Regularization

ESM5205 Learning from Big Data | Oct 2, 2019

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

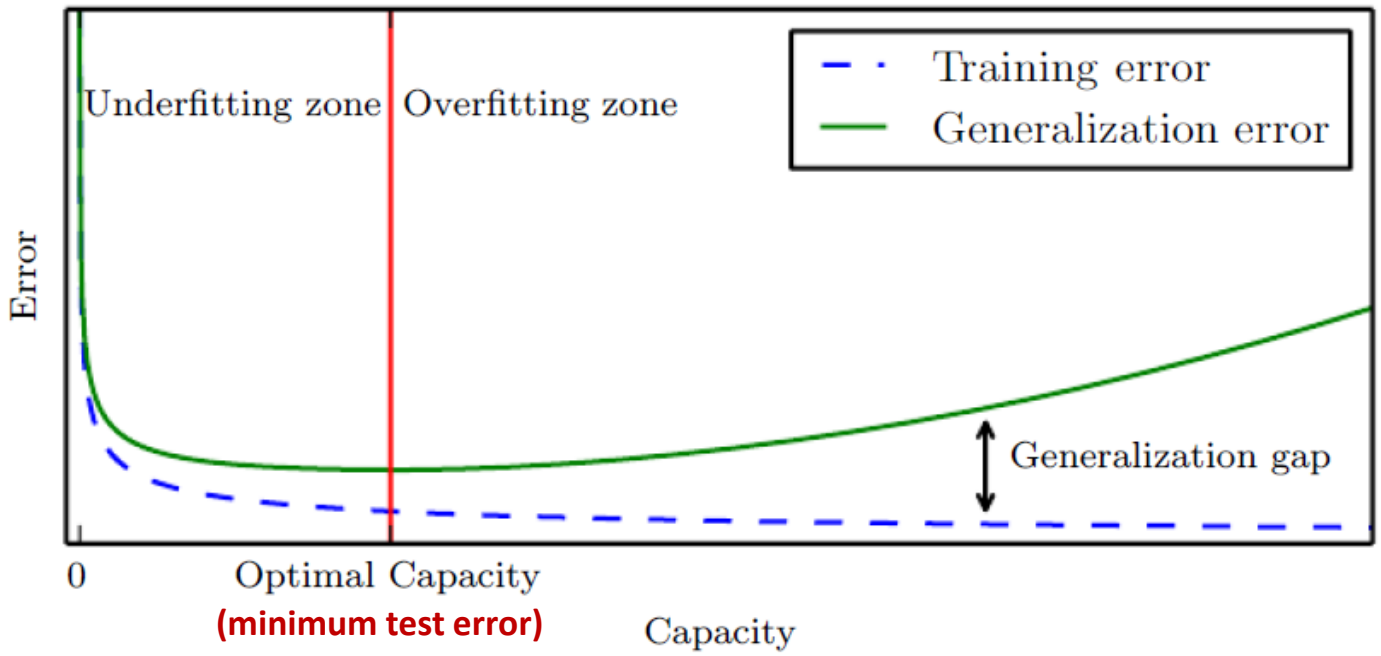
Generalization

- Suppose that a training dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ is given.
- We use a supervised learning algorithm to train a model f by minimizing the **training error** (the error on the training set D)

- The model f must perform well on new, previously unseen data points, which is called **Generalization**.
- To evaluate generalization performance, we use a **test set** $D^{(\text{test})}$ consisting of $n^{(\text{test})}$ data points that were collected separately from the training set D.
- We want the **generalization error**, *a.k.a.* **test error** (the error on the test set) to be low as well.

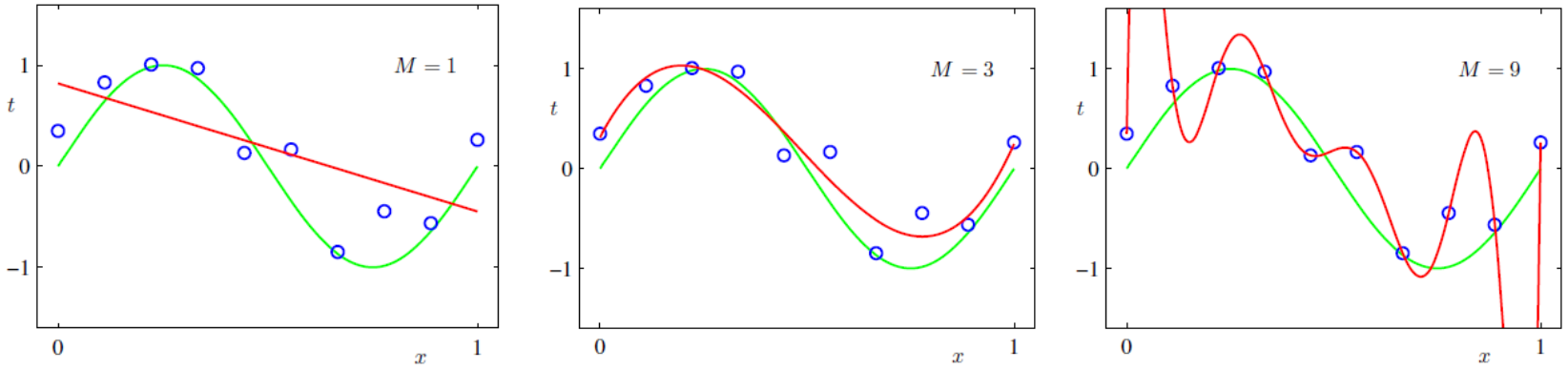
Generalization Failures

- **Overfitting:** the gap between the training error and test error is too large.
- **Underfitting:** the model is not able to obtain a sufficiently low error value on the training set.
- We can control whether a model is more likely to **overfit or underfit** by altering the **capacity**.

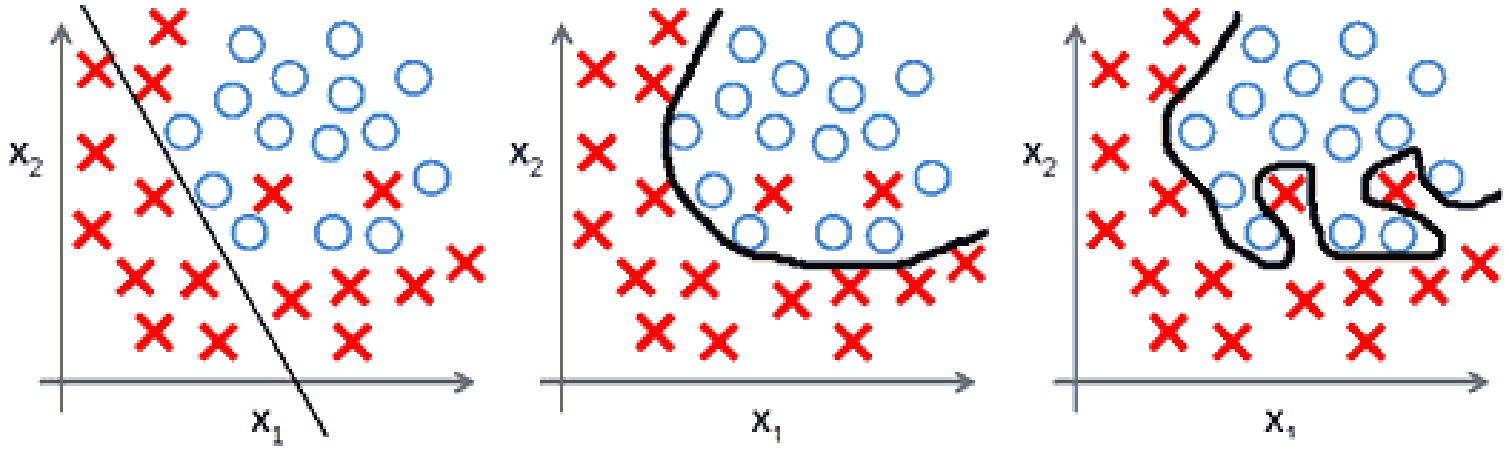


Generalization Failures

- ***Example*: in Regression…**

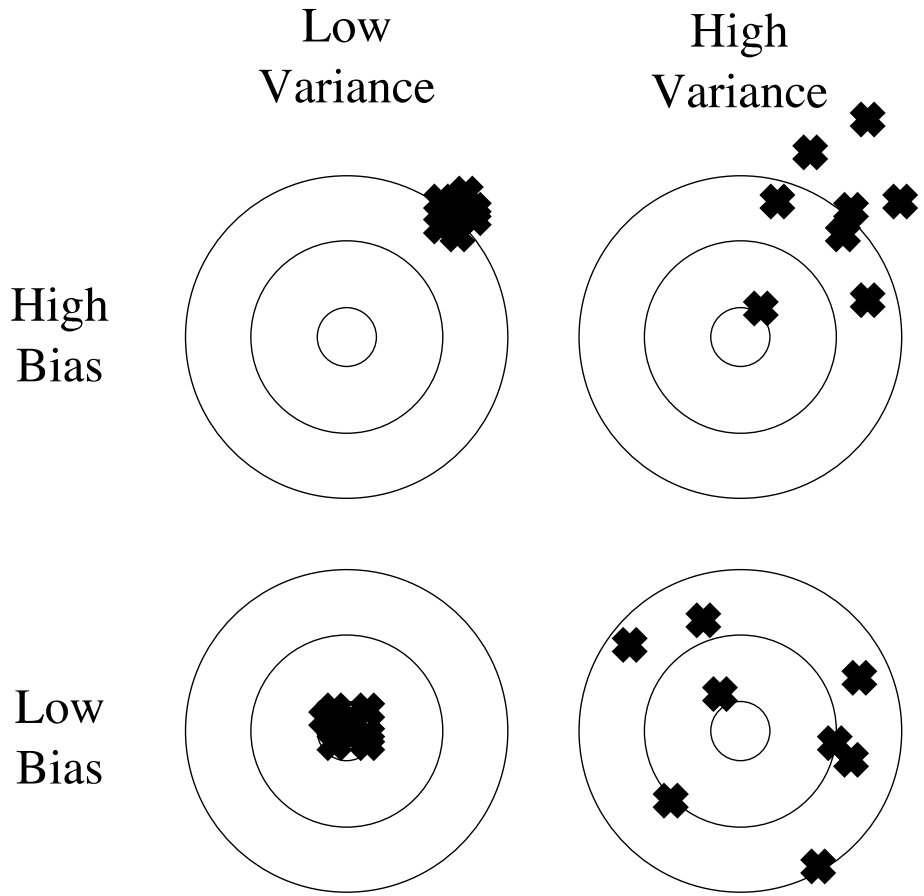


- ***Example*: in Classification…**



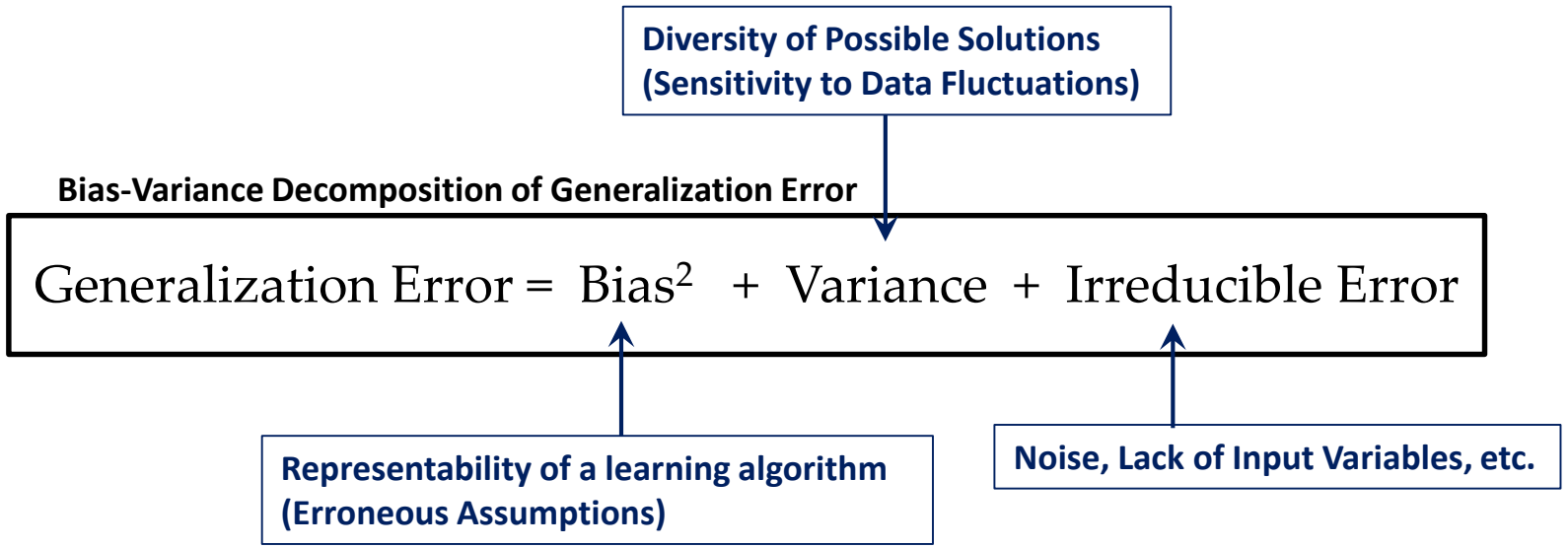
Bias and Variance

- **Generalization is related to "Bias" and "Variance"**
  - What is a Bias?
  - What is a Variance?



Bias and Variance

- **Bias-Variance Decomposition**



Bias and Variance

- **Bias-Variance Decomposition of MSE (Regression Case)**
  - Suppose that there's an unknown true function with noise $y = f^*(\boldsymbol{x}) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2)$
  - We want to find a prediction $\hat{y}$ that best approximates the true value $f^*(\boldsymbol{x})$ by means of a learning algorithm.

  - **(Expected) Generalization error** (in squared error) on an unseen data point $\boldsymbol{x}$ is :

$$\mathrm{E}[(y - \hat{y})^2] = (f^*(\boldsymbol{x}) - \mathrm{E}[\hat{y}])^2 + \mathrm{E}[(\hat{y} - \mathrm{E}[\hat{y}])^2] + \sigma^2$$

$$\mathrm{bias}[\hat{y}]^2 = (f^*(\boldsymbol{x}) - \mathrm{E}[\hat{y}])^2$$
$$\mathrm{Var}[\hat{y}] = \mathrm{E}[(\hat{y} - \mathrm{E}[\hat{y}])^2]$$

▶ proof: *try yourself!*

Bias and Variance

- **Bias-Variance Decomposition of MSE (Regression Case)**

$$
\begin{aligned}
\mathrm{E}[(y-\hat{y})^2] &= \mathrm{E}[(f^*(\boldsymbol{x})+\epsilon-\hat{y})^2] \\
&= \mathrm{E}[(f^*(\boldsymbol{x})+\epsilon-\hat{y}+E[\hat{y}]-E[\hat{y}])^2] \\
&= \mathrm{E}[((f^*(\boldsymbol{x})-E[\hat{y}])+\epsilon-(\hat{y}-E[\hat{y}]))^2] \\
&= \mathrm{E}[(f^*(\boldsymbol{x})-E[\hat{y}])^2]+\mathrm{E}[\epsilon^2]+\mathrm{E}[(\hat{y}-E[\hat{y}])^2] \\
&\qquad +2\mathrm{E}[\epsilon(f^*(\boldsymbol{x})-E[\hat{y}])]-2\mathrm{E}[\epsilon(\hat{y}-E[\hat{y}])]-2\mathrm{E}[(f^*(\boldsymbol{x})-E[\hat{y}])(\hat{y}-E[\hat{y}])] \\
&= (f^*(\boldsymbol{x})-E[\hat{y}])^2+\mathrm{E}[\epsilon^2]+\mathrm{E}[(\hat{y}-E[\hat{y}])^2] \\
&\qquad +2\mathrm{E}[\epsilon](f^*(\boldsymbol{x})-E[\hat{y}])-2\mathrm{E}[\epsilon]\mathrm{E}\big[\hat{y}-E[\hat{y}]\big]-2(f^*(\boldsymbol{x})-E[\hat{y}])\mathrm{E}\big[\hat{y}-E[\hat{y}]\big] \\
&= (f^*(\boldsymbol{x})-E[\hat{y}])^2+\mathrm{E}[\epsilon^2]+\mathrm{E}[(\hat{y}-E[\hat{y}])^2] \\
&= \mathrm{bias}[\hat{y}]^2+\sigma^2+\mathrm{Var}[\hat{y}]^2
\end{aligned}
$$

Bias and Variance

- **Bias and Variance in Supervised Learning**
  - **Bias** (typically, lower capacity → higher bias)
    - The difference between the model prediction and the true value
    - With too high bias, a model pays little attention to the training data, thus can be oversimplified. → *underfitting*
    - ***Example***
      low bias: linear regression applied to linear data, deep neural network
      high bias: constant function, linear regression applied to non-linear data
  - **Variance** (typically, higher capacity → higher variance)
    - The variability of the model prediction for a given data point.
    - With too high variance, a model pays much attention to the training data, thus does not generalize on unseen data. → *overfitting*
    - ***Example***
      low variance: constant function, linear regression
      high variance: polynomial regression with high degree, deep neural network

Bias and Variance

- **Bias and Variance in Supervised Learning**
  - **The bias and variance by a learning algorithm**
    - The number of parameters/hyperparameters
    - The choice of hyperparameters
    - Randomness in the learning algorithm used (*e.g.*, neural network parameter initialization)
  - **Other sources of the bias and variance**
    - Characteristics of the data (*e.g.*, size, dimensionality, etc.)
    - Noise in the data (input and output variables)

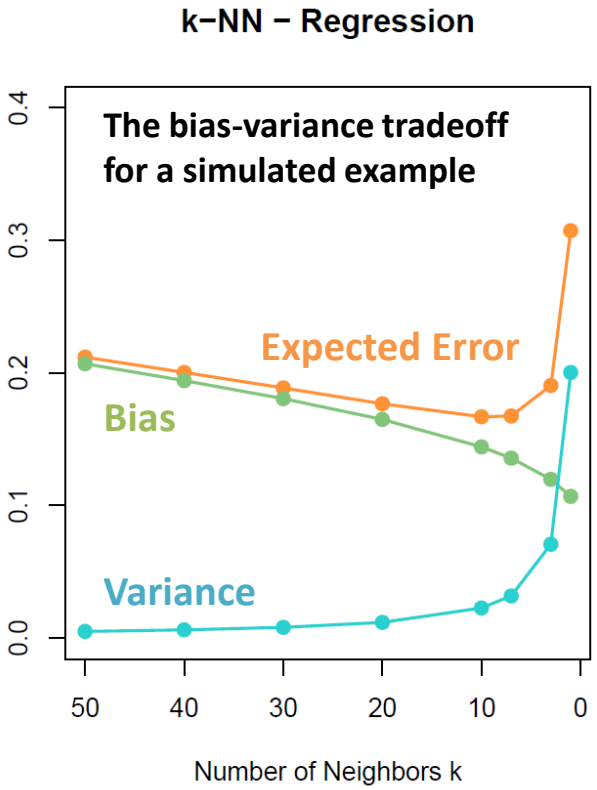
Bias and Variance

- ***Example***: *k*-nearest neighbors regression

$$\hat{y} = \frac{1}{k}\sum_{(x_{(i)},y_{(i)})\in kNN(x)} y_{(i)}$$

- $\text{bias}[\hat{y}]^2 = \left[f^*(\boldsymbol{x}) - \frac{1}{k}\sum_{(x_{(i)},y_{(i)})\in kNN(x)} f^*(\boldsymbol{x}_{(i)})\right]^2$
- $\text{Var}[\hat{y}] = \text{Var}\left[\frac{1}{k}\sum_{(x_{(i)},y_{(i)})\in kNN(x)} y_{(i)}\right] = \frac{\sigma^2}{k}$
- What if we increase the hyperparameter *k*?

bias↑, variance↓



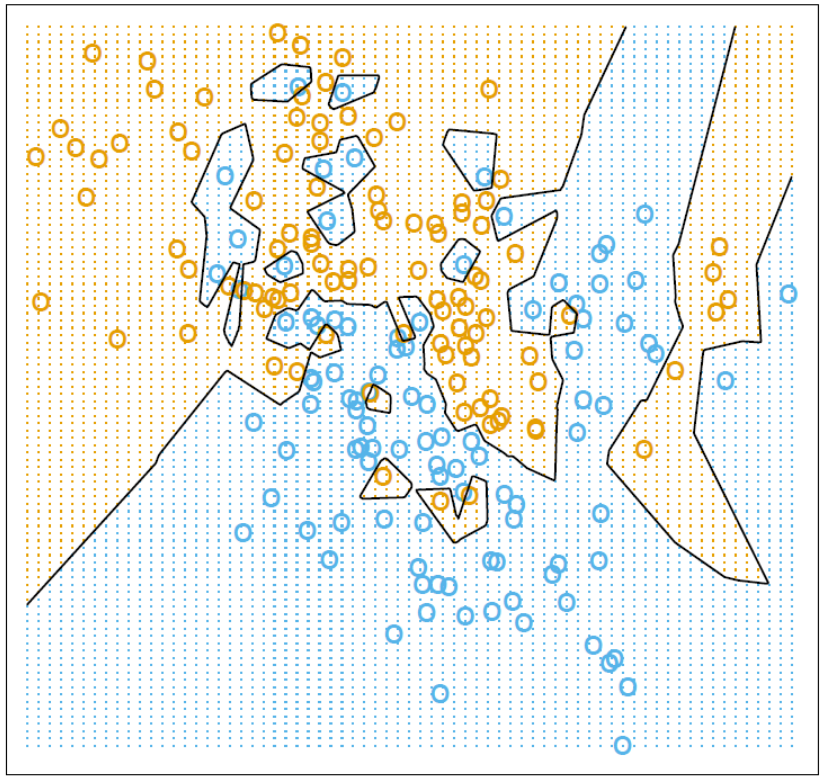
$$\text{bias}[\hat{y}]^2 = (f^*(\boldsymbol{x}) - E[\hat{y}])^2$$
$$\text{Var}[\hat{y}] = E[(\hat{y} - E[\hat{y}])^2]$$

Bias and Variance

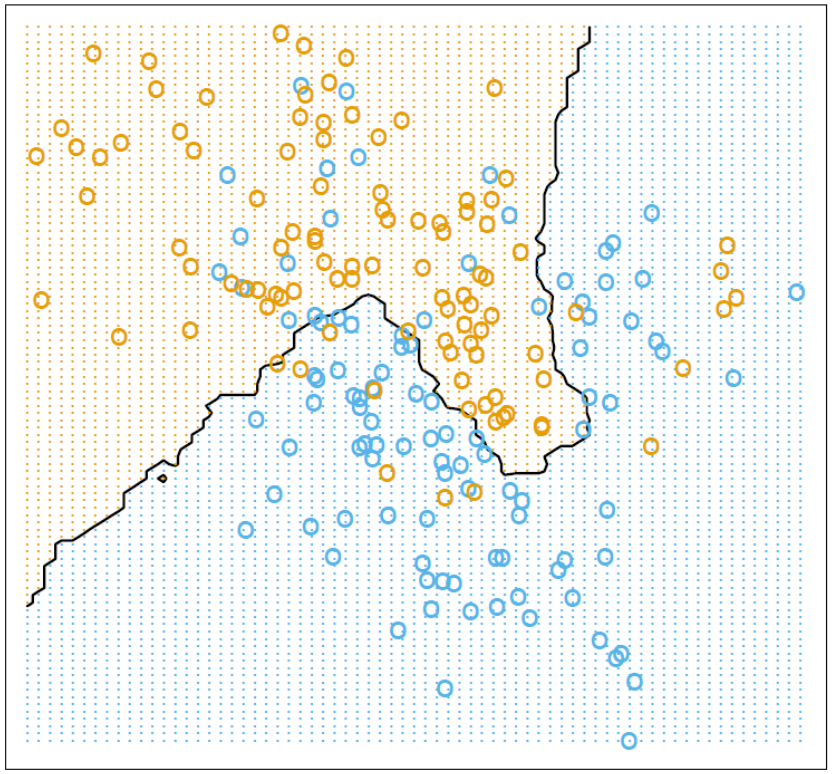
- ***Example***: *k*-nearest neighbors classification

$$\hat{y} = \underset{j}{\text{argmax}} \sum_{(\boldsymbol{x}_{(i)}, y_{(i)}) \in kNN(\boldsymbol{x})} I(y_{(i)} = j)$$

1−Nearest Neighbor Classifier

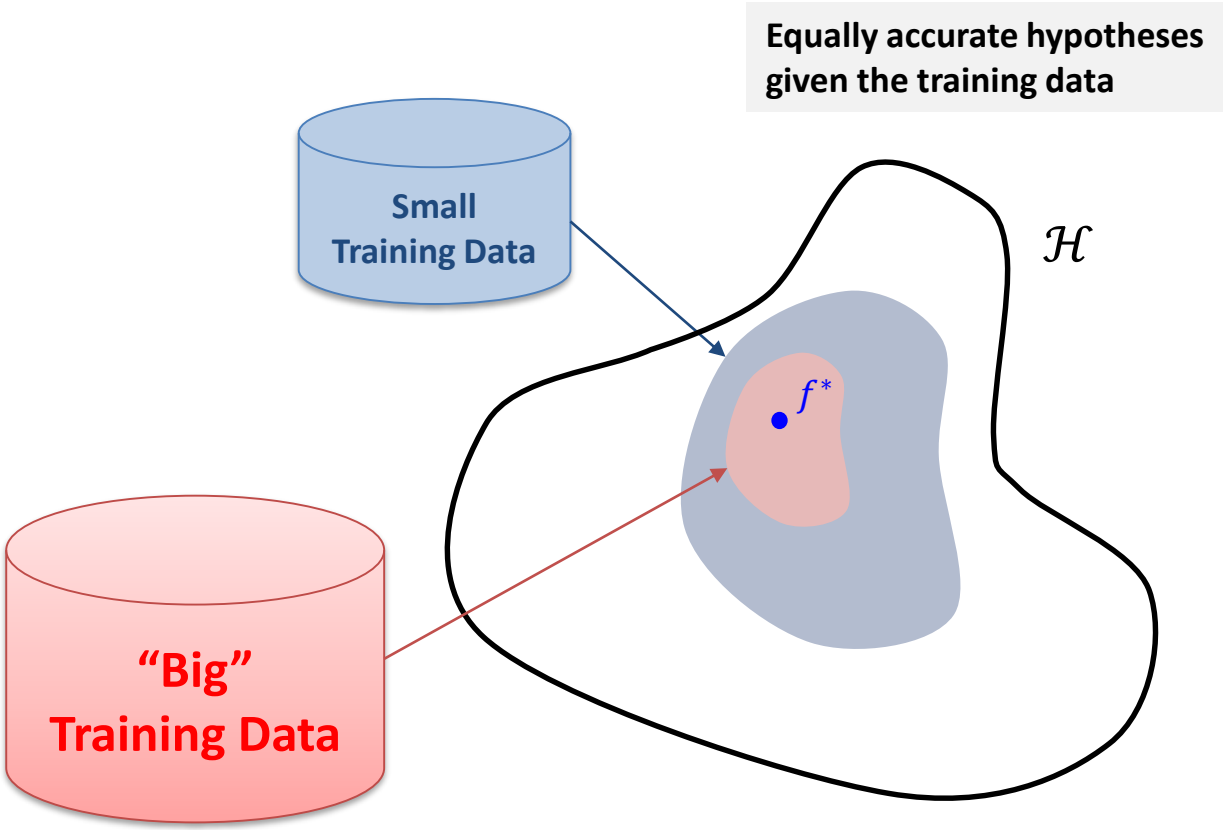


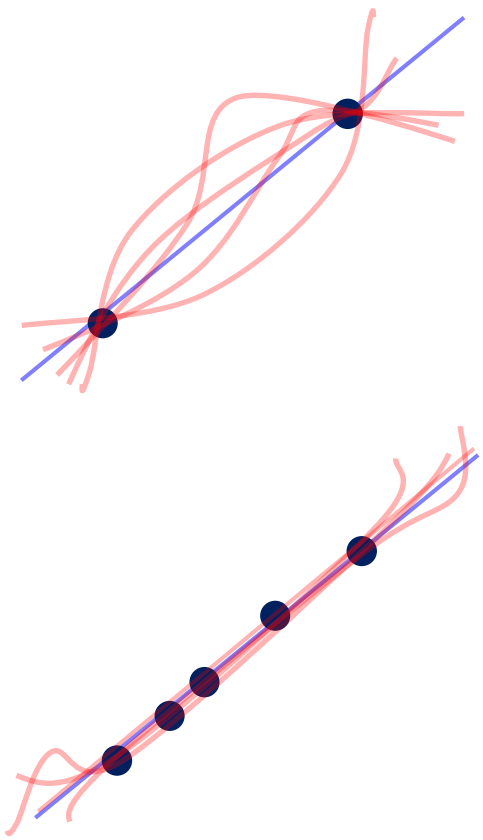
15-Nearest Neighbor Classifier



Bias and Variance

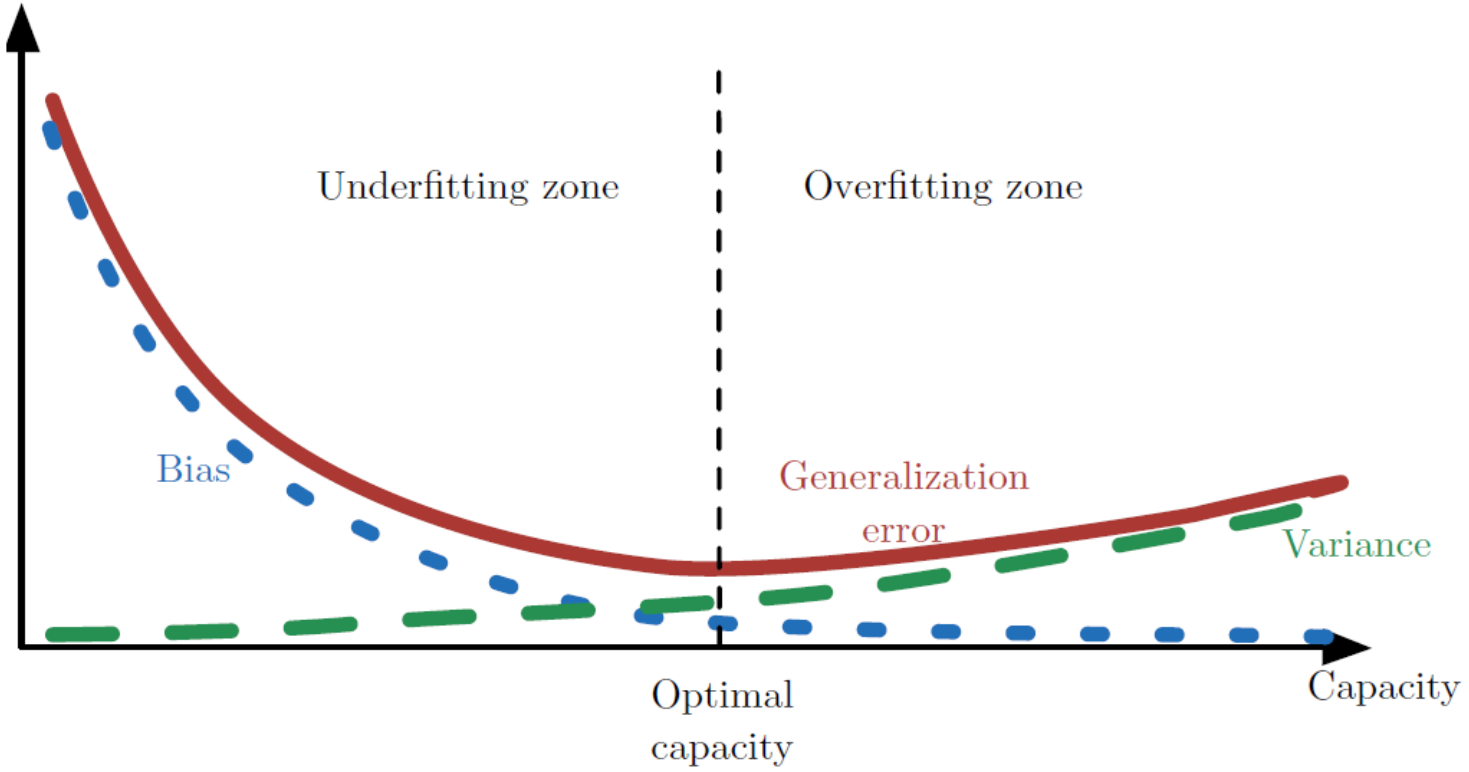
- ***Example*:** the effect of the size of training data
  - **bigger data → lower variance**





Bias and Variance

- **Bias-Variance Trade-off in Supervised Learning**
  - In general, low bias ↔ high variance and high bias ↔ low variance



Regularization

- **Regularization: Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error**
- **It works by reducing variance significantly while not overly increasing the bias.**

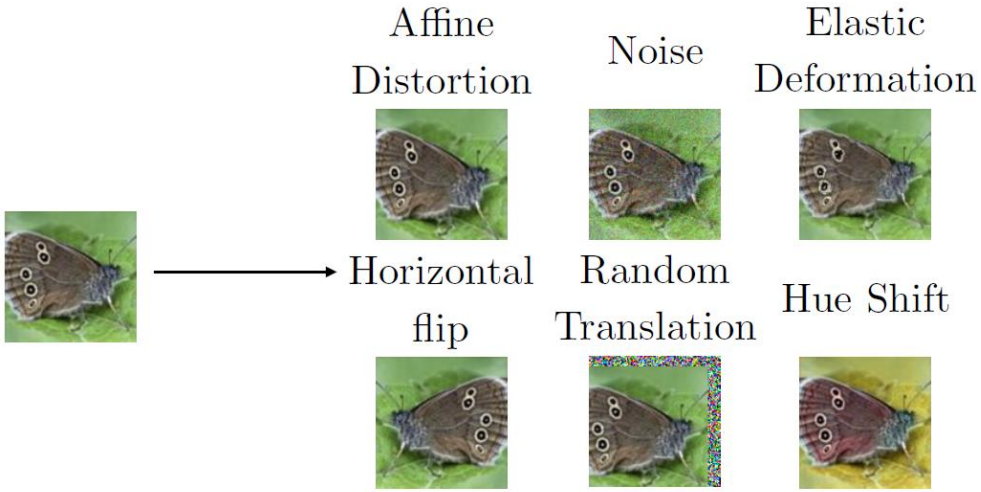
  → *to avoid overfitting*

- **Regularization Strategies for Neural Networks**
  - Data Augmentation
  - Parameter Norm Penalties
  - Early Stopping
  - Dropout
  - Batch Normalization
  - Multitask Learning
  - Transfer Learning
  - Ensemble (Bagging, etc.)
  - …

Data Augmentation

- **The best** (but often not easy) **way to make a machine learning model generalize better is to train it on more data**.
- But, in practice, the amount of data we have is limited. (data collection cost…)

- **Data Augmentation**
  - **Create fake data** and add them to the training set.
  - Practically very effective! But application is limited.
  - ***Example: Various possible (label-preserving) transformations for image data***
    - *flip, rotate, crop, translation, distortion, rescale, contrast, noise, etc…*



Parameter Norm Penalties

- **If the parameters θ are unconstrained, they can explode** → high variance

- **Regularizing parameter norms**
  : Adding a norm penalty $\Omega(\boldsymbol{\theta})$ to the cost function $J(\boldsymbol{\theta})$

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta})$$

λ controls the trade-off

  - L2 Regularization (weight decay)
    - $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_2^2$

  - L1 Regularization
    - $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$

Parameter Norm Penalties

- **L2 Regularization:** Adding a L2-norm penalty to $J(\boldsymbol{\theta})$

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

  - **In the gradient descent,**

$$\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \epsilon\nabla_{\boldsymbol{\theta}}\tilde{J}(\boldsymbol{\theta}) = \boldsymbol{\theta} - \epsilon(\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) + \lambda\boldsymbol{\theta})$$
$$= (1 - \epsilon\lambda)\boldsymbol{\theta} - \epsilon\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$$

shrinking the parameter vector

Parameter Norm Penalties

- **L2 Regularization:** Adding a L2-norm penalty to $J(\boldsymbol{\theta})$

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

  - **Linear regression with L2 Regularization** (Ridge Regression)

$$\tilde{J}(\mathbf{w}) = \frac{1}{n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

▶ optimization → a closed-form solution (normal equation)

$$\nabla_{\mathbf{w}}\tilde{J}(\mathbf{w}) = \frac{1}{n}\nabla_{\mathbf{w}}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda\mathbf{w} = 0$$

…

…

…

$$\mathbf{w}^* = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

Parameter Norm Penalties

- **L1 Regularization:** Adding a L1-norm penalty to $J(\boldsymbol{\theta})$

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1$$

  - **In the gradient descent,**

$$\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} \tilde{J}(\boldsymbol{\theta}) = \boldsymbol{\theta} - \epsilon \left(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \lambda \cdot \operatorname{sign}(\boldsymbol{\theta}) \right)$$

$$= \left(\boldsymbol{\theta} - \epsilon\lambda \cdot \operatorname{sign}(\boldsymbol{\theta}) \right) - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

encourage θ_j to be zero
(sparsity property: some parameters have an optimal value of zero)

Parameter Norm Penalties

- **L1 Regularization:** Adding a L1-norm penalty to $J(\boldsymbol{\theta})$

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1$$

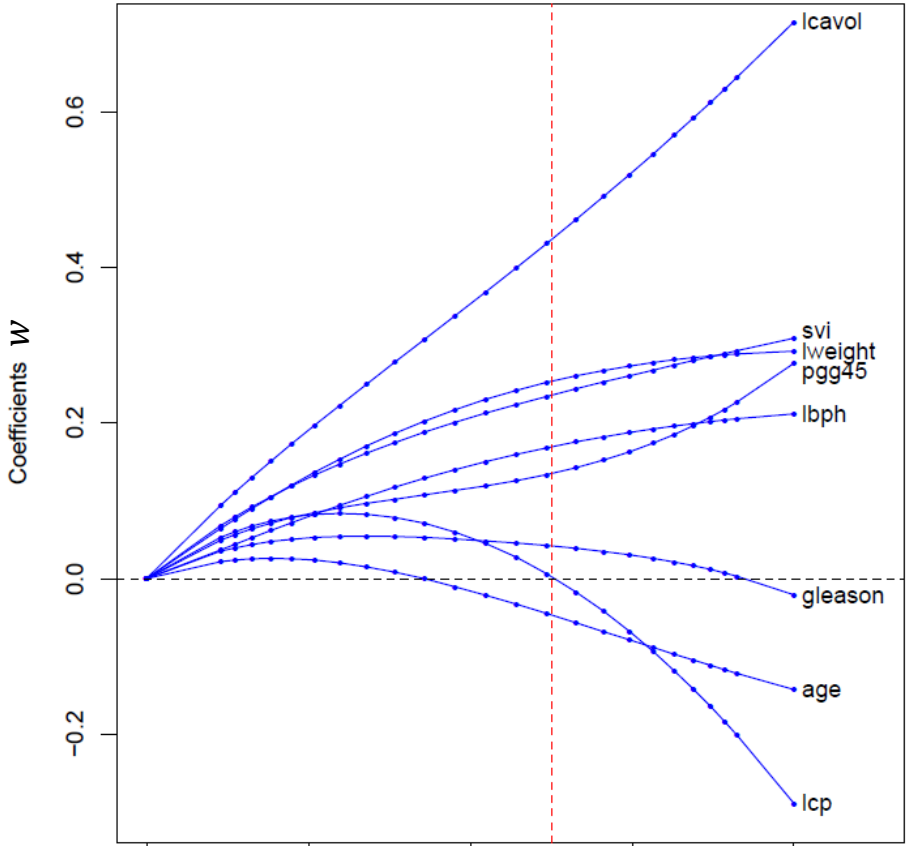
  - **Linear regression with L1 Regularization** (Lasso Regression)

$$\tilde{J}(\mathbf{w}) = \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|_1$$

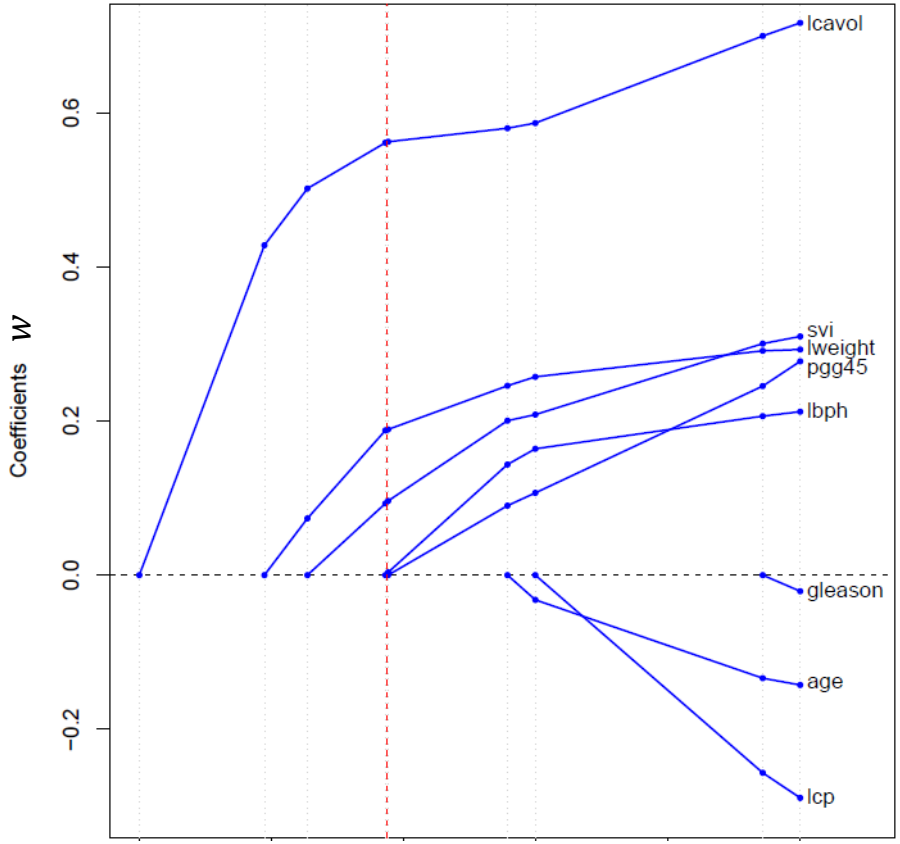
▶ optimization → use gradient descent

Parameter Norm Penalties

- ***Example***: Ridge Regression vs Lasso Regression
  - $y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$



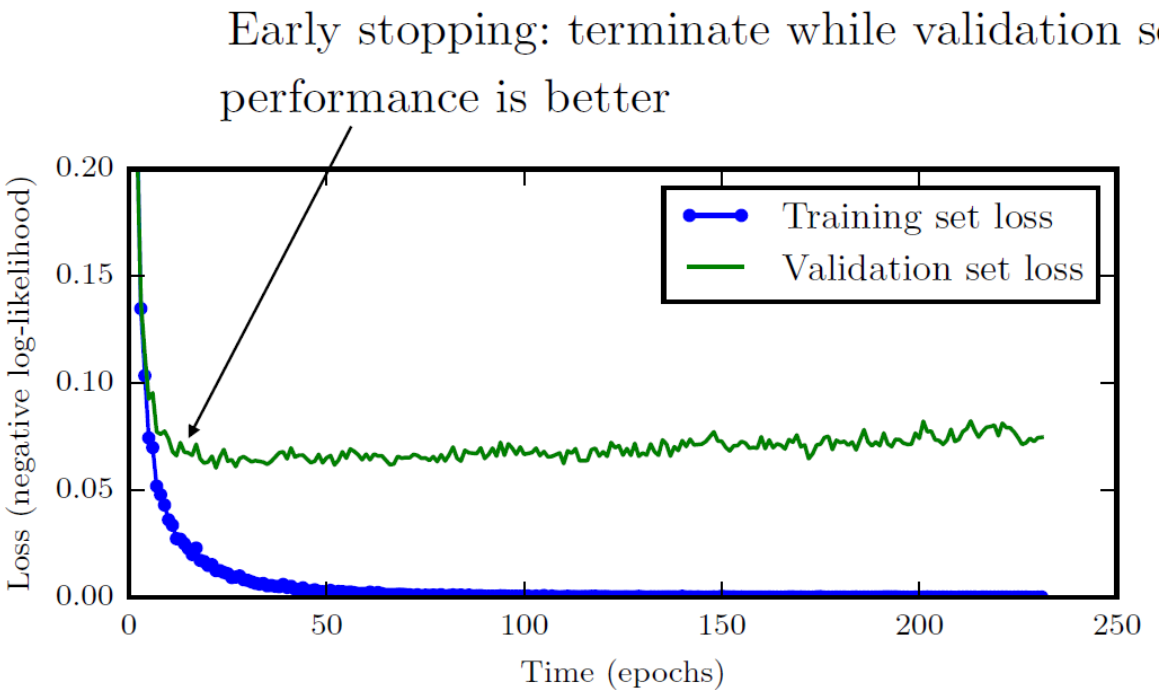
constant function — larger λ — smaller λ — standard linear regression



constant function — larger λ — smaller λ — standard linear regression

Early Stopping

- **The most commonly used form of regularization in neural network training**
- **"Don't train the network too much to prevent overfitting!"**
- **Use the validation error to decide when to stop training**
  : Choose the parameters that minimize the validation error (hopefully, lower the test error)
- **The number of training epochs is a hyperparameter.**

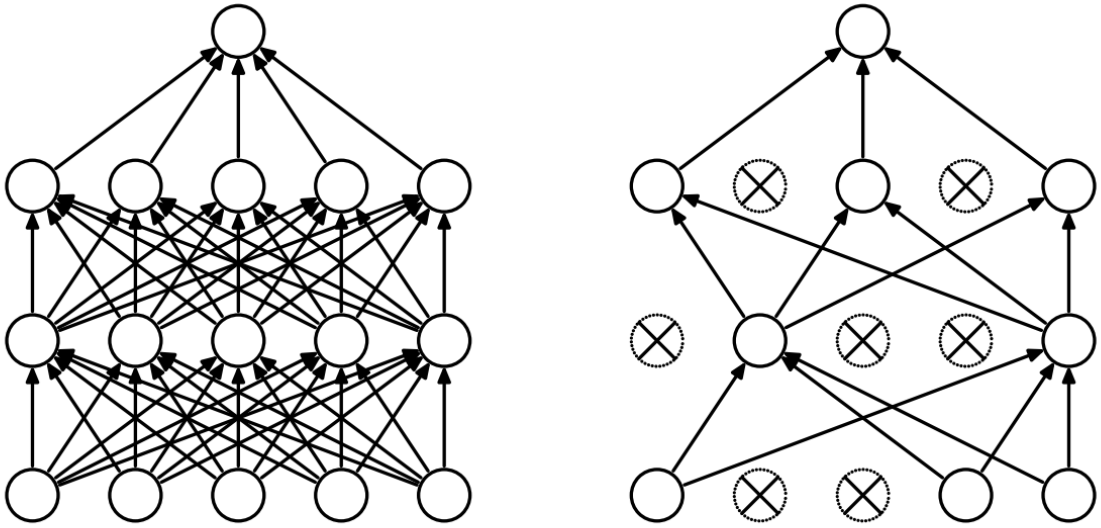


Training with Early Stopping

1. **Start with small random parameters**
2. **Update the parameters to minimize the training error**
3. **Stop when the validation error fails to decrease**

Dropout

- **Dropout:** Improving generalization by preventing complex co-adaptations on training data
  - **"randomly set some units to zero for each iteration"**
  - helps to break "symmetry" between different units in a layer
  - approximately combines exponentially many (~2^n, n is the number of hidden units) different sub-networks by random sampling of hidden units

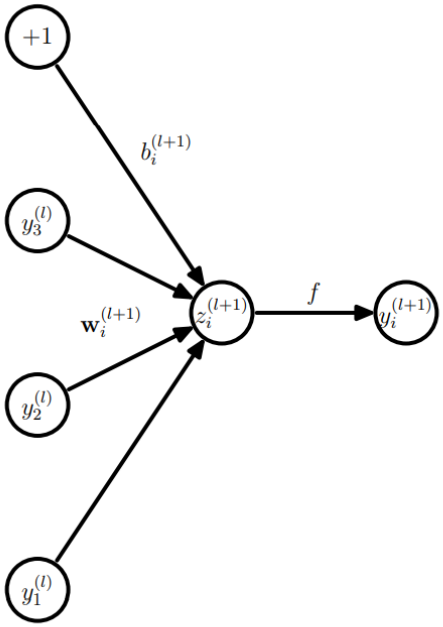


(a) Standard Neural Net

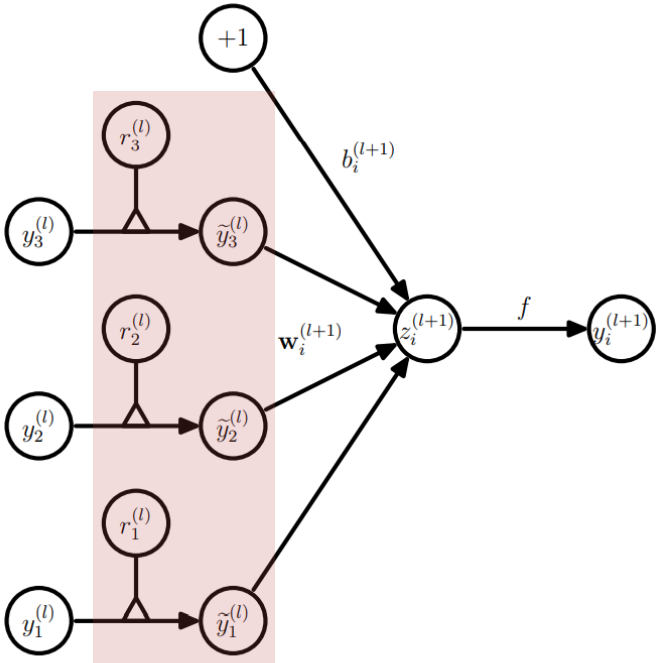
(b) After applying dropout.

Dropout

- **Training Phase:** At each iteration, randomly drop hidden units. Each unit is retained and updated with a probability *p* *(how?)*



(a) Standard network



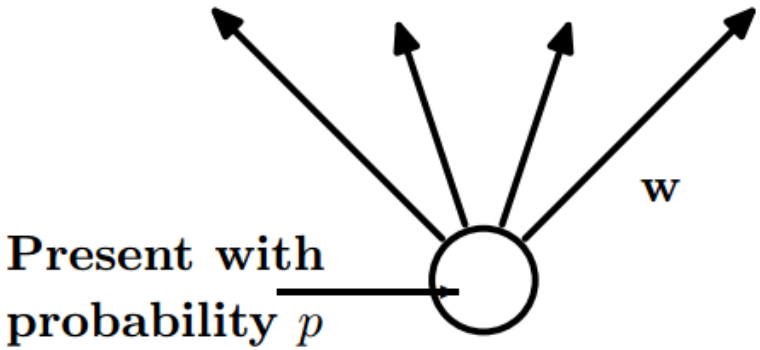
(b) Dropout network

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

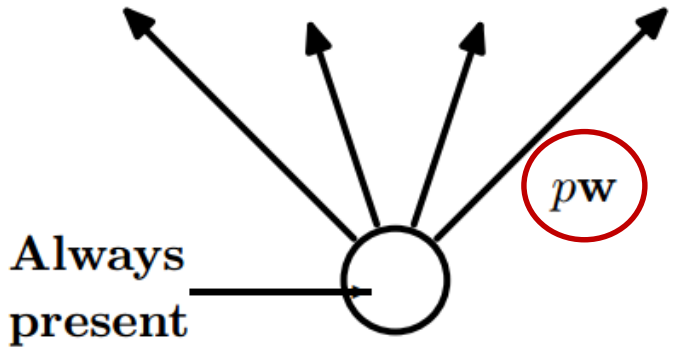
$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

Dropout

- **Test Phase:** Each unit is scaled by *p*



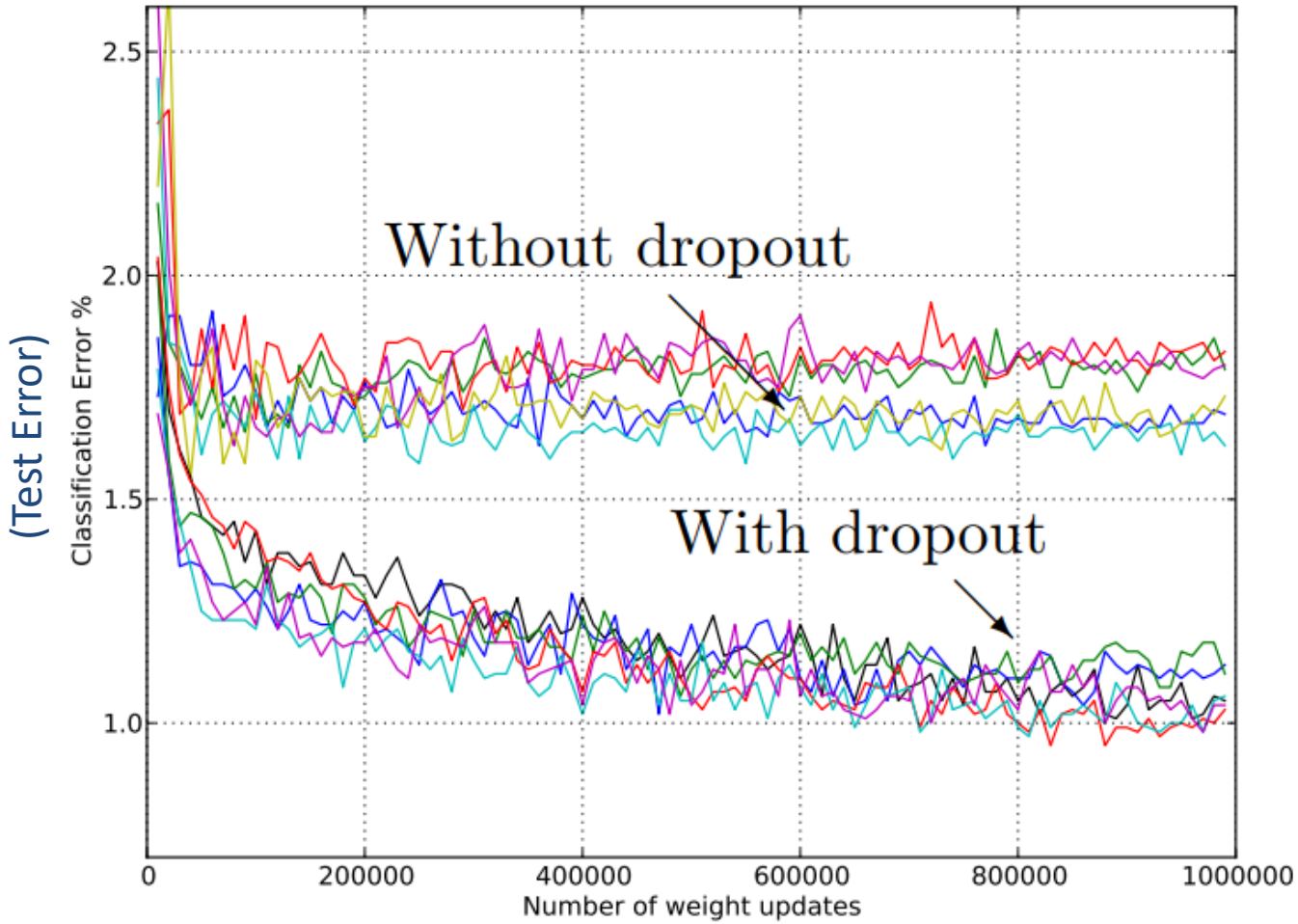
(a) At training time



(b) At test time

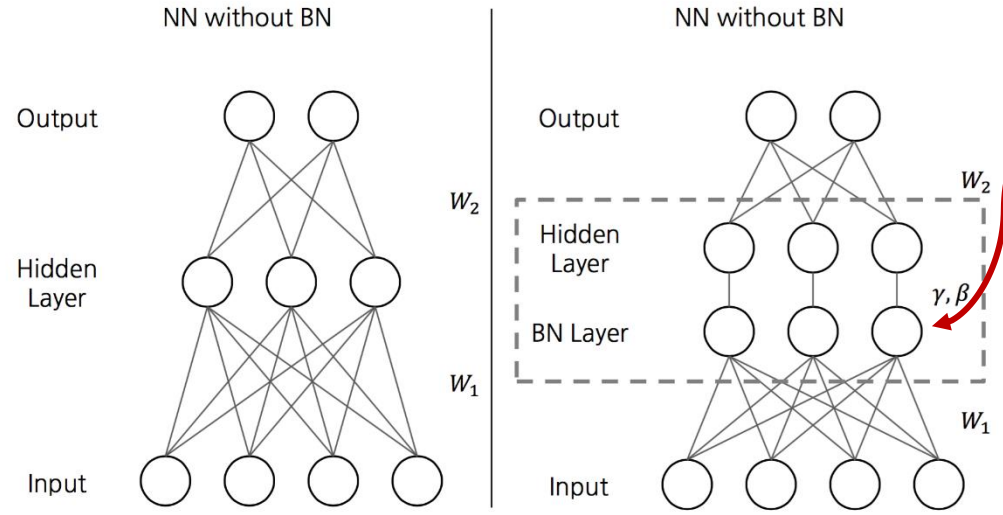
Dropout

- ***Example*:** Empirical study (Srivastava et al., 2014)



Batch Normalization

- **Batch Normalization:** Reparameterization of a neural network to improve optimization
  - Normalize distribution of each hidden unit to *zero mean and one variance* across each minibatch
    - Training phase: μ and σ are calculated from the minibatch
    - Test phase: use running averages of μ and σ during training
  - Simply normalizing each hidden unit may change what it can represent
    - learn the scale and shift (parameters γ and β) for each hidden unit
  - Benefits of Batch Normalization
    - accelerate the training of a neural network
    - reduce effects of exploding and vanishing gradients
    - improve generalization



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.

Multitask Learning

- **Multitask Learning:** Improving generalization by pooling the data arising out of several related tasks. Different tasks share the input and some hidden layers.

** multi-class/multi-label classification?*

Example data for multi-task learning

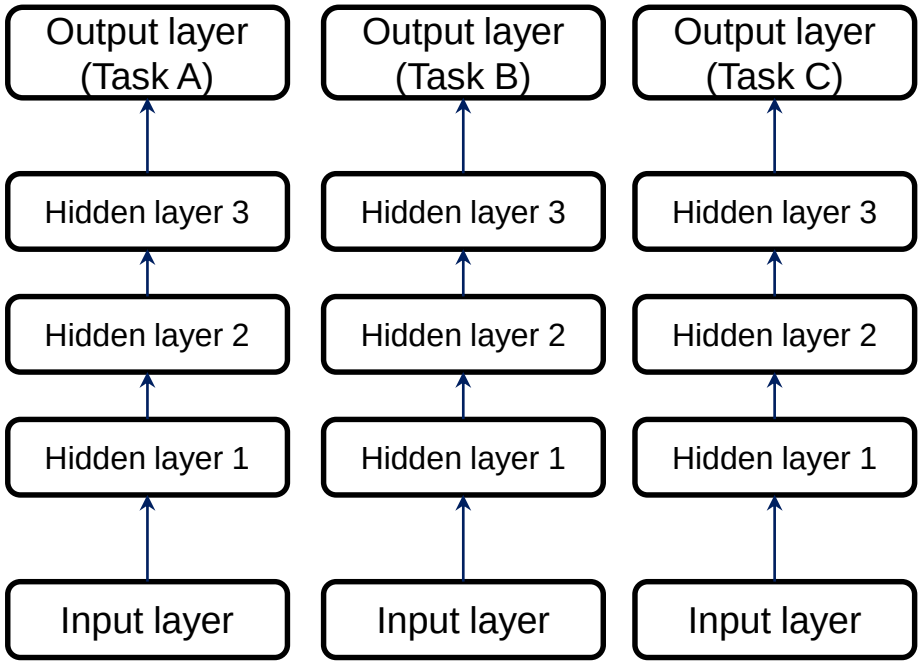
id	X_1	X_2	X_3	…	X_d	Y_A	Y_B	Y_C
1	x_{11}	x_{12}	x_{13}	…	x_{1d}	y_{1a}	–	y_{1c}
2	x_{21}	x_{22}	x_{23}	…	x_{2d}	-	y_{2b}	y_{2c}
3	x_{31}	x_{32}	x_{33}	…	x_{3d}	-	y_{3b}	-
4	x_{41}	x_{42}	x_{43}	…	x_{4d}	-	y_{4b}	-
5	x_{51}	x_{52}	x_{53}	…	x_{5d}	y_{5a}	-	y_{5c}
6	x_{61}	x_{62}	x_{63}	…	x_{6d}	-	y_{6b}	y_{6c}
7	x_{71}	x_{72}	x_{73}	…	x_{7d}	-	y_{7b}	-
…	…	…	…	…	…	…	…	…

Multitask Learning

- **Multitask Learning Approach for Neural Networks: Parameter Sharing**
  - **Task-specific parameters:** only for the target task, benefit from the data of the target task
  - **Generic parameters:** shared across all the tasks, benefit from the pooled data of all the tasks

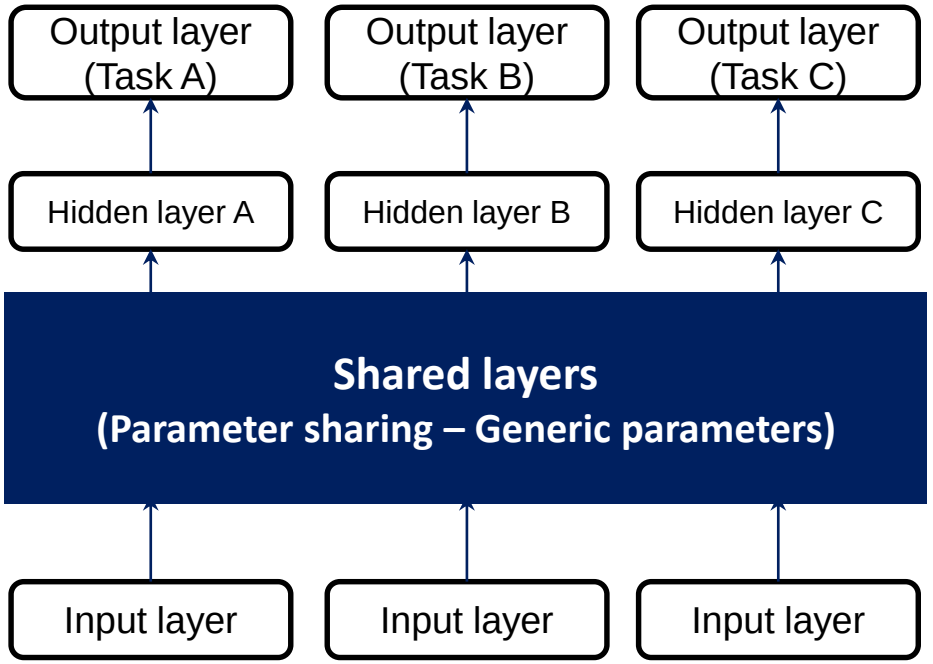
Independent objective functions

$J_A(\boldsymbol{\theta}_A)$ $J_B(\boldsymbol{\theta}_B)$ $J_C(\boldsymbol{\theta}_C)$



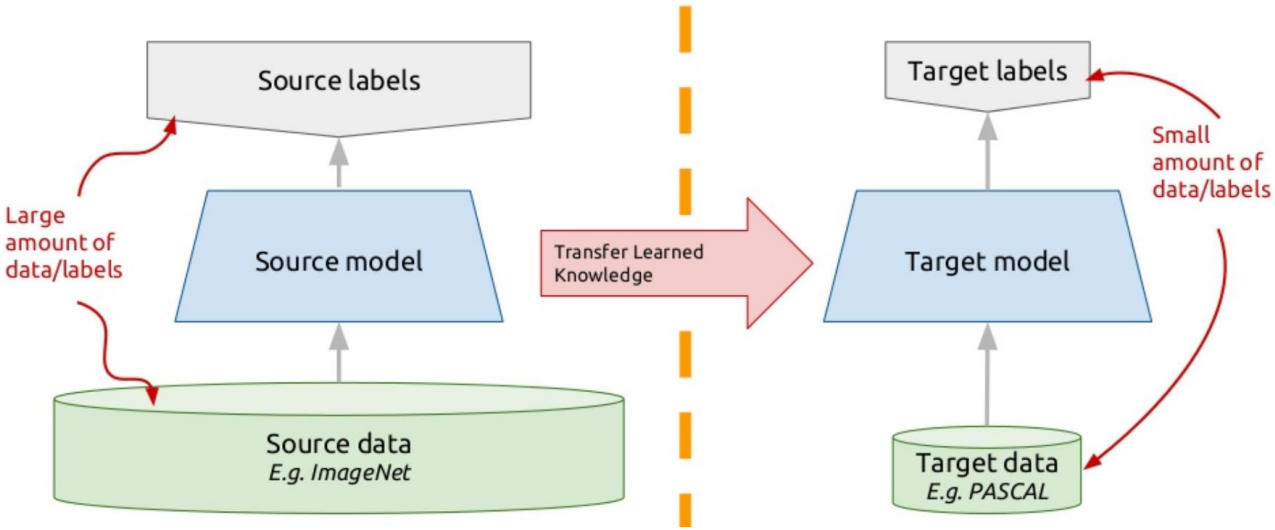
Joint objective function

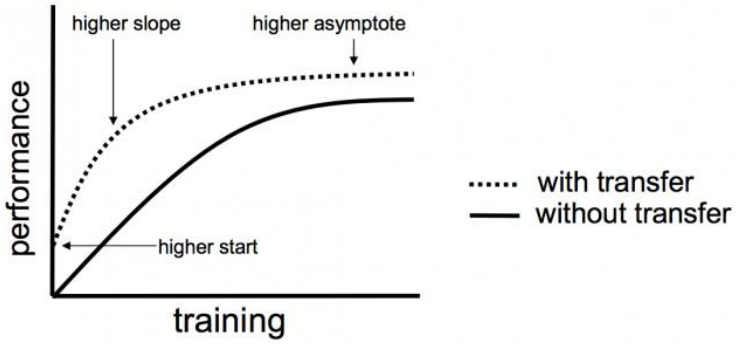
$J(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B, \boldsymbol{\theta}_C, \boldsymbol{\theta}_{pool})$
$= \alpha_A J_A(\theta_A, \boldsymbol{\theta}_{pool}) + \alpha_B J_B(\boldsymbol{\theta}_B, \boldsymbol{\theta}_{pool}) + \alpha_c J_C(\boldsymbol{\theta}_C, \boldsymbol{\theta}_{pool})$



Transfer Learning

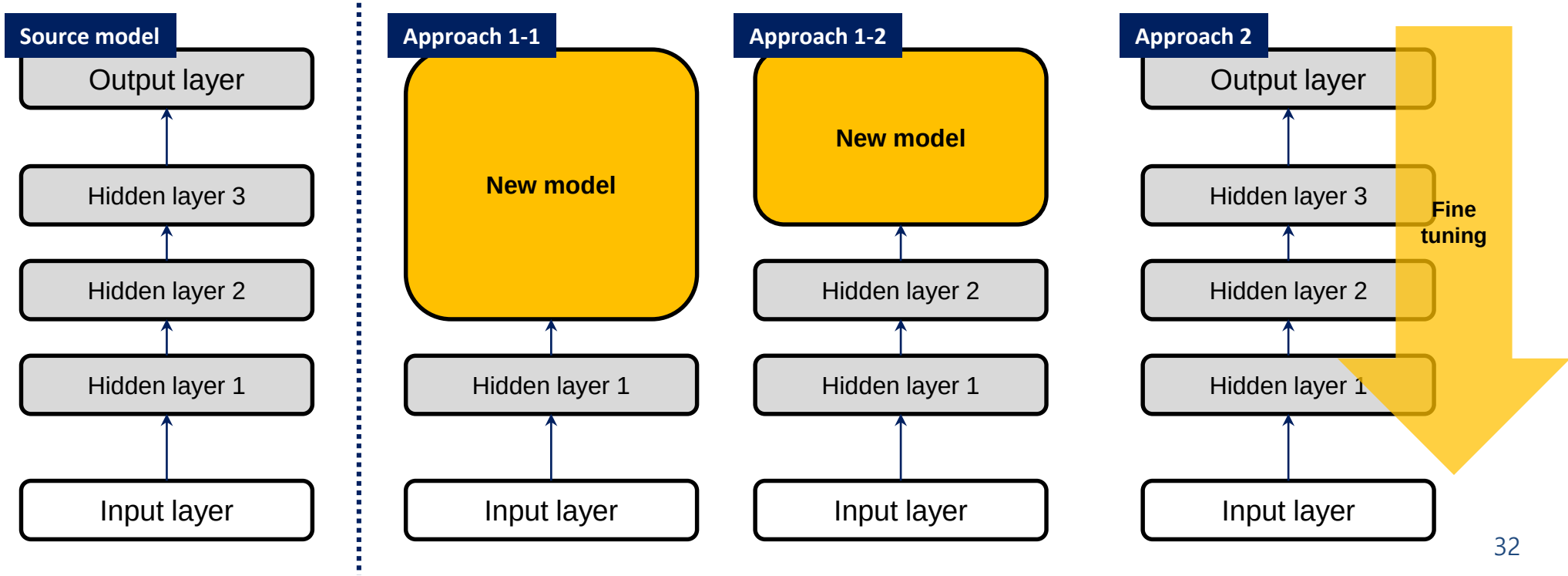
- **Transfer Learning**: Improving generalization in a new task through the transfer of knowledge from a related task that has already been learned.
  - Take a “source model” trained for a related “source task”, and adapt the source model to build a “target model” for the “target task”
    → faster/efficient training, requires less training data





Transfer Learning

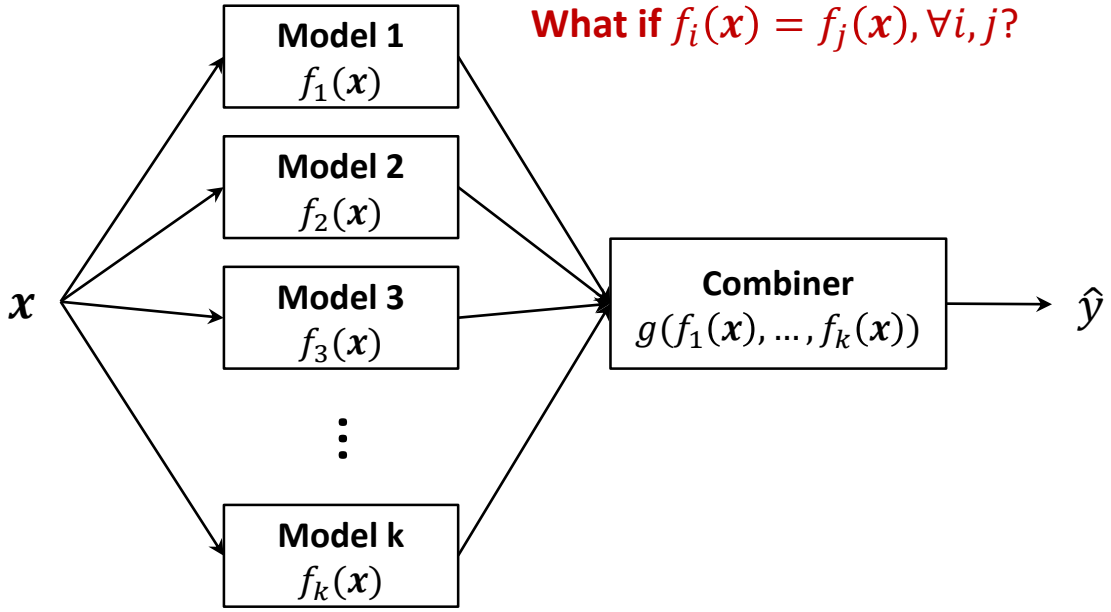
- **Transfer Learning Approaches for Neural Networks**
  - Use the source model as a **feature extractor**, train a off-the-shelf model with new data
    - which hidden layer (from general to specific)?, how to train the new model?
  - Use the source model as an **initialization**, fine-tune the model with new data
    - learning rate for backpropagation?



Ensemble Learning

- **Ensemble**: Improving generalization by combining several different models.
  - The output of the *i*-th base model: $\hat{y} = f_i(\boldsymbol{x})$
  - The output of an ensemble: $\hat{y} = g(f_1(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$

$f_i(\boldsymbol{x}) \neq f_j(\boldsymbol{x})$ **for diversity**!



Ensemble Learning

- **Two main steps of constructing an ensemble:** (1) Diversity generation, (2) Combination
  - **Diversity Generation**
    - **Data manipulation**
      : Data points / Variables
      : Sampling / Partitioning
    - **Learning algorithm**
      : Homogeneous / Heterogeneous
    - …
  - **Combination**
    - Equal / Weighted
    - Linear / Non-linear
    - Static / Dynamic
    - …
  - Different ensemble methods construct the ensemble of models in different ways.
  - Some of them work as **regularization**.

Ensemble Learning: Bagging

- **Bagging ("bootstrap aggregating")**
  - **One of the most popular ensemble methods. Simple but very effective!**

  - **[Training Phase]** for a training set *D*,
    1. Take *k* bootstrap samples (sampling with replacement) from *D*
    2. Train *k* different base models on these bootstrap samples

    Output: $f_1(\boldsymbol{x}), \dots, f_k(\boldsymbol{x})$

  - **[Test Phase]** for a test data point $\boldsymbol{x}$,
    1. let all models predict
    2. take an average (for regression) or majority vote (for classification)

    Output: $\hat{y} = \frac{1}{k}\sum_i f_i(\boldsymbol{x})$ (for regression) or $\hat{y} = \underset{j}{\operatorname{argmax}} \sum_i I(f_i(\boldsymbol{x}) = j)$ (for classification)

Ensemble Learning: Bagging

- **Bagging ("bootstrap aggregating")**
  - If the base models make **uncorrelated** errors, then their ensemble can improve performance.
  - **Bagging reduces the variance of the prediction.** *why?*

  - ***Example*:** Regression Case
    - $\hat{y} = \frac{1}{k}\sum_i \hat{y}_i$
    - $\mathrm{Var}[\hat{y}] = \frac{1}{k^2}\mathrm{Var}[\sum_i \hat{y}_i] = \frac{1}{k^2}\{\sum_i \mathrm{Var}[\hat{y}_i] + 2\sum_{i>j} \mathrm{Cov}[\hat{y}_i, \hat{y}_j]\}$
    - if $\mathrm{Var}[\hat{y}_i] = \delta^2$ and $\mathrm{Cov}[\hat{y}_i, \hat{y}_j] = 0, \forall i, j \rightarrow \mathrm{Var}[\hat{y}] = \frac{\delta^2}{k}$