Efficient Gibbs Sampling for Fields of Experts Image Models

Jonas Arnfred

June 8, 2012

1 Introduction

For most machine learning algorithms there are two main criteria for the success of new improvements: The quality and the efficiency of the algorithm. An algorithm with a high quality output will push the boundaries of what's possible, but if it isn't fast the applications to real life situations become drastically limited. Most people do not have days to wait for a simulation to be done, nor do they have a cluster of nodes to run the algorithm on, but what's more, many applications benefit from a close to real-time output or work with input sizes drastically larger than the test examples used to demonstrate the algorithms.

The Field of Experts framework proposed in [3] attempts to address both sides of the problem by simplifying export models to use convolution filters making it feasible to sample images from a generative framework. The particular denoising algorithm proposed in [4] makes use of these advantages to a large part, but in efforts to prove the worth of the Fields of Experts algorithm in terms of quality, they sacrifice efficiency to achieve the best possible result. This leaves it an open question whether Fields of Experts has the potential to reach a good compromise between speed and quality.

In this paper I try to explore how much we need to sacrifice in terms of quality in order to gain extra efficiency. Due to the limited scope of my project, this exploration is done entirely within the frame of an already learned image prior taken from [4]. In the first part of my paper I analyse the main efficiency bottleneck of the algorithm and discuss what variables I can adjust in the hopes of seeing an increase in speed. In the second part I address this discussion and provide data for the behaviour of Field of Experts given adjustments of these variables.

Due to the exploratory nature of the project, I present this data not as a final conclusive material on the matter, but more as an indicator of the directions it might be worthwhile to look in to down the road.

2 Method

To analyse where the performance bottleneck is located in the Field of Experts, we need to take a look at how the prior is defined. In [3] and [4] the framework is defined as follows: Given an image of size $n_x n_y = n$, $\mathbf{u} \in \mathbb{R}^n$, the probability density of an image $\mathbf{u}$ is:

$$p(\mathbf{u};\Theta) = \frac{1}{Z(\Theta)} e^{-\epsilon||\mathbf{u}||^2/2} \prod_{r=1}^{R} \prod_{i=1}^{n} t_r(s_{ri}) \tag{1}$$

where s_{ri} is the circular convolution of u with the filter f_r, $r = 1, \ldots, R$, Θ is a collection of model parameters $(f_r, \alpha_r, \sigma_r)$ and Z is the partition function. The potentials $t_r(s)$ used are mixtures of Gaussian defined as:

$$t_r(s) = \sum_{j=1}^{J} \alpha_{rj} \mathcal{N}(s|o, \sigma_{rj}^2) \tag{2}$$

where J is the number of mixtures used in the model, α_{rj} is the scale for the jth gaussian and σ_{rj}^2 the variance. To sample from this density [4] propose using an auxiliary-variable Gibbs sampler which introduces the indicator variables $z_{ri} \in 1, \ldots, J^R$ that chooses the component of the mixture selected. This way $t_r(s_{ri})$ given z_{ri} is equivalent to a gaussian and much easier to sample from. Using Gibbs sampling we can alternate between sampling $\mathbf{z}^{(t+1)} \sim p(\mathbf{z}|\mathbf{u};\Theta)$ and $\mathbf{u}^{(t+1)} \sim p(\mathbf{u}|\mathbf{z};\Theta)$ where t is the current iteration. With a little restructuring of (1) and (2) the conditionals for $\mathbf{u}$ and $\mathbf{z}$ are given by

$$p(\mathbf{z}_{ri}|\mathbf{u};\Theta) \propto \alpha_{rz_{ri}} \times \mathcal{N}(s|o, \sigma_{rz_{ri}}^2) \tag{3}$$

$$p(\mathbf{u}|\mathbf{z};\Theta) \propto \mathcal{N}\left(\mathbf{u}; 0, \left(\epsilon\mathbf{I} + \sum_{i=1}^{R} B_r^T diag(1/\sigma_{rz_{ri}}^2) B_r\right)^{-1}\right) \tag{4}$$

where B_r are the matrices that correspond to a convolution of $\mathbf{u}$ with the filter $\mathbf{f}_r$. To sample from $p(\mathbf{u}|\mathbf{z};\Theta)$ we sample $n_0 \sim \mathcal{N}(0, \mathbf{I}) \in \mathbb{R}^{Rn}$. If we denote $A = \epsilon\mathbf{I} + \sum_{i=1}^{R} B_r^T diag(1/\sigma_{rz_{ri}}^2) B_r$ and $n_1 = \sum_{i=1}^{R} B_r^T diag(1/\sigma_{rz_{ri}}^2)$ we can then sample $u = A^{-1} n_1 \sim \mathcal{N}(0, A^{-1})$. Given the noisy image $\mathbf{y}$ we can to sample the posterior for image denoising by

$$p(\mathbf{u}|\mathbf{y},\mathbf{z};\Theta) \propto p(\mathbf{y}|\mathbf{u}) \cdot p(\mathbf{u}|\mathbf{z};\Theta) \tag{5}$$

$$\propto \mathcal{N}\left(\mathbf{u};\, \tilde{A}y/\sigma^2,\, \tilde{A}\right) \tag{6}$$

where $\tilde{A}$ is defined as $(\mathbf{I}/\sigma^2 + A^{-1})^{-1}$.

This means that in order to sample from the posterior distribution we need to solve two linear systems: $\mathbf{u} = \tilde{A}^{-1} \cdot n_1$ and $\mathbf{u}_\mu = \tilde{A}^{-1} \cdot \mathbf{y}/\sigma^2$. Since these two systems are solved for every iteration of the Gibbs sampling, it's important that they are solved fast for the algorithm to be efficient.

2.1 Optimizing for Conjugate Gradient

The denoise algorithm proposed by [4] uses choleski decomposition to solve the systems which doesn't scale well to images and doesn't provide much room for optimizations. The first step of speeding up the algorithm was thus to implement a Gibbs sampler that uses the conjugate gradient algorithm. This allowed for several possible ways to increase the speed of the algorithm: Preconditioning, lower error tolerance, resizing of the scales ($\sigma^2_{rz_{ri}}$) and finally eliminating the more extreme values of the scales.

Since the goal of the implemented denoising algorithm is not to achieve superior denoising results, but rather serve as a tool for analysis, it differs from the implementation specified in [4] on one account. Instead of running four Gibbs samplers concurrently, only one sampler is run. This makes it more difficult to estimate when to stop the algorithm, so instead of using the variance in between the four concurrent samplers to estimate when to stop, I let the Gibbs sampler run for a set amount of conjugate gradient iterations. This facilitates analysis since the accumulated amount of conjugate gradient iterations provides a common atomic unit across different runs.

In order to speed up conjugate gradient we can use a preconditioner to improve the condition number of A. If are trying to solve the system $Ax = b$ and have a matrix M for which $\kappa(M^{-1}A) \ll \kappa(A)$ then we can usually solve $Ax = b$ faster by solving $M^{-1}Ax = M^{-1}b$ instead [2]. The problem lies in finding an M which is invertible and in fact improves the condition number.

Alternatively (and much easier) we can increase how small the error must be before we stop iterating. This in turn means that we are accepting solutions that are further from the true solution of the system which might negatively impact the quality of the denoising.

Instead of looking at the algorithm, it might prove useful to change parameters to create equations that are faster to solve. One way to do this is by changing the scales used in the gaussian mixture. In [4] a fixed number of 15 scales is specified as $s = exp(0, \pm 1, \ldots, \pm 5, \pm 7, \pm 9)$. To scale s I introduce a factor $p \in \{1, .95, .9, \ldots, 0.1\}$ in $s = exp(\ldots)^p$. This ensures that the more extreme scales are altered more than those in between.

3 Results

psnr $\approx$ 22 | psnr $\approx$ 24 | psnr $\approx$ 26 | psnr $\approx$ 28 | psnr $\approx$ 30

Figure 1: Examples of images at different Signal to Noise ratio

To test how the aforementioned adjustments affect the performance of the denoising algorithm I calculate the signal to noise ratio (PSNR) for each iteration of conjugate gradient. All the data shown has been collected for three different values of sigma (0.05, 0.1 and 0.15) in relation to the images defined on [0,1]. Most figures show only data for one value of sigma except for cases where there are remarkable differences in behaviour. For every iteration of the Gibbs sampler we are solving two linear systems, but since they aren't synchronized it would be confusing to show both in the same plot. For considerations of consistency only the iterations of the system solving for $\mathbf{u}_\mu$ will be shown in the figures, except for cases where differences in behaviors between the two systems warrant a special mention. $\mathbf{u}_\mu$ was chosen because the Rao-Blackwellised Gibbs sampling samples the denoised image exclusively from $\mathbf{u}_\mu$.

In order to evaluate what different levels of signal to noise ratio translates to in image quality, figure 1 showcases some examples for values typically encountered during the experiments. To compare the signal to noise ratio of an imaged with added gaussian noise at a standard deviation of 0.05, 0.1 or 0.15 (for an image defined on [0,1]) is respectively 26.3, 20.0 and 16.5 approximately.

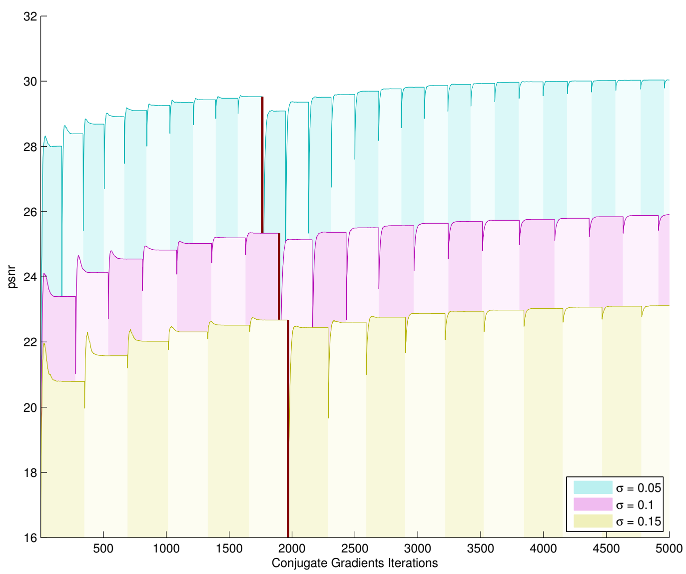

Figure 2: Variation of Sigma

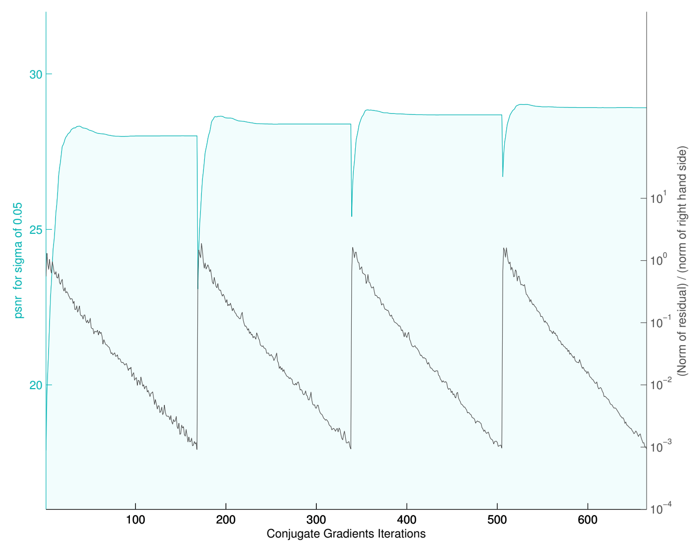

Figure 3: Four iterations for sigma of 0.05 together with the residual norm

Figure 2 shows a run with no modified parameters for the same three values of sigma. The area underneath each line has been shaded in alternating colors to indicate the duration of each conjugate gradient run. At the burn-off where the initial samples are discarded, the plot is marked with a red vertical line. The burn-off happens when a conjugate gradient run finishes and has surpassed a certain amount of iterations. This explains why it doesn't happen at exactly

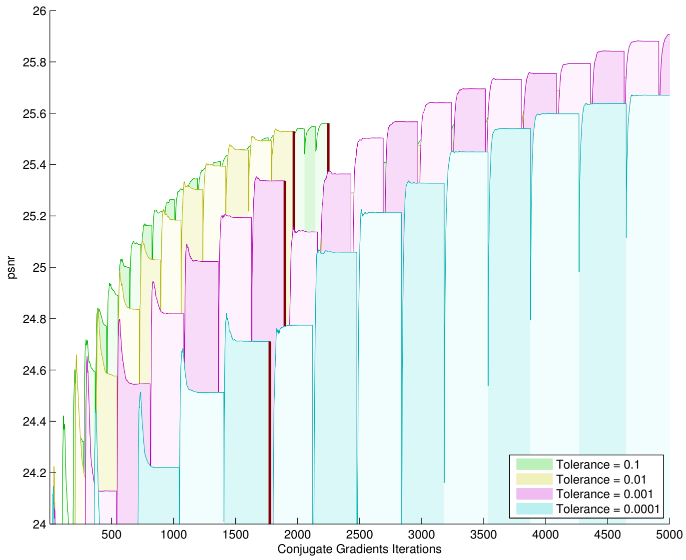

Figure 4: Variation of Tolerance at sigma = 0.1

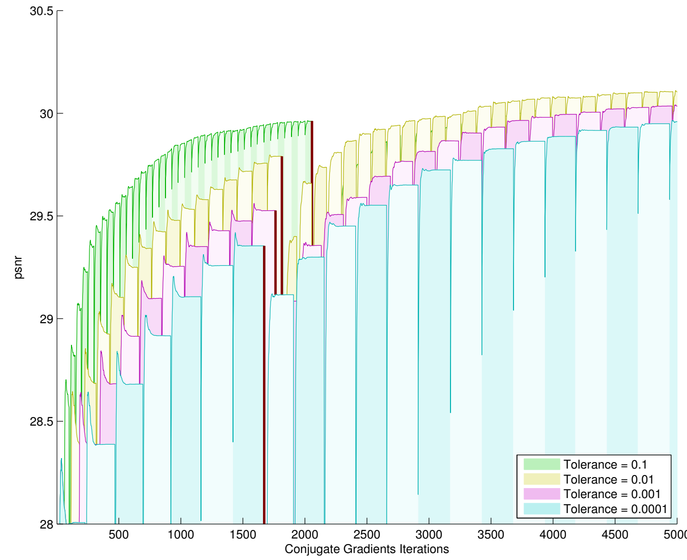

Figure 5: Variation of Tolerance at sigma = 0.05

the same point. For all experiments the burn-off was set to happen after the total sum (both systems of linear equations) of iterations had surpassed 3000. The run would be terminated after 10000 total iterations. This means that $\mathbf{u}_\mu$ has been calculated for around 5000 iterations. For all overview plots the x-axis goes from 1 to 5000 iterations. The y-axis on the other hand varies between plots. From the plot we can observe how the values of σ influence the signal to noise ratio as well as the amount of iterations for each conjugate gradient run. For $\sigma = 0.05$ the mean amount of iterations is FILLME, while it is respectively FILLME and FILLME for $\sigma = 0.1$ and $\sigma = 0.15$.

If we take a closer look in figure 3 we can see how each iteration of conjugate gradient starts out with a low signal to noise ratio but quickly reaches a maximum before it settles. The exponential decay of the norm of the residual is plotted on top of the graph, showing that no real change in the signal to noise ratio takes place after the norm of the residual passes below 0.1.

The effects of lowering the tolerance can be seen in figure 4 and 5. For $\sigma = 0.1$ in figure 4 the higher tolerance improves the steepness of the curve, but comes at a cost. Ultimately the signal to noise ratio of the more exact cases with tolerance = 0.001 or 0.0001 surpasses the higher tolerances. For the smallest standard deviation ($\sigma = 0.05$) this isn't necessarily the case as the lower tolerances still maintain a better FILLME (add percent) signal to noise ratio after 5000 iterations.

Figure 6 depicts effect of removing the more extreme scales. As expected the conjugate gradient iterations become a lot quicker, but it comes at the cost of decreased performance. A similar behavior can be seen in figure 7. Here the

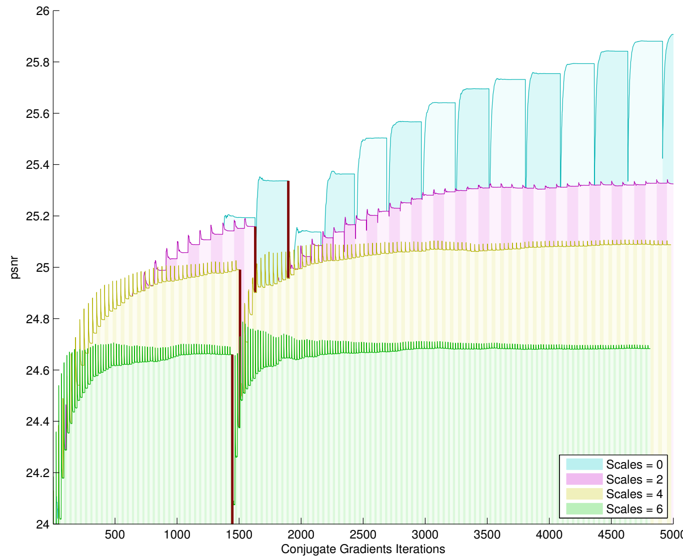

Figure 6: Removal of scales at sigma = 0.1

scales have been removed two by two, greatly improving the speed of conjugate gradient but at the cost of quality. The more scales removed, the earlier the plots converge toward a constant value. In figure 8 a more detailed account of the same data can be seen. A grid is made by plotting the maximum signal to noise ratio for each interval of a 100 iterations, as the scale factor is varied from 1 to 0.1 in steps of 0.05.

4 Discussion

To interpret the results, it's important to keep in mind that while the scales were hand picked, the model was learned and optimized for these scales. This means that removing scales or changing them comes with the added cost of running the model in an environment it hasn't been optimized for. It might not be impossible to learn another model with smaller scales that perform as well as the current even if the results don't show greater rates for reduced scales. Furthermore these results are based on simulations run on a single image and as such prone to fluctuations and issues that might not show up were more images considered.

In this light we can learn from the results based on modifying the scales that the conjugate gradient algorithm does indeed converge faster when the scales are less extreme, a useful lesson if we are to learn our own scales later on.

For applications where speed is more important than a few percents gain in signal to noise ratio, adjusting the tolerance of the conjugate gradient algorithm can prove useful. Higher tolerance rates have shown to provide faster climbs

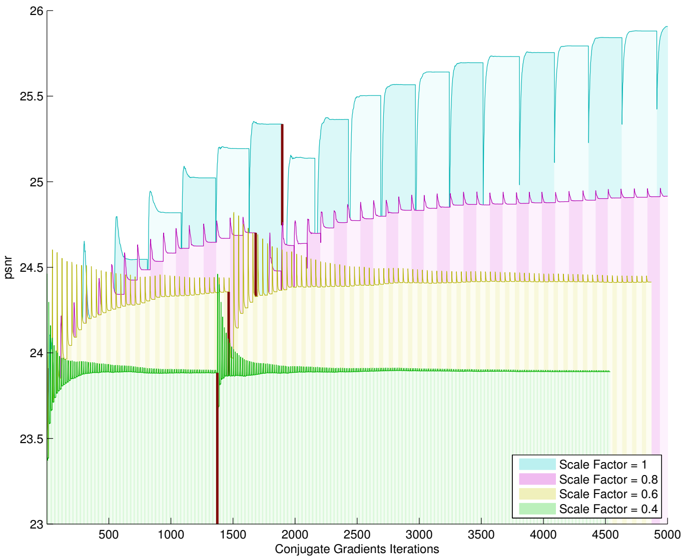

Figure 7: Variation of Scales at sigma = 0.1

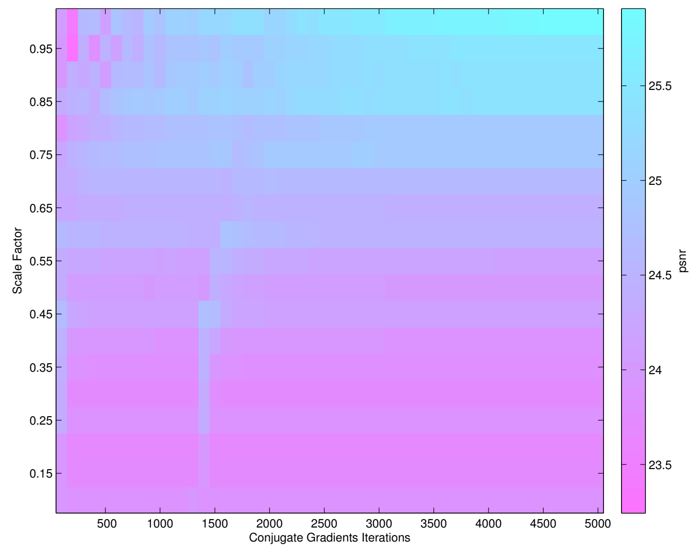

Figure 8: Variation of Scales at sigma = 0.1

but tend to come at the price of a lower global maximum of signal to noise ratio. However for low noise applications the costs associated with increasing the tolerance are reduced. The gains in speed are relatively small though.

In [4] a lot of weight is put on the fact that the prior resembles the long tailed distributions seen in natural images. It would be interesting to see if there is space for compromise between the realistic but slow distributions and their fast but inaccurate counterparts.

5 Conclusion

In order to improve the efficiency of the Fields of Experts framework I implemented a denoising algorithm using framework in combination with the conjugate gradient algorithm. Based on this implementation I demonstrated that adjusting the scales of the auxiliary variable Gibbs sampler to create a prior with a smaller long tail does increase conjugate gradient convergence, but doesn't improve overall performance. Moreover, I showed that adjusting the tolerance of the conjugate gradient algorithm can improve performance, especially for low variances.

References

[1] Matthias Seeger. Efficient inference and learning for field of experts image models. *Internal Memo*, 2012.

[2] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. 1994.

[3] Michael J. Black Stefan Roth. Fields of experts. *International Journal of Computer Vision*, 82(2):205–229, 2009.

[4] Stefan Roth Uwe Schmidt, Qi Gao. A generative perspective on mrfs in low-level vision. *Computer Vision and Pattern Recognition*, 2010.