Loss Functions and Optimization

A *Loss Function* is a function that penalizes incorrect prediction. The best fit *minimizes* the loss function.

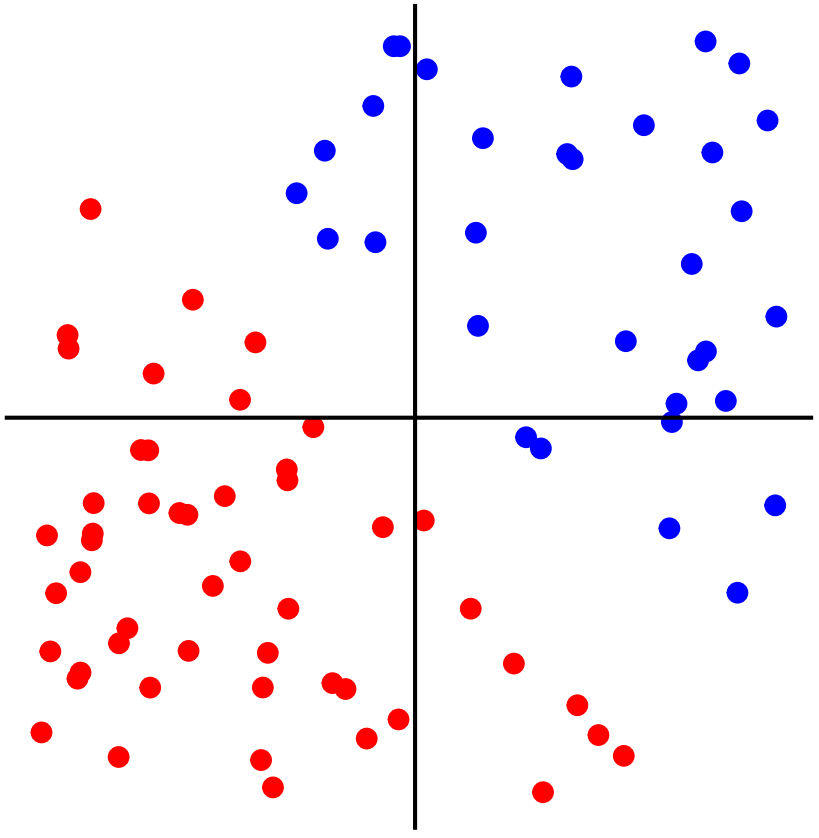
- For linear regression, we choose the β's to minimize the sum of squared residuals:

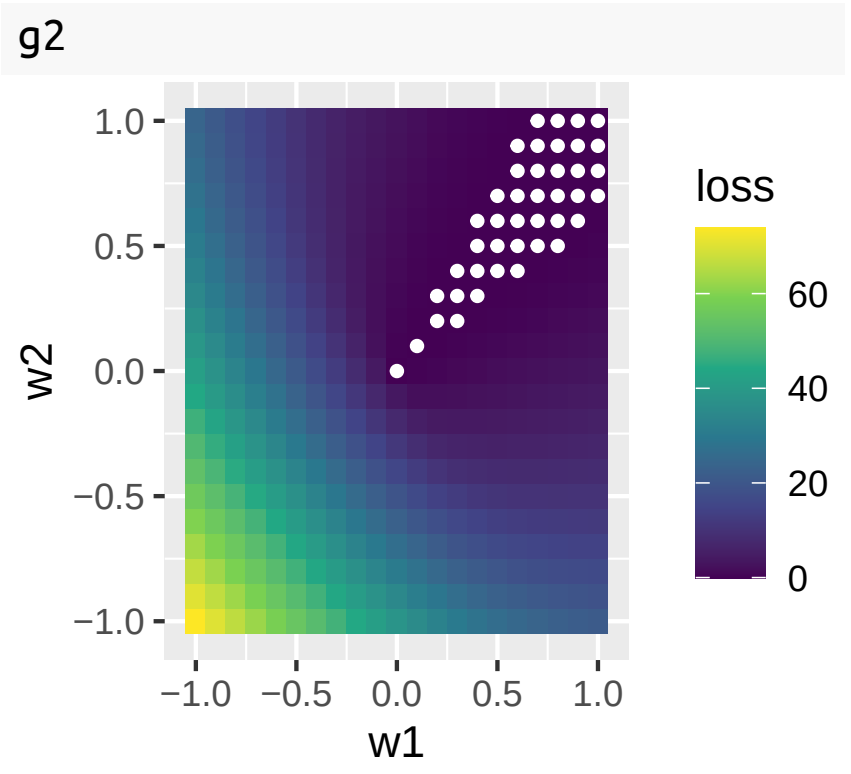
$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

- For the perceptron, we might choose $\vec{w}$ to minimize

$$-\sum_{i=1}^{n} (y_i - f(\vec{x}_i))(\vec{w} \cdot \vec{x}_i).$$

For linerly separable regions, the perceptron minimizes its loss function at many values of $\vec{w}$.

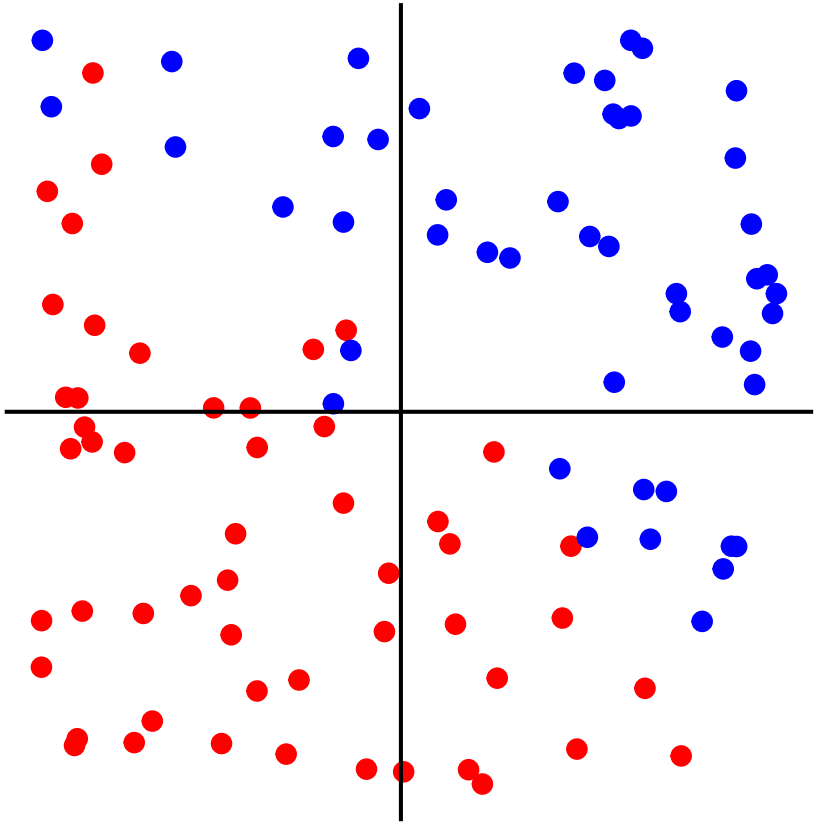


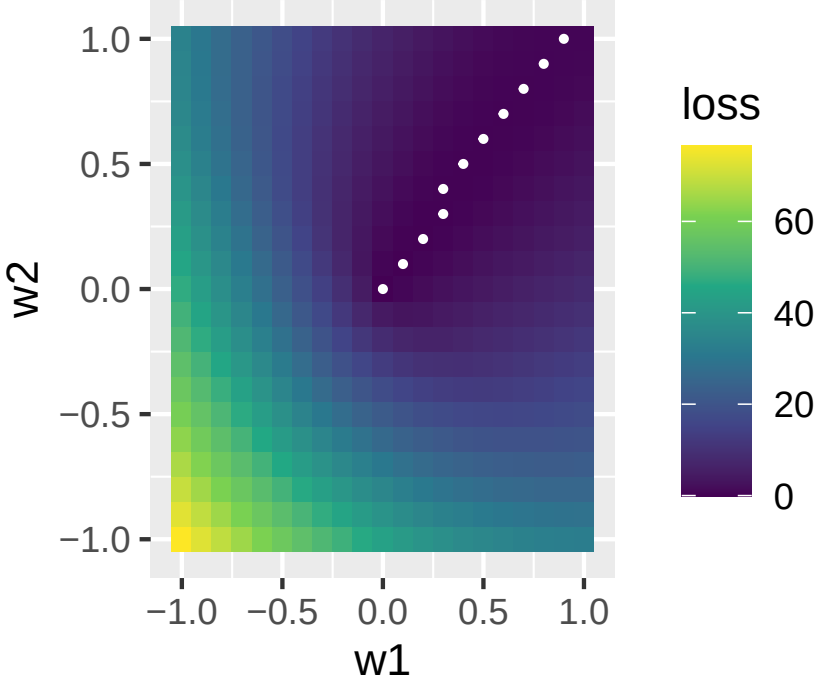


Note: Dot (•) indicates loss exactly 0.

Optimization problems often require a constraint on parameters to arrive at a unique solution.

- Only $\vec{w}$'s direction matters; so constrain the length of $\vec{w}$ to be 1.
- For overlapping regions, we do get a unique dividing line!





Note: * indicates minimum loss at that distance from the origin.

Methods of Optimization

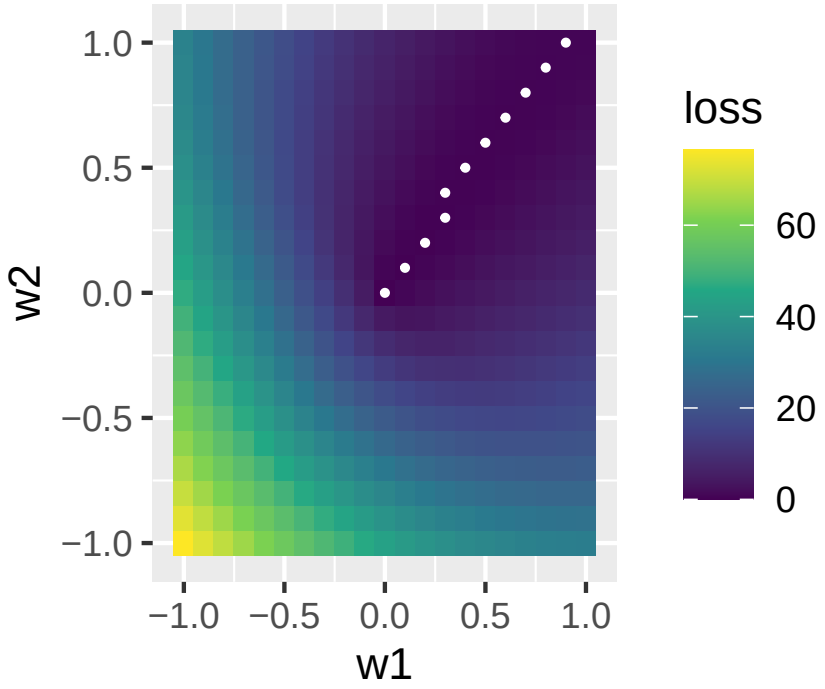
Outside the realms of calculus, no method of optimization is perfect. Here are a few important types:

- Grid search
- Gradient descent
- Nelder-Mead
- Simulated annealing

Those who implement practical and efficient methods of optimization are heroes. It's easy to describe these methods in general terms, but the details are anything but easy!

Grid search creates a set of pre-defined parameter values and searches those for a minimum.

- We just did that!
- Pro: Relatively easy.
- Con: Misses details if your grid is too coarse, and computationally intensive if your grid is too fine.

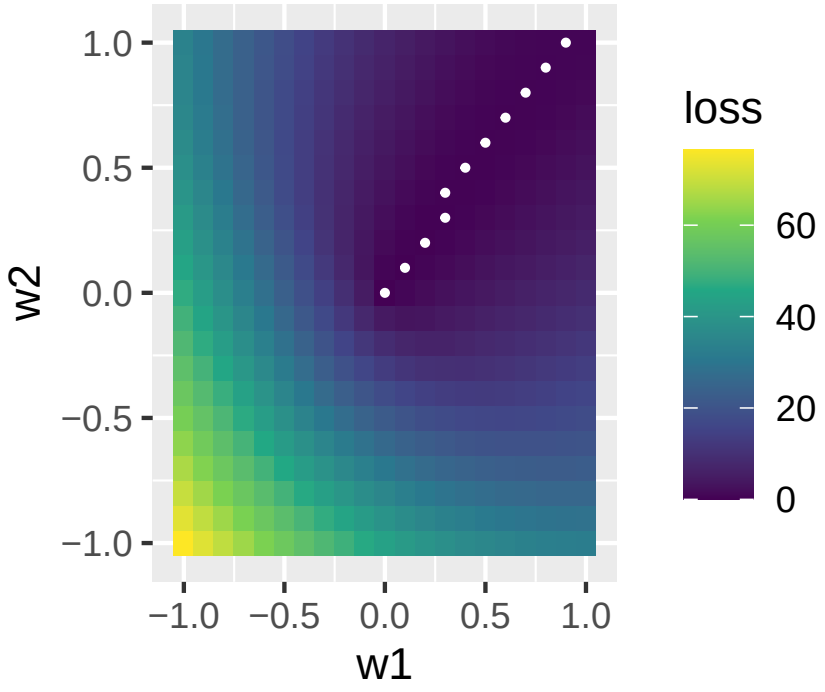


Gradient descent picks a starting point, then moves in the direction of steepest descent.

- Pro: Will reliably find local minimums.
- Con: Requires ability to calculate the slope of your "surface."
  - A "surface" specified or approximated by a well-defined formula.
  - A little calculus…
- Con: Won't always find global minimum.
- You can try several starting points and take the smallest of the results.

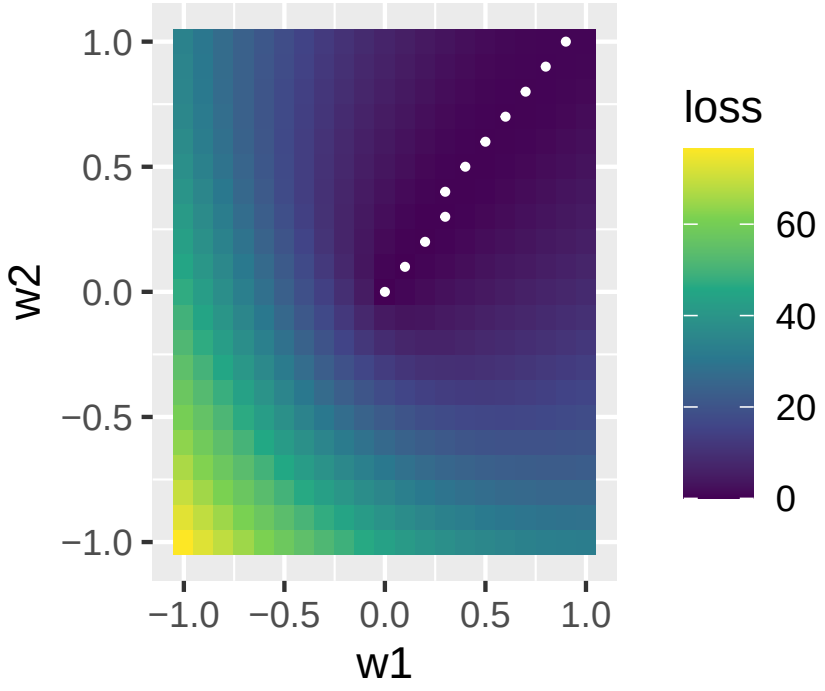
Stochastic gradient descent performs gradient descent using a random subset of the data.

- Our perceptron used a form of stochastic gradient descent.
- At each step, only *one* data point was used to move the direction vector toward an optimum value.
- Pro: Computationally less intense for large data sets, compared to full gradient descent.



The Nelder-Mead method uses a moving polytope to search out minimum values.

- Uses a *simplex* with n+1 vertices if the parameter space is n-dimensional.
- Through a series of flips, expansions and contractions, the simplex searches to contain the minimum point.
- The “amoeba” method.

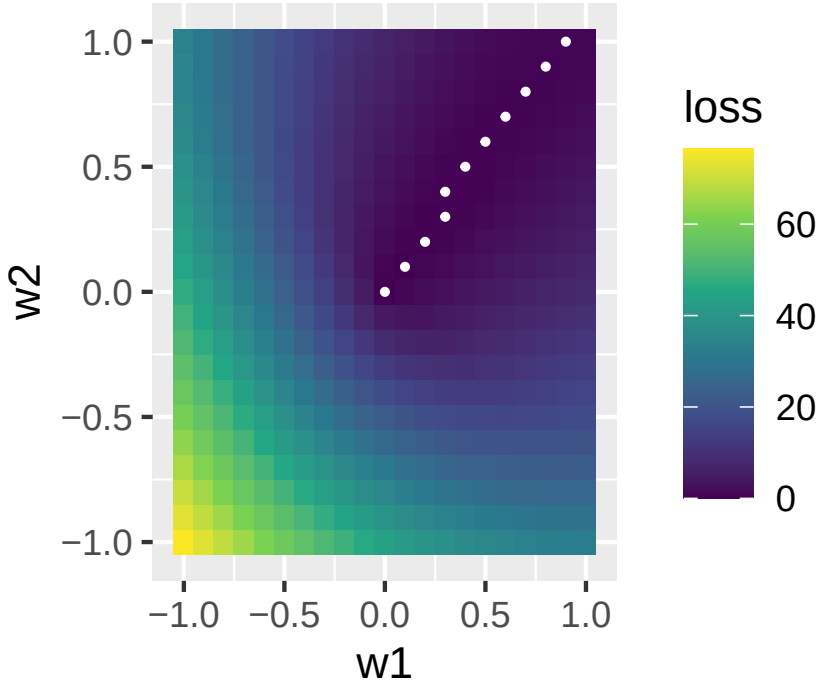


Simulated annealing allows random parameter jumps at high "temperature" and favors safe jumps at lower T.

- Inspired by physical annealing processes.
- Think of f as an "energy" function.

1. Choose a starting point p.
2. Consider a jump to a neighboring state p'.
3. Accept the jump…
    - Always if $f(p') < f(p)$
    - With probability $P(T)$ if $f(p') >= f(p)$.

- $P(T)$ is larger for larger T.
- T decreases as the algorithm progresses.

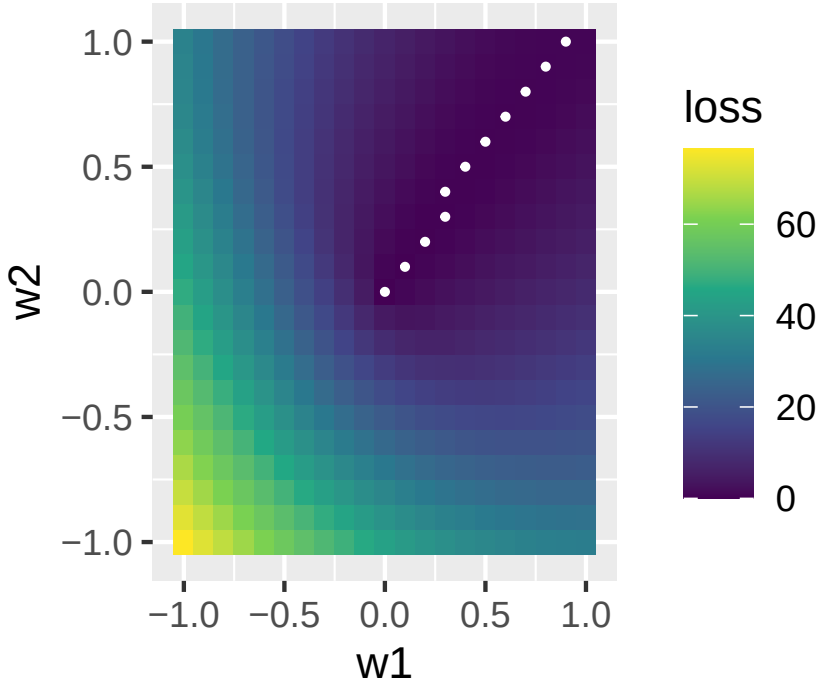


Simulated annealing allows random parameter jumps at high "temperature" and favors safe jumps at lower T.

- Inspired by physical annealing processes.
- Think of f as an "energy" function.

1. Choose a starting point p.
2. Consider a jump to a neighboring state p'.
3. Accept the jump…
    - Always if $f(p') < f(p)$
    - With probability $P(T)$ if $f(p') >= f(p)$.

- $P(T)$ is larger for larger T.
- T decreases as the algorithm progresses.



The Hope:

- "Hot" jumps get you out of local minima.
- Settle in to a minimum as the system cools.

Many optimization algorithms share similar computational considerations.

- Encoding constrains in the function to be minimized before you start.
- Choice of a good starting value
- How big is a “jump”?
- Defining the “edges” of allowable parameter values.
- How do you know when to stop (often called “convergence”)?
    - When each jump changes the result less than a threshold value.
    - After a fixed number of steps.