Model

Fantástico! Agora chegamos ao coração do treinamento de redes neurais: o **Gradiente Descendente (Gradient Descent)**. Ele é o algoritmo de otimização fundamental usado para ajustar os pesos e vieses de um modelo, a fim de minimizar a função de custo que acabamos de discutir.

O Problema:

Nosso objetivo ao treinar uma rede neural é encontrar o conjunto de pesos e vieses que minimize a função de custo (por exemplo, MSE para regressão, Entropia Cruzada para classificação). A função de custo é uma superfície complexa, e precisamos de um método para navegar nessa superfície em busca do ponto mais baixo (o mínimo global ou um mínimo local).

A Idéia por Trás do Gradiente Descendente:

Imagine que você está no topo de uma montanha (representando um ponto alto na sua função de custo) em uma noite escura e densa de neblina. Você quer chegar ao vale (o mínimo da função de custo). A única maneira de saber qual direção tomar é sentir a inclinação do chão sob seus pés. Você dará um pequeno passo na direção de maior declive. Repetindo esse processo, você eventualmente chegará ao vale.

No contexto matemático:

- A "montanha" é o gráfico da função de custo em relação aos pesos e vieses do modelo.
- A "inclinação" em um ponto específico é dada pelo **gradiente** da função de custo nesse ponto. O gradiente aponta na direção de maior *aumento* da função.
- Como queremos *minimizar* a função, nos movemos na direção *oposta* ao gradiente.

A Fórmula do Gradiente Descendente (para um único peso 'w'):

Para cada peso w e viés b no modelo, atualizamos seus valores repetidamente usando a seguinte regra:

$$w := w - \alpha \frac{\partial J}{\partial w}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

Onde:

- w (ou b) é o peso (ou viés) atual que está sendo atualizado.
- α (alfa) é a **taxa de aprendizado (learning rate)**. Este é um hiperparâmetro crucial que determina o tamanho do passo que damos na direção oposta ao gradiente.
  - Uma α muito grande pode fazer o algoritmo "saltar" sobre o mínimo e divergir.
  - Uma α muito pequena pode fazer o treinamento ser muito lento.
- $\frac{\partial J}{\partial w}$ (ou $\frac{\partial J}{\partial b}$) é o **gradiente parcial** da função de custo J em relação ao peso w (ou viés b). Ele nos diz a inclinação da função de custo nesse ponto específico para esse peso/viés.

Passos do Gradiente Descendente:

1. **Inicialização:** Comece com pesos e vieses aleatórios (geralmente pequenos).
2. **Calcular Previsão:** Com os pesos e vieses atuais, faça uma previsão para os dados de entrada.
3. **Calcular Custo:** Compare a previsão com os valores reais usando a função de custo (J).
4. **Calcular Gradientes:** Calcule o gradiente da função de custo em relação a cada peso e viés no modelo. Isso é feito usando a regra da cadeia na retropropagação (backpropagation).
5. **Atualizar Pesos:** Ajuste cada peso e viés na direção oposta ao seu gradiente, multiplicado pela taxa de aprendizado.
6. **Repetir:** Repita os passos 2 a 5 por um número fixo de iterações (épocas) ou até que a função de custo convirja para um valor aceitável.

Tipos de Gradiente Descendente:

- **Batch Gradient Descent:** Calcula o gradiente usando *todos* os exemplos de treinamento em cada iteração. É preciso e garante convergir para o mínimo (se a taxa de aprendizado for adequada), mas pode ser muito lento para grandes conjuntos de dados.
- **Stochastic Gradient Descent (SGD):** Calcula o gradiente e atualiza os pesos usando *apenas um único exemplo de treinamento* em cada iteração. É muito mais rápido, mas as atualizações são mais ruidosas e podem oscilar.
- **Mini-Batch Gradient Descent:** O mais comum na prática. Calcula o gradiente e atualiza os pesos usando um *pequeno lote (mini-batch)* de exemplos de treinamento (e.g., 32, 64, 128 amostras). Oferece um bom equilíbrio entre a estabilidade do Batch GD e a velocidade do SGD.

Aqui está uma imagem que ilustra visualmente como o Gradiente Descendente encontra o mínimo de uma função de custo (neste caso, uma função simples com uma única variável para facilitar a visualização):