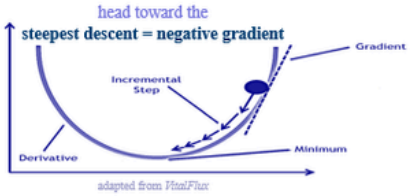
Stochastic Gradient Function

Accelerating the Computation

The package currently uses gradient-based optimization (L-BFGS-B) to perform GCP.

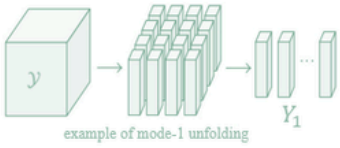


Theorem [3]: The gradients for the GCP objective are as follows:

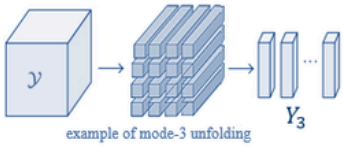
MTTKRP

Matricized Tensor Times Khatri – Rao Product

$$\frac{\partial F}{\partial U_k} = \underbrace{Y_k}_{\text{Matricized/mode } k \text{ unlfoding of } \mathcal{Y} \text{ that holds the element-wise gradients: } \frac{\partial f}{\partial m_i}(x_i, m_i)} \underbrace{U_n \odot \cdots \odot U_{k+1} \odot U_{k-1} \odot \cdots \odot U_1}_{= Z_k \text{ the Khatri-Rao Product (column-wise Kronecker product)}}$$



example of mode-1 unfolding



example of mode-3 unfolding

Problem: The dense MTTKRP in the gradient is **computationally burdensome**, especially when the data is very large!

Solution: Replace full gradients with stochastic gradients obtained by randomly sampling subsets of the data tensor,

structured sparsity due to the random sampling, leading to an efficient sparse MTTKRP

$$\frac{\partial \tilde{F}}{\partial U_k} = \tilde{Y}_k Z_k$$

then use stochastic gradient optimization (Adam).

```
1 using GCPDecompositions.TensorKernels:mttkrp
```

```
1 using GCPDecompositions, LinearAlgebra
```

The Function

```
stoc_grad (generic function with 1 method)
```

```
function stoc_grad(X::Array{},M::CPD,s::Int64,loss)
    w = length(X)
    Y = zeros(size(X))

    for c in 1:s

        multi_index = [rand(1:size(X,k)) for k in 1:ndims(X)]

        m = getindex(M,multi_index...)
        x = X[multi_index...]

        Y[multi_index...] += ((w/s)*(GCPLosses.deriv(loss,x,m)))

    end
    G = [mttkrp(Y,M.U,k) for k in 1:ndims(X)]
return G
end

```

This function takes a tensor, its decomposition, the number of samples to take (user-specified), and the loss function used. It computes the gradients (partial derivatives) of the loss function with respect to the factor matrices

Usage and Example

```
X =
88×60×40 Array{Int64, 3}:
[:, :, 1] =
 73    4  59  68   6  54  36  35  19  90  …  38  95  25  61  92  22   60  84  22  49
 79    4   5  19  26  23  39  27  61   8      23  80  89  71  55  41   67  77  25  28
 55  100  66  70  22  28  60  82   2  30       9  21  63  32  28  10    1   3  44  10
  6   45  68  43   4  26  26   3   7  87      47  42  20  58  21  57   73  31  69  53
 55   56  41  20  63  28  46  87  56  50      69  77  10  44  44  44   44  63  33  25
 25   52  16  59   3  30   6  14  66  82  …  93  70  23  79  89  99   50  84  58  18
 98   85   3  65   7  86  79  58  81   7      62  14  78  11  44  81  100   1  45  64
  ⋮                    ⋮                  ⋱   ⋮                    ⋮
 24   33  58  95  37  57  68  49  74  26      46  87  33  19  68  21   76  85  19  88
 16   27  41  29  51  55   7  94  44   6      89  96  13  34  74  33   50   5  68  32
 21   51  53   8  96  93   2  29  49  46      41  92  92  22  73  82   76  66  80  97
 85   30  93  86  91  53   2  39  73  41  …  83  45  39  38  55  97   20   2  58  42
 17   26   6  61  37  85  70  62   9  20      53  44  25  96  48  97   11  82  69  29
 11    9  97  26  79  93  38  45   8  84      90  96  49  37  95  91   54  42   2  44

[:, :, 2] =
 17  32   51  86  28  63  98  59  43  10  …  37  85  35   4  31  29   48  80  96  74
 98  66   97   3  64  40  72   8  69  70      46  94  29  82  87  23   29  70  73  30
 10  13   77  29  65  92  60  87  92  23      47   3  18   7  63  94   80  21   9  69
 18  64   48  64  66  51  95  32  22  90      21  95   5  76  56   9   22  32  78  63
 24  73   52   2   7  79  63  58  89  45      39  30  67  71  50  10   42  85  51  82
 75  65   78  75  68  37   2  56  44  91  …  57  91  31  37  65  56   49  85  55  67
 54   8   85  96  91  93  91   1   2  16      26  46   8  76  72  53  100  54  28   2
  ⋮                    ⋮                  ⋱   ⋮                    ⋮
 48  20   55  89  41  55  12  41  13  74      84  64  77  30  84  93   78  53  27  65
 68  31   48  93  45  64  72  86   1  70      13  84  35  33  46  77   73  71  41  20
 93  91   59  92   1  98  54  16  38   3      54  35  99  18  36  99    6  71  63  77
 58  20    7  57  15   6  17  17  52  93  …  14  27  93  33  11  37   98  51  62  46
 44  66  100  33  98  78   5   9  48  71      55  77  65  81  62  38   97  95  69  37
 65  92   77  37  39  99  87  24  27  69      56  80  25  92  27   6   84  51  94  56
```

```
1 X = rand(1:100,88,60,40) # random tensor
```

```
M = 88×60×40 CPD{Float64, 3, Vector{Float64}, Matrix{Float64}} with 3 components
    λ weights:
    3-element Vector{Float64}: …
    U[1] factor matrix:
    88×3 Matrix{Float64}: …
    U[2] factor matrix:
    60×3 Matrix{Float64}: …
    U[3] factor matrix:
    40×3 Matrix{Float64}: …
```

```
1 M = gcp(X,3) # rank 3 decomposition
```

```
Symbol[
    1:  :Gamma
    2:  :Huber
    3:  :LeastSquares
    4:  :NegativeBinomialOdds
    5:  :NonnegativeLeastSquares
    6:  :Poisson
    7:  :PoissonLog
    8:  :Rayleigh
]
```

```
1 names(GCPDecompositions.GCPLosses, all = true)[22:29] # common loss functions
```

```
Matrix{Float64}[
    1:  88×3 Matrix{Float64}:
         -2129.05   -1589.26     1274.79
           374.866    892.824   -1271.69
          -384.402   -364.93      815.28
          -797.562    157.78    -1916.34
          2243.35    2525.62      151.021
             0.0        0.0         0.0
          -602.891   -427.13      -43.4523
             ⋮
          -437.23     -82.7044    126.605
         -3199.29   -2023.98    -1559.43
          -243.993   -736.299     502.742
           339.312    834.4     -1075.08
           192.163    741.117    -340.375
          1010.16   -1021.21      90.4616
    2:  60×3 Matrix{Float64}:
         -1398.81   -1628.25      48.6438
         -1025.39   -1019.9      416.079
          -450.601   -803.531   1593.65
          -521.053   -831.444    761.255
         -2261.13   -1880.37     556.429
         -1584.51   -1827.83    1210.05
         -2319.13   -1511.63    1434.87
             ⋮
          -744.325   -761.158   2564.18
          -951.189   -613.81     336.226
         -1952.73   -1848.79    1258.51
          -379.555   -367.823   2615.27
         -2727.04   -3320.0     1703.88
          -118.267    -57.2831   225.08
    3:  40×3 Matrix{Float64}:
          1423.7      1963.74     1819.43
           343.723     877.552     682.881
          -837.441   -1408.53    -1222.16
           853.596     709.936     902.533
         -1108.78      -20.7763   -424.21
            28.0353   -427.953    -226.547
           873.44     1073.74     1091.6
             ⋮
         -1304.93     -513.266    -977.012
           673.2      1606.08     1316.47
           290.698    2711.29     1781.39
          -110.711    1349.14      710.821
           139.947    -115.663      23.1855
          1590.15     1712.23     1839.54
]
```

```
1 stoc_grad(X,M,400,GCPLosses.LeastSquares())
```

Functionality

Substituting computing full gradients with this stochastic approach improves the user experience by decreasing run time and making large/dense tensors manageable, as these calculations are faster and more efficient.