Convolutional Sparse Coding

Amir Kosrowshahi, Urs Köster

May 23, 2014

1 Sparse coding model

Sparse coding [**?**, **?**, **?**] is a latent variable model that attempts to describe data in terms of a small number of additive components, or basis functions, selected out of a large dictionary. Let $y_i(t)$, the data on channel i at time t, be written as a temporal convolution of a set of basis functions $\phi_{ij}(t)$, with i and j denoting channel and basis function, respectively,

$$y_i(t) = \sum_j \phi_{ij}(t) * x_j(t) + \epsilon_i(t) \tag{1}$$

with $\epsilon_i(t) \sim \mathcal{N}(0, \sigma_n)$ small, uncorrelated gaussian noise on each channel. This model is illustrated in Fig. 1. To estimate model parameters, the data is assumed to be an identically distributed, independent ensemble of length T time samples with C channels $\mathbf{Y} = \{\mathbf{y}^{(i)}\}_{i=1\ldots D}$ with $\mathbf{y}^{(i)} \in \mathbb{R}^{C\times T}$. The log-likelihood of the model is,

$$\begin{aligned}\mathcal{L}(\mathbf{\Phi}, \sigma_n) &= \log p(\mathbf{Y}|\mathbf{\Phi}, \sigma_n, \lambda)\\ &= \sum_{i=1}^{D} \log p(\mathbf{y}^{(i)}|\mathbf{\Phi}, \sigma_n, \lambda)\\ &= \sum_{i=1}^{D} \log \int \mathrm{d}\mathbf{x}\, p(\mathbf{y}^{(i)}|\mathbf{x}, \mathbf{\Phi}, \sigma_n)\, p(\mathbf{x}|\lambda)\end{aligned}$$

where,

$$p(\mathbf{y}|\mathbf{x}, \mathbf{\Phi}, \sigma_n) \propto \exp\left(-\frac{1}{2\sigma_n^2}\sum_{t=1}^{T}\|\mathbf{y}_t - \sum_\tau \mathbf{\Phi}_\tau \mathbf{x}_{t-\tau}\|^2\right)$$

with $\mathbf{\Phi}_\tau \in \mathbb{R}^{C\times N}$, $\mathbf{x}_\tau \in \mathbb{R}^N$ with N the number of basis functions. The sparse prior on the coefficients $\mathbf{x}$ is parametrized with λ. The goal of learning is to maximize the likelihood $\mathcal{L}(\mathbf{\Phi}, \sigma_n)$. The derivative of $\mathcal{L}(\mathbf{\Phi}, \sigma_n)$ is taken with respect to the model parameter $\mathbf{\Phi}_\alpha$,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{\Phi}_\alpha} \propto -\sum_{i=1}^{D}\int \mathrm{d}\mathbf{x}\, p(\mathbf{x}|\mathbf{y}^{(i)}, \mathbf{\Phi}, \sigma_n, \lambda)\sum_{t=1}^{T}(\mathbf{y}_t - \sum_\tau \mathbf{\Phi}_\tau \mathbf{x}_{t-\tau})\,\mathbf{x}_{t-\alpha}^T$$

A number of ways exist to estimate the intractable integral in this expression, including a Laplace approximation [**?**] and Hamiltonian Monte Carlo sampling [**?**]. The simplest approach, taken here, is to assume the posterior distribution $p(\mathbf{x}|\mathbf{y}^{(i)}, \mathbf{\Phi}, \sigma_n, \lambda)$ is sufficiently peaked and to take one sample at its mode [?], that is,

$$\begin{aligned}\mathbf{x}^{(i)} &= \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}^{(i)}, \mathbf{\Phi}, \sigma_n, \lambda) \\ &= \arg\max_{\mathbf{x}} \log p(\mathbf{y}^{(i)}, \mathbf{x}|\mathbf{\Phi}, \sigma_n, \lambda) \\ &= \arg\min_{\mathbf{x}} \left(\frac{1}{2\sigma_n^2} \sum_{t=1}^{T} \|\mathbf{y}_t - \sum_{\tau} \mathbf{\Phi}_\tau \mathbf{x}_{t-\tau}\|^2 - \log p(\mathbf{x}|\lambda) \right) \end{aligned} \tag{2}$$

Model likelihood was maximized using an alternating scheme where $\mathbf{x}^{(i)}$ were inferred and then used to update $\mathcal{L}(\mathbf{\Phi}, \sigma_n)$ [?]. An additional simplifying assumption was made to take the parameter of the gaussian noise σ_n as given, though a prior could be imposed on it and estimated along with $\mathbf{\Phi}$. In practice, an appropriate σ_n is approximated from descriptive statistics of the data. As the data is practically infinite, only a small sample of $\mathbf{y}^{(i)}$ was chosen in each step. Additionally, the model has a degeneracy due to the sparse prior $p(\mathbf{x}|\lambda)$ shrinking coefficients to zero, causing the norm of $\mathbf{\Phi}$ to grow without bound. Therefore a convex constraint was imposed such that,

$$\sum_{i=1}^{C} \sum_{\tau=1}^{P} \phi_{ij\tau}^2 \leq 1$$

where P are the number of time taps in the basis functions. The constraint makes the learning update of $\mathbf{\Phi}$ a quadratically constrained quadratic program (QCQP) [?]. In practice, however, this problem was solved by making a small stochastic gradient learning step and renormalizing the basis functions on each iteration. A full QCQP solver was implemented using a convolutional adaptation of the method proposed in [?], but found that for neural datasets, the algorithm was prone to get stuck in local minima. During learning, the basis functions were recentered gradually over many iterations. This reflected an implicit prior that the basis functions should be temporally localized.

1.1 Matching pursuit inference

The nature of the spiking and LFP datasets required the inference problem 2 to be solved the using a different strategy for each case. For the spike dataset, which is made up primarily of highly sparse spike activity separated in time as well as in space among the channels and mixed with approximately gaussian noise, a greedy algorithm, matching pursuit [?, ?, ?], was chosen. This algorithm is efficient for a high degree of assumed sparsity when the basis functions can be assumed to be relatively incoherent. Its goal is to represent the data with at most k basis functions,

$$\mathbf{x}^* = \arg\min_{\|\mathbf{x}\|_0 \leq k} \frac{1}{2\sigma_n^2} \sum_{t=1}^{T} \|\mathbf{y}_t - \sum_{\tau} \mathbf{\Phi}_\tau \mathbf{x}_{t-\tau}\|^2$$

where $\|\mathbf{x}\|_0$ denotes the number of non-zero elements of $\mathbf{x}$. For arbitrary $\mathbf{\Phi}$, this problem is NP-hard [?]. Matching pursuit is a greedy strategy that finds an approximate solution and overcomes this combinatorial complexity. Additionally, to make the learned basis functions and coefficients more physiologically interpretable, the coefficients were forced to be non-negative.

1.2 L_1-regularized quasi-Newton inference

For the LFP dataset, an L_1-regularized method was used to induce sparsity on the coefficients. The LFP dataset is sampled at a lower rate and is hence considerably smaller than the spike dataset, reducing requirements for computational efficiency by a large factor. An L_1 method performed better at learning

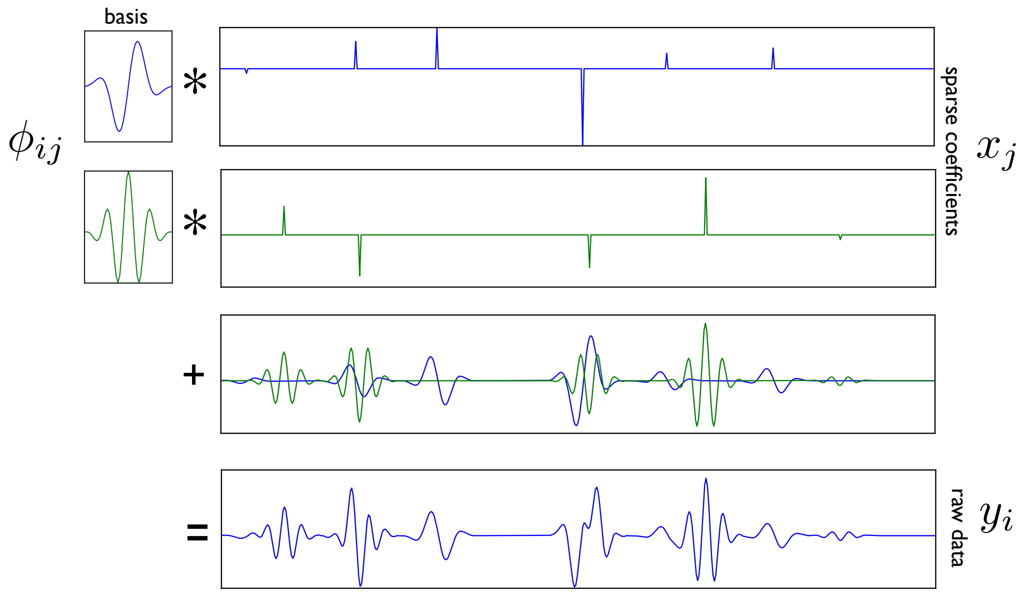

Figure 1. Convolutional sparse coding. A convolutional sparse coding model is a linear generative model where data is represented as a sum of a convolution of a set of bases with coefficients which are expected to be sparse and independent in both time and channel. This schematic shows how to reconstruct a portion of one channel of data at bottom using two basis elements at left convolved with corresponding sparse coefficients at right and summed. Estimation of basis functions ϕ_{ij} and coefficients x_j (Eq. 1) are called 'learning' and 'inference', respectively. Bases ϕ_{ij} are learned for spiking and LFP datasets separately for each recording penetration and coefficients are inferred for the entire recordings. This new representation has many desirable properties.

in a less sparse regime where basis functions were more coherent and 'explaining away' was more critical. Briefly, the prior on coefficients was assumed to be exponentially distributed,

$$p(\mathbf{x}|\lambda) \propto e^{-\lambda\|\mathbf{x}\|_1}$$

giving the following convex minimization problem for inference,

$$\mathbf{x}^* = \arg\min_{\mathbf{x}>0} \frac{1}{2\sigma_n^2} \sum_{t=1}^{T} \|\mathbf{y}_t - \sum_{\tau} \mathbf{\Phi}_\tau \mathbf{x}_{t-\tau}\|_2^2 + \lambda\|\mathbf{x}\|_1$$

This objective is closely related to the Lasso, which has been intensively studied [**?**, **?**] and for which an abundance of methods exist. Despite its widespread use as a feature selection method, it is important to note that an L_1 regularizer has a deficiency. If the data is indeed generated by a Laplacian distribution, its order statistics are not sufficiently sparse to guarantee recovery [**?**]. Additionally, both inference methods used are not causal, and coefficients inferred for a given time can be affected by data in the future. In contrast, state-space models such as Kalman filters and hidden Markov models are causal by design, though operations such as smoothing are inherently acausal. Creating a causal inference algorithm in this setting is an open problem that is the subject of future work.

2 Implementation

For convolutional matching pursuit, the algorithm was implemented in `python` using the `numpy` [?] library. For L_1-regularized inference, a method [?] based on the widely used l-BFGS quasi-Newton algorithm [?] was chosen, which uses a particular choice of sub-gradient whenever the optimization attempts to cross from one octant to another. The advantages of this algorithm over the many others is that it does not require computing the Gram matrix, only requires an objective and gradient to be defined, is numerically stable for large numbers of parameters, converges quickly to an approximate solution, and can be efficiently run in parallel on multiple cores of a processor as it uses only BLAS level 1 operations.

Three versions of this algorithm were implemented. The first was a `cython` [?, ?] wrapper of the `C++` library `liblbfgs` [?], which explicitly implements all linear algebra with SSE2 instructions. Next, a version in `cython` was implemented that could handle a vector of L_1 regularization parameters λ, a non-negative constraint on the coefficients, and would run more efficiently on multicore architectures through its use of vendor linear algebra libraries. Lastly, a version in `cython` and `pycuda` [?] was implemented to run fully on the NVIDA GPU avoiding all host to GPU transfers during the optimization. In this implementation, the convolutions in the objective were implemented both as a bank of 2-D FFTs as well as a single 3-D FFT. The 3-D FFT method, despite its theoretical inefficiency in this case, gave an approximately 30x speed-up versus computing the convolutions in the time domain on the CPU. The 2-D FFT bank gave only a 6x speed-up, most likely due to the GPU being constrained to computing one 2-D FFT at a time. Array slicing was implemented with 1-D texture maps and all norms, projections, and reductions were custom written to take advantage of the parallelism in the GPU.

The full learning algorithm was parallelized using the Message Passing Interface (MPI) in `python` using `mpi4py` [?]. On each learning iteration, the root node sampled the data from disk and distributed this data amongst the nodes using an MPI `Scatter`. All nodes then performed inference using one of the algorithms described above and the results were reduced on the root node with an MPI `Gather`. A learning step was taken and the basis was then MPI `Broadcast` to all nodes. This framework allowed the learning algorithm to exploit parallelism on a single multicore CPU with or without a GPU, a cluster of multicore CPU's, as well as a hybrid cluster of CPU's and GPU's.

2.1 Parallel inference of coefficients for a full dataset

After a basis was learned for a spike or LFP dataset, coefficients were inferred for the whole dataset in a chunk-wise parallel fashion (Fig. 2). Given N parallel computational nodes, the data was divided into N large chunks. Within each chunk, inference was performed in sequence on blocks of time length T with $T >> P$, where P is the number of time taps in the learned basis $\mathbf{\Phi}$. Blocking was used at it is computationally impractical to perform inference on arbitrarily large time segments. A block of length T yielded coefficients $\mathbf{x}$ of time length $T + P - 1$. After one block was completed, all except a $P - 1$ length of its tail was written to disk. The $2P - 2$ tail portion of this block was used for initializing the next block. For the new block, the first $P - 1$ coefficients were held fixed whereas the next $P - 1$ coefficients were used to warm start the inference in the case of L_1 or were set to zero for matching pursuit.

The non-linear nature of inference raised the possibility that the implementation would have blocking artifacts. However, inference was tested by blocking over a several block region as well as inferring coefficients on the whole region with results in good agreement. This is due to the high degree of sparsity used as well as a side-effect of the convolutional formulation of the objective, that coefficients on the $P - 1$ borders received less derivative information and were more likely to remain at zero.

Parallel blocks were stitched by performing inference on adjoining regions of length $3P - 2$, with only the inner $P - 2$ portion being free to be modified by the optimization. The parallel blocking allowed the inference to scale almost linearly and to compute coefficients for a recording session with approximately the same order of time as the experiment itself. The datasets with metadata were written to disk using a generic gzip level 4 compressed HDF5 format which afforded a 20-100 fold compression over the original

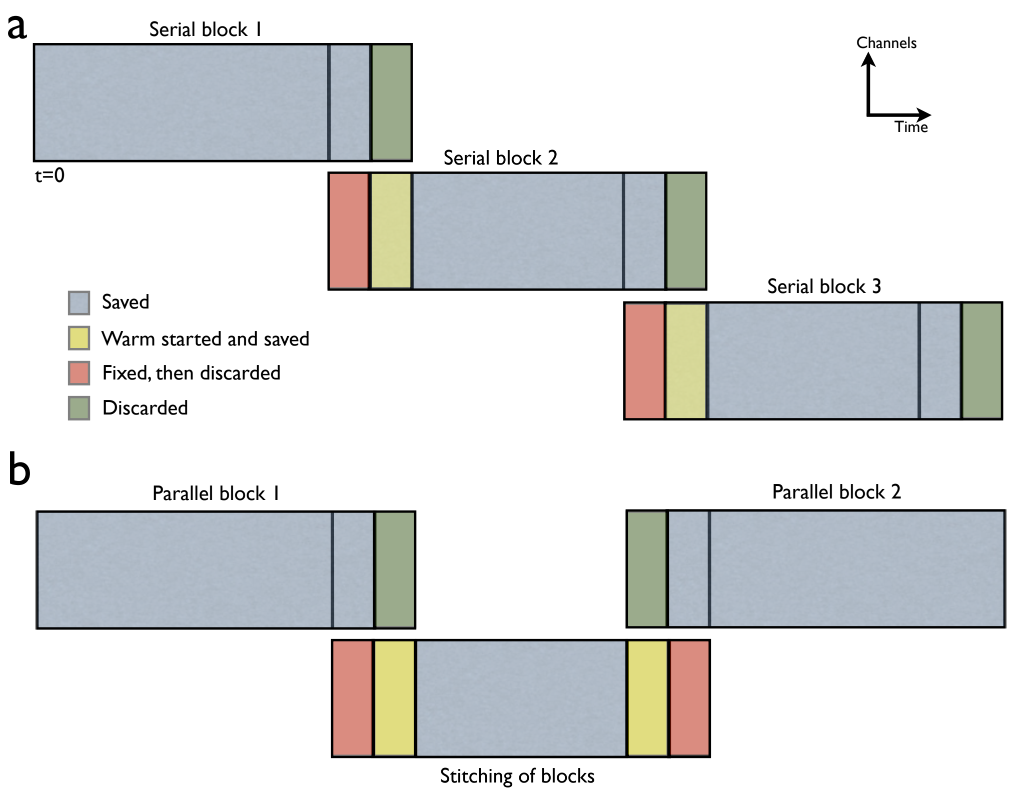

Figure 2. Inferring coefficients over a full dataset. (a) Each parallel chunk of data is blocked into tractable portions and processed serially by using tail portions of previous blocks to warm-start the following block. **(b)** When each parallel chunk is completed, the regions between chunks are processed to stitch chunks into one long set of coefficients. The stitched portion is warm-started from it's adjoining blocks.

dataset, depending primarily on the level of coefficient sparsity.