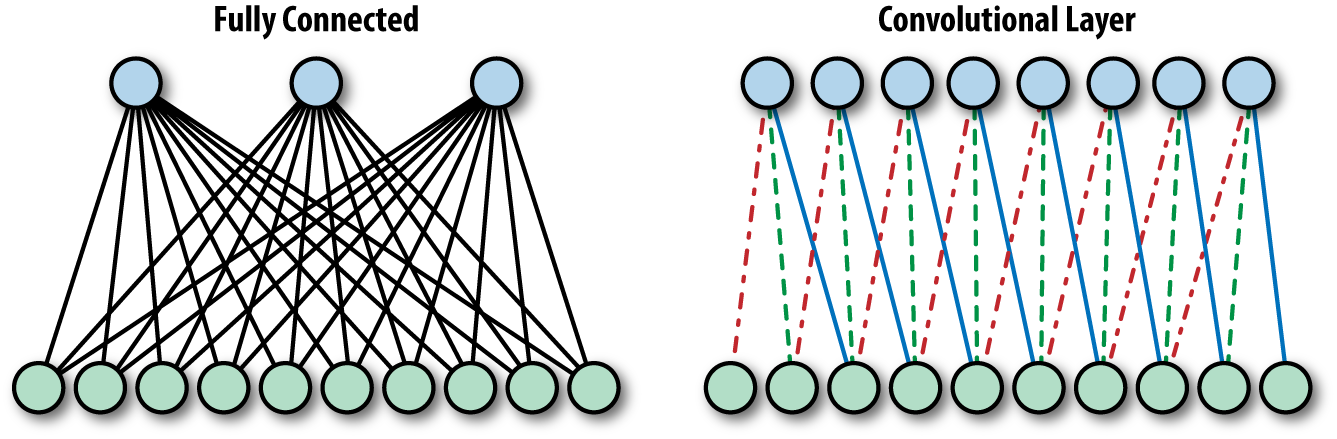
CNN(Convolutional Neural Network)

Introduction to CNNs

합성곱 신경망은 특히 전망이 좋은 딥러닝으로 지난 몇 년간 특별한 위치를 차지해왔다. 이미지 처리에서 파생된 합성곱 계층은 사실상 모든 딥러닝의 일부분으로 들어갔으며 대부분의 경우 매우 성공적이다.

**완전 연결(Fully Connected)신경망**과 **합성곱(Convolution)신경망** 간의 근본적인 차이점은 이어지는 계층 산의 연결 패턴이다. 완전 연결 신경망은 각각의 유닛이 앞의 계층의 모든 유닛과 연결되어 있는 반면에 합성곱 계층에서 각각의 유닛은 이전 계층에서 근접해 있는 몇 개의 유닛들에게만 연결된다. 게다가 모든 유닛은 이전 계층에 동일한 방법으로 연결되어 같은 값의 가중치와 구조를 공유한다.



CNN이 도입된 배경으로는 신경과학적 영감과 이미지의 본질에 대한 통찰, 학습 이론에 관한 것들이 있다.

신경과학적 관점에서의 설명은, 시각 처리가 이루어짐에 따라 정보는 입력의 좀 더 넓은 부분으로 증가해가며 통합되고 이는 계층적으로 이루어진다. 합성곱 신경망도 동일한 패턴을 따르는데, 각각의 합성곱 계층은 네트워크의 깊은 곳으로 들어감에 따라 이미지의 더 큰 영역을 보게 된다. 일반적으로 합성곱 계층 다음에 완전 연결 계층으로 이어지는데, 생물학에서 영감을 받은 비유로 보면 이 계층은 전역적 정보를 다루는 시각 처리의 상위 단계처럼 동작한다고 볼 수 있다.

공학 관점의 설명은, 이미지와 그 안에 담긴 내용의 본질에서 출발한다. 이미지 속에서 어떤 물체를 찾을 때, 이미지 내의 위치와는 무관하게 찾을 수 있어야 한다. 이는 동일한 내용이 이미지의 다른 위치에서 발견될 수 있다는 자연스러운 이미지의 속성을 반영한다. 이것을 **불변성(Invariance)**라고 하며, 이런 종류의 불변성은 회전이 발생하거나 조명 조건이 변하더라도 유지되어야 한다. 그러므로 객체 인식 시스템을 만들 때는 변환에 대한 불변성이 있어야 한다. 이 관점에서 합성곱 신경망은 전체의 공간 영역에서 이미지의 동일한 특징을 계산한다.

합성곱 구조는 하나의 정규화 메커니즘으로 볼 수 있다. 이 관점에서 합성곱 계층은 완전 연결 계층과 유사하지만 특정 크기의 행렬의 완전한 공간에서 가중치를 찾는 대신 정해진 크기의 합성곱을 설명할 수 있는 행렬로 제한하여, 보통 매우 작은 합성곱의 크기로 자유도(Degree of Freedom)를 줄인다.

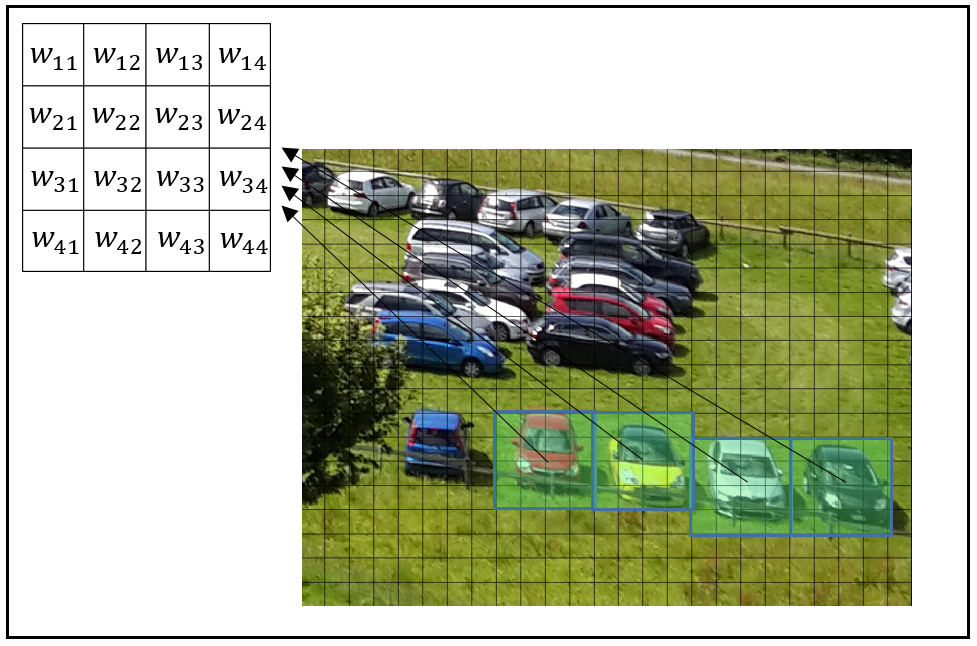
MNIST: TAKE 2

Convolution

합성곱 연산은 합성곱 신경망에서 계층이 연결되는 가장 기본적인 방법이다. tensorflow에서 기본 제공되는 conv2d() 함수를 사용한다.

tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding=’SAME’)

x는 입력 이미지 또는 네트워크를 따라 이전 합성곱 계층들에서 얻어진 아래층의 특징 맵 데이터이다. CNN모델에서는 합성곱 계층을 층층리 쌓아 올리는데 특정 맵(Feature Map)은 일반적으로 이들 각 계층의 출력을 칭하는 말이다. 필터는 합성곱 필터를 나타내는 네트워크의 학습된 가중치인 W로 매개변수화된다. 이는 아래의 그림에서 볼 수 있는 작은 ‘sliding window’들의 가중치의 집합이다.



이미지 전반에 적용된 동일한 ‘sliding window’의 합성곱 필터

이 연산의 결과는 x와 W의 형태에 따라 달라지며, 여기서는 4차원 값이다. 이미지 데이터 x는 다음과 같은 모양이다.

**[None, 28, 28, 1]**

이 표현은 이미지의 개수는 모르고(None), 각각은 28\*28 픽셀이며, 색 채널은 1개임을 나타낸다. 가중치 W의 형태는 다음과 같다.

**[5, 5, 1, 32]**

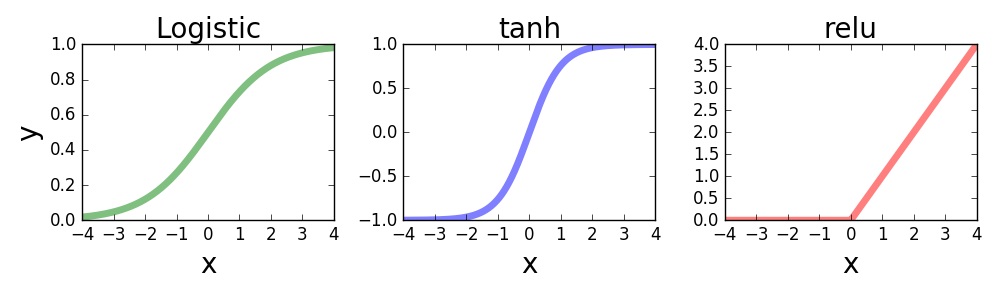
여기서 앞의 5\*5\*1은 이미지에서 합성곱에 사용될 작은 ‘window’의 크기를 나타내는데 여기서는 5\*5 영역이다. 다중 색상 채널(RGB)을 갖는 이미지에서 각 이미지는 RGB 값들의 3차원 tensor로 간주할 수 있지만, MNIST Database의 1채널 데이터는 2차원 tensor이며 합성곱 필터는 2차원 영역에 적용된다.

마지막 32는 특징 맵의 수이다. 즉 합성곱 계층의 가중치 집합을 여러 개를 가질 수 있으며, 이 경우는 32개가 있다는 뜻이다. 이러한 동일한 특징 다수를 계산하려는 것이며 그래서 여러 개의 합성곱 필터를 사용하려는 것이다.

strides 인수는 이미지 x 위에서 필터 W의 공간적 이동을 제어한다. 값 [1, 1, 1, 1]은 필터가 각 차원에서 한 픽셀 간격으로 입력 데이터에 적용되며 ‘완전한’ 합성곱에 대응한다는 뜻이다. 마지막으로 padding을 ‘SAME’으로 설정하면 연산 결과의 크기가 x의 크기와 같도록 x의 테두리에 padding이 적용된다.

Activation functions

일반적으로 선형 계층에 비선형 활성화 함수(Activation function)를 적용하는 것이 일반적이다. 선형 연산의 특성상 연달아 선형 연산을 연결해도 하나의 선형 연산식으로 표현되므로 중간에 비선형 활성화 함수를 사용하지 않으면 네트워크의 깊이가 모델의 표현력에 아무런 도움을 줄 수 없다.



흔히 사용되는 활성화 함수

Pooling

합성곱 계층 다음에는 pooling을 하는 것이 일반적인데, 기술적으로 pooling의 이미는 보통 각 특징 맵 내에서 어떤 지역적 집계 함수를 사용해 데이터의 크기를 줄이는 것을 뜻한다.

pooling의 배경에는 기술적인 이유와 이론적인 이유가 있는데, 기술적 측면에서 pooling은 차례로 처리되는 데이터의 크기를 줄인다. 이 과정으로 모델의 전체 매개변수의 수를 크게 줄일 수 있는데, 합성곱 계층 뒤에 완전 연결 계층을 사용할 때 특히 효과적이다.

pooling을 적용하는 이론적 근거는 계산된 특징이 이미지 내의 위치의 사소한 변화에 영향으 받지 않기를 바란다는 것이다. 예를 들어 이미지의 우상단에서 눈을 찾을 때, ‘눈을 찾는 특징’을 공간적으로 모아내면 어떤 형태의 불변성을 찾아내서 이미지 간의 공간적 변화를 극복할 수 있는 모델을 만들 수 있다.

tf.nn.max\_pool(x, ksize[1, 2, 2, 1], strides=[1, 2, 2, 1], padding=’SAME’)

max\_pool은 미리 정의한 크기의 각 영역에서 입력값의 최댓값을 출력한다. ksize 인수는 pooling의 크기를 정의하고 strides 인수는 합성곱 계층과 마찬가지로 x에서 움직이는 pooling조각이 얼마나 크게 건너뛸지 혹은 미끄러질지를 제어한다. 이를 2\*2 grid로 설정하면 pooling의 결과는 원본의 높이와 폭은 절반이 되고 크기는 1/4이 된다.

Dropout

dropout은 정규화를 위한 트릭이며 네트워크가 전체 뉴런에 걸쳐 학습된 표현을 배포하도록 강제한다. 학습 중 값을 0으로 세팅해서 계층 내의 유닛 중 임의의 사전에 설정된 부분을 ‘꺼버리는’ 방식으로 dropout시킨다. dropout된 뉴런은 각 연산마다 다르게 무작위로 선택되며, 네트워크가 dropout 이후에도 표현을 학습하도록 가테한다. 이 과정은 여러 네트워크의 ‘ensemble’학습으로도 간주할 수 있으며, 따라서 일반화를 강화한다고 할 수 있다.

예제에서 dropout을 적용하는 계층에 사용하는 유일한 인수는 keep\_prob로, 각 단계에서 학습을 계속할 뉴런의 비율이다.

tf.nn.dropout(layer, keep\_prob=keep\_prob)

이 값은 수정 가능해야 하므로 tf.placeholder를 사용해야 한다. 학습 단계에서는 0.5를 테스트 단계에서는 1.0을 넘겨줄 것이다.

The Model

(layers.py)

**| weight\_variable() |**

네트워크의 완전 연결 계층이나 합성곱 계층의 가중치를 지정한다. 표준편차가 0.1인 절단정큐분포를 사용하여 무작위로 초기화한다.

**| bias\_variable() |**

완전 연결 계층이나 합성곱 계층의 편향값을 정의한다. 이들 값은 모두 상수 0.1로 초기화된다.

**| conv2d() |**

일반적으로 사용할 합성곱을 정의한다. 건너뜀 없는 완전한 합성곱으로 입력과 같은 크기를 출력한다.

**| max\_pool\_2x2 |**

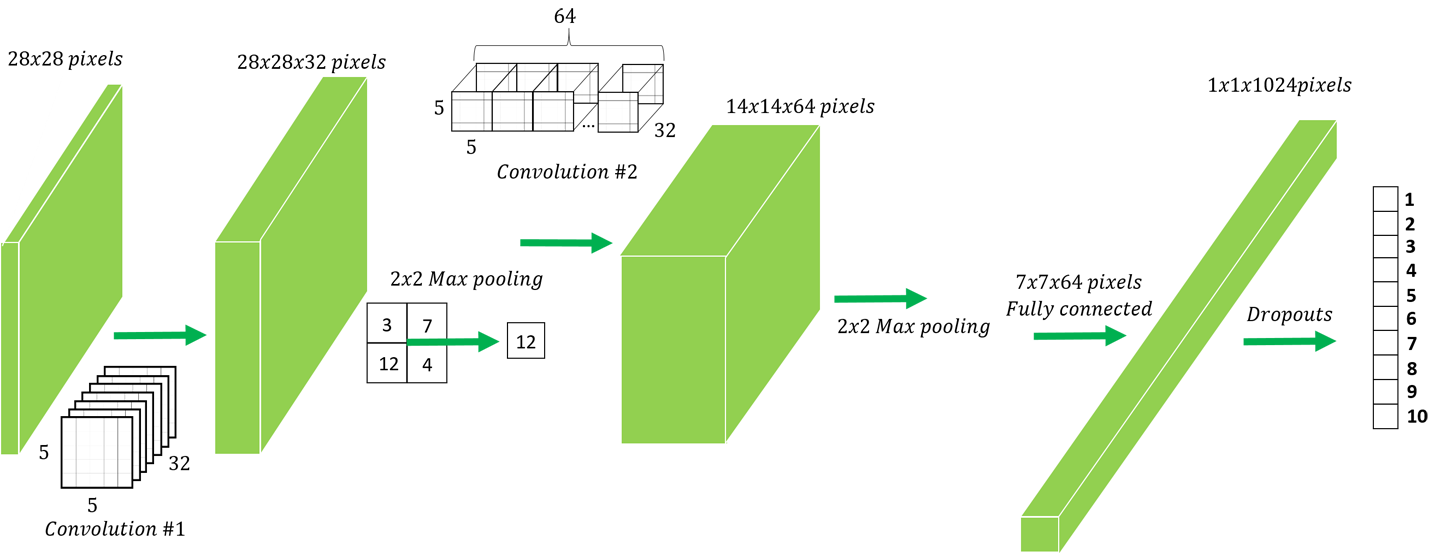
최댓값 pooling을 통해 높이와 넓이 차원을 각각 절반으로 줄여 전체적으로 특징 맵의 크기를 1/4로 줄인다.

**| conv\_layer() |**

사용할 실제 합성곱 계층으로 conv2d()에 정의된 선형 합성곱에 편향값을 더한 후 비선형 ReLU를 적용한다.

**| full\_layer() |**

편향을 적용한 표준적인 완전 연결 계층이다. ReLU를 적용하지 않았음을 주목하자. 이 함수를 사용해 최종 출력에 동일한 계층을 적용하는데, 그러면 비선형 부분이 필요하지 않다.



예제 합성곱 신경망의 구조

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

y\_ = tf.placeholder(tf.float32, shape=[None, 10])

x\_image = tf.reshape(x, [-1, 28, 28, 1])

conv1 = conv\_layer(x\_image, shape=[5, 5, 1, 32])

conv1\_pool = max\_pool\_2x2(conv1)

conv2 = conv\_layer(conv1\_pool, shape=[5, 5, 32, 64])

conv2\_pool = max\_pool\_2x2(conv2)

conv2\_flat = tf.reduce(conv2\_pool, [-1, 7\*7\*64])

full\_1 = tf.nn.relu(full\_layer(conv2\_flat, 1024))

keep\_prob = tf.placeholder(tf.float32)

full1\_drop = tf.nn.dropout(full\_1, keep\_prob=keep\_prob)

y\_conv = full\_layer(full1\_drop, 10)

먼저 이미지와 정답 레이블의 placeholder(x, y\_)를 정의한다. 그 후 이미지 데이터를 28\*28\*1 크기의 2차원 이미지 형식으로 재구성(reshape)한다. 이전 장의 MNIST 모델에서는 모든 픽셀 간의 관계를 고려하지 않고 처리하였으므로 데이터의 공간적인 요소를 고려할 필요가 없었지만 이미지를 대상으로 하는 합성곱 신경망 프레임워크에서는 공간적 의미의 활용이 가장 중요한 요소이다.

이어서 두 개의 연속된 합성곱 계층과 pooling 계층을 만든다. 각각 5\*5 합성곱과 32 및 64개의 특징 맵을 가지며, 그 뒤에 1024개의 유닛을 가진 하나의 완전 연결 계층이 이어진다. 완전 연결 계층에서는 더 이상 공간적 측면을 고려할 필요가 없으므로 완전 연결 계층을 적용하기 전에 이미지를 하나의 벡터 형태로 평탄화한다.

두 개의 합성곱과 pooling 계층에서 나오는 이미지의 크기는 7\*7\*64이다. 원래의 28\*28픽셀 이미지는 두 번의 pooling 과정에서 14\*14의 크기로 축소된 후 다시 7\*7의 크기로 축소된다. 두 번째 합성곱 계층에서 만들어진 특징 맵의 수는 64개이다. 모델에서 학습된 매개변수의 전체 개수는 완전 연결 계층에서는 큰 비율로 증가한다. max\_pool을 사용하지 않는다면 이 수는 16배로 증가했을 것이다. 마지막으로 출력은 데이터의 레이블의 개수에 해당하는 10개의 유닛을 가진 완전 연결 계층이다.

나머지는 2장의 첫 번째 MNIST모델과 동일하지만 살짝 바뀐 것들이 있다.

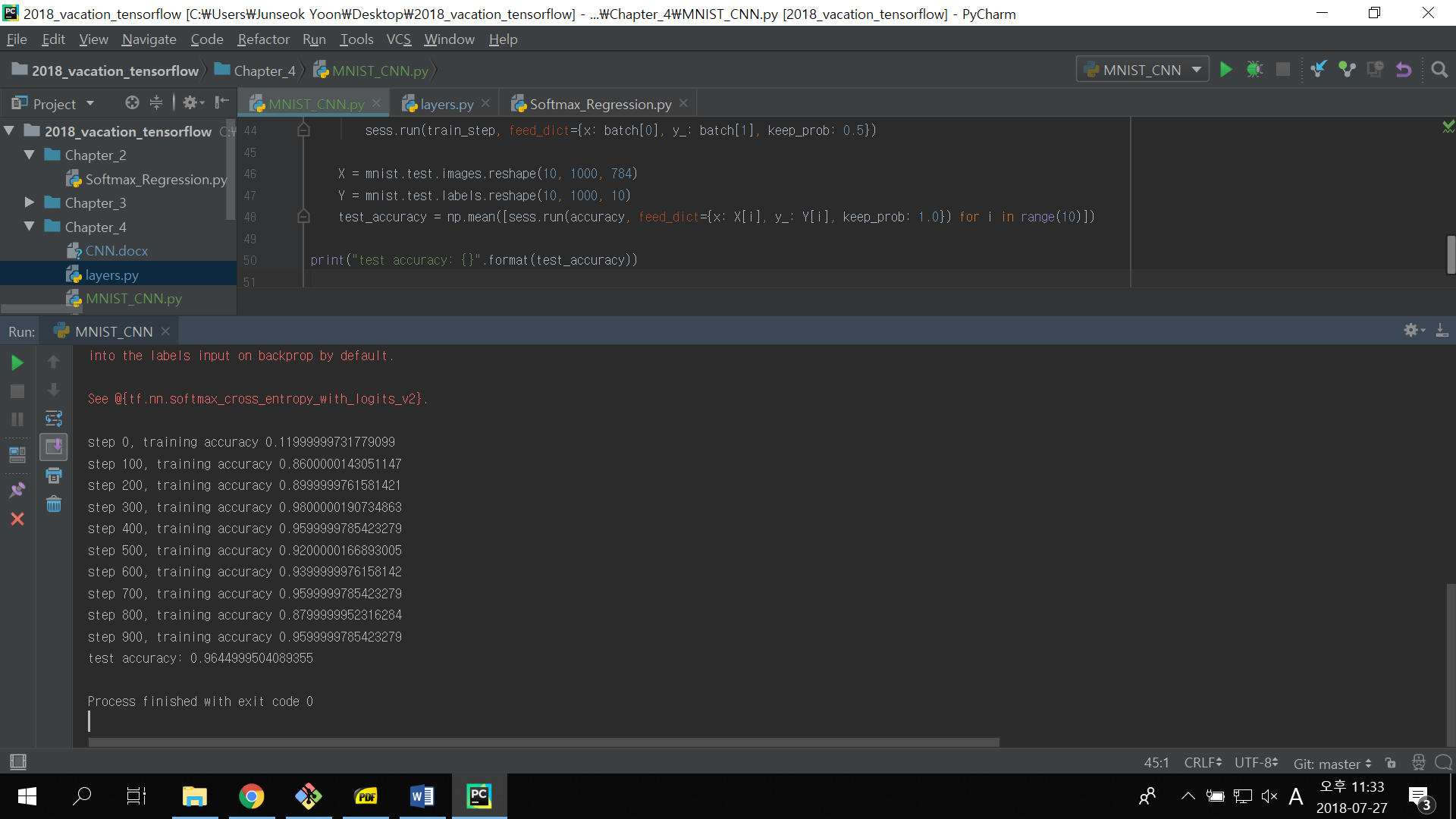
| train\_accuracy |

매 100회의 학습이 진행될 때마다 모델의 정확도를 출력한다. 학습 단계가 완료되기 전에 해당 중간 시점에서 학습 데이터에 대한 모델의 성능을 추정할 수 있다.

| test\_accuracy |

테스트 과정을 1,000개의 이미지로 구성된 10개의 블록으로 나누었다. 대량의 데이터를 사용하는 경우 매우 중요한 작업이다.

(MNIST\_CNN.py)



MNIST\_CNN 실행 결과

CIFAR10

Loading the CIFAR10 Dataset

(CIFAR10\_DataManager.py)

CIFAR10 데이터베이스에는 다음의 파일들이 있다.

* data\_batch\_1, data\_batch\_2, data\_batch\_3, data\_batch\_4, data\_batch\_5
* test\_batch
* batches\_meta
* readme.html

data\_batch\_X 파일은 학습 데이터가 포함된 직렬화된 데이터 파일들이며, test\_batch 파일은 테스트 데이터가 포함된 유사한 직렬화 파일이다. batches\_meta 파이은 숫자에서 의미 레이블로의 매핑을 담고 있다. readme.html 파일은 ‘CIFAR-10 데이터셋’ 웹 페이지의 사본이다.

CifarLoader 클래스와 CifarDataManager 클래스에서 사용되 두 유틸리티 함수에는 unpickle() 함수와 one\_hot() 함수가 있는데 unpickle() 함수는 이미지 데이터와 레이블 값에 해당하는 data와 labels 필드를 가진 dict를 반환한다. one\_hot() 함수는 0에서 9까지의 정수인 레이블 값을 길이가 10인 벡터로 바꾸는데, 이 벡터는 레이블 값에 해당하는 위치의 원소는 1이고 나머지는 모두 0인 벡터이다.

Simple CIFAR10 Models

(CIFAR10\_CNN.py)

앞에서 MNIST 데이터를 대상으로 좋은 결과를 보여준 모델로 시작해보자. MNIST 데이터는 28\*28 픽셀의 회색조 이미지로 구성되어 있었던 반면, CIFAR10 이미지는 32\*32 픽셀의 컬러 이미지로 구성되어 있다. 이 때문의 연산 그래프의 설정을 조금 바꿔야 한다.

(3conv\_CIFAR10\_CNN.py)

128개의 특징 맵과 드롭아웃을 가진 세 번째 합성곱 계층을 추가하고 완전 연결 계층의 유닛 수를 1024개에서 512개로 줄인 합성곱 신경망 모델.

지금까지의 방법과 널리 알려진 방법들 사이에는 여전히 큰 차이가 있다. 이 차이를 좁히기 위해서 시도할 수 있는 여러 가지 개별 작업들이 있다.

| 모델의 크기 |

CIFAR10 데이터나 이와 유사한 데이터의 처리는 더 많은 조정 가능한 매개변수를 가진 더 심층적인 네트워크를 적용해 개선할 수 있다.

| 계층이나 기법 추가 |

여기서 사용한 계층 이외에 지역 응답 정규화(Local Response Normalization)와 같은 다른 유형의 계층을 추가할 수 있다.

| 도메인 지식 |

도메인 지식을 활용한 전처리를 통해 결과를 개선할 수 있다. 이런 예제의 경우 전통적인 방식의 이미지 처리 과정을 적용할 수 있다.

| 데이터 확대 |

주어진 데이터를 사용하여 학습 데이터를 추가로 생성할 수 있다. 예를 들어서 개의 이미지는 가로 방향으로 뒤집어도 역시 개의 이미지이다. 살짝 이동하거나 회전하는 방법도 널리 사용된다.

| 성공적인 기법이나 아키텍처의 재활용 |

대부분의 공학 분야와 마찬가지로, 입증된 기법에서 출발해 이를 사용자의 요구에 맞도록 조정하는 것도 좋은 방법이다. 딥러닝에서는 사전 학습된 모델을 미세 조정하는 식으로 이루어지곤 한다.