📔 Day 19 – Optimizers in Deep Learning

🔍 Understanding Optimizers

In deep learning, *optimizers* play a vital role in the learning process. They're like smart guides that help the neural network improve its performance by minimizing the loss function. They do this by adjusting the model's internal parameters (weights and biases) based on the error calculated during training.

An effective optimizer can significantly reduce the training time, improve accuracy, and ensure better convergence.

💡 Why Optimizers Are Important

Optimizers impact how efficiently and effectively a neural network learns. Their importance includes:

- 📈 Controlling how fast or slow a model learns (learning rate)
- ▯ Helping avoid local minima or saddle points in the loss function
- 🔁 Determining the direction and magnitude of parameter updates
- ⏱️ Improving training stability and convergence speed

⚙️ Popular Types of Optimizers

✅ 1. Stochastic Gradient Descent (SGD)

- **How it works**: Updates weights based on a single training example at a time.
- **Formula**:

θ=θ−η⋅∇J(θ)\theta = \theta - \eta \cdot \nabla J(\theta)θ=θ−η⋅∇J(θ)

where:

  - θ\thetaθ: weights
  - η\etaη: learning rate
  - ∇J(θ)\nabla J(\theta)∇J(θ): gradient of cost function

- **Pros**: Easy to implement, low memory usage
- **Cons**: May fluctuate and converge slowly

✅ 2. Momentum

- Enhances SGD by adding a "velocity" term from previous updates.
- This smooths out updates and helps accelerate in the right direction.
- Especially useful in ravines and curved cost surfaces.

✅ 3. RMSProp (Root Mean Square Propagation)

- Adapts learning rates individually for each parameter.
- Keeps a moving average of the squared gradients.
- Frequently used in training recurrent neural networks (RNNs).

✅ 4. Adam (Adaptive Moment Estimation)

- Hybrid of Momentum and RMSProp.
- Maintains both average of gradients and squared gradients.
- Most commonly used due to its robust performance across various tasks.

□ Hands-On: 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 and normalize data

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

# Function to build 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

# Compare performance of different optimizers
```

```
optimizers = {'SGD': SGD(), 'RMSProp': RMSprop(), 'Adam': Adam()}
history_dict = {}
or 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 validation accuracy
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("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

📊 Quick Comparison

Optimizer	**Strengths**	**Limitations**
SGD	Simple, memory-efficient	May be slow, unstable
Momentum	Accelerates convergence	Requires tuning of momentum rate
RMSProp	Works well on noisy data	Sensitive to learning rate
Adam	Adaptive, efficient, widely used	Uses more memory

📌 Conclusion

Choosing the right optimizer can make a significant difference in training speed, stability, and final model accuracy. While each optimizer has its strengths, **Adam** is often the default choice for most deep learning tasks due to its adaptability and balance of speed and performance.

👉 **Pro Tip**: Always monitor performance, and don't hesitate to experiment with different optimizers based on your data and model complexity.