PART 1

Introduction to Deep Learning & Deep Belief Nets

A spectrum of machine learning tasks

Typical Statistics-----Artificial Intelligence

- Low-dimensional data (*e.g.* less than 1000 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.

 The main problem is distinguishing true structure from noise.

- High-dimensional data (*e.g.* more than 1000 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model (*e.g.* the mapping from images to captions).
- The main problem is figuring how to represent the complicated structure in a way that allows it to be learned. e.g.

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.

Two major issues in deep learning that I will not discuss

- Deep vs Shallow
 - Are deep nets really needed? (yes)
 - What can be proved? (not much)
- How do we map a task onto a neural network?
 - Attention and recursion.
 - Intelligent fixations 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?

Stochastic binary units (an odd choice)

- 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?



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!

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.

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.

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.

A restricted Boltzmann Machine

- 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.

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).

The return of backpropagation

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.
- Backpropagation can then be used to fine-tune the model for better discrimination.
 - This overcomes many of the limitations of standard backpropagation.

Acoustic modeling 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.

20

The best previous result on TIMIT was 24.4% and this required averaging several models.

We can do much better now using less pre-processing.

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.

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.

Is unsupervised pre-training really necessary?

- It is not necessary for the optimization to work.
- It helps a lot with the generalization if you do not have much labelled data compared with the number of parameters in your model.
- If you have enough computer power you should always be in the parameters >> labels regime.
 - Your brain has 10^14 synapses and you live for 10^9 seconds.

The ILSVRC-2012 competition on ImageNet

- The dataset has 1.2 million high-resolution training images.
- There are 1000 different classes of object.
- The task is to get the "correct" class in your top 5 bets.

- Some of the best existing computer vision methods were tried on this dataset by leading computer vision groups from Oxford, INRIA, XRCE, ...
 - Computer vision systems in 2012 used complicated multi-stage systems with lots of hand-engineering.
 - The early stages were typically tuned by optimizing a few parameters.

A neural network for ImageNet (terms in red will be explained later)

- Alex Krizhevsky et. al. (NIPS 2012) developed a very deep convolutional neural net (Le Cun 1987)
- Its architecture was:
 - 7 hidden layers not counting some max pooling layers.
 - The early layers were convolutional.
 - The last two layers were globally connected.

- The activation functions were rectified linear units in every hidden layer.
 - These train much faster and are more expressive than logistic units.
- The globally connected layers had most of the parameters.
 - Dropout was used to prevent these layers from overfitting

Examples from the test set (with the network's guesses)

• University of Toronto (Krizhevsky et. al.) • 16.4%

Error rates on the ILSVRC-2012 competition

- University of Tokyo
- Oxford University Computer Vision Group
- INRIA (French national research institute in CS) + XRCE (Xerox Research Center Europe)
- University of Amsterdam

- 26.1%
- •
- 26.9%
- •
- 27.0%
- 29.5%

Convolutional Neural Nets

(currently the dominant approach for object recognition)

- Use many different copies of the same feature detector with different positions.
 - Could also replicate across scale and orientation (but tricky and expensive)
 - Replication greatly reduces the number of free parameters to be learned.
- Use several different feature types, each with its own map of replicated detectors.
 - Allows each patch of the image to be represented in several ways.

The red connections all have the same weight.

Backpropagation with weight constraints

- It's easy to modify the backpropagation algorithm to incorporate linear constraints between the weights.
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
 - So if the weights started off satisfying the constraints, they will continue to satisfy them.

To constrain: $w_1 = w_2$ *we need*: $\Delta w_1 = \Delta w_2$

compute:
$$\frac{\partial E}{\partial w_1}$$
 and $\frac{\partial E}{\partial w_2}$
use $\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$ for w_1 and w_2

Pooling the outputs of replicated feature detectors

- Get a small amount of translational invariance at each level by averaging four neighboring replicated detectors to give a single output to the next level.
 - This reduces the number of inputs to the next layer of feature extraction, thus allowing us to have many more different feature maps.
 - Taking the maximum of the four works better.

Rectified linear units

 Instead of using the logistic sigmoid as the nonlinearity of a neuron, use rectification:

This non-linearity makes deep nets much easier to train and much better at dealing with real values.

Dropout: An efficient way to average many large neural nets.

- Consider a neural net with one hidden layer.
- Each time we present a training example, we randomly omit each hidden unit with probability 0.5.
- So we are randomly sampling from 2^AH different architectures.
 - All architectures share weights.

Dropout as a form of model averaging

- We sample from 2^AH models. So only a few of the models ever get trained, and they only get one training example.
- The sharing of the weights means that every model is very strongly regularized.
 - It's a much better regularizer than just trying to keep the weights small.

But what do we do at test time?

- We could sample many different architectures and take the geometric mean of their output distributions.
- It better to use all of the hidden units, but to halve their outgoing weights.
 - This exactly computes the geometric mean of the predictions of all 2⁺H models.

What if we have more hidden layers?

- Use dropout of 0.5 in every layer.
- At test time, use the "mean net" that has all the outgoing weights halved.
- This is not exactly the same as averaging all the separate dropped out models, but it's a pretty good approximation, and its fast.

What about the input layer?

- It helps to use dropout there too, but with a higher probability of keeping an input unit.
 - This trick is already used by the "denoising autoencoders" developed in Yoshua Bengio's group.
- One form of dropout in the input layer is to only look at a large randomly selected patch of the image.
 - This creates a lot more training examples!

What was actually wrong with backpropagation in 1986?

- We all drew the wrong conclusions about why it failed. The real reasons were:
- 1. Our labeled datasets were thousand of times too small.
- 2. Our computers were millions of times too slow.
- 3. We initialized the weights in a stupid way.
- 4. We used the wrong type of non-linearity.

A few years ago, Jeff Dean decided that with enough computation, neural networks might do amazing things. He built a lot of infrastructure to make it possible to train big nets on lots of data. It is beginning to look as if he was right.

Recurrent Neural Networks (with many of the details suppressed for clarity)

Recurrent Neural Networks

- RNNs are very powerful, because they combine two properties:
 - Distributed hidden state that allows them to store a lot of information about the past efficiently.
 - Non-linear dynamics that allows them to update their hidden state in complicated ways.
 - Deep ones work even better.

Back-propagation through time

- The connections in a recurrent net form a directed acyclic graph.
- Back-propagation through the DAG can be used to train the weights.
 - Targets can be provided for "output" neurons at any time-step.
 - The weights are shared over time so they add up the derivatives they get over all time-steps.

A radically new way to do machine translation (Suskever, Vinyals and Le, 2014)

- For each language we have a deep encoder RNN and a deep decoder RNN.
- The encoder RNN for the source language reads in the sequence of words in the source sentence.
 - Its final hidden state represents the thought that the sentence expresses.

The deep decoder RNN

- The decoder RNN for the target language starts with the thought produced by the encoder RNN.
- It defines a distribution over sentences in the target language.

How the decoder RNN specifies a distribution over translations

- First it outputs a probability distribution over possible first words.
- We pick a word from this distribution and feed it back into the RNN as an input.
- Given this first word, it then specifies a distribution over second words.
- Continue until you pick a full stop.
- During training, we only need to input the "correct" words.

How the encoder and decoder networks are trained

- Given a sentence pair, use back-propagation through time to maximize the log probability of producing the specified translation.
- Currently this system has only been trained for one pair of languages.
 - It already beats the state-of-the-art on that data.
 - It took less than one person year to develop.
- It will do much better when we use more data and jointly train encoders and decoders for many languages simultaneously.
 - The European parliament gives 25-way stereo for the thought. We can backprop through all 25 decoders.

Combining vision and language (a simplified account of recent work by Vinyals *et. al.*)

- The activity vector in the last hidden layer of a deep convolutional net trained on ImageNet is a "percept" that encodes what is in the image.
- Map this percept to the initial hidden state of a deep recurrent neural net.
- Train the RNN to say what it sees in the image.
 - Use an additional set of 200,000 images that each come with several captions (MS-COCO).
 - Do not retrain the convnet

Two pizzas sitting on top of a stove top oven.

(GT: Three different types of pizza on top of a stove.)

A group of people shopping at an outdoor market.

(GT: People are crouched around in an open market.)

A close up of a child holding a stuffed animal

(GT: A young girl asleep on the sofa cuddling a stuffed bear.)

THE END