Gaussian Process Regression and Emulation

STAT8810, Fall 2017

M.T. Pratola

September 16, 2017

Today

Stochastic Gradient Descent;
GP's for "big data"

Maximum Likelihood in a Big Data World

- Recall our log-likelihood function $\ell(\boldsymbol{\theta}; \mathbf{y}, \mathbf{X})$. When the data are independent, our log-likelihood is

$$\ell(\boldsymbol{\theta}; \mathbf{y}, \mathbf{X}) = \sum_{i=1}^{n} \ell(\boldsymbol{\theta}; y_i, \mathbf{x}_i)$$

  for a scalar response y_i and a vector input $\mathbf{x}_i$.
- When n is very large, maximizing the likelihood is expensive since calculating the likelihood involves all n of the observed data $(y_i, \mathbf{x}_i)$.

Maximum Likelihood in a Big Data World

- Thinking back to our Newton-Raphson algorithm, this means that each iterative update to the parameter, $\boldsymbol{\theta}_t \rightarrow \boldsymbol{\theta}_{t+1}$, induces a huge computational cost.
- In practice, this means that the optimization process will be terminated relatively early (in terms of number of iterations t) due to this cost.
- In a highly influential paper, Bottou† investigates the relative tradeoff of information cost versus computational cost and shows that *stochastic gradient descent* arrives at a better tradeoff in the big data setting.‡

† Bottou: *Large-scale machine learning with stochastic gradient descent*, Proceedings of COMPSTAT'2010, pp. 177-186 (2010).

‡ under some reasonable assumptions, naturally.

Simple Supervised Learning Setup

- Consider predicting y by $\hat{y}$ where the predictor $\hat{y}$ is chosen from a family of functions $\mathcal{F} = \{f_{\boldsymbol{\theta}}(\mathbf{x})\}$ parameterized by weight vector $\boldsymbol{\theta}$.
- The loss induced by a function $f \in \mathcal{F}$ is defined as the negative log-likelihood function,

$$Q(z; \boldsymbol{\theta}) = -\ell(y, \mathbf{x}; \boldsymbol{\theta}),$$

  where $z = (y, \mathbf{x})$.
- We would like choose a function so as to minimize the *expected risk*,

$$\overline{Q} = \int Q(z; \boldsymbol{\theta}) dP(z) = -\int \ell((y, \mathbf{x}; \boldsymbol{\theta}) dP(z)$$

- In practice we usually settle for minimizing the *empirical risk*,

$$\overline{Q}_n = \frac{1}{n} \sum_{i=1}^{n} Q(z_i; \boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \mathbf{x_i}; \boldsymbol{\theta}).$$

First, some things we know...

- The MSE of an estimator $\hat{\ell}$ of ℓ has the following decomposition, known as the bias-variance tradeoff:

$$E[(\hat{\ell} - \ell)^2] = \text{Bias}(\hat{\ell})^2 + \text{Var}(\hat{\ell})$$

  where $\text{Bias}(\hat{\ell}) = E[\hat{\ell}] - \ell$.
- The law of large numbers tells us that an estimator of the form

$$\bar{\ell}_n = \frac{1}{n}\sum_{i=1}^{n} \ell_i \to \ell$$

  as $n \to \infty$, where convergence can be in probability (weak law) or almost surely (strong law).
- For an estimator of the form $\bar{\ell}_n = \frac{1}{n}\sum_{i=1}^{n} \ell_i$, Chebyshev's inequality tells us that

$$P\left(|\bar{\ell}_n - \ell| > \epsilon\right) \leq \frac{\sigma^2}{n\epsilon^2}$$

  where $\sigma^2 = \text{Var}(\ell_i)$.

Interpretation

- When n is large, as in the big data setting, then we would like to believe that our estimator of the loss is fairly accurate.
  - Nonetheless, until we reach infinity, it must be the case that there is some bias, and therefore our gradient *is targetting the wrong thing*.
  - In other words, while we are guaranteed to (eventually) minimize the empirical loss, we have no guarantee (and indeed it is very unlikely) that we will minimize the expected loss.

Interpretation

- At the same time, in the big data setting we are spending a ton of computational effort to perform our updates at each iteration of our optimization algorithm $\boldsymbol{\theta}_t \rightarrow \boldsymbol{\theta}_{t+1} \rightarrow \ldots$.
  - Given that we know we are targetting the wrong endpoint, is this the most sensible use of computational resources?
  - Does it make sense to get as best of an estimate of the information content for $\boldsymbol{\theta}_t$ to update to $\boldsymbol{\theta}_{t+1}$ given that we will practically be computationally limited in how many such updates we can afford to make?
  - In other words, could we trade some variance for a procedure that would be quicker? Even better, could we trade some variance for a procedure that would be quicker and also target the expected loss rather than the empirical loss?

Information versus Computation

- Those facts that we know (bias-variance tradeoff, asymptotic convergence of sample means) are all "information efficiency" properties, but they don't tell us much about computational tradeoffs.
- Really, they pretend that computation is "for free".

Algorithms

Bottou (2010) investigates 4 optimization algorithms:

- Gradient Descent (GD):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} Q(z_i; \boldsymbol{\theta}_t)$$

Algorithms

Bottou (2010) investigates 4 optimization algorithms:

- Second-order Gradient Descent (2GD):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \Gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} Q(z_i; \boldsymbol{\theta}_t)$$

(which is N-R when $\Gamma_t = \mathbf{H}^{-1}(\boldsymbol{\theta}_t)$)

Algorithms

Bottou (2010) investigates 4 optimization algorithms:

- Stochastic Gradient Descent (SGD):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \nabla_{\boldsymbol{\theta}} Q(z_t; \boldsymbol{\theta}_t)$$

  where z_t is a randomly picked example from our training dataset.

Algorithms

Bottou (2010) investigates 4 optimization algorithms:

- Second-order Stochastic Gradient Descent (2SGD):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \Gamma_t \nabla_{\boldsymbol{\theta}} Q(z_t; \boldsymbol{\theta}_t)$$

  where z_t is a randomly picked example from our training dataset.

Big Data Learning

- Let $f^* = \arg\min_f \bar{Q}(f)$ represent the best possible prediction function.
- Let $f^*_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} \bar{Q}(f)$ represent the best possible prediction function restricted to our family of functions $\mathcal{F}$.
- Let $f_n = \arg\min_{f \in \mathcal{F}} \bar{Q}_n(f)$ represent the empirical optimum function restricted to our family $\mathcal{F}$. This assumes that we the optimization routine used can reach f_n and we let it run long enough to indeed recover it.
- Let $\tilde{f}_n$ represent the prediction function found that minimizes the empirical risk to a pre-defined level of accuracy, $\bar{Q}_n(\tilde{f}_n) < \bar{Q}_n(f_n) + \rho$.

Big Data Learning

- Let $\mathcal{E} = E\left[\bar{Q}(\tilde{f}_n) - \bar{Q}(f^*)\right]$ be the excess error of our solution $\tilde{f}_n$. This error can be decomposed into three terms:

$$\mathcal{E} = E\left[\bar{Q}(f_{\mathcal{F}}^*) - \bar{Q}(f^*)\right] + E\left[\bar{Q}(f_n) - \bar{Q}(f_{\mathcal{F}}^*)\right] + E\left[\bar{Q}(\tilde{f}_n) - \bar{Q}(f_n)\right]$$

$$= \mathcal{E}_{app} + \mathcal{E}_{est} + \mathcal{E}_{opt}.$$

Big Data Learning

- The approximation error, $\mathcal{E}_{app} = E\left[\bar{Q}(f^*_{\mathcal{F}}) - \bar{Q}(f^*)\right]$, measures how closely functions in our family $\mathcal{F}$ can approximate the best solution f^*.
- The estimation error, $\mathcal{E}_{est} = E\left[\bar{Q}(f_n) - \bar{Q}(f^*_{\mathcal{F}})\right]$, measures the effect of minimizing the empircal risk, $\bar{Q}_n(f)$ instead of the expected risk, $\bar{Q}(f)$.
- The optimization error, $\mathcal{E}_{opt} = E\left[\bar{Q}(\tilde{f}_n) - \bar{Q}(f_n)\right]$ measures the impact of approximate optimization (e.g. computational runtime limitations) on the expected risk.

Big Data Learning

- Given a time limit on computation, T_{max}, a limit on the samples, n_{max}, and a target optimization accuracy, ρ, our optimization problem may be more accurately stated as

$$min_{\mathcal{F},\rho,n}\mathcal{E} = \mathcal{E}_{app}+\mathcal{E}_{est}+\mathcal{E}_{opt} \text{ subject to } \begin{cases} n \leq n_{max} \\ T(\mathcal{F},\rho,n) \leq T_{max} \end{cases}$$

  - in small-scale learning problems, we are always far away from T_{max} so we simply choose $n = n_{max}$ and arbitrarily set ρ to effectively eliminate the optimization error $\mathcal{E}_{opt}$.
  - but in large-scale problems, we are always limited by T_{max}. So we need to account for the tradeoffs between n, ρ and $T(\mathcal{F},\rho,n)$.

Optimization Accuracy

- A summary of these algorithms' optimization performance (in terms of accuracy, ρ) is given in Bottou (2010):

	GD	GD2	SGD	SGD2
time per iteration	n	n	1	1
iterations to ρ	$log(\frac{1}{\rho})$	$log(log(\frac{1}{\rho}))$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
time to ρ	$nlog(\frac{1}{\rho})$	$nlog(log(\frac{1}{\rho}))$	$\frac{1}{\rho}$	$\frac{1}{\rho}$

- And under some reasonable assumptions, they argue that the asymptotic behavior of the excess error goes like

$$\begin{aligned} \mathcal{E} &= \mathcal{E}_{app} + \mathcal{E}_{est} + \mathcal{E}_{opt} \\ &\sim \mathcal{E}_{app} + \left(\frac{log(n)}{n}\right)^{\alpha} + \rho \quad \dagger \end{aligned}$$

for some $\alpha \in [\frac{1}{2}, 1]$.

† Although they aren't specific, we assume $\sim$ has the usual meaning $lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$.

Optimization Accuracy

- Assuming one would like the three sources of error to asymptotically decrease at the same rate, we have

$$\mathcal{E} \sim \mathcal{E}_{app} \sim \mathcal{E}_{est} \sim \mathcal{E}_{opt} \sim \left(\frac{log(n)}{n}\right)^{\alpha} \sim \rho$$

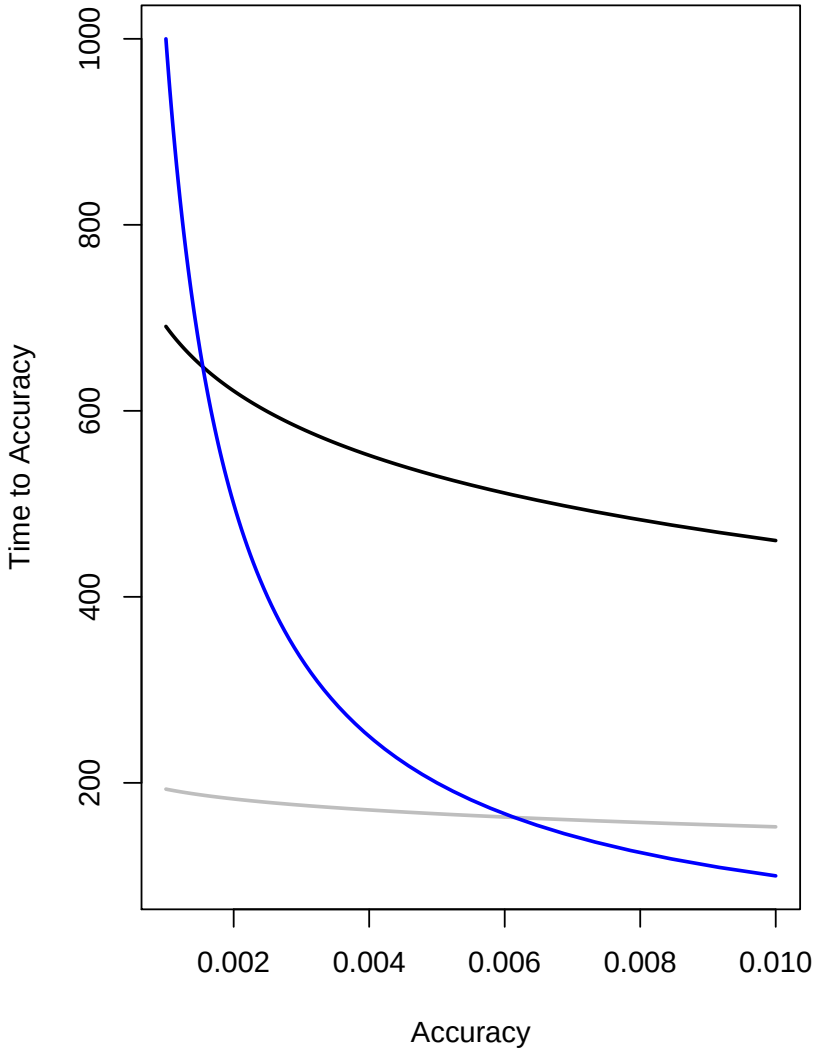
- These asymptotic equivalences allow one to express *n* and *rho* in terms of the excess error $\mathcal{E}$. Then subbing into row (3) of the table, we can compare the algorithms in terms of time to excess error.

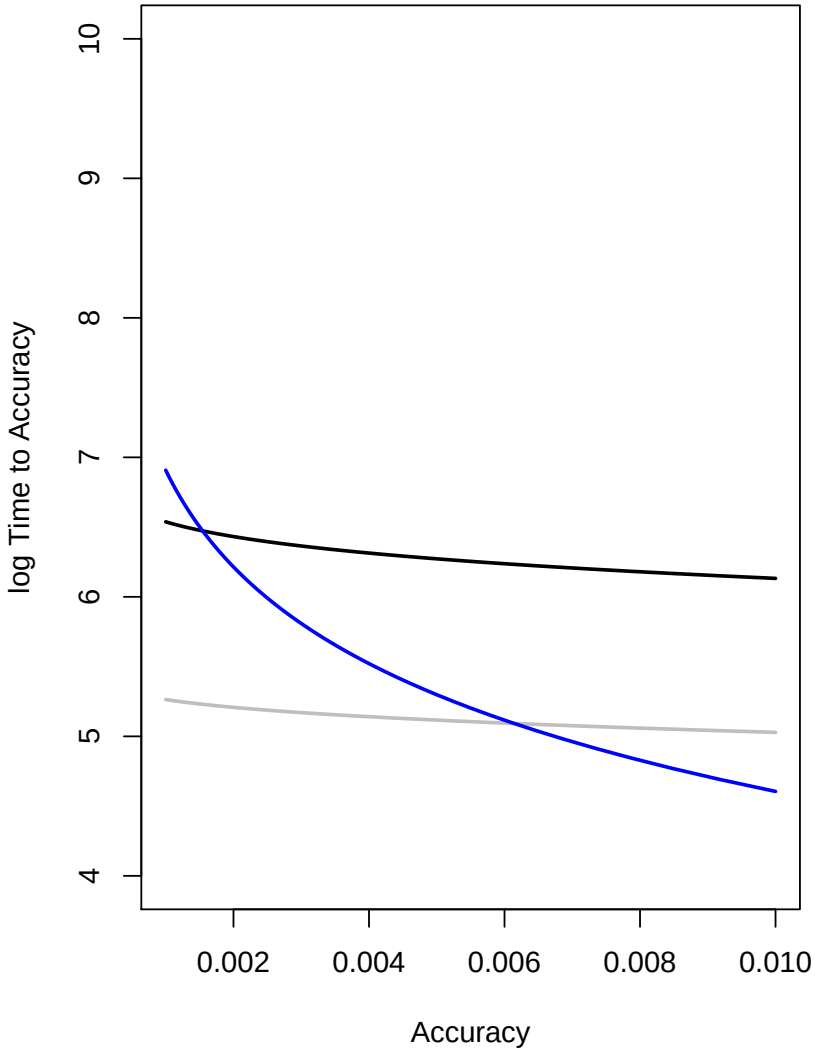
Optimization Accuracy

	GD	GD2	SGD	SGD2
time per iteration	n	n	1	1
iterations to ρ	$log(\frac{1}{\rho})$	$log(log(\frac{1}{\rho}))$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
time to ρ	$nlog(\frac{1}{\rho})$	$nlog(log(\frac{1}{\rho}))$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
time to $\mathcal{E}$	$\frac{1}{\mathcal{E}^{1/\alpha}} log(1/\mathcal{E})^2$	$\frac{1}{\mathcal{E}^{1/\alpha}} log(1/\mathcal{E}) log(log(1/\mathcal{E}))$	$1/\mathcal{E}$	$1/\mathcal{E}$

- We see that in the small-sample case (row 3), SGD/SGD2 are very slow to converge.
- Yet, in the big-data setting (row 4), SGD/SGD2 require the least amount of time to reach a specified expected risk.

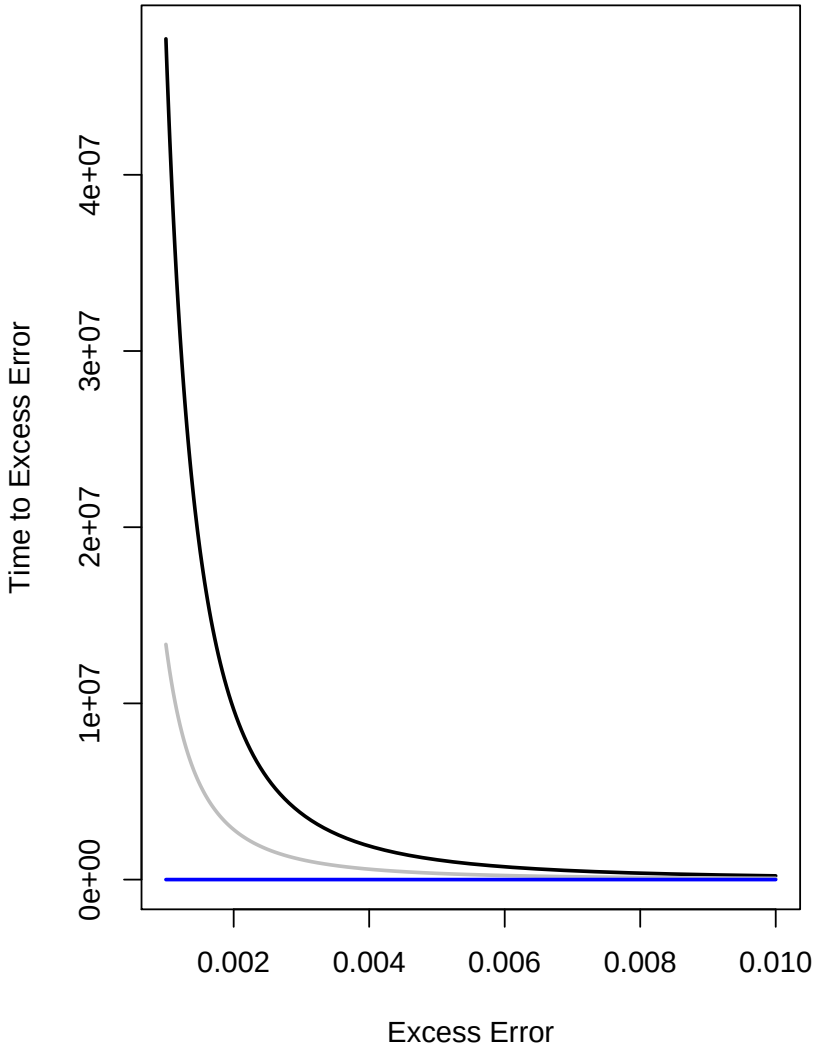
Asymptotic Performance: Time to Accuracy

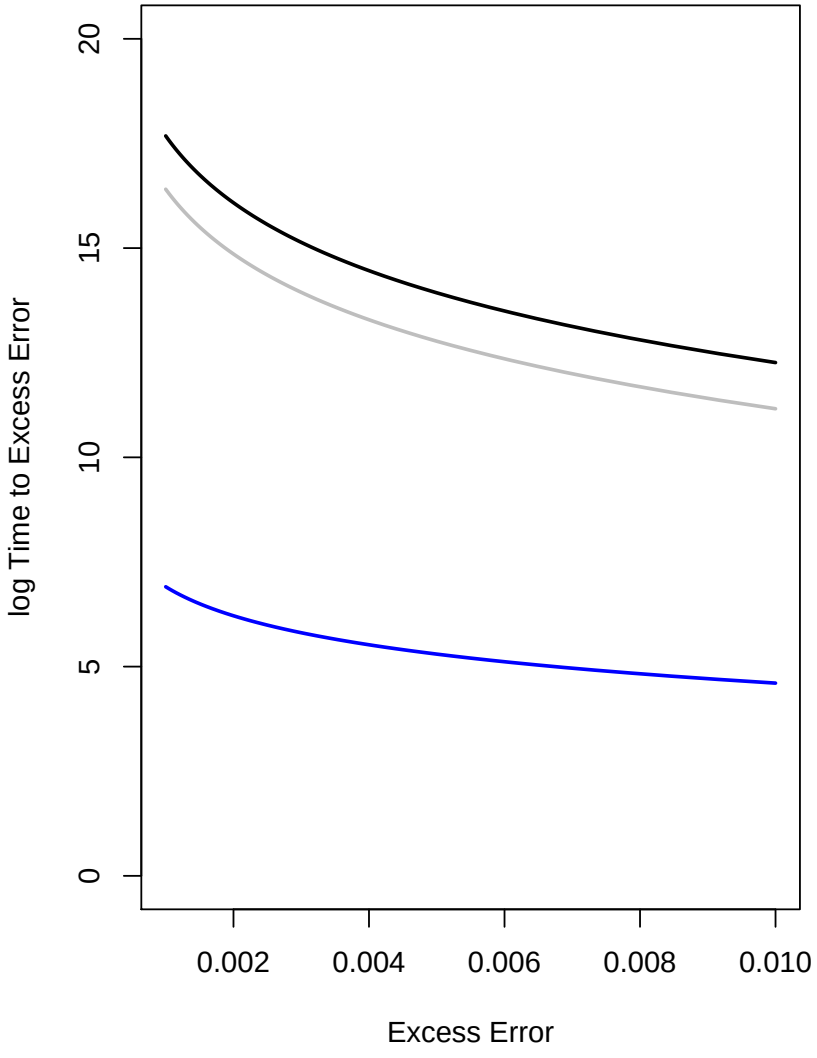




Black=GD, Grey=GD2, Blue=SGD. Here $n = 100$ and $\alpha = 0.5$.
SGD2 differs by a constant so asympotitically equivalent to SGD.

Asymptotic Performance: Time to Excess Error





Black=GD, Grey=GD2, Blue=SGD. Here $\alpha = 0.5$. SGD2 differs by a constant so asympotitically equivalent to SGD.

Stochastic Gradient Descent

- Idea of SGD/SGD2 is that we draw a sample from $P(z)$, compute the loss and update $\boldsymbol{\theta}$, then rinse and repeat.
- In practice, the algorithm draws each sample randomly from the dataset, then the whole procedure is repeated a number of times (called epochs).
- For stochastic gradient descent, convergence requires $\sum_t \gamma_t^2 < \infty$ and $\sum_t \gamma_t = \infty$.
- Best convergence speed, under some regularity conditions, requires $\gamma_t \sim t^{-1}$. In this case, the expected residual error $E[\rho] \sim t^{-1}$.
- There are batch-variants (say draw a small number of samples from $P(z)$) but we won't go into those.

Stochastic Gradient Descent

Input: initial learning rate γ_0, *max_epochs*, $\mathbf{X}$, $\mathbf{y}$

Input: initial parameter estimate $\boldsymbol{\theta}_0$

initialize iteration $t = 0$

for e **in** 1 **to** *max_epochs* **begin**

    randomly shuffle $\mathbf{X}, \mathbf{y}$

    for i **in** 1 **to** n **begin**

        $\text{grad} = \nabla_{\boldsymbol{\theta}} Q(f(\mathbf{x}_i), y_i; \boldsymbol{\theta}_t)$

        $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma_t \times \text{grad}$

        $t = t + 1$

        if *stopping criterion met* **then break**

    end

end

SGD Example

```
library(sgd) # offers various implementations of SGD
set.seed(42)
n = 1e4
d = 10
X = matrix(rnorm(n*d), ncol=d)
theta = rep(5, d+1)
eps = rnorm(n)
y = cbind(1, X) %*% theta + eps
dat = data.frame(y=y, x=X)
fit.sgd = sgd(y ~ ., data=dat, model="lm")
fit.lm = lm(y ~ ., data=dat)
```

SGD Example

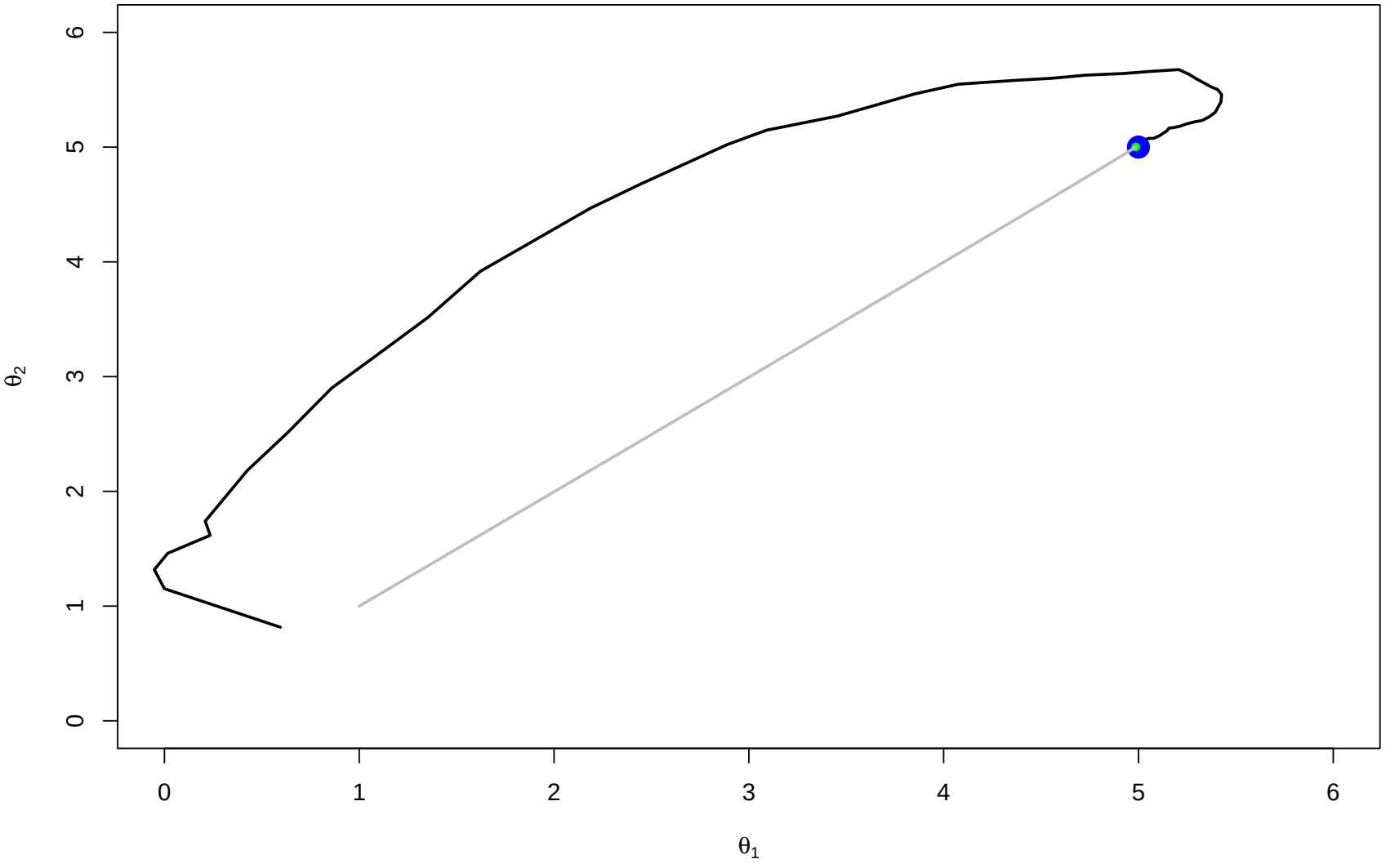
```
Q.lm.grad<-function(theta,y,X) {
    n=length(y)
    -(2/n)*(t(y)-theta%*%t(X))%*%X
}
X=cbind(1,X)
err=Inf
theta.cg.hist=NULL
theta.cg=rep(1,11)
theta.cg.hist=cbind(theta.cg.hist,theta.cg)
eta=0.1
n=length(y)
```

SGD Example

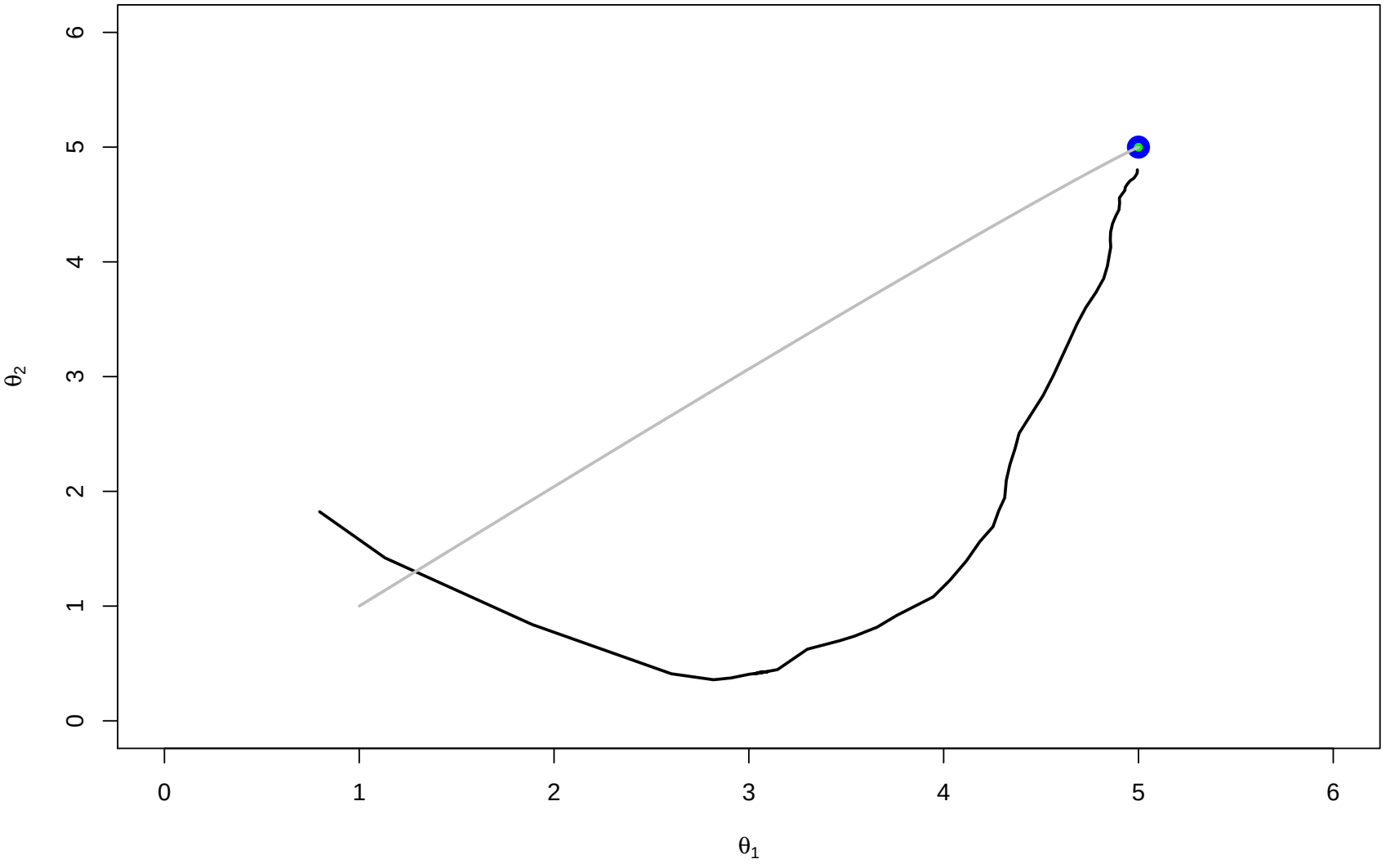
```
while(err>1e-5)
{
    gradient = Q.lm.grad(theta.cg,y,X)
    theta.cg = as.vector(theta.cg - eta*gradient)
    err = sum(gradient^2)
    theta.cg.hist=cbind(theta.cg.hist,theta.cg)
}

plot(fit.sgd$estimates[1,],fit.sgd$estimates[2,],
    type='l',lwd=2,xlab=expression(theta[1]),
    ylab=expression(theta[2]),xlim=c(0,6),ylim=c(0,6))
points(theta[1],theta[2],pch=20,cex=3,col="blue")
points(fit.lm$coefficients[2],fit.lm$coefficients[3],
    pch=20,cex=1,col="green")
lines(theta.cg.hist[2,],theta.cg.hist[3,],lwd=2,
    col="grey")
```

SGD Example



SGD Example



SGD Example

Number of iterations for SGD algorithm:

```
max(fit.sgd$pos)
```

```
## [1] 1189
```

(not even 1 epoch).

Effective number of iterations for CG algorithm:

```
ncol(theta.cg.hist)*n
```

```
## [1] 430000
```

SGD for GPs?

- So SGD may be useful when our likelihood factorizes as

$$\ell(\mathbf{y}) = \sum_{i=1}^{n} \ell(y_i)$$

- Is this helpful in our GP regression case? No.
- However, it turns out that an interesting approximation to the assumed joint distribution of the Gaussian Process can be made, and the approximation is constructed so as to induce the form we need to apply SGD.

Variational Inference

- The idea is to approximate the distribution we want, say p, with a distribution that has a simpler form, say q.
- The simpler form of q is prescribed. For instance, in our situation we would want it to decompose as $\prod_{i=1}^{n} q_i$
- But we need to construct q so that it is, in some sense, close to the desired distribution p.
- We need some way to measure the similarity of p and q.

Kullback-Liebler Divergence

- Is a measure of distributional dissimilarity:

$$\begin{aligned} KL(q||p) &= \int q(y) log\left(\frac{q(y)}{p(y)}\right) dy \\ &= E_{q(y)}\left[log\left(\frac{q(y)}{p(y)}\right)\right] \end{aligned}$$

- $KL(q||p) \geq 0$ with equality iff $q(y) = p(y)$ almost everywhere.
- If $q \equiv N(\mu_q, \Sigma_q)$ and $p \equiv N(\mu_p, \Sigma_p)$ then

$$KL(q||p) = \frac{1}{2}[tr(\Sigma_p^{-1}\Sigma_q) + (\mu_p - \mu_q)^T\Sigma_p^{-1}(\mu_p - \mu_q) - n - ln\left(\frac{det(\Sigma_p)}{det(\Sigma_q)}\right)]$$

- $KL(q||p) = -\int q(y) log\left(\frac{p(y)}{q(y)}\right) dy$

Kullback-Liebler Divergence

- A popular application of the KL divergence follows in the following sense. Let $p(y, z) = p(z|y)p(y)$
- Then, we have

$$ln(p(y)) = \mathcal{L}(q(z)) + KL(q(z)||p(y))$$

where

$$\mathcal{L}(q(z)) = \int q(z) ln \left(\frac{p(y, z)}{q(z)} \right) dz$$

and

$$KL(q(z)||p(y)) = -\int q(z) ln \left(\frac{p(z|y)}{q(z)} \right) dz$$

Kullback-Liebler Divergence

Check:

$$\int [log(p(y,z)) - log(q(z))]q(z)dz - [\int (log(p(z|y)) - log(q(z)))q(z)dz]$$

$$= \int q(z) log\left(\frac{log(p(y,z))}{p(z|y)}\right) dz$$

$$= \int q(z) log\left(\frac{log(p(z|y)p(y))}{p(z|y)}\right) dz$$

$$= p(y)$$

Kullback-Liebler Divergence

- Interpretation is $q(z)$ is our approximation to the conditional distribution $p(z|y)$ and $KL(q||p)$ measures how far away our approximation is.
- This means that the log-likelihood, $ln(p(y))$ is factored into a KL term and the $\mathcal{L}$ term.
- Since $KL \geq 0$, the $\mathcal{L}$ term represents a lower-bound on the log-likelihood.

Breaking the dependency

- Recall that our predictor was the BLUP:

$$\hat{f}(x) = \mathbf{r}^{\mathbf{T}}\mathbf{R}^{-1}\mathbf{y}$$

  where $\mathbf{r}^T = (cor(f(\mathbf{x}), f(\mathbf{x}_1), \ldots, cor(f(\mathbf{x}), f(\mathbf{x}_n)))$
- By the properties of the MVN distribution, this is nothing but the conditional expectation given our data.

MVN Conditional Mean

If

$$\begin{pmatrix} f(\mathbf{x}) \\ \hline f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_n) \end{pmatrix} \sim N\left(\left(\begin{array}{c} 0 \\ \mathbf{0}_n \end{array} \right), \sigma^2 \left[\begin{array}{cc} \mathbf{1} & \mathbf{r}^T \\ \mathbf{r} & \mathbf{R} \end{array} \right] \right)$$

then

$$f(\mathbf{x}) | f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n) \sim N\left(\mathbf{r}^T \mathbf{R}^{-1} \mathbf{y}, \sigma^2 (1 - \mathbf{r}^T \mathbf{R}^{-1} \mathbf{r} \right)$$

Note that the conditional mean of this distribution is
$E[f(\mathbf{x}) | (f_1, \ldots, f_n)^T = \mathbf{y}] = \mathbf{r}^T \mathbf{R}^{-1} \mathbf{y} = \hat{f}(\mathbf{x})$

GPs with SGD

- Hensman et al (2013)† combine three ideas to arrive at an approximate inference procedure for GP regression that allows the use of SGD.
- They combine
  - the properties of conditional Normals and *landmark points* or *inducing variables*
  - an assumption that our function is observed with some noise (not unlike adding a small diagonal term to **R** in our usual setup)
  - a variational approximation for performing model fitting on a distribution that factors in a useful way for using SGD

† Hensman, Fusi, and Lawrence: *Gaussian processes for big data*, arXiv preprint arXiv:1309.6835 (2013).

The Basic Idea

- Let's look at our conditional distribution again, but this time lets pretend we are predicting at many points rather than a single point:

If

$$\begin{pmatrix} f(\mathbf{x_1}) \\ \vdots \\ f(\mathbf{x_n}) \\ \hline f(\mathbf{x}'_1) \\ \vdots \\ f(\mathbf{x}'_m) \end{pmatrix} \sim N\left(\begin{pmatrix} \mathbf{0}_n \\ \mathbf{0}_m \end{pmatrix}, \sigma^2 \begin{bmatrix} \mathbf{R}_{nn} & \mathbf{R}_{nm} \\ \mathbf{R}_{mn} & \mathbf{R}_{nn} \end{bmatrix}\right)$$

The Basic Idea

Then

$$f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n) | \left(f(\mathbf{x}'_1), \ldots, f(\mathbf{x}'_m)\right) = \mathbf{y} \sim$$
$$N\left(\mathbf{R}_{nm}\mathbf{R}_{mm}^{-1}\mathbf{y}, \sigma^2(\mathbf{R}_{nn} - \mathbf{R}_{nm}\mathbf{R}_{mm}^{-1}\mathbf{R}_{mn}\right)$$

- If $m << n$, wouldn't it be great if we could learn our parameter estimates only using $f_1, \ldots, f_m$? Our matrix inversions would be much less costly!
- This is not possible. But even if it were, where would we choose the locations $x'_1, \ldots, x'_m$?

GPs with SGD

- Let $\mathbf{y}$ be our data vector of noisy observations of $f(\mathbf{x}_i),\ i = 1, \ldots, n$:

$$y_i = f(\mathbf{x}_i) + \epsilon$$

  where $\epsilon \sim N(0, \beta^{-1})$.
- Let $\mathbf{f} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))$ be the vector of function observations at settings $\mathbf{x}_1, \ldots, \mathbf{x}_n$.
- Let $\mathbf{u} = (f(\mathbf{x}'_1), \ldots, f(\mathbf{x}'_m))$ be the vector of "inducing variables" which are the observations of the same function at landmark points $\mathbf{x}'_1, \ldots, \mathbf{x}'_m$.

GPs with SGD

- We will model our $(\mathbf{f}, \mathbf{u})$ as a realization of a (smooth) Gaussian Process:

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{u} \end{pmatrix} \sim N\left(\begin{pmatrix} \mathbf{0}_n \\ \mathbf{0}_m \end{pmatrix}, \begin{bmatrix} \mathbf{K}_{nn} & \mathbf{K}_{nm} \\ \mathbf{K}_{mn} & \mathbf{K}_{mm} \end{bmatrix}\right)$$

which admits the factorization

$$p(\mathbf{f}|\mathbf{u})p(\mathbf{u}),$$

where

$$p(\mathbf{u}) = N\left(\mathbf{0}_m, \mathbf{K}_{mm}\right)$$

and

$$p(\mathbf{f}|\mathbf{u}) = N\left(\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{u}, \tilde{\mathbf{K}}\right),$$

where

$$\tilde{\mathbf{K}} = \mathbf{K}_{nn} - \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{mn}.$$

GPs with SGD

- Our observed data has conditional distribution

$$\mathbf{y}|\mathbf{f} \sim N(\mathbf{f}, \beta^{-1}\mathbf{I}_{nn}) = \prod_{i=1}^{n} N(f_i, \beta^{-1})$$

- This leads to the conditional likelihood function

$$\mathbf{y}|\mathbf{u} \sim N(\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{u}, \beta^{-1}\mathbf{I}_{nn} + \tilde{\mathbf{K}})$$

GPs with SGD

- The approach starts with placing a lower-bound on the conditional likelihood,

$$log(p(\mathbf{y}|\mathbf{u})) \geq \int log(p(\mathbf{y}|\mathbf{f}))p(\mathbf{f}|\mathbf{u})d\mathbf{f} \equiv E_{\mathbf{f}|\mathbf{u}}[log(p(\mathbf{y}|\mathbf{f}))]$$

- Since we have $p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{n} N(f_i, \beta^{-1})$ one can arrive at

$$exp(\mathcal{L}) = \prod_{i=1}^{n} N(y_i|\mu_i, \beta^{-1}) exp(-\frac{1}{2}\tilde{k}_{ii})$$

where $\boldsymbol{\mu} = \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{u}$.

GPs with SGD

- Next they introduce the variational distribution $q(\mathbf{u}) \sim N(\mathbf{m}, \mathbf{S})$ and lower-bound the overall data likelihood,

$$log(p(\mathbf{y})) \geq \int (\mathcal{L} + log(p(\mathbf{u})) - log(q(\mathbf{u})))q(\mathbf{u})d\mathbf{u} := \mathcal{L}'$$

- This turns out to have the following separable form,

$$\sum_{i=1}^{n} \left(logN\left(y_i | \mathbf{k}_i^T \mathbf{K}_{mm}^{-1} \mathbf{m}, \beta^{-1}\right) - \frac{1}{2}\beta \tilde{k}_{ii} - \frac{1}{2} tr(S\Lambda_i) \right) - KL(p(\mathbf{u})||q(\mathbf{u})$$

where $\Lambda_i = \beta \mathbf{K}_{mm}^{-1} \mathbf{k}_i \mathbf{k}_i^T \mathbf{K}_{mm}^{-1}$

GPs with SGD

- Taking the derivitave with respect to the parameters of the variational distribution, we arrive at

$$\hat{\mathbf{m}} = \beta \Lambda^{-1} \mathbf{K}_{mm}^{-1} \mathbf{K}_{mn} \mathbf{y}$$

$$\hat{\mathbf{S}} = \Lambda^{-1}$$

where $\Lambda = \mathbf{K}_{mm}^{-1} + \sum \Lambda_i$.

GPs with SGD

- The overall procedure iterates over the following steps until convergence:

1. optimize the landmark locations $\mathbf{x}'_1, \ldots, \mathbf{x}'_m$ with resepct to $\mathcal{L}$.
   - this causes the landmark locations to be chosen so as to minimize the $\tilde{k}_{ii}$, essentially ensuring the landmarks are never too far away from the training data
2. optimize $\mathbf{m}, \mathbf{S}$ with respect to $\mathcal{L}'$.
   - in practice they recommend using a batch-sequential variant of SGD.
3. optimize kernel hyperparameters and β with respect to $\mathcal{L}'$.

The implementation is available as Python package `GPy` at `https://github.com/SheffieldML/GPy`.

Examples

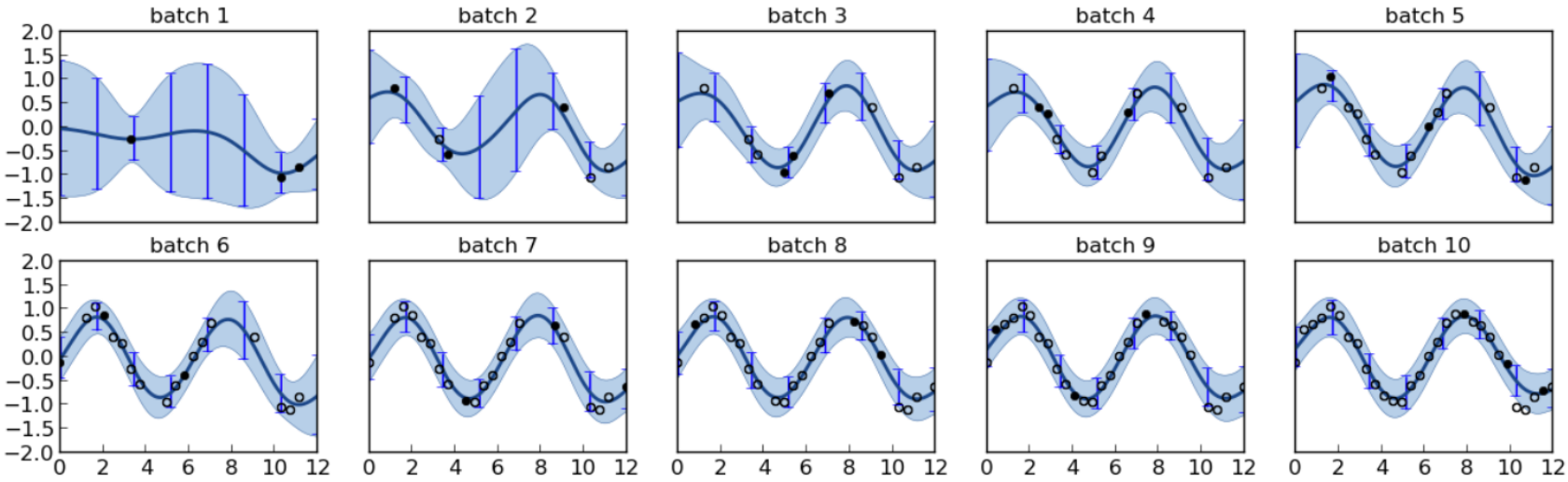


Figure 1: Trivial GP example

Each pane shows the GP after updating with 1 iteration of batch-SGD. The batches are shown as solid dots, while data from previous batches are shown as empty dots. The location of inducing variables and the distribution of $q(\mathbf{u})$ is shown as the vertical bars. Source: Hensman et al (2013).

Examples

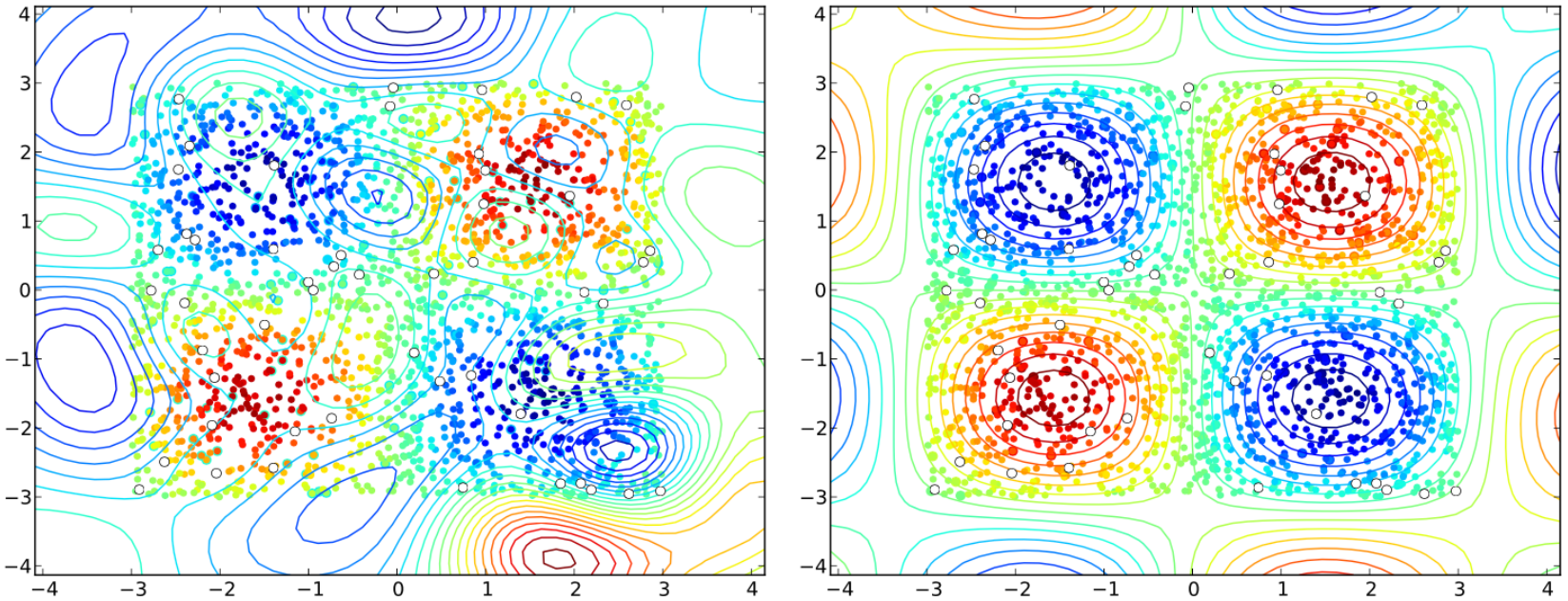


Figure 2: Simple 2D GP example

Colored points are data while model fit is represented by the contours. Locations of the inducing variables are shown as the empty dots. Source: Hensman et al (2013).

GPs with SGD

- In practice, in the examples they demonstrate they seem to choose the inducing points a priori using some method which is unclear, and then hold them fixed.
- In an apartment price dataset with 10k training points, $m = 800$ inducing points were used when regressing log(Price) on lattitude/longitude.
  - training was batch-SGD with a batch size of 1,000.
- In an airline delays dataset with 700k training points and $p = 8$ predictors, $m = 1,000$ inducing points were used.
  - training was batch-SGD with a batch size of 5,000.

GPs with SGD

- Positives:
  - it is cool that they were able to linearize the inference to take advantage of SGD.
- Possible Negatives:
  - m itself may grow too large
  - how to chose number of inducing points?
  - SGD is still a sequential procedure - we might want parallelism for really huge data.
  - what about non-stationarity?