Exercise Sheet 8

Zena Al Khalili (7009151)
Sangeet Sagar (7009050
{zeal00001,sasa00001}@stud.uni-saarland.de

January 19, 2021

Exercise 8.1 - Possible Problems

a)

We know that second derivative tells us how well we can expect a gradient descent step to perform, so we take Tylor's series approximation of a cost function f(x):

$$f(x) \approx f(x_0) + (x - x_0)^T g + \frac{1}{2}(x - x_0)^T H(x - x_0)$$

if we use a learning rate epsilon then:

$$x = x_0 - \epsilon g$$

and the approximation of cost function will be:

$$f(x_0 - \epsilon g) \approx f(x_0) - \epsilon g^T g + \frac{1}{2}\epsilon^2 g^T H$$

The second term, is the expected improvement on the cost function value. the third term of the above approximation is the correction we must apply to account for the curvature of the function. When the correction amount is too big exceeding the improvement amount, the gradient step can move uphill causing large changes to cost function.
To spot ill-conditioning:

- loss values do not seem to decrease.
- NaN values start to appear in loss function values.

To solve ill-conditioning:

- use Adaptive Learning Methods like Adam, Adadelta or SGD with annealing.
- pre-processing of input. e.g: normalizing pixel values of a image.

b)

The phenomena of exploding gradient happens in very deep feed forward Neural Network when we multiply by a matrix after each time step t.
The problem happens exactly in the backward pass in the following equation:

$$\frac{\partial J}{\partial W} = \sum_{i=0}^{T} \left(\prod_{i=k+1}^{y} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

in the term:

$$\prod_{i=k+1}^{y} \frac{\partial h_i}{\partial h_{i-1}}$$

When it's greater than the absolute value of 1 in magnitude, the term goes to infinity exponentially fast, and its value becomes a NaN due to the unstable process, causing the phenomena of exploding gradient. Exploding gradient makes learning unstable.
To overcome the problem of exploding gradient there are 2 ways:

- Truncated Back Propagation: During the back propagation process, we run forward and backward through a "Window" called chunk of a specific size instead of the entire Network. However, this methods can cause the problem of vanishing gradient.
- Gradient Clipping: this methods works by introducing a threshold that clips high magnitudes of the gradient using the following formula:

$$clip(g, threshold) = g.max\left(1, \frac{threshold}{\|g\|}\right)$$

  Where g is the gradient.
  This method doesn't change the direction of the gradient, but its magnitude.

c)

Deep Neural Networks have non-convex cost functions, because of the model non-identifiablity, which means these cost functions will have infinite amount of local minima. However, not all local minima are problematic because they are equivalent to each other in cost function values. Local minima become problematic when they have high value of cost function in comparison to global minima. When having a critical point with low value of cost function it will be a local minima, when having a critical point with high value of cost function it's a saddle point, and when having a critical point with extremely large value of cost function it will be a local maxima.

Exercise 8.2 - Batch Size

a)

Stochastic gradient descent or SGD is a common optimization tool used to optimize iteratively optimize given object function.

- **Advantages**:
  - This is faster to train on large datasets as the parameter updates are done more frequently.
  - It is quite a suitable option for noisy data as it samples an example (a batch size of 1) to compute the gradient.
- **Disadvantages**:
  - Since SGD performs updates frequently, it cause the objective function to fluctuate a lot and this in turn leads to a slower rate of convergence.
  - These frequent updates are computationally expensive since it samples one example in an iteration.

Batch gradient descent (or vanilla gradient descent), computes the gradient of the cost function with respect to the entire training data:

- **Advantages**:
  - In terms of convergence, it is more stable compared to SGD as the gradients are computed for each training example but the update is made only after the entire training set is evaluated
  - Since the updates are made after all training examples are evaluated, vectorization can be used.
  - Since the updates are less frequent, gradient descent computation is computationally efficient.
- **Disadvantages**:
  - The updates at the end of a training iteration is computationally expensive as it needs to update all the prediction errors that has accumulated over the course of an iteration.
  - Since the updates are stable, it can result in an un-optimized convergence at random local minima or saddle point.

Mini-batch stochastic gradient descent (mini-batch SGD) is quite similar to batch gradient descent except the training data is split into batches and the gradients are computed over these small batches.

- **Advantages**:
  - The updates are more than batch gradient descent but less than SGD, hence we have a more optimized convergence.
  - In terms of computational powers, this is more efficient than the other two techniques discussed above.
- **Disadvantages**:
  - Mini batch raises the need for an additional mini-batch size hyperparameter.

b)

It is important to shuffle the data after each iteration because we want all the batches in an iteration to have the representation and flavor of the entire dataset. Shuffling decreases variance and is likely to increase bias thus reducing the tendency to overfit the data.

c)

The training batch size has a huge impact while training a neural network. We can either choose a small batch size or a large batch size and both have their pros and cons depending upon the system we are training. Smaller batch size converges faster and makes it easier to fit in the GPU memory, however, the downside is that model is not guaranteed to converge to a global optimum. While on the other hand, larger batch sizes make larger gradient steps than smaller batch sizes. This improves the performance by reducing the communication overhead caused by moving the training data to the GPU. Hence the larger the batch (up to a certain extent; 32 is by default a good value), the more effectively the GPU can run.

Now the question do we need a CPU or GPU? Training neural network in CPU is the suitable option when we have a relatively smaller dataset, while we use GPUs for a larger dataset. A CPU is composed of just a few cores that are designed to handle a wide range of tasks quickly whereas GPUs have a large number of simple cores that allow parallel computing through thousands of threads computing at a time. This exactly fits our need for training a neural network on a large dataset.

Exercise 8.3 - SGD with Momentum

With SGD, if the fluctuations occur only along one dimension, with the steps getting smaller and smaller, it would converge at the saddle point. SGD can sometimes break out of simple saddle points, if the fluctuations are along other directions, and if the step size is large enough for it to go over the flatness. But sometimes the saddle regions can be fairly complex.

The SGD with Momentum introduce the velocity that is the sum of previous gradients, with a decay element. Velocity is the direction and speed at which a parameter moves through parameter space. This speed will help gradient move faster (because of all the momentum it accumulates) and overcome local minima and flat regions (because the momentum may propel it out of a local minimum) to reach global minima.