Aside 3:
Creating *predictors*

Feed-Forward Neural Network

Target function: $y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d$

Dataset

$$D := \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$

Representation – *e.g., Feed-forward neural network*

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b, \quad \boldsymbol{W} \in \mathbb{R}^{h \times d}, \ \boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^h, b \in \mathbb{R}$$

Evaluation – *e.g., Mean Squared Error*

$$L(D) = \frac{1}{N} \sum_{i=1}^{N} (\tilde{y}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

Optimization – *gradient descent and variants*

$$\Delta \boldsymbol{W} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{W}} L(\tilde{y}^{(i)}, y^{(i)}) \qquad \Delta \boldsymbol{b} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{b}} L(\tilde{y}^{(i)}, y^{(i)})$$

$$\Delta \boldsymbol{w} = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial \boldsymbol{w}} L(\tilde{y}^{(i)}, y^{(i)}) \qquad \Delta b = -\eta \frac{1}{N} \sum_{D} \frac{\partial}{\partial b} L(\tilde{y}^{(i)}, y^{(i)})$$

Predictors?

Optimization:
The aim is finding the parameters that make the representation best approximating the target function over the dataset

Fundamental question:
How good is the approximator when applied to data items that are *not* in the dataset?

Overfitting

When the training process becomes too specific to the training set

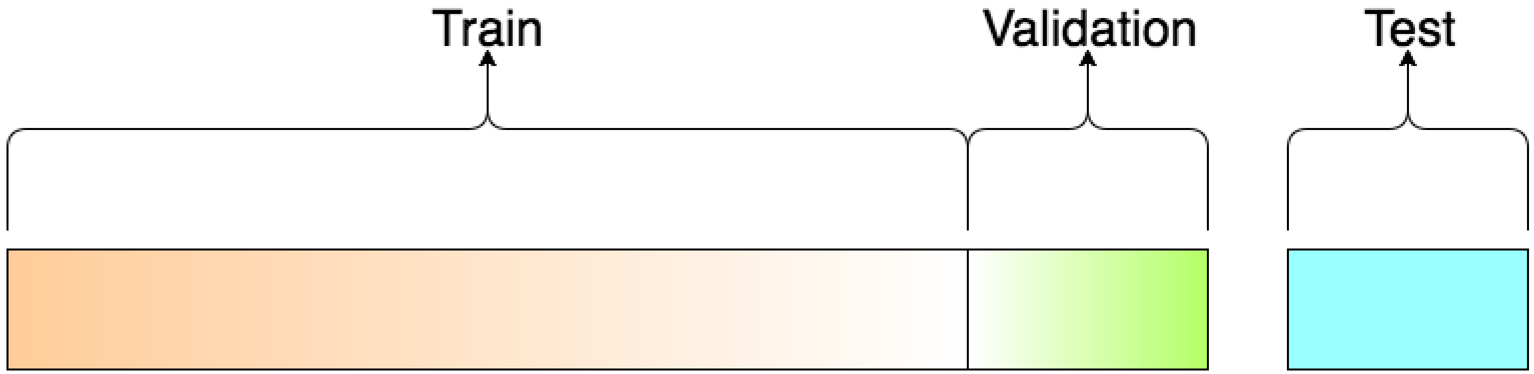
- ***Training set, validation set, test set***

  Splitting the dataset

$$D = D_{train} \cup D_{val} \cup D_{test}$$

$$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N} = \{(\boldsymbol{x}^{(j)}, y^{(j)})\}_{j=1}^{N_{train}} \cup \{(\boldsymbol{x}^{(k)}, y^{(k)})\}_{k=1}^{N_{val}} \cup \{(\boldsymbol{x}^{(l)}, y^{(l)})\}_{l=1}^{N_{test}}$$

$$N_{train} \gg N_{val}, N_{test}$$



Overfitting

When the training process becomes too specific to the training set

- ***Training set, validation set***

Splitting the dataset

$$D = D_{train} \cup D_{val} \cup D_{test}$$

$$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N} = \{(\boldsymbol{x}^{(j)}, y^{(j)})\}_{j=1}^{N_{train}} \cup \{(\boldsymbol{x}^{(k)}, y^{(k)})\}_{k=1}^{N_{val}} \cup \{(\boldsymbol{x}^{(l)}, y^{(l)})\}_{l=1}^{N_{test}}$$

$$N_{train} \gg N_{val}, N_{test}$$

Training is made on D_{train} only

At each *epoch* — *when the whole D_{train} has been processed*

the loss function is evaluated on D_{val}

After some epochs, the performance on D_{val} might get <u>worse</u>

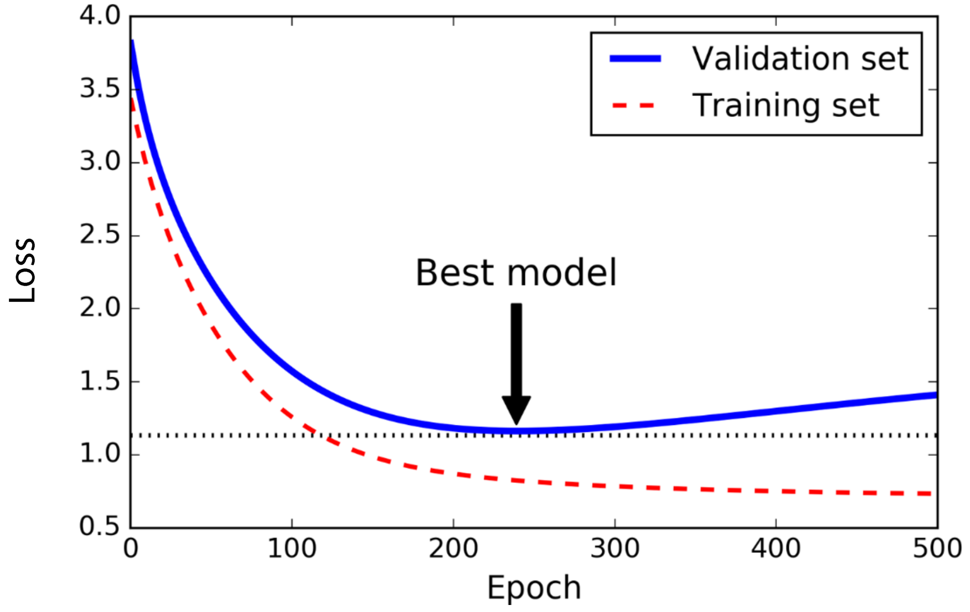


Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

k-Fold Cross-Validation

- ***One dataset, multiple splits***
  1) Divide the dataset into k splits (i.e. *folds*)
  2) Use k - 1 folds for training and 1 fold for testing
  3) Unless all combinations have been considered, change combination and go back to 2)

Consider the *average test loss* across all possible combinations

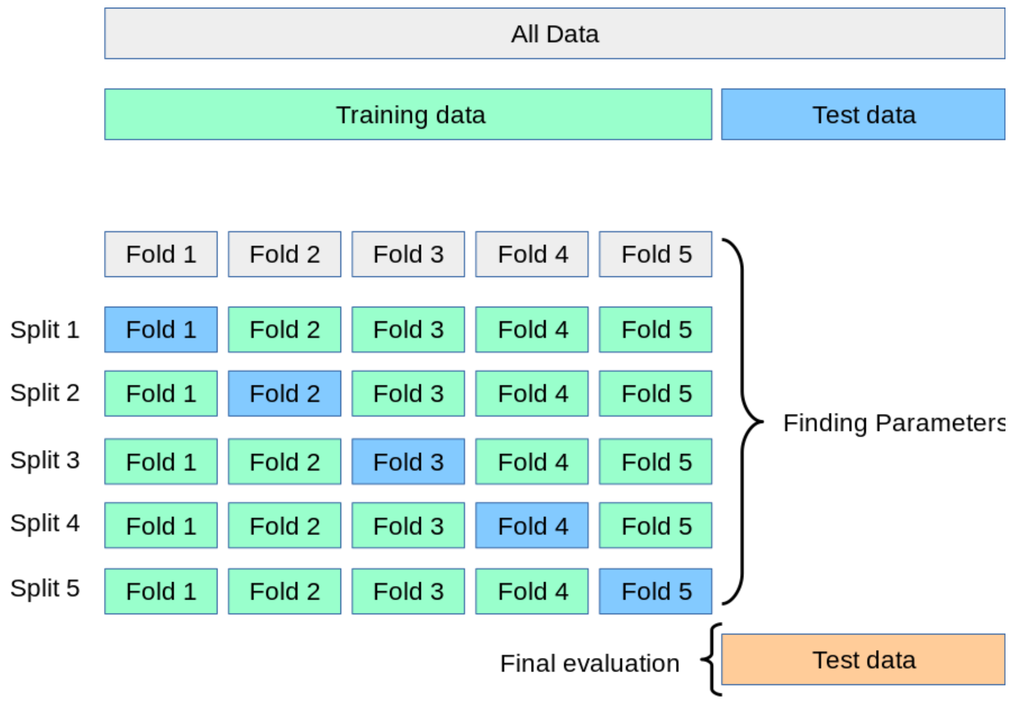


Image from https://www.kdnuggets.com/2020/01/data-validation-machine-learning.html