MYONGJI UNIVERSITY

기초2 + Trend

2018. 7. 9

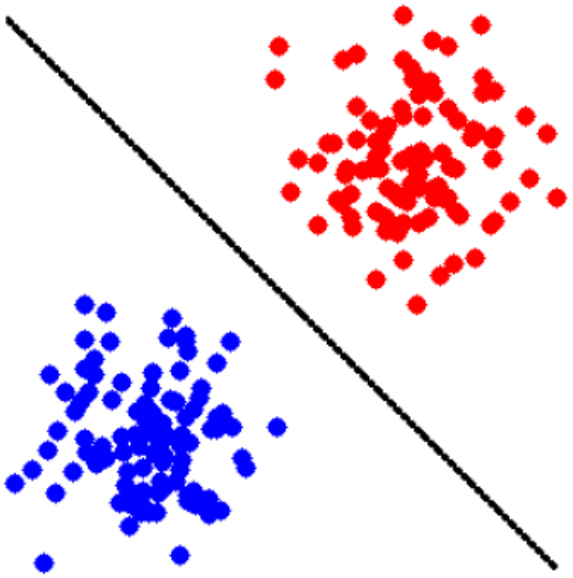
Lecture Notes: http://eclass.mju.ac.kr

목차

- 간단한 분류 모델
- Neural Network 소개
- CNN 모델 소개
- Software 2.0에 대한 개념 소개 (Andrey Karpathy)

간단한 분류

- 선형 분류기



- 주어진 데이터 셋 X에 대해서, 두 개의 집합(빨간색, 파란색)을 가질 때, 두 집합을 분류하는 모델을 구하고자 함

목적 함수

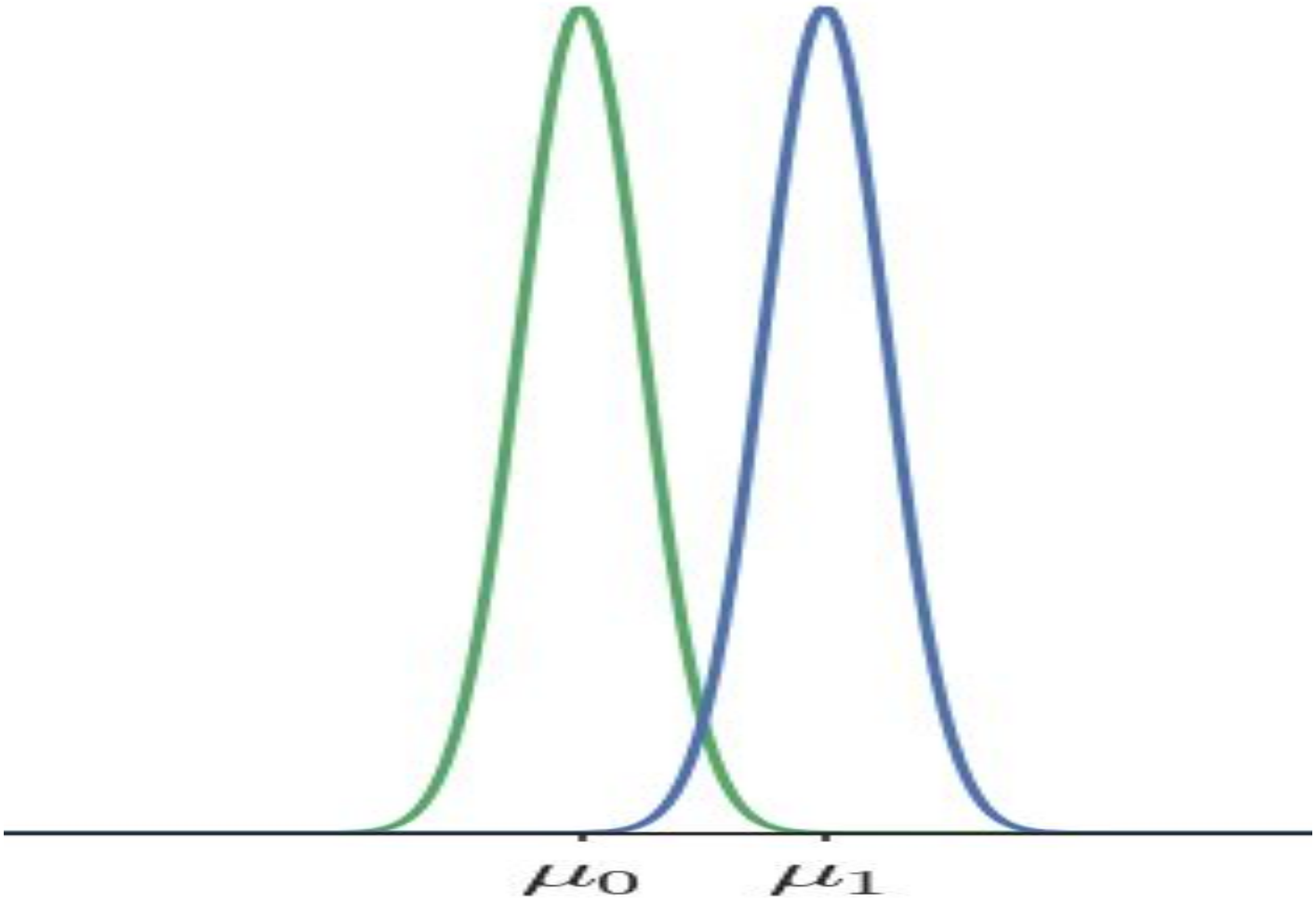
- 입력(input) : N개의 2차원 point
- 출력(label) : 빨간색 / 파란색

- 모델 : 선형 모델 (W x + b)

  ⇨ 구해야 할 값은 W와 b

- 목표: 분류를 잘하자
  - 정확한 분류의 개수?
  - 분류간 경계가 명확?

완벽한 분류는 존재하나?



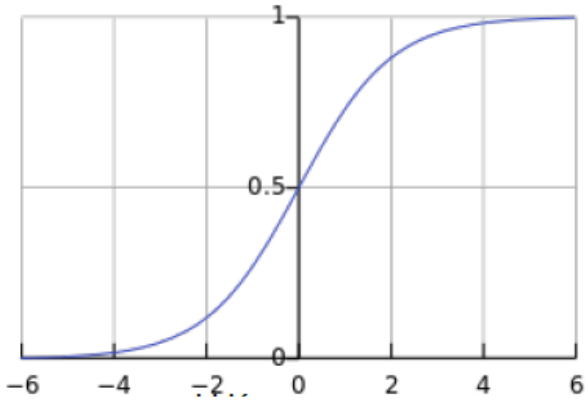
Logistic Regression (LR)

- LR은 통계 모델의 하나로 binary 분류에 활용
  즉, 두 개의 class를 정의 (y = 1 or 0)

- 확률 개념을 도입하여 p(y = 1 | x) 일 확률과 p(y = 0 | x)일 확률을 계산하여 두 개 확률 중 큰 값으로 class를 결정
  (p(y = 0 | x) = 1 – p(y = 1 | x))

- Logistic regression은 sigmoid 함수를 사용

$$p(y = 1|x; \theta) = \sigma(\theta^T x) = \frac{1}{1+exp(-\theta^T x)}$$



Cross Entropy

- 두 개의 class 0과 class 1 분류시, class 1의 예측 값은

$$\mathbf{h_\theta(x_i) = \sigma(W\,x_i + b)}$$

로 정의하고, 해당 결과값을 확률로 여긴다면,

$$p(y_i = 1|x_i) = h_\theta(x_i) \qquad p(y_i = 0|x_i) = 1 - h_\theta(x_i)$$

클래스 1이 될 확률 클래스 0이 될 확률

- 이를 하나의 식으로 표시하면

$$p(y_i|x_i) = [h_\theta(x_i)]^{(y_i)}[1 - h_\theta(x_i)]^{(1-y_i)}$$

클래스 {0,1}이 해당 클래스에 맞도록 분류될 확률

Cross Entropy

- 입력 데이터 N개가 모두 독립이라면(i.i.d.), 주어진 입력과 출력이 $\mathbf{h_\theta}$에 의해 나올 확률은

$$L(x,y) = \prod_{i=1}^{N} [h_\theta(x_i)]^{(y_i)} [1 - h_\theta(x_i)]^{(1-y_i)}$$

- 계산을 위해서 log를 취하면

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

클래스 1이 클래스 1으로 분류될 확률

클래스 0이 클래스 0으로 분류될 확률

Cross Entropy : Multiclass

- 지금까지 두 개의 클래스 {yi = 1, yi = 0} 경우

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

클래스 1일 확률 — 클래스 1이 아닐 확률 = 클래스 0일 확률

- 다양한 class가 있는 경우에는, 각 클래스에 대해 정확히 분류될 cross-entropy를 정의하고, 이를 최소화

$$-\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} \log(h_\theta(x_i)_j)$$

클래스 j로 분류될 확률

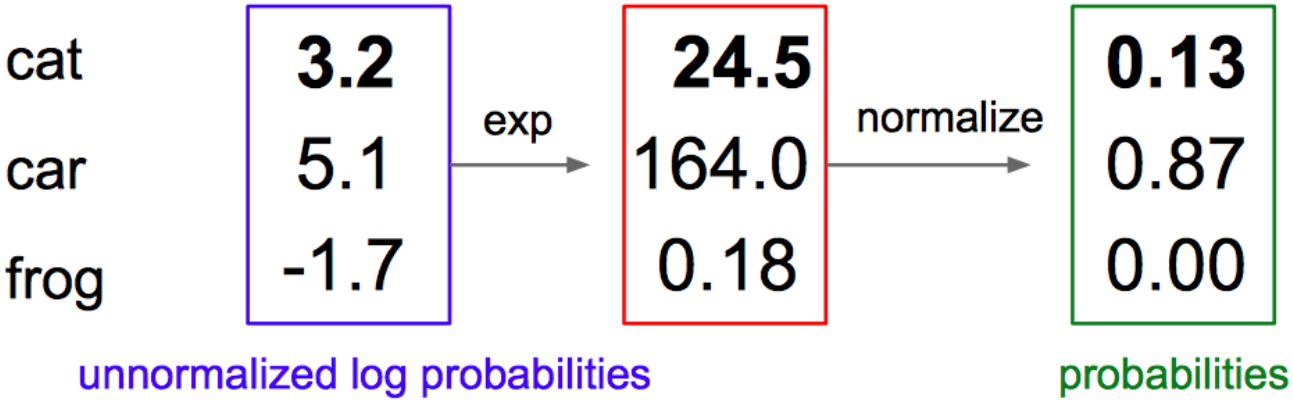
내용은 어렵지만, 활용은 쉽다.

- Y_pred = tf.matmul(X, W) + b
- tf.losses.softmax_cross_entropy(Y_true, Y_pred)

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

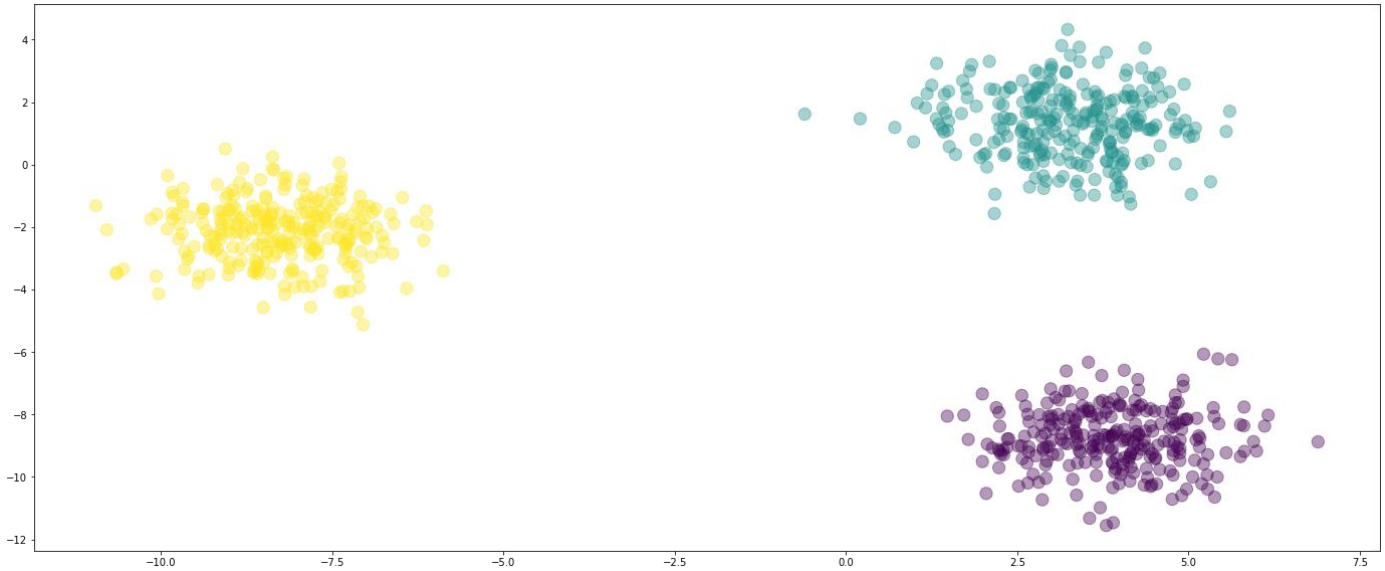


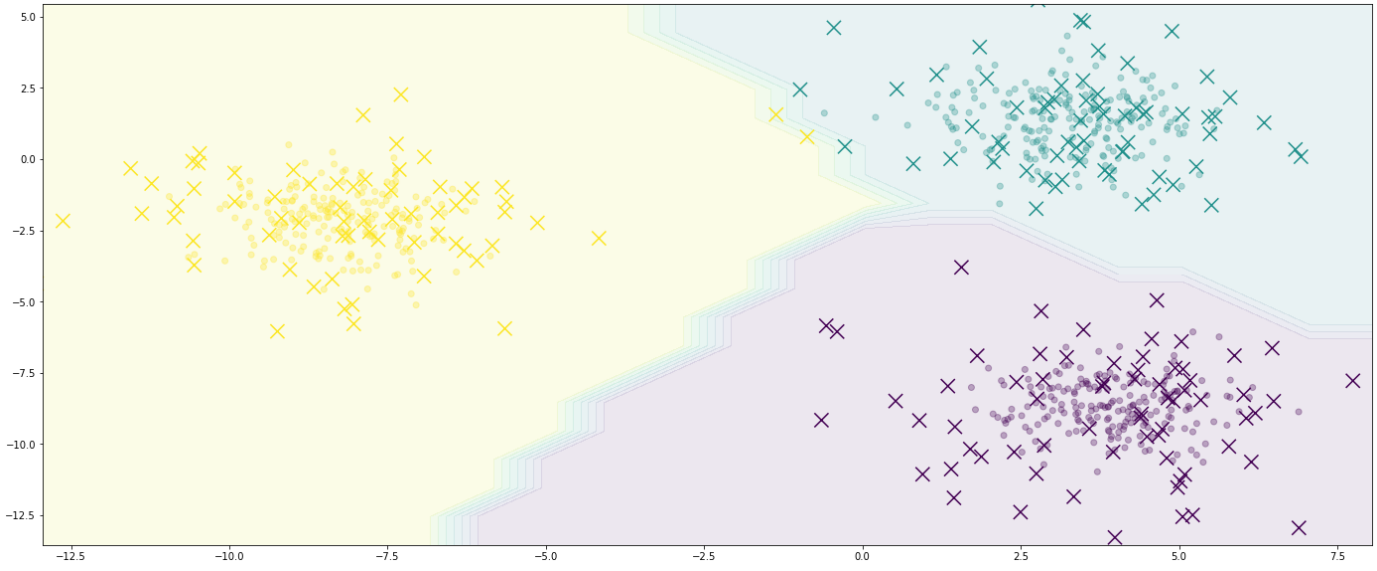
샘플 코드

```
import tensorflow as tf

n_features = X_values.shape[1]
n_classes = len(set(y_flat))

weights_shape = (n_features, n_classes)
bias_shape = (1, n_classes)

X = tf.placeholder(dtype=tf.float32)
Y_true = tf.placeholder(dtype=tf.float32)

W = tf.Variable(dtype=tf.float32, initial_value=tf.random_normal(weights_shape))  # Weights of t
b = tf.Variable(dtype=tf.float32, initial_value=tf.random_normal(bias_shape))

Y_pred = tf.matmul(X, W)  + b

loss_function = tf.losses.softmax_cross_entropy(Y_true, Y_pred)
learner = tf.train.GradientDescentOptimizer(0.1).minimize(loss_function)
```

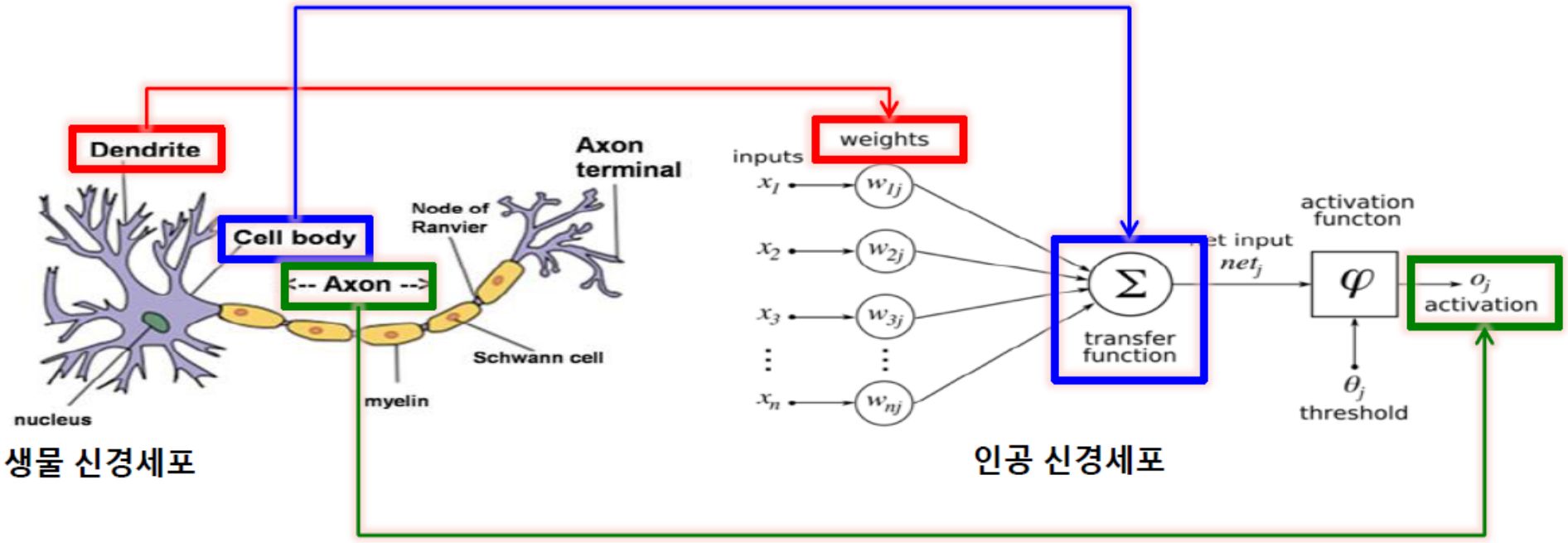
```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(5000):
        result = sess.run(learner, {X: X_train, Y_true: y_train})
        if i % 100 == 0:
            print("Iteration {}:\tLoss={:.6f}".format(i, sess.run(loss_function, {X: X_test, Y_t
    y_pred = sess.run(Y_pred, {X: X_test})
    W_final, b_final = sess.run([W, b])
```

샘플 코드


4
2
0
−2
−4
−6
−8
−10
−12
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5


5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

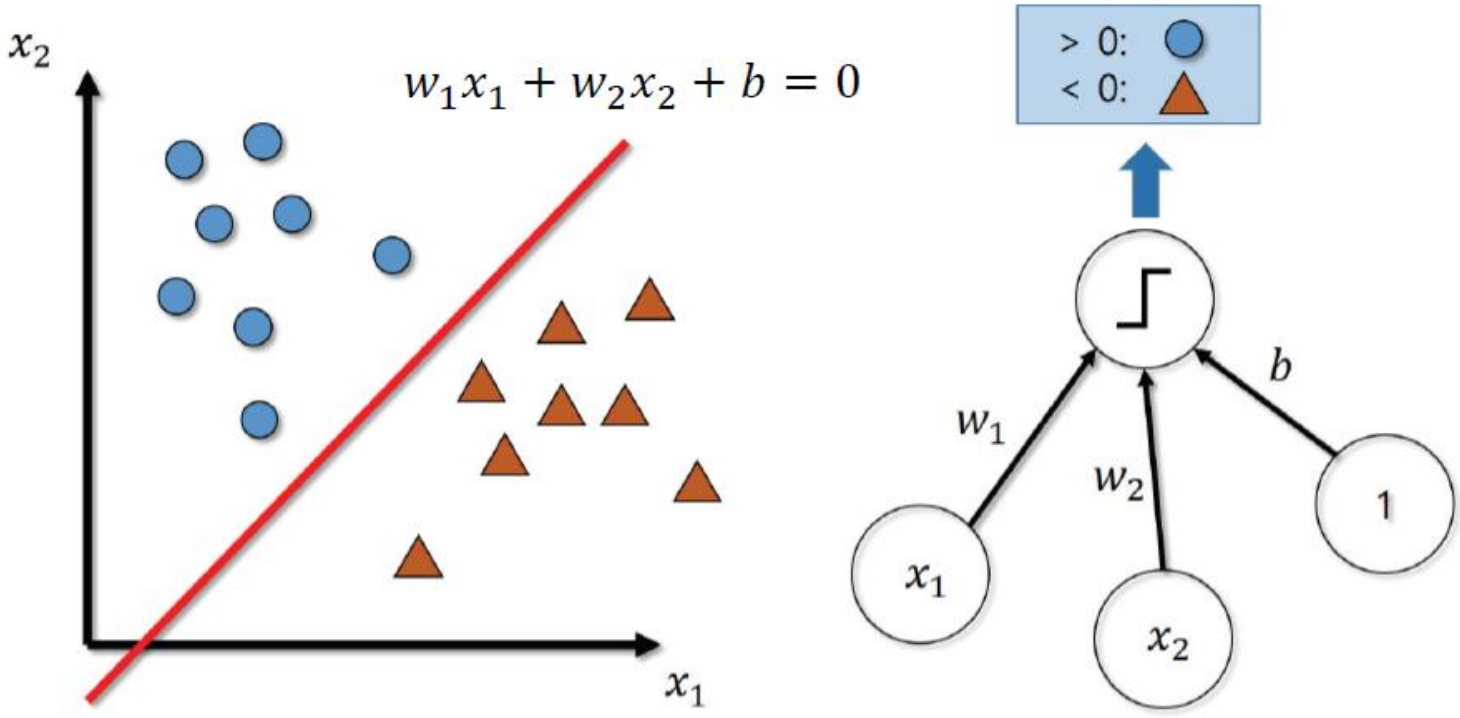
2.1 퍼셉트론



인공 신경세포 (Artificial Neuron)

2.1 퍼셉트론

- 단순 퍼셉트론의 작동 원리



2.2 델타 규칙

- 델타규칙에 의한 학습 알고리듬
  - 학습 데이터 N개

$$D_N = \left\{\left(\mathbf{x}^{(d)}, y^{(d)}\right)\right\}_{d=1}^{N}$$

  - 가중치 벡터 w를 갖는 선형 유닛의 계산

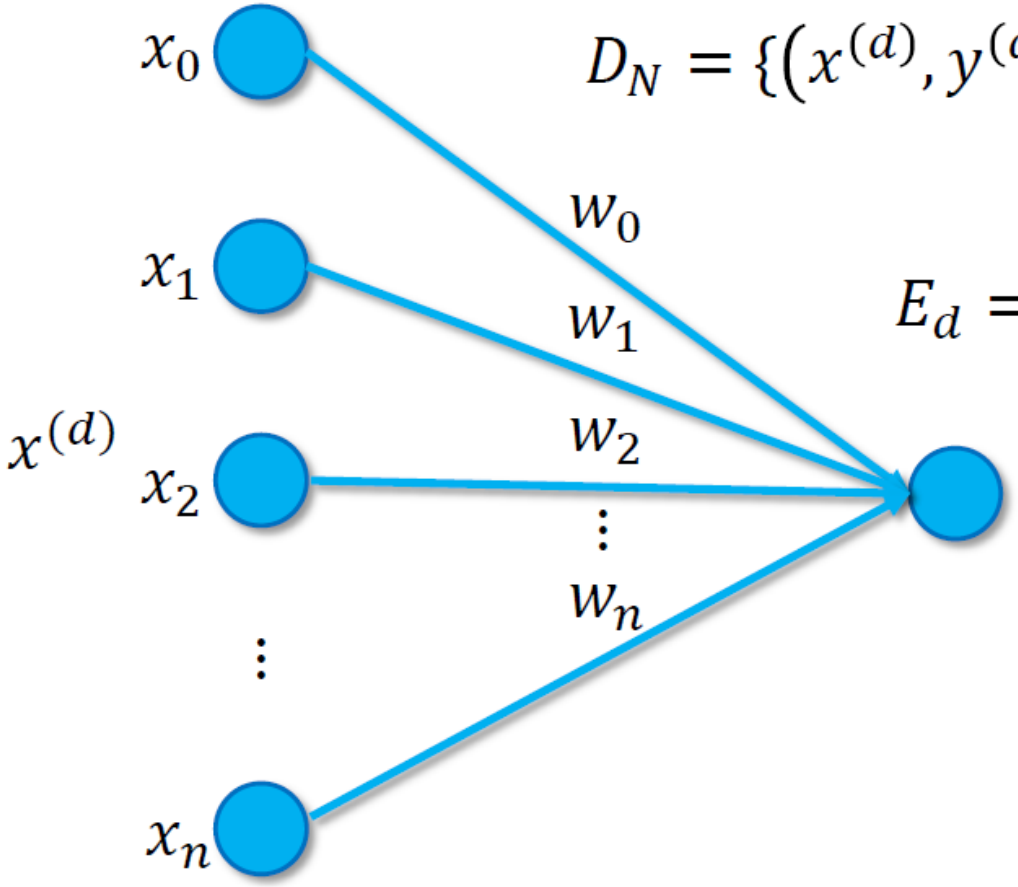
$$f^{(d)} = f\left(\mathbf{x}^{(d)}; \mathbf{w}\right) = \sum_i w_i x_i^{(d)}$$

  - 학습 데이터집합에 대한 오차 E

$$E_N = \sum_{d=1}^{N} E_d \qquad E_d = \frac{1}{2}\left(y^{(d)} - f^{(d)}\right)^2$$

2.2 델타 규칙

- 선형퍼셉트론의 구조



$$D_N = \{(x^{(d)}, y^{(d)})\}_{d=1}^{N}$$

$$E_d = \frac{1}{2}(y^{(d)} - f^{(d)})^2$$

$$f^{(d)} \Leftrightarrow y^{(d)}$$

$$f^{(d)} = f(x^{(d)}; w) = \sum_i w_i x_i$$

2.2 델타 규칙

- 델타규칙에 의한 학습 알고리듬(cont.)
  - **오차를 줄여주는 방향**으로 시냅스 가중치를 교정

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E_d}{\partial w_i}$$

오차를 가중치 w_i에 대해서 미분

$$\frac{\partial E_d}{\partial w_i} = \frac{\partial E_d}{\partial f^{(d)}} \frac{\partial f^{(d)}}{\partial w_i} = \frac{\partial}{\partial f^{(d)}} \frac{1}{2} \left(y^{(d)} - f^{(d)}\right)^2 \frac{\partial f^{(d)}}{\partial w_i}$$

$$= \frac{1}{2}(-2)\left(y^{(d)} - f^{(d)}\right) x_i^{(d)} = -\left(y^{(d)} - f^{(d)}\right) x_i^{(d)}$$

2.2 델타 규칙

- 델타규칙에 의한 학습 알고리듬(cont.)
  - i번째 가중치의 학습식
    - $\eta \in (0,1)$ 은 학습률 (learning rate)

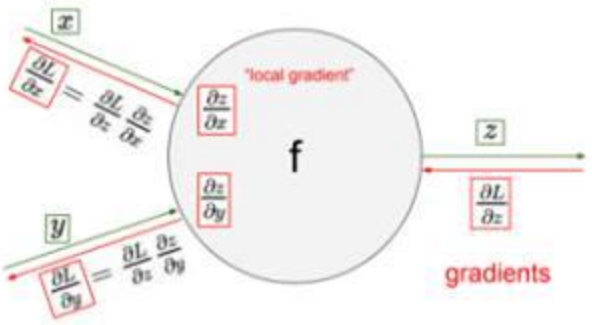
$$w_i \leftarrow w_i + \eta\left(y^{(d)} - f^{(d)}\right)x_i^{(d)}$$

Backpropagation
How NN's learn

To let the computers learn the right parameters to approximate a function, we use a set of *training dataset* and *loss function*

- Training dataset: $T = \{(x_i, y_i) | x_i \in X, y_i \in Y\}$
- Loss function: $\ell(\hat{y}, y)$ measures how *far* a prediction $\hat{y} = NN(x)$ is from the truth y

"Learning" is now an *optimization problem*; given T, find a set of NN params. that minimize $\ell_{\text{overall}} = \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{y}_i, y_i)$.



Use *chain rule of derivatives*:

$$\frac{\partial \ell_{\text{overall}}}{\partial h_j} = \frac{\partial \ell_{\text{overall}}}{\partial h_{j+1}} \times \frac{\partial h_{j+1}}{\partial h_j}$$

recursively backwards through the layers of NN

내용은 어렵지만, 활용은 쉽다.

- Y_pred = tf.matmul(X, W) + b
- cost = tf.losses.softmax_cross_entropy(Y_true, Y_pred)

- learner = tf.train.GradientDescentOptimizer(0.1).minimize(cost)

Convolution Neural Network

딥러닝 동향

- [포즈인식/딥러닝]
  https://www.youtube.com/watch?v=EMjPqgLX14A

- [자율주행/강화학습]
  https://youtu.be/eRwTbRtnT1I

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