Kisitli Boltzmann Makinalari (Restricted Boltzmann Machines -RBM-)

Bir RBM icinde ikisel (binary) degerler tasiyan gizli (hidden) h degiskenler, ve yine ikisel gorunen (visible) degiskenler v vardir. Z aynen once gordugumuz Boltzman Makinalarinda (BM) oldugu gibi normalizasyon sabitidir.

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

E tanimina "enerji" olarak ta atif yapilabiliyor.

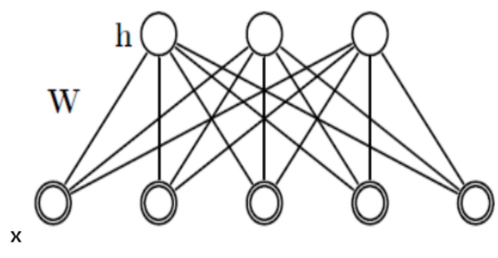
$$E(x, h) = -h^T W x - c^T x - b^T h$$

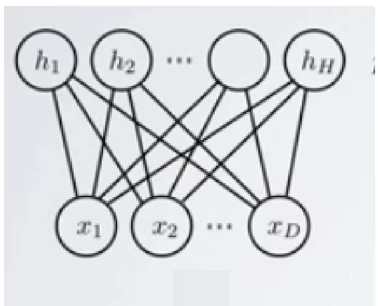
BM'lerden farkli olarak RBM taniminda c, b degiskenleri var. Bu degiskenler yanlilik (bias) icin, yani veri icindeki genel egilimi saptamalari icin modele konulmustur. Ayrica $h^T W x$ terimi var, bu BM'deki $x^T W x$ biraz farkli, gizli degiskenler, h uzerinden x'ler arasinda baglanti yapiyor. Bir baska ilginc farklilik BM ile tum x ogeleri birbirine baglanabiliyordu, RBM ile daha az (ya da fazla) olabilecek h katmaninda baglantilar paylasiliyor. Ozellikle azaltma durumunda RBM ozellik alanini azaltarak bir tur genellemeyi gerceklestirebiliyor.

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.





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.

RBM'lerin "kisitli" olarak tanimlanmalarinin sebebi gizli degiskenlerin kendi aralarinda, ve gorunen degiskenlerin kendi aralarinda direk baglantiya izin verilmemis olmasidir, bu bakimdan "kisitlanmislardir". Baglantilara, W uzerinden sadece gizli ve gorunen degiskenler (tabakalar) arasinda izin verilmistir. Bu tabii ki matematiksel olarak bazi kolayliklar sagliyor.

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 tamamini c'ye set edip, sifirinci kolonunun tamamini b'ye set 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 olmak yerine bastan (2)'deki ifadeyle yola cikmak daha kisa. Giristeki aciklamalari literaturde gorulebilecek bazi anlatimlari aciklamak icin yaptik.

Neyse, 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 "farkli bir sekilde marjinalize edilmis hali" olarak gostermis oluruz 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 $\frac{\partial \ln Z}{\partial w_{ij}}$,

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

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

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

```
import numpy as np
import itertools

class RBM:

  def __init__(self, num_visible, num_hidden, learning_rate = 0.1,\
               max_epochs = 1000):
    self.num_hidden = num_hidden
    self.num_visible = num_visible
    self.learning_rate = learning_rate
    self.norm_dict = {}
    self.weights = 0.1 * np.random.randn(self.num_visible, self.num_hidden)
    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):
    num_examples = data.shape[0]

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

    for epoch in range(self.max_epochs):
      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:])

      for h,v in itertools.izip(pos_hidden_states.astype(float),
                                pos_visible_states):
        v = np.insert(v, 0, 1)
        self.norm_dict[(tuple(h),tuple(v))] = 1

      pos_associations = np.dot(data.T, pos_hidden_probs)

      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)

      neg_associations = np.dot(neg_visible_probs.T, neg_hidden_probs)

      self.weights += self.learning_rate * \
          ((pos_associations - neg_associations) / num_examples)
```

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

    self.norm_c = self.norm_constant()

  def run_hidden(self, data):

    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 run_visible(self, data):
    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 norm_constant(self):
    sum = 0
    for h,v in self.norm_dict:
      h = np.array(h); v = np.array(v)
      sum += np.dot(np.dot(h.T,self.weights.T), v)
    return sum

  def predict_proba(self, X):
    hs = self.run_visible(X)
    hs = np.insert(hs, 0, 1,axis=1)
    res = []
    for i in range(len(X)):
      tmp = np.dot(hs[i],self.weights.T)
      res.append(np.dot(tmp.T,np.insert(X[i], 0, 1)))
    return np.array(res) / self.norm_c

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

from sklearn import neighbors
from sklearn.cross_validation import train_test_split
```

```python
import numpy as np, rbm

x = np.loadtxt('../../stat/stat_mixbern/binarydigits.txt')
labels = np.ravel(np.loadtxt('../../stat/stat_mixbern/bindigitlabels.txt'))
X_train, X_test, y_train, y_test = train_test_split(x, labels, test_size=0.2,random_st
print X_train.shape

clfs = {}
for label in [0,5,7]:
    x = X_train[y_train==label]
    clf = rbm.RBM(num_visible = 64, num_hidden = 4, max_epochs = 500)
    clf.fit(x)
    clfs[label] = clf

res = []
for label in [0,5,7]:
    res.append(clfs[label].predict_proba(X_test))
res3 = np.argmax(np.array(res).T,axis=1)
res3[res3==1] = 5
res3[res3==2] = 7
res3 = map(lambda x: float(x), res3)
print 'RBM', np.sum(res3==y_test) / float(len(y_test))

clf = neighbors.KNeighborsClassifier()
clf.fit(X_train,y_train)
res3 = clf.predict(X_test)
print 'KNN', np.sum(res3==y_test) / float(len(y_test))
```

```
(80, 64)
RBM 1.0
KNN 1.0
```