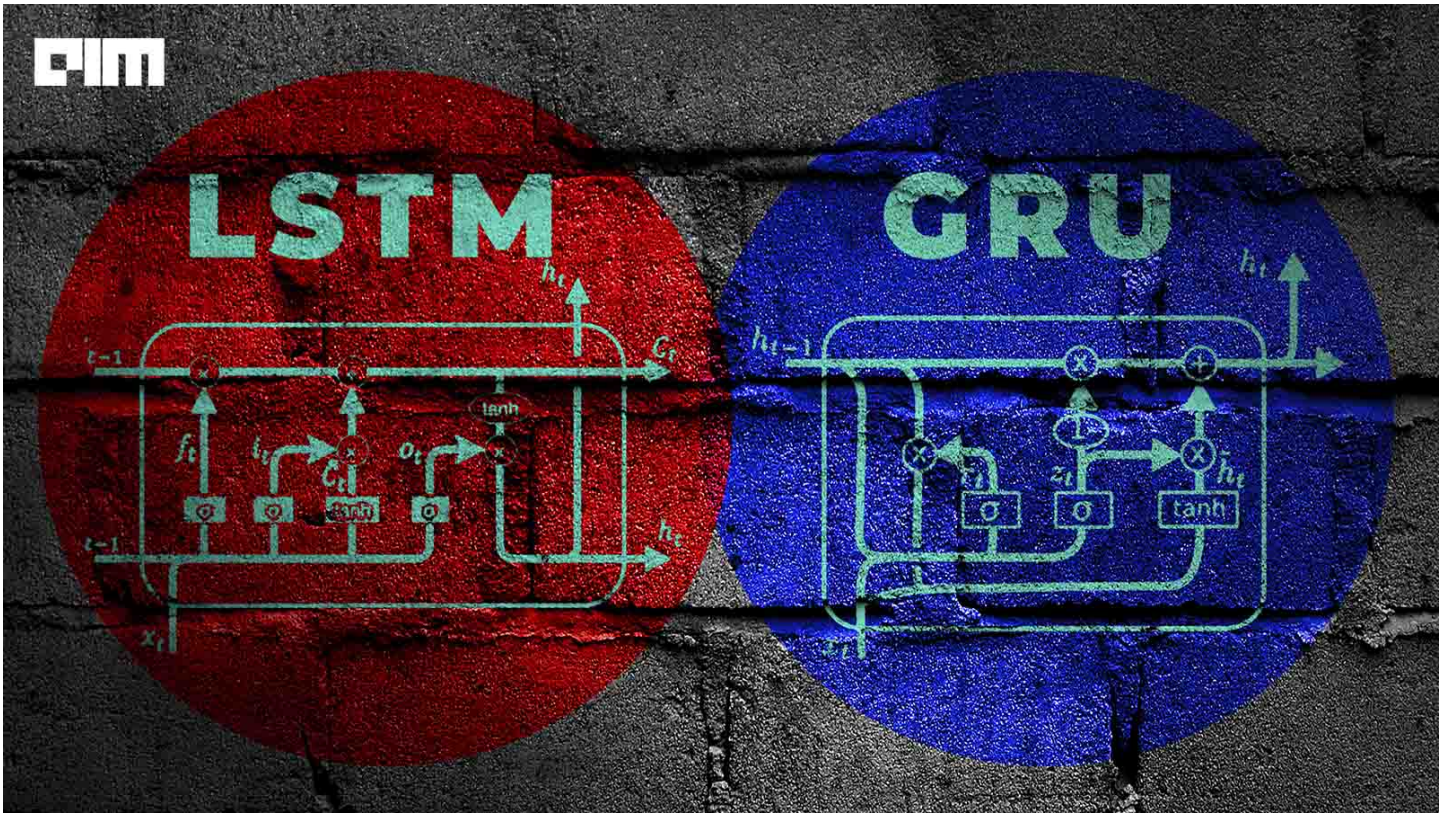
PUBLISHED ON AUGUST 28, 2021
IN DEVELOPERS CORNER
(HTTPS://ANALYTICSINDIAMAG.COM/CATEGORY/DEVELOPERS_CORNER/)

LSTM Vs GRU in Recurrent Neural Network: A Comparative Study

Long Short Term Memory in short LSTM is a special kind of RNN capable of learning long term sequences. They were introduced by Schmidhuber and Hochreiter in 1997. It is explicitly designed to avoid long term dependency problems. Remembering the long sequences for a long period of time is its way of working.

By Vijaysinh Lendave(https://analyticsindiamag.com/author/vijaysinh-lendaveanalyticsindiamag-com/)



A recurrent neural network (https://analyticsindiamag.com/implementing-a-recurrent-neural-network-rnn-from-scratch/) is a type of ANN (https://analyticsindiamag.com/ann-with-linear-regression/) that is used when users want to perform predictive operations on sequential or time-series (https://analyticsindiamag.com/guide-to-implementing-time-series-analysis-predicting-bitcoin-price-with-rnn/) based data. These Deep learning layers are commonly used for ordinal or temporal problems such as Natural Language Processing (https://analyticsindiamag.com/how-to-identify-entities-in-nlp/), Neural Machine Translation, automated image captioning tasks (https://analyticsindiamag.com/hands-on-guide-to-effective-image-captioning-using-attention-mechanism/) and likewise. Today's modern voice assistance devices such as Google Assistance, Alexa, Siri are incorporated with these layers to fulfil hassle-free experiences for users.

What is the difference between RNN and CNN?

The main difference between the RNN and CNN is that RNN is incorporated with memory to take any information from prior inputs to influence the Current input and output. Training methods for this network are the same. While traditional neural networks assume that both input and output are independent of each other, RNN gives the output based on previous input and its context.

Another distinguishing parameter is that RNN shares parameters across each layer of the network. While feedforward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network.

Below at left one is a representation of standard RNN and the right one is a representation of Feed-Forward Network.

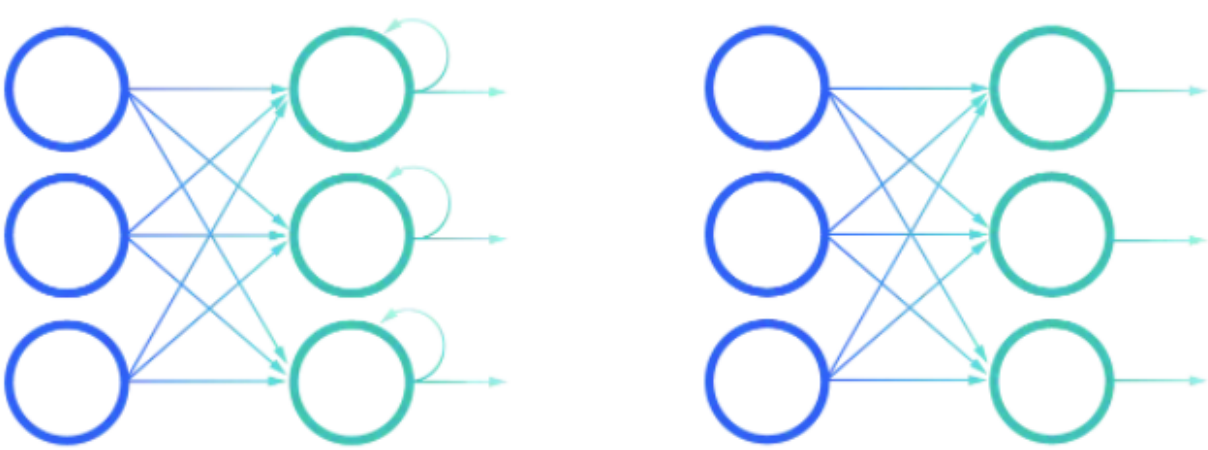
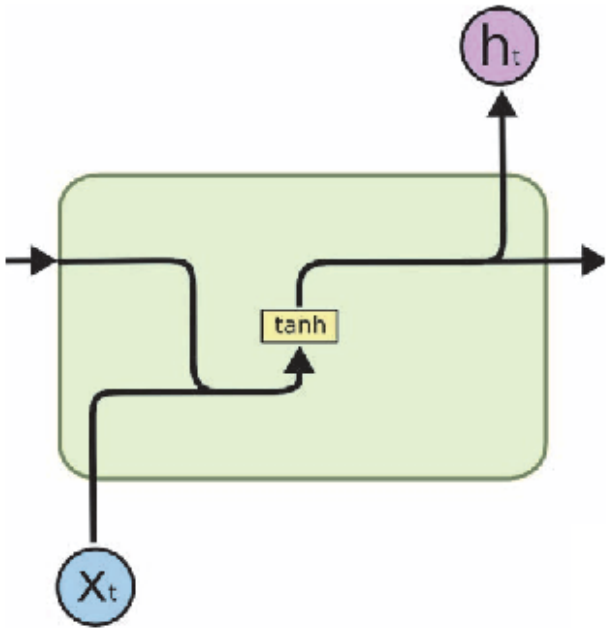


Image Source (https://www.ibm.com/cloud/learn/recurrent-neural-networks)

In the last few years for RNN's, there has been an incredible success in a variety of problems such as speech recognition, language modelling, translation, image captioning and list goes on.

Let's take a look at a single unit of RNN architecture. Where it takes input from the previous step and current state Xt and incorporated with Tanh as an activation function, here we can explicitly change the activation function.



Sometimes we only need to look at recent information to perform a present task. But this is not the case we face all the time. When a standard RNN network is exposed to long sequences or phrases it tends to lose the information because it can not store the long sequences and as the methodology is concerned it focuses only on the latest information available at the node. This problem is commonly referred to as Vanishing gradients.

In RNN to train networks, we backpropagate through time and at each time step or loop operation gradient is being calculated and the gradient is used to update the weights in the networks. Now if the effect of the previous sequence on the layer is small then the relative gradient is calculated small. Then if the gradient of the previous layer is smaller then this makes weights to be assigned to the context smaller and this effect is observed when we deal with longer sequences. Due to this network does not learn the effect of earlier inputs and thus causing the short term memory problem.

To overcome this problem specialized versions of RNN are created like LSTM, GRU, Time Distributed layer, ConvLSTM2D layer.

- Gated Recurrent Unit – What Is It And How To Learn (https://analyticsindiamag.com/gated-recurrent-unit-what-is-it-and-how-to-learn/)
- How To Code Your First LSTM Network In Keras (https://analyticsindiamag.com/how-to-code-your-first-lstm-network-in-keras/)
- How To Implement LSTM RNN Network For Sentiment Analysis (https://analyticsindiamag.com/how-to-implement-lstm-rnn-network-for-sentiment-analysis/)
- Salesforce Launches AI-Economist: A Complete Guide With Python Codes (https://analyticsindiamag.com/salesforce-launches-ai-economist-a-complete-guide-with-python-codes/)
- Bill Gates Says No To Sharing Covid Vaccine Recipe With India (https://analyticsindiamag.com/bill-gates-says-no-to-sharing-covid-vaccine-recipe-with-india/)

Working of LSTM

What is Long Short Term Memory or LSTM?

Long Short Term Memory in short LSTM is a special kind of RNN capable of learning long term sequences. They were introduced by Schmidhuber *and Hochreiter* in 1997. It is explicitly designed to avoid long term dependency problems. Remembering the long sequences for a long period of time is its way of working.

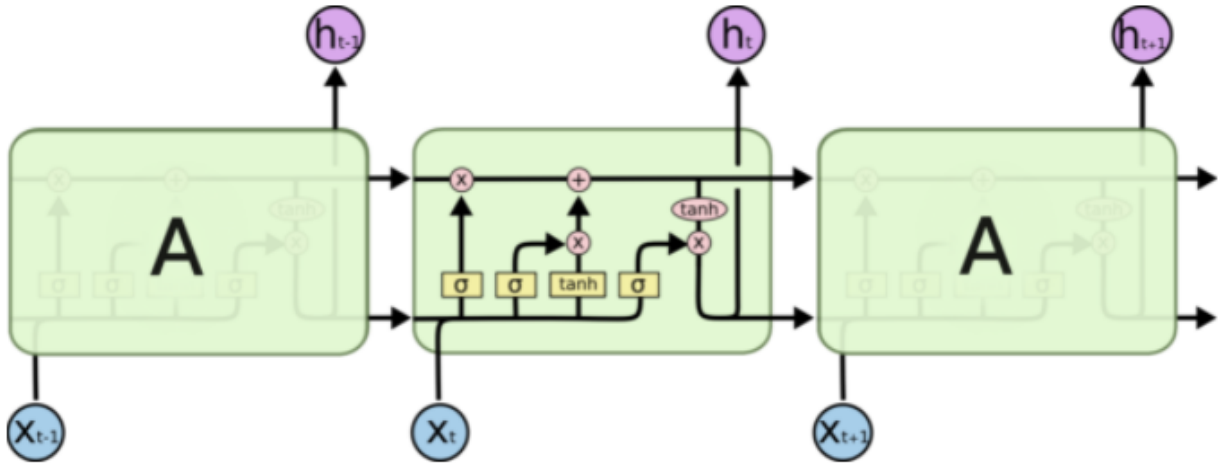


Image Source (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

The popularity of LSTM is due to the Getting mechanism involved with each LSTM cell. In a normal RNN cell, the input at the time stamp and hidden state from the previous time step is passed through the activation layer to obtain a new state. Whereas in LSTM the process is slightly complex, as you can see in the above architecture at each time it takes input from three different states like the current input state, the short term memory from the previous cell and lastly the long term memory.

These cells use the gates to regulate the information to be kept or discarded at loop operation before passing on the long term and short term information to the next cell. We can imagine these gates as Filters that remove unwanted selected and irrelevant information. There are a total of three gates that LSTM uses as *Input Gate, Forget Gate,* and *Output Gate*.

Input Gate

The input gate decides what information will be stored in long term memory. It only works with the information from the current input and short term memory from the previous step. At this gate, it filters out the information from variables that are not useful.

Forget Gate

The forget decides which information from long term memory be kept or discarded and this is done by multiplying the incoming long term memory by a forget vector generated by the current input and incoming short memory.

Output Gate

The output gate will take the current input, the previous short term memory and newly computed long term memory to produce new short term memory which will be passed on to the cell in the next time step. The output of the current time step can also be drawn from this hidden state.

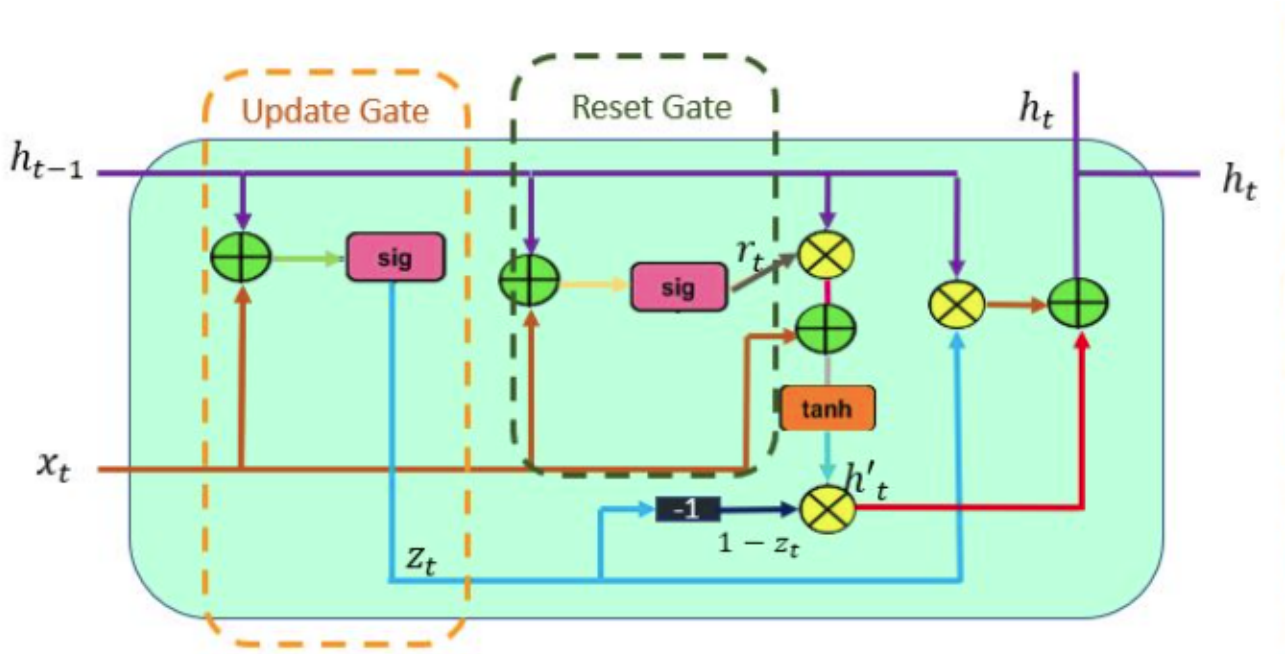
So this is all about the mechanism of LSTM to realise this with practical implementation. Here (https://colab.research.google.com/drive/1-c36k29ptuMUXUrQzH5SrQ-24t5pFZZL?usp=sharing) I have demonstrated the

LSTM use case in which you can check input and output sequences with their shape.

Working of GRU

What is Gated Recurrent Unit or GRU?

The workflow of the Gated Recurrent Unit, in short GRU, is the same as the RNN but the difference is in the operation and gates associated with each GRU unit. To solve the problem faced by standard RNN, GRU incorporates the two gate operating mechanisms called Update gate and Reset gate.



Update gate

The update gate is responsible for determining the amount of previous information that needs to pass along the next state. This is really powerful because the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient.

Reset gate

The reset gate is used from the model to decide how much of the past information is needed to neglect; in short, it decides whether the previous cell state is important or not.

First, the reset gate comes into action it stores relevant information from the past time step into new memory content. Then it multiplies the input vector and hidden state with their weights. Next, it calculates element-wise multiplication between the reset gate and previously hidden state multiple. After summing up the above steps the non-linear activation function is applied and the next sequence is generated.

This is all about the operation of GRU, the practical examples are included in the notebooks.

LSTM Vs GRU

Now we have seen the operation of both the layers to combat the problem of vanishing gradient. So you might wonder which one is to use? As GRU is relatively approaching its tradeoffs haven't been discussed yet.

According to empirical evaluation, there is not a clear winner. The basic idea of using a getting mechanism to learn long term dependencies is the same as in LSTM.

What is the difference between GRU & LSTM?
The few differencing points are as follows:
The GRU has two gates, LSTM has three gates
GRU does not possess any internal memory, they don't have an output gate that is present in LSTM
In LSTM the input gate and target gate are coupled by an update gate and in GRU reset gate is applied directly to the previous hidden state. In LSTM the responsibility of reset gate is taken by the two gates i.e., input and target.

Conclusion

Through this article, we have understood the basic difference between the RNN, LSTM and GRU units. From working of both layers i.e., LSTM and GRU, GRU uses less training parameter and therefore uses less memory and executes faster than LSTM whereas LSTM is more accurate on a larger dataset. One can choose LSTM if you are dealing with large sequences and accuracy is concerned, GRU is used when you have less memory consumption and want faster results.

More Great AIM Stories

Top Cybersecurity Budgets Around The World (https://analyticsindiamag.com/top-cybersecurity-budgets-around-the-world/)

How Did Airbnb Achieve Model Metric Consistency At Scale (https://analyticsindiamag.com/how-did-airbnb-achieve-model-metric-consistency-at-scale/)

Andrew Ng To Kickstart A New Generation Of AI (https://analyticsindiamag.com/andrew-ng-to-kickstart-a-new-generation-of-ai/)

What Are DQN Reinforcement Learning Models (https://analyticsindiamag.com/what-are-dqn-reinforcement-learning-models/)

Tech Behind Locobuzz's COVID Relief Platform (https://analyticsindiamag.com/tech-behind-locobuzzs-covid-relief-platform/)

Virtual Round Table Discussion On Analytics As A Driver Of Talent Transformation – 30th June (https://analyticsindiamag.com/register-for-a-virtual-round-table-discussion-on-analytics-as-a-driver-of-talent-transformation/)

(https://analyticsindiamag.com/author/vijaysinh-lendaveanalyticsindiamag-com/)

Vijaysinh is an enthusiast in machine learning and deep learning. He is skilled in ML algorithms, data manipulation, handling and visualization, model building.

Our Upcoming Events

Masterclass, Virtual

AI Application in Semiconductor Design

16th Jun

Register (https://attendee.gotowebinar.com/register/8244307342033124 00)

Conference, in-person (Bangalore)

MachineCon 2022

24th Jun

Register (https://machinecon.analyticsindiamag.com/tickets/)

Workshop, Virtual

Advanced SYCL Concepts for Heterogenous Computing

24th Jun

Register (https://attendee.gotowebinar.com/register/8182474697530800652)

Conference, Virtual

Deep Learning DevCon 2022

30th Jul

Register (https://dldc.adasci.org/get-the-tickets/)

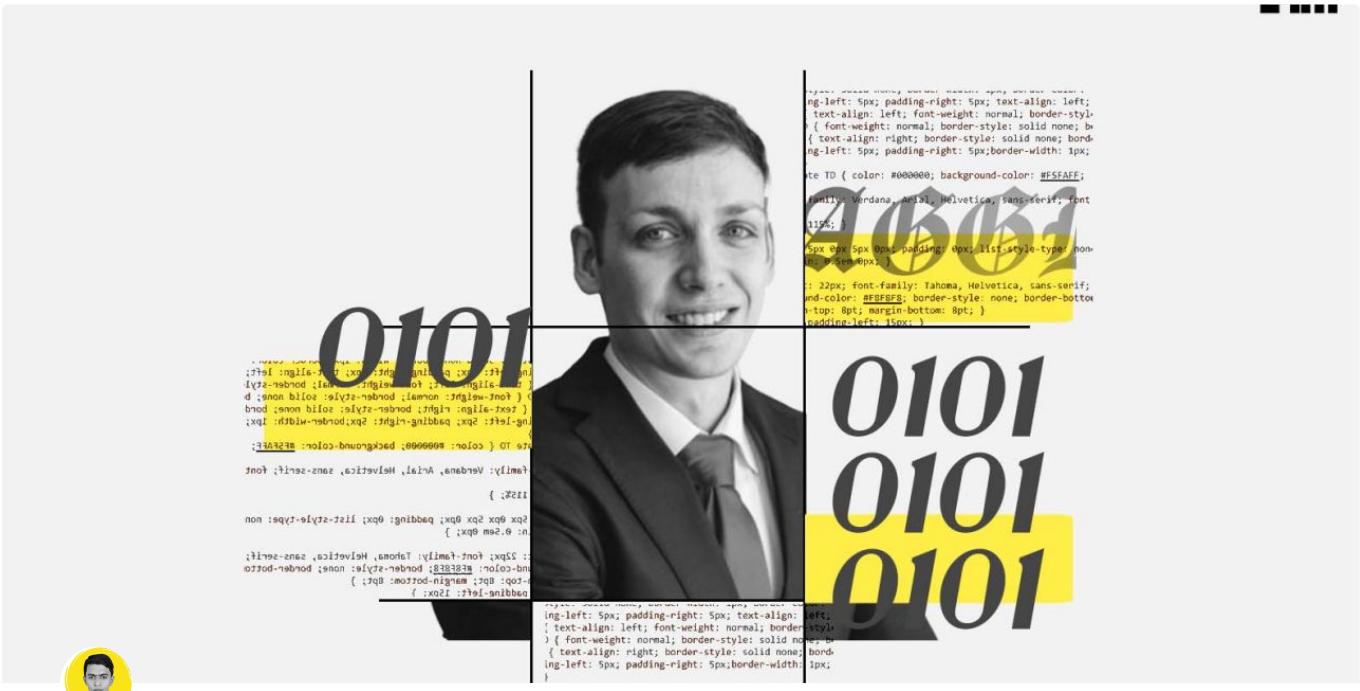
Conference, in-person (Bangalore)

Cypher 2022

21-23rd Sep

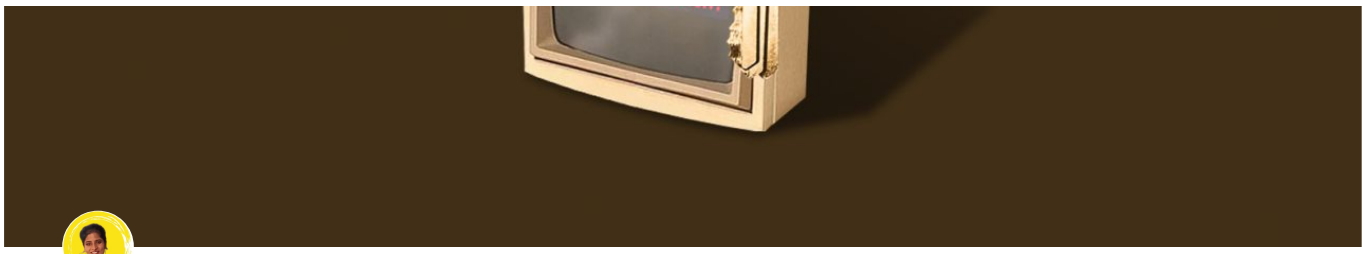
Register (https://www.analyticsindiasummit.com/cypher-2022/register/)

MORE FROM AIM



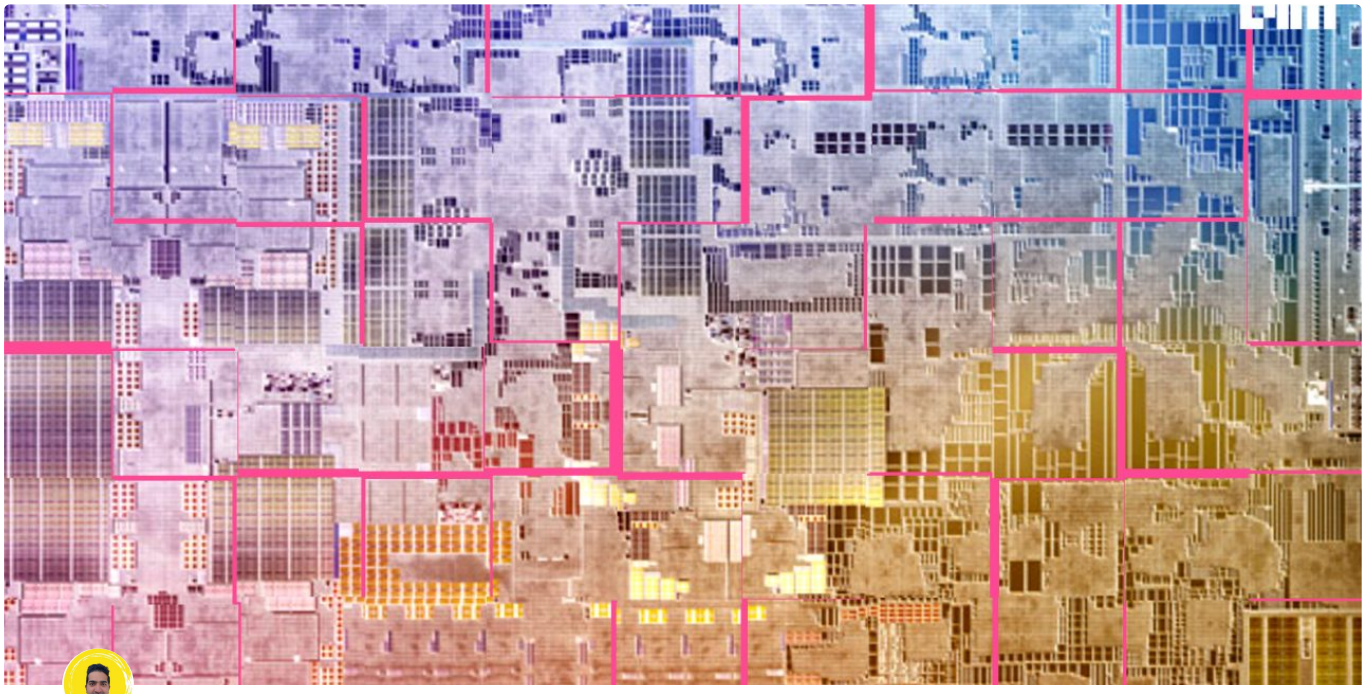
The story of Kaggle Grandmaster Janio Martinez Bachmann (https://analyticsindiamag.com/the-story-of-kaggle-grandmaster-janio-martinez-bachmann/)

I tend to get nervous when I must code next to a person



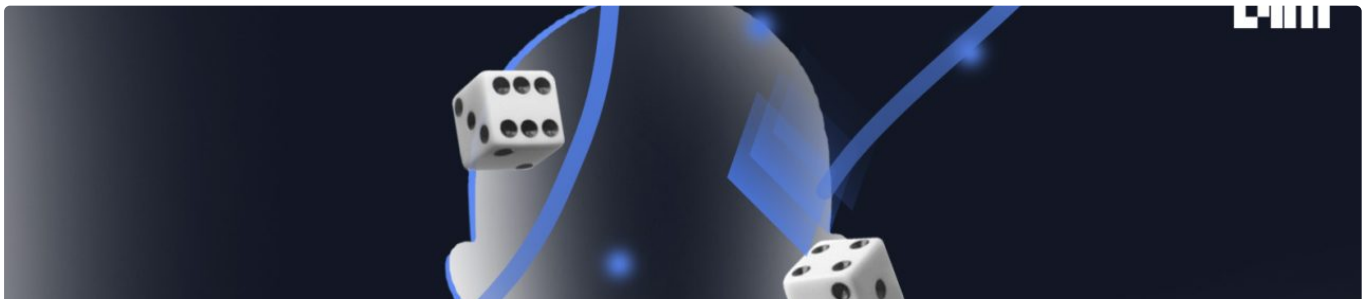
India needs dedicated cybersecurity laws: Kanwaljeet Kaur (https://analyticsindiamag.com/india-needs-dedicated-cybersecurity-laws-kanwaljeet-kaur/)

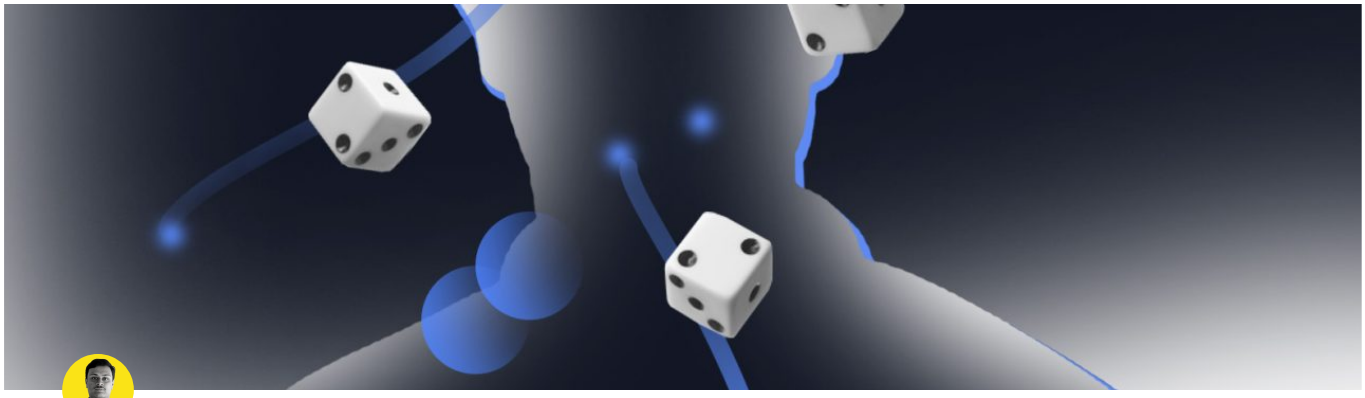
The government needs to inspire enough confidence that user data be protected from potential misuse.



Apple WWDC 2022 highlights: M2 chip with neural engine, search updates, APIs and more! (https://analyticsindiamag.com/apple-wwdc-2022-highlights-m2-chip-with-neural-engine-search-updates-apis-and-more/)

M2 incorporates the enhanced second-generation 5-nanometer technology.





How to learn from uncertainty using probabilistic machine learning? (https://analyticsindiamag.com/how-to-learn-from-uncertainty-using-probabilistic-machine-learning/)

This article briefs on the various ways a machine learning model learns from the data with uncertainty and why is probabilistic machine learning the best.

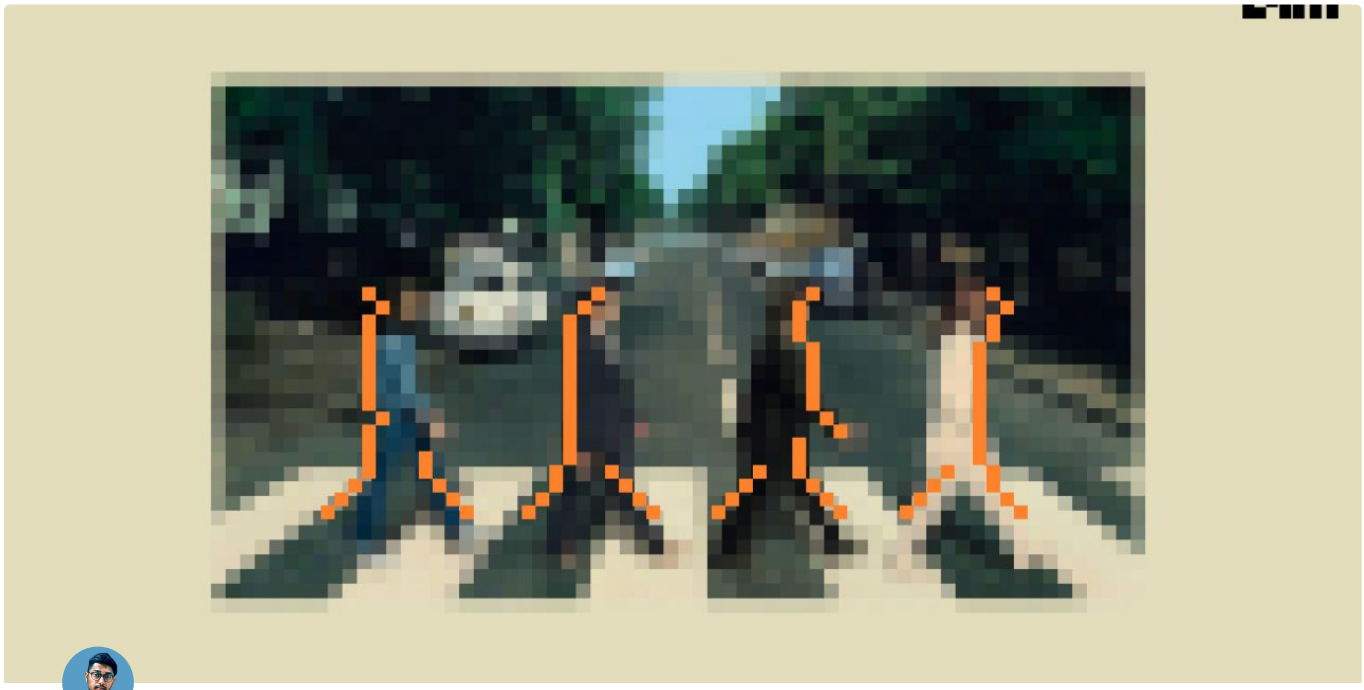
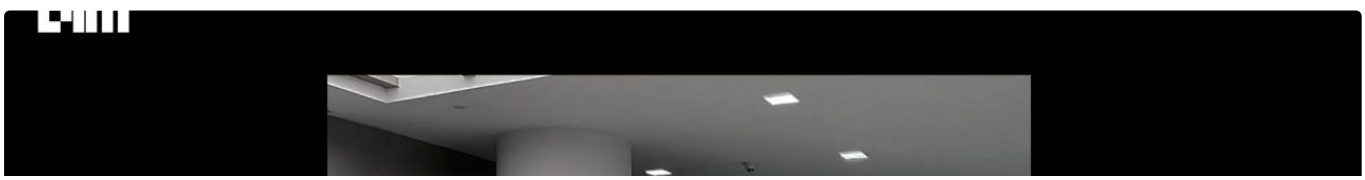
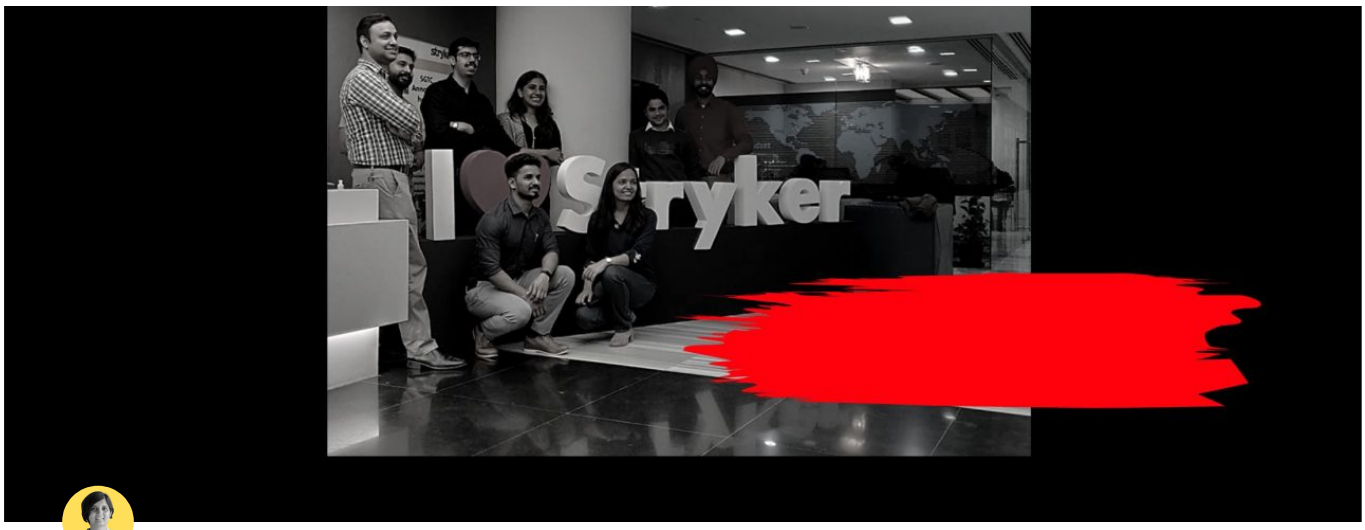


Image annotation techniques with implementation in OpenCV (https://analyticsindiamag.com/image-annotation-techniques-with-implementation-in-opencv/)

Image annotation is labelling of objects in an image



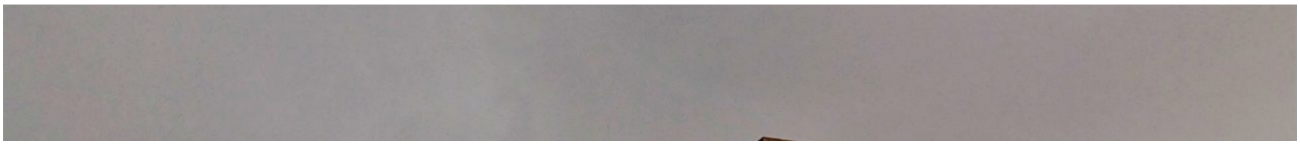


Medical tech giant Stryker opens captive in Gurgaon (https://analyticsindiamag.com/medical-tech-giant-stryker-opens-captive-in-gurgaon/)

SGTC strengthens our ability to innovate and develop new products and solutions that help improve and save lives around the world.

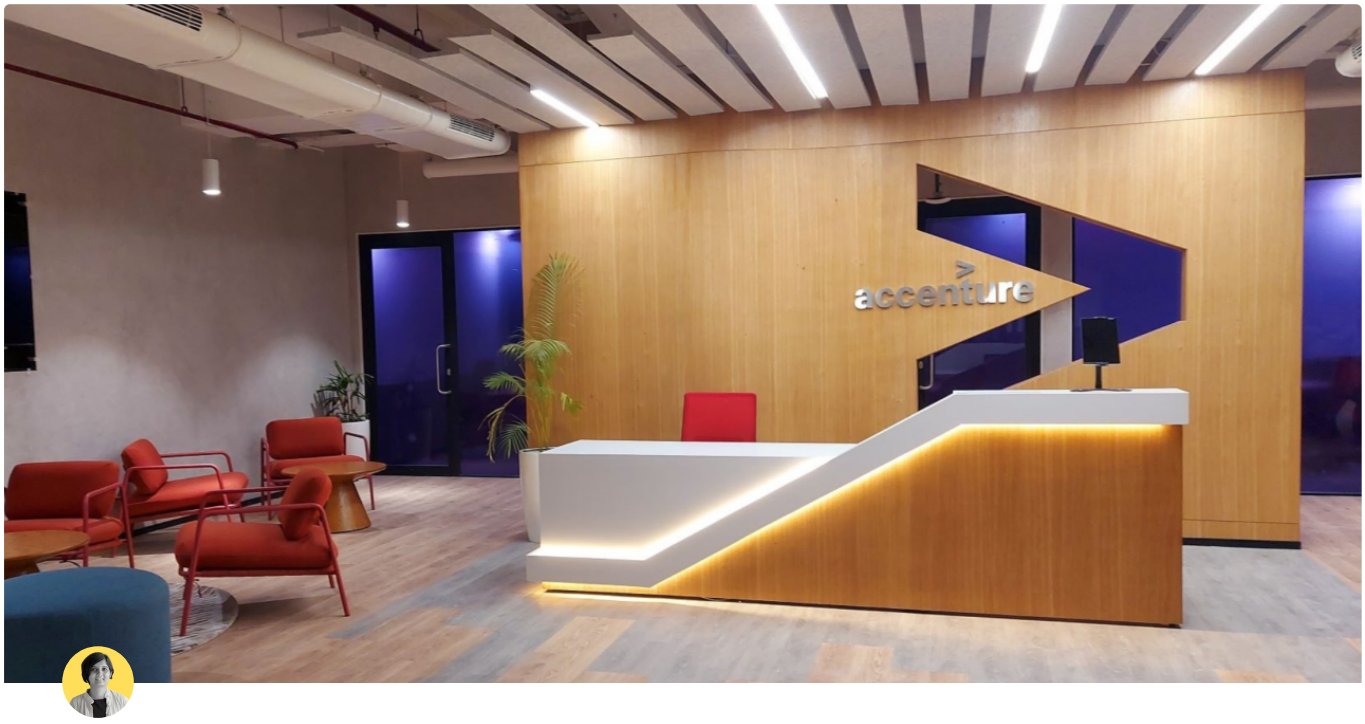
Why Maruti invested in Dave AI (https://analyticsindiamag.com/why-maruti-invested-in-dave-ai/)

Ananthakrishnan GopalAshok Balasundaram and Sriram P H founded SSPL in 2016.



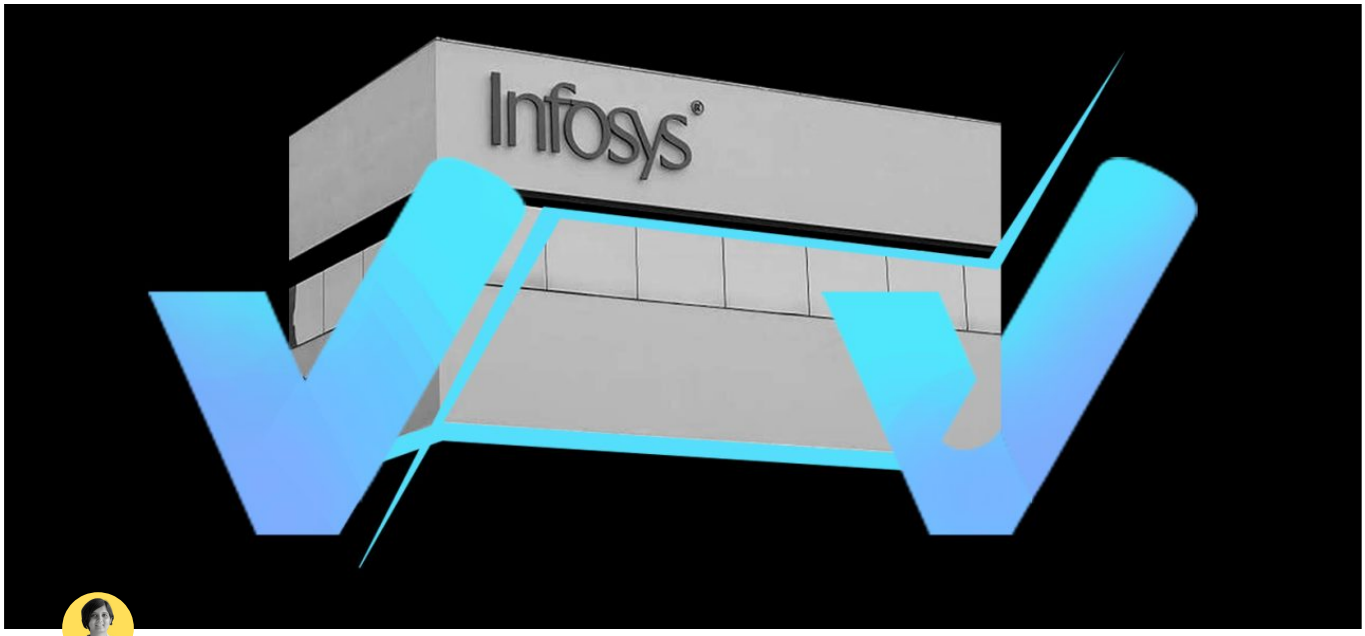
IIT Jodhpur launches MTech in AR/VR (https://analyticsindiamag.com/iit-jodhpur-launches-mtech-in-ar-vr/)

Open to all working professionals, registration for the course will be open until 17th June.



Accenture opens a Technology Center in Indore (https://analyticsindiamag.com/accenture-opens-a-technology-center-in-indore/)

Accenture has Advanced Technology Centers in Mumbai, Bangalore, Chennai, Hyderabad, Pune, Kolkata, and Delhi.



CredAvenue + Infosys: When tech firms reimagine co-lending (https://analyticsindiamag.com/credavenue-infosys-when-tech-firms-reimagine-co-lending/)

CredAvenue streamlines the entire co-lending process for lenders and originators by bringing banks and NBFCs in a unified marketplace.

(https://analyticsindiamag.com)

(https://www.linkedin.com/company/analytics-india-magazine/)

Understanding LSTM Networks

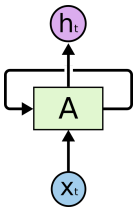
Posted on August 27, 2015

Recurrent Neural Networks

Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence.

Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

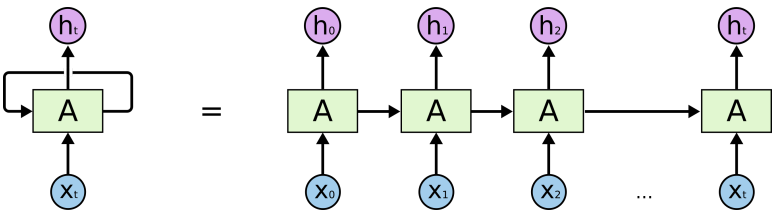
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, A, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



An unrolled recurrent neural network.

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

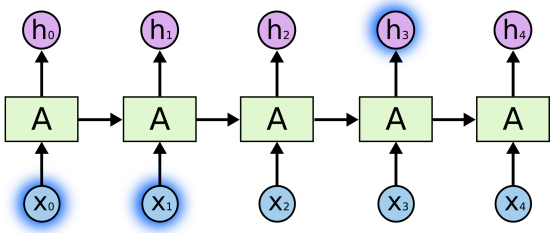
And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning... The list goes on. I'll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathy's excellent blog post, The Unreasonable Effectiveness of Recurrent Neural Networks (http://karpathy.github.io/2015/05/21/rnn-effectiveness/). But they really are pretty amazing.

Essential to these successes is the use of "LSTMs," a very special kind of recurrent neural network which works, for many tasks, much much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It's these LSTMs that this essay will explore.

The Problem of Long-Term Dependencies

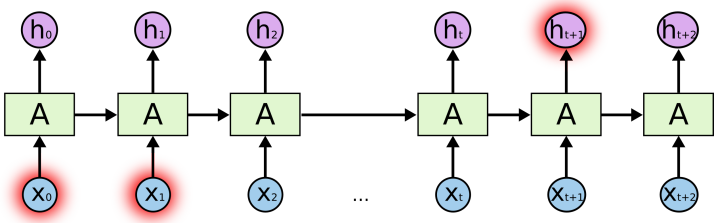
One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they'd be extremely useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the *sky*," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France... I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



In theory, RNNs are absolutely capable of handling such "long-term dependencies." A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] (http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf) and Bengio, et al. (1994) (http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf), who found some pretty fundamental reasons why it might be difficult.

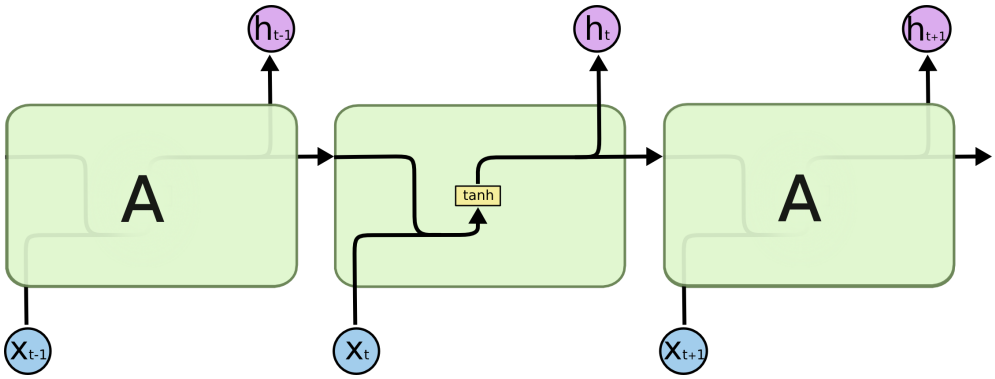
Thankfully, LSTMs don't have this problem!

LSTM Networks

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997) (http://www.bioinf.jku.at/publications/older/2604.pdf), and were refined and popularized by many people in following work.[1] They work tremendously well on a large variety of problems, and are now widely used.

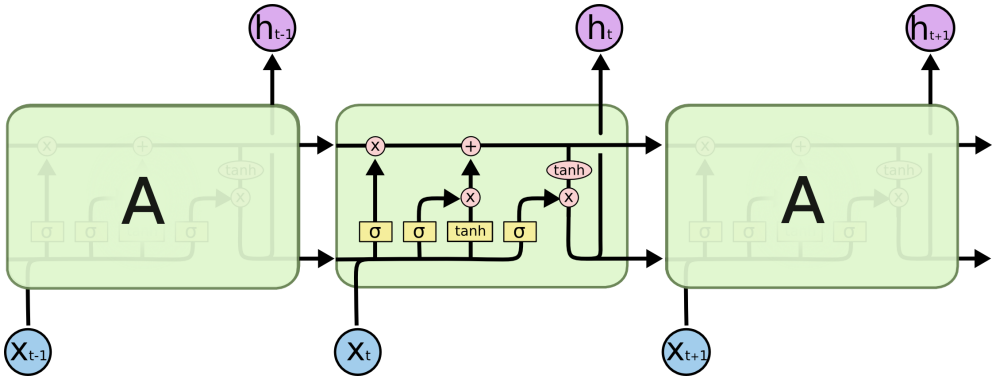
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



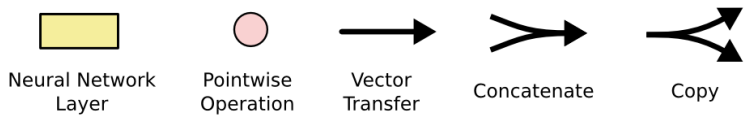
The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.

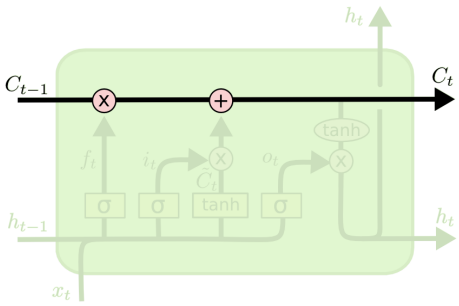


In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The Core Idea Behind LSTMs

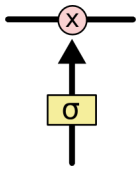
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



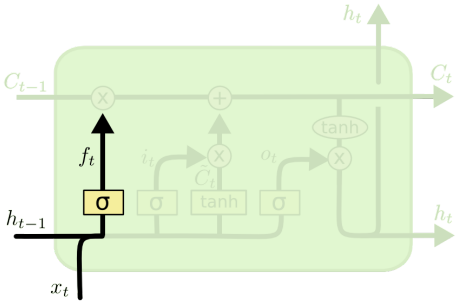
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

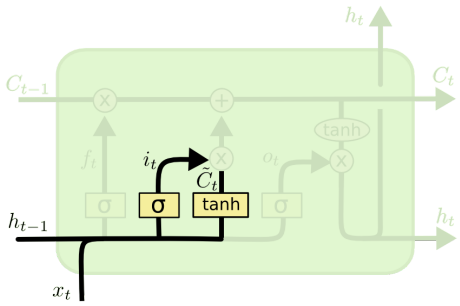
Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

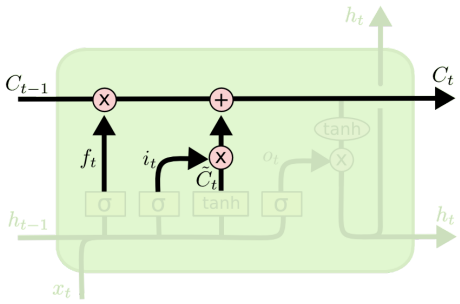


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

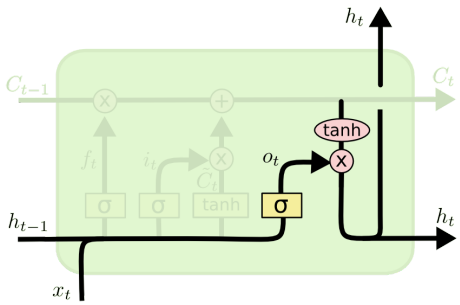
In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

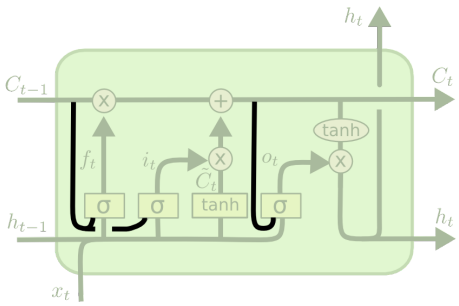


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Variants on Long Short Term Memory

What I've described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it's worth mentioning some of them.

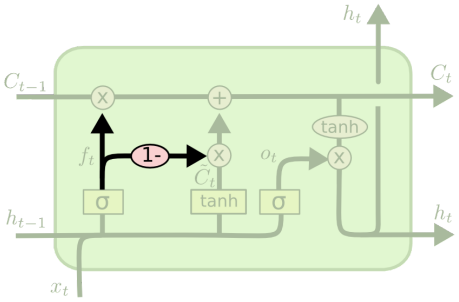
One popular LSTM variant, introduced by Gers & Schmidhuber (2000) (ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf), is adding "peephole connections." This means that we let the gate layers look at the cell state.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

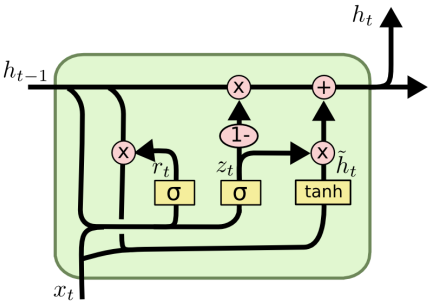
The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



$$C_t = f_t * C_{t-1} + \boldsymbol{(1 - f_t)} * \tilde{C}_t$$

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014) (http://arxiv.org/pdf/1406.1078v3.pdf). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015) (http://arxiv.org/pdf/1508.03790v2.pdf). There's also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014) (http://arxiv.org/pdf/1402.3511v1.pdf).

Which of these variants is best? Do the differences matter? Greff, et al. (2015) (http://arxiv.org/pdf/1503.04069.pdf) do a nice comparison of popular variants, finding that they're all about the same. Jozefowicz, et al. (2015) (http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.

Conclusion

Earlier, I mentioned the remarkable results people are achieving with RNNs. Essentially all of these are achieved using LSTMs. They really work a lot better for most tasks!

Written down as a set of equations, LSTMs look pretty intimidating. Hopefully, walking through them step by step in this essay has made them a bit more approachable.

LSTMs were a big step in what we can accomplish with RNNs. It's natural to wonder: is there another big step? A common opinion among researchers is: "Yes! There is a next step and it's attention!" The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs. In fact, Xu, *et al.* (2015) (http://arxiv.org/pdf/1502.03044v2.pdf) do exactly this – it might be a fun starting point if you want to explore attention! There's been a number of really exciting results using attention, and it seems like a lot more are around the corner...

Attention isn't the only exciting thread in RNN research. For example, Grid LSTMs by Kalchbrenner, *et al.* (2015) (http://arxiv.org/pdf/1507.01526v1.pdf) seem extremely promising. Work using RNNs in generative models – such as Gregor, *et al.* (2015) (http://arxiv.org/pdf/1502.04623.pdf), Chung, *et al.* (2015) (http://arxiv.org/pdf/1506.02216v3.pdf), or Bayer & Osendorfer (2015) (http://arxiv.org/pdf/1411.7610v3.pdf) – also seems very interesting. The last few years have been an exciting time for recurrent neural networks, and the coming ones promise to only be more so!

Acknowledgments

I'm grateful to a number of people for helping me better understand LSTMs, commenting on the visualizations, and providing feedback on this post.

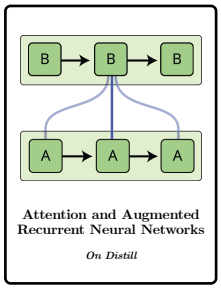
I'm very grateful to my colleagues at Google for their helpful feedback, especially Oriol Vinyals (http://research.google.com/pubs/OriolVinyals.html), Greg Corrado (http://research.google.com/pubs/GregCorrado.html), Jon Shlens (http://research.google.com/pubs/JonathonShlens.html), Luke Vilnis (http://people.cs.umass.edu/~luke/), and Ilya Sutskever (http://www.cs.toronto.edu/~ilya/). I'm also thankful to many other friends and colleagues for taking the time to help me, including Dario Amodei (https://www.linkedin.com/pub/dario-amodei/4/493/393), and Jacob Steinhardt (http://cs.stanford.edu/~jsteinhardt/). I'm especially thankful to Kyunghyun Cho (http://www.kyunghyuncho.me/) for extremely thoughtful correspondence about my diagrams.

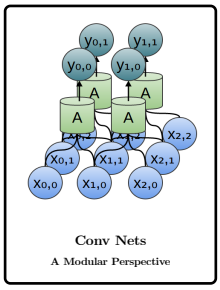
Before this post, I practiced explaining LSTMs during two seminar series I taught on neural networks. Thanks to everyone who participated in those for their patience with me, and for their feedback.

---

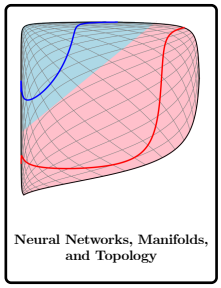
1. In addition to the original authors, a lot of people contributed to the modern LSTM. A non-comprehensive list is: Felix Gers, Fred Cummins, Santiago Fernandez, Justin Bayer, Daan Wierstra, Julian Togelius, Faustino Gomez, Matteo Gagliolo, and Alex Graves (https://scholar.google.com/citations?user=DaFHynwAAAAJ&hl=en).↩

More Posts



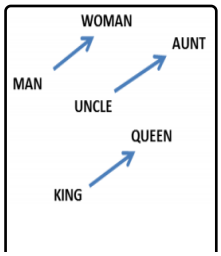


(http://distill.pub/2016/augmented-rnns/)



(../../posts/2014-07-Conv-Nets-Modular/)

(../../posts/2014-03-NN-Manifolds-Topology/)



(../../posts/2014-07-NLP-RNNs-Representations/)

Deep Learning, NLP, and Representations

69 Comments (/posts/2015-08-Understanding-LSTMs/#disqus_thread)

Introduction to Gated Recurrent Unit (GRU)

Shipra Saxena — Published On March 17, 2021 and Last Modified On March 18th, 2021

Advanced Deep Learning Videos

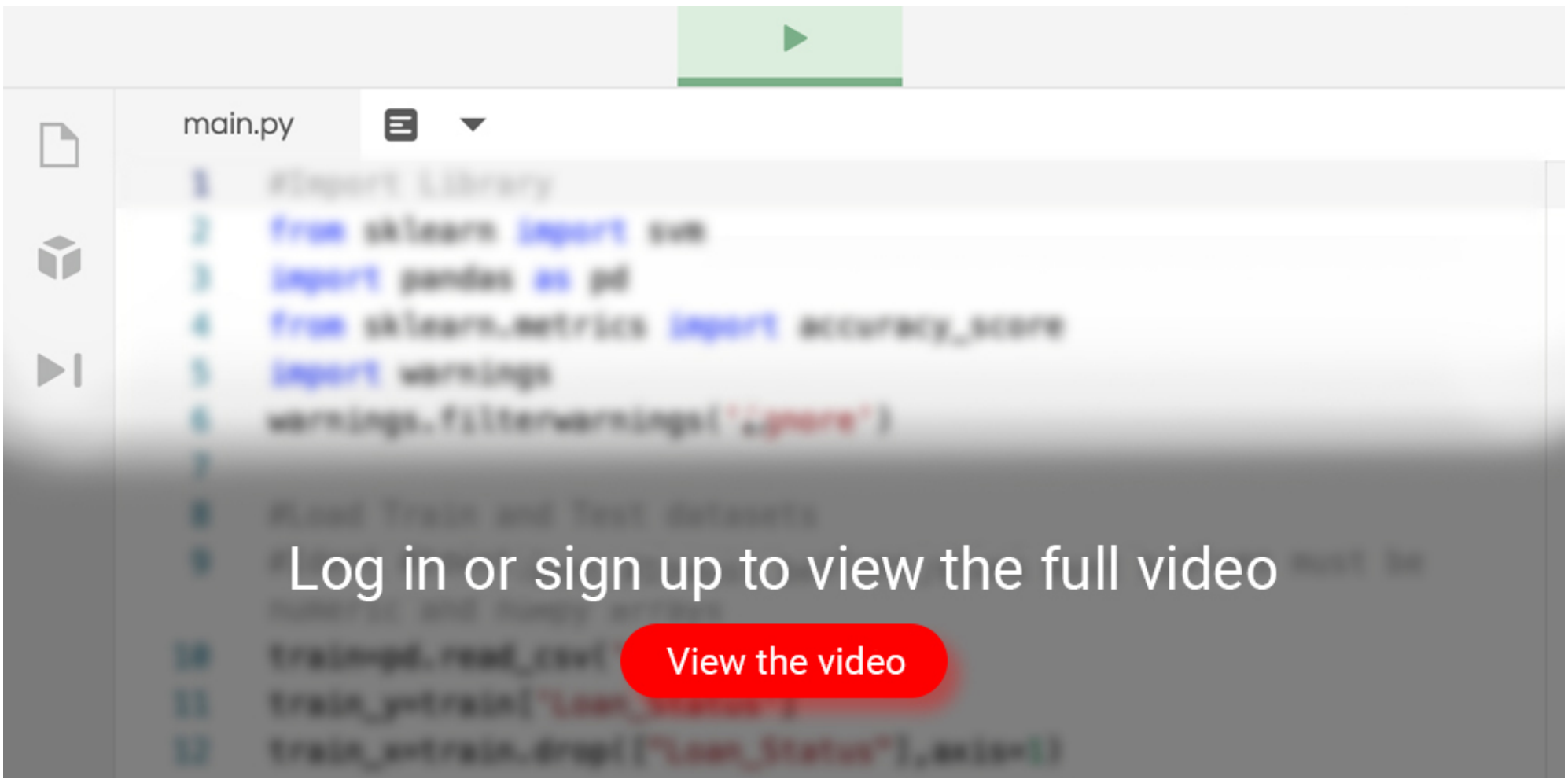
Objective

- In sequence modeling techniques, the Gated Recurrent Unit is the newest entrant after RNN and LSTM, hence it offers an improvement over the other two.
- Understand the working of GRU and how it is different from LSTM

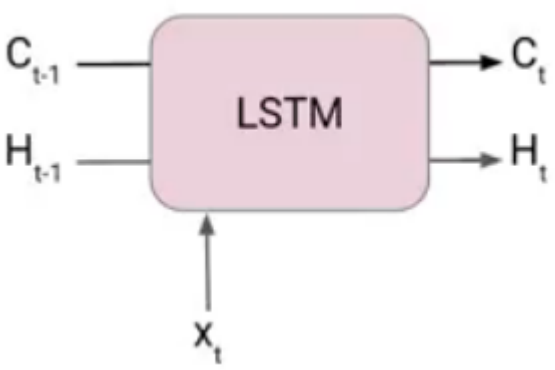
Introduction

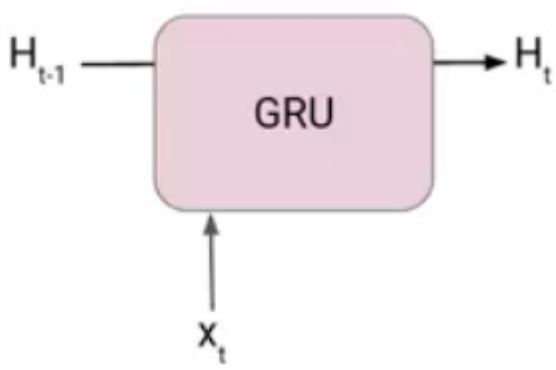
GRU or Gated recurrent unit is an advancement of the standard RNN i.e recurrent neural network. It was introduced by Kyunghyun Cho et al in the year 2014.

Note: If you are more interested in learning concepts in an Audio-Visual format, We have this entire article explained in the video below. If not, you may continue reading.



GRUs are very similar to Long Short Term Memory(LSTM). Just like LSTM, GRU uses gates to control the flow of information. They are relatively new as compared to LSTM. This is the reason they offer some improvement over LSTM and have simpler architecture.





Introduction to Gated Recurrent Unit (GRU)

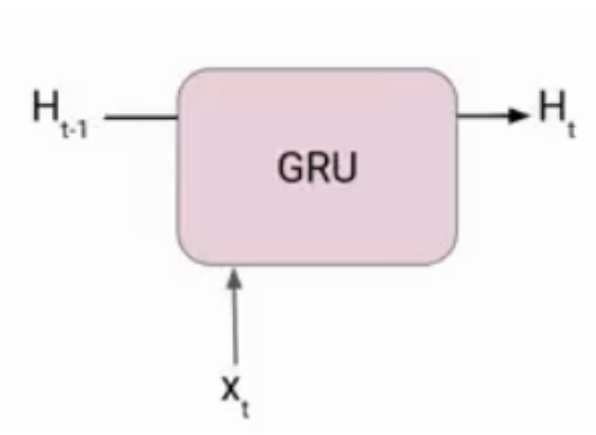
state(Ht). Due to the simpler architecture, GRUs are faster to train.

In case you are unaware of the LSTM network, I will suggest you go through the following article-

- *Introduction to Long Short term Memory(LSTM)*

The architecture of Gated Recurrent Unit

Now lets' understand how GRU works. Here we have a GRU cell which more or less similar to an LSTM cell or RNN cell.



At each timestamp t, it takes an input Xt and the hidden state Ht-1 from the previous timestamp t-1. Later it outputs a new hidden state Ht which again passed to the next timestamp.

Now there are primarily two gates in a GRU as opposed to three gates in an LSTM cell. The first gate is the Reset gate and the other one is the update gate.

Reset Gate (Short term memory)

The Reset Gate is responsible for the short-term memory of the network i.e the hidden state (Ht). Here is the equation of the Reset gate.

$$r_t = \sigma\left(x_t * U_r + H_{t-1} * W_r\right)$$

If you remember from the LSTM gate equation it is very similar to that. The value of **rt** will range from 0 to 1 because of the sigmoid function. Here Ur and Wr are weight matrices for the reset gate.

Update Gate (Long Term memory)

Similarly, we have an Update gate for long-term memory and the equation of the gate is shown below.

$$u_t = \sigma\left(x_t * U_u + H_{t-1} * W_u\right)$$

The only difference is of weight metrics i.e Uu and Wu.

How GRU Works

Introduction to Gated Recurrent Unit (GRU)

generate what is known as the candidate hidden state. As shown below

Candidate Hidden State

$$\hat{H}_t = \tanh(x_t * U_g + (r_t \circ H_{t-1}) * W_g)$$

It takes in the input and the hidden state from the previous timestamp t-1 which is multiplied by the reset gate output rt. Later passed this entire information to the tanh function, the resultant value is the candidate's hidden state.

$$\hat{H}_t = \tanh(x_t * U_g + (r_t \circ H_{t-1}) * W_g)$$

The most important part of this equation is how we are using the value of the reset gate to control how much influence the previous hidden state can have on the candidate state.

If the value of rt is equal to 1 then it means the entire information from the previous hidden state Ht-1 is being considered.

Likewise, if the value of rt is 0 then that means the information from the previous hidden state is completely ignored.

Hidden state

Once we have the candidate state, it is used to generate the current hidden state Ht. It is where the Update gate comes into the picture. Now, this is a very interesting equation, instead of using a separate gate like in LSTM in GRU we use a single update gate to control both the historical information which is Ht-1 as well as the new information which comes from the candidate state.

$$H_t = u_t \circ H_{t-1} + (1-u_t) \circ \hat{H}_t$$

Now assume the value of ut is around 0 then the first term in the equation will vanish which means the new hidden state will not have much information from the previous hidden state. On the other hand, the second part becomes almost one that essentially means the hidden state at the current timestamp will consist of the information from the candidate state only.

$$H_t = u_t \circ H_{t-1} + (1-u_t) \circ \hat{H}_t$$

Similarly, if the value of ut is on the second term will become entirely 0 and the current hidden state will entirely depend on the first term i.e the information from the hidden state at the previous timestamp t-1.

$$H_t = u_t \circ H_{t-1} + (1-u_t) \circ \hat{H}_t$$

Hence we can conclude that the value of ut is very critical in this equation and it can range from 0 to 1.

In case, you are interested to know more about GRU I suggest you read this Paper.

End Notes

So just to summarize, Let's see how different GRU is from LSTM.

Introduction to Gated Recurrent Unit (GRU)

Whereas in GRU we have a Reset gate and Update gate.

In LSTM we have two states Cell state or Long term memory and Hidden state also known as Short term memory. In the case of GRU, there is only one state i.e Hidden state (Ht).

If you are looking to kick start your Data Science Journey and want every topic under one roof, your search stops here. Check out Analytics Vidhya's Certified AI & ML BlackBelt Plus Program

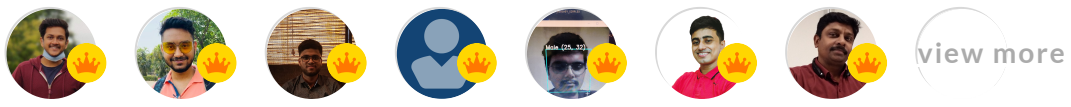
This is all about GRU in this article. If you have any queries, let me know in the comments section!

Gated Recurrent Unit

About the Author

Shipra Saxena

Our Top Authors



Introduction to Gated Recurrent Unit (GRU)

Improving your Deep Learning model using Model Checkpointing- Part 1

Why Are Generative Adversarial Networks(GANs) So Famous And How Will GANs Be In The Future?

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

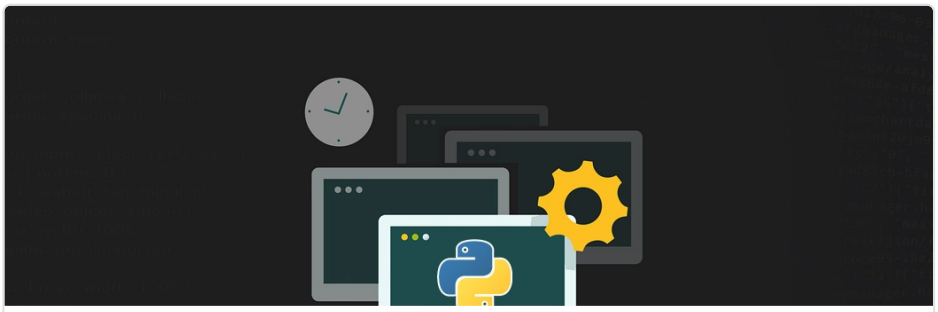
Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

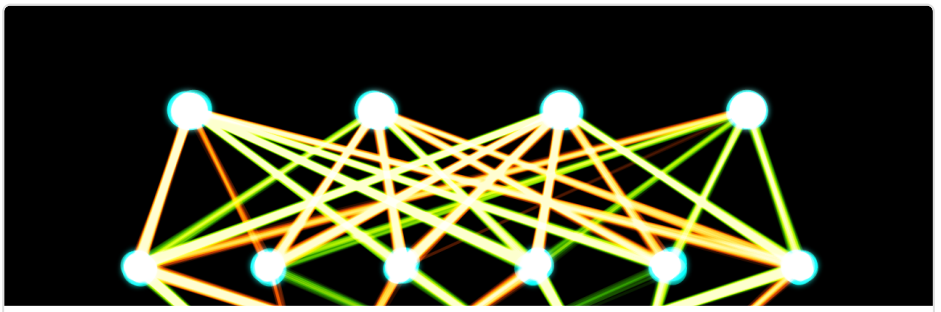
Submit

Top Resources



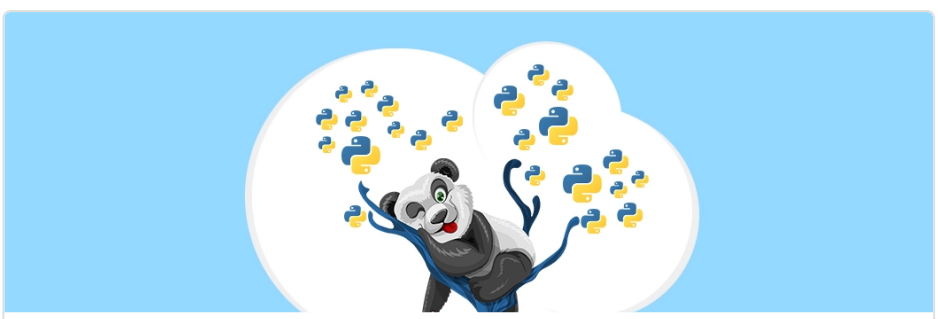
Python Tutorial: Working with CSV file for Data Science

Harika Bonthu - AUG 21, 2021



Boost Model Accuracy of Imbalanced COVID-19 Mortality Prediction Using GAN-based..

Bala Gangadhar Thilak Adiboina - OCT 07, 2020



Joins in Pandas: Master the Different Types of Joins in..

Understanding Random Forest

Introduction to Gated Recurrent Unit (GRU)