Neural networks curve fitting

This page presents a neural network curve fitting example. This example shows and details how to create nonlinear regression with TensorFlow.

The following has been performed with the following version:

- Python 3.6.9 64 bits
- Matplotlib 3.1.1
- TensorFlow 2.1.0

Try the example online on Google Colaboratory.

Problem definition

The goal of this example is to approximate a nonlinear function given by the following equation:

$$y = 0.1.\, x.\, \cos(x) \tag{1}$$

The blue dots are the training set, the red line is the output of the network:

Source code

Each line is explained in the next section. Source code and example can be run online on Google Colaboratory

Explanation

First, we import the libraries:

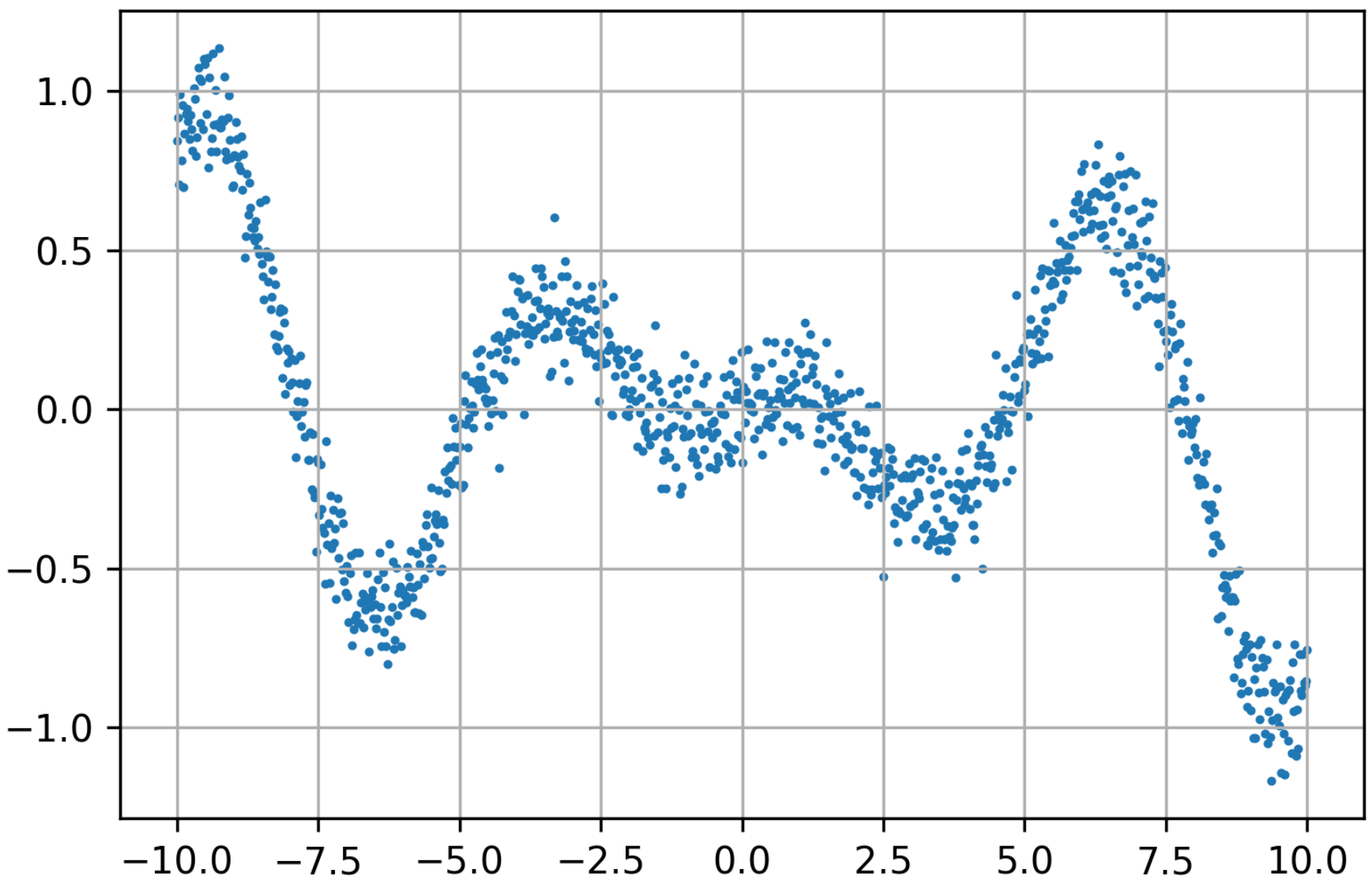
- **numpy** for arrays and matrices
- **matplotlib** for displaying charts
- **google.colab** for downloading files (only if working with Google Colaboratory)
- **Tensorflow** for neural networks
- **math** for the cosinus function

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from google.colab import files
import tensorflow as tf
import math
```

Then, we create the training data. `x_data` composed of 1000 points, and normal noise is added to the y-coordinate of each point:

```
# Create noisy data
x_data = np.linspace(-10, 10, num=1000)
y_data = 0.1*x_data*np.cos(x_data) + 0.1*np.random.normal(size=1000)
print('Data created successfully')
```

Here is the training set:



Once our training dataset is built, we can create our network:

- First layer is a single linear unit layer (for the input)
- Second layer is a 64 units RELU layer
- Third layer is a 64 units RELU layer
- Last layer is a single linear unit (for the output)

RELU is probably not the best choice for this application, but it works fine. ELU should provide smotther results.

```
# Create the model
model = keras.Sequential()
model.add(keras.layers.Dense(units = 1, activation = 'linear', input_shape=[1]))
model.add(keras.layers.Dense(units = 64, activation = 'relu'))
model.add(keras.layers.Dense(units = 64, activation = 'relu'))
model.add(keras.layers.Dense(units = 1, activation = 'linear'))
model.compile(loss='mse', optimizer="adam")

# Display the model
model.summary()
```

The model is compiled with the following optimization parameters:

- Optimization algorithm is Adam (`optimizer="adam"`), more info here.
- Loss is the regression loss based on Mean Square Error (`loss='mse'`). More information about metrics on this page.

Once the model is defined, let's train our network:

- `x_data` is the input
- `y_data` is the expected output
- `epochs=100` means our network will be trained 100 times with our dataset
- `verbose=1` display progression and loss in the console.

```
# Training
model.fit( x_data, y_data, epochs=100, verbose=1)
```

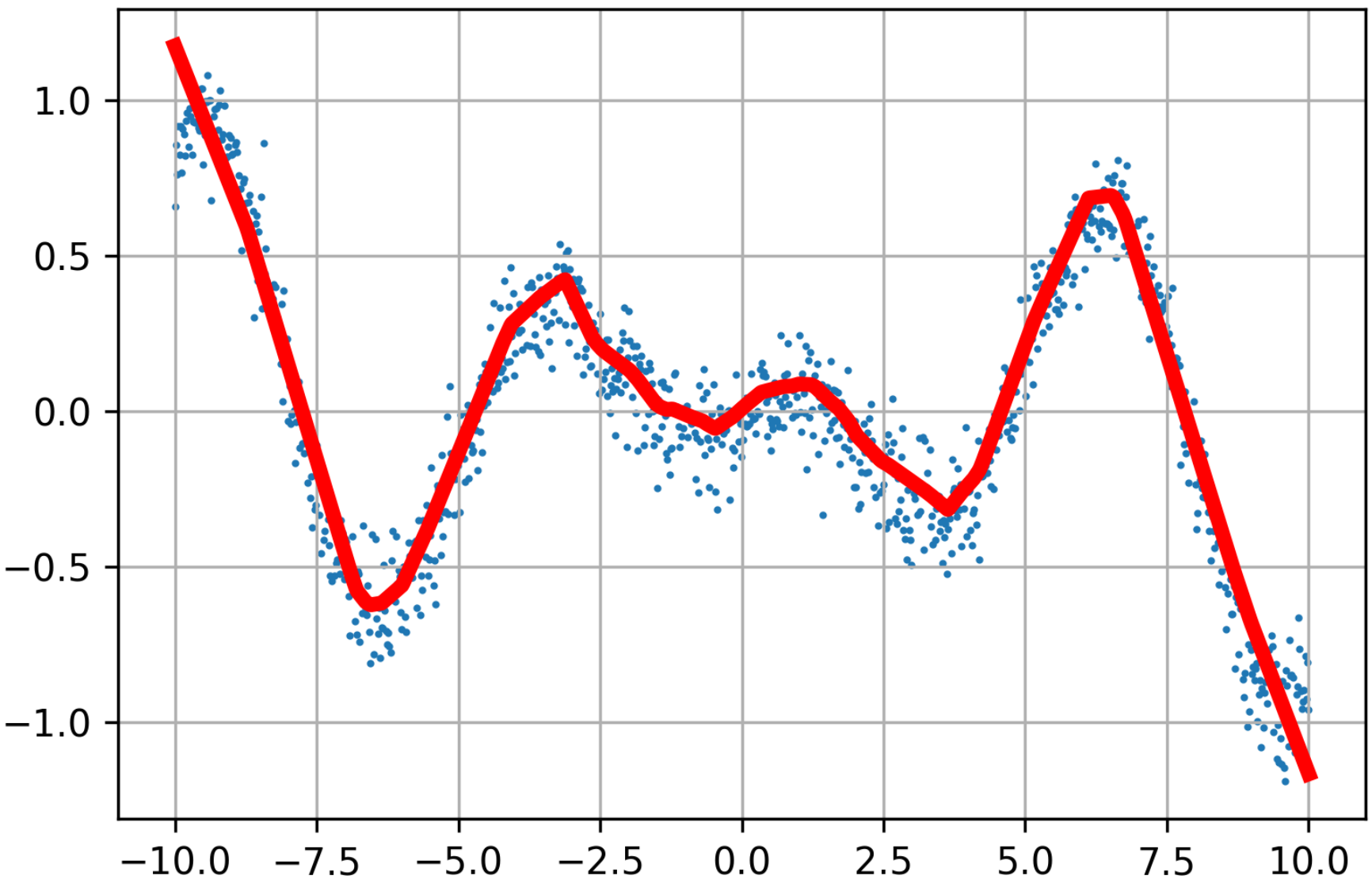
It should display something like (loss should decrease):

```
Train on 1000 samples
Epoch 1/100
1000/1000 [==============================] - 0s 321us/sample - loss: 0.2125
Epoch 2/100
1000/1000 [==============================] - 0s 49us/sample - loss: 0.1914
Epoch 3/100
1000/1000 [==============================] - 0s 50us/sample - loss: 0.1932
Epoch 4/100
1000/1000 [==============================] - 0s 60us/sample - loss: 0.1922

...

Epoch 97/100
1000/1000 [==============================] - 0s 59us/sample - loss: 0.0180
Epoch 98/100
1000/1000 [==============================] - 0s 53us/sample - loss: 0.0188
Epoch 99/100
1000/1000 [==============================] - 0s 54us/sample - loss: 0.0161
Epoch 100/100
1000/1000 [==============================] - 0s 55us/sample - loss: 0.0147
```

Once trainning is over, we can predict and display the output for each input:

```
# Compute the output
y_predicted = model.predict(x_data)

# Display the result
plt.scatter(x_data[::1], y_data[::1])
plt.plot(x_data, y_predicted, 'r', linewidth=4)
plt.grid()
plt.show()
```

Here is the result:



Source code

You can try this example online on Google Colaboratory

Previous Summary

See also

- Datasets for deep learning
- Gradient descent example
- How popular are neural networks over the years?
- Install TensorFlow and Keras for Linux
- Learning rule demonstration
- Linear regression example
- Most popular activation functions for deep learning
- Most relevant deep learning research papers
- Neural Network Perceptron
- Simplest neural network with TensorFlow
- Simplest perceptron
- Single layer training algorithm
- Single layer classification example
- Gradient descent for neural networks
- Single layer limitations

- Neural networks

Last update : 09/02/2022