



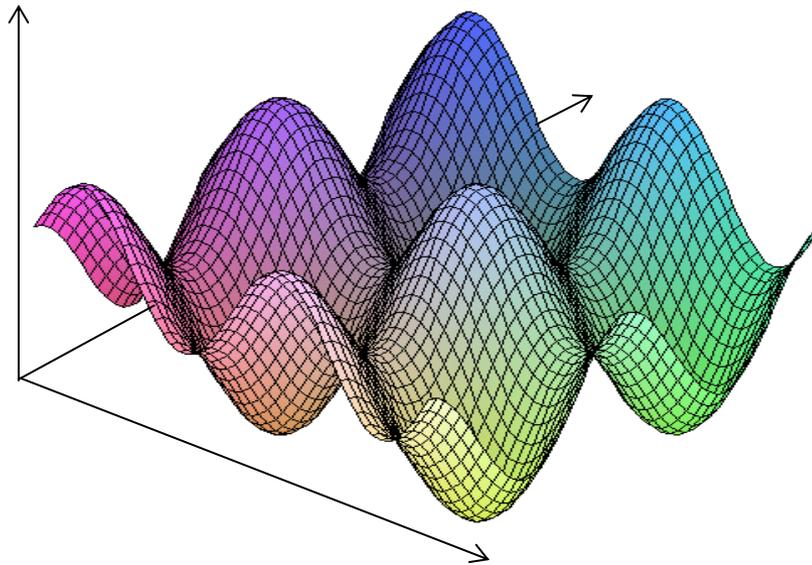
Artificial Neural Networks for Fault Diagnosis: Exercise Session



Joaquín Figueroa
joaquineduardo.figueroa@polimi.it

Exercise 1: function regression

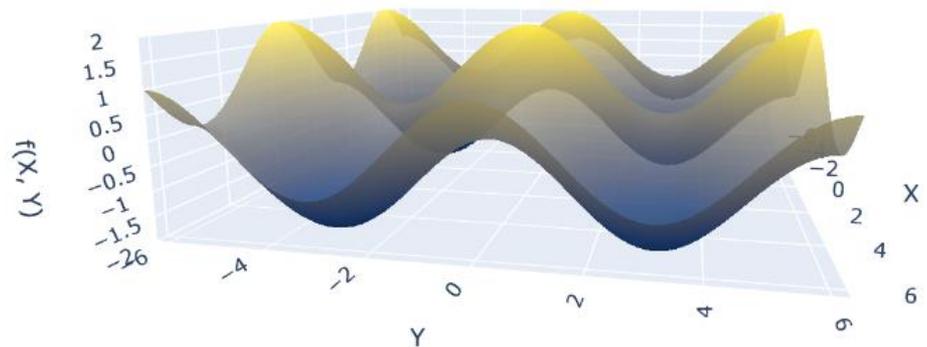
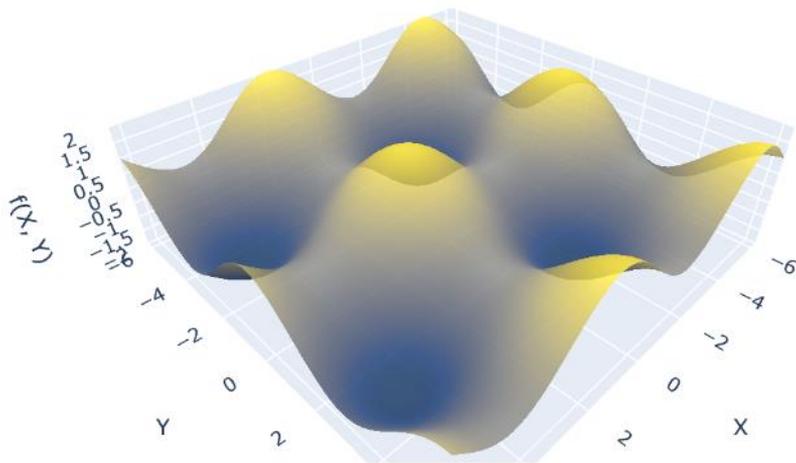
$$f(x, y) = \sin(x) + \cos(y) \quad x \in [-2\pi, 2\pi], y \in [-2\pi, 2\pi]$$



**Train an ANN to reproduce the analytic function $f(x,y)$.
Show the capability of the net to estimate the values of a surface with
a large number of peaks and valleys (local maxima and minima)**

Step 0: Plot the analytic function

```
# Plot the analytical function  $f(x, y) = \sin(x) + \cos(y)$   
plot_function()
```



- The first step is to create samples of the function. Each pattern is composed by three signals:

	input ₁	input ₂	output
ex ₁ :	x ₁	y ₁	f ₁
ex ₂ :	x ₂	y ₂	f ₂
...
ex _n :	x _n	y _n	f _n

The number of patterns has to be large enough to guarantee the possibility of:

- 1) training the ANN → good coverage of all the training space. In this case, since f has many peaks and valleys, we need a large number of training pattern (e.g. $N_{train}=1000$)
- 2) Test the performance of the developed ANN → we need a test set of n_{test} patterns (e.g. $n_{test}=1000$)

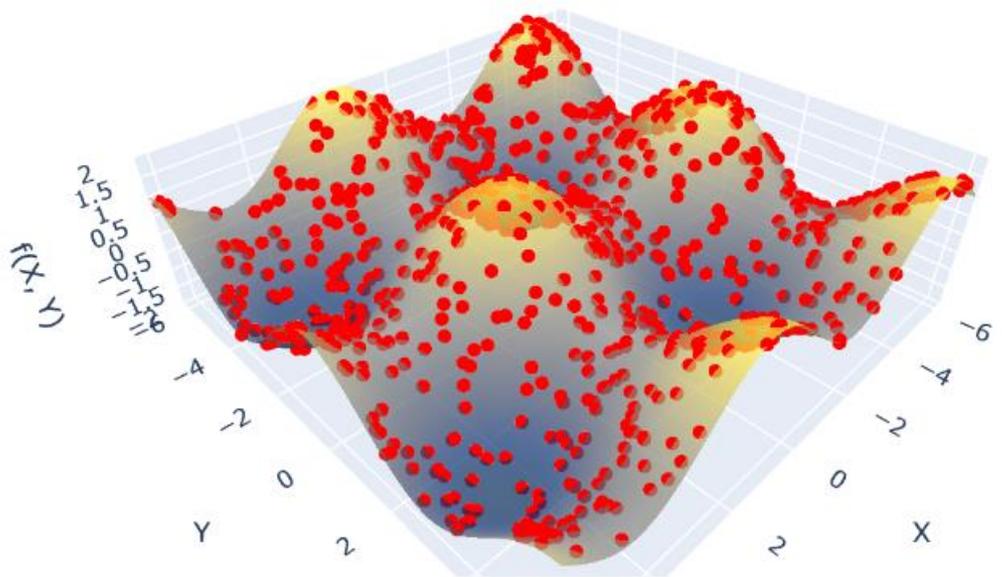


We simulate a dataset of $n=(n_{train}+n_{test})= 2000$ patterns

```
# Generate data
n = 1600 # Total number of patterns
x, y, f = generate_data(n)

# Split into training and test sets
n_train = n // 2 # Number of training samples
x_train = x[:n_train]
y_train = y[:n_train]
f_train = f[:n_train]
x_test = x[n_train:]
y_test = y[n_train:]
f_test = f[n_train:]

# Plot training data
plot_training_data(x_train, y_train, f_train)
```



Step 2: data normalization in [0,1]

```
# Normalize data
x_norm, y_norm, f_norm, data_ranges = normalize_data(x, y, f, n_train)
x_min, x_max, y_min, y_max, f_min, f_max = data_ranges
```

Step 3: Create training sets

```
# Prepare training and test sets
P_train = np.vstack((x_norm[:n_train], y_norm[:n_train])).T # Training set input
T_train = f_norm[:n_train] # Training set output (target)
P_test = np.vstack((x_norm[n_train:], y_norm[n_train:])).T # Test set input
T_test = f_norm[n_train:] # Test set output
```

Step 4: Build and train the artificial neural network for 500 epochs.

```
# Build and train the model
model = build_model()
model = train_model(model, P_train, T_train, epochs=50)
```

```
Epoch 1/50
25/25 ————— 1s 2ms/step - loss: 0.4026
Epoch 2/50
25/25 ————— 0s 1ms/step - loss: 0.1957
Epoch 3/50
25/25 ————— 0s 1ms/step - loss: 0.1115
Epoch 4/50
25/25 ————— 0s 2ms/step - loss: 0.0815
Epoch 5/50
25/25 ————— 0s 1ms/step - loss: 0.0679
Epoch 6/50
25/25 ————— 0s 1ms/step - loss: 0.0707
Epoch 7/50
25/25 ————— 0s 1ms/step - loss: 0.0657
Epoch 8/50
25/25 ————— 0s 1ms/step - loss: 0.0660
Epoch 9/50
25/25 ————— 0s 1ms/step - loss: 0.0660
Epoch 10/50
25/25 ————— 0s 2ms/step - loss: 0.0658
Epoch 11/50
25/25 ————— 0s 2ms/step - loss: 0.0667
Epoch 12/50
25/25 ————— 0s 1ms/step - loss: 0.0595
Epoch 13/50
25/25 ————— 0s 1ms/step - loss: 0.0607
Epoch 14/50
25/25 ————— 0s 2ms/step - loss: 0.0606
Epoch 15/50
25/25 ————— 0s 2ms/step - loss: 0.0588
```

Finally, we can verify the performance of our network on the test set made of 1000 test patterns, not yet used for the training.

Step 5: Compute the output of the network given the test set

```
# Make predictions on test set
output_ANN_n = model.predict(P_test).flatten()
```

Step 6: Denormalize the outputs

```
# Denormalize output
output_ANN = denormalize_output(output_ANN_n, f_min, f_max)
```

Step 7: provide a measure of the ANN error on the test set

```
# Calculate Mean Squared Error on test set  
MSE = calculate_mse(output_ANN, f_test)  
print('Mean Squared Error on test set:', MSE)
```

$$MSE = \frac{\sum_{i=1}^{n_{test}} (ANN(i) - true(i))^2}{2 n_{test}}$$

Step 8: Plot the results

```
# Plot the actual vs ANN outputs  
plot_results(x_test, y_test, f_test, output_ANN)
```

- **The training of the net is based on available data representing the input/output non-linear relation**
- **The training phase automatically sets the net parameters (weights), so that to perform the best interpolation of the input/output data**
- **Can the quality of the approximation be improved?**

Exercise 1: Sensibility to the number of neurons in the hidden layer

- What number of neurons in the hidden layer can lead to a satisfactory ANN?

Exercise 2: Sensibility to the number of training patterns

- **What is the minimum number of training patterns necessary to obtain a satisfactory ANN?**

Exercise 2: Prediction of the RUL of an Aircraft Engine

Methodological steps

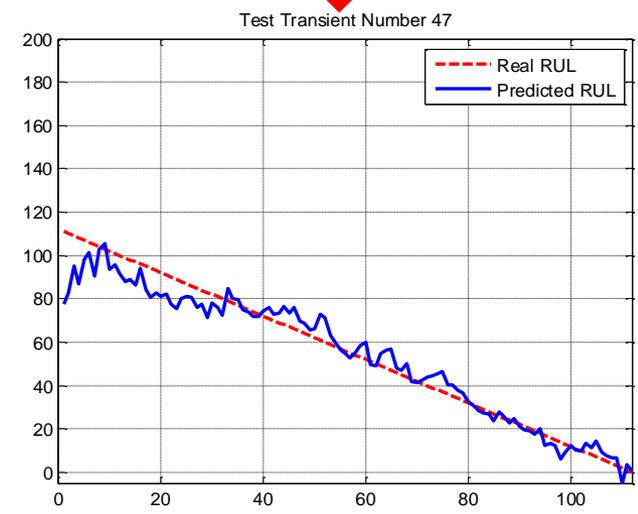


Data Preprocessing
•NORMALIZATION

Construction of degradation indicators

Detection of the degradation onset

RUL Prediction (ANN)





C-MAPPS

Dataset

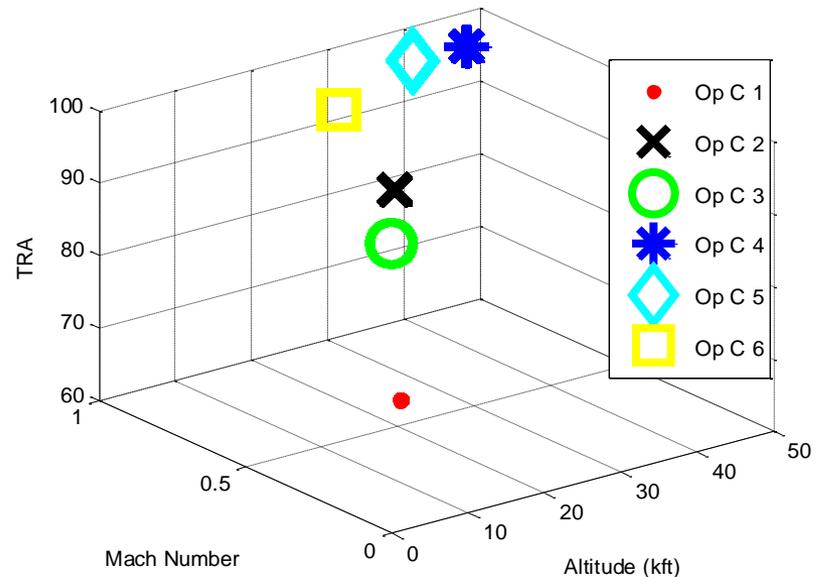
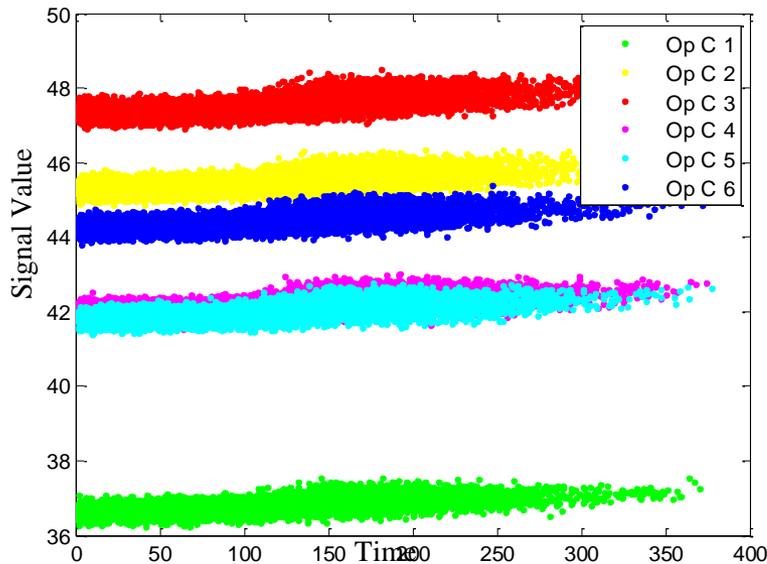


C-MAPPS Dataset: Aircraft Turbofan Engine in Variable Operating Conditions

- 260 Transients (degradation trajectories that reach the failed state)
 - 1 Failure mode
 - 21 Measured Signals
- 6 signals represent Operating Condition



Clustering of the operating Conditions





Data Preprocessing

- **NORMALIZATION**

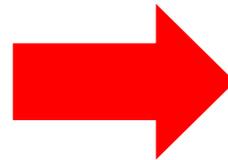
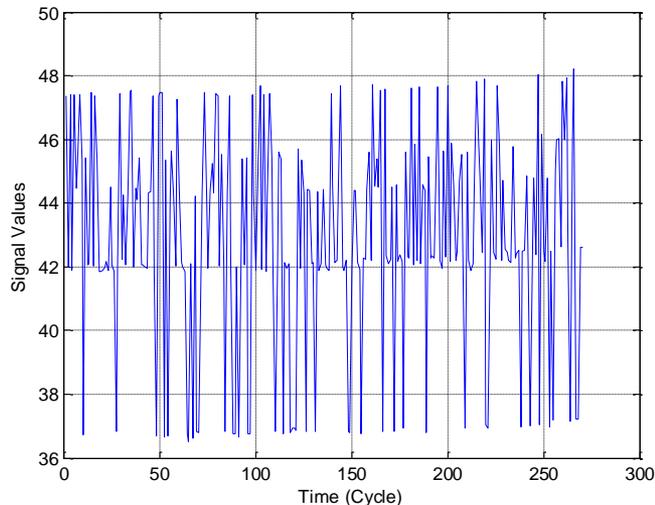


Data Preprocessing: Signal Normalization

Signal Normalization Procedure (For Each Operating Condition of each Signal)

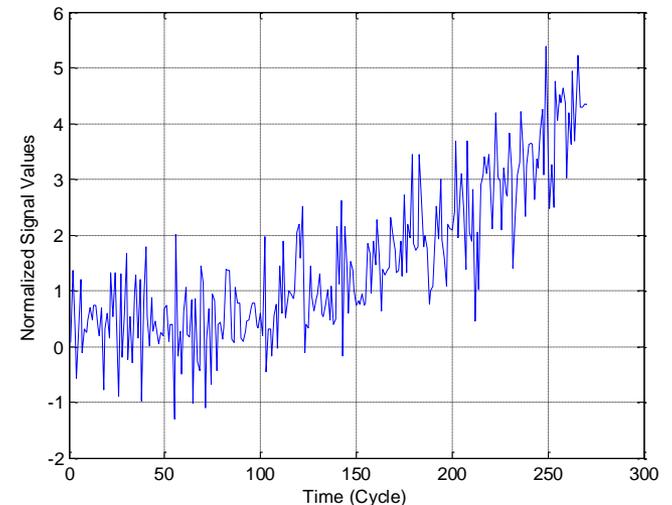
- Computation of mean μ_i and standard deviation σ_i of the i -th signal
- Normalization of the signal:
$$Sig_{norm}^i(t) = \frac{Sig^i(t) - \mu_i}{\sigma_i}$$
- The normalized signal Sig_{norm}^i is now independent from the current operating condition

Original Signal



**IDENTIFICATION
OF THE
DEGRADATION
TREND**

Normalized Signal





Feature Selection

- **PROGNOSTIC
METRICS
COMPUTATION**



Feature Selection: Prognostic Metrics (I)

Computation of the Prognostic Metrics for each considered feature and each signals

$$\text{Prognosability} = \exp\left(-\left(\frac{\sigma_{fail}}{|\mu_{fail} - \mu_{Healthy}|}\right)\right) \longrightarrow w_p = 0.8$$

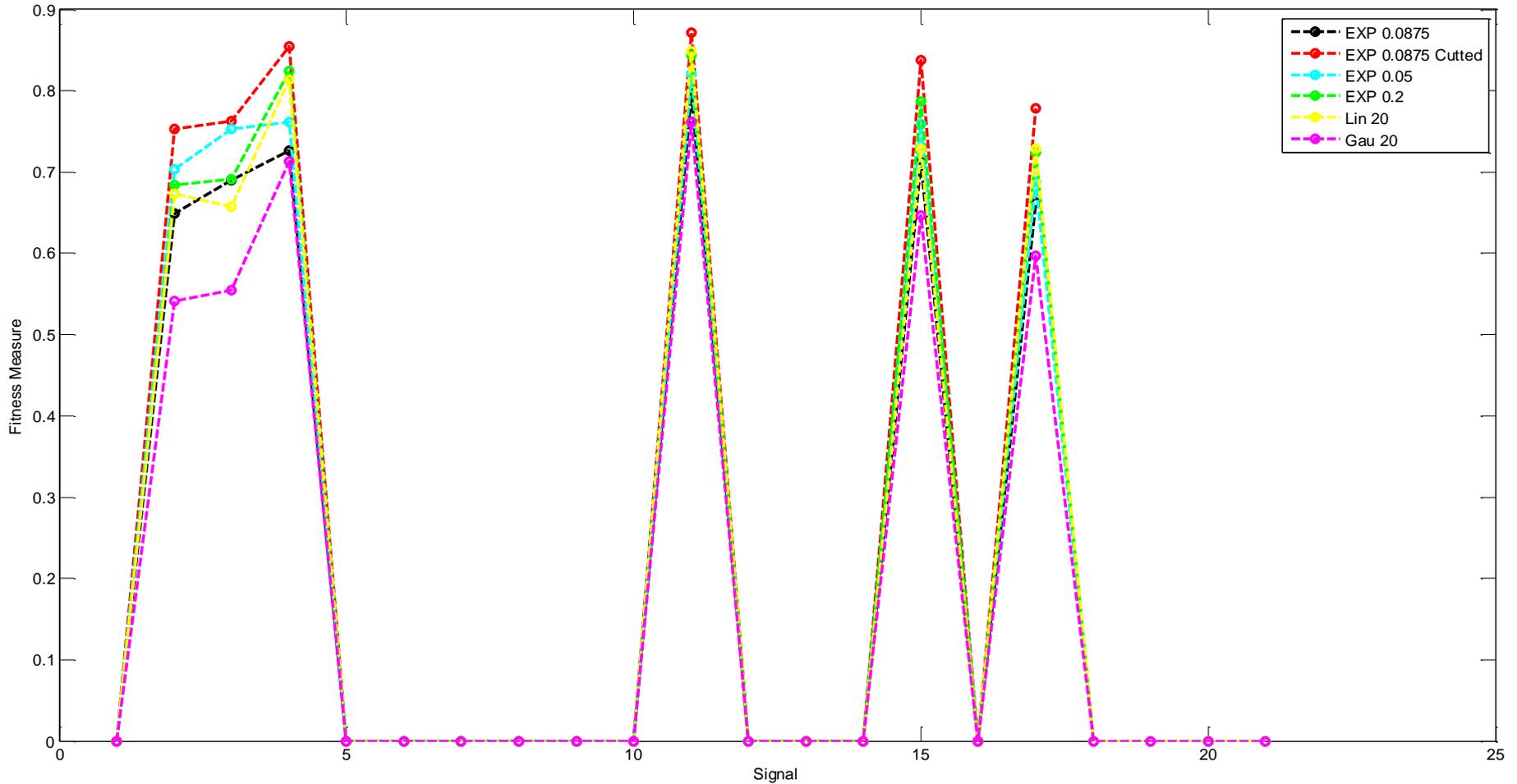
$$\text{Monotonicity} = \text{mean}\left(\left|\frac{\# \text{ pos } d/dx}{n-1} - \frac{\# \text{ neg } d/dx}{n-1}\right|\right) \longrightarrow w_m = 0.05$$

$$\text{Trendability} = \min(|\text{corrcoeff}_{ij}|) \longrightarrow w_t = 0.1$$

$$\text{Fitness_Function} = w_m \cdot \text{Monotonicity} + w_p \cdot \text{Prognosability} + w_t \cdot \text{Trendability}$$



Feature Selection: Prognostic Metrics (II)



- **Best set of signals identified: [2 3 4 11 15 17]**



RUL Prediction

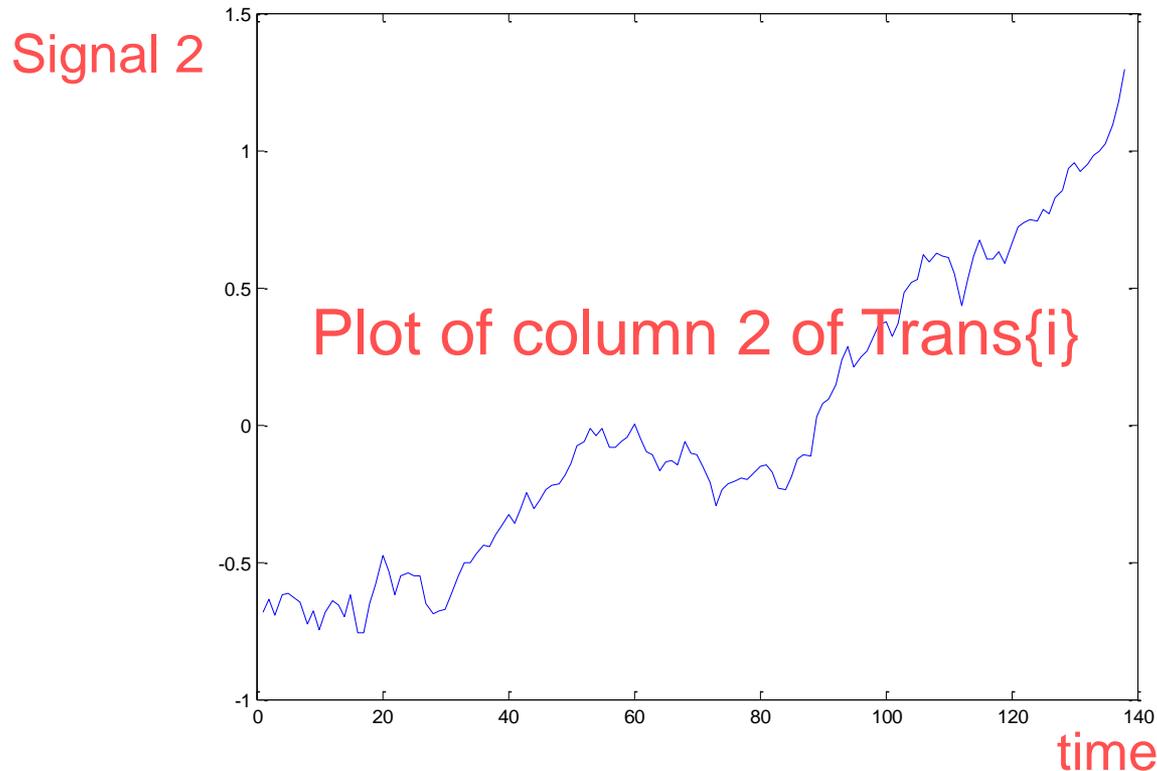
ANN



Information available

- A matlab structure:
 “trans” which contains the 260 degradation trajectories

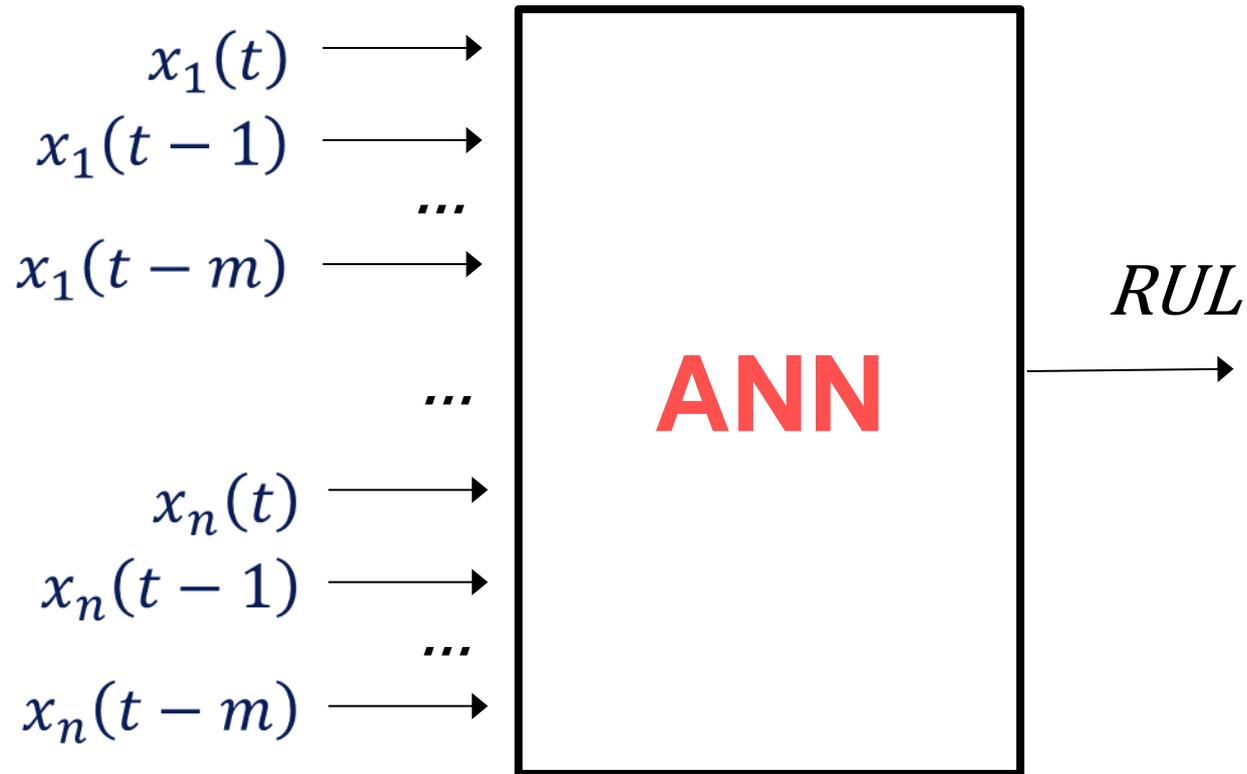
Trans{i}='matrix of size: (duration of the engine, number of signals=6)

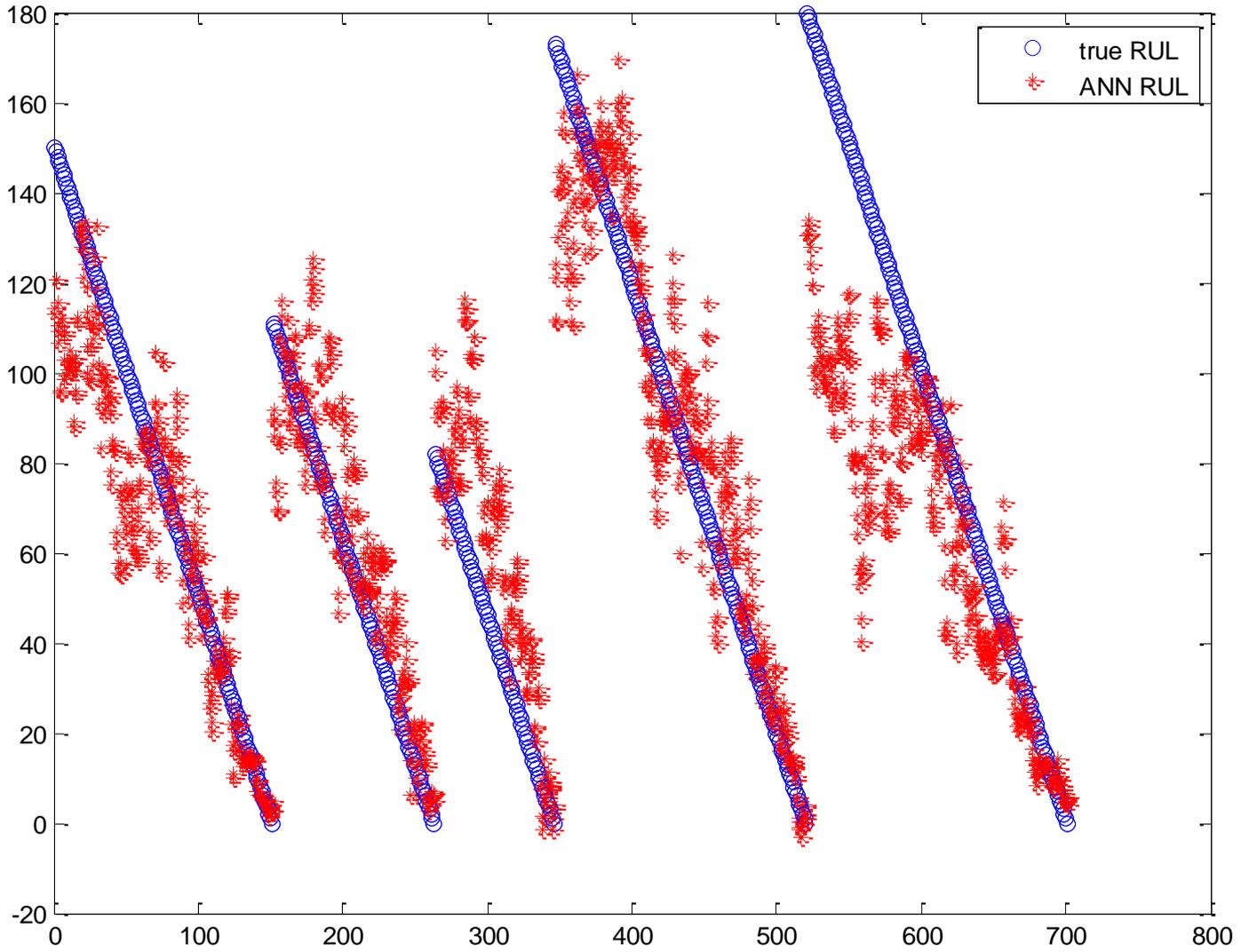




ANN Architecture

- Start considering a time window of $m=4$ and 6 nodes in the hidden layer.







Exercise: Improve Performance