Pattern Recognition and Machine Learning:

Neural Networks

ESC 2024 Spring Session 5주차
한지희 장덕재

Contents

1. Feed-Forward Neural Networks
2. Network Training
3. Error Backpropagation
4. Applications of Backpropagation
5. Regularization in Neural Networks
6. Mixture Density Networks
7. Bayesian Neural Networks

1

Feed-Forward Network Functions

1. Basic Network Model
2. Functional Form of Basic Network Model
3. Generalization of Basic Network Model
4. Properties of Feed-Forward Network Model

(Review) Functional Form of Linear Model

- Linear model for regression or classification: linear combination of fixed nonlinear basis function

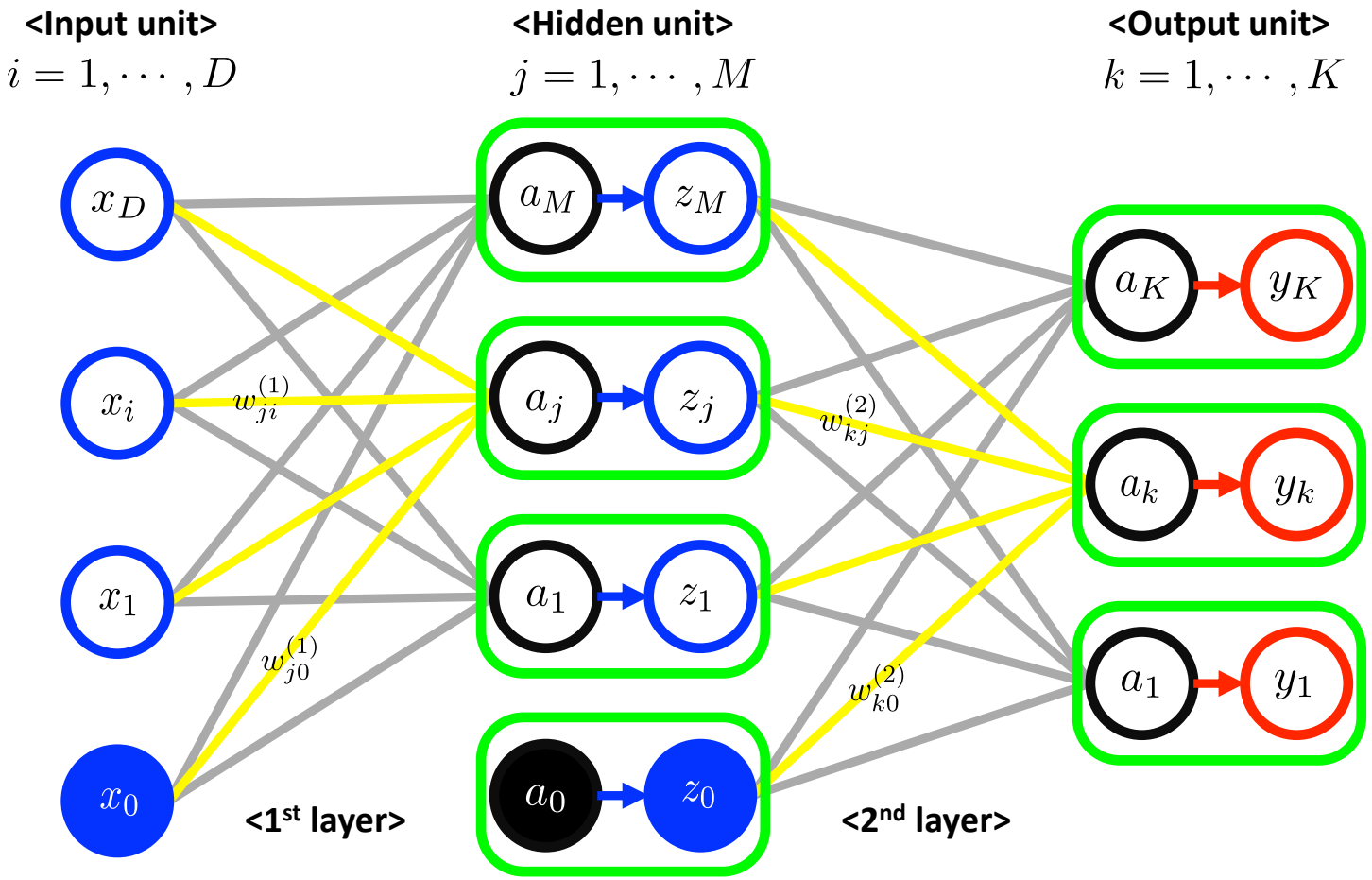
$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

$$f(\cdot) = \begin{cases} \text{identity function in regression model} \\ \text{nonlinear activation function in classification model} \end{cases}$$

  - Curse of dimensionality → need to adapt the basis function to data
  - Neural Network model: fix the number of basis function but allow them to be adaptive using adaptive parameter!

- Goals:
  ① extend this model by making the basis function depend on parameter
  ② adjust parameter along with coefficient w_j during training

Basic Network Model



$x_1, \cdots, x_D$: input vectors

$y_1, \cdots, y_K$: output vectors

a_j, a_k : activations

$w_{ji}^{(1)}$: 1st layer weight from ith unit to jth unit

x_0, z_0 : basis (activation = 1)

→ : activation function $h(\cdot)$(hidden unit)

→ : activation function $h^{\star}(\cdot)$(output unit)

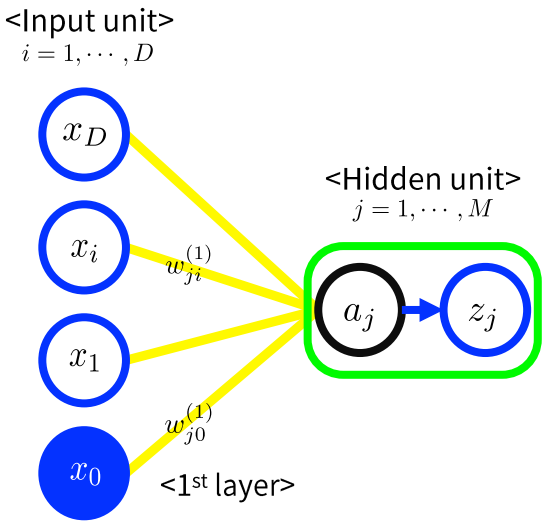
※ activation function: nonlinear & differentiable

Functional Form of Basic Neural Model

1. Hidden Unit

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

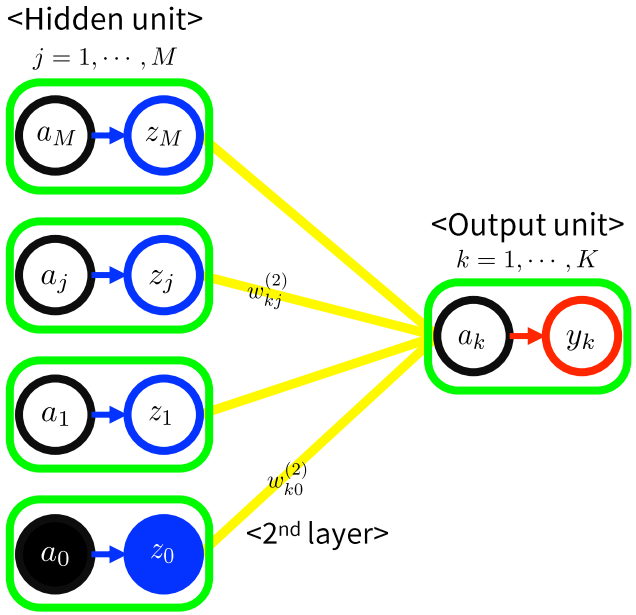
$$z_j = h(a_j)$$



2. Output unit

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

$$y_k = h^{\star}(a_k)$$



3. Overall Network Function

$$y(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

$$= \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)}\right)\right)$$

Network model is <u>a nonlinear function</u>
from input to output
<u>controlled by a vector of adjustable parameters</u>

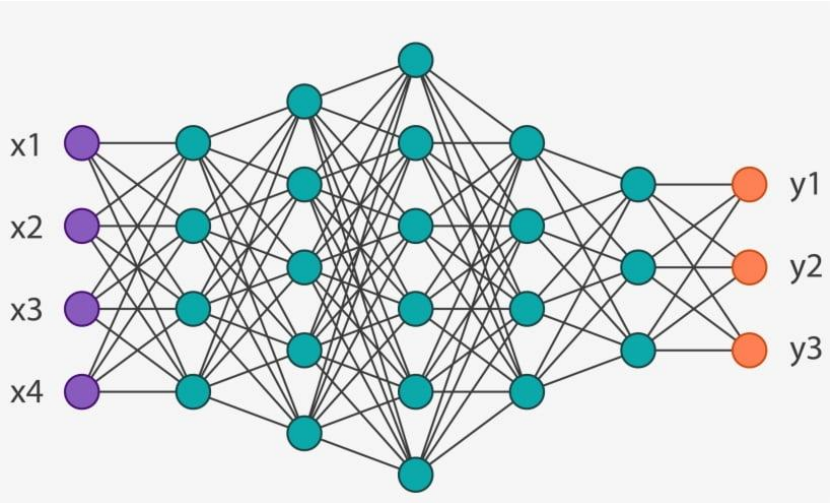
※ **output unit activation function**

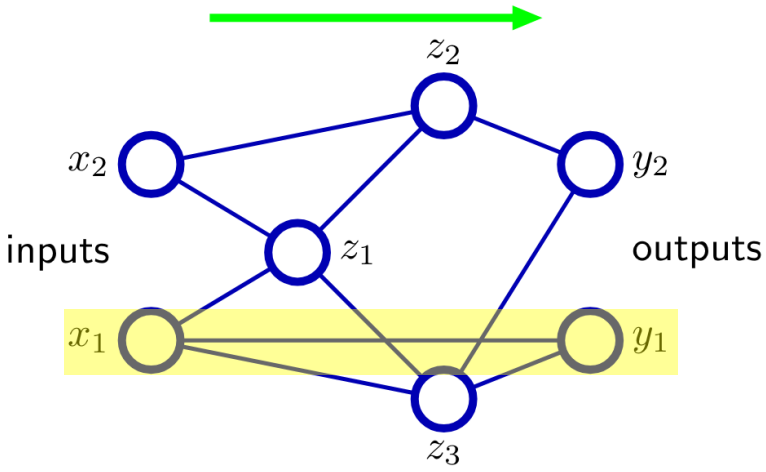
: determined by the nature of data, assumed dist'n of target variables, ⋯

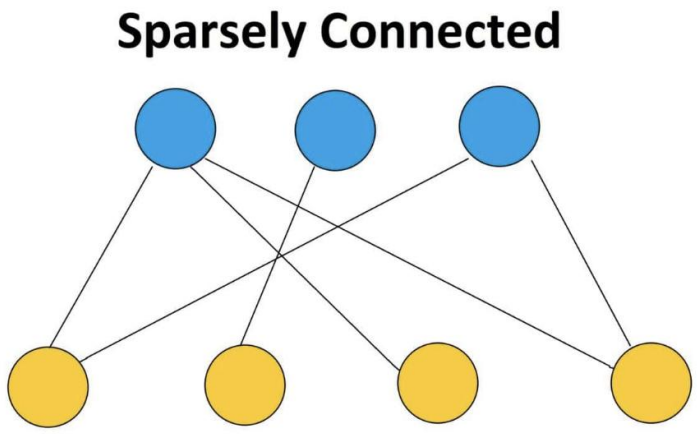
- standard regression : identity function
- multiple binary classification: logistic sigmoid
- multiclass classification: softmax

Generalization of Basic Network Model

1. Add additional layers
2. Include skip-layer connections
3. Sparse Network (↔ Dense network)







Properties of Feed-Forward Network Model

1. Universal approximator
2. Symmetries of Weight-space

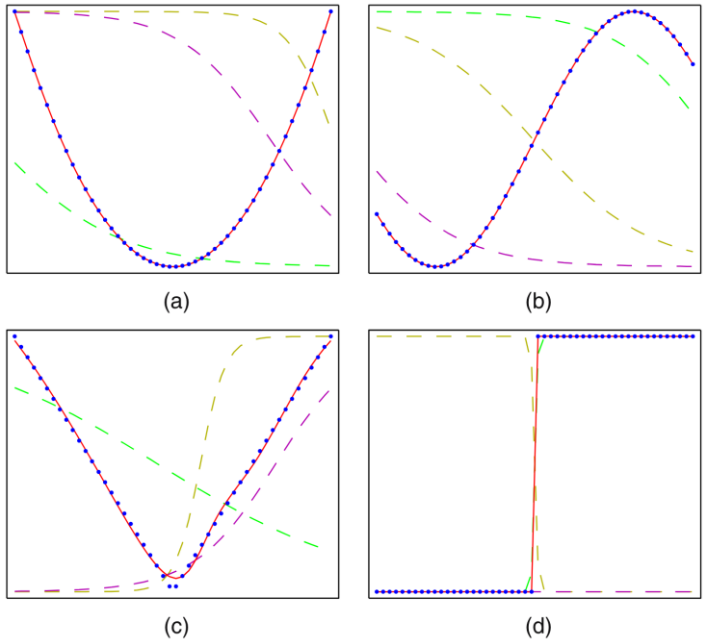
   : multiple distinct weight vector can give rise to the same mapping functions from inputs to outputs

   ① Sign-flip symmetry

   : change the signs of a particular group of weights

   → input-output mapping represented by the network is unchanged

   e.g., 2-layer NN with 'tanh' activation function (odd function)



   ② Interchange symmetry

   : interchange the values of all the weights leading both into and out of a particular hidden unit with the corresponding values of weights associated with a different hidden unit

   → input-output mapping represented by the network is unchanged

2

Network Training

1. Parameter Optimization in Network Model
2. Local Quadratic Approximation
3. Gradient Descent Optimization

Parameter Optimization in Network Model

- Goal: To find $\mathbf{w}^\star = \arg\min_{\mathbf{w}} E(\mathbf{w})$

  $$\mathbf{w} \to \mathbf{w} + \delta\mathbf{w} \Rightarrow \mathrm{E}(\mathbf{w}) \to \mathrm{E}(\mathbf{w}) + \delta\,\mathrm{E}(\mathbf{w})$$

  $$\text{where } \delta\,\mathrm{E}(\mathbf{w}) \simeq \delta\mathbf{w}^T \nabla\,\mathrm{E}(\mathbf{w})$$

  $$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \mathrm{E}(\mathbf{w}) \text{ s.t., } \nabla\,\mathrm{E}(\mathbf{w}^\star) = 0$$

  $\Rightarrow$ $\boxed{\text{move in direction of } -\nabla\,\mathrm{E}(\mathbf{w})}$

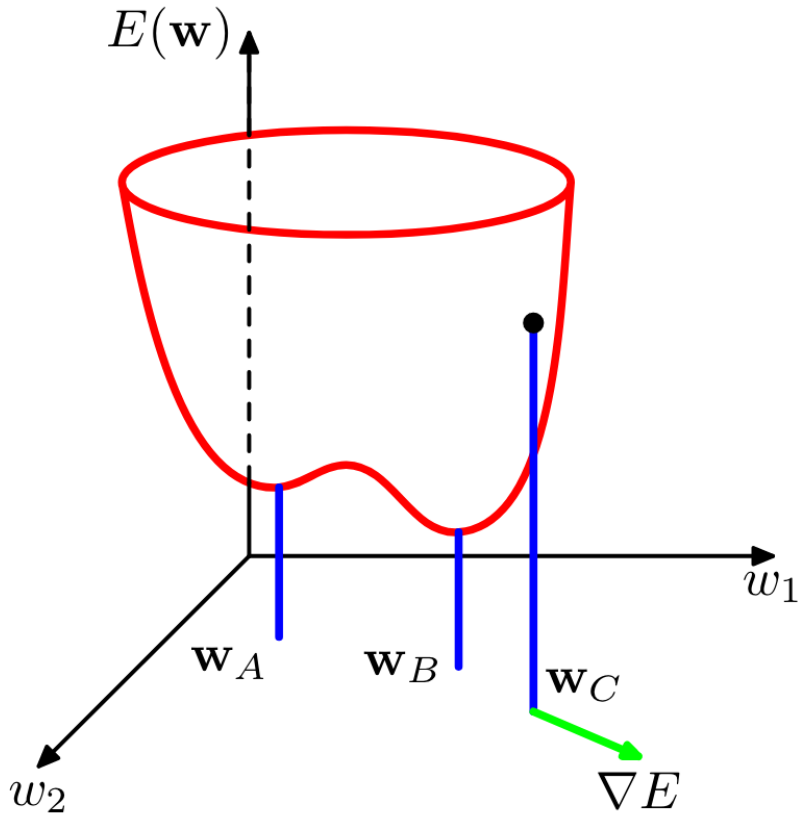
- Since $\mathrm{E}(\mathbf{w})$ has a highly nonlinear dependency on weights and biases, there will be many points in weight space at which gradient vanishes.

  + symmetries of weight space

  $\Rightarrow$ compare several local minima to find sufficiently good solution

- **Optimization of continuous nonlinear function**
  1. Choose initial value $\mathbf{w}^{(0)}$
  2. Move through weight space in a succession of of steps in form

  $$\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} + \boxed{\Delta\mathbf{w}^{(\tau)}}$$ ← Use gradient information



Taylor expansion and Approximation

- Taylor polynomial

$$f(x) \simeq p_n(x) = \frac{f(a)}{0!} + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

: $x = a$ 에서 $f(x)$ 와 동일한 미분계수를 갖는 n차 다항함수 $p_n(x)$로 $f(x)$를 근사할 수 있다.

✓ $x = a$ 근처에서만 성립한다.

✓ $x \to a,\ n \to \infty$ 일 수록 $p_n(x)$ 는 $f(x)$를 더 잘 근사한다.

1. Linear approximation

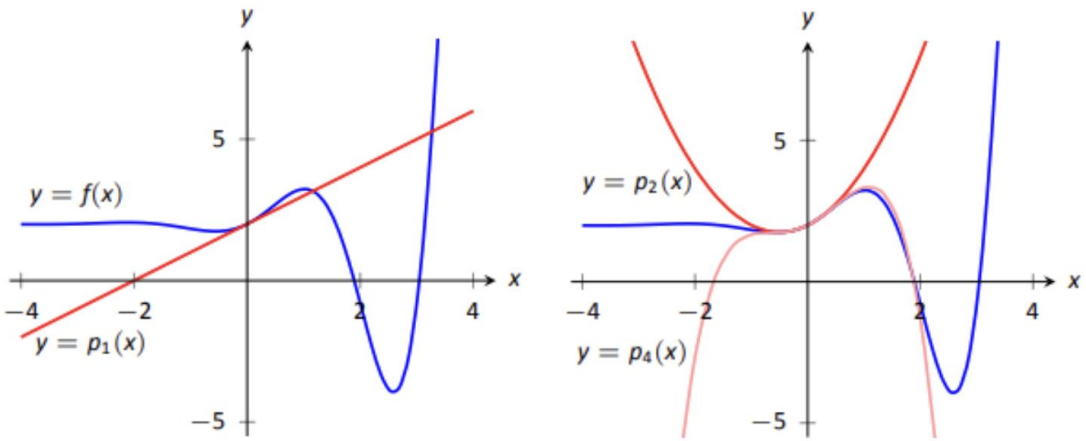
$$f(x) \simeq L(x) = f(a) + f'(a)(x-a)$$

$$f(\mathbf{x}) \simeq L(\mathbf{x}) = f(\mathbf{x}_0) + (\mathbf{x}-\mathbf{x}_0)^T\mathbf{b}$$

2. Quadratic approximation

$$f(x) \simeq Q(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2$$

$$f(\mathbf{x}) \simeq Q(\mathbf{x}) = f(\mathbf{x}_0) + (\mathbf{x}-\mathbf{x}_0)^T\mathbf{b} + \frac{1}{2}(\mathbf{x}-\mathbf{x}_0)^T\mathbf{H}(\mathbf{x}-\mathbf{x}_0)$$



- Quadratic approximation of $E(\mathbf{w})$ around $\hat{\mathbf{w}}$

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w}-\hat{\mathbf{w}})^T\mathbf{b} + \frac{1}{2}(\mathbf{w}-\hat{\mathbf{w}})^T\mathbf{H}(\mathbf{w}-\hat{\mathbf{w}})$$

$$\nabla E(\mathbf{w}) \simeq \mathbf{b} + \mathbf{H}(\mathbf{w}-\hat{\mathbf{w}}) \qquad \text{where } \mathbf{b} = \nabla E(\hat{\mathbf{w}}),\ \mathbf{H} = \nabla\nabla E(\hat{\mathbf{w}})$$

Local Quadratic Approximation around w*

- Let $\mathbf{w}^\star = \arg\min_{\mathbf{w}} E(\mathbf{w})$ then $\nabla \mathrm{E}(\mathbf{w}^\star) = \mathbf{b} = 0$

$$E(\mathbf{w}) \simeq E(\mathbf{w}^\star) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^\star)^T\mathbf{H}(\mathbf{w}-\mathbf{w}^\star)$$

$$\nabla E(\mathbf{w}) \simeq \mathbf{H}(\mathbf{w}-\mathbf{w}^\star) \qquad \text{where } \mathbf{H} = \nabla\nabla E(\mathbf{w}^\star)$$

- Consider the eigenvalue equation: $\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i$
- Expand $(\mathbf{w}-\mathbf{w}^\star)$ as a linear combination of $\mathbf{u}_i$: $(\mathbf{w}-\mathbf{w}^\star) = \sum_i \alpha_i \mathbf{u}_i$
- Regard it as a transformation of coordinate system (origin → $\mathbf{w}^\star$, axes → $\mathbf{u}_i$)

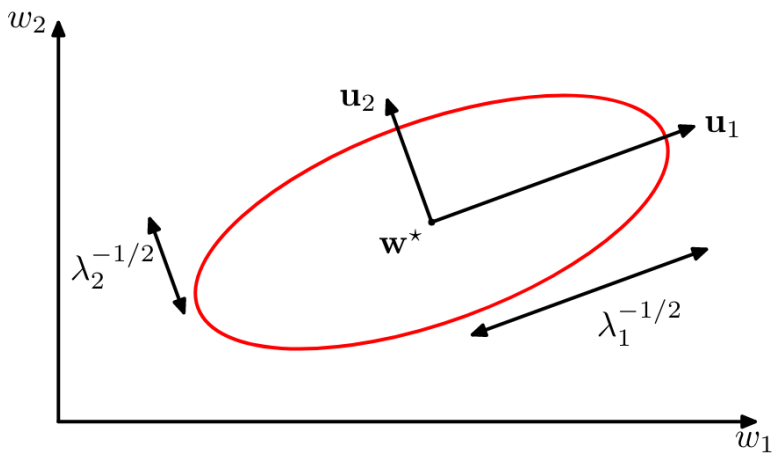
  : $E(\mathbf{w}) = E(\mathbf{w}^\star) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$
- $\mathbf{H}$ is positive definite iff $\lambda_i > 0$

$$\mathbf{v}^T\mathbf{H}\mathbf{v} = \sum_i c_i^2 \lambda_i > 0 \; (\because \mathbf{v} = \sum_i c_i \mathbf{u}_i)$$

- A stationary point $\mathbf{w}^\star$ is minimum iff Hessian matrix evaluated at $\mathbf{w}^\star$ is p.d.

※ **Second derivative test** (f has a critical value at $\mathbf{a}$)

- If Hessian is p.d. (all $\lambda_i > 0$) at $\mathbf{a}$, f has local minimum at $\mathbf{a}$.
- If Hessian is n.d. (all $\lambda_i < 0$) at $\mathbf{a}$, f has local maxima at $\mathbf{a}$.
- If Hessian has both positive and negative λ_i, f has saddle point at $\mathbf{a}$.



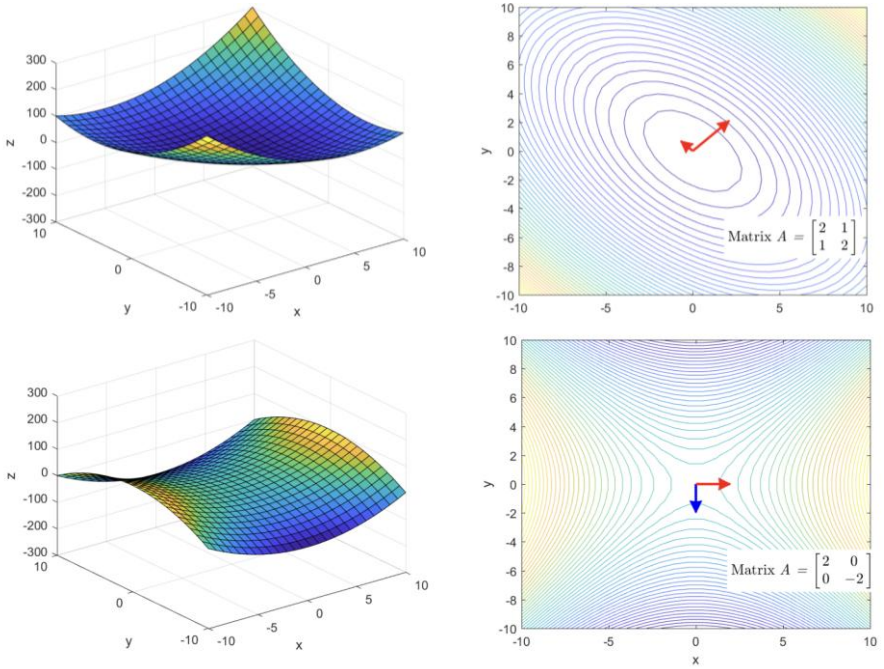


그림 출처: https://angeloyeo.github.io/2020/06/17/Hessian.html

Gradient Descent Optimization

- **Optimization of continuous nonlinear function**
  1. Choose initial value $\mathbf{w}^{(0)}$
  2. Move through weight space in a succession of of steps in form

$$\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} + \boxed{\Delta\mathbf{w}^{(\tau)}} \quad \leftarrow \quad \text{Use gradient information}$$

1) Batch Gradient Descent

- Choose weight update to comprise small step in the direction of negative gradient

$$\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} \boxed{-\eta\nabla \mathrm{E}(\mathbf{w}^{(\tau)})}$$

- $E(\mathbf{w})$ is defined with respect to entire training set
  → require entire training set to evaluate $\nabla E(\mathbf{w})$

- To find sufficiently good minimum, run multiple times using different randomly chosen starting point and compare resulting performance

2) On-line Gradient Descent (= sequential / stochastic GD)

- $E(\mathbf{w})$ is based on maximum likelihood for a set of independent observations

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

- Update weight by cycling data in sequence or randomly selecting and replacing

$$\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} \boxed{-\eta\nabla \mathrm{E_n}(\mathbf{w}^{(\tau)})}$$

- handle redundancy in data much more efficiently
- Possibility of escaping from local minima

3

Error Backpropagation

1. Two Stages in Iterative Procedure to Minimize Error Function
2. Derivation of Backpropagation
3. A Simple Example
4. Efficiency of Backpropagation
5. Summary of Optimization in Network Model

Two Stages in Iterative Procedure for Minimizing Error Function

- Optimization in continuous nonlinear function
  I. Choose initial value $\mathbf{w}^{(0)}$
  II. Update parameter for minimizing error function $E(\mathbf{w})$

Iteratively
  i) Evaluate the derivatives of error function with respect to $\mathbf{w}$ ← Backpropagation
  ii) **Compute the adjustment** to be made to the weights ← GD

- Backpropagation: a local message passing scheme in which information is sent alternatively forwards and backwards through the network

Derivation of Backpropagation

- Goal: To evaluate $\nabla E_n(\mathbf{w})$ where $E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$
- Consider a general feed-forward Network: $a_j = \sum_i w_{ji} z_i$
- Then, $z_j = h(a_j)$
- To update w_{ji} , we need to evaluate $\dfrac{\partial E_n}{\partial w_{ji}}$

by chain rule,

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$

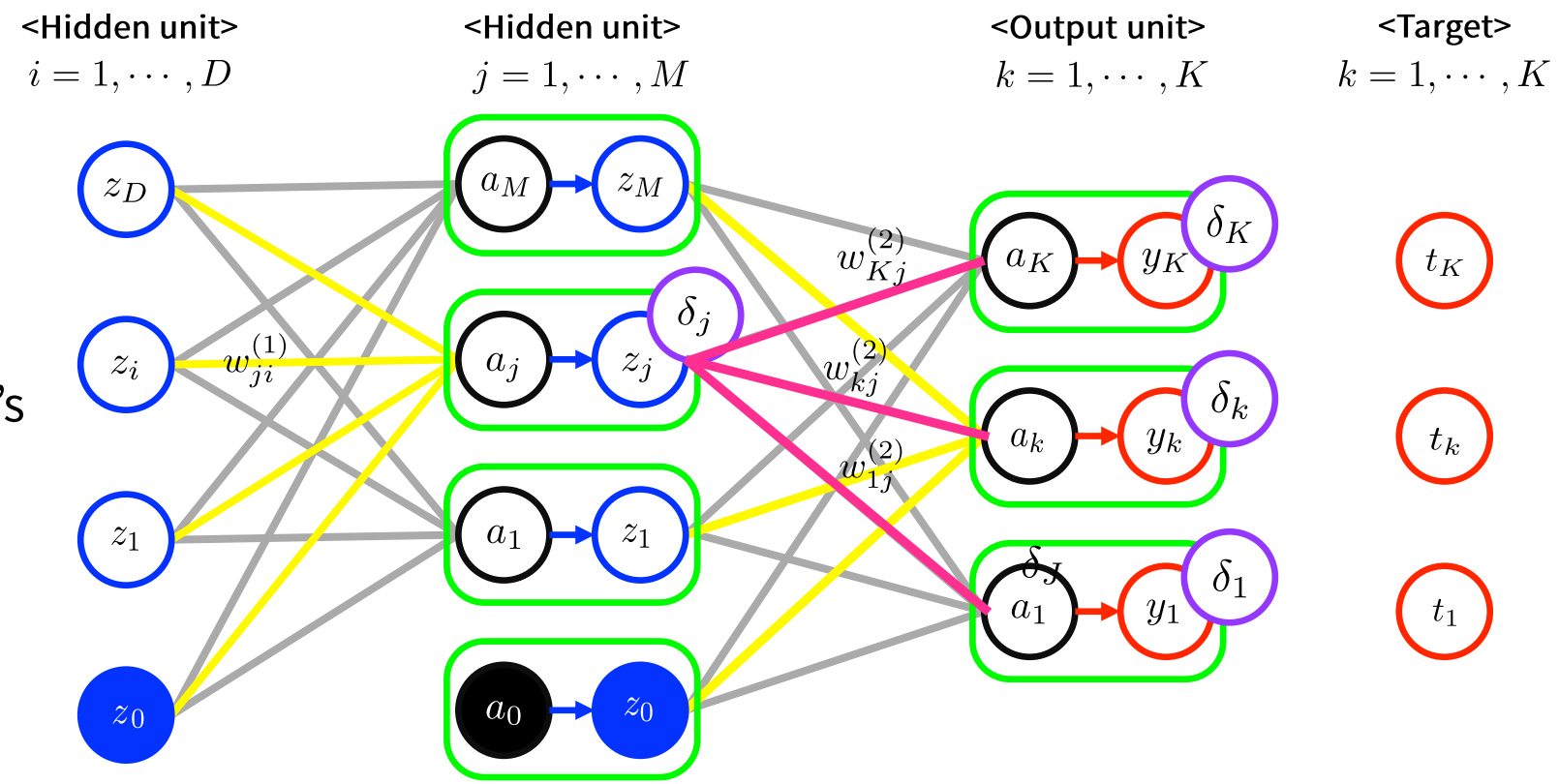
$$= h'(a_j)\sum_k w_{kj}\delta_k \text{ where } \delta_k = y_k - t_k$$

$$\therefore \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i = h'(a_j)\sum_k w_{kj}\delta_k z_i$$

If output unit activation function is canonical link function, then $\dfrac{\partial E}{\partial a_k} = y_k - t_k$

Summary of Backpropagation

① Forward propagation
② Evaluate δ_k for all output units
③ Obtain δ_j by backpropagating δ_k's
④ Evaluate required derivatives



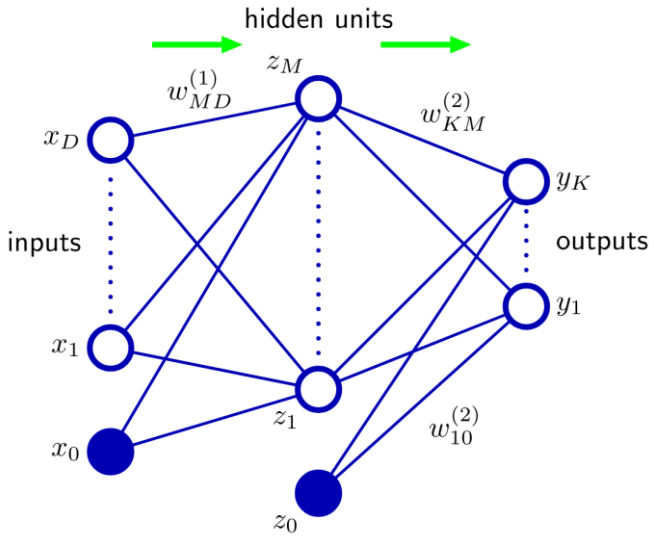
Derivative using backprop. is allowed for general forms for (1) Network topology (2) error function (3) activation function

A Simple Example

- Setting:
  1. Two-layer Network
  2. Sum-of-squares error function $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$
  3. Hidden unit activation function = tanh, Output unit activation function = identity

$$h(a) \equiv \tanh(a)$$

$$\text{where } \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}, \quad h'(a) = 1 - h(a)^2$$



① Forward propagation

② Compute δ_k for each output unit

③ Obtain δ_j

④ Evaluate derivatives at each layer

Efficiency of Backpropagation

- Computational efficiency of back propagation: forward-propagation: $O(W)$, backpropagation: $O(W)$

- vs. numerical differentiation
  1. Finite differences

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji})}{\epsilon} + O(\epsilon)$$

  2. Central differences

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji}+\epsilon) - E_n(w_{ji}-\epsilon)}{2\epsilon} + O(\epsilon^2)$$

  → forward-propagation: $O(W)$, backpropagation: $O(W^2)$

- In practice, after training model using backpropagation, **compare the derivatives with those obtained using central differences to check the correctness of implementation of backpropagation**

Summary of Optimization in Network Model

- Given: input vector $\{\mathbf{x}_i\}$, target vector $\{\mathbf{t}_k\}$
- Goal: To find $\mathbf{w}^\star = \arg\min_{\mathbf{w}} E(\mathbf{w})$
- How?
  1. Initiate weights $\mathbf{w} := \mathbf{w}^{(0)}$
  2. Forward propagation → get activations at each unit: $a_i, z_i \to \cdots \to a_k, y_k$
  3. Error back propagation → get error at each layer: $\delta_k \to \delta_j \to \delta_i \to \cdots$
  4. Evaluate derivatives at each layer: $\dfrac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$
  5. Update weights: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} - \eta \nabla \mathrm{E_n}(\mathbf{w}^{(\tau)})$

4

Applications of Backpropagation

1. The Jacobian Matrix
2. The Hessian Matrix
   - 2.1. Approximation of Hessian and Its Inverse
   - 2.2. Exact Evaluation of Hessian

The Jacobian Matrix

- Elements of Jacobian matrix are given by **the derivatives of network outputs with respect to inputs:**

$$J_{ki} = \frac{\partial y_k}{\partial x_i}$$

- Useful for system built from a number of distinct modules
- Provide a measure of **local sensitivity** of the outputs to change in each input variable:

$$\Delta y_k \simeq \sum_i \frac{\partial y_k}{\partial x_i} \Delta x_i \qquad \text{valid when } |\Delta x_i| \text{ is small}$$

- Jacobian matrix is the matrix **representing best linear map approximation** near a certain point
- network mapping represented by a trained neural network will be **nonlinear**
  → elements of Jacobian are **not constants**
  → must be **re-evaluated** for each new input vector
- Jacobian matrix can be evaluated using back propagation

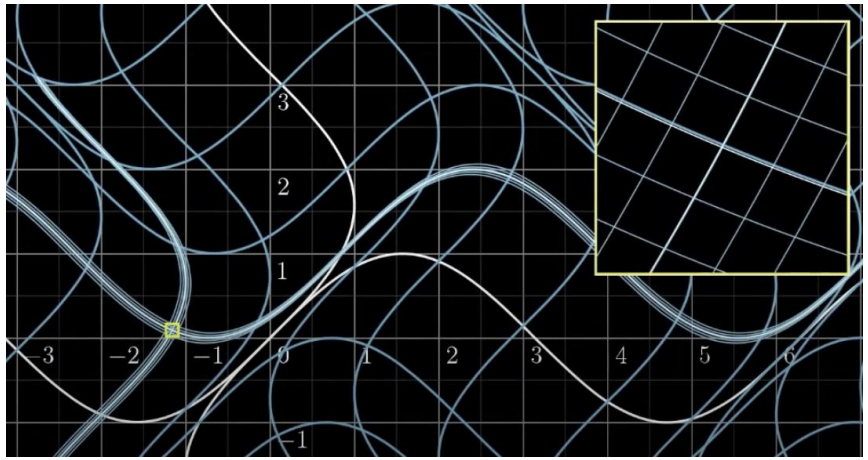
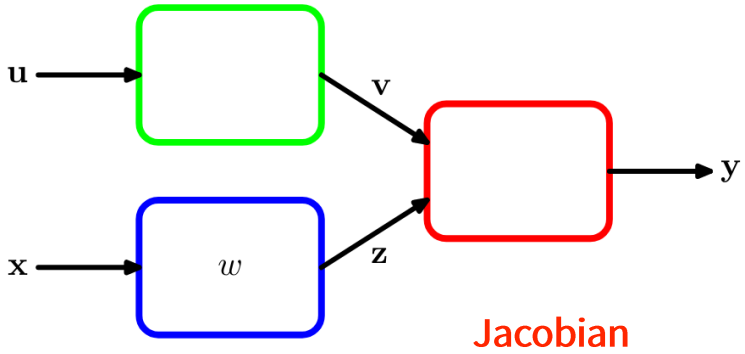


그림 출처: https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/jacobian/v/the-jacobian-matrix



$$\frac{\partial E}{\partial w} = \sum_{k,j} \frac{\partial E}{\partial y_k} \boxed{\frac{\partial y_k}{\partial z_j}} \frac{\partial z_j}{\partial w}$$

Evaluation of Jacobian Matrix

- Jacobian can be evaluated using an extension for backpropagation
- Procedure for evaluating Jacobian:
  1. Apply the input vector corresponding to the Jacobian's input space
  2. Forward propagate to obtain activations of all hidden and output unit
  3. For each row $\mathbf{k}$ of Jacobian matrix, backpropagate from output to input

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j}\frac{\partial a_j}{\partial x_i} = \sum_j w_{ji}\frac{\partial y_k}{\partial a_j}$$

$$\frac{\partial y_k}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l}\frac{\partial a_l}{\partial a_j} = h'(a_j)\sum_l w_{lj}\frac{\partial y_k}{\partial a_l}$$

- If output unit activation function is sigmoid, $\frac{\partial y_k}{\partial a_j} = \delta_{kj}y_k - y_k y_j$

  or softmax, $\frac{\partial y_k}{\partial a_j} = \delta_{kj}\sigma'(a_j)$

- Implementation of backpropagation algorithm can be checked by using numerical differentiation.

The Hessian Matrix

- Elements of Hessian matrix are given by **the second derivatives of error:**

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}}$$

- Hessian plays an important role in
  1. Several nonlinear optimization algorithms to consider the second-order properties of the error surface
  2. Re-training a feed-forward network following a small change in the training data
  3. Identifying the least significant weights in pruning algorithm (use inverse)
  4. Laplace approximation for a Bayesian NN (5.7절) (use inverse, eigenvalues, determinants)
- Require high computational effort: $O(W^2)$

Approximation of Hessian and Its Inverse (1)

- Sometimes **the inverse of Hessian** is required rather than Hessian itself
  → approximate Hessian to find its inverse easily

1. Diagonal approximation

- Replace off-diagonal with 0

$$\frac{\partial^2 E_n}{\partial w_{ji}^2} = \frac{\partial^2 E_n}{\partial a_j^2} z_i^2$$

$$\frac{\partial^2 E_n}{\partial a_j^2} = h'(a_j)^2 \sum_k \sum_{k'} w_{kj} w_{k'j} \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} + h''(a_j) \sum_k w_{kj} \frac{\partial E^n}{\partial a_k}$$

- Neglect off-diagonals in second derivative terms

$$\frac{\partial^2 E_n}{\partial a_j^2} = h'(a_j)^2 \sum_k w_{kj}^2 \frac{\partial^2 E_n}{\partial a_k^2} + h''(a_j) \sum_k w_{kj} \frac{\partial E_n}{\partial a_k}$$

Approximation of Hessian and Its Inverse (2)

2. Outer-product approximation

- In regression problem, sum-of-squares error function is: $E = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2$
- Its Hessian matrix is: $\mathbf{H} = \nabla\nabla E = \sum_{n=1}^{N}\nabla y_n \nabla y_n + \sum_{n=1}^{N}(y_n - t_n)\nabla\nabla y_n$
- If the network has been trained properly, $y_k \to t_k$, then $\mathbf{H} \simeq \sum_{n=1}^{N}\mathbf{b}_n\mathbf{b}_n^{\mathrm{T}}$
- In multiclass classification, $\mathbf{H} \simeq \sum_{n=1}^{N} y_n(1-y_n)\mathbf{b}_n\mathbf{b}_n^{\mathrm{T}}$
- **Sequential procedure** for building up Hessian by including data points one at a time until L+1 = N :

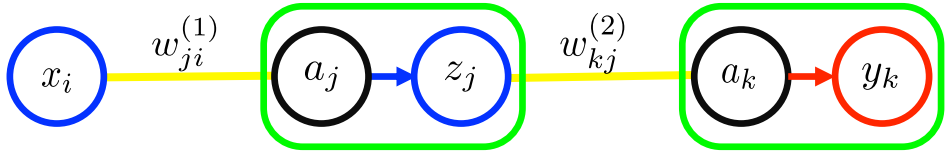
$$\mathbf{H}_{L+1} = \mathbf{H}_L + \mathbf{b}_{L+1}\mathbf{b}_{L+1}^{\mathrm{T}}$$

$$\mathbf{H}_{L+1}^{-1} = \mathbf{H}_L^{-1} - \frac{\mathbf{H}_L^{-1}\mathbf{b}_{L+1}\mathbf{b}_{L+1}^{\mathrm{T}}\mathbf{H}_L^{-1}}{1 + \mathbf{b}_{L+1}^{\mathrm{T}}\mathbf{H}_L^{-1}\mathbf{b}_{L+1}}$$ by Woodbury Identity

- Set initial matrix $\mathbf{H}_0 := \alpha\mathbf{I}$
- We can find Hessian matrix by using numerical differentiation (finite differences, central differences).

Exact Evaluation of Hessian Matrix

- Hessian can be evaluated exactly using an extension for backpropagation
- Denote: $\delta_k = \dfrac{\partial E_n}{\partial a_k} \qquad M_{kk'} \equiv \dfrac{\partial^2 E_n}{\partial a_k \partial a_{k'}}$



1. Both weights in first layer:

$$\frac{\partial^2 E_n}{\partial w_{kj}^{(2)} \partial w_{k'j'}^{(2)}} = z_j z_{j'} M_{kk'}$$

2. Both weights in second layer:

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{j'i'}^{(1)}} = x_i x_{i'} h''(a_{j'}) I_{jj'} \sum_k w_{kj'}^{(2)} \delta_k + x_i x_{i'} h'(a_{j'}) h'(a_j) \sum_k \sum_{k'} w_{k'j'}^{(2)} w_{kj}^{(2)} M_{kk'}$$

3. One weight in each layer:

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{kj'}^{(2)}} = x_i h'(a_{j'}) \left\{ \delta_k I_{jj'} + z_j \sum_{k'} w_{k'j'}^{(2)} H_{kk'} \right\}$$

5

Regularization in Neural Networks

Control Model Complexity

Regularization

- 이 장에서의 '정규화'란 모델의 복잡도를 낮추는 것을 의미한다
- 또한 모델의 복잡도는 파라미터의 크기, 모델의 민감도(입력 데이터 혹은 파라미터 변화에 대한) 그리고 모델의 크기(파라미터의 수, 히든 유닛의 수 등) 등에 좌우된다
- 파라미터의 크기 조절: weight decay, early stopping
- 모델의 민감도 조절: Tangent propagation, Training with transformed data, Convolutional Neural Network
- 모델의 크기: Soft weight sharing

5-1

Regularization in Neural Networks

Reduce Parameter Size: Weight decay & Early Stopping

Regularization

- 이 장에서의 '정규화'란 모델의 복잡도를 낮추는 것을 의미한다
- 또한 모델의 복잡도는 파라미터의 크기, 모델의 민감도(입력 데이터 혹은 파라미터 변화에 대한) 그리고 모델의 크기(파라미터의 수, 히든 유닛의 수 등) 등에 좌우된다
- 파라미터의 크기 조절: weight decay, early stopping
- 모델의 민감도 조절: Tangent propagation, Training with transformed data, Convolutional Neural Network
- 모델의 크기: Soft weight sharing

5-1

Regularization in Neural Networks

Reduce Parameter Size: Weight decay & Early Stopping

Weight Decay

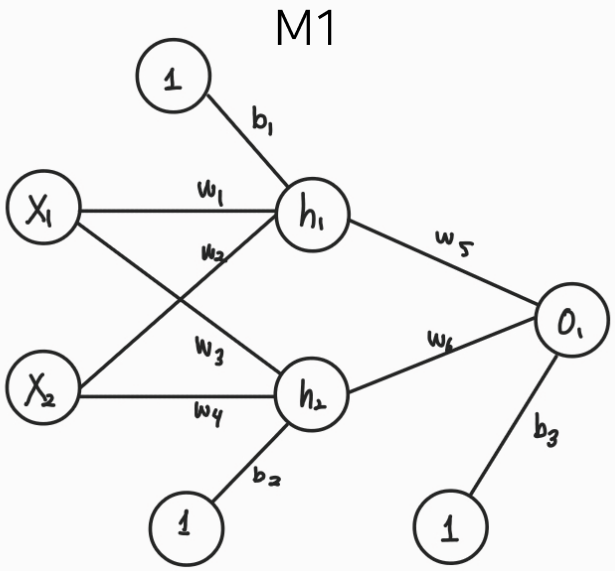
- $\tilde{E}(w) = E(w) + \frac{\lambda}{2} w^T w$
- 회귀에서 ridge와 비슷한 개념. 뉴럴 네트워크에서 사용되면 weight decay라고 불리고 regression에서 사용되면 ridge라고 불림
- $\frac{\lambda}{2} w^T w$는 베이지안 관점에서 생각했을 때 negative log Gaussian prior로 해석될 수 있다.
  - $p(w) = N(w|0, \lambda^{-1} I) \propto \exp\left(-\frac{\lambda}{2} w^T w\right)$
  - $-\ln p(w) \propto \frac{\lambda}{2} w^T w$
  - 그렇다고 weight decay를 사용하는 뉴럴 네크워크 모델이 베이지안 뉴럴 네트워크라는 것은 아님!
  - 나중에 베이지안 뉴럴 네트워크로 확장했을 때, 그렇게 해석할 수 있다는 것

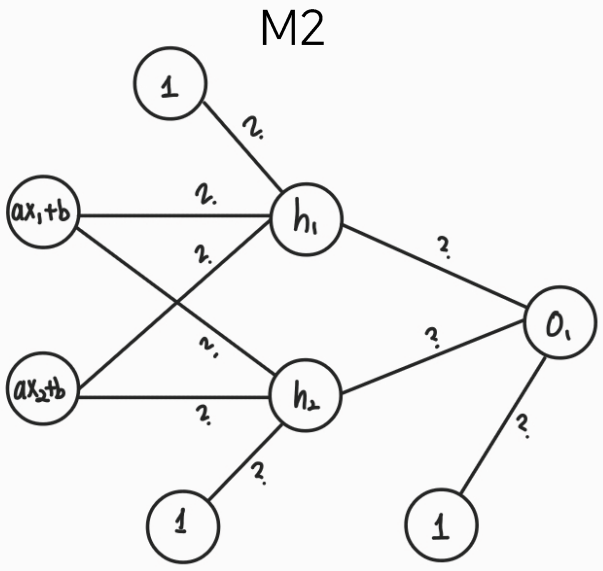
Weight Decay

- Weight decay를 정규화항으로 사용한 뉴럴 네트워크는 입력과 출력의 선형 변환에 따라 모델의 성능이 달라진다는 문제가 있다
- 즉, weight decay는 inconsistent한 모델을 만든다는 단점이 존재한다!
- 왜 이러한 단점이 발생하는지 살펴보기 위해 선형 변환에 consistent한 모델을 고려해보자

Consistent Neural Network

- 왼쪽의 뉴럴 네트워크에서 Square Error function을 사용하여 파라미터를 구했다고 하자
- 그 다음 입력값에 모두 동일하게 a를 곱하고 b를 더했다.
  - $x_1 \rightarrow a * x_1 + b$
  - $x_2 \rightarrow a * x_2 + b$
- 입력값을 그대로 사용하여 구한 모델을 M1, 입력값에 선형 변환을 한 후 구한 모델을 M2라고 하자
- 모델이 입력값의 선형 변환에 Consistent하다는 것은, M1과 M2의 성능이 동일하다는 것을 의미한다. 그리고 두 모델의 성능이 동일하다는 것은 모든 입력에 대해서 M1과 M2의 error가 동일하다는 것을 의미한다

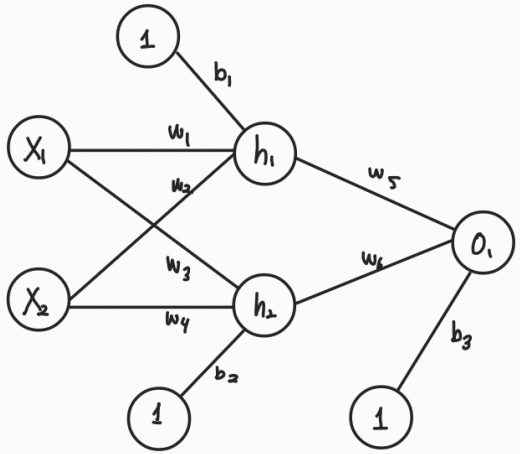


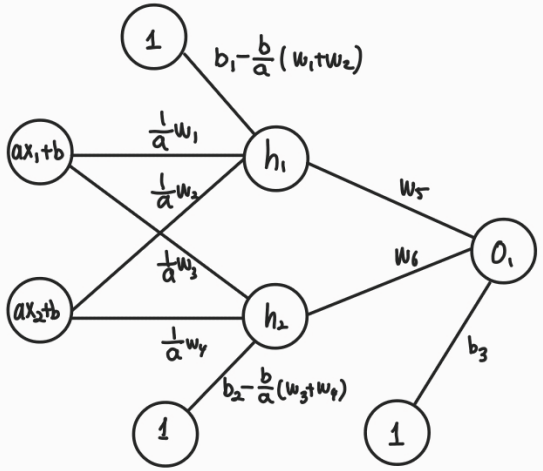


- 두 모델이 학습하는 파라미터의 값들은 다를 것이다.
- 그러나 error가 같으므로 출력값은 같을 것이다

Consistent Neural Network

- 만약 M2에서 hidden layer에 들어가는 값이 M1과 같다면, 두 번째 계층의 파라미터들은 수정하지 않더라도 최종적으로 같은 결과를 보일 것이다.
- 이를 달성하기 위해 다음과 같이 1계층의 파라미터들을 조정할 수 있다:
  - $w_i \rightarrow \frac{1}{a} w_i \ , \ i = 1, 2, 3, 4$
  - $b_1 \rightarrow b_1 - \frac{b}{a}(w_1 + w_2)$
  - $b_2 \rightarrow b_2 - \frac{b}{a}(w_3 + w_4)$
- 출력 변수에 선형 변환을 해 주는 경우에는 2계층의 가중치들을 적절하게 조정해줌으로써 모델을 consistent하게 만들어 줄 수 있다.
- 물론 데이터를 변환했을 때, 꼭 파라미터들이 위와 같은 형태로 변환되지는 않을 수도 있다. 예를 들어서 hidden unit의 순서가 바뀔 수도 있다. 하지만 결과적으로 모델의 성능은 동일하게 된다





Inconsistency of weight decay

- Weight decay에서는 Consistent NN에서처럼 계층별로 파라미터를 구분 지을 수가 없다. 왜냐하면 Regularization hyperparameter λ가 모든 파라미터들에 constrain를 걸어 버리기 때문이다
- λ를 높여서 파라미터들에 대해 더 많은 constraint를 걸거나, 낮춰서 constrain를 완화하는 방식으로 파라미터들을 조절해야 하는데, 이 경우 앞선 모델처럼 1계층(혹은 2계층)만 타겟팅해서 조절하는 것이 불가능하다.
- 따라서 입력의 선형 변환에 따라 모델이 inconsistent하게 되고, 어떠한 입력에 대해선 모델이 더 좋은 성능을 보이게 된다.

Consistent Weight Decay

- Weight decay에서의 문제점은 계층 별로 따로 구분 지을 수 없기 때문에 발생했다. 그렇다면 해결책은 계층 별로 따로 파라미터를 조정할 수 있도록 Weight decay를 수정해 주면 된다
- $\frac{\lambda_1}{2}\sum_{w\in W_1} w^2 + \frac{\lambda_2}{2}\sum_{w\in W_2} w^2$
- 여기서 W_1 은 1계층의 가중치만을 포함하고 있고 W_2 는 2계층의 가중치만을 포함하고 있다
- 하지만 이러한 regularizer는 베이지안 관점으로 확장해서 생각하기가 어렵다.

- 정리하자면, 단순 weight decay는 파라미터의 사이즈를 줄일 수 있는 가장 일반적인 방법론이지만, 선형 변환에 inconsistent하다는 단점이 있다. 이를 보완하기 위해 계층별 weight decay를 생각해 볼 수 있다. 하지만 이러한 방법은 선형 변환에 consistent하기는 해도 베이지안 관점으로 확장시키기 어렵다는 단점이 있다.

Early Stopping

- 파라미터의 사이즈를 조절하는 또 다른 방법으로는 '조기 종료'가 있다.
- 뉴럴 네트워크는 iterative optimization method를 이용해서 파라미터를 fitting한다. 이러한 방법론은 훈련 데이터에 대해선 Error를 계속 감소시켜 나간다
- 하지만 validation set에 대해선 error가 초반엔 감소했다고 점차 overfitting되어 증가하기 시작한다.
- 따라서 validation set의 error가 가장 작은 지점에서 iteration을 종료시키는 것이 더 좋을 수도 있다.
- 이러한 방식으로 학습을 조기에 종료하는 방법론이 바로 Early Stopping이다.

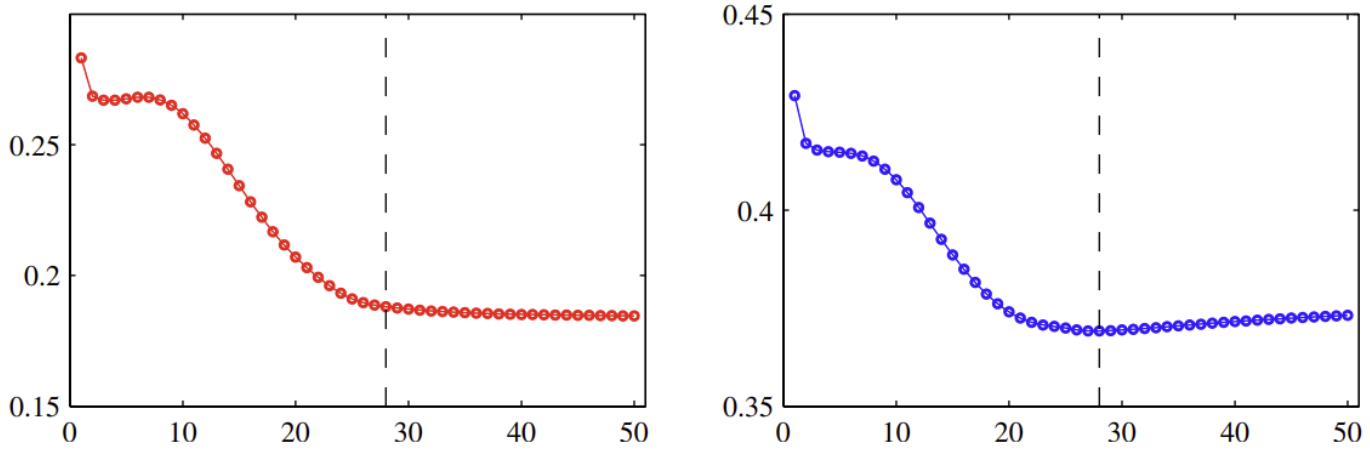
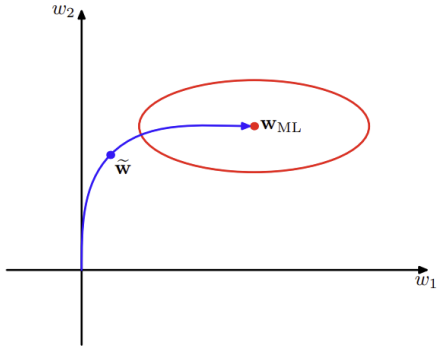


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

Figure 5.13 A schematic illustration of why early stopping can give similar results to weight decay in the case of a quadratic error function. The ellipse shows a contour of constant error, and $\mathbf{w}_{\mathrm{ML}}$ denotes the minimum of the error function. If the weight vector starts at the origin and moves according to the local negative gradient direction, then it will follow the path shown by the curve. By stopping training early, a weight vector $\widetilde{\mathbf{w}}$ is found that is qualitatively similar to that obtained with a simple weight-decay regularizer and training to the minimum of the regularized error, as can be seen by comparing with Figure 3.15.



- Early Stopping은 Weight Decay를 사용한 정규화와 비슷한 습성을 보인다.
- $\lambda = \frac{1}{\tau\eta}$; τ: *iteration index*, η: *learning rate*

5-2

Regularization in Neural Networks

불변성을 학습시켜 모델의 민감도를 낮추자

Invariances

- 이제부터는 모델의 민감도를 낮추는 방법에 대해서 알아볼 것이다
- 이미지 분류 모델을 예로 들어 보면, 똑같은 그림이 약간 이동하거나 회전했다고 해서 분류를 다르게 해서는 안된다. 즉 특정한 변화에 대해서는 불변(invariant) 하도록 민감도를 낮추는 방법에 대해 알아볼 것이다. 총 4가지 방법이 있다

1. 만족시켜야 할 불변성을 바탕으로 훈련 데이터를 변환하여 훈련 데이터셋에 추가한다. 예를 들어 이미지 분류 문제의 경우 각 훈련 데이터에 대해서 위치가 변형된 이미지를 만들어서 훈련 데이터셋에 추가적으로 포함시킨다
2. 입력이 변환되었을 때 모델 출력값이 변하게 되는 것에 대하여 불이익을 주는 Regularization term을 추가한다. 이를 적용한 것이 바로 Tangent Propagation이다
3. 해당 변환에 대해서 불변하는 특징들을 추출하는 방식으로 불변성을 사전 처리 과정에 포함시킨다.
4. 뉴럴 네트워크의 구조에 불변성을 포함시킨다. Convolutional Neural Network가 이러한 방법을 적용한 모델이다.

먼저 2번 방법을 활용한 Tangent Propagation부터 살펴보자

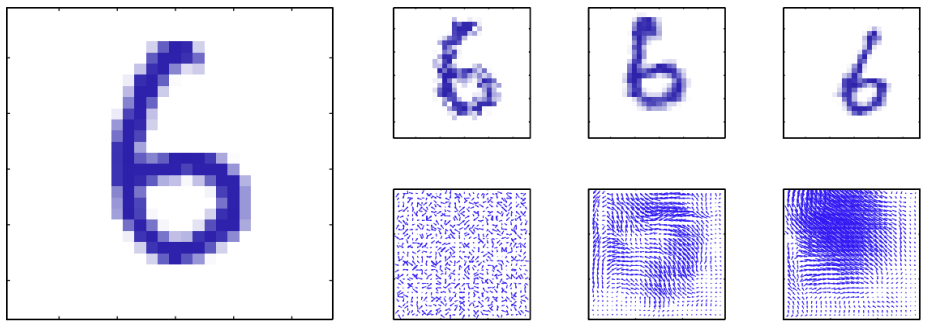


Figure 5.14 Illustration of the synthetic warping of a handwritten digit. The original image is shown on the left. On the right, the top row shows three examples of warped digits, with the corresponding displacement fields shown on the bottom row. These displacement fields are generated by sampling random displacements $\Delta x, \Delta y \in (0, 1)$ at each pixel and then smoothing by convolution with Gaussians of width 0.01, 30 and 60 respectively.

Tangent Propagation

- Notation
  - ζ: 변환 매개변수
  - $s(x_n, \zeta) \rightarrow x_n$에 변환을 적용했을 때의 벡터
  - $s(x_n, 0) = x_n \rightarrow$ 변환을 적용하지 않았을 때는 기존의 벡터와 동일
  - $\tau = \frac{\partial s}{\partial \zeta} \rightarrow$ 방향 도함수
  - $\tau_n = \frac{\partial s(x_n, \zeta)}{\partial \zeta}|_{\zeta=0}$

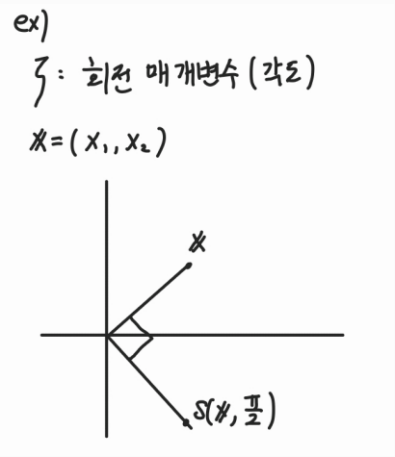
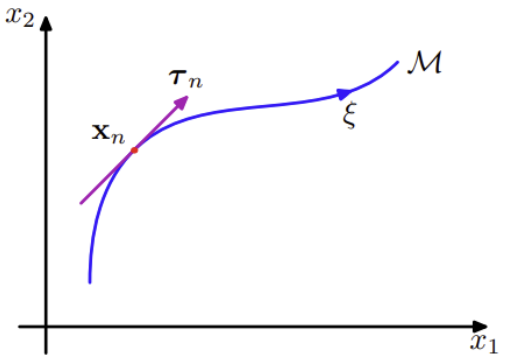


Figure 5.15 Illustration of a two-dimensional input space showing the effect of a continuous transformation on a particular input vector $\mathbf{x}_n$. A one-dimensional transformation, parameterized by the continuous variable ξ, applied to $\mathbf{x}_n$ causes it to sweep out a one-dimensional manifold $\mathcal{M}$. Locally, the effect of the transformation can be approximated by the tangent vector $\boldsymbol{\tau}_n$.



- 파란색 곡선은 단일 점 x_n에 대해 모든 가능한 변환을 적용했을 때의 점들을 나타낸 것이다.
- Tangent Propagation에서는 보라색 선으로 나타나 있는 순간적인 변화에 대해 관심을 가진다.
- 순간적인 변화량을 바탕으로 입력 데이터에 미세한 변화가 있었을 때 출력값이 어떻게 변하는지를 추적한 뒤, 그것에 페널티를 줌으로써 변화에 대한 민감도를 낮추는 것이다.

Tangent Propagation

- Tangent Propagation에서 Regularization term을 유도할 때 Jacobian 행렬 개념이 나온다.
- $y = [y_1, \dots, y_K]$ → 출력값은 K 차원
- $x = [x_1, \dots, x_D]$ → 입력은 D차원

- $J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_K}{\partial x_1} & \cdots & \frac{\partial y_K}{\partial x_D} \end{bmatrix}$ K x D matrix (matrix of functions of x)

- $J_n = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_K}{\partial x_1} & \cdots & \frac{\partial y_K}{\partial x_D} \end{bmatrix}\Big|_{x=x_n}$ → 샘플의 개수만큼 Jacobian이 만들어짐. 이제는 함수에 대한 행렬이 아닌 실수값들을 원소로 가짐(n = 1, ⋯, N)
- Jacobian 행렬의 의미: 기본적으로 y는 x에 대한 비선형 함수지만, 국소적인 부분만을 보았을 때는 선형적이라고 할 수 있다. Jacobian matrix는 국소적으로 선형변환이라고 했을 때, x의 변화에 대해 y가 얼마나 변하는지를 나타낸다
  - $J_n u \approx f(x_n + u) - f(x_n)$

Tangent Propagation

$$\left.\frac{\partial y_k}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{D} \frac{\partial y_k}{\partial x_i} \frac{\partial x_i}{\partial \xi}\bigg|_{\xi=0} = \sum_{i=1}^{D} J_{ki}\tau_i$$

- Jacobian의 k번째 행과 탄젠트 벡터(국소적 변화)의 곱

$$\Omega = \frac{1}{2}\sum_n \sum_k \left(\left.\frac{\partial y_{nk}}{\partial \xi}\right|_{\xi=0} \right)^2 = \frac{1}{2}\sum_n \boxed{\sum_k \left(\sum_{i=1}^{D} J_{nki}\tau_{ni} \right)^2}.$$

- 빨간색 부분만 보면 $||J_n\tau_n||^2$
  - 즉 x_n에 매우 작은 변화(τ_n)를 줬을 때, 출력값의 변화를 하나의 값으로 수치화한 것
- 데이터별로 작은 변화를 줬을 때 출력값의 변화를 모두 다 합친 것
- 이것을 Regularization term으로 활용하겠다는 것은 국소적 변화에 따른 출력값의 변화를 억제하겠다는 것!

$$\widetilde{E} = E + \lambda\Omega$$

- 오류 함수

Training with transformed data

- 이제 앞서 보았던 1번 방법에 대해 알아보자.
- 변환된 데이터를 만들어내기 위해 매개변수 ζ를 확률변수로 간주하자 → $p(\zeta)$
  - ζ는 데이터들과는 독립
  - Assume Zero-mean and low variance (즉 이 분포에서 값을 뽑을 다음 데이터를 변환해도, 미세한 변화만을 보일 확률이 높다)
- 만약에 무한히 많은 데이터들이 어떠한 분포를 따라서 존재한다면, 우리는 오류 함수를 다음과 같이 기댓값의 형태로 나타낸다(이때, 오류 함수는 Sum-of-squares error function)

$$E = \frac{1}{2} \iint \{y(\mathbf{x}) - t\}^2 p(t|\mathbf{x}) p(\mathbf{x}) \, \mathrm{d}\mathbf{x} \, \mathrm{d}t$$

- 이제 각 데이터들에 대해서 무한대의 복사본을 만든다고 해 보자. 이때 각각의 복사본들은 분포 $p(\zeta)$에서 추출한 매개변수 ζ에 의해 섭동할 것이다.
- 그러면 추가된 데이터를 포함한 데이터 집합에 대한 오류 함수를 다음과 같이 적을 수 있다.

$$\widetilde{E} = \frac{1}{2} \iiint \{y(\mathbf{s}(\mathbf{x}, \xi)) - t\}^2 p(t|\mathbf{x}) p(\mathbf{x}) p(\xi) \, \mathrm{d}\mathbf{x} \, \mathrm{d}t \, \mathrm{d}\xi.$$

- $\zeta = 0$일 때는 $s(x, \zeta) = x$이므로 당연히 위의 오류 함수는 기존의 데이터도 포함하는 함수이다.

Training with transformed data

- Expand $s(x,\zeta)$ as a Taylor series in powers of ζ

$$
\begin{aligned}
\mathbf{s}(\mathbf{x},\xi) &= \mathbf{s}(\mathbf{x},0) + \xi \left.\frac{\partial}{\partial \xi}\mathbf{s}(\mathbf{x},\xi)\right|_{\xi=0} + \frac{\xi^2}{2}\left.\frac{\partial^2}{\partial \xi^2}\mathbf{s}(\mathbf{x},\xi)\right|_{\xi=0} + O(\xi^3) \\
&= \mathbf{x} + \xi\boldsymbol{\tau} + \frac{1}{2}\xi^2\boldsymbol{\tau}' + O(\xi^3)
\end{aligned}
$$

- 이를 활용하여 $y(s(x,\zeta))$를 나타내면 다음과 같다:

$$
y(\mathbf{s}(\mathbf{x},\xi)) = y(\mathbf{x}) + \xi\boldsymbol{\tau}^{\mathrm{T}}\nabla y(\mathbf{x}) + \frac{\xi^2}{2}\left[(\boldsymbol{\tau}')^{\mathrm{T}}\nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}}\nabla\nabla y(\mathbf{x})\boldsymbol{\tau}\right] + O(\xi^3).
$$

- 최종적으로 $\tilde{E}$를 나타내면 다음과 같다:

$$
\begin{aligned}
\widetilde{E} &= \frac{1}{2}\iint \{y(\mathbf{x}) - t\}^2 p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t \\
&+ \mathbb{E}[\xi]\iint \{y(\mathbf{x}) - t\}\boldsymbol{\tau}^{\mathrm{T}}\nabla y(\mathbf{x})p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t \\
&+ \mathbb{E}[\xi^2]\iint \left[\{y(\mathbf{x}) - t\}\frac{1}{2}\left\{(\boldsymbol{\tau}')^{\mathrm{T}}\nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}}\nabla\nabla y(\mathbf{x})\boldsymbol{\tau}\right\}\right. \\
&\quad \left. + \left(\boldsymbol{\tau}^{\mathrm{T}}\nabla y(\mathbf{x})\right)^2\right] p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t + O(\xi^3).
\end{aligned}
$$

Training with transformed data

- $E[\zeta] = 0, Var[\zeta] = E[\zeta^2] = \lambda$

$$
\begin{aligned}
\widetilde{E} &= \frac{1}{2}\iint \{y(\mathbf{x}) - t\}^2 p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t \\
&+ \mathbb{E}[\xi]\iint \{y(\mathbf{x}) - t\}\boldsymbol{\tau}^{\mathrm{T}}\nabla y(\mathbf{x})p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t \\
&+ \mathbb{E}[\xi^2]\iint \left[\{y(\mathbf{x}) - t\}\frac{1}{2}\left\{(\boldsymbol{\tau}')^{\mathrm{T}}\nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}}\nabla\nabla y(\mathbf{x})\boldsymbol{\tau}\right\}\right. \\
&\quad \left. + \left(\boldsymbol{\tau}^{\mathrm{T}}\nabla y(\mathbf{x})\right)^2\right] p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t + O(\xi^3).
\end{aligned}
$$

$$\widetilde{E} = E + \lambda\Omega$$

$$
\begin{aligned}
\Omega &= \int \left[\{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}\frac{1}{2}\left\{(\boldsymbol{\tau}')^{\mathrm{T}}\nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}}\nabla\nabla y(\mathbf{x})\boldsymbol{\tau}\right\}\right. \\
&\quad \left. + \left(\boldsymbol{\tau}^{T}\nabla y(\mathbf{x})\right)^2\right] p(\mathbf{x})\,\mathrm{d}\mathbf{x}
\end{aligned}
$$

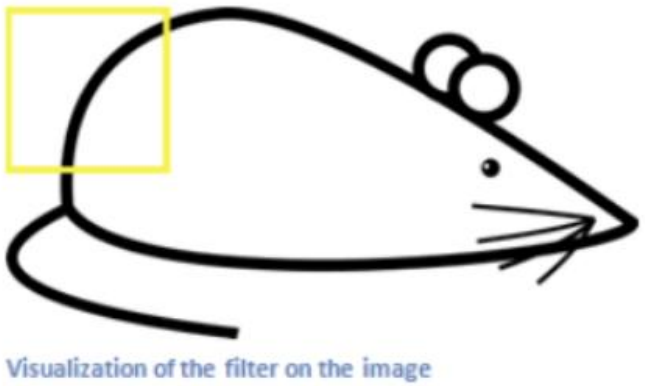
- Regularization term을 더 정리하면 다음과 같이 나타낼 수 있음

$$\Omega = \frac{1}{2}\int \left(\boldsymbol{\tau}^{T}\nabla y(\mathbf{x})\right)^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

- 이는 tangent propagation에서의 regularization term과 동일함
- 단 앞서서는 유한한 개수의 데이터가 있는 경우였고, 여기는 무한한 수의 데이터가 있는 경우
- 또한 앞서서는 출력값이 총 K개 있는 경우였고, 여기는 출력값이 하나
- 따라서 식의 형태에 약간의 차이가 있음. 단 본질은 동일

Convolutional Networks

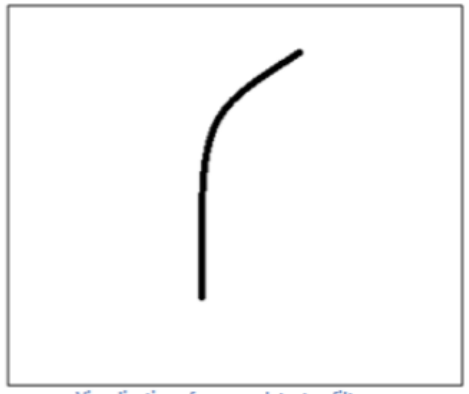
- 이제 뉴럴 네트워크의 구조에 불변성을 포함시키는 방법에 대해 살펴보자
- CNN에서는 filter 라는 것을 사용하여 어떠한 이미지의 특징을 탐지해 낸다.
- 예를 들어서 다음과 같은 쥐 이미지를 식별하려고 할 때, 다음의 filter를 사용할 수 있다.



Visualization of the filter on the image

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

박재우 교수님 강의안

- 이때 쥐가 수직, 수평 이동을 하더라도, original data에 동일한 패턴이 있는 경우, filter는 그것을 포착할 수 있다. 즉 CNN 자체가 구조에 불변성을 포함하고 있는 것이다
- 실제 CNN에서는 우리가 filter를 지정하지는 않는다. Filter에 있는 모든 원소들을 파라미터로 간주하고 Backpropagation을 통해 이를 fitting시킨다

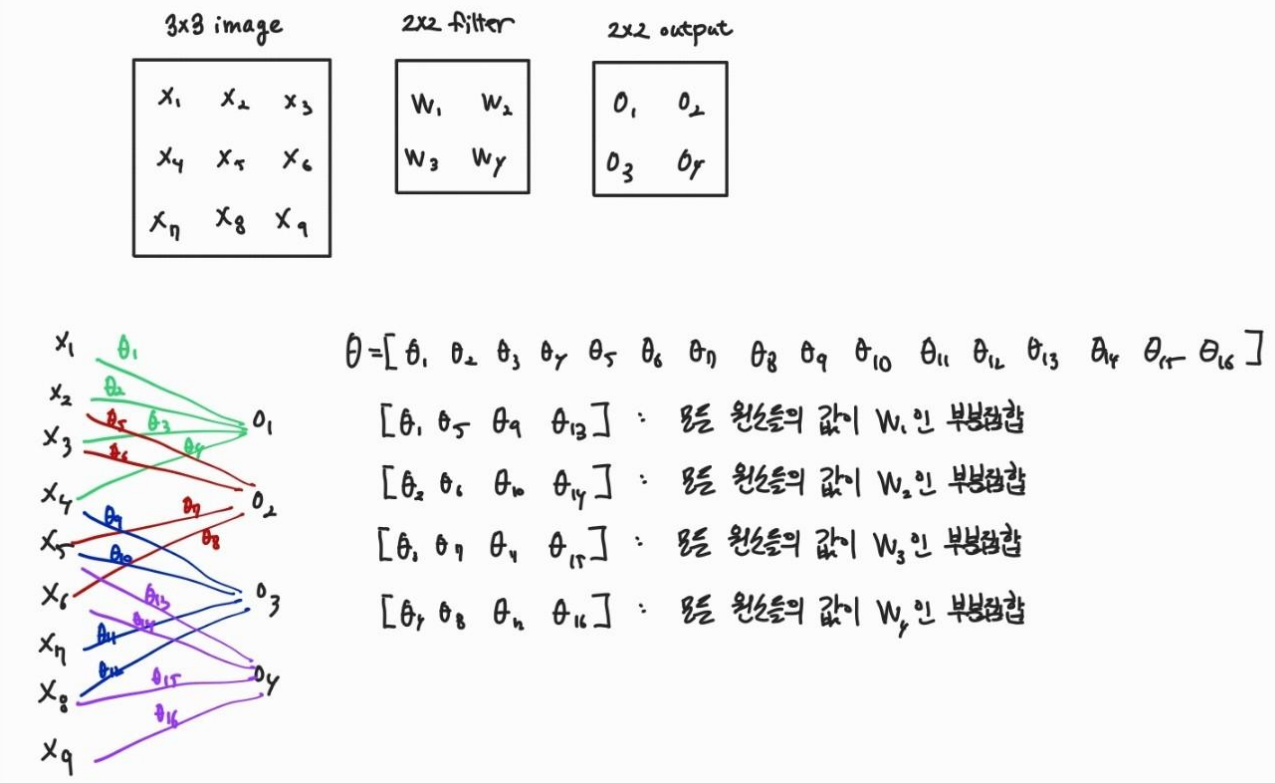
5-3

Regularization in Neural Networks

파라미터의 개수 줄이기: 유효 파라미터의 개수를 줄이자

Weight Sharing

- 이제 정규화의 마지막 부분 '모델의 크기'를 조절하는 법에 대해 살펴보자
- 가중치 공유란 전체 파라미터 집합 중에 특정한 부분 집합에 해당하는 것들은 동일한 값을 가지는 것이다. 특정한 부분 집합이 모두 동일한 값을 가지면 실질적인 파라미터의 수, 즉 유효 복잡도는 감소하게 된다.
- 사실 CNN에서도 가중치 공유가 사용되었다.
- 하지만 가중치 공유 기법은 제약 조건을 미리 명시할 수 있는 특정 문제들에 대해서만 적용할 수 있다. 따라서 일반적으로 활용하기는 쉽지 않다
- 대신에 파라미터들이 동일해야 한다는 강한 제약 조건을, 파라미터들이 비슷하도록 권장하는 정규화의 형태로 변형시킨 "Soft weight sharing" 기법을 더 흔히 활용한다.



Soft weight sharing

- 만약 파라미터들의 사전 분포가 다음과 같다면, 파라미터들이 일정한 그룹을 형성하고 있다고 생각할 수 있다.
- 따라서 파라미터의 사전 분포를 다음과 같은 가우시안 혼합 분포로 설정을 하고, 이로부터 Regularization term을 구하게 되면, 파라미터들이 일정한 군집을 이루도록 제한을 줄 수 있다.

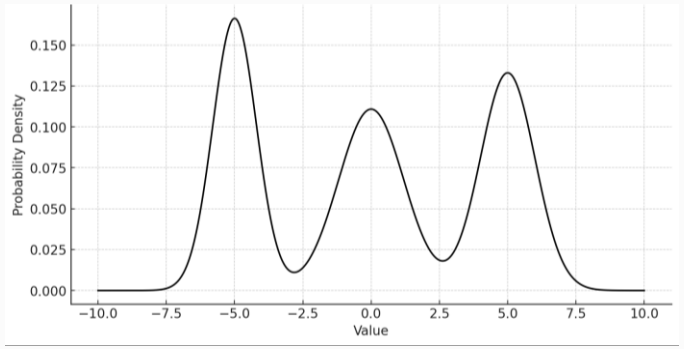


Image from ChatGPT

$$p(\mathbf{w}) = \prod_i p(w_i)$$

$$p(w_i) = \sum_{j=1}^{M} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2)$$

- 앞서 보았던 consistent weight decay와 파라미터들을 그룹으로 나눈다는 점에서 유사하지만, consistent weight decay의 경우 평균(μ)과 혼합 비율(π)를 지정해 주지 않았다는 점에서 차이가 존재한다. Weight decay의 경우 목적성이 파라미터의 크기를 조절하는 것에만 있었기 때문이다. 정규화항을 만들어내는 사전 함수의 형태는 비슷할지라도 목적성은 엄밀히 다르다.
- 전체 Error function과 Regularization term은 다음과 같다

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda\Omega(\mathbf{w})$$

$$\Omega(\mathbf{w}) = -\sum_i \ln\left(\sum_{j=1}^{M} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2)\right)$$

- 여기서 정규화항은 negative log prior이다.

Derivatives of the total error function

$$\frac{\partial \widetilde{E}}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda \sum_j \gamma_j(w_i) \frac{(w_i - \mu_j)}{\sigma_j^2}.$$

$$\frac{\partial \widetilde{E}}{\partial \mu_j} = \lambda \sum_i \gamma_j(w_i) \frac{(\mu_i - w_j)}{\sigma_j^2}$$

$$\frac{\partial \widetilde{E}}{\partial \sigma_j} = \lambda \sum_i \gamma_j(w_i) \left(\frac{1}{\sigma_j} - \frac{(w_i - \mu_j)^2}{\sigma_j^3} \right)$$

$$\frac{\partial \widetilde{E}}{\partial \eta_j} = \sum_i \{\pi_j - \gamma_j(w_i)\}.$$

$$\pi_j = \frac{\exp(\eta_j)}{\sum_{k=1}^{M} \exp(\eta_k)}.$$

(Softmax)

- Error function의 미분값들의 형태로부터 정규화항의 효과를 이해해볼 수 있다.
- 세 종류의 derivatives에 공통적으로 $\gamma_j(w_i)$라는 부분이 나오는데, 이를 이해해야 가중치들이 어떻게 업데이트되는지 직관적으로 이해할 수 있다.
- $\gamma_j(w_i)$는 responsibility라는 개념으로 PRML 2.3장에 소개되어 있다

Responsibility

- Gaussian Mixture distribution에 대한 pdf는 다음과 같이 나타낼 수 있다: $p(w) = \sum_{k=1} \pi_k N(w|\mu_k, \Sigma_k)$
- $\pi_k = p(k)$를 k번째 가우시안을 뽑을 사전 확률이라고 하자
- 그러면 사후 확률 $p(k|w)$를 responsibility라고 부르며 다음과 같이 나타낸다
  - $\gamma_j(w) \equiv p(k|w) = \frac{\pi_j N(w|\mu_j, \Sigma_j)}{\sum_k \pi_k N(w|\mu_k, \Sigma_k)}$
  - 직관적으로는, w가 주어졌을 때, 그것이 k번째 가우시안에서 나왔을 확률이다
- 이를 바탕으로 Error function의 미분값에 대해 이해해보자

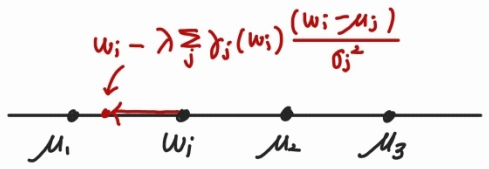
Derivatives of the total error function wrt weights

$$\frac{\partial \widetilde{E}}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda \sum_j \gamma_j(w_i) \frac{(w_i - \mu_j)}{\sigma_j^2}.$$

Gradient descent

$$w_i^{(new)} = w_i - \frac{\partial \widetilde{E}}{\partial w_i}$$

$$= w_i - \frac{\partial E}{\partial w_i} - \lambda \sum_j \gamma_j(w_i) \frac{(w_i - \mu_j)}{\sigma_j^2} \qquad (\text{assume } j=1,2,3)$$



If $\gamma_1(w_i) \gg \gamma_2(w_i), \gamma_3(w_i)$

Then $\sum_j \gamma_j(w_i) \frac{(w_i - \mu_j)}{\sigma_j^2} \approx \gamma_1(w_i) \frac{(w_i - \mu_j)}{\sigma_1^2} > 0$

- 결론: 가장 responsibility가 높은 μ_j의 방향으로 가중치의 이동이 조정됨. 물론 $\frac{\partial E}{\partial w_i}$ 부분에 의해 실제 방향은 알 수 없겠지만, E(w)만을 Error function으로 사용하는 모델에 비해, 정규화항이 추가된 모델의 가중치가 responsibility가 높은 μ_j의 방향으로 constraint가 걸린다는 것을 확인할 수 있음
- 각 가중치들은 responsibility가 높은 방향으로 이동하도록 constraint이 걸려 있으므로, 가중치들이 일정한 군집을 이루도록 학습된다는 것을 확인할 수 있다(soft weight sharing)

Derivatives of the total error function wrt μ

$$\frac{\partial \widetilde{E}}{\partial \mu_j} = \lambda \sum_i \gamma_j(w_i) \frac{(\mu_i - w_j)}{\sigma_j^2}$$

- 반대로 μ_j의 경우 가중치로 당겨지는 것을 확인할 수 있다
- 평균이 가중치를 당기로, 가중치가 평균을 당기는 구조다.

Derivatives of the total error function wrt σ

$$\frac{\partial \widetilde{E}}{\partial \sigma_j} = \lambda \sum_i \gamma_j(w_i) \left(\frac{1}{\sigma_j} - \frac{(w_i - \mu_j)^2}{\sigma_j^3} \right)$$

- σ에 대한 미분값은 다음과 같이 나타난다.
- 하지만 보통 σ를 양수로 학습하기 위해 σ자체를 미분하기보다 보조변수 $\{\nu_j\}$를 도입하여 $\sigma_j^2 = \exp(\nu_j)$로 설정한 후 ν_j에 대해 minimization을 진행한다.

Derivatives of the total error function wrt π

- 혼합 계수 π_j에 대해 미분할 경우에는 다음의 제약 조건을 고려해야 한다

$$\sum_j \pi_j = 1, \qquad 0 \leqslant \pi_i \leqslant 1$$

- 이 경우 보조 변수 $\{\eta_j\}$와 소프트맥스 함수를 이용해서 제약 조건을 만족시킬 수 있다

$$\pi_j = \frac{\exp(\eta_j)}{\sum_{k=1}^{M} \exp(\eta_k)}.$$

- 이 경우 Regularized Error function의 $\{\eta_j\}$에 대한 미분은 다음의 형태를 취한다

$$\frac{\partial \widetilde{E}}{\partial \eta_j} = \sum_i \{\pi_j - \gamma_j(w_i)\}.$$

- 여기서 π_j가 j 성분의 responsibility들의 평균을 향해 가까워진다는 것을 알 수 있다.

6

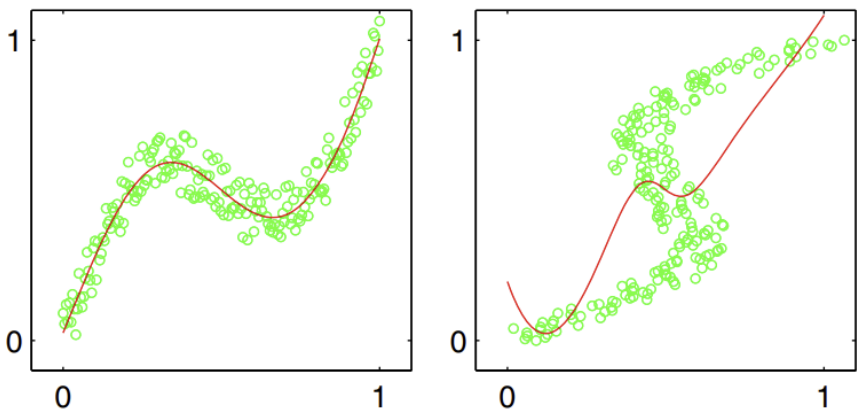
Mixture Density Networks

How to deal with 'Multimodal' conditional distribution $p(t|x)$

Mixture Density Network

- 지도 학습의 목표는 조건부 분포 $p(t|x)$를 모델하는 것이다.
- 이때 대부분의 단순한 회귀 문제에서는 조건부 분포로 가우시안 분포를 사용한다.
- 하지만 조건부 분포가 가우시안이 아닌 Multimodal 분포일 경우, 가우시안으로 조건부 분포를 가정하게 되면 예측 성능이 매우 좋지 않을 수 있다.

Figure 5.19 On the left is the data set for a simple 'forward problem' in which the red curve shows the result of fitting a two-layer neural network by minimizing the sum-of-squares error function. The corresponding inverse problem, shown on the right, is obtained by exchanging the roles of x and t. Here the same network trained again by minimizing the sum-of-squares error function gives a very poor fit to the data due to the multimodality of the data set.

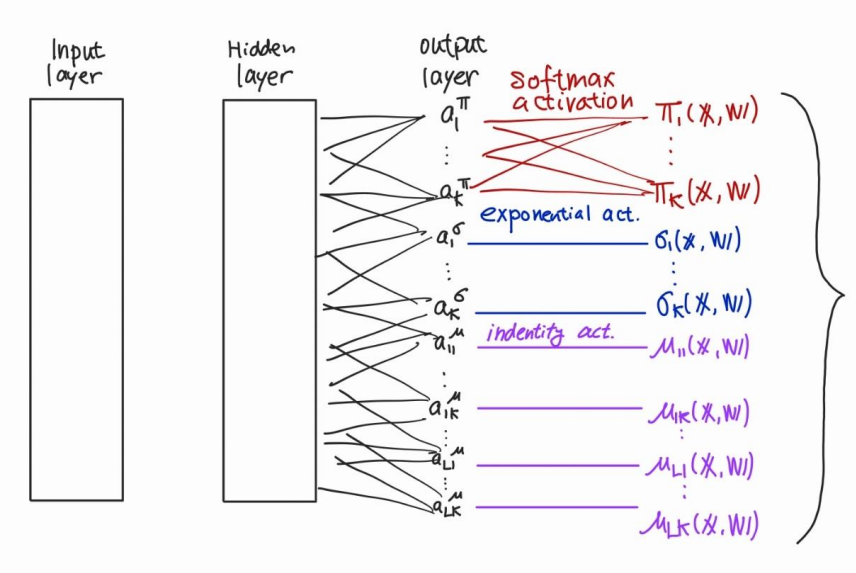


- 따라서 조건부 분포를 가우시안 혼합 분포로 설정을 해 줄 것이다.
  - $p(t|x) = \sum_{k=1}^{K} \pi_k(x) N(t|\mu_k(x), I\sigma_k^2(x))$

- 조건부 분포에 어떠한 혼합 분포(가우시안 혼합 분포, 베르누이 혼합 분포 등)를 사용한 네트워크를 바로 Mixture Density Network라고 부른다
- 참고로 앞서 soft weight sharing에서 보았던 가우시안 혼합 분포는 파라미터의 prior 분포를 그렇게 설정한 것이다. 어떤 분포를 가우시안 혼합 분포로 설정을 했는지 잘 구분하자

Mixture Density Network

- $p(t|x) = \sum_{k=1}^{K} \pi_k(x) N(t|\mu_k(x), I\sigma_k^2(x))$
- 이러한 조건부 분포를 가지는 모델의 출력의 개수는?
  - 가우시안 혼합 분포가 K개의 가우시안으로 이루어져 있으면, 혼합 계수 $\pi_k(x)$를 결정하는 출력 유닛 a_k^{π}가 총 K개 존재한다
  - 또한 K개의 서로 다른 평균 벡터와 공분산 행렬이 있는데, 평균 벡터와 공분산 행렬의 크기는 t의 크기에 좌우된다.
  - t의 크기가 L이라면, 벡터 $\mu_k(x)$의 성분 $\mu_{kj}(x)$를 결정하는 L x K 개의 출력 유닛 a_k^{μ}가 필요하다
  - 마지막으로 공분산 행렬을 대각 원소가 모두 같은 대각 행렬로 가정한다면, $\sigma_k(x)$를 결정하는 K개의 출력 유닛 a_k^{σ}가 필요하다
  - 즉 총 출력값의 개수는 (L+2)*K
  - 타깃 변수의 조건부 평균만을 예측하는 일반 네트워크가 L개의 출력값을 가지는 것과 대조된다



$$E(\mathbf{w}) = -\sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{k} \pi_k(\mathbf{x}_n, \mathbf{w}) \mathcal{N}\left(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n, \mathbf{w}), \sigma_k^2(\mathbf{x}_n, \mathbf{w})\right) \right\}$$

- Error Function
- w에 대해서 미분하여 Backpropagation을 통해 파라미터를 학습한다

Mixture Density Network

- 이러한 방식으로 fitting된 조건부 분포는 x에 따라서 매우 유연하게 변할 수 있다. $k = 1, \ldots, K$ 에서 하나의 $\pi_k(x)$가 나머지 혼합 계수들에 비해 훨씬 더 크다면 단봉(unimodal) 분포를 형성할 것이고, 만약 세 개의 혼합 계수들이 유의하다면 삼봉 분포를 형성할 것이다.

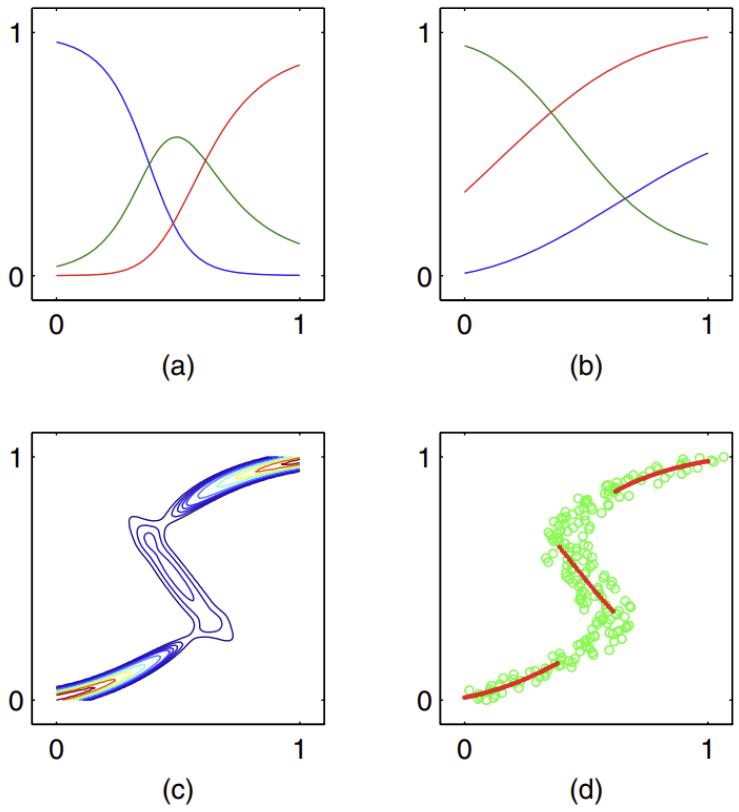


Figure 5.21 (a) Plot of the mixing coefficients $\pi_k(x)$ as a function of x for the three kernel functions in a mixture density network trained on the data shown in Figure 5.19. The model has three Gaussian components, and uses a two-layer multilayer perceptron with five 'tanh' sigmoidal units in the hidden layer, and nine outputs (corresponding to the 3 means and 3 variances of the Gaussian components and the 3 mixing coefficients). At both small and large values of x, where the conditional probability density of the target data is unimodal, only one of the kernels has a high value for its prior probability, while at intermediate values of x, where the conditional density is trimodal, the three mixing coefficients have comparable values. (b) Plots of the means $\mu_k(x)$ using the same colour coding as for the mixing coefficients. (c) Plot of the contours of the corresponding conditional probability density of the target data for the same mixture density network. (d) Plot of the approximate conditional mode, shown by the red points, of the conditional density.

7

Bayesian Neural Network

Find Posterior Distribution of parameters

Introduction

- 지금까지 뉴럴 네트워크에 대한 논의는 Maximum Likelihood를 이용해서 파라미터들을 구하는 데 집중되어 있었다.
- 물론 정규화항을 추가한 모델에 대해선 Bayesian 모델의 관점에서 해석할 수도 있었다.

	Model with regularization term	Bayesian model
Regularization	$\lambda\Omega$	$-\ln$ prior
Error	$E+\lambda\Omega$	$\ln$ posterior
Solution	$\underset{w}{\operatorname{argmin}}\,(E+\lambda\Omega)$	MAP

- 하지만 직접적으로 w에 대한 posterior distribution을 구하지는 않았다. 사실 w를 확률변수로 가정하지조차 않았다고 볼 수 있다. 정규화항에 대해서 설명할 때 prior 분포로 '해석' 가능하다고 언급만 했을 뿐, 실제로 그러한 가정이 없더라도 적당한 결론을 얻는 데에는 아무런 영향이 없었다.
- 하지만 이제부터는 명시적으로 w를 분포로 생각을 할 것이며, 그것의 posterior를 구할 것이다. Why?
  - Predictive distribution 을 구할 수 있다
  - Evidence Framework를 사용하여 hyperparameter를 결정하고 서로 다른 모델들 간의 비교를 할 수 있다(다른 수의 히든 유닛을 가진 모델)
- Posterior 계산의 어려움
  - 단순하게 생각했을 때, conditional distribution $p(t|x)$와 prior $p(w)$가 모두 가우시안이라면 posterior도 가우시안이 되어야만 할 것 같다.
  - 하지만 Multilayer Neural Network의 경우 w에 대한 비선형적인 연산이 이뤄지기 때문에, posterior는 가우시안이 아니게 되어 버리는 문제가 발생한다.
  - 흔히 Error function이 non-convex 한 경우도 많이 있는데, 이는 곧 log posterior가 매우 복잡한 형태로 되어 있음을 시사한다
- Posterior 계산 방법
  - Variational Inference(Chapter 10)
  - Laplace Approximation + Assumption on Covariance

Bayesian Neural Networks for Regression

- Conditional Distribution
  - $p(t|\boldsymbol{x}, \boldsymbol{w}, \beta) = N(t|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1})$
  - 여기서 $y(\boldsymbol{x}, \boldsymbol{w})$는 x가 주어졌을 때 뉴럴 네트워크의 출력값이다
  - β는 precision(inverse variance)
  - 일단 간단한 케이스를 살펴보기 위해 t는 1차원이라고 가정
- Prior Distribution over w
  - $p(\boldsymbol{w}|\alpha) = N(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}I)$
- Likelihood function
  - $p(D|\boldsymbol{x}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} N(t_n|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1})$
  - $D = \{t_1, \dots, t_N\}$
  - Since $t_i's$ are i.i.d. Likelihood is product of conditional distribution
- Posterior Distribution
  - $p(\boldsymbol{w}|D, \alpha, \beta) \propto prior \times likelihood =$
    $N(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}I) \prod_{n=1}^{N} N(t_n|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1})$
  - 이 식을 w에 대해 풀어서 쓰면 $y(\boldsymbol{x}, \boldsymbol{w})$에 의해 w에 대한 non-linear한 형태로 나오게 된다
  - 따라서 posterior는 가우시안이 아니다

Laplace Approximation(review)

- 라플라스 근사는 어떠한 확률분포를 가우시안으로 근사시킨다.
- 가우시안 분포는 exp(-quadratic)의 형태를 가지고 있으므로, 가우시안 분포에 log를 취하면 -quadratic이 나온다.
- 이 점에 주목하여 어떠한 pdf에 log를 씌운 값을 -quadratic의 형태로 만든다면, 해당 pdf를 가우시안으로 근사할 수 있다

- 임의의 pdf를 f(z)라고 하자. 이때 ln f(z)에 대해서 테일러 전개를 하면 다음과 같이 근사시킬 수 있다
  - $\ln f(z) \approx \ln f(z_0) + (z - z_0)\frac{d \ln f(z)}{dz}|_{z=z_0} + \frac{1}{2}(z - z_0)^2 \frac{d^2 \ln f(z)}{dz^2}|_{z=z_0}$
  - 이 식을 간편화하기 위해 f(z)의 최빈값을 찾아야 한다. 최빈값에서의 1차 미분값은 0이기 때문이다.
  - z_0가 최빈값이었다고 했을 때, 위의 식은 다음과 같이 더 간편화될 수 있다
    - $\ln f(z) \approx \ln f(z_0) - \frac{1}{2}(z - z_0)^2 \left(-\frac{d^2 \ln f(z)}{dz^2}|_{z=z_0}\right)$.
  - 또한 최빈값의 경우 일반적으로 (local) maximum에 해당하고 그 경우 해당 지점에서의 2차 미분값은 마이너스가 된다
  - 따라서 위의 식은 -quadratic의 형태를 만족시킨다.
  - 따라서 f(z)는 근사적으로 $Normal(z_0, \left(-\frac{d^2 \ln f(z)}{dz^2}|_{z=z_0}\right)^{-1})$이다
- M차원 분포에 대해서 Laplace Approximation을 적용하면 다음과 같은 결과를 얻을 수 있다.
  - $f(\boldsymbol{z}) \approx Normal(\boldsymbol{z}_0, \left(-\nabla\nabla \ln f(\boldsymbol{z})|_{\boldsymbol{z}=\boldsymbol{z}_0}\right)^{-1})$

Laplace Approximation on Posterior Distribution

- 일단 log posterior는 다음과 같다

$$\ln p(\mathbf{w}|\mathcal{D}) = -\frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} - \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \mathrm{const}$$

- 앞서 보았듯이 일단 posterior 의 최빈값과 이 지점에서의 2차 미분값(헤시안 행렬)을 구해야 한다
- Posterior의 최빈값은 MAP에 해당하는데, 이는 1부에서 다룬 Conjugate Gradient를 이용하여 구할 수 있다.
- 2차 미분값은 다음과 같이 나타낼 수 있다.

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w}|\mathcal{D}, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H}$$

- 그러면 $p(\boldsymbol{w}|D) \approx N(\boldsymbol{w}|\boldsymbol{w}_{MAP}, A^{-1})$
- 앞으로 근사된 사후 분포를 $q(\boldsymbol{w}|D)$로 나타낼 것이다

$$H = \nabla\nabla\frac{1}{2}\sum_{n=1}^{N}\{y(\boldsymbol{x_n}, \boldsymbol{w}) - t_n\}^2$$

- 헤시안 행렬을 구하는 방법에 대해서는 1부에 소개됨
- 혹은 외적 근사법을 이용하여 근사적인 값을 구할 수도 있음

Predictive Distribution

- $p(t_{new}|\boldsymbol{x}_{new}, D) = \int p(t_{new}|\boldsymbol{x}_{new}, \boldsymbol{w})q(\boldsymbol{w}|D)d\boldsymbol{w}$
  - Posterior 자리에 Approximated Posterior가 들어감
  - $p(t_{new}|\boldsymbol{x}_{new}, \boldsymbol{w}) = N(t_{new}|y(\boldsymbol{x}_{\boldsymbol{new}}, \boldsymbol{w}), \beta^{-1})$ 이므로 여전히 w에 대한 non-linear한 형태로 표현이 된다
  - 따라서 전체를 w에 대해 적분하기가 매우 까다롭다. 따라서 $y(\boldsymbol{x}_{\boldsymbol{new}}, \boldsymbol{w})$ 이 부분을 w에 대해 linear하도록 근사시켜 줄 것이다.
    - $y(\boldsymbol{x}, \boldsymbol{w}) \approx y(\boldsymbol{x}, \boldsymbol{w}_{MAP}) + (\boldsymbol{w} - \boldsymbol{w}_{MAP})^T \nabla_{\boldsymbol{w}} y(\boldsymbol{x}, \boldsymbol{w})|_{\boldsymbol{w}=\boldsymbol{w}_{MAP}}$
    - 그러면 다음과 같은 conditional pdf를 얻을 수 있다
      - $p(t|x, w) \approx N(t|y(\boldsymbol{x}, \boldsymbol{w}_{MAP}) + (\boldsymbol{w} - \boldsymbol{w}_{MAP})^T \nabla_{\boldsymbol{w}} y(\boldsymbol{x}, \boldsymbol{w})|_{\boldsymbol{w}=\boldsymbol{w}_{MAP}}, \beta^{-1})$
  - 최종적으로 근사된 posterior와 근사된 conditional pdf 를 곱한 뒤 그것을 w에 대해 marginalize out 해 주면, 다음과 같은 predictive distribution 을 얻을 수 있다.

$$p(t|\mathbf{x}, \mathcal{D}, \alpha, \beta) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}), \sigma^2(\mathbf{x})\right)$$

$$\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g}.$$

$$\mathbf{g} = \nabla_{\mathbf{w}} y(\mathbf{x}, \mathbf{w})|_{\mathbf{w}=\mathbf{w}_{\mathrm{MAP}}}$$

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w}|\mathcal{D}, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H}$$

Hyperparameter optimization

- 앞서서 posterior 와 predictive distribution 을 구할 땐, α, β(prior의 precision, conditional pdf의 precision)는 임의의 값을 고정한 상태로 구했었다.
- 여기서는 chapter 3.5에서 소개되었던 evidence framework를 활용해서 hyperparameter를 구하는 방법을 살펴볼 것이다. Empirical Bayes라는 방법으로 불리기도 한다
- Evidence(Marginal Likelihood): $p(D|\alpha, \beta) = \int p(D|\boldsymbol{w}, \beta) p(\boldsymbol{w}|\alpha) d\boldsymbol{w}$
- $\hat{\alpha}, \hat{\beta} = argmax_{\alpha,\beta}\, p(D|\alpha, \beta)$ → 즉 결국 MLE를 구하는 문제가 된다

$$\ln p(\mathcal{D}|\alpha, \beta) \simeq -E(\mathbf{w}_{\mathrm{MAP}}) - \frac{1}{2}\ln|\mathbf{A}| + \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi)$$

- 따라서 수리통계학2에서 배웠듯이 위의 log evidence를 α, β에 대해 편미분한 뒤 그것을 0으로 두고 방정식을 풀면 된다.
- 하지만 방정식을 푸는 것이 굉장히 까다롭기 때문에 수치해석적인 방법을 통해 해를 구하게 된다(자세한 내용은 교재 참고)

Bayesian Neural Network for Classification

- Log likelihood function: $\ln p(D|\boldsymbol{w}) = \sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$
  - Likelihood는 conditional pdf의 product
  - $t_n \in \{0,1\}$
  - $y_n \equiv y(\boldsymbol{x}_n, \boldsymbol{w})$
- 여기서도 laplace approximation 을 통해 posterior를 구해야 한다.
  - 먼저 w_{MAP}를 구한다(1부에서 나왔던 최적화 알고리즘 이용)
  - 헤시안 행렬 H 구하기(외적 근사법 등)
  - 위의 값들을 이용해 posterior 를 가우시안으로 근사할 수 있다.
- Predictive Distribution 을 구할 때도 앞서 설명한 방식대로 진행할 수 있으나 한 가지 차이점이 존재한다
  - Conditional distribution 에 들어가는 $y(\boldsymbol{x}_n, \boldsymbol{w})$ 부분이 w에 대해 비선형적이어서 문제가 됐었기 때문에, 이를 선형적으로 근사했었다
  - 하지만 Classification의 경우 $y(\boldsymbol{x}_n, \boldsymbol{w})$ 는 0~1 사이의 값을 가지기 때문에 이에 대해서 선형적으로 근사를 하는 것은 적절하지 않다.
  - 대신에 sigmoid activation에 들어가기 직전의 값에 선형 근사를 해 줄 수 있다.

$$a(\mathbf{x}, \mathbf{w}) \simeq a_{\mathrm{MAP}}(\mathbf{x}) + \mathbf{b}^{\mathrm{T}}(\mathbf{w} - \mathbf{w}_{\mathrm{MAP}})$$

$$\left.\begin{aligned} a_{\mathrm{MAP}}(\mathbf{x}) &= a(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}) \\ \mathbf{b} &\equiv \nabla a(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}) \end{aligned}\right\}$$ Backpropagation을 통해 구할 수 있음

  - 최종적으로 predictive distribution 을 구해 주면 다음과 같다

$$p(t=1|\mathbf{x}, \mathcal{D}) = \sigma\left(\kappa(\sigma_a^2)\mathbf{b}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}}\right)$$

$$\kappa(\sigma^2) = (1 + \pi\sigma^2/8)^{-1/2}.$$

감사합니다