Exercises for the course

Machine Learning 2

Summer semester 2021

Abteilung Maschinelles Lernen
Institut für Softwaretechnik und theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Exercise Sheet 12

Exercise 1: Deep SVDD (20 P)

Consider a dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^d$, and a simple linear feature map $\phi(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ with trainable parameters $\boldsymbol{w}$ and b. For this simple scenario, we can formulate the deep SVDD problem as:

$$\min_{\boldsymbol{w},b} \quad \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{w}^\top \boldsymbol{x}_i + b - 1\|^2$$

where we have hardcoded the center parameter of deep SVDD to 1. We then classify new points $\boldsymbol{x}$ to be anomalous if $\|\boldsymbol{w}^\top \boldsymbol{x} + b - 1\|^2 > \tau$.

(a) *Give* a choice of parameters $(\boldsymbol{w}, b)$ that minimizes the objective above for any dataset $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$.

(b) We now consider a regularizer for our feature map ϕ which simply consists of forcing the bias term to $b = 0$. *Show* that under this regularizer, the solution of deep SVDD is given by:

$$\boldsymbol{w} = \Sigma^{-1} \bar{\boldsymbol{x}}$$

where $\bar{\boldsymbol{x}}$ and Σ are the empirical mean and uncentered covariance.

Exercise 2: Restricted Boltzmann Machine (30 P)

The restricted Boltzmann machine is a system of binary variables comprising inputs $\boldsymbol{x} \in \{0,1\}^d$ and hidden units $\boldsymbol{h} \in \{0,1\}^K$. It associates to each configuration of these binary variables the energy:

$$E(\boldsymbol{x}, \boldsymbol{h}) = -\boldsymbol{x}^\top W \boldsymbol{h} - \boldsymbol{b}^\top \boldsymbol{h}$$

and the probability associated to each configuration is then given as:

$$p(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z} \exp(-E(\boldsymbol{x}, \boldsymbol{h}))$$

where Z is a normalization constant that makes probabilities sum to one. Let $\mathrm{sigm}(t) = \exp(t)/(1 + \exp(t))$ be the sigmoid function.

(a) *Show* that $p(h_k = 1 \,|\, \boldsymbol{x}) = \mathrm{sigm}\big(\boldsymbol{x}^\top W_{:,k} + b_k\big)$.

(b) Show that $p(x_j = 1 \,|\, \boldsymbol{h}) = \mathrm{sigm}\big(W_{j,:}^\top \boldsymbol{h}\big)$.

(c) *Show* that

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp(-F(\boldsymbol{x}))$$

where

$$F(\boldsymbol{x}) = -\sum_{k=1}^{K} \log\big(1 + \exp\big(\boldsymbol{x}^\top W_{:,k} + b_k\big)\big)$$

is the free energy and where Z is again a normalization constant.

Exercise 3: Programming (50 P)

Download the programming files on ISIS and follow the instructions.

KDE and RBM for Anomaly Detection

In this programming exercise, we compare in the context of anomaly detection two energy-based models: kernel density estimation (KDE) and the restricted Boltzmann machine (RBM).

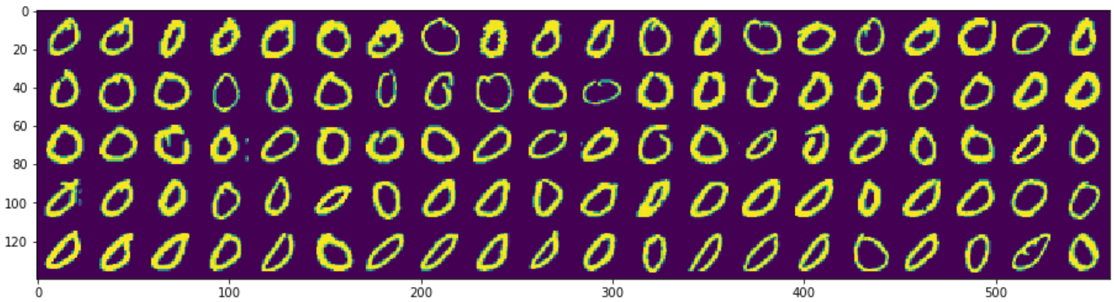
In [1]:
```
import utils
import numpy
import scipy,scipy.special,scipy.spatial
import sklearn,sklearn.metrics
%matplotlib inline
import matplotlib
from matplotlib import pyplot as plt
```

We consider the MNIST dataset and define the class "0" to be normal (inlier) and the remain classes (1-9) to be anomalous (outlier). We consider that we have a training set `Xr` composed of 100 normal data points. The variables `Xi` and `Xo` denote normal and anomalous test data.

In [2]:
```
Xr,Xi,Xo = utils.getdata()
```

The 100 training points are visualized below:

In [3]:
```
plt.figure(figsize=(16,4))
plt.imshow(Xr.reshape(5,20,28,28).transpose(0,2,1,3).reshape(140,560))
plt.show()
```



Kernel Density Estimation (15 P)

We first consider kernel density estimation which is a shallow model for anomaly detection. The code below implement kernel density estimation.

Task:

- Implement the function `energy` that returns the energy of the points `X` given as input as computed by the KDE energy function (cf. slide Kernel Density Estimation as an EBM).

In [4]:
```
class AnomalyModel:

    def auroc(self):
        Ei = self.energy(Xi)
        Eo = self.energy(Xo)
        return sklearn.metrics.roc_auc_score(
            numpy.concatenate([Ei*0+0,Eo*0+1]),
            numpy.concatenate([Ei,Eo])
        )

class KDE(AnomalyModel):

    def __init__(self,gamma):
        self.gamma = gamma

    def fit(self,X):
        self.X = X

    def energy(self,X):

        # ------------------------------------------------
        # TODO: Replace by your code
        # ------------------------------------------------
        import solution
        E = solution.kde_energy(self,X)
        # ------------------------------------------------

        return E
```

The following code applies KDE with different scale parameters `gamma` and returns the performance of the resulting anomaly detection model measured in terms of area under the ROC.

In [5]:
```
for gamma in numpy.logspace(-2,0,10):

    kde = KDE(gamma)
    kde.fit(Xr)
    print('gamma = %5.3f  AUROC = %5.3f'%(gamma,kde.auroc()))
```

```
gamma = 0.010  AUROC = 0.957
gamma = 0.017  AUROC = 0.962
gamma = 0.028  AUROC = 0.969
gamma = 0.046  AUROC = 0.976
gamma = 0.077  AUROC = 0.981
gamma = 0.129  AUROC = 0.983
gamma = 0.215  AUROC = 0.983
gamma = 0.359  AUROC = 0.982
gamma = 0.599  AUROC = 0.982
gamma = 1.000  AUROC = 0.981
```

We observe that the best performance is obtained for some intermediate value of the parameter `gamma` .

Restricted Boltzmann Machine (35 P)

We now consider a restricted Boltzmann machine composed of `100` binary hidden units ($\boldsymbol{h} \in \{0,1\}^{100}$). The joint energy function of our RBM is given by:

$$E(\boldsymbol{x},\boldsymbol{h}) = -\boldsymbol{x}^\top \boldsymbol{a} - \boldsymbol{x}^\top W \boldsymbol{h} - \boldsymbol{h}^\top \boldsymbol{b}$$

The model can be marginalized over its hidden units and the energy function that depends only on the input $\boldsymbol{x}$ is then given as:

$$E(\boldsymbol{x}) = -\boldsymbol{x}^\top \boldsymbol{a} - \sum_{k=1}^{100} \log(1 + \exp(\boldsymbol{x}^\top W_{:,k} + b_k))$$

The RBM training algorithm is already implemented for you.

Tasks:

- **Implement the energy function $E(\boldsymbol{x})$**
- **Augment the function `fit` with code that prints the AUROC every 100 iterations.**

In [6]:
```
def sigm(t): return numpy.tanh(0.5*t)*0.5+0.5
def realize(t): return 1.0*(t>numpy.random.uniform(0,1,t.shape))

class RBM(AnomalyModel):

    def __init__(self,X,h):
        self.mb = X.shape[0]
        self.d = X.shape[1]
        self.h = h
        self.lr = 0.1

        # Model parameters
        self.A = numpy.zeros([self.d])
        self.W = numpy.random.normal(0,self.d**-.25 * self.h**-.25,[self.
d,self.h])
        self.B = numpy.zeros([self.h])

    def fit(self,X,verbose=False):

        Xm = numpy.zeros([self.mb,self.d])

        for i in numpy.arange(1001):

            # Gibbs sampling (PCD)
            Xd = X*1.0
            Zd = realize(sigm(Xd.dot(self.W)+self.B))
            Zm = realize(sigm(Xm.dot(self.W)+self.B))
            Xm = realize(sigm(Zm.dot(self.W.T)+self.A))

            # Update parameters
            self.W += self.lr*((Xd.T.dot(Zd) - Xm.T.dot(Zm)) / self.mb -
0.01*self.W)
            self.B += self.lr*(Zd.mean(axis=0)-Zm.mean(axis=0))
            self.A += self.lr*(Xd.mean(axis=0)-Xm.mean(axis=0))

            if verbose:
                # --------------------------------------------------
                # TODO: Replace by your code
                # --------------------------------------------------
                import solution
                solution.track_auroc(self,i)
                # --------------------------------------------------


    def energy(self,X):

        # --------------------------------------------------
        # TODO: Replace by your code
        # --------------------------------------------------
        import solution
        E = solution.rbm_energy(self,X)
        # --------------------------------------------------

        return E
```

We now train our RBM on the same data as the KDE model for approximately 1000 iterations.

In [7]:
```
rbm = RBM(Xr,100)
rbm.fit(Xr,verbose=True)
```

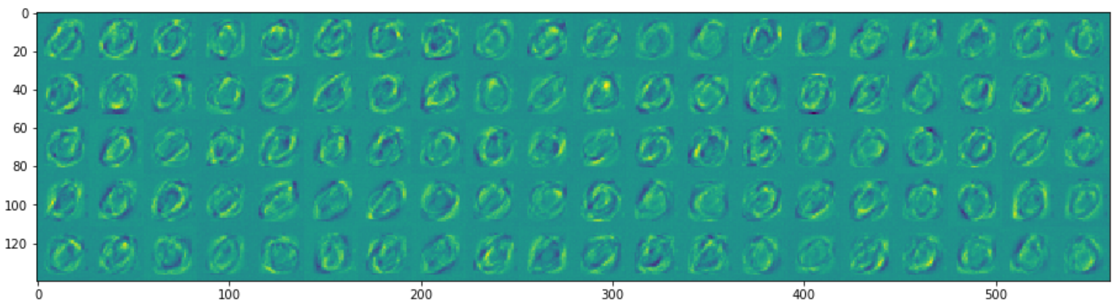
```
it =     0  AUROC = 0.962
it =   100  AUROC = 0.943
it =   200  AUROC = 0.985
it =   300  AUROC = 0.987
it =   400  AUROC = 0.988
it =   500  AUROC = 0.986
it =   600  AUROC = 0.987
it =   700  AUROC = 0.987
it =   800  AUROC = 0.989
it =   900  AUROC = 0.986
it =  1000  AUROC = 0.990
```

We observe that the RBM reaches superior levels of AUROC performance compared to the simple KDE model. An advantage of the RBM model is that it learns a set of parameters that represent variations at multiple scales and with specific orientations in input space. We would like to visualize these parameters:

Task:

- **Render as a mosaic the weight parameters (`W`) of the model. Each tile of the mosaic should correspond to the receptive field connecting the input image to a particular hidden unit.**

In [8]:
```
# ------------------------------------------------
# TODO: Replace by your code
# ------------------------------------------------
import solution
solution.plot_weights(rbm)
# ------------------------------------------------
```



Exercise Sheet 12

1 Deep SVDD

a) Minimum at $\vec{w} = \vec{0}$, $b = 1$

b)
$$J = \min_W \frac{1}{N} \sum_{i=1}^{N} \|w^T x_i - 1\|^2 = \min_W \frac{1}{N} \sum_{i=1}^{N} (w^T x_i - 1)^T (w^T x_i - 1)$$
$$= \min_W \frac{1}{N} \sum_{i=1}^{N} w^T x_i x_i^T w - 2 w^T x_i + 1$$
$$= \min_W w^T \frac{\sum_{i=1}^{N} x_i x_i^T}{N} w - \frac{2}{N} \sum_{i=1}^{N} x_i^T w + \frac{1}{N} \sum_{i=1}^{N} 1$$

$$\frac{\partial J}{\partial w} = (\Sigma + \Sigma^T) w - 2 \frac{1}{N} \sum_{i=1}^{N} x_i = 0$$

$$\Leftrightarrow 2 \Sigma w - 2 \bar{x} = 0 \quad \Leftrightarrow \quad w = \Sigma^{-1} \bar{x}$$

2 Restricted Boltzmann Machine

a)
$$p(h_k = 1 \mid x) = \frac{p(x, h_k = 1)}{p(x)} = \frac{\sum_{h_{-k}} p(x, h_k = 1, h_{-k})}{p(x)}$$
$$= \frac{\sum_{h_{-k}} p(x, h_k = 1, h_{-k})}{\sum_{q \in \{0,1\}} \sum_{h_{-k}} p(x, h_k = q, h_{-k})}$$
$$= \frac{\sum_{h_{-k}} \frac{1}{Z} \exp(x^T W_{:,k} + b_k - E(x, h_{-k}))}{\sum_{q \in \{0,1\}} \sum_{h_{-k}} \frac{1}{Z} \exp(x^T W_{:,k} q + b_k q - E(x, h_{-k}))}$$
$$= \frac{\exp(x^T W_{:,k} + b_k) \sum_{h_{-k}} \exp(-E(x, h_{-k}))}{\sum_{q \in \{0,1\}} \exp(x^T W_{:,k} q + b_k q) \sum_{h_{-k}} \exp(-E(x, h_{-k}))}$$
$$= \frac{\exp(x^T W_{:,k} + b_k)}{1 + \exp(x^T W_{:,k} + b_k)} = \mathrm{sigm}(x^T W_{:,k} + b_k)$$

b) $p(x_j = 1 | h) = \frac{p(x_j = 1, h)}{p(h)} = \frac{\sum_{x_{-j}} p(x_j = 1, x_{-j}, h)}{\sum_{q \in \{0,1\}} \sum_{x_{-j}} p(x_j = q, x_{-j}, h)}$

$$= \frac{\sum_{x_{-j}} \frac{1}{Z} \exp(W_{j,:}^T h - E(x_{-j}, h))}{\sum_{q \in \{0,1\}} \sum_{x_{-j}} \frac{1}{Z} \exp(W_{j,:}^T h q - E(x_{-j}, h))}$$

$$= \frac{\exp(W_{j,:}^T h)}{1 + \exp(W_{j,:}^T h)} = \mathrm{sigm}(W_{j,:}^T h)$$

c) $p(x) = \sum_h p(x, h) = \sum_h \frac{1}{Z} \exp(-E(x, h))$

$$= \sum_h \frac{1}{Z} \exp(x^T W h - b^T h) = \sum_h \frac{1}{Z} \exp(\sum_k x^T W_{:,k} h_k - b_k^T h_k$$

$$= \frac{1}{Z} \sum_h \prod_k \exp(x^T W_{:,k} h_k - b_k^T h_k)$$

$$= \frac{1}{Z} \prod_k (1 + \exp(x^T W_{:,k} - b_k))$$

$$= \frac{1}{Z} \exp(\log(\prod_k (1 + \exp(x^T W_{:,k} - b_k))))$$