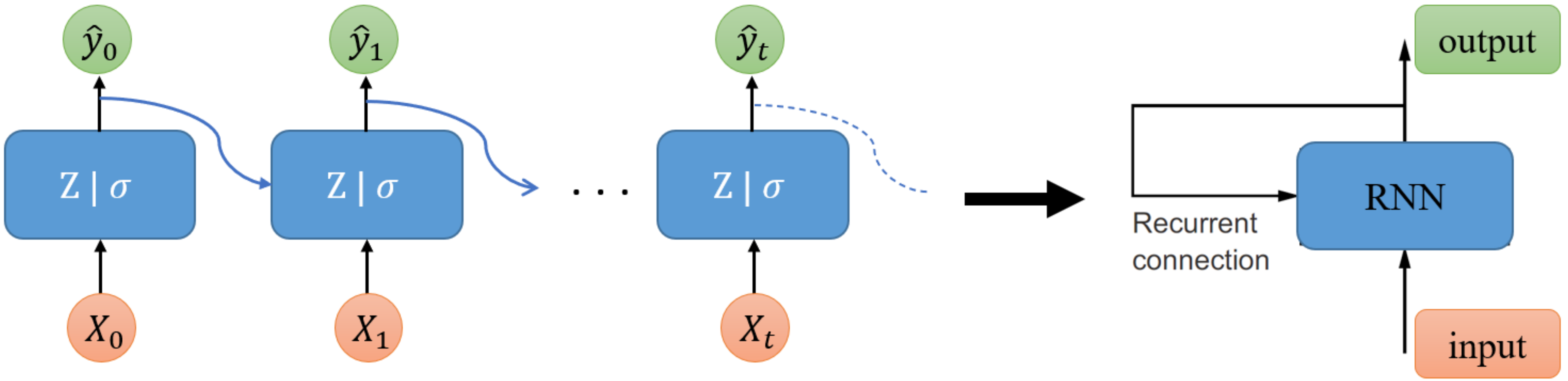
Module 7 – Part I
Deep Sequence Modeling
Recurrent Neural Networks (RNN)



JON M. HUNTSMAN SCHOOL OF BUSINESS
UtahStateUniversity

Pedram Jahangiry

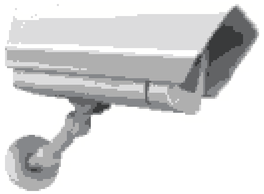
Road map!

- Module 1- Introduction to Deep Forecasting
- Module 2- Setting up Deep Forecasting Environment
- Module 3- Exponential Smoothing
- Module 4- ARIMA models
- Module 5- Machine Learning for Time series Forecasting
- Module 6- Deep Neural Networks
- **Module 7- Deep Sequence Modeling (RNN, LSTM)**
- Module 8- Transformers (Attention is all you need!)
- Module 9- Prophet and Neural Prophet

What is Sequence Data?

- Sequence data refers to any data that has a specific **order** or sequence to it!

- Sequence Data
  - Time Series Data
    - Regular TS
    - Irregular TS
  - Text Data
  - Audio Data
  - Video Data

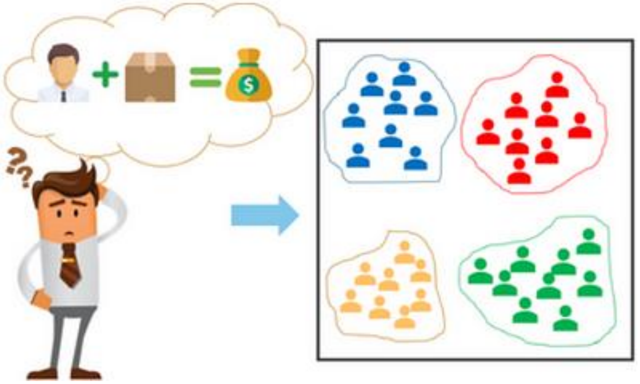


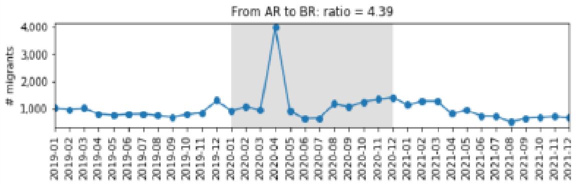
Time series Tasks

TS tasks

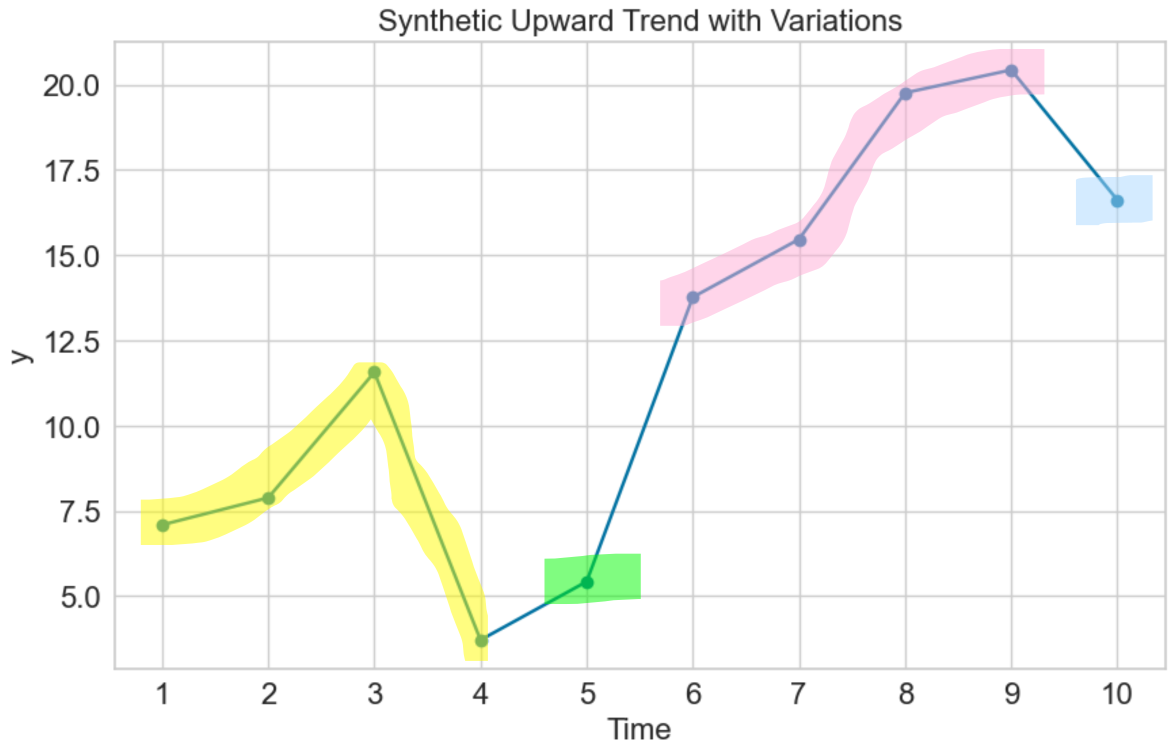
- Forecasting
  - Qualitative
  - Quantitative
- Classification
- Clustering
- Anomaly/Event Detection

WE ARE HERE





ML Data Transformation (Single-output)



Features (X)				Target (y)
y_{t-4}	y_{t-3}	y_{t-2}	y_{t-1}	y_t
y_1	y_2	y_3	y_4	y_5
y_2	y_3	y_4	y_5	y_6
y_3	y_4	y_5	y_6	y_7
y_4	y_5	y_6	y_7	y_8
y_5	y_6	y_7	y_8	y_9
y_6	y_7	y_8	y_9	y_{10}

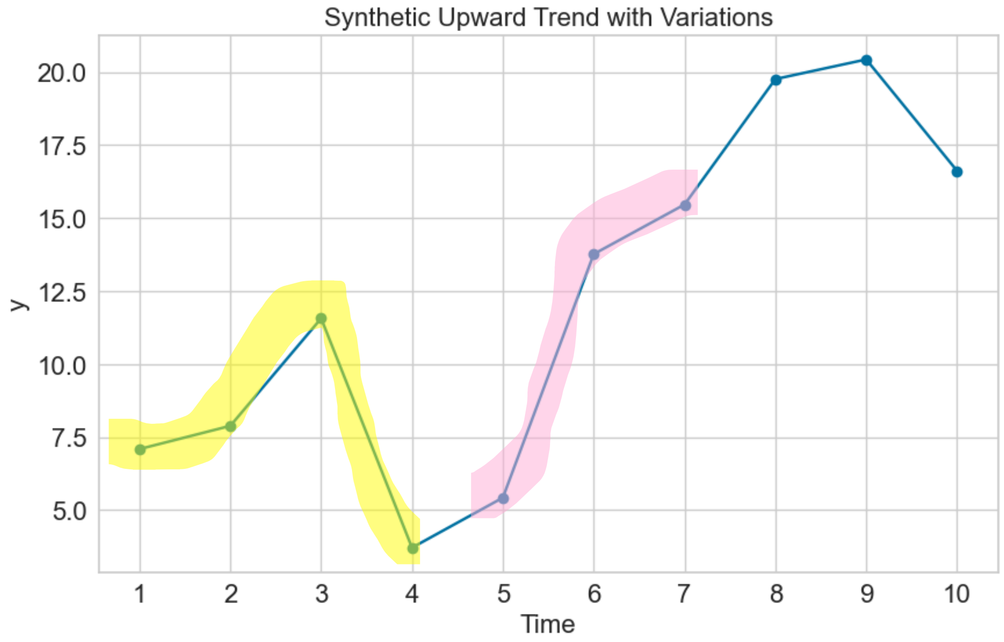
TS raw data → TS Supervised data

y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}

Jon M. Huntsman School of Business
UtahStateUniversity

Pedram Jahangiry

ML Data Transformation (Multi-output)

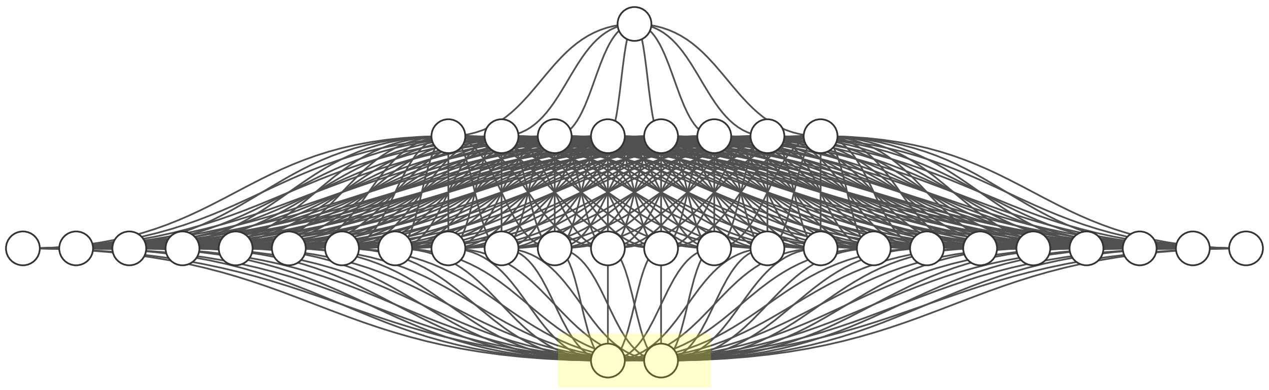


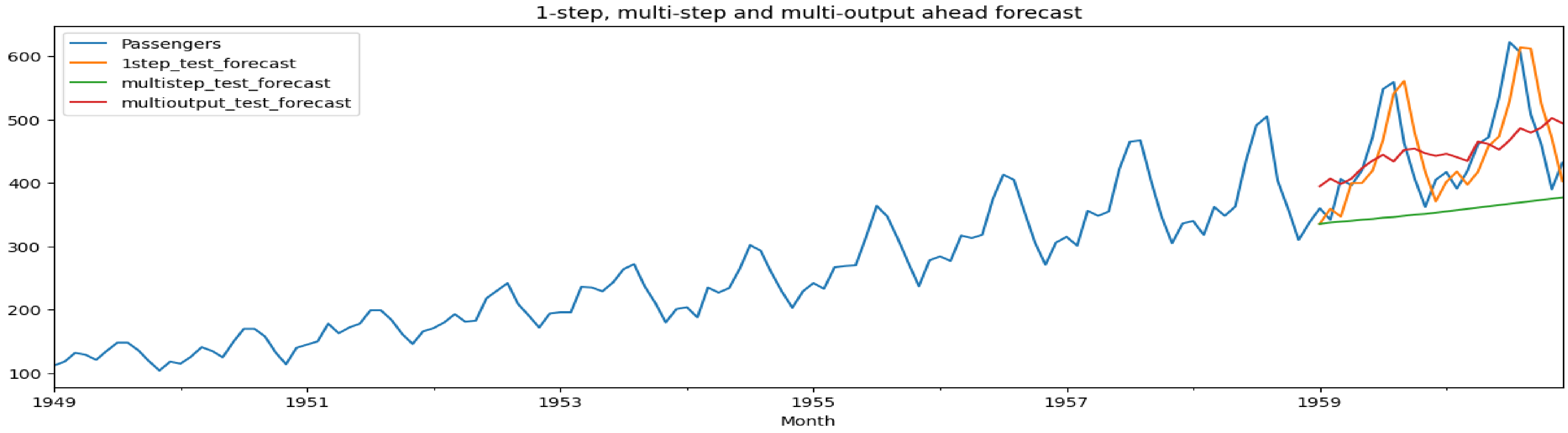
Features (X)				Target (y)		
y_{t-4}	y_{t-3}	y_{t-2}	y_{t-1}	y_t	y_{t+1}	y_{t+2}
y_1	y_2	y_3	y_4	y_5	y_6	y_7
y_2	y_3	y_4	y_5	y_6	y_7	y_8
y_3	y_4	y_5	y_6	y_7	y_8	y_9
y_4	y_5	y_6	y_7	y_8	y_9	y_{10}

TS raw data → TS Supervised data (h=3)

y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}

DL Example: DNN with 2 lags

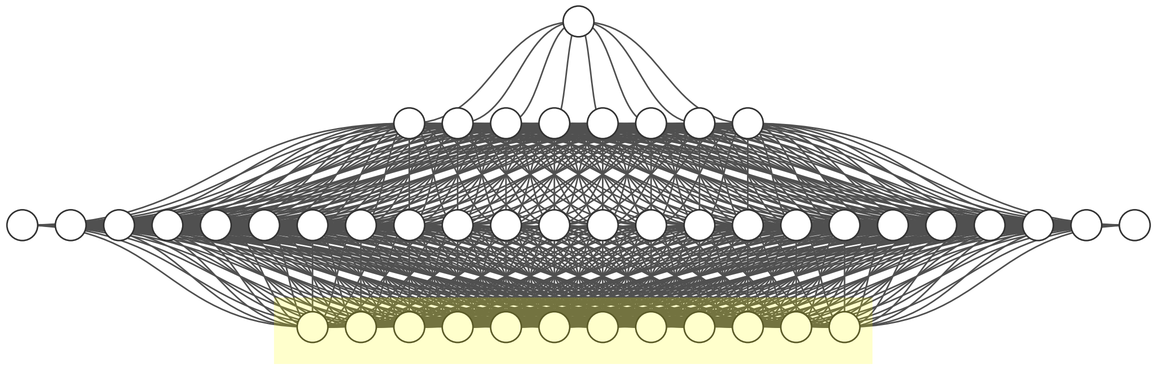


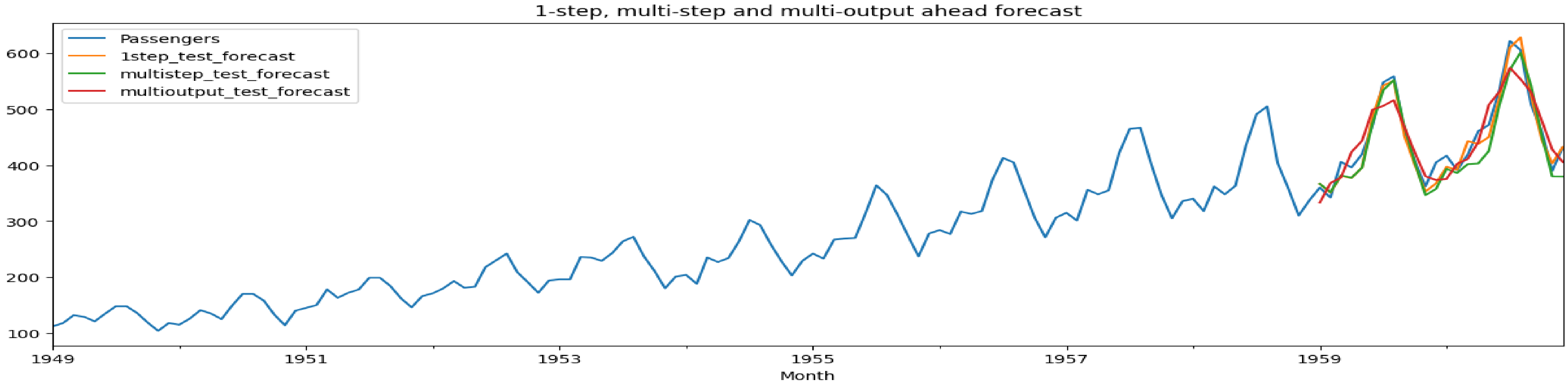


JON M HUNTSMAN SCHOOL OF BUSINESS
UtahStateUniversity

Pedram Jahangiry

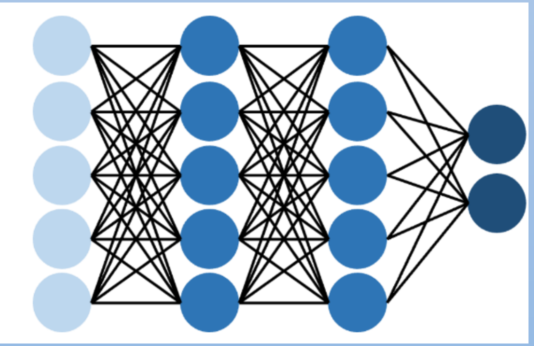
DL Example: DNN with 12 lags



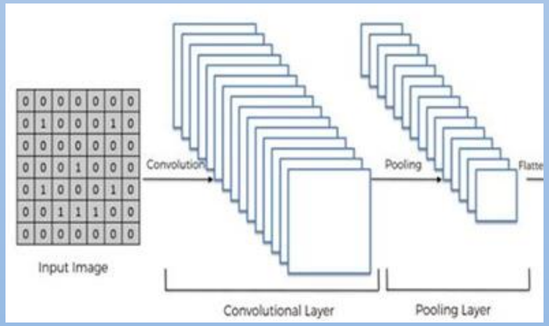


Deep Learning Architectures

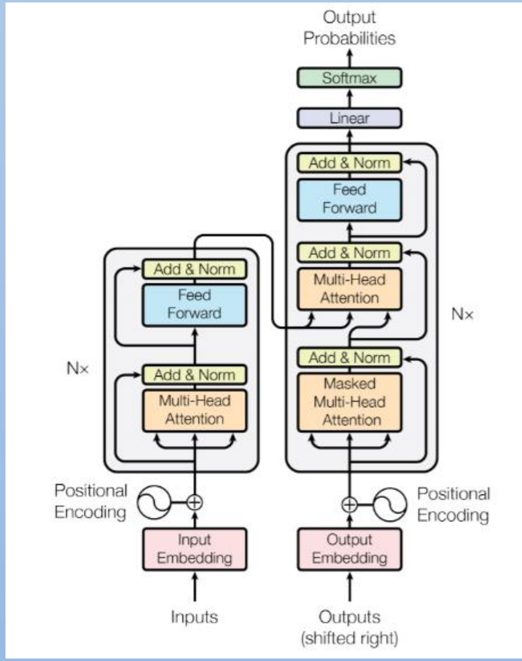
Standard NN



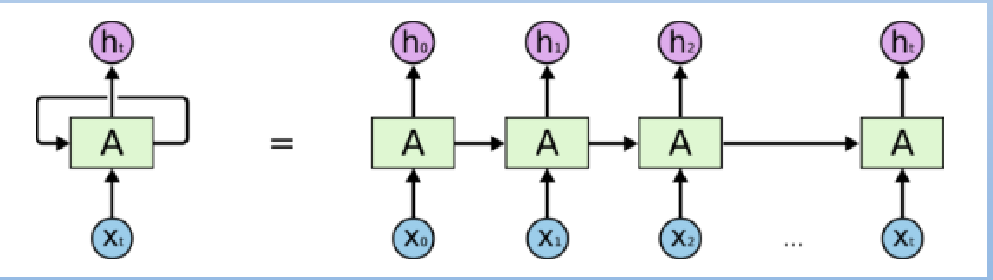
Convolutional NN



Transformers

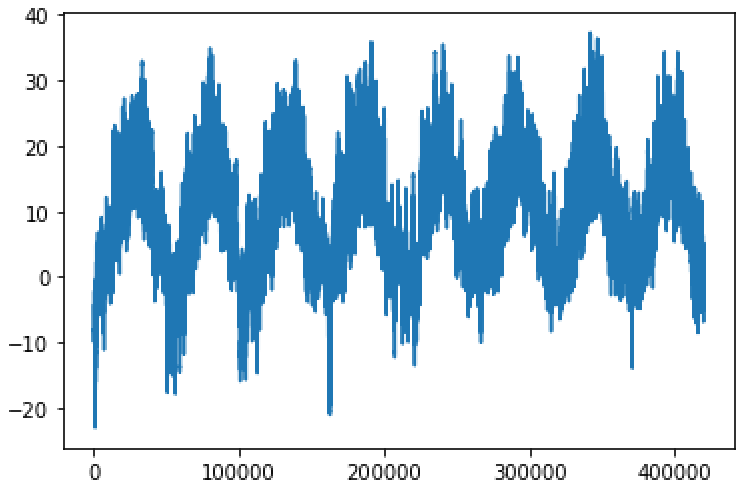


Recurrent NN

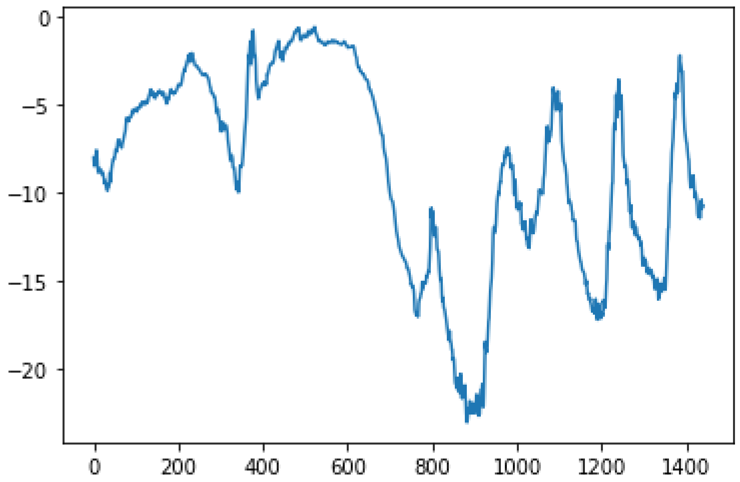


A simple timeseries example

- A temperature forecasting example: [deep-learning-with-python-notebooks]
- Predicting the temperature 24 hours in the future
  - Target: temperature
  - Features: 14 different variables including pressure, humidity, wind direction and etc
  - Data recorded every 10 minutes from 2009-2016



Temperature between 2009-2016



Temperature in the first 10 days: 10*24*6 = 1440

Preparing the data

- Given the previous 5 days (120 hours) and samples once per hour, can we predict temperature in 24 hours (after the end of the sequence)?
- Data batches:
  - Sequence length = 120
  - [1,2,3,…,120][144]
  - [2,3,4,…,121][145]
  - [3,4,5,…,122][146]
  - Bath size: 256 of these samples are shuffled and batched
  - Sample shape: (256, 120, 14)
  - Target shape: (256,)

Naïve forecaster: common-sense baseline

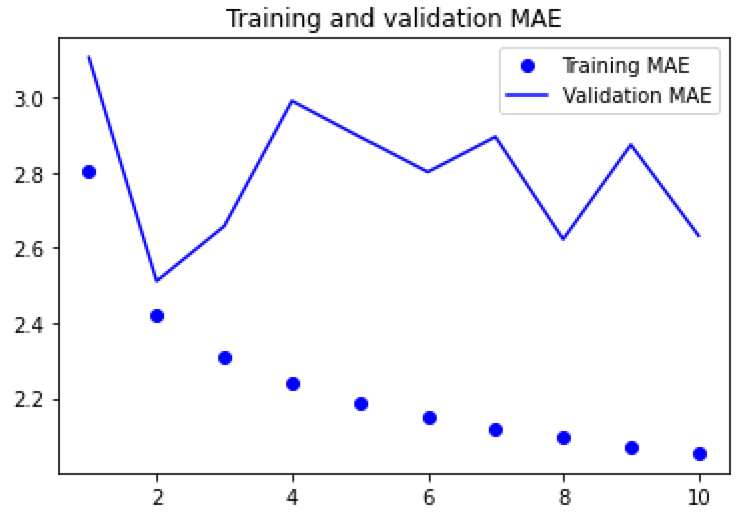
- Temperature 24 hours from now = Temperature right now
- This is our random walk with no drift forecaster.

- Performance:
  - Validation MAE = 2.44 degrees Celsius
  - Test MAE = 2.62 degrees Celsius
  - The baseline model is off by about 2.5 degrees on average. Not bad!!

Let's try DNN (Deep Neural Networks)

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Flatten()(inputs)
x = layers.Dense(16, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 120, 14)]         0

 flatten (Flatten)           (None, 1680)              0

 dense (Dense)               (None, 16)                26896

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 26,913
Trainable params: 26,913
Non-trainable params: 0
_________________________________________________________________
```



- Test MAE = 2.62 degrees Celsius
- No improvement!!
- Flattening a timeseries data is not a good idea!

Let's try CNN (Convolutional Neural Networks)

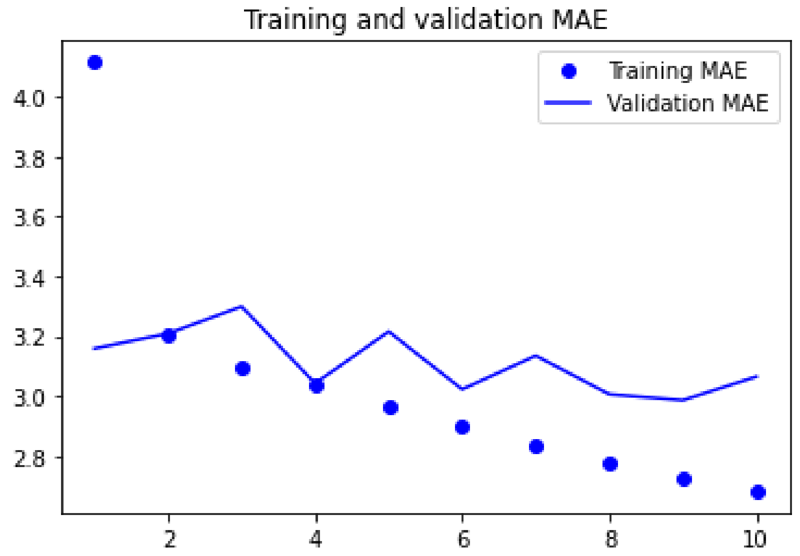
- Motivation: Maybe a temporal convnet could reuse the same representations across different days, much like a spatial convnet can reuse the same representations across different locations in an image!

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Conv1D(8, 24, activation="relu")(inputs)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 12, activation="relu")(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 6, activation="relu")(x)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 120, 14)]         0

 conv1d (Conv1D)             (None, 97, 8)             2696

 max_pooling1d (MaxPooling1D  (None, 48, 8)            0
 )

 conv1d_1 (Conv1D)           (None, 37, 8)             776

 max_pooling1d_1 (MaxPooling  (None, 18, 8)            0
 1D)

 conv1d_2 (Conv1D)           (None, 13, 8)             392

 global_average_pooling1d (G  (None, 8)                0
 lobalAveragePooling1D)

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 3,873
Trainable params: 3,873
Non-trainable params: 0
_________________________________________________________________
```

CNN performance

- Test MAE = 3.10 degrees Celsius
- Even worse than the densely connected model!!
  - CNN treats every segment of the data the same way!
  - Pooling layers are destroying order information.



JON M. HUNTSMAN SCHOOL OF BUSINESS
UtahStateUniversity

Pedram Jahangiry

Sequence Modeling

To model sequence data efficiently, we need a new architecture that:

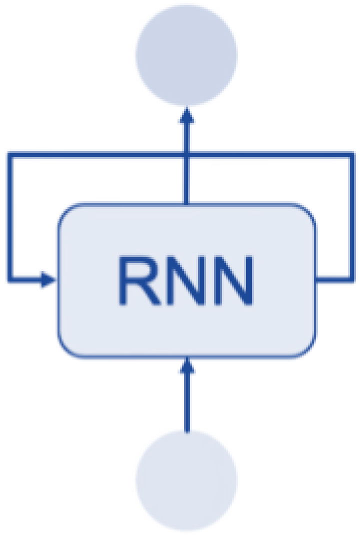
- Preserve the order
- Account for long-term dependencies
- Handle input-length
- Share parameters across the sequence

JON M. HUNTSMAN SCHOOL OF BUSINESS
UtahStateUniversity

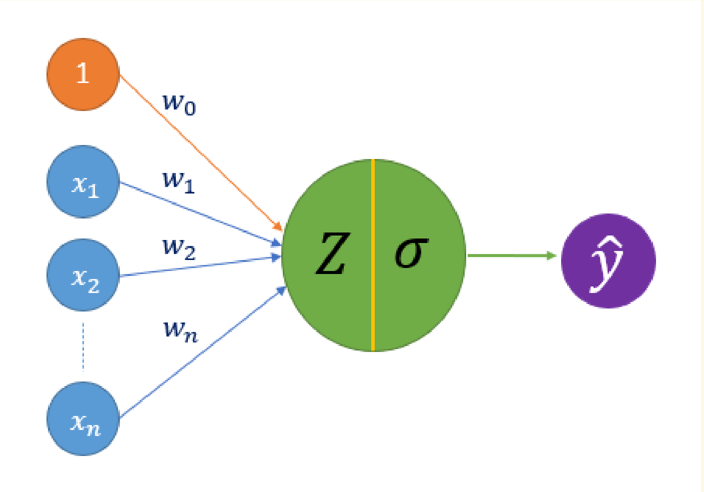
Pedram Jahangiry

What is RNN (Recurrent Neural Network)?

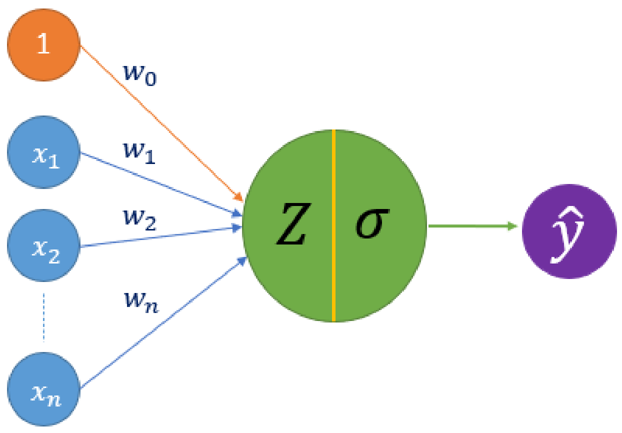
- The architecture of RNNs is inspired by the way biological intelligence processes information incrementally while maintaining an internal model of what it is processing.
- This ability to remember previous inputs and incorporate them into the current output allows RNNs to model sequential data.
- RNN maintains a state that contains information relative to what it has seen so far
- RNNs can be thought of as neural networks with an internal loop, which allows them to process sequences of varying lengths and learn from temporal dependencies.

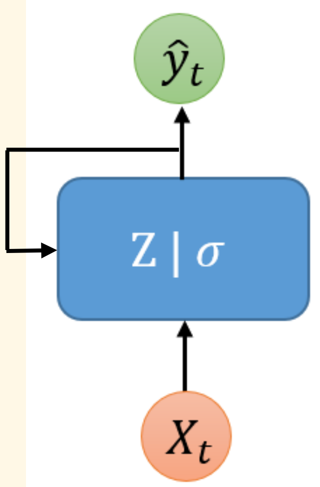


Perceptron vs Recurrent Cell



Perceptron



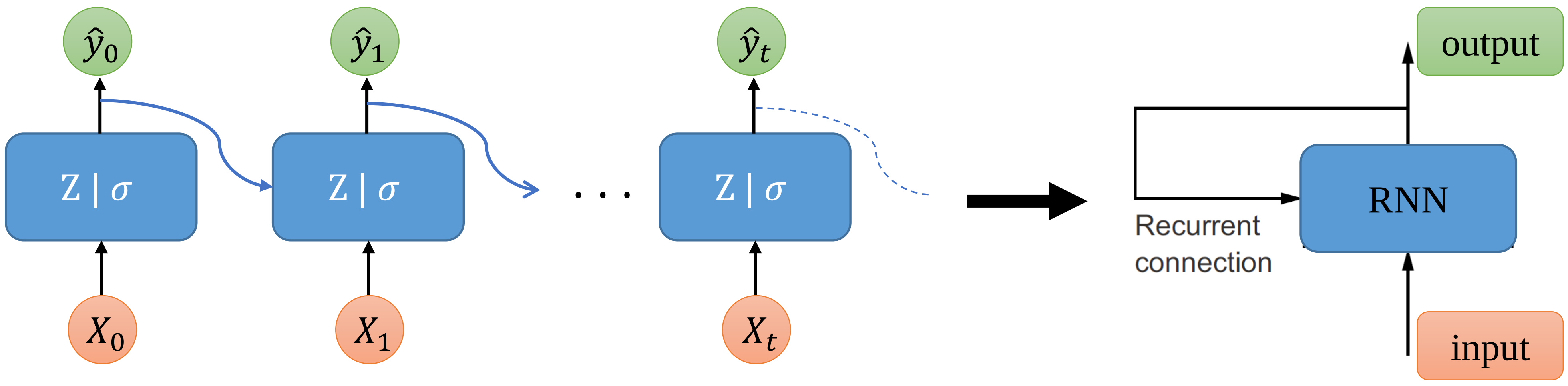


Recurrent Cell

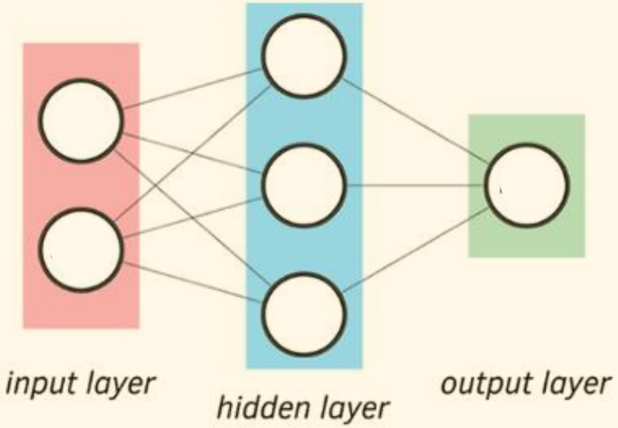
Jon M. Huntsman School of Business
Utah State University

Pedram Jahangiry

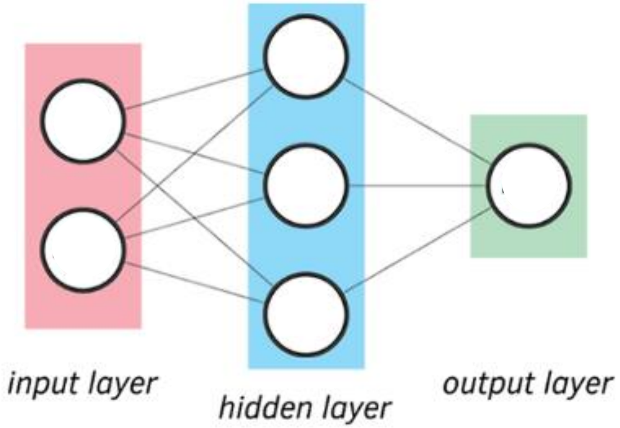
Unrolling the Recurrent Cell

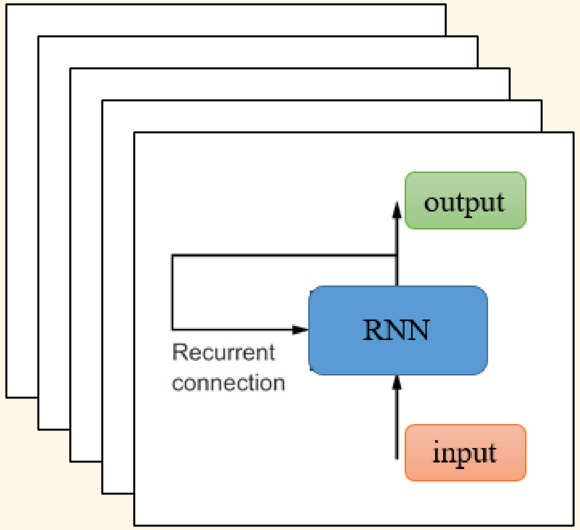


Dense Layer vs Recurrent Layer



Dense Layer





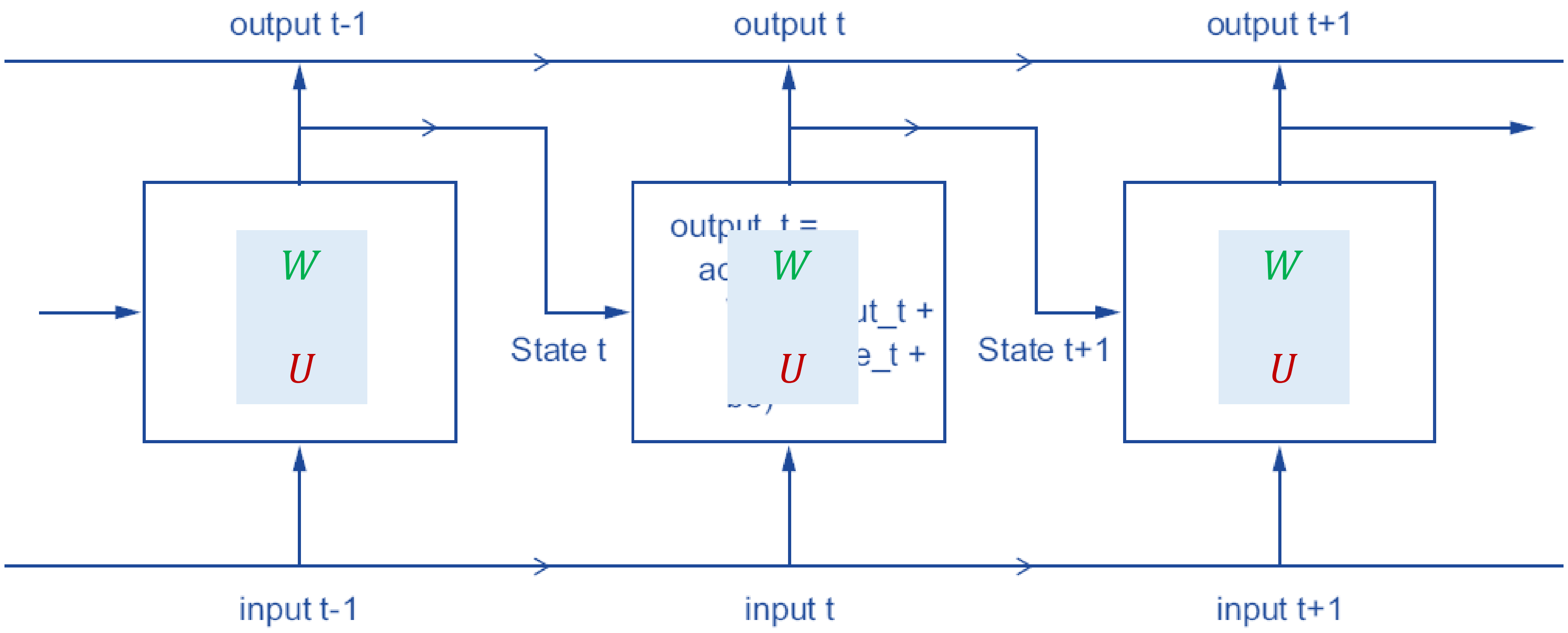
Recurrent Layer

Jon M. Huntsman School of Business Utah State University

Pedram Jahangiry

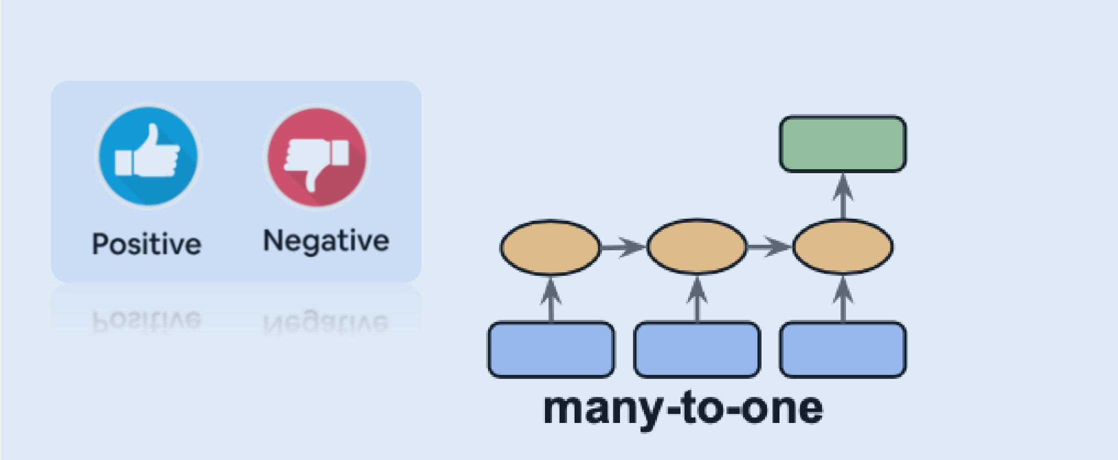
Inside the Recurrent Cell

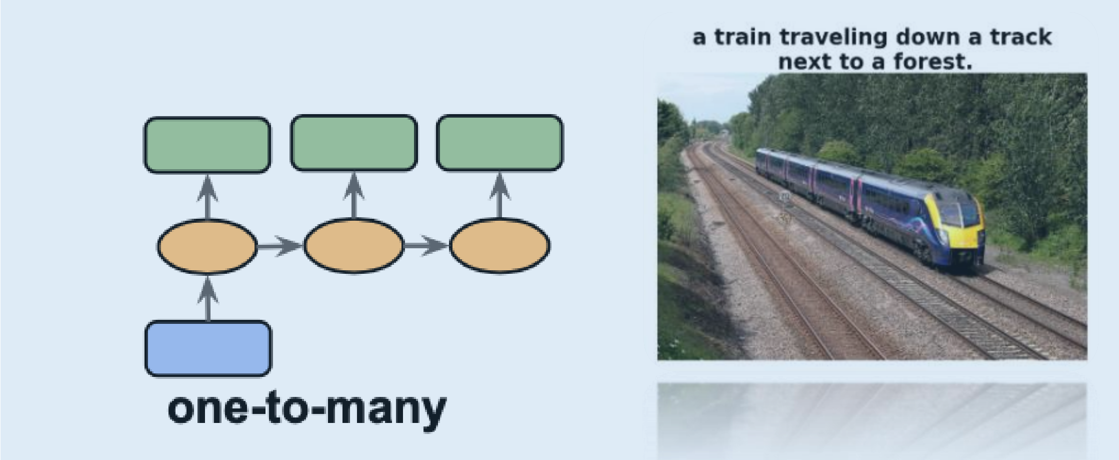
$$output_t = f(input_t, State_t)$$

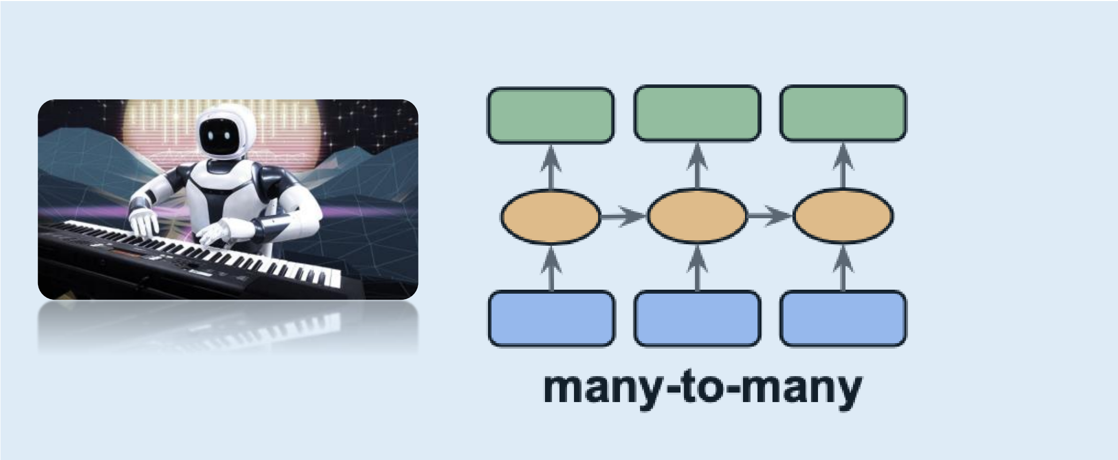


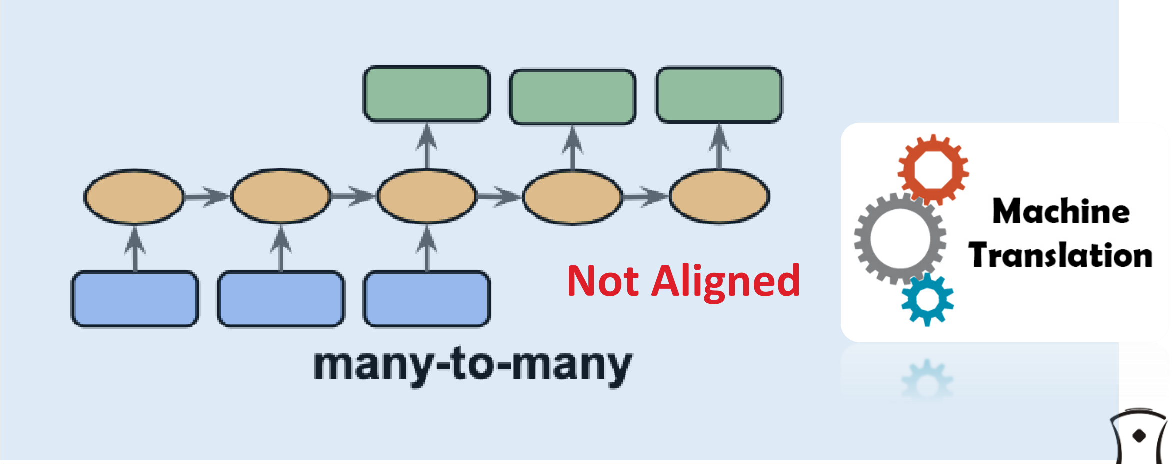
$$s_{t+1} = activation(WX_t + Us_t + b)$$

RNN architectures



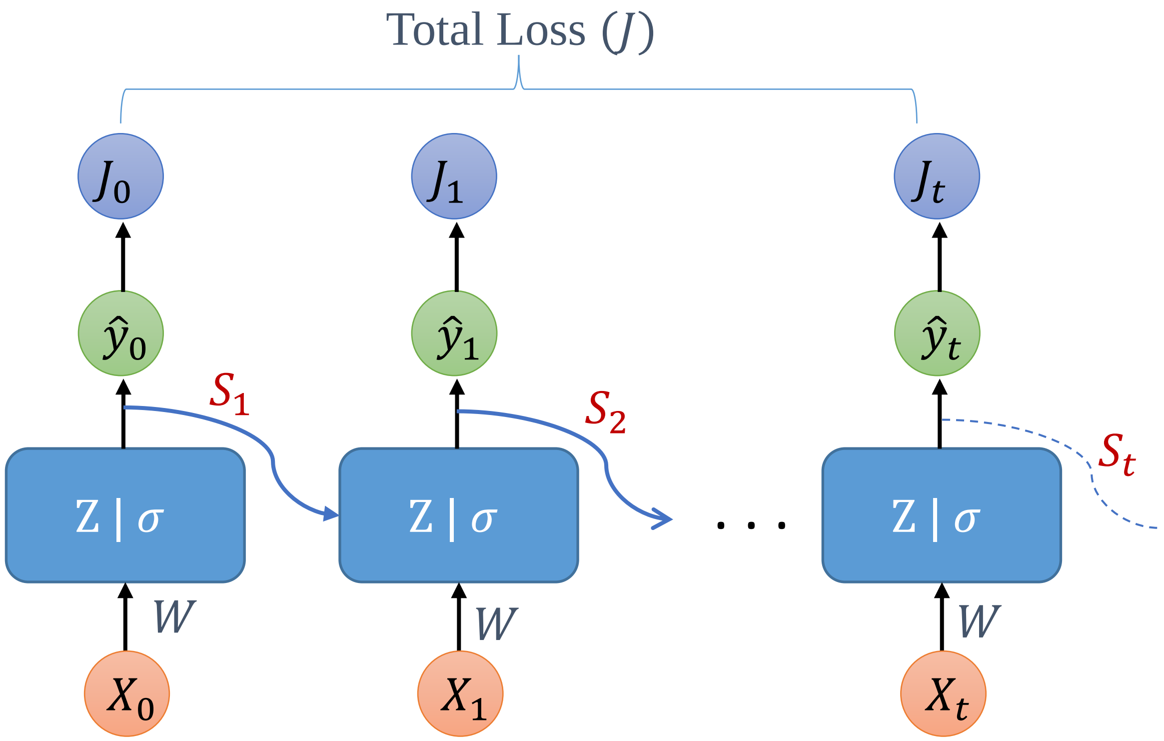






How does RNN learn representations?

- Backpropagation Through Time (BPTT)
- $\frac{\partial J}{\partial P}$ P are the parameters
- $\frac{\partial J}{\partial W} = \frac{\partial J_0}{\partial W} + \frac{\partial J_1}{\partial W} + ..$
- $\frac{\partial J_0}{\partial W} = \frac{\partial J_0}{\partial y_0}\frac{\partial y_0}{\partial S_0}\frac{\partial S_0}{\partial W}$
- $\frac{\partial J_1}{\partial W} = \frac{\partial J_1}{\partial y_1}\frac{\partial y_1}{\partial S_1}\frac{\partial S_1}{\partial W}$, $\frac{\partial S_1}{\partial W} = \frac{\partial S_1}{\partial S_0}\frac{\partial S_0}{\partial W}$
- …
- $\frac{\partial J_t}{\partial W} = \sum_{k=0}^{t}\frac{\partial J_t}{\partial y_t}\frac{\partial y_t}{\partial S_t}\frac{\partial S_t}{\partial S_k}\frac{\partial S_k}{\partial W}$

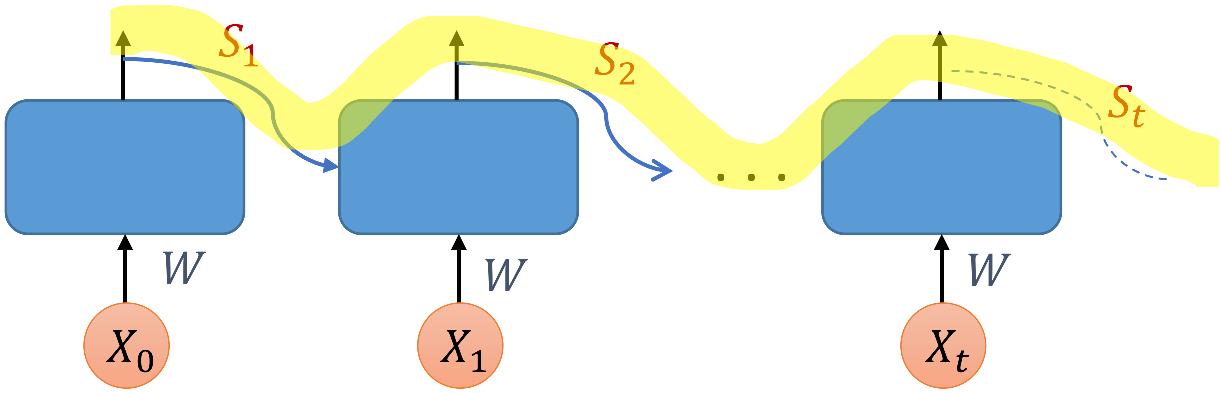


Vanishing Gradient Problem

- As the time horizon gets bigger, this product gets longer and longer.
- We are multiplying a lot of small numbers → smaller gradients → biased parameters unable to capture long term dependencies.

- $\frac{\partial J_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial J_t}{\partial y_t} \frac{\partial y_t}{\partial S_t} \frac{\partial S_t}{\partial S_k} \frac{\partial S_k}{\partial W}$
- $\frac{\partial S_{10}}{\partial S_0} = \frac{\partial S_{10}}{\partial S_9} \frac{\partial S_9}{\partial S_8} \frac{\partial S_8}{\partial S_7} \frac{\partial S_7}{\partial S_6} \dots \frac{\partial S_1}{\partial S_0}$

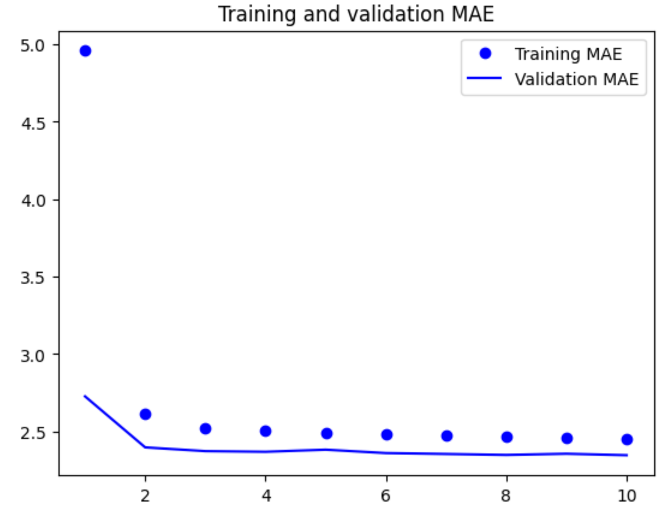
$$S_t = activation(W X_{t\_1} + U s_{t-1})$$



Let's try a simple RNN

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.SimpleRNN(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 120, 14)]         0

 simple_rnn (SimpleRNN)      (None, 16)                496

 dense_3 (Dense)             (None, 1)                 17

=================================================================
Total params: 513 (2.00 KB)
Trainable params: 513 (2.00 KB)
Non-trainable params: 0 (0.00 Byte)
```

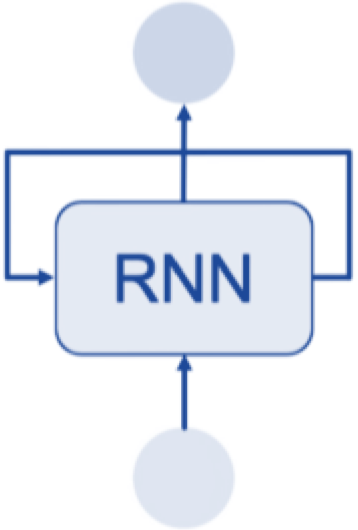


- Baseline Test MAE = 2.62
- Simple RNN Test MAE = 2.51
- beats the naïve forecaster.

Beyond RNN

RNN can handle the following sequence modeling criteria:

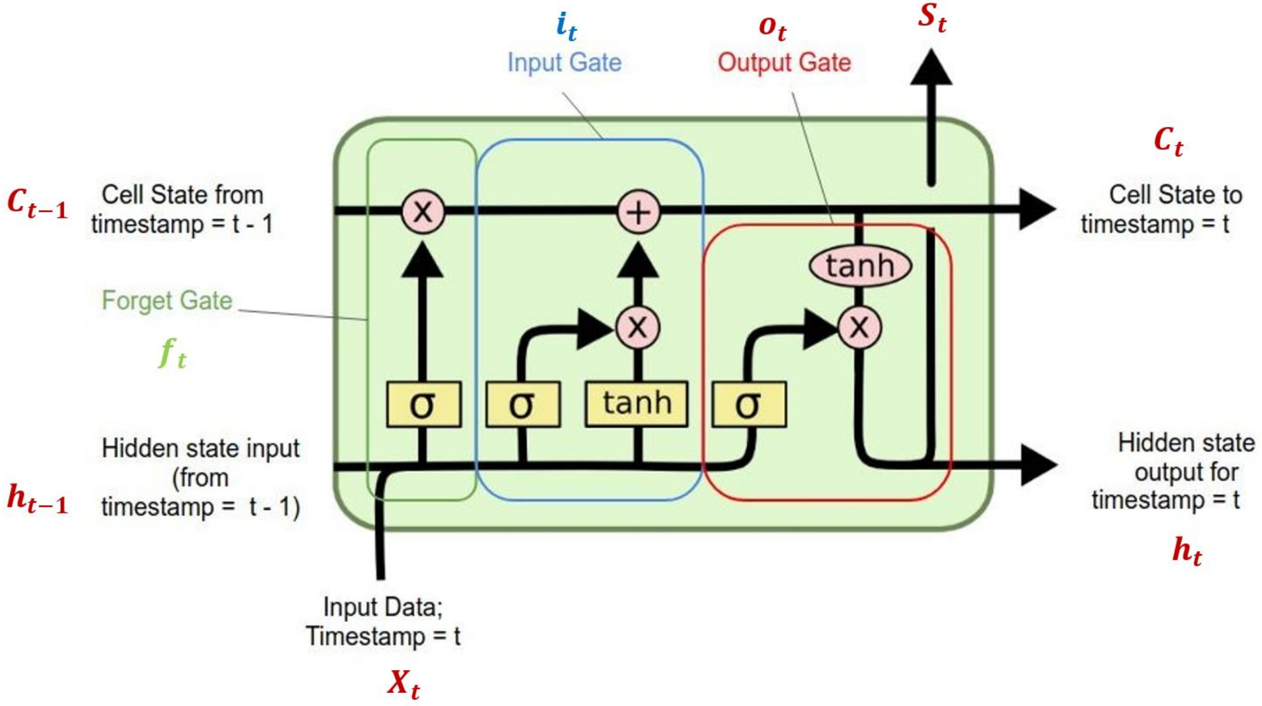
- Preserve the order
- Handle input-length
- Share parameters across the sequence



RNN limitations:

- Does not account for long-term dependencies (only remember short term history)
- Vanishing Gradient Problem

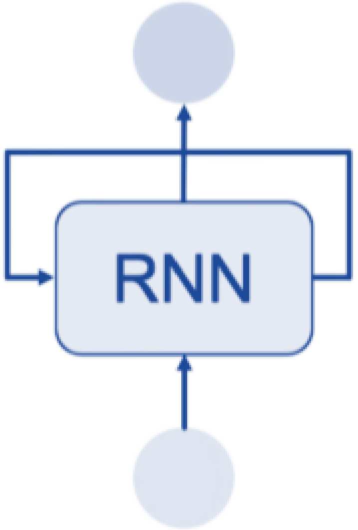
Module 7 – Part II
Deep Sequence Modeling
(Gated cells, LSTM)



Beyond RNN

RNN can handle the following sequence modeling criteria:

- Preserve the order
- Handle input-length
- Share parameters across the sequence

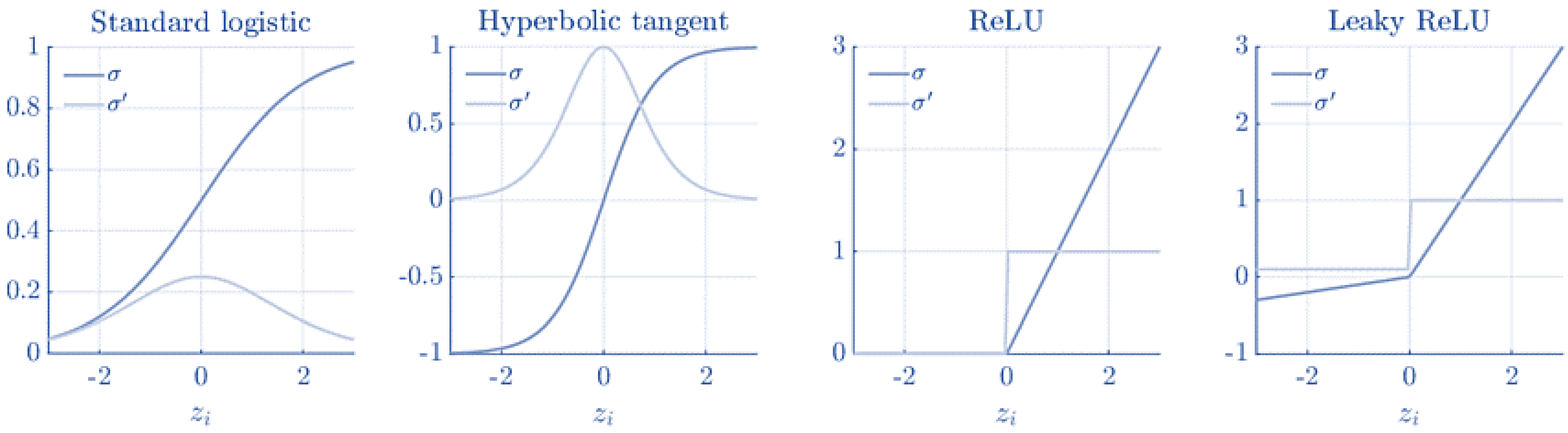


RNN limitations:

- Does not account for long-term dependencies (only remember short term history)
- Vanishing Gradient Problem

How to solve vanishing gradient problem

1. Use Activation Function that prevents fast shrinkage of gradient



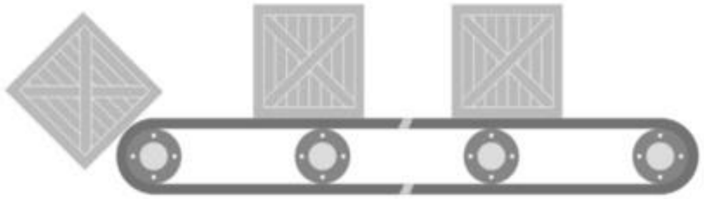
$$S_t = activation(WX_{t-1} + Us_{t-1})$$

How to solve vanishing gradient problem

1. Use Activation Function that prevents fast shrinkage of gradient
2. Use weight initialization techniques that ensure that the initial weights are not too small
3. Use gradient clipping which limits the magnitude of the gradients from becoming too small (vanishing gradient) or too large (exploding gradient)
4. Use batch normalization, which normalizes the input to each layer and helps to reduce the range of activation values and thus the likelihood of vanishing gradients.
5. Use a different optimization algorithm that is more resilient to vanishing gradients, such as Adam or RMSprop.
6. **Gated cells:** Use some sort of **skip connections**, which allow gradients to bypass some of the layers in the network and thus prevent them from becoming too small.

Gated cells

- Instead of using a simple RNN cell, let's use a more complex cell with gates which control the flow of information.
- Think of a conveyer belt running parallel to the sequence being processed:
  - Information can jump on → transported to a later timestep → jump off when needed.
  - This is what a gated cell does! Analogous to residual connections we saw before.

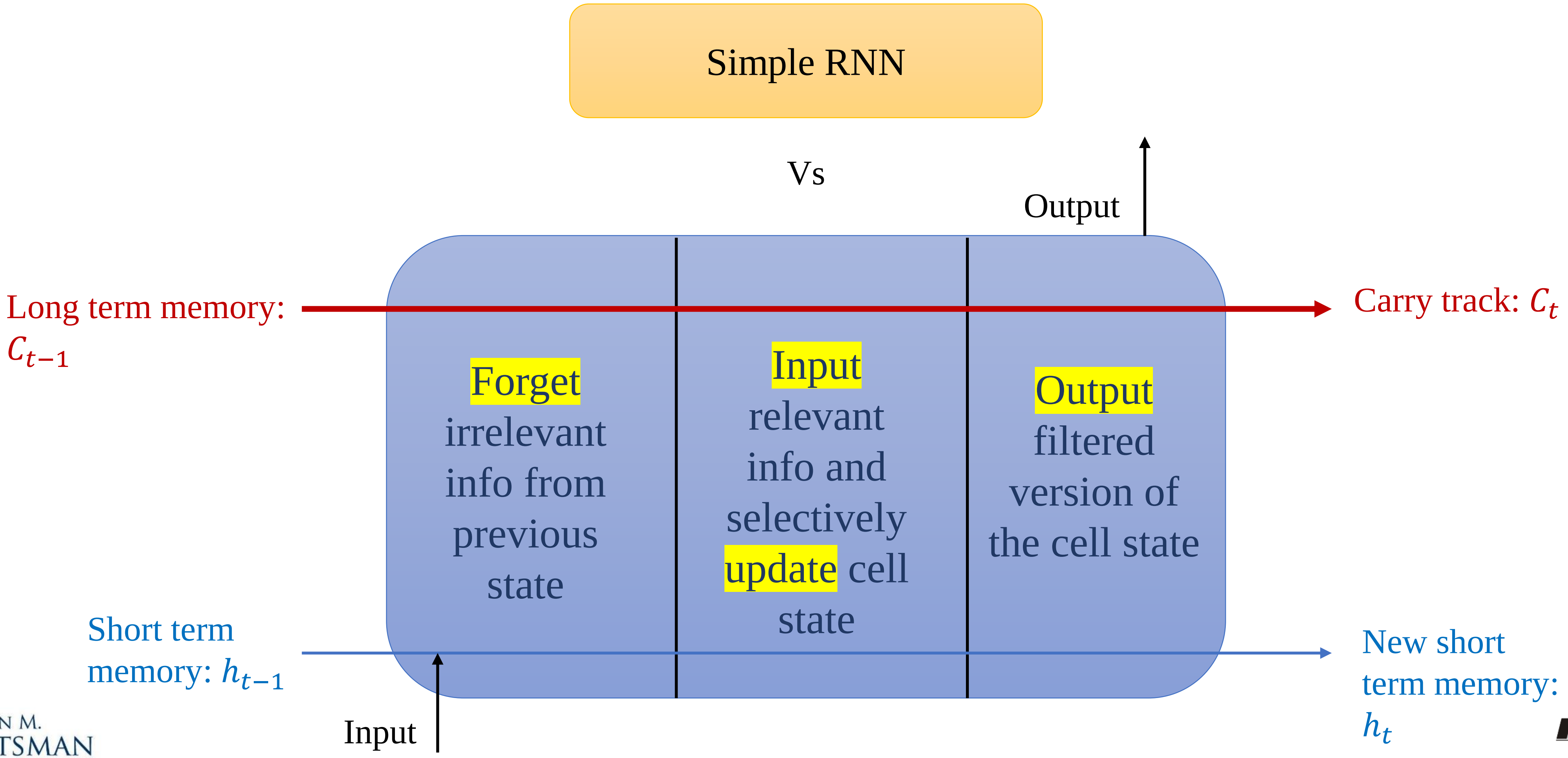


- Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are two examples of gated cells that can keep track of information throughout many timesteps.

Jon M. Huntsman School of Business
UtahStateUniversity

Pedram Jahangiry

Inside the LSTM cell

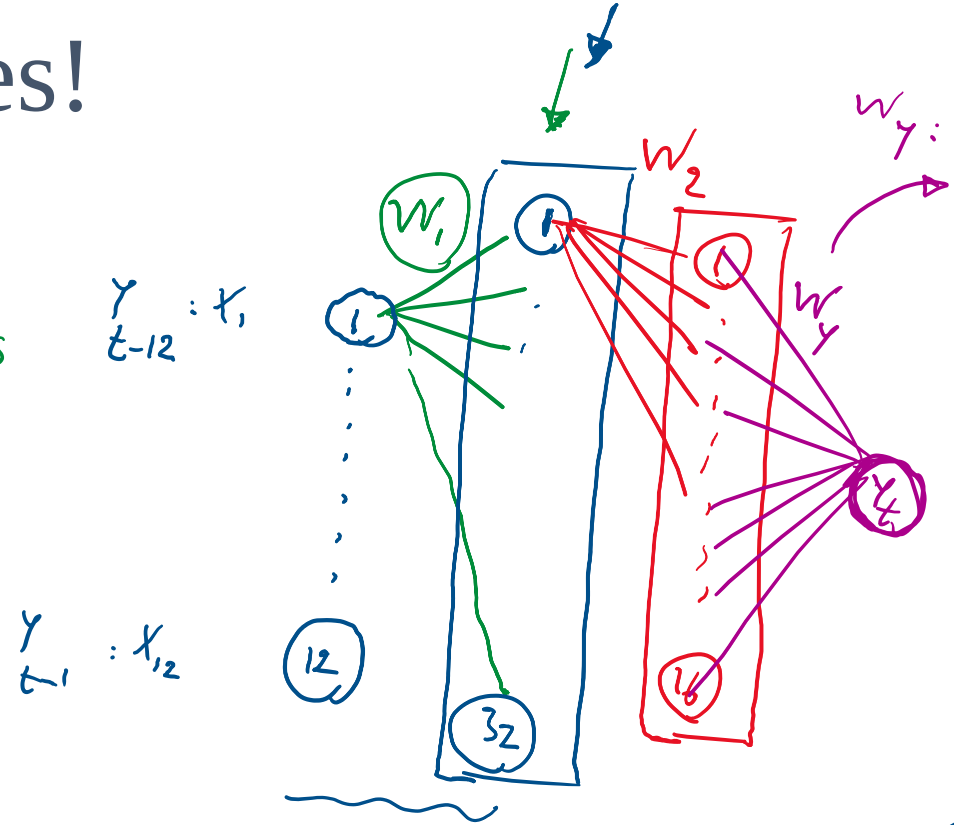


All about shapes!

DNN

univariate ts with 1000 hours

X: 12 lags



$W_y : (16\times1)$ $b_y : 1\times1$

$W_2 : 32\times16$

$b_2 : 16\times1$

$$\underbrace{Y_t = \sigma(W_y^T \cdot output_2 + b_y)}_{(1\times16)(16\times1) + 1\times1} \quad 1\times1$$

$X : (12\times1)$

$W_1 : (12\times32)$ ←

$b_1 : (32\times1)$

$$output_1 = \sigma(W_1^T \cdot X + b_1) \quad : \quad 32\times1$$

$$\underbrace{(32\times12)(12\times1) + (32\times1)}$$

$$output_2 = \sigma(W_2^T \cdot output_1 + b_2) \rightarrow 16\times1$$

$$(16\times32)(32\times1) + 16\times1$$

All about shapes!

· RNN: Seq-length = 12 → Memory lenght

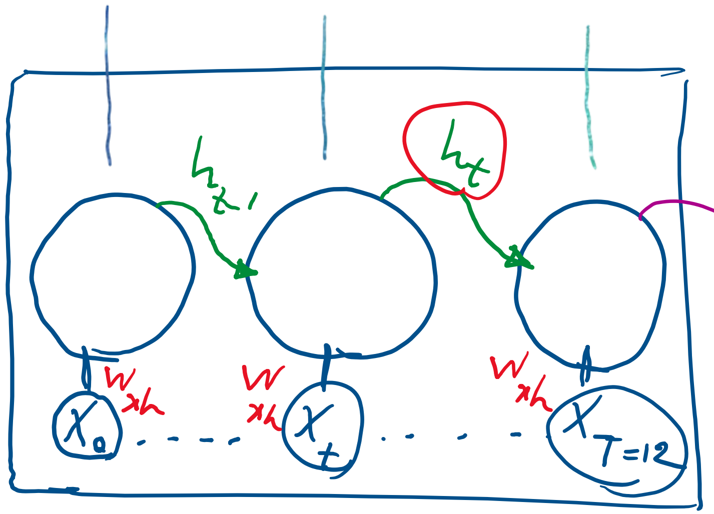
1 SimpleRNN(16) → Memory depth

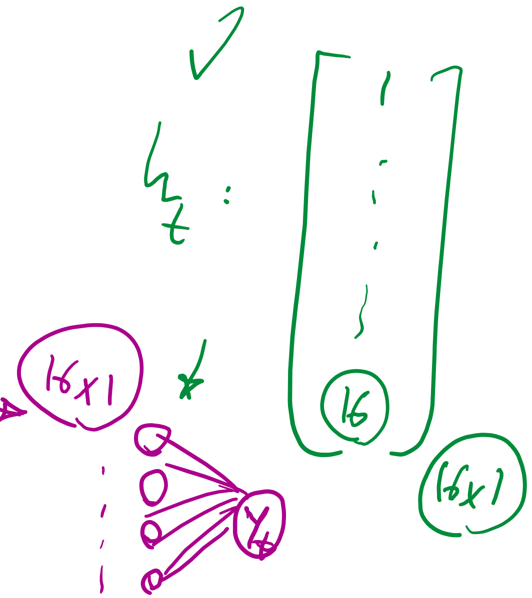
↓

hidden state dim

$h_t = 16$

RNN_1 : (None, 16)





$$h_t = \sigma(\underbrace{W_{xh}^T}_{16\times1} \cdot \underbrace{X_t}_{1\times1} + \underbrace{W_{hh}}_{16\times16} \underbrace{h_{t-1}}_{16\times1} + \underbrace{b}_{16\times1})$$

to h_t

from h_{t-1}

1 — 16

2

3

16

RNN_1 : (None, 12, 16)

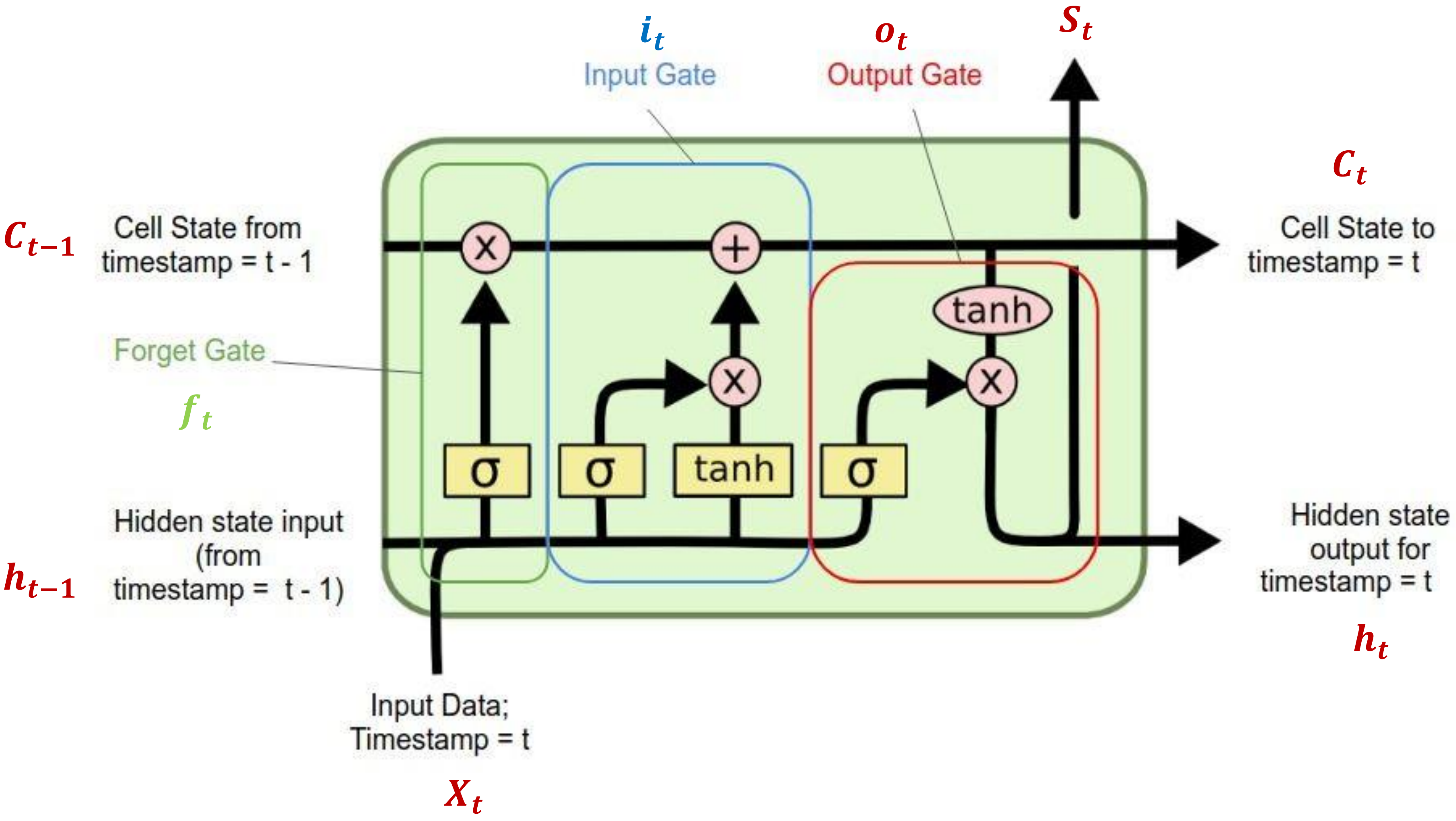
RNN_2 : (None, 32)

32

16x1

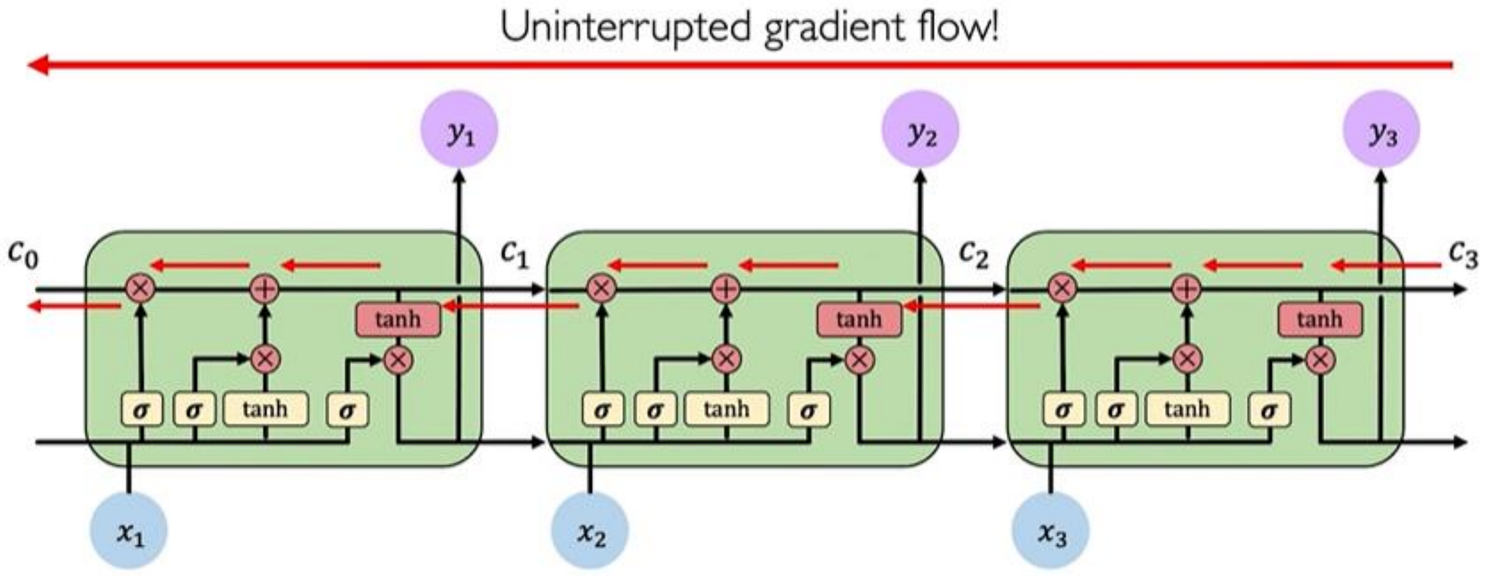
32x1

LSTM details



LSTM takeaway

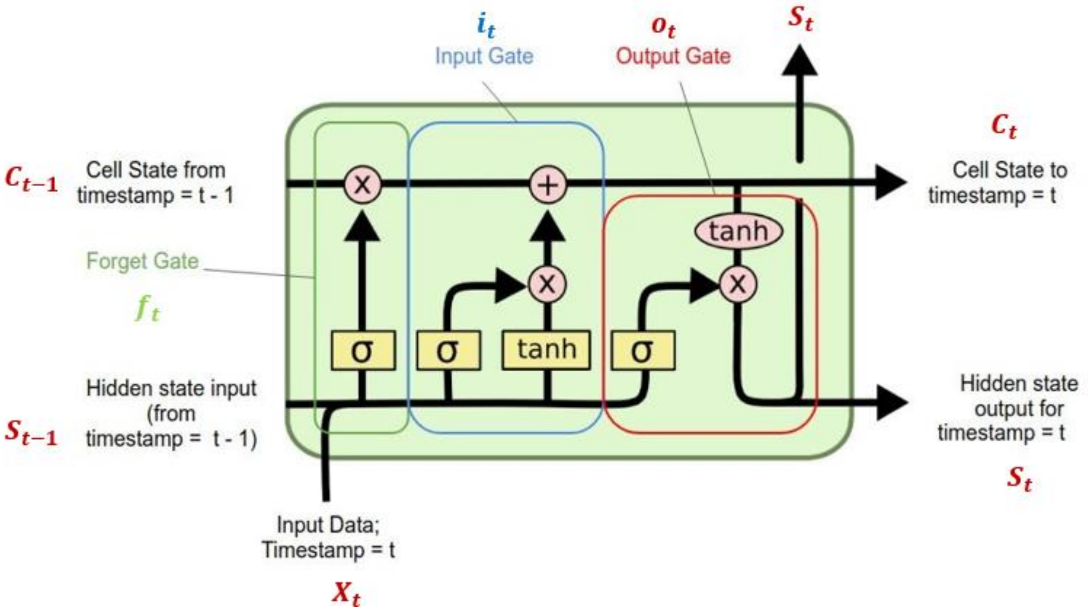
- LSTM uses gates to regulate the information flow (allows past information to be reinjected later)
- This new cell state (carry) can better capture longer term dependencies
- LSTM fights the vanishing gradient problem



Let’s try LSTM on the temperature example

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

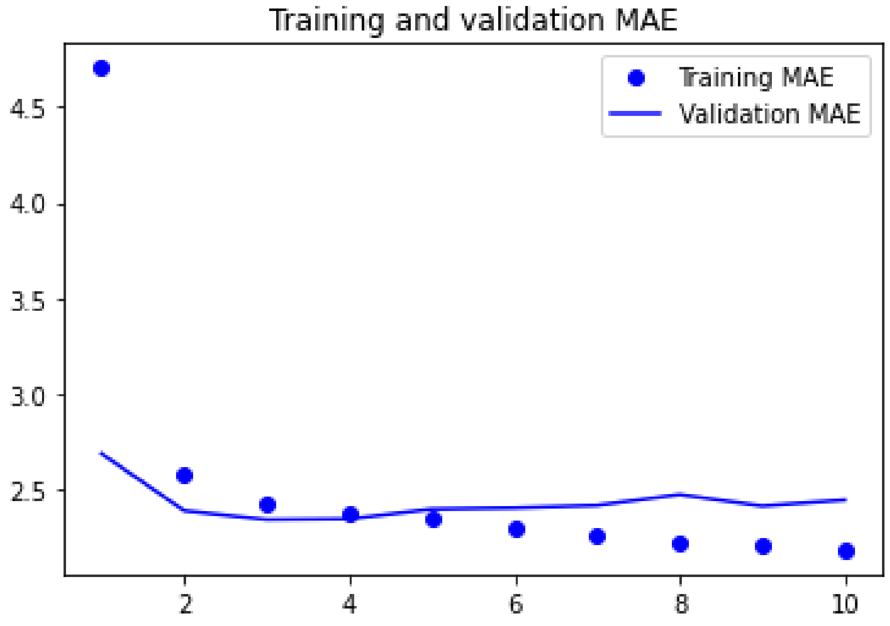
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 120, 14)]         0

 lstm (LSTM)                 (None, 16)                1984

 dense_3 (Dense)             (None, 1)                 17

=================================================================
Total params: 2,001
Trainable params: 2,001
Non-trainable params: 0
_________________________________________________________________
```




```
output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + dot(c_t, Vo) + bo)
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
```

LSTM performance

- Baseline Test MAE = 2.62
- Simple LSTM Test MAE = 2.53
- Also beats the naïve forecaster.
- Overfitting?



Can we do better?

Improving the simple LSTM model

- We can improve the performance of the simple LSTM model by:

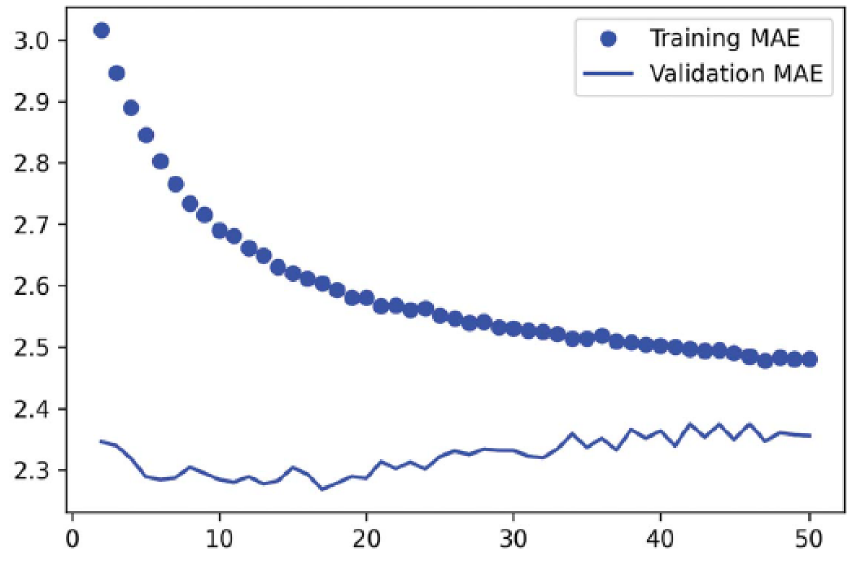
1. Recurrent Dropout : use drop out to fight overfitting in the recurrent layers (in addition to drop out for the dense layers)
2. Stacking recurrent layers: increase model complexity to boost representation power
3. Using bidirectional RNN: processing the same information differently! Mostly used in NLP.

Recurrent Drop out

- The same dropout pattern should be applied at every timestep

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(32, recurrent_dropout=0.25)(inputs)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

- Baseline Test MAE = 2.62
- Simple RNN, Test MAE = 2.51
- Simple LSTM, Test MAE = 2.53
- LSTM with dropout, Test MAE = 2.45



Jon M. Huntsman School of Business
Utah State University

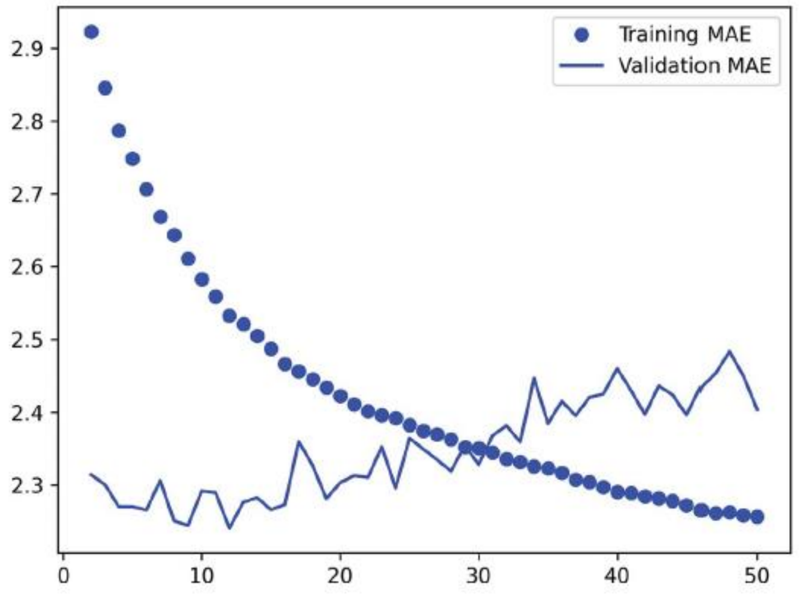
Pedram Jahangiry

Stacking Recurrent Layers

- Let's train a dropout-regulated, stacked GRU model.
- GRU is a slightly simpler version (hence, faster) of LSTM architecture

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.GRU(32, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

- Baseline Test MAE = 2.62
- Simple RNN, Test MAE = 2.51
- Simple LSTM, Test MAE = 2.53
- Stacking GRU, Test MAE = 2.39

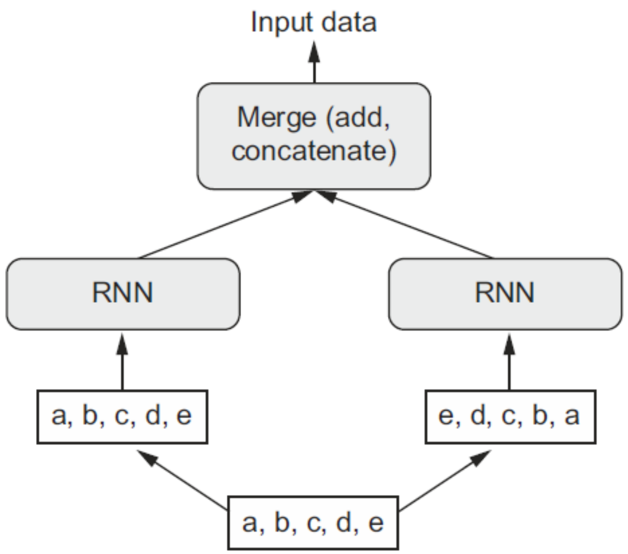


Bidirectional RNN

- Bidirectional RNN process the input sequence both chronologically and antichronologically.
- Idea: capturing patterns (representations) that might be overlooked by a unidirectional RNN.

- For the temperature example, the bidirectional LSTM strongly underperforms even the common-sense baseline.

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Bidirectional(layers.LSTM(16))(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```



Final message

- Deep learning is more an art than science! Too many moving part!
  - Number of units in each recurrent layer
  - Number of stacked layers
  - Amount of dropout and recurrent dropout
  - Number of dense layers
  - Sequence horizon!
  - Optimizers, learning rates and etc
  - ….

- Apply RNN to datasets that past is a good predictor of the future! Not the stock market!

Jon M. Huntsman School of Business
UtahStateUniversity

Pedram Jahangiry

Road map!

- ✓ Module 1- Introduction to Deep Forecasting
- ✓ Module 2- Setting up Deep Forecasting Environment
- ✓ Module 3- Exponential Smoothing
- ✓ Module 4- ARIMA models
- ✓ Module 5- Machine Learning for Time series Forecasting
- ✓ Module 6- Deep Neural Networks
- ✓ Module 7- Deep Sequence Modeling (RNN, LSTM)
- Module 8- Transformers (Attention is all you need!)
- Module 9- Prophet and Neural Prophet