.Variable(tf.random_normal([512]))
    L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
    L2 = tf.nn.dropout(L2, keep_prob=keep_prob)

with tf.variable_scope('layer3') as scope:
    W3 = tf.get_variable("W", shape=[512, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
    b3 = tf.Variable(tf.random_normal([512]))
    L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
    L3 = tf.nn.dropout(L3, keep_prob=keep_prob)

with tf.variable_scope('layer4') as scope:
    W4 = tf.get_variable("W", shape=[512, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
```

```
    b4 = tf.Variable(tf.random_normal([512]))
    L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
    L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

with tf.variable_scope('layer5') as scope:
    W5 = tf.get_variable("W", shape=[512, 10],
                         initializer=tf.contrib.layers.xavier_initializer())
    b5 = tf.Variable(tf.random_normal([10]))
    hypothesis = tf.matmul(L4, W5) + b5

print(W1, W5)

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)
```

```
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        sess.run(optimizer, feed_dict=feed_dict)
        avg_cost += sess.run(cost, feed_dict=feed_dict) / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob: 1}))

# plt.imshow(mnist.test.images[r:r + 1].
#                 reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()
```

2. 텐서보드 예제

```
# Lab 13 Tensorboard
import tensorflow as tf
import random

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

TB_SUMMARY_DIR = './tb/mnist'

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# Image input
```

```
x_image = tf.reshape(X, [-1, 28, 28, 1])
tf.summary.image('input', x_image, 3)

# dropout (keep_prob) rate  0.7~0.5 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow
with tf.variable_scope('layer1') as scope:
    W1 = tf.get_variable("W", shape=[784, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
    b1 = tf.Variable(tf.random_normal([512]))
    L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
    L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

    tf.summary.histogram("X", X)
    tf.summary.histogram("weights", W1)
    tf.summary.histogram("bias", b1)
    tf.summary.histogram("layer", L1)

with tf.variable_scope('layer2') as scope:
    W2 = tf.get_variable("W", shape=[512, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
    b2 = tf.Variable(tf.random_normal([512]))
    L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
    L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
```

```
        tf.summary.histogram("weights", W2)
        tf.summary.histogram("bias", b2)
        tf.summary.histogram("layer", L2)

with tf.variable_scope('layer3') as scope:
        W3 = tf.get_variable("W", shape=[512, 512],
                                initializer=tf.contrib.layers.xavier_initializer())
        b3 = tf.Variable(tf.random_normal([512]))
        L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
        L3 = tf.nn.dropout(L3, keep_prob=keep_prob)

        tf.summary.histogram("weights", W3)
        tf.summary.histogram("bias", b3)
        tf.summary.histogram("layer", L3)

with tf.variable_scope('layer4') as scope:
        W4 = tf.get_variable("W", shape=[512, 512],
                                initializer=tf.contrib.layers.xavier_initializer())
        b4 = tf.Variable(tf.random_normal([512]))
        L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
        L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

        tf.summary.histogram("weights", W4)
        tf.summary.histogram("bias", b4)
        tf.summary.histogram("layer", L4)
```

```
with tf.variable_scope('layer5') as scope:
    W5 = tf.get_variable("W", shape=[512, 10],
                         initializer=tf.contrib.layers.xavier_initializer())
    b5 = tf.Variable(tf.random_normal([10]))
    hypothesis = tf.matmul(L4, W5) + b5

    tf.summary.histogram("weights", W5)
    tf.summary.histogram("bias", b5)
    tf.summary.histogram("hypothesis", hypothesis)



# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

tf.summary.scalar("loss", cost)

# Summary
summary = tf.summary.merge_all()

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

```
# Create summary writer
writer = tf.summary.FileWriter(TB_SUMMARY_DIR)
writer.add_graph(sess.graph)
global_step = 0

print('Start learning!')

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
        writer.add_summary(s, global_step=global_step)
        global_step += 1

        avg_cost += sess.run(cost, feed_dict=feed_dict) / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
```

```
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
        X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob: 1}))

# plt.imshow(mnist.test.images[r:r + 1].
#                 reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()
```

3. Save & Restore 예제

```
# Lab 13 Saver and Restore
import tensorflow as tf
import random
# import matplotlib.pyplot as plt
import os

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility
```

```
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

CHECK_POINT_DIR = TB_SUMMARY_DIR = './tb/mnist2'


# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# Image input
x_image = tf.reshape(X, [-1, 28, 28, 1])
tf.summary.image('input', x_image, 3)

# dropout (keep_prob) rate  0.7~0.5 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow
with tf.variable_scope('layer1'):
```

```
    W1 = tf.get_variable("W", shape=[784, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
    b1 = tf.Variable(tf.random_normal([512]))
    L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
    L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

    tf.summary.histogram("X", X)
    tf.summary.histogram("weights", W1)
    tf.summary.histogram("bias", b1)
    tf.summary.histogram("layer", L1)

with tf.variable_scope('layer2'):
    W2 = tf.get_variable("W", shape=[512, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
    b2 = tf.Variable(tf.random_normal([512]))
    L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
    L2 = tf.nn.dropout(L2, keep_prob=keep_prob)

    tf.summary.histogram("weights", W2)
    tf.summary.histogram("bias", b2)
    tf.summary.histogram("layer", L2)

with tf.variable_scope('layer3'):
    W3 = tf.get_variable("W", shape=[512, 512],
                         initializer=tf.contrib.layers.xavier_initializer())
    b3 = tf.Variable(tf.random_normal([512]))
```

```
        L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
        L3 = tf.nn.dropout(L3, keep_prob=keep_prob)

        tf.summary.histogram("weights", W3)
        tf.summary.histogram("bias", b3)
        tf.summary.histogram("layer", L3)

with tf.variable_scope('layer4'):
        W4 = tf.get_variable("W", shape=[512, 512],
                                initializer=tf.contrib.layers.xavier_initializer())
        b4 = tf.Variable(tf.random_normal([512]))
        L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
        L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

        tf.summary.histogram("weights", W4)
        tf.summary.histogram("bias", b4)
        tf.summary.histogram("layer", L4)

with tf.variable_scope('layer5'):
        W5 = tf.get_variable("W", shape=[512, 10],
                                initializer=tf.contrib.layers.xavier_initializer())
        b5 = tf.Variable(tf.random_normal([10]))
        hypothesis = tf.matmul(L4, W5) + b5

        tf.summary.histogram("weights", W5)
        tf.summary.histogram("bias", b5)
```

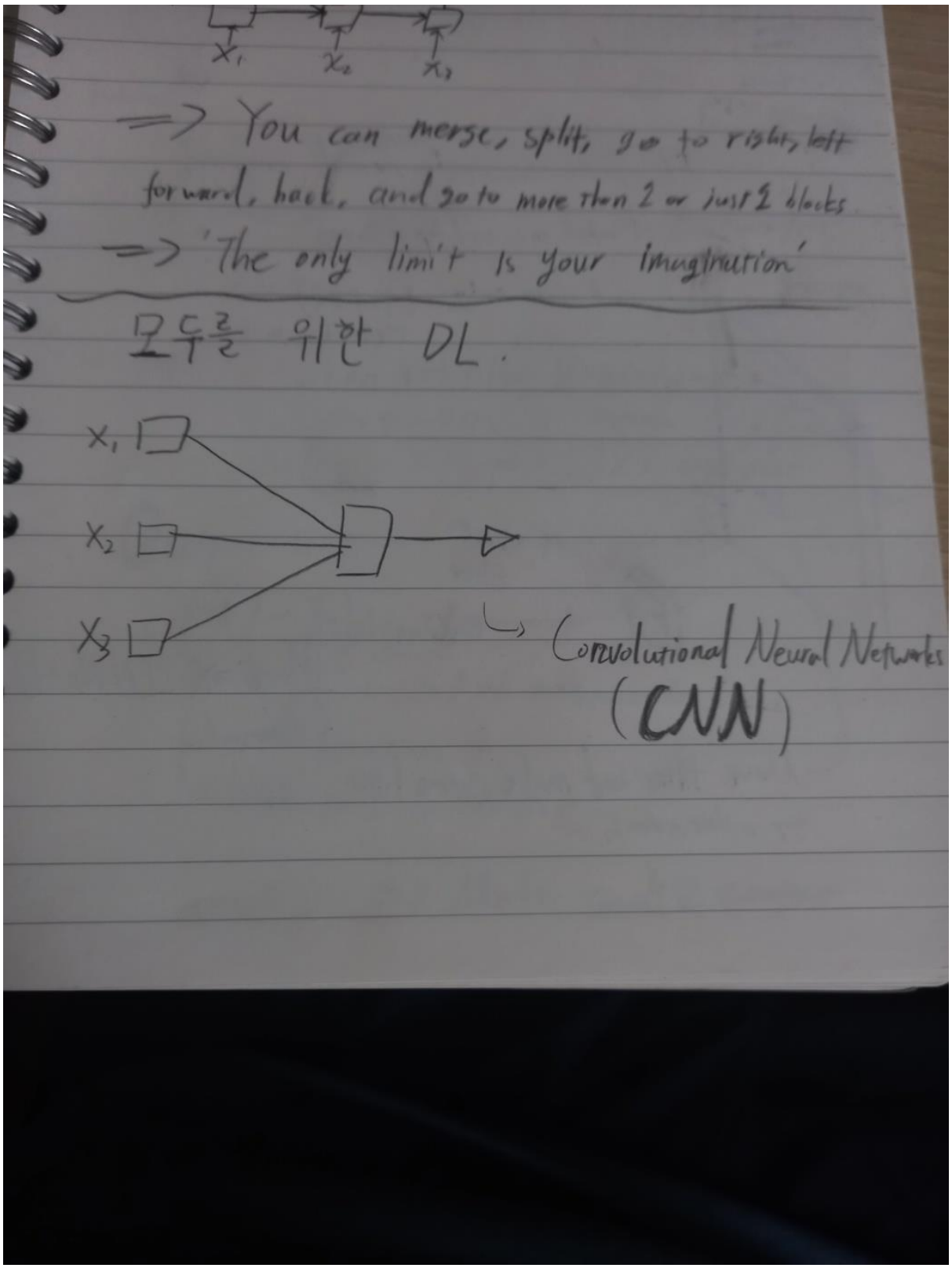
```
    tf.summary.histogram("hypothesis", hypothesis)



# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

tf.summary.scalar("loss", cost)

last_epoch = tf.Variable(0, name='last_epoch')

# Summary
summary = tf.summary.merge_all()

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# Create summary writer
writer = tf.summary.FileWriter(TB_SUMMARY_DIR)
writer.add_graph(sess.graph)
global_step = 0

# Saver and Restore
saver = tf.train.Saver()
```

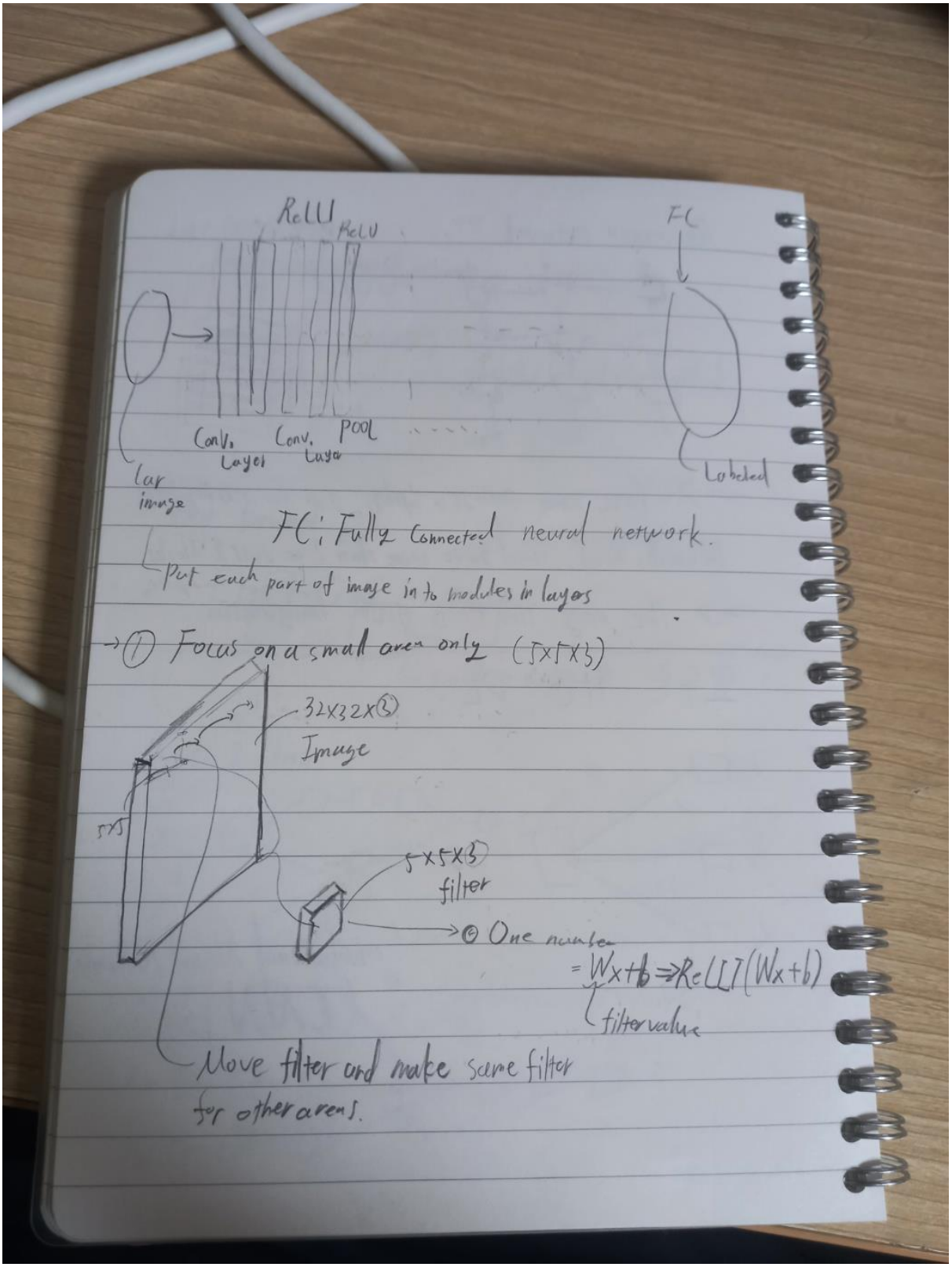
```
checkpoint = tf.train.get_checkpoint_state(CHECK_POINT_DIR)

if checkpoint and checkpoint.model_checkpoint_path:
    try:
        saver.restore(sess, checkpoint.model_checkpoint_path)
        print("Successfully loaded:", checkpoint.model_checkpoint_path)
    except:
        print("Error on loading old network weights")
else:
    print("Could not find old network weights")

start_from = sess.run(last_epoch)

# train my model
print('Start learning from:', start_from)

for epoch in range(start_from, training_epochs):
    print('Start Epoch:', epoch)

    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
```

```
            writer.add_summary(s, global_step=global_step)
            global_step += 1

            avg_cost += sess.run(cost, feed_dict=feed_dict) / total_batch

        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

        print("Saving network...")
        sess.run(last_epoch.assign(epoch + 1))
        if not os.path.exists(CHECK_POINT_DIR):
            os.makedirs(CHECK_POINT_DIR)
        saver.save(sess, CHECK_POINT_DIR + "/model", global_step=i)

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob: 1}))
```

```
# plt.imshow(mnist.test.images[r:r + 1].
#                    reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()
```

- 개념 정리


x1
x2
x3
=> You can merge, split, go to right, left forward, back, and go to more than 2 or just 2 blocks.
=> 'The only limit is your imagination'
모듈을 위한 DL.
x1
x2
x3
Convolutional Neural Networks
(CNN)



FC; Fully Connected neural network.

Put each part of image into modules in layers

→① Focus on a small area only (5×5×3)

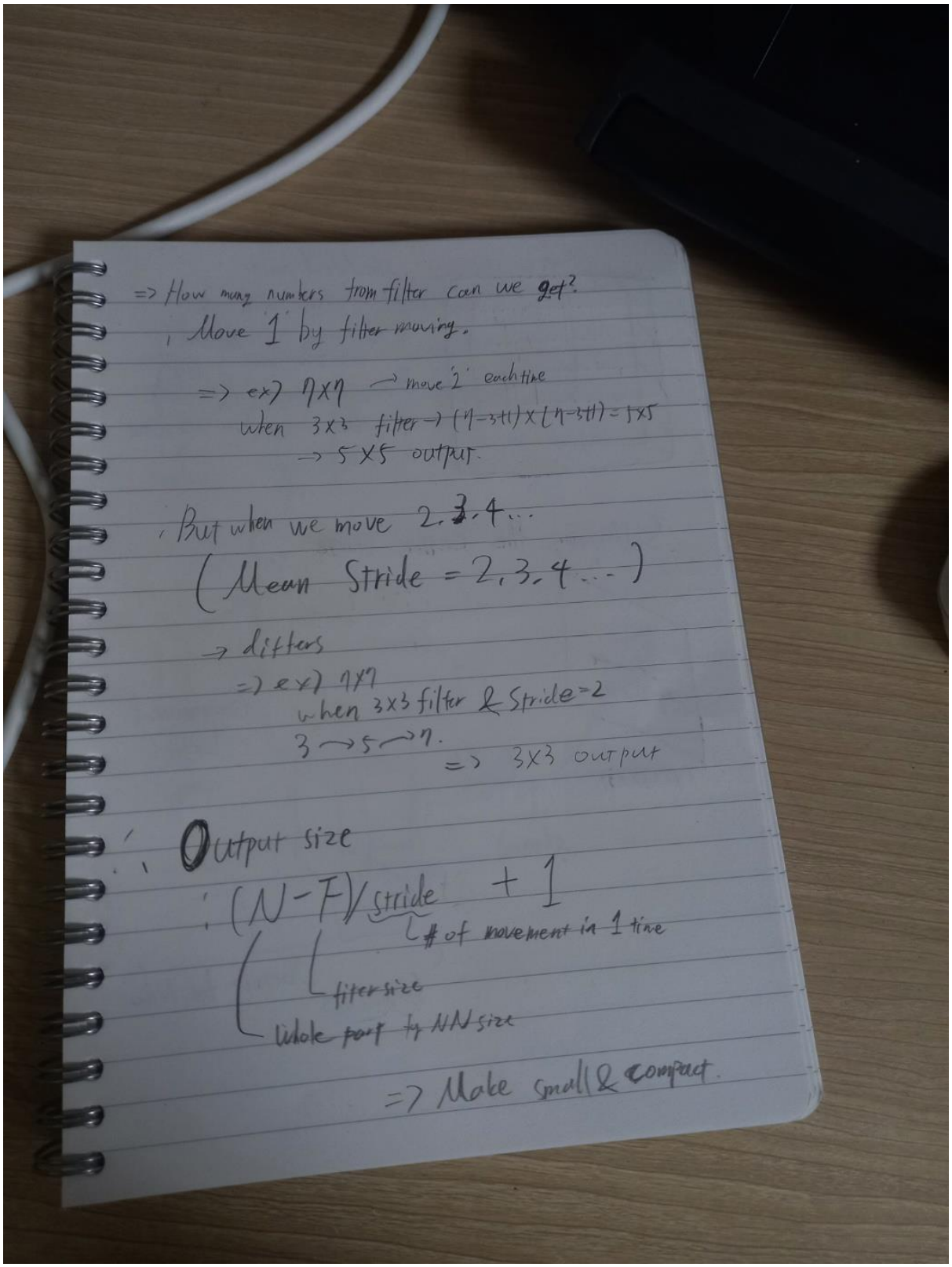
32×32×3
Image
5×5
5×5×3
filter

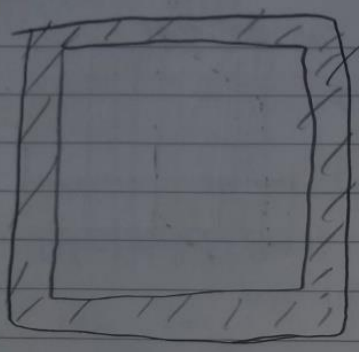
→② One number

$= Wx+b \Rightarrow ReLU(Wx+b)$

filter value

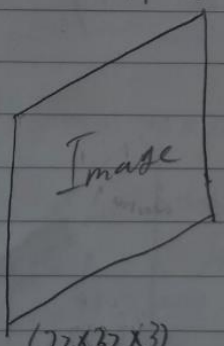
Move filter and make same filter for other areas.

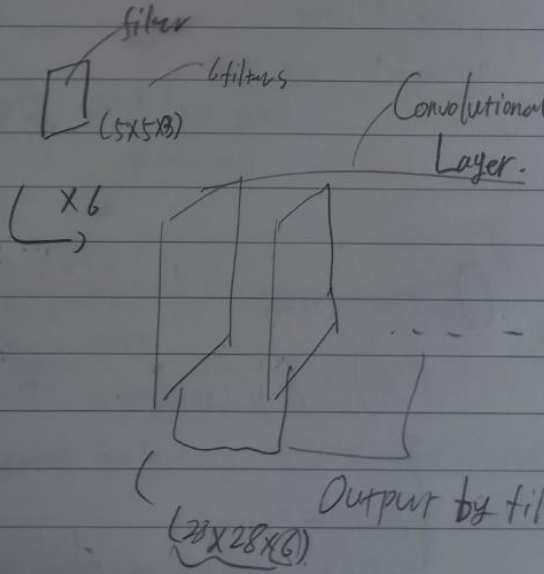

=> How many numbers from filter can we get?
, Move 1 by filter moving.
=> ex) 7x7 → move '2' each time
when 3x3 filter → (7-3+1)x(7-3+1) = 5x5
→ 5x5 output.
, But when we move 2, 3, 4...
(Mean Stride = 2, 3, 4...)
→ differs
=> ex) 7x7
when 3x3 filter & Stride=2
3 → 5 → 7.
=> 3x3 output
∴ Output size
(N-F)/stride + 1
# of movement in 1 time
filter size
whole part by NN size
=> Make small & compact.

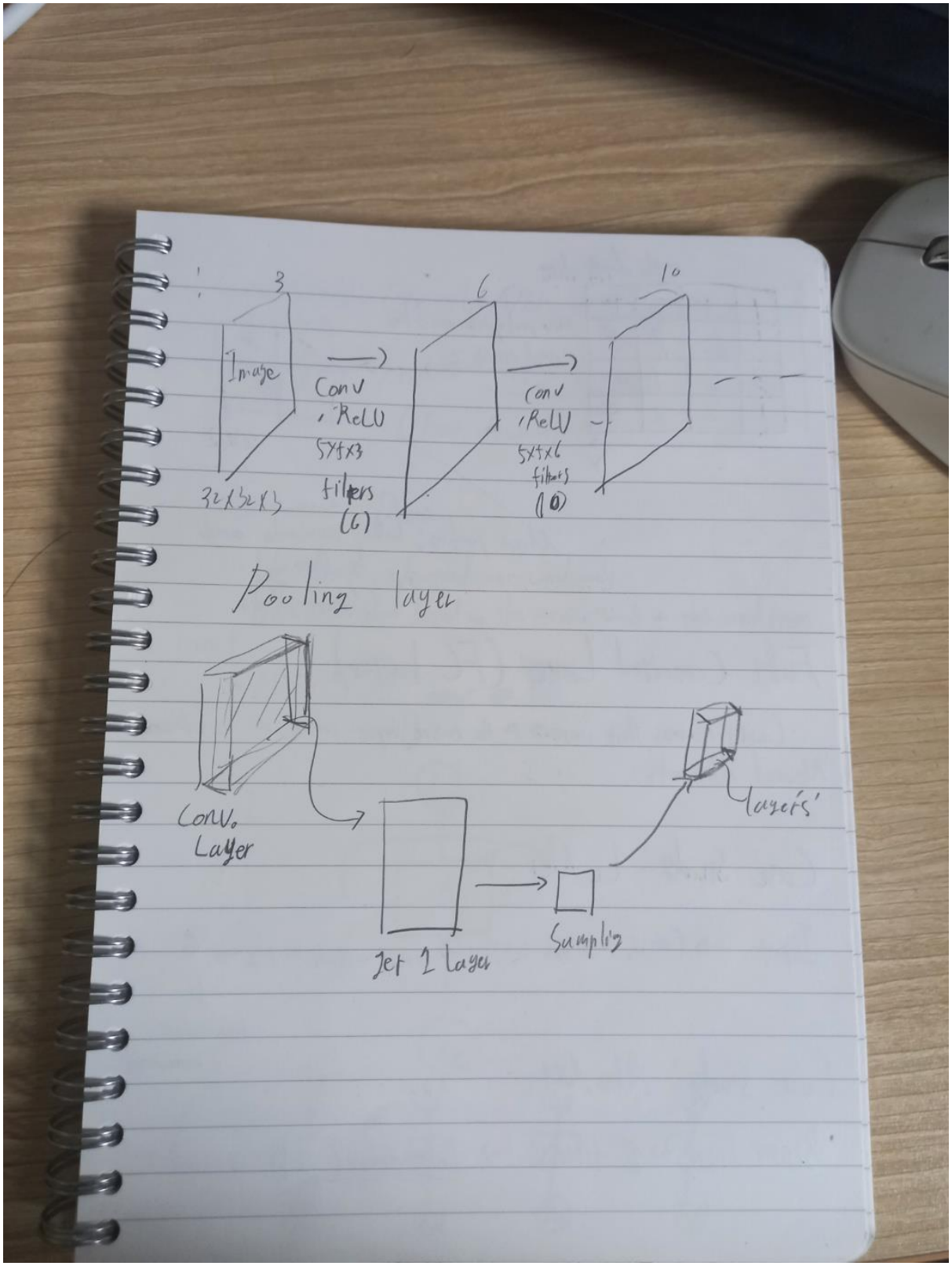


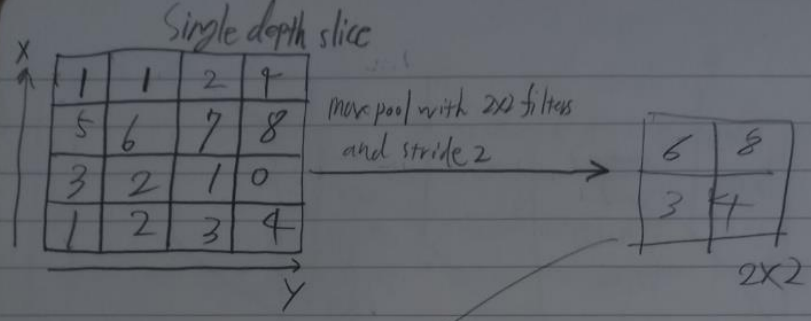
added part
fulled with '0'
(To notice it is the end.)
=> zero pad
ex) add zero pad to 7x7
to make output from 3x3 filter
is 3x3.

Swipping the entire image.






3
Image
32x32x3
Conv
, ReLU
5x5x3
filters
(6)
6
Conv
, ReLU
5x5x6
filters
(10)
10
Pooling layer
Conv. Layer
layers'
Sampling
Jet 1 layer



Max pooling: take maximum sample by filter (Sampling)

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

Case Study: LeNet-5

Input → Conv. → Sub sampling → Conv → Subsample 2 → FC → Gaussian Connections

Case Study: AlexNet

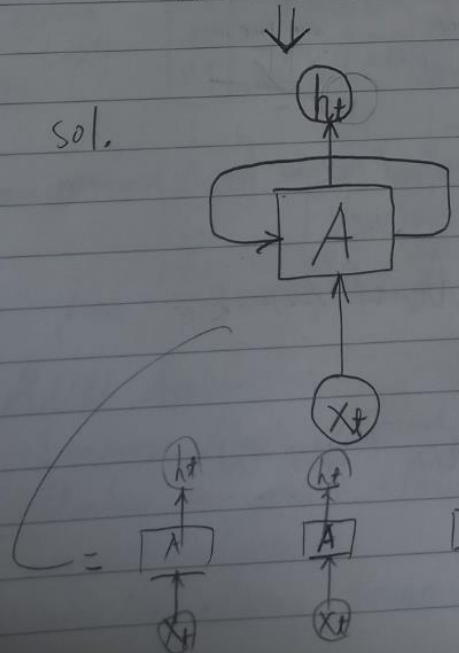
Bigger Input → 96 filters → Parameters (35k)

Case Study: GoogLeNet → Rezo Blat

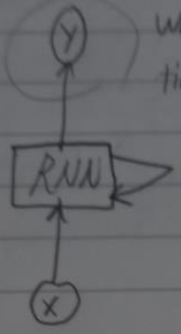
Case Study: ResNet: 152 layers
(Fast forward)

Sequence Data

- We don't understand one word only
- We understand based on the previous words + this word. (time series).
- NN / CNN cannot do this



Recurrent Neural Network



We usually want to predict a vector at some time steps

We can process a sequence of vectors x by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

h_t: new state

f_W: some function with parameters W

h_{t-1}: old state

x_t: input vector at some time step

Notice: the 'same' function and the same set of parameters are used at every time step.

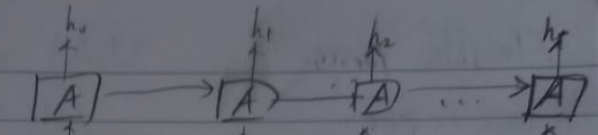
(Vanilla) Recurrent Neural Network

$$h_t = f_W(h_{t-1}; x_t)$$

$$\hookrightarrow h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

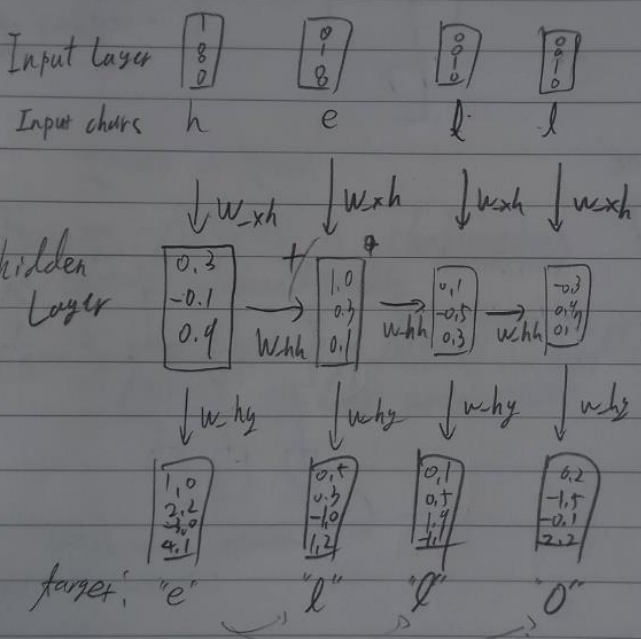
$$y_t = W_{hy} h_t$$

Character-level language mde



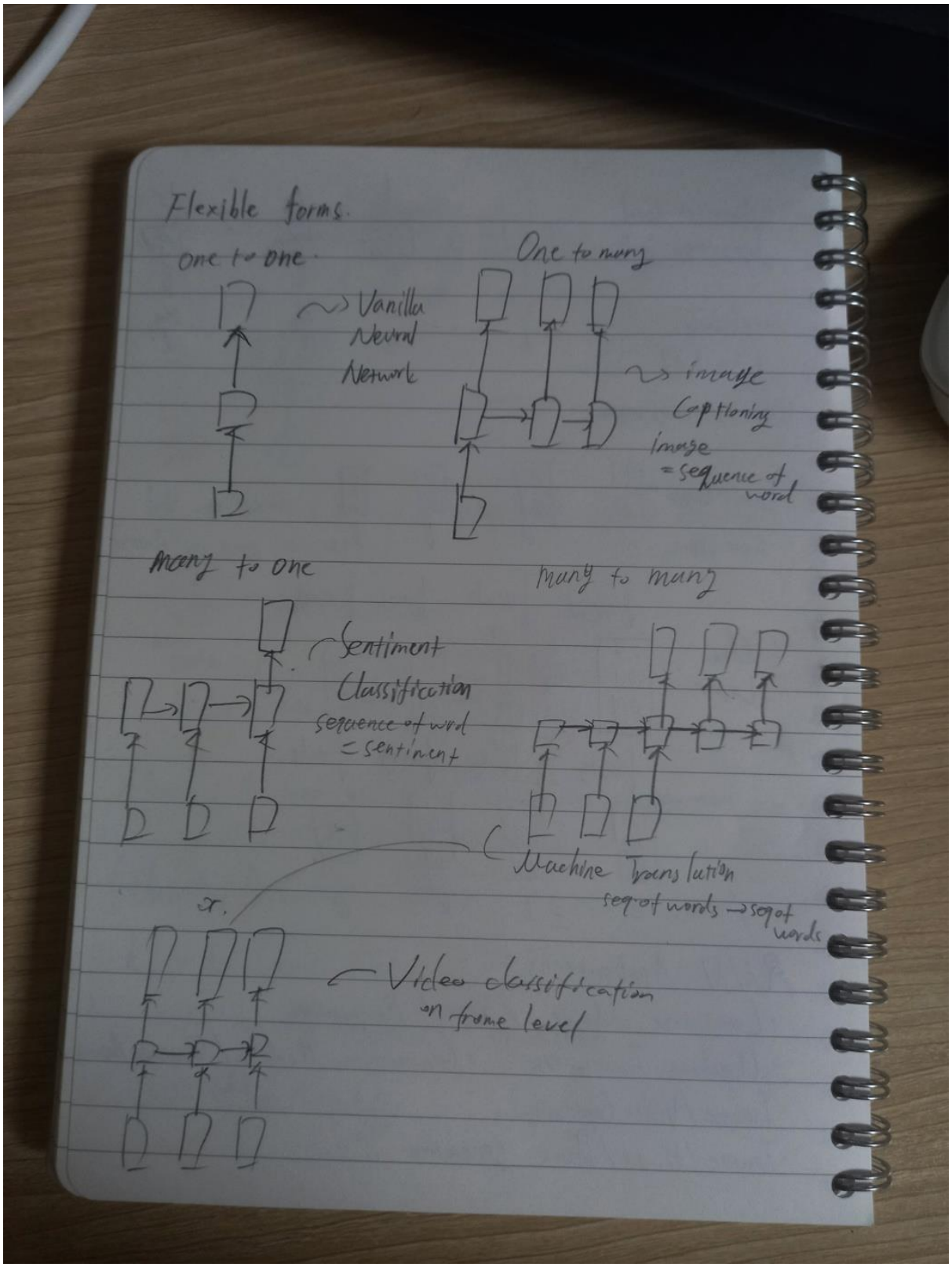
Vocabulary:
[he, h, o]

Example training sequence: "hello"



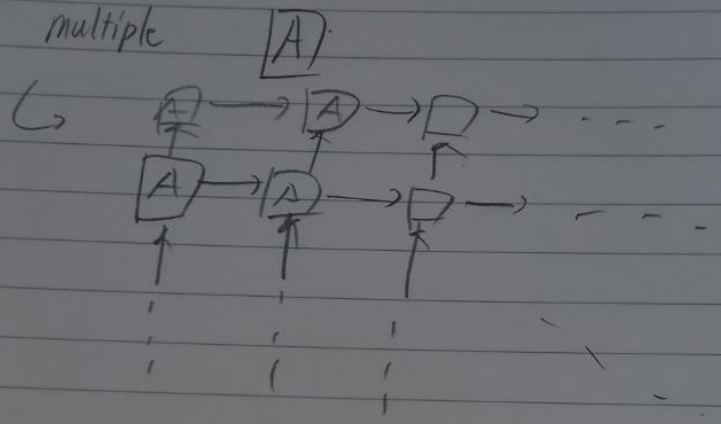
RNN Applications

- Language Mode / Speech Recognition
- Machine Translation
- Conversation Modeling / Question Answering
- Image/Video Captioning
- Image/Music/Dance generation


Flexible forms.
one to one.
Vanilla Neural Network
One to many
image Captioning
image = sequence of word
many to one
Sentiment Classification
sequence of word = sentiment
many to many
Machine Translation
seq of words → seq of words
or.
Video classification on frame level

Multi-Layer RNN

multiple



Training RNNs is challenging.

- Several advanced models
  - Long Short Term Memory (LSTM)
  - GRU by Cho et al. 2014