Pattern Recognition Letters 37 (2014) 115–123

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Hierarchical spatiotemporal feature extraction using recurrent online clustering

S.R. Young *, A. Davis, A. Mishtal, I. Arel

Machine Intelligence Lab, Department of Electrical Engineering and Computer Science, The University of Tennessee, United States

ARTICLE INFO

Article history:
Available online 1 August 2013

Keywords:
Deep machine learning
Online clustering
Recurrent clustering
Unsupervised feature extraction
Spatiotemporal signals
Pattern recognition

ABSTRACT

Deep machine learning offers a comprehensive framework for extracting meaningful features from complex observations in an unsupervised manner. The majority of deep learning architectures described in the literature primarily focus on extracting spatial features. However, in real-world settings, capturing temporal dependencies in observations is critical for accurate inference. This paper introduces an enhancement to DeSTIN – a compositional deep learning architecture in which each layer consists of multiple instantiations of a common node – that learns to represent spatiotemporal patterns in data based on a novel recurrent clustering algorithm. Contrary to mainstream deep architectures, such as deep belief networks where layer-by-layer training is assumed, each of the nodes in the proposed architecture is trained independently and in parallel. Moreover, top-down and bottom-up information flows facilitate rich feature formation. A semi-supervised setting is demonstrated achieving state-of-the-art results on the MNIST classification benchmarks. A GPU implementation is discussed further accentuating the scalability properties of the proposed framework.

1. Introduction

The mainstream approach for addressing high-dimensional data is to employ a feature extraction process charged with mapping the data to a lower-dimensional space, while retaining as much information as possible in order to accurately process (e.g. classify) the data. As a result, it is argued that the intelligence behind most pattern recognition engines rests on the human-engineered feature extraction process used. At times, such feature extraction can be challenging to construct and highly application-dependent. Moreover, if incomplete or erroneous features are produced, the classification process is inherently limited in its performance.

Recent research into the mammalian brain has inspired new ideas for designing systems that represent information. This research claims that the neocortex, which is responsible for many cognitive abilities, does not perform explicit pre-processing of sensory signals. Instead, these signals are propagated through a complex hierarchy of modules (Lee and Mumford, 2003), that learn to represent the information based on the regularities that exist in the signals over time. This finding led to the emergence of the field of deep machine learning (DML) (Arel et al., 2010), which focuses on extracting data efficiently in an unsupervised manner.

While the spatial information in real-life data is important, the temporal information is also very important. When humans observe a sequence of patterns, they can gain an understanding of what is occurring that could not be inferred from a single pattern. We often infer a meaning from events observed in short time periods (Wallis and Bulthoff, 1999; Wallis and Rolls, 1997). Thus, modeling the temporal regularities that exist in a sequence of patterns is vital to effective information representation. Therefore, capturing the spatiotemporal dependencies should be viewed as a principal objective for deep learning systems.

Despite the importance of representing temporal information, there has been little work published on deep learning architectures that naturally integrate the spatial and temporal components that exist in many spatial sensor datasets such as microphone arrays, video, or traffic monitoring. This paper introduces an enhancement to DeSTIN – a compositional deep learning architecture in which each layer consists of multiple instantiations of a common node. Spatiotemporal dependencies are naturally captured by employing a novel recurrent clustering algorithm that serves as an unsupervised learning mechanism embedded in each node of the DeSTIN hierarchy. Since each node operates independently and in parallel to all other nodes, the proposed framework offers exceptional scalability attributes, particularly in the context of graphical processor unit (GPU) implementation. Modern GPUs are highly parallel programmable processors that have a peak arithmetic and memory

* Corresponding author. Tel.: +1 (865) 226 9746.
E-mail addresses: syoung22@utk.edu (S.R. Young), adavis72@utk.edu (A. Davis), amishtal@utk.edu (A. Mishtal), itamar@ieee.org (I. Arel).
URL: http://mil.engr.utk.edu (S.R. Young).

0167-8655/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.patrec.2013.07.013

bandwidth much greater than any CPU (Owens et al., 2008) and they do so at a very affordable price. Thus, the ability to map DeSTIN to a GPU implementation is very important in order to tackle large problems that use typical image or video resolutions, as opposed to the small resolutions used in many of the current datasets being used in DML research such as the MNIST (Lecun and Cortes, 1998) and CIFAR-10 (Krizhevsky et al., 2009) datasets.

Several enhancements to previous realizations of DeSTIN are described. Previously, each node was performing several dissimilar tasks resulting from modeling of the system dynamics. The memory and/or computation requirements of these methods dwarfed the resource requirements of the clustering algorithm that resides at the core of each node. Even without the resource requirements imposed by these methods, the fact that there are many dissimilar operations required makes mapping the architecture to a parallel computing platform, such as a GPU, very challenging. The enhancements introduced here aim to include this functionality into the core clustering algorithm in order to significantly relax the resource requirements thus greatly enhancing the scalability attributes of the system. In addition to shifting more functionality to each node, changes were made to the clustering algorithm in order to eliminate unnecessary computation and facilitate the formation of richer features.

The rest of the paper is structured as follows. Section 2 provides an overview of deep machine learning and the DeSTIN architecture, which serves as basis for the proposed scheme. Section 3 describes the enhanced DeSTIN node and its utilization within the hierarchical architecture. In Section 4 a GPU implementation is described, highlighting the scalability attributes of the proposed scheme. Section 5 provides simulation results on a set of benchmark tasks, while in Section 6 the conclusions are drawn.

2. Deep machine learning and the DeSTIN architecture

2.1. Overview of deep learning architectures

Deep belief networks (DBN) (Arel et al., 2010; Hinton et al., 2006) and Convolutional Neural Networks (CNN) (Arel et al., 2010; Lee et al., 2009) are two of the mainstream DML paradigms that have been successfully demonstrated in addressing pattern recognition problems in high dimensional data (e.g. images). CNNs are a family of multi-layer neural networks particularly designed for use on data structured on a two-dimensional grid, such as images and videos. CNNs were proposed as a deep learning framework that is motivated by minimal data preprocessing requirements. In CNNs, small portions of the image (dubbed a local receptive field) are treated as inputs to the lowest layer of the hierarchical structure. Information generally propagates through the different layers of the network, and at each layer digital filtering is applied in order to obtain salient features of the data observed. This method provides a level of invariance to shift, scale and rotation as the local receptive field allows the neuron or processing unit access to elementary features such as oriented edges or corners. One of the seminal papers on the topic (LeCun et al., 1998) describes an application of CNNs to the problem of handwriting analysis. Essentially, the input image is convolved with a set of N small filters whose coefficients are either trained or pre-determined using some criteria. Thus, the first (or lowest) layer of the network consists of "feature maps" which are the result of the convolution processes, with an additive bias and possibly a compression or normalization of the features. This is followed by a sub-sampling (typically a 2×2 averaging operation) which further reduces the dimensionality and offers some robustness to spatial shifts. The sub-sampled feature map then receives a weighting and trainable bias and finally propagates through an activation function. Some variants of this exist with as few as one map per layer (Chen et al., 2006) or summations of multiple maps (LeCun et al., 1998).

DBNs, initially introduced by Hinton et al. (2006), are probabilistic generative models which stand in contrast to the discriminative nature of traditional neural nets. Generative models provide a joint probability distribution over observable data and labels, facilitating the estimation of both $P(Obs.|Label)$ as well as $P(Label|Obs.)$, while discriminative models are limited to estimating the latter, $P(Label|Obs.)$. DBNs address problems encountered when traditionally applying back-propagation to neural networks, namely: (1) necessity of a substantial labeled data set for training, (2) slow learning (i.e. convergence) times, and (3) inadequate parameter selection techniques that lead to poor local optima. Although there has been some success in training DBNs in a completely unsupervised manner (Lee et al., 2011), most current work also performs supervised training after this unsupervised step. DBNs do not explicitly address learning of temporal relationships between observables, though there has been recent work in stacking temporal RBMs (Sutskever et al., 2007) or generalizations of these, dubbed temporal convolution machines (Lockett et al., 2009), for learning sequences. The application of such sequence learners to audio based problems, where DBNs have made recent headway, offer an avenue for exciting future research. CNNs have recently been trained with a temporal coherence objective to leverage the frame to frame coherence found in videos (Mobahi et al., 2009), though this objective need not be specific to CNNs.

Recent literature treats pure multi-layer perceptron (MLP) neural networks with more than two hidden layers as deep learning architectures. Although one can argue that technically this is a correct assertion, the mere fact that a learning system hosts multiple layers is insufficient to be considered as a deep learning architecture. The latter should also encompass the idea of a hierarchy of abstraction, whereby as one ascends the hierarchy more abstract notions are formed. This is not directly attainable in a simple MLP consisting of a large number of hidden layers. Moreover, there is no temporal information representation in such schemes.

The concept of partitioning large data structures into smaller, more manageable units, and revealing dependencies that may or may not exist between such units, has proven to be a very promising direction of research. However, there remains a need for an architecture that can represent temporal information with the same ease in which spatial structure is discovered. Additionally, some key constraints are imposed on the learning schemes driving these architectures, namely the need for layer-by-layer training and often times pre-training, which limit their scalability and accuracy.

2.2. The DeSTIN architecture

The focus of the work presented here is the Deep Spatiotemporal Inference Network (DeSTIN) architecture, first introduced in Arel et al. (2009). DeSTIN consists of multiple instantiations of an identical functional unit called a node which learns through a completely unsupervised learning process, unlike most of the previously discussed DML methods that rely on labeled information. These nodes are arranged in layers and each node is assigned children nodes from the layer below and a parent node from the layer above as shown in Fig. 1. Nodes at the lowest layer receive as input a subset of the raw sensory data while nodes at all other layers receive the belief states, or outputs, from their children nodes as input. This subset, or receptive field, will be unique for each node. The receptive fields are disjoint in the work presented here, but could be overlapping depending on the application. Each node attempts to capture the salient spatiotemporal regularities contained in its input and continuously update a belief state meant

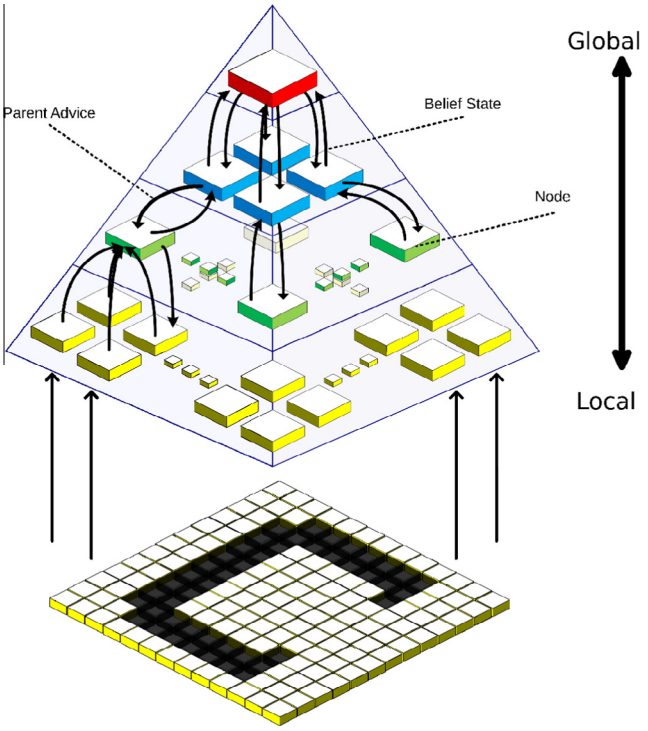

Fig. 1. Typical node configuration and signal flow in a DeSTIN hierarchical architecture.

to characterize the input and the sequences thereof. The belief state (or belief) is a probability vector that indicates the probability of each possible state given the information we know about the system. The beliefs formed throughout the architecture can then be used as rich features for a classifier that can be trained using supervised learning. Beliefs extracted from the lower layers will characterize local features and beliefs from higher layers will characterize global features. Thus, DeSTIN can be viewed as an unsupervised feature extraction engine that forms features from data based on regularities it observes. This stands in contrast to the user-engineered features based approach, which relies on previous knowledge of the problem at hand. The unsupervised nature of DeSTIN renders it much simpler to train than other DML architectures (Karnowski et al., 2012). It also suggests that the features generated by DeSTIN need not be unique in the sense that they converge to singular values in order to be meaningful to a given application.

As outlined above, the core function of each node is to form a belief state that characterizes the inputs observed. This belief state is expressed through the following conditional probability function

$$b_t(s_t|a) = \frac{\Pr(o|s_t)\left\{\sum_{s_t\in S}\Pr(s_t|s_{t-1},a_{t-1})b(s_{t-1})\right\}}{\sum_{s'_t\in S}\left\{\Pr(o|s'_t)\sum_{s''_t\in S}\Pr(s''_t|s_{t-1},a)b(s_{t-1})\right\}} \quad (1)$$

which serves as an update equation, as the system transitions from one time step to the next. This function maps the input o from the layer below, belief state b (which is a function of the system state s), and parent's belief state (i.e. advice) a from the layer above to an updated belief state $b_t(s_t)$. The denominator of this equation is a normalization factor. This equation should be viewed as two parts: (1) a posterior over the observations, $\Pr(o|s_t)$, that is modulated by a (2) construct that reflects the system dynamics, $\sum_{s_t\in S}\Pr(s_t|s_{t-1},a_{t-1})b(s_{t-1})$. These building blocks of the architecture are the pieces of information which must be learned from the data.

In this paper, several changes to previous implementations of DeSTIN are presented. Previously, each node was performing several dissimilar tasks resulting from modeling of the system dynamics. The memory and/or computation requirements of these methods dwarfed the resource requirements of the clustering algorithm that is supposed to be the core of each DeSTIN node. Even without the resource requirements imposed by these methods, the fact that there are many dissimilar operations required renders mapping the architecture to a GPU platform rather challenging. The changes presented here aim to include this functionality into the core clustering algorithm in order to lessen these resource requirements and make the process of implementing DeSTIN in GPU platforms a more attainable task. In addition to shifting more of the functionality of DeSTIN into the clustering algorithm at the individual nodes, modifications were made to the clustering algorithm itself in order to eliminate unnecessary computation and help generate richer features.

2.3. Incremental clustering

Since DeSTIN was designed as a system that is scalable using simple hardware, an incremental clustering algorithm is employed for learning $\Pr(o|s_t)$ in order to minimize memory requirements. Young et al. (2010) introduced the winner-take-all incremental clustering algorithm used as the core of each DeSTIN node. This algorithm finds centroids which are represented by a mean μ and variance σ^2 in each dimension. Based on the centroids formed and their relationship to the input vector o, $\Pr(o|s_t)$ is obtained where s_t corresponds to a particular centroid in the set of centroids. A key idea of this algorithm was the introduction of the starvation trace which addressed centroids that happen to be initialized far from any dense regions of the observation space and thus would never be selected for update. The starvation trace is used to shrink the apparent distance of a starved centroid to all input vectors until it is selected for update. A starvation trace value, ψ, is maintained for each centroid and is decayed by a constant, γ, each time that centroid is not updated and increases once the centroid is selected, as reflected by

$$\psi_c = \gamma\psi_c + (1-\gamma)\mathbb{1}_{x=c} \quad (2)$$

where x represents the chosen centroid. Starvation trace is utilized to weigh the distances used to select the centroid to be updated, such that

$$x = \text{argmin}_{c\in C}\left[\psi_c\|o-\mu_c\|\right] \quad (3)$$

where x is the centroid to be updated and C is the set of all centroids.

When a centroid is selected for an update, its mean is updated in the direction of the input vector and its variance estimate is updated as follows:

$$\mu_x = \mu_x + \alpha(o-\mu_x) \quad (4)$$

$$\sigma_x^2 = \sigma_x^2 + \beta\left[(o-\mu_x)^2 - \sigma_x^2\right] \quad (5)$$

Upon updating the selected centroid, the posterior distribution, $\Pr(o|s')$, is obtained using the normalized Euclidean distance between the input and each centroid c, such that

$$n_c = \sum_{i=1}^{d}\frac{(o_i-\mu_{c,i})^2}{\sigma_{c,i}^2} \quad (6)$$

$$p_c = \frac{n_c^{-1}}{\sum_{c'\in C}n_{c'}^{-1}} \quad (7)$$

where p_c represents the probability the observation belongs to the centroid c.

This is a departure from previous work (Karnowski et al., 2012; Karnowski et al., 2010), where the posterior distribution was calculated as either a simple function of the Euclidean distance or by sampling an exponential probability density function centered at the centroid mean. The former method is lacking because it does not take into account the variance of the data a centroid represents. The latter method is lacking because it tends to form unreasonably confident beliefs for input vectors that are not near any centroid. It also complicates the calculation without adding any more information content to the belief construct.

3. The DeSTIN node revisited

3.1. Recurrent clustering

In order to more easily map the DeSTIN architecture to a parallel implementation, the mechanism used by the original DeSTIN architecture to pass information top-down, reflected by the construct $\Pr(s_t|s_{t-1}, a_{t-1})$, needed to be revised. The philosophical approach taken was to integrate the feedback/recurrence mechanism as an inherent part of the clustering process. In previous work, the temporal regularities $\Pr(s_t|s_{t-1}, a_{t-1})$ were captured by maintaining a table populated with the likelihoods of transitioning between states or through a function approximation method that attempted to predict the next state given the current state. Though keeping a table of transition probabilities seems simple enough, the manner in which it was being used necessitated that an array be kept for every movement made across the image and for every possible belief state provided by the parent node. This results in a table that has a memory requirement for a single node of $M_{tab} = K^2AL$, where K is the number of centroids for the node, L is the number of movements, and A is the number of belief states from the parent. In addition to the large memory requirement, using this table mandated an additional set of operations outside of the core node functionalities of clustering and calculating $\Pr(o|s_t)$. The other previously used mechanism to estimate $\Pr(s_t|s_{t-1}, a_{t-1})$ is function approximation (Karnowski et al., 2011). While this method has a more modest memory footprint, it requires an extensive set of operations outside the core functionality of the node. These additional operations make it incredibly difficult to map the DeSTIN architecture to a parallel platform like a GPU. For this reason, it is desired to couple the learning of temporal regularities and parent advice more closely with the clustering mechanism.

To address this problem, recurrent clustering is proposed as illustrated in Fig. 2. Recurrent clustering takes as input the external input augmented by the node's previous time step belief. This allows the clustering to form beliefs that are based on both spatial and temporal attributes. Consequently, μ and σ^2 have dimensions $K \times (N + K)$, where K is the number of centroids and N is the number of input dimensions. It is important to note that in addition to allowing the clustering mechanism to capture temporal dependencies, this method allows the clustering algorithm to form centroids that represent relationships between the spatial and temporal features of the data. During the clustering process, the centroids will converge to values that represent spatial and temporal regularities in the data. Previously, the clustering algorithm could only observe the input vector and characterize it's similarity to other input vectors, but now the clustering results in belief states that represent information about transitions between belief states or, more generally, about a sequence of transitions between belief states, since each belief is dependent on the preceding belief.

There are many hazards to consider in designing the recurrent clustering mechanism due mainly to the introduction of a feedback loop. The most important aspect to consider is the method used for determining which centroid is to be updated and the calculation of its respective belief state. It is imperative that the clusters formed characterize both the temporal and spatial attributes of the data. As a result, it is important to balance the contributions of the spatial and temporal components of the input structure when selecting the centroid to update. This means that a selection method that uses the centroid variances to weight the importance of each dimension cannot be used because it encourages the recurrent clustering algorithm to form very confident beliefs that look only at the temporal features. The result is a system that can only act as a counter and provides no information about the input it observes. For this reason, the selection method used is based solely on the Euclidean distance between the centroid means and the combined input/belief vector, as suggested by the selection rule of Eq. (3). If the variances in the beliefs were expected to be much different than the variances in the input data, it might be necessary to use a normalized Euclidean distance with a constant normalization vector in order to prevent either the spatial or temporal features from improperly dominating the clustering process, however this has not been necessary in DeSTIN or any other applications explored here. Once the winning centroid has been updated, the belief state is calculated as outlined in Eqs. (6) and (7).

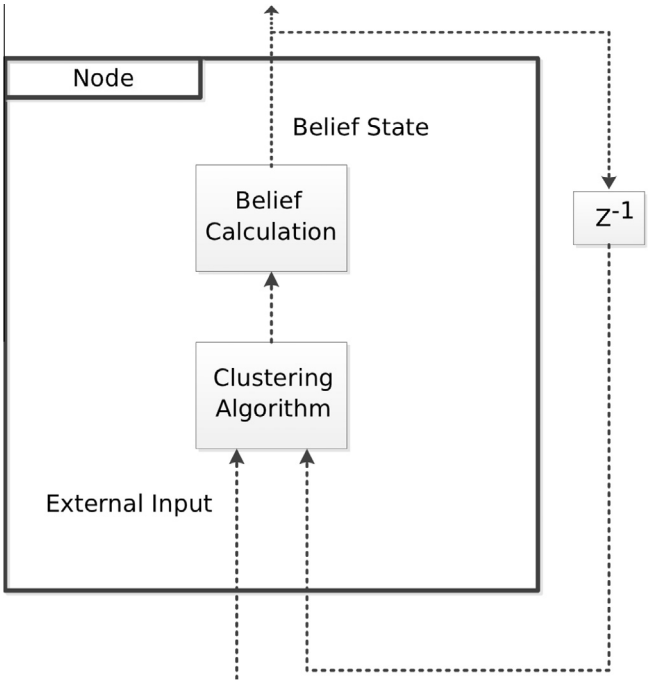

Fig. 2. In recurrent clustering the previous belief is latched and augmented to the input over which clustering is performed.

3.2. Enhanced cortical circuit

Examining Eq. (1) reveals that the system needs to be able to estimate the probability of the subsequent state given the information received from the parent node. This may be achieved simply by providing the belief state of the parent node as an additional input to the clustering algorithm of the child node, as depicted in Fig. 3. Thus, parent belief is handled much like the node's own previous belief and hence harmful feedback is avoided using the same mechanisms already employed inside the recurrent clustering algorithm. The system is now able to form beliefs based on local spatial information (the input), local temporal information (the node's previous belief state), and a more global form of advice in the form of the parent node's belief state.

The revised DeSTIN architecture is greatly simplified relative to its predecessor. The memory footprint has been reduced and consolidated into a simple set of two-dimensional matrices. Taking into account the dominating constructs involved, namely the centroid means, variances, starvation traces, and previous belief state, the memory requirement for a single node becomes $M_{node} = 2K(K+N) + 2K$ where K is the number of centroids and N is the number of input dimensions. There are only two core processes taking place at each node and those are very similar and share the same data structure. This reduced architecture, outlined in Algorithm 1, makes implementing DeSTIN on a GPU a far more realistic undertaking. It also suggests that larger topologies, which would be needed for larger problems (e.g. streaming video data), can fit onto a single GPU.

Algorithm 1. DeSTIN Pseudocode: This process is performed at every node in the pipelined hierarcy each when an example is presented to the hierarchy

$o \leftarrow [child_1.p_c \ \ldots \ child_N.p_c \ self.p_c \ parent.p_c]$
if TRAINING **then**
  $x \leftarrow \text{argmin}_{c \in C}[\psi_c \| o - \mu_c \|]$
  $\mu_x \leftarrow \mu_x + \alpha(o - \mu_x)$
  $\sigma_x^2 \leftarrow \sigma_x^2 + \beta\left[(o - \mu_x)^2 - \sigma_x^2\right]$
  $\psi_c \leftarrow \gamma\psi_c + (1-\gamma)\mathbb{1}_{x=c}$
end if
$n_c \leftarrow \sum_{i=1}^{d} \frac{(o_i - \mu_{c,i})^2}{\sigma_{c,i}^2}$
{Synchronize Nodes}
$p_c \leftarrow \frac{n_c^{-1}}{\sum_{c' \in C} n_{c'}^{-1}}$

4. Mapping DeSTIN to a parallel platform

4.1. Algorithmic analysis

In conventional software implementations, for each new observation an iteration of the DeSTIN algorithm for a single node is outlined by the following steps:

1. Calculate the distance from the observation to each centroid and take its reciprocal to obtain the unnormalized belief vector.
2. Normalize the belief vector and determine the winning centroid.
3. Apply the centroid update rule to the winning centroid.

To motivate an efficient implementation of DeSTIN using parallel processing platforms, the parallel algorithm's work, span, and degree of parallelism must be derived (Cormen et al., 2009). The distance calculation is the simple Euclidean distance, and its work is $O(|N||D|_{max}|C|_{max})$, where $|N|$ is the number of nodes in the network, and $|D|_{max}$ and $|C|_{max}$ are the maximal dimensionality and maximum number of centroids in the network, respectively. The belief vector calculation is $O(|N||C|)$, as we need to iterate over the list of centroids twice for each centroid: once to obtain the normalization sum and the winner, and once to normalize the vector. The update equation is $O(|N||D|)$, as we need to update each component of the winning centroid for each node in the network. Altogether, the overall work T_1 of the algorithm is $O(|N|)$.

On a parallel computing platform, the distance calculation's span is $O(log|D|)$ – each centroid is independent of the others, and the summation for the Euclidean distance is most efficiently computed through parallel reduction. Similarly, the belief vector's span is $O(log|C|)$, as a reduction sum is necessary to obtain the normalization term. The update equation's span is $O(1)$, as the update in each dimension is entirely independent of other dimensions and other nodes. Altogether, the span T_∞ for one iteration of the algorithm is $O(log|D|_{max} + log|C|_{max})$, yielding the parallelism of DeSTIN to be:

$$\frac{T_1}{T_\infty} = \frac{O(|N||D|_{max}|C|_{max})}{O(log|D|_{max} + logC_{max})}$$

There is a clear motivation for implementing DeSTIN over a parallel computing platform, especially given that the number of nodes in a DeSTIN hierarchy grows in $O(4^l)$, where l is the number of layers in the network.

4.2. Implementation details

4.2.1. Pipelining the DeSTIN architecture

In the serial implementation of DeSTIN, each layer depends on the result of the previous layer prior to completing its calculations. Since a serial implementation is only capable of processing one node at a time, this is not a detriment to speed. However, in a parallel implementation, this poses severe penalties on performance. If a layer must wait on another layer to be processed, then some parallel resources are idle when they could be utilized more efficiently. To address this deficiency, a pipelined approach is considered. During each time step of the algorithm, a particular node reads its input, executes the online clustering algorithm, and writes its output to a pipeline register. The output for each node on a given layer is then copied into the input for each node on the subsequent layer. Thus, each node n in layer l is operating on the beliefs of layer $l-1$ at time $t-l$, implying an increasing time delay for each layer.

4.2.2. GPU implementation considerations

At the core of the high-level CUDA API is the manipulation of grids, blocks, and threads (NVIDIA, 2012). A thread is an individual computation, a block is a collection of threads, and a grid is a collection of blocks. Threads communicate only with other threads in the same block, so it is ideal that each block is computationally independent of other blocks. The distance calculation works on a two-dimensional grid composed of blocks. The index in dimension i references a particular node and the index in the j dimension references a node's particular centroid. Each block is made of a one-dimensional array of threads where each thread k pertains to a particular component of a node's belief. Each thread computes the square of the difference d_k between node i's centroid's location $c_{j,k}^i$ and the input observation o_k^i to the node, such that

$$d_k = \left(c_{j,k}^i - o_k^i\right)^2$$

A parallel reduction sums up the vector d_k and stores the reciprocal of this summation into b_j^i, which is the j^{th} belief for node i.

The belief normalization works on a one-dimensional grid where each block in the i dimension references a particular node. Each block is a one-dimensional collection of threads where each thread in the j dimension references a node's belief. A parallel reduction computes two numbers: the summation of node i's belief and the index j of node i's most confident belief. Each belief is then normalized and the winning centroid is stored for the update procedure.

The belief update works on a one-dimensional grid where each block in dimension i references a particular node and each block is a one-dimensional collection of threads where each thread in dimension j relates to each component in the winning centroid's state. The update for each state component is calculated independently and in parallel.

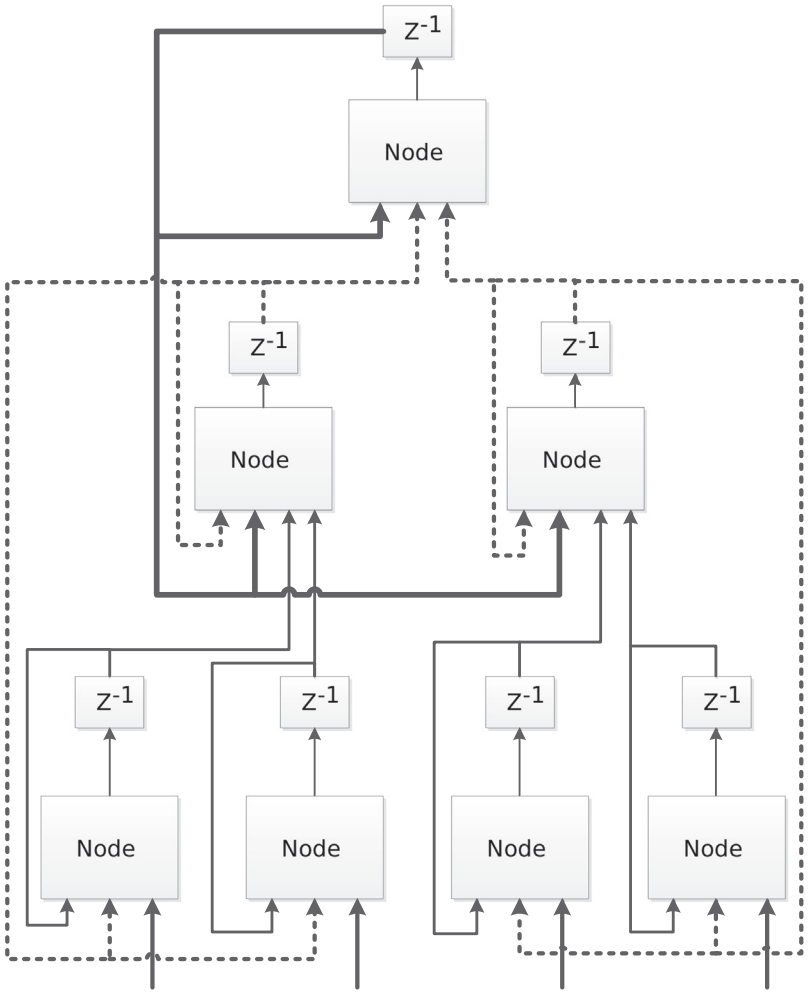

Fig. 3. A 4-layer DeSTIN architecture illustrating the bottom-up and top-down signaling that is involved. All nodes operate independently and in parallel such that each layer is delayed by one unit of time relative to the layer below it.

4.3. Speedup gain

To evaluate the relative speed gain of the GPU implementation compared to a CPU implementation, the network is instantiated with a four-layer hierarchy and a varying number of centroids per node and trains on some arbitrary input for 10,000 iterations. Constant-time operations such as network initialization are ignored and only the 10,000 iterations are timed. Fig. 4 shows a significant decrease in execution time for the GPU implementation versus the CPU implementation.

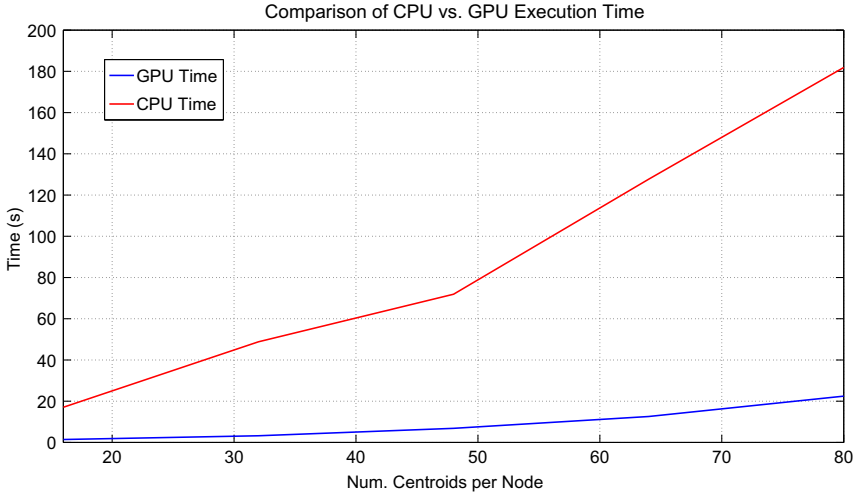

Fig. 4. A comparison of GPU (GeForce GT 640) versus CPU (Intel Core i7-3770) execution time given a four-layer hierarchy and a varying number of centroids per node.

5. Simulation results

We next present a series of experimental results that demonstrate the revised DeSTIN architecture's ability to capture temporal dependencies. In particular, we begin by focusing on the recurrent clustering algorithm as a key enabler for DeSTIN. First, its capabilities will be explored in depth in order to demonstrate its contribution in extracting temporal information from data even when the time scales of the important information is large. Consequently, the performance of the algorithm within a fully-hierarchical structure, as applied to a standard benchmark, will be presented.

5.1. Recurrent clustering for time-series prediction and sequence detection

A key attribute expected of recurrent clustering is recognition of patterns across time. As means of demonstrating this capability, the proposed recurrent clustering algorithm is applied to time-series prediction tasks. The first task explored is a frequency doubler where the objective is for system to take as input a sampled sinusoid signal with a period of N and to produce belief states that can be used as features to a simple feed-forward neural network whose output should be a sinusoid with half the period. This problem requires that the belief state captures information that at least spans the current and previous inputs.

Fig. 5 depicts the results of the frequency doubler test case. The algorithm was run with 24 centroids and the feed-forward neural net is resourced with 32 hidden neurons. As can be seen, the incremental algorithm was easily able to create features capturing temporal dependencies even for fairly slow, small changes taking place in the input, as reflected by larger periods. When the input signal period became too large, prediction error began to grow as a result of the small differences between samples which are challenging to represent using limited centroids. However, the resulting prediction remains consistently better than a random guess.

The second experiment was targeted at the algorithm's ability to capture temporal attributes, particularly in the context of detecting a binary sequence of interest within a general stream of binary inputs. The goal was to demonstrate the property of latching onto long-term temporal regularities. The length of the sequence of interest was varied in order to observe the impact of long sequences on the accuracy of the algorithm. The sequence of interest was a randomly chosen binary sequence of specific length. To further increase the challenge at hand, the overall input sequence was generated by randomly selecting (with probability 0.5) either the sequence of interest or the sequence of interest with the first binary element inverted. The belief states for the sequences of interest and the sequence with only the first bit altered were provided to a feed-forward neural network for the purpose of classifying each sequence. If the belief states accurately learn to represent regularities in the sequences presented, the classifier should be able to achieve a classification rate of 100%. A purely random selection (i.e. guessing) is represented by a classification rate of 50%.

Classification results for the sequence detection task are presented in Fig. 6 for varying sequence lengths and number of centroids. The classification rate observed decays exponentially with the length of the sequence, which is anticipated as a result of the unsupervised nature of the algorithm. Since there is no supervision that guides the algorithm to best identify a sequence of any specific length, the beliefs will always hold more information about more recent observations. The results indicate that there is an optimal range for the number of centroids used where the algorithm performs best. In the case of too few centroids, the belief state may not capture long time spans, while if there are too many centroids, the belief state may represent features in the data that are not relevant for identifying the sequence of interest. However, the algorithm exhibits weak sensitivity to the number of clusters, which is a desired property.

5.2. MNIST dataset

Leveraging the ability of the recurrent clustering algorithm to capture temporal dependencies, the scheme is next evaluated in the context of a full-scale deep architecture applied to a larger classification problem – the MNIST dataset (Lecun and Cortes, 1998). This dataset has been used extensively in the literature

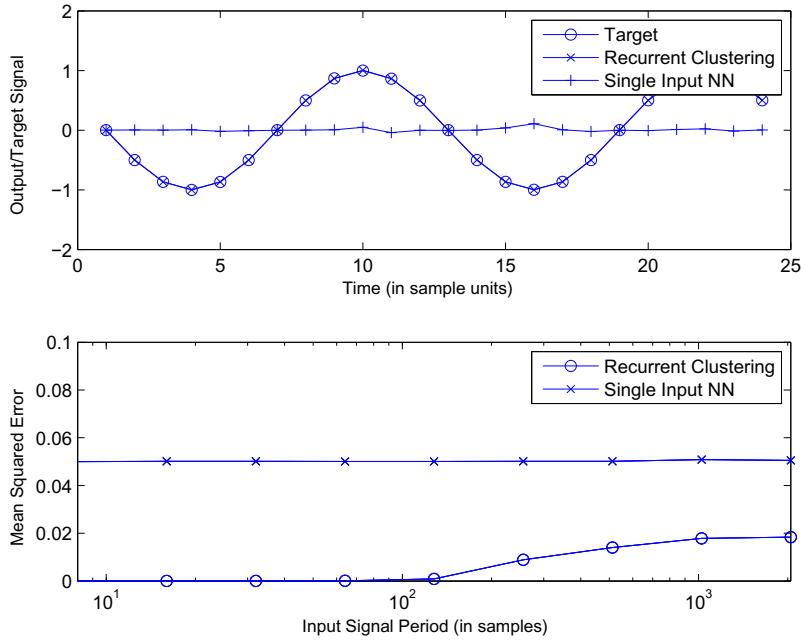

Fig. 5. Frequency doubler results: (top) the target vs. output plot is given for recurrent clustering and for a case where only the current value of the input is provided to the clustering algorithm. The period of the original signal is 24 sample units. (Bottom) the ability of the algorithm to capture information in long period sine waves is evaluated.

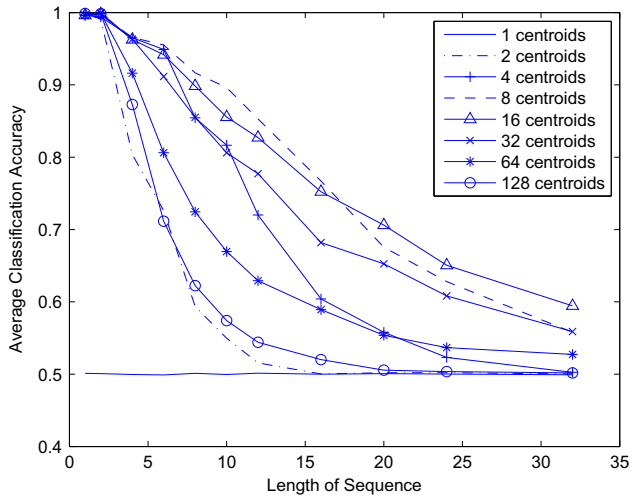

Fig. 6. Sequence detection results: this plot illustrates the average classification accuracy as a function of the length of the sequence of interest. The recurrent clustering algorithm is resourced with a varying number of centroids.

and contains 60,000 training images of digits 0–9 and 10,000 testing images. The images are 28×28 pixels in size, roughly centered, and are gray-scale. The images were padded with zeros to a size of 32×32 pixels in order to accommodate the movement sequence that provided the input to the lowest layer of the DeSTIN architecture. The 60,000 training images were elastically deformed (Simard et al., 2003) in order to form an additional 120,000 training images.

The DeSTIN hierarchy employed consisted of 3 layers with 4×4 nodes in the bottom layer, 2×2 nodes in the middle layer, and 1 node at the top layer. The movement sequence used is shown in Fig. 7. Each of the nodes in the bottom layer received a different 4×4 pixel patch of the input image, which results in the bottom layer viewing a 16×16 window during each movement. Nodes in every layer hosted a different number of centroids with the bottom, middle, and top layers having 32, 24, and 32 centroids, respectively. For training purposes, a random sampling of 15,000 of the training set images was used. Only 15,000 images are used since the clustering algorithm only needs to be able to accurately calculate the mean and variance of each centroid. Thus, as long as there are enough samples to accurately characterize the regularities in the data, no benefit will be gained by training on additional samples. Next, all 180,000 training images and 10,000 testing images were provided to the DeSTIN network in order to generate feature vectors for each image. A feature vector for each image consisted of the belief state of every node in the hierarchy sampled at every 12th movement. These feature vectors were then provided to a supervised learning based classifier in order to obtain classification results.

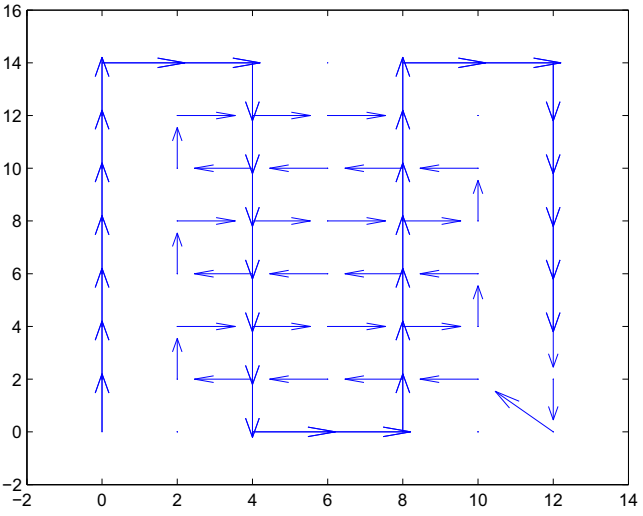

Fig. 7. This figure is a pictorial representation of the sequence of movements used for the MNIST dataset.

The supervised classifier consisted of an ensemble of 11 feed-forward neural networks trained with negative correlation learning. Each network hosted two hidden layers with 128 and 64 hidden neurons, respectively, and was trained to predict the posterior probability distribution over the classes. The cross-entropy error function was used in conjunction with a softmax output activation function, which ensured that the network outputs were within the range [0,1] and summed to one. All 180,000 training feature vectors (both elastic and non-elastic) were used in training, and inputs to the networks were scaled to the range [-1,1]. Using this experimental setup, a classification accuracy of 98.71% was achieved which is comparable to previous work involving the first-generation DeSTIN architecture which involved an additional layer and more complex computations. These results are also competitive results obtained for this benchmark achieved with other state of the art methods (Kégl et al., 2009; Salakhutdinov and Hinton, 2007; Simard et al., 2003).

5.3. PEMS-SF dataset

The proposed method was also tested on the PEMS-SF database (Cuturi et al., 2011). This dataset gives the relative occupancy rate of many lanes of traffic on the San Francisco area freeways. The data from 963 sensors was collected over 440 days every 10 min and the task is to classify each day as the correct day of the week (e.g. Monday). The dataset consists of 267 training samples and 173 testing samples. The DeSTIN hierarchy used was the same as before, except it had 50, 30, and 20 centroids in the nodes in the three layers respectively. Only 256 (16×16) of the 963 sensors were used in our tests to accommodate the 16×16 "viewing window" of the bottom layer and only 33 time-steps, every 4th starting from the beginning, were provided to the system. Now, instead of using the next movement over an image, the input layer is provided with data from the next time-step. Then, a feature vector was produced from the belief state of every node for every 11th time-step. This resulted in a feature vector made up of belief states that characterized each third of the day. This feature vector was then provided to the same classifier as before. A classification accuracy of 80.92% was achieved, which is comparable to other state of the art methods (Cuturi et al., 2011; Cuturi and Doucet, 2011) as can be seen in Table 1. Although this method did not establish new state of the art results on this dataset, it has demonstrated an ability to perform well on two very different types of datasets without any preprocessing or changes in the method to handle these differences. This ability to discover structure in many different types of data without altering the dataset to fit the method being used

Table 1
Comparison of results on PEMS-SF.

Method	Classification accuracy (%)
AR-Kernel	75
AR-Kernel using k	81
BOV Kernel	82
GA Kernel	79
SS Kernel	81
DeSTIN	81

is important to the proliferation of DML methods into new problem domains.

6. Conclusions and future work

This paper proposed a highly-scalable deep learning architecture that inherently captures both spatial and temporal dependencies in complex data yielding a rich, general-purpose feature extraction engine. Experimental results on image recognition tasks clearly demonstrated the performance attributes of the system achieved without the need for any user-defined features or parameter tuning. The scalability properties of the method, originating from its massively parallel nature, were substantiated via a GPU implementation, paving the path for employing the framework to other large-scale application domains, such as action sequence recognition in video streams or using DeSTIN as a state inference engine for control problems. The work done here in mapping DeSTIN to the GPU has cleared a trail for implementing it in custom analog hardware. While GPUs offer a far better price for performance compared to CPUs, custom analog hardware could provide much lower power requirements and scalability.

Disclaimer:

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, the Department of the Army, or the U.S. Government.

Acknowledgments

This work was supported by the Intelligence Advanced Research Projects Activity (IARPA) via Army Research Office (ARO) agreement No. W911NF-12-1-0017. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

References

Arel, I., Rose, D., Coop, R. 2009. Destin: a scalable deep learning architecture with application to high-dimensional robust pattern recognition, in: Proceedings of AAAI Workshop on Biologically Inspired Cognitive Architectures, pp. 1150–1157.

Arel, I., Rose, D.C., Karnowski, T.P., 2010. Deep machine learning – a new frontier in artificial intelligence research. IEEE Computational Intelligence Magazine 5 (4), 13–18.

Chen, Y.-N., Han, C.-C., Wang, C.-T., Jeng, B.-S., Fan, K.-C., 2006. The application of a convolution neural network on face and license plate detection. Proceedings of the 18th International Conference on Pattern Recognition, ICPR '06, vol. 03. IEEE Computer Society, Washington, DC, USA, ISBN 0-7695-2521-0, pp. 552–555. http://dx.doi.org/10.1109/ICPR.2006.1115.

Cormen, T.H., Liserson, C.E., Rivest, R.L., Stein, C., 2009. Introduction to Algorithms, third ed. MIT Press.

Cuturi, M., 2011. Fast global alignment kernels. In: Getoor, L., Scheffer, T. (Eds.), Proceedings of the 28th International Conference on Machine Learning (ICML-11). ACM, New York, NY, USA, ISBN 978-1-4503-0619-5, pp. 929–936.

Cuturi, M., Doucet, A. 2011. Autoregressive Kernels For Time Series, ArXiv e-prints.

Hinton, G.E., Osindero, S., Teh, Y.-W., 2006. A fast learning algorithm for deep belief nets. Neural Computation, 0899-7667 18 (7), 1527–1554. http://dx.doi.org/10.1162/neco.2006.18.7.1527.

Karnowski, T., Arel, I., Rose, D. 2010. Deep spatiotemporal feature learning with application to image classification, in: Ninth International Conference on Machine Learning and Applications (ICMLA), pp. 883–888. doi:http://dx.doi.org/10.1109/ICMLA.2010.138.

Karnowski, T.P., Arel, I., Young, S. 2011. Modeling temporal dynamics with function approximation in deep spatio-temporal inference network, In: BICA, pp. 174–179.

Karnowski, T.P. 2012. Deep Machine Learning with Spatio-Temporal Inference, Ph.D. Thesis, The University of Tennessee, Knoxville, Tennessee. <http://trace.tennessee.edu/utkgraddiss/1315.html>.

Kégl, B., Busa-Fekete, R., 2009. Boosting products of base classifiers. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09. ACM, New York, NY, USA, ISBN 978-1-60558-516-1, pp. 497–504. http://dx.doi.org/10.1145/1553374.1553439.

Krizhevsky, A., Hinton, G. 2009. Learning multiple layers of features from tiny images, Master's Thesis, University of Toronto, Toronto, Canada. <http://www.cs.toronto.edu/k˜riz/learning-features-2009-TR.pdf>.

Lecun, Y., Cortes, C. 1998. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 0018-9219 86 (11), 2278–2324. http://dx.doi.org/10.1109/5.726791.

Lee, T.S., Mumford, D., 2003. Hierarchical Bayesian inference in the visual cortex. The Journal of the Optical Society of America A 20 (7), 1434–1448. http://dx.doi.org/10.1364/JOSAA.20.001434 <http://josaa.osa.org/abstract.cfm?URI=josaa-20-7-1434> .

Lee, H., Grosse, R., Ranganath, R., Ng, A.Y., 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09. ACM, New York, NY, USA, ISBN 978-1-60558-516-1, pp. 609–616. http://dx.doi.org/10.1145/1553374.1553453.

Lee, H., Grosse, R., Ranganath, R., Ng, A.Y., 2011. Unsupervised learning of hierarchical representations with convolutional deep belief networks. Communications of the ACM 54 (10), 95–103.

Lockett, A.J., Miikkulainen, R. 2009. Temporal convolution machines for sequence learning, Tech. Rep. AI-09-04, Department of Computer Sciences, the University of Texas at Austin. <http://nn.cs.utexas.edu/?lockett:aitr09-04>.

Mobahi, H., Collobert, R., Weston, J., 2009. Deep learning from temporal coherence in video. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09. ACM, New York, NY, USA, ISBN 978-1-60558-516-1, pp. 737–744.

NVIDIA. 2012. NVIDIA CUDA Programming Guide 4.2, Tech. Rep. <https://developer.nvidia.com/cuda-toolkit-42-archive>.

Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., Phillips, J., 2008. GPU computing. Proceedings of the IEEE, 0018-9219 96 (5), 879–899. http://dx.doi.org/10.1109/JPROC.2008.917757.

Salakhutdinov, R., Hinton, G.E., 2007. Learning a nonlinear embedding by preserving class neighbourhood structure. Journal of Machine Learning Research – Proceedings Track 2, 412–419.

Simard, P., Steinkraus, D., Platt, J. 2003. Best practices for convolutional neural networks applied to visual document analysis, in: Proceedings of Seventh International Conference on Document Analysis and Recognition, pp. 958 – 963. doi:http://dx.doi.org/10.1109/ICDAR.2003.1227801.

Sutskever, I., Hinton, G. 2007. Learning multilevel distributed representations for high-dimensional sequences, Tech. Rep. <http://www.cs.toronto.edu/h˜inton/absps/trbmTR.pdf>.

Wallis, G., Bulthoff, H., 1999. Learning to recognize objects. Trends in Cognitive Sciences 3 (1), 23–31.

Wallis, G., Rolls, E.T., 1997. Invariant face and object recognition in the visual system. Progress in Neurobiology 51, 167–194.

Young, S., Arel, I., Karnowski, T.P., Rose, D. 2010. A fast and stable incremental clustering algorithm, in: Seventh International Conference on Information Technology: New Generations (ITNG), pp. 204–209. doi:http://dx.doi.org/10.1109/ITNG.2010.148.