



UNIVERSITÀ
DI PAVIA

Aside 2: *Creating Predictors*

Feed-Forward Neural Network

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

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

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

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

Evaluation – e.g., Mean Squared Error

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

Optimization – gradient descent and variants

$$\begin{aligned} \Delta \mathbf{W} &= -\eta \frac{1}{N} \sum_D \frac{\partial}{\partial \mathbf{W}} L(\tilde{y}^{(i)}, y^{(i)}) & \Delta \mathbf{b} &= -\eta \frac{1}{N} \sum_D \frac{\partial}{\partial \mathbf{b}} L(\tilde{y}^{(i)}, y^{(i)}) \\ \Delta \mathbf{w} &= -\eta \frac{1}{N} \sum_D \frac{\partial}{\partial \mathbf{w}} L(\tilde{y}^{(i)}, y^{(i)}) & \Delta b &= -\eta \frac{1}{N} \sum_D \frac{\partial}{\partial b} L(\tilde{y}^{(i)}, y^{(i)}) \end{aligned}$$

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

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

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

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



Overfitting

When the training process becomes too specific to the training set

■ Training set, validation set

Splitting the dataset

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

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

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

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 worse

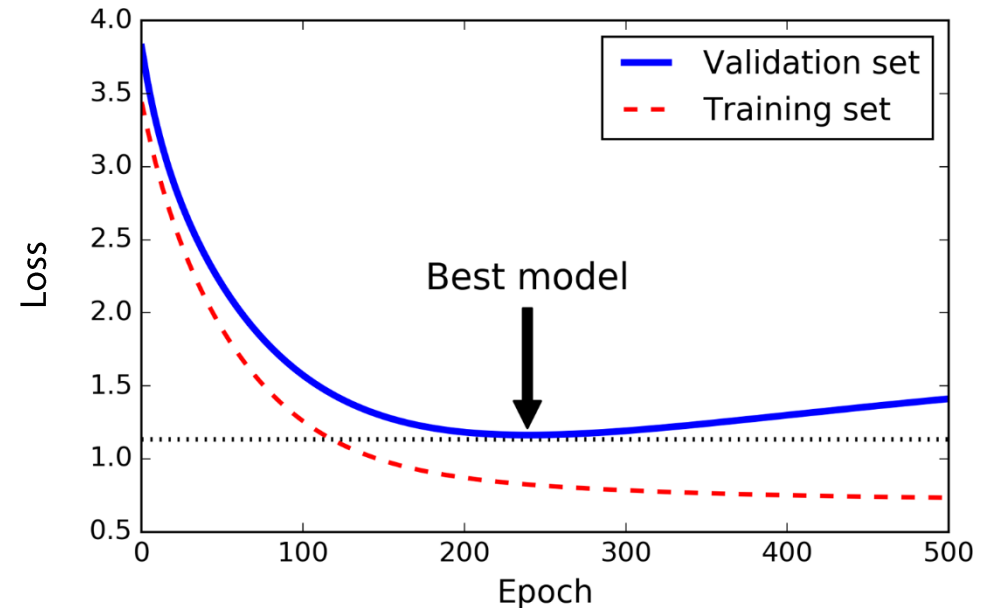


Image from <https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html>

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

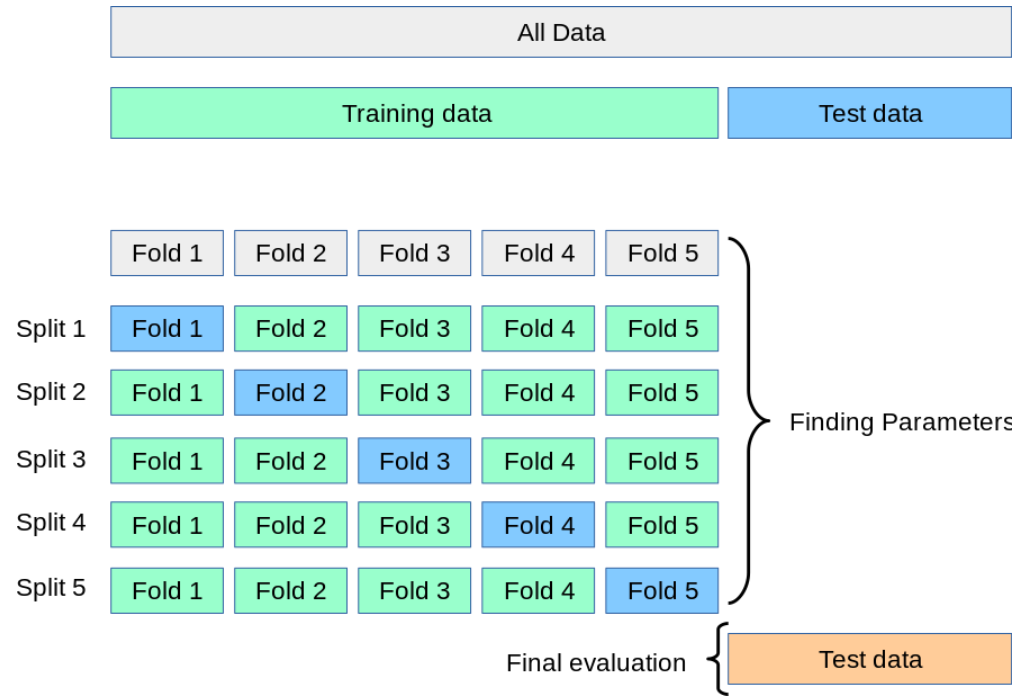


Image from <https://www.kdnuggets.com/2020/01/data-validation-machine-learning.html>