CS81: Learning words with Deep Belief Networks

George Dahl
gdahl@cs.swarthmore.edu

Kit La Touche
kit@cs.swarthmore.edu

Abstract

In this project, we use a Deep Belief Network (Hinton et al., 2006) to learn words in a fixed-size vocabulary, given input in multiple modalities (image and audio data). The goal of this project is like that of Plunkett et al. (1992): to model vocabulary acquisition, and address the Symbol Grounding problem from a connectionist standpoint. Our model learns to classify both spoken and hand-written digits in three distinct learning tasks. First, we train our network only on the image data, second, we train only on the audio data, and finally, we train on a combined dataset of paired image and audio data. Unlike Plunkett et al. (1992), we use a generative model, which allows us to fix the class labels and generate input vectors that our model considers good representatives of that class. The model also achieves high accuracy on the classification tasks.

1 Introduction

Imagine, some day (far) in the future, that you want your pet robot to find your missing sock. You tell it to do so, and it heads off, looking for a sock. In order to even begin to solve this problem, the robot needs many sophisticated capabilities. Primarily, the robot needs to understand that the audio signal "sock" it receives is correlated with a range of images of "socks" that it might perceive through its visual sensors.

Of course, we are a long way from solving the Sock-Finding Problem; a solution would require more sophisticated image and audio processing, and some sort of syntactic processing, to name but a few of the necessary components. The experiments we performed in this project show one way a robot could learn to associate input from multiple sensory modes with a given label.

Our work builds on work by Plunkett et al. (1992), in which a neural network was trained to associate fixed labels with abstract images of black dots on a white field. They used a normal feed-forward neural net with an autoencoder topology and a peculiar training regimen intended to allow the trained network to map both inputs to labels and labels to inputs. Inputs and outputs to their network were concatenations of labels and image vectors.

The primary goal of Plunkett et al. (1992) was to model human vocabulary acquisition and then use the model to both comprehend and produce "language" (they modeled a highly simplified and restricted form of what we might think of as language). Their work is in part an attempt at a connectionist answer to the Symbol Grounding Problem. (Harnad, 1990)

The Symbol Grounding Problem is a long-standing issue in AI which can be boiled-down to the following: how can words (which are arbitrary symbols) gain meaning, rather than simply circular definition in terms of other symbols, or brittle denotation of specific sensory states? In particular, how can a system ascribe semantic value (meaning) to symbols in a way that is intrinsic to the system, and not merely our interpretation of it?

This problem, of course, requires a clear idea of what is meant by "meaning", or, really, what the nature of meaning is. The argument put forth in Harnad (1990) is essentially that connectionist systems make semantics intrinsic by definition: if the "meaning" of a symbol is the set of other symbols and, crucially, subsymbolic elements that get activated along with it, then a connectionist system that correlates input in various modes would partially address this problem.

So, the symbol grounding problem seems to be solvable through only one route: the association of

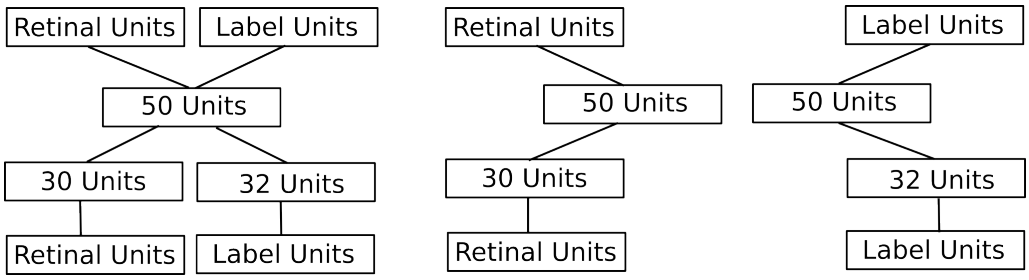

Figure 1: Architecture of the Plunkett et al. (1992) system.

prototypes in many modes with the same internal label. This is, of course, what our network sets out to do. This is also what the model in Plunkett et al. (1992) is designed to do.

There is an important sense in which we (and Plunkett) do not actually address the symbol grounding problem: we provide the network with training data that is split into different categories. A more complete system would have to provide its own system of categorization. Such a system might associate audio and visual input based on temporal cooccurance and use some sort of shared sensorimotor context to place audio/visual input pairs into different classes.

In addition to providing the seed of a connectionist answer to the Symbol Grounding Problem, Plunkett et al. (1992) were also interested in robots that might produce language. Indeed, it would be quite useful for a robot to be able to produce linguistically meaningful utterances in response to its environment and internal state. This task would of course require that the system associate input from different modes as being in the same category when appropriate, but it would also require a system capable of generating output like its input, not simply classifying inputs. This will lead us naturally to use a generative model.

We have extended the task in Plunkett et al. (1992) by using real-world audio and image data. Plunkett et al. (1992) liken their labels to elements of a vocabulary. We have made this comparison more plausible by replacing their labels with recordings of humans speaking words from a small, fixed, vocabulary. This means that a variety of different utterances of a word can be paired with a given image, rather than only a single fixed label. We also use real-world images of handwritten digits instead of contrived image prototypes. Although the vocabulary we are working with is smaller (10 items as opposed to 32), the task we have created is in most respects much harder. Our model must learn to correctly identify the class that an utterance/image pair belongs to even though there are many different utterances that are instances of that class, and each one can be paired with any different image instance that belongs to that class. As in Plunkett et al. (1992), we want our system to be able to generate its best guess of what an instance of a given class might be.

As well as generalizing the task, we have used a different connectionist model for learning. Instead of using feed-forward neural networks with nonstandard training regimens, we use Deep Belief Networks (Hinton et al., 2006), which are true generative models.

Plunkett et al. (1992) used a normal feedforward neural network, but had to structure and train it in an odd fashion to allow for both comprehension and production. They had three hidden layers, in two tiers. (See the leftmost network in Figure 1.) They first used backpropagation to train the network to autoassociate only images. Only weights on the path from the image input units to the image output units were updated (this path is the middle of Figure 1). Then they repeated this procedure for the label part of the input and output layers (this path corresponds to the rightmost part of Figure 1). Finally, they trained all weights in the network to autoassociate image/label pairs. This training procedure has a couple of theoretical problems. The most important one is that the weights for the 50 hidden units in the penultimate layer were being trained to optimize three different, and potentially contradictory, objective functions. The mixture between these objective functions was ill defined and dependent on the order of the training phases and how long each phase

was run. Furthermore, none of the phases of training actually optimized the weights to perform the tasks that the network was tested on, since updates were never performed "diagonally" through the network. These theoretical issues came about because traditional feed-forward networks are not generative models.

1.1 Generative vs. Discriminative models

There are two main types of probabilistic models: generative and discriminative models. The distinction between the two is based on what probability distribution they model. Generally one assumes that the goal of training is to predict some output variable y given the value of an input variable x. Discriminative models (such as traditional feed-forward neural networks trained in a way that allows their output to be interpreted as approximate posterior class probabilities[1]) directly model the probability of an output given an input. The alternative is a generative model, in which one models the joint probability distribution of the input and the output. Thus, while a discriminative model will estimate $P(y|x)$, a generative model will estimate $P(x, y)$ from which one can obtain either $P(y|x)$ or $P(x|y)$ using Bayes' theorem.

The various tradeoffs between generative and discriminative models are a fascinating area of research. However, given that we actually want to generate samples from $P(x|y)$ as well as perform classification, a generative model is most natural.

One particular tradeoff, however, bears mentioning: the asymptotic error of generative systems is typically greater than for discriminative systems. However, this error bound is reached more quickly than with a discriminative model. (Ng and Jordan, 2002) This tradeoff is one reason the final supervised fine-tuning phase in Deep Belief Network training is so helpful; after pre-training, the weights can be updated to minimize the appropriate loss function directly.

1.2 Deep Belief Networks

We use Deep Belief Networks (DBNs) (Hinton et al., 2006) for all our learning experiments. A DBN is composed of multiple layers of stochastic binary neurons. The top two layers form an associative memory, specifically a Restricted Boltzmann Machine, or RBM (all layers are at one point parts of Restricted Boltzmann Machines). DBNs are energy based models and thus every configuration of neuron activations has an energy associated with it; the energy function is determined by the weights. If neuron activations are updated with activation probabilities based on a logistic sigmoid applied to the neuron's net input, they will eventually reach an equilibrium distribution. The lower the energy of a configuration, the higher the probability of reaching it will be. The distinction between input and output layers is somewhat less sharp than in other neural network models; if activations are fixed on any of the visible units regardless of whether they are called input or output units the remaining visible units can get activations through repeated stochastic updates.

Figure 2 is a diagram of the DBN architecture that we used. The associative memory is formed by the 2000 top units, the 10 label units, and the topmost layer of 500 units. Thus, it is a 510 dimensional associative memory.

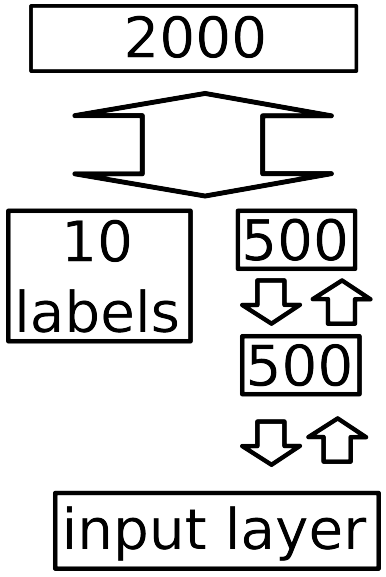

Figure 2: Architecture of the DBN.

1.2.1 Training Deep Belief Networks

Training of a Deep Belief Network is divided into two phases. The first phase is a greedy, unsupervised, layer-by-layer pre-training phase which is de-

[1]There is another distinction at work here: probabilistic versus non-probabilistic models that do not divide the classification problem into separate inference and decision stages.

signed to initialize the weights of the network to values in the neighborhood of a good local optimum of the error surface. This pre-training phase allows the DBN to make use of unlabeled data, which is often very desirable since most data is unlabeled. The second phase of training is a supervised, global fine-tuning phase that is very similar to traditional neural network training and can use normal gradient or conjugate gradient descent.

The pre-training phase considers a single layer in isolation and trains layers closest to the input layer first. Pre-training treats the current layer as the hidden units of a Restricted Boltzmann Machine and the previous layer as the visible units of the same RBM. While the first hidden layer is being trained, the actual training data can be used to obtain visible unit activations. In subsequent layers, the hidden activations of the previous layer are used as input data. Generally the fine-tuning phase takes the longest; the pre-training is quite fast.

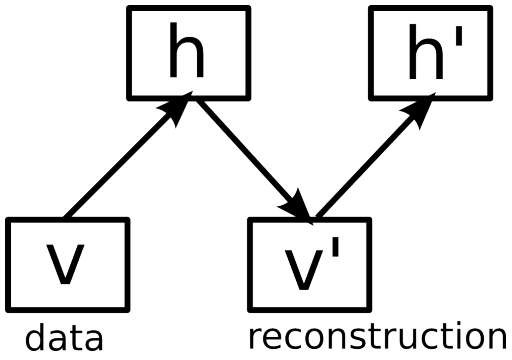

Figure 3: Training of an RBM.

At this point it is useful to go into a bit more detail on how pre-training a single layer works. Pre-training uses a local Hebbian update rule to maximize the log-probability of the data. The weights between the visible layer and the hidden layer are learned using contrastive divergence. To compute the updates to the weights, we first update the hidden activations (h_j) stochastically based on the visible units (v_i) and the weights. Then we stochastically update the visible unit activations based on the hidden unit activations and the weights to obtain a reconstruction of the training data. These reconstructed visible unit activations will be denoted v_i'. Finally we compute new hidden unit activations (h_j') based on the reconstruction of the training data. A diagram of this process is depicted in Figure 3. Given these quantities, the change in the weight between visible unit i and hidden unit j is:

$$\Delta w_{ij} = \epsilon[\langle v_i h_j \rangle - \langle v_i' h_j' \rangle],$$

where $\langle . \rangle$ denotes expectation with respect to the training data and ϵ is the learning rate. Technically, there is also a momentum term that repeats a fraction of the weight updates from the previous epoch.

1.2.2 Sampling from the Class-conditional Distributions of DBNs

Generating samples from the class-conditional distributions ($P(x|y)$) of a trained Deep Belief Network is conceptually simple even if it is relatively computationally expensive. The following steps generate a sample from $P(x|y)$ for a trained DBN:

1. Initialize the top level of the associative memory in an unbiased way. This is accomplished by propagating a random input vector up to the top of the network.

2. Alternate between stochastically updating the penultimate and ultimate layers until they converge to an equilibrium. In other words, let the associative memory settle on a low energy state. The second to last layer includes both the class-label units as well as the second to last hidden layer. Whenever the label units would be updated, instead (re-)set them to the values that represent the class being conditioned on.

3. Propagate activations back down to the input layer.

Of course this procedure will only make human-interpretable input vectors if the input representation is already human-interpretable. In order to sample from the marginal distribution $P(x)$ one simply does the above procedure without clamping values onto labels, and with the top level of the associative memory initialized randomly (instead of with an upward pass from a random input).

2 Experiments

We conducted a number of tests of the Deep Belief Network using the same hidden layer architecture

as Hinton et al. (2006), namely three hidden layers of 500, 500, and 2000 units. Figure 2 shows a diagram of the network architecture. We trained the network to classify images of handwritten digits, audio recordings of people speaking the words for different digits, and audio recordings paired with appropriate images. We did not perform extensive parameter optimization in any of our experiments and used a learning rate of 0.1 and a momentum factor of 0.9 during pre-training.

2.1 Data sets

Our data sets were, of necessity and by design, different from those of Plunkett et al. (1992). For our image data, we used the MNIST database of handwritten digits.[2] The MNIST database consists of 60,000 images of handwritten digits that have only had minimal preprocessing performed on them, along with a test set of 10,000 images. The images are all 28 by 28 pixels and roughly centered, but otherwise not preprocessed.

For audio data, we used the JEIDA/JCSD corpus of isolated Japanese digits.[3] Given that there are multiple ways to say some digits between zero and nine in Japanese, we used a subset of ten of these words. The subset we used had approximately 6000 spoken digits, with four tokens from each speaker. The speakers varied in age and sex, but the audio itself was of approximately uniform quality, all at a sample rate of 16 kHz.

2.2 Image-only task

For this task, we duplicated some of the experiments of Hinton et al. (2006), where they tested a Deep Belief Network on the MNIST dataset of images. We used 50 epochs of pre-training for each layer. We performed 50 fine-tuning epochs, sometimes stopping earlier by hand because of time constraints. The images were represented to the network as a raster-style flattening of the two-dimensional array of pixels.

[2]Available at `http://yann.lecun.com/exdb/mnist/`.

[3]Available from the LDC: `http://www.ldc.upenn.edu/`.

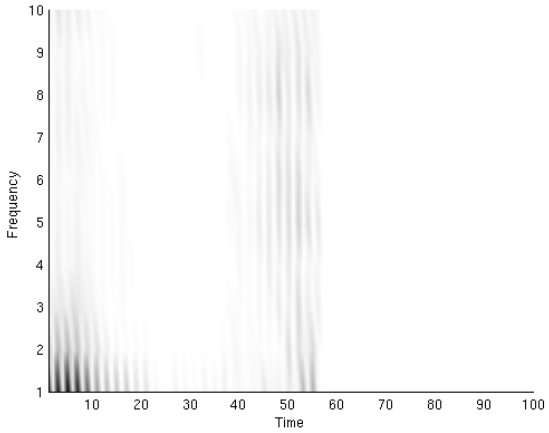

Figure 4: Spectrogram of “ichi”.

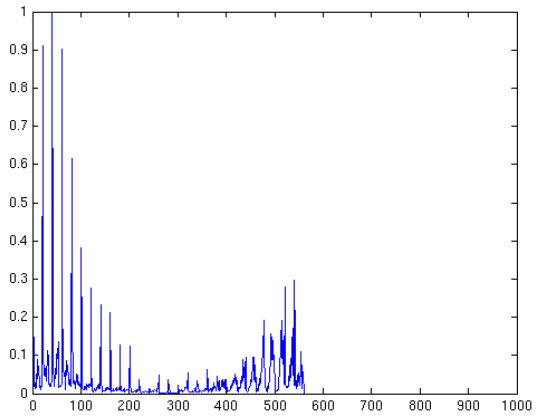

Figure 5: Processed vector of “ichi”.

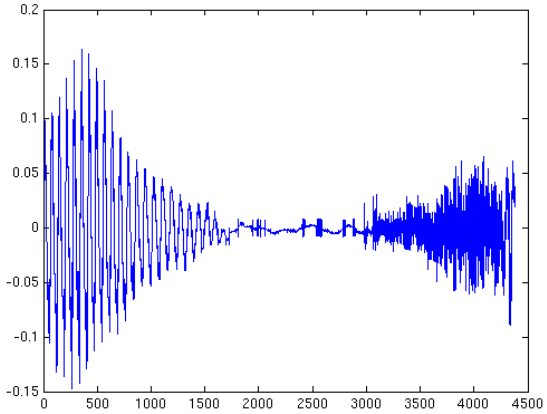

Figure 6: Waveform of “ichi”.

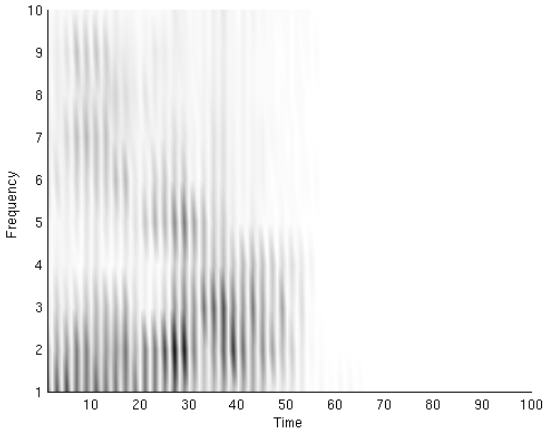

Figure 7: Spectrogram of "zero".

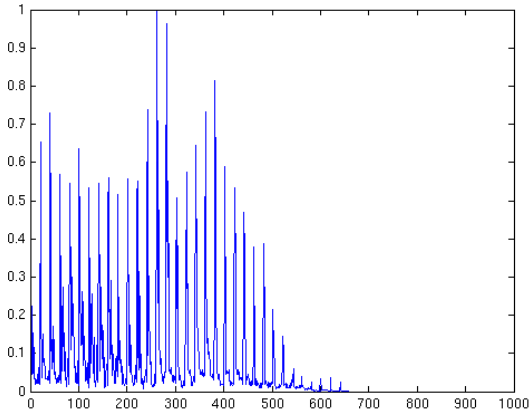

Figure 8: Processed vector of "zero".

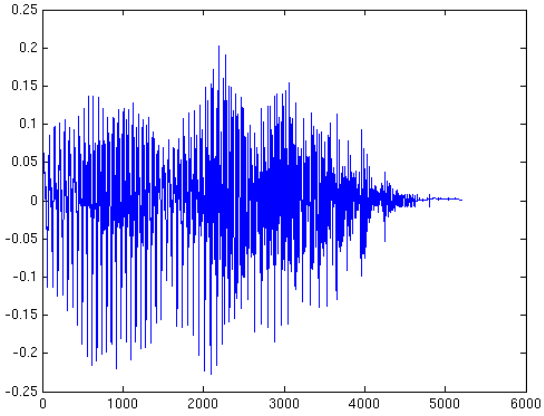

Figure 9: Waveform of "zero".

2.3 Audio-only task

For this task, we adapted the Deep Belief Network to categorize audio input. Our audio representation was simplistic, but sufficient for the network to achieve good categorization accuracy. We automatically trimmed silence from the beginning and end of each audio recording and fixed the length at 0.5 seconds (8000 samples). For recordings that were too short, we trimmed silence from the beginning and padded the end with trailing silence as necessary.

We took Fast Fourier Transforms (FFTs) of each 10ms chunk of the audio, then concatenated these frames into one long vector. In order to reduce the input dimensionality and smooth the spectral information, we averaged every four adjacent frequency components in each frame. This step, unfortunately, made our audio representation not suitable for playback, although in principle it could be avoided. We also normalized the entire vector to have a maximum value of 1.0.

We divided the 6000 tokens in our audio dataset into a training set of 4500 patterns and a test set of 1500 patterns by holding out the fourth utterance of each digit for each of the 150 speakers. This implies that the network was only tested on data from speakers that had been in the training set. Presumably, the network would fare worse on unheard speakers, though we did not have time to test this.

Figures 4 through 9 show human-readable spectrograms, the processed vectors passed to the network, and the original, unprocessed waveform, for "ichi" and "zero". One can see how there is still enough information in this simplistic audio representation to distinguish the words.

2.4 Combined task

Finally, we performed the same classification task on a data set of combined audio and image data. We randomly paired images and audio denoting the same digit to create the input vectors for the combined task. This random process produced a small number of duplicates which we did not bother to remove. The image and audio representations were the same as for the individual tasks above; we simply concatenated them to produce a longer input vector. We expected the accuracy of the network on this task to be higher than in either of the other ones be-

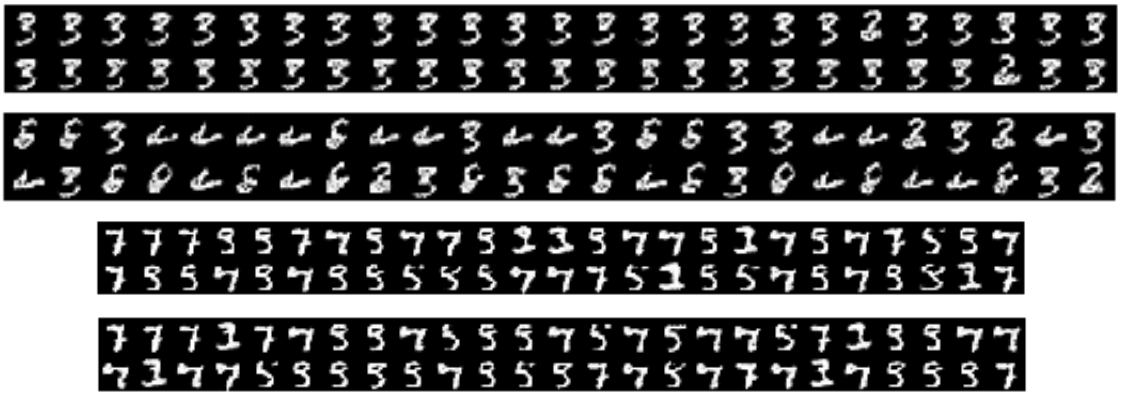

Figure 10: Threes and zeroes generated from the image-only network, and eights and threes generated from the combined network.

cause the network is given strictly more information, which it can use to determine the correct classification. Also, the digits that are most ambiguous when handwritten are not always the digits that are most ambiguous when spoken in Japanese.

Note that there was no requirement that we pair images with audio which denoted the same digit as those images. We could easily have mispaired things, and taught the network that the word "zero" goes with the image 4, for example. Just like our category labels, the pairings of data are totally arbitrary. It is more exciting, though, to use real-world pairings.

3 Results

The system performed very well on all three of our tasks. In the image-only experiment, we, like Hinton et al. (2006), quickly reached 100% accuracy on the training data. We achieved 98.88% classification accuracy on the testing data.

The audio-only task did not fare as well, but this was to be expected, as the dataset was a tenth the size of the one used for the image-only task. The network only reached 92.92% accuracy on the test data (from 95.14% accuracy on the training data) after 200 epochs of fine-tuning. This is evidence for slight over-fitting to the training data. The best way to combat this would be to use more data, and do fewer epochs of fine-tuning.

The combined task did even better than the image only task, which was exactly as hoped. After only 25 epochs of fine-tuning, the combined task achieved an accuracy of 100% on the training data, and 99.69% on the test data.

The network used in Plunkett et al. (1992) never exceeded 85% classification accuracy. Their task was, as stated, somewhat more abstracted, and simpler. Despite using fixed tags rather than real-world, varying audio, their peculiar feed-forward neural net was unsuited to the task.

3.1 Generated images

Figure 10 shows some sample generated images. The top shows the image-only network's idea of what a three looks like. The next shows our attempt at getting that same network to show us a zero; either there is an error in our generating code, or the network really prefers to think about threes. All this would mean is that there is a valley of significantly lower energy for things that look three-like, which is plausible, but somewhat of a problem. We currently believe that we are not correctly conditioning on the class label.

Similarly, the combined network produced images which were recognizably digits, but not the ones we asked it for. For whatever reasons, it preferred to dream of sevens, and would produce them even when we asked for threes or eights. The combined network also generated audio output, of course, but this would not sound recognizable, given our audio representation. The spectrogram of the generated audio looked plausible — it looked like a spectrogram of natural language audio.

4 Conclusions

DBNs are very successful classifiers, and can also act as generative models, which is a very desirable property for our tasks. Though DBNs, like most machine learning algorithms, will always benefit from more data, in the tests we ran, they achieved striking accuracy with a relatively small dataset.

We have trained a DBN to acquire a small vocabulary. Our three tasks together give us two procedures for converting spoken audio into generated images, or vice versa. If we want to generate images of handwritten digits corresponding to a spoken digit, we could do either of the following: first, we could use our image-only and audio-only DBNs together, to classify an input, and then, using the other, generate image or audio based on that classification. Alternatively, we could use the combined network, and fix values on either the audio or image portion of the input, and reconstruct the unknown values to appropriately complete the input vector.

This use of a DBN also addresses, in some sense, the symbol grounding problem. It is exactly this sort of correlation of multi-modal input, particularly through a sub-symbolic representation, that seems to be the only solution to this problem.

References

S. Harnad. 1990. The symbol grounding problem. *Physica D*, 42:335–346.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorith for deep belief nets. *Neural Computation*.

Andrew Ng and Michael Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes.

Kim Plunkett, Chris Sinha, Martin F. Møller, and Ole Strandsby. 1992. Symbol grounding or the emergence of symbols? vocabulary growth in children and a connectionist net. *Connection Science*, 4(3 & 4):293–312.