```
import numpy as np

class Convolution1d :
    def __init__(self, filt) :
        self.__filt = filt
        self.__r = filt.size
        self.T = TransposedConvolution1d(self.__filt)

    def __matmul__(self, vector) :
        r, n = self.__r, vector.size

        return np.asarray([np.sum(self.__filt*vector[i:i+r]) for i in range(n-r+1)])  # IMPLEMENT THIS

class TransposedConvolution1d :

# Transpose of 1-dimensional convolution operator used for the
# transpose-convolution operation A.T@(...)

    def __init__(self, filt) :
        self.__filt = filt
        self.__r = filt.size

    def __matmul__(self, vector) :
        r = self.__r
        n = vector.size + r - 1

        flip_filt = self.__filt[::-1]

        return np.asarray([np.sum(flip_filt[np.maximum(0,r-1-i):min(n-i,r)]
                           *vector[np.maximum(0,i+1-r):min(i+1,n-r+1)]) for i in range(n)]) # IMPLEMENT THIS

def huber_loss(x) :
    return np.sum( (1/2)*(x**2)*(np.abs(x)<=1) + (np.sign(x)*x-1/2)*(np.abs(x)>1) )
def huber_grad(x) :
    return x*(np.abs(x)<=1) + np.sign(x)*(np.abs(x)>1)


r, n, lam = 3, 20, 0.1

np.random.seed(0)
k = np.random.randn(r)
b = np.random.randn(n-r+1)
A = Convolution1d(k)
#from scipy.linalg import circulant
#A = circulant(np.concatenate((np.flip(k),np.zeros(n-r))))[r-1:,:]


x = np.zeros(n)
alpha = 0.01
for _ in range(100) :
    x = x - alpha*(A.T@(huber_grad(A@x-b))+lam*x)

print(huber_loss(A@x-b)+0.5*lam*np.linalg.norm(x)**2)
```

Result

```
~ python conv1D.py
0.4587586843129764
```