DDAF Reference Manual

Data-Driven, Dynamic, Online, and Attention-Based Activation Functions

Shyamal Suhana Chandra

2025

Contents

1 Introduction

This reference manual provides comprehensive documentation for the DDAF (Data-Driven, Dynamic, Online, and Attention-Based Activation Functions) library. The library implements adaptive activation functions for various neural network architectures.

2 API Overview

2.1 Header File

All functions and types are declared in `include/ddaf.h`. Include this header in your code:

```
#include "ddaf.h"
```

2.2 Core Types

2.2.1 Activation Types

```
typedef enum {
    DDAF_TYPE_DATA_DRIVEN = 0,
    DDAF_TYPE_DYNAMIC,
    DDAF_TYPE_ONLINE,
    DDAF_TYPE_ATTENTION
} ddaf_type_t;
```

2.2.2 Architecture Types

```
typedef enum {
    DDAF_ARCH_CNN = 0,
    DDAF_ARCH_RNN,
    DDAF_ARCH_LSTM,
    DDAF_ARCH_GRU,
    DDAF_ARCH_TRANSFORMER,
    DDAF_ARCH_HIERARCHICAL_TRANSFORMER,
    DDAF_ARCH_BIGBIRD,
    DDAF_ARCH_MOE
} ddaf_arch_t;
```

3 Core API Functions

3.1 Context Management

3.1.1 ddaf_create_context

Creates a new activation function context.

```
ddaf_context_t* ddaf_create_context(
    ddaf_type_t type,
    ddaf_arch_t arch,
    size_t param_size
);
```

Parameters:

- `type`: Activation function type
- `arch`: Target architecture
- `param_size`: Size of additional parameters (usually 0)

Returns: Pointer to context on success, NULL on failure.

3.1.2 ddaf_destroy_context

Destroys an activation function context and frees associated memory.

```
void ddaf_destroy_context(ddaf_context_t* ctx);
```

Parameters:

- `ctx`: Context to destroy

3.2 Forward and Backward Pass

3.2.1 ddaf_forward

Performs forward pass through activation function.

```
int ddaf_forward(
    ddaf_context_t* ctx,
    const float* input,
    float* output,
    size_t size
);
```

Parameters:

- `ctx`: Activation context
- `input`: Input array
- `output`: Output array (must be pre-allocated)
- `size`: Number of elements

Returns: 0 on success, -1 on failure.

3.2.2 ddaf_backward

Performs backward pass (gradient computation).

```
int ddaf_backward(
    ddaf_context_t* ctx,
    const float* grad_output,
    float* grad_input,
    size_t size
);
```

Parameters:

- `ctx`: Activation context
- `grad_output`: Gradient from next layer
- `grad_input`: Gradient to previous layer (must be pre-allocated)
- `size`: Number of elements

Returns: 0 on success, -1 on failure.

4 Architecture-Specific Initialization

4.1 CNN

```
int ddaf_cnn_init(
    ddaf_context_t* ctx,
    size_t channels,
    size_t height,
    size_t width
);
```

Initializes context for CNN architecture.
Parameters:

- `ctx`: Context (must be created with DDAF_ARCH_CNN)
- `channels`: Number of channels
- `height`: Feature map height
- `width`: Feature map width

4.2 RNN

```
int ddaf_rnn_init(
    ddaf_context_t* ctx,
    size_t hidden_size,
    size_t seq_len
);
```

Initializes context for RNN architecture.

4.3 LSTM

```
int ddaf_lstm_init(
    ddaf_context_t* ctx,
    size_t hidden_size,
    size_t seq_len
);
```

Initializes context for LSTM architecture.

4.4 GRU

```
int ddaf_gru_init(
    ddaf_context_t* ctx,
    size_t hidden_size,
    size_t seq_len
);
```

Initializes context for GRU architecture.

4.5 Transformer

```
int ddaf_transformer_init(
    ddaf_context_t* ctx,
    size_t d_model,
    size_t n_heads,
    size_t seq_len
);
```

Initializes context for Transformer architecture.
Parameters:

- `d_model`: Model dimension
- `n_heads`: Number of attention heads
- `seq_len`: Sequence length

4.6 Hierarchical Transformer

```
int ddaf_hierarchical_transformer_init(
    ddaf_context_t* ctx,
    size_t d_model,
    size_t n_heads,
    size_t n_levels
);
```

Initializes context for Hierarchical Transformer.

4.7 Big Bird

```
int ddaf_bigbird_init(
    ddaf_context_t* ctx,
    size_t d_model,
    size_t n_heads,
    size_t seq_len,
    size_t block_size
);
```

Initializes context for Big Bird architecture.

4.8 Mixture of Experts

```
int ddaf_moe_init(
    ddaf_context_t* ctx,
    size_t d_model,
    size_t n_experts,
    size_t k_experts
);
```

Initializes context for MoE architecture.
Parameters:

- `n_experts`: Total number of experts
- `k_experts`: Number of experts to use per sample

5 Core Activation Type Initialization

5.1 Data-Driven

```
int ddaf_init_data_driven(
    ddaf_context_t* ctx,
    size_t stat_size
);
```

Initializes data-driven activation parameters.

5.2 Dynamic

```
int ddaf_init_dynamic(
    ddaf_context_t* ctx,
    size_t param_count
);
```

Initializes dynamic activation parameters.

5.3 Online

```
int ddaf_init_online(
    ddaf_context_t* ctx,
    size_t buffer_size
);
```

Initializes online activation with buffer.

5.4 Attention-Based

```
int ddaf_init_attention(
    ddaf_context_t* ctx,
    size_t d_model,
    size_t n_heads,
```

```
    size_t seq_len
);
```

Initializes attention-based activation.

6 Memory Management

6.1 Memory Pool

The library uses memory pools for efficient temporary allocations during forward and backward passes. Pools are automatically created with contexts and destroyed when contexts are destroyed.

7 Usage Examples

7.1 Basic CNN Usage

```
// Create context
ddaf_context_t* ctx =
    ddaf_create_context(DDAF_TYPE_DATA_DRIVEN,
                        DDAF_ARCH_CNN, 0);

// Initialize
ddaf_cnn_init(ctx, 64, 32, 32);

// Forward pass
float input[64*32*32];
float output[64*32*32];
ddaf_forward(ctx, input, output, 64*32*32);

// Backward pass
float grad_output[64*32*32];
float grad_input[64*32*32];
ddaf_backward(ctx, grad_output, grad_input, 64*32*32);

// Cleanup
ddaf_destroy_context(ctx);
```

7.2 Transformer Usage

```
// Create context
ddaf_context_t* ctx =
    ddaf_create_context(DDAF_TYPE_ATTENTION,
                        DDAF_ARCH_TRANSFORMER, 0);

// Initialize
ddaf_transformer_init(ctx, 512, 8, 128);

// Use context...
// (forward/backward passes)

```

```
// Cleanup
ddaf_destroy_context(ctx);
```

8 Error Handling

All functions return error codes:

- `0`: Success
- `-1`: Failure (check for NULL pointers, invalid parameters, etc.)

Always check return values and handle errors appropriately.

9 Building the Library

9.1 CMake

```
mkdir build
cd build
cmake ..
make
```

This creates both static (`libddaf_static.a`) and shared (`libddaf_shared.so`) libraries.

10 Performance Considerations

- Memory pools reduce allocation overhead
- Forward and backward passes are optimized for cache efficiency
- Consider batch processing for better performance
- Use appropriate activation types for your use case

11 Limitations

- Currently supports single-precision floating point (float)
- Memory pools have fixed size (1MB default)
- Some architectures may have specific size constraints

12 Copyright