📝 Day 19: Optimizers in Deep Learning

◆ Introduction to Optimizers

Optimizers are crucial components in deep learning. They help the model improve by minimizing the loss function and adjusting the weights and biases of the neural network based on the gradients calculated through backpropagation.

An optimizer updates the model parameters to reduce the error. The quality of learning depends heavily on the optimizer used.

◆ Why Optimizers Matter

- Control the **speed** and **quality** of learning
- Help models **converge faster**
- Avoid local minima and saddle points
- Determine how **weights are updated** during training

◆ Types of Optimizers

✅ 1. Stochastic Gradient Descent (SGD)

- **Formula**:
  `θ = θ - η * ∇J(θ)`
  where `θ` is the weight, `η` is the learning rate, and `∇J(θ)` is the gradient of the cost.
- **Pros**: Simple, memory-efficient
- **Cons**: Can oscillate, slow convergence

✅ 2. Momentum

- Adds a fraction of the previous update to the current one.
- Helps accelerate SGD in the relevant direction and dampens oscillations.

✅ 3. RMSProp

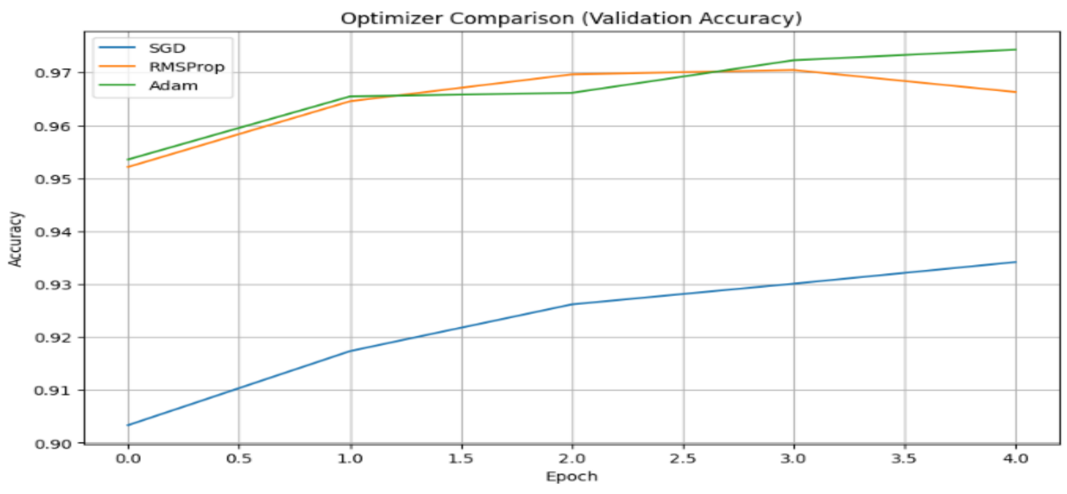
- Adjusts the learning rate for each parameter.
- Uses a moving average of squared gradients to normalize updates.

✅ 4. Adam (Adaptive Moment Estimation)

- Combines Momentum and RMSProp.
- Maintains per-parameter learning rates that are adapted based on first and second moments of the gradients.
- **Most commonly used optimizer today.**

◆ Code Example: Comparing Optimizers in TensorFlow

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import SGD, RMSprop, Adam
import matplotlib.pyplot as plt

# Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Simple model
def create_model(optimizer):
    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

# Train and compare optimizers
optimizers = {'SGD': SGD(), 'RMSProp': RMSprop(), 'Adam': Adam()}
history_dict = {}

for name, opt in optimizers.items():
    print(f"\nTraining with {name} optimizer:")
    model = create_model(opt)
    history = model.fit(x_train, y_train, epochs=5, validation_split=0.2,
verbose=0)
    history_dict[name] = history

# Plot accuracy comparison
plt.figure(figsize=(10, 6))
for name, history in history_dict.items():
    plt.plot(history.history['val_accuracy'], label=name)
plt.title("Optimizer Comparison (Validation Accuracy)")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```



◆ Summary

Optimizer	Strengths	Weaknesses
SGD	Simple, lightweight	Slow, may oscillate
Momentum	Faster convergence	Needs tuning
RMSProp	Good for RNNs	Learning rate needs adjustment
Adam	Adaptive, fast, best for most problems	Slightly higher memory

📌 **Conclusion**:
Choosing the right optimizer significantly affects the training time and final accuracy of a neural network. **Adam** is the go-to optimizer for most deep learning tasks due to its efficiency and adaptability.