C05M05S02- Variants of RNNs

Deep Learning & Neural Networks - Variants of RNNs

Introduction

Some variants of the original RNN architecture.

First, a novel RNN architecture - the **Bidirectional RNN** which is capable of reading sequences in the 'reverse order' as well and has proven to boost performance significantly.

Then two important cutting-edge variants of the RNN which have made it possible to train large networks on real datasets. Although RNNs are capable of solving a variety of sequence problems, their architecture itself is their biggest enemy. Problems of exploding and vanishing gradients occur during the training of RNNs. This problem is solved by two popular **gated RNN architectures** - the **Long, Short Term Memory (LSTM)** and the **Gated Recurrent Unit (GRU).**

In this session:

- Bidirectional RNNs
- LSTMs and GRUs

Prerequisites

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

Bidirectional RNNs

Various types of tasks that can be solved using RNNs - POS tagging, sentiment classification, machine translation, speech recognition, video classification, etc.

There is a fundamental difference between these tasks - in some tasks, the entire sequence is available to the network before it starts making predictions; in some other tasks network has to make predictions continuously as new inputs come in.

To **assign a sentiment score** to a piece of text (say a customer review), network can see entire review text before assigning them a score. In tasks such as **predicting the next word** given previous few typed words, network does not have access to words in the future time steps while predicting next word.

These **two types of tasks** are called **offline and online sequence processing** respectively.

There is a neat trick with **offline tasks** - since network has access to the entire sequence before making predictions, why not use this task to make the network 'look at the future elements in the sequence' while training, hoping that this will make the network learn better?

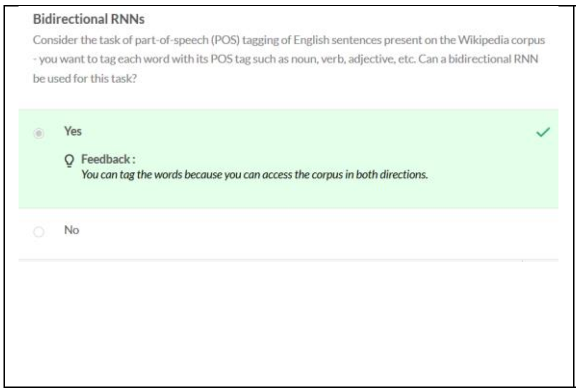
This idea is exploited in **bidirectional RNNs**.

In the following lecture, professor Raghavan explains the two types of tasks and how bidirectional RNNs boost the performance of offline processing tasks.

Thus, there are two types of sequences:

1. **Online sequence**: No access to the entire sequence before we start processing it. The network has to make predictions as it sees each input coming in.
2. **Offline sequence**: The entire sequence is available before we start processing it.

A bidirectional RNN can only be applied to **offline sequences**.



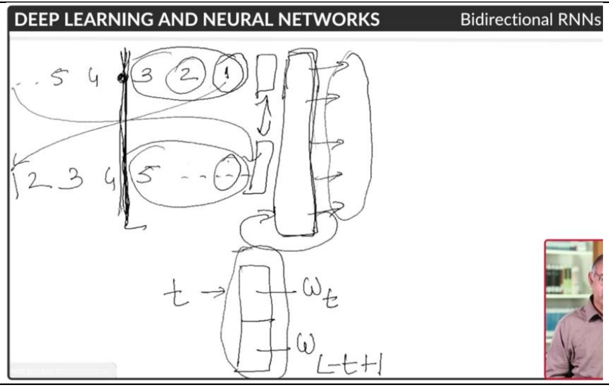
Bidirectional RNNs

Consider the task of part-of-speech (POS) tagging of English sentences present on the Wikipedia corpus - you want to tag each word with its POS tag such as noun, verb, adjective, etc. Can a bidirectional RNN be used for this task?

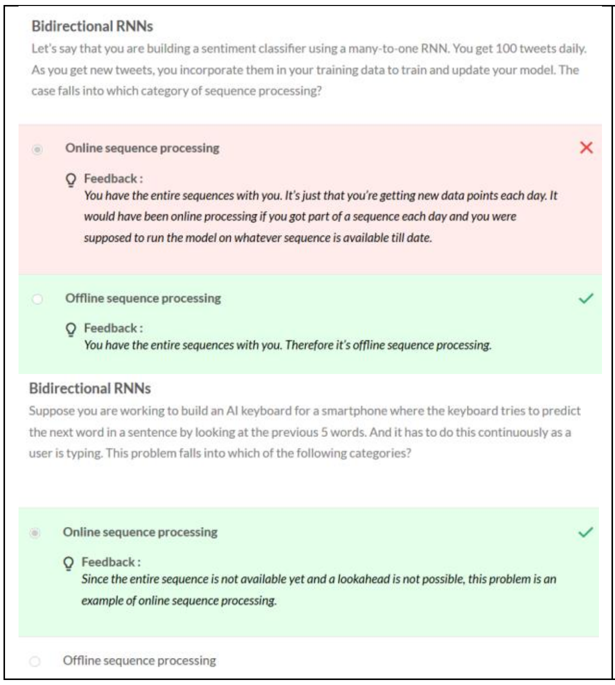
[✓ Yes] ✓

Feedback :
You can tag the words because you can access the corpus in both directions.

No



By using bidirectional RNNs, it is almost certain that we will get better results. However, bidirectional RNNs take almost double time to train since number of parameters of network increase. Therefore, a tradeoff between training time and performance. Decision to use a bidirectional RNN depends on the computing resources and the performance aimed.



Bidirectional RNNs

Let's say that you are building a sentiment classifier using a many-to-one RNN. You get 100 tweets daily. As you get new tweets, you incorporate them in your training data to train and update your model. The case falls into which category of sequence processing?

● Online sequence processing ✗

Feedback :
You have the entire sequences with you. It's just that you're getting new data points each day. It would have been online processing if you got part of a sequence each day and you were supposed to run the model on whatever sequence is available till date.

○ Offline sequence processing ✓

Feedback :
You have the entire sequences with you. Therefore it's offline sequence processing.

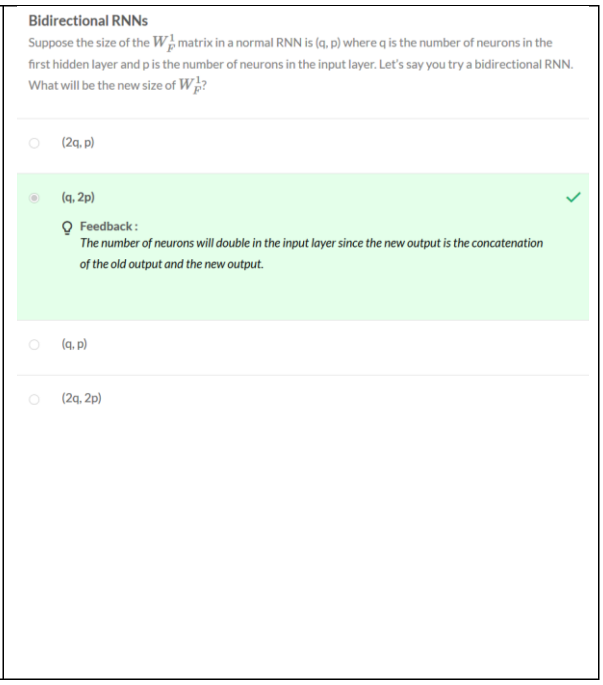
Bidirectional RNNs

Suppose you are working to build an AI keyboard for a smartphone where the keyboard tries to predict the next word in a sentence by looking at the previous 5 words. And it has to do this continuously as a user is typing. This problem falls into which of the following categories?

● Online sequence processing ✓

Feedback :
Since the entire sequence is not available yet and a lookahead is not possible, this problem is an example of online sequence processing.

○ Offline sequence processing



Bidirectional RNNs

Suppose the size of the W_P^1 matrix in a normal RNN is (q, p) where q is the number of neurons in the first hidden layer and p is the number of neurons in the input layer. Let's say you try a bidirectional RNN. What will be the new size of W_P^1?

○ (2q, p)

● (q, 2p) ✓

Feedback :
The number of neurons will double in the input layer since the new output is the concatenation of the old output and the new output.

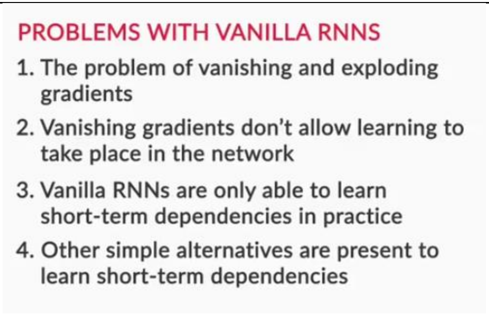
○ (q, p)

○ (2q, 2p)

Next session will implement bidirectional RNNs in Python. Next, we will study **LSTMs** and how they help get rid of vanishing gradients problem.

Long, Short-term Memory Networks

In the previous session, we had discussed the problem of **vanishing and exploding gradients** that RNNs face. Let’s revisit this problem once and then we'll be ready to study how LSTMs solve it.



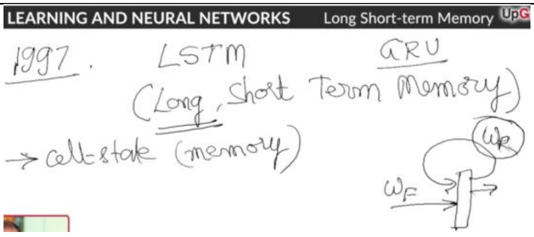
PROBLEMS WITH VANILLA RNNS

1. The problem of vanishing and exploding gradients
2. Vanishing gradients don’t allow learning to take place in the network
3. Vanilla RNNs are only able to learn short-term dependencies in practice
4. Other simple alternatives are present to learn short-term dependencies

In RNNs, main problem is **vanishing gradients** problem.

To solve it, many attempts have been made to tweak the vanilla RNNs so that gradients don’t die in long sequences. Most popular and successful is **long, short-term memory network**, or the **LSTM**. LSTMs have almost replaced vanilla RNNs.

Main drastic improvement that LSTMs have brought is because of a novel change in the **structure of a neuron** itself. In the case of LSTMs, the neurons are called cells, and an **LSTM cell** is different from a normal neuron in many ways.



One of the fundamental differences between an RNN and an LSTM is that an LSTM has an **explicit memory unit** which stores information relevant for learning some task. In standard RNN, only way network remembers past information is through updating the hidden states over time, but it does not have an explicit memory to store information.

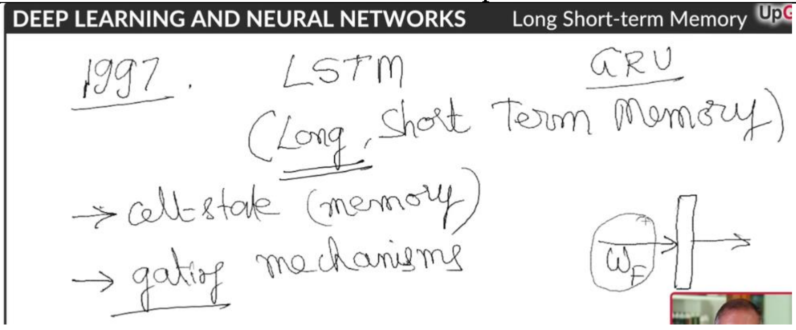
In LSTMs, memory units retain pieces of information even when sequences get really long. In the next section, we’ll look at another important feature of the LSTM cell - **gating mechanisms**.

Additional Reading

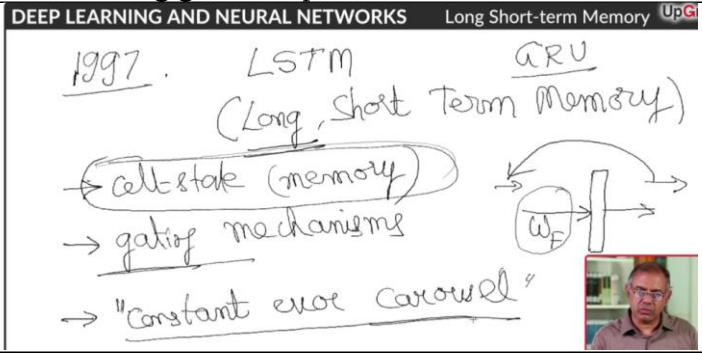
1. One of the first successful speech recognition solutions using LSTMs, Alex Graves et al

Characteristics of an LSTM Cell

Previous section briefly covered notion of 'cell state' or an **explicit memory** of an LSTM. This lecture takes a look at the next important characteristic of LSTMs.

 DEEP LEARNING AND NEURAL NETWORKS Long Short-term Memory 1997. LSTM GRU (Long, Short Term Memory) → cell-state (memory) → gating mechanisms W_F	The **gating mechanisms** allow modifying the state in certain ways. How exactly the gating mechanisms works is in next section, though the main idea is that gating mechanisms **regulate the information** that the network stores (and passes on to the next layer) or forgets.

Now, let's look at the third and final characteristic of an LSTM cell which helps **get rid of the vanishing gradient** problem.

 DEEP LEARNING AND NEURAL NETWORKS Long Short-term Memory 1997. LSTM GRU (Long, Short Term Memory) → cell-state (memory) → gating mechanisms W_F → "Constant error carousel"	Structure of an LSTM cell allows an LSTM network to have a smooth and uninterrupted flow of gradients while backpropagating. This flow is also called the **constant error carousel**. This third characteristic more or less a result of the first two characteristics and is the reason LSTMs are able to solve the problem of vanishing and exploding gradients.

LSTM is characterised by the following three main properties:

- The cells have an explicit 'memory'
- The gating mechanisms
- Constant error carousel

Next section covers the structure of an LSTM cell to better understand its three properties.

Structure of an LSTM Cell

Previous segments discussed the three main characteristics of an LSTM cell:

1. Presence of an explicit 'memory' called the cell state
2. Gating mechanisms
3. Constant error carousel

This lecture covers **LSTM cell** in detail.

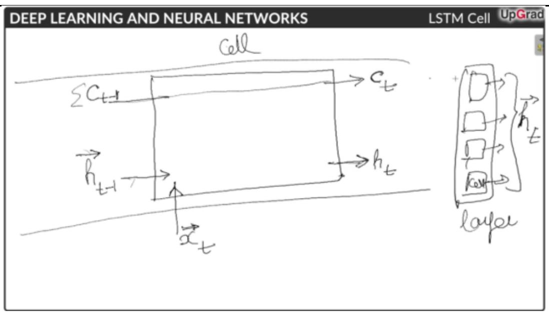
Before that, let's look at some standard notations used in the LSTM cell. There are a few changes in the notations. In the following sections, we have used 'h' and 'a' interchangeably to denote the activations coming out of a layer.

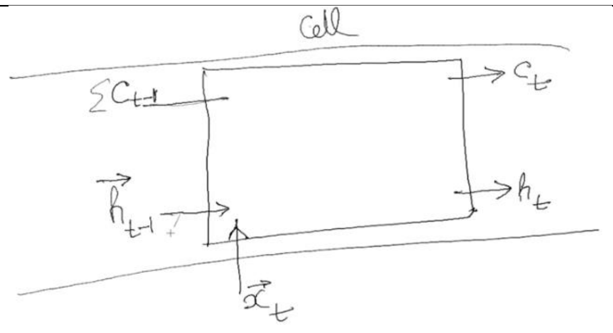
Now, let's look at the structure of an LSTM cell.

An LSTM cell is analogous to a neuron in an RNN - each LSTM layer contains multiple LSTM cells. The cell receives the following inputs:

- The output of the previous time step h_{t-1} (a vector)
- The current input x_t (a vector)
- The previous cell state c_{t-1} (usually a scalar)

We are referring to an LSTM cell in a certain layer 'l' and are not denoting the variables explicitly as c^l_t,h^l_t, etc. but simply c_t,h_t, etc. for simplicity.



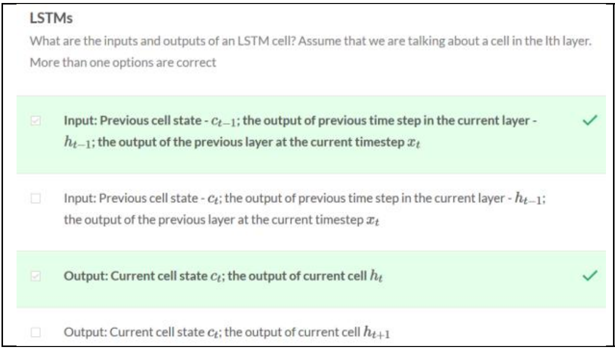


LSTM cell

The cell produces two outputs:

- The current cell state c_t (a scalar)
- The current state output h_t (a scalar)

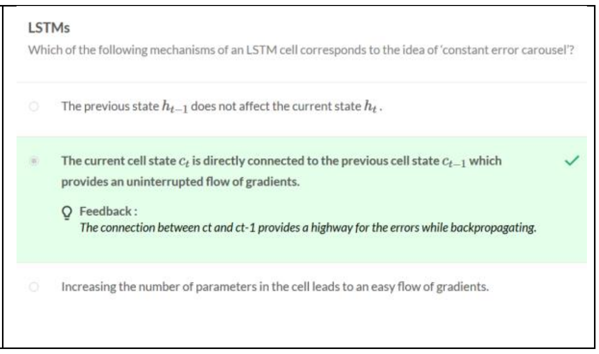
Each cell in LSTM layer will produce an output h_t which will then be combined to form a vector and fed to next LSTM layer.



LSTMs

What are the inputs and outputs of an LSTM cell? Assume that we are talking about a cell in the lth layer. More than one options are correct

- [✓] **Input: Previous cell state - c_{t-1}; the output of previous time step in the current layer - h_{t-1}; the output of the previous layer at the current timestep x_t** ✓
- [ ] Input: Previous cell state - c_t; the output of previous time step in the current layer - h_{t-1}; the output of the previous layer at the current timestep x_t
- [✓] **Output: Current cell state c_t; the output of current cell h_t** ✓
- [ ] Output: Current cell state c_t; the output of current cell h_{t+1}



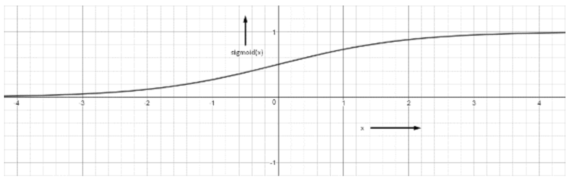
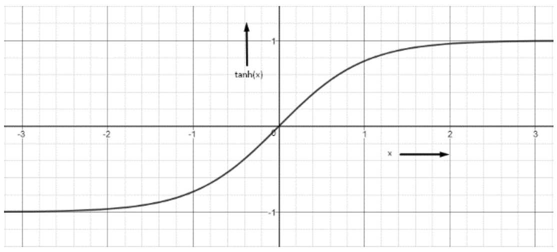
LSTMs

Which of the following mechanisms of an LSTM cell corresponds to the idea of 'constant error carousel'?

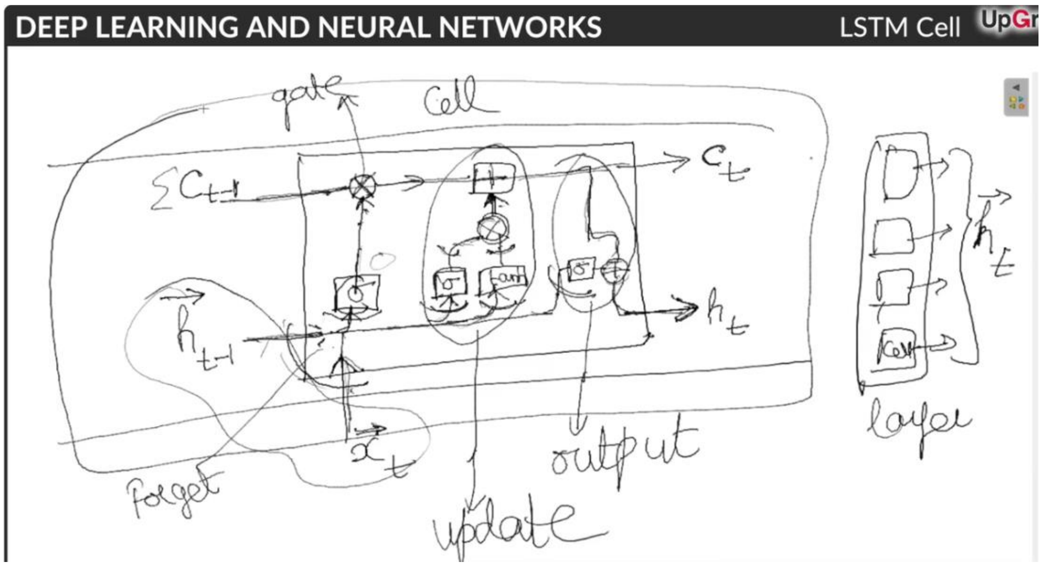
- ○ The previous state h_{t-1} does not affect the current state h_t .
- ● **The current cell state c_t is directly connected to the previous cell state c_{t-1} which provides an uninterrupted flow of gradients.** ✓

  Feedback :
  The connection between ct and ct-1 provides a highway for the errors while backpropagating.
- ○ Increasing the number of parameters in the cell leads to an easy flow of gradients.

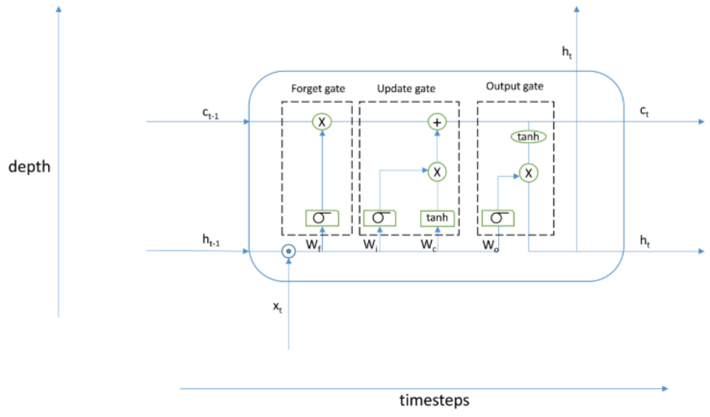
Before moving ahead to study LSTM cell in further detail, it will be useful to recall the common activation functions used in neural networks. Both these functions are shown below.

Sigmoid function outputs a value between 0 and 1 while the **tanh** outputs a value between -1 and +1.  **The sigmoid function**	The following graph shows the tanh function. Values of the function lie between -1 and +1.  **The tanh function**

Now, let's look at how the gating mechanism of the LSTM cell.



Structure of LSTM cell with 3 gating mechanisms - **forget gate**, **update gate** and **output gate.**



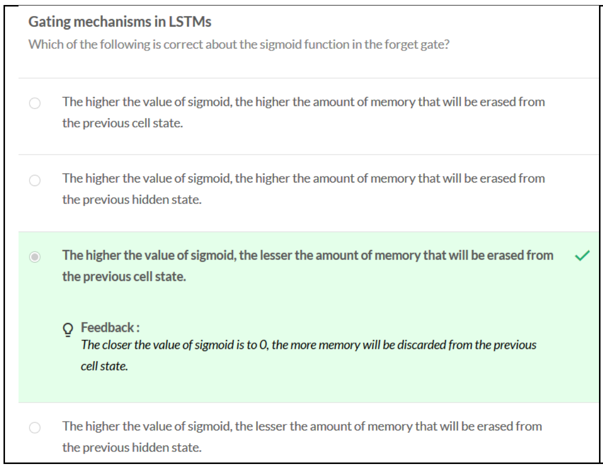
Full LSTM Cell

Let's understand the intuition of each gate with a specific example while working on a **video tagging problem** where we need to tag the action that takes place in each frame of the video.

- **Forget gate**: Controls how much information needs to be discarded from the previous cell state (c_{t-1}) depending on the new input. In the video tagging problem, whenever a new action takes place, this gate needs to decide how much information to retain from the previous frames. If the same action is happening over and over again, then very less information should be discarded. When the action changes, the forget gate 'forgets' a lot of information.
- **Update gate**: Makes an **update to previous cell state** by writing a new piece of information to it. In the video tagging problem, when the action changes, this gate will update the cell state with information relevant to the new action. In case action is same as previous frame, negligible information will be written to the cell state. If the scene changes drastically, the update will be drastic too.

The new cell state c_t is the cumulative result of the information discarded from c_{t-1} by the forget gate and the new information freshly updated to c_{t-1} by the update gate.

- **Output gate**: Controls how much information needs to be passed on to the next LSTM layer based on the current cell state.

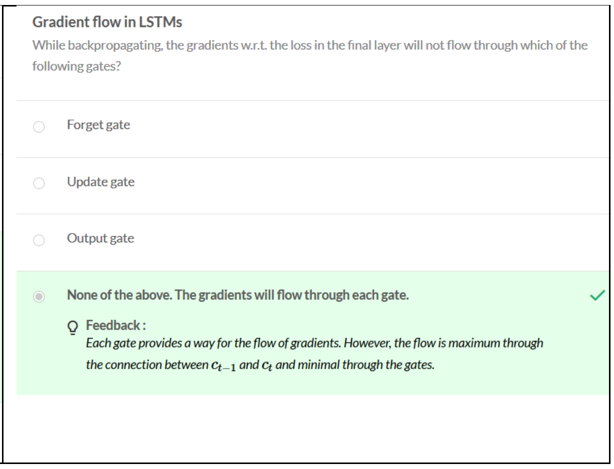


Gating mechanisms in LSTMs

Which of the following is correct about the sigmoid function in the forget gate?

- The higher the value of sigmoid, the higher the amount of memory that will be erased from the previous cell state.
- The higher the value of sigmoid, the higher the amount of memory that will be erased from the previous hidden state.
- [✓ The higher the value of sigmoid, the lesser the amount of memory that will be erased from the previous cell state.] ✓

  Feedback :
  The closer the value of sigmoid is to 0, the more memory will be discarded from the previous cell state.
- The higher the value of sigmoid, the lesser the amount of memory that will be erased from the previous hidden state.

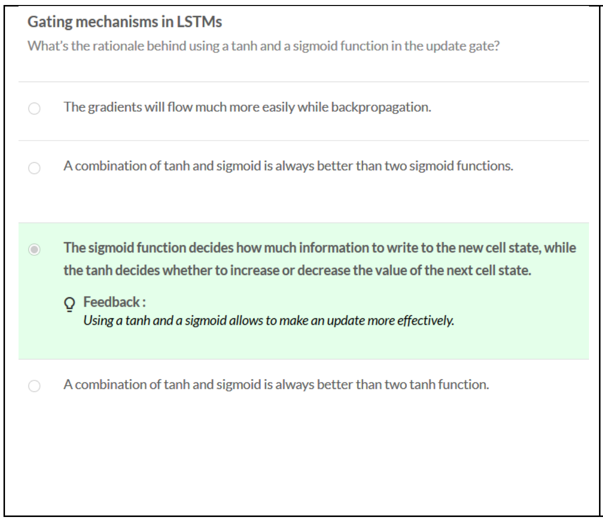


Gradient flow in LSTMs

While backpropagating, the gradients w.r.t. the loss in the final layer will not flow through which of the following gates?

- Forget gate
- Update gate
- Output gate
- [✓ None of the above. The gradients will flow through each gate.] ✓

  Feedback :
  Each gate provides a way for the flow of gradients. However, the flow is maximum through the connection between c_{t-1} and c_t and minimal through the gates.

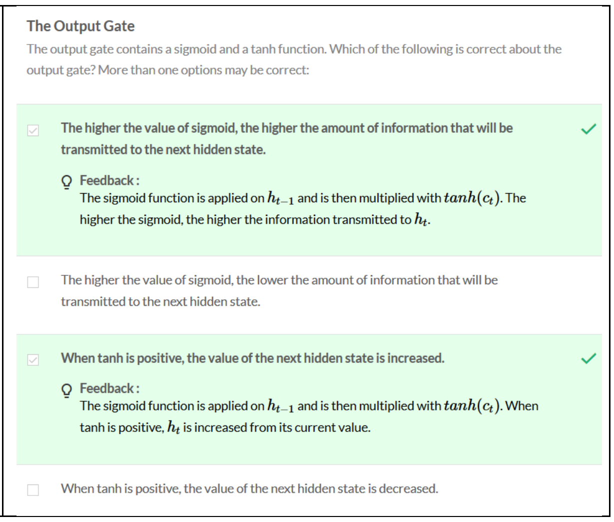


Gating mechanisms in LSTMs

What's the rationale behind using a tanh and a sigmoid function in the update gate?

- ○ The gradients will flow much more easily while backpropagation.
- ○ A combination of tanh and sigmoid is always better than two sigmoid functions.
- [✓ **The sigmoid function decides how much information to write to the new cell state, while the tanh decides whether to increase or decrease the value of the next cell state.**]

  Feedback :
  Using a tanh and a sigmoid allows to make an update more effectively.
- ○ A combination of tanh and sigmoid is always better than two tanh function.



The Output Gate

The output gate contains a sigmoid and a tanh function. Which of the following is correct about the output gate? More than one options may be correct:

- [✓ **The higher the value of sigmoid, the higher the amount of information that will be transmitted to the next hidden state.**] ✓

  Feedback :
  The sigmoid function is applied on h_{t-1} and is then multiplied with $tanh(c_t)$. The higher the sigmoid, the higher the information transmitted to h_t.
- ☐ The higher the value of sigmoid, the lower the amount of information that will be transmitted to the next hidden state.
- [✓ **When tanh is positive, the value of the next hidden state is increased.**] ✓

  Feedback :
  The sigmoid function is applied on h_{t-1} and is then multiplied with $tanh(c_t)$. When tanh is positive, h_t is increased from its current value.
- ☐ When tanh is positive, the value of the next hidden state is decreased.

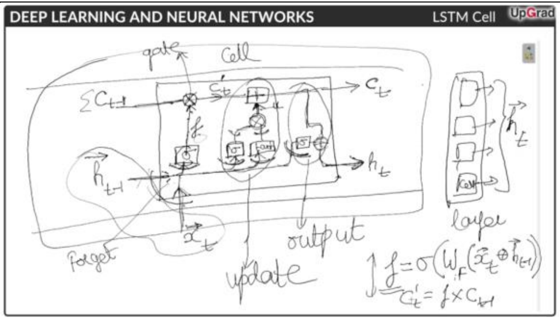
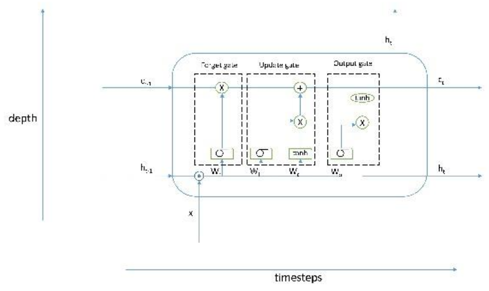
In the next section, we will look at the feedforward equations of an LSTM network.

LSTM Network: Feedforward Equations

After structure of an LSTM cell, it'll be easier to follow LSTM feedforward equations. Feedforward equations of a standard RNN network is

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

LSTM equation will also be written in same fashion, that is, using concatenated weight matrices and concatenated activations. Let's now look at the LSTM feedforward equations.

 DEEP LEARNING AND NEURAL NETWORKS LSTM Cell UpGrad gate Cell ΣC_{t-1} S_t' C_t f $\vec{h}_{t-1}$ h_t $\vec{x}_t$ forget update output layer h_t $f = \sigma(W_f(\vec{x}_t \oplus \vec{h}_{t-1}))$ $C_t' = f \times C_{t-1}$	Here is a detailed architecture of an LSTM cell.  depth timesteps Forget gate Update gate Output gate c_{t-1} c_t h_{t-1} h_t h_t x W_f W_i W_c W_o tanh tanh **LSTM Cell**

In feedforward, first the previous activations h_{t-1} and the current input x_t get concatenated (shown by the dot operator). The concatenated vector goes into each of the three gates. The 'x' denotes element-wise multiplication while the '+' denotes element-wise addition between two vectors/matrices. Output gate has another tanh function though it is *not* a gate (there are no weights involved in that operation, as shown in the figure). The feedforward equations of an LSTM are as follows:

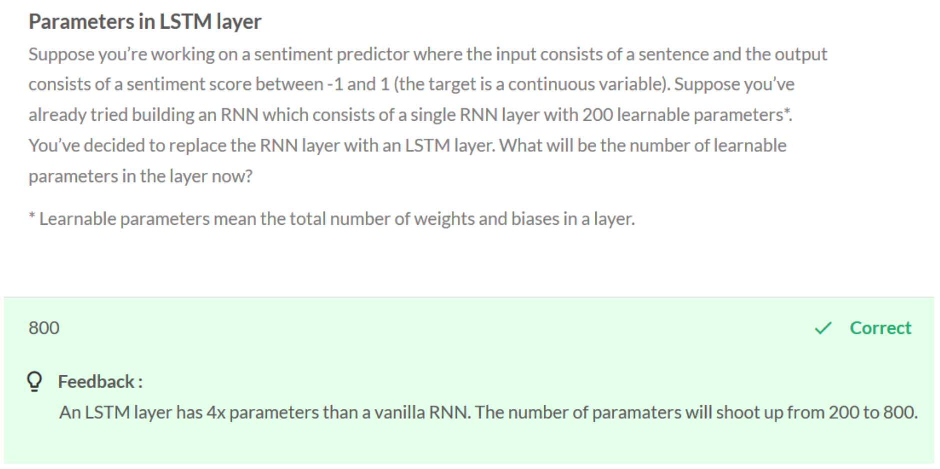
$$\begin{aligned}
f_t &= sigmoid(W_f \, [\, h_{t-1} \,,\, x_t \,] + b_f) \\
i_t &= sigmoid(W_i \, [\, h_{t-1} \,,\, x_t \,] + b_i) \\
c_t' &= tanh(W_c \, [\, h_{t-1} \,,\, x_t \,] + b_c) \\
c_t &= f_t \, c_{t-1} \; + \; i_t \, c_t' \\
o_t &= sigmoid(W_o \, [\, h_{t-1} \,,\, x_t \,] + b_o) \\
h_t &= o_t \, tanh(c_t)
\end{aligned}$$

In the RNN cell, exactly one matrix W (concatenation of the feedforward and recurrent matrix). In case of an LSTM cell, four weight matrices: W_f,W_i,W_c,W_o. Each of these is a concatenation of feedforward and recurrent weight matrices. We can write the weights of an LSTM as:

$$\begin{aligned}
W_f \; &= [\, W_{Ff} \mid W_{Rf} \,] \\
W_i \; &= [\, W_{Fi} \mid W_{Ri} \,] \\
W_c \; &= [\, W_{Fc} \mid W_{Rc} \,] \\
W_o \; &= [\, W_{Fo} \mid W_{Ro} \,]
\end{aligned}$$

An LSTM layer has **4x parameters** as compared to a normal RNN layer. The increased number of parameters leads to increased computational costs. For the same reason, an LSTM is more likely to overfit the training data than a normal RNN.

Most real-world sequence problems such as speech recognition, translation, video processing are complex enough to need LSTMs.



Parameters in LSTM layer

Suppose you're working on a sentiment predictor where the input consists of a sentence and the output consists of a sentiment score between -1 and 1 (the target is a continuous variable). Suppose you've already tried building an RNN which consists of a single RNN layer with 200 learnable parameters*. You've decided to replace the RNN layer with an LSTM layer. What will be the number of learnable parameters in the layer now?

* Learnable parameters mean the total number of weights and biases in a layer.

800 ✓ Correct

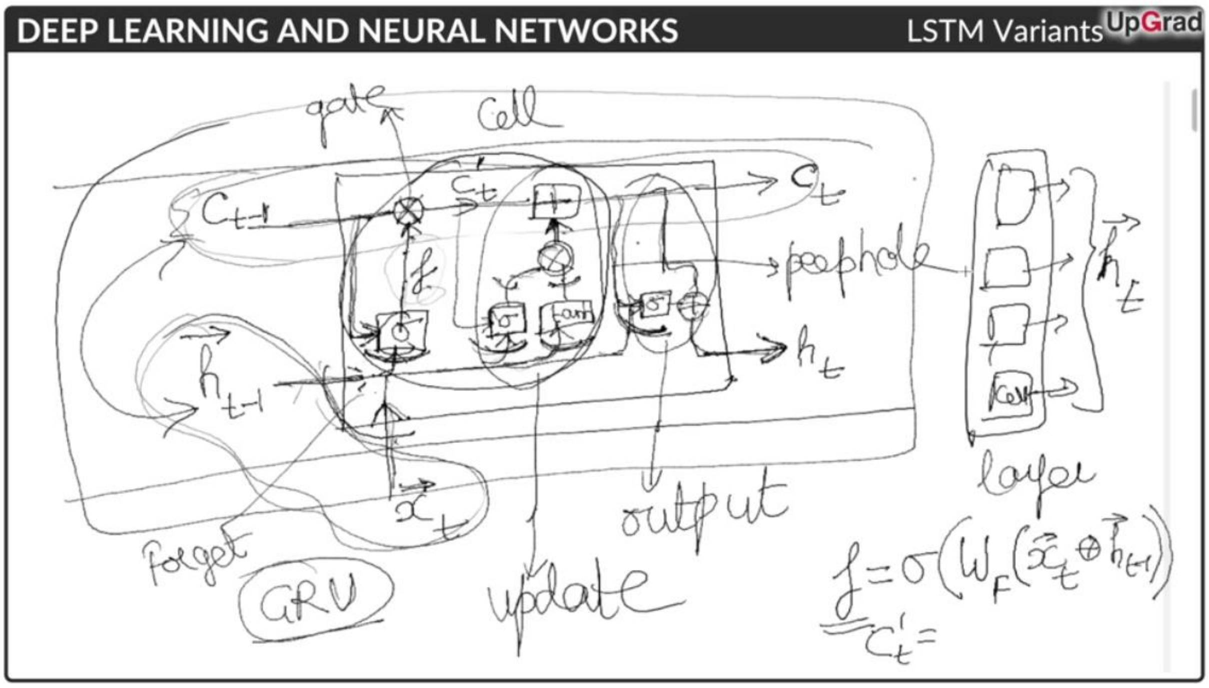
Feedback :

An LSTM layer has 4x parameters than a vanilla RNN. The number of paramaters will shoot up from 200 to 800.

Next section will look at a couple of other variants of LSTM cell.

GRUs and Other Variants

Keeping in mind the computational expenses and the problem of overfitting, researchers have tried to come up with alternate structures of the LSTM cell. The most popular one of these alternatives is the **gated recurrent unit (GRU)** which was introduced in late 2014. Let’s look at these alternative structures in a bit more detail.



A detailed discussion on GRUs is beyond the scope. However, implement GRUs in Keras in the next session (to build a POS tagger).

LSTMs and GRUs have almost completely replaced the standard RNNs in practice because they’re more effective and faster to train than vanilla RNNs (despite the larger number of parameters). In most sequence problems, the problem of vanishing gradient is far severe than training time.

The next segment contains the summary of this session.

Summary

Some commonly used variants of RNNs.

First, **bidirectional RNNs**. In case of offline sequences, we can reverse the order of a sequence. We can use bidirectional RNNs to feed sequences in regular as well as in the reverse order. This gives better result in most cases.

Long short-term memory networks or LSTMs has three features of an LSTM cell:

- Presence of an explicit memory
- Gating mechanisms
- Constant error carousel

Structure of an LSTM cell and its feed forward equations. Number of parameters in an LSTM layer are **4x** the number of parameters in a standard RNN.

Finally, we briefly looked at some other variants of LSTM such as **GRUs** and LSTMs with peephole connections. The number of parameters in a GRU layer are **3x** the number of parameters in a standard RNN layer.

Next is graded questions.