Dynamically Driven Recurrent Neural Networks (RNN)

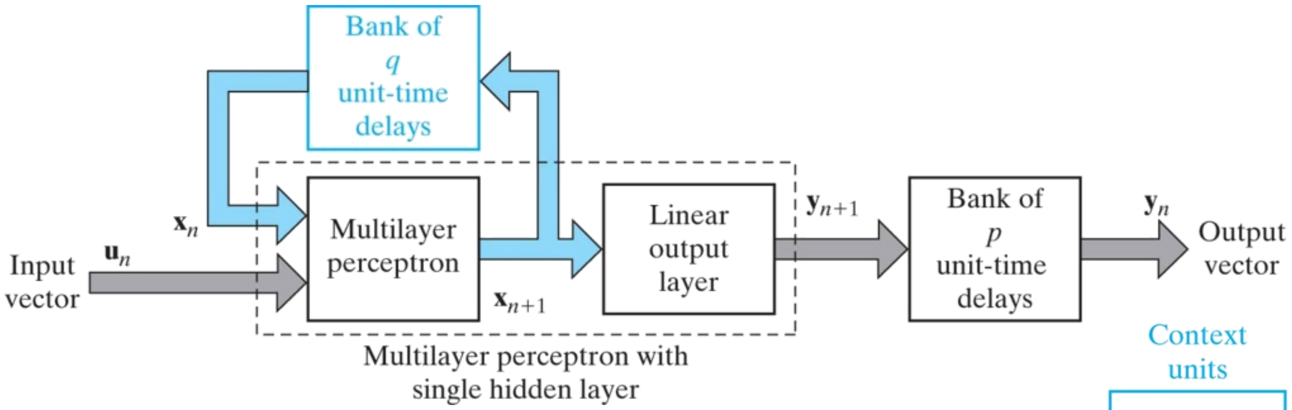
Dr. Ratna Babu Chinnam
Industrial & Systems Engineering
Wayne State University

Primary Source: Simon Haykin (2009)

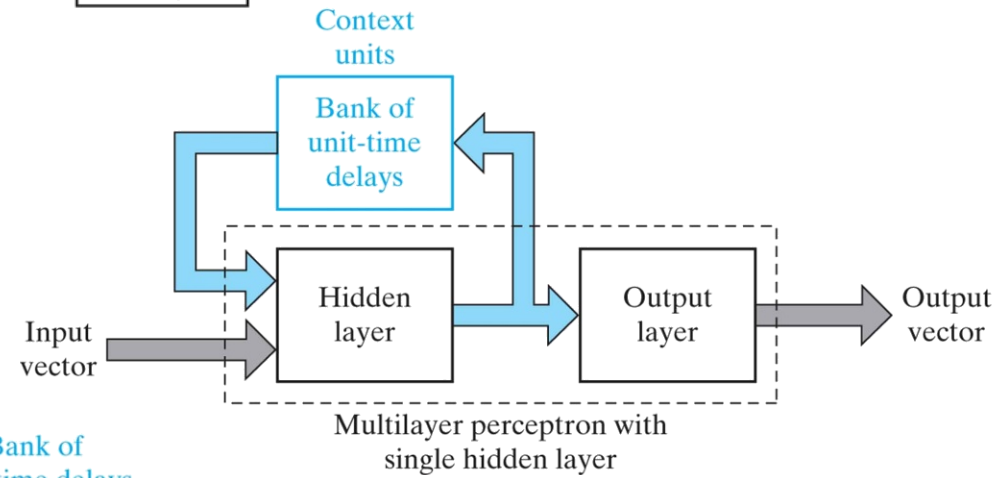
Dynamically Driven Recurrent Neural Networks

- Recurrent networks have one or more “*feedback*” loops
  - Network responds “*temporally*” to an externally applied input signal, providing ability to map “*dynamic*” systems
  - Allow network to acquire “*state*” representations
- Feedback can take a variety of forms, leading to a variety of networks
- In recent years, RNNs had incredible success to a variety of problems
  - Forecasting, speech recognition, language modeling, translation, image captioning …
  - Check out Andrej Karpathy’s excellent blog post: The Unreasonable Effectiveness of Recurrent Neural Networks
- Alternative to focused/distributed TLFNs and can do much more!
  - Whereas TLFNs employ FIR filters for memory, recurrent networks generally leverage IIR (infinite impulse response) filters
  - Reduces memory requirement and result in compact and more effective networks
- *Challenge*: Calculation of gradients in recurrent networks needs far more care

Example RNN Architectures

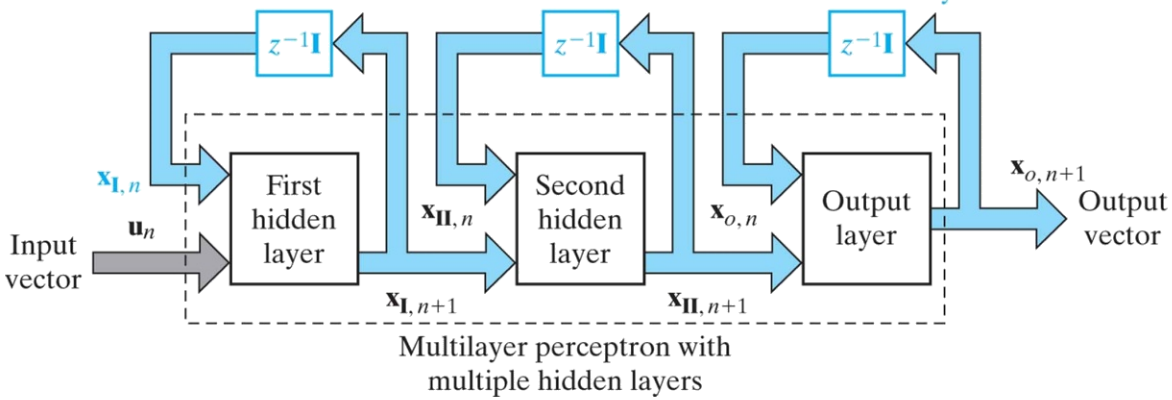


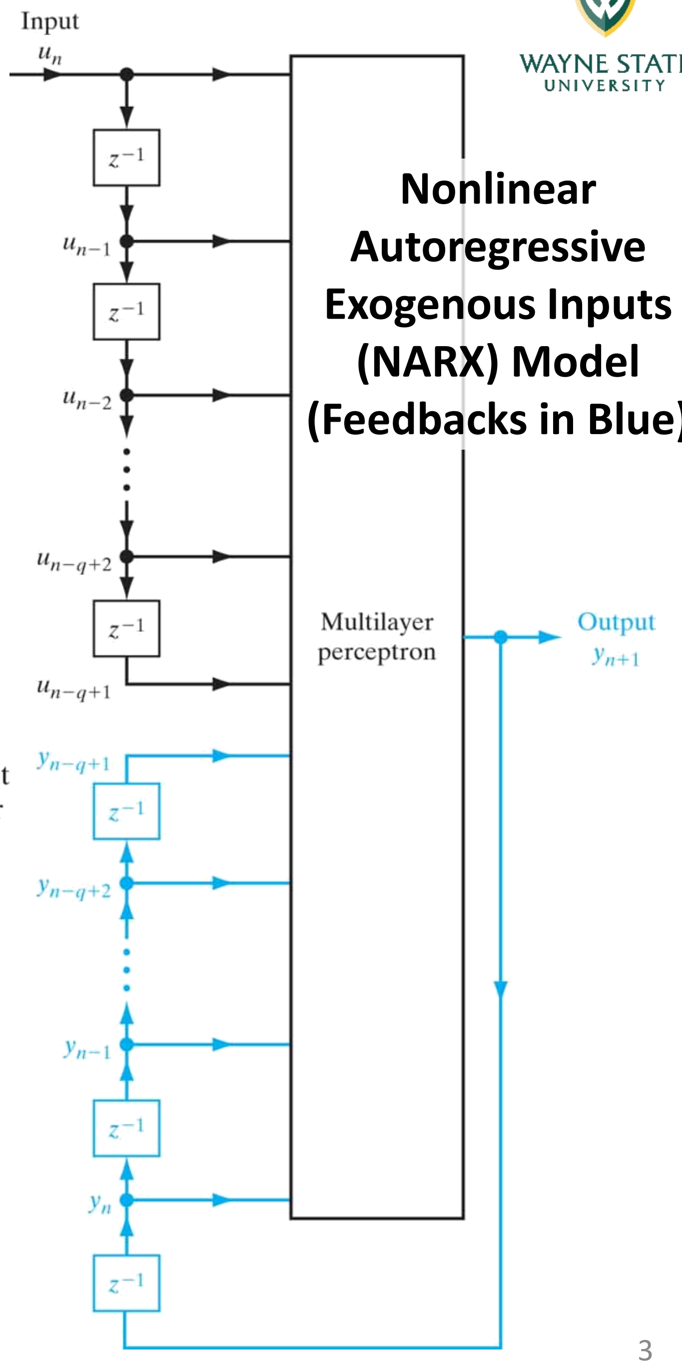
State-Space Model



Elman Network or Simple Recurrent Network

Recurrent Multilayer Perceptron





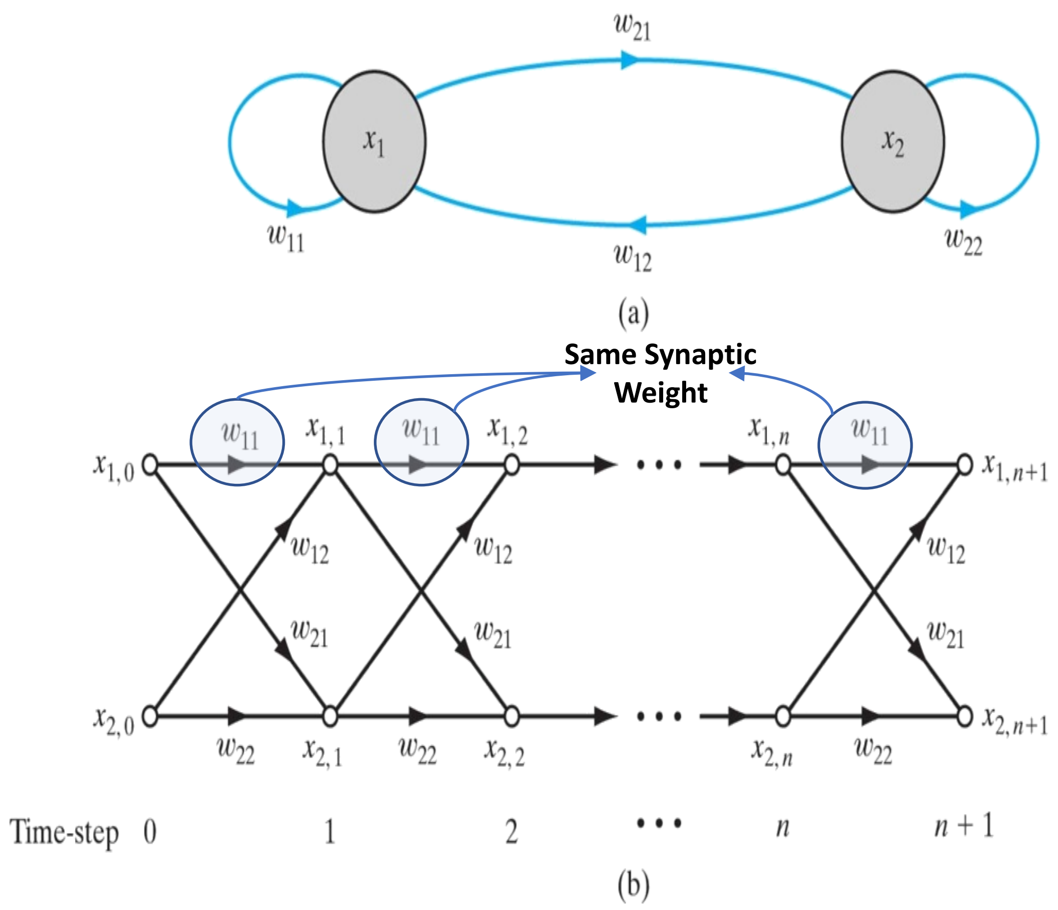
Nonlinear Autoregressive Exogenous Inputs (NARX) Model (Feedbacks in Blue)

Learning Algorithms for RNNs

- Recurrent networks can be trained in two modes
  - ***Epoch-wise Training***
    - For each time-series episode/epoch, network starts from some initial state until it reaches the end of the episode and then switches to next episode/epoch
    - Example: For machining “cutting tool” diagnostics applications, time-series dataset from each cutting tool collected over its entire life forms an episode
  - ***Continuous Training:*** Suitable for on-line learning
- Two gradient descent learning algorithms popular for RNNs:
  - **Back-Propagation Through Time** (BPTT) Algorithm
  - **Real-Time Recurrent Learning** (RTRL) Algorithm
- BPTT requires less computation but more memory than RTRL
  - BPTT is more suitable for off-line learning and RTRL for online learning

Back-Propagation Through Time (BPTT) Algorithm

- An extension of the standard back-propagation algorithm
- Derived by "*unfolding*" the temporal operation of network into a layered "feedforward" network
  - Weight Sharing: Same synaptic weights are repeating across network
- Network topology grows by one layer for every time-step
- Two options for training:
  - Epochwise BPTT
  - Truncated BPTT



(a) Architectural Graph of 2-Neuron Recurrent Network
(b) Signal-flow Graph of Network Unfolded in Time

Epochwise Back-Propagation Through Time

- Let n_0 and n_1 denote start, end times of epoch with cost

$$\xi_{Total} = \frac{1}{2}\sum_{n=n_0}^{n_1} \sum_{j \in H} e_{j,n}^2$$

  where $\mathcal{H}$ is set of neuron indices j with desired responses
- *Forward pass* over entire epoch data through network for interval (n_0, n_1)
  - Complete record of inputs, network state, outputs are all saved in memory
- *Backward pass* over record is performed to compute values of local gradients:

$$\delta_{j,n} = \begin{cases} \varphi'(v_{j,n})e_{j,n} & \text{for } n = n_1 \\ \varphi'(v_{j,n})\left[e_{j,n} + \sum_{k \in H} w_{kj}\delta_{k,n+1}\right] & \text{for } n_0 < n < n_1 \end{cases}$$

- Once back propagation is performed back to time $n_0 + 1$, weights are adjusted:

$$\Delta w_{ji} = -\eta \frac{\partial \xi_{Total}}{\partial w_{ji}} = \eta \sum_{n=n_0+1}^{n_1} \delta_{j,n}\, x_{i,n-1}$$

Impact of Weight Sharing

Truncated Back-Propagation Through Time

- Truncates unfolding to a finite depth h to reduce memory requirement
- Weight adjustments made on a continuous basis (no full passes)
  - Like pattern mode of learning with ordinary back-propagation algorithm
- Employs instantaneous value of the sum of squared errors: $\xi_n = \frac{1}{2}\sum_{j\in\mathcal{H}} e_{j,n}^2$
- Leads to following local gradient expression:

$$\delta_{j,l} = \begin{cases} \varphi'(v_{j,l})e_{j,l} & \text{for } l = n \\ \varphi'(v_{j,n})\left[\cancel{e_{j,n}} + \sum_{k\in\mathcal{H}} w_{kj,l}\delta_{k,l+1}\right] & \text{for } n-h < l < n \end{cases}$$

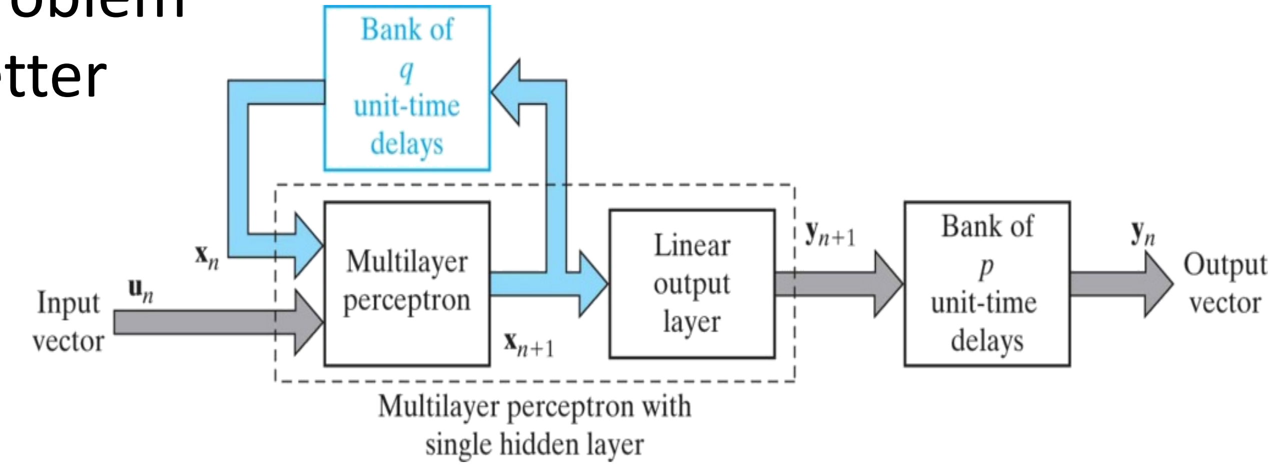
- Weight updates are as follows:

$$\Delta w_{ji,n} = \eta \sum_{l=n-h+1}^{n} \delta_{j,l}\, x_{i,l-1}$$

WAYNE STATE UNIVERSITY

Real-Time Recurrent Learning (RTRL) Algorithm

- Derived here for "state-space" model
  - Suffers from vanishing-gradients problem
  - Second-order methods perform better
- Summary of RTRL Algorithm



State-Space Model

Parameters:
m = dimensionality of the input space
q = dimensionality of the state space
p = dimensionality of the output space
$\mathbf{w}_j$ = synaptic-weight vector of neuron j, $j = 1, 2, \ldots, q$

Initialization:
1. Set the synaptic weights of the algorithm to small values selected from a uniform distribution.
2. Set the initial value of the state vector $\mathbf{x}(0) = \mathbf{0}$.
3. Set $\mathbf{\Lambda}_{j,0} = \mathbf{0}$ for $j = 1, 2, \ldots, q$.

Computations: Compute the following for $n = 0, 1, 2, \ldots$;

$$\mathbf{e}_n = \mathbf{d}_n - \mathbf{W}_c\mathbf{x}_n$$
$$\Delta\mathbf{w}_{j,n} = \eta\mathbf{W}_c\mathbf{\Lambda}_{j,n}\mathbf{e}_n$$
$$\mathbf{\Lambda}_{j,n+1} = \mathbf{\Phi}_n(\mathbf{W}_{a,n}\mathbf{\Lambda}_{j,n} + \mathbf{U}_{j,n}), \quad j = 1, 2, \ldots, q$$

The definitions of $\mathbf{x}_n$, $\mathbf{\Lambda}_n$, $\mathbf{U}_{j,n}$ and $\mathbf{\Phi}_n$ are given in Eqs. (15.42). (15.45), (15.46), and (15.47), respectively.

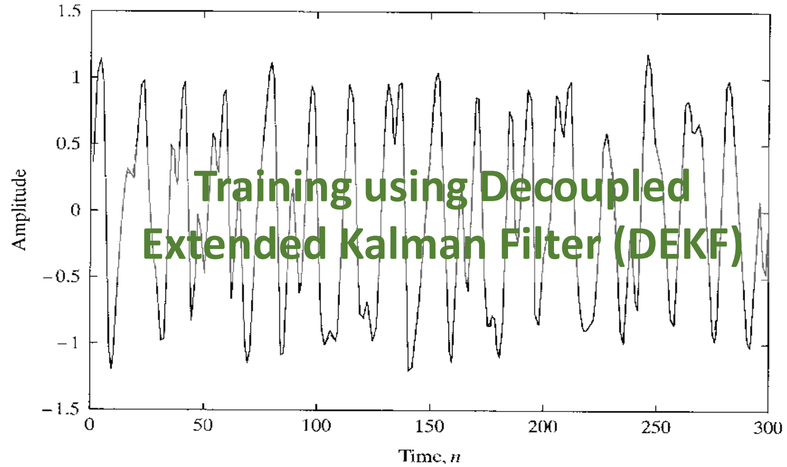
Computer Experiment: RMLP vs Focused TLFN

- **TASK:** One-step ahead forecasting of a frequency modulated signal:

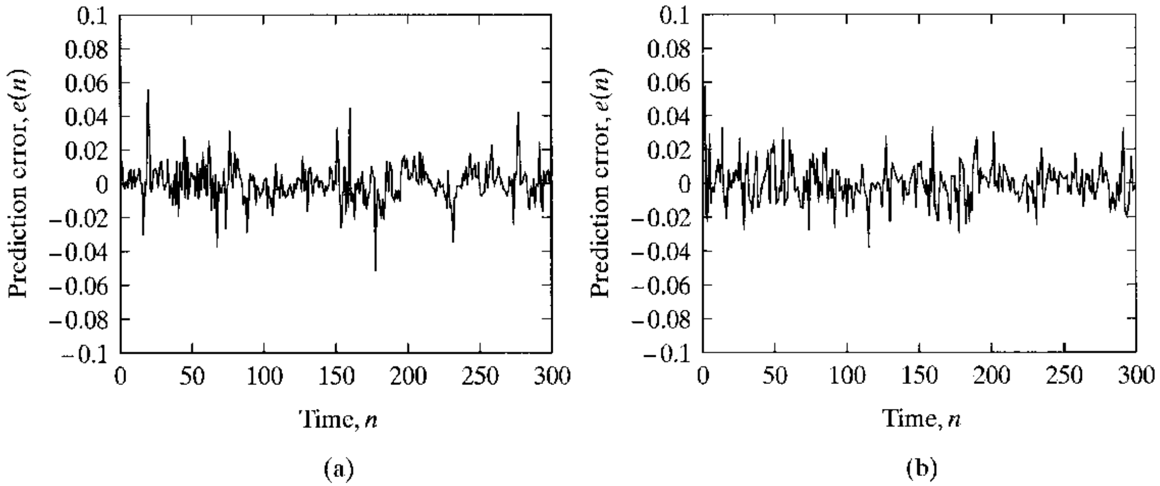
  $x(n) = \sin(n + \sin(n^2))\ \ n = 0,1,2,\cdots$

- **Parameters of RMLP Network:**
  - One input node
  - One hidden layer of 10 neurons
  - One linear output neuron
- **Parameters of Focused TLFN:**
  - One input node with 20 taps
  - One hidden layer of 10 neurons
  - One linear output neuron

Note: RMLP has slightly more synaptic weights than the focused TLFN, but half the memory (10 recurrent nodes vs 20 taps)



Actual (continuous) and Predicted (dashed) Waveforms for RMLP Trained Using RTRL



DEKF **Errors: (a) RMLP with RTRL, Variance = 1.1839 x 10^{-4}. (b) Focused TLFN, Variance = 1.3351 x 10^{-4}.**

Long Short-Term Memory (LSTM) Networks

WAYNE STATE UNIVERSITY

- Human's thoughts have persistence!
  - **Example**: If you are reading a novel about a vacation to France, entry into France might be discussed in the first chapter but we remember that as we read the entire novel ("**context**" is **maintained naturally**)
- Traditional neural networks cannot do this
  - General **recurrent networks** attempt this, but **do not manage it explicitly**
  - Another name for this is ***stability-plasticity dilemma***:
    - Well-known constraint for artificial and biological neural systems
  - Learning requires **plasticity for integration of new knowledge** but also **stability to prevent forgetting of previous knowledge**
- Essential to success of RNNs in recent years is use of "LSTMs"
  - Most exciting RNN results are achieved with LSTMs. - Colah (2015)

SOURCE: Colah's Blog: Understanding LSTM Networks | LINK

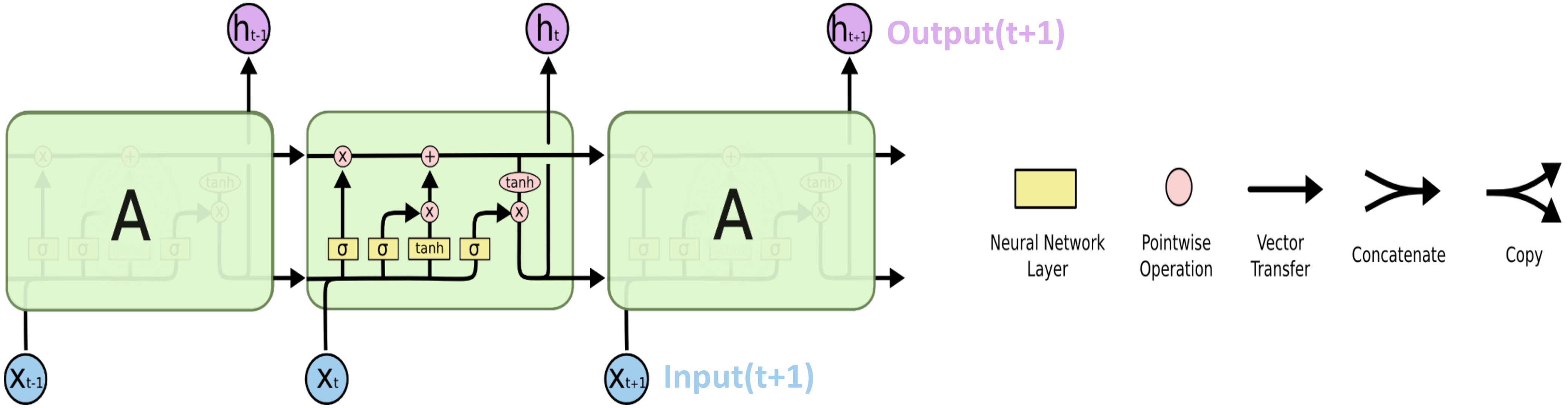
Long Short-Term Memory (LSTM) Networks …

WAYNE STATE UNIVERSITY

- An LSTM network is a ***type of recurrent neural network*** (RNN) that can learn long-term dependencies between time steps of sequence data, **seeking better "context" management**
- **LSTMs are designed to avoid the long-term dependency problem**
  - **Remembering information for prolonged periods of time is their default behavior**, not something they struggle to learn!

Long Short-Term Memory (LSTM) Networks …

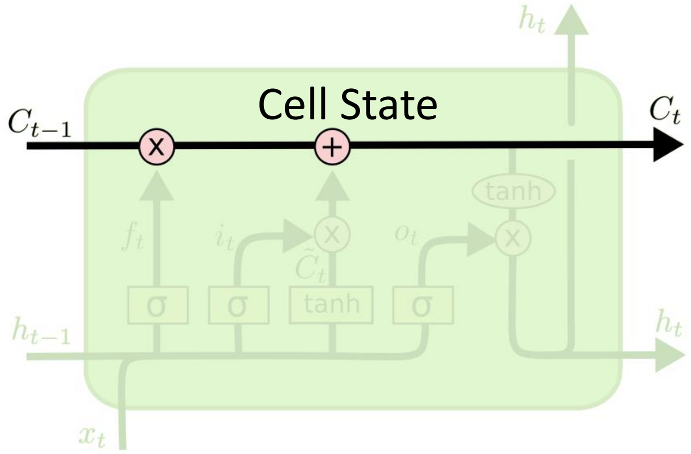
- Many RNNs have the form of a chain of repeating layer modules
- **Repeating module of an LSTM has a different structure**
    - Instead of a single "layer", there are four, interacting in a special way

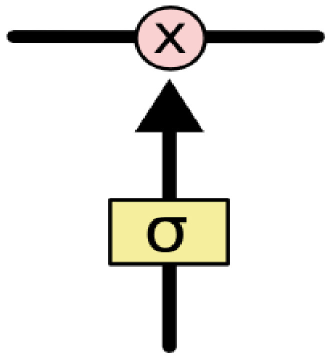


- "Pink" circles represent pointwise operations (e.g., vector addition) while "yellow" boxes are learned neural network layers
- Merging lines denote concatenation while forking denote content being copied and going to different locations

Core Idea Behind LSTMs

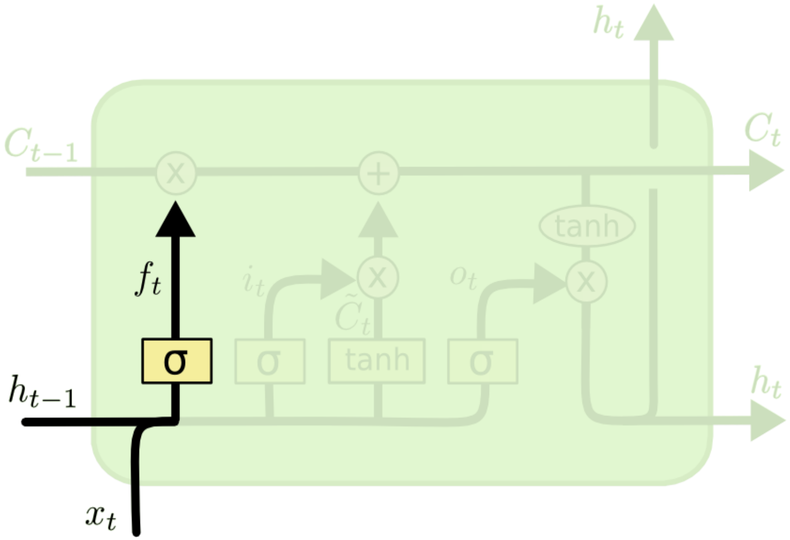
- **Key to LSTMs is "cell" state (maintaining context)**, the horizontal line running through top
  - **Cell state is like a conveyor belt:** It runs the entire chain with some minor linear interactions (easy for information to just flow along it)
- **LSTM has ability to remove/add information to cell state, regulated by structures called "gates"**
- **Gates are a way to optionally let information through**
  - Composed of a sigmoid layer and a pointwise multiplication operation
  - **Output numbers between 0-1:** "0" means "let nothing through" and "1" means "let everything through"
- **LSTM has three of these gates**, to protect and control cell state





Step-by-Step LSTM Walk Through

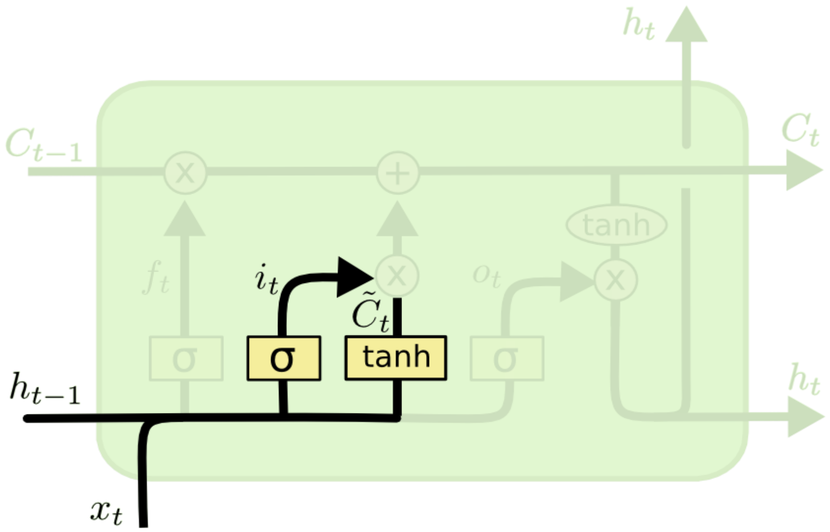
- **First Step: Decide information to throw away from cell state**
- **Decision is made by a sigmoid layer** called "**forget gate** **layer**"
  - It looks at h_{t-1} and x_t, and outputs a number between 0-1 for each number in cell state C_{t-1}



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Step-by-Step LSTM Walk Through …

- **Next Step: Decide what new information to store in the cell state**
- First, a sigmoid layer "**input gate layer**" decides which values we will update
- Next, a $tanh$ layer creates a vector of **new candidate values**, $\tilde{C}_t$, to add to the state
- We will combine these two to create an update to state

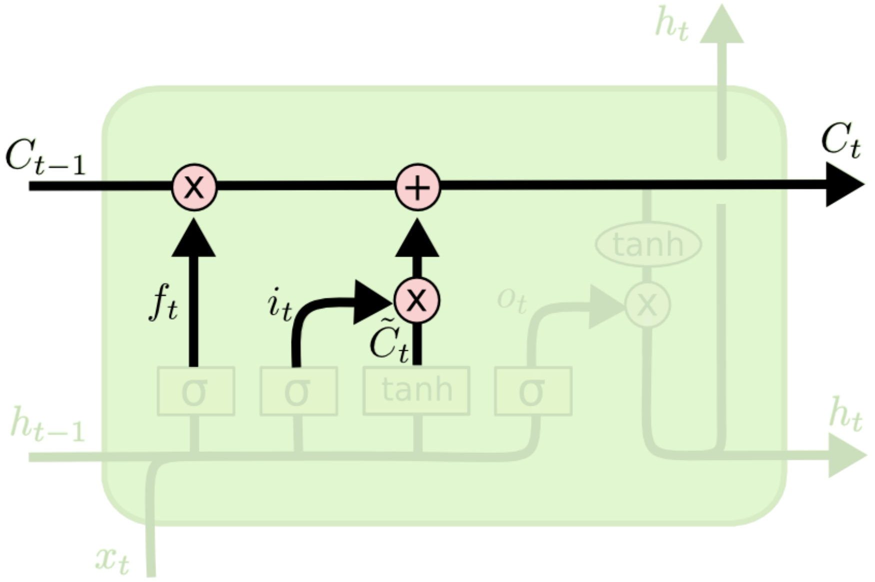


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Step-by-Step LSTM Walk Through …

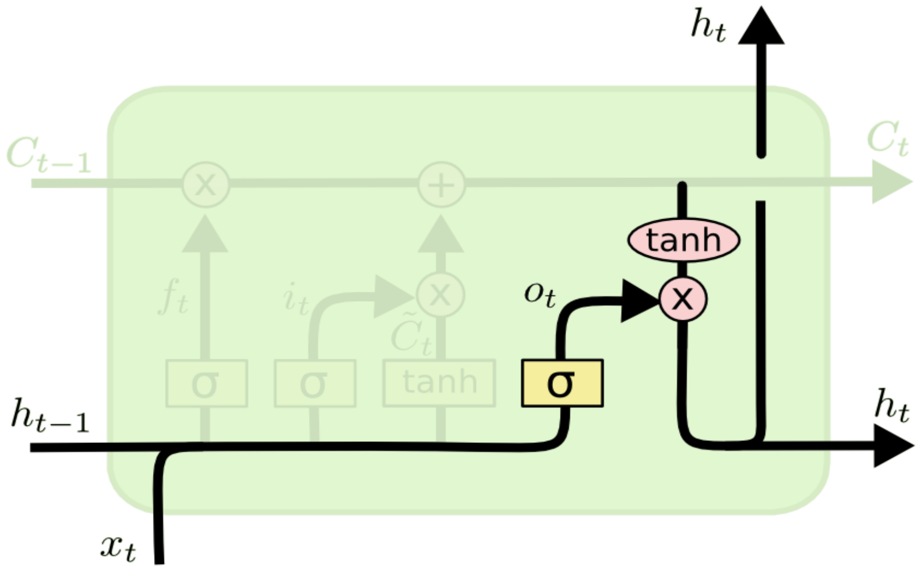
- **It is time to update old cell state**, C_{t-1}, into the new cell state C_t
- We multiply old state by f_t, forgetting things we decided to forget earlier
- Then we add $i_t * \tilde{C}_t$
  - These are the new candidate values, scaled by how much we decided to update each state value



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step-by-Step LSTM Walk Through …

- **Final Step: Decide what to output**
  - Will depend on cell state but will be filtered
- **Output gate**: A sigmoid layer decides what parts of cell state to output
- We put the cell state through $tanh$ (to push the values to be [−1, 1]) and multiply it by output of sigmoid gate, so that we only output the parts we decided to



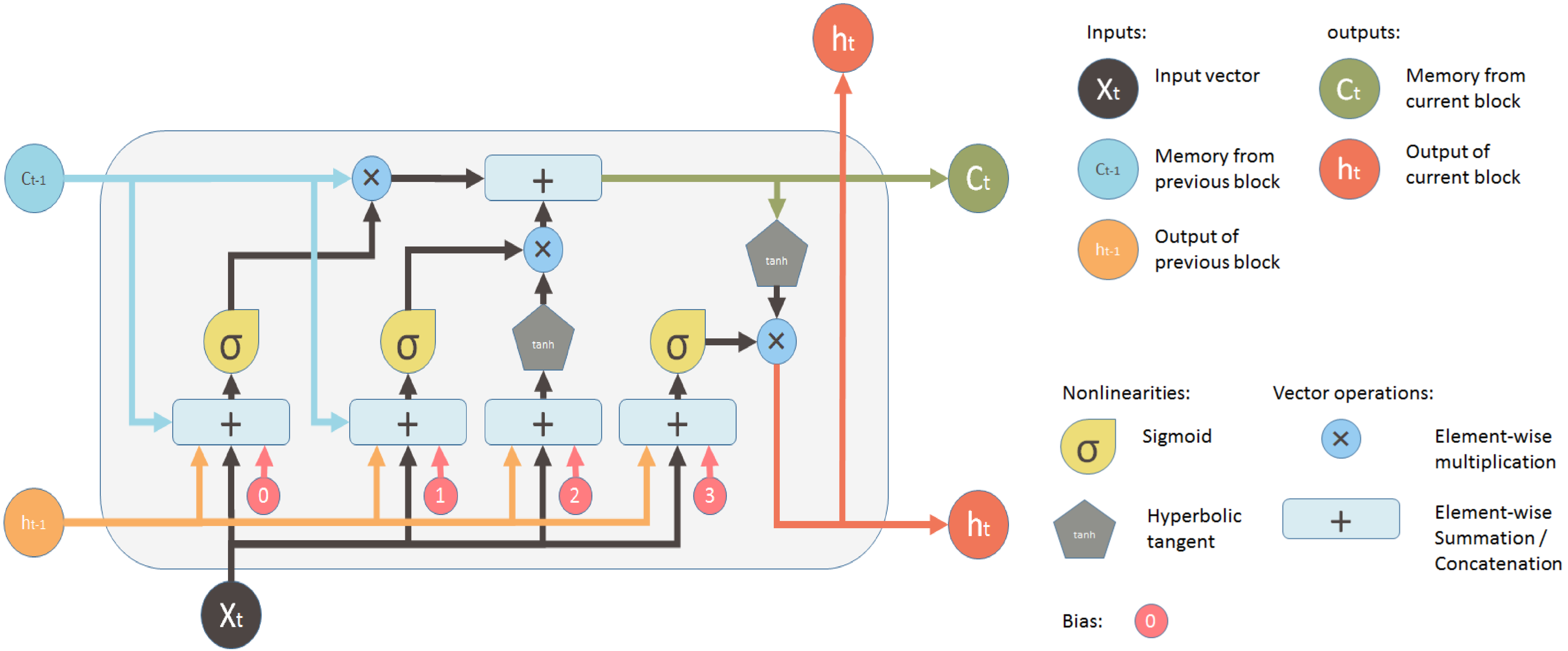
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Other LSTM variants exist!

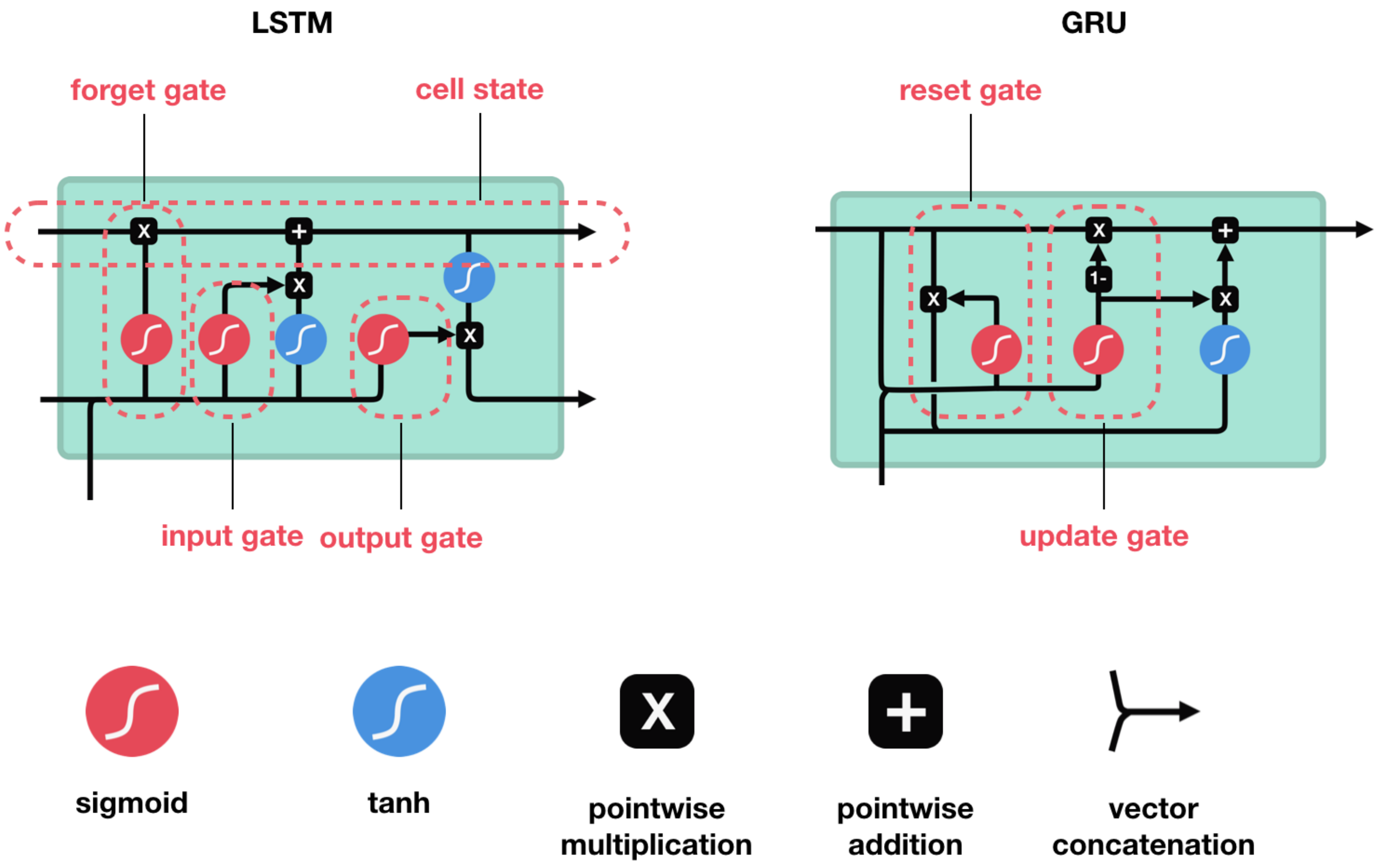
While sophisticated, is it sophisticated enough?

Long Short-Term Memory (LSTM) Module



Gated Recurrent Unit (GRU) Module

- **GRU is like an LSTM but simpler**
- It has **no cell state** and uses the hidden state to transfer information
  - Has only two gates: reset and update
- GRU has fewer tensor operations; little speedier to train then LSTMs
- No clear winner; try both options (GRU and LSTM)



Credit: Michael Nguyen | Link

Implementing LSTMs in Matlab | LINK

- Components of LSTM Networks:
  - “**Sequence Input Layer**”: Create using sequenceInputLayer
  - **LSTM Layer**: Create using lstmLayer
  - **Bidirectional LSTM Layer**:
    - Useful when you want network to learn from complete time series at each time step; Create using bilstmLayer
  - **Deep LSTMs**: Insert extra LSTM layers with output mode 'sequence' before LSTM layer

“Sequence-to-Label” Classification

Output mode of last LSTM layer must be ‘last’


```
numFeatures = 12;
numHiddenUnits1 = 125;
numHiddenUnits2 = 100;
numClasses = 9;
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits1,'OutputMode','sequence')
    lstmLayer(numHiddenUnits2,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

Note: For “sequence-to-sequence” classification, output mode of last LSTM layer must be ‘sequence’

Implementing LSTMs using TensorFlow: Tutorial with Code by Rowel Atiena

Matlab LSTM Example: Recognize Speaker*

Try in Matlab | [LINK]

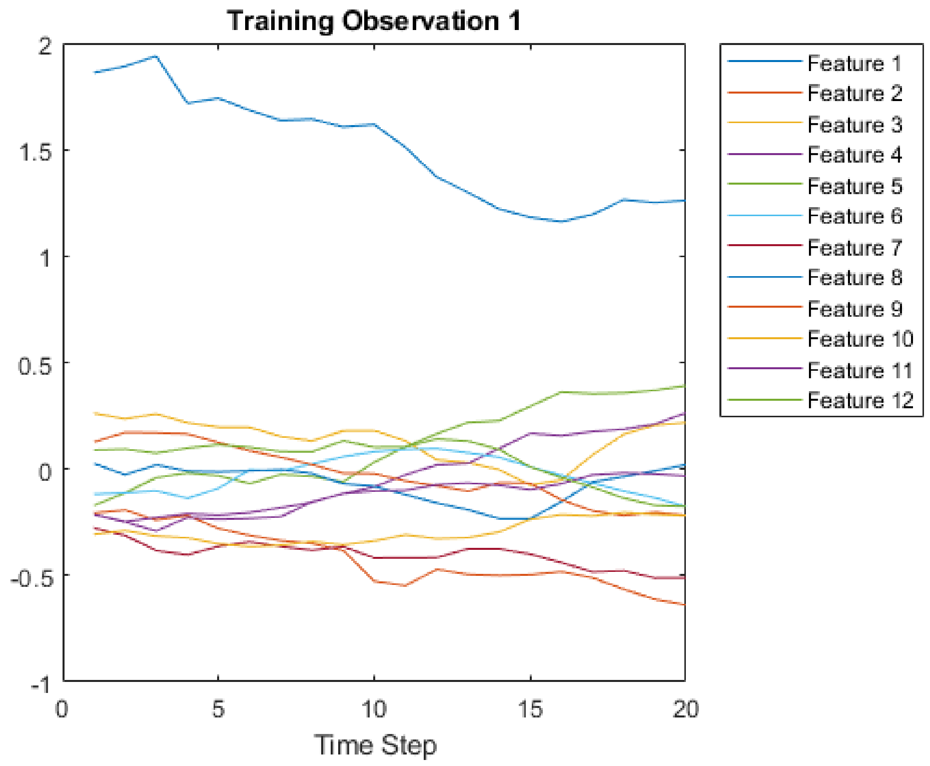
```
openExample('nnet/ClassifySequenceDataUsingLSTMNetworksExample')
```

- **Task**: Trains an LSTM network to recognize speaker given time series data representing two Japanese vowels spoken in succession
- Example uses Japanese Vowels dataset from Kudo et al. (1999)
- Training data from nine speakers
- Each sequence has 12 features and varies in length
- Dataset contains 270 training observations and 370 test observations

Load Sequence Data

- Load Japanese Vowels training data:
  - Y is a vector of labels "1","2",...,"9", correspond to nine speakers

```
[XTrain,YTrain]=japaneseVowelsTrainData
```

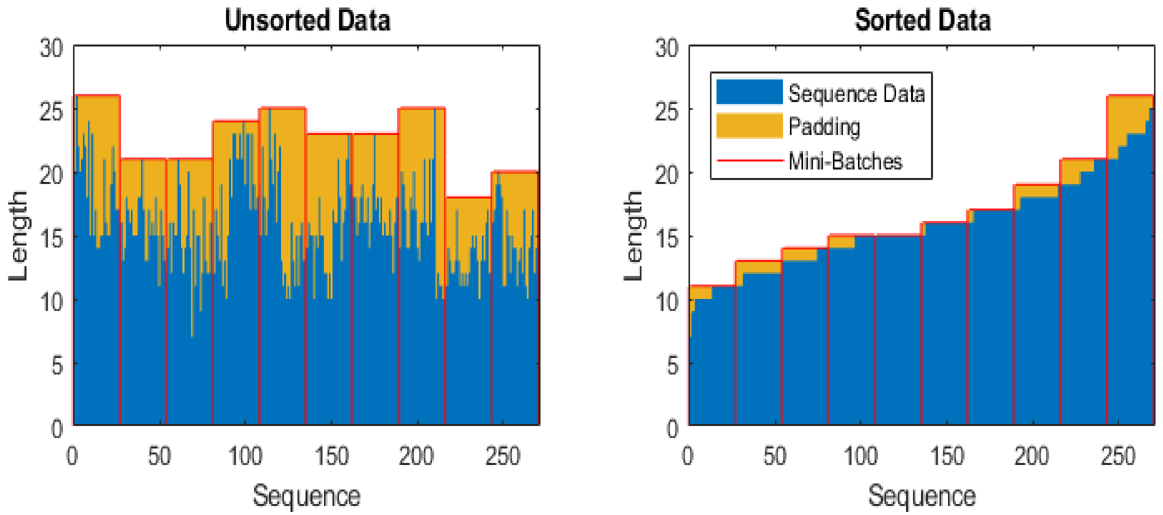


SOURCE: Kudo, M., Toyama, J., & Shimbo, M. (1999). Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, *20*(11-13), 1103-1111. *UCI ML Repository: Japanese Vowels Dataset*.

Matlab LSTM Example: Recognize Speaker …

Prepare Data for Padding

- During training, software splits data into mini-batches and pads sequences so that they have same length
  - Too much padding can have a negative impact on network performance
- To prevent too much padding, sort data by sequence length, and choose a mini-batch size so that sequences in a mini-batch have a similar length



- Get sequence lengths for observation:

```
numObservations = numel(XTrain);
for i=1:numObservations
    sequence=XTrain{i};
    sequenceLengths(i)=size(sequence,2);
end
```

- Sort data by sequence length:

```
[sequenceLengths,idx]=sort(sequenceLengths);
XTrain = XTrain(idx);
YTrain = YTrain(idx);
```

- Choose a mini-batch size of 27 to divide training data evenly and reduce amount of padding in the mini-batches:

```
miniBatchSize = 27;
```

Matlab LSTM Example: Recognize Speaker …

Define LSTM Network Architecture

- Specify input size to sequences of size 12
- Specify an bidirectional LSTM layer with 100 hidden units, and output last element of sequence
- Specify nine classes by including a fully connected layer of size 9, followed by a softmax layer and a classification layer.
  - If you can access full sequences for prediction, use bidirectional LSTM layer

Define Structure:

```
inputSize = 12;
numHiddenUnits = 100;
numClasses = 9;

layers = [ ...
 sequenceInputLayer(inputSize)
 bilstmLayer(numHiddenUnits,'OutputMode','last’)
 fullyConnectedLayer(numClasses)
 softmaxLayer
 classificationLayer]
```

Specify Training Options:

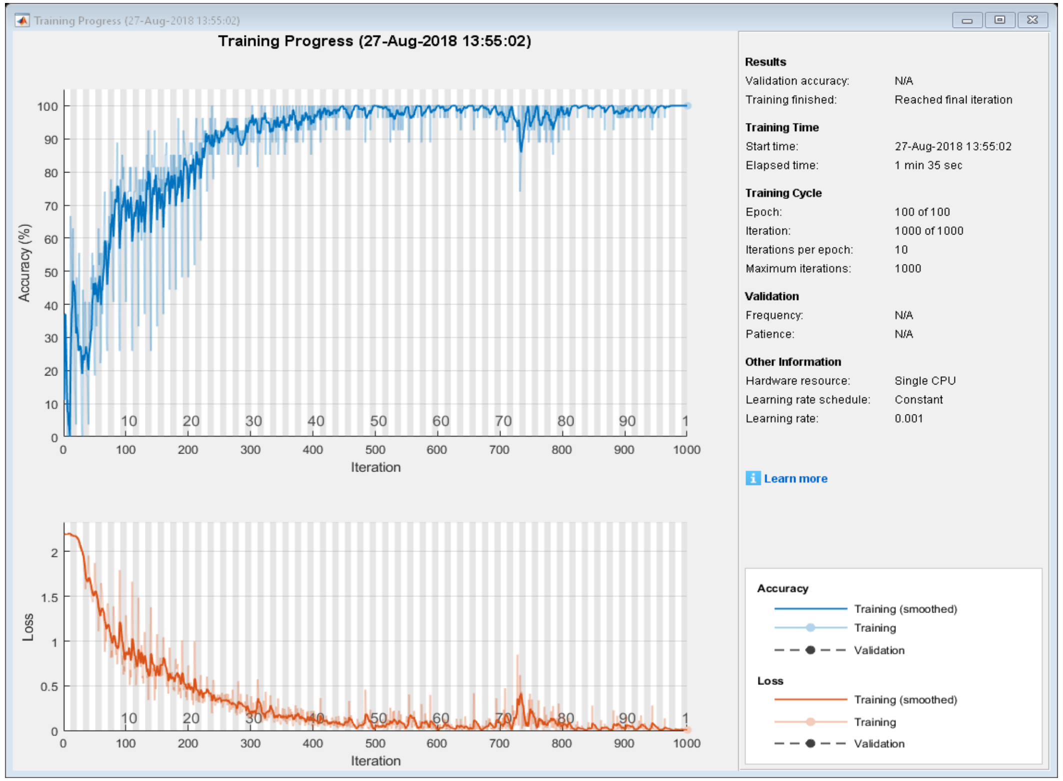
- Specify solver to be 'adam', gradient threshold to be 1, and max epochs to 100.
- To reduce padding, choose a mini-batch size of 27. To pad data to same length, set sequence length to be 'longest'. To ensure data remains sorted by sequence length, specify to never shuffle data.
- Since mini-batches are small with short sequences, training is better suited for CPU. Specify 'ExecutionEnvironment' to be 'cpu'. To train on a GPU, if available, set 'ExecutionEnvironment' to 'auto' (this is default value).

```
maxEpochs = 100;
miniBatchSize = 27;
options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    'GradientThreshold',1, 'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','longest', ...
    'Shuffle','never', 'Verbose',0, ...
    'Plots','training-progress');
```

Matlab LSTM Example: Recognize Speaker …

Train LSTM Network

```
net = trainNetwork(XTrain,YTrain,
layers,options);
```



- Load test dataset

```
[XTest,YTest] = japaneseVowelsTestData;
```

Test LSTM Network

- Sort test dataset by length

```
numObservationsTest = numel(XTest);
for i=1:numObservationsTest
 sequence = XTest{i};
 sequenceLengthsTest(i) = size(sequence,2);
end
[sequenceLengthsTest,idx] =
                  sort(sequenceLengthsTest);
XTest = XTest(idx);
YTest = YTest(idx);
```

- Classify test data

```
miniBatchSize = 27;
YPred = classify(net,XTest, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','longest');
```

- Calculate classification accuracy

```
acc = sum(YPred == YTest)./numel(YTest)
acc = 0.9324
```

Matlab LSTM Example: Remaining Useful Life

Try in Matlab | LINK

```
openExample('nnet/SequencetoSequenceRegressionUsingDeepLearningExample')
```

- **Task**: Predict remaining useful life (RUL) of turbofan engines for predictive maintenance (Saxena et al. 2008)
- RUL measured in cycles and time-series data represents engine sensors
- Training data contains simulated data for 100 engines
  - Each sequence has 17 features, varies in length, and corresponds to a full run to failure (RTF) instance
  - Test data contains 100 partial sequences and RUL at the end of each sequence
- Dataset contains 100 training and 100 test observations

Download Data

- Download Turbofan Engine Data Set from repository
- Each engine starts with unknown degrees of initial wear and manufacturing variation
  - Engine operating normally at the start of each time series and develops a fault at some point
  - In training set, fault grows until system failure
- Data contains text files with 26 columns of numbers, separated by spaces.
  - Each row is a snapshot of data taken during a single operational cycle
  - Column 1: Unit number
  - Column 2: Time in cycles
  - Columns 3–5: Operational settings
  - Columns 6–26: Sensor measurements 1–17

SOURCE: Saxena, A. et al. "Damage propagation modeling for aircraft engine run-to-failure simulation." In *Prognostics and Health Management, 2008. PHM 2008. International Conference on*, pp. 1-9. IEEE, 2008. | *NASA Ames Prognostics Data Repository* | REPOSITORY

Matlab LSTM Example: Remaining Useful Life …

Prepare Training Data

- `prepareDataTrain()` extracts data from filenamePredictors and returns cell arrays XTrain and YTrain

```
dataFolder = "data";
filenamePredictors =
fullfile(dataFolder,"train_FD001.txt");
[XTrain,YTrain] =
prepareDataTrain(filenamePredictors);
```

Remove Constant Rows & Normalize

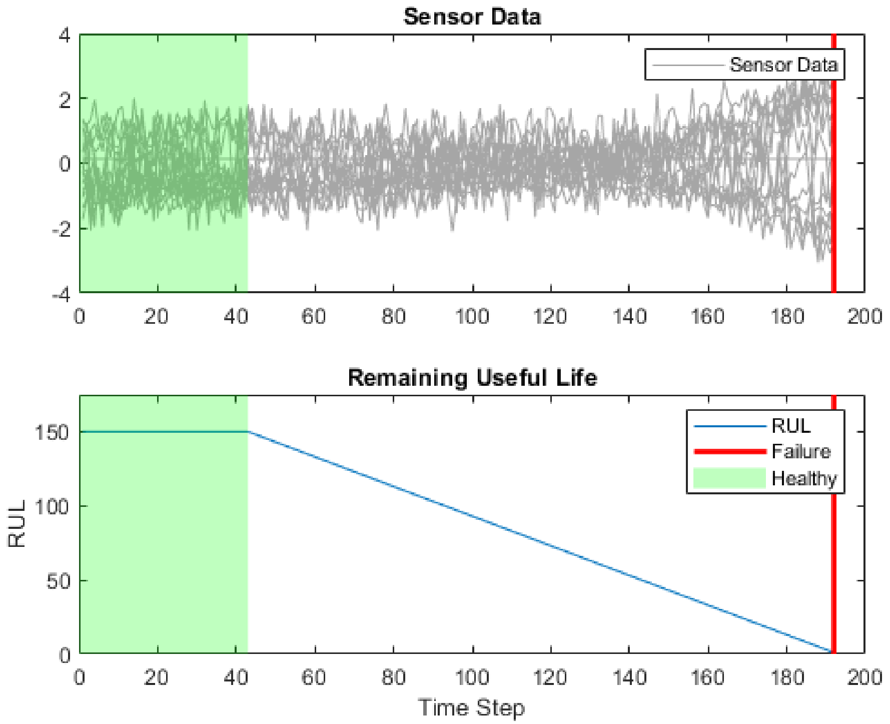
```
m = min([XTrain{:}],[],2); M =
max([XTrain{:}],[],2);
idxConstant = M == m;
for i = 1:numel(XTrain)
    XTrain{i}(idxConstant,:) = [];
end
mu = mean([XTrain{:}],2); sig =
std([XTrain{:}],0,2);
for i = 1:numel(XTrain)
    XTrain{i} = (XTrain{i} - mu) ./ sig;
end
```

Clip Reponses

- To learn more from sequence data when engines are close to failing, clip responses at threshold 150 cycles

```
thr = 150;
for i = 1:numel(YTrain)
    YTrain{i}(YTrain{i} > thr) = thr;
end
```

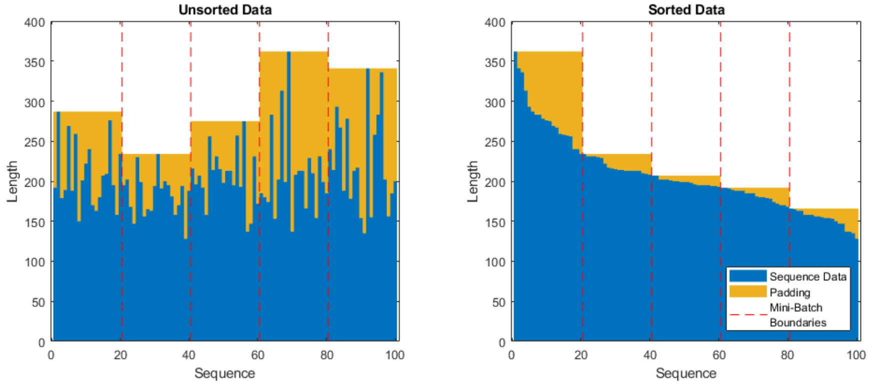
- First observation and clipped response



SOURCE: Saxena, A. et al. "Damage propagation modeling for aircraft engine run-to-failure simulation." In *Prognostics and Health Management, 2008. PHM 2008. International Conference on*, pp. 1-9. IEEE, 2008. | *NASA Ames Prognostics Data Repository* | REPOSITORY

Matlab LSTM Example: Remaining Useful Life …

Sort & Pad Data



Define LSTM Network Structure:

```
numResponses = size(YTrain{1},1);
featureDimension = size(XTrain{1},1);
numHiddenUnits = 200;

layers = [ ...
 sequenceInputLayer(featureDimension)
 lstmLayer(numHiddenUnits,'OutputMode','sequence’)
 fullyConnectedLayer(50)
 dropoutLayer(0.5)
 fullyConnectedLayer(numResponses)
 regressionLayer]
```

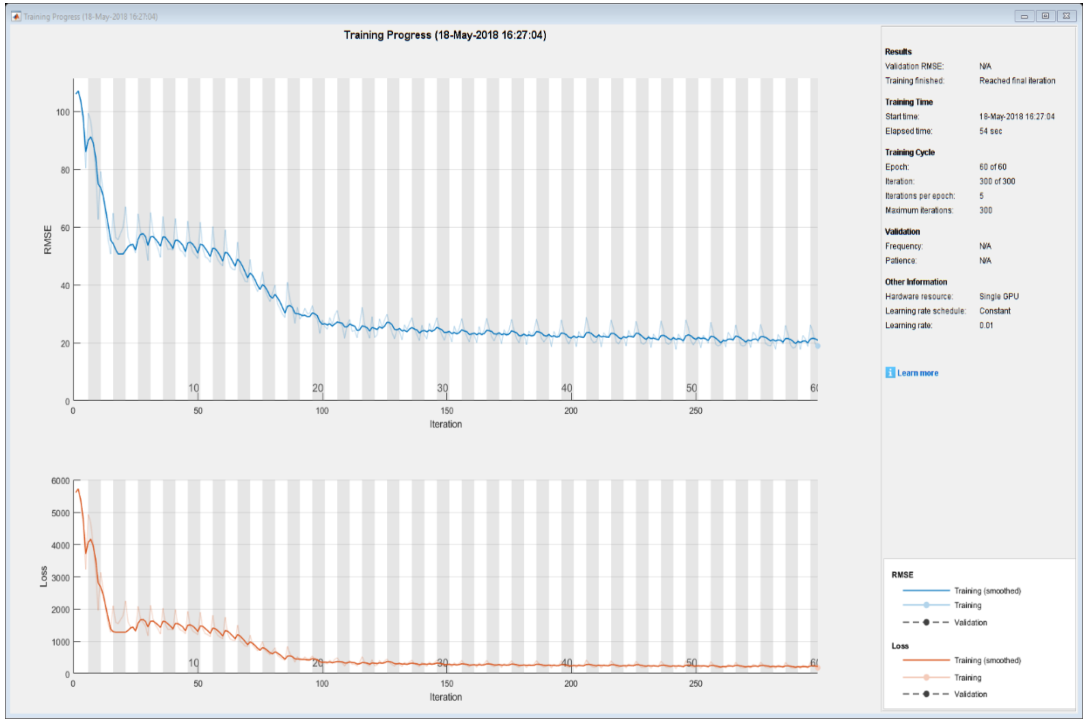
Specify Training Options:

```
maxEpochs = 60;
miniBatchSize = 20;
options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    'InitialLearnRate',0.01, ...
   'GradientThreshold',1,
'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','longest', ...
    'Shuffle','never', 'Verbose',0, ...
    'Plots','training-progress');
```

Matlab LSTM Example: Remaining Useful Life …

Train LSTM Network

```
net = trainNetwork(XTrain,YTrain,
layers,options);
```



- Load test data

```
filenamePredictors = fullfile(dataFolder,"test_FD001.txt");
filenameResponses = fullfile(dataFolder,"RUL_FD001.txt");
[XTest,YTest] =
prepareDataTest(filenamePredictors,filenameResponses);
```

Test LSTM Network

- Remove constant rows and normalize

```
for i = 1:numel(XTest)
    XTest{i}(idxConstant,:) = [];
    XTest{i} = (XTest{i} - mu) ./ sig;
    YTest{i}(YTest{i} > thr) = thr;
end
```

- Predict (specify batch size of 1 to avoid padding)

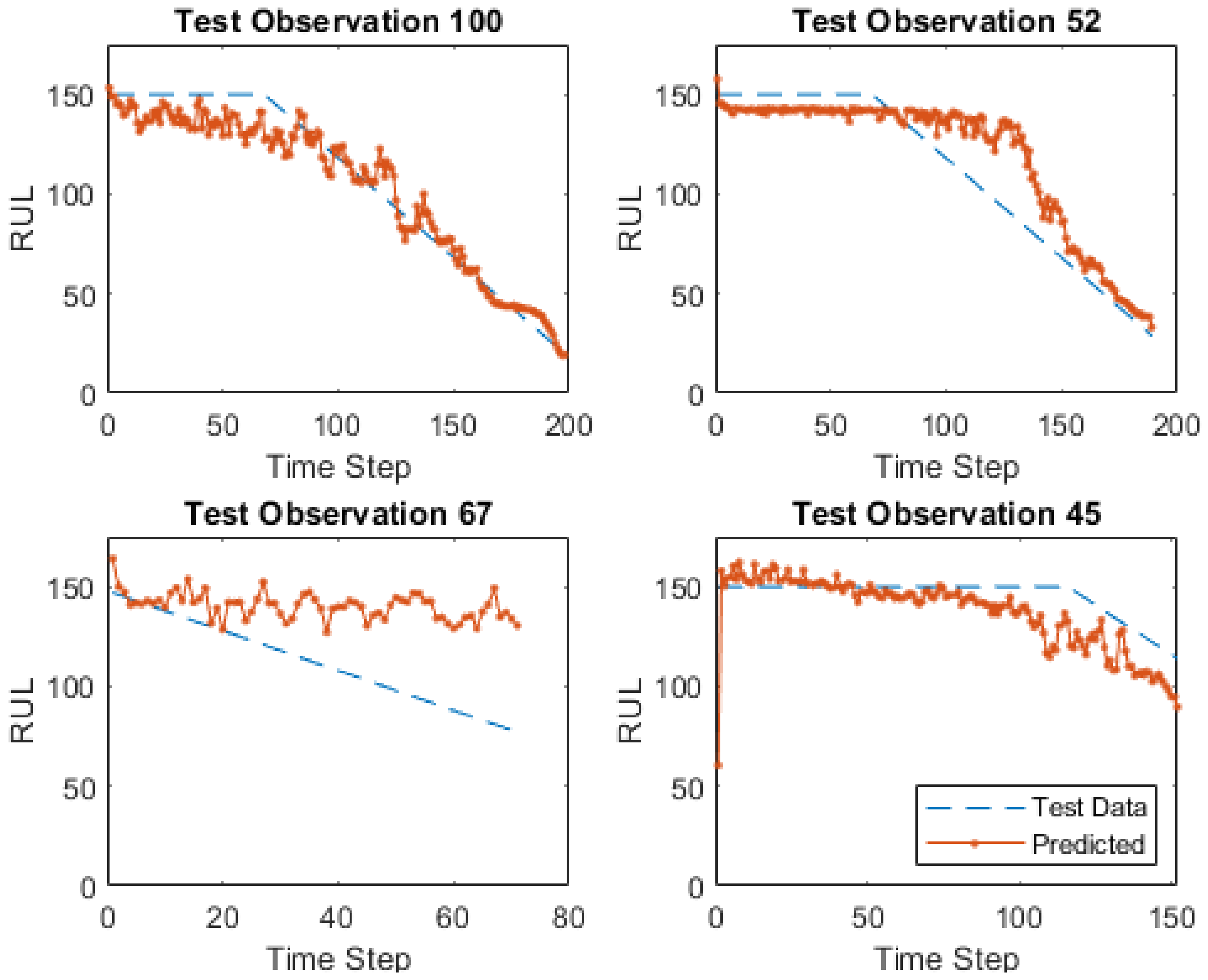
```
YPred = predict(net,XTest,'MiniBatchSize',1);
```

- Notes: Network makes predictions on partial sequence one time step at a time
  - At each time step, network predicts using value at this time step, and network state calculated from previous time steps only.
  - Network updates its state between each prediction
  - Predict function returns a sequence of these predictions
  - Last element of prediction corresponds to predicted RUL for partial sequence

Matlab LSTM Example: Remaining Useful Life …

Visualize Some Predictions

```
idx = randperm(numel(YPred),4);
figure
for i = 1:numel(idx)
    subplot(2,2,i)
    plot(YTest{idx(i)},'--')
    hold on
    plot(YPred{idx(i)},'.-')
    hold off
    ylim([0 thr + 25])
    title("Test Observation " + idx(i))
    xlabel("Time Step")
    ylabel("RUL")
end
legend(["Test Data"
"Predicted"],'Location','southeast')
```

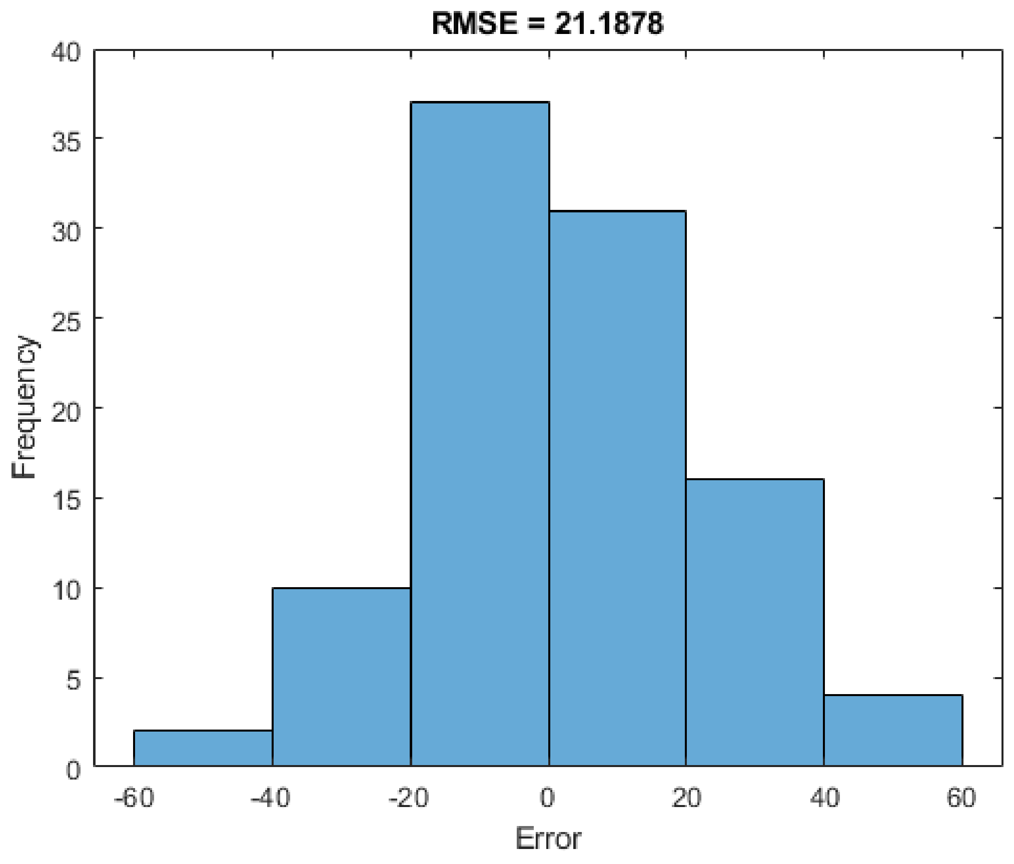


- For given partial sequence, predicted current RUL is last element of predicted sequences

Matlab LSTM Example: Remaining Useful Life …

- Calculate RMSE of predictions and visualize prediction error in a histogram

```
for i = 1:numel(YTest)
    YTestLast(i) = YTest{i}(end);
    YPredLast(i) = YPred{i}(end);
end
figure
rmse = sqrt(mean((YPredLast - YTestLast).^2))
rmse =
    21.1878
histogram(YPredLast - YTestLast)
title("RMSE = " + rmse)
ylabel("Frequency")
xlabel("Error")
```



Tensorflow RNN (LSTM) Case Studies in Python

WAYNE STATE UNIVERSITY

Time Series Forecasting (Univariate & Multivariate) using LSTMs: Weather Dataset

- This time series forecasting tutorial uses a weather time series dataset recorded by the Max Planck Institute for Biogeochemistry

Text Classification using LSTMs: IMDB Movie Review Sentiment

- This text classification tutorial trains an RNN (LSTM) on the IMDB large movie review dataset for sentiment analysis

Canvas: Implementing RNNs & LSTMs using TensorFlow & Keras in Python:

- **Time Series Forecasting (Univariate & Multivariate) using LSTMs: Weather Dataset**
  Python Jupyter Notebook Code | HTML Output
- **Text Classification using LSTMs: IMDB Movie Review Sentiment**
  Python Jupyter Notebook Code | HTML Output