인공지능개론 4장 질문노트

(꼭 PDF로 변환후 제출하기)

ICT 융합공학부 20학번 박준형 / ICT 융합공학부 20학번 김정호

4장

준형: 정호야, 기계학습에서 두 가지 주요 접근법이 뭔지 알고 있니?

정호: 당연하지! 기계학습의 주요 접근법은 지도 학습과 비지도 학습이야. 지도 학습은 입력 데이터와 그에 상응하는 레이블을 사용하여 모델을 훈련시키는 방법이고, 비지도 학습은 레이블 없이 입력 데이터만을 사용하여 모델을 학습시키는 방법이야.

정호: 그렇다면 준형아 너는 왜 훈련 데이터와 시험 데이터를 나누는 건지 알고있어?

준형: 훈련 데이터와 시험 데이터를 나누는 이유는 모델의 일반화 성능을 평가하기 위해서야. 훈련 데이터로 모델을 학습시키고, 시험 데이터로 학습된 모델의 성능을 평가하여 새로운 데이터에 대한 예측 능력을 확인할 수 있어.

정호: 경사법(경사 하강법)에서 학습률에 대해 나는 궁금해. 사용하는 학습률 같은 매개변수는 하이퍼파라미터야. 우리가 지금 매개변수의 최적값을 자동으로 구하기 위해 이런 경사법, 미분 등 여러 방법을 사용해 보았어. 학습률은 사람이 직접 조정해가면서 실험을 한다고 배웠는데 이런 학습률 조차 자동으로 획득할 수는 없는걸까?

준형: 그 점에 대해 내가 한번 찾아보았어. 일부 고급 기술 및 알고리즘은 학습률을 조정하기 위해 자동화 기법을 사용해. 예시로 학습률 스케줄링, 최근에는 Adam 옵티마이저 알고리즘도 있다고해. 간단하게 설명해보면 우선 학습률 스케줄링은 학습이 진행됨에 따라 동적으로 조정하는 방법인데, 초기에 큰 학습률로 빠르게 수렴 후 일정한 간격(매 에폭 또는 반복횟수)마다 학습률을 감소시키는 것이야. 그리고 Adam 옵티마이저 알고리즘은 경사의 제곱에 대한 지수 가중 이동 평균을 사용하여 각 매개변수의 학습률을 자동으로 조절하여 수동에서의 불편한 점을 개선할 수 있어.

정호: 역시 학습률 또한 자동으로 획득할 수 있는 부분이었구나!! 준형이 너는 공부하면서 이해가 안되거나 궁금한 점 뭐 없었어??

준형: 나는 신경망 학습에 대해 공부하면서 에폭이라는 단어가 자주 나오는 것 같은데 이 단어의 개념을 정확히 이해하고 싶어.

정호: 나도 그게 궁금했었는데 이번에 한번 제대로 알아보자! 우선, 에폭은 머신러닝에서 학습 과정 중 데이터셋이 모델에 대해 한번씩 순전파와 역전파를 거치는 단위야. 쉽게 말하면 그냥 한 번의 에폭은 모든 훈련 데이터가 한 번씩 모델을 통과하여 학습에 사용된다는 것을 의미해. 이러한 에폭도 위에서 말한 하이퍼파라미터로 정의가 되며 적절한 에폭 수를 선택하는 것이 모델의 성능과 학습 시간에 큰 영향을 미치게돼. 따라서 적절한 적절한 에폭 수를 설정해 주는 것이 중요한 포인트인것 같아.

준형: 고마워 정호야! 덕분에 에폭에 대해 정확히 알수 있었어.

정호: 나도 알려주면서 개념을 한번더 정리할 수 있었어. 다음에 또 대화나누자! 즐거웠어~

jupyter week6_0410 Last Checkpoint: 13 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[1]:
```
import sys, os
sys.path.append(os.pardir)
from dataset.mnist import load_mnist

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=False, flatten=True, one_hot_label=False)

# 각 데이터의 형상 출력
print('x_train = ' ,x_train.shape, 't_train = ', t_train.shape)
print('x_test = ' ,x_test.shape, 't_test = ', t_test.shape)
```

```
x_train =  (60000, 784) t_train =  (60000,)
x_test =  (10000, 784) t_test =  (10000,)
```

[2]:
```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from PIL import Image


def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)

img = x_train[0]
label = t_train[0]
print(label)  # 5

print(img.shape)  # (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
print(img.shape)  # (28, 28)
```

jupyter week6_0410 Last Checkpoint: 13 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
print(label)  # 5

print(img.shape)  # (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
print(img.shape)  # (28, 28)

img_show(img)
```

```
5
(784,)
(28, 28)
```

[3]:
```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
from common.functions import *
from common.gradient import numerical_gradient

class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    def predict(self, x):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']

        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)

        return y
```

jupyter week6_0410 Last Checkpoint: 13 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
    # x:입력 데이터, t:정답 레이블
    def loss(self, x, t):
        y = self.predict(x)

        return cross_entropy_error(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    # x:입력 데이터, t:정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

        return grads

    def gradient(self, x, t):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']
        grads = {}

        batch_num = x.shape[0]

        # forward
        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
```

jupyter week6_0410 Last Checkpoint: 13 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)

        # backward
        dy = (y - t) / batch_num
        grads['W2'] = np.dot(z1.T, dy)
        grads['b2'] = np.sum(dy, axis=0)

        dz1 = np.dot(dy, W2.T)
        da1 = sigmoid_grad(a1) * dz1
        grads['W1'] = np.dot(x.T, da1)
        grads['b1'] = np.sum(da1, axis=0)

        return grads
```

[ ]:

```
import numpy as np
from dataset.mnist import load_mnist

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0]
batch_size = 100  # 미니배치 크기
learning_rate = 0.1

train_loss_list = []

# 1에폭당 반복
iter_per_epoch = max(train_size / batch_size, 1)

for i in range(iters_num):
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
```

jupyter week6_0410 Last Checkpoint: 13 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
# 1에폭당 반복
iter_per_epoch = max(train_size / batch_size, 1)

for i in range(iters_num):
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    grad = network.numerical_gradient(x_batch, t_batch)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)  # 수정: loss 메서드 호출
    train_loss_list.append(loss)
```

[2]:

```
# coding: utf-8
import sys, os
sys.path.append(os.pardir)
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1
```

jupyter week6_0410 Last Checkpoint: 13 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    #grad = network.numerical_gradient(x_batch, t_batch)
    grad = network.gradient(x_batch, t_batch)

    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

jupyter week6_0410 Last Checkpoint: 13 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
plt.legend(loc='lower right')
plt.show()
```

```
train acc, test acc | 0.09736666666666667, 0.0982
train acc, test acc | 0.78235, 0.7883
train acc, test acc | 0.8792666666666666, 0.8828
train acc, test acc | 0.8982833333333333, 0.9008
train acc, test acc | 0.9080833333333334, 0.9106
train acc, test acc | 0.9153333333333333, 0.9184
train acc, test acc | 0.9199333333333334, 0.9218
train acc, test acc | 0.92365, 0.9253
train acc, test acc | 0.9278833333333333, 0.9295
train acc, test acc | 0.9308333333333333, 0.9312
train acc, test acc | 0.9336833333333333, 0.9348
train acc, test acc | 0.9366666666666666, 0.9367
train acc, test acc | 0.9389166666666666, 0.9382
train acc, test acc | 0.9419, 0.9416
train acc, test acc | 0.9440333333333333, 0.944
train acc, test acc | 0.94575, 0.9442
train acc, test acc | 0.94775, 0.9456
```