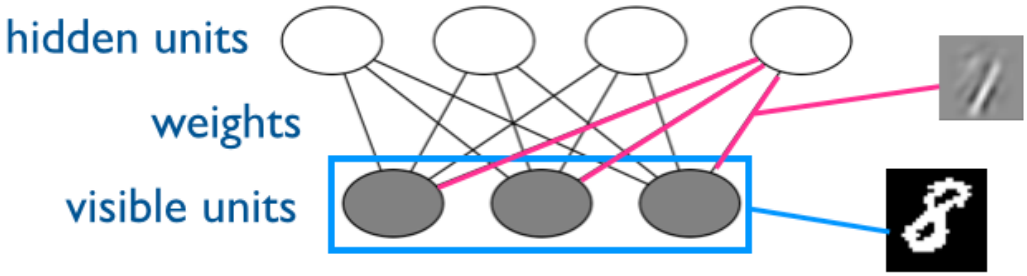
Deep learning from the bottom up

Created by: Roger Grosse
Intended for: machine learning practitioners

> This document was started by Roger Grosse, but as an experiment we have made it publicly editable. (You need to be logged in to edit.)

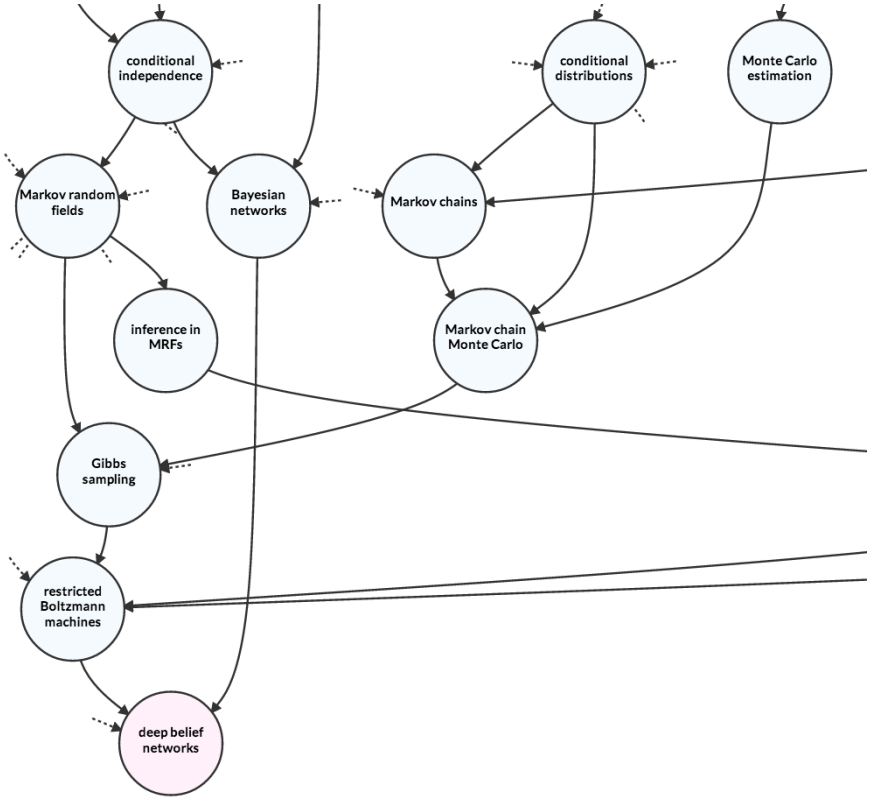


In applied machine learning, one of the most thankless and time consuming tasks is coming up with good features which capture relevant structure in the data. Deep learning is a new and exciting subfield of machine learning which attempts to sidestep the whole feature design process, instead learning complex predictors directly from the data. Most deep learning approaches are based on neural nets, where complex high-level representations are built through a cascade of units computing simple nonlinear functions.

Probably the most accessible introduction to neural nets and deep learning is Geoff Hinton's Coursera course. There you'll learn about the key ideas and be able to implement simple versions of the algorithms yourself. (Geoff is a pioneer in the field, and invented or influenced a large fraction of the work discussed here.)

But it's one thing to learn the basics, and another to be able to get them to work well. The field isn't at the point yet where you can just plug your data into the algorithm and have it work automatically. You'll need to be able to diagnose problems: is the model overfitting? Is the optimization procedure getting stuck? Should you add more units? More layers? Unfortunately, there aren't any recipes for these questions, and you'll need to do a lot of tinkering and experimentation yourself. For this, **you'll need to really understand the inner workings of the algorithms and how they relate to other key ideas in machine learning. This roadmap is meant to help you to achieve such a deeper understanding.**

If you are new to Metacademy, you can find a bit more about the structure and motivation here. Links to Metacademy concepts are shown in red; these will give you a full learning plan for the concept, assuming only high school calculus. For example:



These learning plans automatically get updated as new information is added to Metacademy. External links are shown in green; you're more or less on your own here, though we try to fill in background links where we can. There's no need to go through this roadmap linearly; you can follow whatever you need or find interesting. The learning plans will fill in the background.

You can also check out one of several review papers, which give readable overviews of recent progress in the field:

+ Y. Bengio. Learning deep architectures for AI. Foundations and Trends in Machine Learning, 2009.
+ Y. Bengio, A. Courville, and P. Vincent. Representation learning: a review and new perspectives. 2014

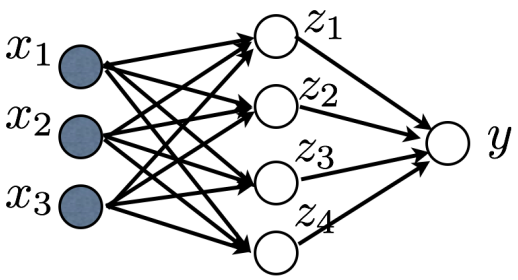
Supervised models

If you're interested in using neural nets, it's likely that you want to automatically predict something. Supervised learning is a machine learning framework where you have a particular task you'd like the computer to solve, and a training set where the correct predictions are labeled. For instance, you might want to automatically classify email messages as spam or not-spam, and in supervised learning, you have a dataset of 100,000 emails labeled as "spam" or "not spam" that you use to train your classifier so it can classify new emails it has never seen before.

Before diving into neural nets, you'll first want to be familiar with “shallow” machine learning algorithms, such as linear regression, logistic regression, and support vector machines (SVMs). These are far easier to implement, and there also exist pretty good software packages (e.g. scikit.learn). They serve as a sanity check for your neural net implementations: you should at least be able to beat these simple generic approaches. Plus, neural nets are built out of simple units which are closely related to these models. Therefore, by taking the time to learn about these, you automatically gain a deeper understanding of neural nets.

In order to have any hope of doing supervised learning, you need to understand the idea of generalization, the ability to make good predictions on novel examples. You'll need to understand how to balance the tradeoff between underfitting and overfitting: you want your model to be expressive enough to model relevant aspects of the data, but not so complex that it “overfits” by modeling all the idiosyncrasies. In the case of regression, this can be formalized in terms of bias and variance, which provides a useful intuition more generally. You should be able to measure generalization performance using cross-validation.

The vanilla deep learning model is the feed-forward neural net, which is trained with backpropagation.



Vision is one of the major application areas of deep learning, and convolutional nets have been applied there with tremendous success.

Recurrent neural nets are a kind of neural net model for data with temporal structure. Backpropagation through time is an elegant training algorithm, but it's a beast to get to work in practice.

Unsupervised models

In supervised learning, you have data labeled with the correct predictions for a particular task. But in many cases, labeled data is hard to obtain, or the correct behavior is hard to define. All you have is a lot of unlabeled data. This is the setting known as unsupervised learning. For instance, you may want to classify emails as "spam" or "not spam" but you don't have a dataset of labeled emails -- you only have the emails without spam/not-spam labels.

What can you do with unlabeled data? One thing you can do is simply look for patterns. Maybe your data is explainable in terms of a small number of underlying factors, or dimensions. This can be captured with principal component analysis or factor analysis. Or maybe you think the data are better explained in terms of clusters, where data points within a cluster are more similar than data points in different clusters. This can be captured with k-means or mixture of Gaussians.

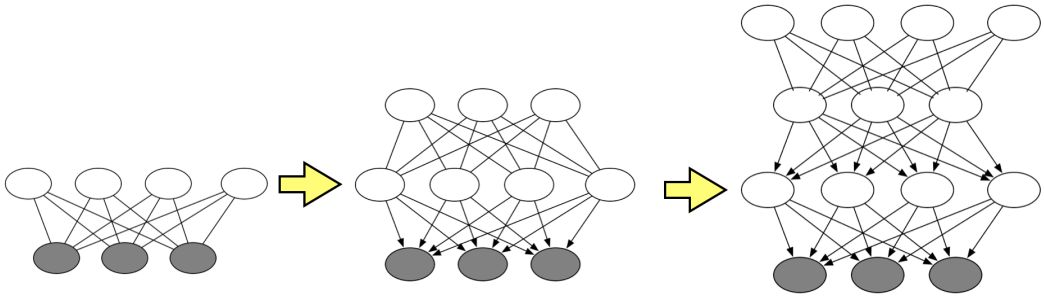
In the context of neural nets, there is another reason to care about unsupervised learning: it can often help you solve a supervised task better. In particular, unlabeled data is often much easier to obtain than labeled data. E.g., if you're working on object recognition, labeling the objects in images is a laborious task, whereas unlabeled data includes the billions of images available on the Internet.

Unsupervised pre-training has been shown to improve performance of supervised neural nets on a wide variety of tasks. The idea is that you start by training an unsupervised neural net on the unlabeled data (I'll cover examples shortly), and then convert it to a supervised network with a similar architecture. As a result of having to model the data distribution, the network will be primed to pick up relevant structure. Also, for reasons that are still not very well understood, **deep unsupervised models are often easier to train than deep supervised ones**. Initializing from an unsupervised network helps the optimizer avoid local optima.

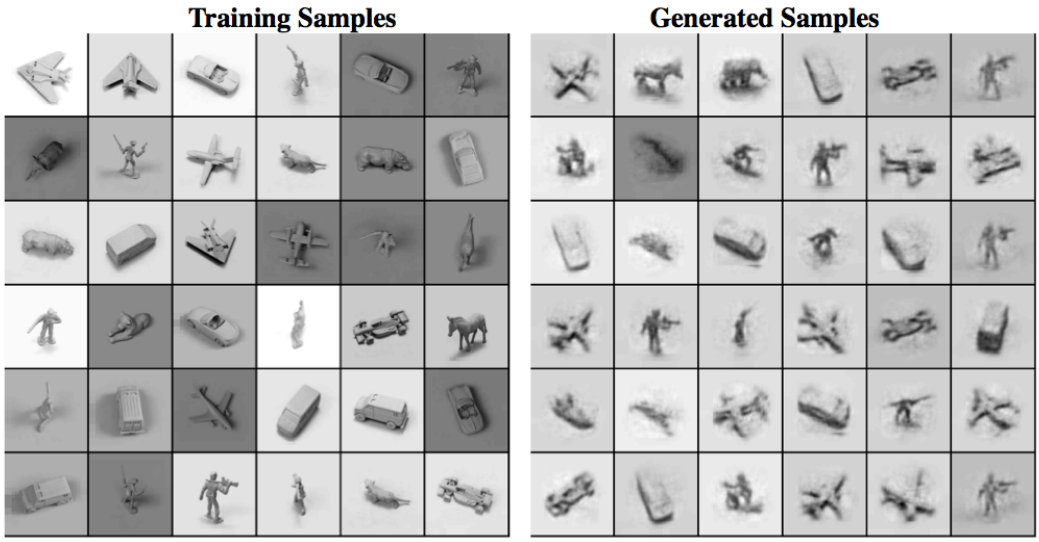
The evidence for generative pre-training is still mixed, and many of the most successful applications of deep neural nets have avoided it entirely, especially in the big data setting. But it has a good enough track record that it is worth being aware of.

So what are these unsupervised neural nets? The most basic one is probably the autoencoder, which is a feed-forward neural net which tries to predict its own input. While this isn't exactly the world's hardest prediction task, one makes it hard by somehow constraining the network. Often, this is done by introducing a bottleneck, where one or more of the hidden layers has much lower dimensionality than the inputs. Alternatively, one can constrain the hidden layer activations to be sparse (i.e. each unit activates only rarely), or feed the network corrupted versions of its inputs and make it reconstruct the clean ones (this is known as a denoising autoencoder).

Another approach to unsupervised learning is known as generative modeling. Here, one assumes the data are drawn from some underlying distribution, and attempts to model the distribution. Restricted Boltzmann machines (RBMs) are a simple generative neural network with a single hidden layer. They can be stacked to form multilayer generative models, including deep belief nets (DBNs) and deep Boltzmann machines (DBMs). There are a wide variety of variations on this basic idea, many of which are covered below.



DBMs can learn to model some pretty complex data distributions:



Generative modeling is a deep and rich area, and you can find lots more examples in the Bayesian machine learning roadmap.

Optimization algorithms

You've defined your neural net architecture. How the heck do you train it? The basic workhorse for neural net training is stochastic gradient descent (SGD), where one visits a single training example at a time (or a "minibatch" of training examples), and takes a small step to reduce the loss on those examples. This requires computing the gradient of the loss function, which can be done using backpropagation. Be sure to check your gradient computations with finite differences to make sure you've derived them correctly. SGD is conceptually simple and easy to implement, and with a bit of tuning, can work very well in practice.

There is a broad class of optimization problems known as convex optimization, where SGD and other local search algorithms are guaranteed to find the global optimum. This occurs because the function being optimized is "bowl shaped" (convex) and local improvements in the optimization function work towards the global optimum. Much of machine learning research is focused on trying to formulate things as convex optimization problems. Unfortunately, deep neural net training is usually not convex, so you are only guaranteed to

find a local optimum. This is a bit disappointing, but ultimately it's something we can live with. For most feed-forward networks and generative networks, the local optima tend to be pretty reasonable. (Recurrent neural nets are a different story — more on that below.)

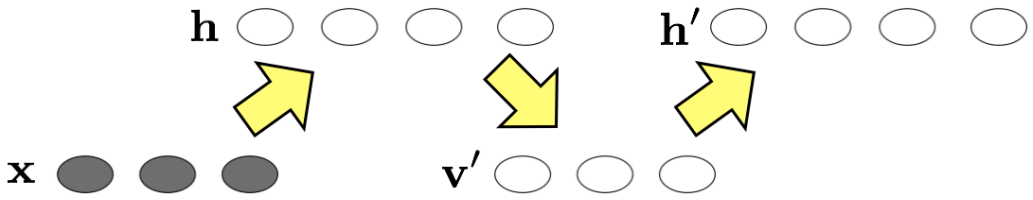
A bigger problem than local optima is that the curvature of the loss function can be pretty extreme. While neural net training isn't convex, the problem of curvature also shows up for convex problems, and many of the techniques for dealing with it are borrowed from convex optimization. As general background, it's useful to read the following sections of Boyd and Vandenberghe's book, Convex Optimization:

+ Sections 9.2-9.3 talk about gradient descent, the canonical first-order optimization method (i.e. a method which only uses first derivatives)
+ Section 9.5 talks about Newton's method, the canonical second-order optimization method (i.e. a method which accounts for second derivatives, or curvature)

While Newton's method is very good at dealing with curvature, it is impractical for large-scale neural net training for two reasons. First, it is a batch method, so it requires visiting every training example in order to make a single step. Second, it requires constructing and inverting the Hessian matrix, whose dimension is the number of parameters. (Matrix inversion is only practical up to tens of thousands of parameters, whereas neural nets typically have millions.) Still, it serves as an idealized second-order training method which one can try to approximate. Practical algorithms for doing so include:

+ conjugate gradient
+ limited memory BFGS

Compared with most neural net models, training RBMs introduces another complication: computing the objective function requires computing the partition function, and computing the gradient requires performing inference. Both of these problems are intractable. (This is true for learning Markov random fields (MRFs) more generally.) Contrastive divergence and persistent contrastive divergence are widely used approximations to the gradient which often work quite well in practice. Evaluating the models remains a difficult problem, though. One can estimate the model likelihood using annealed importance sampling, but this is delicate, and failures in estimation tend to overstate the model's performance.



Even once you understand the math behind these algorithms, the devil's in the details. Here are some good practical guides for getting these algorithms to work in practice:

+ G. Hinton. A practical guide to training restricted Boltzmann machines. 2010.

+ J. Martens and I. Sutskever. Training deep and recurrent networks with Hessian-free optimization. Neural Networks: Tricks of the Trade, 2012.
+ Y. Bengio. Practical recommendations for gradient-based training of deep architectures. Neural Networks: Tricks of the Trade, 2012.
+ L. Bottou. Stochastic gradient descent tricks. Neural Networks: Tricks of the Trade, 2012.

Other tricks

Dropout

Dropout is a way to add regularization to neural networks. The trick is that during training, certain number of neurons are randomly droped i.e. their contribution to the next set of neurons is removed in forward pass and their weights aren't updated in backward pass. Intuitively it makes the network less dependent on a particular neuron and thus it's called a regularization. The weights of the model are decreased before testing to approximate the effect of averaging all the 'different' networks trained by droping different neurons. [TODO: dropout]

[TODO: rectified linear units]

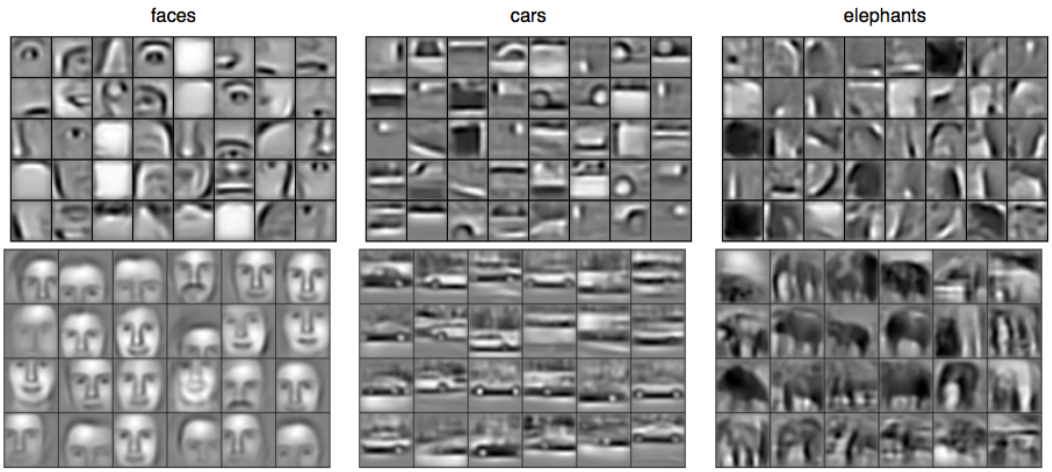
[TODO: GPU implementation]

Applications

Vision

Computer vision has been one of the major application areas of neural nets and deep learning. As early as 1998, convolutional nets were successfully applied to recognizing handwritten digits, and the MNIST handrwritten digit dataset has long been a major benchmark for neural net research. More recently, convolutional nets made a big splash by significantly pushing forward the state of the art in classifying between thousands of object categories. Vision was a large part of DeepMind's system which learned to play Atari games using only the raw pixels.

There's also been lots of work on generative models of images. Various work has focused on learning sparse representations and on modeling the local covariance structure of images. If you build a deep generative model with a convolutional architecture, you can high-level feature representations of objects:



Text

[TODO]

Speech

[TODO]

Software

+ Caffe is an increasingly popular deep learning software package designed for image-related tasks, e.g. object recognition. It's one of the fastest deep learning packages available -- it's written in C++ and CUDA.
+ The University of Toronto machine learning group has put together some nice GPU libraries for Python. GNumPy gives a NumPy-like wrapper for GPU arrays. It wraps around Cudamat, a GPU linear algebra library, and npmat, which pretends to be a GPU on a CPU machine (for debugging).
+ PyLearn is a neural net library developed by the University of Montreal machine learning group. It is intended for researchers, so it is built to be customizable and extendable.
+ PyLearn is built on top of Theano, a Python library for neural nets and related algorithms (also developed at Montreal), which provides symbolic differentiation and GPU support.
+ If for some reason you hate Python, Torch is a powerful machine learning library for Lua.

Relationships with other machine learning techniques

Neural nets share non-obvious relationships with a variety of algorithms from the rest of

machine learning. Understanding these relationships will help you decide when particular architectural decisions are appropriate.

Many neural net models can be seen as nonlinear generalizations of "shallow" models. Feed-forward neural nets are essentially nonlinear analogues of algorithms like logistic regression. Autoencoders can be seen as nonlinear analogues of dimensionality reduction algorithms like PCA.

RBMs with all Gaussian units are equivalent to Factor analysis. RBMs can also be generalized to other exponential family distributions.

Kernel methods are another set of techniques for converting linear algorithms into nonlinear ones. There is actually a surprising relationship between neural nets and kernels: Bayesian neural nets converge to Gaussian processes (a kernelized regression model) in the limit of infinitely many hidden units. (See Chapter 2 of Radford Neal's Ph.D. thesis. Background: Gaussian processes)

Relationship with the brain

If these models are called "neural" nets, it's natural to ask whether they have anything to do with how the brain works. In a certain sense, they don't: you can understand and apply the algorithms without knowing anything about neuroscience. Mathematically, feed-forward neural nets are just adaptive basis function expansions. But the connections do run pretty deep between practical machine learning and studies of the mind and brain.

Unfortunately, Metacademy doesn't have any neuroscience content (yet!), so the background links in this section will be fairly incomplete. Doubly unfortunately, neuroscience and cognitive science seem not to have the same commitment to open access that machine learning does, so this section might only be useful if you have access to a university library.

When trying to draw parallels between learning algorithms and the brain, we need to be precise about what level we're talking about. In "The philosophy and the approach" (Chapter 1 of Vision: a Computational Investigation), David Marr argued for explicitly separating different levels of analysis: computation, algorithms, and implementation. (This is worth reading, even if you read nothing else in this section.) While not all researchers agree with this way of partitioning things, it's useful to keep in mind when trying to understand exactly what someone is claiming.

Neuroscience

Jeff Hawkins's book On Intelligence aims to present a unifying picture of the

computational role of the neocortex. While the theory itself is fairly speculative, the book is an engaging and accessible introduction to the structure of the cortex.

Many neural net models have learned similar response properties to neurons in the primary visual cortex (V1).

+ Olshausen and Field's sparse coding model (background) was the first to demonstrate that a purely statistical learning algorithm discovered filters similar to those of V1. (Whether or not this is a neural net is a matter of opinion.) Since then, a wide variety of representation learning algorithms based on seemingly different ideas have recovered similar representations.
+ Other statistical models [TODO] have learned topological representations similar to the layout of cell types in V1.
+ Karklin and Lewicki fit a more sophisticated statistical model which reproduced response properties of complex cells.
+ While the connection between V1 and learned filters may seem tidy, Olshausen highlights a lot of things we still don't understand about V1.

For more on the neuroscience of the visual system, check out Eye, Brain, and Vision, a freely available book written by David Hubel, one of the pioneers who first studied V1. (Chapters 3, 4, and 5 are the most relevant.)

There have also been neural nets explicitly proposed as models of the brain. Riesenhuber and Poggio's HMAX model is a good example. Jim DiCarlo found that deep convolutional networks yield neurons which behave similarly to those high up in the primate visual hierarchy.

Cognitive science

It's not just at the level of neurons that researchers have tried to draw connections between the brain and neural nets. Cognitive science refers to the interdisciplinary study of thought processes, and can be thought of a study of the mind rather than the brain. Connectionism is a branch of cognitive science, especially influential during the 1980s, which attempted to model high-level cognitive processes in terms of networks of neuron-like units. (Several of the most influential machine learning researchers came out of this tradition.)

McClelland and Rumelhart's book Parallel Distributed Processing (volumes 1 and 2) is the connectionist Bible. Other significant works in the field include:

+ J. McClelland and T. Rogers. The parallel distributed processing approach to semantic cognition. Nature Reviews Neuroscience, 2003.
+ [TODO]

One of the most perplexing questions about the brain is how neural systems can model the compositional structure of language. Linguists tend to model language in terms of recursive structures like grammars, which are very different from the representations used in most neural net research. Paul Smolensky and Geraldine Legendre's book The Harmonic Mind presents a connectionist theory of language, where neurons implement a system of constraints between different linguistic features.