1 Introduction

2 0.0 Background on de-noising autoencoders

Autoencoders, or autoassociators as they are sometimes called, are a variant of the simple three-layer artificial neural network where the output is expected to equal the input and the hidden layer is smaller or sparser than than the input layer. An autoencoder takes an input $\mathbf{x} \in [0,1]^d$ and first maps it (with an encoder) to a hidden representation $\mathbf{y} \in [0,1]^{d'}$ through a deterministic mapping, e.g.:

$$\mathbf{y} = s(\mathbf{Wx} + \mathbf{b})$$

Where s is a non-linearity such as the sigmoid. The latent representation $\mathbf{y}$, or code is then mapped back (with a decoder) into a reconstruction $\mathbf{z}$ of same shape as $\mathbf{x}$ through a similar transformation, e.g.:

$$\mathbf{z} = s(\mathbf{W'y} + \mathbf{b'})$$

$\mathbf{z}$ should be seen as a prediction, or *reconstruction*, of $\mathbf{x}$ given the code $\mathbf{y}$. The parameters of this model ($\mathbf{W}$, $\mathbf{W'}$, $\mathbf{b}$, and $\mathbf{b'}$) are optimized such that the average reconstruction error is minimized[2].

Show figure to explain autoencoder.

The aim of the autoencoder to learn the code $\mathbf{y}$ a distributed representation that captures the coordinates along the main factors of variation in the data (similarly to how principal component analysis (PCA) captures the main factors of variation in the data). Because $\mathbf{y}$ is viewed as a lossy compression of $\mathbf{x}$, it cannot be a good compression (with small loss) for all $\mathbf{x}$, so learning drives it to be one that is a good compression in particular for training examples, and hopefully for others as well, but not for arbitrary inputs. That is the sense in which an auto-encoder generalizes: it gives low reconstruction error to test examples from the same distribution as the training examples, but generally high reconstruction error to uniformly chosen configurations of the input vector.

If there is one linear hidden layer (the code) and the mean squared error criterion is used to train the network, then the hidden units learn to project the input in the span of the first principal components of the data. If the hidden layer is non-linear, the auto-encoder behaves differently from PCA, with the ability to capture multi-modal aspects of the input distribution. The departure from PCA becomes even more important when we consider stacking multiple encoders (and their corresponding decoders) when building a deep auto-encoder [Hinton06].

Explain the necessity of sparsity using the notion of data compression. Cite the evidence that the brain uses sparse representations.

The auto-encoder alone is not sufficient to be the basis of a deep architecture because it has a tendency towards over-fitting. The denoising autoencoder (dA) is an extension of a classical autoencoder introduced specifically as a building block for deep networks[10]. It attempts to re-construct a corrupted version of the input. The error in $\mathbf{z}$ is still compared against the un-corrupted input. The stochastic corruption process consists in randomly setting some of the inputs (as many as half of them) to zero. Hence the denoising auto-encoder is trying to predict the corrupted (i.e. missing) values from the uncorrupted (i.e., non-missing) values, for randomly selected subsets of missing patterns. This modification allows the dA to generalize well and produces compounding benefits when

the dA's are stacked into a deep network[4]. Hinton(google tech talk 3) suggests that the stochastic timing of the action potentials observed in neurons is a similar feature evolved to alleviate this problem of over-fitting.

3   0.1 Stacked de-noising autoencoders

Stacked denoising autoencoders, abbreviated SdA to differentiate them from other learning algorithms with that acronym, are not just neural networks with additional hidden layers, but a structure with individual levels of simple three-layer denoising autoencoders. First, a single denoising autoencoder is trained on the data. It's hidden layer converges on a sparse distributed representation of the training set. This essentially replaces the step where a researcher would have to design a collection of good features. Then, a second denoising autoencoder is trained to reconstruct corrupted versions of the activation of the hidden layer of the first dA for the collection of training examples. (the first level does not learn during this time). After a sufficient number of levels have been added, the encoders and decoders from each level are assembled into one long network and fine-tuned using back-propagation.
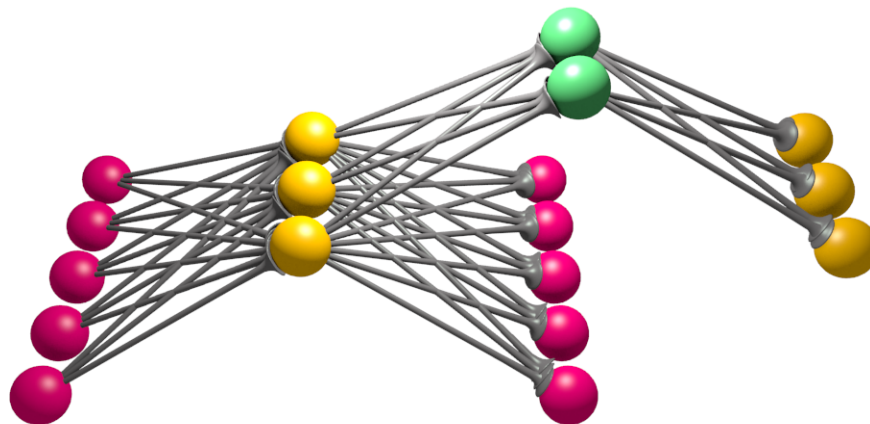

Figure 1: An SdA with two autoencoders. Each autoencoder is a a simple three layer network. The input/output to the first layer is the pink level, and the yellow level is it's distributed code, or hidden layer. The yellow hidden layer is then used as input/output to another autoencoder, it's hidden layer is shown in green.

Explain the necessity of a deep network and the curse of dimensionality.

Cite the many new developments in the field of deep nets.

Stacked denoising autoencoders are more complex neural networks, who's basic component are a variant of simple, three-layer networks. The are still simple compared to the neocortex, but have some of the same performance characteristics. The neocortex has several layers, but those layers do not correspond to the layers of the stack in a stacked denoising neural autoencoder. An SdA

only models the lateral connections that pyramidal cells make between areas of the cortex. Each pyramidal cell is a sort of representative for around 10,000 other neuron's in it's vicinity. For example, the areas V1 through V4 in the visual cortex are organized in a functional hierarchy, but they are not in a physical stack of levels. Pyramidal cells make lateral connections between the levels, which are arranged laterally, within the sheet of the neocortex.

Note that I am using the word 'layer' to refer to the three layers in a single dA, and the word 'level' to refer to the dA's within an SdA.

4   1.0 Successes of SdAs on machine perception tasks

Tell the story of Hinton's grad student who kept adding another layer, and cite the proof of the increase in the upper bound on prediction accuracy.

Cite Hinton's work on images, video, sound, and mocap.

The largest experiment so far with an SdA, by Google, was able to learn feature detectors for the human face, and a cat, among other things, by training on 10 millions images taken from youtube videos[7]. They used 9 levels of autoencoders. Some other tweaks were used as well, such as local contrast normalization (which basically saves them a level) and Max pooling, a standard procedure in machine vision. They trained the network for three days on a cluster with computing power in the neighborhood of 1 PFLOP. This image broke out on reddit with the title "first machine forms concept on it's own". While that's an oversimplification, and it's not the first, it is essentially what Google set out to do.

This appears to be a white white male human face, and the fact that it was selected by the researches probably reflects their own bias, and does not preclude that the SdA also arrived at codes for black, female, child, elderly, or non-human faces. Although that may not be as efficient as a code which can represent all faces as the sum of a pale white genderless face and some hair and color and eyes. 9 levels up in the hierarchy the categories are very abstract, so this is at least plausible.
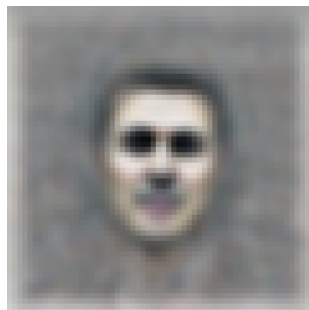

Figure 2: A representation of the face detection feature learned by Google's SdA. This image is generated by computing the input that would maximally activate the hidden node in the top layer that has come to represent faces. The SdA was trained on 10 million 200x200 images from youtube[7].

Training neural networks is an easy process to paralellize and distribute across many computers. However, all the extra training that occurs in an SdA requires even more computer power than usual. The de-noising step is essentially just a supersampling step to create more training examples, and adds an order of magnitude or two to the training time. For this reason, and because of falling costs, the latest developments in SdAs have been run on GPUs. Nvidia's CUDA shader language combined with a top of the line GPU (or several) affords 10 to 1000 fold increases in speed for applications which are very paralellizable. Using Python and Theano[6], a binding to the GPU accelerated linear algebra library LAPACK on Ubuntu, and a single Nvidia GTX 570 capable of 1405 GFLOPS, I was able to achieve about 20x increase in speed on a simple SdA pre-training operation on the MINST dataset, over a quad-core Intel i5-2500k CPU at 4.5 Ghz, measured to be capable of only 59 GFLOPS.

5 1.1 Applications of SdAs to planning tasks

Find some examples of this

The essence of planning is the attainment of a goal through a series of intermediate steps which do not necessarily lead directly toward that goal[?].

6 2.0 Plausibility of SdA's as a model of the mammalian neocortex

The primate visual cortex is one of the best understood brain regions in the animal kingdom. We know that it is composed of several levels, each more abstract than the one before it. The cortical columns in V1 are sensitive to a small receptive field of light coming in from the lateral geniculate cortex, which in turn receives information from the optic nerve. They are each slightly sensitive to certain angles and colors more than others. V2, which has connections drawing from V1 and other areas, has cells that are sensitive to mildly complex shapes somewhat invariant to translation, rotation, and scale, and will detect these shapes within a slightly larger receptive field. This gradual increase in abstraction continues as you move up the chain of levels, until you have neurons that are sensitive to something invariant like "person" or "thing moving towards me at an alarming speed"

The SdA method is evolving with functionality and performance in mind, and not explicitly to mimic the cortex, but we find that it has many of the same strengths and weaknesses. One such strength is it's plasticity. An SdA pre-trained to see video, performed better at a hearing task after fine-tuning, than a randomly initialized network fine-tuned in the same way. This is presumably because sound and video share some macro-statistical properties, such as having a high kurtosis.

An SdA also goes through different phases of learning that resemble the phases in a person's maturation. We know that plasticity in the brain decreases over time, and at about 30, the brain changes over to a new mode where it begins to fine tune existing connections and grows almost no new connections in the cortex.

7 2.1 Philosophical points

The levels in the SdA are levels of abstraction. What are the implications of a possible root at the top of this tree?

Perception is like nested stack of expectations, specific expectations of the near future based on a deluge of sensory data, and a smaller collection of abstract expectations of the medium range future based on a shower of specific, short-range expectations and some more sensory data. Recurse until lost.

Does this tree have a root? is there some "most abstract representation" of our sensory experience that we can expect in the indefinitely long run, that decodes to the specifics that we will in fact observe from this point on? I think that this very theory is a candidate. But that's not as important as it sounds. What any abstract expectation actually predicts depends on the more specific models below it, and the sensory data yet to be observed. "This too shall pass" would make a perfectly good root for our tree, if giving it that role didn't actually turn it into a paradox. A root here is really not more important than a leaf.

Let me re-iterate. I am operating on the assumption for this project, that perception and planning are both part of the same process, so the most abstract representations are joint models of sensory experience and action. The model of sensory data is influenced by three main factors. The accuracy, the sparsity, and the desirability, or pleasurableness to the creature. At all levels, the representations which jointly satisfy these three metrics come to dominate. In the case of a game this last factor would be the points or wins/losses experienced by the player. The player perceives the state of the game and the state of it's own recent actions, and expects a likely, sparse, and desirable scenario next, and then is made to take any actions it expected of itself which are legal in the rules of the game. It may not be the most brilliant way to play a game, but I think It would at least perform OK at Puerto Rico, a game about economic growth.

Jeff Hawkins said in 2004, Neural networks were a genuine improvement of the AI approach because their architecture is based, though very loosely, on real nervous systems...But as the neural network phenomenon exploded on the scene, it mostly settled on the class of ultra-simple models that didn't meet any of [his criteria for biological plausibility]. Most neural networks consisted of a small number of neurons connected in three rows...I thought the field would quickly move on to more realistic networks, but it didn't. Because these simple neural networks were able to do interesting things, research seemed to stop there for years. [3]

8 3.0 Expected performance of SdA vs. dA on the competitive growth game, Puerto Rico

I have no dataset of good gamestate and moves available to train from, so that precludes the possibility of a controlled experiment where the SdA and the dA are both trained on the same data. But I can pit the algorithms against eachother, and count how many wins each strategy gets.

I can also pit the algorithms against a simple greedy approach which always chooses the option that leads to the most points on the next turn, obviously, I'd expect them to beat it.

9 3.1 Implementation of Puerto Rico

The rules of the board game, Puerto Rico, with a minor variation proposed by it's author Andreas Seyfarth to balance the game, were encoded in Python 2.7. The implementation was strictly functional, though Python is not a strictly functional language. It was chosen for it's rapid prototyping and bug-repellant qualities. There is a data structure to represent the game state, and a data structure to represent a move. There are functions to take a move, and a gamestate, and produce the next game state.

Why was Puerto Rico chosen? * It is a complex problem of competitive growth relevant to today's global economy (even though it's about 17th century imperial colonialism, the problems are still systemically relevant) * The greediest move is not always the best move. The greediest move often promotes an indecisive strategy that can be easily beaten by a player dedicated to a plan. Choosing a way of life, and sticking to it is a good strategy, thus a compound plan is needed, with strategic choices at multiple levels of abstraction. * Additionally, the most sustainable strategy is not the best either, because the game does eventually end, and there is an implicit transition in the middle of the game when players switch to unsustainable exploitation of existing infrastructure. * Puerto rico has some randomness in it's initial configuration, but the state transition function is deterministic. * The end conditions can be straightforwardly extended for longer games. * the setup can be straightforwardly generalized to arbitrarily many players. Both of these extensions may be needed to make the game more difficult if the SDA masters it too easily, And to create a game for which the brute force approach is infeasible.

10 3.2 Implementation of SdA-based decider

11 4.0 Performance of SdA-based player against dA player and various other opponents

performance matrix? SdA vs. dA SdA vs. SdA SdA vs. me SdA vs. online players who think they are playing a human. (definitely don't have time to do this)

12 4.1 Conclusion and lessons learned

References

[1] S. C. Baker, R. D. Rogers, A. M. Owen, C. D. Frith, R. J. Dolan, R. S. J. Frackowiak, T. W. Robbins. Neural systems engaged by planning: a PET study of the Tower of London task. Neuropsychologia,Vol. 34, No. 6, pp. 515-526, 1996

[2] Bengio, P. Lamblin, D. Popovici and H. Larochelle, Greedy Layer-Wise Training of Deep Networks, in Advances in Neural Information Processing Systems 19 (NIPS06), pages 153-160, MIT Press 2007.

[3] Bengio, Learning deep architectures for AI, Foundations and Trends in Machine Learning 1(2) pages 1-127.

[4] Jeff Hawkins and Sandra Blakeslee. On Intelligence. Times Books, 2004.

[5] G.E. Hinton and R.R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science, 28 July 2006, Vol. 313. no. 5786, pp. 504 - 507.

[6] G.E. Hinton, S. Osindero, and Y. Teh, A fast learning algorithm for deep belief nets, Neural Computation, vol 18, 2006

[7] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. Theano: A CPU and GPU Math Expression Compiler. Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX

[8] Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, Andrew Ng. Building high-level features using large scale unsupervised learning. International Conference in Machine Learning, 2012.

[9] MarcAurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient Learning of Sparse Representations with an Energy-Based Model. Advances in Neural Information Processing Systems, 2006.

[10] Seyfarth, Andreas, Puerto Rico Rulebook. Rio Grande Games, 2001

[11] Vincent, H. Larochelle Y. Bengio and P.A. Manzagol, Extracting and Composing Robust Features with Denoising Autoencoders, Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML08), pages 1096 - 1103, ACM, 2008.