گزارش کار فایل HW1_Q4.ipynb

.1

```
self.fc1 = nn.Linear(784, 512)
self.fc2 = nn.Linear(512, 256)
self.fc3 = nn.Linear(256, 128)
self.fc4 = nn.Linear(128, num_classes)
```

در اینجا 4 لایه خطی تعریف کردیم

- Fc1 : ورودی با ابعاد 784 اندازه تصویر 28x28 را به512 نورون متصل می‌کند.
- Fc2 : خروجی 512 نورون را به 256 نورون کاهش می‌دهد.
- Fc3 : تعداد نورون‌ها را از256 به128 کاهش می‌دهد.
- FC4 : آخرین لایه که از 128 نورون به تعداد کلاس‌ها (num_classes) که همان تعداد خروجی است، متصل می‌شود.

.2

```
# Create our model
model = Simple_MLP(10)
```

در اینجا یک شیء از کلاس Simple_MLP با ۱۰ کلاس خروجی ساخته می‌شود. به این معنی که مدل برای طبقه‌بندی ۱۰ کلاس مختلف طراحی شده است.

```
#Create our loss function
criterion = nn.CrossEntropyLoss()
```

تابع هزینه یا criterion از نوع CrossEntropyLoss است که در مسائل طبقه‌بندی استفاده می‌شود. این تابع، تفاوت بین خروجی مدل و برچسب‌های واقعی را محاسبه می‌کند و برای به‌روزرسانی مدل استفاده می‌شود.

```
lr = 0.001
optimizer = optim.Adam(model.parameters(), lr=lr)
# Number of Epochs
n_epochs = 30
```

نرخ یادگیری (lr) برابر 0.001 تنظیم شده است.

بهینه‌ساز از نوع Adam است که یک الگوریتم بهینه‌سازی پیشرفته برای بهبود سرعت همگرایی است.

تعداد epochs برابر 30 تنظیم شده است.

```
model.train()
# Forward pass of model

outputs = model(data)

# Calculate loss
loss = criterion(outputs, target)

# Zero gradients
optimizer.zero_grad()

# Backprop loss
loss.backward()

# Optimization Step
optimizer.step()
```

مدل ترین میشود.

داده ها را به مدل می دهیم تا خروجی (outputs) به دست آید.

خطا بین خروجی مدل و برچسب‌های واقعی با استفاده از criterion محاسبه و در loss ذخیره می‌شود.

گرادیان‌های قبلی موجود در بهینه‌ساز پاک می‌شود تا از جمع شدن گرادیان‌های جدید با قبلی‌ها جلوگیری شود.

با استفاده از()backward گرادیان‌ها محاسبه می‌شوند که برای به‌روزرسانی پارامترهای مدل ضروری هستند.

در این مرحله، بهینه‌ساز با استفاده از گرادیان‌ها، پارامترهای مدل را به‌روزرسانی می‌کند.

```
model.eval()
# Forward pass of model
outputs = model(data)
```

مدل به حالت ارزیابی تغییر می‌کند که در آن برخی لایه‌ها مانند Dropout و BatchNorm به حالت ثابت در می‌آیند و تغییر نمی‌کنند.

داده ها را به مدل می دهیم و خروجی‌ها در outputs ذخیره می‌شوند.

```
loss = criterion(outputs, target)
loss_logger.append(loss.item())
_, predicted = torch.max(outputs, 1)
correct_predictions += (predicted == target).sum().item()
total_predictions += target.size(0)
```

خطا بین خروجی مدل و برچسب‌های واقعی با استفاده از criterion محاسبه و در loss_logger ذخیره می‌شود.

torch.max(outputs, 1) از خروجی‌ها بالاترین مقدار را در هر دسته می‌گیرد، که همان پیش‌بینی مدل است.

تعداد نمونه‌های درست پیش‌بینی‌شده (predicted == target) محاسبه و به correct_prediction اضافه می‌شود.

تعداد کل نمونه‌ها به total_predictions اضافه می‌شود.

```
model, optimizer, train_loss = train_epoch(model, train_loader, criterion, optimizer, train_loss)
```

در هر اپاک ، تابع train_epoch فراخوانی شده و مدل یک دور آموزش می‌بیند.
خروجی‌ها شامل مدل به‌روزشده، بهینه‌ساز و خطاهای آموزش (train_loss) هستند.

```
test_loss, acc = test_model(model, test_loader, criterion, test_loss)
test_acc.append(acc)
```

در هر ایپاک ، تابع test_model فراخوانی شده و خطای تست و دقت محاسبه می‌شوند. دقت هر اپوک به test_acc اضافه می‌شود تا دقت در طول آموزش ثبت شود.

```
plt.figure(figsize=(12, 6))
plt.plot(train_loss, label='Training Loss:', color='red')
plt.plot(test_loss, label='Test Loss:', color='blue')
plt.title('Training and Test Losses:')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

```
plt.figure(figsize=(12, 6))
plt.plot(test_acc, label='Test Accuracy', color='orange')
plt.title('Test Accuracy Over Epochs:')
plt.xlabel('Epochs:')
plt.ylabel('Accuracy (%)')
plt.ylim(0, 100)
plt.legend()
plt.grid(True)
plt.show()
```

و در آخر پلات آن را می کشیم.

محمد حقیقت - 403722042