1. Batch Gradient Descent (Full Gradient Descent)

Batch Gradient Descent computes the gradient of the loss function with respect to the parameters for the entire training dataset and updates the parameters in the direction of the negative gradient.

Equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

Variables:

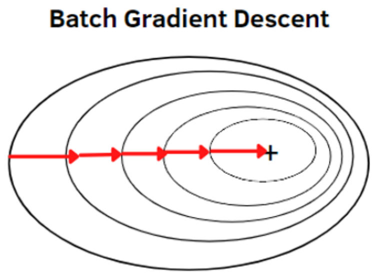
- θ: Parameters to be optimized.
- η: Learning rate.
- $\nabla_\theta J(\theta)$: Gradient of the loss function $J(\theta)$ with respect to θ.

- **Advantages:**
  - Convergence is more stable because it uses the full dataset to calculate gradients.
  - It can be more accurate in finding the global minimum.
- **Disadvantages:**
  - Can be very slow and computationally expensive for large datasets.
  - Requires more memory to compute the gradients.
- **Use Case**:

Suitable for small to medium-sized datasets where computational resources are sufficient. It computes the gradient of the cost function for the entire dataset, which can be slow and resource-intensive for very large datasets.

- Example:

Linear regression on a dataset with a few thousand samples.



2. Stochastic Gradient Descent (SGD)

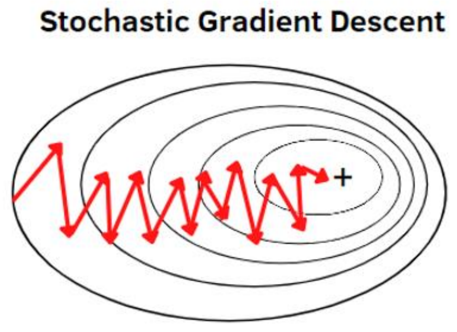
SGD updates the weights by computing the gradient of the loss with respect to the weights for a single data point at a time.

- Equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$
- Variables:
  - $x^{(i)}$, $y^{(i)}$: A single training example and its label.
  - Others as above.
- **Advantages:**
  - Simplicity and ease of implementation.
  - Memory efficiency since it uses only one sample at a time.
- **Disadvantages:**
  - Can be slow to converge.
  - Highly sensitive to the learning rate.
  - Can get stuck in local minima.
- **Use Case**:

Useful for large datasets and online learning where the model is updated incrementally with each training example. It introduces noise into the optimization process, which can help escape local minima.

- **Example**:

Real-time recommendation systems where the model needs to be updated frequently with new user data.



3. Mini-Batch Gradient Descent

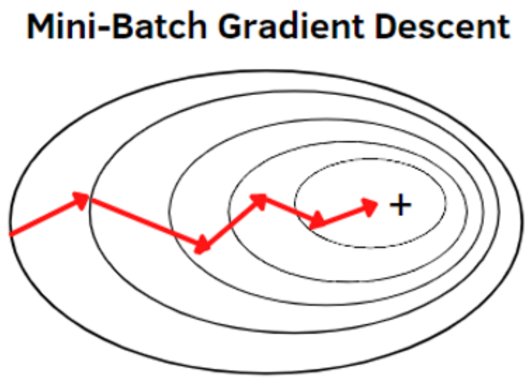
A variant of SGD, where the gradients are computed for small batches of data.

- Equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$
- Variables:
  - $x^{(i:i+n)}$, $y^{(i:i+n)}$: A mini-batch of training examples and their labels.
  - Others as above.
- **Advantages:**
  - Balances the efficiency of SGD and the stability of batch gradient descent.
  - Reduces variance in the weight updates, leading to more stable convergence.
- **Disadvantages:**
  - Requires tuning of the batch size.
  - Still sensitive to learning rate.
- **Use Case**:

Balances between batch gradient descent and SGD by updating the model using small batches of data. This improves computational efficiency and can provide more stable convergence.

- **Example**:

Training deep neural networks on large datasets where a batch size of 32, 64, or 128 is common.



4. Momentum

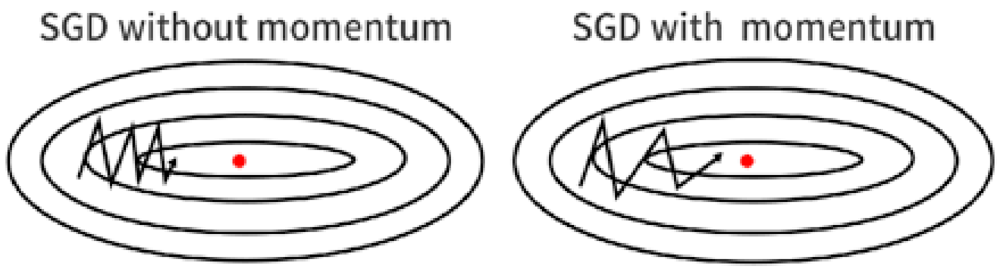
Momentum accelerates SGD by adding a fraction of the previous update to the current update.

- **Equation**:
  - $v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$
  - $\theta = \theta - v_t$
- **Variables**:
  - v_t: Velocity at time t.
  - γ: Momentum factor (usually between 0.0 and 1.0).
- **Advantages:**
  - Can help accelerate convergence and reduce oscillations.
  - Helps to escape local minima.
- **Disadvantages:**
  - Requires tuning of the momentum parameter.
  - Can overshoot the minima if not properly tuned.
- **Use Case**:

Helps accelerate SGD in relevant directions and dampens oscillations. Useful in cases where the optimization landscape has high curvature, small but consistent gradients, or noisy gradients.

- **Example**:

Training convolutional neural networks for image classification tasks.



5. Nesterov Accelerated Gradient (NAG)

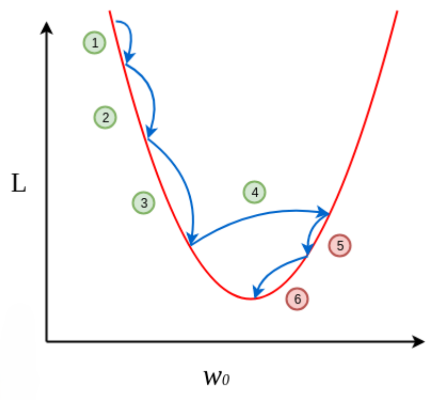
An improvement over momentum, NAG looks ahead to where the gradient is going and adjusts the update accordingly.

- **Equation**:
  - $v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
  - $\theta = \theta - v_t$
- **Variables**: Same as Momentum.
- **Advantages:**
  - Provides a more accurate update direction.
  - Can lead to faster convergence than standard momentum.
- **Disadvantages:**
  - More complex to implement.
  - Requires careful tuning of parameters.
- **Use Case**:

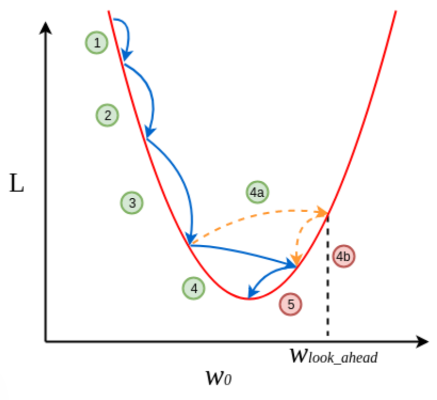
An improvement over momentum that looks ahead to the future position of the parameters. Effective in scenarios where momentum struggles with too much overshooting.

- **Example**:

Training recurrent neural networks for time-series prediction.



Momentum-Based Gradient Descent



Nesterov Accelerated Gradient Descent

6. Adagrad (Adaptive Gradient Algorithm)

Adagrad adapts the learning rate for each parameter based on the historical gradients.

- Equation:
  - $r_t = r_{t-1} + \nabla_\theta J(\theta)^2$
  - $\theta = \theta - \frac{\eta}{\sqrt{r_t + \epsilon}} \nabla_\theta J(\theta)$
- Variables:
  - r_t: Accumulated gradient.
  - ϵ: Smoothing term to prevent division by zero.
- **Advantages:**
  - No need to manually tune the learning rate.
  - Effective for sparse data.
- **Disadvantages:**
  - Learning rate can become too small, leading to slow convergence.
- **Use Case**:

Adapts the learning rate for each parameter based on the frequency of updates, making it suitable for dealing with sparse data and feature-specific learning rates.

- **Example**:

Natural language processing tasks like text classification, where certain features (words) are sparse.

7. RMSprop (Root Mean Square Propagation)

RMSprop addresses the diminishing learning rate problem in Adagrad by using a moving average of squared gradients.

- Equation:
  - $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)\nabla_\theta J(\theta)^2$
  - $\theta = \theta - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla_\theta J(\theta)$
- Variables:
  - ρ: Decay rate (usually between 0.0 and 1.0).
- **Advantages:**
  - Effective for non-stationary objectives.
  - Maintains a good balance between convergence speed and stability.
- **Disadvantages:**
  - Requires tuning of decay rate.
  - Can be sensitive to initial learning rate.
- **Use Case**:

Maintains a moving average of squared gradients to normalize the gradient, suitable for non-stationary problems and mini-batch learning. Addresses the diminishing learning rates problem in Adagrad.

- **Example**:

Training deep reinforcement learning models.

8. Adam (Adaptive Moment Estimation)

Adam combines the ideas of momentum and RMSprop, maintaining a moving average of both the gradients and their squares.

- Equation:
  - $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta)$
  - $v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla_\theta J(\theta)^2$
  - $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
  - $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
  - $\theta = \theta - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$
- Variables:
  - m_t: First moment estimate.
  - v_t: Second moment estimate.
  - β_1, β_2: Exponential decay rates for the moment estimates.
- **Advantages:**
  - Works well in practice for a wide range of problems.
  - Requires less tuning of the learning rate.
- **Disadvantages:**
  - Computationally expensive.
  - Can sometimes lead to poor generalization.
- **Use Case**:

Combines the advantages of both RMSprop and momentum by maintaining moving averages of both the gradients and their second moments. Widely used due to its robustness and efficiency.

- **Example**:

General-purpose optimizer for training deep neural networks in various domains such as computer vision, NLP, and more.

9. AdaDelta

An extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate.

- **Equation**:
  - $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)\nabla_\theta J(\theta)^2$
  - $\Delta\theta = -\frac{\sqrt{E[\Delta\theta^2]_{t-1}+\epsilon}}{\sqrt{E[g^2]_t+\epsilon}}\nabla_\theta J(\theta)$
  - $\theta = \theta + \Delta\theta$
- **Variables**: Similar to RMSprop with different update rule.
- **Advantages:**
  - Reduces the need to set an initial learning rate.
  - Adapts the learning rate based on a moving window of gradient updates.
- **Disadvantages:**
  - More complex than Adagrad.
  - May not perform as well as Adam on some problems.
- **Use Case**:

An extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Suitable for models requiring a more adaptive learning rate adjustment.

- **Example**:

Training models with adaptive learning rate requirements without needing to set a default learning rate.

10. Adamax

A variant of Adam based on the infinity norm.

- Equation:
  - $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta)$
  - $u_t = \max(\beta_2 u_{t-1}, |\nabla_\theta J(\theta)|)$
  - $\theta = \theta - \frac{\eta}{u_t} m_t$
- Variables:
  - u_t: Maximum of the past gradients' magnitudes.
- **Advantages:**
  - Sometimes more stable than Adam.
  - Effective for large models.
- **Disadvantages:**
  - Less commonly used, less empirical evidence on its performance.
- **Use Case**:

A variant of Adam based on the infinity norm. Works well when dealing with sparse gradients or when the model has large parameter magnitudes.

- **Example**:

Deep learning applications with very sparse updates or extremely large parameter values.

11. Nadam

A combination of Adam and NAG, which incorporates Nesterov momentum into Adam.

- **Equation**:
  - $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta)$
  - $v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla_\theta J(\theta)^2$
  - $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
  - $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
  - $\theta = \theta - \eta\frac{\hat{m}_t+(1-\beta_1)\nabla_\theta J(\theta)}{\sqrt{\hat{v}_t}+\epsilon}$
- **Variables**: Similar to Adam with an added correction term.
- **Advantages:**
  - Can provide better convergence properties than Adam.
  - Effective for many types of neural networks.
- **Disadvantages:**
  - More complex to implement.
  - Requires careful parameter tuning.
- **Use Case**:

Combines NAG with Adam, adding momentum to Adam's adaptive learning rate. Useful for achieving faster convergence in deep learning models.

- **Example**:

Complex neural network architectures where both acceleration and adaptive learning rates are beneficial, such as training GANs.

Choosing an Optimizer

The choice of optimizer depends on the specific problem, the architecture of the neural network, and the characteristics of the data. Here are some guidelines:

- **SGD** is often used when training very large datasets with a simple model.
- **Momentum** and **NAG** are good choices when SGD is too slow.
- **Adagrad** and **RMSprop** are useful for dealing with sparse data or problems with noisy gradients.
- **Adam** and its variants (**Adamax**, **Nadam**) are popular choices for most deep learning applications due to their adaptive nature and good performance across various tasks.
- **AdaDelta** can be used when dealing with diminishing learning rates in Adagrad.