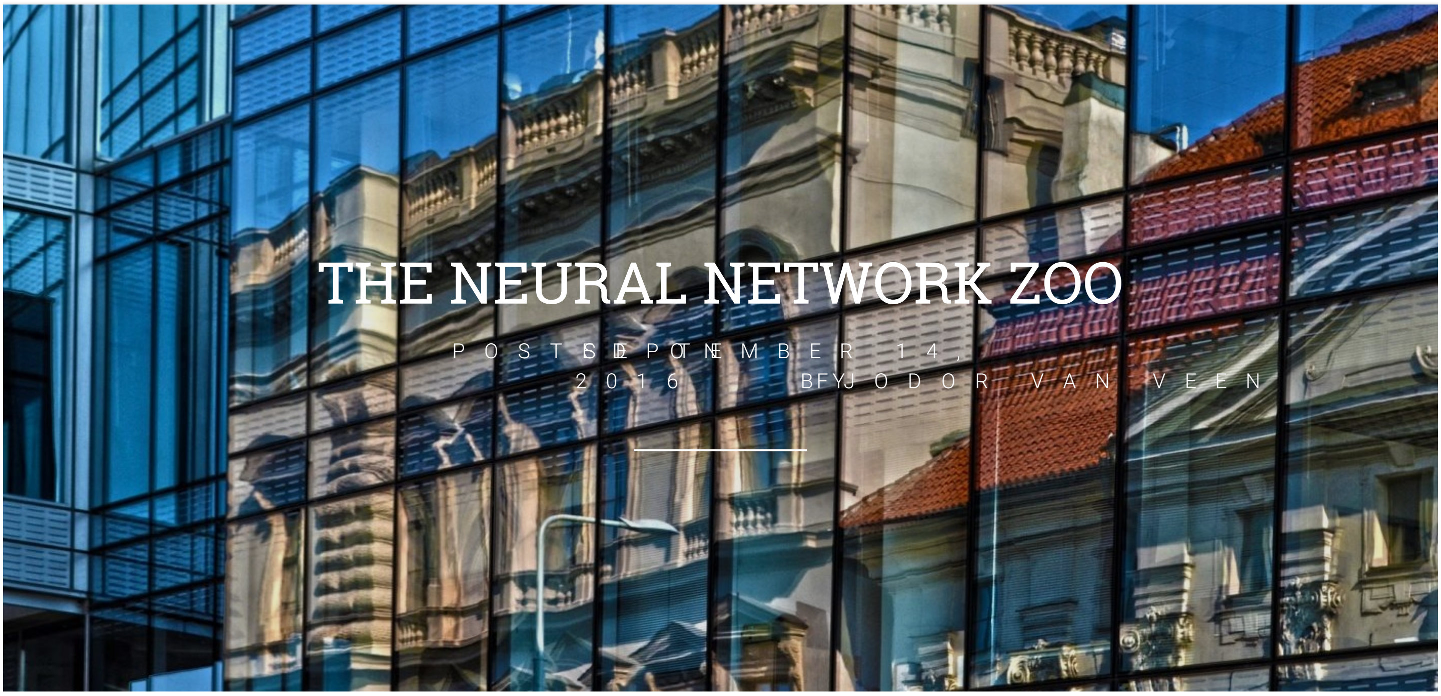
THE ASIMOV
INSTITUTE



THE NEURAL NETWORK ZOO

POSTED SEPTEMBER 14, 2016 BY FJODOR VAN VEEN

With new neural network architectures popping up every now and then, it’s hard to keep track of them all. Knowing all the abbreviations being thrown around (DCIGN, BiLSTM, DCGAN, anyone?) can be a bit overwhelming at first.

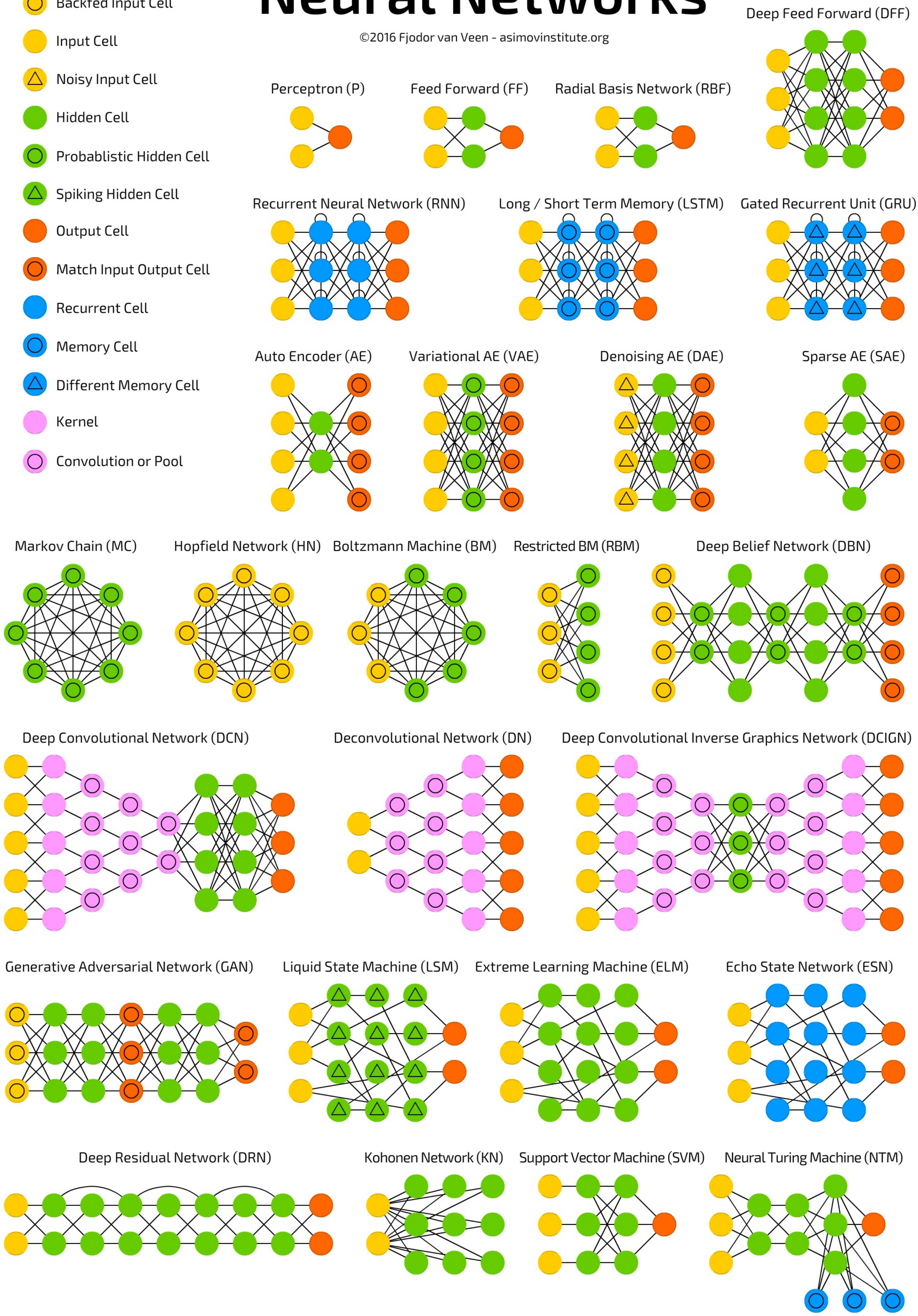
So I decided to compose a cheat sheet containing many of those architectures. Most of these are neural networks, some are completely different beasts. Though all of these architectures are presented as novel and unique, when I drew the node structures… their underlying relations started to make more sense.

A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- 🟡 Backfed Input Cell
- 🟡 Input Cell
- 🟡 Noisy Input Cell
- 🟢 Hidden Cell
- 🟢 Probablistic Hidden Cell
- 🟢 Spiking Hidden Cell
- 🟠 Output Cell
- 🟠 Match Input Output Cell
- 🔵 Recurrent Cell
- 🔵 Memory Cell
- 🔵 Different Memory Cell
- 🟣 Kernel
- 🟣 Convolution or Pool



Deep Feed Forward (DFF)

Perceptron (P)

Feed Forward (FF)

Radial Basis Network (RBF)

Recurrent Neural Network (RNN)

Long / Short Term Memory (LSTM)

Gated Recurrent Unit (GRU)

Auto Encoder (AE)

Variational AE (VAE)

Denoising AE (DAE)

Sparse AE (SAE)

Markov Chain (MC)

Hopfield Network (HN)

Boltzmann Machine (BM)

Restricted BM (RBM)

Deep Belief Network (DBN)

Deep Convolutional Network (DCN)

Deconvolutional Network (DN)

Deep Convolutional Inverse Graphics Network (DCIGN)

Generative Adversarial Network (GAN)

Liquid State Machine (LSM)

Extreme Learning Machine (ELM)

Echo State Network (ESN)

Deep Residual Network (DRN)

Kohonen Network (KN)

Support Vector Machine (SVM)

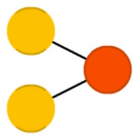
Neural Turing Machine (NTM)

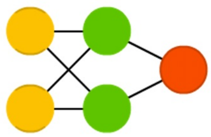
One problem with drawing them as node maps: it doesn't really show how they're used. For example, variational autoencoders (VAE) may look just like autoencoders (AE), but the training process is actually quite different. The use-cases for trained networks differ even more, because VAEs are generators, where you insert noise to get a new sample. AEs, simply map whatever they get as input to the closest training sample they "remember". I should add that this overview is in no way clarifying how each of the different node types work internally (but that's a topic for another day).

It should be noted that while most of the abbreviations used are generally accepted, not all of them are. RNNs sometimes refer to recursive neural networks, but most of the time they refer to recurrent neural networks. That's not the end of it though, in many places you'll find RNN used as placeholder for any recurrent architecture, including LSTMs, GRUs and even the bidirectional variants. AEs suffer from a similar problem from time to time, where VAEs and DAEs and the like are called simply AEs. Many abbreviations also vary in the amount of "N"s to add at the end, because you could call it a convolutional neural network but also simply a convolutional network (resulting in CNN or CN).

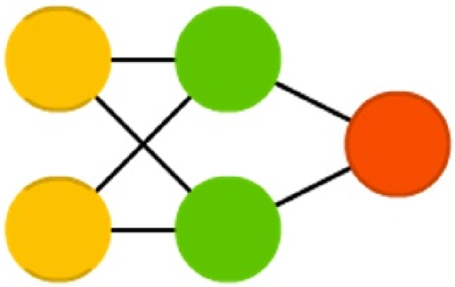
Composing a complete list is practically impossible, as new architectures are invented all the time. Even if published it can still be quite challenging to find them even if you're looking for them, or sometimes you just overlook some. So while this list may provide you with some insights into the world of AI, please, by no means take this list for being comprehensive; especially if you read this post long after it was written.

For each of the architectures depicted in the picture, I wrote a very, very brief description. You may find some of these to be useful if you're quite familiar with some architectures, but you aren't familiar with a particular one.

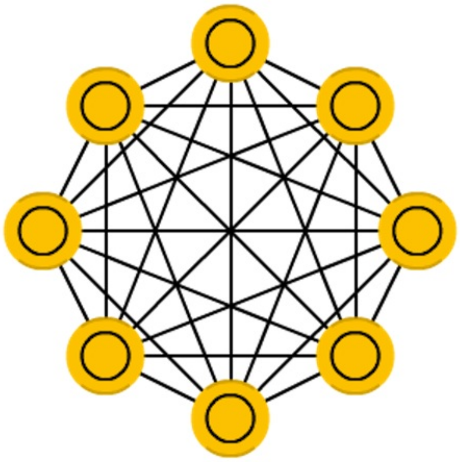




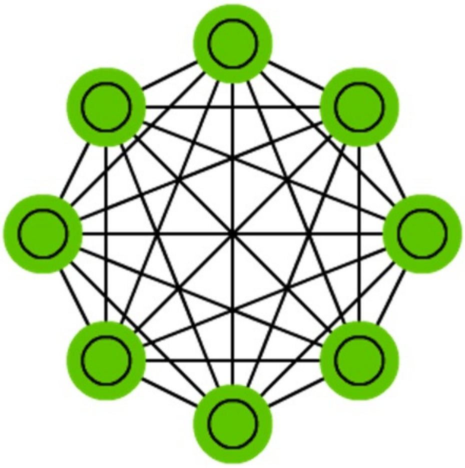
Feed forward neural networks (FF or FFNN) and perceptrons (P) are very straight forward, they feed information from the front to the back (input and output, respectively). Neural networks are often described as having layers, where each layer consists of either input, hidden or output cells in parallel. A layer alone never has connections and in general two adjacent layers are fully connected (every neuron form one layer to every neuron to another layer). The simplest somewhat practical network has two input cells and one output cell, which can be used to model logic gates. One usually trains FFNNs through back-propagation, giving the network paired datasets of "what goes in" and "what we want to have coming out". This is called supervised learning, as opposed to unsupervised learning where we only give it input and let the network fill in the blanks. The error being back-propagated is often some variation of the difference between the input and the output (like MSE or just the linear difference). Given that the network has enough hidden neurons, it can theoretically always model the relationship between the input and output. Practically their use is a lot more limited but they are popularly combined with other networks to form new networks.



Radial basis function (RBF) networks are FFNNs with radial basis functions as activation functions. There's nothing more to it. Doesn't mean they don't have their uses, but most FFNNs with other activation functions don't get their own name. This mostly has to do with inventing them at the right time.

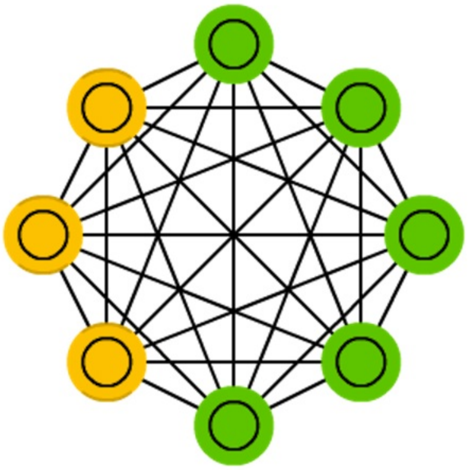


A Hopfield network (HN) is a network where every neuron is connected to every other neuron; it is a completely entangled plate of spaghetti as even all the nodes function as everything. Each node is input before training, then hidden during training and output afterwards. The networks are trained by setting the value of the neurons to the desired pattern after which the weights can be computed. The weights do not change after this. Once trained for one or more patterns, the network will always converge to one of the learned patterns because the network is only stable in those states. Note that it does not always conform to the desired state (it's not a magic black box sadly). It stabilises in part due to the total "energy" or "temperature" of the network being reduced incrementally during training. Each neuron has an activation threshold which scales to this temperature, which if surpassed by summing the input causes the neuron to take the form of one of two states (usually -1 or 1, sometimes 0 or 1). Updating the network can be done synchronously or more commonly one by one. If updated one by one, a fair random sequence is created to organise which cells update in what order (fair random being all options (n) occurring exactly once every n items). This is so you can tell when the network is stable (done converging), once every cell has been updated and none of them changed, the network is stable (annealed). These networks are often called associative memory because the converge to the most similar state as the input; if humans see half a table we can image the other half, this network will converge to a table if presented with half noise and half a table.

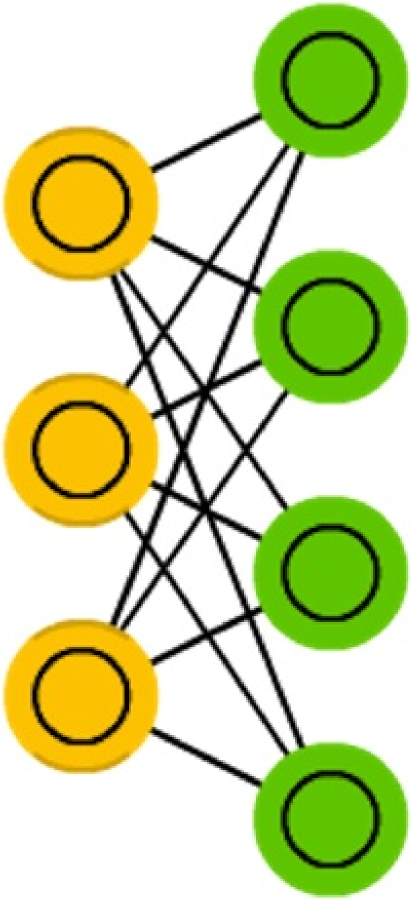


Markov chains (MC or discrete time Markov Chain, DTMC) are kind of the predecessors to BMs and HNs.

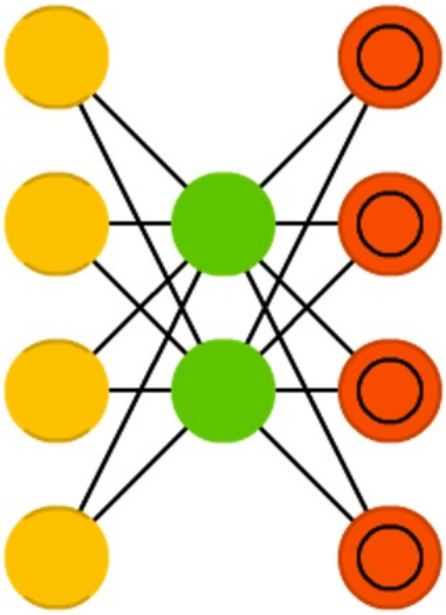
They can be understood as follows: from this node where I am now, what are the odds of me going to any of my neighbouring nodes? They are memoryless (i.e. Markov Property) which means that every state you end up in depends completely on the previous state. While not really a neural network, they do resemble neural networks and form the theoretical basis for BMs and HNs. MC aren't always considered neural networks, as goes for BMs, RBMs and HNs. Markov chains aren't always fully connected either.



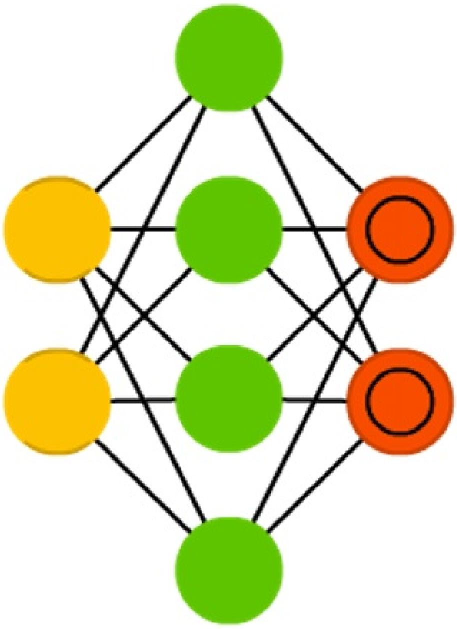
Boltzmann machines (BM) are a lot like HNs, but: some neurons are marked as input neurons and others remain "hidden". The input neurons become output neurons at the end of a full network update. It starts with random weights and learns through back-propagation, or more recently through contrastive divergence (a Markov chain is used to determine the gradients between two informational gains). Compared to a HN, the neurons mostly have binary activation patterns. As hinted by being trained by MCs, BMs are stochastic networks. The training and running process of a BM is fairly similar to a HN: one sets the input neurons to certain clamped values after which the network is set free (it doesn't get a sock). While free the cells can get any value and we repetitively go back and forth between the input and hidden neurons. The activation is controlled by a global temperature value, which if lowered lowers the energy of the cells. This lower energy causes their activation patterns to stabilise. The network reaches an equilibrium given the right temperature.



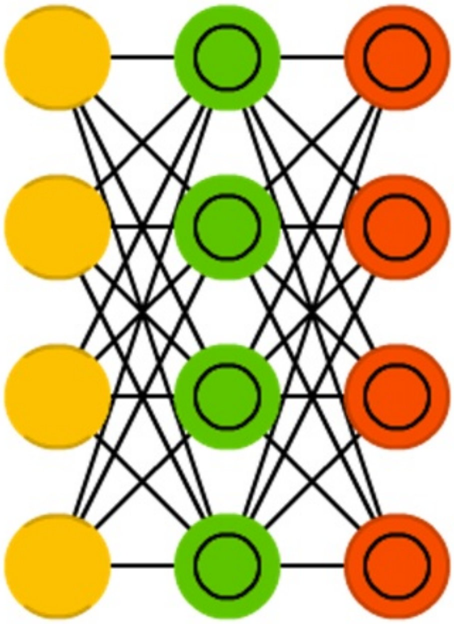
Restricted Boltzmann machines (RBM) are remarkably similar to BMs (surprise) and therefore also similar to HNs. The biggest difference between BMs and RBMs is that RBMs are a better usable because they are more restricted. They don’t trigger-happily connect every neuron to every other neuron but only connect every different group of neurons to every other group, so no input neurons are directly connected to other input neurons and no hidden to hidden connections are made either. RBMs can be trained like FFNNs with a twist: instead of passing data forward and then back-propagating, you forward pass the data and then backward pass the data (back to the first layer). After that you train with forward-and-back-propagation.



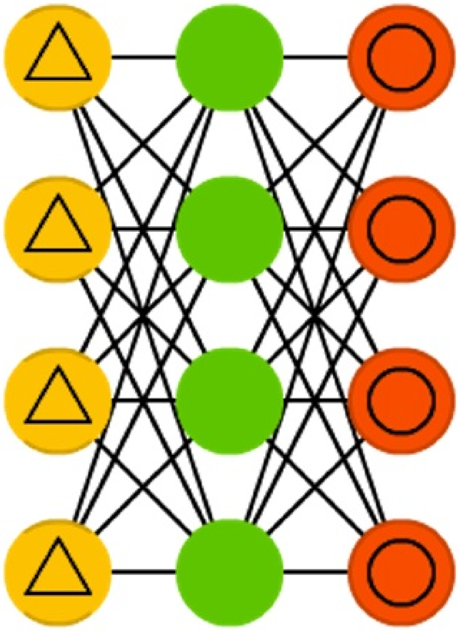
Autoencoders (AE) are somewhat similar to FFNNs as AEs are more like a different use of FFNNs than a fundamentally different architecture. The basic idea behind autoencoders is to encode information (as in compress, not encrypt) automatically, hence the name. The entire network always resembles an hourglass like shape, with smaller hidden layers than the input and output layers. AEs are also always symmetrical around the middle layer(s) (one or two depending on an even or odd amount of layers). The smallest layer(s) is|are almost always in the middle, the place where the information is most compressed (the chokepoint of the network). Everything up to the middle is called the encoding part, everything after the middle the decoding and the middle (surprise) the code. One can train them using backpropagation by feeding input and setting the error to be the difference between the input and what came out. AEs can be built symmetrically when it comes to weights as well, so the encoding weights are the same as the decoding weights.



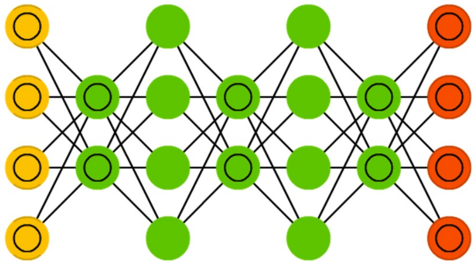
Sparse autoencoders (SAE) are in a way the opposite of AEs. Instead of teaching a network to represent a bunch of information in less "space" or nodes, we try to encode information in more space. So instead of the network converging in the middle and then expanding back to the input size, we blow up the middle. These types of networks can be used to extract many small features from a dataset. If one were to train a SAE the same way as an AE, you would in almost all cases end up with a pretty useless identity network (as in what comes in is what comes out, without any transformation or decomposition). To prevent this, instead of feeding back the input, we feed back the input plus a sparsity driver. This sparsity driver can take the form of a threshold filter, where only a certain error is passed back and trained, the other error will be "irrelevant" for that pass and set to zero. In a way this resembles spiking neural networks, where not all neurons fire all the time (and points are scored for biological plausibility).



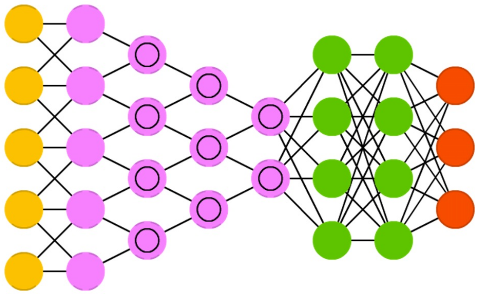
Variational autoencoders (VAE) have the same architecture as AEs but are "taught" something else: an approximated probability distribution of the input samples. It's a bit back to the roots as they are bit more closely related to BMs and RBMs. They do however rely on Bayesian mathematics regarding probabilistic inference and independence, as well as a re-parametrisation trick to achieve this different representation. The inference and independence parts make sense intuitively, but they rely on somewhat complex mathematics. The basics come down to this: take influence into account. If one thing happens in one place and something else happens somewhere else, they are not necessarily related. If they are not related, then the error propagation should consider that. This is a useful approach because neural networks are large graphs (in a way), so it helps if you can rule out influence from some nodes to other nodes as you dive into deeper layers.



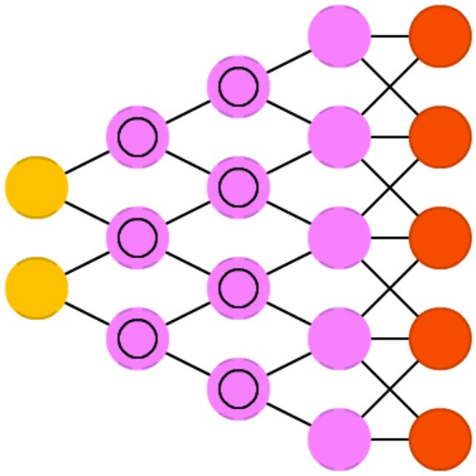
Denoising autoencoders (DAE) are AEs where we don't feed just the input data, but we feed the input data with noise (like making an image more grainy). We compute the error the same way though, so the output of the network is compared to the original input without noise. This encourages the network not to learn details but broader features, as learning smaller features often turns out to be "wrong" due to it constantly changing with noise.



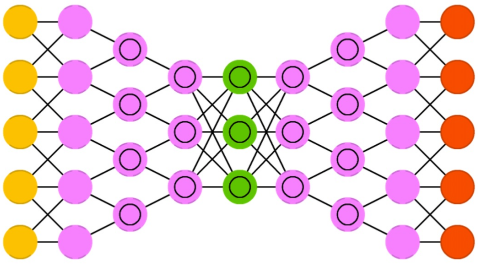
Deep belief networks (DBN) is the name given to stacked architectures of mostly RBMs or VAEs. These networks have been shown to be effectively trainable stack by stack, where each AE or RBM only has to learn to encode the previous network. This technique is also known as greedy training, where greedy means making locally optimal solutions to get to a decent but possibly not optimal answer. DBNs can be trained through contrastive divergence or back-propagation and learn to represent the data as a probabilistic model, just like regular RBMs or VAEs. Once trained or converged to a (more) stable state through unsupervised learning, the model can be used to generate new data. If trained with contrastive divergence, it can even classify existing data because the neurons have been taught to look for different features.



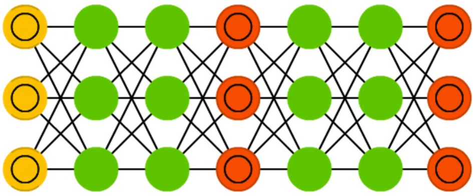
Convolutional neural networks (CNN or deep convolutional neural networks, DCNN) are quite different from most other networks. They are primarily used for image processing but can also be used for other types of input such as as audio. A typical use case for CNNs is where you feed the network images and the network classifies the data, e.g. it outputs "cat" if you give it a cat picture and "dog" when you give it a dog picture. CNNs tend to start with an input "scanner" which is not intended to parse all the training data at once. For example, to input an image of 200 x 200 pixels, you wouldn't want a layer with 40 000 nodes. Rather, you create a scanning input layer of say 20 x 20 which you feed the first 20 x 20 pixels of the image (usually starting in the upper left corner). Once you passed that input (and possibly use it for training) you feed it the next 20 x 20 pixels: you move the scanner one pixel to the right. Note that one wouldn't move the input 20 pixels (or whatever scanner width) over, you're not dissecting the image into blocks of 20 x 20, but rather you're crawling over it. This input data is then fed through convolutional layers instead of normal layers, where not all nodes are connected to all nodes. Each node only concerns itself with close neighbouring cells (how close depends on the implementation, but usually not more than a few). These convolutional layers also tend to shrink as they become deeper, mostly by easily divisible factors of the input (so 20 would probably go to a layer of 10 followed by a layer of 5). Powers of two are very commonly used here, as they can be divided cleanly and completely by definition: 32, 16, 8, 4, 2, 1. Besides these convolutional layers, they also often feature pooling layers. Pooling is a way to filter out details: a commonly found pooling technique is max pooling, where we take say 2 x 2 pixels and pass on the pixel with the most amount of red. To apply CNNs for audio, you basically feed the input audio waves and inch over the length of the clip, segment by segment. Real world implementations of CNNs often glue an FFNN to the end to further process the data, which allows for highly non-linear abstractions. These networks are called DCNNs but the names and abbreviations between these two are often used interchangeably.



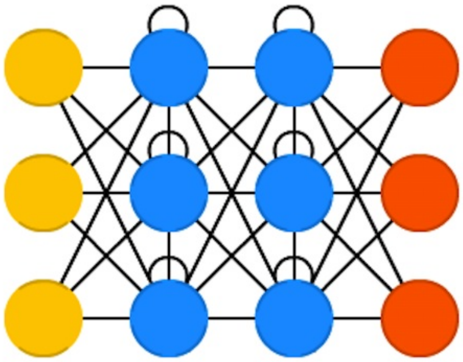
Deconvolutional networks (DN), also called inverse graphics networks (IGNs), are reversed convolutional neural networks. Imagine feeding a network the word "cat" and training it to produce cat-like pictures, by comparing what it generates to real pictures of cats. DNNs can be combined with FFNNs just like regular CNNs, but this is about the point where the line is drawn with coming up with new abbreviations. They may be referenced as deep deconvolutional neural networks, but you could argue that when you stick FFNNs to the back and the front of DNNs that you have yet another architecture which deserves a new name. Note that in most applications one wouldn't actually feed text-like input to the network, more likely a binary classification input vector. Think <0, 1> being cat, <1, 0> being dog and <1, 1> being cat and dog. The pooling layers commonly found in CNNs are often replaced with similar inverse operations, mainly interpolation and extrapolation with biased assumptions (if a pooling layer uses max pooling, you can invent exclusively lower new data when reversing it).



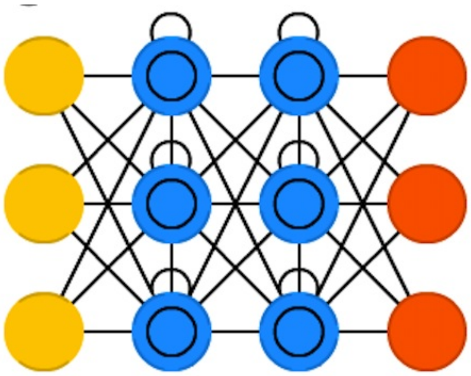
Deep convolutional inverse graphics networks (DCIGN) have a somewhat misleading name, as they are actually VAEs but with CNNs and DNNs for the respective encoders and decoders. These networks attempt to model "features" in the encoding as probabilities, so that it can learn to produce a picture with a cat and a dog together, having only ever seen one of the two in separate pictures. Similarly, you could feed it a picture of a cat with your neighbours' annoying dog on it, and ask it to remove the dog, without ever having done such an operation. Demo's have shown that these networks can also learn to model complex transformations on images, such as changing the source of light or the rotation of a 3D object. These networks tend to be trained with back-propagation.



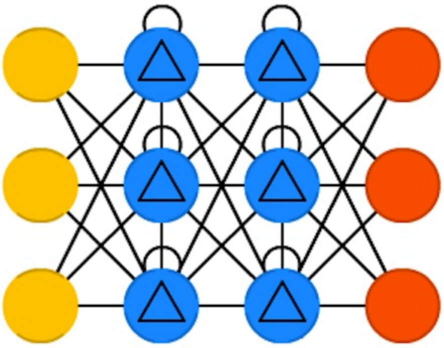
Generative adversarial networks (GAN) are from a different breed of networks, they are twins: two networks working together. GANs consist of any two networks (although often a combination of FFs and CNNs), with one tasked to generate content and the other has to judge content. The discriminating network receives either training data or generated content from the generative network. How well the discriminating network was able to correctly predict the data source is then used as part of the error for the generating network. This creates a form of competition where the discriminator is getting better at distinguishing real data from generated data and the generator is learning to become less predictable to the discriminator. This works well in part because even quite complex noise-like patterns are eventually predictable but generated content similar in features to the input data is harder to learn to distinguish. GANs can be quite difficult to train, as you don't just have to train two networks (either of which can pose it's own problems) but their dynamics need to be balanced as well. If prediction or generation becomes to good compared to the other, a GAN won't converge as there is intrinsic divergence.



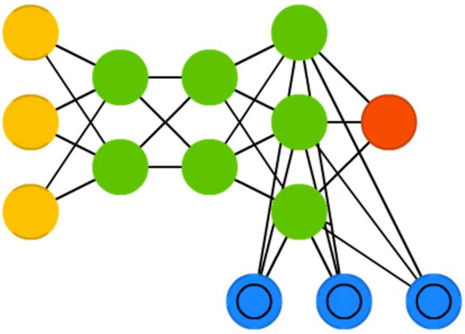
Recurrent neural networks (RNN) are FFNNs with a time twist: they are not stateless; they have connections between passes, connections through time. Neurons are fed information not just from the previous layer but also from themselves from the previous pass. This means that the order in which you feed the input and train the network matters: feeding it "milk" and then "cookies" may yield different results compared to feeding it "cookies" and then "milk". One big problem with RNNs is the vanishing (or exploding) gradient problem where, depending on the activation functions used, information rapidly gets lost over time, just like very deep FFNNs lose information in depth. Intuitively this wouldn't be much of a problem because these are just weights and not neuron states, but the weights through time is actually where the information from the past is stored; if the weight reaches a value of 0 or 1 000 000, the previous state won't be very informative. RNNs can in principle be used in many fields as most forms of data that don't actually have a timeline (i.e. unlike sound or video) can be represented as a sequence. A picture or a string of text can be fed one pixel or character at a time, so the time dependent weights are used for what came before in the sequence, not actually from what happened x seconds before. In general, recurrent networks are a good choice for advancing or completing information, such as autocompletion.



Long / short term memory (LSTM) networks try to combat the vanishing / exploding gradient problem by introducing gates and an explicitly defined memory cell. These are inspired mostly by circuitry, not so much biology. Each neuron has a memory cell and three gates: input, output and forget. The function of these gates is to safeguard the information by stopping or allowing the flow of it. The input gate determines how much of the information from the previous layer gets stored in the cell. The output layer takes the job on the other end and determines how much of the next layer gets to know about the state of this cell. The forget gate seems like an odd inclusion at first but sometimes it's good to forget: if it's learning a book and a new chapter begins, it may be necessary for the network to forget some characters from the previous chapter. LSTMs have been shown to be able to learn complex sequences, such as writing like Shakespeare or composing primitive music. Note that each of these gates has a weight to a cell in the previous neuron, so they typically require more resources to run.

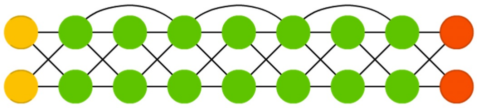


Gated recurrent units (GRU) are a slight variation on LSTMs. They have one less gate and are wired slightly differently: instead of an input, output and a forget gate, they have an update gate. This update gate determines both how much information to keep from the last state and how much information to let in from the previous layer. The reset gate functions much like the forget gate of an LSTM but it's located slightly differently. They always send out their full state, they don't have an output gate. In most cases, they function very similarly to LSTMs, with the biggest difference being that GRUs are slightly faster and easier to run (but also slightly less expressive). In practice these tend to cancel each other out, as you need a bigger network to regain some expressiveness which then in turn cancels out the performance benefits. In some cases where the extra expressiveness is not needed, GRUs can outperform LSTMs.

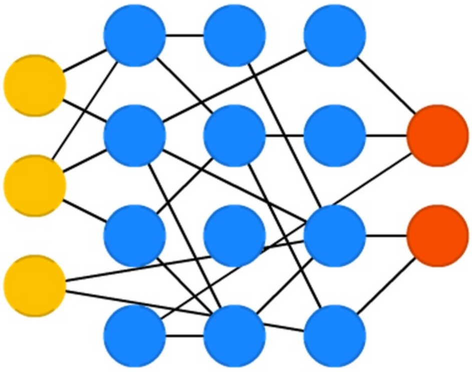


Neural Turing machines (NTM) can be understood as an abstraction of LSTMs and an attempt to un-black-box neural networks (and give us some insight in what is going on in there). Instead of coding a memory cell directly into a neuron, the memory is separated. It's an attempt to combine the efficiency and permanency of regular digital storage and the efficiency and expressive power of neural networks. The idea is to have a content-addressable memory bank and a neural network that can read and write from it. The "Turing" in Neural Turing Machines comes from them being Turing complete: the ability to read and write and change state based on what it reads means it can represent anything a Universal Turing Machine can represent.

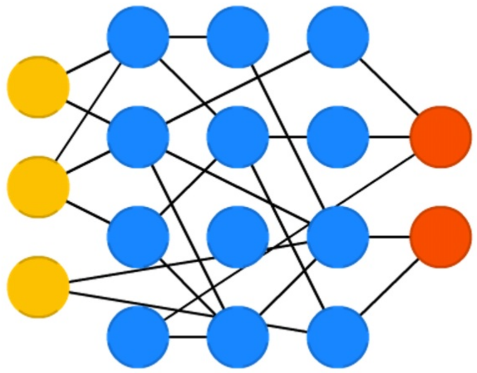
Bidirectional recurrent neural networks, bidirectional long / short term memory networks and bidirectional gated recurrent units (BiRNN, BiLSTM and BiGRU respectively) are not shown on the chart because they look exactly the same as their unidirectional counterparts. The difference is that these networks are not just connected to the past, but also to the future. As an example, unidirectional LSTMs might be trained to predict the word “fish” by being fed the letters one by one, where the recurrent connections through time remember the last value. A BiLSTM would also be fed the next letter in the sequence on the backward pass, giving it access to future information. This trains the network to fill in gaps instead of advancing information, so instead of expanding an image on the edge, it could fill a hole in the middle of an image.



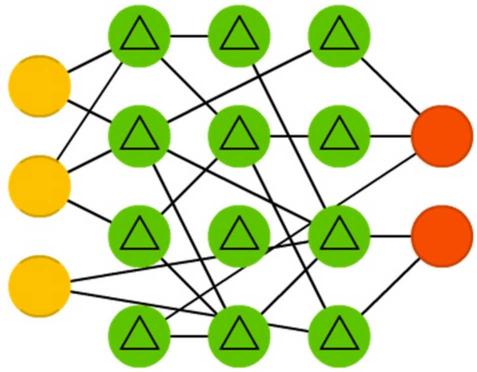
Deep residual networks (DRN) are very deep FFNNs with extra connections passing input from one layer to a later layer (often 2 to 5 layers) as well as the next layer. Instead of trying to find a solution for mapping some input to some output across say 5 layers, the network is enforced to learn to map some input to some output + some input. Basically, it adds an identity to the solution, carrying the older input over and serving it freshly to a later layer. It has been shown that these networks are very effective at learning patterns up to 150 layers deep, much more than the regular 2 to 5 layers one could expect to train. However, it has been proven that these networks are in essence just RNNs without the explicit time based construction and they’re often compared to LSTMs without gates.



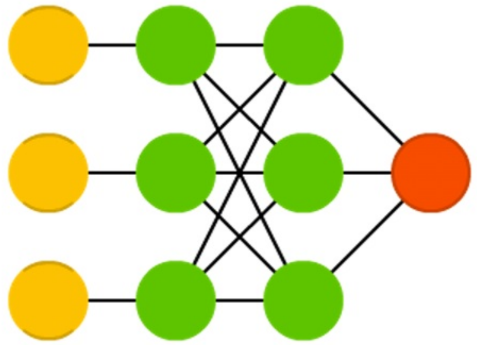
Echo state networks (ESN) are yet another different type of (recurrent) network. This one sets itself apart from others by having random connections between the neurons (i.e. not organised into neat sets of layers), and they are trained differently. Instead of feeding input and back-propagating the error, we feed the input, forward it and update the neurons for a while, and observe the output over time. The input and the output layers have a slightly unconventional role as the input layer is used to prime the network and the output layer acts as an observer of the activation patterns that unfold over time. During training, only the connections between the observer and the (soup of) hidden units are changed.



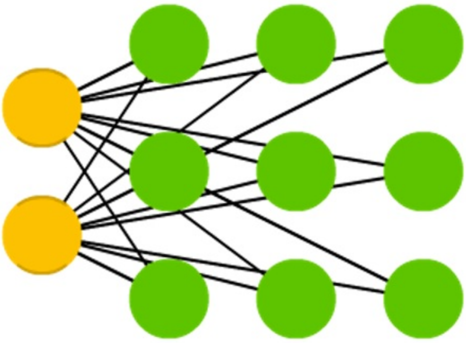
Extreme learning machines (ELM) are basically FFNNs but with random connections. They look very similar to LSMs and ESNs, but they are used more like FFNNs. This is not just because they are not recurrent nor spiking, but also because these use simple backpropagation through the entire network, instead of dealing with the input/output.



Liquid state machines (LSM) are similar soups, looking a lot like ESNs. The real difference is that LSMs are a type of spiking neural networks: sigmoid activations are replaced with threshold functions and each neuron is also an accumulating memory cell. So when updating a neuron, the value is not set to the sum of the neighbours, but rather added to itself. Once the threshold is reached, it releases its' energy to other neurons. This creates a spiking like pattern, where nothing happens for a while until a threshold is suddenly reached.



Support vector machines (SVM) find optimal solutions for classification problems. Classically they were only capable of categorising linearly separable data; say finding which images are of Garfield and which of Snoopy, with any other outcome not being possible. During training, SVMs can be thought of as plotting all the data (Garfields and Snoopys) on a graph (2D) and figuring out how to draw a line between the data points. This line would separate the data, so that all Snoopys are on one side and the Garfields on the other. This line moves to an optimal line in such a way that the margins between the data points and the line are maximised on both sides. Classifying new data would be done by plotting a point on this graph and simply looking on which side of the line it is (Snoopy side or Garfield side). Using the kernel trick, they can be taught to classify n-dimensional data. This entails plotting points in a 3D plot, allowing it to distinguish between Snoopy, Garfield AND Simon's cat, or even higher dimensions distinguishing even more cartoon characters. SVMs are not always considered neural networks.



And finally, **Kohonen networks (KN, also self organising (feature) map, SOM, SOFM)** "complete" our zoo. KNs utilise competitive learning to classify data without supervision. Input is presented to the network, after which the network assesses which of its neurons most closely match that input. These neurons are then adjusted to match the input even better, dragging along their neighbours in the process. How much the neighbours are moved depends on the distance of the neighbours to the best matching units. KNs are sometimes not considered neural networks either.

Any feedback and criticism is welcome. At the Asimov Institute we do deep learning research and development, so be sure to follow us on twitter for future updates and posts! Thank you for reading!

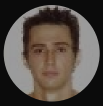
[UPDATE 15 sept 2016] I would like to thank everybody for their insights and corrections, all feedback is hugely appreciated. I will add links and a couple more suggested networks in a future update, stay tuned.

Click here for a high res version of the full image.

DEEP LEARNING

‹ ARTISTIC STYLE TRANSFER BLEND

Type your comment here...



VOLKAN CIRIK

SEP 14, 2016

how about recursive neural networks?

REPLY

FJODOR VAN VEEN

SEP 14, 2016

Interesting, another branch of networks. Will definitely incorporate them in a potential follow up post!

REPLY

COLIN MORRIS

SEP 14, 2016

Awesome. Minor correction regarding Boltzmann machines. “Compared to a HN, the neurons also sometimes have binary activation patterns but at other times they are stochastic”. The units are always stochastic, and usually also binary. But there are variations where units are instead Gaussian, binomial, etc.

REPLY

FJODOR VAN VEEN

SEP 14, 2016

Good stuff. Thank you for your feedback!

REPLY

B@RMALEY.E>

SEP 14, 2016

I don’t understand why in Markov chains you have a fully-connected graph. It should be either a chain, or a generalized “chain” where previous m nodes have edges to current node (if we’re talking about m-order markov chains)

REPLY

FJODOR VAN VEEN
SEP 14, 2016

Thank you for your feedback! When you say chain, you mean something like this?
http://blogs.canisius.edu/mathblog/wp-content/uploads/sites/31/2013/02/markovchainpic3.png
Or more like this? http://1.bp.blogspot.com/-uLVbBbNTDTI/TbWSnEa4ZpI/AAAAAAAAABY/XYrTCXRhSkQ/s1600/markovchain.png
Note that all architectures have a rather finite representation here. As pointed out elsewhere, the DAEs often have a complete or overcomplete hidden layer, but not always. Most examples I can find are either arbitrary chains or fully connected graphs.

REPLY

B@RMALEY.EXE
SEP 15, 2016

Yes, something like this but without loops. Basically a bunch of probabilistic cells (if you make them hidden, you get a HMM) chained in a linear way.

REPLY

FJODOR VAN VEEN
SEP 15, 2016

Hmm, will take that into consideration. The cells themselves are not probabilistic though, the connections between them are. I'm considering giving a few example networks for some architectures, especially the ones that vary greatly like this one.

DANIEL FAY
SEP 14, 2016

Cool! I think you could give the denoising autoencoder a higher-dimensional hidden layer since it doesn't need a bottleneck.

REPLY

FJODOR VAN VEEN

SEP 14, 2016

From what I understand, DAEs are simply AEs but used differently, which themselves are just symmetric FFNNs used differently. It is true that DAEs are often either complete or overcomplete, since overcomplete AEs usually learn identity; this is what DAEs are trying to prevent. The size restrictions are rarely explicitly defined though. Thank you for reading!

REPLY

JIM

SEP 15, 2016

This reply is a hidden layer

REPLY

STYLIANOS IORDANIS

SEP 15, 2016

Excellent work!
Could you please enhance the article by adding and presenting the Hidden Markov models using exactly the same approach ? That would be very enlightening, and I am very curious to read your explanation. Thank you

REPLY

FJODOR VAN VEEN

SEP 15, 2016

Working on it [:

MUHONG GUO

SEP 15, 2016

Awesome! Could you also add some references for each network?

REPLY

FJODOR VAN VEEN

SEP 15, 2016

I’d love to, if I can find the time [:

REPLY

SUMMIT SUEN

SEP 15, 2016

Awesome work! Btw, should the middle neurons in the Deep convolutional inverse graphics networks (DCIGN) be probabilistic hidden cells?

REPLY

FJODOR VAN VEEN

SEP 15, 2016

Yes. Good observation, will correct!

REPLY

KRISTOFER

SEP 15, 2016

Very nice summary!

From the pictures of the denoising autoencoder and the variational autoencoder, it looks like they also have to compress the data (since the hidden layer has fewer elements than the input and output layers), just the ordinary autoencoder. Is that the case?

Also, is there some specific name for the ordinary autoencoder to let people know that you are talking about an autoencoder that compresses the data? Perhaps "compressive autoencoder"? To me, the term "autoencoder" includes all kinds of autoencoders, i.e. also denoising, variational and sparse autoencoders, not just "compressive" (?) autoencoders. (So to me it feels a bit wrong to talk about "autoencoders" like all of them compress the data.)

REPLY

FJODOR VAN VEEN
SEP 15, 2016

A common problem with the networks. RNN stands for either recurrent NN or recursive NN. Worse yet, people use RNN to indicate LSTM, GRU, BiGRU and BiLSTM. And no, there is no name for that.

And yes, a problem with this "one shot" representation is that you cannot capture all uses. AE, VAE, SAE and DAE are all autoencoders, each of which somehow tries to reconstruct their input. Only SAEs almost always have an overcomplete code (more hidden neurons than input / output), but the others can have compressed, complete or overcomplete codes.

Thank you for your feedback!

REPLY

DJEB
SEP 15, 2016

Amazing. What software did you used to plot these figures ? Cheers !

REPLY

FJODOR VAN VEEN

SEP 15, 2016

I drew them in Adobe Animate, they're not plots. Yes it was a lot of work to draw the lines.

REPLY

ROB

SEP 15, 2016

Never thought of SVMs as a network, though they can be used that way. This seems to be a different take on SVMs than I've heard before.

REPLY

PHILIP REGENIE

SEP 15, 2016

Absolutely the best article and classification I have read in 10 years. Clearly written, easily understood. Points anyone reading to research those areas applicable to their problem set.

REPLY

JAGANNATH RAJAGOPAL

SEP 15, 2016

One of the most useful and comprehensive posts I've seen in a while. Very nicely done 🙂

I'm the creator of Deeplearning.TV on YouTube. If you would like to follow this up with a series of videos explaining this, I would be honored to collaborate with you 🙂

REPLY

YORK FUN
SEP 15, 2016

Very nice work！
I have a question，why svm could be seen as as a 3 level network，and the input cells are not full connected to the first hiden level, what's that mean?

REPLY

FJODOR VAN VEEN
SEP 15, 2016

The idea behind deeper SVMs is that they allow for classification tasks more complex than binary. Usually it would just be the directly one-to-one connected stuff as seen in the first layer. It's set up like that because each neuron in the input represents a point in the possibility space, and the network tries to separate the inputs with a margin as large as possible.

REPLY

YORK FUN
SEP 16, 2016

Thanks for your quick reply！
Could you please give me some reference papers about
svm-network, i 'm interested in this research how neural network could union these algorthims.

REPLY

CONIC
SEP 15, 2016

Can you eventually give a link to a high resolution image of these networks? I was thinking about getting a poster printed for myself.

REPLY

FJODOR VAN VEEN

SEP 15, 2016

Cool! See the bottom of the post.

REPLY

ABATESINS

SEP 15, 2016

Nice job, summarizing and representing all of these! One observation regarding GANs is that the discriminator, as originally conceived, has only two output neurons (whether the sample passed to it came from the true distribution or from the generated one). The corresponding graphic shows three.

REPLY

FJODOR VAN VEEN

SEP 15, 2016

Yes. On it. Although you could also make do with one, I think I'll draw it with two. Thank you!

REPLY

ABATESINS

SEP 15, 2016

You're right! The discriminator does actually a regression on that one decision so one output is indeed more precise (although having two neurons drawn looks more representative to me)

REPLY

FJODOR VAN VEEN

SEP 15, 2016

Fixed it.

REPLY

DANIJAR HAFNER

SEP 15, 2016

Hi, thanks for the very nice visualization! A common mistake with RNNs is to not connect neurons within the same layer. While each LSTM neuron has its own hidden state, its output feeds back to all neurons in the current layer. The mistake also appears here.

REPLY

FJODOR VAN VEEN

SEP 15, 2016

Keen eye! Forgot to draw the lines, but very aware of the fine workings. It will make it a spiderweb of lines, but eh [:
[UPDATE] fixed

REPLY

GARRETT SMITH

SEP 15, 2016

Are you excellent images available for reuse under a particular license? Do you have an attribution policy?

REPLY

FJODOR VAN VEEN
SEP 16, 2016

As long as you mention the author and link to the Asimov Institute, use them however and wherever you like!

REPLY

JOHN M DANSKIN
SEP 16, 2016

Hi Fjodor, thanks for the presentation. As a nit, if you check out your page with one of the many color-blindness filters, you'll see that the green and gold circles are indistinguishable to people with red-green color-blindness. I have workarounds, especially the color-shifting google plugin this page inspired me to find, but others might not. Color-blindness affects about 8% of men. Again, thanks for the presentation!

REPLY

FJODOR VAN VEEN
SEP 16, 2016

If you look closely, you'll find only completely blind people will not be able to read the image. There are slight lines on the circle edges with unique patterns for each of the five different colours.

REPLY

THE ASIMOV INSTITUTE
FOR ARTIFICIAL CREATIVITY & CONST

BLOG

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 …

© 2016, THE ASIMOV INSTITUTE
PROUDLY POWERED BY WORDPRESS
THEME: FYEULUIXT ADOBRYNER