A note about Backpropagation and Random Walk

Backpropagation, Random Walk

Backpropagation:

- **Definition**: Backpropagation is the core algorithm used to train neural networks. It's a supervised learning technique that efficiently calculates gradients of the loss function with respect to the network's weights and biases.

- **Process:** The process works by:
  - Performing a forward pass through the network to generate predictions
  - Calculating the error/loss by comparing predictions to actual targets
  - Propagating this error backward through the network
  - Computing gradients for each parameter
  - Updating parameters to minimize the error

- **Backpropagation Rule**: The backpropagation rule uses the chain rule of calculus to calculate how each weight contributes to the final error. The weight update formula is:

  $\Delta w_{ij} = -\eta \times \partial E/\partial w_{ij}$

  Where:

  - Δw_{ij} is the change in weight between neurons i and j
  - η (eta) is the learning rate
  - $\partial E/\partial w_{ij}$ is the partial derivative of the error with respect to the weight

  For a neuron in the output layer, the gradient is calculated as: $\partial E/\partial w_{ij} = \partial E/\partial y_j \times \partial y_j/\partial net_j \times \partial net_j/\partial w_{ij}$

For hidden layers, the error is backpropagated from the subsequent layer.

- **Parameters Updated During Training**

  During backpropagation, two types of parameters are updated:

  1. **Weights (w)**: The connection strengths between neurons that determine how much influence one neuron has on another
  2. **Biases (b)**: Offset values added to each neuron that allow the model to fit the data better

  These are the learnable parameters of the network that get adjusted through gradient descent to minimize the loss function.

Random Walk Optimization:

Random Walk Optimization is a stochastic optimization technique where:

1. The algorithm starts at an initial point in the parameter space
2. Random steps are taken in different directions
3. If a step improves the objective function, the algorithm moves to that point
4. If not, it may accept the worse solution with some probability (to escape local optima)
5. The process continues for a specified number of iterations or until convergence

The method is simple to implement but often inefficient for high-dimensional spaces. It's primarily useful for exploring rough landscapes where gradient-based methods might fail due to discontinuities or multiple local optima.