9 Gradient Descent

There are many situations in which we wish to minimize an objective function with respect to a parameter vector:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w}) \tag{1}$$

but no closed-form solution for the minimum exists. 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.

The solution we will use in this course 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. First, we obtain 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. Then, from this initial estimate, we note that the direction of steepest descent from this point is to follow the 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:

$$\nabla E \equiv \begin{bmatrix} \frac{dE}{dw_1} \\ \vdots \\ \frac{dE}{dw_N} \end{bmatrix} \tag{2}$$

The key point is that, if we follow the negative gradient direction in a small enough distance, *the objective function is guaranteed to decrease*. (This can be shown by considering a Taylor-series approximation to the objective function).

It is easiest to visualize this process by considering $E(\mathbf{w})$ as a surface parameterized by $\mathbf{w}$; we are trying to finding the deepest pit in the surface. We do so by taking small downhill steps in the negative gradient direction.

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**

Note that this process depends on three choices: the initialization, the termination conditions, and the step-size λ. For the termination condition, one can run until a preset number of steps has elapsed, or monitor convergence, i.e., terminate when

$$|E(\mathbf{w}_{i+1}) - E(\mathbf{w}_i)| < \epsilon \tag{3}$$

for some preselected constant ϵ, or terminate when either condition is met.

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

The solution is to use **line search**, namely, at each step, to search for a step-size that reduces the objective function as much as possible. For example, a simple gradient search with line search procedure is:

pick initial value $\mathbf{w}_1$
$i \leftarrow 1$
loop
  $\Delta \leftarrow \nabla E|_{\mathbf{w}_i}$
  $\lambda \leftarrow 1$
  while $E(\mathbf{w}_i - \lambda\Delta) \geq E(\mathbf{w}_i)$
    $\lambda \leftarrow \frac{\lambda}{2}$
  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 $E(\mathbf{w}_i - \lambda\Delta) \geq E(\mathbf{w}_i)$
    $\lambda \leftarrow \frac{\lambda}{2}$
  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 unconstrained optimization, I recommend the L-BFGS-B library, which is available for download on the web. It is written in Fortran, but there are wrappers for various languages out there. This method will be vastly superior to gradient descent for most problems.

9.1 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:

$$\left.\frac{dE}{dw}\right|_w \approx \frac{E(w+h) - E(w)}{h} \tag{4}$$

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").

Aside:
The term **backpropagation** is sometimes used to refer to an efficient algorithm for computing derivatives for Artificial Neural Networks. Confusingly, this term is also used to refer to gradient descent (without line search) for ANNs.

