TensorFlow 기초 및 실습

OSIA 동계 단기강좌

김지섭

2017년 2월 14일

TensorFlow 소개

- TensorFlow는 무엇인가?
  - 기본적으로, 딥러닝 프레임워크
  - 보다 일반적으로, 추상화된 GPU 컴퓨팅 프레임워크
- 핵심적인 기능들
  - GPU 컴퓨팅을 위한 추상화된 API
  - Symbolic 표현과 자동 미분
  - 다중 서버 분산화, 다중 GPU 분산화 지원
  - 손쉬운 시각화 도구 (TensorBoard)

TensorFlow의 워크플로우

- **Computational Graph 만들기**
  - Symbolic 변수, 입력, 그리고 상수를 선언.
  - 계산하고자 하는 수식을 위에서 선언한 원소들을 이용하여 나타냄.
  - 필요에 따라 유틸리티 함수들을 추가함(변수 값 업데이트, 로깅 등).
- **Computational Graph 실행**
  - 새로운 세션 열기.
  - 모든 Symbolic 변수의 값을 초기화 (GPU에 값을 로딩하는 과정).
  - 세션을 이용하여 Graph를 실행 및 계산.

TensorFlow 기본

- 세 가지 Symbolic 원소들:
  - **tf.placeholder**: 값을 가지고 있지 않음. 보통 입력을 나타내는데 사용.
  - **tf.constant**: 상수 값을 가지고 있음.
  - **tf.Variable**: 값을 가지고 있으나 계산 과정에서 값이 바뀔 수 있음. 보통 학습 파라미터를 나타내는데 사용.
- 여러 연산을 이용하여 수식을 구성:
  - 예) +-*/, tf.exp, tf.log, tf.sin, ...

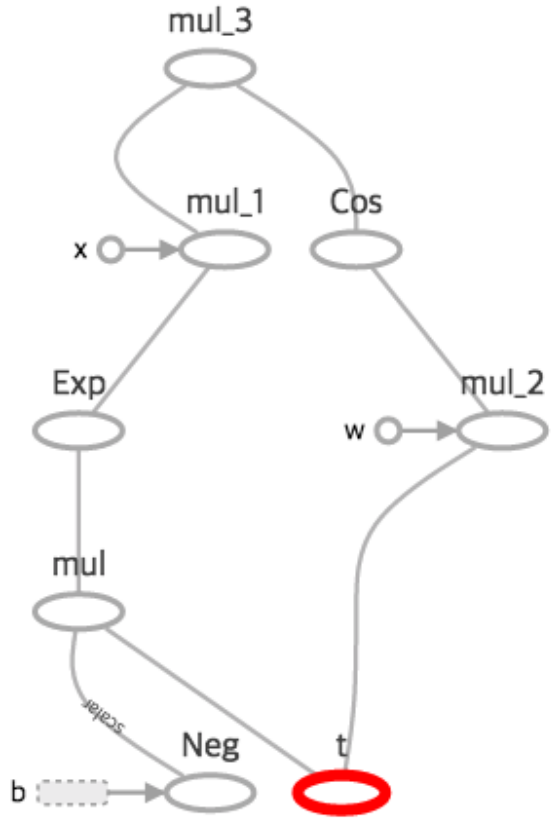
TensorFlow 기본 (예제)

- 예제 수식

$$y = A\exp(-bt)cos(wt)$$

TensorFlow 코드

```
t=tf.placeholder(tf.float32, name='t')
b=tf.Variable(0.0, name='b')
w=tf.constant(2.0, name='w')
A=3.0
y=A*tf.exp(-b*t)*tf.cos(w*t)
```



자동 미분(Automatic Differentiation)

- 자동 미분(Auto-Diff)은 Symbolic 미분이나 수치 미분과는 다르다.
- 자동 미분은 수식이 단순한(exp, sin 등 도함수를 알고 있는) 함수로 쪼갤 수 있을 때까지 **반복적으로 Chain Rule** 을 적용하여 미분을 계산한다.
- 이는 정확한 미분값을 계산해주지만 **도함수의 수식을 도출해주지는 못한다**(Symbolic 미분과의 차이점).
- 참고: https://en.wikipedia.org/wiki/Automatic_differentiation

TensorBoard를 통한 시각화

- TensorBoard는 웹에 기반한 시각화 도구이다.
- 기본적으로, Computational Graph를 보여준다.
- 추가적으로, 계산시(모델 학습시) 이벤트를 로깅하였다면 이에 대해 각 타입에 맞는 그래프를 보여준다. 가능한 타입은 다음과 같다.
  - 이미지
  - 오디오
  - 스칼라 값의 추이
  - 확률 분포의 추이
- 예제코드는 다음에서 찾을 수 있다. *here*

TensorBoard를 이용한 시각화 (예제)

- 예제

TensorFlow 코드

```
# Add loggers
tf.summary.image('input', img_data)
tf.summary.scalar('loss', loss_val)
tf.summary.histogram('activations', act_dist)

# Merge all loggers into a single node in the graph
merged = tf.summary.merge_all()
writer = tf.summary.FileWriter('/tmp/logdir', sess.graph)

# Evaluate the logging values and write them.
summary = sess.run(merged)
writer.add_summary(summary)
writer.close()
```

분산 컴퓨팅 예제

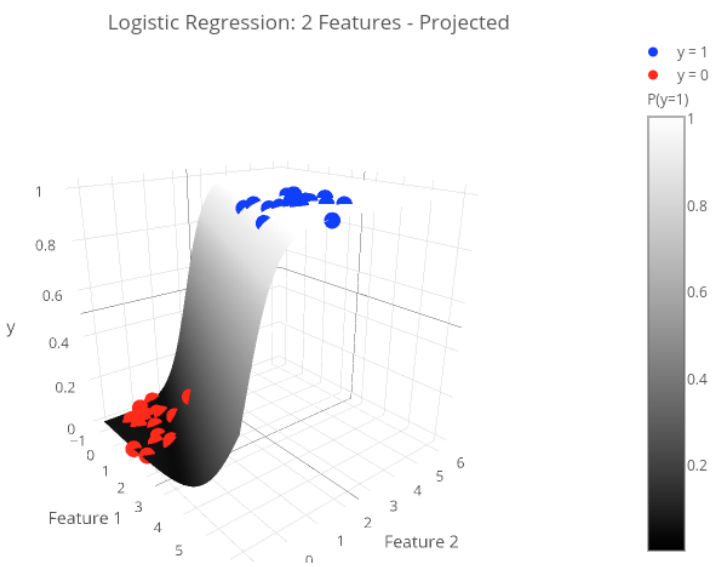
TensorFlow 코드

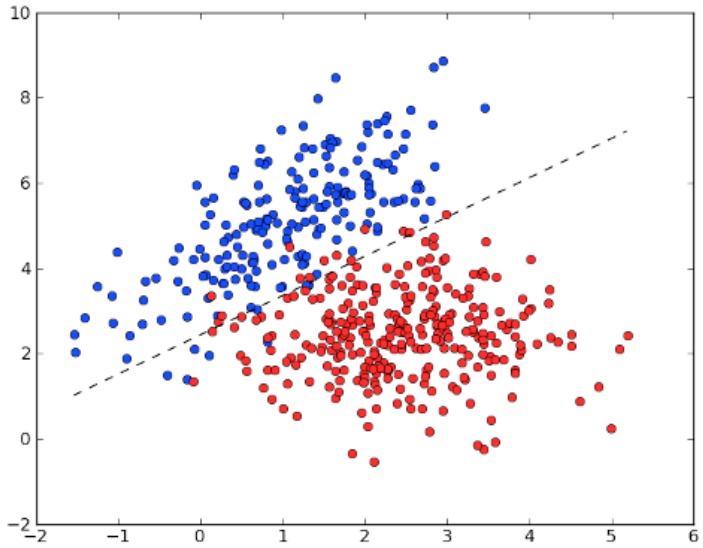
```
with tf.device("/job:ps/task:0"):
  weights_1 = tf.Variable(...)
  biases_1 = tf.Variable(...)

with tf.device("/job:ps/task:1"):
  weights_2 = tf.Variable(...)
  biases_2 = tf.Variable(...)

with tf.device("/job:worker/task:7"):
  input, labels = ...
  layer_1 = tf.nn.relu(tf.matmul(input, weights_1) + biases_1)
  train_op = ...

with tf.Session("grpc://worker7.example.com:2222") as sess:
  for _ in range(10000):
    sess.run(train_op)
```

Logistic Regression

- Logistic Regression (2-클래스)
  - **입력**: $x_i \in \mathbb{R}^d$
  - **출력**: $y_i \in \mathbb{R}$, $y_i = \sigma(Wx_i + b)$, 여기서 $\sigma(z) = 1/(1 + e^{-z})$ 는 Sigmoid 함수.
  - **목표**: $t_i \in \{0, 1\}$
  - **Loss**: Cross-Entropy $-\sum_i [t_i \log y_i + (1 - t_i) \log(1 - y_i)]$
- Logistic Regression (n-클래스)
  - **입력**: $x_i \in \mathbb{R}^d$
  - **출력**: $y_i \in \mathbb{R}^n$, $y_i = \sigma(Wx_i + b)$, 여기서 $\sigma : \mathbb{R}^n \to \mathbb{R}^n$, $\sigma(\mathbf{z}) = e^{-\mathbf{z}} / \sum e^{-\mathbf{z}}$ 는 Softmax 함수.
  - **목표**: $t_i \in \{1, 2, \cdots n\}$
  - **Loss**: Cross-Entropy $-\sum_i \sum_k \mathbb{I}_k(t_i) \log y_{ik}$

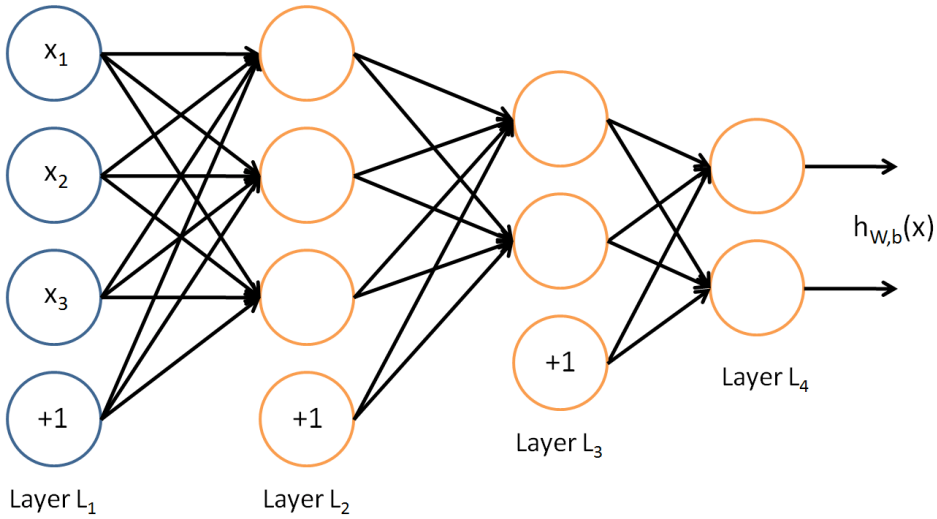
Logistic Regression (예시)





Deep Neural Network (DNN)

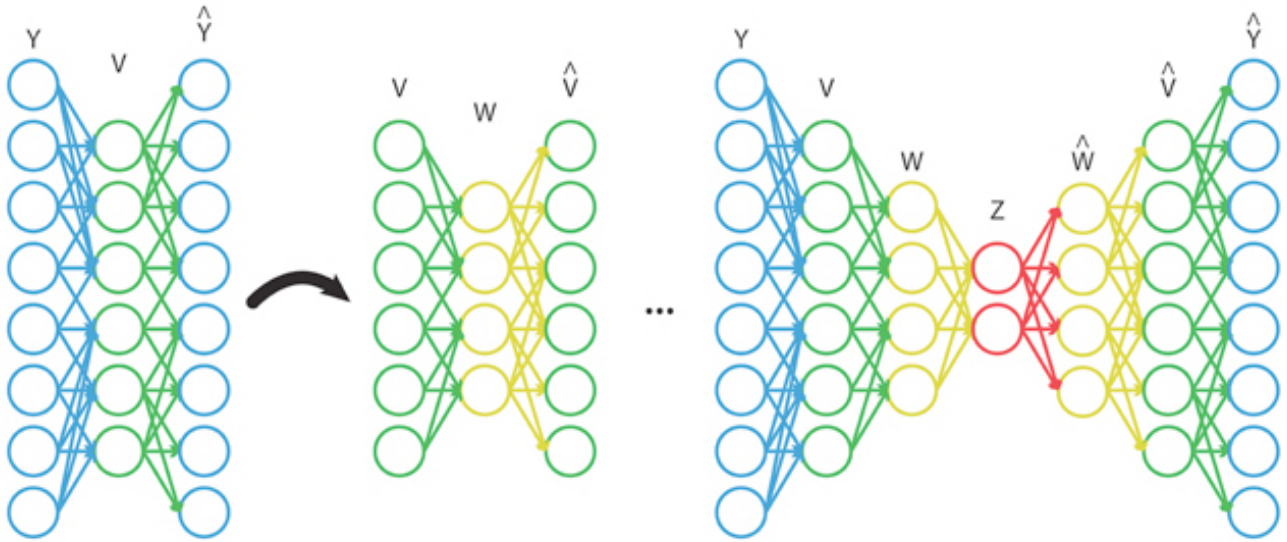
- Deep Neural Network (감독 학습)
  - **입력**: $x_i \in \mathbb{R}^d$
  - **출력**: $y_i = f(W^{(L)} \cdots f(W^{(2)} f(W^{(1)} x_i + b^{(1)}) + b^{(2)}) + \cdots + b^{(L)}) \in \mathbb{R}^n$
  - **목표**: $t_i \in \mathbb{R}^n$ 또는 $t_i \in \{1, 2, \cdots n\}$
  - **Loss**: MSE $(\sum_i \|y_i - t_i\|^2)$ 또는 Cross-Entropy $(-\sum_i \sum_k \mathbb{I}_k(t_i) \log y_{ik})$



- An example code can be found *here.*

Stacked Auto-Encoder

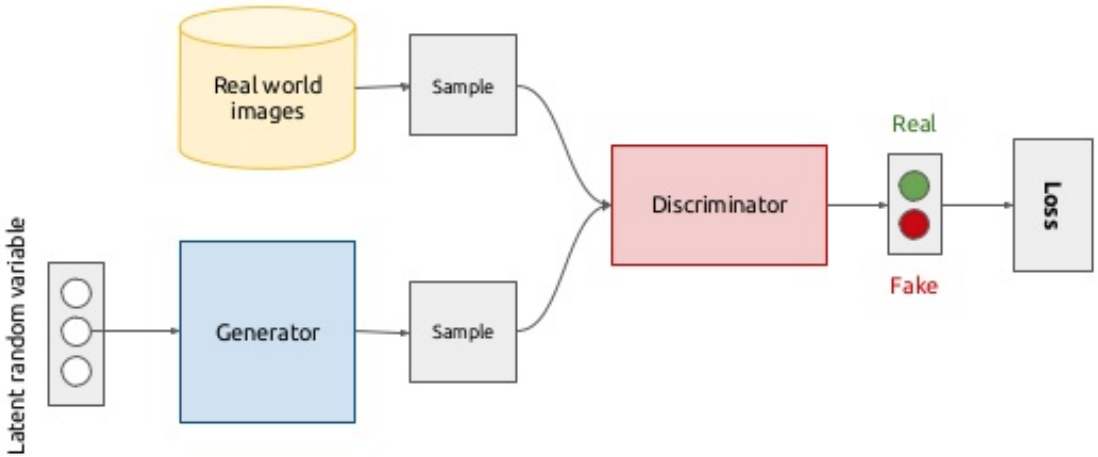
- Stacked Auto-Encoder (Unsupervised training using Back-prop)
  - **Input** and **Output**: Same as DNN
  - **Reconstruction**: $r_i = g(W^{(1)T} \cdots g(W^{(L)T} y_i + a^{(L)}) + \cdots + a^{(1)}) \in \mathbb{R}^d$
  - **Loss**: MSE $(\sum_i \|r_i - x_i\|^2)$ or Cross-Entropy $(-\sum_i \sum_k \mathbb{I}_k(x_i) \log r_i)$



- An example code can be found *here* (in Theano).

Generative Adversarial Network

- Generative Adversarial Network (Unsupervised adversarial training)
  - **Real Input**: $x_i \in \mathbb{R}^d$
  - **Latent Variable**: $z_i \in \mathbb{R}^k$
  - **Fake Input** (GEN): $\tilde{x}_i = g(W^{(1)} \cdots g(W^{(L)} z_i + a^{(L)}) + \cdots + a^{(1)}) \in \mathbb{R}^d$
  - **Classifier Output** (DIS):
    $y_i = f(U^{(L)} \cdots f(U^{(1)} x_i + b^{(1)}) + \cdots + b^{(L)}) \in \mathbb{R}$
  - **Training**: $\min_{Gen} \max_{Dis} \sum_{i \in Real} \log y_i + \sum_{i \in Fake} \log(1 - y_i)$



- An example code (convolution version) can be found *here.*

기계학습 모델 구현 실습

- Logistic Regression 모델 실습
- Deep Neural Network 모델 실습