Recurrent Neural Networks

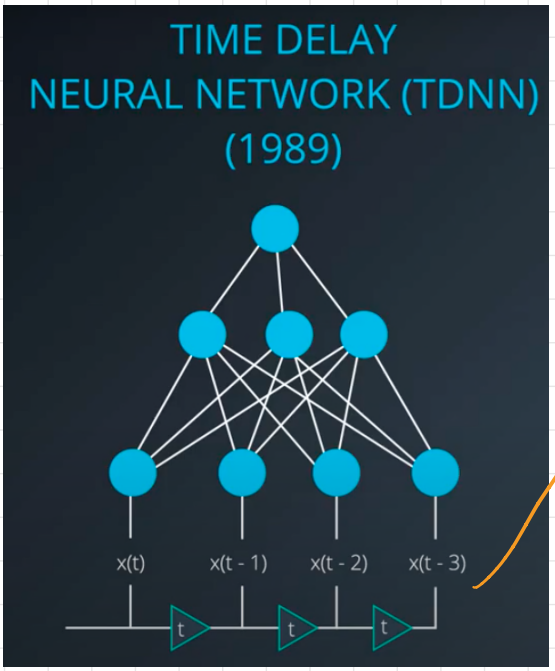
RNN

- CNN excels in tasks that rely on finding **visible** & **spatial** patterns in data
- RNN => incorporating **memory** into our NN
  - Analyze **sequential** data
  - **Text processing / generation**
  - **Sketch RNN** —> generate drawings

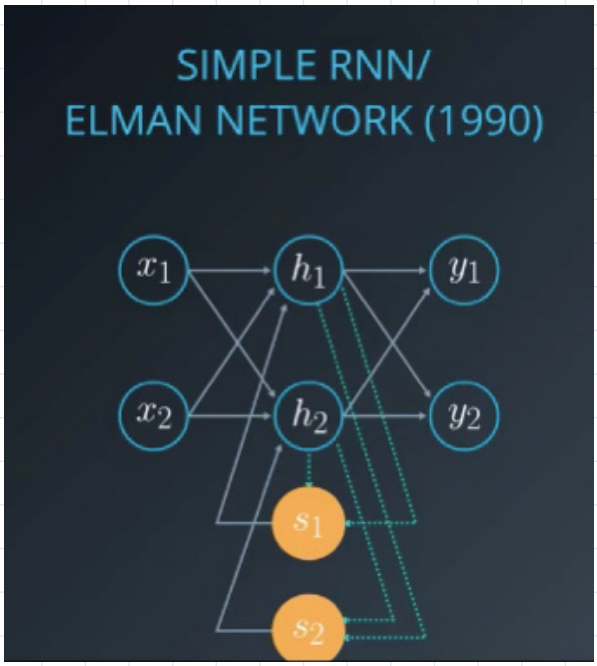
- Temporal Dependencies
  - current output depends on current & **past** input

RNN History

- Feedforward can't detect temporal dependency
- Biological NN have recurrent connections.

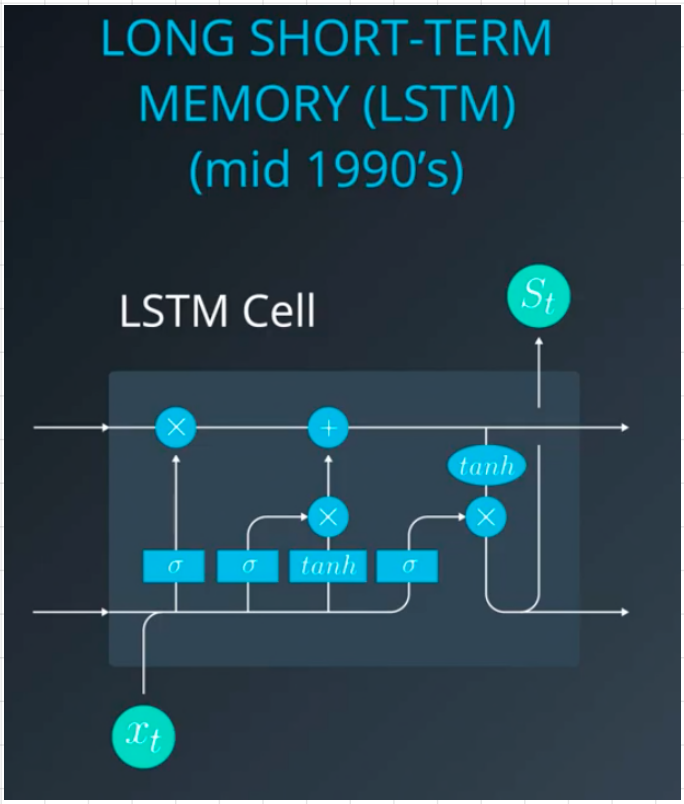


input from past timesteps
But, temp. dependency was limited to the window of time chosen



Vanishing Gradients & Geometric Series

As mentioned in this video, RNNs have a key flaw, as capturing relationships that span more than 8 or 10 steps back is practically impossible. This flaw stems from the "**vanishing gradient**" problem in which the contribution of information decays geometrically over time.



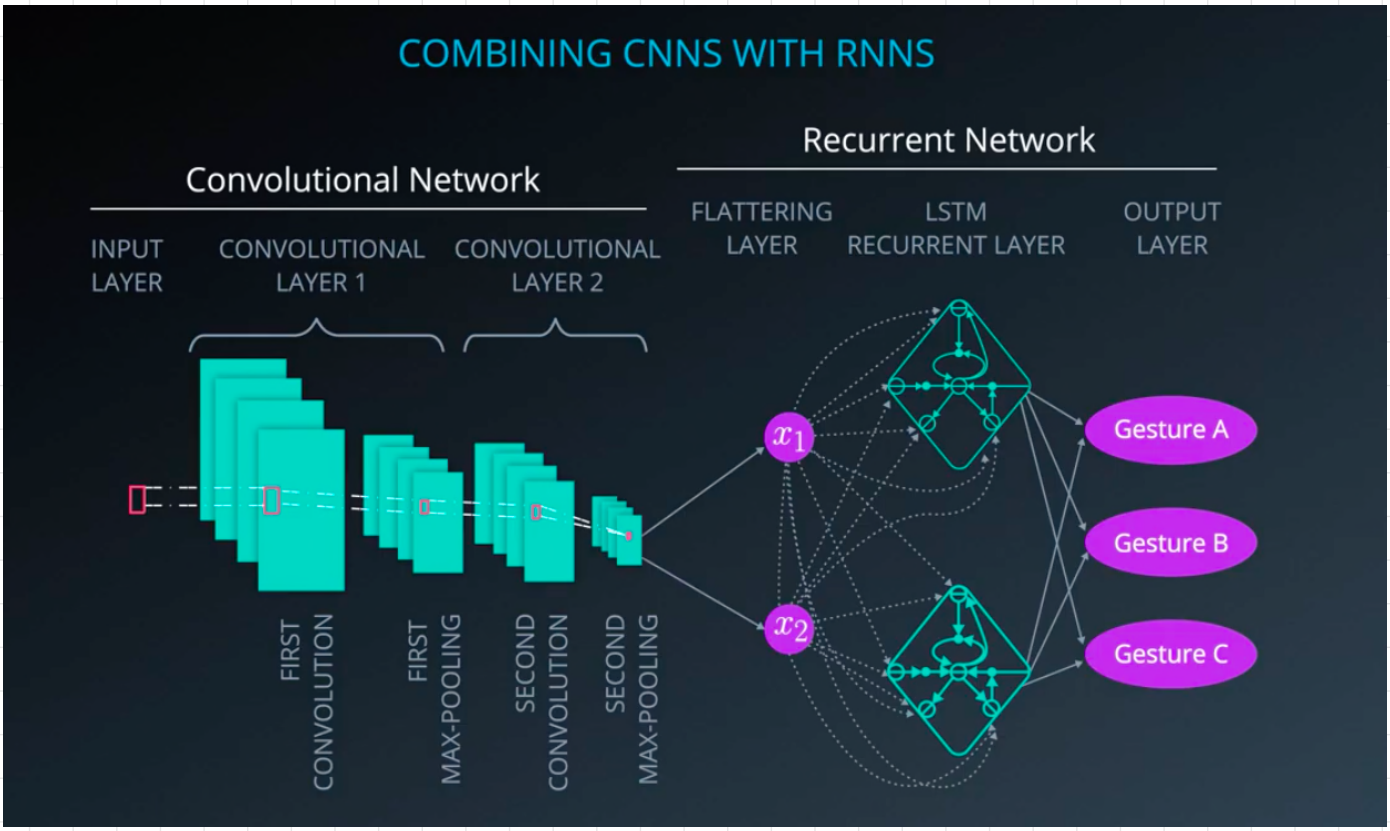
Combat VG.

- Some signals, called state variables, can be kept fixed by using gates
- Arbitary time intervals.
- GRU

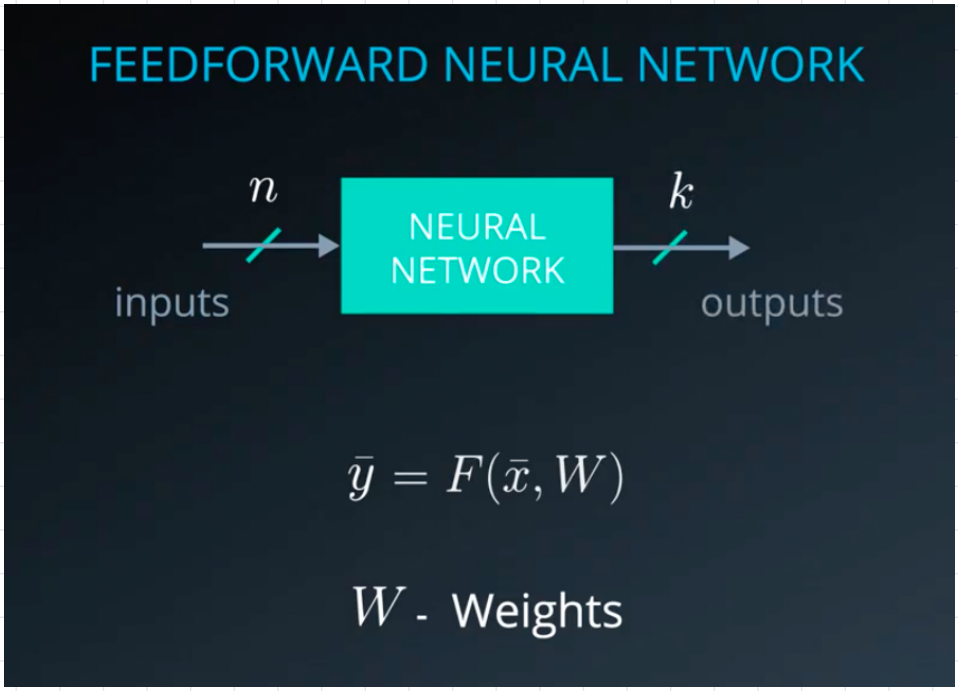
RNN Applications

- Speech recognition → words
  - Siri
- Time Series Prediction
  - Stock
  - movie selection. — Netflix
  - Traffic prediction — Waze
- NLP
  - Machine translation
  - Question answerings (AI)
  - Chat bots
- Gesture Recogmition
  - EyeSight Inc.

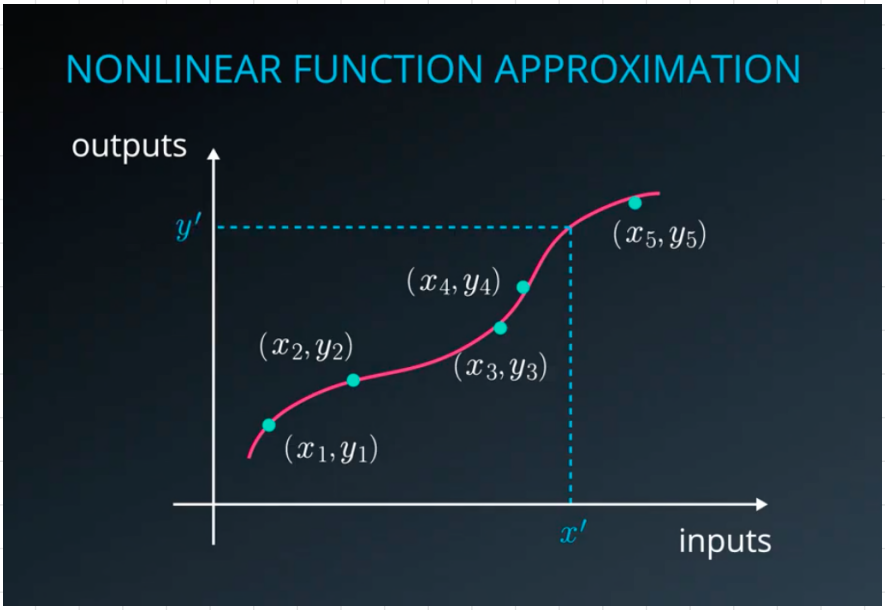
FFNN Review



CNN: Used as a feature extractor
RNN: Used where memory is considered
] Gesture Recog.



Using a non-linear activation function

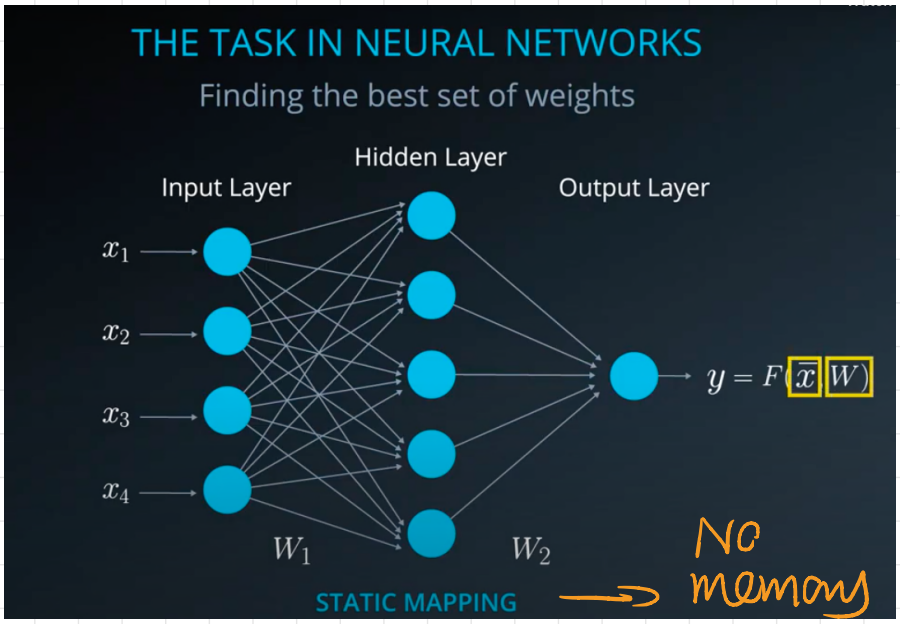


Find a curve
Predict labels.

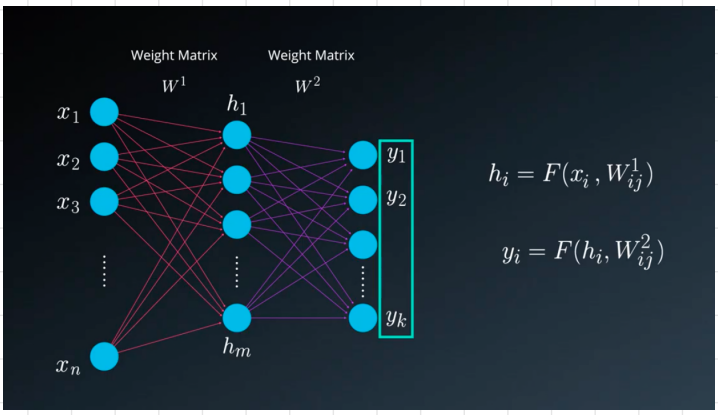
Regression

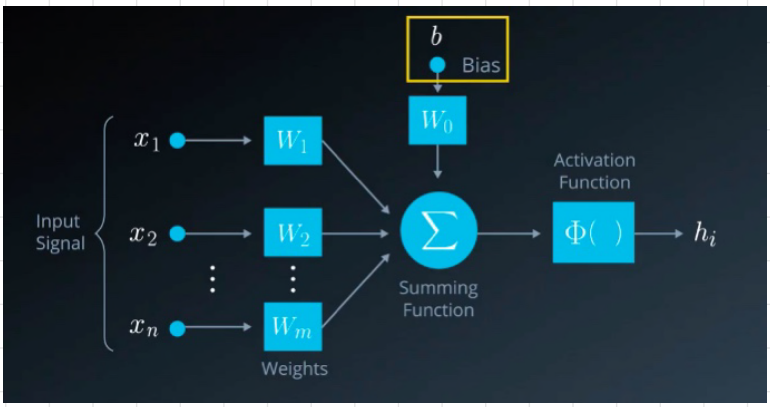
&

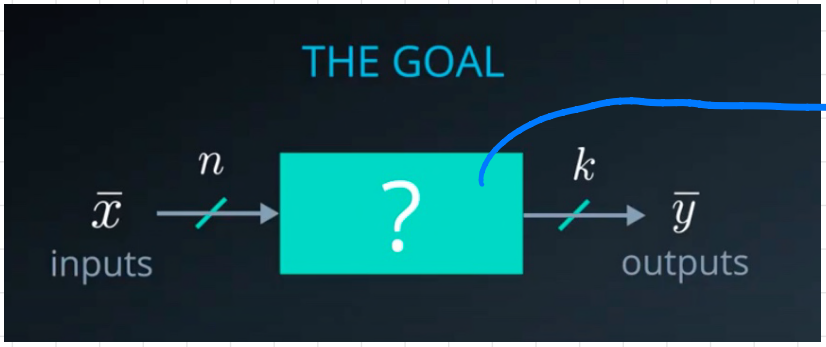
Classification



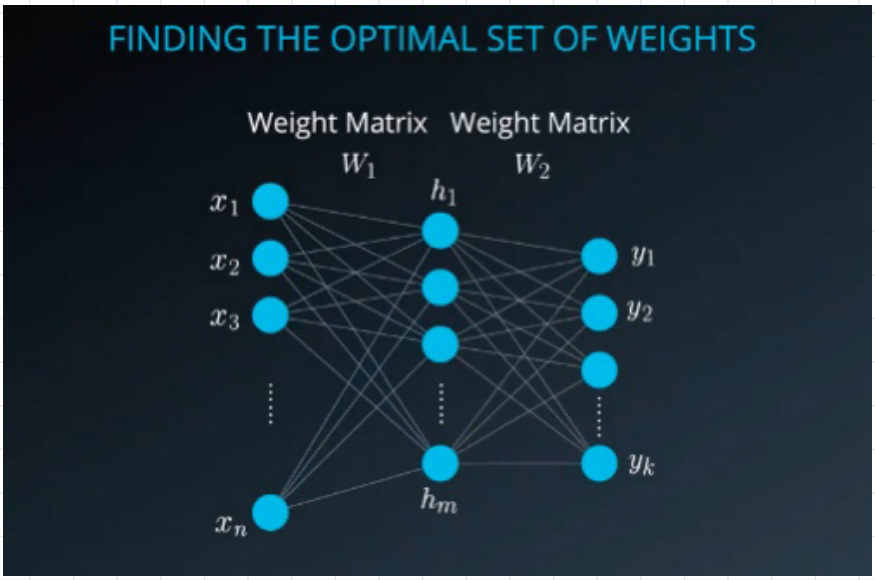
Outputs depend only on the inputs X & weights W



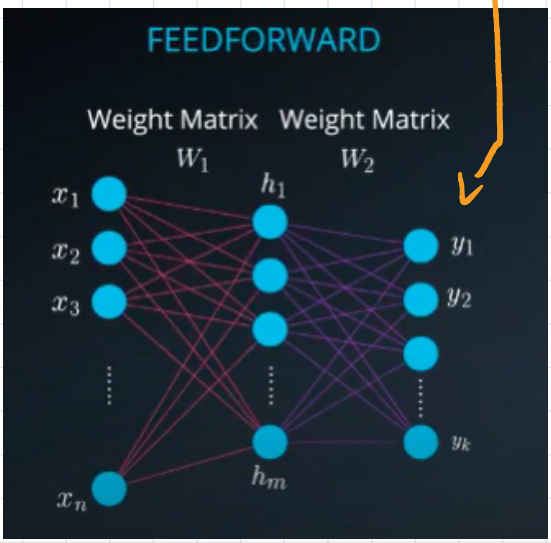


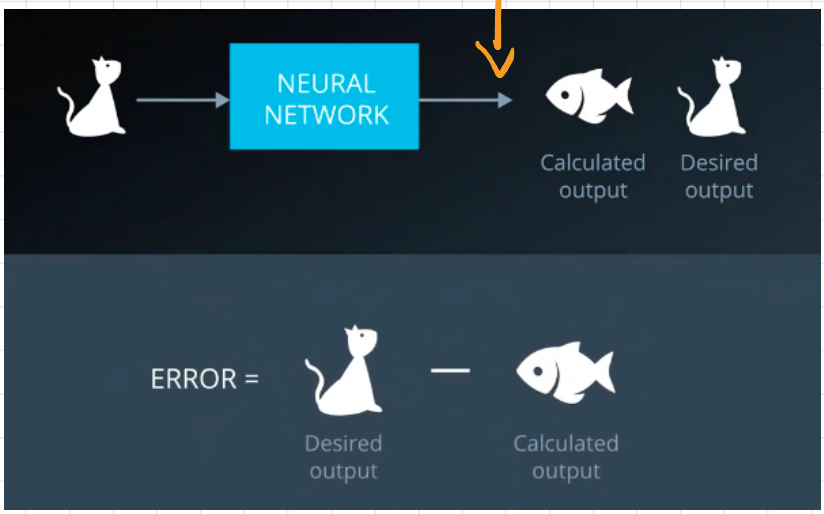


Design the toolbox

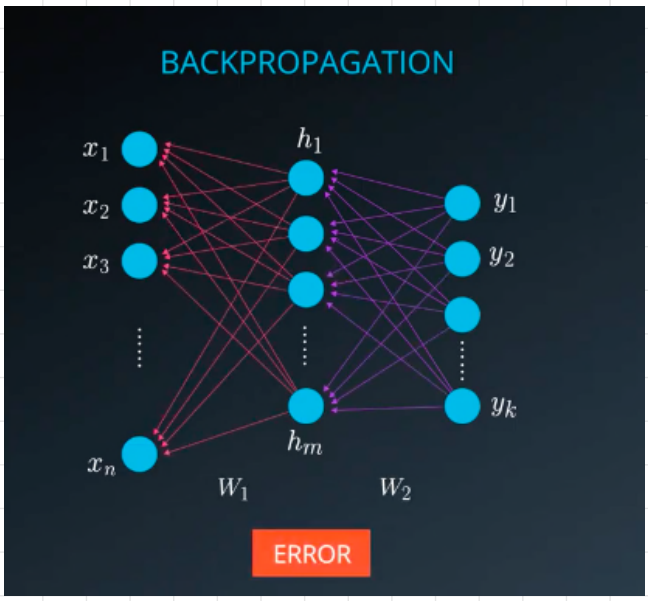


In FF, calculate output & evaluate errors.





In BP, update weights to minimize error.
& start the FF process again!



The Feedforward Process

Step #1

FINDING $\bar{h}$

$$[h'_1\ h'_2\ h'_3] = [x_1\ x_2\ x_3 \dots x_n] \cdot \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \\ \vdots & \vdots & \vdots \\ W_{n1} & W_{n2} & W_{n3} \end{bmatrix}$$

n Rows

3 Columns

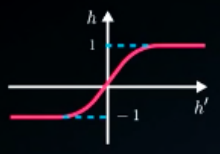
$$\bar{h}' = \bar{x} \cdot W_1$$

- Applying Φ to ensure the outputs don't explode!
- Allowing network to represent **nonlinear** relationship bt. inputs & outputs

FINDING $\bar{h}$

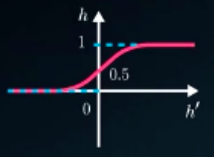
ACTIVATION FUNCTION

$$\bar{h} = \phi(\bar{h}')$$



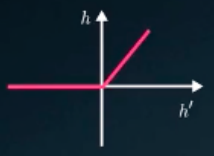
HYPERBOLIC TANGENT FUNCTION

$$f(x) = tanh(x)$$



SIGMOID FUNCTION

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



ReLU FUNCTION

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ x & for\ x \geq 0 \end{cases}$$

$$\bar{h} = \Phi(\bar{x} \cdot W_1)$$

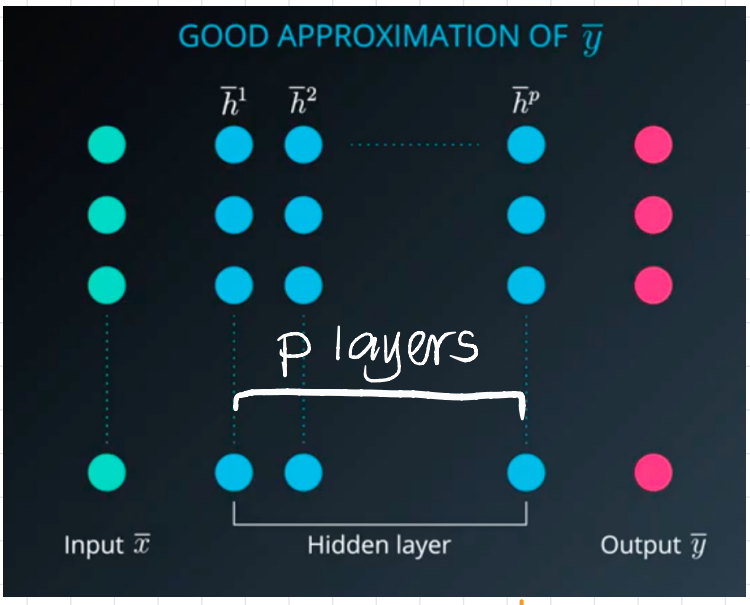
Step #2

$$\bar{y} = \bar{h} \cdot W_2$$

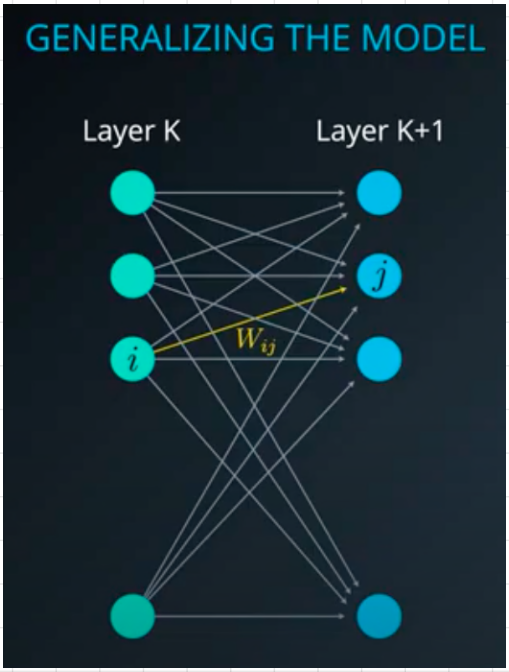
Ensure value $0 \leq value \leq 1$

(+ apply $\sigma(x)$ softmax)

Generalization

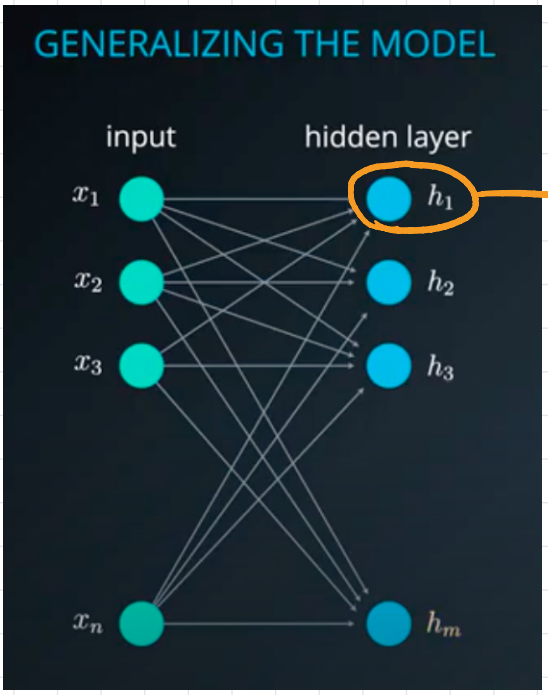


Need more layers
&
more neurons
to get a good
approximation of $\bar{y}$.



$$\bar{h}^p = \Phi(\bar{x} \cdot W_p)$$

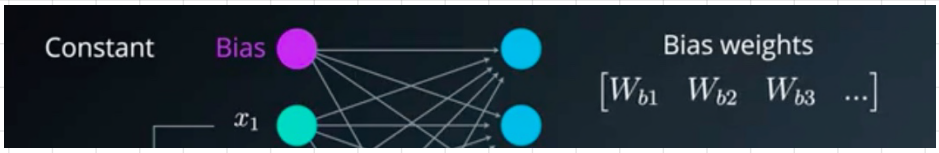
p^{th} layer activation, depends on Φ applied to inputs times the weights connecting 1 layer to the next.



n_x inputs
m_h outputs

$$h_1 = \Phi\left(\sum_i^m x_i W_{i1}\right)$$

$$h_m = \Phi\left(\sum_i^m x_i \cdot W_{im}\right)$$



- constant bias value
- changing bias weights

Errors:

Error: $\bar{e} = (\bar{d} - \bar{y})$

Loss Functions: $E = (\bar{d} - \bar{y})^2$

- MSE
  - Regression
- Cross Entropy Loss
  - Classification

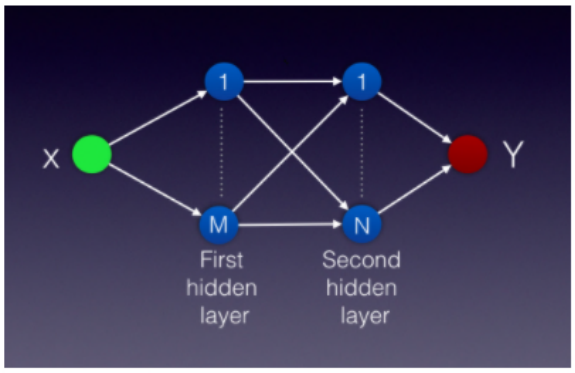
Quiz

The following picture is of a feedforward network with

- A single input x
- Two hidden layers
- A singe output

The first hidden layer has M neurons.

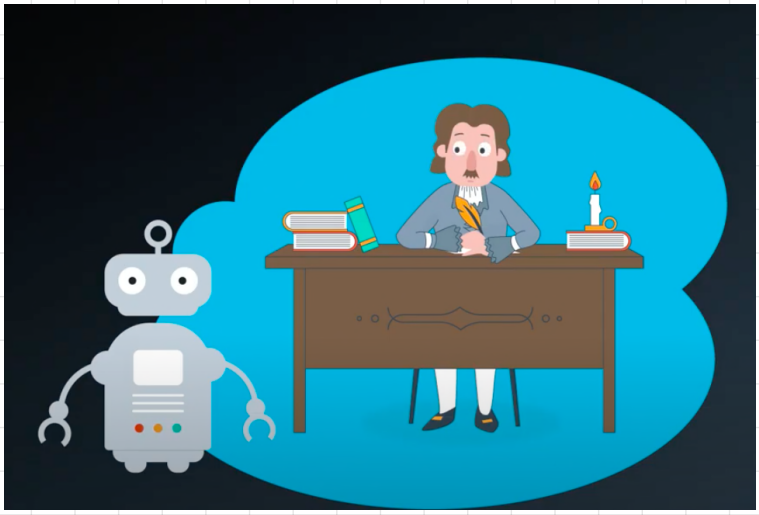
The second hidden layer has N neurons.

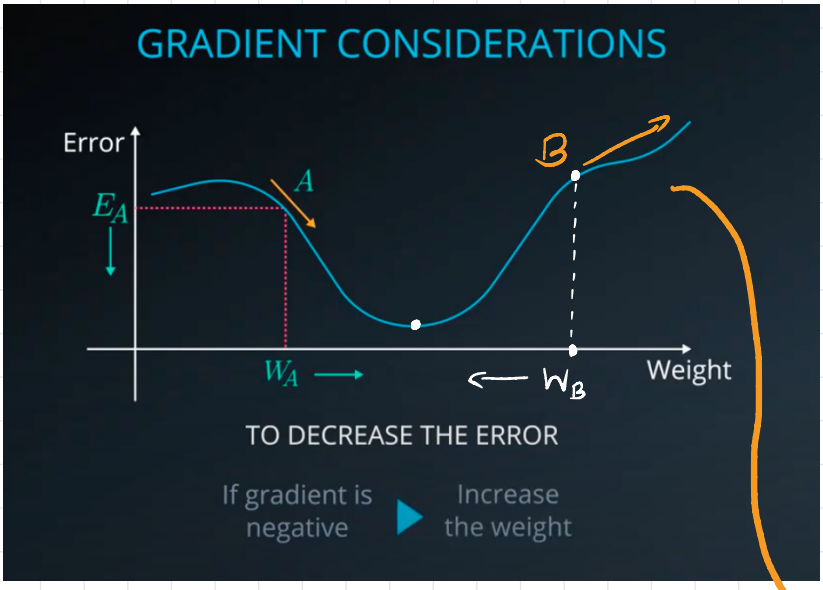


Total Calculations needed : $m + n + mn$.

Backpropagation Review

Evaluate the dependence of error on each weight value and to update the weights to ↓ Loss



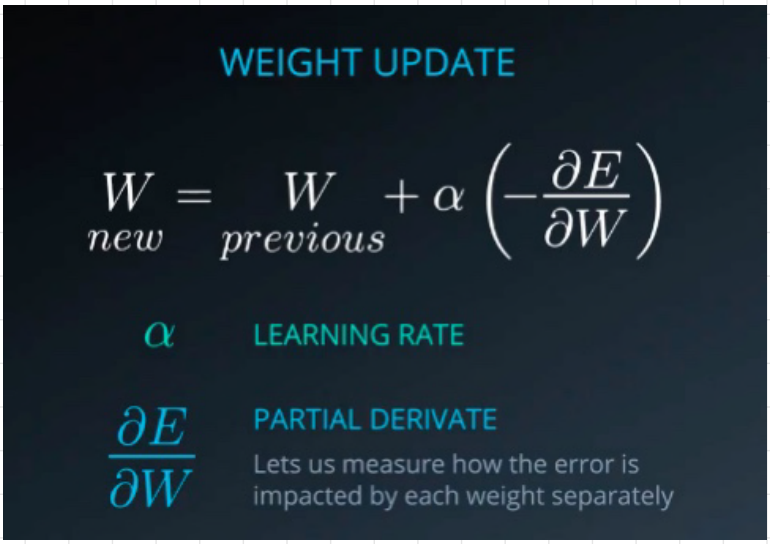


In this case, evaluating $\vec{\nabla}E$ allows us to take a step in the direction of the greatest decrease in the error. thereby ↑ weights to ↓ E.

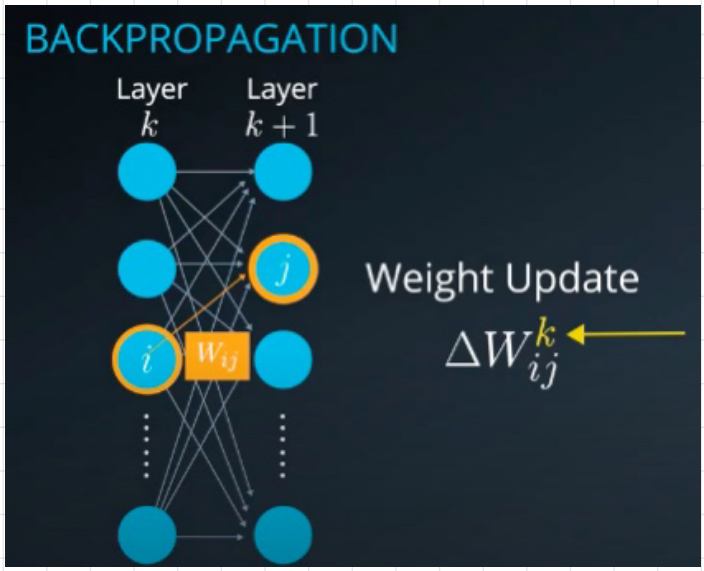
or

↓ weights to ↓ E

Negative direction of $\vec{\nabla}E$

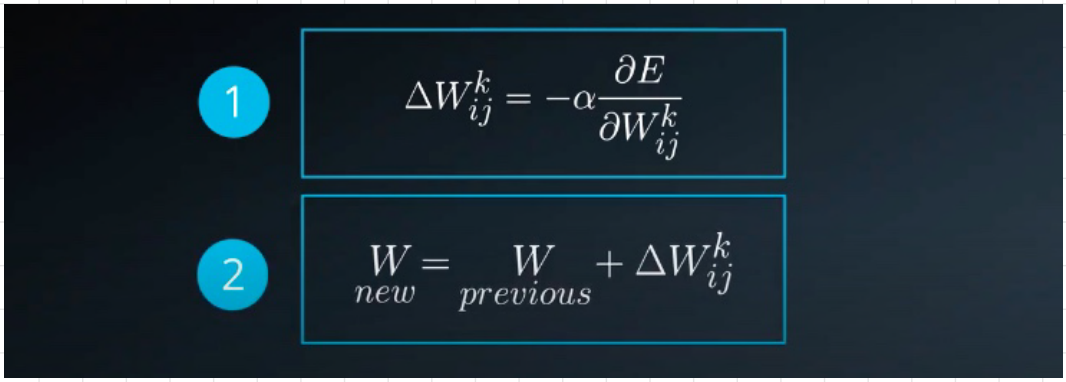


$W = W + \alpha \nabla W(-E)$



W_{ij}^{k}

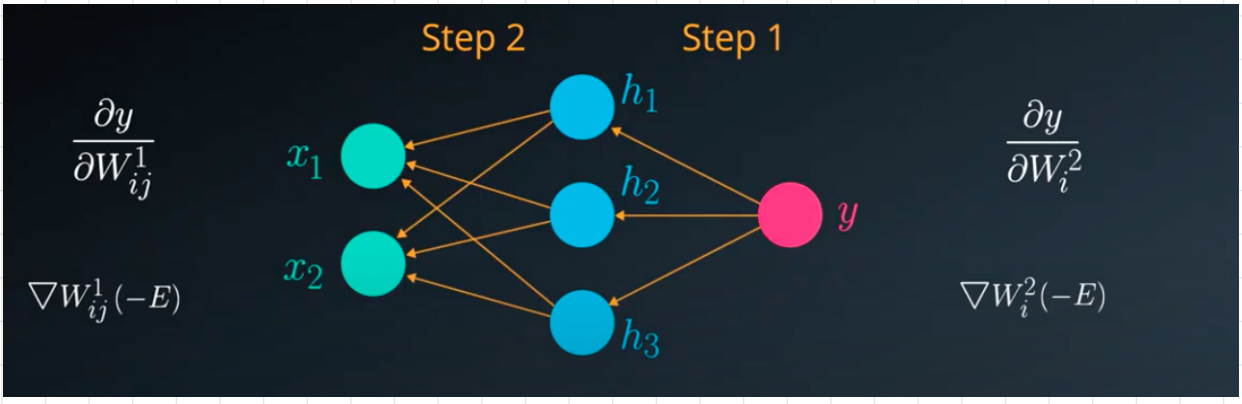
Weight connecting i^{th} neuron to j^{th} neuron originating from k^{th} layer.



$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}} = -\frac{\partial}{2}\frac{\partial (\bar{d}-\bar{y})^2}{\partial w_{ij}} = -\partial \cdot \frac{\partial (\bar{d}-\bar{y})}{\partial w_{ij}} (\bar{d}-\bar{y})$$

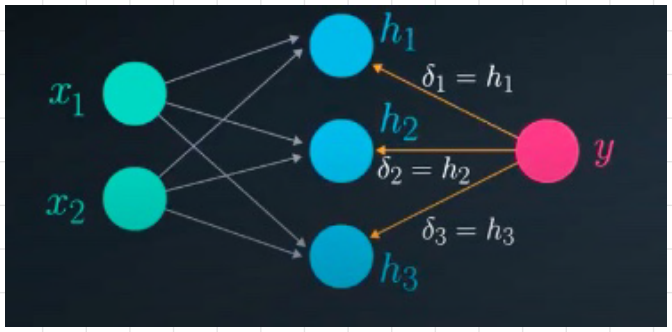
$$\Delta w_{ij} = \partial (\bar{d}-\bar{y}) \frac{\partial \bar{y}}{\partial w_{ij}}$$

Where $\delta_{ij} = \frac{\partial \bar{y}}{\partial w_{ij}}$

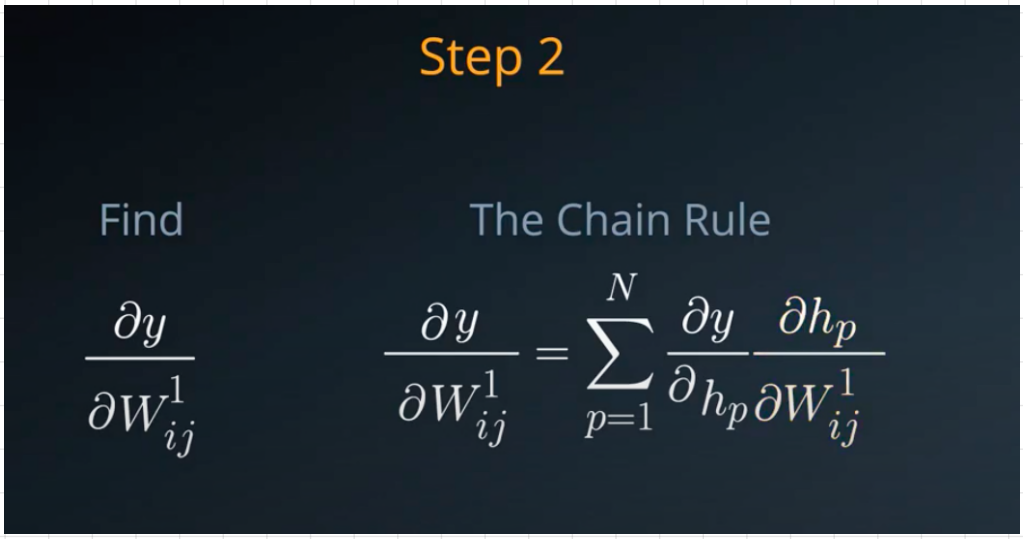


Step 2. Update wts of Layer 1.

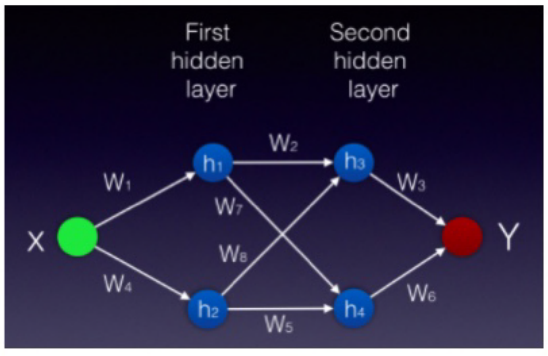
$$\frac{\partial y}{\partial w_i^2} = \frac{\partial \sum_i^n h_i w_i^2}{\partial w_i^2} = h_i$$



Step 2. Layer 1 update.



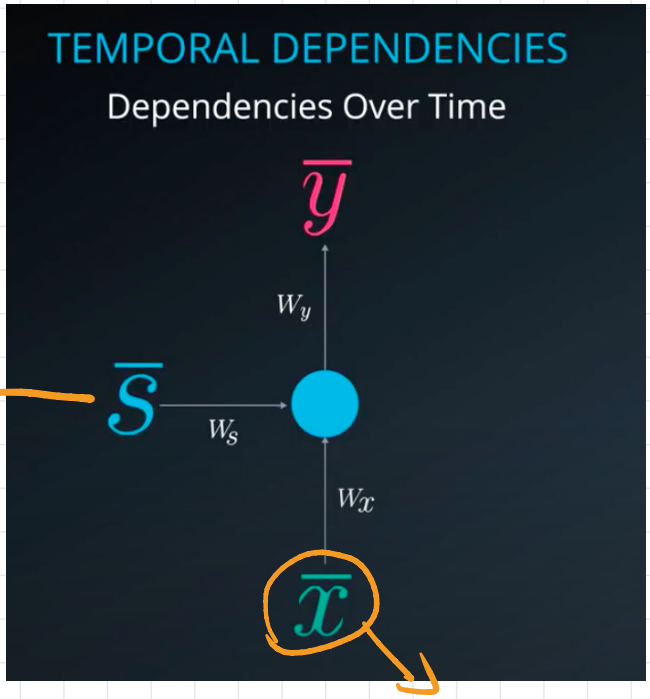
Example:



Equation A $\frac{\partial y}{\partial W_1} = \frac{\partial y}{\partial h_3}\frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial W_1} + \frac{\partial y}{\partial h_4}\frac{\partial h_4}{\partial h_1}\frac{\partial h_1}{\partial W_1}$

RNN Basics

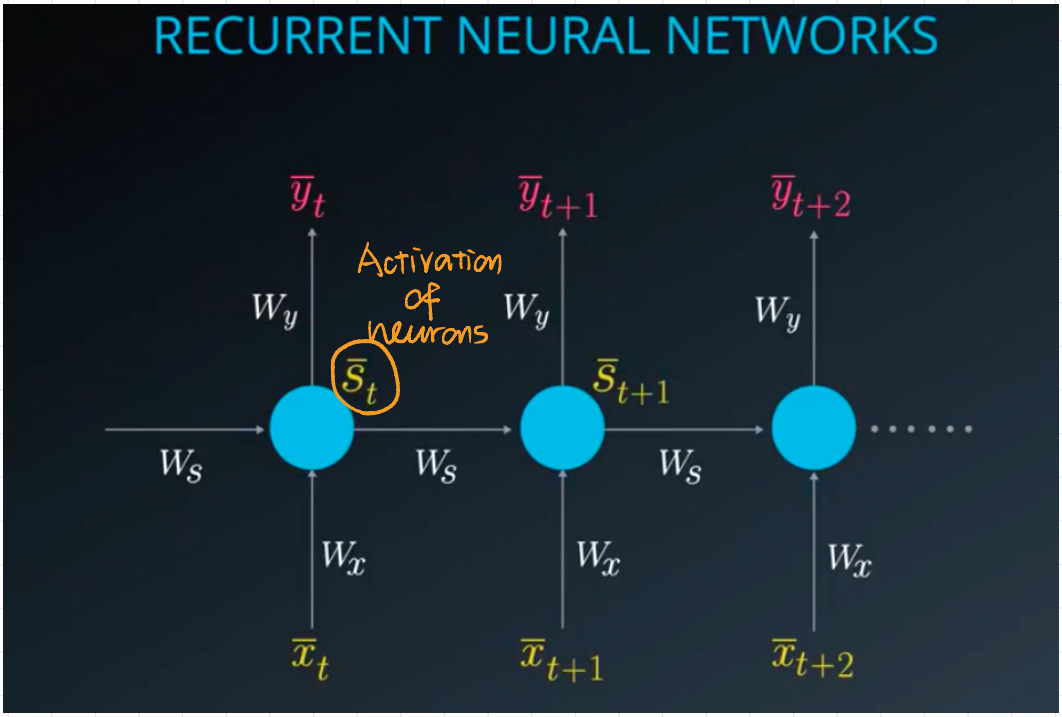
RNN I

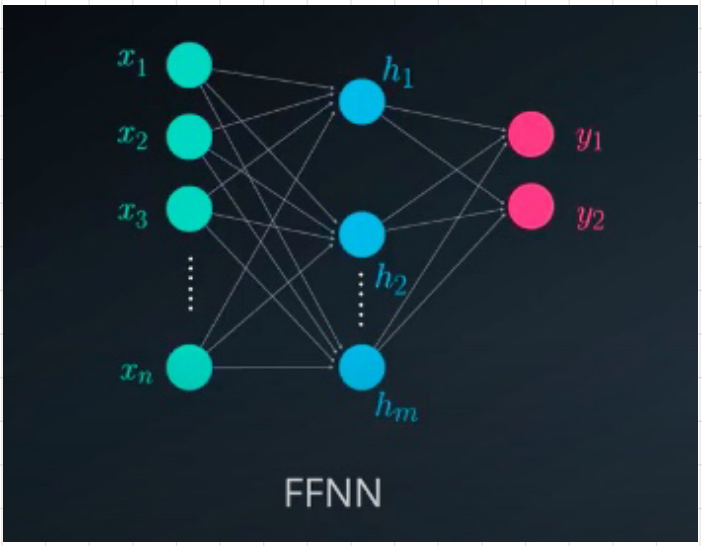


Memory

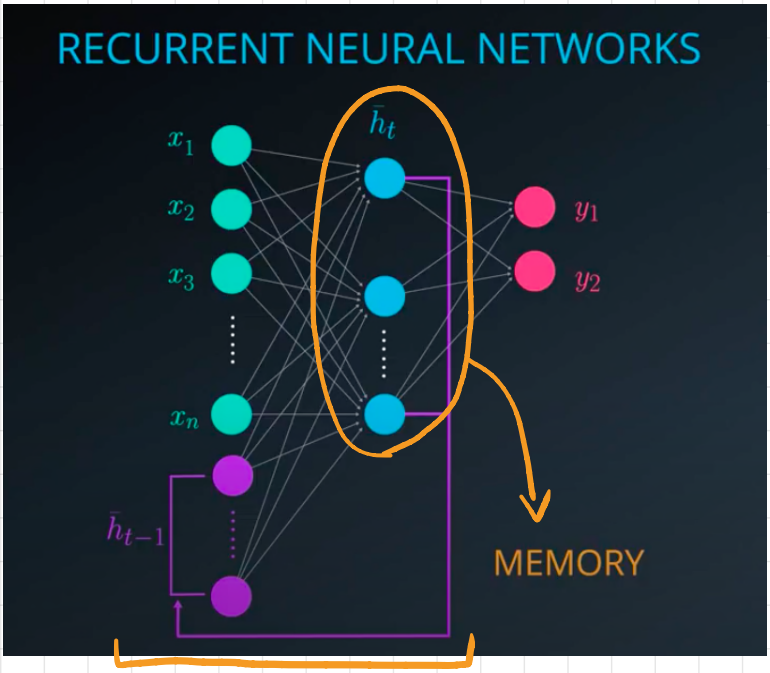
current input

E.g.
Predicting the next word in a sentence



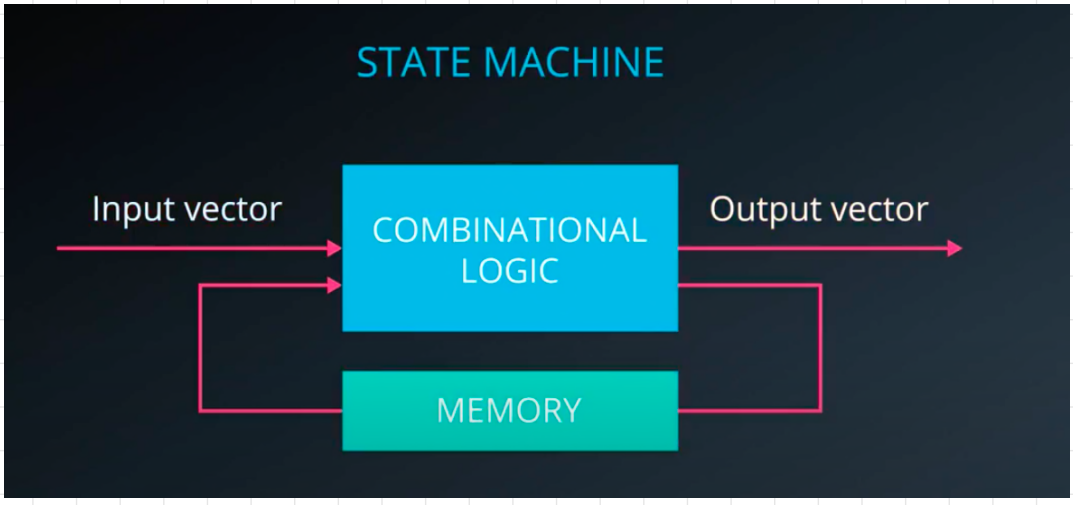


No feedback



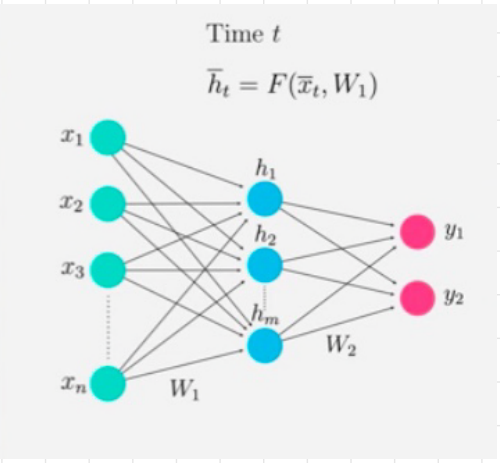
include the feedback/ memory element.

- Memory ≡ output of hidden layer, serve as additional input units at following hidden steps.
- No longer use "$\bar{h}$" as activation of neurons, but use "$\bar{S}_t$" as state (system w/ memory)

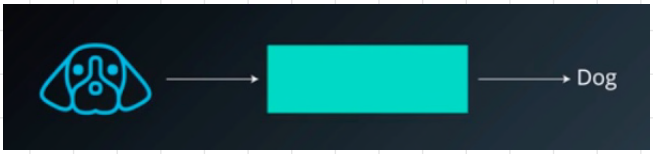


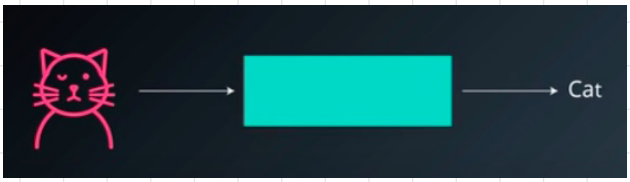
Just like a state machine but only more intelligent and more elegant.

Training itself.



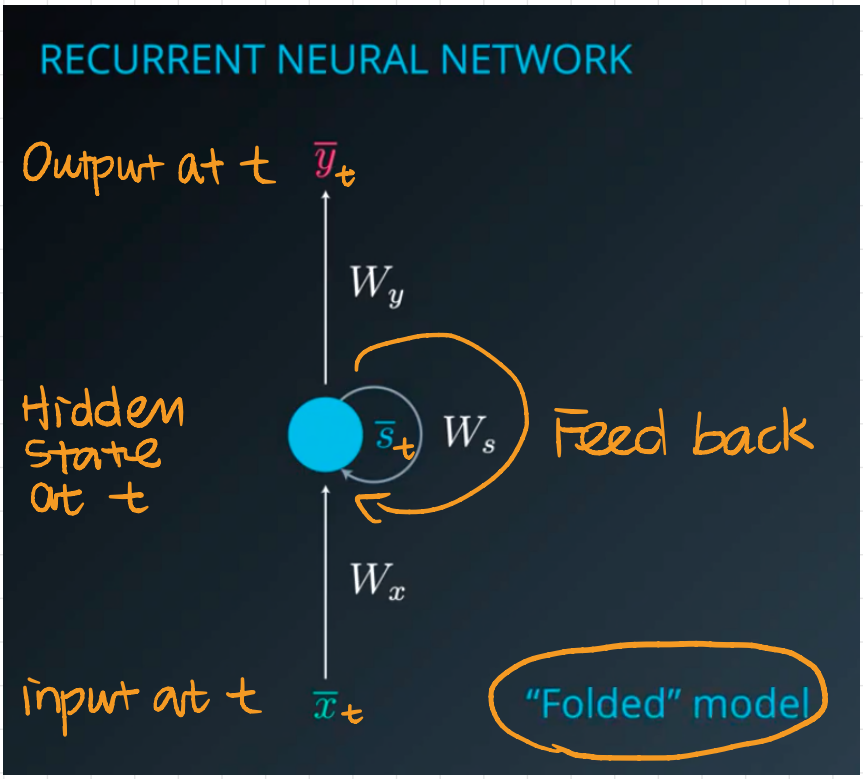
In FFNN, output depends only on weights and current input



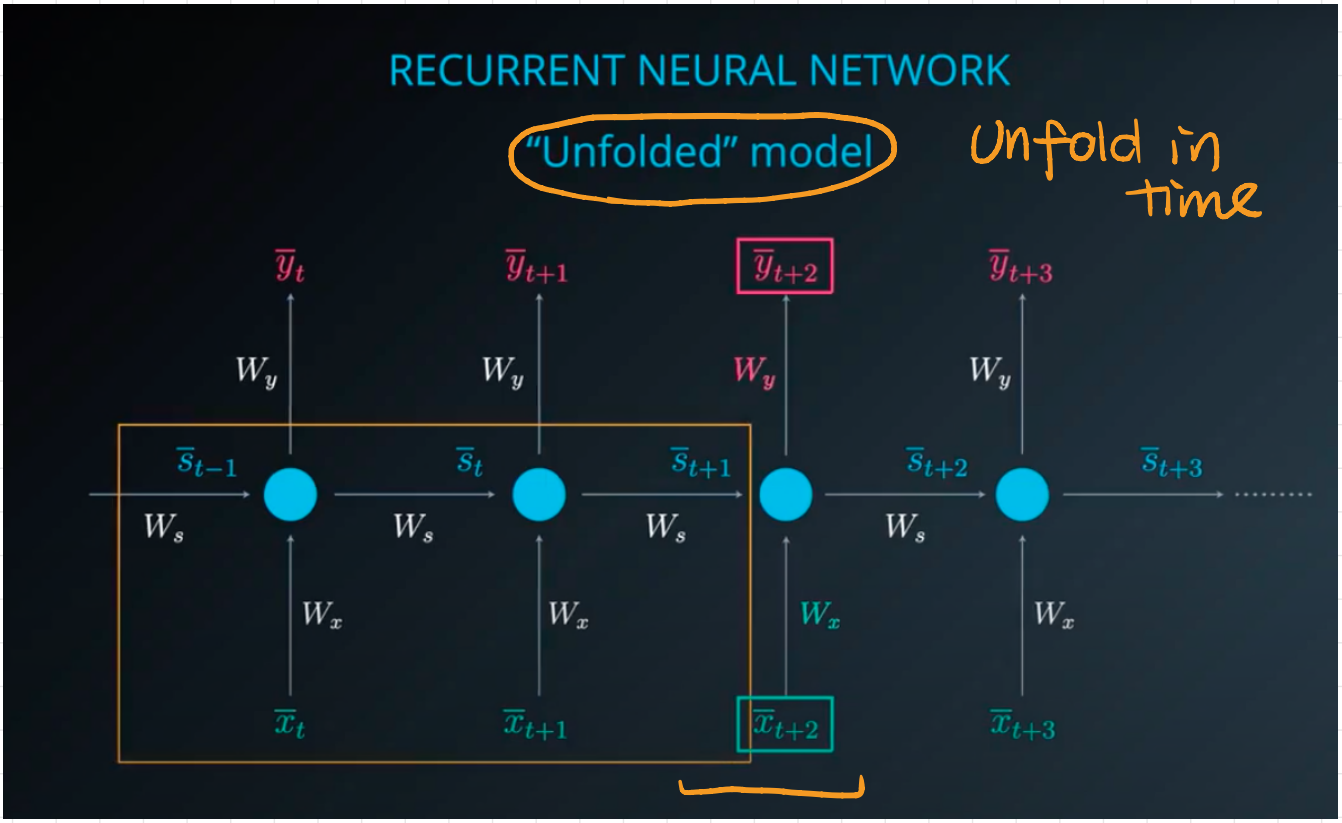


"Assume the inputs are independent of each other, i.e. no significance to the sequence so we train the system by randomly drawing input & target pairs!

In contrast



Basically, all steps at some time looks exactly the same.



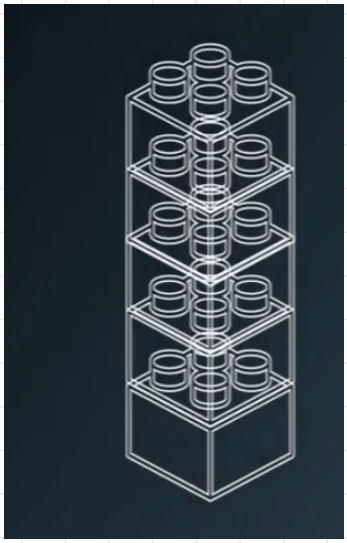
Depends on not only $\bar{x}_{t+2}$, but also all previous inputs

FFNN v.s. RNN

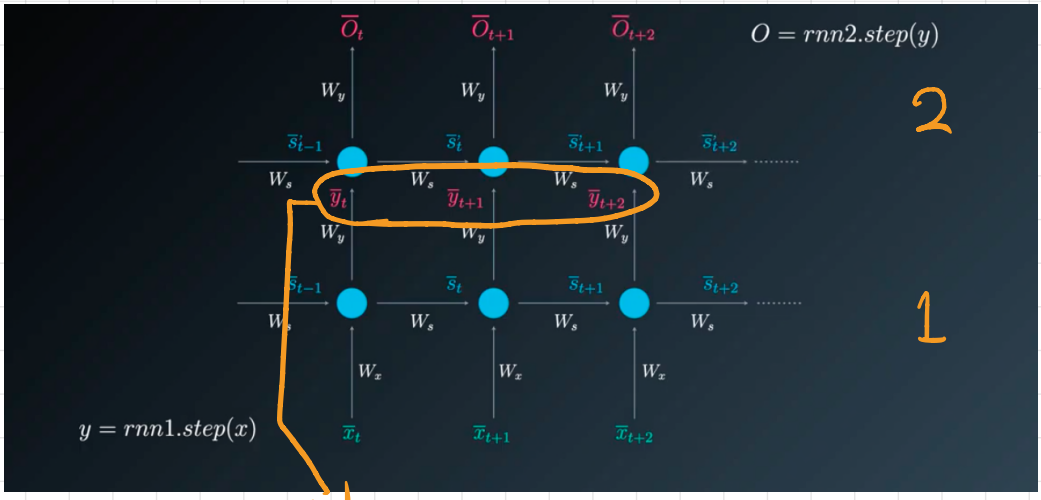
$\bar{h} = \Phi(\bar{x} \cdot W_x)$

$\bar{S}_t = \Phi(\bar{x}_t \cdot W_x + \bar{x}_{t-1} \cdot W_s)$

$\bar{y}_t = \sigma(\bar{S}_t \cdot W_y)$



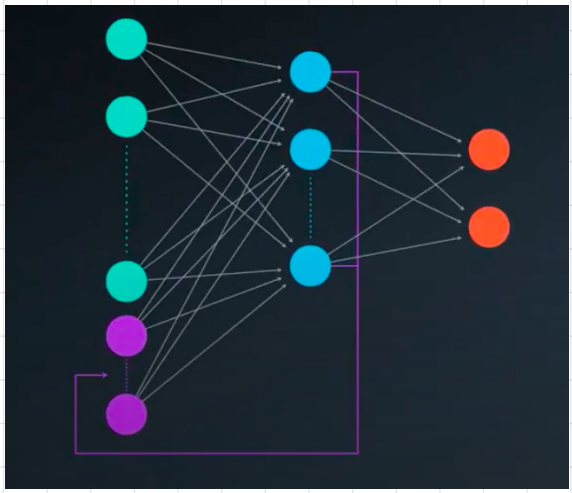
RNN can be stacked.

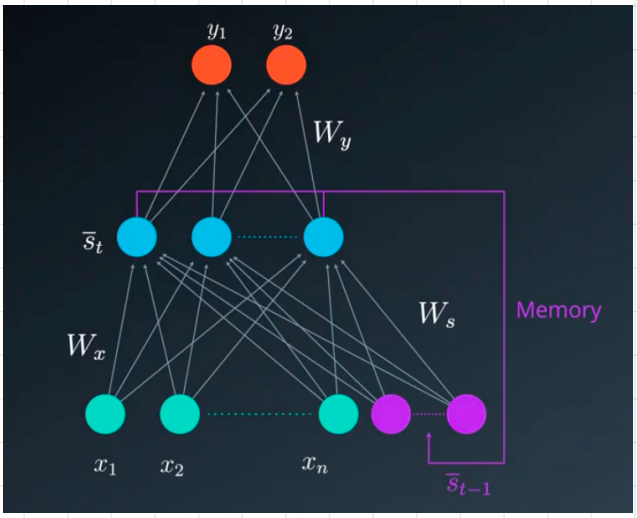


Output from layer 1 can be input for layer 2.

The Unfolded Model

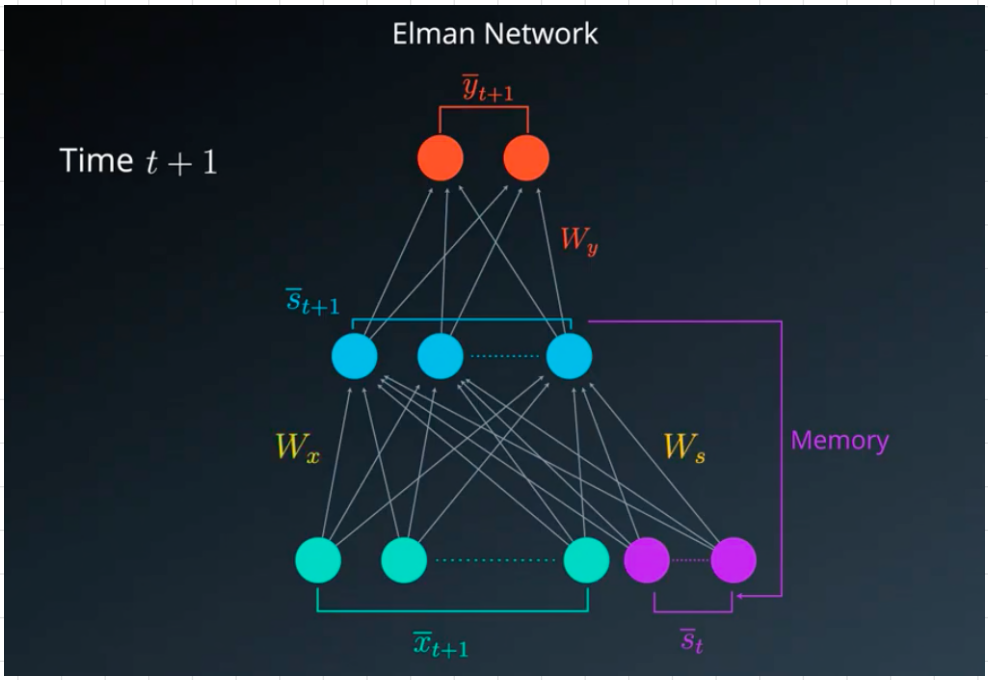
The Elman Model → Rotate 90°





Time t

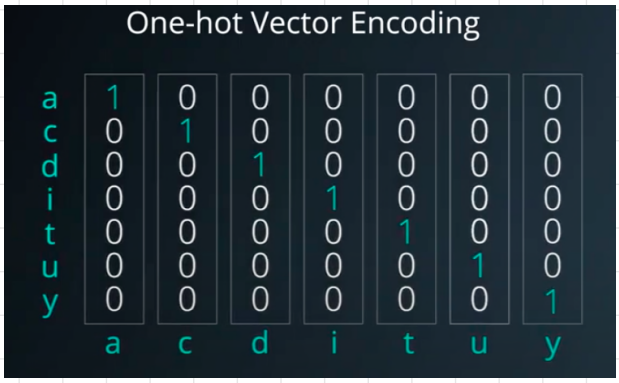
- Input vector $\bar{X}_t$ connect to the state by W_x
- State vector $\bar{S}_{t-1}$ ⟺ state by W_s
- $\bar{X}_t$ & $\bar{S}_{t-1}$ produce desired state S_t
- S_t ⟺ $\bar{y}_t$ by W_y

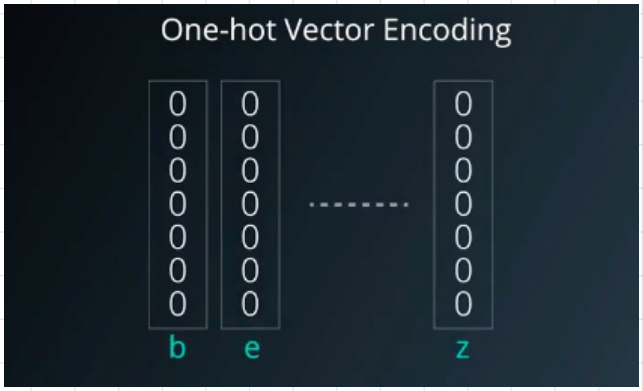


At Time t+1

- Weights remain the same
- Input → $\bar{X}_{t+1}$
- State vector ↳ $\bar{S}_t$
- State → $\bar{S}_{t+1}$
- Output → $\bar{y}_{t+1}$

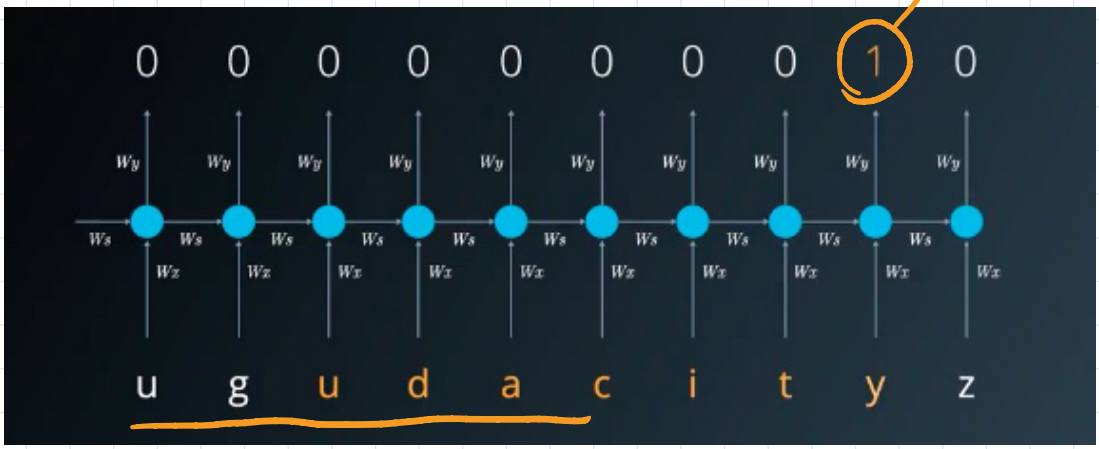
RNN Example:



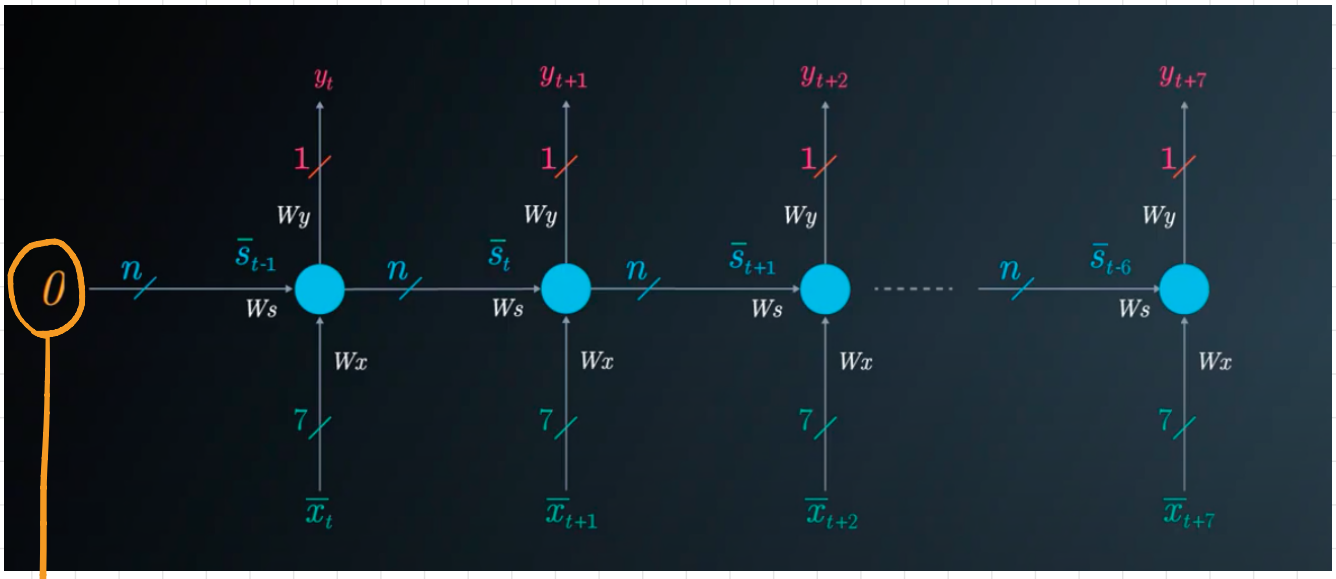


We want

Equals 1 only when desired sequence is detected



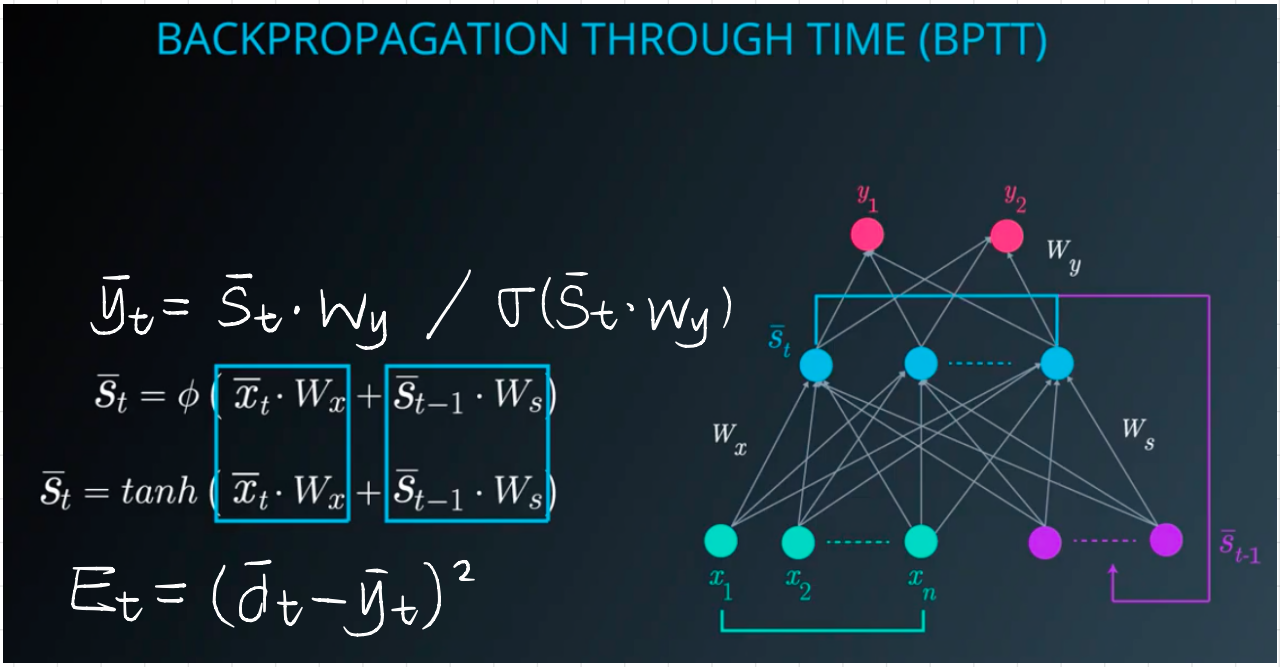
Random, occasionally insert the desired sequence.



- 1st State vector set to 0, allow the next $\bar{S}_t$ to evolve.
- Set target to $\begin{cases} 0, \text{ when "Udacity" not detected} \\ 1, \text{ when detected} \end{cases}$
- Output also take on values $0 \leq \bar{y} \leq 1$

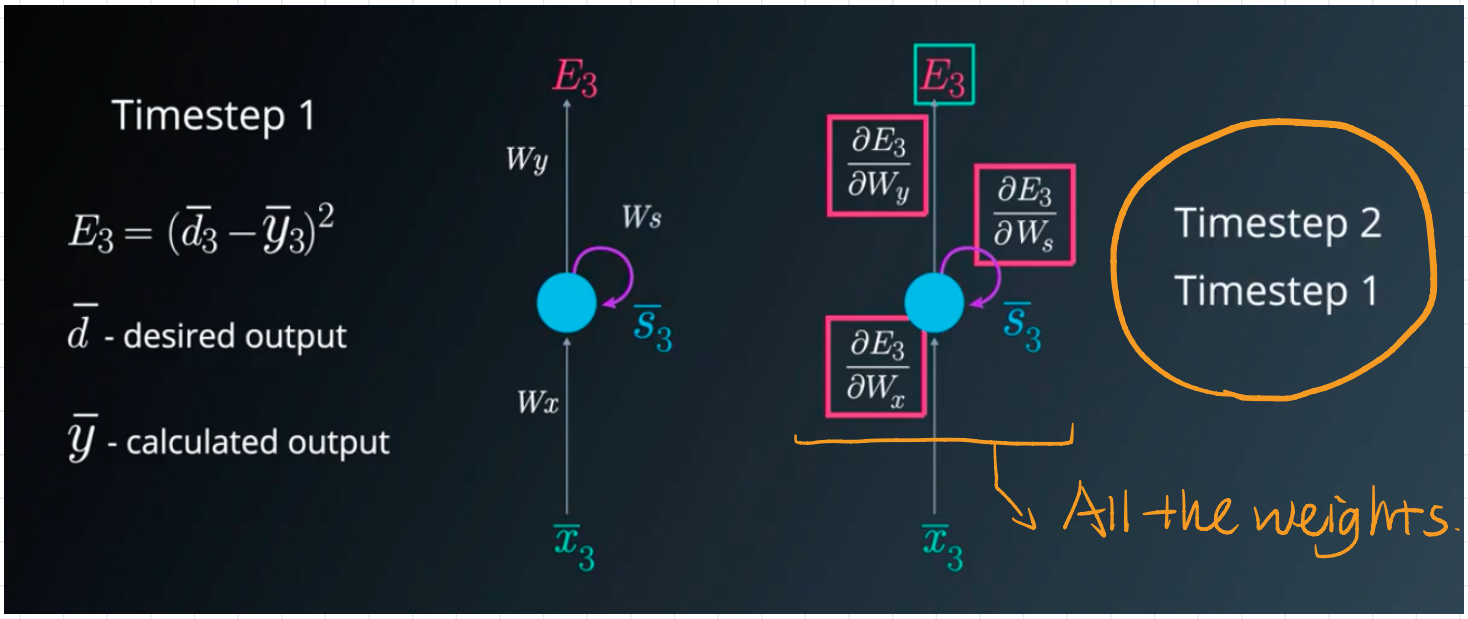
Set threshold = 0.9. then if $\bar{y}$ > threshold, sequence is detected.

Backprop. Through Time



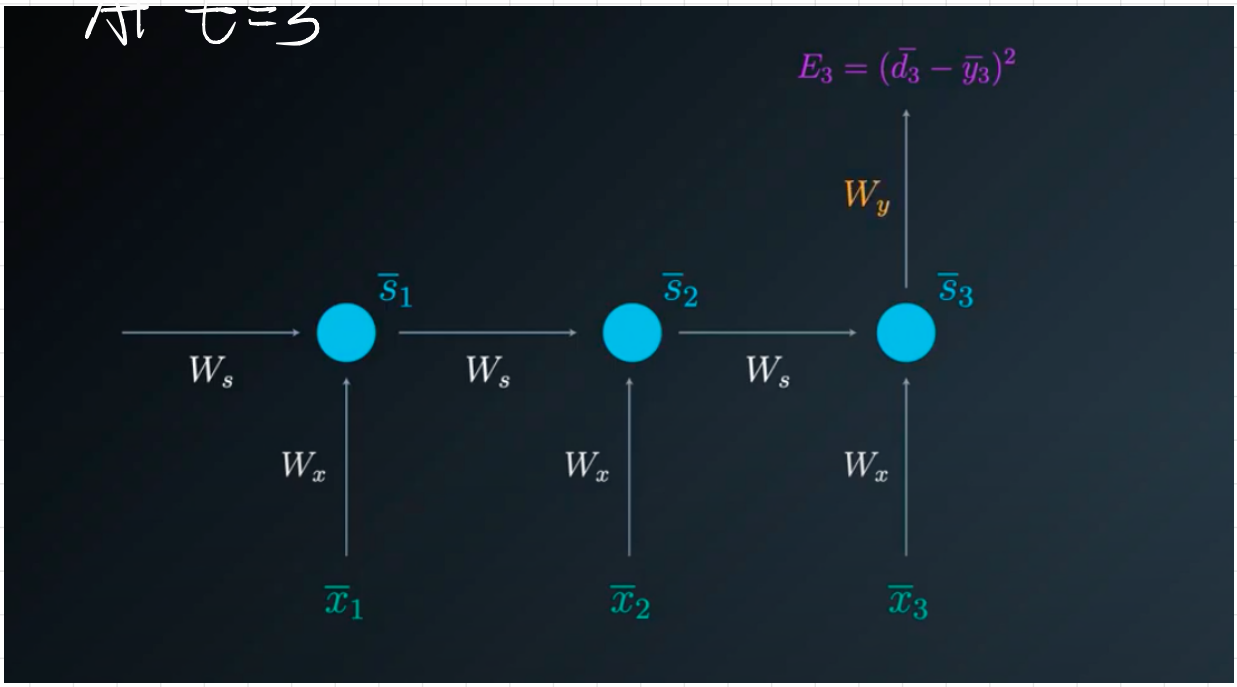
In BPTT, we don't independently train the system at a specific time t. instead take into account all that has happened before.

t, t-1, t-2, t-3 - - -



Adjusting Ws

At $t=3$



$$\frac{\partial E_3}{\partial W_y} = \frac{\partial E_3}{\partial \bar{y}_3} \frac{\partial \bar{y}_3}{\partial W_y}$$

$\frac{\partial E_3}{\partial W_s}$ (Accumulative Gradient!)

Adjusting Weight Matrix W_s

Accumulative Gradient at time $t = 3$

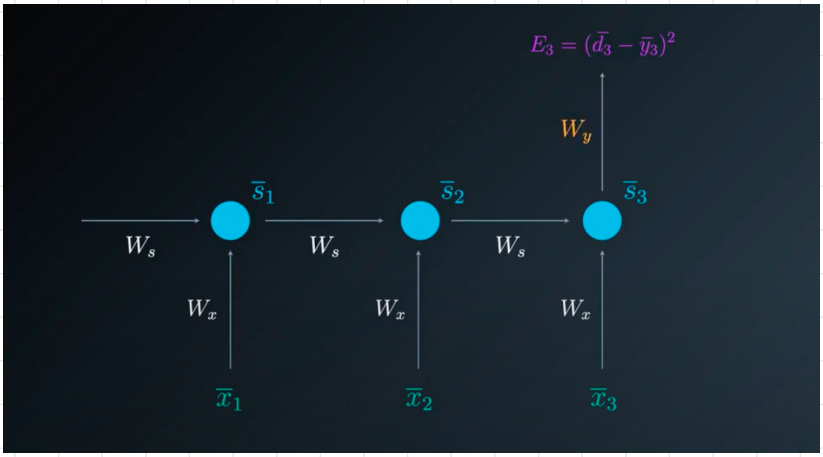
$$\frac{\partial E_3}{\partial W_s} = \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial W_s} + \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial W_s} + \frac{\partial E_3}{\partial \bar{y}_3} \cdot \frac{\partial \bar{y}_3}{\partial \bar{s}_3} \cdot \frac{\partial \bar{s}_3}{\partial \bar{s}_2} \cdot \frac{\partial \bar{s}_2}{\partial \bar{s}_1} \cdot \frac{\partial \bar{s}_1}{\partial W_s}$$

Considering $\bar{S_3}$ Considering $\bar{S_2}$ Considering $\bar{S_1}$

Adjusting Weight Matrix W_s

$$\frac{\partial E_N}{\partial W_s} = \sum_{i=1}^{N} \frac{\partial E_N}{\partial \bar{y}_N} \cdot \frac{\partial \bar{y}_N}{\partial \bar{s}_i} \cdot \frac{\partial \bar{s}_i}{\partial W_s}$$

Adjusting W_x



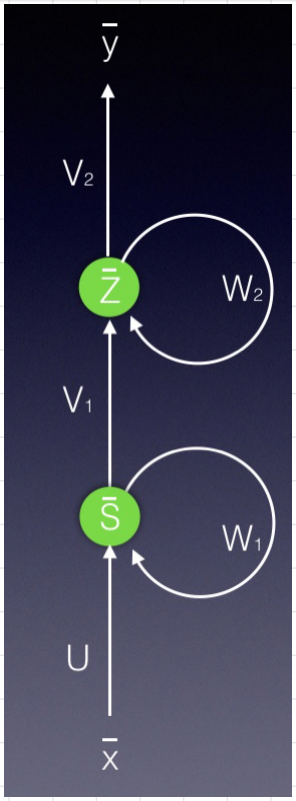
$$\frac{\partial E_3}{\partial W_x} = \frac{\partial E_3}{\partial \bar{y}_3}\frac{\partial \bar{y}_3}{\partial \bar{s}_3}\frac{\partial \bar{s}_3}{\partial W_x}$$

$$+ \frac{\partial E_3}{\partial \bar{y}_3}\frac{\partial \bar{y}_3}{\partial \bar{s}_3}\frac{\partial \bar{s}_3}{\partial \bar{s}_2}\frac{\partial \bar{s}_2}{\partial W_x}$$

$$+ \frac{\partial E_3}{\partial \bar{y}_3}\frac{\partial \bar{y}_3}{\partial \bar{s}_3}\frac{\partial \bar{s}_3}{\partial \bar{s}_2}\frac{\partial \bar{s}_2}{\partial \bar{s}_1}\frac{\partial \bar{s}_1}{\partial W_x}$$

$$\frac{\partial E_N}{\partial W_x} = \sum_{i=1}^{N} \frac{\partial E_N}{\partial \bar{y}_N}\frac{\partial \bar{y}_N}{\partial \bar{s}_i}\frac{\partial \bar{s}_i}{\partial W_x}$$

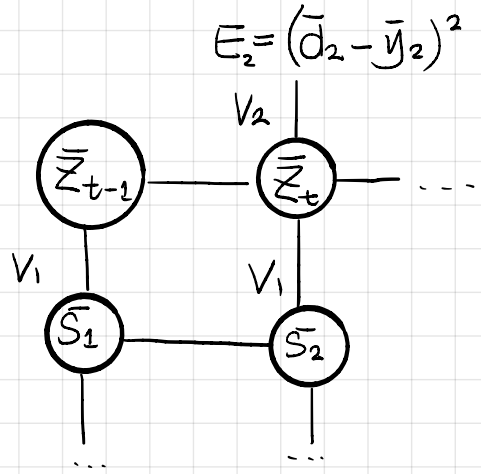
Quizzes:



#1. Represent $\bar{Z}$

$$\bar{Z}_t = (\bar{S}_t \cdot V_1) + (\bar{Z}_{t-1} \cdot W_2)$$

#2. Update rule for V_1, $\frac{\partial E}{\partial V_1}$

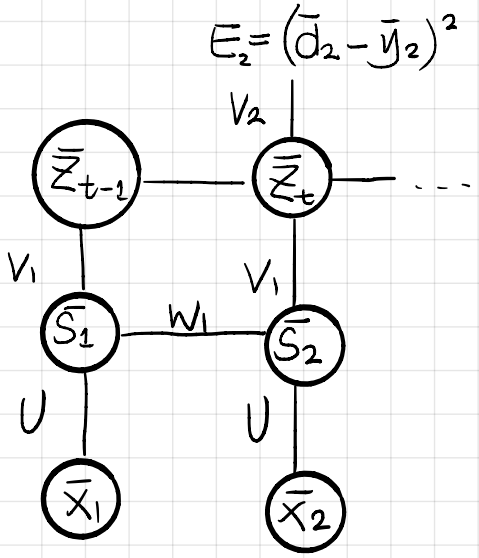


$$\frac{\partial E_2}{\partial V_1} = \frac{\partial E_2}{\partial \bar{y}_2}\frac{\partial \bar{y}_2}{\partial \bar{Z}_2}\frac{\partial \bar{Z}_2}{\partial V_1}$$

$$+ \frac{\partial E_2}{\partial \bar{y}_2}\frac{\partial \bar{y}_2}{\partial \bar{Z}_2}\frac{\partial \bar{Z}_2}{\partial Z_1}\frac{\partial Z_1}{\partial V_1}$$

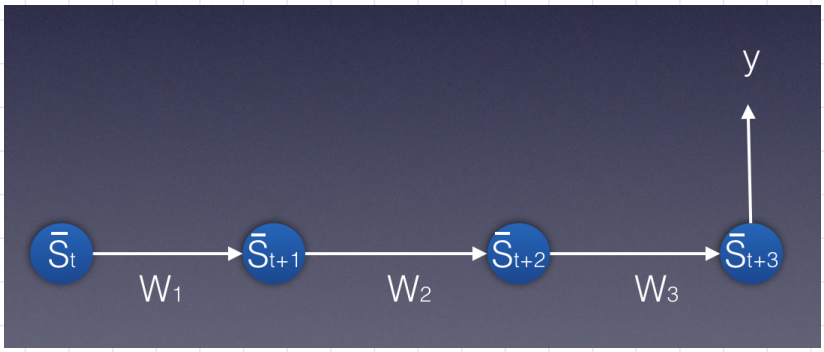
$$\frac{\partial E_t}{\partial V_1} = \frac{\partial E_t}{\partial \bar{y}_t}\frac{\partial \bar{y}_t}{\partial \bar{Z}_i}\frac{\partial \bar{Z}_i}{\partial V_1}$$

#3. Update rule for U



$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial \bar{y}_2}\frac{\partial \bar{y}_2}{\partial \bar{z}_2}\frac{\partial \bar{z}_2}{\partial \bar{s}_2}\frac{\partial \bar{s}_2}{\partial U} + \frac{\partial E_2}{\partial \bar{y}_2}\frac{\partial \bar{y}_2}{\partial \bar{z}_2}\frac{\partial \bar{z}_2}{\partial \bar{s}_2}\frac{\partial \bar{s}_2}{\partial \bar{s}_1}\frac{\partial \bar{s}_1}{\partial U} + \frac{\partial E_2}{\partial \bar{y}_2}\frac{\partial \bar{y}_2}{\partial \bar{z}_2}\frac{\partial \bar{z}_2}{\partial \bar{z}_1}\frac{\partial \bar{z}_1}{\partial \bar{s}_1}\frac{\partial \bar{s}_1}{\partial U}$$

Some more MATH



$$\frac{\partial y}{\partial W_3} = \frac{\partial y}{\partial \bar{s}_{t+3}}\frac{\partial \bar{s}_{t+3}}{\partial W_3}$$

$$\frac{\partial y}{\partial W_2} = \frac{\partial y}{\partial \bar{s}_{t+3}}\frac{\partial \bar{s}_{t+3}}{\partial \bar{s}_{t+2}}\frac{\partial \bar{s}_{t+2}}{\partial W_2}$$

$$\frac{\partial y}{\partial W_1} = \frac{\partial y}{\partial \bar{s}_{t+3}}\frac{\partial \bar{s}_{t+3}}{\partial \bar{s}_{t+2}}\frac{\partial \bar{s}_{t+2}}{\partial \bar{s}_{t+1}}\frac{\partial \bar{s}_{t+1}}{\partial W_1}$$

$$\frac{\partial y}{\partial W} = \frac{\partial y}{W_1} + \frac{\partial y}{W_2} + \frac{\partial y}{\partial W_3}$$

$$W_1 = W_2 = W_3 = W$$

From *equation 52*, *equation 51* and the *set of equations 50* we derive that:

$$\frac{\partial y}{\partial W} = \frac{\partial y}{\partial \bar{s}_{t+3}} \frac{\partial \bar{s}_{t+3}}{\partial W} + \frac{\partial y}{\partial \bar{s}_{t+3}} \frac{\partial \bar{s}_{t+3}}{\partial \bar{s}_{t+2}} \frac{\partial \bar{s}_{t+2}}{\partial W} + \frac{\partial y}{\partial \bar{s}_{t+3}} \frac{\partial \bar{s}_{t+3}}{\partial \bar{s}_{t+2}} \frac{\partial \bar{s}_{t+2}}{\partial \bar{s}_{t+1}} \frac{\partial \bar{s}_{t+1}}{\partial W}$$

Equation 53

Equation 53 summarizes the mathematical procedure of BPTT and can be simply written as:

$$\frac{\partial y}{\partial W} = \sum_{i=t+1}^{t+3} \frac{\partial y}{\partial \bar{s}_{t+3}} \frac{\partial \bar{s}_{t+3}}{\partial \bar{s}_i} \frac{\partial \bar{s}_i}{\partial W}$$

Equation 54

$$\frac{\partial \bar{y}}{\partial W} = \sum_{i=t+1}^{t+N} \frac{\partial \bar{y}}{\partial \bar{s}_{t+N}} \frac{\partial \bar{s}_{t+N}}{\partial \bar{s}_i} \frac{\partial \bar{s}_i}{\partial W}$$

Summary.

RECURRENT NEURAL NETWORKS SUMMARY

Training in mini batches using Gradient Descent

Updating the weights once every N steps

Reduce the complexity

Remove the noise

$$\delta_{ij} = \frac{1}{m} \sum_{k}^{m} \delta_{ij}{}_{k}$$

- No need to backprop. every sample
- If backprop. further, >6-8 layers, then vanishing gradient is an issue
- Exploding gradient, then

Gradient Clipping

Timestep t

$\delta = \frac{\partial y}{\partial W_{ij}}$ > Threshold ?

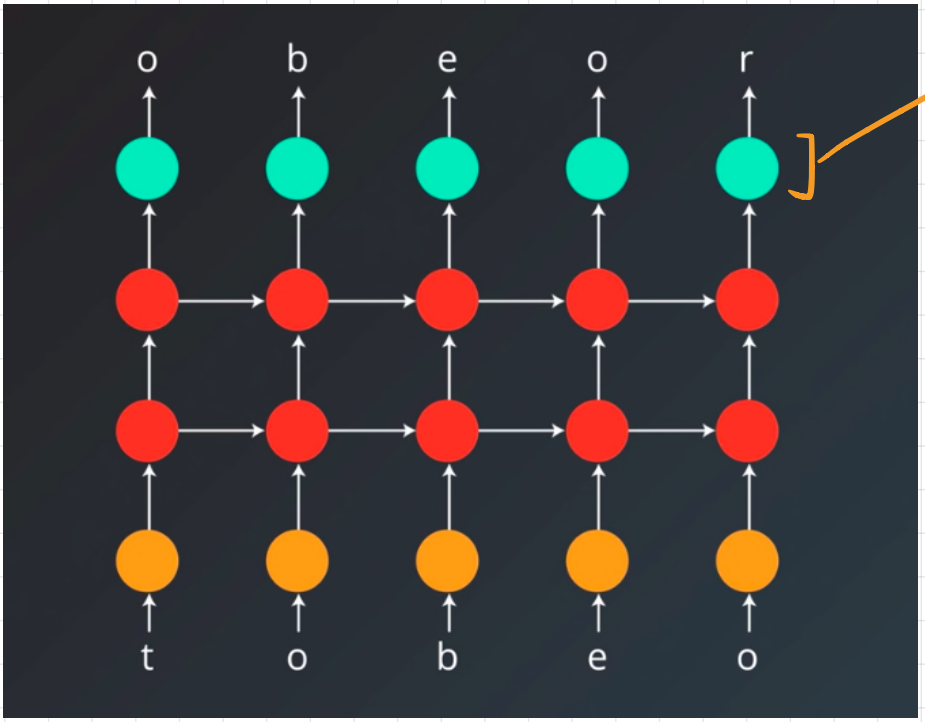
If yes, normalize the gradient

penalize large gradients

Implementing Character-wise RNN

Read the text 1 character at a time & generate new one 1 at a time

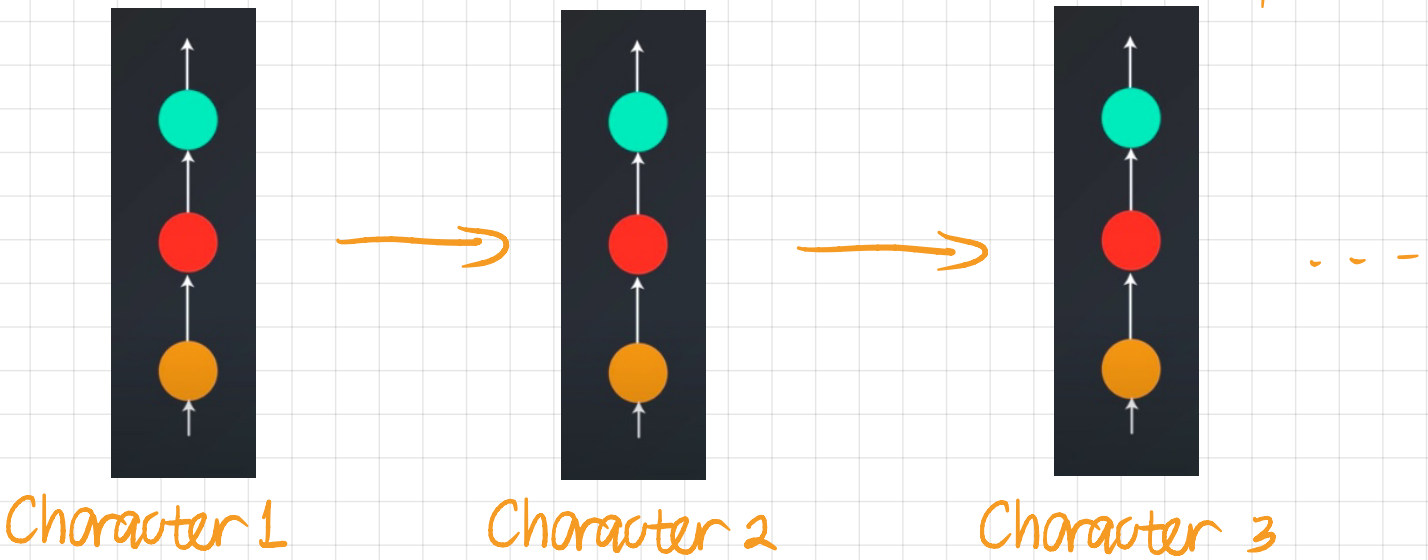
"To be or not to be" — Shakespere



Softmax → Prob. of each character

Output layer

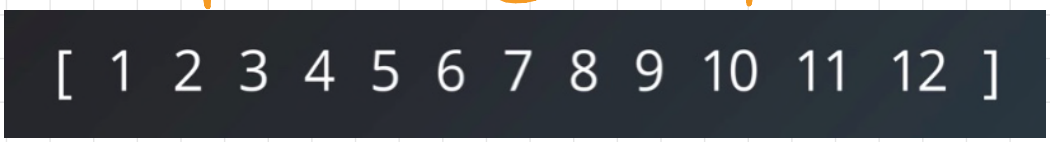
- Target ≡ input sequence, but shifted over 1.
- Each character is predicting the next character in the sequence.



Sequence Batching

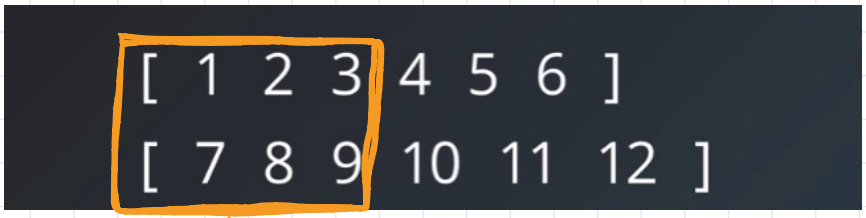
- Difficult : batching
- Take advantage of matrix multiplication.

Instead of processing 1 sequence



BS = 1

Split into 2 & process.

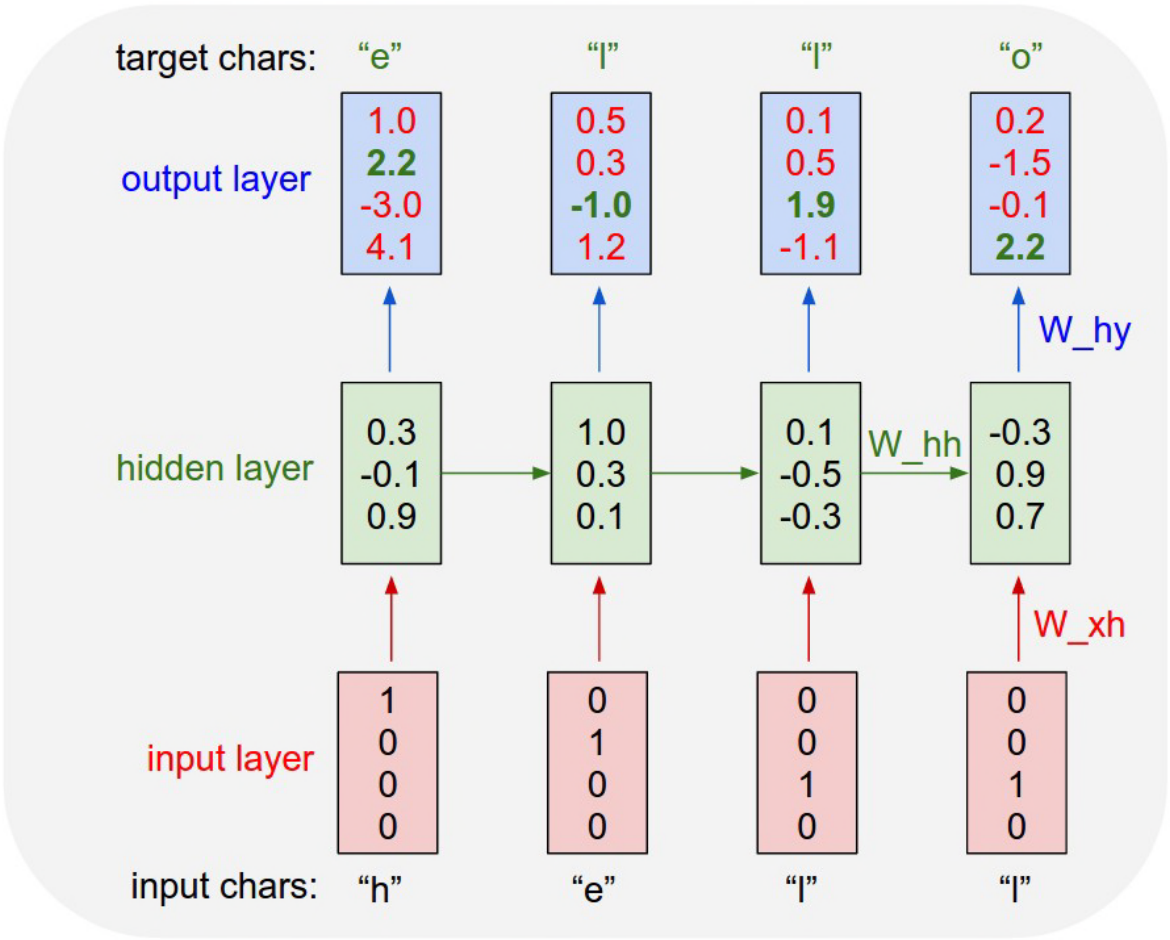


Batch_size = 2

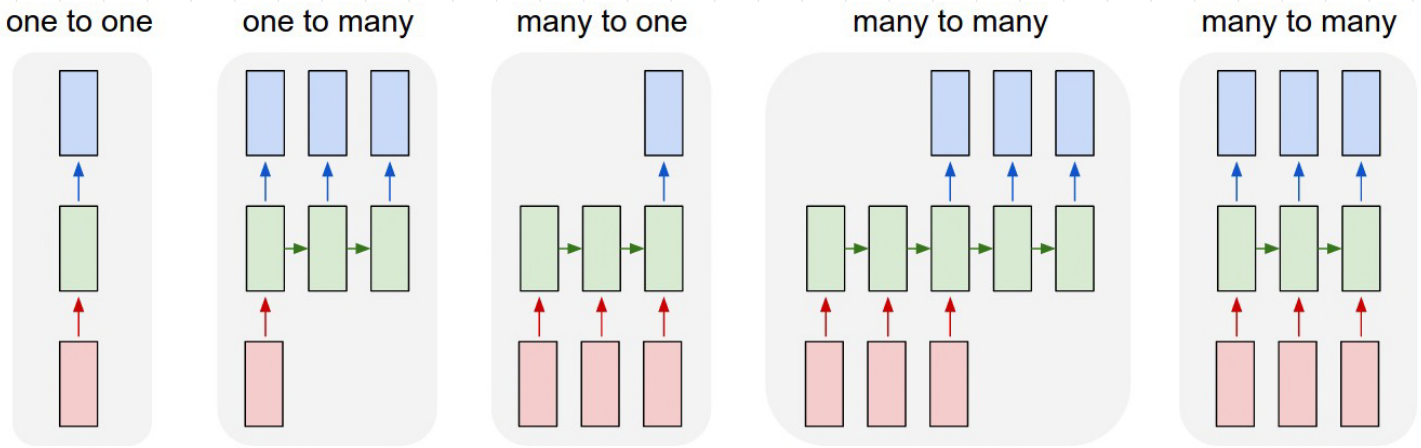
Sequence length of choice, here, it equals to 3

Then this is the 1st batch of data ≡ mini-sequence

- Retain the hidden state of 1 batch & use it at the start of the next batch
  - sequence info is transferred across batches


target chars:
“e”
“l”
“l”
“o”
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
“h”
“e”
“l”
“l”

Some own research:



(1) Image classification

(2) Sequence output: image → words

(3) Seq input: Setiment analysis

(4) Seq in → out: Machine Translation

(5) Synced seq in-out: video classification (each frame)

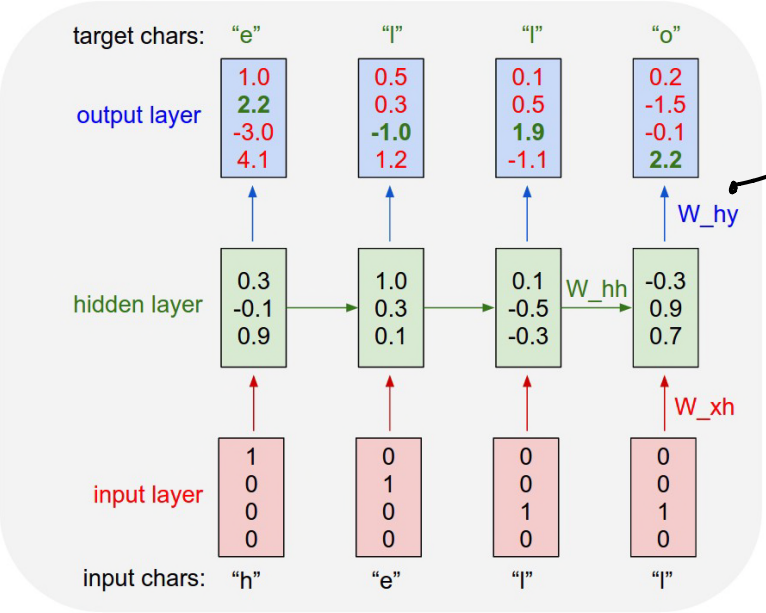
If training vanilla neural nets is optimization over functions, training recurrent nets is optimization over programs.

Vanilla RNN

Update hidden state:

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} X_t)$$

Char RNN - Prob of next letter given in the context of prev. one.



→ CrossEntropyLoss

] 3 hidden units

- Perform backprop. to nudge wts.

Temperature:

Temp of Softmax ↓, RNN becomes more confident.

Higher temp → more mistakes.

The Evolution of Samples when Training:

Iter 100

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

Iter 300

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

Iter 500

```
we counter. He stutn co des. His stanted out one ofler that concossions and was
to gearang reay Jotrets and with fre colt otf paitt thin wall. Which das stimn
```

Iter 700

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

Iter 1300

```
"Kite vouch!" he repeated by her
door. "But I would be done and quarts, feeling, then, son is people...."
```

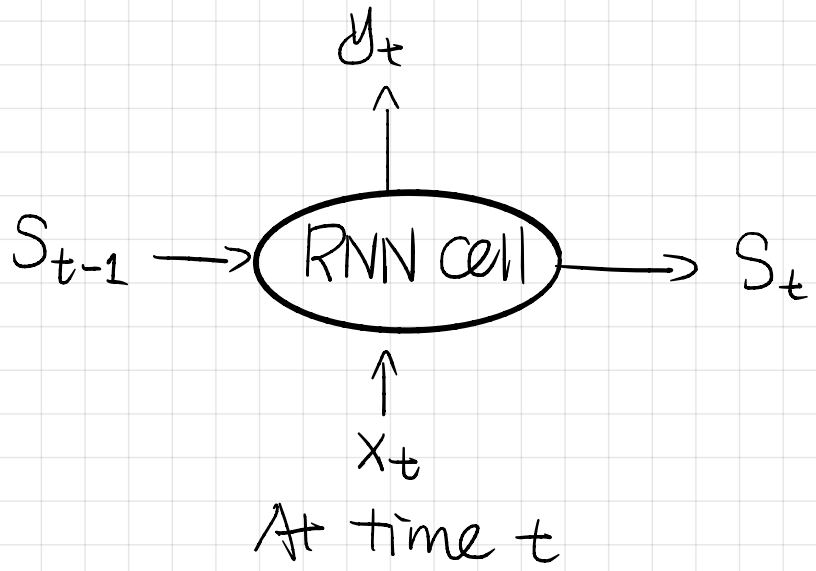
Iter 2000!

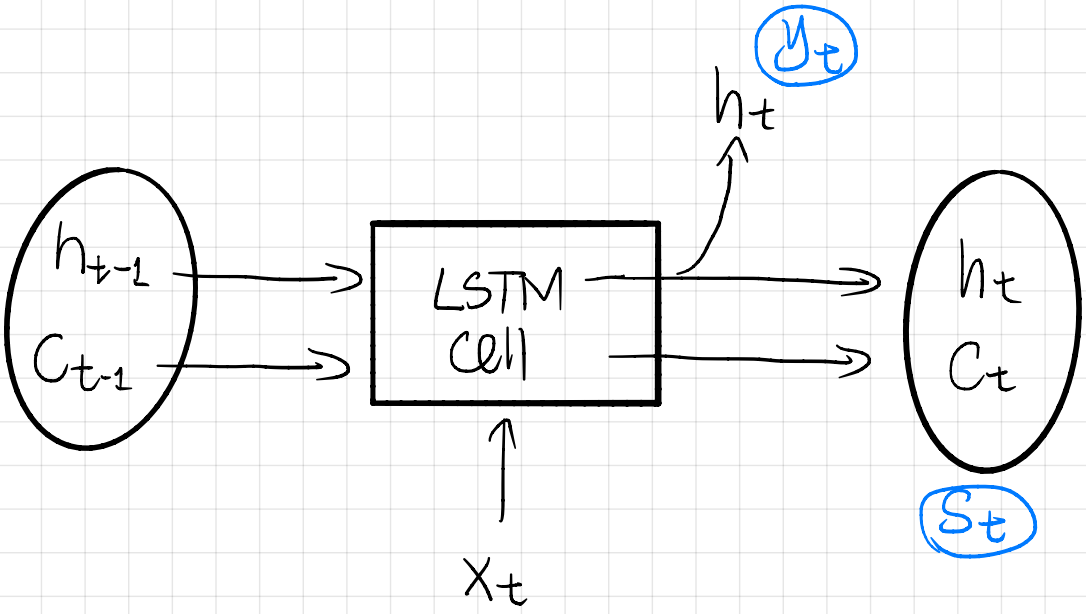
```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Checkout Blog:

Andrej Karpathy Blog

Further readings to clear out confusions





- LSTM 用 cell值(C_t) & output gate 计算的 h_t
- 但输出层只用 hidden state 的信息

LSTM: $S_t = [(C_t^{(1)}, h_t^{(1)}), (C_t^{(2)}, h_t^{(2)}) \cdots]$, $y_t = h_t^{\ell}$

知乎: 学会区分 RNN的output和state.