GPU-Accelerated Neural Network from Scratch – Project Plan

Goal

Implement a working neural-network training pipeline where all heavy linear algebra runs on GPU via custom CUDA/OpenCL kernels.
Deliverables: working MLP, gradient checks, benchmarks, demo notebook, polished README.

Tech Stack & Environment

- CUDA + C++ (preferred), or OpenCL/Vulkan compute.
- Python for orchestration & plots.
- pybind11 or Python C-API for bindings.
- cmake/nvcc for build, optional Dockerfile.
- matplotlib for visualizations.
- Hardware: any NVIDIA GPU.

High-Level Milestones

1. Scaffolding (repo, build, dataset, tests)
2. CPU baseline MLP in NumPy
3. GPU kernels: GEMM, activations, reductions
4. GPU forward/backward pass for MLP
5. Training loop & optimizers
6. Benchmarking vs CPU/cuBLAS
7. Polishing: float16, autograd, CNNs

Key Kernel Ideas

1. Tiled GEMM kernel with shared memory.
2. Activation & elementwise ops kernels.
3. Softmax + cross-entropy kernel (numerically stable).
4. Manual backward propagation via kernels.

Numerical Correctness & Tests

- Gradient checking with finite differences.
- Validate GPU vs CPU on small inputs.
- Edge cases: zero dims, small/large batches.
- Tolerance: relative error < 1e-4.

Benchmarking Plan

- Compare GPU vs NumPy CPU baseline.
- Metrics: forward/backward runtime, epoch time, throughput.
- Batch sizes: 1, 8, 32, 128, 512.
- Profiling with nvprof/nsys.
- Expect 5–50x speedups on large batches.

Repo Structure

gpu-ml-from-scratch/
■■ src/cpp_cuda/ (kernels, bindings)
■■ src/cpu/ (NumPy reference)

■■ python/ (training code)
■■ notebooks/ (demo.ipynb)
■■ tests/ (unit tests)
■■ Dockerfile, README.md, setup.py

Python API Example

```
model = [Linear(784,256,'cuda'), ReLU(), Linear(256,10,'cuda')]
opt = SGD(model.parameters(), lr=0.01)

for x,y in dataloader:
logits = forward(model, x)
loss, grad = SoftmaxCrossEntropy.forward_and_grad(logits, y)
backward(model, grad)
opt.step()
```

Demo & README Ideas

- Interactive Jupyter demo with sliders.
- Loss curve plots and sample classifications.
- Benchmark charts and profiling snapshots.
- README: goals, build/run, design notes, results, future work.

Interview Talking Points

- Why tiled GEMM? (shared memory efficiency).
- Gradient validation approach.
- Data transfer overheads (pinned memory, async copy).
- Bottlenecks and batch size tradeoffs.
- How to extend to conv layers and mixed precision.

Pitfalls

- Softmax instability (fix with x - max(x)).
- Synchronization issues in kernels.
- Bugs at small dimensions.
- Over-optimizing before correctness checks.