머신러닝의 기본 프로세스 - 가설 정의, 손실함수 정의, 최적화 정의

- 앞 강의에서 선형 회귀 모델을 살펴보면서 모든 머신러닝 모델의 기본 프로세스에 대해 알아보았습니다.
- 이 3가지 프로세스는 제대로 숙지하지 못할 경우, 앞으로의 내용을 이해하는데 어려움이 있을 수 있는 매우 중요한 과정이므로 여기서 1번 더 언급하겠습니다.

① 학습하고자 하는 **가설**$_{\text{Hypothesis}}$ $h(\theta)$을 수학적 표현식으로 나타낸다.

② 가설의 성능을 측정할 수 있는 **손실함수**$_{\text{Loss Function}}$ $J(\theta)$을 정의한다.

③ 손실 함수 $J(\theta)$**을 최소화**$_{\text{Minimize}}$ 할 수 있는 학습 알고리즘을 설계한다.

Batch Gradient Descent, Mini-Batch Gradient Descent, Stochastic Gradient Descent

- 이 3가지 과정 중에서 손실 함수 J(θ)를 최소화할 수 있는 최적화 알고리즘으로 가장 일반적으로 사용되는 기법이 경사하강법입니다.
- 경사하강법은 다음 수식처럼 손실 함수의 미분값과 러닝 레이트의 곱만큼을 원래 파라미터에 뺀 값으로 파라미터를 한 스텝 업데이트합니다.

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1)$$

- 가설로 선형회귀 모델(y=Wx)을 사용하고 손실함수로 $\mathrm{MSE}(\frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2)$를 사용할 경우 경사하강법의 한 스텝 업데이트를 위해 계산하는 손실함수의 미분값은 아래와 같이 나타낼 수 있습니다.

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Batch Gradient Descent

- 위 수식에서 우리는 트레이닝 데이터 n개의 손실함수 미분값을 모두 더한 뒤 평균을 취해서 파라미터를 한 스텝 업데이트하였습니다.
- 이런 방법은 경사하강법의 한 스텝 업데이트할 때 **전체 트레이닝 데이터를 하나의 Batch**로 만들어 사용하기 때문에 **Batch Gradient Descent**라고 부릅니다.
- 하지만 실제 문제를 다룰 때는 트레이닝 데이터의 개수가 매우 많아 질 수 있습니다. 이런 상황에서 Batch Gradient Descent를 사용할 경우 파라미터를 한 스텝 업데이트하는데 많은 시간이 걸리게되고(한 스텝 업데이트하는데 전체 배치에 대한 미분값을 계산해야하므로), 결과적으로 최적의 파라미터를 찾는데 오랜 시간이 걸리게 됩니다.

Stochastic Gradient Descent

- 이와 반대로 경사하강법의 한스텝 업데이트를 진행할때 1개의 트레이닝 데이터만 사용하는 기법을 Stochastic Gradient Descent 기법이라고 부릅니다. Stochastic Gradient Descent 기법에서 한 스텝 업데이트를 위해 계산하는 손실함수의 미분값은 아래 수식으로 나타낼 수 있습니다.

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} (\hat{y} - y)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} (\hat{y} - y)^2$$

- Stochastic Gradient Descent 기법을 사용할 경우 **파라미터를 자주 업데이트** 할 수 있지만 파라미터를 한번 업데이트 할 때 전체 트레이닝 데이터의 특성을 고려하지 않고 각각의 트레이닝 데이터의 특성만을 고려하므로 **부정확한 방향으로 업데이트가 진행**될 수도 있습니다.

Mini-Batch Gradient Descent

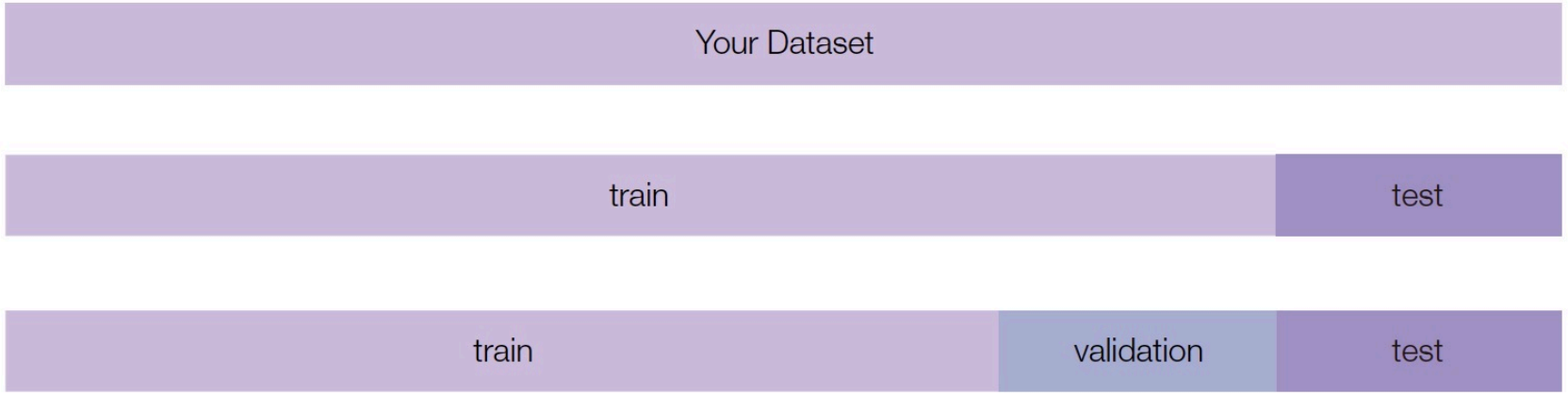
- 따라서, 실제 문제를 해결할 때는 Batch Gradient Descent와 Stochastic Gradient Descent의 절충적인 기법인 Mini-Batch Gradient Descent를 많이 사용합니다.
- Mini-batch Gradient Descent는 전체 트레이닝 데이터 Batch가 1000(n)개라면 이를 100(m)개씩 묶은 Mini-Batch 개수만큼의 손실 함수 미분값 평균을 이용해서 파라미터를 한 스텝을 업데이트하는 기법입니다.
- Mini-Batch Gradient Descent 기법에서 한 스텝 업데이트를 위해 계산하는 손실 함수의 미분값은 아래 수식으로 나타낼 수 있습니다.

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)=\frac{\partial}{\partial\theta_0}\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i-y_i)^2$$

$$\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)=\frac{\partial}{\partial\theta_1}\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i-y_i)^2$$

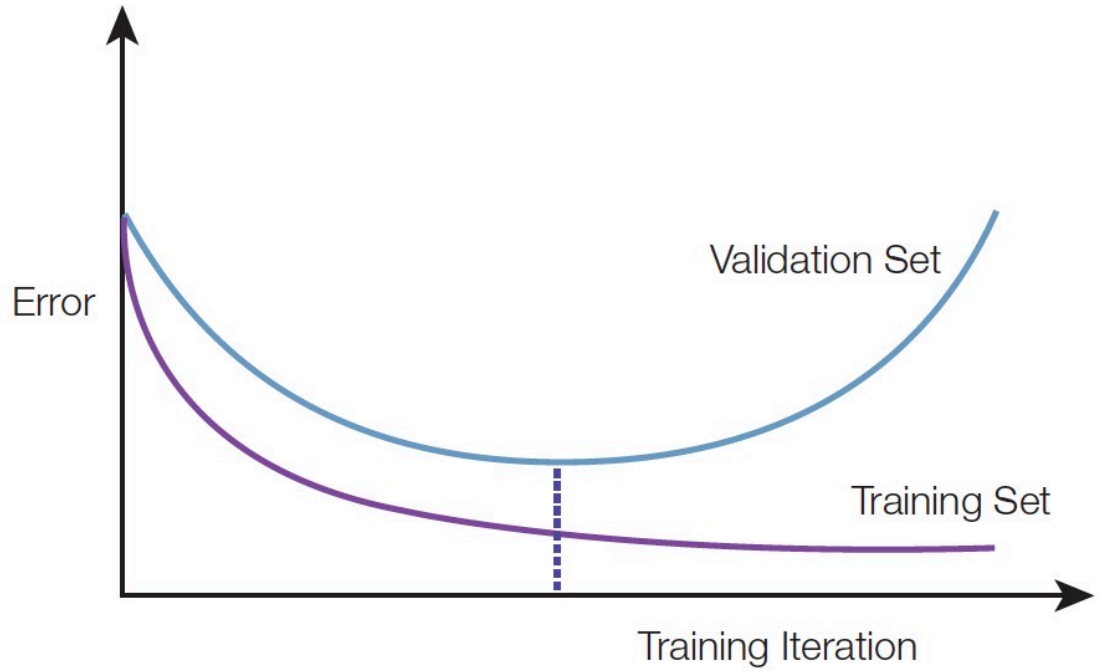
Training Data, Validation Data, Test Data

- 머신러닝 모델은 크게 **트레이닝 과정**과 **테스트 과정**으로 나뉩니다. 트레이닝 과정에서는 대량의 데이터와 충분한 시간을 들여서 모델의 최적의 파라미터를 찾습니다.
- 테스트 과정에서는 트레이닝 과정에서 구한 최적의 파라미터로 구성한 모델을 트레이닝 과정에서 보지 못한 새로운 데이터에 적용해서 모델이 잘 학습됐는지 테스트하거나 실제 문제를 풀기 위해 사용합니다. 보통 모델이 잘 학습됐는지 체크 할 때는 테스트$_{Test}$, 실제 문제를 푸는 과정을 추론$_{Inference}$이라고 부릅니다.
- 이렇게 트레이닝과 테스트를 수행하기 위해서 가지고 있는 데이터 중 일부는 트레이닝 데이터, 일부는 테스트 데이터로 나눕니다. 아래 그림은 전체 데이터를 트레이닝 데이터, 테스트 데이터로 나누는 과정을 보여줍니다.
- 여기에 더 나아가서 전체 데이터를 **트레이닝$_{training}$ 데이터, 검증용$_{validation}$ 데이터, 테스트$_{test}$ 데이터**로 나누기도 합니다.
- 검증용 데이터는 트레이닝 과정에서 학습에 사용하지는 않지만 중간중간 테스트하는데 사용해서 학습하고 있는 모델이 오버피팅에 빠지지 않았는지 체크하는데 사용됩니다. 즉, 직관적으로 설명하면 검증용 데이터는 트레이닝 과정 중간에 사용하는 테스트 데이터로 볼 수 있습니다.



Overfitting, Underfitting

- 아래 그림은 학습 과정에서 트레이닝 에러와 검증 에러를 출력한 그래프입니다.
- 처음에는 트레이닝 에러와 검증 에러가 모두 작아지지만 일정 횟수 이상 반복할 경우 트레이닝 에러는 작아지지만 검증 에러는 커지는 **오버피팅**에 빠지게 됩니다.
- 따라서 트레이닝 에러는 작아지지만 검증 에러는 커지는 지점에서 업데이트를 중지하면 최적의 파라미터를 얻을 수 있습니다.



Overfitting, Underfitting

- **오버피팅**Overfitting은 학습 과정에서 머신러닝 알고리즘의 파라미터가 트레이닝 데이터에 과도하게 최적화되어 트레이닝 데이터에 대해서는 잘 동작하지만 새로운 데이터인 테스트 데이터에 대해서는 잘 동작하지 못하는 현상을 말합니다. 오버피팅은 모델의 표현력이 지나치게 강력할 경우 발생하기 쉽습니다.
- 그림 4-3은 오버피팅, 언더피팅의 경우를 보여줍니다. 그림 4-3의 가장 오른쪽 그림을 보면 모델이 트레이닝 데이터의 정확도를 높이기 위해 결정 직선Decision Boundary을 과도하게 꼬아서 그린 모습을 볼 수 있습니다. 이럴 경우 트레이닝 데이터와 조금 형태가 다른 새로운 데이터를 예측할 때도 성능이 떨어지게 됩니다.
- 이에 반해 그림 4-3의 가장 왼쪽 그림은 언더피팅Underfitting에 빠진 상황을 보여줍니다. 언더피팅은 오버피팅의 반대 상황으로 모델의 표현력이 부족해서 트레이닝 데이터도 제대로 예측하지 못하는 상황을 말합니다. 마지막으로 그림 4-3의 중앙에 있는 그림은 오버피팅과 언더피팅에 빠지지 않고 파라미터가 적절히 학습된 경우를 보여줍니다.
- 딥러닝의 경우 모델의 표현력이 강력하기 때문에 오버피팅에 빠지기 쉽습니다. 따라서 오버피팅 문제를 완화하기 위해서 드롭아웃Dropout과 같은 다양한 Regularization* 기법을 사용합니다.

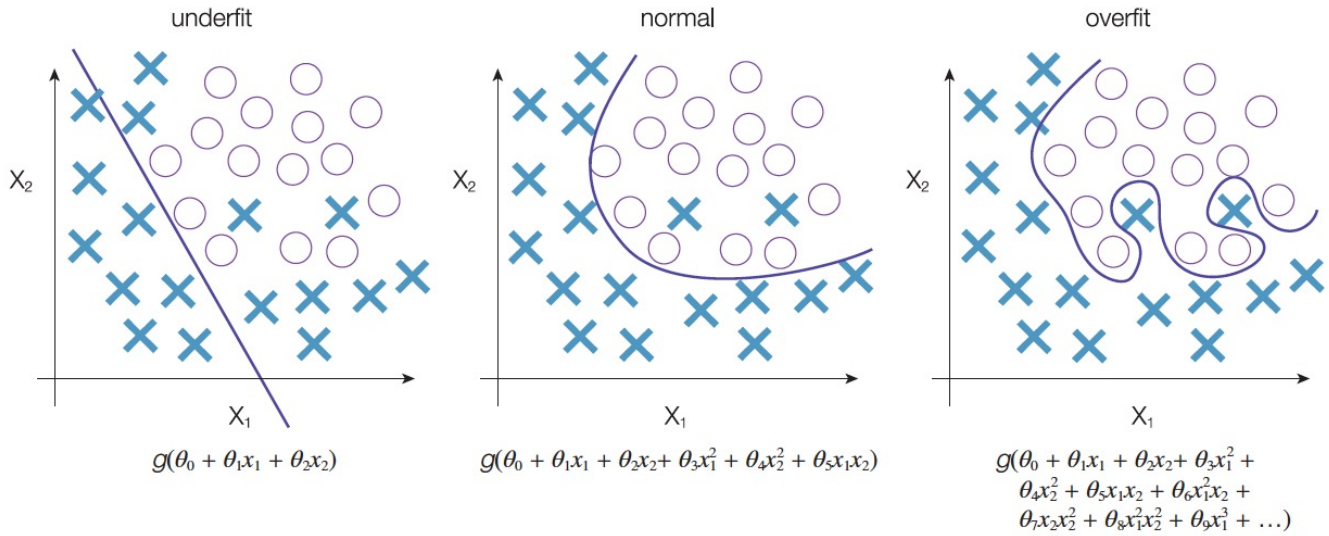


그림 4-3 | 언더피팅, 적절히 학습된 경우, 오버피팅

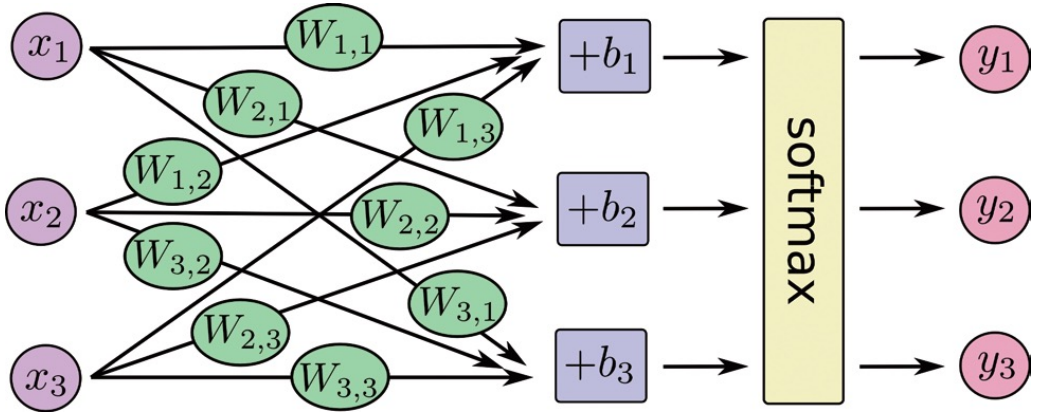
소프트맥스 회귀Softmax Regression

- 이번 장에서는 **소프트맥스 회귀Softmax Regression** 기법에 대해 알아봅시다. 소프트맥스 회귀는 n개의 레이블을 분류하기 위한 가장 기본적인 모델입니다. 모델의 아웃풋에 Softmax 함수를 적용해서 모델의 출력값이 각각의 레이블에 대한 확신의 정도를 출력하도록 만들어주는 기법입니다.

- 구체적으로 Softmax 함수는 Normalization 함수로써 출력값들의 합을 1로 만들어줍니다. Softmax 함수는 아래 수식으로 표현됩니다.

$$\text{softmax}(\text{x})_\text{i} = \frac{\exp(\text{x}_\text{i})}{\sum_\text{j} \exp(\text{x}_\text{j})}$$

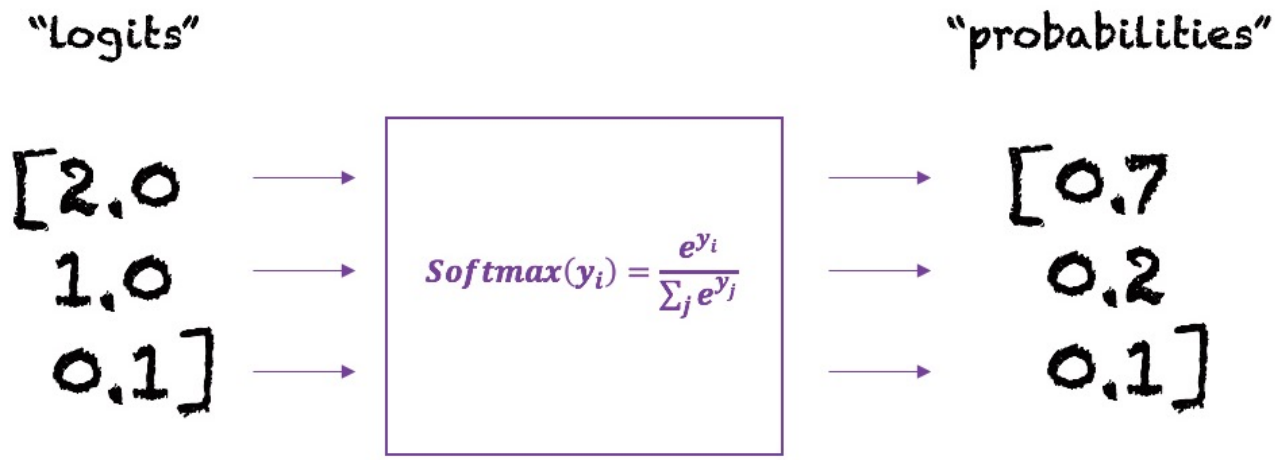
- Softmax 함수를 마지막에 씌우게 되면 모델의 출력값이 레이블에 대한 확률을 나타내게 됩니다(=출력값들의 합이 1이 되므로). 아래 그림은 소프트맥스 회귀의 구조를 보여줍니다.



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

소프트맥스 회귀Softmax Regression

- 아래 그림은 모델의 출력값 Logits에 Softmax 함수를 적용한 결과를 보여줍니다.
- 이 모델은 인풋에 대해서 각각의 레이블일 확률을 0.7(70%), 0.2(20%), 0.1(10%)로 확신하는 것으로 해석할 수 있습니다. 예를 들어, 어떤 이미지가 [개, 고양이, 말]인지 분류하는 분류기를 만든 경우, 이 모델은 지금 인풋으로 들어온 이미지가 70% 확률로 개, 20% 확률로 고양이, 10% 확률로 말이라고 판단한 것으로 해석할 수 있습니다.



크로스 엔트로피Cross-Entropy 손실 함수

- 소프트맥스 회귀Softmax Regression를 비롯한 분류 문제에는 크로스 엔트로피Cross-Entropy 손실 함수를 많이 사용합니다. 크로스 엔트로피 손실 함수도 평균제곱오차(MSE)와 같이 모델의 예측값이 참값과 비슷하면 작은 값, 참값과 다르면 큰 값을 갖는 형태의 함수로 아래와 같은 수식으로 나타낼수 있습니다.

$$H_{y'}(y) = -\sum_{i} y'_i \log(y_i)$$

- 위 수식에서 y'는 참값, y는 모델의 예측값을 나타냅니다. 일반적으로 분류 문제에 대해서는 MSE보다 크로스 엔트로피 함수를 사용하는 것이 학습이 더 잘되는 것으로 알려져 있습니다.
- 따라서 대부분의 텐서플로 코드들에서 크로스 엔트로피 손실 함수를 사용합니다. 본 강의에서도 대부분의 코드 구현에서 크로스 엔트로피 함수를 손실 함수로 사용합니다.

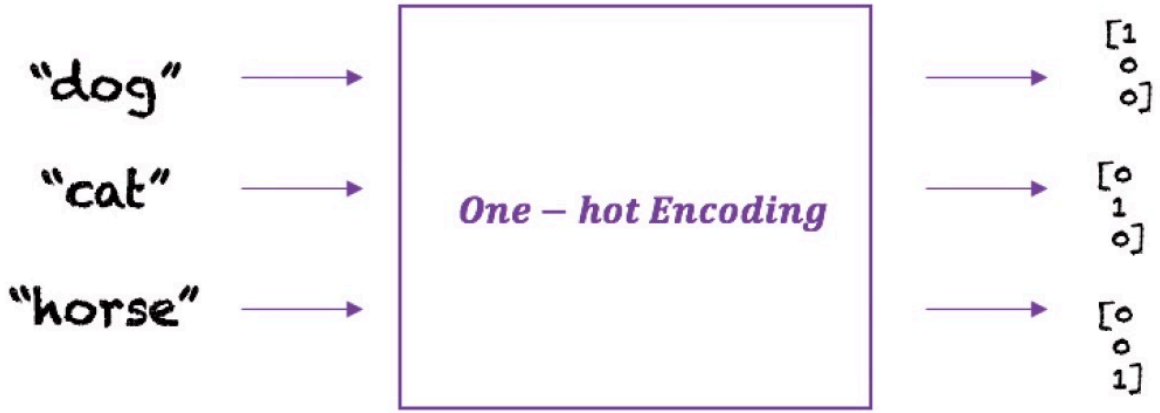
MNIST 데이터셋

- 머신러닝 모델을 학습시키기 위해서는 데이터가 필요합니다. 하지만 데이터를 학습에 적합한 형태로 정제하는 것은 많은 시간과 노력이 필요한 일입니다. 따라서 많은 연구자가 학습하기 쉬운 형태로 데이터들을 미리 정제해서 웹상에 공개해놓았습니다.
- 그중에서도 MNIST 데이터셋은 머신러닝을 공부하는 사람들이 가장 먼저 접하게 되는 데이터셋으로 머신러닝 분야의 “Hello World”라고 볼 수 있습니다.
- 구체적으로 MNIST 데이터는 60,000장의 트레이닝 데이터와 10,000장의 테스트 데이터로 이루어진 데이터셋으로 0~9사이의 28×28 크기의 필기체 이미지로 구성되어있습니다. 아래 그림은 MNIST 데이터의 예제를 보여줍니다.

One-hot Encoding

- **One-hot Encoding**은 범주형 값Categorical Value을 이진화된 값Binary Value으로 바꿔서 표현하는 것을 의미합니다. 범주형 값은 예를 들어 “개”, “고양이”, “말”이라는 3개의 범주형 데이터가 있을 때 이를 [“개”=1, “고양이”=2, “말”=3]이라고 단순하게 Integer Encoding으로 변환하여 표현하는 것입니다.

- 이에 반해 One-hot Encoding을 사용하면 범주형 데이터를 “개”=[1 0 0], “고양이”=[0 1 0], “말”=[0 0 1] 형태로 해당 레이블을 나타내는 인덱스만 1의 값을 가지고 나머지 부분은 0의 값을 가진 Binary Value로 표현합니다.



- 단순한 Integer Encoding의 문제점은 머신러닝 알고리즘이 정수 값으로부터 잘못된 경향성을 학습하게 될 수도 있다는 점입니다. 예를 들어, 위의 예시의 경우 Integer Encoding을 사용할 경우 머신러닝 알고리즘이 [“개”(=1)와 “말”(=3)의 평균(1+3/2=2)은 “고양이”(=2)이다.] 라는 지식을 학습할 수도 있는데, 이는 명백히 잘못된 학습입니다. 따라서 머신 러닝 알고리즘을 구현할 때 타겟 데이터를 One-hot Encoding 형태로 표현하는 것이 일반적입니다.

TensorFlow 2.0을 이용한 Softmax Regression 구현

- Softmax Regression 알고리즘을 TensorFlow 2.0 코드로 구현해봅시다.
- https://github.com/solaris33/deep-learning-tensorflow-book-code/blob/master/Ch04-Machine_Learning_Basic/mnist_classification_using_softmax_regression_v2_keras.py

Chapter 3 - 텐서플로우 기초와 텐서보드

- 텐서플로우 기초 – 그래프 생성과 그래프 실행 (Code) (TF v2 Code)
- 플레이스홀더 (Code) (TF v2 Code)
- 선형 회귀(Linear Regression) 알고리즘 (Code) (TF v2 Code)
- 선형 회귀(Linear Regression) 알고리즘 + 텐서보드(TensorBoard) (Code) (TF v2 Code) (TF v2 Keras Code)

Chapter 4 - 머신러닝 기초 이론들

- 소프트맥스 회귀(Softmax Regression)를 이용한 MNIST 숫자분류기 (Code) (TF v2 Code) (TF v2 Keras Code)
- tf.nn.sparse_softmax_cross_entropy_with_logits API를 사용한 소프트맥스 회귀(Softmax Regression)를 이용한 MNIST 숫자분류기 (Code) (TF v2 Code)

Thank you!