What is Deep Learning?

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

Deep Learning is Large Neural Networks

Andrew Ng from Coursera and Chief Scientist at Baidu Research formally founded Google Brain that eventually resulted in the productization of deep learning technologies across a large number of Google services.

He has spoken and written a lot about what deep learning is and is a good place to start.

In early talks on deep learning, Andrew described deep learning in the context of traditional artificial neural networks. In the 2013 talk titled “Deep Learning, Self-Taught Learning and Unsupervised Feature Learning” he described the idea of deep learning as:

Using brain simulations, hope to:

– Make learning algorithms much better and easier to use.

– Make revolutionary advances in machine learning and AI.

I believe this is our best shot at progress towards real AI

Later his comments became more nuanced.

The core of deep learning according to Andrew is that we now have fast enough computers and enough data to actually train large neural networks. When discussing why now is the time that deep learning is taking off at ExtractConf 2015 in a talk titled “What data scientists should know about deep learning“, he commented:

very large neural networks we can now have and … huge amounts of data that we have access to

He also commented on the important point that it is all about scale. That as we construct larger neural networks and train them with more and more data, their performance continues to increase. This is generally different to other machine learning techniques that reach a plateau in performance.

for most flavors of the old generations of learning algorithms … performance will plateau. … deep learning … is the first class of algorithms … that is scalable. … performance just keeps getting better as you feed them more data

Finally, he is clear to point out that the benefits from deep learning that we are seeing in practice come from supervised learning. From the 2015 ExtractConf talk, he commented:

almost all the value today of deep learning is through supervised learning or learning from labeled data

Earlier at a talk to Stanford University titled “Deep Learning” in 2014 he made a similar comment:

one reason that deep learning has taken off like crazy is because it is fantastic at supervised learning

Andrew often mentions that we should and will see more benefits coming from the unsupervised side of the tracks as the field matures to deal with the abundance of unlabeled data available.

Jeff Dean is a Wizard and Google Senior Fellow in the Systems and Infrastructure Group at Google and has been involved and perhaps partially responsible for the scaling and adoption of deep learning within Google. Jeff was involved in the Google Brain project and the development of large-scale deep learning software DistBelief and later TensorFlow.

In a 2016 talk titled “Deep Learning for Building Intelligent Computer Systems” he made a comment in the similar vein, that deep learning is really all about large neural networks.

When you hear the term deep learning, just think of a large deep neural net. Deep refers to the number of layers typically and so this kind of the popular term that’s been adopted in the press. I think of them as deep neural networks generally.

He has given this talk a few times, and in a modified set of slides for the same talk, he highlights the scalability of neural networks indicating that results get better with more data and larger models, that in turn require more computation to train.

In addition to scalability, another often cited benefit of deep learning models is their ability to perform automatic feature extraction from raw data, also called feature learning.

Yoshua Bengio is another leader in deep learning although began with a strong interest in the automatic feature learning that large neural networks are capable of achieving.

He describes deep learning in terms of the algorithms ability to discover and learn good representations using feature learning. In his 2012 paper titled “Deep Learning of Representations for Unsupervised and Transfer Learning” he commented:

Deep learning algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features

An elaborated perspective of deep learning along these lines is provided in his 2009 technical report titled “Learning deep architectures for AI” where he emphasizes the importance the hierarchy in feature learning.

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features.

In the soon to be published book titled “Deep Learning” co-authored with Ian Goodfellow and Aaron Courville, they define deep learning in terms of the depth of the architecture of the models.

The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning.

This is an important book and will likely become the definitive resource for the field for some time. The book goes on to describe multilayer perceptrons as an algorithm used in the field of deep learning, giving the idea that deep learning has subsumed artificial neural networks.

The quintessential example of a deep learning model is the feedforward deep network or multilayer perceptron (MLP).

Peter Norvig is the Director of Research at Google and famous for his textbook on AI titled “Artificial Intelligence: A Modern Approach“.

In a 2016 talk he gave titled “Deep Learning and Understandability versus Software Engineering and Verification” he defined deep learning in a very similar way to Yoshua, focusing on the power of abstraction permitted by using a deeper network structure.

a kind of learning where the representation you form have several levels of abstraction, rather than a direct input to output

Why Call it “Deep Learning”?
Why Not Just “Artificial Neural Networks”?

Geoffrey Hinton is a pioneer in the field of artificial neural networks and co-published the first paper on the backpropagation algorithm for training multilayer perceptron networks.

He may have started the introduction of the phrasing “deep” to describe the development of large artificial neural networks.

He co-authored a paper in 2006 titled “A Fast Learning Algorithm for Deep Belief Nets” in which they describe an approach to training “deep” (as in a many layered network) of restricted Boltzmann machines.

Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory.

This paper and the related paper Geoff co-authored titled “Deep Boltzmann Machines” on an undirected deep network were well received by the community (now cited many hundreds of times) because they were successful examples of greedy layer-wise training of networks, allowing many more layers in feedforward networks.

In a co-authored article in Science titled “Reducing the Dimensionality of Data with Neural Networks” they stuck with the same description of “deep” to describe their approach to developing networks with many more layers than was previously typical.

We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

In the same article, they make an interesting comment that meshes with Andrew Ng’s comment about the recent increase in compute power and access to large datasets that has unleashed the untapped capability of neural networks when used at larger scale.

It has been obvious since the 1980s that backpropagation through deep autoencoders would be very effective for nonlinear dimensionality reduction, provided that computers were fast enough, data sets were big enough, and the initial weights were close enough to a good solution. All three conditions are now satisfied.

In a talk to the Royal Society in 2016 titled “Deep Learning“, Geoff commented that Deep Belief Networks were the start of deep learning in 2006 and that the first successful application of this new wave of deep learning was to speech recognition in 2009 titled “Acoustic Modeling using Deep Belief Networks“, achieving state of the art results.

It was the results that made the speech recognition and the neural network communities take notice, the use “deep” as a differentiator on previous neural network techniques that probably resulted in the name change.

The descriptions of deep learning in the Royal Society talk are very backpropagation centric as you would expect. Interesting, he gives 4 reasons why backpropagation (read “deep learning”) did not take off last time around in the 1990s. The first two points match comments by Andrew Ng above about datasets being too small and computers being too slow.

Deep learning excels on problem domains where the inputs (and even output) are analog. Meaning, they are not a few quantities in a tabular format but instead are images of pixel data, documents of text data or files of audio data.

Yann LeCun is the director of Facebook Research and is the father of the network architecture that excels at object recognition in image data called the Convolutional Neural Network (CNN). This technique is seeing great success because like multilayer perceptron feedforward neural networks, the technique scales with data and model size and can be trained with backpropagation.

This biases his definition of deep learning as the development of very large CNNs, which have had great success on object recognition in photographs.

In a 2016 talk at Lawrence Livermore National Laboratory titled “Accelerating Understanding: Deep Learning, Intelligent Applications, and GPUs” he described deep learning generally as learning hierarchical representations and defines it as a scalable approach to building object recognition systems:

deep learning [is] … a pipeline of modules all of which are trainable. … deep because [has] multiple stages in the process of recognizing an object and all of those stages are part of the training”

Jurgen Schmidhuber is the father of another popular algorithm that like MLPs and CNNs also scales with model size and dataset size and can be trained with backpropagation, but is instead tailored to learning sequence data, called the Long Short-Term Memory Network (LSTM), a type of recurrent neural network.

We do see some confusion in the phrasing of the field as “deep learning”. In his 2014 paper titled “Deep Learning in Neural Networks: An Overview” he does comment on the problematic naming of the field and the differentiation of deep from shallow learning. He also interestingly describes depth in terms of the complexity of the problem rather than the model used to solve the problem.

At which problem depth does Shallow Learning end, and Deep Learning begin? Discussions with DL experts have not yet yielded a conclusive response to this question. […], let me just define for the purposes of this overview: problems of depth > 10 require Very Deep Learning.

Demis Hassabis is the founder of DeepMind, later acquired by Google. DeepMind made the breakthrough of combining deep learning techniques with reinforcement learning to handle complex learning problems like game playing, famously demonstrated in playing Atari games and the game Go with Alpha Go.

In keeping with the naming, they called their new technique a Deep Q-Network, combining Deep Learning with Q-Learning. They also name the broader field of study “Deep Reinforcement Learning”.

In their 2015 nature paper titled “Human-level control through deep reinforcement learning” they comment on the important role of deep neural networks in their breakthrough and highlight the need for hierarchical abstraction.

To achieve this, we developed a novel agent, a deep Q-network (DQN), which is able to combine reinforcement learning with a class of artificial neural network known as deep neural networks. Notably, recent advances in deep neural networks, in which several layers of nodes are used to build up progressively more abstract representations of the data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data.

Finally, in what may be considered a defining paper in the field, Yann LeCun, Yoshua Bengio and Geoffrey Hinton published a paper in Nature titled simply “Deep Learning“. In it, they open with a clean definition of deep learning highlighting the multi-layered approach.

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

Later the multi-layered approach is described in terms of representation learning and abstraction.

Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. […] The key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure.

This is a nice and generic a description, and could easily describe most artificial neural network algorithms. It is also a good note to end on.

Summary

In this post you discovered that deep learning is just very big neural networks on a lot more data, requiring bigger computers.

Although early approaches published by Hinton and collaborators focus on greedy layerwise training and unsupervised methods like autoencoders, modern state-of-the-art deep learning is focused on training deep (many layered) neural network models using the backpropagation algorithm. The most popular techniques are:

Multilayer Perceptron Networks.
Convolutional Neural Networks.
Long Short-Term Memory Recurrent Neural Networks.
I hope this has cleared up what deep learning is and how leading definitions fit together under the one umbrella.

Multi-Layer Perceptron Neural Networks
We are going to cover a lot of ground very quickly in this post. Here is an idea of what is ahead:
Multi-Layer Perceptrons.
Neurons, Weights and Activations.
Networks of Neurons.
Training Networks.

We will start off with an overview of multi-layer perceptrons.

The field of artificial neural networks is often just called neural networks or multi-layer perceptrons after perhaps the most useful type of neural network. A perceptron is a single neuron model that was a precursor to larger neural networks.

It is a field that investigates how simple models of biological brains can be used to solve difficult computational tasks like the predictive modeling tasks we see in machine learning. The goal is not to create realistic models of the brain, but instead to develop robust algorithms and data structures that we can use to model difficult problems.

The power of neural networks come from their ability to learn the representation in your training data and how to best relate it to the output variable that you want to predict. In this sense neural networks learn a mapping. Mathematically, they are capable of learning any mapping function and have been proven to be a universal approximation algorithm.

The predictive capability of neural networks comes from the hierarchical or multi-layered structure of the networks. The data structure can pick out (learn to represent) features at different scales or resolutions and combine them into higher-order features. For example from lines, to collections of lines to shapes.

Neurons

The building block for neural networks are artificial neurons.

These are simple computational units that have weighted input signals and produce an output signal using an activation function.

Model of a Simple Neuron
Model of a Simple Neuron
Neuron Weights

You may be familiar with linear regression, in which case the weights on the inputs are very much like the coefficients used in a regression equation.

Like linear regression, each neuron also has a bias which can be thought of as an input that always has the value 1.0 and it too must be weighted.

For example, a neuron may have two inputs in which case it requires three weights. One for each input and one for the bias.

Weights are often initialized to small random values, such as values in the range 0 to 0.3, although more complex initialization schemes can be used.

Like linear regression, larger weights indicate increased complexity and fragility. It is desirable to keep weights in the network small and regularization techniques can be used.

Activation

The weighted inputs are summed and passed through an activation function, sometimes called a transfer function.

An activation function is a simple mapping of summed weighted input to the output of the neuron. It is called an activation function because it governs the threshold at which the neuron is activated and strength of the output signal.

Historically simple step activation functions were used where if the summed input was above a threshold, for example 0.5, then the neuron would output a value of 1.0, otherwise it would output a 0.0.

Traditionally non-linear activation functions are used. This allows the network to combine the inputs in more complex ways and in turn provide a richer capability in the functions they can model. Non-linear functions like the logistic also called the sigmoid function were used that output a value between 0 and 1 with an s-shaped distribution, and the hyperbolic tangent function also called tanh that outputs the same distribution over the range -1 to +1.

More recently the rectifier activation function has been shown to provide better results.

Networks of Neurons

Neurons are arranged into networks of neurons.

A row of neurons is called a layer and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology.

Model of a Simple Network
Model of a Simple Network
Input or Visible Layers

The bottom layer that takes input from your dataset is called the visible layer, because it is the exposed part of the network. Often a neural network is drawn with a visible layer with one neuron per input value or column in your dataset. These are not neurons as described above, but simply pass the input value though to the next layer.

Hidden Layers

Layers after the input layer are called hidden layers because that are not directly exposed to the input. The simplest network structure is to have a single neuron in the hidden layer that directly outputs the value.

Given increases in computing power and efficient libraries, very deep neural networks can be constructed. Deep learning can refer to having many hidden layers in your neural network. They are deep because they would have been unimaginably slow to train historically, but may take seconds or minutes to train using modern techniques and hardware.

Output Layer

The final hidden layer is called the output layer and it is responsible for outputting a value or vector of values that correspond to the format required for the problem.

The choice of activation function in he output layer is strongly constrained by the type of problem that you are modeling. For example:

A regression problem may have a single output neuron and the neuron may have no activation function.
A binary classification problem may have a single output neuron and use a sigmoid activation function to output a value between 0 and 1 to represent the probability of predicting a value for the class 1. This can be turned into a crisp class value by using a threshold of 0.5 and snap values less than the threshold to 0 otherwise to 1.
A multi-class classification problem may have multiple neurons in the output layer, one for each class (e.g. three neurons for the three classes in the famous iris flowers classification problem). In this case a softmax activation function may be used to output a probability of the network predicting each of the class values. Selecting the output with the highest probability can be used to produce a crisp class classification value.

Training Networks

Once configured, the neural network needs to be trained on your dataset.

Data Preparation

You must first prepare your data for training on a neural network.

Data must be numerical, for example real values. If you have categorical data, such as a sex attribute with the values “male” and “female”, you can convert it to a real-valued representation called a one hot encoding. This is where one new column is added for each class value (two columns in the case of sex of male and female) and a 0 or 1 is added for each row depending on the class value for that row.

This same one hot encoding can be used on the output variable in classification problems with more than one class. This would create a binary vector from a single column that would be easy to directly compare to the output of the neuron in the network’s output layer, that as described above, would output one value for each class.

Neural networks require the input to be scaled in a consistent way. You can rescale it to the range between 0 and 1 called normalization. Another popular technique is to standardize it so that the distribution of each column has the mean of zero and the standard deviation of 1.

Scaling also applies to image pixel data. Data such as words can be converted to integers, such as the popularity rank of the word in the dataset and other encoding techniques.

Stochastic Gradient Descent

The classical and still preferred training algorithm for neural networks is called stochastic gradient descent.

This is where one row of data is exposed to the network at a time as input. The network processes the input upward activating neurons as it goes to finally produce an output value. This is called a forward pass on the network. It is the type of pass that is also used after the network is trained in order to make predictions on new data.

The output of the network is compared to the expected output and an error is calculated. This error is then propagated back through the network, one layer at a time, and the weights are updated according to the amount that they contributed to the error. This clever bit of math is called the backpropagation algorithm.

The process is repeated for all of the examples in your training data. One of updating the network for the entire training dataset is called an epoch. A network may be trained for tens, hundreds or many thousands of epochs.

Weight Updates

The weights in the network can be updated from the errors calculated for each training example and this is called online learning. It can result in fast but also chaotic changes to the network.

Alternatively, the errors can be saved up across all of the training examples and the network can be updated at the end. This is called batch learning and is often more stable.

Typically, because datasets are so large and because of computational efficiencies, the size of the batch, the number of examples the network is shown before an update is often reduced to a small number, such as tens or hundreds of examples.

The amount that weights are updated is controlled by a configuration parameters called the learning rate. It is also called the step size and controls the step or change made to network weight for a given error. Often small weight sizes are used such as 0.1 or 0.01 or smaller.

The update equation can be complemented with additional configuration terms that you can set.

Momentum is a term that incorporates the properties from the previous weight update to allow the weights to continue to change in the same direction even when there is less error being calculated.
Learning Rate Decay is used to decrease the learning rate over epochs to allow the network to make large changes to the weights at the beginning and smaller fine tuning changes later in the training schedule.
Prediction

Once a neural network has been trained it can be used to make predictions.

You can make predictions on test or validation data in order to estimate the skill of the model on unseen data. You can also deploy it operationally and use it to make predictions continuously.

> The Case for Convolutional Neural Networks
>
> Given a dataset of gray scale images with the standardized size of 32×32 pixels each, a traditional feedforward neural network would require 1024 input weights (plus one bias).
>
> This is fair enough, but the flattening of the image matrix of pixels to a long vector of pixel values loses all of the spatial structure in the image. Unless all of the images are perfectly resized, the neural network will have great difficulty with the problem.

The network topology and the final set of weights is all that you need to save from the model. Predictions are made by providing the input to the network and performing a forward-pass allowing it to generate an output that you can use as a prediction.

Convolutional Neural Networks
Convolutional Neural Networks are a powerful artificial neural network technique.

These networks preserve the spatial structure of the problem and were developed for object recognition tasks such as handwritten digit recognition. They are popular because people are achieving state-of-the-art results on difficult computer vision and natural language processing tasks.

Building Blocks of Convolutional Neural Networks

There are three types of layers in a Convolutional Neural Network:

Convolutional Layers.
Pooling Layers.
Fully-Connected Layers.
1. Convolutional Layers

Convolutional layers are comprised of filters and feature maps.

Filters

The filters are the “neurons” of the layer. The have input weights and output a value. The input size is a fixed square called a patch or a receptive field.

If the convolutional layer is an input layer, then the input patch will be pixel values. If the deeper in the network architecture, then the convolutional layer will take input from a feature map from the previous layer.

Feature Maps

The feature map is the output of one filter applied to the previous layer.

A given filter is drawn across the entire previous layer, moved one pixel at a time. Each position results in an activation of the neuron and the output is collected in the feature map. You can see that if the receptive field is moved one pixel from activation to activation, then the field will overlap with the previous activation by (field width – 1) input values.

Zero Padding

The distance that filter is moved across the the input from the previous layer each activation is referred to as the stride.

If the size of the previous layer is not cleanly divisible by the size of the filters receptive field and the size of the stride then it is possible for the receptive field to attempt to read off the edge of the input feature map. In this case, techniques like zero padding can be used to invent mock inputs for the receptive field to read.

2. Pooling Layers

The pooling layers down-sample the previous layers feature map.

Pooling layers follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layers feature map. As such, pooling may be consider a technique to compress or generalize feature representations and generally reduce the overfitting of the training data by the model.

They too have a receptive field, often much smaller than the convolutional layer. Also, the stride or number of inputs that the receptive field is moved for each activation is often equal to the size of the receptive field to avoid any overlap.

Pooling layers are often very simple, taking the average or the maximum of the input value in order to create its own feature map.

3. Fully Connected Layers

Fully connected layers are the normal flat feed-forward neural network layer.

These layers may have a non-linear activation function or a softmax activation in order to output probabilities of class predictions.

Fully connected layers are used at the end of the network after feature extraction and consolidation has been performed by the convolutional and pooling layers. They are used to create final non-linear combinations of features and for making predictions by the network.

Worked Example of a Convolutional Neural Network

You now know about convolutional, pooling and fully connected layers. Let's make this more concrete by working through how these three layers may be connected together.

1. Image Input Data

Let's assume we have a dataset of grayscale images. Each image has the same size of 32 pixels wide and 32 pixels high, and pixel values are between 0 and 255, g.e. a matrix of 32x32x1 or 1024 pixel values.

Image input data is expressed as a 3-dimensional matrix of width * height * channels. If we were using color images in our example, we would have 3 channels for the red, green and blue pixel values, e.g. 32x32x3.

2. Convolutional Layer

We define a convolutional layer with 10 filters and a receptive field 5 pixels wide and 5 pixels high and a stride length of 1.

Because each filter can only get input from (i.e. "see") 5×5 (25) pixels at a time, we can calculate that each will require 25 + 1 input weights (plus 1 for the bias input).

Dragging the 5×5 receptive field across the input image data with a stride width of 1 will result in a feature map of 28×28 output values or 784 distinct activations per image.

We have 10 filters, so that is 10 different 28×28 feature maps or 7,840 outputs that will be created for one image.

Finally, we know we have 26 inputs per filter, 10 filters and 28×28 output values to calculate per filter, therefore we have a total of 26x10x28x28 or 203,840 “connections” in our convolutional layer, we we want to phrase it using traditional neural network nomenclature.

Convolutional layers also make use of a nonlinear transfer function as part of activation and the rectifier activation function is the popular default to use.

3. Pool Layer

We define a pooling layer with a receptive field with a width of 2 inputs and a height of 2 inputs. We also use a stride of 2 to ensure that there is no overlap.

This results in feature maps that are one half the size of the input feature maps. From 10 different 28×28 feature maps as input to 10 different 14×14 feature maps as output.

We will use a max() operation for each receptive field so that the activation is the maximum input value.

4. Fully Connected Layer

Finally, we can flatten out the square feature maps into a traditional flat fully connected layer.

We can define the fully connected layer with 200 hidden neurons, each with 10x14x14 input connections, or 1960 + 1 weights per neuron. That is a total of 392,200 connections and weights to learn in this layer.

We can use a sigmoid or softmax transfer function to output probabilities of class values directly.

Convolutional Neural Networks Best Practices

Now that we know about the building blocks for a convolutional neural network and how the layers hang together, we can review some best practices to consider when applying them.

Input Receptive Field Dimensions: The default is 2D for images, but could be 1D such as for words in a sentence or 3D for video that adds a time dimension.
Receptive Field Size: The patch should be as small as possible, but large enough to “see” features in the input data. It is common to use 3×3 on small images and 5×5 or 7×7 and more on larger image sizes.
Stride Width: Use the default stride of 1. It is easy to understand and you don’t need padding to handle the receptive field falling off the edge of your images. This could increased to 2 or larger for larger images.
Number of Filters: Filters are the feature detectors. Generally fewer filters are used at the input layer and increasingly more filters used at deeper layers.
Padding: Set to zero and called zero padding when reading non-input data. This is useful when you cannot or do not want to standardize input image sizes or when you want to use receptive field and stride sizes that do not neatly divide up the input image size.

Pooling: Pooling is a destructive or generalization process to reduce overfitting. Receptive field is almost always set to to 2×2 with a stride of 2 to discard 75% of the activations from the output of the previous layer.
Data Preparation: Consider standardizing input data, both the dimensions of the images and pixel values.
Pattern Architecture: It is common to pattern the layers in your network architecture. This might be one, two or some number of convolutional layers followed by a pooling layer. This structure can then be repeated one or more times. Finally, fully connected layers are often only used at the output end and may be stacked one, two or more deep.
Dropout: CNNs have a habit of overfitting, even with pooling layers. Dropout should be used such as between fully connected layers and perhaps after pooling layers.

Recurrent Neural Networks

There is another type of neural network that is dominating difficult machine learning problems that involve sequences of inputs called recurrent neural networks.

Recurrent neural networks have connections that have loops, adding feedback and memory to the networks over time. This memory allows this type of network to learn and generalize across sequences of inputs rather than individual patterns.

A powerful type of Recurrent Neural Network called the Long Short-Term Memory Network has been shown to be particularly effective when stacked into a deep configuration, achieving state-of-the-art results on a diverse array of problems from language translation to automatic captioning of images and videos.

Support For Sequences in Neural Networks

There are some problem types that are best framed involving either a sequence as an input or an output.

For example, consider a univariate time series problem, like the price of a stock over time. This dataset can be framed as a prediction problem for a classical feedforward multilayer Perceptron network by defining a windows size (e.g. 5) and training the network to learn to make short term predictions from the fixed sized window of inputs.

This would work, but is very limited. The window of inputs adds memory to the problem, but is limited to just a fixed number of points and must be chosen with sufficient knowledge of the problem. A naive window would not capture the broader trends over minutes, hours and days that might be relevant to making a prediction. From one prediction to the next, the network only knows about the specific inputs it is provided.

Univariate time series prediction is important, but there are even more interesting problems that involve sequences.

Consider the following taxonomy of sequence problems that require a mapping of an input to an output (taken from Andrej Karpathy).

One-to-Many: sequence output, for image captioning.
Many-to-One: sequence input, for sentiment classification.

Many-to-Many: sequence in and out, for machine translation.
Synched Many-to-Many: synced sequences in and out, for video classification.
We can also see that a one-to-one example of input to output would be an example of a classical feedforward neural network for a prediction task like image classification.

Support for sequences in neural networks is an important class of problem and one where deep learning has recently shown impressive results State-of-the art results have been using a type of network specifically designed for sequence problems called recurrent neural networks.

Recurrent Neural Networks

Recurrent Neural Networks or RNNs are a special type of neural network designed for sequence problems.

Given a standard feed-forward multilayer Perceptron network, a recurrent neural network can be thought of as the addition of loops to the architecture. For example, in a given layer, each neuron may pass its signal latterly (sideways) in addition to forward to the next layer. The output of the network may feedback as an input to the network with the next input vector. And so on.

The recurrent connections add state or memory to the network and allow it to learn broader abstractions from the input sequences.

The field of recurrent neural networks is well established with popular methods. For the techniques to be effective on real problems, two major issues needed to be resolved for the network to be useful.

How to train the network with Backpropagation.
How to stop gradients vanishing or exploding during training.
1. How to Train Recurrent Neural Networks

The staple technique for training feedforward neural networks is to back propagate error and update the network weights.

Backpropagation breaks down in a recurrent neural network, because of the recurrent or loop connections.

This was addressed with a modification of the Backpropagation technique called Backpropagation Through Time or BPTT.

Instead of performing backpropagation on the recurrent network as stated, the structure of the network is unrolled, where copies of the neurons that have recurrent connections are created. For example a single neuron with a connection to itself (A->A) could be represented as two neurons with the same weight values (A->B).

This allows the cyclic graph of a recurrent neural network to be turned into an acyclic graph like a classic feed-forward neural network, and Backpropagation can be applied.

2. How to Have Stable Gradients During Training

When Backpropagation is used in very deep neural networks and in unrolled recurrent neural networks, the gradients that are calculated in order to update the weights can become unstable.

They can become very large numbers called exploding gradients or very small numbers called the vanishing gradient problem. These large numbers in turn are used to update the weights in the network, making training unstable and the network unreliable.

This problem is alleviated in deep multilayer Perceptron networks through the use of the Rectifier transfer function, and even more exotic but now less popular approaches of using unsupervised pre-training of layers.

In recurrent neural network architectures, this problem has been alleviated using a new type of architecture called the Long Short-Term Memory Networks that allows deep recurrent networks to be trained.

Long Short-Term Memory Networks

The Long Short-Term Memory or LSTM network is a recurrent neural network that is trained using Backpropagation Through Time and overcomes the vanishing gradient problem.

As such it can be used to create large (stacked) recurrent networks, that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results.

Instead of neurons, LSTM networks have memory blocks that are connected into layers.

A block has components that make it smarter than a classical neuron and a memory for recent sequences. A block contains gates that manage the block's state and output. A unit operates upon an input sequence and each gate within a unit uses the sigmoid activation function to control whether they are triggered or not, making the change of state and addition of information flowing through the unit conditional.

There are three types of gates within a memory unit:

Forget Gate: conditionally decides what information to discard from the unit.
Input Gate: conditionally decides which values from the input to update the memory state.
Output Gate: conditionally decides what to output based on input and the memory of the unit.
Each unit is like a mini state machine where the gates of the units have weights that are learned during the training procedure.

You can see how you may achieve a sophisticated learning and memory from a layer of LSTMs, and it is not hard to imagine how higher-order abstractions may be layered with multiple such layers.