10 Gradient Descent

Many problems in machine vision entail optimization, i.e., the minimization of an objective function $E(\mathbf{w})$ with respect to a parameter vector $\mathbf{w}$:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w}) . \tag{10.1}$$

In machine learning, this optimization is normally a data-fitting objective function, but similar problems arise throughout computer science, numerical analysis, physics, finance, and many other fields.

In simple cases one can derive a closed-form expression for the solution, but in many cases no such solution for the minimum exists. The solution we will use in this course for such problems is called **gradient descent**. It works for any differentiable energy function. However, it does not come with many guarantees – it is only guaranteed to find a local minima in the limit of infinite computation time.

Gradient descent is iterative. We begin with an initial estimate, $\mathbf{w}_1$, of the unknown parameter vector. How we obtain this vector depends on the problem. One approach is to randomly-sample values for the parameters. Another is to find a solution to a closely related problem which does have a closed-form solution.

From this initial estimate, given that we want to find the parameter vector that minimizes the objtective, it is natural to adjust the parameter values so that the objective decreases. The direction of steepest descent from $\mathbf{w}_1$ is given by negative gradient $-\nabla E$ of the objective function evaluated at $\mathbf{w}_1$. The gradient is defined as a vector of derivatives with respect to each of the parameters:

$$\nabla E \equiv \begin{bmatrix} \frac{dE}{dw_1} \\ \vdots \\ \frac{dE}{dw_N} \end{bmatrix} \tag{10.2}$$

The key point is that, if we follow the negative gradient direction for a sufficiently small distance, *the objective function is guaranteed to decrease*. (This can be shown by considering a Taylor-series approximation to the objective function about the current estimate of the parameter vector).

It is easiest to visualize this process by considering $E(\mathbf{w})$ as a surface parameterized by $\mathbf{w}$. We are trying to find the deepest pit in the surface. We do so by taking a series of small downhill steps, each time in the negative gradient direction from wherever we are on the surface. The entire process, in its simplest form, can be summarized as follows:

```
pick initial value $\mathbf{w}_1$
$i \leftarrow 1$
loop
    $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \lambda \nabla E|_{\mathbf{w}_i}$
    $i \leftarrow i + 1$
end loop
```

This process depends on three key choices, namely, the initialization, the termination condition, and the step-size λ. For the termination condition, one can run until a preset number of steps has elapsed, or monitor convergence. For example, one can test to see how much the objective has decreased as a consequence of the last iteration:

$$|E(\mathbf{w}_{i+1}) - E(\mathbf{w}_i)| < \epsilon \,, \tag{10.3}$$

for some preselected constant ϵ. Or one could terminate when either condition is met.

The simplest way to determine the step-size λ is to pick a single value in advance; this approach is often taken in practice. However it is somewhat unreliable. If we choose a step-size that is too large, then the objective function might actually get worse after some steps. If the step-size is too small, then the algorithm will take a very long time to make significant progress.

One solution is to use **line search**. That is, at each step, search along a single direction for the step-size that reduces the objective function as much as possible. A simple gradient search with line search procedure is as follows:

```
pick initial value $\mathbf{w}_1$
$i \leftarrow 1$
**loop**
    $\Delta \leftarrow \nabla E|_{\mathbf{w}_i}$
    $\lambda \leftarrow 1$
    **while** $E(\mathbf{w}_i - \lambda\Delta) \geq E(\mathbf{w}_i)$
        $\lambda \leftarrow \lambda/2$
    **end while**
    $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \lambda\Delta$
    $i \leftarrow i + 1$
**end loop**
```

A more sophisticated approach is to reuse step-sizes between iterations:

```
pick initial value $\mathbf{w}_1$
$i \leftarrow 1$
$\lambda \leftarrow 1$
**loop**
    $\Delta \leftarrow \nabla E|_{\mathbf{w}_i}$
    $\lambda \leftarrow 2\lambda$
    **while** $E(\mathbf{w}_i - \lambda\Delta) \geq E(\mathbf{w}_i)$
        $\lambda \leftarrow \lambda/2$
    **end while**
    $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \lambda\Delta$
    $i \leftarrow i + 1$
**end loop**
```

There are many, many more advanced methods for numerical optimization. For machine learning problems there are a host of methods that have been successful, including the use of gradient momentum and approximations to the Hessian of the objective.

For unconstrained optimization, the L-BFGS-B library is very useful. It is available for download on the web; it's written in Fortran, but there are wrappers for various languages out there. This method will be vastly superior to gradient descent for most problems.

10.1 Finite Differences

The gradient of any function can be computed approximately by numerical computations. This is useful for debugging your gradient computations, and in situations where it's too difficult or tedious to implement the complete derivative. The numerical approximation follows directly from the definition of derivative:

$$\left.\frac{dE}{dw}\right|_{w} \approx \frac{E(w+h)-E(w)}{h}\,, \tag{10.4}$$

for some suitably small stepsize h. Computing this value for each element of the parameter vector gives you an approximate estimate of the gradient ∇E.

It is strongly recommend that you use this method to debug your derivative computations; many errors can be detected this way! (This test is analogous to the use of "assertions"). *Automatic differentiation* is also very helpful.