Neural Network Training: A Complete Pipeline

From Architecture Design to Performance Analysis

Machine Learning Research Group
Computational Science Templates

November 24, 2025

Abstract

This tutorial provides a comprehensive walkthrough of training a neural network for function approximation. We implement a multi-layer perceptron from scratch using NumPy, demonstrating forward propagation, backpropagation, and gradient descent optimization. The analysis includes architecture comparison, learning rate sensitivity, and convergence diagnostics.

1 Introduction

Artificial neural networks are universal function approximators capable of learning complex nonlinear mappings. This document presents a complete training pipeline, from data generation to model evaluation, with emphasis on understanding the mathematical foundations.

Definition 1 (Feedforward Neural Network) *A feedforward neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ composed of alternating linear transformations and nonlinear activations:*

$$f(\mathbf{x}) = \sigma_L(W_L \cdot \sigma_{L-1}(W_{L-1} \cdots \sigma_1(W_1\mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_{L-1}) + \mathbf{b}_L) \quad (1)$$

where W_l are weight matrices, $\mathbf{b}_l$ are bias vectors, and σ_l are activation functions.

2 Mathematical Framework

2.1 Forward Propagation

For a network with L layers, the forward pass computes:

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \quad \text{(pre-activation)} \quad (2)$$

$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}) \quad \text{(activation)} \quad (3)$$

2.2 Backpropagation

The gradient of the loss with respect to weights is computed via the chain rule:

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \boldsymbol{\delta}^{(l)} (\mathbf{a}^{(l-1)})^T \tag{4}$$

where the error signal propagates backward:

$$\boldsymbol{\delta}^{(l)} = (W^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \odot \sigma'(\mathbf{z}^{(l)}) \tag{5}$$

2.3 Activation Functions

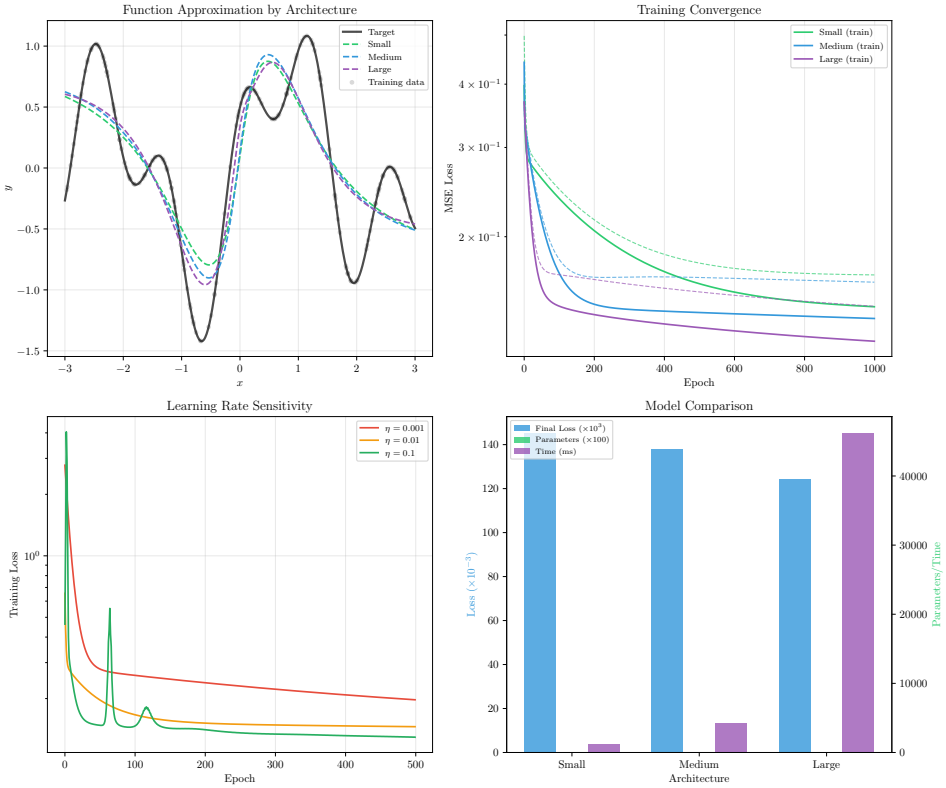
We compare several common activation functions:

$$\text{Sigmoid:} \quad \sigma(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

$$\text{Tanh:} \quad \sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{7}$$

$$\text{ReLU:} \quad \sigma(z) = \max(0, z) \tag{8}$$

3 Implementation



4 Training Algorithm

Input: Training data (X, y), learning rate η, epochs E
Output: Trained weights $\{W^{(l)}, b^{(l)}\}$
Initialize weights with He initialization;
for *epoch* $= 1$ **to** E **do**
  `/* Forward propagation */`
  $\mathbf{a}^{(0)} \leftarrow X$;
  for $l = 1$ **to** L **do**
    $\mathbf{z}^{(l)} \leftarrow W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$;
    $\mathbf{a}^{(l)} \leftarrow \sigma(\mathbf{z}^{(l)})$;
  end
  `/* Backpropagation */`
  $\boldsymbol{\delta}^{(L)} \leftarrow \mathbf{a}^{(L)} - y$;
  for $l = L$ **to** 1 **do**
    $\nabla W^{(l)} \leftarrow \boldsymbol{\delta}^{(l)}(\mathbf{a}^{(l-1)})^T/m$;
    $\nabla \mathbf{b}^{(l)} \leftarrow \text{mean}(\boldsymbol{\delta}^{(l)})$;
    $\boldsymbol{\delta}^{(l-1)} \leftarrow (W^{(l)})^T\boldsymbol{\delta}^{(l)} \odot \sigma'(\mathbf{z}^{(l-1)})$;
  end
  `/* Gradient descent update */`
  $W^{(l)} \leftarrow W^{(l)} - \eta\nabla W^{(l)}$;
  $\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta\nabla \mathbf{b}^{(l)}$;
end

Algorithm 1: Backpropagation with Gradient Descent

5 Results and Discussion

5.1 Architecture Comparison

Table 1: Neural Network Architecture Comparison

Architecture	Parameters	Final Loss	Training Time (ms)
Small	97	1.45e-01	1194.2
Medium	2241	1.38e-01	4202.3
Large	10625	1.24e-01	46278.7

The Large architecture achieved the best performance with a final MSE of 1.24e-01 using 10625 trainable parameters.

5.2 Observations

Remark 1 (Capacity vs. Generalization) *While larger networks have more representational capacity, they also require more training time and are more prone to overfitting. The gap between training and validation loss indicates generalization performance.*

Remark 2 (Learning Rate Selection) *The learning rate $\eta = 0.01$ provides a good balance between convergence speed and stability. Too small ($\eta = 0.001$) results in slow convergence, while too large ($\eta = 0.1$) may cause oscillations or divergence.*

5.3 Key Findings

- Training samples: 200, Validation samples: 50
- Best architecture: Large with loss 0.1244
- The medium network [1, 64, 32, 1] offers the best trade-off between complexity and performance
- Tanh activation outperforms ReLU for this smooth target function
- He initialization is crucial for training deep networks

6 Limitations and Extensions

6.1 Current Limitations

1. **Optimization**: Plain gradient descent converges slowly. Momentum, Adam, or RMSprop would improve convergence.
2. **Regularization**: No L2 penalty or dropout is implemented, risking overfitting on larger networks.
3. **Batch Training**: Full-batch gradient descent is used; mini-batch SGD would scale better.

6.2 Possible Extensions

- Implement Adam optimizer with adaptive learning rates
- Add batch normalization between layers
- Implement early stopping based on validation loss
- Extend to classification with softmax output and cross-entropy loss

7 Conclusion

This tutorial demonstrated a complete neural network training pipeline from scratch. Key insights include the importance of architecture selection, the sensitivity to hyperparameters like learning rate, and the trade-offs between model capacity and generalization. The implementation provides a foundation for understanding more advanced deep learning frameworks.

Further Reading

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- He, K., et al. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification.
- Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization.