Online Sparse Matrix Gaussian Process Regression and Vision Applications

Ananth Ranganathan[1] and Ming-Hsuan Yang[2]

[1] Honda Research Institute, Mountain View, CA 94041
aranganathan@honda-ri.com
[2] University of California, Merced, CA 95344
mhyang@ucmerced.edu

Abstract. We present a new Gaussian Process inference algorithm, called Online Sparse Matrix Gaussian Processes (OSMGP), and demonstrate its merits with a few vision applications. The OSMGP is based on the observation that for kernels with local support, the Gram matrix is typically sparse. Maintaining and updating the sparse Cholesky factor of the Gram matrix can be done efficiently using Givens rotations. This leads to an exact, online algorithm whose update time scales linearly with the size of the Gram matrix. Further, if approximate updates are permissible, the Cholesky factor can be maintained at a constant size using hyperbolic rotations to remove certain rows and columns corresponding to discarded training examples. We demonstrate that, using these matrix downdates, online hyperparameter estimation can be included without affecting the linear runtime complexity of the algorithm. The OSMGP algorithm is applied to head-pose estimation and visual tracking problems. Experimental results demonstrate that the proposed method is accurate, efficient and generalizes well using online learning.

1 Introduction

Learning regression functions from data has been an important problem in machine learning and computer vision with numerous applications. In recent years, kernel machines such as Support Vector Machines and Gaussian Processes have demonstrated great success in learning nonlinear mappings between high dimensional data and their low dimensional representations. For numerous vision applications where we have continuous stream of data, it is of great interest to learn nonlinear regression functions in an online manner.

In this paper, we propose a new Gaussian Process (GP) regression algorithm, called Online Sparse Matrix Gaussian Process (OSMGP) regression, that is exact and allows fast online updates in linear time for kernel functions with local support. This combination of exact inference and fast online updates is a novel contribution. We show that when the Gram matrix is sparse, as is the case when kernels with local support are used, an efficient representation is to maintain and update the Cholesky factor of the Gram matrix instead of the matrix itself. During online learning, when a new point is added to the training sequence, this introduces a new row and column into the Gram matrix. Instead of recomputing the Cholesky factor for the matrix, which would be expensive,

D. Forsyth, P. Torr, and A. Zisserman (Eds.): ECCV 2008, Part I, LNCS 5302, pp. 468–482, 2008.

we use Givens rotations to incrementally update it. Givens rotations are guaranteed to update the factorization in $O(n)$ time for a sparse matrix, where the Gram matrix has size $n \times n$, but can be much faster in practice.

We demonstrate that even though the Cholesky factor of the Gram matrix may become dense due to repeated applications of Givens rotations as training points are added, this can be overcome through the use of variable reordering, which restores sparsity of the matrix. If the hyperparameters of the GP are learned offline and not changed during online execution, the overall algorithm is linear. However, if these are learned automatically using a maximum likelihood method, this introduces a periodic quadratic update when the optimization is performed and the complete Gram matrix is recomputed.

As an additional contribution we propose the use of matrix downdating using hyperbolic rotations to also learn the hyperparameters of the GP in constant time. Hyperbolic rotations are used to incrementally recompute a matrix factorization when a row and column from the matrix are removed. This operation can be performed in $O(n)$ time similar to Givens rotations. Downdating introduces an approximation into the GP update but enables the Gram matrix to be maintained at a constant size by removing a training point from the training set whenever a new point is added. Thus, the size of the training set does not change. Hence, recomputing the Gram matrix after re-learning the hyperparameters can be done in constant time.

The proposed OSMGP algorithm is applied to head pose estimation and visual tracking problems. In particular, for head pose estimation, we use a dataset that is both comprehensive and exhaustive, containing 17 people and over 30,000 test images with ground-truth. Extensive comparisons with the existing methods show the accuracy, robustness and generality of our pose estimation system. For the tracking problem, we use publicly available datasets and demonstrate that the accuracy of the regression tracker is comparable to existing systems.

2 Gaussian Process Regression

A Gaussian Process is a distribution over the space of functions, which is usually defined as a collection of random variables, any subset of which have a Gaussian distribution. GPs can be viewed as probabilistic kernel machines, and hence, can provide not only a mean value prediction but also the uncertainty measured in terms of standard deviation for a test sample. A large standard deviation signals the absence of any training data in the neighborhood of the test sample, and provides an indication of poor generalization. A Gaussian Process is completely defined by a mean function $m(\mathbf{x})$ and a covariance function $k(\mathbf{x}, \mathbf{x}')$. A random function $f(\mathbf{x})$ distributed according to a GP is written as $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$. The GP is transformed into a probabilistic kernel machine if we take the covariance function to be a semi-positive definite Mercer kernel, such that the covariance between points $\mathbf{x}_i$ and $\mathbf{x}_j$ is given by $k(\mathbf{x}_i, \mathbf{x}_j)$.

For performing regression, we assume the availability of n training inputs $X = \{\mathbf{x}_{1:n}\}$ and corresponding outputs $\mathbf{y} = \{y_{1:n}\}$. The covariance function of the GP is then given by the $n \times n$ Gram matrix $K(X, X) = K$. Typically, the kernel function has a number of parameters θ, which are also called the hyperparameters of the GP, that have

to be learned using the training set. For example, for the Radial Basis Function (RBF) kernel given as $k(\mathbf{x}_i, \mathbf{x}_j) = c \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2}{2\eta^2}\right\}$, the hyperparameters are $\theta = (c, \eta)$. Hyperparameter learning is usually done by maximizing the marginal log-likelihood

$$p(\mathbf{y}|X, \theta) = -\frac{1}{2}\log\left|K + \sigma^2 I\right| - \frac{1}{2}\mathbf{y}^T\left(K + \sigma^2 I\right)^{-1}\mathbf{y} - \frac{n}{2}\log 2\pi \tag{1}$$

where I is the identity matrix of same dimensions as K and σ is the standard deviation of additive Gaussian noise. This learning step is performed offline since it involves inverting a potentially large Gram matrix.

The regression-based prediction for a given test point $\mathbf{x}^\star$ is given by the conditional distribution on the test output given the training data and the test input. This is again a Gaussian distribution $p(\mathbf{y}^\star|X, Y, \mathbf{x}^\star) = \mathcal{N}(\mu^\star, \Sigma^\star)$ with the predictive mean and covariance given by

$$\mu^\star = \mathbf{k}^{\star\mathbf{T}}\left(K + \sigma^2 I\right)^{-1}\mathbf{y}, \quad \Sigma^\star = k(\mathbf{x}^\star, \mathbf{x}^\star) - \mathbf{k}^{\star\mathbf{T}}\left(K + \sigma^2 I\right)^{-1}\mathbf{k}^\star \tag{2}$$

where $\mathbf{k}^\star = [k(\mathbf{x}^\star, \mathbf{x}_1), k(\mathbf{x}^\star, \mathbf{x}_2), \ldots, k(\mathbf{x}^\star, \mathbf{x}_n)]$.

The GP regression algorithm, as described above, is not useful for applications with large datasets since it does not scale with the number of data points. Training is $O(n^3)$ in complexity by (1) as it involves inversion of the Gram matrix. This rules out the straight-forward use of GPs in an incremental fashion. The runtime prediction given by (2) is, however, only $O(n)$ for computing the mean prediction but $O(n^2)$ for computing the variance, assuming that the inverse Gram matrix has been stored from the training phase and that it does not change during runtime. A comprehensive reference for GPs and their use in classification and regression tasks can be found in [1].

While many instances of modifications to the GP algorithm are available that overcome this hurdle to online computation, all of these involve approximations to the GP to reduce the complexity of the representation. A number of approximation schemes are based on the observation that very few training points contribute to the bulk of learned GP. Sparse GP schemes have been proposed, among others, by Snelson and Ghahramani [2], who based their approximation on learning "active points", and Csato and Opper[3], who proposed Online GPs that learn and update a sparse set of basis vectors. A unifying synthesis of these and other sparse GP methods is given in [4].

If the number of active points is D and the number of training points is n, these methods reduce the complexity of updating the GP to $O(nD^2)$ from the initial $O(n^3)$. In contrast, OSMGPs are $O(n)$ without any approximations, albeit for the special case of compact kernels.

We now present Online Sparse Matrix Gaussian Processes that can perform exact incremental updates to the GP in $O(n)$ time for kernel functions with local support.

3 Online Sparse Matrix Gaussian Processes

The proposed OSMGP algorithm works under the assumption that the covariance matrix of the GP is sparse. The use of kernel functions having local support results in most of the entries in the Gram matrix being close to zero since the kernel decays rapidly

as the distance between the vectors being evaluated increases. Many commonly used infinite-dimensional kernels have local support, a case in the point being the widely used Radial Basis Function (RBF), also known as the Gaussian or squared exponential kernel. This allows the use of sparse matrix algorithms that can perform online updates in linear time and are also exact, in contrast to the existing sparse GP algorithms.

To ensure the sparsity of the covariance matrix, we use "compactified" kernel functions [5]. This is because although kernels such as the RBF may produce a covariance matrix with many small entries, the entries in the matrix should be exactly zero for sparse matrix algorithms to be applicable. While thresholding the entries of the covariance matrix may seem the most straight-forward way to obtain a sparse matrix, this may result in the matrix not being positive definite. Compactifying the kernel function, i.e., modifying the kernel function to get one with compact support, ensures a positive definite matrix without compromising on the other characteristics of the kernel. For example, the RBF kernel can be compactified as

$$k(\mathbf{x}_i, \mathbf{x}_j) = c \exp\left(\frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{\eta^2}\right) \times \max\left(0, 1 - \left|\frac{\mathbf{x}_i - \mathbf{x}_j}{d}\right|\right)$$

where c and η are the RBF kernel parameters, and d defines the compact region over which the kernel has support. This modified kernel is positive definite [5].

The main difference between the OSMGP and existing GP algorithms is the representation of the covariance matrix as a sparse Cholesky factor. This is possible because the Cholesky factor of a sparse matrix is also sparse for some reordering of the variables. In the following discussion, we will take the upper triangular Cholesky factor to be the quantity that is maintained and updated.

3.1 GP Updates Using Kullback-Leibler Divergence

At each step during online learning of the GP, the algorithm is presented with a training pair that is used to update the existing model. The GP posterior is computed by taking into account the training output. Assuming at time t, the model is given by $p_t(f)$, it is updated upon receiving the training output y_{t+1} using Bayes law as $p_{t+1}(f) \propto p(y_{t+1}|f)p_t(f)$ where $p(y_{t+1}|f)$ is the measurement model.

Following Csato and Opper [3], we find the GP closest to the true posterior in the Kullback-Leibler divergence sense. This is done through moment matching using the parametrization lemma of [3]. Subsequently, we get the updated GP as

$$\langle f \rangle_{t+1} = \langle f \rangle_t + q^{(t+1)} \mathbf{k}_{t+1}, \;\; K_{t+1} = K_t + r^{(t+1)} \mathbf{k}_{t+1} \mathbf{k}_{t+1}^T \tag{3}$$

where $\langle . \rangle$ denotes the expectation operation, $\mathbf{k}_{t+1} = [K(\mathbf{x}_{t+1}, \mathbf{x}_1), \ldots, K(\mathbf{x}_{t+1}, \mathbf{x}_t)]^T$ and the update variables q and r are given as

$$q^{(t+1)} = \frac{\partial}{\partial \langle f_{t+1} \rangle_t} \ln \langle p(\mathbf{y}_{t+1}|f_{t+1}) \rangle_t \,, \;\; r^{(t+1)} = \frac{\partial^2}{\partial^2 \langle f_{t+1} \rangle_t} \ln \langle p(\mathbf{y}_{t+1}|f_{t+1}) \rangle_t \tag{4}$$

where $\langle . \rangle_t$ is the expectation with respect to GP at time t.

Updating the GP model using (3) involves a $O(n)$ update for the mean and an update for the covariance that is potentially $O(n^2)$. Here n is the number of training samples

presented thus far to the algorithm. However, this is a rank one update to a sparse matrix where the dimensions of the matrix increase by one during this update. The Cholesky factor of the covariance can, in this case, be updated in $O(n)$ through the use of Givens rotations which is described next.

3.2 Givens Rotations for Incremental Covariance Matrix Update

A standard approach to perform efficient, incremental Cholesky factorization uses *Givens rotations* [6] to zero out the entries below the diagonal, one at a time. To zero out the (i, j) entry, a_{ij} of a matrix A, we apply the Givens rotation

$$G \triangleq \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \tag{5}$$

to rows i and j, with $i > j$, which represents a rotation in a two-dimensional subspace of the states. ϕ is chosen so that a_{ij}, the (i, j) entry of the matrix, becomes 0.

$$(\cos\phi, \sin\phi) = \begin{cases} (1,0) & \text{if } \beta = 0 \\ \left(\frac{-\alpha}{\beta\sqrt{1+\left(\frac{\alpha}{\beta}\right)^2}}, \frac{1}{\sqrt{1+\left(\frac{\alpha}{\beta}\right)^2}} \right) & \text{if } |\beta| > |\alpha| \\ \left(\frac{1}{\sqrt{1+\left(\frac{\beta}{\alpha}\right)^2}}, \frac{-\beta}{\alpha\sqrt{1+\left(\frac{\beta}{\alpha}\right)^2}} \right) & \text{otherwise} \end{cases}$$

where $\alpha \triangleq a_{jj}$ and $\beta \triangleq a_{ij}$. This is illustrated in Figure 1 which shows how a Givens rotation can be applied to a matrix R which is triangular but for one entry. Note that the single Givens rotation does not ensure that the resulting matrix R' is triangular since the operation may give non-zero values to other elements to the right of a_{ij} in the ith and jth rows, shown in red in Figure 1.

After all the non-zero entries below the diagonal are zeroed out by application of Givens rotations, the upper triangular entries contain the new Cholesky factor. Note that a sparse matrix yields a sparse Cholesky factor for an appropriate variable ordering.

Applying Givens rotations yields an efficient update algorithm. In general, the maximum number of Givens rotations needed for adding a new row of size n is $O(n^2)$. However, as both the covariance matrix and the new row are sparse, only $O(n)$ Givens rotations are needed. We observe that in practice it is typically much faster than this.

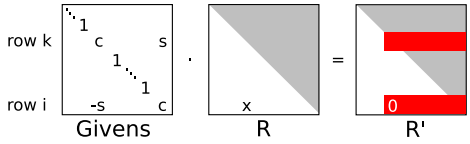


Fig. 1. Using a Givens rotation as a step in transforming a general matrix into upper triangular form. The (i, j)th entry, marked 'x' here, is eliminated, changing some of the entries in the ith and jth rows marked in red (dark), depending on sparsity.

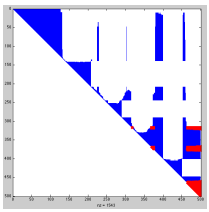


Fig. 2. Updating a sparse Cholesky factorization after adding a row and column is $O(n)$ using Givens rotations for a $n \times n$ matrix but is often much faster in practice. In the above sparsity pattern of Cholesky factor (non-zero entries are marked in blue or red), entries whose values have changed after the update are shown in red and unchanged entries are shown in blue. In this 500×500 matrix, only about 1,500 entries are modified.

As an example, Figure 2 shows the entries in a matrix of size 500×500 that change value upon updating the factorization after adding a row and a column. The algorithm would be $O(n^2)$ if the whole matrix needed to be recomputed. However, only a small number of entries are recomputed in practice.

4 Variable Reordering and Hyperparameter Optimization

While the above sections dealt only with GP update, the need to periodically learn the GP hyperparameters and maintain the sparsity of the Cholesky factor of the Gram matrix can cause inefficiencies. We address these issues in this section.

First, while the parameters can be learned offline using an initial training set, this makes the GP less responsive to changes in the test set during runtime. However, once new hyperparameters are available, the covariance matrix has to be recomputed completely and re-factorized using a batch Cholesky decomposition. This operation could take $O(n^3)$ in theory but is closer to $O(n^2)$ in practice if sparse Cholesky decomposition methods are used [7].

Second, the runtime of the OSMGP depends crucially on the sparsity of the Gram matrix. However, as Givens rotations are incrementally performed to update the GP, *fill-in* can occur in the Gram matrix. Fill-in is defined as non-zero entries beyond the sparsity pattern of the Gram matrix, i.e., entries that are zero in the Gram matrix become non-zero in the Cholesky factor. This occurs because the Cholesky factor of a sparse matrix is guaranteed to be sparse for some variable orderings but not for all of them.

We avoid fill-in by *variable reordering*, a technique well known in the linear algebra community, using a heuristic to efficiently find a good column ordering. The order of the columns (and rows) in the Gram matrix influences the variable elimination order and therefore also the resulting number of entries in the Cholesky factor. While obtaining the best column variable ordering is NP hard, efficient heuristics such as the COLAMD (*col*umn *a*pproximate *m*inimum *d*egree) ordering [8] and Nested Dissection [9] perform well in practice. The Cholesky factor after applying the COLAMD ordering, for instance, shows negligible fill-in, as can be seen in Figure 3. Reordering the variables also needs a re-factorization of the Gram matrix with its attendant higher complexity.

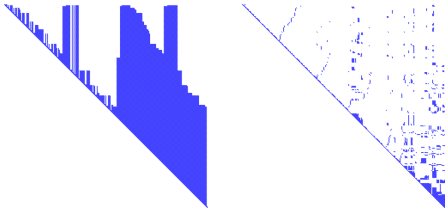


Fig. 3. The sparsity pattern of the Cholesky factor (non-zero entries marked in blue) for factorization without reordering (left), and using COLAMD reordering (right). The reordered Cholesky factor has very few non-zero entries and so is much sparser.

We propose fast incremental updates with periodic variable reordering and hyperparameter optimization using (1), where the latter two are combined in a single step, thus requiring only a single re-factorization. When combined with incremental updates, this avoids fill-in, relearns hyperparameters to provide a responsive model, and still yields a fast algorithm as is supported by the results in Section 6.

5 Matrix Downdates and $O(1)$ Operation

The complete OSMGP algorithm as described above has $O(n)$ runtime complexity. This is because the GP update step (3) has $O(n)$ runtime due to the use of Givens rotations while the regression prediction (2) also has $O(n)$ runtime since it can be implemented using sparse back-substitution. The mean prediction can be found by computing $(K + \sigma^2 I)^{-1}\mathbf{y}$ as the solution to the linear system $(R^T R)\mathbf{x} = \mathbf{y}$ where R is the upper triangular Cholesky factor of the Gram matrix. This linear system can be solved using two back-substitution operation. Although in normal operation, back-substitution is an $O(n^2)$ operation, it is $O(n)$ for sparse matrices.

While the linear runtime complexity may be good for most applications, in many other situations we require a constant time scaling, at least for the prediction. Further, since the covariance matrix in the above case grows with the number of training samples, storage requirement also increases over time. We can get around these constraints and obtain a constant time algorithm by introducing approximations into the current scheme, which is exact except for the posterior projection (4).

Our approach is based on the sliding window approach to least squares problems [10], where old measurements are successively discarded as new ones arrive. For the OSMGP, this implies discarding an old training sample for every new one that is provided, thus keeping the number of samples based on which the GP is learned constant. We propose this "oldest first" discarding strategy since, during online operation, it is more likely that future test samples are similar to the latest observed samples. However, other discarding strategies can also be accommodated in the algorithm.

Maintaining the covariance matrix at a fixed size of $W \times W$, where W is the window size, makes both the prediction and the GP updates have $O(W)$ time instead of $O(n)$. Further, even the hyperparameter optimization and variable ordering can be done in $O(W^2)$. Note that W can be quite large (in the thousands) and yet, can be done efficiently, since all the operations are carried out on sparse matrices

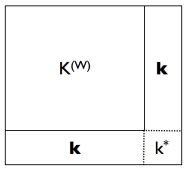


Fig. 4. Illustration of variable grouping for the downdate process. See text for explanation.

The only operation to be described in this constant time, constant space algorithm (for fixed W) is the update of the GP when discarding a training sample. Discarding a training sample involves deleting a row and a column from the covariance matrix. Inverting the rank one update from (3), this can be done using a rank one downdate of the covariance matrix. Assuming, without loss of generality, that the $(W+1)$th row and column are to be removed from a $(W+1)\times(W+1)$ matrix $\tilde{K}$ to get a downdated $W \times W$ matrix K, this can be done as

$$K = K^{(W)} - \frac{\mathbf{k}\mathbf{k}^{\mathbf{T}}}{k^{\star}}$$

where $\mathbf{k}$, $k^{\star}$, and $K^{(W)}$ are defined as in Figure 4.

The rank one downdate can be performed efficiently using Hyperbolic rotations [6], which are defined analogous to Givens rotations. To zero out the (i,j) entry, a_{ij} of a matrix A, we apply the Hyperbolic rotation

$$H \triangleq \begin{bmatrix} \cosh\phi & -\sinh\phi \\ -\sinh\phi & \cosh\phi \end{bmatrix} \tag{6}$$

to rows i and j, with $i > j$. The parameter ϕ is chosen so that a_{ij}, the (i,j) entry of the matrix, becomes 0.

$$(\cosh\phi, \sinh\phi) = \begin{cases} (1,0) & \text{if } \beta = 0 \\ \left(\frac{\alpha}{\beta\sqrt{1-\left(\frac{\alpha}{\beta}\right)^2}}, \frac{1}{\sqrt{1-\left(\frac{\alpha}{\beta}\right)^2}} \right) & \text{if } |\beta| > |\alpha| \\ \left(\frac{1}{\sqrt{1-\left(\frac{\beta}{\alpha}\right)^2}}, \frac{\beta}{\alpha\sqrt{1-\left(\frac{\beta}{\alpha}\right)^2}} \right) & \text{otherwise} \end{cases}$$

where $\alpha \triangleq a_{jj}$ and $\beta \triangleq a_{ij}$. As with the Givens rotations, we apply hyperbolic rotations until all the elements of the row and column in question have been zeroed out. This is a linear time operation for sparse matrices.

Hyperbolic rotations have a drawback that they can be numerically unstable. However, if the initial and final matrices after the downdate have full rank, which is the case here, instability usually does not occur. Though we have not encountered numerical instability in our experiments, an unstable case can be dealt with using more sophisticated downdate techniques, e.g., [11].

The approximate, constant time OSMGP algorithm can now be summarized as follows. Whenever a training sample is presented, we include it and update the GP using

Givens rotations. If the window size has not yet been reached, no more work needs to be done. However, if the window size is exceeded, we then discard the oldest training sample by removing the corresponding row and column from the Cholesky factor of the covariance matrix, using hyperbolic rotations for downdating.

6 Applications

We now illustrate OSMGPs using two challenging computer vision applications, namely head pose estimation and tracking. The accuracy of the OSMGP regression algorithms is compared against the existing methods in both cases.

6.1 Head Pose Estimation

Information about head pose provides an important cue in human interaction that people routinely estimate and use effortlessly. Our motivation for studying head pose estimation is to enable more natural interaction between humans and machines. Following a person's viewpoint provides machines with valuable contextual information and also enables faster reaction to user actions such as pointing. However, automatic estimation of head pose from visual data has proven to be a difficult problem. This is due to the difficulty in dealing with wide variations in pose angle and also with generalizing the estimation algorithm across face images of different identities. Finally, all this has to be performed in real-time to keep up with the speed of human head motion.

A number of head pose estimation techniques have been proposed , ranging from prototype matching [12] to 3D modeling [13] to feature-based tracking [14]. A few techniques that use regression also exist, e.g. using neural networks [15], and Support Vector Machines [16]. Another work related to ours is given in [17], which discusses head pose estimation using an offline semi-supervised algorithm for GP regression. However, all of these techniques are limited in that they either deal with a limited range of angles or use coarse discretization in angle space. In addition, few systems focus on real-time applicability. In contrast, our method works in a large, continuous angle space and operates in real time.

Our OSMGP-based head pose estimation system is fully automatic and incorporates face detection, tracking, and head pose estimation. Face detection is performed using a cascade detector [18], while tracking is performed using an incremental visual tracker [19]. The tracked face image is used as input to the GP regression algorithm. Histogram equalization is performed on the tracker output to remove illumination changes to some extent. Since the roll angle of the head pose is given by the orientation of the tracking window, only the yaw and pitch angles are learned using GP regression.

The tracker operates on an normalized image of size 32×32 pixels. We experiment with several dimensionality reduction techniques and opt for Principal Component Analysis (PCA) since it gives the best results in conjunction with GP regression. From our experiments, mixtures of probabilistic PCA tend to cluster training images by identity (rather than pose) for non-extreme poses, which is undesirable, while clustering face images in the subspace by the ground truth pose angles produces overlapping and incoherent clusters. Some dimensionality reduction techniques are evaluated in the results.

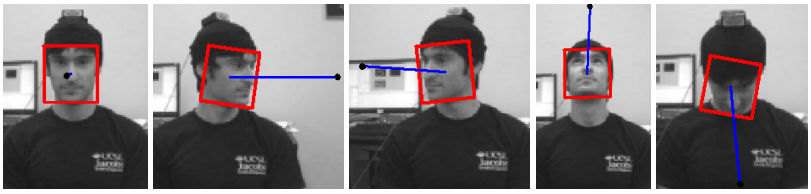


Fig. 5. Sample output of our head pose estimation system for (yaw,pitch,roll) values in degrees of (left to right) $(-2.2, -1.6, 1.8)$, $(-55.6, -6.6, 11.7)$, $(59.0, -3.3, -9.0)$, $(2.4, -24.1, 3.2)$, and $(1.0, 19.7, 11.9)$. The cap with mounted IMU is used to collect ground truth data.

We have gathered a large dataset of labeled training images. This is accomplished by instrumenting subjects with a cap on which a 3D Inertia Measurement Unit (IMU) is mounted. The IMU unit used for this purpose introduces noise into the dataset due to slippage of the IMU and due to magnetic noise in the vicinity. The data set consists of 17 subjects with over 30,000 images with ground truth pose angles. The head poses in the dataset range from -60^o to 60^o in yaw, -55^o to 40^o in pitch, and -35^o to 35^o in roll. We intend to make this dataset publicly available in the near future.

Our head pose estimation system is implemented in C++ and runs at 8 frames per second. We perform frame-based pose estimation which has the advantages of easier failure recovery and applicability to any still image. For fair comparison with other GP algorithms, we used a Matlab implementation. Publicly available Matlab implementations of the Online GP (OGP) [3] and the Sparse Pseudo-input GP (SPGP) [2] are used. In all the experiments, PCA is used to project the face images onto a 30 dimensional space on which the regression functions are learned. Our system assumes that the images are captured using a parallel frontal projection, i.e., the subject's head is orthogonal to the image plane. However, a wide variation in the head pose angles is supported. Figure 5 shows some head pose estimation results.

We first compare the efficiency of OSMGPs with OGP. For this purpose, the exact OSMGP algorithm with no downdates is used, while OGP is used with 200 basis vectors. An initial GP model is learned offline with 5,000 training points using the respective algorithms. Figure 6 shows the GP update time on a log scale for the two algorithms for a further 15,000 online updates. Note that once the maximum basis vector number is reached, OGP has constant time operation, while OSMGP has a $O(n)$ growth. However, even after 15,000 updates, the runtime of the exact OSMGP is still less than OGP. While OSMGP runtime increases monotonically if approximations using downdating are not performed, this shows that even when using an exceedingly large window size, 15,000 in this case, OSMGP is still a faster algorithm than OGP. Hyperparameter optimization is performed every 1,000 steps for OSMGPs which manifests itself as spikes in the time graph. Variable reordering is not required as the matrix remained sparse. No optimization is performed in the case of OGPs. In spite of this, the average time taken by the OSMGP per step is less than the OGP.

We compare the results of using OSMGP regression for head pose estimation against other popular kernel regression methods. The methods compared are the OGP, the Multivariate Relevance Vector Machine (MVRVM) [20], and SPGP. The publicly available implementation of MVRVM is used for fair comparison. In addition, we also evaluate

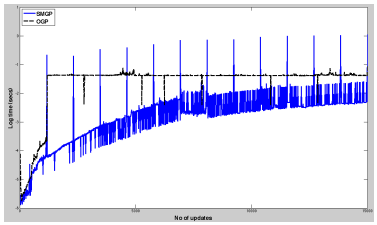


Fig. 6. Time per online update step for 15,000 steps comparing the OGP algorithm with the OSMGP. Time in seconds is shown on a log scale. Optimization is performed every 1,000th step in the OSMGP which accounts for the spikes in time.

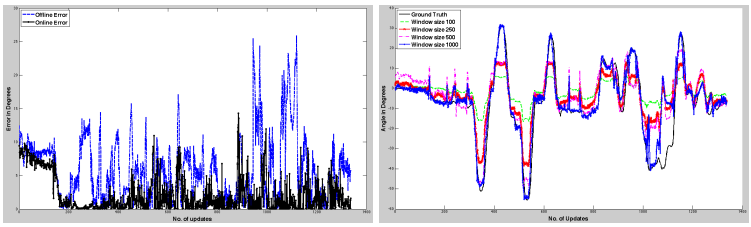
the use of different dimensionality reduction techniques in conjunction with the OSMGP, such as mixtures of PPCA with 5 clusters, mixtures of PPCA clustered by training pose with 5 clusters, and the Gaussian Process Latent Variable Model (GPLVM) [21]. OSMGP and OGP are trained online with hyperparameter estimation every 500 steps. No downdating is done for the OSMGP, while 250 basis vectors are used for the OGP. All the models are first learned offline using a 5,000 point dataset. A 15,000 point test set, the same as the one used in the timing experiment above, is used to obtain pose predictions. Note that the prediction error can be further reduced by filtering the output using the variance provided by the GP. Mean errors for the yaw and pitch angles are given in Table 1. OSMGP performs better than all the other algorithms. The OGP is slightly worse, mainly due to the approximation involved in the use of basis vectors. SPGP and MVRVM give significantly worse results due to the lack of online learning and hyperparameter estimation. Other dimensionality reduction techniques produce results that, in many cases, are rather poor. This is because the clustering is quite poor in the case of mixtures of PPCA, while GPLVM often clusters by identity of the face, rather than by pose, as desired.

A second experiment demonstrates the benefit of online learning with hyperparameter estimation. Two OSMGP models are trained offline using 5,000 training points.

Table 1. Overall pose estimation accuracy of various regression algorithms on the 15,000 point test set. Columns 2 and 3 show the error for cases when the test subject is included and excluded in the training set respectively. The first number in each entry is the mean yaw error while the second is mean pitch error. Mean roll error obtained from the tracker is 5.88^o. The last three rows tabulate the results from the OSMGP algorithm when used with different dimensional reduction techniques.

Algorithm	*Mean Pose Error 1*	*Mean Pose Error 2*
OSMGP	2.99^o, 3.07^o	6.11^o, 7.53^o
OGP	3.57^o, 3.81^o	7.13^o, 8.11^o
SPGP	4.01^o, 4.03^o	9.70^o, 10.69^o
MVRVM	4.02^o, 4.17^o	9.26^o, 10.97^o
OSMGP + Mixture of PPCA	8.80^o, 8.11^o	12.19^o, 16.64^o
OSMGP + GPLVM	10.93^o, 10.98^o	10.80^o, 11.03^o
OSMGP + MPPCA by pose	7.14^o, 8.09^o	11.58^o, 12.91^o

Subsequently, both models are used to predict poses for a sequence of 1,340 test images. The first OSMGP is updated online by providing the corresponding training point after each prediction, while the second OSMGP is kept fixed. Further, the hyperparameters of the first OSMGP are updated after every 100 steps. Variable reordering is performed in both cases as required and no downdates are applied. Figure 7(a) gives the results of the experiment in the form of the error as measured from the ground truth given by the IMU. It can be seen that although both models start by predicting poses with the same error, the model with online learning quickly improves and demonstrates much lower error over the whole sequence. The mean error over yaw and pitch for the OSMGP with online learning is 3.07^o, while for the second, fixed OSMGP, it is 7.11^o, i.e., worse by more than a factor of two.



(a) (b)

Fig. 7. (a) Comparison of mean prediction error, pitch and yaw combined, for a OSMGP with online learning as opposed to one with no online learning. (b) A comparison of predicted pitch for various window sizes over the same test sequence.

The effect of varying the window size in the approximate, constant time OSMGP is illustrated in Figure 7(b). This shows the predictive pitch angle for the same sequence of 1,340 images as above with various window sizes ranging from 100 to 1,000. No hyperparameter estimation is done in any of the OSMGPs in this case although online updates are performed. A window size of 1,000 produces a mean prediction error of 4.02 degrees in the pitch angle which is close to the value observed in the above experiment. The ground truth, as obtained from the IMU, is also shown in the figure for reference. The mean errors for window sizes of 100, 250, and 500 are 7.07^o, 6.59^o, and 5.22^o respectively.

Figure 7(b) also shows that most of the prediction error occurs in cases of extreme poses, i.e., absolute yaw or pitch angles greater than about 40^o. One reason for this is that at these angles, the visual tracker is somewhat unstable as little facial information is available. Second, the number of training images with these poses are also relatively few, since people do not maintain these positions for a significant amount of time. Correspondingly, the mean errors in Table 1 are skewed due to these extreme poses. If the pose range is constrained, prediction errors can be reduced significantly. Further, due to the skew in training data, the variance given by the GP predictor is much larger at these extreme poses than that for other poses. For instance, the maximum standard deviation of 2.59^o is obtained for a prediction of yaw and pitch values equal to 61.2^o and 31.7^o respectively. In contrast, for the corresponding angle predictions of 0.9^o and 1.2^o, the standard deviation has the more typical value of 0.28^o.

6.2 Visual Tracking

We next illustrate the application of OSMGPs to visual tracking in a regression-based framework. The tracking algorithm is similar to [22] in which an RVM is used, whereas in this work the proposed OSMGP algorithm is used to learn a regression function. As in [22], we use "seed images", where the location of the object of interest is known. The object extracted from these seed images is perturbed along the two translation axes to obtain training images using which a regression function from the image to the displacement is learned. Perturbed training images are generated in a 40 pixel window along both translation axes. Since our main goal is to demonstrate the applicability of OSMGPs, we do not take into account rotation and scaling in our tracker although these can be accommodated. We also do not perform any state prediction but simply update the tracked region on a frame-by-frame basis.

The regression-based tracker is implemented using OSMGP, MVRVM, and SPGP. Two publicly available datasets from [19], namely the Fish and Sylvester datasets are used to evaluate these algorithms. Both datasets are challenging as the target objects undergo large appearance variation due to change in illumination and viewpoints. To create the training sets, three seed images are chosen randomly in each dataset and perturbed as explained above. Ground truth is collected for computing the tracking errors of each tracker and for initialization. The tracking error of each frame is computed as $L2$ distance between the center of the tracked region and the ground truth. Further, if the tracking error of a tracker is larger than 40 pixels along any axis, the tracker is declared to have lost track and is reinitialized using the ground truth of that frame. The OSMGP with downdating and a window size of 500 is used. Both the MVRVM and the SPGP are learned with an active point set of size 100, as beyond this performance improvement is negligible. The tracking results from these experiments are presented in Table 2. The OSMGP based tracker has the least number of tracking failures and the least mean tracking error, especially for the Fish sequence. As the target object in the Fish sequence undergoes large and continuous illumination change, it is important to learn regression function in an online manner. The tracking results of the OSMGP based tracker for the two sequences is illustrated in Figure 8, and more results can be found in the supplementary material. All the trackers run at approximately 5 frames per second with MATLAB implementations on a 2.4GHz machine. The OSMGP is not significantly faster since the Gram matrix is not as sparse as in the head pose estimation scenario. As the perturbed images look alike in this type of tracking algorithm, the corresponding Gram matrix entries are non-zero and thus the matrix is relatively non-sparse. Consequently, their runtime goes up as the complexity of OSMGP depends on

Table 2. Tracking performance of different regression algorithms on the two test sequences

Algorithm	Fish Sequence		Silvester Sequence	
	# of Failures	***Mean Tracking Error***	***# of Failures***	***Mean Tracking Error***
OSMGP	6	8.3 pixels	1	1.6 pixels
SPGP	12	13.6 pixels	4	3.1 pixels
MVRVM	11	11.2 pixels	2	3.1 pixels

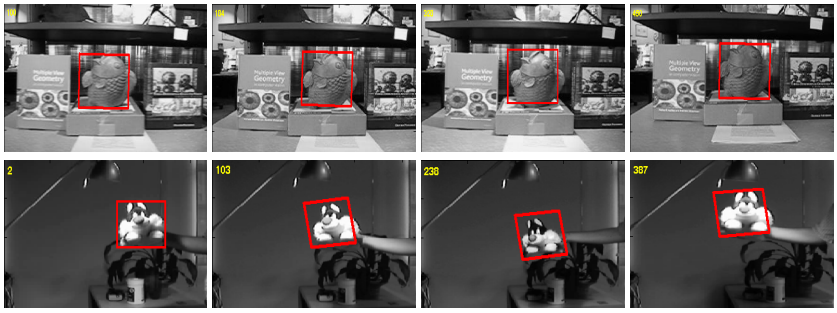


Fig. 8. Results of the OSMGP tracker for both test datasets on approximately 500 frames of video

the number of Givens and Hyperbolic rotations. In the head pose estimation case, most of the Gram matrix entries are zero as those images are obtained from different people with large pose variation. These experiments show that even in the case where the Gram matrix is not sparse, OSMGPs are still faster than the existing regression algorithms and can be significantly faster in many cases (such as the head pose estimation application).

7 Conclusion

We have presented a new Gaussian Process algorithm, Online Sparse Matrix Gaussian Processes, that can be used for exact, online learning with $O(n)$ time growth or in an approximate manner with $O(1)$ time. The algorithm is applicable to kernels with compact support, which result in a sparse covariance matrix. We demonstrated the use of the OSMGP algorithm in the context of two vision applications:

- a 3D head pose estimation system that operates in continuous angle space without any discretization, generalizes across faces of different identities, and provides good results even for large angle variations.
- a regression-based tracker that can operate under significant illumination and viewpoint changes.

In principle, the idea of sparse matrix manipulations can be extended to other probabilistic kernel machines, such as Relevance Vector Machines. We plan to pursue this idea, and extend the proposed algorithm to the semi-supervised learning setting.

References

1. Rasmussen, C.E., Williams, C.: Gaussian Processes for Machine Learning. MIT Press, Cambridge (2006)
2. Snelson, E., Ghahramani, Z.: Sparse Gaussian processes using pseudo-inputs. In: Advances in Neural Information Processing Systems, pp. 1259–1266 (2006)
3. Csato, L., Opper, M.: Sparse online gaussian processes. Neural Computation 14(2), 641–669 (2002)

4. Quinonero-Candela, J., Rasmussen, C., Williams, C.: Approximation methods for gaussian process regression. In: Large-Scale Kernel Machines, pp. 203–224. MIT Press, Cambridge (2007)
5. Hamers, B., Suykens, J., Moor, B.D.: Compactly Supported RBF Kernels for Sparsifying the Gram Matrix in LS-SVM Regression Models. In: Proceedings of the International Conference on Artificial Neural Networks, pp. 720–726 (2002)
6. Golub, G., Loan, C.V.: Matrix Computations. Johns Hopkins University Press (1996)
7. Kaess, M., Ranganathan, A., Dellaert, F.: Fast incremental square root information smoothing. In: Proceedings of International Joint Conference on Artificial Intelligence, pp. 2129–2134 (2007)
8. Davis, T., Gilbert, J., Larimore, S., Ng, E.: A column approximate minimum degree ordering algorithm. ACM Transactions on Mathematical Software 30(3), 353–376 (2004)
9. Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal 49(2), 291–307 (1970)
10. Zhao, K., Fuyun, L., Lev-Ari, H., Proakis, J.: Sliding window order-recursive least-squares algorithms. IEEE Transactions on Acoustics, Speech, and Signal Processing 42(8), 1961–1972 (1994)
11. Bjorck, A., Park, H., Elden, L.: Accurate downdating of least-squares solutions. SIAM Journal on Matrix Analysis and Applications 15(2), 549–568 (1994)
12. Kruger, N., Potzsch, M., von der Malsburg, C.: Determination of face position and pose with a learned representation based on labeled graphs. Image and Vision Computing 15(8), 665–673 (1997)
13. Yang, R., Zhang, Z.: Model-based head pose tracking with stereo vision. In: Proceedings of the International Conference on Automatic Face and Gesture Recognition, pp. 242–247 (2002)
14. Yao, P., Evans, G., Calway, A.: Using affine correspondence to estimate 3D facial pose. In: Proceedings of IEEE International Conference on Image Processing, pp. 919–922 (2001)
15. Rae, R., Ritter, H.: Recognition of human head orientation based on artificial neural networks. IEEE Transactions on Neural Networks 9(2), 257–265 (1998)
16. Li, Y., Gong, S., Liddell, H.: Support vector regression and classification based multi-view face detection and recognition. In: Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition, pp. 300–305 (2000)
17. Williams, O., Blake, A., Cipolla, R.: Sparse and semi-supervised visual mapping with the S3GP. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 230–237 (2006)
18. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: IEEE Conf. on Computer Vision and Pattern Recognition, vol. 1, pp. 511–518 (2001)
19. Ross, D., Lim, J., Lin, R.S., Yang, M.H.: Incremental learning for robust visual tracking. International Journal of Computer Vision 1–3, 125–141 (2008)
20. Thayananthan, A., Navaratnam, R., Stenger, B., Torr, P., Cipolla, R.: Multivariate relevance vector machines for tracking. In: Proceedings of European Conference on Computer Vision, vol. 3, pp. 124–138 (2006)
21. Lawrence, N.: Gaussian process latent variable models for visualization of high dimensional data. In: Advances in Neural Information Processing Systems, pp. 329–336 (2004)
22. Williams, O., Blake, A., Cipolla, R.: Sparse bayesian regression for efficient visual tracking. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(8), 1292–1304 (2005)