Batch Training with Dynamic Networks

```
import numpy as np

def train(weight, learn, bias, target, initial, data, epoch):
    actual_output = 0
    error = []
    len_kargs = 0
    flag = False
    for j in range(epoch):
        accumulate_weight = np.array([0.0, 0.0])
        initial_value=initial
        for i in range(len(data)):

            p = np.array([[data[i]], [initial_value]])
            actual_output = (weight.dot(p))+bias
            error = target[i]-actual_output
            del_weight = np.transpose(learn*error*p)
            accumulate_sum=accumulate_weight+ del_weight
            accumulate_weight=accumulate_sum
            initial_value =data[i]
        weight = accumulate_weight
        print(f"{j+1} Epoch weight  = {weight}")

if __name__ == "__main__":
    pi = 1
    p = [2, 3, 4]
    weight = np.array([0, 0])
    target = [3, 5, 6]
    train(weight=weight, learn=0.02, bias=0,
          target=target, initial=pi, data=p, epoch=1)
```

Batch Training with Static Networks

```
import numpy as np

def train(weight, learn, target,epoch, **kargs):
    actual_output = 0
    error = 0
    weight_change = 0
    len_kargs = 0
    bias = 0

    for k in range(epoch):
        bias_accumulator=0
        weight_accumulator=0
        for i in kargs:
            len_kargs = len(kargs[i])
            for j in range(len_kargs):
                bias_accumulator += learn*error
                actual_output = (weight.dot(kargs[i][j]))+bias
                error = target[j]-actual_output[0][0]
                weight_change = np.transpose(learn*error*kargs[i][j])
                weight_accumulator+=weight_change
        weight=weight_accumulator
        bias=bias_accumulator
        print(f"weight = {weight},bias = {bias} ")

if __name__ == "__main__":
    p1 = np.array([[1], [2]])
    p2 = np.array([[2], [1]])
    p3 = np.array([[2], [3]])
    p4 = np.array([[3], [1]])
    weight = np.array([[0, 0]])
    target = [4, 5, 7, 7]
    train(weight=weight, learn=0.01, target=target, kargs=[p1, p2, p3, p4],epoch=1)
```

Concurrent Inputs in a Dynamic Network

```
import numpy as np

def net(*args):
    weight = [[1,2]]
    bias= 0
    lst1 = []
    lst2 = []
    lst3 = []
    for i in args:
        lst1.append([i[0]])
        lst2.append([i[1]])
    for i in range(len(lst1)):
        temp =[]
        if i==0:
            temp.append(lst1[i])
            temp.append([0])
            lst3.append(temp.copy())
            temp.clear()
            temp.append(lst2[i])
            temp.append([0])
            lst3.append(temp.copy())
            temp.clear()
        else:
            temp.append(lst1[i])
            temp.append(lst1[i-1])
            lst3.append(temp.copy())
            temp.clear()
            temp.append(lst2[i])
            temp.append(lst2[i-1])
            lst3.append(temp.copy())
            temp.clear()
    result=[]

    for i in lst3:
        result.append(matrix_multiplication(weight,i))


    count=0
    final = []
    temp =[]
    for i in range(len(result)):
        count+=1
        if count==1:
            temp.append(result[i][0][0]+bias)
        if count==2:
            count=0
```

```
            temp.append(result[i][0][0]+bias)
            final.append(temp.copy())
            temp.clear()
    print(final)


def matrix_multiplication(A,B):
    rowA =len(A)
    colA = len(A[0])
    rowB = len(B)
    colB = len(B[0])
    if colA == rowB:
        result = []
        for i in range(rowA):
            temp =[]
            for j in range(colB):
                temp.append(0)
            result.append(temp.copy())
            temp.clear()


        for i in range(rowA):
            for j in range(colB):
                for k in range(rowB):
                    result[i][j] += A[i][k] * B[k][j]
        return result

    else:
        return "Not Possible"

if __name__ == "__main__":

    p1 = [1, 4]
    p2 = [2, 3]
    p3 = [3, 2]
    p4 = [4, 1]
    net(p1,p2,p3,p4)
```

Incremental Training of Static Networks

```
import numpy as np

def train(weight, learn, target, **kargs):
    actual_output = 0
    error = 0
    weight_change = 0
    len_kargs = 0
    bias = 0

    for i in kargs:
        len_kargs = len(kargs[i])
        for j in range(len_kargs):
            actual_output = (weight.dot(kargs[i][j]))+bias
            error = target[j]-actual_output[0][0]
            print("Actual Output=",actual_output,"Error = ",error )

            weight_change = np.transpose(learn*error*kargs[i][j])
            weight=weight+weight_change
            bias += learn*error


if __name__ == "__main__":
    p1 = np.array([[1], [2]])
    p2 = np.array([[2], [1]])
    p3 = np.array([[2], [3]])
    p4 = np.array([[3], [1]])
    weight = np.array([[0, 0]])
    target = [4, 5, 7, 7]
    train(weight=weight, learn=0.1, target=target, kargs=[p1, p2, p3, p4])
```

Incremental Training with Dynamic Networks

```
import numpy as np

def train(weight, learn, bias, target, initial, data):
    actual_output = 0
    error = []
    len_kargs = 0
    flag = False
    for i in range(len(data)):
        p = np.array([[data[i]], [initial]])
        actual_output = (weight.dot(p))+bias
        error = target[i]-actual_output[0]
        del_weight = np.transpose(learn*error*p)
        weight = del_weight+weight
        print(f"actual output = {actual_output[0]},error = {error} ")
            initial=data[i]

if __name__ == "__main__":
    pi = 1
    p = [2, 3, 4]
    bias = 0
    weight = np.array([0, 0])
    target = [3, 5, 7]
    train(weight=weight, learn=0.1, bias=bias,
          target=target, initial=pi, data=p)
```

Sequential Inputs in a Dynamic Network

```
import numpy as np

def sequentialInputsDynamicNetwork(weight,bias,**kargs):
    sum = []
    for i in kargs:
        for j in range(len(kargs[i])):
            if j==0:
                kargs[i][j] = np.append(kargs[i][j],0)
                sum.append(weight.dot(kargs[i][j])+bias)

            else:
                kargs[i][j]  = np.append(kargs[i][j],kargs[i][j-1][0])
                sum.append([weight.dot(kargs[i][j])+bias])


    print(sum)

if __name__ == "__main__":

    bias = 0
    w = np.array([[ 1,2]] )
    p1 = np.array([1])
    p2 = np.array([2])
    p3 = np.array([3])
    p4 = np.array([4])

    sequentialInputsDynamicNetwork(weight=w,bias=bias,kargs=[p1,p2,p3,p4])
```