Competitive Learning for Deep Temporal Networks

Robert Gens
Computer Science and Engineering
University of Washington
Seattle, WA 98195
rcg@cs.washington.edu

Pedro Domingos
Computer Science and Engineering
University of Washington
Seattle, WA 98195
pedrod@cs.washington.edu

Abstract

We propose the use of competitive learning in deep networks for understanding sequential data. Hierarchies of competitive learning algorithms have been found in the brain [1] and their use in deep vision networks has been validated [2]. The algorithm is simple to comprehend and yet provides fast, sparse learning. To understand temporal patterns we use the depth of the network and delay blocks to encode time. The delayed feedback from higher layers provides meaningful predictions to lower layers. We evaluate a multi-factor network design by using it to predict frames in movies it has never seen before. At this task our system outperforms the prediction of the Recurrent Temporal Restricted Boltzmann Machine [3] on novel frame changes.

1 Introduction

Deep networks can encode complex functions in less space than shallow functional equivalents [4]. The two most popular ways to build a deep network are Restricted Boltzmann Machines and Autoencoders. Both of these ideas date back to investigations into neural learning in the 1980's [5]. Competitive Learning was also proposed at this time, but it has not received very much attention in contemporary neural networks [6]. However, there may be good reason to consider their use. Recent research has found competitive learning algorithms in critical information processing regions of the brain [1]. There is also some evidence that these algorithms could be responsible for learning the major building blocks of the fast feedforward visual pathways in primates [2]. This validation of competitive learning as building block of brain networks makes it a promising candidate for deep networks research.

We expand upon the work of competitive learning by creating a large heterogeneous hierarchy to understand video. Much like the visual cortex we divide our architecture into a motion subnetwork and a shape subnetwork [7]. Our network is different from those of Serre et al. [8] and Jhuang et al. [9] in that we incorporate both shape and motion into one network and learn weights for both subnetworks at the same time. We seek to design a network that will allow competitive learning units to make predictions. By identifying the key factors that make up the input in our subnetworks, we can then learn how these factors combine to predict the current state. This will require that our network represents time.

The recent Recurrent Temporal RBM and Temporal RBM show the importance of delayed signals and hidden-to-hidden connections in networks that process sequential information [3, 10]. As we stack layers in a deep network, we might want to make use of the depth of the network to encode time, so we will explore the use of feedback to make predictions. That is, a unit in a higher layer with a larger receptive field may be able to prime a lower unit for a future stimulus.

In this paper we develop a framework with which we can connect multiple layers of competitive learning units in multiple subnetworks. We then create a vision network composed of shape and

motion sections with delayed signals and feedback. Finally we test our network against natural video and synthetic motion. Our results show that this is a promising approach to deep temporal networks.

The architecture of our system provides insight into designing networks that process other sequential data such as speech. The analogs to shape and motion for auditory processing are timbre and melody. A subnetwork that recognizes combinations of static pitches communicating with a subnetwork that measures changes in pitch would be able to anticipate phonemes as they are spoken. On a higher level, both visual and auditory processing are require a hierarchical grammar, which could be achieved with a mixture of clustering, delays, and sequence hashing [1]. The top-down feedback mechanism we demonstrate may be useful for other predictive systems that operate on hierarchical data.

2 Competitive Learning Network for Video

We propose a heterogenous deep architecture that can encode time through depth and delayed connections. In traditional neural network architectures we rely on the learning to discover the factors of the input. We are inspired by the organization in the brain to divide our network into two predetermined pathways: shape and motion. We use competitive learning units to simplify the nodes of the network so we can investigate different architectures. By establishing this division we can not only design subnetworks that best accommodate their functions but also allow devise the ways in which subnetworks communicate with each other.

2.1 Competitive learning units

The basic learning unit we use is the prototypical competitive learner from [6]. A unit receives a fixed number of n_i inputs (vector $\vec{i}$) and over time learns a fixed number of n_j clusters of those inputs. The dimension of each cluster matches that of the inputs. At each learning step, the inputs are compared with each cluster using a dot product. The cluster that best matches the input 'wins' (see Figure 1) and is altered so that it better resembles the input.

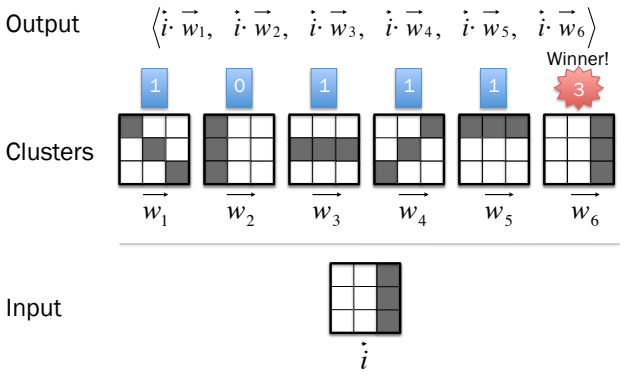

Figure 1: This diagrams the process of competitive learning in a single unit. There are $n_j = 6$ clusters and the input $\vec{i}$ is a $3x3$ binary vector. For each round of learning the input is compared with every cluster using a dot product. The cluster with the largest product wins the round and its weights are changed according to the equation below. The output of a competitive learning unit is a vector composed of the products for each cluster (shown in colored boxes).

The change in the weights of a cluster is

$$\Delta w_{ij} = \begin{cases} 0 & \text{if cluster } j \text{ loses} \\ \alpha \frac{c_{ik}}{\| \vec{i} \|} - \alpha w_{ij} & \text{if cluster } j \text{ wins} \end{cases}$$

where w_{ij} is the weight connecting input i to cluster j, α is the learning rate, and c_{ik} is value of input i in example k. We normalize the weights by the magnitude of the input vector as opposed to

the number of inputs as in the original algorithm. Though only one cluster in a unit can 'win' at a time, the output of the unit is a dimension-J vector composed of the dot product of the input with each of n_j clusters

$$\vec{o} = \langle \vec{i} \cdot \vec{w}_1, \vec{i} \cdot \vec{w}_2, ..., \vec{i} \cdot \vec{w}_{n_j} \rangle$$

where $\vec{w}_j$ is the vector of weights for all n_i inputs in cluster j. We sample the initial weights from a uniform distribution on the unit interval $(0, 1)$. If a competitive learning unit receives input from another unit, it is the entire output vector that is incorporated into the unit's inputs for learning. Over a relatively small number of training examples proportionate to the learning rate and the number of clusters, the competitive learning unit learns a sparse alphabet of symbols that regularly occur in the input. We use this functionality to learn in both the shape and motion subnetworks. In the former case, the unit learns a set of oriented edges in space. In the later, it learns edges in space-time. Time is incorporated using delay blocks.

2.2 Delay blocks

In order to measure changes over time, we simulate the signal delays found in biological neural networks. A delay block receives only one signal x and outputs the d most recent values of that signal. Thus the output of a delay block is the vector of $\langle x(t), x(t-1), ..., x(t-(d-1)) \rangle$.

2.3 Network topology

We propose an initial design for a low-level multi-factor vision network in Figure 2. The shape and motion subnetworks are each comprised of k layers. Lower layers within a subnetwork feed to higher layers. The image from the video only feeds directly to the lowest shape units. Motion units take their input from delay blocks of the maximum activation of a small neighborhood of shape units. For example, a motion unit that looks at a $2x2$ grid of delayed (assume $d = 3$) shape units at the current time t will receive three versions of the maximum element of the output vector from each of the four shape units, one for each of $\{t, t-1, t-2\}$.

In a two-layer network, the output vector of a second layer motion unit becomes part of the input to a first layer shape unit. This feedback is delayed by a single time step. The input vector of a layer one shape unit that watches n pixels $r_1 ... r_n$ at time t is

$$\vec{i}_{S1}(t) = \langle r_1(t), ..., r_n(t), o_{M2}(t-1, 1), ..., o_{M2}(t-1, n_j) \rangle$$

where $o_{M2}(t-1, j)$ refers to the j-th element of the output vector of a layer two motion unit with n_j clusters at time $t - 1$. Using this notation the input to a layer one motion unit at time t is

$$\vec{i}_{M1}(t) = \langle o_{S1}(t, j'), o_{S1}(t-1, j'), o_{S1}(t-2, j'), ... \rangle$$

where j' denotes the winning cluster at that time and the trailing ellipsis signifies that the unit can receive from multiple $S1$ units. This shows that this motion unit receives three copies of the winning output of each shape unit in its receptive field for a delay block of $d = 3$. The input to a second layer shape unit is

$$\vec{i}_{S2}(t) = \langle o_{S1}(t, 1), ..., o_{S1}(t, n_j), ... \rangle$$

where $S1$ is one of the layer one shape units that provides input. The layer two motion unit receives input the output vectors from layer one motion units as well as the delayed winning outputs of layer two shape units:

$$\vec{i}_{M2}(t) = \langle o_{M1}(t, 1), ..., o_{M1}(t, n_j), ..., o_{S2}(t, j'), o_{S2}(t-1, j'), o_{S2}(t-2, j') ... \rangle$$

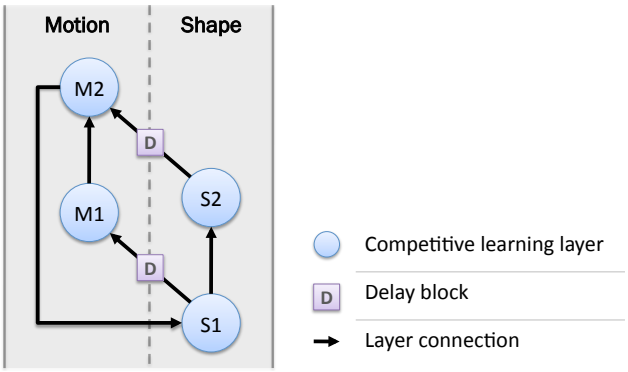

Figure 2: A two-layer network for watching video consisting of two subsections: motion and shape. The delayed outputs of shape layers are fed to motion layers. With this design we test the ability of the second motion layer to help make predictions in the first shape layer.

2.4 Generating a frame

We visualize the activity of our network by projecting the weights of the winning cluster back to the lower units that provide the corresponding input. Units that feed to multiple higher units take an average of the back projected signals. The lowest shape layer votes on values of the pixels of the patch it receives as input, but we multiply the weights by a constant q to roughly scale from their normalized values. The backwards signals of this frame generation are not part of the learning process. The predicted value of a pixel is

$$p(x,y) = \frac{\sum_{\mathbf{s1}(\mathbf{x},\mathbf{y})} q w_{j'xy}}{n_{s1}}$$

where $\mathbf{s1}(\mathbf{x},\mathbf{y})$ is the set of all units in the first shape layer that connect directly to pixel (x,y), n_{s1} is the size of this set of first layer shape units, and $w_{j'xy}$ is the weight connecting pixel (x,y) to the winning cluster j' for a given unit.

3 Experiments

3.1 Topographic layout

The size of each layer and the inputs of any unit in that layer are detailed in Table 1 and illustrated in Figure 3. This is by no means an optimal design, but we chose it because it performed better at prediction tasks than other preliminary network designs. For our experiments we set the number of layers $k = 2$.

Table 1: Size and quantity of units in the vision network

Layer	Quantity	Inputs	Receptive Field	Total Weights
S1	36	2x2 video image, at least one M2	2x2	40,320
S2	9	2x2 S1	4x4	20,736
M1	9	2x2 S1 (delayed)	4x4	82,944
M2	4	2x2 S2 (delayed), 2x2 M1	8x8	46,080
			Total:	190,080

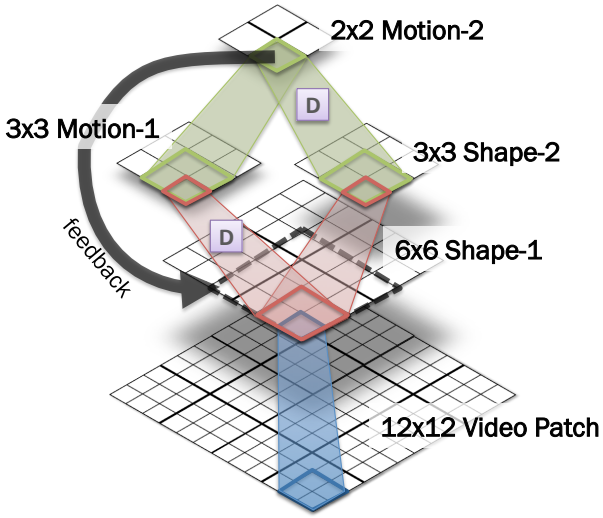

Figure 3: This diagram traces the connections from four pixels on the video up to the second motion layer. The feedback from the second layer of motion is shown to provide input to a 4x4 region of the units in the first shape layer. These regions overlap by two pixels, so some units in this layer receive feedback from one, two, or four second layer units.

3.2 Comparing with RTRBM

The RTRBM system we test against is composed of a visible and a hidden layer with corresponding weight matrices for visible-to-hidden (W_{VH}) and hidden-to-hidden (W_{HH}) connections. These matrices are learned at every time step using Contrastive Divergence[11]. In the future we would like to test our approach against the multi-layer Temporal Restricted Boltzmann Machines produced by the same authors [10]. For the purposes of this workshop we used the RTRBM because it produces more realistic samples than the TRBM.

For comparison purposes we set the number of hidden units in the RTRBM so that the total number of weights in each network are approximately equal. We calculate the size of each layer of our vision network in in Table 1. An RTRBM with 12^2 visible nodes (defined by experiment) and 370 hidden nodes has slightly more weights than our network.[1]

3.3 Prediction in natural video

We used two popular off-the-shelf movies[2] to train and test for this task. Only one movie was used for learning each network, and the other movie was used to test prediction. For every frame of the movie, the pixels[3] of a *12x12* region were fed to the visible units of the RTRBM or the lowest layer of our network. The same region of pixels was shown to both networks. A small patch of the video will suffer the most from the *aperture problem* and we would like to test our prediction with this challenge

In the factored network, each unit in a layer would competitively learn[4] from its input. Learning the RTRBM was performed as described in [3], but the network was made to watch the movie only once in sequence as opposed to redundant random subsequences.

To test prediction, we disabled the learning mechanism and had each network watch the second movie. Every 100 frames, however, we would sever the connection from the visible units. Upon

[1] $12^2 * 370 + 370^2 = 190,180 \approx 190,080$

[2] *Lost in Translation (2003)* for training and *The Ten Commandments (1956)* for testing

[3] The value of a pixel ranges from 0 to 1 in 256 steps.

[4] In our experiments we set $d = 4$, $\alpha = .005$, $q = .1$, and $n_j = 24$.

watching this blank frame, we asked both networks to generate the current frame of the movie based on its internal state. It is trivial to sample the visible nodes in the RTRBM given the state of the hidden nodes and the learned visible-to-hidden connections: $V = W_{VH} \cdot H$.

Reconstructing the video frame from the higher levels of the competitive network is described above in section 2.4. Before calculating the error, we subtracted the mean from both the predicted and actual frame. We summed the squares of the differences between corresponding pixels in the predicted and actual frames. This test was performed 1,000 times, spanning the first 100,000 frames (roughly an hour) of the testing movie.

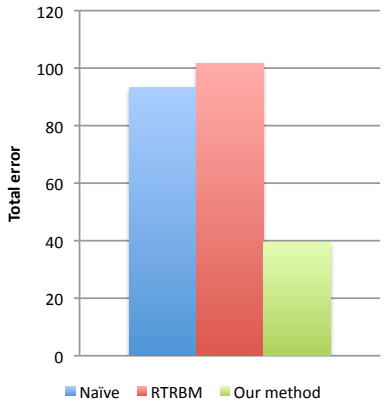

Figure 4: Total error for each method on the top 5% of frames by size of change in the video.

We also measured the squared error of the naive prediction that the missing frame would be the same as the previous frame. For any of the 1,000 trial frames the probability that the frame changes significantly from the one before is very low. For frames where there is a large change, our method does better than the RTRBM and naive predictors as shown in Figure 4. The majority of frames have little change, and for these both our system and the RTRBM perform worse than the naive predictor. Our network performs this way because there is no motion signal to guide the prediction; it could be fixed with recurrent links on the lowest shape layer.

This outcome of this experiment shows that our network makes better predictions than the naive and RTRBM approaches when there is movement. This should not be a surprise considering the network design.

3.4 Prediction of oriented moving gratings

We created synthetic video to test the ability of the network to predict successive frames with an exhaustive set of motions. We used an oriented cosine pattern generated by the equations

$$\begin{aligned} val &= \cos\left(\frac{x * \cos(\theta) + y * \sin(\theta)}{s} + t + r\right) \\ \theta &= 2\pi \frac{\text{current angle}}{\text{number of angles}} \\ s &= 2\pi S \\ t &= 2\pi \frac{\text{frame number}}{T} \end{aligned}$$

where (x, y) are the coordinates in the $12x12$ patch, S is the wavelength in pixels, T is the temporal period in frames, and r is the offset. The network learned from the same movie as in the previous section. For testing purposes we disable learning and show the network eighteen frames of the synthetic motion but omit the last frame. We then compare the internal state of units in the network

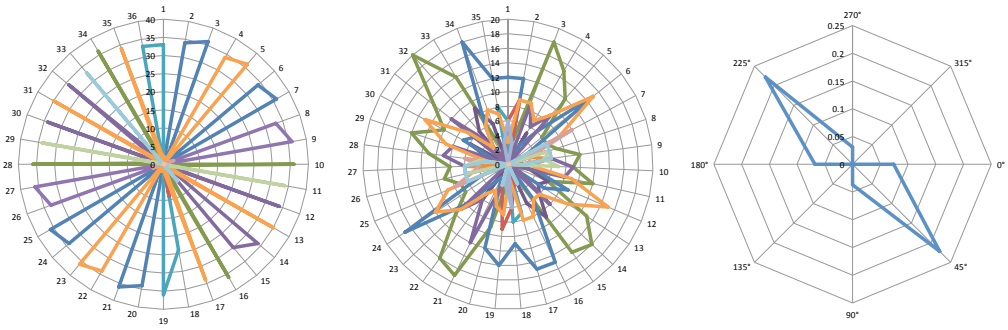

Figure 5: Left: the response of a layer two motion unit to different angles of static edges. The different colors correspond to the clusters inside the unit. The radius is a measure of how active each cluster was out of a total of 36 frames. Middle: the response of the same layer two motion unit to different angles of motion. Right: the radius of this plot shows the fraction of total frames predicted correctly (tested on different motion parameters over all offsets) for a layer one shape unit that receives the strongest feedback from a layer two motion unit in the upper-left corner of the frame.

at that blank frame with the state of the network had the final frame been shown. As the internal state of the network is a deterministic process, we are able to reproduce identical sequences of internal states given a video that is longer than the sum of linked delays. For any set of parameters (θ, S, T) we also vary the frame at which we test prediction through the r parameter. The results of the test for predicting a low level shape unit that receives feedback from four high level motion units are shown in Figure 5.

4 Conclusion

We have shown that competitive learning units can be stacked into deep networks much like RBMs and Auto-encoders to learn from input unsupervised. We motivated the division of the deep network into functional units and tested the ability of feedback connections to aid in making predictions. The simple learning units allowed us to effortlessly add delay blocks and feedback between layers so we could study their effects on performance.

Competitive learning is a promising approach to deep temporal networks that we would like to explore more completely. We would like to adapt our competitive learning network to become generative by using model-based clustering. The are also many other factors of vision networks that we could potentially design around, such as color, texture, and motion derivatives. Despite the quick computation of competitive learners, our first vision network is rather small and relatively shallow; we will work towards networks that make the most of their depth.

References

[1] R. Granger. Engines of the brain: The computational instruction set of human cognition. *AI Magazine*, 27(2):15, 2006.

[2] T. Masquelier, T. Serre, S. Thorpe, and T. Poggio. Learning complex cell invariance from natural videos: A plausibility proof. Technical report, Massachusetts Institute of Technology, Cambridge, MA, 2007.

[3] I. Sutskever, G. Hinton, and G. Taylor. The Recurrent Temporal Restricted Boltzmann Machine. In *NIPS*, volume 21, page 2008, 2008.

[4] Y. Bengio. Learning deep architectures for AI. *Foundations & Trends in Mach. Learn., to appear*, 2009.

[5] D.E. Rumelhart and J.L. McClelland. Parallel distributed processing, explotation in the microstructure of cognition-Vol. 1: Foundations. 1987.

[6] D.E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9(1):75–112, 1985.

[7] D.J. Felleman and D.C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1):1, 1991.

[8] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, page 994. Citeseer, 2005.

[9] H. Jhuang, T. Serre, L. Wolf, and T. Poggio. A biologically inspired system for action recognition. In *Proc. IEEE Int. Conf. Computer Vision*, volume 1, page 5. Citeseer, 2007.

[10] I. Sutskever and G.E. Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *Proceeding of the Eleventh International Conference on Artificial Intelligence and Statistics*, pages 544–551. Citeseer, 2007.

[11] G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.