머신러닝 과제03

제출: 각 문제에 대한 source code 및 report (word format)을 zip으로 압축, KLAS에 제출
제출시 파일이름: 학번_이름.zip (예: 2014200154_홍길동.zip)

(이론)_____

1 NOR 게이트와 AND 게이트의 동작을 데이터로 간주하면 다음과 같다. 이들을 100% 옳게 분류하는 퍼셉트론을 각각 제시하시오.

$$\text{NOR 분류}\begin{cases}\mathrm{x}_1=(0,0)^{\mathrm{T}},\ y_1=\ \ 1\\ \mathrm{x}_2=(1,0)^{\mathrm{T}},\ y_2=-1\\ \mathrm{x}_3=(0,1)^{\mathrm{T}},\ y_3=-1\\ \mathrm{x}_4=(1,1)^{\mathrm{T}},\ y_4=-1\end{cases}\qquad \text{AND 분류}\begin{cases}\mathrm{x}_1=(0,0)^{\mathrm{T}},\ y_1=-1\\ \mathrm{x}_2=(1,0)^{\mathrm{T}},\ y_2=-1\\ \mathrm{x}_3=(0,1)^{\mathrm{T}},\ y_3=-1\\ \mathrm{x}_4=(1,1)^{\mathrm{T}},\ y_4=\ \ 1\end{cases}$$

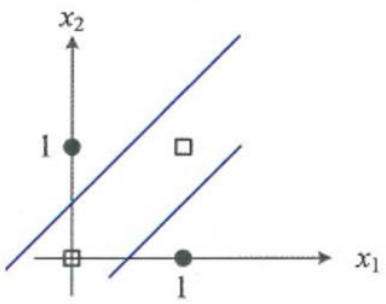
2 식 (3.7)에 있는 퍼셉트론의 목적함수를 다음과 같이 다르게 정의할 수 있다. 이 식을 미분하는 과정을 보이고, 미분 결과를 사용하여 가중치 갱신 규칙을 식 (3.9)처럼 제시하시오.

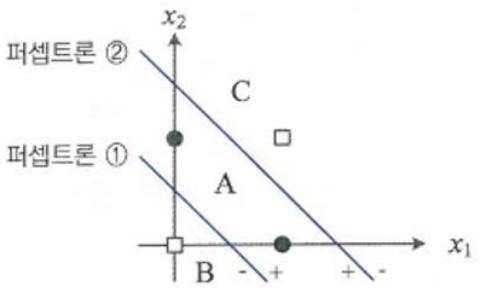
$$J(\mathbf{w})=\sum_{i=1}^{n}\|y_i-\tau(\mathbf{w}^{\mathrm{T}}\mathrm{x}_i)\|_2^2$$

$$J(\mathbf{w})=\sum_{\mathrm{x}_k\in Y}-y_k(\mathbf{w}^{\mathrm{T}}\mathrm{x}_k) \qquad (3.7)$$

델타 규칙: $w_i=w_i+\ \rho\sum_{\mathrm{x}_k\in Y}\mathrm{y}_k\mathrm{x}_{ki}\,,\qquad i=0,1,\cdots,d \qquad (3.9)$

3 XOR 문제는 [그림 3-8(a)] 대신 다음 그림과 같이 해결할 수도 있다. 이 그림에 해당하는 다층 퍼셉트론을 제시하시오.





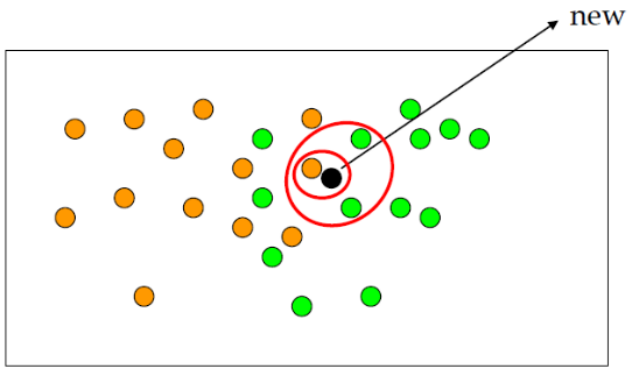
(a) 퍼셉트론 2개를 이용한 공간분할

그림 3-8 XOR 문제의 해결

4 분류 문제를 푸는 $k-\mathrm{NN}^{k-\text{nearest neighbor}}$ 알고리즘이 있다. 이 알고리즘은 테스트 샘플에 가장 가까운 k개의 샘플을 훈련집합에서 찾은 다음, k개의 샘플의 부류를 보고 가장 빈도가 높은 부류로 분류한다.

(1) $k-\mathrm{NN}$을 아래 예제에서 설명하는 알고리즘 형태로 제시하시오.

ex) k−NN은 새로운 데이터가 주어졌을 때 기존 데이터 가운데 **가장 가까운 k개 이웃의 정보로 새로운 데이터를 예측하는 알고리즘**으로, 아래 그림처럼 검은색 점의 범주 정보는 주변 이웃들을 가지고 추론해낼 수 있다. 예를 들어, 점 간의 거리를 계산하는 공식으로 Euclidean Distance($\mathrm{D}(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$)을 이용한다고 할 때에, 만약 k가 1이라면 가장 가까운 1개의 점은 오렌지색이므로 오렌지색으, k가 3이라면 가장 가까운 3개의 점 중 2개의 점이 녹색이므로, 가장 수가 많은 녹색으로 **분류**(Classification)하여야 한다.

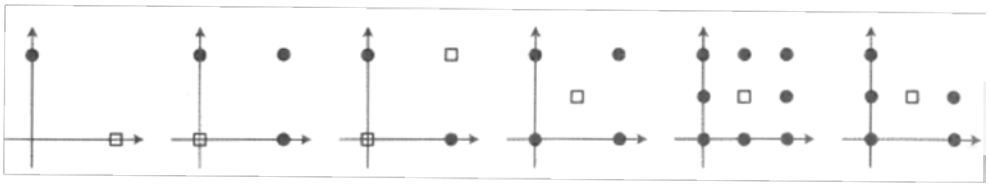


(2) $k-\mathrm{NN}$ 은 훈련집합을 모두 메모리에 저장하고 있어야 하는 메모리 기반 방법이다. MNIST를 훈련집합으로 사용할 때 메모리양을 산정하시오.(MNIST 훈련집합은 6만 개 샘플을 가지는데, 한 샘플은 784차원 특징 벡터로 표현된다.)

(3) 초당 1억 개의 사칙 연산을 수행하는 컴퓨터에서 $5-\mathrm{NN}$으로 테스트 샘플 하나를 분류하는 데 걸리는 시간을 추정하시오.

(4) 다음 6가지 상황에 대해 $1-\mathrm{NN}$이 공간을 어떻게 분할하는지 그리시오.

Hint 보로노이 도형 Voronoi diagram 으로 분할한다.



(실습)_____

1. 아래의 참고비디오를 시청 후, 아래의 소스코드를 실행하시오. 각 line 별로 주석을 작성하시오.

참고 비디오 (Lecture 2):
https://www.youtube.com/watch?v=I-Fe9Ekxxj4&list=PLlMkM4tgfjnJ3I-dbhO9JTw7gNty6o_2m&index=3

1) gradient(x, y) 함수의 기능과, 수식의 의미를 설명하시오.

2) w 초기값을 [0.01, 0.1, 1, 100]으로 변화 시켰을 때의 각 w에 따른 그래프를 도시하고, w의 초기값이 학습에 어떤 영향을 주는지 분석하시오.

- (그래프1) x축: epoch, y축: w, (그래프2) x축: epoch, y축: loss 으로 도시

3) 학습률(lr) 값을 [0.01, 0.1, 1, 100]으로 변화 시켰을 때의 각 학습률에 따른 그래프를 도시하고, 학습률이 학습에 어떤 영향을 주는지 분석하시오.

- (그래프1) x축: epoch, y축: w, (그래프2) x축: epoch, y축: loss 으로 도시

```
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0  # a random guess: random value
lr = 0.1 # learning rate
# our model forward pass


def forward(x):
    return x * w


# Loss function
def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)


# compute gradient
```

```python
def gradient(x, y):  # d_loss/d_w
    return 2 * x * (x * w - y)

# Before training
print("predict (before training)",  4, forward(4))

# Training loop
for epoch in range(10):
    for x_val, y_val in zip(x_data, y_data):
        grad = gradient(x_val, y_val)
        w = w - lr * grad
        print("Wtgrad: ", x_val, y_val, round(grad, 2))
        l = loss(x_val, y_val)

    print("progress:", epoch, "w=", round(w, 2), "loss=", round(l, 2))

# After training
print("predict (after training)",  "4 hours", forward(4))
```

2. 아래의 참고비디오를 시청 후, 아래의 소스코드를 실행하시오. 각 line 별로 주석을 작성하시오.

참고 비디오 (Lecture 3):
https://www.youtube.com/watch?v=b4Vyma9wPHo&list=PLlMkM4tgfjnJ3I-dbhO9JTw7gNty6o_2m&index=4

1) 실습 1번 코드와 다른 부분을 명시 하시오. Line 31의 w.grad.data는 어떤 값을 가지는가.

2) Autograd의 기능에 대해 설명하시오.

```
import torch
from torch.autograd import Variable

x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = Variable(torch.Tensor([1.0]),  requires_grad=True)  # Any random value

# our model forward pass


def forward(x):
    return x * w

# Loss function


def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)

# Before training
print("predict (before training)",  4, forward(4).data[0])

# Training loop
for epoch in range(10):
    for x_val, y_val in zip(x_data, y_data):
        l = loss(x_val, y_val)
        l.backward()
        print("Wtgrad: ", x_val, y_val, w.grad.data[0])
        w.data = w.data - 0.01 * w.grad.data

        # Manually zero the gradients after updating weights
        w.grad.data.zero_()

    print("progress:", epoch, l.data[0])

# After training
print("predict (after training)",  4, forward(4).data[0])
```