Unit 2

MACHINE INTELLIGENCE

Training an RNN using Backpropagation

Course Instructor:
Dr. Pooja Agarwal
Dept. of Computer Science and Engineering

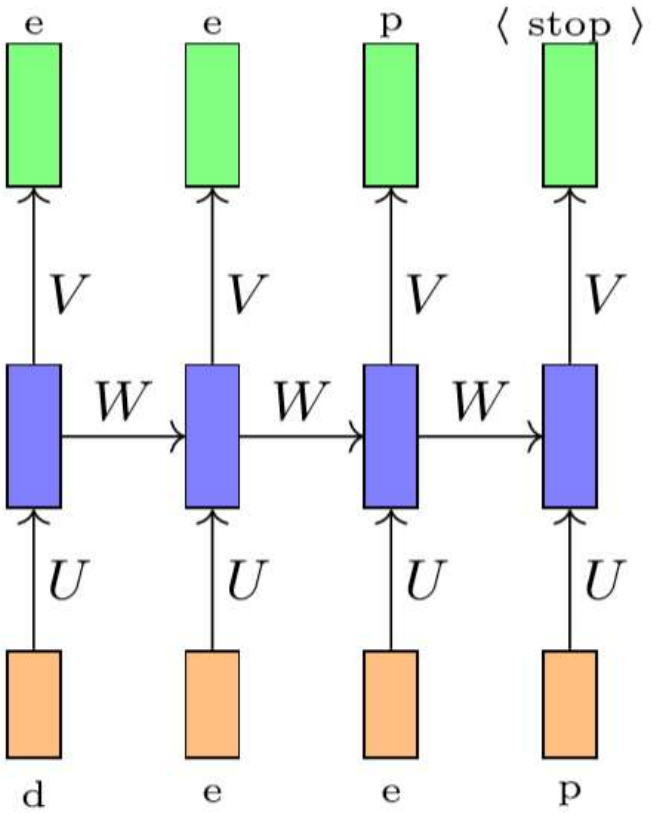
MACHINE INTELLIGENCE

Acknowledgement

The slides are prepared using the resources from various Universities from abroad and in India. Also, some material is taken from reliable resources from the internet throughout this course. The inputs to the slides are broadly adapted from slides by Prof. Mitesh M Khapra from IITM and by various professors within the university.

Machine Intelligence

Error or LOSS

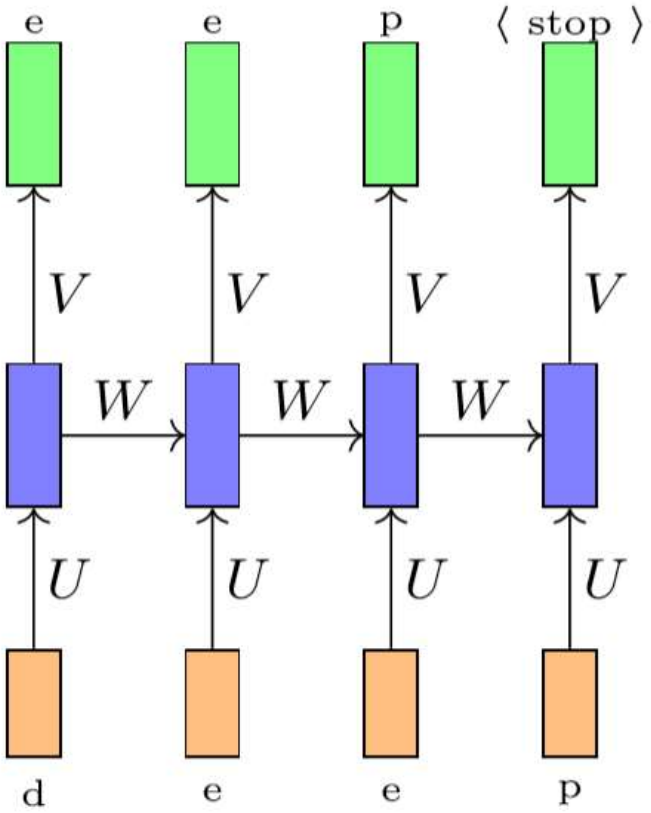


- Once again consider the task of auto completion (predicting the next character)
- Lets say, there are only 4 characters in the vocabulary (d, e, p, <stop>)
- At each time step, we predict one of these 4 characters
- **What is a suitable output function for this task ?**
  - Softmax!!!!
- **What is a suitable loss function for this task ?**
  - Cross Entropy!!!

PES UNIVERSITY
CELEBRATING 50 YEARS

Machine Intelligence

Error or LOSS (Additional FYI)



- Cross-entropy can be thought to calculate the total entropy between the distributions.
- **Cross-entropy** is a measure coming from information theory and tells **that how to calculate the total entropy between different distribution**.
- The cross-entropy between two probability distributions, such as Q from P, can be stated formally as:

  H(P, Q)

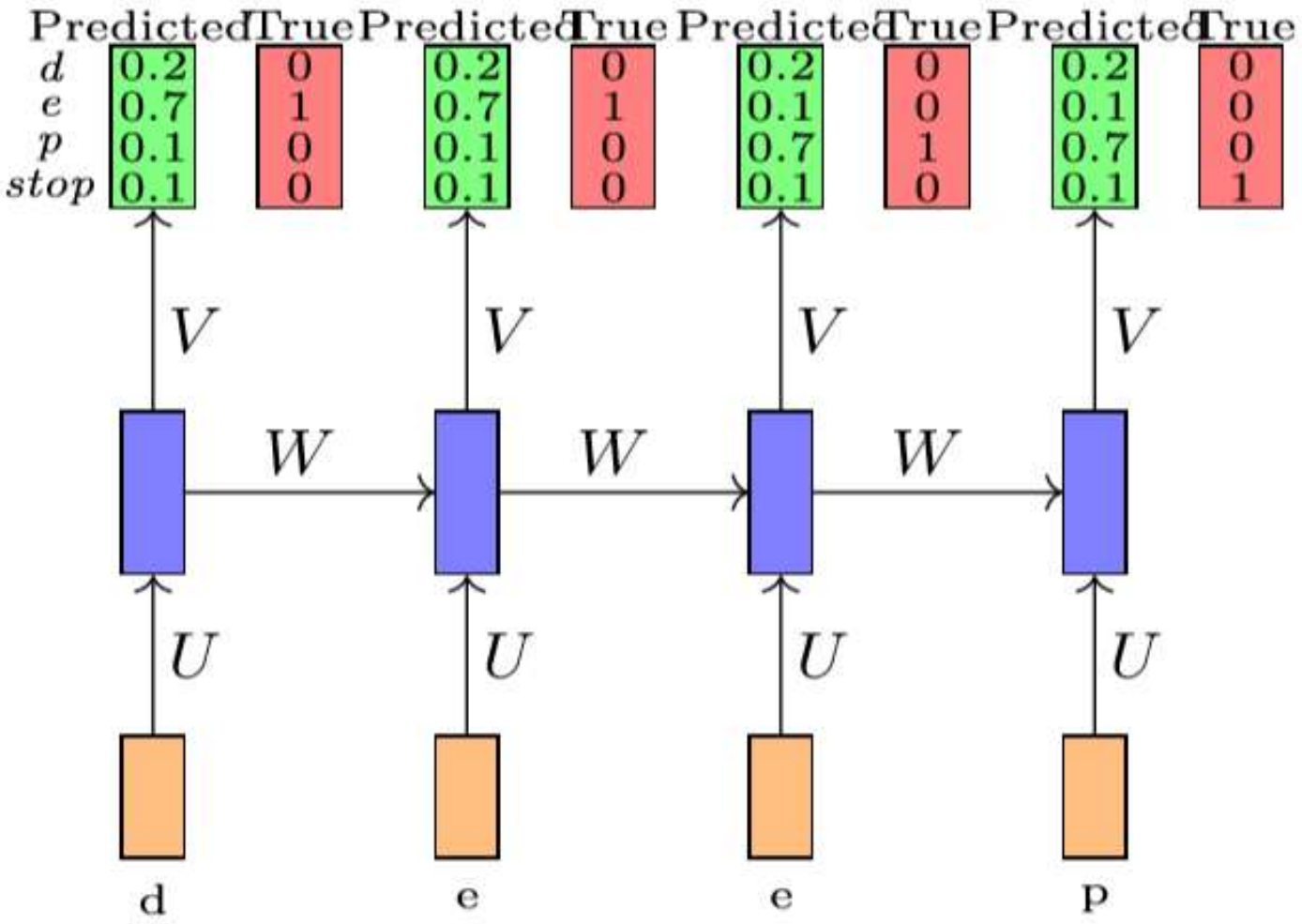
where H() is the cross-entropy function, P may be the target distribution and Q is the approximation of the target distribution.

For more info refer to : https://machinelearningmastery.com/cross-entropy-for-machine-learning/

Machine Intelligence

Error or LOSS

Now let's see how to come up with loss function or error function



- Suppose we initialize U, V, W **randomly** and the network predicts the probabilities as shown in this figure.
- And the true probabilities are as shown.
- So here we have 2 questions:

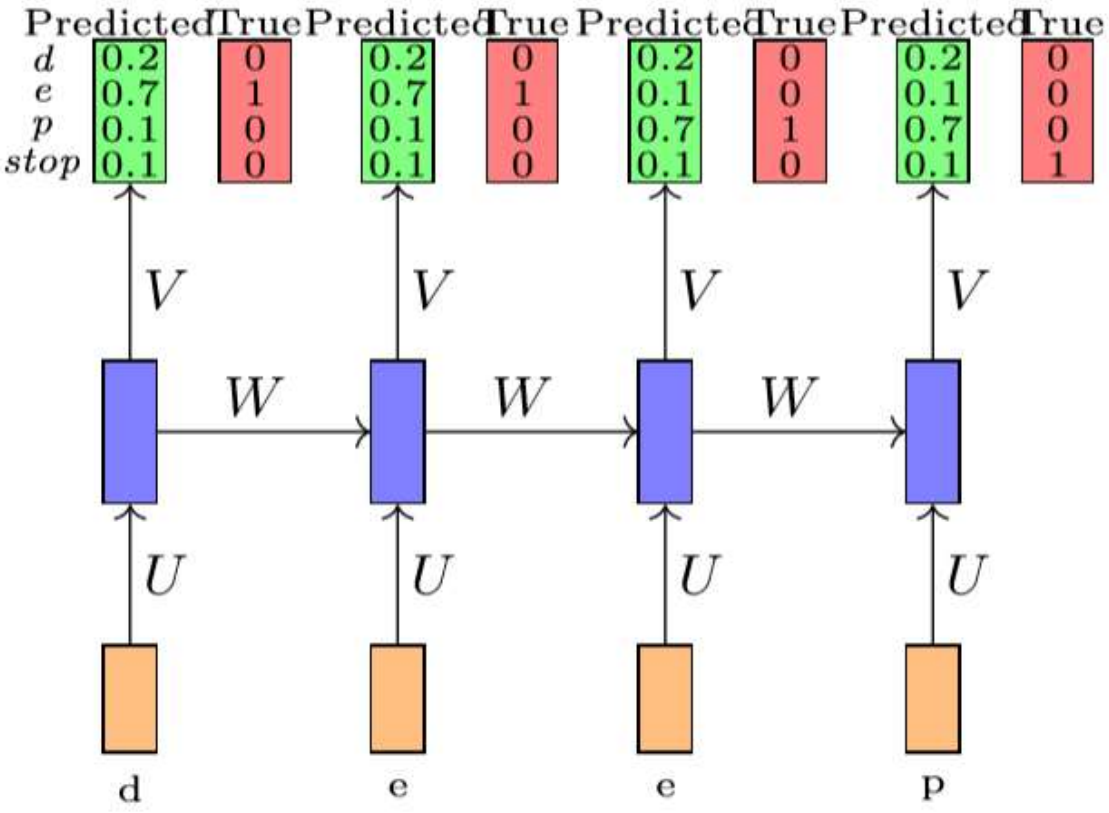
➢ **What is the total loss(error) made by the model** ?

➢ **How do we back propagate this error and update (fine tune) the parameters to minimize the error/loss**

(θ = {U,V,W,b,c}) of the network ?

Machine Intelligence

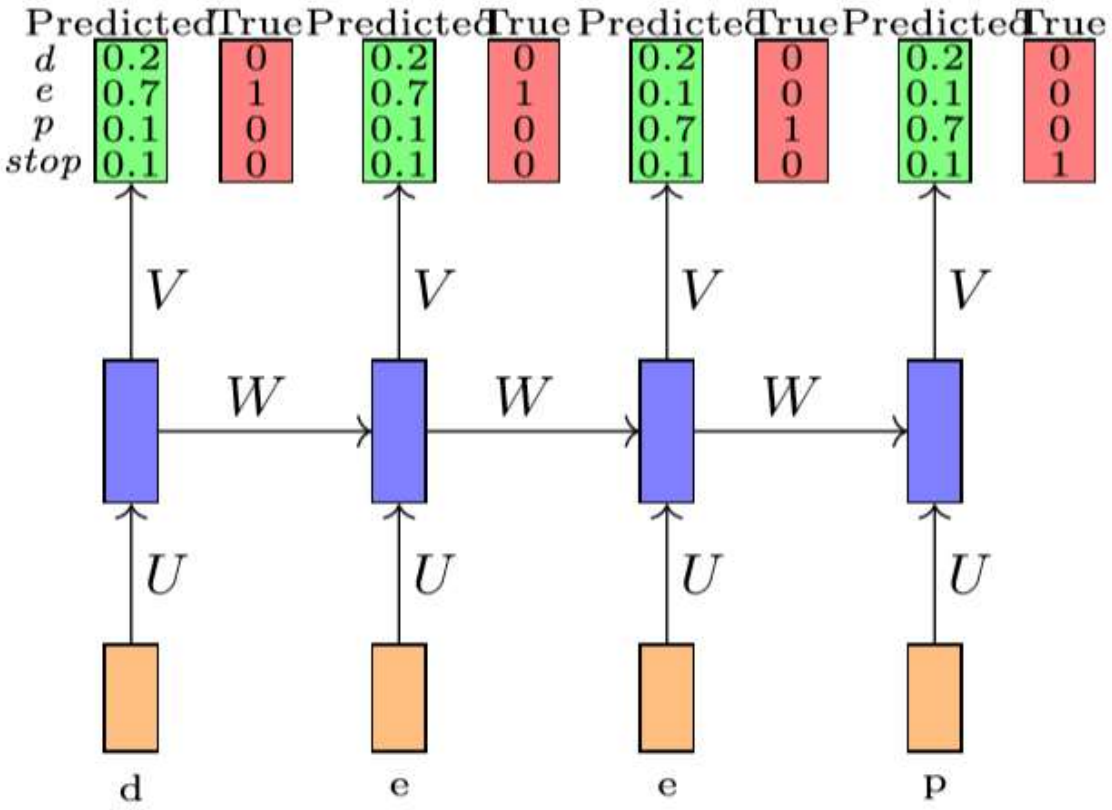
Error or LOSS



- Considering the varying length for each sequential data, we also assume the parameters in each time step are the same across the whole sequenece.
- Otherwise, it will be hard to compute the gradients. In addition, sharing the weights for any sequential length can generalize the model well.
- As for sequential labeling, we can use the maximum likelihood to estimate model parameters.
- In other words, we can **minimize the negative log likelihood the objective function**

Machine Intelligence

Error or LOSS



- Total loss(error), $\mathcal{L}(\theta)$ is simply the **sum of the loss over all time-steps, $\mathcal{L}_t(\theta)$**

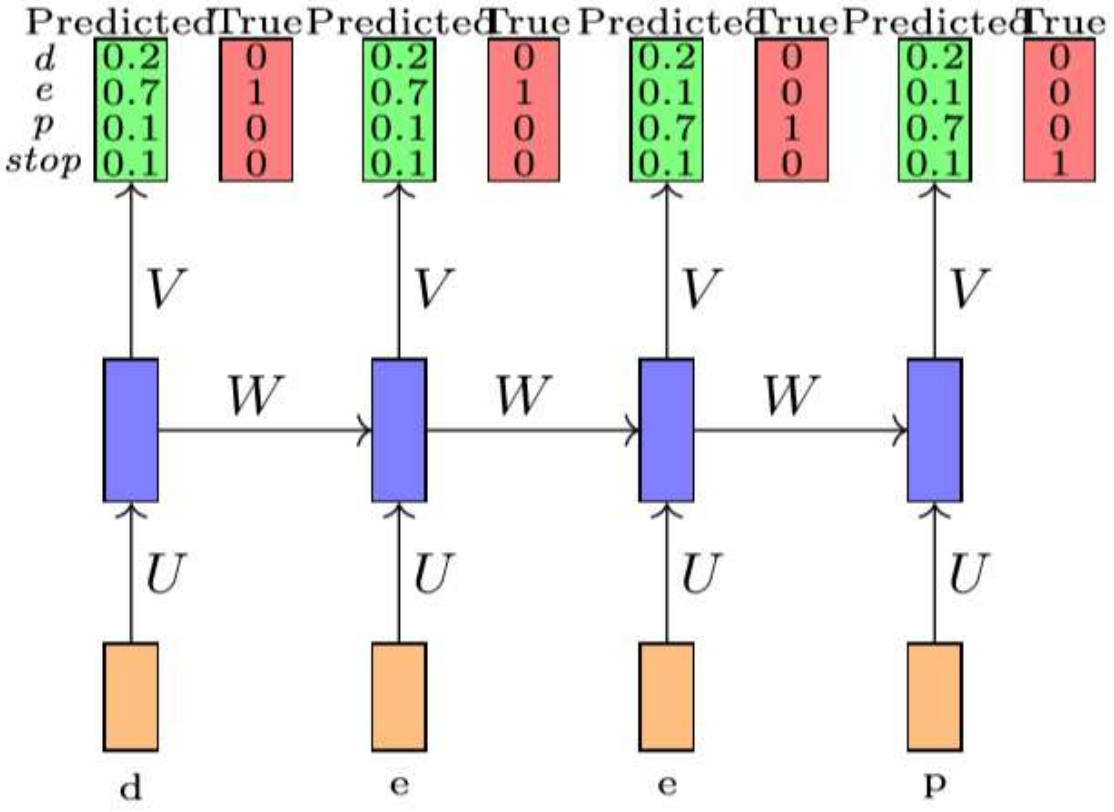
$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \mathcal{L}_t(\theta)$$

$$\mathcal{L}_t(\theta) = -y_i \log(\widehat{y_{tc}})$$

Where $\widehat{y_{tc}}$= predicted output of character at time step t and yi is the actual output and $\mathcal{L}_t(\theta)$ is loss or error at time t

$$T = \text{number of timesteps}$$

LOSS ! LOSS! LOSS!



- Error can be minimized by finding optimal weights for the RNN U,W,V.
- So, We'll use gradient descent method.
- Calculate the gradient of the loss function(error) wrt all the weights, then update the weights according to this

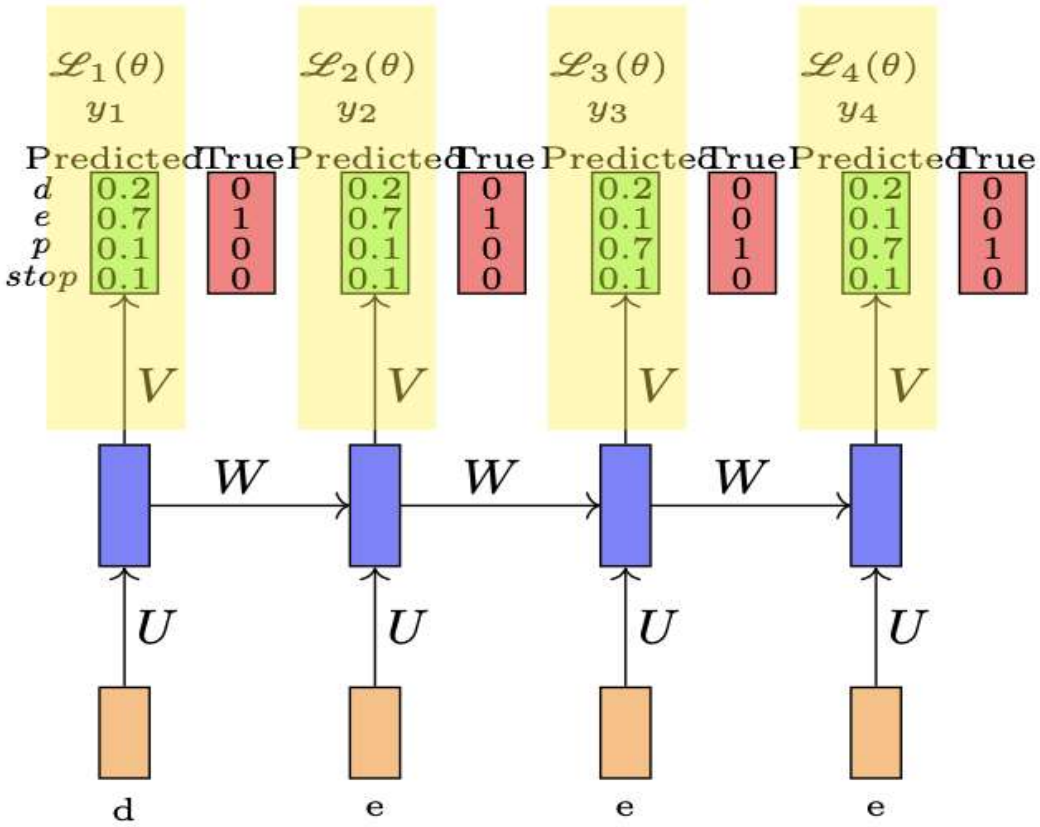
$$V = V - \alpha \frac{\partial L}{\partial V}$$

$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$U = U - \alpha \frac{\partial L}{\partial U}$$

Machine Intelligence

Gradient of loss (error) wrt to V



- From the loss function $\mathscr{L}$
- **So grad of $\mathscr{L}$ wrt V** will be summation of grad of $\mathscr{L}_i$ wrt V

$$\frac{\partial L}{\partial V} = \frac{\partial L_0}{\partial V} + \frac{\partial L_1}{\partial V} + \frac{\partial L_2}{\partial V} + \ldots \frac{\partial L_{T-1}}{\partial V}$$

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

- Each term is the summation and is the derivative of the loss w.r.t. the weights in the output layer . We have already done this in Backpropagation derivation in Unit 2.
- Let us consider a single layer , and it will be its summation over 0 to T-1 or 1 to T

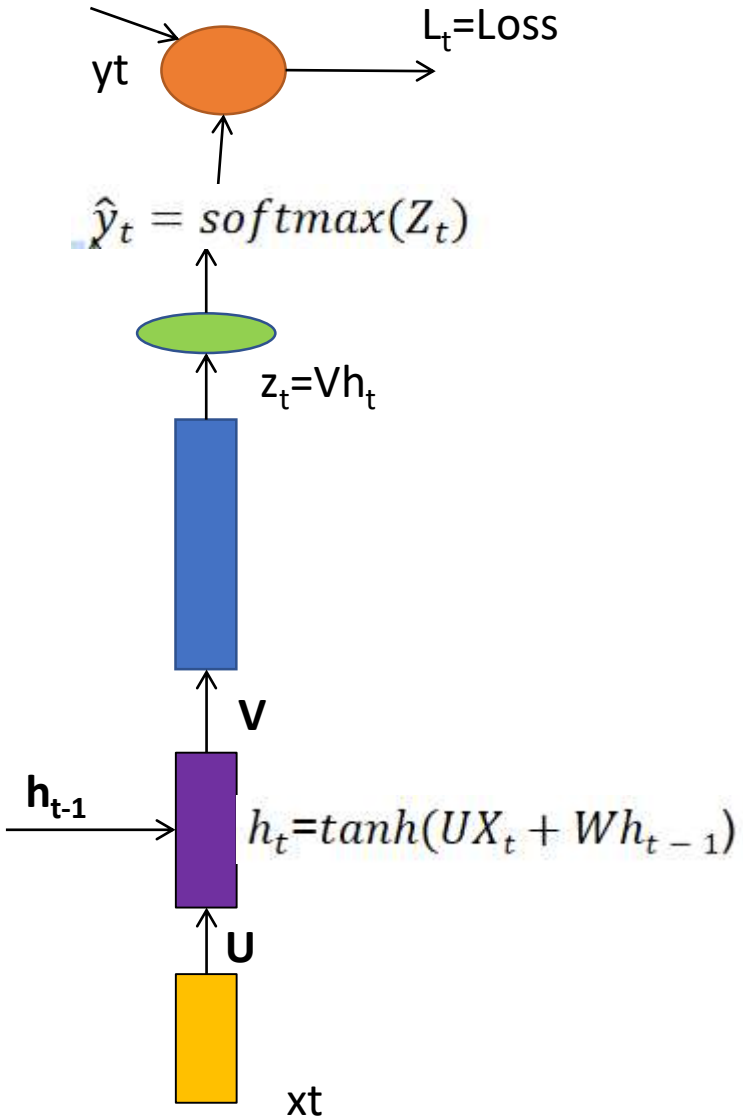
Machine Intelligence

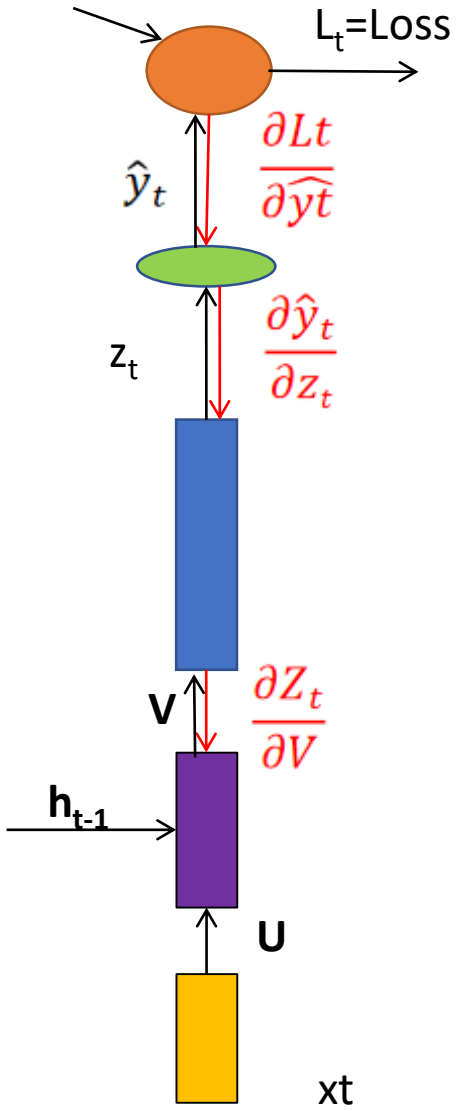
Gradient of loss wrt to V

2 $L_0 = -\, y_0 log(\widehat{y_0})$

$\widehat{y_0}$=softmax(h_0V)

h_0=tanh(UX_0+Wh_{init})





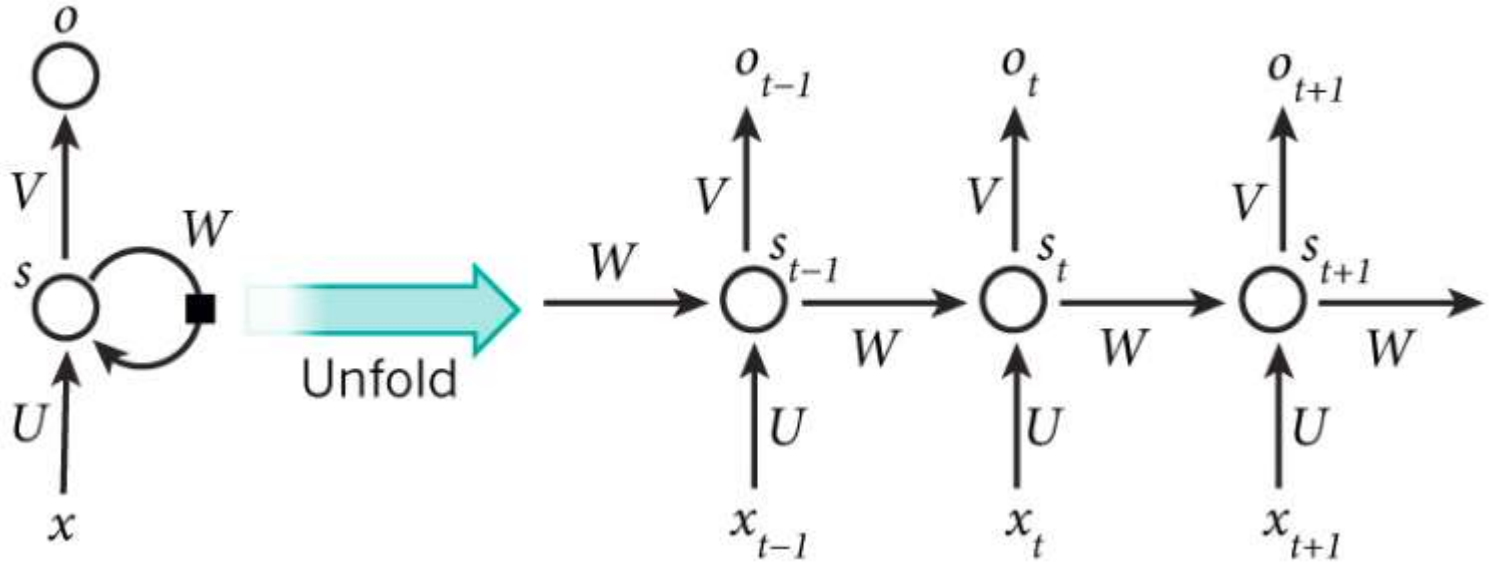
- So for one time stamp, its shown in the figure
- So consider grad of L wrt to V ,with help of chain rule, we will have the following
- solving this using chain rule we get

$$\frac{\partial L}{\partial V} = \sum_{j=0}^{T-1} (\hat{y}_j - y_j) \otimes h_j$$

Machine Intelligence

The Math Behind BPTT

Before we start diving into the calculations, we will start by defining a simple recurrent neural network and unfolding it to explain what happens in each time step t during the execution of back-propagation.



Machine Intelligence

The Math Behind BPTT

- As you may have noticed in the figure, we have three weight matrices involved in the calculation of each output **o** at time step **t** :
  U are the weights associated with the inputs **x_t**.
  W are the weights associated with the hidden state of the RNN.
  V are the weights associated with the outputs of the RNN.

- Training a neural network requires the execution of the forward and back-propagation pass.
- The forward pass involves applying the activation functions on the inputs and returning the predictions at the end ,
- while the back-propagation pass involves the calculation of the objective function gradients' with respect to the weights of the network to finally update those weights.

Machine Intelligence

The Math Behind BPTT

Before we start computing the derivatives, let's start by recalling the equations for the forward pass.

Note:

For simplicity reasons, we assume that the network doesn't use bias and that the activation function is the identity function (f(x)=x).

$$\begin{cases} h_t = Ux_t + Wh_{t-1} \\ o_t = Vh_t \end{cases}$$

where :

h_t : is the hidden state

o_t : is the output at time step t

x_t : is the inputat time step t

Machine Intelligence

The Math Behind BPTT

We will consider the objective function **L** the loss **l** over time steps from the beginning of the sequence. It is defined by the following equation

$$L = \frac{1}{T}\sum_{t=1}^{T} l(o_t, y_t)$$

Now that the objective function has been defined, we can start explaining the computations made in the BPTT algorithm.
Looking to the top of our RNN architecture, the first derivative that we need to compute is the derivative with respect to the weights V that are associated with the outputs **o_t**. For that we will be using the chain rule.

$$\begin{cases} \frac{\partial L}{\partial V} = \frac{\partial L}{\partial o_t} \cdot \frac{\partial o_t}{\partial V} \\ \frac{\partial L}{\partial o_t} = \frac{\partial l(o_t, y_t)}{T . \partial o_t} \\ \frac{\partial o_t}{\partial V} = h_t \\ \frac{\partial L}{\partial V} = \sum_{t=1}^{T} \frac{\partial l}{\partial o_t} . h_t^{\intercal} \end{cases}$$

The Math Behind BPTT

Looking again to the architecture of the network, the next gradient we should compute is the one with respect to **W** (the weights of the hidden state).
For that let's first consider the gradient of **L** at time step **t+1** with respect to the weights **W**, using the chain rule we get

$$\frac{\partial L_{t+1}}{\partial W} = \frac{\partial L_{t+1}}{\partial o_{t+1}} \cdot \frac{\partial o_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial W}$$

From the equations of the forward pass, we know that the hidden state at time step **t+1** is dependent on the hidden state at time step **t**, with that in mind the above equation become

$$\frac{\partial L_{t+1}}{\partial W} = \frac{\partial L_{t+1}}{\partial o_{t+1}} \cdot \frac{\partial o_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_t} \cdot \frac{\partial h_t}{\partial W}$$

Machine Intelligence

The Math Behind BPTT

However, here the things become tricky because also the hidden state at time step **t** is dependent on all the past hidden states, so we need their gradients with respect to **W** and for that we will be using the chain rule for multi-variable function.

Reminder:
The multi-variable chain rule is defined as follow:
*Let **z = f(x,y)** where x and y themselves depend on one or more variables.*
*The multi-variable chain rule allow us to differentiate **z** with respect to any of the variable that are involved.*
*Let **x** and **y** be differentiable at **t** and suppose that **z = f(x,y)** is differentiable at the point (**x(t),y(t)**). Then **z = f(x(t),y(t))** is differentiable at **t** and :*

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial x}.\frac{\partial x}{\partial t} + \frac{\partial z}{\partial y}.\frac{\partial y}{\partial t}$$

The Math Behind BPTT

Great! now it becomes easy for us to compute the gradient of the objective function **L** with respect to **W** for the whole time steps.

$$\frac{\partial L}{\partial W} = \sum_{t}^{T} \sum_{k=1}^{t+1} \frac{\partial L_{t+1}}{\partial o_{t+1}} \cdot \frac{\partial o_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

Finally, we need to compute the gradients with respect to the weights **U** that are associated with the inputs.
Since **U** appears also in the equation of the hidden state just like **W**, using the same pattern to compute the gradients with respect to **W**, we get the gradients with respect to **U** as follow :

$$\frac{\partial L}{\partial U} = \sum_{t}^{T} \sum_{k=1}^{t+1} \frac{\partial L_{t+1}}{\partial o_{t+1}} \cdot \frac{\partial o_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_k} \cdot \frac{\partial h_k}{\partial U}$$

Machine Intelligence

The Math Behind BPTT

Awesome! we are now done with the computations, here are the three gradient equations we computed :

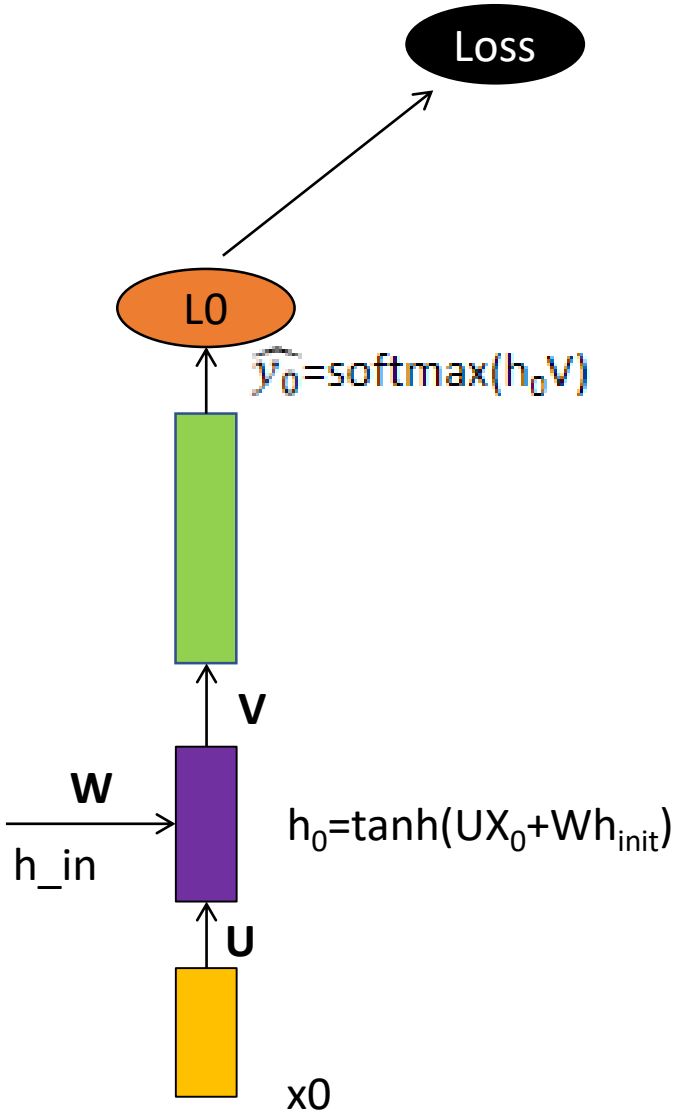
$$\begin{cases} \frac{\partial L}{\partial V} = \sum_{t=1}^{T} \frac{\partial l}{\partial o_t} . h_t^T \\ \frac{\partial L}{\partial W} = \sum_{t}^{T} \sum_{k=1}^{t+1} \frac{\partial L_{t+1}}{\partial o_{t+1}} . \frac{\partial o_{t+1}}{\partial h_{t+1}} . \frac{\partial h_{t+1}}{\partial h_k} . \frac{\partial h_k}{\partial W} \\ \frac{\partial L}{\partial U} = \sum_{t}^{T} \sum_{k=1}^{t+1} \frac{\partial L_{t+1}}{\partial o_{t+1}} . \frac{\partial o_{t+1}}{\partial h_{t+1}} . \frac{\partial h_{t+1}}{\partial h_k} . \frac{\partial h_k}{\partial U} \end{cases}$$

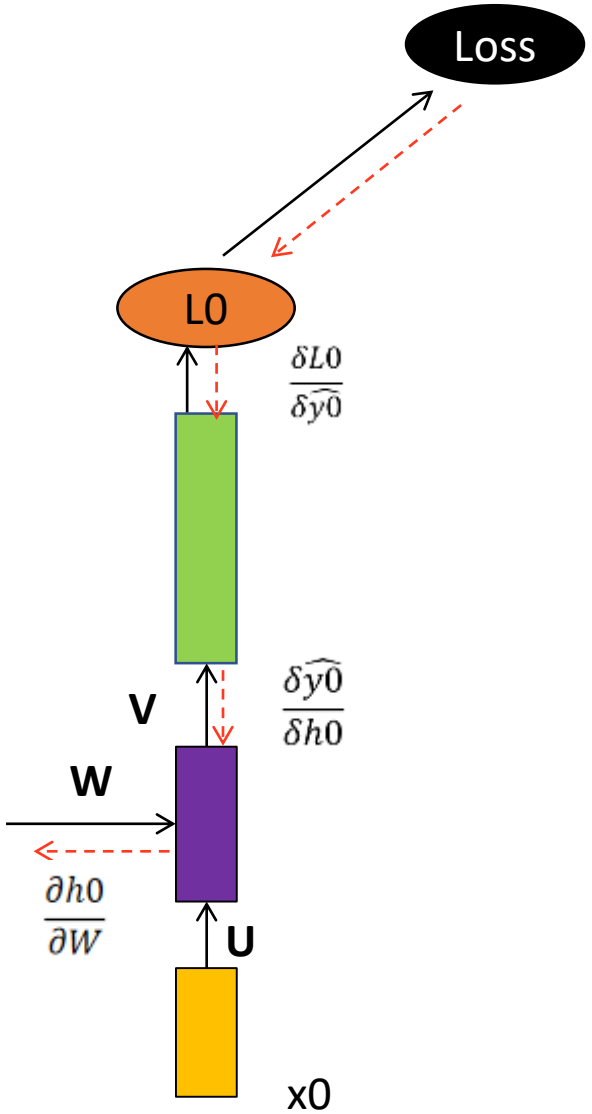
One final thought is: have you noticed that the gradient of the hidden state at time step **t+1** with respect to the hidden state at time step **k** is in itself a chain rule?! If so can you elaborate the two last gradients we computed?
Using the chain rule we have :

$$\prod_{j=k}^{t} \frac{\partial h_{j+1}}{\partial h_j} = \frac{\partial h_{t+1}}{\partial h_k} = \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_{k+1}}{\partial h_k}$$

Machine Intelligence

Grad of loss wrt to W(Pls look into the following slides if you wish to bcoz we have already discussed the math behind it)





let us start with time stamp t=0

2 $L_0 = -\, y_0 log(\hat{y_0})$

$\hat{y_0}$=softmax(h_0V)

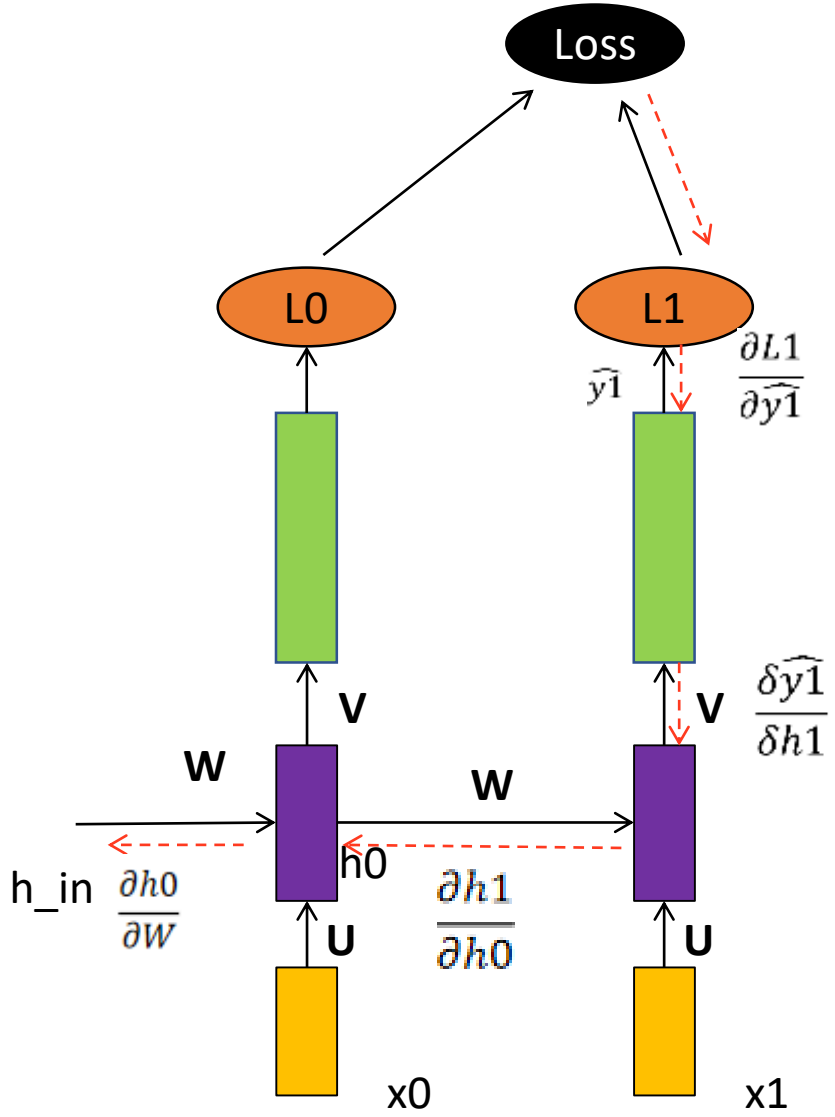
h_0=tanh(UX_0+Wh_{init})

since L 0 and W are not directly related we will use chain rule

$$\frac{\partial L0}{\partial W} = \frac{\partial L0}{\partial \hat{y_0}} \frac{\partial \hat{y_0}}{\partial h0} \frac{\partial h0}{\partial W}$$

Machine Intelligence

Grad of loss wrt to W



let us do for time stamp t=1

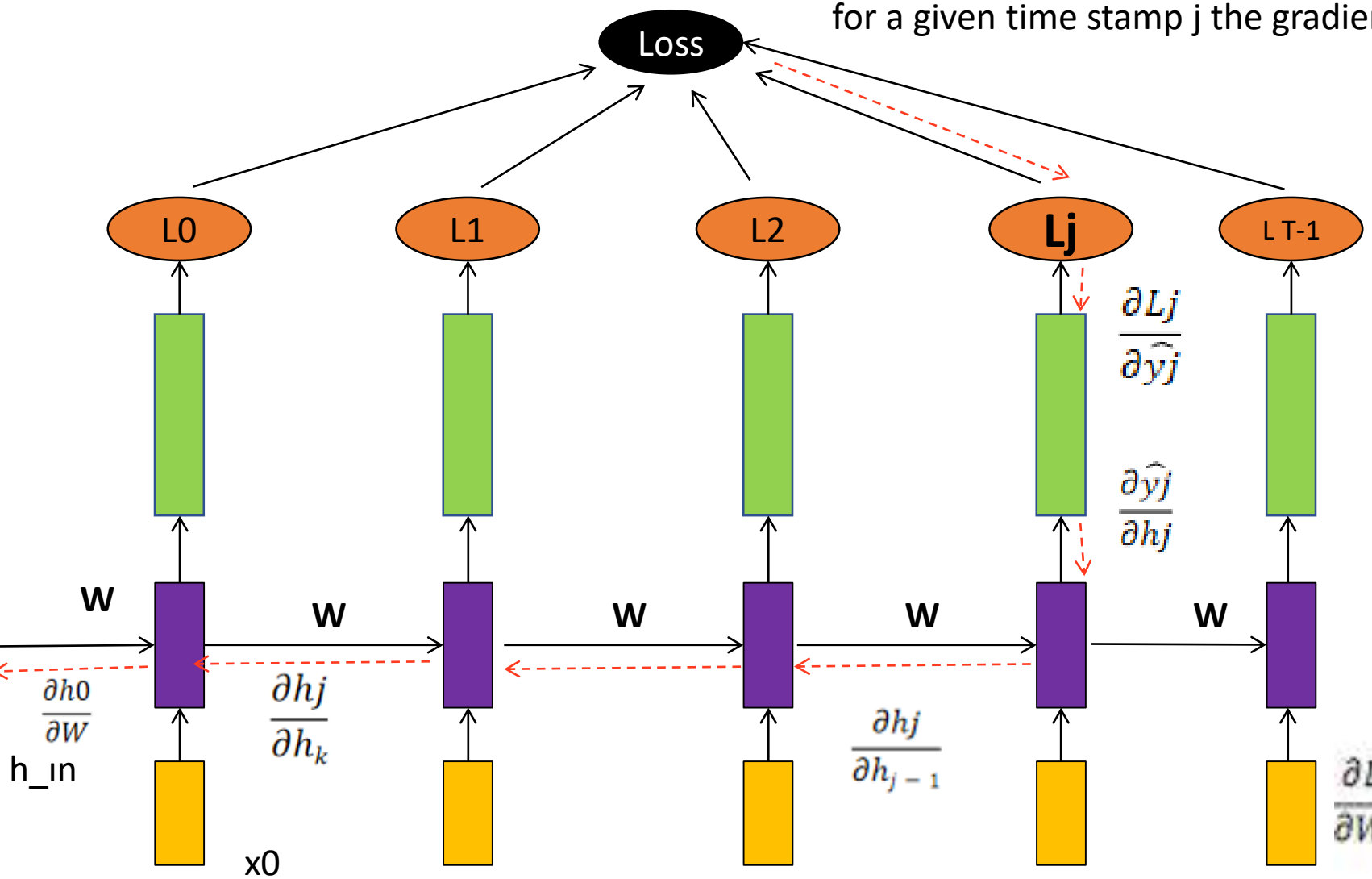
$$\frac{\partial L1}{\partial W} = \frac{\partial L1}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial h1} \frac{\partial h1}{\partial W}$$

- but h1 is not a independent variable its dependent on h0
- Hence we continue our chain rule

$$\frac{\partial L1}{\partial W} = \frac{\partial L1}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial h1} \frac{\partial h1}{\partial W} + \frac{\partial L1}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial h1} \frac{\partial h1}{\partial h0} \frac{\partial h0}{\partial W}$$

Machine Intelligence

Grad of loss wrt to W



for a given time stamp j the gradient will flow in this manner

- This is known as Back propagation through time
- where your gradients flow not just till the current time step ,instead till the initial time step

we can write the generalized equation

$$\frac{\partial L_j}{\partial W} = \sum_{k=0}^{j} \frac{\partial Lj}{\partial yj} \frac{\partial yj}{\partial hj} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\delta hk}{\delta W}$$

summing to all time step

$$\frac{\partial L}{\partial W} = \sum_{j=0}^{T-1} \sum_{k=0}^{j} \frac{\partial Lj}{\partial yj} \frac{\partial yj}{\partial hj} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\delta hk}{\delta W}$$

Machine Intelligence

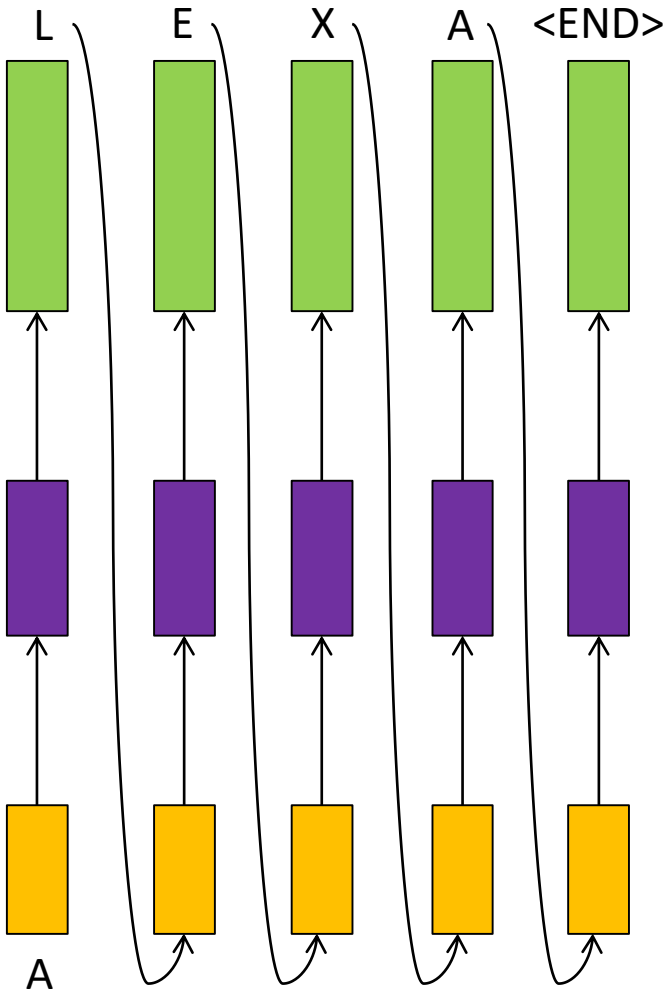
Grad of loss wrt to U (Similar to W)

It will be same as W ,practice and trace the flow of gradients

$$\frac{\partial L}{\partial U} = \sum_{j=0}^{T-1} \sum_{k=0}^{j} (\hat{y}_j - y_j) \prod_{m=k+1}^{j} W^T diag(1 - tanh^2(Wh_{m-1} + Ux_m)) \otimes x_k$$

Machine Intelligence

Various architectures or types of RNN : 1 to 1



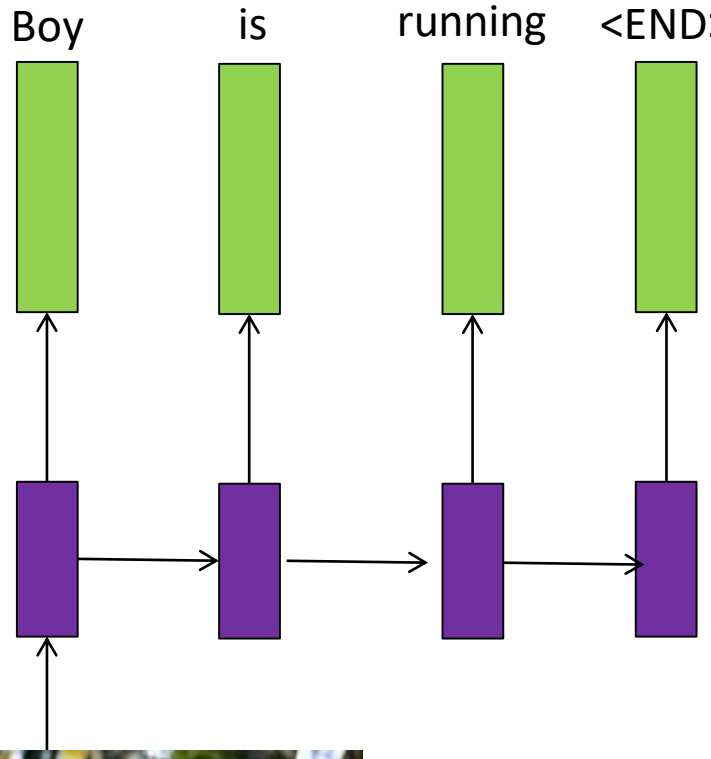
- In a one-to-one architecture, a single input is mapped to a single output, and the output from the time step t is fed as an input to the next time step t+1.
- Such architectures are known as One to One Architecture

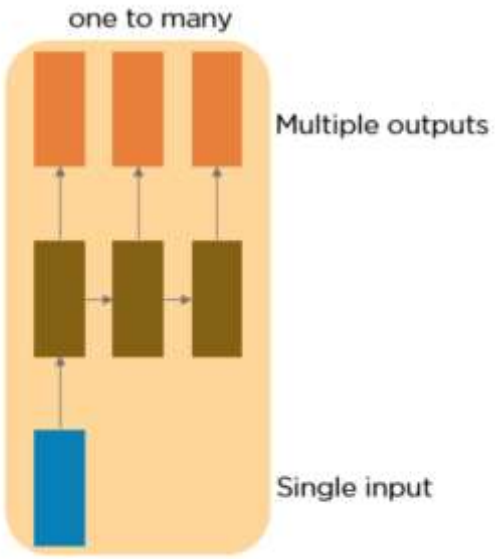
This type of neural network is known as the **Vanilla Neural Network**

Machine Intelligence

One-to-Many

Consider the task of image captioning



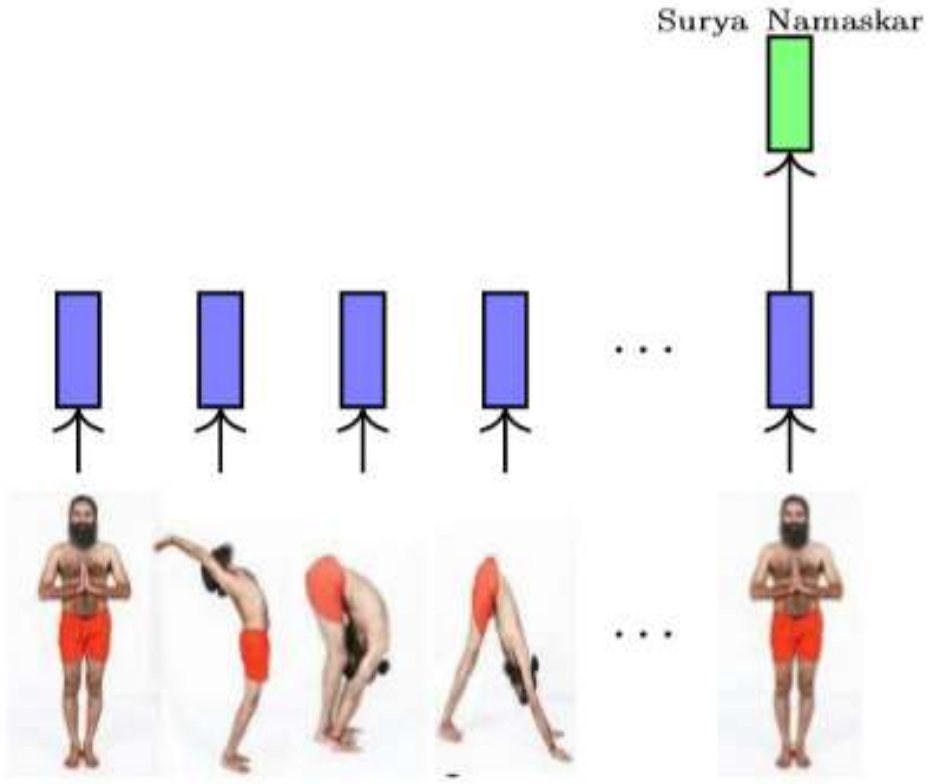


- A single input is mapped to multiple hidden states and multiple output values, which means RNN takes a single input and maps it to an output sequence.
- Although we have a single input value, we share the hidden states across time steps to predict the output.
- Such architectures are known as One to Many Architecture
- An example of this is the image caption or **music generation.**

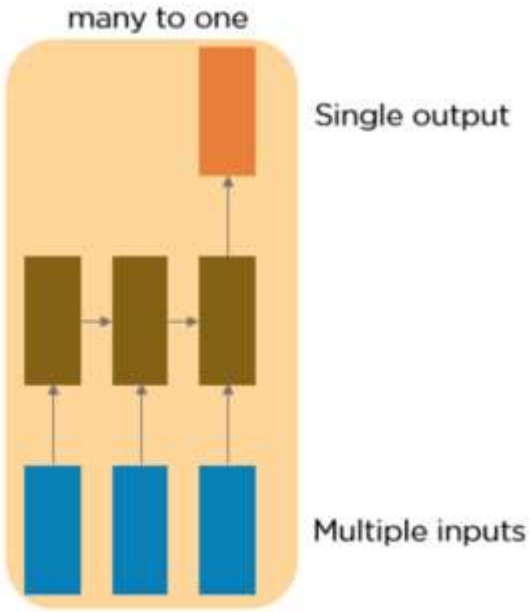
Machine Intelligence

Many-to-One

For example, classifying an action as Suryanamaskar or not Surya namaskar



- The architecture accepts sequence of input and maps it to a single output value.
- Such architecture are known as Many to One Architecture

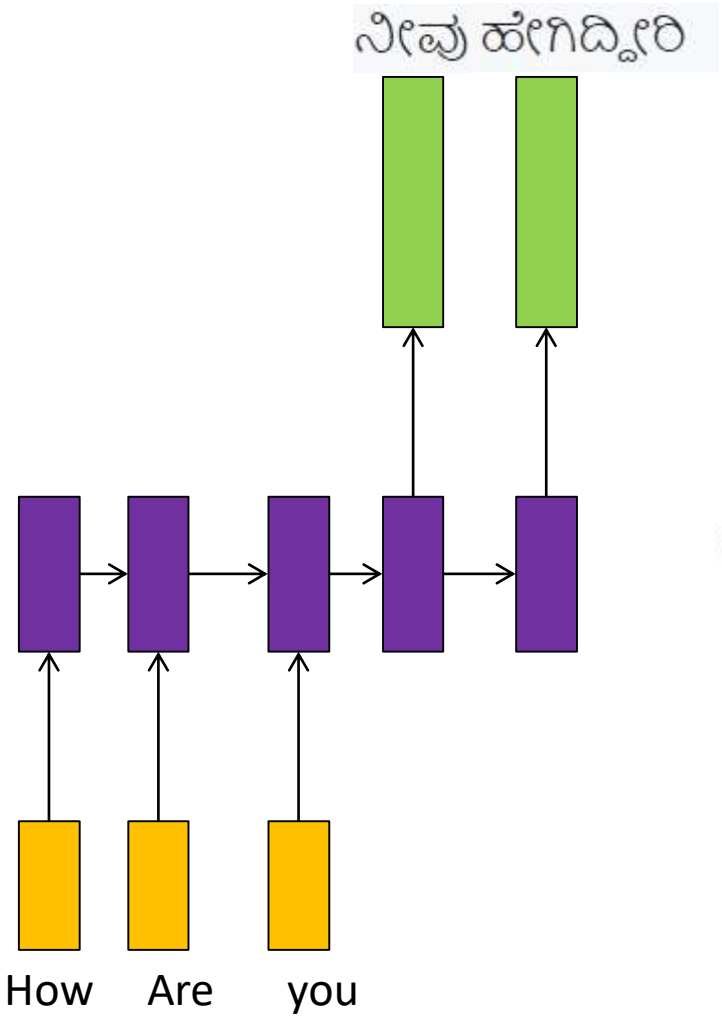


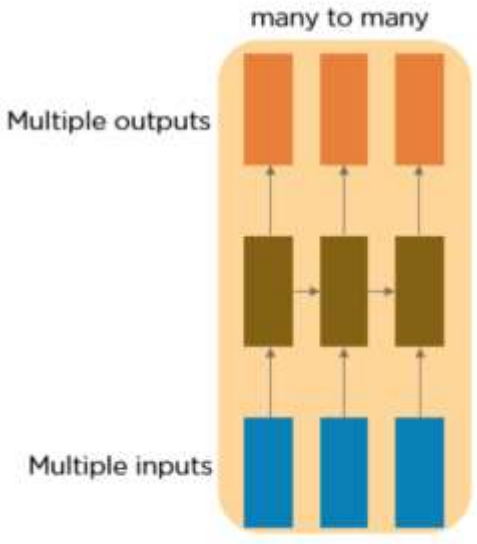
Sentiment analysis is a good example of this kind of network where a given sentence can be classified as expressing positive or negative sentiments.

Machine Intelligence

Many-to-Many

Consider the task of language translation





- Sentence in one language may not be of same size in another language
- Eg:

  English: How are you

  Kannada: ನೀವು ಹೇಗಿದ್ದೀರಿ

- For such problem, we design an architecture that accepts a sequence of input of arbitrary length and maps to a sequence of output of arbitrary length.

Such architecture is known as **Many to Many Architecture**

Machine translation, Named Entity recognition(NER) are its examples.

THANK YOU