Universität des Saarlandes
Prof. Dr. Dietrich Klakow
Lehrstuhl für Signalverarbeitung
NNIA Winter Term 2019/2020

Exercise Sheet 8

Philip Georgis [s8phgeor], Pauline Sander [s8pasand], Vilém Zouhar [vizo00001]

(Solutions)

Deadline: 19. 1. 2021

Exercises

Exercise 8.1 - Possible Problems

a)

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx (\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^T \boldsymbol{H} \boldsymbol{g} \tag{1}$$

is the second order approximation of the Taylor series. The first part is the current value, the second part is the first derivative, the expected improvement of the value at $\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}$ compared to $\boldsymbol{x}^{(0)}$, assuming there is no curvature. The third part corrects for the second part by adding the curvature (second derivative) back in. However, when the Hessian matrix is ill-conditioned, the third term gets larger ($\boldsymbol{g}^T \boldsymbol{H} \boldsymbol{g}$ grows by an order of magnitude) than the second term and $f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g})$ will always be larger than the current value, no matter how small ϵ is.

If we have an ill-conditioned Hessian matrix the gradient norm will rise/stay high, even though the loss/cost decreases. (see Deeplearningbook p. 280) If we have an ill-conditioned learning rate, it can be that the loss does not change. After a large number of epochs the loss/cost can suddenly skyrocket. (see blogpost)

If the Hessian matrix is ill-conditioned, we can use Newton's method instead. Ill-conditioned learning rates can be dealt with by adaptive learning methods.

b) The gradient of the last layer of weights of a 100 layer Feed Forward Neural Network would be calculated the following way:

$$\frac{\partial L}{\partial RelU(l^{(100)})} \times \frac{\partial RelU(l^{(100)})}{\partial l^{(100)}} \times \frac{\partial l^{(100)}}{\partial RelU(l^{(99)})} \times ... \times \frac{\partial RelU(l^{(1)})}{\partial l^{(1)}} \times \frac{\partial l^{(1)}}{\partial W^{(1)}} \tag{2}$$

The derivative of RelU is either 0 or 1. If it is 0, the whole term will equate to 0. If it is 1 **and** the weights contain large values, there will be many terms ≥ 1 (in the weights) which will cause the gradients of lower layers to explode. In other words: Assuming we have the same layer repeated multiple times and the values are positive, then by eigenvalue decomposition this turns into $QL^{100}Q^{-1}$ and if there is an eigenvalue greater than 1, it will explode.
This might lead the network to take unwanted 'leaps' when updating those layers.
A way to avoid gradient explosion is gradient clipping. This means that we define a maximal gradient; all gradients larger than the maximum will be set back to this value.

c) Neural networks are non-identifiable models, because their hidden parts (weights) can be arranged in many different ways (weights can be swapped) but still arrive at the same output. They therefore have lots of local minima. However, these different solutions are mathematically equivalent. The local minima of mathematically equivalent models are identical, and therefore not problematic. This leaves us which a much smaller number of local minima.

   Only those local minima which have a higher cost than the global minimum are problematic. We know we have reached a critical point (such as a local minimum), when the gradient norm becomes very small. It is unknown how big of a problem such minima are in the everyday training of neural networks.

Exercise 8.2 - Batch Size

a) Stochastic GD is quite easy to implement (input is just one vector of features, not a matrix) and is very fast, but can fail to converge to the best result, because we are estimating the loss on all of training data with just one example. When we take the step after computing gradient on all examples (batch GD), we are not approximating and with reasonable learning rate, every step lowers the loss. This is however unfeasible, even for medium-sized models with real-world data (such as MT). A compromise is to use some number m of examples in between (minibatch GD). This still provides accurate results, but the whole minibatch fits in the memory. Multiple GPUs can be used to split the whole batch into smaller minibatches that are computed on each card and then combined ((a-)synchronous SGD).

b) If we did not shuffle the data, there could be systematic differences (biases) in the distributions of the examples in minibatches. This would lead to poor approximation of the train data gradient. When it is shuffled, the expected value is exactly the train data gradient. This is obviously not necessary for batch GD.

c) Simple well defined and explored operations (e.g. matrix multiplication) can make use of the GPU power, as it can be very well optimized especially for larger batches. More complex operations or models, which are not optimized by the library and e.g. access some global mutable state are impossible to run on a GPU and require the standard CPU/RAM model.

Exercise 8.3 - SGD with Momentum

Standard stochastic gradient descent often exhibits the defect of getting stuck at stationary points (which may be saddle points or local minima). Because the gradient is zero at such points, the update step is unable to continue moving in any direction – the next step is equivalent to the current step. Adding a momentum term to the update step alleviates this problem by also taking into account the averaged gradient at previous time steps, which has the effect of promoting acceleration in dimensions whose gradients consistently point in the same direction and slowing down progress in dimensions whose gradients change directions. This additionally reduces oscillation back and forth, and thus allows for faster convergence. Whereas vanilla SGD would be unable to make further progress once a strict saddle point has been reached (i.e., it gets stuck) in a non-convex function, SGD with momentum will continue to move in the direction where progress has been most consistent prior to that point, thus using previous gradient information (= "momentum") to traverse the saddle point and

reach a point where the gradient is once again non-zero.

As an analogy, consider a person walking down a slope, taking the path of steepest descent at each step, versus a ball rolling down a slope. If the person stops after each step and considers the direction of the next step to take, no momentum will have been accumulated and the person will simply stop walking once they reach a flat surface. In contrast, the ball continually gains speed as long as the direction of the slope is roughly consistent, and will continue moving in this direction for some time even once at the flat surface.