Fuzzy Logic & Neural Networks (CS-514)

Dr. Sudeep Sharma

IIIT Surat

sudeep.sharma@iiitsurat.ac.in

Parameters Optimization

Weights and bias Optimization in Neural Networks

- Optimizing weights and biases in neural networks is crucial for training models effectively and achieving high performance.
- There are several optimization methods used to adjust the weights and biases during training.

- Gradient Descent (GD)
- Gradient Descent with Momentum (GDM)
- Adaptive Gradient Algorithm (AdaGrad)
- Root Mean Square Propagation (RMSProp)
- Adaptive Moment Estimation (Adam)

Parameters Optimization

Weights and bias Optimization in Neural Networks

- The choice of optimization method depends on several factors, including the dataset size, model architecture, and specific problem.
- Experimentation is key to finding the best optimizer for a particular task.

- Gradient Descent (GD)
- Gradient Descent with Momentum (GDM)
- Adaptive Gradient Algorithm (AdaGrad)
- Root Mean Square Propagation (RMSProp)
- Adaptive Moment Estimation (Adam)

Parameters Optimization

Gradient Descent (GD)

- Gradient Descent (GD) is an optimization algorithm used to minimize the loss function of a model.
- It iteratively updates the model's parameters (weights and biases) to find the values that minimize the loss function.
- Let's define a loss function $L(\theta)$ where θ represents the parameters of the model. The goal is to minimize this loss function.
- The gradient of the loss function with respect to θ is a vector of partial derivatives

Parameters Optimization

Gradient Descent (GD)

- Update Rule: The parameters are updated iteratively using the formula:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t)$$

where:

- θ_t are the parameters at iteration(epoch) t,
- α is the learning rate, a hyperparameter that controls the size of the steps taken toward the minimum,
- $\nabla L(\theta_t)$ is the gradient of the loss function with respect to θ_t .

Parameters Optimization

Gradient Descent with Momentum (GDM)

- Adding the Momentum term in GD can lead to faster convergence and helps to escape local minima.
- The update rule for Gradient Descent with Momentum involves two main components:
- Velocity (v): This is the exponentially weighted moving average of past gradients.
- Parameter Update (θ): The model parameters are updated using the velocity vector rather than the raw gradient.

Parameters Optimization

Gradient Descent with Momentum (GDM)

- The momentum update rules are:

$$v_{t+1} = \gamma v_t + \alpha \nabla L(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

$$v_t = \alpha \sum_{i=1}^{t} \gamma^{t-i} \nabla L(\theta_{i-1})$$

where:

- v_t is the velocity vector at iteration t,
- γ (gamma) is the momentum coefficient, typically a value between 0 and 1 (e.g., 0.9),
- α (alpha) is the learning rate,
- $\nabla L(\theta_t)$ is the gradient of the loss function with respect to the parameters θ_t.

Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

- It is an optimization method designed to adapt the learning rate of each parameter individually during training.
- AdaGrad modifies the standard gradient descent update rule by scaling the learning rate for each parameter based on the historical sum of squared gradients.
- This scaling helps parameters with large gradients to have their updates reduced and parameters with small gradients to have their updates increased.

Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

- The AdaGrad update rule for each parameter θ_i at iteration t is given by:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{G_{i,t} + \epsilon}} \nabla_{\theta_i} L(\theta_t)$$

where:

- $G_{i,t}$ is the cumulative sum of the squares of the gradients for parameter θ_i up to time step t:

$$G_{i,t} = \sum_{j=0}^{t} (\nabla_{\theta_i} L(\theta_j))^2$$

- ϵ is a small constant (e.g., 10^{-8}) added to prevent division by zero.
- $\nabla_{\theta i} L(\theta_t)$ is the gradient of the loss function with respect to parameter θ_i at iteration t.

Parameters Optimization

Root Mean Square Propagation (RMSProp)

- RMSProp is an adaptive learning rate optimization algorithm designed to improve upon AdaGrad by addressing its diminishing learning rate problem.
- RMSProp scales the learning rate by dividing it by an exponentially decaying average of squared gradients, rather than the cumulative sum of squared gradients like AdaGrad.
- This method is particularly well-suited for deep learning models, where the loss landscape is often non-convex and noisy.

Parameters Optimization

Root Mean Square Propagation (RMSProp)

- The update rule for a parameter θ_i at iteration t is given by:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{E[g_{i,t}^2] + \epsilon}} \nabla_{\theta_i} L(\theta_t)$$

where:

- $g_{i,t} = \nabla\theta_i L(\theta_t)$ is the gradient of the loss function with respect to parameter θ_i at iteration t.
- $E[g_{i,t}^2]$ is the exponentially decaying average of past squared gradients for parameter θ_i .
- The $E[g_{i,t}^2]$ is updated as:

$$E[g_{i,t}^2] = \gamma E[g_{i,t-1}^2] + (1-\gamma) g_{i,t}^2$$

- γ is the decay rate (a hyperparameter, typically set to 0.9).

Parameters Optimization

Adaptive Moment Estimation (Adam)

- **Adam** is an optimization algorithm that combines the benefits of AdaGrad and RMSProp methods.
- Adam maintains an adaptive learning rate for each parameter by computing first (mean) and second moments (variance) of the gradients.
- The first moment is an exponentially decaying average of past gradients, while the second moment is an exponentially decaying average of past squared gradients.

Parameters Optimization

Adaptive Moment Estimation (Adam)

- The update rule for a parameter θ_i at iteration t is given by:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{\hat{v}_{i,t} + \epsilon}} \hat{m}_{i,t}$$

where:

- $\hat{v}_{i,t}$ is the bias-corrected first moment estimate (mean of the gradients).
- $\hat{m}_{i,t}$ is the bias-corrected second moment estimate (variance of the gradients).

Parameters Optimization

Adaptive Moment Estimation (Adam)

- First Moment (Mean) Estimate:

$$m_{i,t} = \beta_1 m_{i,t-1} + (1 - \beta_1) g_{i,t}$$

where:

- $m_{i,t}$ is the moving average of the gradient.

- β_1 is the exponential decay rate for the first moment (typically β_1=0.9).

- $g_{i,t}$=$\nabla\theta_i L(\theta_t)$ is the gradient of the loss function with respect to parameter θ_i at iteration t.

Parameters Optimization

Adaptive Moment Estimation (Adam)

- Second Moment (Variance) Estimate:

$$v_{i,t} = \beta_2 v_{i,t-1} + (1 - \beta_2) g_{i,t}^2$$

where:

- $v_{i,t}$ is the moving average of the squared gradient.
- β_2 is the exponential decay rate for the second moment (typically β_2=0.999).

Parameters Optimization

Adaptive Moment Estimation (Adam)

- Since $m_{i,\mathrm{t}}$ and $v_{i,\mathrm{t}}$ are initialized as zeros, they are biased toward zero, therefore, bias-corrected estimates:

$$\hat{m}_{i,t} = \frac{m_{i,t}}{1-\beta_1}, \quad \hat{v}_{i,t} = \frac{v_{i,t}}{1-\beta_2}$$

Coding Parameters Optimization

Gradient Descent (GD)

- Update Rule: The parameters are updated iteratively using the formula:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t)$$

where:

- θ_t are the parameters at iteration(epoch) t,
- α is the learning rate, a hyperparameter that controls the size of the steps taken toward the minimum,
- $\nabla L(\theta_t)$ is the gradient of the loss function with respect to θ_t .

Coding Parameters Optimization

Gradient Descent (GD)

- Example: Let the function to be minimized is:

$$L(\theta) = \theta^2$$

Iteration 1:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t) \rightarrow \theta_1 = \theta_0 - \alpha \nabla L(\theta_0)$$

Let

- θ_0 = 10
- α = 0.1
- $\nabla L(\theta_0)$ = 2*θ_0 = 20.

$$\theta_1 = \theta_0 - \alpha \nabla L(\theta_0) \rightarrow \theta_1 = 10 - 0.1 * 20 = 8$$

Coding Parameters Optimization

Gradient Descent (GD)

- Example: Let the function to be minimized is:

$$L(\theta) = \theta^2$$

Iteration 2:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t) \rightarrow \theta_2 = \theta_1 - \alpha \nabla L(\theta_1)$$

Now

- $\theta_1 = 8$
- $\alpha = 0.1$
- $\nabla L(\theta_1) = 2*\theta_1 = 16.$

$$\theta_2 = \theta_1 - \alpha \nabla L(\theta_1) \rightarrow \theta_2 = 8 - 0.1*16 = 6.4$$

Coding Parameters Optimization

Gradient Descent (GD)

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function f(x) = x^2
def f(x):
    return x**2

# Derivative of f(x), which is 2*x
def gradient(x):
    return 2 * x
```

Coding Parameters Optimization

Gradient Descent (GD)

```python
# Gradient Descent function
def gradient_descent(starting_x, learning_rate, num_iterations):
    x = starting_x  # Initial value of x
    x_values = []   # To store the values of x during the iterations
    f_values = []   # To store the values of f(x)

    for i in range(num_iterations):
        x_values.append(x)
        f_values.append(f(x))

        grad = gradient(x)
        x = x - learning_rate * grad  # Update x using gradient

        # Print the iteration details
        print(f"Iteration {i+1}: x = {x}, f(x) = {f(x)}")

    return x_values, f_values
```

Coding Parameters Optimization

Gradient Descent (GD)

```
# Parameters for gradient descent
starting_x = 10    # Start far from the minimum
learning_rate = 0.1
num_iterations = 25

# Perform gradient descent
x_values, f_values = gradient_descent(starting_x, learning_rate, num_iterations)

x_var = np.linspace(-10,10,num_iterations)
y_var = f(x_var)
plt.plot(x_var,y_var)

# Plot the results
plt.plot(x_values,f_values, '-*', label='Optimized Parameter')
plt.plot(x_var,y_var, label = 'Loss Function')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent Optimization of f(x) = x^2')
plt.legend()
plt.show()
```

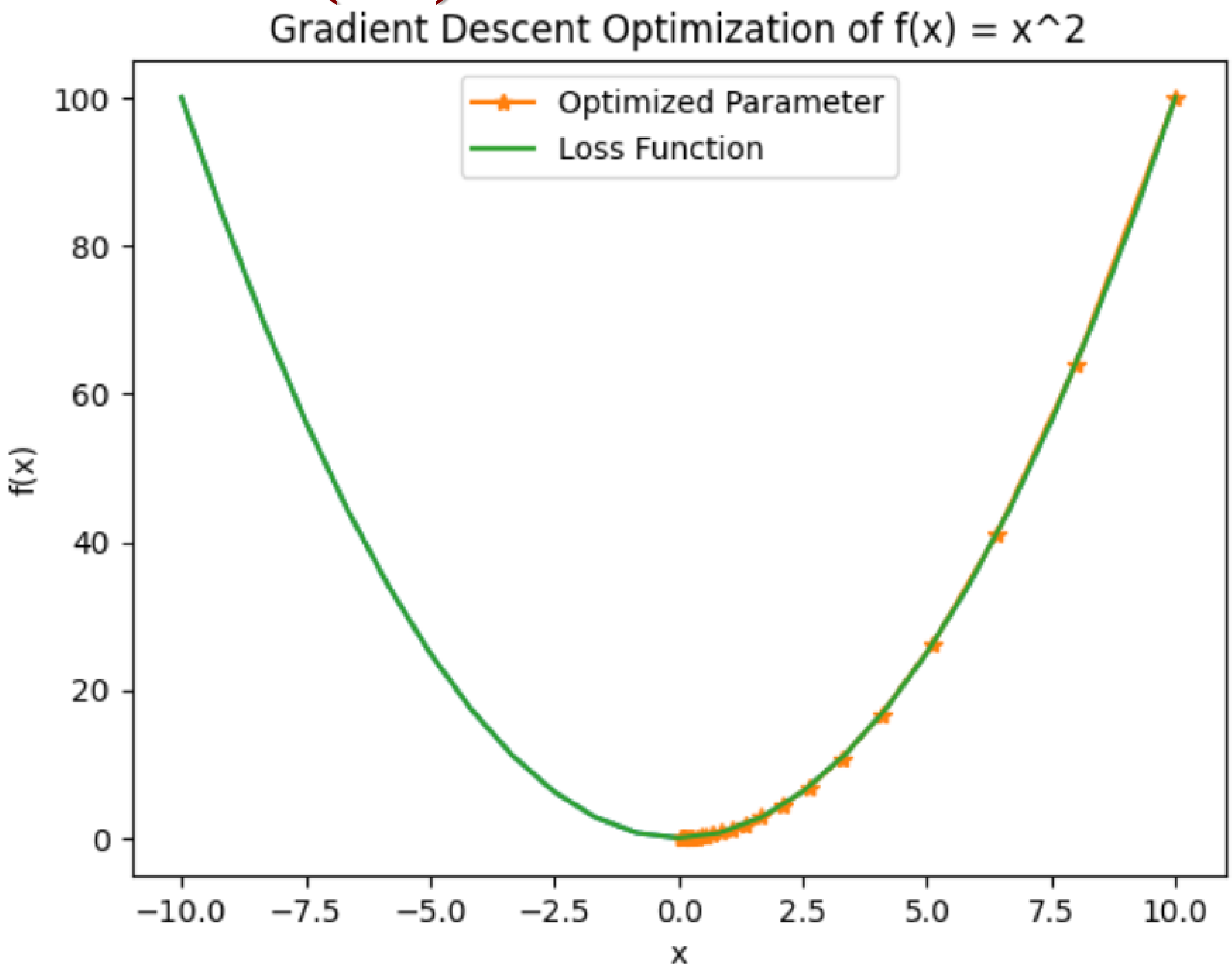
Coding Parameters Optimization

Gradient Descent (GD)

```
Iteration 1: x = 8.0, f(x) = 64.0
Iteration 2: x = 6.4, f(x) = 40.96000000000001
Iteration 3: x = 5.12, f(x) = 26.2144
Iteration 4: x = 4.096, f(x) = 16.777216
Iteration 5: x = 3.2768, f(x) = 10.73741824
Iteration 6: x = 2.62144, f(x) = 6.871947673600001
Iteration 7: x = 2.0971520000000003, f(x) = 4.398046511104002
Iteration 8: x = 1.6777216000000004, f(x) = 2.814749767106561З
Iteration 9: x = 1.3421772800000003, f(x) = 1.801439850948199
Iteration 10: x = 1.0737418240000003, f(x) = 1.152921504606847‌5
Iteration 11: x = 0.8589934592000003, f(x) = 0.7378697629483825
Iteration 12: x = 0.6871947673600002, f(x) = 0.4722366482869647‌7
Iteration 13: x = 0.5497558138880001, f(x) = 0.3022314549036574
Iteration 14: x = 0.43980465111040007, f(x) = 0.19342813113834073
Iteration 15: x = 0.35184372088832006, f(x) = 0.12379400392853807
Iteration 16: x = 0.28147497671065601, f(x) = 0.0792281625142643‌4
Iteration 17: x = 0.22517998136852482, f(x) = 0.050706024009129186
Iteration 18: x = 0.18014398509481985, f(x) = 0.03245185536584268
Iteration 19: x = 0.14411518807585588, f(x) = 0.020769187434139313
Iteration 20: x = 0.11529215046068471, f(x) = 0.013292279957849162
Iteration 21: x = 0.09223372036854777, f(x) = 0.008507059173023463
Iteration 22: x = 0.07378697629483821, f(x) = 0.005444517870735016‌5
Iteration 23: x = 0.05902958103587057, f(x) = 0.003484491437270‌41
Iteration 24: x = 0.04722366482869646, f(x) = 0.002230074519853063
Iteration 25: x = 0.03777893186295716‌6, f(x) = 0.0014272476927059603
```

Coding Parameters Optimization

Gradient Descent (GD)



Coding Parameters Optimization

Gradient Descent with Momentum (GDM)

- The momentum update rules are:

$$v_{t+1} = \gamma v_t + \alpha \nabla L(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

$$v_t = \alpha \sum_{i=1}^{t} \gamma^{t-i} \nabla L(\theta_{i-1})$$

where:

- v_t is the velocity vector at iteration t,
- γ (gamma) is the momentum coefficient, typically a value between 0 and 1 (e.g., 0.9),
- α (alpha) is the learning rate,
- $\nabla L(\theta_t)$ is the gradient of the loss function with respect to the parameters θ_t.

Coding Parameters Optimization

Gradient Descent with Momentum (GDM)

- Example: For the same function

Iteration 1

$$v_{t+1} = \gamma v_t + \alpha \nabla L(\theta_t) \rightarrow v_1 = \gamma v_0 + \alpha \nabla L(\theta_0)$$

$$\theta_{t+1} = \theta_t - v_{t+1} \rightarrow \theta_1 = \theta_0 - v_1$$

Let:

- v_0 is 0, θ_0 is 10,
- γ (gamma) is 0.9,
- α (alpha) is 0.1,
- $\nabla L(\theta_0)$ is $2*\theta_0$.

$$v_1 = \gamma v_0 + \alpha \nabla L(\theta_0) \rightarrow v_1 = 0.9*0 + 0.1*20 = 2$$

$$\theta_1 = \theta_0 - v_1 \rightarrow \theta_1 = 10 - 2 = 8$$

Coding Parameters Optimization

Gradient Descent with Momentum (GDM)

- Example: For the same function

Iteration 2

$$v_{t+1} = \gamma v_t + \alpha \nabla L(\theta_t) \rightarrow v_2 = \gamma v_1 + \alpha \nabla L(\theta_1)$$

$$\theta_{t+1} = \theta_t - v_{t+1} \rightarrow \theta_2 = \theta_1 - v_2$$

Let:

- v_1 is 2, θ_0 is 8,
- γ (gamma) is 0.9,
- α (alpha) is 0.1,
- $\nabla L(\theta_1)$ is $2*\theta_1$.

$$v_2 = \gamma v_1 + \alpha \nabla L(\theta_1) \rightarrow v_2 = 0.9*2 + 0.1*16 = 3.4$$

$$\theta_2 = \theta_1 - v_2 \rightarrow \theta_2 = 8 - 3.4 = 4.6$$

Coding Parameters Optimization

Gradient Descent with Momentum (GDM)

```
import numpy as np
import matplotlib.pyplot as plt

# Define the function f(x) = x^2
def f(x):
    return x**2

# Derivative of f(x), which is 2*x
def gradient(x):
    return 2 * x
```

Coding Parameters Optimization

Gradient Descent with Momentum (GDM)

```python
# Gradient Descent with Momentum function
def gradient_descent_with_momentum(starting_x, learning_rate, momentum_factor, num_iterations):
    x = starting_x  # Initial value of x
    velocity = 0    # Initialize velocity to 0
    x_values = []   # To store the values of x during the iterations
    f_values = []   # To store the values of f(x)

    for i in range(num_iterations):
        x_values.append(x)
        f_values.append(f(x))

        grad = gradient(x)
        velocity = momentum_factor * velocity - learning_rate * grad  # Update velocity
        x = x + velocity  # Update x using the velocity

        # Print the iteration details
        print(f"Iteration {i+1}: x = {x}, f(x) = {f(x)}, velocity = {velocity}")

    return x_values, f_values
```

Coding Parameters Optimization

Gradient Descent with Momentum (GDM)

```
# Parameters for gradient descent with momentum
starting_x = 10    # Start far from the minimum
learning_rate = 0.1
momentum_factor = 0.9  # Momentum factor (between 0 and 1)
num_iterations = 25

# Perform gradient descent with momentum
x_values_momentum, f_values_momentum = gradient_descent_with_momentum \
                    (starting_x, learning_rate, momentum_factor, num_iterations)



x_var = np.linspace(-10,10,num_iterations)
y_var = f(x_var)
plt.plot(x_var,y_var)

# Plot the results
plt.plot(x_values_momentum,f_values_momentum, '-*', label='Optimized Parameter')
plt.plot(x_var,y_var, label = 'Loss Function')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent with Momentum for f(x) = x^2')
plt.legend()
plt.show()
```

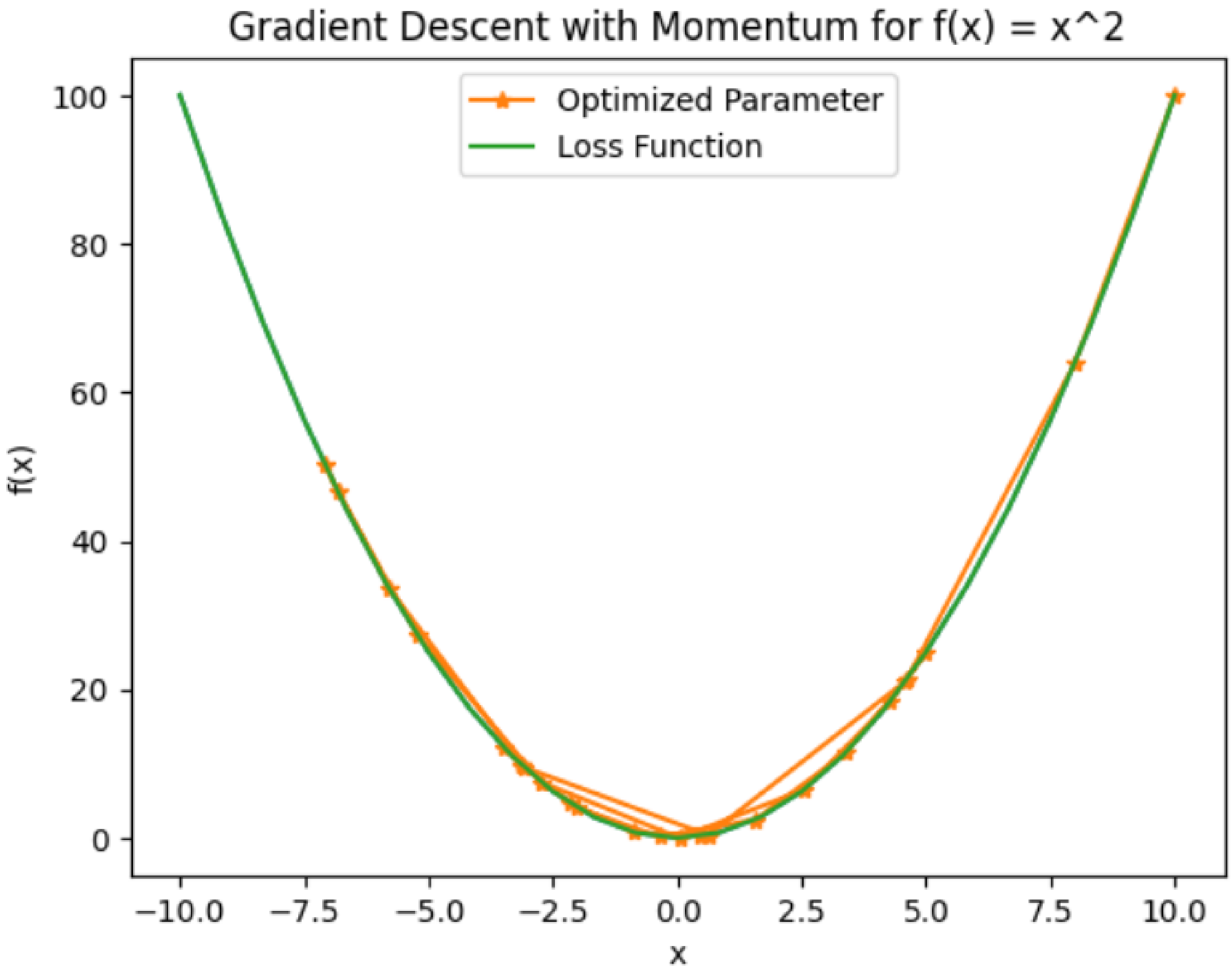
Coding Parameters Optimization

Gradient Descent with Momentum (GDM)

```
Iteration 1: x = 8.0, f(x) = 64.0, velocity = -2.0
Iteration 2: x = 4.6, f(x) = 21.159999999999997, velocity = -3.4000000000000004
Iteration 3: x = 0.6199999999999992, f(x) = 0.3843999999999999, velocity = -3.9800000000000004
Iteration 4: x = -3.0860000000000007, f(x) = 9.523396000000005, velocity = -3.706
Iteration 5: x = -5.8042, f(x) = 33.68873764, velocity = -2.7181999999999995
Iteration 6: x = -7.089739999999999, f(x) = 50.264413267599984, velocity = -1.2855399999999995
Iteration 7: x = -6.828777999999999, f(x) = 46.63220897328399, velocity = 0.2609620000000036
Iteration 8: x = -5.228156599999998, f(x) = 27.333621434123543, velocity = 1.6006214000000003
Iteration 9: x = -2.7419660199999982, f(x) = 7.518377654834631, velocity = 2.48619058
Iteration 10: x = 0.0439987060000133, f(x) = 0.0019358861296745532, velocity = 2.785964725999996
Iteration 11: x = 2.542567218200008, f(x) = 6.46464805906529, velocity = 2.498568512199994
Iteration 12: x = 4.28276543554, f(x) = 18.342079775856124, velocity = 1.740198217339997
Iteration 13: x = 4.992390744037995, f(x) = 24.92396534115629, velocity = 0.709625308497998
Iteration 14: x = 4.632575372878599, f(x) = 21.460754585401293, velocity = -0.359815371159402
Iteration 15: x = 3.382226464259419, f(x) = 11.439455855536771, velocity = -1.2503489086191801
Iteration 16: x = 1.580467153650273, f(x) = 2.4978764237673956, velocity = -1.801759310609146
Iteration 17: x = -0.3572096566280132, f(x) = 0.12759873878830308, velocity = -1.9376768102782862
Iteration 18: x = -2.029676854552868, f(x) = 4.119588133907623, velocity = -1.6724671979248549
Iteration 19: x = -3.128961961774664, f(x) = 9.790402958232752, velocity = -1.099285107221796
Iteration 20: x = -3.4925261659193474, f(x) = 12.197739019631296, velocity = -0.3635642041446836
Iteration 21: x = -3.1212287164656933, f(x) = 9.74206870049008, velocity = 0.3712974494536542
Iteration 22: x = -2.1628152686642657, f(x) = 4.67776988636728, velocity = 0.9584134478014276
Iteration 23: x = -0.8676801119101276, f(x) = 0.7528687766043716, velocity = 1.295135156754138
Iteration 24: x = 0.4714775515506222, f(x) = 0.22229108161616962, velocity = 1.3391576634607498
Iteration 25: x = 1.5824239383551726, f(x) = 2.504065520679495, velocity = 1.1109463868045504
```

Coding Parameters Optimization

Gradient Descent with Momentum (GDM)



Coding Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

- The AdaGrad update rule for each parameter θ_i at iteration t is given by:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{G_{i,t} + \epsilon}} \nabla_{\theta_i} L(\theta_t)$$

where:

- $G_{i,t}$ is the cumulative sum of the squares of the gradients for parameter θ_i up to time step t:

$$G_{i,t} = \sum_{j=0}^{t} (\nabla_{\theta_i} L(\theta_j))^2$$

- ϵ is a small constant (e.g., 10^{-8}) added to prevent division by zero.
- $\nabla_{\theta i} L(\theta_t)$ is the gradient of the loss function with respect to parameter θ_i at iteration t.

Coding Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

- For the same example: Iteration 1:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{G_{i,t}} + \epsilon} \nabla_{\theta_i} L(\theta_t) \rightarrow \theta_{i,1} = \theta_{i,0} - \frac{\alpha}{\sqrt{G_{i,0}} + \epsilon} \nabla_{\theta_i} L(\theta_0)$$

- $G_{i,0}$ is:

$$G_{i,t} = \sum_{j=0}^{t} (\nabla_{\theta_i} L(\theta_j))^2 \rightarrow G_{i,0} = 400$$

- ϵ is 10^{-8} , $\nabla_{\theta i} L(\theta_0)$ is 20, $\alpha = 1.0$.

$$\theta_{i,1} = \theta_{i,0} - \frac{\alpha}{\sqrt{G_{i,0}} + \epsilon} \nabla_{\theta_i} L(\theta_0)$$

$$\theta_{i,1} = 10 - \frac{1}{\sqrt{400} + 10^{-8}} * 20 = 9$$

Coding Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

- For the same example: Iteration 2:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{G_{i,t} + \epsilon}} \nabla_{\theta_i} L(\theta_t) \rightarrow \theta_{i,2} = \theta_{i,1} - \frac{\alpha}{\sqrt{G_{i,1} + \epsilon}} \nabla_{\theta_i} L(\theta_1)$$

- $G_{i,1}$ is:

$$G_{i,1} = \sum_{j=0}^{1} (\nabla_{\theta_i} L(\theta_j))^2 \rightarrow G_{i,1} = 20*20 + 18*18 = 724$$

- ϵ is 10^{-8}, $\nabla_{\theta i} L(\theta_1)$ is 18, $\alpha = 1.0$.

$$\theta_{i,2} = \theta_{i,1} - \frac{\alpha}{\sqrt{G_{i,0} + \epsilon}} \nabla_{\theta_i} L(\theta_0)$$

$$\theta_{i,1} = 9 - \frac{1}{\sqrt{724 + 10^{-8}}} * 18 = 8.33104$$

Coding Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the function f(x) = x^2
def f(x):
    return x**2

# Derivative of f(x), which is 2*x
def gradient(x):
    return 2 * x
```

Coding Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

```python
# AdaGrad function
def adagrad(starting_x, learning_rate, num_iterations, epsilon=1e-8):
    x = starting_x  # Initial value of x
    grad_squared_sum = 0  # Initialize the sum of squared gradients
    x_values = []   # To store the values of x during the iterations
    f_values = []   # To store the values of f(x)

    for i in range(num_iterations):
        x_values.append(x)
        f_values.append(f(x))

        grad = gradient(x)
        grad_squared_sum += grad**2  # Accumulate the sum of squared gradients

        # Update rule for AdaGrad
        adjusted_learning_rate = learning_rate / (np.sqrt(grad_squared_sum) + epsilon)
        x = x - adjusted_learning_rate * grad  # Update x

        # Print the iteration details
        print(f"Iteration {i+1}: x = {x}, f(x) = {f(x)}, learning rate = {adjusted_learning_rate}")

    return x_values, f_values
```

Coding Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

```
# Parameters for AdaGrad
starting_x = 10    # Start far from the minimum
learning_rate = 1  # Initial learning rate (larger than usual)
num_iterations = 25
epsilon = 1e-8  # Small value to prevent division by zero

# Perform AdaGrad
x_values_adagrad, f_values_adagrad = adagrad(starting_x, learning_rate, num_iterations, epsilon)

x_var = np.linspace(-10,10,num_iterations)
y_var = f(x_var)
plt.plot(x_var,y_var)



# Plot the results
plt.plot(x_values_adagrad,f_values_adagrad, '-*', label='Optimized Parameter')
plt.plot(x_var,y_var, label = 'Loss Function')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('AdaGrad for f(x) = x^2')
plt.legend()
plt.show()
```

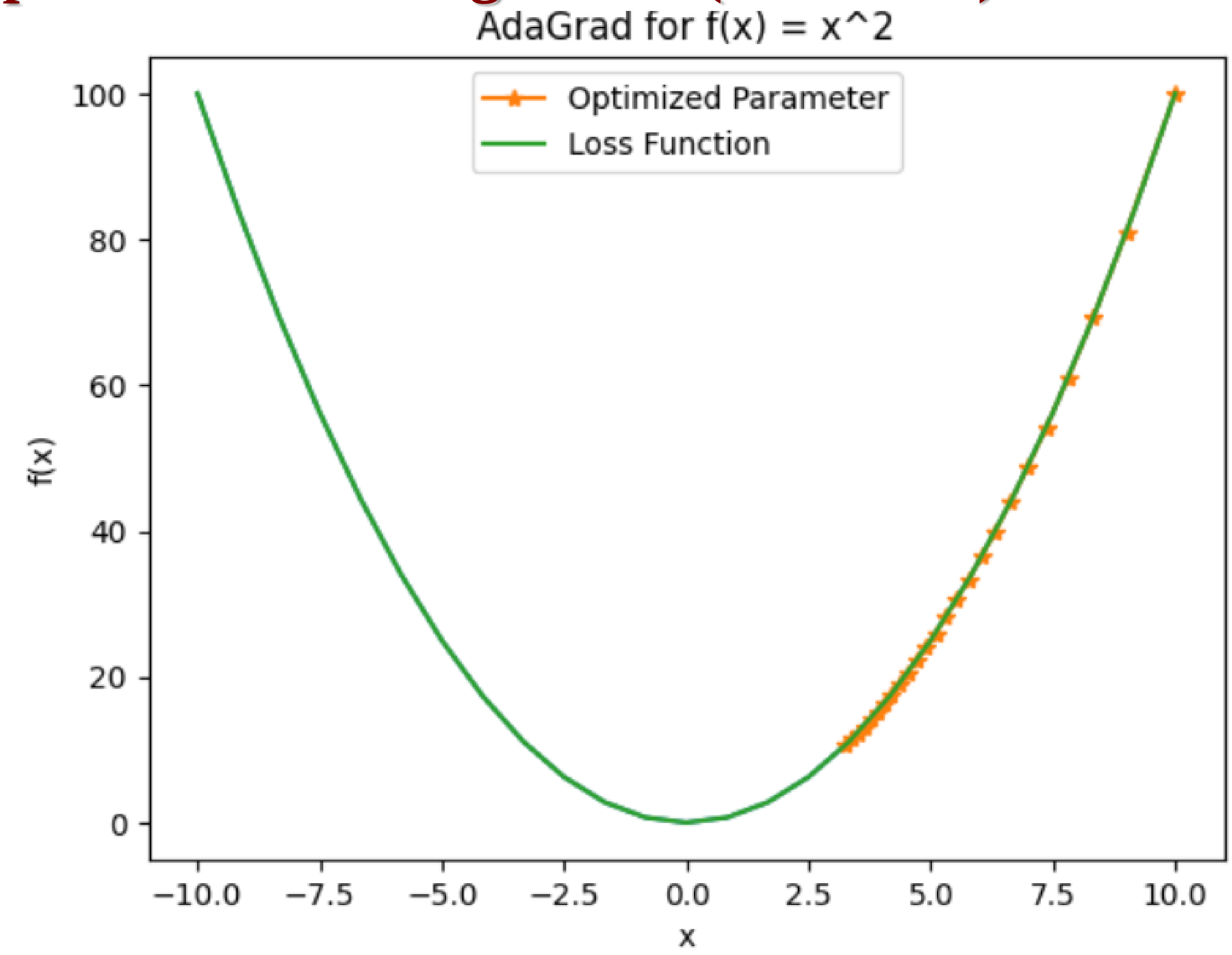
Coding Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)

```
Iteration 1: x = 9.0000000005, f(x) = 81.000000009, learning rate = 0.049999999975
Iteration 2: x = 8.331035269105636, f(x) = 69.406148655082, learning rate = 0.03716470729762175
Iteration 3: x = 7.804561814351674, f(x) = 60.911185114036286, learning rate = 0.031597120750785204
Iteration 4: x = 7.362231327484282, f(x) = 54.202450119390974, learning rate = 0.028337944998654396
Iteration 5: x = 6.977148621485869, f(x) = 48.68060288630216, learning rate = 0.026152581253515023
Iteration 6: x = 6.634323432648377, f(x) = 44.014247408987345, learning rate = 0.02456771436556293
Iteration 7: x = 6.32439447017695, f(x) = 39.99796541440478, learning rate = 0.023357993141111116
Iteration 8: x = 6.041052051430944, f(x) = 36.49430988809801, learning rate = 0.022400754734870177
Iteration 9: x = 5.779803025647223, f(x) = 33.4061230152808, learning rate = 0.021622808706129093
Iteration 10: x = 5.537312004775913, f(x) = 30.66182423823544, learning rate = 0.020977446791463652
Iteration 11: x = 5.311021038418302, f(x) = 28.206944470521815, learning rate = 0.020433286598482796
Iteration 12: x = 5.0989162104553465, f(x) = 25.99894652124431, learning rate = 0.019968366386487038
Iteration 13: x = 4.899377258503165, f(x) = 24.00389752113799, learning rate = 0.0195668004450658
Iteration 14: x = 4.711076758751052, f(x) = 22.194244226844315, learning rate = 0.019216778971787356
Iteration 15: x = 4.532910267695452, f(x) = 20.54727549497885, learning rate = 0.018909317357719394
Iteration 16: x = 4.363946542413079, f(x) = 19.04402942503907, learning rate = 0.018637444302231274
Iteration 17: x = 4.203391208154996, f(x) = 17.66849764879472, learning rate = 0.018395657771887196
Iteration 18: x = 4.050559684115051, f(x) = 16.40703375457822, learning rate = 0.018179550328724727
Iteration 19: x = 3.9048566381226113, f(x) = 15.247905364290222, learning rate = 0.017985544882086084
Iteration 20: x = 3.7657601436683796, f(x) = 14.180949459641296, learning rate = 0.017810704379803668
Iteration 21: x = 3.632809287493172, f(x) = 13.197303319296648, learning rate = 0.017652592186300917
Iteration 22: x = 3.5055943516746795, f(x) = 12.289191758493416, learning rate = 0.017509167940147702
Iteration 23: x = 3.3837489454617797, f(x) = 11.449756925913706, learning rate = 0.01737870871378658
Iteration 24: x = 3.2669436337362607, f(x) = 10.672920706009883, learning rate = 0.017259748522740728
Iteration 25: x = 3.154880728403642, f(x) = 9.953272410452696, learning rate = 0.017151031345535826
```

Coding Parameters Optimization

Adaptive Gradient Algorithm (AdaGrad)



Coding Parameters Optimization

Root Mean Square Propagation (RMSProp)

- The update rule for a parameter θ_i at iteration t is given by:
$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{E[g_{i,t}^2] + \epsilon}} \nabla_{\theta_i} L(\theta_t)$$

where:

- $g_{i,t} = \nabla \theta_i L(\theta_t)$ is the gradient of the loss function with respect to parameter θ_i at iteration t.
- $E[g_{i,t}^2]$ is the exponentially decaying average of past squared gradients for parameter θ_i .
- The $E[g_{i,t}^2]$ is updated as:
$$E[g_{i,t}^2] = \gamma E[g_{i,t-1}^2] + (1-\gamma) g_{i,t}^2$$
- γ is the decay rate (a hyperparameter, typically set to 0.9).

Coding Parameters Optimization

Root Mean Square Propagation (RMSProp)

- For the same example: Iteration 1

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{E[g_{i,t}^2] + \epsilon}} \nabla_{\theta_i} L(\theta_t) \rightarrow \theta_{i,1} = \theta_{i,0} - \frac{\alpha}{\sqrt{E[g_{i,0}^2] + \epsilon}} \nabla_{\theta_i} L(\theta_0)$$

where:

- $g_{i,0} = \nabla\theta_i L(\theta_0)$ is 20, γ is set to 0.9.
- The $E[g_{i,0}^2]$ is calculated as:

$$E[g_{i,t}^2] = \gamma E[g_{i,t-1}^2] + (1-\gamma) g_{i,t}^2$$

$$E[g_{i,0}^2] = 0.9 * 0.0 + (1-0.9) * 400 = 40$$

$$\theta_{i,1} = \theta_{i,0} - \frac{\alpha}{\sqrt{E[g_{i,0}^2] + \epsilon}} \nabla_{\theta_i} L(\theta_0)$$

$$\theta_{i,1} = 10 - \frac{1}{\sqrt{40 + 10^{-8}}} * 20 = 6.83772$$

Coding Parameters Optimization

Root Mean Square Propagation (RMSProp)

- For the same example: Iteration 2

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{E[g_{i,t}^2] + \epsilon}} \nabla_{\theta_i} L(\theta_t) \rightarrow \theta_{i,2} = \theta_{i,1} - \frac{\alpha}{\sqrt{E[g_{i,1}^2] + \epsilon}} \nabla_{\theta_i} L(\theta_1)$$

where:

- $g_{i,1} = \nabla\theta_i L(\theta_1)$ is 13.68, γ is set to 0.9.
- The $E[g_{i,1}^2]$ is calculated as:

$$E[g_{i,1}^2] = \gamma E[g_{i,0}^2] + (1-\gamma) g_{i,0}^2$$

$$E[g_{i,0}^2] = 0.9 * 40.0 + (1-0.9) * 13.68 * 13.68 = 54.7142$$

$$\theta_{i,2} = \theta_{i,1} - \frac{\alpha}{\sqrt{E[g_{i,1}^2] + \epsilon}} \nabla_{\theta_i} L(\theta_1)$$

$$\theta_{i,1} = 6.83772 - \frac{1}{\sqrt{54.71 + 10^{-8}}} * 13.68 = 4.98823$$

Coding Parameters Optimization

Root Mean Square Propagation (RMSProp)

```
# RMSProp function
def rmsprop(starting_x, learning_rate, num_iterations, decay_rate=0.9, epsilon=1e-8):
    x = starting_x  # Initial value of x
    grad_squared_avg = 0  # Initialize the moving average of squared gradients
    x_values = []   # To store the values of x during the iterations
    f_values = []   # To store the values of f(x)

    for i in range(num_iterations):
        x_values.append(x)
        f_values.append(f(x))

        grad = gradient(x)

        # Update the moving average of squared gradients
        grad_squared_avg = decay_rate * grad_squared_avg + (1 - decay_rate) * grad**2

        # Update rule for RMSProp
        adjusted_learning_rate = learning_rate / (np.sqrt(grad_squared_avg) + epsilon)
        x = x - adjusted_learning_rate * grad  # Update x

        # Print the iteration details
        print(f"Iteration {i+1}: x = {x}, f(x) = {f(x)}, learning rate = {adjusted_learning_rate}")

    return x_values, f_values
```

Coding Parameters Optimization

Root Mean Square Propagation (RMSProp)

```
# Parameters for RMSProp
starting_x = 10    # Start far from the minimum
learning_rate = 1  # Initial learning rate
decay_rate = 0.9  # Decay rate for the moving average
num_iterations = 25
epsilon = 1e-8  # Small value to prevent division by zero

# Perform RMSProp
x_values_rmsprop, f_values_rmsprop = \
           rmsprop(starting_x, learning_rate, num_iterations, decay_rate, epsilon)

# Plot the results
plt.plot(x_values_rmsprop,f_values_rmsprop, '-*', label='Optimized Parameter')
plt.plot(x_var,y_var, label = 'Loss Function')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('RMSProp for f(x) = x^2')
plt.legend()
plt.show()
```

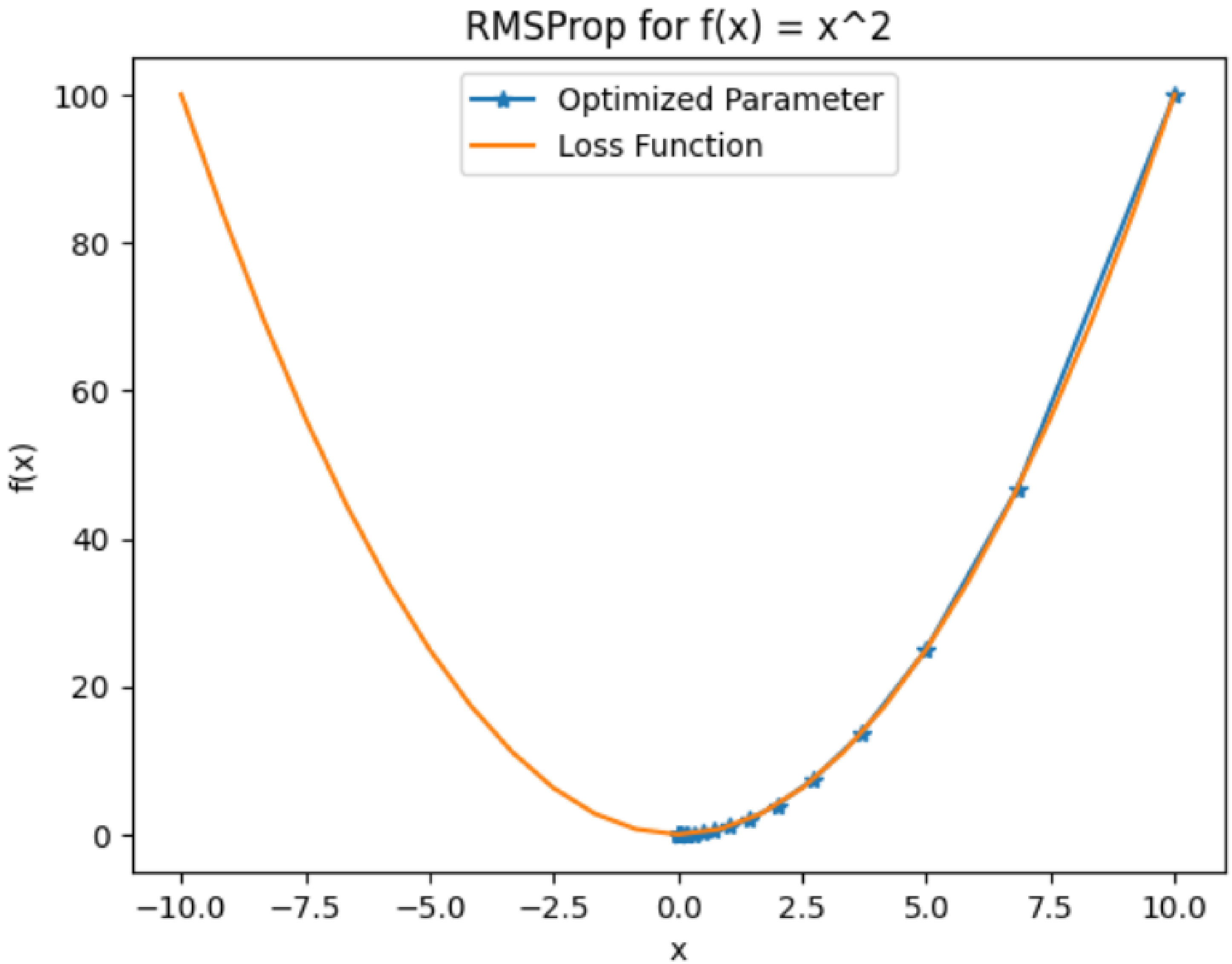
Coding Parameters Optimization

Root Mean Square Propagation (RMSProp)

```
Iteration 1: x = 6.8377223448316204, f(x) = 46.75444686500963, learning rate = 0.15811388275841898
Iteration 2: x = 4.988706075578886, f(x) = 24.88718830851769, learning rate = 0.13520703064598233
Iteration 3: x = 3.6918055386894855, f(x) = 13.629428135498362, learning rate = 0.12998365881266208
Iteration 4: x = 2.728248854368664, f(x) = 7.443341811363928, learning rate = 0.13049938224303984
Iteration 5: x = 1.9979515428511783, f(x) = 3.9918103675814036, learning rate = 0.1338399373554363
Iteration 6: x = 1.4429607379737774, f(x) = 2.0821356913338285, learning rate = 0.1388899562812721 3
Iteration 7: x = 1.0241749420328934, f(x) = 1.0489343118880805, learning rate = 0.14511337173627742
Iteration 8: x = 0.7123800671598245, f(x) = 0.507485360086636, learning rate = 0.15221758611579808
Iteration 9: x = 0.4843702896369644, f(x) = 0.23461457748299677, learning rate = 0.1600337993958115
Iteration 10: x = 0.3211707872482439, f(x) = 0.10315067458165673, learning rate = 0.16846564073019274
Iteration 11: x = 0.20717894627728273, f(x) = 0.0429231157805652, learning rate = 0.17746296596217664
Iteration 12: x = 0.12969144164975607, f(x) = 0.016819870037192083, learning rate = 0.18700622341186024
Iteration 13: x = 0.07856808539588585, f(x) = 0.006172944042775211, learning rate = 0.19709610597102334
Iteration 14: x = 0.04592360102576324, f(x) = 0.0021089771311734824, learning rate = 0.2077464673196173 5
Iteration 15: x = 0.025810939758880276, f(x) = 0.0006662046112365466, learning rate = 0.21897957496407694
Iteration 16: x = 0.013895417466702722, f(x) = 0.00019308262657394 71, learning rate = 0.23082310065982795
Iteration 17: x = 0.007133675126461505, f(x) = 5.088932080989557 6e-05, learning rate = 0.24330835530649686
Iteration 18: x = 0.003474537027013623 3, f(x) = 1.2072407552088668e-05, learning rate = 0.25646935377493935
Iteration 19: x = 0.001595907751163574, f(x) = 2.546921550223976e-06, learning rate = 0.270342388243986 8
Iteration 20: x = 0.0006863492170486867, f(x) = 4.710752477433452 3e-07, learning rate = 0.28496588648426874
Iteration 21: x = 0.00027401748994190 85, f(x) = 7.508558479406392e-08, learning rate = 0.3003804163132957 6
Iteration 22: x = 0.00010049385410321911, f(x) = 1.0099014712519089e-08, learning rate = 0.3166287594917322
Iteration 23: x = 3.341299705846426e-05, f(x) = 1.1164283724289412e-09, learning rate = 0.3337560174369213 5
Iteration 24: x = 9.902961940402537e-06, f(x) = 9.806865519306119e-11, learning rate = 0.3518097325559441 7
Iteration 25: x = 2.558132747664291e-06, f(x) = 6.544043154672456e-12, learning rate = 0.3708400192255859 6
```

Coding Parameters Optimization

Root Mean Square Propagation (RMSProp)



Coding Parameters Optimization

Adaptive Moment Estimation (Adam)

- The update rule for a parameter θ_i at iteration t is given by:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{\hat{v}_{i,t} + \epsilon}} \hat{m}_{i,t}$$

where:

- $\hat{v}_{i,t}$ is the bias-corrected first moment estimate (mean of the gradients).

- $\hat{m}_{i,t}$ is the bias-corrected second moment estimate (variance of the gradients).

Coding Parameters Optimization

Adaptive Moment Estimation (Adam)

- First Moment (Mean) Estimate:

$$m_{i,t} = \beta_1 m_{i,t-1} + (1 - \beta_1) g_{i,t}$$

where:

- $m_{i,\mathrm{t}}$ is the moving average of the gradient.

- β_1 is the exponential decay rate for the first moment (typically β_1=0.9).

- $g_{i,\mathrm{t}}$=$\nabla\theta_i L(\theta_t)$ is the gradient of the loss function with respect to parameter θ_i at iteration t.

Coding Parameters Optimization

Adaptive Moment Estimation (Adam)

- Second Moment (Variance) Estimate:

$$v_{i,t} = \beta_2 v_{i,t-1} + (1-\beta_2) g_{i,t}^2$$

where:

- $v_{i,t}$ is the moving average of the squared gradient.
- β_2 is the exponential decay rate for the second moment (typically β_2=0.999).
- Since $m_{i,t}$ and $v_{i,t}$ are initialized as zeros, they are biased toward zero, therefore, bias-corrected estimates:

$$\hat{m}_{i,t} = \frac{m_{i,t}}{1-\beta_1}, \ \hat{v}_{i,t} = \frac{v_{i,t}}{1-\beta_2}$$

Coding Parameters Optimization

Adaptive Moment Estimation (Adam)

- Iteration 1:

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\alpha}{\sqrt{\hat{v}_{i,t} + \epsilon}} \hat{m}_{i,t} \rightarrow \theta_{i,1} = \theta_{i,0} - \frac{\alpha}{\sqrt{\hat{v}_{i,0} + \epsilon}} \hat{m}_{i,0}$$

$$\theta_{i,0} = 10, \ \alpha = 0.5, \beta_1 = 0.9, \ \beta_2 = 0.999, \ \epsilon = 10^{-8}$$

$$v_{i,-1} = 0, \ m_{i,-1} = 0, \ g_{i.0} = 20$$

$$m_{i,t} = \beta_1 m_{i,t-1} + (1 - \beta_1) g_{i,t} \rightarrow m_{i,0} = (1 - 0.9) * 20 = 2$$

$$v_{i,t} = \beta_2 v_{i,t-1} + (1 - \beta_2) g_{i,t}^2 \rightarrow v_{i,0} = (1 - 0.999) * 400 = 0.4$$

$$\hat{m}_{i,0} = \frac{m_{i,0}}{1 - \beta_1} = \frac{2}{1 - 0.9} = 20, \ \hat{v}_{i,0} = \frac{v_{i,0}}{1 - \beta_2} = \frac{0.4}{1 - 0.999} = 400$$

$$\theta_{i,1} = \theta_{i,0} - \frac{\alpha}{\sqrt{\hat{v}_{i,0} + \epsilon}} \hat{m}_{i,0} \rightarrow \theta_{i,1} = 10 - \frac{0.5}{\sqrt{400}} * 20 = 9.5$$

Coding Parameters Optimization

Adaptive Moment Estimation (Adam)

```
# Adam function
def adam(starting_x, learning_rate, num_iterations, beta1=0.9, beta2=0.999, epsilon=1e-8):
    x = starting_x  # Initial value of x
    m = 0  # Initialize the first moment (mean of gradients)
    v = 0  # Initialize the second moment (mean of squared gradients)
    t = 0  # Time step
    x_values = []   # To store the values of x during the iterations
    f_values = []   # To store the values of f(x)
    for i in range(num_iterations):
        t += 1
        x_values.append(x)
        f_values.append(f(x))
        grad = gradient(x)
        # Update biased first moment estimate (m) and second moment estimate (v)
        m = beta1 * m + (1 - beta1) * grad
        v = beta2 * v + (1 - beta2) * grad**2
        # Compute bias-corrected first and second moment estimates
        m_hat = m / (1 - beta1**t)
        v_hat = v / (1 - beta2**t)
        # Update rule for Adam
        x = x - learning_rate * m_hat / (np.sqrt(v_hat) + epsilon)
        # Print the iteration details
        print(f"Iteration {i+1}: x = {x}, f(x) = {f(x)}, m = {m}, v = {v}")
    return x_values, f_values
```

Coding Parameters Optimization

Adaptive Moment Estimation (Adam)

```
# Parameters for Adam
starting_x = 10    # Start far from the minimum
learning_rate = 0.5  # Initial learning rate
beta1 = 0.9  # Decay rate for first moment
beta2 = 0.999  # Decay rate for second moment
num_iterations = 25
epsilon = 1e-8  # Small value to prevent division by zero

# Perform Adam optimization
x_values_adam, f_values_adam = \
    adam(starting_x, learning_rate, num_iterations, beta1, beta2, epsilon)

# Plot the results
plt.plot(x_values_adam,f_values_adam, '-*', label='Optimized Parameter')
plt.plot(x_var,y_var, label = 'Loss Function')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Adam Optimization for f(x) = x^2')
plt.legend()
plt.show()
```

Coding Parameters Optimization

Adaptive Moment Estimation (Adam)

```
Iteration 1: x = 9.500000000250001, f(x) = 90.25000000475002, m = 1.9999999999999996, v = 0.4000000000000036
Iteration 2: x = 9.00083242855694, f(x) = 81.01498440696223, m = 3.7000000000499993, v = 0.7606000000190007
Iteration 3: x = 8.503116957169732, f(x) = 72.30299798730745, m = 5.130166485756387, v = 1.0838993376468309
Iteration 4: x = 8.007524472326658, f(x) = 64.12044817491032, m = 6.317773228614694, v = 1.372027430258414l
Iteration 5: x = 7.514778901283166, f(x) = 56.47190193517063, m = 7.287500800218556, v = 1.627137195527797l
Iteration 6: x = 7.025658644527531, f(x) = 49.35987938942442, m = 8.061706500453333, v = 1.8513976660729519
Iteration 7: x = 6.5409974707103276, f(x) = 42.7846479118389, m = 8.660667579313506, v = 2.046985785964577
Iteration 8: x = 6.061684707194276, f(x) = 36.74402148943296, m = 9.10280031552422, v = 2.216077391825968
Iteration 9: x = 5.588664535297112, f(x) = 31.233171288087686, m = 9.404857225410653, v = 2.360837400391874
Iteration 10: x = 5.122934178536807, f(x) = 26.244454597620592, m = 9.58210440992901, v = 2.4834092481438326
Iteration 11: x = 4.665540757532327, f(x) = 21.767270560195318, m = 9.64848080464347, v = 2.58590365728617l
Iteration 12: x = 4.217576580430411, f(x) = 17.787952211795076, m = 9.616740875685588, v = 2.670386835869666
Iteration 13: x = 3.780172647157486, f(x) = 14.289705242317636, m = 9.49858210420311, v = 2.7388682578809767
Iteration 14: x = 3.3544901739954085, f(x) = 11.252604327431746, m = 9.304758423214297, v = 2.793288210592366
Iteration 15: x = 2.9417099961522313, f(x) = 8.653657701461961, m = 9.04518061569195, v = 2.835505339691501
Iteration 16: x = 2.5430197831736097, f(x) = 6.466949617612353, m = 8.7290045533532, v = 2.8672844651576574
Iteration 17: x = 2.1595991060771524, f(x) = 4.663868298969236, m = 8.364708054652601, v = 2.8902849791629492
Iteration 18: x = 1.792602523645347l, f(x) = 3.2134238077796673, m = 7.960157070402771, v = 2.9060501673796635
Iteration 19: x = 1.4431410019801139, f(x) = 2.082655951596167, m = 7.522661868091564, v = 2.915997812443402З
Iteration 20: x = 1.112262135376103, f(x) = 1.2371270577914086, m = 7.0590238816784305, v = 2.9214124384373434
Iteration 21: x = 0.8009297829721906, f(x) = 0.6414885172518804, m = 6.575573920585808, v = 2.923439534230072
Iteration 22: x = 0.5100038570394293, f(x) = 0.26010393419509464, m = 6.078202485121665, v = 2.9230820487648494
Iteration 23: x = 0.24022107746654114, f(x) = 0.057706166059185965, m = 5.5723830080173835, v = 2.921199382452865
Iteration 24: x = -0.007822471879864429, f(x) = 6.119106631126973e-05, m = 5.063188922708954, v = 2.9185090077346487
Iteration 25: x = -0.2336861946874083, f(x) = 0.0546092375874813, m = 4.555305536062085, v = 2.91559074349117
```

Coding Parameters Optimization

Adaptive Moment Estimation (Adam)