ComparePolynom

September 1, 2020

1 Compare polynomial fitting with neural network regressor

1.1 In this example, a noisy cosine function is fitted by 3 polynomes of different orders, and 3 neural networks with different layers sizes.

1.1.1 Import usefull libraries.

```
[1]: import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
     %matplotlib inline
```

1.1.2 The polynomial fit is performed using the scikit modules.

```
[2]: import tensorflow
     print(tensorflow.__version__)
```

```
2.0.0
```

```
[3]: # Import stuff to perform the polynomial fit
     from sklearn.preprocessing import PolynomialFeatures
     from sklearn.linear_model import LinearRegression
     # Import tool to compute rms
     from sklearn.metrics import mean_squared_error
     # Import the TMNI. If not installed, install it
     try:
         import ai4neb
     except:
         !pip install -U git+https://github.com/morisset/AI4neb.git
         import ai4neb
```

1.1.3 Define the function we want to interpolate.

```
[4]: def true_fun(x):
         return np.cos(1.5 * np.pi * x)
```

1.1.4 Define some parameters. The X_train and y_train sets are used to determine the polynome coefficients and also to train the neural networks.

```
[5]: # A random seed to reproduce the results
     np.random.seed(0)

     # The number of points used to fit the function
     n_samples = 30

     # Noise to be added to the points used to fit the function
     noise = 0.1

     # The training set: n_samples X points, with the noisy correspoing y
     X = np.sort(np.random.rand(n_samples))
     y = true_fun(X) + np.random.randn(n_samples) * noise
     X_train = X
     y_train_true = y

     # The set of points to verify the fit quality
     X_test = np.linspace(0, 1, 100)
     y_test_true = true_fun(X_test)
```

1.1.5 A fit to the data points is done using a polynome or order 3

```
[27]: degree = 3
      polynomial_features = PolynomialFeatures(degree=degree,
                                               include_bias=False)
      model = LinearRegression()
      X_TRANSF = polynomial_features.fit_transform(X_train[:, np.newaxis])
      model.fit(X_TRANSF, y_train_true)
```

```
[27]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

1.1.6 The RMS of the fit computed on the training set used to determine the coefficients is computed.

```
[28]: y_train = model.predict(X_TRANSF)
      rms_train = np.sqrt(mean_squared_error(y_train,y_train_true))
```

1.1.7 The RMS of the fit computed on the test sample (100 points between 0 and 1) is computed.

```
[29]: y_test = model.predict(polynomial_features.fit_transform(X_test[:, np.newaxis]))
rms_test = np.sqrt(mean_squared_error(y_test,y_test_true))
```

1.1.8 A lambda function of the fit is obtained using the determined coefficients:

$$y(X) = A + B.X + C.X^2 + D.X^3$$

```
[32]: poly = lambda x: model.intercept_ + model.coef_[0] * x + model.coef_[1] * x**2
      ↪+ model.coef_[2] * x**3
```

A plot is done to show the original function, the training sample, the polynomial fit and the explicit function.

```
[33]: f, ax = plt.subplots()
ax.plot(X_test, y_test, label="Model")
ax.plot(X_test, poly(X_test), label="Model explict", ls=':')
ax.plot(X_test, y_test_true, label="True function")
ax.scatter(X, y, edgecolor='b', s=20, label="Samples")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_xlim((0, 1))
ax.set_ylim((-2, 2))
ax.legend(loc="best")
ax.set_title("Degree {}\n RMS train: {:.3f}, test: {:.3f}, test2: {:.3f}".
 ↪format(degree,
            rms_train, rms_test, rms_test));
```

Degree 3
RMS train: 0.112, test: 0.064, test2: 0.064
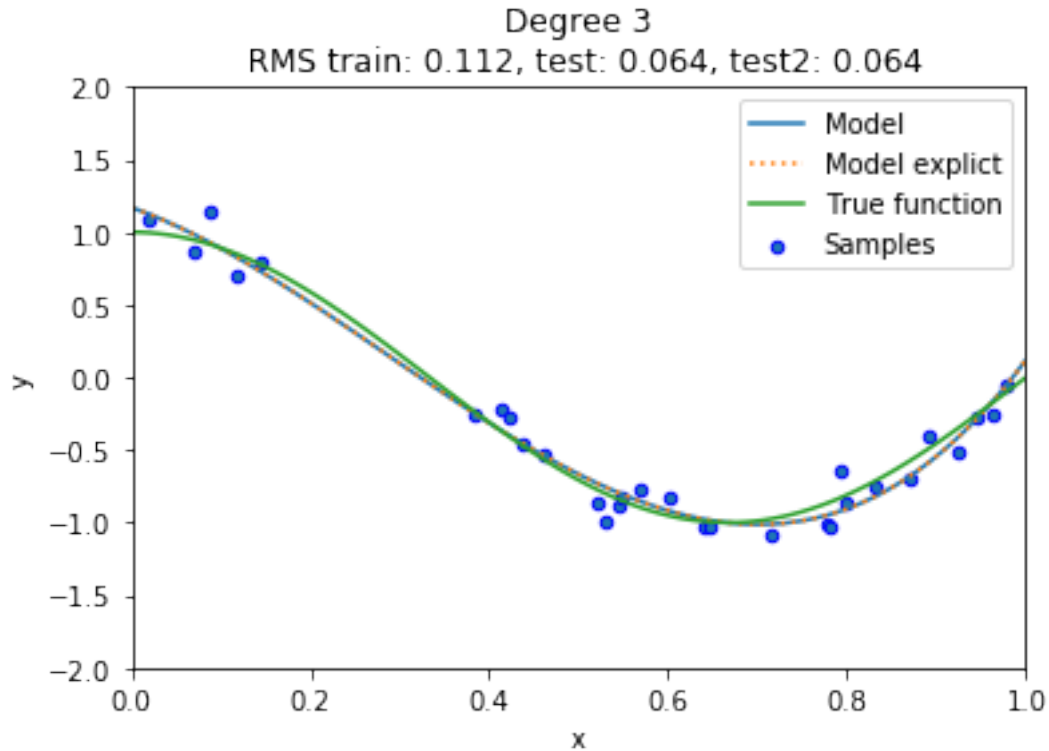
1.1.9 A Neural Network is used on the same data points.

1.1.10 It is trained on the training sets. Hyper-parameters can be changed. The network is the minimal possible: 1 layer of 2 cells. Each cell is performing an *atanh* operation.

```
[17]: RM = ai4neb.manage_RM(RM_type='SK_ANN', X_train=X_train, y_train=y_train_true,␣
       ↪scaling=False,
                       verbose=True, random_seed=10)
      RM.init_RM(hidden_layer_sizes=(2,),
                 tol=1e-7, max_iter=20000,
                 activation='tanh',
                 solver='adam')
      RM.train_RM()
```

```
Instantiation. V 0.17
Training set size = 30, Test set size = 0
Training set size = 30, Test set size = 0
Regression Model SK_ANN
WARNING: training data not scaled
Training 1 inputs for 1 outputs with 30 data
RM trained, with 10633 iterations. Score = 0.974
MLPRegressor(activation='tanh', alpha=0.0001, batch_size='auto', beta_1=0.9,
```

```
            beta_2=0.999, early_stopping=False, epsilon=1e-08,
            hidden_layer_sizes=(2,), learning_rate='constant',
            learning_rate_init=0.001, max_fun=15000, max_iter=20000,
            momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
            power_t=0.5, random_state=10, shuffle=True, solver='adam',
            tol=1e-07, validation_fraction=0.1, verbose=False,
            warm_start=False)
Training time 2.2 s.
```

1.1.11 Predictions of the ANN are performed on the training and test sets.

```
[18]: RM.set_test(X_train)
      RM.predict()
      y_train = RM.pred
      rms_train = np.sqrt(mean_squared_error(y_train, y_train_true))
      RM.set_test(X_test)
      RM.predict()
      y_test = RM.pred
      rms_test = np.sqrt(mean_squared_error(y_test, y_test_true))
```

```
WARNING: test data not scaled
Predicting from 1 inputs to 1 outputs using 30 data in 0.00 secs.
WARNING: test data not scaled
Predicting from 1 inputs to 1 outputs using 100 data in 0.00 secs.
```

1.1.12 The neuron network can be explictly computed with the weigths and the biases, a total of **7** parameters.

1.1.13 Hidden layer:

$$cel_0(X) = atanh(w_0^0.X + b_0^0) cel_1(X) = atanh(w_1^0.X + b_1^0)$$
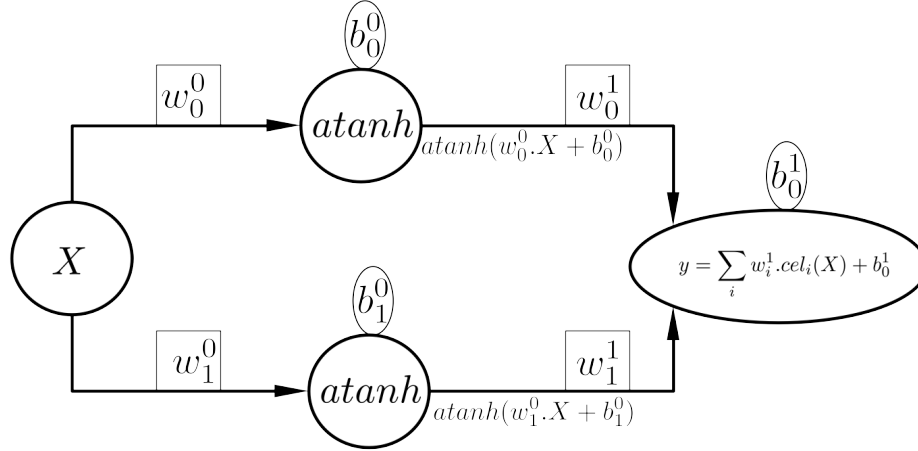
1.1.14 Output:

$$y(X) = \sum_i w_i^1.cel_i(X) + b_0^1 = w_0^1.atanh(w_0^0.X + b_0^0) + w_1^1.atanh(w_1^0.X + b_1^0) + b_0^1$$

```
[36]: from IPython.display import Image
      Image(filename = "ANN_1_2.png", width = 800)
```

[36]:

The diagram shows:
- b_0^0, w_0^0, $atanh$, $atanh(w_0^0.X + b_0^0)$, w_0^1
- X
- b_0^1, $y = \sum_i w_i^1.cel_i(X) + b_0^1$
- b_1^0, w_1^0, $atanh$, $atanh(w_1^0.X + b_1^0)$, w_1^1

1.1.15 The weights and biases of the individual neurons can be obtained.

- `coefs_` are the weigths of each synapse. It is a 3 dims list:
  `coefs_[i_layer][i_previous_neuron][i_neuron]`
- `intercepts` are the biases of each neuron: `intercept_[i_layer][i_neuron]`

```
[34]: rm = RM.RMs[0]
      print(rm.coefs_)
      print(rm.intercepts_)
```

```
[array([[-3.27986762,  2.38700192]]), array([[1.85081241],
       [2.18101657]])]
[array([ 1.35423595, -2.3252666 ]), array([1.58891653])]
```

```
[25]: activation = lambda x: np.tanh(x)
      cel_0 = lambda x: activation(x * rm.coefs_[0][0][0] + rm.intercepts_[0][0])
      cel_1 = lambda x: activation(x * rm.coefs_[0][0][1] + rm.intercepts_[0][1])
      output = lambda x: cel_0(x) * rm.coefs_[1][0][0] + cel_1(x) * rm.
      ↪coefs_[1][1][0] + rm.intercepts_[1][0]
```

1.1.16 A plot is done to show the original function, the training sample, the ANN regression and the explicit ANN.

```
[26]: f, ax = plt.subplots()
      ax.plot(X_test, y_test, label="Model")
      ax.plot(X_test, output(X_test), label="explicit Model", ls=':')
      ax.plot(X_test, y_test_true, label="True function")
```

```
ax.scatter(X, y, edgecolor='b', s=20, label="Samples")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_xlim((0, 1))
ax.set_ylim((-2, 2))
ax.legend(loc="best")
ax.set_title("ANN\n RMS train: {:.3f}, test: {:.3f}".format(
                rms_train, rms_test));
```
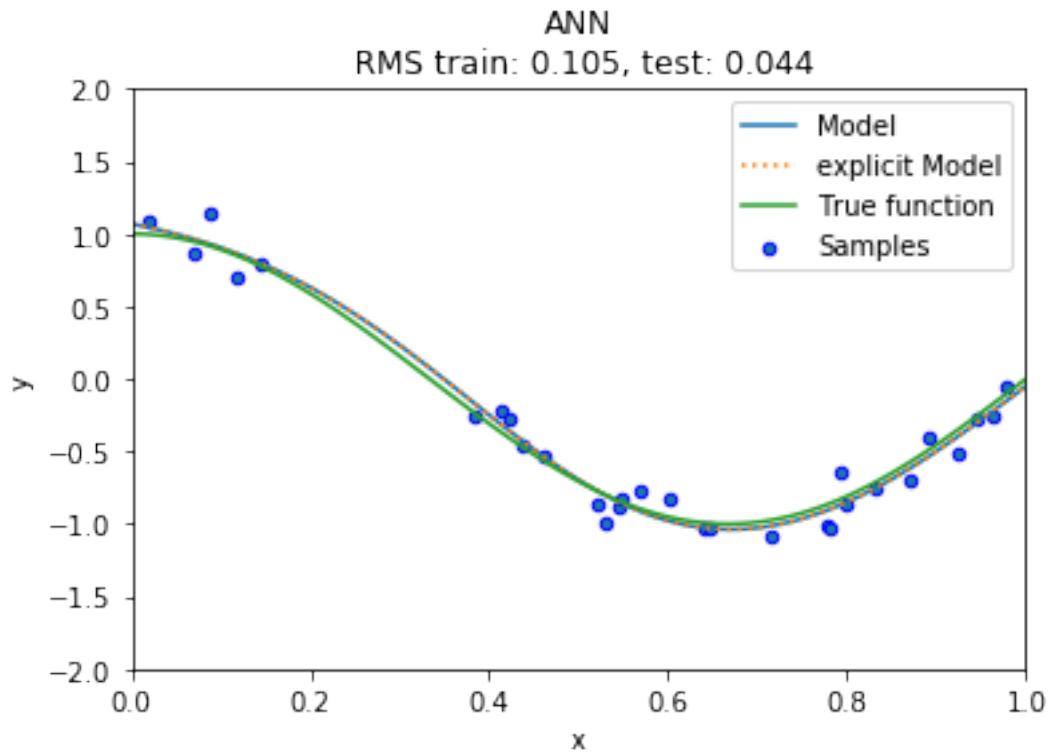

1.1.17 A comparison is made between 3 polynomial fits and 3 ANN computations.

```
[13]: f, axes = plt.subplots(2, 3, figsize=(14, 10))

degrees = [1, 4, 15]
for i in range(len(degrees)):
    ax = axes[0,i]

    polynomial_features = PolynomialFeatures(degree=degrees[i],
                                             include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline([("polynomial_features", polynomial_features),
```

```python
                            ("linear_regression", linear_regression)])
    pipeline.fit(X_train[:, np.newaxis], y_train_true)
    y_train = pipeline.predict(X_train[:, np.newaxis])
    y_test = pipeline.predict(X_test[:, np.newaxis])
    rms_train = np.sqrt(mean_squared_error(y_train,y_train_true))
    rms_test = np.sqrt(mean_squared_error(y_test,y_test_true))
    ax.plot(X_test, y_test, label="Model")
    #ax.plot(X_test, y_test_true, label="True function")
    ax.scatter(X, y, edgecolor='b', s=20, label="Samples")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_xlim((0, 1))
    ax.set_ylim((-2, 2))
    ax.legend(loc="best")
    ax.set_title("Degree {}\n RMS train: {:.3f}, test: {:.3f}".format(
 →degrees[i],
                rms_train, rms_test))

hidden_layer_sizes_set = ( (3,), (10,), (100, 100))
hidden_layer_sizes_strs = ('3', '10', '100-100')
for i in range(len(hidden_layer_sizes_set)):
    scaleit=True
    RM = ai4neb.manage_RM(RM_type='SK_ANN', X_train=X_train,
 →y_train=y_train_true, scaling=scaleit,
                    verbose=True, random_seed=10)
    RM.init_RM(hidden_layer_sizes=hidden_layer_sizes_set[i],
            tol=1e-6, max_iter=10000,
#              epochs = 10000,
            activation='relu',
            solver='adam')
    RM.train_RM()
    RM.set_test(X_train)
    RM.predict()
    y_train = RM.pred
    RM.set_test(X_test)
    RM.predict()
    y_test = RM.pred
    rms_train = np.sqrt(mean_squared_error(y_train, y_train_true))
    rms_test = np.sqrt(mean_squared_error(y_test, y_test_true))
    ax = axes[1,i]
    ax.plot(X_test, y_test, label="Model")
    #ax.plot(X_test, y_test_true, label="True function")
    ax.scatter(X, y, edgecolor='b', s=20, label="Samples")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_xlim((0, 1))
    ax.set_ylim((-2, 2))
```

```
    ax.legend(loc="best")
    ax.set_title("ANN = {}\n RMS train: {:.3f}, test: {:.3f}".
→format(hidden_layer_sizes_strs[i],
                   rms_train, rms_test))
f.tight_layout()
```

```
Instantiation. V 0.17
Training set size = 30, Test set size = 0
Train data scaled.
Test data scaled.
Training set size = 30, Test set size = 0
Training set size = 30, Test set size = 0
Regression Model SK_ANN
Training 1 inputs for 1 outputs with 30 data
RM trained, with 5405 iterations. Score = 0.959
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=(3,), learning_rate='constant',
             learning_rate_init=0.001, max_fun=15000, max_iter=10000,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
             power_t=0.5, random_state=10, shuffle=True, solver='adam',
             tol=1e-06, validation_fraction=0.1, verbose=False,
             warm_start=False)
Training time 1.3 s.
Test data scaled.
Training set size = 30, Test set size = 30
Predicting from 1 inputs to 1 outputs using 30 data in 0.00 secs.
Test data scaled.
Training set size = 30, Test set size = 100
Predicting from 1 inputs to 1 outputs using 100 data in 0.00 secs.
Instantiation. V 0.17
Training set size = 30, Test set size = 0
Train data scaled.
Test data scaled.
Training set size = 30, Test set size = 0
Training set size = 30, Test set size = 0
Regression Model SK_ANN
Training 1 inputs for 1 outputs with 30 data
RM trained, with 2518 iterations. Score = 0.974
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=(10,), learning_rate='constant',
             learning_rate_init=0.001, max_fun=15000, max_iter=10000,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
             power_t=0.5, random_state=10, shuffle=True, solver='adam',
             tol=1e-06, validation_fraction=0.1, verbose=False,
             warm_start=False)
```

```
Training time 0.6 s.
Test data scaled.
Training set size = 30, Test set size = 30
Predicting from 1 inputs to 1 outputs using 30 data in 0.00 secs.
Test data scaled.
Training set size = 30, Test set size = 100
Predicting from 1 inputs to 1 outputs using 100 data in 0.00 secs.
Instantiation. V 0.17
Training set size = 30, Test set size = 0
Train data scaled.
Test data scaled.
Training set size = 30, Test set size = 0
Training set size = 30, Test set size = 0
Regression Model SK_ANN
Training 1 inputs for 1 outputs with 30 data
RM trained, with 893 iterations. Score = 0.983
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=(100, 100), learning_rate='constant',
             learning_rate_init=0.001, max_fun=15000, max_iter=10000,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
             power_t=0.5, random_state=10, shuffle=True, solver='adam',
             tol=1e-06, validation_fraction=0.1, verbose=False,
             warm_start=False)
Training time 0.7 s.
Test data scaled.
Training set size = 30, Test set size = 30
Predicting from 1 inputs to 1 outputs using 30 data in 0.00 secs.
Test data scaled.
Training set size = 30, Test set size = 100
Predicting from 1 inputs to 1 outputs using 100 data in 0.00 secs.
```