A Deep Belief Network Approach to Learning Depth from Optical Flow

Reuben Feinman
Honors Thesis
Division of Applied Mathematics, Brown University
Providence, RI 02906
`Advised by Thomas Serre and Stuart Geman`

Abstract

It is well established that many living beings make use of motion information encoded in the visual system as a means to understand the surrounding environment and to assist with navigation tasks. In particular, a phenomenon referred to as motion parallax is known to be instrumental in the perception of depth when binocular cues are distorted or otherwise unavailable [6]. The key idea is that objects which are closer to a given observer appear to move faster through the visual field than those at a further distance. Due to this phenomenon, optical flow can be a very powerful tool for the decoding of spatial information. When exact data is available regarding the speed and direction of optical flow, recovering depth is trivial [1]. In general, however, sources of this information are unreliable, therefore a learning approach can come of great use in this domain.

We describe the application of a deep belief network (DBN) to the task of learning depth from optical flow in videos. The network takes as input motion features generated by a hierarchical feedforward architecture of spatio-temporal feature detectors [4]. This system was developed as an extension of the HMAX model [12] and is inspired by the neurobiology of motion processing in the visual cortex. Our network is trained using a set of graphical scenes generated by a gaming engine, such that ground truth depth information is available for each video sequence. We show that, by using such a DBN, we are able to improve considerably over standard linear classifiers. In addition, we show that our results can be further improved by performing Markov random field smoothing on the output depth field map. Our findings suggest that cortex-like mechanisms are a conceivable tool for artificial vision systems that operate on video data.

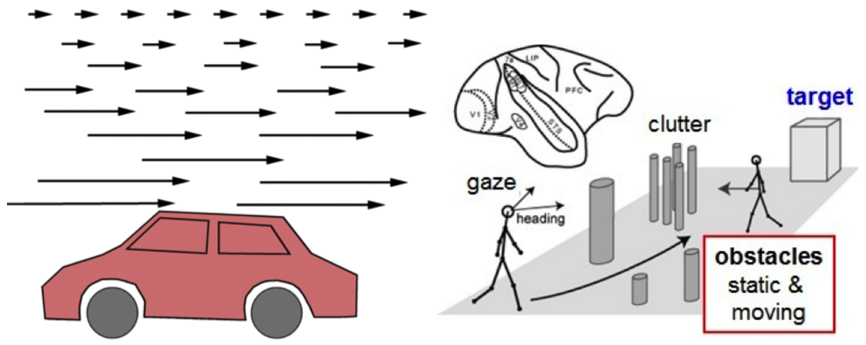
1 Motivation

The human brain is arguably the most powerful information processing device in existence, with a significant portion of its computational power devoted to sensory processing and in particular, vision. Modern deep learning systems have attempted to compete with the brain in the realm of visual perception, drawing loose inspiration from the biological mechanisms of the visual cortex. Despite the latest efforts, however, artificial vision systems have failed to replicate the power and versatility of the brain with regard to vision tasks. Here, we would like to design a computer vision system that relies on biologically inspired neural representations. Rather than owing to elaborate spiking neural networks, our representations are obtained using mathematical abstractions that extend current deep architectures while remaining interpretable in the context of biophysical mechanisms. To demonstrate the power of these representations, we will design a decoding task using artifical scenes so that ground truth information about the environment is available at our disposal. The key contribution of this work is the development of a theoretically-sound learning pipeline that allows for robust decoding of spatial information from biologically inspired neural representations.

2 Optical Flow and Motion Parallax

Research conducted over the past few decades has established that the visual systems of bees and a number of other small insects are finely sensitive to motion cues [13]. These insects cannot rely on the mechanisms of binocular disparity or accommodation that are used by primates to extract 3D spatial information; the form of the eye in these insects suggests that they measure range over long distances by motion parallax as opposed to binocular overlap. It has been shown that motion-sensitive neurons in the visual pathway of bees play a significant role in the process of goal localization [8]. Similarly, in primates, a functional link has been discovered between depth perception and the activity of neurons in the middle temporal (MT) area [5]. These neurons exhibit selectivity to direction and speed of motion, and the middle superior temporal area (MST) uses inputs received from this region to compute optical flow. The processing stages in this region of the brain are believed to assist with the interpretation of spatial relationships and thereby aid with the task of navigation.

source: opticflow.bu.ede



(a) Motion Parallax **(b)** Human Navigation

Figure 1: Motion parralax assists with the decoding of spatial information required for navigation tasks

Motion parallax refers to the concept that objects which are closer to a particular observer move faster through the visual field than those that are further away. A depiction of this phenomenon is presented in Figure 1a. As the observer in this scenario exhibits translational motion, the optical flow detected in each region of his visual field is a function of the environment range for that region. In this diagram, the optical flow is shown to diminish between regions of the visual field corresponding to foreground and those corresponding to distant background. By providing a means for depth perception, optical flow assists with navigation tasks that involve the avoiding of obstacles and the localizing of targets. A diagram of human naviagtion is provided in Figure 1b. As the subject moves through an environment cluttered with obstacles, optical flow helps him determine an optimal path by providing information about the locations of obstacles and targets.

3 The Motion Model

When precise optical flow information is available, including the direction and magnitude of this flow, recovering depth is trivial given a stationary environment. A thorough mathematical analysis of this process is provided in [1]. Equation 5 of this paper describes the mapping from direction and magnitude of optical flow to depth when the motion of the camera is constant and known. Unfortunately, sources of this information are unreliable, and the optical flow data that is received is often obscure. We would like to evaluate the potential of our optical flow feature representations in regards to the task of depth prediction.

To construct optical flow feature representations, we make use of a biologically inspired model of motion processing that closely follows the organization of the visual cortex [4]. The model bears a feedforward hierarchical architecture within which increasingly complex and invariant features are constructed by alternating between layers of template matching and max pooling. A visualization of this model is depicted in Figure 2. During template matching, convolutions are performed on layer inputs with a battery of spatio-temporal filters, tuned to a selection of different speeds and orientations. In the pooling phase, local max pooling is performed over sub-populations in order to obtain invariance to position and scale. The model takes in a 20-frame video sequence, and it outputs a motion energy feature map of dimensionality ($\#Speeds$ x $\#Directions$ x $Height$ x $Width$) for the center frame of this sequence. Figure 3 presents a visualization of the model outputs, using luminance to depict decoded speed and chromenance to depict direction.

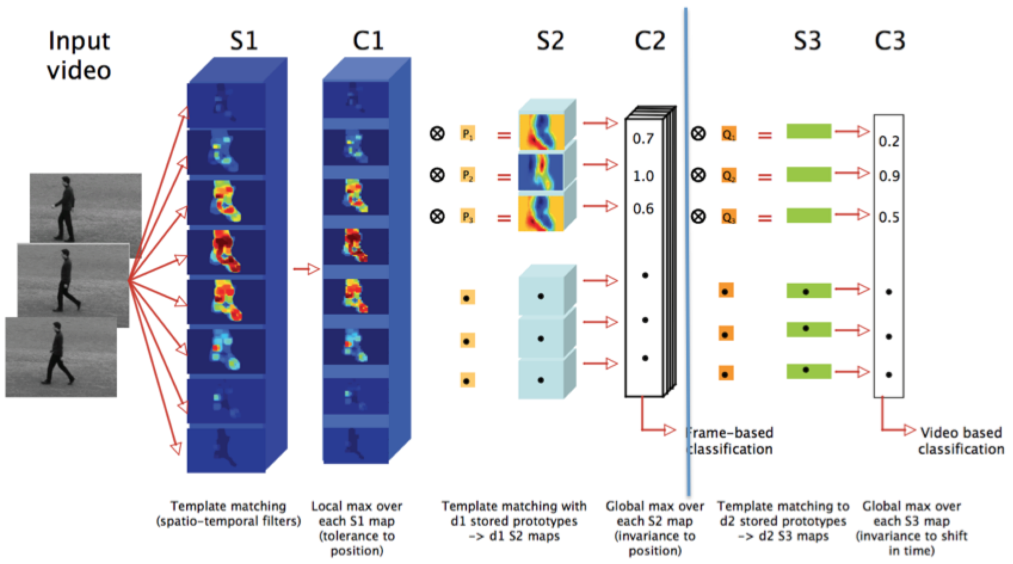

Input video
S1
C1
S2
C2
S3
C3
0.7
1.0
0.6
0.2
0.9
0.5
rame-based lassification
Video based classification
Template matching (spatio-temporal filters)
Local max over each S1 map (tolerance to position)
Template matching with d1 stored prototypes -> d1 S2 maps
Global max over each S2 map (invariance to position)
Template matching to d2 stored prototypes -> d2 S3 maps
Global max over each S3 map (invariance to shift in time)

Figure 2: Sketch of the MT motion model. Our version cuts off at C2 of this diagram. Source: [4]

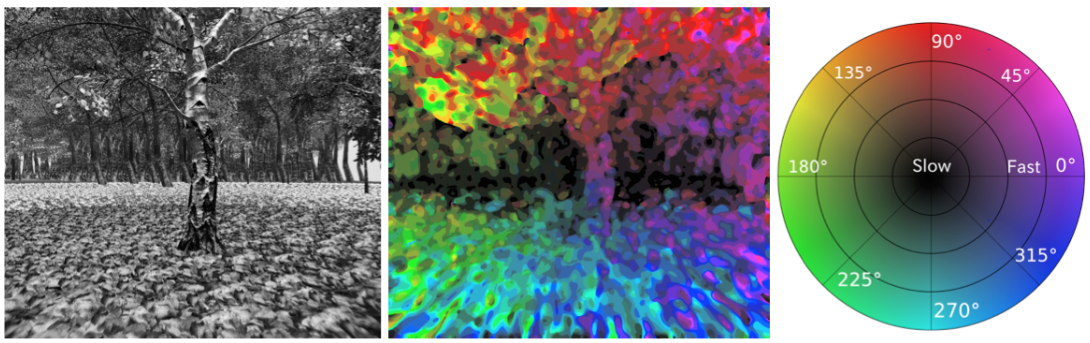


Figure 3: A visualization of motion energy responses. Decoded speeds and directions correspond to an argmax taken over speed and direction feature vectors at each pixel.

The motion processing step can be thought of as a data preprocessing phase; while we make use of the motion model to generate feature representations, we do not alter the model's parameters during training.

4 A Data-Driven Approach

4.1 Deep Learning

Equipped with our motion feature model, we would like to build an algorithm to map from motion features to depth prediction at each pixel of the visual field. One possible approach, it could be argued, would be to develop a function that maps from argmax decoded speed to depth prediction. In following such an approach, however, we naively discard a great deal of motion information provided by our feature representations, reducing the entropy of the information set upon which our model is constructed. As an alternative to such route, we propose a data-driven approach that is grounded in a deep neural network predictive model.

A great deal of progress has been made in the field of deep learning over recent years, driven by a realm of algorithms that consist of multiple layers of nonlinear processing. An example deep learning architecture is depicted in Figure 4. Given a training set of data features and corresponding ground truth labels, these algorithms are able to learn highly complex functions from feature inputs to desired outputs by means of supervised training mechanisms. The performances of these algorithms have been shown to scale significantly with the number of training examples and the number of model parameters [2].

source: www.deeplearning.stanford.edu

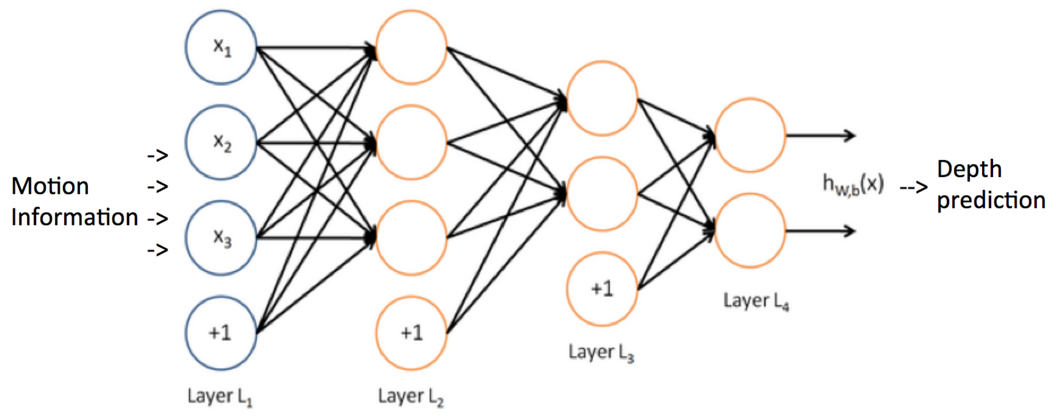


Figure 4: Diagram of an artificial neural network. The network consists of a feedforward hierachical architecture, whereby linear combinations of the outputs from each layer are passed forward and processed by the nodes of the following layer.

At each hidden layer node, linear combinations of outputs from the previous layer are processed through a nonlinearity at the nodes of the current layer so as to facilitate a more complex input to output function for the model. The standard activation function, and the one which we describe in our model, is the sigmoid function. This activation is depicted in Figure 5.

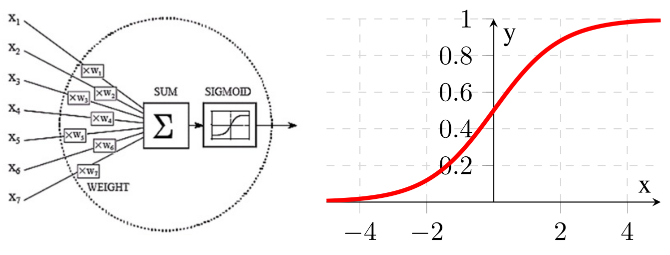


Figure 5: linear combinations of outputs from the previous layer are passed through a sigmoid nonlinearity at each node of a given layer.

$$f(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

4.2 The Backpropagation Algorithm

In building an optimal neural network model, we would like to find the set of weights and biases such that our model's outputs match with greatest possible precision the labels of the training examples that it is fed. To do so, we compose a cost function that details how well our network outputs fit with the training data labels. By analyzing the mathematical relationship between this function's output and each model parameter we can update these parameter values iteravely such that we approach a locally optimal solution set. This process is known as gradient descent, and it is depicted in Figure 6. In [11] Rumelhart showed that networks of neurone-like units could be trained by a process of backpropagation, whereby error gradients with respect to the cost function at the output layer are differentially backpropagated downward to the level of input. In doing so, model parameters at each layer may be updated iteratively in the direction of these gradients to facilitate convergence toward a local optimum.

source: Vasilis Vryniotis

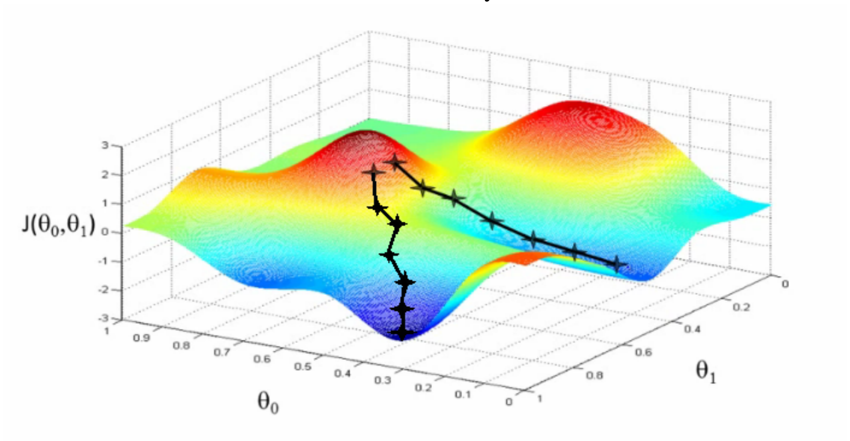


Figure 6: An illustration of the gradient descent algorithm. In each iteration, gradient values are used to determine step updates.

The standard cost function at the output layer is squared error:

$$C = \sum_{samples} (y_{prediction} - y_{label})^2$$

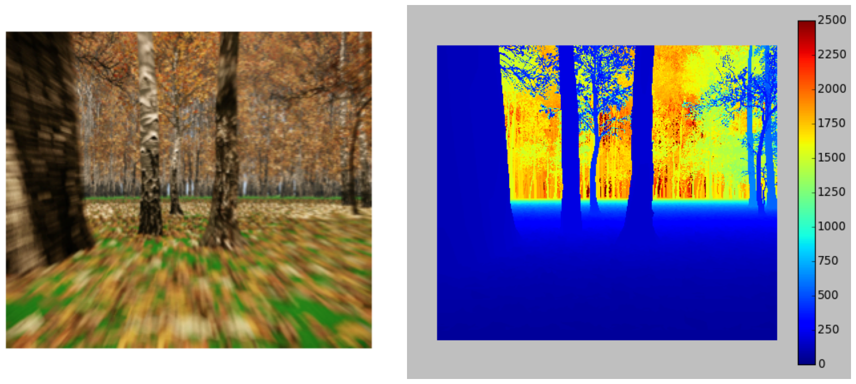
For each weight in our network, we would like to update its value such that we reduce this cost

$$\Delta W \propto -\tfrac{\partial C}{\partial W}$$

and similarly for each bias value. Rumelhart provides a rigorous mathematical derivation of the gradient values for parameters at each layer of the network in the case of sigmoidal activation. Rather than computing these gradients by hand, we can use symbolic differentiation, which is easily accessible thanks to recent software tools. We make use of these gradients, as well as a gradient momentum term, to train our network by gradient descent.

4.3 Computer Graphics

In order to obtain a large number of labeled training examples in a fast and cost-effective manner, we make use of graphical scenes generated by the CryEngine game engine [9]. This software allows for the development of a variety of visual scenes displayed in first person perspective, with ground truth depth map sequences provided for each scene. To build a training set for our model, we generated a database of graphical videos in which an observer is moving forward through a variety of different forest scenes. As the parallax mapping differs between forward and translational motion, we focus on the subcase of forward motion for our predictive task. A key frame from one of our sequences is shown in Figure 7 alongside the corresponding ground truth depth map.



(a) RGB Frame **(b)** Ground Truth

Figure 7: Computer graphics provide a large number of training examples for supervised learning. We can obtain this data essentially for free.

4.4 Model Configuration

We make use of a neural network that takes as inputs the motion feature vectors from a 3x3 pixel window, centered at the pixel for which we would like to predict depth. By using this window, as opposed to a single pixel, we provide the model with a greater context range for inference. Since each vector has 72 elements (9 speeds x 8 directions = 72), our network takes in 648 values at the input layer. It is constructed with 3 hidden layers of 800 units each and a linear output layer. We train via mini-batch gradient descent with a batch size of 5000, making use of the RMSprop learning rule as well as the dropout regularization technique. Our model is implemented from the ground up in Python, making use of the Theano software library. This library enables transparent use of the GPU for fast linear algebra computations.

5 Deep Belief Networks

5.1 Random Initialization: Failure

We have described an algorithm for updating our weights toward an optimal solution, however, the issue still lingers regarding how to initalize these weight values. A standard approach is to use random initialization, by means of a random number generator. The idea is that, regardless of where we begin, we can approach a solution that is pretty good with our supervised backpropogation algorithm. In [7] a sparse initialization scheme is described, whereby the majority of the weights at each layer are set to 0 and a select few are sampled from a normal distribution with 0 mean and standard deviation < 1. In addition, all biases are initially set to 0. This scheme is designed so that hidden units are initially both unsaturated and highly differentiated, helping to ensure a smooth entry into the gradient descent process. The method is considered the state-of-the-art in regards to random initialization approaches for supervised learning. Beginning with this initialization mechanism, however, our model was unable to converge to a meaningful solution in any of a number of trials.

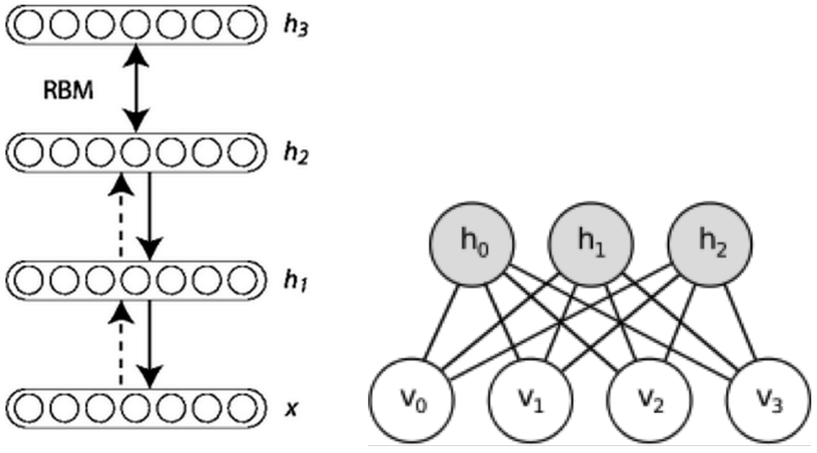
5.2 The Generative Model

As an alternative to random initialization, Geoffrey Hinton showed in [3] that we can do better by initializing our network via a phase of unsupervised learning. To do so, we build a generative probablistic model to describe the distribution of our network graph, and we perform sequential maximum likelihood learning over this model such that the network is initialized with a hierarchy of feature detectors.

$$P(x, h^1, ..., h^l) = (\prod_{k=0}^{l-2} P(h^k, h^{k+1})) * P(h^{l-1}, h^l)$$

At each layer, nodes are trained to extract useful features from the units in the previous layer, thereby uncovering hidden structure from unlabeled training data by a method of belief propagation. Hinton showed that, by learning this generative model in an unsupervised pre-training phase, we can converge to a more optimal solution during supervised backpropagation and avoid getting stuck in local maxima. He described a learning procedure for this model whereby each sequential pair of layers is designated as a Restricted Boltzmann Machine (RBM). A depiction of this architecture is given in Figure 8. The RBM model consists of a hidden layer and a visible layer; each unit of the hidden layer is independent of all others conditioned on the vector of visible unit values, and vice-versa for the visible units. The RBMs are trained sequentially from the layer of input onward by a greedy maximum likelihood learning algorithm known as contrastive divergence. During this process, the first RBM makes use of the input values as its visible data, and each RBM thereafter uses the values passed on through weights learned by the previous RBMs as visible unit data.

source: http://deeplearning.net



(a) DBN **(b)** Restricted Boltzmann Machine

Figure 8: Deep belief netorks are constructed of stacked RBMs

5.3 Contrastive Divergence via Gibbs Sampling

The contrastive divergence algorithm revolves around an energy-based probability model that describes a distribution across the RBM graph. In the standard (Bernoulli-Bernoulli) RBM, units of both the hidden and visible layers retain binary values. A slight modification can be made to this model such that the visible layer units have continuous values; this model is known as a Gaussian-Bernoulli RBM. The energy equations of these two models are as follows:

Standard RBM:

$$E(\vec{v}, \vec{h}) = -\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij} v_i h_j - \sum_{i=1}^{m} b_i v_i - \sum_{j=1}^{n} c_j h_j$$

Gausian-Bernoulli RBM:

$$E(\vec{v}, \vec{h}) = -\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij} h_j \frac{v_i}{\sigma_i} - \sum_{i=1}^{m} \frac{(b_i - v_i)^2}{2\sigma_i^2} - \sum_{j=1}^{n} c_j h_j$$

Here, $(\vec{v}, \vec{h})$ are the visible and hidden unit vectors, W is our weight matrix, and $(\vec{b}, \vec{c})$ are our visible and hidden bias vectors, respectively. The probability distribution over the visible and hidden units is then

$$P(\vec{v}, \vec{h}) = \frac{1}{Z} e^{-E(\vec{v}, \vec{h})}$$

where Z represents the partition function required to normalize this distribution. The goal of maximum likelihood learning is to find the set of values for our model's parameters $(W, \vec{b}, \vec{c})$ such that the likelihood of the observed training data is maximized under this probability model. In each RBM, however, we only observe visible unit data, therefore we must sum over all possible hidden states to generate a new probability equation for our maximum likelihood algorithm. In doing so, we develop a new "free energy", denoted $F(\vec{v})$, that is a function of solely the visible units. The probability can then be written as

$$P(\vec{v}) = \frac{1}{Z} e^{-F(\vec{v})}$$

To retain the previous value for Z, the free energy equation then becomes

$$F(\vec{v}) = -\vec{b}^T \vec{v} - \sum_{j=1}^{n} log(1 + e^{c_j + \vec{W_j} \vec{b}})$$

Now, to perform a maximum likelihood estimation on this new probability distribution, the natural approach is to run a gradient descent on each of the model parameters, updating the values of these parameters iteratively in the direction of the derived gradient values. We can deduce by analysis that the gradients of log probability with respect to each parameter Θ are of the form

$$-\frac{\partial log P(\vec{v})}{\partial \Theta} = \frac{\partial F(\vec{v})}{\partial \Theta} - \sum_{\vec{v_p}} P(\vec{v_p}) \frac{\partial F(\vec{v_p})}{\partial \Theta}$$

where $\vec{v_p}$ are the possible states of the vissible unit vector. The second term thus represents an expected value of the free energy derivative under our probability distribution. Now, unfortunately, obtaining exact values for these gradients is computationally intractable; computing this expected value would require that we sum over all possible configurations of the input. To make this computation tractable, however, we can substitute this second term with a Monte Carlo estimate of the derivative:

$$-\frac{\partial log P(\vec{v})}{\partial \Theta} \approx \frac{\partial F(\vec{v})}{\partial \Theta} - \frac{1}{|N|} \sum_{\vec{v_s} \in N} \frac{\partial F(\vec{v_s})}{\partial \Theta}$$

Here N is our sample set, and $\vec{v_s}$ are the visible unit vector samples. Given the structure of the RBM model, we can obtain these samples very efficiently by the MCMC method known as Gibbs sampling. To perform Gibbs sampling, we alternate between sampling hidden unit values from our distribution conditioned on the visible units and sampling visible unit values from the probability conditioned on the hidden unit values:

$$\vec{h_s} \sim P(\vec{h} | \vec{v} = \vec{v_s}) = Sigmoid(W^T \vec{v_s} + \vec{c})$$

$$\vec{v_s} \sim P(\vec{v} | \vec{h} = \vec{h_s}) = Sigmoid(W \vec{h_s} + \vec{b})$$

An illustration of this process is provided in Figure 9. The Gibbs algorithm guarentees that the K-L divergence between the probability distribution from which we are sampling and our model's equilibrium distribution is reduced with each iteration. Typically, when performing Gibbs sampling, values are initalized randomly, and a burn period is allowed before samples are collected. To perform this training quickly, however, we begin with a training example for the visible unit states. In this sense, contrastive divergence is a "greedy" algorithm, as we assume that the distribution from which we begin is close to $P(\vec{v})$, and thus we run our chain for only a small number of iterations. In the particular case of our DBN, we use the standard CD-1, meaning that a single sample is used for the MC estimate.

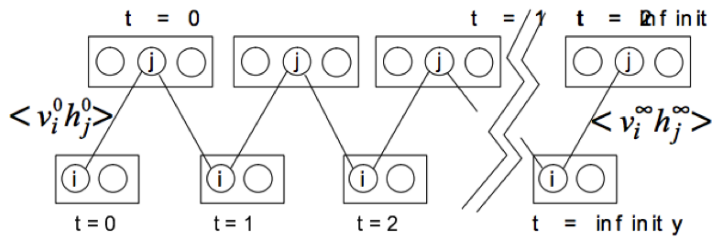


Figure 9: A depiction of the Gibbs sampling algorithm. Source: [3]

5.4 Results

To pre-train the DBN we use a Gaussian-Bernoulli RBM at the first layer, as our input data is continuous, and standard RBMs at each of the other 2 layers. Subsequently, supervised training is performed via backpropogation with our ground truth depth maps from CryEngine. Given an appropriate number of pre-training epochs, we found that our model was able to converge to a solution that performs much better than linear classifiers. To test the performance of our model, we built a test set of sequences from our database drawn from a series of visual scenes significantly distinct from those in our training set. A number of example predictions are shown in Figure 10, displayed next to the predictions of a simple linear regression model trained for the same task. As a metric to evaluate how well we are doing, we use the R^2 coefficient of determination classification score. Figure 11 depicts a plot of test R^2 performance vs. model type.

$$R^2 = 1 - \frac{\sum_i (d_i - \hat{d}_i)^2}{\sum_i (d_i - \bar{d})^2}$$

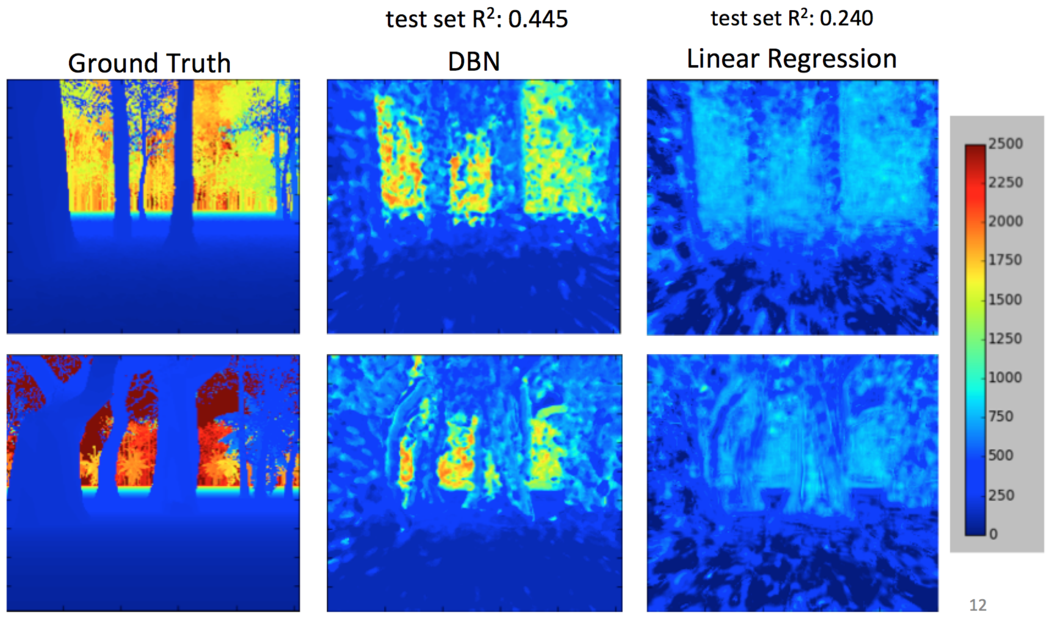


Figure 10: A comparison of results. The DBN does a much better job segmenting the scene into seperate spatial components.

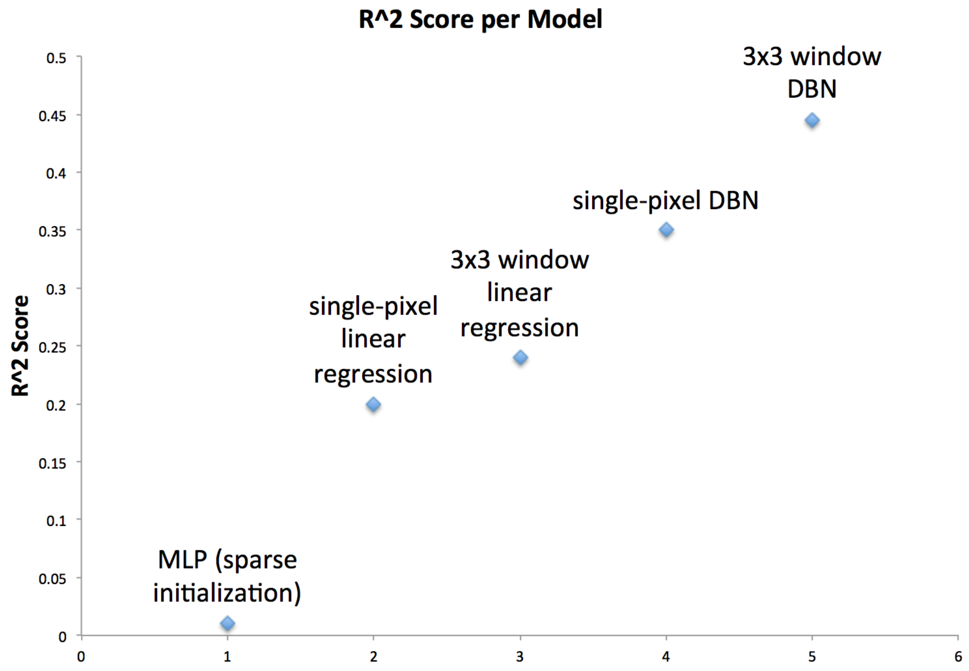


Figure 11: R^2 plotted against model. Scores improve considerably over linear classifiers with our DBN.

5.5 Additional Experiment

As an additional test to evaluate our generative feature-learning process, we conducted one more small experiment. From the results charted in Figure 11, one can see that an R^2 score of 0.240 was obtained by a linear classifier built upon the 648 feature input model. As a manner to test the features generated by our RBM model, we designed a neural net experiment with a single hidden layer of 648 units. This network was pretrained with a Gaussian-Bernoulli RBM via the standard procedure; however, during the supervised phase, in order to isolate the unsupervised feature generation process, we did not backpropogate error gradients down the network. Instead, we trained a simple linear classifier on the features at the output layer, feeding the original feature inputs through our pre-trained network to obtain these new representations. In doing so, we ensure that the hierarchical representations formulated within our network are credible to the unsupervised learning process, rather than to the backpropogation algorithm. Training this model in such a manner, we obtained an R^2 classification score of 0.302, showing significant improvement over the original feature representations. As a control test, we performed one further experiment where the weights between layers are initalized randomly, as opposed to via unsupervised learning. With all other specifics held the same, this model achieved an R^2 performance of 0.252.

Our results provide that, by uncovering hidden structure within the data via an unsupervised learning phase, we are able to improve upon our original feature representations. Furthermore, it could be argued that the quantity of improvement observed in such an experiment constitutes a merit of the baseline feature representations used; certain features may very well facilitate the discovery of hidden structure better than others. In future work, it would be interesting to test whether or not aptitute for simple linear classification tasks is directly correlated with aptitude for this type of unsupervised structural learning.

6 Markov Random Field Smoothing

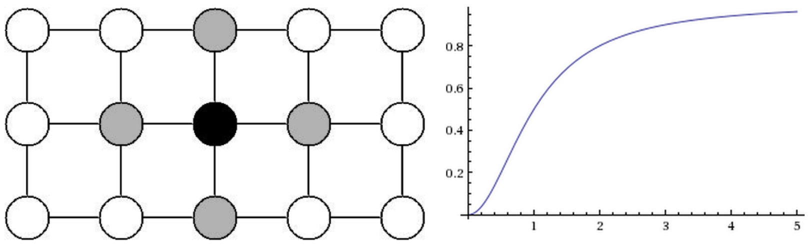
As a mechanism to improve our prediction results, we would like to capitalize upon the power of a receptive field as applied to the task of structure decoding. In order to develop a spatially-coherent perception of the surrounding environment, we propose a Markov random field model to describe the distribution of depth values over all pixels in our prediction map. This MRF is defined by two potential functions:

$$\phi = \sum_i (o_i - d_i)^2$$

$$\psi = \sum_{<i,j>} \frac{(d_i - d_j)^2}{(d_i - d_j)^2 - \alpha_1}$$

Notation:

- $o_i \equiv$ original DBN depth prediction for pixel i
- $\sum_{<i,j>} \equiv$ sum over all neighboring pairs i, j
- $\alpha_1 \equiv$ one of two model smoothing parameters



(a) the MRF graph **(b)** smoothing potential

Figure 12: Neighboring pixels of a given node are shown in (a), and our smothing potential ψ plotted vs. $(d_i$ - $d_j)$ is shown in (b).

Our definition of the neighboring pixel set is depicted in Figure 12a. With the first of these two potential functions, ϕ, we assign a cost to the deviation of a given depth vector from our original DBN prediction map. With the latter of these two functions, ψ, we assign a cost to lack of smoothness in a given depth vector, i.e. to a greater total of differences between neighboring pixel values. Rather than the naive cost function of squared difference, however, we propose a function with a cutoff cost such that we avoid oversmoothing at segment boundaries of the scene. A plot of this function is shown in Figure 12b. This function ensures that we do not assign exponentially larger cost to neighboring pixels at segment boundaries relative to those in smooth regions of the scene. The smoothing parameter α_1 determines where this cutoff is reached. Given these two potential functions, we can define a probability distribution over the depth vector conditioned on our original predictions as

$$P(d|o; \alpha_1, \alpha_2) = \frac{1}{Z} e^{-(\alpha_2 \psi + \phi)}$$

where Z is the partition function that normalizes our distribution and α_2 is a second smoothing parameter that determines the weight placed on smoothness vs. original model consistency. Under this distribution, the depth value of a given pixel is dependent on each of its neighboring nodes, and conditioned on the values of those nodes, it is independent from all other nodes in the graph. The values of α_1 and α_2 are determined by cross-validation, and thereafter we perform a maximum-a-posteriori estimation to determine the depth vector that constitutes the mode of this distribution. In doing so, we were able to achieve noticeable performance improvements over our original depth maps. On average, an R^2 score increase of ~ 0.04 was seen with each test frame, and visualizations appear to be much more coherent and smooth (see Figure 13).

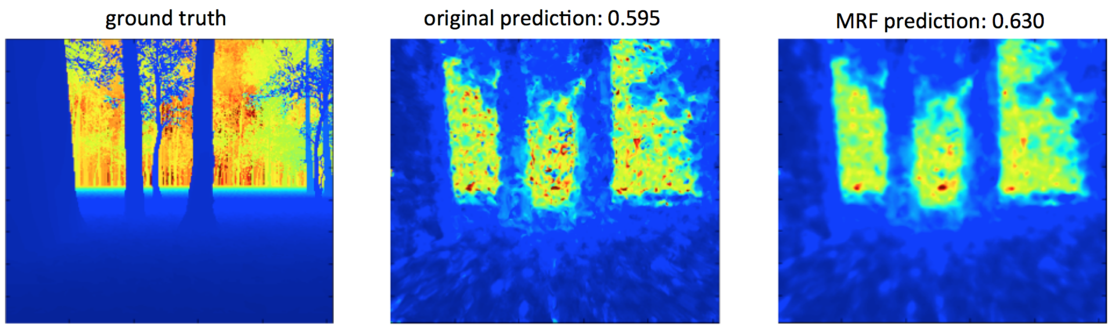
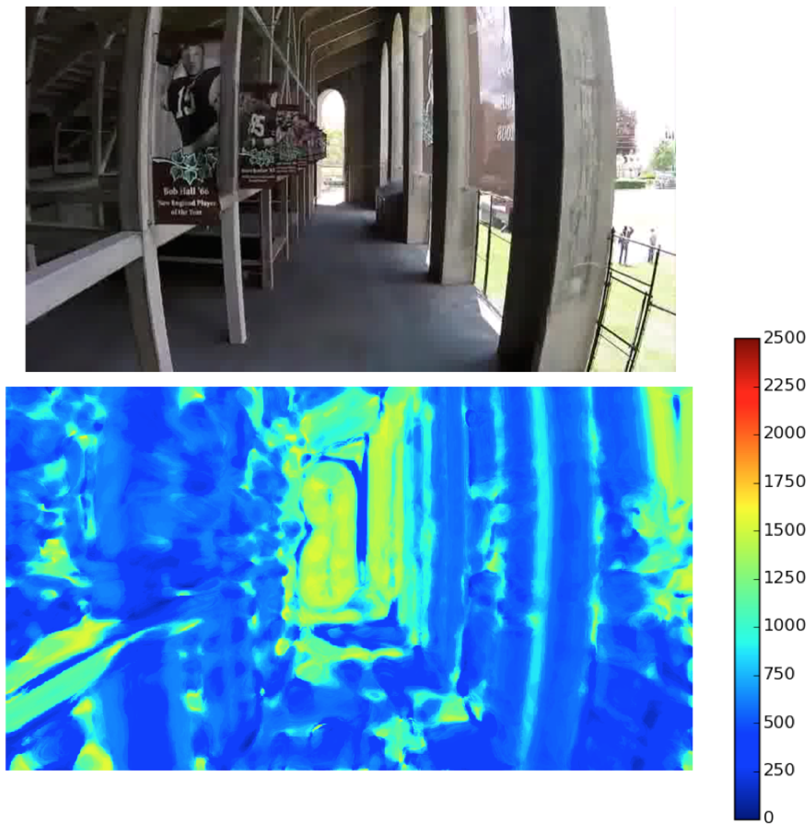
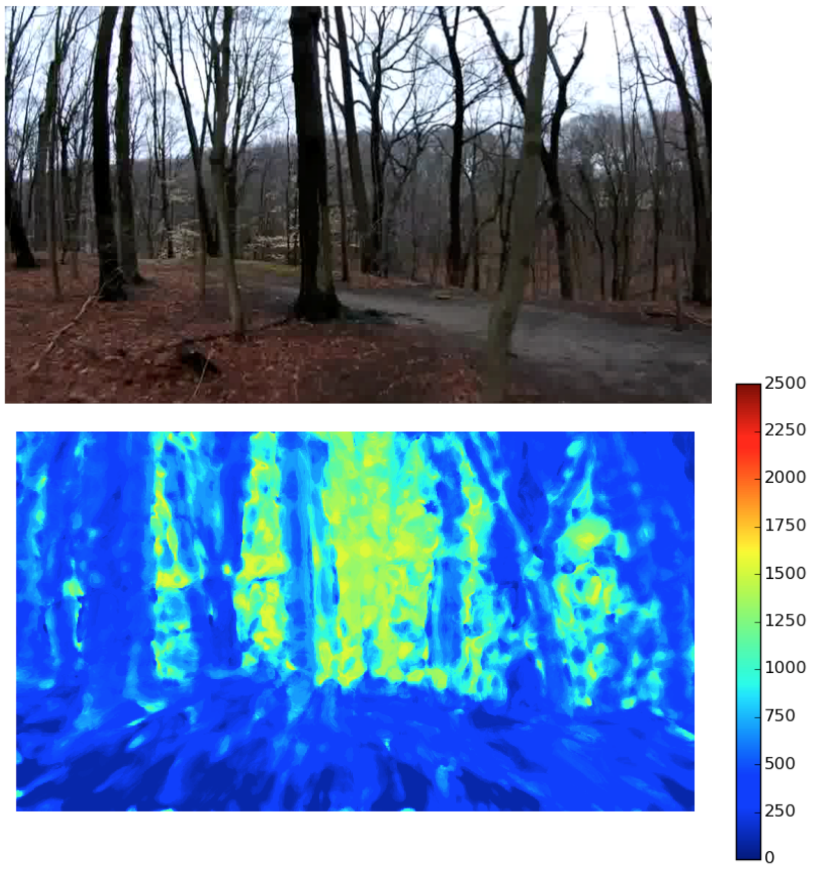


Figure 13: A depiction of MRF results.

7 Drone Test

As a final measure to evaluate the performance of our trained model, we tested its predictions on a set of videos recorded by a Phantom Vision 2 drone. This drone was flown through a series of visual environments, mimicking the forward motion and perspective of the observer from our CryEngine video sequences. Shown below are frames from two visual scenes of this database, corresponding to a forest environment and a university stadium. Along with each frame are the associated depth maps output by our model when provided with the video sequences. Considering that no real-world videos were included in supervised training, our model has decoded a signifant information set regarding the structure of the surrounding environment.





8 Concluding Remarks

We have shown that our feature-based representations of optical flow possess great potential for the task of depth prediction. This potential was not immediately obvious, however, by the results of linear regression classifiers. In [12] it was shown that the HMAX model could mimic the performance of state-of-the-art systems while abiding by biological constraints; to do so, classification results were demonstrated with simple linear classifiers built upon the output features. As a further experiment, it would be interesting to test how well this model can perform when provided with an additional unsupervised learning extension. In [10] it was demonstrated that generative probablistic frameworks are promising as hypothetical models of cortical learning and perception. By developing a generative model that extends from our feature representations, we are able to exploit information regarding the global distribution of these features by means of unsupervised learning. In doing so, we may extract deep hierachical representations of our inputs in an unsupervised manner, and these reperesentations may then be fine-tuned in a subsequent supervised phase according to task.

At a time when artificial vision systems continue to remain poor cousins to their biological counterparts, a great deal of effort is being concerted toward unlocking the cortical computations not captured by these existing systems. Through our experiments, we have demonstrated the aptitude of biologically-inspired vision models for a particular inference task. By combining a neurophysically accurate computational model of sensory processing in the primate cortex with a generative probablistic model that extends from the output responses, we have shown that we are able to perform significantly well at predicting depth field maps for visual media inputs. Furthermore, we have provided a mathematical abstraction to model the role that context plays in the decoding of spatially coherent environment structure, and we have verified that this algorithm improves our decoding performance.

References

[1] P. Baraldi, E. D. Micheli, and S. Uras. Motion and Depth from Optical Flow. *Procedings of the Alvey Vision Conference 1989*, 2:35.1–35.4, 1989.

[2] Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y Ng. Large Scale Distributed Deep Networks. *NIPS 2012: Neural Information Processing Systems*, pages 1–11, 2012.

[3] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504 – 507, 2006.

[4] H Jhuang, T Serre, L Wolf, and T Poggio. A Biologically Inspired System for Action Recognition. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007.

[5] Hyunggoo R Kim, Dora E Angelaki, and Gregory C Deangelis. A Functional Link between MT Neurons and Depth Perception Based on Motion Parallax. *Journal of Neuroscience*, 35(6):2766–2777, 2015.

[6] M. Lehrer, M. V. Srinivasan, S. W. Zhang, and G. a. Horridge. Motion cues provide the bee's visual world with a third dimension, 1988.

[7] James Martens. Deep learning via Hessian-free optimization. *27th International Conference on Machine Learning*, 951:735–742, 2010.

[8] Marcel Mertes, Laura Dittmar, Martin Egelhaaf, and Norbert Boeddeker. Visual motion-sensitive neurons in the bumblebee brain convey information about landmarks during a navigational task. 8(September):1–13, 2014.

[9] Martin Mittring. Finding next gen: CryEngine 2. *ACM SIGGRAPH 2007 courses*, pages 97–121, 2007.

[10] David Reichert, Peggy Series, and Amos Storkey. Hallucinations in Charles Bonnet Syndrome Induced by Homeostasis: a Deep Boltzmann Machine Model. pages 1–9, 2011.

[11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[12] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):411–426, 2007.

[13] Mandyam V. Srinivasan, Michael Poteser, and Karl Kral. Motion detection in insect orientation and navigation. *Vision Research*, 39:2749–2766, 1999.