Kisitli Boltzmann Makinalari (Restricted Boltzmann Machines -RBM-)

RBM aynen BM orneginde oldugu gibi bir dagilimdir. Verilen x, h icin bir olasilik degeri geri dondurebilir.

$$p(x, h; W) = \exp(-E(x, h))/Z$$

Standart RBM icin h, x ikiseldir (binary). Gizli (hidden) tabaka h, ve "gorunen (visible)" tabaka x vardir. Z aynen once gordugumuz Boltzman Makinalarinda (BM) oldugu gibi normalizasyon sabitidir.

Daha ilerlemeden once bir vurguyu yapalim; spesifik bir RBM'i tanimlayan sey onun W matrisidir. Gizli degiskenler bazen karisiklik yaratabiliyor, bu degiskenler aynen gorunen degiskenler gibi degiskendirler. Yani belli h'lerin "olasiligi" sorulabilir, ya da onlar uretilebilir. Fakat RBM'i egitirken sadece gorunen kismi tarafindan egitiriz. Gizli tabaka bu sirada orneklem ile arada sirada ici doldurulur, bu tabii ki W'ye bagli olarak yapilacaktir. Gizli tabaka daha dusuk boyutlu oldugu icin git/gel bir tur ozetleme yapar ki ogrenim bu sirada ortaya cikar.

Devam edelim, E tanimina "enerji" olarak ta atif yapilabiliyor.

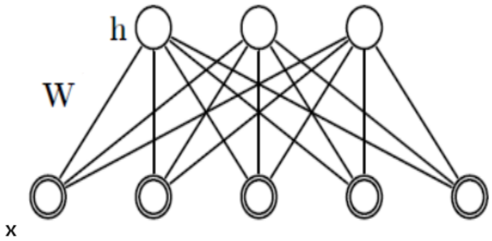
$$E(x, h) = -h^T W x - c^T x - b^T h$$

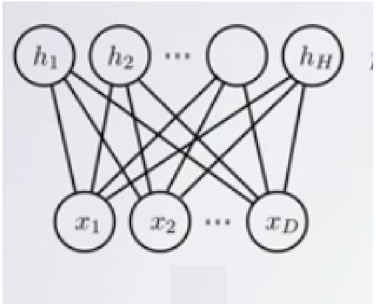
BM'lerden farkli olarak RBM'de c, b degiskenleri var. Bu degiskenler yanlilik (bias) icin, yani veri icindeki genel egilimi saptamalari icin modele konulmustur. Ayrica $h^T W x$ terimi var, bu BM'deki $x^T W x$'den biraz farkli, daha once belirttigimiz gibi, h uzerinden x'ler arasinda baglanti yapiyor. BM ile tum x ogeleri birbirine baglanabiliyordu, RBM ile h katmaninda baglantilar paylasiliyor. Bu h uzerinden baglanti zorunlulugu RBM'in ozetleme alanini azaltarak bir tur genellemeyi gerceklestirebiliyor. Bu yuzden onlara "kisitli" Boltzmann makinalari adi veriliyor. Gizli degiskenlerin kendi aralarinda, ve gorunen degiskenlerin kendi aralarinda direk baglantiya izin verilmemistir. Baglantilara, W uzerinden sadece gizli ve gorunen degiskenler (tabakalar) arasinda izin verilmistir. Bu tabii ki matematiksel olarak bazi kolayliklar sagliyor.

Formul alttaki gibi de acilabilir,

$$= -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j$$

RBM'lerin alttaki gibi resmedildigini gorebilirsiniz.





Tekrar vurgulayalim, h, x degiskenleri olasilik teorisinden bilinen rasgele degiskenlerdir, yani hem x'e hem de h'e "zar attirabiliriz" / bu degiskenler uzerinden orneklem toplayabiliriz.

Ayrica, RBM'ler aynen BM'ler gibi bir olasilik yogunluk fonksiyonu uzerinden tanimlanirlar, onceki formulde gordugumuz gibi, tum mumkun degerleri uzerinden entegralleri (ya da toplamlari) alininca sonuc 1 olur, vs.

Devam edelim, ana formulden hareketle cebirsel olarak sunlar da dogrudur,

$$p(x,h;W) = \exp(-E(x,h))/Z$$

$$= \exp(h^T W x + c^T x + b^T h)/Z \tag{2}$$

$$= \exp(h^T W x) \exp(c^T x) \exp(b^T h)/Z$$

cunku bir toplam uzerindeki exp, ayri ayri exp'lerin carpimi olur. Ayni mantikla, eger ana formulu matris / vektor yerine ayri degiskenler olarak gormek istersek,

$$p(x,h;W) = \frac{1}{Z} \prod_j \prod_k \exp(W_{jk} h_j x_k) \prod_k \exp(c_k x_k) \prod_j \exp(b_j h_j)$$

Notasyonu kolaylastirmak amaciyla b, c terimlerini W icine absorbe edebiliriz, $x_0 = 1$ ve $h_0 = 1$ degerlerini mecbur tutarsak ve $w_{0,:} = c$ ve $w_{:,0} = b$ dersek, yani W'nin sifirinci satirinin tamaminin c oldugunu, sifirinci kolonunun tamaminin b oldugunu kabul edersek RBM ana formulunu tekrar elde etmis oluruz, fakat artik

$$E(x,h) = -h^T W x$$

$$= -\sum_j \sum_k W_{j,k} h_j x_k$$

ve

$$p(x,h;W) = \exp(h^T W x)/Z$$

yeterli olacaktir. Bir diger kolaylik x, h yerine tek degisken kullanmak,

Eger $y \equiv (x,h)$ olarak alirsak,

$$P(x,h;W) = \frac{1}{Z(W)} \exp \left[\frac{1}{2} y^T W y \right]$$

Aslinda acik konusmak gerekirse "enerji" gibi kavramlarla ugrasmak, ya da icinde eksi terimler iceren bir grup degiskenin tekrar eksisini almak ve eksilerin etkisinin notralize etmis olmaya gerek yok, bunun yerine bastan (2)'deki ifadeyle yola cikmak daha kisa olur. Icinde enerji olan aciklamalari biraz da literaturde gorulebilecek anlatimlara aciklik getirmek icin yaptik.

Simdi h uzerinden marjinalize edersek,

$$P(x;W) = \sum_h \frac{1}{Z(W)} \exp \left[\frac{1}{2} y^T W y \right]$$

$$P(x;W) = \frac{1}{Z(W)} \sum_h \exp \left[\frac{1}{2} y^T W y \right] \tag{1}$$

Ve $Z(W)$

$$Z(W) = \sum_{h,x} \exp \left[\frac{1}{2} y^T W y \right]$$

(1) denkleminde bolumunden sonraki kisma $Z_x(W)$ dersek, sanki ayni exp denkleminin x'ler uzerinden marjinalize edilmis hali olarak gosterebiliriz onu, ve boylece daha kisa bir formul kullanabiliriz,

$$P(x;W) = \frac{1}{Z(W)} \underbrace{\sum_h \exp \left[\frac{1}{2} y^T W y \right]}_{Z_x(W)}$$

O zaman

$$P(x;W) = \frac{Z_x(W)}{Z(W)}$$

elde ederiz. Veri uzerinden maksimum olurluk icin, yine log uzerinden bir hesap yapariz, BM icin yapmistik bunu,

$$\mathcal{L} = \ln\big(\prod_{n=1}^{N} P(x^n; W)\big) = \sum_{n=1}^{N} \ln P(x^n; W)$$

$$= \sum_{n=1}^{N} \ln \frac{Z_{x^{(n)}}(W)}{Z(W)} = \sum_{n=1}^{N} \big(\ln Z_{x^{(n)}} - \ln Z\big)$$

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{n=1}^{N} \big(\frac{\partial \ln Z_{x^{(n)}}}{\partial w_{ij}} - \frac{\partial \ln Z}{\partial w_{ij}}\big) \tag{3}$$

Parantez icindeki 1. turevi alalim,

$$\frac{\partial \ln Z_{x^{(n)}}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \ln \bigg[\sum_h \exp\big(\frac{1}{2} {y^n}^T W y^n\big)\bigg]$$

$$= \frac{1}{Z_{x^{(n)}}} \bigg[\sum_h \frac{\partial}{\partial w_{ij}} \exp\big(\frac{1}{2} {y^n}^T W y^n\big)\bigg]$$

$$= \frac{1}{Z_{x^{(n)}}} \bigg[\sum_h \exp\big(\frac{1}{2} {y^n}^T W y^n\big) \frac{\partial}{\partial w_{ij}} {y^n}^T W y^n\bigg]$$

$$= \frac{1}{Z_{x^{(n)}}} \sum_h \exp\big(\frac{1}{2} {y^n}^T W y^n\big) y_i y_j$$

$$= \sum_h \frac{1}{Z_{x^{(n)}}} \exp\big(\frac{1}{2} {y^n}^T W y^n\big) y_i y_j$$

$Z_{x^{(n)}}$'nin ne oldugunu hatirlarsak, exp ifadesinin h uzerinden marjinalize edilmis hali,

$$= \sum_h \frac{\exp\big(\frac{1}{2} {y^n}^T W y^n\big)}{\sum_h \exp\big(\frac{1}{2} y^T W y\big)} y_i y_j$$

Eger bolumun ustunu ve altini Z ile bolsek,

$$= \sum_h \frac{\exp\big(\frac{1}{2} {y^n}^T W y^n\big)/Z}{\sum_h \exp\big(\frac{1}{2} y^T W y\big)/Z} y_i y_j$$

Ust kisim $P(y; W)$ yani $P(x, h; W)$ alt kisim $P(x; W)$ olmaz mi? Evet! Ve,

$$P(h|x^n; W) = \frac{P(x^n, h; W)}{P(x^n; W)}$$

olduguna gore,

$$= \sum_{h} P(h|x^n; W) y_i y_j$$

elde ederiz. Bunu da $< y_i y_j >_{P(h|x^n;W)}$ olarak yazabiliriz.

Simdi parantez icindeki 2. turevi alalim, yani $\frac{\partial \ln Z}{\partial w_{ij}}$,

$$\frac{\partial \ln Z}{\partial w_{ij}} = \sum_{h,x} \frac{1}{Z} \exp\big(\frac{1}{2} y^T W y\big) y_i y_j = \sum_{h,x} P(y; W) y_i y_j$$

ki bu son ifadeyi de $< y_i y_j >_{P(y;W)}$ olarak yazabiliriz. Tamamini, yani (3) ifadesini, artik soyle yazabiliriz,

$$\sum_{n=1}^{N} \big(\frac{\partial \ln Z_{x^{(n)}}}{\partial w_{ij}} - \frac{\partial \ln Z}{\partial w_{ij}}\big) = \sum_{n=1}^{N} < y_i y_j >_{P(h|x^n;W)} - < y_i y_j >_{P(y;W)} \tag{4}$$

Bu formulu de BM icin yaptigimiz gibi bir gradyan guncelleme formulune donusturebiliriz. Guncelleme formulunun hangi hesaplari gerektirdigine gelince; Ilk terim tum h'ler uzerinden, ikincisi ise tum x, h'ler uzerinden bir olasilik hesabi ve ornekleme gerektirecek. Bu durum cetin hesap (intractable) denen bir durum, ozellikle x, h sarti icin; daha once BM icin bu problemi Gibbs orneklemesi ile cozmustuk. Ayni cozumu burada da uygulayabiliriz, fakat belki daha iyi bir yaklasim su olacak.

CD Yontemi (Contrastive Divergence)

RBM'leri egitmek icin kullanilan en populer yontem CD yontemidir. Bu yontemi gecmeden once bazi matematiksel kolayliklari bilmek gerekli.

RBM grafigine bakarsak, eger x biliniyor ise bu h degiskenlerini bagimsiz hale getirir (kosullu olasilik kurali), ve ayni sekilde h biliniyor ise x bagimsiz hale gelir. Bunu gorsel olarak bile anlamak cok kolay, elimizle tum x'leri kapatalim mesela ve h dugumlerine bakalim, aralarinda hicbir baglanti yoktur degil mi? Ayni sekilde h kapatinca x'ler "baglantisiz" hale gelir.

Bu bagimsizliktan yola cikarak, daha once BM icin yaptigimiz gibi, olasiliklar su basit formullere donusur,

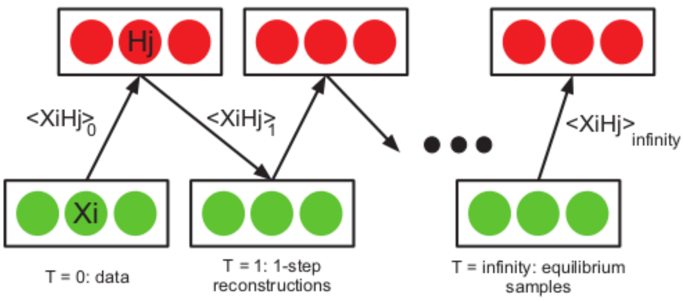
$$P(h_i = 1|x) = \sigma\bigg(\sum_{j=1}^{m} w_{ij} x_j \bigg)$$

$$P(x_i = 1|h) = \sigma\bigg(\sum_{i=1}^{n} w_{ij} h_i \bigg)$$

ve tabii ki $\sigma(x) = 1/(1+e^{-x})$. Daha once 1 olma olasiligini nasil ornekleme cevirecegimizi de gormustuk zaten.

Simdi CD'nin ne olduguna gelelim. Eger RBM icin gereken orneklemeyi klasik Gibbs ile yaparsak ornekleme zincirini "yeterince uzun sure" isletmek gerekir ki dagilimin olasi noktalari gezilmis olsun. Fakat, ozellikle yuksek boyutlu durumlarda, tum x, h kombinasyonlarini dusunursek bu cok buyuk bir alandir ve gezme islemi cok, cok uzun zaman alabilir. Bunun yerine, ve ustteki bagimsizlik formullerinden hareketle CD yontemi bulunmustur, bu yonteme gore ornekleme verinin *kendisinden* baslatilir (kiyasla Gibbs rasgele bir noktadan), dongunun mesela ilk adiminda x^0 (ki bu tum verinin tamami), baz alinarak $p(h^0|v^0)$ hesaplanir (ustteki sigmoid), onun uzerinden h^0 orneklemi alinir, sonra h^0 baz alinir ve x^1 uretilir, bu boyle devam eder. Boylece mumkun h ve x'ler gezilmis olur.

Literaturde su sekildeki resim bolca gorulebilir,



Bu yontem pur Gibbs orneklemesine kiyasla cok daha hizli isler ve iyi sonuclar verir. Teorik olarak niye isledigi [1,2] makalelerinde bulunabilir. CD aslinda (4) hedef formulunu degil baska bir hedefi optimize ediyor, fakat sonuc orijinal gradyan adimlarinin yapmak istedigine yakin. [3] baz alinarak, su sekilde kodlanabilir,

```
import numpy as np
import itertools

class RBM:

  def __init__(self, num_hidden, learning_rate,max_epochs, num_visible=10):
    self.num_hidden = num_hidden
    self.num_visible = num_visible
    self.learning_rate = learning_rate
    # Agirlik matrisi W'yi yarat (buyukluk num_visible x num_hidden),
    # bunun icin Gaussian dagilimi kullan, ortalama=0, standart sapma 1.
    self.weights = 0.1 * np.random.randn(self.num_visible, self.num_hidden)
    # Egilim (bias) icin ilk satir ve ilk kolona 1 degeri koy
    self.weights = np.insert(self.weights, 0, 0, axis = 0)
    self.weights = np.insert(self.weights, 0, 0, axis = 1)
    self.max_epochs = max_epochs
```

```
def fit(self, data):
  """
  Makinayi egit

  Parametreler
  ----------
  data: Her satirin "gorunen" veri oldugu bir matris
  """

  num_examples = data.shape[0]

  # Ilk kolona egilim / meyil (bias) olarak 1 ekle
  data = np.insert(data, 0, 1, axis = 1)

  for epoch in range(self.max_epochs):
    # Veriyi baz alarak gizli veriyi uret.
    pos_hidden_activations = np.dot(data, self.weights)
    pos_hidden_probs = self._logistic(pos_hidden_activations)
    pos_hidden_states = pos_hidden_probs > \
        np.random.rand(num_examples, self.num_hidden + 1)

    tmp = np.array(pos_hidden_states).astype(float)
    pos_visible_states = self.run_hidden(tmp[:,1:])

    # Dikkat, baglantilari hesaplarken h tabakasinin aktivasyon
    # olasiliklarini kullaniyoruz h'nin kendi degerlerini (0/1)
    # kullanmiyoruz. Bunu da yapabilirdik, daha fazla detay icin
    # Hinton'un "A Practical Guide to Training Restricted Boltzmann
    # Machines" makalesine bakilabilir
    pos_associations = np.dot(data.T, pos_hidden_probs)

    # Simdi gorunen veriyi gizli veriyi baz alip tekrar uret
    neg_visible_activations = np.dot(pos_hidden_states, self.weights.T)
    neg_visible_probs = self._logistic(neg_visible_activations)
    neg_visible_probs[:,0] = 1 # Fix the bias unit.
    neg_hidden_activations = np.dot(neg_visible_probs, self.weights)
    neg_hidden_probs = self._logistic(neg_hidden_activations)

    # Yine ayni durum, aktivasyon olasiliklari kullaniliyor
    neg_associations = np.dot(neg_visible_probs.T, neg_hidden_probs)

    # Agirliklari guncelle
    self.weights += self.learning_rate * \
        ((pos_associations - neg_associations) / num_examples)

    error = np.sum((data - neg_visible_probs) ** 2)

def run_visible(self, data):
  """
  RBM'in egitilmis olduguna farz ederek, gorunen veri uzerinde
  RBM'i islet, ve h icin bir orneklem al

  Parametreler
  ----------
  data: Her satirin gorunen veri oldugu bir matris
```

```
        Returns
        --------
        hidden_states: data icindeki her satira tekabul eden gizli h verisi
        """
        num_examples = data.shape[0]

        hidden_states = np.ones((num_examples, self.num_hidden + 1))

        data = np.insert(data, 0, 1, axis = 1)

        hidden_activations = np.dot(data, self.weights)
        hidden_probs = self._logistic(hidden_activations)
        hidden_states[:,:] = hidden_probs > \
            np.random.rand(num_examples, self.num_hidden + 1)
        hidden_states = hidden_states[:,1:]
        return hidden_states


    def run_hidden(self, data):
        """
        run_visible'a benzer, sadece gizli veri icin gorunen veri uret
        """

        num_examples = data.shape[0]

        visible_states = np.ones((num_examples, self.num_visible + 1))

        data = np.insert(data, 0, 1, axis = 1)

        visible_activations = np.dot(data, self.weights.T)
        visible_probs = self._logistic(visible_activations)
        visible_states[:,:] = visible_probs > \
            np.random.rand(num_examples, self.num_visible + 1)

        visible_states = visible_states[:,1:]
        return visible_states

    def _logistic(self, x):
        return 1.0 / (1 + np.exp(-x))
```

RBM ve Siniflama

Siniflama (classification) islemi yapmak icin BM orneginde bir normalizasyon sabiti hesaplamistik. Burada degisik bir yoldan gidecegiz; ki bu yol ileride Derin Ogrenim icin faydali olacak.

Egittikten sonra bir RBM, icindeki W'ye gore, herhangi bir "gorunur" veri noktasi x icin bir gizli bir h uretebilir. Bunu ustteki formulasyondan zaten biliyoruz. Ayrica, h genellikle daha az boyutta olduguna gore (hatta olmasa bile) bu h uretiminin bir tur transformasyon oldugu, veri uzerinde bir "ozetleme" yaptigi iddia edilebilir. O zaman teorik olarak, gorunur veri yerine, gorunur veriden uretilen gizli veriyi kullanirsak ve bu veriyi alip baska bir siniflayiciya verirsek, mesela lojistik regresyon gibi, bu h'ler ve etiketler uzerinden takip edilen bir (supervised) egitim yapabiliriz. Yani, once RBM egitiyoruz, tum verinin h karsiligini aliyoruz, sonra bunlari lojistik regresyona veriyoruz. Alttaki kodda bunun orneginin gorebiliriz.

Bu kod, ayrica, k-Katlama (k-fold) teknigini uyguluyor, veriyi 3 parcaya bolup sirasiyla tum parcalari birer kez test, digerlerini egitim verisi yapiyor, boylece verinin tamami uzerinden egitim/test yapmis olunuyor. Sonuc,

```
import cPickle, numpy as np, rbm
from sklearn.linear_model import LogisticRegression

from sklearn.base import BaseEstimator
from sklearn.base import TransformerMixin

class SKRBM(BaseEstimator, TransformerMixin):
    """
    Bu sinif bizim RBM kodu ile sklearn arasinda baglantiyi kurmak
    icin yazildi, tek yaptigi parametreleri alip RBM'i cagirmak, bu
    baglanti aslinda KFold icin muhakkak lazim degil (run_visible ile
    de bu isi halledebilirdik), fakat GridSearch kullanmak gerekirse
    class'in icinde transform() cagrisinin olmasi lazim.
    """
    def __init__(self, n_components=-1, learning_rate=-1, n_iter=-1,num_visible=-1):
      self.n_components = n_components
      self.learning_rate = learning_rate
      self.n_iter = n_iter
      self.num_visible = num_visible
      self.rbm_ = rbm.RBM(num_hidden=self.n_components,
                          learning_rate=self.learning_rate,
                          max_epochs=self.n_iter,num_visible=num_visible)

    def transform(self, X):
      return self.rbm_.run_visible(X)

    def fit(self, X, y=None):
      self.rbm_.fit(X)
      return self

X = np.loadtxt('../../stat/stat_mixbern/binarydigits.txt')
Y = np.ravel(np.loadtxt('../../stat/stat_mixbern/bindigitlabels.txt'))

np.random.seed(0)

from sklearn.cross_validation import KFold
scores = []
cv = KFold(n=len(X),n_folds=3)
for train, test in cv:
    X_train, Y_train = X[train], Y[train]
    X_test, Y_test = X[test], Y[test]
    r = SKRBM(n_components=40, learning_rate=0.3, n_iter=500,num_visible=64)
    r.fit(X_train)
    clf = LogisticRegression(C=1000)
    clf.fit(r.transform(X_train), Y_train)
    res3 = clf.predict(r.transform(X_test))
```

```
    scores.append(np.sum(res3==Y_test) / float(len(Y_test)))

print np.mean(scores)
```

```
! python test_rbmkfold.py
```

```
1.0
```

Basari yuzde 100! Altta karsilastirma icin KNN teknigi kullandik,

```
import cPickle, numpy as np, gzip
from sklearn import neighbors
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegression

X = np.loadtxt('../../stat/stat_mixbern/binarydigits.txt')
Y = np.ravel(np.loadtxt('../../stat/stat_mixbern/bindigitlabels.txt'))

from sklearn.cross_validation import KFold
scores = []
cv = KFold(n=len(X),n_folds=3)
for train, test in cv:
    X_train, Y_train = X[train], Y[train]
    X_test, Y_test = X[test], Y[test]
    clf = neighbors.KNeighborsClassifier(n_neighbors=1)
    clf.fit(X_train, Y_train)
    scores.append(clf.score(X_test, Y_test))

print np.mean(scores)
```

```
! python test_knnkfold.py
```

```
0.98009506833
```

Kaynaklar

[1] Hinton, G., Training Products of Experts by Minimizing Contrastive Divergence

[2] Louppe, G., Collaborative filtering, Scalable approaches using restricted Boltzmann machines, Master Tezi, 2010

[3] https://github.com/echen/restricted-boltzmann-machines