H13-1

a)

Names	Email Address
Qingyang Zhao	jhun0324@berkeley.edu

Description of Team: Best Group Ever

How did I work?

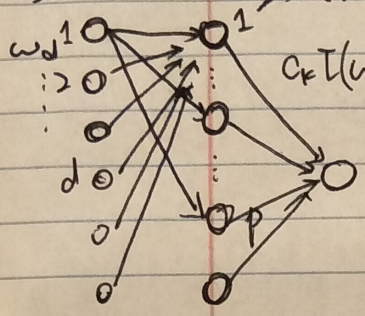
Comments:

b) I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

Qingyang Zhao

HW13-2. R^d R R.

1. a( ) → (w_k, b_k, c_k) $k=1\cdots p$



$c_k T(w_k^T x + b_k)$.

$h(x) = \sum_{k=1}^{p} c_k T(w_k^T x + b_k)$

a) Thresholding function: $T: \mathbb{R} \to \mathbb{R}$, $T(z) \to 1$ as $z \to \infty$; $T(z) \to 0$ as $z \to -\infty$

① $\sigma(t) = \frac{1}{1+e^{-t}}$ $1+e^{-t} \neq 0$ thus $\sigma(t): \mathbb{R} \to \mathbb{R}$

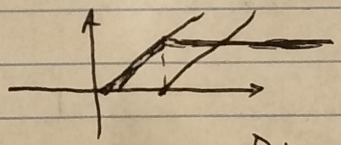
as $t \to \infty$ $\lim_{t\to\infty} \sigma(t) = \lim_{t\to\infty} \frac{1}{1+e^{-t}} = \lim_{t\to\infty} \frac{1}{1+0} = 1$

as $t \to -\infty$ $\lim_{t\to\infty} \sigma(t) = \lim_{t\to -\infty} \frac{1}{1+\infty} = 0$

② ~~Relu(t)~~ Relu(t)-Relu(t-1)

$\max(0,t) - \max(0,t-1)$

$$= \begin{cases} 0 & t<0 \\ t & 0 \le t \le 1 \\ 1 & t>1 \end{cases}$$



thus as $t \to \infty$ Relu(t)-Relu(t-1) = 1

as $t \to -\infty$ Relu(t)-Relu(t-1) = 0.

Thus, they are all thresholding functions.

b) In the graphs

c) Since S is Indicator function, $S = \begin{cases} 1, & \langle w', x\rangle + b' \ge 0 \\ 0, & \langle w', x\rangle + b < 0 \end{cases}$

lhs = {0, 1} Fixed x

For rhs, since closure includes the limit points, $\exists w, b$ s.t.

$cl(T(\langle w, x\rangle + b)) \supseteq \{0, 1\}$ (0: $x \to -\infty$; 1: $x \to +\infty$)

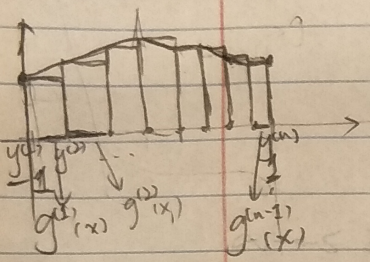
lhs ⊆ rhs

$S(\langle w', x\rangle + b') \subseteq cl(\{T(\langle w, x\rangle + b) \text{ for some } w, b\})$

d) use $S_\delta(y)$ to estimate $C(y')$ $\quad y' \in [-1, 1]$

Define Piecewise Constant function as: $g^{(i)}(x) = \begin{cases} C(y^{(i)}), & y^{(i)} \le x < y^{(i+1)} \\ 0, & \text{otherwise} \end{cases}$ $\quad i = 1, \cdots, n-1, \quad n = |S_\delta|$

First pick a set $P_\delta = \{ y^{(i)} \mid -1 = y^{(1)} < y^{(2)} < \cdots < y^{(n)} = 1, \ |g^{(i)}(x) - C(x)| \le \varepsilon \text{ for } \forall x \in [y^{(i)}, y^{(i+1)}) \}$



Since $|g^{(i)}(x) - C(x)| \le \varepsilon$

$\int_{y^{(i)}}^{y^{(i+1)}} |g^{(i)}(x) - C(x)| dx \le \Delta y^{(i)} \cdot \varepsilon$, where $\Delta y^{(i)} = y^{(i+1)} - y^{(i)}$

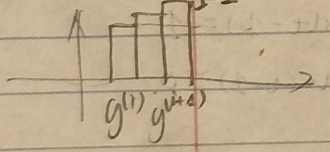
use $\sum_{i=1}^{n-1} g^{(i)}(x)$ to estimate $C(x)$

$$\text{error} = \int_{-1}^{1} \left| \sum_{i=1}^{n-1} g^{(i)}(x) - C(x) \right| dx \le \sum_{i=1}^{n-1} \int_{y^{(i)}}^{y^{(i+1)}} |g^{(i)}(x) - C(x)| dx \le \sum_{i=1}^{n-1} \Delta y^{(i)} \cdot \varepsilon = 2\varepsilon$$

Since, we can use Linear Combination of step function represents for $\sum_{i=1}^{n-1} g^{(i)}(x)$.

We conclude that $C(y')$ can be approximated by Linear Combination of Step Combination with ε-closeness.

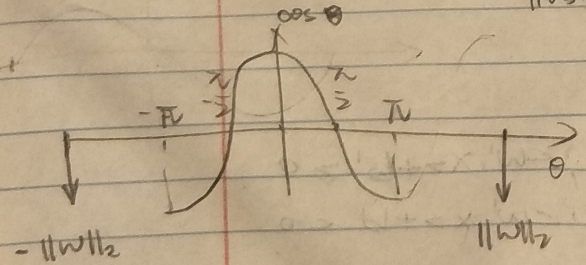
Supplement: $\sum_{j=1}^{n} g^{(j)}(x) = S_1(x) + \sum_{i=2}^{n} (C(y^{(i+1)}) - C(y^{(i)})) S_i(x)$ where $S_i(x) = \begin{cases} 1, & x \ge y^{(i)} \\ 0, & x < y^{(i)} \end{cases}$

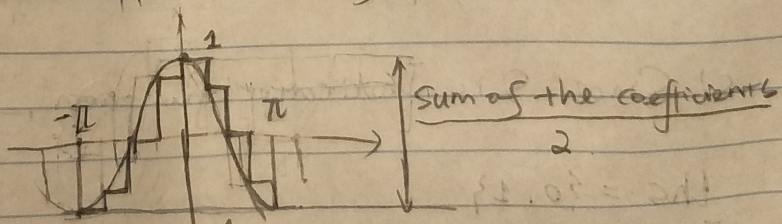


e) $\cos(\langle w, x \rangle) = \cos(\|w\|_1 y') \quad y' \in [-1, 1]$

$\|w\|_1 y' \in [-\|w\|_1, \|w\|_1]$

Function Approximation (if $\|w\|_1 = \pi$)





We want to know how many periods include in $[-\|w\|_2, \|w\|_2]$

$$\sum_{i=1} |C(z_i) - C(z_w)| \le \frac{2\|w\|_2}{2\pi} \times 4 \le 2\|w\|_1$$

$\frac{2\|w\|_2}{2\pi}$: # of periods; 4: range of 'cosine'

f) $\mathcal{F} = \{ f : \mathbb{R}^d \to \mathbb{R} \text{ s.t. } f(x) = \frac{1}{2\|w\|_1} \cos(\langle w, x \rangle) \text{ for some } w \neq 0 \}$.

$f(x) = \frac{1}{2\|w\|_1} \cos(\langle w, x \rangle)$

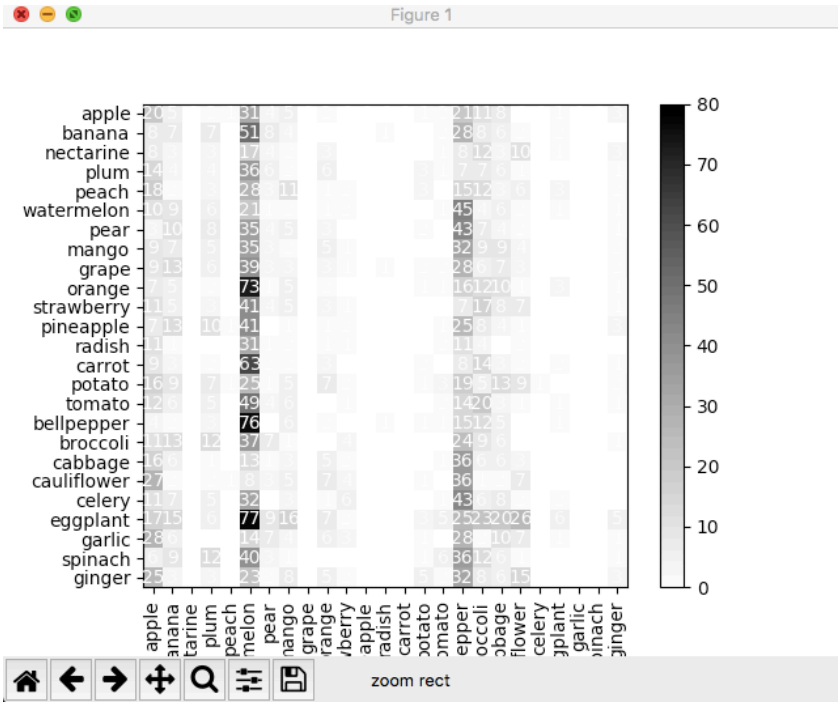
$= \sum_k \alpha_k S_k(\langle w, x \rangle)$

since $\sum_k \alpha_k \leq 2\|w\|_1$ $\quad \sum_k \frac{\alpha_k}{2\|w\|_1} \leq 1$

Thus $\mathcal{F} \in \overline{\text{conv}}\{ p : p(x) = \tau(\langle w, x \rangle + b) \text{ or } p(x) = -\tau(\langle w, x \rangle + b), \text{ for some } w, b \}$

g)

a)



b)

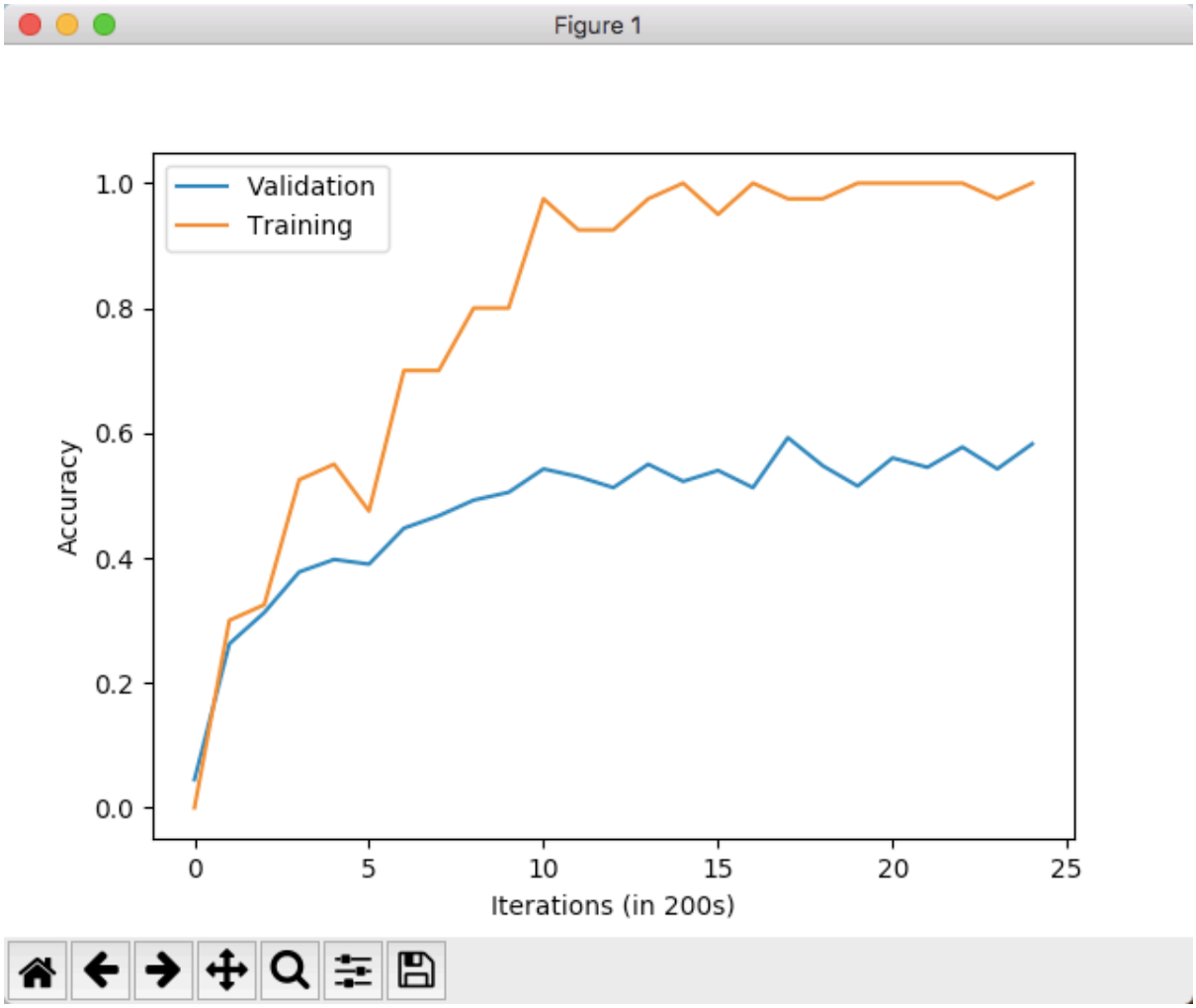
data_manager.py

```
def load_train_set(self):
    '''
    Loads the train set
    '''

    self.train_data = self.load_set('train')
    self.feature = []
    self.label = []
    for i in range(len(self.train_data)):
        self.feature.append(self.train_data[i]['features'])
        self.label.append(self.train_data[i]['label'])
    self.feature = np.asarray(self.feature, dtype = np.float32)
    self.label = np.asarray(self.label, dtype = np.float32)


def load_validation_set(self):
    '''
    Loads the validation set
    '''

    self.val_data = self.load_set('val')
    self.val_feature = []
    self.val_label = []
    for i in range(len(self.val_data)):
        self.val_feature.append(self.val_data[i]['features'])
        self.val_label.append(self.val_data[i]['label'])
    self.val_feature = np.asarray(self.val_feature, dtype = np.float32)
    self.val_label = np.asarray(self.val_label, dtype = np.float32)
```

```
    def get_train_batch(self):

        '''

        Compute a training batch for the neural network
        The batch size should be size 40

        '''
        # random pick image
        ind = np.random.choice(len(self.train_data), self.batch_size)
        return self.feature[ind,:,:,:], self.label[ind,:]

    def get_validation_batch(self):

        '''
        Compute a training batch for the neural network

        The batch size should be size 400

        '''
        #FILL IN
        ind = np.random.choice(len(self.val_data), self.val_batch_size)
        return self.val_feature[ind,:,:,:], self.val_label[ind,:]
```

c)

```
    def optimize(self):

        self.train_losses = []
        self.test_losses = []

        '''
        Performs the training of the network.
        Implement SGD using the data manager to compute the batches
        Make sure to record the training and test loss through out the process
        '''
        #x = tf.placeholder(tf.float32, [None, self.data.image_size, self.data.image_size,
3])
        #y_ = tf.placeholder(tf.float32, [None, self.net.num_class])
        for i in range(self.max_iter):
            feature_batch, label_batch = self.data.get_train_batch()
            feed_dict = {self.net.images: feature_batch, self.net.labels: label_batch}
            self.sess.run(self.train, feed_dict = feed_dict)
            if np.mod(i, self.summary_iter) == 0:
                val_feature_batch, val_label_batch = self.data.get_validation_batch()
                val_feed_dict = {self.net.images: val_feature_batch, self.net.labels:
val_label_batch}

self.test_losses.append(self.sess.run(self.net.accurracy,feed_dict=val_feed_dict))

self.train_losses.append(self.sess.run(self.net.accurracy,feed_dict=feed_dict))
                print "loss for Train:", self.train_losses[-1]
                print "loss for Validation:", self.test_losses[-1]
```



d)

viz_features.py

```
def vizualize_features(self,net):

    images = [0,10,100]
    '''
    Compute the response map for the index images
    '''
    f, axarr = plt.subplots(len(images), 5)
    j = 0
    for img in images:
        val_feature = np.asarray([self.val_data[img]['features']], dtype = np.float32)
        val_label = np.asarray([self.val_data[img]['label']],dtype = np.float32)
        feed_dict = {net.images: val_feature, net.labels: val_label}
        out_img = self.sess.run(net.conv2d_out, feed_dict = feed_dict)
        # the shape o
        #f out_img is (1,90,90,5)
        for i in range(out_img.shape[3]):
            # cv2.namedWindow("Response Map")
            axarr[j, i].imshow(self.revert_image(out_img[0,:,:,i]))
            #plt.imshow(self.revert_image(out_img[0,:,:,i]))
        j = j + 1
    plt.show()


def revert_image(self,img):
    '''
```

```
loss for Train: 0.0
loss for Validation: 0.045
loss for Train: 0.3
loss for Validation: 0.2625
loss for Train: 0.325
loss for Validation: 0.3125
loss for Train: 0.525
loss for Validation: 0.3775
loss for Train: 0.55
loss for Validation: 0.3975
loss for Train: 0.475
loss for Validation: 0.39
loss for Train: 0.7
loss for Validation: 0.4475
loss for Train: 0.7
loss for Validation: 0.4675
loss for Train: 0.8
loss for Validation: 0.4925
loss for Train: 0.8
loss for Validation: 0.505
loss for Train: 0.975
loss for Validation: 0.5425
loss for Train: 0.925
loss for Validation: 0.53
loss for Train: 0.925
loss for Validation: 0.5125
loss for Train: 0.975
loss for Validation: 0.55
loss for Train: 1.0
loss for Validation: 0.5225
loss for Train: 0.95
loss for Validation: 0.54
loss for Train: 1.0
loss for Validation: 0.5125
loss for Train: 0.975
loss for Validation: 0.5925
loss for Train: 0.975
loss for Validation: 0.5475
loss for Train: 1.0
loss for Validation: 0.515
loss for Train: 1.0
loss for Validation: 0.56
loss for Train: 1.0
loss for Validation: 0.545
loss for Train: 1.0
loss for Validation: 0.5775
loss for Train: 0.975
loss for Validation: 0.5425
loss for Train: 1.0
loss for Validation: 0.5825
```

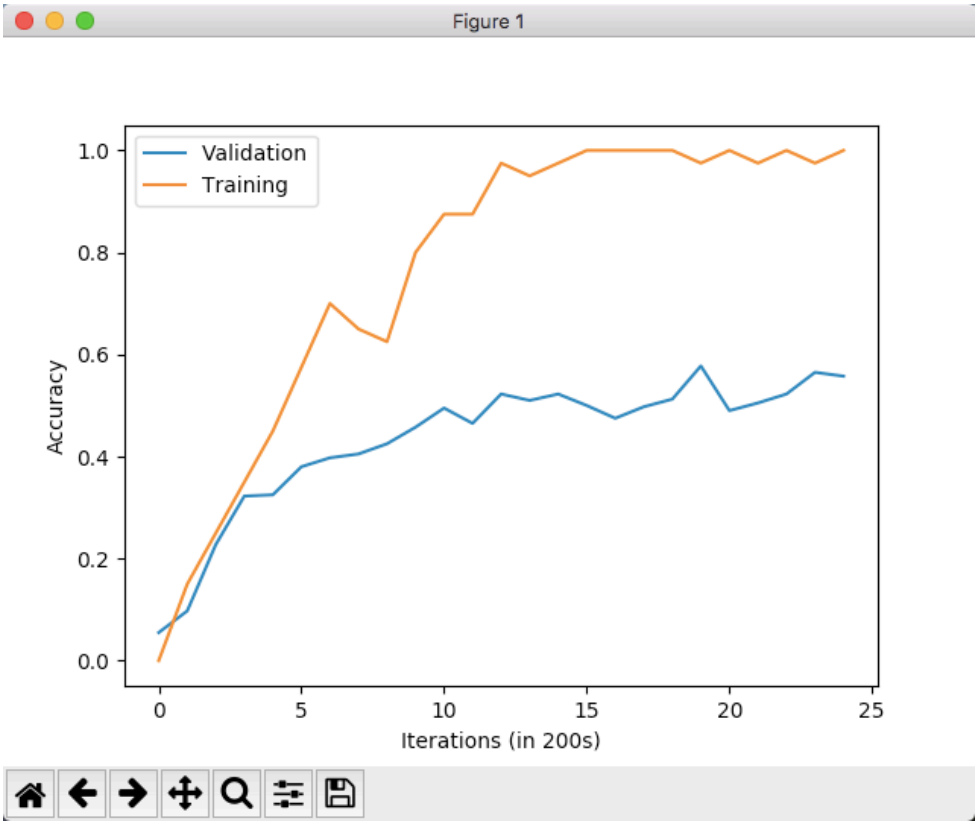
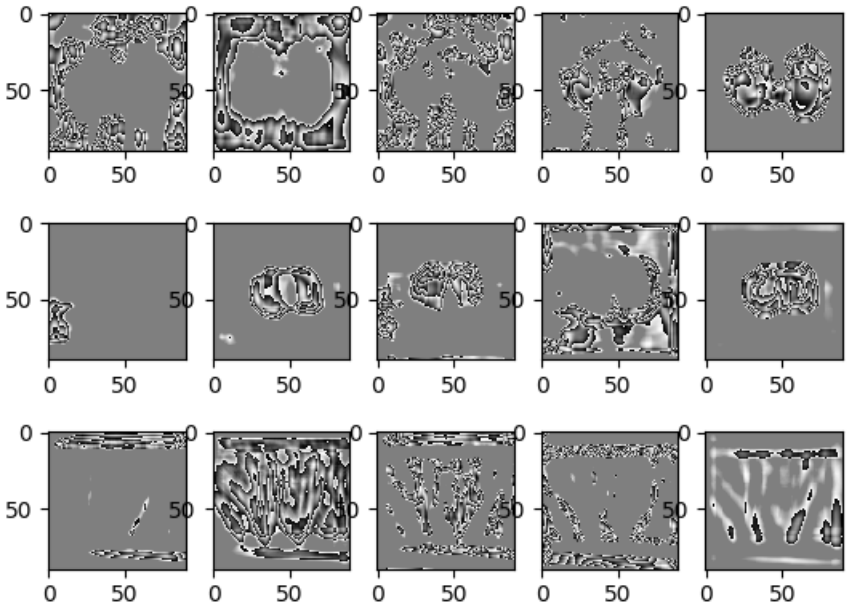


Figure 1
Validation
Training
Accuracy
Iterations (in 200s)
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25

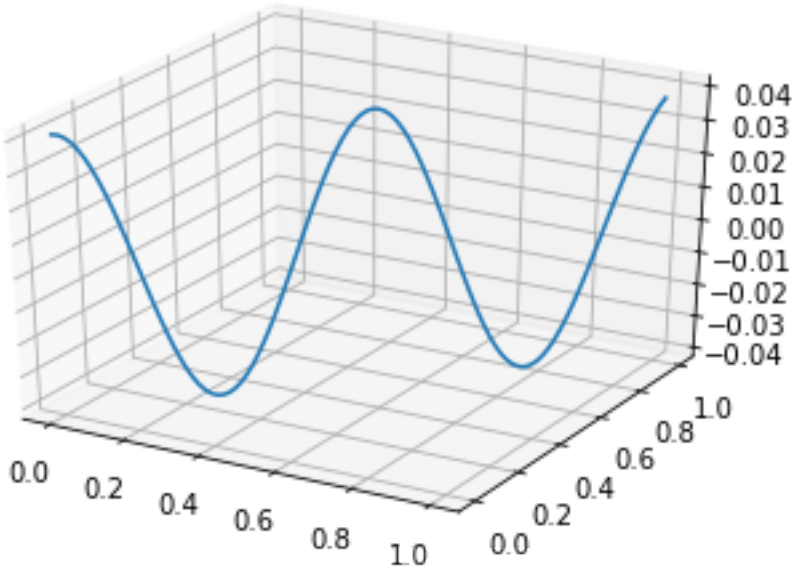

0
50

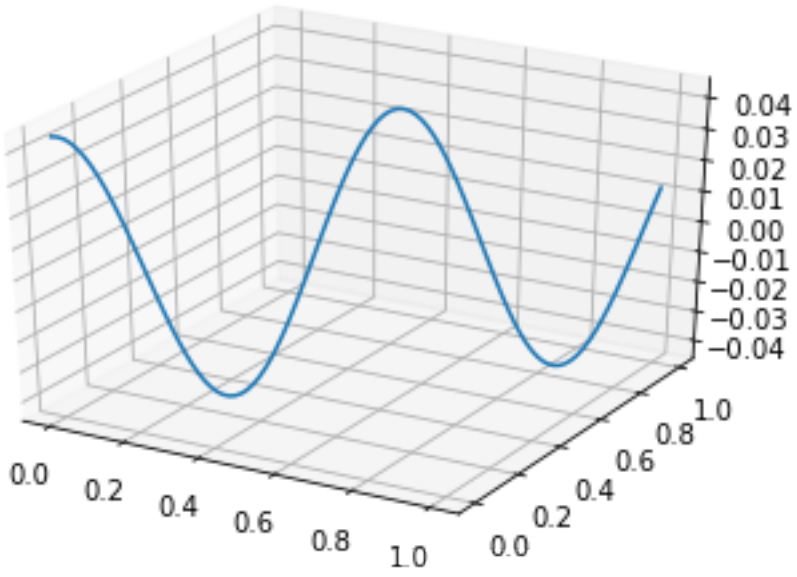
HW13-2b

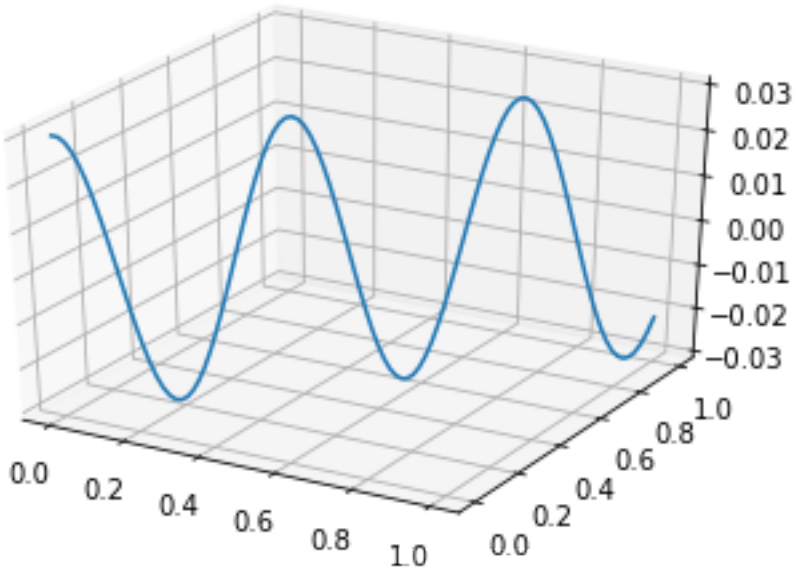
December 1, 2017

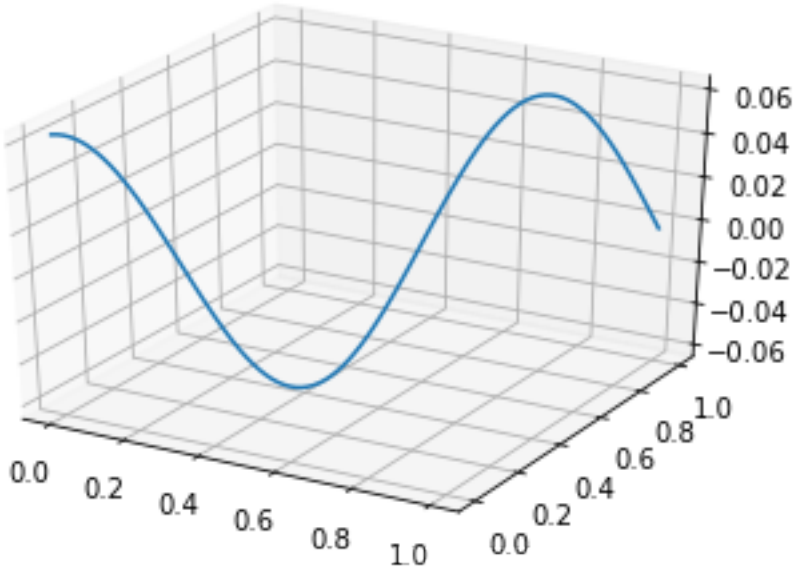
In [7]:
```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
```

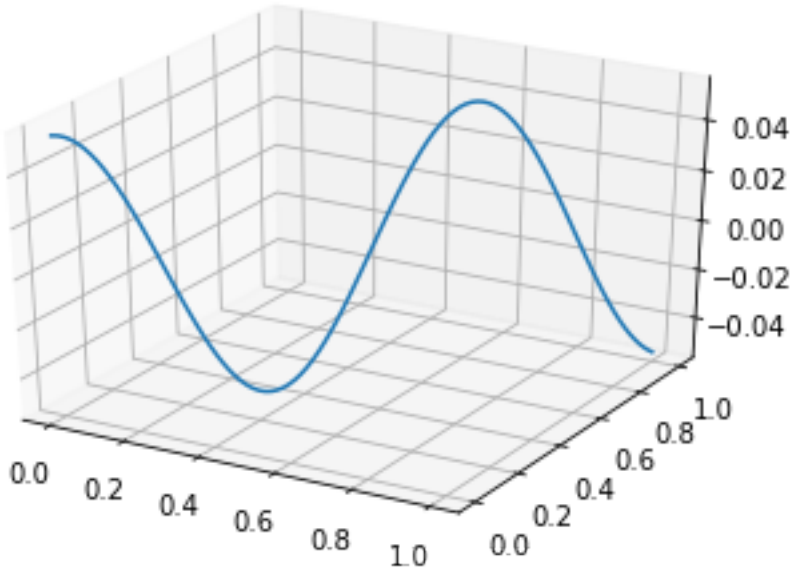
In [21]:
```
mpl.rcParams['legend.fontsize'] = 10

for i in range(5):
    w = np.random.random_sample(2)+np.random.random_sample(2)*10
    fig =plt.figure()
    ax = fig.gca(projection = '3d')
    x = np.linspace(0,1,100)
    y = np.linspace(0,1,100)
    z = 1/(np.linalg.norm(w,1)*2)*np.cos(x*w[0] + y*w[1])
    #label = 'w is ' + w[0] + ',' + w[1]
    ax.plot(x, y, z)
plt.show()
```




0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.04
0.02
0.00
−0.02
−0.04
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0