

 x 

The main title of the slide, featuring the CODINGO logo on the left, a large black "x" in the center, and the POSCO logo on the right. The POSCO logo is in a blue, lowercase, sans-serif font.

 스마트 팩토리 3기

The subtitle of the slide, featuring the "K-Digital Training" logo on the left and the Korean text "스마트 팩토리 3기" (Smart Factory 3rd Generation) on the right. The "K-Digital Training" logo has "K" in green, "Digital" in blue, and "Training" in purple.

머신러닝 과정

머신 러닝의 핵심요소

1. 데이터
2. 모델
3. 학습 알고리즘
4. 평가 및 예측

머신 러닝의 핵심요소

1. 데이터

- 머신러닝은 데이터를 기반으로 모델을 학습하고 예측하는 방법.
- 데이터는 입력 데이터와 해당하는 출력 또는 정답 데이터로 구성.
- 풍부하고 다양한 데이터를 수집하고 준비하는 것이 머신러닝의 핵심 요소 중 하나.

2. 모델

- 모델은 데이터의 패턴과 관계를 학습하기 위해 사용되는 수학적인 알고리즘의 집합.
- 모델은 입력 데이터를 특징(feature)으로 표현하고 이를 기반으로 예측을 수행.
- 모델은 다양한 종류가 있으며, 각각의 모델은 특정한 문제 유형에 적합한 구조와 알고리즘을 가지고 있다.

3. 학습 알고리즘

- 학습 알고리즘은 데이터를 사용하여 모델의 파라미터를 조정하고 최적화하는 방법을 정의.
- 학습 알고리즘은 주어진 데이터에 대한 오차를 최소화하거나 성능 지표를 최대화하는 방향으로 모델을 향상시킴.
- 일반적으로는 경사 하강법 (gradient descent)과 같은 최적화 알고리즘이 사용.

4. 평가 및 예측

- 학습된 모델은 평가 데이터를 사용하여 성능을 평가하고 새로운 입력 데이터에 대한 예측을 수행.
- 모델의 성능은 정확도, 정밀도, 재현율 등의 지표를 사용하여 측정.
- 이를 통해 모델의 일반화 능력을 평가하고 필요에 따라 모델을 조정하거나 다른 알고리즘을 시도할 수 있음.

머신 러닝의 과정

1. 문제정의
2. 모델
3. 학습 알고리즘
4. 평가 및 예측
5. 모델 평가
6. 예측
7. 모델 개선

머신 러닝의 과정

1. 문제정의

- 머신러닝을 적용할 문제를 정의.
- 예를 들어, 주어진 데이터로부터 사진에 나타난 객체를 분류하는 문제를 해결하고자 할 수 있다.

2. 모델

- 문제 해결을 위해 필요한 데이터를 수집.
- 데이터는 입력 데이터와 해당하는 출력 데이터(레이블 또는 클래스)로 구성.
- 수집된 데이터는 전처리 과정을 거쳐 결측치 처리, 정규화, 범주형 데이터의 인코딩 등의 작업을 수행하여 데이터를 준비.

3. 학습 알고리즘

- 전체 데이터를 학습 데이터와 테스트 데이터로 분할.
- 학습 데이터는 모델을 학습시키는 데 사용되고, 테스트 데이터는 학습된 모델의 성능을 평가하는 데 사용.
- 일반적으로 70-80%를 학습 데이터로, 나머지를 테스트 데이터로 사용하는 것이 일반적.

4. 평가 및 예측

- 문제에 적합한 모델을 선택하고 학습을 진행.
- 선택한 모델은 데이터를 통해 학습되어 가중치와 편향이 조정.
- 이 과정에서 학습 알고리즘과 최적화 기법을 사용하여 모델의 파라미터를 조정.

5. 모델 평가

- 학습된 모델을 테스트 데이터에 적용하여 성능을 평가.
- 이를 통해 모델의 정확도, 정밀도, 재현율, F1 점수 등의 지표를 계산.
- 평가 결과를 통해 모델의 성능을 분석하고 필요에 따라 모델을 조정하거나 다른 모델을 시도.
하이퍼 파라미터

6. 예측

- 학습된 모델이 새로운 입력 데이터에 대해 예측을 수행.
- 입력 데이터를 모델에 주입하여 출력을 얻고, 예측된 결과를 활용.

7. 모델 개선

- 모델의 성능이 만족스럽지 않은 경우, 추가적인 조치를 취하여 모델을 개선.
- 이는 데이터의 품질 향상, 모델 구조 변경, 하이퍼 파라미터 조정 등을 포함.

데이터 전처리

- 데이터 사이언티스트들이 소모하는 전체시간의 약 60%를 차지
- 오류 데이터 제거, 이상값 제거
- 일관성 있는 데이터 형태로 변환
- 정규화(normalization)
- 데이터 축소

Training set과 Test set



- 트레이닝 셋
 - 모델을 학습하는 데 사용되는 데이터
 - 해당 데이터를 이용하여 모델의 파라미터를 조정하고 입출력 관계를 학습
- 테스트 셋 일반화 성능
 - 모델의 일반화 성능을 평가하는 데 사용
 - 학습이 완료된 모델에 대해서 테스트셋의 데이터를 사용하여 예측하고 실제 출력과 비교하여 모델의 성능을 평가
 - 해당 데이터는 모델을 개선하는 데에는 사용되지 않음
- 검증 셋
 - 트레이닝 중간에 모델의 성능평가를 하기 위해서 사용

Training set과 Test set



- 구분하여 사용하는 이유

1. 일반화 성능 평가

- 처음 보는 데이터에 대해 얼마나 잘 동작하는지 알기 위함

2. 오버피팅 검증

- 모델이 과적합되지 않았는지 검증
- 테스트셋의 성능이 트레이닝 셋보다 현저히 낮다면 과적합된 상태
일 가능성 높음

Training set과 Test set



- 구분하여 사용하는 이유

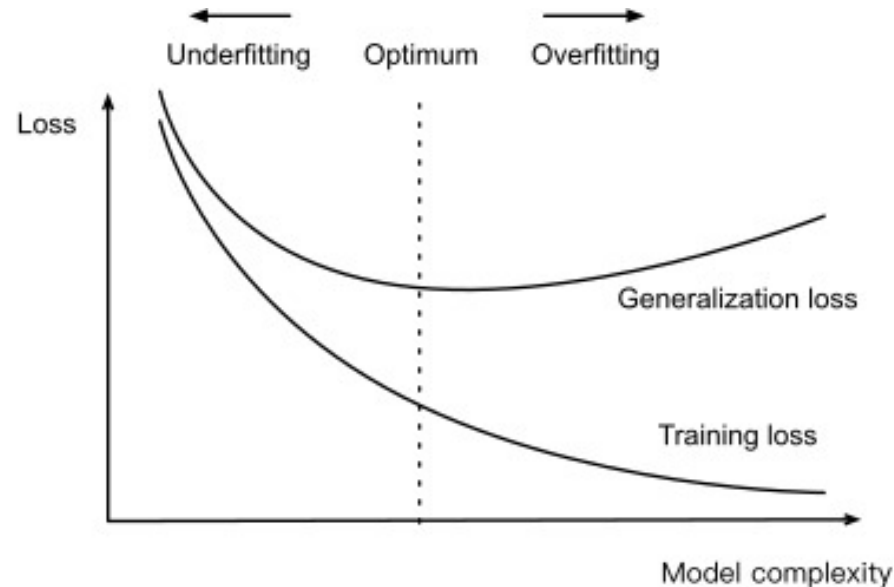
3. 하이퍼 파라미터 튜닝

1. 검증셋을 두어 하이퍼 파라미터 튜닝
2. 트레이닝셋으로 학습하고, 검증셋으로 성능을 평가하여 최적의 파라미터 선택
3. 이후 테스트 셋을 사용하여 최종 평가

- 트레이닝셋, 검증셋, 테스트셋 각각은 독립적이어야함

Overfitting

- 학습 데이터에 대해 모델이 지나치게 적합되어, 새로운 데이터나 테스트 데이터에서 성능이 저하되는 현상
- 트레이닝 데이터는 정확도가 높는데 테스트 데이터 정확도는 낮은 경우의 심가능



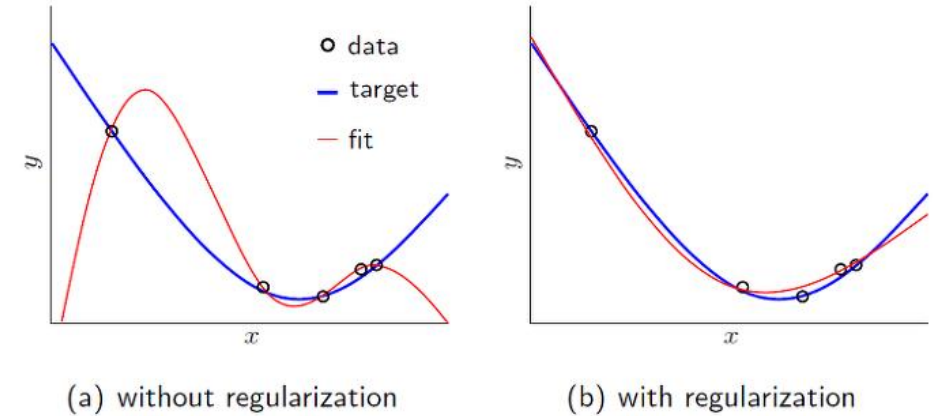
Overfitting

- 원인
 - 매개변수의 수
 - 매개변수의 수가 많을 수록 모델의 표현력이 증가하지만 과도하게 적합되기 쉬워짐
 - 학습 데이터의 부족
 - 학습 데이터의 양이 적을 경우, 주어진 데이터에 과도하게 적합 될 수 있음
 - 모델 복잡성
 - 딥러닝 모델의 구조가 복잡할수록, 모델은 학습 데이터에 더 적합 하려고 할 수 있음
 - 학습 시간과 반복 횟수
 - 학습을 오래 진행하거나 반복 횟수를 많이 설정하면, 모델이 학습 데이터에 과적합 될 수 있음.
조기 종료(Early Stopping) 등의 방법 사용

Overfitting

- 해결

- Training data를 늘린다
- Regularization(정규화)
 - W 가 너무 큰 값들을 가지지 않도록 수정
 - 함수의 복잡도를 낮추기 위한 방법
 - Cost function 뿐만이 아니라 weight 도 작아지는 쪽으로 학습(weight decay)

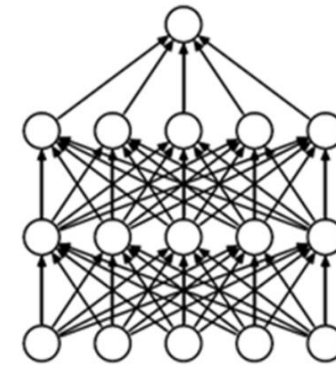


Overfitting

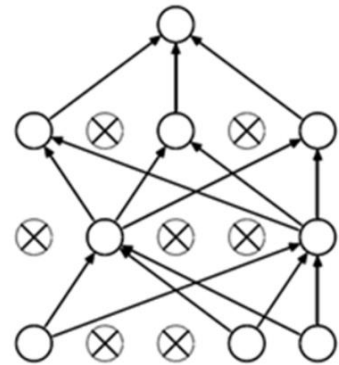
- 해결

- Drop-Out

- 신경망의 크기가 커질 경우 사용할 수 있는 방법
 - Random하게 일부 뉴런을 disable 하여 layer에 포함된 weight 중에서 일부만 참여시키는 방법
 - 평균 효과를 얻을 수 있어서 regularization과 비슷한 효과
 - 특정 뉴런의 w 나 b 가 너무 커졌을 때 해당 뉴런에 너무 많은 영향을 받지 않게 하는 효과(뉴런 동조화 방지)



(a) Standard Neural Net



(b) After applying dropout.

Weight normalization

- L1, L2 정규화

- Norm

- 벡터의 크기를 측정하는 방법, 두 벡터 사이의 거리를 측정하는 방법

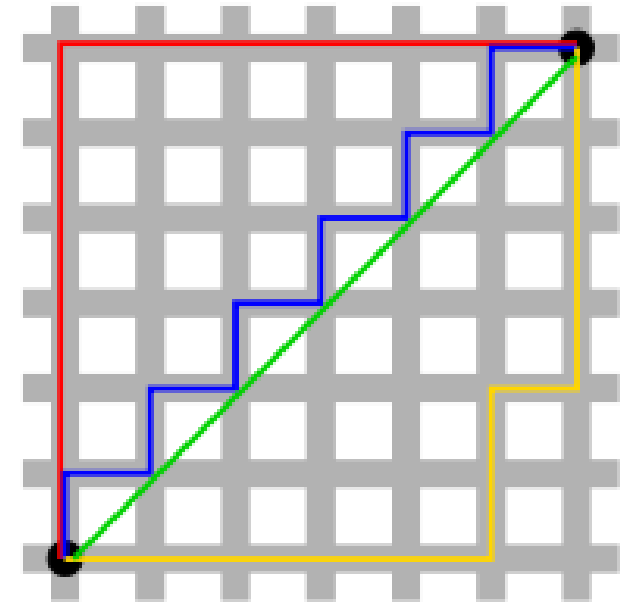
- Lp Norm =
$$\|\mathbf{x}\|_p = \sqrt[p]{|x_1|^p + |x_2|^p + \dots + |x_n|^p} = \left(\sum_{k=1}^n |x_k|^p \right)^{1/p}$$

- L1 Norm : 두 벡터 각 원소들 차이의 절대값의 합

- L2 Norm: 두 벡터 사이의 직선 거리

Weight normalization

- L1, L2 정규화
 - L1 Norm과 L2 Norm의 직관적 차이
 - 오른쪽 그림은 두개의 벡터를 잇는 선을 나타냄
 - 초록색선이 L2 Norm(Euclidean distance)
 - 나머지 빨강,파랑,노랑 선은 모두 L1 Norm(Taxicab geometry)



Weight normalization

- L1 Loss : 타겟값과 예측값의 차의 절대 값

$$L = \sum_{i=1}^n |y_i - f(x_i)|$$

- L2 Loss : 타겟값과 예측값의 차의 제곱

$$L = \sum_{i=1}^n (y_i - f(x_i))^2$$

Weight normalization

- L1 Regularization
 - Cost function 에 가중치의 **절대값**을 더해줌
 - L1 정규화를 사용하는 모델을 **Lasso** 모델이라고 함
 - **불필요한 Weight를 0으로 만들어버려서 Feature selection의 효과**
 - weight 의 크기에 상관없이(편미분의 결과) 부호에 따라 일정한 상수값을 빼거나 더해지게 된다
 - 모델은 cost를 줄이는 방향으로 학습하기 때문에 가중치들이 작아지고 일부는 0으로 수렴
 - 람다 : 정규화 항의 가중치, 해당 값이 높을수록 가중치가 0으로 수렴

$$Cost = \frac{1}{n} \sum_{i=1}^n \{L(y_i, \hat{y}_i) + \frac{\lambda}{2} |w|\}$$

$L(y_i, \hat{y}_i)$: 기존의 Cost function

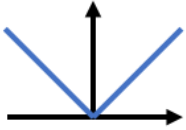
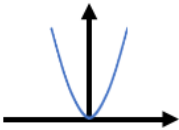
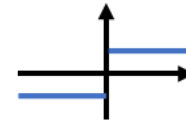

Weight normalization

- L2 Regularization
 - Cost function 에 가중치의 제곱을 포함하여 더함
 - 가중치가 너무 크지 않은 방향으로 학습되게 함. (Weight decay)
 - cost 뿐만 아니라 가중치 또한 줄어드는 방향으로 학습
 - Weight 가 클수록 더 빠르게 감소
 - L2 정규화를 사용하는 모델을 Ridge 모델이라고 부름
 - 불필요한 Feature를 0에 가깝게 만드는 특성

$$Cost = \frac{1}{n} \sum_{i=1}^n \{L(y_i, \hat{y}_i) + \frac{\lambda}{2} |w|^2\}$$

- L1 정규화, L2 정규화를 모두 사용하는 것을 Elastic Net이라고 함

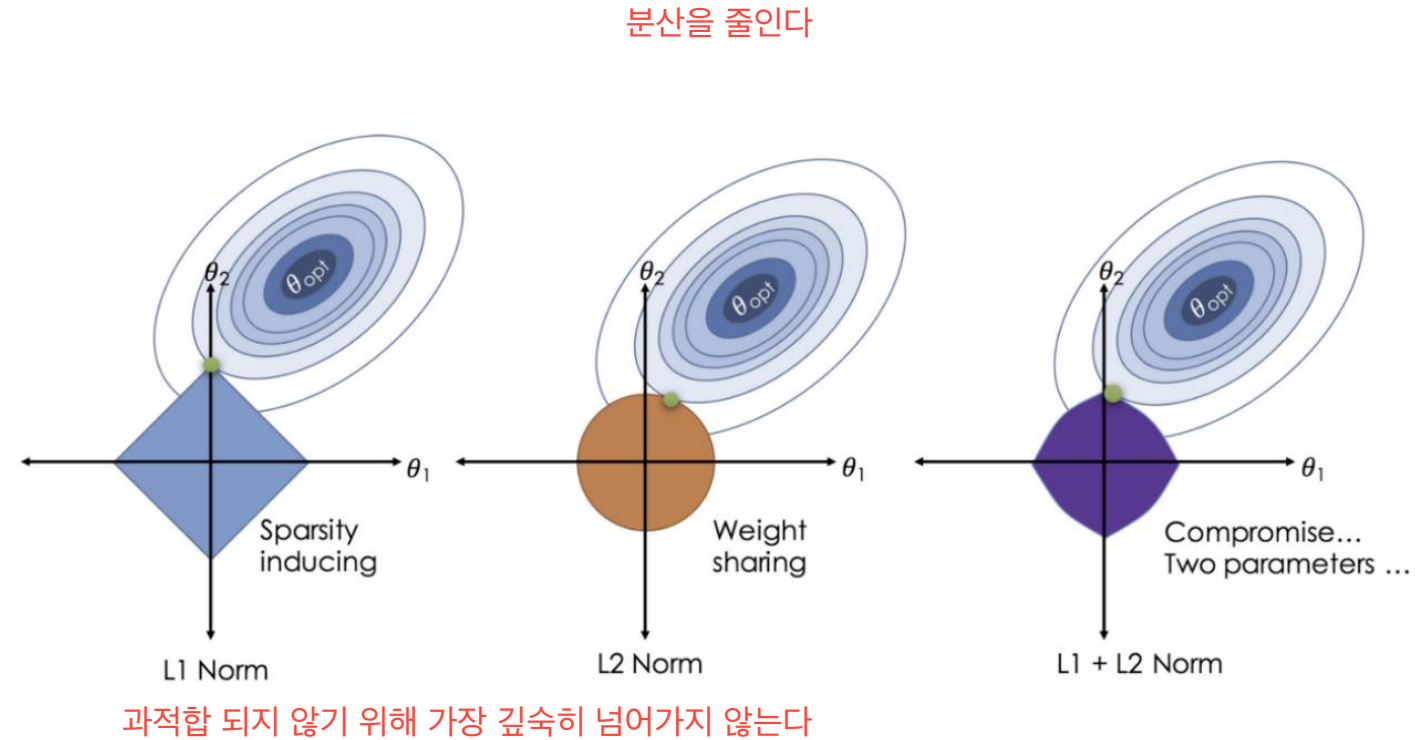
Weight normalization

구분	Lasso		Ridge	
수식	$\ w\ _1$		$\frac{1}{2} \ w\ ^2$	
미분	$\begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases}$		w	
특성	<ul style="list-style-type: none"> 가중치 값을 정확하게 0으로 만들 중요한 특징을 '선택'하는 효과 모델에 Sparsity를 가함. 		<ul style="list-style-type: none"> 큰 가중치의 값을 작게 만들 모델 전반적인 복잡도를 감소시키는 효과 가중치의 값이 0이 되게 하지는 못함 	

출처: <https://gaussian37.github.io/dl-concept-regularization/>

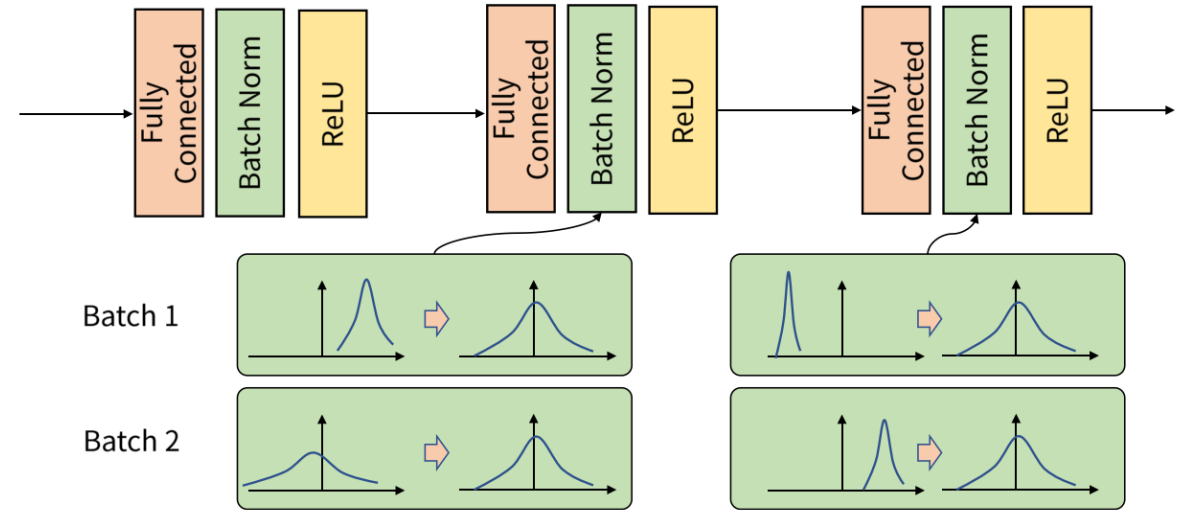
Weight normalization

- 실제 최적값에 대한 bias에 손해를 보더라도 variance를 낮춰 Overfitting 발생을 낮추는 것
- loss function의 최적값은 규제 영역 내에서 Global Optimum과 제일 가까운 지점

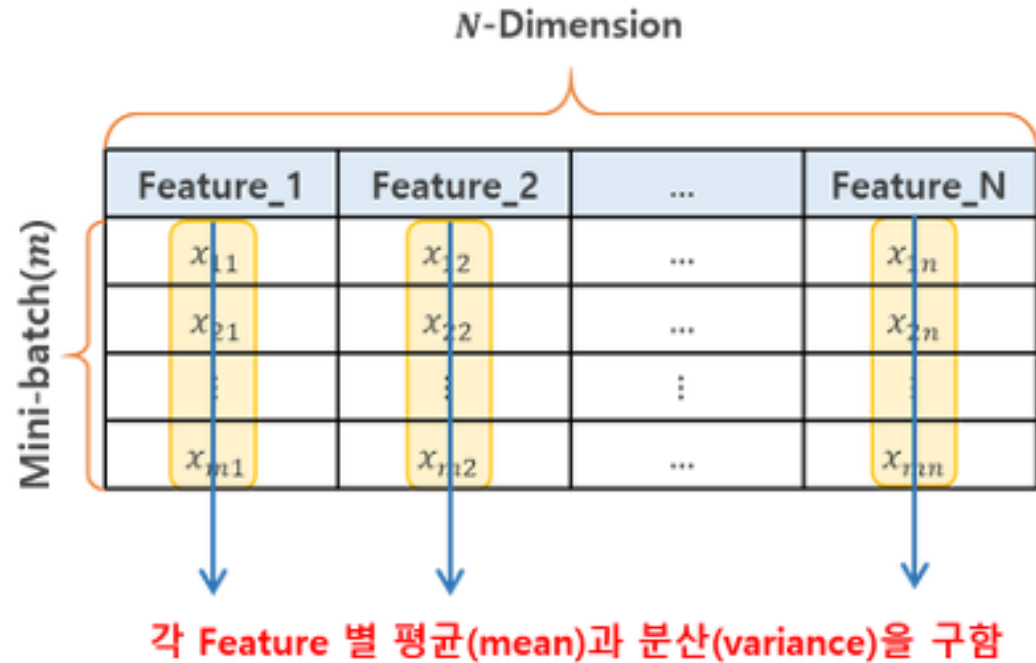


BN(batch normalization)

- 배치 정규화
- 학습 과정에서 각 배치 단위 별로 데이터가 다양한 분포를 가지더라도 **각 배치별로 평균과 분산을 이용해 정규화 하는 것**
- 평균은 0, 표준 편차는 1로 데이터의 분포를 조정
- 한계 : batch 의 크기에 영향을 받음, batch 의 크기가 너무 작거나 너무 커지면 잘 동작하지 않음



BN(batch normalization)



Normalize



$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

BN(batch normalization)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

- ϵ : 분모가 0이 되는 것을 막기 위한 작은 숫자(10^{-5})
- \hat{x}_i : 평균이 0, 분산이 1로 정규화된 입력 데이터

BN(batch normalization)

- Scale(γ)과 Shift(β)를 해주는 이유
 - 정규화의 결과는 대부분 0에 가까운 값이 된다
 - 이러한 입력이 시그모이드 활성화 함수의 입력값으로 들어가게 되면, 비선형 함수인 sigmoid가 선형 구간에 빠지게 된다
 - 이러한 문제를 해결하기 위해서 정규화된 데이터에 scale 과 shift를 적용
 - γ 와 β 는 초기값 1, 0으로 시작해서, 역전파에 의해 학습되고 조정된다

BN(batch normalization)

- 테스트 단계에서의 BN
 - 테스트 단계나 추론 단계에서는 평균과 표준편차를 계산할 미니배치가 없기 때문에 전체 Training set의 평균과 표준편차 사용
 - 각 n개의 미니배치에 대한 평균과 표준 편차를 이용하여 전체 Training set의 평균과 표준편차를 대신한다

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n \mu_B^{(i)}$$
$$\hat{\sigma} = \frac{1}{n} \sum_{i=1}^n \sigma_B^{(i)}$$

BN(batch normalization)

- 테스트 단계에서의 BN
 - 위의 방법 대신, 모델 학습 단계에서 지수 감소(exponential decay) 이동 평균법(moving average)를 사용하여 평균과 표준편차를 계산할 수 있다.

$$\begin{aligned}\hat{\mu} &\leftarrow \alpha \hat{\mu} + (1 - \alpha) \mu_B^{(i)} && // \text{moving mean} \\ \hat{\sigma} &\leftarrow \alpha \hat{\sigma} + (1 - \alpha) \sigma_B^{(i)} && // \text{moving stddev}\end{aligned}$$

- α : 모멘텀값, 1에 가까운 0.9, 0.99, 0.999 로 설정

BN(batch normalization)

- 장점

- tanh나 sigmoid 같은 활성화 함수의 그래디언트 소실 문제 감소
- 가중치 초기화에 덜 민감
- 학습률을 크게 잡아도 잘 수렴한다.
- 오버피팅을 억제

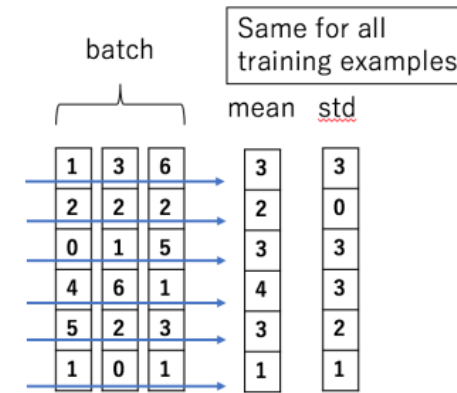
- 단점

- 미니배치 크기에 의존(배치사이즈가 작으면 동작하지 않음)
- RNN model에 적용하기 힘들다(Layer Normalization 사용)
 - 각 시점마다 다른 데이터가 연속적으로 나오기 때문에

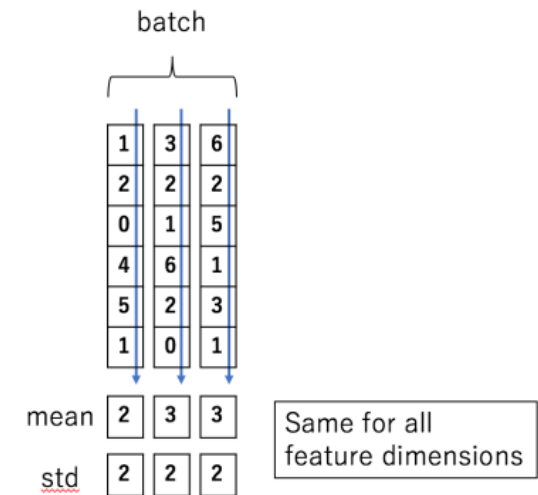
LN(layer normalization)

- 레이어 정규화
- BN은 batch에 있는 모든 sample들에 대해서 각 feature의 평균과 분산을 구하여 정규화
- LN은 각 sample의 feature들에 대해서 평균과 분산을 구하여 정규화
- BN은 batch size와 관련이 깊지만 LN은 batch size와는 전혀 상관이 없다

Batch Normalization



Layer Normalization



출처: <https://electronicsmarket.org/best-4x10-car-speakers/>

LN(layer normalization)

- 장점
 - 작은 batch size에서도 이용 가능
 - RNN모델에서 더 효과적
 - 일반화 성능 향상 가능
- 단점
 - 추가 계산 및 메모리 오버헤드가 발생할 수 있다
 - 피드 포워드 네트워크 모델에서는 BN만큼 잘 동작하지 않을 수 있다.
 - 학습률, 가중치 초기화에 민감하다.

- 주요 특징

1. 계층적인 구조

- 여러 개의 은닉층으로 구성되어 있음.
- 각 은닉층은 이전 층의 출력을 입력으로 받아 새로운 특징을 학습.
- 계층적인 구조를 통해서 복잡한 패턴과 추상적인 특징 학습

2. 역전파 알고리즘(backpropagation)

- 역전파 알고리즘을 사용하여 모델의 가중치와 편향을 조정
- 출력과 실제 값 사이의 오차를 역으로 전파하여 각층의 가중치를 업데이트하고, 이를 반복하여 모델을 학습시킴

딥러닝 모델

- 주요 특징

- 3. 대량의 데이터와 연산

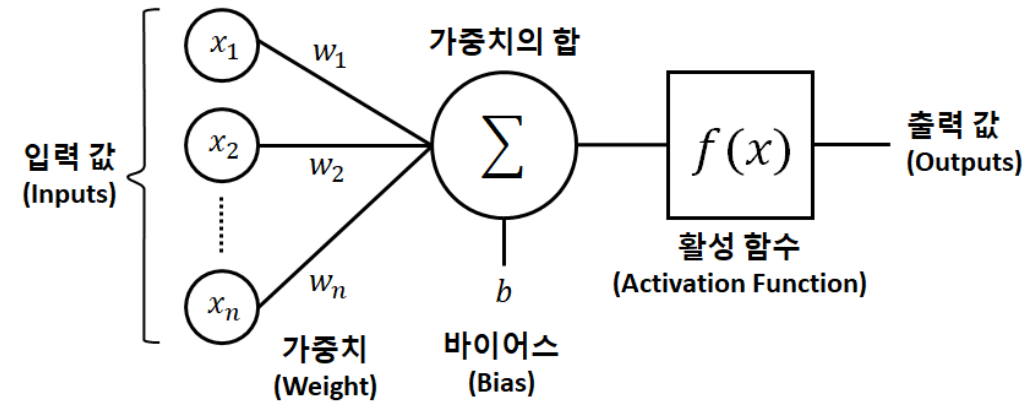
- 대량의 데이터와 많은 연산이 필요

- 4. 종단간 학습

- 입력데이터와 출력 데이터만을 사용하여 모델을 학습시킬 수 있음(end-to-end)

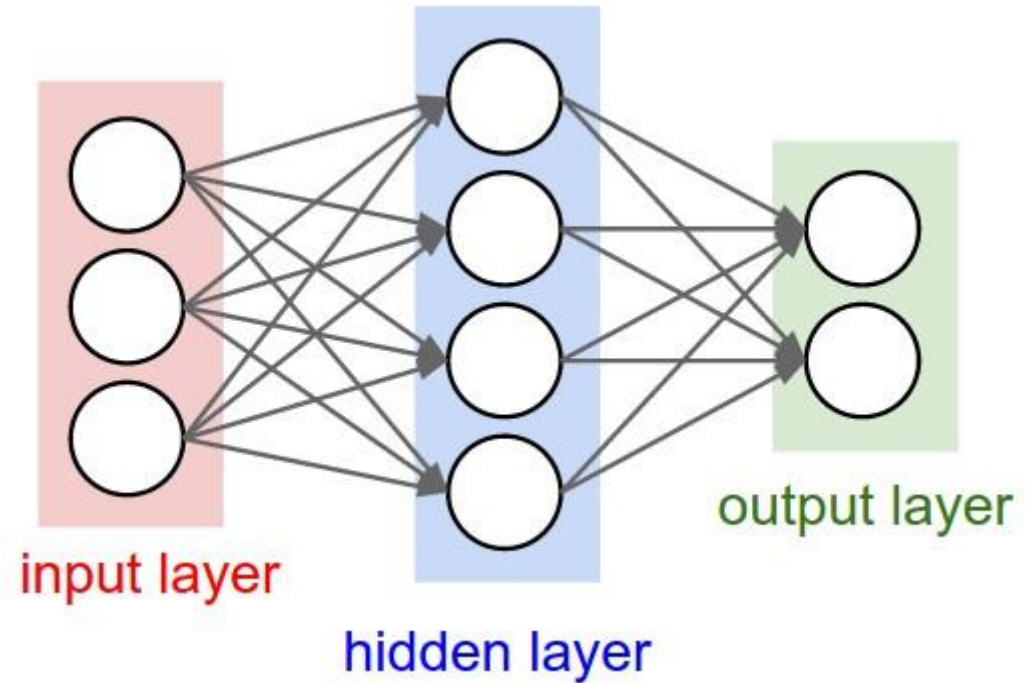
Perceptron

- Perception 과 Neuron의 합성어(인공뉴런)
- 단일 계산층
- 입력 값에 각 weight 값을 곱해서 더해준다
- 마지막에 활성 함수를 이용해 결과값을 도출한다
- 활성함수를 쓰는 이유 –
 1. 선형분류기를 비선형 분류기로 바꿔준다
 2. 역전파를 위해서 미분가능한 모양(비선형)으로 바꿔줘야 한다



MLP(Multi Layer Perceptron)

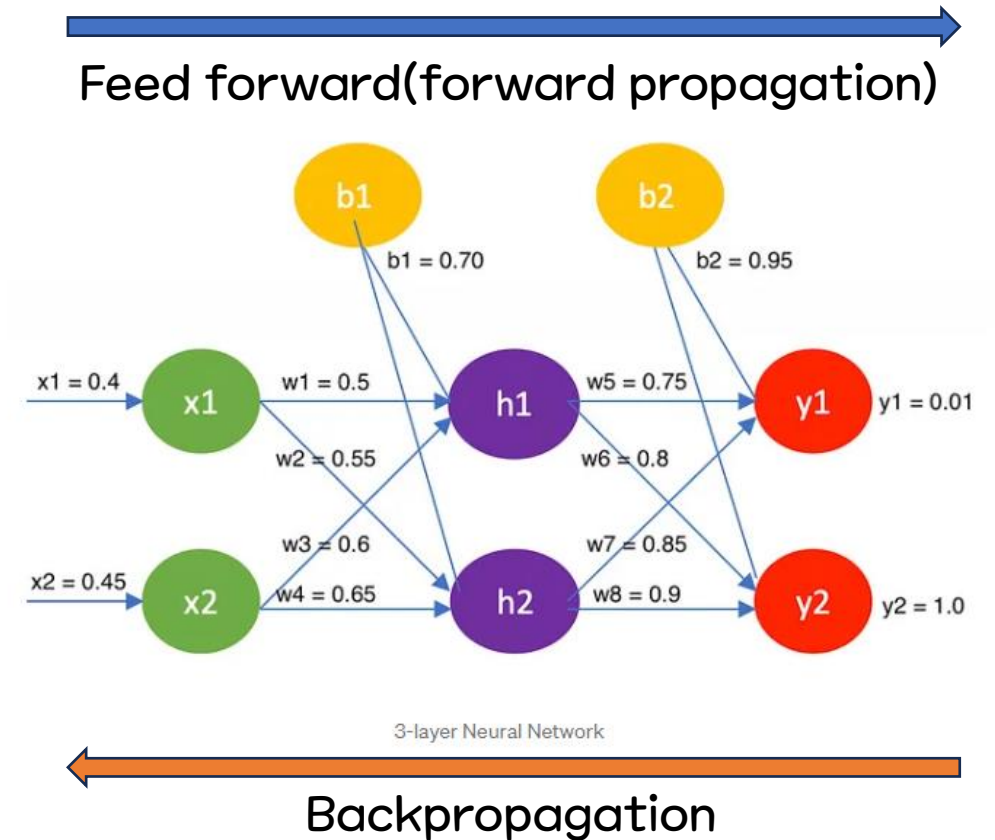
- 다층 퍼셉트론
- 입력층과 출력층 사이에 각각 전체 결합하는 은닉층을 넣은 뉴럴 네트워크
- 인접한 두 층의 뉴런간에는 완전 연결 (fully connected)
- 은닉계층은 하나 이상으로 구성



출처: <https://buomsoo-kim.github.io/keras/2018/04/21/Easy-deep-learning-with-Keras-2.md/>

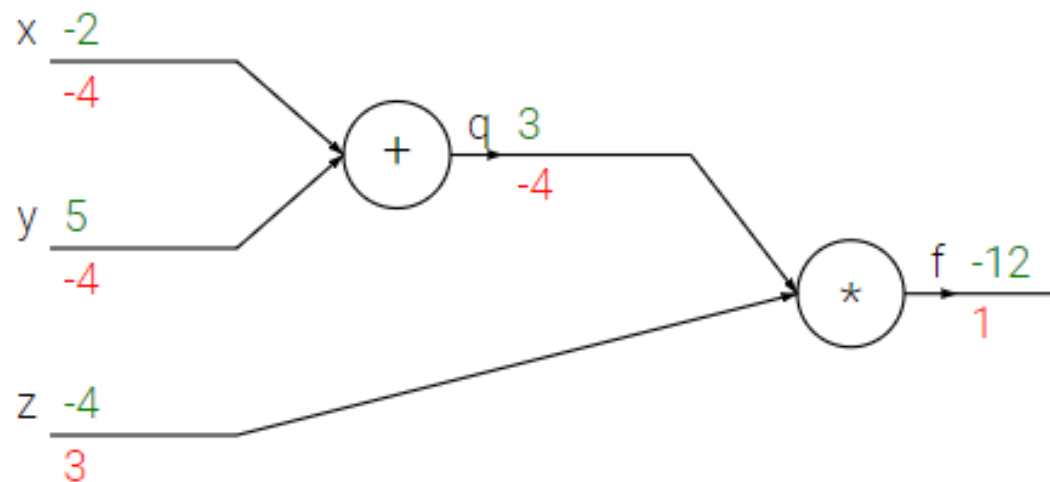
역전파(backpropagation)

- 결과값과 오차가 계산 되었을 때 그 w 와 b 를 어떻게 업데이트 하느냐에 대한 알고리즘
- 발생한 오차에 대해서 각각 w 에 대한 기여도를 계산하여 w 를 업데이트
- 이를 input layer까지 반복한다.



출처: <https://medium.com/analytics-vidhya/back-propagation-algorithm-170b5f16790a>

역전파 예제



- 녹색은 forward pass
- 빨간색은 backward pass에 의한 gradient

역전파 예제

- $f(x, y, z) = (x + y)z$

- 우리가 원하는 값은 최종 output에 대한 각 input의 gradient인

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

- $q = x + y, \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1,$

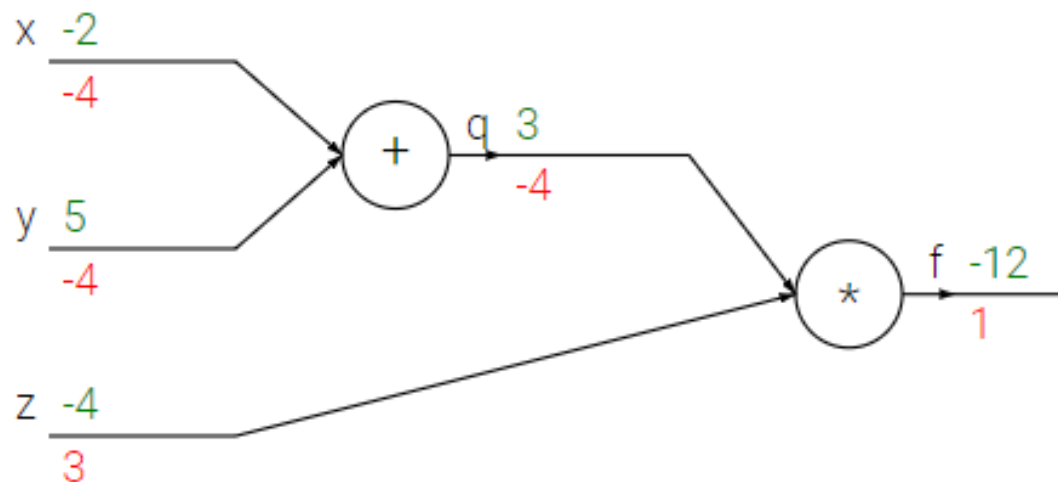
- $f = qz, \quad \frac{\partial f}{\partial q} = z = -4, \frac{\partial f}{\partial z} = q = 3,$

역전파 예제

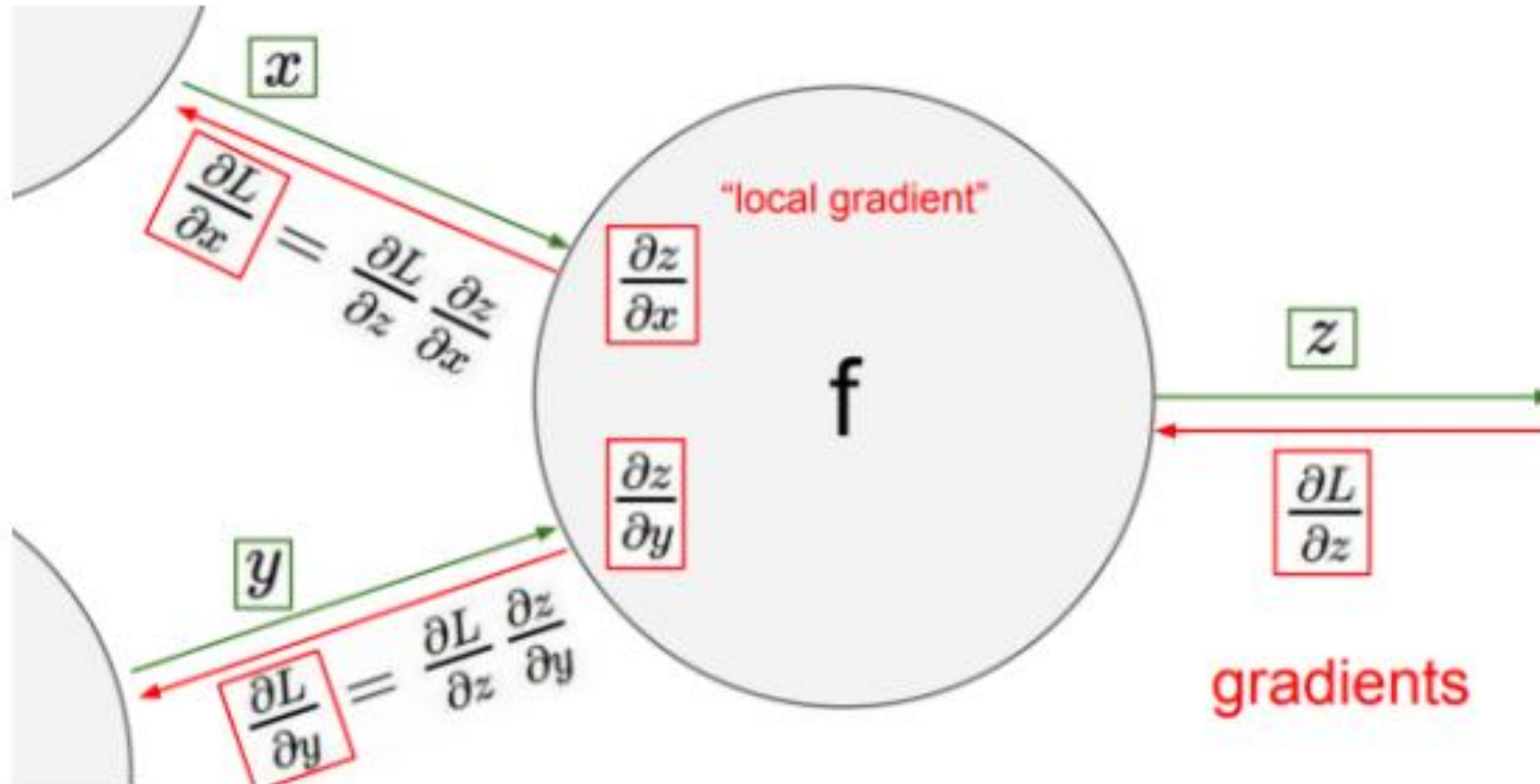
- $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} = z * 1 = -4$ (체인 룰에 의해)

- $\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} = z * 1 = -4$ (체인 룰에 의해)

- $\frac{\partial f}{\partial q} = -4, \frac{\partial f}{\partial x} = -4, \frac{\partial f}{\partial y} = -4$

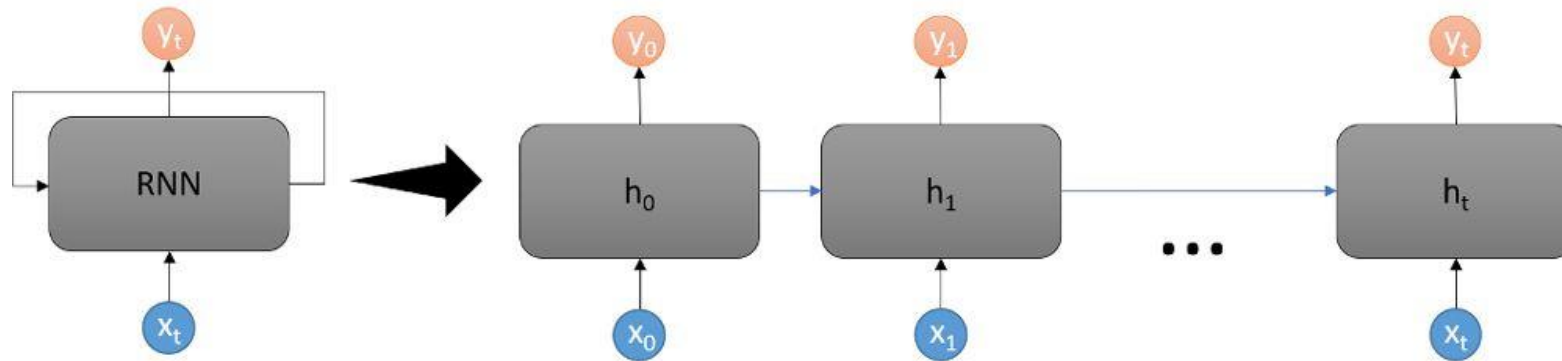


역전파 예제



RNN(Recurrent Neural Network)

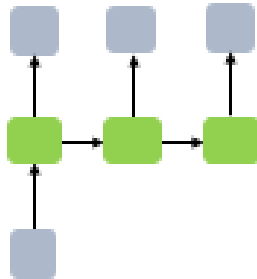
- 순환 신경망
- 입력과 출력을 시퀀스 단위로 처리
- 시퀀스들을 처리하기 위해 고안된 모델을 시퀀스 모델이라고 하며 RNN은 가장 기본적인 시퀀스 모델
- 활성화 함수의 결과값을 출력으로도 보내면서 다음 노드의 입력으로도 보내는 특징



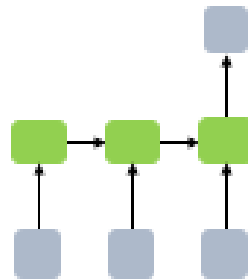
출처: <https://kr.mathworks.com/discovery/rnn.html>

RNN(Recurrent Neural Network)

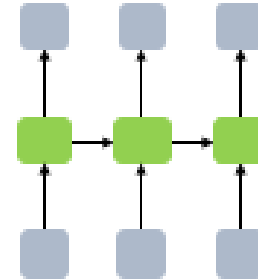
- 입력과 출력의 길이를 다르게 설계 할 수 있기 때문에 다양한 용도로 사용 가능
 - 일대다 : image to text
 - 다대일 : 문장 감정 분류, 스팸 메일 분류
 - 다대다 : 챗봇, 번역기



일 대 다(one-to-many)



다 대 일(many-to-one)



다 대 다(many-to-many)

출처:<https://kr.mathworks.com/discovery/rnn.html>

RNN(Recurrent Neural Network)

- 자연어 처리, 신호 분류, 비디오 분석등에 사용
- 한계 : **길어질수록 앞의 정보가 뒤로 충분히 전달되지 못함**(장기 의존성 문제)