COMPARISON OF UNSUPERVISED PRE-TRAINING METHODS

Jifu Zhao, Jinsheng Wang

NetID: jzhao59

1. INTRODUCTION

As one important part of artificial intelligence (AI), machine learning has been widely used in our daily life, such as natural language processing (NLP), zip code recognition, self-driving cars and so on. Among those applications, supervised learning is the most common and perhaps the most powerful form of machine learning [**?**]. For supervised learning, the choice of appropriate representations of original features is very important [**?**]. For a long time, most machine learning applications require us to specify the input $\boldsymbol{x}$ and target $\boldsymbol{y}$. The choice of feature representation $\boldsymbol{x}$ is usually determined by experts. And sometimes the input $\boldsymbol{x}$ may be just the raw feature without pre-processing.

Researchers have been working on discovering the hidden structures of the data for years. In this process, several methods have been developed and found to be useful in some domains, such as Principal Component Analysis (PCA), Independent Component Analysis (ICA), Nonnegative Matrix Factorization (NMF), Restricted Boltzmann Machine (RBM), Auto-encoders and its variants and so on. These methods works in a unsupervised way and the output from these methods are thought to be a transformation of the raw input features. For supervised learning, some supervised learning mechanism will be built based on the output from these pre-training methods rather than the raw features. As expected, a good representation can help the supervised learning algorithms work better.

In this work, we focus on the comparison of several unsupervised per-training methods. More specifically, we will focus on PCA, kernel PCA and auto-encoders.

2. METHODS

2.1. PCA

Principal Component Analysis (PCA) is one of the most widely used pre-training methods for many applications. The goal of PCA is to find a lower dimensional linear space such that the variance of the orthogonal projection of the original data is maximized [**?**]. PCA is widely used for dimensionality reduction, feature extraction and data visualization. (Since this part has been covered in class, we won't cover too much derivations about PCA. Instead, we will focus on its applications.)

2.2. Kernel PCA

As a linear transformation, PCA works well in some applications where the input data can be well represented by linear transformation. However, in a lot of applications, this assumption doesn't holds. We need some non-linear transformation for these scenarios. One direct idea is to map the input $\boldsymbol{x}$ into some high dimensional space $\phi(\boldsymbol{x})$ then perform PCA in this high dimensional space. However, sometimes the dimensionality of the new space is so high that the naïve implementation of PCA in this new space becomes very expensive.

For this situation, Schölkopf [**?**, **?**] proposed the method called kernel PCA where the computation can be performed in the original feature space rather then the new space using so-called kernel trick. In this way, through kernel PCA, a non-linear transformation makes it possible for us to find some complex representations.

2.3. Auto-encoders

Auto-encoders use the input as the target, trying to find a reconstruction $r(x) = g(h(x))$ through so-called encoder $h(\cdot)$ and decoder $g(\cdot)$ [**?**]. In most cases, an auto-encoder system is a multiple layer deep neural network which has a bottleneck structure. A 5 layer auto-encoder system is shown in Figure 1.

Auto-encoder systems can be trained through back-prorogation. However, it works well only if we have a good initialization [**?**]. To get a good initialization, Hinton and Salakhutdinov [**?**] then proposed an effective way which utilize a stack of RBMs for pre-training. After unrolling, the whole auto-encoder system can be fine-tuned.

3. DATA

In this work, we plan to use the MNIST dataset [**?**]. The MNIST dataset contains a huge amount of handwirtten digits. It has 60000 training examples (more precisely, 55000 training examples, 5000 validation examples) and 10000 test examples. It has 10 classes which corresponding to digits from 0 to 9. Each example is a 28×28 pixel grayscale image.

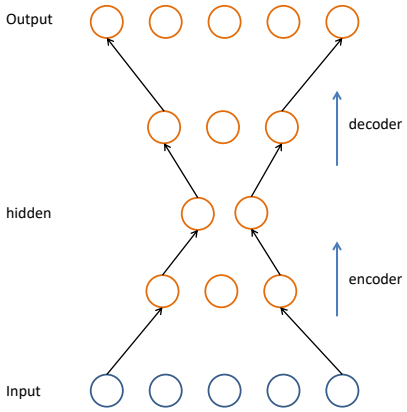

Fig. 1. Illustration of auto-encoder system.

The pixel values range from 0 to 1. A subset of 36 training examples are shown in Figure 2.

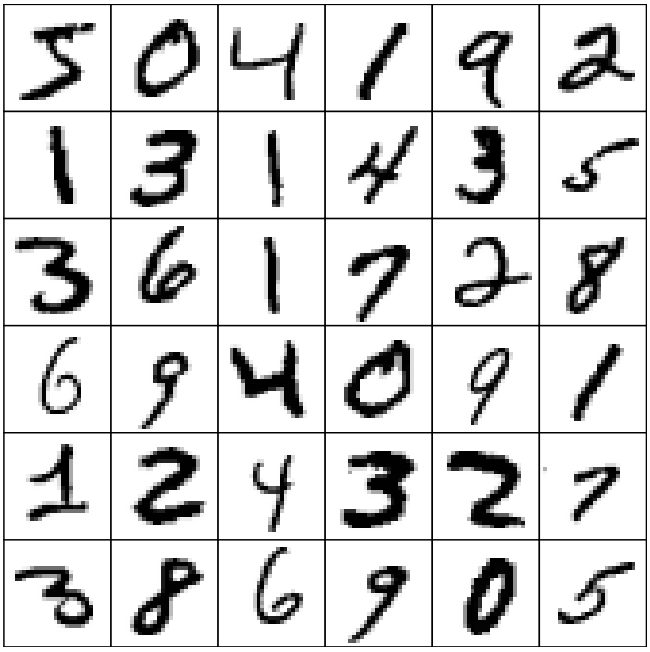

Fig. 2. Examples of MNIST dataset.

4. EXPERIMENTS

This work is an extension of the assignment 4, where we were required to compare the classification accuracy for raw image, PCA, RBM and stacked RBMs. In this work, we will use similar procedure, but focus on kernel PCA and auto-encoder systems.

Our work can be roughly divided into two parts. The first part will be focused on the comparison of PCA and kernel PCA. More specifically, we will conduct experiments on PCA and kernel PCA with polynomial kernel and rbf kernel. Through choosing different number of transformed features, we will build a supervised multiclass classifier to classify the dataset into 10 classes. Through calculating the error rate of different methods, we hope to compare the performance of different methods.

The second part will be focused on the auto-encoders. Except the study on number of hidden nodes and number of layers, we will also study the influence of pre-training using stacks of RBMs as described by Hinton and Salakhutdinov [**?**]. Then the same classification strategy will be implemented.

Since the goal of this work is to study different pre-training methods, the final classifier won't be very complicated, a multiclass Support Vector Machine (SVM) or a two-layer multi-class logistic regression neural network will be used.

5. RESOURCE FEASIBILITY

With a modern computer (i.e., Intel i7 processor, 16 GB RAM), most of the work described above can be implemented without any problem. However, for kernel PCA part, the kernel matrix K has the dimensionality of $n \times n$, where n is the number of given samples. In this work, for the MNIST dataset, there will be 60000 training examples. The dimensionality of K will be 60000×60000. To store this matrix in memory, it will need more than 25 GB (using numpy.float64 data type), which is not possible for our current computer. For this part, we will implement the algorithm on Amazon Web Service (AWS).

6. TIMELINE

We expect to finish the theoretical derivation part before the Thanksgiving ($11/24/2016$). Then we will focus on actual programming part after the Thanksgiving. For coding part, we will implement kernel PCA through Python3 + Numpy. For auto-encoder part, we will implement it through TensorFlow + TFLearn. We expect to conduct the actual experiment part from $11/28/2016$ and finish all parts before $12/07/2016$.