```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import numpy as np

class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc = nn.Linear(28 * 28, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        return self.fc(x)

def generate_adversarial(model, data, target, epsilon):
    data.requires_grad = True
    loss = nn.CrossEntropyLoss()(model(data), target)
    loss.backward()
    return torch.clamp(data + epsilon * data.grad.sign(), 0, 1)

def tangent_prop_loss(model, data, tangent_vectors, target, weight):
    output = model(data)
    loss = nn.CrossEntropyLoss()(output, target)

    tangent_loss = 0
    for t in tangent_vectors:
        tangent_noise = t.expand_as(data)
        tangent_loss += ((model(data + tangent_noise) - output) ** 2).mean()

    return loss + weight * tangent_loss

def tangent_distance(x1, x2, tangents):
    return min(np.linalg.norm(x1 - x2 - t.numpy()) for t in tangents)

def tangent_classifier(x, training_data, labels, tangents):
    distances = [tangent_distance(x, train_x, tangents) for train_x in training_data]
    return labels[np.argmin(distances)]

def train(model, train_loader, optimizer, epsilon, tangents, weight):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        adv_data = generate_adversarial(model, data, target, epsilon)
        optimizer.zero_grad()
        loss = tangent_prop_loss(model, data, tangents, target, weight) + \
               nn.CrossEntropyLoss()(model(adv_data), target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print(f"Batch {batch_idx}: Loss = {loss.item()}")


transform = transforms.ToTensor()
train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleNN().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01)
tangents = [torch.randn(1, 1, 28, 28, device=device) * 0.1 for _ in range(5)]

train(model, train_loader, optimizer, epsilon=0.2, tangents=tangents, weight=0.1)
```

```
Batch 0: Loss = 6.078482627868652
Batch 100: Loss = 3.8467941284179688
Batch 200: Loss = 3.310997724533081
Batch 300: Loss = 2.724601984024048
Batch 400: Loss = 2.688321590423584
Batch 500: Loss = 2.602125883102417
Batch 600: Loss = 2.771756649017334
Batch 700: Loss = 2.552194356918335
Batch 800: Loss = 2.20986270904541
Batch 900: Loss = 2.582984685897827
```