~\Documents\documents_general\structured_courses\math564\evaluations\projects\p04\nnloss.py

```python
def nnloss(x,p,nargout):
    """
    Feed-Forward Neural Network loss (or) classfication
    """

    train_data,train_class,layer_size,task=p

    # construct weight matrices
    n=len(layer_size)
    W=[np.array([])]*(n-1)
    a=0
    for k in range(n-1):
        c=layer_size[k]
        r=layer_size[k+1]
        b=a+r*c
        W[k]=(x[a:b,0]).reshape((r,c))
        a=b

    # forward computation
    LL=[np.array([])]*(n-1)
    t=W[0].dot(train_data)
    LL[0]=activation(t)
    for k in range(1,n-1):
        t=W[k].dot(LL[k-1])
        LL[k]=activation(t)

    # if the task is to classify then stop
    if task=='classify':
        return LL[n-2]

    # if the task is to compute loss then do so
    # along with gradient if requested
    f=0.5*((LL[n-2]-train_class)**2).sum()

    # gradient computation by backpropagation
    if nargout>1:
        g=np.zeros((0,1))
        t=LL[n-2]-train_class
        for k in range(n-2,-1,-1):
            h=t*LL[k]*(1.-LL[k])
            t=(W[k].T).dot(h)
            if k>0:
                G=h.dot(LL[k-1].T)
            else:
                G=h.dot(train_data.T)
            g=np.concatenate((G.reshape((-1,1)),g))
        return f,g
    else:
        return f

def activation(x):
    import numpy as np
    return 1./(1.+np.exp(-x))

```