CS5014

Reparameterise loss function

Lei Fang

01/02/2021

Contents

1 Reparameterisation for gradient descent

Suppose you are asked to optimise a function $L(\theta)$ with respect to θ. Your first choice probably should be gradient descent if L is a continous and smooth function of θ. However, if θ's value range is restricted, e.g. $\theta > 0$, $0 < \theta < 1$, gradient descent might fail as the algorithm might step out side of the legal region. A common trick is to find a surrogate $-\infty < \beta < \infty$ such that

$$f(\beta) = \theta$$

and apply gradient descent with respect to β instead. By doing this, gradient descent will usually perform more stably without the danger of undefined gradients. When the gradient descent converges, we can apply the same transform to find the corresponding $\theta \leftarrow f(\beta)$[1].

Common choices of such f are usually simples functions.

- for $\theta > 0$, exponential function can be used as $\exp(\beta) = \theta$, and $\beta \in R$;
- for $\theta \in [0, 1]$, sigmoid function is one of the options as $\sigma(\beta) = \theta$, which squeezes $\beta \in R$ to a value between 0 and 1. Note that

  $$\sigma(x) = \frac{1}{1 + \exp(-x)},$$

  and a plot of a sigmoid function is shown below in Fig 1.

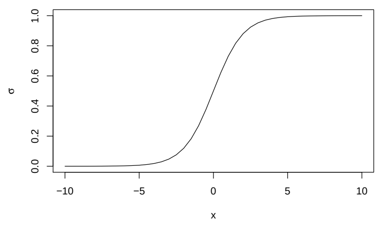


Figure 1: Sigmoid function that squeezes every real value to between 0 and 1

The new gradient w.r.t β can be found by using chain rule:

$$\frac{d\mathcal{L}(f(\beta))}{d\beta} = \frac{d\mathcal{L}(\theta)}{d\theta}\frac{d\theta}{d\beta}.$$

Equivelently, you can also write $\mathcal{L}$ as a function of β explicitly $\mathcal{L}(\beta)$ and take the derivative with β directly. You should get exactly the same result.

Note that the following two cases are just toy examples. In real world applications, there is no reason to use gradient descent to find those parameters as there are simple closed form analytical solutions (by setting the gradient to zero and find the estimator). I use them as examples to show how the trick works.

1.1 Bernoulli likelihood

In the lecture, we did an example with $0 < \theta < 1$ for the Bernoulli likelihood model: $P(y^{(i)}) = \theta^{y^{(i)}}(1-\theta)^{1-y^{(i)}}$ and $y^{(i)} = 1, 0$. The log likelihood function of the model is

$$\mathcal{L}(\theta) = \log P(\{y^{(i)}\}_1^m|\theta) = R\log\theta + (m-R)\log(1-\theta),$$

where $R = \sum_{i=1}^m y^{(i)}$ is the total count of heads in the dataset, and m is total size of the data.

The problematic bit is θ has to fall into the range of $(0, 1)$, which is hard to deal with when it comes to gradient descent. Therefore, we use sigmoid transform

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

to relate θ to β:

$$\sigma(\beta) = \theta \in (0, 1);$$

Note that $\beta \in R$ is no longer restricted. Then we can simply do gradient ascent (we are maximising the likelihood) to find the ML estimator.

[1] $\leftarrow$ means assignment.

The relevant gradients are

$$\frac{d\mathcal{L}}{d\theta} = \frac{R}{\theta} - \frac{(m-R)}{1-\theta}; \frac{d\theta}{d\beta} \equiv \frac{d\sigma(\beta)}{d\beta} = \sigma(\beta)(1-\sigma(\beta)).$$

Therefore, the gradient is

$$\begin{aligned}\frac{d\mathcal{L}(\beta)}{d\beta} &= \left(\frac{R}{\sigma(\beta)} - \frac{(m-R)}{1-\sigma(\beta)}\right)\sigma(\beta)(1-\sigma(\beta)) = R(1-\sigma(\beta)) - (m-R)\sigma(\beta) \\ &= R - m\sigma(\beta)\end{aligned}$$

The gradient ascent step is:

$$\beta_{t+1} \leftarrow \beta_t + \alpha \frac{d\mathcal{L}}{d\beta}(\beta_t),$$

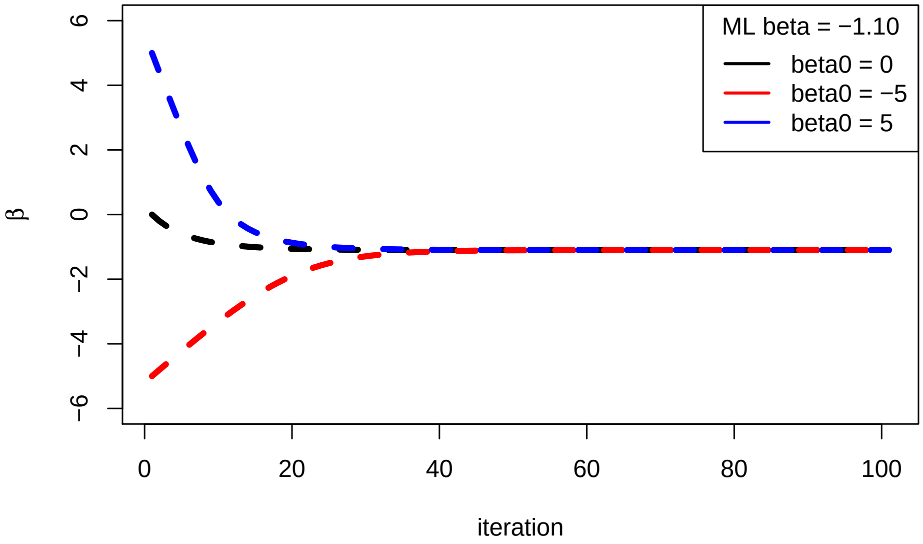
or

$$\beta_{t+1} \leftarrow \beta_t + \alpha \left(R - m\sigma(\beta_t)\right).$$

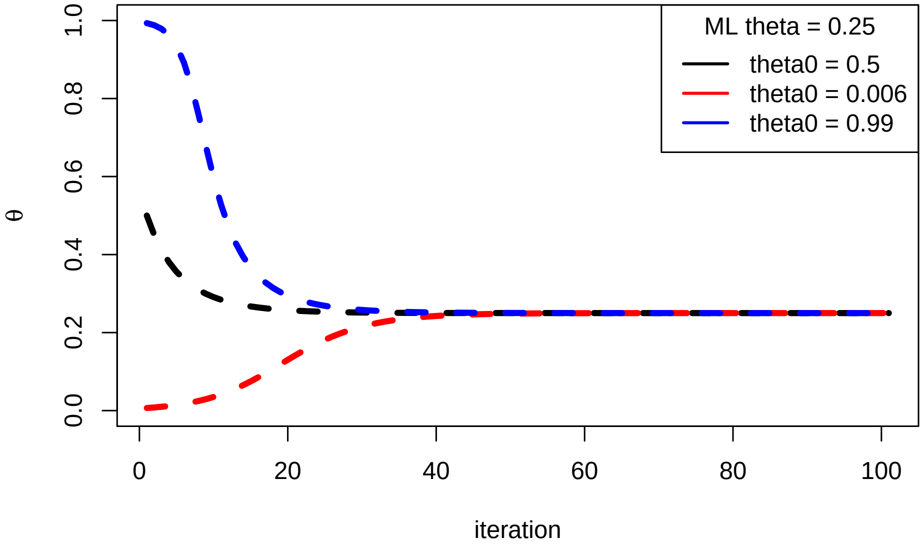
Lastly and most importantly, the main reason I introduce the trick is actually to show you the connection between this simple one parameter model and logistic regression. By applying the sigmoid trick, it is obvious logistic regression is just a generalisation of this simple model: one is $P(y^{(i)}) = \sigma(\beta)$ and the other is $P(y^{(i)}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)})$. The only difference is for logistic regression we have input $\boldsymbol{x}$, while for the Bernoulli model we only have y (actually x can be considered as dummy variable 1 or intercept). Of course, the technique itself is useful in general on its own merits.

You can run the following code in R to verify the algorithm indeed works. For the following experiment, $N = 80, R = 20$, we know the ML estimator should be $\theta_{ML} = R/N = 0.25$; the corresponding $\beta_{ML} = \sigma^{-1}(\theta_{ML}) = -1.1$.

```r
sigmoid <- function(x){
  return(1/(1+exp(-x)))
}

# gradient w.r.t. beta; direct translation
berGrad <- function(m,r, beta){
  sigValue <- sigmoid(beta)
  # (r/sigValue - (n-r)/(1-sigValue))*sigValue*(1-sigValue)
  return(r-m*sigValue)
}

# gradient ascent by using the gradient function above
berNDescent <- function(alpha, iter, m, r, beta0){
  beta <- beta0
  beta_history <- vector(mode="numeric", length = iter+1)
  beta_history[1] <- beta
  for(i in 1:iter){
    grad <- berGrad(m,r,beta)
    beta <- beta + alpha*grad
    beta_history[i+1] <- beta
  }
  return(beta_history)
}
par(mar=c(5,5,3,0.1)+.1)

iter=100
# some fake data
M <- 80;
R <- 20;
# ML estimator for theta is r/m
theta_ml = R/M;
# find the corresponding target for beta
inverseSigm <- function(theta){
  log(theta/(1-theta))
}
beta_ml = inverseSigm(theta_ml)

tr1 <- (berNDescent(0.01, iter, M, R, 0))
tr2<-(berNDescent(0.01, iter, M, R, -5))
tr3<-(berNDescent(0.01, iter, M, R, 5))
plot(tr1,type="l", ylim=c(-6,6), xlab="iteration",
     ylab=expression(paste(beta)), lty=2, lwd=4)
lines(tr2, col="red", lty=2, lwd=4)
lines(tr3, col="blue", lty=2, lwd=4)
labels <- c("beta0 = 0", "beta0 = -5", "beta0 = 5");
legendtitle <- sprintf("ML beta = %.2f", beta_ml)
legend("topright",  title=legendtitle,
  labels, lwd=2, lty=c(1, 1, 1), col=c("black", "red", "blue"))
```



```
# transform back to theta by using sigmoid
plot(sigmoid(tr1),type="l", ylim=c(0,1), xlab="iteration",
     ylab=expression(theta), lty=2, lwd=4)
lines(sigmoid(tr2), col="red", lty=2, lwd=4)
lines(sigmoid(tr3), col="blue", lty=2, lwd=4)
labels <- c("theta0 = 0.5", "theta0 = 0.006", "theta0 = 0.99");
legendtitle <- sprintf("ML theta = %.2f", theta_ml)
legend("topright",  title=legendtitle,
  labels, lwd=2, lty=c(1, 1, 1), col=c("black", "red", "blue"))
```



1.2 Gaussian likelihood

Here is another example with the case $\theta > 0$. For positive parameters, exponential function can transform any number on the real line, β, to a positive number:

$$\theta = f(\beta) = e^{\beta}.$$

Apply the chain rule, the new gradient is (as $\frac{d\theta}{d\beta} = e^{\beta}$):

$$\frac{d\mathcal{L}}{d\beta} = \frac{d\mathcal{L}}{d\theta}\frac{d\theta}{d\beta} = \frac{d\mathcal{L}}{d\theta} \cdot e^{\beta}$$

We consider the Gaussian likelihood model. The Gaussian log likelihood function is:

$$\begin{aligned}\mathcal{L}(\mu, \sigma^2) &= \log p(\mathcal{D}|\mu, \sigma^2) = \log \prod_{i=1}^{m} p(y^{(i)}; \mu, \sigma^2) = \sum_{i=1}^{m} \log p(y^{(i)}; \mu, \sigma^2) \\ &= \sum_{i=1}^{m} \left(-\frac{1}{2}\log(2\pi\sigma^2) - \frac{(y^{(i)} - \mu)^2}{2\sigma^2} \right) \\ &= -\frac{m}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y^{(i)} - \mu)^2.\end{aligned}$$

To simplify the problem, let us assume μ is known. The only unknown parameter is σ^2, the variance. Note that

$$\frac{d\mathcal{L}(\sigma^2)}{d\sigma^2} = -\frac{m}{2\sigma^2} + \frac{\sum_{i=1}^{m}(y^{(i)} - \mu)^2}{2(\sigma^2)^2};$$

apply the chain rule, we have the new gradient

$$\frac{d\mathcal{L}}{d\beta} = \frac{d\mathcal{L}(\sigma^2)}{d\sigma^2}e^{\beta} = \left(-\frac{m}{2e^{\beta}} + \frac{\sum_{i=1}^{m}(y^{(i)} - \mu)^2}{2(e^{\beta})^2} \right) e^{\beta} = -\frac{m}{2} + \frac{\sum_{i=1}^{m}(y^{(i)} - \mu)^2}{2e^{\beta}}.$$

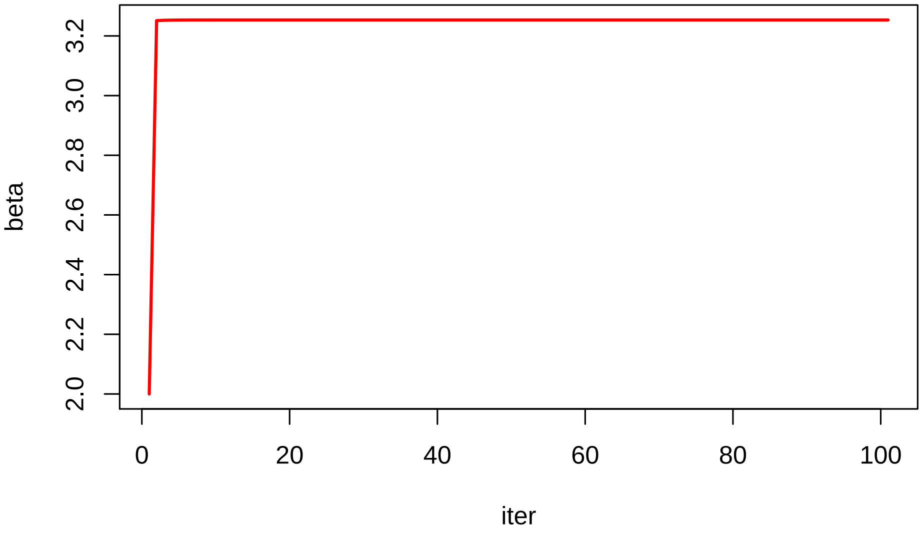
The gradient ascent step is:

$$\beta_{t+1} \leftarrow \beta_t + \alpha\frac{d\mathcal{L}}{d\beta}(\beta_t).$$

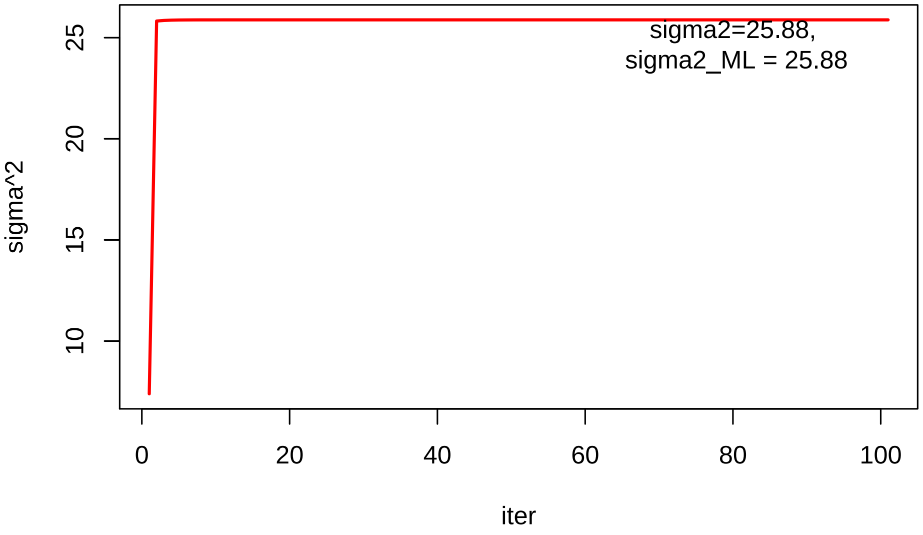
```
trueMu <- 0
trueSig2 <- 25
M <- 1000
# artificially generate some gaussian data
```

```r
# with mean and variance specified
Y<-rnorm(M, trueMu, sd= sqrt(trueSig2))
# sum of squared errors sum_{i}(yi-mu)^2
SSE <- sum((Y-trueMu)^2)
# closed form solution; to verify
# whether gradient ascent works
sigm2ML <- SSE/M

# calculate the gradient of L (gaussian loglikelihood)
# w.r.t reparameterised beta, where e^beta = sigma2
# m is total observation count
# sse is sum of squared error, here we assume mu is known
# beta is reparameterised parameter, and beta \in R
sigm2grad<- function(m, sse, beta){
  g<-(-m/(2*exp(beta))+sse/(2*(exp(beta)^2)))*exp(beta)
  return(g)
}

logLik <- function(m, sigm2, sse){
  return(-m/2*log(2*pi*sigm2) - 1/(2*sigm2)*sse)
}

# alpha is learning rate
# beta0 is the starting value
# m and sse are defined as before
gradientAscentForVariance <- function(iter, alpha, beta0, m, sse){
  beta <- beta0
  beta_history <- vector(mode="numeric", length = iter+1)
  log_lik_history <- vector(mode="numeric", length = iter+1)
  beta_history[1] <- beta
  log_lik_history[1] <- logLik(m, exp(beta), sse)
  for(i in 1:iter){
    grad <- sigm2grad(m, sse, beta)
    # gradient ascent for logLik rather than descent
    beta <- beta + alpha*grad
    beta_history[i+1] <- beta
    log_lik_history[i+1] <- logLik(m, exp(beta), sse)
    cat("The log likelihood at ", i, "iteration is: ",
        log_lik_history[i+1], "\n", sep = "")
  }
  # transform back to sigma2
  sigm2_history <- exp(beta_history)
  return(list(beta_history, sigm2_history, log_lik_history))
}
## starting value for beta
```

```
beta0<- 2
sigma2_0 <- exp(beta0)
result <- gradientAscentForVariance(100, 0.001, beta0, M, SSE)
plot.ts(result[[1]], col="red", type="l",
        xlab="iter", ylab="beta",  lwd=2)
```



```
plot.ts(result[[2]], col="red", type="l",
        xlab="iter", ylab="sigma^2", lwd=2)
label_string<- sprintf("sigma2=%.2f,\n sigma2_ML = %.2f",
                       result[[2]][iter], sigm2ML)
text(80, result[[2]][81]-1.2, label_string)
```


sigma^2
iter
sigma2=25.88,
sigma2_ML = 25.88