

C05M05S01- What Makes a Neural Network Recurrent

Deep Learning & Neural Networks - Recurrent Neural Networks

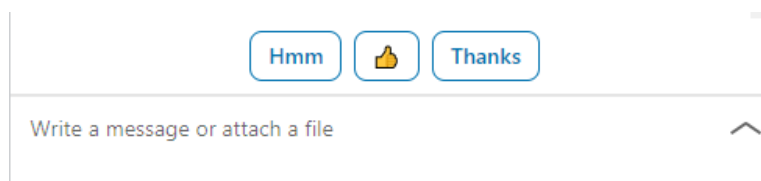
1. Introduction
2. What are Sequences?
3. What Makes the Network Recurrent?
4. Architecture of an RNN
5. Feeding Sequences to RNNs
6. Comprehension: RNN Architecture
7. Types of RNNs - I
8. Training RNNs
9. Types of RNNs - II
10. Vanishing and Exploding Gradients in RNNs
11. Summary
12. Graded Questions

Introduction

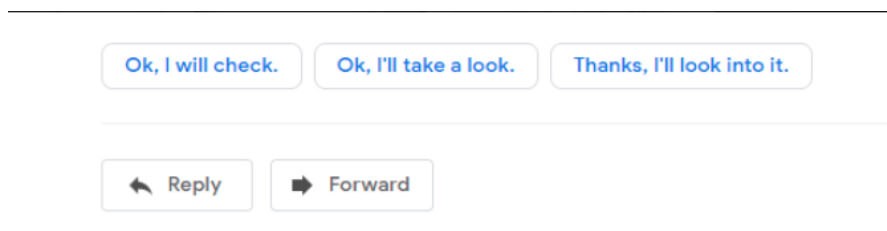
Recurrent Neural Networks or **RNNs** are specially designed to work with **sequential data**, i.e. data where there is a natural notion of a 'sequence' such as text (sequences of words, sentences etc.), videos (sequences of images), speech etc. RNNs have been able to produce state-of-the-art results in fields such as natural language processing, computer vision, and time series analysis.

One particular domain RNNs have revolutionised is **natural language processing**. RNNs have given, and continue to give, state-of-the-art results in areas such as machine translation, sentiment analysis, question answering systems, speech recognition, text summarization, text generation, conversational agents, handwriting analysis and numerous other areas. In computer vision, RNNs are being used in tandem with CNNs in applications such as image and video processing.

Many RNN-based applications have already penetrated consumer products. Take, for example, the **auto-reply** feature in many chat applications, as shown below:

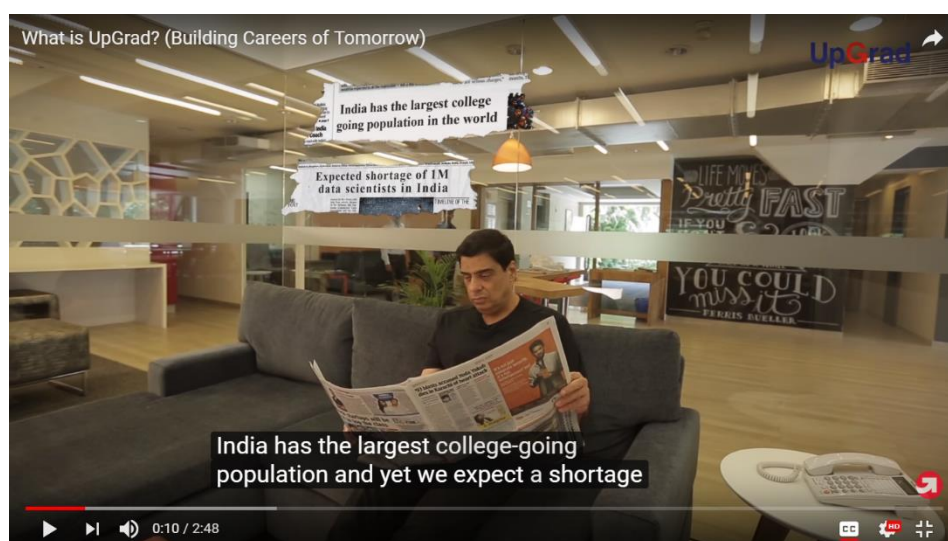


LinkedIn's auto-reply feature



Gmail's auto-reply feature

Auto-generated subtitles on YouTube has surprisingly good accuracy. This is an example of **automatic speech recognition (ASR)** which is built using RNNs.



Automatically generated captions in YouTube

Similarly, when talking to a support team of a food delivery app, or any other support team for that matter, We get an auto-reply in the initial stages of interaction where the support team asks about details such as order date, problem description and other basic things. Many of these conversational systems, informally called '**chatbots**', are trained using RNNs.

RNNs are also being used in applications other than NLP. Recently, OpenAI, a non-profit artificial intelligence research company came really close to defeating the world champions of Dota 2, a popular and complex battle arena game. The game was played between a team of five bots (from OpenAI) and a team of five players (world champions). The bots were trained using reinforcement learning and recurrent neural networks.

There are various companies who are generating music using RNNs. [Jukedeck](#) is one such company.

There are many other problems which are yet to be solved, and RNNs look like a promising candidate to solve them.

In this session:

- What are sequences?
- The architecture of an RNN
- Types of RNNs
- Drawbacks of RNN and motivation for its other variants

Prerequisites

There are no prerequisites for this session other than knowledge of the previous modules on Neural Networks and the courses on Statistics and ML.

What are Sequences?

Just like CNNs were specially designed to process images, **Recurrent Neural Networks (RNNs)** are specially designed to process **sequential data**.

In **sequential data**, entities occur in a particular order. If you break the order, you don't have a meaningful sequence anymore. For example, you could have a sequence of words which makes up a document. If you jumble the words, you will end up having a nonsensical document. Similarly, you could have a sequence of images which makes up a video. If you shuffle the frames, you'll end up having a different video. Likewise, you could have a piece of music which comprises of a sequence of notes. If you change the notes, you'll mess up the melody.

Recurrent neural networks are variants of the vanilla neural networks which are tailored to learn sequential patterns. Let's understand sequences from professor Raghavan.

UpGrad	
IID DATA <ol style="list-style-type: none">1. Independently and identically distributed (IID) data2. Each data point is sampled independently of any other data point3. Each of the data point is sampled from same distribution of data	EXAMPLES OF SEQUENTIAL DATA <ol style="list-style-type: none">1. Music2. Language3. Time series4. Image5. Video

You saw some examples of sequence problems. Let's now hear an interesting unconventional example of a problem involving sequences which can be solved using RNNs.

EXAMPLE PROBLEM INVOLVING A SEQUENCE		NEURAL NETWORKS	
Unsorted array	Sorted array	<div>1. Normal neural networks can approximate any given function</div> <div>2. Recurrent neural networks can simulate any given algorithm</div>	
[82, 6, 25, 12, 71]	[6, 12, 25, 71, 82]		
[34, 56, 1, 22, 55]	[1, 22, 34, 55, 56]		
[-3, 10, 0, 74, 9]	[-3, 0, 9, 10, 74]		
[34, 35, 67, 12, 8]	[8, 12, 34, 35, 67]		
<div><div>[-3, 10, 0, 74, 9]</div><div>Machine learning model</div><div>[-3, 0, 9, 10, 74]</div></div>			

Although sorting is a problem that involves an algorithm, but the fact that RNNs can learn an algorithm speaks volume about their capacity to solve hard learning problems.

What are sequences?

Which of the following is not a sequence problem? More than one options are correct.

☐ Predicting the action of a person in the next scene of a video using the past actions.

☒ Forecasting sales of a company using previous sales data



Q Feedback :

Time series data is sequential in nature.

☒ Classifying the action of a person in an image



Q Feedback :

An image is a static entity which does not involve sequences.

☐ Completing a sentence using the previous words of the sentence

☒ Predicting the salary of a person based on their age, the university that they graduated from, and experience in the industry



Q Feedback :

For a particular customer, the features listed in the answer don't have any kind of order or sequence.

In the next section, you'll look at the motivation behind an RNN.

What Makes the Network Recurrent?

Learn how a normal feedforward network is modified to work with sequences.

Let's quickly recall the feedforward equations of a normal neural network:

$z^l = W^l h^{l-1} + b^l$ $h^l = f^l(z^l)$ <p>where,</p> <p>W^l : weight matrix at layer l</p> <p>b^l : bias at layer l</p> <p>z^l : input into layer l</p> <p>f^l : activation function at layer l</p> <p>h^l : output or activations from layer l</p>	<p>In this session, we'll use a slightly different notation for the activation - rather than using h for the outputs (or activations) of a layer, we will use a. Thus, the feedforward equations become:</p> $z^l = W^l a^{l-1} + b^l$ $a^l = f^l(z^l)$
--	---

Let's now understand what makes this specialised neural network 'recurrent'.

<p>& NEURAL NETWORKS RNN - Motivation</p> <p>$W \cdot a^{(t)} + b^{(t)} = z^{(t)}$</p> <p>$a_t^{(l)} = \sigma(z^{(l)})$</p> <p>$a_{t+1}^{(l)} = g(a_t^{(l)}, z_{t+1}^{(l)})$</p> <p>Example: I did go for the match</p>	<p>Main difference between normal neural nets and RNNs is that RNNs have two 'dimensions' - time t (along the sequence length) and depth l (usual layers). Basic notation itself changes from a^l to a_t^l. In RNNs, it is somewhat incomplete to say '<i>the output at layer l</i>'; we rather say '<i>the output at layer l and time t</i>'.</p> <p>One way to think about RNNs is that the network changes its state with time (as it sees new words in a sentence, new frames in a video etc.). For e.g. we say that the state of a_t^l changes to a_{t+1}^l as it sees the next element in the sequence (word, image etc.)</p>
---	--

Thus, the output of layer l at time t+1, a_{t+1}^l , depends on two things:

1. Output of previous layer at the same time step a_{t+1}^{l-1} (this is the *depth* dimension).
2. Its own previous state a_t^l (this is the *time* dimension)

In other words, a_{t+1}^l is a function of a_{t+1}^{l-1} and a_t^l :

$$a_{t+1}^l = g(a_{t+1}^{l-1}, a_t^l)$$

We say that there is a **recurrent** relationship between a_{t+1}^l and its previous state a_t^l , and hence the name Recurrent Neural Networks.


These notations and ideas will be clearer going forward. Let's now look at the feedforward equations of an RNN in the following lecture.

G & NEURAL NETWORKS

Feedforward Equations of an RNN

$$a_{t+1}^{(l)} = \sigma(W_F^+ a_{t+1}^{(l-1)} + W_R a_t^{(l)} + b^{(l)})$$

$a_2^{(2)}$ $a_A^{(2)}$
This is a sample input.
 w_1 w_2 w_3 w_4 w_5
 $z_3^{(2)}$



To summarise, the output of layer l at time $t + 1$, $a_{t+1}^{(l)}$, depends on 1) the output of the previous layer at the same time step $a_{t+1}^{(l-1)}$ and 2) its own previous state $a_t^{(l)}$:

$$a_{t+1}^{(l)} = g(a_{t+1}^{(l-1)}, a_t^{(l)})$$

The feedforward equations are simply an extension of the vanilla neural nets - the only difference being that now there is a weight associated with both $a_{t+1}^{(l-1)}$ and $a_t^{(l)}$:

$$a_{t+1}^{(l)} = \sigma(W_F \cdot a_{t+1}^{(l-1)} + W_R \cdot a_t^{(l)} + b^{(l)}).$$

The W_F 's are called the **feedforward weights** and the W_R 's are called the **recurrent weights**.

Sequence Problems

The basic characteristic of a 'sequential dataset/problem' which is absent in non-sequence problems is that (choose the option that captures the essence of a 'sequence'):

☐ A sequence contains multiple elements which are fed to the network one by one, not all together

☐ The elements of a sequence can be reordered without affecting the sequence

☒ There is a significance dependence between the elements of a sequence, and that the order of the elements is important

Q Feedback:

The main idea is that of dependence and order - in a sequence, each element depends on the previous one and so on.

RNNs - Basic Idea

The most crucial idea of RNNs which makes them suitable for sequence problems is that:

☐ There are two sets of weight matrices W_F and W_R which increases the 'learning capacity' of the network

☒ The state of the network updates itself as it sees new elements in the sequence

Q Feedback:

This is the core idea of an RNN - it updates what it has learnt as it sees new inputs in the sequence. The 'next state' is influenced by the previous one, and so it can learn the dependence between the subsequent states which is a characteristic of sequence problems.

☐ They can take sequences as inputs which normal neural nets cannot do

RNN Feedforward

In the RNN feedforward expression below, m , n , o , p are variables.

$$a_{t+1}^{(l)} = \sigma(W_m \cdot a_{t+1}^{(n)} + W_o \cdot a_t^{(p)} + b^{(l)})$$

Choose the correct values of m , n , o , p . More than one options are correct.

☒ $m = F$

Q Feedback:

$a_{t+1}^{(l)} = \sigma(W_F \cdot a_{t+1}^{(l-1)} + W_R \cdot a_t^{(l)} + b^{(l)})$

☒ $n = l - 1$

Q Feedback:

$a_{t+1}^{(l)} = \sigma(W_F \cdot a_{t+1}^{(l-1)} + W_R \cdot a_t^{(l)} + b^{(l)})$

☐ $n = l$

☒ $o = R$

Q Feedback:

$a_{t+1}^{(l)} = \sigma(W_F \cdot a_{t+1}^{(l-1)} + W_R \cdot a_t^{(l)} + b^{(l)})$

☒ $p = l$

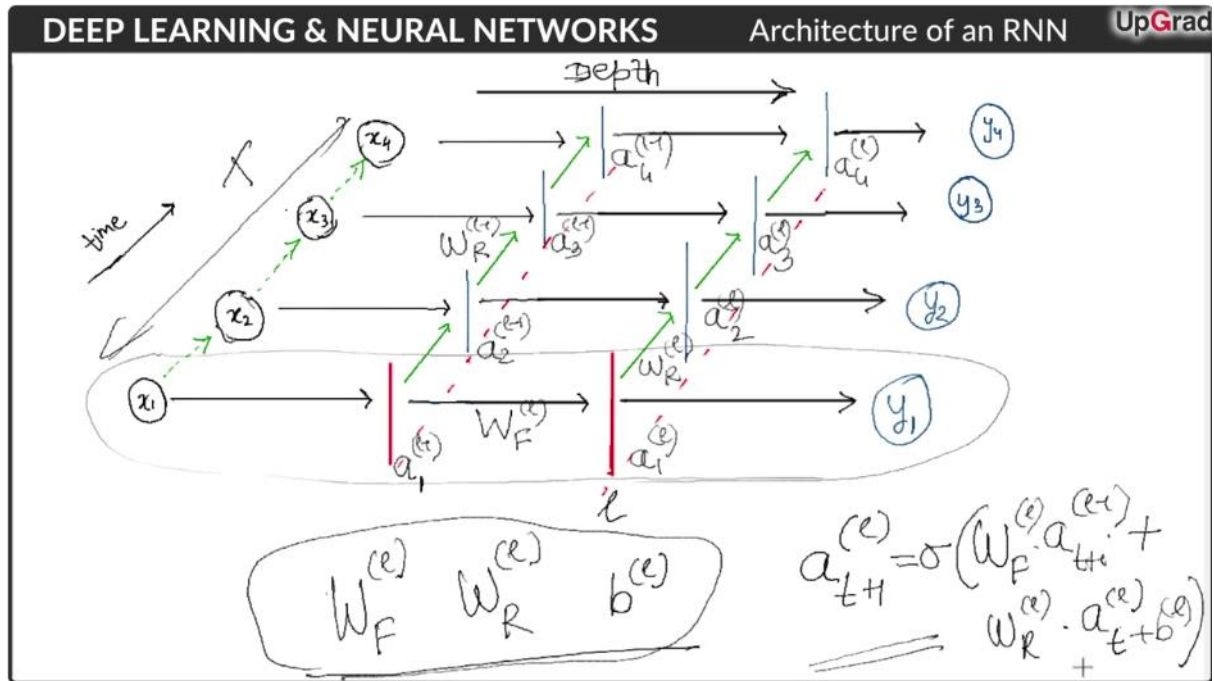
Q Feedback:

$a_{t+1}^{(l)} = \sigma(W_F \cdot a_{t+1}^{(l-1)} + W_R \cdot a_t^{(l)} + b^{(l)})$

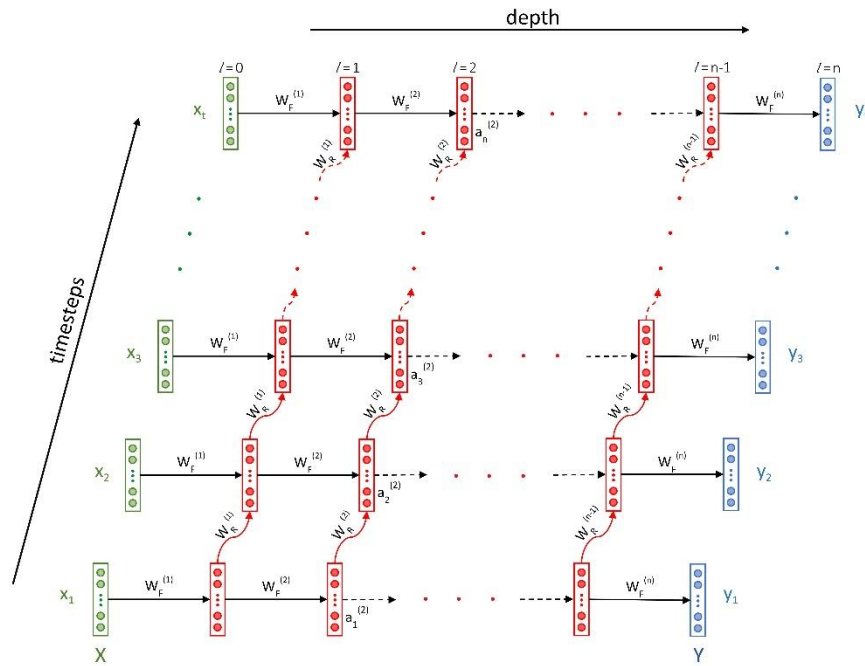
☐ $p = l - 1$

Architecture of an RNN

Let's now look at how the architecture of an RNN visually and compare it to a normal feedforward network.



The following figure shows the RNN architecture along with the feedforward equations:



$$z_t^{(l)} = W_F^{(l)} a_t^{(l-1)} + W_R^{(l)} a_{t-1}^{(l)} + b^{(l)}$$

$$a_t^{(l)} = f^{(l)}(z_t^{(l)})$$

Architecture of an RNN

The green layer is the **input layer** in which the x_i 's are elements in a sequence - words of a sentence, frames of a video, etc. The layers in red are the '**recurrent layers**' - they represent the various states evolving over time as new inputs are seen by the network.

The blue layer is the **output layer** where the y_i 's are the outputs emitted by the network at each time step. For example, in a parts of speech (POS) tagging task (assigning tags such as noun, verb, adjective etc. to each word in a sentence), the y_i 's will be the POS tags of the corresponding x_i 's. In this figure, input and output sequences are of equal lengths, but this is not necessary. To classify a sentence as 'positive/negative' (sentiment-wise), output layer will emit just one label (0/1) at the end of T timesteps.

Layers of an RNN are similar to vanilla neural nets (MLPs) - each layer has some neurons and is interconnected to previous and next layers. Only difference is that now each layer has a copy of itself along the time dimension (the various states of a layer, shown in red colour). Thus, the layers **along the time dimension**

. have the same number of neurons (since they represent the various states of same l th layer over time).

For example, let's say that layer-2 and layer-3 have 10 and 20 neurons respectively. Each of the red copies of the second layer will have 10 neurons, and likewise for layer-3.

Flow of information in RNNs is as follows: each layer gets the input from two directions - activations from the previous layer at the current timestep and activations from the current layer at the previous timestep. Similarly, the activations (outputs from each layer) go in two directions - towards the next layer at the current timestep (through W_F), and towards the next timestep in the same layer (through W_R).

<p>Architecture of an RNN</p> <p>The 'depth' of an RNN refers to:</p> <p><input type="radio"/> The number of time-steps of the input sequence</p> <p><input checked="" type="radio"/> The number of layers in the RNN</p> <p>🔗 Feedback : <i>Depth refers to the number of layers, not time steps.</i></p>	<p>Feedforward in RNNs</p> <p>The inputs going into the second layer at the 8th time step are (more than one options may be correct):</p> <p><input checked="" type="checkbox"/> a_7^2 ✓</p> <p>🔗 Feedback : The input to any layer l at time t has two components 1) the feedforward direction: the output from previous layer $l - 1$ at the same time t and 2) the recurrent direction: the output from the same layer l from the previous time step $t - 1$.</p> <p><input checked="" type="checkbox"/> a_8^1 ✓</p> <p>🔗 Feedback : The input to any layer l at time t has two components 1) the feedforward direction: the output from previous layer $l - 1$ at the same time t and 2) the recurrent direction: the output from the same layer l from the previous time step $t - 1$.</p> <p><input type="checkbox"/> a_8^2</p> <p><input type="checkbox"/> a_7^1</p>
<p>Architecture of an RNN - Neurons in Layers</p> <p>Which of the following pairs of outputs (activations) will necessarily have exactly the same sizes? Choose all the correct pairs:</p> <p><input type="checkbox"/> a_3^3, a_3^4</p> <p><input checked="" type="checkbox"/> a_3^4, a_5^4 ✓</p> <p>🔗 Feedback : The outputs of any layer at different steps will be of exactly the same size since they basically represent the output of the same layer evolving over time.</p> <p><input type="checkbox"/> a_5^2, a_5^4</p> <p><input checked="" type="checkbox"/> a_3^2, a_6^2 ✓</p> <p>🔗 Feedback : The outputs of any layer at different steps will be of exactly the same size since they basically represent the output of the same layer evolving over time.</p>	<p>Architecture of an RNN</p> <p>Each timestep at layer l can have its own separate activation function. The statement is:</p> <p><input type="radio"/> True</p> <p><input checked="" type="radio"/> False ✓</p> <p>🔗 Feedback : <i>All the timesteps at a particular layer are virtual copies of the same layer. The only difference between different timesteps at a particular layer is their state.</i></p>

Feeding Sequences to RNNs

In a previous segment, we had discussed sequences briefly. Let's now take a look at how various types of sequences are fed to RNNs.

DETECTING INCORRECT GRAMMAR

Sentence	Grammar
I really liked the Harry Potter movie series.	Correct
He didn't came to school yesterday.	Incorrect
Do you have a pen?	Correct
I am finished my lunch.	Incorrect
You bring the car?	Incorrect
....
....
The weather is really pleasant outside.	Correct
We were both in the same class.	Correct

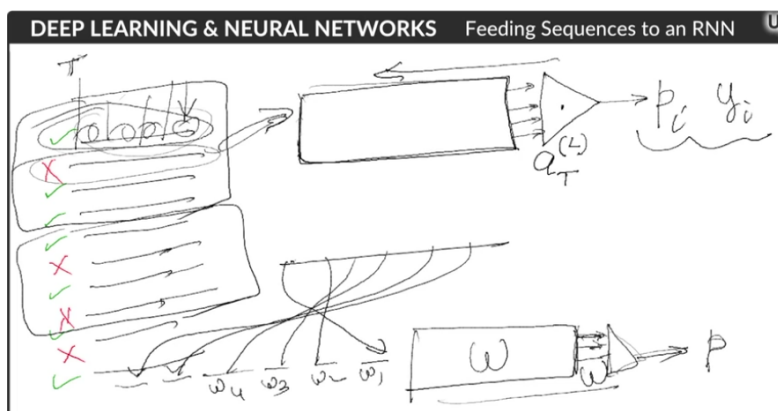
G & NEURAL NETWORKS Feeding Sequences to an RNN UpGrad

$$a_{t+1}^{(2)} = \sigma(W_F^{(2)} a_{t+1}^{(1)} + W_R^{(2)} a_t^{(1)} + b^{(2)})$$

$a_2^{(2)}$ $a_4^{(2)}$

This is a sample input.

Now that you understand how an RNN consumes one sequence, let's see how do you train a batch of such sequences.



How to feed data to an RNN. In the case of an RNN, each data point is a sequence.

Individual sequences are assumed to be **independently and identically distributed (I.I.D.)**, though the entities within a sequence may have a dependence on each other.

As network keeps ingesting new elements in the sequence it updates its current state (i.e. updates its activations after consuming each element in the sequence). After sequence is finished (say after T time steps), output from the last layer of the network $a_T^{(1)}$ captures the representation of the entire sequence. Do whatever you want with this output - use it to classify the sentence as correct/incorrect, feed it to a regression output, etc. This is exactly analogous to the way CNNs are used to learn a representation of images, and one can use those representations for a variety of tasks.

Data can be fed in **batches** just like any normal neural net - a batch here comprises of multiple data points (sequences).

IID Data

We discussed that the inputs to an RNN are sequences, and that sequences are assumed to be independently and identically distributed (IID). Which of the following types of data are IID distributed? More than one options may be correct:

- ☐ In a sentence classification task (grammatically correct/incorrect), the words in each sentence are IID
- ☒ In a sentence classification task (grammatically correct/incorrect), the individual sentences used for training are IID ✓
- ☐ In a video classification task (contains violence/does not contain violence), the individual frames in each video are IID
- ☒ In a video classification task (contains violence/does not contain violence), the individual videos are IID ✓

Feeding Sequences to RNN

Which of the following is true about the state of an RNN?

- ☐ The state changes only after the RNN consumes the entire sequence.
- ☒ The state changes each time after the RNN consumes an element of the sequence. ✓
Q Feedback :
The state of the RNN changes after it consumes each entity of the sequence.
- ☐ A and B both will result in the same state, so it doesn't matter.

Feeding Sequences to RNN

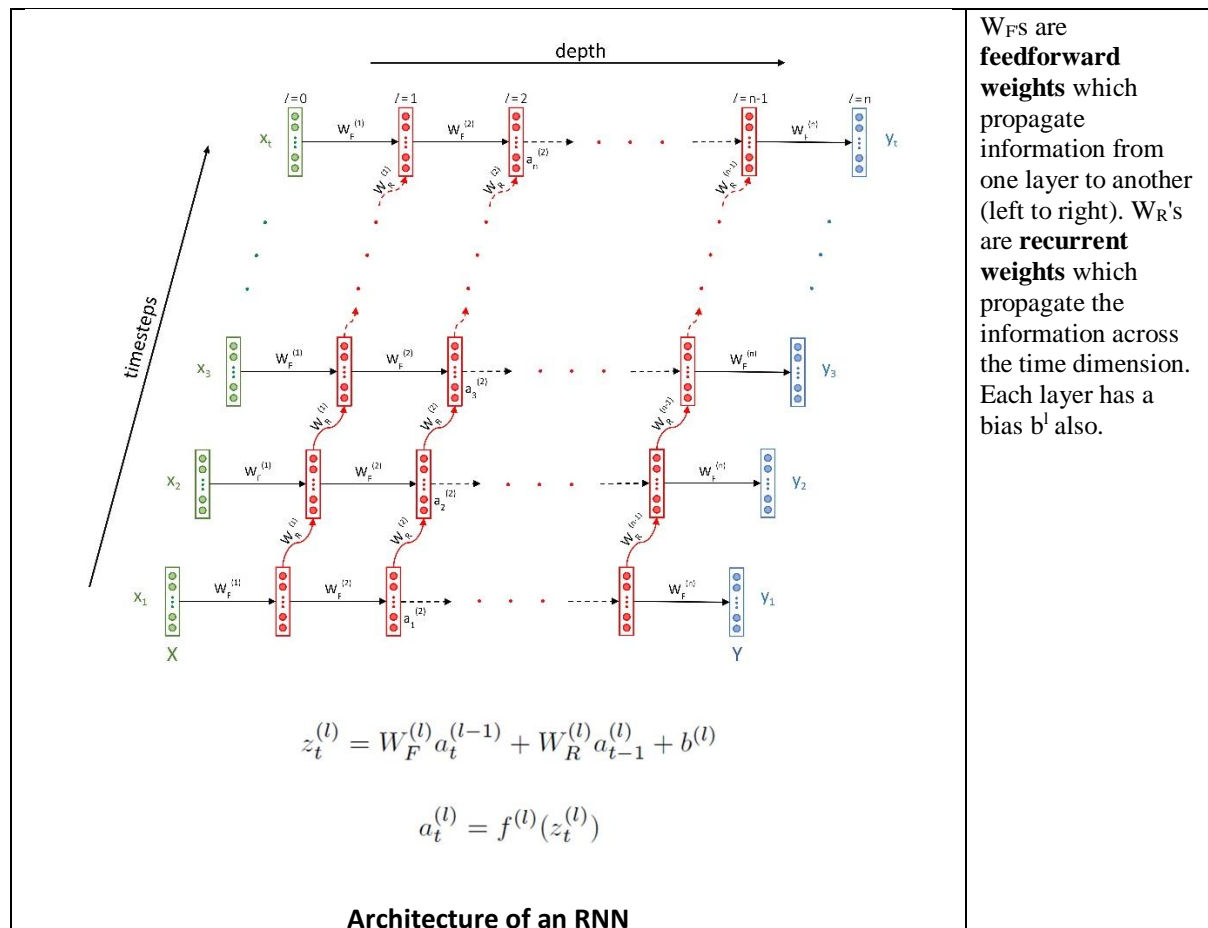
Consider a batch of 50 sequences. Suppose you have trained an RNN model with these sequences. Let's call this model A. Now, suppose you want to train another model B which has the exact same architecture as model A. Assume that the training is not stochastic in nature (that is, all the seed values are same while initialising the network parameters). Which of the following actions will result in a different set of weights for model B after the training is done in the exact same fashion as done in case of model A?

- ☒ Shuffling the order of entities within each sequence. ✓
Q Feedback :
Since entities within a sequence are dependent on each other, changing the sequence will result in a different sequence altogether. This will result in the RNN learning a different representation for the sequence and model B would eventually end up having different set of weights than model A.
- ☐ Shuffling the order of the sequences in the training data.
- ☐ A and B both
- ☐ None of the above

Comprehension: RNN Architecture

Previous section covered the feedforward equations of RNNs. Let's now analyse the architecture in a bit more detail - we will compute the dimensions of the weight and bias matrices, the outputs of layers etc. We will also look at a concise form of writing the feedforward equations.

Architecture of an RNN and its feedforward equations are as follows:



Let's now analyse the dimensions of the weight matrices and the biases. Dimensions of the weights W_F and the bias term b from the knowledge of the first module on neural networks.

<p>RNN Comprehension</p> <p>What are the dimensions of the weight W_F matrix at layer l, that is, dimensions of W_F^l?</p> <div style="margin-top: 10px;"> <input type="radio"/> (#neurons in layer l, #neurons in layer $l+1$) </div> <div style="margin-top: 10px;"> <input type="radio"/> (#neurons in layer $l+1$, #neurons in layer l) </div> <div style="margin-top: 10px; background-color: #e0ffe0; padding: 5px;"> <input checked="" type="radio"/> (#neurons in layer l, #neurons in layer $l-1$) </div> <div style="margin-top: 10px;"> <p>Feedback:</p> <p>W_F is the same weight matrix that you studied in the first module as W.</p> </div> <div style="margin-top: 10px;"> <input type="radio"/> (#neurons in layer $l-1$, #neurons in layer l) </div>	<p>RNN Comprehension</p> <p>What are the dimensions of the bias term b at layer l, that is, dimensions of b^l?</p> <div style="margin-top: 10px;"> <input type="radio"/> (#neurons in layer $l+1$, 1) </div> <div style="margin-top: 10px; background-color: #e0ffe0; padding: 5px;"> <input checked="" type="radio"/> (#neurons in layer l, 1) </div> <div style="margin-top: 10px;"> <p>Feedback:</p> <p>Bias at layer l is a vector with the number of neurons as its size.</p> </div> <div style="margin-top: 10px;"> <input type="radio"/> (#neurons in layer $l-1$, 1) </div> <div style="margin-top: 10px;"> <input type="radio"/> (1, #neurons in layer $l-1$) </div>
--	--

W_F 's are the usual feedforward weights. Now let's look at the dimensions of the other entities.

Recurrent weights W_R connect same layer to its different states across different time steps. Recurrent weights of layer-3 W_R^3 connects outputs of third layer from one-time step to the next: a_1^3 to a_2^3 , a_2^3 to a_3^3 and so on. Each of these outputs **have the same size** (it is same layer, so number of neurons at each time step is same, and hence output size is the same). Thus, all W_R 's are **square matrices**. The size of the output vector a_{t+1}^l (for any l and t) is the same as that of a_t^l . In other words, the recurrent operation does not change the size of the output.

biases of each layer b_l , as usual, have the **size equal to the number of neurons** in that layer.

Number of parameters and shapes of outputs at each layer. Since data is usually fed in batches (of size m), output from each layer a_t^l is a matrix of size (l, m) . The following table summarise the sizes of the various components of an RNN.

Term	Size
$W_F^{(l)}$	(#neurons in layer l , #neurons in layer $l-1$)
$W_R^{(l)}$	(#neurons in layer l , #neurons in layer l)
$b^{(l)}$	(#neurons in layer l , 1)
$z_t^{(l)}$	(#neurons in layer l , batch_size)
$a_t^{(l)}$	(#neurons in layer l , batch_size)

Matrix Dimensions

Consider a neural network with three neurons in input layer (layer-0), 7 neurons in the only hidden layer (layer-1) and a single neuron in the output softmax layer (layer-2). Consider a batch size of 64. We'll denote parameters of layers 0, 1 and 2 as (W^0, b^0) , (W^1, b^1) , (W^2, b^2) respectively.

This network is being trained to classify each input sentence as grammatically correct or incorrect. The sequence size, that is, the number of words in a sentence (= the number of time steps T) is 10.

'what if all sentences do not have exactly 10 words?', padding.

<p>Matrix Dimensions in an RNN In the above example, what will be the size of W_F^1 (Feedforward weight matrix at layer 1)?</p> <p><input type="radio"/> (3, 7)</p> <p><input type="radio"/> (21, 1)</p> <p><input checked="" type="radio"/> (7, 3)</p> <p>Feedback: Refer the table to see the dimensions of W_F^1. The size of matrix W_F^1 is equal to (#neurons in layer 1, #neurons in layer 0).</p> <p><input type="radio"/> (21, 64)</p>	<p>Matrix Dimensions in an RNN In the above example, what will be the size of W_R^1 (Recurrent weights at the hidden layer, layer-1)?</p> <p><input type="radio"/> (3, 3)</p> <p><input checked="" type="radio"/> (7, 7)</p> <p>Feedback: Refer the table to see the dimensions of W_R^1. The size of matrix W_R^1 is equal to (#neurons in layer 1, #neurons in layer 1).</p> <p><input type="radio"/> (21, 64)</p> <p><input type="radio"/> (64, 64)</p>
--	---

<p>Matrix Dimensions in an RNN What is the size of the matrix $W_F^1 a_4^0$? Assume that a batch size of 64 is used.</p> <p><input checked="" type="radio"/> (7, 64)</p> <p>Q Feedback : Dimensions of W_F^1 are (7, 3). Dimensions of a_4^0 are (3, 64). When you multiply these matrices you'll get a resultant matrix of size (7, 64)</p> <p><input type="radio"/> (2, 7)</p> <p><input type="radio"/> (4, 64)</p> <p><input type="radio"/> (2, 64)</p>	<p>Matrix Dimensions in an RNN What is the size of the matrix $W_R^1 a_6^1$? Assume that a batch of 64 data points is used.</p> <p><input checked="" type="radio"/> (7, 64)</p> <p>Q Feedback : Dimensions of W_R^1 are (7, 7). Dimensions of a_6^1 are (7, 64). When you multiply these matrices you'll get a resultant matrix of size (7, 64)</p> <p><input type="radio"/> (6, 64)</p> <p><input type="radio"/> (7, 6)</p> <p><input type="radio"/> (2, 64) First, find out the dimension of W_R^1 and then the dimension of a_6^1.</p>
<p>Matrix Dimensions in an RNN What is the size of b^1?</p> <p><input type="radio"/> (3, 1)</p> <p><input checked="" type="radio"/> (7, 1)</p> <p>Q Feedback : There are 7 neurons in the first layer. Therefore the size of bias at first layers is (7, 1).</p> <p><input type="radio"/> (2, 1)</p> <p><input type="radio"/> (64, 1)</p>	<p>Matrix Dimensions in an RNN What is the size of z_3^1? Assume that a batch of 64 data points is used.</p> <p><input type="radio"/> (64, 21)</p> <p><input type="radio"/> (3, 64)</p> <p><input type="radio"/> (7, 3)</p> <p><input checked="" type="radio"/> (7, 64)</p> <p>Q Feedback : The size of matrix $W_F^1 a_3^0$ and $W_R^1 a_2^1$ is (7, 64). The size of the bias (7, 1). The resultant term z_3^1 will also have the size (7, 64).</p>

Number of Trainable Parameters

Consider the first layer (the hidden layer) of the RNN having seven neurons parametrised by W_F^1, W_R^1, b^1 . How many trainable parameters are there in this layer? Note that all the time steps have the same recurrent weights W_R (i.e. you do not have 10 recurrent weights for 10 time steps, just one).

☐ 70

☒ 77

Q Feedback :
 W_F^1, W_R^1, b^1 are respectively of sizes (7, 3), (7, 7) and (7, 1). Thus, the number of parameters is $21+49+7=77$.

☐ 15

RNNs: Simplified Notations

A concise, simplified notation scheme for RNNs. The RNN feedforward equations are:

$$z_t^l = W_F^l a_t^{l-1} + W_R^l a_{t-1}^l + b^l$$

$$a_t^l = f^l(z_t^l)$$

The above equation can be written in the following matrix form:

$$z_t^l = [W_F^l \quad W_R^l] \begin{bmatrix} a_t^{l-1} \\ a_{t-1}^l \end{bmatrix} + b^l$$

In the above equation, the term $\begin{bmatrix} W_F^l & W_R^l \end{bmatrix} \begin{bmatrix} a_t^{l-1} \\ a_{t-1}^l \end{bmatrix}$ is equal to $W_F^l a_t^{l-1} + W_R^l a_{t-1}^l$

Merge the two weight matrices into one to get the following notation:

$$z_t^l = W^l [a_t^{l-1}, a_{t-1}^l] + b^l$$

where, W^l denotes the feedforward + recurrent weights of layer l formed by stacking (or concatenating) W_F^l and W_R^l side by side and $[a_t^{l-1}, a_{t-1}^l]$ is formed by stacking the activations a_t^{l-1} and a_{t-1}^l on top of each other. Try doing a consistency check of the dimensions once and convince yourself that the new notations are consistent with the old one.

This form is not only more concise but also more computationally efficient. Rather than doing two matrix multiplications and adding them, the network can do one large matrix multiplication.

Now consider the same example with the modified notations in mind. You have a neural network with three neurons in the input layer (layer-0), 7 neurons in the hidden layer (layer-1) and one neuron in the output softmax layer (layer-2). Consider a batch size of 64. The sequence size is 10.

<p>Matrix Dimensions in an RNN What is the size of W^1?</p> <p><input type="radio"/> (14, 10)</p> <p><input type="radio"/> (14, 3)</p> <p><input checked="" type="radio"/> (7, 10)</p> <p>Feedback: The size of W_F^1 and W_R^1 is (7, 3) and (7, 7) respectively. After you concatenate these matrices column-wise, you'll get a matrix of size (7, 10).</p> <p><input type="radio"/> (14, 7)</p>	<p>Matrix Dimensions in an RNN What is the size of $[a_6^0, a_4^1]$?</p> <p><input checked="" type="radio"/> (10, 64)</p> <p>Feedback: The size of matrices a_6^0 and a_4^1 is (3, 64) and (7, 64), respectively. After you concatenate these row-wise, that is, put one on top of the other, you'll get the resulting dimension as (10, 64).</p> <p><input type="radio"/> (3, 128)</p> <p><input type="radio"/> (7, 128)</p> <p><input type="radio"/> (3, 7)</p>
---	--

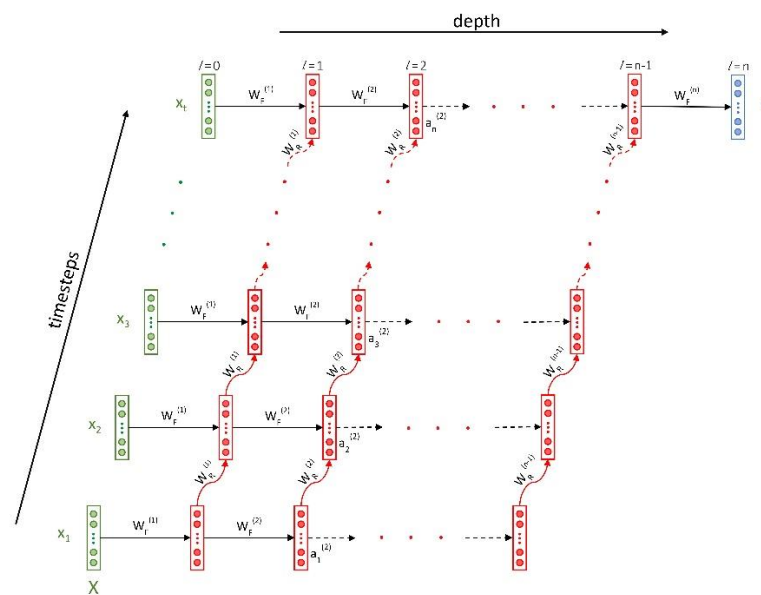
Next few sections cover various types of RNN architectures and some problems that can be solved using them.

Types of RNNs - I

Architecture of RNNs - there is an input sequence fed to the input layer, and an output sequence coming out from the output layer. The interesting part is that we can change the sizes and types of the input-output layers for different types of tasks. Let's discuss some commonly used RNN architectures:

Many-to-one RNN

In this architecture, the input is a sequence while the output is a single element. We have already discussed an example of this type - classifying a sentence as grammatically correct/incorrect. The figure below shows the **many-to-one** architecture:



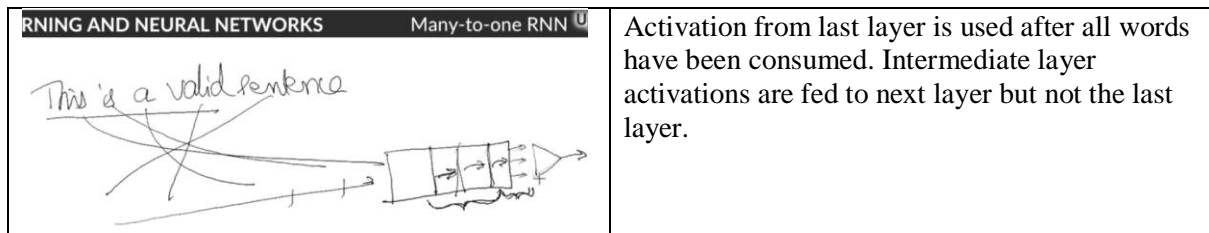
Many-to-one architecture

Each element of the input sequence x_i is a **numeric vector**. For words in a sentence, use a one-hot encoded representation, use word embeddings etc. Output is produced after the last timestep T (after the RNN has seen all the inputs).

Some other examples of many-to-one problems are:

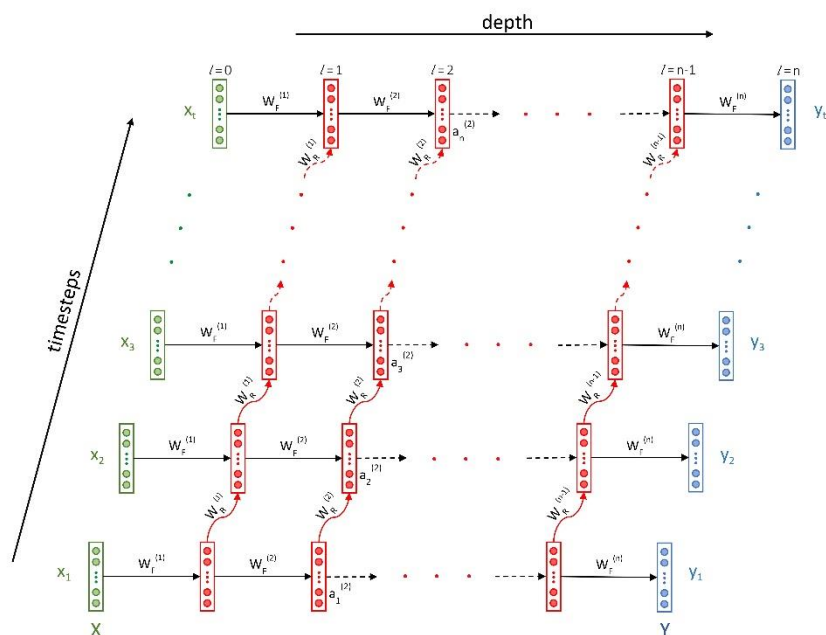
- Predicting the sentiment score of a text (between -1 to 1). For e.g., we can train an RNN to assign sentiment scores to customer reviews etc. Note that this can be framed as either a **regression problem** (where the output is a continuous number) or a **classification problem** (e.g. when the sentiment is positive/neutral/negative)
- Classifying videos into categories. For example, to classify YouTube videos into two categories 'contains violent content / does not contain violence'. The output can be a single softmax neuron which predicts the probability that a video is violent.

This architecture in the third session will generate C programming code using an RNN. Following lecture learns how to use a many-to-one architecture.



Many-to-many RNN: Equal input and output length

In this type of RNN, the input (X) and output (Y) both are a sequence of multiple entities spread over timesteps. The following image shows such an architecture.



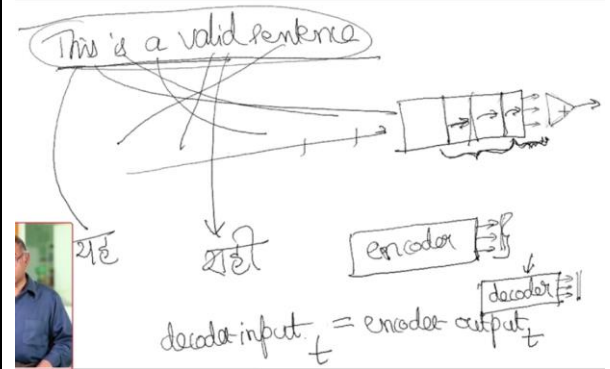
Many-to-many architecture

In this architecture, the network spits out an output at each timestep. There is a one-to-one correspondence between the input and output at each timestep. We can use this architecture for various tasks. In the third session of this module, this architecture is used to build a **part-of-speech tagger** where each word in the input sequence is tagged with its part-of-speech at every timestep.

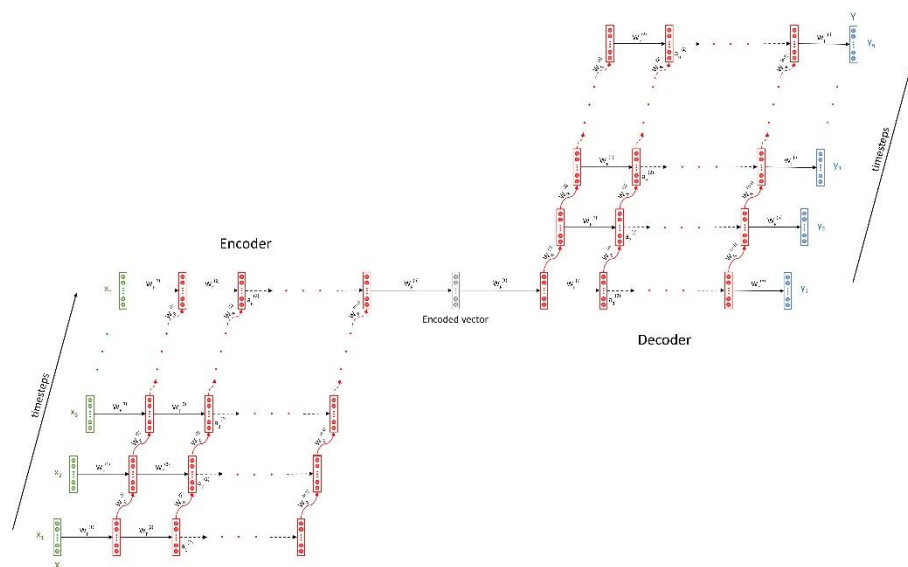
Many-to-many RNN: Unequal input and output lengths

In the previous many-to-many example of POS tagging, we had assumed that the lengths of the input and output sequences are equal. However, this is not always the case. There are many problems where the **lengths of the input and output sequences are different**. For example, consider the task of **machine translation** - the length of a Hindi sentence can be different from the corresponding English sentence.

Let's see how such problems can be solved using RNNs.

LEARNING AND NEURAL NETWORKS Encoder-Decoder RNN	
	<p>Encoder-decoder architecture is used in tasks where the input and output sequences are of different lengths.</p> <p>Encoder output is generated after every timestamp and fed into decoder for forming the sentence.</p>

The architecture is shown below:



Encoder-decoder architecture

The above architecture comprises of two components - an encoder and a decoder both of which are RNNs themselves. The output of the encoder, called the **encoded vector** (and sometimes also the '**context vector**'), captures a representation of the input sequence. The encoded vector is then fed to the decoder RNN which produces the output sequence.

Input and output can now be of different lengths since there is no one-to-one correspondence between them anymore. This architecture gives the RNNs much-needed flexibility for real-world applications such as language translation.

Encoder-decoder architecture

"The encoder and the decoder RNNs cannot have different number of layers". The statement is:

☐ True

☒ False



Feedback :

Since the encoder and decoder are different RNNs, they can have different depths.

Question-Answering

Say you want to build a question-answering system which takes in a question (such as "When did India gain independence?") and outputs an answer. The answer is an English sentence (just like a human would answer the question). Assume that the RNN has access to multiple such question-answer pairs for training. The suitable architecture for this task is:

☐ Many-to-one

☐ Many-to-many with one-to-one correspondence between the input and output sequences

☒ Encoder-decoder

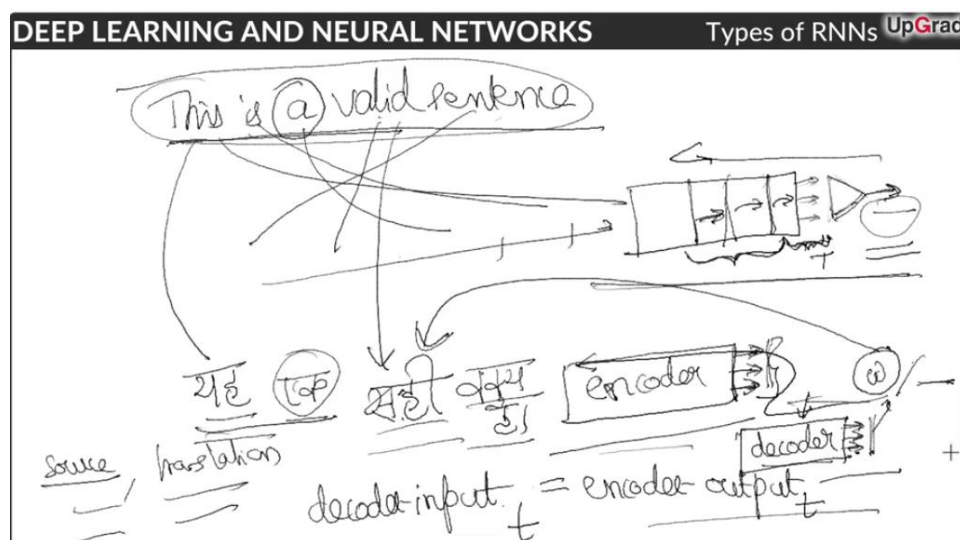


Feedback :

The encoder will 'encode' the question into a 'context or encoded vector' which can be fed to the decoder. The decoder outputs a sentence which can be of different length than the input.

Training RNNs

Previous segment covered different types of RNN architectures. Let's now study how RNNs are trained. The training procedure differs slightly for different architectures - let's study the training process and these differences.



Loss calculation depends on the type of task and the architecture. In a **many-to-one** architecture (such as classifying a sentence as correct/incorrect), loss is the difference between predicted and actual label. Loss is computed and backpropagated after entire sequence has been digested by the network.

In a **many-to-many** architecture, network emits an output at multiple time steps, and loss is calculated at each time step. **Total loss** (= sum of losses at each time step) is propagated back into network after entire sequence has been ingested.

Let's formally write down the loss expressions for a general RNN-based network. Let the network have T_1 input time steps and T_2 output time steps. The input-output pairs are thus $(x_1, x_2, \dots, x_{T_1})$ and $(y_1, y_2, \dots, y_{T_2})$ and T_1 and T_2 can be unequal.

In **many-to-one** architectures, the output length $T_2 = 1$, i.e. there is only a single output y_{out} for each sequence. If the actual correct label of the sequence is y , then the loss L for each sequence is (assuming a cross-entropy loss):

$$L = \text{cross-entropy}(y_{out}, y)$$

In a **many-to-many** architecture, if the actual output is $(y_1, y_2, \dots, y_{T_2})$ and the predicted output is $(y'_1, y'_2, \dots, y'_{T_2})$, then the loss L for each sequence is:

$$L = \sum_{i=1}^{T_2} \text{cross-entropy}(y'_i, y_i).$$

Encoder-Decoder RNN

Which of the following is true about backpropagation in an encoder-decoder RNN?

☐ The loss is calculated after each sequence.

☒ The loss is calculated after every timestep of a sequence.

🔍 Feedback :

The loss is calculated after each timestep.

☐ The decoded sequence is compared with the encoded sequence to calculate the loss.

Encoder-decoder RNN

"The loss is backpropagated to the encoder through the decoder". The statement is:

☐ True

🔍 Feedback :

Backpropagation starts from the output sequence of the decoder and ends at the input sequence.

☒ False

🔍 Feedback :

Backpropagation starts from the output sequence of the decoder and ends at the input sequence.

Encoder-Decoder RNNs

Which of the following is true about backpropagation in an encoder-decoder RNN? More than one options are correct.

☒ The loss corresponding to each element of the output sequence can be computed as they are being produced by the decoder ✓

🔍 Feedback :

The loss corresponding to each output element can be computed on the fly as the outputs are being emitted by the decoder layer. They are added to get the total loss after all the time steps are over.

☐ The loss corresponding to each element of the output sequence can be computed only after the entire sequence has been digested by the network

☐ The loss is backpropagated after the entire sequence (or a batch of sequences) has been ingested by the network ✓

🔍 Feedback :

Backpropagation only starts when the total loss of a sequence is computed (in fact, only after the total loss for a batch of sequences is computed).

☒ The total loss of a sequence is the sum of the losses of the individual output elements of the sequence ✓

🔍 Feedback :

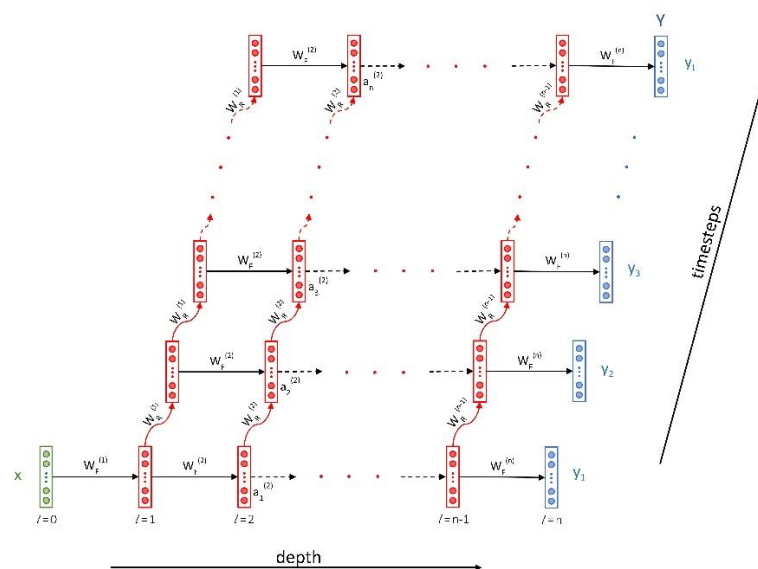
Self-explanatory.

Types of RNNs - II

Besides many-to-one and many-to-many RNN architectures, there is another type of RNN which hasn't been used in the fields of computer vision or natural language processing. Rather, it has been used by a wide variety of industries such as music and arts.

One-to-many RNN

In this type of RNN, the input is a single entity, and output consists of a sequence. The following image shows such an architecture:

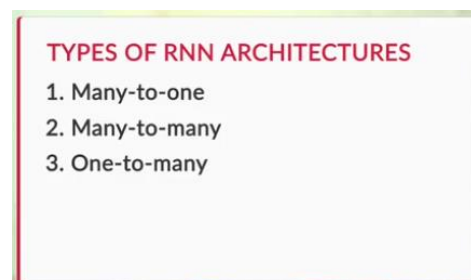


One-to-many architecture

In the above diagram, x denotes a single entity as input, and Y denotes multiple entities (y_1, y_2, \dots, y_n) and so on as outputs.

This type of architecture is generally used as a **generative model**. Among popular use of this architecture are applications such as generating music (given a genre, for example), generating landscape images given a keyword, generating text given an instruction/topic, etc.

In the following lecture, professor Raghavan explains how can a one-to-many architecture be used.



A one-to-many RNN can be used as a generative model.

<p>Types of RNN architectures</p> <p>Suppose you have a corpus of English documents along with the summary of each document. You want to train an RNN model to build a document summarizer. Which of the following architectures is suited best for this problem?</p> <p><input type="radio"/> One-to-many architecture</p> <p><input type="radio"/> Many-to-one architecture</p> <p><input checked="" type="radio"/> Encoder-decoder architecture ✓</p> <p>Q Feedback : The input comprises of a sequence, the document, which is greater in length than the length of the output sequence, the summary. Therefore an encoder-decoder architecture will be the most suitable of all the enlisted ones.</p> <p><input type="radio"/> Standard many-to-many</p> <p><input type="radio"/> None of the above</p>	<p>Types of RNN architectures</p> <p>Let's say you want to predict whether a person will click on an advertisement link based on features such as browsing time on the website, timestamp, gender, age, occupation, ethnicity, etc. Which of the following RNN architectures will be most suitable for this kind of problem?</p> <p><input type="radio"/> One-to-many</p> <p><input type="radio"/> Many-to-one</p> <p><input type="radio"/> Many-to-many</p> <p><input checked="" type="radio"/> None of the above ✓</p> <p>Q Feedback : The problem is doesn't involve any sequences. Therefore, an RNN won't be suitable for this type of problem. A standard feedforward network more appropriate in this case.</p>
<p>Types of RNN architectures</p> <p>Let's say you're working for a company which wants to build an AI keyboard for smartphones. The aim of the company is to build such a keyboard which predicts the next word based on previous few words that the user has typed. For example, when someone types "Hey, I will be late to the ___", the network should predict the next word (say) "office".</p> <p>Which RNN architecture would be the most suitable for this task?</p> <p><input type="radio"/> One-to-many architecture</p> <p><input checked="" type="radio"/> Many-to-one architecture ✓</p> <p>Q Feedback : The input is a sequence of words based on which the next word, a single entity, will be predicted. Hence, a many-to-one architecture will be most suitable in this case.</p> <p><input type="radio"/> Encoder-decoder architecture</p> <p><input type="radio"/> Standard many-to-many</p>	<p>Types of RNN architectures</p> <p>Suppose you're working for an online video streaming company that wants to caption each frame of the videos with the actor name that is present in each scene. What architecture do you think would be most suitable for this job?</p> <p><input type="radio"/> One-to-many architecture</p> <p><input type="radio"/> Many-to-one architecture</p> <p><input checked="" type="radio"/> Encoder-decoder architecture ✗</p> <p>Q Feedback : Think of the input and output of the network. The input is a video and the output is the actor's name in the scene.</p> <p><input type="radio"/> Standard many-to-many architecture ✓</p> <p>Q Feedback : The input is a video, which is nothing but a sequence of images, and the output is the actor's name in the scene.</p>

To summarise, these are the four main types of RNN architectures that you can use to build different kinds of applications:

- Many-to-one
- Many-to-many (same input-output lengths)
- Many-to-many, or encoder-decoder (different input-output lengths)
- One-to-many (generative models)

In the next section, we will discuss some important practical problems encountered while training RNNs.

Vanishing and Exploding Gradients in RNNs

Note: This is a text-only page introducing some important practical problems in training RNNs. We have introduced the basic idea on this page and have put the detailed discussion (which involves some detailed algebra) as an optional session in the interest of time. We highly recommend going through the optional session which is the last session of this module.

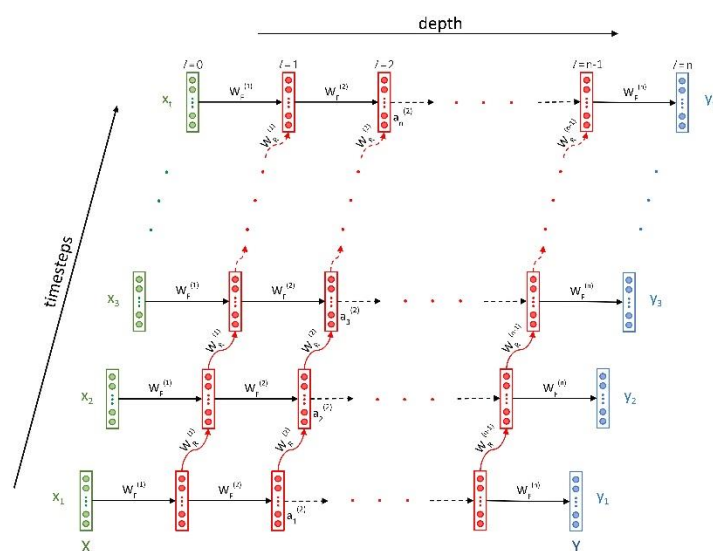
The Gradient Propagation Problem in RNNs

Although RNNs are extremely versatile and can (theoretically) learn extremely complex functions, the biggest problem is that they are **extremely hard to train** (especially when the sequences get very long).

RNNs are designed to learn patterns in sequential data, i.e. patterns across 'time'. RNNs are also capable of learning what are called **long-term dependencies**. For example, in a machine translation task, we expect the network to learn the interdependencies between the first and the eighth word, learn the grammar of the languages, etc. This is accomplished through the **recurrent layers** of the net - each state learns the cumulative knowledge of the sequence seen so far by the network.

Although this feature is what makes RNNs so powerful, it introduces a severe problem - as the sequences become longer, it becomes much harder to backpropagate the errors back into the network. The gradients 'die out' by the time they reach the initial time steps during backpropagation.

RNNs use a slightly modified version of backpropagation to update the weights. In a standard neural network, the errors are propagated from the output layer to the input layer. However, in RNNs, errors are propagated not only from right to left but also through the time axis.



Many-to-many architecture

Refer to the figure above - notice that the output y_t is not only dependent on the input x_t , but also on the inputs x_{t-1} , x_{t-2} , and so on till x_1 . Thus, the loss at time t depends on *all* the inputs and weights that appear before t . For e.g. if you change W_F^2 , it will affect all the outputs a_1^2, a_2^2, a_3^2 , which will eventually affect the output y_t (through a long feedforward chain).

This implies that the gradient of the loss at time t needs to be propagated backwards through *all the layers and all the time steps*. To appreciate the complexity of this task, consider a typical speech recognition task - a typical spoken English sentence may have 30-40 words, so you have to backpropagate the gradients through 40-time steps *and* the different layers.

This type of backpropagation is known as **backpropagation through time** or **BPTT**.

The exact backpropagation equations are covered in the optional session, though here let's just understand the high-level intuition. The feedforward equation is:

$$a_t^l = f(W_F^l a_t^{l-1} + W_R^l a_{t-1}^l + b^l)$$

Now, recall that in backpropagation we compute the gradient of subsequent layers with respect to the previous layers. In the time dimension, we compute the gradient of output at time t with respect to the output at $t - 1$, i.e. $\frac{\partial a_t^l}{\partial a_{t-1}^l}$. This quantity depends linearly on W_R , so $\frac{\partial a_t^l}{\partial a_{t-1}^l}$ is some function of W_R :

$$\frac{\partial a_t^l}{\partial a_{t-1}^l} = g(W_R)$$

Similarly, extending the gradient one time step backwards, the gradient $\frac{\partial a_{t-1}^l}{\partial a_{t-2}^l}$ will also be a function of W_R , and so on. The problem is that these gradients are eventually multiplied with each other during backpropagation ($\frac{\partial a_t^l}{\partial a_{t-1}^l} \cdot \frac{\partial a_{t-1}^l}{\partial a_{t-2}^l} \dots \frac{\partial a_2^l}{\partial a_1^l}$), and so the matrix W_R is raised to higher and higher power of its own, $(W_R)^n$, as the error propagates backwards in time.

The longer the sequence, the higher the power. This leads to **exploding or vanishing gradients**. If the individual entries of W_R are greater than one, the values in $(W_R)^n$ will explode to extremely large values; if the entries are lesser than one, $(W_R)^n$ will make them extremely small.

This problem seriously impedes the learning mechanism of the neural network.

Workarounds to solve the problem of exploding gradients impose an upper limit to the gradient while training, commonly known as **gradient clipping**. By controlling the maximum value of a gradient, we can do away with the problem of exploding gradients.

But the problem of vanishing gradients is a more serious one. The vanishing gradient problem is so rampant and serious in the case of RNNs that it renders RNNs useless in practical applications. One way to get rid of this problem is to use short sequences instead of long sequences. But this is more of a compromise than a solution - it restricts the applications where RNNs can be used.

To get rid of the vanishing gradient problem, researchers have been tinkering around with the RNN architecture for a long while. The most notable and popular modifications are the **long short-term memory units (LSTMs)** and the **gated recurrent units (GRUs)**.

<p>Backpropagation through time (BPTT) Refer the image of the architecture of RNN and then answer the following question.</p> <p>While backpropagating, one is supposed to calculate the gradient of the loss w.r.t. the weights sitting at different layers. While backpropagating, which of the following entities will have an influence on the gradient of y_3?</p> <p><input type="radio"/> x_1 only</p> <p><input type="radio"/> x_2 only</p> <p><input type="radio"/> x_3 only</p> <p><input checked="" type="radio"/> x_1, x_2 and x_3 ✓</p> <p>Q Feedback : y_3 will depend on each input till x_3.</p> <p><input type="radio"/> y_3 will depend on the entire input sequence X that is, x_1 to x_n.</p>	<p>Backpropagation through time (BPTT) Which of the following techniques will have no effect on the either of the two problems: vanishing and the exploding gradients?</p> <p><input type="radio"/> Making the sequences longer or shorter.</p> <p><input type="radio"/> Putting an upper limit to the gradient by clipping it.</p> <p><input checked="" type="radio"/> Changing the architecture from many-to-many to many-to-one ✓</p> <p>Q Feedback : Changing the architectures will have no effect in the architecture as far as calculations are concerned. The vanishing and exploding gradients problem is because of sequence length, not because of the type of architecture in use.</p> <p><input type="radio"/> None of the above will affect the exploding or vanishing gradient problem.</p>
--	---

Summary

Basic architecture of RNNs, some variants of the architecture applied to different types of tasks, and how the information flows between various layers during feedforward and backpropagation.

RNNs are designed to work on **sequences** which are present in many domains such as time series data, natural language processing, computer vision, music and audio, etc. Order of the elements in sequences is very important and we need something more than a standard feedforward neural network to capture the relationships between entities in a sequence.

Architecture of an RNN and its **feedforward equations**. There are two types of weight matrices - the feedforward weights which propagate the information through the depth of the network, and the recurrent weights which propagate information through time. The basic feedforward equation is:

$$a_t^l = f(W_F^l a_t^{l-1} + W_R^l a_{t-1}^l + b^l)$$

We also discussed the different **types of RNNs** and the tasks they are applicable to:

1. Many-to-one architecture
2. Many-to-many architecture
 1. Standard many-to-many architecture
 2. Encoder-Decoder architecture
3. One-to-many architecture

Finally, we briefly studied **backpropagation through time (BPTT)** and how it leads to the problem of **vanishing and exploding gradients** in RNNs.