Machine Learning and Bayesian Inference

Dr Sean Holden

Computer Laboratory, Room FC06

Telephone extension 63725

Email: sbh11@cam.ac.uk

https://www.cl.cam.ac.uk/~sbh11

Question:

"In the iteratively reweighted least squares algorithm we use the Newton method, potentially because calculating the Hessian is tractable and we can calculate it explicitly.

According to a quick Google search, second-order methods are not used much for perceptron training (e.g., Adam is a first order method).

In theory, couldn't we calculate second derivatives with a backpropagation algorithm, or with the automated differentiation feature of dedicated languages such as PyTorch?

Would it be computationally too expensive to be worth the increased precision per iteration?"

Answer: part 1

Beware of the "quick Google search"!

- Numerous methods exploiting the matrix of second derivatives (the *Hessian*) have been used over several decades.
- We will see later an instance that requires them when the course discusses the use of *Bayesian inference* to estimate *error bars*.

The issue of *acknowledgement of prior art* is currently rather controversial in neural network research.

See:

https://people.idsia.ch/~juergen/scientific-integrity-turing-award-deep-learning.html

and the extensive discussion on:

https://mailman.srv.cs.cmu.edu/mailman/listinfo/connectionists

Question and Answer Session 1

Here is the relevant slide...

Iterative re-weighted least squares

The Newton-Raphson method *generalizes easily to functions of a vector*:

To minimize $E : \mathbb{R}^n \rightarrow \mathbb{R}$ iterate as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}^{-1}(\mathbf{w}_t) \left.\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}_t}.$$

Here the *Hessian* is the matrix of *second derivatives* of $E(\mathbf{w})$

$$\mathbf{H}_{ij}(\mathbf{w}) = \frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j}.$$

All we need to do now is to *work out the derivatives...*

To see what's going on, I'm going to show a sequence of slides that comes up later on...

Reminder: Taylor expansion

In *one dimension* the *Taylor expansion* about a point $x_0 \in \mathbb{R}$ for a function $f : \mathbb{R} \to \mathbb{R}$ is

$$\begin{aligned} f(x) \approx f(x_0) &+ \frac{1}{1!}(x - x_0)f'(x_0) \\ &+ \frac{1}{2!}(x - x_0)^2 f''(x_0) \\ &+ \cdots + \frac{1}{k!}(x - x_0)^k f^k(x_0). \end{aligned}$$

What does this look like for the kinds of function we're interested in? As an *example* We can try to approximate

$$\exp\left(-f(x)\right)$$

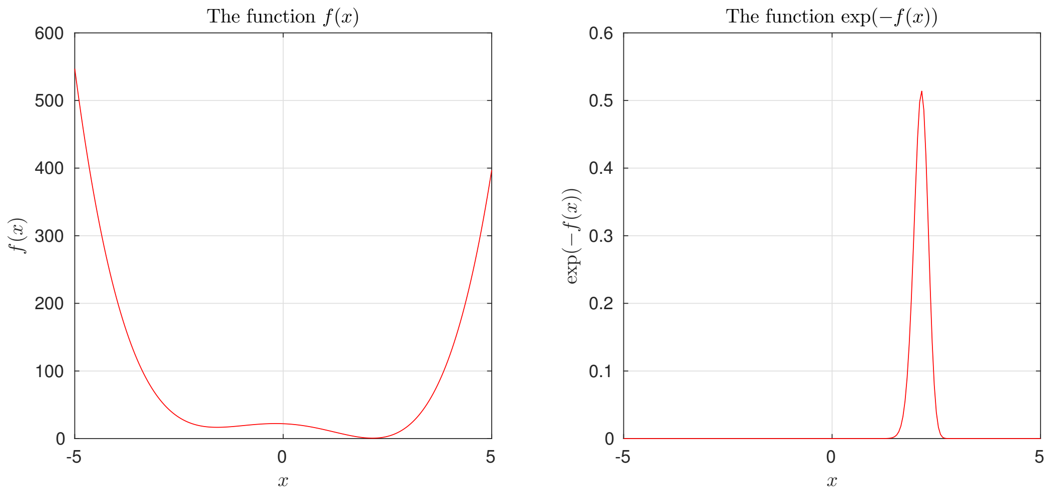
where

$$f(x) = x^4 - \frac{1}{2}x^3 - 7x^2 - \frac{5}{2}x + 22.$$

This has a *form similar to* $S(\mathbf{w})$, but in one dimension.

Reminder: Taylor expansion

The functions of interest look like this:



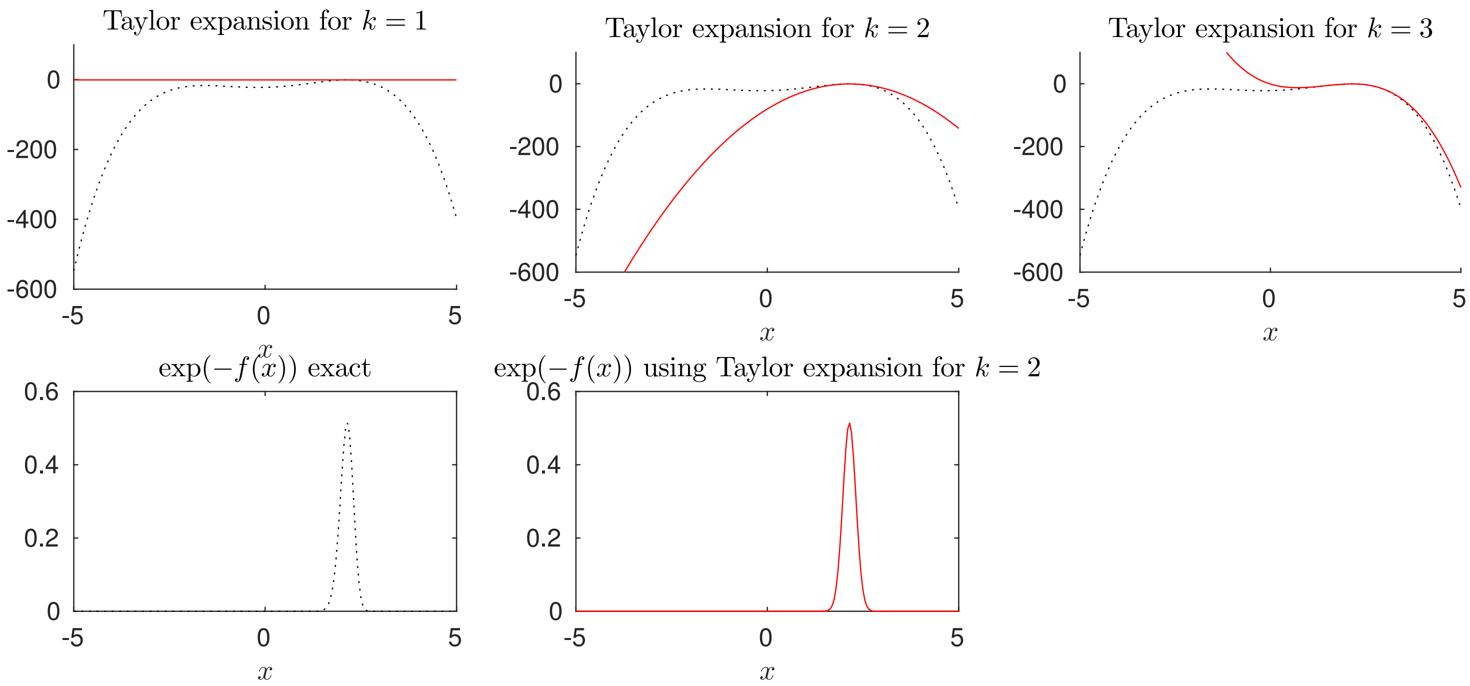
By replacing $-f(x)$ with its *Taylor expansion about its maximum*, which is at

$$x_{\max} = 2.1437$$

we can see what the *approximation to* $\exp(-f(x))$ looks like. Note that the $\exp$ *hugely emphasises peaks*.

Reminder: Taylor expansion

Here are the approximations for $k = 1$, $k = 2$ and $k = 3$.



The use of $k = 2$ looks promising...

Reminder: Taylor expansion

In *multiple dimensions* the Taylor expansion for $k = 2$ is

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \frac{1}{1!}(\mathbf{x} - \mathbf{x}_0)^T \left.\nabla f(\mathbf{x})\right|_{\mathbf{x}_0} + \frac{1}{2!}(\mathbf{x} - \mathbf{x}_0)^T \left.\nabla^2 f(\mathbf{x})\right|_{\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0)$$

where ∇ denotes *gradient*

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

and $\nabla^2 f(\mathbf{x})$ is the matrix with elements

$$M_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$$

(Looks complicated, but it’s just the obvious extension of the 1-dimensional case.)

Question and Answer Session 1

Getting back to business...

- Writing $\mathbf{H}$ for the Hessian evaluated at $\mathbf{w}_t$, the Taylor expansion of $E(\mathbf{w})$ at some point $\mathbf{w}_t$ is an *approximation*

$$E(\mathbf{w}) \simeq E(\mathbf{w}_t) + (\mathbf{w} - \mathbf{w}_t)^T \left.\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}_t} + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^T \mathbf{H}(\mathbf{w}_t)(\mathbf{w} - \mathbf{w}_t).$$

- What happens if I solve for the *minimum of the approximation*?
- Well, *differentiation the approximation*, set to 0 and solve.
- We get

$$\mathbf{w}_{\text{min}} = \mathbf{w}_t - \mathbf{H}^{-1}(\mathbf{w}_t) \left.\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}_t}.$$

Hooray—we've derived the *Newton update*!

So, what are the issues here?

- If $E(\mathbf{w})$ is *quadratic* then the Newton update goes *to the minimum* in a *single step*.
- If $E(\mathbf{w})$ is *convex* then you can *iterate* the Newton update.
- But, $E(\mathbf{w})$ is generally *not convex* for neural networks.

And there are other issues:

- The Hessian needs $O(d^2)$ space.
- Computing $\mathbf{H}^{-1}$ is $O(d^3)$.
- You need to compute $\mathbf{H}^{-1}$ *at every iteration*!
- *Saddle points* can be particularly problematic. (And numerically tricky...)

So for various reasons Newton updates in their basic form, and use of $\mathbf{H}$ in general, can be problematic.

Nonetheless, there have been many attempts to *gain the advantages without the drawbacks*:

- The *Conjugate Gradient Method*: do *line searches* in a *sequence of directions* that are, in a sense, *conjugate to one-another*.
- The *Broyden-Fletcher-Goldfarb-Shanno (BFGS) method*: use a *sequence of approximations* to $\mathbf{H}^{-1}$ that are computed using *efficient updates*.
- The *reduced memory BFGS method* and so on...