Gradient Descent

- Updates weights & biases after
- Weights can be seen as strength of connection
- Bias : Alter activation to be active / inactive

Training

1) Initialize random weights & bias (Cost will be bad)
2) Find slope of cost fn. └ cost = (Predict − Actual)²
   - └ if slope +ve : move left } Approach minimum
   - if slope −ve : move right }
   - weights & bias → [cost fn] → cost
   - * slope is for 2D.
   - Higher dimension = direction that decrease most quickly
   - may be local
   - may be global

2.1) Gradient of multi-variable fn gives steepest increase
   - Gradient gives steepest decrease.

Gist:

1) Find Gradient
2) Take controlled steps in dirn of gradient
3) Repeat.

Cost vs Loss

Cost : Average error across dataset

Loss : Error for single node

Gradient Descent

- An optimization algorithm used to minimize cost fn by iteratively adjusting params (weights & Bias) in steepest descent.

Suppose $\vec{W} = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}$

$-\nabla C(\vec{W}) = \begin{bmatrix} 0.03 \\ 0.15 \\ \vdots \\ -0.37 \end{bmatrix}$ ← should increase just a little (Not very meaningful)

← should increase a lot ← more impactful (important)

Caveat of Multi-Layer Perceptron

- Just a bunch of activations, weights & biases.
- No exact meaning, simply Math.
- Just memorizing by altering weights & biases.
  - ↳ Learns structured data quicker.
    - └ most local minima same quality ≈ global minima.