GNU

Lab Seminar: 2024. 01.27

Advanced Learning Algorithms

(Neural Networks & training)

IDEALAB
Improving lives through learning

Su-Gyeong Ban
School of Computer Science
Gyeongsang National University (GNU)

GNU 경상국립대학교 Gyeongsang National University

Contents

- Feedback
- Neural Networks
- Perceptron
- TensorFlow
- Activation Functions
- Multiclass Classification
- Back Propagation

Feedback (1/5)

- **Implementation of MSE**
  - Scikit-learn 모듈을 사용한 코드와 사용하지 않은 코드 구현

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

```python
def mean_squared_error(y_true, y_pred):

    n = len(y_true)
    mse = sum((y_true[i] - y_pred[i]) ** 2 for i in range(n)) / n

    return mse
```

라이브러리 사용하지 않고 구현한 코드

```python
from sklearn.metrics import mean_squared_error

y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]

mse = mean_squared_error(y_true, y_pred)
```

Scikit-learn 모듈을 사용하여 구현한 코드

Feedback (2/5)

- **Implementation of RMSE**
  - Scikit-learn 모듈을 사용한 코드와 사용하지 않은 코드 구현

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

```python
def root_mean_squared_error(y_true, y_pred):

    n = len(y_true)
    mse = sum((y_true[i] - y_pred[i]) ** 2 for i in range(n)) / n
    rmse = mse ** 0.5

    return rmse
```

라이브러리 사용하지 않고 구현한 코드

```python
from sklearn.metrics import mean_squared_error

def root_mean_squared_error_sklearn(y_true, y_pred):

    mse = mean_squared_error(y_true, y_pred)
    rmse = mse ** 0.5

    return rmse
```

Scikit-learn 모듈을 사용하여 구현한 코드

Feedback (3/5)

- **Overfitting Solutions**
  - 더 많은 데이터 수집
    - 모델이 특정 데이터에 과도하게 적합하지 않고 일반화되도록 다양한 데이터 수집
  - 모델 복잡도 줄이기
    - 더 간단한 모델 구조 선택, 모델의 파라미터 수 줄이기
  - 정규화
    - 정규화 기법으로 모델의 가중치를 제한하거나 제어
    - 대표적인 방법으로 L1 정규화, L2 정규화 등이 있음

Feedback (4/5)

- **Regularization (1/2)**
  - L1 regularization
    - 가중치의 절댓값을 제약하는 방식
    - 가중치의 절댓값의 합에 비례하여 정규화 항 추가
    - 특정 가중치를 0으로 만들어 feature selection 효과를 줌으로써 불필요한 특성 제거
      - Lasso Regression는 선형 회귀 모델에 L1 정규화를 적용한 것

$$\text{L1: } \Omega(w) = \sum_{i=1}^{n} |w_i|$$

Feedback (5/5)

- **Regularization (2/2)**
  - L2 regularization
    - 가중치의 제곱을 제약하는 방식
    - 가중치의 제곱의 합에 비례하여 정규화 항 추가
    - 가중치를 0으로 만들지 않고 작은 값으로 만들어 모든 특성이 조금씩 모델에 영향을 줄 수 있도록 함

      - Ridge Regression는 선형 회귀 모델에 L2 정규화를 적용한 것

$$\text{L2: } \Omega(w) = \sum_{i=1}^{n} w_i^2$$

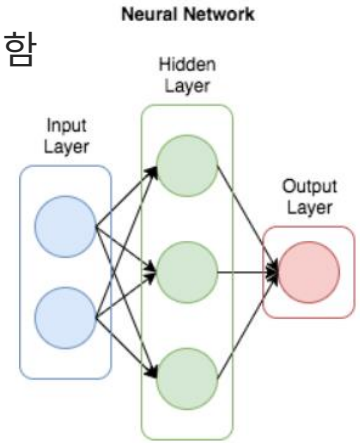
Neural Networks (1/4)

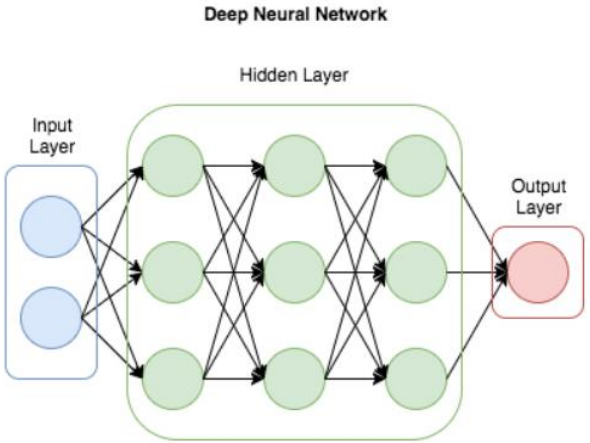
- **Hypothesis that mimics the brain**
  - 가설 1 : 뇌가 하는 모든 매력적이고 놀라운 것들을 모방하기 위해 수많은 소프트 웨어가 필요하다는 것
  - 가설 2 : 뇌가 배우는 방식을 모방하는 단 하나의 학습 알고리즘을 만드는 것
    - 신경 재배선 실험

      - 뇌는 청각을 담당하는 영역이나 촉각을 담당하는 영역에 시각 정보를 보내면 보는 법을 배움
    - 물리적으로 같은 뇌가 시각, 청각, 촉각을 처리할 수 있다는 것은 2번째 가설이 맞다는 것을 말함

Neural Networks (2/4)

- **What is Neural Networks?**
  - 사람의 뇌를 모방할 수 있는 기계를 만들기 위한 아주 오래된 알고리즘
  - 데이터를 처리하도록 컴퓨터를 가르치는 인공지능 방식
  - 입력층, 은닉층, 출력층으로 구성
    - 장점 : 어려운 머신러닝 문제에 잘 작동함
    - 단점 : 논리적이지 않음



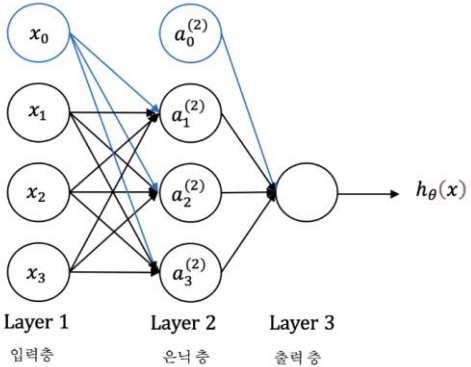


Neural Networks (3/4)

- **Input Layer**
  - 입력 벡터가 자리잡는 층
- **Output Layer**
  - 최종 출력값이 자리잡는 층
- **Hidden Layer**
  - 입력층과 출력층 사이에 위치하는 모든 층
  - 신경망에서 더욱 복잡한 연상을 위해 가상의 뉴런을 생성하는 것
    - 많을수록 신경망은 더욱 어려운 예측에 성공할 수 있음

Neural Networks (4/4)

- **What is Forward Propagation?**
  - Neural Network 모델의 입력층부터 출력층까지 정방향으로 연산을 수행하는 것
  - 최종 layer까지 계산한 후 실제 label과 오차를 계산



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) = g(z_1^{(2)})$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) = g(z_2^{(2)})$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) = g(z_3^{(2)})$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \quad a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)}x = \Theta^{(1)}a^{(1)} \qquad (x = a^{(1)})$$

$$a^{(2)} = g(z^{(2)}) = g(\Theta^{(1)}a^{(1)})$$

출력층

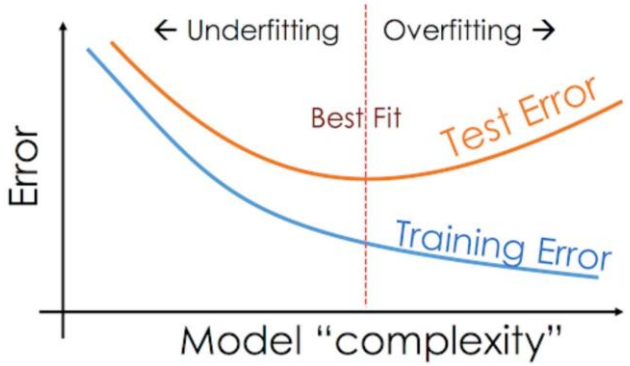
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$z^{(3)} = \Theta^{(2)}a^{(2)}$$

$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

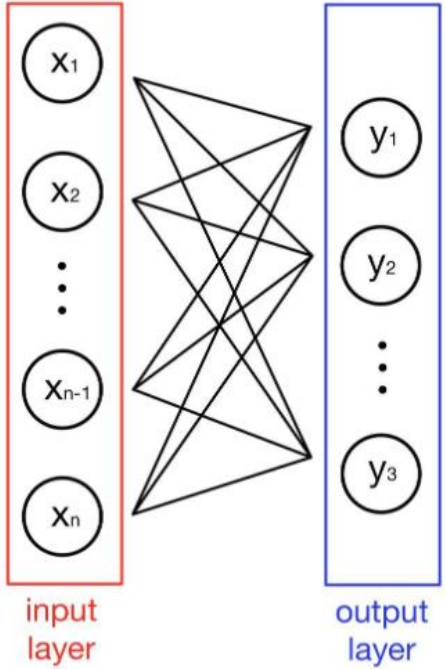
Perceptron (1/4)

- **What is Perceptron?**
  - 인지한 정보를 활용해 뉴런처럼 작동하는 인공 세포
    - 다수의 입력을 받아 하나의 신호로 출력
      - 각 입력과 상호작용하는 각각의 가중치들이 출력을 결정
      - Single-Layer Perceptron, Multi-Layer Perceptron
    - 은닉층과 퍼셉트론 수는 정확도와 비례하지 않음



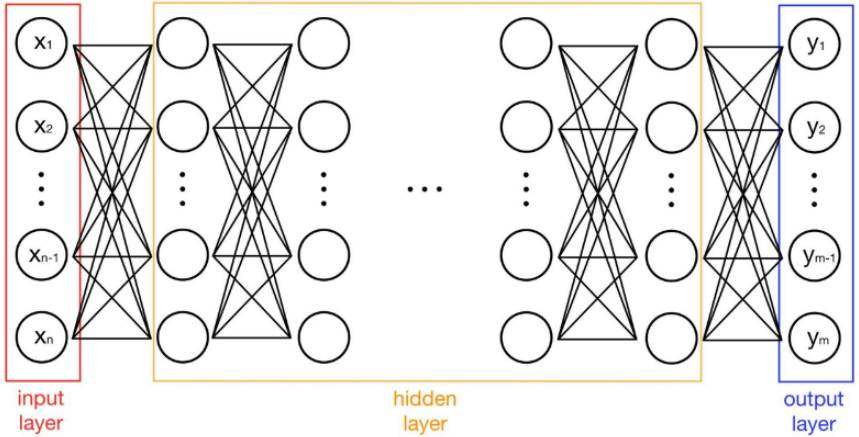
Perceptron (2/4)

- **Single-Layer Perceptron**
  - 은닉층 없이 Layer가 하나만 존재하는 퍼셉트론
  - 주어진 데이터에 대해 가장 알맞은 선형 모델을 찾음
    - 데이터의 분포가 선형이 아니라면 성능이 좋지 않음



Perceptron (3/4)

- **Multi-Layer Perceptron (1/2)**
  - 층이 2개 이상 존재하는 신경망
    - 은닉층 1개인 다층 퍼셉트론 : 얕은 신경망(shallow neural network)
    - 은닉층 2개 이상인 다층 퍼셉트론 : 깊은 신경망(deep neural network)
      - 깊은 신경망을 학습시키는 것이 딥러닝



Perceptron (4/4)

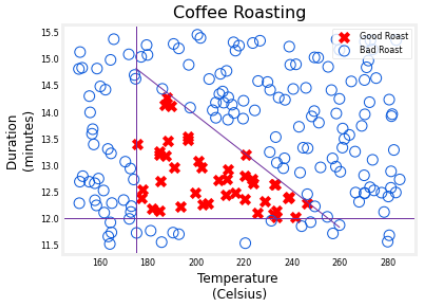
- **Multi-Layer Perceptron (2/2)**
  - 비선형으로 분포하는 데이터들에 대해 학습 가능
    - 층과 층 사이에서 선형으로 표현된 데이터를 비선형으로 바꿔주는 과정 필요
      - 가중치에 대해 선형방정식을 계산하기 때문
      - 활성화 함수가 이 과정 담당
      - 활성화 함수 : sigmoid 함수, softmax 함수, ReLU 함수, ELU 함수, tanh 함수 등

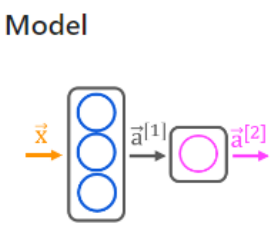
TensorFlow (1/2)

- **What is TensorFlow?**
  - Google에서 제공하는 머신러닝 프레임워크
  - 인공 신경망같은 기계학습 응용프로그램 및 딥러닝에 사용
  - 기본적으로 C++로 구현되어 있으며 Python, Java 등 다양한 언어 지원
    - 계산 구조, 목표 함수만 정의하면 자동으로 미분 계산 처리

TensorFlow (2/2)

- **ex) Coffee roasting**
  - tf.keras.Input(shape=(2,))
    - Tensorflow가 가중치 및 편향 매개변수의 크기 조절
    - 입력 데이터가 model.fit 문에서 지정될 때, Tensorflow가 네트워크 매개변수의 크기를 조정
  - Sigmoid function를 포함하는 것이 최적의 방법은 아님
  - 손실함수에 의해 수치적 안정성 향상시킴





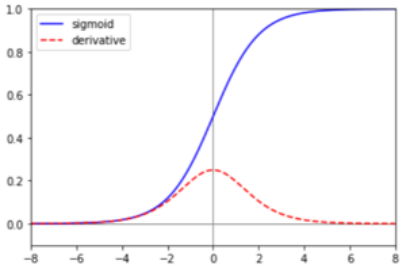

```
model = Sequential(
    [
        tf.keras.Input(shape=(2,)),
        Dense(3, activation='sigmoid', name = 'layer1'),
        Dense(1, activation='sigmoid', name = 'layer2')
    ]
)
```

```
yhat = np.zeros_like(predictions)
for i in range(len(predictions)):
    if predictions[i] >= 0.5:
        yhat[i] = 1
    else:
        yhat[i] = 0
```

Activation Functions (1/3)

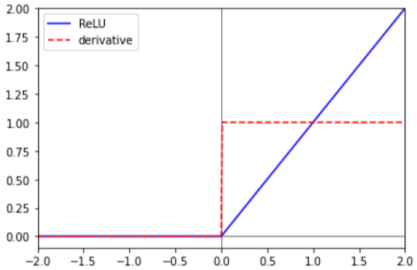
- **Sigmoid function**
  - 입력값을 0과 1사이의 값으로 변환
  - 입력값의 절댓값이 크면 기울기 0에 수렴
  - 기울기 소실 현상 발생

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$



- **ReLU(Rectified Linear Unit) function**
  - 0보다 큰 입력값의 경우 그대로 출력
  - 0 이하의 값은 다음 층에 전달 X (dying ReLU 현상)
  - Sigmoid function이 가지고 있는 기울기 소실 현상 완화
  - CNN(Convolution Neural Network)에 자주 쓰임

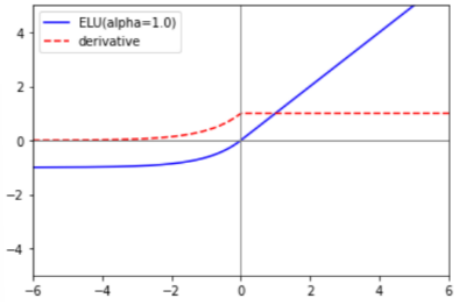
$$ReLU(x) = \max(0, x)$$

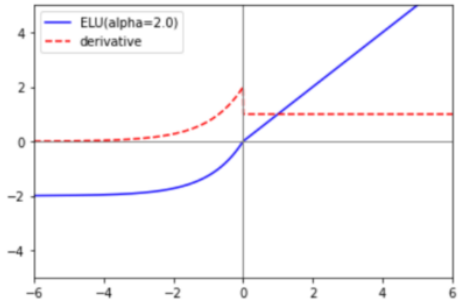


Activation Functions (2/3)

- **ELU(Exponential Linear Unit) function**
  - ReLU의 장점을 가지고 dying ReLU현상을 해결한 함수
  - 파라미터를 이용하여 입력값이 0이하일 때 지수함수 부분의 scale 조절 가능
    - 알파가 1인 경우, 연속인 도함수를 가짐
    - 알파가 1 이외의 경우, SeLU(Scaled ELU) 라고 부르며 불연속인 도함수를 가짐

$$ELU(x) = \begin{cases} x & (x > 0) \\ \alpha(e^x - 1) & (x \leq 0) \end{cases}$$

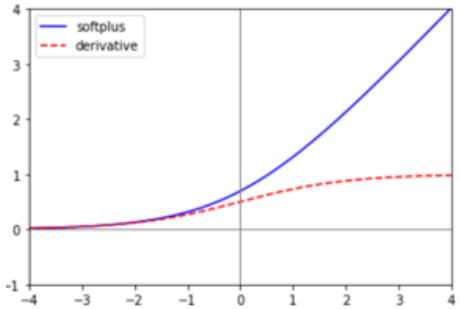




Activation Functions (3/3)

- **Softmax function**
  - ReLU function를 부드럽게 근사시킨 함수
  - 전구간에서 미분이 가능한 함수
  - 도함수는 sigmoid function가 되기 때문에 기울기가 0~1 사이 값을 가짐

$$softmax(x) = \ln(1 + e^x)$$



Multiclass Classification (1/2)

- **Binary Classification**
  - 타깃의 값이 어떤 기준에 대하여 참 또는 거짓의 값을 가지는 것
- **Multiclass Classification**
  - 타깃이 가질 수 있는 값이 3개 이상
  - 타킷의 종류가 여러 개이기 때문에 출력 값도 여러 개
  - 입력 값 하나당 하나의 클래스에만 대응될 수 있음
    - Ex) 숫자 0~9까지의 손으로 쓴 숫자 데이터 세트
      - 이진 분류 모델 : 입력 받은 숫자 사진이 0인지 아닌지 분류하는 것
      - 다중 분류 모델 : 입력 받은 숫자 사진이 0부터 9 중 어떤 숫자인지 분류하는 것

Multiclass Classification (2/2)

- **Loss function of Multiclass Classification**
  - 이진 교차 엔트로피의 일반화 버전인 크로스 엔트로피 손실함수 사용
    - 타깃이 1인 클래스를 제외한 모든 항이 0이 됨

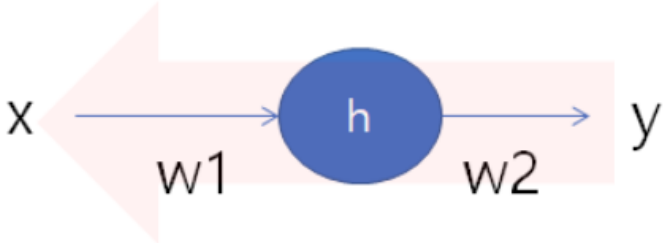
$$L = -\sum_{i=1}^{n} y_i log(a_i)$$

$$L = -log(a_{y==1})$$

Back Propagation (1/2)

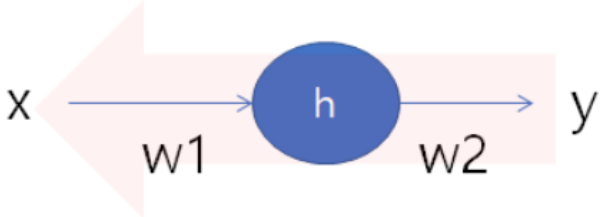
- **What is Back Propagation?**
  - 순전파 과정을 역행하는 과정
    - 오차를 기반으로 가중치 값들을 업데이터 하기 위해
    - 각 층마다 활성화 함수로 sigmoid 함수 사용

- 모델의 깊이가 깊어질수록 sigmoid의 미분 인자가 하나씩 추가되어 최종 결과인 기울기 값이 점점 작아지는 현상 발생 : 기울기 소실 현상 (Vanishing Gradient Problem)



Back Propagation (2/2)

- 손실함수를 통해 실제 정답과의 오차를 계산하여 가중치를 차례로 업데이트
  - 손실함수를 MSE, 가중치 업데이트는 경사하강법 사용
  - 역전파는 끝에서부터 계산하므로 w2에 대한 업데이트 먼저 진행
    - 경사하강법 식에 손실함수 대입 후, 편미분 연산을 기반으로 새로운 w 확정
  - w2 연산이 끝나면 w1에 대해서 같은 방식으로 연산
    - 활성화 함수 변화를 통해 기울기 소실 현상 해결



경상국립대학교

Gyeongsang National University

Improving lives through learning

IDEALAB