Gradient descend

To understand how weights are adjusted in neuronal networks, its important to understand the mathematical foundation of gradient descent. The simplest form of gradient descent is implemented by Newton's method.

$$x_{k+1} = x_k - \frac{f'(x_k)}{f(x_k)}$$

Each step, the index is moved in the direction of the negative gradient, eventually approaching a local minima. While conditions exist under which the method does not converge, it typically approaches a local minima rather quickly as shown in Figure 1.

```rust
fn f(x: f64) -> f64 {
    x.powi(2)
}

fn f_deriv(x: f64) -> f64 {
    2.0 * x
}

fn main() {
    let mut x = 1.5;
    for _ in 0..10 {
        let y = f(x);
        println!("f({x}) = {y}");

        x -= y / f_deriv(x); // Adjust x
    }
}
```

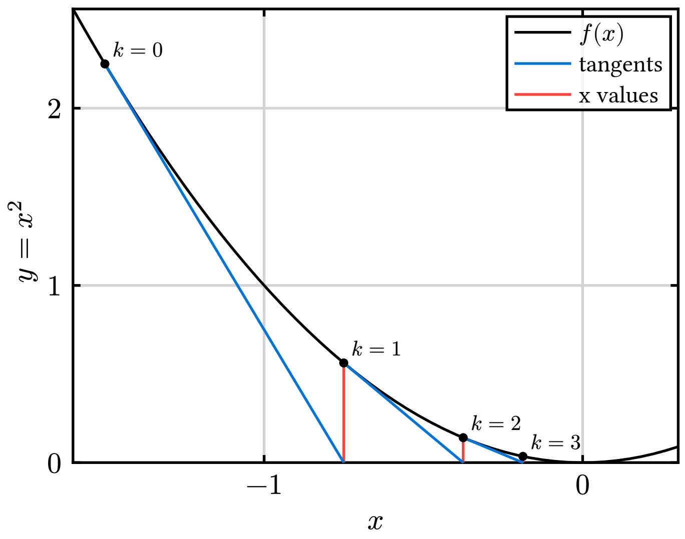

Figure 1: Descend of Newton's method for $f(x) = x^2$ starting from $x_0 = 1.5$

For machine learning however, there are typically many parameters in contrast to the single parameter that can be optimized with Newton's method. Yet, the underlying mechanism doesn't change. Instead, partial derivatives are calculated to form a gradient, which describes the multi-dimensional vector of the greatest slope at a given point. Adjusting the parameters by applying the negative gradient will move them towards a local minimum. For two parameters, this can be visualized in three-dimensional space as shown in Figure 2.

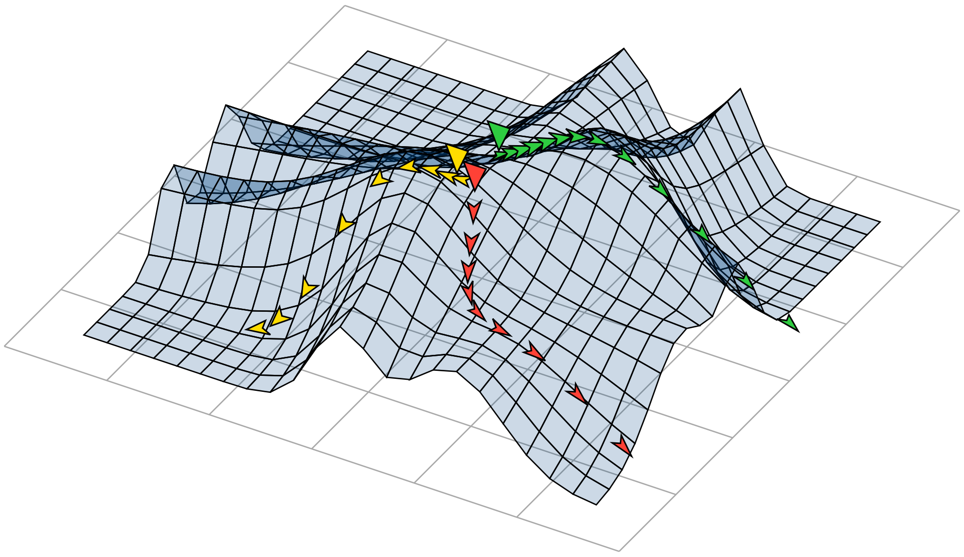
Figure 2: Visualization of gradient descend in three-dimensional space at different starting points