Tensorflow

Tensorflow란

Tensorflow는 구글 내 연구와 제품개발을 위한 목적으로 구글 브레인팀이 만든 오픈소스 라이브러리.
데이터 플로우 프로그래밍과 뉴럴 네트워크같은 기계학습 응용프로그램에도 사용된다.

Tensorflow는 google에서 딥러닝을 할 수 있는 오픈소스 라이브러리를 공개한 funtion들의 집합으로 만들어진 것이다.
딥러닝 하는 사람들은 대부분 tensorflow 라이브러리를 가지고 한다.

AI > 머신러닝 > 딥러닝

TensorFlow : Machine Learning for Everyone

텐서플로우 소개영상을 보면 중간에 inception3 나 v3모델같은게 나오는데,
구글이 텐서플로우 라이브러리만 공개하는 것이 아니라
각종 모델,

이를테면 image calassificaiton model을 공개 해두었다.

이 모델들은 이미지 구분을 잘 할 수 있어서,
인터넷으로 내 맘대로 이미지를 가져와서
파이썬에 그 다운받은 모델을 통과시키면
이 이미지는 꽃이다, 혹은 무슨 종류의 꽃이다 이런 것까지 잘 예측을 해준다.

사람들이 처음부터 자신의 데이터들을 학습시켜서
이미지를 예측하게 만들기는 굉장히 어려운데,
구글에서는 이것을 이미 오픈해두었다.

Inception3가 일반적인 이미지를 잘 예측해주는 모델의 이름인데,
그 모델에 더해서 내가 개인적으로 가지고 있는 어떤 이미지들로 학습을 시킨다.

그렇게 모델 뒷단에 학습을 시키면 그 다음 예측을 더 잘 해줄 수 있다.

처음부터 모델을 만드는 것 보다
구글에서 인셉션3 등 각종 예측을 잘 해주는 모델들을 다운받아서
re-training 시켜서 사용하는 것만 잘해도 tensorflow를 잘 활용할 수 있다.

Tensorflow를 왜 많이 사용하는가?

= > Language도 파이썬을 제일 많이 사용하는데, 쓰기 쉽게 되어있으니 많이 사용하는 것 이다.

Data Flow Graph

Tensorflow 란 ? 하면 데이터 플로우 그래프라고 보면 된다.

데이터를 흘려보내서 어떤 식으로 데이터가 흘러가서

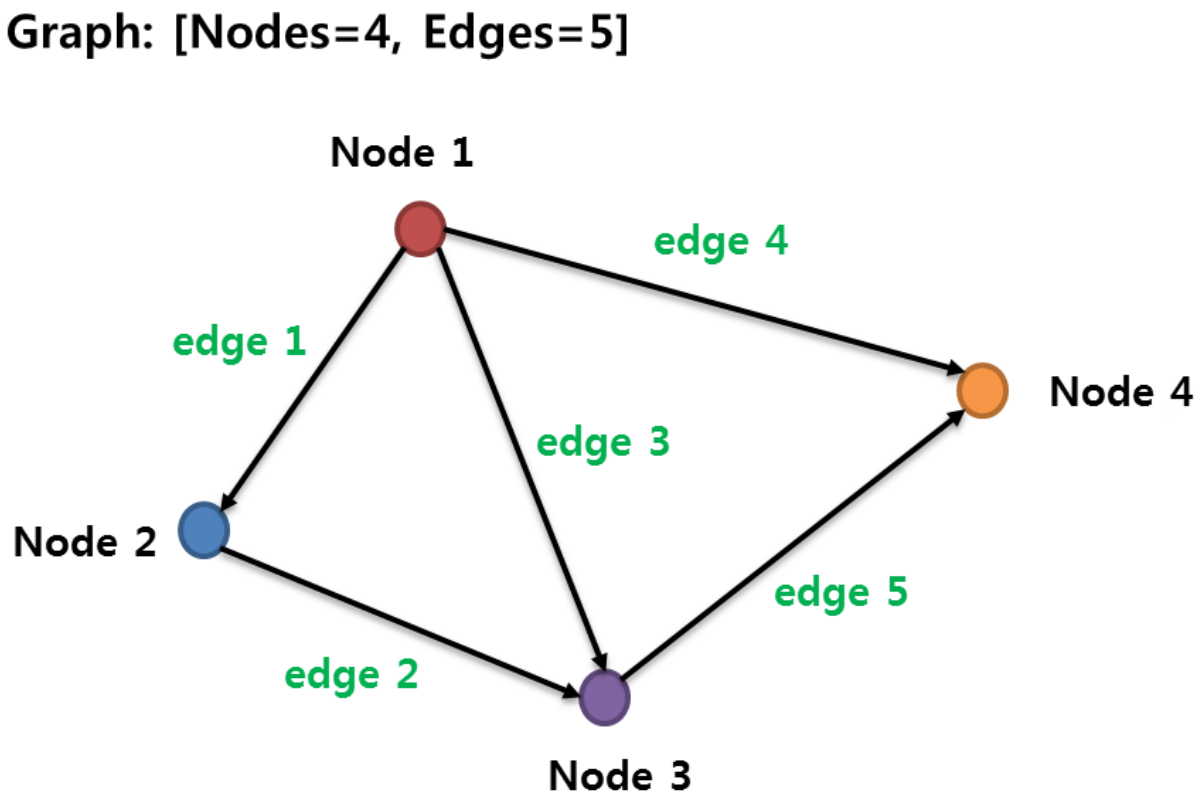
어떻게 계산되고, 이런 데이터가 흘러가는 그래프의 덩어리라고 보면 된다.

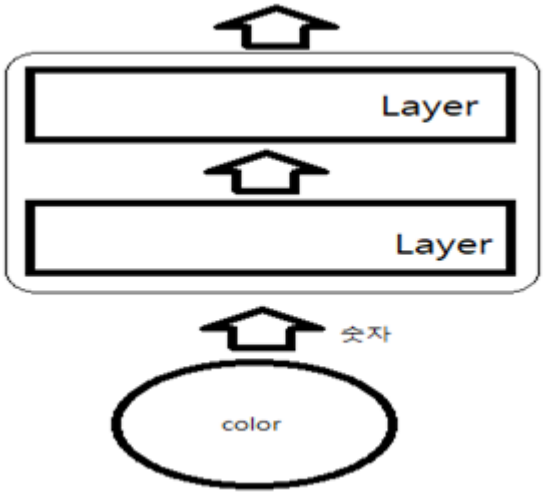
Data Flow Graph란 ?

- Tensorflow는 Data Flow Graph형태로 작동한다.
- Data Flow Graph 구성요소
  - 노드(Node) : 텐서를 처리하는 연산
  - 엣지(Edge) : 노드 사이를 이동하는 다차원 데이터배열(텐서, Tensor)를 나타낸다.
  - 텐서, 즉 데이터가 노드를 지나면서 연산이 이루어진다.
  - 텐서가 그래프상에서 이동하여 Tensorflow이다.

사실 이미지들은 숫자들의 덩어리이기 때문에 이 숫자들을 가지고 위로 올라가면서 쭉 계산을 해 숫자들이 계산되면서 위로 데이터가 흘러가서 마지막에 예측을 하는것이다.

이렇게 만든 것을 Data Flow Graph 라고 하고, 잘 예측할 수 있는 모델을 만들고자 한다.





Tensor란 무엇인가?

- Tensor는 배열(array)로서 어떤 값이든 될 수 있다.
- Tensor를 설명할 땐 랭크(Rank)와 모양 ()

`tf.Tensor` 객체의 **랭크**는 그 차원의 수입니다. 랭크의 동의어는 **order** 또는 **degree**, **n-dimension**입니다. 텐서플로의 랭크는 수학에서 사용하는 행렬의 랭크와는 다릅니다. 다음 표에서 알 수 있는 것처럼, 텐서플로의 각 랭크는 각각 다른 수학 개체(entity)에 해당됩니다.

랭크	수학 개체
0	스칼라(Scalar) (크기(magnitude)만)
1	벡터(Vector) (크기와 방향(direction))
2	행렬(Matrix) (숫자 표)
3	3-텐서 (숫자 큐브(cube))
n	n-텐서 (알 수 있을겁니다(you get the idea))

0차원 배열은 그냥 이름은 scalar라고 부르고,

1차원 배열음 vector 이런식으로 이름을 붙여놓은 것이다.

2차원 배열은 Matrix,

3차원 3차원 Tensor,

n차원은 n차원 Tensor이다.

- 모양(Shape)은 각각의 차원에 몇 개의 원소가 들어있는지를 의미한다.
- t = [[1,2,3],[4,5,6],[7,8,9]]이면 shape은 [3,3]이 된다.

배열의 Shape가 [] 이렇게 들어가 있는 것은 0차원이라 배열이 아니고 값이 하나인 것이다.

케라스 주요 특징

케라스는 아래 4가지의 주요 특징을 가지고 있다.

- 모듈화 (Modularity)
  - 케라스에서 제공하는 모듈은 독립적이고 설정 가능하며, 가능한 최소한의 제약사항으로 서로 연결될 수 있다. 모델은 시퀀스 또는 그래프로 이러한 모듈들을 구성한것.
  - 특히 신경망 층, 비용함수, 최적화기, 초기화기법, 활성화함수, 정규화기법은 모두 독립적인 모듈이며, 새로운 모델을 만들기 위해 이러한 모듈을 조합할 수 있다.
- 최소 주의(Minimalism)
  - 각 모듈은 땁고 간결하다.
  - 모든 코드는 한 번 훑어보는 것으로도 이해가능해야 한다.
  - 단 반복 속도와 혁신성에는 다소 떨어질수 있다.
- 쉬운 확장성
  - 새로운 클래스나 함수로 모듈을 아주 쉽게 추가할 수 있다.
  - 따라서 고급 연구에 필요한 다양한 표현을 할 수 있다.
- 파이썬 기반
  - Caffe 처럼 별도의 모델 설정 파일이 필요없으며 파이썬 코드로 모델들이 정의 된다.

케라스 기본 개념

케라스의 가장 핵심적인 데이터 구조는 바로 모델입니다. 케라스에서 제공하는 시퀀스 모델로 원하는 레이어를 쉽게 순차적으로 쌓을 수 있습니다. 다중 출력이 필요하는 등 좀 더 복잡한 모델을 구성하려면 케라스 함수 API를 사용하면 됩니다. 케라스로 딥러닝 모델을 만들 때는 다음과 같은 순서로 작성합니다. 다른 딥러닝 라이브러리와 비슷한 순서이지만 훨씬 직관적이고 간결합니다.

1. 데이터셋 생성하기
   - 원본 데이터를 불러오거나 시뮬레이션을 통해 데이터를 생성합니다.
   - 데이터로부터 훈련셋, 검증셋, 시험셋을 생성합니다.
   - 이 때 딥러닝 모델의 학습 및 평가를 할 수 있도록 포맷 변환을 합니다.
2. 모델 구성하기
   - 시퀀스 모델을 생성한 뒤 필요한 레이어를 추가하여 구성합니다.
   - 좀 더 복잡한 모델이 필요할때는 케라스 함수 API를 아용 합니다.
3. 모델 학습과정 설정하기
   - 학습하기 전에 학습에 대한 설정을 수행합니다.
   - 손실 함수 및 최적화 방법을 정의합니다.
   - 케라스에서는 compile() 함수를 사용합니다.
4. 모델 학습시키기
   - 훈련셋을 이용하여 구성한 모델로 학습시킵니다.
   - 케라스에서는 fit() 함수를 사용합니다.
5. 학습과정 살펴보기
   - 모델 학습 시 훈련셋, 검증셋의 손실 및 정확도를 측정합니다.
   - 반복횟수에 따른 손실 및 정확도 추이를 보면서 학습 상황을 판단합니다.

6. 모델 평가하기
    - 준비된 시험셋으로 학습한 모델을 평가합니다.
    - 케라스에서는 evaluate() 함수를 사용합니다.
7. 모델 사용하기
    - 임의의 입력으로 모델의 출력을 얻습니다.
    - 케라스에서는 predict() 함수를 사용합니다.

예시

```
# 0. 사용할 패키지 불러오기
from keras.utils import np_utils
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation

# 1. 데이터셋 생성하기
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 784).astype('float32') / 255.0
x_test = x_test.reshape(10000, 784).astype('float32') / 255.0
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

# 2. 모델 구성하기
model = Sequential()
model.add(Dense(units=64, input_dim=28*28, activation='relu'))
model.add(Dense(units=10, activation='softmax'))

# 3. 모델 학습과정 설정하기
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=[
'accuracy'])

# 4. 모델 학습시키기
hist = model.fit(x_train, y_train, epochs=5, batch_size=32)

# 5. 학습과정 살펴보기
print('## training loss and acc ##')
print(hist.history['loss'])
print(hist.history['acc'])

# 6. 모델 평가하기
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=32)
print('## evaluation loss and_metrics ##')
print(loss_and_metrics)

# 7. 모델 사용하기
xhat = x_test[0:1]
yhat = model.predict(xhat)
print('## yhat ##')
print(yhat)
```

```
Epoch 1/5
60000/60000 [==============================] - 2s - loss: 0.6861 - acc:
0.8214
Epoch 2/5
60000/60000 [==============================] - 2s - loss: 0.3472 - acc:
0.9021
Epoch 3/5
60000/60000 [==============================] - 2s - loss: 0.2972 - acc:
0.9159
Epoch 4/5
60000/60000 [==============================] - 2s - loss: 0.2671 - acc:
0.9243
Epoch 5/5
60000/60000 [==============================] - 2s - loss: 0.2444 - acc:
0.9311
## training loss and acc ##
[0.68605235149065658, 0.34716726491451261, 0.29719433775146803, 0.2670892
3313419025, 0.24437546383142472]
[0.82138333333333335, 0.90213333333333334, 0.91591666666666671, 0.9243333
3333333334, 0.93113333333333337]
 9728/10000 [============================>.] - ETA: 0s## evaluation loss
 and_metrics ##
[0.229048886179924, 0.93520000000000003]
## yhat ##
[[  8.87035931e-05   1.47768702e-07   9.51653055e-04   3.75617994e-03
    2.82607380e-06   4.90140701e-05   2.10885389e-08   9.94055033e-01
    6.05004097e-05   1.03587494e-03]]
```

간단한 모델 만들기

Sequential모델
케라스에서는 **층(layer)**을 조합하여 모델**(model)**을 만듭니다