Optimization (II):SVM

Jin Young Choi

Seoul National University

Outline

- Constraint Convex Optimization
- linear/quadratic programming
- dual problem, KKT conditions
- Minimization techniques
  - Gradient Descent Minimization
  - Newton Minimization
  - Gauss Newton Minimization
  - (In)Equality Constraint Minimization
- Structural Risk Minimization
  - Support Vector Machine
    - https://www.csie.ntu.edu.tw/~cjlin/papers/bottou_lin.pdf
    - Bottou and Lin, Support Vector Machine Solvers
  - Support vector data description
- Error Backpropagation Learning for a Neural Network

Support Vector Machine

Binary(Two-class) Classifier

- Binary Classifier

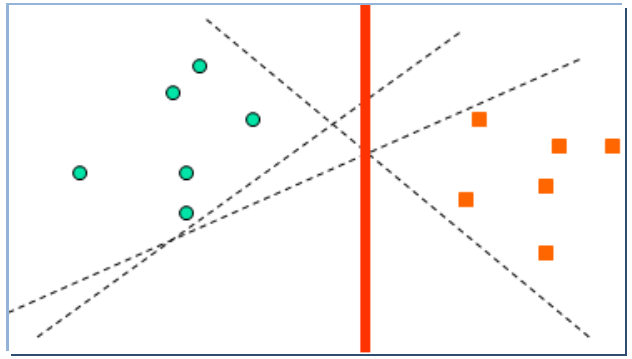
$$g(\boldsymbol{y}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{y}, \;\; \boldsymbol{y} = (\boldsymbol{x}, 1)$$

- Learning Rule

$$\mathbf{w} = \mathbf{w}_{now} + \eta_{now}\delta_{now}\mathbf{y}_i$$

$$\delta_{now} = (d_i - \mathbf{w}^t{}_{now}\mathbf{y}_i) \;\; , \;\; \text{for Widrow-Hoff}$$

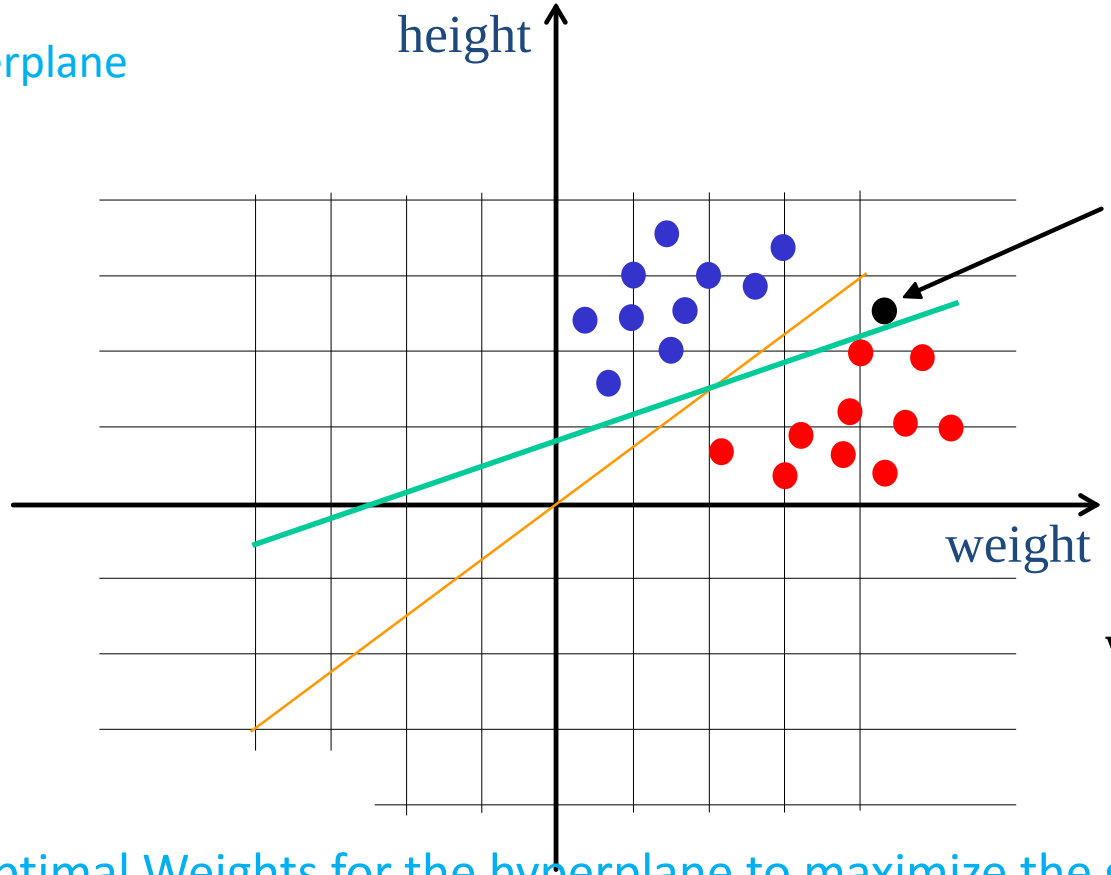
$$\delta_{now} = d_i \;\; , \;\; \text{for Perceptron, Hebian}$$



$$\mathbf{w}_{final} = \sum_{i=1}^{N} \alpha_i d_i \mathbf{y}_i, \;\; \text{for } \delta_{now} = d_i$$

Structural Risk Minimization

Optimal hyperplane

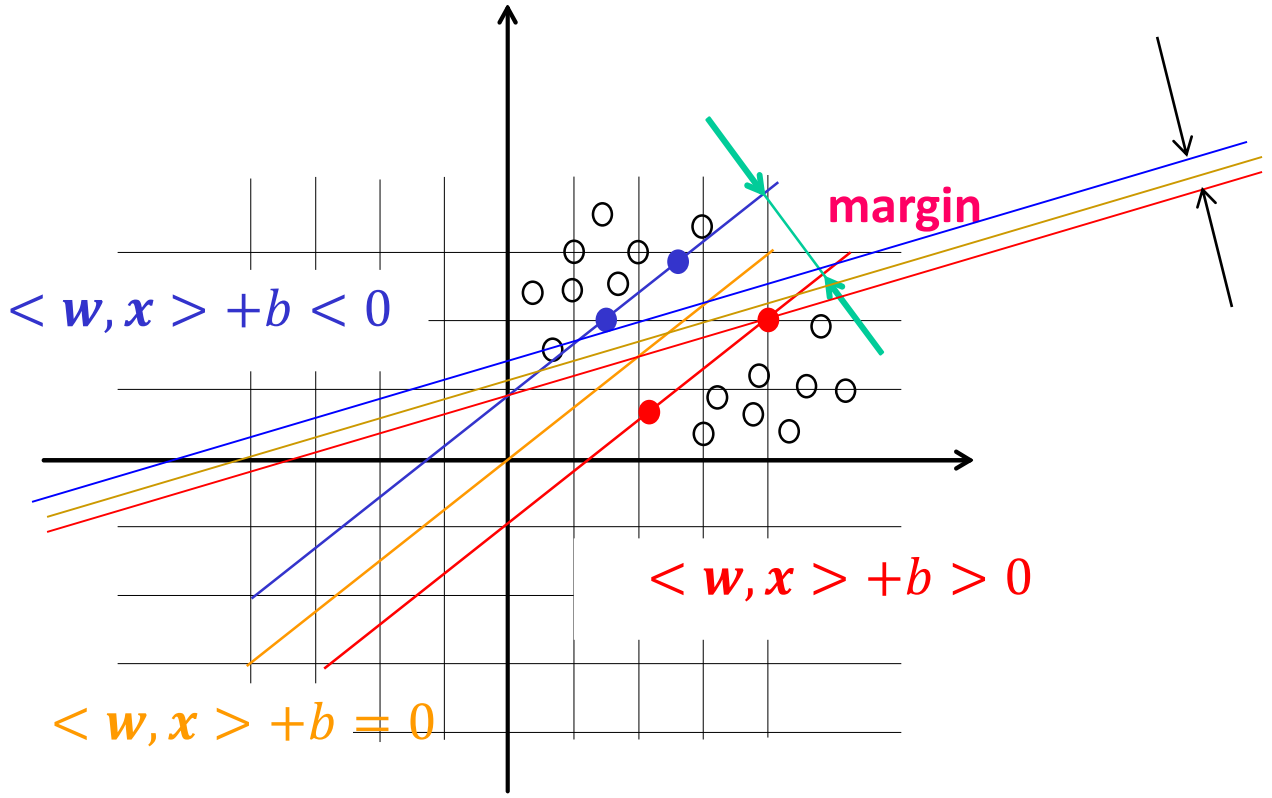


$$\mathbf{w}_{final} = \sum_{i=1}^{N} \alpha_i d_i \mathbf{y}_i,$$

What is optimal Weights for the hyperplane to maximize the generalization ?

→ Structural Risk Minimization

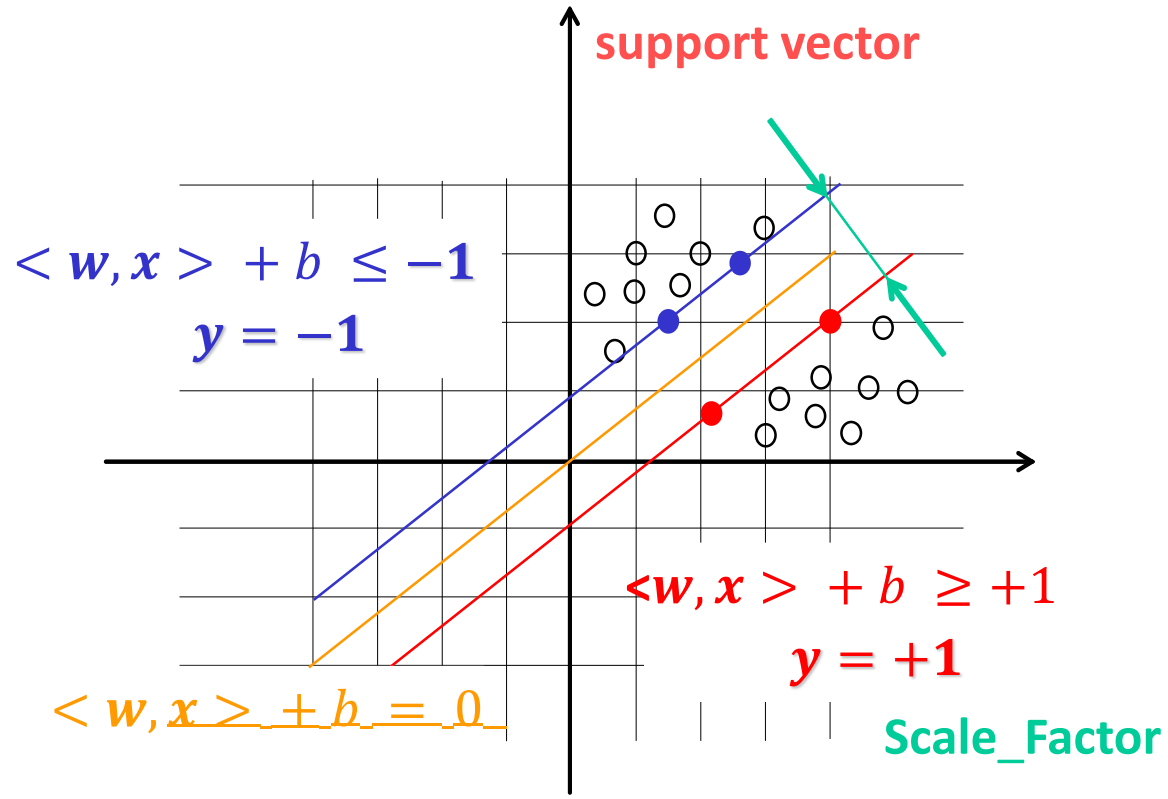
Optimal Hyperplane



Maximal Margin Hyperplane=Optimal Hyperplane

$$w \cdot x =< w, x >= w^T x = w_1 x_1 + w_2 x_2 = |w||x|\cos\theta$$

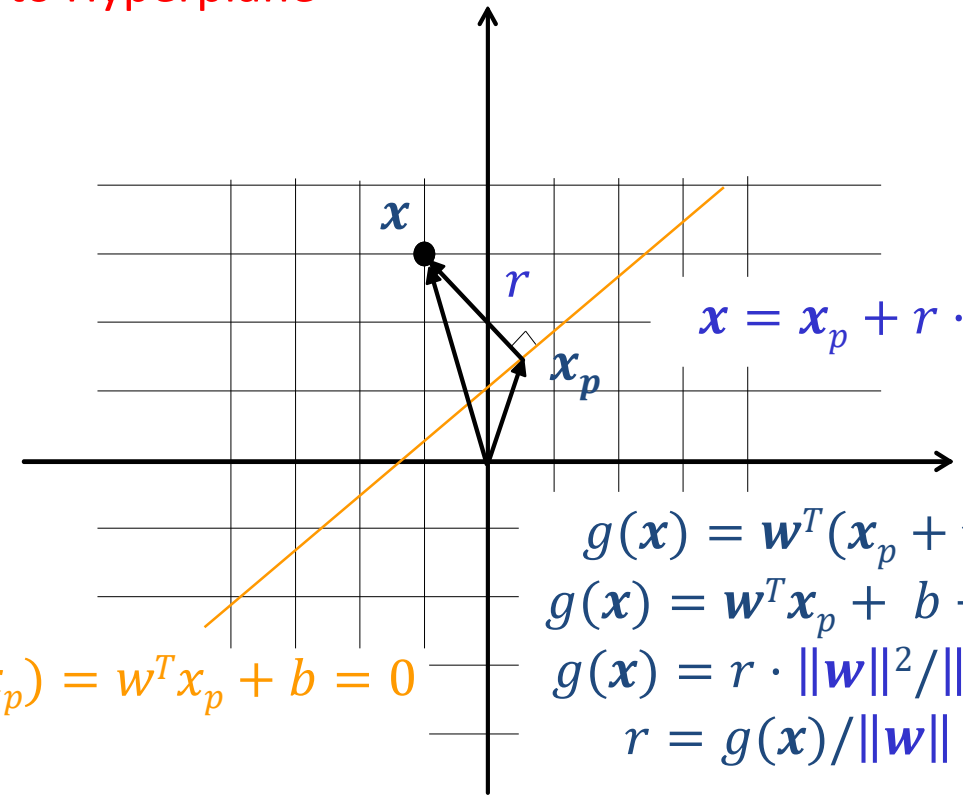
Optimal Hyperplane



$y = w^T x + b \leftarrow N + 1$ 차원 $\quad hyperplane\ since\ [x, y] \in R^{N+1}$
$0 = w^T x + b \leftarrow N$ 차원 $\quad hyperplane\ since\ [x] \in R^N$

Optimal Hyperplane

- Distance from Sample to Hyperplane



$$x = x_p + r \cdot w/\|w\|$$

$$y = g(x_p) = w^T x_p + b = 0$$

$$g(x) = w^T(x_p + r \cdot w/\|w\|b$$

$$g(x) = w^T x_p + \ b + r \cdot w^T w/\|w\|$$

$$g(x) = r \cdot \|w\|^2/\|w\|$$

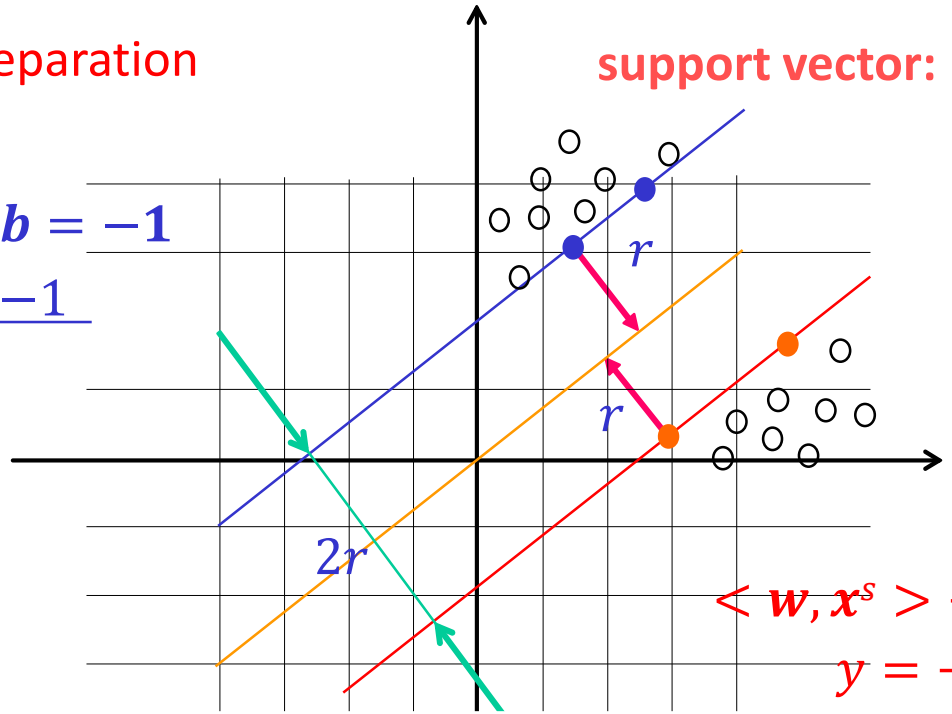
$$r = g(x)/\|w\|$$

$$r = g(x)/\|w\|$$

Optimal Hyperplane

- Margin of Separation

support vector: $\boldsymbol{x^s}$



$r = g(\boldsymbol{x^s})/\|\boldsymbol{w}\|$

Where, $g(\boldsymbol{x^s}) = \boldsymbol{w}^T\boldsymbol{x^s} + b = \pm 1 \; for \; ds = \pm 1$

margin: $\boldsymbol{2r = 2/\|w\|}$

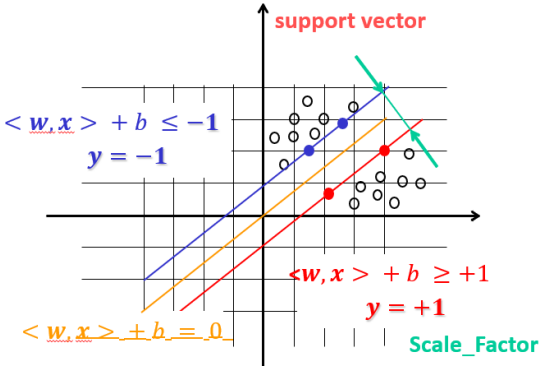
$$\left| \frac{1}{\|\mathbf{w}\|} - \frac{-1}{\|\mathbf{w}\|} \right| = \frac{2}{\|\mathbf{w}\|}$$

Optimal Hyperplane

margin: $\left| \frac{1}{\|\mathbf{w}\|} - \frac{-1}{\|\mathbf{w}\|} \right| = \frac{2}{\|\mathbf{w}\|}$

$$< \boldsymbol{w}, \boldsymbol{x} > + b \leq -1, \qquad y = -1$$

$$< \boldsymbol{w}, \boldsymbol{x} > + b \geq +1, \qquad y = +1$$



Constrained Optimization Problem

(2nd order) objective/cost function: convex

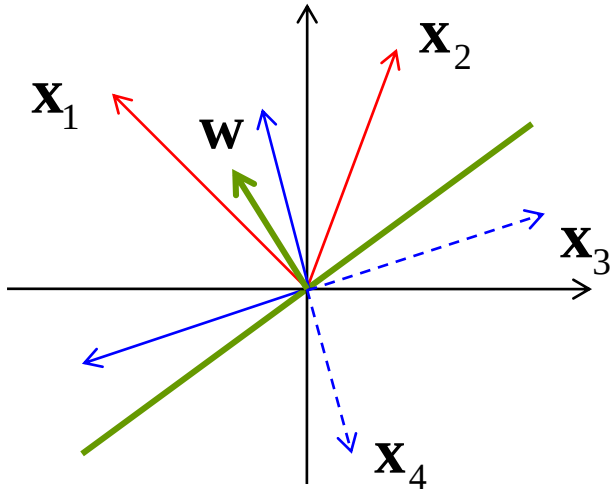
$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \qquad i = 1, 2, \ldots, n \end{aligned}$$

(1st order) constraints: convex

Relation btw SVM & Perceptron Rule

- The weight obtained by Perceptron Rule

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \, y_i \mathbf{x}_i,$$



- The linear classifier can be represented by

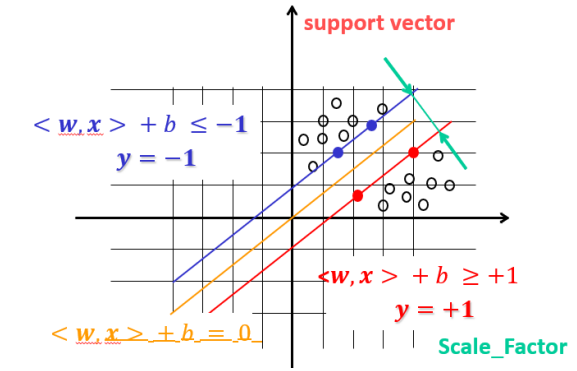
$$g(x) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b$$

- Data appear only inside dot products
- SVM chooses smallest number of samples

Lagrange Function

$$\begin{aligned} \underset{w,b}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1, \quad i = 1, \ldots, n. \end{aligned}$$

$$L_P(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1]$$



Solution:

$$\frac{dLp}{d\mathbf{w}} = 0 \qquad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\frac{dLp}{db} = 0 \qquad 0 = \sum_{i=1}^{n} \alpha_i y_i \qquad (\alpha_i \geq 0)$$

KKT condition: $\alpha_i = 0$ unless $y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$

Dual constraints:

$\lambda_i \geq 0, \ i = 1, \ \ldots, \ m$

Complementary slackness:

$\lambda_i g_i(\theta) = 0, \ i = 1, \ \ldots, \ m$

Gradient of Lagrangian w.r.t. θ vanishes:

$$\nabla E_0(\theta) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\theta) + \sum_{i=1}^{p} \nu_i \nabla h_i(\theta) = 0$$

Remark on Support Vector

- KKT Condition

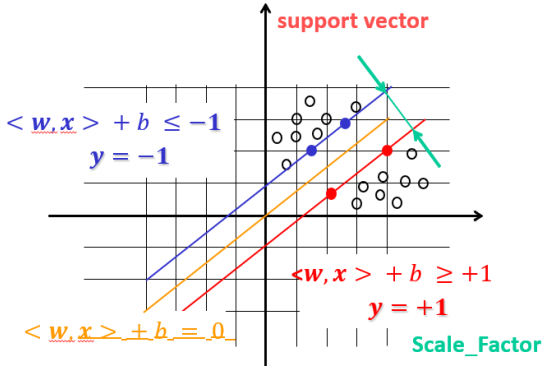
$$\alpha_i[\,1 - y_i(w^T x_i + b)] = 0$$

If $\alpha_i \neq 0$ then $y_i(w^T x_i + b) = 1$

$\Longrightarrow$ x_i is support vector, $\alpha_i > 0$

If $y_i(w^T x_i + b) \neq 1$ then $\alpha_i = 0$

$\Longrightarrow$ x_i is not support vector



$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

w is only related to support vectors, x_i

Dual Problem

$$L_p(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

$$= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{w}^T \mathbf{x}_i - b \sum_{i=1}^{n} \alpha_i y_i$$

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$0 = \sum_{i=1}^{n} \alpha_i y_i \qquad (\alpha_i \geq 0)$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$-\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {\mathbf{x}_i}^T \mathbf{x}_j$$

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {\mathbf{x}_i}^T \mathbf{x}_j$$

Large scale quadratic problem

$n \times n > 1000$

$$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {\mathbf{x}_i}^T \mathbf{x}_j = -\frac{1}{2}\alpha^T Q \alpha + \alpha^T \mathbf{1}$$

Dual Problem

$$\max. \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad ; \quad \alpha_i \geq 0$$

$$-\frac{1}{2} \alpha^T Q \alpha + \alpha^T \underline{1}$$

$$Q = \begin{bmatrix} & y_i y_j x_i^T x_j & \end{bmatrix}$$

- Optimizing L_D only depends on the input patterns in the form of a set of dot product $< x, x >$
- Not depend on the dimension of the input pattern
- Can replace dot product with Kernel

SVM Summary

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i^\top \mathbf{x} + b^*\right)$$

- **optimal weight**

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i$$

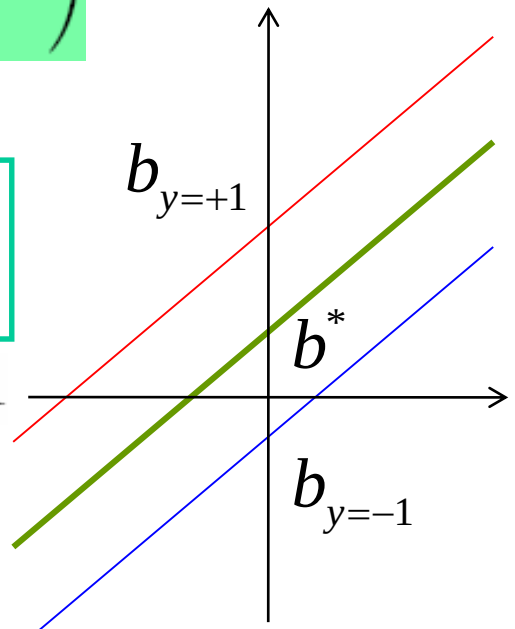
$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$$

$$b_{y=+1} = 1 - \boldsymbol{w}^{*T} \boldsymbol{x}^{(s)}$$

$$b_{y=-1} = -1 - \boldsymbol{w}^{*T} \boldsymbol{x}^{(s)}$$

- **optimal bias**

$$b^* = \frac{1}{2}(b_{y=+1} + b_{y=-1})$$



Soft Margin Technique

Problem: Can't satisfy $y_i[\mathbf{w}^\top \mathbf{x}_i + b] \geq 1$ for all i

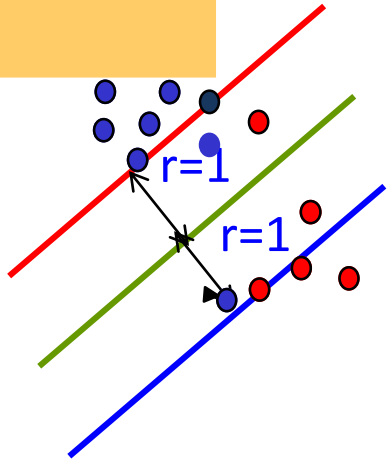
Adopting Slack Variable

$$y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) \geq 1 - \xi_i$$

$$\mathbf{w}^\top \mathbf{x}_i + b \geq +1 - \xi_i \qquad \text{for } y_i = +1,$$

$$\mathbf{w}^\top \mathbf{x}_i + b \leq -1 + \xi_i \qquad \text{for } y_i = -1,$$

$$\xi_i \geq 0 \qquad k = 1, 2, \ldots, n$$



Minimizing Errors

$\xi_i > 1$ $\sum_{i=1}^{n} I(\xi_i > 1) = \#$ errors min. $\sum_{i=1}^{n} I(\xi_i > 1)$

For QP, replace $I(\xi_i > 1)$ by ξ_i

Lagrange Function for Soft Margin Tech.

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & L_P(\mathbf{w},\xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned}$$

penalize errors

C

penalize complexity

$$\begin{aligned} L(\mathbf{w},b,\alpha,\xi) &= \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{i=1}^{m}\xi_i \\ &\quad - \sum_{i=1}^{m}\alpha_i\left[y_i\left(\mathbf{w}\cdot\mathbf{x}_i + b\right) - 1 + \xi_i\right] - \sum_{i=1}^{m} r_i\xi_i \end{aligned}$$

★ $\alpha_i \geq 0$ and $r_i \geq 0$

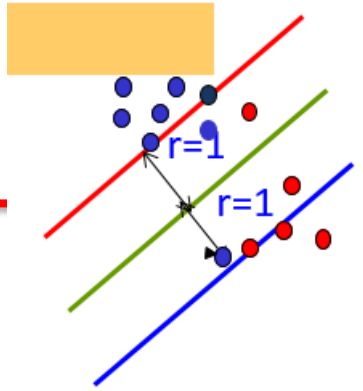
$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m}\alpha_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{m}\alpha_i y_i = 0$$

★ $$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - r_i = 0$$

$\alpha_i[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 + \xi_i] = 0, \quad r_i\xi_i = 0$

Dual Problem for Soft Margin Tech.



violation

$$\alpha_i = C$$
$$\rightarrow r_i = 0$$
$$\rightarrow \xi_i \neq 0$$

$$\max. \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad ; \quad \boxed{0 \leq \alpha_i \leq C} \quad \leftarrow r_i = C - \alpha_i \geq 0$$

correct

$$\alpha_i = 0$$
$$\rightarrow r_i = C$$
$$\rightarrow \xi_i = 0$$

$$\alpha_i \geq 0 \text{ and } r_i \geq 0$$

$$\alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] = 0, \quad r_i \xi_i = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - r_i = 0$$

support

$$0 < \alpha_i < C$$
$$\rightarrow r_i \neq 0$$
$$\rightarrow \xi_i = 0$$

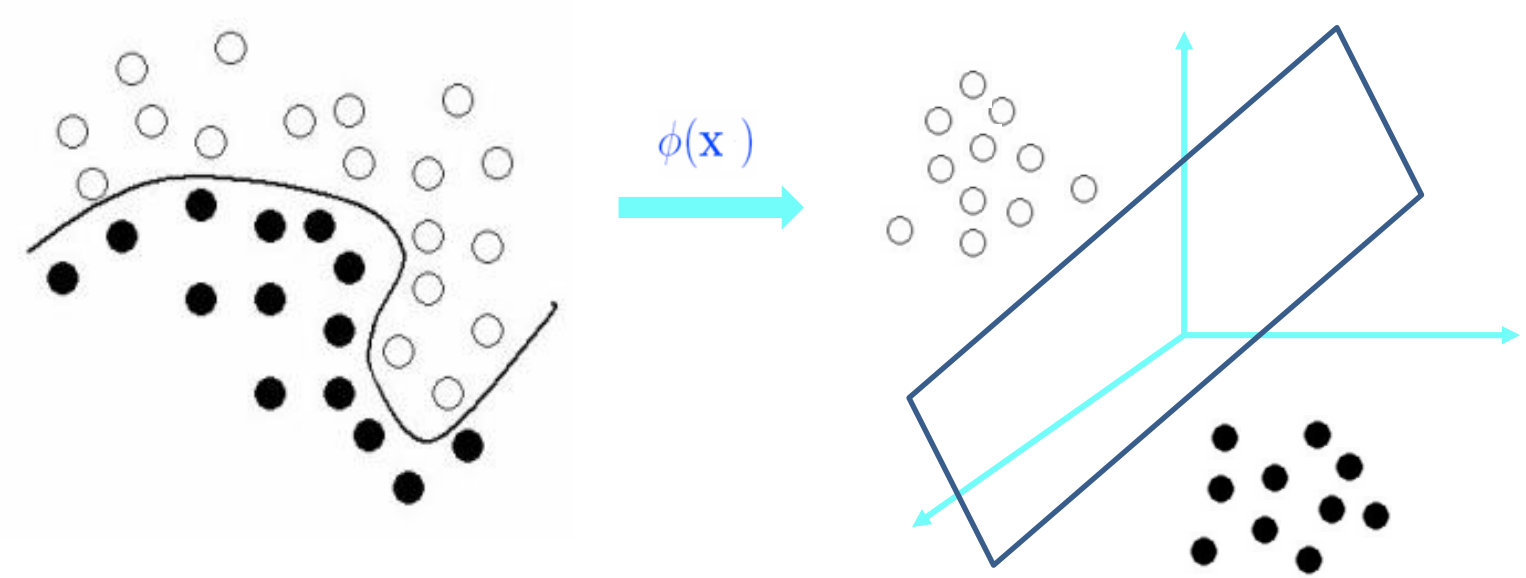
$0 < \alpha_i < C$ *support* pattern

$\alpha_i = 0$ or $\alpha_i = C$ *Non-support* pattern

The optimal value of *C* is determined experimentally, it cannot be readily related to the characteristics of the dataset or model

Nonlinear Support Vector Machines

- If the problem is not linear (1st order plane separable), Kernel trick will be used.



Dual Problem and Classifier in the Feature Space

$$\max. \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \quad \leftarrow \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad ; \quad 0 \le \alpha_i \le C$$

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i^\top \mathbf{x} + b^* \right) \quad \leftarrow \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})$$

The feature space $\phi(\mathbf{x}_i)$ can be huge or infinite !

The feature $\phi(\mathbf{x}_i)$ has the form of inner product

Kernel

- **Kernel: a function k that takes 2 variables and computes a scalar value (a kind of similarity)**

$$k(\mathrm{x}, \mathrm{y}) = (\Phi(\mathrm{x}) \cdot \Phi(\mathrm{y}))$$

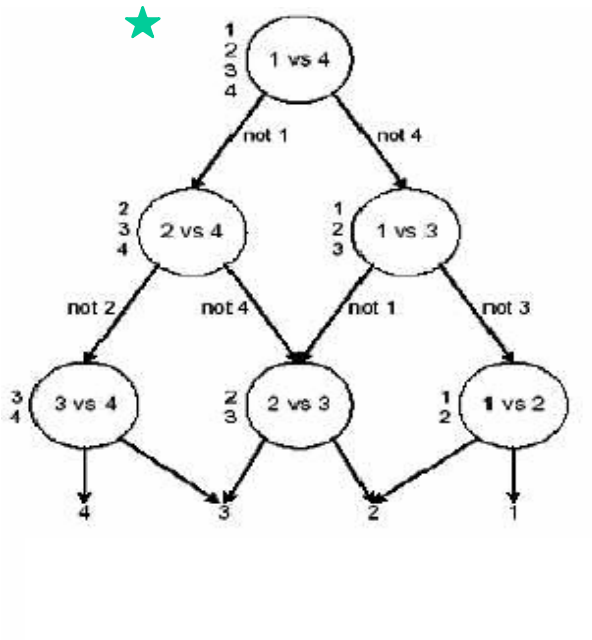
- **Kernel_Matrix: $m \times m$ matrix K with elements $K_{ij} = k(xi, x_j)$.**

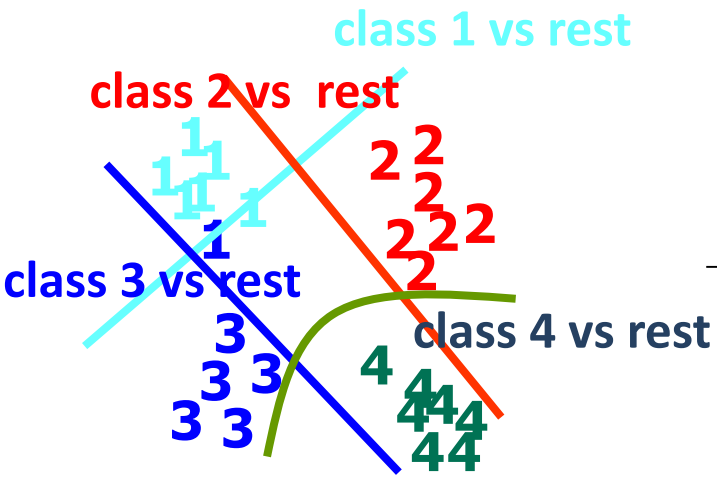
- **Standard Kernels**

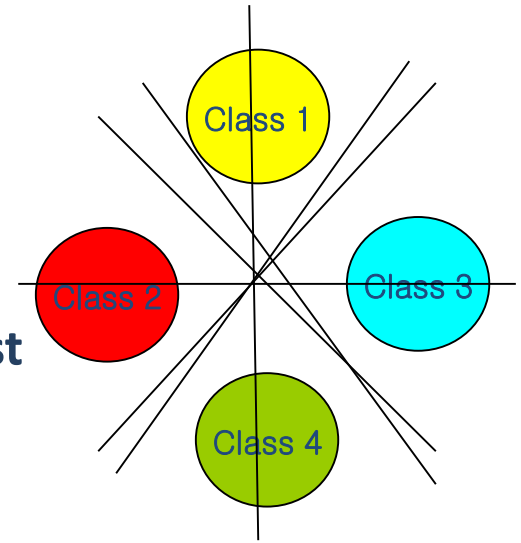
  - **Polynomial Kernel** $k(x, x_i) = (x^T x_i + 1)^d$
  - **Radial Basis Function Kernel** $k(x, x_i) = \exp(- \| x - x_i \|^2 / 2\sigma^2)$
  - **Sigmoid Kernel** $k(x, x_i) = \tanh(\beta_0 x^T x_i + \beta_1)$

Multiclass Support Vector Machines

- DAG (Directed Acyclic Graph) SVM: (k-1)! Classifiers, (2n/k)× (k-1)!
- One-against-all: k classifiers, n×k data training
- One-against-one SVM: k(k-1)/2 classifiers, (2n/k)× (k(k-1)/2)=n×(k-1)





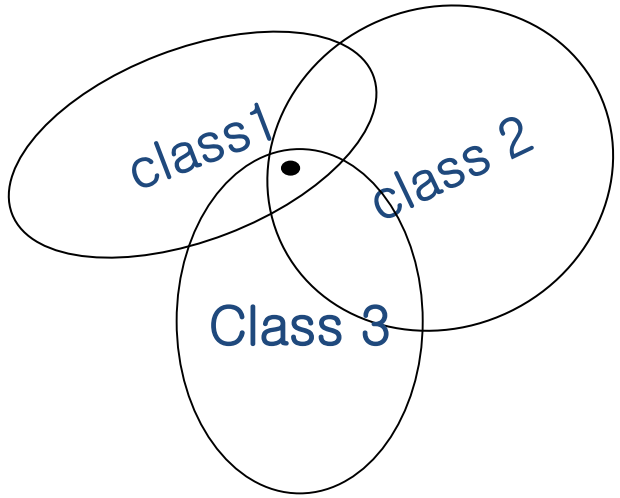


SVDD (Support Vector Data Description)

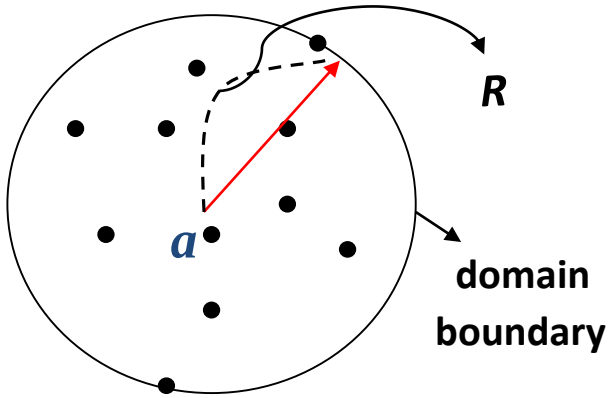
- Primal Problems:

  minimize $F(R, \mathbf{a}) = R^2$

  subject to $\|\mathbf{x}_i - \mathbf{a}\|^2 \leq R^2, \quad \forall i$



SVDD



SVDD (Support Vector Data Description)

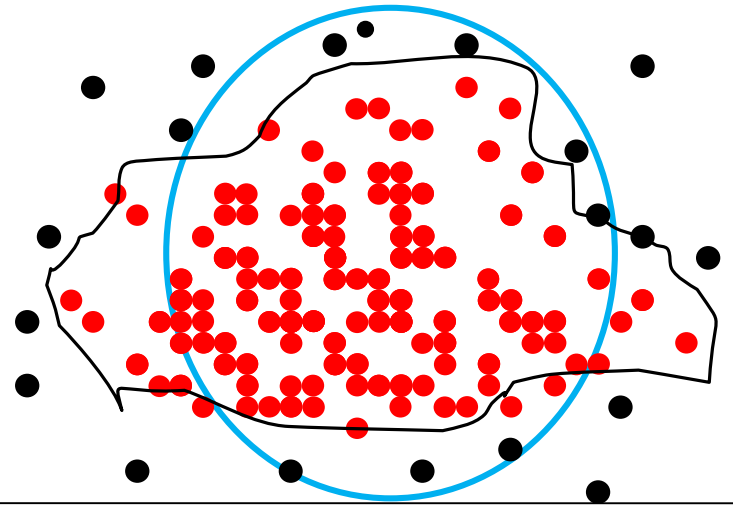
- Primal Problems:

  minimize $$F(R, \mathbf{a}) = R^2 + C \sum_i \xi_i$$

  subject to $$\|\mathbf{x}_i - \mathbf{a}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0 \quad \forall i$$

- Lagrangian

$$L(R, \mathrm{a}, \xi, \alpha) = R^2 + C \sum_i \xi_i + \sum_i \alpha_i (\|x_i - \mathrm{a}\|^2 - R^2 - \xi_i) - \gamma_i \xi_i$$



SVDD (Support Vector Data Description)

- Lagrangian

$$L(R, \mathrm{a}, \xi, \alpha) = R^2 + C\sum_i \xi_i + \sum_i [\alpha_i(\|x_i - \mathrm{a}\|^2 - R^2 - \xi_i) - \gamma_i \xi_i]$$

- KKT Conditions

$$\frac{\partial L}{\partial R} = 0: \quad \sum_i \alpha_i = 1$$

$$\frac{\partial L}{\partial \mathbf{a}} = 0: \quad \mathbf{a} = \frac{\sum_i \alpha_i \mathbf{x}_i}{\sum_i \alpha_i} = \sum_i \alpha_i \mathbf{x}_i$$

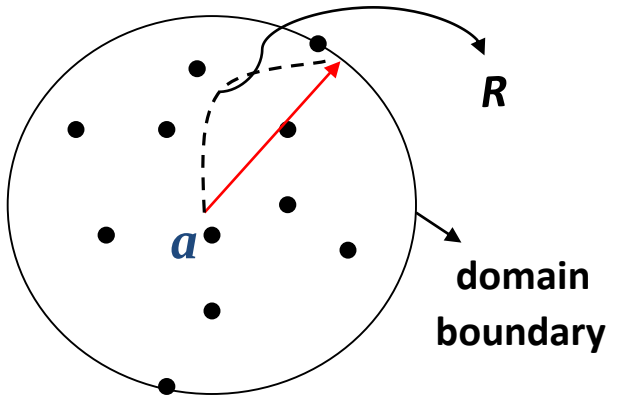
$$\frac{\partial L}{\partial \xi_i} = 0: \quad C - \alpha_i - \gamma_i = 0$$

$$\alpha_i\{R^2 + \xi_i - (\|x_i\|^2 - 2\mathbf{a}\cdot x_i + \|\mathbf{a}\|^2)\} = 0$$

$$\gamma_i \xi_i = 0$$

$$\alpha_i \geq 0, \gamma_i \geq 0, \quad \mathbf{a} = \sum_i \alpha_i x_i, \quad R = \|x_s - \mathbf{a}\|$$

$$0 \leq \alpha_i \leq C$$



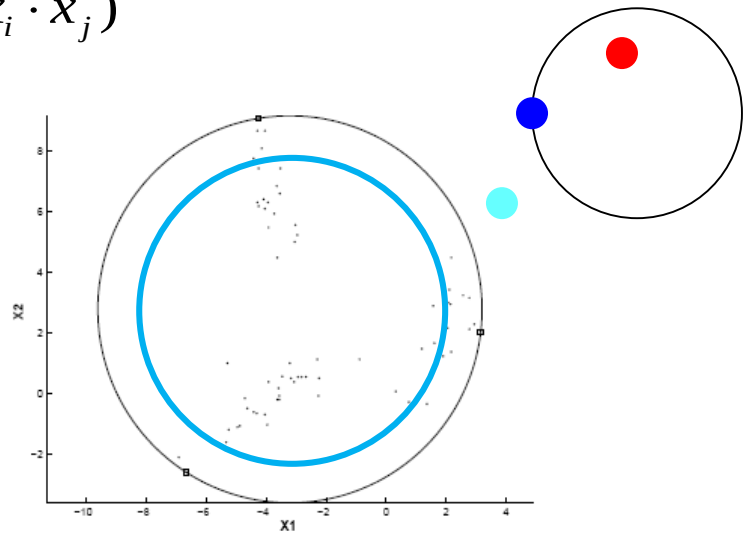
Dual problem of SVDD

- Dual Function of SVDD is given by

$$\textbf{Maximize} \quad L = \sum_i \alpha_i (x_i \cdot x_i) - \sum_{i,j} \alpha_i \alpha_j (x_i \cdot x_j)$$

$$\textbf{Subject to} \quad \sum_i \alpha_i = 1, \;\; C \geq \alpha_i \geq 0$$

- Example: Three Gaussians



$$C - a_i - \gamma_i = 0$$

$$\|\mathbf{x}_i - \mathbf{a}\|^2 < R^2 \rightarrow \alpha_i = 0, \;\; \gamma_i = C, \quad \xi_i = 0$$

$$\|\mathbf{x}_i - \mathbf{a}\|^2 = R^2 \rightarrow 0 < \alpha_i < C, \gamma_i \neq 0, \;\; \xi_i = 0$$

$$\|\mathbf{x}_i - \mathbf{a}\|^2 > R^2 \rightarrow \alpha_i = C \;\; \gamma_i = 0, \quad \xi_i > 0$$

$$\|\mathbf{z} - \mathbf{a}\|^2 = (\mathbf{z} \cdot \mathbf{z}) - 2 \sum_i \alpha_i (\mathbf{z} \cdot \mathbf{x}_i) + \sum_{i,j} \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \leq R^2$$

$$\mathbf{a} = \frac{\sum_i \alpha_i \mathbf{x}_i}{\sum_i \alpha_i} = \sum_i \alpha_i \mathbf{x}_i$$

Generalizing to Kernel SVDD

- We replace all inner product $\boldsymbol{x_i} \cdot \boldsymbol{xj}$ by a proper $\boldsymbol{K}(\boldsymbol{xi}\ \boldsymbol{x_j})$ and the problem of finding a data domain description is given by

$$L = \sum_{i=1}^{n} \alpha_i K(x_i, x_i) - \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j),$$

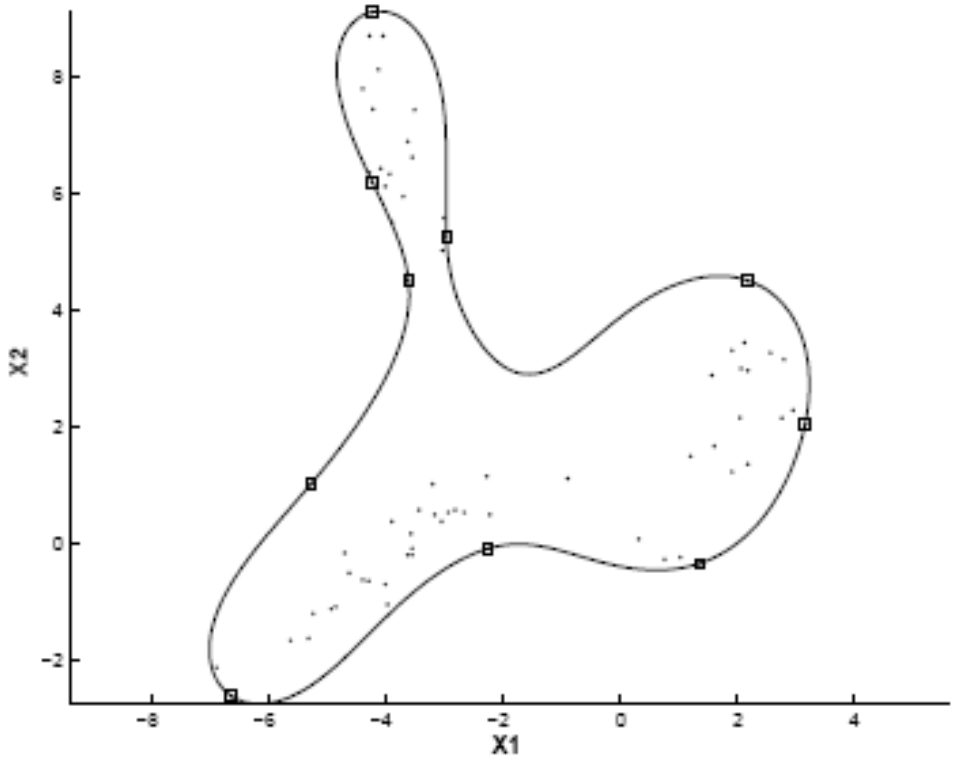
$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i = 1, \quad i = 1, \ldots, n.$$

$$D^2(z) = 1 - 2 \sum_{i=1}^{n} \alpha_i K(z, x_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j).$$

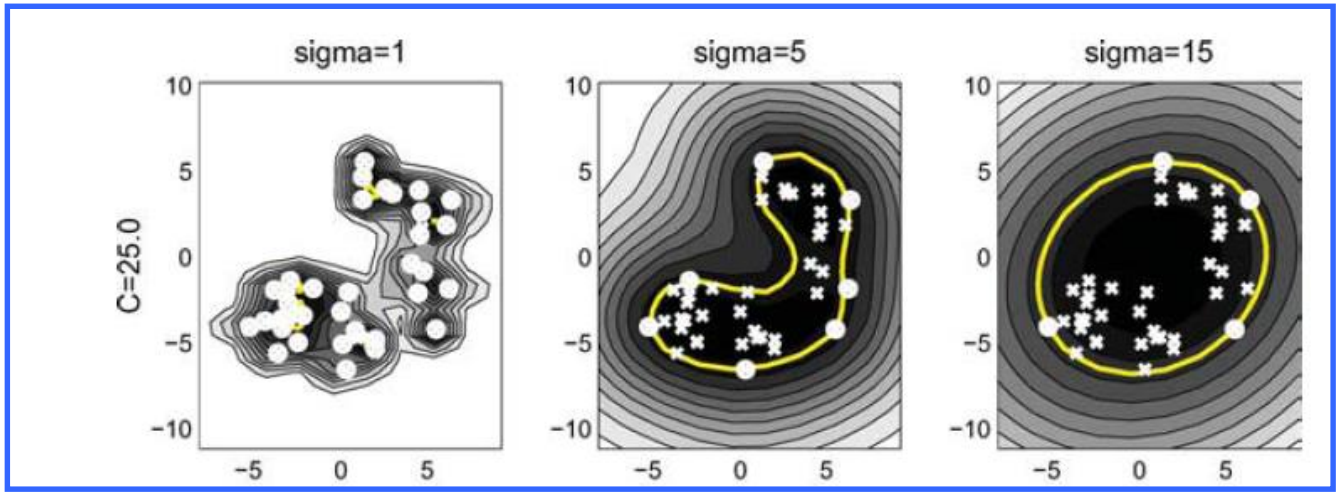
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / s^2)$$

Example in 2D by Gaussian kernel

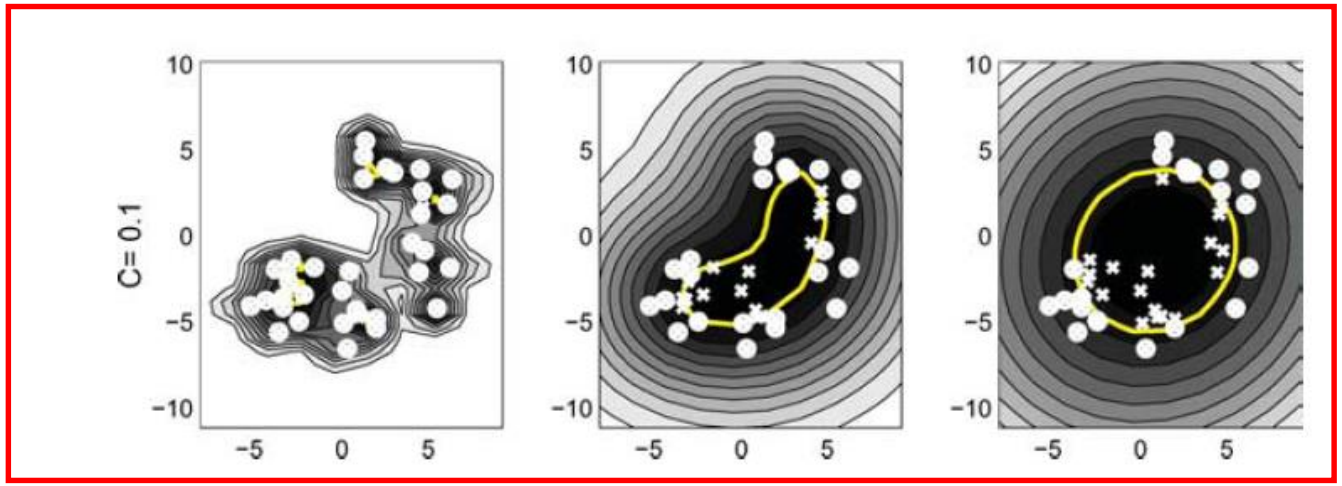
- Domain description of the data from three Gaussian distribution



The effect of parameter (c, s)



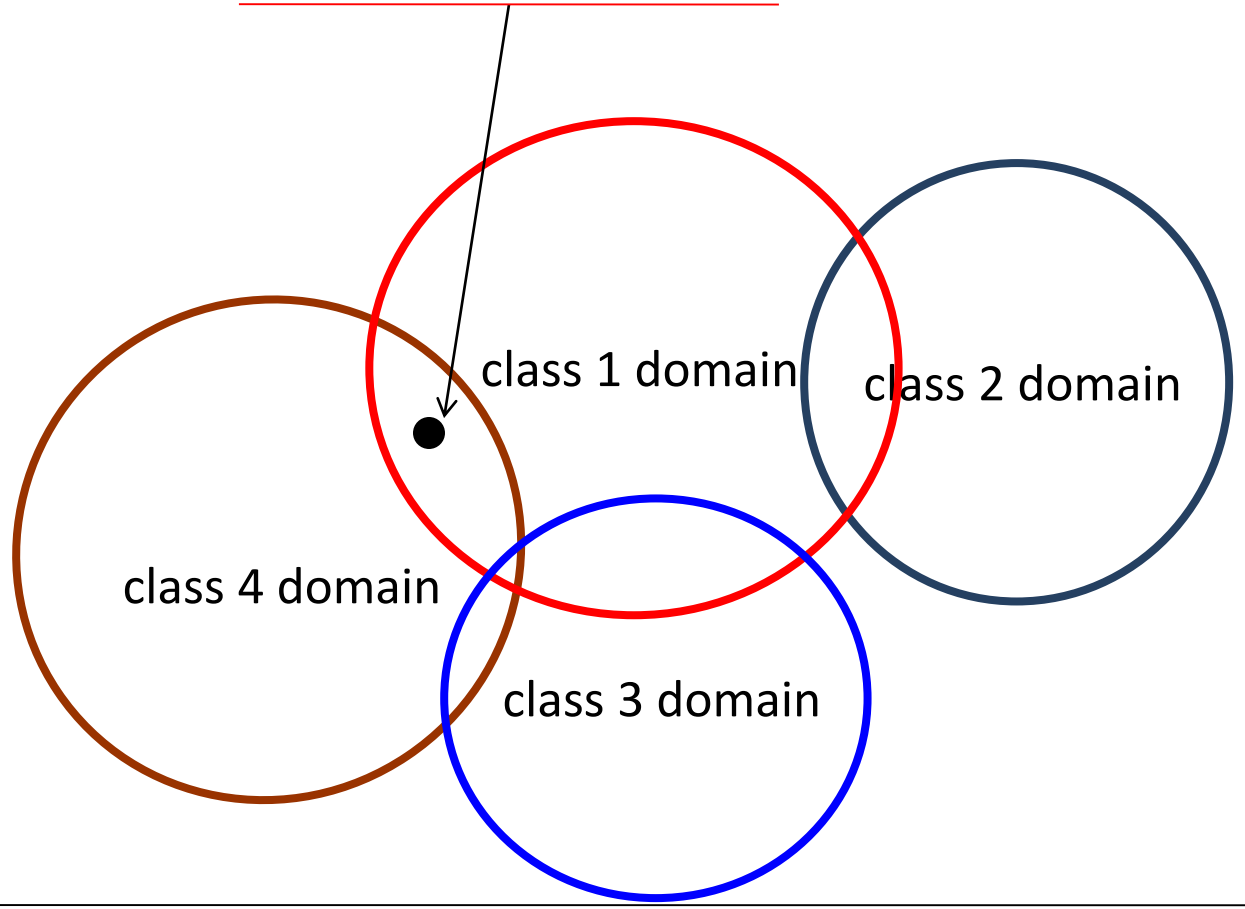
when c ≥1
All data are included



When c<1
All data not included

SVDD-based Multi-class Classifier

- Each class is trained with its own class.
- In the near of boundary, patterns are overlapped



Support Vector Domain Density Description

- Support Vector Domain Density Description(SV-DDD)$D_l^2(x)$
  - Step 1 Get feature space distance function n m n from SVDD training.

$$D_l^2(x) = 1 - 2\sum_{i=1} \alpha_i K(x, x_i) + \sum_{i=1} \sum_{j=1} \alpha_i \alpha_j K(x_i, x_j)$$

  - Step 2 Estimate parameter of proposed SV-DDD func

$$\bar{p}_l(x|\sigma) = \frac{1}{(2\pi\sigma_l^2)^{d/2}} \exp\left(\frac{D_l^2(x)}{2\sigma_l^2}\right) \quad \hat{\sigma}^2 = \frac{1}{nd}\sum_{i=1}^{n} D^2(x_i)$$

  - Step 3 U

$$\text{class of } x \equiv \arg\max_{l=1,\ldots,k} p(C_l) \cdot \bar{p}(x|C_l)$$

$$\text{where } p(C_l) = n_l/N \text{ and } \bar{p}(x|C_l) = \left(\frac{1}{2\pi\hat{\sigma}_l^2}\right)^{d/2} \exp\left(-\frac{D_l^2(x)}{2\hat{\sigma}_l^2}\right).$$

Support Vector Domain Density Description

- SV-DDD -parameter estimation by Maximum Likelihood

$$\bar{p}(X|\sigma) = \prod_{i=1}^{n} \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{D^2(x_i)}{2\sigma^2}\right)$$

We maximize the following log-likelihood

$$\begin{aligned}\mathcal{L}(\sigma) &\equiv \ln \bar{p}(X|\sigma), \\ &= \sum_{i=1}^{n} \ln \bar{p}(x_i|\sigma), \\ &= \sum_{i=1}^{n} \left(\ln \frac{1}{(2\pi\sigma^2)^{d/2}} - \frac{D^2(x_i)}{2\sigma^2}\right)\end{aligned}$$

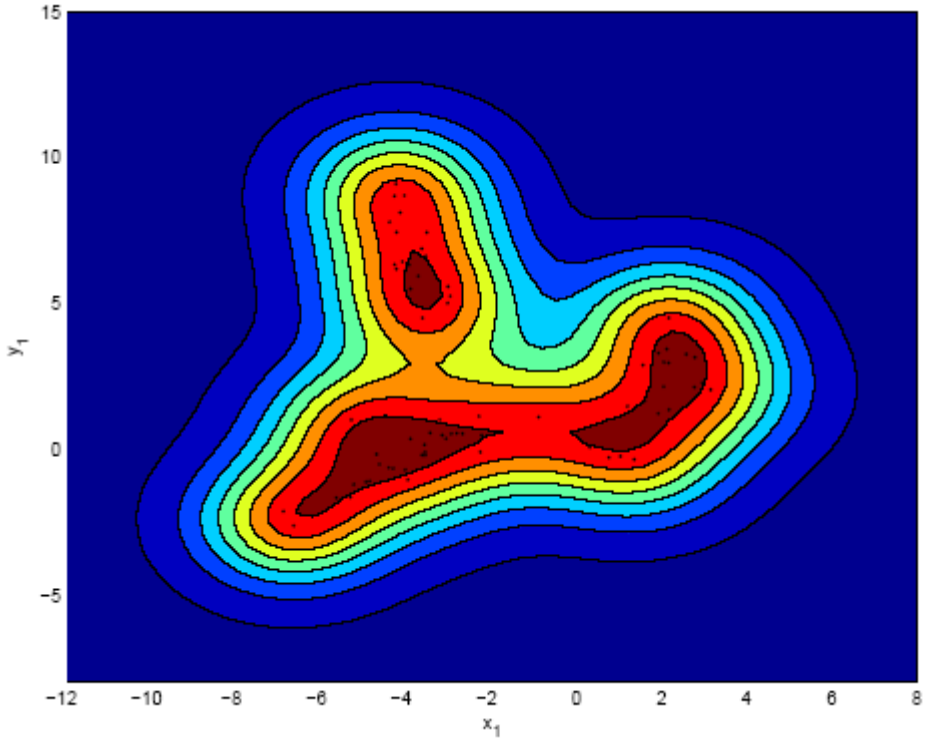
$$\nabla_\sigma \mathcal{L} = \sum_{i=1}^{n} \left(-\frac{d}{\sigma} + \frac{D^2(x_i)}{\sigma^3}\right)$$

$\hat{\sigma}^2$ can be obtained as follows by setting to $z\nabla_\sigma \mathcal{L}$.

$$\hat{\sigma}^2 = \frac{1}{nd}\sum_{i=1}^{n} D^2(x_i) = \frac{1}{nd}\sum_{i=1}^{n} \|\phi(x_i) - a\|^2, \text{ where } a = \sum_{i=1}^{n} \alpha_i \phi(x_i)$$

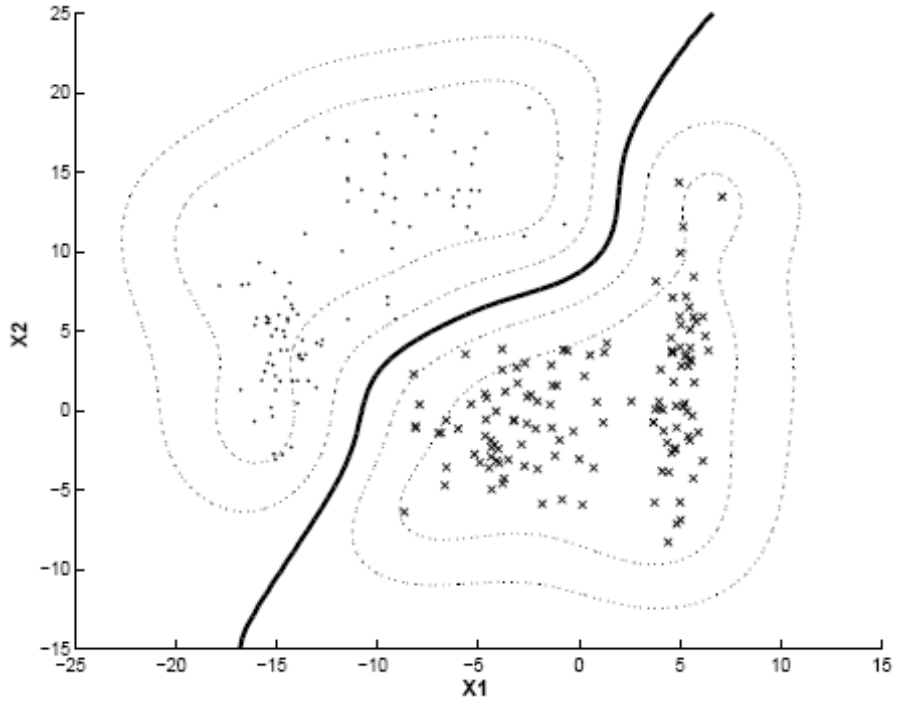
Support Vector Domain Density Description

SV-DDD



Support Vector Domain Density Description

- Hypersurface by domain density description



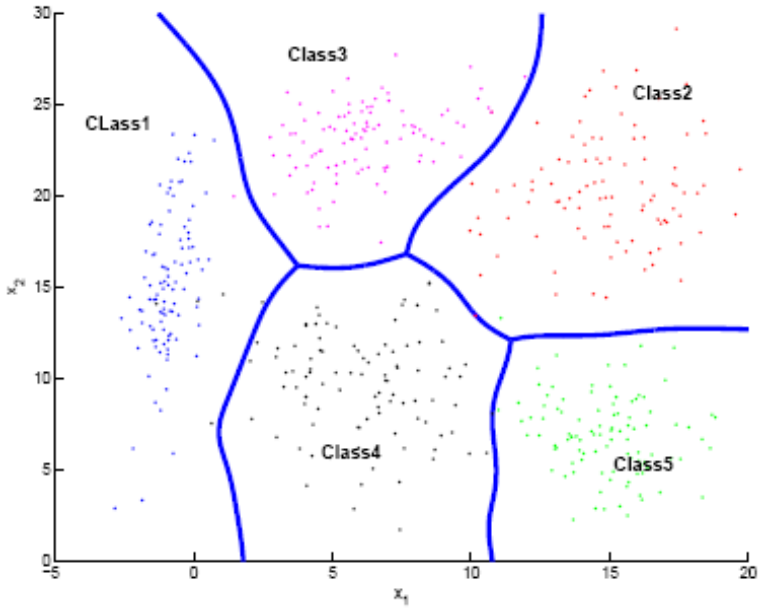
Experiments (Artificial data)

- Artificial data

  2D data that include 5 classes generated from unimodal PDF.

  Each class has training 100 samples and test sample 200 samples.

  We repeat classification experiments 10 times.



Experiments (UCI Data)

- Accuracy

Data set	OAA	OAO	DAG	HAH	BTS	parzen	proposed method
iris	97.33	97.33	96.67	97.33	97.33	96.67	97.33
glass	67.53	63.35	67.53	67.53	68.58	67.96	67.96
vowel	91.52	93.53	93.43	93.34	92.93	92.71	92.67
sonar	84.62	84.62	84.62	87.02	87.02	86.53	88.44
segment	94.45	94.35	94.90	94.56	93.24	94.81	94.81
ionosphere	90.73	90.73	90.73	90.60	90.60	94.02	94.59

- Training Speed

Data set	OAA	OAO	DAG	HAH	BTS	parzen	proposed method
iris	2.83	1.15	1.03	1.41	1.25	-	0.09
glass	11.55	3.16	3.22	3.48	3.67	-	0.07
vowel	1565	79.59	79.12	791	97.44	-	0.54
sonar	1.13	1.13	1.13	1.51	1.15	-	0.11
segment	4441	155.12	166.65	264.48	527.72	-	15.18
ionosphere	7.12	7.12	7.12	6.04	7.81	-	1.21

Experiments (Yale Database)

- Small dataset with many classes
- The number of data per class : 11

- Accuracy

Ratio	parzen	OAA	OAO	DAG	HAH	BTS	proposed method
4 : 7	91.21	89.71	91.02	91.32	86.35	72.38	90.98
5 : 6	91.72	90.24	91.21	90.99	89.17	76.67	91.24
6 : 5	91.85	90.71	91.73	91.56	89.22	77.33	92.17
7 : 4	93.33	91.94	92.13	92.63	91.48	85.33	93.33

- Training Speed

Ratio	parzen	OAA	OAO	DAG	HAH	BTS	proposed method
4 : 7	-	3.132	1.318	1.428	0.566	0.387	0.032
5 : 6	-	4.898	1.786	2.082	0.6667	0.542	0.032
6 : 5	-	7.321	2.359	2.394	0.917	0.872	0.033
7 : 4	-	10.536	3.123	3.085	1.683	1.594	0.035

Summary

- Constraint Convex Optimization
- linear/quadratic programming
- dual problem, KKT conditions
- Minimization techniques
  - Gradient Descent Minimization
  - Newton Minimization
  - Gauss Newton Minimization
  - (In)Equality Constraint Minimization
- Structural Risk Minimization
  - Support Vector Machine
    - https://www.csie.ntu.edu.tw/~cjlin/papers/bottou_lin.pdf
    - Bottou and Lin, Support Vector Machine Solvers
  - Support vector data description
- Error Backpropagation Learning for a Neural Network

Optimization (III): EBP, CNN

Jin Young Choi

Seoul National University

Outline

- Constraint Convex Optimization
- dual problem, KKT conditions
- Minimization techniques
  - Gradient Descent Minimization
  - Newton Minimization
  - Gauss Newton Minimization
  - (In)Equality Constraint Minimization
- Structural Risk Minimization
  - Support Vector Machine
    - https://www.csie.ntu.edu.tw/~cjlin/papers/bottou_lin.pdf
    - Bottou and Lin, Support Vector Machine Solvers
  - Support vector data description
- Error Backpropagation Learning for a Neural Network
- Convolution Neural Network

Backpropagation Learning Rule

- Empirical Risk Function:

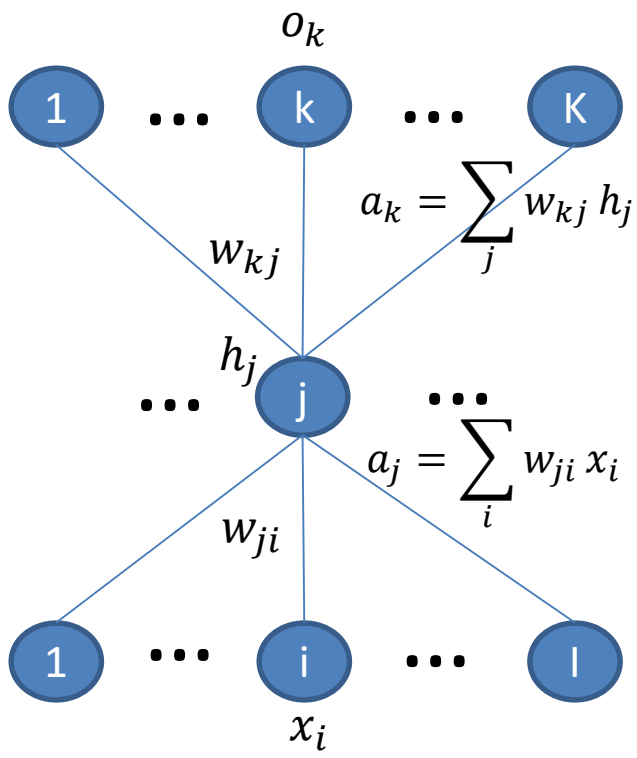
$$E(w) \equiv \frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2 = \frac{1}{2}||t - o(x,w)||^2$$

- Gradient descent for output layer:

$$\Delta w_{kj} = -\eta \frac{\partial E_d}{\partial w_{kj}}$$

- Chain rule:

$$\frac{\partial E_d}{\partial w_{kj}} = \frac{\partial E_d}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = \frac{\partial E_d}{\partial a_k} h_j$$



Backpropagation Learning Rule

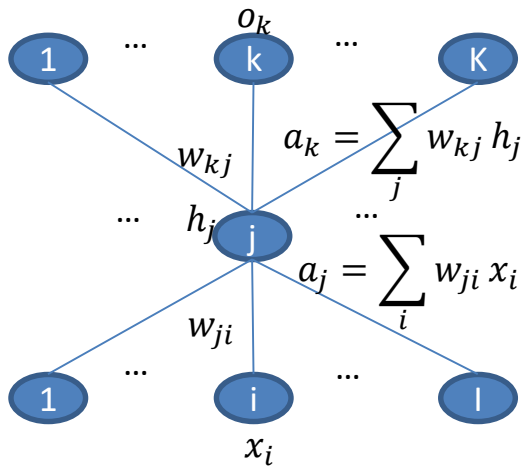
- To learn the regression problem, the linear activation function was used and the sum-of-squares error function was used as the loss function. Let's define the loss function as $E(w) = \frac{1}{2}||o(x,\ w) - t\ ||^2$. At this time, the k-th value of $o(x,\ w)$ is defined by $o_k = a_k = w_k^T h = \sum_j w_{kj} h_j$. Then find $\frac{\partial \mathrm{E}}{\partial a_k}$.

  Sol.)

  Since $E(w) = {}^1/_2 \sum_k (a_k - t_k)^2$, $\frac{\partial \mathrm{E}}{\partial a_k} = a_k - t_k = o_k - t_k = -(t_k - o_k) = -\delta_k$.

$$\frac{\partial E_d}{\partial w_{kj}} = \frac{\partial E_d}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = \frac{\partial E_d}{\partial a_k} h_j$$

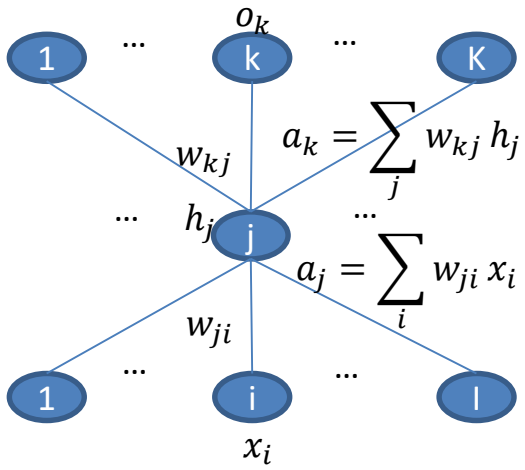
$$\Delta w_{kj} = -\eta \frac{\partial E_d}{\partial w_{kj}} = \eta \delta_k h_j$$



Backpropagation Learning Rule

- For multi-label classification (ex, output: 0110100), sigmoid activation function is used and the loss is defined by the cross entropy loss function:
  $E(w) = -\sum_k^K [t_k \log o_k(x,w) + (1-t_k)\log\big(1-o_k(x,w)\big)]$, where

$o_k = \sigma(a_k) = \frac{1}{1+e^{-a_k}}$. Then find $\frac{\partial \mathrm{E}}{\partial a_k}$.



Sol.)

Since $\frac{\partial o_k}{\partial a_k} = \sigma(a_k)\big(1-\sigma(a_k)\big) = o_k(1-o_k)$,

$$\frac{\partial \mathrm{E}}{\partial a_k} = -t_k \frac{1}{o_k}\frac{\partial o_k}{\partial a_k} - (1-t_k)\frac{-1}{1-o_k}\frac{\partial o_k}{\partial a_k}$$
$$= -t_k \frac{1}{o_k} o_k(1-o_k) - (1-t_k)\frac{-1}{1-o_k} o_k(1-o_k)$$
$$= -t_k(1-o_k) + (1-t_k)o_k = o_k - t_k = -(t_k - o_k) = -\delta_k$$

$$\frac{\partial E_d}{\partial w_{kj}} = \frac{\partial E_d}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = \frac{\partial E_d}{\partial a_k} h_j$$

$$\Delta w_{kj} = -\eta \frac{\partial E_d}{\partial w_{kj}} = \eta \delta_k h_j$$

Backpropagation Learning Rule

- For multi-class classification (ex, [0 0 0 1 0 0]), the softmax activation function is used and the loss is defined by the cross entropy loss function: $E(w) = -\sum_i^K t_i \log(o_i(x,w))$, where $o_k(x,w) = \frac{e^{a_k}}{\sum_j e^{a_j}}$. The target value $t_k \in \{0,1\}$ is labelled by 1 hot vector. Then find $\frac{\partial \mathrm{E}}{\partial a_k}$.

  Sol.)

$$\frac{\partial E_n}{\partial a_k} = \frac{\partial}{\partial a_k}\left(-\sum_i^K t_i \log\left(\frac{e^{a_i}}{\sum_j e^{a_j}}\right)\right) = \frac{\partial}{\partial a_k}\left(-\sum_i^K [t_i \log(e^{a_i}) - t_i \log\left(\sum_j e^{a_j}\right)]\right)$$

$$= \frac{\partial}{\partial a_k}\left(-\sum_i^K [t_i a_i - t_i \log\left(\sum_j e^{a_j}\right)]\right) = -t_k + \sum_i t_i \frac{e^{a_k}}{\sum_j e^{a_j}}$$

$$= -t_k + \frac{e^{a_k}}{\sum_j e^{a_j}} \sum_i t_i = o_k - t_k = -(t_k - o_k) = -\delta_k$$

$$\frac{\partial E_d}{\partial w_{kj}} = \frac{\partial E_d}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = \frac{\partial E_d}{\partial a_k} h_j$$

$$\Delta w_{kj} = -\eta \frac{\partial E_d}{\partial w_{kj}} = \eta \delta_k h_j$$

Backpropagation Learning Rule

- Empirical Risk Function:

$E_d(w)$

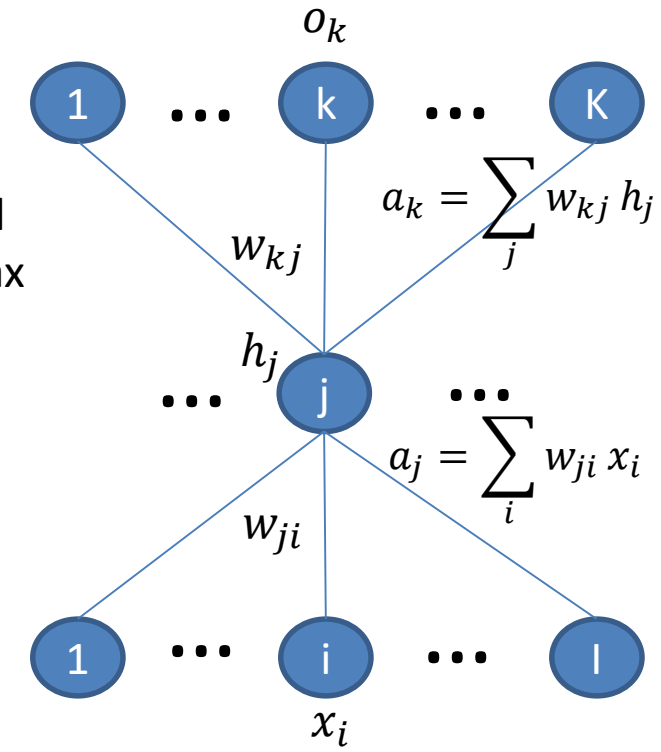
Regression: L_2, linear
01001101: cross-entropy, sigmoid
00001000: cross-entropy, soft-max

- Gradient descent for hidden layer:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

- Chain rule:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial a_j} x_i$$

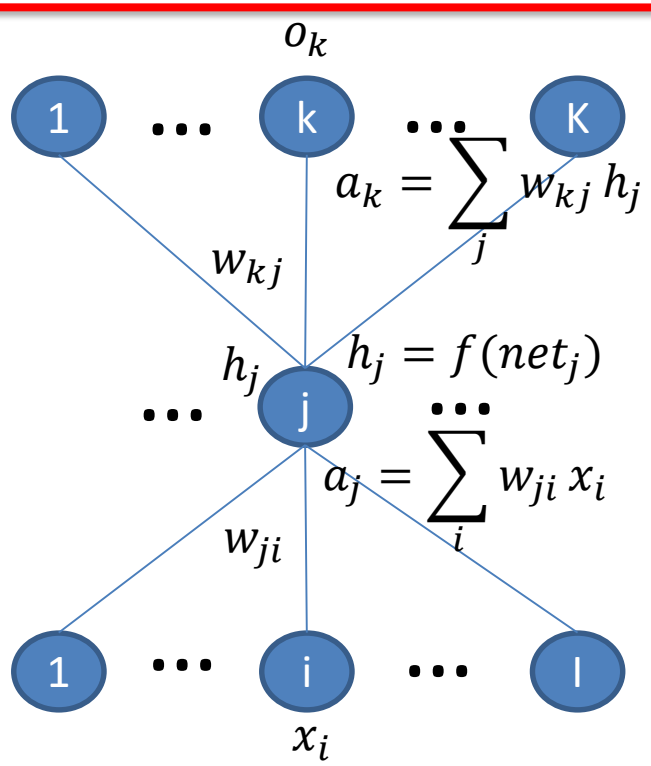


Backpropagation Learning Rule

- Chain rule:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial a_j} x_i, \qquad \frac{\partial E_d}{\partial a_k} = -\delta_k$$

$$\begin{aligned}
\frac{\partial E_d}{\partial a_j} &= \sum_{k \in outputs} \frac{\partial E_d}{\partial a_k} \frac{\partial a_k}{\partial h_j} \frac{\partial h_j}{\partial a_j} \\
&= \sum_{k \in outputs} -\delta_k \frac{\partial a_k}{\partial h_j} \frac{\partial h_j}{\partial a_j} \\
&= \sum_{k \in outputs} -\delta_k w_{kj} \frac{\partial h_j}{\partial a_j} \\
&= \sum_{k \in outputs} -\delta_k w_{kj} f'(a_j) \\
&= -\delta_j
\end{aligned}$$



$$\Delta w_{ji} = \eta \delta_j x_i,$$

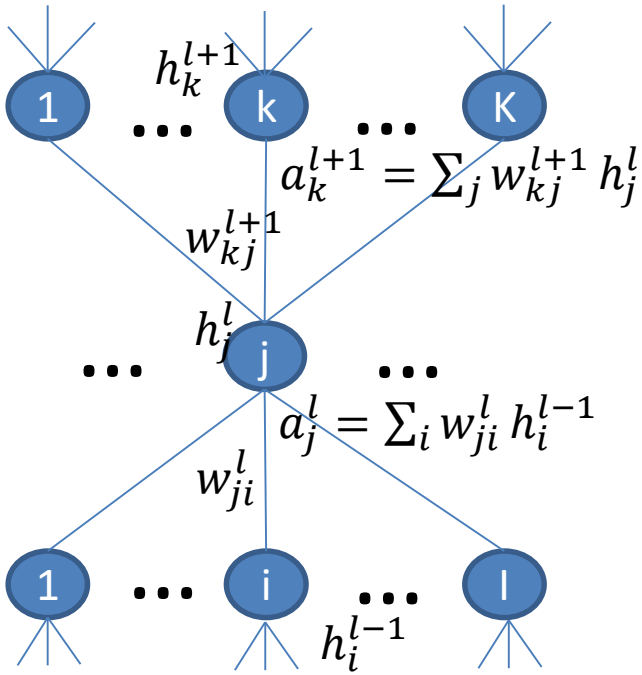
$$\delta_j = f'(a_j) \sum_{k \in outputs} \delta_k w_{kj}$$

Backpropagation Learning Rule

$$\Delta w_{ji}^{l} = \eta \delta_j^l h_i^{l-1},$$
$$\delta_j^l = f'(a_j^{l+1}) \sum_{k \in l+1 layer} \delta_k^{l+1} w_{kj}^{l+1}$$

$$\delta_k^L = -\frac{\partial E_d}{\partial a_k} = t_k - o_k$$

Regression: L_2, linear
01001101: cross-entropy, sigmoid
00001000: cross-entropy, soft-max



Matrix Form (Backward. EBP)

$$\Delta W^l = \eta \delta^l {h^{l-1}}^T + \rho \Delta W^{l(old)}$$
$$\delta^l = Diag[f'(net^l)] {W^{l+1}}^T \delta^{l+1}$$

Matrix Form (Forward)

$$h^0 = x$$
$$h^{l+1} = Diag[f] \circ W^{l+1} h^l$$

Error Back Propagation rule

2-layer Neural Network:

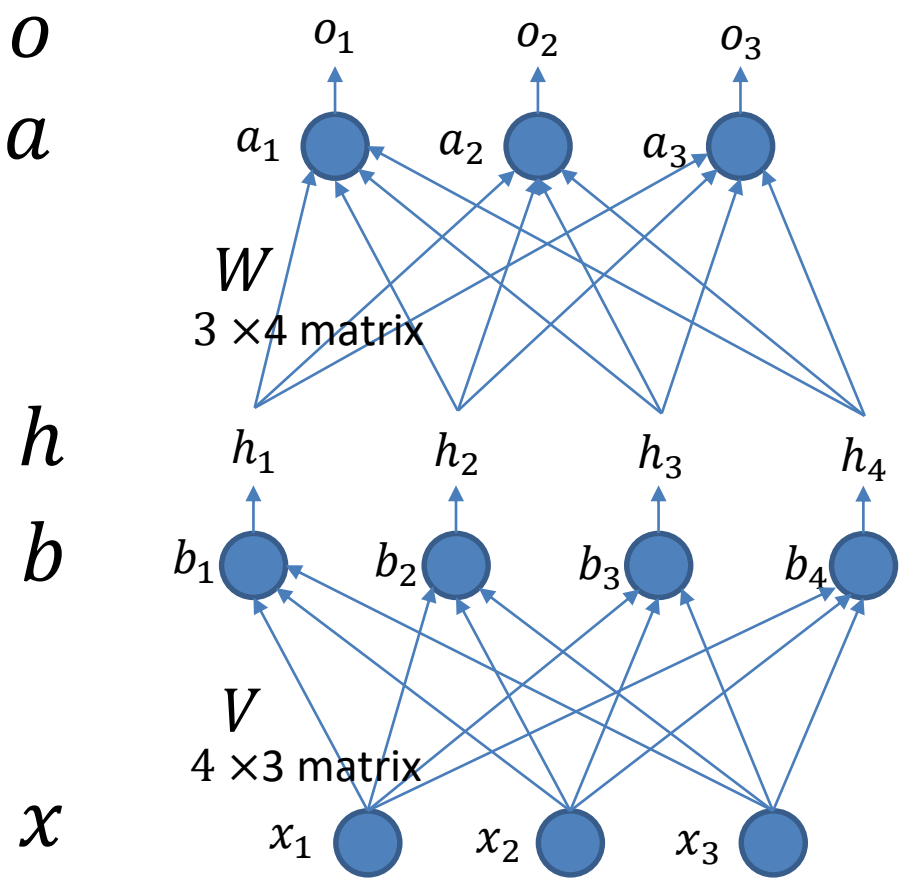
x: 3 ×1 input vector

V: 4 ×3 weight matrix

h: 4 ×1 hidden feature

W: 3 ×4 weight matrix

o: 3 ×1 output vector



Error Back Propagation rule

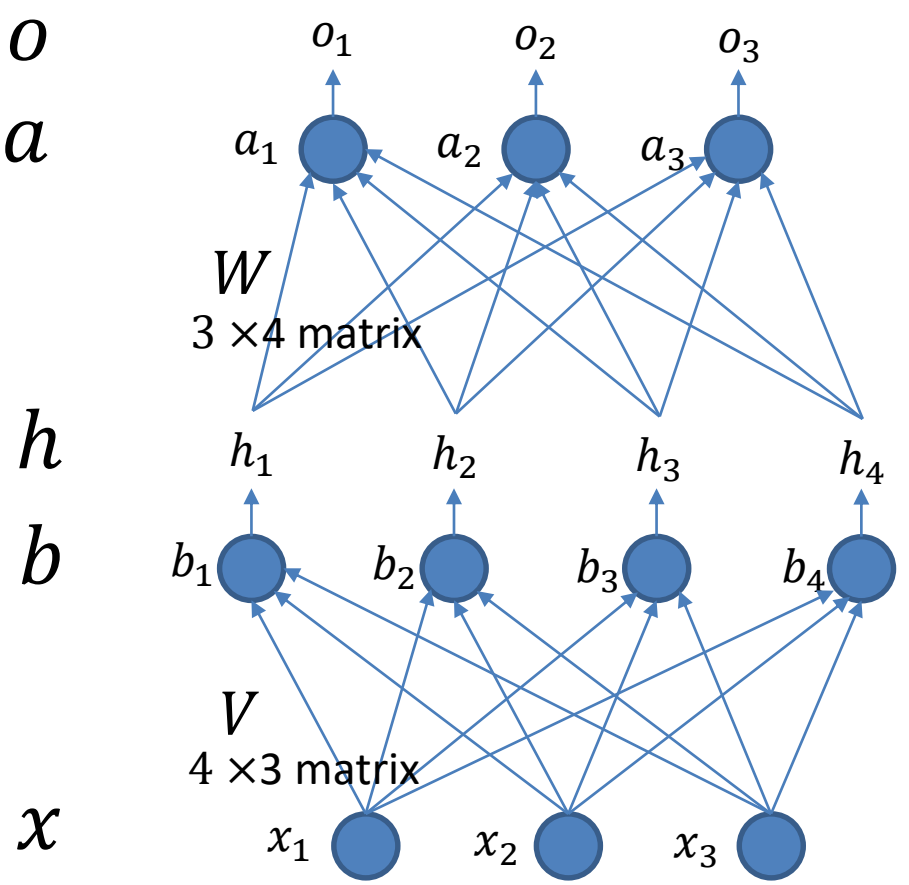
Forward Pass (first layer):

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \\ v_{41} & v_{42} & v_{43} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = Vx$$

$$h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = \begin{bmatrix} r(b_1) \\ r(b_2) \\ r(b_3) \\ r(b_4) \end{bmatrix}$$

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = Wh$$

$$o = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \end{bmatrix} = \begin{bmatrix} \sigma(a_1) \\ \sigma(a_2) \\ \sigma(a_3) \end{bmatrix}$$



Error Back Propagation rule

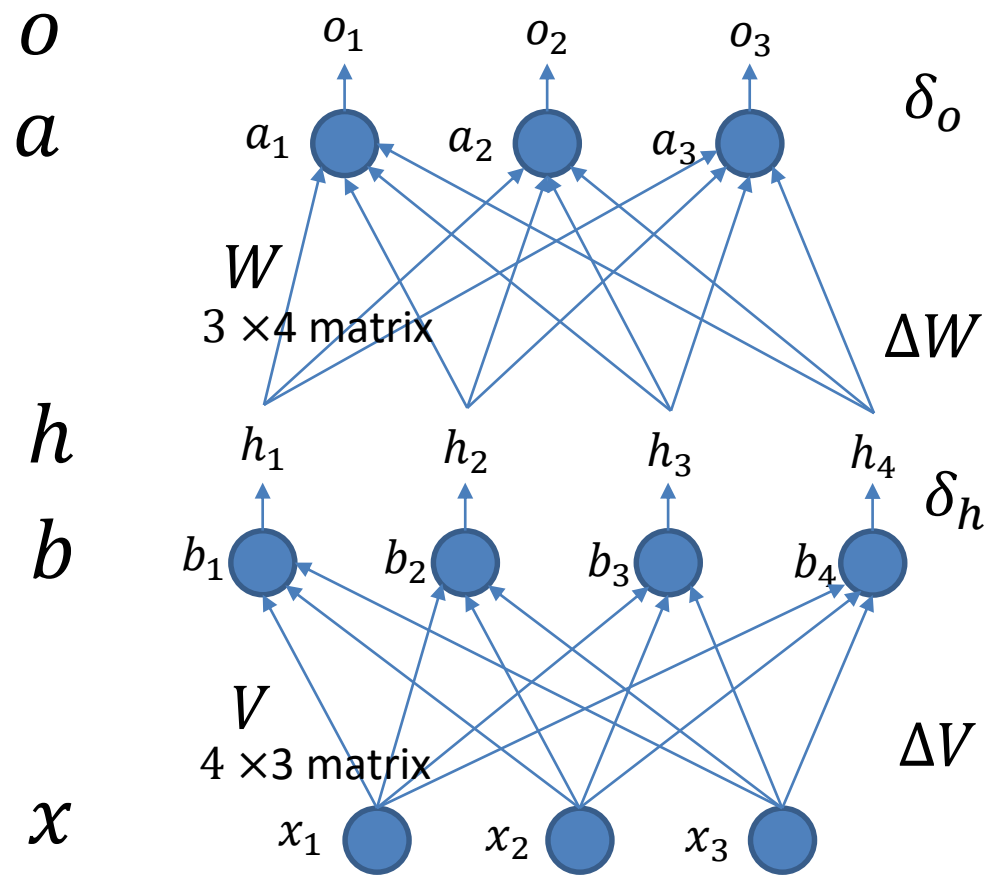
Backward Pass (second layer):

$$\delta_o = \begin{bmatrix} t_1 - o_1 \\ t_2 - o_2 \\ t_3 - o_3 \end{bmatrix} = \begin{bmatrix} \delta_{o_1} \\ \delta_{o_2} \\ \delta_{o_3} \end{bmatrix}$$

$$\Delta W = \eta \delta_o h^T = \eta \begin{bmatrix} \delta_{o_1} \\ \delta_{o_2} \\ \delta_{o_3} \end{bmatrix} [h_1 \quad h_2 \quad h_3 \quad h_4]$$

3 ×4 matrix 3 ×1 1 ×4

$$W^{new} = W^{old} + \Delta W$$



Error Back Propagation rule

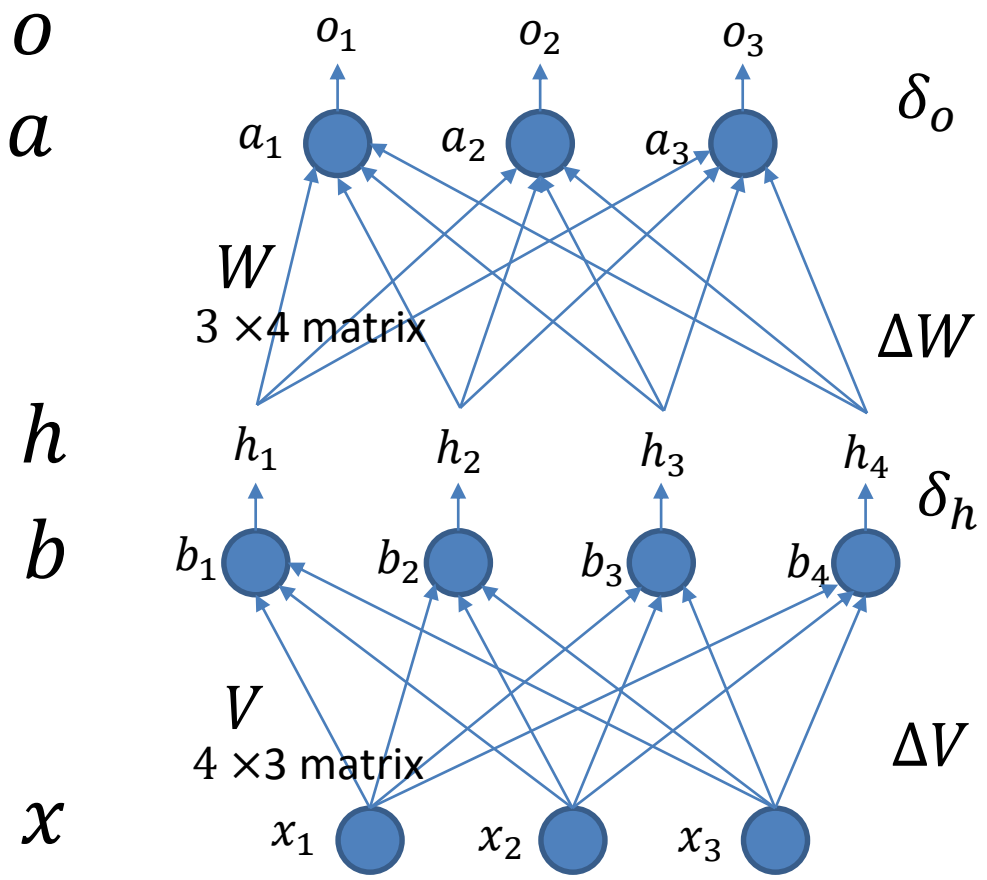
Backward Pass (first layer):

$$\begin{bmatrix}\delta_{h_1}\\ \delta_{h_2}\\ \delta_{h_3}\\ \delta_{h_4}\end{bmatrix} = \begin{bmatrix}r'(b_1)\bar{\delta}_{h_1}\\ r'(b_2)\bar{\delta}_{h_2}\\ r'(b_2)\bar{\delta}_{h_3}\\ r'(b_2)\bar{\delta}_{h_3}\end{bmatrix}, \quad \begin{bmatrix}\bar{\delta}_{h_1}\\ \bar{\delta}_{h_2}\\ \bar{\delta}_{h_3}\\ \bar{\delta}_{h_3}\end{bmatrix} = \begin{bmatrix}w_{11} & w_{21} & w_{31}\\ w_{12} & w_{22} & w_{32}\\ w_{13} & w_{23} & w_{33}\\ w_{14} & w_{24} & w_{34}\end{bmatrix}\begin{bmatrix}\delta_{o_1}\\ \delta_{o_2}\\ \delta_{o_3}\end{bmatrix}$$

$$\Delta V = \eta\delta_h x^T = \eta\begin{bmatrix}\delta_{h_1}\\ \delta_{h_2}\\ \delta_{h_3}\\ \delta_{h_4}\end{bmatrix}[x_1 \quad x_2 \quad x_3]$$

4 ×3 matrix 4 ×1 1 ×3

$$V^{new} = V^{old} + \Delta V$$



Error Back Propagation rule

Forward Pass :

$$b = Vx,\ h = r(b)$$

$$a = Wh,\ o = \sigma(a)$$

Backward Pass :

$$E = \frac{1}{2}\|t - o\|_2^2$$

$$\nabla_W E = (t - o)(-h^T) = -\delta_o h^T$$

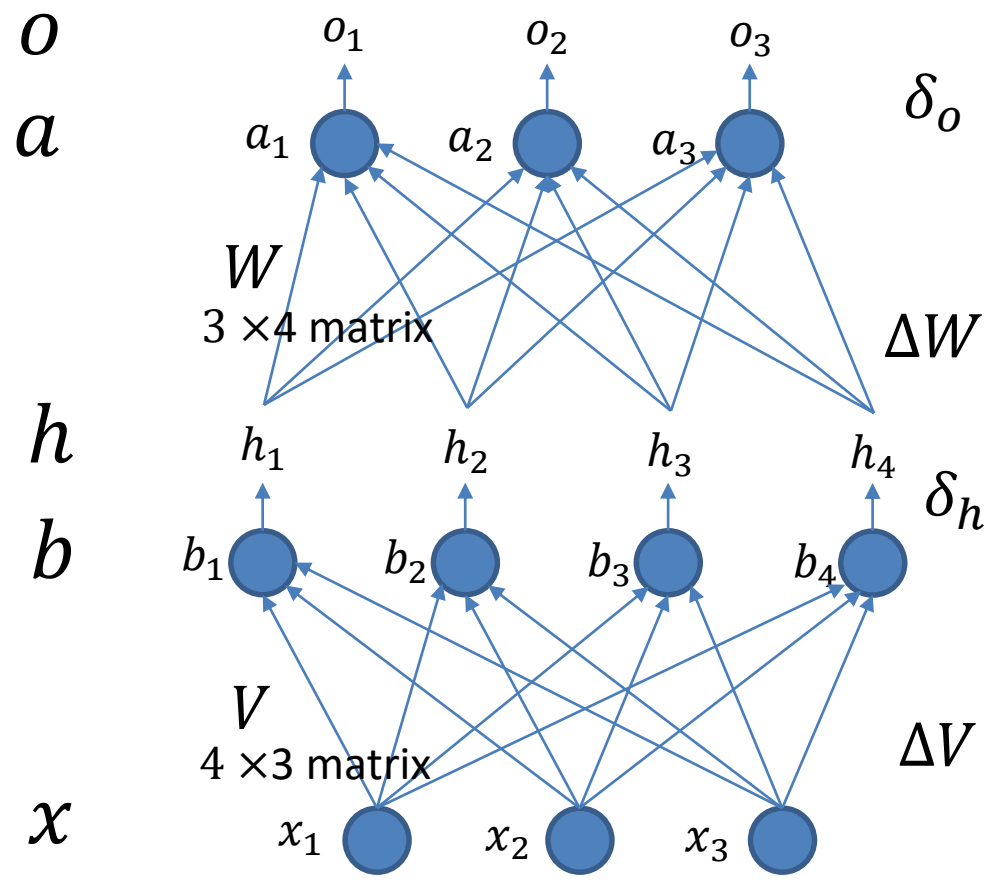
$$W^{new} = W^{old} + \eta\delta_o h^T$$

$$\nabla_V E = -W^T(t - o)(\nabla_V h)$$

$$= -(Diag(r'(b)W^T\delta_o x^T)$$

$$= -\delta_h x^T$$

$$V^{new} = V^{old} + \eta\delta_h x^T$$



Convolution Neural Network

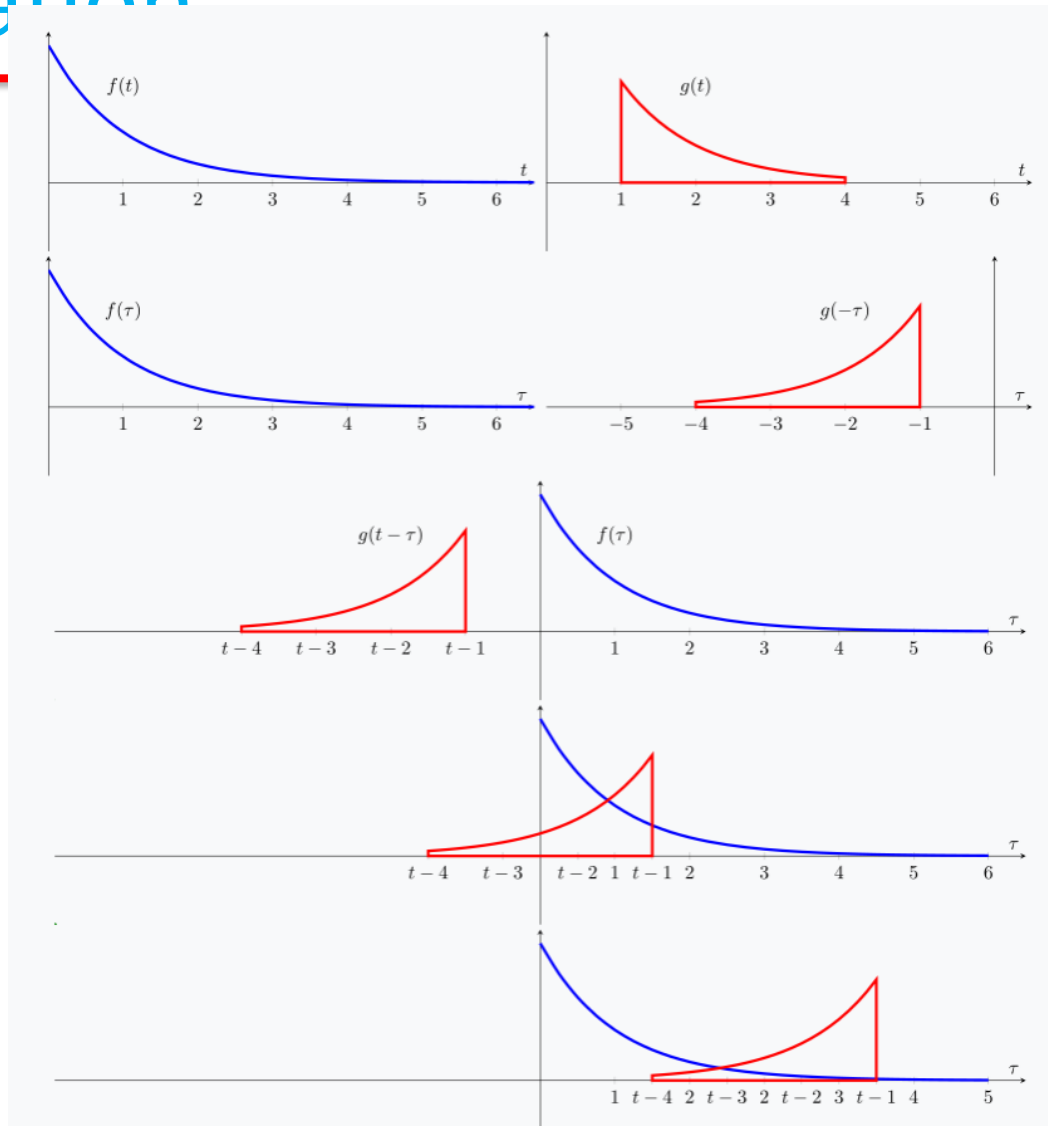
Convolution and Cross-Correlation

- **Convolution Integral (Temporal)**

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau$$

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(t-\tau) g(\tau)\, d\tau$$

$$L \circ (f * g) = F(s)G(s)$$



Convolution and Cross-Correlation

- **Convolution Sum (Temporal)**

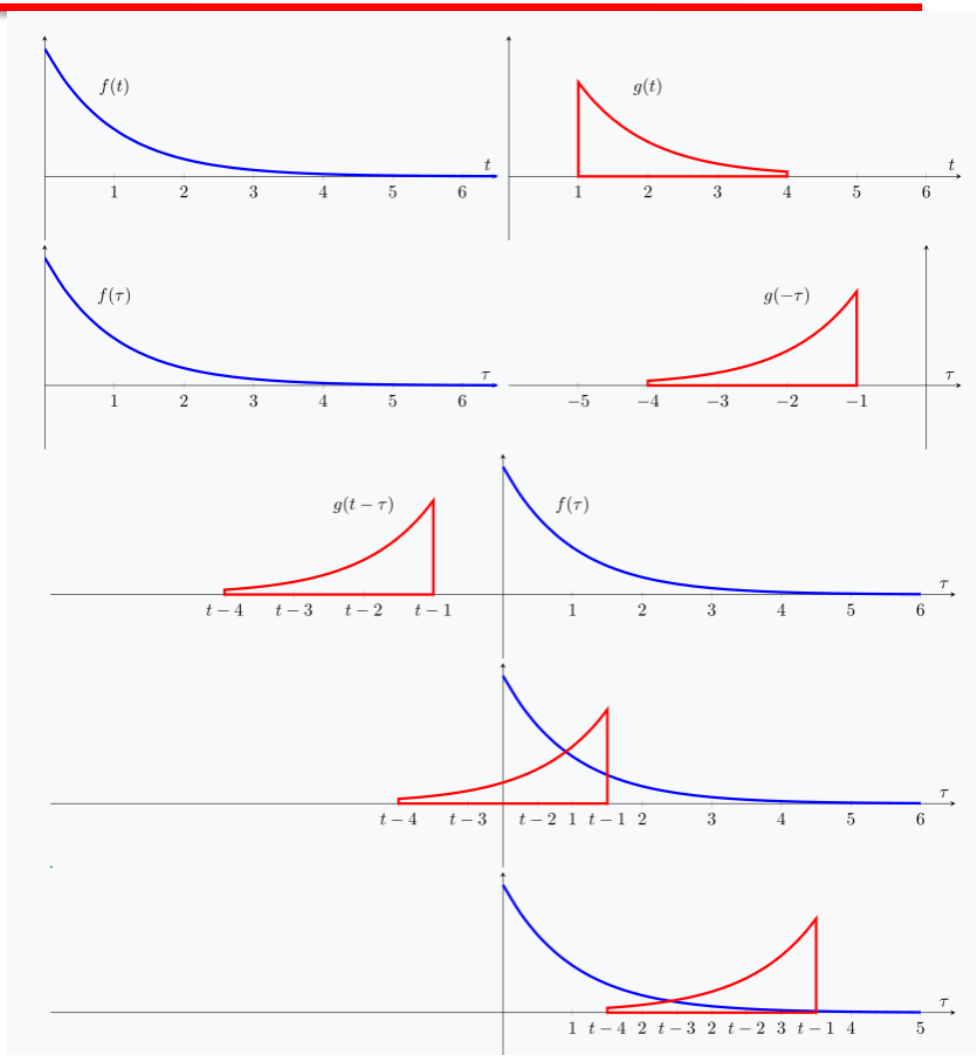
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[n-m]g[m]$$

- Cross correlation (without kernel flipping)

$\rightarrow f * g[n] = \sum_{m=-\infty}^{\infty} f[m]g[m-n]$

- Many CNNs implement cross-correlation but call it convolution (without kernel flipping)



Convolution and Cross-Correlation

Circular Convolution Sum

$$h_k = f * g = \sum_{i=0}^{n-1} f_i g_{k-i}$$

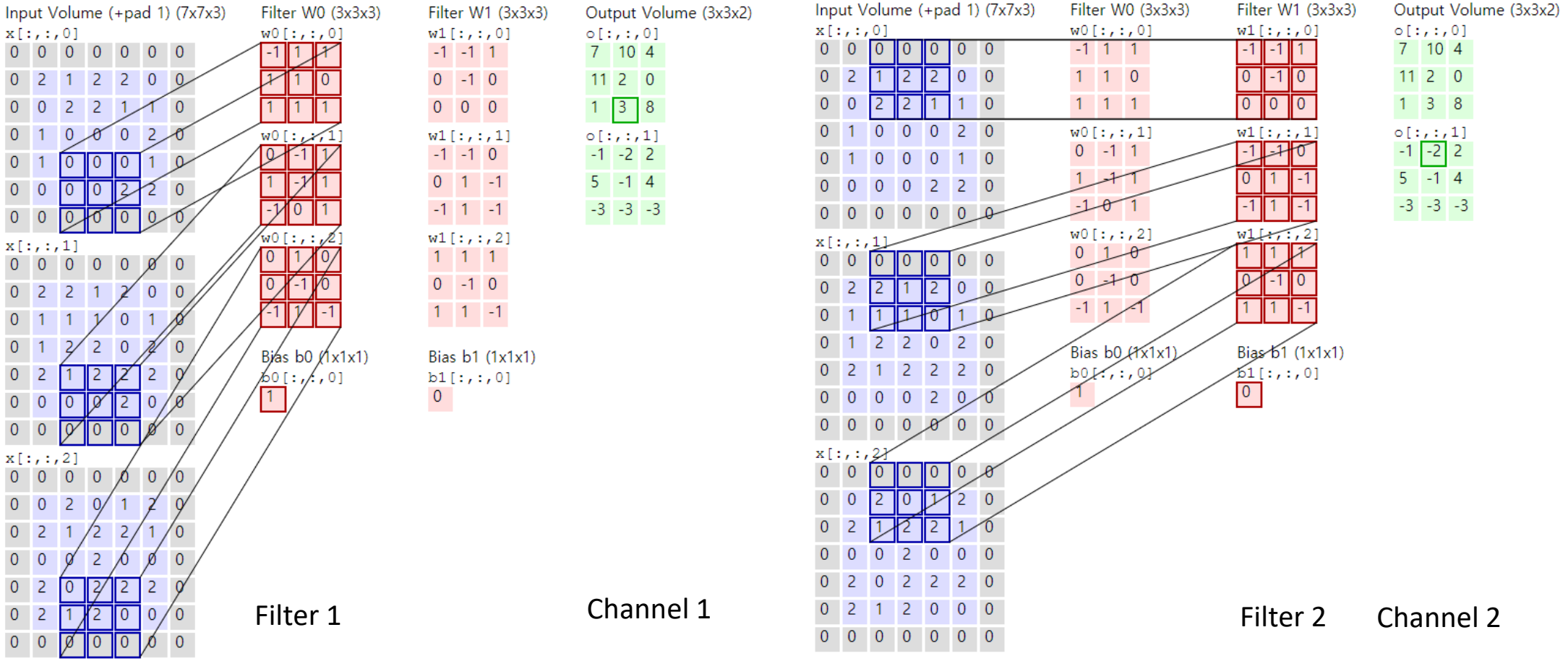
$$g \triangleq (\ldots, g_{n-1}, g_0, g_1, \cdots, g_{n-1}, g_0, g_1, \cdots, g_{n-1}, \ \ldots)$$
$$f \triangleq (f_0, f_1, \cdots, f_{n-1})$$
$$h \triangleq (\ldots, h_{n-1}, h_0, h_1, \ldots, h_{n-1}, h_0, h_1, \ldots, h_{n-1}, \ \ldots)$$

$$f * g = \begin{bmatrix} g_0 & g_{n-1} & \cdots & g_2 & g_1 \\ g_1 & g_0 & g_{n-1} & \cdots & g_2 \\ \vdots & g_1 & g_0 & \ddots & \vdots \\ g_{n-2} & \vdots & \ddots & \ddots & g_{n-1} \\ g_{n-1} & g_{n-2} & \cdots & g_1 & g_0 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-2} \\ f_{n-1} \end{bmatrix}$$

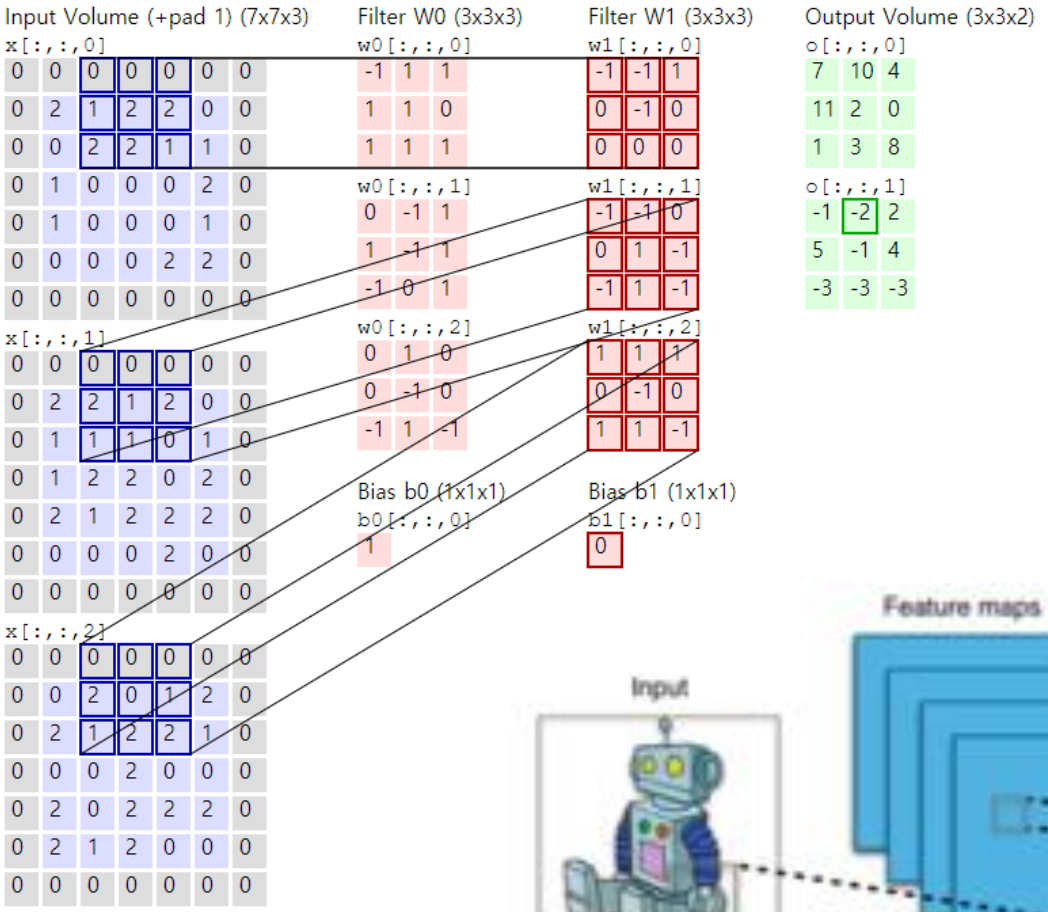
Circulant Matrix

Convolution and Cross-Correlation

- **Convolution Sum (Spatial)**



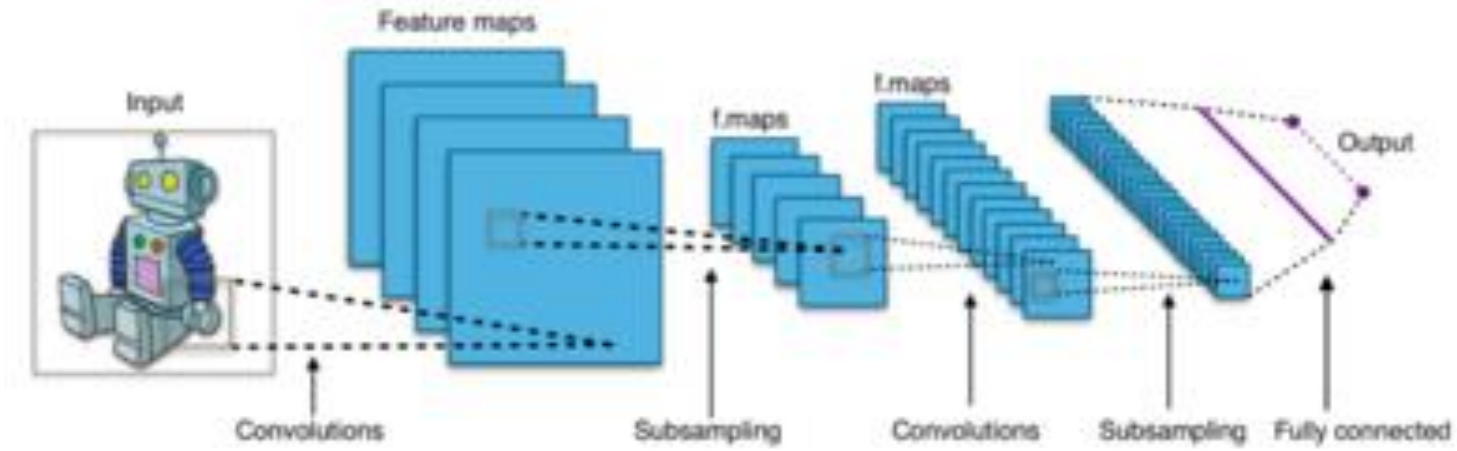
Convolutional Neural Networks (CNNs)



- Three key ideas behind CNN
  - Local connectivity
  - Invariance to location
  - Invariance to translation

Convolution layer

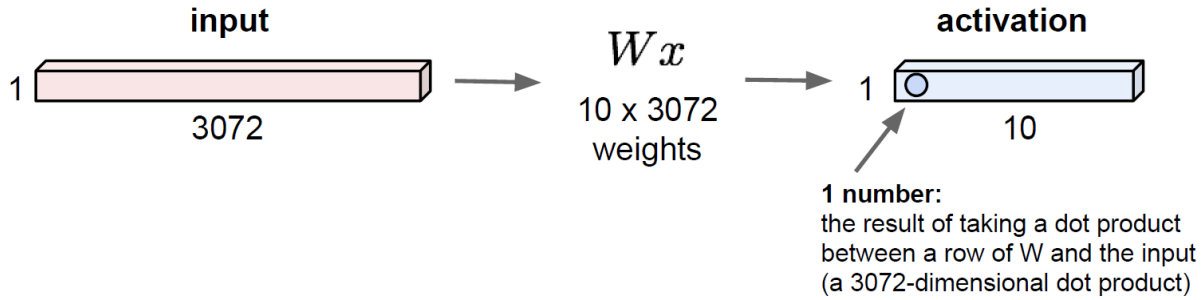
Pooling layer



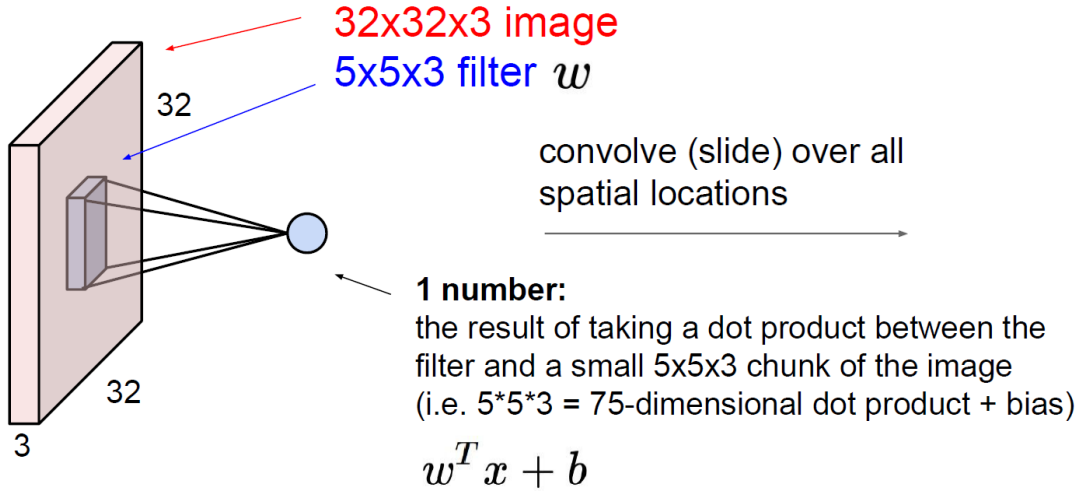
Fully Connected Layer and Convolution Layer

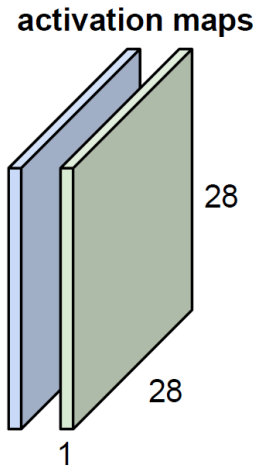
- Fully connected layer

32x32x3 image -> stretch to 3072 x 1



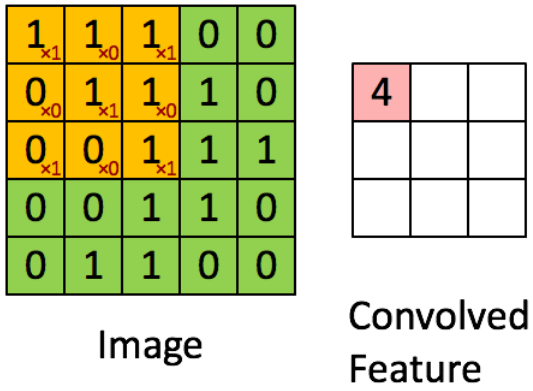
- Convolution

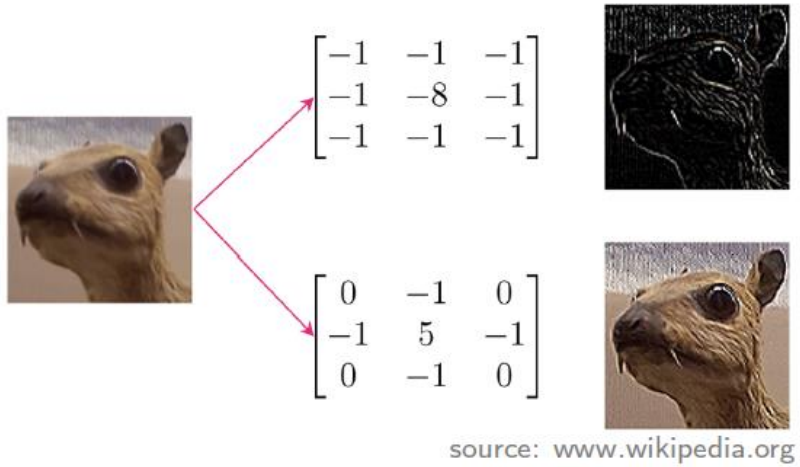




Convolution Layer

- Input is convolved with a set of filters, giving feature maps (without kernel flipping)

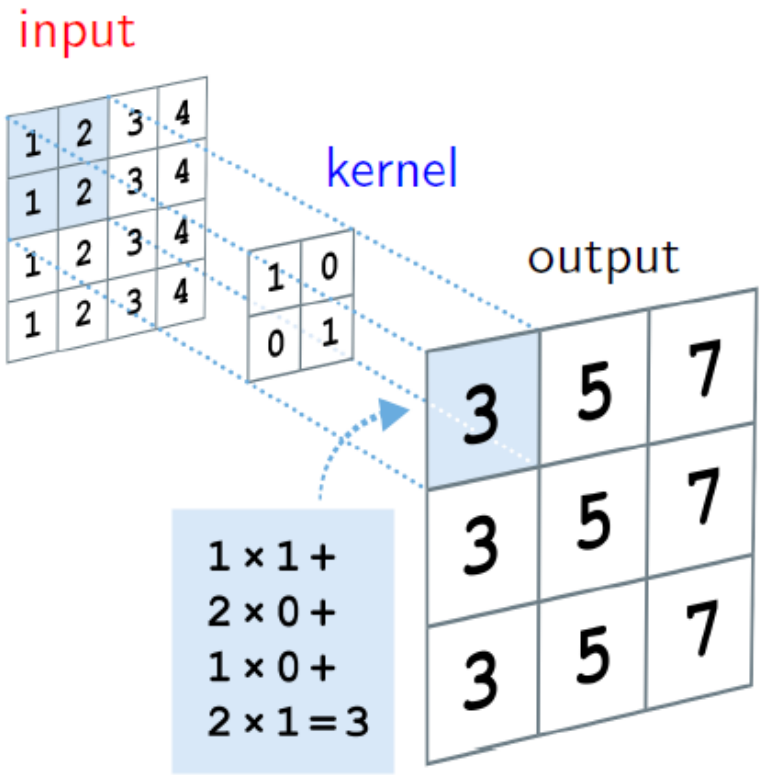




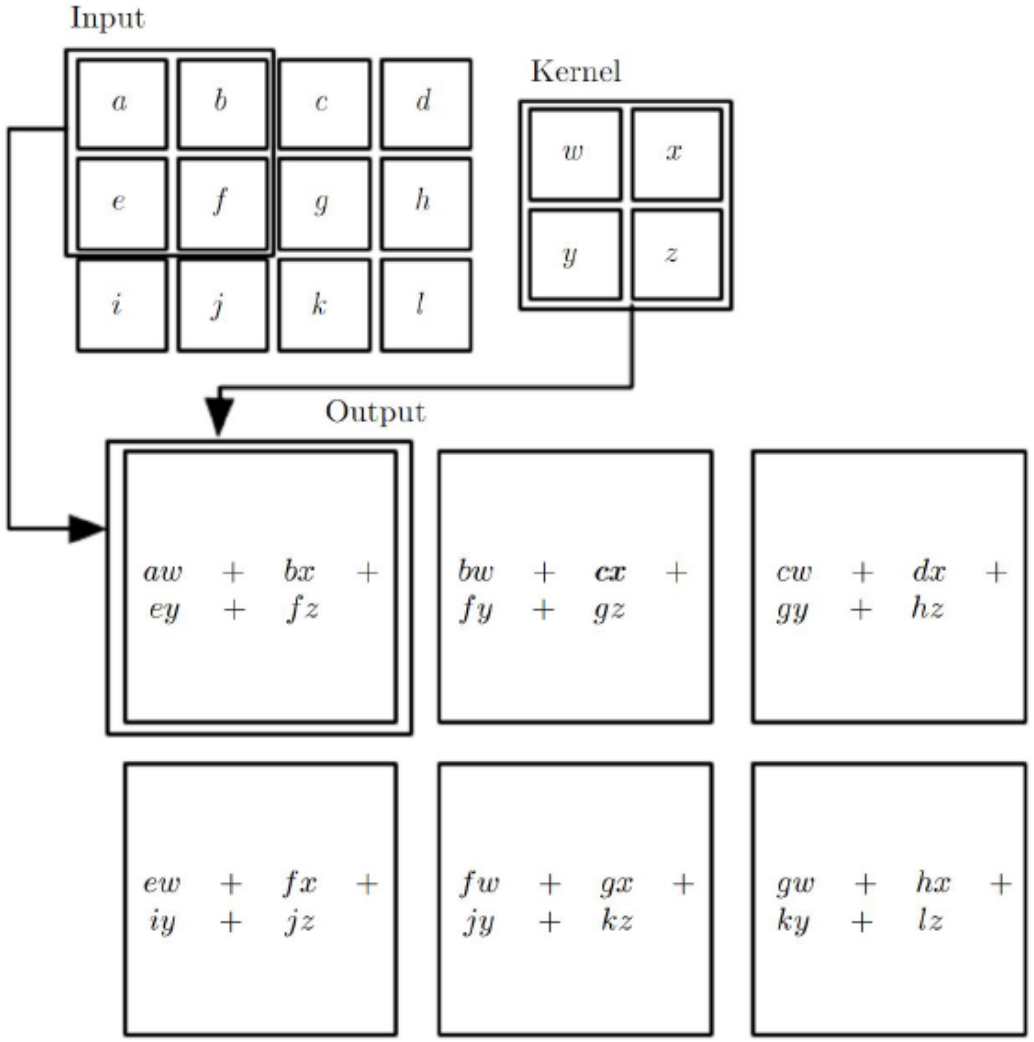
source: www.wikipedia.org

- Filter (a.k.a. kernel or convolution matrix)
  - Different filters extract different features (e.g. edges)
  - Filter weights: trained with data
  - Weight sharing & local connectivity

Convolution Example



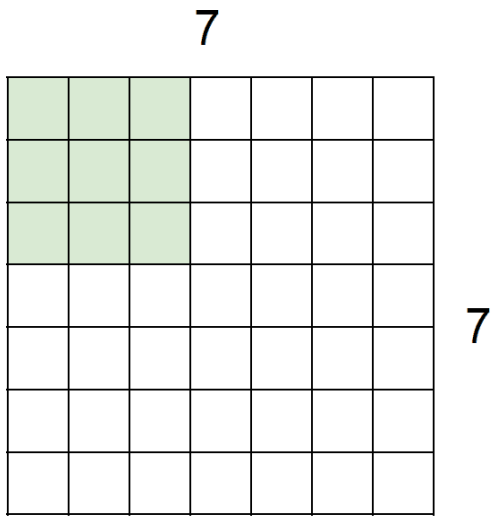
source: [Angermueller et al., 2016]



source: [Goodfellow et al., 2016]

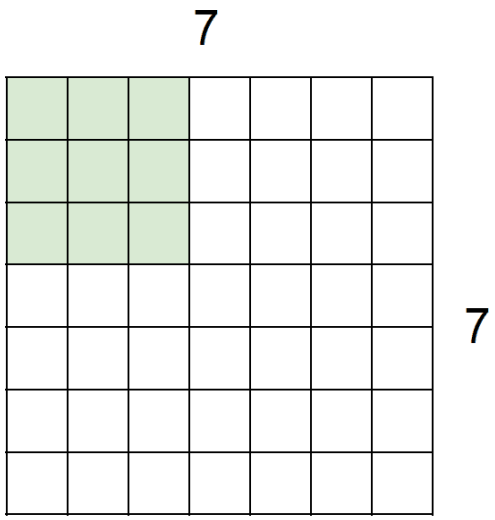
Convolution with Stride

- The number of pixels between adjacent receptive fields
  = down-sampling the output of full convolution function



7x7 input
3x3 filter with stride 1

➔ 5x5 output



7x7 input
3x3 filter with stride 2

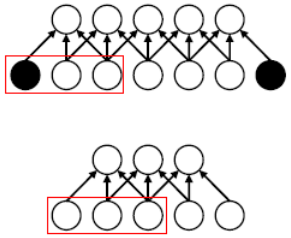
➔ 3x3 output

Zero-padding

- Implicitly zero-pad input to make it wider

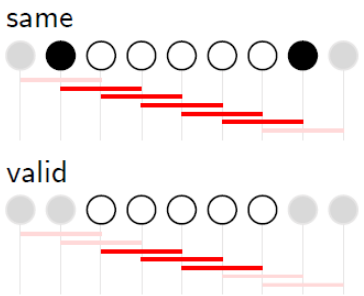
type	output		# zeros padded		
			left	right	total
same	m	k even	$\lfloor \frac{k-1}{2} \rfloor$	$\lfloor \frac{k-1}{2} \rfloor + 1$	$k-1$
		k odd	$\frac{k-1}{2}$	$\frac{k-1}{2}$	
valid	$m-(k-1)$		0	0	0

(m: input width; k: kernel width; $s = 1$)

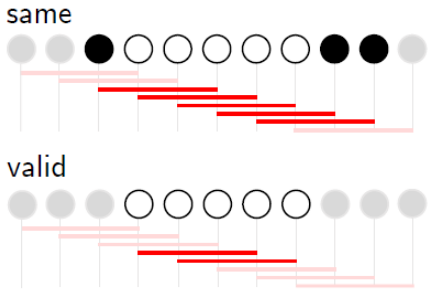


- Examples

$m = 5, k = 3$



$m = 5, k = 4$

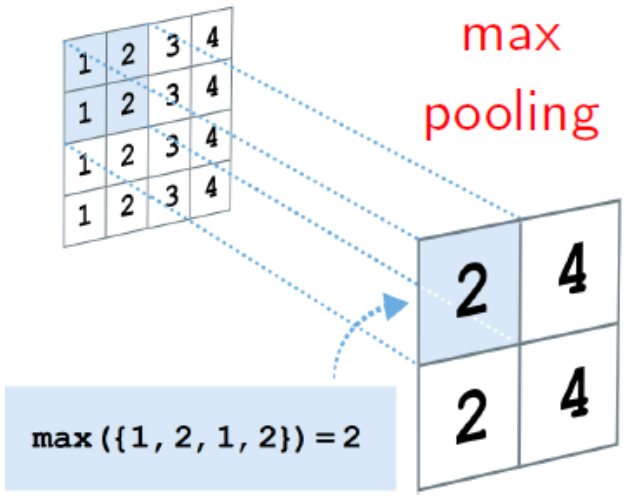


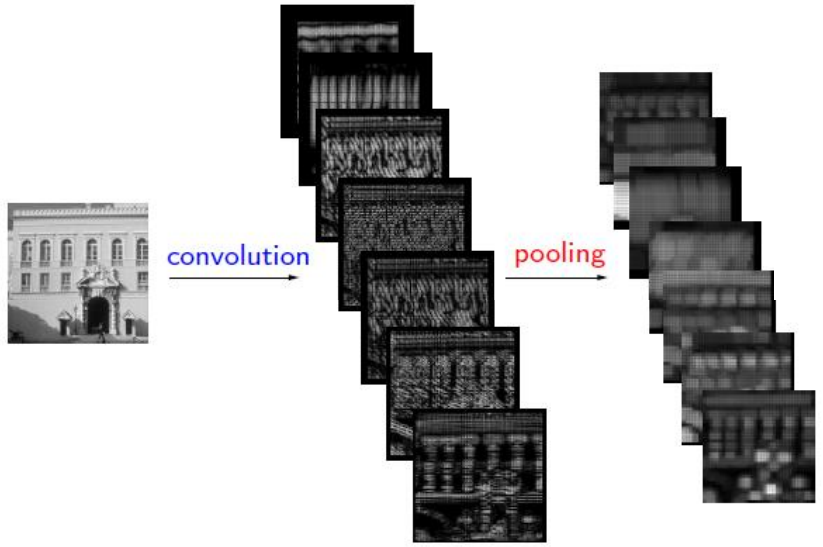
Pooling Layer

- Nonlinear down-sampling

  Aggregates statistics of local features (with max or average operation)

  Reduced variance: provides invariance to local transformations





source: [Angermueller et al., 2016, Thériault et al., 2013]

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

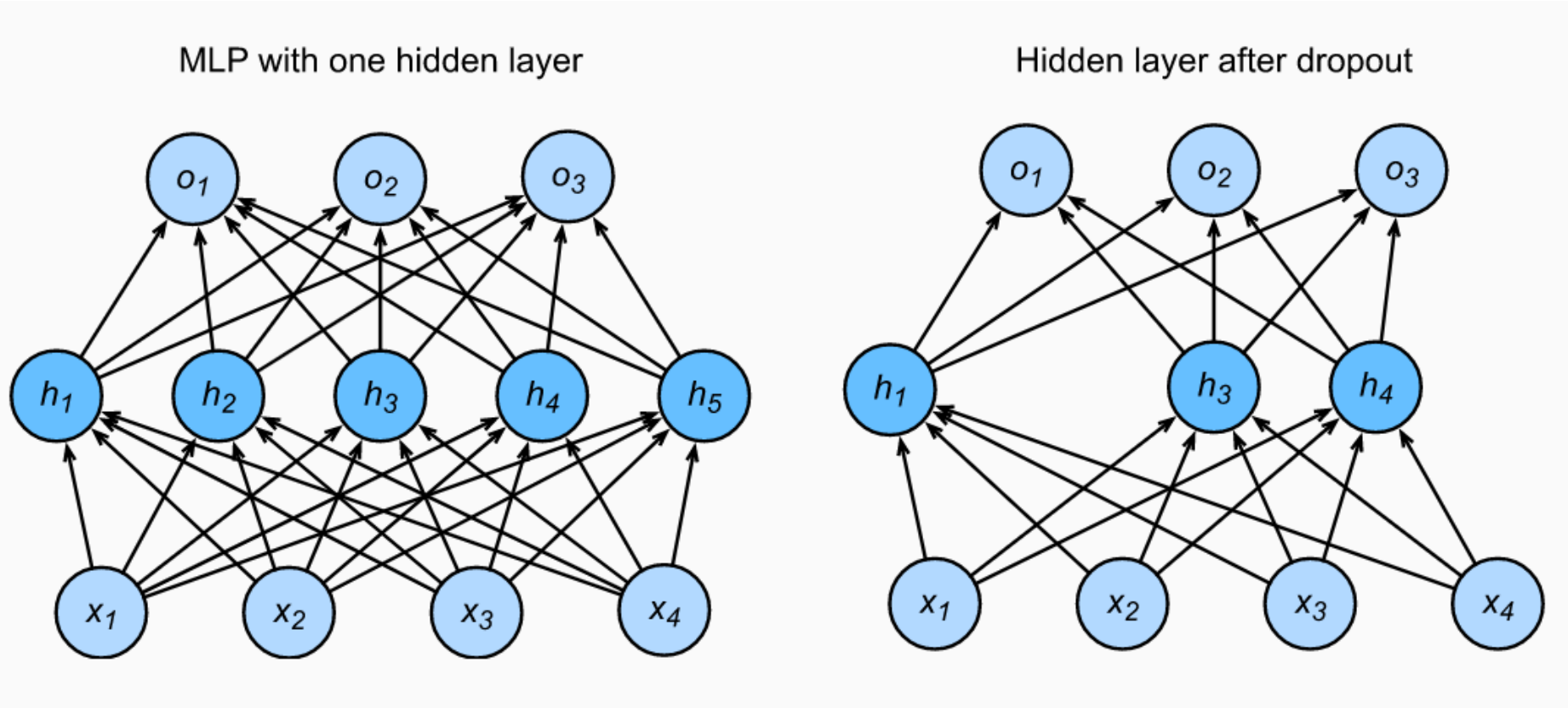
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

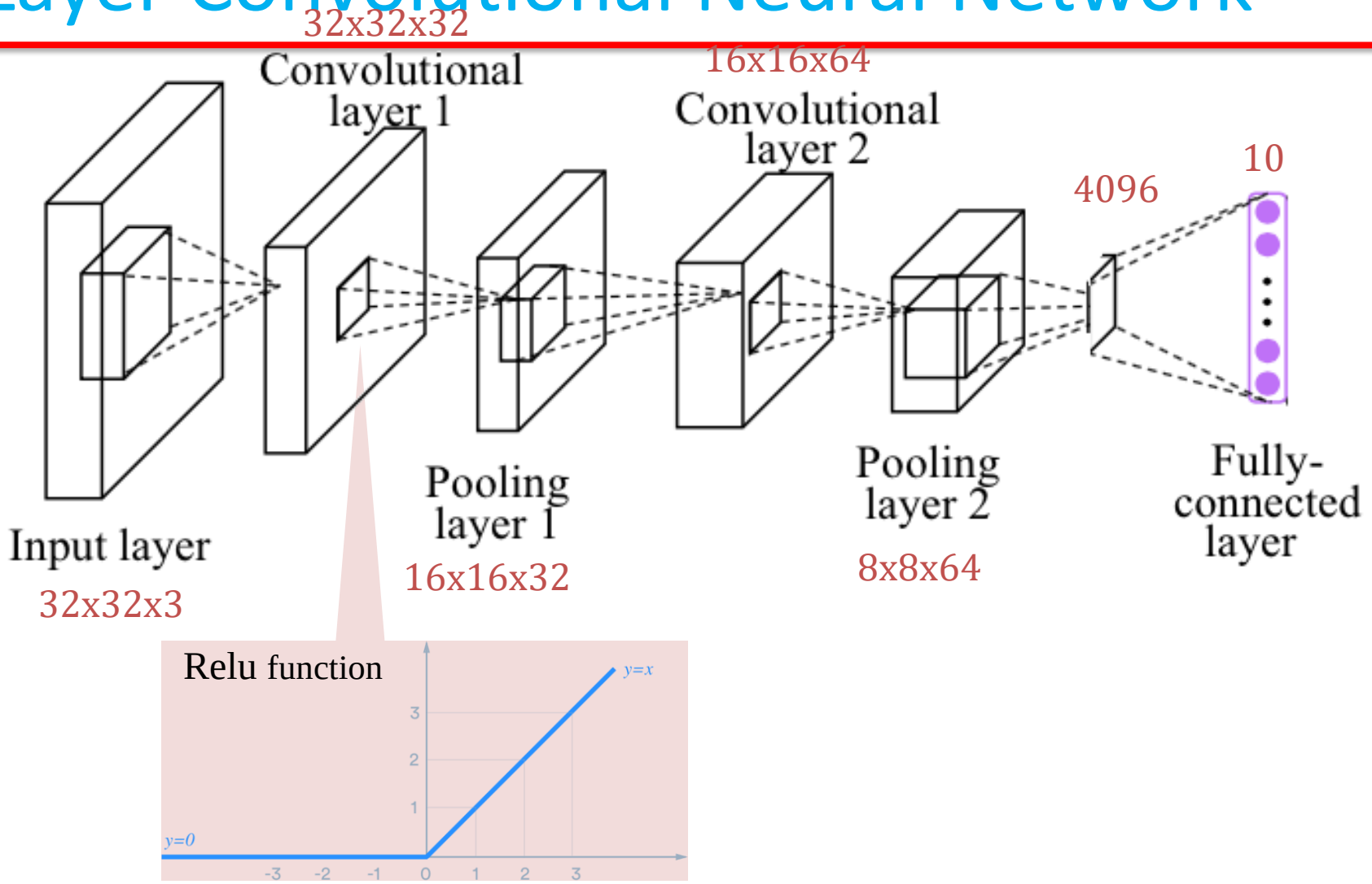
https://sacko.tistory.com/44

Dropout to mitigate overfitting

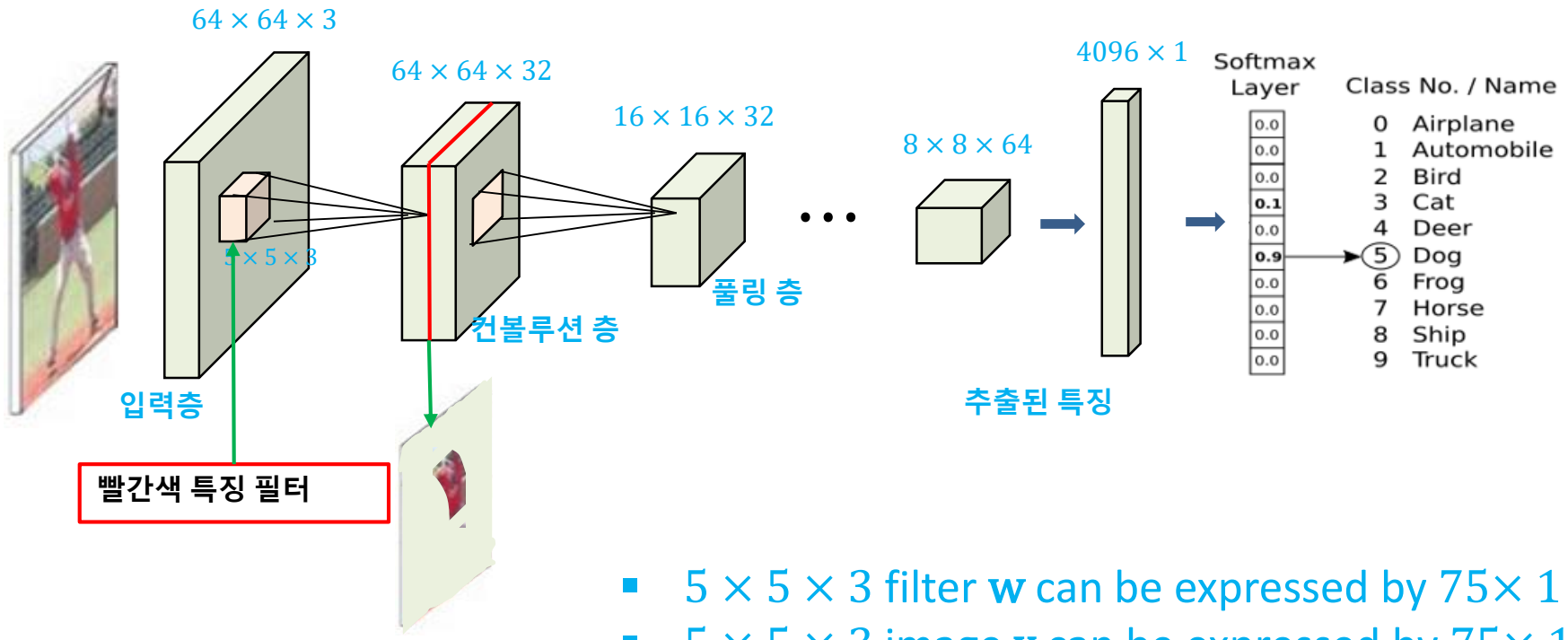


https://ko.d2l.ai/chapter_deep-learning-basics/dropout.html

Two Layer Convolutional Neural Network



Two Layer Convolutional Neural Network

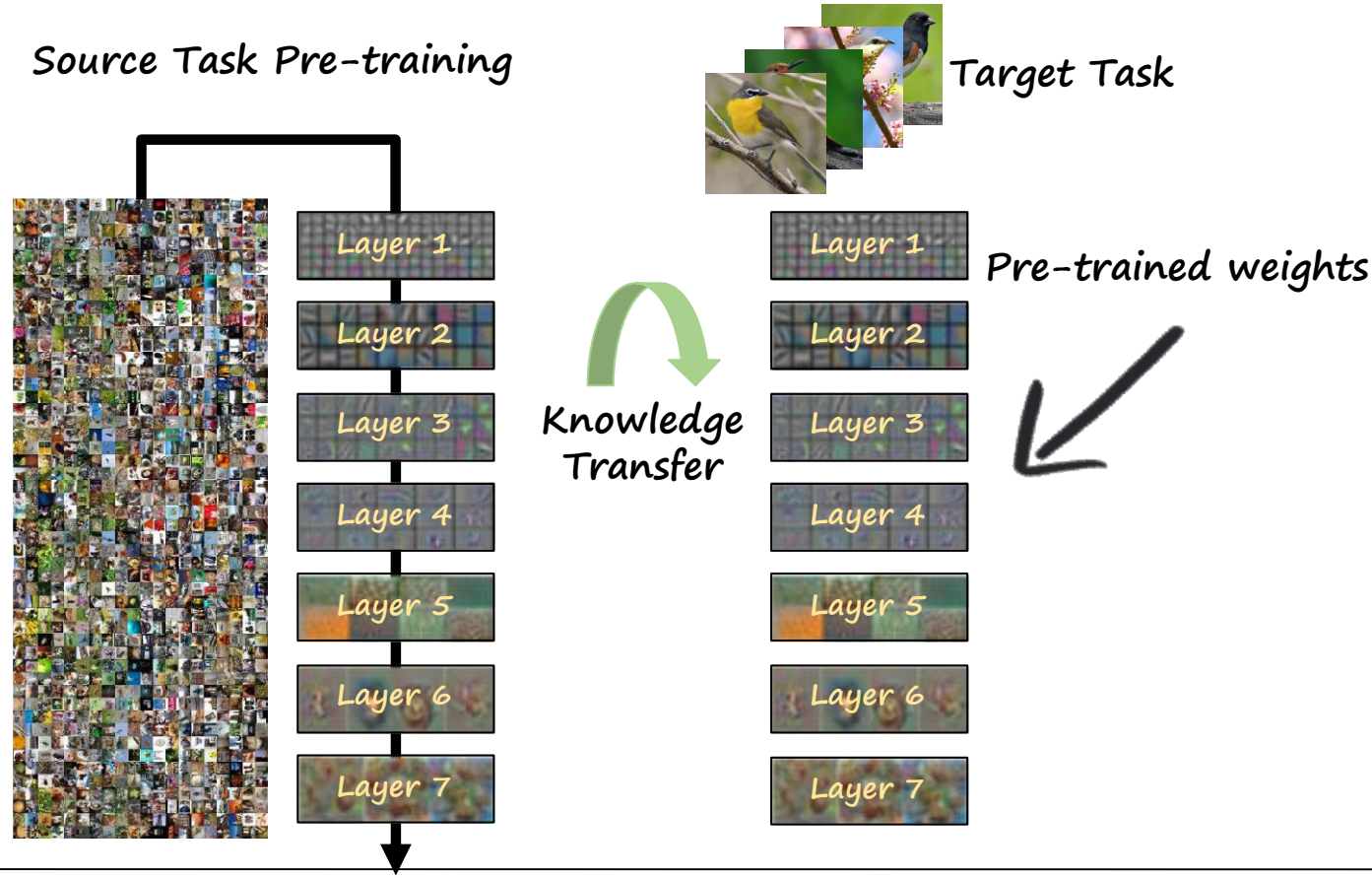


- $5 \times 5 \times 3$ filter $\mathbf{w}$ can be expressed by 75×1 vector
- $5 \times 5 \times 3$ image $\mathbf{x}$ can be expressed by 75×1 vector
- convolution output
- $y = \sigma(\mathbf{w}^T \mathbf{x} + b)$
- selective local backpropagation

Transfer Learning for Small Datasets: Fine-Tuning, Knowledge Distillation

Learning Rates in Fine-Tuning

- Transfer Learning is a promising alternative in small datasets (Image Retriaval).
- **Fine-tuning** is a type of Transfer Learning. (⇔ from scratch)

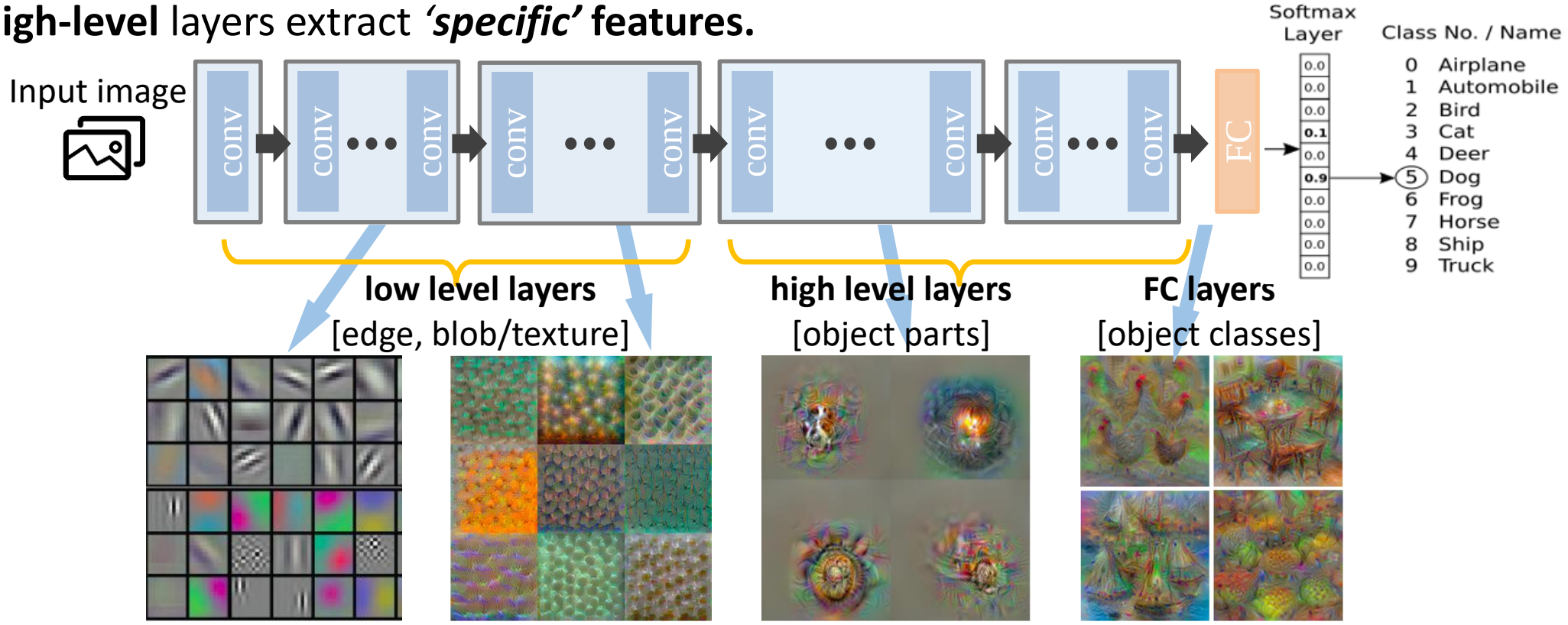


The Characteristics of Each Layer

- In previous studies,

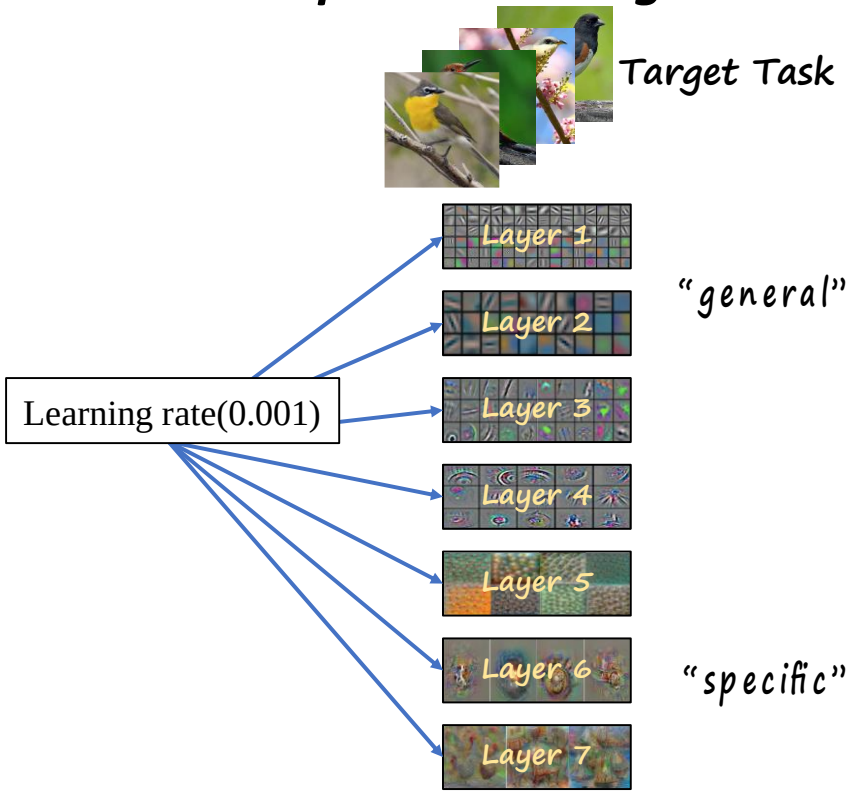
1) Visualizing and understanding convolutional networks ECCV2014
2) How transferable are features in deep neural networks? NeuIPS2014
3) Understanding deep image representations by inverting them CVPR2015

- **Low-level** layers extract ***'general'*** **features.**
  High-level layers extract ***'specific'*** **features.**

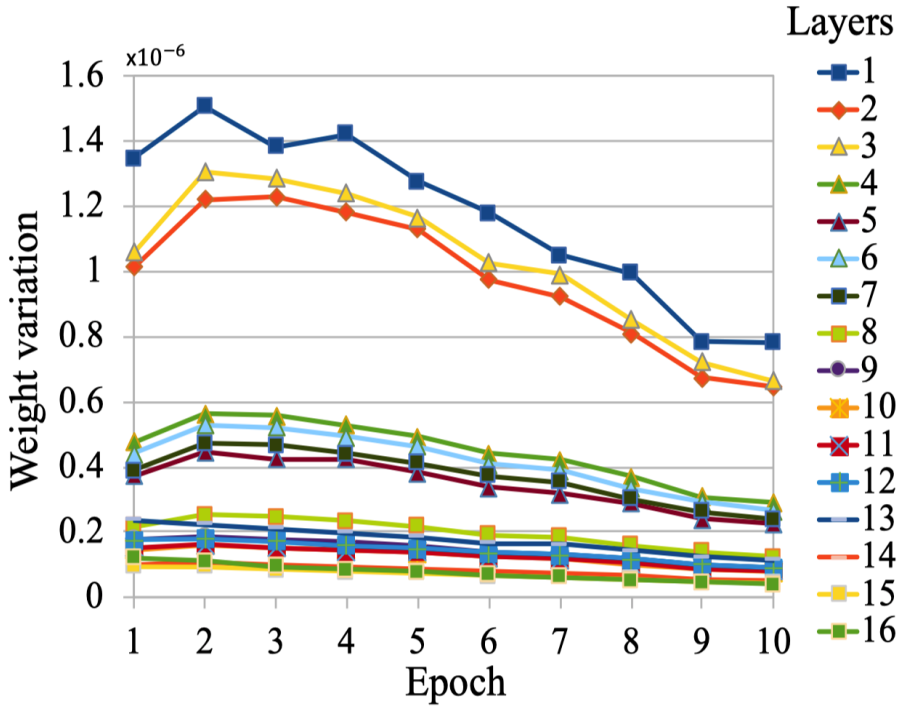


Layer-wise Learning Rate Tuning

- Most existing fine-tuning methods adopt a single LR regardless of layers.
- *LR is a hyper-parameter that **controls how much** we are **adjusting** the weights of our network with **respect the loss gradient.***



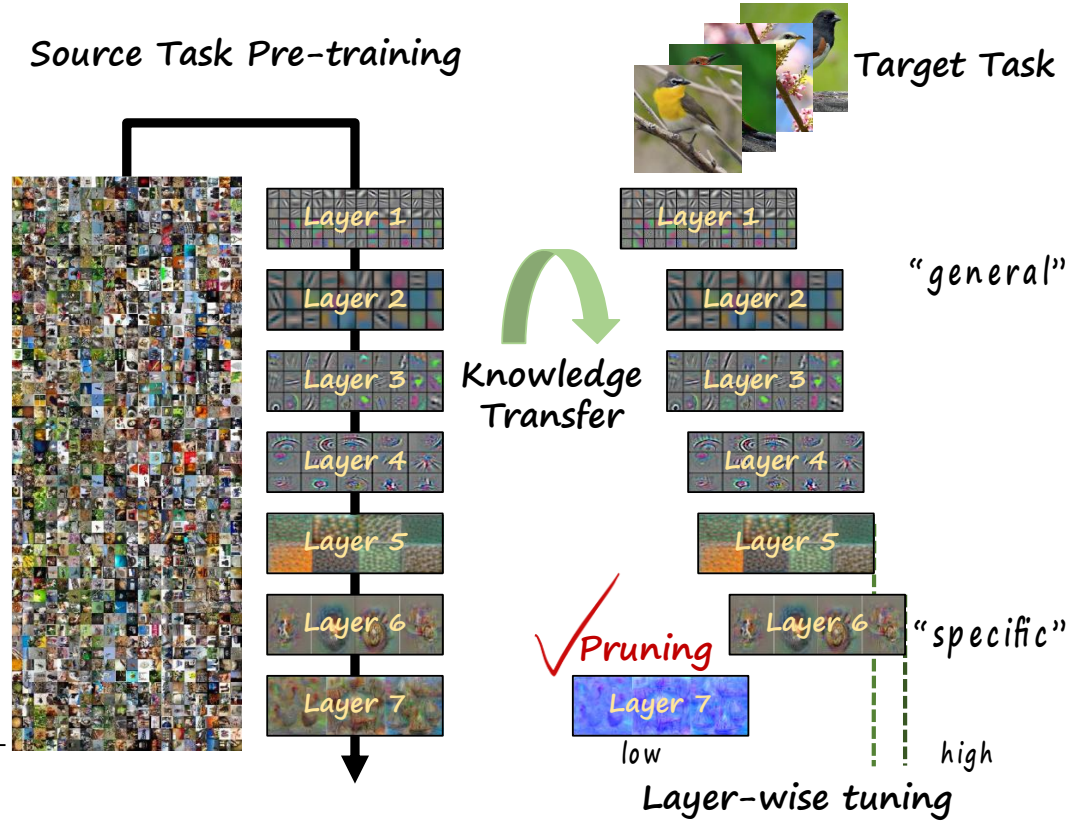
The 'Layer' is defined as 16 layers.



Hypotheses

The proposed method begins on the following two hypotheses:

- Hypothesis 1: All high-level layers are **not needed for the target task**.
- Hypothesis 2: **Low-level** layers need to change **Small**.
  High-level layers need to change **Large**.



Goal of Preliminary experiments

Experiment 1

- The high-level layer with small weight variation **may not contribute to the new task learning**.

✓To verify this,
Removed one by one from the highest layer.

Experiment 2

- Large variations in low-level layers may **destroys general features**
- Small variations in high-level layers may **not be adapt to target task**

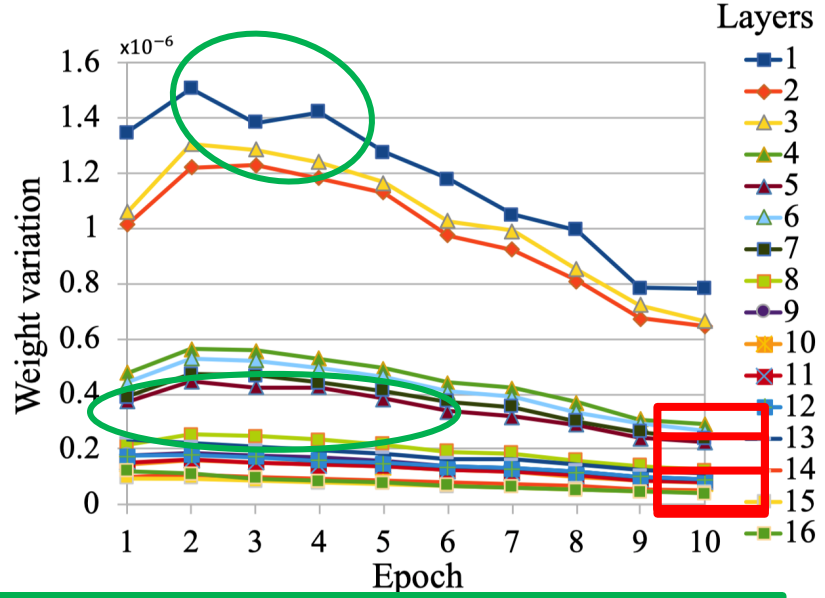
✓To verify this,
Adjusting layer-wise LRs

Results of Preliminary experiments

- Layer-wise pruning and Layer-wise tuning learning rates.

Recall @ K score

	R@1	R@2	R@4	R@8
Original	63.49	75.03	84.00	90.58
Prune until 16	66.95	77.63	86.39	92.00
Prune until 15	67.00	78.49	86.85	92.07
Prune until 14	51.00	64.16	75.44	85.94
Prune until 15†	67.15	78.61	86.83	92.08
Prune until 15‡	67.29	79.15	87.36	92.66
Prune until 15⋆	**68.33**	**79.88**	**87.69**	**92.71**



← LRs in Layers 1,2,3 were reduced to 1/10.

← LRs in Layer 13,14 were increased by 10 times.

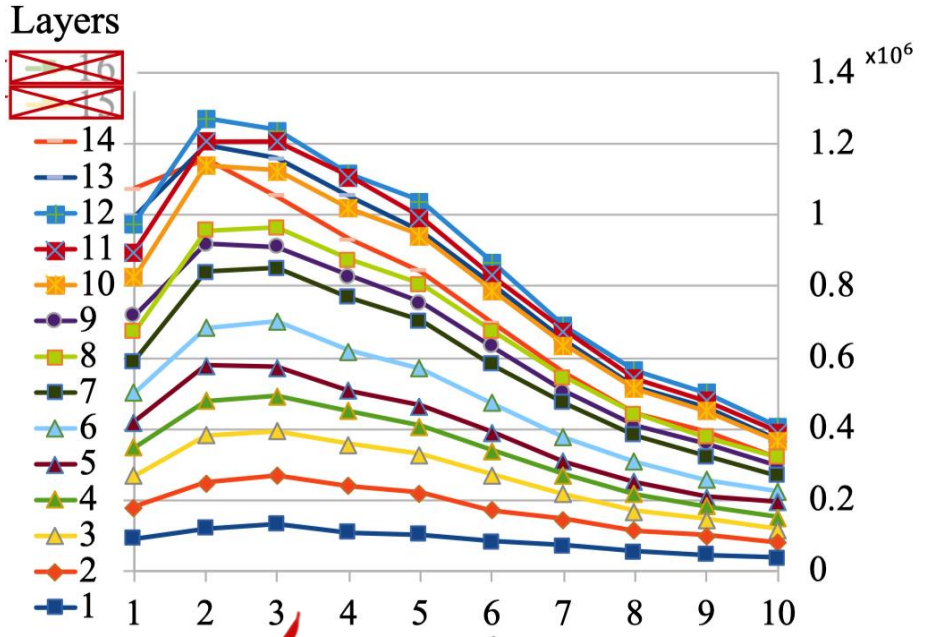
← LRs in all layers were empirically adjusted to meet Hypothesis 2 by trial and error way

Results of Preliminary experiments

- Layer-wise pruning and Layer-wise tuning learning rates.

Recall @ *K* score

	R@1	R@2	R@4	R@8
Original	63.49	75.03	84.00	90.58
Prune until 16	66.95	77.63	86.39	92.00
Prune until 15	67.00	78.49	86.85	92.07
Prune until 14	51.00	64.16	75.44	85.94
Prune until 15†	67.15	78.61	86.83	92.08
Prune until 15‡	67.29	79.15	87.36	92.66
Prune until 15⋆	**68.33**	**79.88**	**87.69**	**92.71**



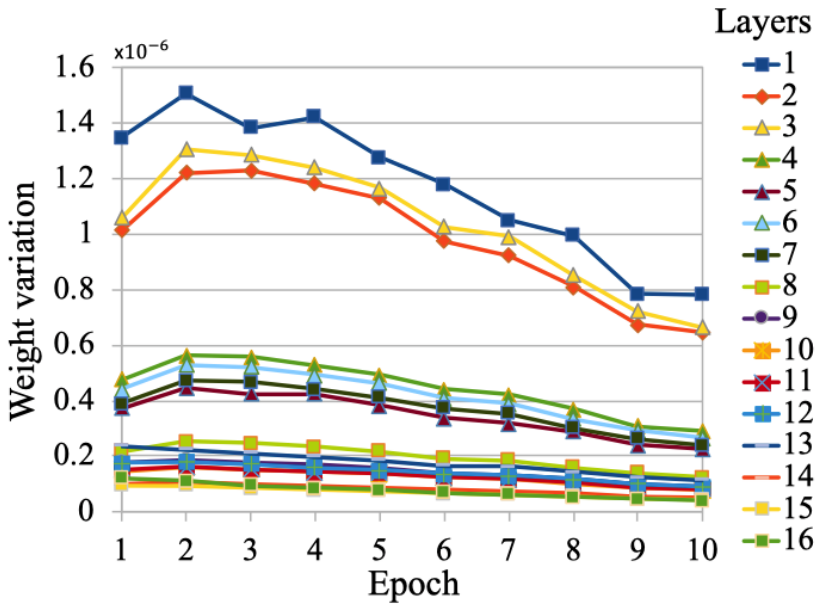
← LRs in Layers 1,2,3 were reduced to 1/10.

← LRs in Layer 13,14 were increased by 10 times.

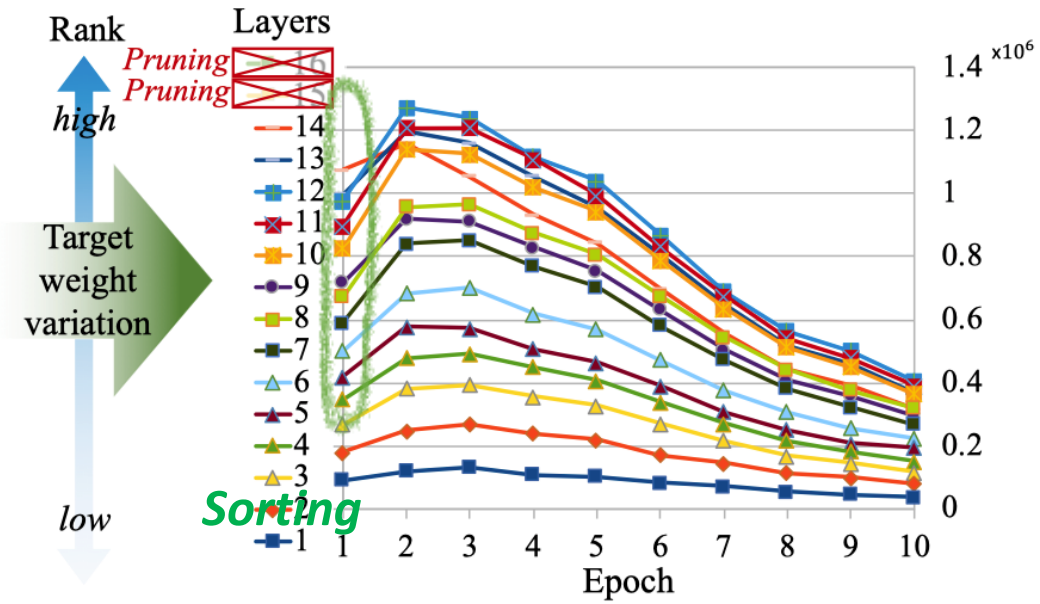
← LRs in all layers were empirically adjusted to meet Hypothesis 2 by trial and error way

Results of experiments

- There are a couple of high-level layers that **do not contribute**.
- Adjusts the LRs depending on the role of each layer contribute to performance improvement.



Single LR over all layers



Layer-wise pruning and adjusting LRs

Original	63.49	75.03	84.00	90.58	➔	Prune until 15*	**68.33**	**79.88**	**87.69**	**92.71**

Qualitative evaluation

- We evaluate our method qualitatively by CAM visualization.

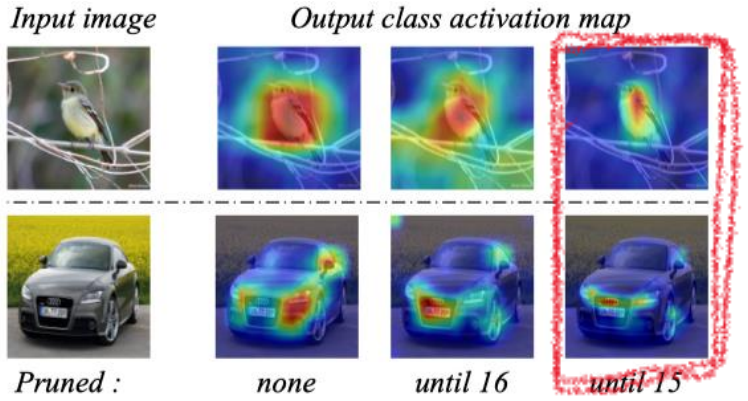


Figure 4. The class activation map (CAM) visualization of the last layers by applying different layers pruning

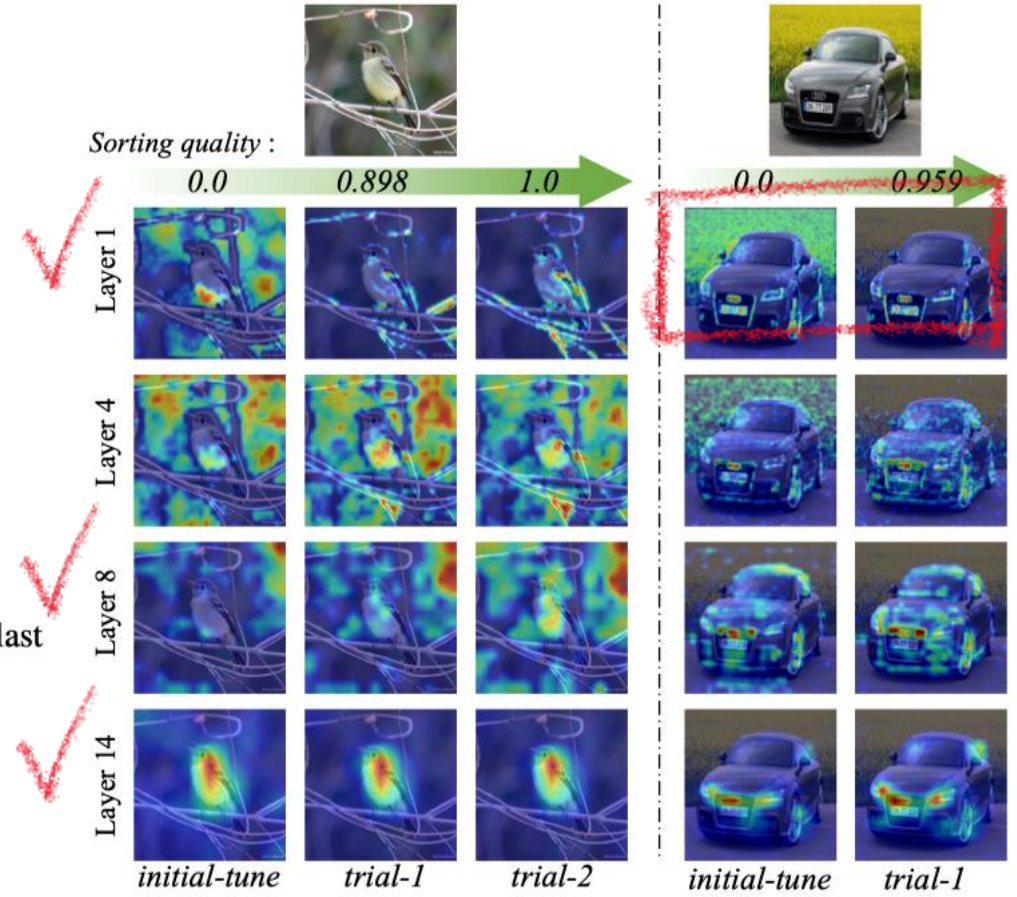


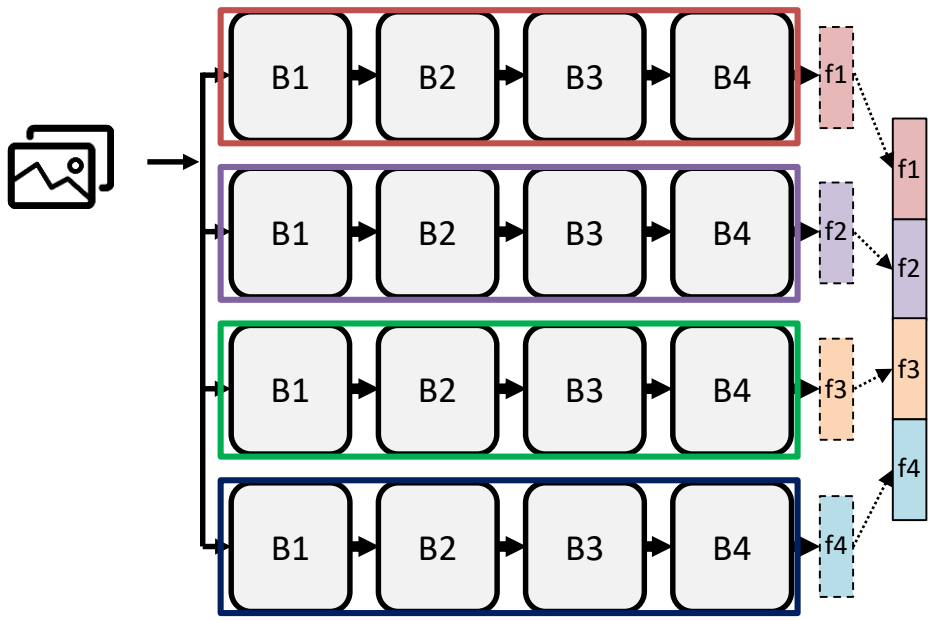
Figure 5. The class activation map (CAM) visualization of several layers (1, 4, 8, 14) according to the sorting quality. *initial-tune* is done by the conventional fine-tuning with single LR

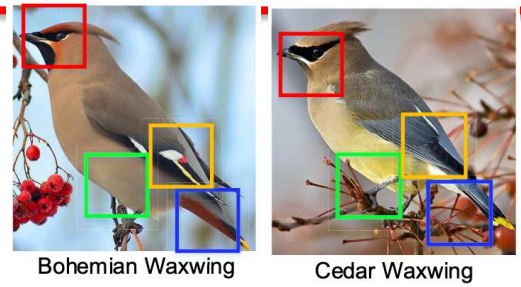
Structure Design for Fine-Tuning

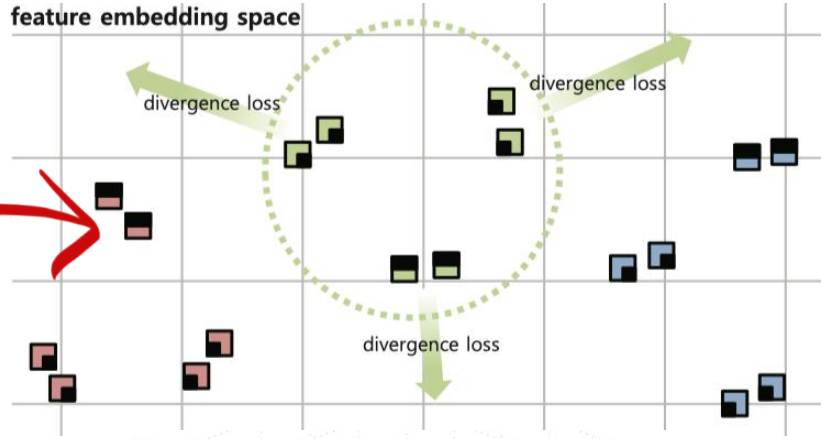
- Feature vectors of diverse characteristics of are needed

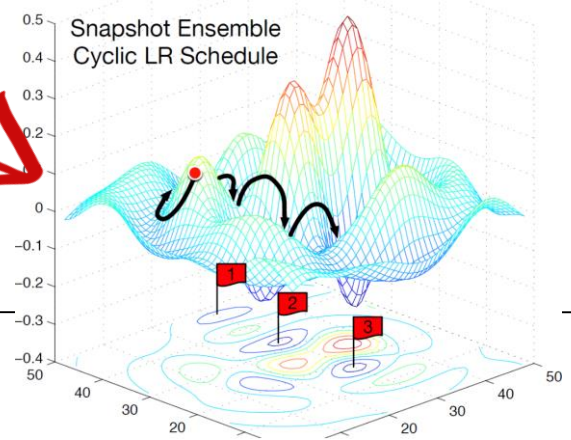
  However!!

  Disadvantage: Multiple models require considerable memory and heavy computation



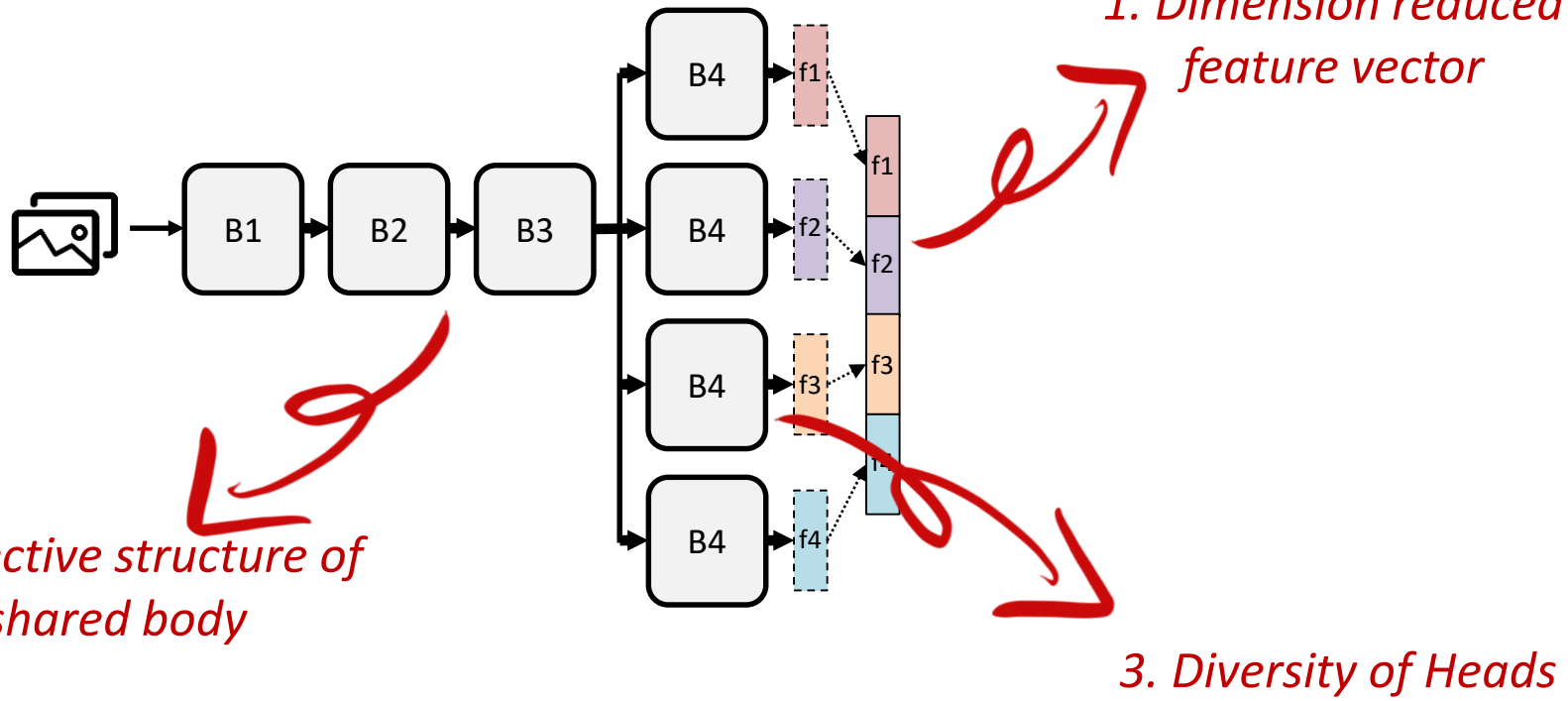






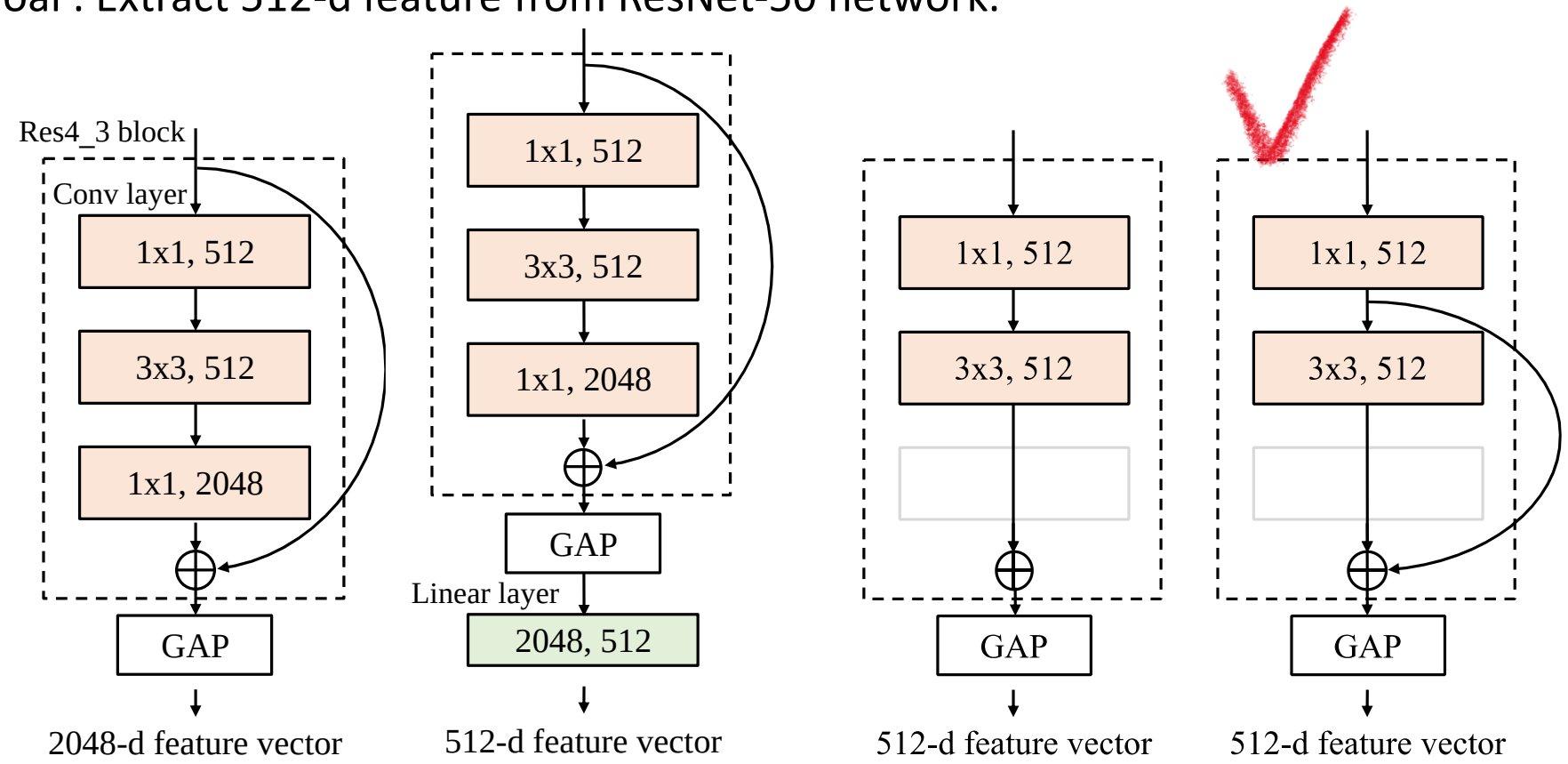
Proposed Design Concepts

- However, previous studies did not clearly state how their multi-head structure was designed and employed.
- We provide three design concept and evaluated it.



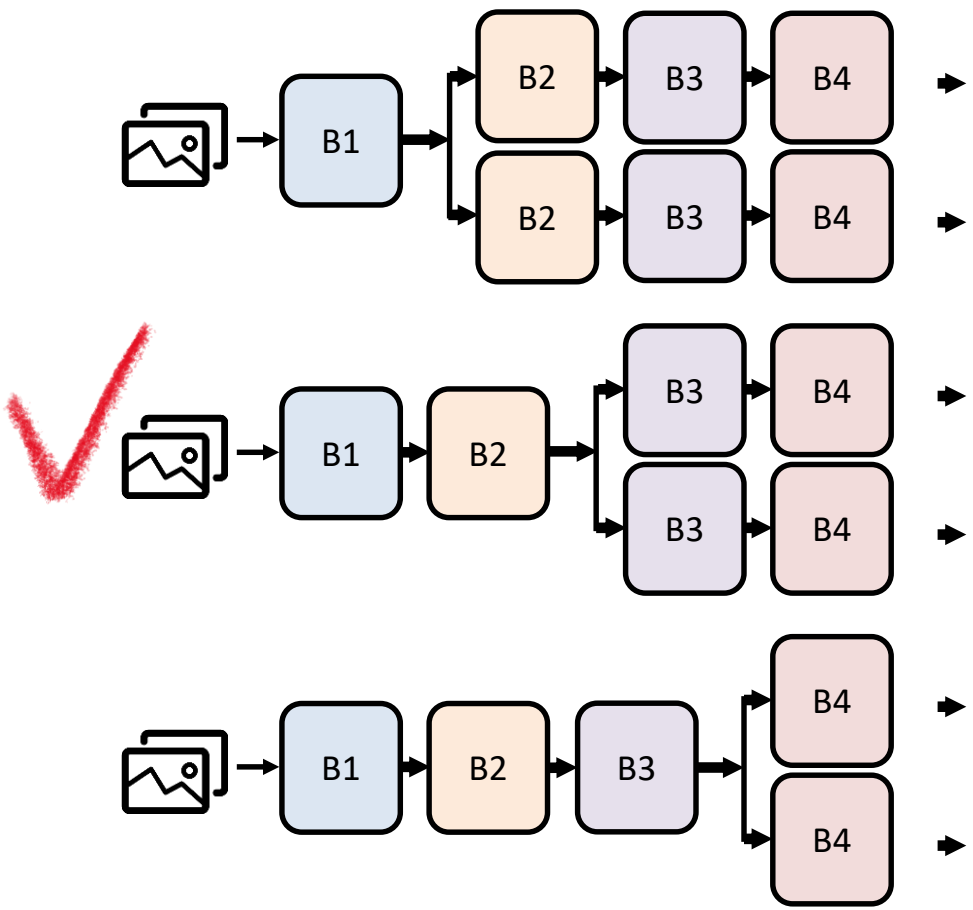
1. Dimension reduced feature vector

- Goal : Extract 512-d feature from ResNet-50 network.



2. Effective structure of shared low-level

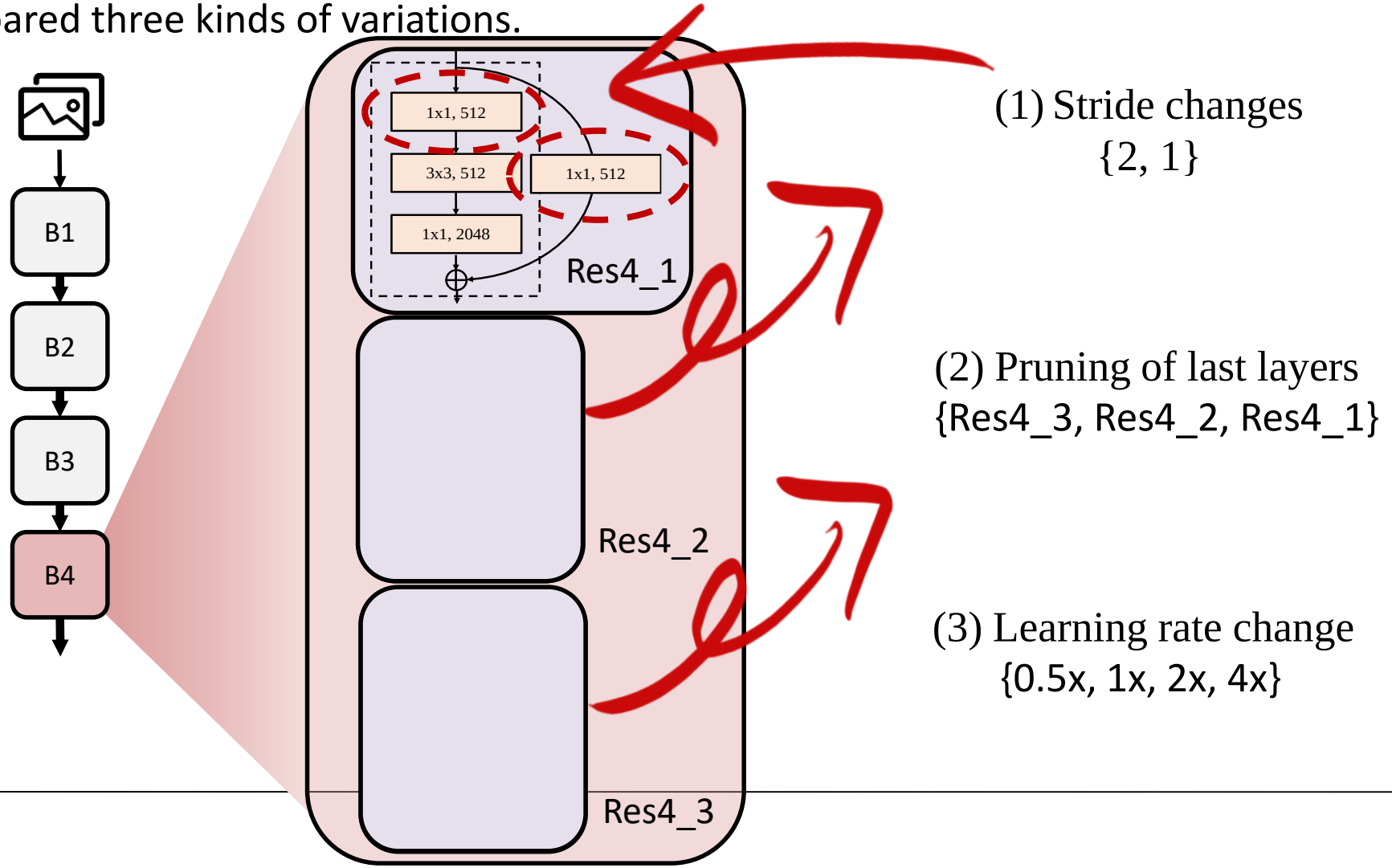
- Goal : Investigate the effective number of shared low-level layers



Memory-saving	Performance
Not Good	*Good*
Good	***Best***
Best	*Not Good*

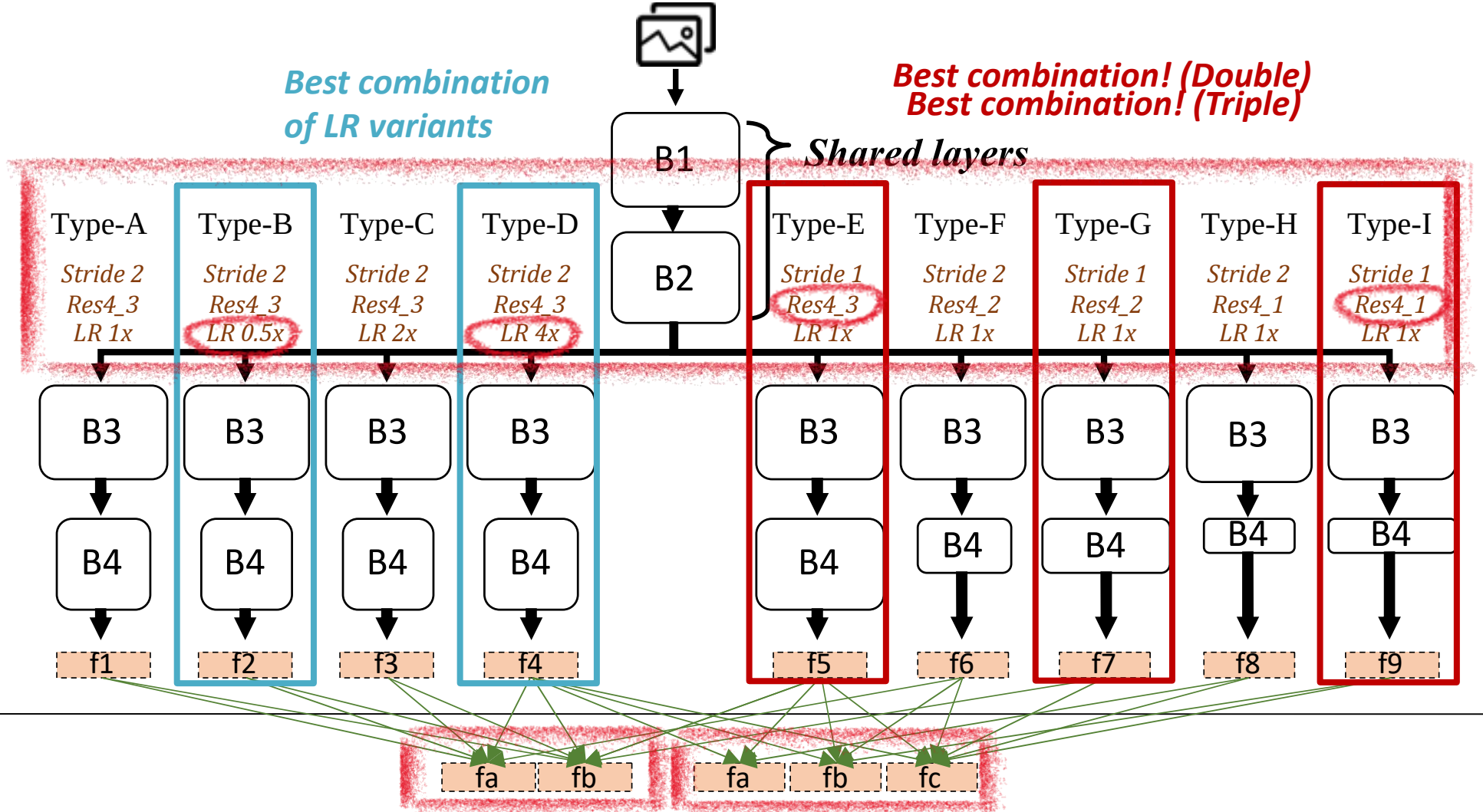
3. Diversity of high-level heads

- Goal : Investigate which structural combination is favorable for the ensemble
- We prepared three kinds of variations.



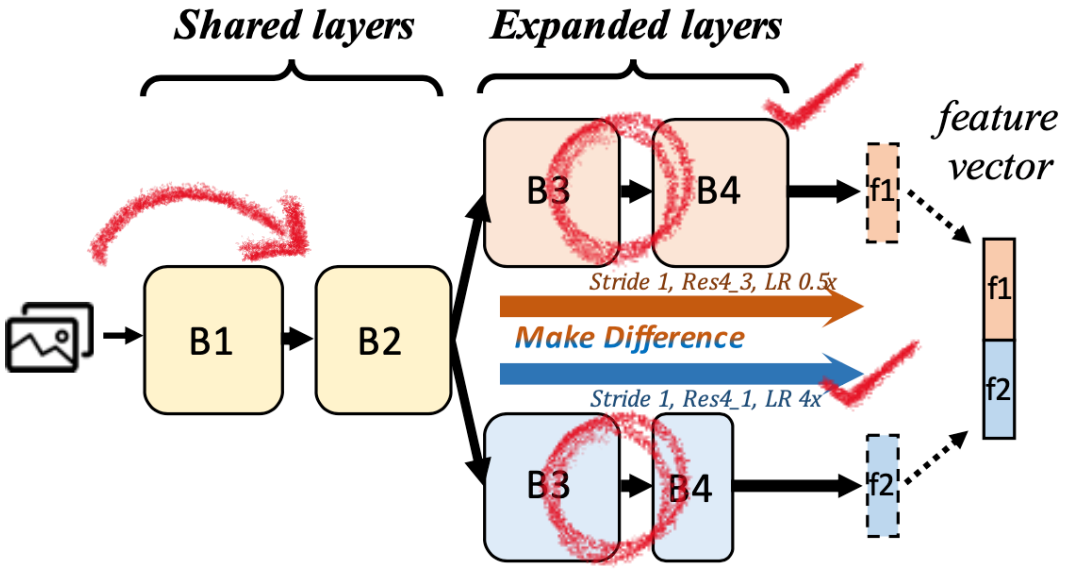
3. Diversity of high-level heads

- Goal : Investigate which structural combination is favorable for the ensemble
- Variants: (1) Stride (2) Pruning last layers (3) Learning rate



Heterogeneous Double-head Ensemble(HDhE)

- *Penultimate+ (4096-d → 1024-d)*
- Shared *Until B2*
- Head-1: stride 1, Res4_3, LR 0.5x
- Head-2: stride 1, Res4_1, LR 4x



		CUB-200			Cars-196			SOP			Inshop		
Method	Backbone	R@1	R@2	R@4	R@1	R@2	R@4	R@1	R@10	$R@10^2$	R@1	R@10	R@20
HDC [1]	Inception-v1†	53.6	65.7	77.0	73.7	83.2	89.5	69.5	84.4	82.8	62.1	84.9	89.0
A-BIER [2]	Inception-v1†	57.5	68.7	78.3	82.0	89.0	93.2	74.2	86.9	94.0	83.1	95.1	96.9
ABE-8 [3]	Inception-v1†	60.6	71.5	79.8	85.2	90.5	93.9	76.3	88.4	94.8	87.3	96.7	97.9
MS [4]	Inception-v1	65.7	77.0	86.4	84.1	90.4	94.0	78.2	90.5	96.0	89.7	97.9	98.5
DREML [5]	ResNet-18†	63.9	75.0	83.1	86.0	91.7	95.0	-	-	-	78.4	93.7	95.8
NormSoft [6]	ResNet-50	61.3	73.9	83.5	84.2	90.4	94.4	72.7	86.2	93.8	-	-	-
Margin [7]	ResNet-50	63.6	74.4	83.1	79.6	86.5	91.9	79.5	91.5	96.7	89.4	97.8	98.7
SCHM [8] *CVPR2019*	:ption-v1	66.2	76.3	84.1	83.6	-	-	77.6	89.1	94.7	91.9	98.0	98.7
SoftTriple [9] *ICCV2019*	-Inception	65.4	76.4	84.5	84.5	90.7	94.5	78.3	90.3	95.9	-	-	-
Proxy-Anchor *CVPR2020*	-Inception	68.4	79.2	86.8	86.1	91.7	95.0	79.1	90.8	96.2	91.5	98.1	98.8
Ms-HDhE (Ours)	ResNet-50†	71.3	81.4	88.4	89.8	94.3	97.1	84.0	93.5	97.1	91.2	98.0	98.6
HDhE (Ours)	ResNet-50†	72.9	82.3	89.0	90.6	95.0	97.2	84.5	93.8	97.4	92.1	98.2	98.8
Tri-HDhE (Ours)	ResNet-50†	**73.5**	**83.3**	**89.5**	**92.2**	**95.7**	**97.4**	**85.0**	**94.3**	**97.6**	**93.3**	**98.6**	**99.0**

5.1% 6.1% 5.9% 1.8%

Summary

- Constraint Convex Optimization
- dual problem, KKT conditions
- Minimization techniques
  - Gradient Descent Minimization
  - Newton Minimization
  - Gauss Newton Minimization
  - (In)Equality Constraint Minimization
- Structural Risk Minimization
  - Support Vector Machine
    - https://www.csie.ntu.edu.tw/~cjlin/papers/bottou_lin.pdf
    - Bottou and Lin, Support Vector Machine Solvers
  - Support vector data description
- Error Backpropagation Learning for a Neural Network
- Convolution Neural Network