perceptron

February 7, 2025

[147]:
```
import numpy as np
```

Problem 7

a)

[148]:
```
# Part a

n = 25
cluster1 = np.random.randn(n,3) #0 centered

cluster1 = np.hstack([cluster1, np.full((n,1),1)])

cluster2 = np.random.randn(n,3) + np.array([4,4,4])

cluster2 = np.hstack([cluster2, np.full((n,1),-1)])


data = np.vstack((cluster1,cluster2))
# data.shape
x = data[:,:-1] # Main data without labels
y = data[:,-1] # Labels
```

[149]:
```
#Part b

# x = data[:,:-1]

norms = np.linalg.norm(x,axis=1)
beta2_i = np.argmax(norms)
beta2 = np.max(norms)

print(f"Beta^2 index: {beta2_i}\nBeta^2 val: {round(beta2,2)}")
```

```
Beta^2 index: 45
Beta^2 val: 9.18
```

[150]:
```
# Defining Class
```

```python
class Perceptron:
    def __init__(self,w,bias=True,constraint=False):
        self.bias = bias
        self.constraint = constraint
        if bias:
            self.weights = np.random.randn(w+1)
        else:
            self.weights = np.random.randn(w)

        if self.constraint:
            self.weights /= np.linalg.norm(self.weights)
            self.weights *= 0.1
            self.w0 = self.weights


    def fit_sto(self,data,labels,ret=False,verb=True):
        steps = 0
        correct = False
        if self.bias:
            data = add_bias(data)

        while not correct:
            incorrect = 0
            for x,y in zip(data,labels):
                score = x @ self.weights * y
                # print(x @ self.weights)
                if score <= 0:
                    self.weights += (y*x)
                    steps += 1
                    incorrect +=1
                    # show(self.weights)
                    # sleep(0.5)
            if incorrect == 0:
                break
        if verb:
            print(f"{steps} updates made")
        if ret:
            return steps

    def fit_batch(self,data,labels,lr = 0.3, theta = 0.15, epochs = 5000):
        steps = 0
        if self.bias:
            data = add_bias(data)

        while steps < epochs:
            misclassified = (labels * (self.weights @ data.T) <= 0) # Called
 ↪map because it's used to index. Gives indices of all incorrect
```

```
            # print(misclassified.shape)
            if not np.any(misclassified): # If there are no incorrect
                break

            update = lr * (labels[None,misclassified,] * data[misclassified].T )

            self.weights += update.sum(axis=1)

            steps += 1
        print("Updates:", steps)



    def pred(self,data):
        # print(self.weights.shape,data.shape)
        if self.bias:
            data = add_bias(data)

        ypred = np.sign(self.weights @ data.T)
        return ypred


def test_init(weights,dim=3):
    "Testing different initialization strategies"
    p = Perceptron(dim)
    p.weights = weights
    return p.fit_sto(x,y,ret=True,verb=False) #Returns num of steps for␣
 ↪convergence




def add_bias(matrix):
    matrix = np.hstack([matrix, np.full((matrix.shape[0],1),1)])
    return matrix# print(ypred,"\n",y)
```

[151]:
```
#Part C
#It converged relatively quickly.
qr = Perceptron(3)

qr.fit_batch(x,y)

ypred= qr.pred(x)

percentage = (np.mean((ypred * y) == 1)) * 100
print(percentage)
# print(qr.weights)
```

Updates: 48
100.0

[152]:
```
#Part D & Part E
#The spherical constraint was enforced in the __init__ function of the
 ↪perceptron class.
# It is applied when the 'constraint' parameter is set to True
qr= Perceptron(3,constraint=True)

qr.fit_sto(x,y)

ypred= qr.pred(x)

percentage = (np.mean((ypred * y) == 1)) * 100
print(percentage)
w_tilde = qr.weights
```

17 updates made
100.0

[153]:
```
#Part F

# My algorithm took 12 iterations (may vary if re-run)
# The maximum upper bound for my initialization is 20. So it worked well within
 ↪the bounds

data_b = add_bias(x)

# print(w_tilde.shape,data_b.shape)
gamma = np.min((w_tilde @ data_b.T) * y)
alpha = beta2 / gamma
def upperbounds(w0,alpha=alpha,w_tilde=w_tilde,beta2=beta2):
    k_0 = np.ceil((np.linalg.norm(w0 - alpha * w_tilde) ** 2) / beta2)
    return k_0
k_0 = upperbounds(qr.w0)


mpp = {
"Vector of Zeros:": np.zeros(4),
"Gaussian Mean 0:": np.random.randn(4),
"Gaussian Mean 100:": np.random.randn(4) + np.array([100,100,100,100]),
"Uniform |x| <= 1": np.random.uniform(-1,1,4)




}
```

```python
for k,v in mpp.items():
    print(k, "\nNum Updates:", test_init(v), "\nUpperbound:
↪",upperbounds(v),"\n")



# print(k_0)
```

```
Vector of Zeros:
Num Updates: 13
Upperbound: 33.0

Gaussian Mean 0:
Num Updates: 10
Upperbound: 40.0

Gaussian Mean 100:
Num Updates: 35
Upperbound: 369.0

Uniform |x| <= 1
Num Updates: 10
Upperbound: 39.0
```

[154]:

```python
# w_tilde @ data_b.T * y
```

[161]:

```python
# x2 = np.array([
# (-1, 0),
# (-1, -1),
# (-2, 0),
# (-2, 1),
# (1, 0),
# (0, 1),
# (0, 2),
# (0, 3),
# ])



# y2 = np.array([-1,-1,-1,-1,1,1,1,1])
# # print(x2.shape,y.shape)
# w_tilde = np.array([1.,1.])
# w_0 = np.zeros_like(w_tilde)
# # w_tilde *= 1000000000
# # w_tilde *= 0.000001
# gamma = np.min(np.linalg.norm(w_tilde @ x2.T * y2.T))

# norms = np.linalg.norm(x2,axis=1)
```

```
# beta2_i = np.argmax(norms)
# beta2 = np.max(norms)

# # print(np.sign(w_tilde @ x2.T))
# w2 =np.linalg.norm(w_tilde)**2
# print(beta2)
# print(gamma)
# print(beta2 * w2)
```

```
3.0
5.0
6.000000000000002
```