NEURAL NETWORK ZOO PREQUEL: CELLS AND LAYERS

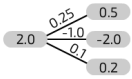
POSTED ON MARCH 31, 2017 BY FJODOR VAN VEEN

An informative chart to build

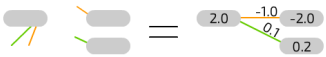
Neural Network Cells

©2016 Fjodor van Veen - asimovinstitute.org

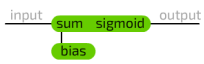
Fixed Weight
(fixed at 1)



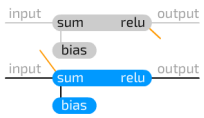
Weights



Recurrent Weights
(grouped by colour)

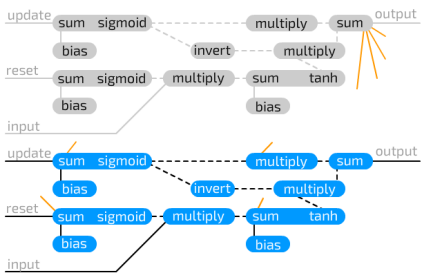


Feed Forward Cell
(basic cell)



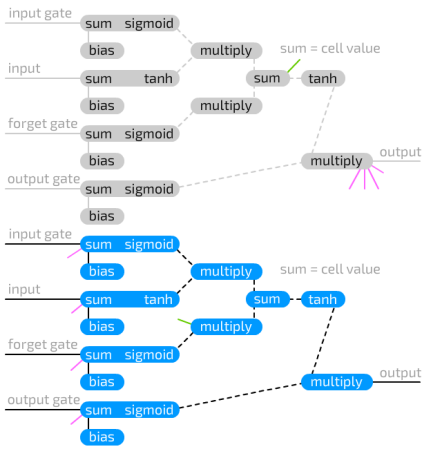
Recurrent Cell
(previous iteration)

Recurrent Cell



GRU Cell
(previous iteration)

GRU Cell

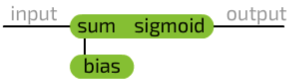


LSTM Cell
(previous iteration)

LSTM Cell

Cells

The Neural Network Zoo shows different types of cells and various layer connectivity styles, but it doesn't really go into how each cell type works. A number of cell types I originally gave different colours to differentiate the networks more clearly, but I have since found out that these cells work more or less the same way, so you'll find descriptions under the basic cell images.



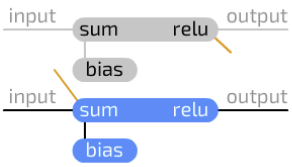
Feed Forward Cell
(basic cell)

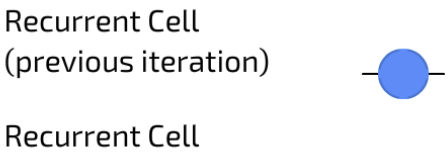
A basic neural network cell, the type one would find in a regular feed forward architecture, is quite simple. The cell is connected to other neurons via weights, i.e. it can be connected to all the neurons in the previous layer. Each connection has its own weight, which is often just a random number at first. A weight can be negative, positive, very small, very big or zero. The value of each of the cells it's connected to is multiplied by its respective connection weight. The resulting values are all added together. On top of this, a bias is also added. A bias can prevent a cell from getting stuck on outputting zero and it can speed up some operations, reducing the amount of neurons required to solve a problem. The bias is also a number, sometimes constant (often -1 or 1) and sometimes variable. This total sum is then passed through an activation function, the resulting value of which then becomes the value of the cell.

Convolutional cells are much like feed forward cells, except they're typically connected to only a few neurons from the previous layer. They are often used to preserve spatial information, because they are connected not to a few random cells but to all cells in a certain proximity. This makes them practical for data with lots of localised information, such as images and sound waves (but mostly images). Deconvolutional cells are just the opposite: these tend to decode spatial information by being locally connected to the next layer. Both cells often have a lot of clones which are trained independently; each clone having it's own weights but connected exactly the same way. These clones can be thought of as being located in separate networks which all have the same structure. Both are essentially the same as regular cells, but they are used differently.

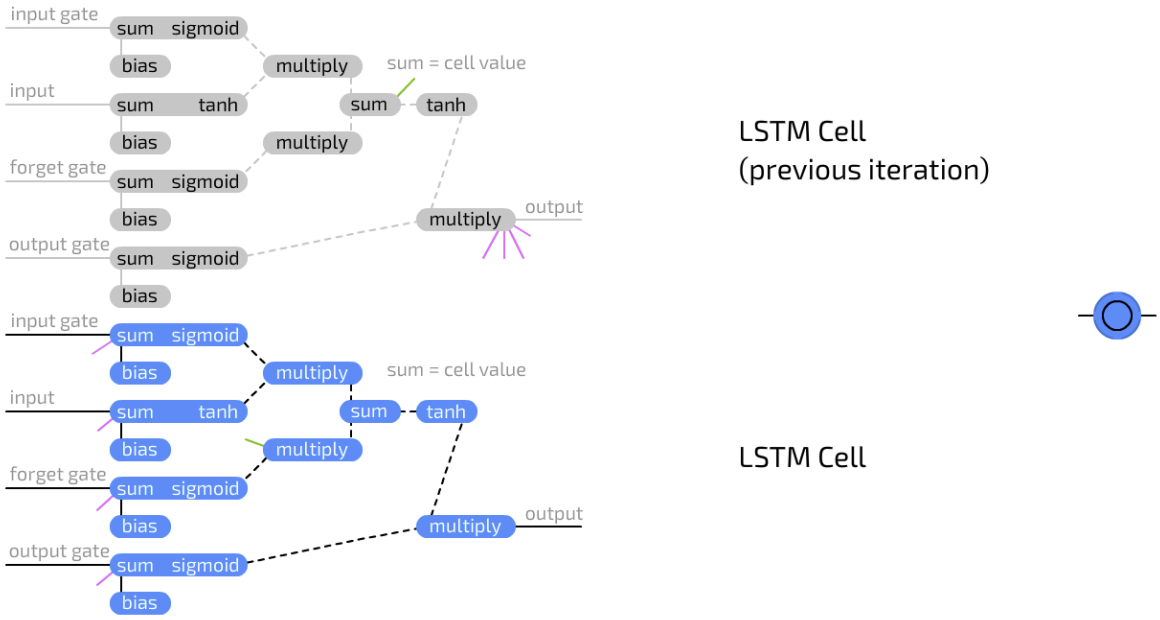
Pooling and interpolating cells are frequently combined with convolutional cells. These cells are not really cells, more just raw operations. Pooling cells take in the incoming connections and decide which connection gets passed through. In images, this can be thought of as zooming out on a picture. You can no longer see all the pixels, and it has to learn which pixels to keep and which to discard. Interpolating cells perform the opposite operation: they take in some information and map it to more information. The extra information is made up, like if one where to zoom in on a small resolution picture. Interpolating cells are not the only reverse operation of pooling cells, but they are relatively common as they are fast and simple to implement. They are respectively connected much like convolutional and deconvolutional cells.

Mean and standard deviation cells (almost exclusively found in couples as probabilistic cells) are used to represent probability distributions. The mean is the average value and the standard deviation represents how far to deviate from this average (in both directions). For example, a probabilistic cell used for images could contain the information on how much red there is in a particular pixel. The mean would say for example 0.5, and the standard deviation 0.2. When sampling from these probabilistic cells, one would enter these values in a Gaussian random number generator, resulting in anything between 0.4 and 0.6 being quite likely results, with values further away from 0.5 being less and less likely (but still possible). They are often fully connected to either the previous or the next layer and they do not have biases.

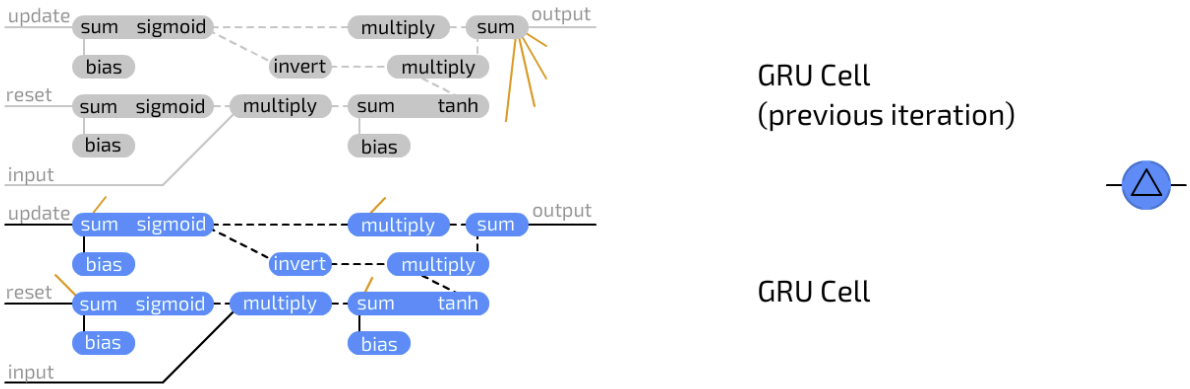




Recurrent cells have connections not just in the realm of layers, but also over time. Each cell internally stores its previous value. They are updated just like basic cells, but with extra weights: connected to the previous values of the cells and most of the time also to all the cells in the same layer. These weights between the current value and the stored previous value work much like a volatile memory (like RAM), inheriting both properties of having a certain “state” and vanishing if not fed. Because the previous value is a value passed through an activation function, and each update passes this activated value along with the other weights through the activation function, information is continually lost. In fact, the retention rate is so low, that only four or five iterations later, almost all of the information is lost.



Long short term memory cells are used to combat the problem of the rapid information loss occurring in recurrent cells. LSTM cells are logic circuits, copied from how memory cells were designed for computers. Compared to RNN cells which store two states, LSTM cells store four: the current and last value of the output and the current and last values of the state of the "memory cell". They have three "gates": input, output, forget, and they also have just the regular input. Each of these gates has its own weight meaning that connecting to this type of cell entails setting up four weights (instead of just one). The gates function much like flow gates, not fence gates: they can let everything through, just a little bit, nothing, or, anything in between. This works by multiplying incoming information by a value ranging from 0 to 1, which is stored in this gate value. The input gate, then, determines how much of the input is allowed to be added to the cell value. The output gate determines how much of the output value can be seen by the rest of the network. The forget gate is not connected to the previous value of the output cell, but rather connected to the previous memory cell value. It determines how much of the last memory cell state to retain. Because it's not connected to the output, much less information loss occurs, because no activation function is placed in the loop.



Gated recurrent units (cells) are a variation of LSTM cells. They too use gates to combat information loss, but do so with just 2 gates: update and reset. This makes them slightly less expressive but also slightly faster, as they use less connections everywhere. In essence there are two differences between LSTM cells and GRU cells: GRU cells do not have a hidden cell state protected by an output gate, and they combine the input and forget gate into a single update gate. The idea is that if you want to allow a lot of new information, you can probably forget some old information (and the other way around).

Layers

The most basic way of connecting neurons to form graphs is by connecting everything to absolutely everything. This is seen in Hopfield networks and Boltzmann machines. Of course, this means the number of connections grows exponentially, but the expressiveness is uncompromised. This is referred to as completely (or fully) connected.

After a while it was discovered that breaking the network up into distinct layers is a useful feature, where the definition of a layer is a set or group of neurons which are not connected to each other, but only to neurons from other group(s). This concept is for instance used in Restricted Boltzmann Machines. The idea of **using layers** is nowadays generalised for any number of layers and it can be found in almost all current architectures. This is (perhaps confusingly) also called fully connected or completely connected, because actually completely connected networks are quite uncommon.

Convolutionally connected layers are even more constrained than fully connected layers: we connect every neuron only to neurons in other groups that are close by. Images and sound waves contain a very high amount of information if used to feed

directly one-to-one into a network (e.g. using one neuron per pixel). The idea of convolutional connections comes from the observation that spatial information is probably important to retain. It turned out that this is a good guess, as it's used in many image and sound wave based neural network applications. This setup is however less expressive than fully connected layers. In essence it is a way of "importance" filtering, deciding which of the tightly grouped information packets are important; convolutional connections are great for dimensionality reduction. At what spatial distance neurons can still be connected depends on the implementation, but ranges higher than 4 or 5 neurons are rarely used. Note that "spatial" often refers to two-dimensional space, which is why most representations show three-dimensional sheets of neurons being connected; the connection range is applied in all dimensions.

Another option is of course to **randomly connected neurons**. This comes in two main variations as well: by allowing for some percentage of all possible connections, or to connect some percentage of neurons between layers. Random connections help to linearly reduce the performance of the network and can be useful in large networks where fully connected layers run into performance problems. A slightly more sparsely connected layer with slightly more neurons can perform better in some cases, especially where a lot of information needs to be stored but not as much information needs to be exchanged (a bit similar to the effectiveness of convolutionally connected layers, but then randomised). Very sparsely connected systems (1 or 2%) are also used, as seen in ELMs, ESNs and LSMs. Especially in the case of spiking networks this makes a lot of sense, because the more connections a neuron has, the less energy each weight will carry over, meaning less propagating and repeating patterns.

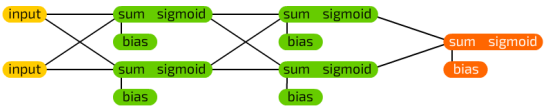
Time delayed connections are connections between neurons (often from the same layer, and even connected with themselves) that don't get information from the previous layer, but from a layer from the past (previous iteration, mostly). This allows temporal (time, sequence or order) related information to be stored. These types of connections are often manually reset from time to time, to clear the "state" of the network. The key difference with regular connections is that these connections are continuously changing, even when the network isn't being trained.

The following image shows some small sample networks of the types described above, and their connections. I use it when I get stuck on just exactly what is connected to what (which is particularly likely when working with LSTM or GRU cells):

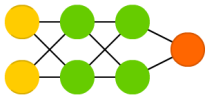
An informative chart to build

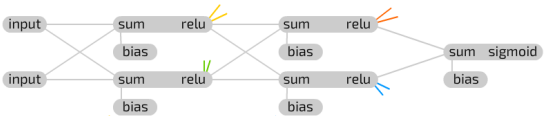
Neural Network Graphs

©2016 Fjodor van Veen - asimovinstitute.org

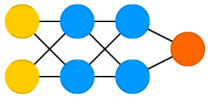


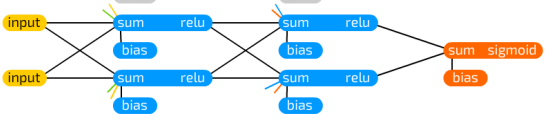
Deep Feed Forward Example



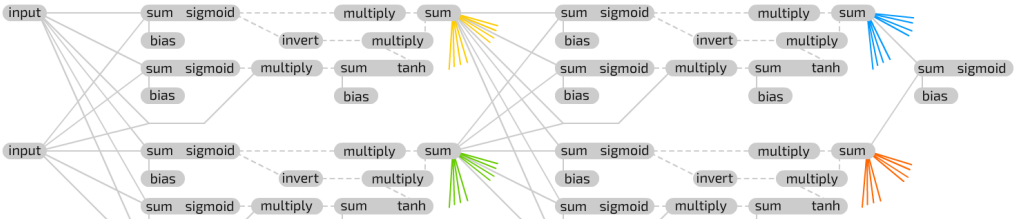


Deep Recurrent Example
(previous iteration)

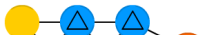




Deep Recurrent Example



Deep GRU Example
(previous iteration)



THE ASIMOV INSTITUTE

FOR ARTIFICIAL CREATIVITY & CONSTRAINT

BLOG

Neural Network Zoo Prequel: Cells and Layers
March 31, 2017

Analyzing Six Deep Learning Tools for Music Generation
October 5, 2016

The Neural Network Zoo
September 14, 2016

Artistic Style Transfer Blending
July 31, 2016

A Quick Guide to Installing TensorFlow on mac OS
June 15, 2016

LINKS

GitHub
Bitbucket
Twitter
LinkedIn

ADMIN

Log in
Entries RSS
Comments RSS
WordPress.org

SEARCH

Search …

© 2017, THE ASIMOV INSTITUTE
PROUDLY POWERED BY WORDPRESS
THEME: YUUTA BY FELIX DORNER