1 Line Search

In order to find the right step size which minimize the loss function, a line search has to be implemented.

The step size alpha is nothing more than a scalar: the learning rate for the conjugate gradient algorithm, which tells how far is right to move along a given direction.

So, fixed the values of the weights $\mathbf{W}$ and the descent direction $\mathbf{d}$, the main goal is to find the right value for α that is able to minimize the loss function:

$$\min_{\alpha} \quad \mathcal{E}(\mathbf{W} + \alpha \mathbf{d}). \tag{1}$$

Of course, we have to deal with a tradeoff: we want a good reduction, but we can't spend too much time computing the exact value for the optimum solution. So, the smarter way to get it is to use an inexact line search, that try some candidate step size and accepts the first one satisfying some conditions.

This search is performed in two phases:

- a *bracketing phase*, that finds an initial interval containing a minimizer;
- an *interpolation phase* that, given the interval, finds the right step length in it.

We decided to use one of the most popular line search condition: the *Armijo-Wolfe* condition.

The search for the better α is led by two condition:

- the *Armijo* one:

  $$\mathcal{E}(W_k + \alpha_k d_k) \leq \mathcal{E}(W_k) + \sigma_1 \alpha \nabla \mathcal{E}_k^T d_k \tag{2}$$

  which ensure that α gives a sufficient decrease of the objective function, being this reduction proportional to the step length α and the directional derivative $\nabla \mathcal{E}_k^T d_k$.

  The constant σ_1 has been set $\sigma_1 = 10^{-4}$, since it is suggested in literature to be quite small.

- the *Strong Wolfe* condition:

  $$|\nabla \mathcal{E}(W_k + \alpha_k d_k)^T d_k| \leq \mathcal{E}(W_k) + \sigma_2 |\nabla \mathcal{E}_k^T d_k| \tag{3}$$

  which garantees to choose steps whose size is not too small.

  It is also known as curvature condition and ensures that, moving of a step α along the given direction, the slope of our function if greater than σ_2 times the original gradient (if the slope is only slightly negative, the function cannot decrease rapidly along that direction, so it's better to stop the search).

  In this case, the constant σ_2 is equal to 0.1, since a smaller value gives a more accurate line search. Futhermore, having choosen the strong condition, which doesn't allow the derivative to be too positive, we are sure that the α found lies close to a stationary point of the function.

The algorithm satisfing the Strong Wolfe conditions is implemented through three functions, as described in the pseudocodes 1, 2, 3: `line_search`, `zoom` and `interpolate_alpha`.

Since two consecutive values may be similar in finite-precision arithmetic, we set a threshold in both the `line_search` and `interpolate_alpha` functions, which garantees that the algorithm stops if two values of α are too close or if the maximum number of iterations has been reached.

The `line_search` function try to find and return a good α; if it fails, it returns an interval in which continue the searching, invoking the `zoom` function, which decreases the size of the interval, until it finds and returns a good step length.

`Zoom` invokes another function, `interpolate_alpha`, which is nothing more than the implementation of a bisection interpolation in order to find a trial α inside the given interval.

Algorithm 1 Line Search

procedure LINE_SEARCH

  $\alpha_0 \leftarrow 0$;
  $i \leftarrow 1$;
  while $i \leq max\_iter$ **do**
    Evaluate$\mathcal{E}(\alpha_i)$;
    if $[\mathcal{E}(\alpha_i) > \mathcal{E}(0) + \sigma_1 \alpha_i \nabla \mathcal{E}_0^T d_0]$ or $[\mathcal{E}(\alpha_i) \leq \mathcal{E}(\alpha_{i-1})$ and $i > 1]$ **then**
      $\alpha_* \leftarrow$ **zoom**(α_{i-1}, α_i); **return** α_*;
    Evaluate $\nabla \mathcal{E}_i$
    if $|\nabla \mathcal{E}_i| \leq -\sigma_2 \nabla \mathcal{E}_0^T d_0$ **then**
      $\alpha_* \leftarrow \alpha_i$; **return** α_*;
    if $\nabla \mathcal{E}_i \geq 0$ **then**
      $\alpha_* \leftarrow$ **zoom**(α_i, α_{i-1}); **return** α_*;
    if $(|\mathcal{E}_i - \mathcal{E}_{i-1}| \leq threshold$ **then**
      $\alpha_* \leftarrow \alpha_i$ **return** α_*;
    Choose $\alpha_{i+1} \in (\alpha_i, \alpha_{max})$;
    $i \leftarrow i + 1$;

Algorithm 2 Zoom

```
procedure ZOOM
    repeat
        α_j ← interpolate_alpha(α_lo, α_hi);
        Evaluate E(α_j);
        if [E(α_j) > E(0) + σ_1 α_j ∇E_0^T d_0] or [E(α_j) ≤ E(α_lo)  then
            α_hi ← α_j;
        else
            Evaluate ∇E_j^T d_j;
            if  |∇E_j^T d_j| ≤ −σ_2 ∇E_0^T d_0 then
                α_* ← α_j; return α_*;
            if  ∇E_j^T d_j(α_hi − α_lo) ≥ 0  then
                α_hi ← α_lo;
            if  (|E_j − E_0| ≤ threshold  then
                α_* ← α_j return α_*;
        α_lo ← α_j;
```

Algorithm 3 Interpolate

```
procedure INTERPOLATE_ALPHA
    i ← 1;
    while i ≤ max_iter do
        α_mid ← (α_hi − α_lo)/2
        Evaluate E(α_mid);
        if  [E(α_mid) == 0] or [(α_hi − α_lo)/2 < threshold]  then return α_mid;
        Evaluate E(_midalpha_lo);
        if  sign(E(α_mid)) == sign(E(α_lo))  then
            α_lo ← α_mid;
        else
            α_hi ← α_mid;
        i ← i + 1;
=0
```