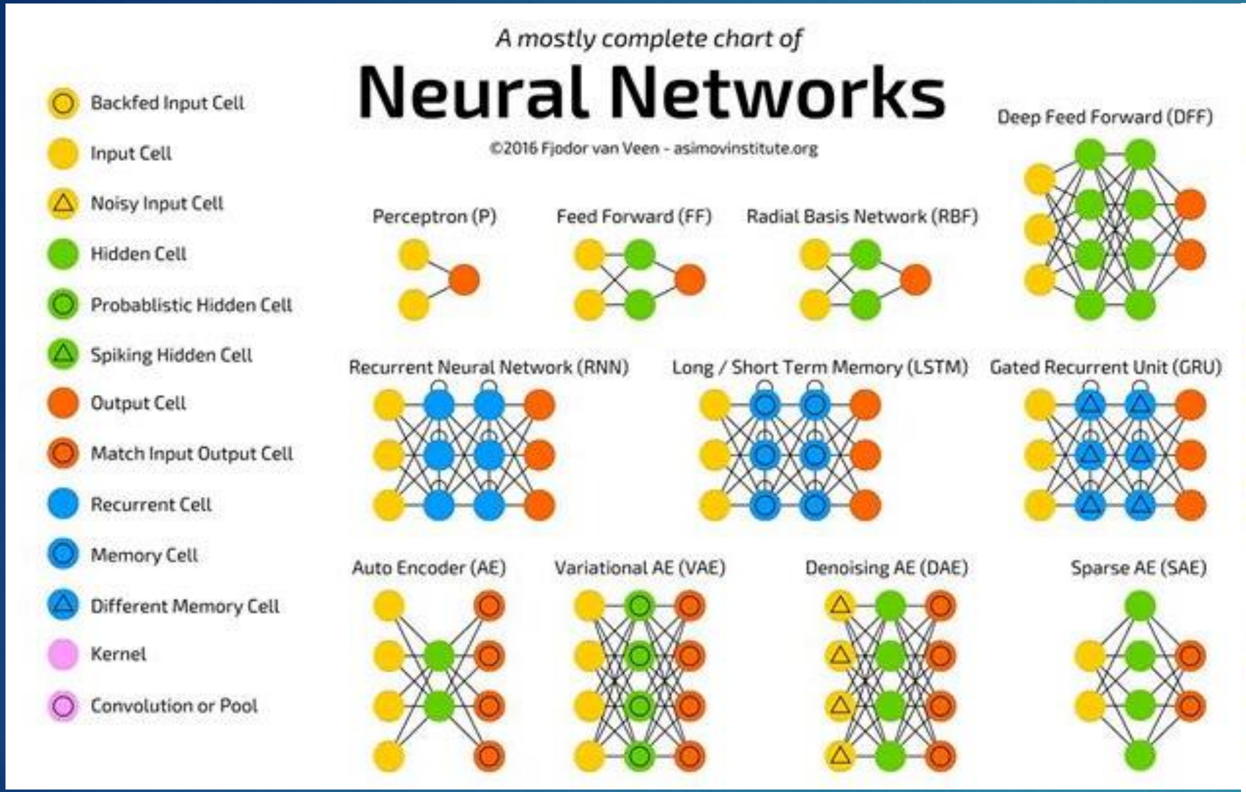
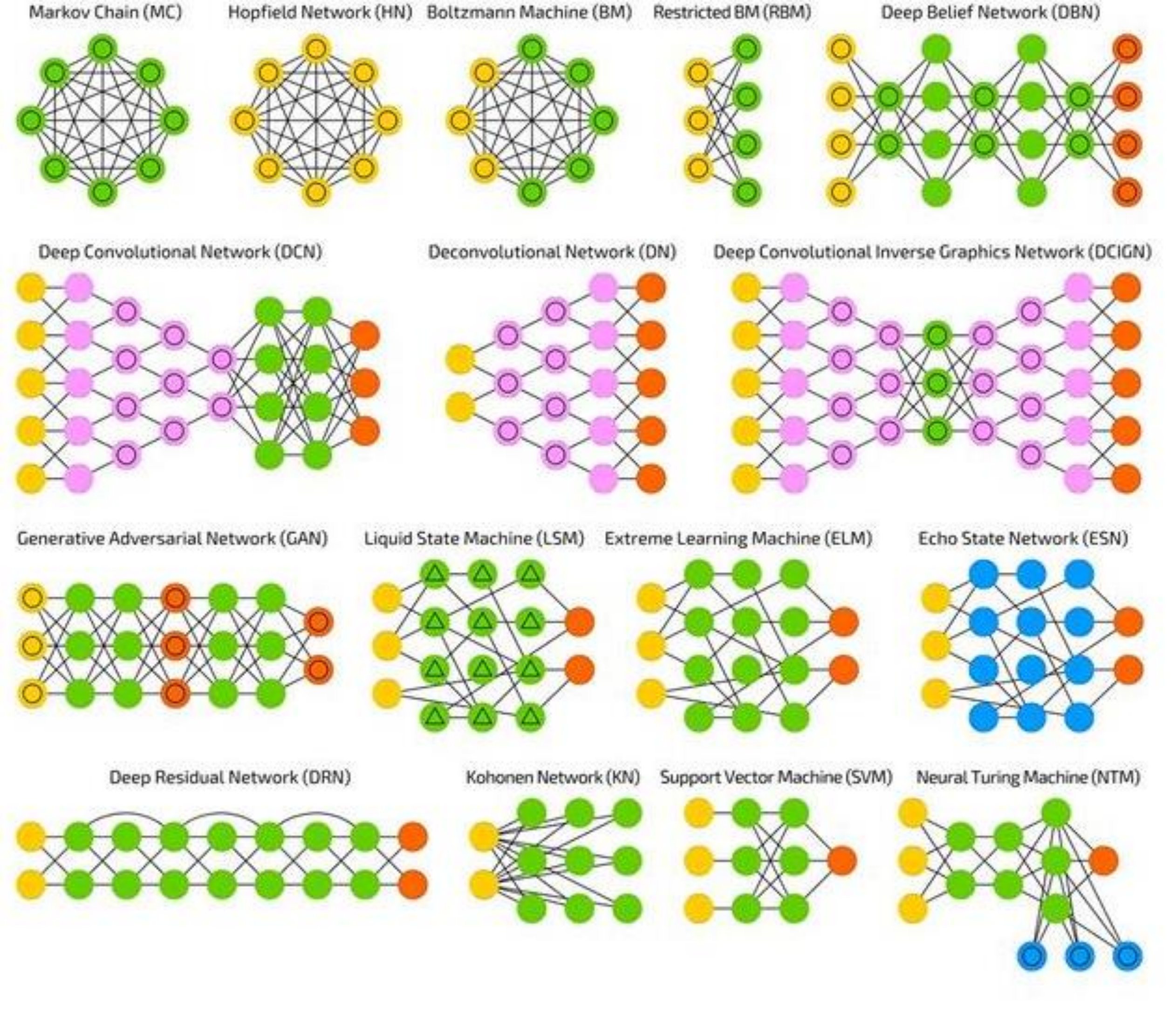
Building Time Series Predictive Model with Hyper-Parameters Tuning using Differential Evolution Algorithm

KYU CHO

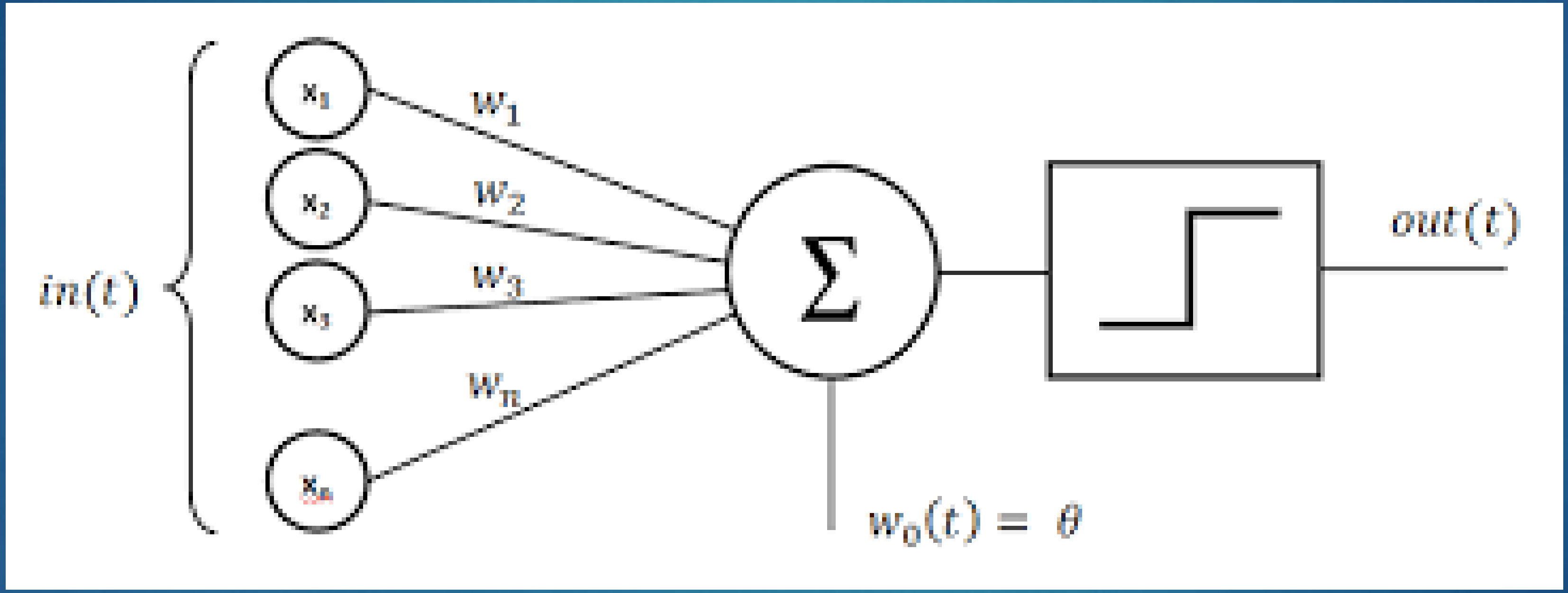
10/27/16

Table of Contents


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)

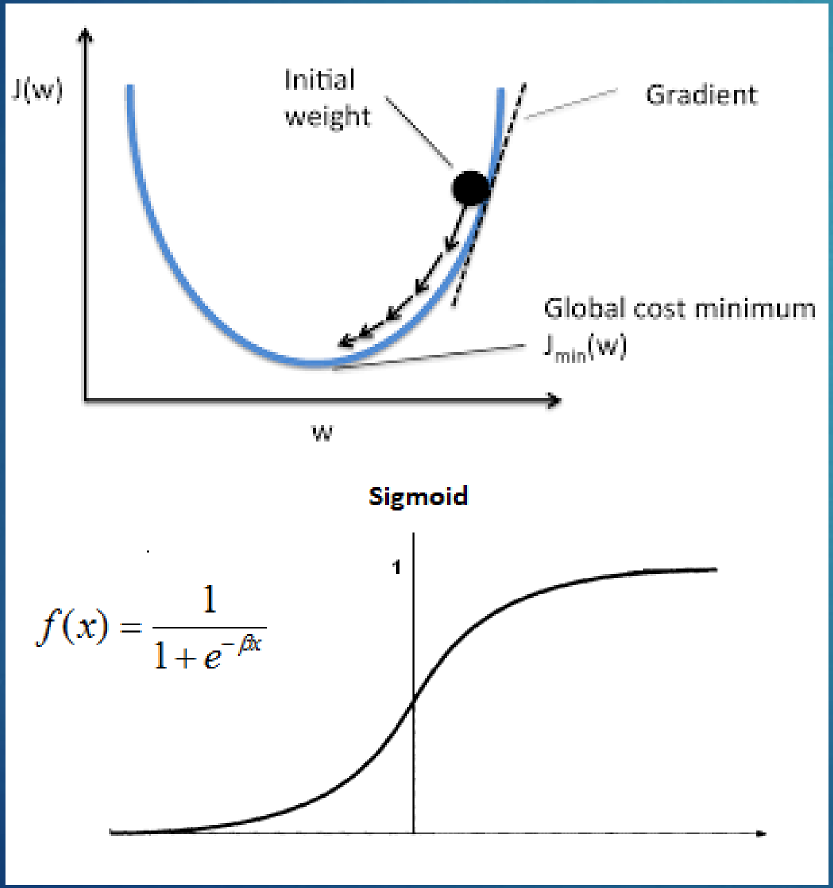

Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

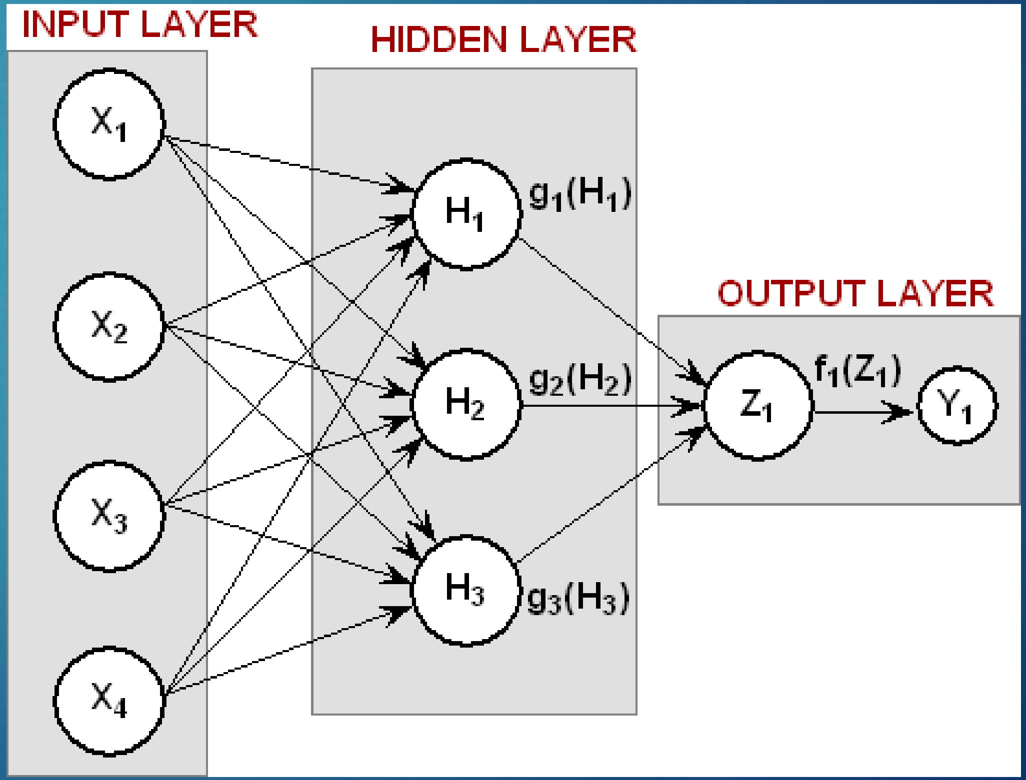
Perceptron



Feed Forward

- Gradient Descent Algorithm!





Momentum

- Regular Momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta).$$

$$\theta = \theta - v_t.$$

- Nesterov Accelerated Gradient (NAG)
  - Look ahead and then correct the position if it makes a mistakes

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}).$$

$$\theta = \theta - v_t.$$

- https://www.youtube.com/watch?v=OWzkRD6MjYI
- And gifs

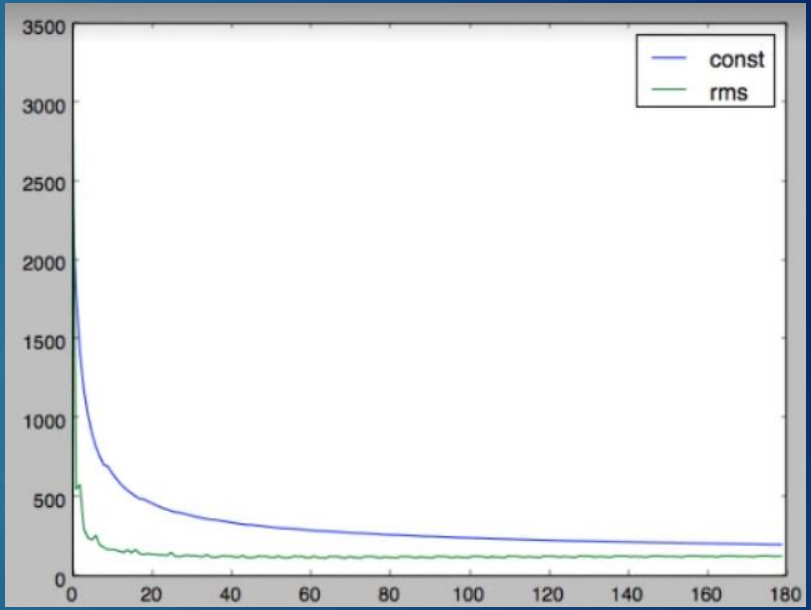
Learning Rate (Hyper-Parameter1)

- Point: Decay the learning rate as we close to the solution

- Step Decay
  - Ex) Reduce every 5, 10 or 20 steps
- Exponential Decay
  - Ex) LR = exp(LR)
- 1/t decay
  - EX) LR = 1/(1 + LR)

Learning Rate (Hyper-Parameter1)

- Adagrad - It adapts the learning rate to the parameters.
  - Keep the current changes in to the cache
    - More changes now and slower changes later
  - w -= LR * gradient / sqrt(cache) + epsilon)
  - But it's too aggressive

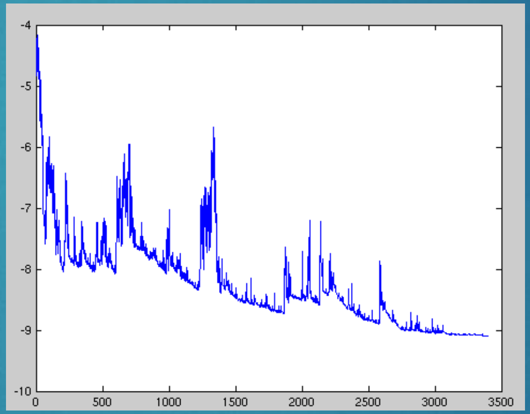


- RMSprop
  - Make cache self decay (leaky cache)
  - cache = decay_rate * cache + (1-decay_rate)*gradient^2
  - w -= LR * gradient / sqrt(cache) + epsilon)

Full vs Batch vs Stochastic GD (Hyper-Parameter 2)

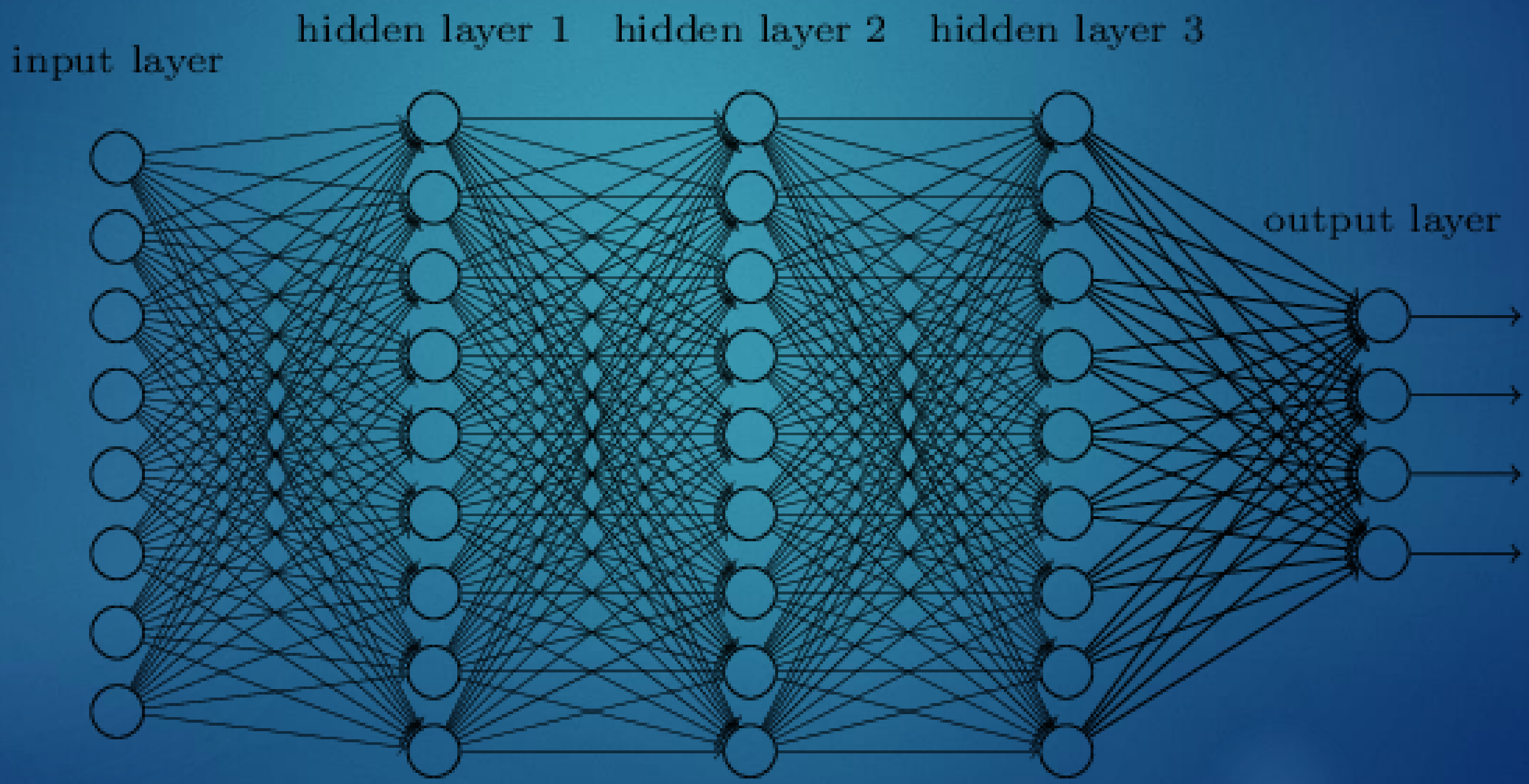
- Full gradient descent
- O(N)

- Stochastic gradient descent
- O(1) $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$.



- Batch gradient descent
- O(batch size) $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$.
- Batch size is another hyper-parameter

Number of Node and Layers (Hyper-Parameter 3)



Why RNN?

- Feed Forward
  - Can’t take sequences
    - Ex) "I love dogs and cats" vs "Dogs love cats and I“
  - Number of layer grow exponentially as the number of input grows
- Markov Chain
  - longer seq. have prob. that approach 0 because it keeps multiplying values < 1
  - Markov Assumption, is less accurate
    - Ex) p(cats) ~= p(cat | and)

Why RNN?

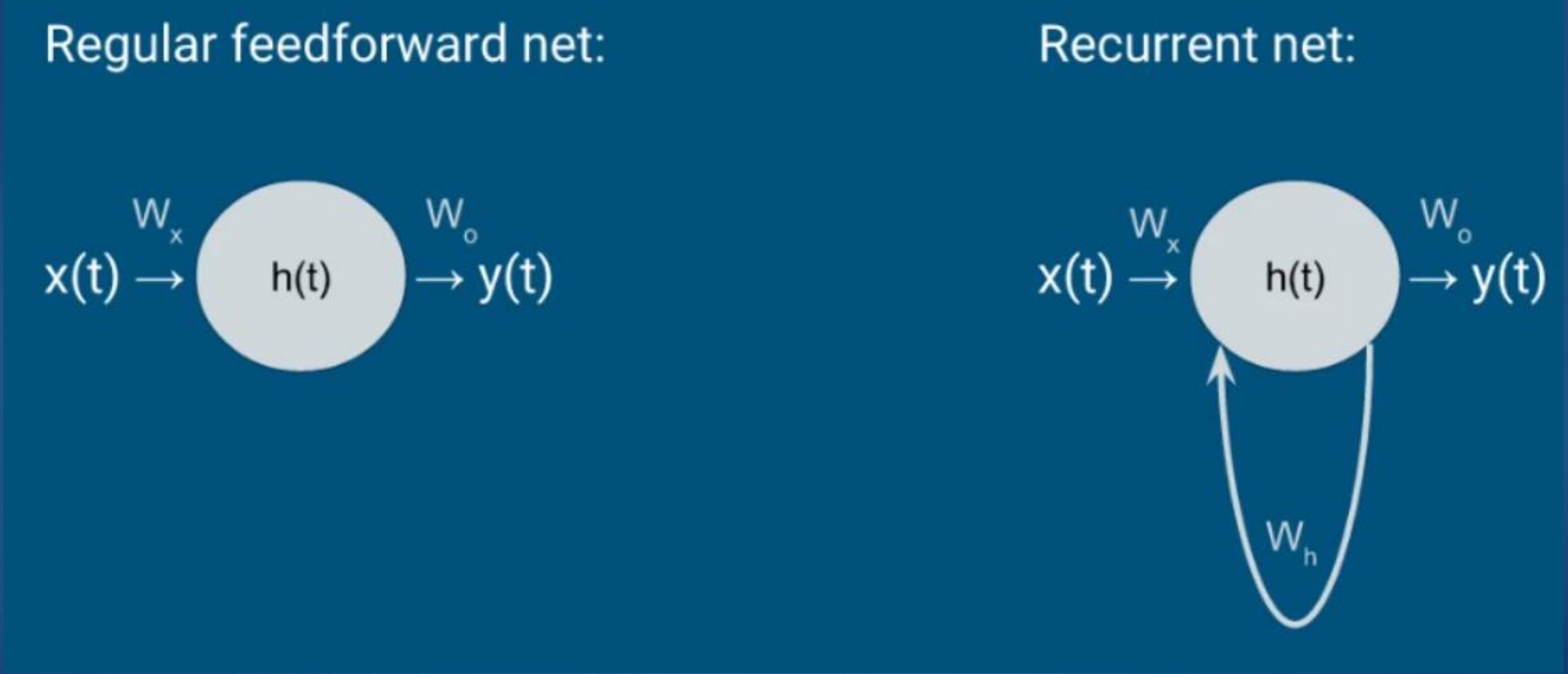
- Predict a label over entire sequence
  - Ex) "hello world" -> male, "bob hi" -> female
- Predict a label for every step of input sequence
  - No need to wait till the end of the node

Simple RNN

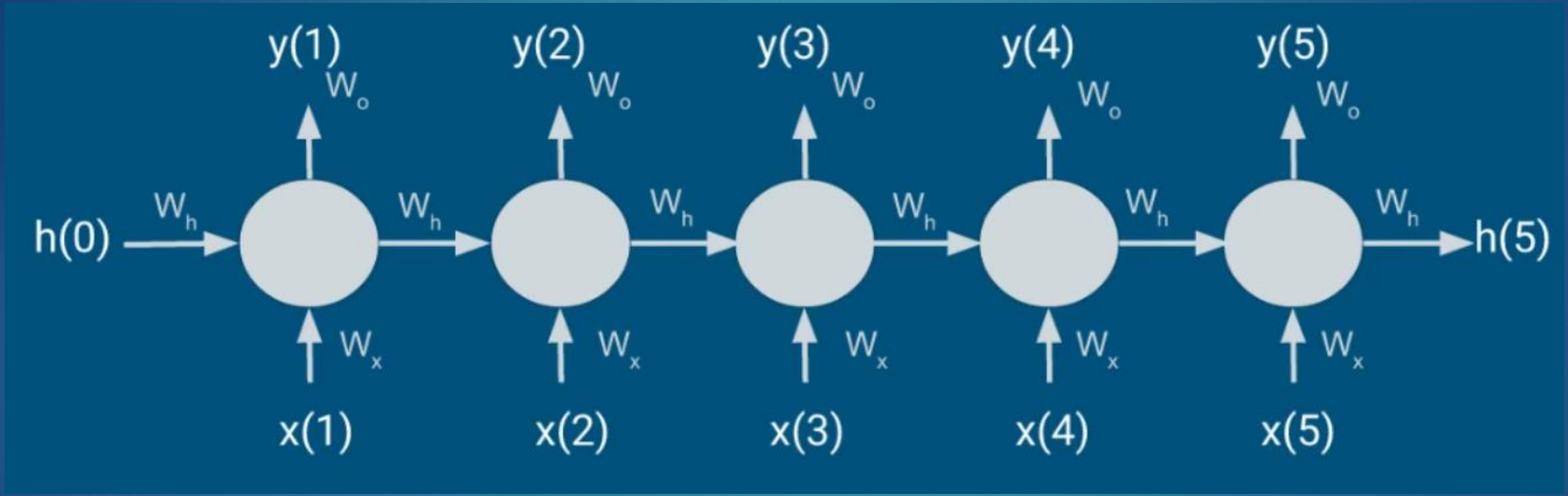
$$h(t) = f(W_h^T h(t-1) + W_x^T x(t) + b_h)$$

$$y(t) = \mathrm{softmax}(W_o^T h(t) + b_o)$$

f = sigmoid, tanh, relu

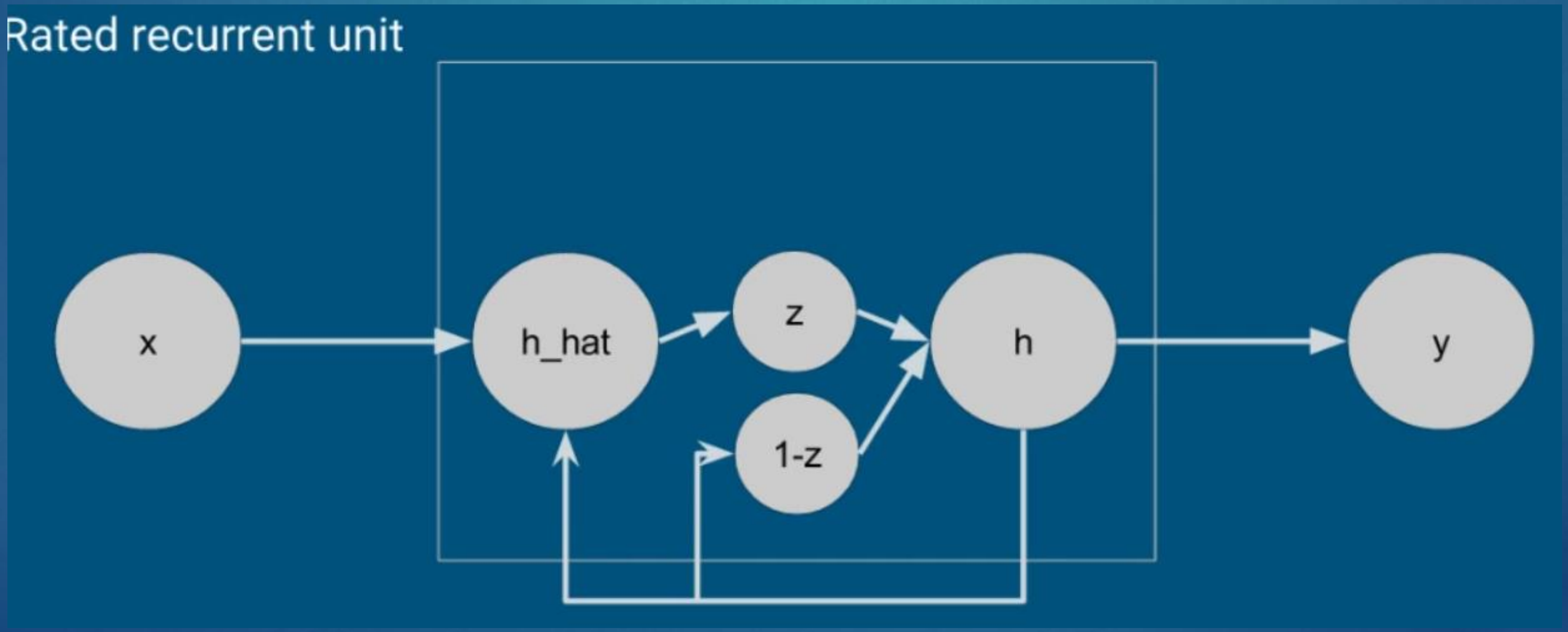


Unfold Simple RNN



Rated Recurrent Unit (RRU)

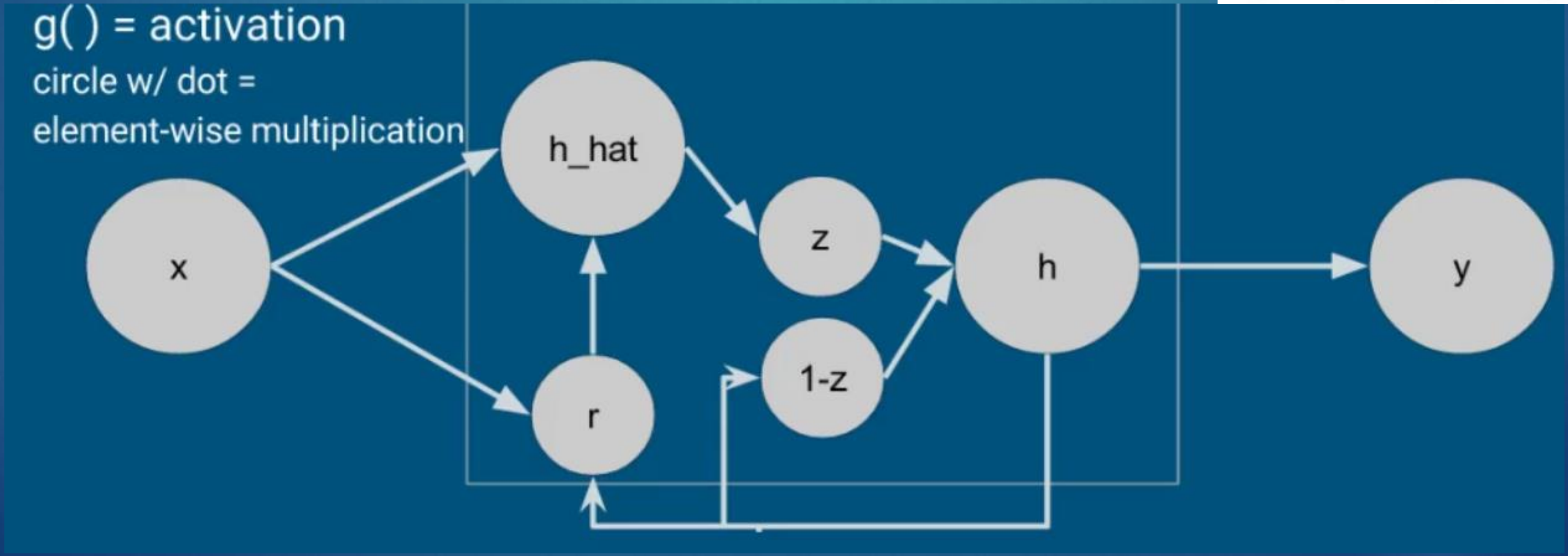
- Simple RNN has lack of variation, since it's multiplying the previous output over and over.
- RRU solves the problem by adding more variation



Gate Recurrent Unit (GRU) - 2014

- Regular RNN has to take entire previous history
  - lots of unnecessary data -> wastes of memory
  - Need to forget some of the data (r = reset gate)

$$
\begin{aligned}
r_t &= \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r) \\
z_t &= \sigma(x_t W_{xz} + h_{t-1} W_{hz} + b_z) \\
\hat{h}_t &= g\,(x_t W_{xh} + (r_t \odot h_{t-1}) W_{hh} + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t.
\end{aligned}
$$



Conclusion

- Why make RNN more complex?
  - Adding more parameters makes the model more expressive and able to solve more complex problem.