package main

import (

"fmt"

"math/rand"

u "./utils"

)

type RBM struct {

N int

n\_visible int

n\_hidden int

W [][]float64

hbias []float64

vbias []float64

}

func RBM\_\_construct(this \*RBM, N int, n\_visible int, n\_hidden int, W [][]float64, hbias []float64, vbias []float64) {

a := 1.0 / float64(n\_visible)

this.N = N

this.n\_visible = n\_visible

this.n\_hidden = n\_hidden

if W == nil {

this.W = make([][]float64, n\_hidden)

for i := 0; i < n\_hidden; i++ { this.W[i] = make([]float64, n\_visible) }

for i := 0; i < n\_hidden; i++ {

for j := 0; j < n\_visible; j++ {

this.W[i][j] = u.Uniform(-a, a)

}

}

} else {

this.W = W

}

if hbias == nil {

this.hbias = make([]float64, n\_hidden)

} else {

this.hbias = hbias

}

if vbias == nil {

this.vbias = make([]float64, n\_visible)

} else {

this.vbias = vbias

}

}

func RBM\_contrastive\_divergence(this \*RBM, input []int, lr float64, k int) {

ph\_mean := make([]float64, this.n\_hidden)

ph\_sample := make([]int, this.n\_hidden)

nv\_means := make([]float64, this.n\_visible)

nv\_samples := make([]int, this.n\_visible)

nh\_means := make([]float64, this.n\_hidden)

nh\_samples := make([]int, this.n\_hidden)

/\* CD-k \*/

RBM\_sample\_h\_given\_v(this, input, ph\_mean, ph\_sample)

for step := 0; step < k; step++ {

if step == 0 {

RBM\_gibbs\_hvh(this, ph\_sample, nv\_means, nv\_samples, nh\_means, nh\_samples)

} else {

RBM\_gibbs\_hvh(this, nh\_samples, nv\_means, nv\_samples, nh\_means, nh\_samples)

}

}

for i := 0; i < this.n\_hidden; i++ {

for j := 0; j < this.n\_visible; j++ {

this.W[i][j] += lr \* (ph\_mean[i] \* float64(input[j]) - nh\_means[i] \* float64(nv\_samples[j])) / float64(this.N)

}

this.hbias[i] += lr \* (float64(ph\_sample[i]) - nh\_means[i]) / float64(this.N)

}

for i := 0; i < this.n\_visible; i++ {

this.vbias[i] += lr \* float64(input[i] - nv\_samples[i]) / float64(this.N)

}

}

func RBM\_sample\_h\_given\_v(this \*RBM, v0\_sample []int, mean []float64, sample []int) {

for i := 0; i < this.n\_hidden; i++ {

mean[i] = RBM\_propup(this, v0\_sample, this.W[i], this.hbias[i])

sample[i] = u.Binomial(1, mean[i])

}

}

func RBM\_sample\_v\_given\_h(this \*RBM, h0\_sample []int, mean []float64, sample []int) {

for i := 0; i < this.n\_visible; i++ {

mean[i] = RBM\_propdown(this, h0\_sample, i, this.vbias[i])

sample[i] = u.Binomial(1, mean[i])

}

}

func RBM\_propup(this \*RBM, v []int, w []float64, b float64) float64 {

pre\_sigmoid\_activation := 0.0

for j := 0; j < this.n\_visible; j++ {

pre\_sigmoid\_activation += w[j] \* float64(v[j])

}

pre\_sigmoid\_activation += b

return u.Sigmoid(pre\_sigmoid\_activation)

}

func RBM\_propdown(this \*RBM, h []int, i int, b float64) float64 {

pre\_sigmoid\_activation := 0.0

for j := 0; j < this.n\_hidden; j++ {

pre\_sigmoid\_activation += this.W[j][i] \* float64(h[j])

}

pre\_sigmoid\_activation += b

return u.Sigmoid(pre\_sigmoid\_activation)

}

func RBM\_gibbs\_hvh(this \*RBM, h0\_sample []int, nv\_means []float64, nv\_samples []int, nh\_means []float64, nh\_samples []int) {

RBM\_sample\_v\_given\_h(this, h0\_sample, nv\_means, nv\_samples)

RBM\_sample\_h\_given\_v(this, nv\_samples, nh\_means, nh\_samples)

}

func RBM\_reconstruct(this \*RBM, v []int, reconstructed\_v []float64) {

h := make([]float64, this.n\_hidden)

var pre\_sigmoid\_activation float64

for i := 0; i < this.n\_hidden; i++ {

h[i] = RBM\_propup(this, v, this.W[i], this.hbias[i])

}

for i := 0; i < this.n\_visible; i++ {

pre\_sigmoid\_activation = 0.0

for j := 0; j < this.n\_hidden; j++ {

pre\_sigmoid\_activation += this.W[j][i] \* h[j]

}

pre\_sigmoid\_activation += this.vbias[i]

reconstructed\_v[i] = u.Sigmoid(pre\_sigmoid\_activation)

}

}

func test\_rbm() {

rand.Seed(0)

learning\_rate := 0.1

training\_epochs := 1000

k := 1

train\_N := 6

test\_N := 2

n\_visible := 6

n\_hidden := 3

// training data

train\_X := [][]int {

{1, 1, 1, 0, 0, 0},

{1, 0, 1, 0, 0, 0},

{1, 1, 1, 0, 0, 0},

{0, 0, 1, 1, 1, 0},

{0, 0, 1, 0, 1, 0},

{0, 0, 1, 1, 1, 0},

}

// construct RBM

var rbm RBM

RBM\_\_construct(&rbm, train\_N, n\_visible, n\_hidden, nil, nil, nil)

// train

for epoch := 0; epoch < training\_epochs; epoch++ {

for i := 0; i < train\_N; i++ {

RBM\_contrastive\_divergence(&rbm, train\_X[i], learning\_rate, k)

}

}

// test data

test\_X := [][]int {

{1, 1, 0, 0, 0, 0},

{0, 0, 0, 1, 1, 0},

}

reconstructed\_X := make([][]float64, test\_N)

for i := 0; i < test\_N; i++ { reconstructed\_X[i] = make([]float64, n\_visible)}

// test

for i := 0; i < test\_N; i++ {

RBM\_reconstruct(&rbm, test\_X[i], reconstructed\_X[i])

for j := 0; j < n\_visible; j++ {

fmt.Printf("%.5f ", reconstructed\_X[i][j])

}

fmt.Printf("\n")

}

}

func main() {

test\_rbm()

}