utils

September 1, 2019

In [1]:
```
import numpy as np


        def sigmoid(z):

                return 1./(1. + np.exp(-z))


        def loss(Y_out, Y):

                return -(1. / Y.shape[1]) * np.sum(np.multiply(Y, np.log(Y_out)))


        def feed_forward(X, model):

            '''
            Z = W1.dot(x) + b1
            H = sigmoid(Z)
            U = C.dot(H) + b2
            R = softmax(U)
            '''
            cache = {}

            cache["Z"] = np.matmul(model["W"], X) + model["b1"][:, None]
            cache["H"] = sigmoid(cache["Z"])
            cache["U"] = np.matmul(model["C"], cache["H"]) + model["b2"][:, None]
            cache["R"] = np.exp(cache["U"]) / np.sum(np.exp(cache["U"]), axis=0)

            return cache



        def back_propagate(X, Y, model, cache):

                '''
            db2 = cache[R] - Y
                dC = db2.dot(H.T)
                delta = C.T .dot(db2)
```

```
        db1 = delta.multiply (sig_prim (cache[Z]))
        dW = db1.dot(X.T)

    '''
        # batch_size = X.shape[1]

        batch_size = 1

        db2 =cache["R"] - Y
        dC = np.matmul(db2, cache["H"].T)
        delta = np.matmul(model["C"].T, db2)
        db1 = delta * sigmoid(cache["Z"]) * (1 - sigmoid(cache["Z"]))
        dW = np.matmul(db1, X.T)
        db1 = (1. / batch_size) * np.sum(db1, axis=1)
        db2 = (1. / batch_size) * np.sum(db2, axis=1)

        grads = {"b1": db1, "b2": db2, "W": dW, "C": dC}

        return grads
```

In [ ]: