# 학습 모델 저장 및 재사용

## MNIST 모델 학습

김 대 환

2022.

# MNIST 학습모델 저장 및 재사용

- **MNIST 모델 학습 개요**
  - 머신 러닝 계의 `hello world`와 같은 손 글씨 숫자 인식 문제 (MNIST)에 대한 신경망 학습
  - 학습 모델을 파일로 저장하여 필요할 때 재사용
  - 텐서플로우에 기본 내장된 mnist 모듈을 이용하여 학습 데이터 로드
- **학습 모델의 저장과 복구, 그리고 학습에 사용되는 변수 생성**

# MNIST 학습모델 저장 및 재사용

- 학습 모델의 저장과 복구, 그리고 학습에 사용되는 변수 생성

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

# 모델 저장을 위한 부분
import os

# save_file = './model_mnist.ckpt'
SAVER_DIR = "modelMNIST"
checkpoint_path = os.path.join(SAVER_DIR, "modelMNIST")
ckpt = tf.train.get_checkpoint_state(SAVER_DIR)

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

learning_rate = 0.001
training_epochs = 5
batch_size = 100
```

# MNIST 학습모델 저장 및 재사용

- **신경망 학습 모델 구성**
  - 입력 값의 차원은 [배치 크기, 특성 값]으로 구성
    - ✓ 28x28 크기의 이미지에 대해 특성 값은 784 개로 설정하며 분류는 0 ~ 9까지 10개로 한다
  - 신경망의 각 계층을 `**764개의 입력 특성 값 -> 256개의 뉴런 -> 256개의 뉴런 -> 10개의 분류 결과**`로 구성
  - 각 계층의 활성 함수는 `**ReLU**` 사용
  - 마지막 계층은 **softmax cross entropy** 손실 함수를 적용
  - 최적화는 **AdamOptimzer** 사용하며, **learning rate**은 0.001 사용
  - `global_step` 변수를 정의하여 학습용 변수들을 최적화할 때마다 값을 1씩 증가

# MNIST 학습모델 저장 및 재사용

- 신경망 학습 모델 구성

```
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

W1 = tf.Variable(tf.random_normal([784, 256], stddev=0.01))
L1 = tf.nn.relu(tf.matmul(X, W1))

W2 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
L2 = tf.nn.relu(tf.matmul(L1, W2))

W3 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
bias = tf.Variable(tf.random_normal([10]))
model = tf.add(tf.matmul(L2, W3), bias)

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate).minimize(cost)
```

# MNIST 학습모델 저장 및 재사용

- **세션을 열고 최적화 실행**
  - 세션 실행과 학습 결과를 저장하거나 불러올 함수 정의
  - 체크포인터(checkpoint) 파일은 학습 모델을 저장한 파일
  - 전체 이미지를 (6만장) 5회 반복 (epoch) 학습한다
  - 한 번에 학습할 이미지 개수를 (batch size) 100개로 한정
    - ✓ 텐서플로우의 mnist 모델은 next_batch 함수를 이용하여 지정한 크기만큼 학습할 데이터를 가져올 수 있다.
  - epoch 값을 이용해 몇 번의 학습을 진행했는지 global_step 변수에 저장

- **학습 진행**
  - 최적화 후에 학습된 변수들을 지정한 체크포인트 파일에 저장
  - 체크포인트 파일 위치는 `os.path.join` 함수를 통해 `./modelMNIST` 경로로 설정
    - ✓ 저장할 파일 이름에 `global_step` 값을 추가한다.

# MNIST 학습모델 저장 및 재사용

- **학습진행**

```
saver = tf.compat.v1.train.Saver()

with tf.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())

    # 저장된 모델과 파라미터가 있으면 이를 불러온다(restore)
    # Restore 된 모델을 이용해서 테스트 데이터에 대한 정확도를 출력한다.

    if ckpt and ckpt.model_checkpoint_path:
        saver.restore(sess, ckpt.model_checkpoint_path)
        is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
        accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
        print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

# MNIST 학습모델 저장 및 재사용

- **학습진행**

```
else:
        for epoch in range(training_epochs):
            total_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)
                _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs, Y: batch_ys})
                total_cost += cost_val

            # Tensorboard에서 epoch별 스칼라값 확인하기 위함
            # writer.add_summary(summary, global_step = epoch)

            print('Epoch: %d, ' % (epoch + 1), 'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))

            # epoch별 모델 체크포인트 저장
            # saver.save(sess, checkpoint_path, global_step = epoch)
            saver.save(sess, checkpoint_path, global_step = epoch)
```

# MNIST 학습모델 저장 및 재사용

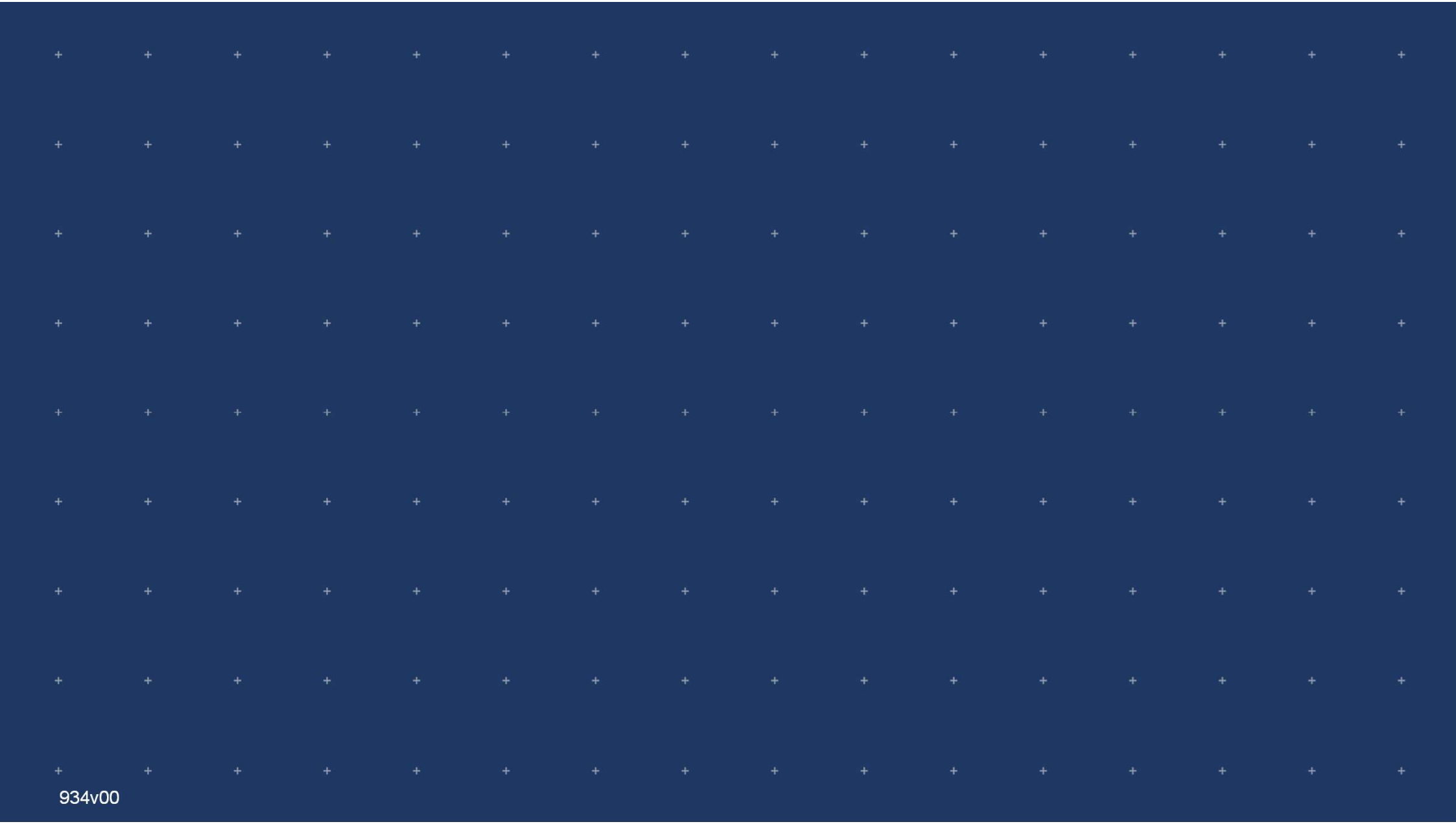
- **결과 확인**
  - model로 예측한 값과 실제 레이블인 Y의 값을 비교
  - `tf.argmax` 함수를 이용하여 예측한 값에서 가장 큰 값을 예측한 레이블로 평가
    - ✓ 예를 들어, [0.1 0 0 0.7 0 0.2 0 0 0 0] 인 경우에는 5 가 된다

```
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

  - 학습을 반복(epoch) 할 때마다 정확도가 개선 됨을 확인
  - 첫 번째 실행 후 학습 모델이 저장 되기 때문에 이 후에는 학습된 모델을 불러와서 예측을 진행

```
Epoch: 1,  Avg. cost = 0.441
Epoch: 2,  Avg. cost = 0.135
Epoch: 3,  Avg. cost = 0.088
Epoch: 4,  Avg. cost = 0.065
Epoch: 5,  Avg. cost = 0.048
Accuracy: 0.9743
```


934v00