Untitled

June 5, 2020

In [ ]:
```
import numpy as np
get_ipython().magic('matplotlib inline')
import matplotlib.pyplot as plt
```

In [ ]:
```
import sys
sys.path.appened("..")
import grading
grading=grading.Grader(assignment_key="fpGDPl9d0HiwKogh-hZg",all_parts=["xU7U4", "HyTF6"
```

In [ ]:
```
COURSERA_TOKEN="fpGDPl9d0HiwKogh"
COURSERA_EMAIL="kotagiritharun000@gmail.com"
```

In [ ]:
```
with open('train.npy','rb')as fin:
    X=np.load(fin)
with open('target.npy','rb')as fin:
    y=np.load(fin)
    plt.scatter(X[:,0],X[:,1],c=y,cmap=plt.cm.Paired,s=20)
    plt.show()
```

In [ ]:
```
print(X.shape)
print(y.shape)
```

In [ ]:
```
def expand(X)
X_expanded[:,0],X_expanded[:,1]=X[:,0],X[:,1]
X_expanded[:,2],X_expanded[:,3]=X[:,0]**2,X[:,1]**2
X_expanded[:,4],X_expanded[:,5]=X[:,0]*X[:,1],np.ones(X.shape[0])
return X_expanded
```

In [ ]:
```
X_expanded = expand(X)
```

In [ ]:
```
dummy_X = np.array([
        [0,0],
        [1,0],
        [2.61,-1.28],
        [-0.59,2.1]
    ])

dummy_expanded = expand(dummy_X)
dummy_expanded_ans = np.array([[ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  1.    ],
```

```
                                        [ 1.    ,  0.    ,  1.    ,  0.    ,  0.    ,  1.    ],
                                        [ 2.61  , -1.28  ,  6.8121,  1.6384, -3.3408,  1.    ],
                                        [-0.59  ,  2.1   ,  0.3481,  4.41  , -1.239 ,  1.    ]])
        assert isinstance(dummy_expanded,np.ndarray), "please make sure you return numpy array"
        assert dummy_expanded.shape == dummy_expanded_ans.shape, "please make sure your shape is
        assert np.allclose(dummy_expanded,dummy_expanded_ans,1e-3), "Something's out of order wi
        print("Seems legit!")
```

In [ ]:
```
def probability(X, w):
    z = np.dot(X,w)
    a = 1./(1+np.exp(-z))
    return np.array(a)
```

In [ ]:
```
dummy_weights = np.linspace(-1, 1, 6)
ans_part1 = probability(X_expanded[:1, :], dummy_weights)[0]
```

In [ ]:
```
print(ans_part1)
```

In [ ]:
```
grader.set_answer("xU7U4", ans_part1)
```

In [ ]:
```
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

In [ ]:
```
def compute_loss(X, y, w):
    l = X.shape[0]
    a = probability(X, w)
    cross_entropy = y*np.log(a) +(1-y)*np.log(1-a)
    cost = -np.sum(cross_entropy)/float(l)
     cost = np.squeeze(cost)
    assert(cost.shape == ())
     return cost
```

In [ ]:
```
ans_part2 = compute_loss(X_expanded, y, dummy_weights)
```

In [ ]:
```
print(ans_part2)
```

In [ ]:
```
grader.set_answer("HyTF6", ans_part2)
```

In [ ]:
```
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

In [ ]:
```
def compute_grad(X, y, w):
    m = X.shape[0]
    A = probability(X, w)
    dZ = A - y
    dW = np.dot(dZ, X) / float(m)
    return dW
```

In [ ]:
```
ans_part3 = np.linalg.norm(compute_grad(X_expanded, y, dummy_weights))
```

In [ ]:
```
print(ans_part3)
```

In [ ]:
```
grader.set_answer("uNidL", ans_part3)
```

In [ ]:
```
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

In [ ]:
```
from IPython import display
h = 0.01
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, h), np.arange(y_min, y_max, h))
def visualize(X, y, w, history):
    Z = probability(expand(np.c_[xx.ravel(), yy.ravel()]), w)
    Z = Z.reshape(xx.shape)
    plt.subplot(1, 2, 1)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.subplot(1, 2, 2)
    plt.plot(history)
    plt.grid()
    ymin, ymax = plt.ylim()
    plt.ylim(0, ymax)
    display.clear_output(wait=True)
    plt.show()
```

In [ ]:
```
visualize(X, y, dummy_weights, [0.5, 0.5, 0.25])
```

In [ ]:
```
np.random.seed(42)
w = np.array([0, 0, 0, 0, 0, 1])
eta= 0.1
n_iter = 100
batch_size = 4
loss = np.zeros(n_iter)
plt.figure(figsize=(12, 5))
for i in range(n_iter):
    ind = np.random.choice(X_expanded.shape[0], batch_size)
    loss[i] = compute_loss(X_expanded, y, w)
    if i % 10 == 0:
        visualize(X_expanded[ind, :], y[ind], w, loss)

    dW = compute_grad(X_expanded[ind, :], y[ind], w)
    w = w - eta * dW
visualize(X, y, w, loss)
plt.clf()
```

In [ ]:
```
ans_part4 = compute_loss(X_expanded, y, w)
```

In [ ]:
```
grader.set_answer("ToK7N", ans_part4)
```

In [ ]:
```
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

In [ ]:
```
np.random.seed(42)
w = np.array([0, 0, 0, 0, 0, 1])
eta = 0.05
alpha = 0.9
nu = np.zeros_like(w)
n_iter = 100
batch_size = 4
loss = np.zeros(n_iter)
plt.figure(figsize=(12, 5))
for i in range(n_iter):
    ind = np.random.choice(X_expanded.shape[0], batch_size)
    loss[i] = compute_loss(X_expanded, y, w)
    if i % 10 == 0:
    visualize(X_expanded[ind, :], y[ind], w, loss)
    dW = compute_grad(X_expanded[ind, :], y[ind], w)
    nu = alpha*nu+eta*dW
    w = w - nu
visualize(X, y, w, loss)
plt.clf()
```

In [ ]:
```
ans_part5 = compute_loss(X_expanded, y, w)
```

In [ ]:
```
grader.set_answer("GBdgZ", ans_part5)
```

In [ ]:
```
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

In [ ]:
```
np.random.seed(42)
w = np.array([0, 0, 0, 0, 0, 1.])
eta = 0.1
alpha = 0.9
G = np.zeros_like(w)
g2 = np.zeros_like(w)
eps = 1e-8
n_iter = 100
batch_size = 4
loss = np.zeros(n_iter)
plt.figure(figsize=(12,5))
for i in range(n_iter):
    ind = np.random.choice(X_expanded.shape[0], batch_size)
    loss[i] = compute_loss(X_expanded, y, w)
    if i % 10 == 0:
        visualize(X_expanded[ind, :], y[ind], w, loss)
    dW = compute_grad(X_expanded[ind, :], y[ind], w)
    g2 = dW**2
    G = alpha*G+(1-alpha)*g2
    w = w - eta*dW/np.sqrt(G+eps)
    visualize(X, y, w, loss)
    plt.clf()
```

In [ ]:
```
ans_part6 = compute_loss(X_expanded, y, w)
```

In [ ]:
```
grader.set_answer("dLdHG", ans_part6)
```

In [ ]:
```
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```