Automatic Differentiation Variational Inference

Daeyoung Lim

Department of Statistics
Korea University

January 6, 2017

Variational Approximations

- We want to make approximate inferences
- MCMC is exact but takes too long
- blah blah blah... Same old story.

Variational Approximations

- We want to make approximate inferences
- MCMC is exact but takes too long
- blah blah blah... Same old story.

Variational Approximations

- We want to make approximate inferences
- MCMC is exact but takes too long
- blah blah blah... Same old story.

Automatic Differentiation

- Computer doing math
- Same as the pencil-and-paper solution
- Possible with 'ifs' and 'fors'..
- No matter how complicated, every function is a combination of elementary operations (add, subtract, multiply, divide, exponentiate, log, exp, sin, cos, tan, asin, acos, atan, etc...)
- Unlike numerical differention, AD does not produce round-off errors
- Python library `autograd` or C++ library `stan-math` provide AD functionality

Automatic Differentiation

- Computer doing math
- Same as the pencil-and-paper solution
- Possible with 'ifs' and 'fors'..
- No matter how complicated, every function is a combination of elementary operations (add, subtract, multiply, divide, exponentiate, log, exp, sin, cos, tan, asin, acos, atan, etc...)
- Unlike numerical differention, AD does not produce round-off errors
- Python library `autograd` or C++ library `stan-math` provide AD functionality

Automatic Differentiation

- Computer doing math
- Same as the pencil-and-paper solution
- Possible with ‘ifs’ and ‘fors’..
- No matter how complicated, every function is a combination of elementary operations (add, subtract, multiply, divide, exponentiate, log, exp, sin, cos, tan, asin, acos, atan, etc...)
- Unlike numerical differention, AD does not produce round-off errors
- Python library `autograd` or C++ library `stan-math` provide AD functionality

Automatic Differentiation

- Computer doing math
- Same as the pencil-and-paper solution
- Possible with 'ifs' and 'fors'..
- No matter how complicated, every function is a combination of elementary operations (add, subtract, multiply, divide, exponentiate, log, exp, sin, cos, tan, asin, acos, atan, etc...)
- Unlike numerical differention, AD does not produce round-off errors
- Python library `autograd` or C++ library `stan-math` provide AD functionality

Automatic Differentiation

- Computer doing math
- Same as the pencil-and-paper solution
- Possible with 'ifs' and 'fors'..
- No matter how complicated, every function is a combination of elementary operations (add, subtract, multiply, divide, exponentiate, log, exp, sin, cos, tan, asin, acos, atan, etc...)
- Unlike numerical differention, AD does not produce round-off errors
- Python library `autograd` or C++ library `stan-math` provide AD functionality

Automatic Differentiation

- Computer doing math
- Same as the pencil-and-paper solution
- Possible with 'ifs' and 'fors'..
- No matter how complicated, every function is a combination of elementary operations (add, subtract, multiply, divide, exponentiate, log, exp, sin, cos, tan, asin, acos, atan, etc...)
- Unlike numerical differention, AD does not produce round-off errors
- Python library `autograd` or C++ library `stan-math` provide AD functionality

ADVI

- With a generic formula, we can use AD to get the gradients for VI
- The idea is based on Gaussian approximation (or any transformable distribution with a standard version like t-dist, logistic-dist)
- The support should be the whole real line and the model should be differentiable
- If any of the parameters does not satisfy 'full real line' condition, we transform it
- For example, $\sigma^2 > 0$ in linear regression

$$\alpha = \log\left(e^{\sigma^2} - 1\right) \tag{1}$$

ADVI

- With a generic formula, we can use AD to get the gradients for VI
- The idea is based on Gaussian approximation (or any transformable distribution with a standard version like t-dist, logistic-dist)
- The support should be the whole real line and the model should be differentiable
- If any of the parameters does not satisfy 'full real line' condition, we transform it
- For example, $\sigma^2 > 0$ in linear regression

$$\alpha = \log\left(e^{\sigma^2} - 1\right) \tag{1}$$

ADVI

- With a generic formula, we can use AD to get the gradients for VI
- The idea is based on Gaussian approximation (or any transformable distribution with a standard version like t-dist, logistic-dist)
- The support should be the whole real line and the model should be differentiable
- If any of the parameters does not satisfy 'full real line' condition, we transform it
- For example, $\sigma^2 > 0$ in linear regression

$$\alpha = \log\left(e^{\sigma^2} - 1\right) \tag{1}$$

ADVI

- With a generic formula, we can use AD to get the gradients for VI
- The idea is based on Gaussian approximation (or any transformable distribution with a standard version like t-dist, logistic-dist)
- The support should be the whole real line and the model should be differentiable
- If any of the parameters does not satisfy 'full real line' condition, we transform it
- For example, $\sigma^2 > 0$ in linear regression

$$\alpha = \log\left(e^{\sigma^2} - 1\right) \tag{1}$$

ADVI

- With a generic formula, we can use AD to get the gradients for VI
- The idea is based on Gaussian approximation (or any transformable distribution with a standard version like t-dist, logistic-dist)
- The support should be the whole real line and the model should be differentiable
- If any of the parameters does not satisfy ‘full real line’ condition, we transform it
- For example, $\sigma^2 > 0$ in linear regression

$$\alpha = \log\left(e^{\sigma^2} - 1\right) \tag{1}$$

ADVI: Idea

- We will always think the posterior is a Gaussian distribution

$$q(\theta) = \mathcal{N}\left(\mu, LL'\right) \tag{2}$$

- Then we need to tune the mean vector and covariance matrix to best approximate the true posterior distribution
- As always, minimize the reverse KL divergence

ADVI: Idea

- We will always think the posterior is a Gaussian distribution

$$q(\theta) = \mathcal{N}\left(\mu, LL'\right) \quad (2)$$

- Then we need to tune the mean vector and covariance matrix to best approximate the true posterior distribution
- As always, minimize the reverse KL divergence

ADVI: Idea

- We will always think the posterior is a Gaussian distribution

$$q(\theta) = \mathcal{N}\left(\mu, LL'\right) \tag{2}$$

- Then we need to tune the mean vector and covariance matrix to best approximate the true posterior distribution
- As always, minimize the reverse KL divergence

KL divergence

- Can be thought of as the semi-measure of distance between distributions (not symmetric, no triangular inequality)
- In fact, in VI, a symmetric measure wouldn't make sense because we need to measure how far our approximation is from the true posterior. There is a direction actually.
- $-\mathrm{E}\left[\log p(X)\right]$ is the 'entropy'
  - $=$ the information gained by observing a random event(data)
  - $=$ uncertainty contained in the random event(data)
  - $=$ the expected number of 'nats' or 'bits' needed to encode the information

KL divergence

- Can be thought of as the semi-measure of distance between distributions (not symmetric, no triangular inequality)
- In fact, in VI, a symmetric measure wouldn't make sense because we need to measure how far our approximation is from the true posterior. There is a direction actually.
- $-\mathrm{E}\left[\log p(X)\right]$ is the 'entropy'
  - $=$ the information gained by observing a random event(data)
  - $=$ uncertainty contained in the random event(data)
  - $=$ the expected number of 'nats' or 'bits' needed to encode the information

KL divergence

- Can be thought of as the semi-measure of distance between distributions (not symmetric, no triangular inequality)
- In fact, in VI, a symmetric measure wouldn't make sense because we need to measure how far our approximation is from the true posterior. There is a direction actually.
- $-\mathsf{E}\left[\log p(X)\right]$ is the 'entropy'
  - $=$ the information gained by observing a random event(data)
  - $=$ uncertainty contained in the random event(data)
  - $=$ the expected number of 'nats' or 'bits' needed to encode the information

KL divergence

- Also called the 'relative entropy' or 'information gain'
- $\mathrm{KL}(q||p) = \mathrm{E}_q\left[\log q(X)\right] - \mathrm{E}_q\left[\log p(X)\right]$
- Measures the amount of information gained by using p instead of q. Thus, q is regarded as true.
- Determining a distribution by minimizing KL divergence does not yield degenerate solutions because the negative entropy term $\mathrm{E}_q\left[\log q(X)\right]$ functions as a regularizer

KL divergence

- Also called the ‘relative entropy’ or ‘information gain’
- $\mathsf{KL}(q||p) = \mathsf{E}_q\left[\log q(X)\right] - \mathsf{E}_q\left[\log p(X)\right]$
- Measures the amount of information gained by using p instead of q. Thus, q is regarded as true.
- Determining a distribution by minimizing KL divergence does not yield degenerate solutions because the negative entropy term $\mathsf{E}_q\left[\log q(X)\right]$ functions as a regularizer

KL divergence

- Also called the 'relative entropy' or 'information gain'
- $\mathsf{KL}(q||p) = \mathsf{E}_q[\log q(X)] - \mathsf{E}_q[\log p(X)]$
- Measures the amount of information gained by using p instead of q. Thus, q is regarded as true.
- Determining a distribution by minimizing KL divergence does not yield degenerate solutions because the negative entropy term $\mathsf{E}_q[\log q(X)]$ functions as a regularizer

KL divergence

- Also called the 'relative entropy' or 'information gain'
- $\mathsf{KL}(q||p) = \mathsf{E}_q\left[\log q(X)\right] - \mathsf{E}_q\left[\log p(X)\right]$
- Measures the amount of information gained by using p instead of q. Thus, q is regarded as true.
- Determining a distribution by minimizing KL divergence does not yield degenerate solutions because the negative entropy term $\mathsf{E}_q\left[\log q(X)\right]$ functions as a regularizer

Stochastic Optimization

- To generalize VI, we need a handier way of getting the gradient (We can't handcraft the updating algorithm every time we have a different model. Or we just don't want to...)
- Stochasticity comes from two sources:
  - Minibatch: not using the entire data (for scalability)
  - Monte Carlo estimation of the gradient (for generality)
- The gradient is expressed as follows

$$\nabla_\mu \mathcal{L} = \mathrm{E}_{\mathcal{N}(\eta)} \left[\nabla_\theta \log p(x,\theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log |\det J_{T^{-1}}(\zeta)| \right] \quad (3)$$

$$\nabla_L \mathcal{L} = \mathrm{E}_{\mathcal{N}(\eta)} \left[\left(\nabla_\theta \log p(x,\theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log |\det J_{T^{-1}}(\zeta)| \right) \eta' \right] \quad (4)$$

$$+ \left(L^{-1}\right)' \quad (5)$$

Stochastic Optimization

- To generalize VI, we need a handier way of getting the gradient (We can't handcraft the updating algorithm every time we have a different model. Or we just don't want to...)
- Stochasticity comes from two sources:
  - Minibatch: not using the entire data (for scalability)
  - Monte Carlo estimation of the gradient (for generality)
- The gradient is expressed as follows

$$\nabla_\mu \mathcal{L} = \mathsf{E}_{\mathcal{N}(\eta)} \left[\nabla_\theta \log p(x,\theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log |\det J_{T^{-1}}(\zeta)| \right] \tag{3}$$

$$\nabla_L \mathcal{L} = \mathsf{E}_{\mathcal{N}(\eta)} \left[\left(\nabla_\theta \log p(x,\theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log |\det J_{T^{-1}}(\zeta)| \right) \eta' \right] \tag{4}$$

$$+ \left(L^{-1}\right)' \tag{5}$$

Stochastic Optimization

- To generalize VI, we need a handier way of getting the gradient (We can't handcraft the updating algorithm every time we have a different model. Or we just don't want to...)
- Stochasticity comes from two sources:
  - Minibatch: not using the entire data (for scalability)
  - Monte Carlo estimation of the gradient (for generality)
- The gradient is expressed as follows

$$\nabla_\mu \mathcal{L} = \mathsf{E}_{\mathcal{N}(\eta)} \left[\nabla_\theta \log p(x, \theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log |\det J_{T^{-1}}(\zeta)| \right] \tag{3}$$

$$\nabla_L \mathcal{L} = \mathsf{E}_{\mathcal{N}(\eta)} \left[\left(\nabla_\theta \log p(x, \theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log |\det J_{T^{-1}}(\zeta)| \right) \eta' \right] \tag{4}$$

$$+ \left(L^{-1} \right)' \tag{5}$$

Stochastic Optimization (Subsampling)

- If IID data are assumed, log-likelihood becomes summation

$$\log \prod_{i=1}^{n} L(\theta \mid y_i) = \sum_{i=1}^{n} \ell(\theta \mid y_i) \tag{6}$$

- Stochastic optimization with minibatch of size B is tricking yourself into believing that

$$\sum_{i=1}^{n} \ell(\theta \mid y_i) \approx \frac{n}{B} \sum_{i=1}^{B} \ell(\theta \mid y_{b_{(i)}}) \tag{7}$$

randomly choosing $b_{(i)}$ without replacement

- Reduce the step size ρ_t as we go
- Must satisfy 2 conditions

$$\sum_{t=1}^{\infty} \rho_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \rho_t^2 < \infty \tag{8}$$

Stochastic Optimization (Subsampling)

- If IID data are assumed, log-likelihood becomes summation

$$\log \prod_{i=1}^{n} L(\theta \mid y_i) = \sum_{i=1}^{n} \ell(\theta \mid y_i) \tag{6}$$

- Stochastic optimization with minibatch of size B is tricking yourself into believing that

$$\sum_{i=1}^{n} \ell(\theta \mid y_i) \approx \frac{n}{B} \sum_{i=1}^{B} \ell(\theta \mid y_{b_{(i)}}) \tag{7}$$

randomly choosing $b_{(i)}$ without replacement

- Reduce the step size ρ_t as we go
- Must satisfy 2 conditions

$$\sum_{t=1}^{\infty} \rho_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \rho_t^2 < \infty \tag{8}$$

Stochastic Optimization (Subsampling)

- If IID data are assumed, log-likelihood becomes summation

$$\log \prod_{i=1}^{n} L(\theta \mid y_i) = \sum_{i=1}^{n} \ell(\theta \mid y_i) \tag{6}$$

- Stochastic optimization with minibatch of size B is tricking yourself into believing that

$$\sum_{i=1}^{n} \ell(\theta \mid y_i) \approx \frac{n}{B} \sum_{i=1}^{B} \ell(\theta \mid y_{b_{(i)}}) \tag{7}$$

  randomly choosing $b_{(i)}$ without replacement
- Reduce the step size ρ_t as we go
- Must satisfy 2 conditions

$$\sum_{t=1}^{\infty} \rho_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \rho_t^2 < \infty \tag{8}$$

Stochastic Optimization (Subsampling)

- If IID data are assumed, log-likelihood becomes summation

$$\log \prod_{i=1}^{n} L(\theta \mid y_i) = \sum_{i=1}^{n} \ell(\theta \mid y_i) \tag{6}$$

- Stochastic optimization with minibatch of size B is tricking yourself into believing that

$$\sum_{i=1}^{n} \ell(\theta \mid y_i) \approx \frac{n}{B} \sum_{i=1}^{B} \ell(\theta \mid y_{b_{(i)}}) \tag{7}$$

  randomly choosing $b_{(i)}$ without replacement
- Reduce the step size ρ_t as we go
- Must satisfy 2 conditions

$$\sum_{t=1}^{\infty} \rho_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \rho_t^2 < \infty \tag{8}$$

ADVI with everything combined

- Minimize KL divergence = maximize ELBO
- Lower bound is

$$\mathrm{E}_q\left[\log p(x,\theta)\right] - \mathrm{E}_q\left[\log q(\theta)\right] \tag{9}$$

- The every element of the parameter vector must have the whole real line as its support
- If it doesn't, transform

$$\zeta = T(\theta) \tag{10}$$

We should also consider the change in density by multiplying the Jacobian term

- Since we assume $\zeta \sim \mathcal{N}(\mu, LL')$, we take $\eta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and use the following transformation again

$$S(\eta) = L\eta + \mu = \zeta \tag{11}$$

ADVI with everything combined

- Minimize KL divergence = maximize ELBO
- Lower bound is

$$\mathsf{E}_q\left[\log p(x,\theta)\right] - \mathsf{E}_q\left[\log q(\theta)\right] \tag{9}$$

- The every element of the parameter vector must have the whole real line as its support
- If it doesn't, transform

$$\zeta = T(\theta) \tag{10}$$

We should also consider the change in density by multiplying the Jacobian term
- Since we assume $\zeta \sim \mathcal{N}(\mu, LL')$, we take $\eta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and use the following transformation again

$$S(\eta) = L\eta + \mu = \zeta \tag{11}$$

ADVI with everything combined

- Minimize KL divergence = maximize ELBO
- Lower bound is

$$\mathsf{E}_q\left[\log p(x,\theta)\right] - \mathsf{E}_q\left[\log q(\theta)\right] \tag{9}$$

- The every element of the parameter vector must have the whole real line as its support
- If it doesn't, transform

$$\zeta = T(\theta) \tag{10}$$

We should also consider the change in density by multiplying the Jacobian term

- Since we assume $\zeta \sim \mathcal{N}(\mu, LL')$, we take $\eta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and use the following transformation again

$$S(\eta) = L\eta + \mu = \zeta \tag{11}$$

ADVI with everything combined

- Minimize KL divergence = maximize ELBO
- Lower bound is

$$\mathsf{E}_q\left[\log p(x, \theta)\right] - \mathsf{E}_q\left[\log q(\theta)\right] \tag{9}$$

- The every element of the parameter vector must have the whole real line as its support
- If it doesn't, transform

$$\zeta = T(\theta) \tag{10}$$

  We should also consider the change in density by multiplying the Jacobian term
- Since we assume $\zeta \sim \mathcal{N}(\mu, LL')$, we take $\eta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and use the following transformation again

$$S(\eta) = L\eta + \mu = \zeta \tag{11}$$

ADVI with everything combined

- Minimize KL divergence = maximize ELBO
- Lower bound is

$$\mathsf{E}_q\left[\log p(x,\theta)\right] - \mathsf{E}_q\left[\log q(\theta)\right] \tag{9}$$

- The every element of the parameter vector must have the whole real line as its support
- If it doesn't, transform

$$\zeta = T(\theta) \tag{10}$$

  We should also consider the change in density by multiplying the Jacobian term
- Since we assume $\zeta \sim \mathcal{N}(\mu, LL')$, we take $\eta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and use the following transformation again

$$S(\eta) = L\eta + \mu = \zeta \tag{11}$$

ADVI with everything combined

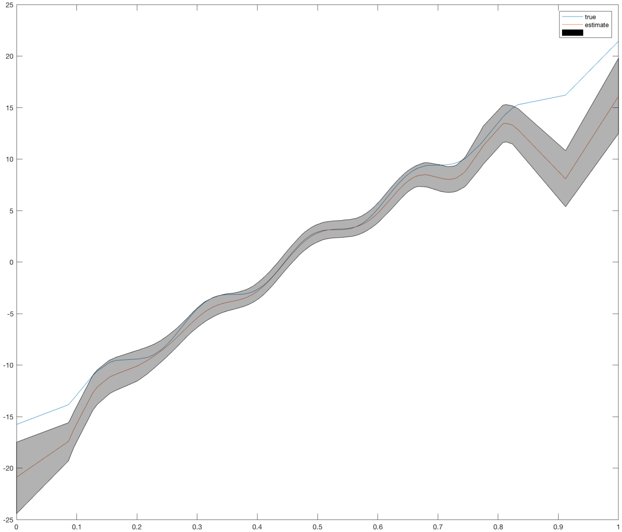
- With the above,

$$
\begin{aligned}
\mathcal{L} &= \mathsf{E}_{q(\zeta)}\left[\log p(x, T^{-1}(\zeta)) + \log\left|\det J_{T^{-1}}(\zeta)\right|\right] - \mathsf{E}_{q(\zeta)}\left[\log q(\zeta)\right] \\
\nabla_\mu \mathcal{L} &= \mathsf{E}_{\mathcal{N}(\eta)}\left[\nabla_\theta \log p(x, \theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log\left|\det J_{T^{-1}}(\zeta)\right|\right] \\
\nabla_L \mathcal{L} &= \mathsf{E}_{\mathcal{N}(\eta)}\left[\left(\nabla_\theta \log p(x, \theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log\left|\det J_{T^{-1}}(\zeta)\right|\right) \eta'\right] \\
&\quad + \left(L^{-1}\right)'
\end{aligned}
$$

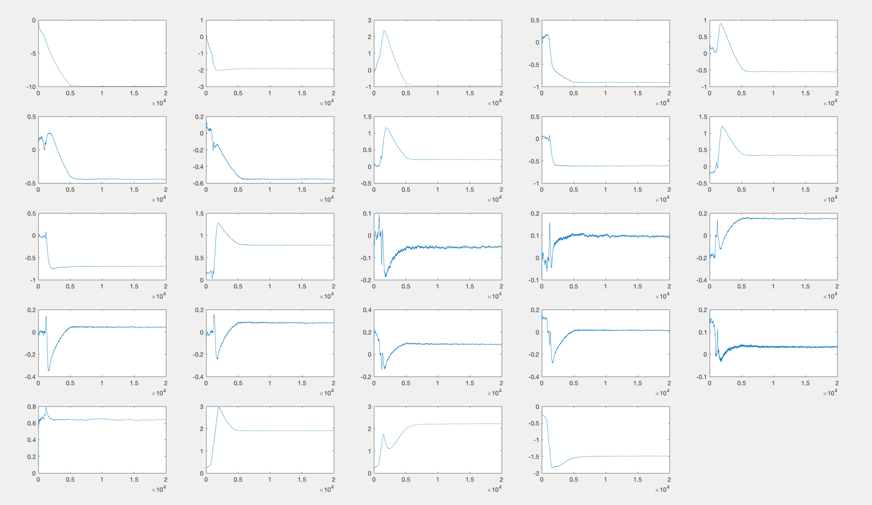
Application of ADVI to BSAR

- Model
  - $\boldsymbol{y} \,|\, \boldsymbol{\theta}, \sigma^2 \sim \mathcal{N}\left(\boldsymbol{\varphi\theta}, \sigma^2 \mathbf{I}_n\right)$
  - Priors
    - $\theta_j \,|\, \sigma, \tau, \gamma \sim \mathcal{N}\left(0, \sigma^2 \tau^2 e^{-j\gamma}\right)$
    - $\sigma^2 \sim \text{InvGam}\left(\frac{r_{0,\sigma}}{2}, \frac{s_{0,\sigma}}{2}\right)$
    - $\tau^2 \sim \text{InvGam}\left(\frac{r_{0,\tau}}{2}, \frac{s_{0,\tau}}{2}\right)$
    - $\gamma \sim \mathrm{Exp}(w_0)$
  - Transformation
    - $\alpha = \log(\exp(\sigma^2) - 1)$
    - $\beta = \log(\exp(\tau^2) - 1)$
    - $\xi = \log(\exp(\gamma) - 1)$
  - $\Theta = (\boldsymbol{\theta}', \alpha, \beta, \xi) \sim \mathcal{N}(\mu, LL')$

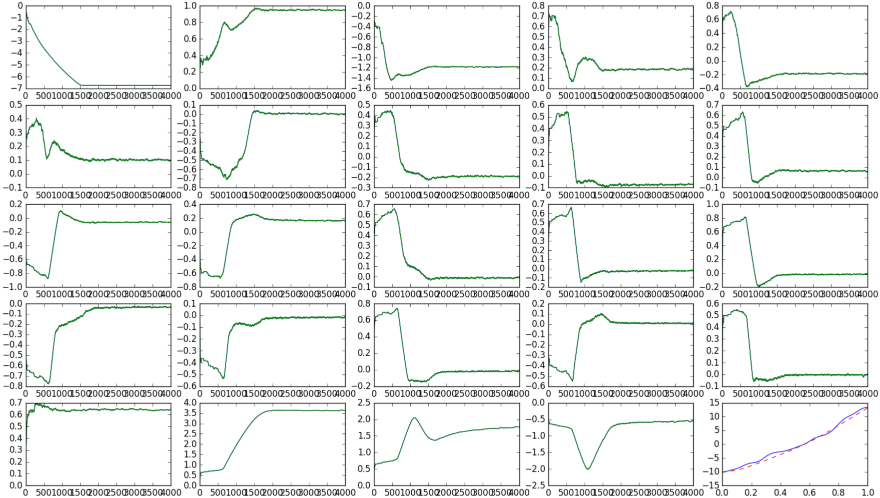
Application of ADVI to BSAR



Application of ADVI to BSAR



Application of ADVI to BSAR



Extension

- Variational Boosting
  - Boosting is accumulating weak learners to gain strength
  - In the context of VI, a weak learner is the variational distribution
  - Accumulating weak learners will make the variational distribution a Gaussian mixture
  - Generic property of ADVI lends itself to extension

Extension

- Variational Boosting
  - Boosting is accumulating weak learners to gain strength
  - In the context of VI, a weak learner is the variational distribution
  - Accumulating weak learners will make the variational distribution a Gaussian mixture
  - Generic property of ADVI lends itself to extension

Extension

- Variational Boosting
  - Boosting is accumulating weak learners to gain strength
  - In the context of VI, a weak learner is the variational distribution
  - Accumulating weak learners will make the variational distribution a Gaussian mixture
  - Generic property of ADVI lends itself to extension

Extension

- Variational Boosting
  - Boosting is accumulating weak learners to gain strength
  - In the context of VI, a weak learner is the variational distribution
  - Accumulating weak learners will make the variational distribution a Gaussian mixture
  - Generic property of ADVI lends itself to extension

Extension

- Variational Boosting
  - Boosting is accumulating weak learners to gain strength
  - In the context of VI, a weak learner is the variational distribution
  - Accumulating weak learners will make the variational distribution a Gaussian mixture
  - Generic property of ADVI lends itself to extension

Things I couldn't do

These are things I could have done if I had had more time...

- Use minibatch to scale up
- Compare how fast ADVI is to MCMC when dataset is large (with data subsampling)
- Impose shape restriction
- BSAR GLM
- BSAR GLM with shape restriction