平成２９年度茨城大学工学部情報工学科
卒業研究論文

VAEによる用例ベクトルの次元縮約

平成30年2月5日提出

茨城大学　工学部情報工学科
14T4071T 渡部 勇樹

指導教員：新納　浩幸　教授

平成 29 年度茨城大学工学部情報工学科卒業研究

VAE による
用例ベクトルの次元縮約

氏名：14T4071T 渡部 勇樹
指導教員：新納 浩幸 教授

論文要旨

本論文では Variational AutoEncoder (VAE) を利用して、語義曖昧性解消で用いる対象単語に対する用例ベクトルの次元縮約を試みる。

語義曖昧性解消などの分類問題の解決にはラベル付きデータを大量に用意することが最も効果的である。ただしラベル付きデータは手作業で構築しなければならないために、大量のラベル付きデータを実際に準備することは膨大なコストがかかり困難である。また語義曖昧性解消では対象単語毎に大量のラベル付きデータが必要であるため、この問題は更に深刻である。ただしラベルなしデータならば大量に入手できるという状況もあり得る。語義曖昧性解消では対象単語を含む用例文がラベルなしデータに対応するため、その入手は比較的容易である。

少量のラベル付きデータと大量のラベルなしデータを用いて分類器を学習する手法は、半教師あり学習と呼ばれる。一般に半教師あり学習は、教師あり学習よりも精度の高い分類器を構築でき、実践的な手法として注目されている。そして近年、深層生成モデルを利用した半教師あり学習が提案され、従来までの半教師あり学習の手法よりも精度の高い分類器を学習できることが示された。その中でも VAE を利用した半教師あり学習は理論的に明確であり、他の深層生成モデルを利用した半教師あり学習よりも実装が容易である。VAE による半教師あり学習は、まずラベルなしデータを分類が容易になるような低次元の空間にマップする。語義曖昧性解消では、この操作は用例ベクトルを次元縮約することに相当する。

本論文では 用例ベクトルの次元縮約プログラムを実装し、その効果と問題について調査する。対象単語の前後２単語を分散表現データ nwjc2vec を用いて 800 次元のベクトルに変換し、これを用例ベクトルとして次元縮約する。

SemEval2 の日本語辞書タスクで用いられて対象単語「子ども」（語義は２つ）の訓練データとテストデータの 100 用例について、それぞれの 800 次元のベクトルを VAE を用いて２次元にマップし、語義の分布図を作成した。VAE による用例ベクトルの次元縮約自体は可能であることが示されたが、分類が効果的になるように次元縮約できたのかどうかは不明である。今後、半教師あり学習を実装し、この点を確認する必要がある。

目次

図目次

第 1 章

序論

1.1 概要

本論文では Variational AutoEncoder (VAE) を利用して、語義曖昧性解消で用いる対象単語に対する用例ベクトルの次元縮約を試みる。

語義曖昧性解消などの分類問題の解決にはラベル付きデータを大量に用意することが最も効果的である。ただしラベル付きデータは手作業で構築しなければならないために、大量のラベル付きデータを実際に準備することは困難である。また語義曖昧性解消では対象単語毎に大量のラベル付きデータが必要であるため、この問題は更に深刻である。ただしラベルなしデータならば大量に入手できるという状況もあり得る。語義曖昧性解消では対象単語を含む用例文がラベルなしデータに対応するため、その入手は比較的容易である。

少量のラベル付きデータと大量のラベルなしデータを用いて分類器を学習する手法は、半教師あり学習と呼ばれる。一般に半教師あり学習は、教師あり学習よりも精度の高い分類器を構築でき、実践的な手法として注目されている。そして近年、深層生成モデルを利用した半教師あり学習が提案され、従来までの半教師あり学習の手法よりも精度の高い分類器を学習できることが示された。その中でも VAE を利用した半教師あり学習は理論的に明確であり、他の深層生成モデルを利用した半教師あり学習よりも実装が容易である。VAE による半教師あり学習は、まずラベルなしデータを分類が容易になるような低次元の空間にマップする。語義曖昧性解消では、この操作は用例ベクトルを次元縮約することに相当する。

本論文では VAE による用例ベクトルの次元縮約のプログラムを実装し、その効果と問題について調査する。対象単語の前後 2 単語を分散表現データ nwjc2vec を用いて 800 次元のベクトルに変換し、これを用例ベクトルとして次元縮約する。

SemEval2 の日本語辞書タスクで用いられて対象単語「子ども」(語義は 2 つ) の訓練データとテストデータの 100 用例について、それぞれの 800 次元のベクトルを VAE を用いて 2 次元にマップし、語義の分布図を作成した。VAE による用例ベクトルの次元縮約自体は可能であることが示されたが、分類が効果的になるように次元縮約できたのかどうかは不明である。今後、半教師あり学習を実装し、この点を確認する必要がある。

1.2 構成

本論文では、初めに理論とその手法について説明する。第 2 章で AE(Auto Encoder) について概説し、その後 VAE(Variational Auto Encoder) の仕組みについて説明する。第 3 章で単語の分散表現につい

て、第 4 章では、語義曖昧性解消 (Word Sense Disambiguation, WSD) について概説する。第 5 章で VAE を用例ベクトルに用い次元縮約した際の結果と問題について記述しする。最後に、第 6 章で結論と今後の課題について述べる。

第 2 章

VAE

2.1 次元縮約

機械学習において、データセットは高次元な場合が多く、膨大になりやすい。データを圧縮することができれば単純に、高速に計算を行うことができるようになるほか、データの類似性を見出しやすくなるといった利点がある。ただし、圧縮により元のデータが少なからず損失している点には注意が必要である。また、データを二次元や三次元まで落とし込むことで可視化することができる。

2.2 AE

AE(オートエンコーダ) は、ニューラルネットワークの一種で、情報量を小さくした特徴表現を獲得するためにある。入力データは訓練データのみで、教師データは利用しないため、教師なし学習の一つとみなすことができる。入力データ X から潜在変数 z に変換するニューラルネットワークを Encoder という。この時、z の次元が入力 X より小さい場合、次元縮約とみなすことができて、これこそが AE の核である。逆に、潜在変数 z をインプットして元データを復元するニューラルネットワークを Decoder という。以下に AE の流れを図 2.1 で表す。

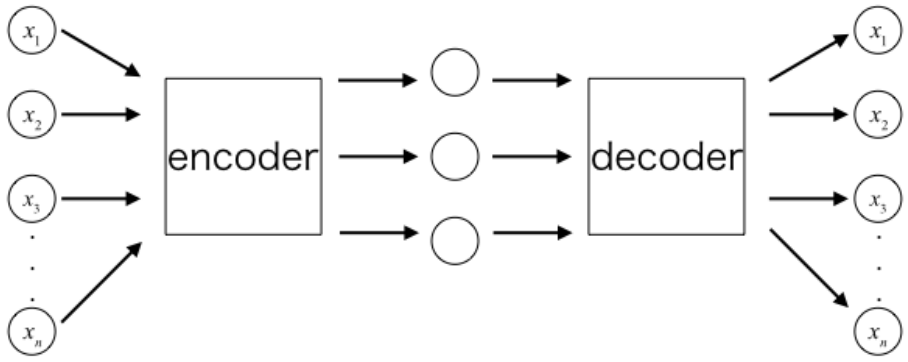

図 2.1　AE の流れ

2.2.1　教師なし学習

一般的な教師なし学習では、入力データのみを与え学習させる。正しい出力は与えられないため、なんらかの基準を設けてそれを最適にするような出力の割り当てを求める。教師あり学習とは異なり、予測の正しさといった基準がないため、基準の設計の仕方によって結果は異なる。

AE での教師なし学習では、入力と出力を同じデータにして学習することを考える。もし、各層の入力データが同じ次元になるとしたら、データはただ単に出力層に向かってコピーされればいい。しかし、隠れ層の次元を小さくするとすれば同様のデータを小さな情報量に圧縮するなんらかの方法をニューラルネットワークは学習しないといけない。これが AE のやっていることである。一度オートエンコーダを訓練すると、エンコーダとデコーダを別々に使うことができる。つまり、エンコーダを情報量を小さくした特徴表現獲得に使うことが可能になる。

2.3　VAE

VAE（Variational Auto Encoder）は、AE の処理に加え確率分布を仮定するものである。通常のオートエンコーダだと、何かしら潜在変数 z にデータを押し込めているものの、その構造がどうなっているかは、よくわからない。VAE は、潜在変数 z を確率分布という構造に押し込めることを可能にする。

2.3.1　生成モデル

生成モデルとは観測データを生成する確率分布を想定し、観測データからその確率分布を推定する方法である。ここでは、観測データ x はとある因子 z から生成されたと考える。

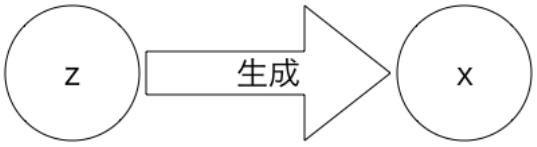

図 2.2　x は z から生成される

しかし、生成過程は一般的に未知である。そのためにデータ x を活用して生成過程を逆向きに辿って x から z を推論する。

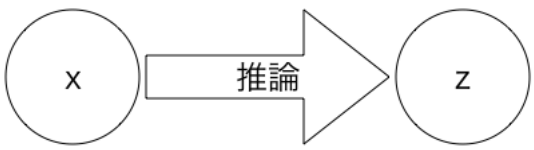

図 2.3　z を x から推論

2.3.2　VAE の仕組み

encoder 部分 $q_\phi(z|x)$ はデータ x から潜在変数 z を推論し、重みとバイアス ϕ を持つ確率分布である。一方、decoder 部分 $p_\theta(x|z)$ は潜在変数 z からデータ x を生成し、重みとバイアス θ を持つ確率分布である。このモデルがどのように学習するかについて以下に示す。

1. データ x をサンプリングして Encoder に入力する
2. Encoder は出力次元に相当する潜在分布の平均 μ と分散 σ を出力する
3. 潜在分布の平均と分散から、潜在変数 z をサンプリングし、z を作る
4. 潜在変数 z を Decoder に入力し、Decoder はデータ x' を出力する
5. サンプリングされたデータそのものを復元するように Encoder と Decoder を学習し、Encoder は復元のための学習に加えて、仮定した事前分布に近くなるよう学習する

ここまでの流れを図 2.4 に表す。

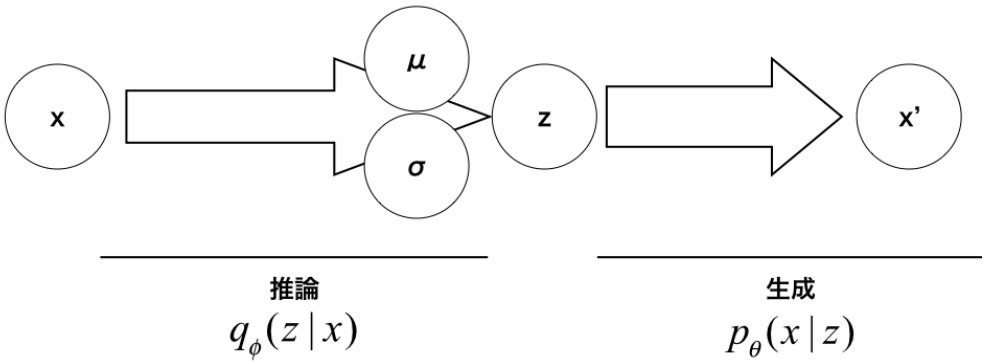

図 2.4　VAE の流れ

次に、最適化の方法を示す。

最尤推定

1. パラメータ θ の値が未知である時、とりあえず θ_0 だと仮定する。
2. その仮定のもとで、実際に観測した事象が起きる確率 $L(\theta_0)$ を考える。
3. $L(\theta_0)$ が大きいような θ_0 をもっともらしい推定値とする。

目的関数

VAE の目的は z の対数周辺尤度 $logp_\theta(x)$ を最大化することである。これは、データとして x を入手したということは、x の正規確率は高いはずだという仮定に基づくものである。イェンゼンの不等式を用いて $logp_\theta(x)$ を以下のように変形することができる。

$$\log p_\theta(x) = D_{KL}(q_\theta(z|x)||p_\theta(z|x)) + L(\theta,\phi,x) \geqq L(\theta,\phi,x)$$

VAE では $\log p_\theta(x)$ を直接最大化するのが困難なため、その下限値つまり $L(\theta, \phi, x)$ を最大化する。

変分下界の展開

以下の式変形により、最適にすべき項が導出できる。

$$L(\theta, \phi, x) = -D_{KL}(q_\phi(z|x)||p_\phi(z)) + E_{q_\phi(z|x)}[\log p_\theta(x)|z]$$

- $-D_{KL}(q_\phi(z|x)||p_\phi(z))$ 正則化項：KL DIvergence（Regularization Parameter）
- $E_{q_\phi(z|x)}[\log p_\theta(x)|z]$　　復元誤差：(Reconstruction Error)

この二つの和を最大化することでパラメータの最適化を行う。

第 3 章

単語の分散表現

3.1 word2vec

※従来の手法として Bag of Words について記述できる。

機械学習において、自然言語を処理するためには文書や単語を数値として扱う必要がある（ベクトル化)。代表的な手法としては、単純に文書での全出現単語を並べて各次元でその単語が存在するかどうかを判断する方法やニューラルネットワークを使う方法があるが本論文では後者の手法を扱う。

word2vec は、ニューラルネットワークを用いた、単語に対する概念を低次元の密なベクトルとして獲得する手法である。word2vec は「同じ文脈中に現れる単語同士は意味が似ている」という仮定に基づいており、テキスト中の各単語を周辺の単語から予測する擬似的な単語予測のタスクを設定し、それを大量のテキストからニューラルネットワークで学習することで、単語に対する分布表現を獲得する。

word2vec を実現するモデルとして、skip-gram モデル（SG）と continuous BOW モデル（CBOW）の 2 つが提案されている。

CBOW

与えられた文脈の中で出現している文脈語を用いて、ある対象語が出現しているかどうかを予測可能な意味表現を学習するモデル例えば、「私は毎日会社へ出社する。」という文があったとする。

形態素は [私, は, 毎日, 会社, へ, 出社, する] となる。対象語を「会社」とすると、それ以外の語を文脈語として用い、「会社」の出現を予測する。

CBOW モデルでは、文章中の文脈語からその文章中に「会社」という単語が出現するがどうかを予測できるよう単語の意味表現ベクトルを更新することが目的である。

SG

対象語を使って、文脈に出現している文脈語を予測するモデル。対象語の「会社」を用いて、他の単語 [私, は, 毎日, 会社, へ, 出社, する] の出現を予測する。

第 4 章

多義語とその曖昧性

4.1 多義語

我々が日常の意思疎通のために用いる言語を自然言語という。自然言語で使われる単語の中には複数の意味を持つものが存在する。そのような単語を多義語といい、単語が持つ意味を語義という。

単語「首」を例にとると以下の意味がある。

- 頭と体をつなぐ細い部分
- 解雇、罷免

このように多義語とは、単語は同じでも全く意味が異なるものである。

4.2 語義の曖昧性

動詞「やる」について考えると、以下のような異なった語義が存在する。

- 彼は課題をやった。(ある動作を行う)
- プレゼントとしてその時計をやった。(譲渡する)
- 目を向こうへやった。(視線を投げる)
- 彼の会社に人をやった。(遣いを出す)
- 彼はピアノをやった。(演奏する)
- 机の上の本を向こうへやった。(物をどかす)

我々は、普段会話をしたり文字を読んだりすることにより文脈から語義を分類する能力を感覚的に身につけている。しかし、機械には感覚というものがないため、これらの文章があったときにどの語義を取るのか分類することができない。これを語義の曖昧性と呼び、機械翻訳、意見抽出など自然言語処理の様々なタスクにおいて大きな障害となる。機械に正しい語義を分類させるためには語義曖昧性解消（Word Sense Disambiguation）を組み込む必要がある。

4.3 語義曖昧性解消

語義曖昧性解消とは、複数の語義を持つ単語について、文章中に出現する単語の意味を一意に決定することである。これを行うためには、分類器を定める必要があり、それには学習が必要である。学習には教

師あり学習、半教師あり学習、教師なし学習といった手法が利用される。ここでは半教師あり学習について説明する。

4.4　半教師あり学習

半教師あり学習は、以下 2 点の理由から少量のラベル有りデータと大量のラベル無しデータを学習とデータとして使用する学習手法である。

- ラベル有りデータは構築にコストがかかる
- ラベルなしデータはコストをかけずに大量に手に入れることができる

⇒ 少量のラベル有りデータと大量のラベル無しデータを学習データとして使用する

図 4.1　画像を例としたデータの比較

図のように、ラベル有データは少なく、ラベルのないデータは大量に手に入れることができる。半教師あり学習では、少ないデータで学習した分類学習器に対し、ラベルのないデータで推論を行い、高い確率で推論した結果は正しいと結果と定めて、そのデータに推論したラベルを付与して学習データにするというものである。つまり、ラベル有りデータに加え、大量のラベル無しデータから学習することで、ラベル有りデータのみで学習した場合より、より予測制度の高いクラス分類を実現するのが目的である。

第 5 章

実験

5.1 先行実験

GitHub にある VAE のプログラムを用い、MNIST の画像を入力データとし、中間層での潜在変数を２次元に落とし込むことが可能であることは確認されている。自分でも、プログラムを動かし可視化することができた。

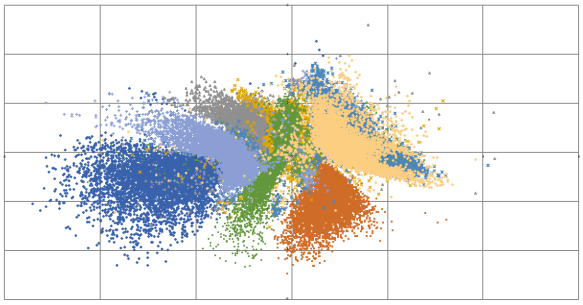

図 5.1　潜在変数の可視化

5.2 実験設定

本論文では、MNIST を用いて分類器の学習を行う。学習させた分類器を用いて、semEval「子供」のデータをうまく分類することが目的である。

5.3 結果

語義の分布図は作成することができた。VAE による用例ベクトルの次元縮約自体は可能であることが確認できた。

第 6 章

結論

本論文では、VAE を用いた用例ベクトルの次元縮約を提案した。MNIST で学習させた分類器を用い、文書を分類できれば精度が良くなるのでは無いかと考えたからである。VAE を用いた MNIST の次元縮約は、うまく分類できることが確認できた。VAE による用例ベクトルの次元縮約自体は可能であることが示されたが、分類が効果的になるように次元縮約できたのかどうかは不明である。今後、半教師あり学習を実装し、この点を確認する必要がある。

謝辞

本研究を進めるにあたり、多くのご指導、ご協力を頂いた指導教員の新納浩幸教授に感謝致します。また、日常の議論を通じて多くの知識や示唆をいただいた新納研究室の皆様に感謝します。

参考文献

[1] Diederik P. Kingma,”Semi-Supervised Learning with Deep Generative Models”,2014

[2] Diederik P Kingma,”Auto-Encoding Variational Bayes”,2013

[3] 山木翔馬 新納浩幸 古宮嘉那子 佐々木稔, 分散表現から得た用例間類似度を素性に加えた語義曖昧性解消,2016

付録

本論文で用いたコード

train_vae.py

```
"""Chainer example: train a VAE on MNIST
"""
from __future__ import print_function
import argparse
import os

import chainer
from chainer import training
from chainer.training import extensions
import numpy as np

import net


def main():
    parser = argparse.ArgumentParser(description='Chainer example: VAE')
    parser.add_argument('--initmodel', '-m', default='',
                        help='Initialize the model from given file')
    parser.add_argument('--resume', '-r', default='',
                        help='Resume the optimization from snapshot')
    parser.add_argument('--gpu', '-g', default=-1, type=int,
                        help='GPU ID (negative value indicates CPU)')
    parser.add_argument('--out', '-o', default='result',
                        help='Directory to output the result')
    parser.add_argument('--epoch', '-e', default=100, type=int,
                        help='number of epochs to learn')
    parser.add_argument('--dimz', '-z', default=20, type=int,
                        help='dimention of encoded vector')
    parser.add_argument('--batchsize', '-b', type=int, default=100,
                        help='learning minibatch size')
    parser.add_argument('--test', action='store_true',
                        help='Use tiny datasets for quick tests')
    args = parser.parse_args()

    print('GPU: {}'.format(args.gpu))
    print('# dim z: {}'.format(args.dimz))
    print('# Minibatch-size: {}'.format(args.batchsize))
    print('# epoch: {}'.format(args.epoch))
    print('')

    # Prepare VAE model, defined in net.py
    model = net.VAE(784, args.dimz, 500)
```

```
# Setup an optimizer
optimizer = chainer.optimizers.Adam()
optimizer.setup(model)

# Initialize
if args.initmodel:
    chainer.serializers.load_npz(args.initmodel, model)

# Load the MNIST dataset
train, test = chainer.datasets.get_mnist(withlabel=False)
if args.test:
    train, _ = chainer.datasets.split_dataset(train, 100)
    test, _ = chainer.datasets.split_dataset(test, 100)

train_iter = chainer.iterators.SerialIterator(train, args.batchsize)
test_iter = chainer.iterators.SerialIterator(test, args.batchsize,
                                             repeat=False, shuffle=False)

# Set up an updater. StandardUpdater can explicitly specify a loss function
# used in the training with 'loss_func' option
updater = training.updaters.StandardUpdater(
    train_iter, optimizer,
    device=args.gpu, loss_func=model.get_loss_func())

trainer = training.Trainer(updater, (args.epoch, 'epoch'), out=args.out)
trainer.extend(extensions.Evaluator(test_iter, model, device=args.gpu,
                                    eval_func=model.get_loss_func(k=10)))
trainer.extend(extensions.dump_graph('main/loss'))
trainer.extend(extensions.snapshot(), trigger=(args.epoch, 'epoch'))
trainer.extend(extensions.LogReport())
trainer.extend(extensions.PrintReport(
    ['epoch', 'main/loss', 'validation/main/loss',
     'main/rec_loss', 'validation/main/rec_loss', 'elapsed_time']))
trainer.extend(extensions.ProgressBar())

if args.resume:
    chainer.serializers.load_npz(args.resume, trainer)

# Run the training
trainer.run()

# Visualize the results
def save_images(x, filename):
    import matplotlib.pyplot as plt
    fig, ax = plt.subplots(3, 3, figsize=(9, 9), dpi=100)
    for ai, xi in zip(ax.flatten(), x):
        ai.imshow(xi.reshape(28, 28))
    fig.savefig(filename)

model.to_cpu()
train_ind = [1, 3, 5, 10, 2, 0, 13, 15, 17]
x = chainer.Variable(np.asarray(train[train_ind]))
with chainer.using_config('train', False), chainer.no_backprop_mode():
    x1 = model(x)
save_images(x.data, os.path.join(args.out, 'train'))
save_images(x1.data, os.path.join(args.out, 'train_reconstructed'))

test_ind = [3, 2, 1, 18, 4, 8, 11, 17, 61]
x = chainer.Variable(np.asarray(test[test_ind]))
```

```
    with chainer.using_config('train', False), chainer.no_backprop_mode():
        x1 = model(x)
    save_images(x.data, os.path.join(args.out, 'test'))
    save_images(x1.data, os.path.join(args.out, 'test_reconstructed'))

    # draw images from randomly sampled z
    z = chainer.Variable(
        np.random.normal(0, 1, (9, args.dimz)).astype(np.float32))
    x = model.decode(z)
    save_images(x.data, os.path.join(args.out, 'sampled'))


if __name__ == '__main__':
    main()
```

data.py

```
import gzip
import os

import numpy as np
import six
from six.moves.urllib import request

parent = 'http://yann.lecun.com/exdb/mnist'
train_images = 'train-images-idx3-ubyte.gz'
train_labels = 'train-labels-idx1-ubyte.gz'
test_images = 't10k-images-idx3-ubyte.gz'
test_labels = 't10k-labels-idx1-ubyte.gz'
dim = 784


def load_mnist(images, labels, num):
    data = np.zeros(num * dim, dtype=np.uint8).reshape((num, dim))
    target = np.zeros(num, dtype=np.uint8).reshape((num, ))

    with gzip.open(images, 'rb') as f_images,\
            gzip.open(labels, 'rb') as f_labels:
        f_images.read(16)
        f_labels.read(8)
        for i in six.moves.range(num):
            target[i] = ord(f_labels.read(1))
            for j in six.moves.range(dim):
                data[i, j] = ord(f_images.read(1))

    return data, target


def get_pkl_file_name(test):
    if test:
        return 'mnist.test.pkl'
    else:
        return 'mnist.pkl'


def download_mnist_data(test=False):
    if test:
        num_train = 100
        num_test = 100
```

```
    else:
        num_train = 60000
        num_test = 10000

    print('Downloading {:s}...'.format(train_images))
    request.urlretrieve('{:s}/{:s}'.format(parent, train_images), train_images)
    print('Done')
    print('Downloading {:s}...'.format(train_labels))
    request.urlretrieve('{:s}/{:s}'.format(parent, train_labels), train_labels)
    print('Done')
    print('Downloading {:s}...'.format(test_images))
    request.urlretrieve('{:s}/{:s}'.format(parent, test_images), test_images)
    print('Done')
    print('Downloading {:s}...'.format(test_labels))
    request.urlretrieve('{:s}/{:s}'.format(parent, test_labels), test_labels)
    print('Done')

    print('Converting training data...')
    data_train, target_train = load_mnist(train_images, train_labels,
                                          num_train)
    print('Done')
    print('Converting test data...')
    data_test, target_test = load_mnist(test_images, test_labels, num_test)
    mnist = {'data': np.append(data_train, data_test, axis=0),
             'target': np.append(target_train, target_test, axis=0)}
    print('Done')
    print('Save output...')
    with open(get_pkl_file_name(test), 'wb') as output:
        six.moves.cPickle.dump(mnist, output, -1)
    print('Done')
    print('Convert completed')


def load_mnist_data(test=False):
    pkl_file_name = get_pkl_file_name(test)
    if not os.path.exists(pkl_file_name):
        download_mnist_data(test)
    with open(pkl_file_name, 'rb') as mnist_pickle:
        mnist = six.moves.cPickle.load(mnist_pickle)
    return mnist
```

net.py

```
import six

import chainer
import chainer.functions as F
from chainer.functions.loss.vae import gaussian_kl_divergence
import chainer.links as L


class VAE(chainer.Chain):
    """Variational AutoEncoder"""

    def __init__(self, n_in, n_latent, n_h):
        super(VAE, self).__init__()
        with self.init_scope():
            # encoder
            self.le1 = L.Linear(n_in, n_h)
```

```
            self.le2_mu = L.Linear(n_h, n_latent)
            self.le2_ln_var = L.Linear(n_h, n_latent)
            # decoder
            self.ld1 = L.Linear(n_latent, n_h)
            self.ld2 = L.Linear(n_h, n_in)

    def __call__(self, x, sigmoid=True):
        """AutoEncoder"""
        return self.decode(self.encode(x)[0], sigmoid)

    def encode(self, x):
        h1 = F.tanh(self.le1(x))
        mu = self.le2_mu(h1)
        ln_var = self.le2_ln_var(h1)  # log(sigma**2)
        return mu, ln_var

    def decode(self, z, sigmoid=True):
        h1 = F.tanh(self.ld1(z))
        h2 = self.ld2(h1)
        if sigmoid:
            return F.sigmoid(h2)
        else:
            return h2

    def get_loss_func(self, C=1.0, k=1):
        """Get loss function of VAE.
        The loss value is equal to ELBO (Evidence Lower Bound)
        multiplied by -1.
        Args:
            C (int): Usually this is 1.0. Can be changed to control the
                second term of ELBO bound, which works as regularization.
            k (int): Number of Monte Carlo samples used in encoded vector.
        """
        def lf(x):
            mu, ln_var = self.encode(x)
            batchsize = len(mu.data)
            # reconstruction loss
            rec_loss = 0
            for l in six.moves.range(k):
                z = F.gaussian(mu, ln_var)
                rec_loss += F.bernoulli_nll(x, self.decode(z, sigmoid=False)) \
                    / (k * batchsize)
            self.rec_loss = rec_loss
            self.loss = self.rec_loss + \
                C * gaussian_kl_divergence(mu, ln_var) / batchsize
            chainer.report(
                {'rec_loss': rec_loss, 'loss': self.loss}, observer=self)
            return self.loss
        return lf
```