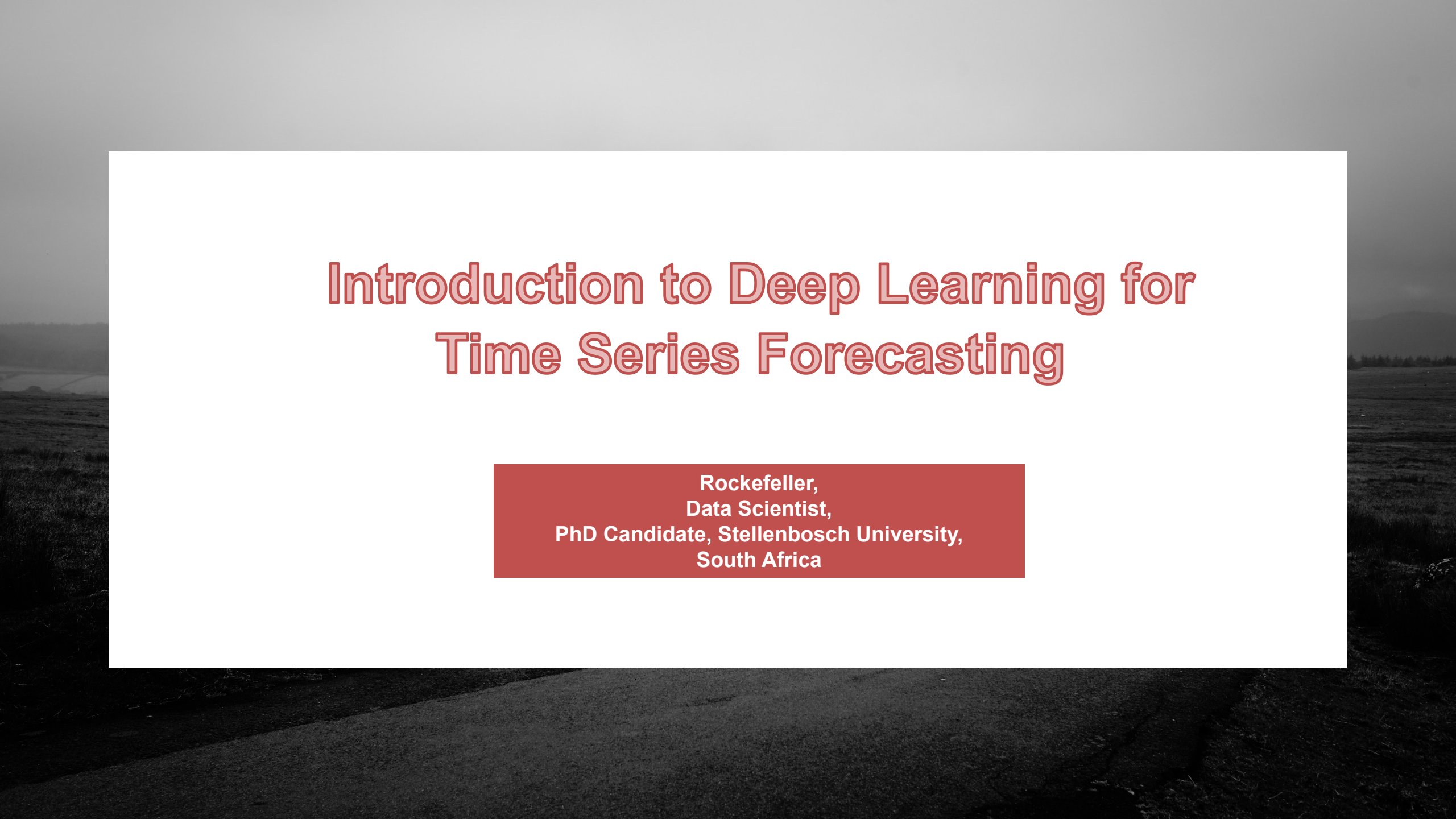


Introduction to Deep Learning for Time Series Forecasting

Rockefeller,
Data Scientist,
PhD Candidate, Stellenbosch University,
South Africa

Outline || Goals

- **Time Series forecasting as a Supervised Learning Task**
- **How do Feed Forward Neural Networks Learn?**
- **From Feedforward Neural Networks to Recurrent Neural Networks**
- **Recurrent Neural Network (RNN) Architecture**
- **An LSTM Architecture**
- **Hands on!!!!**

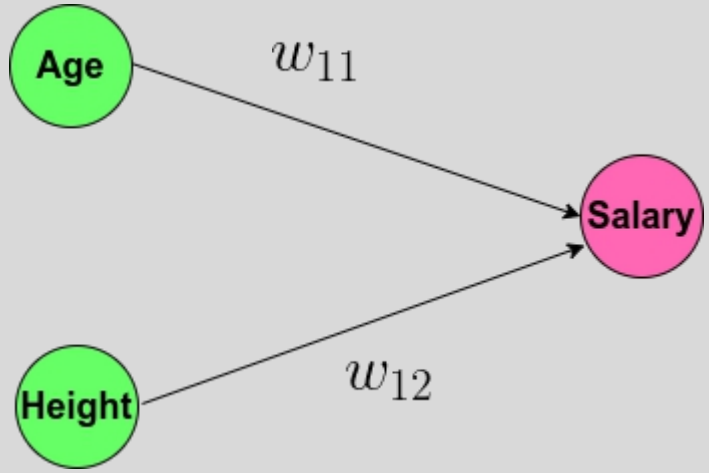
Target Audience

- **Beginner to intermediate Level**
- **Know the basics of Python || Pytorch**
- **Know the basics concepts of Machine Learning**
- **Have some little experience of building machine learning models**

Raw observations

Age	Height	Salary
24	165	20
46	142	36
33	155	65
46	172	35

Shallow Neural Network

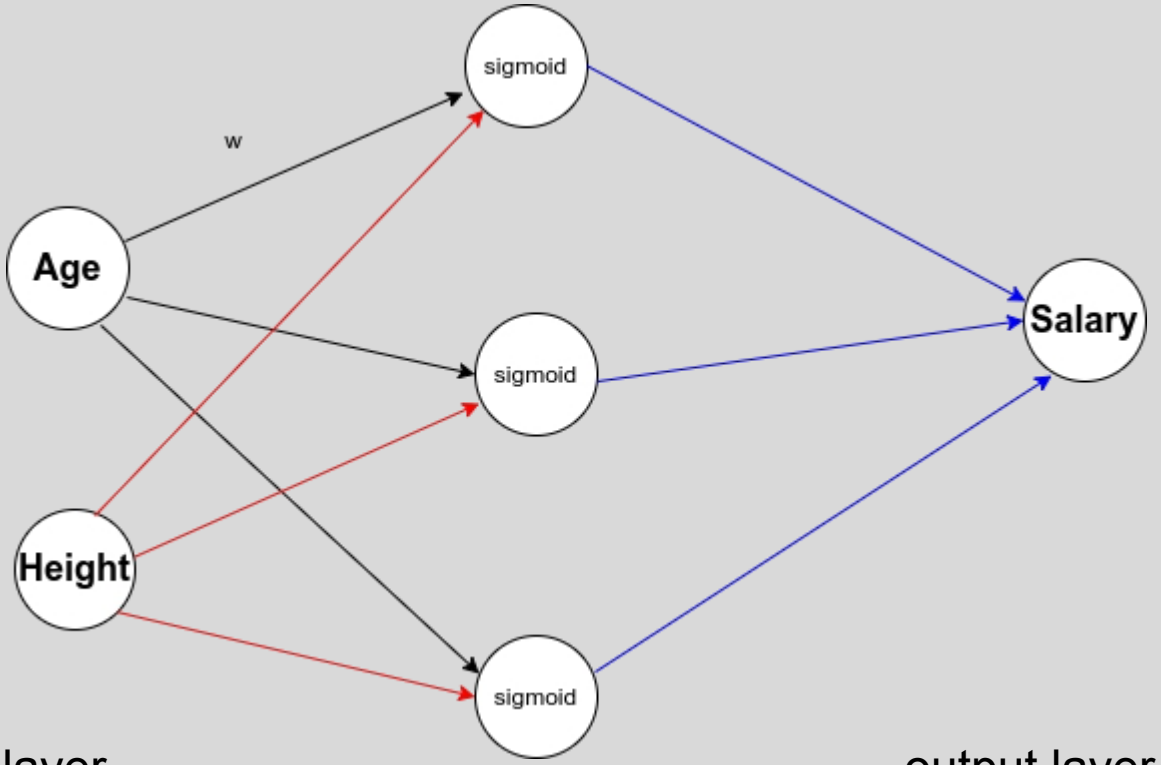


$$Salary = f_1(w_{11} * Age + w_{12} * Height + b_1)$$

Raw observations

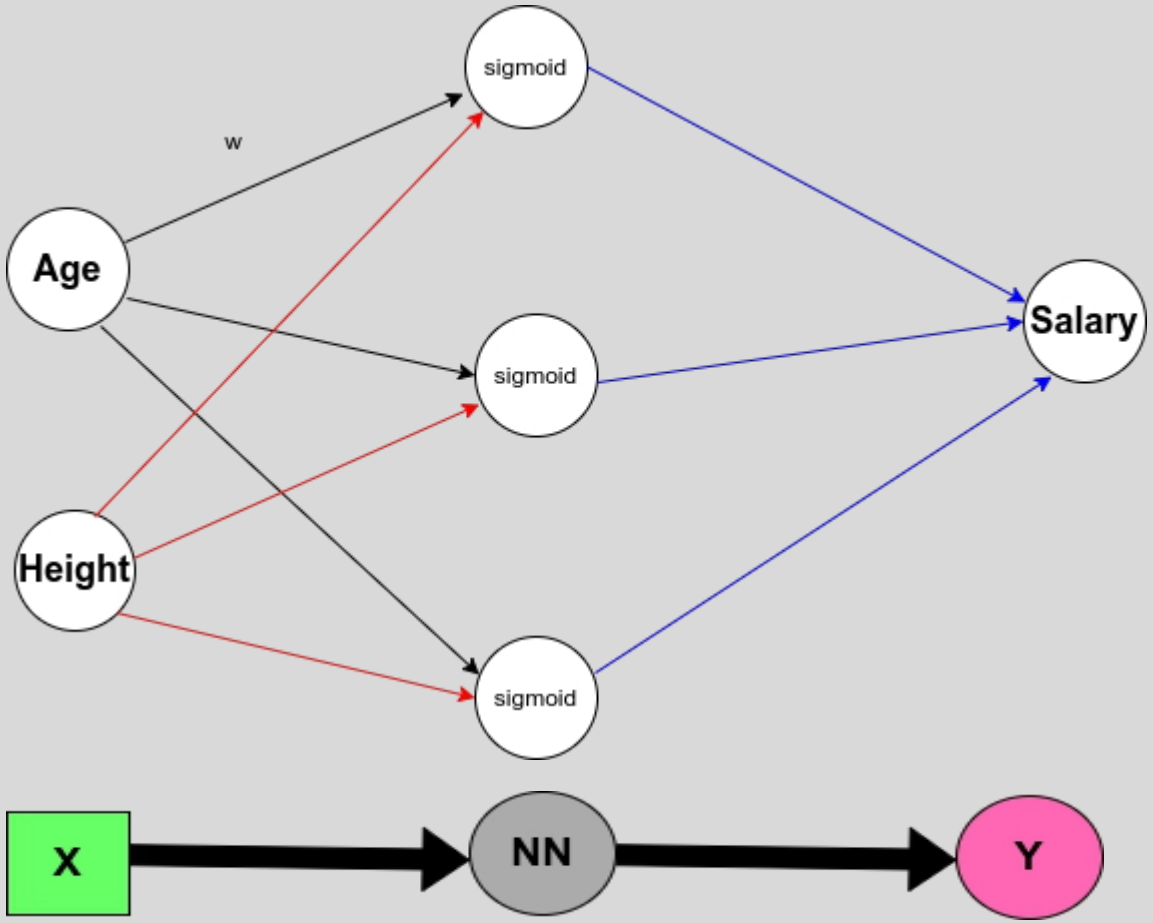
Age	Height	Salary
24	165	20
46	142	36
33	155	65
46	172	35

Deep Neural Network



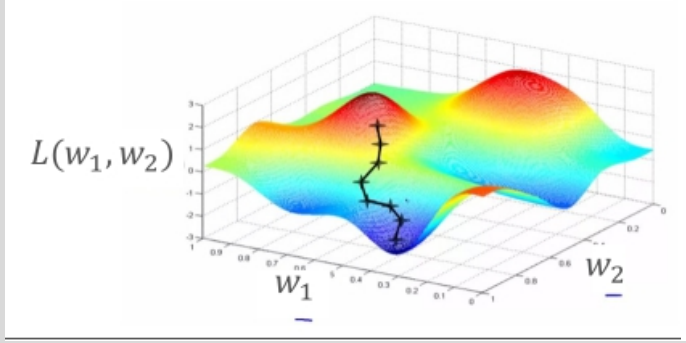
$$Salary = f_n(...(f_2(W_{21}f_1(w_{11} * Age + w_{12} * Height + b_1) + b_2) + ... + b_n)$$

Feed forward Neural Network



Training Overview

- The loss function L(weight, biases) measures the discrepancy between predicted and true values.
- Activation functions could be **tanh, sigmoid, ReLu** depending on the use case.
- Optimization algorithm (gradient-based) tries to find weights and biases that minimize the loss function



Optimization algorithm

Initialization: $w = w_0$

While stopping criterion not met:

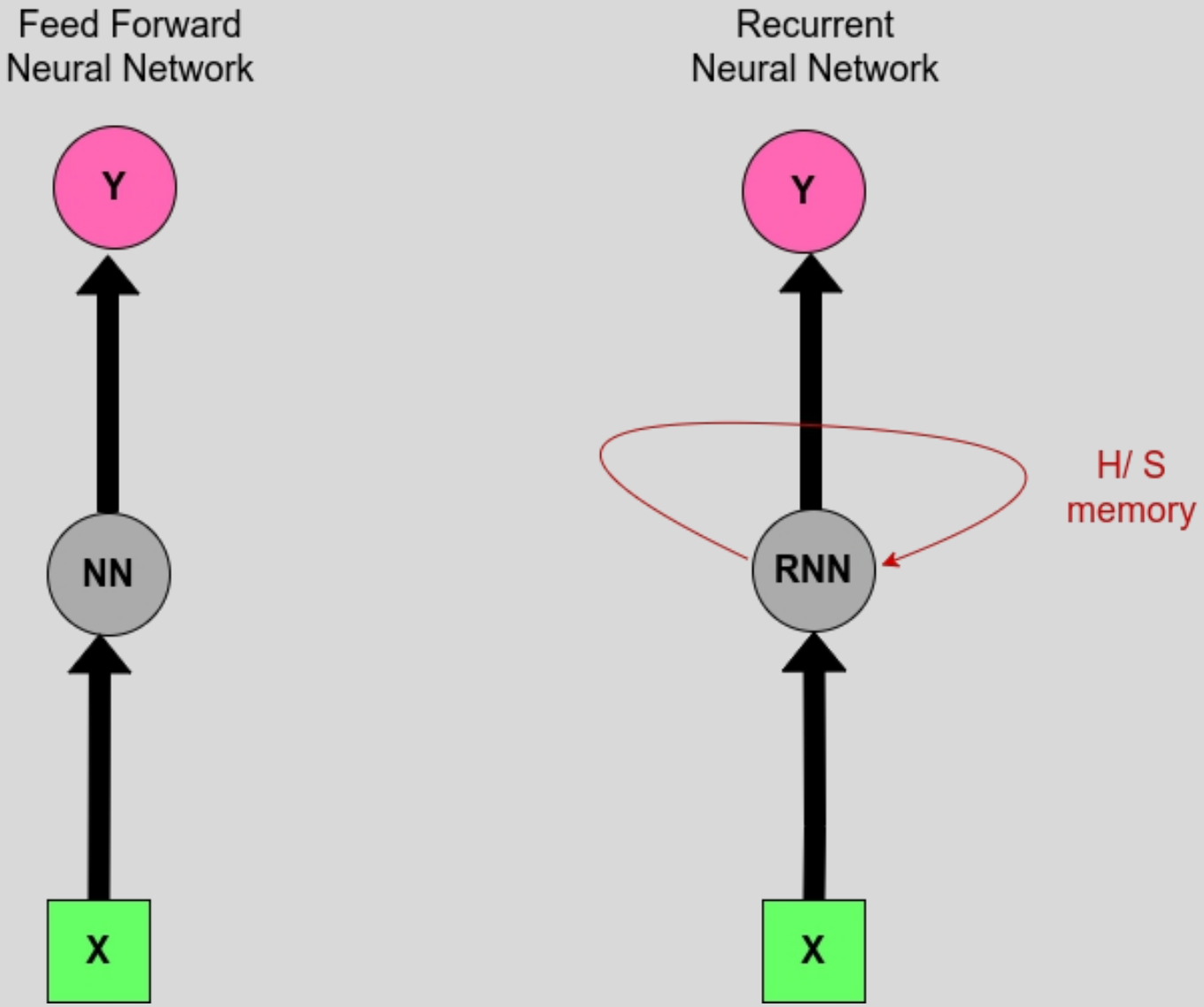
$$w = w - \alpha \cdot \nabla L(w)$$

- **Time Series requires are observations taken sequentially in time**
- **Time Series forecasting predicts future based on the past**

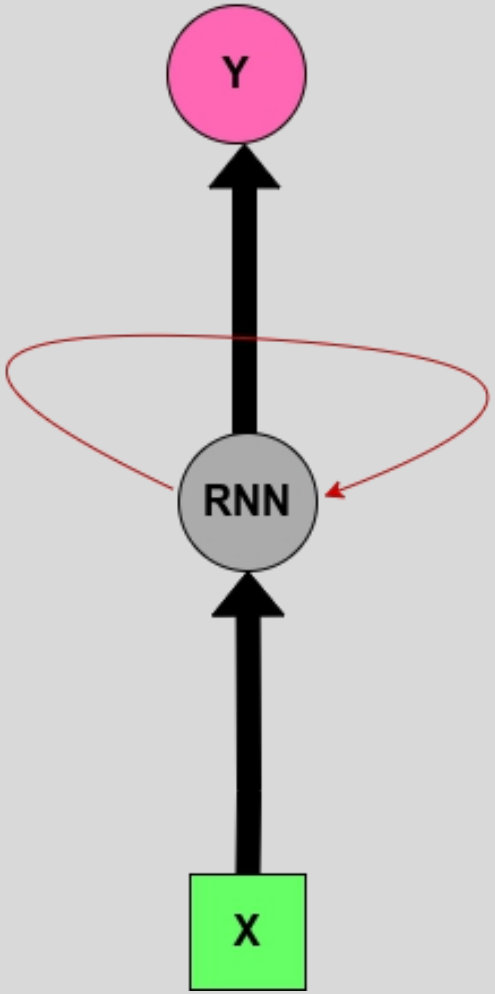
Age	Height	Salary
24	165	20
46	142	36
33	155	65
46	172	35

	Date	Temperature
History	2022-11-01	19
	2022-11-02	17
	2022-11-03	12
	2022-11-04	22
	2022-11-05	25
	2022-11-06	21
Future	2022-11-07	?
	2022-11-08	?
	2022-11-09	?

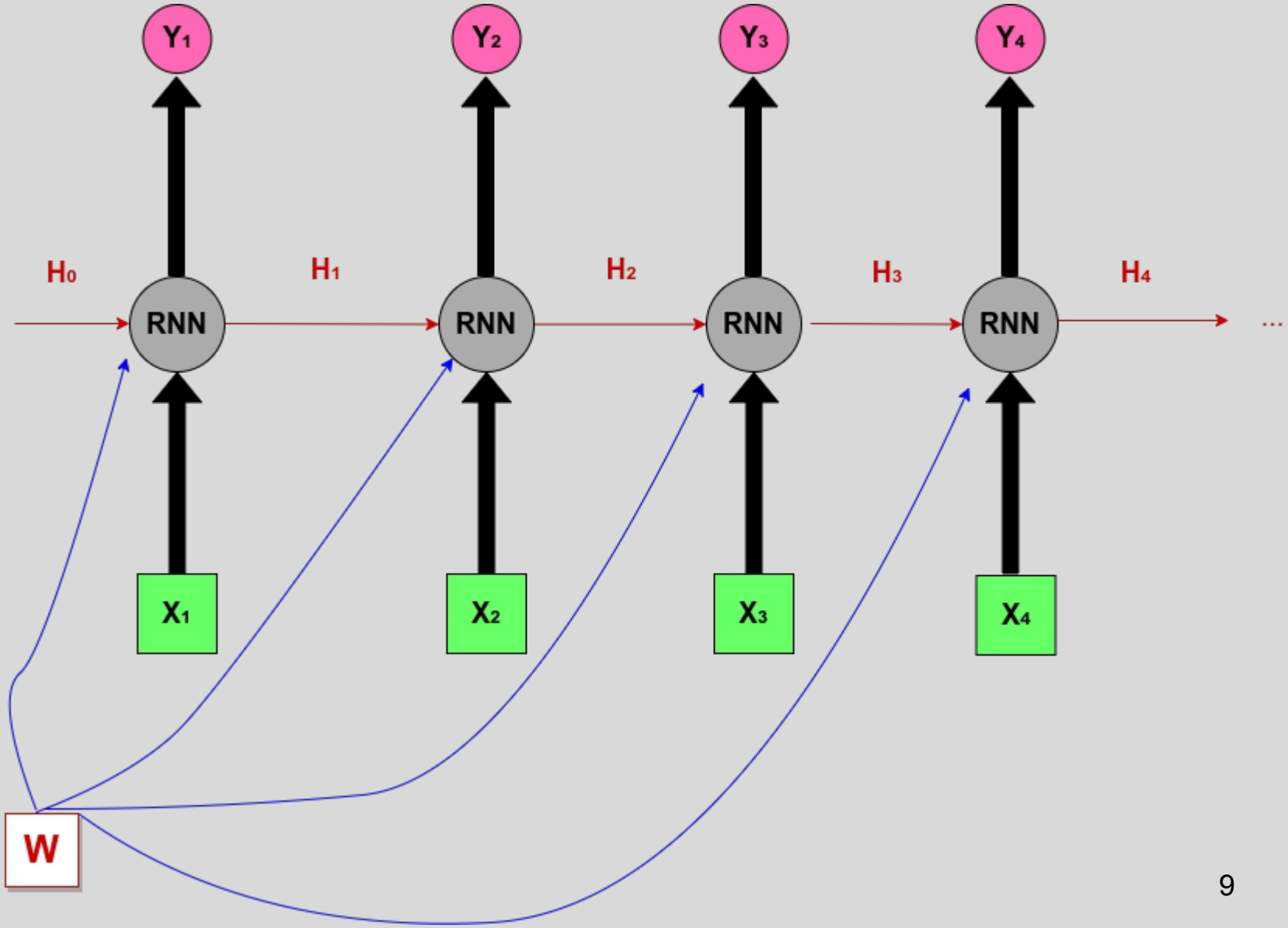
Feed forward Neural Net vs Recurrent Neural Net



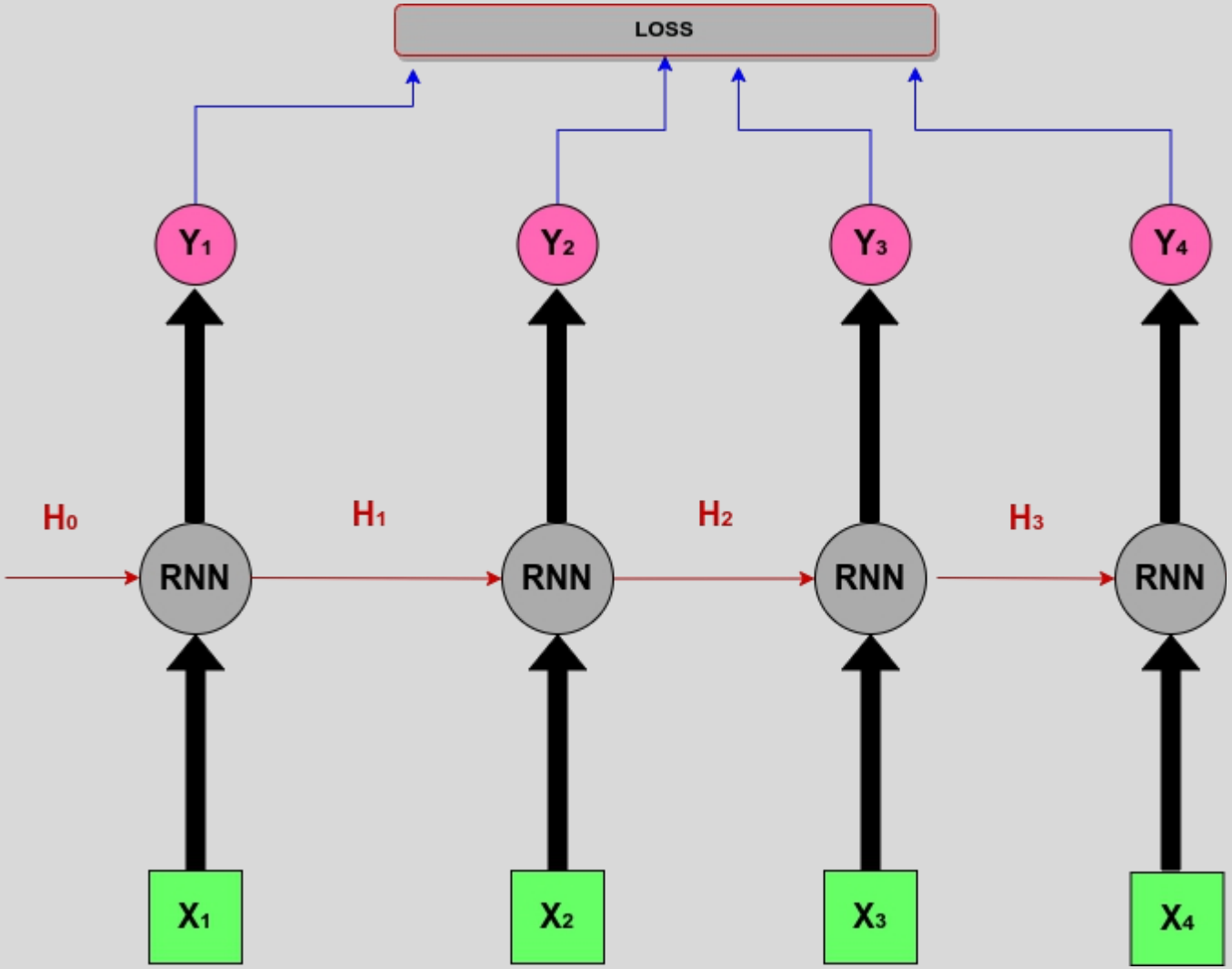
Recurrent
Neural Network



Unfolded Recurrent Neural Network

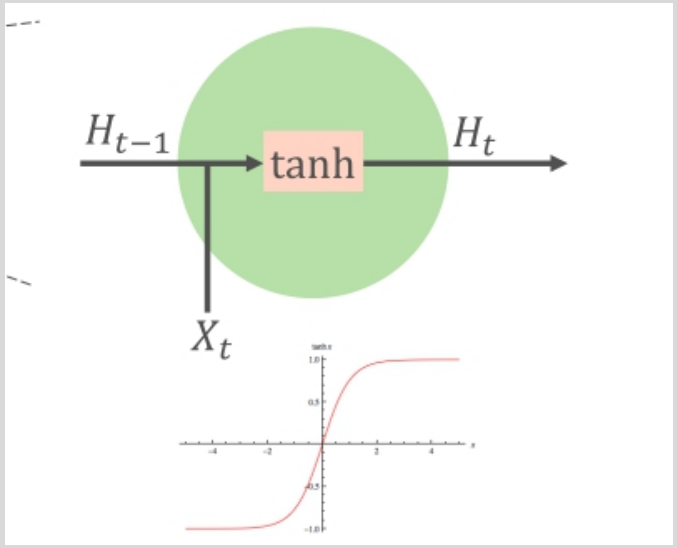


Training a Recurrent Neural Network



forward through the entire sequence to compute loss

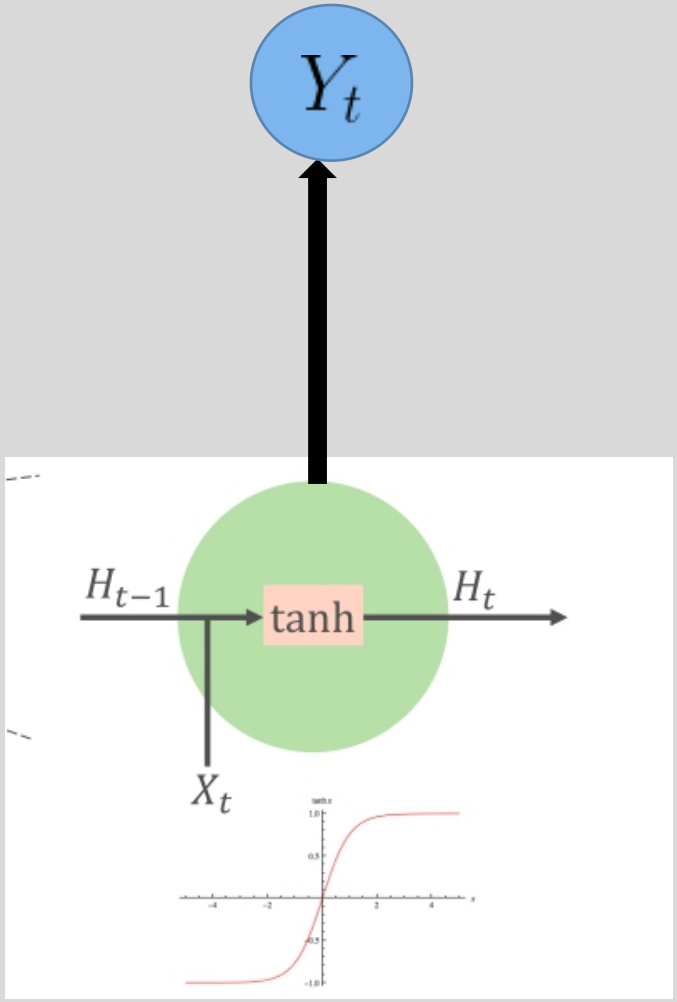
An RNN Cell



$$H_t = \tanh(W_h H_{t-1} + W_x X_t)$$

Transition Equation

An RNN Cell



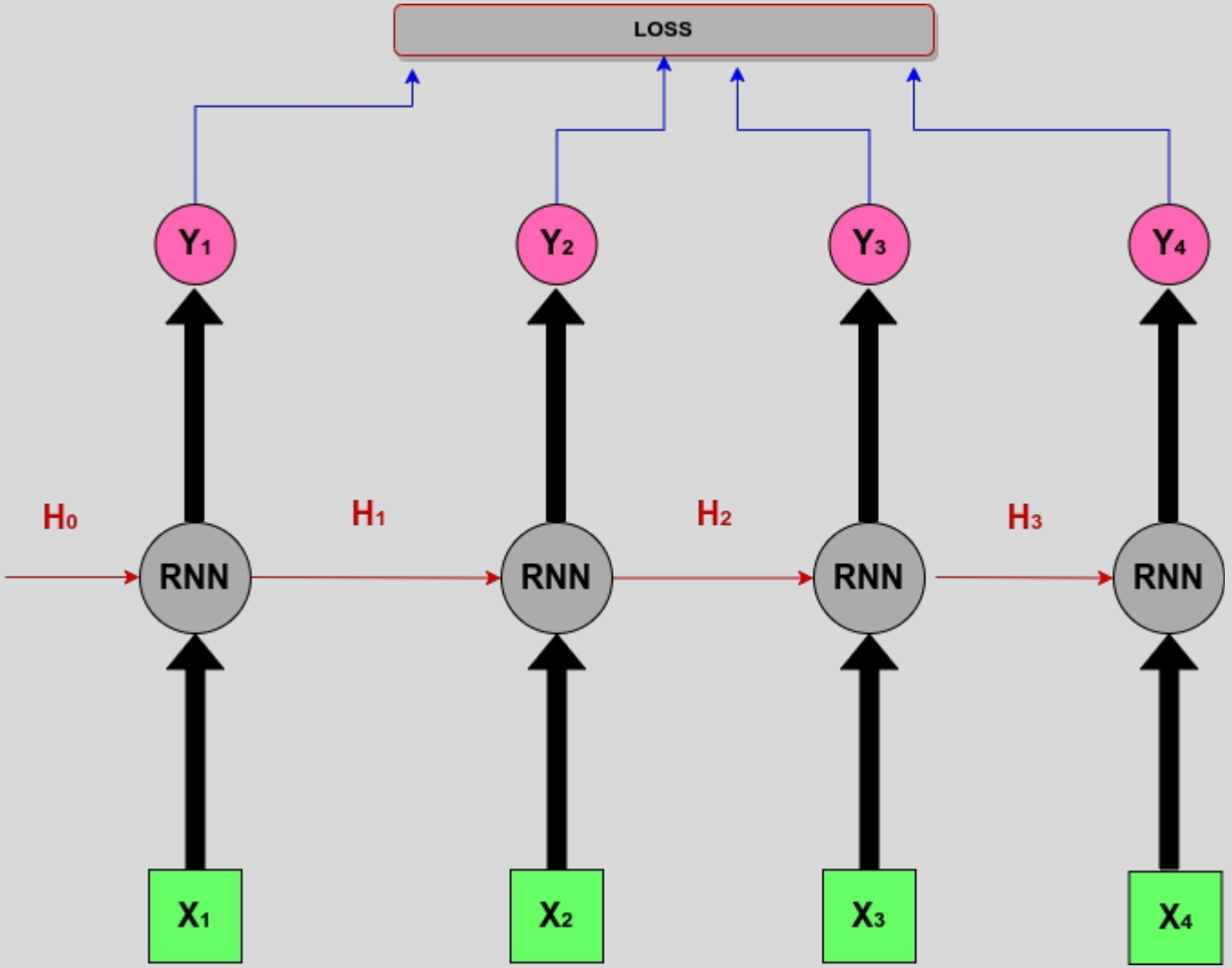
$$Y_t = W_y H_t$$

Output Equation

$$H_t = \tanh(W_h H_{t-1} + W_x X_t)$$

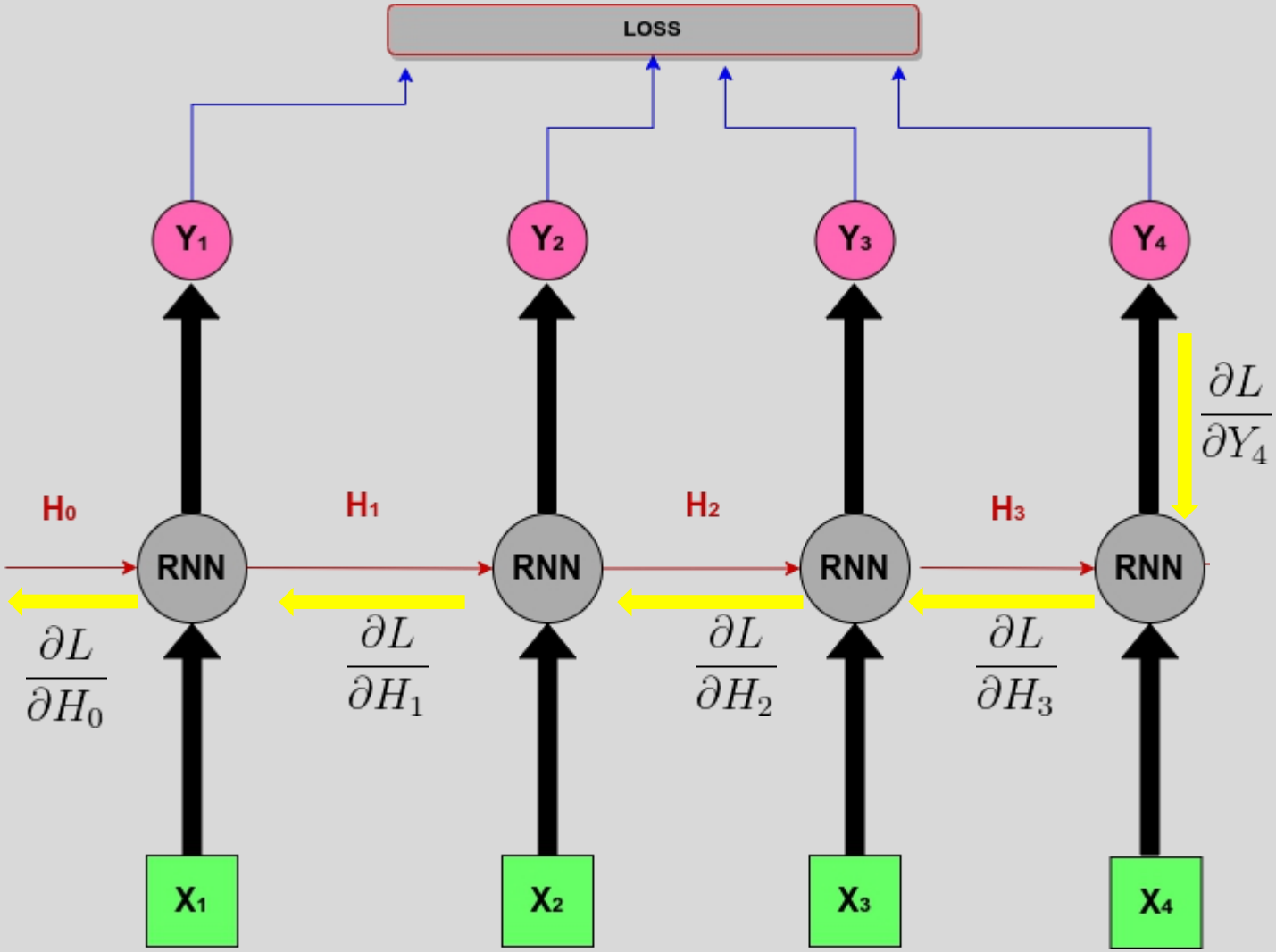
Transition Equation

Training a Recurrent Neural Network



forward through the entire sequence to compute loss

Training a Recurrent Neural Network



forward through the entire sequence to compute loss

then, backward through the entire sequence to compute gradient

To improve on Vanilla Recurrent Neural Network

- During Backpropagation, the computation of the gradient of H_0 involves repeated tanh and many factors of W. This could lead to:

- Exploding Gradient (e.g. 5*5*5*5*5*....) The gradient will grow exponentially and the program will crash. Usually, this is solved through gradient clipping which consists of clipping the gradient when it goes higher than a treshold value.

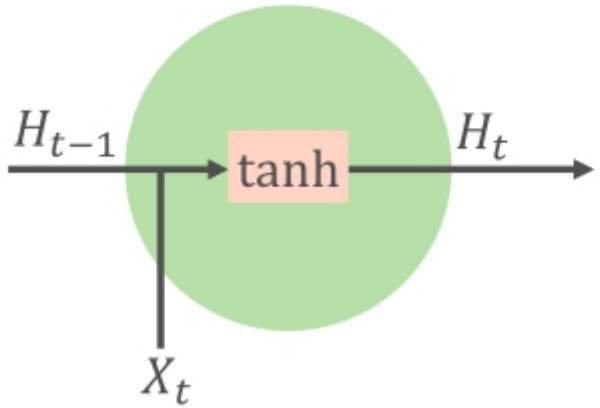
- Vanishing Gradient (e.g. 0.7*0.7*0.7*0.7....). Here, this is more problematic because it is not obvious whren they occur or how to deal with them.

- Solution: Change in activation function, proper initialization, regularization, or change of architecture to LSTM or GRU.

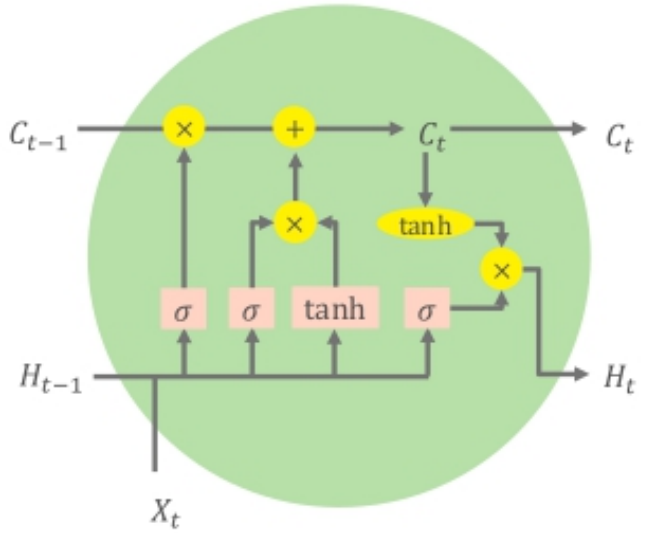
An RNN Cell vs an LSTM Cell

Vanilla RNN:

$$H_t = \tanh(\mathbf{W} \cdot [H_{t-1}, X_t])$$



LSTM



Hidden cell state: $C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$

Hidden state: $H_t = o_t \times \tanh(C_t)$

Forget gates: $f_t = \sigma(\ W_f \cdot [H_{t-1}, X_t]\ )$

Input gates: $i_t = \sigma(\ W_i \cdot [H_{t-1}, X_t]\ )$

Candidate gates/states: $\tilde{C}_t = \tanh(\ W_g \cdot [H_{t-1}, X_t])$

Output gates: $o_t = \sigma(\ W_o \cdot [H_{t-1}, X_t]\ )$