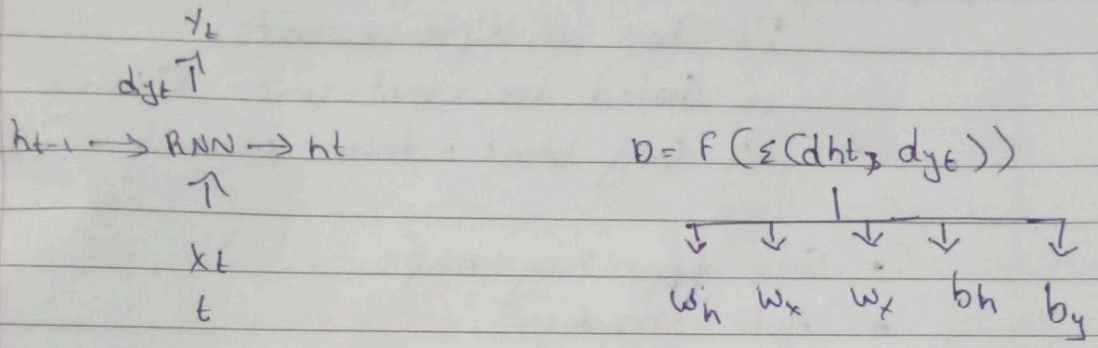
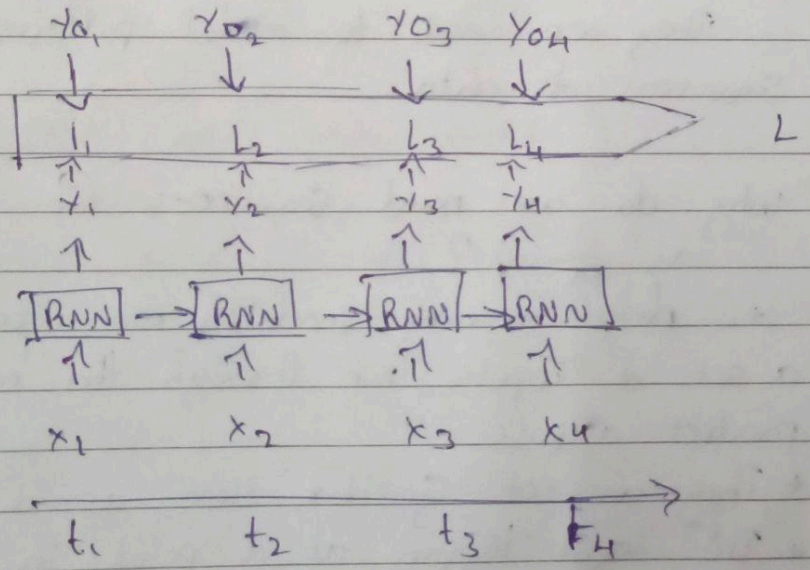
As AI Grows a number of fields are growing and application are grown for natural process

RNN Back Propagation



RNN loss:-



RNN are foundation for application that deal with sequences of data

RNN:- recurrent neural networks.

Deep learning Architectures:-

* CNN (convolutional neural networks) are popular in image recognition
* RNN (Recurrent neural networks) are used for creating models for sequential use cases
  It has multiple variants,
  1, Gated recurrent unit (GRU)
  2, Long short-term memory (LSTM)

* Self organizing maps.
* Auto Encoders.

RNN also called as sequence models.
They are used to model patterns in sequences of data

→ Why do we need sequences for neural network as special?
+ Deep learning models are used to take a set of inputs run through the model, and predict outputs
* They do not consider time as a factor
* The input happen at a point in time and output are generated based on these inputs
* These models are not dependent on what happened before and what will happen after
* That means they do not model relationship accross time

Example:- Predict word

let consider an example we want to predict the word between won (___ match

won ___ match.

we can that ~~we~~ There is no single answer to this problem because

He won his match

If the word before won "He" then the right prediction would be the word his
This Because we need to consider the gender which is indicated by word he, that occured two words before in the sequence
Similarly ~~we~~ if the word befor won is "she" then the right prediction would be her

She won her match.

If word before won is "They", it indicates a plural and the right prediction would be their

They won their match.

Here, the prediction depends upon what happend prior in this specific sequence

Sequence model:-

Sequences model predict future sequence based on the past sequence patterns

Example:- Predict word

let consider an example we want to predict the word between won (~~and~~ ___ match

won ___ match.

we can that ~~we~~ There is no single answer to this problem because

He won his match

If the word before won "He" then the right prediction would be the word his

This Because we need to consider the gender which is indicated by word he, that occured two words before in the sequence

Similarly ~~we~~ if the word befor won is "she" then the right prediction would be her

She won her match.

If word before won is "They", it indicates a plural and the right prediction would be their.

They won their match.

Here, the prediction depends upon what happend prior in this specific sequence

Sequence model:-

Sequences model predict future sequence based on the past sequence patterns

Example:- Predict word

let consider an Example we want to predict the word between won (~~and~~ ___ match

won ___ match.

we can that ~~as~~ There is no single answer to this problem because

He won his match

If the word before won "He" then the right prediction would be the word his

This Because we need to consider the gender which is indicated by word he, that occured two words before in the sequence

Similarly ~~or~~ if the word befor won is 'she" then the right prediction would be her

She won her match.

If word before won is "They" , if indicates a plural and the right prediction would be their

They won their match.

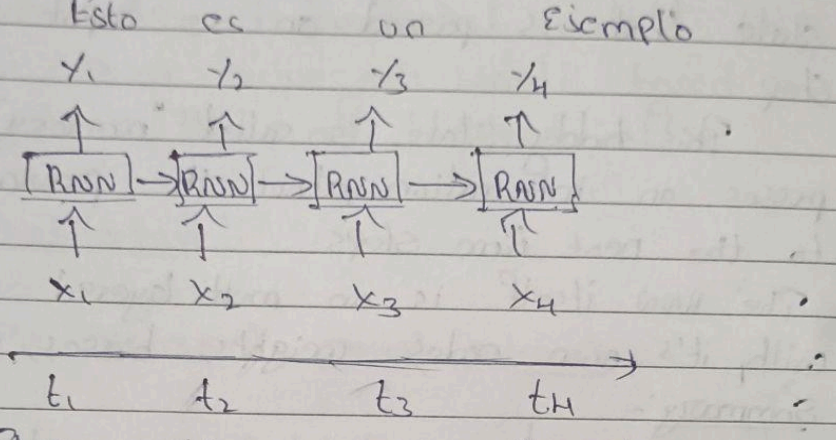
Here, the prediction depends upon what happend prior in this specific sequence

Sequence model:-

Sequences model predict future sequence based on the past sequence patterns

This use case is doing a one to one translation But real-life use cases may produce zero to multiple words for each input

RNN Example: Translation



This is an Example.

Note:-

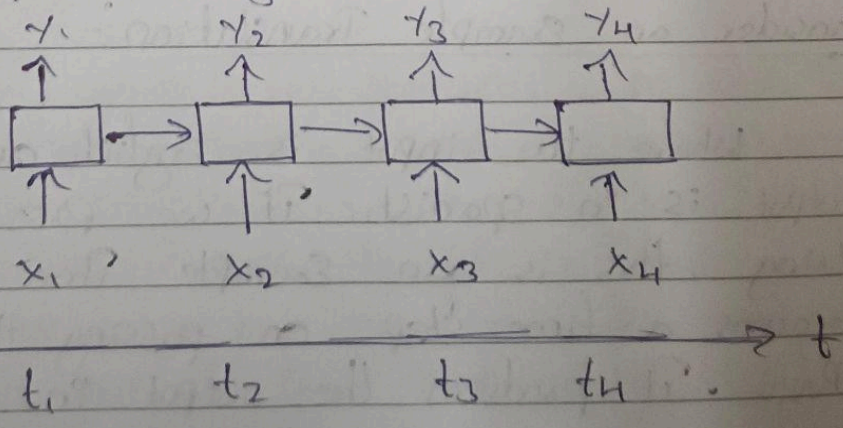
There are different types of RNN's based on the input & output mapping.

Types of RNN:-

1) Many to many RNN.

* Where each time step will have inputs and outputs

Ex:- Speech recognition.



Where as each word is spoken the output would be the text representation of word.

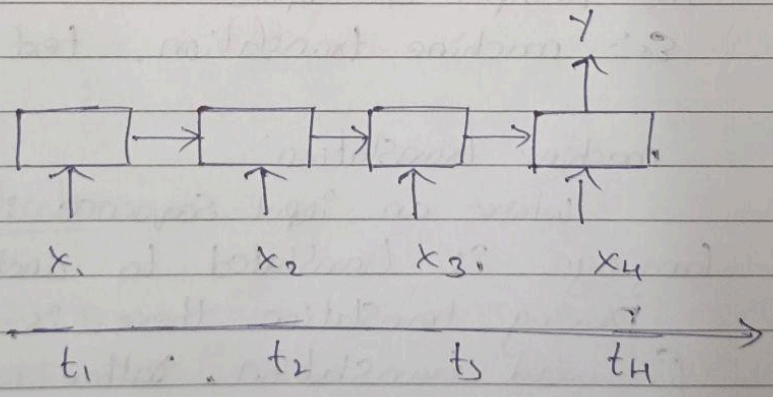
2. many to one RNN:

In this we have an input at each time step, but there are no corresponding outputs.

We get an output only produced after all the inputs are provided in the final step

Ex:- predicting price of stocks for next day given its historical prices.

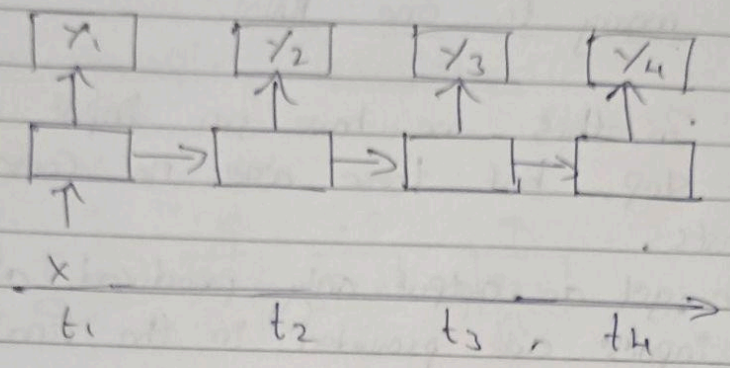
We can also use this type of RNN in classification problems like sentiment analysis that will take a sequence of text as input and produce a single output at end.



3. one to many RNN's

This type of architecture are real
* A single input produces multiple outputs
Example:- music synthesis

where single input is provided to the model & model generates a sequences of musical notes



Encoder - Decoder RNN:-

which are papular in transformers

In this case, a set of input are provided first going through encoding phase, then a set of outputs are generated out of the decoder

The number of output may not match the number of inputs

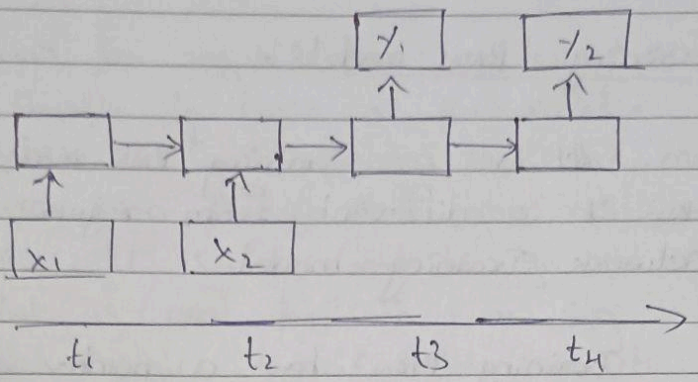
Ex:- machine translation, text summarization

machine translation:-

where an input sequence in a specific language is translated to another language

During translation, there is no word for word translation. Rather we have semantics is passed on to the new language

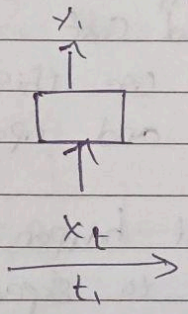
Similarly text summarization where inputs is summarized to smaller word count



One - to - one RNN's

This is a classical neural networks fidden states (core) makes RNN's special.

Ex: Predict Customer propensity.



RNN Application:

* Time series forecasting
* Text classification
* Topic modeling
* sentiment analysis
* named Entity recoginition
* Speech to text
* text to speech
* machine translation
* Question answering

* Text Summarization

Training RNN models:-

Let us see Training of RNN model and How it is different from regular neural network Training model.

Training RNN's have a number of similarities with the base training process

Training data will undergoes preparation and pre-processing before they are used for model training.

weights and biases will be initialized; Their is a similar forward propagation step in RNN. too After forward progation, we will compute prediction loss and cost

Then we will use this value to perform back-propagation and update the weights and biases.

Gradient descent happen until the observed accuracy improves to expected levels

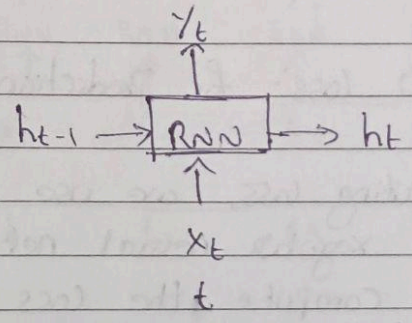
→ What is unique about RNN training?

we train by time steps and each time step may produce an output depending upon the type of architecture

* In addition we compute hidden state and propagate this to future time steps
* This also plays a role in loss function (or) loss computation and back propagation
* Finally, when there are multiple outputs are compute cumulative Error & perform back-propagation.

Forward propagation with RNN's

Similar to regular RNN we will use multiple sample to train an RNN model.
Each sample is a sequence that has multiple time steps. Even though there are multiple time steps. It is essential to go in same order that is used for each time step
* The model contains layers, nodes, weights, biases and activation functions. Similar to regular neural networks
* In addition, a hidden state is also computed and used for model training
* The hidden state from the previous step is used to compute the output of the current state as well as the next hidden state.
* Formula used for these computations.



x_t input that goes through an single RNN network. In addition, there is a hidden state H_{t-1} that come from the previous time step we produce h_t, the next hidden state. as well as Y_t, the current output
For the first time step, the previous hidden state would be initialized to zeroes.
To compute "h_t", we will use two set of weight "W_h" and "W_x" & we use one bias B_H

y → Dependent variable	b → Bias
x → Independent variable	h → Hidden State
w → Weight	f → Activation State

we first use the formula h_t Equal to.

$$h_t = f(W_h h_{t-1} + W_x x_t + b_h)$$

for $$y_t = f(W_{yht} + b_y)$$

h_t uses both the input x_t, previous state H_{t-1}. So both inputs influence the computation of Y_t unlike regular neural networks. which use "one" sets of weights and biasses. we have "three" sets of weights and two sets of biasses. Their values are initialized randomly & adjusted during the gradient descent process

Computing RNN loss: for Prediction.

For computing loss, we use similar loss functions like regular neural networks. we will (use) compute the loss for each time step and then aggregate the loss across time steps. to compute overall loss for the given samples. Whe we then compute the cost across a batch for multiple samples & use it for back propagation.

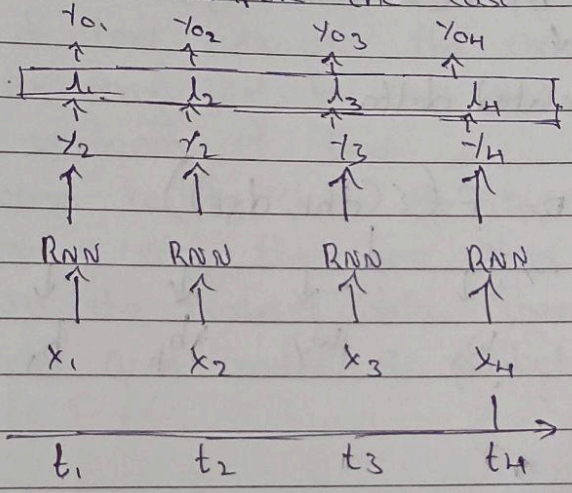
Example

We have an RNN with four time steps with four input and four outputs for each output, we also have corresponding actual target value that is available during training

We then compute the difference between the predicted and actual values of each time step to find individual loss.

We then aggregate the loss values across time steps. to get overall loss for all samples

Loss values accross samples in a batch are then used to compute the cost.



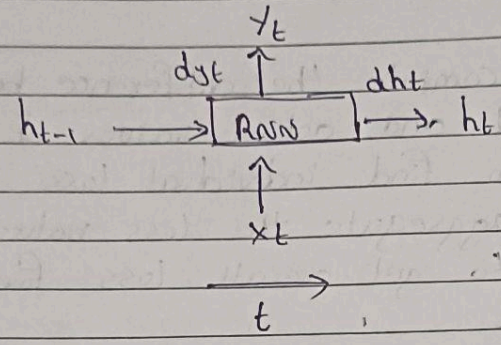
Back Propagation

previous we have known How to compute the cost function for a batch. we will cost function to compute derivatives (or) delta for both hidden state and the output for each step The derivative represents the adjustment that needs to be made to various weights & biases

we then compute cummulative (data) delta for all the time steps. we then apply this cumulative delta to all the weights in

in biases for the RNN model.

* we will continue the gradient descent process until the cost values reaches acceptable thresholds



$D = F(\Sigma(dh_t, dy_t))$

↓

Commulated delta.

$D = f(\Sigma(dh_t, dy_t))$

↓ ↓ ↓ ↓ ↓

w_h w_x w_y b_h b_y

apply to weight & biase so to adjust the weight & biase for RNN model.

we first compute the derivate (or) delta dh_t for output h_t based on the cost computed

we will similarly compute the delta for output dy_t

Then we will compute the cumulative delta as a function of all the deltas of individual

time steps.

We then apply this derivate to adjust the weight & biases for RNN model.

Once the weights & biases are adjusted we will continue with gradient descent for model training.

Prediction with RNN:-

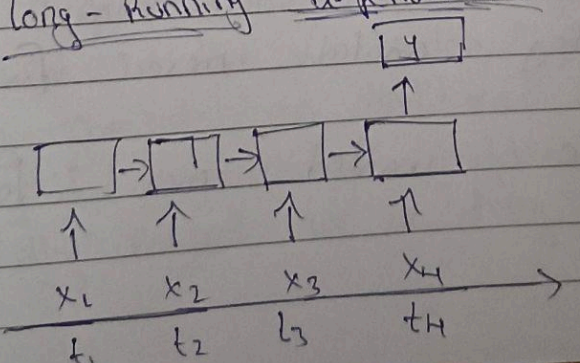
Prediction with RNN are pretty straight forward as similar to regular neural networks The input to the RNN during prediction, will vary depending upon the type of model. It may be the list of previous time steps in the case of time series prediction. (or) Sequence of text. like in the case of NLP. The output may just be the next time step or another sequence of text.

Whatever be the case, the input data will be prepared using the same steps are used to prepare the training data. Then input data is fed into RNN model & prediction are obtained

The vanishing gradient problem:-

To understant the problem of vanishing gradient we will consider an Example.

Long-Running Dependencies.



An RNN that takes an input the last four words in a sequence & predict the next word.

So, In this case, the input is x_1 to x_4 & there is a single output Y at the end

for example if we provide input as "they wanted more for."

And ~~we~~ want our model to predict themselves

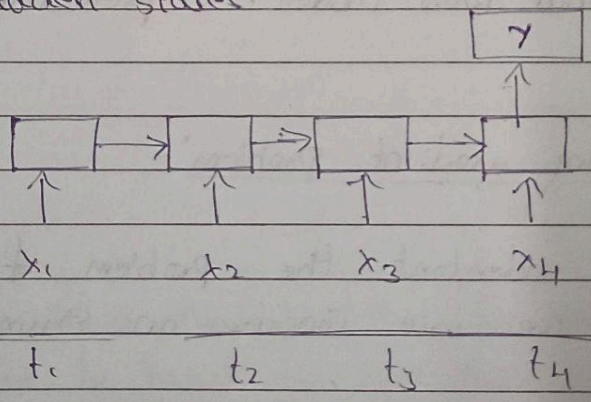
But their is a problem at here

Consider a similar input sequence, "he wanted more for"

In this case, prediction should be himself

Between these two sequences, the second, third and fourth words are the same. & output is dependent on first word.

In other word, the o/p Y at t_4 is dependent on the input at t_1. So there is a "long running dependency" that needs to be properly captured and propagated through the hidden states



They wanted more for themselves

He wanted more for himself

Vanishing Gradient in RNN:-

In Regular RNN, the output at any given time steps is influenced heavily by the input at that time step.

* As the hidden state propagates through the network, the weights of the inputs decay over time step.

For example, at T_2, the hidden state generated would be based on x_1, x_2 with x_1 having a higher weight

When we get to T_3, x_3 will have a higher weight than x_2 & x_2 will have a higher weight than x_1.

The weight of x_1 decays and looses significance This is okay in most sequence models where the next values is heavily influenced by nearest previous values, But in some cases we need to presist long running dependencies

The gated recurrent unit (GRU):-

GRU is an alternative sequence model architecture that helps prolong the memory of certain time steps. and take care long run dependency.

In a regular RNN, the hidden state from the previous layer is aggregated with inputs in the next layer to compute the new hidden state.

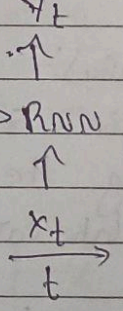
As we progress through this weights of a given input in the hidden state will decay

* A GRU compute an additional update gate to decide if the previous hidden state needs to be fully passed on or fully ignored

* The update gate produce a result of zero or one.
* If the value is zero then the current input is ignored. & previous hidden state become the current hidden state.
* If the value is one then the previous hidden state is fully ignored. and a new hidden state is computed. Entirely based on the input at that time step.

* There are some implementation of GRU that will use an additional reset gate also.
* The gates ensure propagation of selected states accross network without & help preserve certain features need for prediction down the line.

Computing Hidden State with GRU.



$$h_t = f(w_h h_{t-1} + w_x t_x + b_h)$$
$$y_t = f(w_y h_t + b_y)$$

we have original RNN formulas for hidden state & output shown here on the left.

In case of GRU we still have the same two input x_t, h_{t-1} similar & o/p y_t, h_t

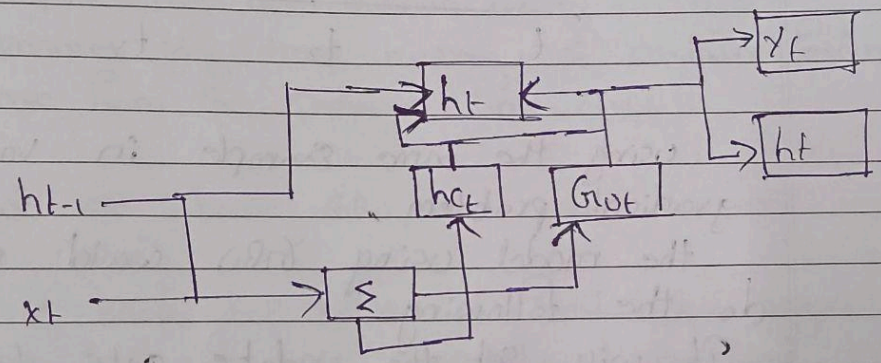
we first compute a candidate hidden state for this time step. using the same formula for hidden state as the regular RNN.

we then compute an update gate based on the previous hidden state & the current input. we use separate set of weight & biases for computing the update gate.

we finally use activation function to return one or zero.

now, we compute the hidden state for this time step. using formula.

$$h_t = G_{ut} h_{ct} + (1 - G_{ut}) h_{t-1}$$



$$h_{ct} = f(w_h h_{t-1} + w_x x_t + b_h)$$

$$G_{ut} = f(w_u [h_{t-1}, x_t] + b_u)$$

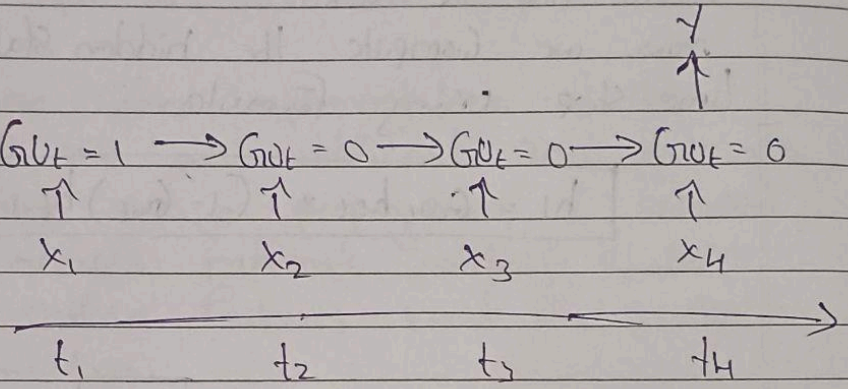
$$h_t = G_{ut} h_{ct} + (1 - G_{ut}) h_{t-1}$$

In this formula if the update gate G_{ut} is one then it returns only the candidate hidden state & completely ignores the previous hidden state. If the value is zero then it ingnores the candidate hidden state. & return the

Same value as the previous hidden state. So, the previous hidden state is entirely ~~from~~ consumed or ignored.

Now, we can proceed to compute the output using the same formula as the RNN.

GRU Gate Example :-



using the same example in vanishing gradient problem,

the model using GRU would probably do the following

It will set the update gate to one at t_1. Then it will set the update gate for T_2, T_3 & T_4 to zero.

Thus, the hidden state created at T_1 will propagate without decay to T_4 & fully influence the prediction Y

Long Short term memory (LSTM)

Another popular architecture in RNN's that address the vanishing gradient problem is long short term memory or LSTM (The weight of an input with hidden state)

* Regular RNN pass on the previous hidden state to the next time step, with incremental delay as we move across time steps
* LSTM uses multiple gates to decide if the previous hidden state needs to be passed on or ignored.
* unlike GRU, it's possible to use a previous hidden state as well as current import to compute the next hidden state.
* It ensures long term memory of certain features that are from the previous time steps.

When do we choose GRU and when do we choose LSTM?

* LSTM is know to better at predicting longer sequences but will have computational overhead.
* GRU is more efficient than LSTM. but can handle reasonably sized sequences well.

Let's look at the various gates computed for LSTM. we start of with the construct and formulate as GRU

we will compute the candidate hidden state and an update gate is LSTM also with the same formula.

$$hc_t = f(w_h h_{t-1} + w_x x_t + b_h)$$

$$GU_t = f(W_u[h_{t-1}, x_t] + b_u)$$

The activation functions may be different
In addition will compute a further gate.
It's formula is similar to the update gate.

$$GF_t = f(w_F [h_{t-1}, x_t] + b_F)$$

Except that it has its own weights and biases.
we also compute another gate called output gate. again with its own weights & biases

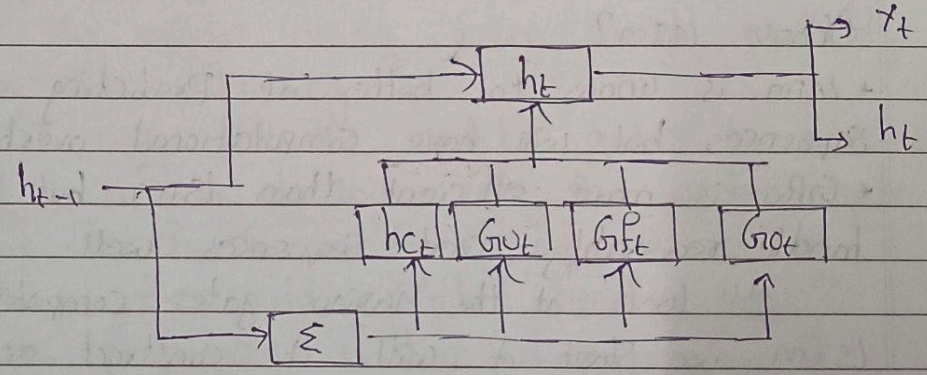
$$GO_t = f(w_O [h_{t-1}, x_t] + b_O)$$

finally we compute hidden state. through each time step using all these gates.

$$h_t = G_{Ot}(G_{Ut} h_{Ct} + G_{Ft} h_{t-1})$$

The update gate is used to (indistinct) the candidate hidden state.

Computing Hidden State with LSTM.



The further (indistinct) gates is used to filter the previous hidden state.
The output gate is then used to (indistinct) a certain output to produce final hidden state.

Bi-Directional RNN's (BRNNs)

BRNN's are another kind of RNN architecture also called Bi-directional RNN's. using example we will try to understand what is BRNN and it's mechanics.

Consider the problem of predicting the missing word in a sentence
In these case we have sequences.

He prepared an application
and we need to find the word in between which is 'an'

Let's look at another example where previous sequence is "he prepared" and the next word is meal.

He prepared a meal

And we need to predict the missing word which is "a"

As we can see, the previous sequence is the same & the predicted word is dependent upon the future word or sequences. Sometimes it's dependent on both the previous and next sequences.

The architecture we have seen so far only predict based on the past sequences. So, as we see a time step in a sequence is dependent upon both the previous time steps and future time steps.

we need to be able to make a model both of them to build RNN.

This is solved by "bidirectional RNN's" which passes hidden states in both direction.

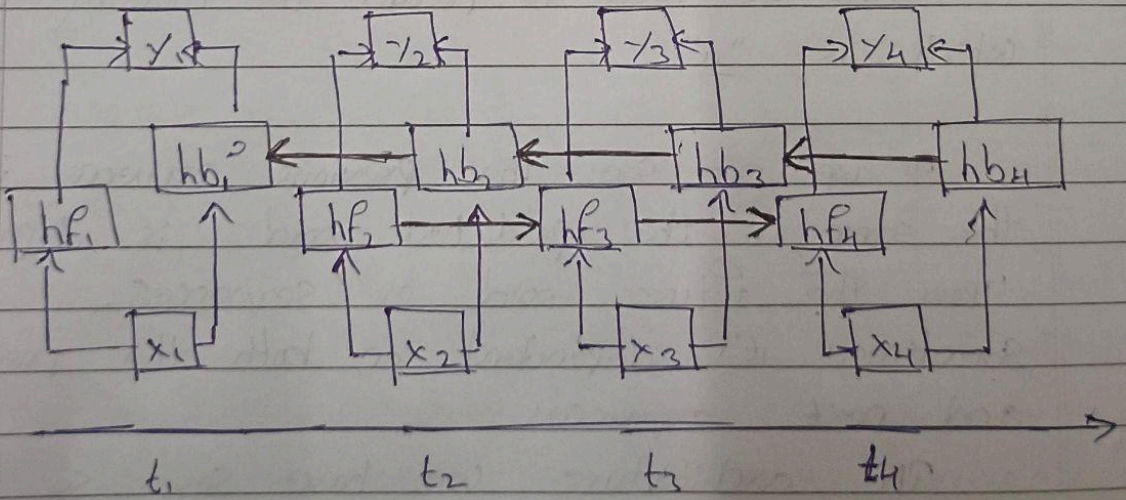
A popular use case for bidirectional RNN is a "named entity recognition".

where we try to identify an entity based on the past and future sequences.

Similarly, in "speech recognition", we will use past and future utterances to predict the current utterance (ಉಚ್ಚಾರ, ಹೇಳಿ) (a spoken word or statement or vocal sound)

"Grammar Checkers" also use a similar model to predict the right word to use in a given context.

How ~~By~~ BRNN works:-



we again have multiple time steps T_1 to T_4 and we have input x_1 to x_4 Then there are output y_1 to y_4. Similar to regular RNN

we first compute a forward hidden state hf_1. The forward hidden state is the same as what we have seen so far in other unidirectional models, now we will also compute a backward hidden state in additional to forward hidden state. This is again computated at each time step.

The forward hidden state are then passed The hf_1 is used to compute the next hidden state hF_2 and so on.

The backward hidden state is passed backwards So, we start from hb_4 to and then pass it on to previous time step to compute hb_3 and so on. Then we compute Y at each time step. using both forward and hidden states. This ensure that the feature that are before and after current time step. are taken into account for predicting the output. at each χ time step.