Knowledge Matters: Importance of Prior Information for Optimization

Çağlar Gülçehre
gulcehrc@iro.umontreal.ca
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC, Canada

Yoshua Bengio
yoshua.bengio@umontreal.ca
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC, Canada

Abstract

We explore the effect of introducing prior information into the intermediate level of neural networks for a learning task on which all the state-of-the-art machine learning algorithms tested failed to learn. We motivate our work from the hypothesis that humans learn such intermediate concepts from other individuals via a form of supervision or guidance using a curriculum. The experiments we have conducted provide positive evidence in favor of this hypothesis. In our experiments, a two-tiered MLP architecture is trained on a dataset with 64x64 binary inputs images, each image with three sprites. The final task is to decide whether all the sprites are the same or one of them is different. Sprites are pentomino tetris shapes and they are placed in an image with different locations using scaling and rotation transformations. The first level of the two-tiered MLP is pre-trained with intermediate level targets being the presence of sprites at each location, while the second level takes the output of the first level as input and predicts the final task target binary event. The two-tiered MLP architecture, with a few tens of thousand examples, was able to learn the task perfectly, whereas all other algorithms (include unsupervised pre-training, but also traditional algorithms like SVMs, decision trees and boosting) all perform no better than chance. We hypothesize that the optimization difficulty involved when the intermediate pre-training is not performed is due to the *composition* of two highly non-linear tasks. Our findings are also consistent with hypotheses on cultural learning inspired by the observations of optimization problems with deep learning, presumably because of effective local minima.

1 Introduction

There is a recent emerging interest in different fields of science for *cultural learning* (Henrich and McElreath, 2003) and how groups of individuals exchanging information can learn in ways superior to individual learning. This is also witnessed by the emergence of new research fields such as "Social Neuroscience". Learning from other agents in an environment by the means of cultural transmission of knowledge with a peer-to-peer communication is an efficient and natural way of acquiring or propagating common knowledge. The most popular belief on how the information is transmitted between individuals is that bits of information are transmitted by small units, called memes, which

share some characteristics of genes, such as self-replication, mutation and response to selective pressures (Dawkins, 1976).

This paper is based on the hypothesis (which is further elaborated in Bengio (2012)) that human culture and the evolution of ideas have been crucial to counter an optimization difficulty: this optimization difficulty would otherwise make it intractable for human brains to capture high level knowledge of the world. Here we use machine learning experiments to investigate some elements of this hypothesis by seeking answers for the following questions: are there machine learning tasks which are intrinsically hard for a lone learning agent but that may become very easy when intermediate concepts are provided by another agent as additional intermediate learning cues, in the spirit of Curriculum Learning (Bengio *et al.*, 2009a)? What makes such learning tasks more difficult? Can we verify that we are dealing with an optimization and local minima issue of deep networks, i.e., that using the same architecture but only changing initial conditions can change the outcome from complete success to complete failure? These are the questions discussed (if not completely addressed) here, which relate to the following broader question: how can humans (and potentially one day, machine) learn complex concepts?

In this paper, we present results on an artificial learning task involving binary 64×64 images. Each image in the dataset contains 3 pentomino tetris sprites (simple shapes). The task is to figure out if all the sprites in the image are same or if there is a different sprite in the image. We have tested several state of art machine learning algorithms and none of them could perform better than a random predictor on the test set. Nevertheless by providing hints about the intermediate concepts (the presence of particular sprite classes), the problem can easily be solved where the same-architecture neural network without the intermediate concepts guidance fails. We also find current unsupervised pre-training algorithms to fail solve this problem. We demonstrate this issue with a two-tiered neural network, pre-training the first level to recognize the category of sprites independently of their orientation and scale at different locations, while the second level learns from the output of the first level and predicts the binary task of interest.

Of course, the objective is not to propose a novel learning algorithm or architecture, but rather to refine our understanding of the learning difficulties involved with composed tasks (here a logical formula composed with the detection of object classes), in particular the optimization difficulties involved for deep neural networks. The results also bring empirical evidence in favor of some of the hypotheses from Bengio (2012), discussed below, as well as introducing a particular form of curriculum learning (Bengio *et al.*, 2009a).

1.1 Curriculum Learning and Cultural Evolution Against Local Minima

The idea that learning can be enhanced by guiding the learner through intermediate easier tasks is old, starting with animal training (shaping) (Skinner, 1958; Peterson, 2004; Krueger and Dayan, 2009). Bengio *et al.* (2009a) introduce a computational hypothesis related to a presumed optimization difficulty of directly learning the target task: the good solutions correspond to hard-to-find-by-chance local minima, and intermediate tasks prepare the learner's internal configuration (parameters) in a way similar to continuation methods in global optimization (which go through a sequence of intermediate optimization problems, starting with a convex one where local minima are no issue, and gradually morphing into the target task of interest).

In a related vein, Bengio (2012) makes the following inferences based on experimental observations of deep learning and neural network learning:

Point 1: Training deep architectures is easier when some hints are given about the function that the intermediate levels should compute (Hinton *et al.*, 2006; Weston *et al.*, 2008; Salakhutdinov and Hinton, 2009; Bengio, 2009). *The experiments performed here expand in particular on this point.*

Point 2: It is much easier to train a neural network with supervision (where we provide it examples of when a concept is present and when it is not present in a variety of examples) than to expect unsupervised learning to discover the concept (which may also happen but usually leads to poorer renditions of the concept).

Point 3: Directly training all the layers of a deep network together not only makes it difficult to exploit all the extra modeling power of a deeper architecture but in many cases it actually

yields worse results as the number of *required layers* is increased (Larochelle *et al.*, 2009; Erhan *et al.*, 2010). *The experiments performed here also reinforce that observation.*

Point 4: Erhan *et al.* (2010) observed that no two training trajectories end up in the same local minimum, out of hundreds of runs. This suggests that the number of functional local minima (i.e. corresponding to different functions, each of which possibly corresponding to many instantiations in parameter space) must be very large.

Point 5: Unsupervised pre-training, which changes the initial conditions of the descent procedure, sometimes allows to reach substantially better local minima (in terms of generalization error!), and these better local minima do not appear to be reachable by chance alone (Erhan *et al.*, 2010). *The experiments performed here provide another piece of evidence in favor of explanatory hypotheses based on an optimization difficulty due to local minima.*[1]

Based on the above points, Bengio (2012) then proposed the following hypotheses regarding learning of high-level abstractions.

- **Optimization Hypothesis:** A biological agent performs an approximate optimization with respect to some implicit objective function when it learns.
- **Deep Abstractions Hypothesis:** Higher level abstractions represented in brains require deeper computations (involving the composition of more non-linearities).
- **Local Descent Hypothesis:** The brain of a biological agent relies on approximate local descent and gradually improves itself while learning.
- **Local Minima Hypothesis:** The learning process of a single human learner (not helped by others) is limited by effective local minima.
- **Deeper Harder Hypothesis:** The effect of local minima becomes more critical as the required depth of the architecture increases.
- **Abstractions Harder Hypothesis:** High-level abstractions are unlikely to be discovered by a single human learner by chance, because these abstractions are represented by a deep subnetwork of the brain.
- **Guided Learning Hypothesis:** A human brain can learn high level abstractions if guided by the signals produced by other agents that acts as hints or indirect supervision for these high-level abstractions.
- **Memes Divide-and-Conquer Hypothesis:** Linguistic exchange, individual learning and the recombination of memes constitute an efficient evolutionary recombination operator in the meme-space. This helps human learners to *collectively* build better internal representations of their environment, including fairly high-level abstractions.

This paper is focused on "*Point 1*" and testing the "*Guided Learning Hypothesis*", using machine learning algorithms to provide experimental evidence. The experiments performed also provide evidence in favor of the "*Deeper Harder Hypothesis*" and associated "*Abstractions Harder Hypothesis*". Machine Learning is still far beyond the current capabilities of humans, and it is important to tackle those obstacles to approach AI. For this purpose, we are particularly interested in tasks that humans learn effortlessly from very few examples, while machine learning algorithms fail miserably.

2 Culture and Optimization Difficulty

As hypothesized in the "*Local Descent Hypothesis*", human brains would rely on a local approximate descent, just like a multi-layer Perceptron. The main argument in favor of this hypothesis relies on the biologically-grounded assumption that although firing patterns in the brain change rapidly, synaptic strengths underlying these neural activities change only gradually, making sure that behaviors are generally consistent across time. If a learning algorithm is based on a form of local (e.g. gradient-based) descent, it can be sensitive to local minima (Bengio, 2012). Note that throughout

[1] Recent work showed that rather deep feedforward networks can be very successfully trained when large quantities of labeled data are available (Ciresan *et al.*, 2010; Glorot *et al.*, 2011; Krizhevsky *et al.*, 2012). Nonetheless, the experiments reported here suggest that it all depends on the task being considered, since even with very large quantities of labeled examples, the deep networks trained here were unsuccessful.

this paper, when talking about "local minima", we refer to the local minima of the generalization error. We are mostly interested in the online setting where the online gradient (associated with the next example) is an unbiased estimator of the gradient of generalization error.

When one trains a neural network, at some point in the training phase the evaluation of error seems to saturate, even if new examples are introduced. In particular Erhan *et al.* (2010) find that early examples have a much larger weight in the final solution. It looks like the learner is stuck in or near a local minimum. But since it is difficult to verify if this is near a true local minimum or simply an effect of strong ill-conditioning, we call such a "stuck" configuration an *apparent local minimum*, whose definition depends not just on the optimization objective but also on the limitations of the optimization algorithm.

Erhan *et al.* (2010) highlighted both the issue of apparent local minima and a regularization effect when initializing a deep network with unsupervised pre-training. Interestingly, as the network gets deeper the effect of local minima seems to be get more pronounced. That might be because of the number of local minima increases, or maybe the good ones are harder to reach.

As a result of Point 4 we hypothesize that it is very difficult for an individual's brain to discover some higher level abstractions by chance only. As mentioned in the "*Guided Learning Hypothesis*" humans get hints from other humans and learn high-level concepts by the guidance of other humans[2]. Curriculum learning (Bengio *et al.*, 2009b) and incremental learning (Solomonoff, 1989), are examples of this. This is done by properly choosing the sequence of examples seen by the learner, where simpler examples are introduced first and more complex examples shown when the learner is ready for them. The hypothesis about why curriculum works states that curriculum learning acts as a continuation method that allows one to discover a good minimum, by first finding a good minimum of a smoother error function. Recent experiments on human subjects also shows that humans *teach* using a curriculum strategy Khan *et al.* (2011).

Some parts of the human brain are known to have a hierarchical organization (i.e. visual cortex) consistent with the deep architecture studied in machine learning papers. As we go from the sensory level to higher levels of the visual cortex, we find higher level areas corresponding to more abstract concepts.

Training neural networks and machine learning algorithms by guidance and giving prior information about the task is well-established and in fact constitutes the main approach to solving industrial problems with machine learning. The contribution of this paper is rather on rendering the optimization issue explicit and providing evidence on the type of problems for which this optimization issues arise. This prior information/hint can be viewed as an inductive bias for a particular task which is a necessary apparatus to obtain a good generalization error (Mitchell, 1980). One of the most interesting and earlier findings in that line of research was done with Explanation Based Neural Networks (EBNN) in which a neural network transfers knowledge across multiple learning tasks. An EBNN uses previously learned domain knowledge as a initialization or search bias (i.e. to constrain the learner in the parameter space) (O'Sullivan, 1996; Mitchell and Thrun, 1993).

Another related work in machine learning is mostly focused on reinforcement learning algorithms, based on incorporating prior knowledge in terms of logical rules to the learning algorithm as a prior knowledge to speed up and bias learning (Kunapuli *et al.*, 2010; Towell and Shavlik, 1994).

3 Experimental Setup

Some tasks, which seem reasonably easy for humans to learn, are nonetheless appearing almost impossible to learn for current state-of-art machine learning algorithms. Here we study more closely a task which becomes learnable if one provides to the learner hints about appropriate intermediate concepts. Interestingly, the task we used in our experiments is not only hard for deep neural networks but also for non-parametric machine learning algorithms such as SVMs, boosting and decision trees.

[2]But some high-level concepts may also be hardwired in the brain, as assumed in the universal grammar hypothesis (Montague, 1970), or in nature vs nurture discussions in cognitive science.

3.1 Pentomino Dataset

In order to test our hypothesis, we designed an artificial dataset for object recognition using 64×64 binary images[3]. If the task is two tiered (i.e., with guidance provided), the task in the first level is to recognize each pentomino object class[4] in the image. The second level and final task is to figure out if all the pentominos in the image are of the same class or not. Hence after a neural network learned the categories of each object in the dataset, the task becomes a form of noisy XOR operation between the object categories detected in the image. The types of pentomino objects that we have used for generating the dataset are as follows:

1. Pentomino L sprite
2. Pentomino N sprite
3. Pentomino P sprite
4. Pentomino F sprite
5. Pentomino Y sprite
6. Pentomino J sprite
7. Pentomino N2 sprite: Mirror of "Pentomino N" sprite.
8. Pentomino Q sprite
9. Pentomino F2 sprite: Mirror of "Pentomino F" sprite.
10. Pentomino Y2 sprite: Mirror of "Pentomino Y" sprite.

As shown in Figures 1 and 2, the synthesized images are fairly simple and do not have any texture. Foreground pixels are "1" and background pixels are "0". Images are generated iid. For notational convenience, assume that the domain of raw input images is X, the set of sprites is S, the set of intermediate object categories is Y for each possible location in the image and the set of final task outcomes is Z. We perform two different types of rigid body transformation: sprite rotation $rot(X, \gamma)$ where $\Gamma = \{\gamma\colon (\gamma = 90 \times \phi) \wedge [(\phi \in \mathbb{N}), (0 \leq \phi \leq 3)]\}$ and scaling $scale(X, \alpha)$ where $\alpha \in \{1, 2\}$ is the scaling factor. The data generative procedure is summarized below.

Sprite transformations: Before placing the sprites in an empty image, for each image $x \in X$, we randomly decide on the $z \in Z$ to have (or not) a different sprite in the image. Conditioned on the constraint given by z, we randomly select three sprites s_{ij} from S without replacement. Using a uniform probability distribution over all possible scales, we choose a scale and accordingly scale each sprite image. Then we randomly rotate each sprite by a multiple of 90 degrees.

Sprite placement: Upon completion of sprite transformations, we generate a 64×64 uniform grid divided into 8×8 blocks, each block being of size 8×8 pixels, and randomly select three different blocks from the 64=8×8 on the grid and place the transformed objects into *different* blocks.

Each sprite is centered in the block in which it is located. Thus there is no object translation inside the blocks. The only translation invariance is due to the location of the block inside the image.

A pentomino sprite is guaranteed to not to overflow the block in which it is located, and there are no collisions or overlaps between sprites, making the task simpler. The largest possible pentomino sprite can be fit into an 8×4 mask.

3.2 Learning Algorithms Evaluated

Cross-validation

We have first cross-validated our models by using 5-fold cross-validation on thirty two thousand examples for training and eight thousand examples for testing.

In all neural network algorithms, we have used SGD and backpropagation for training.

[3]The source code for the script that generates the artificial pentomino datasets (Arcade-Universe) is available at: `https://github.com/caglar/Arcade-Universe`. This implementation is based on Olivier Breuleux's bugland dataset generator.

[4] For a human learner, knowing the category of each pentomino sprite in the image seems unnecessary.

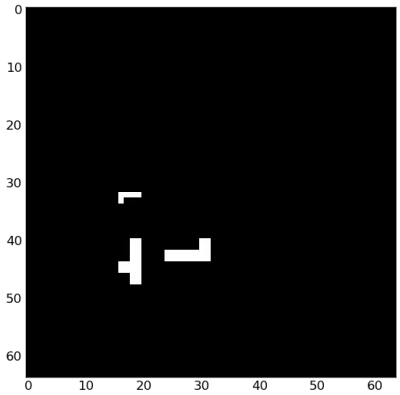

Figure 1: An example image from the dataset which has *different object types* in it.

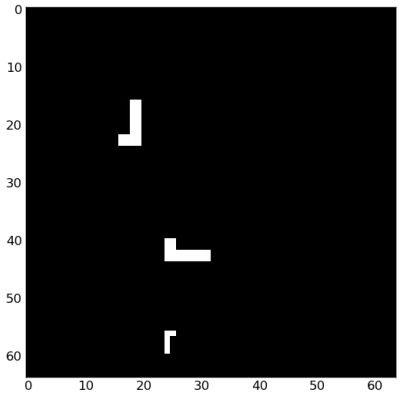

Figure 2: An example image from the dataset that has only one type of pentomino object in it, but with different orientations and scales.

3.2.1 Decision Trees

We used the decision tree implementation in the scikit-learn (Pedregosa *et al.*, 2011) python package which is an implementation of the CART (Regression Trees) algorithm. The CART algorithm constructs the decision tree recursively and partitions the input space such that the samples belonging to the same category are grouped together (Olshen and Stone, 1984). We used The Gini index as the impurity criteria. We evaluated the hyper-parameter configurations with a grid-search. We cross-validated the maximum depth (max_depth) of the tree (for preventing the algorithm to severely overfit the training set) and minimum number of samples required to create a split (min_split). 20 different configurations of hyper-parameter values were evaluated. We obtained the best validation error with $max\_depth = 300$ and $min\_split = 8$.

3.2.2 Support Vector Machines

We used the "Support Vector Classifier (SVC)" implementation from the scikit-learn package which in turn uses the libsvm's Support Vector Machine (SVM) implementation. SVM is a non-parametric technique that maps the data into a high dimensional space (if a kernel is used) and separates different classes with hyperplane(s) such that the support vectors for each category will be separated by a large margin. We cross-validated three hyper-parameters of the model using grid-search: C, γ and the type of kernel($kernel\_type$). C is the penalty term (weight decay) for the SVM and γ is a hyper-parameter that controls the width of the Gaussian for the RBF kernel. For the polynomial kernel, γ controls the flexibility of the classifier (degree of the polynomial) as the number of parameters increases (Hsu *et al.*, 2003; Ben-Hur and Weston, 2010). We evaluated forty-two hyper-parameter configurations. That includes, two kernel types: $\{RBF,\ Polynomial\}$; three gammas: $\{1e-2,\ 1e-3,\ 1e-4\}$ for the RBF kernel, $\{1, 2, 5\}$ for the polynomial kernel, and seven C values: $\{0.1, 1, 2, 4, 8, 10, 16\}$ values. As a result of the grid search and cross-validation, we have obtained the best test error by using the RBF kernel, with $C = 2$ and $\gamma = 1$.

3.2.3 Multi Layer Perceptron

We have our own implementation of Multi Layer Perceptron based on Theano. We have used 2 hidden layers and RELU (Rectified Linear Units) activation function. We used 2048 hidden units per layer. We cross-validated three hyper-parameters of the model using random-search in sampled the learning rates from the log-space: ϵ (learning rate), $L1$ penalty, $L2$ penalty and evaluated 64 hyperparameter values. The range of the hyperparameter values for $\epsilon = [0.0001, 1]$, $L1 = 0., 1e-6, 1e-5, 1e-4$ and $L2 = 0, 1e-6, 1e-5$. As a result of the random search we have used $L1 = 1e-6$, $L2 = 1e-5$ and $\epsilon = 0.05$.

3.2.4 Random Forests

We used scikit-learn's implementation of "Random Forests" decision tree learning. Random Forests algorithm creates an ensemble of decision trees by randomly selecting for each tree a subset of features and apply bagging to combine the individual decision trees (Breiman, 2001). We have used grid-search and cross-validated the max_depth, min_split, and number of trees ($n\_estimators$). We have done the grid-search on the following hyperparameter values, $n\_estimators = \{5, 10, 15, 25, 50\}$, $max\_depth = \{100, 300, 600, 900\}$, and $min\_splits = \{1, 4, 16\}$. We obtained the best validation error with $max\_depth = 300$, $min\_split = 4$ and $n\_estimators = 10$.

3.2.5 k-Nearest Neighbors

We used scikit-learn's implementation of k-Nearest Neighbors (k-NN). k-NN is an instance-based, lazy learning algorithm that selects the training examples closest in Euclidean distance to th etest example. It assigns a class label to the test example based on the category of the k closest neighbors. The hyper-parameters, we have evaluated in the cross-validation are, number of neighbors (k) and $weights$. The $weights$ hyper-parameter can be either "uniform" or "distance". With "uniform", the value assigned to the query point is computed by the majority vote of the nearest neighbours. With "distance", each value assigned to the query point is computed by weighted majority votes where the weights are computed with the inverse distance between the query point and the neighbors. We have

used $n\_neighbours = \{1, 2, 4, 6, 8, 12\}$ and $weights = \{"uniform", "distance"\}$ for hyper-parameter search. As a result of cross-validation and grid search, we obtained the best validation error with $k = 2$ and $weights$="uniform".

3.2.6 Convolutional Neural Nets

We used a Theano implementation of Convolutional Neural Networks (CNN) from the deep learning tutorial on `deeplearning.net` which is based on a vanilla version of a CNN LeCun *et al.* (1998). Our CNN has two convolutional layers. Following each convolutional layer, we have a max-pooling layer. In the crossvalidation we have sampled 36 learning rates from logspace in the range $[0.0001, 1]$ and number of kernels from the range $[10, 20, 30, 40, 50, 60]$ uniformly. For the first convolutional layer we used 9×9 receptive fields in order to guarantee that each object fits inside the receptive field. The number of features for the first layer is 30. For the second convolutional layer, we used 7×7 receptive fields with 60 features. The stride for both convolutional layers is 1. We downsample convolved images by a factor of 2 after each pooling operation. The selected learning rate for the CNN is 0.01 and we have used 8 training epochs.

3.2.7 Stacked Denoising Autoencoders

Denoising Autoencoders (DA) are a form of stochastic autoencoder. DA forces the hidden layer to discover more robust features and prevent it from simply learning the identity by reconstructing the input from a corrupted version of it (Vincent *et al.*, 2010). We used Stacked DA's, stacking two DAs, resulting in an unsupervised transformation with two hidden layers. Parameters of all layers are then fine-tuned with supervised fine-tuning using logistic regression as the classifier and SGD as gradient based optimization algorithm. We used 1024 hidden units and 0.2 as the corruption level with binomial corruption. We've manually tried different learning rates for the DA and the supervised fine-tuning. The selected learning rate is $\epsilon_0 = 0.01$ for DA and $\epsilon_1 = 0.1$ for supervised fine-tuning. Both L1 and L2 penalty for DA's and logistic regression are set to 1e-6.

3.2.8 Intermediate Knowledge Guided Neural Network (IKGNN)

The IKGNN is a two-level deep neural network, in which the first level's training objective is the detection and classification of the pentomino sprite classes in an 8×8 patch. The Level 1 Neural Net (L1NN) is applied to each of the 8×8=64 non-overlapping patches of the 64×64 input image, in order to produce the input for the Level 2 Neural Net (L2NN). L1NN is trained with the intermediate target Y. Y specifies the type of (if any) pentomino sprite present for each patch of the image. Because a possible answer at a given location can be "none of the object type/empty patch", y_p (for patch p) can take 11 values, 1 for rejection and the rest is for the 10 different pentomino classes.

$$y_p = \begin{cases} 0 & \text{if patch } p \text{ is empty} \\ s \in S & \text{if the patch } p \text{ contains a pentomino sprite} \end{cases}$$

The IKGNN architecture is a two tiered network that takes advantage during training of prior information about intermediate-level relevant factors. Because the sum of the training losses decomposes into the loss on each patch, the L1NN can be pre-trained patch-wise. Each patch-specific component of the L1NN is a fully connected MLP with 8×8 inputs and 11 outputs with a softmax output layer. As seen on Figure 3 we trained the L1NN with respect to the intermediate target values (Y) and concatenate these outputs (for all the patches) into a one large vector (64×11). Then we standardize this output vector by subtracting its mean (over training examples) and dividing by its standard deviation (over training examples) for each element of the output vector. This normalized output vector of L1NN (of length 704) is then fed to the L2NN MLP, which has a single binomial output unit for the final task probability prediction (with a sigmoid unit) for the binary event Z, as seen on Figure 4.

IKGNN uses rectifier hidden units as activation function, $max(0,\ X)$. We found a significant boost by using rectification compared to hyperbolic tangent and sigmoid activation functions. The L1NN has a highly overcomplete architecture with 2048 hidden units per patch, and L1 and L2 weight decay regularization terms are respectively, 1e-6 and 1e-5. The learning rate for the L1NN is 0.75. 2 training epochs were enough for the IKGNN to learn the features of the first layer. The L2NN has

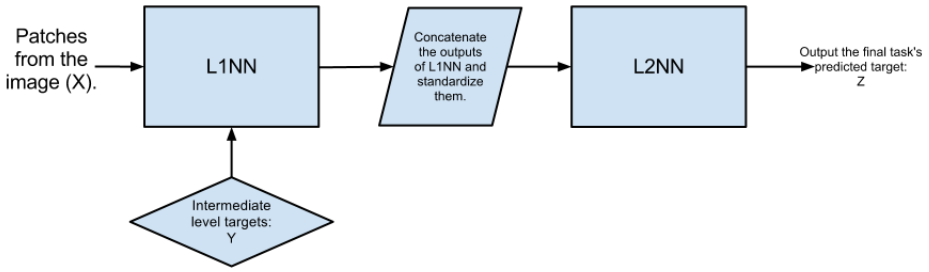

Figure 3: Information flow diagram for the IKGNN. L1NN is trained on the patches with respect to intermediate target labels and L2NN is trained on the concatenated and standardized output of L1NN with respect to the final task's target labels.

1024 hidden units. L1 and L2 penalty terms for the L2NN is 1e-6 with a learning rate of 0.1. Both L1NN (patch-wise) and L2NN are fully connected neural networks.

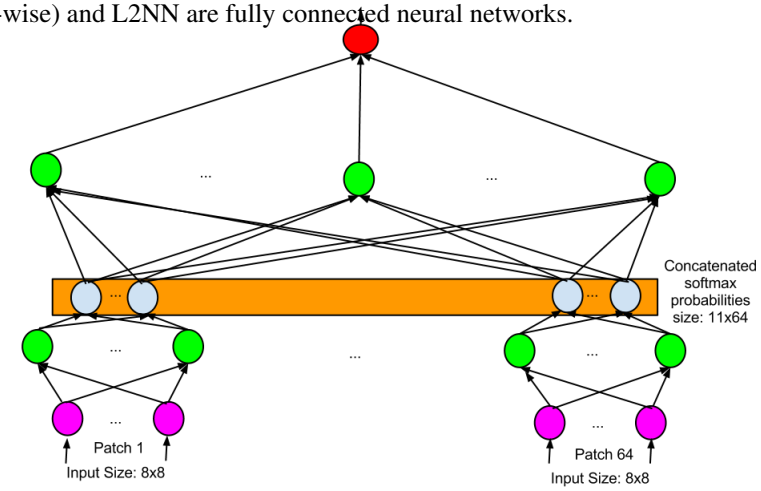

Figure 4: A basic architectural overview of the IKGNN. L1NN is trained on each 8x8 patch extracted from the image and the softmax output probabilities of all 64 patches are concatenated into a 64x11 vector.

3.2.9 Deep and Structured MLP without Hints

We have used the same connectivity pattern (and rather deep architecture) that we used for the IKGNN but without using the intermediate targets (Y) and we directly predict the final outcome of the task (Z) by using the same number of hidden units, same connectivity and same activation function for the hidden units. We have evaluated 120 hyperparameter values by randomly selecting number of hidden units from $[64, 128, 256, 512, 1024, 2048]$, L1 penalty values from $[1e-6, 1e-5, 1e-4]$, L2 penalty values from $[1e-5, 1e-4, 1e-3, 1e-2]$ and randomly sampled 20 learning rates from the logspace in the $[0.008, 0.8]$. We used two fully connected hidden layers with the hidden units of size 1024 (same as L1NN) per patch and 2048 (same as L2NN) with twenty training epochs. For this network we have obtained the best results with learning rate 0.05.

4 Experimental Results and Analysis

This section provides results of experiments that we did on the pentomino dataset with different number of training examples in order to see the effect of introducing intermediate knowledge with respect to the size of training set. For the experimental results shown on 4 we have used 3 training set sizes (20k, 40k and 80k examples), generated with different random seeds (so they don't overlap). Figure 1 shows the error bars for an ordinary MLP with two hidden layers. For that MLP, the number of training epochs is 8, and there are two hidden layers with 4096 feature detectors. The learning rate we used in our experiments is 0.01. The activation function of the MLP is tanh nonlinearity and L1, L2 penalty are both 1e-6.

Algorithm	20k dataset		40k dataset		80k dataset	
	Training Error	Test Error	Training Error	Test Error	Training Error	Test Error
SVM RBF	26.23	50.24	28.175	50.25	30.2225	49.62
KNN	24.7	50.02	25.3025	49.51	25.613	49
Decision Tree	5.84	48.6	6.285	49.41	6.89625	49.87
Randomized Trees	3.23	49.81	3.44	50.49	3.532	49.06
MLP	26.515	49.26	33.16	49.91	27.22875	50.13
CNN	50.557	49.759	49.447	49.759	50.23	49.759
2 layer sDA	49.42	50.32	50.225	50.32	49.75	50.32
Deep structured MLP without Hints	50.54	49.86	49.81	49.68	49.71	50.27
IKGNN	**0.215**	**30.68**	**0**	**3.09**	**0**	**0.01**

Table 1: The error percentages with different learning algorithms on pentomino dataset with different number of training examples.

The L1NN learns to classify patches with respect to the intermediate concepts very quickly. According to our experiments five thousands training examples are enough for the L1NN to learn the intermediate targets perfectly.

Table 4 shows that, without guiding hints, none of the state of art learning algorithms could perform noticeably better than a random predictor on the test set. This shows the importance of intermediate hints introduced in the IKGNN. The decision trees and SVMs can overfit the training set but they could not generalize on the test set.

In the experiment results in the 5 we have used a MLP with three hidden layers and used $tanh(.)$ as the activation function where each layer has 2048 hidden units with 0.05 learning rate.

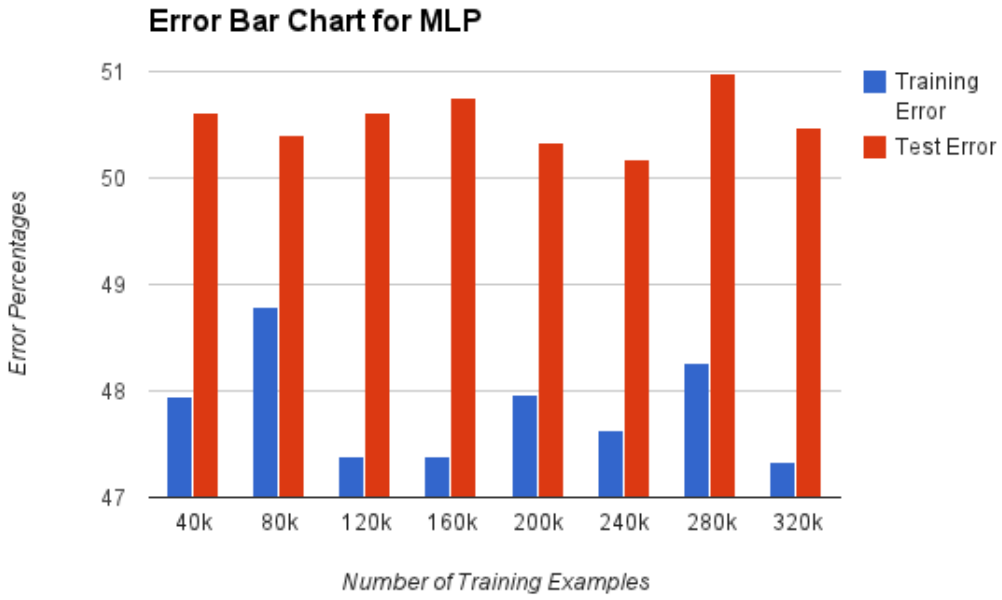

Figure 5: Training and test error bar charts for a regular MLP with 3 hidden layers. There is no significant improvement on the generalization error of the MLP as the new training examples are introduced.

5 Conclusion and Discussion

In this paper we have shown an example of task which seems almost impossible to solve by standard black-box machine learning algorithms, but can be almost perfectly solved when introducing an intermediate pre-trained representation guided by prior knowledge. The task has the particularity that it is defined by the composition of two non-linear sub-tasks (object detection on one hand, and a non-linear logical operation similar to XOR on the other hand).

What is interesting is that in the case of the neural network, we can compare two networks with exactly the same architecture but a different pre-training, one of which uses the known intermediate concepts to teach an intermediate representation to the network. Without using these intermediate targets the neural network has failed to learn the task, both on the training set and in terms of generalization, showing a clear **optimization** issue. The situation remains the same even as we increase the training set size 8-fold.

These findings also bring supporting evidence to the "Guided Learning Hypothesis" and "Deeper Harder Hypothesis" from Bengio (2012): higher level abstractions, which are expressed by composing simpler concepts, are more difficult to learn (with the learner often getting in an effective local minimum), but that difficulty can be overcome if another agent provides hints of the importance of learning other, intermediate-level abstractions which are relevant to the task.

Many interesting questions remain open. Would a network without any guiding hint eventually find the solution with a enough training time, i.e., is the optimization difficulty due to ill-conditioning or due to a local minimum? Clearly, one can reach good solutions from an appropriate initialization, pointing in the direction of a local minima issue, but it may be that good solutions are also reachable from other initializations, albeit going through a tortuous ill-conditioned path in parameter space. Why did our attempts at learning the intermediate concepts in an unsupervised way fail? Are these results specific to the task we are testing or a limitation of the unsupervised feature learning algorithm tested? Trying with many more unsupervised variants and exploring explanatory hypotheses for the observed failures could help us answer that. Finally, and most ambitious, can we solve these kinds of problems if we allow a community of learners to collaborate and collectively discover and combine partial solutions in order to obtain solutions to more abstract tasks like the one presented here? Indeed, we would like to discover learning algorithms that can solve such tasks without the use of prior knowledge as specific and strong as the one used in the IKGNN here. These experiments could be inspired by and inform us about potential mechanisms for collective learning through cultural evolutions in human societies.

References

Ben-Hur, A. and Weston, J. (2010). A user's guide to support vector machines. *Methods in Molecular Biology*, **609**, 223–239.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, **2**(1), 1–127. Also published as a book. Now Publishers, 2009.

Bengio, Y. (2012). Evolving culture vs local minima. Technical Report ArXiv 1203.2990v1, Université de Montréal.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009a). Curriculum learning. In *ICML'09*.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009b). Curriculum learning. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM.

Breiman, L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32.

Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, **22**, 1–14.

Dawkins, R. (1976). *The Selfish Gene*. Oxford University Press.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, **11**, 625–660.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *AISTATS'2011*.

Henrich, J. and McElreath, R. (2003). The evolution of cultural evolution. *Evolutionary Anthropology: Issues, News, and Reviews*, **12**(3), 123–135.

Hinton, G. E., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, **18**, 1527–1554.

Hsu, C., Chang, C., Lin, C., *et al.* (2003). A practical guide to support vector classification.

Khan, F., Zhu, X., and Mutlu, B. (2011). How do humans teach: On curriculum learning and teaching dimension. In *Advances in Neural Information Processing Systems 24 (NIPS'11)*, pages 1449–1457.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Krueger, K. A. and Dayan, P. (2009). Flexible shaping: how learning in small steps helps. *Cognition*, **110**, 380–394.

Kunapuli, G., Bennett, K., Maclin, R., and Shavlik, J. (2010). The adviceptron: Giving advice to the perceptron. *Proceedings of the Conference on Artificial Neural Networks In Engineering (ANNIE 2010)*.

Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, **10**, 1–40.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

Mitchell, T. (1980). *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ.

Mitchell, T. and Thrun, S. (1993). Explanation-based neural network learning for robot control. *Advances in Neural information processing systems*, pages 287–287.

Montague, R. (1970). Universal grammar. *Theoria*, **36**(3), 373–398.

Olshen, L. and Stone, C. (1984). Classification and regression trees. *Belmont, Calif.: Wadsworth*.

O'Sullivan, J. (1996). Integrating initialization bias and search bias in neural network learning.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., *et al.* (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, **12**, 2825–2830.

Peterson, G. B. (2004). A day of great illumination: B. F. Skinner's discovery of shaping. *Journal of the Experimental Analysis of Behavior*, **82**(3), 317–328.

Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, volume 8.

Skinner, B. F. (1958). Reinforcement today. *American Psychologist*, **13**, 94–99.

Solomonoff, R. (1989). A system for incremental learning based on algorithmic probability. In *Proceedings of the Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pages 515–527. Citeseer.

Towell, G. and Shavlik, J. (1994). Knowledge-based artificial neural networks. *Artificial intelligence*, **70**(1), 119–165.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, **11**, 3371–3408.

Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1168–1175, New York, NY, USA. ACM.