

메디치소프트 기술연구소

딥러닝 8일

신경망 Optimizer

• 개요

- 신경망에서 사용하는 최적화 방식에 대해, 간략히 설명하고 이를 구현한 tensorflow 첨부 작성
- 추후에 Optimizer 선택에 기반이 되는 자료
- 근래에는 주로 Adam 옵티마이저를 주로 사용

• Tensorflow 목록

- tf.train 모듈에 포함된 옵티마이저들이다. 이 중에서 볼드체로 표시된 6가지 옵티마이저 API와 알고리즘에 대해 다룬다. 아래 목록 이외에도 tf.contrib.opt 를 살펴보면 더 많은 옵티마이저 API 가 있다.

➤**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**

신경망 Optimizer - Gradient Descent

. 개요

- Neural Network의 가장 기본적인 학습 방법으로, Neural Network에서 내놓은 결과값과 실제 결과값의 차이를 정의하는 Loss 함수를 최소화 하기 위하여 기울기를 이용하는 방법이다.
- Descent에서는 네트워크 파라미터들에 대해 기울기의 반대 방향으로 일정 크기 만큼 이동해내는 것을 반복하여 loss 함수의 값을 최소화하는 네트워크 파라미터들의 값을 찾는다.
- loss 함수를 계산할 때 전체 train 데이터 셋을 사용
- 전체 train 데이터 셋에 대한 계산을 한 뒤에 네트워크 파라미터를 업데이트 하기 때문에 계산량이 너무 커 최적화된 네트워크 파라미터를 찾아가는 속도가 느려 현재는 거의 쓰이지 않고 있다.

$$\theta(t) = \theta(t-1) - \text{learning_rate} * \text{gradient}$$

```
tf.train.GradientDescentOptimizer
```

```
__init__(  
    learning_rate,  
    use_locking=False,  
    name='GradientDescent'  
)
```

learning_rate: A Tensor or a floating point value. The learning rate to use.

신경망 Optimizer - Momentum

. 개요

- 기존의 Gradient Descent를 통해 이동하는 과정에 일종의 관성의 원리를 추가
- 구현과 원리가 매우 간단하면서 Gradient Descent 보다 좋은 성능
- 관성의 원리를 추가한다는 것은 이전 계산에 의해 나온 기울기를 일정한 크기만큼 반영하여 새로운 기울기와 합하여 사용
- 기존의 기울기를 일정 부분 유지하면서 새로운 기울기를 적용하여, 관성과 같은 효과를 주는 방법

$$v(t) = \text{momentum} * v(t-1) + \text{learning_rate} * \text{gradient}$$
$$\text{theta} = \text{theta} - v(t)$$

```
tf.train.MomentumOptimizer  
__init__(  
    learning_rate,  
    momentum,  
    use_locking=False,  
    name='Momentum',  
    use_nesterov=False  
)
```

신경망 Optimizer - Adagrad

. 개요

- 시간에 따른 그래디언트 제곱값을 추적해 학습률을 조정하는 알고리즘
- 잘 업데이트되지 않는 파라미터의 학습률을 높이기 때문에 스파스한 데이터에서 특히 유용
- 네트워크 파라미터를 업데이트 할 때 iteration이 반복 될때, 각 변수의 step size를 다르게 설정하여 이동하는 방법
- 장점: 사용하게 되면 학습을 진행하는 동안, learning rate decay에 대한 고려를 하지 않아도 된다
- 단점: 학습을 진행할 수록 learning rate 값이 너무 줄어드는 문제가 존재

$$G(t) = G(t-1) + \text{gradient}^2$$

$$\theta(t+1) = \theta(t) - \text{learning_rate} * \text{gradient} / \sqrt{G(t)} \quad (\text{입실론})$$

```
tf.train.AdagradOptimizer  
__init__(  
    learning_rate,  
    initial_accumulator_value=0.1,  
    use_locking=False,  
    name='Adagrad'  
)
```

신경망 Optimizer - Adadelata

. 개요

- Adadelata는 Adagrad를 개선하기 위해 제안된 방법으로, 하이퍼파라미터에 덜 민감하고 학습률을 너무 빨리 떨어뜨리지 않도록 막는다.
- Adagrad의 단점을 극복하기 위하여 개발된 최적화 알고리즘이다. 이를 위하여, G 값을 구할 때 합을 구하는 대신 지수 평균을 이용하여 구하는 방법이다. Adadelata는 Adagrad와 다르게 learning rate 값을 단순히 η 으로 사용하는 대신 learning rate의 변화 값에 제곱을 가지고 지수 평균 값을 이용하여 loss함수를 최적화시키는 네트워크 파라미터의 값을 구하게 된다.

```
tf.train.AdadeltaOptimizer  
__init__(  
    learning_rate=0.001,  
    rho=0.95,  
    epsilon=1e-08,  
    use_locking=False,  
    name='Adadelta'  
)
```

신경망 Optimizer - Adam(1/2)

. 개요

- 파라미터 업데이트를 그래디언트의 평균과 분산으로부터 직접 추정하고 편향(bias) 조정항을 추가
- Adam은 RMSProp와 Momentum의 장점을 합친 최적화 알고리즘이다. Adam은 Momentum과 유사하게 지금까지 학습이 진행되며 계산된 기울기의 지수 평균을 저장하며, RMSProp와 유사하게 기울기의 제곱 값의 지수 평균을 저장한다. 기울기의 지수평균과 기울기의 제곱 값의 지수 평균은 다음과 같이 나타낸다

$$m(t) = \text{beta1} * m(t-1) + (1-\text{beta1}) * \text{gradient}$$
$$v(t) = \text{beta2} * v(t-1) + (1-\text{beta2}) * \text{gradient}^2$$

위 수식과 같이 m 과 v 가 계산되지만, m 과 v 는 학습 초기에 0으로 초기화 되어 있기 때문에 학습의 초반부에 m_t, v_t 가 0에 가깝게 bias가 설정되어 있기때문에 이를 unbiased 하게 만들어주는 작업을 거친다. 다음과 같은 보정을 통해 unbiased된 값을 통하여 기울기가 들어갈 자리에 $m'(t), G(t)$ 가 들어갈 자리에 $v'(t)$ 를 넣어 계산을 진행한다.

$$m'(t) = m(t) / (1-\text{beta1}(t))$$
$$v'(t) = v(t) / (1-\text{beta2}(t))$$

$$\text{theta}(t) = \text{theta}(t-1) - \text{learning_rate} * m'(t) / \text{sqrt}(v'(t) + \text{epsilon})$$

신경망 Optimizer – Adam(2/2)

• 개요

RMSProp와 Momentum의 방법을 동시에 이용하기 때문에 현재 Neural Network 학습에서 많이 사용되는 Optimizer 이며, Adam은 Adagrad와 마찬가지로 learning rate decay에 대한 고려가 필요없다. 그 이유는 learning rate를 1/2씩 exponential 하게 줄여주는 기능을 수행하고 있기 때문이다. 실제 학습 결과에서도 자동으로 learning rate decay에 대한 고려가 들어간 학습의 최적화 알고리즘이 더 나은결과를 보여주기 때문이다.

```
tf.train.AdamOptimizer
__init__(
    learning_rate=0.001,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam'
)
```


[AI 기술참조] 앙상블 기법(다중 모델 조합 알고리즘)

알고리즘	배깅 (Bootstrap Aggregating, Bagging)
정의	훈련용 데이터 집합으로부터 크기가 같은 표본을 여러 번 단순확률 반복 추출하여 각각에 대한 분류기를 생성하고, 생성된 분류기들의 결과를 종합하여 의사결정을 내리는 방법.
모델링 개념도	
장·단점	전체 데이터를 여러 번 복원추출하여 다수의 샘플 데이터를 대상으로 분류기를 실행하여 분류의 정확도가 향상 표본 데이터가 작은 경우 그 표본 데이터가 전체 데이터를 잘 반영하지 못함 데이터에 노이즈가 많은 경우 특이점이 추정을 크게 왜곡시킬 가능성이 존재
활용사례	각종 분류분석에서 예측력을 높이기 위해 활용

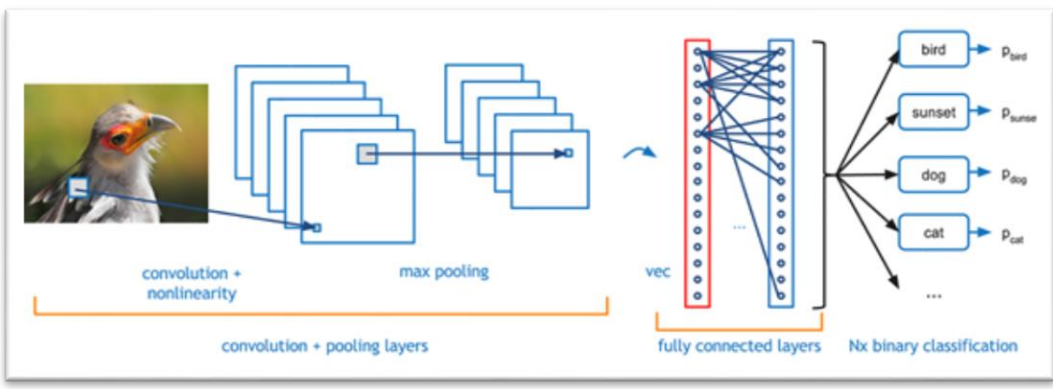
[AI 기술참조] 앙상블 기법(다중 모델 조합 알고리즘)

알고리즘	부스팅 (Boosting)
정의	초기 샘플 데이터를 조작하여 다수의 분류기를 생성하는 기법 중 하나 배경과는 달리 순차적(Sequential) 방법 이전 분류기의 학습 결과를 토대로 다음 분류기의 학습 데이터의 샘플 가중치를 조정하여 학습을 진행
모델링 개념도	
장·단점	노이즈가 없는 데이터에 대해서는 부스팅이 배경보다 우수 과적합 문제로부터 자유롭지 못함
활용사례	각종 분류분석에서 예측력을 높이기 위해 활용

[AI 기술참조] 앙상블 기법(다중 모델 조합 알고리즘)

알고리즘	랜덤 포레스트 (Random Forest, RF)
정의	DT의 단점을 개선하기 위한 알고리즘 다수의 DT를 결합하여 하나의 모델을 생성
모델링 개념도	
장·단점	다양성을 극대화하여 예측력이 상당히 우수 안정성이 상당히 높음 데이터 세트의 규모가 작으면 성능이 높지 않음
활용사례	엑스박스360의 키넥트에서의 신체 트래킹 컴퓨터 단층 촬영 영상 내 해부학 구조 검출 및 위치파악 다채널 MRI 내 고악성도 신경교종 검출

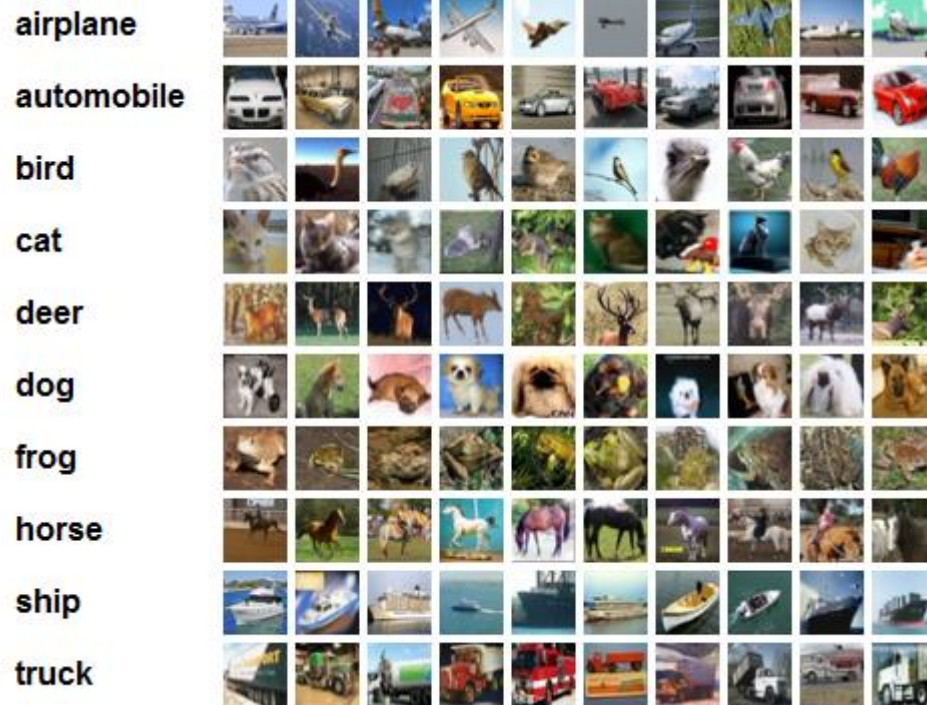
[AI 기술참조] CNN (Convolution Neural Network)

알고리즘	CNN
정의	인간의 시신경 구조를 모방해 만들어진 인공신경망 알고리즘. 인공신경망에 기존의 필터기술을 병합하여 2차원 영상을 잘 습득할 수 있도록 최적화시킨 알고리즘. Convolutional Layer에서 합성곱 연산을 통해서 특징을 추출하고 Subsampling을 통해 차원을 축소하여 이미지를 단순화 시킨 후 Fully Connected Layer에서 이전 레이어의 처리 결과를 연결하여 이미지를 분류.
모델링 개념도	 <p>The diagram illustrates the CNN architecture. It starts with an input image of a bird. This is followed by a series of layers labeled 'convolution + nonlinearity' and 'max pooling', which are grouped under 'convolution + pooling layers'. The output is a vector 'vec'. This vector is then processed by 'fully connected layers' to produce 'Nx binary classification' results. The classification results are shown as probabilities for different classes: bird (P_{bird}), sunset (P_{sunset}), dog (P_{dog}), cat (P_{cat}), and others.</p>
장·단점	2차원 데이터의 입력이 용이 학습 데이터 훈련이 용이 적은 매개변수 사용으로 활용이 쉬움 특징 추출을 위해 다수의 은닉층이 사용되고 학습과 테스트 시간이 많이 걸린다.
활용사례	흑백 사진 또는 흑백 영상의 컬러 복원 저해상도 사진(또는 모자이크) 픽셀 복원 필기 인식 차량 모델 또는 번호판 인식

CIFAR-10

• 개요

- MNIST 데이터셋 다음에 보통 CIFAR-10 데이터셋 검증 사용
- airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
- 각각의 레이블마다 32×32 크기 이미지인 50,000개의 training 데이터셋, 10,000개의 test 데이터셋이 존재하고, 결과적으로 총 60,000개의 32×32 크기의 이미지로 데이터셋이 구성



CIFAR-10

```
import tensorflow as tf
import numpy as np
from tensorflow.python.keras.datasets.cifar10 import load_data
# CIFAR-10 데이터를 다운로드 받기 위한 helper 모듈인 load_data 모듈을 임포트합니다.

# 다음 배치를 읽어오기 위한 next_batch 유틸리티 함수를 정의합니다.
def next_batch(num, data, labels):
    """
    ~~~~~
    Return a total of `num` random samples and labels.
    """
    ~~~~~
    idx = np.arange(0, len(data))
    np.random.shuffle(idx)
    idx = idx[:num]
    data_shuffle = [data[i] for i in idx]
    labels_shuffle = [labels[i] for i in idx]

    return np.asarray(data_shuffle), np.asarray(labels_shuffle)
```

CIFAR-10

"""CIFAR-10 이미지를 분류하기 위한 Convolutional Neural Networks 그래프를 생성합니다.

인자들(Args): x: (N_examples, 32, 32, 3) 차원을 가진 input tensor, CIFAR-10 데이터는 32x32 크기의 컬러이미지이다.

리턴값들>Returns): tuple (y, keep_prob). y는 (N_examples, 10)형태의 숫자(0-9) tensor이다.

keep_prob는 dropout을 위한 scalar placeholder이다

CNN 모델을 정의합니다.

def build_CNN_classifier(x):

```
    |
    |
    x_image = x
    # 첫번째 convolutional layer - 하나의 grayscale 이미지를 64개의 특징들(feature)으로 맵핑(mapping)한다.
    W_conv1 = tf.Variable(tf.truncated_normal(shape=[5, 5, 3, 64], stddev=5e-2))
    b_conv1 = tf.Variable(tf.constant(0.1, shape=[64]))
    h_conv1 = tf.nn.relu(tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1], padding='SAME') + b_conv1)
    # 첫번째 Pooling layer
    h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')
    # 두번째 convolutional layer -- 32개의 특징들(feature)을 64개의 특징들(feature)로 맵핑(mapping)한다.
    W_conv2 = tf.Variable(tf.truncated_normal(shape=[5, 5, 64, 64], stddev=5e-2))
    b_conv2 = tf.Variable(tf.constant(0.1, shape=[64]))
    h_conv2 = tf.nn.relu(tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1], padding='SAME') + b_conv2)
    # 두번째 pooling layer.
    h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')
    # 세번째 convolutional layer
    W_conv3 = tf.Variable(tf.truncated_normal(shape=[3, 3, 64, 128], stddev=5e-2))
    b_conv3 = tf.Variable(tf.constant(0.1, shape=[128]))
    h_conv3 = tf.nn.relu(tf.nn.conv2d(h_pool2, W_conv3, strides=[1, 1, 1, 1], padding='SAME') + b_conv3)
    # 네번째 convolutional layer
    W_conv4 = tf.Variable(tf.truncated_normal(shape=[3, 3, 128, 128], stddev=5e-2))
    b_conv4 = tf.Variable(tf.constant(0.1, shape=[128]))
    h_conv4 = tf.nn.relu(tf.nn.conv2d(h_conv3, W_conv4, strides=[1, 1, 1, 1], padding='SAME') + b_conv4)
    # 다섯번째 convolutional layer
    W_conv5 = tf.Variable(tf.truncated_normal(shape=[3, 3, 128, 128], stddev=5e-2))
    b_conv5 = tf.Variable(tf.constant(0.1, shape=[128]))
    h_conv5 = tf.nn.relu(tf.nn.conv2d(h_conv4, W_conv5, strides=[1, 1, 1, 1], padding='SAME') + b_conv5)

    # Fully Connected Layer 1 -- 2번의 downsampling 이후에, 우리의 32x32 이미지는 8x8x128 특징맵(feature map)이 된다.
    # 이를 384개의 특징들로 맵핑(mapping)한다.
    W_fc1 = tf.Variable(tf.truncated_normal(shape=[8 * 8 * 128, 384], stddev=5e-2))
    b_fc1 = tf.Variable(tf.constant(0.1, shape=[384]))
```

```
# Fully Connected Layer 1 -- 2번의 downsampling 이후에, 우리의 32x32 이미지는 8x8x128 특징맵(feature map)이 된다.  
# 이를 384개의 특징들로 맵핑(mapping)한다.  
W_fc1 = tf.Variable(tf.truncated_normal(shape=[8 * 8 * 128, 384], stddev=5e-2))  
b_fc1 = tf.Variable(tf.constant(0.1, shape=[384]))  
  
h_conv5_flat = tf.reshape(h_conv5, [-1, 8*8*128])  
h_fc1 = tf.nn.relu(tf.matmul(h_conv5_flat, W_fc1) + b_fc1)  
  
# Dropout - 모델의 복잡도를 컨트롤한다. 특징들의 co-adaptation을 방지한다.  
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)  
  
# 384개의 특징들(feature)을 10개의 클래스-airplane, automobile, bird...-로 맵핑(mapping)한다.  
W_fc2 = tf.Variable(tf.truncated_normal(shape=[384, 10], stddev=5e-2))  
b_fc2 = tf.Variable(tf.constant(0.1, shape=[10]))  
logits = tf.matmul(h_fc1_drop, W_fc2) + b_fc2  
y_pred = tf.nn.softmax(logits)  
  
return y_pred, logits
```


CIFAR-10

```
# 인풋 아웃풋 데이터, 드롭아웃 확률을 입력받기위한 플레이스홀더를 정의합니다.
x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
y = tf.placeholder(tf.float32, shape=[None, 10])
keep_prob = tf.placeholder(tf.float32)

# CIFAR-10 데이터를 다운로드하고 데이터를 불러옵니다.
(x_train, y_train), (x_test, y_test) = load_data()

# scalar 형태의 레이블(0~9)을 One-hot Encoding 형태로 변환합니다.
y_train_one_hot = tf.squeeze(tf.one_hot(y_train, 10), axis=1)
y_test_one_hot = tf.squeeze(tf.one_hot(y_test, 10), axis=1)

# Convolutional Neural Networks(CNN) 그래프를 생성합니다.
y_pred, logits = build_CNN_classifier(x)

# Cross Entropy를 비용함수(loss function)으로 정의하고, RMSPropOptimizer를 이용해서 비용 함수를 최소화합니다.
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=logits))
train_step = tf.train.RMSPropOptimizer(1e-3).minimize(loss)

# 정확도를 계산하는 연산을 추가합니다.
correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

CIFAR-10

세션을 열어 실제 학습을 진행합니다.

```
with tf.Session() as sess:
```

모든 변수들을 초기화한다.

```
sess.run(tf.global_variables_initializer())
```

10000 Step만큼 최적화를 수행합니다.

```
for i in range(10000):
```

```
    batch = next_batch(128, x_train, y_train_one_hot.eval())
```

100 Step마다 training 데이터셋에 대한 정확도와 loss를 출력합니다.

```
    if i % 100 == 0:
```

```
        train_accuracy = accuracy.eval(feed_dict={x: batch[0], y: batch[1], keep_prob: 1.0})
```

```
        loss_print = loss.eval(feed_dict={x: batch[0], y: batch[1], keep_prob: 1.0})
```

```
        print("반복(Epoch): %d, 트레이닝 데이터 정확도: %f, 손실 함수(loss): %f" % (i, train_accuracy, loss_print))
```

20% 확률의 Dropout을 이용해서 학습을 진행한다.

```
sess.run(train_step, feed_dict={x: batch[0], y: batch[1], keep_prob: 0.8})
```

학습이 끝나면 테스트 데이터에 대한 정확도를 출력합니다.

```
test_batch = next_batch(10000, x_test, y_test_one_hot.eval())
```

```
print("테스트 데이터 정확도: %f" % accuracy.eval(feed_dict={x: test_batch[0], y: test_batch[1], keep_prob: 1.0}))
```

2018-08-09 01:18:50.095107: I T:\src\github\tensorflow\tensorflow\core\plat

반복(Epoch): 0, 트레이닝 데이터 정확도: 0.125000, 손실 함수(loss): 126.775803