Recurrent Neural Network

AICTE-MARGDARSHAN sponsored workshop on Computational Intelligence for Multimedia

T. T. Mirnalinee
Prof/CSE
SSN College of Engineering
mirnalineett@ssn.edu.in

Introduction

- Conventional feedforward neural networks can be used to approximate *any* spatially finite function
- Functions which have a *fixed* input space there is always a way of encoding these functions as neural networks.

$$y_j(t) = f(net_j(t))$$

$$net_j(t) = \sum_i^n x_i(t) v_{ji} + \theta_j$$

- Recurrent neural networks are fundamentally different from feedforward architectures in the sense that they not only operate on an input space but also on an internal *state* space

Introduction

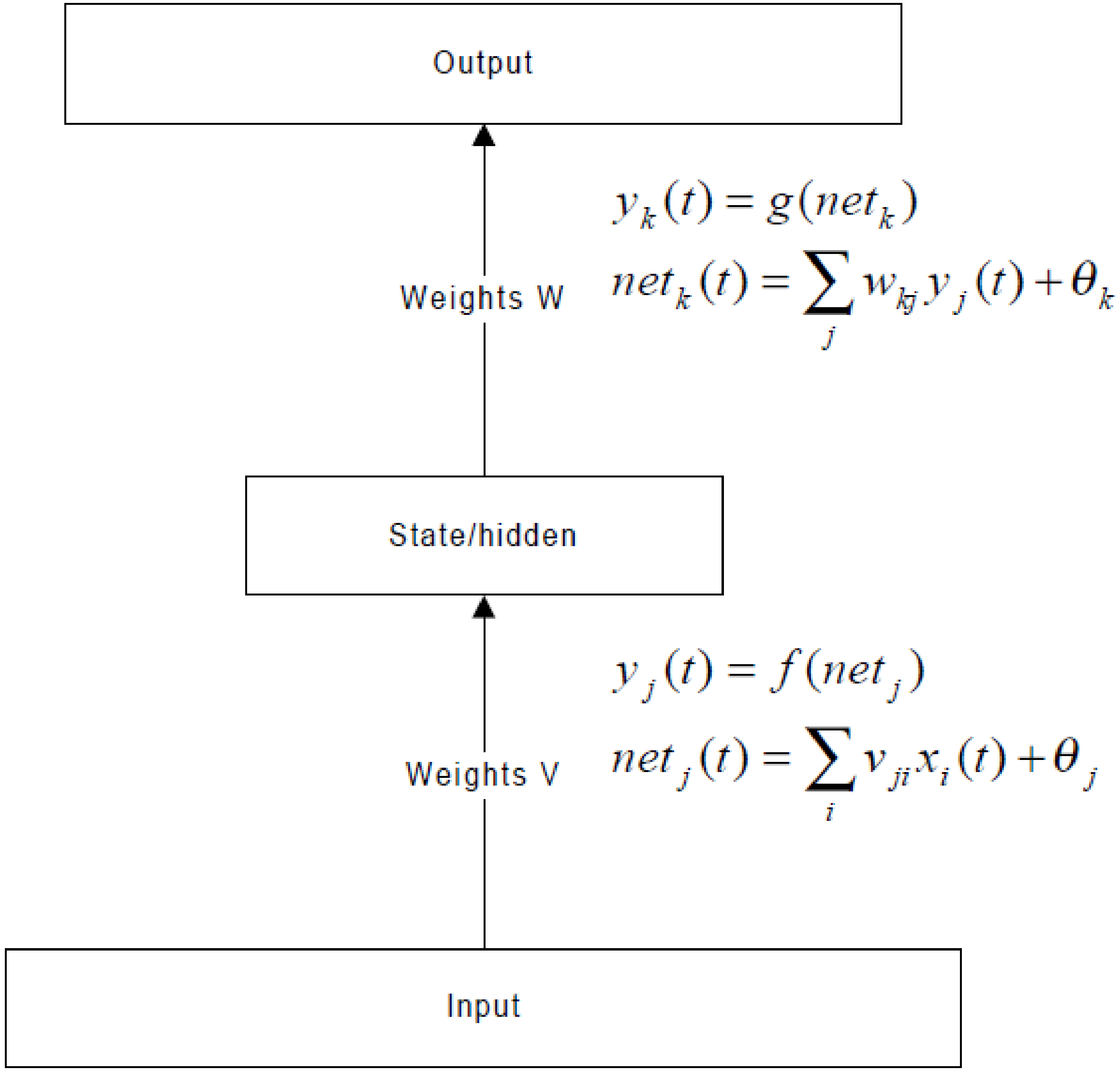
- The state space enables the representation of temporally/sequentially extended dependencies
- simple recurrent network, the input vector is similarly propagated through a weight layer, but also combined with the previous state activation through an additional *recurrent* weight layer, U

$$net_j(t) = \sum_i^n x_i(t)v_{ji} + \sum_h^m y_h(t-1)u_{jh}) + \theta_j$$

$$y_k(t) = g(net_k(t))$$

$$net_k(t) = \sum_j^m y_j(t)w_{kj} + \theta_k$$

Simple Feed forward Network



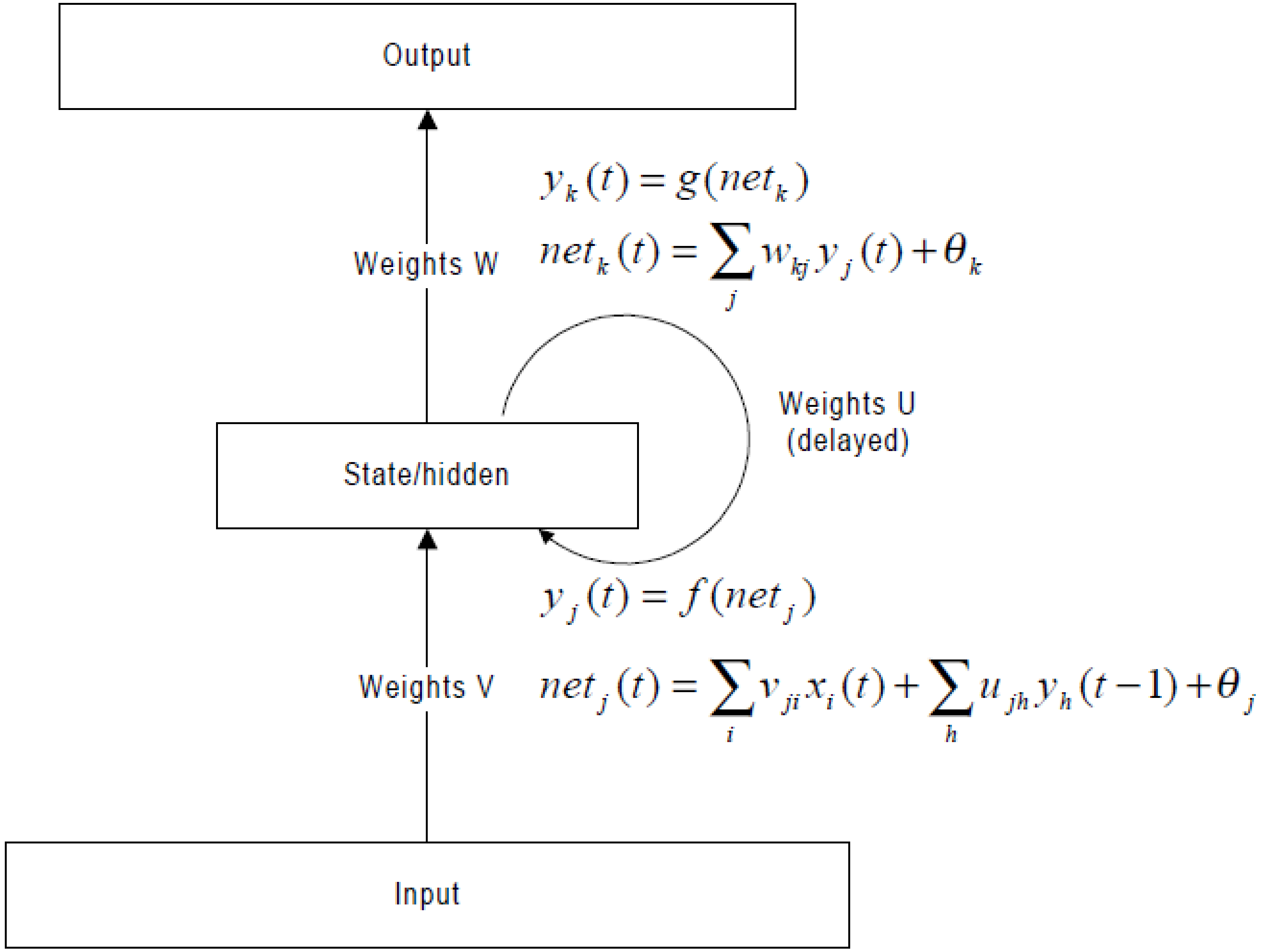
$$y_k(t) = g(net_k)$$

$$net_k(t) = \sum_j w_{kj} y_j(t) + \theta_k$$

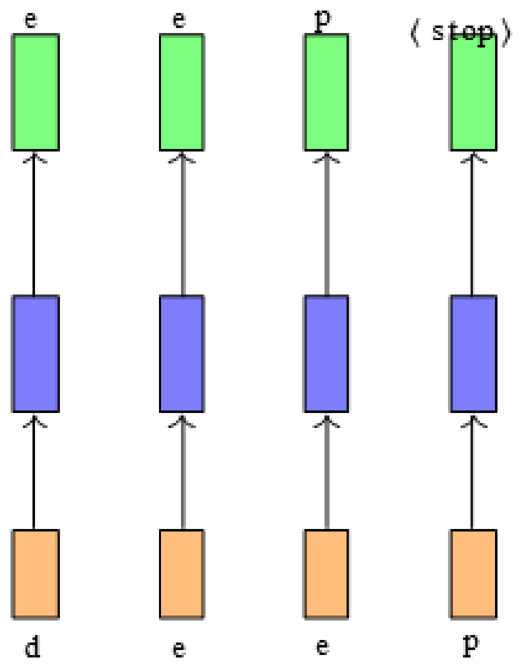
$$y_j(t) = f(net_j)$$

$$net_j(t) = \sum_i v_{ji} x_i(t) + \theta_j$$

Simple Recurrent Network



Sequence to Sequence Learning



In many applications the input is not of a fixed size

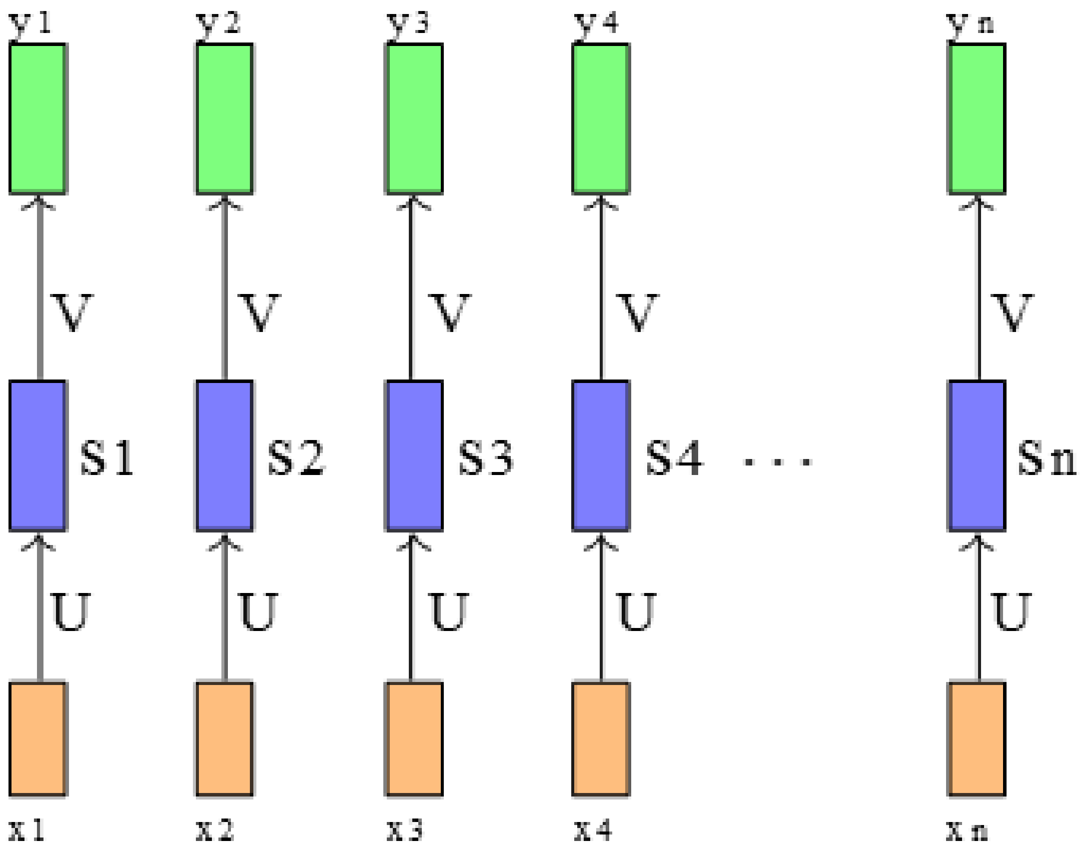
Further successive inputs may not be independent of each other

For example, consider the task of auto completion

Given the first character 'd' you want to predict the next character 'e' and so on

Sequence to Sequence Learning

- Successive inputs are no longer independent
- The length of the inputs and the number of predictions you need to make is not fixed (for example, “learn”, “deep”, “machine” have different number of characters)
- Each network (input : character output : character)

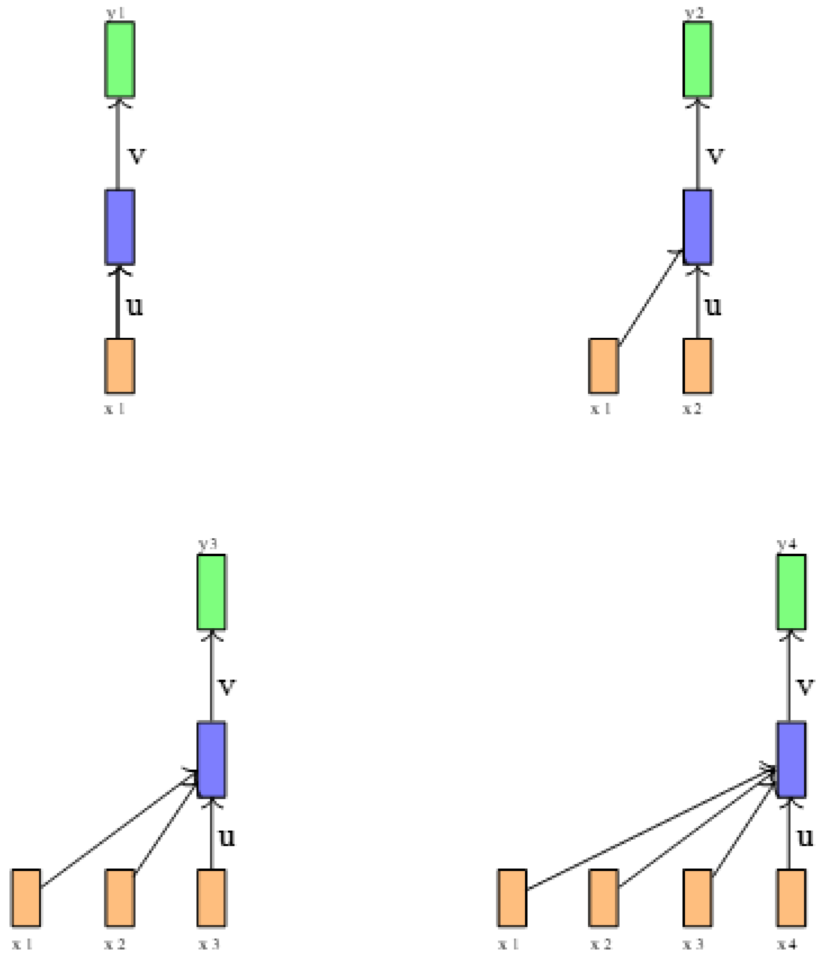


y1
y2
y3
y4
yn
V
V
V
V
V
S1
S2
S3
S4
· · ·
Sn
U
U
U
U
U
x1
x2
x3
x4
xn

Sequence Problems

- Tagging a sentence
- Sentiment Analysis
- Image captioning
- Video Captioning
- Speech processing

- Dependence between inputs
- Variable number of inputs
- Function executed at each time step is the same



First, the function being computed at each time-step now is different

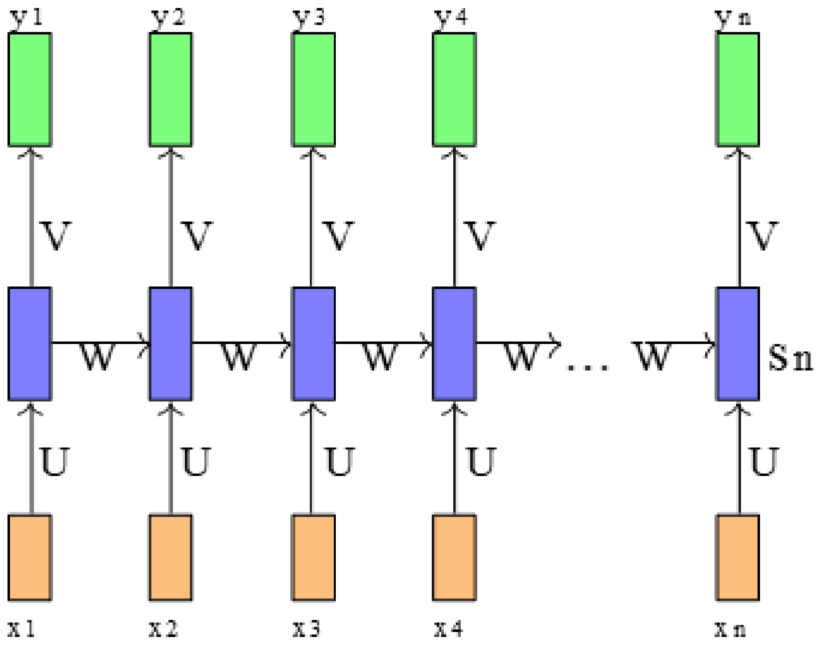
y1 = f1(x1)

y2 = f2(x1,x2)

y3 = f3(x1,x2,x3)

The network is now sensitive to the length of the sequence

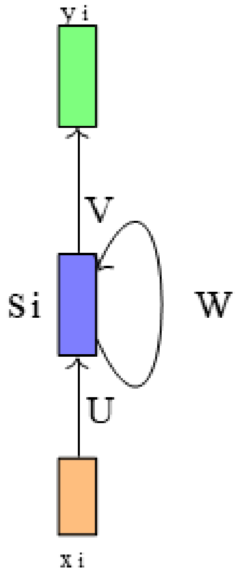
For example a sequence of length 10 will require f1,...,f10 whereas a sequence of length 100 will require f1,…,f100

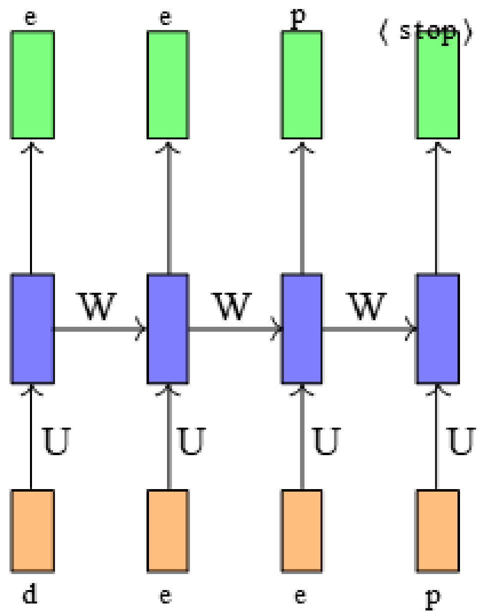


- The solution is to add a recurrent connection in the network,

$$s_i = \sigma(Ux_i + Ws_{i-1} + b)$$
$$y_i = O(V\ s_i + c)$$
or
$$y_i = f\,(x_i, s_{i-1}, W, U, V, b, c)$$

- s_i is the state of the network at timestep i
- The parameters are W,U,V,c,b which are shared across timesteps
- The same network (and parameters) can be used to compute $y_1, y_2, \ldots, y_{10}$ or y_{100}

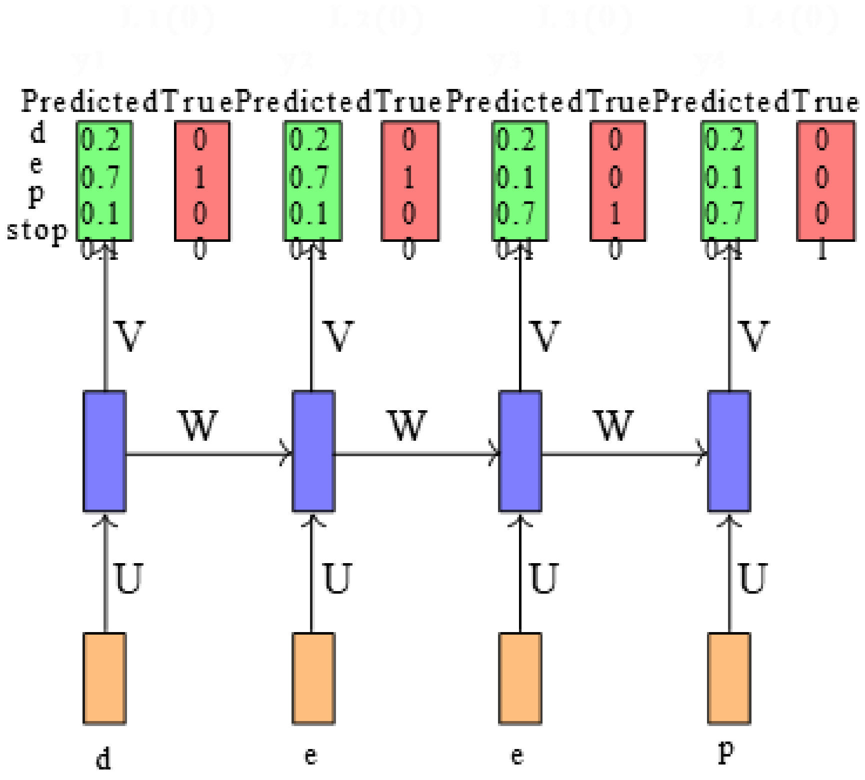




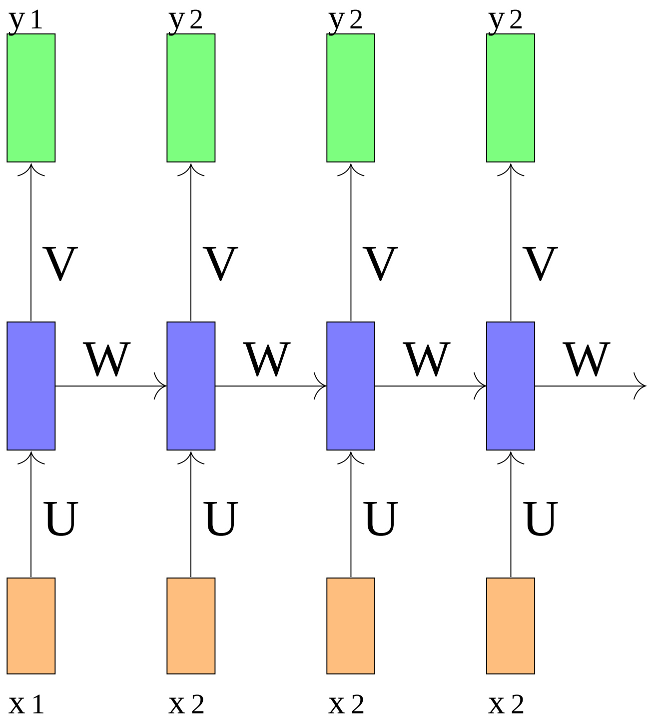
Suppose we consider the task of autocompletion (predicting the next character)

For simplicity we assume that there are only 4 characters in our vocabulary (d,e,p, <stop>)

At each timestep we want to predict one of these 4 characters



- Suppose we initialize U,V,W randomly and the network predicts the probabilities as shown
- And the true probabilities are as shown
- We need to answer two questions
- What is the total loss made by the model ?
- How do we backpropagate this loss and update the parameters (θ = {U,V,W,b,c}) of the network ?



- Before proceeding let us look at the dimensions of the parameters carefully

$x_i \in R^{d}$ (n-dimensional input)

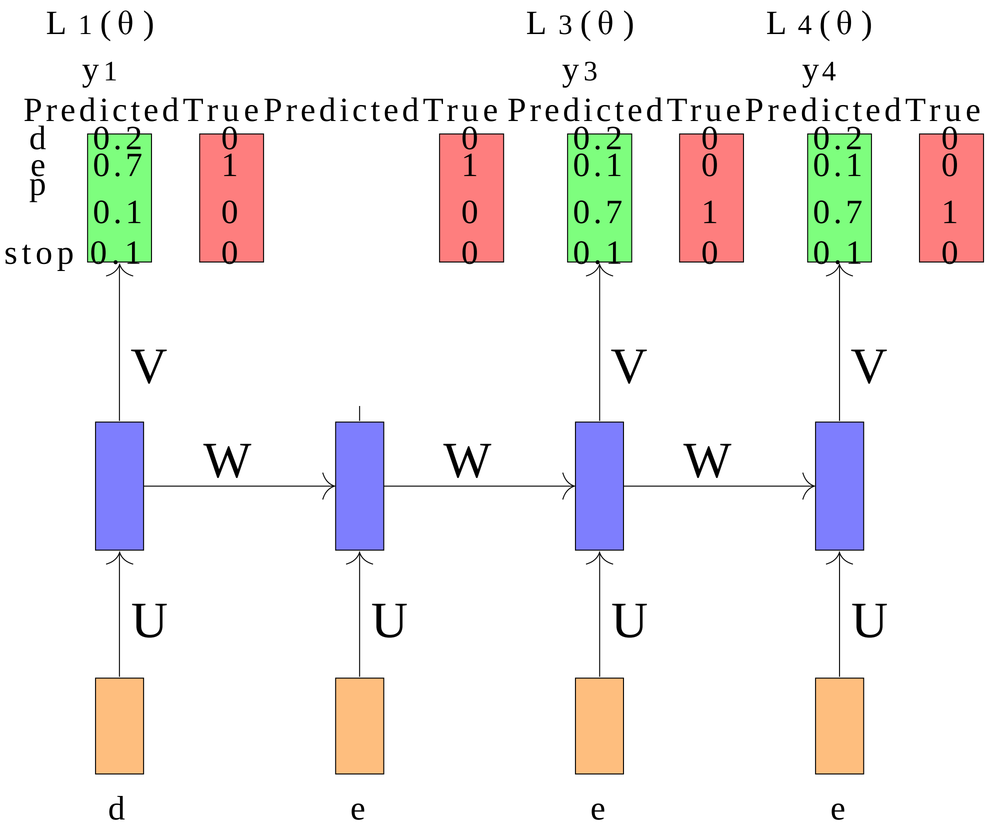
$s_i \in R^{k}$ (d-dimensional state)

$y_i \in R$ (say k classes)

$U \in R^{n \times d}$

$V \in R^{d \times k}$

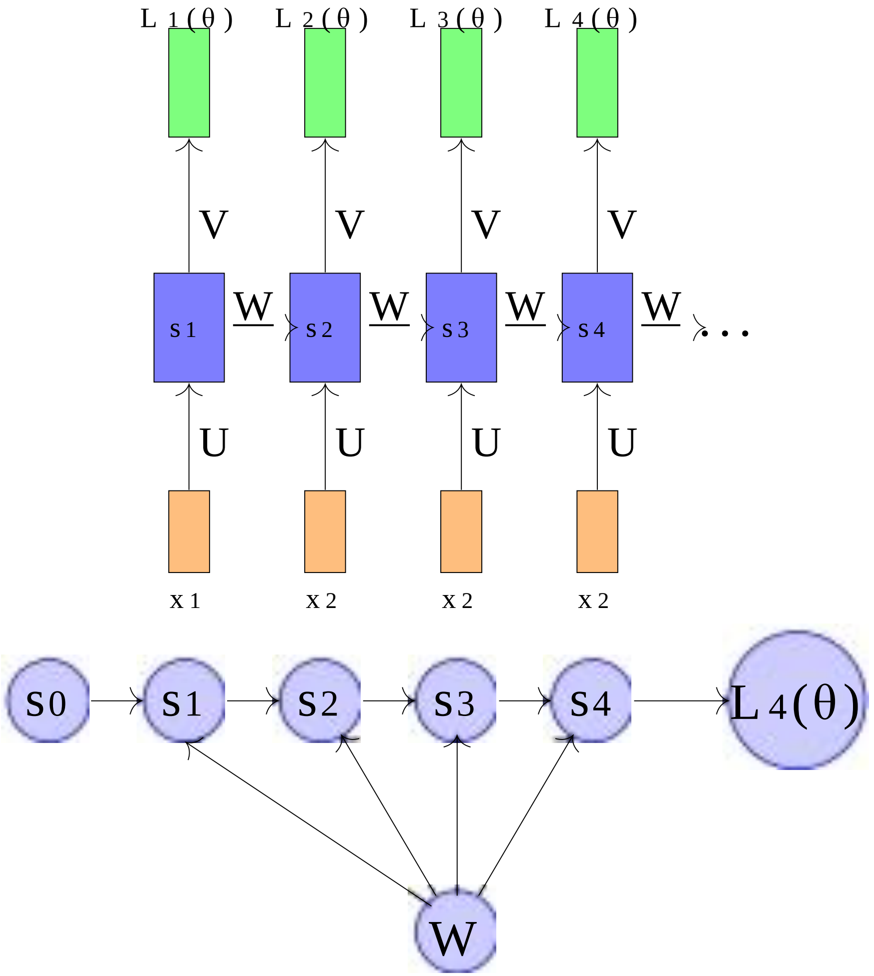
$W \in R^{d \times d}$



- Let us consider the derivative $\frac{\partial \mathscr{L}(\theta)}{\partial W}$

$$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}_t(\theta)}{\partial W}$$

- By the chain rule of derivatives we know that $\frac{\partial \mathscr{L}_t(\theta)}{\partial W}$ is obtained by summing gradients along all the paths from $\mathscr{L}_t(\theta)$ to W



- Recall that

$$s_4 = \sigma(W s_3 + b)$$

- In such an ordered network, we can't compute $\frac{\partial s_4}{\partial W}$ by simply treating s_3 as a constant (because it also depends on W)
- In such networks the total derivative $\frac{\partial s_4}{\partial W}$ has two parts
- Explicit : $\frac{\partial^+ s_4}{\partial W}$, treating all other inputs as constant
- Implicit : Summing over all indirect paths from s_4 to W
-

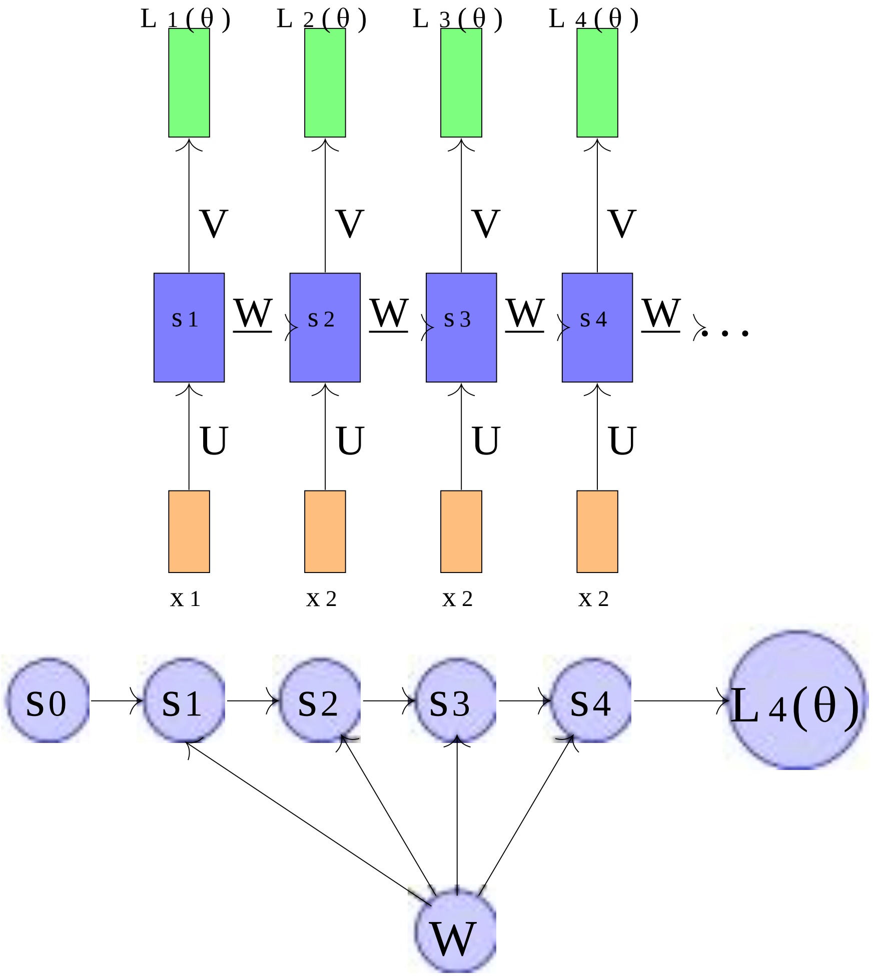
$$\frac{\partial s_4}{\partial W} = \underbrace{\frac{\partial s_4}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial W}}_{\text{implicit}}$$

$$= \frac{\partial s_4}{\partial W} + \frac{\partial s_4}{\partial s_3}\left[\underbrace{\frac{\partial s_3}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}}_{\text{implicit}}\right]$$

$$= \frac{\partial s_4}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\left[\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W}\right]$$

$$= \frac{\partial s_4}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\left[\frac{\partial s_1}{\partial W}\right]$$

$$\frac{\partial s_4}{\partial W} = \frac{\partial s_4}{\partial s_4}\frac{\partial s_4}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial W} + \frac{\partial s_4}{\partial s_2}\frac{\partial s_2}{\partial W} + \frac{\partial s_4}{\partial s_1}\frac{\partial s_1}{\partial W} = \sum_{k=1}^{4}\frac{\partial s_4}{\partial s_k}\frac{\partial s_k}{\partial W}$$



$$\frac{\partial L_4(\theta)}{\partial W} = \frac{\partial L_4(\theta)}{\partial s_4}\frac{\partial s_4}{\partial W}$$

$$\frac{\partial s_4}{\partial W} = \sum_{k=1}^{4} \frac{\partial s_4}{\partial s_k}\frac{\partial^+ s_k}{\partial W}$$

$$\therefore \frac{\partial L_t(\theta)}{\partial W} = \frac{\partial L_t(\theta)}{\partial s_t}\sum_{k=1}^{t} \frac{\partial s_t}{\partial s_k}\frac{\partial^+ s_k}{\partial W}$$

- This algorithm is called backpropagation through time (BPTT) as we backpropagate over all previous time steps

- The total loss is simply the sum of the loss over all time-steps

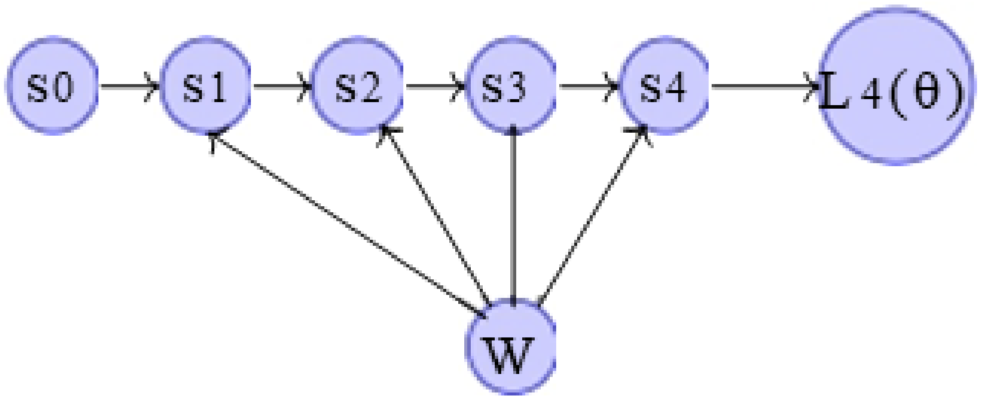
$$L(\theta) = \sum_{t=1}^{T} L_t(\theta)$$

$$L_t(\theta) = -\log(y_{tc})$$

y_{tc} = predicted probability of true character at time-step t

- For backpropagation we need to compute the gradients w.r.t. W,U,V,b,c

- $L_4(\theta)$ depends on s_4
- s_4 in turn depends on s_3 and w s_3 in turn depends on s_2 and w s_2 in turn depends on s_1 and w s_1 in turn depends on s_0 and w



Learning by Back propagation

$$C = \frac{1}{2}\sum_{p}^{n}\sum_{k}^{m}(d_{pk} - y_{pk})^2$$

$$\Delta w = -\eta \frac{\partial C}{\partial w}$$

$$\Delta v_{ji} = \eta \sum_{p}^{n} \delta_{pj} x_{pi}$$

$$\Delta w_{kj} = \eta \sum_{p}^{n} \delta_{pk} y_{pj}$$

$$\Delta u_{jh} = \eta \sum_{p}^{n} \delta_{pj}(t) y_{ph}(t-1).$$

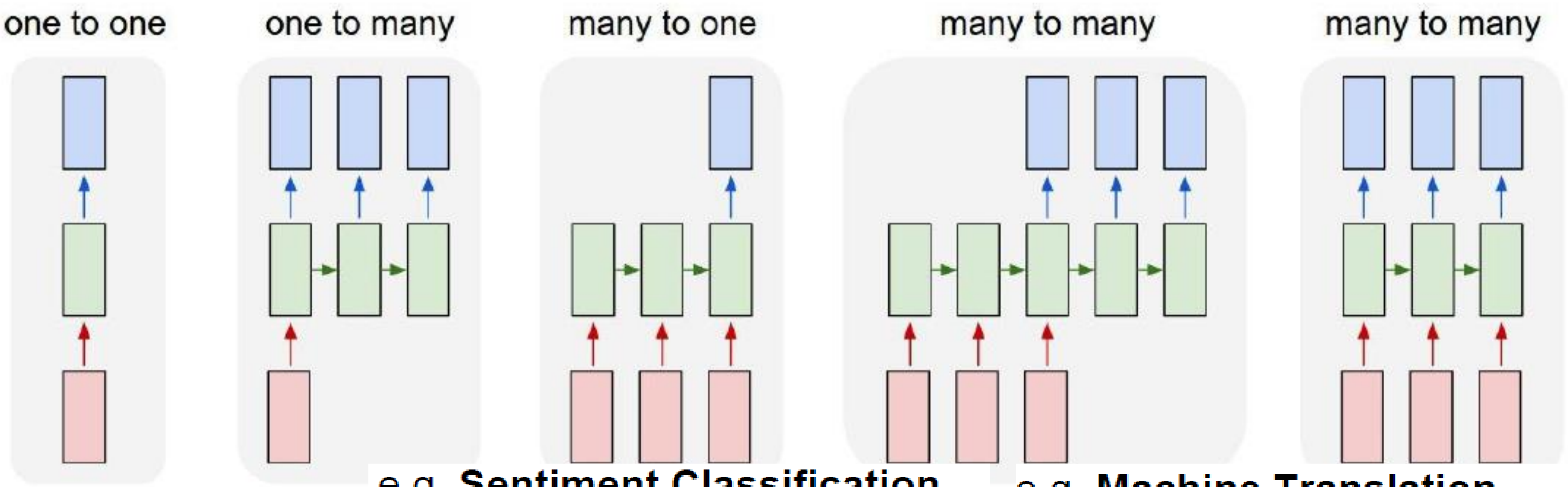
$$g(net) = \frac{1}{1 + e^{-net}}.$$

Simple recurrent network

- A simple recurrent network has activation feedback which embodies

short-term memory. A state layer is updated not only with the external input of the network but also with activation from the previous forward propagation. The feedback is modified by a set of weights as to enable automatic adaptation through learning

Various forms of RNN



Vanilla Neural Networks

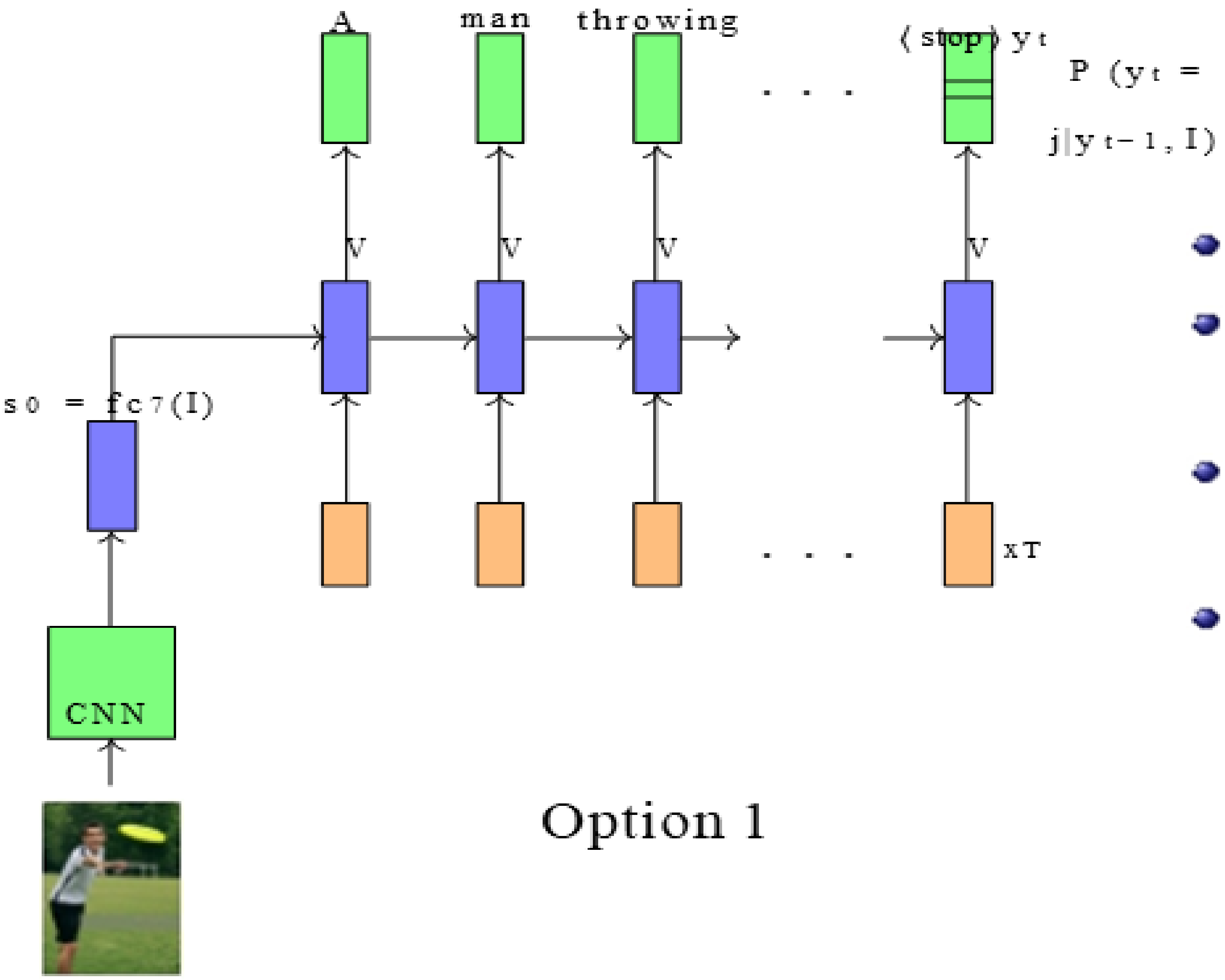
e.g. **Image Captioning**
image -> sequence of words

e.g. **Sentiment Classification**
sequence of words -> sentiment

e.g. **Machine Translation**
seq of words -> seq of words

e.g. **Video classification on frame level**

Image Captioning



Option 1

- Option 1: Set $s_0 = fc_7(I)$
- depend on $fc_7(I)$
- pends on $fc_7(I)$
- $P(y_t = j|s_t, fc_7(I))$

Vanishing Gradient

- RNN suffers from vanishing gradient problem
  - What happens to the magnitude of the gradients as we backpropagate through many layers?
  - If the weights are small, the gradients shrink exponentially.
  - If the weights are big the gradients grow exponentially.

- In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
- We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.
- So RNNs have difficulty dealing with long-range dependencies

- Solution LSTM
    - Selectively read
    - Selectively write
    - Selectively forget

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU
- improve gradient flow
- Backward flow of gradients in RNN can explode or vanish.
- Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research