IPAM Summer School 2012 Tutorial on: Deep Learning

Geoffrey Hinton Canadian Institute for Advanced Research & Department of Computer Science University of Toronto

Overview of the tutorial

- A brief history of deep learning.
- How to learn multi-layer generative models of unlabelled data by learning one layer of features at a time.
 - What is really going on when we stack RBMs to form a deep belief net.
- How to use generative models to make discriminative training methods work much better for classification and regression.
- How to modify RBMs to deal with real-valued input.

PART 1

Introduction to Deep Learning & Deep Belief Nets

A brief history of deep learning

- The backpropagation algorithm for learning multiple layers of non-linear features was invented several times in the 1970's and 1980's (Werbos, Amari?, Parker, LeCun, Rumelhart et. al.)
- Backprop clearly had great promise, but by the 1990's people in machine learning had largely given up on it because:
 - It did not seem to be able to make good use of multiple hidden layers (except in "timedelay" and convolutional nets).
 - It did not work well in recurrent networks.

How to learn many layers of features (~1985)

Back-propagate error signal to get derivatives for learning



What is wrong with back-propagation?

- It requires labeled training data.
 Almost all data is unlabeled.
- The learning time does not scale well
 - It is very slow in networks with multiple hidden layers. Why?
- It can get stuck in poor local optima.
 - These are often quite good, but for deep nets they are far from optimal.

Some major issues in deep learning

- Deep vs Shallow
 - Are deep nets really needed?
 - Why?
- Supervised vs Unsupervised
 - Is almost all learning unsupervised?
 - Is unsupervised learning more efficient?
 - Was unsupervised learning just a quick fix because we had weak optimization methods and/or small datasets?

More major issues in deep learning

- Priors and Prejudice
 - How much prior knowledge should we try to wire into the weights or the activities of the networks?
 - Smugging knowledge into the networks without prejudicing the solution by manipulating the training set.
- How do we map a task onto a neural network?
 - Attention and recursion.
 - Intelligent decomposition vs brute force scanning.

Overcoming the limitations of back-propagation by using unsupervised learning

- Keep the efficiency and simplicity of using a gradient method for adjusting the weights, but use it for modeling the structure of the sensory input.
 - Adjust the weights to maximize the probability that a generative model would have produced the sensory input.
 - Learn p(image) not p(label | image)
 - If you want to do computer vision, first learn computer graphics
- What kind of generative model should we learn?

Belief Nets

- A belief net is a directed acyclic graph composed of stochastic variables.
- We get to observe some of the variables and we would like to solve two problems:
- The inference problem: Infer the states of the unobserved variables.
- The learning problem: Adjust the interactions between variables to make the network more likely to generate the observed data.



We will use nets composed of layers of stochastic binary variables with weighted connections. Later, we will generalize to other types of variable.

Stochastic binary units (Bernoulli variables)

- These have a state of 1 or 0.
- The probability of turning on is determined by the weighted input from other units (plus a bias)



$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j s_j w_{ji})}$$

Learning Deep Belief Nets

- It is easy to generate an unbiased example at the leaf nodes, so we can see what kinds of data the network believes in.
- It is hard to infer the posterior distribution over all possible configurations of hidden causes.
- It is hard to even get a sample from the posterior.
- So how can we learn deep belief nets that have millions of parameters?



The learning rule for sigmoid belief nets

 p_i

- Learning is easy if we can get an unbiased sample from the posterior distribution over hidden states given the observed data.
- For each unit, maximize the log probability that its binary state in the sample from the posterior would be generated by the sampled binary states of its parents.

$$s_{j} = p(s_{i} = 1) = \frac{1}{1 + \exp(-\sum_{j} s_{j} w_{ji})}$$
$$\Delta w_{ji} = \varepsilon s_{j} (s_{i} - p_{i})$$

learning rate

Explaining away (Judea Pearl)

- Even if two hidden causes are independent, they can become dependent when we observe an effect that they can both influence.
 - If we learn that there was an earthquake it reduces the probability that the house jumped because of a truck.



Why it is usually very hard to learn sigmoid belief nets one layer at a time

- To learn W, we need the posterior distribution in the first hidden layer.
- Problem 1: The posterior is typically complicated because of "explaining away".
- Problem 2: The posterior depends on the prior as well as the likelihood.
 - So to learn W, we need to know the weights in higher layers, even if we are only approximating the posterior. All the weights interact.
- Problem 3: We need to integrate over all possible configurations of the higher variables to get the prior for first hidden layer. Its hopeless!



Some methods of learning deep belief nets

- Monte Carlo methods can be used to sample from the posterior.
 - But its painfully slow for large, deep models.
- In the 1990's people developed variational methods for learning deep belief nets
 - These only get approximate samples from the posterior.
 - Nevetheless, the learning is still guaranteed to improve a variational bound on the log probability of generating the observed data.

A breakthrough that makes deep learning efficient

- To learn deep nets efficiently, we need to learn one layer of features at a time. This does not work well if we assume that the latent variables are independent in the prior :
 - The latent variables are not independent in the posterior so inference is hard for non-linear models.
 - The learning tries to find independent causes using one hidden layer which is not usually possible.
- We need a way of learning one layer at a time that takes into account the fact that we will be learning more hidden layers later.
 - We solve this problem by using an undirected model.

Two types of generative neural network

- If we connect binary stochastic neurons in a directed acyclic graph we get a Sigmoid Belief Net (Radford Neal 1992).
- If we connect binary stochastic neurons using symmetric connections we get a Boltzmann Machine (Hinton & Sejnowski, 1983).
 - If we restrict the connectivity in a special way, it is easy to learn a Boltzmann machine.

Restricted Boltzmann Machines (Smolensky ,1986, called them "harmoniums")

- We restrict the connectivity to make learning easier.
 - Only one layer of hidden units.
 - We will deal with more layers later
 - No connections between hidden units.
- In an RBM, the hidden units are conditionally independent given the visible states.
 - So we can quickly get an unbiased sample from the posterior distribution when given a data-vector.
 - This is a big advantage over directed belief nets

hidden



visible

The Energy of a joint configuration (ignoring terms to do with biases)

binary state of binary state of hidden unit j

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij}$$
Energy with configuration v on the visible units and h on the hidden units

$$-\frac{\partial E(v,h)}{\partial w_{ij}} = v_i h_j$$

Weights \rightarrow Energies \rightarrow Probabilities

- Each possible joint configuration of the visible and hidden units has an energy
 - The energy is determined by the weights and biases (as in a Hopfield net).
- The energy of a joint configuration of the visible and hidden units determines its probability:

$$p(v,h) \propto e^{-E(v,h)}$$

 The probability of a configuration over the visible units is found by summing the probabilities of all the joint configurations that contain it.

Using energies to define probabilities

- The probability of a joint configuration over both visible and hidden units depends on the energy of that joint configuration compared with the energy of all other joint configurations.
- The probability of a configuration of the visible units is the sum of the probabilities of all the joint configurations that contain it.





A picture of the maximum likelihood learning algorithm for an RBM



Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty$$

A quick way to learn an RBM



Start with a training vector on the visible units.

Update all the hidden units in parallel

Update the all the visible units in parallel to get a "reconstruction".

Update the hidden units again.

$$\Delta w_{ij} = \mathcal{E}\left(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1\right)$$

This is not following the gradient of the log likelihood. But it works well. It is approximately following the gradient of another objective function (Carreira-Perpinan & Hinton, 2005).

Three ways to combine probability density models (an underlying theme of the tutorial)

- Mixture: Take a weighted average of the distributions.
 - It can never be sharper than the individual distributions.
 It's a very weak way to combine models.
- **Product:** Multiply the distributions at each point and then renormalize (this is how an RBM combines the distributions defined by each hidden unit)
 - Exponentially more powerful than a mixture. The normalization makes maximum likelihood learning difficult, but approximations allow us to learn anyway.
- Composition: Use the values of the latent variables of one model as the data for the next model.
 - Works well for learning multiple layers of representation, but only if the individual models are undirected.

Training a deep network (the main reason RBM's are interesting)

- First train a layer of features that receive input directly from the pixels.
- Then treat the activations of the trained features as if they were pixels and learn features of features in a second hidden layer.
- It can be proved that each time we add another layer of features we improve a variational lower bound on the log probability of the training data.
 - The proof is slightly complicated.
 - But it is based on a neat equivalence between an RBM and a deep directed model (described later)

The generative model after learning 3 layers

- To generate data:
 - 1. Get an equilibrium sample from the top-level RBM by performing alternating Gibbs sampling for a long time.
 - 2. Perform a top-down pass to get states for all the other layers.

So the lower level bottom-up connections are not part of the generative model. They are just used for inference.



Why does greedy learning work? An aside: Averaging factorial distributions

- If you average some factorial distributions, you do NOT get a factorial distribution.
 - In an RBM, the posterior over the hidden units is factorial for each visible vector.
 - But the aggregated posterior over all training cases is not factorial (even if the data was generated by the RBM itself).

Why does greedy learning work?

The weights, W, in the bottom level RBM define p(v|h) and they also, indirectly, define p(h).

So we can express the RBM model as

$$p(v) = \sum_{h} p(h) p(v \mid h)$$

If we leave p(v|h) alone and improve p(h), we will improve p(v).

To improve p(h), we need it to be a better model of the aggregated posterior distribution over hidden vectors produced by applying W to the data. Another view of why layer-by-layer learning works (Hinton, Osindero & Teh 2006)

- There is an unexpected equivalence between RBM's and directed networks with many layers that all use the same weights.
 - This equivalence also gives insight into why contrastive divergence learning works.

An infinite sigmoid belief net that is equivalent to an RBM

- The distribution generated by this infinite directed net with replicated weights is the equilibrium distribution for a compatible pair of conditional distributions: p(v|h) and p(h|v) that are both defined by W
 - A top-down pass of the directed net is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium.
 - So this infinite directed net defines the same distribution as an RBM.



Inference in a directed net with replicated weights

- The variables in h0 are conditionally independent given v0.
 - Inference is trivial. We just multiply v0 by W transpose.
 - The model above h0 implements a complementary prior.
 - Multiplying v0 by W transpose gives the product of the likelihood term and the prior term.
- Inference in the directed net is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium starting at the data.



• The learning rule for a sigmoid belief net is:

$$\Delta W_{ij} \propto S_j (S_i - \hat{S}_i)$$

• With replicated weights this becomes:

$$s_{j}^{0}(s_{i}^{0} - s_{i}^{1}) + s_{i}^{1}(s_{j}^{0} - s_{j}^{1}) + s_{i}^{1}(s_{i}^{1} - s_{i}^{2}) + \dots + s_{j}^{\infty}s_{i}^{\infty}$$



Learning a deep directed network

- First learn with all the weights tied
 - This is exactly equivalent to learning an RBM
 - Contrastive divergence learning is equivalent to ignoring the small derivatives contributed by the tied weights between deeper layers.





- Then freeze the first layer of weights in both directions and learn the remaining weights (still tied together).
 - This is equivalent to learning another RBM, using the aggregated posterior distribution of h0 as the data.





How many layers should we use and how wide should they be?

- There is no simple answer.
 - Extensive experiments by Yoshua Bengio's group (described later) suggest that several hidden layers is better than one.
 - Results are fairly robust against changes in the size of a layer, but the top layer should be big.
- Deep belief nets give their creator a lot of freedom.
 - The best way to use that freedom depends on the task.
 - With enough narrow layers we can model any distribution over binary vectors (Sutskever & Hinton, 2007)

What happens when the weights in higher layers become different from the weights in the first layer?

- The higher layers no longer implement a complementary prior.
 - So performing inference using the frozen weights in the first layer is no longer correct. But its still pretty good.
 - Using this incorrect inference procedure gives a variational lower bound on the log probability of the data.
- The higher layers learn a prior that is closer to the aggregated posterior distribution of the first hidden layer.
 - This improves the network's model of the data.
 - Hinton, Osindero and Teh (2006) prove that this improvement is always bigger than the loss in the variational bound caused by using less accurate inference.

Fine-tuning with a contrastive version of the "wake-sleep" algorithm

After learning many layers of features, we can fine-tune the features to improve generation.

- 1. Do a stochastic bottom-up pass
 - Adjust the top-down weights to be good at reconstructing the feature activities in the layer below.
- Do a few iterations of sampling in the top level RBM
 Adjust the weights in the top-level RBM.
- 3. Do a stochastic top-down pass
 - Adjust the bottom-up weights to be good at reconstructing the feature activities in the layer above.

Removing structured noise by using top-down effects



- After an initial bottom up pass, gradually turn down the bottom up weights and turn up the top down weights.
- Look at what the first hidden layer would like to reconstruct.
- It removes noise!
- Does it improve recognition of noisy images?

Removing structured noise by using top-down effects

	3	3	3	3	N.	19. A.	$\frac{1}{2} \frac{1}{2} \frac{1}$	in the second se	1997 1997	0
	ar. Ann	$\xi_{m_{\rm R}^0}$	der i		24 - -	54	i.j	Lį	Ŋ	ų
noisy image	ħ	ŝ		3	3	til Til	4	ų	4	9
Ū	B	S	B	З	3	হা	9	4	9	9
	B	S	З	З	3	ণ্	4	4	9	9
	8	S	б	Э	3	IJ	4	4	4	9

$\hat{z}_{r}\xi$	0.4 1	ž.	i.	i h	Ч	*	ьţ	4	Ĺŝ
E	B	Ť	Z	3		4621. 34. 2.0			
Ĩ	B	S.	Ħ	F	Ŋ	ų.	S.	ŝ.	C,
g	8	8	З	3	9	Ą	9	63	Ц
8	8	8	8	З	4	4	9	63	Ц
8	8	8	8	3	Ч	ч	4	4	ч

PART 2:

Using backpropagation for fine-tuning a generative model to be better at discrimination

Fine-tuning for discrimination

- First learn one layer at a time greedily.
- Then treat this as "pre-training" that finds a good initial set of weights which can be finetuned by a local search procedure.
 - Contrastive wake-sleep is one way of finetuning the model to be better at generation.
- Backpropagation can be used to fine-tune the model for better discrimination.
 - This overcomes many of the limitations of standard backpropagation.

Why backpropagation works better with greedy pre-training: The optimization view

- Greedily learning one layer at a time scales well to really big networks, especially if we have locality in each layer.
- We do not start backpropagation until we already have sensible feature detectors that should already be very helpful for the discrimination task.
 - So the initial gradients are sensible and backprop only needs to perform a local search from a sensible starting point.

Why backpropagation works better with greedy pre-training: The overfitting view

- Most of the information in the final weights comes from modeling the distribution of input vectors.
 - The input vectors generally contain a lot more information than the labels.
 - The precious information in the labels is only used for the final fine-tuning.
 - The fine-tuning only modifies the features slightly to get the category boundaries right. It does not need to discover features.
- This type of backpropagation works well even if most of the training data is unlabeled.
 - The unlabeled data is still very useful for discovering good features.

First, model the distribution of digit images

The top two layers form a restricted Boltzmann machine whose free energy landscape should model the low dimensional manifolds of the digits.

The network learns a density model for unlabeled digit images. When we generate from the model we get things that look like real digits of all classes.

But do the hidden features really help with digit discrimination?

Add 10 softmaxed units to the top and do backpropagation.



Results on permutation-invariant MNIST task

- Very carefully trained backprop net with 1.6% one or two hidden layers (Platt; Hinton)
- SVM (Decoste & Schoelkopf, 2002) 1.4%
- Generative model of joint density of 1.25% images and labels (+ generative fine-tuning)
- Generative model of unlabelled digits 1.15% followed by gentle backpropagation (Hinton & Salakhutdinov, Science 2006)

Unsupervised "pre-training" also helps for models that have more data and better priors

- Ranzato et. al. (NIPS 2006) used an additional 600,000 distorted digits.
- They also used convolutional multilayer neural networks that have some built-in, local translational invariance.

Back-propagation alone: 0.49%

Unsupervised layer-by-layer pre-training followed by backprop: 0.39% (record)

Phone recognition with a DNN pre-trained as a deep belief net (Mohamed, Dahl & Hinton 2009)



- After the standard post-processing using a bi-phone model this gets 23.0% phone error rate.
- The best previous result on TIMIT was 24.4% and this required averaging several models.

We can do much better now using the spectrogram!

Learning Dynamics of Deep Nets

the next 4 slides describe work by Yoshua Bengio's group



Before fine-tuning

After fine-tuning

Effect of Unsupervised Pre-training

Erhan et. al. AISTATS' 2009



Effect of Depth

without pre-training

with pre-training



Learning Trajectories in Function Space (a 2-D visualization produced with t-SNE) Erhan et. al. AISTATS' 2009

- Each point is a model in function space
- Color = epoch
- Top: trajectories without pre-training. Each trajectory converges to a different local min.
- Bottom: Trajectories with pre-training.
- No overlap!



Why unsupervised pre-training makes sense



If image-label pairs were generated this way, it would make sense to try to go straight from images to labels. For example, do the pixels have even parity?



If image-label pairs are generated this way, it makes sense to first learn to recover the stuff that caused the image by inverting the high bandwidth pathway.

Modeling real-valued data

- For images of digits it is possible to represent intermediate intensities as if they were probabilities by using "mean-field" logistic units.
 - We can treat intermediate values as the probability that the pixel is inked.
- This will not work for real images.
 - In a real image, the intensity of a pixel is almost always, almost exactly the average of the neighboring pixels.
 - Mean-field logistic units cannot represent precise intermediate values.

Replacing binary variables by integer-valued variables (Teh and Hinton, 2001)

- One way to model an integer-valued variable is to make N identical copies of a binary unit.
- All copies have the same probability, of being "on" : p = logistic(x)
 - The total number of "on" copies is like the firing rate of a neuron.
 - It has a binomial distribution with mean N p and variance N p(1-p)

A better way to implement integer values

- Make many copies of a binary unit.
- All copies have the same weights and the same adaptive bias, b, but they have different fixed offsets to the bias:

$$b-0.5, b-1.5, b-2.5, b-3.5, \dots$$





- Contrastive divergence learning works well for the sum of binary units with offset biases.
- It also works for rectified linear units. These are much faster to compute than the sum of many logistic units with different biases.

output = max(0, x + noise(x)) many possible noise models

How to train a bipartite network of rectified linear units

• Just use contrastive divergence to lower the energy of data and raise the energy of nearby configurations that the model prefers to the data.



data

reconstruction

Start with a training vector on the visible units.

Update all hidden units in parallel with sampling noise

Update the visible units in parallel to get a "reconstruction".

Update the hidden units again

$$\Delta w_{ij} = \varepsilon \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon} \right)$$

3D Object Recognition: The NORB dataset Stereo-pairs of grayscale images of toy objects.



- 6 lighting conditions, 162 viewpoints
- -Five object instances per class in the training set
- A different set of five instances per class in the test set
- 24,300 training cases, 24,300 test cases

Simplifying the data

- Each training case is a stereo-pair of 96x96 images.
 - The object is centered.
 - The edges of the image are mainly blank.
 - The background is uniform and bright.
- To make learning faster I simplified the data:
 - Throw away one image.
 - Only use the middle 64x64 pixels of the other image.
 - Downsample to 32x32 by averaging 4 pixels.

Simplifying the data even more so that it can be modeled by rectified linear units

- The intensity histogram for each 32x32 image has a sharp peak for the bright background.
- Find this peak and call it zero.
- Call all intensities brighter than the background zero.
- Measure intensities downwards from the background intensity.



Test set error rates on NORB after greedy learning of one or two hidden layers using rectified linear units

Full NORB (2 images of 96x96)

- Logistic regression on the raw pixels 20.5%
- Gaussian SVM (trained by Leon Bottou) 11.6%
- Convolutional neural net (Le Cun's group) 6.0%
 (convolutional nets have knowledge of translations built in)

Reduced NORB (1 image 32x32)

- Logistic regression on the raw pixels
 30.2%
- Logistic regression on first relu hidden layer 14.9%
- Logistic regression on second relu hid layer 10.2%

See Nair & Hinton ICML 2010 for better results



The receptive fields of some rectified linear hidden units.

A standard type of real-valued visible unit

 We can model pixels as Gaussian variables. Alternating Gibbs sampling is still easy, though learning needs to be much slower.





Welling et. al. (2005) show how to extend RBM's to the exponential family. See also Bengio et. al. (2007)

Gaussian-Binary RBM's

- Lots of people have failed to get these to work and its extremely hard to learn tight variances for the visible units.
- It took a long time to figure out why it is so hard to learn the visible variances.



When sigma is much less than 1, the bottomup effects are too big and the top-down effects are too small.

The solution:

- Use as many hidden units as it takes to provide big enough top-down effects using small weights
- Relu's do this automatically.
 - If a relu has a bias of zero, it exhibits scale equivariance: $R(a \mathbf{x}) = a R(\mathbf{x})$
 - This is a very nice property to have for images.
 - It is like the equivariance to translation exhibited by convolutional nets.

 $R(shift(\mathbf{x})) = shift(R(\mathbf{x}))$

An early example of Gaussian-Relu RBMs

- Gaussian-Relu RBM's can be made to learn very nice convolutional filters on 32x32 color images (Alex Krizhevsky)
 - With an extra hidden layer of binary units and a few tricks, these gave record-breaking discrimination on CIFAR-10.



before fine-tuning



after fine-tuning

Why are rectified linear units so powerful?

- The non-linearity allows later layers to ignore irrelevant variations.
- The linear part of a hidden relu unit can model covariance between pixel intensities much more efficiently than the binary hidden units in a binary Gaussian RBM.
- An RBM with Gaussian visible units and Relu hidden units is a competitor to a mixture of factor analysers as a density model.
 - But MFA usually wins.