Time-series prediction with recurrent neural networks and TensorFlow

Solomiia Kurchaba

8/09/2017

Plan

- TensorFlow;
- RNN from inside;
- Experiments:
  - methodology;
  - optimisation of time window;
  - optimisation of hidden state size;
  - optimisation of the training time;

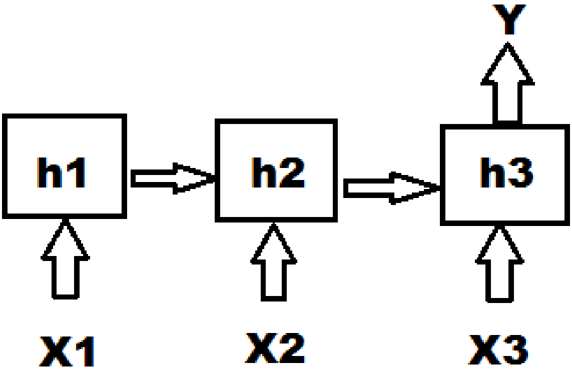
TensorFlow: What it is?

- Open-source library for machine learning using data flow graphs;
- Accessed through Python, while the actual computation is written in C++;
- Allows to define own machine learning models and train it with the data;
- Scalable across multiple computers, and multiple GPUs/CPUs within the one computer;
- Possibility of visualisation with TensorBoard;

TensorFlow. Main elements of computation graph

- Variable
  - Being modified throughout the computation process;
- Placecholder
  - Responsible for a data representation;
- Operation
  - Graph node that performs operation on tensors;

RNN - computation model that is used for sequential data modelling.



RNN data preparation. Simple example

$$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

- Time window: $n = 3$;
- Number of time-series features: $f = 1$;

Then, the data should be prepared in the following way:

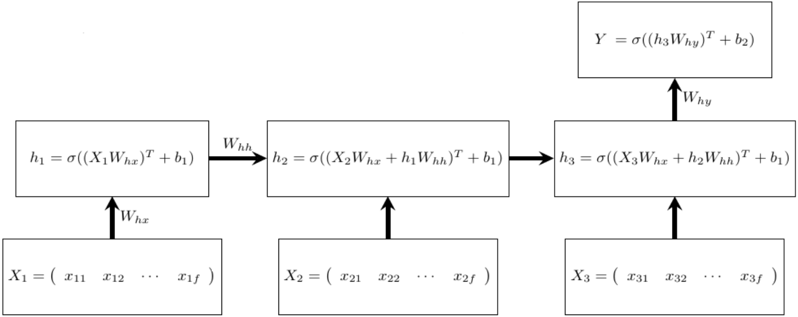
$$X^1 = \left\{ \begin{array}{l} x_1 = 1 \\ x_2 = 2 \\ x_3 = 3 \end{array} \right\} Y^1 = x_4 = 4$$

$$X^2 = \left\{ \begin{array}{l} x_2 = 2 \\ x_3 = 3 \\ x_4 = 4 \end{array} \right\} Y^2 = x_5 = 5$$

$$X^3 = \left\{ \begin{array}{l} x_3 = 3 \\ x_4 = 4 \\ x_5 = 5 \end{array} \right\} Y^3 = x_6 = 6$$

$$\vdots$$

Recurrent Neural Network. Generalisation



RNN in TensorFlow. Example

```
from tensorflow.contrib import rnn
tf.reset_default_graph()
X=tf.placeholder(tf.float32,[None,time_wind,nb_var])
Y=tf.placeholder(tf.float32,[None,output])

tf.set_random_seed(1)
x=tf.unstack(X,time_wind,1) # as we use the static RNN, the dimension of the input should be reduced.

#After unstacking we receive "time_wind" tensors of the shape (length, nb_var)

basic_cell=tf.contrib.rnn.BasicRNNCell(num_units=hidden) # here the first set of weights and biases is defined
rnn_output, states=rnn.static_rnn(basic_cell,x,dtype=tf.float32)

stacked_rnn_output=tf.reshape(rnn_output[-1],[-1,hidden])
stacked_outputs=tf.layers.dense(stacked_rnn_output,output) # the output layer. (+additional set of weights and biases)
loss=tf.reduce_mean(tf.squared_difference(stacked_outputs, Y))
optimizer=tf.train.AdamOptimizer(learning_rate=lr)
training_op=optimizer.minimize(loss)
init=tf.global_variables_initializer()
```

- *time_wind*: size of the historical time window;
- *nb_var*: number of features of the time-series;
- *output*: number of output features;
- *hidden*: number of hidden units;

Model parameters

- Size of historical time window;
- Number of hidden units (hidden state size);
- Learning rate;
- Training time;

Basic model

- Time window: 1;
- Hidden state: 1;
- Learning rate: 0.1;
- Training epochs: 1000;

Datasets

- Sinusoid;

$$y(x) = sin(2\pi x) + N(0, 0.3)$$

- Real-world multivariate dataset (Temp.csv);
  - Univaruate time-series;
  - Multivariate time-series;
- Autoregressive model AR(1), AR(2), AR(3):
  - $y_t = N(0,1) - 0.6y_{t-1}$
  - $y_t = N(0,1) - 0.6y_{t-1} + 0.4y_{t-2}$
  - $y_t = N(0.1) + 0.88y_{t-1} - 0.8y_{t-2} + 0.55y_{t-3}$

Time window

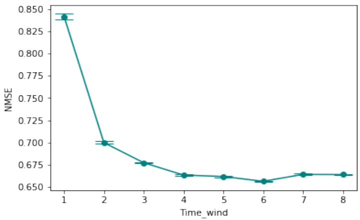


Figure: Sinusoid

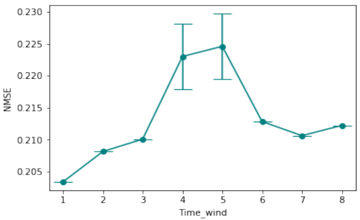


Figure: AR(1) model

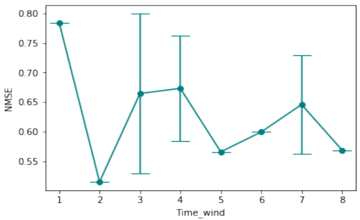


Figure: AR(2) model

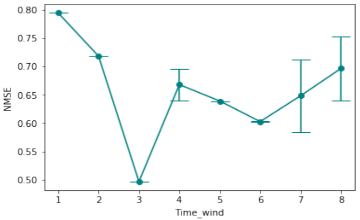


Figure: AR(3) model

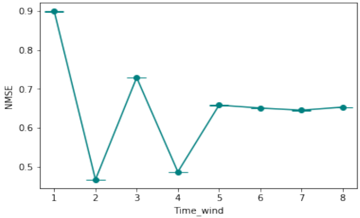


Figure: Real dataset. Univariate time-serie

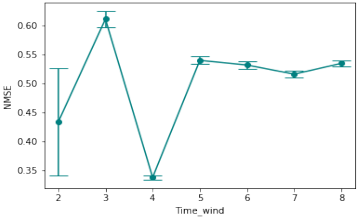


Figure: Real dataset. Multivariate prediction

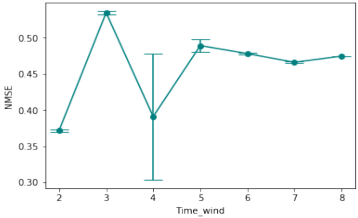


Figure: Real dataset. Multivariate time-series. Univariate prediction

Hidden state size

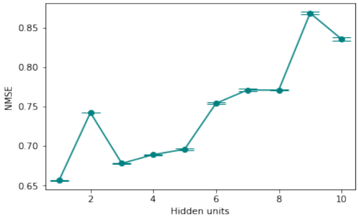


Figure: Sinusoid

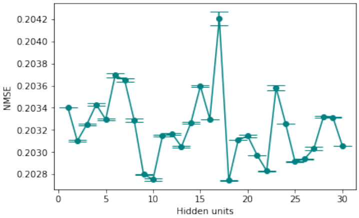


Figure: AR model

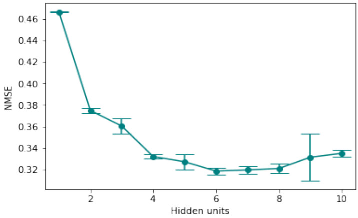


Figure: Real dataset. Univariate time-series

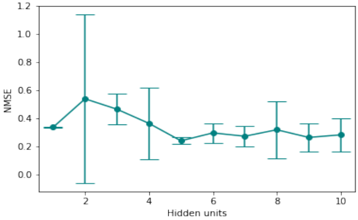


Figure: Real dataset. Multivariate prediction

Sinusoid. Training time

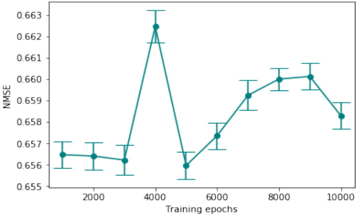


Figure: Learning rate=0.1

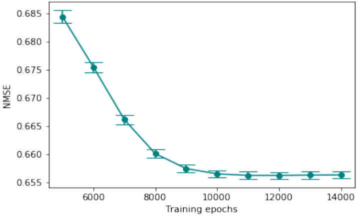


Figure: Learning rate=0.001

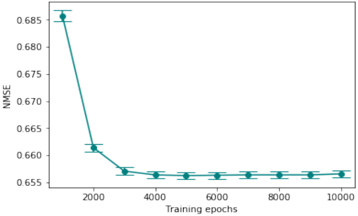


Figure: Learning rate=0.01

- Average naive model: 0.5902 ± 0.0138
- Constant naive model: 0.1588 ± 0.0295
- Best accuracy (in NMSE): 0.6562 ± 0.0630

Autoregressive model AR(1). Training time

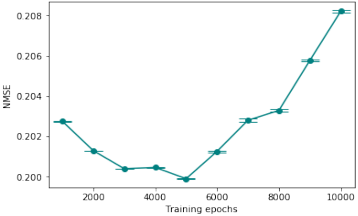


Figure: Learning rate=0.1

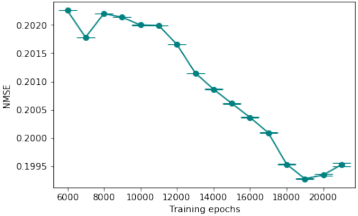


Figure: Learning rate=0.001

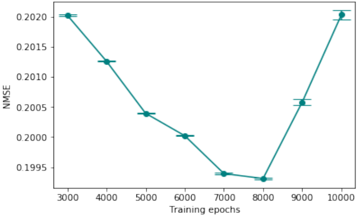


Figure: Learning rate=0.01

- Average naive model: 1.6758
- Constant naive model: 4.9564
- Best accuracy (in NMSE): $0.1993 \pm 3.7553 \cdot 10^{-6}$

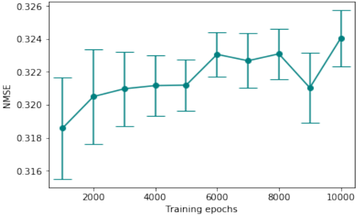


Figure: Learning rate=0.1

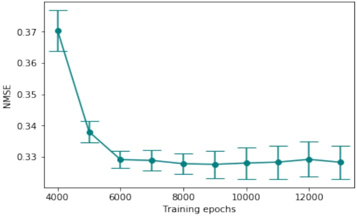


Figure: Learning rate=0.001

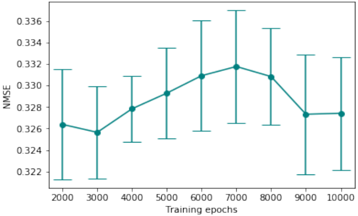


Figure: Learning rate=0.01

- Average naive model: 47.9806
- Constant naive model: 17.3820
- Best accuracy (in NMSE): 0.3186 ± 0.0133

Real data-set. Multivariate time-series. Training time

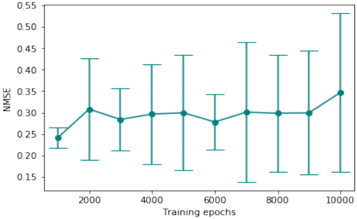


Figure: Learning rate=0.1

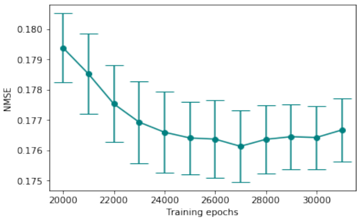


Figure: Learning rate=0.001

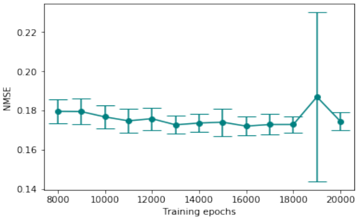


Figure: Learning rate=0.01

- Average naive model: 47.5974
- Constant naive model: 16.3648
- Best accuracy (in NMSE): 0.1721 ± 0.0049

Summary. The most successful parameter values

Parameter	Sinusoid	Univariate	Multivariate	AR
Time window	6	2	4	1
Hidden size	1	6	5	8
Learning rate	0.1	0.1	0.01	0.001
Training time	5000	1000	16000	19000

- Size of time window reflects autocorrelation characteristics of the time-series;
- The more information we provide to the system, the less hidden units we need;
- Decrement of the learning rate improves the precision of the forecast;
- Additional feature of the time-series may worsen the precision of the prediction;
- Too long training process could cause overfitting (especially with the high learning rate);

Thank you!