🔧 AdaDelta and RMSProp Optimization Algorithms

🎯 Problem Addressed

Traditional gradient descent may face the problem of **exploding or vanishing learning rates (α)** due to unbounded gradients. AdaDelta and RMSProp both address this by dynamically adjusting the learning rate using a form of gradient normalization.

📌 Key Idea: Exponential Weighted Average of Squared Gradients

To prevent uncontrolled growth of the learning rate:

- Use an exponentially decaying average of past squared gradients:

$$S_{d\omega_t} = \beta S_{d\omega_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial \omega}\right)^2$$

- Similarly for bias:

$$S_{db_t} = \beta S_{db_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial b}\right)^2$$

- Here,
  - β is typically set to 0.9 or 0.95.
  - The smaller term $(1-\beta)$ ensures that the contribution from the current gradient is limited, preventing large spikes.

⚙️ Learning Rate Adjustment (AdaDelta/RMSProp Style)

To compute the effective learning rate:

$$\eta' = \frac{\eta}{\sqrt{S_{d\omega_t}} + \epsilon}$$

- ϵ is a small constant to prevent division by zero.
- This approach **scales down** the learning rate when gradients are large and **scales it up** when gradients are small.

🔄 Weight and Bias Update Rules

- Weight update:

$$\omega_t = \omega_{t-1} - \eta' \cdot \frac{\partial L}{\partial \omega_{t-1}}$$

- Bias update:

$$b_t = b_{t-1} - \eta' \cdot \frac{\partial L}{\partial b_{t-1}}$$

These updates are computed at each mini-batch iteration using the updated exponential moving averages.

💡 Summary of Steps in Mini-Batch Training

1. **Compute gradients**:
   - $\frac{\partial L}{\partial \omega}$,
   - $\frac{\partial L}{\partial b}$
2. **Update squared gradient averages**:
   - $S_{d\omega}$ and S_{db}
3. **Adjust learning rate** η' using RMSProp/AdaDelta formula.
4. **Update parameters** ω_t, b_t

🤔 Are AdaDelta and RMSProp the Same?

Not exactly, but very similar:

- **Common ground**:
  Both use the idea of exponential moving averages of squared gradients to adaptively scale the learning rate per parameter.

- **Differences**:

  - **RMSProp**: Uses a fixed global learning rate scaled by the running average.

  - **AdaDelta**: Eliminates the need to manually set a learning rate by using ratios of accumulated updates to gradients.

Hence, while the mathematical intuition and core ideas are very close, **AdaDelta goes a step further by removing the dependency on the learning rate hyperparameter**.