Comprehensive Theoretical Reports on MLP Training, Optimization, Regularization, and Stability

By Shaurya Handu

Overview

This document contains three in-depth theoretical reports:

1. Backpropagation and Gradient-Based Optimization in MLPs
2. Regularization in MLPs
3. Weight Initialization and Training Stability in MLPs

Each section includes mathematical formulations, conceptual explanations, and connections to practical training of deep neural networks.

1 Backpropagation and Gradient-Based Optimization in MLPs

1.1 Forward Pass of an MLP

A multilayer perceptron (MLP) with L layers computes:

$$\mathbf{h}^{(0)} = \mathbf{x}$$

$$\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{h}^{(l)} = f^{(l)}\left(\mathbf{a}^{(l)}\right)$$

where:

$$\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}, \quad \mathbf{b}^{(l)} \in \mathbb{R}^{n_l},$$

and $f(\cdot)$ is a non-linear activation function (ReLU, sigmoid, tanh, etc.)

The output layer produces $\hat{y}$.

1.2 Loss Functions in Neural Networks

Typical choices:

- Binary Cross Entropy (BCE):

$$\mathcal{L}_{\text{BCE}} = -\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right]$$

- Mean Squared Error (MSE):

$$\mathcal{L}_{\text{MSE}} = \frac{1}{2}(y-\hat{y})^2$$

1.3 Backpropagation

Backpropagation computes $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}}$ using the chain rule.

Let the error at layer l be:

$$\delta^{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l)}}$$

For output layer:

$$\delta^{(L)} = (\hat{y} - y) \odot f'(\mathbf{a}^{(L)})$$

For hidden layers (chain rule):

$$\delta^{(l)} = \left(\mathbf{W}^{(l+1)T}\delta^{(l+1)}\right) \odot f'(\mathbf{a}^{(l)})$$

Gradients:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \delta^{(l)}\mathbf{h}^{(l-1)^T}, \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

1.4 Gradient Descent

Basic update rule:

$$\theta := \theta - \eta \nabla_\theta \mathcal{L}$$

where η is the learning rate.

1.5 Problems: Vanishing and Exploding Gradients

When $|f'(x)| < 1$ (sigmoid, tanh):

$$\nabla \sim \prod_l f'^{(l)} \to 0$$

When weights are too large:

$$\nabla \sim \prod_l \mathbf{W}^{(l)} \to \infty$$

1.6 Adam Optimizer

Adam maintains exponential moving averages:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Bias-corrected:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Parameter update:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

1.7 RMSprop

Tracks squared gradients:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2$$

Update:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} g_t$$

1.8 Learning Rate

Too large $\implies$ divergence. Too small $\implies$ slow learning, stuck in local minima.

—

2 Regularization in MLPs

2.1 Overfitting vs Underfitting

- Overfitting: model learns noise; low training loss, high test loss.
- Underfitting: model too simple; high training and test loss.

2.2 L1 and L2 Regularization

L2 (weight decay):

$$\mathcal{L}_{\text{reg}} = \lambda \sum_i w_i^2$$

Gradient becomes:

$$w := w - \eta(\nabla w + \lambda w)$$

L1 regularization:

$$\mathcal{L}_{\text{reg}} = \lambda \sum_i |w_i|$$

Promotes sparsity.

2.3 Dropout

Randomly zeros neurons:

$$h_i^{(l)} := \begin{cases} 0, & \text{with prob } p, \\ \frac{h_i}{1-p}, & \text{otherwise.} \end{cases}$$

Prevents co-adaptation of neurons.

2.4 Batch Normalization

Given pre-activation a:

$$\mu = \frac{1}{m}\sum_i a_i, \qquad \sigma^2 = \frac{1}{m}\sum_i (a_i - \mu)^2$$

Normalize:

$$\hat{a} = \frac{a - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Then learn affine transform:

$$y = \gamma \hat{a} + \beta$$

Benefits:

- Stabilizes training
- Allows higher learning rate
- Regularizing effect

2.5 Layer Normalization

Same idea as batchnorm but normalizes across features, not batch axis.

2.6 Early Stopping

Stop training when validation loss increases:

$$\mathcal{L}_{val}(t) < \mathcal{L}_{val}(t-1)$$

Prevents overfitting without modifying the model.
—

3 Weight Initialization and Training Stability

3.1 Why Initialization Matters

Poor initialization can cause:

- vanishing/exploding gradients
- slow convergence
- unstable optimization

3.2 Xavier (Glorot) Initialization

Designed for tanh/sigmoid activations.
Goal:

$$\mathrm{Var}(a^{(l)}) = \mathrm{Var}(a^{(l-1)})$$

Weights sampled as:

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right) \quad \text{or} \quad W \sim U\left(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right)$$

3.3 He Initialization

Designed for ReLU activations.

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right)$$

Compensates for half the neurons being zero due to ReLU.

3.4 ReLU and Dying ReLU

ReLU:

$$f(x) = \max(0, x)$$

Dying ReLU occurs when:

$$x < 0 \quad \Rightarrow \quad f(x) = 0, f'(x) = 0$$

Neuron receives no gradient and stops learning.
Alternatives: LeakyReLU, ELU, GELU.

3.5 Softmax and Numerical Stability

Softmax:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Stable form:

$$\text{softmax}(z_i) = \frac{e^{z_i - \max_j z_j}}{\sum_j e^{z_j - \max_k z_k}}$$

Prevents overflow from large exponentials.

3.6 Gradient Clipping

Prevents exploding gradients:

$$g := \frac{\tau}{\|g\|} g, \quad \text{if } \|g\| > \tau$$

Conclusion

These three reports together provide a deep theoretical understanding of MLP training, optimization, stability, and generalization. The mathematics of backpropagation, initializations, and regularization form the foundation of modern deep learning techniques.