**Not Snap To Grid**

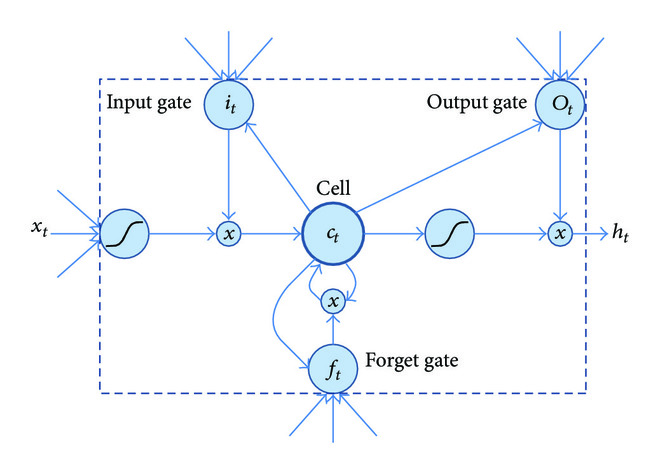
A generic example of a Convolutional Neural Network model. The usual architecture alternates convolution and subsampling layers. Fully connected neurons are used in the last layers.

The output of the convolution operation is usually run through a nonlinear activation function and then further modified by means of a pooling function, which replaces the output in a certain location with a value obtained from nearby outputs. This pooling function helps make the representation learned invariant to small translations of the input and performs subsampling of the input data. The most common pooling function is max pooling, which replaces the output with the maximum activation within a rectangular neighborhood. Convolution and pooling layers are stacked together to achieve feature learning in a hierarchical way. For example, when learning from images, layers closer to the input learn low-level feature representations (i.e., edges and corners) and those closer to the output learn higher level representations (i.e., contours and parts of objects). Once the features of interest have been learned, their activations are used in final layers, which are usually made up of fully connected neurons, to classify the input or perform value regression with it.

In contrast to MLPs, Recurrent Neural Networks (RNNs) are models in which the output is a function of not only the current inputs but also of the previous outputs, which are encoded into a hidden state . This means that RNNs have memory of the previous outputs and therefore can encode the information present in the sequence itself, something that MLPs cannot do. As a consequence, this type of model can be very useful to learn from sequential data. The memory is encoded into an internal state and updated as indicated in the following equation:where  represents the hidden state at time step . The weight matrices  (input-to-hidden) and  (hidden-to-hidden) determine the importance given to the current input and to the previous state, respectively. The activation is computed with a third weight matrix  (hidden-to-output) as indicated by the following equation:

RNNs are usually trained using Backpropagation Through Time (BPTT), an extension of backpropagation which takes into account temporality in order to compute the gradients. Using this method with long temporal sequences can lead to several issues. Gradients accumulated over a long sequence can become immeasurably large or extremely small. These problems are referred to as exploding gradients and vanishing gradients, respectively. Exploding gradients are easier to solve, as they can be truncated or squashed, whereas vanishing gradients can become too small for networks to learn from and for the resolution of a computer to enable its representation.

Long Short-Term Memory (LSTM) models are a type of RNN architecture proposed in 1997 by Hochreiter and Schmidhuber [17] which successfully overcomes the problem of vanishing gradients by maintaining a more constant error through the use of gated cells, which effectively allow for continuous learning over a larger number of time steps. A typical LSTM cell is depicted in Figure [3](https://www.hindawi.com/journals/js/2017/3296874/fig3/). The input, output, and forget gate vector activations in a standard LSTM are given as follows:

[](https://www.hindawi.com/journals/js/2017/3296874/fig3/" \t "https://www.hindawi.com/journals/js/2017/3296874/_blank)