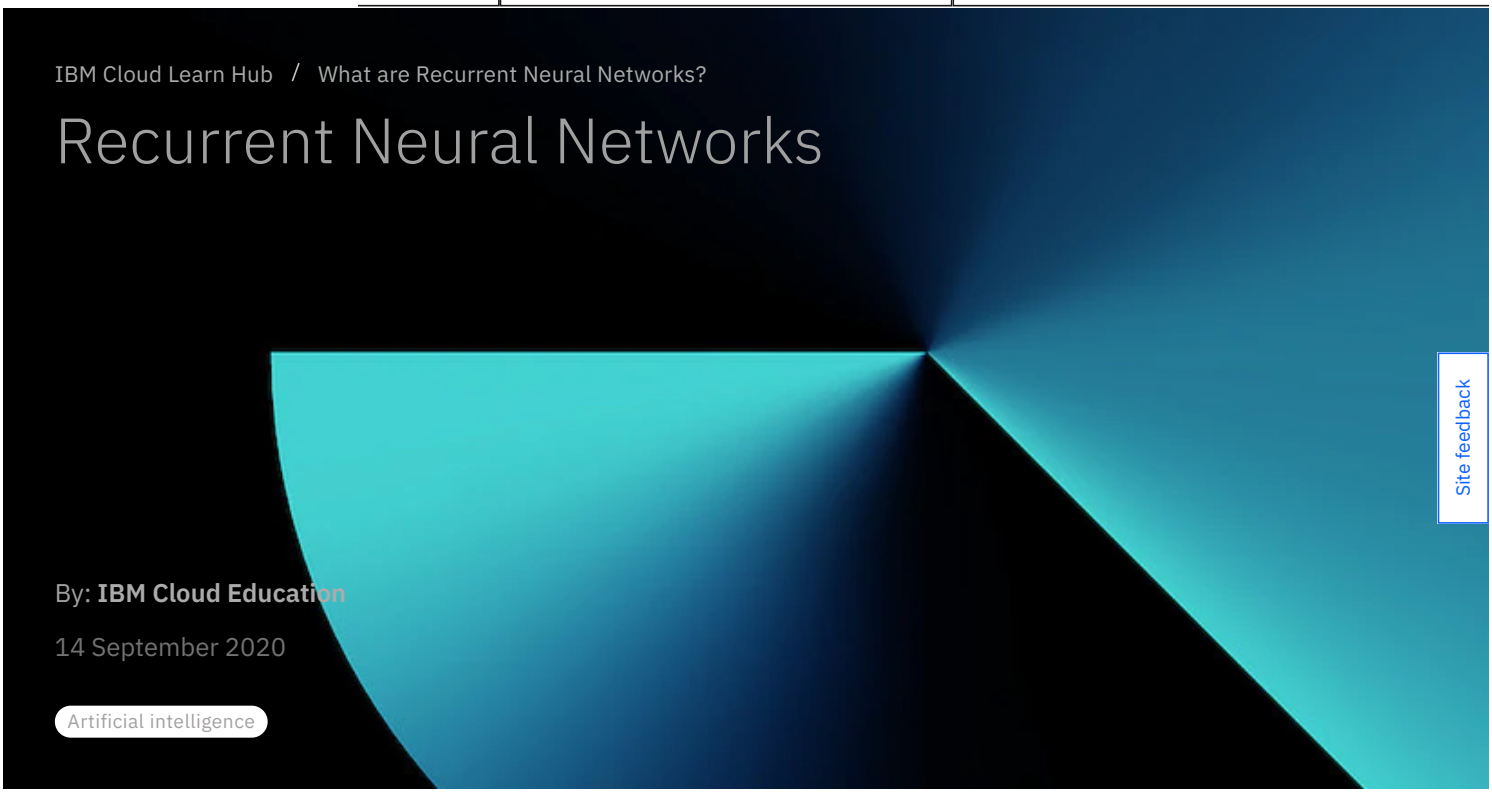
Recurrent Neural Networks



By: **IBM Cloud Education**

14 September 2020

Artificial intelligence

Recurrent Neural Networks

Learn how recurrent neural networks use sequential data to solve common temporal problems seen in language translation and speech recognition.

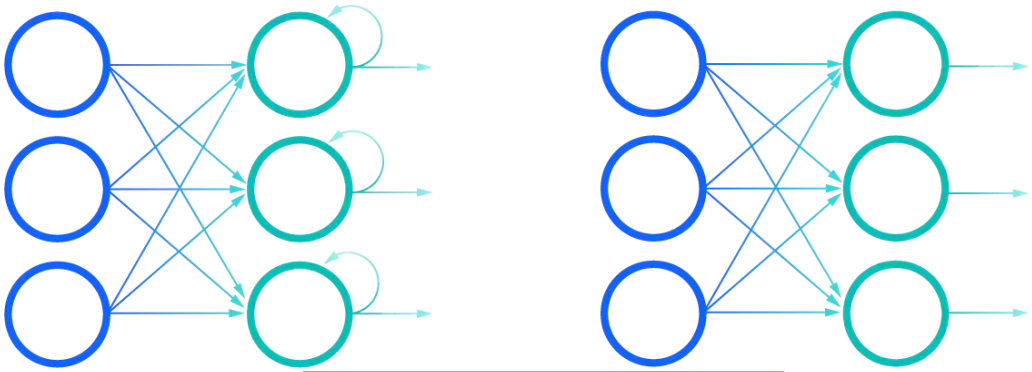
What are recurrent neural networks?

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate. Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn. They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence. While future events would also be helpful in determining the output of a given sequence, unidirectional recurrent neural networks cannot account for these events in their predictions.

Sequential data meaning continuously analyzing words in text in order. Applied for machine translation, speech recognition, image captioning, NLP applications I want to explore in my original work!

Special in fact that it uses "memory"

Recurrent Neural Network vs. Feedforward Neural Network



Main difference is RNN gives back the output result to the network for use

Comparison of Recurrent Neural Networks (on the left) and Feedforward Neural Networks (on the right)

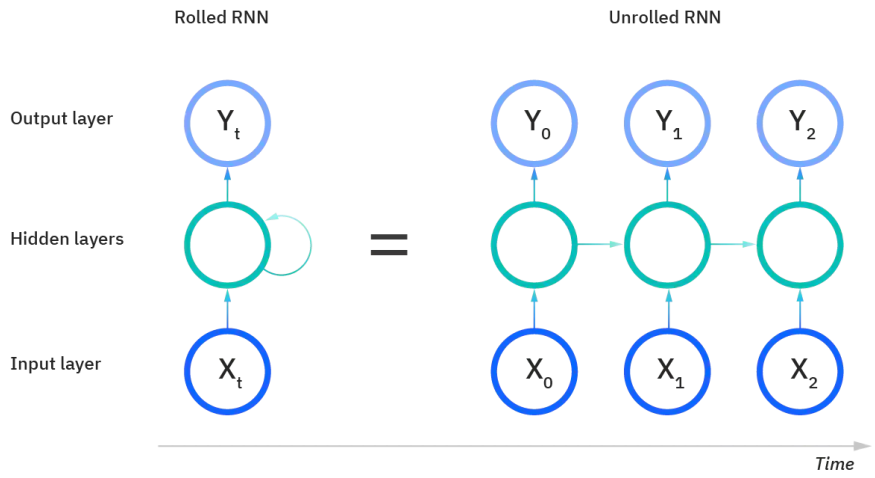
Let's take an idiom, such as "feeling under the weather", which is commonly used when someone is ill, to aid us in the explanation of RNNs. In order for the idiom to make sense, it needs to be expressed in that specific order. As a result, recurrent networks need to account for the position of each word in the idiom and they use that information to predict the next word in the sequence.

The use case helps show how all the positions of each word are connected to each other in meaning and used for text prediction and analysis.

Looking at the visual below, the "rolled" visual of the RNN represents the whole neural network, or rather the entire predicted phrase, like "feeling under the weather." The "unrolled" visual represents the individual layers, or time steps, of the neural network. Each layer maps to a single word in that phrase, such as "weather". Prior inputs, such as "feeling" and "under", would be represented as a hidden state in the third timestep to predict the output in the sequence, "the".

Site feedback

Let's talk



This is an RNN with all the steps that take in place in the cycle

Another distinguishing characteristic of recurrent networks is that they share 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. That said, these weights are still adjusted in the through the processes of backpropagation and gradient descent to facilitate reinforcement learning.

Different to regular neural networks that they share the same weight parameters in each layers of the network because it is "recurring."

Recurrent neural networks leverage backpropagation through time (BPTT) algorithm to determine the gradients, which is slightly different from traditional backpropagation as it is specific to sequence data. The principles of BPTT are the same as traditional backpropagation, where the model trains itself by calculating errors from its output layer to its input layer. These calculations allow us to adjust and fit the parameters of the model appropriately. BPTT differs from the traditional approach in that BPTT sums errors at each time step whereas feedforward networks do not need to sum errors as they do not share parameters across each layer.

Uses the multiple error calculations from output to change parameters for the inputs. Sums the errors because they each layer shares the same parameters.

Through this process, RNNs tend to run into two problems, known as exploding gradients and vanishing gradients. These issues are defined by the size of the gradient, which is the slope of the loss function along the error curve. When the gradient is too small, it continues to become smaller, updating the weight parameters until they become insignificant—i.e. 0. When that occurs, the algorithm is no longer learning. Exploding gradients occur when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN. One solution to these issues is to reduce the number of hidden layers within the neural network, eliminating some of the complexity in the RNN model.

Does have some negatives such as vanishing gradients meaning no learning and unstable models due to large gradients which is why I like transformers.

Types of recurrent neural networks

Let's talk

Site feedback

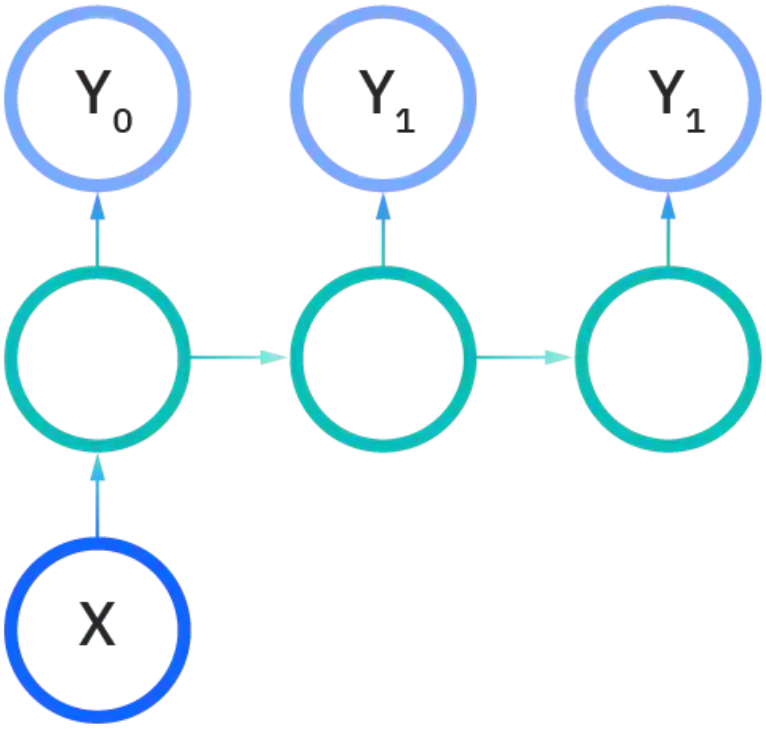
Feedforward networks map one input to one output, and while we've visualized recurrent neural networks in this way in the above diagrams, they do not actually have this constraint. Instead, their inputs and outputs can vary in length, and different types of RNNs are used for different use cases, such as music generation, sentiment classification, and machine translation. Different types of RNNs are usually expressed using the following diagrams:

Some applications of recurrent neural networks in the field of NLP!

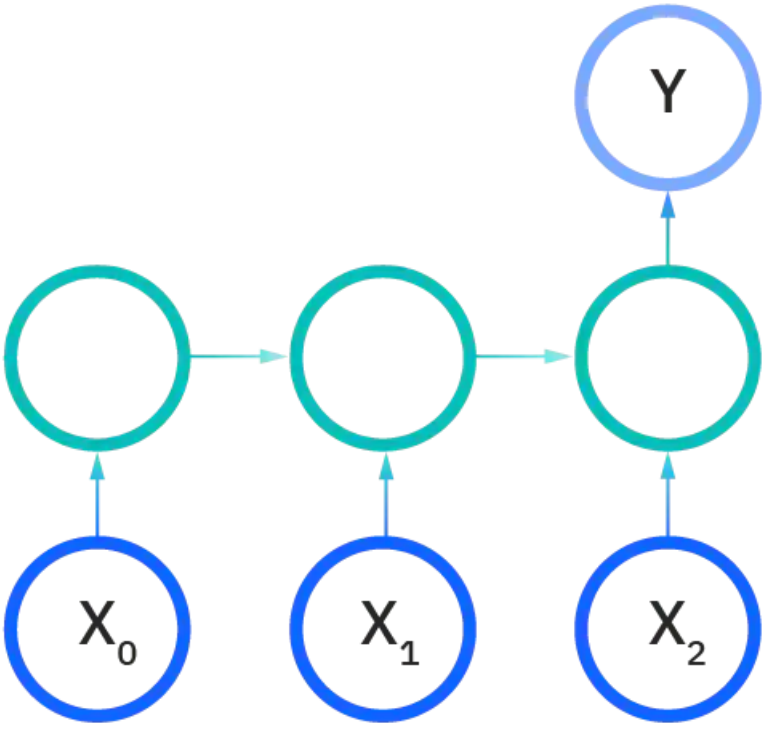
One-to-one:

Below are some possible representations of input, hidden, and output layers in recurrent neural networks showing their flexibility with many types of changes.

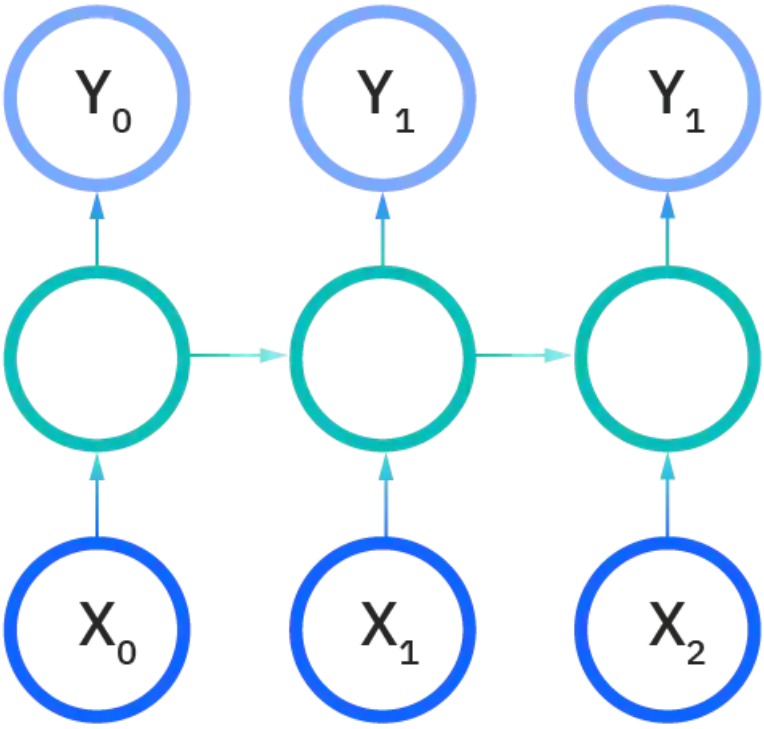
One-to-many:



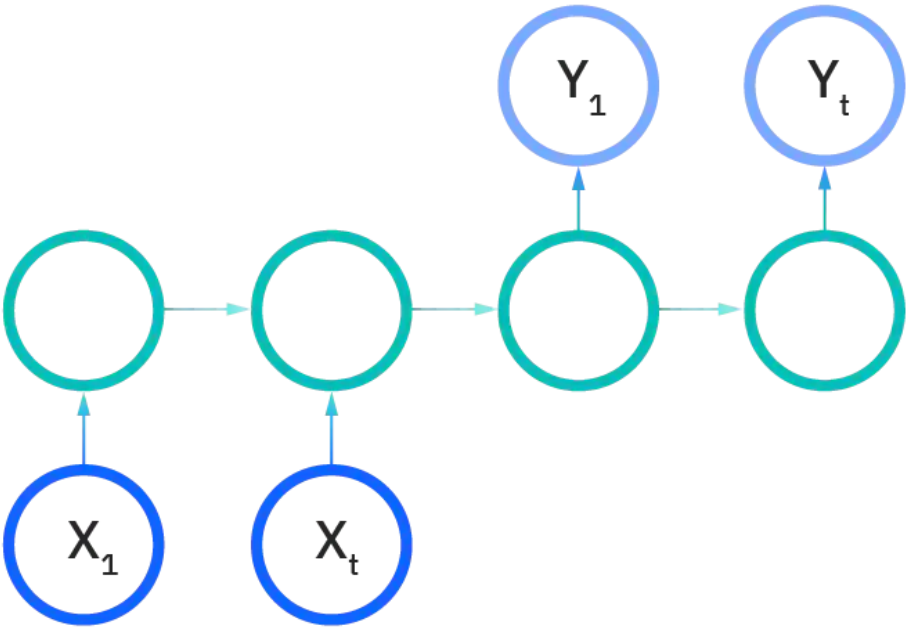
Many-to-one:



Many-to-many:



Many-to-many:

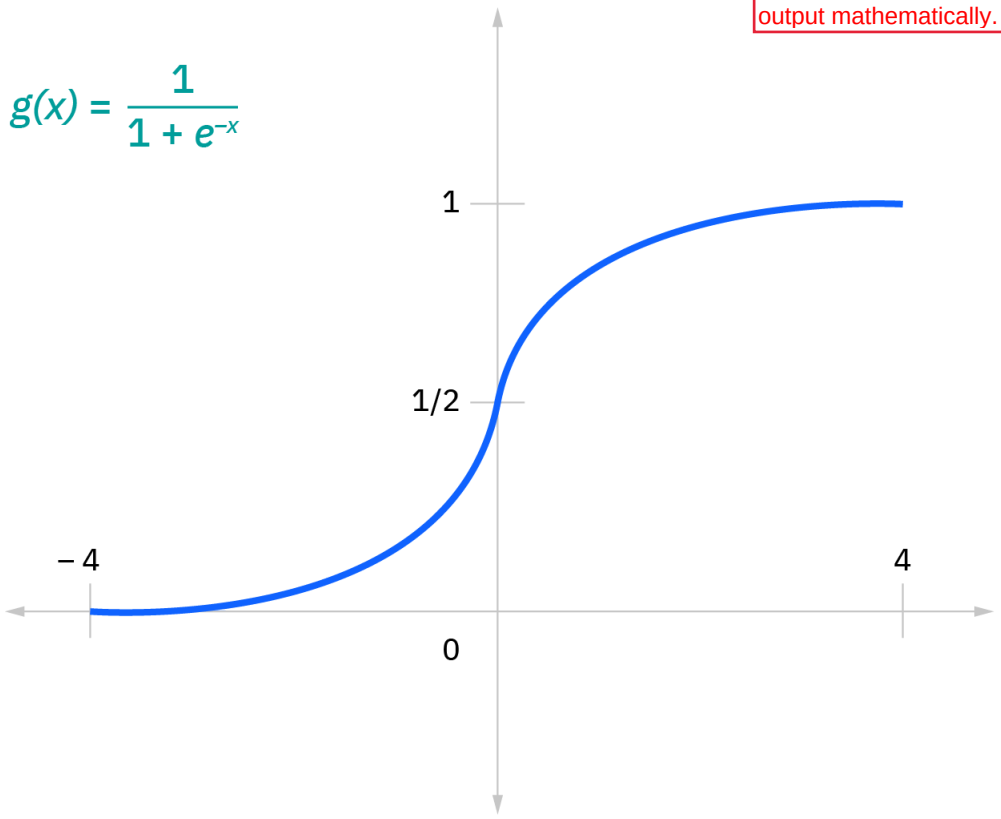


Common activation functions

As discussed in the Learn article on Neural Networks, an activation function determines whether a neuron should be activated. The nonlinear functions typically convert the output of a given neuron to a value between 0 and 1 or -1 and 1. Some of the most commonly used functions are defined as follows:

Sigmoid: This is represented with the formula g(x) = 1/(1 + e^-x).

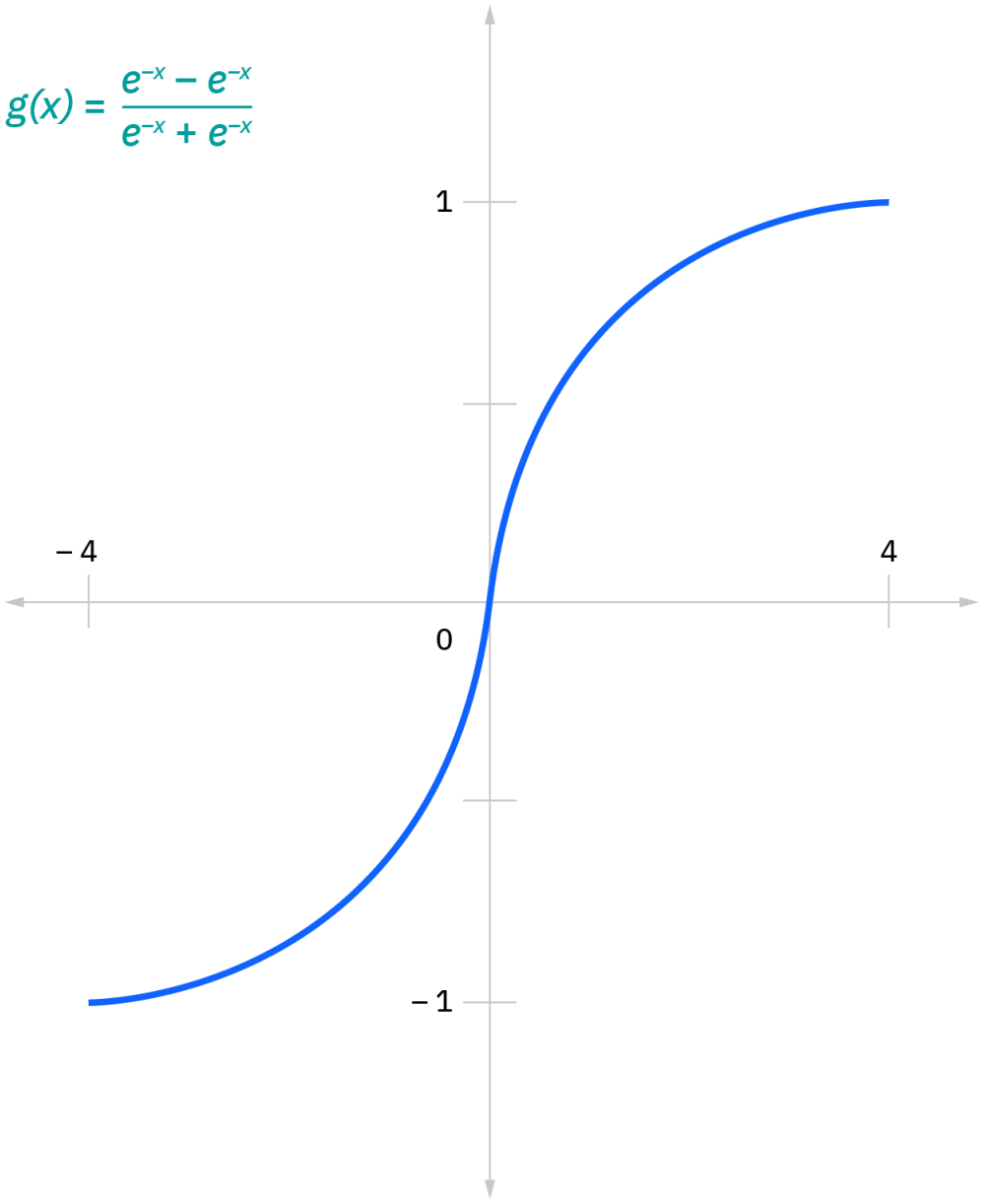
This relates to my knowledge from last year of using activation functions such as the Sigmoid, ReLU, and TanH to convert the output mathematically.



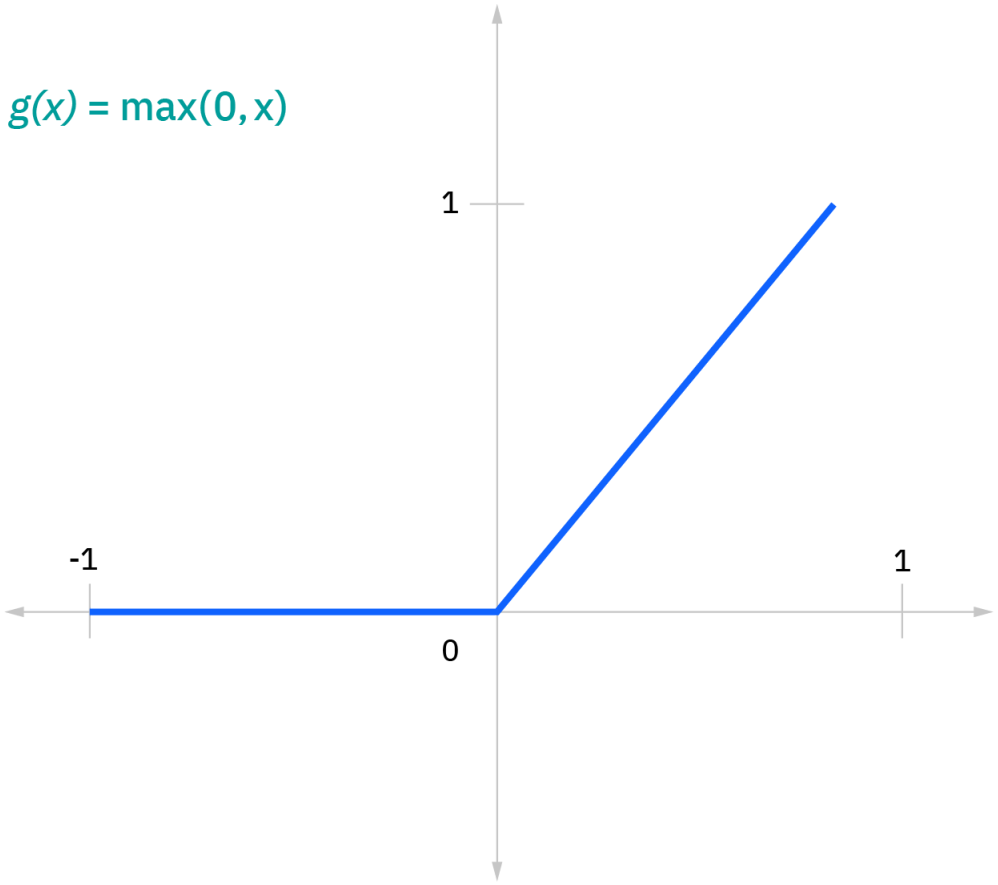
Tanh: This is represented with the formula g(x) = (e^-x - e^-x)/(e^-x + e^-x).

Site feedback

Let's talk



Relu: This is represented with the formula g(x) = max(0 , x)



Variant RNN architectures

Bidirectional recurrent neural networks (BRNN): These are a variant network architecture of RNNs. While unidirectional RNNs can only drawn from previous inputs to make predictions about the current state, bidirectional RNNs pull in future data to improve the accuracy of it. If we return to the example of "feeling under the weather" earlier in this article, the model can better predict that the second word in that phrase is "under" if it knew that the last word in the sequence is "weather."

BRNN helps improve RNNs through better accuracy by going into both forward and reverse word/sequence order of data.

Long short-term memory (LSTM): This is a popular RNN architecture, which was introduced by Sepp Hochreiter and Juergen Schmidhuber as a solution to vanishing gradient problem. In their paper (PDF, 388 KB) (link resides outside IBM), they work to address the problem of long-term dependencies. That is, if the previous state that is influencing the current prediction is not in the recent past, the RNN model may not be able to accurately predict the current state. As an example, let's say we wanted to predict the italicized words in following, "Alice is allergic to nuts. She can't eat *peanut*

LSTMs solve the vanishing gradient problem by remembering the initial data through taking care of long-term dependencies. Use cells and 3 gates to remember important data for long sequences of data.

s talk

butter." The context of a nut allergy can help us anticipate that the food that cannot be eaten contains nuts. However, if that context was a few sentences prior, then it would make it difficult, or even impossible, for the RNN to connect the information. To remedy this, LSTMs have "cells" in the hidden layers of the neural network, which have three gates–an input gate, an output gate, and a forget gate. These gates control the flow of information which is needed to predict the output in the network. For example, if gender pronouns, such as "she", was repeated multiple times in prior sentences, you may exclude that from the cell state.

Gated recurrent units (GRUs): This RNN variant is similar the LSTMs as it also works to address the short-term memory problem of RNN models. Instead of using a "cell state" regulate information, it uses hidden states, and instead of three gates, it has two—a reset gate and an update gate. Similar to the gates within LSTMs, the reset and update gates control how much and which information to retain.

GRUs help solve the short-term memory problem using hidden states with only 2 gates to control how much and which information to keep.

Site feedback

Recurrent neural networks and IBM Cloud

For decades now, IBM has been a pioneer in the development of AI technologies and neural networks, highlighted by the development and evolution of IBM Watson. Watson is now a trusted solution for enterprises looking to apply advanced natural language processing and deep learning techniques to their systems using a proven tiered approach to AI adoption and implementation.

IBM products, such as IBM Watson Machine Learning, also support popular Python libraries, such as TensorFlow, Keras, and PyTorch, which are commonly used in recurrent neural networks. Utilizing tools like, IBM Watson Studio and Watson Machine Learning, your enterprise can seamlessly bring your open-source AI projects into production while deploying and running your models on any cloud.

IBM Watson helps use NLP and RNNs for many business use cases, and their open-source AI projects could help me understand deployment and evenuse p re-trained models to test my learnings in the real world context.

For more information on how to get started with artificial intelligence technology, explore IBM Watson Studio.

Sign up for an IBMid and create your IBM Cloud account.

Featured products

IBM Watson Studio

Let's talk

Watson Machine Learning Accelerator

Related links

Data science →

Why IBM Cloud

Why IBM Cloud
Hybrid Cloud approach
Trust and security
Open Cloud
Data centers
Case studies

Products and Solutions

Cloud Paks
Cloud pricing
View all products
View all solutions

Site feedback

Learn about

What is Hybrid Cloud?
What is Cloud Computing?
What is Confidential Computing?
What is a Data Lake?
What is a Data Warehouse?
What is Artificial Intelligence (AI)?
What is Machine Learning?
What is DevOps?
What is Microservices?

Resources

Get started
Docs
Architectures
IBM Garage
Training and Certifications
Partners
Cloud blog
Hybrid Cloud careers
My Cloud account

Let's talk