Applied Text Analytics & Natural Language Processing

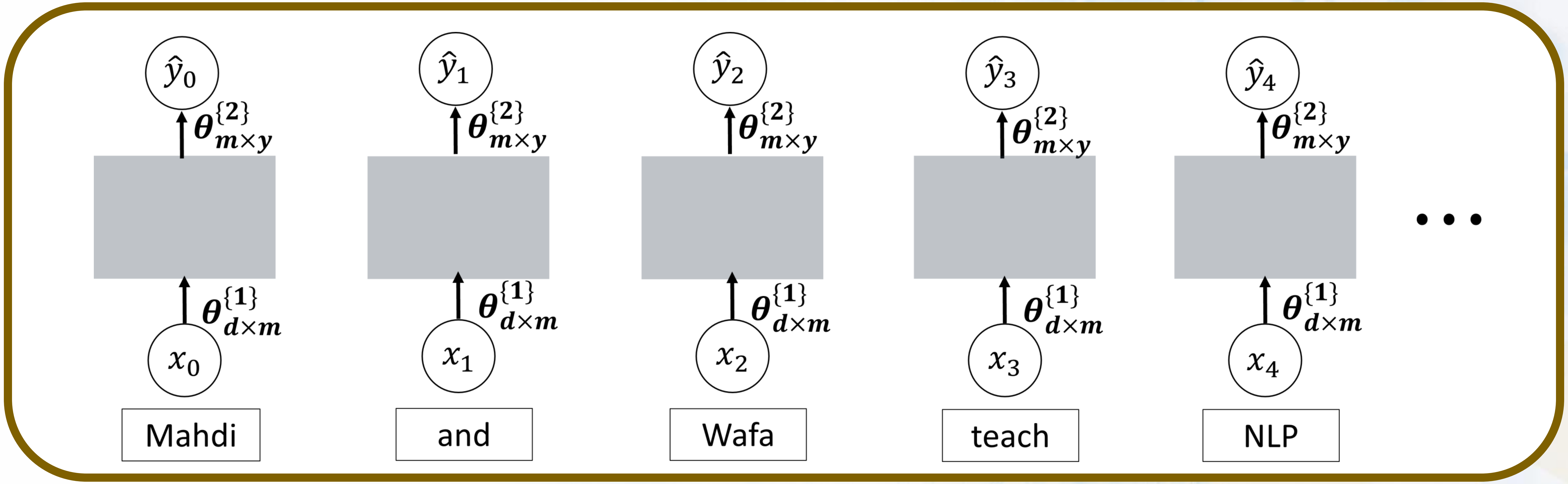
with Dr. Mahdi Roozbahani
& Wafa Louhichi

Deep Learning
Recurrent Neural Networks (RNN) – Part 2

Some of the slides are based on Ming Li (University of Waterloo – Deep Learning Part) with some modifications

Let's Go Back to our NER Problem Using a Feed-Forward Approach

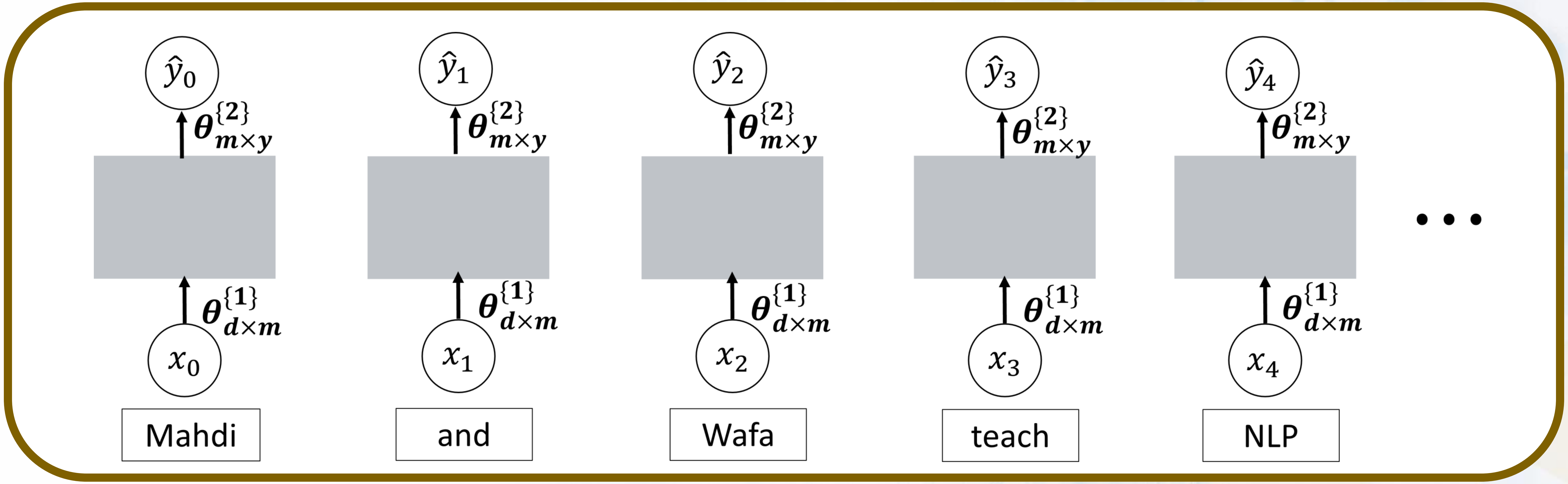
Sentence: Mahdi and Wafa teach NLP



- ***m:*** number of hidden neurons is a ***Hyper-Parameter*** and needs to be optimized
- There are two sets of different parameters (1) input to hidden layers: $\boldsymbol{\theta}^{1}_{d\times m}$ (2) hidden layers to output: $\boldsymbol{\theta}^{2}_{m\times y}$

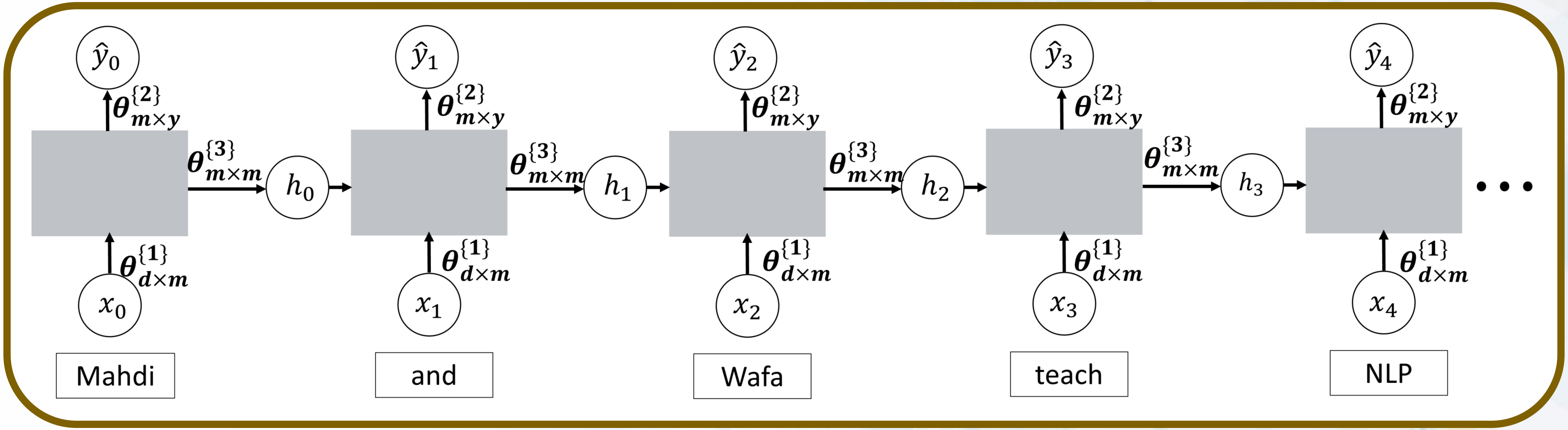
Let's Go Back to our NER Problem Using a Feed-Forward Approach

Sentence: Mahdi and Wafa teach NLP



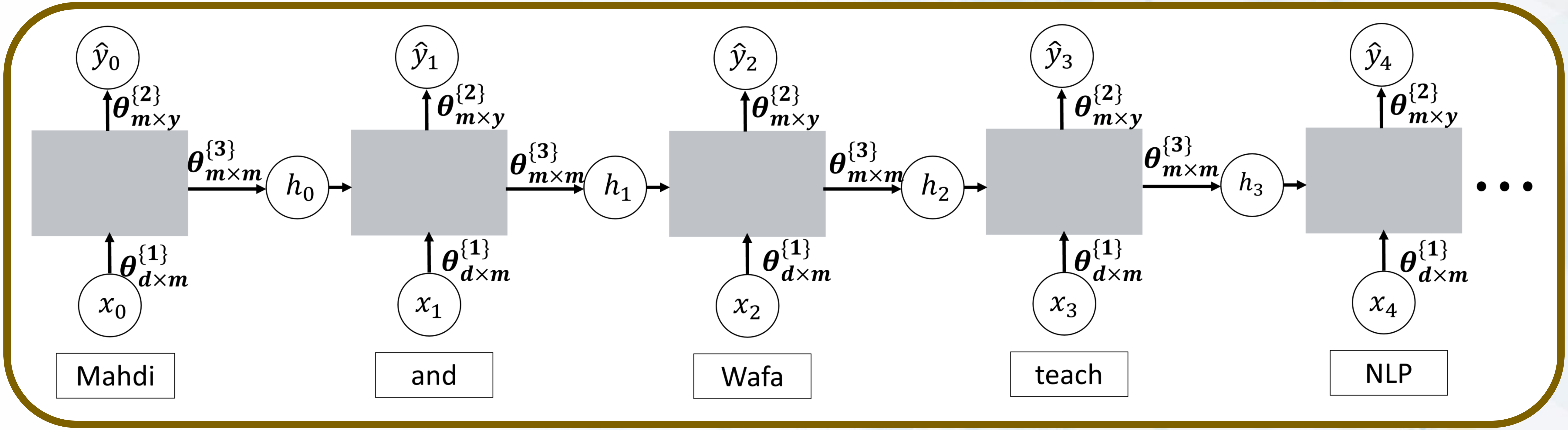
- ***m:*** number of hidden neurons is a ***Hyper-Parameter*** and needs to be optimized
- There are two sets of different parameters (1) input to hidden layers: $\boldsymbol{\theta}^{1}_{d\times m}$ (2) hidden layers to output: $\boldsymbol{\theta}^{2}_{m\times y}$

RNN Concept to Connect Different Time Steps



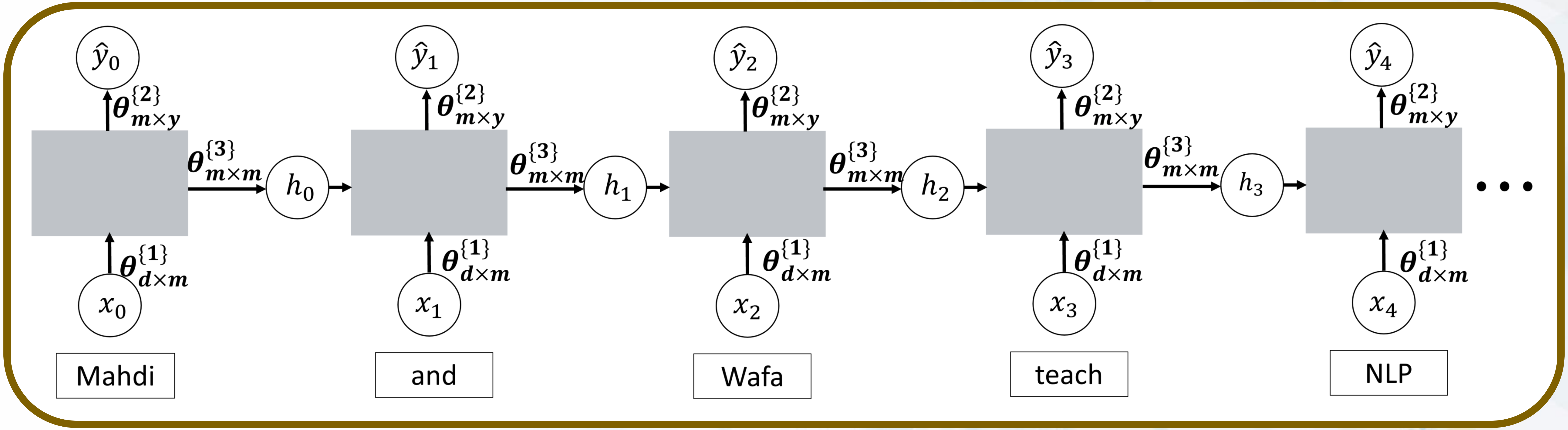
- We introduced a new set of parameters ($\boldsymbol{\theta}^{3}_{m\times m}$) which generates a new vector of hidden neurons (h_t) with size ***m***

RNN Concept to Connect Different Time Steps



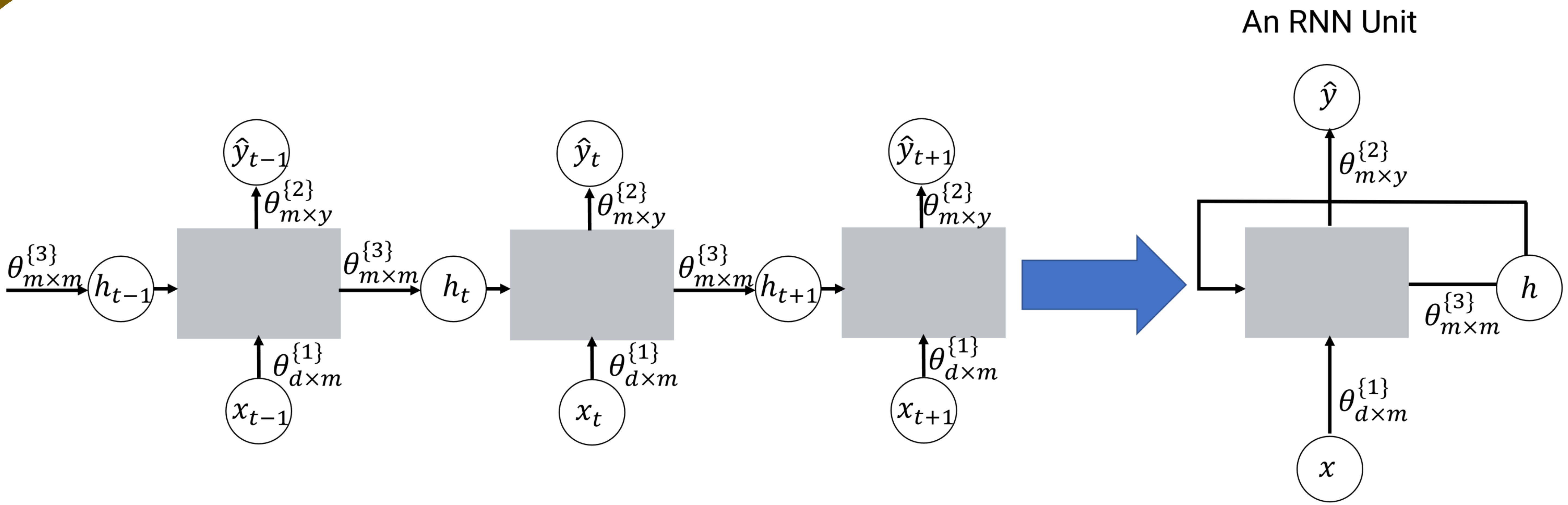
- We introduced a new set of parameters ($\boldsymbol{\theta}^{3}_{m\times m}$) which generates a new vector of hidden neurons (h_t) with size ***m***

RNN Concept to Connect Different Time Steps



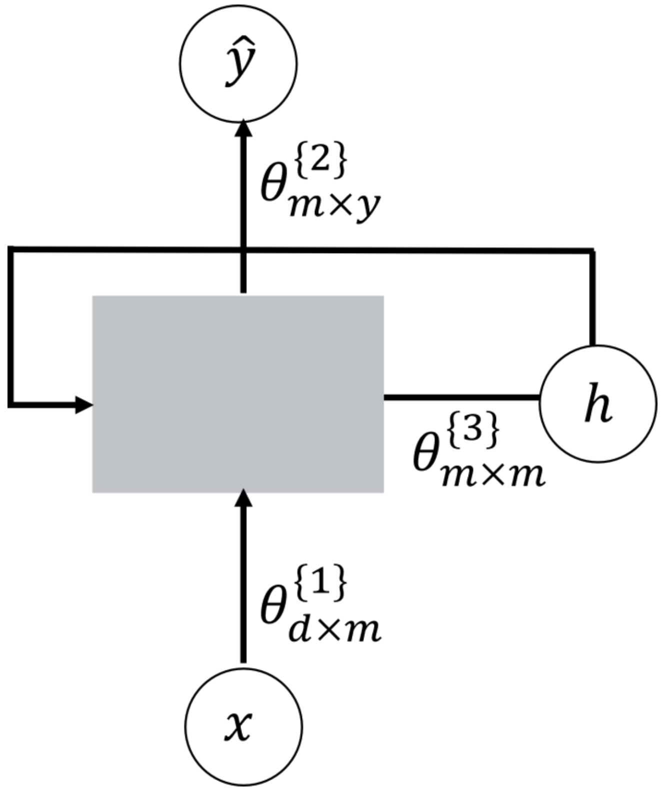
- We introduced a new set of parameters ($\boldsymbol{\theta}^{3}_{m\times m}$) which generates a new vector of hidden neurons (h_t) with size ***m***

Simple Representation of RNN (rnn_units)




```
tf.keras.layers.SimpleRNN(rnn_units)
```

Forward Pass: How to Calculate Past Memory (h) in RNN



$\boldsymbol{\theta}^{\{1\}}$: Weight (parameter) matrix associated with input data

$\boldsymbol{\theta}^{\{2\}}$: Weight (parameter) matrix associated with output data

$\boldsymbol{\theta}^{\{3\}}$: Weight (parameter) matrix associated with hidden state

Activation function such as Tanh (hyperbolic tangent)

Input

Past memory (previous step)

$$h_t = f(x_t, h_{t-1}, \theta)$$

Model parameters (weight)

The output will be a vector of size ***m***

The output will be a vector of size ***m***

Bias which is a vector of size ***m***

$$h_t = \tanh(\underbrace{x_t\theta^{\{1\}}} + \underbrace{h_{t-1}\theta^{\{3\}}} + b)$$

Forward Pass: How to Calculate Output for Each Step ($\hat{y}$) in RNN

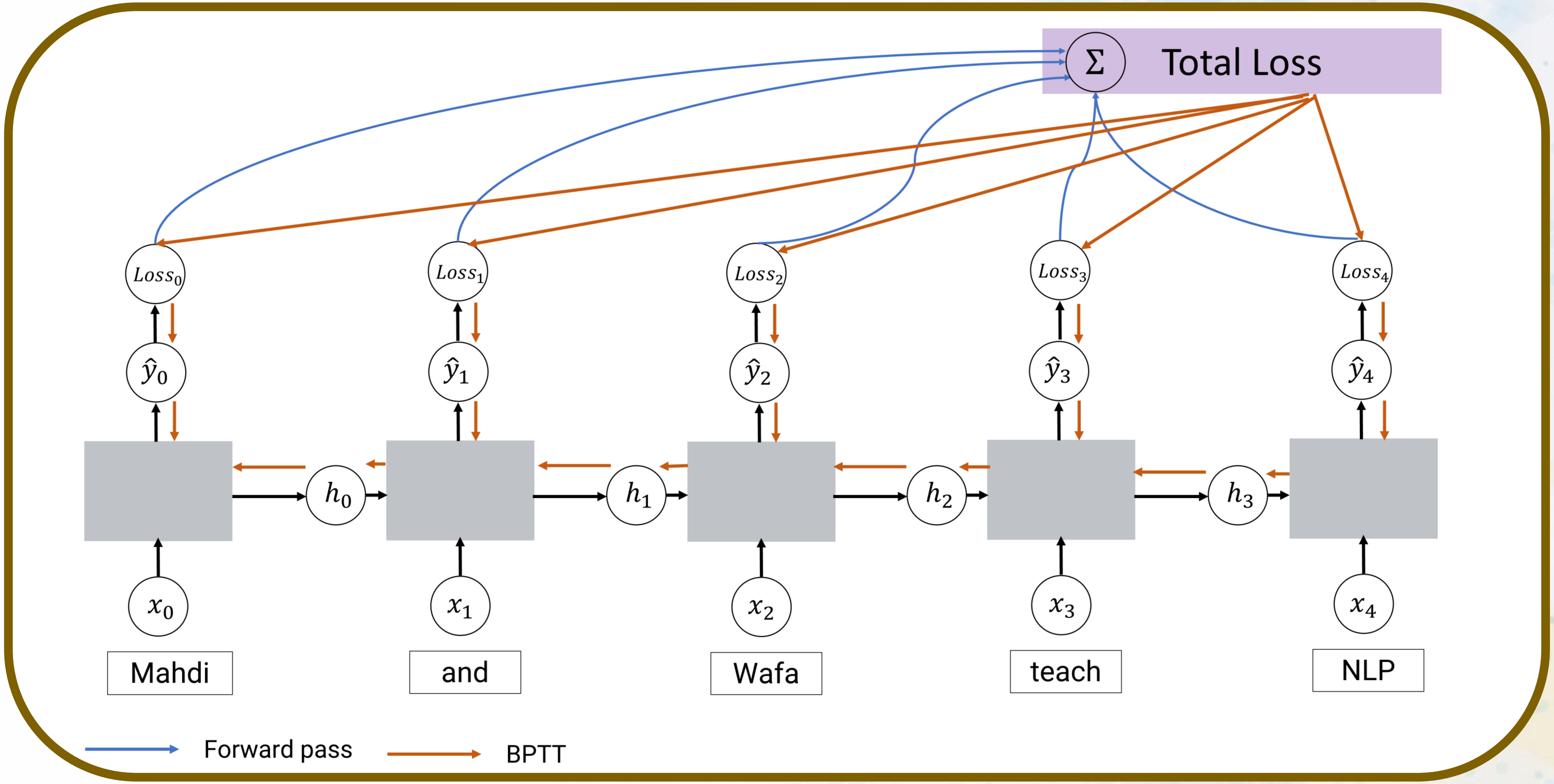
Forward Pass: How to Calculate Output for Each Step ($\hat{y}$) in RNN

Scaling the output between 0 and 1

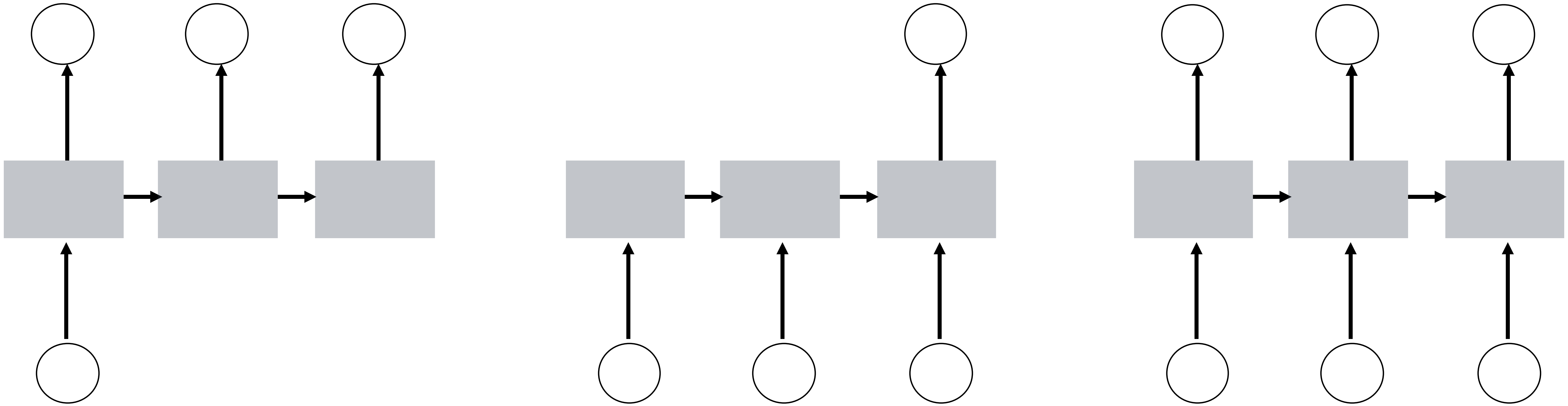
The output will be a scalar for our NER problem

$$\hat{y}_t = \text{softmax}\,(h_t\theta^{\{2\}})$$

Backpropagation Through Time (BPTT)



Different RNNs Models



One-to-Many
Text Generation;
Image Captioning

Many-to-One
Sentiment
classification

Many-to-Many
Part of speech tagging (POS),
NER, Translation, Forecasting

Some Problems with RNN:

Forward pass, backpropagation and repeated gradient computation can lead to two major issues:

Exploding gradient (high gradient values leading to very different weights in every optimization iteration)

* Solution: Gradient clipping (clip a gradient when it goes higher than a threshold)

Vanishing gradient (low gradient values that stall the model from optimizing the parameters)

*Solution:

- ReLu activation function
- LSTM, GRUs (different architectures)
- Better weight initialization

RNN suffers from short-term memory for a long sentence where words of interest may be placed far from each other in a sentence (vanishing gradient). Other architectures can help with this: LSTM, GRUs

Summary

- We learned about RNN
- The past memory
- Different architectures
- Some possible issues with RNN