Date of the Session:__/__/___ **Time of the Session:___to_____**

SKILLING-7:

Implement Adam and SGD algorithms with only NumPy library and use it to implement DNN classification model

```
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset

class SGD_Numpy:
    def __init__(self, parameters, lr=0.01):
        self.lr, self.parameters = lr, parameters

    def step(self, grads):
        for p, g in zip(self.parameters, grads):
            p -= self.lr * g

class Adam_Numpy:
    def __init__(self, parameters, lr=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8):
        self.lr, self.beta1, self.beta2, self.epsilon = lr, beta1, beta2, epsilon
        self.m, self.v = [np.zeros_like(p.detach().numpy()) for p in parameters],
[np.zeros_like(p.detach().numpy()) for p in parameters]
        self.t, self.parameters = 0, parameters

    def step(self, grads):
        self.t += 1
        for i, (p, g) in enumerate(zip(self.parameters, grads)):
            self.m[i] = self.beta1 * self.m[i] + (1 - self.beta1) * g
            self.v[i] = self.beta2 * self.v[i] + (1 - self.beta2) * (g ** 2)
            m_hat, v_hat = self.m[i] / (1 - self.beta1 ** self.t), self.v[i] / (1 - self.beta2 **
self.t)
            p.data -= self.lr * m_hat / (np.sqrt(v_hat) + self.epsilon)

class DNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
```

```
        super(DNN, self).__init__()
        self.fc1, self.relu, self.fc2 = nn.Linear(input_dim, hidden_dim), nn.ReLU(),
nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        return self.fc2(self.relu(self.fc1(x)))

torch.manual_seed(42)
X, y = torch.randn(500, 10), torch.randint(0, 2, (500,))
dataset, dataloader = TensorDataset(X, y), DataLoader(TensorDataset(X, y),
batch_size=32, shuffle=True)

input_dim, hidden_dim, output_dim = 10, 16, 2
model = DNN(input_dim, hidden_dim, output_dim)  # Initialize model first
params = list(model.parameters())  # Then extract parameters
optimizer, epochs, loss_fn = Adam_Numpy(params, lr=0.01), 10, nn.CrossEntropyLoss()

for epoch in range(epochs):
    total_loss = 0
    for batch_X, batch_y in dataloader:
        logits, loss = model(batch_X), loss_fn(model(batch_X), batch_y)
        model.zero_grad()
        loss.backward()
        grads = [p.grad.detach().numpy() for p in model.parameters()]
        optimizer.step(grads)
        total_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {total_loss / len(dataloader)}")
```

Output:

Epoch 1, Loss: 0.7037773504853249
Epoch 2, Loss: 0.6819578520953655
Epoch 3, Loss: 0.6730007231235504
Epoch 4, Loss: 0.6657436639070511
Epoch 5, Loss: 0.6607115715742111
Epoch 6, Loss: 0.6507641337811947
Epoch 7, Loss: 0.649769201874733
Epoch 8, Loss: 0.6370874531567097
Epoch 9, Loss: 0.6313175074756145
Epoch 10, Loss: 0.6257837526500225

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_____out of_____ Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation: