기초3 + Trend

2018. 7. 16

Lecture Notes: http://eclass.mju.ac.kr

오늘의 목차

- RNN 개념 (Sequence 2 Sequence)
  - SKT 정상근 박사 자료 참고

- Optimizer
  - (참고) Sebastian Ruder (박사과정 학생)의
https://www.slideshare.net/SebastianRuder/optimization-for-deep-learning

- tf.gradients

Optimizer

- 변수들을 object(또는 loss) function에 맞추어 최적화
- 텐서플로우의 최적화 함수들

  - `tf.train.Optimizer`
  - `tf.train.GradientDescentOptimizer`
  - `tf.train.AdadeltaOptimizer`
  - `tf.train.AdagradOptimizer`
  - `tf.train.AdagradDAOptimizer`
  - `tf.train.MomentumOptimizer`
  - `tf.train.AdamOptimizer`
  - `tf.train.FtrlOptimizer`
  - `tf.train.ProximalGradientDescentOptimizer`
  - `tf.train.ProximalAdagradOptimizer`
  - `tf.train.RMSPropOptimizer`

Optimizer

Gradient Descent → Momentum → AdaGrad → AdaDelta

RMSProp

Adam

텐서플로우에서 Optimizer 사용

- `tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)`

- `tf.train.MomentumOptimizer(learning_rate, momentum=0.9, use_nesterov=True).minimize(loss)`

- `tf.train.RMSPropOptimizer(learning_rate, decay=0.99 , momentum=0.9, epsilon=1e-10).minimize(loss)`

- `tf.train.AdamOptimizer(learning_rate, beta1=0.9, beta2=0.99, epsilon=1e-8).minimize(loss)`

Learning rate decay

- `tf.train.exponential_decay`
- `tf.train.inverse_time_decay`
- `tf.train.natural_exp_decay`
- `tf.train.piecewise_constant`
- `tf.train.polynomial_decay`
- `tf.train.cosine_decay`
- `tf.train.linear_cosine_decay`
- `tf.train.noisy_linear_cosine_decay`

tf.train.cosine_decay_restarts

```
tf.train.cosine_decay_restarts(
    learning_rate,
    global_step,
    first_decay_steps,
    t_mul=2.0,
    m_mul=1.0,
    alpha=0.0,
    name=None
)
```

tf.gradients

- 기존의 optimization함수들은 loss에 대해서 각 변수의 gradient를 구해서 특정 비율(학습률)에 따라, 해당 변수의 값을 update하였다. 따라서, 가장 기본이 되는 함수는 tf.gradients이고, 이것이 어떻게 동작하는지 궁금할 것이다. (아닌가? ㅋ)

- tf.gradients(ys=, xs=, stop_gradients=)

tf.gradients

```
g1 = tf.Graph()
with g1.as_default():
    W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
    b = tf.Variable(tf.zeros([1]))
    y = W * x_data + b
    loss = tf.reduce_mean(tf.square(y - y_data))
    dW, db = tf.gradients(loss, [W, b])
    update_W = tf.assign(W, W - 0.5 * dW)
    update_b = tf.assign(b, b - 0.5 * db)
```

||

```
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```

tf.gradients

```python
import tensorflow as tf
import numpy as np
x = np.random.normal(size=(1,10), loc=1.0, scale=0.1)
y = np.random.normal(size=(1,3), loc=0.0, scale=0.5)

X = tf.placeholder(dtype=tf.float32, shape=(1,10))
Y = tf.placeholder(dtype=tf.float32, shape=(1,3))

W = tf.Variable(tf.random_normal(shape=(10,3), mean=0.0, stddev=1.0))
b = tf.Variable(tf.random_normal(shape=(1,3), mean=0.0, stddev=1.0))
Y2 = tf.matmul(X, W) + b
loss = tf.pow(Y2 - Y,2)

[dW, db] = tf.gradients(loss, [W, b]) # full derivative
[pW, pb] = tf.gradients(loss, [W, b], stop_gradients=[W, b]) # partial derivative

[dX] = tf.gradients(loss, [X])    # loss against input X

# dX = tf.gradients(loss, [X])    # AttributeError: 'list' object has no attribute 'shape'

# *** gradients는 loss가 vector일 경우, 각각을 x에 대해서 미분하고, 그것의 합을 return한다.
[dX1] = tf.gradients(loss[0,0], [X])
[dX2] = tf.gradients(loss[0,1], [X])
[dX3] = tf.gradients(loss[0,2], [X])
```

```
loss :  Tensor("Pow:0", shape=(1, 3), dtype=float32)
dW :  (10, 3)
db :  (1, 3)
pW :  (10, 3)
pb :  (1, 3)
dX :  (1, 10)
dX1 :  (1, 10)
```

tf.gradients

```python
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    [dw, pw, dx, dx1, dx2, dx3] = sess.run([dW, pW, dX, dX1, dX2, dX3], feed_dict={X: x, Y: y})
    print(dw)
    print(pw)
    print(dx)
    print(dx1 + dx2 + dx3)
```

```
dw:  [[ 3.57577968 -6.74424076  4.95763016]
 [ 4.12490654 -7.77994299  5.71896553]
 [ 4.21664143 -7.95296383  5.84615135]
 [ 3.41820407 -6.44703913  4.73916006]
 [ 4.25212193 -8.0198822   5.89534283]
 [ 3.52583718 -6.65004444  4.88838768]
 [ 4.135221   -7.79939747  5.73326635]
 [ 3.28278828 -6.19163275  4.55141306]
 [ 3.9165771  -7.38701534  5.4301281 ]
 [ 4.15295267 -7.83284092  5.75785065]]
pw:  [[ 3.57577968 -6.74424076  4.95763016]
 [ 4.12490654 -7.77994299  5.71896553]
 [ 4.21664143 -7.95296383  5.84615135]
 [ 3.41820407 -6.44703913  4.73916006]
 [ 4.25212193 -8.0198822   5.89534283]
 [ 3.52583718 -6.65004444  4.88838768]
 [ 4.135221   -7.79939747  5.73326635]
 [ 3.28278828 -6.19163275  4.55141306]
 [ 3.9165771  -7.38701534  5.4301281 ]
 [ 4.15295267 -7.83284092  5.75785065]]
dx:  [[  1.20921648   3.64001036   3.55404782  -6.57240772   2.32383347
    3.11868715  -0.98194981   8.24107552   7.81137276  16.9954071 ]]
dx[]:  [[  1.20921648   3.64001036   3.55404782  -6.57240772   2.32383347
    3.11868715  -0.98194981   8.24107552   7.81137276  16.9954071 ]]
```

Contractive Autoencoder (Jacobian)

2. How to extract robust features

To encourage robustness of the representation $f(x)$ obtained for a training input x we propose to penalize its *sensitivity* to that input, measured as the Frobenius norm of the Jacobian $J_f(x)$ of the non-linear mapping. Formally, if input $x \in \mathbb{R}^{d_x}$ is mapped by encoding function f to hidden representation $h \in \mathbb{R}^{d_h}$, this sensitivity penalization term is the sum of squares of all partial derivatives of the extracted features with respect to input dimensions:

$$\|J_f(x)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2 . \qquad (1)$$

$$\mathcal{J}_{\mathrm{CAE}}(\theta) = \sum_{x \in D_n} \left(L(x, g(f(x))) + \lambda \|J_f(x)\|_F^2 \right)$$

Software 2.0

- Andrej Karpathy는 Director of AI@테슬라

- 현재까지 Software 개발은 Software 1.0로 정의

- 이미지 인식 등 Neural Network를 통한 개발 방법은 기존의 개발 방법과는 차이가 있으며, 이를 Software 2.0으로 정의하고, 무엇이 차이가 있는지, 어떤 이점이 있는지, 향후에 무엇이 필요한지를 설명함

- https://www.youtube.com/watch?v=zywIvINSlaI

참고자료

- RNN 개념 (Sequence 2 Sequence)
  - SKT 정상근 박사 자료 참고

- Optimizer

https://www.slideshare.net/SebastianRuder/optimization-for-deep-learning

http://ruder.io/deep-learning-optimization-2017/

https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/

(한글) http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

(한글) http://aikorea.org/cs231n/neural-networks-3/#sgd