Fully Connected Neural Networks (FCNN)

Machine Learning Hardware Course - Lab 2

Today's Lab Objectives

- Implement FCNNs with different architectures
- Analyze the impact of network depth and width
- Compare activation functions and regularization techniques
- Measure memory usage and computational efficiency
- Identify optimal architectures for different use cases

What are Fully Connected Neural Networks?

- Every neuron in one layer connects to every neuron in the next layer
- Input layer → Hidden layer(s) → Output layer
- Each connection has a learnable weight
- Non-linear activation functions introduce complexity

Show Image

Key Architectural Choices

Network Depth (number of hidden layers)

- Deeper networks can learn more complex functions
- But may be harder to train (vanishing gradients)

Network Width (neurons per layer)

- Wider networks can represent more features
- But increase parameter count quadratically

Activation Functions

- Introduce non-linearity

- Common choices: ReLU, Sigmoid, Tanh, ELU

Regularization

- Prevent overfitting
- Techniques: Dropout, L1/L2 regularization

Implementation in TensorFlow/Keras

python Copy

```python
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

We'll use this as a starting point and experiment with:

- Different numbers of layers
- Different layer widths
- Various activation functions
- Different dropout rates

Performance Metrics We'll Measure

Accuracy Metrics

- Test accuracy (%)
- Training-validation gap (overfitting indicator)

Computational Metrics

- Training time (seconds)
- Inference time (milliseconds per sample)
- Parameter count

Efficiency Metrics

- Accuracy per million parameters
- Parameters trained per second

- Memory usage

Hardware Implications

Parameters and Memory

- More parameters = more memory required
- Both for model storage and during computation

Training Efficiency

- Training time scales with model size
- But not necessarily linearly!

Inference Speed

- Critical for deployment scenarios
- Affected by model architecture and hardware

Power Consumption

- Larger models consume more energy
- Important for edge/mobile devices

Today's Experiments

1. **Network Depth**: Compare 1-4 hidden layers
2. **Network Width**: Test 64, 128, 256, 512 neurons
3. **Activation Functions**: ReLU, Sigmoid, Tanh, ELU
4. **Regularization**: Dropout rates from 0.0 to 0.6
5. **Dataset Comparison**: MNIST vs. Fashion MNIST
6. **Memory Profiling**: Measure memory requirements

Expected Patterns

Depth vs. Width

- Depth increases modeling power with fewer parameters
- Width can be more parallelizable on GPUs

Activation Functions

- ReLU is typically faster to train

- Sigmoid/Tanh may have better properties for some tasks

Regularization

- Dropout improves generalization but may slow convergence
- Higher dropout = stronger regularization

Practical Applications

Mobile/Edge Deployment

- Prioritize smaller models with fast inference
- Memory and power constraints

Cloud-based Services

- Can handle larger models
- May prioritize accuracy over size/speed

Real-time Systems

- Low inference latency is critical
- May sacrifice some accuracy for speed

Pareto Efficiency Concept

- Multiple competing objectives (accuracy, speed, size)
- A model is "Pareto efficient" if no other model is better in all objectives
- Helps identify the best trade-offs

Show Image

Lab Timeline

- Environment Setup: 10 minutes
- Dataset Preparation: 10 minutes
- FCNN Implementation: 30 minutes
- Hyperparameter Experimentation: 30 minutes

- Model Profiling: 20 minutes
- Performance Analysis & Worksheet: 20 minutes

Tips for Success

- Use GPU runtime in Colab (Runtime > Change runtime type)
- Pay attention to the relative differences between models
- Record results systematically for the worksheet
- If you encounter memory issues, reduce batch size or restart runtime
- Focus on understanding the patterns more than absolute numbers

---

Let's Get Started!

1. Open Google Colab
2. Create a new notebook
3. Enable GPU runtime
4. Follow the lab guide step-by-step

Good luck!