📅 Daily Work Report

Date: July 16, 2025
Author: Dua Mohyyuddin

---

1. 🌅 Morning: Setup & Initial Exploration

✅ Credentials & Environment Setup:

- Received and configured developer credentials.
- Set up Python environment using `conda/pip` with required libraries:
    - `numpy`, `matplotlib`, etc.

🧠 Neural Network Refresher:

Reviewed concepts including:

- Backpropagation & autograd systems.
- Basic NN architectures: MLPs, activation functions.
- Comparison of PyTorch's `autograd` with micrograd.

---

2. 🔍 Current Implementation Analysis

`micrograd.py` provides:

- Core autograd engine via `Value` class.
- Basic neural network components:

    - `Neuron`, `Layer`, `MLP`.
- Training loop using gradient descent.
- Graph-based computation visualization.

---

3. 🧩 Proposed Extensions

A. ✴️ Enhanced `Value` Class (Activation Functions)

```
class Value:
    # ... (existing code)

    def relu(self):
        out = Value(max(0, self.data), (self,), 'ReLU')
        def _backward():
            self.grad += (out.data > 0) * out.grad
        out._backward = _backward
        return out

    def sigmoid(self):
        s = 1 / (1 + math.exp(-self.data))
        out = Value(s, (self,), 'Sigmoid')
        def _backward():
            self.grad += s * (1 - s) * out.grad
        out._backward = _backward
        return out
```

B. ⚙️ Optimizer Classes

```
class SGD:
    def __init__(self, params, lr=0.01):
        self.params = params
        self.lr = lr
```

```
    def step(self):
        for p in self.params:
            p.data -= self.lr * p.grad

    def zero_grad(self):
        for p in self.params:
            p.grad = 0.0


class Adam:
    def __init__(self, params, lr=0.001, beta1=0.9, beta2=0.999):
        self.params = params
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.m = [0] * len(params)
        self.v = [0] * len(params)
        self.t = 0

    def step(self):
        self.t += 1
        for i, p in enumerate(self.params):
            self.m[i] = self.beta1 * self.m[i] + (1 - self.beta1) *
p.grad
            self.v[i] = self.beta2 * self.v[i] + (1 - self.beta2) *
p.grad**2
            m_hat = self.m[i] / (1 - self.beta1**self.t)
            v_hat = self.v[i] / (1 - self.beta2**self.t)
            p.data -= self.lr * m_hat / (math.sqrt(v_hat) + 1e-8)

    def zero_grad(self):
        for p in self.params:
            p.grad = 0.0
```

C. 📉 Loss Functions

```
def mse_loss(preds, targets):
    return sum((y_pred - y_true)**2 for y_pred, y_true in zip(preds,
targets))

def cross_entropy(preds, targets):
    # Binary classification
    loss = sum(-(y_true * math.log(y_pred.data) +
                 (1 - y_true) * math.log(1 - y_pred.data))
               for y_pred, y_true in zip(preds, targets))
    return loss
```

D. 📦 Batch Processing (Mini-batch Training)

```
class DataLoader:
    def __init__(self, X, y, batch_size=32, shuffle=True):
        self.X = X
        self.y = y
        self.batch_size = batch_size
        self.shuffle = shuffle

    def __iter__(self):
        n = len(self.X)
        if self.shuffle:
            idx = np.random.permutation(n)
            self.X = [self.X[i] for i in idx]
            self.y = [self.y[i] for i in idx]

        for i in range(0, n, self.batch_size):
            yield (self.X[i:i+self.batch_size],
                   self.y[i:i+self.batch_size])
```

---

4. 🧪 Example Usage of Extended Features

```
# Initialize network
```

```
model = MLP(3, [4, 4, 1])
optim = Adam(model.parameters(), lr=0.01)

# Training loop with batching
loader = DataLoader(xs, ys, batch_size=2)
for epoch in range(100):
    for x_batch, y_batch in loader:
        optim.zero_grad()
        preds = [model(x) for x in x_batch]
        loss = mse_loss(preds, y_batch)
        loss.backward()
        optim.step()
    print(f"Epoch {epoch}, Loss: {loss.data:.4f}")
```

---

5. 🧠 Visualization Enhancements

```
def draw_graph(root, filename=None):
    # ... (existing visualization logic)

    # Add gradient values to graph
    for node, (x, y) in pos.items():
        if node.grad != 0:
            plt.text(x, y - 0.3, f"∇: {node.grad:.2f}",
                     ha='center', va='top', fontsize=7, color='blue')

    if filename:
        plt.savefig(filename)
    plt.show()
```

---

6. 🧪 Testing the Extended Version

```
def test_extended_features():
    print("\nTesting extended features...")
    # Test activations
```

```
x = Value(0.5)
print(f"ReLU(0.5): {x.relu().data:.4f}")
print(f"Sigmoid(0.5): {x.sigmoid().data:.4f}")

# Test Adam optimizer
model = MLP(2, [4, 1])
optim = Adam(model.parameters())
y = model([1.0, -1.0])
y.backward()
optim.step()
print("Adam optimization step completed")
```

---

7. 🌇 Afternoon: FiCAP Project Analysis

🎯 Task:

Reviewed FiCAP functional requirements and created development estimates.

🛠️ FiCAP Goals:

- Assist victims of **romance/investment scams**.
- AI-powered, emotionally aware, modular system.

📌 Proposed Architecture

Module	Functionality	Tools/Frameworks	Est. Days
Interviewer Agent	Emotionally aware Q&A	GPT-4 API, LangChain, React	20
Document Parser Agent	Extract text from PDFs/images	Tesseract OCR, spaCy	15

Pattern Matching Agent	Detect scam patterns	Sentence Transformers, Pinecone	14
Report Generator Agent	Generate legal reports (PDFs, etc.)	GPT, HTML/PDF templates	9
Lead Coordinator Agent	Manage flow among agents	LangChain / Async orchestrator	13

Strategy: MVP-first → iterate on quality and complexity.

---

8. 🌙 Evening: Communication & Problem-Solving

✅ Tasks Completed:

- Explained micrograd extension work to supervisors.
- Walked through FiCAP estimate plan and design decisions.
- Proposed:
    - **Scalability:** Async job queues (e.g., Celery).
    - **Cost Optimization:** Use GPT-4 selectively, combine with OSS tools.

---

9. ✅ Key Takeaways

- **Autograd Mastery:** Strengthened foundational understanding via hands-on `micrograd` modification.
- **FiCAP Planning:** Created a modular, agent-driven plan for scam victim support system.
- **Effective Communication:** Translated technical designs into non-technical language for stakeholders.

Signed,

Dua Mohyyuddin
Email: duamohyyuddin@gmail.com