Neural Network Homework 5 (BackPropogation)

By:
Monil Shah
mds747

1) Epoch = 100000 , learning rate= 0.2, weights range = [-0.5,0.5]

In [32]:
```
import numpy as np
def tanh(x):
    return np.tanh(x)
def tanh_prime(x):
    return 1.0 - x**2
```

In [33]:
```
class NeuralNetwork:
    def __init__(self, layers, activation='tanh'):
            self.activation = tanh
            self.activation_prime = tanh_prime
            self.weights = []
            self.count = 0
            for i in range(1, len(layers) - 1):
                r = np.random.uniform(-0.5,0.5,(layers[i-1] + 1, layers[i] +
                self.weights.append(r)
            r = np.random.uniform(-0.5,0.5, (layers[i] + 1, layers[i+1]))
            self.weights.append(r)
    def fit(self, X, y, learning_rate=0.02, epochs=387):
        termination=0
        counter=4
        ones = np.atleast_2d(np.ones(X.shape[0]))
        X = np.concatenate((ones.T, X), axis=1)
        for k in range(epochs):
            i = np.random.randint(X.shape[0])
            a = [X[i]]
            for l in range(len(self.weights)):
                    dot_value = np.dot(a[l], self.weights[l])
                    activation = self.activation(dot_value)
                    a.append(activation)
            error = y[i] - a[-1]
            deltas = [error * self.activation_prime(a[-1])]
            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)*self.activat
            deltas.reverse()
            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)
            if k%4 == 0:
                termination = 0
            termination = termination+(error*error);
            print(termination)

            if k % 387 == 0:
                print('epochs:', k)
                print('error:',error)



    def predict(self, x):
        a = np.hstack((np.ones(1).T, np.array(x)))
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a
```

In [34]:
```
if __name__ == '__main__':
    nn = NeuralNetwork([2,4,1])
    X = np.array([[-1, -1],
                  [-1, 1],
                  [1, -1],
                  [1, 1]])
    y = np.array([-1, 1, 1, -1])
    nn.fit(X, y)
    print("==========================================================")
    print("Results:")
    for e in X:
        print(e,nn.predict(e))
```

```
[ 0.79872693]
epochs: 0
error: [-0.89371524]
[ 1.89099467]
[ 2.37781703]
[ 3.13914192]
[ 1.77995845]
[ 2.26068342]
[ 2.72334385]
[ 3.84496438]
[ 1.04147979]
[ 1.81458948]
[ 3.63915626]
[ 5.33263131]
[ 0.80205228]
[ 2.40431046]
[ 3.19729936]
[ 4.36224802]
[ 1.5687656]
```

2) Epoch = 10000 , learning rate= 0.2, weights range = [-0.5,0.5]

In [ ]:
```
class NeuralNetwork:

    def __init__(self, layers, activation='tanh'):
            self.activation = tanh
            self.activation_prime = tanh_prime


            self.weights = []

            for i in range(1, len(layers) - 1):
                r = np.random.uniform(-0.5,0.5,(layers[i-1] + 1, layers[i] +
                self.weights.append(r)

            r = np.random.uniform(-0.5,0.5,(layers[i] + 1, layers[i+1]))
            self.weights.append(r)
```

In [ ]:
```
def fit(self, X, y, learning_rate=0.02, epochs=10000):
    ones = np.atleast_2d(np.ones(X.shape[0]))
    X = np.concatenate((ones.T, X), axis=1)
    for k in range(epochs):
        i = np.random.randint(X.shape[0])
        a = [X[i]]
        for l in range(len(self.weights)):
            dot_value = np.dot(a[l], self.weights[l])
            activation = self.activation(dot_value)
            a.append(activation)
        error = y[i] - a[-1]
        deltas = [error * self.activation_prime(a[-1])]
        for l in range(len(a) - 2, 0, -1):
            deltas.append(deltas[-1].dot(self.weights[l].T)*self.activation_
        deltas.reverse()
        for i in range(len(self.weights)):
            layer = np.atleast_2d(a[i])
            delta = np.atleast_2d(deltas[i])
            self.weights[i] += learning_rate * layer.T.dot(delta)

        if k % 1000 == 0:
            print('epochs:', k)
            print('error:',error)
```

In [ ]:
```
 def predict(self, x):
        a = np.hstack((np.ones(1).T, np.array(x)))
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a
```

In [ ]:
```
if __name__ == '__main__':

    nn = NeuralNetwork([2,2,1])

    X = np.array([[-1, -1],
                  [-1, 1],
                  [1, -1],
                  [1, 1]])

    y = np.array([-1, 1, 1, -1])

    nn.fit(X, y)
    print("==========================================================")
    print("Results:")
    for e in X:

        print(e,nn.predict(e))
```

3) Epoch = 1000 , learning rate= 0.2, weights range = [-0.5,0.5]

In [ ]:

```
class NeuralNetwork:

    def __init__(self, layers, activation='tanh'):
            self.activation = tanh
            self.activation_prime = tanh_prime


            self.weights = []

            for i in range(1, len(layers) - 1):
                r = np.random.uniform(-0.5,0.5,(layers[i-1] + 1, layers[i] +
                self.weights.append(r)

            r = np.random.uniform(-0.5,0.5, (layers[i] + 1, layers[i+1]))
            self.weights.append(r)


    def fit(self, X, y, learning_rate=0.02, epochs=500):
        ones = np.atleast_2d(np.ones(X.shape[0]))
        X = np.concatenate((ones.T, X), axis=1)
        for k in range(epochs):
            i = np.random.randint(X.shape[0])
            a = [X[i]]
            for l in range(len(self.weights)):
                dot_value = np.dot(a[l], self.weights[l])
                activation = self.activation(dot_value)
                a.append(activation)
            error = y[i] - a[-1]
            deltas = [error * self.activation_prime(a[-1])]
            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)*self.activat
            deltas.reverse()
            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)

            if k % 50 == 0:
                print('epochs:', k)
                print('error:',error)


    def predict(self, x):
        a = np.hstack((np.ones(1).T, np.array(x)))
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a
if __name__ == '__main__':
    nn = NeuralNetwork([2,2,1])
    X = np.array([[-1, -1],
                  [-1, 1],
                  [1, -1],
                  [1, 1]])
    y = np.array([-1, 1, 1, -1])
    nn.fit(X, y)
    print("=======================================================")
```

```
    print("Results:")
    for e in X:
        print(e,nn.predict(e))
```

In [ ]:
```
def fit(self, X, y, learning_rate=0.02, epochs=1000):
    ones = np.atleast_2d(np.ones(X.shape[0]))
    X = np.concatenate((ones.T, X), axis=1)
    for k in range(epochs):
        i = np.random.randint(X.shape[0])
        a = [X[i]]
        for l in range(len(self.weights)):
            dot_value = np.dot(a[l], self.weights[l])
            activation = self.activation(dot_value)
            a.append(activation)
        error = y[i] - a[-1]
        deltas = [error * self.activation_prime(a[-1])]
        for l in range(len(a) - 2, 0, -1):
            deltas.append(deltas[-1].dot(self.weights[l].T)*self.activation_
        deltas.reverse()
        for i in range(len(self.weights)):
            layer = np.atleast_2d(a[i])
            delta = np.atleast_2d(deltas[i])
            self.weights[i] += learning_rate * layer.T.dot(delta)

        if k % 100 == 0:
            print('epochs:', k)
            print('error:',error)
```

In [ ]:
```
  def predict(self, x):
        a = np.hstack((np.ones(1).T, np.array(x)))
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a

if __name__ == '__main__':

    nn = NeuralNetwork([2,2,1])

    X = np.array([[-1, -1],
                  [-1, 1],
                  [1, -1],
                  [1, 1]])

    y = np.array([-1, 1, 1, -1])

    nn.fit(X, y)
    print("==========================================================")
    print("Results:")
    for e in X:

        print(e,nn.predict(e))
```

4) Epoch = 500 , learning rate= 0.2, weights range

= [-0.5,0.5]

In [ ]:

```
class NeuralNetwork:

    def __init__(self, layers, activation='tanh'):
            self.activation = tanh
            self.activation_prime = tanh_prime


            self.weights = []

            for i in range(1, len(layers) - 1):
                r = np.random.uniform(-0.5,0.5,(layers[i-1] + 1, layers[i] +
                self.weights.append(r)

            r = np.random.uniform(-0.5,0.5, (layers[i] + 1, layers[i+1]))
            self.weights.append(r)


    def fit(self, X, y, learning_rate=0.02, epochs=500):
        ones = np.atleast_2d(np.ones(X.shape[0]))
        X = np.concatenate((ones.T, X), axis=1)
        for k in range(epochs):
            i = np.random.randint(X.shape[0])
            a = [X[i]]
            for l in range(len(self.weights)):
                dot_value = np.dot(a[l], self.weights[l])
                activation = self.activation(dot_value)
                a.append(activation)
            error = y[i] - a[-1]
            deltas = [error * self.activation_prime(a[-1])]
            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)*self.activat
            deltas.reverse()
            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)

            if k % 50 == 0:
                print('epochs:', k)
                print('error:',error)


    def predict(self, x):
        a = np.hstack((np.ones(1).T, np.array(x)))
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a
if __name__ == '__main__':
    nn = NeuralNetwork([2,2,1])
    X = np.array([[-1, -1],
                  [-1, 1],
                  [1, -1],
                  [1, 1]])
    y = np.array([-1, 1, 1, -1])
    nn.fit(X, y)
    print("=======================================================")
```

```
    print("Results:")
    for e in X:
        print(e,nn.predict(e))
```

Comments:

As per multiple iterations run, it is commentable that the increase in error is inversely proportional to the number of epochs. More epochs, less error.
Also results go far from expected , when the number of epochs is decreased.

In [ ]: