

Recurrent Neural Networks Tutorial (DRAFT)

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688P, Spring Semester 2026

April 30, 2026

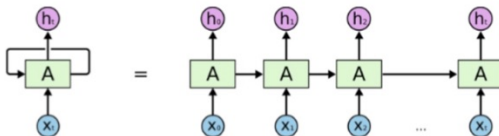
Overview

- 1 Standard Recurrent Neural Networks (RNN)
 - Learning Sequences
 - Single- and Multi-Layer RNN Architectures
- 2 Long Short-Term Memory (1997)
- 3 Gated Recurrent Units (2014)
- 4 Applications
 - Learn Time Series (Amplitude Varying Sine Wave)
 - Memorize pi to 20 decimal places (3.14159265358979323846)
 - Learn Time Series (Weather Conditions in Seattle)
 - Generate New Text (Learn Shakespeare, Write Shakespeare)
 - Learn Time Series (Identify Trends in Chart Sequences)

Quick Review

Machine Learning Capabilities (1997-2014)

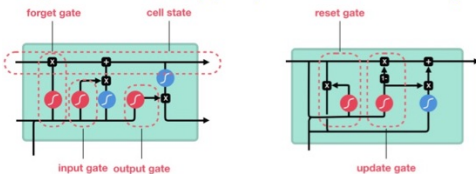
Recurrent Neural Networks (RNN): Learn sequences in data streams (text, speech)



Hidden state "h" serves two purposes:

- Make an output prediction.
- Represent features in the previous steps

Long Short-Term Memory (1997) Gated Recurrent Units (2014)



Key Features of LSTM:

- Standard RNN suffers from vanishing gradients for modeling of long-term dependencies.
- LSTM gives cells the ability to **remember values for long periods of time**.
- Gates regulate the flow of information in / out of the cell, and what should be remembered or discarded.



sigmoid



tanh



pointwise
multiplication



pointwise
addition



vector
concatenation

Applications:

- Time series prediction.
- Time-series anomaly detection.

Machine Learning Capabilities (1997-2014)

Learning Streams of Text

- Download complete works of Shakespeare (5.4 million characters)
- Train machine to remember text.
- Write new Shakespeare!

Sample after 1000 Iterations

OTHERS. Allow up, whe, Will a moutice,
Reason thot boy Skents allies; comn jusgius,
And encarembrover heth lasty!
TASTOR. When no; devires at though me beligo jody?
NANCHELOT. Trom juthar and bur itnot spokk,
That as take have wandisho

Sample after 2000 Iterations

VIBOTARD. Walk this boy and door as am 'stone!
NTRURLEN. Being entainare af Haion.
Els that by much that, I mion's now, who make foil the kidll
CLOTE. Why which Ham'd?
QUICKLY. And stand quast of I; this Fi

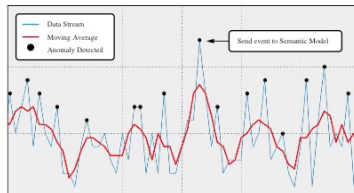
Sample after 2500 Iterations

FORD. Nay, You're, excount: and now did yet.
PAROLLES. Take DUTBY
This who is begin Gnoban a bows; but yet which that have be,
Oll, thou stan, and me not ready withered gids
And he in the pleases or pardon us.
Her. I pray you, how can, and tu

Sample after 3000 Iterations

Rume, 'tigonig, dear?
My lam, an hour, chork'd more in my grain,
I am thing forent innoan, maden! I thank I do speak you?
ARRESSTURO. But his bounes, giving to know, foward to
the distvenail.' The to you well know yes, my lowi.
SECOND CMMENOR. He needs, for the reforeds are;

Time Series Anomaly Detection



Time Series Prediction



Standard Recurrent Neural Networks

Sequence Modeling: Design Criteria

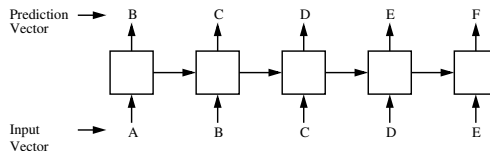
Learning Sequences

To model sequences we need to:

- Handle variable-length sequences
- Maintain information about order
- Share parameters across the sequence
- Track long-term dependencies

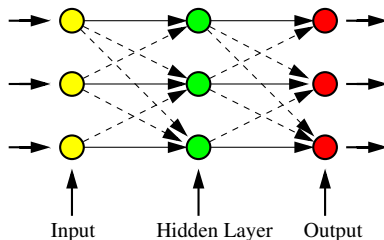
Simple Example:

The sequence $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$ can be modeled:

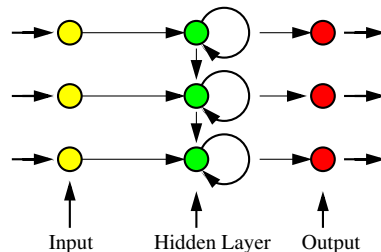


Feedforward Neural Networks vs RNNs

Feedforward Neural Network



Recurrent Neural Network

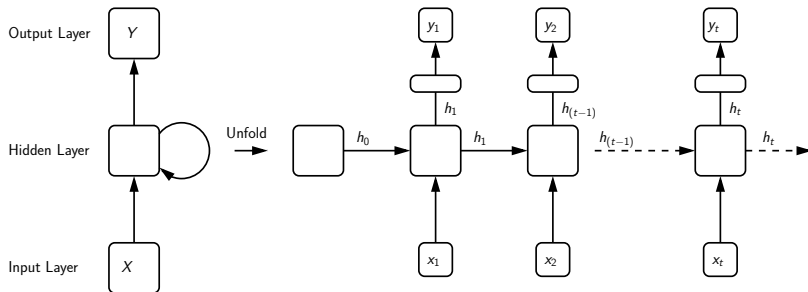


Key Points:

- Feedforward neural networks have **no memory**.
- Standard RNN's have **short-term memory**.

Standard RNN Model

RNN Model with One Hidden Layer ...



Hidden states:

- Represent features in the previous steps.
- Make an output prediction.

Standard RNN Model

Update in Activation

$$h_t = g_1(W_{aa}h_{(t-1)} + W_{ax}x_t + b_a). \quad (1)$$

Update in Output

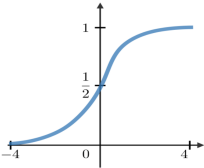
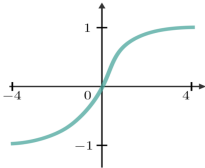
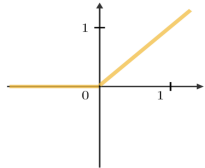
$$y_t = g_2(W_{ya}h_t + b_y) \quad (2)$$

Here:

- h_t is output from state cell at time t .
- W_{aa} , W_{ax} , W_{ya} , b_a , and b_y are coefficients that are shared temporally.
- $g_1()$ and $g_2()$ are activation functions.

Standard RNN Model

Commonly Used Activation Functions

Sigmoid	Tanh	ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

Note. The $\sigma()$ and $\tanh()$ activations squish values to always be in the $[-1, 1]$ range.

Standard RNN Model: Training Procedure

Loss Function: Loss L is the sum of loss functions evaluated at preceding time steps, i.e.,

$$L_n(\hat{y}, y) = \sum_{t=1}^n L(\hat{y}_t, y_t). \quad (3)$$

Training Procedure: Backpropagation is done at each point in time.

At timestep t , the derivative of the loss L with respect to the weight matrices W can be written:

$$\frac{\partial L}{\partial W} = \sum_{i=1}^T \frac{\partial L_i}{\partial W} \quad (4)$$

Standard RNN Model: Training Procedure

Gradient Update Rule

- Gradients are used to update the weights in a neural network,

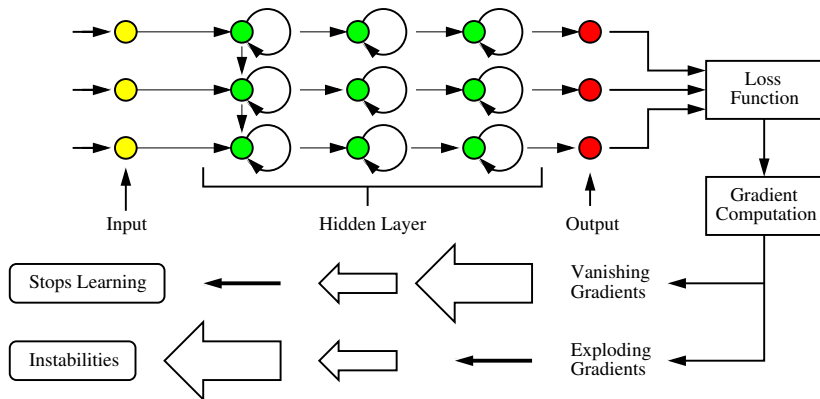
$$\text{New Weight} = \text{Weight} - \text{Learning Rate} * \text{Gradient}. \quad (5)$$

Vanishing and Exploding Gradients

- Layers that get **very small** gradient updates **stop learning**. And because this usually happens in the early layers, layers beyond that don't learn.
- Exploding gradients lead to oscillations – instability – in learning of the weights.

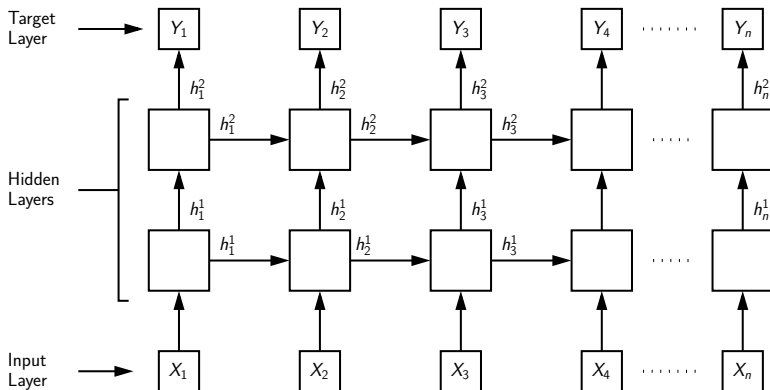
Standard RNN Model: Training Procedure

Challenge: Vanishing and Exploding Gradients ...

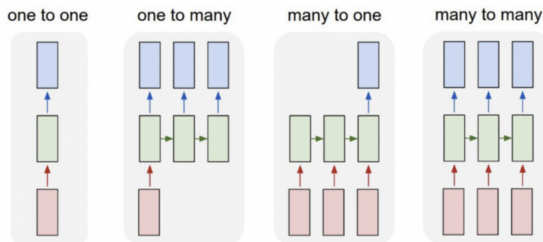


RNN Model Extesions

RNN Model with Multiple Layers



RNN Model Extensions and Applications



One-to-One: Predict future values based on historical values.

One-to-Many: Write music and text (poems, articles, code).

Many-to-One: Sentiment classification.

Many-to-Many: Name-entity recognition, machine translation, identify objects, actions and events within image/video frames.

Standard RNN Model: Summary

Advantages:

- Model can process input of any length.
- Model size does not increase with length of input.
- Model takes into account historical information.

Weaknesses:

- Computation time can be slow.
- Accessing information from a long time ago can be difficult.
- Cannot look ahead and consider future input.

Long Short-Term Memory

(Under the Hood)

Long Short-Term Memory (LSTM)

Motivation

- Standard RNNs forget what is seen in longer sequences – hence, the term **short-term memory**.

Key Features

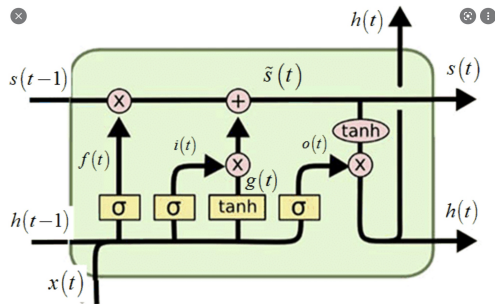
- LSTM gives cells the ability to **remember values** for **long periods of time**.

Approach

- Cell states act as a **communications highway** that can **transfer information** all the way down the **sequence chain**. This is long-term memory. Hidden states serve as short-term memory.
- Gates **regulate** the **flow of information** in/out of the cell, as well as what should be **remembered** or **discarded**.

Long Short-Term Memory (LSTM)

LSTM Architecture

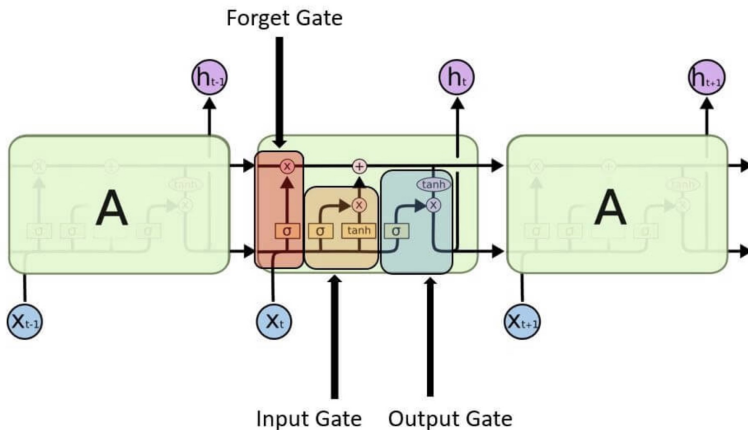


Here:

- $s(t)$ is the cell state passed forward to the next block.
- $h(t)$ is output passed forward to the next block.
- $x(t)$ is block input.

LSTM Core Concepts

LSTM Core Concepts: Cell state information. Forget, input, and output gate operations.



LSTM Core Concepts

Forget Gate Operation: $f_{(t)} \in (0, 1)$.

- Forget gates pass information from previous state and current input is to a sigmoid function,

$$f_{(t)} = \sigma(W_f \cdot [h_{(t-1)}, x_{(t)}] + b_f). \quad (6)$$

Points to note:

- Forget gates decide what information should be discarded or kept?
- Output values close to 0 mean discard; output values close to 1 mean keep.
- Here, W_f = weight matrix between forget gate and input gate. b_f = connection bias at timestep t .

LSTM Core Concepts

Input Gate Operation: $i_{(t)} \in (0, 1)$.

$$i_{(t)} = \sigma(W_i \cdot [h_{(t-1)}, x_{(t)}] + b_i). \quad (7)$$

Output Gate Operation: $o_{(t)}$.

- The output gate operation decides the value of the next hidden state.

$$o_{(t)} = \sigma(W_o \cdot [h_{(t-1)}, x_{(t)}] + b_o). \quad (8)$$

Cell Input Activation Vector: $g_{(t)} \in (-1, 1)$.

$$g_{(t)} = \tanh(W_c \cdot [h_{(t-1)}, x_{(t)}] + b_c). \quad (9)$$

LSTM Core Concepts and Equations

Cell State Vector: $s_{(t)} \in \mathbb{R}$.

$$s_{(t)} = f_{(t)} \odot s_{(t-1)} + i_{(t)} \odot g_{(t)}. \quad (10)$$

Here, $g_{(t)}$ is the cell state information produced by $\tanh()$; $s_{(t)}$ is the cell state information; $\odot$ is the Hadamard element-wise product operator.

LSTM Output: $h_{(t)} \in (-1, 1)$.

$$h_{(t)} = o_{(t)} \odot \tanh(s_{(t)}). \quad (11)$$

LSTM output is a sequence of hidden states , $h_1, h_2, \dots, h_T$.

LSTM Mathematical Equations

Sequence of Equations:

$$f_{(t)} = \sigma(W_f \cdot [h_{(t-1)}, x_{(t)}] + b_f),$$

$$i_{(t)} = \sigma(W_i \cdot [h_{(t-1)}, x_{(t)}] + b_i),$$

$$o_{(t)} = \sigma(W_o \cdot [h_{(t-1)}, x_{(t)}] + b_o),$$

$$g_{(t)} = \tanh(W_c \cdot [h_{(t-1)}, x_{(t)}] + b_c),$$

$$s_{(t)} = f_{(t)} \odot s_{(t-1)} + i_{(t)} \odot g_{(t)},$$

$$h_{(t)} = o_{(t)} \odot \tanh(s_{(t)}).$$

For More Information on Variants:

- Google: LSTM wikipedia.

Gated Recurrent Units

Gated Recurrent Units

Motivation

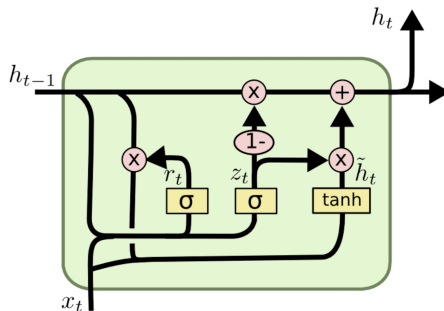
- Overcome shortcomings of standard RNN.
- Simplify implementation (c.f., LSTM).
- Improve ease of learning and performance.

Approach

- LSTM employs **forget** and **output gates** to **control information change** in the **hidden state**.
- GRU uses a **single reset gate** to achieve the same goal.

Gated Recurrent Units

GRU Architecture



Here:

- $x_{(t)}$ is the block input.
- $h_{(t)}$ is output passed forward to the next block.

Gated Recurrent Units

Reset Gate Vector: $r_{(t)}$

$$r_{(t)} = \sigma(W_r x_{(t)} + U_r h_{(t-1)} + b_r). \quad (12)$$

The reset gate $r_{(t)}$ determines how much of a hidden state to carry forward from a previous time-step. W , U and b are parameter matrices and bias vector.

Update Gate Vector: $z_{(t)}$

$$z_{(t)} = \sigma(W_z x_{(t)} + U_z h_{(t-1)} + b_z). \quad (13)$$

The update gate determines the relative strength of the contribution of the matrix-based update and a more direct contribution from the hidden vector $h_{(t-1)}$.

Gated Recurrent Units

The model can decide, for example, to **copy** all **information** from the past and **eliminate** the **vanishing gradient problem**.

Candidate Activation Vector: $\hat{h}_{(t)}$

$$\hat{h}_{(t)} = \tanh(Wx_{(t)} + U(r_{(t)} \odot h_{(t-1)})) \quad (14)$$

Here, $\odot$ is the Hadamard element-wise product operator.

Output Vector: $h_{(t)}$

$$h_{(t)} = z_{(t)} \odot h_{(t-1)} + (1 - z_{(t)}) \odot \hat{h}_{(t)}. \quad (15)$$

The output vector is passed to the next cell.

GRU Mathematical Equations

Summary: The output of GRU is a sequence of hidden states (output vectors), $h_1, h_2, \dots, h_T$, calculated by the sequence of equations:

$$r_{(t)} = \sigma(W_r x_{(t)} + U_r h_{(t-1)} + b_r)$$

$$z_{(t)} = \sigma(W_z x_{(t)} + U_z h_{(t-1)} + b_z)$$

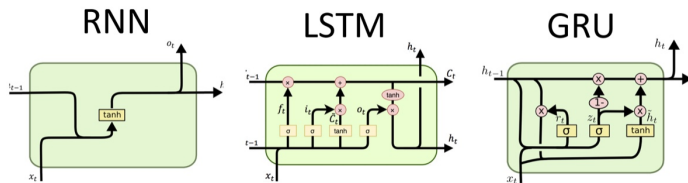
$$\hat{h}_{(t)} = \tanh(W x_{(t)} + U(r_{(t)} \odot h_{(t-1)}))$$

$$h_{(t)} = z_{(t)} \odot h_{(t-1)} + (1 - z_{(t)}) \odot \hat{h}_{(t)}.$$

We assume that the input $x_{(t)}$ is a d-dimensional vector and the hidden states are p-dimensional. The transformation matrices in the first layer are sized accordingly.

Gated Recurrent Units

Comparison of Models: Simple vs LSTM vs GRU



LSTM vs GRU

- LSTM cells employ three gates – input, forget and output. GRU uses only reset and update gates.
- LSTM has cell states (long-term memory) and hidden states (short-term memory). GRU has only one hidden state.

Example 1

Learn Time Series

(Amplitude Varying Sine Wave)

Example 1: Amplitude Varying Sine Wave

Problem Statement: Use LSTM in pytorch for sequence prediction in an amplitude varying sine wave.

Case 1: Time varying sine wave:

$$y(t) = 10te^{-t/25} \sin(t), \quad t \in [0, 100] \text{ sec.} \quad (16)$$

Max value: $y_{\max} = 91.73$.

Case 2: Time varying sine wave + uniformly distributed noise:

$$y(t) = 10te^{-t/25} \sin(t) + 20 \cdot \text{uniform}[-1, 1], \quad t \in [0, 100] \text{ sec.} \quad (17)$$

Max value possible: $y_{\max} = 111.73$.

Example 1: Amplitude Varying Sine Wave

Solution Procedure:

- Case 1 and 2: Synthetically generate sine wave data.
- Case 2 only: add uniformly distributed random noise.
- Generate training sequences from raw data.
- Convert generated sequences into pytorch tensor.
- Initialize LSTM model, loss function and optimizer.
- Train LSTM model (no epochs = 100, 1,000, 2,000).
- Plot generated data and LSTM predictions.

Example 1: Amplitude Varying Sine Wave

Generate Sine Wave Data:

```
[ 0.00e+00  9.96e-02  3.95e-01  8.78e-01  1.54e+00  2.35e+00  3.31e+00
 4.39e+00  5.57e+00  6.81e+00  8.10e+00  9.40e+00  1.07e+01  1.19e+01
 1.31e+01  1.41e+01  1.50e+01  1.58e+01  1.63e+01  1.67e+01  1.68e+01
 1.67e+01  1.63e+01  1.56e+01  1.47e+01  1.35e+01  1.20e+01  1.03e+01
 8.33e+00  6.11e+00  3.68e+00  1.05e+00 -1.74e+00 -4.66e+00 -7.69e+00
-1.08e+01 -1.39e+01 -1.70e+01 -2.01e+01 -2.31e+01 -2.59e+01 -2.86e+01
-3.10e+01 -3.33e+01 -3.52e+01 -3.68e+01 -3.81e+01 -3.90e+01 -3.95e+01
-3.96e+01 -3.92e+01 -3.85e+01 -3.72e+01 -3.56e+01 -3.35e+01 -3.10e+01
-2.81e+01 -2.48e+01 -2.11e+01 -1.72e+01 -1.29e+01 -8.43e+00 -3.72e+00
 1.13e+00  6.09e+00  1.11e+01  1.61e+01  2.11e+01  2.59e+01  3.06e+01
```

... lines of data removed ...

```
-7.61e+00 -5.50e+00 -3.34e+00 -1.17e+00  1.01e+00  3.16e+00  5.27e+00
 7.31e+00  9.27e+00  1.11e+01  1.29e+01  1.44e+01  1.59e+01  1.72e+01
 1.82e+01  1.92e+01  1.99e+01  2.04e+01  2.07e+01  2.08e+01  2.06e+01
 2.03e+01  1.98e+01  1.91e+01  1.82e+01  1.71e+01  1.58e+01  1.44e+01
 1.29e+01  1.13e+01  9.50e+00  7.66e+00  5.76e+00  3.80e+00  1.83e+00
-1.59e-01 -2.13e+00 -4.07e+00 -5.95e+00 -7.77e+00 -9.50e+00 -1.11e+01
-1.26e+01 -1.40e+01 -1.52e+01 -1.63e+01 -1.72e+01 -1.79e+01 -1.84e+01
-1.87e+01 -1.89e+01 -1.89e+01 -1.86e+01 -1.82e+01 -1.76e+01 -1.69e+01
-1.60e+01 -1.49e+01 -1.37e+01 -1.23e+01 -1.08e+01 -9.27e+00]
```

```
--- Raw data shape: (1000,) ...
```

```
--- Max sin wave data value: 91.735661 ...
```

Example 1: Amplitude Varying Sine Wave

Generate Training Sequences from Raw Data:

```
def create_sequences(data, seq_length):

    print("--- ");
    print("--- Enter create_sequences() ...");
    print("--- ----- ...");

    xs, ys = [], []
    for i in range(len(data) - seq_length):
        x = data[i:(i + seq_length)]
        y = data[i + seq_length]

        print("--- Item {:3d}: seq: {:s} --> pred: {:.3f} ".format( i, str(x), y ));

        xs.append(x)
        ys.append(y)

    print("--- ----- ...");
    print("--- Leave create_sequences() ...");
    print("--- ");

    return np.array(xs), np.array(ys)\
```

Example 1: Amplitude Varying Sine Wave

Summary of Training Sequences:

Item	Training Sequence (sequence length = 10)											Prediction			
0:	seq:	[	0.00	0.10	0.39	0.88	1.54	2.35	3.31	4.39	5.57	6.81]	-->	pred:	8.098
1:	seq:	[	0.10	0.39	0.88	1.54	2.35	3.31	4.39	5.57	6.81	8.1]	-->	pred:	9.396
2:	seq:	[	0.39	0.88	1.54	2.35	3.31	4.39	5.57	6.81	8.1	9.4]	-->	pred:	10.675
3:	seq:	[	0.88	1.54	2.35	3.31	4.39	5.57	6.81	8.1	9.4	10.68]	-->	pred:	11.907
4:	seq:	[	1.54	2.35	3.31	4.39	5.57	6.81	8.1	9.4	10.68	11.91]	-->	pred:	13.060
5:	seq:	[	2.35	3.31	4.39	5.57	6.81	8.1	9.4	10.68	11.91	13.06]	-->	pred:	14.106
6:	seq:	[	3.31	4.39	5.57	6.81	8.10	9.4	10.68	11.91	13.06	14.11]	-->	pred:	15.015
7:	seq:	[	4.39	5.57	6.81	8.1	9.40	10.68	11.91	13.06	14.11	15.02]	-->	pred:	15.761
8:	seq:	[	5.57	6.81	8.10	9.4	10.68	11.91	13.06	14.11	15.02	15.76]	-->	pred:	16.320

... lines of output removed ...

```
986: seq: [-18.4 -18.74 -18.89 -18.86 -18.64 -18.23 -17.64 -16.88 -15.95 -14.87] --> pred: -13.654
987: seq: [-18.74 -18.89 -18.86 -18.64 -18.23 -17.64 -16.88 -15.95 -14.87 -13.65] --> pred: -12.304
988: seq: [-18.89 -18.86 -18.64 -18.23 -17.64 -16.88 -15.95 -14.87 -13.65 -12.3 ] --> pred: -10.839
989: seq: [-18.86 -18.64 -18.23 -17.64 -16.88 -15.95 -14.87 -13.65 -12.3 -10.84] --> pred: -9.274
```

Example 1: Amplitude Varying Sine Wave

Convert: Generated Sequences into torch.Tensor ...

```
trainX = torch.tensor( X[:, :, None], dtype=torch.float32)
trainY = torch.tensor( y[:, None],      dtype=torch.float32)
```

Tensor Output (Abbreviated):

```
--- Train X shape: torch.Size([990, 10, 1]) ...
```

```
tensor([[[[ 0.0000], [ 0.0996], [ 0.3950], ..., [ 4.3929], [ 5.5678], [ 6.8121]],
          [[ 0.0996], [ 0.3950], [ 0.8777], ..., [ 5.5678], [ 6.8121], [ 8.0977]],
          [[ 0.3950], [ 0.8777], [ 1.5359], ..., [ 6.8121], [ 8.0977], [ 9.3955]],
          ...,
          [[-18.7396], [-18.8941], [-18.8588], ..., [-15.9533], [-14.8746], [-13.6537]],
          [[-18.8941], [-18.8588], [-18.6353], ..., [-14.8746], [-13.6537], [-12.3037]],
          [[-18.8588], [-18.6353], [-18.2270], ..., [-13.6537], [-12.3037], [-10.8388]]]])
```

```
--- Train Y shape: torch.Size([990, 1]) ...
```

```
tensor([[ 8.0977e+00],
        [ 9.3955e+00],
```

```
... lines of output removed ...
```

```
[-1.0839e+01],
[-9.2744e+00]])
```

Example 1: Amplitude Varying Sine Wave

LSTM: model ...

```
class LSTMModel( nn.Module ):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim):
        super(LSTMModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.layer_dim = layer_dim
        self.lstm       = nn.LSTM(input_dim, hidden_dim, layer_dim, batch_first=True)
        self.fc          = nn.Linear(hidden_dim, output_dim)

    def forward(self, x, h0=None, c0=None):
        if h0 is None or c0 is None:
            h0 = torch.zeros(self.layer_dim, x.size(
                0), self.hidden_dim).to(x.device)
            c0 = torch.zeros(self.layer_dim, x.size(
                0), self.hidden_dim).to(x.device)

        out, (hn, cn) = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :]) # Take last time step

        return out, hn, cn
```

Example 1: Amplitude Varying Sine Wave

Initialize LSTM model, Loss Function and Optimizer:

```
model      = LSTMModel( input_dim=1, hidden_dim=100, layer_dim=1, output_dim=1 )
criterion  = nn.MSELoss()
optimizer  = torch.optim.Adam(model.parameters(), lr=0.01)
```

LSTM Model Summary:

```
LSTMModel( (lstm): LSTM(1, 100, batch_first=True)
              (fc): Linear(in_features=100, out_features=1, bias=True) )
--- Layer: lstm.weight_ih_l0 | Size: torch.Size([400, 1]) |
      Values: tensor([[[-0.0007], [ 0.0536]], grad_fn=<SliceBackward0>)]
--- Layer: lstm.weight_hh_l0 | Size: torch.Size([400, 100]) |
      Values: tensor([[ 0.0639,  0.0074, -0.0472, ... details removed ...
                        -0.0838, -0.0541,  0.0885,  0.0915],
                        [-0.0926,  0.0705,  0.0501, ... details removed ...
                        0.0795,  0.0776, -0.0150, -0.0882, -0.0904,  0.0934,  0.0442,  0.0436,
                        -0.0865,  0.0926,  0.0947,  0.0903]], grad_fn=<SliceBackward0>)]
--- Layer: lstm.bias_ih_l0 | Size: torch.Size([400]) |
      Values: tensor([-0.0364,  0.0426], grad_fn=<SliceBackward0>)]
--- Layer: lstm.bias_hh_l0 | Size: torch.Size([400]) |
      Values: tensor([ 0.0471, -0.0464], grad_fn=<SliceBackward0>)]
--- Layer: fc.weight | Size: torch.Size([1, 100]) |
      Values: tensor([[ 0.0759, -0.0749, -0.0771, ... details removed ...
                        0.0269, -0.0913, -0.0373,  0.0040]], grad_fn=<SliceBackward0>)]
--- Layer: fc.bias | Size: torch.Size([1]) | Values: tensor([0.0149], grad_fn=<SliceBackward0>)]
```

Example 1: Amplitude Varying Sine Wave

Training: Train LSTM model (no epochs = 100) ...

```
num_epochs = 100      # <--- number of epochs for training ...
h0, c0 = None, None

for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()

    outputs, h0, c0 = model(trainX, h0, c0)

    loss = criterion(outputs, trainY)
    loss.backward()
    optimizer.step()

    h0, c0 = h0.detach(), c0.detach()

    if (epoch + 1) % 100 == 0:
        print(f'--- Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

Example 1: Amplitude Varying Sine Wave

Summary of Training: no epochs = 100 (under trained) and 2,000.

Case 1: No epochs = 100

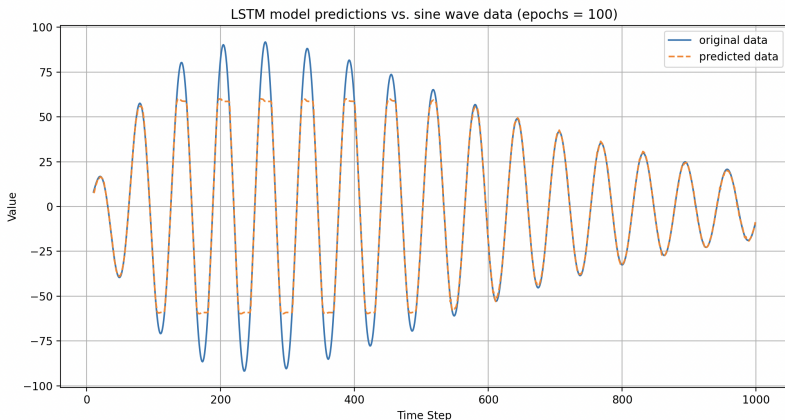
```
-----
Epoch [ 10/100], Loss: 1556.1985
Epoch [ 20/100], Loss: 1126.5245
Epoch [ 30/100], Loss:  767.4795
Epoch [ 40/100], Loss:  520.1786
Epoch [ 50/100], Loss:  353.1136
Epoch [ 60/100], Loss:  246.9728
Epoch [ 70/100], Loss:  178.8142
Epoch [ 80/100], Loss:  129.4018
Epoch [ 90/100], Loss:   94.5380
Epoch [100/100], Loss:   72.6519
-----
```

Case 1: No epochs = 2,000

```
-----
Epoch [ 100/2000], Loss: 72.6519
Epoch [ 200/2000], Loss:  7.8072
Epoch [ 300/2000], Loss:  1.7722
Epoch [ 400/2000], Loss:  0.5927
Epoch [ 500/2000], Loss:  0.2796
Epoch [ 600/2000], Loss:  0.1396
Epoch [ 700/2000], Loss:  0.0810
Epoch [ 800/2000], Loss:  0.0525
Epoch [ 900/2000], Loss:  0.0423
Epoch [1000/2000], Loss:  0.0345
Epoch [1100/2000], Loss:  0.0260
Epoch [1200/2000], Loss:  0.0106
Epoch [1300/2000], Loss:  0.0155
Epoch [1400/2000], Loss:  0.0211
Epoch [1500/2000], Loss:  0.0062
Epoch [1600/2000], Loss:  0.0042
Epoch [1700/2000], Loss:  0.0161
Epoch [1800/2000], Loss:  0.0055
Epoch [1900/2000], Loss:  0.0018
Epoch [2000/2000], Loss:  0.0138
-----
```

Example 1: Amplitude Varying Sine Wave

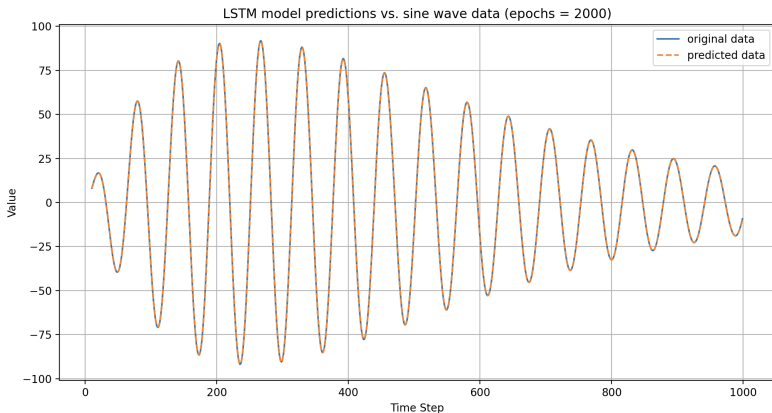
Case 1: Synthetic sine wave (epochs = 100).



Source Code: [python-code-ai.d/rnn/TestRNN-Synthetic-Sine-Wave01.py](#)

Example 1: Amplitude Varying Sine Wave

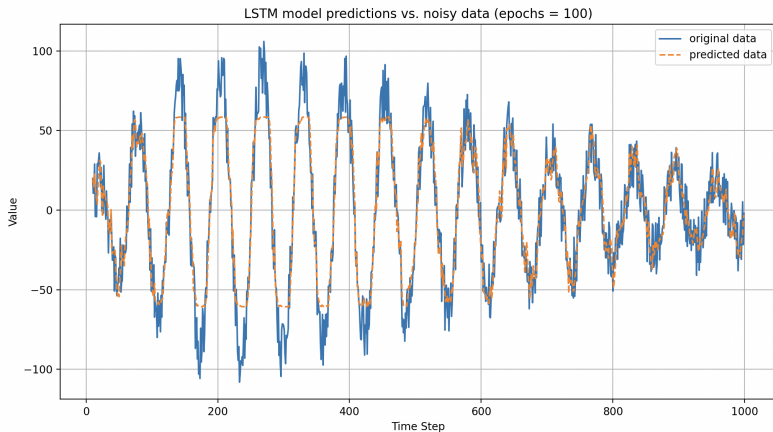
Case 1: Synthetic sine wave (epochs = 2,000).



Source Code: python-code-ai.d/rnn/TestRNN-Synthetic-Sine-Wave02.py

Example 1: Amplitude Varying Sine Wave

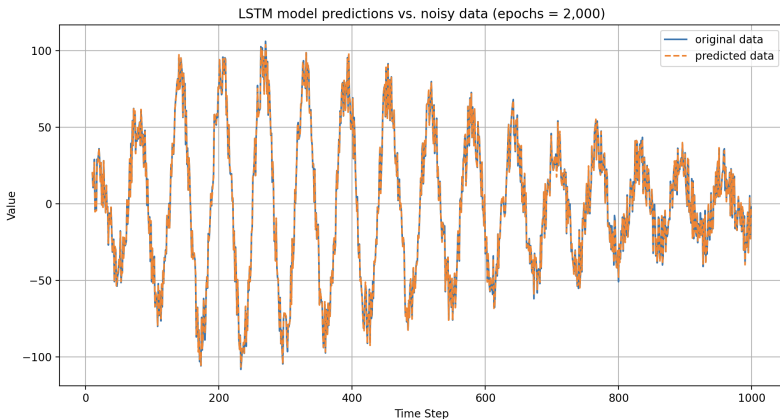
Case 2: Synthetic sine wave + noise (epochs = 100)



Source Code: [python-code-ai.d/rnn/TestRNN-Synthetic-Sine-Wave03.py](https://github.com/Python-Code-AI/d/rnn/TestRNN-Synthetic-Sine-Wave03.py)

Example 1: Amplitude Varying Sine Wave

Case 2: Synthetic sine wave + noise (epochs = 2,000).



Source Code: [python-code-ai.d/rnn/TestRNN-Synthetic-Sine-Wave04.py](https://github.com/python-code-ai.d/rnn/TestRNN-Synthetic-Sine-Wave04.py)

Example 1: Amplitude Varying Sine Wave

Computational Experiments:

Case 1: Sine wave alone (max data value = 91.73)

Extreme Value Predictions:

```
-----
Epochs =   100  --->  max predicted value = 60.01  <--- network is under-trained.
Epochs = 1,000  --->  max predicted value = 90.43
Epochs = 2,000  --->  max predicted value = 91.52  <--- best estimate of extreme
                                                           values.
```

Case 2: Sine wave + noise (max possible data value ≈ 111.73)

Extreme Value Predictions:

```
-----
Epochs =   100  --->  max predicted value = 58.48  <--- network is under-trained.
Epochs = 1,000  --->  max predicted value = 100.83
Epochs = 2,000  --->  max predicted value = 103.80  <--- best estimate of extreme
                                                           values.
```

Example 2

Memorize π to 20 (and 100) decimal places

Example 2: Memorize pi to 20 decimal places

Problem Statement

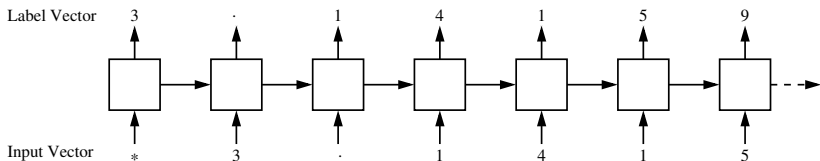
- Memorize first 20 digits of π (3.14159265358979323846).

Alphabet (11 characters: 9 digits + * + .)

[*, 3, ., 1, 4, 5, 9, 2, 6, 8, 7]

Sequence and High-level Architecture (23 data points)

* 3 . 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4 6



Example 2: Memorize pi to 20 decimal places

Annotated Encoding for Input Vector

```
[[ [ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0], <--- *
   [ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ... 0, 0], <--- 3
   [ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0], <--- .
   [ 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0], <--- 1
   [ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 1, 0], <--- 4
   [ 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, ... 0, 0], <--- 5
   [ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ... 0, 0], <--- 9
   [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ... 0, 0], <--- 2
   [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ... 0, 1], <--- 6
   [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0], <--- 8
   [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0] ]] <--- 9
```

Size: (length = 253 (11×23))

Note: Input vector uses * character to initialize the input.

Example 2: Memorize pi to 20 decimal places

Annotated Encoding for Label/Output Vector

```
[[ [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 1 ], <--- *
   [ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ... 0, 0 ], <--- 3
   [ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0 ], <--- .
   [ 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0 ], <--- 1
   [ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0 ], <--- 4
   [ 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, ... 0, 0 ], <--- 5
   [ 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ... 0, 0 ], <--- 9
   [ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ... 0, 0 ], <--- 2
   [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ... 1, 0 ], <--- 6
   [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, ... 0, 0 ], <--- 8
```

Size: (length = 253 (11 × 23))

Note: Label vector uses * character to terminate sequence.

Example 2: Memorize pi to 20 decimal places

DL4J: Initialize Test Sentence and Alphabet

```

1      // Create test sequence ...
2
3      private static String testSentence = "*3.14159265358979323846";
4
5      // Convert character string to char array ...
6
7      private static final char[] LEARNSTRING = testSentence.toCharArray();
8
9      // A list of all possible characters
10
11     private static final List<Character> LEARNSTRING_CHARS_LIST = new ArrayList<>();
12
13     // Create a dedicated list of possible chars in LEARNSTRING_CHARS_LIST
14
15     LinkedHashSet<Character> LEARNSTRING_CHARS = new LinkedHashSet<>();
16     for (char c : LEARNSTRING)
17         LEARNSTRING_CHARS.add(c);
18
19     // Populate list of possible characters ...
20
21     LEARNSTRING_CHARS_LIST.addAll(LEARNSTRING_CHARS);

```

Example 2: Memorize pi to 20 decimal places

DL4J: Initialize Input Data and Label Data Arrays

```
1  INDArray input  = Nd4j.zeros(1, LEARNSTRING_CHARS_LIST.size(), LEARNSTRING.length);
2  INDArray labels = Nd4j.zeros(1, LEARNSTRING_CHARS_LIST.size(), LEARNSTRING.length);
```

DL4J: Populate Input and Label Data Arrays

```
1  int samplePos = 0;
2  for (char currentChar : LEARNSTRING) {
3      char nextChar = LEARNSTRING[(samplePos + 1) % (LEARNSTRING.length)];
4
5      // input neuron for current-char is 1 at "samplePos"
6
7      input.putScalar(
8          new int[] { 0, LEARNSTRING_CHARS_LIST.indexOf(currentChar), samplePos }, 1);
9
10     // output neuron for next-char is 1 at "samplePos"
11
12     labels.putScalar(
13         new int[] { 0, LEARNSTRING_CHARS_LIST.indexOf(nextChar), samplePos }, 1);
14
15     samplePos++;
16 }
```

Example 2: Memorize pi to 20 decimal places

DL4J: RNN Architecture Assembly

```

1      // Set parameters in neural network configuration ...
2
3      NeuralNetConfiguration.Builder builder = new NeuralNetConfiguration.Builder();
4
5      builder.seed(123);
6      builder.biasInit(0);
7      builder.miniBatch(false);
8      builder.updater(new RmsProp(0.001));
9      builder.weightInit(WeightInit.XAVIER);
10
11     // Assemble network layers ...
12
13     ListBuilder listBuilder = builder.list();
14
15     listBuilder.layer(0, new LSTM.Builder().nIn( LEARNSTRING_CHARS.size() )
16         .nOut(HIDDEN_LAYER_WIDTH)
17         .activation(Activation.TANH).build() );
18
19     listBuilder.layer(1, new LSTM.Builder().nIn(HIDDEN_LAYER_WIDTH)
20         .nOut(HIDDEN_LAYER_WIDTH)
21         .activation(Activation.TANH).build() );

```

Example 2: Memorize pi to 20 decimal places

DL4J: RNN Architecture Assembly (continued)

```
1
2    // Create RNN output layer ...
3
4    RnnOutputLayer.Builder outputLayerBuilder = new RnnOutputLayer.Builder(LossFunction.M
5    outputLayerBuilder.activation(Activation.SOFTMAX);
6    outputLayerBuilder.nIn(HIDDEN_LAYER_WIDTH);
7    outputLayerBuilder.nOut(LEARNSTRING_CHARS.size());
8
9    listBuilder.layer(HIDDEN_LAYER_CONT, outputLayerBuilder.build());
10
11   // Create multilayer network configuration ...
12
13   MultiLayerConfiguration conf = listBuilder.build();
14   MultiLayerNetwork net = new MultiLayerNetwork(conf);
15
16   net.init();
17   net.setListeners(new ScoreIterationListener(1));
```

Example 2: Memorize pi to 20 decimal places

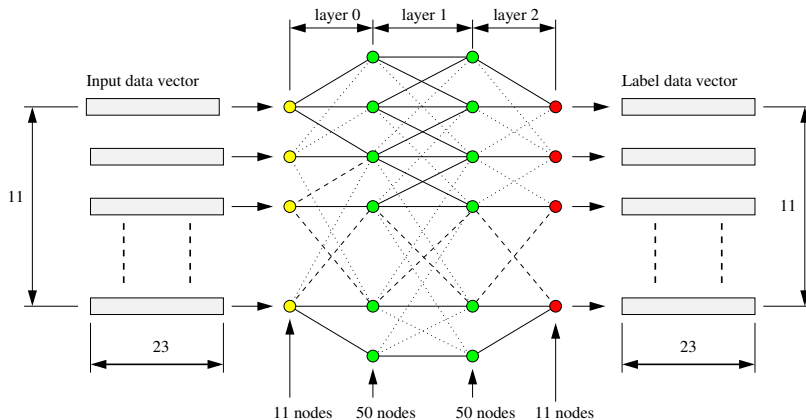
DL4J: RNN Architecture Summary

LayerName (Type)	nIn,nOut	NoParams	ParamsShape
layer0 (LSTM)	11,50	12,400	W:{11,200}, RW:{50,200}, b:{1,200}
layer1 (LSTM)	50,50	20,200	W:{50,200}, RW:{50,200}, b:{1,200}
layer2 (RnnOutput)	50,11	561	W:{50,11}, b:{1,11}

Total Parameters:	33,161	Frozen Parameters: 0	
Trainable Parameters:	33,161		

Example 2: Memorize pi to 20 decimal places

RNN Architecture + Input data vector + Label data vector



Example 2: Memorize pi to 20 decimal places

DL4J: RNN Training Procedure (1,000 epochs)

```

1      for (int epoch = 0; epoch < 1001; epoch++) {
2          // train the data
3
4          net.fit(trainingData);
5
6          // clear current stance from the last example
7
8          net.rnnClearPreviousState();
9
10         // put the first character into the rnn as an initialisation
11
12         INDArray testInit = Nd4j.zeros(1, LEARNSTRING_CHARS_LIST.size(), 1);
13         testInit.putScalar(LEARNSTRING_CHARS_LIST.indexOf(LEARNSTRING[0]), 1);
14
15         // run one step -> IMPORTANT: rnnTimeStep() must be called, not
16         // output()
17         // the output shows what the net thinks what should come next
18
19         INDArray output = net.rnnTimeStep(testInit);

```

Example 2: Memorize pi to 20 decimal places

DL4J: RNN Training Procedure (continued)

```

1      // Now the net should guess LEARNSTRING.length more characters
2
3      for (char ignored : LEARNSTRING) {
4
5          // First process the last output of the network to a concrete
6          // neuron, the neuron with the highest output has the highest
7          // chance to get chosen
8
9          int sampledCharacterIdx = Nd4j.getExecutioner().exec(new IMax(output, 1)).get(0);
10
11         // Use the last output as input
12
13         INDArray nextInput = Nd4j.zeros(1, LEARNSTRING_CHARS_LIST.size(), 1);
14         nextInput.putScalar(sampledCharacterIdx, 1);
15         output = net.rnnTimeStep(nextInput);
16     }
17 }

```

Example 2: Memorize pi to 20 decimal places

DL4J: Iterations of Learning (1,000 epochs)

```
--- Epoch    0: 9994455599333333333333333333
--- Epoch   50: 1111555555999333386****
--- Epoch  100: 11115255558999333846***
--- Epoch  150: 11115265558979333846***
--- Epoch  200: .11159265358979323846**
--- Epoch  250: ..1115265358979323846**
--- Epoch  300: ..14152265358979323846*
--- Epoch  350: ..14159263589799323846*
--- Epoch  400: ..1415265358979323846**
--- Epoch  450: 3.14159265358979323846*
```

... output removed ...

```
--- Epoch 1000: 3.14159265358979323846*
```

Example 2: Memorize pi to 20 decimal places

DL4J: Retrieve Outcomes from RNN Model (Loop over epochs)

```

1      for (int epoch = 0; epoch < 1001; epoch++) {
2
3          if(epoch == 0 || epoch%50 == 0 ) System.out.printf("--- Epoch %4d: ", epoch);
4
5          // train the data, then clear stance from last example ...
6
7          net.fit(trainingData);
8          net.rnnClearPreviousState();
9
10         // put the first character into the rnn as an initialisation
11
12         INDArray testInit = Nd4j.zeros(1, LEARNSTRING_CHARS_LIST.size(), 1);
13         testInit.putScalar(LEARNSTRING_CHARS_LIST.indexOf(LEARNSTRING[0]), 1);
14
15         // run one step -> IMPORTANT: rnnTimeStep() must be called, not
16         // output()
17         // the output shows what the net thinks what should come next
18
19         INDArray output = net.rnnTimeStep(testInit);
20
21         // Now the net should guess LEARNSTRING.length more characters

```

Example 2: Memorize pi to 20 decimal places

DL4J: Retrieve Outcomes from RNN Model (continued)

```

23     for (char ignored : LEARNSTRING) {
24
25         // First process the last output of the network to a concrete
26         // neuron, the neuron with the highest output has the highest
27         // chance to get chosen
28
29         int sampledCharacterIdx = Nd4j.getExecutioner().exec(new IMax(output, 1)).get(0);
30
31         // Print the chosen output
32
33         if(epoch == 0 || epoch%50 == 0 )
34             System.out.print( LEARNSTRING_CHARS_LIST.get(sampledCharacterIdx) );
35
36         // Use the last output as input
37
38         INDArray nextInput = Nd4j.zeros(1, LEARNSTRING_CHARS_LIST.size(), 1);
39         nextInput.putScalar(sampledCharacterIdx, 1);
40         output = net.rnnTimeStep(nextInput);
41     }
42
43     if(epoch == 0 || epoch%50 == 0 )
44         System.out.print("\n");
45 }

```

Example 2: Memorize pi to 100 decimal places

Extend Model to 100 digits of pi

- The first 20 digits of π does not contain the digit zero.
- Hence, we only need an 11 character alphabet to memorize the sequence.
- The first 100 digits of π contains eight zeros:

*3.14159265358979323846

26433832795028841971

69399375105820974944

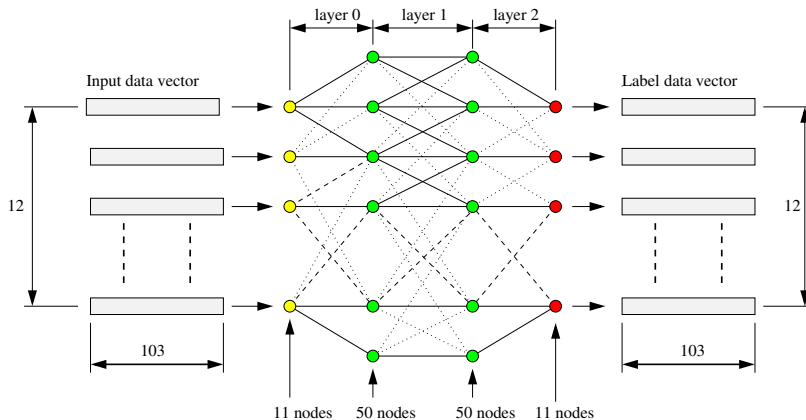
59230781640628620899

86280348253421170679

- The alphabet size is 12 and input data vector has dimension: 12×103 .

Example 2: Memorize pi to 100 decimal places

RNN Architecture + Input data vector + Label data vector



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Example 3

Learning a Time Series

(Weather Conditions in Seattle)

Example 3: Weather in Seattle

Problem Statement

- Assemble LSTM RNN model for time-series analysis of weather conditions in Seattle.
- Forecast time-variation of weather conditions (i.e., min/max temperture, rainfall, general conditions).

Step-by-Step Procedure

- Import and clean the dataset.
- Preprocess (scale) and transform the dataset.
- Identify trends and seasonality.
- Uncover relationships (correlations) between variables.
- Train the machine learning model.

Example 3a: Max/min temperature in Seattle

Weather Datafile: 1,461 data items ...

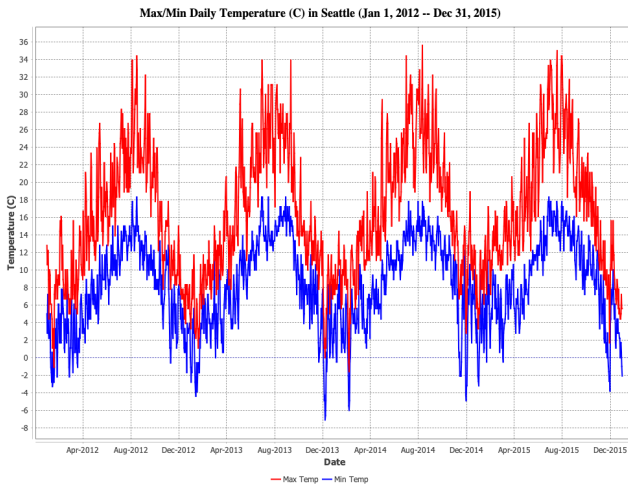
	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain
...	...	...	...	...	...	...
1456	2015-12-27	8.6	4.4	1.7	2.9	rain
1457	2015-12-28	1.5	5.0	1.7	1.3	rain
1458	2015-12-29	0.0	7.2	0.6	2.6	fog
1459	2015-12-30	0.0	5.6	-1.0	3.4	sun
1460	2015-12-31	0.0	5.6	-2.1	3.5	sun

[1461 rows x 6 columns]

Source: python-code-ai.d/data/cities/seattle/seattle-weather.csv

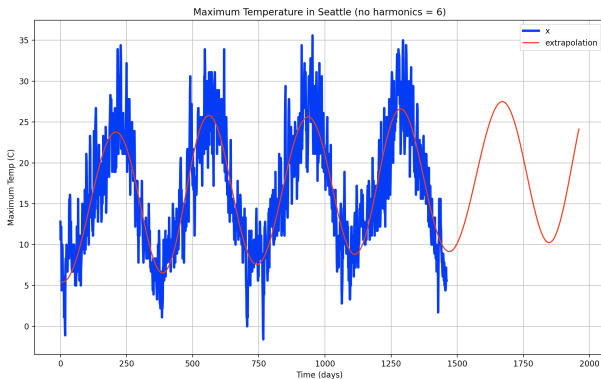
Example 3a: Max/min temperature in Seattle

Time-History: Max/min temperature in Seattle



Example 3a: Max/min temperature in Seattle

Preliminary Analysis: Extrapolated Fourier series with no harmonics = 6.



Source Code: See python-code.d/math/

Example 3a: Max/min temperature in Seattle

Fourier Analysis:

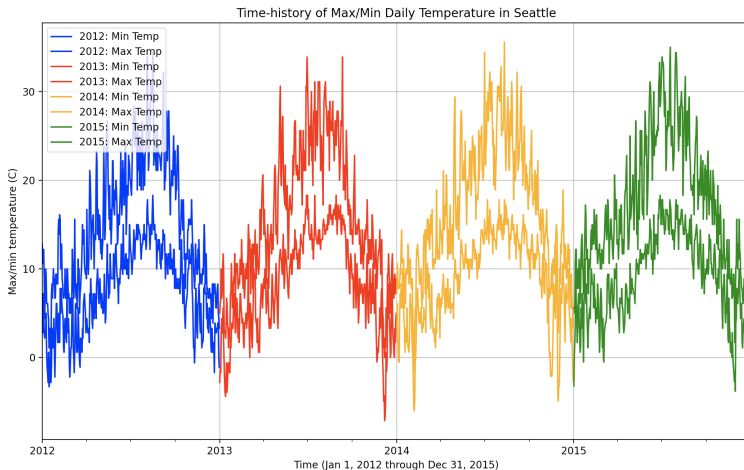
- Provides a way for **general functions** to be represented as the **sum** of simpler **trigonometric** (or exponential) **functions**.
- Google: [fourier analysis examples](#).

Preliminary Analysis: Points to note:

- It seems that while **extrapolated fourier series** do a **great job** of capturing the general periodic nature of max/min temperature measurements, **predictions** of **extreme values** may be **poor**.
- We need to understand: by using a combination of weather measurements, can RNN outperform fourier series in the prediction of extreme values?

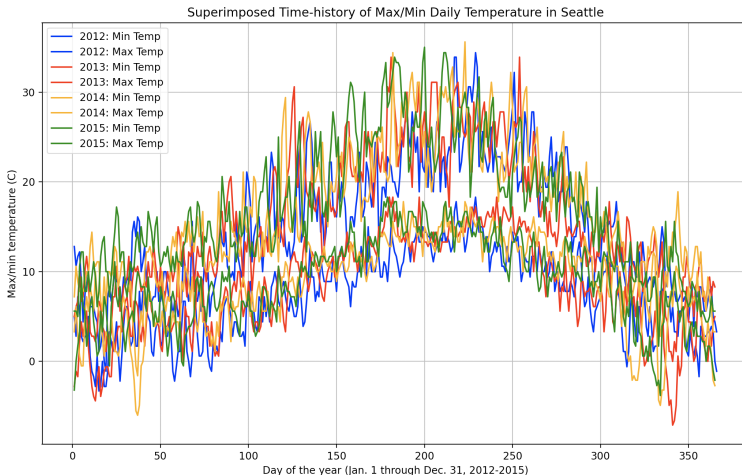
Example 3a: Weather in Seattle

Organize Data: Partition data into year-long batches:



Example 3a: Weather in Seattle

Organize Data: Map time to day-of-the-year.



Example 3a: Weather in Seattle

DL4J: Data Schema.

Schema():

idx	name	type	meta data
0	"Date"	String	StringMetaData(name="Date",)
1	"Prec . .n"	Double	DoubleMetaData(name="Precipitation", allowNaN=false,allowInfinite=false)
2	"TempMax"	Double	DoubleMetaData(name="TempMax", allowNaN=false,allowInfinite=false)
3	"TempMin"	Double	DoubleMetaData(name="TempMin", allowNaN=false,allowInfinite=false)
4	"Wind"	Double	DoubleMetaData(name="Wind", allowNaN=false,allowInfinite=false)
5	"Weather"	Categorical	CategoricalMetaData(name="Weather", stateNames=["fog", "rain", "drizzle", "sun", "snow"])

Example 3a: Max/min temperature in Seattle

DL4J: Data Transformation Process ...

```
TransformProcess(initialSchema=Schema():
```

```
  idx   name      type      meta data
```

```
    0   "Date"    String    StringMetaData(name="Date",)
```

```
    ...
```

```
    actionList=[ DataAction(
```

```
                  RemoveColumnsTransform([Precipitation, Wind, Weather]))
```

```
    ])
```

DL4J: Transformed Data Schema

```
Schema():
```

```
  idx   name      type      meta data
```

```
    0   "Date"    String    StringMetaData(name="Date",)
```

```
    1   "TempMax" Double    DoubleMetaData(name="TempMax", ... )
```

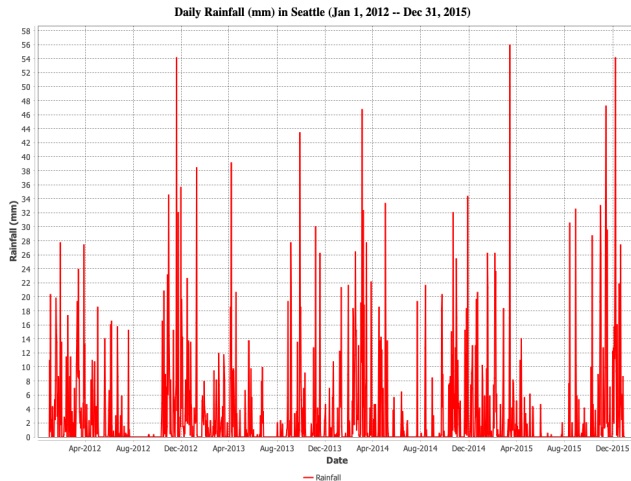
```
    2   "TempMin" Double    DoubleMetaData(name="TempMin", ... )
```

Example 3a: Max/min temperature in Seattle

DL4J: Training procedure ...

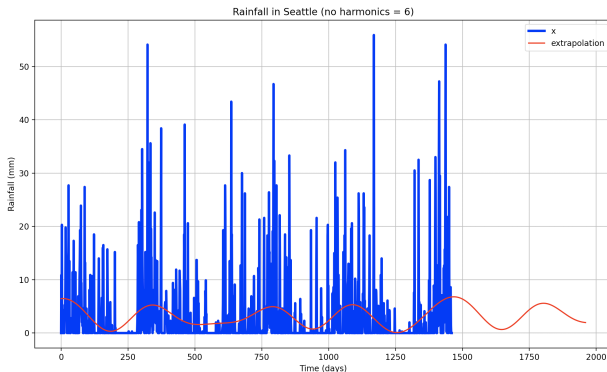
Example 3b: Daily Rainfall in Seattle

Time-History: Rainfall in Seattle



Example 3b: Daily Rainfall in Seattle

Preliminary Analysis: Extrapolated Fourier series with no harmonics = 6.



Source Code: See python-code.d/math/

Example 3b: Daily Rainfall in Seattle

DL4J: Data Transformation Process ...

```
TransformProcess(initialSchema=Schema():
```

idx	name	type	meta data
0	"Date"	String	StringMetaData(name="Date",)
1	"Precipitation"	Double	DoubleMetaData(name="Precipitation", ...)
2	"TempMax"	Double	DoubleMetaData(name="TempMax", ...)
....			
5	"Weather"	Categorical	CategoricalMetaData(name="Weather", ...),

```

actionList=[
    DataAction(RemoveColumnsTransform([TempMax, TempMin, Wind, Weather]))
])

```

DL4J: Transformed Data Schema

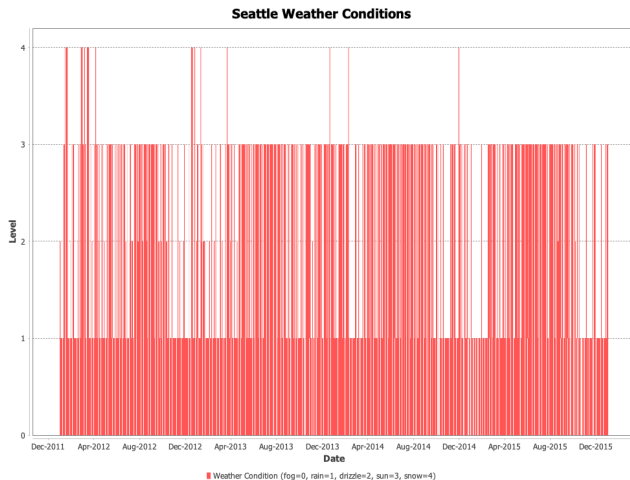
```
Schema():
```

idx	name	type	meta data
0	"Date"	String	StringMetaData(name="Date",)
1	"Precipitation"	Double	DoubleMetaData(name="Precipitation", ...)

Example 3b: Daily Rainfall in Seattle

Example 3c: Weather conditions in Seattle

Weather Conditions: Rain, fog, drizzle, clear and sunny.



Example 3c: Weather conditions in Seattle

DL4J: Data Transformation Process ...

```
TransformProcess(initialSchema=Schema():
```

```
  idx  name      type      meta data
    0  "Date"    String    StringMetaData(name="Date",)
    ....
    5  "Weather" Categorical CategoricalMetaData(name="Weather",
                                stateNames=["rain","snow","drizzle","sun","fog"])
```

```
  actionList = [
    DataAction( RemoveColumnsTransform([Precipitation, TempMax, TempMin, Wind]))
    DataAction( CategoricalToIntegerTransform(columnName=Weather, columnIdx=1,
        stateNames=[snow, rain, drizzle, fog, sun],
        statesMap={fog=0, rain=1, drizzle=2, sun=3, snow=4})) ]) ]
```

DL4J: Transformed Data Schema

```
  idx  name      type      meta data
    0  "Date"    String    StringMetaData( name="Date",)
    1  "Weather" Integer  IntegerMetaData( name="Weather",
                                minAllowed=0,maxAllowed=4)
```

Example 4

Generating New Text

(Learn Shakespeare, Write Shakespeare)

Example 4: Learn Shakespeare, Write Shakespeare

Problem Statement

- **Train machine** to **remember** complete works of Shakespeare, then use to model **write new Shakespeare!**

Sample of Shakespeare

```

onh with here yet, thou sleeper'd thy tingress!
  Lold afoness Standous most cry is avain firsts.
PORIO, SERVONT, BELANTIER Roiso folbathing better.
WILLIAMA. Would weres this faith; mert too.
I mean his lady's known TIMON and ASEDATN stones.
ANTONIOUS Core, I had,,
Com. Now thou art thou h

```

Example 4: Learn Shakespeare, Write Shakespeare

Solution Procedure

- Define problem parameters.
- Download complete works of Shakespeare (5.4 million characters).
- Setup single DatasetIterator.
- ...
- ...

Example 4: Learn Shakespeare, Write Shakespeare

DL4J: Set Problem Parameters

```
int lstmLayerSize = 200; // Number of units in each LSTM layer.
int miniBatchSize = 32; // Size of mini batch during training.
int exampleLength = 1000; // Length of training example sequence.

int tbpttLength = 50; // Length for truncated backpropagation
                        through time.

int numEpochs = 1; // Total number of training epochs.

int generateSamplesEveryNMinibatches = 10; // How frequently to generate
                                           samples from the network?

int nSamplesToGenerate = 4; // No samples to generate after
                             each training epoch.

int nCharactersToSample = 300; // Length of each sample to generate.

String generationInitialization = null; // Optional character
                                         initialization.
```

Example 4: Learn Shakespeare, Write Shakespeare

DL4J: Downloading the Data

Example 4: Learn Shakespeare, Write Shakespeare

DL4J: Organize Data into Batches

Example 4: Learn Shakespeare, Write Shakespeare

DL4J: Assemble Neural Network Architecture

```
1      MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
2          .seed(12345)
3          .l2(0.0001)
4          .weightInit(WeightInit.XAVIER)
5          .updater(new Adam(0.005))
6          .list()
7          .layer( new LSTM.Builder().nIn(iter.inputColumns()).nOut(lstmLayerSize)
8                  .activation(Activation.TANH).build()
9                  )
10         .layer( new LSTM.Builder().nIn(lstmLayerSize).nOut(lstmLayerSize)
11                 .activation(Activation.TANH).build()
12                 )
13
14         // MCXENT + softmax for classification
15
16         .layer( new RnnOutputLayer.Builder(LossFunction.MCXENT)
17                 .activation(Activation.SOFTMAX)
18                 .nIn(lstmLayerSize).nOut(nOut).build())
19         .backpropType(BackpropType.TruncatedBPTT)
20         .tbPTTForwardLength(tbpttLength)
21         .tbPTTBackwardLength(tbpttLength).build();
22
23     MultiLayerNetwork net = new MultiLayerNetwork(conf);
24     net.init();
25     net.setListeners(new ScoreIterationListener(1));
```

Example 4: Learn Shakespeare, Write Shakespeare

DL4J: Neural Network Architecture

LayerName (Type)	nIn,nOut	TotalParams	ParamsShape
layer0 (LSTM)	77,200	222,400	W:{77,800}, RW:{200,800}, b:{1,800}
layer1 (LSTM)	200,200	320,800	W:{200,800}, RW:{200,800}, b:{1,800}
layer2 (RnnOutput)	200,77	15,477	W:{200,77}, b:{1,77}

Total Parameters: 558,677

Trainable Parameters: 558,677

Frozen Parameters: 0

Example 4: Learn Shakespeare, Write Shakespeare

DL4J: Training the Model

Example 4: Learn Shakespeare, Write Shakespeare

New Shakespeare after completion of 10 minibatches

--- Sample: 0 ---

lshe wasle;

FEGPMEON.SI. Ande ouncelor homith ans, theo nomi. we gome ming.

CEIKMIAE. . ard satung oingese blomess sole:

' Thontlly hat aom, oud et sigetere blerl'erdyanolith'sor.

HSCGETERC. Pud ger fopee shilche'd coree.

So hat wove gurgeninno, angemid treaot souclдор thiyy iotbense;

--- Sample: 1 ---

Fos Oow in wose adnasqyee anserintee mavith, anc'n',

dith, notherinoth, I ullelesd, Thou coRdy's,

Cerenne resilg,

Gleurt and, sor,

Hond. Lhtoun, she ohid bowteruncicistutun. V0crene,, Hiffall.

Thivells, hil wither?

JAROD. been,

Example 4: Learn Shakespeare, Write Shakespeare

New Shakespeare after completion of 170 minibatches

--- Sample: 0 ---

our reward is hope thy giad, be.

Whithister's been, I, mind. Jien, Penmorace may far'et,
Thou art not begit a' tears. 'That. Peacish, there-bitch!

SIR TOBY. Where and thou that by amun. so you!

That is you, kneel, for no furst,

End be use essess the news.

YORK. No, he is give think to

--- Sample: 1 ---

onh with here yet, thou sleeever'd thy tingress!

Lold afoness Standous most cry is avain firsts.

PORIO, SERVONT, BELANTIER Roiso folbathing better.

WILLIAMA. Would weres this faith; mert too.

I mean his lady's known TIMON and ASEDATN stones.

ANTONIOUS Core, I had,,

Com. Now thou art thou h

Example 5

Learn Time Series

(Identify trends in Control Chart Sequence Data)

Example 5: Control Chart Sequence Data

Problem Statement

- This example learns how to **classify univariate time series** as belonging to one of six categories:

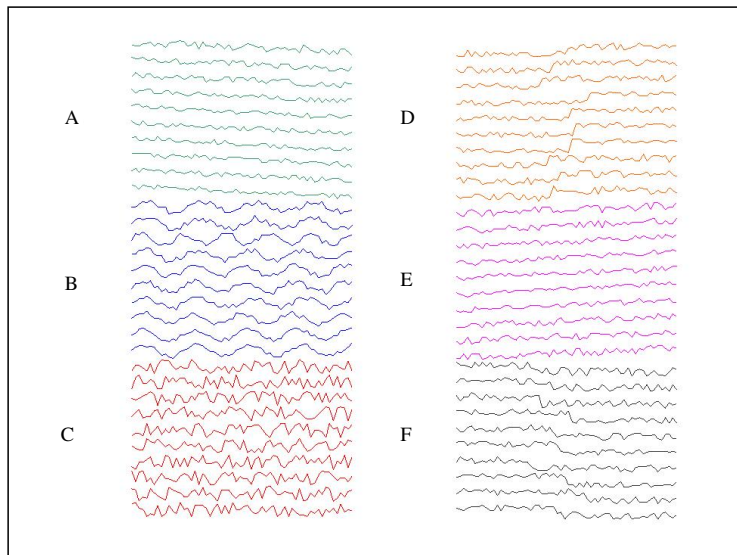
Data

- UCI Synthetic Control Chart Time Series Data Set contains 600 sequences of data.
- Partition data: 450 items for training; 150 items for testing.

Six Categories of Datastream

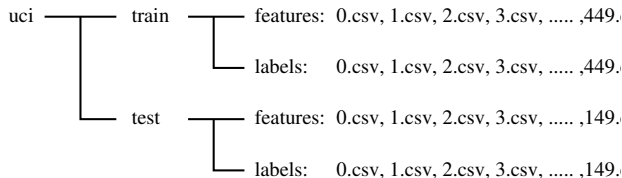
- **A and E:** Decreasing and Increasing Trend.
- **B:** Cyclic.
- **C:** Normal.
- **D and F:** Upward and Downward Shift.

Example 5: Representative Data Streams



Example 5: Control Chart Sequence Data

Training and Testing Corpus



CSV Data File Format and Label Mappings

Features: 0.csv		Labels: 0.csv		Output Mapping
20.59	← line 01	3	←	0 ← A
18.51				1 ← B
⋮	⋮			2 ← C
⋮	⋮			3 ← D
⋮	⋮			4 ← E
16.32				5 ← F
10.72	← line 60			

Example 5: Control Chart Sequence Data

DL4J: Load Training Data, Features and Labels

```
// Initialize features and labels training directory ...

private static File baseDir = new File("/resources/uci/");
private static File baseTrainDir      = new File(baseDir, "train");
private static File featuresDirTrain = new File(baseTrainDir, "features");
private static File labelsDirTrain   = new File(baseTrainDir, "labels");

// Read sequence of data files: 0.csv, 1.csv, ... 449.csv.

SequenceRecordReader trainFeatures = new CSVSequenceRecordReader();
trainFeatures.initialize(
    new NumberedFileInputSplit(
        featuresDirTrain.getAbsolutePath() + "/%d.csv", 0, 449));

SequenceRecordReader trainLabels = new CSVSequenceRecordReader();
trainLabels.initialize(
    new NumberedFileInputSplit(
        labelsDirTrain.getAbsolutePath() + "/%d.csv", 0, 449));
```

Example 5: Control Chart Sequence Data

DL4J: Create Minibatches and Dataset Iterator

```
int miniBatchSize    = 10;
int numLabelClasses  =  6;
```

```
DataSetIterator trainData = new SequenceRecordReaderDataSetIterator(
    trainFeatures, trainLabels, miniBatchSize, numLabelClasses,
    false, SequenceRecordReaderDataSetIterator.AlignmentMode.ALIGN_END );
```

DL4J: Normalize the Training Data

```
DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainData);           //Collect training data statistics
trainData.reset();
```

Example 5: Control Chart Sequence Data

DL4J: Assemble Neural Network Architecture

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()

// Random number generator seed for improved repeatability. Optional.

.seed(123)
.weightInit(WeightInit.XAVIER)
.updater(new Nadam())

// Not always required, but helps with this data set

.gradientNormalization(GradientNormalization.ClipElementWiseAbsoluteValue)
.gradientNormalizationThreshold(0.5)
.list()
.layer( new LSTM.Builder().activation(Activation.TANH).nIn(1).nOut(10).build() )
.layer( new RnnOutputLayer.Builder( LossFunctions.LossFunction.MCXENT )
.activation( Activation.SOFTMAX).nIn(10).nOut(numLabelClasses).build() )
.build();

MultiLayerNetwork net = new MultiLayerNetwork(conf);
net.init();
```

Example 5: Control Chart Sequence Data

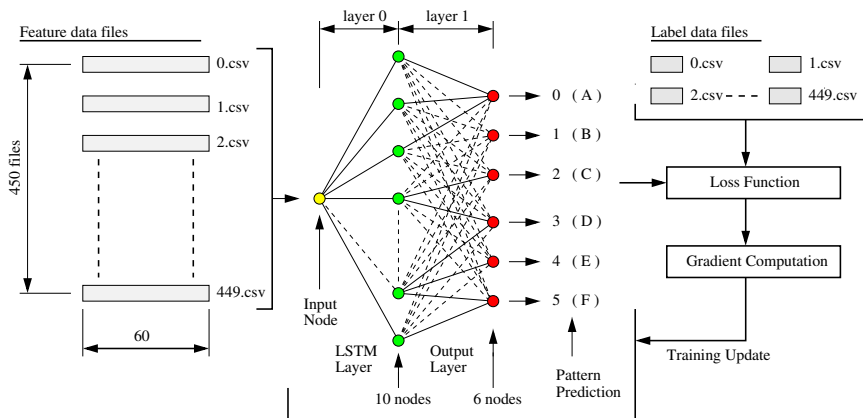
DL4J: Recurrent Neural Network Architecture

LayerName (LayerType)	nIn,nOut	Total Params	Params Shape
layer0 (LSTM)	1,10	480	W:{1,40}, RW:{10,40}, b:{1,40}
layer1 (RnnOutputLayer)	10,6	66	W:{10,6}, b:{1,6}
Total Parameters: 546			
Trainable Parameters: 546			
Frozen Parameters: 0			

Note: 1 input; 6 outputs.

Example 5: Control Chart Sequence Data

RNN Architecture + Sequences of Feature and Label vectors



Example 5: Control Chart Sequence Data

DL4J: Training the Model for 40 Epochs

```
int nEpochs = 40;  
net.fit(trainData, nEpochs);
```

Evaluation Metrics and Confusion Matrix

Accuracy: 0.8867 Recall: 0.8890 <--- It works!
Precision: 0.8886 F1 Score: 0.8883

	0	1	2	3	4	5	

26	0	0	0	0	0	0	0 = 0
0	29	0	0	0	0	0	1 = 1
0	0	15	0	7	0	0	2 = 2
0	0	0	20	0	1	0	3 = 3
0	0	9	0	21	0	0	4 = 4
0	0	0	0	0	0	22	5 = 5

References

- Amidi A., and Amidi S., Cheatsheets on Machine Learning and Deep Learning, Various courses on ML at Stanford and MIT, 2018 – 2020.
- Apache DataVec. Library for machine learning ETL (Extract, Transform, Load) Operations. See:
<https://github.com/deeplearning4j/DataVec>.
- Foster D., Generative Deep Learning: Teaching Machines to Paint, Write, Compose and Play, O'Reilly, 2019.
- Hochreiter S., and Schmidhuber J., Long Short-Term Memory, Neural Computation, Vol. 9, No. 8, 1997, pp. 1735-1780.
- Nielsen A., Practical Time Series Analysis: Prediction with Statistics and Machine Learning, O'Reilly, 2020.