NAPCAS User Guide

NAPCAS Team

May 2025

Contents

1 Introduction

NAPCAS is a lightweight deep learning framework implemented in C++ with Python bindings, designed to mimic PyTorch's functionality. It supports neural network components such as linear layers, convolutional layers, activation functions, loss functions, optimizers, and a data loader.

2 Components

2.1 Tensor

The `Tensor` class is the core data structure for storing multi-dimensional arrays. It supports operations like reshaping, indexing, and gradient computation.

```
import napcas
tensor = napcas.Tensor([2, 3], [1.0, 2.0, 3.0, 4.0, 5.0, 6.0])
print(tensor.shape())  # [2, 3]
```

2.2 Linear Layer

The `Linear` layer performs a linear transformation: $y = xW + b$.

```
linear = napcas.Linear(10, 5)
input = napcas.Tensor([1, 10], [0.5] * 10)
output = napcas.Tensor([1, 5])
linear.forward(input, output)
```

2.3 Conv2d Layer

The `Conv2d` layer applies a 2D convolution operation, optimized with Eigen and im2col.

```
conv = napcas.Conv2d(1, 16, 3)
input = napcas.Tensor([1, 1, 28, 28], [0.5] * 784)
output = napcas.Tensor([1, 16, 26, 26])
conv.forward(input, output)
```

2.4 Activation Functions

Supported activation functions: `ReLU`, `Sigmoid`, `Tanh`.

```
relu = napcas.ReLU()
input = napcas.Tensor([1, 5], [-1.0, -0.5, 0.0, 0.5, 1.0])
output = napcas.Tensor([1, 5])
relu.forward(input, output)
```

2.5 Loss Functions

Supported loss functions: `MSELoss`, `CrossEntropyLoss`.

```
mse = napcas.MSELoss()
y_pred = napcas.Tensor([1, 5], [0.5] * 5)
y_true = napcas.Tensor([1, 5], [1.0] * 5)
loss = mse.forward(y_pred, y_true)
```

2.6 Optimizers

Supported optimizers: SGD, Adam.

```
linear = napcas.Linear(10, 5)
sgd = napcas.SGD([linear], lr=0.01)
adam = napcas.Adam([linear], lr=0.001)
sgd.step()
```

2.7 DataLoader

The DataLoader loads data from a CSV file in batches.

```
dataloader = napcas.DataLoader("dataset.csv", 64)
inputs, targets = dataloader.next()
```

3 Example: Training a Neural Network

```
import napcas
from napcas import Linear, ReLU, MSELoss, SGD, DataLoader

# Create model
model = [
    Linear(784, 128),
    ReLU(),
    Linear(128, 10)
]

# Initialize loss and optimizer
criterion = MSELoss()
optimizer = SGD(model, lr=0.01)

# Load data
dataloader = DataLoader("mnist.csv", 64)

# Training loop
for epoch in range(5):
    total_loss = 0.0
    for inputs, targets in dataloader.next():
        output = inputs
        for layer in model:
            temp = napcas.Tensor(inputs.shape())
            layer.forward(output, temp)
            output = temp
        loss = criterion.forward(output, targets)
        grad = criterion.backward(output, targets)
        grad_input = grad
        for layer in reversed(model):
            temp = napcas.Tensor(inputs.shape())
            layer.backward(grad_input, temp)
            grad_input = temp
        optimizer.step()
        total_loss += loss
    print(f"Epoch {epoch+1}, Loss: {total_loss}")
```

4 Error Handling

NAPCAS includes strict shape checking in all `forward` and `backward` methods to prevent runtime errors. For example:

- `Linear` checks that input shape is `[batch_size, in_features]`.
- `Conv2d` checks that input is a 4D tensor `[batch, in_channels, height, width]`.

5 Performance Optimization

NAPCAS uses Eigen for matrix operations in `Linear` and `Conv2d` (with im2col for convolutions) to improve performance.

6 Testing

Run unit tests with:

```
pytest tests/test_napcas_components.py
pytest tests/test_dataloader_autograd.py
```