

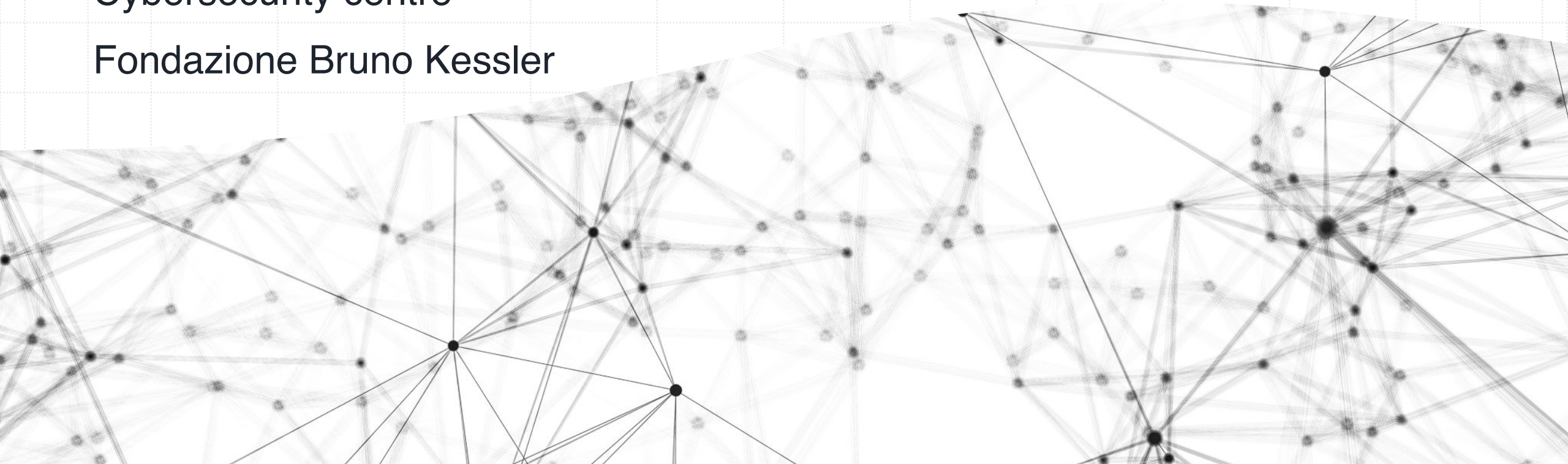


# Hyper-parameters tuning

**Roberto Doriguzzi Corin**

Cybersecurity centre

Fondazione Bruno Kessler





# Outline

- Relevant hyper-parameters in cyber-security
- Available techniques (e.g., grid search)
- **Laboratory:** hyper-parameters tuning with grid-search and randomized search



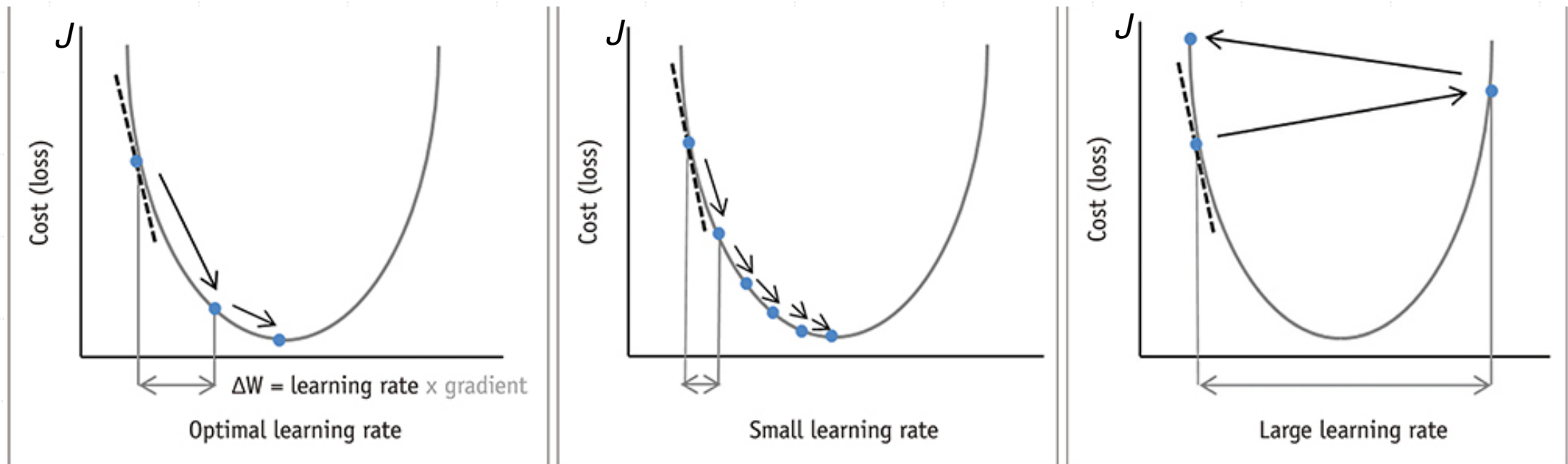
# Definition

- **Hyperparameters tuning** is the process of finding the set of optimal hyperparameters for a learning algorithm
- Hyperparameters include:
  - Architectural parameters (e.g., number of hidden layers and neurons)
  - Training parameters (e.g., learning rate, batch size)

# Training hyper-parameters (1)

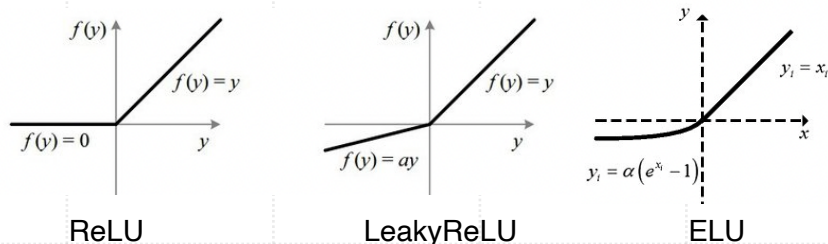
- **Batch size:** size of the mini-batch used to compute the next step of gradient descent. It can have a significant impact on the training time and it is often chosen based on the capability of the machine to execute parallel computation. It is recommended to use the largest batch that fits the GPU RAM (when GPU is available)
- **Learning rate:** determines the size of the gradient descent step. If it is **too small**, the training process will have to go through many iterations before convergence. If it is **too large**, it may make the algorithm to diverge.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$



# Training hyper-parameters (2)

- **Optimizers:** there are faster optimizers than batch/stochastic/mini-batch gradient descent. One of these is the Adam (Adaptive Momentum Estimation) algorithm, which usually converges faster than SGD
- **Activation function:** usually the **ReLU** activation function works well for the hidden layers. Other options for hidden layers are **Leaky ReLU**, **ELU**, among others

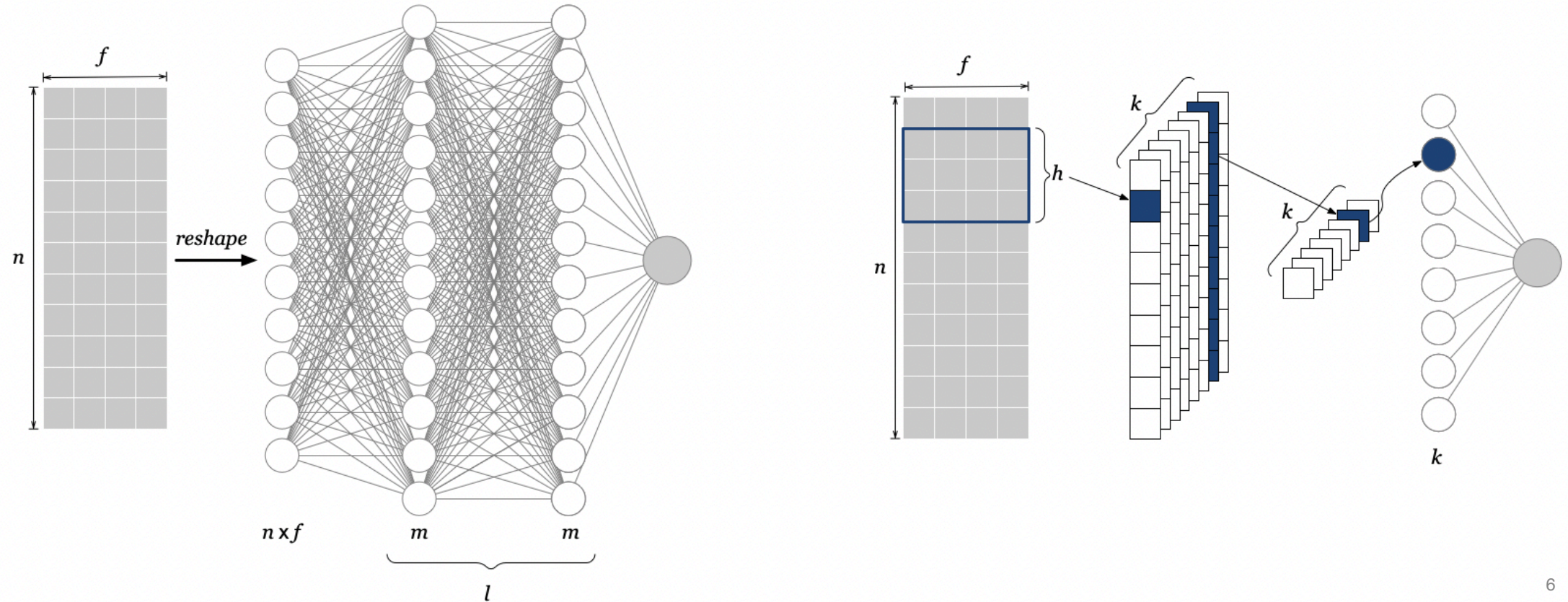


- Leaky ReLU and ELU try to avoid the problem of dying neurons. ELU converges faster but is slower than ReLU at test time (the exponential).
- LeakyReLU is a generalisation of ReLU, with coefficient  **$a$**  as a hyperparameter.
- **$\alpha$**  is an hyperparameter for ELU

- For the output layers, the activation function depends on the problem (e.g., **sigmoid** for binary problems, **softmax** for multi-class problems)

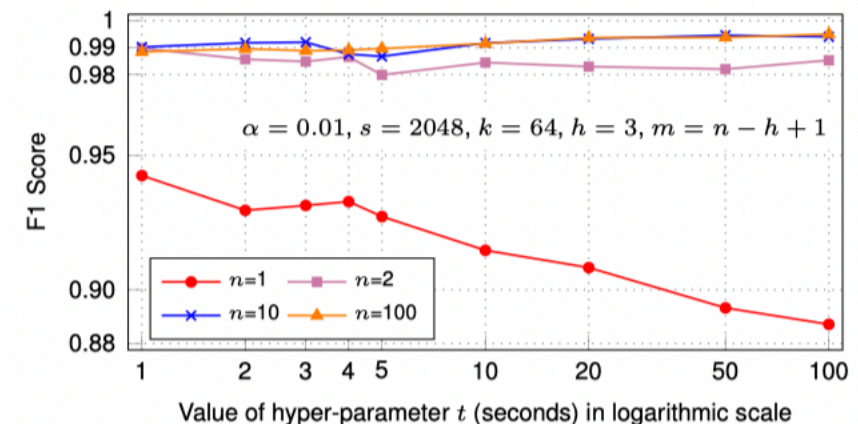
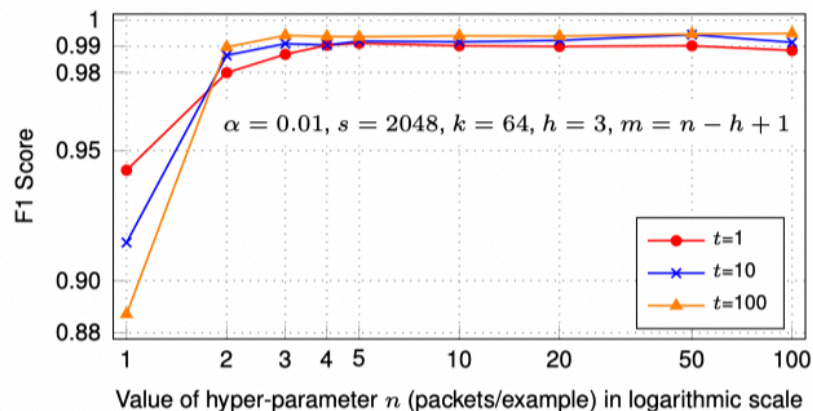


# Architectural Hyper-parameters



# Problem-specific hyperparameters

- **Maximum number of packets/flow:** the input layer has a fixed shape that you must set. This shape is determined by the flow representation of choice. In the case of packet-based representation, the maximum number of packets/flow must be set in advance
- **Time window:** in a real-world application scenarios, the network traffic is collected for a certain time window and then sent to the IDS for analysis. The size of the time window must be also set, as it determines how the traffic flows are collected.



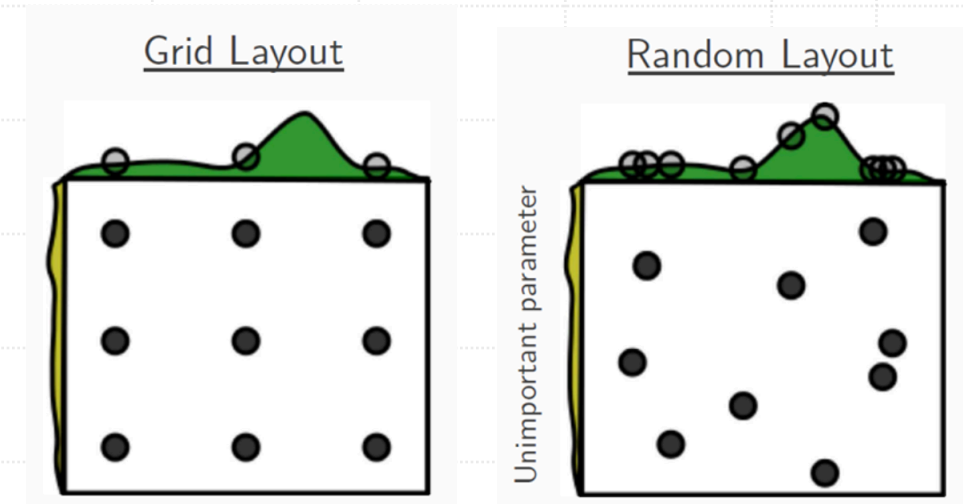
# Grid search

- Given a set of hyperparameters and a list of possible choices for each of them, a grid search algorithm will train a model multiple times, each time with a different combination of hyperparameters. E.g.,
  - $LR = [0.1, 0.01, 0.001]$
  - $BATCH\_SIZE = [1024, 2048]$
  - $CONV\_KERNELS = [1, 2, 4, 8, 16, 32, 64]$
  - $PACKETS\_PER\_FLOW = [1, 2, 3, 4, 5, 10, 20, 50, 100]$
- The algorithm will train the model for  $3 \times 2 \times 7 \times 9 = 378$  times.
- The best combination is the one that produces the highest accuracy on the validation set.



# Randomized search

- Grid search is a good approach when the number of combinations of hyperparameters is limited and the model size permits fast training
- Alternatively, with the **random search approach** the model is trained using random combinations of hyperparameters
- With randomized search, one can **set the maximum number of combinations to test**, so to control the total training time
- As with grid search, the best combination (among those randomly selected) is the one that produces the highest accuracy on the validation set



Source: <https://www.jeremyjordan.me/>

# Randomized search strategies

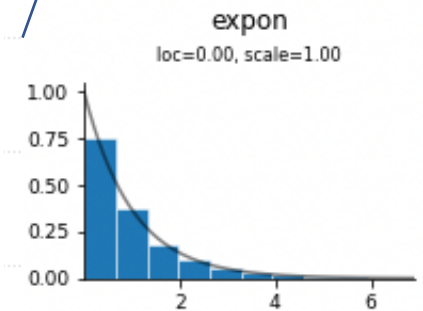
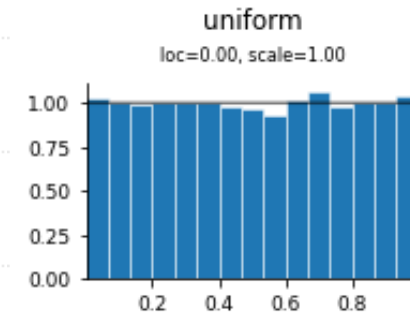


**Pre-defined set:** select a random number of random combinations from a given set



**Random:** select a random number based on a statistical distribution of the hyper-parameter

Expert knowledge that is given to the search process





# Dataset split

- The dataset must be split into **training**, **validation** and **test** set.
- It is common to use 90% of the data for training and 10% for testing, with 10% of the training set used for validation
  - More precisely training set 81%, validation set 9%, test set 10%

The proportions **depend on the size of the dataset**: in a dataset with 10 Millions of samples, 1% of the data (100000 samples) devoted to testing could be enough.

# Kfold Cross-validation

- When we have limited data, dividing the dataset into Train and Validation sets may cause some data points with useful information to be excluded from the training procedure, and the model fails to learn the data distribution properly.
- One way to solve this problem is called cross-validation. The following procedure is followed for each of the  $k$  “folds”:
  - A model is trained using  $k-1$  of the folds as training data;
  - the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).
- The performance of the model is computed as the **average validation score** of the model obtained on  $k$  different splits
  - Drawback: the training time is multiplied by the number of validation sets  $k$ .

Dataset				
Training set				Test set
Fold1	Fold2	Fold3	Fold4	
Fold1	Fold2	Fold3	Fold4	
Fold1	Fold2	Fold3	Fold4	
Fold1	Fold2	Fold3	Fold4	

*The number of folds  $k$  is usually set between 5 and 10*



# Stratified KFold

- Stratified Kfold ensures that the folds are made by preserving the percentage of samples for each class.
- Thus, if the training set is balanced with 50% benign samples and 50% DDoS samples, each of the folds will be balanced in the same way
- Stratified Kfold is the default setting in the python libraries for cross-validation
  - E.g., the sci-kit method GridSearchCV



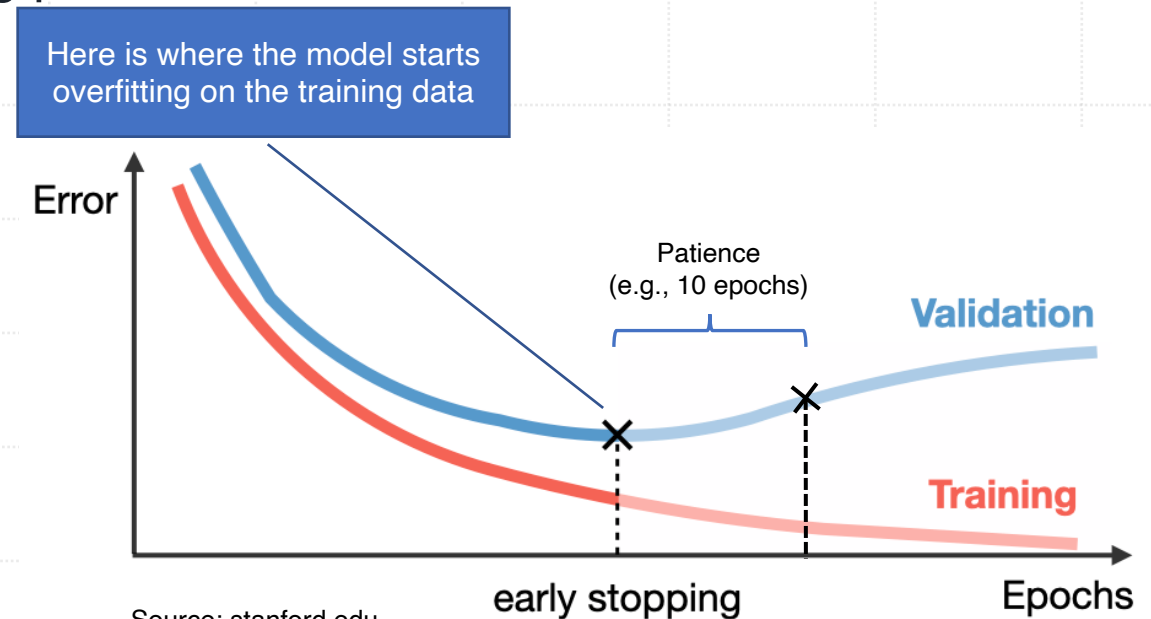
# Early stopping


Given a combination of hyperparameters, when do we stop training and jump to the next combination?

Early stopping is a strategy to stop the training process when the error on the validation set reaches the minimum value.

Usually the curves are not so smooth and it can be hard to understand if we have reached the minimum or not.

**Patience:** one solution is to stop only if the error stays above the minimum for a pre-defined number of epochs.





# Implementation of a hyperparameter tuning process

1. Add early-stopping to Grid Search and test by adding more folds
2. Add training-specific hyper-parameters such as *learning rate* and *batch size*
3. Implement a Randomized Search with sci-kit libraries using a continuous variable for the learning rate
4. Compare Grid Search with Randomized Search in terms of execution time and accuracy of the best model on the validation set
5. Replace the model architecture with a CNN and the relevant hyperparameters to tune (e.g., number of kernels, kernel height, max pooling, etc)