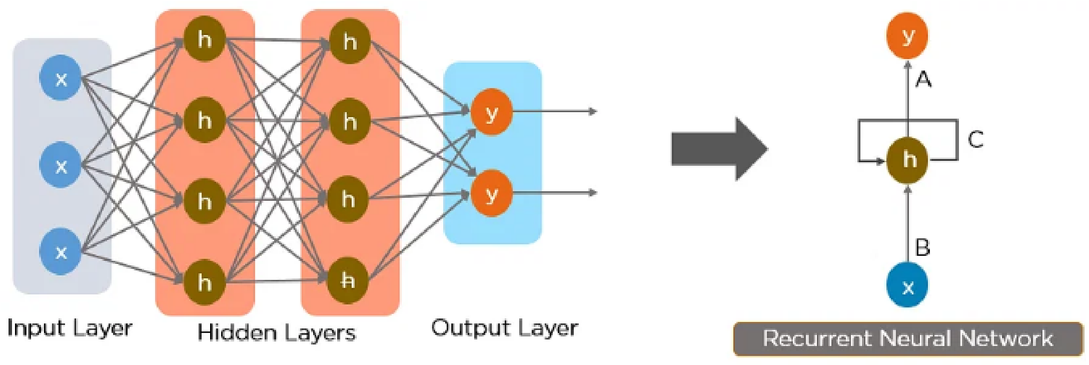
5.1 Recurrent neural networks

A recurrent neural network is a neural network that is specialized for processing a sequence of data x(t)= x(1), . . . , x(τ) with the time step index t ranging from 1 to τ. For tasks that involve sequential inputs, such as speech and language, it is often better to use RNNs. In a NLP problem, if you want to predict the next word in a sentence it is important to know the words before it.

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far.

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step are fed as input to the current step.

RNN have a "memory" which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.



Here, "x" is the input layer, "h" is the hidden layer, and "y" is the output layer. A, B, and C are the network parameters used to improve the output of the model. At any given time t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output.

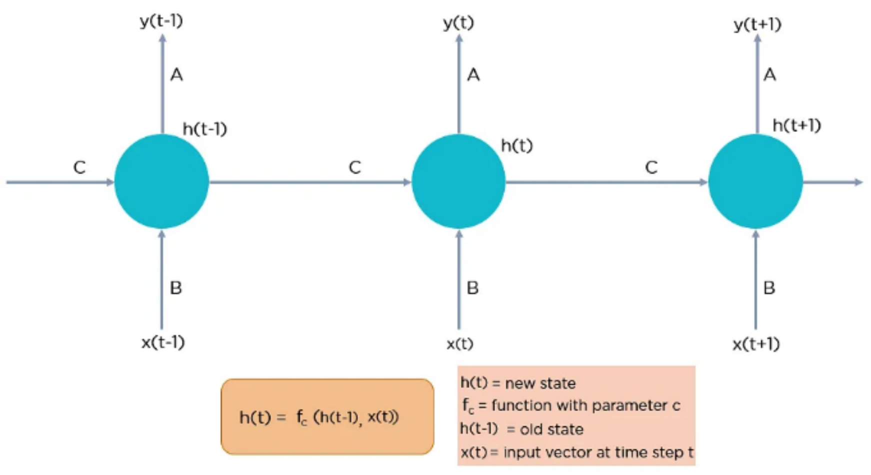
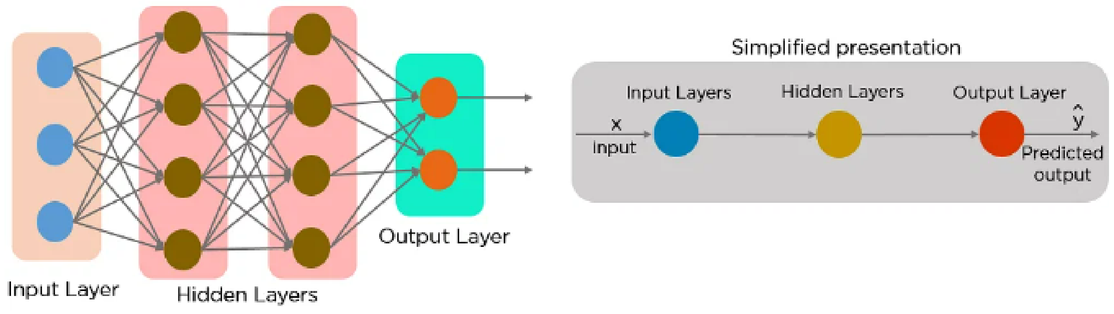

y(t-1)
y(t)
y(t+1)
A
h(t-1)
h(t)
h(t+1)
C
B
x(t-1)
x(t)
x(t+1)
h(t) = f_c (h(t-1), x(t))
h(t) = new state
f_c = function with parameter c
h(t-1) = old state
x(t) = input vector at time step t

Fig: Fully connected Recurrent Neural Network

Feed-Forward Neural Networks vs Recurrent Neural Networks:

A feed-forward neural network allows information to flow only in the forward direction, from the input nodes, through the hidden layers, and to the output nodes. There are no cycles or loops in the network.



In a feed-forward neural network, the decisions are based on the current input. It doesn't memorize the past data, and there's no future scope. Feed-forward neural networks are used in general regression and classification problems.

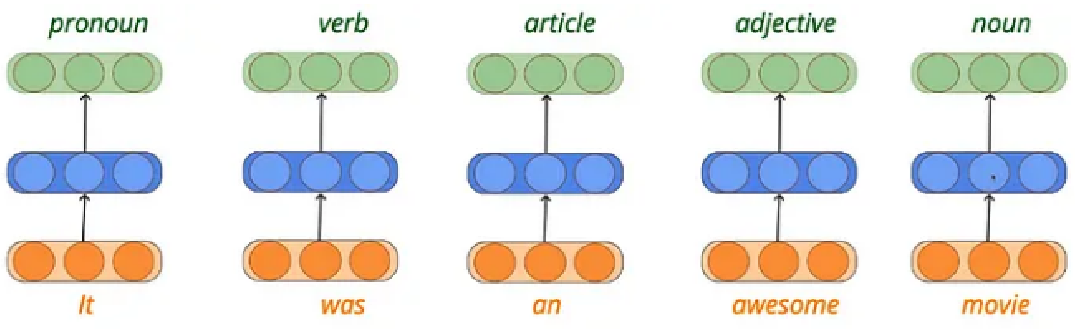
I. Sequence Learning Problems

In all of the networks that we have covered so far(Fully Connected Neural Network(FCNN), Convolutional Neural Network(CNN)):

- the output at any time step is independent of the previous layer input/output
- the input was always of the fixed-length/size

  In "Sequence Learning Problems", the "two properties of FCNN and CNNs do not hold" and the output at any timestep depends on previous input/output and the length of the input is not fixed.

1. Part of speech tagging:
   Given a "sequence of words", the idea is to "predict part of the speech tag for each word" (whether that word is a pronoun, noun, verb, article, adjective, and so on)

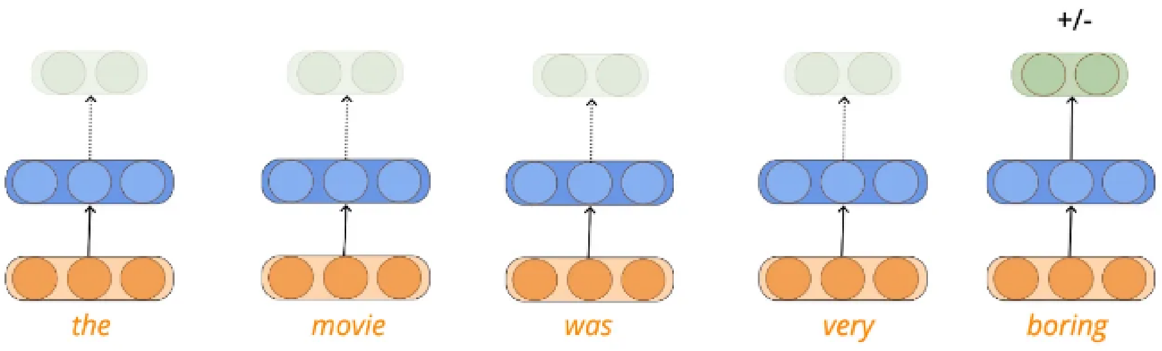


Here also the "output depends" not only on the "current input" but "also on the previous input(s)" for example:
Say the current input as 'movie' (last layer in the snippet above) and the previous input was 'awesome' which is an adjective; "the moment model sees an adjective" it would be more or less confident that the "next word is actually going to be a noun"

So, the confidence in predicting that the “movie (word) is a noun” would be higher “if the previous input was an adjective”.

2. **Sentimental Analysis:**
   It’s not mandatory to produce an output at each step(time step/layer) for example for sentiment analysis, the model looks at all the words in a sentence and gives/predicts a final output as positive/negative sentiment conveyed by the sentence.

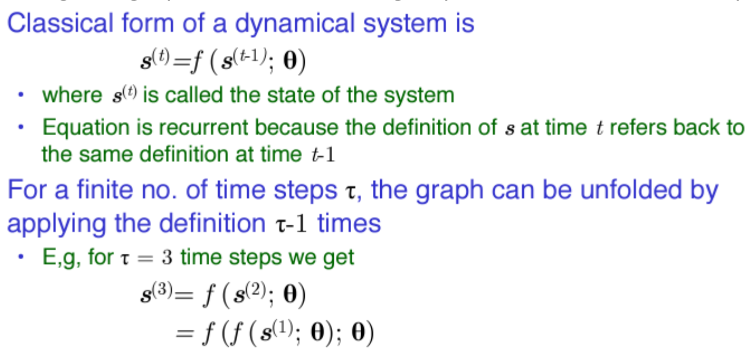
   This could be considered as output is produced at every time step but the model ignore those output and reports only the final output (and somehow this final output is dependent on all the previous inputs as well). This is also termed a Sequence Classification Problem.



3. **Sequence Learning problems using video and speech data:**
   Speech Recognition —Think of speech as a sequence of phonemes and give the speech signal as the input, the idea would be to map each signal to its respective phoneme in the language
   Video Labeling- A video is a sequence of frames (there might be some processing on these frames), one task could be to label every frame in the video.

II. <u>**Unfolding Computational graphs:**</u>
   A Computational Graph is a way to formalize the structure of a set of computations. Such as mapping inputs and parameters to outputs and loss. We can unfold a recursive or recurrent computation into a computational graph that has a repetitive structure corresponding to a chain of events.

   Unfolding this graph results in sharing of parameters across a deep network structure.



Classical form of a dynamical system is

$$s^{(t)} = f\left(s^{(t-1)}; \boldsymbol{\theta}\right)$$

- where $s^{(t)}$ is called the state of the system
- Equation is recurrent because the definition of s at time t refers back to the same definition at time t-1

For a finite no. of time steps τ, the graph can be unfolded by applying the definition τ-1 times

- E,g, for $\tau = 3$ time steps we get

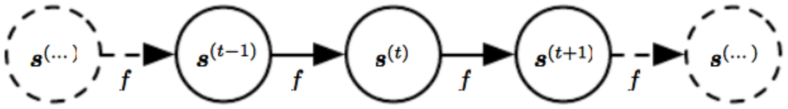
1.
$$\begin{aligned} s^{(3)} &= f\left(s^{(2)}; \boldsymbol{\theta}\right) \\ &= f\left(f\left(s^{(1)}; \boldsymbol{\theta}\right); \boldsymbol{\theta}\right) \end{aligned}$$

Unfolding equation by repeatedly applying the definition in this way has yielded expression without recurrence.

The classical dynamical system described by

$$s^{(t)} = f(s^{(t-1)}; \theta) \text{ and } s^{(3)} = f(f(s^{(1)}; \theta); \theta)$$

is illustrated as an unfolded computational graph



Each node represents state at some time t

Function f maps state at time t to the state at $t+1$

The same parameters (the same value of θ used to parameterize f) are used for all time steps

Dynamical system driven by external signal:

As another example, consider a dynamical system driven by external (input) signal $x^{(t)}$

$s^{(t)}=f(s^{(t-1)}, x^{(t)}; \theta)$.

State now contains information about the whole past input sequence.

Note that the previous dynamic system was simply $s^{(t)}=f(s^{(t-1)}; \theta)$.

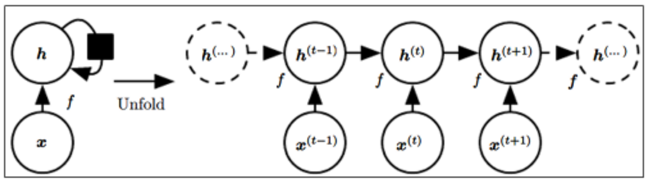
2. **Recurrent neural networks can be built in many ways:**
   Much as almost any function is a feedforward neural network, any function involving recurrence can be considered to be a recurrent neural network.

Many recurrent neural nets use same equation (as dynamical system with external input) to define values of hidden units

- To indicate that the state is hidden rewrite using variable h for state:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$

- Illustrated below:



Typical RNNs have extra architectural features such as output layers that read information out of state h to make predictions

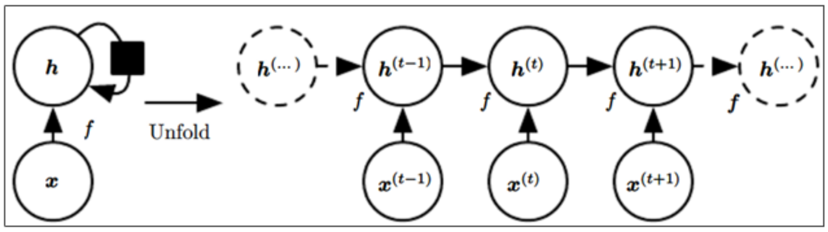
3.

A recurrent network with no outputs

- This network just processes information from input $\boldsymbol{x}$ by incorporating it into state $\boldsymbol{h}$ that is passed forward through time

Circuit diagram: Black square indicates Delay of one time step

Unfolded computational graph: each node is now associated with one time instance



Typical RNNs will add extra architectural features such as output layers to read information out of the state $\boldsymbol{h}$ to make predictions

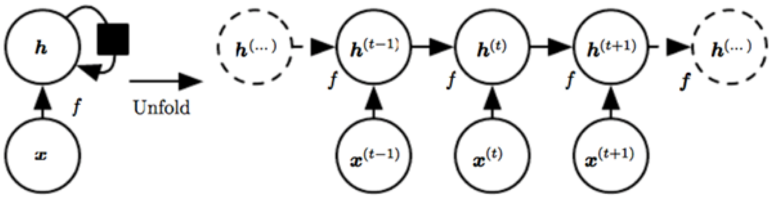
Predicting the Future from the Past

When RNN is required to perform a task of predicting the future from the past, network typically learns to use $\boldsymbol{h}^{(t)}$ as a lossy summary of the task-relevant aspects of the past sequence of inputs upto t

The summary is in general lossy since it maps a sequence of arbitrary length $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t\text{-}1)},..,\boldsymbol{x}^{(2)},\boldsymbol{x}^{(1)})$ to a fixed length vector $\boldsymbol{h}^{(t)}$

Unfolding: from circuit diagram to computational graph

- Equation $\boldsymbol{h}^{(t)}=f\,(\,\boldsymbol{h}^{(t\text{-}1)},\ \boldsymbol{x}^{(t)};\ \boldsymbol{\theta}\ )$ can be written in two different ways: circuit diagram or an unfolded computational graph



- Unfolding is the operation that maps a circuit to a computational graph with repeated pieces
- The unfolded graph has a size dependent on the sequence length

Process of Unfolding

We can represent unfolded recurrence after t steps with a function $g^{(t)}$:

$$h^{(t)} = g^{(t)}\left(x^{(t)},\ x^{(t-1)}, \ldots, x^{(2)}, x^{(1)}\right)$$
$$= f\left(h^{(t-1)},\ x^{(t)};\ \theta\right)$$

Function $g^{(t)}$ takes in whole past sequence $\left(x^{(t)},\ x^{(t-1)}, \ldots, x^{(2)}, x^{(1)}\right)$ as input and produces the current state but the unfolded recurrent structure allows us to factorize $g^{(t)}$ into repeated application of a function f

The unfolding process introduces two major advantages as discussed next.

The unfolding process introduces two major advantages:

1. Regardless of sequence length, learned model has same input size:
   - because it is specified in terms of transition from one state to another state rather than specified in terms of a variable length history of states
2. Possible to use same function f with same parameters at every step:

These two factors make it possible to learn a single model f

- that operates on all time steps and all sequence lengths
- rather than needing separate model g(t) for all possible time steps

Learning a single shared model allows:

- Generalization to sequence lengths that did not appear in the training
- Allows model to be estimated with far fewer training examples than would be required without parameter sharing.

Both recurrent graph and unrolled graph are useful

Recurrent graph is succinct

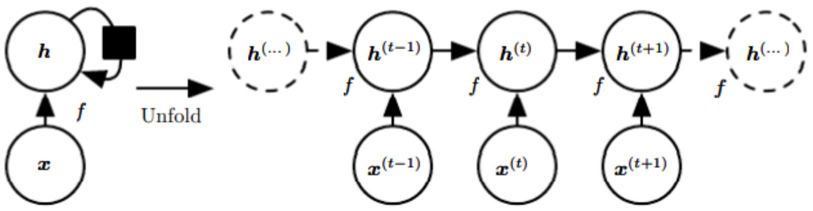
Unrolled graph provides explicit description of what computations to perform

- Helps illustrate the idea of information flow forward in time
  - Computing outputs and losses
- And backwards in time
  - Computing gradients
- By explicitly showing path along which information flows

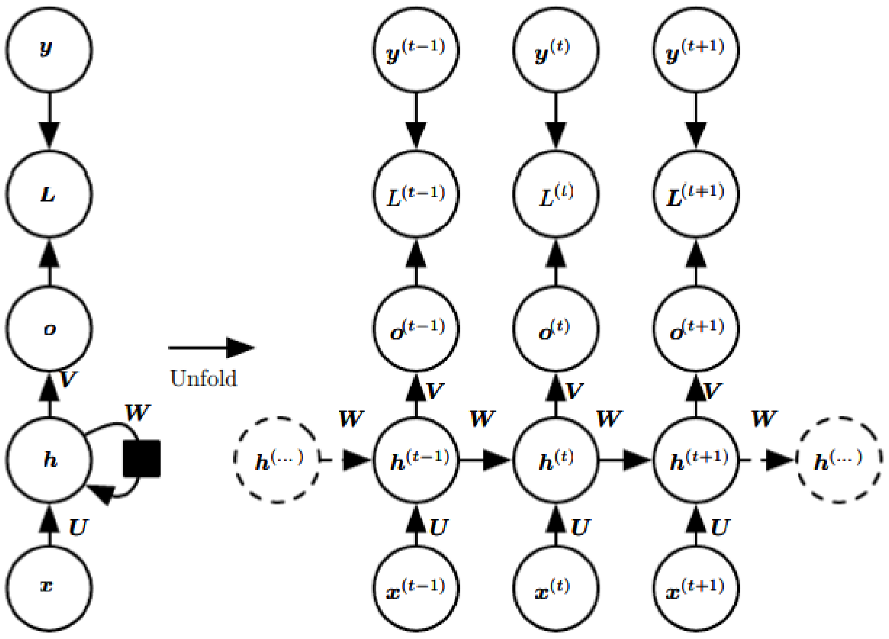
III. Recurrent Neural Network

Recurrent Neural Networks (RNN) are a part of the neural network's family used for processing sequential data. For example, consider the following equation:

$$h^t = f(h^{t-1}; x) \qquad \text{e.q 1}$$



A. Architecture:



Architecture of recurrent neural network where x, h, o, L, y represents input, hidden state, output, loss, and target value respectively.

Recurrent Neural Network maps an input sequence x values to a corresponding sequence of output 'o' values.

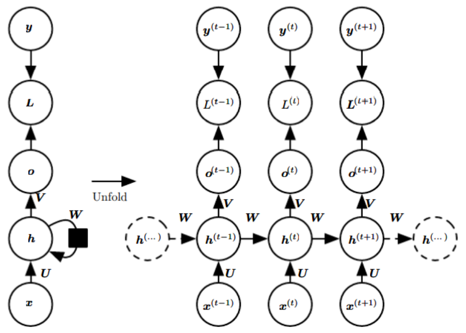
A loss 'L' measure the difference between the actual output y and the predicted output o.

The RNN has also input to hidden connection parameterized by a weight matrix U, hidden to hidden connections parameterized by a weight matrix W, and hidden-to-output connections parameterized by a weight matrix V.

The softmax operation is applies as a post-processing step to obtain a vector y^ of normalized probability of the output.

Then from time step t = 1 to t = n we apply the following equation:

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$



These are the **forward propagation equations** of the recurrent neural network where U, V, W are the weight matrix that is shared among each time step.

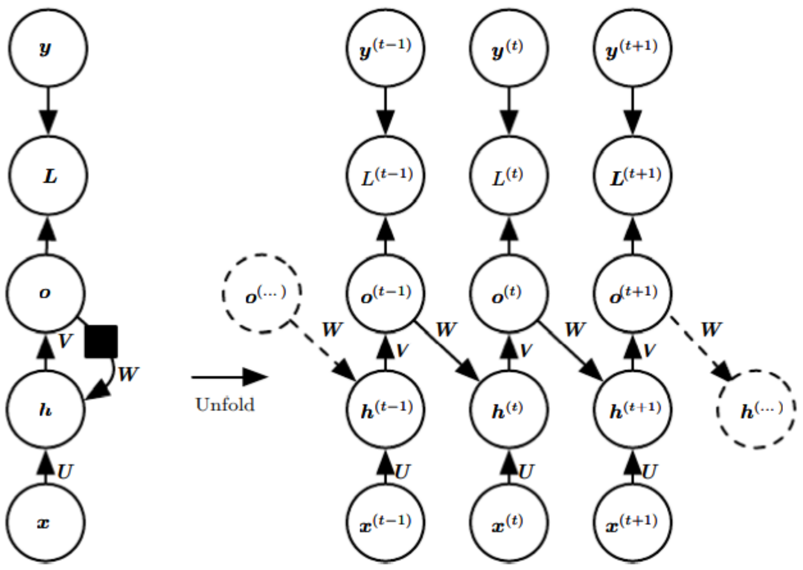
The above equations are also known as forwarding propagation of RNN where the b and c are the bias vectors and tanh and softmax are the activation functions.

To update the weight matrix U, V, W we calculate the gradient of the loss function for each weight matrix

i.e. ∂L/∂U, ∂L/∂V, ∂L/∂W, and update each weight matrix with the help of a back-propagation algorithm.

When a back-propagation algorithm is applied to RNN, it is sometimes also known as BPTT i.e. backpropagation through time.

B. Another variation that can be done in recurrent neural network architecture is that we can change the recurrent connection from hidden to hidden state and make it from output to hidden state.



Example of RNN:

This is an example of an RNN that maps an input sequence to an output sequence of the same length

Total loss for a given sequence for a given sequence of $\boldsymbol{x}$ values with a sequence of $\boldsymbol{y}$ values is the sum of the losses over the time steps

If $L^{(t)}$ is the negative log-likelihood of $\boldsymbol{y}^{(t)}$ given $\boldsymbol{x}^{(1)},..\boldsymbol{x}^{(t)}$ then

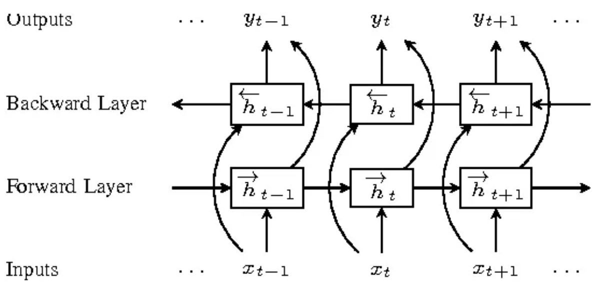
$$L\left(\left\{\boldsymbol{x}^{(1)},..\boldsymbol{x}^{(t)}\right\},\left\{\boldsymbol{y}^{(1)},..\boldsymbol{y}^{(t)}\right\}\right)=\sum_t L^{(t)}$$
$$=-\sum_t \log p_{\text{model}}\left(\boldsymbol{y}^{(t)} \mid \left\{\boldsymbol{x}^{(1)},..\boldsymbol{x}^{(t)}\right\}\right)$$

- where p_{model} is given by reading the entry for $\boldsymbol{y}^{(t)}$ from the model's output vector $\hat{\boldsymbol{y}}^{(t)}$

(The negative log-likelihood function is commonly used as a loss function in machine learning because it is a natural measure of the distance between the predicted probability distribution and the true probability distribution)

IV. <u>Bidirectional RNNs</u>

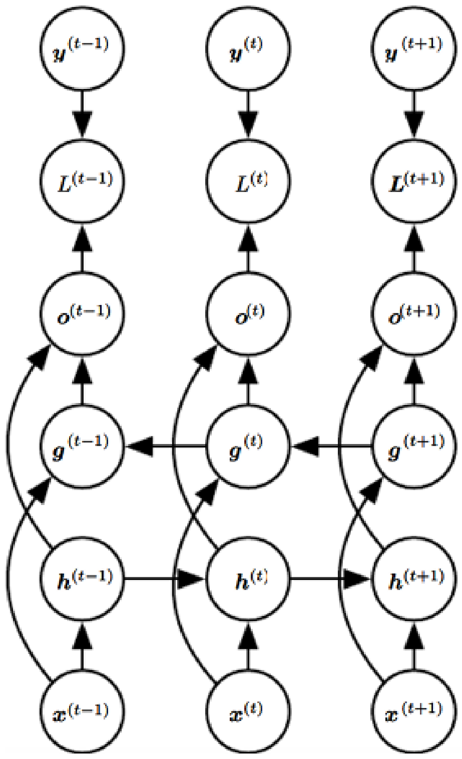
Combine an RNN that moves forward through time from the start of the sequence
Another RNN that moves backward through time beginning from the end of the sequence
A bidirectional RNN consists of two RNNs which are stacked on the top of each other.
The one that processes the input in its original order and the one that processes the reversed input sequence.
The output is then computed based on the hidden state of both RNNs.



Outputs
$\cdots$ y_{t-1} y_t y_{t+1} $\cdots$
Backward Layer
$\overleftarrow{h}_{t-1}$ $\overleftarrow{h}_{t}$ $\overleftarrow{h}_{t+1}$
Forward Layer
$\overrightarrow{h}_{t-1}$ $\overrightarrow{h}_{t}$ $\overrightarrow{h}_{t+1}$
Inputs
$\cdots$ x_{t-1} x_t x_{t+1} $\cdots$

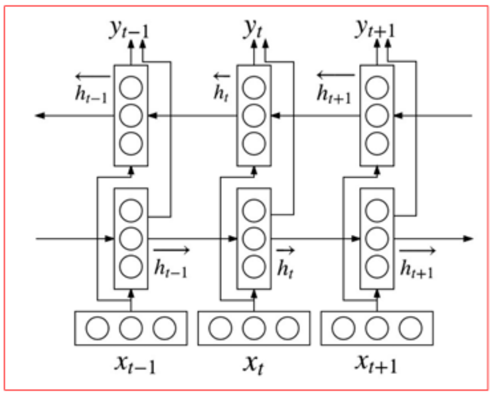
Bi Directional RNN for word sequence

Consider the word sequence “I love mango juice”. The forward layer would feed the sequence as such. But, the Backward Layer would feed the sequence in the reverse order “juice mango love I”. Now, the outputs would be generated by concatenating the word sequences at each time and generating weights accordingly. This can be used for POS tagging problems as well.



$y^{(t-1)}$ $y^{(t)}$ $y^{(t+1)}$
$L^{(t-1)}$ $L^{(t)}$ $L^{(t+1)}$
$o^{(t-1)}$ $o^{(t)}$ $o^{(t+1)}$
$g^{(t-1)}$ $g^{(t)}$ $g^{(t+1)}$
$h^{(t-1)}$ $h^{(t)}$ $h^{(t+1)}$
$x^{(t-1)}$ $x^{(t)}$ $x^{(t+1)}$

Maps input sequences x to target sequences y with loss $L^{(t)}$ at each step t
h recurrence propagates to the right g recurrence propagates to the left.
This allows output units $o^{(t)}$ to compute a representation that depends both the past and the future.



y_{t-1}
y_t
y_{t+1}
$\overleftarrow{h}_{t-1}$
$\overleftarrow{h}_t$
$\overleftarrow{h}_{t+1}$
$\overrightarrow{h}_{t-1}$
$\overrightarrow{h}_t$
$\overrightarrow{h}_{t+1}$
x_{t-1}
x_t
x_{t+1}

Need for bidirectionality:
In speech recognition, the correct interpretation of the current sound may depend on the next few phonemes because of co-articulation and the next few words because of linguistic dependencies.