Foundation of Machine Learning 15주차

정재헌, 우지수 / 2023.07.06

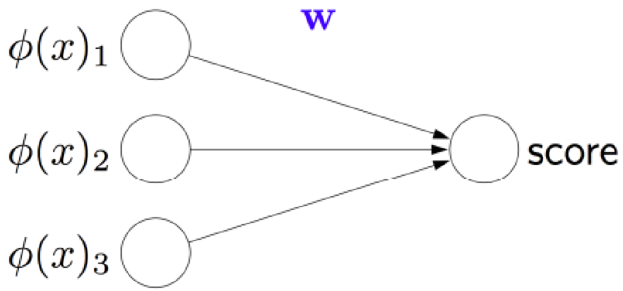
Computational Data Science LAB

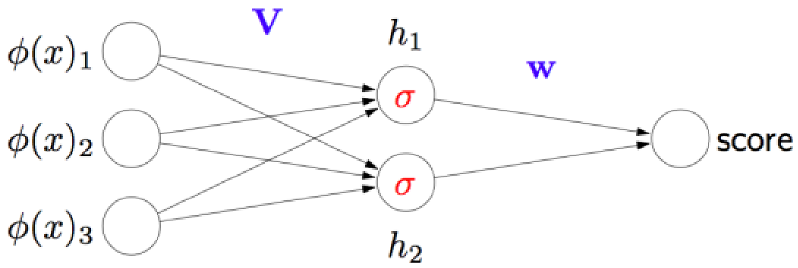
CONTENTS

01 | Neural Network

introduction

$$h_i = \sigma\left(w_i^T \phi(x)\right)$$





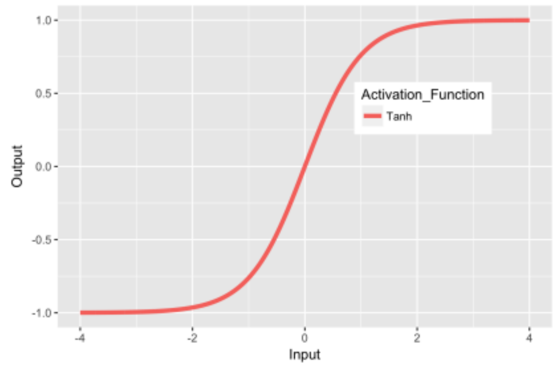
- Linear Prediction Functions
  - ✓ SVM, ridge regression, Lasso
  - ✓ 학습된 parameter w와 특징벡터 $\phi(x)$를 곱해 score 계산

$$\rightarrow \boldsymbol{score} = w^T \phi(x)$$

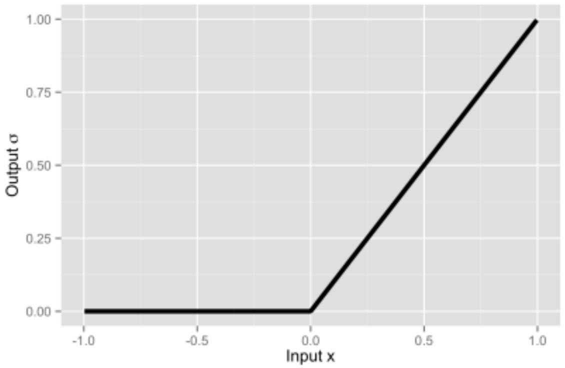
- Basic Neural Network(Multilayer Perceptron)
  - ✓ Hidden node(h_1, h_2)가 존재하는 extra layer를 추가
  - ✓ 이 때 σ는 비선형의 활성화함수

$$\rightarrow \boldsymbol{score} = w_1 h_1 + w_2 h_2$$
$$= w_1 \sigma\left(v_1^T \phi(x)\right) + w_2 \sigma\left(v_2^T \phi(x)\right)$$

Activation Functions



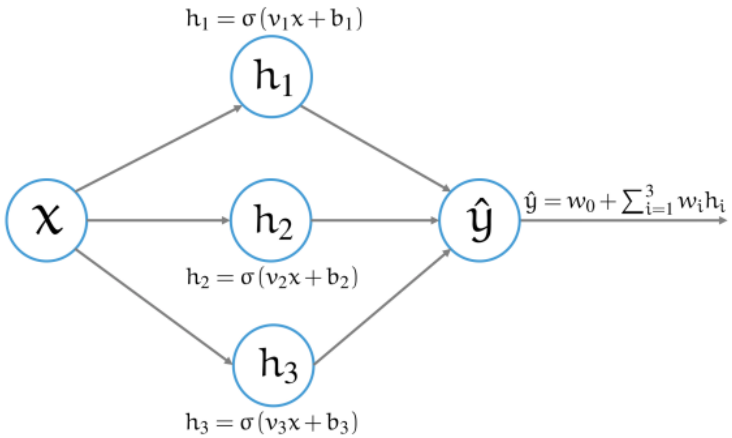
- Hyperbolic tangent function
  - ✓ 일반적인 활성화 함수 中 하나
    $\rightarrow \boldsymbol{\sigma}(\boldsymbol{x}) = \tanh(x)$



- Rectified linear function(ReLU)
  - ✓ 최근 들어서 더 많이 사용하는 함수
  - ✓ 계산이 더 빠르다는 장점
    $\rightarrow \boldsymbol{\sigma}(\boldsymbol{x}) = \max(0, x)$

01 Neural Network

Regression with Multilayer Perceptron(MLPs)



❖ MLPs가 3개의 node를 가진 hidden layer를 가정할 때 :

$$f(x) = w_0 + w_1 h_1(x) + w_2 h_2(x) + w_3 h_3(x)$$

❖ 이 때, 각 node에서의 연산 :

$$h_i(x) = \sigma(v_i x + b_i), for\ i = 1, 2, 3$$

❖ 우리가 학습 시켜야 하는 parameters :

$$b_1, b_2, b_3, v_1, v_2, v_3, w_0, w_1, w_2, w_3 \in \mathbb{R}$$

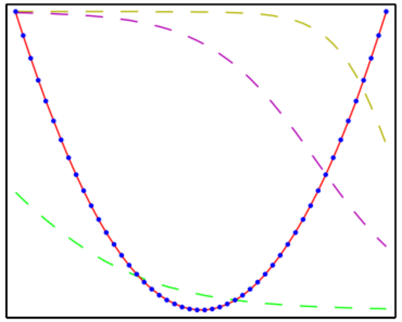
Methods of choosing the best parameters

- Empirical risk minimization
  - ✓ $\theta = (b_1, b_2, b_3, v_1, v_2, v_3, w_0, w_1, w_2, w_3)$
  - ✓ 가장 좋은 parameter를 선정하기 위해 empirical risk minimization 진행
  - ✓ Training set$(x_1, y_1), \dots, (x_n, y_n)$에 대해서 :

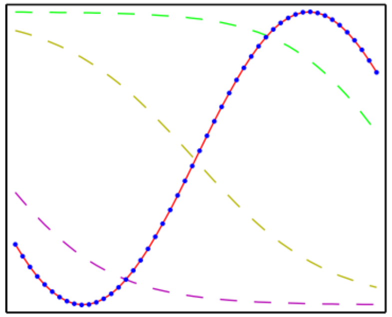
    $$\hat{\theta} = \underset{\theta \in \mathbb{R}^{10}}{argmin} \frac{1}{n} \sum_{i=1}^{n} (f_\theta(x_i) - y_i)^2$$

- Gradient Methods
  - ✓ $f(x) = w_0 + \sum_{i=1}^{3} w_i h_i(x) = w_0 + \sum_{i=1}^{3} w_i \tanh(v_i x + b_i)$
  - ✓ 해당 함수는 미분가능하기에 Gradient Method를 일반적으로 사용
  - ✓ 하지만, 목적함수가 concave하다면 전역 최솟값을 찾지 못할 수 있음

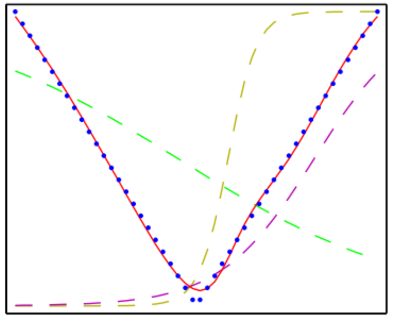
Approximation Properties of Multilayer Perceptrons



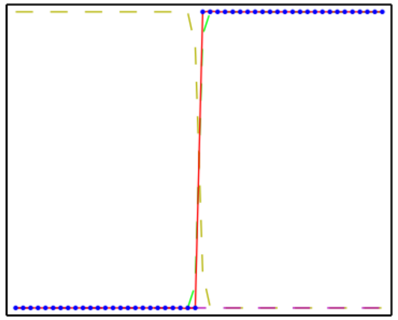
$f(x) = x^2$



$f(x) = \sin(x)$



$f(x) = | x |$



$f(x) = 1(x > 0)$

❖ Blue dots : training points

❖ Red line : final output

Approximation Properties of Multilayer Perceptrons

- Universal Approximation theorems
  - ✓ Leshno and Schocken (1991)이 제시한 이론
  - ✓ 이 이론에 따르면 매우 큰 하나의 은닉층을 가진 신경망은 활성화 함수가 선형이 아닌 경우에 한해 임의의 연속함수를 근사 시킬 수 있다는 이론

  - ➢ In more words :
    - ▪ $\psi(\cdot)$ 가 비선형(활성화) 함수이고 $f: K \to \mathbb{R}$을 집합 $K \subset \mathbb{R}^m$에서 임의의 연속함수라고 가정
    - ▪ 모든 $\epsilon > 0$에 대해서, 은닉 유닛의 수 N과 파라미터 $v_i, b_i \in \mathbb{R}, w_i \in \mathbb{R}^m$이 존재할 때 :
    - ▪ $F(x) = \sum_{i=1}^{N} v_i \psi(w_i^T x + b_i)$
    - ▪ 해당 함수가 모든 $x \in K$에 대해서 $| F(x) - f(x) | < \epsilon$을 만족시킴
    - ▪ 이 때, $F(x)$는 근사된 함수, $f(x)$는 주어진 함수로서 이 둘의 차이의 절댓값은 오차범위 ϵ보다 작은 것을 만족
    - ▪ 따라서 해당함수 $f(x)$를 오차범위 ϵ내에서 균일하게 근사가 가능

Recall: Multinomial Logistic Regression

- Multinomial Logistic Regression
  - ✓ $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{1, \dots, k\}$
  - ✓ 각 x에 대해서, k개의 class에 대한 분포를 생성하고자 함
  - ✓ 해당 범주형 분포를 확률벡터 $\theta = (\theta_1, \dots, \theta_k) \in \mathbb{R}^k$로 표현
    - ➢ 이 때, θ_y의 총합은 1($\sum_{y=1}^{k} \theta_y = 1$)
  - ✓ 각 x에 대해서 각 class의 score 함수를 계산
    - ➢ Score 함수의 형태 : $x \mapsto (< w_1, x >, \dots, < w_k, x >) \in \mathbb{R}^k$
  - ✓ $\mathbb{R}^k$ 벡터들을 확률벡터 θ에 매핑 해야함(각 class에 대한 확률값을 얻기 위해 Softmax 함수 활용)
    - ➢ Softmax 함수를 활용해서 범주형 분포에 매핑
    - ➢ $(s_1, \dots, s_k) \mapsto \theta = Softmax(s_1, \dots, s_k) = \left(\frac{\exp(s_1)}{\sum_{i=1}^{k} \exp(s_i)}, \dots, \frac{\exp(s_k)}{\sum_{i=1}^{k} \exp(s_i)} \right)$

How to learn Multinomial Logistic Regression

- Multinomial Logistic Regression learning
  - ✓ 주어진 x에 대해서 class y의 예측확률 :
    - ➢ $\hat{p}(y \mid x) = Softmax(< w_1, x >, \dots, < w_k, x >)_y$
  - ✓ 학습 과정으로서는 주어진 학습 데이터의 log-likelihood를 최대화 하는 것이 목표
    - ➢ 따라서 목적함수를 최대화 시켜줌 :
    - ➢ $argmax_{w_1,\dots,w_k \in \mathbb{R}^d} \sum_{i=1}^{n} \log\left[Softmax(< w_1, x >, \dots, < w_k, x >)_{y_i}\right]$
  - ✓ 해당 과정을 통해 log-likelihood를 최대화 시켜주는 가중치를 찾아 입력변수와 클래스 간의 관계를 모델링

Nonlinear Generalization of Multinomial Logistic Regression

- Multinomial Logistic Regression
  - ✓ Score 함수의 형태 : $x \mapsto (< w_1, x >, \dots, < w_k, x >) \in \mathbb{R}^k$
- 비선형 score function 활용
  - ✓ Score 함수의 형태 : $x \mapsto (f_1(x), \dots, f_k(x)) \in \mathbb{R}^k$
- 비선형 score function 클래스에 대한 예측확률 계산
  - ✓ $\hat{p}(\, y \mid x\,) = Softmax(f_1(x), \dots, f_k(x))_y$
- 학습단계에서 log-likelihood 최대화
  - ✓ $\underset{w_1,\dots,w_k \in \mathbb{R}^d}{arg\max} \sum_{i=1}^{n} \log\left[Softmax(f_1(x), \dots, f_k(x))_y\right]$

Multilayer Perceptron: Standard Recipe

- Multilayer Perceptron
  - ✓ Input space : $\mathcal{X} = \mathbb{R}^d$, Action space : $\mathcal{A} = \mathbb{R}^k$(for k-class classification)
  - ✓ $\sigma : \mathbb{R} \to \mathbb{R}$의 non-polynomial activation function이라고 가정(e.g. $tanh$ or $ReLU$)
  - ✓ 모든 hidden layer는 m개의 unit으로 구성되어 있다고 가정
    - ➢ 이 때, 첫번째 은닉층은 다음과 같이 형성 :
    - ➢ $h^{(1)}(x) = \sigma\left(W^{(1)}x + b^{(1)}\right)$
  - ✓ 각 다음 은닉층은 이전 층의 출력 $o \in \mathbb{R}^m$을 그대로 받아들여 다음과 같이 표현 :
    - ➢ $h^{(j)}(o) = \sigma\left(W^{(j)}o + b^{(j)}\right)$, for $j = 1, \dots, D$
    - ➢ D : 은닉층의 개수, $W^{(j)} \in \mathbb{R}^{m \times m}$: 매개변수 행렬, $b^{(D+1)} \in \mathbb{R}^k$: 편향벡터
  - ✓ 마지막 층은 $Affine$ 매핑으로 표현됨
    - ➢ $a(o) = W^{(D+1)}o + b^{(D+1)}$

Multilayer Perceptron: Standard Recipe

- Multilayer Perceptron
  - ✓ 따라서 전체 신경망 함수는 다음과 같이 표현 :
    - ➢ $f(x) = \left(a \cdot h^{(D)} \cdot \dots \cdot h^{(1)}\right)(x)$
    - ➢ 이를 통해 필요한 k개의 score function을 얻을 수 있음
  - ✓ log-likelihood를 최대화해서 training data에 대해서 class 예측 확률을 구함
    - ➢ $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log\left[Softmax\left(f(x_i)\right)_{y_i}\right]$
    - ➢ 이 때, $\theta = \left(W^{(1)}, \dots, W^{(D+1)}, b^{(1)}, \dots, b^{(D+1)}\right)$

1. Tikhonov Regularization

- ✓ ℓ_2나 ℓ_1 정규화 항을 목적함수에 추가해 λ를 조절하면 weight decay를 시행 시킬 수 있음 :
  - ➢ $J(w, v) = \sum_{i=1}^{n}\left(y_i - f_{w,v}(x_i)\right)^2 + \lambda_1 \, || \, w \, ||^2 \; + \lambda_2 \, || \, v \, ||^2$
  - ➢ weight decay : 가중치의 크기가 작아지도록 유도하고, 복잡한 모델의 효과를 제한해 과적합을 방지

2. Regularization by Early Stopping

- ✓ early stopping은 정규화의 한 방법으로 과적합 방지를 위해 훈련을 일찍 중단하는 방법 :
  - ① 훈련 중간에 일정한 간격으로 검증 데이터 셋에서 성능을 확인
  - ② 검증오차가 다시 상승하는 즉시 훈련을 바로 중단하지는 않음
  - ③ "인내심($patience$)" 매개변수를 설정, 이는 검증오차의 최솟값을 찾은 후에도 추가적인 훈련단계를 진행할 횟수
    - ➢ 인내심($patience$) 매개변수는 개선이 안된다고 바로 종료시키지 않고, 개선을 위해 몇 번의 에포크를 기다릴지 설정
  - ④ 검증오차가 T 단계에서 최솟값을 찾으면, 일정상수 c를 사용해 인내심을 $patience + cT$로 업데이트
  - ⑤ 중단하기 전 최소한의 인내심($patience$) 만큼 추가 단계를 실행

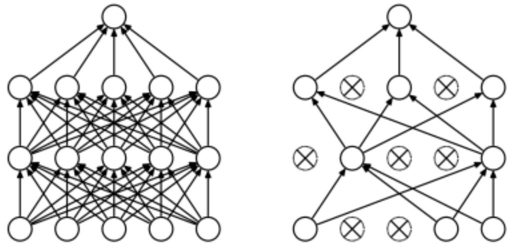
Methods of Neural Network Regularization

3. Max-norm Regularization

- ✓ 각 은닉 노드에서 들어오는 가중치 벡터의 최대 norm을 제한해서 사용 :
  - ➢ 각 은닉 노드에서 들어오는 가중치 벡터의 $\| w \|_2$이 특정상수 c 보다 크지 않게 제한
  - ➢ $\| w \|_2 \leq c$
  - ➢ 이 때 $\| w \|_2 > c$ 이면, 반지름이 c인 구 위에 $\| w \|_2$를 투영시키고 투영시키게 되면 norm의 크기가 제한됨
    - ▪ 과적합 방지 및 모델의 복잡성을 제어할 수 있음

4. Dropout for Regularization

- ✓ 고정된 확률 p를 선택해서 경사 하강 단계 이전에 각 노드는 확률 p로 "dropout"됨
- ✓ Dropout이 적용된 노드는 제거되며 해당 노드와 연결된 링크들도 제거됨(경사 하강 이후 모든 노드 복원)
- ✓ 학습 단계에서는 노드의 일부가 확률 p로서 비활성화되지만 예측단계에서는 모든 노드가 존재함
  - ➢ p가 커지면 입력 단위의 다양성을 높여서 모델의 일반화 능력을 향상시킴
- ✓ 어떤 노드라도 무작위로 사라질 수 있기에 노드간 지식 분산이 강제됨

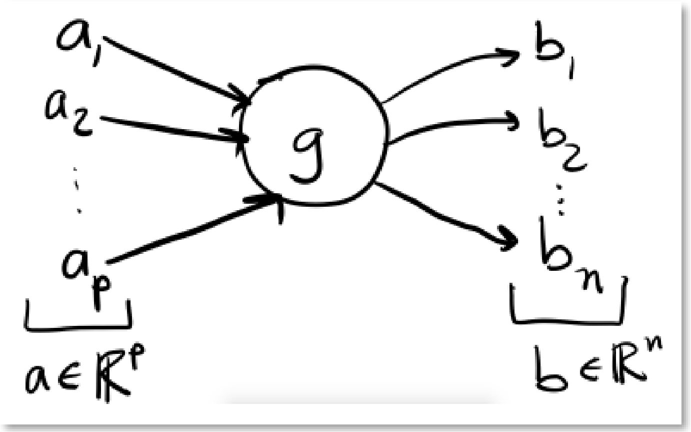


(a) Standard Neural Net (b) After applying dropout.

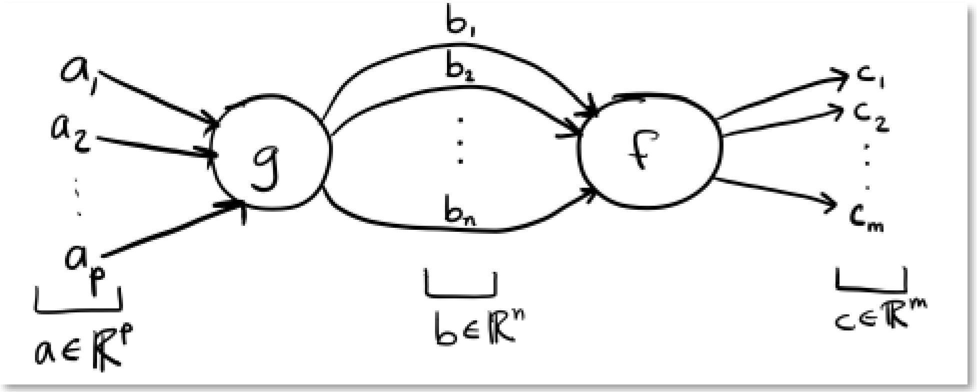
Partial Derivatives of an Affine Function

- Partial Derivatives
  - ✓ 해당 편도함수에서 $\frac{\partial b_i}{\partial a_j}$는 a_j를 변경함에 따라 b_i의 순간변화율
  - ✓ 만약 a_j를 약간 변화시켜 $a_j + \delta$가 된다면, b_i는 $b_i + \frac{\partial b_i}{\partial a_j}\delta$로 근사하게 됨

- Partial Derivatives of an Affine Function
  - ✓ 선형함수 $g(x) = Mx + c$를 정의
    - ➢ $M \in \mathbb{R}^{n \times p}$, $c \in \mathbb{R}$
  - ✓ 만약 $b = g(a)$라면, b_i는 M의 i번째 행에 의존
    - ➢ $b_i = \sum_{k=1}^{p} M_{ik} a_k + c_i$, $\frac{\partial b_i}{\partial a_j} = M_{ij}$
  - ✓ 따라서 선형 함수의 경우 M의 각 원소는 변화율을 직접적으로 나타냄



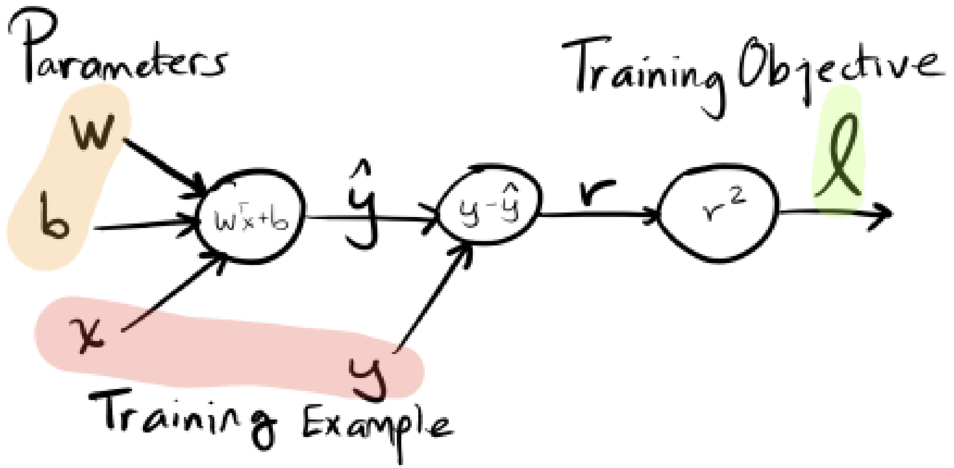
- Chain Rule
  - ✓ $g : R^p \to R^n and\ f : R^n \to R^m$, Let $b = g(a), c = f(b)$
  - ✓ $\frac{\partial c_i}{\partial a_j} = \sum_{k=1}^{n} \frac{\partial c_i}{\partial b_k}\frac{\partial b_k}{\partial a_j}$
  - ✓ a_j의 변화는 b_1부터 b_n까지 각각에 영향을 미칠 수 있음,
  - ✓ b_1부터 b_n까지의 변화는 각각 c_i에 영향을 미칠 수 있음
  - ✓ a_j로부터 c_i로 이어지는 각각의 경로에서 변화가 일어날 때, c_i의 전체 변화는 해당 경로들을 따라 변화가 발생한 값들의 합이라는 것을 알려줌



Example

1. Least Squares Regression

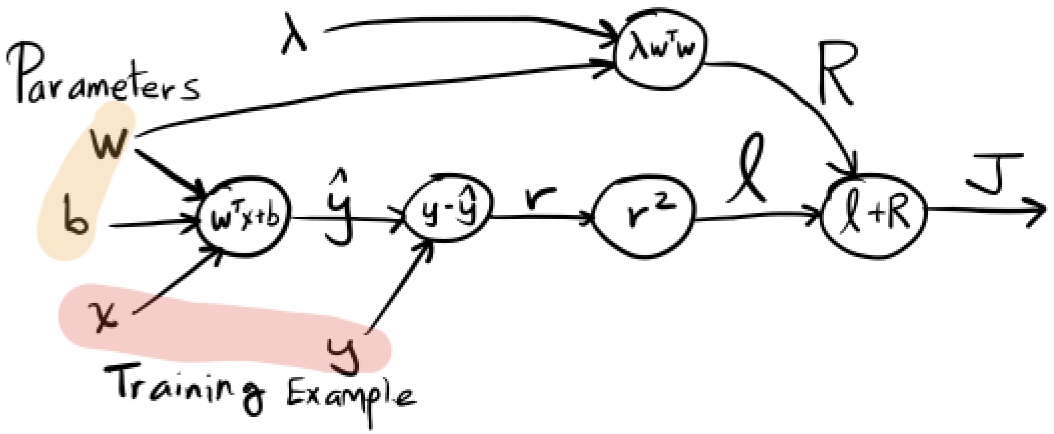
- ✓ 일반적인 training point(x, y)에서 손실함수 $\ell(w, b) = [(w^T x + b) - y]^2$
  - ➢ (prediction) $\hat{y} = \sum_{j=1}^{d} w_j x_j + b$
  - ➢ (residual) $r = y - \hat{y}$
  - ➢ (loss) $\ell = r^2$
- ✓ $\frac{\partial l}{\partial r} = 2r$
- ✓ $\frac{\partial l}{\partial \hat{y}} = \frac{\partial l}{\partial r}\frac{\partial r}{\partial \hat{y}} = (2r)(-1) = -2r$
- ✓ $\frac{\partial l}{\partial b} = \frac{\partial l}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b} = (-2r)(1) = -2r$
- ✓ $\frac{\partial l}{\partial w_j} = \frac{\partial l}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j} = (-2r)x_j = -2rx_j$



Example

2. Ridge Regression

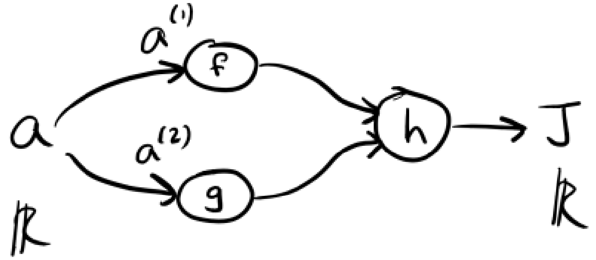
- ✓ ℓ_2 정규화 항에서의 목적함수 $J(w,b) = [(w^T x + b) - y]^2 + \lambda w^T w$
  - ➢ (prediction) $\hat{y} = \sum_{j=1}^{d} w_j x_j + b$
  - ➢ (residual) $r = y - \hat{y}$
  - ➢ (loss) $\ell = r^2$
  - ➢ (regularization) $R = \lambda w^T w$
  - ➢ (objective) $J = \ell + R$



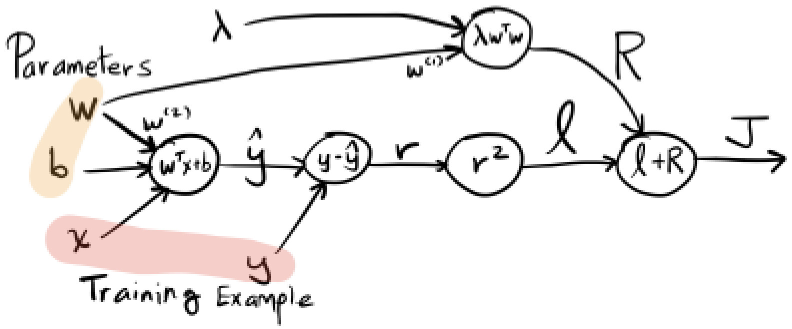
- ✓ $\frac{\partial J}{\partial \ell} = \frac{\partial J}{\partial R} = 1$
- ✓ $\frac{\partial J}{\partial \hat{y}} = \frac{\partial J}{\partial \ell}\frac{\partial \ell}{\partial r}\frac{\partial r}{\partial \hat{y}} = (1)(2r)(-1) \ = \ -2r$
- ✓ $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b} = \ (-2r)(1) \ = \ -2r$
- ✓ $\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j} = \ (-2r)x_j \ = \ -2rx_j$

Handling Nodes with Multiple Children

- $a \mapsto J = h(f(a), g(a))$
  - ✓ a의 독립적인 복사본 $a^{(1)}, a^{(2)}$가 존재
    - ➢ $\frac{\partial J}{\partial a} = \frac{\partial J}{\partial a^{(1)}}\frac{\partial a^{(1)}}{\partial a} + \frac{\partial J}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial a} = \frac{\partial J}{\partial a^{(1)}} + \frac{\partial J}{\partial a^{(2)}}$
    - ➢ a에 대한 미분은 $a^{(1)}$과 $a^{(2)}$ 각각에 대한 미분의 합과 같음

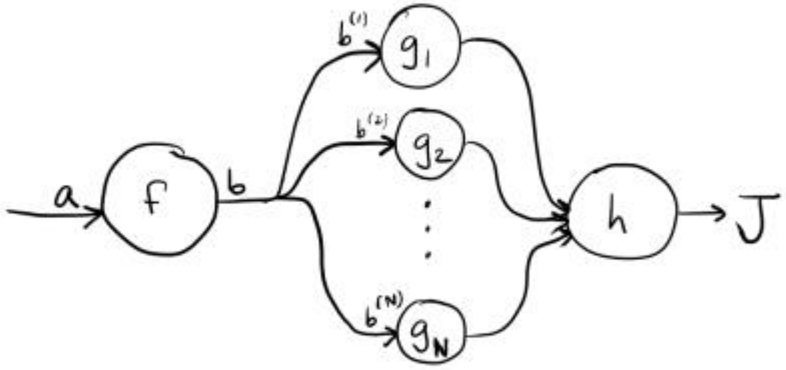


- Ridge Regression에 적용
  - ✓ $\frac{\partial J}{\partial \hat{y}} = \frac{\partial J}{\partial \ell}\frac{\partial \ell}{\partial r}\frac{\partial r}{\partial \hat{y}} = (1)(2r)(-1) \ = \ -2r$
  - ✓ $\frac{\partial J}{\partial w_j^{(2)}} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j^{(2)}} = \frac{\partial J}{\partial \hat{y}} x_j$
  - ✓ $\frac{\partial J}{\partial w_j^{(1)}} = \frac{\partial J}{\partial R}\frac{\partial R}{\partial w_j^{(1)}} = (1)(2\lambda w_j^{(1)})$
  - ✓ $\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial w_j^{(1)}} + \frac{\partial J}{\partial w_j^{(2)}}$



Backpropagation

- Backpropagation(역전파)
  - ✓ 역전파 알고리즘은 input과 output을 모두 알고 있는 상태에서 신경망을 학습시키는 방법
  - ✓ 실제값과 모델의 예측 결과인 output이 얼마나 차이가 나는지 확인하고 그 오차를 줄이기 위해 가중치와 편향에 대한 gradient를 뒤에서 앞으로 업데이트 진행



- 모든 node가 하나의 input을 받고 하나의 output을 받는다고 가정
  - ✓ 노드 f에 대한 역전파:
    - ➢ Input : $\frac{\partial J}{\partial b_j^{(1)}}, \dots, \frac{\partial J}{\partial b^{(N)}}$
    - ➢ Output : $\frac{\partial J}{\partial b_j} = \sum_{k=1}^{N} \left(\frac{\partial J}{\partial b_j^{(k)}}\right)$

$$\rightarrow \frac{\partial J}{\partial a_i} = \sum_{j=1}^{n} \frac{\partial J}{\partial b_j} \frac{\partial b_j}{\partial a_i}$$

Q&A

감사합니다.