

# Sequence Modelling

## RNN, LSTM, TDNN

Dr. K. T. Deepak

Assistant Professor, ECE Department  
IIIT Dharwad

## References

- Artificial Neural Networks, B. Yegnanarayana, Prentice Hall of India 2015.
- Neural Networks A Comprehensive Foundation, Simon Haykin, 2nd Edition.
- Neural Networks and Deep Learning, Charu C. Agarwal, Springer International.
- Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

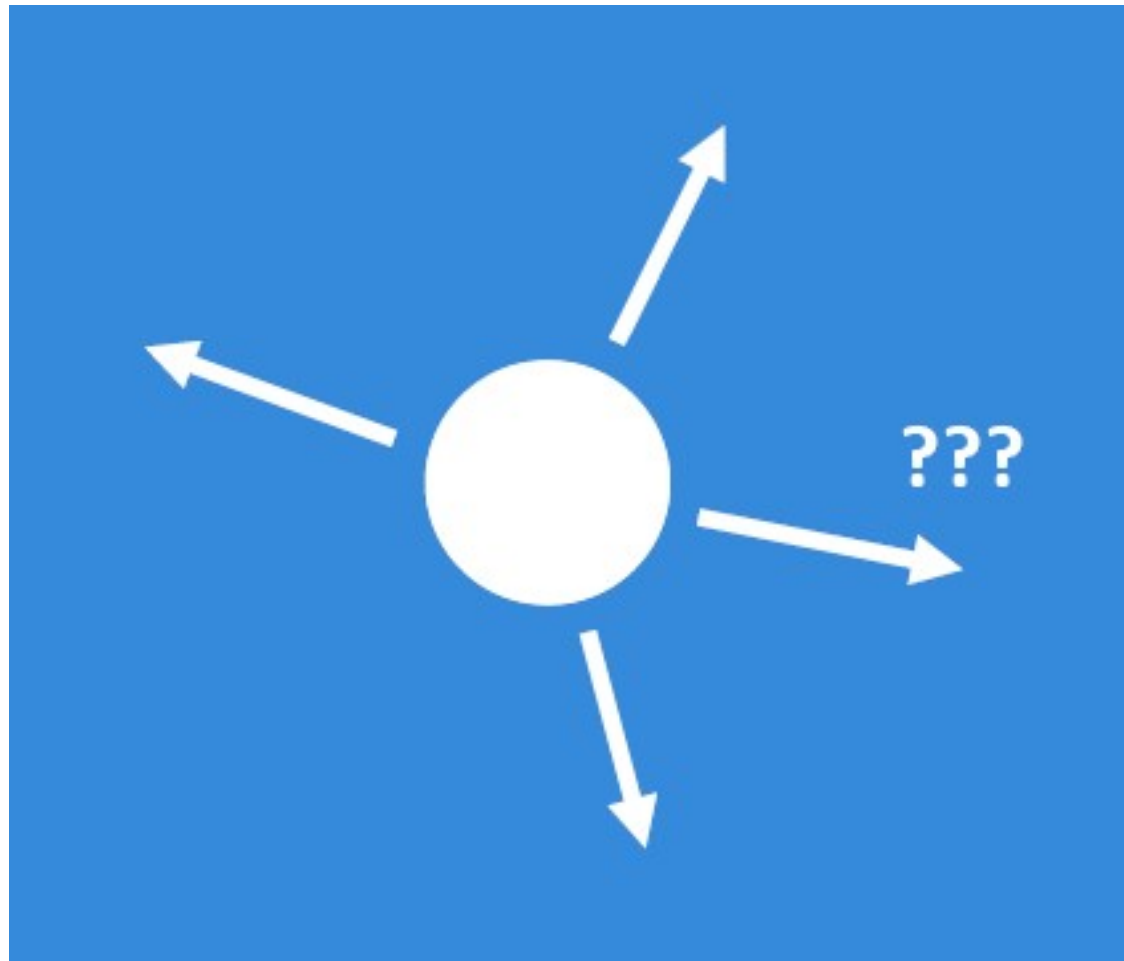
- MIT: Introduction to Deep Learning  
<http://introtodeeplearning.com/>
- Simplilearn: <https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn>
- AnalyticsVidya:  
<https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/>
- <https://www.geeksforgeeks.org/ml-back-propagation-through-time/>
- <http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/>

# Importance of Sequence Modelling

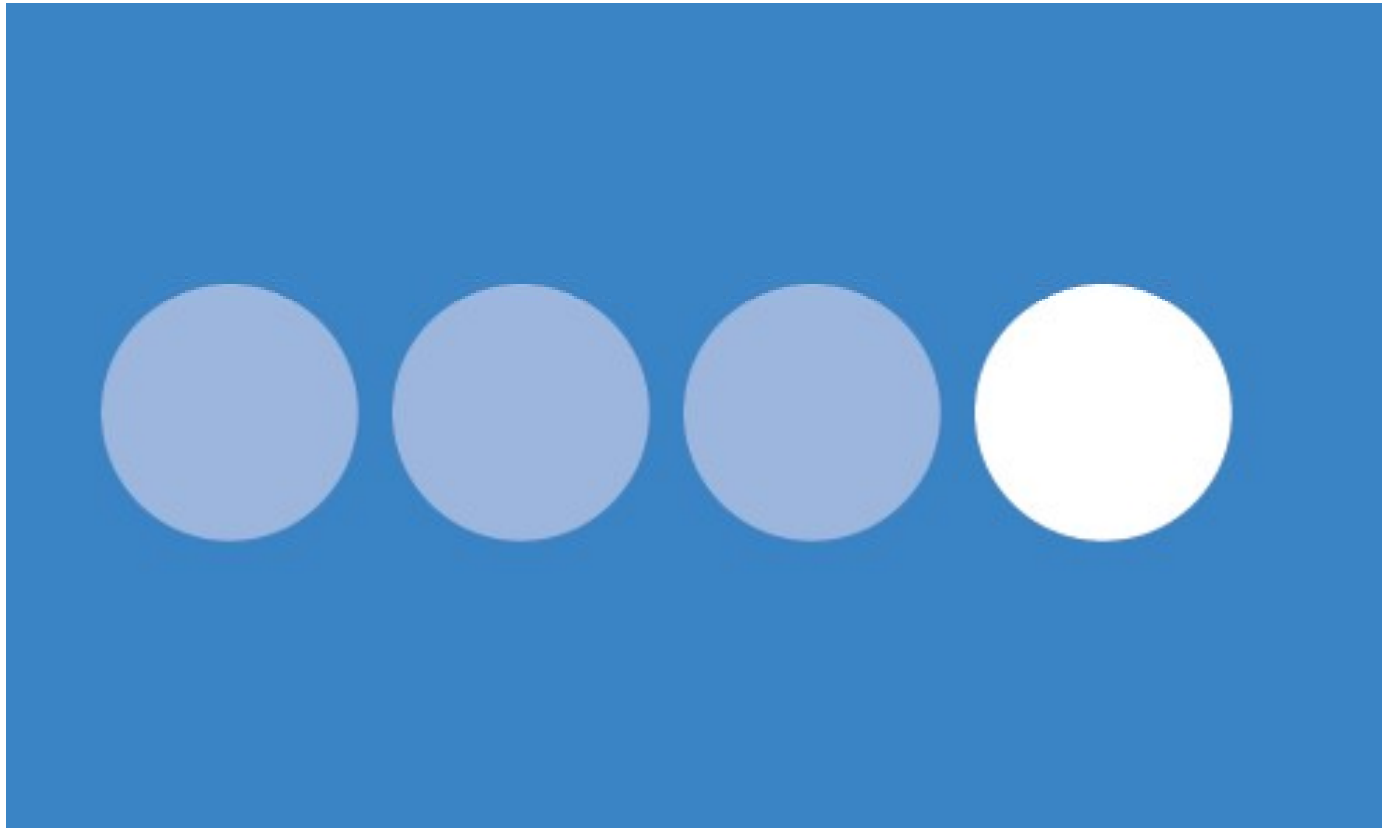
Given an image of a ball,  
can you predict where it will go next?



Given an image of a ball,  
can you predict where it will go next?



Given an image of a ball,  
can you predict where it will go next?



# Sequence Modelling

- In general the neural architectures are inherently designed for multidimensional data in which the attributes are **largely independent** of one another.
- However, certain data types such as time-series (speech, audio, video), text, and biological data contain **sequential dependencies** among the attributes.

# Time Series Data

- In a time-series data set, the values on **successive time-stamps** are closely **related to one another**.
- If one uses the values of these time-stamps as **independent features**, then key information about the **relationships** among the values of these **time-stamps** is **lost**.
- For example, the value of speech at time  $t$  *is closely related to its values in* the previous window.
- This information is lost when the values at individual time-stamps are treated independently of one another.

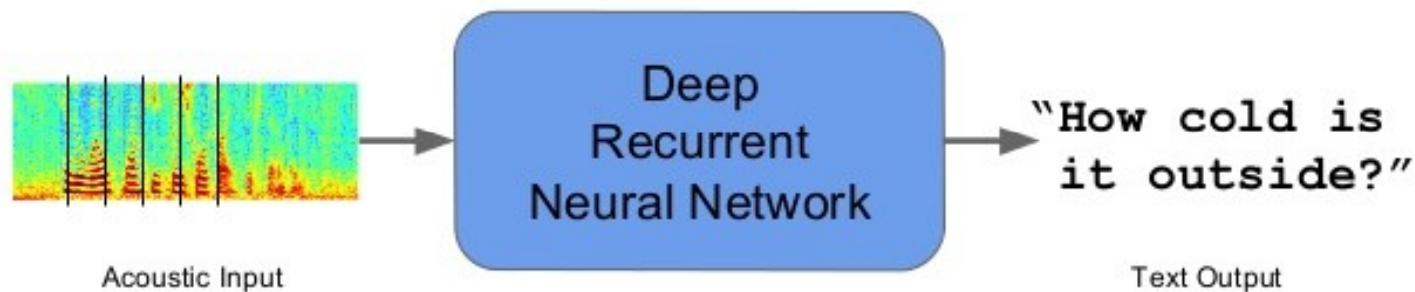


# Textual Data

- Although **text** is often processed as a **bag of words**, one can obtain **better semantic insights** when the ordering of the words is used.
- In such cases, it is **important to construct models** that take the **sequencing information** into account.
- **Text data** is the **most common use case** of recurrent neural networks.

# Applications

## Speech Recognition



Reduced word errors by more than 30%

Google Research Blog - August 2012, August 2015



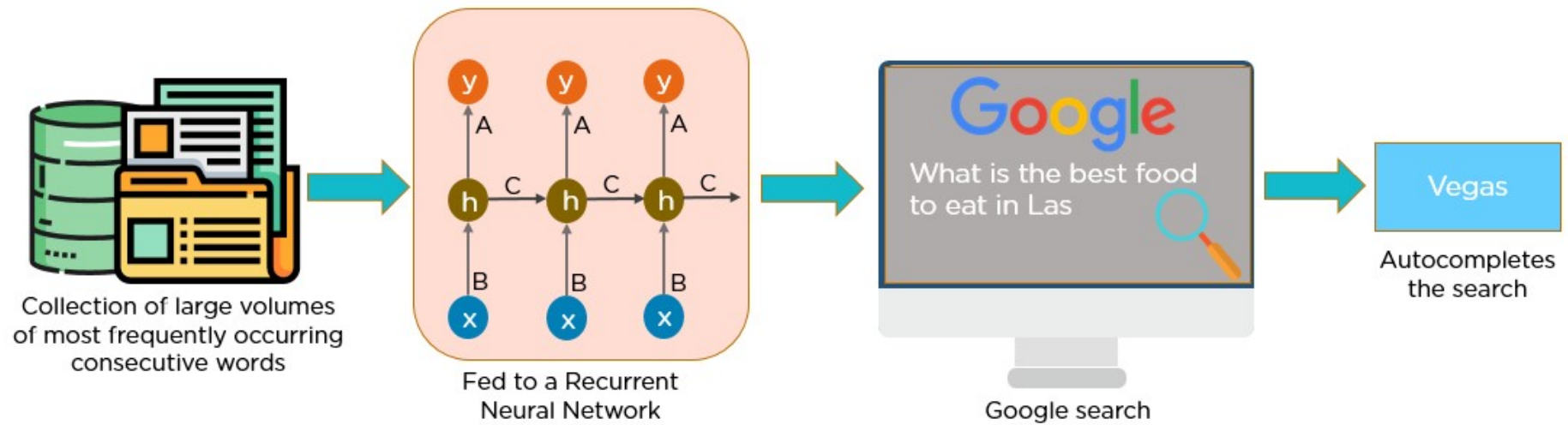
# Applications



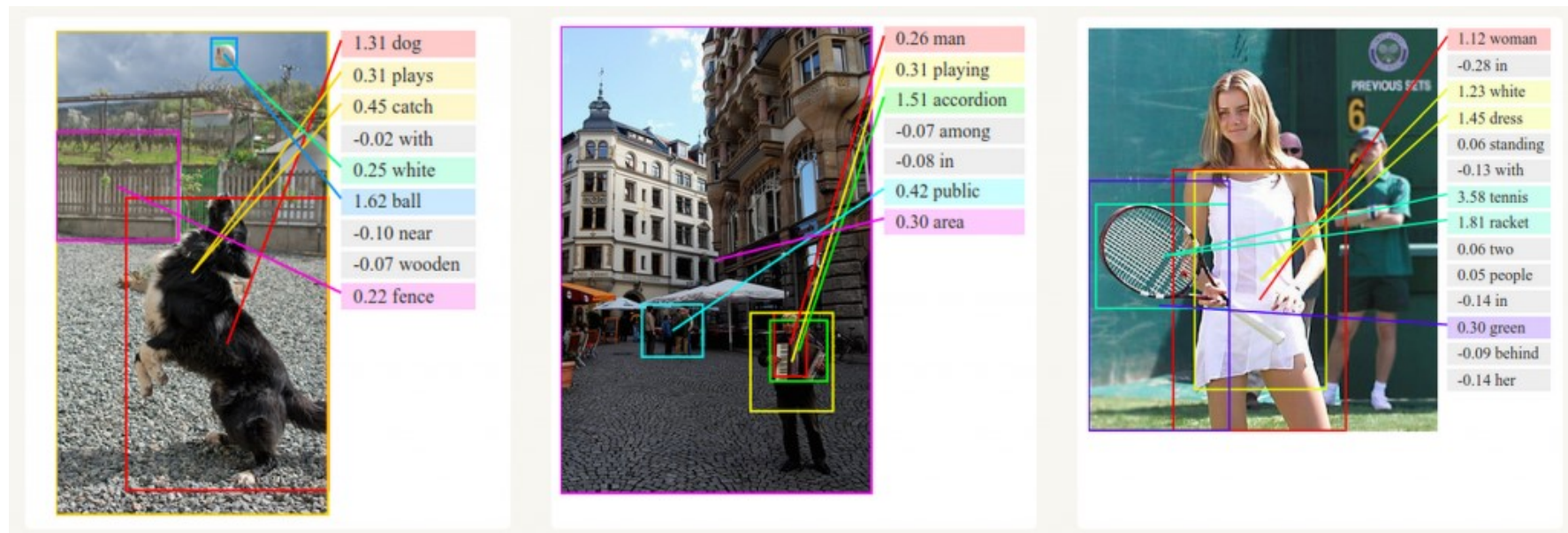
Machine Translation

Here the person is speaking in English and it is getting translated into Chinese, Italian, French, German and Spanish languages

# Applications



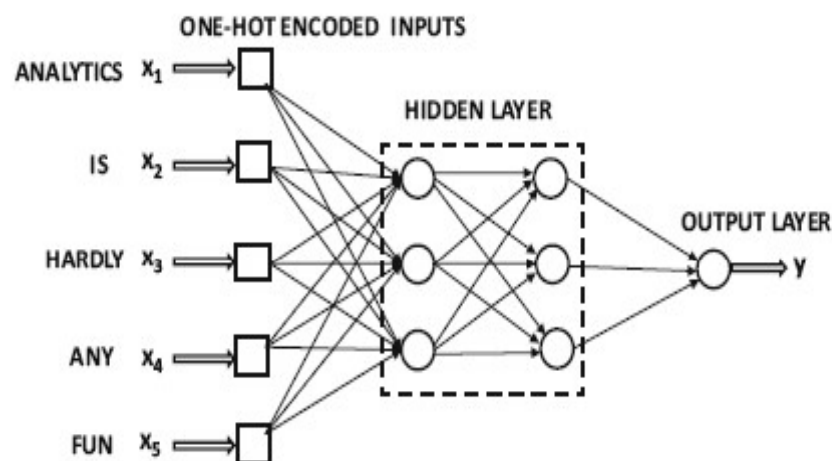
# Applications



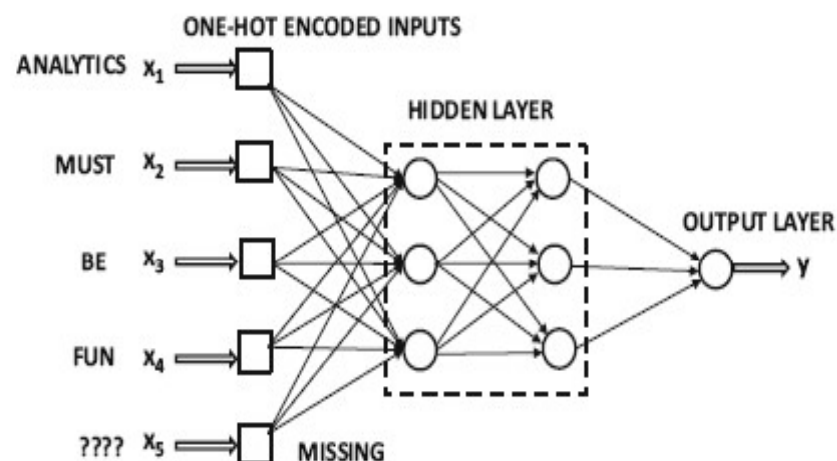
# Real Valued or Symbolic Representation

- The individual values in a sequence can be either **real-valued or symbolic**.
- Real-valued sequences are also referred to as time-series.
- In practical applications, the use of symbolic values is more common.

# Conventional Network for Sentiment Analysis



(a) 5-word sentence  
*"Analytics is hardly any fun."*



(b) 4-word sentence  
*"Analytics must be fun."*

# Challenges using Conventional Network

- An attempt to use a conventional neural network for sentiment analysis faces the challenge of **variable-length inputs**.
- The network architecture also **does not** contain any helpful information about **sequential dependencies** among successive words.



# Challenges using Conventional Network

- Many sequence-centric applications like text are often processed as **bags of words**.
- Such approach **ignores ordering of words** in the document.
- **May work well** for documents of **reasonable size**.
- In applications where the semantic interpretation of the sentence is important, or in which the size of the text segment is relatively small (e.g., a single sentence), such an approach is simply inadequate.

# Challenges using Conventional Network

- Consider the following pair of sentences:
  1. The cat chased the mouse.
  2. The mouse chased the cat.
- The two sentences are **clearly very different** (and the second one is unusual).
- The **bag-of-words** representation would **deem them identical**.
- A greater degree of **linguistic intelligence** is required for more sophisticated applications in difficult settings such as *sentiment analysis, machine translation, or information extraction*.

# How do we resolve this problem?

- *It is important to somehow encode information about the word ordering more directly.*
- *Its good if the architecture of the network supports such encoding.*
- The goal of such an approach would be to reduce the parameter requirements with increasing sequence length.

# Recurrent Neural Network

- Recurrent neural networks provide an excellent example of (parameter-wise) *frugal architectural design*.
- The ability to receive and process inputs in the **same order** as they are present in the sequence.
- The treatment of inputs at each time-stamp in a similar manner **in relation to previous history of inputs**.

# Recurrent Neural Network

- In a recurrent neural network, there is a **one-to-one correspondence** between the **layers** in the network and the **specific positions** in the sequence.
- The position in the sequence is also referred to as its *time-stamp*.
- Therefore, instead of a variable number of inputs in a single input layer, the network contains a variable number of layers.
- Each layer has a single input corresponding to that time-stamp.

# Recurrent Neural Network

- The **inputs are allowed** to directly interact with **down-stream hidden layers** depending on their positions in the sequence.
- **Each layer** uses the **same set of parameters** to ensure similar modeling at each time stamp.
- Therefore the number of parameters is fixed as well.
- The same layer-wise architecture is repeated in time, and therefore the network is **referred to as recurrent**.

# Recurrent Neural Network

To model sequences, we need to:

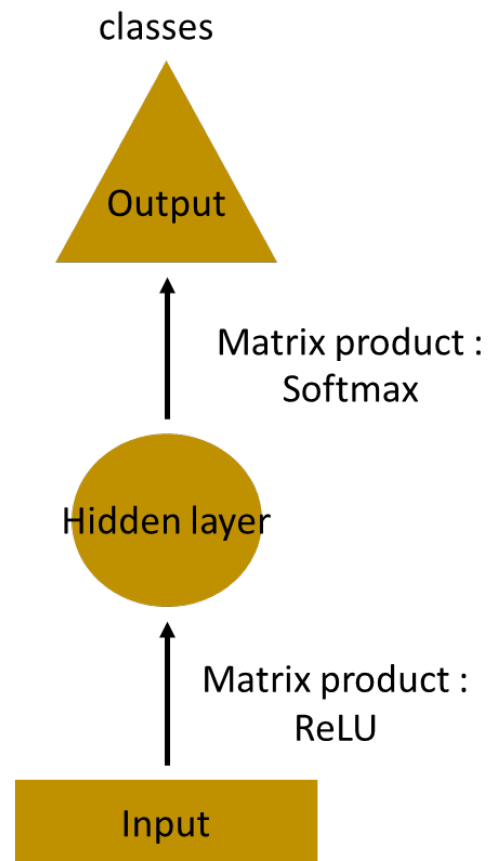
1. Handle variable-length sequences
2. Track long-term dependencies
3. Maintain information about order
4. Share parameters across the sequence

# MLP to RNN

- Let's say the task is to predict the next word in a sentence.
- Let's try accomplishing it using an MLP.
- In the simplest form, we have an input layer, a hidden layer and an output layer.
- The input layer receives the input, the hidden layer activations are applied and then we finally receive the output.



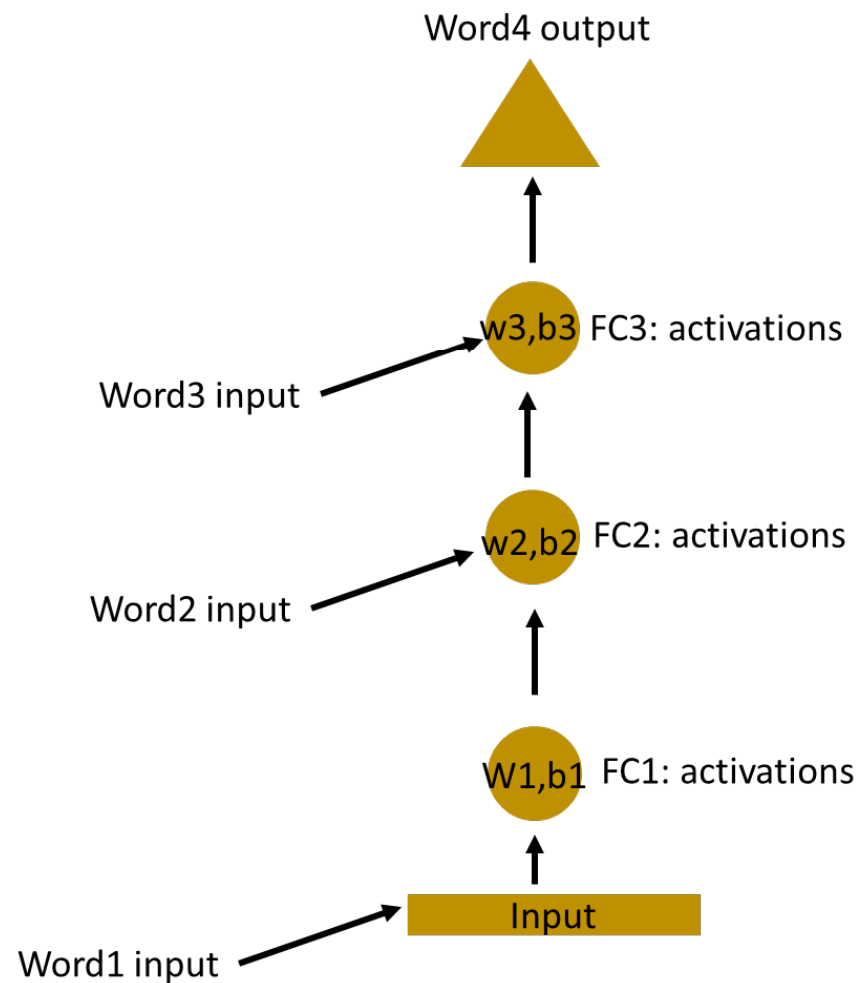
# MLP Approach



# MLP to RNN

- Let's have a deeper network, where multiple hidden layers are present.
- So here, the input layer receives the input and the first hidden layer activations are applied.
- These activations are sent to the next hidden layer, and successive activations through the layers to produce the output.
- Each hidden layer is characterized by its own weights and biases.

# MLP with Deeper Layers



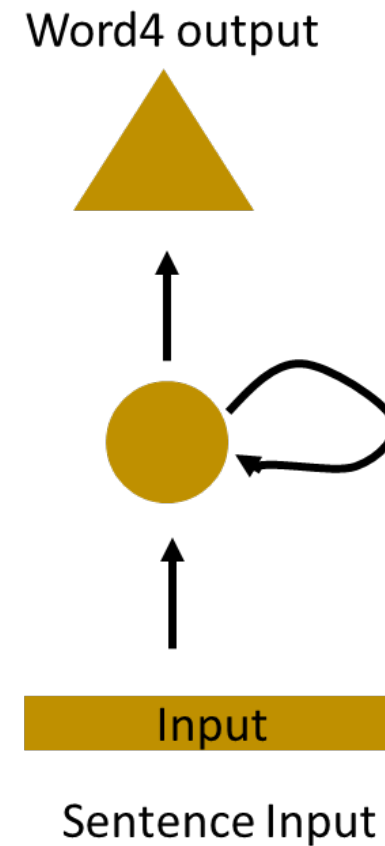
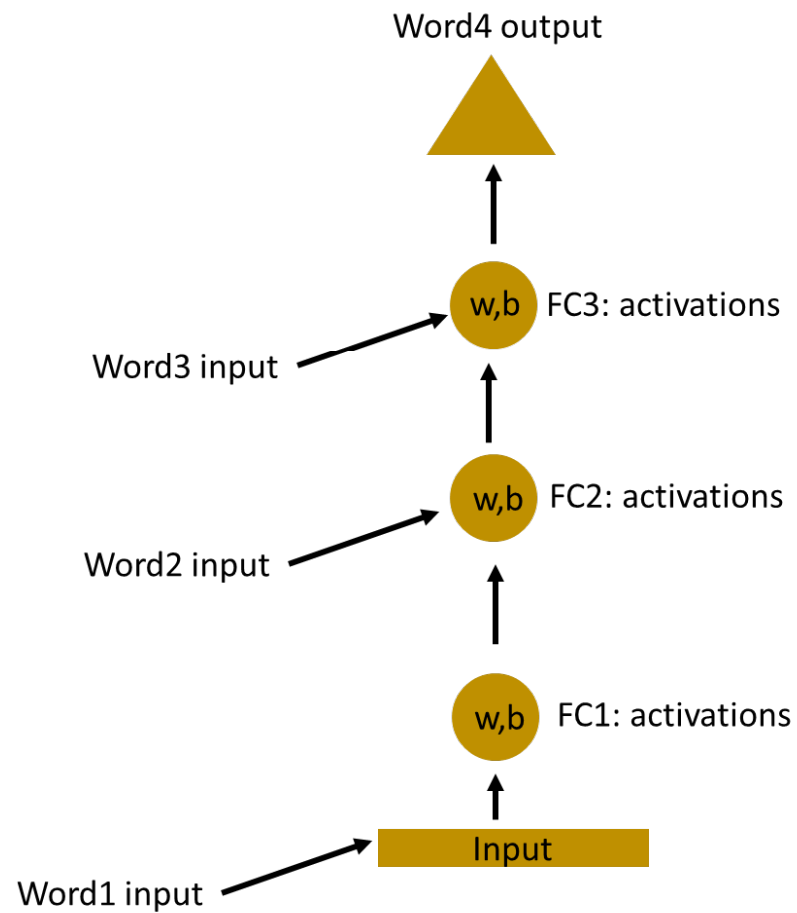
# MLP to RNN

- Since each hidden layer has its own weights and activations, they behave independently.
- Here, the weights and bias of these hidden layers are different.
- Now the objective is to identify the relationship between successive inputs.
- Can we supply the inputs to hidden layers?
- Yes we can!

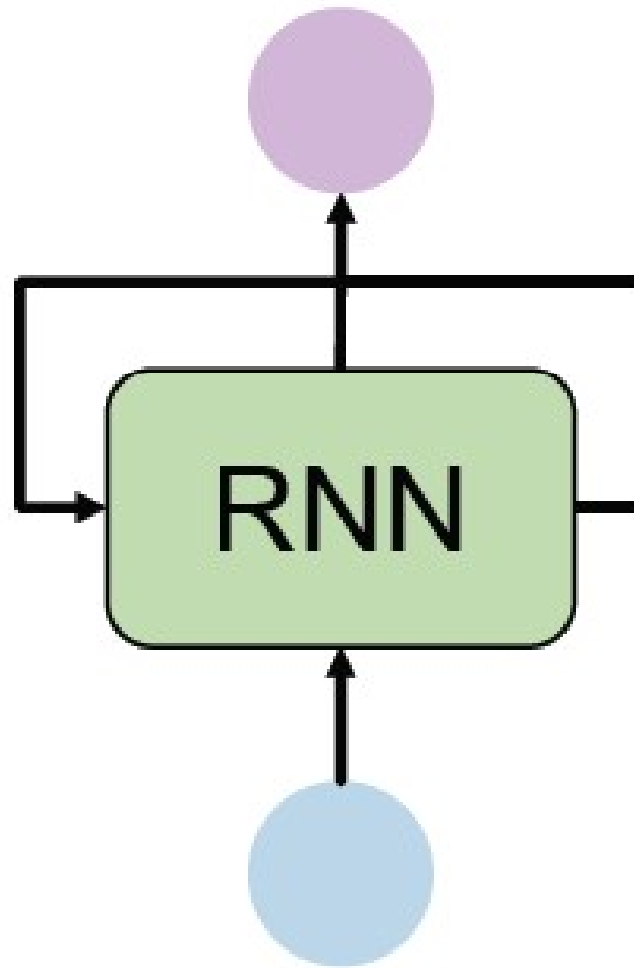
# How?

- Hence each of these layers behave independently and cannot be combined together.
- To combine these hidden layers together, we shall have the same weights and bias for these hidden layers.

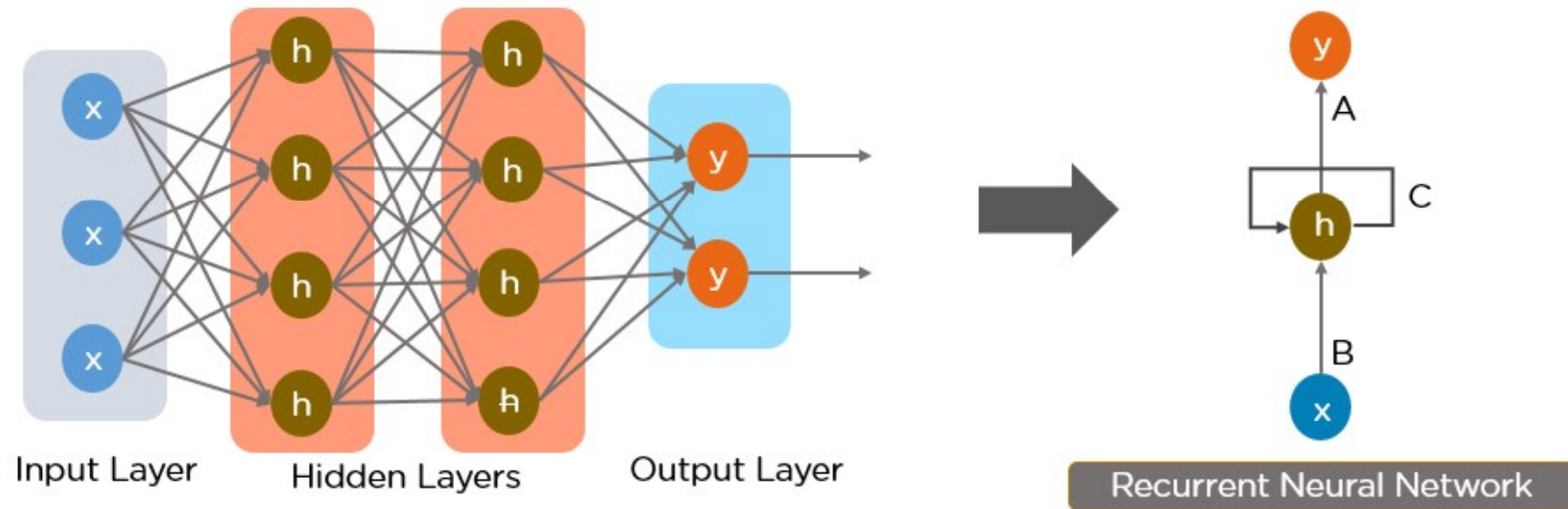
# MLP with Same Weights and Biases



# RNN Architecture

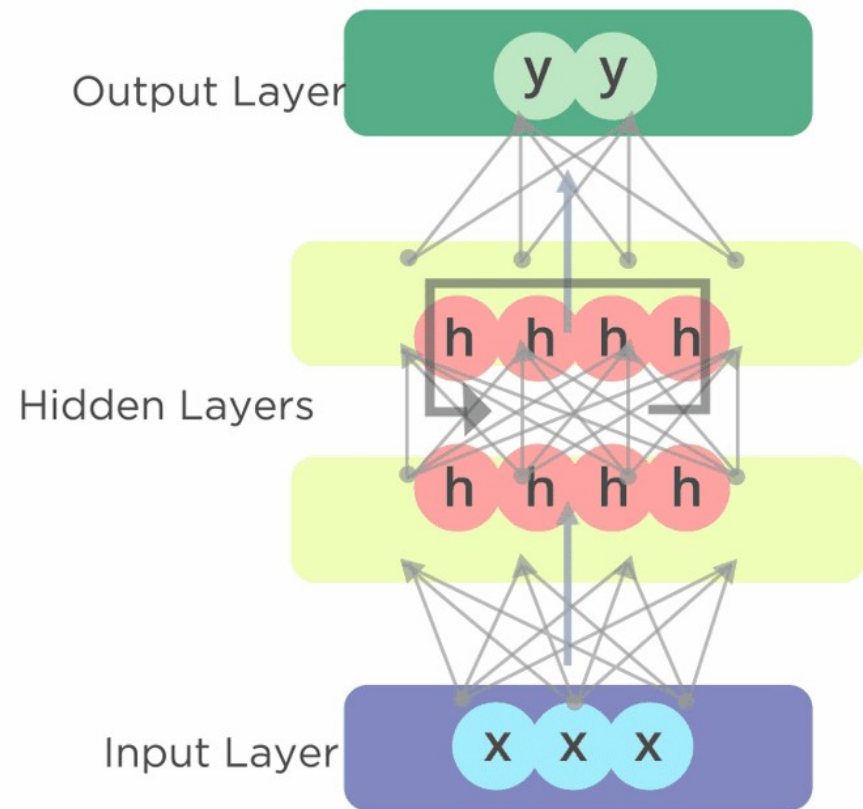


# Recurrent Neural Network





# Network Framework



A, B and C are the parameters

# Network Time Sequence Expansion

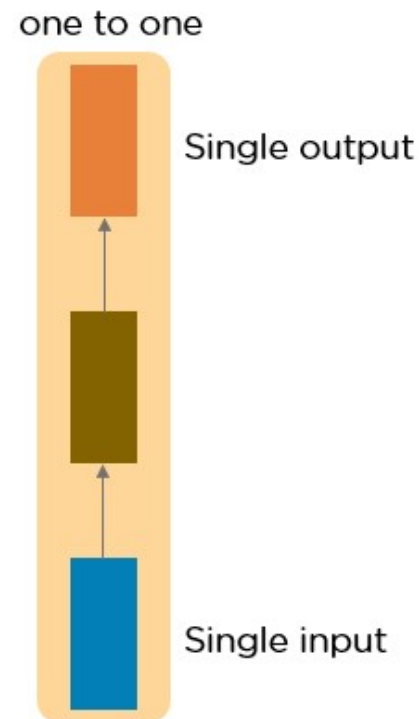
# Types of RNN

There are mainly four types of Recurrent Neural Networks:

- One to One
- One to Many
- Many to One
- Many to Many

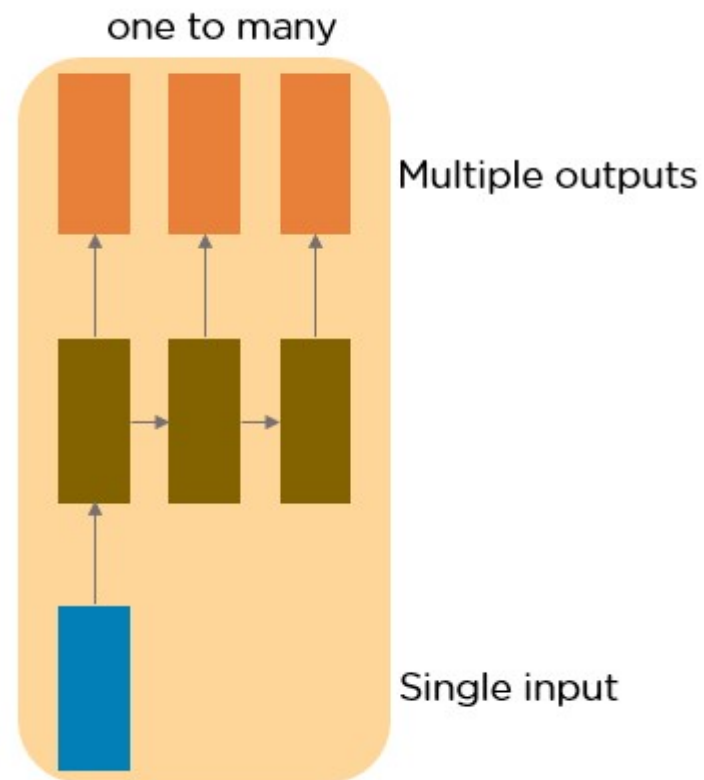
# One to One RNN

- This type of neural network is known as the Vanilla Neural Network.
- It's used for general machine learning problems, which has a single input and a single output.



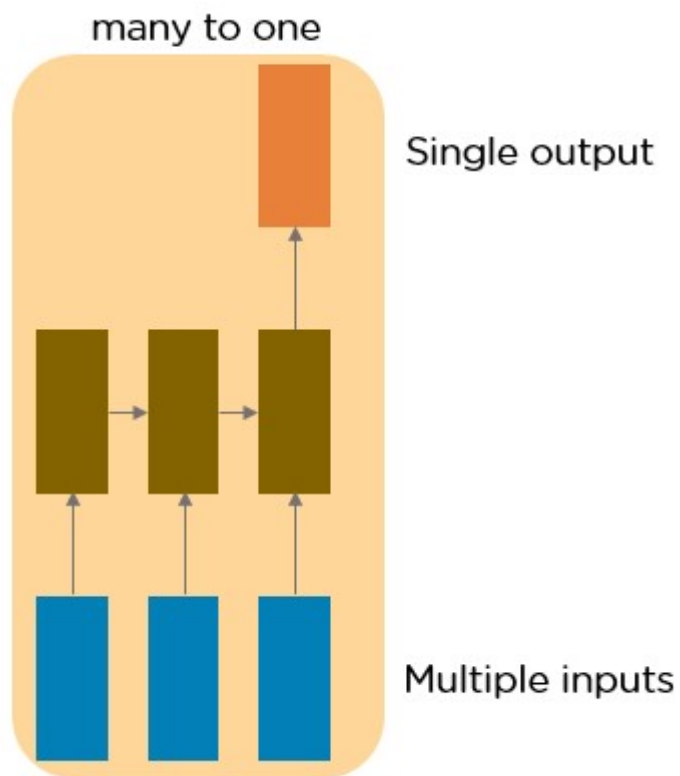
# One to Many RNN

- This type of neural network has a single input and multiple outputs.
- An example of this is the image caption.



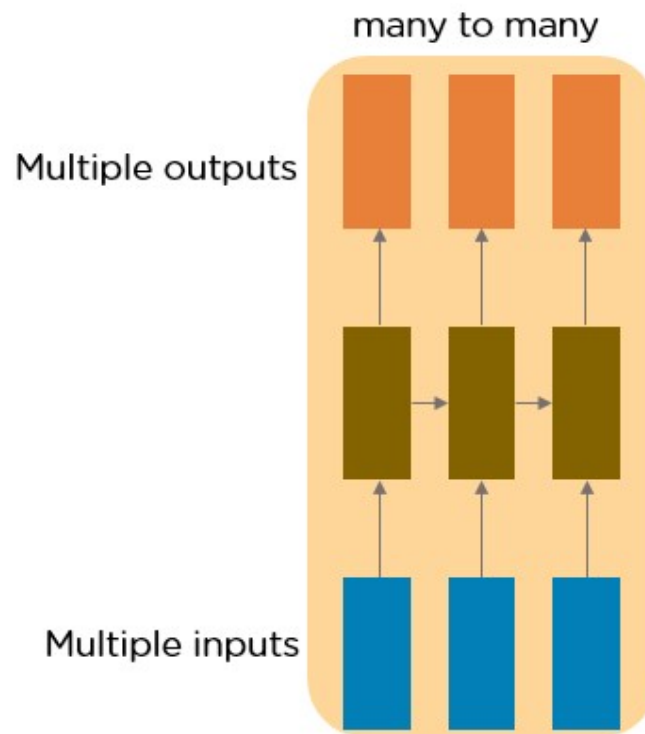
# Many to One RNN

- This RNN takes a sequence of inputs and generates a single output.
- Sentiment analysis (positive or negative).



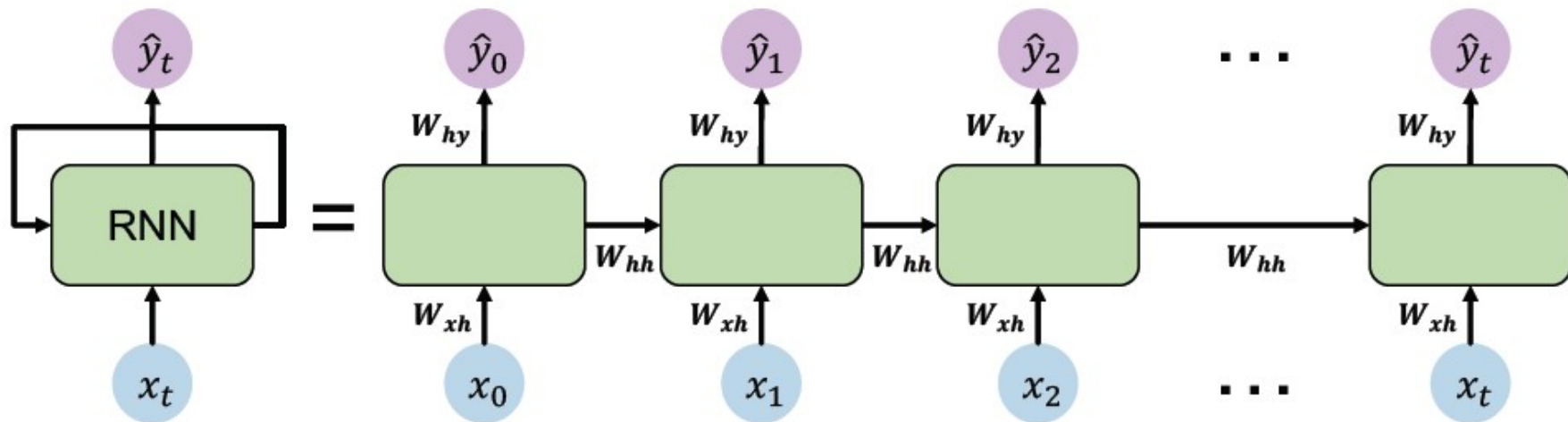
# Many to Many RNN

- This RNN takes a sequence of inputs and generates a sequence of outputs.
- Machine translation is one of the examples.



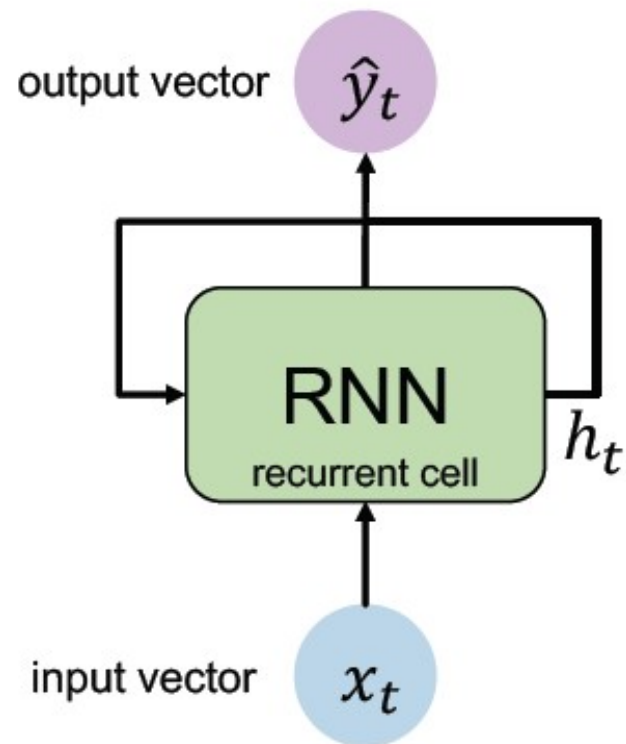
# RNN Weight Matrices

Re-use the **same weight matrices** at every time step





# Recurrent Cell



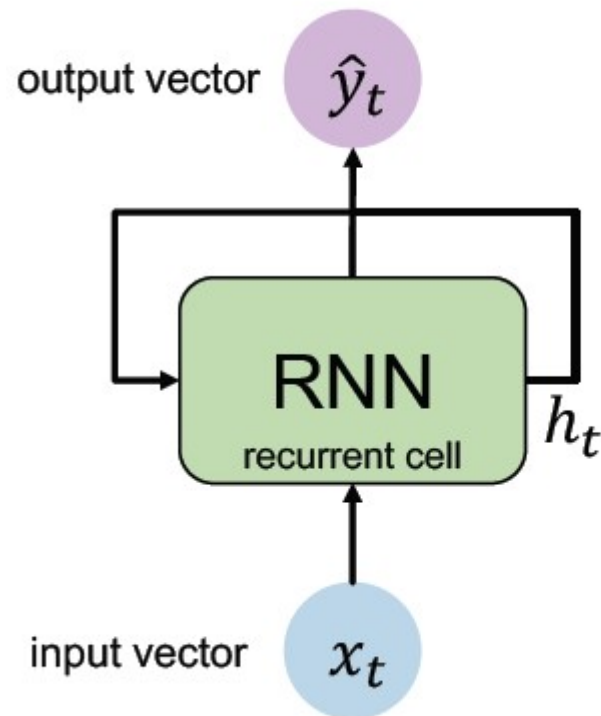
Apply a **recurrence relation** at every time step to process a sequence:

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state      function parameterized by  $W$       old state      input vector at time step  $t$

Note: the same function and set of parameters are used at every time step

# Input, State Update, and Output



Output Vector

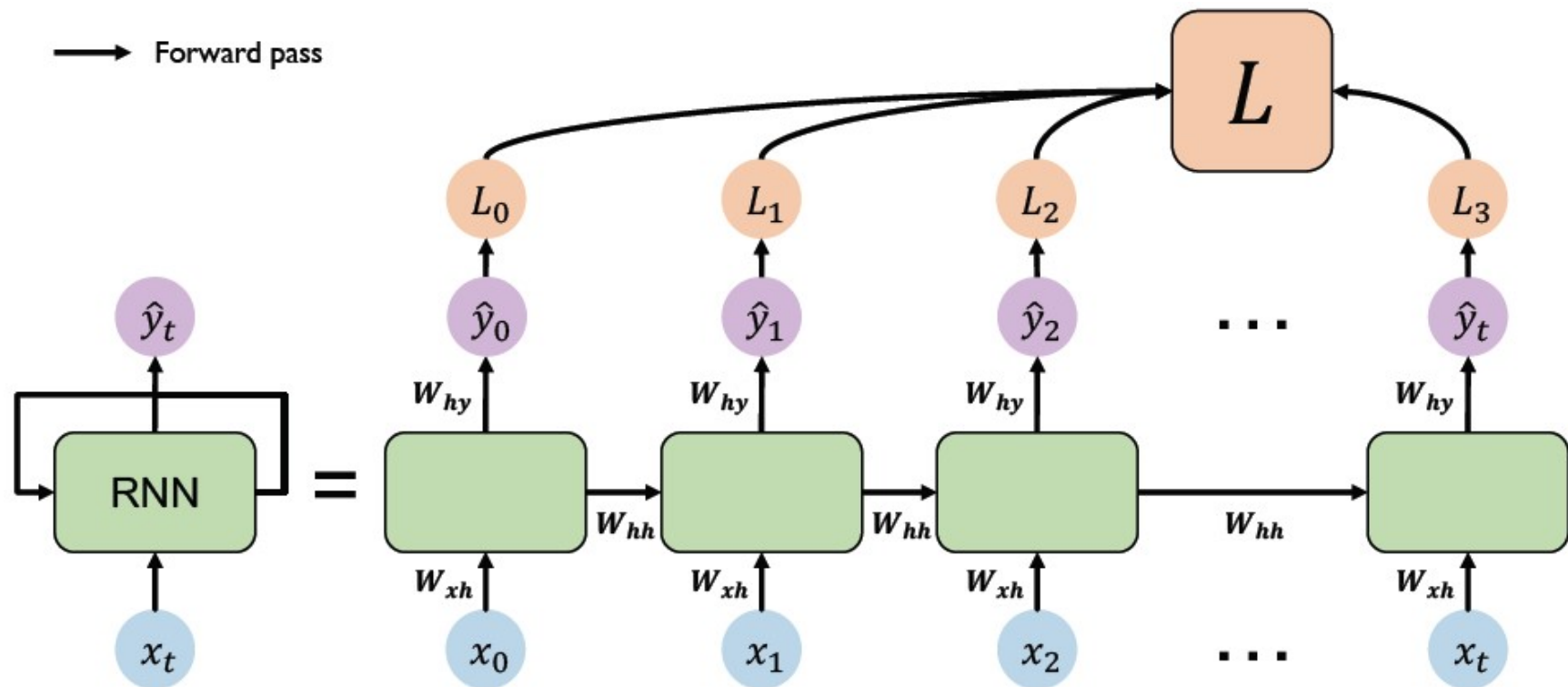
$$\hat{y}_t = \mathbf{W}_{hy}h_t$$

Update Hidden State

$$h_t = \tanh(\mathbf{W}_{hh}h_{t-1} + \mathbf{W}_{xh}x_t)$$

Input Vector

# Forward Propagation



# Forward Propagation

- Let's take a simple task at first.
- Let's take a character level RNN where we have a word "Hello".
- So we provide the first 4 letters i.e. h,e,l,l and ask the network to predict the last letter i.e. 'o'.
- So here the vocabulary of the task is just 4 letters {h,e,l,o}.

# Forward Propagation

- The formula for the current state can be written as

$$h_t = f(h_{t-1}, x_t)$$

- Here,  $h_t$  is the new state,  $h_{t-1}$  is the previous state while  $x_t$  is the current input.

# Forward Propagation

- Taking the simplest form of a recurrent neural network, let's say that the activation function is  $\tanh$ .
- The weight at the recurrent neuron is  $W_{hh}$  and the weight at the input neuron is  $W_{xh}$ , we can write the equation for the state at time  $t$

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

# Forward Propagation

- The Recurrent neuron in this case is just taking the immediate previous state into consideration.
- For longer sequences the equation can involve multiple such states.
- Once the final state is calculated we can go on to produce the output

$$y_t = W_{hy}h_t$$

# Steps in RNN

- A single time step of the input is supplied to the network i.e.  $x_t$  is supplied to the network
- We then calculate its current state using a combination of the current input and the previous state i.e. we calculate  $h_t$
- The current  $h_t$  becomes  $h_{t-1}$  for the next time step
- We can go as many time steps as the problem demands and combine the information from all the previous states
- Once all the time steps are completed the final current state is used to calculate the output  $y_t$
- The output is then compared to the actual output and the error is generated
- The error is then backpropagated to the network to update the weights (we shall go into the details of backpropagation in further sections) and the network is trained.



# Forward Propagation

1	0	0	0
0	1	0	0
0	0	1	1
0	0	0	0
h	e	l	o

- The inputs are one hot encoded.
- Our entire vocabulary is {h,e,l,o} and hence we can easily one hot encode the inputs.

# Weight Matrices


- Now the input neuron would transform the input to the hidden state using the weight  $w_{xh}$ .
- We can randomly initialize the weights as a 3\*4 matrix

wxh			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

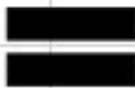
# Step 1

- Now for the letter “h”, for the hidden state we would need  $W_{xh} * X_t$ . By matrix multiplication, we get it as

wxh			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528



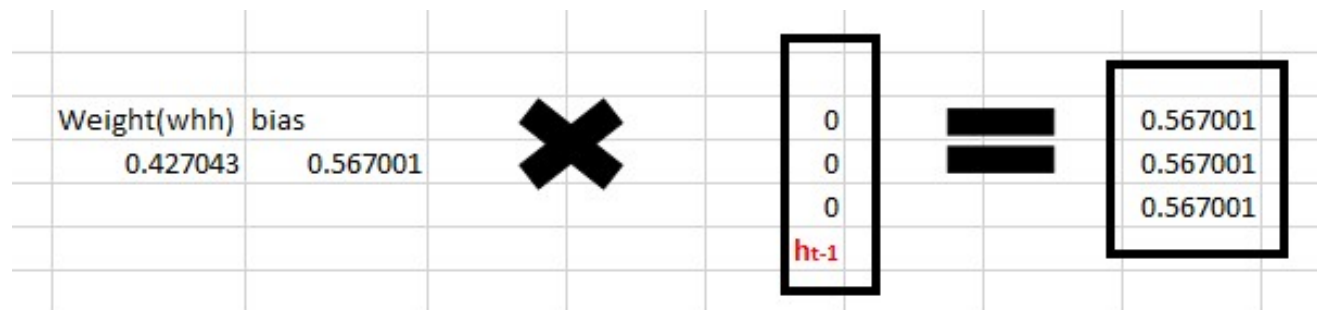
1
0
0
0
h



0.287027
0.902874
0.537524

## Step 2

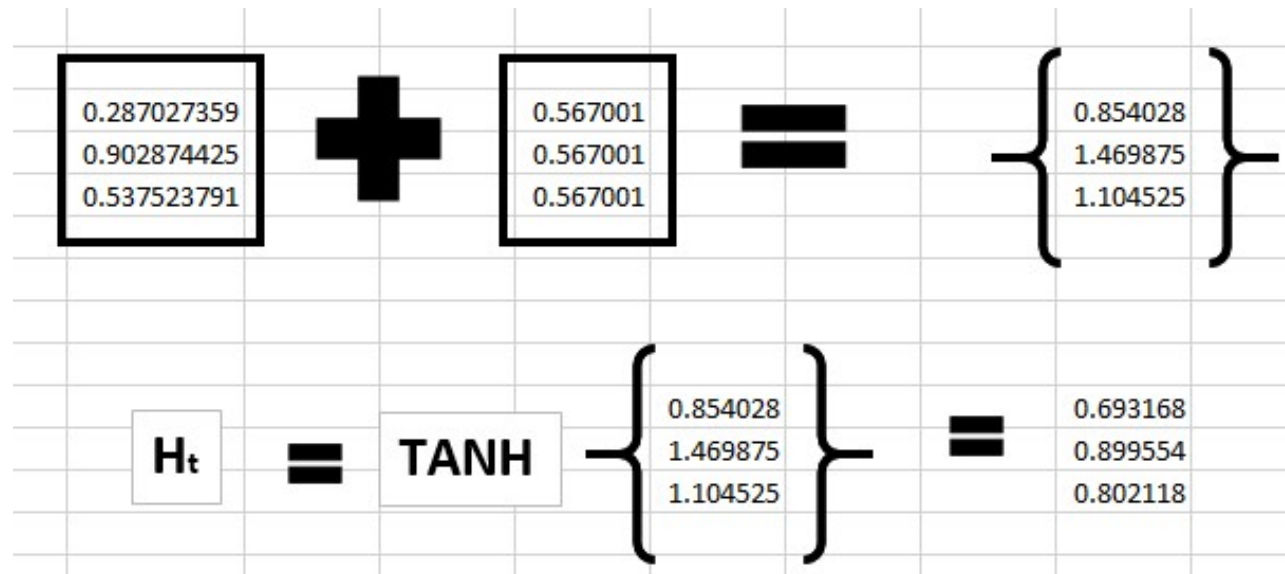
- Now moving to the recurrent neuron, we have  $W_{hh}$  as the weight which is a  $1 \times 1$  matrix and the bias which is also a  $1 \times 1$  matrix
- For the letter “h”, the previous state is  $[0,0,0]$  since there is no letter prior to it. So to calculate  $\rightarrow (w_{hh} * h_{t-1} + \text{bias})$



## Step 3

- Now we can get the current state as

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



## Step 4

- Now we go on to the next state.
- “e” is now supplied to the network.
- The processed output of  $h_{t-1}$ , now becomes  $h_t$ , while the one hot encoded e, is  $x_t$ .
- Let's now calculate the current state  $h_t$

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

# Step 4

- $W_{hh} * h_{t-1} + \text{bias}$  will be

$W_{hh} * h_{t-1} + \text{Bias}$	=	0.427043	×	<table border="1"> <tr><td>0.69316804</td></tr> <tr><td>0.89955366</td></tr> <tr><td>0.8021184</td></tr> </table>	0.69316804	0.89955366	0.8021184	+	0.567001	=	<table border="1"> <tr><td>0.863013</td></tr> <tr><td>0.951149</td></tr> <tr><td>0.90954</td></tr> </table>	0.863013	0.951149	0.90954
0.69316804														
0.89955366														
0.8021184														
0.863013														
0.951149														
0.90954														

- $W_{xh} * x_t$  will be

wxh				×	<table border="1"> <tr><td>0</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> <tr><td>e</td></tr> </table>	0	1	0	0	e	=	<table border="1"> <tr><td>0.84606</td></tr> <tr><td>0.871522</td></tr> <tr><td>0.09224</td></tr> </table>	0.84606	0.871522	0.09224
0															
1															
0															
0															
e															
0.84606															
0.871522															
0.09224															
0.287027359	0.84606	0.572392	0.486813												
0.902874425	0.871522	0.691079	0.18998												
0.537523791	0.09224	0.558159	0.491528												

## Step 5

- Now calculating  $h_t$  for the letter “e”

$H_t$	=	TANH	{	<table border="1"><tr><td>0.863013</td></tr><tr><td>0.951149</td></tr><tr><td>0.90954</td></tr></table>	0.863013	0.951149	0.90954	+	<table border="1"><tr><td>0.84606</td></tr><tr><td>0.871522</td></tr><tr><td>0.09224</td></tr></table>	0.84606	0.871522	0.09224	}	=	<table border="1"><tr><td>0.93653372</td></tr><tr><td>0.94910403</td></tr><tr><td>0.76234056</td></tr></table>	0.93653372	0.94910403	0.76234056
0.863013																		
0.951149																		
0.90954																		
0.84606																		
0.871522																		
0.09224																		
0.93653372																		
0.94910403																		
0.76234056																		

- Now this would become  $h_{t-1}$  for the next state and the recurrent neuron would use this along with the new character to predict the next one.



## Step 6

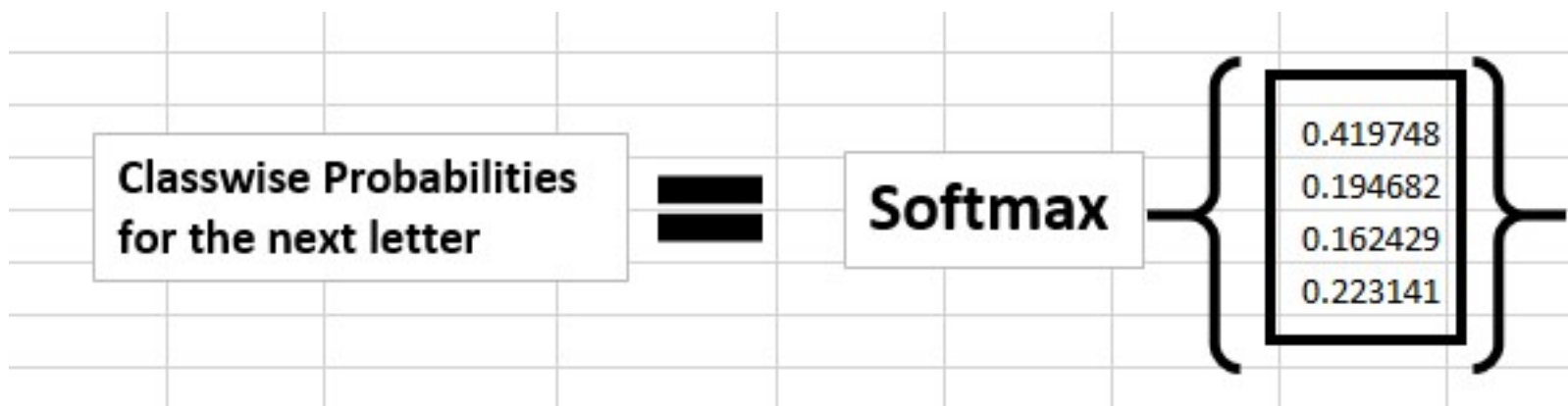
- At each state, the recurrent neural network would produce the output as well.
- Let's calculate  $y_t$  for the letter e.

$$y_t = W_{hy}h_t$$

why			×	Ht	=	yt
0.37168	0.974829459	0.830034886		0.936534		1.90607732
0.39141	0.282585823	0.659835709		0.949104		1.13779113
0.64985	0.09821557	0.334287084		0.762341		0.95666016
0.91266	0.32581642	0.144630018				1.27422602

# Step 7

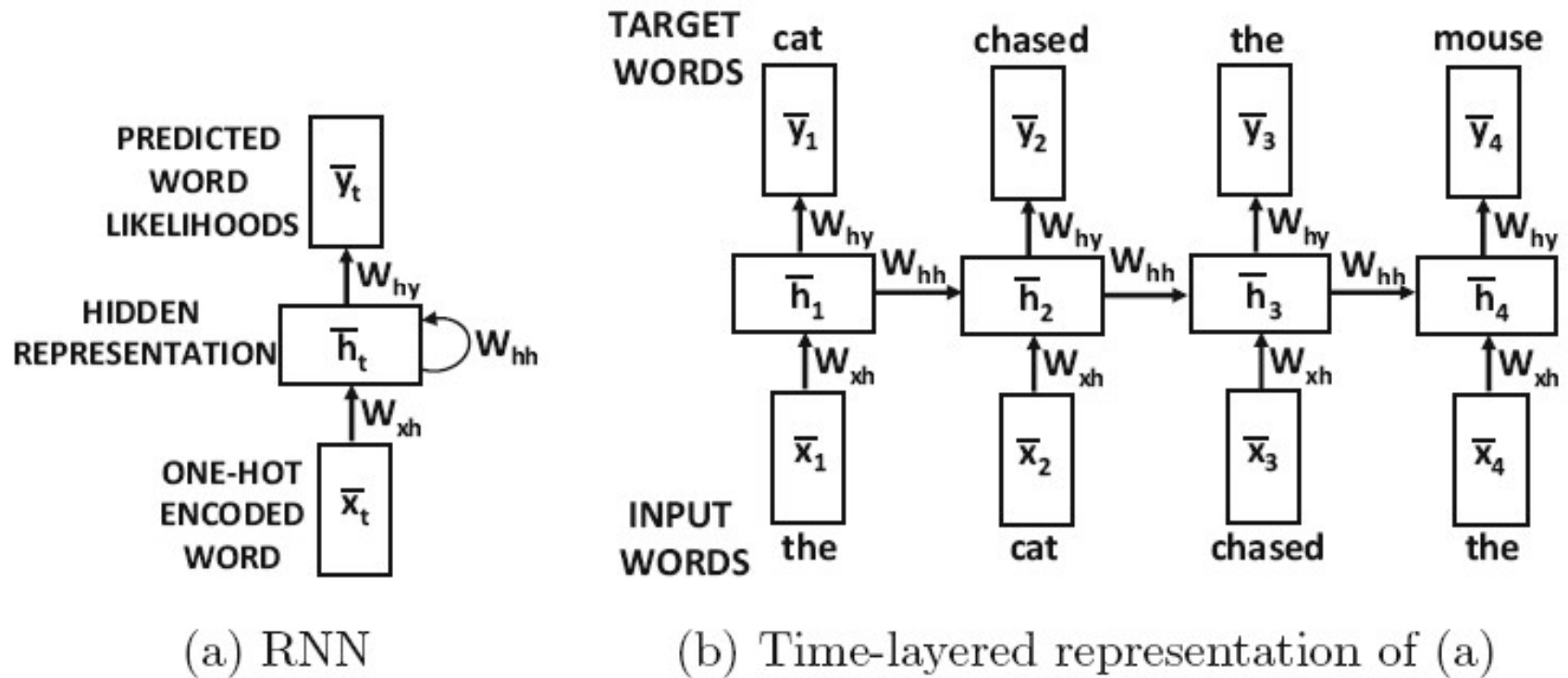
- The probability for a particular letter from the vocabulary can be calculated by applying the softmax function.
- so we shall have  $\text{softmax}(y_t)$



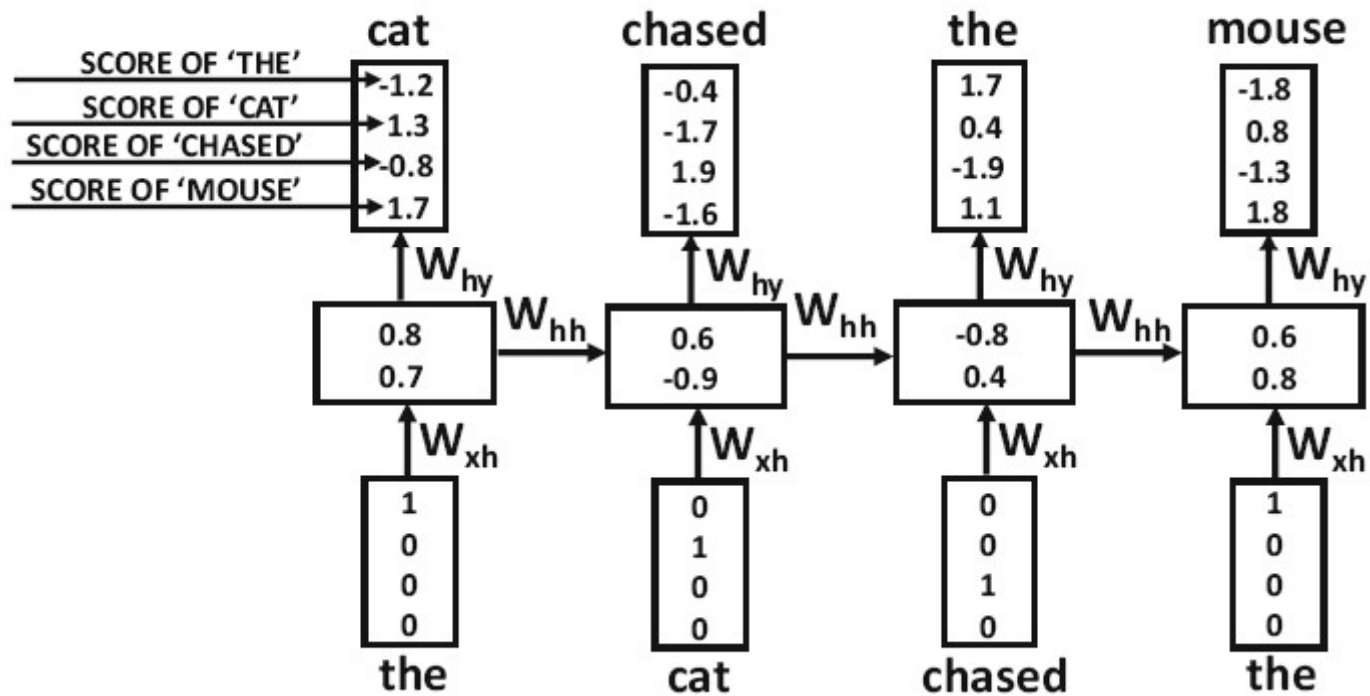
# Output

- If we convert these probabilities to understand the prediction, we see that the model says that the letter after “e” should be h, since the highest probability is for the letter “h”.
- Does this mean we have done something wrong?
- No, so here we have hardly trained the network.
- How do we train? Using Backpropagation.

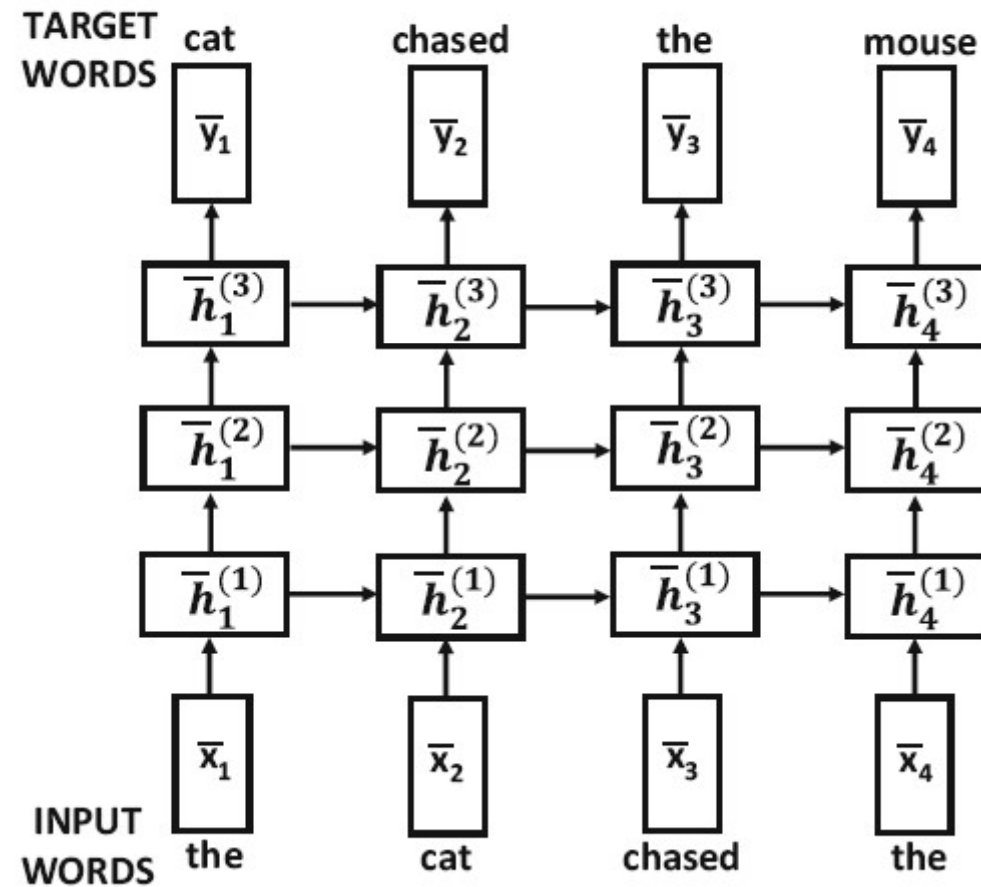
# How to Extend it to Words?



# How to Extend it to Words?



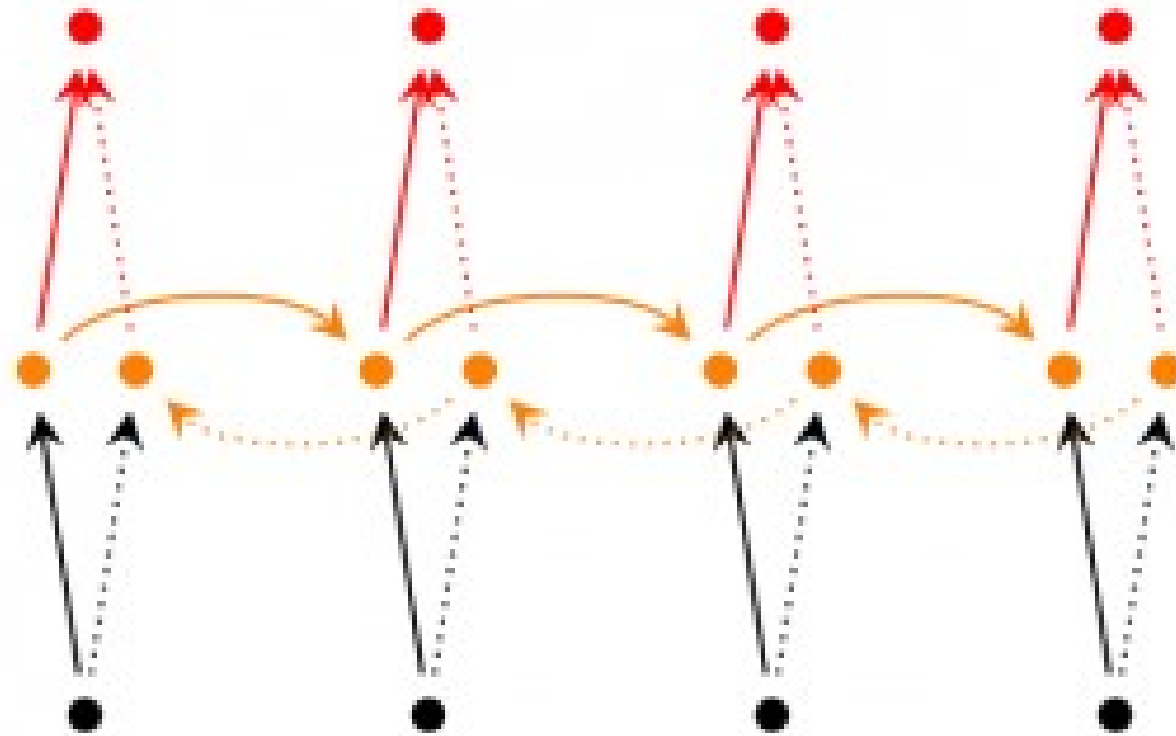
# Multi Layered RNNs



# Extensions of RNNs - Bidirectional

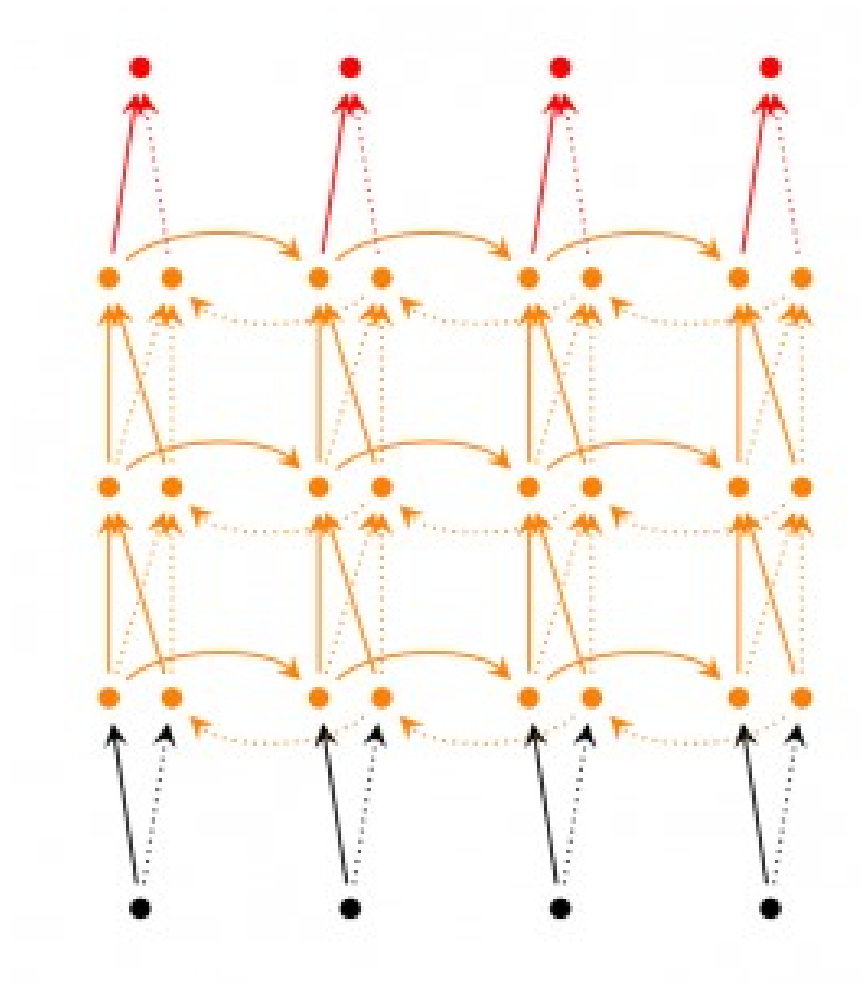
- **Bidirectional RNNs** are based on the idea that the output at time  $t$  may not only depend on the previous elements in the sequence, but also future elements.
- For example, to predict a missing word in a sequence you want to look at both the left and the right context.
- Bidirectional RNNs are quite simple.
- They are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs.

# Bidirectional RNN

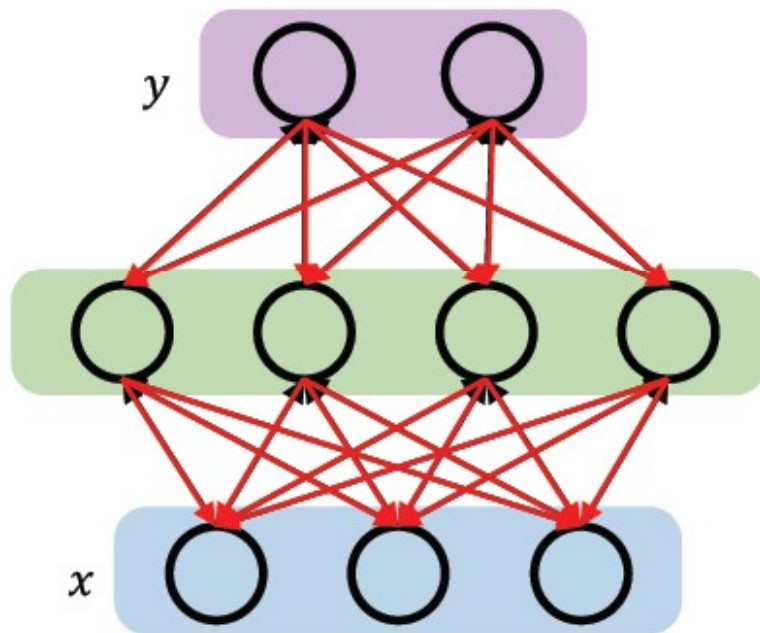




# Deep Bidirectional RNN



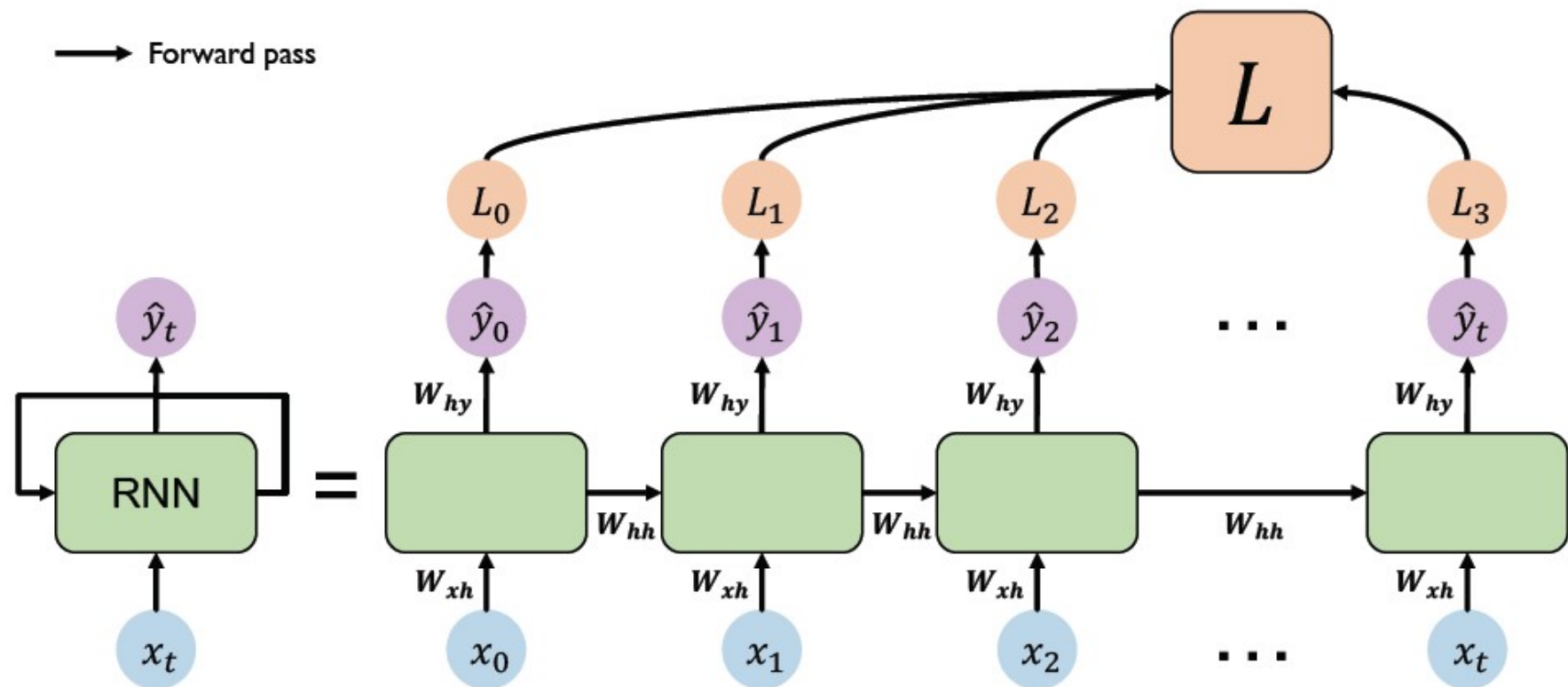
# Backpropagation



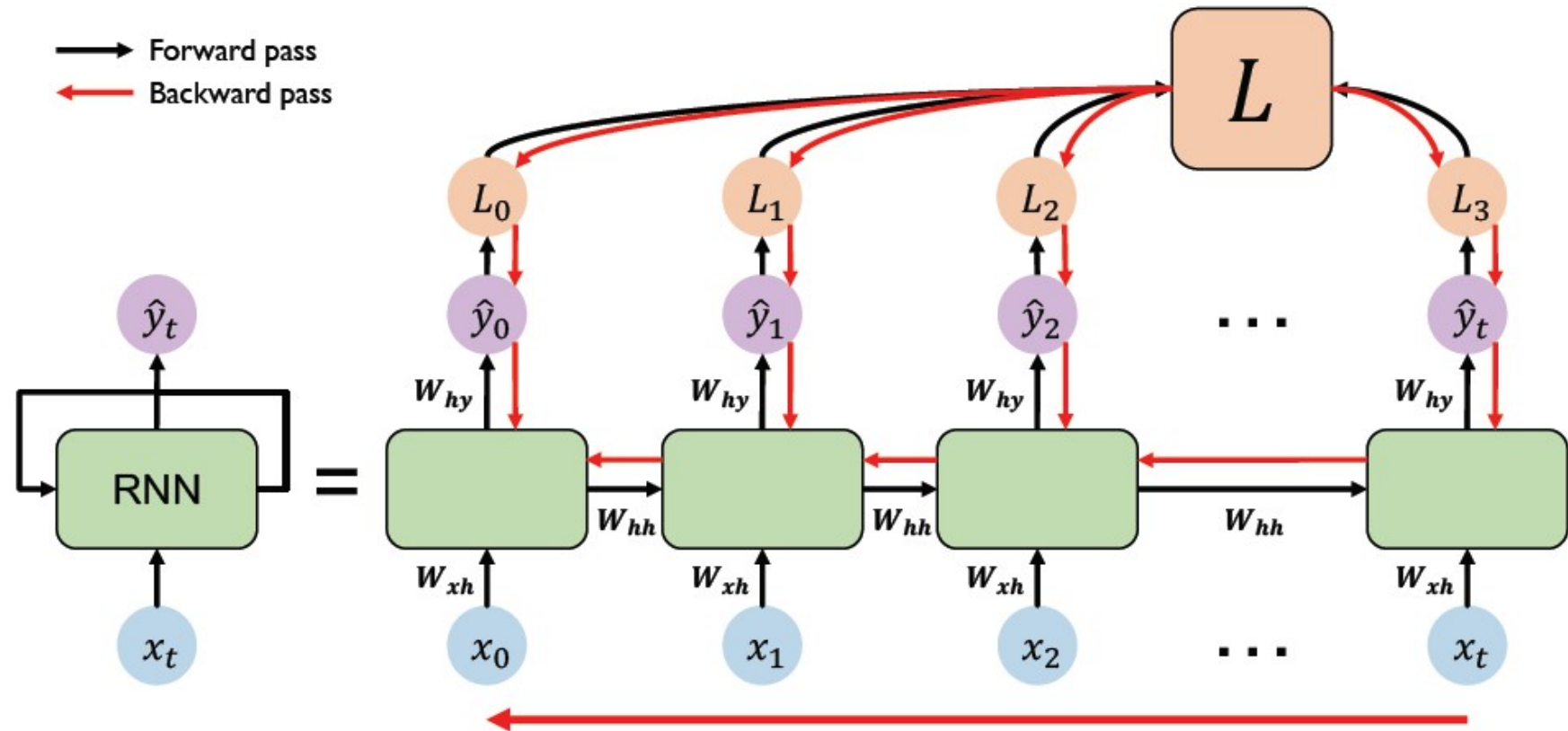
## Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

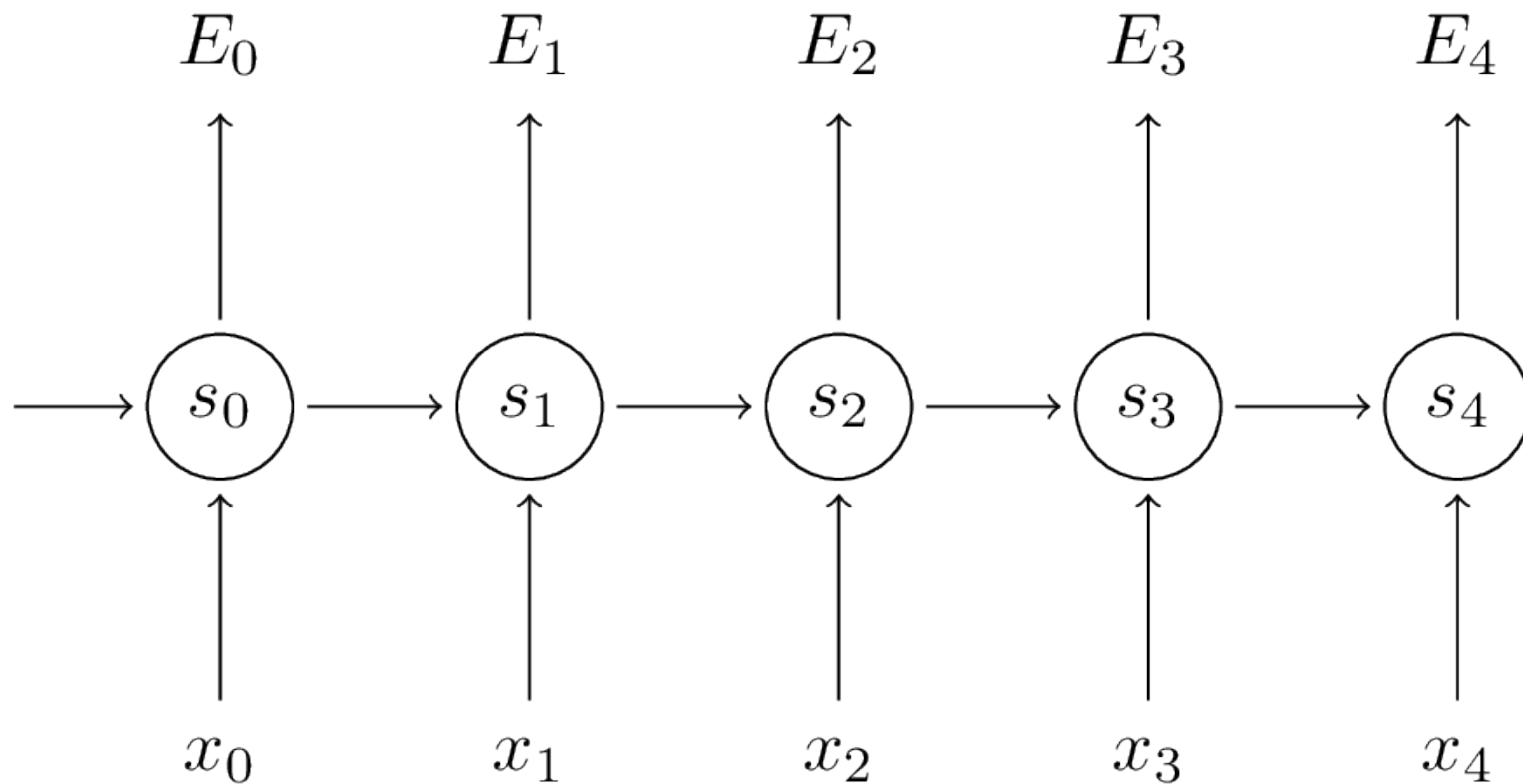
# Forward Pass



# Backward Pass

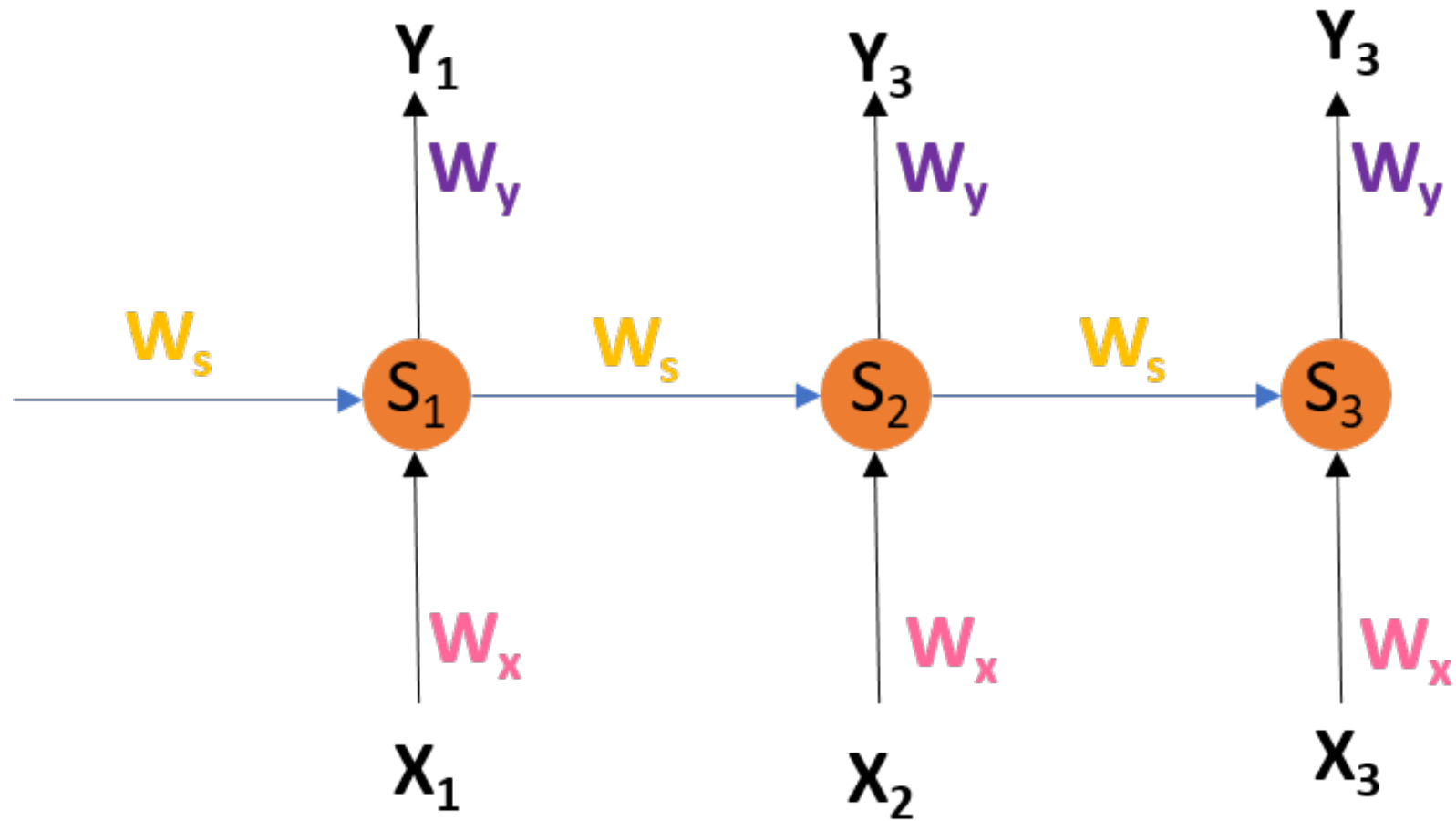


# Backpropagation Through Time (BPTT)



$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

# BPTT



# BPTT

- $S1, S2, S3$  are the hidden states or memory units at time  $t1, t2, t3$  respectively, and  $Ws$  is the weight matrix associated with it.
- $X1, X2, X3$  are the inputs at time  $t1, t2, t3$  respectively, and  $Wx$  is the weight matrix associated with it.
- $Y1, Y2, Y3$  are the outputs at time  $t1, t2, t3$  respectively, and  $Wy$  is the weight matrix associated with it.

# BPTT

- For any time,  $t$ , we have the following two equations:

$$S_t = g_1(W_x x_t + W_s S_{t-1})$$

$$Y_t = g_2(W_Y S_t)$$

- Let us now perform back propagation at time  $t = 3$ . Let the error function be:

$$E_t = (d_t - Y_t)^2$$

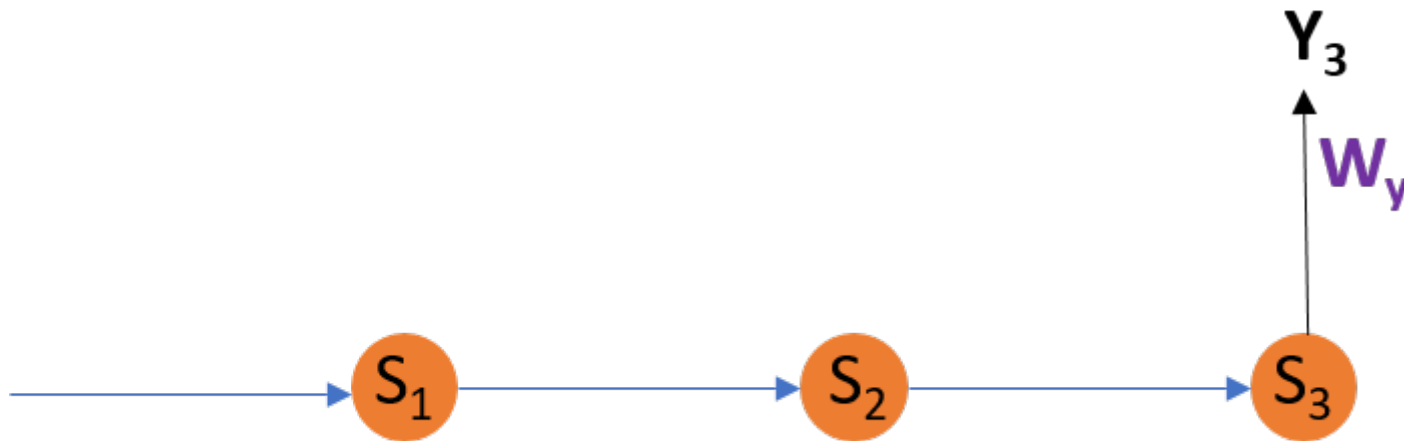
, so at  $t=3$ ,

$$E_3 = (d_3 - Y_3)^2$$



# Adjusting $W_y$

- For better understanding, let us consider the following representation:



# Adjusting Wy

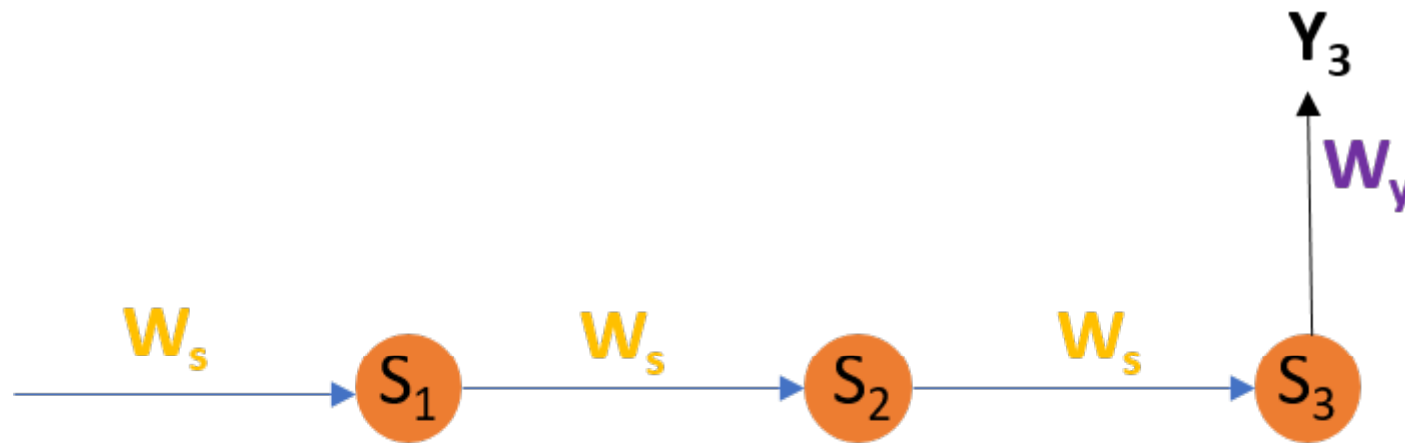
$$\frac{\partial E_3}{\partial W_y} = \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial W_Y}$$

- **Explanation:**

$E_3$  is a function of  $Y_3$ . Hence, we differentiate  $E_3$  w.r.t.  $Y_3$ .

$Y_3$  is a function of  $WY$ . Hence, we differentiate  $Y_3$  w.r.t.  $WY$ .

# Adjusting $W_s$



$$\begin{aligned} \frac{\partial E_3}{\partial W_S} = & \left( \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial W_S} \right) + \\ & \left( \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial W_S} \right) + \\ & \left( \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial S_1} \cdot \frac{\partial S_1}{\partial W_S} \right) \end{aligned}$$

# BPTT

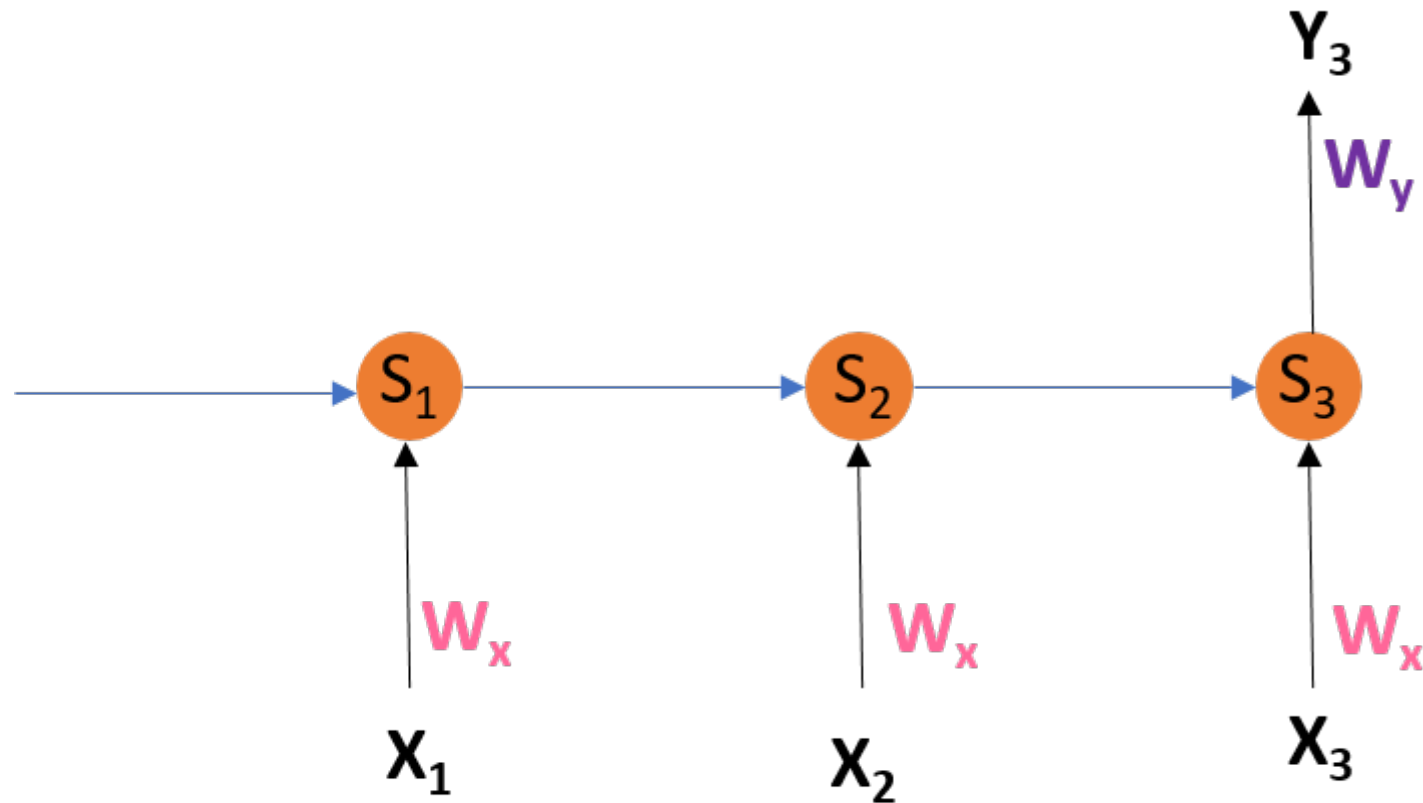
- But we can't stop with this; we also have to take into consideration, the previous time steps.
- So, we differentiate (partially) the Error function with respect to memory units  $S_2$  as well as  $S_1$  taking into consideration the weight matrix  $WS$ .
- We have to keep in mind that a memory unit, say  $S_t$  is a function of its previous memory unit  $S_{t-1}$ .

# BPTT

- Generally, we can express this as

$$\frac{\partial E_N}{\partial W_S} = \sum_{i=1}^N \frac{\partial E_N}{\partial Y_N} \cdot \frac{\partial Y_N}{\partial S_i} \cdot \frac{\partial S_i}{\partial W_S}$$

# Adjusting $WX$



# Adjusting WX

$$\begin{aligned}\frac{\partial E_3}{\partial W_X} = & \left( \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial W_X} \right) + \\ & \left( \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial W_X} \right) + \\ & \left( \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial S_1} \cdot \frac{\partial S_1}{\partial W_X} \right)\end{aligned}$$

$$\frac{\partial E_N}{\partial W_S} = \sum_{i=1}^N \frac{\partial E_N}{\partial Y_N} \cdot \frac{\partial Y_N}{\partial S_i} \cdot \frac{\partial S_i}{\partial W_X}$$

# Limitations

- This method of Back Propagation through time (BPTT) can be used up to a limited number of time steps.
- If we back propagate further, the gradient delta becomes too small.
- This problem is called the “**Vanishing gradient**” problem.
- The problem is that the contribution of information decays geometrically over time.

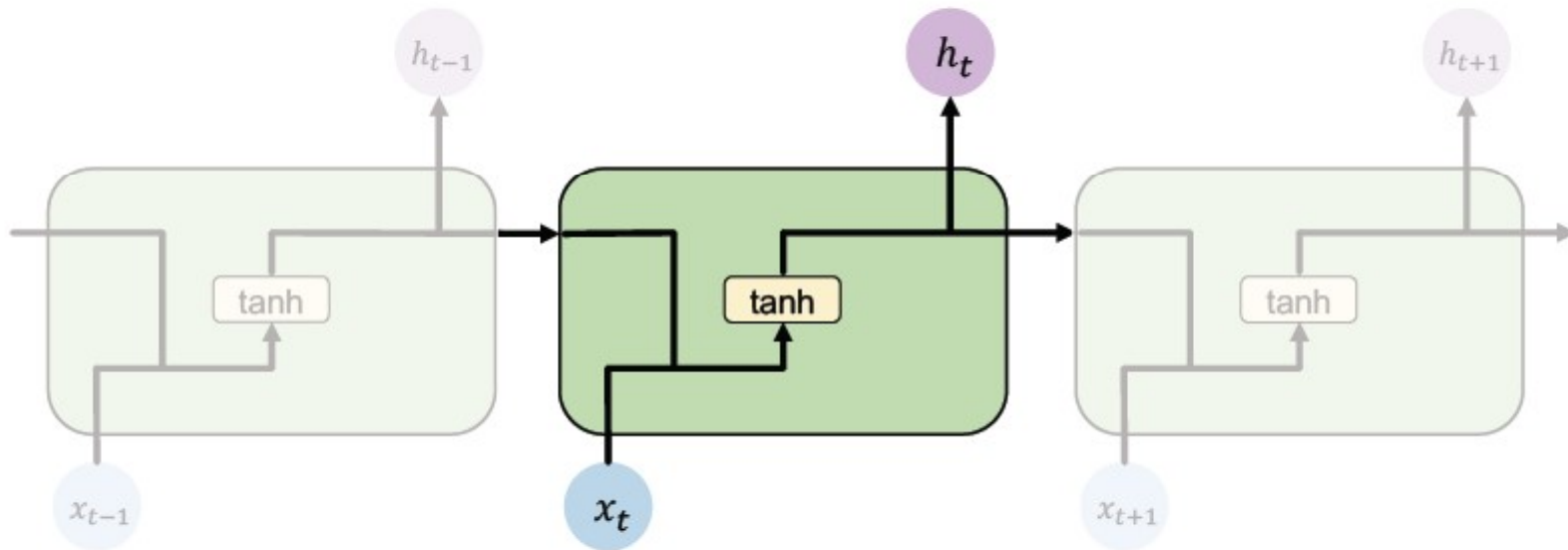


# Beyond RNNs

- One of the famous solutions to this problem is by using what is called **Long Short-Term Memory (LSTM for short)** cells instead of the traditional RNN cells.
- But there might arise yet another problem here, called the **exploding gradient** problem, where the gradient grows uncontrollably large.
- A popular method called **gradient clipping** can be used where in each time step, we can check if the gradient  $\Delta > \text{threshold}$ . If yes, then normalize it.

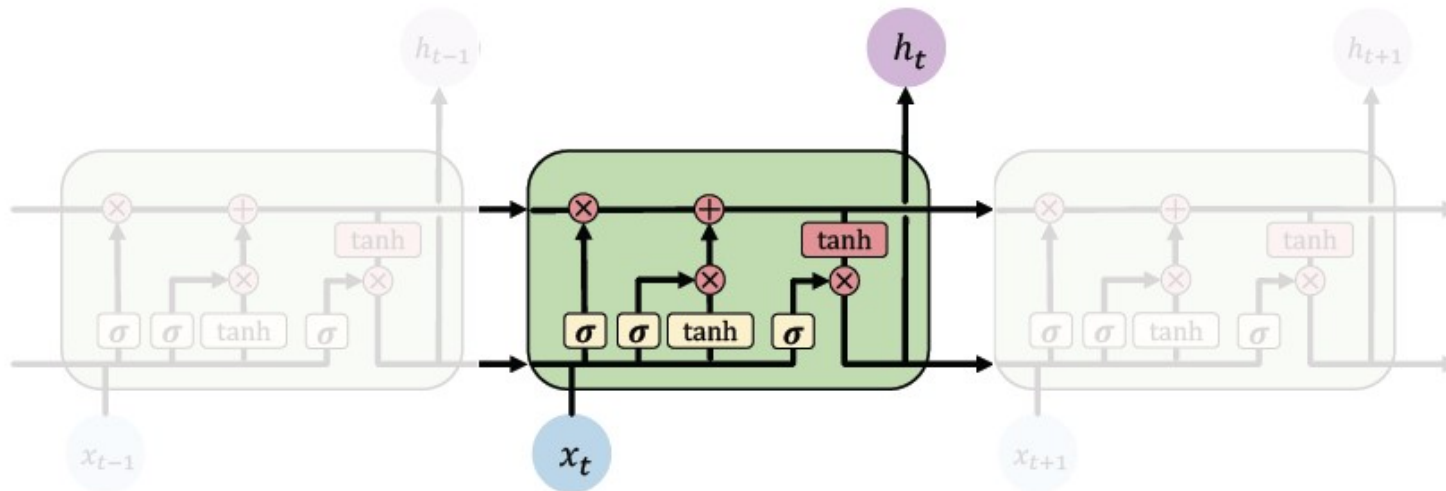
# Standard RNN

In a standard RNN, repeating modules contain a **simple computation node**



# LSTM

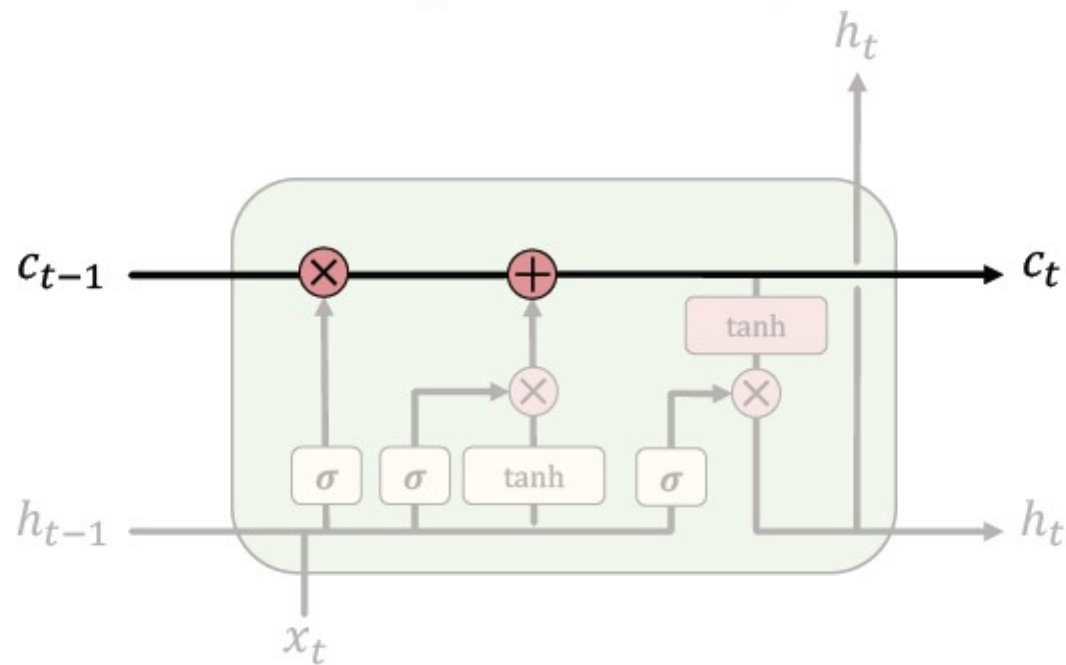
LSTM repeating modules contain interacting layers that **control information flow**



LSTM cells are able to track information throughout many timesteps

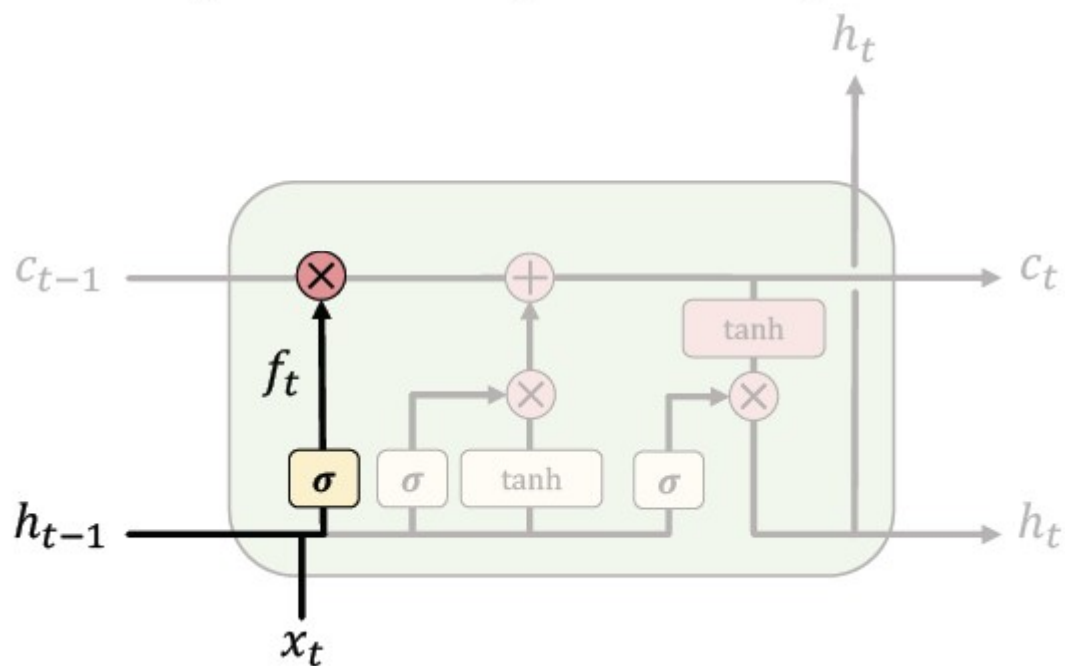
# LSTM

LSTMs maintain a **cell state**  $c_t$  where it's easy for information to flow



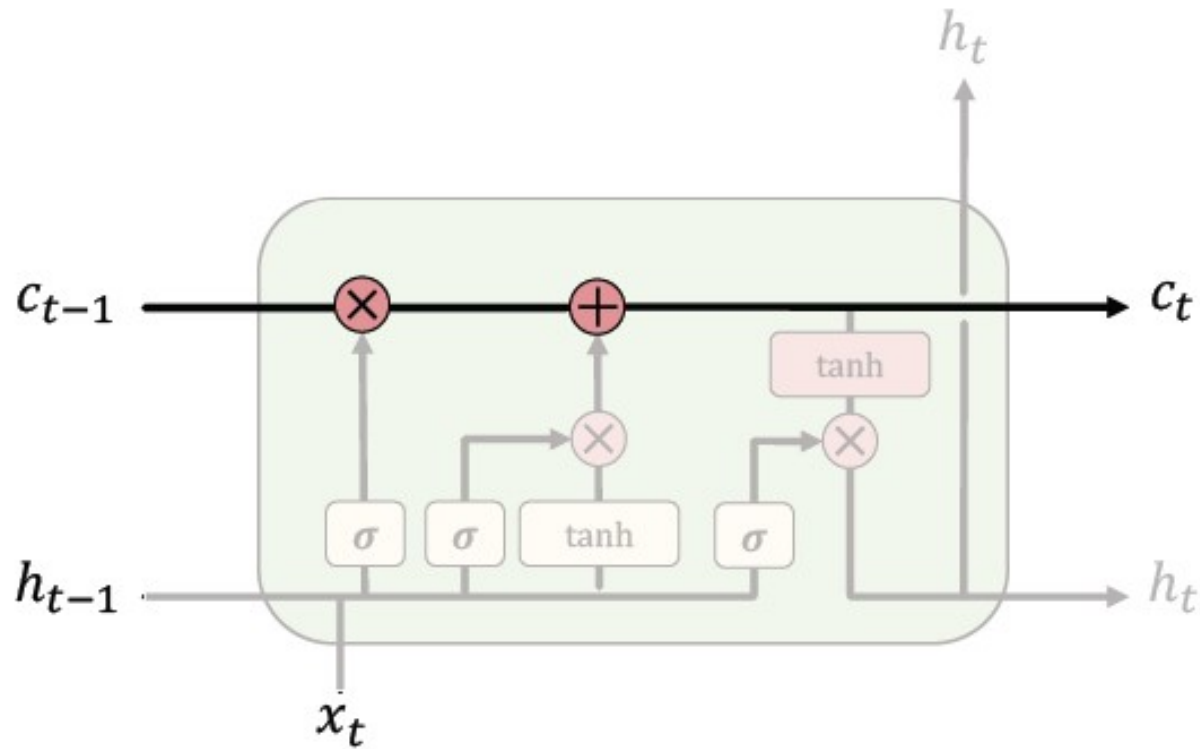
# LSTM

LSTMs **forget** irrelevant parts of the previous state



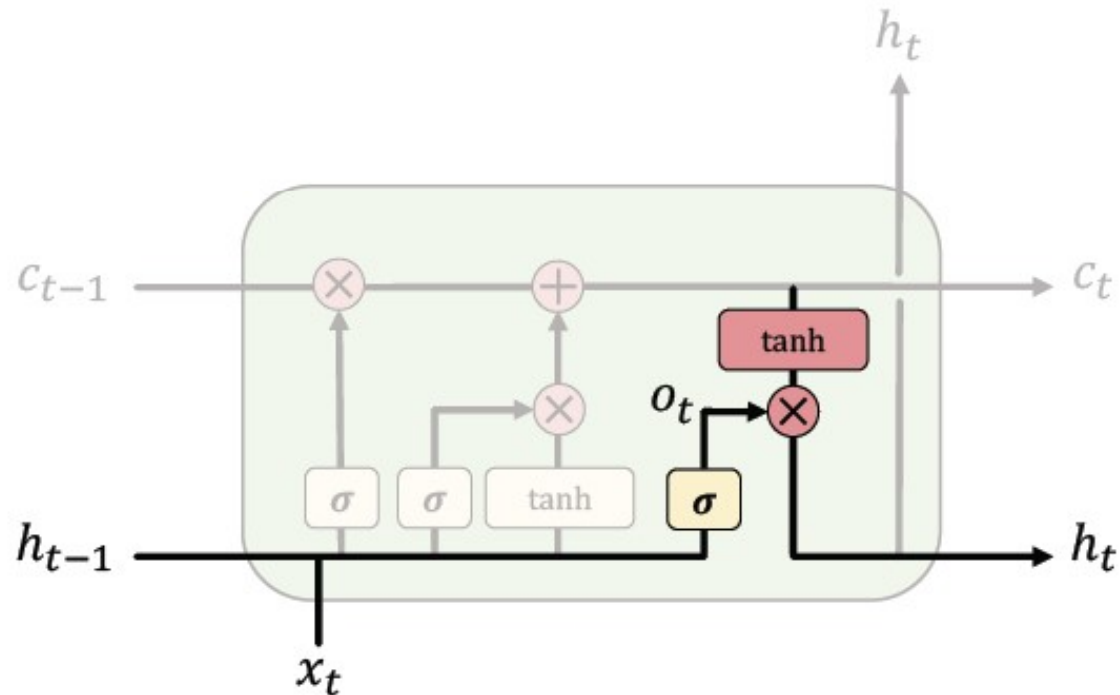
# LSTM

LSTMs **selectively** update cell state values



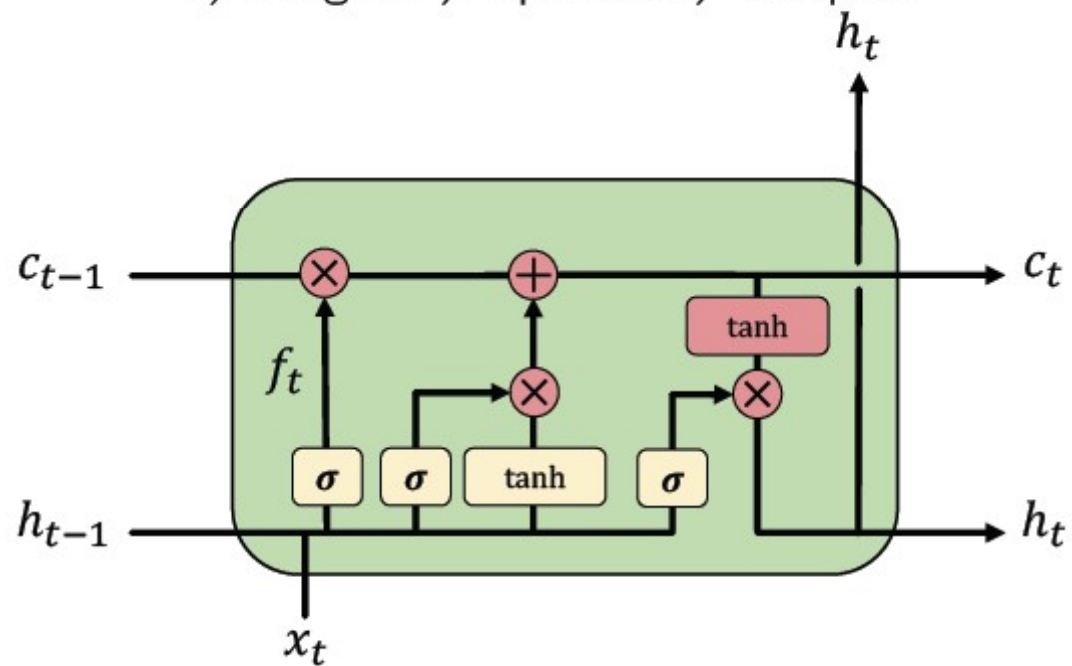
# LSTM

LSTMs use an **output gate** to output certain parts of the cell state



# LSTM

1) Forget 2) Update 3) Output





# Drawbacks of RNN and LSTM

- Recurrent neural network architectures have been shown to efficiently model **long term temporal dependencies between acoustic events**.
- However, the **training time** of recurrent networks is higher than feedforward networks **due to the sequential nature of the learning algorithm**.
- **Due to recurrent connections** in the network, **parallelization during training cannot be exploited** to the same extent as in feed-forward neural networks.

# Alternative Model and Desirable Properties Specific to Speech

- Why FFNN is not suitable to learn speech?
- The inability of most neural network architectures to deal properly with **dynamic nature of speech**.
- **Two important aspects** of this are for a network to **represent temporal relationships between acoustic events**.
- At the same time providing for **invariance under translation in time**.

# Alternative Model and Desirable Properties Specific to Speech

- The specific movement of a formant in time, for example is an important cue to determining the identity of a voiced stop.
- But **it is irrelevant** whether the same set of events occurs a **little sooner or later** in the course of time.
- Without translation invariance, a neural network requires precise segmentation to align the input pattern properly.

# Time Delay Neural Network (TDNN)

328

IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL. 37, NO. 3, MARCH 1989

## Phoneme Recognition Using Time-Delay Neural Networks

ALEXANDER WAIBEL, MEMBER, IEEE, TOSHIYUKI HANAZAWA, GEOFFREY HINTON,  
KIYOHIO SHIKANO, MEMBER, IEEE, AND KEVIN J. LANG

**Abstract**—In this paper we present a Time-Delay Neural Network (TDNN) approach to phoneme recognition which is characterized by two important properties. 1) Using a 3 layer arrangement of simple computing units, a hierarchy can be constructed that allows for the formation of arbitrary nonlinear decision surfaces. The TDNN learns these decision surfaces automatically using error backpropagation [1]. 2) The time-delay arrangement enables the network to discover acoustic-phonetic features and the temporal relationships between them independent of position in time and hence not blurred by temporal shifts

of their “brain-like” appeal<sup>1</sup> but because they offer ways for automatically designing systems that can make use of multiple interacting constraints. In general, such constraints are too complex to be easily programmed and require the use of automatic learning strategies. Such learning algorithms now exist (for an excellent review, see Lippman [2]) and have been demonstrated to discover interesting internal abstractions in their attempts to solve a

# TDNN

## **Two important properties**

1. Using a 3 layer arrangement of simple computing units, a hierarchy can be constructed that allows for the formation of arbitrary nonlinear decision surfaces. The TDNN learns these decision surfaces automatically using error backpropagation.
2. The time-delay arrangement enables the network to discover acoustic-phonetic features and the temporal relationships between them independent of position in time and hence not blurred by temporal shifts in the input.

# TDNN

- The speaker dependent recognition of the phonemes “B”, “D”, and “G” in varying phonetic contexts was chosen.
- HMM were trained to perform the same task.
- TDNN achieves a recognition accuracy of 98.5% compared to 93.7% by HMM.
- Closer inspection reveals that the network “invented” well known acoustic-phonetic features (e.g., F2-rise, F2-fall, vowel onset) as useful abstractions.
- It also developed alternate internal representations to link different acoustic realizations to the same concept.

# TDNN

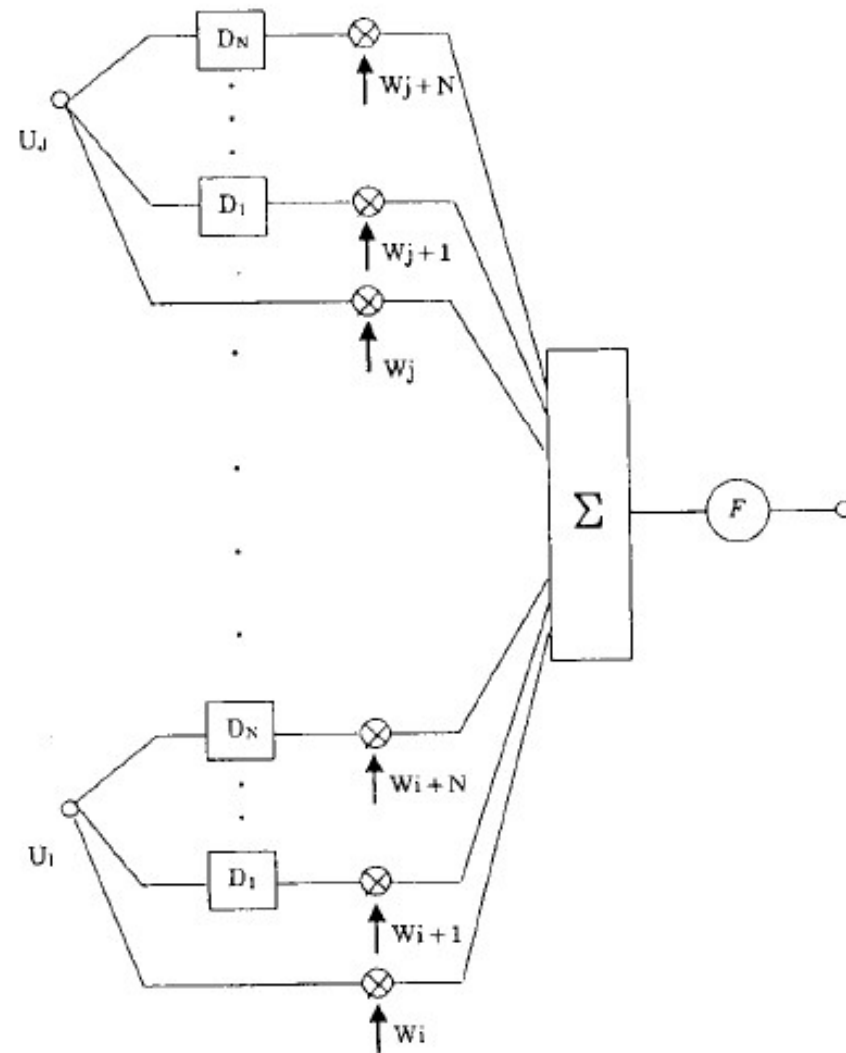
- To be useful for speech recognition, a layered feedforward neural network must have a number of properties.
  1. Should have multiple layers and sufficient interconnections between units in each of these layers. (ability to learn complex nonlinear decision surfaces)
  2. Should have the ability to represent relationships between events in time. These events could be spectral coefficients or any higher level features.

# TDNN

3. Actual features or abstractions learned by the network should be invariant under translation in time.
4. Learning procedure should not require precise temporal alignment of the labels that are to be learned.
5. The number of weights in the network should be sufficiently small compared to the amount of training data so that the network is forced to encode the training data by extracting regularity.



# TDNN



# TDNN

- Basic unit computes the weighted sum of its inputs and then passes this sum through a nonlinear function (sigmoid or RELU).
- The basic unit is modified by introducing delays  $D1$  through  $DN$ .
- The  $J$  inputs of such a unit now will be multiplied by several weights, one for each delay and one for undelayed input.
- For  $N=2$ , and  $J=16$ , for example, 48 weights will be needed to compute the weighted sum of the 16 inputs, with each input now measured at 3 different points in time.

# TDNN for ASR

## A time delay neural network architecture for efficient modeling of long temporal contexts

*Vijayaditya Peddinti<sup>1</sup>, Daniel Povey<sup>1,2</sup>, Sanjeev Khudanpur<sup>1,2</sup>*

<sup>1</sup>Center for Language and Speech Processing &

<sup>2</sup>Human Language Technology Center of Excellence  
Johns Hopkins University, Baltimore, MD 21218, USA

`vijay.p, khudanpur@jhu.edu, dpovey@gmail.com`

### Abstract

Recurrent neural network architectures have been shown to efficiently model long term temporal dependencies between acoustic events. However the training time of recurrent networks is

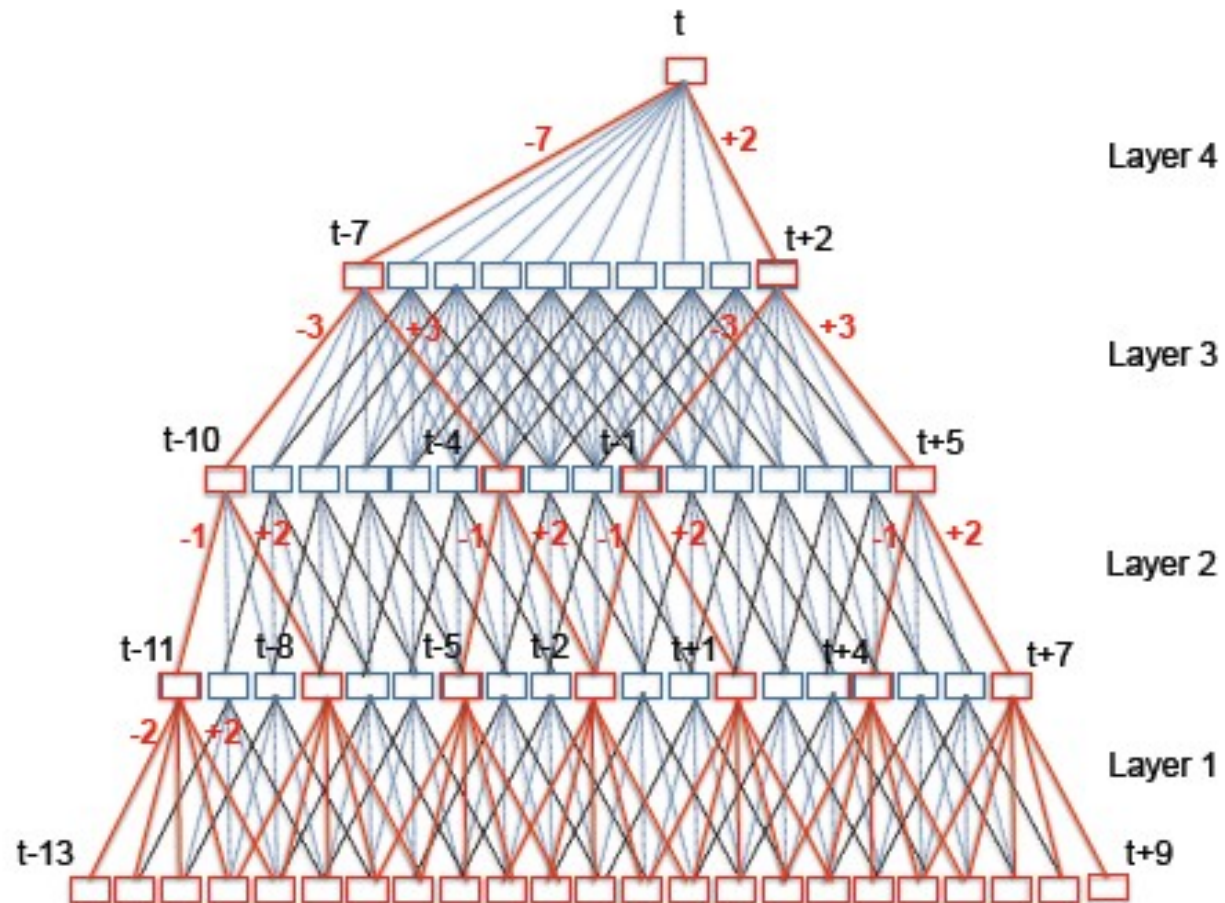
be reduced, while ensuring that information from all time steps in the input context is processed by the network.

Neural network architectures have been shown to benefit from speaker adaptation. However, speaker adaptation techniques like fMLLR [4] require two passes of decoding. The 2-

# TDNN for ASR

- In a TDNN architecture the initial transforms are learnt on narrow contexts and the deeper layers process the hidden activations from a wider temporal context.
- Hence the higher layers have the ability to learn wider temporal relationships.
- Each layer in a TDNN operates at a different temporal resolution, which increases as we go to higher layers of the network.

# TDNN for ASR



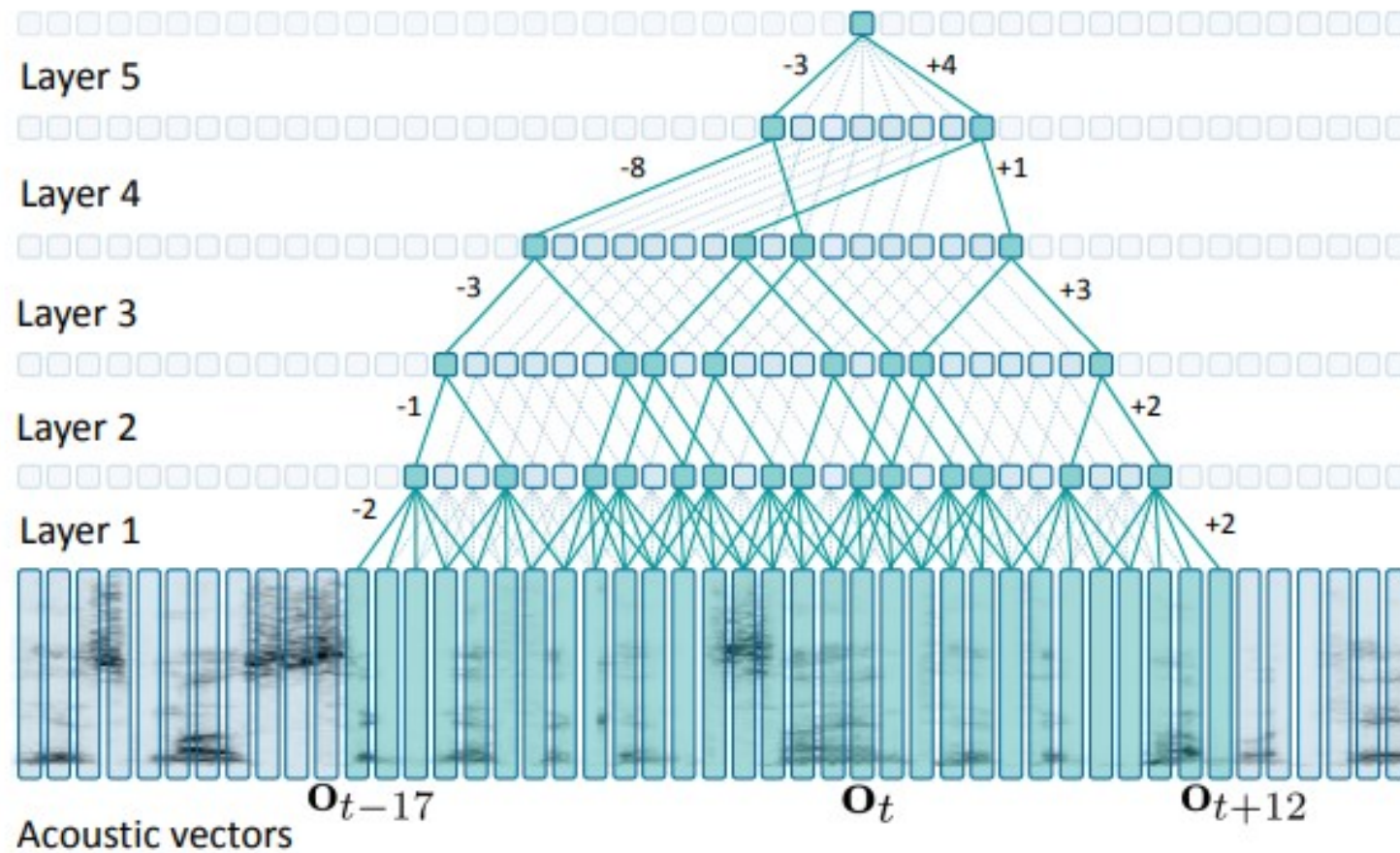
# TDNN

- Time-delay neural networks (TDNNs) belong to a particular type of CNNs that share weights along a single temporal dimension.
- Initially TDNN models have been proposed for the phoneme recognition task in [Lang and Hinton, 1988; Waibel et al., 1989]
- Later they were applied for spoken word recognition [Bottou et al., 1990] on-line handwriting recognition [Guyon et al., 1991] tasks.

# TDNN for ASR

- TDNNs allow the acoustic model to learn the temporal dynamics of the speech signal using short term acoustic feature vectors.
- Recently AMs with the TDNN topology have been shown to outperform state-of-the-art DNN-HMM ASR systems for many tasks [Peddinti et al., 2015; Povey et al., 2016].

# TDNN Architecture





# TDNN for ASR

- Shows an example of layer-wise context expansion scheme for a TDNN model.
- Each frame of the higher level corresponds to a longer context than the lower layers.
- This hierarchical temporal context expansion is different from the using of a wide contextual window of acoustic feature vectors in the input layer.

# Multiple Layers and Abstraction

- Different layers in the TDNN model correspond to different levels of abstraction in information, extracted from the speech signal.
- Local patterns in speech signal can be captured by the lower layers, while more complex structures can be learned by higher levels.

# Proposed Architecture

- Each layer has its own context extension and subsampling characteristics.
- For example, the first layer operates on the window of 5 frames  $\{\mathbf{o}_{t-2}, \mathbf{o}_{t-1}, \mathbf{o}_t, \mathbf{o}_{t+1}, \mathbf{o}_{t+2}\}$  of the input features.
- It is denoted as  $\{-2, -1, 0, 1, 2\}$  or simply as  $[-2, 2]$ .
- Layer 2 operates on the window of 4 frames of the Layer 1:  $[-1, 2]$ .

# Proposed Architecture

- In addition, Layer 2 has a sub-sampling, so it utilizes only boundary frames  $\{-1,2\}$  from context window  $[-1,2]$ .
- The frame connections that are used in TDNN after sub-sampling are shown in the figure with solid lines (the light dotted lines correspond to frame connections without subsampling).
- Finally, we can see, that the top layer has an indirect connection to the input acoustic vectors layer by means of context window  $[-17,12]$ .

# Viewing TDNN as 1-D Convolution

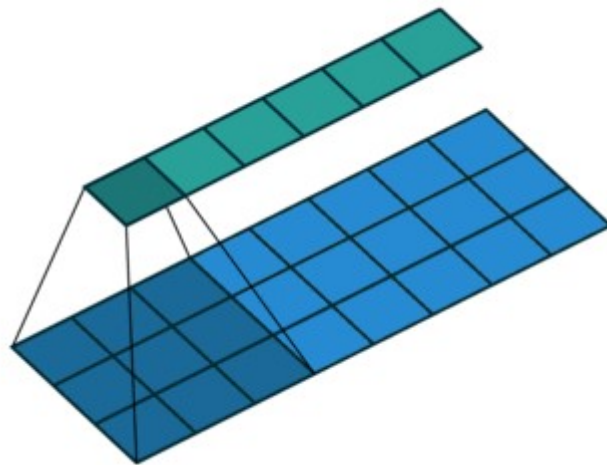
- *The TDNN is essentially a 1-d convolutional neural network without pooling and with dilations.*
- The inputs to the network are the frames of [acoustic features](#).
- The outputs of the TDNN are a probability distribution over each of the phones defined for the target language.
- That is, the goal is to read the audio one frame at a time, and to classify each frame into the most likely phone.

# Viewing TDNN as 1-D Convolution

- In one layer of the TDNN, each input frame is a column vector representing a single time step in the signal, with the rows representing the feature values.
- The network uses a smaller matrix of weights (the kernel or filter), which slides over this signal and transforms it into an output using the convolution operation.

# Viewing TDNN as 1-D Convolution

- We can visualize the kernel (dark blue area) moving over the input (light blue) to produce the output (green) like so:

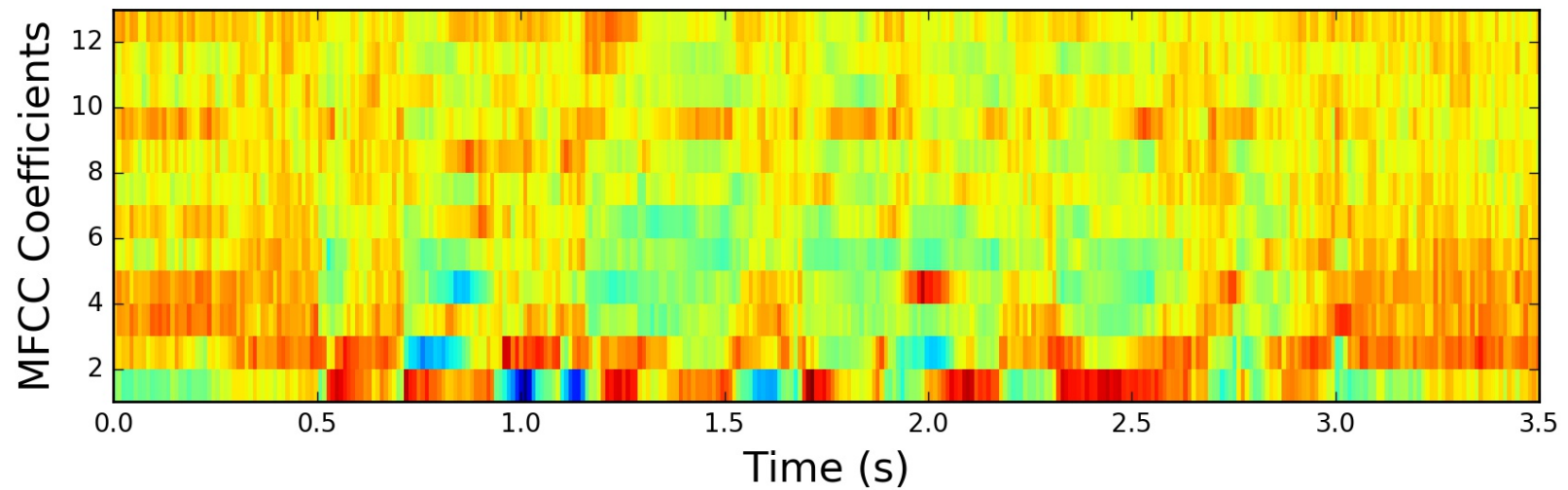
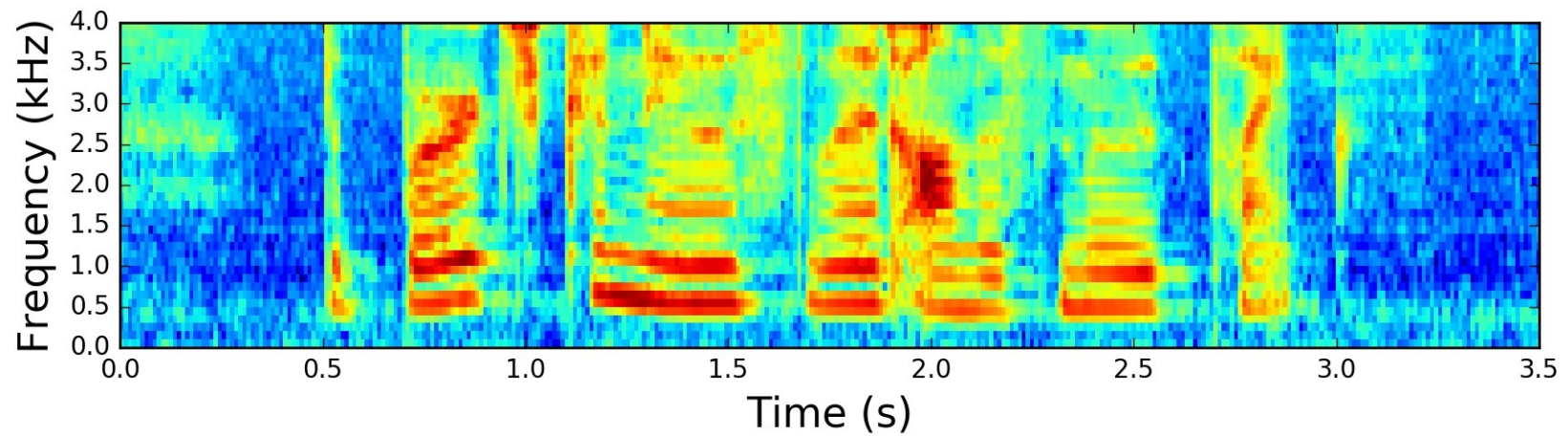


# Viewing TDNN as 1-D Convolution

- Mathematically, let's assume that we have an input vector  $x_t \in \mathbb{R}^m$ .
- $x_t$  contains some numbers in it, such as the amplitudes at a given frequency, or the values in a filter bank bin.
- We collect these vectors into a matrix, where each vector represents one time step  $t$  of our speech signal e.g. one sample taken every 10 milliseconds.

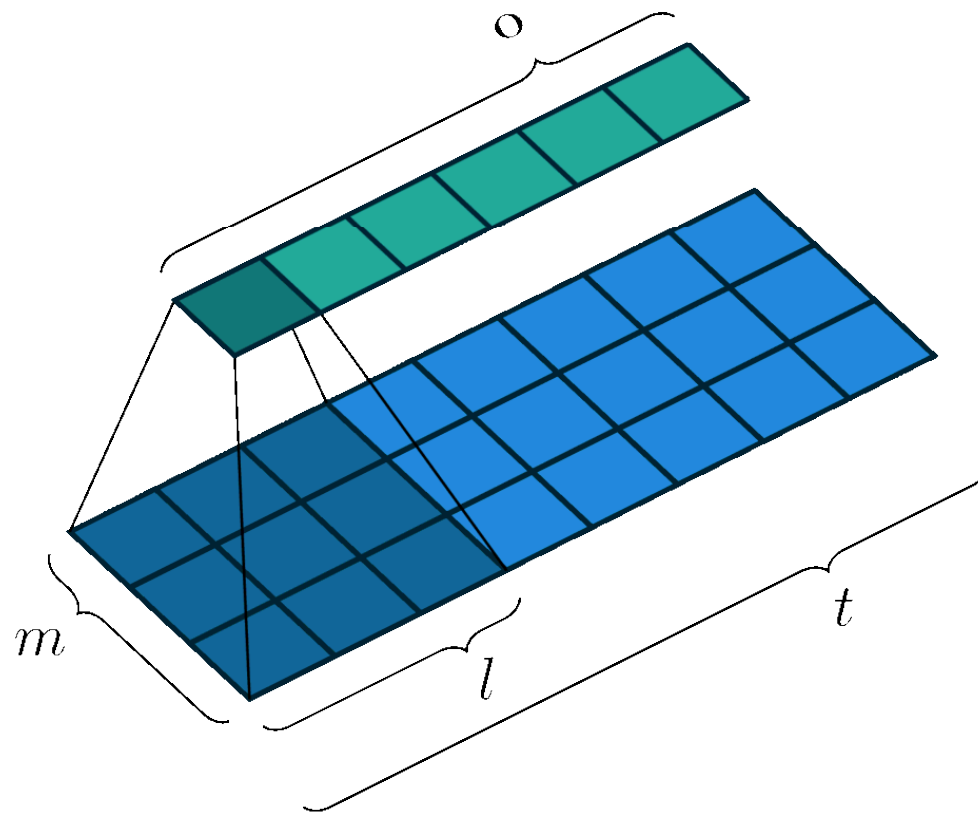


# Spectrogram and MFCCs



# Viewing TDNN as 1-D Convolution

- Observing the input over time, we then have the matrix of input features  $X \in \mathbb{R}^{m \times t}$ .
- Let us also define a trainable weight matrix (the kernel or filter)  $W \in \mathbb{R}^{m \times l}$ .
- where the kernel has the same height of  $m$ , and a width of  $l$ .



# Sliding and Padding

- The kernel  $W$  slides over the input signal, with a stride of  $s$ , so it makes  $s$  steps at a time.
- The region on the input feature map that the kernel covers is called the receptive field.
- Depending on the implementation, the input could be padded with null values of padding of height  $m$  and length  $p$  on either end.
- For simplicity, let's assume we have no padding ( $p=0$ ), and that the kernel slides over the input signal one step at a time ( $s=1$ ).

# Output

- Moving in this way, the output width  $o$ , that is, the number of times that the kernel can “fit” across the length of the input sequence, is given by:

$$o = \left\lfloor \frac{t - l + 2p}{s} \right\rfloor + 1 \quad (1)$$

where  $\lfloor . \rfloor$  indicates the floor function.

# Convolution

- At each time step  $t$ , the TDNN performs a convolution operation.
- Is an operation that takes an element-wise multiplication (Hadamard product) of the kernel weights over the input beneath it, and sums those multiples together.
- In the neural network, a trainable bias  $b$  is added (not pictured in the images above), and the result is passed through a non-linear activation function  $\varphi$  (e.g. sigmoid as in [\[4\]](#), rectified linear, or pp-norm as in [\[3\]](#))

# Convolution

- This forms the output  $z_q \in \mathbf{z}$ , where  $\mathbf{z}$  is the entire output vector after doing this operation for all the time steps (the light green vector in the images).

So, the scalar output of a single element  $z_q \in \mathbf{z}$ , at output step  $q \in \{1, 2, \dots, o\}$ , can be succinctly given by:

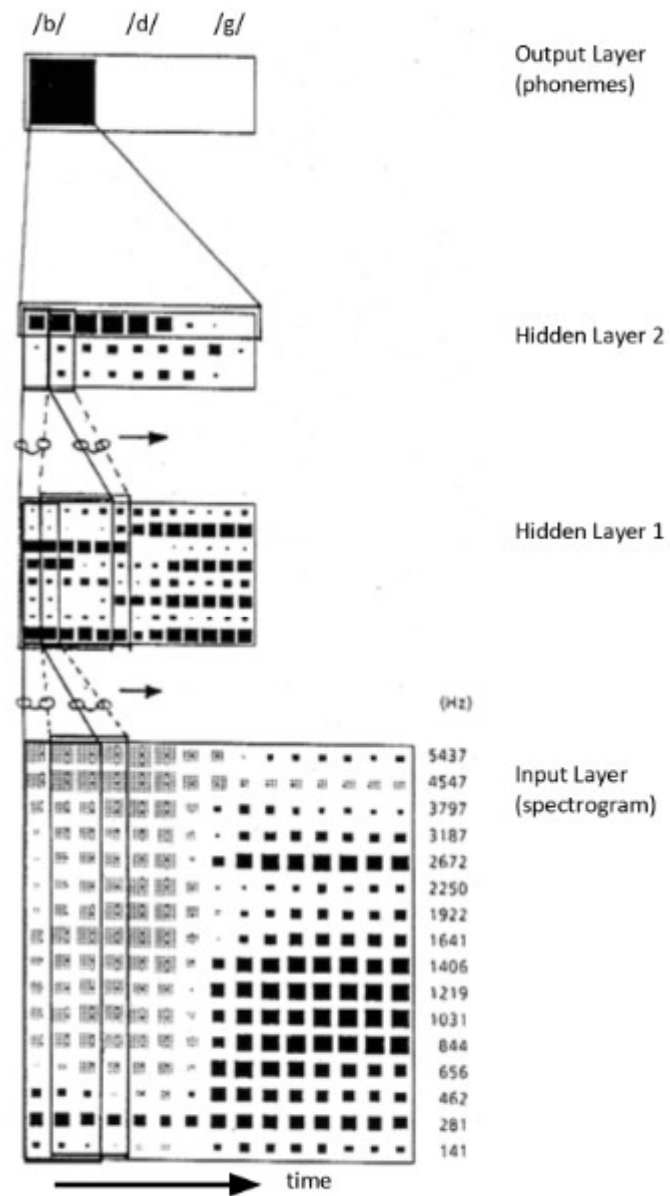
$$z_q = \phi(\mathbf{W} * \mathbf{X}_q + \mathbf{b}) \quad (2)$$

where  $*$  denotes the convolution operation and  $\mathbf{X}_q$  are the inputs in the receptive field.

It can also be equivalently given by:

$$z_q = \phi\left(\sum_{i=1}^m \sum_{k=1}^l w_{i,k} x_{i,k} + b\right) \quad (3)$$

where the first summation ranges over the height of the [acoustic features](#), the second over the [width of the receptive field/width of the kernel](#).





# Convolution

- In this image, we see a contiguous TDNN (no dilations).
- We can see that receptive field for the first hidden layer covers three frames of the input feature map on the interval  $[-1,1]$  (i.e. a total of 3 frames: the target and  $\pm 1$  frames).
- it is convolved with 88 kernels to create the first hidden layer.
- A stride of 1 is used to pass over the entire input signal.

# Convolution

- Then, frames on the interval  $[-2,2]$  (i.e. a total of 5 frames) are convolved with 3 kernels to produce the second hidden layer.
- In the original paper, the rows from the second layer are then summed to produce the output distribution over the 3 output phone classes.

# Convolution through Matrix Multiplication

- To perform the convolution as a matrix operation, the input and kernel can be unrolled, and a dot product can be computed.
- That is, let's say we have an example input matrix and kernel matrix like so:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \end{bmatrix} \quad (5)$$

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \quad (6)$$

# Convolution through Matrix Multiplication

We then create a new sparse matrix for the kernel of size  $o \times n$ , where  $o$  is the number of output steps as described above and  $n$  is the length of the unrolled input vector. The entries in each row are the weights, where needed, or 0s elsewhere (a [Toeplitz](#) matrix).

We can now take the dot product with the inputs like so:

$$\begin{bmatrix} w_{1,1} & w_{1,2} & 0 & 0 & w_{2,1} & w_{2,2} & 0 & 0 & w_{3,1} & w_{3,2} & 0 & 0 \\ 0 & w_{1,1} & w_{1,2} & 0 & 0 & w_{2,1} & w_{2,2} & 0 & 0 & w_{3,1} & w_{3,2} & 0 \\ 0 & 0 & w_{1,1} & w_{1,2} & 0 & 0 & w_{2,1} & w_{2,2} & 0 & 0 & w_{3,1} & w_{3,2} \end{bmatrix} \cdot \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ x_{1,4} \\ x_{2,1} \\ x_{2,2} \\ x_{2,3} \\ x_{2,4} \\ x_{3,1} \\ x_{3,2} \\ x_{3,3} \\ x_{3,4} \end{bmatrix} \quad (7)$$

Thank you!

Mail ID: [deepak@iiitdwd.ac.in](mailto:deepak@iiitdwd.ac.in)