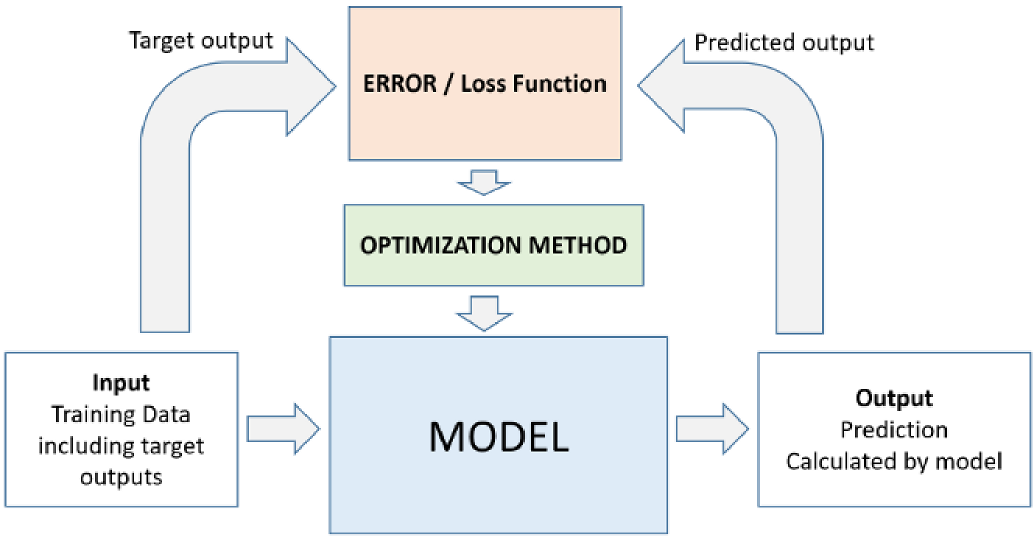
The Journey of Optimizers in Deep Learning

Ahmed Ziada

Optimizers are **algorithms** or methods used to adjust the **weights of neural networks** to **minimize** the **loss function** during training.



In essence, they **guide** the model in the right direction to improve its predictions. Here's an overview, as a story of a hiker:

Imagine a trying to find the **lowest point** in a valley (the global minimum of the loss function). The hiker represents the neural network, and the path to the lowest point is guided by an **optimizer**. Each optimizer is like a different guide, improving upon the methods of the previous one.

1. Gradient Descent: *The Careful Explorer*

The hiker starts their journey with a simple rule:

look at the slope of the ground (the gradient) and take a step downhill. The size of the step is fixed (learning rate).

- **Problem**: If the valley has many ups and downs, the hiker might move too slowly or overshoot the minimum.

➢ Math Time

Gradient Descent (Basic Optimizer)

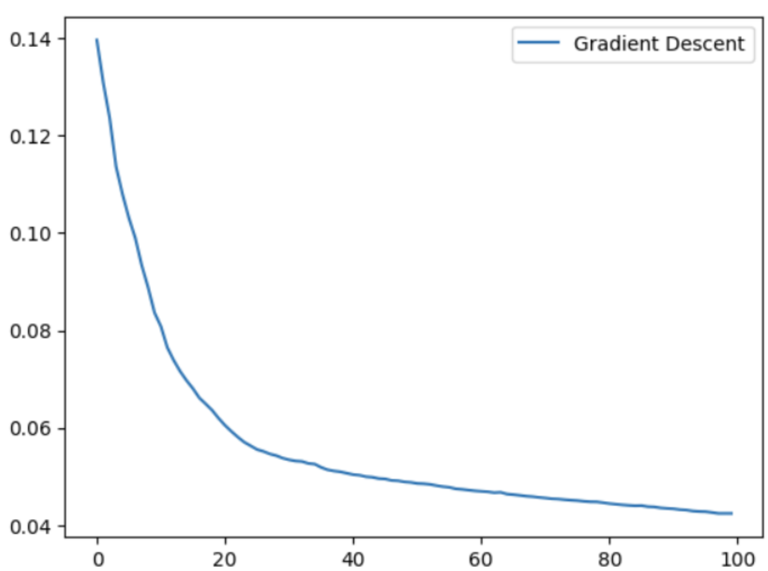
- **How It Works**: Uses the gradient of the loss function w.r.t. the weights to update the weights.
- **Update Rule**:

$$w = w - \eta \cdot \nabla L(w)$$

where:

- w: weights
- η: learning rate (step size)
- $\nabla L(w)$: gradient of the loss

- **Types**:
  - **Batch Gradient Descent**: Uses the entire dataset to compute gradients.
  - **Stochastic Gradient Descent (SGD)**: Uses one sample at a time.
  - **Mini-Batch Gradient Descent**: Uses a small subset of the data (mini-batch).



2. SGD with Momentum: *The Speedy Helper*

Then comes a new guide: **Momentum**. They give the hiker a push in the same direction they've been going, like a rolling ball gaining speed downhill. This helps the hiker move faster and avoid getting stuck in small bumps.

- **Improvement**: Speeds up movement and reduces zig-zagging.

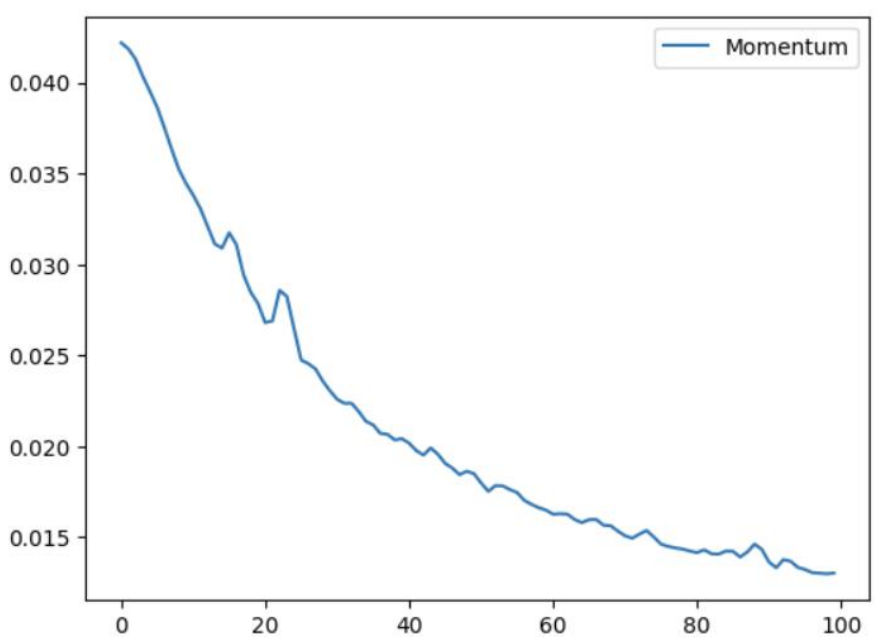
➢ **Math Time**

SGD with Momentum

- **Improvement**: Adds a momentum term to accelerate updates in relevant directions and reduce oscillations.
- **Update Rule**:

$$v_t = \gamma v_{t-1} + \eta \nabla L(w)$$

$$w = w - v_t$$ where γ is the momentum factor.



3. Adagrad: *The Adaptive Learner*

Next, **Adagrad** arrives and says, "Why take the same-sized steps everywhere? Let's adjust the step size for each direction based on past experience." If a path is steep, take smaller steps; if it's flat, take bigger ones.

- **Improvement**: Adjusts learning rates for each parameter dynamically.
- **Problem**: Steps can become too small over time and slow progress.

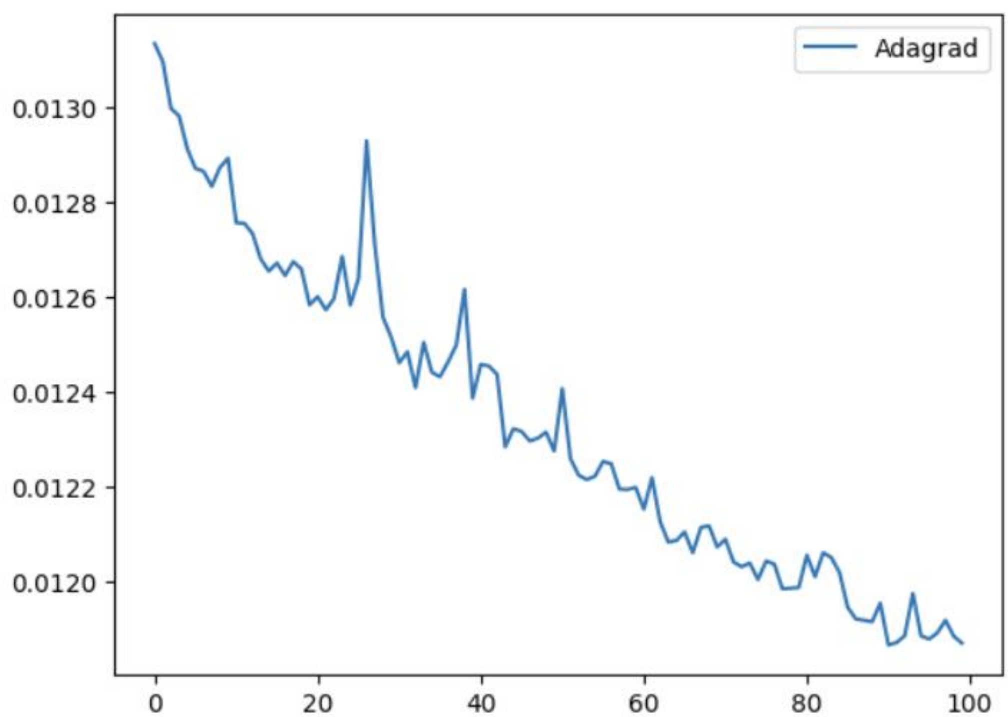
➢ Math Time

Adagrad (Adaptive Gradient Algorithm)

- **Improvement**: Adapts the learning rate for each parameter based on the magnitude of past gradients.
- **Update Rule**:

$$w = w - \frac{\eta}{\sqrt{G_{ii} + \epsilon}} \nabla L(w)$$

where G_{ii} is the sum of squared gradients for each parameter.



4. RMSProp: *The Balancer*

Then comes **RMSProp**, a wise guide who fixes Adagrad's issue. Instead of summing all past information, RMSProp focuses on the recent gradients, balancing step sizes better.

- **Improvement**: Handles steep and flat areas well, without slowing down too much.

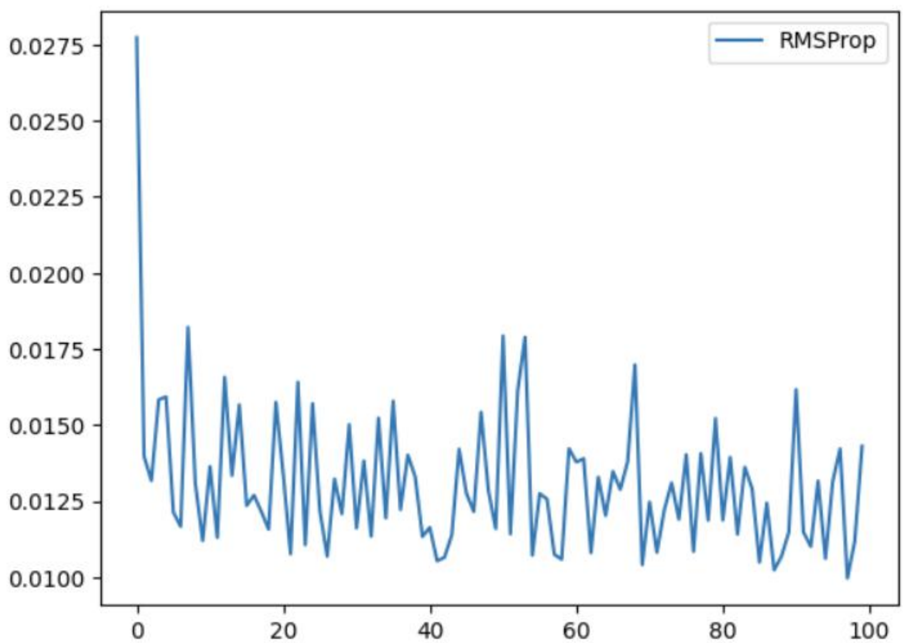
➢ Math Time

RMSProp (Root Mean Square Propagation)

- **Improvement**: Introduces a moving average of squared gradients to normalize updates.
- **Update Rule**:

$$w = w - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} \nabla L(w)$$

where $E[g^2]$ is the exponential moving average of squared gradients.



5. Adam: *The Smart Leader*

Finally, **Adam** (Adaptive Moment Estimation) takes over. Adam is like a genius guide who combines the best ideas from Momentum and RMSProp. Adam keeps track of both:

- The hiker's speed (momentum).
- The terrain's shape (adaptive step sizes).
  With Adam, the hiker reaches the minimum quickly and efficiently!
- **Improvement**: Combines the strengths of Momentum and RMSProp, making it the most popular optimizer.

➢ Math Time

Adam (Adaptive Moment Estimation)

- **Improvement**: Combines the benefits of Momentum and RMSProp by maintaining moving averages of both gradients and squared gradients.
- **Update Rule**:

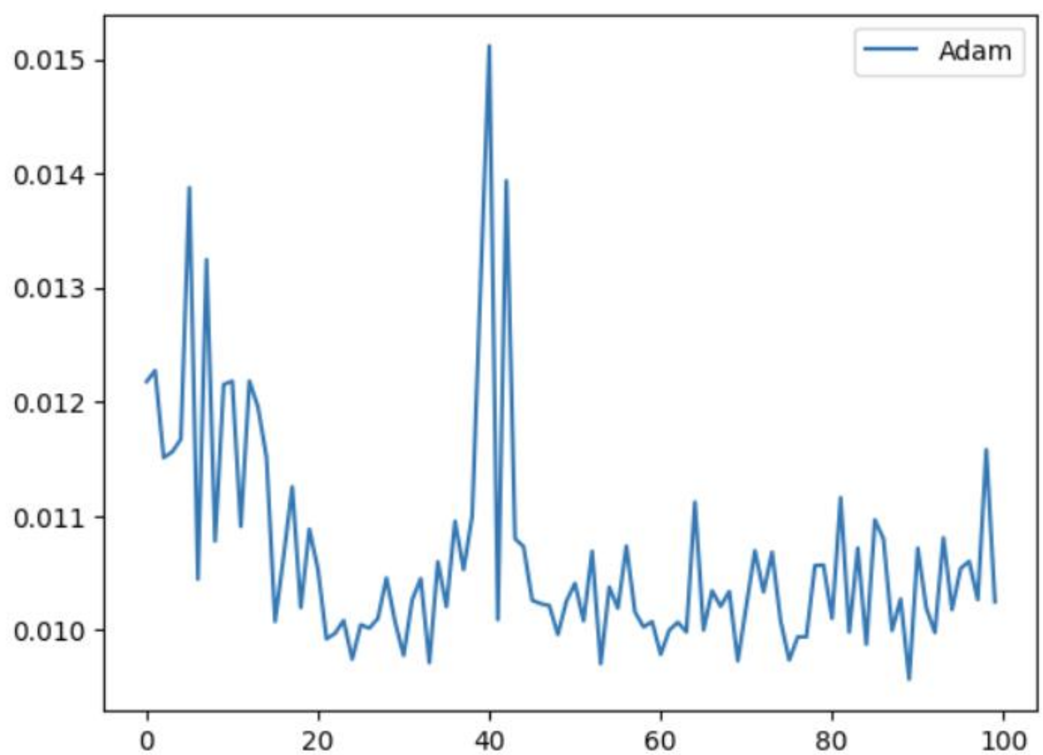
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla L(w)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L(w))^2$$

$$w = w - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

where:

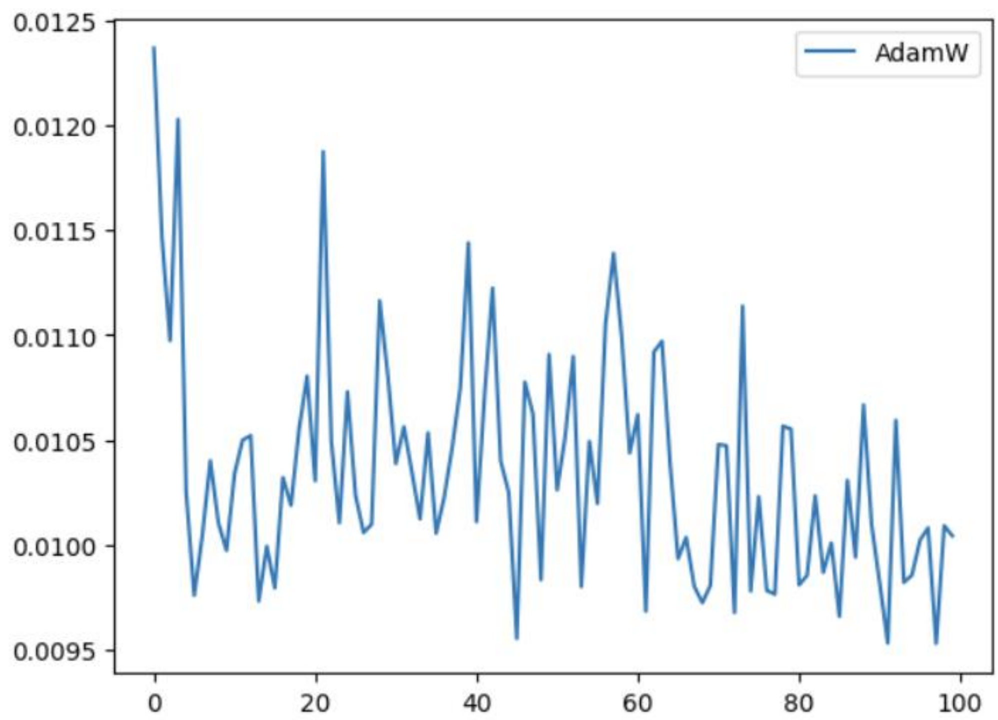
- m_t: first moment (mean of gradients)
- v_t: second moment (variance of gradients)
- β_1, β_2: exponential decay rates.



6. AdamW: *The Disciplined Guide*

Finally, **AdamW** arrives and adds one final tweak: weight decay. It makes sure the hiker doesn't get too lazy (overfitting) and keeps them disciplined while finding the minimum.

- **Improvement**: Adds weight decay regularization directly into Adam, improving generalization.



How to Choose an Optimizer?

1. **Start with Adam**: It works well in most cases and is a great default choice.
2. **Try SGD**: If you care about fine control and want to avoid overfitting.
3. **Use RMSProp**: For recurrent neural networks (RNNs).
4. **Experiment**: Depending on the dataset and model, try other optimizers to compare performance.