Problem 1

Codes:

```
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 * np.dot(self.w.out,self.w.out)
        self.d_out = np.zeros(self.out.shape)
        return self.out

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

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

Test Results:

```
$ python ridge_regression.t.py TestAll.test_L2NormPenaltyNode
DEBUG: (Node l2 norm node) Max rel error for partial deriv w.r.t. w is 8.1359246
93149988e-09.
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Problem 2

Codes:

```
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
        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]
```

Test Results:

```
$ python ridge_regression.t.py TestAll.test_SumNode
DEBUG: (Node sum node) Max rel error for partial deriv w.r.t. a is 5.26355772778
9586e-10.
DEBUG: (Node sum node) Max rel error for partial deriv w.r.t. b is 2.87556162292
49054e-11.
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Problem 3

Codes:

```python
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.prediction = nodes.VectorScalarAffineNode(x=self.x, w=self.w, b=self.b,
                                                       node_name="prediction")
        self.loss = nodes.SquaredL2DistanceNode(a=self.prediction, b=self.y,
                                                       node_name="square loss")
        self.L2Norm = nodes.L2NormPenaltyNode(l2_reg=self.l2_reg, w=self.w,
                                                       node_name="l2 penalty")
        self.objective = nodes.SumNode(a=self.loss, b=self.L2Norm,
                                                       node_name="penalized sq loss")

        # 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)
```

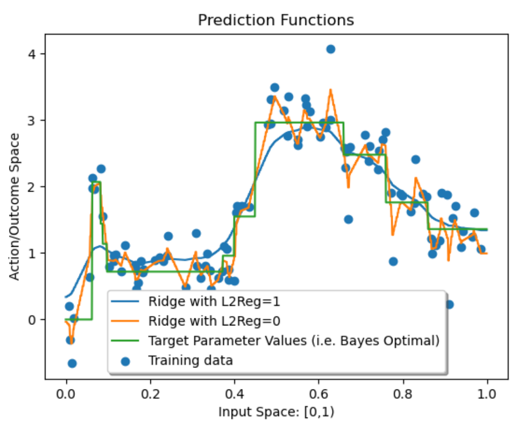
Test Results:

Setting 1: l2reg = 1, step_size=0.00005, max_num_epochs=2000

Average Square Error = **0.20121047965190475**

Setting 2: l2reg = 1, step_size=0.00005, max_num_epochs=500

Average Square Error = **0. 0443814303794584**



Problem 4

We have $\frac{\partial J}{\partial W_{ij}} = \sum_{r=1}^{m} \frac{\partial J}{\partial y_r} \frac{\partial y_r}{\partial W_{ij}}$

Since $\frac{\partial y_r}{\partial W_{ij}} = \begin{cases} x_j & r = i \\ 0 & \text{else} \end{cases}$

$\frac{\partial J}{\partial W_{ij}} = \sum_{r=1}^{m} \frac{\partial J}{\partial y_r} \frac{\partial y_r}{\partial W_{ij}} = \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial W_{ij}} = \frac{\partial J}{\partial y_i} x_j$

Problem 5

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial y_i} x_j \Longrightarrow \frac{\partial J}{\partial W} = \frac{\partial J}{\partial y} \bigotimes x$$

Problem 6

For the i-th entry of $\frac{\partial J}{\partial x}$, $\frac{\partial J}{\partial x_i} = \sum_{r=1}^{m} \frac{\partial J}{\partial y_r} \frac{\partial y_r}{\partial x_i} = \sum_{r=1}^{m} \frac{\partial J}{\partial y_r} W_{ir}$

Therefore, $\frac{\partial J}{\partial x} = W^T \left(\frac{\partial J}{\partial y} \right)$

Problem 7

By chain rule, $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y} \cdot \frac{\partial J}{\partial b}$

Since $\frac{\partial J}{\partial b} = \frac{Wx+b}{\partial b} = I$, $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y} \cdot I = \frac{\partial J}{\partial y}$

Problem 8

$$
\begin{aligned}
\frac{\partial J}{\partial A_i} &= \frac{\partial J}{\partial S_i} \cdot \frac{\partial S_i}{\partial A_i} \\
&= \frac{\partial J}{\partial [\sigma(A)]_i} \cdot \left(\frac{\partial \sigma(A)}{\partial A} \right)_i \\
&= \left(\frac{\partial J}{\partial S} \right)_i \cdot (\sigma'(A))_i \\
&= \left(\frac{\partial J}{\partial S} \odot \sigma(A) \right)_i
\end{aligned}
$$

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

Problem 9

Code:

```python
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.x = x
        self.W = W
        self.b = b

    def forward(self):
        self.out = np.dot(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 = np.dot(self.W.out.T, self.d_out)
        d_b = self.d_out
        self.x.d_out += d_x
        self.W.d_out += d_W
        self.b.d_out += d_b
        return self.d_out

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

Test Results:

```
$ python mlp_regression.t.py TestNodes.test_AffineNode
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. x is 4.2282111645235
95e-09.
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. W is 1.5732493885104
37e-08.
DEBUG: (Node affine) Max rel error for partial deriv w.r.t. b is 1.6365788561328
23e-09.
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
```

Problem 10

Codes:

```
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
    """
    """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-self.out**2)
        self.a.d_out += d_a
        return self.d_out


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

Test Results:

```
$ python mlp_regression.t.py TestNodes.test_TanhNode
DEBUG: (Node tanh) Max rel error for partial deriv w.r.t. a is 1.418068972210059
5e-08.
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Problem 11

Codes:

```
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") # 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.w2 = nodes.ValueNode(node_name="w2")
        self.b1 = nodes.ValueNode(node_name="b1")
        self.b2 = nodes.ValueNode(node_name="b2")

        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.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)
```

Test Results:

Setting 1: step_size=0.001, init_param_scale=.0005, max_num_epochs=5000

Average Square Error = **0.26107056770910664**

Setting 2: step_size=0.0005, init_param_scale=.01, max_num_epochs=500

Average Square Error = **0.0429245017966997**