

# Deep Learning for Time Series Forecasting

2022.07.22.

김동현



# CONTENTS

## Book

- 01. Data Set Split
- 02. Loss Function
- 03. Numerical Differentiation
- 04. Gradient
- 05. Algorithms

<Deep Learning from Scratch 1>  
Ch 4. 신경망 학습

# 01. Data Set Split

- Training Data(훈련 데이터) / Test Data(시험 데이터)  
→ 범용적으로 사용할 수 있는 모델을 만들기 위해.
- 범용 능력  
: 아직 보지 못한 Data(Training Data에 포함되지 않는 Data)로도 문제를 올바르게 풀어내는 능력
- Overfitting  
: 한 Dataset에만 지나치게 최적화(over-optimized)된 상태

## 02. Loss Function

- NN에서 사용하는 지표
- ✓ 일반적으로 사용하는 지표 종류
  - 1) SSE : Sum of Squares for Error
  - 2) CEE : Cross Entropy Error
- 신경망 성능의 '나쁨' 을 나타내는 지표  
: NN이 training data를 얼마나 잘 처리하지 못하는지를 나타냄.

# 02. Loss Function

## 02-1. SSE

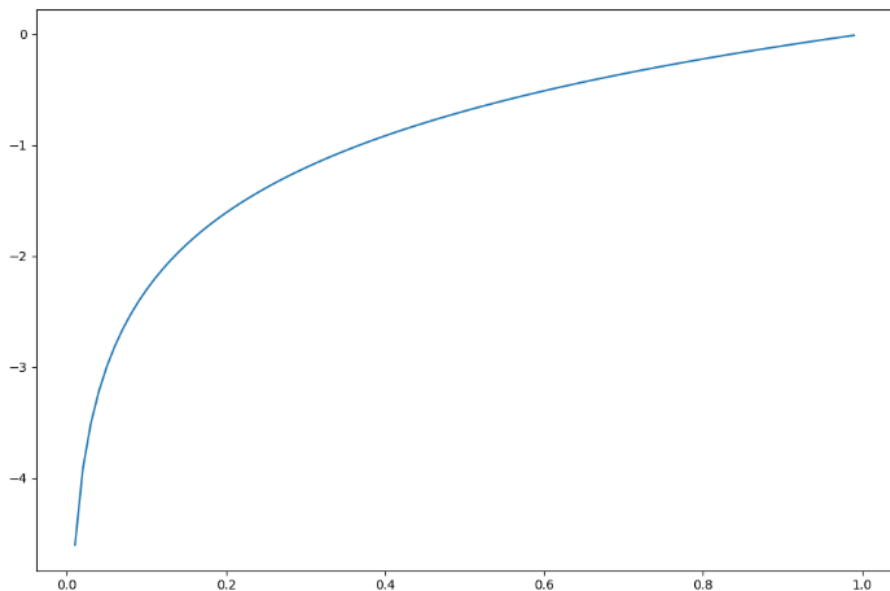
- **Formula** :  $E = \sum_k (y_k - t_k)^2$ 
  - $y_k$  : NN의 출력(추정한 값)
  - $t_k$  : 정답 레이블
  - $k$  : 데이터의 차원 수

```
[5]: # 4.2.1. 오차제곱합(SSE) #  
def sum_squares_error(y, t):  
    return 0.5 * np.sum((y-t)**2)  
  
[6]: # 정답은 '2'  
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] # One-Hot Encoding  
  
[7]: # Example 1 : '2' 일 확률이 가장 높다고 추정함(0.6)  
y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]  
sum_squares_error(np.array(y), np.array(t))  
  
[7]: 0.09750000000000003  
  
[8]: # Example 2 : '7' 일 확률이 가장 높다고 추정함(0.6)  
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]  
sum_squares_error(np.array(y), np.array(t))  
  
[8]: 0.5975
```

## 02. Loss Function

### 02-2. CEE

- **Formula :**  $E = -\sum_k t_k \log y_k$ 
  - $y_k$  : NN의 출력(추정한 값)
  - $t_k$  : 정답 레이블
  - $k$  : 데이터의 차원 수
- **Graph :**  $y = \log x$



- x가 1일 때 y는 0이 되고, x가 0에 가까워질수록 y값은 점점 작아짐.
- > CEE도 마찬가지로, 정답에 해당하는 출력이 커질수록 0에 다가가다가, 그 출력이 1일 때 0이 됨.

# 02. Loss Function

## 02-2. CEE

```
[9]: # 4.2.2. 교차 엔트로피 오차(CEE) #  
def cross_entropy_error(y, t):  
    delta = 1e-7  
    return -np.sum(t * np.log(y + delta))
```

```
[10]: # 정답은 '2'  
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] # One-Hot Encoding
```

```
[11]: # Example 1 : '2'일 확률이 가장 높다고 추정함(0.6)  
y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]  
cross_entropy_error(np.array(y), np.array(t))
```

```
[11]: 0.510825457099338
```

```
[12]: # Example 2 : '7'일 확률이 가장 높다고 추정함(0.6)  
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]  
cross_entropy_error(np.array(y), np.array(t))
```

```
[12]: 2.302584092994546
```

**Q) np.log를 계산할 때 delta를 더한 이유?**

**A)  $\log(0)$ 은  $-\infty$ 가 되어 더 이상 계산을 진행할 수 없기에, 아주 작은 값을 더해 절대 0이 되지 않도록 하기 위해서 설정해준 것.**

## 02. Loss Function

### 02-3. Mini-Batch Training

- Training Data에 대한 Loss Function 값을 구하고, 그 값을 최대한 줄여주는 매개변수를 찾아냄.

-> 대상 : 모든 Training Data

- CEE Formula :  $E = -\frac{1}{N} \sum_n \sum_k t_{nk} \log y_{nk}$

- $y_{nk}$  : NN의 출력(추정한 값)

- $t_{nk}$  : 정답 레이블

- $k$  : 데이터의 차원 수

- $N$  : 데이터 개수

- 첨자 nk : n번째 데이터의 k번째 값

- Training Data로부터 일부만 골라 학습 진행 -> Mini-Batch



## 02. Loss Function

### 02-4. Why?

- 목표 : 높은 정확도를 끌어내는 매개변수 값을 찾는 것  
Q) 'Accuracy' 대신 'Loss Function'을 택하는 이유?  
A) NN에서의 '미분(Differentiation)'의 역할에 주목하면 해결됨.
- NN 학습에서는 최적의 매개변수(가중치, 편향)를 탐색할 때, Loss Function의 값을 가능한 작게 하는 매개변수 값을 찾음.  
→ 매개변수의 미분을 계산, 그 미분 값을 단서로 매개변수의 값을 서서히 갱신하는 과정 반복.
- 의미 : 가중치 매개변수의 값을 아주 조금 변화시켰을 때, Loss Function이 어떻게 변하나?

## 02. Loss Function

### 02-4. Why?

*"신경망을 학습할 때 정확도를 지표로 삼아서는 안 된다. 정확도를 지표로 하면 매개변수의 미분이 대부분의 장소에서 0이 되기 때문이다."*

- 정확도(Accuracy)는 매개변수의 미소한 변화에는 거의 반응을 보이지 않고, 있다 하더라도 그 값이 불연속적으로 갑자기 변화함.  
→ 계단 함수(Step Function)를 활성화 함수로 사용하지 않는 이유와 동일함.
- Step Function은 한순간만 변화를 일으키지만, Sigmoid Function의 미분은 출력이 연속적으로 변하고 곡선의 기울기도 연속적으로 변함.  
→ 미분 값은 어느 장소라도 0이 되지 않음(중요한 성질).

# 03. Numerical Differentiation

## 03-1. Differentiation

- **Formula :**  $\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$

```
[18]: def numerical_diff(f, x):  
      h = 1e-4 # 0.0001  
      return (f(x+h) - f(x-h)) / (2*h)
```

- **Key points**

- 1) 미세한 값  $h = 10^{-4}$ 로 설정함. 이 정도의 값을 사용하면 좋은 결과를 얻는다고 알려져 있음.
- 2) 수치 미분에는 오차가 포함됨. 이를 줄이기 위해,  $(x+h)$ 와  $(x-h)$ 일 때의 함수  $f$ 의 차분을 계산하는 방법을 씀. **-> 중심 차분, 중앙 차분**

# 03. Numerical Differentiation

## 03-1. Differentiation

- **Example :**  $y = 0.01x^2 + 0.1x$

```
[19]: # Example #  
def function_1(x):  
    return 0.01*x**2 + 0.1*x
```

```
[20]: # Numerical Differentiation Apply #  
numerical_diff(function_1, 5)
```

```
[20]: 0.1999999999990898
```

```
[21]: numerical_diff(function_1, 10)
```

```
[21]: 0.2999999999986347
```

- 해석적 해 :  $\frac{df(x)}{dx} = 0.02x + 0.1$
- Ex 1)  $x = 5$ 일 때 0.2
- Ex 2)  $x = 10$ 일 때 0.3

# 03. Numerical Differentiation

## 03-2. Partial Differentiation

• **Example :**  $f(x_0, x_1) = x_0^2 + x_1^2$

**Q1)  $x_0 = 3, x_1 = 4$  일 때,  $x_0$  에 대한 편미분? / A) 해석적 해 :  $2x_0 = 6$**

```
[23]: def function_tmp1(x0):  
      return x0*x0 + 4.0**2.0  
  
      numerical_diff(function_tmp1, 3.0)
```

[23]: 6.000000000000378

**Q2)  $x_0 = 3, x_1 = 4$  일 때,  $x_1$  에 대한 편미분? / A) 해석적 해 :  $2x_1 = 8$**

```
[24]: def function_tmp2(x1):  
      return 3.0**2.0 + x1*x1  
  
      numerical_diff(function_tmp2, 4.0)
```

[24]: 7.999999999999119

## 04. Gradient

- 기울기(Gradient) : 모든 변수의 편미분을 벡터로 정리한 것

```
[22]: def numerical_gradient(f, x):  
      h = 1e-4 # 0.0001  
      grad = np.zeros_like(x) # x와 형상이 같은 배열을 생성  
  
      for idx in range(x.size):  
          tmp_val = x[idx]  
          # f(x+h) 계산  
          x[idx] = tmp_val + h  
          fxh1 = f(x)  
  
          # f(x-h) 계산  
          x[idx] = tmp_val - h  
          fxh2 = f(x)  
  
          grad[idx] = (fxh1 - fxh2) / (2*h)  
          x[idx] = tmp_val # 값 복원  
  
      return grad
```

```
[19]: # Example #  
      def function_2(x):  
          return x[0]**2 + x[1]**2
```

```
[27]: numerical_gradient(function_2, np.array([3.0, 4.0]))
```

```
[27]: array([6., 8.])
```

```
[28]: numerical_gradient(function_2, np.array([0.0, 2.0]))
```

```
[28]: array([0., 4.])
```

```
[29]: numerical_gradient(function_2, np.array([3.0, 0.0]))
```

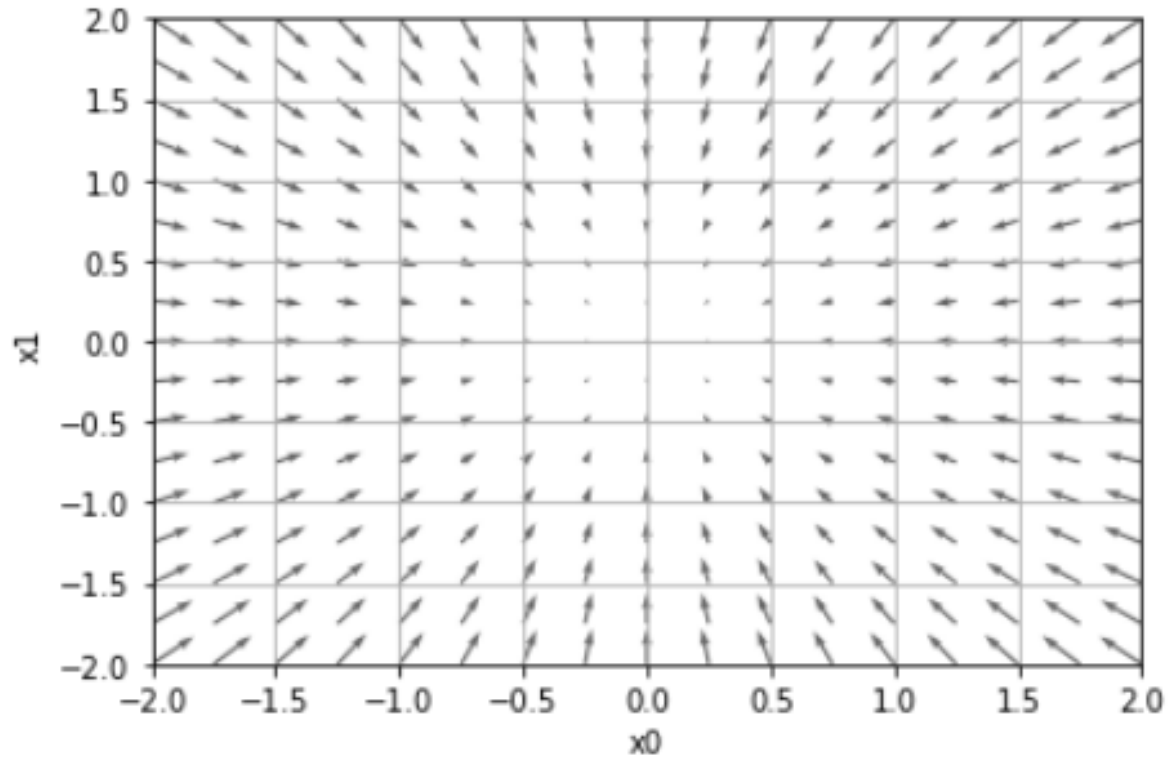
```
[29]: array([6., 0.])
```

→ 해석적 해 :  $[2x_0, 2x_1]$

- 기울기는 각 지점에서 낮아지는 방향을 가리킴.  
→ 기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 크게 줄이는 방향임.

## 04. Gradient

Q) '기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 크게 줄이는 방향'의 의미?



<그림>  $f(x_0, x_1) = x_0^2 + x_1^2$ 의 기울기

# 04. Gradient

## 04-1. Gradient Method

- 최적의 매개변수(가중치, 편향)를 학습 시에 찾아야 함.
- 일반적인 Loss Function은 복잡하고 최솟값이 되는 곳을 짐작하기 어려움.  
-> 기울기를 잘 이용해 함수의 최솟값을 찾으려는 것(Gradient Method, 경사법)
- 주의점 : 각 지점에서 함수의 값을 낮추는 방안을 제시하는 지표가 '기울기' 라는 것
  - 기울어진 방향이 꼭 최솟값을 가리키는 건 아님.
  - 그 방향으로 가야 함수의 값을 줄일 수 있음.



# 04. Gradient

## 04-1. Gradient Method

### Q) 과정?

- 현 위치에서 기울어진 방향으로 일정 거리만큼 이동.
- 이동한 곳에서도 마찬가지로 기울기를 구하고, 또 그 기울어진 방향으로 나아가기를 반복.

-> 함수의 값을 점차 줄이는 것

### • 종류

- 1) Gradient Descent Method(경사 하강법) : 최솟값 찾는 것
- 2) Gradient Ascent Method(경사 상승법) : 최댓값 찾는 것

## 04. Gradient

### 04-1. Gradient Method

- **Formula :** 
$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$
$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

–  $\eta$  : 갱신하는 양(Learning Rate, 학습률)

- 방식 : 변수의 값을 갱신하는 단계를 여러 번 반복하면서 서서히 함수의 값을 줄이는 것

#### Q) Learning Rate 설정?

- 0.01이나 0.0001 등 미리 특정 값으로 정해두어야 함.
- 일반적으로 이 값이 너무 크거나 작으면 '좋은 장소'를 찾아갈 수 없음.
- NN 학습에서는 보통 이 값을 변경하면서 올바르게 학습하고 있는지 확인하면서 진행함.

# 04. Gradient

## 04-1. Gradient Method

Q) Gradient Method을 이용한  $f(x_0, x_1) = x_0^2 + x_1^2$ 의 최솟값?

```
[32]: # Gradient Descent #
def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x = init_x

    for i in range(step_num):
        grad = numerical_gradient(f, x)
        x -= lr * grad
    return x
```

```
[33]: def function_2(x):
        return x[0]**2 + x[1]**2

        init_x = np.array([-3.0, 4.0])
        gradient_descent(function_2, init_x=init_x, lr=0.1, step_num=100)
```

```
[33]: array([-6.11110793e-10,  8.14814391e-10])
```

• **Result : (-6.1e-10, 8.1e-10)**

**-> 실제로 진정한 최솟값은 (0, 0)이므로 Gradient method로 거의 정확한 결과를 얻음.**

# 04. Gradient

## 04-1. Gradient Method

### Q) Learning Rate가 너무 크거나 작으면 어떻게 될까?

```
[36]: # 학습률이 너무 큰 예 : lr=10.0  
init_x = np.array([-3.0, 4.0])  
gradient_descent(function_2, init_x=init_x, lr=10.0, step_num=100)
```

```
[36]: array([-2.58983747e+13, -1.29524862e+12])
```

```
[37]: # 학습률이 너무 작은 예 : lr=1e-10  
init_x = np.array([-3.0, 4.0])  
gradient_descent(function_2, init_x=init_x, lr=1e-10, step_num=100)
```

```
[37]: array([-2.99999994,  3.99999992])
```

- Learning Rate가 큰 경우 : 큰 값으로 발산
  - Learning Rate가 작은 경우 : 거의 갱신되지 않은 채 끝남
- > 값을 적절히 설정해주어야 한다는 것을 알 수 있음(Hyper-parameter).

# 04. Gradient

## 04-1. Gradient Method

### Q) Hyper-parameter?

- Parameter(매개변수) : Training Data와 학습 알고리즘에 의해 자동으로 획득되는 변수
  - Hyper-parameter(초매개변수) : 사람이 직접 설정해야 하는 변수
- > Hyper-parameter는 여러 후보 값 중에서 시험을 통해 가장 잘 학습하는 값을 찾는 과정을 거쳐야 함.

# 04. Gradient

## 04-2. Gradient in NN

- NN 학습에서도 기울기를 구해야 함.
  - 기울기 : 가중치 매개변수에 대한 Loss Function의 기울기

### Example

- Shape = 2X3, 가중치 = W, Loss Function = L인 NN
  - Gradient :  $\frac{\partial L}{\partial w}$  (Loss Function L이 얼마나 변화하는지?)

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$$\frac{\partial L}{\partial w} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{23}} \end{pmatrix}$$

# 04. Gradient

## 04-2. Gradient in NN

```
[52]: # coding: utf-8
import sys, os
sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.functions import softmax, cross_entropy_error
from common.gradient import numerical_gradient

class simpleNet:
    def __init__(self):
        self.W = np.random.randn(2,3) # 정규분포로 초기화

    def predict(self, x):
        return np.dot(x, self.W)

    def loss(self, x, t):
        z = self.predict(x)
        y = softmax(z)
        loss = cross_entropy_error(y, t)

        return loss
```

• 메서드 : 2개

1) predict(x) : 예측 수행

2) loss(x, t) : 손실 함수의 값

# 04. Gradient

## 04-2. Gradient in NN

```
[63]: net = simpleNet()  
print(net.W) # 가중치 매개변수  
  
[[-0.60806075 -0.43637445 -0.89868553]  
 [-0.66366866 -0.33942224  0.62043004]]
```

```
[64]: x = np.array([0.6, 0.9])  
p = net.predict(x)  
print(p)  
print(np.argmax(p)) # 최댓값의 인덱스  
  
[-0.96213824 -0.56730469  0.01917572]  
2
```

```
[65]: t = np.array([0, 0, 1]) # 정답 레이블  
net.loss(x, t) # 손실함수의 값
```

```
[65]: 0.6580896002148787
```

- **net.W** : `np.random.randn(2, 3)`으로 나온 가중치 매개변수
- **p** : 신경망이 예측한 값
- **t** : 정답 레이블
- **net.loss(x, t)** : 손실함수의 값



# 04. Gradient

## 04-2. Gradient in NN

### Q) 기울기(Gradient)?

```
[66]: f = lambda w: net.loss(x, t)
      dW = numerical_gradient(f, net.W)
      print(dW)

[[ 0.11645742  0.17283878 -0.2892962 ]
 [ 0.17468613  0.25925817 -0.4339443 ]]
```

- $\frac{\partial L}{\partial w_{11}}$ 은 대략 0.1  $\rightarrow w_{11}$ 을  $h$ 만큼 늘리면 Loss Function의 값은  $0.1h$ 만큼 증가
- $\frac{\partial L}{\partial w_{23}}$ 은 대략  $-0.4$   $\rightarrow w_{23}$ 을  $h$ 만큼 늘리면 Loss Function의 값은  $0.4h$ 만큼 감소
- Loss Function을 줄인다는 관점에서 본다면,  $w_{11}$ 은 음의 방향으로,  $w_{23}$ 은 양의 방향으로 갱신해야 함.

**$\rightarrow$  신경망의 기울기를 구한 다음, Gradient Method에 따라 가중치 매개변수를 갱신하기만 하면 됨.**

# 05. Algorithms

## 전제

- 신경망에는 적응 가능한 가중치와 편향이 있음.
- 이 가중치와 편향을 Training Data에 적응하도록 조정하는 과정을 '학습' 이라 함.

## 1단계 : Mini-Batch(미니배치)

- Training Data 중 일부를 무작위로 가져옴.
- 목표 : Mini-Batch의 Loss Function 값을 줄이는 것

## 2단계 : Gradient(기울기) 산출

- 1단계의 목표를 위해 각 가중치 매개변수의 기울기를 구함.
- Gradient는 Loss Function의 값을 가장 작게 하는 방향을 제시함.

## 3단계 : 매개변수 갱신

- Gradient 방향으로 아주 조금 갱신함.

## 4단계 : 반복

- 1~3단계를 반복함.

-> Stochastic Gradient Descent(확률적 경사 하강법, SGD)

**THANK YOU**