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 numpy scalar array (e.g. np.array(.01)) (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 = np.array(l2_reg)
        self.w = w

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

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

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

Unit Test see below

2

```
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.b = b
        self.a = a

    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 * 1
```

```
        d_b = self.d_out * 1
        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]
```

Unit Test see below

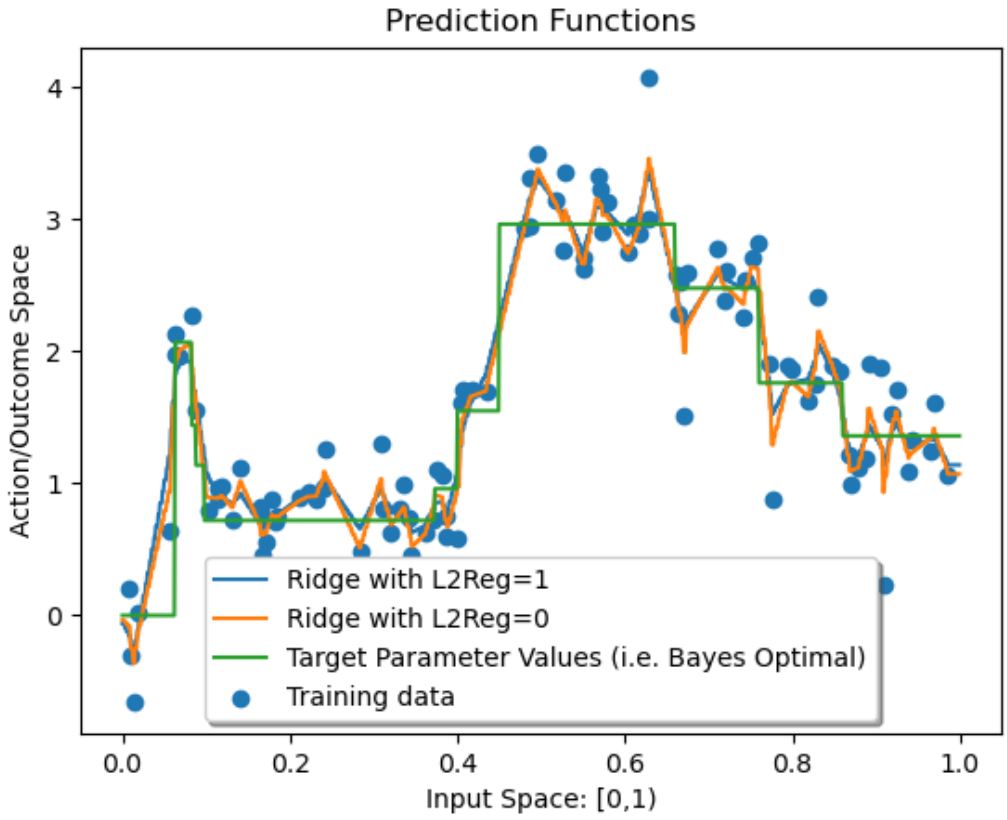
3

```
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.prediction = nodes.VectorScalarAffineNode(x=self.x, w=self.w, b=self.b,
                                                       node_name="prediction")
        # Build computation graph
        # TODO: ADD YOUR CODE HERE
        self.objective = nodes.SquaredL2DistanceNode(a=self.prediction, b=self.y,
                                                         node_name="L2 loss")

        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)
```

Unit Test see below

Unit Test for Question 1,2,3

```
[Running] python -u "d:\Course\#ML\homework\7\code\ridge_regression.t.py"
DEBUG: (Node l2 norm node) Max rel error for partial deriv w.r.t. w is 1.1751955047398415e-09.
.DEBUG: (Node sum node) Max rel error for partial deriv w.r.t. a is 5.263558723993663e-10.
DEBUG: (Node sum node) Max rel error for partial deriv w.r.t. b is 5.263558723993663e-10.
.DEBUG: (Parameter w) Max rel error for partial deriv 5.303010461562305e-09.
DEBUG: (Parameter b) Max rel error for partial deriv 1.0065136343921627e-09.
.
----------------------------------------------------------------------
Ran 3 tests in 0.005s

OK
```




```
[Running] python -u "d:\Course\#ML\homework\7\code\ridge_regression.py"
Epoch  0 : Ave objective= 1.5825705729591257  Ave training loss:  0.8024225617137162
Epoch  50 : Ave objective= 0.2148492038711289  Ave training loss:  0.21077377376442016
Epoch  100 : Ave objective= 0.17298259685376746  Ave training loss:  0.1695915171144192
Epoch  150 : Ave objective= 0.15461778161061596  Ave training loss:  0.15167824724015
Epoch  200 : Ave objective= 0.1425583971020568  Ave training loss:  0.14014755217635028
Epoch  250 : Ave objective= 0.1341295812560543  Ave training loss:  0.13158965394393946
Epoch  300 : Ave objective= 0.12634464437014079  Ave training loss:
 0.12487123112971928
Epoch  350 : Ave objective= 0.12143860468316593  Ave training loss:  0.1192466591680856
Epoch  400 : Ave objective= 0.11681151595596971  Ave training loss:
 0.11439888119186498
Epoch  450 : Ave objective= 0.11225194558373479  Ave training loss:
 0.11039262748511908
Epoch  500 : Ave objective= 0.10907877923706956  Ave training loss:
 0.10667479320533521
Epoch  550 : Ave objective= 0.10559564678202175  Ave training loss:
 0.10346136712228127
Epoch  600 : Ave objective= 0.10231439868947864  Ave training loss:
 0.10054909737837975
Epoch  650 : Ave objective= 0.10007131342876016  Ave training loss:
 0.09794583438770504
Epoch  700 : Ave objective= 0.09718584869009136  Ave training loss:
 0.09581640723784775
Epoch  750 : Ave objective= 0.09522464530108164  Ave training loss:
 0.09360943932395768
Epoch  800 : Ave objective= 0.09321120506295559  Ave training loss:
 0.09139809265730556
Epoch  850 : Ave objective= 0.0913928301609255  Ave training loss:  0.08938173327691443
Epoch  900 : Ave objective= 0.08958542898590277  Ave training loss:
 0.08767731827665708
Epoch  950 : Ave objective= 0.08788717707755257  Ave training loss:
 0.08598428386721885
```

```
Epoch  1000 : Ave objective= 0.08518897344065848  Ave training loss:
 0.08491378811660788
Epoch  1050 : Ave objective= 0.08452784726726374  Ave training loss:
 0.08300957676539894
Epoch  1100 : Ave objective= 0.0833885599515267  Ave training loss:
 0.08154590032876952
Epoch  1150 : Ave objective= 0.0816552556299833  Ave training loss:
 0.08034275066833504
Epoch  1200 : Ave objective= 0.08041019074647822  Ave training loss:
 0.07901101161922897
Epoch  1250 : Ave objective= 0.07941586337356  Ave training loss:  0.07779595035275688
Epoch  1300 : Ave objective= 0.07737871807753507  Ave training loss:
 0.07699453203154799
Epoch  1350 : Ave objective= 0.07723552623873872  Ave training loss:
 0.07553812633442882
Epoch  1400 : Ave objective= 0.07610276785621219  Ave training loss:
 0.07468091384766985
Epoch  1450 : Ave objective= 0.07485198498344664  Ave training loss:
 0.07348860112854573
Epoch  1500 : Ave objective= 0.07414990794587761  Ave training loss:
 0.07248169113581018
Epoch  1550 : Ave objective= 0.07312947956573462  Ave training loss:
 0.07157380473149892
Epoch  1600 : Ave objective= 0.07223403119894542  Ave training loss:
 0.07068777093292077
Epoch  1650 : Ave objective= 0.07133127531229101  Ave training loss:
 0.0697880368608974
Epoch  1700 : Ave objective= 0.06977204935883621  Ave training loss:
 0.0693184043922332
Epoch  1750 : Ave objective= 0.06964508138928145  Ave training loss:
 0.06807979671276168
Epoch  1800 : Ave objective= 0.06870174975578046  Ave training loss:
 0.06746736799575728
Epoch  1850 : Ave objective= 0.06800389812223157  Ave training loss:
 0.06650585351167915
Epoch  1900 : Ave objective= 0.06708688682997964  Ave training loss:
 0.0659808793345207
Epoch  1950 : Ave objective= 0.06618199706409576  Ave training loss:
 0.06563415721933727
Epoch  0 : Ave objective= 0.6453317937882683  Ave training loss:  0.5165006688633932
Epoch  50 : Ave objective= 0.12495942192859377  Ave training loss:  0.1414416855610545
Epoch  100 : Ave objective= 0.09574667819473183  Ave training loss:
 0.09529703828423973
Epoch  150 : Ave objective= 0.07804087262471639  Ave training loss:
 0.07304152243508831
Epoch  200 : Ave objective= 0.07573871160031333  Ave training loss:
 0.06554035459779539
Epoch  250 : Ave objective= 0.06982578999539163  Ave training loss:
 0.07001190455878063
Epoch  300 : Ave objective= 0.06023777894668922  Ave training loss:
 0.06410720189777566
Epoch  350 : Ave objective= 0.05599361383167548  Ave training loss:
 0.05764102725793286
Epoch  400 : Ave objective= 0.05379944910214952  Ave training loss:
 0.04537965416599831
Epoch  450 : Ave objective= 0.05115114492778493  Ave training loss:
 0.04530960958749052
```

4

$$\frac{\partial J}{\partial W_{ij}} = \sum_{j=0}^{m} \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial W_{ij}}$$

Because y_i are independent with all w except w_i, so just need to calculate the derivative then r=i

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial y_i} x_j$$

5

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial y} \otimes x$$

6

$$\frac{\partial J}{\partial X} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial X} = \frac{\partial J}{\partial y} W = W^T \frac{\partial J}{\partial y}$$

7

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial b} = \frac{\partial J}{\partial y} * I = \frac{\partial J}{\partial y}$$

8

$$s = \sigma(A)$$

$$\frac{\partial J}{\partial A} = \frac{\partial J}{\partial S} \frac{\partial S}{\partial A} = \frac{\partial J}{\partial s} \odot \sigma'(A)$$

9

```
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, W, x, b, node_name):
        self.node_name = node_name
        self.out = None
        self.d_out = None
        self.W = W
        self.x = x
        self.b = b

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

    def backward(self):
        d_W = np.outer(self.d_out, self.x.out)
        d_x = self.d_out @ self.W.out
        d_b = self.d_out * 1
        self.W.d_out += d_W
        self.x.d_out += d_x
        self.b.d_out += d_b
        return self.d_out
```

Unit Test see below

10

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

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

Unit Test see below

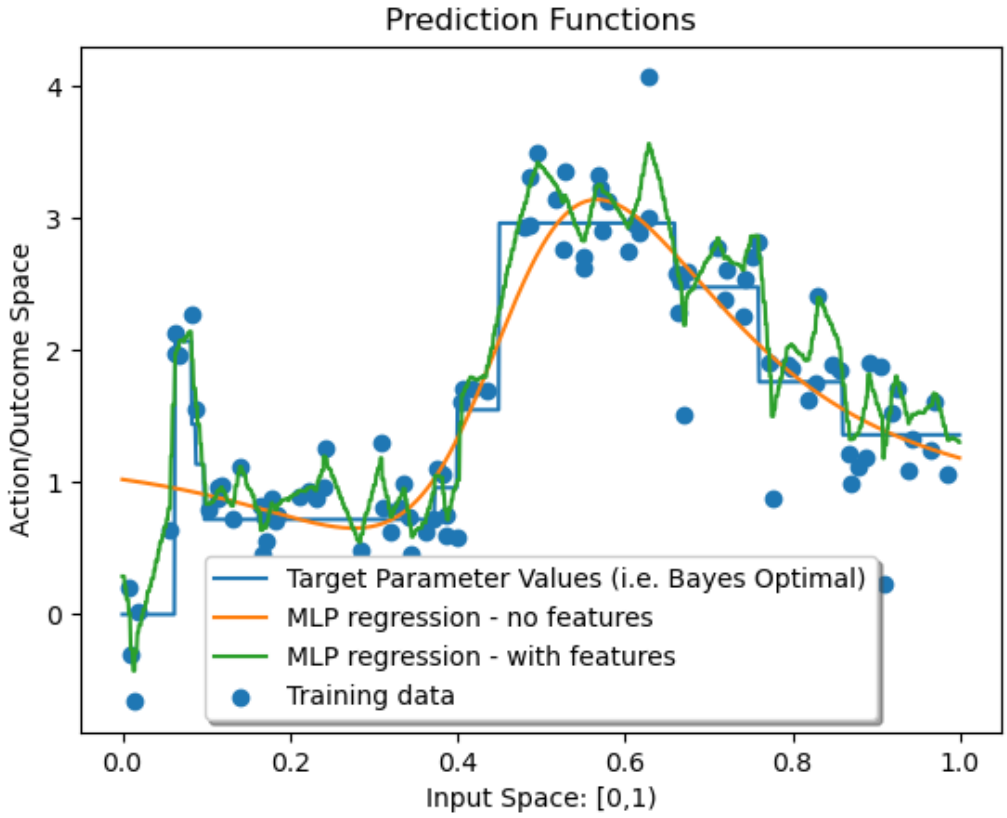
11

```
class MLPRegression(BaseEstimator, RegressorMixin):
    """ MLP regression with computation graph """
    def __init__(self, num_hidden_units=10, step_size=.005, init_param_scale=0.01,
max_num_epochs = 5000):
        self.num_hidden_units = num_hidden_units
        self.init_param_scale = init_param_scale
        self.max_num_epochs = max_num_epochs
        self.step_size = step_size

        # Build computation graph
        # TODO: ADD YOUR CODE HERE
        self.x = nodes.ValueNode(node_name="x")
        self.y = nodes.ValueNode(node_name="y")
        self.W1 = nodes.ValueNode(node_name="W1")
        self.b1 = nodes.ValueNode(node_name="b1")
        self.w2 = nodes.ValueNode(node_name="w2")
        self.b2 = nodes.ValueNode(node_name="b2")

        self.Affain = nodes.AffineNode(x=self.x, W=self.W1,
b=self.b1,node_name="Affain")
        self.Tanh = nodes.TanhNode(a=self.Affain, node_name='Tanh')

        self.prediction = nodes.VectorScalarAffineNode(x=self.Tanh, w=self.w2,
b=self.b2,
                                                       node_name="prediction")
        self.objective = nodes.SquaredL2DistanceNode(a=self.prediction, b=self.y,
                                                     node_name="square loss")
        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.prediction,
                                                    self.objective)
```

Unit Test see below

Unit Test for 9, 10, 11

```
[Running] python -u "d:\Course\#ML\homework\7\code\mlp_regression.t.py"
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. W is 8.524639491444214e-07.
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. x is 2.308585391635347e-08.
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. b is 2.8043132720613354e-09.
.DEBUG: (Node tanh) Max rel error for partial deriv w.r.t. a is 2.874181183736627e-09.
.DEBUG: (Parameter W1) Max rel error for partial deriv 9.149586437139255e-07.
DEBUG: (Parameter b1) Max rel error for partial deriv 7.53514421466467e-08.
DEBUG: (Parameter w2) Max rel error for partial deriv 1.558725197389069e-09.
DEBUG: (Parameter b2) Max rel error for partial deriv 2.4269205125765466e-10.
.
----------------------------------------------------------------------
Ran 3 tests in 0.005s

OK

[Done] exited with code=0 in 1.342 seconds
```




```
[Running] python -u "d:\Course\#ML\homework\7\code\mlp_regression.py"
Epoch  0 : Ave objective= 3.136846171898233  Ave training loss:  2.719898311972865
Epoch  50 : Ave objective= 0.9453680402912678  Ave training loss:  0.9435156692482133
Epoch  100 : Ave objective= 0.945151936869993  Ave training loss:  0.943257714699222
Epoch  150 : Ave objective= 0.9425285787381934  Ave training loss:  0.940573247488134
Epoch  200 : Ave objective= 0.919445272449191  Ave training loss:  0.9168193121952405
Epoch  250 : Ave objective= 0.831529396010504  Ave training loss:  0.8274429280255401
Epoch  300 : Ave objective= 0.7774303378911465  Ave training loss:  0.7731236889174569
Epoch  350 : Ave objective= 0.7709444240848475  Ave training loss:  0.7670126139188689
Epoch  400 : Ave objective= 0.7680576744117285  Ave training loss:  0.7641158982439559
Epoch  450 : Ave objective= 0.7641257264418292  Ave training loss:  0.7604064054066157
Epoch  500 : Ave objective= 0.7594579384337768  Ave training loss:  0.7553455082453985
Epoch  550 : Ave objective= 0.7517158856961411  Ave training loss:  0.7494124327459929
Epoch  600 : Ave objective= 0.7454188137725143  Ave training loss:  0.7414094804272081
Epoch  650 : Ave objective= 0.7359127844216634  Ave training loss:  0.7325984396159931
Epoch  700 : Ave objective= 0.7264372684723779  Ave training loss:  0.7227715260061666
Epoch  750 : Ave objective= 0.7163311804109695  Ave training loss:  0.7126112049182364
```

```
Epoch  800 : Ave objective= 0.7062705353912864  Ave training loss:  0.7026719099977146
Epoch  850 : Ave objective= 0.6971767395585592  Ave training loss:  0.693534147753201
Epoch  900 : Ave objective= 0.688976958205506  Ave training loss:  0.685373572206589
Epoch  950 : Ave objective= 0.6816937768598487  Ave training loss:  0.6781089288482696
Epoch  1000 : Ave objective= 0.6750316477838918  Ave training loss:  0.6712701804093374
Epoch  1050 : Ave objective= 0.668278681990197  Ave training loss:  0.6643704154945128
Epoch  1100 : Ave objective= 0.660727020926526  Ave training loss:  0.6568783948848994
Epoch  1150 : Ave objective= 0.6520934066312195  Ave training loss:  0.6480839071036388
Epoch  1200 : Ave objective= 0.6422262452390598  Ave training loss:  0.6373214804660405
Epoch  1250 : Ave objective= 0.6291721526550346  Ave training loss:  0.6237926638458177
Epoch  1300 : Ave objective= 0.6128118767144588  Ave training loss:  0.6068525625458874
Epoch  1350 : Ave objective= 0.5923201960980147  Ave training loss:  0.5863390710591534
Epoch  1400 : Ave objective= 0.568383847055414  Ave training loss:  0.5616768624584357
Epoch  1450 : Ave objective= 0.5411591819630538  Ave training loss:  0.53344264670602
Epoch  1500 : Ave objective= 0.5113649110583662  Ave training loss:  0.5028726675118378
Epoch  1550 : Ave objective= 0.47783563715318794  Ave training loss:
 0.47125994447460573
Epoch  1600 : Ave objective= 0.4449667730896336  Ave training loss:  0.4399601156141568
Epoch  1650 : Ave objective= 0.4174010804518715  Ave training loss:  0.4107598220835195
Epoch  1700 : Ave objective= 0.3937114505284942  Ave training loss:
 0.38614147491692635
Epoch  1750 : Ave objective= 0.37154497238881795  Ave training loss:
 0.3674968583792322
Epoch  1800 : Ave objective= 0.3561185310824453  Ave training loss:  0.3520797915690234
Epoch  1850 : Ave objective= 0.3451463369644937  Ave training loss:
 0.34399988691472294
Epoch  1900 : Ave objective= 0.33905246975329917  Ave training loss:
 0.33132219444383915
Epoch  1950 : Ave objective= 0.33245862679043525  Ave training loss:
 0.3268285876554865
Epoch  2000 : Ave objective= 0.3277838848447325  Ave training loss:  0.3219214084391727
Epoch  2050 : Ave objective= 0.32623880478000106  Ave training loss:
 0.31868263115299306
Epoch  2100 : Ave objective= 0.3248926226757816  Ave training loss:
 0.31728002167064495
Epoch  2150 : Ave objective= 0.3210922703432869  Ave training loss:
 0.31450366475644015
Epoch  2200 : Ave objective= 0.3183583799292167  Ave training loss:
 0.31564515404634536
Epoch  2250 : Ave objective= 0.31863455887350356  Ave training loss:
 0.3113802856626813
Epoch  2300 : Ave objective= 0.31606535938144814  Ave training loss:
 0.30986579111214074
Epoch  2350 : Ave objective= 0.31526628183692695  Ave training loss:
 0.3084971598745917
Epoch  2400 : Ave objective= 0.31379073650423667  Ave training loss:
 0.30702694596115554
Epoch  2450 : Ave objective= 0.3113765135990029  Ave training loss:  0.3055666986330642
Epoch  2500 : Ave objective= 0.3104515785855117  Ave training loss:  0.3044109323802026
Epoch  2550 : Ave objective= 0.30805466641382734  Ave training loss:  0.302821888788908
Epoch  2600 : Ave objective= 0.30669041618005966  Ave training loss:
 0.3022379751679445
Epoch  2650 : Ave objective= 0.3054116399339222  Ave training loss:
 0.30136133928919334
Epoch  2700 : Ave objective= 0.3049702628860416  Ave training loss:  0.2989984414240244
Epoch  2750 : Ave objective= 0.3041443929350164  Ave training loss:  0.2975589618990875
Epoch  2800 : Ave objective= 0.30130996322969095  Ave training loss:  0.296613272827002
Epoch  2850 : Ave objective= 0.299842229575435  Ave training loss:  0.2954557324385052
Epoch  2900 : Ave objective= 0.2994599943283578  Ave training loss:
 0.29354554620289536
```

```
Epoch  2950 : Ave objective= 0.2977921309777637  Ave training loss:
 0.29226183068539713
Epoch  3000 : Ave objective= 0.2964878241220137  Ave training loss:  0.2909558836300424
Epoch  3050 : Ave objective= 0.2956954713720157  Ave training loss:
 0.29167224711846984
Epoch  3100 : Ave objective= 0.2937842121904275  Ave training loss:  0.2883822618181658
Epoch  3150 : Ave objective= 0.2924412978198799  Ave training loss:
 0.28717441188634807
Epoch  3200 : Ave objective= 0.2901346065838191  Ave training loss:  0.2862091626586251
Epoch  3250 : Ave objective= 0.28923078758506177  Ave training loss:
 0.2846534195978823
Epoch  3300 : Ave objective= 0.2877871804744727  Ave training loss:
 0.28372574635347386
Epoch  3350 : Ave objective= 0.2869550956006552  Ave training loss:
 0.28211001940147723
Epoch  3400 : Ave objective= 0.2855070243909588  Ave training loss:
 0.28090231218895345
Epoch  3450 : Ave objective= 0.2850403001681256  Ave training loss:  0.2797281492225523
Epoch  3500 : Ave objective= 0.28295660864648975  Ave training loss:
 0.2789990891535447
Epoch  3550 : Ave objective= 0.28184076138204284  Ave training loss:
 0.2775741972459157
Epoch  3600 : Ave objective= 0.28084065013923915  Ave training loss:
 0.27642253153908913
Epoch  3650 : Ave objective= 0.28002714556314184  Ave training loss:  0.275131710083088
Epoch  3700 : Ave objective= 0.2787983923013218  Ave training loss:  0.2741609270329542
Epoch  3750 : Ave objective= 0.27724346596073945  Ave training loss:
 0.27281381688412926
Epoch  3800 : Ave objective= 0.2759375755273795  Ave training loss:
 0.27195841093561984
Epoch  3850 : Ave objective= 0.2748537785232761  Ave training loss:  0.2709512878061043
Epoch  3900 : Ave objective= 0.27346233197850983  Ave training loss:
 0.26962860417991297
Epoch  3950 : Ave objective= 0.2731051294222141  Ave training loss:  0.2684964333321585
Epoch  4000 : Ave objective= 0.2719709394943402  Ave training loss:  0.2674664499671715
Epoch  4050 : Ave objective= 0.2707771153803641  Ave training loss:  0.2664812914164701
Epoch  4100 : Ave objective= 0.2697075804631854  Ave training loss:
 0.26545881645855324
Epoch  4150 : Ave objective= 0.2687511588027972  Ave training loss:  0.2645281472752918
Epoch  4200 : Ave objective= 0.26809612263739113  Ave training loss:
 0.2635696726027314
Epoch  4250 : Ave objective= 0.2655623450261927  Ave training loss:  0.2638212423270669
Epoch  4300 : Ave objective= 0.26613701837957054  Ave training loss:
 0.2616240158431987
Epoch  4350 : Ave objective= 0.26478008773185013  Ave training loss:
 0.26079246944473694
Epoch  4400 : Ave objective= 0.263930345396576  Ave training loss:  0.2598176794965304
Epoch  4450 : Ave objective= 0.2633957359402349  Ave training loss:
 0.25892933935382895
Epoch  4500 : Ave objective= 0.2611211443479854  Ave training loss:  0.2585537077742218
Epoch  4550 : Ave objective= 0.26153445412259535  Ave training loss:
 0.2572602896504342
Epoch  4600 : Ave objective= 0.26038350805877586  Ave training loss:  0.256585470829604
Epoch  4650 : Ave objective= 0.2589116223544358  Ave training loss:
 0.25582674376853826
Epoch  4700 : Ave objective= 0.2590944310270491  Ave training loss:  0.2547273257519485
Epoch  4750 : Ave objective= 0.25764993046767654  Ave training loss:
 0.2543186900778664
Epoch  4800 : Ave objective= 0.2576490932994192  Ave training loss:  0.2531202736381201
Epoch  4850 : Ave objective= 0.2566944654312681  Ave training loss:  0.2523489412398516
```

```
Epoch  4900 : Ave objective= 0.25582481695725146  Ave training loss:
 0.2517399075268638
Epoch  4950 : Ave objective= 0.2551318187574174  Ave training loss:  0.2508917133431668
Epoch  0 : Ave objective= 3.2192439822104864  Ave training loss:  2.7240344950050854
Epoch  50 : Ave objective= 0.15012256382136116  Ave training loss:  0.14478889186398688
Epoch  100 : Ave objective= 0.11728306804461448  Ave training loss:
 0.10745305083456783
Epoch  150 : Ave objective= 0.0994112043412914  Ave training loss:  0.09044357728683718
Epoch  200 : Ave objective= 0.08839923102798966  Ave training loss:
 0.07580120926670599
Epoch  250 : Ave objective= 0.07448039092293773  Ave training loss:  0.0674630176503851
Epoch  300 : Ave objective= 0.0704905066762013  Ave training loss:  0.05700972339038473
Epoch  350 : Ave objective= 0.060348773209052346  Ave training loss:
 0.061336711227895455
Epoch  400 : Ave objective= 0.05348730400470994  Ave training loss:
 0.044491038003358165
Epoch  450 : Ave objective= 0.04588650191048771  Ave training loss:
 0.04034381563097155

[Done] exited with code=0 in 99.683 seconds
```

12

```python
class SoftmaxNode(object):
    """ Softmax node
        Parameters:
        z: node for which z.out is a numpy array
    """

    def __init__(self, z, node_name):
        self.node_name = node_name
        self.out = None
        self.d_out = None
        self.z = z

    def forward(self):
        z_exp = np.exp(self.z.out)
        self.out = z_exp / np.sum(z_exp)
        self.d_out = np.zeros(self.out.shape)
        return self.out

    def backward(self):
        d = np.empty([len(self.z.out), len(self.out)])
        for i in range(len(self.z.out)):
            for j in range(len(self.out)):
                d[i][j] = self.out[i] * \
                    (1-self.out[i]) if i == j else -self.out[i] * self.out[j]

        d_z = np.dot(self.d_out, d)
        self.z.d_out += d_z
        return self.d_out

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

Unit Test see below

13

```
class NLLNode(object):
    """ Node computing NLL loss between 2 arrays.
        Parameters:
        y_hat: a node that contains all predictions
        y_true: a node that contains all labels
    """

    def __init__(self, y_hat, y_true, node_name):
        self.node_name = node_name
        self.out = None
        self.d_out = None
        self.y_hat = y_hat
        self.y_true = y_true

    def forward(self):
        self.out = -np.log(self.y_hat.out[self.y_true.out])
        self.d_out = np.zeros(self.out.shape)
        return self.out

    def backward(self):
        d_true = self.d_out * \
            (-self.y_hat.out[self.y_true.out]/np.exp(-self.out))
        d_hat = np.zeros(self.y_hat.out.shape)
        d_hat[self.y_true.out] = 1
        d_hat = self.d_out * (-d_hat/np.exp(-self.out))
        self.y_hat.d_out += d_hat
        self.y_true.d_out += d_true
        return self.d_out

    def get_predecessors(self):
        return [self.y_hat, self.y_true]
```

Unit Test see below

14

```
class MulticlassClassifier(BaseEstimator, RegressorMixin):
    """ Multiclass prediction """
    def __init__(self, num_hidden_units=10, step_size=.005, init_param_scale=0.01,
max_num_epochs = 1000, num_class=3):
        self.num_hidden_units = num_hidden_units
        self.init_param_scale = init_param_scale
        self.max_num_epochs = max_num_epochs
        self.step_size = step_size
        self.num_class = num_class

        # Build computation graph
        # TODO: add your code here
        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")
        self.b1 = nodes.ValueNode(node_name="b1")
        self.w2 = nodes.ValueNode(node_name="W2")
        self.b2 = nodes.ValueNode(node_name="b2")
```

```
        self.L = nodes.AffineNode(W=self.W1, x=self.x, b=self.b1, node_name="L")
        self.h = nodes.TanhNode(a=self.L, node_name="h")
        self.z = nodes.AffineNode(W=self.W2, x=self.h, b=self.b2, node_name="z")

        self.prediction = nodes.SoftmaxNode(z=self.z, node_name="prediction")
        self.objective = nodes.NLLNode(y_hat=self.prediction, y_true=self.y,
node_name='objective')

        self.graph = graph.ComputationGraphFunction(inputs=[self.x],
                                                    outcomes=[self.y],
                                                    parameters=
[self.W1,self.b1,self.W2,self.b2],
                                                    prediction=self.prediction,
objective=self.objective)
```

Unit Test for Question 12, 13, 14

```
[Running] python -u "d:\Course\#ML\homework\7\code\multiclass.t.py"
DEBUG: (Node softmax) Max rel error for partial deriv w.r.t. z is 1.811596538559175e-08.
.DEBUG: (Parameter W1) Max rel error for partial deriv 6.187925179224009e-06.
DEBUG: (Parameter b1) Max rel error for partial deriv 1.4251925203840912e-06.
DEBUG: (Parameter W2) Max rel error for partial deriv 3.985308280512233e-09.
DEBUG: (Parameter b2) Max rel error for partial deriv 2.70768686979549e-09.
.
----------------------------------------------------------------------
Ran 2 tests in 0.007s

OK

[Done] exited with code=0 in 1.797 seconds
```

```
[Running] python -u "d:\Course\#ML\homework\7\code\multiclass.py"
Epoch  0  Ave training loss:  0.10767753468425852
Epoch  50  Ave training loss:  0.0037402729498018867
Epoch  100  Ave training loss:  0.0019509875089186069
Epoch  150  Ave training loss:  0.00131892201003299
Epoch  200  Ave training loss:  0.0009947600104512845
Epoch  250  Ave training loss:  0.0007975221227264001
Epoch  300  Ave training loss:  0.0006649220947379017
Epoch  350  Ave training loss:  0.0005697138957458585
Epoch  400  Ave training loss:  0.0004980771960410213
Epoch  450  Ave training loss:  0.00044225221211177576
Epoch  500  Ave training loss:  0.0003975450315101259
Epoch  550  Ave training loss:  0.0003609495175393885
Epoch  600  Ave training loss:  0.00033045202244361534
Epoch  650  Ave training loss:  0.0003046529432352649
Epoch  700  Ave training loss:  0.0002825495526238341
Epoch  750  Ave training loss:  0.0002634040479431621313
Epoch  800  Ave training loss:  0.0002466648603036149
Epoch  850  Ave training loss:  0.0002319056839595025 8
Epoch  900  Ave training loss:  0.0002187970217752761
Epoch  950  Ave training loss:  0.0002070780161184417 3
Test set accuracy = 1.000

[Done] exited with code=0 in 28.447 seconds
```