Day74_Deep_Learning_VanishingGradient_Dropout_Optimization_LossFur

August 28, 2025

# 1 Deep Learning (Vanishing Gradient, Dropout, Optimization, Loss Functions)

Welcome back to my Deep Learning documentation!

In this notebook, we continue from **Day73**, focusing on important training concepts in neural networks:

- Vanishing Gradient Problem
- Chain Rule in Backpropagation
- Dropout Neurons
- Optimization Techniques
- Loss Functions

# 2 Vanishing Gradient Problem

Deep neural networks often face the **Vanishing Gradient Problem** during training.

## 2.1 What is it?

- In deep networks, gradients become **very small** as they propagate backward.
- This makes weight updates negligible in early layers → **network stops learning**.

## 2.2 Why does it happen?

- Sigmoid and Tanh activations squash values into a small range.
- Their derivatives are small:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \quad \in (0, 0.25)$$

$$\tanh'(x) = 1 - \tanh^2(x) \quad \in (0, 1)$$

- Multiplying many small derivatives → gradient approaches 0.

### 2.3 Symptoms

- Accuracy stops improving after a few epochs.
- Model seems "stuck" at some accuracy.

### 2.4 Solutions

1. Use **ReLU / Leaky ReLU** instead of Sigmoid/Tanh.
2. Apply **Batch Normalization** → keeps activations stable.
3. Use better **optimizers** (Adam, RMSProp).
4. **Dropout Neurons** → improves generalization.

## 3 Chain Rule in Backpropagation

Backpropagation is based on the **Chain Rule of Calculus**.

### 3.1 Formula

If:

$$y = f(g(x))$$

Then derivative:

$$\frac{dy}{dx} = f'(g(x)) \cdot g'(x)$$

### 3.2 In Neural Networks

- Error flows backward from **Output → Hidden → Input**.
- Each layer's gradient is computed as a product of partial derivatives.

Example (simple 2-layer NN):

$$L = f(z), \quad z = g(h), \quad h = w \cdot x$$

$$\frac{dL}{dw} = \frac{dL}{dz} \cdot \frac{dz}{dh} \cdot \frac{dh}{dw}$$

This shows how gradients are **chained** across layers.

# 4 Dropout Neurons

Dropout is a **regularization technique** used to prevent overfitting.

## 4.1 What is Dropout?

- During training, **random neurons are dropped** (set to 0).
- This forces the network to not depend on specific neurons.

## 4.2 Example

- Suppose a layer has 100 neurons.
- If dropout = 0.5 → only ~50 neurons are active at each step.

## 4.3 At Inference (Testing)

- No dropout is applied.
- Weights are **scaled** to match training conditions.

Dropout improves generalization and reduces overfitting.

# 5 Optimization in Deep Learning

Optimizers control **how weights are updated** during training.

## 5.1 Types of Gradient Descent

### 5.1.1 Stochastic Gradient Descent (SGD)

- Updates weights after **each sample**.
- Faster but noisy.

### 5.1.2 Batch Gradient Descent (BGD)

- Uses the **entire dataset** for each update.
- Very accurate but slow.

### 5.1.3 Mini-Batch Gradient Descent

- Uses small batches of data.
- Default choice in practice.

## 5.2 Advanced Optimizers

### 5.2.1 Adam (Adaptive Moment Estimation)

- Combines **momentum + adaptive learning rate**.
- Very popular for deep learning.

### 5.2.2 Adamax

- Variant of Adam.
- Works better with **sparse gradients**.

### 5.2.3 Adadelta

- Dynamically adjusts learning rate.

### 5.2.4 RMSProp

- Keeps moving average of squared gradients.
- Prevents exploding/vanishing updates.

# 6 Loss Functions

Loss functions measure **how far predictions are from actual values**.

## 6.1 Regression Losses

### 6.1.1 Mean Absolute Error (MAE)

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_{true}^{(i)} - y_{pred}^{(i)}|$$

### 6.1.2 Mean Squared Error (MSE)

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(y_{true}^{(i)} - y_{pred}^{(i)}\right)^2$$

### 6.1.3 Log Loss

- Used for probabilistic regression.

## 6.2 Classification Losses

### 6.2.1 Binary Cross-Entropy

For binary classification (0/1):

$$Loss = -\frac{1}{n}\sum_{i=1}^{n}\Big[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\Big]$$

#### 6.2.2 Categorical Cross-Entropy

For multi-class classification (one-hot labels):

$$Loss = -\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

#### 6.2.3 Sparse Categorical Cross-Entropy

- Similar to categorical cross-entropy, but labels are **integers** instead of one-hot.

## 7 Key Insights

- Backpropagation = **Forward Propagation + Chain Rule + Weight Updates**.
- If weights stop updating → **Vanishing Gradient Problem**.
- **Dropout Neurons + L1/L2 Regularization** help reduce overfitting.
- Advanced optimizers (Adam, RMSProp, etc.) improve convergence speed.
- Different loss functions are used for **regression vs classification** tasks.

## 8 Summary

- Vanishing Gradient slows training → solved with ReLU, Dropout, BatchNorm, better optimizers.
- Chain Rule = backbone of backpropagation.
- Dropout Neurons = prevent overfitting.
- Optimizers = decide how learning happens.
- Loss Functions = measure error in regression/classification.

## 9 Conclusion

In this notebook, we covered:
- Vanishing Gradient Problem
- Chain Rule in Backpropagation
- Dropout Neurons

- Optimization Methods (SGD, Adam, RMSProp, etc.)
- Loss Functions (Regression & Classification)

These concepts help us train **deeper and more accurate neural networks**.