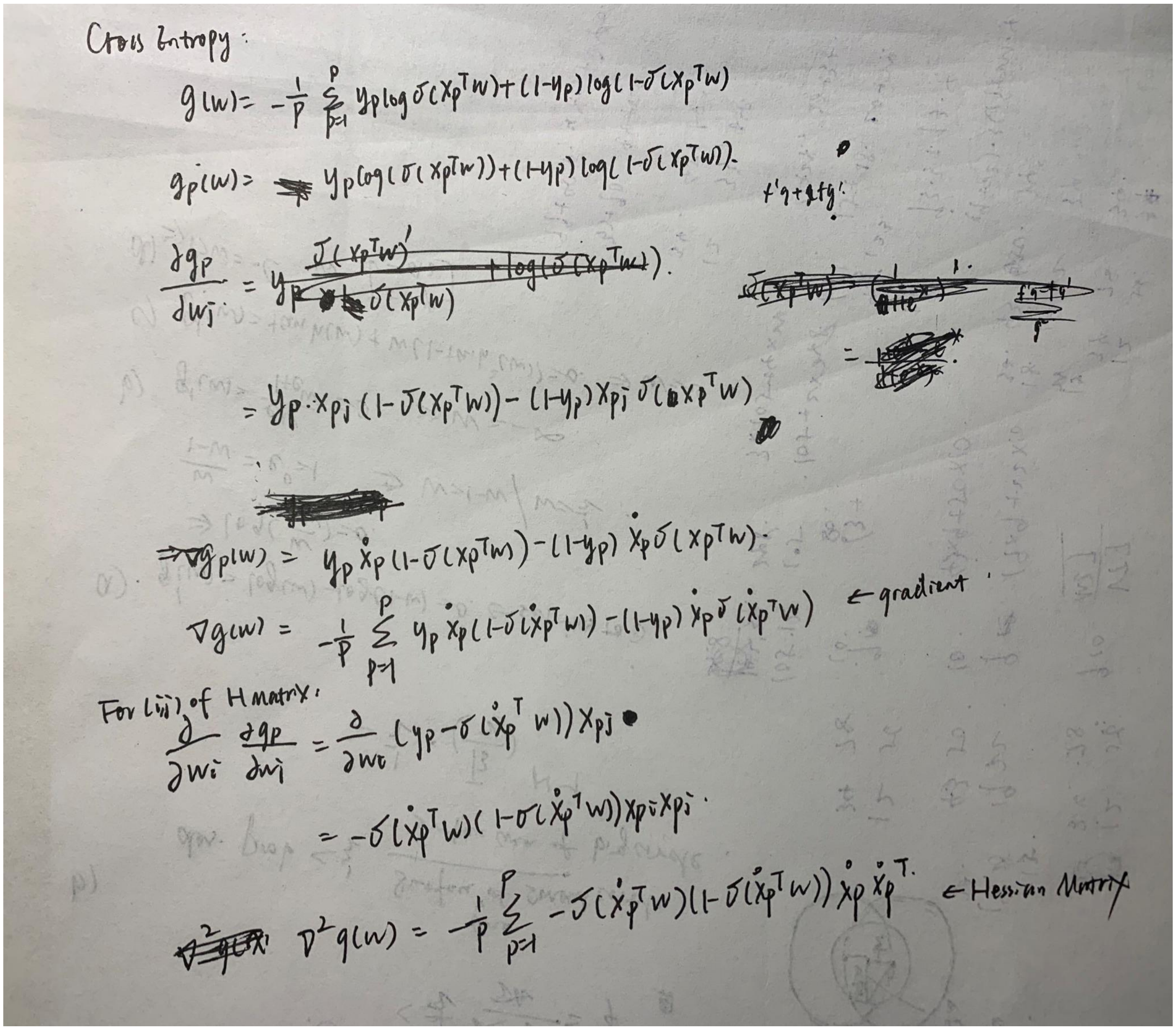
6.5



Cross Entropy:

$$g(w) = -\frac{1}{P}\sum_{p=1}^{P} y_p \log\sigma(x_p^T w) + (1-y_p)\log(1-\sigma(x_p^T w))$$

$$g_p(w) = y_p \log(\sigma(x_p^T w)) + (1-y_p)\log(1-\sigma(x_p^T w)).$$

$f'g+gf'$

$$\frac{\partial g_p}{\partial w_j} = y_p \cancel{\frac{\sigma(x_p^T w)'}{\sigma(x_p^T w)} + \log(\sigma(x_p^T w))}).$$

$$= y_p \cdot x_{pj}(1-\sigma(x_p^T w)) - (1-y_p)x_{pj}\sigma(x_p^T w)$$

$$\nabla g_p(w) = y_p \mathring{x}_p(1-\sigma(x_p^T w)) - (1-y_p)\mathring{x}_p\sigma(x_p^T w).$$

$$\nabla g(w) = -\frac{1}{P}\sum_{p=1}^{P} y_p \mathring{x}_p(1-\sigma(\mathring{x}_p^T w)) - (1-y_p)\mathring{x}_p\sigma(\mathring{x}_p^T w)$$ ← gradient

For (i,j) of H matrix:

$$\frac{\partial}{\partial w_i}\frac{\partial g_p}{\partial w_j} = \frac{\partial}{\partial w_i}(y_p - \sigma(\mathring{x}_p^T w))x_{pj}$$

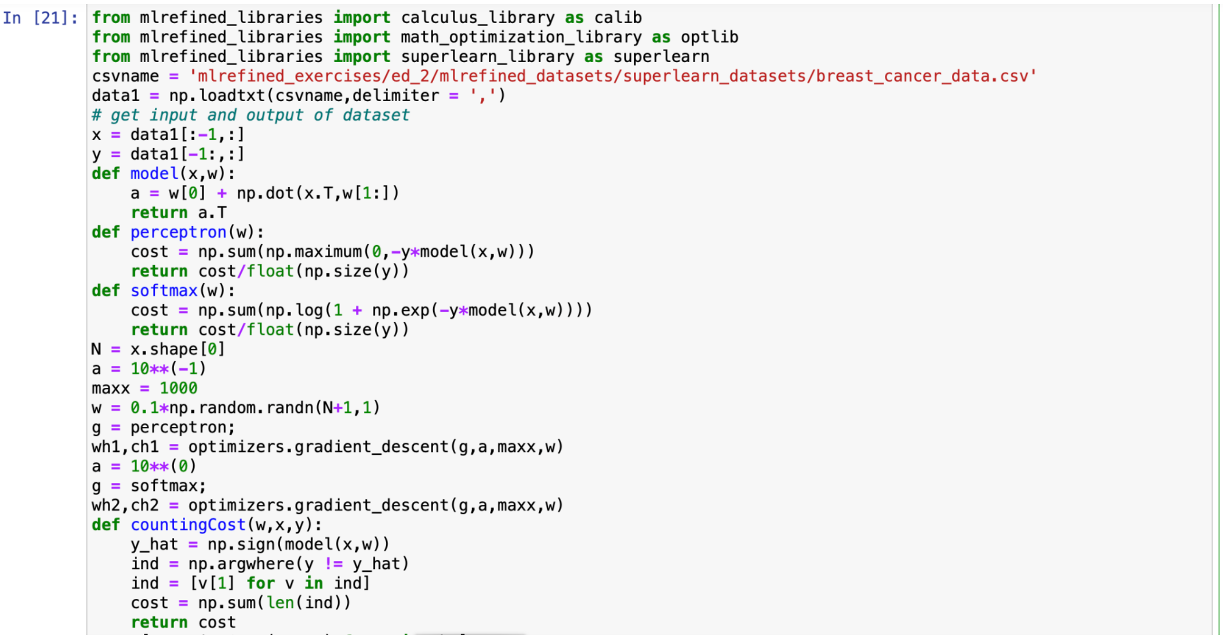
$$= -\sigma(\mathring{x}_p^T w)(1-\sigma(\mathring{x}_p^T w))x_{pi}x_{pj}.$$

$$\nabla^2 g(w) = -\frac{1}{P}\sum_{p=1}^{P} -\sigma(\mathring{x}_p^T w)(1-\sigma(\mathring{x}_p^T w))\mathring{x}_p\mathring{x}_p^T.$$ ← Hessian Matrix

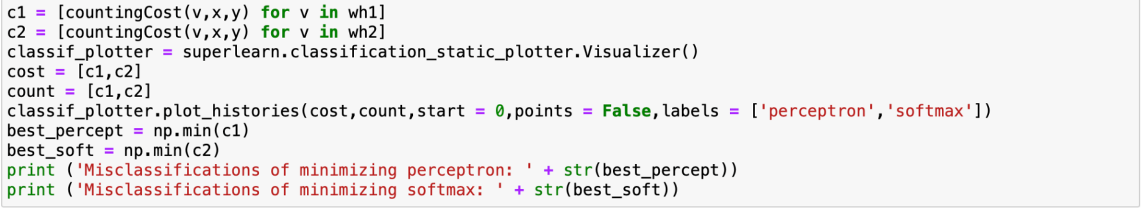
6.10

//

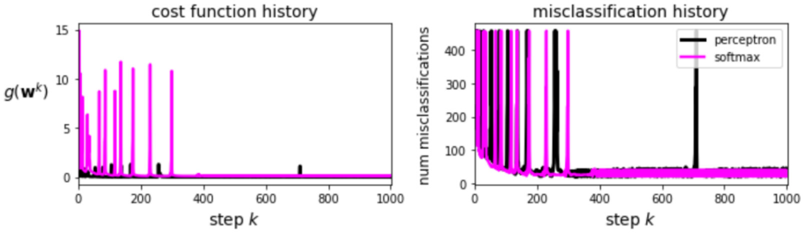
6.13



```
In [21]: from mlrefined_libraries import calculus_library as calib
         from mlrefined_libraries import math_optimization_library as optlib
         from mlrefined_libraries import superlearn_library as superlearn
         csvname = 'mlrefined_exercises/ed_2/mlrefined_datasets/superlearn_datasets/breast_cancer_data.csv'
         data1 = np.loadtxt(csvname,delimiter = ',')
         # get input and output of dataset
         x = data1[:-1,:]
         y = data1[-1:,:]
         def model(x,w):
             a = w[0] + np.dot(x.T,w[1:])
             return a.T
         def perceptron(w):
             cost = np.sum(np.maximum(0,-y*model(x,w)))
             return cost/float(np.size(y))
         def softmax(w):
             cost = np.sum(np.log(1 + np.exp(-y*model(x,w))))
             return cost/float(np.size(y))
         N = x.shape[0]
         a = 10**(-1)
         maxx = 1000
         w = 0.1*np.random.randn(N+1,1)
         g = perceptron;
         wh1,ch1 = optimizers.gradient_descent(g,a,maxx,w)
         a = 10**(0)
         g = softmax;
         wh2,ch2 = optimizers.gradient_descent(g,a,maxx,w)
         def countingCost(w,x,y):
             y_hat = np.sign(model(x,w))
             ind = np.argwhere(y != y_hat)
             ind = [v[1] for v in ind]
             cost = np.sum(len(ind))
             return cost
```



```
c1 = [countingCost(v,x,y) for v in wh1]
c2 = [countingCost(v,x,y) for v in wh2]
classif_plotter = superlearn.classification_static_plotter.Visualizer()
cost = [c1,c2]
count = [c1,c2]
classif_plotter.plot_histories(cost,count,start = 0,points = False,labels = ['perceptron','softmax'])
best_percept = np.min(c1)
best_soft = np.min(c2)
print ('Misclassifications of minimizing perceptron: ' + str(best_percept))
print ('Misclassifications of minimizing softmax: ' + str(best_soft))
```

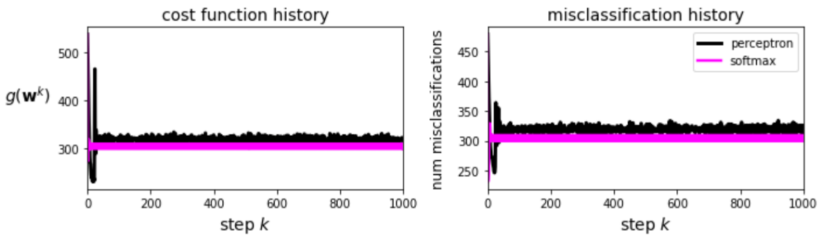


```
the smallest number of misclassifications provided by minimizing the perceptron 19
the smallest number of misclassifications provided by minimizing the softmax 21
```

6.15

```
In [38]: def standardNormalizer(x):
             xmeans = np.nanmean(x,axis = 1)[:,np.newaxis]
             xstds = np.nanstd(x,axis = 1)[:,np.newaxis]
             ind = np.argwhere(xstds < 10**(-2))
             if len(ind) > 0:
                 ind = [v[0] for v in ind]
                 adjust = np.zeros((xstds.shape))
                 adjust[ind] = 1.0
                 xstds += adjust
             ind = np.argwhere(np.isnan(x) == True)
             for i in ind:
                 x[i[0],i[1]] = xmeans[i[0]]
             normalizer = lambda data: (data - xmeans)/xstds
             inverse_norm = lambda data: data*xstds + xmeans
             return normalizer,inverse_norm
```

```
In [39]: csvname = 'mlrefined_exercises/ed_2/mlrefined_datasets/superlearn_datasets/credit_dataset.csv'
         data = np.loadtxt(csvname,delimiter = ',')
         x = data[:-1,:]
         y = data[-1:,:]
         ind0 = np.argwhere(y==-1)
         ind1 = np.argwhere(y==+1)
         normalizer,inverse_norm = standardNormalizer(x)
         x = normalizer(x)
         N = x.shape[0]
         a = 10**(-1)
         maxx = 1000
         w = 0.1*np.random.randn(N+1,1)
         g = perceptron;
         wh1,ch1 = optimizers.gradient_descent(g,a,maxx,w)
         a = 10**(1)
         g = softmax;
         wh2,ch2 = optimizers.gradient_descent(g,a,maxx,w)
         def countingCost(w,x,y):
             y_hat = np.sign(model(x,w))
             ind = np.argwhere(y != y_hat)
             ind = [v[1] for v in ind]
             cost = np.sum(len(ind))
             return cost
         c1 = [countingCost(v,x,y) for v in wh1]
         c2 = [countingCost(v,x,y) for v in wh2]
         classif_plotter = superlearn.classification_static_plotter.Visualizer()
         cost = [c1,c2]
         count = [count_history_1,count_history_2]
         classif_plotter.plot_histories(cost,count,start = 0,points = False,labels = ['perceptron','softmax'])
         best = np.min(ch1)
         acc = (1 - best/y.size)
         print ('Misclassifications of minimizing the perceptron: ' + str(best))
         print ('Accuracy of minimizing: ' + str(acc))
```



```
Misclassifications of minimizing the perceptron: 0.002018065018519048
Accuracy of minimizing: 0.9999979819349815
```

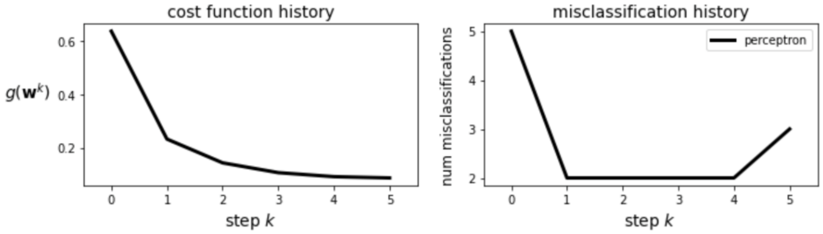
6.16

In [49]:

```
def balanced_accuracy(w,x,y):
    yhat = np.sign(model(x,w))
    ind0 = np.argwhere(y == -1)
    ind0 = [v[1] for v in ind0]
    num0 = len(ind0)
    ind = np.argwhere(np.abs(y[:,ind0] - yhat[:,ind0]) > 0)
    c0 = len(ind)
    ind1 = np.argwhere(y == +1)
    ind1 = [v[1] for v in ind1]
    num1 = len(ind1)
    ind = np.argwhere(np.abs(y[:,ind1] - yhat[:,ind1]) > 0)
    c1 = len(ind)
    acc0 = 1 - c0/num0
    acc1 = 1 - c1/num1
    return (acc0 + acc1)/2
betas = np.array([1.0,5.0])
def sigmoid(t):
    return 1/(1 + np.exp(-t))
def weighted_softmax(w,betas):
    a = sigmoid(model(x,w))
    ind = np.argwhere(y == -1)[:,1]
    cost = -betas[0]*np.sum(np.log(1 - a[:,ind]))
    ind = np.argwhere(y==+1)[:,1]
    cost -= betas[1]*np.sum(np.log(a[:,ind]))
    return cost/y.size
```

In [50]:

```
csvname = 'mlrefined_exercises/ed_2/mlrefined_datasets/superlearn_datasets/3d_classification_data_v2_mbalanced.csv'
data1 = np.loadtxt(csvname,delimiter = ',')
x = data1[:-1,:]
y = data1[-1:,:]
ind0 = np.argwhere(y==-1)
ind1 = np.argwhere(y==+1)
betas = np.array([1.0,1.0])
softmax = lambda w,betas = betas: weighted_softmax(w,betas)
N = x.shape[0]
maxx = 5
w = 0.1*np.random.randn(N+1,1)
g = softmax;
wh1,c1 = optimizers.newtons_method(g,maxx,w)
classif_plotter = superlearn.classification_static_plotter.Visualizer()
cost = [c1]
count = [[counting_cost(v,x,y) for v in wh1]]
classif_plotter.plot_histories(cost,count,start = 0,points = False,labels = ['perceptron','softmax'])
```


cost function history
$g(\mathbf{w}^k)$
step k
misclassification history
num misclassifications
perceptron
step k