텐서플로우

텐서플로우

- 사람의 학습 능력을 모방하기 위해 다양한 컴퓨터 알고리즘과 방법론을 연구하는 분야를 텐서플로우이라고 합니다.
- 선형 맞춤 (linear fitting)과 비선형 변환 (nonlinear transformation or activation)을 반복해 쌓아올린 구조
- 인공신경망은 데이터를 잘 구분할 수 있는 선들을 긋고 이 공간들을 잘 왜곡해 합하는 것을 반복하는 구조
- 예를 들어 컴퓨터가 사진 속에서 고양이를 검출해내야 한다고 생각해야 하는 경우에 '고양이'라는 추상적 이미지는 아마 선, 면, 형상, 색깔, 크기 등 다양한 요소들이 조합된 결과물일 것이고 이것은 아마 ' 선 30cm 이상은 고양이, 이하는 고양이 아님 ' , 또는 ' 갈색은 고양이, 빨간색은 고양이 아님 ' 처럼 간단한 선형 구분으로는 식별해 낼 수 없는 문제인데 딥러닝은 이 과제를 선 긋고 왜곡하고 합하고를 반복하며 복잡한 공간 속에서의 최적의 구분선을 만들어 내는 목적을 가짐
- 대충 선을 긋고 그것들을 살살 살살 움직여가며 구분 결과가 더 좋게 나오도록 선을 움직여 가면서 딥러닝은 아주 많은 데이터와 아주 오랜 시간의 최적화를 통해 데이터를 학습

텐서플로우

- 딥러닝 알고리즘은 대략적으로 다음과 같이 나눌 수 있습니다.
  - ✓ Unsupervised Learning을 기반으로 한 방법: 지도 학습(입력 데이터와 출력 데이터를 모두 이용)
  - ✓ Convolutional Neural Network의 다양한 변형들
  - ✓ Recurrent Neural Network와 게이트 유닛들
- 딥러닝 연구에 널리 사용되는 라이브러리가 텐서플로우
  - ✓ pip install tensorflow

텐서플로우

❖ 설치 확인을 위한 tensorflow 코드 실행

```
# 텐서플로우의 기본적인 구성을 익힙니다.
import tensorflow as tf

# tf.constant: 말 그대로 상수입니다.
hello = tf.constant('Hello, TensorFlow!')

a = tf.constant(10)
b = tf.constant(32)
c = tf.add(a, b)  # a + b 로도 쓸 수 있음
```

텐서플로우

위에서 변수와 수식들을 정의했지만, 실행이 정의한 시점에서 실행되는 것은 아닙니다.

다음처럼 Session 객제와 run 메소드를 사용할 때 계산이 됩니다.

따라서 모델을 구성하는 것과, 실행하는 것을 분리하여 프로그램을 깔끔하게 작성할 수 있습니다.

그래프를 실행할 세션을 구성합니다.

```
sess = tf.Session()
# sess.run: 설정한 텐서 그래프(변수나 수식 등등)를 실행합니다.
print(sess.run(hello))
print(sess.run([a, b, c]))

# 세션을 닫습니다.
sess.close()
```

텐서플로우

❖ 변수 선언을 위한 tensorflow 코드 실행

```
# 플레이스홀더와 변수의 개념을 익혀봅니다
import tensorflow as tf

# tf.placeholder: 계산을 실행할 때 입력값을 받는 변수로 사용합니다.
# None 은 크기가 정해지지 않았음을 의미합니다.
X = tf.placeholder(tf.float32, [None, 3])
print(X)

# X 플레이스홀더에 넣을 값 입니다.
# 플레이스홀더에서 설정한 것 처럼, 두번째 차원의 요소의 갯수는 3개 입니다.
x_data = [[1, 2, 3]]
```

텐서플로우

```
# tf.Variable: 그래프를 계산하면서 최적화 할 변수들입니다. 이 값이 바로 신경망을
좌우하는 값들입니다.
# tf.random_normal: 각 변수들의 초기값을 정규분포 랜덤 값으로 초기화합니다.
W = tf.Variable(tf.random_normal([3, 1]))
b = tf.Variable(tf.random_normal([1, 1]))

# 입력값과 변수들을 계산할 수식을 작성합니다.
# tf.matmul 처럼 mat* 로 되어 있는 함수로 행렬 계산을 수행합니다.
expr = tf.matmul(X, W) + b

sess = tf.Session()
# 위에서 설정한 Variable 들의 값들을 초기화 하기 위해
# 처음에 tf.global_variables_initializer 를 한 번 실행해야 합니다.
sess.run(tf.global_variables_initializer())
```

텐서플로우

```
print("=== x_data ===")
print(x_data)
print("=== W ===")
print(sess.run(W))
print("=== b ===")
print(sess.run(b))
print("=== expr ===")
# expr 수식에는 X 라는 입력값이 필요합니다.
# 따라서 expr 실행시에는 이 변수에 대한 실제 입력값을 다음처럼 넣어줘야합니다.
print(sess.run(expr, feed_dict={X: x_data}))


sess.close()
```

텐서플로우

# 위에서 변수와 수식들을 정의했지만, 실행이 정의한 시점에서 실행되는 것은 아닙니다.
# 다음처럼 Session 객제와 run 메소드를 사용할 때 계산이 됩니다.
# 따라서 모델을 구성하는 것과, 실행하는 것을 분리하여 프로그램을 깔끔하게 작성할 수 있습니다.
# 그래프를 실행할 세션을 구성합니다.
sess = tf.Session()
# sess.run: 설정한 텐서 그래프(변수나 수식 등등)를 실행합니다.
print(sess.run(hello))
print(sess.run([a, b, c]))

# 세션을 닫습니다.
sess.close()

텐서플로우

❖ 텐서플로우에서의 행렬

$$\begin{pmatrix} 1.0 & 2.0 & 3.0 \end{pmatrix} \times \begin{pmatrix} 2.0 \\ 2.0 \\ 2.0 \end{pmatrix}$$

텐서플로우

❖ 텐서 플로우 행렬

```
import tensorflow as tf

x = tf.constant([ [1.0,2.0,3.0] ])
w = tf.constant([ [2.0],[2.0],[2.0] ])
y = tf.matmul(x,w)
print (x.get_shape())

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y)

print(result)
```

텐서플로우

❖ 행렬

텐서플로우에서는 행렬을 차원에 따라서 다음과 같이 호칭한다.

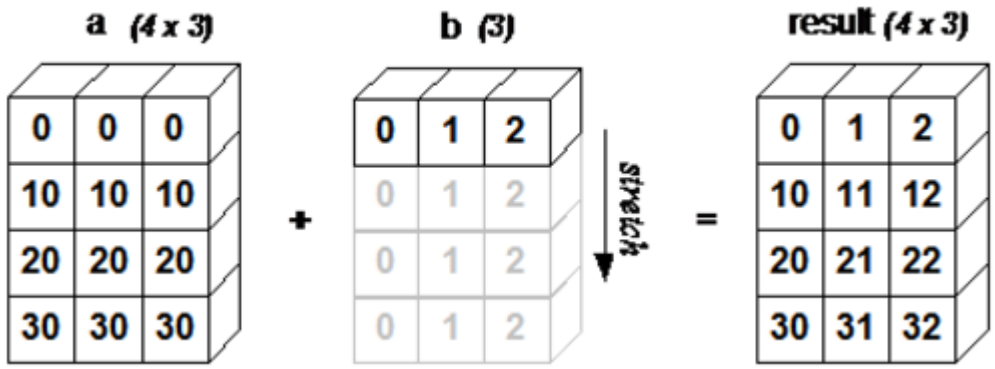
행렬이 아닌 숫자나 상수는 Scalar, 1차원 행렬을 Vector, 2차원 행렬을 Matrix, 3차원 행렬을 3-Tensor 또는 cube, 그리고 이 이상의 다차원 행렬을 N-Tensor라고 한다.

Rank	Math entity	Python example
0	Scalar (magnitude only)	`s = 483`
1	Vector (magnitude and direction)	`v = [1.1, 2.2, 3.3]`
2	Matrix (table of numbers)	`m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	3-Tensor (cube of numbers)	`t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]`
n	n-Tensor (you get the idea)	`....`

텐서플로우

❖ 브로드 캐스팅

브로드 캐스팅은 행렬 연산 (덧셈,뺄셈,곱셈)에서 차원이 맞지 않을때, 행렬을 자동으로 늘려줘서(Stretch) 차원을 맞춰주는 개념으로 늘리는 것은 가능하지만 줄이는 것은 불가능하다.



텐서플로우

❖ 텐서 플로우 행렬 연산

```
import tensorflow as tf
input_data = [
     [1,1,1],[2,2,2]
    ]
x = tf.placeholder(dtype=tf.float32,shape=[2,3])
w  =tf.Variable([[2],[2],[2]],dtype=tf.float32)
b  =tf.Variable([4],dtype=tf.float32)
y = tf.matmul(x,w)+b
print(x.get_shape())

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y,feed_dict={x:input_data})
print(result)
```

텐서플로우

❖ 텐서 플로우를 이용한 회귀 분석

```
# X 와 Y 의 상관관계를 분석하는 기초적인 선형 회귀 모델을 만들고 실행해봅니다.
import tensorflow as tf
x_data = [1, 2, 3]
y_data = [2, 4, 6]
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

# name: 나중에 텐서보드등으로 값의 변화를 추적하거나 살펴보기 쉽게 하기 위해
이름을 붙여줍니다.
X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")
print(X)
print(Y)
```

텐서플로우

```
# X 와 Y 의 상관 관계를 분석하기 위한 가설 수식을 작성합니다.
# y = W * x + b
# W 와 X 가 행렬이 아니므로 tf.matmul 이 아니라 기본 곱셈 기호를 사용했습니다.
hypothesis = W * X + b

# 손실 함수를 작성합니다.
# mean(h - Y)^2 : 예측값과 실제값의 거리를 비용(손실) 함수로 정합니다.
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# 텐서플로우에 기본적으로 포함되어 있는 함수를 이용해 경사 하강법 최적화를 수
행합니다.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
# 비용을 최소화 하는 것이 최종 목표
train_op = optimizer.minimize(cost)
```

텐서플로우

```
# 세션을 생성하고 초기화합니다.
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # 최적화를 100번 수행합니다.
    for step in range(100):
        # sess.run 을 통해 train_op 와 cost 그래프를 계산합니다.
        # 이 때, 가설 수식에 넣어야 할 실제값을 feed_dict 을 통해 전달합니다.
        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y: y_data})
        print(step, cost_val, sess.run(W), sess.run(b))
    # 최적화가 완료된 모델에 테스트 값을 넣고 결과가 잘 나오는지 확인해봅니다.
    print("₩n=== Test ===")
    print("X: 5, Y:", sess.run(hypothesis, feed_dict={X: 5}))
    print("X: 2.5, Y:", sess.run(hypothesis, feed_dict={X: 2.5}))
```

텐서플로우

- 군집화(clustering)
  - ✓ 입력 데이터의 분포 특성(입력값의 유사성)을 분석하여 임의의 복수 개의 그룹으로 나누는 것
  - ✓ 클래스에 대한 정보 없이 단순히 입력 값만 제공 ->{xi}
- K 평균 알고리즘
  - ✓ 군집화 문제를 풀기 위한 자율 학습 알고리즘
  - ✓ 이 알고리즘의 결과는 중심(centroid)라고 부르는 K개의 점으로 이들은 서로 다른 그룹의 중심점을 나타내며 데이터들은 K개의 군집 중 하나에만 속할 수 있습니다.
  - ✓ 알고리즘의 구현 과정
    - 초기단계: K개 중심의 초기 집합을 결정
    - 할당단계: 각 데이터를 가까운군집에 할당
    - 업데이트단계: 각 그룹에 대해 새로운 중심을 계산

텐서플로우

❖ 텐서 플로우를 이용한 군집화

```
import numpy as np
import pandas as pd
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt

#1000개의 데이터를 난수로 생성합니다. 대략 절반 정도는 평균:0.5, 표준편차:0.6의 x값과 평균:0.3, 표준편차:0.9의 y값을 가지고 나머지 절만 정도는 평균:2.5, 표준편차:0.4의 x값과 평균:0.8, 표준편차:0.5의 y값을 가집니다.
num_vectors = 1000
num_clusters = 4
num_steps = 100
vector_values = []
```

텐서플로우

```
for i in range(num_vectors):
  if np.random.random() > 0.5:
    vector_values.append([np.random.normal(0.5, 0.6),
                          np.random.normal(0.3, 0.9)])
  else:
    vector_values.append([np.random.normal(2.5, 0.4),
                          np.random.normal(0.8, 0.5)])
#vector_values 의 2차원 배열의 값을 각각 데이터프레임의 컬럼으로 지정합니다. 시본으로 그래
프를 그립니다.
df = pd.DataFrame({"x": [v[0] for v in vector_values],
                   "y": [v[1] for v in vector_values]})
sns.lmplot("x", "y", data=df, fit_reg=False, size=7)
plt.show()
```

텐서플로우

#vector_values를 사용하여 constant를 만들고 초기 센트로이드 세개를 랜덤하게 선택합니다. 그런 후에 vectors, centroids 텐서에 각각 차원을 추가합니다.

```
vectors = tf.constant(vector_values)
centroids = tf.Variable(tf.slice(tf.random_shuffle(vectors), [0,0], [num_clusters,-1]))
expanded_vectors = tf.expand_dims(vectors, 0)
expanded_centroids = tf.expand_dims(centroids, 1)
print(expanded_vectors.get_shape())
print(expanded_centroids.get_shape())
```

#각 데이터 포인트에서 가장 가까운 센트로이드의 인덱스를 계산합니다.

```
#거리 계산
distances = tf.reduce_sum(tf.square(tf.subtract(expanded_vectors, expanded_centroids)), 2)
assignments = tf.argmin(distances, 0)
```

텐서플로우

```
#각 클러스터의 평균 값을 계산하여 새로운 센트로이드를 구합니다.
means = tf.concat([
  tf.reduce_mean(
      tf.gather(vectors,
                tf.reshape(
                  tf.where(
                    tf.equal(assignments, c)
                  ),[1,-1])
                ),reduction_indices=[1])
  for c in range(num_clusters)], 0)
```

텐서플로우

```
update_centroids = tf.assign(centroids, means)

#변수를 초기화하고 세션을 시작합니다.
init_op = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init_op)

#100번의 반복을 하여 센트로이드를 계산하고 결과를 출력합니다.
for step in range(num_steps):
    _, centroid_values, assignment_values = sess.run([update_centroids, centroids, assignments])

print("centroids")
print(centroid_values)
```

텐서플로우

```
#vector_values 데이터를 클러스터에 따라 색깔을 구분하여 산포도를 그립니다.
data = {"x": [], "y": [], "cluster": []}
for i in range(len(assignment_values)):
    data["x"].append(vector_values[i][0])
    data["y"].append(vector_values[i][1])
    data["cluster"].append(assignment_values[i])
df = pd.DataFrame(data)
sns.lmplot("x", "y", data=df,
           fit_reg=False, size=7,
           hue="cluster", legend=False)
plt.show()
```

텐서플로우

- 분류(classification)
  - ✓ 주어진 데이터 집합을 이미 정의된 몇 개의 클래스로 구분하는 문제
  - ✓ 입력 데이터와 각 데이터의 클래스 라벨이 함께 제공 -> {xi, y(xi)}

텐서플로우

❖ 텐서 플로우를 이용한 분류

털과 날개가 있는지 없는지에 따라, 포유류인지 조류인지 분류하는 신경망 모델을 만들어봅니다.

```
import tensorflow as tf
import numpy as np
# [털, 날개]
x_data = np.array(
    [[0, 0], [1, 0], [1, 1], [0, 0], [0, 1], [1, 1]])
# [기타, 포유류, 조류]
y_data = np.array([
    [1, 0, 0],  # 기타
    [0, 1, 0],  # 포유류
    [0, 0, 1],  # 조류
    [1, 0, 0],
    [1, 0, 0],
    [0, 0, 1]
])
```

텐서플로우

```
# 신경망 모델 구성
######
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

# 신경망은 2차원으로 [입력층(특성), 출력층(레이블)] -> [2, 3] 으로 정합니다.
W = tf.Variable(tf.random_uniform([2, 3], -1., 1.))
# 편향을 각각 각 레이어의 아웃풋 갯수로 설정합니다.
# 편향은 아웃풋의 갯수, 즉 최종 결과값의 분류 갯수인 3으로 설정합니다.
b = tf.Variable(tf.zeros([3]))

# 신경망에 가중치 W과 편향 b을 적용합니다
L = tf.add(tf.matmul(X, W), b)
# 가중치와 편향을 이용해 계산한 결과 값에
# 텐서플로우에서 기본적으로 제공하는 활성화 함수인 ReLU 함수를 적용합니다.
L = tf.nn.relu(L)
```

텐서플로우

```
# 마지막으로 softmax 함수를 이용하여 출력값을 사용하기 쉽게 만듭니다
# softmax 함수는 다음처럼 결과값을 전체합이 1인 확률로 만들어주는 함수입니다.
# 예) [8.04, 2.76, -6.52] -> [0.53 0.24 0.23]
model = tf.nn.softmax(L)

# 신경망을 최적화하기 위한 비용 함수를 작성합니다.
# 각 개별 결과에 대한 합을 구한 뒤 평균을 내는 방식을 사용합니다.
# 전체 합이 아닌, 개별 결과를 구한 뒤 평균을 내는 방식을 사용하기 위해 axis 옵션을 사용합니
다.
# axis 옵션이 없으면 -1.09 처럼 총합인 스칼라값으로 출력됩니다.
# 즉, 이것은 예측값과 실제값 사이의 확률 분포의 차이를 비용으로 계산한 것이며,
# 이것을 Cross-Entropy 라고 합니다.
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(model), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train_op = optimizer.minimize(cost)
```

텐서플로우

```
#########
# 신경망 모델 학습
######
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

for step in range(100):
    sess.run(train_op, feed_dict={X: x_data, Y: y_data})

    if (step + 1) % 10 == 0:
        print(step + 1, sess.run(cost, feed_dict={X: x_data, Y: y_data}))
```

텐서플로우

```
#########
# 결과 확인
# 0: 기타 1: 포유류, 2: 조류
######
# tf.argmax: 예측값과 실제값의 행렬에서 tf.argmax 를 이용해 가장 큰 값을 가져옵니다.
# 예) [[0 1 0] [1 0 0]] -> [1 0]
#     [[0.2 0.7 0.1] [0.9 0.1 0.]] -> [1 0]
prediction = tf.argmax(model, 1)
target = tf.argmax(Y, 1)
print('예측값:', sess.run(prediction, feed_dict={X: x_data}))
print('실제값:', sess.run(target, feed_dict={Y: y_data}))
is_correct = tf.equal(prediction, target)
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도: %.2f' % sess.run(accuracy * 100, feed_dict={X: x_data, Y: y_data}))
```

텐서플로우

❖ RNN(Recurrent Neural Network)

RNN에 대한 기본적인 아이디어는 순차적인 정보를 처리한다는 데 있다. 기존의 신경망 구조에서는 모든 입력(과 출력)이 각각 독립적이라고 가정했지만, 많은 경우에 이는 옳지 않은 방법이다. 한 예로, 문장에서 다음에 나올 단어를 추측하고 싶다면 이전에 나온 단어들을 아는 것이 큰 도움이 될 것이다. RNN이 recurrent 하다고 불리는 이유는 동일한 태스크를 한 시퀀스의 모든 요소마다 적용하고, 출력 결과는 이전의 계산 결과에 영향을 받기 때문이다. 다른 방식으로 생각해 보자면, RNN은 현재 계산된 결과에 대한 "메모리" 정보를 갖고 있다고 볼 수도 있다.

텐서플로우

```
# 챗봇, 번역, 이미지 캡셔닝등에 사용되는 시퀀스 학습/생성 모델인 Seq2Seq 을 구현해봅니다.
# 영어 단어를 한국어 단어로 번역하는 프로그램을 만들어봅니다.
import tensorflow as tf
import numpy as np

# S: 디코딩 입력의 시작을 나타내는 심볼
# E: 디코딩 출력을 끝을 나타내는 심볼
# P: 현재 배치 데이터의 time step 크기보다 작은 경우 빈 시퀀스를 채우는 심볼
#    예) 현재 배치 데이터의 최대 크기가 4 인 경우
#        word -> ['w', 'o', 'r', 'd']
#        to   -> ['t', 'o', 'P', 'P']
char_arr = [c for c in 'SEPabcdefghijklmnopqrstuvwxyz단어나무놀이다소녀키스사랑남자']
num_dic = {n: i for i, n in enumerate(char_arr)}
dic_len = len(num_dic)
```

텐서플로우

```
# 영어를 한글로 번역하기 위한 학습 데이터
seq_data = [['word', '단어'], ['wood', '나무'],
            ['game', '놀이'], ['girl', '소녀'],
            ['kiss', '키스'], ['love', '사랑']]

def make_batch(seq_data):
    input_batch = []
    output_batch = []
    target_batch = []
```

텐서플로우

```
 for seq in seq_data:
        # 인코더 셀의 입력값. 입력단어의 글자들을 한글자씩 떼어 배열로 만든다.
        input = [num_dic[n] for n in seq[0]]
        # 디코더 셀의 입력값. 시작을 나타내는 S 심볼을 맨 앞에 붙여준다.
        output = [num_dic[n] for n in ('S' + seq[1])]
        # 학습을 위해 비교할 디코더 셀의 출력값. 끝나는 것을 알려주기 위해 마지막에 E 를 붙인
다.
        target = [num_dic[n] for n in (seq[1] + 'E')]

        input_batch.append(np.eye(dic_len)[input])
        output_batch.append(np.eye(dic_len)[output])
        # 출력값만 one-hot 인코딩이 아님 (sparse_softmax_cross_entropy_with_logits 사용)
        target_batch.append(target)

    return input_batch, output_batch, target_batch
```

텐서플로우

```
#########
# 옵션 설정
######
learning_rate = 0.01
n_hidden = 128
total_epoch = 100
# 입력과 출력의 형태가 one-hot 인코딩으로 같으므로 크기도 같다.
n_class = n_input = dic_len
#########
# 신경망 모델 구성
######
# Seq2Seq 모델은 인코더의 입력과 디코더의 입력의 형식이 같다.
# [batch size, time steps, input size]
enc_input = tf.placeholder(tf.float32, [None, None, n_input])
dec_input = tf.placeholder(tf.float32, [None, None, n_input])
# [batch size, time steps]
targets = tf.placeholder(tf.int64, [None, None])
```

텐서플로우

```
# 인코더 셀을 구성한다.
with tf.variable_scope('encode'):
    enc_cell = tf.contrib.rnn.BasicRNNCell(n_hidden)
    enc_cell = tf.contrib.rnn.DropoutWrapper(enc_cell, output_keep_prob=0.5)

    outputs, enc_states = tf.nn.dynamic_rnn(enc_cell, enc_input,
                                            dtype=tf.float32)
# 디코더 셀을 구성한다.
with tf.variable_scope('decode'):
    dec_cell = tf.contrib.rnn.BasicRNNCell(n_hidden)
    dec_cell = tf.contrib.rnn.DropoutWrapper(dec_cell, output_keep_prob=0.5)

    # Seq2Seq 모델은 인코더 셀의 최종 상태값을
    # 디코더 셀의 초기 상태값으로 넣어주는 것이 핵심.
    outputs, dec_states = tf.nn.dynamic_rnn(dec_cell, dec_input,
                                            initial_state=enc_states,
                                            dtype=tf.float32)
```

텐서플로우

```
model = tf.layers.dense(outputs, n_class, activation=None)


cost = tf.reduce_mean(
            tf.nn.sparse_softmax_cross_entropy_with_logits(
                logits=model, labels=targets))

optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

##########
# 신경망 모델 학습
######
sess = tf.Session()
sess.run(tf.global_variables_initializer())

input_batch, output_batch, target_batch = make_batch(seq_data)
```

텐서플로우

```
for epoch in range(total_epoch):
    _, loss = sess.run([optimizer, cost],
                       feed_dict={enc_input: input_batch,
                                  dec_input: output_batch,
                                  targets: target_batch})

    print('Epoch:', '%04d' % (epoch + 1),
          'cost =', '{:.6f}'.format(loss))

print('최적화 완료!')
```

텐서플로우

```
#########
# 번역 테스트
######
# 단어를 입력받아 번역 단어를 예측하고 디코딩하는 함수
def translate(word):
    # 이 모델은 입력값과 출력값 데이터로 [영어단어, 한글단어] 사용하지만,
    # 예측시에는 한글단어를 알지 못하므로, 디코더의 입출력값을 의미 없는 값인 P 값으로 채운다.
    # ['word', 'PPPP']
    seq_data = [word, 'P' * len(word)]
    input_batch, output_batch, target_batch = make_batch([seq_data])
    # 결과가 [batch size, time step, input] 으로 나오기 때문에,
    # 2번째 차원인 input 차원을 argmax 로 취해 가장 확률이 높은 글자를 예측 값으로 만든다.
    prediction = tf.argmax(model, 2)
```

텐서플로우

```
result = sess.run(prediction,
                  feed_dict={enc_input: input_batch,
                             dec_input: output_batch,
                             targets: target_batch})

  # 결과 값인 숫자의 인덱스에 해당하는 글자를 가져와 글자 배열을 만든다.
  decoded = [char_arr[i] for i in result[0]]

  # 출력의 끝을 의미하는 'E' 이후의 글자들을 제거하고 문자열로 만든다.
  end = decoded.index('E')
  translated = ''.join(decoded[:end])

  return translated
```

텐서플로우

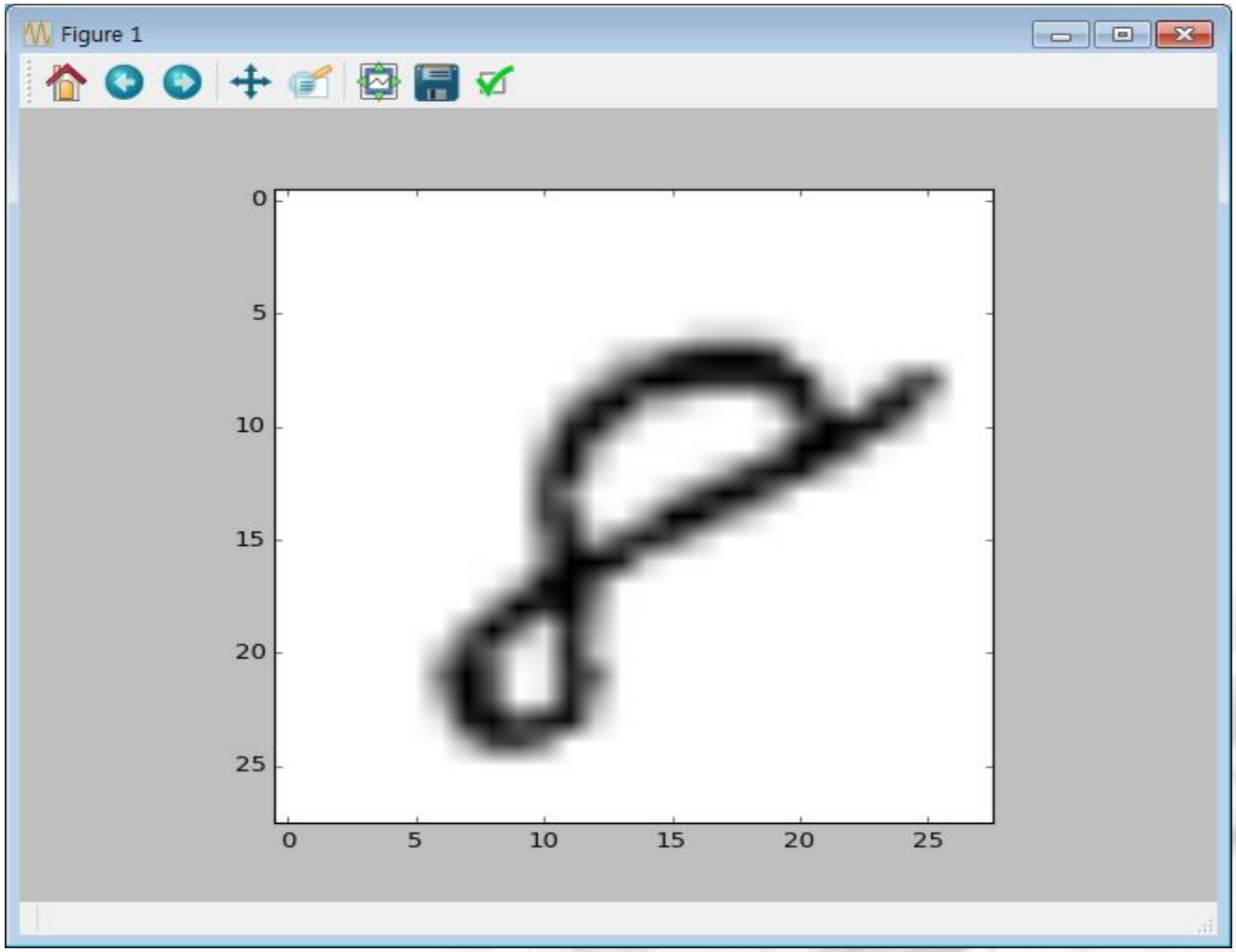
```
print('₩n=== 번역 테스트 ===')

print('word ->', translate('word'))
print('wodr ->', translate('wodr'))
print('love ->', translate('love'))
print('loev ->', translate('loev'))
print('abcd ->', translate('abcd'))
```

텐서플로우

- MNIST 데이터셋
  - ✓ 훈련용 데이터 55000 개 및 테스트용 10000로 이루어진 손글씨 숫자의 흑백 이미지 데이터
  - ✓ http://yann.lecun.com/exdb/mnist에서 다운로드 가능
  - ✓ 패턴 인식을 공부하기 위한 샘플 데이터
  - ✓ 이미지 데이터에는 그 이미지가 어떤 숫자 인지를 나타내는 레이블 정보가 포함되어 있습니다.

텐서플로우



텐서플로우

❖ 이미지 확인

```
import numpy as np
import sys
import os
from array import array

from struct import *
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

#파일 읽기
fp_image = open('train-images.idx3-ubyte','rb')
fp_label = open('train-labels.idx1-ubyte','rb')
```

텐서플로우

```
#사용할 변수 초기화
img = np.zeros((28,28)) #이미지가 저장될 부분
lbl = [ [],[],[],[],[],[],[],[],[],[] ] #숫자별로 저장 (0 ~ 9)
d = 0
l = 0
index=0

s = fp_image.read(16)     #read first 16byte
l = fp_label.read(8)      #read first   8byte


#숫자 데이터를 읽어서 해당하는 데이터를 지정하고 출력
k=0 #테스트용 index
```

텐서플로우

```
#read mnist and show number
while True:
    s = fp_image.read(784) #784바이트씩 읽음
    l = fp_label.read(1) #1바이트씩 읽음
    if not s:
        break;
    if not l:
        break;
    index = int(l[0])
    print(k,":",index)
#unpack
    img = np.reshape( unpack(len(s)*'B',s), (28,28))
    lbl[index].append(img) #각 숫자영역별로 해당이미지를 추가
    k=k+1


print(img)
```

텐서플로우

```
plt.imshow(img,cmap = cm.binary) #binary형태의 이미지 설정
plt.show()

print(np.shape(lbl)) #label별로 잘 지정됬는지 확인

print("read done")
```

텐서플로우

```
m_img = []

for i in range(0,10):
    m_img.append( np.mean(lbl[i],axis=0) )

for i in range(0,10):
    plt.imshow(m_img[i],cmap = cm.binary)
    plt.show()
```

텐서플로우

❖ 이미지 인식 학습

```
#텐서플로우에서 제공하는 툴을 이용해 MNIST 데이터를 다운받습니다.
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
import tensorflow as tf
tf.convert_to_tensor(mnist.train.images).get_shape()

#가중치 텐서와 바이어스 텐서를 만듭니다.
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))

#훈련 이미지 데이터를 넣을 플레이스홀더와 소프트맥스 텐서를 만듭니다.
x = tf.placeholder("float", [None, 784])
y = tf.nn.softmax(tf.matmul(x,W) + b)
```

텐서플로우

#실제 레이블을 담기위한 텐서와 교차 엔트로피 방식을 이용하는 그래디언트 디센트 방식을 선택합니다.

```
y_ = tf.placeholder("float", [None,10])
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

#변수를 초기화하고 세션을 시작합니다.
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

텐서플로우

```
#1000의 반복을 수행하고 결과를 출력합니다. 최종 정확도는 91% 정도 입니다.
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    if i % 100 == 0:
        print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

Windows 배포

- pyinstaller를 이용하면 윈도우의 exe 파일을 만들 수 있습니다.
- 위 라이브러리 설치 후 아래 명령 실행

pyinstaller 파일이름.py

pyinstaller -w 파일이름.py : command 창 제거

pyinstaller --onefile -w 파일이름.py : 하나의 파일로 만들기