Machine Learning HW7

Ben Zhang, bz957

May 11, 2018

3 Ridge Regression

3.1

```
class L2NormPenaltyNode(object):
    """ Node computing l2_reg * ||w||^2 for scalars l2_reg and vector w"""
    def __init__(self, l2_reg, w, node_name):
        """
        Parameters:
        l2_reg: a scalar value >=0 (not a node)
        w: a node for which w.out is a numpy vector
        node_name: node's name (a string)
        """
        self.node_name = node_name
        self.out = None
        self.d_out = None
        self.l2_reg = l2_reg
        self.w = w

    def forward(self):
        self.out = np.sum(self.w.out ** 2)*self.l2_reg   ###
        self.d_out = np.zeros(self.out.shape)
        return self.out

    def backward(self):
        d_w = self.d_out * 2 * self.w.out * self.l2_reg
        self.w.d_out += d_w
        return self.d_out

    def get_predecessors(self):
        return [self.w]
```

3.2

```python
class SumNode(object):
    """ Node computing a + b, for numpy arrays a and b"""
    def __init__(self, a, b, node_name):
        """
        Parameters:
        a: node for which a.out is a numpy array
        b: node for which b.out is a numpy array of the same shape as a
        node_name: node's name (a string)
        """
        self.node_name = node_name
        self.out = None
        self.d_out = None
        self.a = a
        self.b = b

    def forward(self):
        self.out = self.a.out + self.b.out
        self.d_out = np.zeros(self.out.shape)
        return self.out

    def backward(self):
        d_a = self.d_out
        d_b = self.d_out
        self.a.d_out += d_a
        self.b.d_out += d_b
        return self.d_out

    def get_predecessors(self):
        return [self.a, self.b]
```

3.3

```
class RidgeRegression(BaseEstimator, RegressorMixin):
    """ Ridge regression with computation graph """
    def __init__(self, l2_reg=1, step_size=.005,  max_num_epochs = 5000):
        self.max_num_epochs = max_num_epochs
        self.step_size = step_size
        # Build computation graph
        self.x = nodes.ValueNode(node_name="x") # to hold a vector input
        self.y = nodes.ValueNode(node_name="y") # to hold a scalar response
        self.w = nodes.ValueNode(node_name="w") # to hold the parameter vector
        self.b = nodes.ValueNode(node_name="b") # to hold the bias parameter (scalar)
        self.l2_reg = nodes.ValueNode(node_name="l2_reg") # to hold the reg parameter
        self.prediction = nodes.VectorScalarAffineNode(x=self.x, w=self.w, b=self.b,
                                                       node_name="prediction")
        self.squareloss = nodes.SquaredL2DistanceNode(a=self.prediction, b=self.y,
                                                           node_name="square loss")
        self.l2penalty = nodes.L2NormPenaltyNode(l2_reg=l2_reg, w=self.w,
                                                       node_name="l2 penalty")
        self.objective = nodes.SumNode(a=self.squareloss, b=self.l2penalty,
                                            node_name="objective")

        # Group nodes into types to construct computation graph function
        self.inputs = [self.x]
        self.outcomes = [self.y]
        self.parameters = [self.w, self.b]

        self.graph = graph.ComputationGraphFunction(self.inputs, self.outcomes,
                                                          self.parameters,
                                                          self.prediction,
                                                          self.objective)
        # TODO
```

This code passed the test in *ridge$_r$egression.t.py*

```
/Users/zhangben/PycharmProjects/venv/bin/python /Applications/PyCharm.app/Contents/he
Running /Users/zhangben/Google Drive/NYU/Classes/1003MachineLearning/hw/hw7-backprop/c
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/zhangben/Google Drive/NYU/Classes/1003MachineLearning/hw/hw7-
DEBUG: (Node l2 norm node) Max rel error for partial deriv w.r.t. w is 1.00083540292l9
.DEBUG: (Node sum node) Max rel error for partial deriv w.r.t. a is 1.6365788421260423
DEBUG: (Node sum node) Max rel error for partial deriv w.r.t. b is 1.6365788421260423e
.DEBUG: (Parameter w) Max rel error for partial deriv 3.4803029014850967e-09.
DEBUG: (Parameter b) Max rel error for partial deriv 1.0710782025328269e-09.
.
----------------------------------------------------------------------
Ran 3 tests in 0.003s
OK
```

For this parameter setting,

```
l2reg = 1
    estimator = RidgeRegression(l2_reg=l2reg, step_size=0.00005, max_num_epochs=2000)
```

The average square error on the training set is about 0.21 (for all the epoch),the average square error of last epoch is 0.19

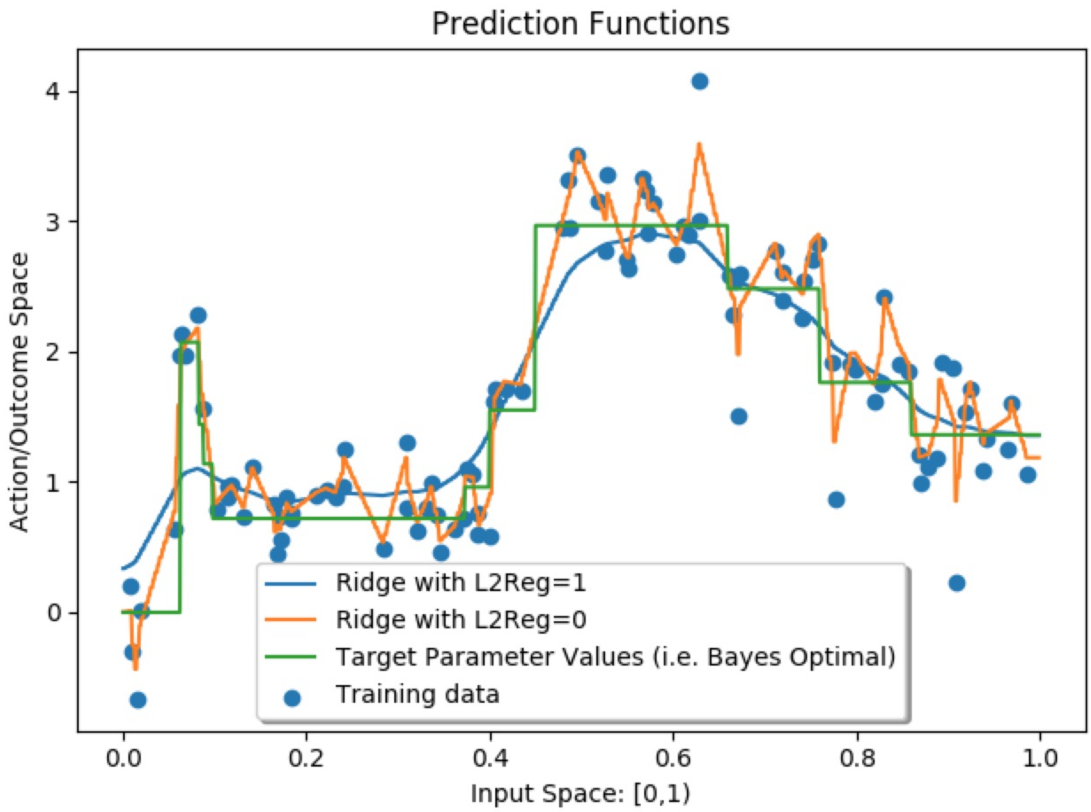
For this parameter setting,

```
 l2reg = 0
    estimator = RidgeRegression(l2_reg=l2reg, step_size=0.0005, max_num_epochs=500)
    estimator.fit(X_train, y_train)
    name = "Ridge with L2Reg="+str(l2reg)
    pred_fns.append({"name":name, "preds": estimator.predict(X) })
```

The average square error on the training set is about 0.09 (for all the epoch),the average square error of last epoch is 0.026

```
/Users/zhangben/PycharmProjects/venv/bin/python "/Users/zhangben/Google Drive/NYU/Classes/1003MachineLearn:
Epoch  0 : Ave objective= 1.1824952961993138  Ave training loss:  0.6463960990296029
Epoch  50 : Ave objective= 0.3212158047986074  Ave training loss:  0.21159871027610827
Epoch  100 : Ave objective= 0.31468140111720166  Ave training loss:  0.20004363361489313
Epoch  150 : Ave objective= 0.3165904315213173  Ave training loss:  0.19830474564021358
Epoch  200 : Ave objective= 0.3160686787534271  Ave training loss:  0.1981266965157925
Epoch  250 : Ave objective= 0.31534423038536125  Ave training loss:  0.19683668417796646
Epoch  300 : Ave objective= 0.31614445968323185  Ave training loss:  0.1973942179156053
Epoch  350 : Ave objective= 0.31406617847386803  Ave training loss:  0.19729557981178755
Epoch  400 : Ave objective= 0.3135215863506393  Ave training loss:  0.19893910160053657
Epoch  450 : Ave objective= 0.3142910132707722  Ave training loss:  0.20009184211031544
Epoch  500 : Ave objective= 0.31444083445655996  Ave training loss:  0.19757106438917962
Epoch  550 : Ave objective= 0.3134930817187636  Ave training loss:  0.20010517902913008
Epoch  600 : Ave objective= 0.3138838198746616  Ave training loss:  0.19844564630580777
Epoch  650 : Ave objective= 0.3135285819301993  Ave training loss:  0.1974881801236934
Epoch  700 : Ave objective= 0.3130506662261157  Ave training loss:  0.19732614789517317
Epoch  750 : Ave objective= 0.3117013712459582  Ave training loss:  0.19783101780306273
Epoch  800 : Ave objective= 0.31168129407553313  Ave training loss:  0.1983611483883822
Epoch  850 : Ave objective= 0.31194493882665475  Ave training loss:  0.1975972825831876
Epoch  900 : Ave objective= 0.3091803049094511  Ave training loss:  0.19803753368486882
Epoch  950 : Ave objective= 0.30730845754053315  Ave training loss:  0.20106822952961476
Epoch  1000 : Ave objective= 0.309648352854229  Ave training loss:  0.1988515657685185
Epoch  1050 : Ave objective= 0.30988723244639055  Ave training loss:  0.19805672982522401
Epoch  1100 : Ave objective= 0.3113556507248592  Ave training loss:  0.19895299706976444
Epoch  1150 : Ave objective= 0.31005154841183946  Ave training loss:  0.1979775922437964
Epoch  1200 : Ave objective= 0.3090916231752089  Ave training loss:  0.1995000126381679
Epoch  1250 : Ave objective= 0.3100790723657532  Ave training loss:  0.1986279496123064
Epoch  1300 : Ave objective= 0.3100129202032706  Ave training loss:  0.19810130047319854
Epoch  1350 : Ave objective= 0.30864372505852944  Ave training loss:  0.20018117630983412
Epoch  1400 : Ave objective= 0.30739788393397194  Ave training loss:  0.1991664830783109
Epoch  1450 : Ave objective= 0.30932908915768725  Ave training loss:  0.1986643511216483
Epoch  1500 : Ave objective= 0.31003686035328387  Ave training loss:  0.19950146408211517
Epoch  1550 : Ave objective= 0.3082133897248385  Ave training loss:  0.20080273488480033
Epoch  1600 : Ave objective= 0.30986572689496766  Ave training loss:  0.19871890220213367
Epoch  1650 : Ave objective= 0.3092978425000933  Ave training loss:  0.19843253742878036
Epoch  1700 : Ave objective= 0.30799576359962655  Ave training loss:  0.19977815901456666
Epoch  1750 : Ave objective= 0.3082377093984994  Ave training loss:  0.20104494011928875
Epoch  1800 : Ave objective= 0.30910853213014977  Ave training loss:  0.19924764482104265
Epoch  1850 : Ave objective= 0.3081311104395419  Ave training loss:  0.1990516125737984
Epoch  1900 : Ave objective= 0.3076586631286824  Ave training loss:  0.201669303892935
Epoch  1950 : Ave objective= 0.30603514269971993  Ave training loss:  0.20955740968713624
```

```
Epoch  0 : Ave objective= 0.6199700676565317  Ave training loss:  0.33779698856819307
Epoch  50 : Ave objective= 0.11640087158103882  Ave training loss:  0.14831928241549905
Epoch  100 : Ave objective= 0.09255635105225568  Ave training loss:  0.06511970822795728
Epoch  150 : Ave objective= 0.07341384555517372  Ave training loss:  0.06614559514424374
Epoch  200 : Ave objective= 0.052895571649770455  Ave training loss:  0.07425199570794411
Epoch  250 : Ave objective= 0.05335887862867526  Ave training loss:  0.0451404005220485
Epoch  300 : Ave objective= 0.04680622490251268  Ave training loss:  0.0361065398823835
Epoch  350 : Ave objective= 0.04685768710835255  Ave training loss:  0.03370235147177942
Epoch  400 : Ave objective= 0.043209080599909105  Ave training loss:  0.030626203721320043
Epoch  450 : Ave objective= 0.039207821860192386  Ave training loss:  0.027362762770559015
```



Prediction Functions
Action/Outcome Space
Input Space: [0,1)
Ridge with L2Reg=1
Ridge with L2Reg=0
Target Parameter Values (i.e. Bayes Optimal)
Training data
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0

4.1.1 The Affine Transformation

4.1.1

1) $\frac{\partial J}{\partial w_{ij}} = \sum_{r=1}^{m} \frac{\partial J}{\partial y_r} \cdot \frac{\partial y_r}{\partial w_{ij}} = \frac{\partial J}{\partial y_1} \cdot \frac{\partial y_1}{\partial w_{ij}} + \cdots + \frac{\partial J}{\partial y_m} \cdot \frac{\partial J_m}{\partial w_{ij}}.$

$\because y_i = \sum_{j=1}^{d} (w_{ij} x_j + b).$

$\therefore \frac{\partial y_{\cancel{i}}}{\partial w_{\cancel{i}j}} = \partial w_{ij} \left(\sum_{j=1}^{d} w_{ij} x_j + b \right) = x_j \delta_{ij} = x_j$

for $r \neq i$, $\frac{\partial y_r}{\partial w_{ij}} = 0$

$\therefore \frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_i} \cdot x_j$

2) $\frac{\partial J}{\partial W} \in R^{m \times d}$, $\frac{\partial J}{\partial y} \in R^{m \times 1}$, $x \in R^{d \times 1}$

$\because \frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_i} \cdot x_j$

$$\therefore \frac{\partial J}{\partial W} = \frac{\partial J}{\partial y} \times x^T = \begin{bmatrix} \frac{\partial J}{\partial y_1} x_1 & \cdots & \frac{\partial J}{\partial y_1} x_d \\ \vdots & \ddots & \vdots \\ \frac{\partial J}{y_m} x_1 & \cdots & \frac{\partial J}{\partial y_m} x_d \end{bmatrix}$$

3). from 1) & 2), we have $\frac{\partial J}{\partial x_i} = \sum_{r=1}^{m} \frac{\partial J}{\partial y_r} \cdot \frac{\partial y_r}{\partial x_i}$

$$\Rightarrow \frac{\partial J}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial J}{\partial y_i} w_{ij}$$

$$\Rightarrow \frac{\partial J}{\partial x} = w^T \left(\frac{\partial J}{\partial y}\right)$$

4). $\frac{\partial J}{\partial b} = \sum_{r=1}^{m} \frac{\partial J}{\partial y_r} \cdot \frac{\partial y_r}{\partial b} = \sum_{r=1}^{m} \frac{\partial J}{\partial y_r} = \frac{\partial J}{\partial y}$

4.1.2 Element-wise Transformers

4.1.2.

$\because$ S has the same dimension as A and $\sigma(A)$,

i.g $\in R^n$ indexed by a single variable

$\therefore$ $\frac{\partial J}{\partial S}$ and $\sigma'(A) = \frac{\partial S}{\partial A}$ are both $\in R^n$

$$\therefore \frac{\partial J}{\partial A_i} = \sum_r \frac{\partial J}{\partial S_r} \cdot \frac{\partial S_r}{\partial A_i} = \frac{\partial J}{\partial S_i} \cdot \sigma'(A_i) = \left(\frac{\partial J}{\partial S} \otimes \sigma'(A)\right)_i$$

$$\therefore \frac{\partial J}{\partial A} = \frac{\partial J}{\partial S} \otimes \sigma'(A)$$

4.2 MLP Implementation

4.2.1

```
class AffineNode(object):
    """Node implementing affine transformation (W,x,b)-->Wx+b, where W is a matrix,
    and x and b are vectors
```

```
        Parameters:
        W: node for which W.out is a numpy array of shape (m,d)
        x: node for which x.out is a numpy array of shape (d)
        b: node for which b.out is a numpy array of shape (m) (i.e. vector of length m
    """
    def __init__(self, W1, x, b, node_name):
        self.node_name = node_name
        self.out = None
        self.d_out = None
        self.W1 = W1
        self.x = x
        self.b = b

    def forward(self):
        self.out = np.dot(self.W1.out, self.x.out) + self.b.out
        self.d_out = np.zeros(self.out.shape)
        return self.out

    def backward(self):
        d_W1 = np.outer(self.d_out, self.x.out)
        d_x = np.dot(self.W1.out.T, self.d_out)
        d_b = self.d_out
        self.W1.d_out += d_W1
        self.x.d_out += d_x
        self.b.d_out += d_b
        return self.d_out

    def get_predecessors(self):
        return [self.W1, self.x, self.b]
    ## TODO
```

4.2.2

```
class TanhNode(object):
    """Node tanh(a), where tanh is applied elementwise to the array a
        Parameters:
        a: node for which a.out is a numpy array
    """
    def __init__(self, a, node_name):
        self.node_name = node_name
        self.out = None
        self.d_out = None
        self.a = a

    def forward(self):
        self.out = np.tanh(self.a.out)
        self.d_out = np.zeros(self.out.shape)
        return self.out

    def backward(self):
        d_a = (1 - np.tanh(self.a.out)**2)*self.d_out
        self.a.d_out += d_a
        return self.d_out

    def get_predecessors(self):
        return [self.a]
    ## TODO
```

4.2.3

```
class MLPRegression(BaseEstimator, RegressorMixin):
    """ MLP regression with computation graph """
    def __init__(self, num_hidden_units=10, step_size=.005, init_param_scale=0.01, ma
        self.num_hidden_units = num_hidden_units
        self.init_param_scale = 0.01
        self.max_num_epochs = max_num_epochs
        self.step_size = step_size

        # Build computation graph
        self.x = nodes.ValueNode(node_name="x") # to hold a vector input
        self.y = nodes.ValueNode(node_name="y") # to hold a scalar response
        self.W1 = nodes.ValueNode(node_name="W1") # to hold the parameter matrix
        self.w2 = nodes.ValueNode(node_name="w2")  # to hold the parameter vector
        self.b1 = nodes.ValueNode(node_name="b1") # to hold the bias parameter (vecto
        self.b2 = nodes.ValueNode(node_name="b2") # to hold the bias parameter (scala
        self.affine = nodes.AffineNode(W=self.W1, x=self.x, b=self.b1,
                                          node_name="affine")
        self.tanh = nodes.TanhNode(a=self.affine, node_name="tanh")
        self.prediction = nodes.VectorScalarAffineNode(x=self.tanh, w=self.w2, b=self.
                                                          node_name="prediction")
        self.objective = nodes.SquaredL2DistanceNode(a=self.prediction, b=self.y,
                                                   node_name="square_loss")

        # Group nodes into types to construct computation graph function
        self.inputs = [self.x]
        self.outcomes = [self.y]
        self.parameters = [self.W1, self.b1,self.w2, self.b2]

        self.graph = graph.ComputationGraphFunction(self.inputs, self.outcomes,
                                                      self.parameters, self.predic
                                                      self.objective)
        ## TODO

    def fit(self, X, y):
        num_instances, num_ftrs = X.shape
        y = y.reshape(-1)
        ## TODO: Initialize parameters (small random numbers -- not all 0, to break sy


        s = self.init_param_scale
        init_values = {"W1": np.random.standard_normal((self.num_hidden_units,
                                                      num_ftrs)),
                       "b1": np.random.standard_normal((self.num_hidden_units)),
                       "w2": np.random.standard_normal((self.num_hidden_units)),
                       "b2": np.array(np.random.randn()) }
        self.graph.set_parameters(init_values)
```

This code passed the test in *$mlp_regression.t.py$*

```
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. W is 1.36372974102e−08.
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. x is 2.17113106792e−09.
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. b is 1.63657896896e−09.
.DEBUG: (Node tanh) Max rel error for partial deriv w.r.t. a is 4.62053093836e−09.
.DEBUG: (Parameter W1) Max rel error for partial deriv 5.50516200215e−07.
DEBUG: (Parameter b1) Max rel error for partial deriv 4.41987841745e−08.
DEBUG: (Parameter w2) Max rel error for partial deriv 1.60618974379e−09.
DEBUG: (Parameter b2) Max rel error for partial deriv 5.83341433218e−10.
.
----------------------------------------------------------------------
Ran 3 tests in 2.784s

OK
```

For this parameter setting,

```
estimator = MLPRegression(num_hidden_units=10, step_size=0.001,
            init_param_scale=.0005,  max_num_epochs=5000)
x_train_as_column_vector = x_train.reshape(x_train.shape[0],1) # fit expects a 2−
x_as_column_vector = x.reshape(x.shape[0],1) # fit expects a 2−dim array
```

The average square error on the training set is about 0.26 (for all the epoch), the average square error of last epoch is 0.2

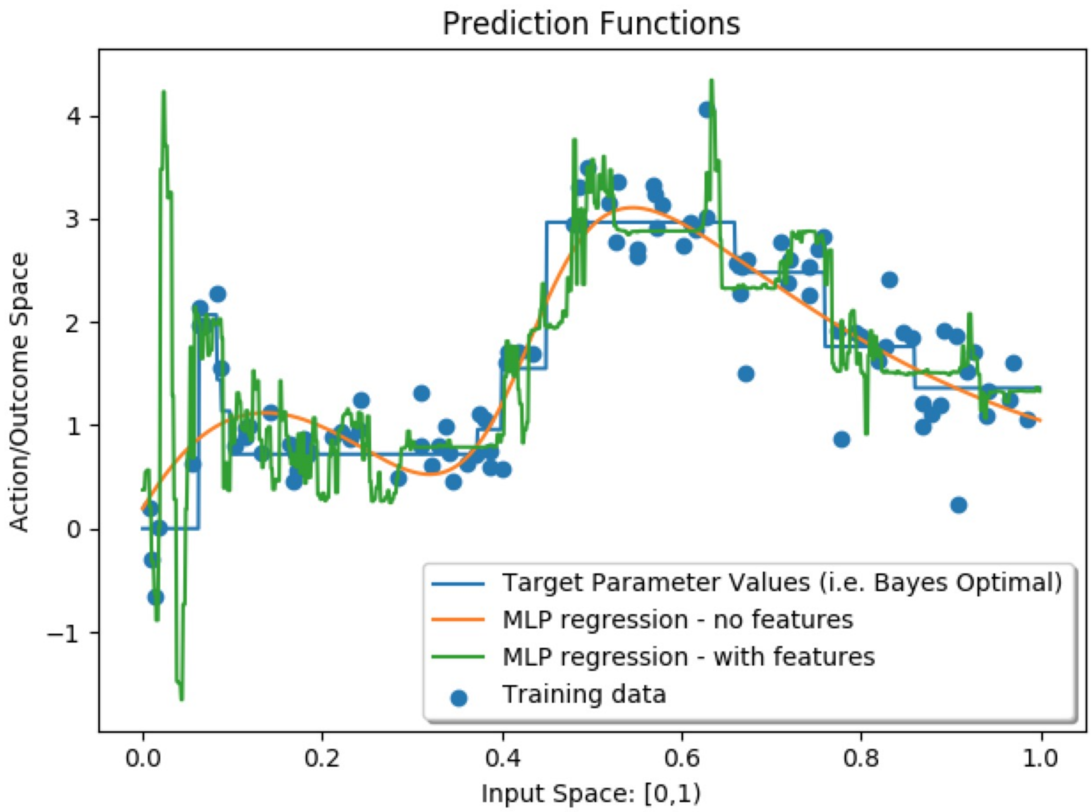
For this parameter setting,

```
estimator = MLPRegression(num_hidden_units=10, step_size=0.0005,
                init_param_scale=.01,  max_num_epochs=500)
```

The average square error on the training set is about 0.27 (for all the epoch), the average square error of last epoch is 0.1

```
Epoch  4350 : Ave objective= 0.21787052489735056  Ave training loss:  0.20907109703833326
Epoch  4400 : Ave objective= 0.21936335849224314  Ave training loss:  0.20873278536023135
Epoch  4450 : Ave objective= 0.21575739984251302  Ave training loss:  0.21247049899088913
Epoch  4500 : Ave objective= 0.2139164061811711  Ave training loss:  0.21209459618823587
Epoch  4550 : Ave objective= 0.21458928018879014  Ave training loss:  0.21147715727708602
Epoch  4600 : Ave objective= 0.21627786902628107  Ave training loss:  0.2078423849872406
Epoch  4650 : Ave objective= 0.21554682177911375  Ave training loss:  0.2083629754779958
Epoch  4700 : Ave objective= 0.21475840474596034  Ave training loss:  0.2083800324209057
Epoch  4750 : Ave objective= 0.2172372542413261  Ave training loss:  0.2071444139441335
Epoch  4800 : Ave objective= 0.21586805162174622  Ave training loss:  0.20793860500505307
Epoch  4850 : Ave objective= 0.21454826316702078  Ave training loss:  0.20673151269863546
Epoch  4900 : Ave objective= 0.2120277483003585  Ave training loss:  0.20942742504061906
Epoch  4950 : Ave objective= 0.21394041497110503  Ave training loss:  0.20756703284005767
Epoch  0 : Ave objective= 2.964237367097052  Ave training loss:  1.9152730770954447
Epoch  50 : Ave objective= 0.14993791042085985  Ave training loss:  0.14548445404902413
Epoch  100 : Ave objective= 0.12704231220538628  Ave training loss:  0.12299736526515767
Epoch  150 : Ave objective= 0.11822350013082848  Ave training loss:  0.11470003747086316
Epoch  200 : Ave objective= 0.11346753480401076  Ave training loss:  0.11038226829966623
Epoch  250 : Ave objective= 0.11033756909596533  Ave training loss:  0.10739508923090915
Epoch  300 : Ave objective= 0.1078779028018294  Ave training loss:  0.10511600404501621
Epoch  350 : Ave objective= 0.10625527177459411  Ave training loss:  0.10337852824848724
Epoch  400 : Ave objective= 0.10458943739804395  Ave training loss:  0.10216547317716809
Epoch  450 : Ave objective= 0.10343198220784604  Ave training loss:  0.10095378967562234

Process finished with exit code 0
```



Prediction Functions
Action/Outcome Space
Input Space: [0,1)
Target Parameter Values (i.e. Bayes Optimal)
MLP regression - no features
MLP regression - with features
Training data
4
3
2
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0