Exercise on making use of

1. torch.nn.MSELoss function for computing the mean squared error (MSE)

and displays the loss during training:

2. Computing Gradients

In [2]:
```
import torch
import torch.optim as optim
import torch.nn as nn
import numpy as np
```

In [3]:
```
#Parameters
# Learning rate(lr-0.01)
# Number of training epochs(n_epochs=10)
# BEGIN SOLUTION
lr = 0.01
n_epochs = 10
# END SOLUTION
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

```
[NVSHARE][WARN]: Couldn't open file /var/run/secrets/kubernetes.io/serviceaccou
nt/namespace to read Pod namespace
[NVSHARE][INFO]: Successfully initialized nvshare GPU
[NVSHARE][INFO]: Client ID = 3fbe7a4e4a525dcb
```

In [5]:
```
# Input features (10 samples, 1 feature) use x_train and random.rand
# Target values (linear relation y = 2.5x + 1) for y_train
# BEGIN SOLUTION
x_train = np.random.rand(10, 1)
y_train = 2.5 * x_train + 1.0
# END SOLUTION
```

In [7]:
```
# Convert NumPy arrays to PyTorch tensors and move to the selected device
x_train_tensor = torch.from_numpy(x_train).float().to(device)
y_train_tensor = torch.from_numpy(y_train).float().to(device)
```

In [9]:
```
# Initialize parameters a (bias) and b (slope)
a = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
b = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
```

In [11]:
```
#Define the optimizer
optimizer = optim.SGD([a, b], lr=lr)
# Define MSE loss
# Use loss_fn
# BEGIn SOLUTION
loss_fn = nn.MSELoss()
# END SOLUTION

# Training loop
for epoch in range(n_epochs):
# Forward pass: compute predicted outputs
# Linear model: yhat = a + b * x
# BEGIN SOLUTION
```

```
    yhat = a + b * x_train_tensor

# END SOLUTION
# Compute the loss using MSELoss for yhat and y_train_tensor
# BEGIN SOLUTION
    loss = loss_fn(yhat, y_train_tensor)
# END SOLUTION

# Backward pass: compute gradients
# BEGIN SOLUTION

    loss.backward()

 # end SOLUTION
# Update parameters using the optimizer
    optimizer.step()
# Reset gradients to zero for the next iteration(.zero_grad())

# BEGIN SOLUTION
    optimizer.zero_grad()
 # END SOLUTION

# Print progress every 10 epochs
    if (epoch + 1) % 10 == 0:
        print(f"Epoch {epoch + 1}/{n_epochs}, Loss: {loss.item()}")
```

```
Epoch 10/10, Loss: 3.7865331172943115
```

In [13]:

```
# Print final parameters
print(f"\nTrained parameters:")
print(f"a (bias): {a.item()}")
print(f"b (slope): {b.item()}")
```

```
Trained parameters:
a (bias): 1.2857469320297241
b (slope): -0.9304391145706177
```