Colab: https://colab.research.google.com/drive/1wygxZrN0zK0oeW1BBoYqaPO5l_UbvQyE?usp=sharing

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

!gdown 1dLOPwh01o3k8p_hK633ixhD1ehz6nNWk
df = pd.read_csv("/content/spiral.csv")
df.head()
```

```
Downloading...
From: https://drive.google.com/uc?id=1dLOPwh01o3k8p_hK633ixhD1ehz6nNWk
To: /content/spiral.csv
100% 12.9k/12.9k [00:00<00:00, 10.5MB/s]
```

	x1	x2	y
0	0.000000	0.000000	0
1	-0.000650	0.010080	0
2	0.009809	0.017661	0
3	0.007487	0.029364	0
4	-0.000027	0.040404	0

```
df.shape
```

```
(300, 3)
```

```
X = df.iloc[:, :-1].to_numpy()
y = df.iloc[:, -1].to_numpy()
```

```
X.shape
```

```
(300, 2)
```

```
y.shape
```

```
(300,)
```

```
d = X.shape[1] # no of features
n = len(np.unique(y)) # classes
m = len(X)
print(d, n, m)
```

```
2 3 300
```

```
W = 0.01*np.random.randn(d,n) # initialise it with random numbers and not zeros - w
```

```
b = np.zeros((1,n))
```

```
W
```

```
    array([[-0.00076015, -0.01205424, -0.01809676],
           [ 0.0270647 ,  0.01662531, -0.00823911]])
```

```
b
```

```
    array([[0., 0., 0.]])
```

```
# step1 - linear
z = np.dot(X, W) + b
print(z.shape)
```

```
    (300, 3)
```

```
# step2 - activation
exp_z = np.exp(z)
probs = exp_z / np.sum(exp_z, axis=1, keepdims=True)
probs.shape
```

```
    (300, 3)
```

```
def loss(y, probs):
  m = y.shape[0]
  error = -np.log(probs[range(m), y]) # prob of the true class, probs[:, y], why no
  return np.sum(error)/m
```

```
loss(y, probs)
```

```
    1.0963809543569694
```

Backprop

```
# dw = dz.X = X.T * dZ
dZ = probs
dZ[range(m), y] -= 1 # subtracting 1 only for the probs corresponding the true
```

```
dW = np.dot(X.T, dZ)#/m
print(dW.shape)
```

```
    (2, 3)
```

```
db = np.sum(dZ, axis=0, keepdims=True)#/m
db.shape
```

```
    (1, 3)
```

```
def backprop(probs, y):
    # we know that dz = pi - I
    dz = probs # dz = pi
    dz[range(m),y] -= 1 # subtacting 1 where i ==j i.e. class label matches
    dz = dz/m # taking average as we have m points
    dW = np.dot(X.T, dz)
    db = np.sum(dz, axis=0, keepdims=True)
    return dW, db
```

```
lr = 0.1
W += -lr * dW
b += -lr * db
```

```
max_iters = 500
lr = 1

d = X.shape[1]
n = len(np.unique(y))
m = X.shape[0]
W = 0.01 * np.random.randn(d,n)
b = np.zeros((1,n))
loss_history = []

for i in range(max_iters):
    # evaluate the class probs
    z = np.dot(X, W) + b
    exp_z = np.exp(z)
    probs = exp_z/np.sum(exp_z, axis=1, keepdims=True)

    # compute the loss: average cross-entropy loss and regularization
    error = -np.log(probs[range(m), y])
    loss = np.sum(error)/m
    loss_history.append(loss)
    if i % 100 == 0:
        print(f"iteration: {i}, loss: {loss}")



    # compute the gradient on score
    dZ = probs
    dZ[range(m),y] -= 1
    dZ = dZ/m
    dW = np.dot(X.T, dZ)
    db = np.sum(dZ, axis=0, keepdims=True)

    # perform a parameter update using gradient descent
    W += -lr * dW
    b += -lr * db
# history = pd.DataFrame({'step': list(range(max_iters)), 'loss': loss_history})
# history.plot(x='step', y='loss',xlabel='step', ylabel='loss')
```

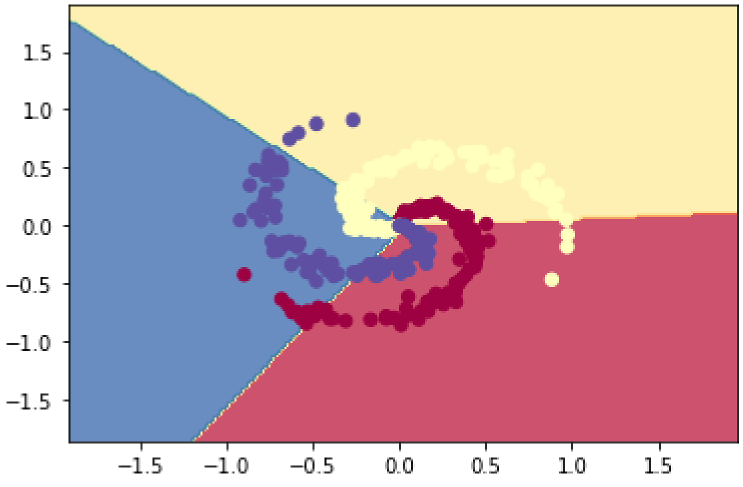
```
iteration: 0, loss: 1.0979803316219083
iteration: 100, loss: 0.7447245946654641
```

```
    iteration: 200, loss: 0.7405321973783513
    iteration: 300, loss: 0.7402161812169866
    iteration: 400, loss: 0.7401859903611652
```

```
def predict(X):
    Z = np.dot(X, W) + b
    Z_e = np.exp(Z)
    probs = Z_e/np.sum(Z_e, axis=1, keepdims=True)
    return np.argmax(probs, axis=1)

print(f"Training Accuracy {np.sum(predict(X) == y)/m*100}")
```

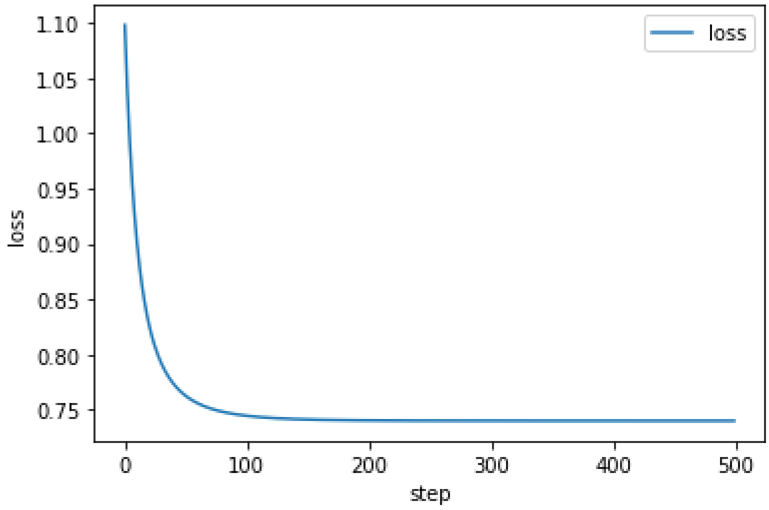
```
    Training Accuracy 52.666666666666664
```

```
# create a 2D grid
step = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, step), np.arange(y_min, y_max, step))

# predict for all the points in the grid
y_hat = predict(np.c_[xx.ravel(), yy.ravel()]) # concatenates along second axis
y_hat = y_hat.reshape(xx.shape)

# plot
fig = plt.figure()
plt.contourf(xx, yy, y_hat, cmap=plt.cm.Spectral, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```



```
class SoftmaxClassfier:
    def __init__(self, n_features, n_outputs):
        self.d = n_features
        self.n = n_outputs
        self.W = 0.01 * np.random.randn(d,n)
        self.b = np.zeros((1,n))
        self.loss = []
    def fwdprop(self, X):
        z = np.dot(X, self.W) + self.b
        exp_z = np.exp(z)
        probs = exp_z / np.sum(exp_z, axis=1, keepdims=True)
        return probs

    def cce_loss(self, probs, y):
        m = y.shape[0]
        error = -np.log(probs[range(m), y])
        return np.sum(error)/m

    def backprop(self, probs, y):
        m = y.shape[0]
        dz = probs
        dz[range(m),y] -= 1
        dz = dz/m
        dW = np.dot(X.T, dz)
        db = np.sum(dz, axis=0, keepdims=True)
        return dW, db

    def fit(self, X, y, lr=0.1, max_iters=50):

        for i in range(max_iters):
            # evaluate the class probs
            probs = self.fwdprop(X)

            # compute the loss: average cross-entropy loss and regularization
            loss = self.cce_loss(probs, y)

            # compute the gradient on score
            dW, db = self.backprop(probs, y)

            # perform a parameter update using gradient descent
            self.W += -lr * dW
            self.b += -lr * db
            self.loss.append(loss)

        self.history = pd.DataFrame({
        'step': list(range(max_iters)),
        'loss': self.loss})

    def plot_loss(self):
        return self.history.plot(x='step', y='loss',xlabel='step', ylabel='loss')

    def predict(self, X):
        probs = self.fwdprop(X)
        return np.argmax(probs, axis=1)

model = SoftmaxClassfier(n_features=2, n_outputs=3)
model.fit(X, y, lr=1, max_iters=500)
model.plot_loss()
print('training accuracy:', np.sum(model.predict(X) == y)/X.shape[0])
```

```
training accuracy: 0.5266666666666666
```



✓ 0s completed at 23:08