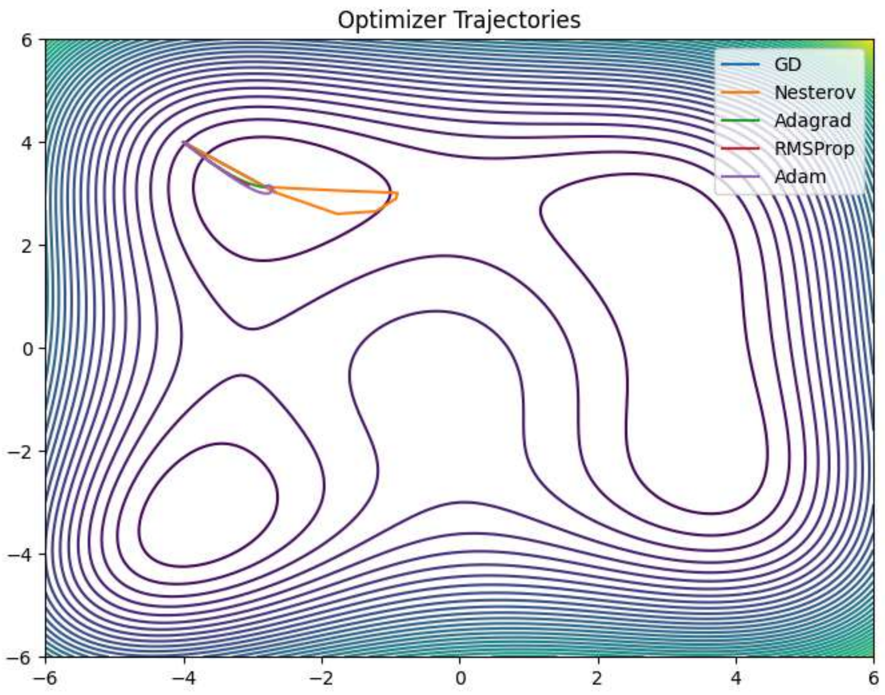
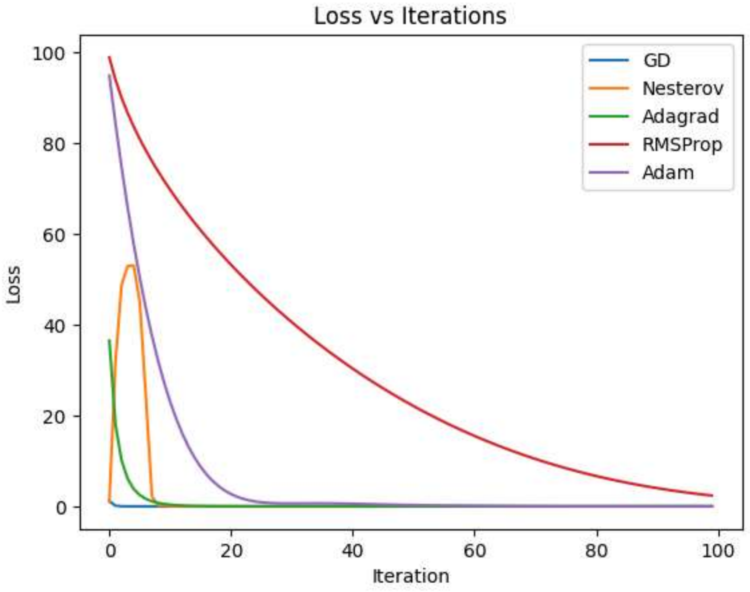
```
pip install numpy matplotlib tensorflow scikit-learn
```

Locate in Drive
Open in playground mode
New notebook in Drive
Open notebook Ctrl+O
Upload notebook
Rename
Move
Move to the bin
Save a copy in Drive
Save a copy as a GitHub Gist
Save a copy in GitHub
Save Ctrl+S
Save and pin revision Ctrl+M S
Revision history
Notebook info
Download ▸
Print Ctrl+P

```
 /usr/local/lib/python3.12/dist-packages (2.0.2)
ib in /usr/local/lib/python3.12/dist-packages (3.10.0)
ow in /usr/local/lib/python3.12/dist-packages (2.19.0)
earn in /usr/local/lib/python3.12/dist-packages (1.6.1)
y>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
s>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.61.1)
er>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
g>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (26.0)
8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
g>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.3.2)
lateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post
=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
se>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
ers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.12.19)
5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorfl
asta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
um>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/loca
<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.32.4)
ls in /usr/local/lib/python3.12/dist-packages (from tensorflow) (75.2.0)
2.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.0)
r>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.3.0)
xtensions>=3.6.6 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (4.15.0
.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.1.1)
.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.76.0)
ard~=2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.19.0)
.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0)
11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.15.1)
s<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.4)
.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.3)
olctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0->tensor
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.18.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorf
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensor
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorf
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1->tensorboa
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->te
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich->kera
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation

# Himmelblau Function
def loss_fn(w):
    w1, w2 = w
    return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2

def grad_fn(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])

# Optimizers
def gradient_descent(init, lr=0.01, steps=100):
    w = init.copy()
    path = [w.copy()]
    loss = []
    for _ in range(steps):
        w -= lr * grad_fn(w)
        path.append(w.copy())
        loss.append(loss_fn(w))
    return np.array(path), loss

def nesterov(init, lr=0.01, gamma=0.9, steps=100):
    w = init.copy()
    v = np.zeros_like(w)
    path = [w.copy()]
    loss = []
    for _ in range(steps):
        lookahead = w - gamma*v
        g = grad_fn(lookahead)
        v = gamma*v + lr*g
        w -= v
        path.append(w.copy())
        loss.append(loss_fn(w))
    return np.array(path), loss

def adagrad(init, lr=0.4, steps=100):
    w = init.copy()
    G = np.zeros_like(w)
    eps = 1e-8
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        g = grad_fn(w)
        G += g**2
        w -= lr*g/(np.sqrt(G)+eps)
        path.append(w.copy())
        loss.append(loss_fn(w))
    return np.array(path), loss

def rmsprop(init, lr=0.01, beta=0.9, steps=100):
    w = init.copy()
    Eg = np.zeros_like(w)
    eps=1e-8
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        g=grad_fn(w)
        Eg = beta*Eg + (1-beta)*(g**2)
        w -= lr*g/(np.sqrt(Eg)+eps)
        path.append(w.copy())
        loss.append(loss_fn(w))
    return np.array(path), loss

def adam(init, lr=0.05, steps=100):
    w = init.copy()
    m = np.zeros_like(w)
    v = np.zeros_like(w)
    b1,b2=0.9,0.999
    eps=1e-8
    path=[w.copy()]
    loss=[]
    for t in range(1,steps+1):
        g=grad_fn(w)
        m=b1*m+(1-b1)*g
        v=b2*v+(1-b2)*(g**2)
        mhat=m/(1-b1**t)
        vhat=v/(1-b2**t)
        w-=lr*mhat/(np.sqrt(vhat)+eps)
        path.append(w.copy())
        loss.append(loss_fn(w))
    return np.array(path), loss

# Run optimizers
init = np.array([-4.0, 4.0])

paths = {}
losses = {}

for name, opt in {
    "GD": gradient_descent,
    "Nesterov": nesterov,
    "Adagrad": adagrad,
    "RMSProp": rmsprop,
    "Adam": adam
}.items():
    paths[name], losses[name] = opt(init)

# Create contour grid
x = np.linspace(-6,6,200)
y = np.linspace(-6,6,200)
X,Y = np.meshgrid(x,y)
Z = loss_fn([X,Y])

# Plot contour with trajectories
plt.figure(figsize=(8,6))
plt.contour(X,Y,Z,50)
for name,path in paths.items():
    plt.plot(path[:,0], path[:,1], label=name)
plt.legend()
plt.title("Optimizer Trajectories")
plt.show()

# Loss vs Iterations
plt.figure()
for name,l in losses.items():
    plt.plot(l,label=name)
plt.legend()
plt.title("Loss vs Iterations")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.show()
```





```
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import StandardScaler

# Load MNIST
(x_train, y_train), _ = tf.keras.datasets.mnist.load_data()

# Select digits 0 and 1
mask = (y_train==0) | (y_train==1)
x = x_train[mask]
y = y_train[mask]

# Flatten images
```

```
x = x.reshape(len(x), -1)

# Reduce to 2 features using mean of halves
f1 = x[:, :392].mean(axis=1)
f2 = x[:, 392:].mean(axis=1)

X = np.vstack((f1,f2)).T

# normalize
X = StandardScaler().fit_transform(X)

y = y.astype(np.float32)
```

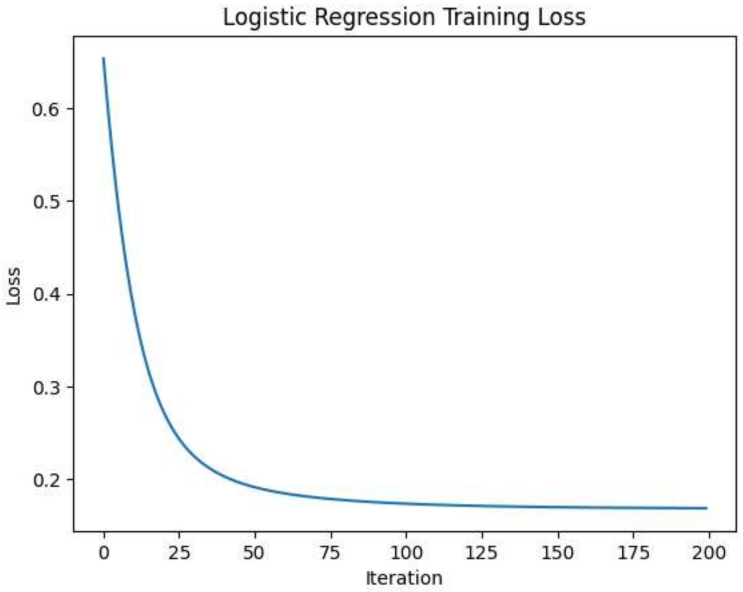
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
def sigmoid(z):
    return 1/(1+np.exp(-z))

def loss(w,X,y):
    z = X.dot(w)
    p = sigmoid(z)
    return -np.mean(y*np.log(p+1e-8)+(1-y)*np.log(1-p+1e-8))

def grad(w,X,y):
    p = sigmoid(X.dot(w))
    return X.T.dot(p-y)/len(y)
```

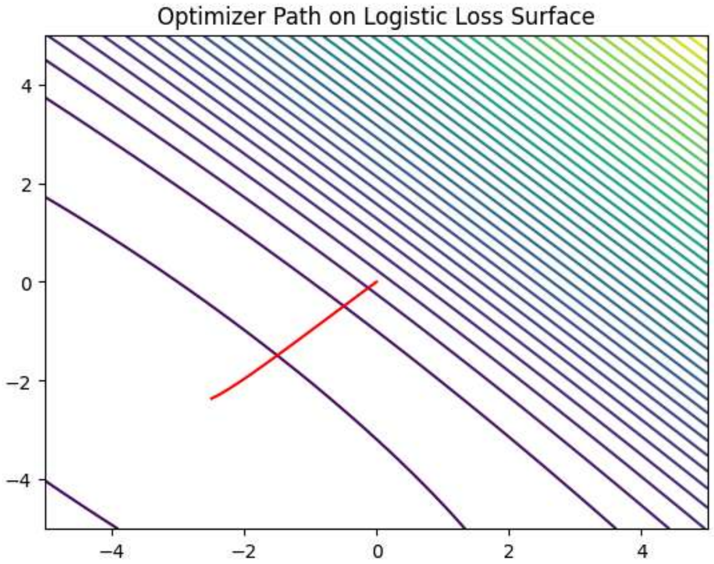
```
def train_adam(X,y,lr=0.05,steps=200):
    w=np.zeros(2)
    m=np.zeros(2)
    v=np.zeros(2)
    b1,b2=0.9,0.999
    losses=[]
    path=[w.copy()]

    for t in range(1,steps+1):
        g=grad(w,X,y)
        m=b1*m+(1-b1)*g
        v=b2*v+(1-b2)*(g**2)
        mhat=m/(1-b1**t)
        vhat=v/(1-b2**t)
        w-=lr*mhat/(np.sqrt(vhat)+1e-8)
        losses.append(loss(w,X,y))
        path.append(w.copy())
    return np.array(path),losses

path,losses=train_adam(X,y)

print("Final weights:",path[-1])
```

```
Final weights: [-2.48805623 -2.3603943 ]
```

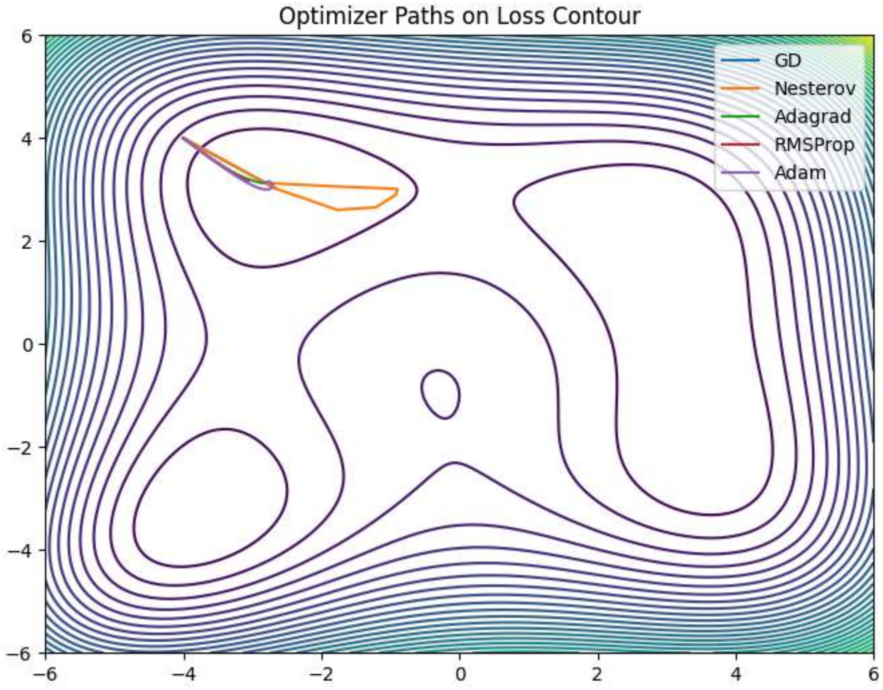
```
import matplotlib.pyplot as plt
plt.plot(losses)
plt.title("Logistic Regression Training Loss")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.show()
```

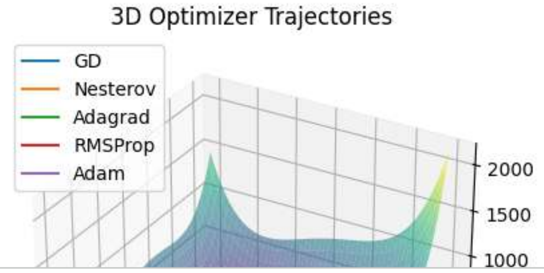


```
w1 = np.linspace(-5,5,100)
w2 = np.linspace(-5,5,100)
W1,W2 = np.meshgrid(w1,w2)
Z = np.zeros_like(W1)

for i in range(W1.shape[0]):
    for j in range(W1.shape[1]):
        Z[i,j]=loss(np.array([W1[i,j],W2[i,j]]),X,y)

plt.contour(W1,W2,Z,50)
plt.plot(path[:,0],path[:,1],'r')
plt.title("Optimizer Path on Logistic Loss Surface")
plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
def J(w):
    w1, w2 = w
    return (w1**2 + w2 - 11)**2 + (w1 + w2**2 - 7)**2
def grad_J(w):
    w1, w2 = w
    dw1 = 4*w1*(w1**2 + w2 - 11) + 2*(w1 + w2**2 - 7)
    dw2 = 2*(w1**2 + w2 - 11) + 4*w2*(w1 + w2**2 - 7)
    return np.array([dw1, dw2])
def GD(init, lr=0.01, steps=100):
    w = init.copy()
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        w -= lr*grad_J(w)
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
def Nesterov(init, lr=0.01, gamma=0.9, steps=100):
    w = init.copy()
    v = np.zeros_like(w)
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        lookahead = w - gamma*v
        g = grad_J(lookahead)
        v = gamma*v + lr*g
        w -= v
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
def Adagrad(init, lr=0.4, steps=100):
    w = init.copy()
    G = np.zeros_like(w)
    eps=1e-8
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        g=grad_J(w)
        G += g**2
        w -= lr*g/(np.sqrt(G)+eps)
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
def RMSProp(init, lr=0.01, beta=0.9, steps=100):
    w=init.copy()
    Eg=np.zeros_like(w)
    eps=1e-8
    path=[w.copy()]
    loss=[]
    for _ in range(steps):
        g=grad_J(w)
        Eg=beta*Eg+(1-beta)*(g**2)
        w -= lr*g/(np.sqrt(Eg)+eps)
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
def Adam(init, lr=0.05, steps=100):
    w=init.copy()
    m=np.zeros_like(w)
    v=np.zeros_like(w)
    b1,b2=0.9,0.999
    eps=1e-8
    path=[w.copy()]
    loss=[]
    for t in range(1,steps+1):
        g=grad_J(w)
        m=b1*m+(1-b1)*g
        v=b2*v+(1-b2)*(g**2)
        mhat=m/(1-b1**t)
        vhat=v/(1-b2**t)
        w -= lr*mhat/(np.sqrt(vhat)+eps)
        path.append(w.copy())
        loss.append(J(w))
    return np.array(path), loss
init = np.array([-4.0, 4.0])
optimizers = {
    "GD": GD,
    "Nesterov": Nesterov,
    "Adagrad": Adagrad,
    "RMSProp": RMSProp,
    "Adam": Adam
}
paths={}
losses={}
for name,opt in optimizers.items():
    paths[name], losses[name] = opt(init)
x=np.linspace(-6,6,200)
y=np.linspace(-6,6,200)
X,Y=np.meshgrid(x,y)
Z=J([X,Y])
plt.figure(figsize=(8,6))
plt.contour(X,Y,Z,40)
for name,p in paths.items():
    plt.plot(p[:,0],p[:,1],label=name)
plt.legend()
plt.title("Optimizer Paths on Loss Contour")
plt.show()
#3D SURFACE
fig=plt.figure()
ax=fig.add_subplot(111,projection='3d')
ax.plot_surface(X,Y,Z,cmap=cm.viridis,alpha=0.6)

for name,p in paths.items():
    z=[J(w) for w in p]
    ax.plot(p[:,0],p[:,1],z,label=name)

ax.set_title("3D Optimizer Trajectories")
plt.legend()
plt.show()
#LOSS CURVES
plt.figure()
for name,l in losses.items():
    plt.plot(l,label=name)
plt.legend()
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.title("Loss vs Iterations")
plt.show()
#Load MNIST
(x_train, y_train), _ = tf.keras.datasets.mnist.load_data()
mask=(y_train==0)|(y_train==1)
x=x_train[mask]
y=y_train[mask]
x=x.reshape(len(x),-1)
f1=x[:,:392].mean(axis=1)
f2=x[:,392:].mean(axis=1)
X=np.vstack((f1,f2)).T
X=StandardScaler().fit_transform(X)
y=y.astype(np.float32)
def sigmoid(z):
    return 1/(1+np.exp(-z))
def loss_lr(w):
    p=sigmoid(X.dot(w))
    return -np.mean(y*np.log(p+1e-8)+(1-y)*np.log(1-p+1e-8))
def grad_lr(w):
    p=sigmoid(X.dot(w))
    return X.T.dot(p-y)/len(y)
def train_adam_lr(steps=200, lr=0.05):
    w=np.zeros(2)
    m=np.zeros(2)
    v=np.zeros(2)
    b1,b2=0.9,0.999
    eps=1e-8
    path=[w.copy()]
    losses=[]
    for t in range(1,steps+1):
        g=grad_lr(w)
        m=b1*m+(1-b1)*g
        v=b2*v+(1-b2)*(g**2)
        mhat=m/(1-b1**t)
        vhat=v/(1-b2**t)
        w -= lr*mhat/(np.sqrt(vhat)+eps)
        path.append(w.copy())
        losses.append(loss_lr(w))
    return np.array(path),losses
path_lr,loss_lr_vals=train_adam_lr()
#loss curve
plt.plot(loss_lr_vals)
plt.title("Logistic Regression Loss")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()
w1=np.linspace(-5,5,100)
w2=np.linspace(-5,5,100)
W1,W2=np.meshgrid(w1,w2)
Z=np.zeros_like(W1)

for i in range(W1.shape[0]):
    for j in range(W1.shape[1]):
        Z[i,j]=loss_lr(np.array([W1[i,j],W2[i,j]]))

plt.contour(W1,W2,Z,40)
plt.plot(path_lr[:,0],path_lr[:,1],'r')
plt.title("Optimizer Path on Logistic Loss Surface")
plt.show()
```





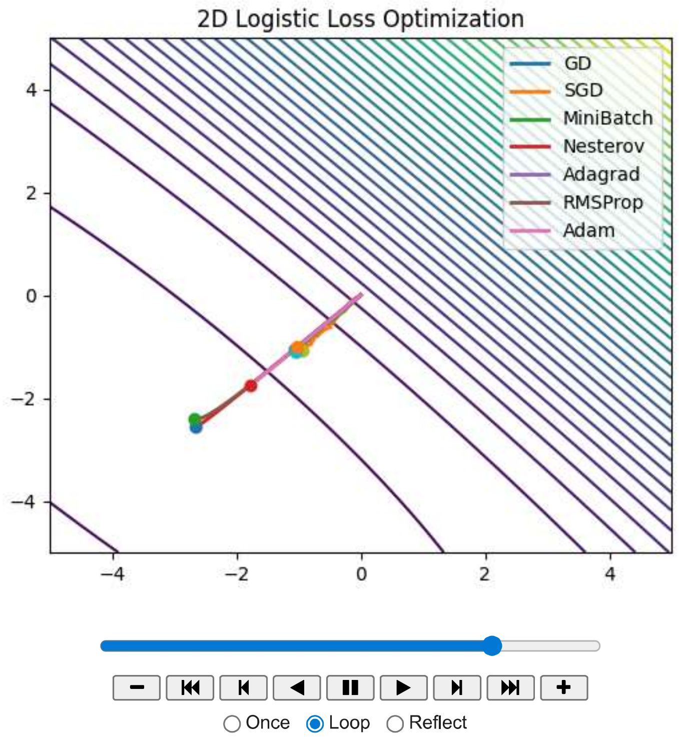
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, cm
from IPython.display import HTML
import tensorflow as tf
from sklearn.preprocessing import StandardScaler

# Load MNIST
(x_train, y_train), _ = tf.keras.datasets.mnist.load_data()

# binary classification (0 vs 1)
mask = (y_train==0) | (y_train==1)
x = x_train[mask]
y = y_train[mask]

# flatten images
x = x.reshape(len(x), -1)

# create TWO features
f1 = x[:,:392].mean(axis=1)
f2 = x[:,392:].mean(axis=1)
X = np.vstack((f1,f2)).T

# normalize
X = StandardScaler().fit_transform(X)
y = y.astype(np.float32)

# logistic functions
def sigmoid(z):
    return 1/(1+np.exp(-z))

def loss_lr(w):
    p = sigmoid(X.dot(w))
    return -np.mean(y*np.log(p+1e-8)+(1-y)*np.log(1-p+1e-8))

def grad_lr(w):
    p = sigmoid(X.dot(w))
    return X.T.dot(p-y)/len(y)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

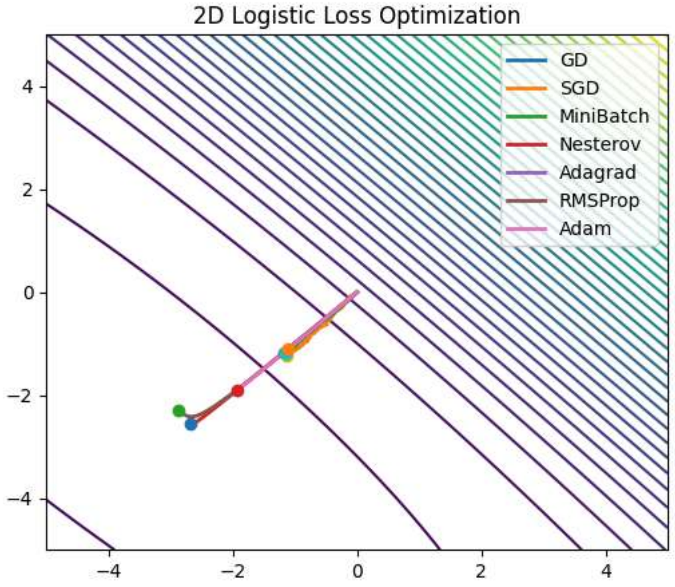
```
steps = 80
lr = 0.1
init = np.zeros(2)

paths = {}

# GD
w=init.copy(); path=[w.copy()]
for _ in range(steps):
    w -= lr*grad_lr(w)
    path.append(w.copy())
paths["GD"]=np.array(path)

# SGD
w=init.copy(); path=[w.copy()]
for _ in range(steps):
    noise=np.random.randn(2)*0.1
    w -= lr*(grad_lr(w)+noise)
    path.append(w.copy())
paths["SGD"]=np.array(path)

# MiniBatch
w=init.copy(); path=[w.copy()]
for _ in range(steps):
    noise=np.random.randn(2)*0.05
    w -= lr*(grad_lr(w)+noise)
    path.append(w.copy())
paths["MiniBatch"]=np.array(path)

# Nesterov
w=init.copy(); v=np.zeros(2); path=[w.copy()]
for _ in range(steps):
    look = w - 0.9*v
    g = grad_lr(look)
    v = 0.9*v + lr*g
    w -= v
    path.append(w.copy())
paths["Nesterov"]=np.array(path)

# Adagrad
w=init.copy(); G=np.zeros(2); path=[w.copy()]
for _ in range(steps):
    g=grad_lr(w)
    G+=g**2
    w-=lr*g/(np.sqrt(G)+1e-8)
    path.append(w.copy())
paths["Adagrad"]=np.array(path)

# RMSProp
w=init.copy(); Eg=np.zeros(2); path=[w.copy()]
for _ in range(steps):
    g=grad_lr(w)
    Eg=0.9*Eg+0.1*(g**2)
    w-=lr*g/(np.sqrt(Eg)+1e-8)
    path.append(w.copy())
paths["RMSProp"]=np.array(path)

# Adam
w=init.copy(); m=v=np.zeros(2); path=[w.copy()]
for t in range(1,steps+1):
    g=grad_lr(w)
    m=0.9*m+0.1*g
    v=0.999*v+0.001*(g**2)
    mhat=m/(1-0.9**t)
    vhat=v/(1-0.999**t)
    w-=0.05*mhat/(np.sqrt(vhat)+1e-8)
    path.append(w.copy())
paths["Adam"]=np.array(path)
```

```
# create loss surface grid
w1 = np.linspace(-5,5,120)
w2 = np.linspace(-5,5,120)
W1,W2 = np.meshgrid(w1,w2)
Z = np.zeros_like(W1)

for i in range(W1.shape[0]):
    for j in range(W1.shape[1]):
        Z[i,j] = loss_lr(np.array([W1[i,j],W2[i,j]]))

fig, ax = plt.subplots(figsize=(6,5))
```

```
ax.contour(W1,W2,Z,40)

lines = {n: ax.plot([],[],lw=2,label=n)[0] for n in paths}
points = {n: ax.plot([],[],'o')[0] for n in paths}

ax.legend()
ax.set_title("2D Logistic Loss Optimization")

def animate(i):
    for name,p in paths.items():
        if i < len(p):
            lines[name].set_data(p[:i,0], p[:i,1])
            points[name].set_data([p[i-1,0]], [p[i-1,1]])
    return list(lines.values()) + list(points.values())

ani = animation.FuncAnimation(fig, animate, frames=steps, interval=120, blit=True)

HTML(ani.to_jshtml())
```



○ Once ● Loop ○ Reflect



```
fig = plt.figure(figsize=(7,6))
ax = fig.add_subplot(111, projection='3d')

def animate3d(i):
    ax.clear()
    ax.plot_surface(W1,W2,Z,cmap=cm.viridis,alpha=0.6)
```